МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ
ІНСТИТУТ імені ІГОРЯ
СІКОРСЬКОГО»

Гордієнко Ю.Г., Таран В.І.

\

ХМАРНІ ОБЧИСЛЕННЯ

Конспект лекцій

Навчальний посібник
для здобувачів ступеня магістра
за освітньою програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення»
за освітньою програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
за освітньою програмою «Інформаційні управляючі системи та технології »
спеціальності 126 «Інформаційні системи та технології»

Електронне мережне навчальне видання

ЗАТВЕРДЖЕНО
на засіданні кафедри обчислювальної техніки
протокол № 10 від 25.05.2022

2022

****

Матеріали:

• Слайди лекцій
• Відеоверсії лекції
• Рекомендовані книги
• Теми для перспективних досліджень і розробок
• Екзаменаційні запитання
• Практичні запитання для самостійного навчання
• Підручник (конспект лекцій)

за посиланням:
https://cloud.comsys.kpi.ua/s/37mEqx2HdKz8oro

****

Cloud Computing

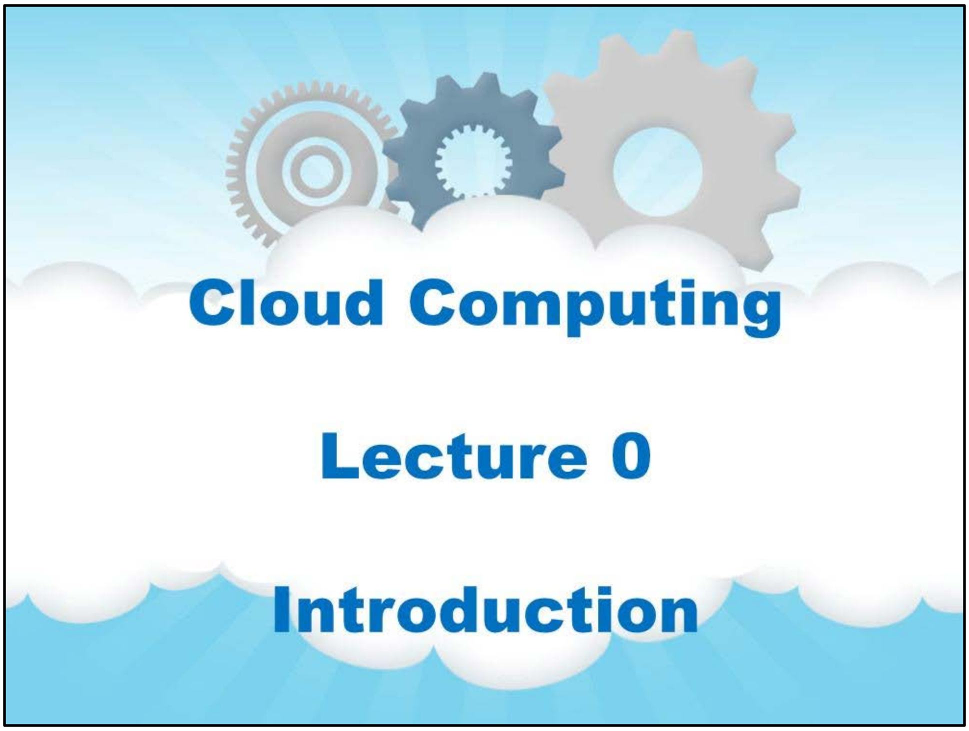
Lecture Manual

Volume 0

Module 0

Introduction to the Context of Cloud Computing

Content



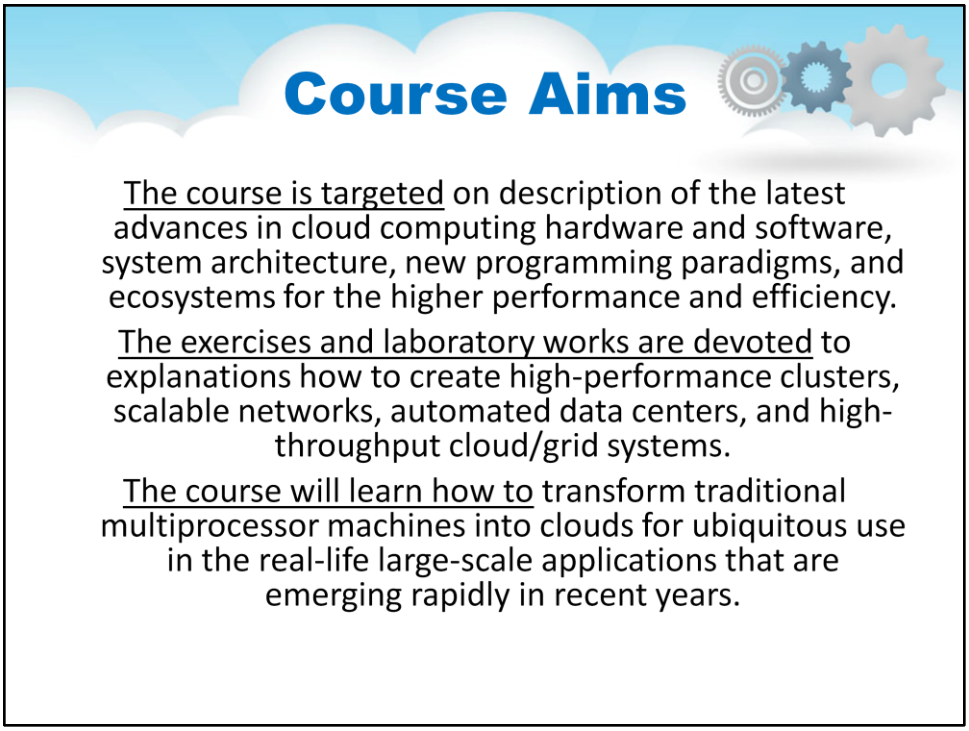
Cloud Computing

Lecture 0

Introduction

The title of the course is “Cloud Computing”

This is the introductory lecture number 0 about the context of Cloud Computing.



Course Aims

The course is targeted on description of the latest advances in cloud computing hardware and software, system architecture, new programming paradigms, and ecosystems for the higher performance and efficiency.

The exercises and laboratory works are devoted to explanations how to create high-performance clusters, scalable networks, automated data centers, and high-throughput cloud/grid systems.

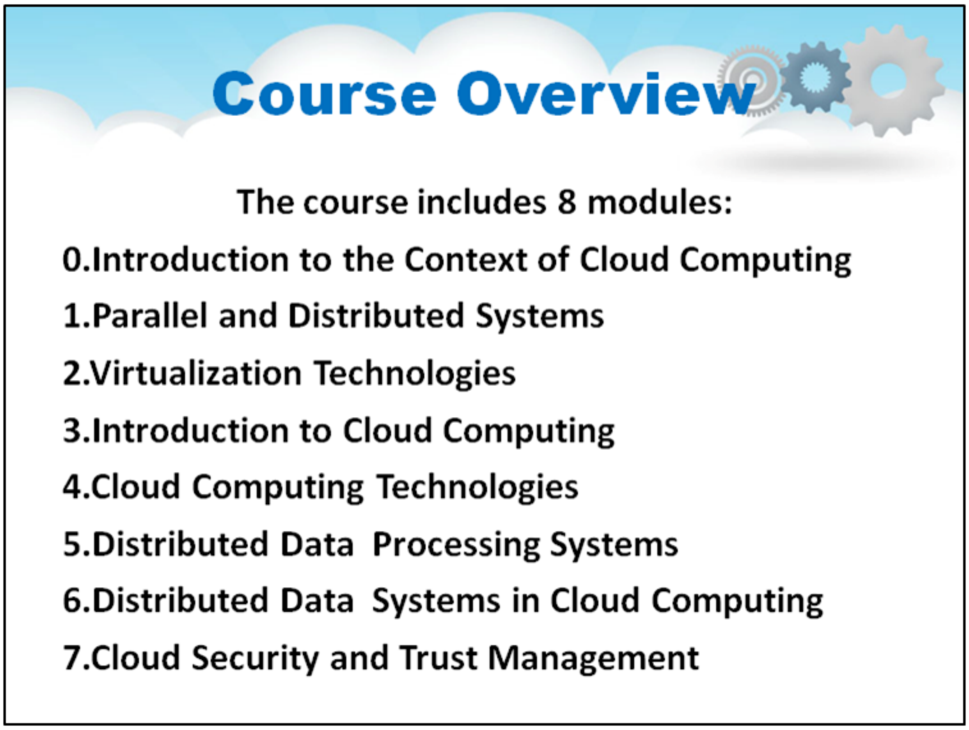
The course will learn how to transform traditional multiprocessor machines into clouds for ubiquitous use in the real-life large-scale applications that are emerging rapidly in recent years.

Let’s consider the main aims of this course:

The course is targeted on description of the latest advances in cloud computing hardware and software, system architecture, new programming paradigms, and ecosystems for the higher performance and efficiency.

The exercises and laboratory works are devoted to explanations how to create high-performance clusters, scalable networks, automated data centers, and high-throughput cloud/grid systems.

The course will learn how to transform traditional multiprocessor machines into clouds for ubiquitous use in the real-life large-scale applications that are emerging rapidly in recent years.



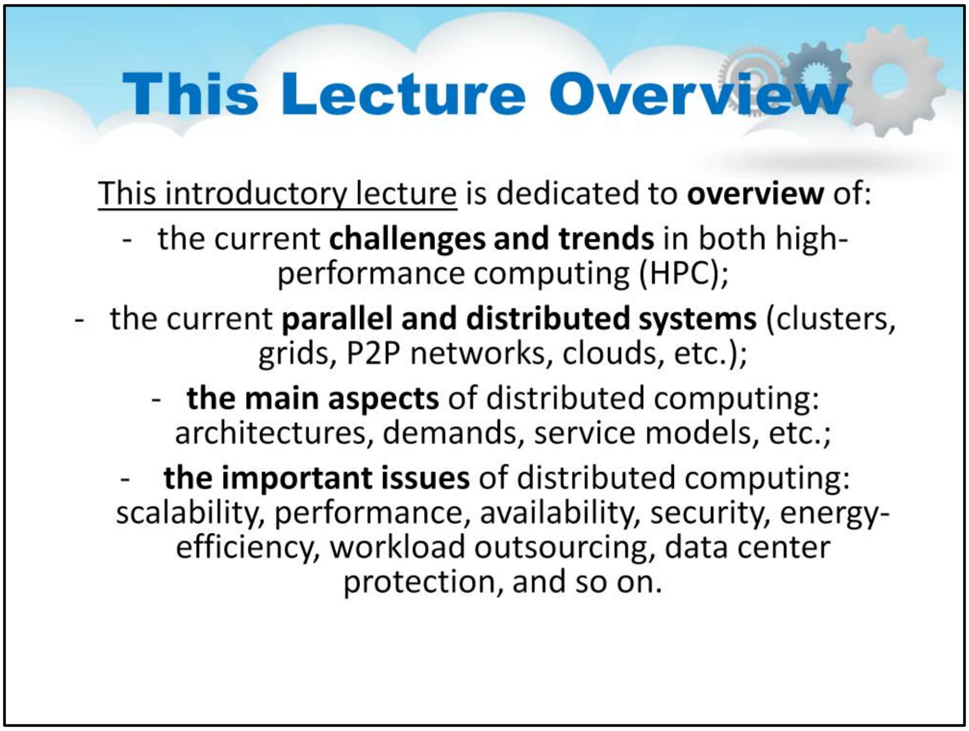
Course Overview

The course includes 8 modules:

0.Introduction to the Context of Cloud Computing
1.Parallel and Distributed Systems
2.Virtualization Technologies
3.Introduction to Cloud Computing
4.Cloud Computing Technologies
5.Distributed Data Processing Systems
6.Distributed Data Systems in Cloud Computing
7.Cloud Security and Trust Management

This course consists of the following modules.

0.Introduction to the Context of Cloud Computing
1.Parallel and Distributed Systems
2.Virtualization Technologies
3.Introduction to Cloud Computing
4.Cloud Computing Technologies
5.Distributed Data Processing Systems
6.Distributed Data Systems in Cloud Computing
7.Cloud Security and Trust Management



This Lecture Overview

This introductory lecture is dedicated to **overview** of:

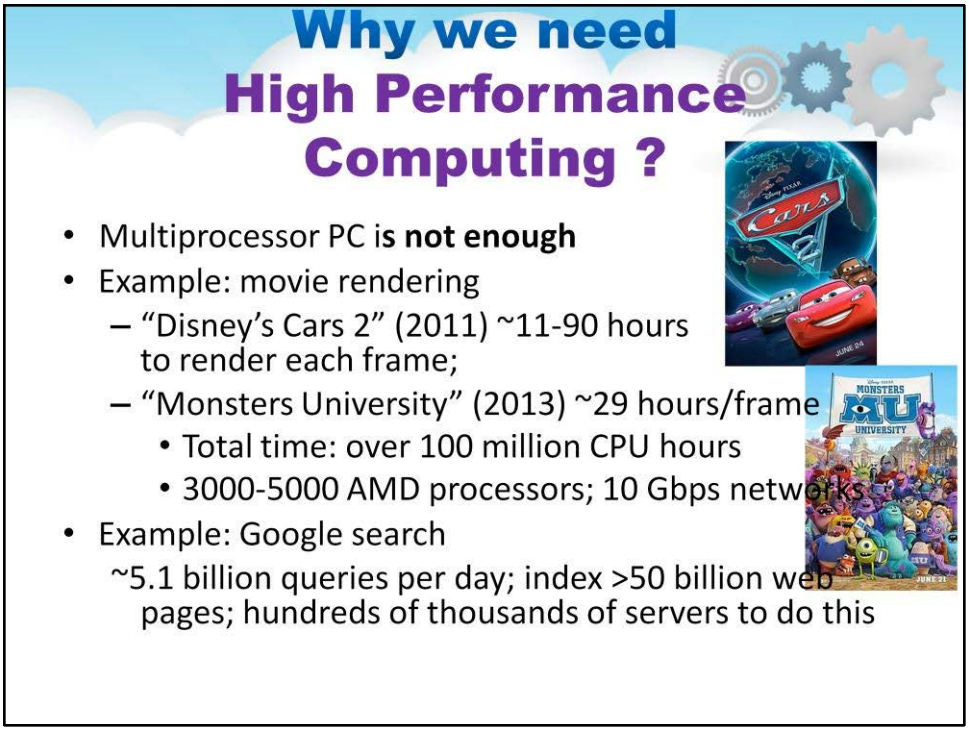
- the current **challenges and trends** in both high-performance computing (HPC);
- the current **parallel and distributed systems** (clusters, grids, P2P networks, clouds, etc.);
- **the main aspects** of distributed computing: architectures, demands, service models, etc.;
- **the important issues** of distributed computing: scalability, performance, availability, security, energy-efficiency, workload outsourcing, data center protection, and so on.

Lecture 0. Introduction to the Context of Cloud Computing

Current Challenges - Overview

Current Challenges

Overview



Why we need High Performance Computing ?

- Multiprocessor PC is **not enough**
- Example: movie rendering
  - “Disney’s Cars 2” (2011) ~11-90 hours to render each frame;
  - “Monsters University” (2013) ~29 hours/frame
    - Total time: over 100 million CPU hours
    - 3000-5000 AMD processors; 10 Gbps networks
- Example: Google search

  ~5.1 billion queries per day; index >50 billion web pages; hundreds of thousands of servers to do this

In response to the question “Why we need High Performance Computing” let’s consider some examples from our life.

To resolve the current computing problems the available multiprocessor personal computer is not enough anymore.

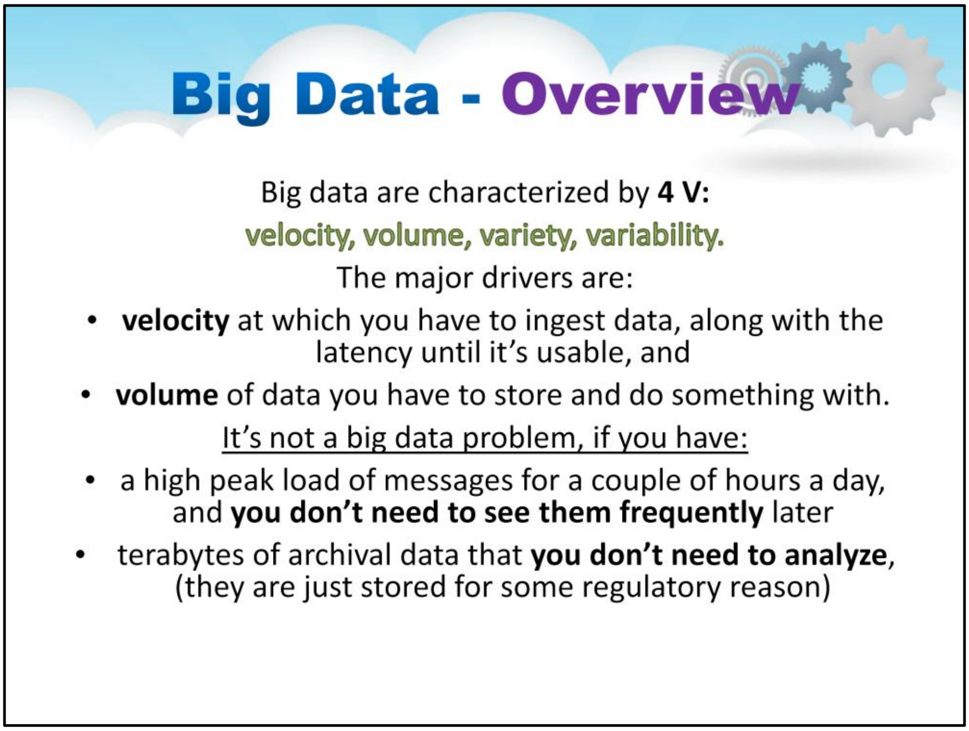
For example, if we would like to create high-quality cartoon movie. we need to perform rendering operation. It include calculation of all elements of characters, objects, their movements, colors, textures, illumination, etc.

For example, for “Disney’s Cars 2” cartoon in 2011 the computing time to render each frame was equal to ~11-90 hours.

For another cartoon movie, “Monsters University” in 2013 the computing time to render each frame was equal to ~29 hours. And the total computing time should be over than 100 million CPU hours! It is impossible to render such movies by the one multiprocessor personal computer, even the most powerful one.

The rendering task was resolved by the high-performance computing system with 3000-5000 AMD processors connected by high-speed 10 Gbps networks.

Another example is Google search engine, which should process ~5.1 billion queries per day and index >50 billion web pages. For this purpose, hundreds of thousands of servers should be used – namely, Cloud Computing infrastructure should do this work!



Big Data - Overview

Big data are characterized by **4 V:**

velocity, volume, variety, variability.

The major drivers are:

- **velocity** at which you have to ingest data, along with the latency until it's usable, and
- **volume** of data you have to store and do something with.

<u>It's not a big data problem, if you have:</u>

- a high peak load of messages for a couple of hours a day, and **you don't need to see them frequently** later
- terabytes of archival data that **you don't need to analyze**, (they are just stored for some regulatory reason)

Another example is the current problem of Big Data related with processing of the high volume of data generated by Internet of People, Internet of Things, and Internet of Everything.

This problem will be considered later in details.

But at this stage, please, find the following overview of the Big Data problem.

Big Data are characterized by **4 V:**

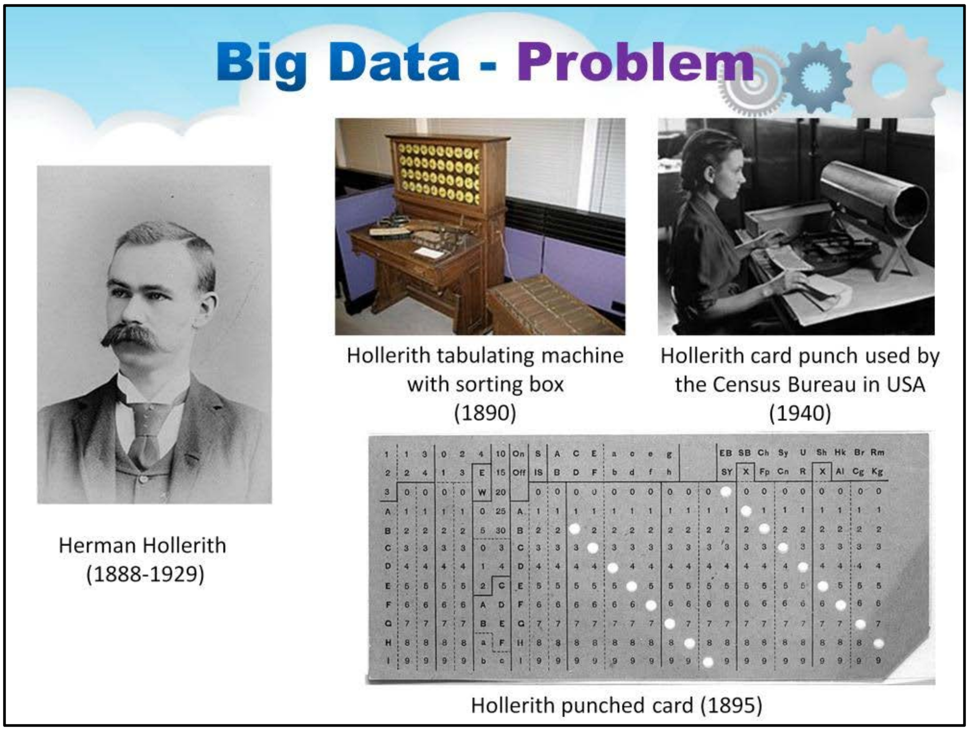
velocity, volume, variety, variability.

The major drivers are:

- **velocity** at which you have to ingest data, along with the latency until it's usable, and
- **volume** of data you have to store and do something with.

<u>But it's not a big data problem, if you have:</u>

- a high peak load of messages for a couple of hours a day, and **you don't need to see them frequently** later
- terabytes of archival data that **you don't need to analyze**, (they are just stored for some regulatory reason)



Big Data - Problem

Herman Hollerith
(1888-1929)

Hollerith tabulating machine
with sorting box
(1890)

Hollerith card punch used by
the Census Bureau in USA
(1940)

Hollerith punched card (1895)

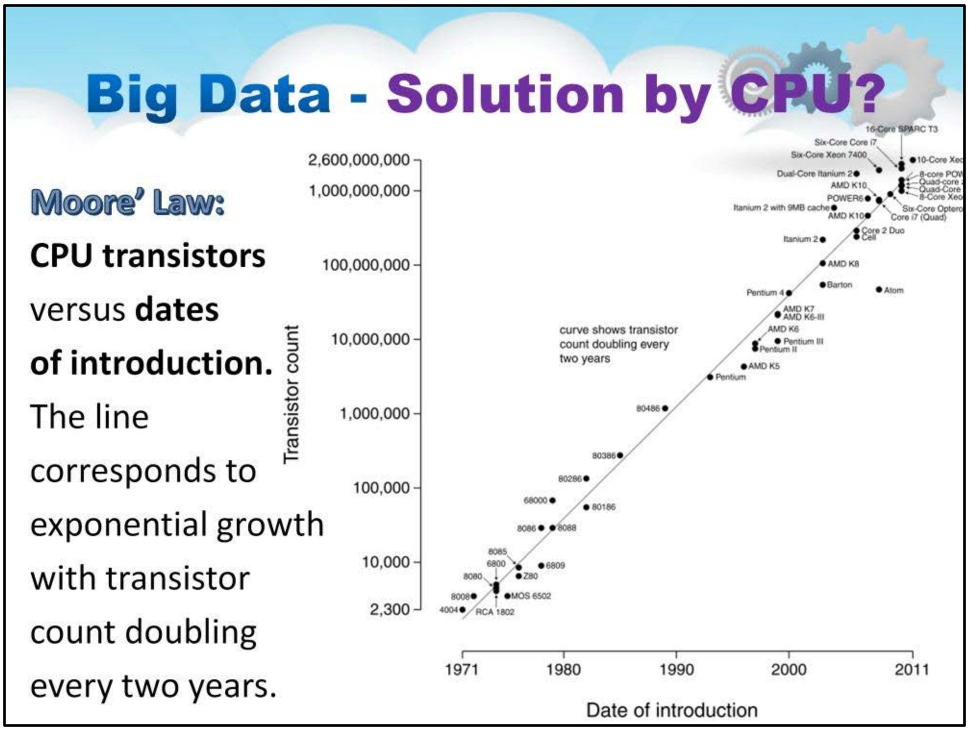
When the first big data problem appeared?

The first big data problem occurred in the 1880s.

In the late 1800s, the processing of the U.S. census was beginning to take close to 10 years. The census runs **every 10 years** and the population, and thus the amount of information was increasing — **problem**!

In 1886, Herman Hollerith started a business to rent machines that could read and tabulate census data on punch cards. The 1890 census took <2 years to complete and handled a larger population (62 million people) and more data points than the 1880 census.

Later Hollerith's business merged with three others to form what became IBM!



Big Data - Solution by CPU?

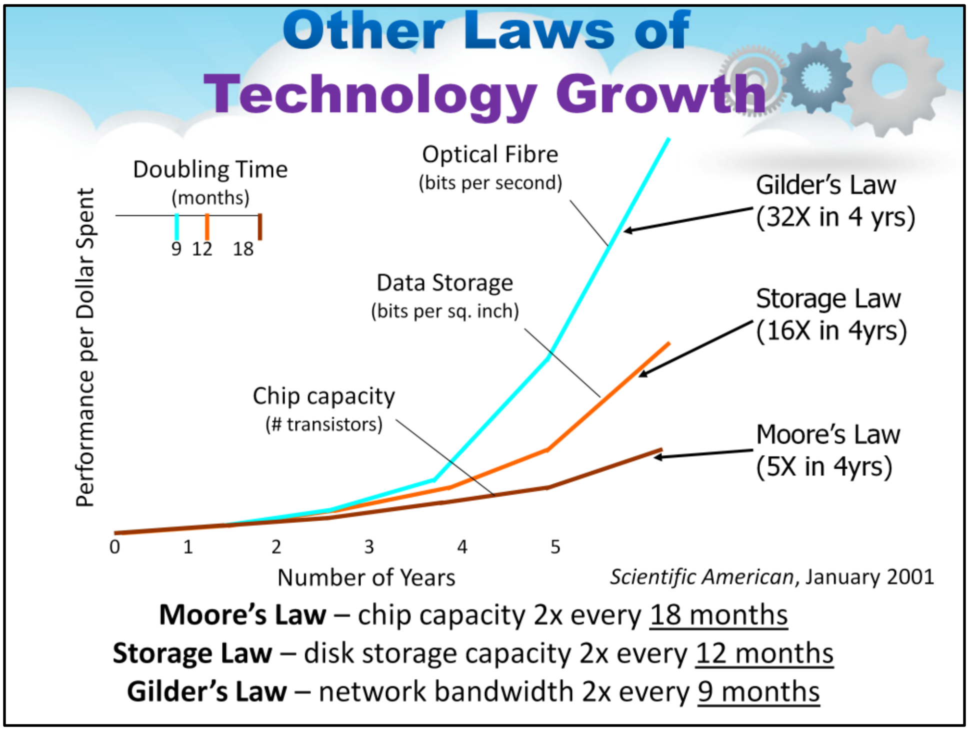
Moore' Law:

CPU transistors versus **dates of introduction.** The line corresponds to exponential growth with transistor count doubling every two years.

Here you can see Moore' Law:
Dependence of the number of transistors inside CPU versus **dates of introduction.**
The line corresponds to exponential growth with transistor count doubling every two years.

You can see that this dependence is saturated during the last years due to physical limitations on the size of elements.

Gordon Moore said in 2005:
"In terms of size [of transistors] you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far—but that's as far out as we've ever been able to see. We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions."
But now the current development of single Central Processing Unit (CPU) computing is not enough to response to the current challenges.



Other Laws of Technology Growth

Scientific American, January 2001

Moore's Law – chip capacity 2x every 18 months

Storage Law – disk storage capacity 2x every 12 months

Gilder's Law – network bandwidth 2x every 9 months

Now there are several other formulations of laws for caharacterization ~~of technology~~ growth:

Moore's Law – Individual computers double in processing power every 18 months

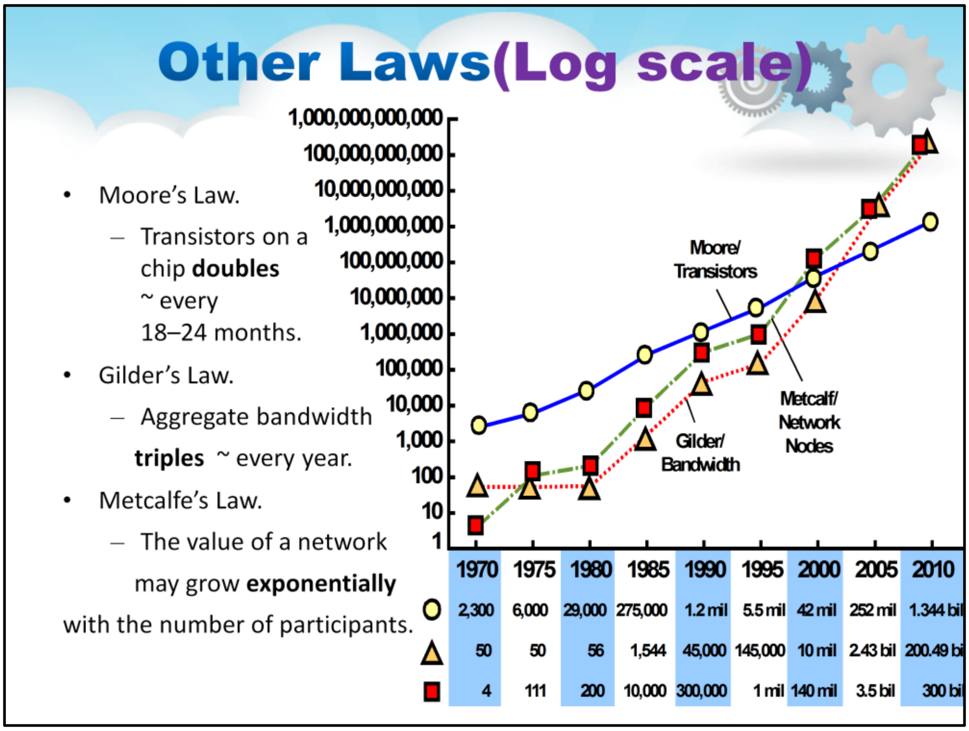
Storage Law – disk storage capacity doubles every 12 Months

Gilder's Law – Network bandwidth doubles every 9 months (but is harder to install)

This exponential growth profoundly changes the landscape of Information technology

More and more data created (because sensors are smaller and processors are cheaper) and stored (because disks are cheaper and more reliable than tape etc) and accessed remotely (because networking is cheaper).

Note that the time needed to fill available storage in increasing (i.e. the speed of writing data to disk is not increasing as quickly as the storage capacity)

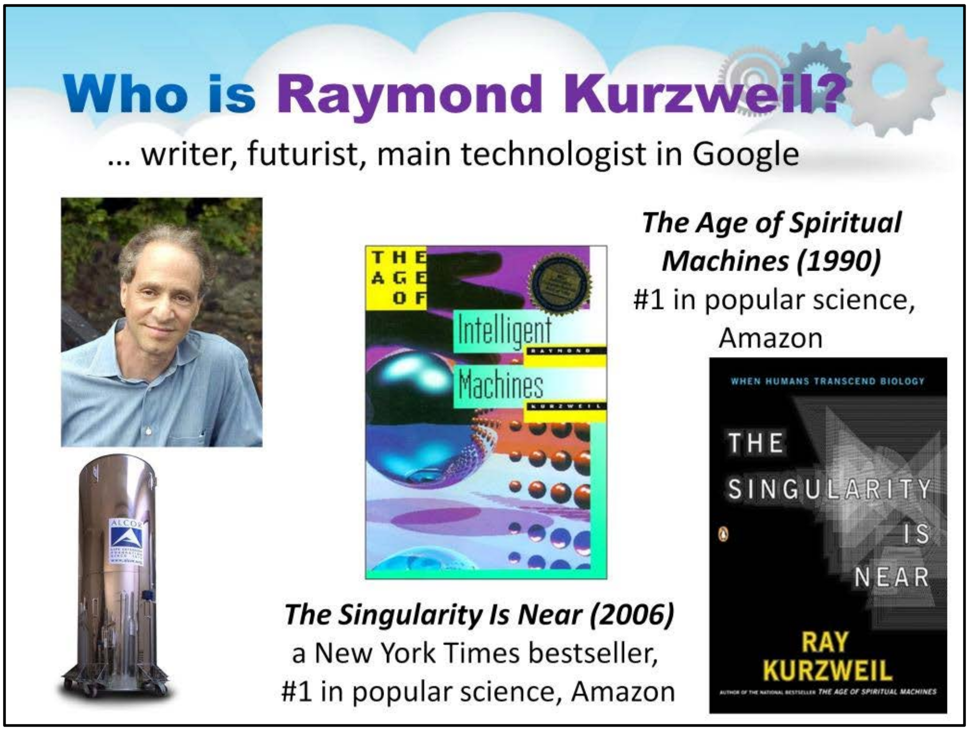


Other Laws(Log scale)

- Moore’s Law.
  - Transistors on a chip **doubles** ~ every 18–24 months.
- Gilder’s Law.
  - Aggregate bandwidth **triples** ~ every year.
- Metcalfe’s Law.
  - The value of a network may grow **exponentially** with the number of participants.

	1970	1975	1980	1985	1990	1995	2000	2005	2010
○	2,300	6,000	29,000	275,000	1.2 mil	5.5 mil	42 mil	252 mil	1.344 bil
△	50	50	56	1,544	45,000	145,000	10 mil	2.43 bil	200.49 bi
■	4	111	200	10,000	300,000	1 mil	140 mil	3.5 bil	300 bil

This is logarithmic plot of these laws.



Who is Raymond Kurzweil?

... writer, futurist, main technologist in Google

The Age of Spiritual Machines (1990)
#1 in popular science,
Amazon

The Singularity Is Near (2006)
a New York Times bestseller,
#1 in popular science, Amazon

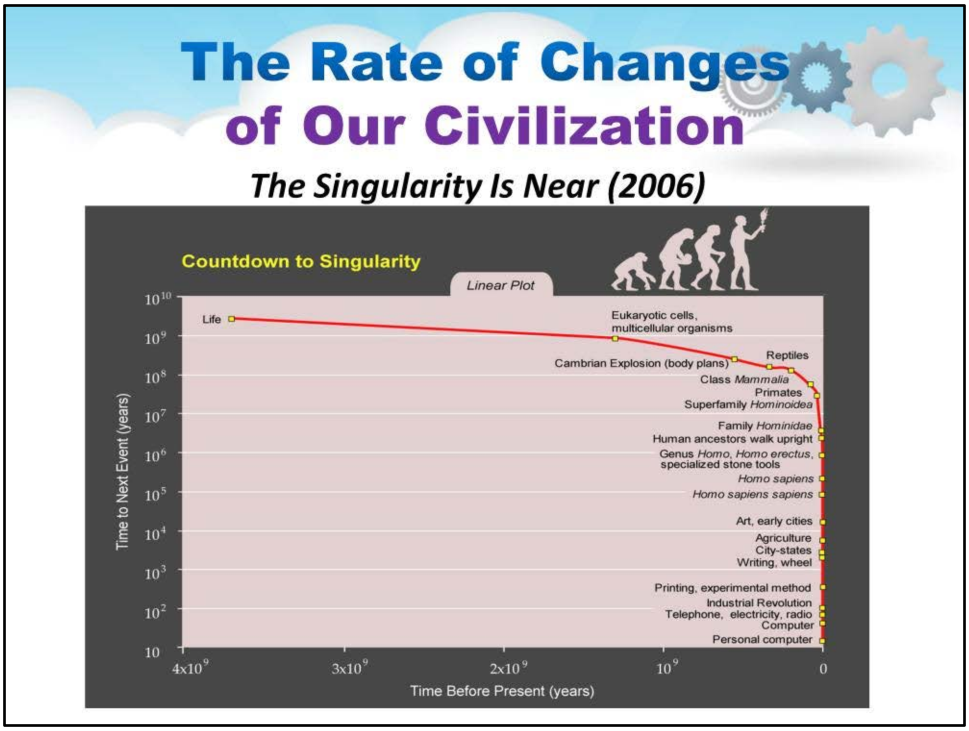
THE AGE OF
Intelligent
Machines
RAYMOND
KURZWEIL
ALCOR
WHEN HUMANS TRANSCEND BIOLOGY
THE SINGULARITY IS NEAR
RAY KURZWEIL
AUTHOR OF THE NATIONAL BESTSELLER THE AGE OF SPIRITUAL MACHINES

Raymond Kurzweil is an American author, computer scientist, inventor and futurist.

Aside from futurism, he is involved in fields such as optical character recognition (OCR), text-to-speech synthesis, speech recognition technology, and electronic keyboard instruments.

He has written books on health, artificial intelligence (AI), singularity.

Kurzweil's first book, The Age of Intelligent Machines, presented his ideas about the future in 1990. Kurzweil extrapolated trends in the improvement of computer chess software performance to predict that computers would beat the best human players "by the year 2000". In May 1997, chess World Champion Garry Kasparov was defeated by IBM's Deep Blue computer in a well-publicized chess match.

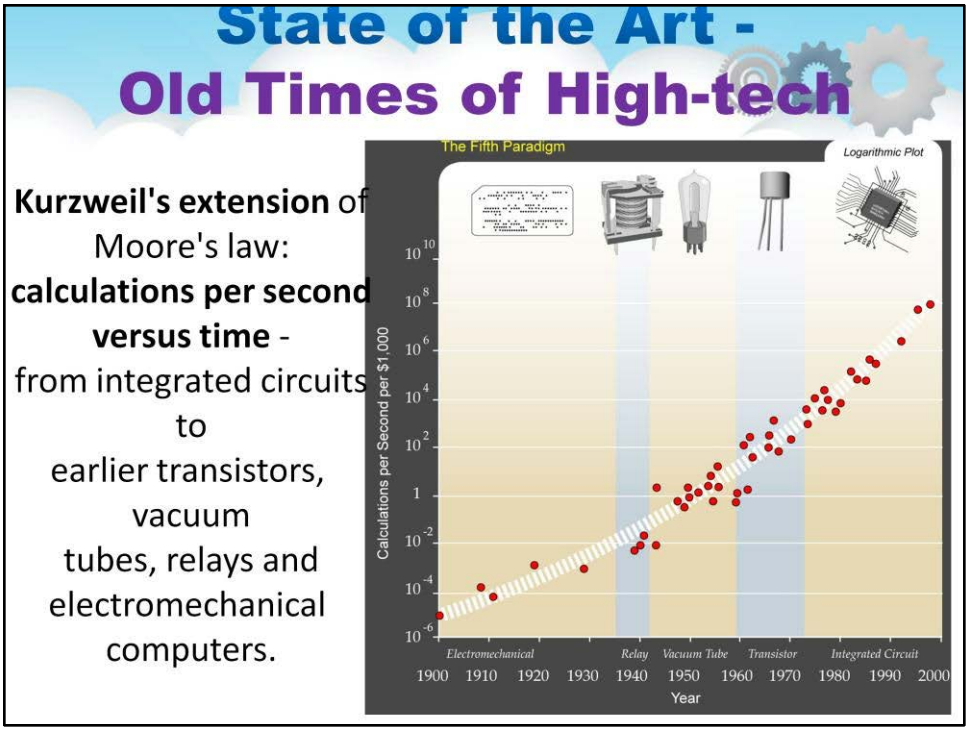


The Rate of Changes of Our Civilization

The Singularity Is Near (2006)

He analyzed the evolution of crucial events of our civilization and proposed the commonly accepted belief that the primary anatomical difference between humans and other primates that allowed for superior intellectual abilities was the evolution of a larger neocortex.

According to Kurzweil, technologists will be creating synthetic neocortexes based on the operating principles of the human neocortex with the primary purpose of extending our own neocortexes.



State of the Art -

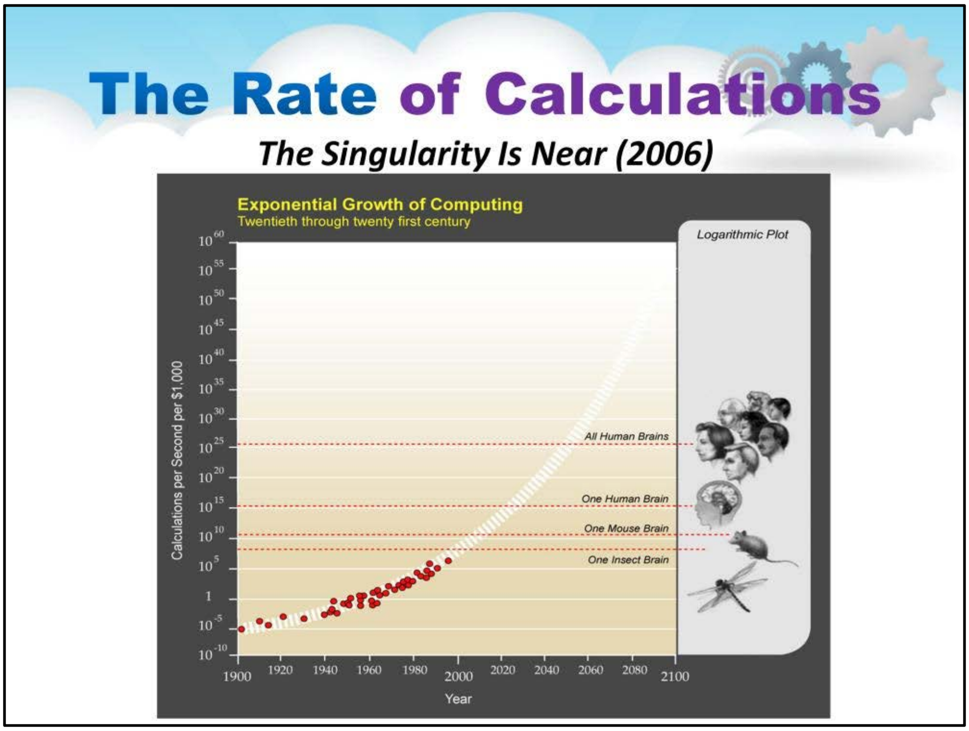
Old Times of High-tech

Kurzweil's extension of Moore's law: **calculations per second versus time** - from integrated circuits to earlier transistors, vacuum tubes, relays and electromechanical computers.

In his 1999 book The Age of Spiritual Machines, Kurzweil proposed "The Law of Accelerating Returns", according to which the rate of change in a wide variety of evolutionary systems (including the growth of technologies) tends to increase exponentially.

He proposed an extension of Moore's law to a wide variety of technologies, and used this to argue in favor of concept of a technological singularity.

Perhaps most significantly, Kurzweil foresaw the explosive growth in worldwide Internet use that began in the 1990s. At the time of the publication of The Age of Intelligent Machines, there were only 2.6 million Internet users in the world,[62] and the medium was unreliable, difficult to use, and deficient in content. He also stated that the Internet would explode not only in the number of users but in content as well, eventually granting users access "to international networks of libraries, data bases, and information services". Additionally, Kurzweil claims to have correctly foreseen that the preferred mode of Internet access would inevitably be through wireless systems, and he was also correct to estimate that the latter would become practical for widespread use in the early 21st century.

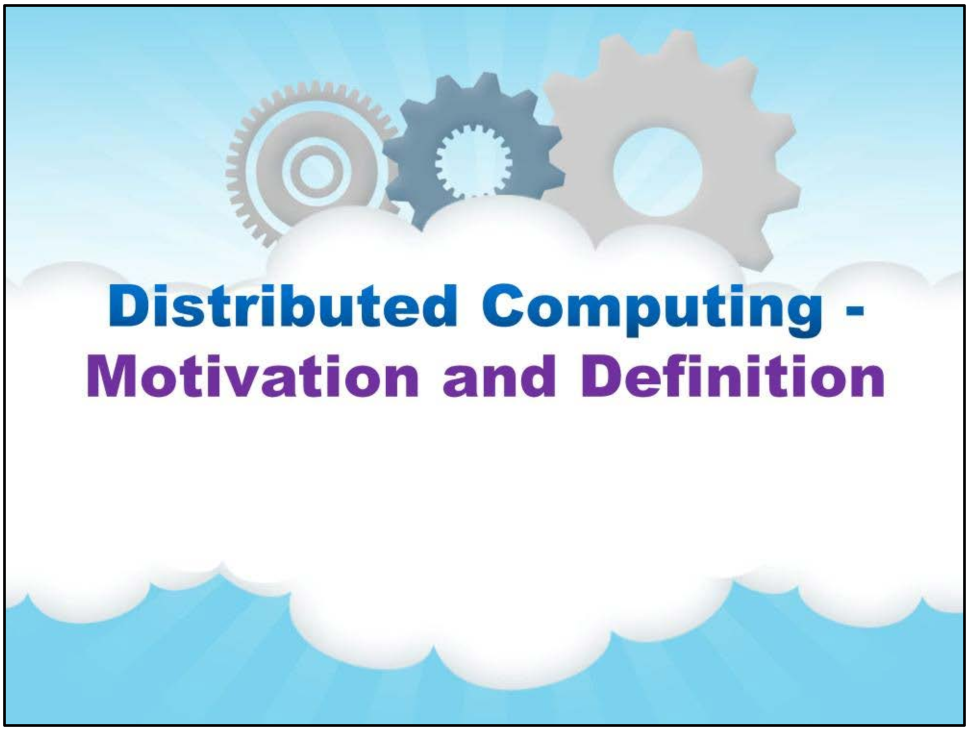


The Rate of Calculations

The Singularity Is Near (2006)

Kurzweil suggests that this exponential technological growth is counter-intuitive to the way our brains perceive the world—since our brains were biologically inherited from humans living in a world that was linear and local—and, as a consequence, he claims it has encouraged great skepticism in his future projections.

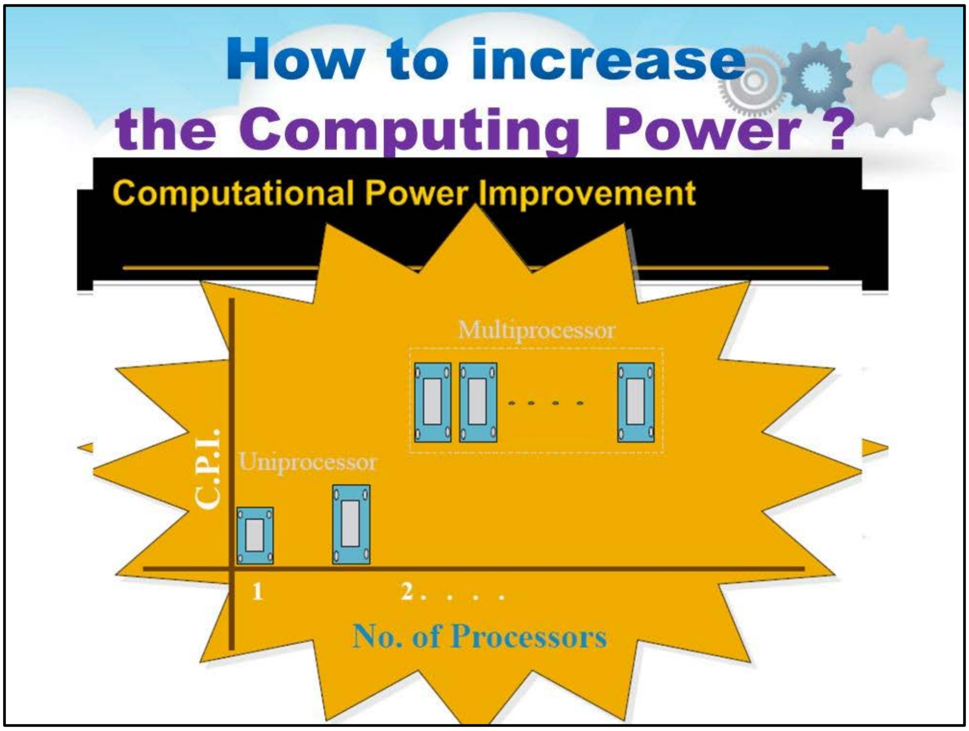
In the context of the before mentioned challenges and Big Data problems, the need of the new paradigms for high-perfomance computing is very high and important.



Distributed Computing - Motivation and Definition

Motivation and Definition

for Distributed Computing



How to increase the Computing Power ?

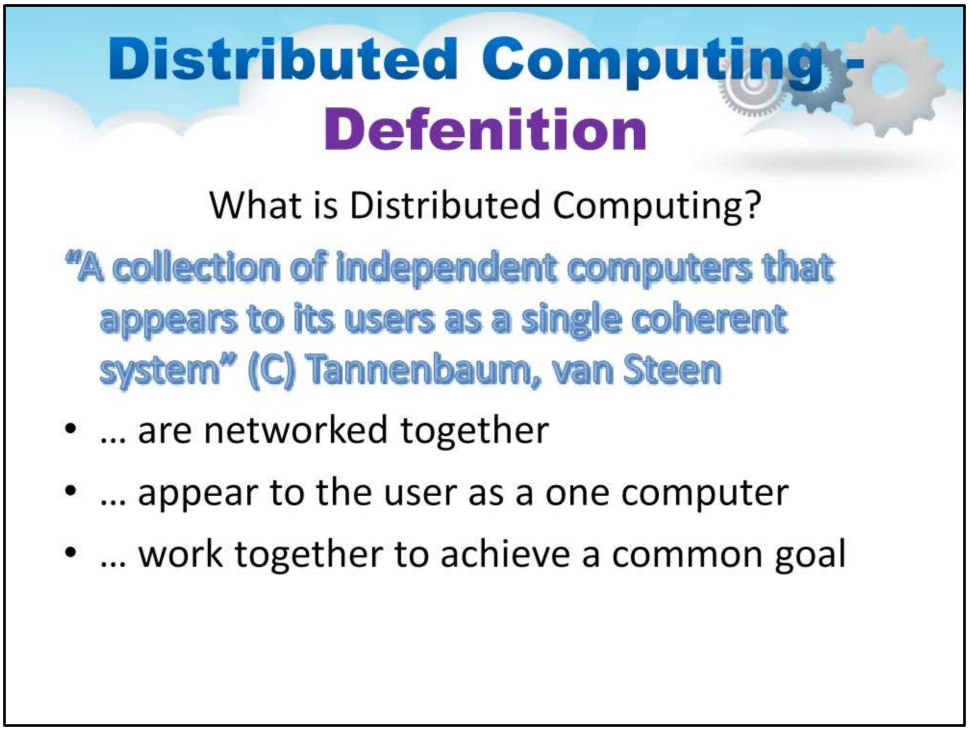
Computational Power Improvement

How to increaze the Computing Power?

Here you can see the ananlogy with the increase of Human Weight.

To some extent any young human can increase the **weight** with the increase of **height**. But after childhood this opportunity is finished, and adult people can increase the **weight** with the increase of **WIDTH** only.

The same situation with computing power.
To some extent we can increase the **computing power** with the increase of the **number of transistors**. But after Moore'Law limit this opportunity is finished, and we can increase the **weight** with the increase of **the number of processors** only.



Distributed Computing - Defenition

What is Distributed Computing?

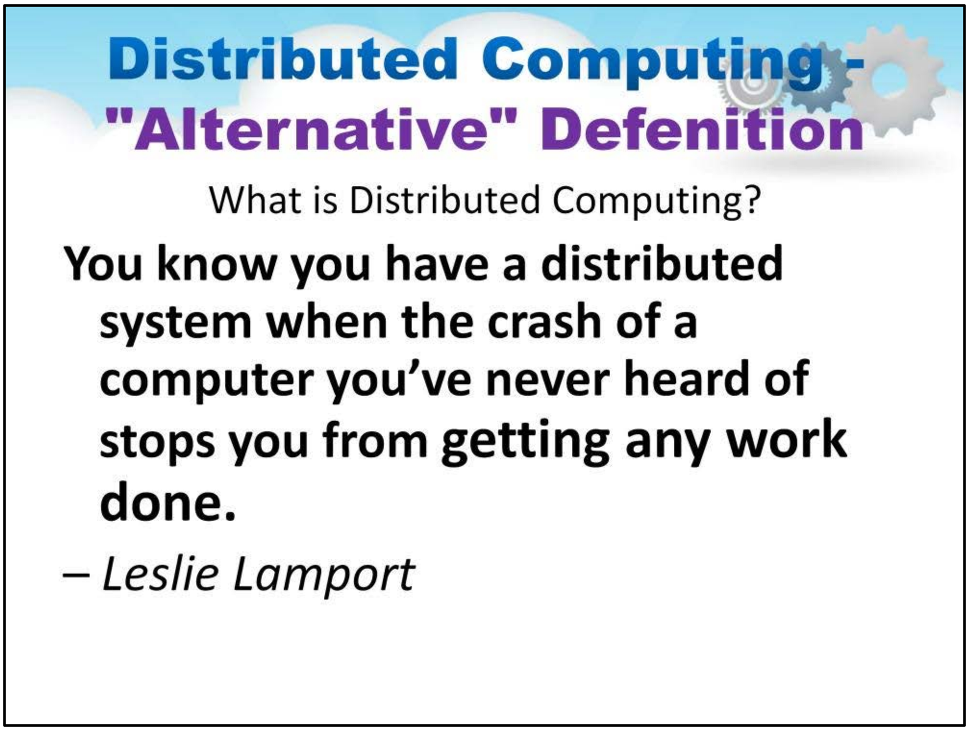
"A collection of independent computers that appears to its users as a single coherent system" (C) Tannenbaum, van Steen

- … are networked together
- … appear to the user as a one computer
- … work together to achieve a common goal

What is Distributed Computing?
"A collection of independent computers that appears to its users as a single coherent system" (C) Tannenbaum, van Steen
… are networked together
… appear to the user as a one computer
… work together to achieve a common goal

We should emphasize the following important aspects of this definition:
• from system architecture point of view: the machines are autonomous; this means they are computers which, in principle, could work independently;
• from user point of view: the distributed system is perceived as a single system solving a certain problem (even though, in reality, we have several computers placed in different locations).

By running a distributed system software the computers are enabled to:
- coordinate their activities
- share resources: hardware, software, data.



Distributed Computing - "Alternative" Defenition

What is Distributed Computing?

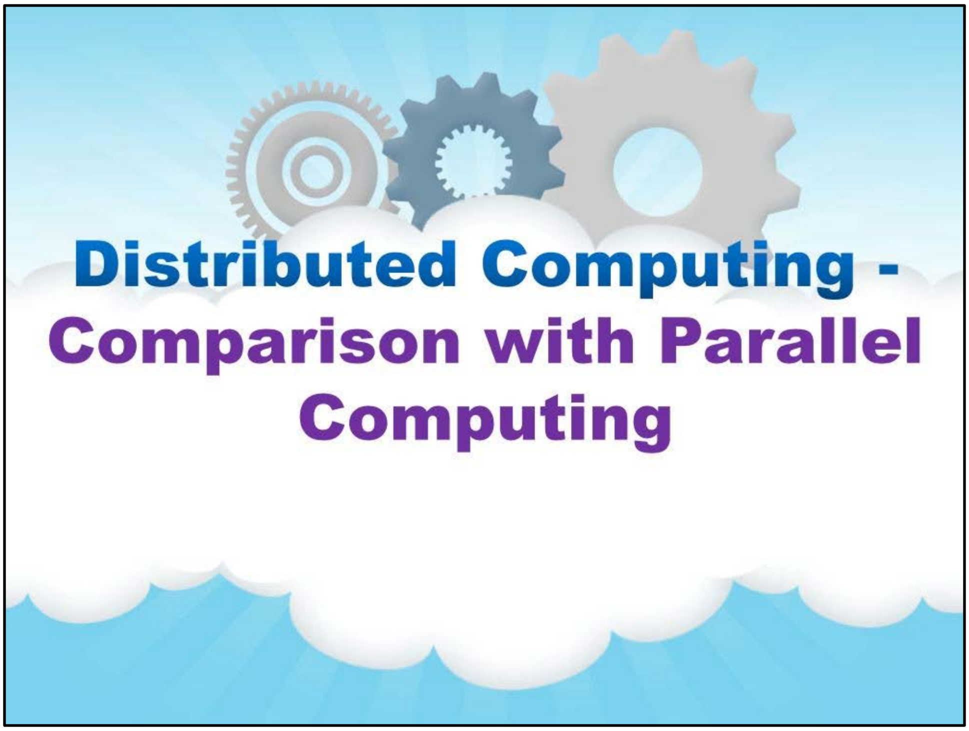
You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done.

– *Leslie Lamport*

It means that:

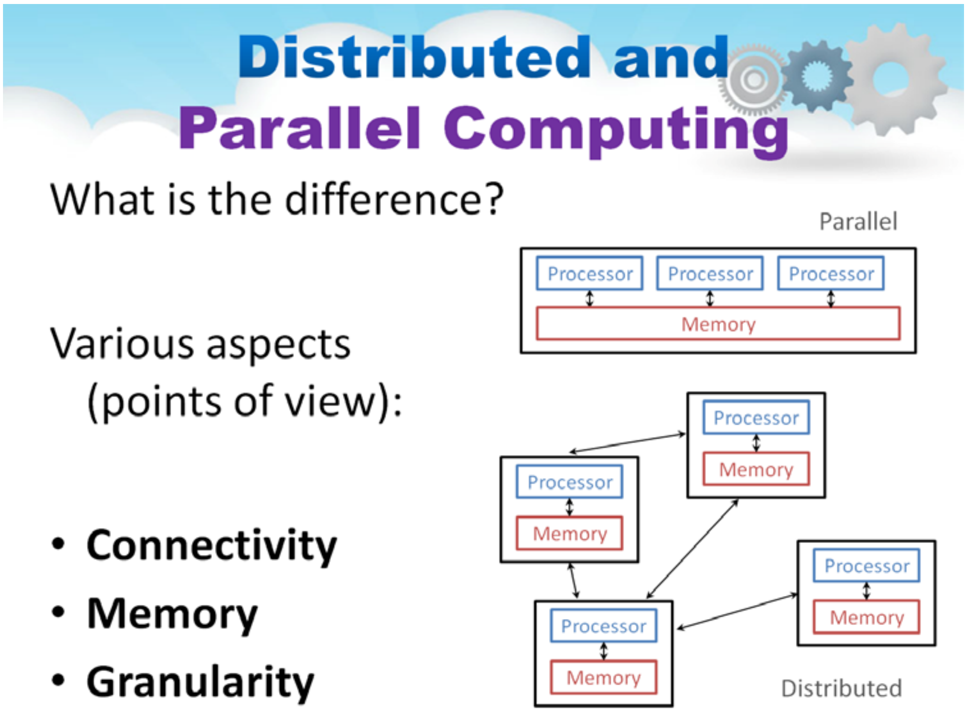
- differences between the various computers, the ways in which they communicate, and the internal organization of the distributed system are hidden from users;

- users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.



Distributed Computing - Comparison with Parallel Computing

Comparison with Parallel Computing



Distributed and Parallel Computing

What is the difference?

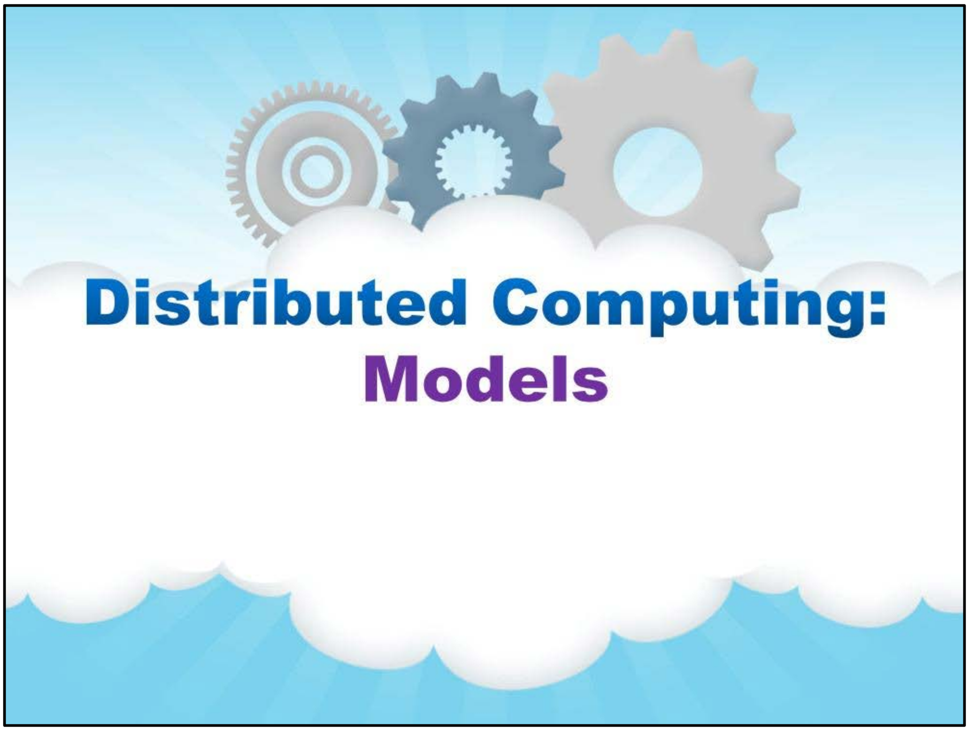
Various aspects (points of view):

- **Connectivity**
- **Memory**
- **Granularity**

Connectivity - The degree of coupling among a set of modules, whether hardware or software, is measured in terms of the interdependency and binding and/or homogeneity among the modules. When the degree of coupling is high (low), the modules are said to be tightly (loosely) coupled.

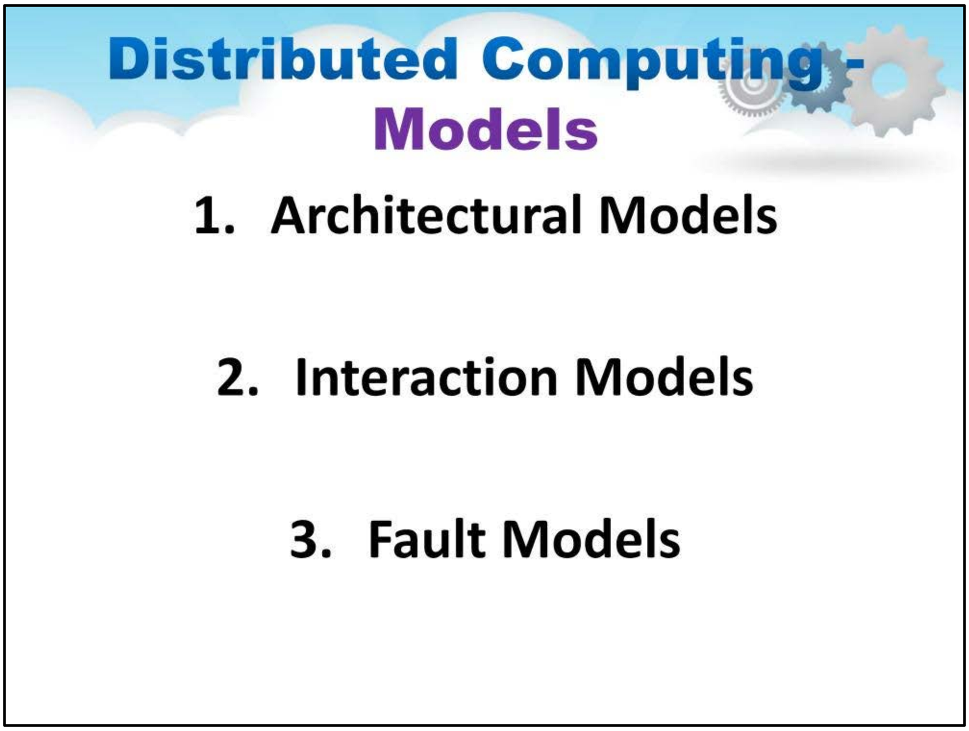
Memory -Shared memory systems are those in which there is a (common) shared address space throughout the system. Communication among processors takes place via shared data variables, and control variables for synchronization among the processors. Semaphores and monitors that were originally designed for shared memory uniprocessors and multiprocessors are examples of how synchronization can be achieved in **shared memory systems**. All multicomputer systems that do not have a shared address space provided by the underlying architecture and hardware – **distributed memory systems** – necessarily communicate by message passing. The abstraction called *shared memory* is sometimes provided to simulate a shared address space. For a distributed system, this abstraction is called **distributed shared memory systems**.

Granularity – The ratio of the amount of computation to the amount of communication within the parallel/distributed program is termed as *granularity. If the degree* of parallelism is coarse-grained (fine-grained), there are relatively many more (fewer) productive CPU instruction executions, compared to the number of times the processors communicate either via shared memory or message passing and wait to get synchronized with the other processors. Programs with fine-grained parallelism are best suited for tightly coupled systems.



Distributed Computing: Models

Models

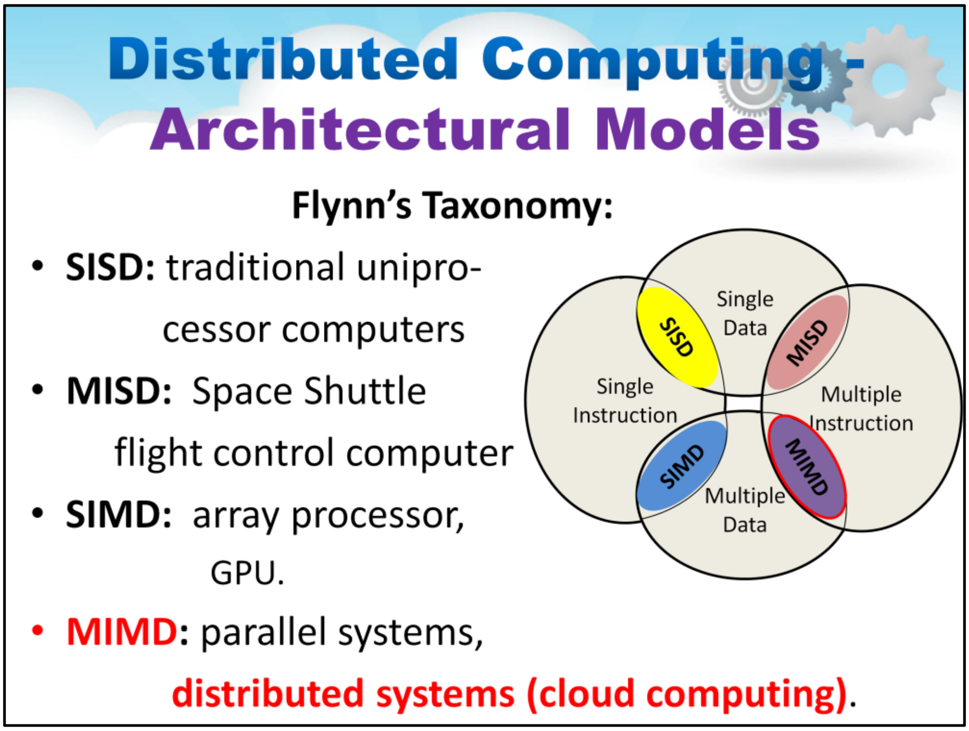


Distributed Computing - Models

1. Architectural Models
2. Interaction Models
3. Fault Models

Models of Distributed Computing are divided in the following categories:

1. **Architectural Models**
2. **Interaction Models**
3. **Fault Models**



Distributed Computing - Architectural Models

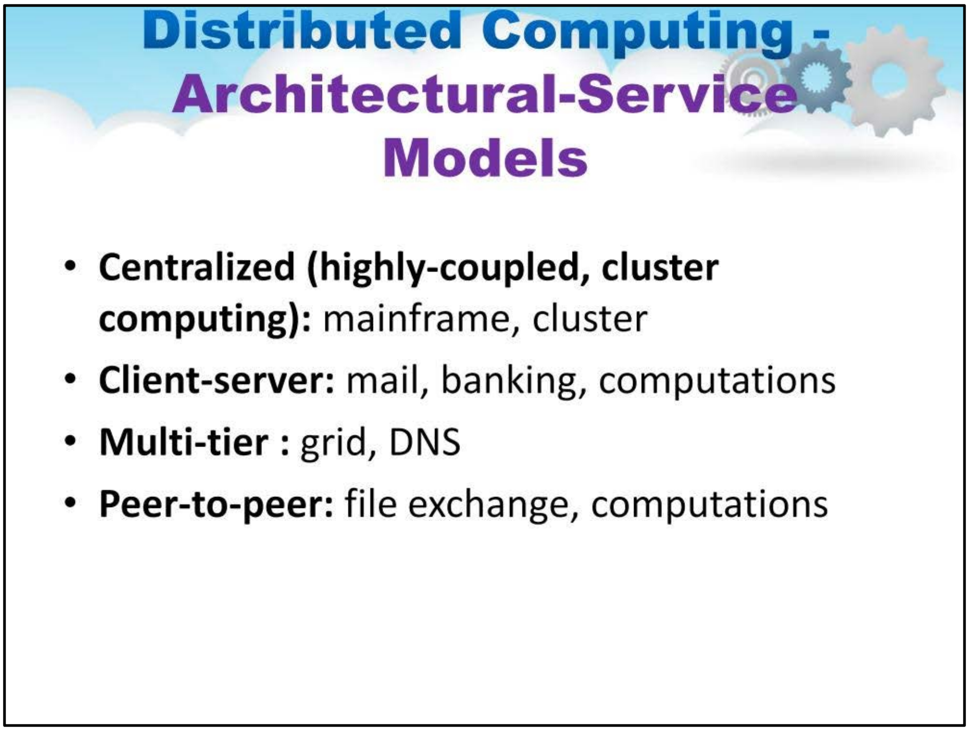
Flynn's Taxonomy:

- **SISD:** traditional unipro-cessor computers
- **MISD:** Space Shuttle flight control computer
- **SIMD:** array processor, GPU.
- **MIMD:** parallel systems, **distributed systems (cloud computing)**.

Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966.
This classification is based upon the number of concurrent **Instruction** streams and **Data** streams available in the architecture: **Single** or **Multiple**.

Flynn's Taxonomy includes:

- **SISD:** traditional uniprocessor computers
- **MISD:** Space Shuttle flight control computer
- **SIMD:** array processor, GPU.
- **MIMD:** parallel systems, **distributed systems (cloud computing)**.



Distributed Computing - Architectural-Service Models

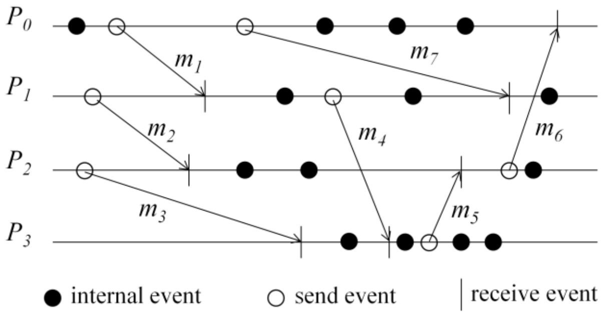
- **Centralized (highly-coupled, cluster computing):** mainframe, cluster
- **Client-server:** mail, banking, computations
- **Multi-tier :** grid, DNS
- **Peer-to-peer:** file exchange, computations

Architectural-Service Models of Distributed Computing are divided in the following categories:

- Centralized (highly-coupled, cluster computing): mainframe, cluster
- Client-server: mail, banking, scientific computations
- Multi-tier: grid systems, domain name servers (DNS)
- Peer-to-peer: file exchange, scientific computations

Distributed Systems - Interaction Models

How do we handle time? Are there time limits on process execution, message delivery, and clock drifts? (The arrows denote the messages.)



- An asynchronous execution is an execution in which

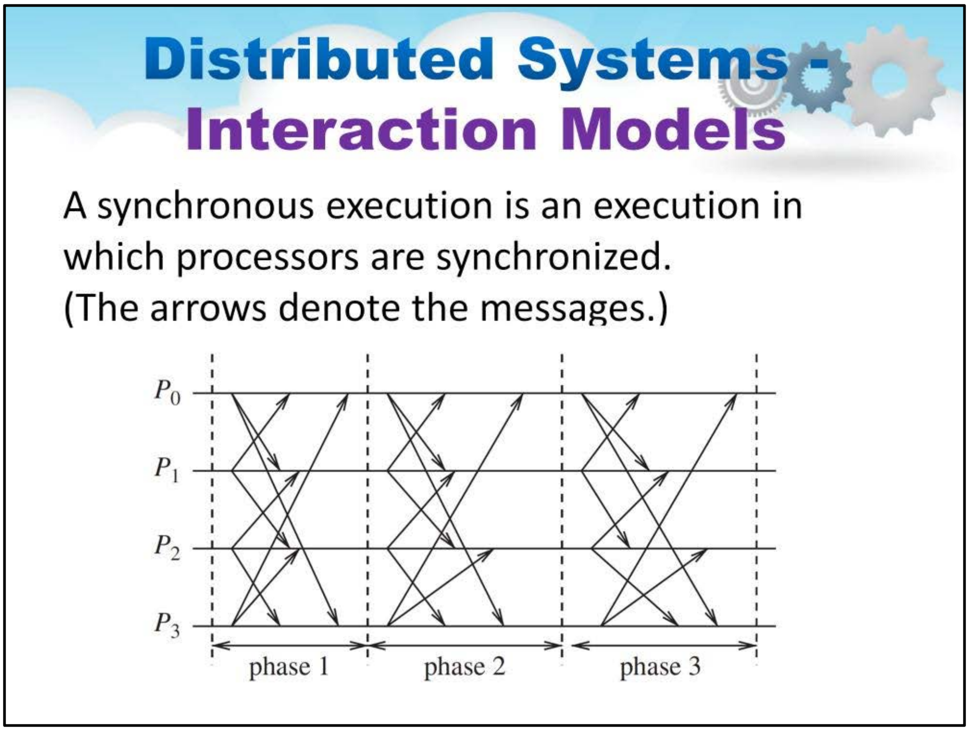
(i) there is no processor synchrony and there is no bound on the drift rate of processor clocks,

(ii) message delays (transmission + propagation times) are finite but unbounded, and

(iii) there is no upper bound on the time taken by a process to execute a step.

An example asynchronous execution with four processes P0 to P3 is shown in this Figure.

The arrows denote the messages; the tail and head of an arrow mark the send and receive event for that message, denoted by a circle and vertical line, respectively. Non-communication events, also termed as internal events, are shown by shaded circles.



Distributed Systems - Interaction Models

A synchronous execution is an execution in which processors are synchronized. (The arrows denote the messages.)

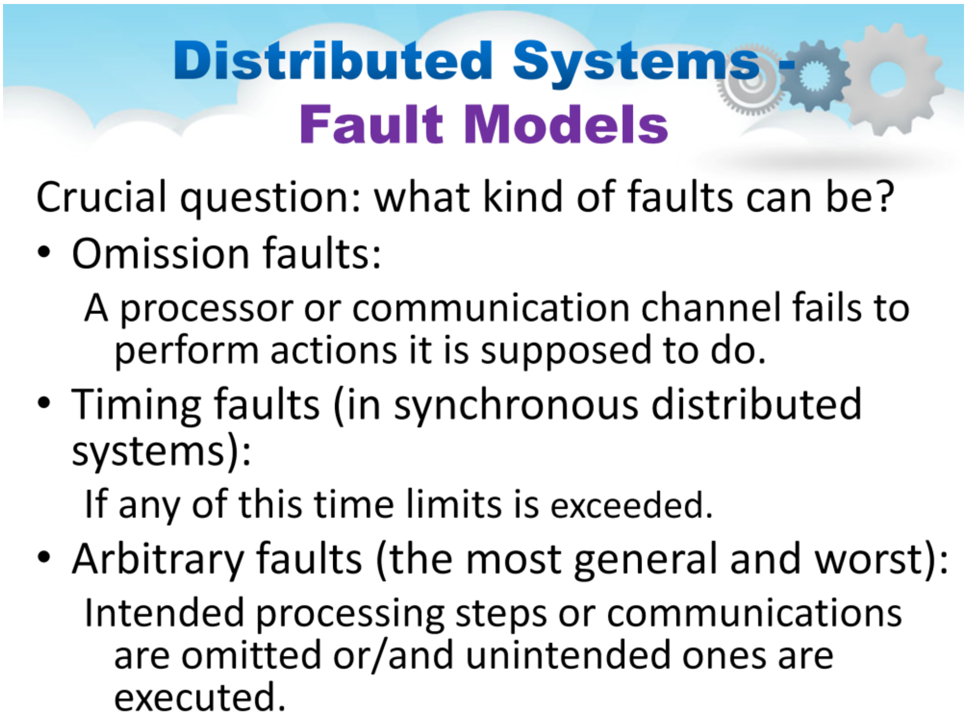
A *synchronous execution is an execution in which*

(i) *processors are synchronized* and the clock drift rate between any two processors is bounded,

(ii) message delivery (transmission + delivery) times are such that they occur in one logical step or round, and

(iii) there is a known upper bound on the time taken by a process to execute a step.

An example of a synchronous execution with four processes P0 to P3 is shown in this Figure.



Distributed Systems - Fault Models

Crucial question: what kind of faults can be?

- Omission faults:

  A processor or communication channel fails to perform actions it is supposed to do.

- Timing faults (in synchronous distributed systems):

  If any of this time limits is exceeded.

- Arbitrary faults (the most general and worst):

  Intended processing steps or communications are omitted or/and unintended ones are executed.

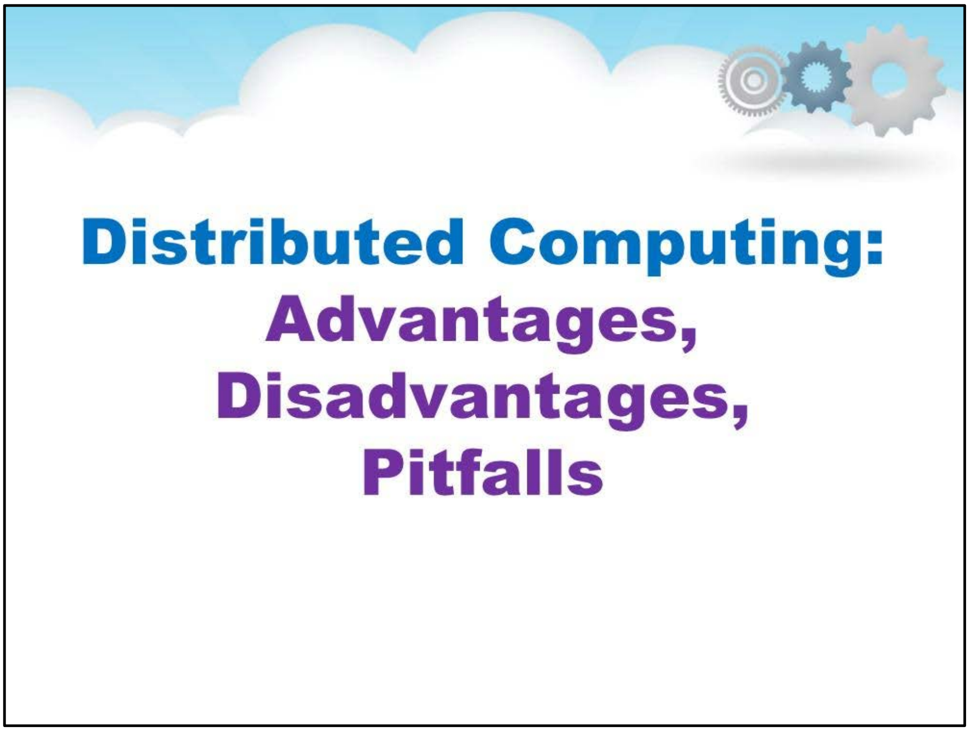
The faults model define failures that may occur in distributed systems. Hadzilacos and Toueg provide the following taxonomy of fault models.

Process **omission** failures: The chief omission failure of a process is to crash. When we say that a process has crashed we mean that it has halted and will not execute any further steps of its program ever.

Communication **omission** failures: The communication channel produces an omission failure if it does not transport a message from *p's outgoing message buffer to q's incoming message buffer. This is* known as 'dropping messages' and is generally caused by lack of buffer space at the receiver or at an intervening gateway, or by a network transmission error, detected by a checksum carried with the message data.

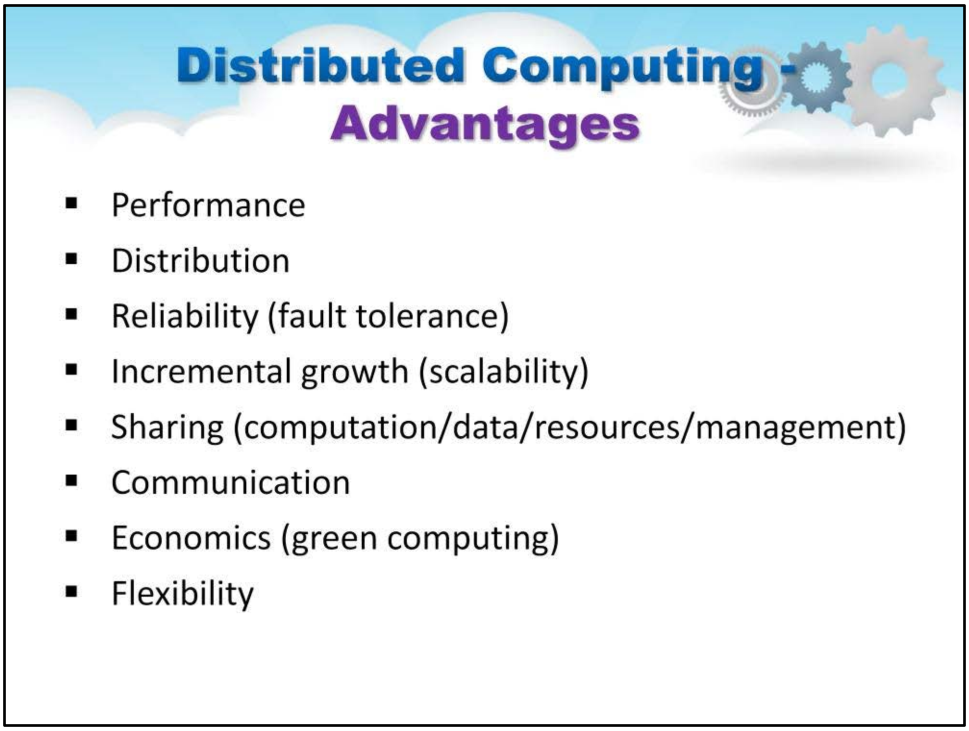
Timing failures: They are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time and clock drift rate. Any one of these failures may result in responses being unavailable to clients within a specified time interval.

Arbitrary failures: The term *arbitrary or Byzantine failure is used to describe the worst* possible failure semantics, in which any type of error may occur. For example, a process may set wrong values in its data items, or it may return a wrong value in response to an invocation.



Distributed Computing: Advantages, Disadvantages, Pitfalls

Advantages, Disadvantages, Pitfalls



Distributed Computing - Advantages

- Performance
- Distribution
- Reliability (fault tolerance)
- Incremental growth (scalability)
- Sharing (computation/data/resources/management)
- Communication
- Economics (green computing)
- Flexibility

Performance: very often a collection of processors can provide higher performance (and better price/performance ratio) than a centralized computer. Enhanced performance through load distributing.

Distribution: many applications are inherently distributed - involve, by their nature, spatially separated machines (banking, commercial, automotive system).

Reliability (fault tolerance): no single point of failure, if some of the machines crash, the system can survive.

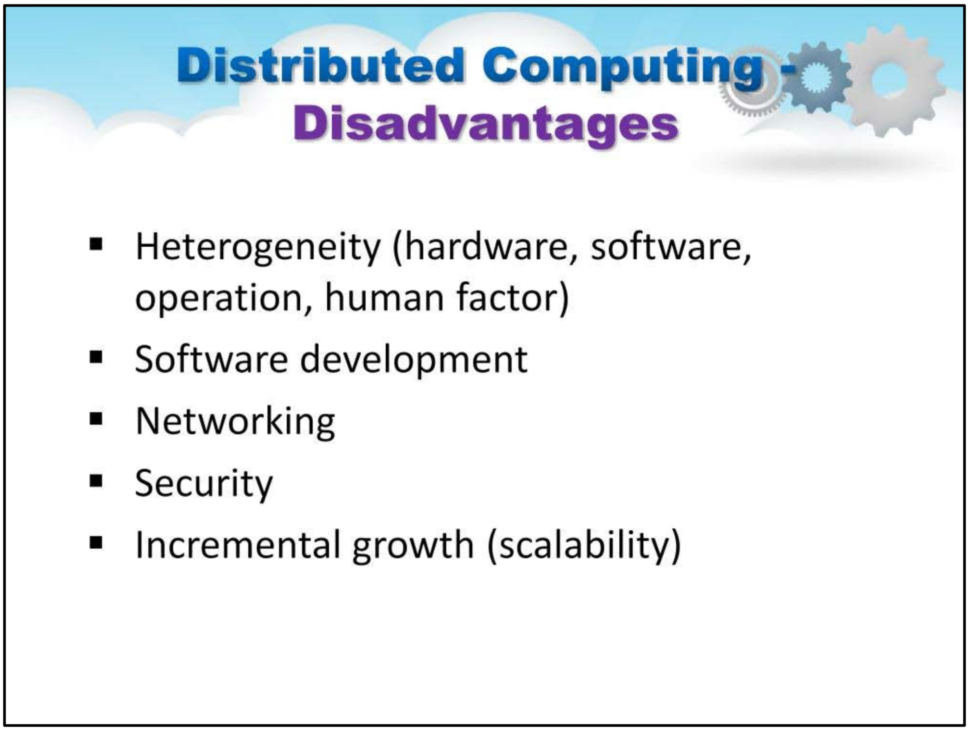
Incremental growth (scalability): as requirements on processing power grow, new machines can be added incrementally.

Sharing of computations/data/resources/management: complex computations can be shared; shared data is essential to many applications (banking, computer supported cooperative work, reservation systems); other resources can be also shared (e.g. expensive printers).

Communication: facilitates human-to-human communication.

Economics (green computing): lower (price/performance) ratio, efficient power consumption.

Flexibility: load balancing, spreading workload over many components.



Distributed Computing - Disadvantages

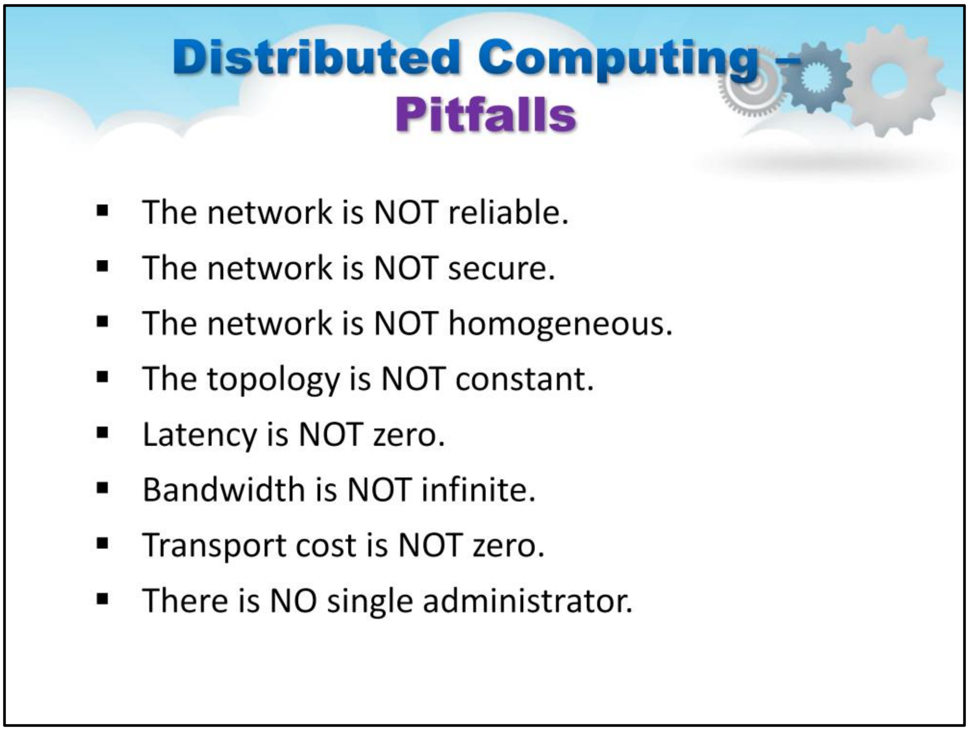
- Heterogeneity (hardware, software, operation, human factor)
- Software development
- Networking
- Security
- Incremental growth (scalability)

Difficulties of developing distributed software: how should operating systems, programming languages and applications look like? Developing a distributed system software is hard.

Networking problems: several problems are created by the network infrastructure, which have to be dealt with: loss of messages, overloading, etc. When network is overloaded/messages lost, rerouting/rewiring the network is costly/difficult.

Security problems: sharing generates the problem of data security. More sharing leads to less security especially in the issues of confidentiality & integrity.

Incremental growth: is hard in practice due to changing of hardware and software.



Distributed Computing – Pitfalls

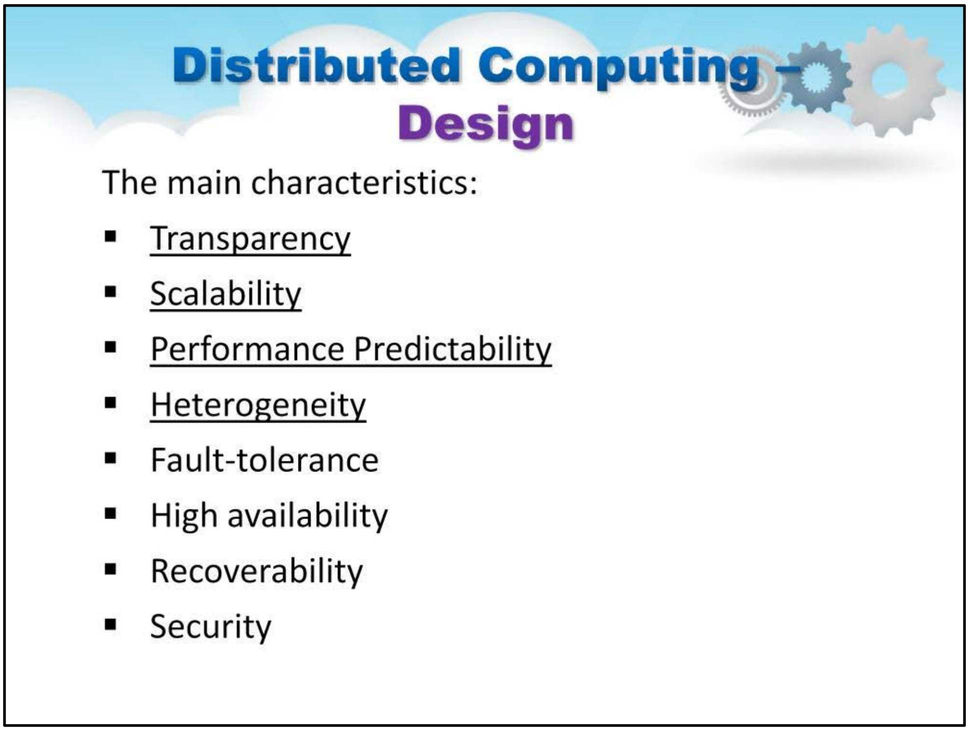
- The network is NOT reliable.
- The network is NOT secure.
- The network is NOT homogeneous.
- The topology is NOT constant.
- Latency is NOT zero.
- Bandwidth is NOT infinite.
- Transport cost is NOT zero.
- There is NO single administrator.

Distributed systems differ from traditional software because components are dispersed across a network. Not taking this dispersion into account during design time is what makes so many systems needlessly complex and results in mistakes that need to be patched later on.

Peter Deutsch, when he worked at Sun Microsystems company, formulated these mistakes as the following false assumptions that everyone makes when developing a distributed application for the first time.

Distributed Computing: Design

Design



Distributed Computing – Design

The main characteristics:

- Transparency
- Scalability
- Performance Predictability
- Heterogeneity
- Fault-tolerance
- High availability
- Recoverability
- Security

In this section we enlist several and consider 4 important chracteristics that should be taken into account to design and build an efficient distributed system.

A distributed system should

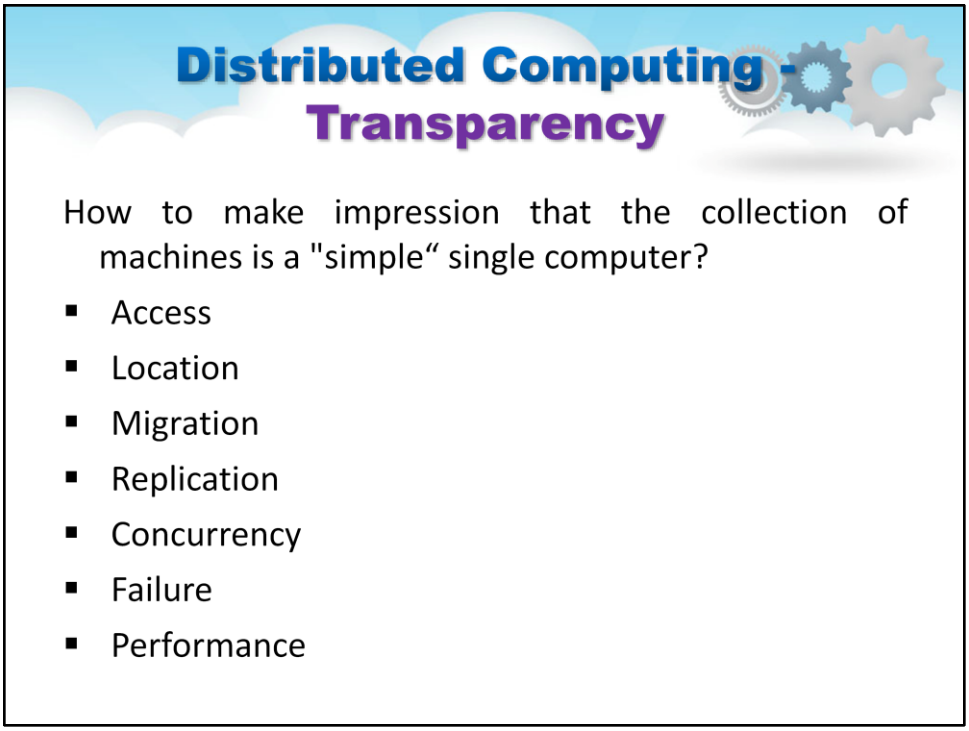
- make resources easily accessible;

- reasonably hide the fact that resources are distributed across a network;

- be open;

- be scalable;

- and others.

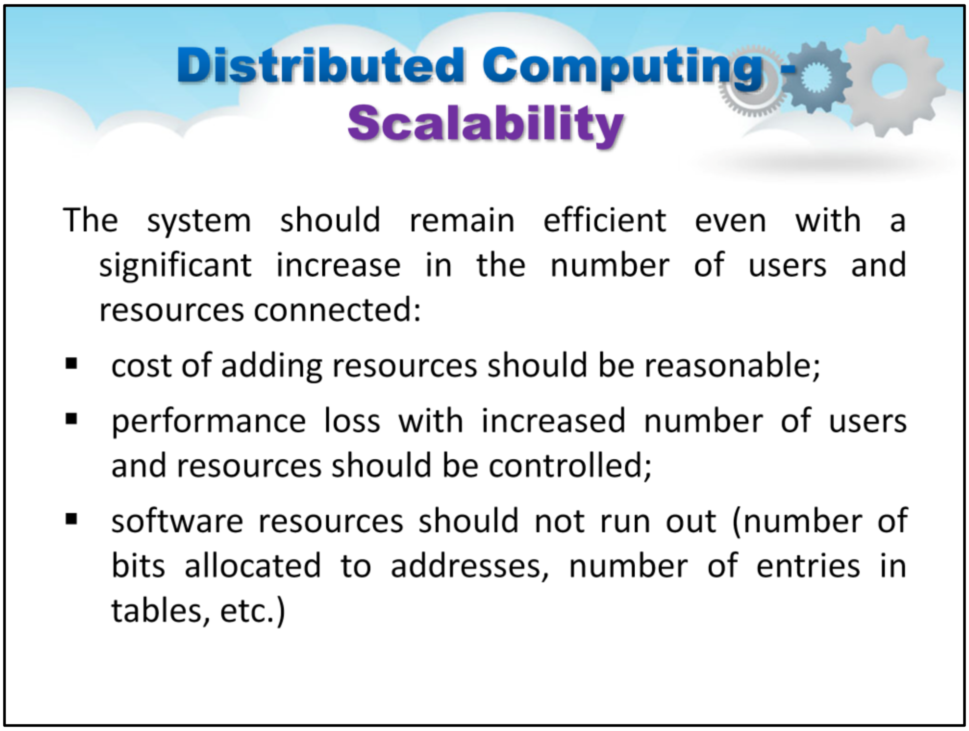


Distributed Computing - Transparency

How to make impression that the collection of machines is a "simple“ single computer?

- Access
- Location
- Migration
- Replication
- Concurrency
- Failure
- Performance

Access transparency: local and remote resources are accessed using identical operations.
Location transparency: users cannot tell where hardware and software resources (CPUs, files, data bases) are located; the name of the resource shouldn’t encode the location of the resource.
Migration (mobility) transparency: resources should be free to move from one location to another without having their names changed.
Replication transparency: the system is free to make additional copies of files and other resources (for purpose of performance and/or reliability), without the users noticing. Example: several copies of a file; at a certain request that copy is accessed which is the closest to the client.
Concurrency transparency: the users will not notice the existence of other users in the system (even if they access the same resources).
Failure transparency: applications should be able to complete their task despite failures occurring in certain components of the system.
Performance transparency: load variation should not lead to performance degradation. This could be achieved by automatic reconfiguration.



Distributed Computing - Scalability

The system should remain efficient even with a significant increase in the number of users and resources connected:

- cost of adding resources should be reasonable;
- performance loss with increased number of users and resources should be controlled;
- software resources should not run out (number of bits allocated to addresses, number of entries in tables, etc.)

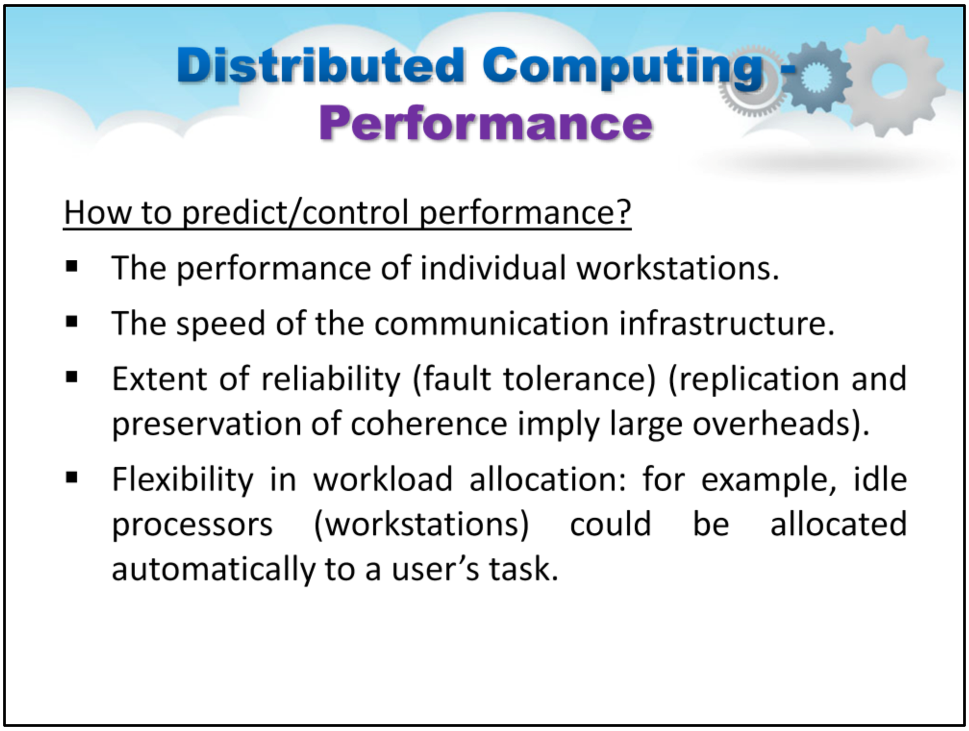
Scalability of a system can be measured along at least three different dimensions (Neuman, 1994):

1.First, a system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system.

2.Second, a geographically scalable system is one in which the users and resources may lie far apart.

3.Third, a system can be administratively scalable, meaning that it can still be easy to manage even if it spans many independent administrative organizations.

Unfortunately, a system that is scalable in one or more of these dimensions often exhibits some loss of performance as the system scales up.



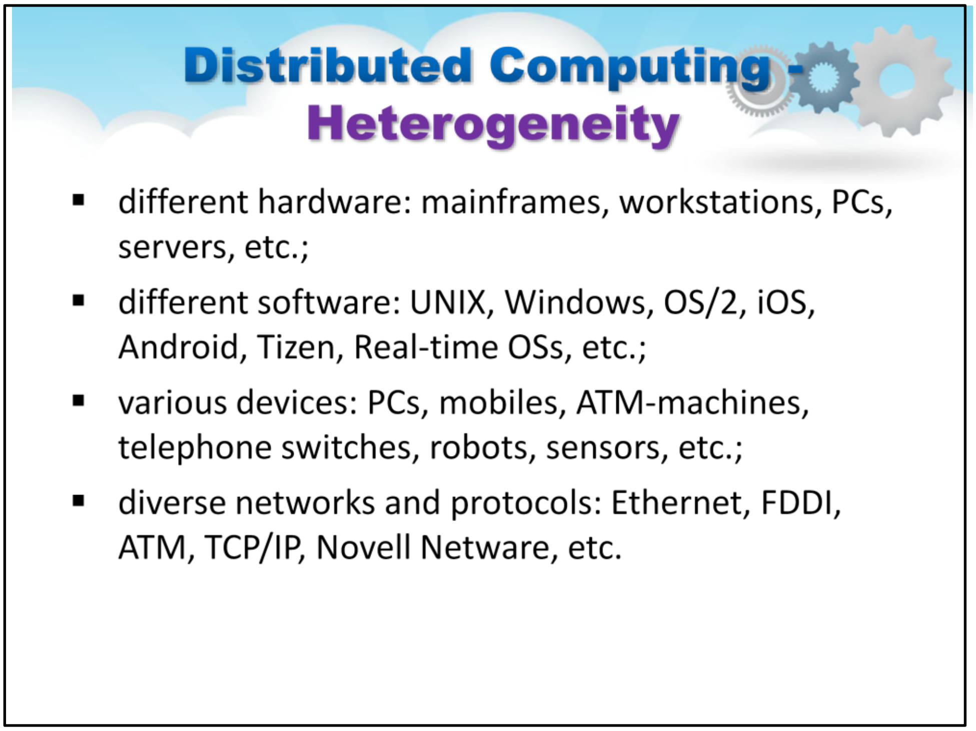
Distributed Computing - Performance

How to predict/control performance?

- The performance of individual workstations.
- The speed of the communication infrastructure.
- Extent of reliability (fault tolerance) (replication and preservation of coherence imply large overheads).
- Flexibility in workload allocation: for example, idle processors (workstations) could be allocated automatically to a user’s task.

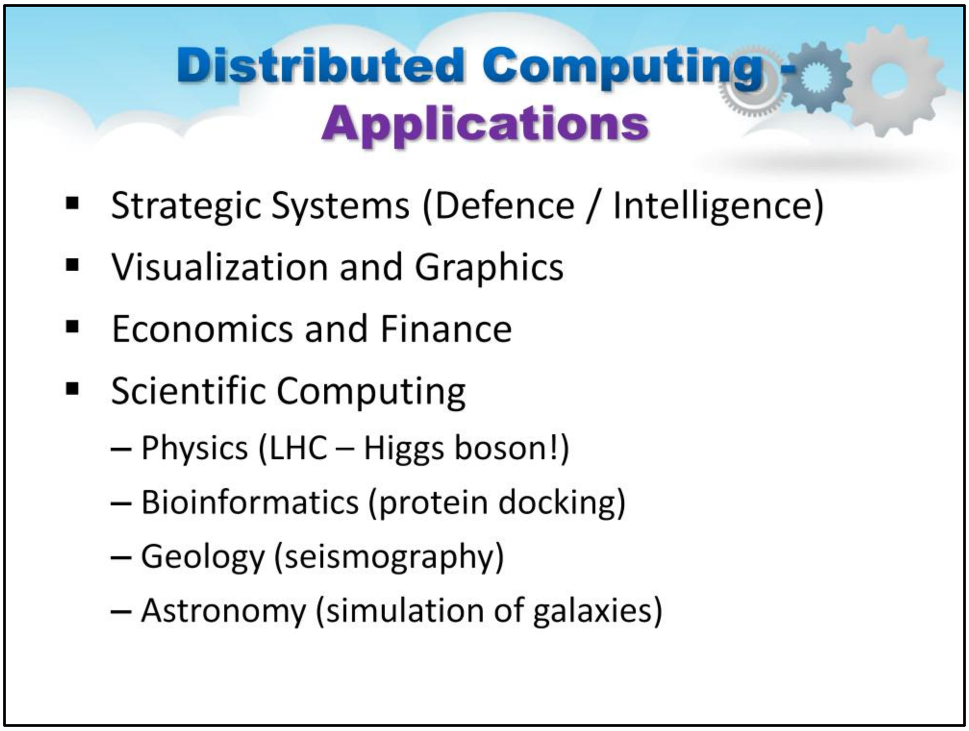
How to predict/control performance?

- The performance of individual workstations.
- The speed of the communication infrastructure.
- Extent of reliability (fault tolerance) (replication and preservation of coherence imply large overheads).
- Flexibility in workload allocation: for example, idle processors (workstations) could be allocated automatically to a user’s task.



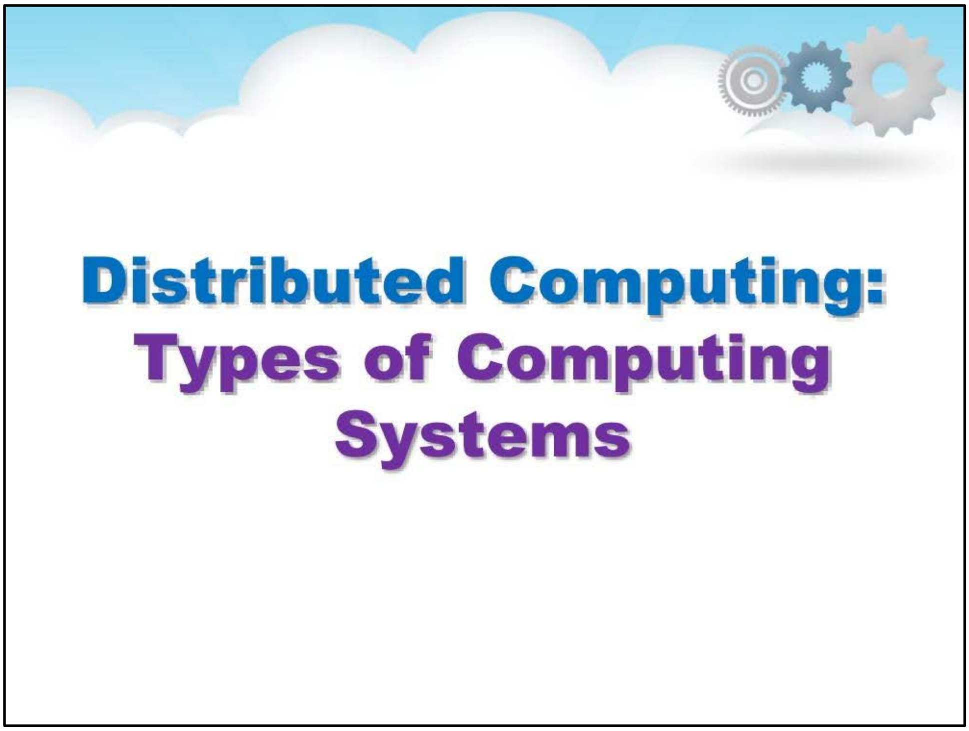
Distributed Computing - Heterogeneity

- different hardware: mainframes, workstations, PCs, servers, etc.;
- different software: UNIX, Windows, OS/2, iOS, Android, Tizen, Real-time OSs, etc.;
- various devices: PCs, mobiles, ATM-machines, telephone switches, robots, sensors, etc.;
- diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.



Distributed Computing - Applications

- Strategic Systems (Defence / Intelligence)
- Visualization and Graphics
- Economics and Finance
- Scientific Computing
  - Physics (LHC – Higgs boson!)
  - Bioinformatics (protein docking)
  - Geology (seismography)
  - Astronomy (simulation of galaxies)



Distributed Computing: Types of Computing Systems

Types of Computing Systems

Before starting to discuss the principles of distributed systems, let us first take a closer look at the various types of distributed systems. In the following we make a distinction between distributed computing systems, distributed information systems, and distributed embedded systems.

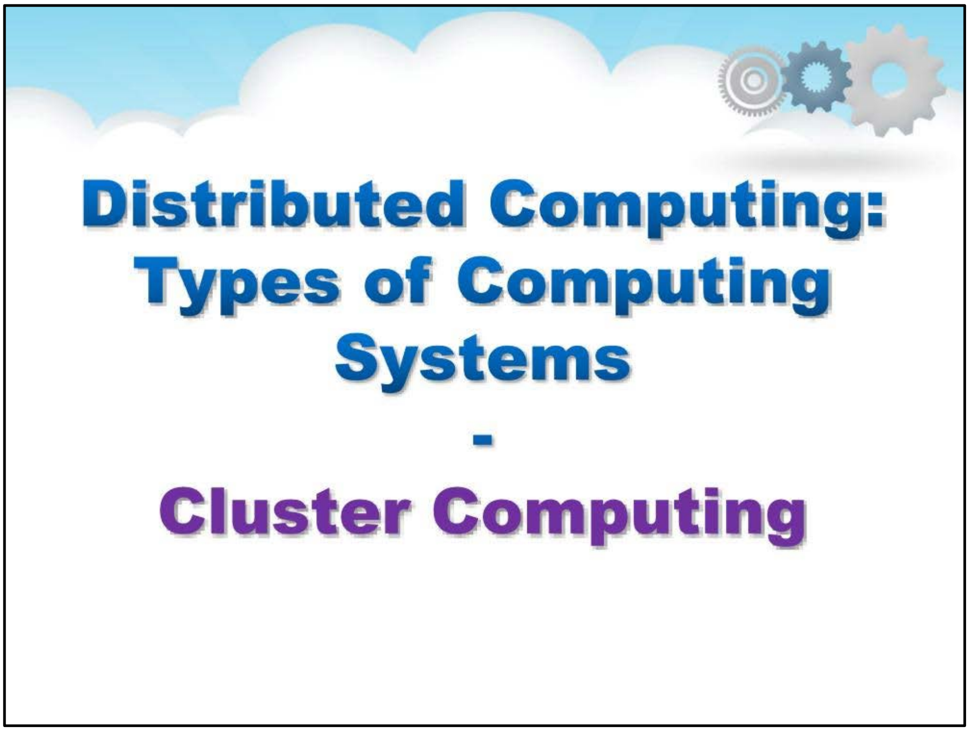
One can make a very rough distinction between two subgroups.

The closely connected computers:

for example, in cluster computing, the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed local-area network, where each node runs the same operating system.

The loosely connected computers:

for example, in grid computing, the underlying hardware consists of a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology



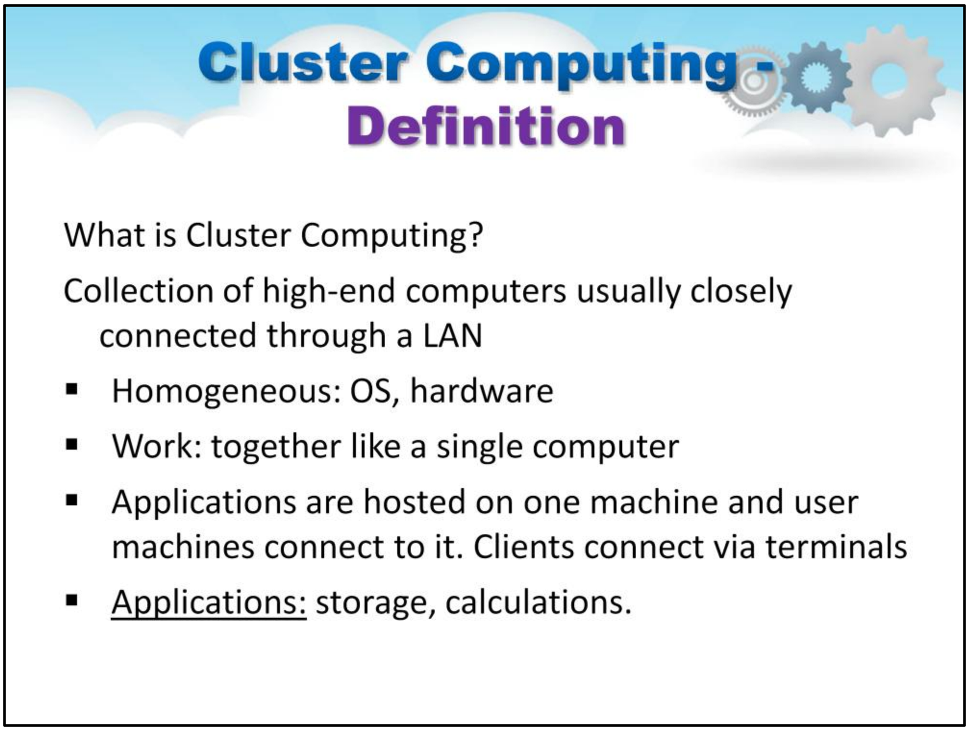
Distributed Computing: Types of Computing Systems

-

Cluster Computing

Cluster Computing

Let’s consider the cluster computing paradigm...



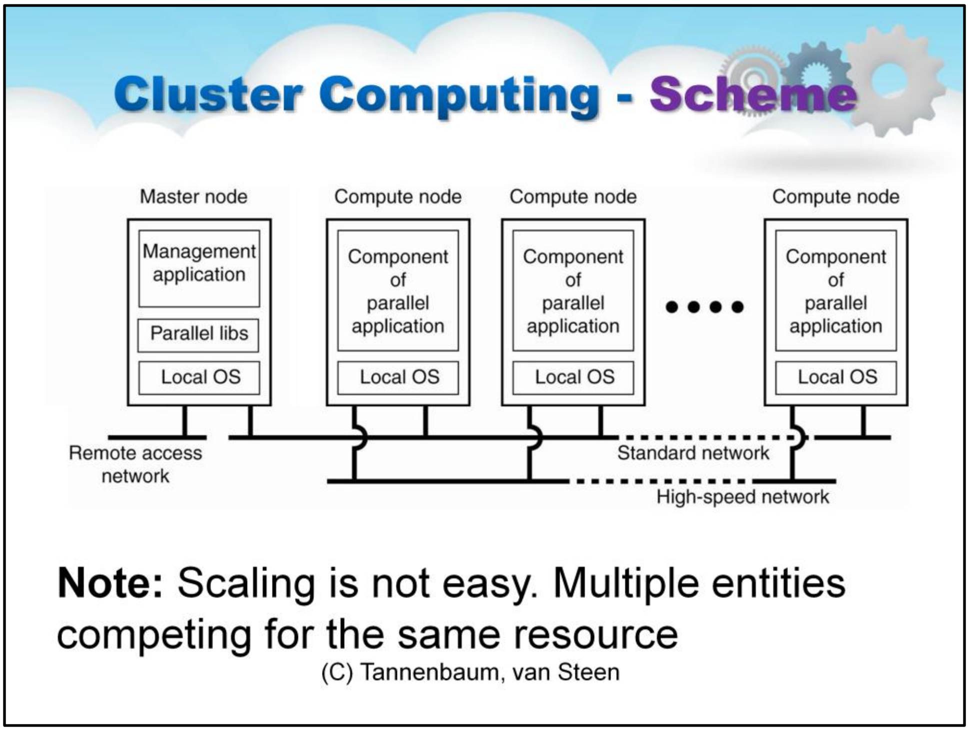
Cluster Computing - Definition

What is Cluster Computing?

Collection of high-end computers usually closely connected through a LAN

- Homogeneous: OS, hardware
- Work: together like a single computer
- Applications are hosted on one machine and user machines connect to it. Clients connect via terminals
- Applications: storage, calculations.

Cluster computing systems became popular when the price/performance ratio of personal computers and workstations improved. At a certain point, it became financially and technically attractive to build a supercomputer using off-the-shelf technology by simply hooking up a collection of relatively simple computers in a high-speed network. In virtually all cases, cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.



Cluster Computing - Scheme

Note: Scaling is not easy. Multiple entities competing for the same resource

(C) Tannenbaum, van Steen

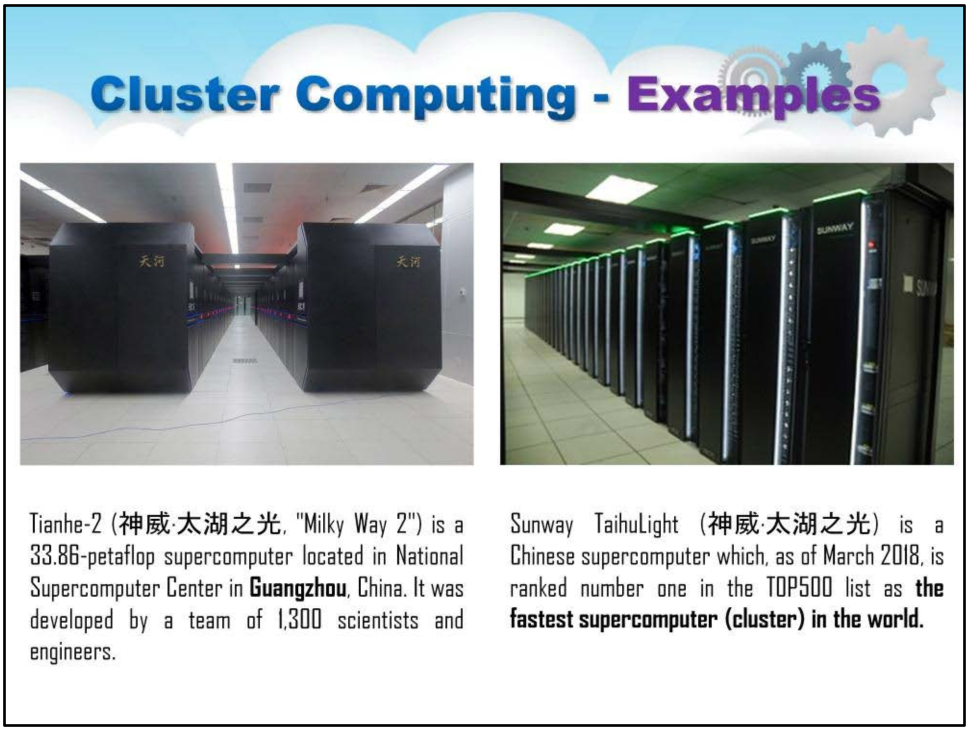
The typical example of a cluster computer is shown in this Figure.

Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node.

The master node typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface for the users of the system.

The master node actually runs the middleware, i.e. software needed for the execution of programs and management of the cluster, while the compute nodes often need nothing else but a standard operating system.

It should be noted that an important part of this middleware is based on the libraries for executing parallel programs, which effectively provide advanced message-based communication facilities only, but are not capable of handling faulty processes, security, etc.



Cluster Computing - Examples

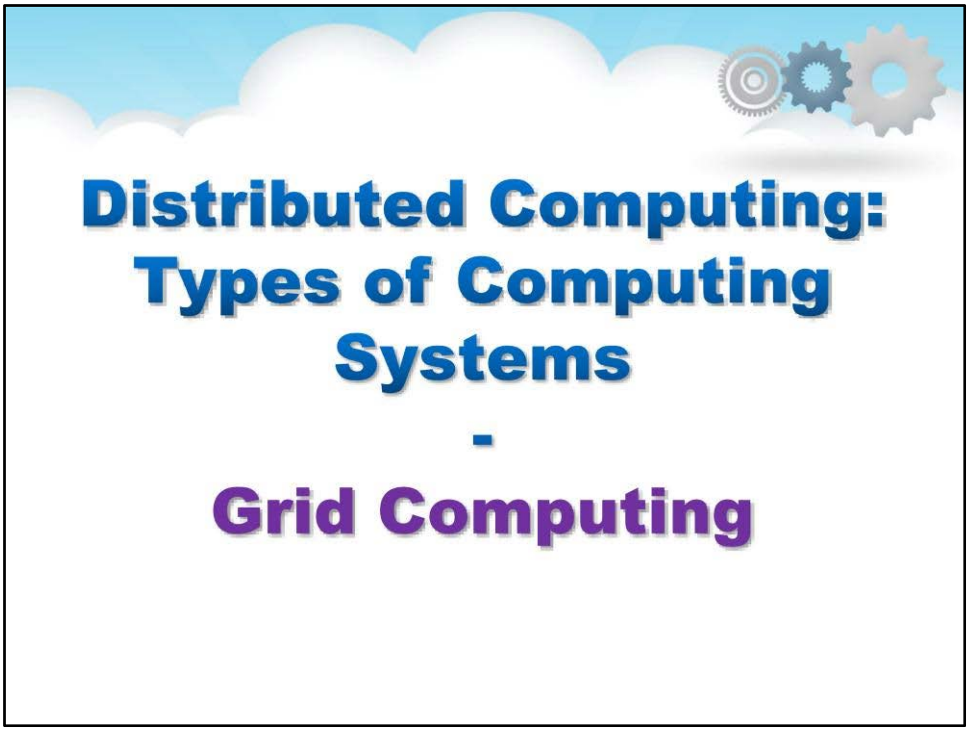
Tianhe-2 (神威·太湖之光, "Milky Way 2") is a 33.86-petaflop supercomputer located in National Supercomputer Center in **Guangzhou**, China. It was developed by a team of 1,300 scientists and engineers.

Sunway TaihuLight (神威·太湖之光) is a Chinese supercomputer which, as of March 2018, is ranked number one in the TOP500 list as **the fastest supercomputer (cluster) in the world.**

Tianhe-2 or TH-2 ("Milky Way 2") is a 33.86-petaflop supercomputer located in National Supercomputer Center in Guangzhou, China. It was developed by a team of 1,300 scientists and engineers.

The development of Tianhe-2 was sponsored by the 863 High Technology Program, initiated by the Chinese government, the government of Guangdong province, and the government of Guangzhou city. It was built by China's National University of Defense Technology (NUDT) in collaboration with the Chinese IT firm Inspur. Inspur manufactured the printed circuit boards and helped with the installation and testing of the system software. It was the world's fastest supercomputer according to the TOP500 lists from 2013 to 2015. The record was surpassed in June 2016 by the Sunway TaihuLight.

The Sunway TaihuLight (Chinese:神威·太湖之光, Shénwēi·tàihú zhī guāng) is a Chinese supercomputer which, as of March 2018, is ranked number one in the TOP500 list as the fastest supercomputer (cluster) in the world, with a LINPACK benchmark rating of 93 petaflops. This is nearly 3 times faster than the previous holder of the record, the Tianhe-2, which ran at 34 petaflops.
It was designed by the National Research Center of Parallel Computer Engineering & Technology (NRCPC) and is located at the National Supercomputing Center in Wuxi in the city of Wuxi, in Jiangsu province, China.



Distributed Computing: Types of Computing Systems

-

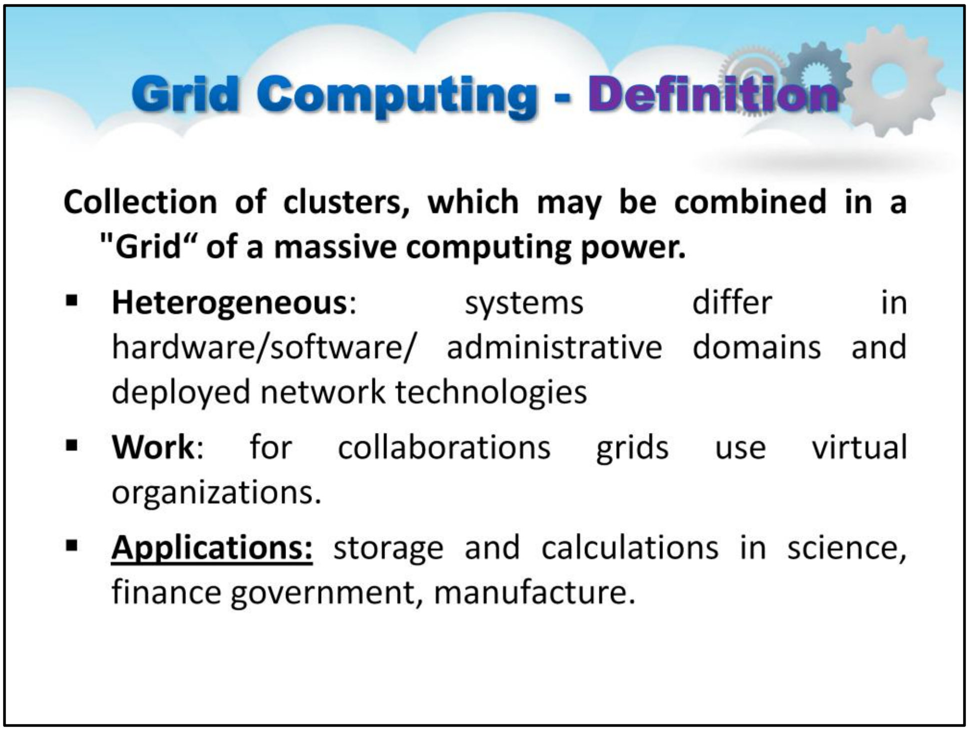
Grid Computing

Grid Computing

A characteristic feature of cluster computing is its homogeneity.

In most cases, the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network.

In contrast, grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.



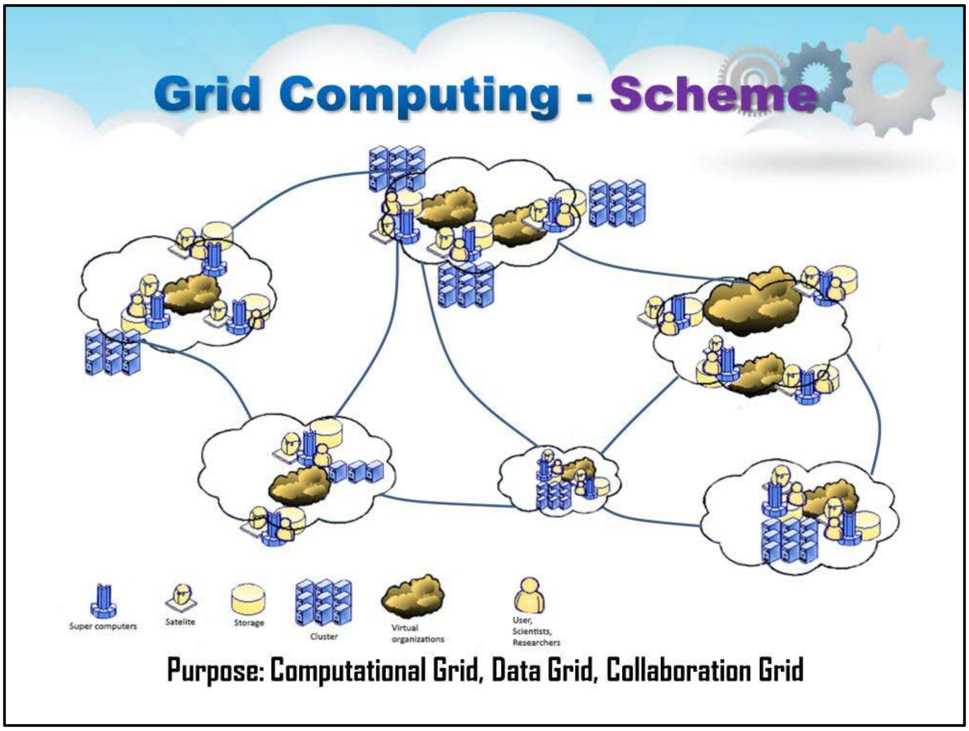
Grid Computing - Definition

Collection of clusters, which may be combined in a "Grid" of a massive computing power.

- **Heterogeneous**: systems differ in hardware/software/ administrative domains and deployed network technologies
- **Work**: for collaborations grids use virtual organizations.
- **Applications:** storage and calculations in science, finance government, manufacture.

A key issue in a grid computing system is that resources from different organizations are brought together to allow the collaboration of a group of people or institutions. Such a collaboration is realized in the form of a virtual organization.

The people belonging to the same virtual organization have access rights to the resources that are provided to that organization. Typically, resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.



Grid Computing - Scheme

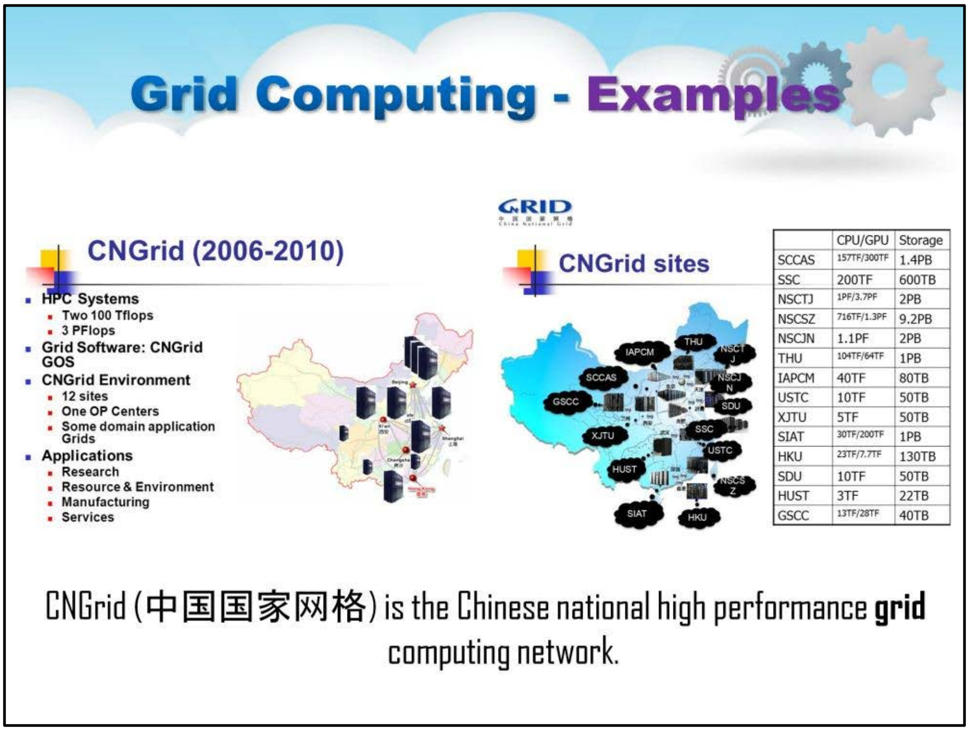
Purpose: Computational Grid, Data Grid, Collaboration Grid

Types of Grids:

Computational Grid: Shared Compute Resources

Data Grid: Access to Large amounts of Data spread across various sites

Collaboration Grid: multiple collaboration systems for collaborating on a common issue



Grid Computing - Examples

CNGrid (2006-2010)

- HPC Systems
  - Two 100 Tflops
  - 3 PFlops
- Grid Software: CNGrid GOS
- CNGrid Environment
  - 12 sites
  - One OP Centers
  - Some domain application Grids
- Applications
  - Research
  - Resource & Environment
  - Manufacturing
  - Services

CNGrid sites

	CPU/GPU	Storage
SCCAS	157TF/300TF	1.4PB
SSC	200TF	600TB
NSCTJ	1PF/3.7PF	2PB
NSCSZ	716TF/1.3PF	9.2PB
NSCJN	1.1PF	2PB
THU	104TF/64TF	1PB
IAPCM	40TF	80TB
USTC	10TF	50TB
XJTU	5TF	50TB
SIAT	30TF/200TF	1PB
HKU	23TF/7.7TF	130TB
SDU	10TF	50TB
HUST	3TF	22TB
GSCC	13TF/28TF	40TB

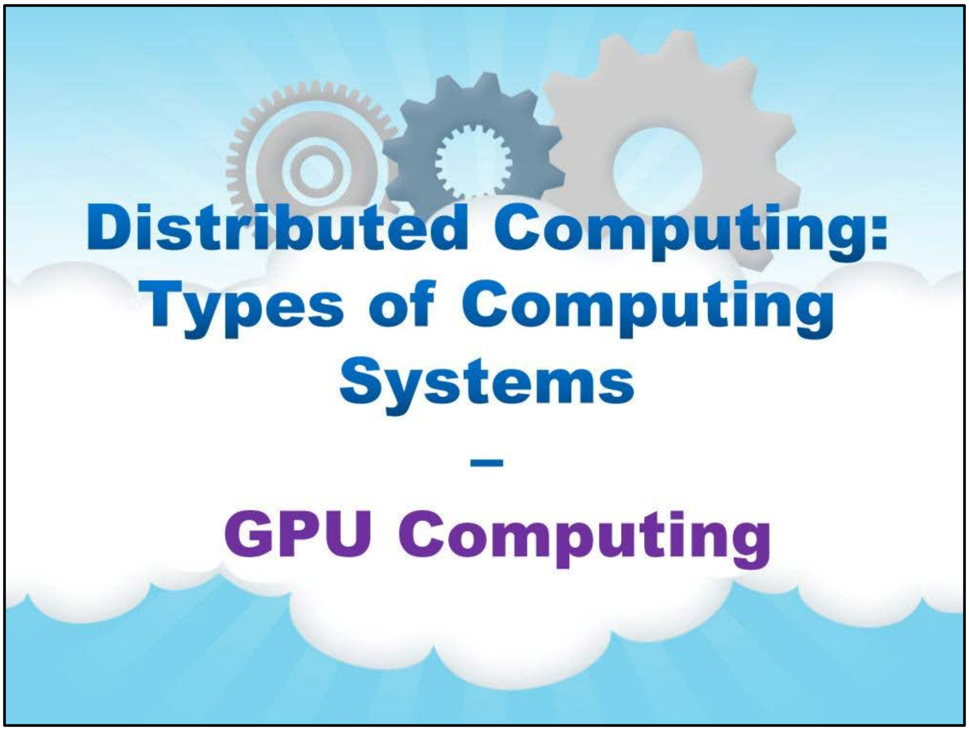
CNGrid (中国国家网格) is the Chinese national high performance **grid** computing network.

CNGrid (Chinese:中国国家网格) is the Chinese national high performance computing network supported by 863 Program.

China National Grid (CNGrid) is a major project supported by the Hi-Tech Research and Development (863) Program of China. CNGrid is the new generation test-bed of information infrastructure aggregating high-performance computing and transaction processing capabilities. Through resource sharing, work in coordination, and service mechanism, CNGrid effectively supports many applications such as scientific research, resource environment, advanced manufacturing, and information services.

CNGrid promotes the construction of national information industry and the development of related industries by technological innovations. China National Grid Software, named CNGrid GOS, is a suite of grid software with independent intellectual property, which is developed by CNGrid software R&D project team.

CNGrid GOS mainly includes a system software, a CA certificate management system and a testing environment, three business version of sub-systems (high performance computing gateway, data grid, and grid workflow), and a monitoring system. This project is undertaken by seven organizations including Institute of Computing Technology of Chinese Academy of Sciences, Jiangnan Institute of Computing Technology, Tsinghua University, National University of Defense Technology, Beihang University, Computer Network Information Center of Chinese Academy of Sciences and Shanghai Supercomputing Center.



Distributed Computing: Types of Computing Systems

–

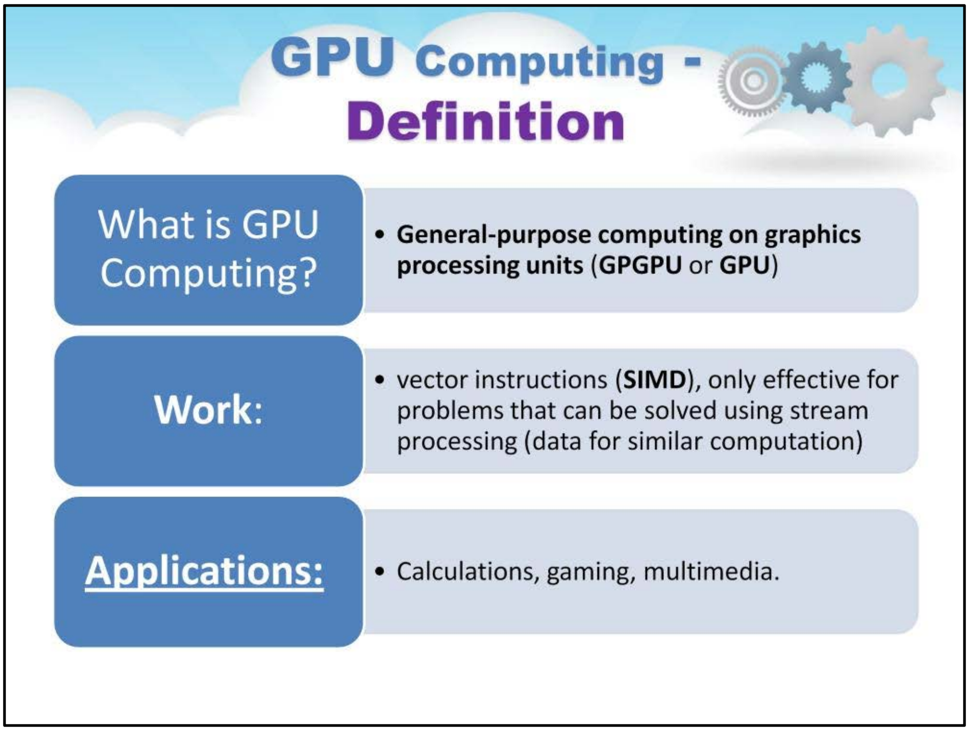
GPU Computing

GPU Computing

A GPU is a graphics coprocessor or accelerator mounted on a computer’s graphics card or video card. A GPU offloads the CPU from tedious graphics tasks in video editing applications.

The world’s first GPU, the GeForce 256, was marketed by NVIDIA in 1999. These GPU chips can process a minimum of 10 million polygons per second, and are used in nearly every computer on the market today. Some GPU features were also integrated into certain CPUs.

Traditional CPUs are structured with only a few cores. For example, the Xeon X5670 CPU has six cores. However, a modern GPU chip can be built with thousands of processing cores.



GPU Computing - Definition

What is GPU Computing?

- **General-purpose computing on graphics processing units (GPGPU** or **GPU**)

Work:

- vector instructions (**SIMD**), only effective for problems that can be solved using stream processing (data for similar computation)

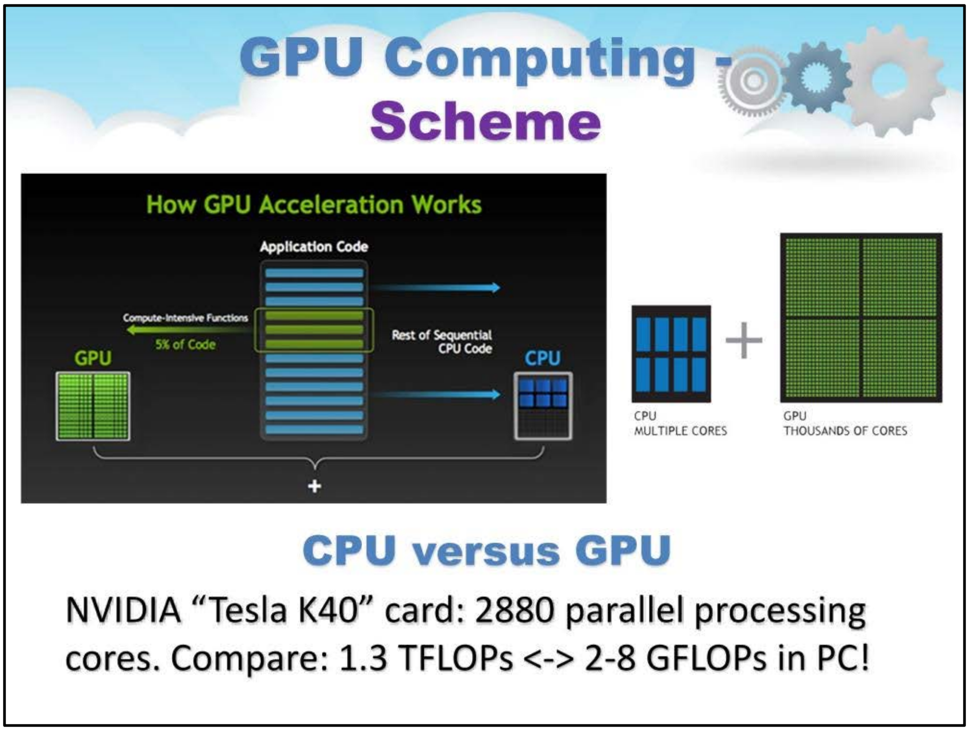
Applications:

- Calculations, gaming, multimedia.

SIMD - why it is distributed? – independent from CPU, several graphic cards can be integrated in PC, clusters, etc

Unlike CPUs, GPUs have a throughput architecture that exploits massive parallelism by executing many concurrent threads slowly, instead of executing a single long thread in a conventional microprocessor very quickly. Lately, parallel GPUs or GPU clusters have been garnering a lot of attention against the use of CPUs with limited parallelism.

General-purpose computing on GPUs, known as GPGPUs, have appeared in the HPC field.



GPU Computing - Scheme

How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential CPU Code
GPU
CPU
+
CPU
MULTIPLE CORES
+
GPU
THOUSANDS OF CORES

CPU versus GPU

NVIDIA “Tesla K40” card: 2880 parallel processing cores. Compare: 1.3 TFLOPs <-> 2-8 GFLOPs in PC!

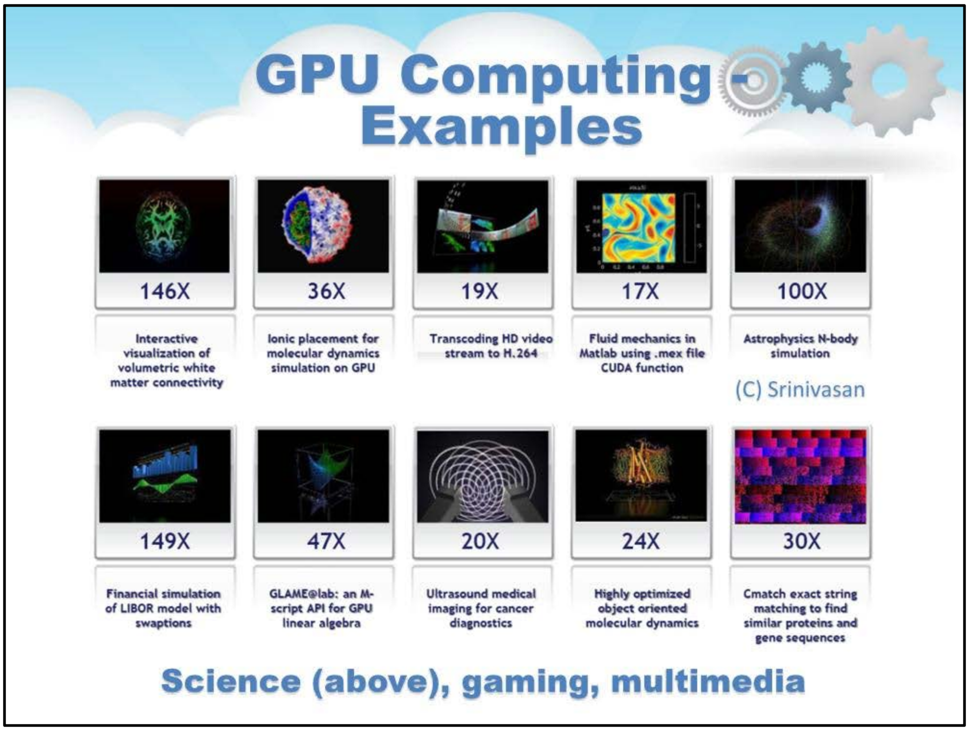
CPU VERSUS GPU

LEFT:

A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU consists of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.

RIGHT:

GPUs have thousands of cores to process parallel workloads efficiently



GPU Computing Examples

146X
Interactive visualization of volumetric white matter connectivity

36X
Ionic placement for molecular dynamics simulation on GPU

19X
Transcoding HD video stream to H.264

17X
Fluid mechanics in Matlab using .mex file CUDA function

100X
Astrophysics N-body simulation

(C) Srinivasan

149X
Financial simulation of LIBOR model with swaptions

47X
GLAME@lab: an M-script API for GPU linear algebra

20X
Ultrasound medical imaging for cancer diagnostics

24X
Highly optimized object oriented molecular dynamics

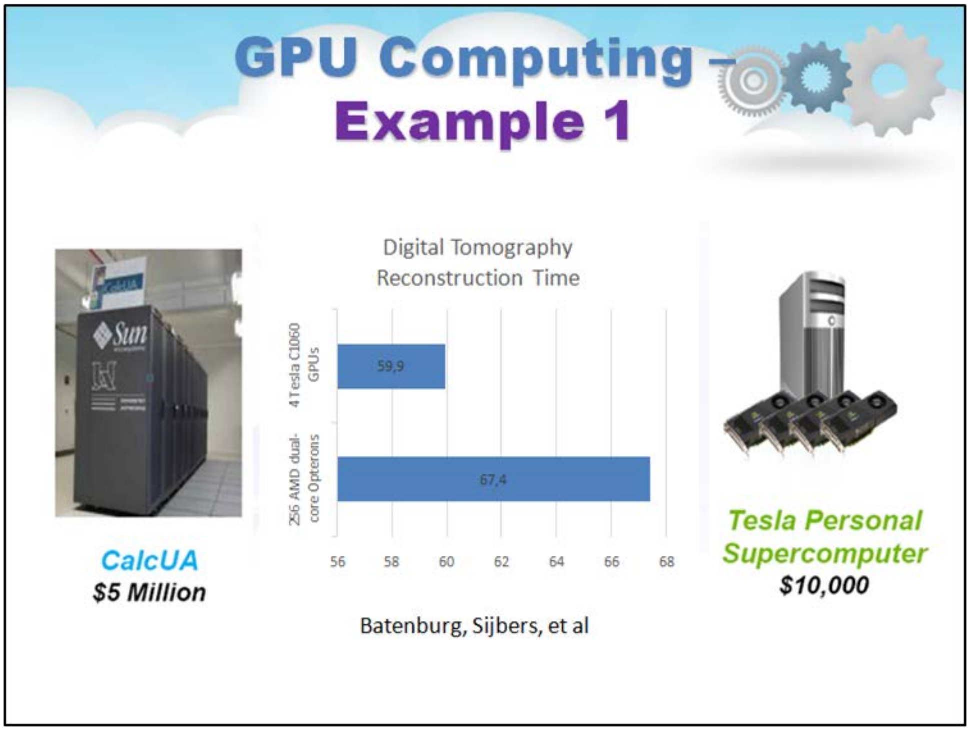
30X
Cmatch exact string matching to find similar proteins and gene sequences

Science (above), gaming, multimedia

Modern GPUs are not restricted to accelerated graphics or video coding. They are used in HPC systems to power supercomputers with massive parallelism at multicore and multithreading levels.

GPUs are designed to handle large numbers of floating-point operations in parallel. In a way, the GPU offloads the CPU from all data-intensive calculations, not just those that are related to video processing. Conventional GPUs are widely used in mobile phones, game consoles, embedded systems, PCs, and servers.

The NVIDIA CUDA Tesla, Fermi, or Volta cards are used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.



GPU Computing – Example 1

CalcUA
$5 Million

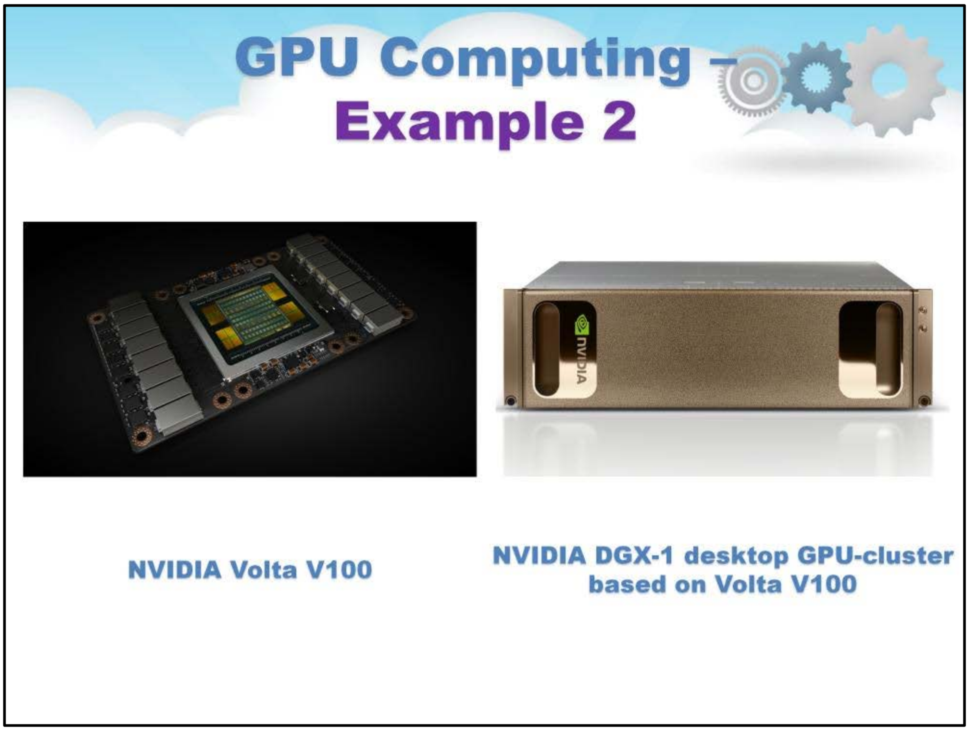
Tesla Personal Supercomputer
$10,000

Batenburg, Sijbers, et al

As today’s standard screening methods frequently fail to diagnose breast cancer before metastases have developed, earlier breast cancer diagnosis is still a major challenge. Three-dimensional ultrasound computer tomography promises high-quality images of the breast, but is currently limited by a time-consuming image reconstruction.

Medical imaging is one of the earliest applications to take advantage of GPU computing to get acceleration. The use of GPUs in this field has matured to the point that there are several medical modalities shipping with NVIDIA's Tesla GPUs now.

* CalcUA”, the supercomputer of the University of Antwerp, which cost 3.5 million euro in March 2005.



GPU Computing – Example 2

NVIDIA Volta V100

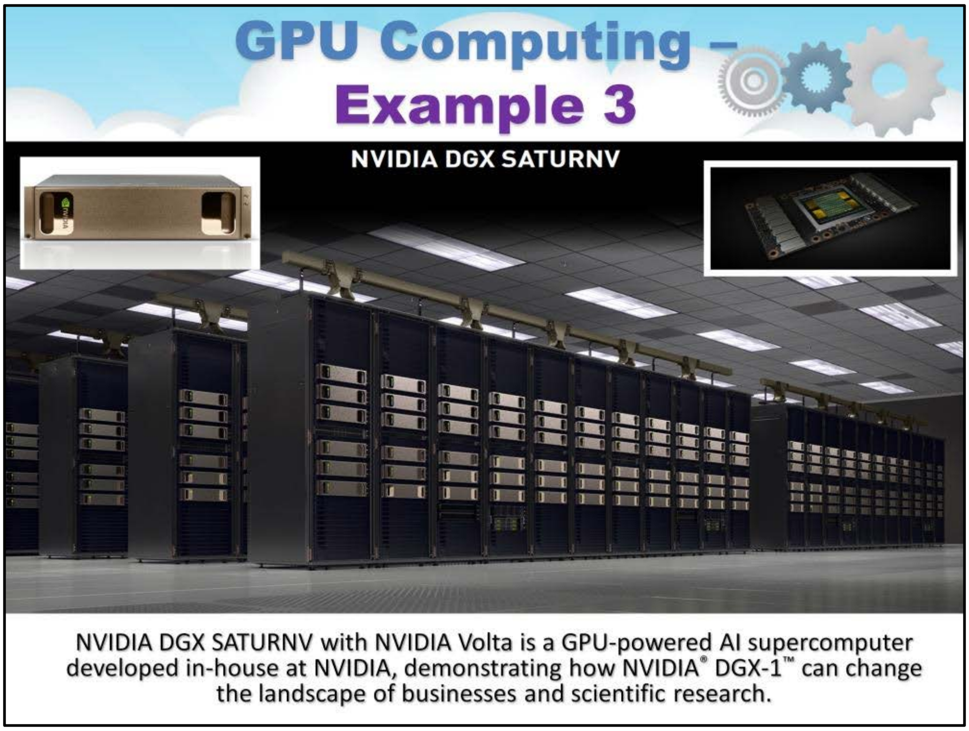
NVIDIA DGX-1 desktop GPU-cluster based on Volta V100

Volta uses next generation NVIDIA NVLink™ high-speed interconnect technology. This delivers 2X the throughput, compared to the previous generation of NVLink.

With over 21 billion transistors, Volta is the world’s most powerful GPU architecture, pairing NVIDIA CUDA® and Tensor Cores, delivering the performance of an AI supercomputer in a GPU.

With Volta-optimized CUDA and NVIDIA Deep Learning SDK libraries like cuDNN, NCCL, and TensorRT™, the industry's top frameworks and applications can easily tap into the power of Volta.

Equipped with 640 Tensor Cores, Volta delivers over 100 teraFLOPS of deep learning performance, a 5X increase compared to prior generation NVIDIA Pascal™ architecture.



GPU Computing – Example 3

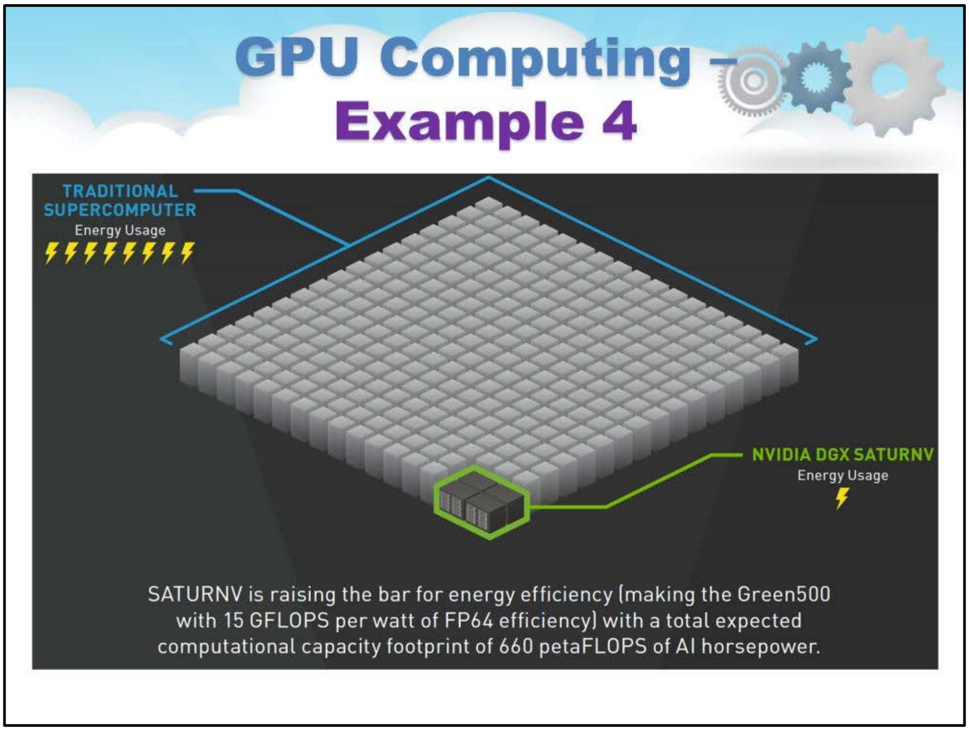
NVIDIA DGX SATURNV

NVIDIA DGX SATURNV with NVIDIA Volta is a GPU-powered AI supercomputer developed in-house at NVIDIA, demonstrating how NVIDIA® DGX-1™ can change the landscape of businesses and scientific research.

NVIDIA DGX SATURNV is POWERED BY NVIDIA TESLA® V100 GPUs and Built on the latest NVIDIA Volta GPU architecture.

with NVIDIA Volta is a GPU-powered AI supercomputer developed in-house at NVIDIA, demonstrating how NVIDIA® DGX-1™ can change the landscape of businesses and scientific research.

The opportunity for AI-accelerated companies is exploding as the world comes to rely on increasingly sophisticated products and services to build better customer experiences, transform supply chains, and improve product quality. NVIDIA leads the AI computing industry, and SATURNV is designed to enable moonshots—empowering the world's most brilliant minds to do their life's work.

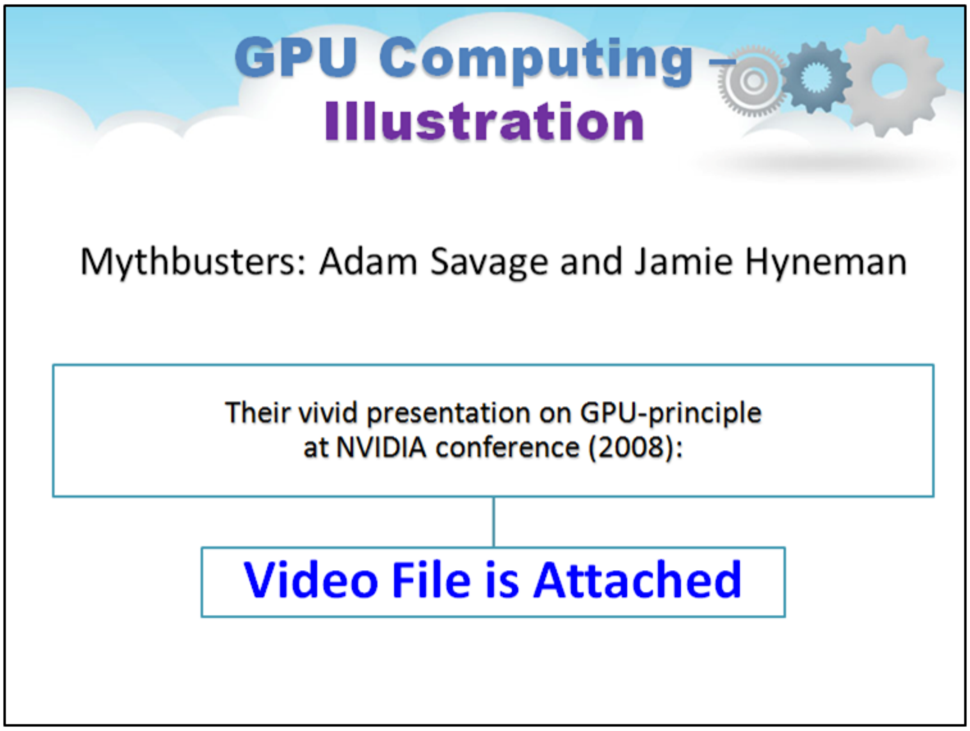


GPU Computing – Example 4

SATURNV is raising the bar for energy efficiency (making the Green500 with 15 GFLOPS per watt of FP64 efficiency) with a total expected computational capacity footprint of 660 petaFLOPS of AI horsepower.

GPU Computing is highly energy-efficient.

For example, DGX SATURNV is raising the bar for energy efficiency (making the Green500 with 15 GFLOPS per watt of FP64 efficiency) with a total expected computational capacity footprint of 660 petaFLOPS of AI horsepower.



GPU Computing – Illustration

Mythbusters: Adam Savage and Jamie Hyneman

Their vivid presentation on GPU-principle
at NVIDIA conference (2008):

Video File is Attached

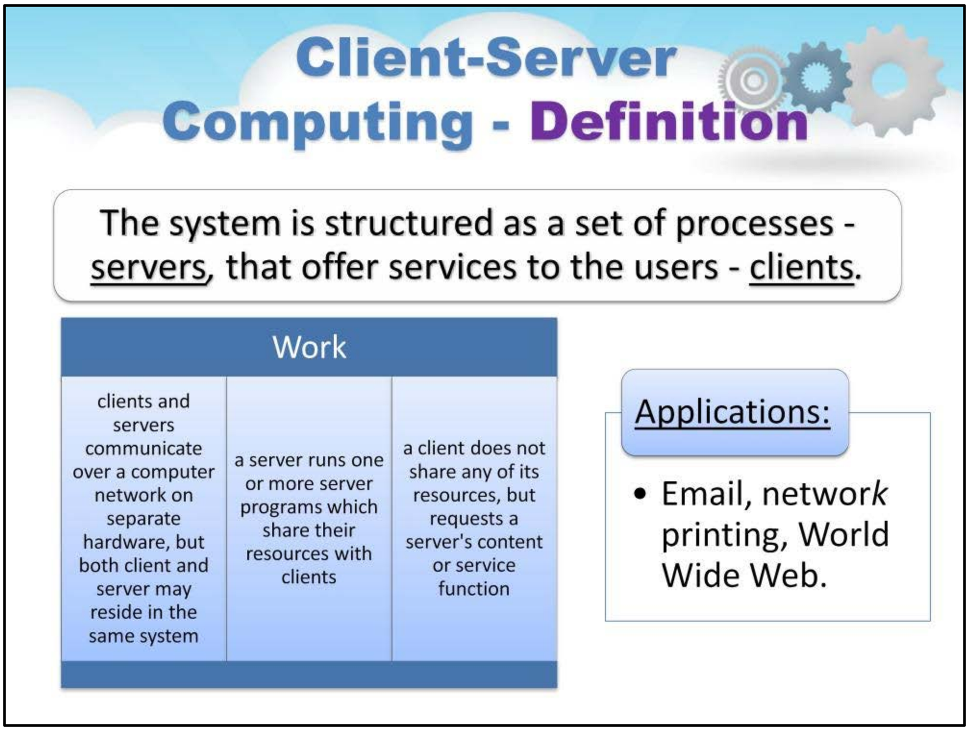
Let's see this video with the vivid demonstration of difference between CPU and GPU computing.

Distributed Computing: Types of Computing Systems

–

Client-Server Computing

Client-Server Computing



Client-Server Computing - Definition

The system is structured as a set of processes - servers, that offer services to the users - clients.

Work

- clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system
- a server runs one or more server programs which share their resources with clients
- a client does not share any of its resources, but requests a server's content or service function

Applications:

- Email, network printing, World Wide Web.

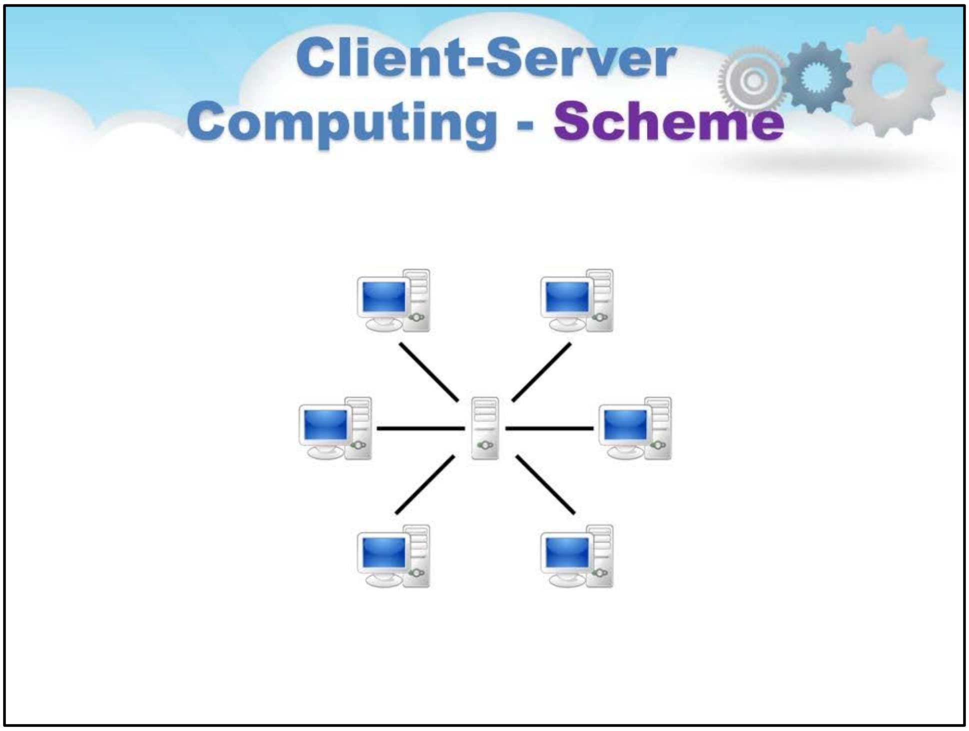
An example of a well-established distributed system is the client-server architecture. In this scenario, client machines (PCs and workstations) are connected to a central server for compute, e-mail, file access, and database applications.

The system is structured as a set of processes - servers, that offer services to the users – clients*:*

- clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system
- a server runs one or more server programs which share their resources with clients
- a client does not share any of its resources, but requests a server's content or service function

Applications:

Email, network printing, World Wide Web.



Client-Server Computing - Scheme

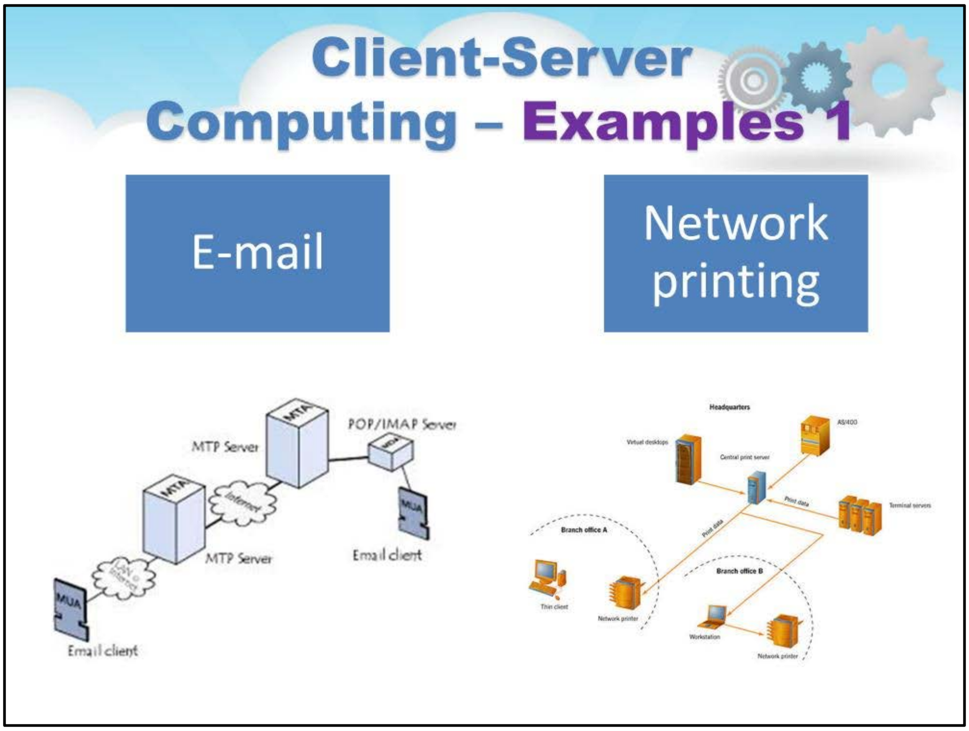
The system is structured as a set of processes - servers, that offer services to the users - clients.

Work:

- clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system

- a server runs one or more server programs which share their resources with clients

- a client does not share any of its resources, but requests a server's content or service function



Client-Server Computing – Examples 1

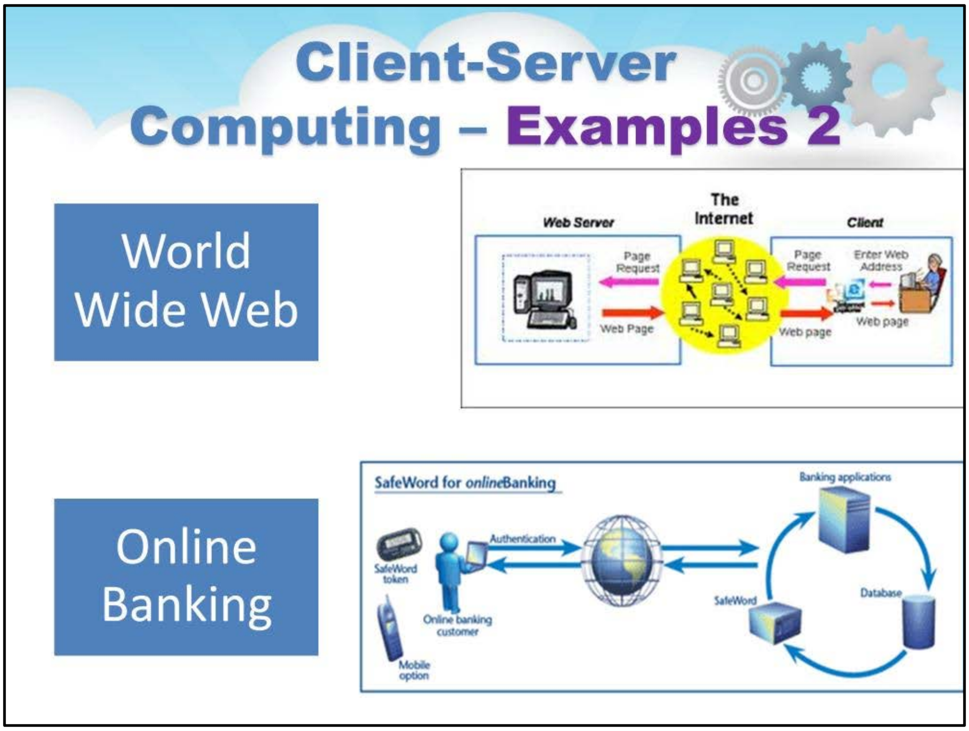
E-mail

Network printing

Here you can see several examples of the most popular client-server systems:

Electronic mail, most commonly referred to as email or e-mail since ca. 1993, is a method of exchanging digital messages from an author to one or more recipients. Modern email operates across the Internet or other computer networks. Some early email systems required that the author and the recipient both be online at the same time, in common with instant messaging. Today's email systems are based on a store-and-forward model. Email servers accept, forward, deliver, and store messages. Neither the users nor their computers are required to be online simultaneously; they need connect only briefly, typically to a mail server, for as long as it takes to send or receive messages.

Network printing: A print server, or printer server, is a device that connects printers to client computers over a network. It accepts print jobs from the computers and sends the jobs to the appropriate printers, queuing the jobs locally to accommodate the fact that work may arrive more quickly than the printer can actually handle it. Ancillary functions include the ability to inspect the queue of jobs to be processed, the ability to reorder or delete waiting print jobs, or the ability to do various kinds of accounting (such as counting pages printer, which may involve reading data generated by the printer(s)).



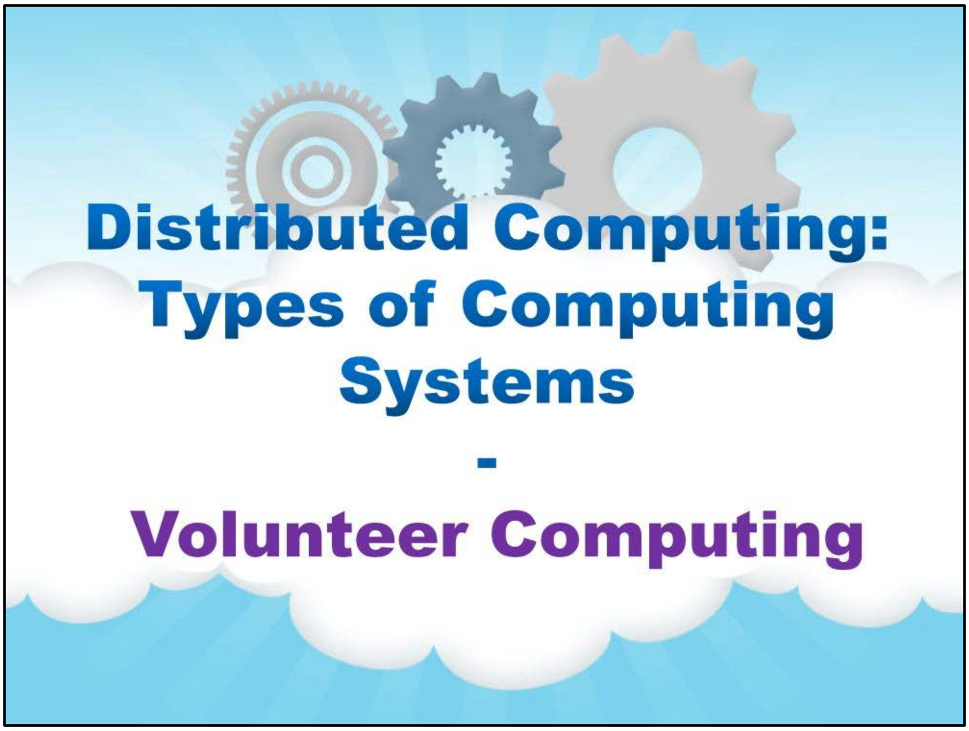
Client-Server Computing – Examples 2

World Wide Web

Online Banking

The World Wide Web is a system of interlinked hypertext documents that are accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them via hyperlinks.

Online banking, also known as internet banking, it is an electronic payment system that enables customers of a bank or other financial institution to conduct a range of financial transactions through the financial institution's website. The online banking system will typically connect to or be part of the core banking system operated by a bank and is in contrast to branch banking which was the traditional way customers accessed banking services.

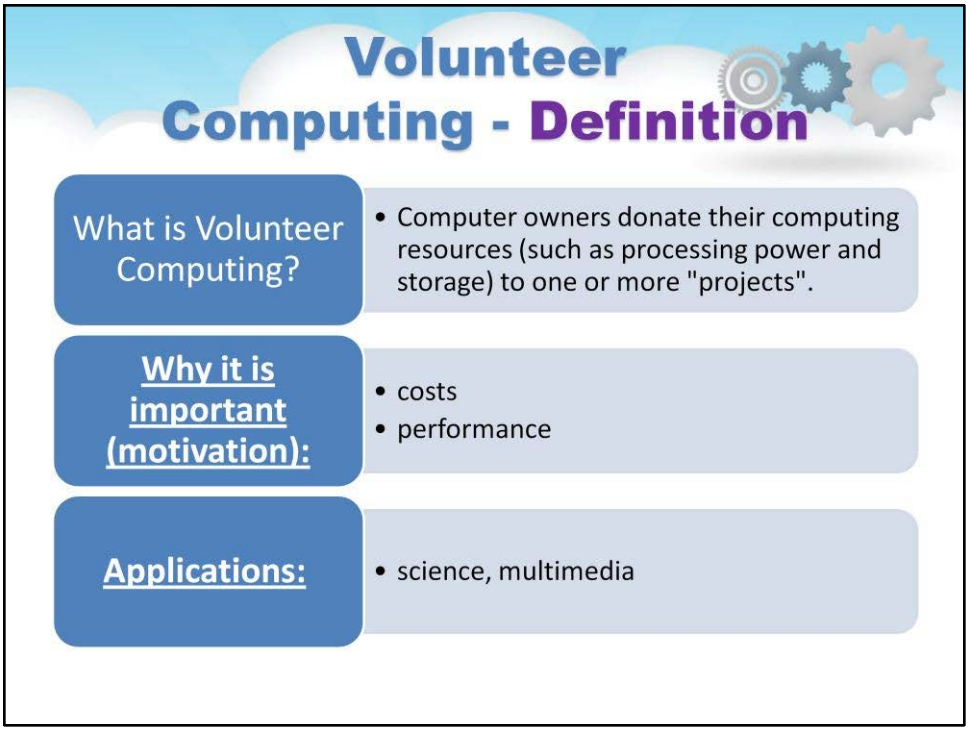


Distributed Computing: Types of Computing Systems

-

Volunteer Computing

Volunteer Computing



Volunteer Computing - Definition

What is Volunteer Computing?

- Computer owners donate their computing resources (such as processing power and storage) to one or more "projects".

Why it is important (motivation):

- costs
- performance

Applications:

- science, multimedia

What is Volunteer Computing?

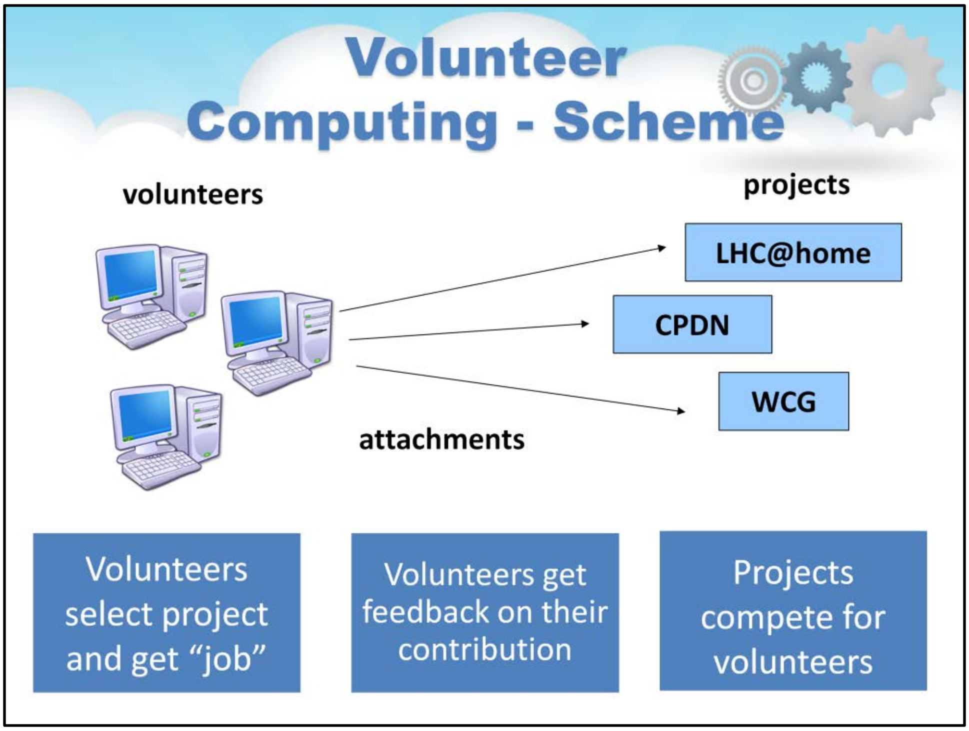
Computer owners donate their computing resources (such as processing power and storage) to one or more "projects".

Why it is important (motivation):

- low costs
- high performance

Applications:

science, multimedia



Volunteer Computing - Scheme

volunteers

projects

LHC@home

CPDN

WCG

attachments

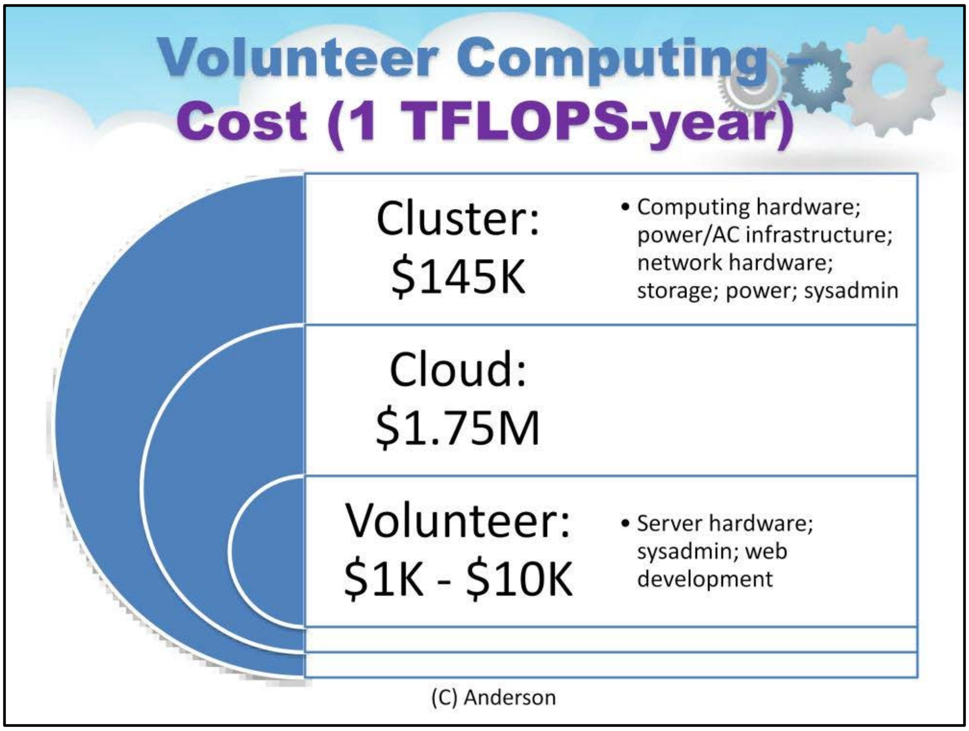
Volunteers select project and get “job”

Volunteers get feedback on their contribution

Projects compete for volunteers

The basic workflow is like this:

- Volunteers select project and get “job”
- Volunteers get feedback on their contribution
- Projects compete for volunteers



Volunteer Computing – Cost (1 TFLOPS-year)

Cluster: $145K

- Computing hardware; power/AC infrastructure; network hardware; storage; power; sysadmin

Cloud: $1.75M

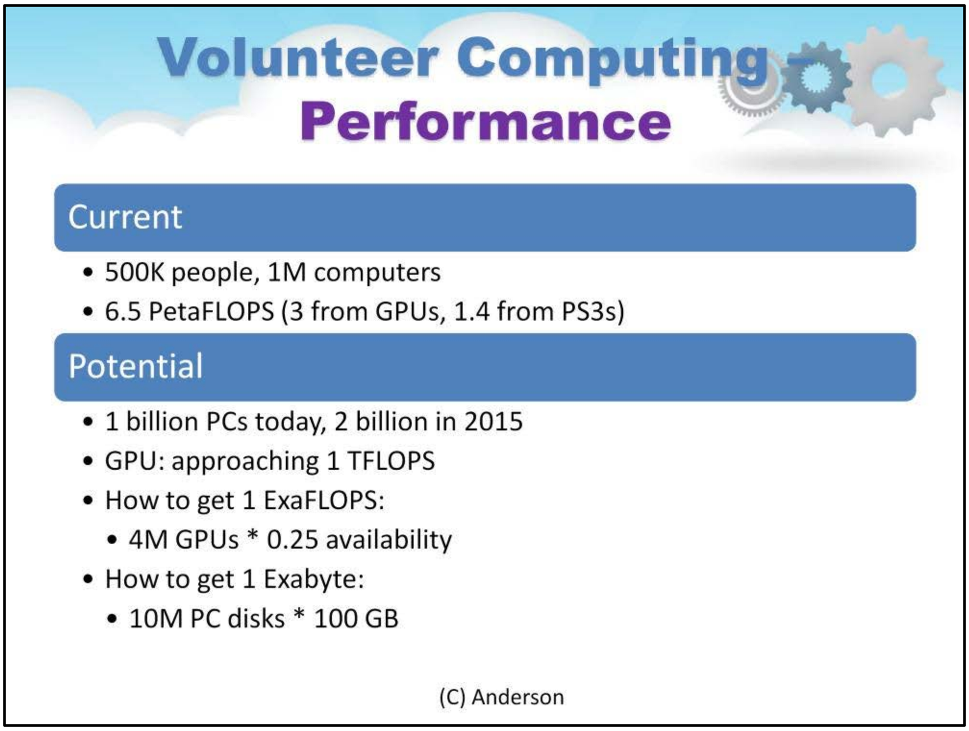
Volunteer: $1K - $10K

- Server hardware; sysadmin; web development

(C) Anderson

The typical budget for construction and operation of various infrastructures:

Cluster: $145K
Computing hardware; power/AC infrastructure; network hardware; storage; power; sysadmin
Cloud: $1.75M
Volunteer: $1K - $10K
Server hardware; sysadmin; web development



Volunteer Computing – Performance

Current

- 500K people, 1M computers
- 6.5 PetaFLOPS (3 from GPUs, 1.4 from PS3s)

Potential

- 1 billion PCs today, 2 billion in 2015
- GPU: approaching 1 TFLOPS
- How to get 1 ExaFLOPS:
  - 4M GPUs * 0.25 availability
- How to get 1 Exabyte:
  - 10M PC disks * 100 GB

(C) Anderson

The cumulative performance of all volunteer computing systems are as follows:

Current

500K people, 1M computers

6.5 PetaFLOPS (3 from GPUs, 1.4 from PS3s)

Potential

1 billion PCs today, 2 billion in 2015

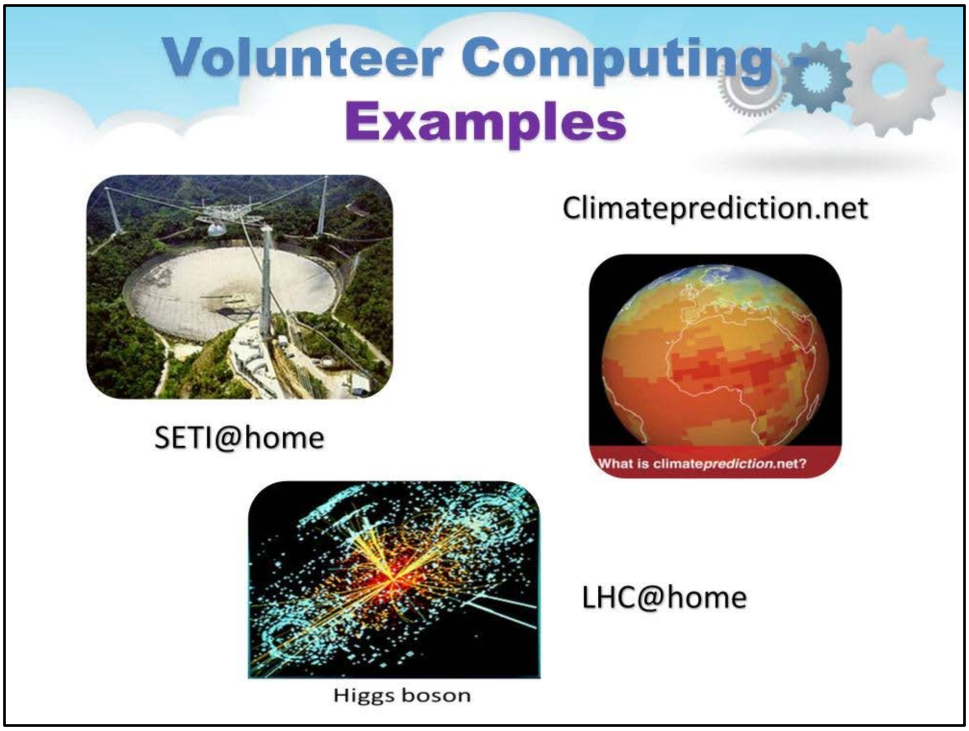
GPU: approaching 1 TFLOPS

How to get 1 ExaFLOPS:

4M GPUs * 0.25 availability

How to get 1 Exabyte:

10M PC disks * 100 GB



Volunteer Computing - Examples

Climateprediction.net

SETI@home

What is climateprediction.net?

LHC@home

Higgs boson

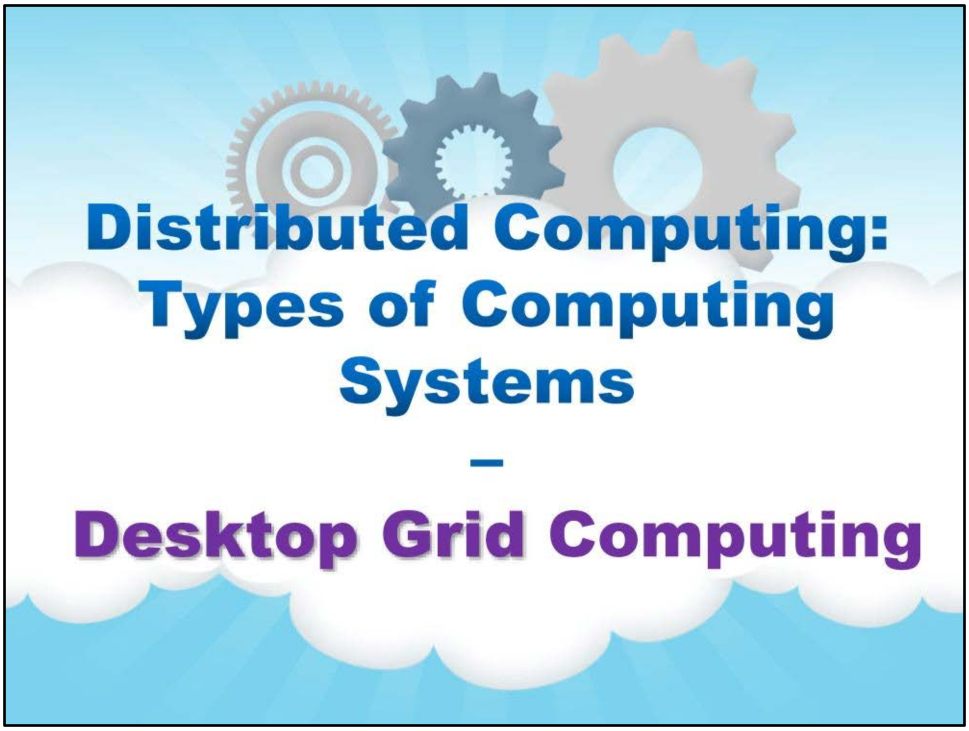
Volunteer computing is crucially important, especially to projects that have limited funding.

SETI@home ("SETI at home") is an Internet-based public volunteer computing project employing the BOINC software platform created by the Berkeley SETI Research Center and is hosted by the Space Sciences Laboratory, at the University of California, Berkeley.

Its purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence, and as such is one of many activities undertaken as part of the worldwide SETI effort.

SETI@home is oldest and the third large-scale use of distributed computing over the Internet for research purposes, after Great Internet Mersenne Prime Search (GIMPS) was launched in 1996 and distributed.net in 1997.

Along with MilkyWay@home and Einstein@home, it is the third major computing project of this type that has the investigation of phenomena in interstellar space as its primary purpose.



Distributed Computing: Types of Computing Systems

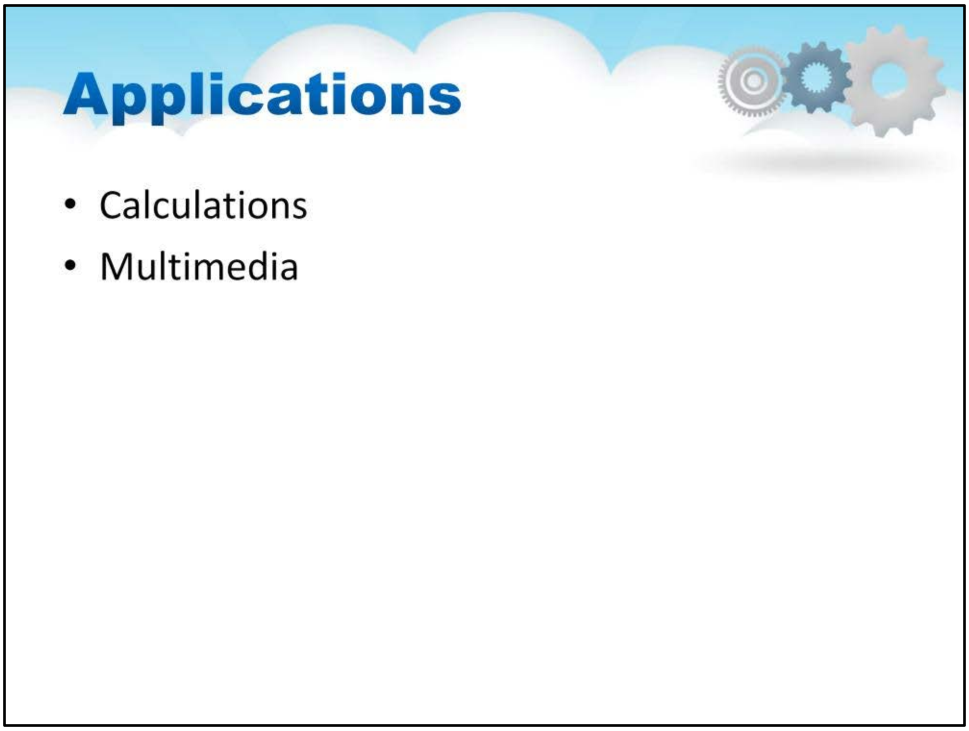
–

Desktop Grid Computing

Desktop Grid Computing

What is Desktop Grid Computing?

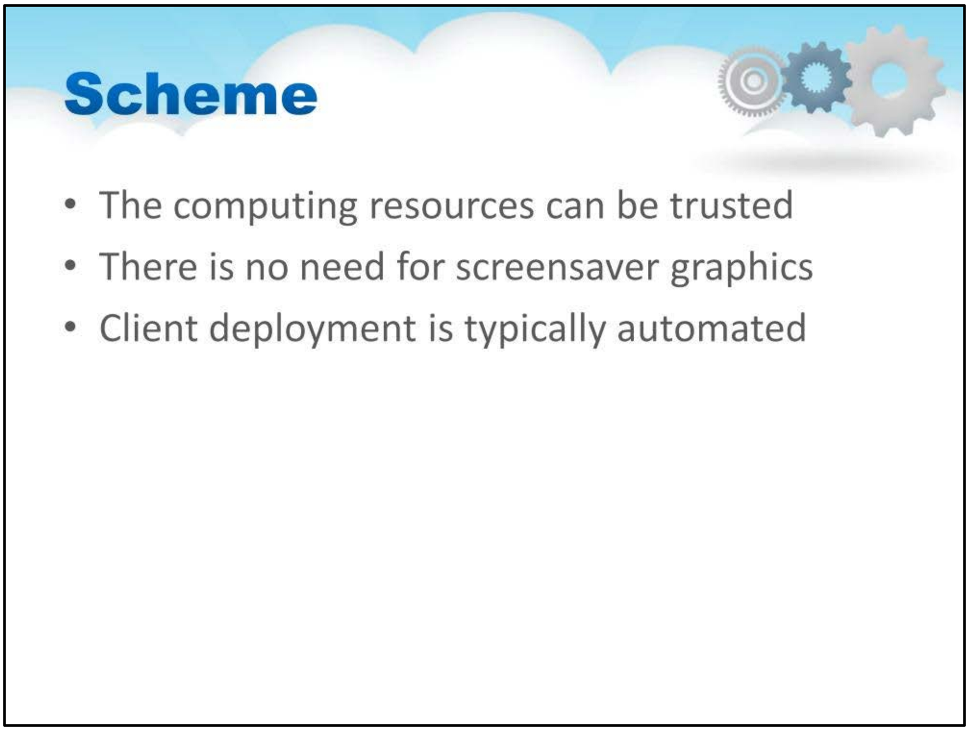
A form of distributed computing in which an organization (business, university, etc.) uses its existing computers (desktop and/or cluster nodes) to handle its own long-running computational tasks.



Applications

- Calculations
- Multimedia

Applications of desktop grid computing: calculations, multimedia

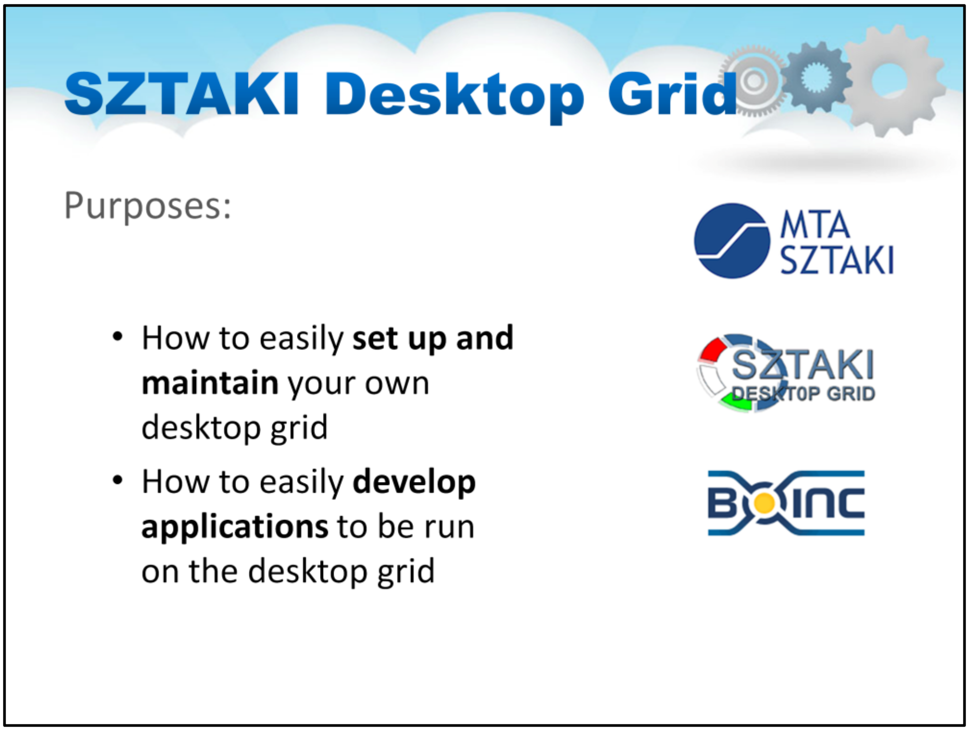


Scheme

- The computing resources can be trusted
- There is no need for screensaver graphics
- Client deployment is typically automated

It is similar to Volunteer Computing, but… differs:

• The computing resources can be trusted; i.e. one can assume that the PCs don't return results that are intentionally wrong or falsified
• There is no need for screensaver graphics; in fact it may be desirable to have the computation be completely invisible and out of the control of the PC user
• Client deployment is typically automated.



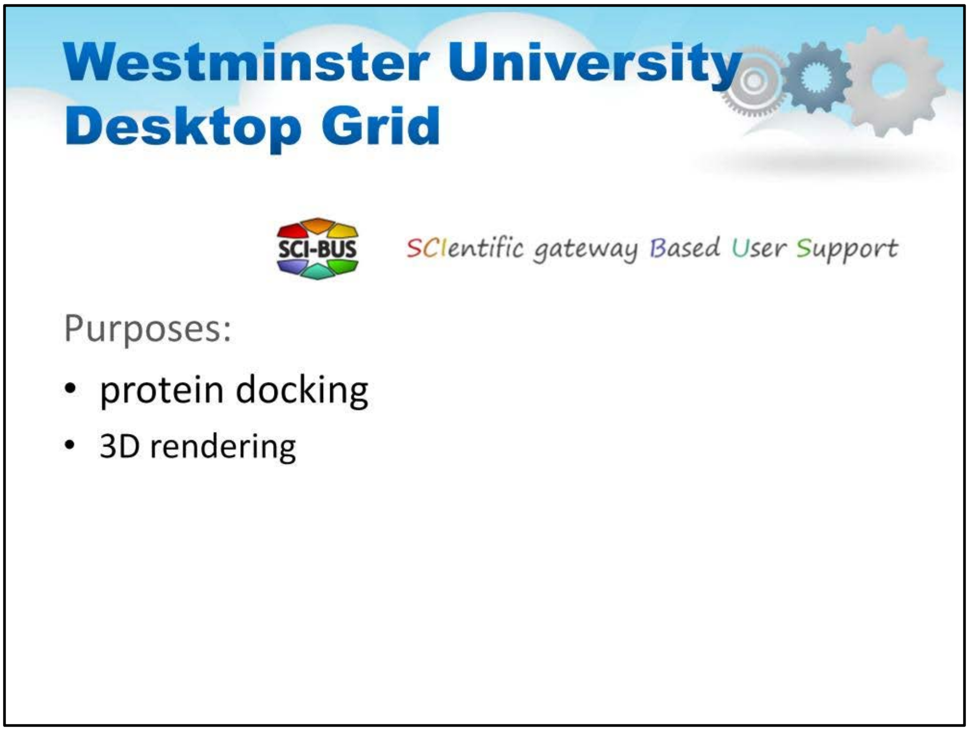
SZTAKI Desktop Grid

Purposes:

- How to easily **set up and maintain** your own desktop grid
- How to easily **develop applications** to be run on the desktop grid

MTA SZTAKI
SZTAKI DESKTOP GRID
BOINC

SZTAKI Desktop Grid (SzDG) is a BOINC project located in Hungary run by the Computer and Automation Research Institute (SZTAKI) of the Hungarian Academy of Sciences.



Westminster University Desktop Grid

SCI-BUS SCIentific gateway Based User Support

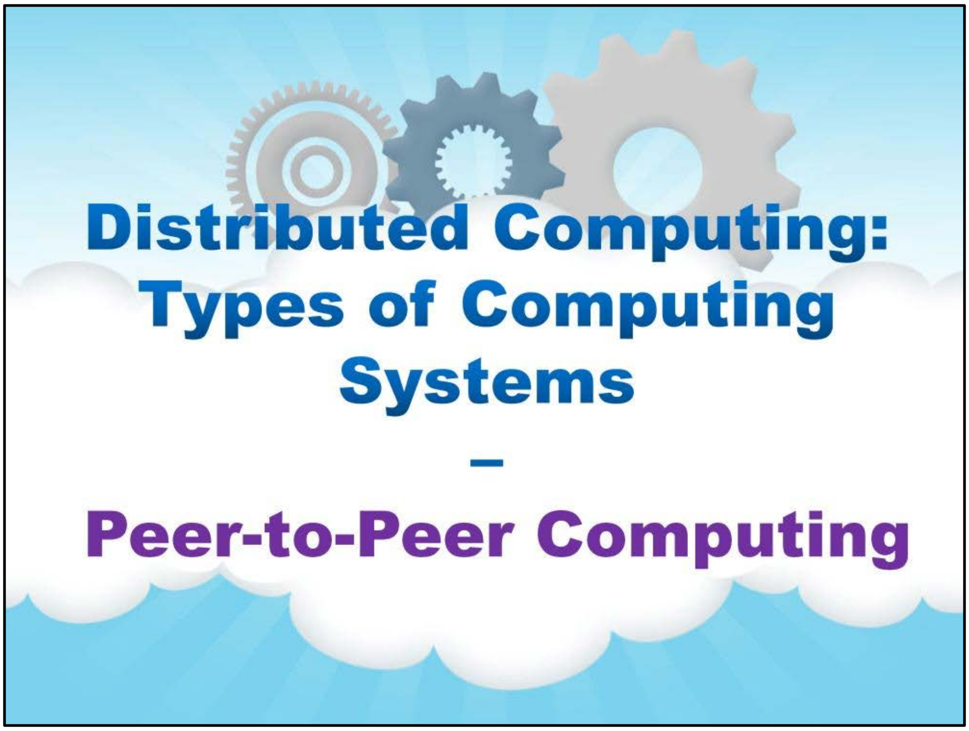
Purposes:

- protein docking
- 3D rendering

The University of Westminster Local Desktop Grid connects laboratory PCs of the university into a BOINC based Desktop Grid infrastructure. The university is set over four main campuses and some additional smaller locations in Central- and North-West-London each of them offering a variable number of mainly windows based PCs for teaching purposes.

The University of Westminster Local Desktop Grid currently includes over 1500 registered machines. These machines are available for desktop grid computations whenever they are switched on but not utilised by students for teaching or other purposes.

The aim of the University of Westminster Desktop Grid Gateway (UoW DG Gateway) is to support computation intensive research and teaching applications at the University of Westminster, based on low cost local resources.



Distributed Computing: Types of Computing Systems

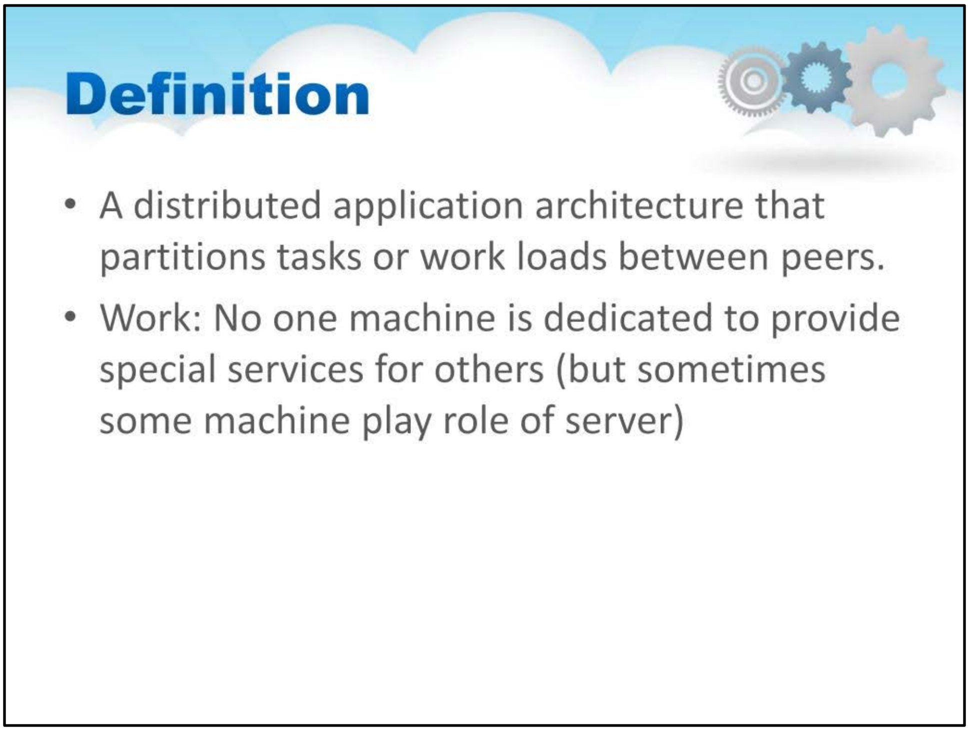
–

Peer-to-Peer Computing

Peer-to-Peer Computing

The P2P architecture offers a distributed model of networked systems.

In this section, P2P systems are introduced at the physical level and overlay networks at the logical level.

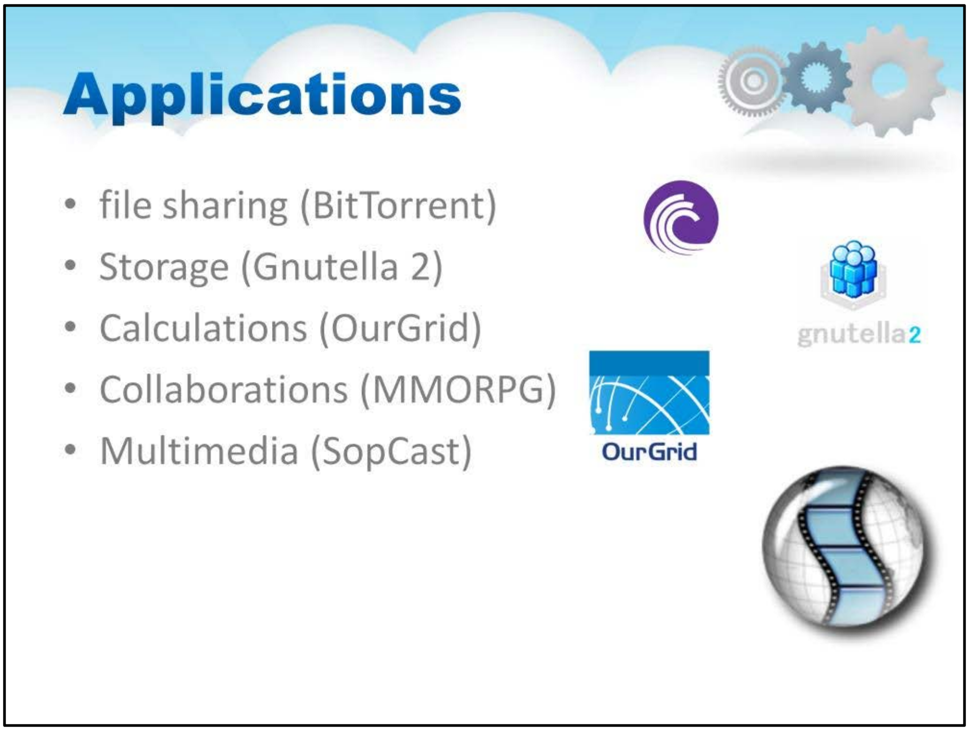


Definition

- A distributed application architecture that partitions tasks or work loads between peers.
- Work: No one machine is dedicated to provide special services for others (but sometimes some machine play role of server)

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts. Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model in which the consumption and supply of resources is divided. Emerging collaborative P2P systems are going beyond the era of peers doing similar things while sharing resources, and are looking for diverse peers that can bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond those that can be accomplished by individual peers, yet that are beneficial to all the peers.



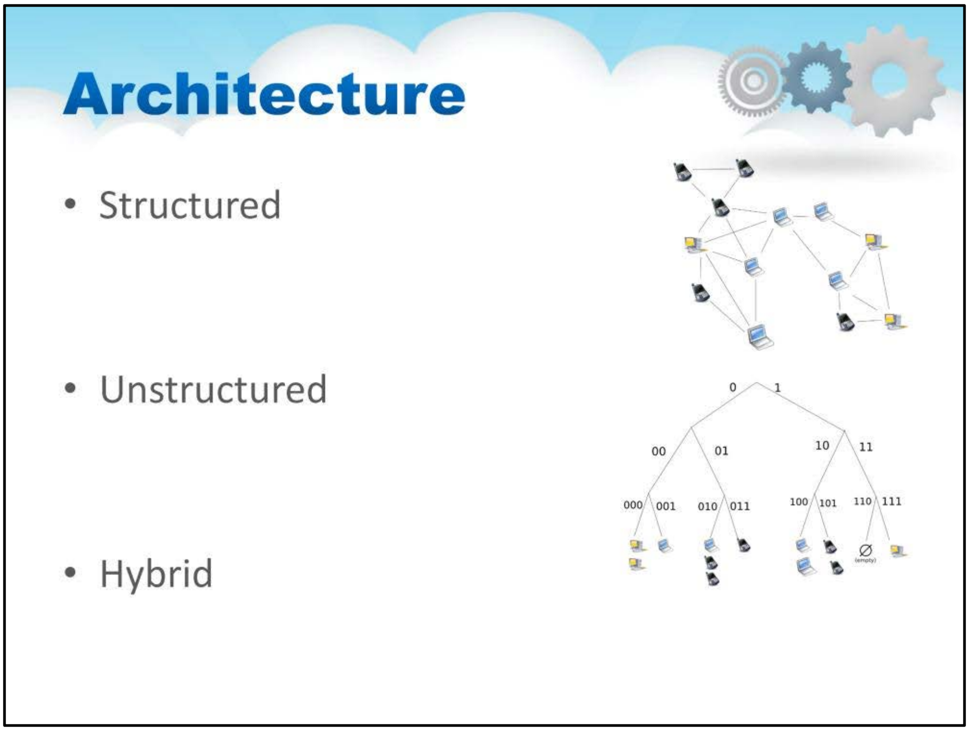
Applications

- file sharing (BitTorrent)
- Storage (Gnutella 2)
- Calculations (OurGrid)
- Collaborations (MMORPG)
- Multimedia (SopCast)

gnutella2
OurGrid

Applications of peer-to-peer computing are very wide:

- File sharing (BitTorrent)
- Storage (Gnutella 2)
- Calculations (OurGrid)
- Collaborations (MMORPG)
- Multimedia (SopCast)



Architecture

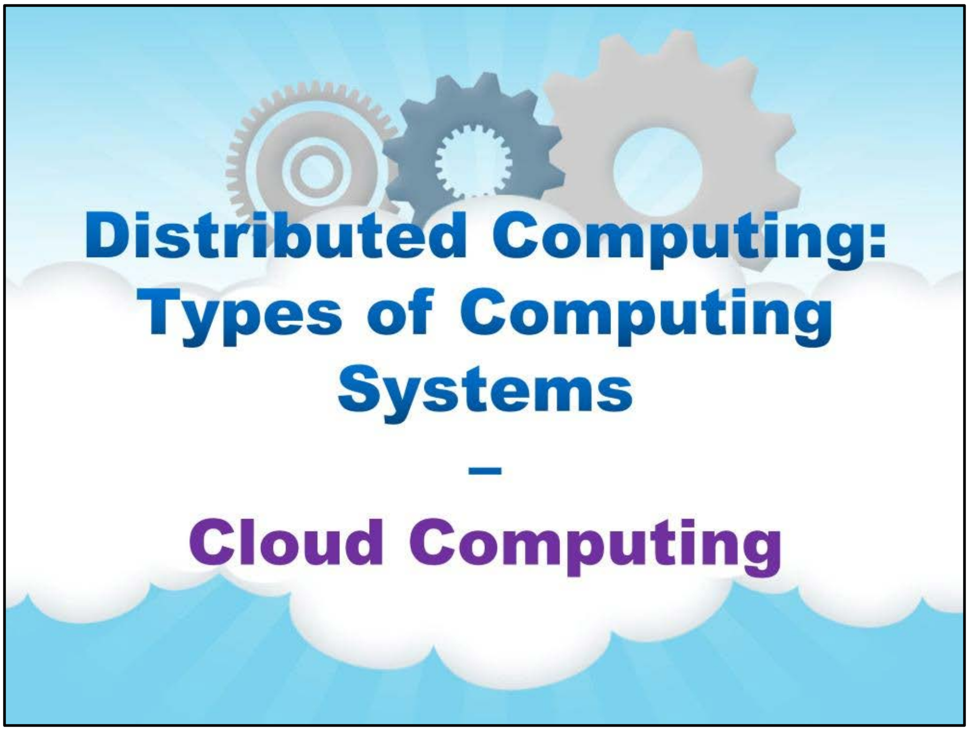
- Structured
- Unstructured
- Hybrid

A peer-to-peer network is designed around the notion of equal peer nodes simultaneously functioning as both "clients" and "servers" to the other nodes on the network.

Unstructured peer-to-peer networks do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other.

In structured peer-to-peer networks the overlay is organized into a specific topology, and the protocol ensures that any node can efficiently search the network for a file/resource, even if the resource is extremely rare.

Hybrid models are a combination of peer-to-peer and client-server models. A common hybrid model is to have a central server that helps peers find each other.

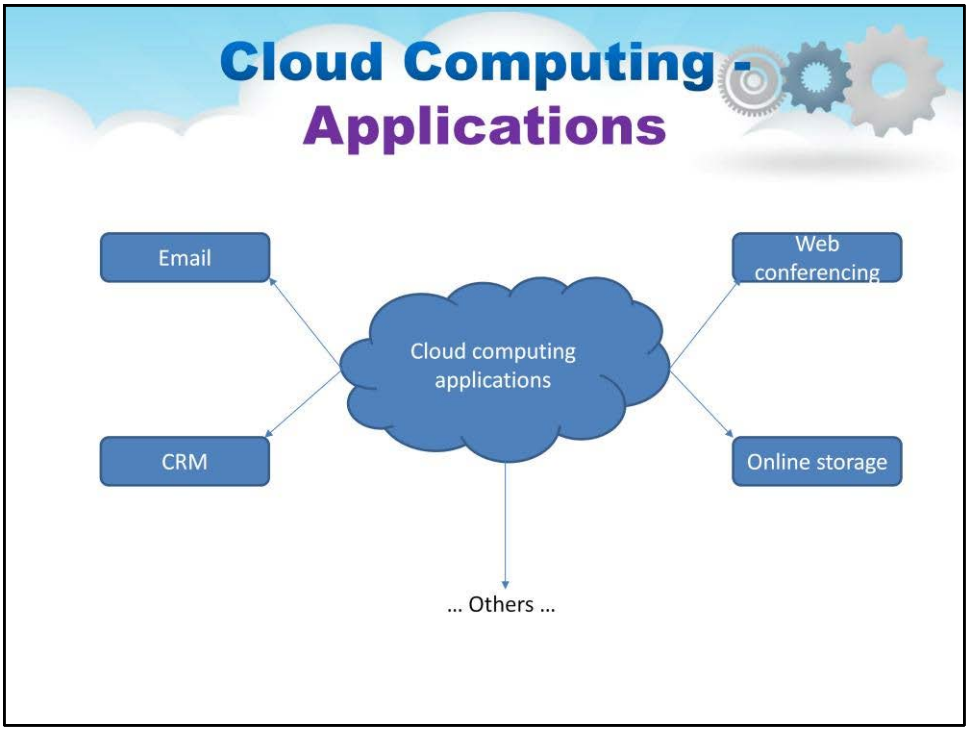


Distributed Computing: Types of Computing Systems

–

Cloud Computing

Cloud Computing



Cloud Computing - Applications

Cloud computing applications

Email

Web conferencing

CRM

Online storage

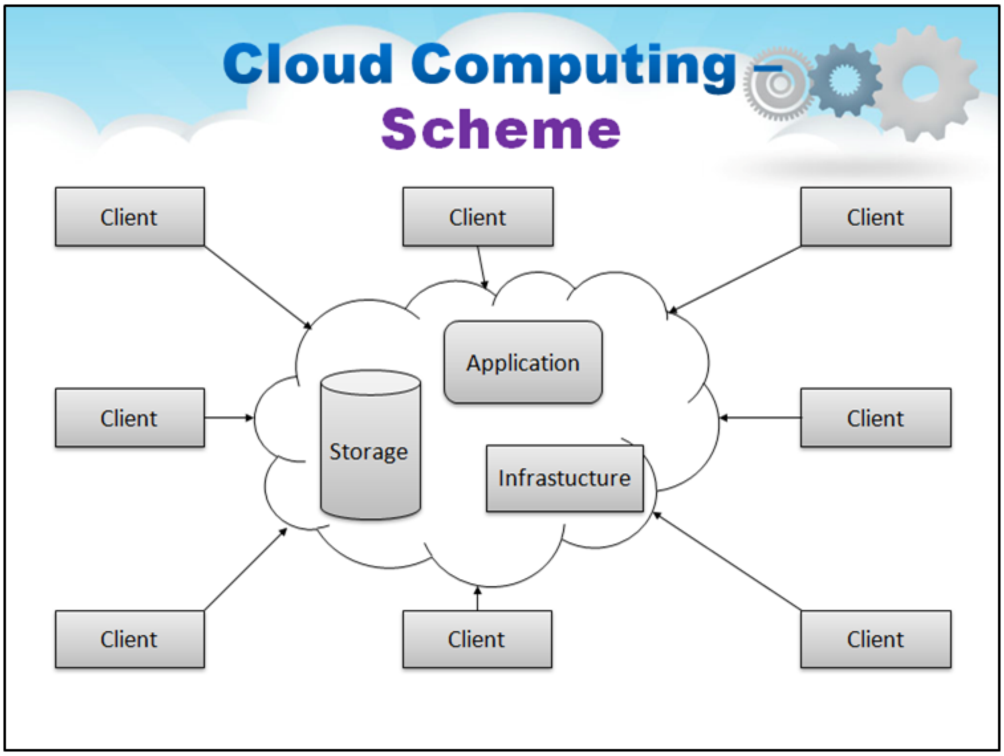
... Others ...

What is Cloud Computing?

This notion will be explained in details below in all modules of this course.

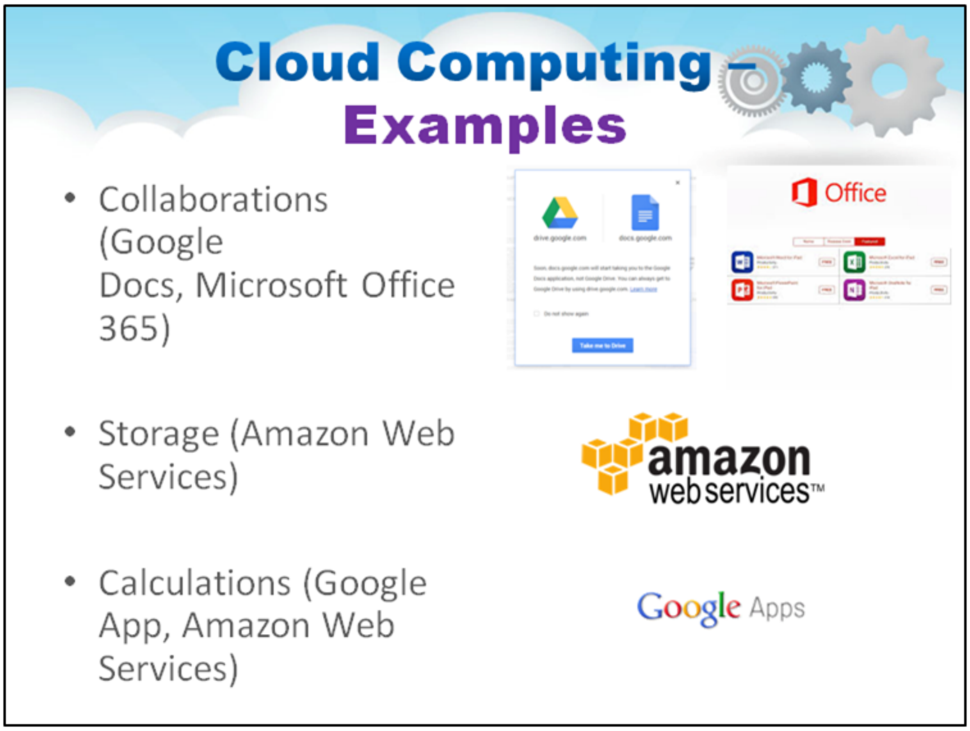
At this stage we will consider the following "jargon" definition:

The delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network (typically the Internet).



Cloud Computing – Scheme

We need not to install a piece of software on our local PC and this is how, the cloud computing overcomes platform dependency issues. Hence, the Cloud Computing is making business application mobile and collaborative.



Cloud Computing – Examples

- Collaborations (Google Docs, Microsoft Office 365)
- Storage (Amazon Web Services)
- Calculations (Google App, Amazon Web Services)

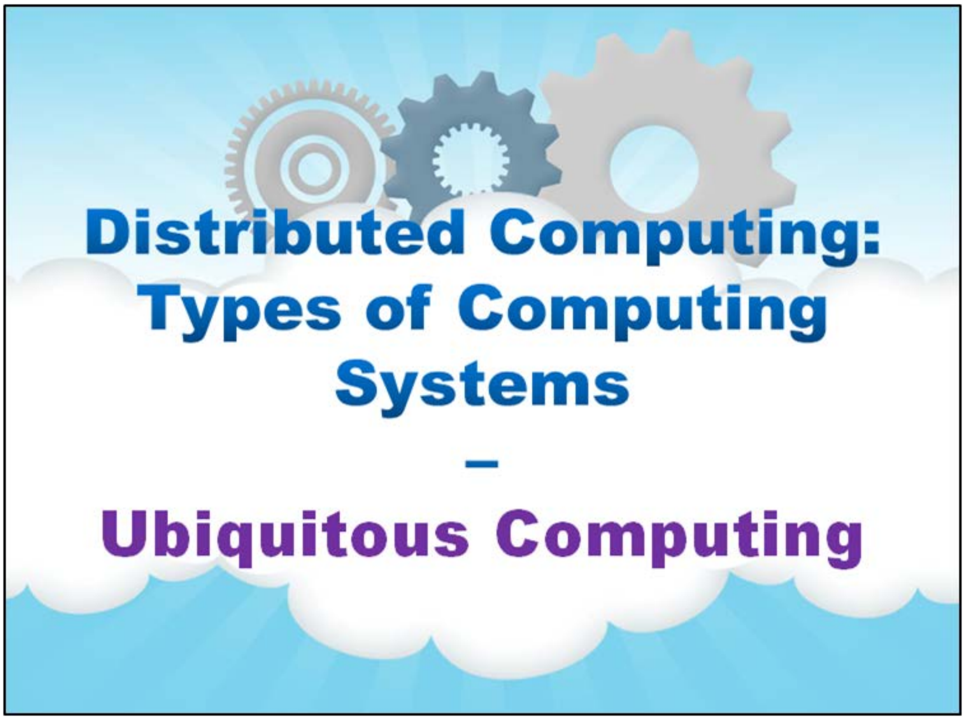
Soon, docs.google.com will start taking you to the Google Docs application, not Google Drive. You can always get to Google Drive by using drive.google.com. Learn more

□ Do not show again

drive.google.com
docs.google.com
Take me to Drive
Office
amazon web services™
Google Apps

Cloud Computing is used in numerous applications like these:

- Collaborations (Google Docs, Microsoft Office 365)
- Storage (Amazon Web Services)
- Calculations (Google App, Amazon Web Services)

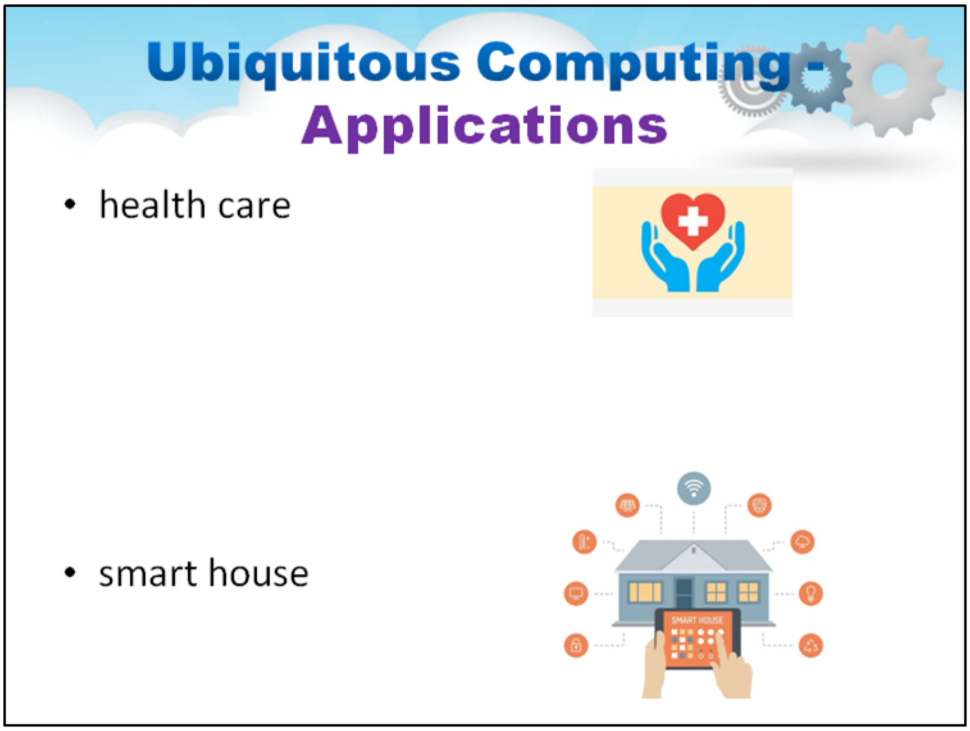


Distributed Computing: Types of Computing Systems

–

Ubiquitous Computing

Ubiquitous Computing



Ubiquitous Computing - Applications

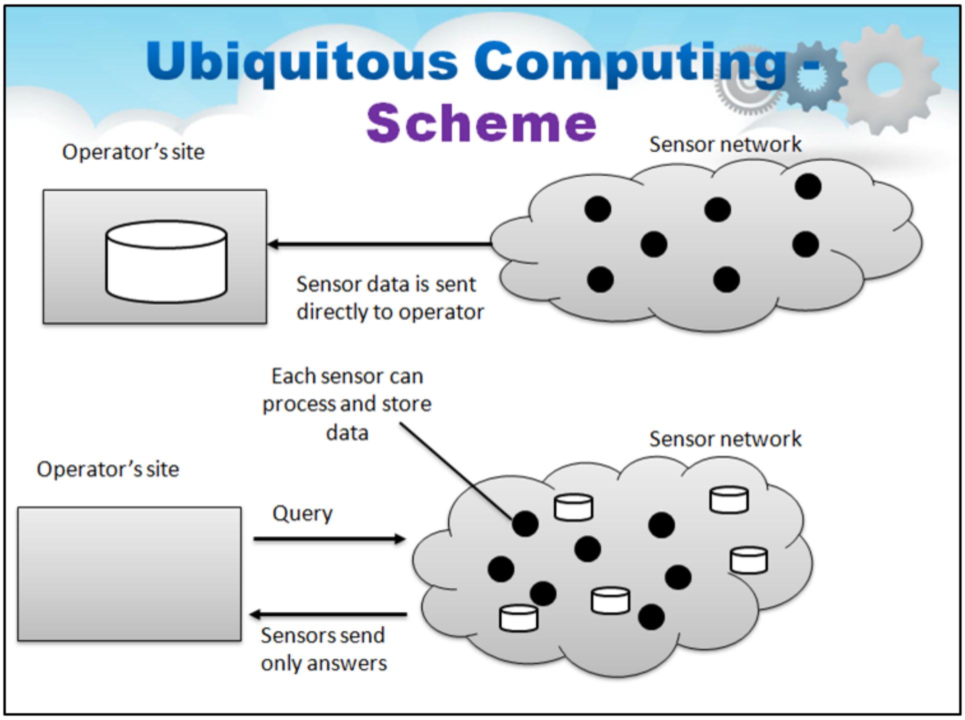
- health care
- smart house

SMART HOUSE

What is Ubiquitous Computing?

In contrast to desktop computing, Ubiquitous Computing can occur everywhere and anywhere, using any device, in any location, and in any format, including laptop computers, tablets and terminals in everyday objects such as a fridge or a pair of glasses

It has various usages, but mostly in applications for Internet of Things and wearable computing. These topics will be covered later.



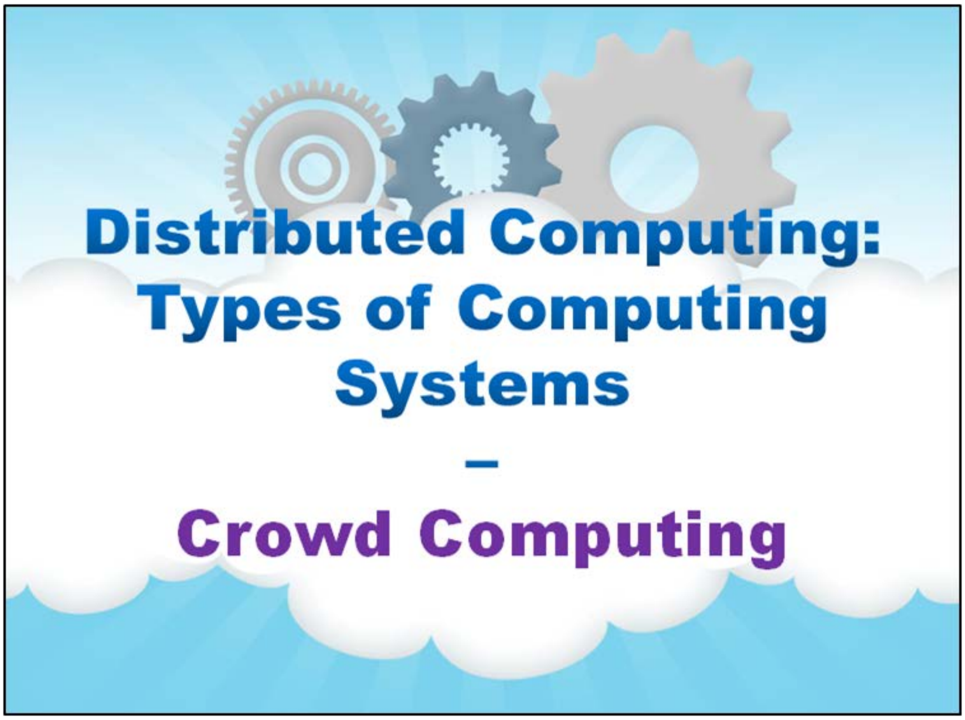
Ubiquitous Computing - Scheme
Operator's site
Sensor network
Sensor data is sent directly to operator
Each sensor can process and store data
Operator's site
Sensor network
Query
Sensors send only answers

Previous (old) paradigm of ubiquitous computing (top part of the Figure):

Organizing a sensor network database, while storing and processing data only at the operator's site.

Cloud-Fog-Dew (new) paradigm of ubiquitous computing (top part of the Figure):

Organizing a sensor network database (Cloud), while storing and processing data only at the controllers near sensors (Fog) or sensors (Dew)

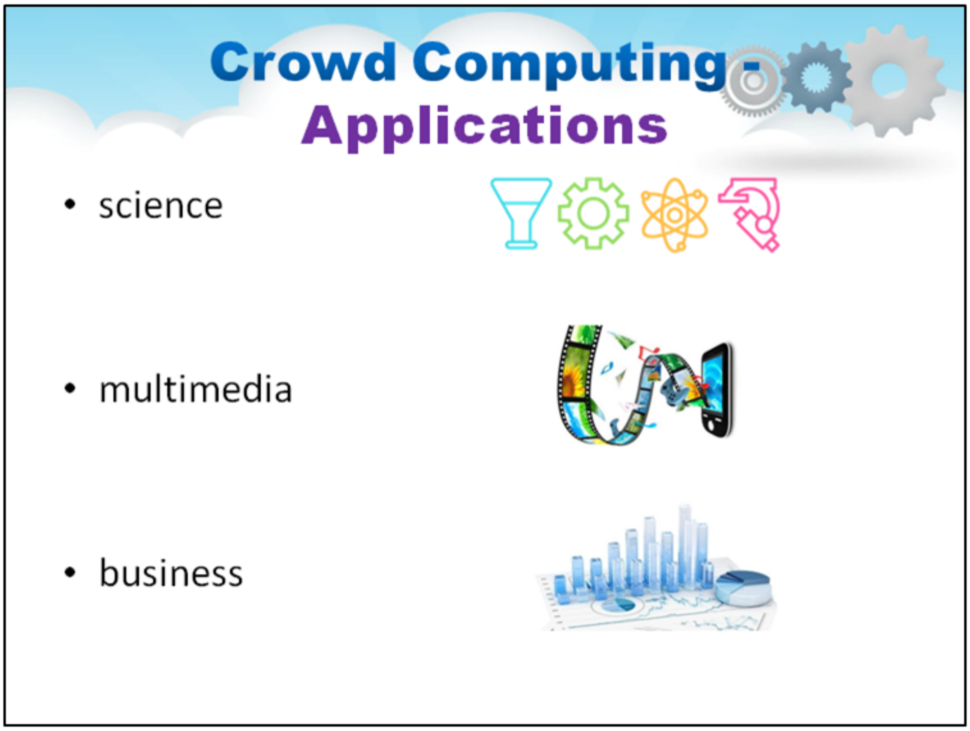


Distributed Computing: Types of Computing Systems

–

Crowd Computing

Crowd Computing

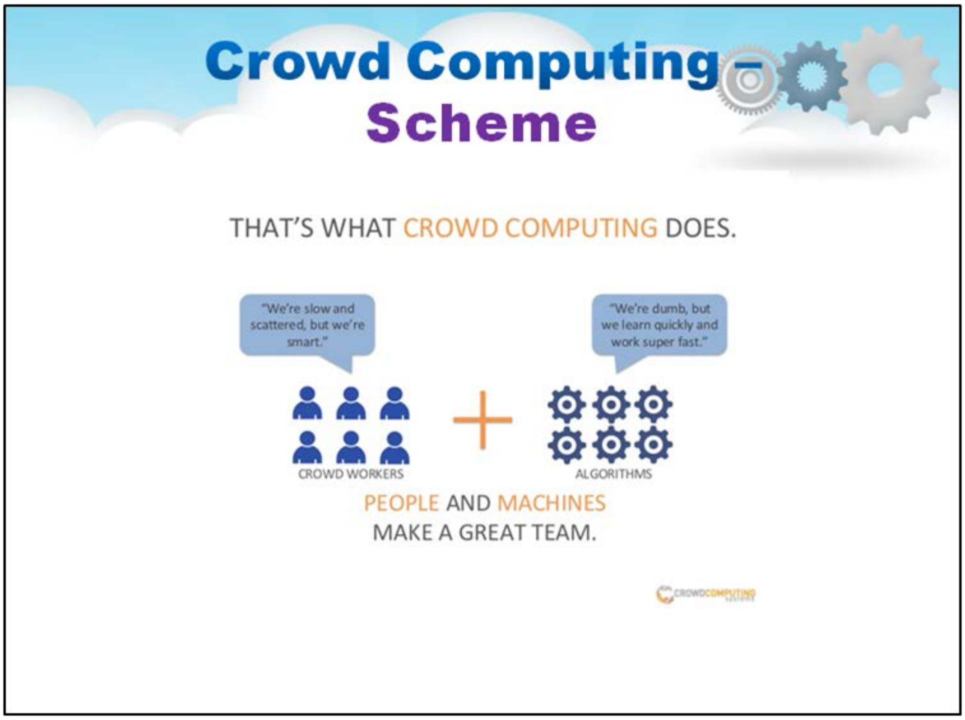


Crowd Computing - Applications

- science
- multimedia
- business

What is Crowd Computing (crowdsourcing)?

“Harnessing the power of people out in the web to do tasks that are hard for individual users or computers to do alone. Like cloud computing, crowd computing offers elastic, on-demand human resources that can drive new applications and new ways of thinking about technology” (C) Rob Miller



Crowd Computing – Scheme

THAT'S WHAT CROWD COMPUTING DOES.

"We're slow and scattered, but we're smart."
"We're dumb, but we learn quickly and work super fast."
CROWD WORKERS
ALGORITHMS

PEOPLE AND MACHINES MAKE A GREAT TEAM.

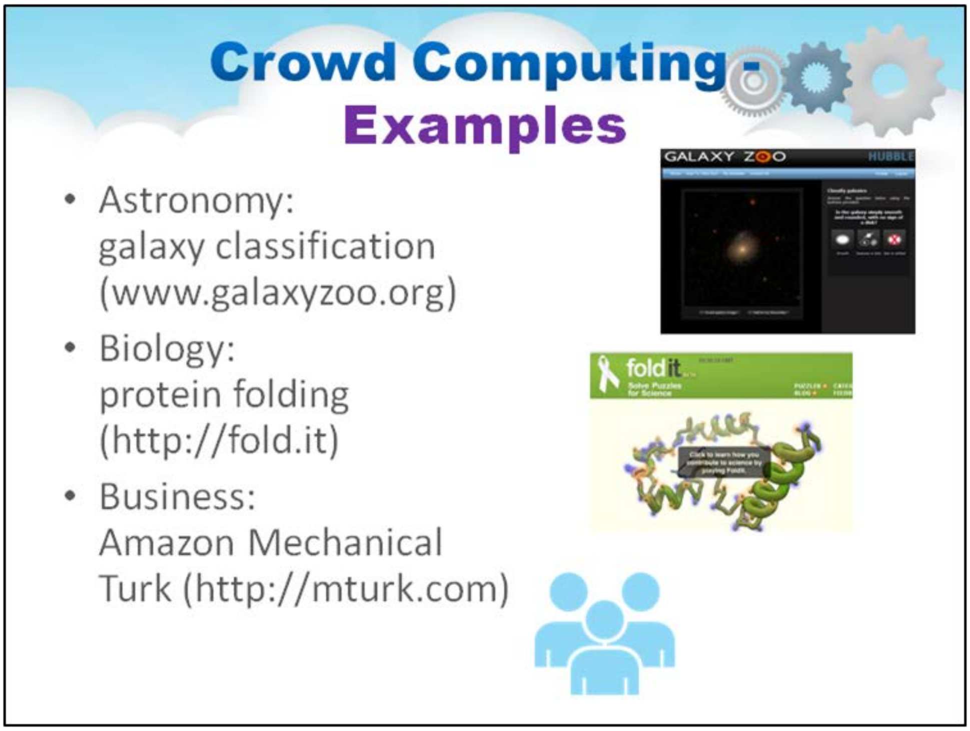
It is actually like

volunteer PC computing,

but crowd computing uses

volunteer brain computing

and combines human intelligence (the crowd) with artificial intelligence (the cloud).



Crowd Computing - Examples

- Astronomy:
  galaxy classification
  (www.galaxyzoo.org)
- Biology:
  protein folding
  (http://fold.it)
- Business:
  Amazon Mechanical
  Turk (http://mturk.com)

GALAXY ZOO
HUBBLE
fold it
Solve Puzzles for Science
Click to learn how you contribute to science by playing Foldit.

Crowd Computing examples are everywhere:

Astronomy:
galaxy classification (www.galaxyzoo.org)

Biology:
protein folding (http://fold.it)

Business:
Amazon Mechanical Turk (http://mturk.com)

Cloud Computing

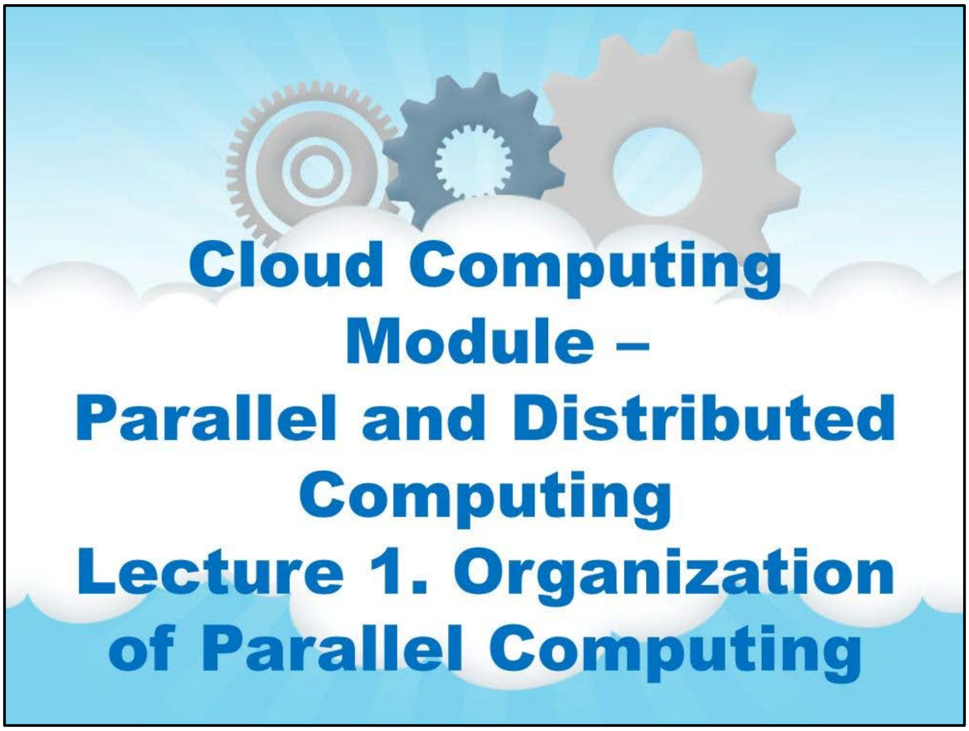
Lecture Manual

Volume 1

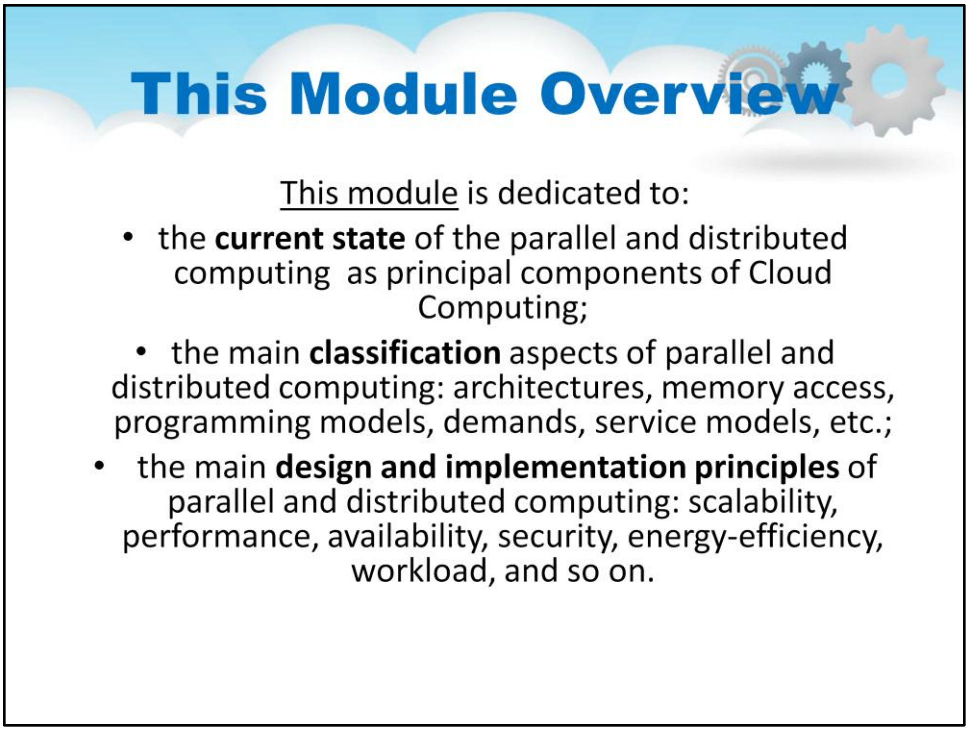
Module 1

Parallel and Distributed Computing

Content



Cloud Computing Module – Parallel and Distributed Computing Lecture 1. Organization of Parallel Computing

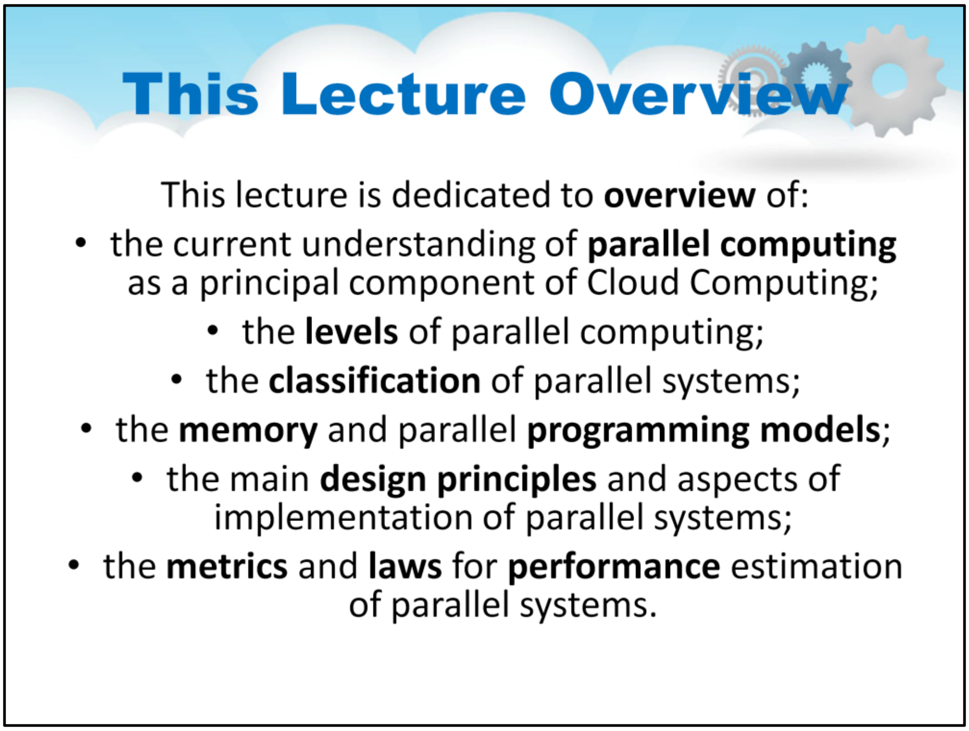


This Module Overview

This module is dedicated to:

- the **current state** of the parallel and distributed computing as principal components of Cloud Computing;
- the main **classification** aspects of parallel and distributed computing: architectures, memory access, programming models, demands, service models, etc.;
- the main **design and implementation principles** of parallel and distributed computing: scalability, performance, availability, security, energy-efficiency, workload, and so on.

Module 1. Parallel and Distributed Computing



This Lecture Overview

This lecture is dedicated to **overview** of:

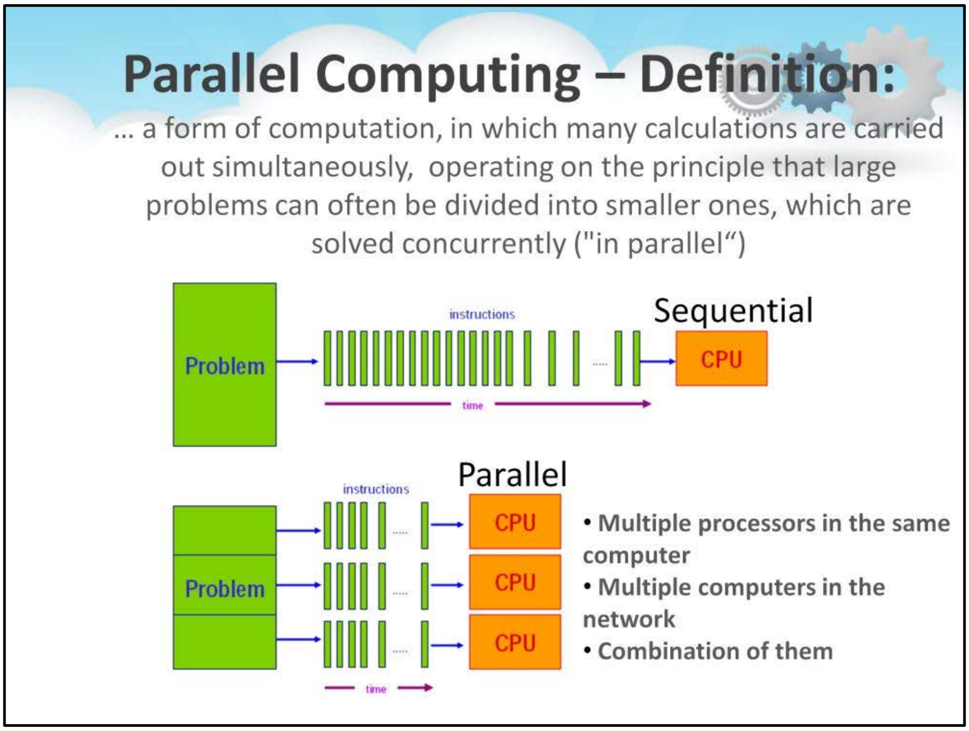
- the current understanding of **parallel computing** as a principal component of Cloud Computing;
- the **levels** of parallel computing;
- the **classification** of parallel systems;
- the **memory** and parallel **programming models**;
- the main **design principles** and aspects of implementation of parallel systems;
- the **metrics** and **laws** for **performance** estimation of parallel systems.

Lecture 1. Organization of Parallel Computing

Parallel Computing - Overview

Parallel Computing

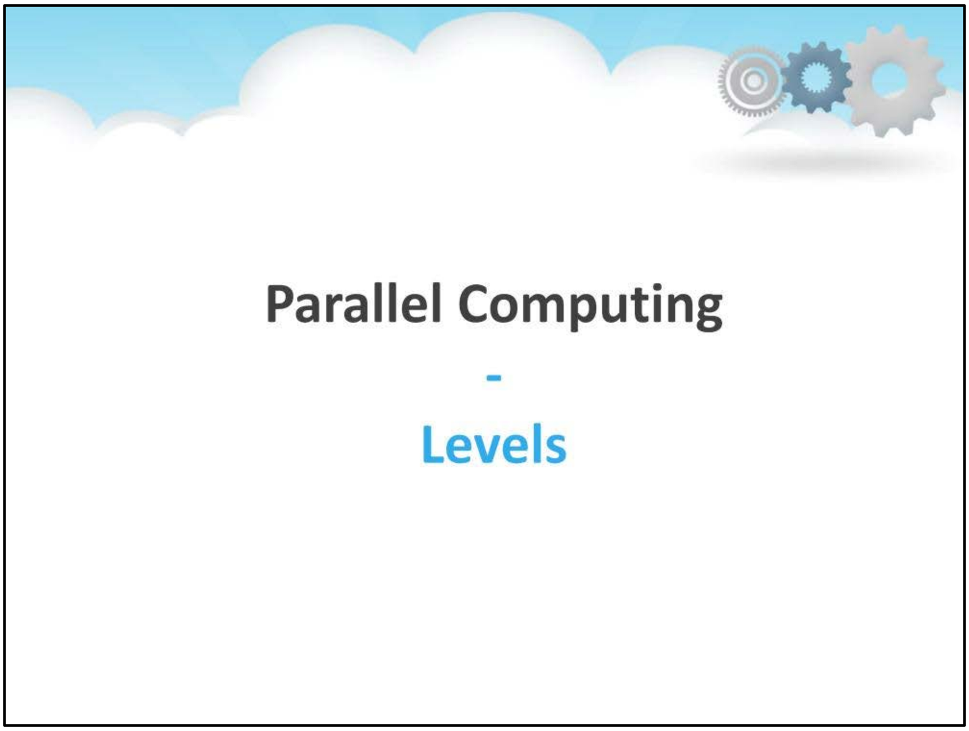
Overview



Parallel Computing – Definition:

… a form of computation, in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are solved concurrently ("in parallel")

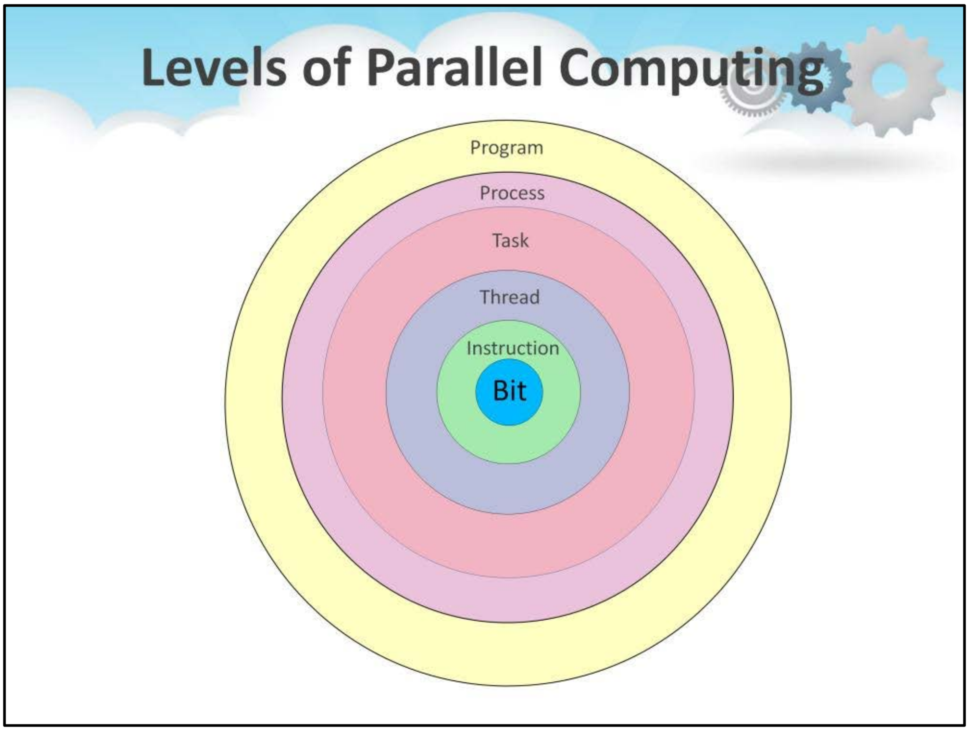
- Multiple processors in the same computer
- Multiple computers in the network
- Combination of them



Parallel Computing

-

Levels



Levels of Parallel Computing

Bit-level: from bit processing to word processing.

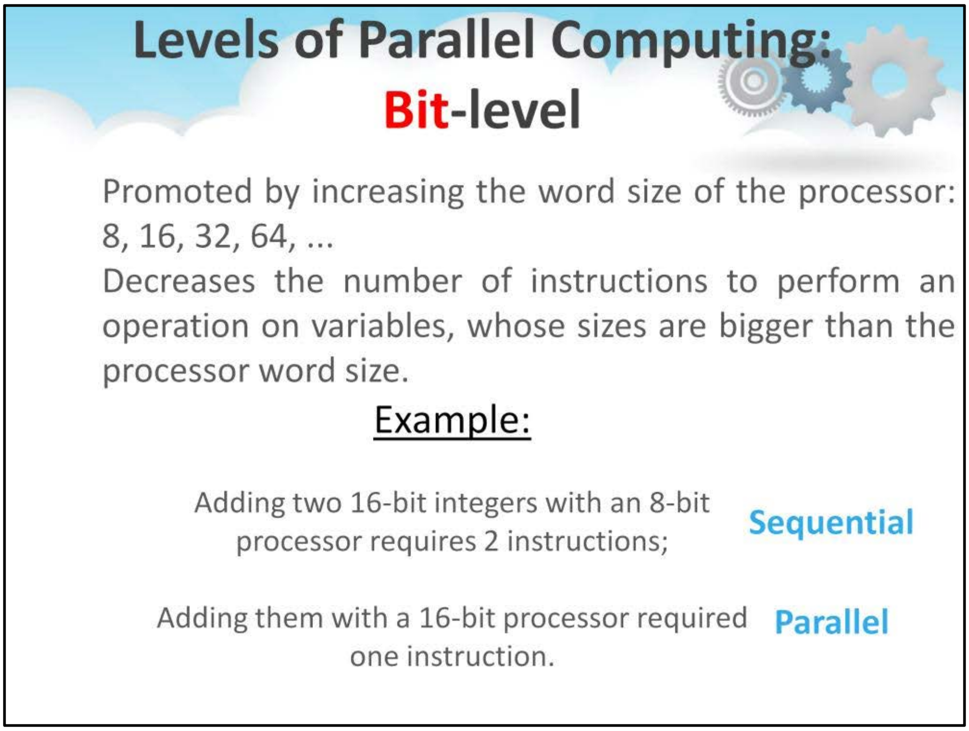
Instruction-level: a single operation of a processor.

Thread-level: stream of execution (has one or multiple instructions).

Task-level: execution path through the address space that has many instructions. (Some times task and process are used interchangeably).

Process-level: instance of a program in execution. It has its own address space, and interacts with other processes only through communication managed by OS. A process might contain one or multiple threads, which share the same address space and interacting directly.

Program-level: an executable file with one or multiple tasks.



Levels of Parallel Computing: Bit-level

Promoted by increasing the word size of the processor: 8, 16, 32, 64, ...

Decreases the number of instructions to perform an operation on variables, whose sizes are bigger than the processor word size.

Example:

Adding two 16-bit integers with an 8-bit processor requires 2 instructions; Sequential

Adding them with a 16-bit processor required one instruction. Parallel

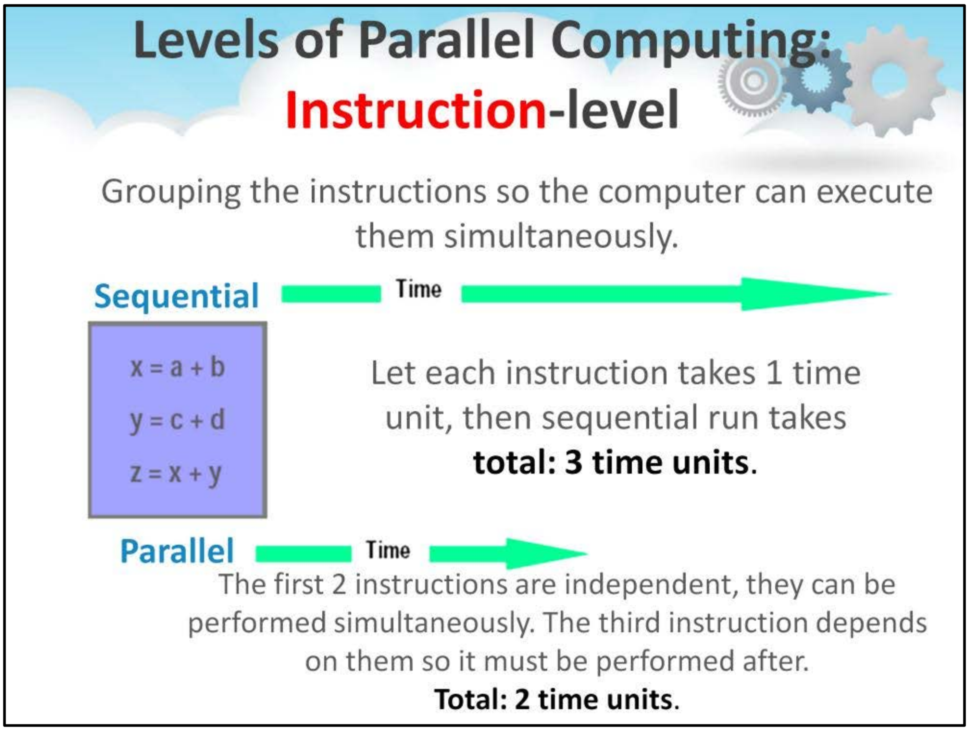
More than sixty years ago, when hardware was bulky and expensive, most computers were designed in a bit-serial fashion.

In this scenario, **bit-level parallelism (BLP)** converts bit-serial processing to word-level processing gradually.

Over the years, users graduated from 4-bit microprocessors to 8-, 16-, 32-, and 64-bit CPUs.

Bit-level parallelism (BLP) led us to the next wave of improvement, known as **instruction-level parallelism (ILP)**.

Let’s consider **instruction-level parallelism (ILP)** in details.



Levels of Parallel Computing:
Instruction-level

Grouping the instructions so the computer can execute them simultaneously.

Sequential

x = a + b
y = c + d
z = x + y

Let each instruction takes 1 time unit, then sequential run takes **total: 3 time units.**

Parallel

The first 2 instructions are independent, they can be performed simultaneously. The third instruction depends on them so it must be performed after.

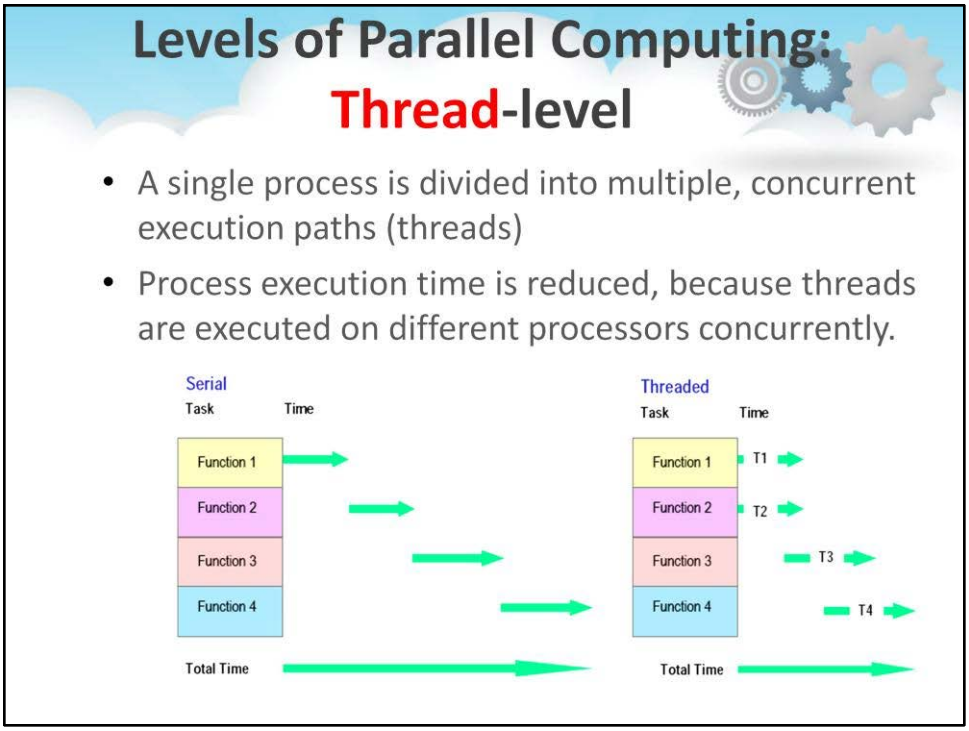
Total: 2 time units.

In **instruction-level parallelism (ILP)**
the processor executes multiple instructions simultaneously rather than only one instruction at a time.

For the past more than 40 years,
ILP evolved significantly with appearance of various techniques like:

- branch prediction,
- dynamic scheduling,
- speculation,
- pipelining,
- superscalar computing,
- VLIW (very long instruction word) architectures, and
- multithreading.

Let’s consider **multithreading** in details.



Levels of Parallel Computing: Thread-level

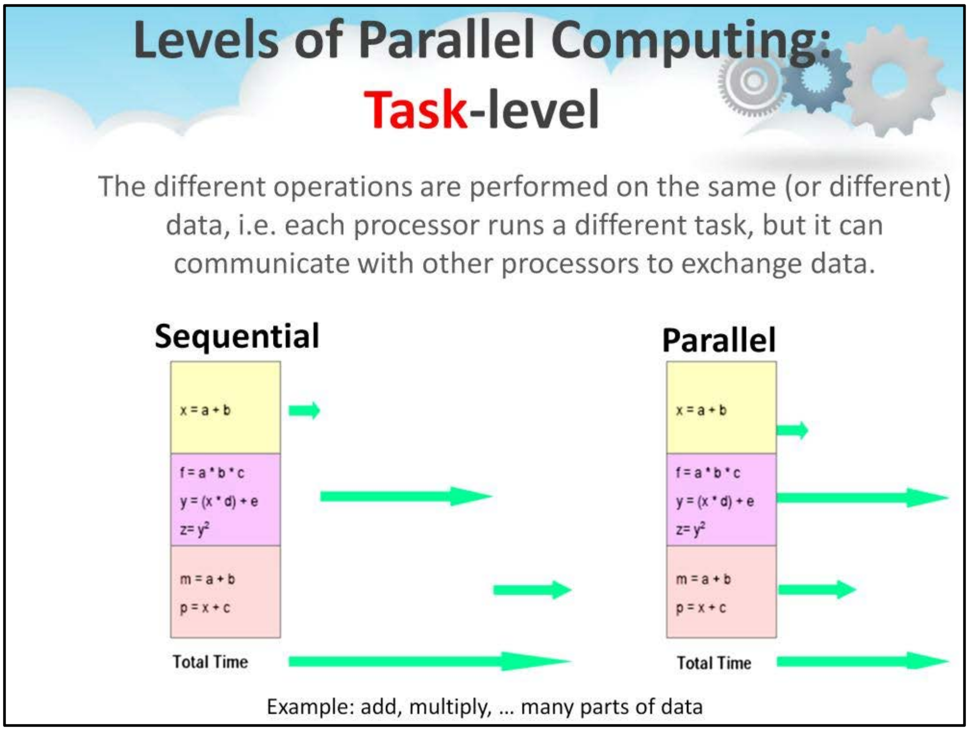
- A single process is divided into multiple, concurrent execution paths (threads)
- Process execution time is reduced, because threads are executed on different processors concurrently.

Multithreading or **thread-level parallelism (TLP)** –
is sharing the functional units of one processor
by multiple processes or threads taking part in overlapped execution.
The purpose can be to execute:

- several programs on one processor or
- to execute a single application as a multithreaded program (real parallel program).

Processes can belong to different users (applications), but threads belong to the same user (application).
It is a higher level of parallelism than instruction level parallelism (ILP), because the execution of each single thread can exploit ILP.

<u>Advantage</u> <u>(in comparison to process level parallelization):</u>
A switch between processes, normally denoted context switch in operating systems terminology, can typically use hundreds or even thousands of clock cycles, while there is multithreaded processors that can switch to another thread within one clock cycle.



Levels of Parallel Computing: Task-level

The different operations are performed on the same (or different) data, i.e. each processor runs a different task, but it can communicate with other processors to exchange data.

Sequential

x = a + b

f = a * b * c

y = (x * d) + e

$z = y^2$

m = a + b

p = x + c

Total Time

Parallel

x = a + b

f = a * b * c

y = (x * d) + e

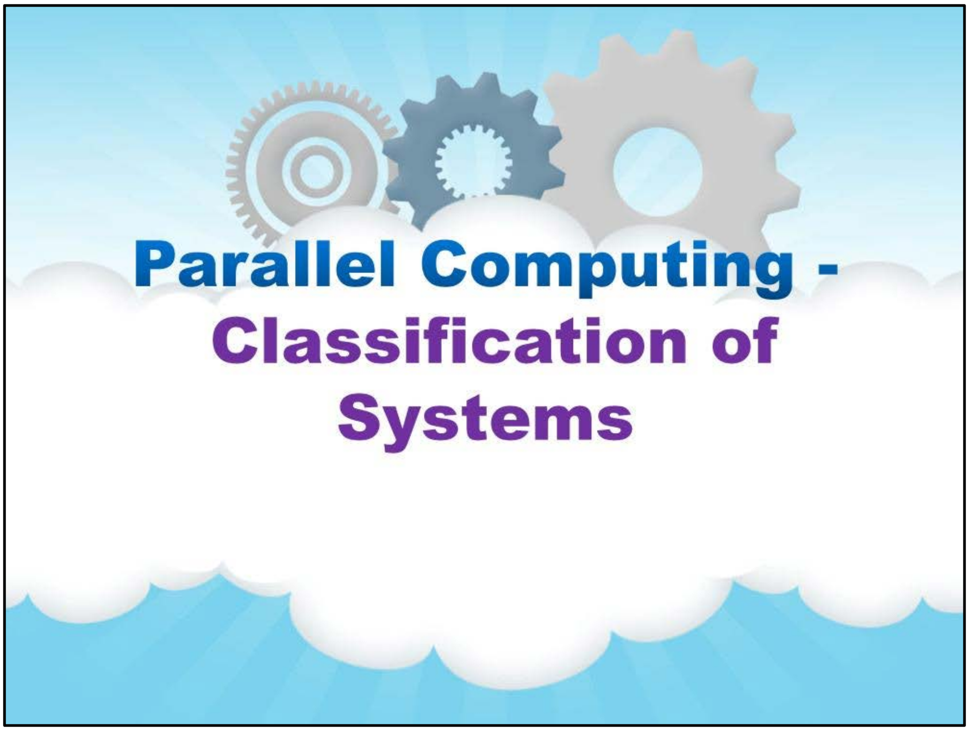
$z = y^2$

m = a + b

p = x + c

Total Time

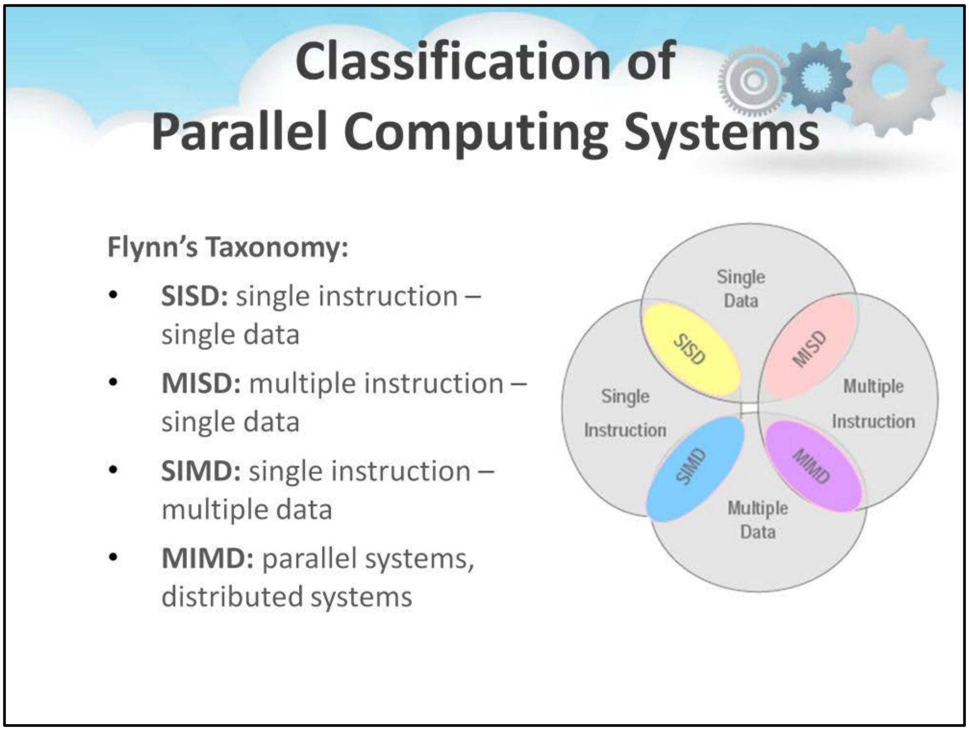
Example: add, multiply, … many parts of data



Parallel Computing - Classification of Systems

Classification of Systems

Now, let’s consider the classification of parallel systems.



Classification of Parallel Computing Systems

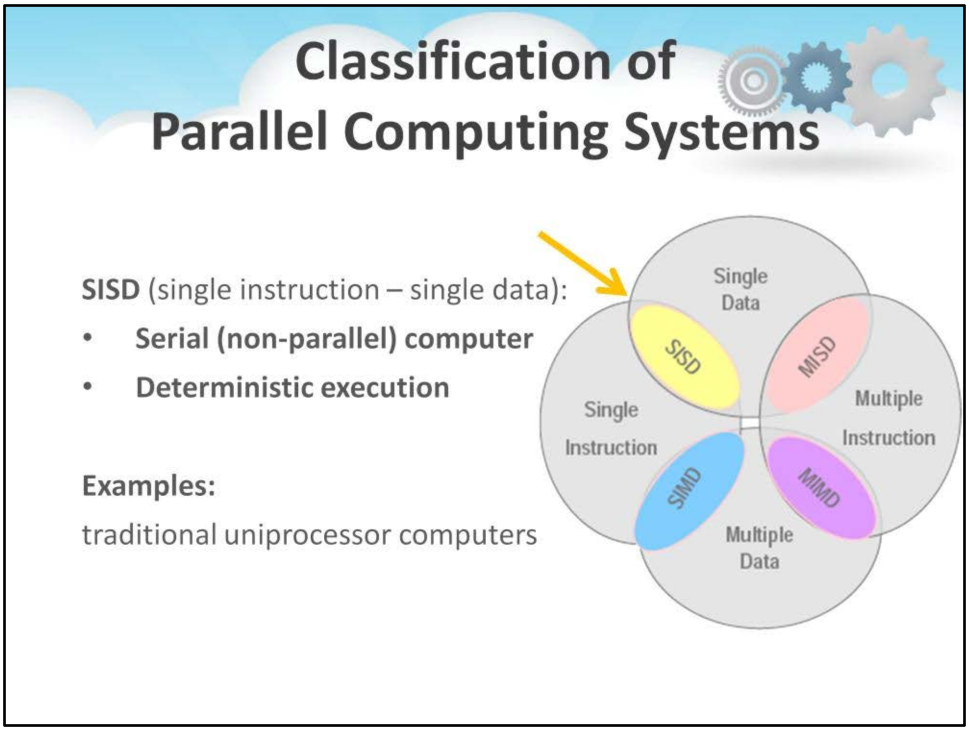
Flynn's Taxonomy:

- **SISD:** single instruction – single data
- **MISD:** multiple instruction – single data
- **SIMD:** single instruction – multiple data
- **MIMD:** parallel systems, distributed systems

Single Data
SISD
MISD
Single Instruction
Multiple Instruction
SIMD
MIMD
Multiple Data

Flynn divided multiprocessors into four categories based on the multiplicity of **instruction streams** and **data streams.**

This has become known as the famous Flynn's taxonomy.



Classification of Parallel Computing Systems

SISD (single instruction – single data):

- **Serial (non-parallel) computer**
- **Deterministic execution**

Examples:

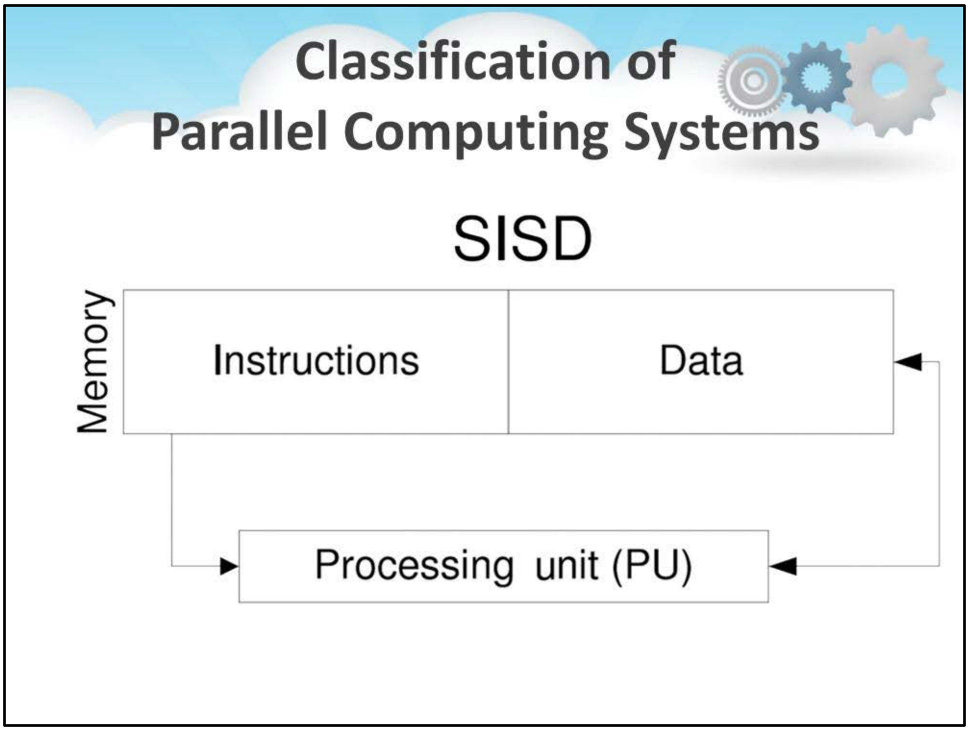
traditional uniprocessor computers

A conventional computer (uniprocessor or von Neumann machine) is termed a **Single Instruction Single Data (SISD)** machine.

Here you can see:

- its place in Flynn taxonomy
- its main characteristics
- and examples of implementation – in traditional uniprocessor computers.

Let’s consider them in details on the next slide…



Classification of Parallel Computing Systems

SISD

Memory

Instructions

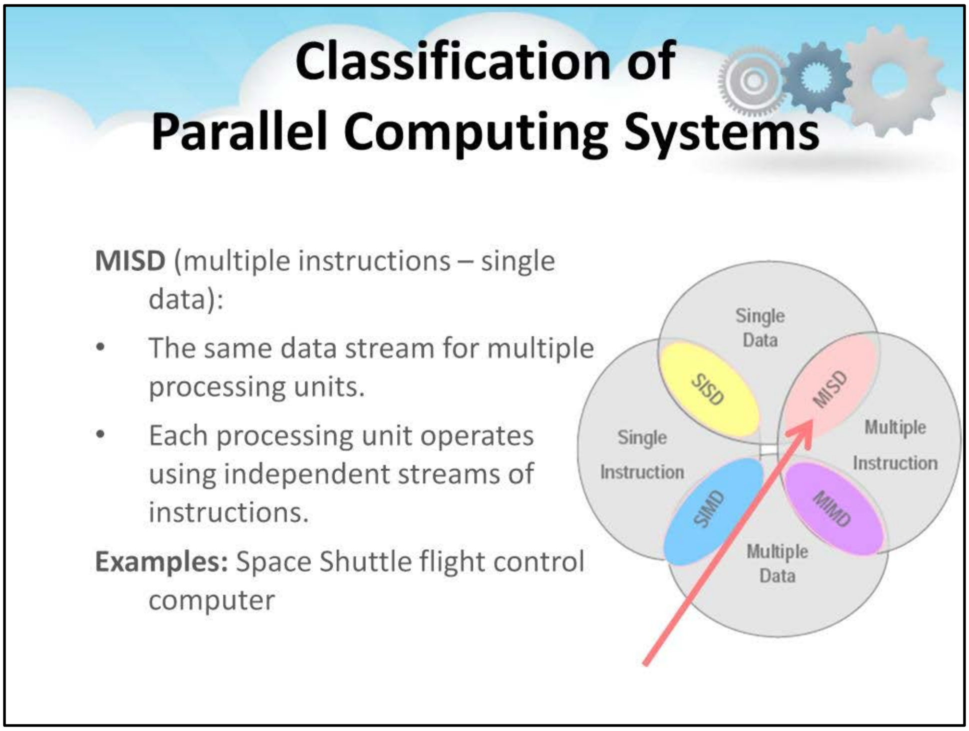
Data

Processing unit (PU)

It has one execution or processing unit (PU).

This **processing unit** is controlled by a **single sequence of instructions**,
and it operates on a **single sequence of data** in memory.

In the early days of computing, the control logic needed to decode the instructions into control signals that manage the execution and data traffic in a processor was a costly component.



Classification of Parallel Computing Systems

MISD (multiple instructions – single data):

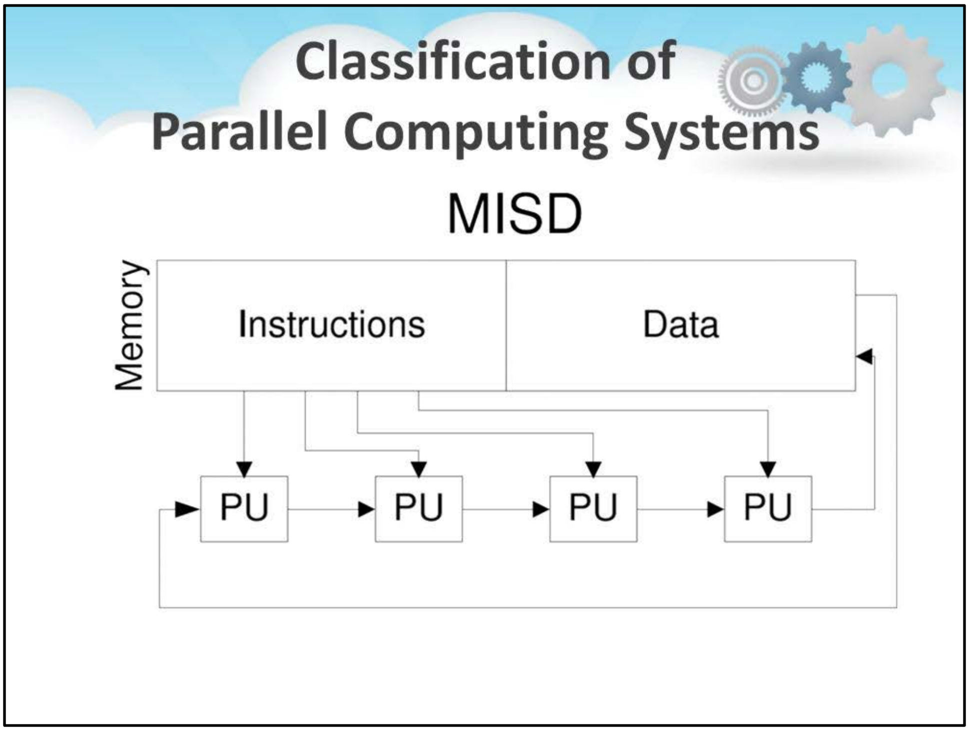
- The same data stream for multiple processing units.
- Each processing unit operates using independent streams of instructions.

Examples: Space Shuttle flight control computer

The **Multiple Instruction Single Data (MISD)** category of machines has been given a mixed treatment in the literature. Some textbooks simply say that no machines of this category have been built, while others present examples. In our view MISD is an important category representing different parallel architectures.

Here you can see:

- its place in Flynn taxonomy
- its main characteristics
- and examples of implementation – in avionics control computers, in specialized hardware structures – let’s consider them in details on the next slide…



Classification of Parallel Computing Systems

MISD

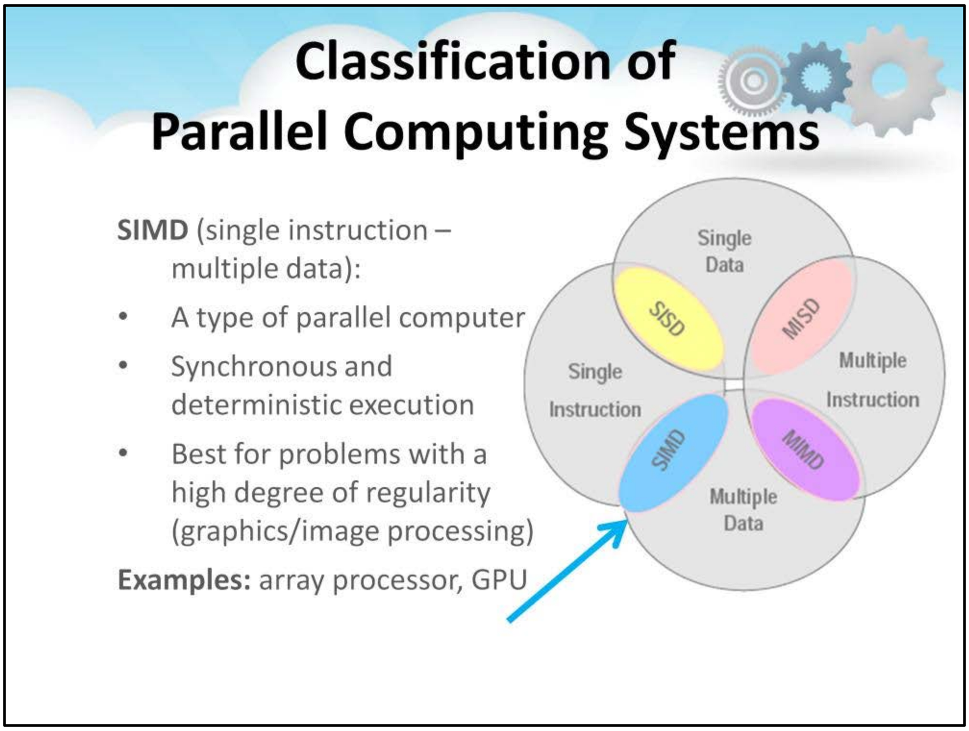
One of the example architectures presented in the classical paper by Flynn is very similar to the variant shown in this Figure.

Here a source data stream is

- sent from the memory to the first PU,
- then a derived data stream is sent to the next PU, where it is processed by another program (instruction stream)
- and so on until it is streamed back to memory.

This kind of computation has by some authors been called a **software pipeline**. It can be efficient for applications such as real-time processing of a stream of images (video) data, where data is streamed through different PUs executing different image processing functions (e.g. filtering or feature extraction).

Another type of parallel architectures that can be classified as MISD is **systolic arrays**. These are specialized hardware structures, often implemented as an **application specific integrated circuit** (ASIC), and use highly pipelined and parallel execution of specific algorithms such as pattern matching or sorting or cryptocurrency mining.



Classification of Parallel Computing Systems

SIMD (single instruction – multiple data):

- A type of parallel computer
- Synchronous and deterministic execution
- Best for problems with a high degree of regularity (graphics/image processing)

Examples: array processor, GPU

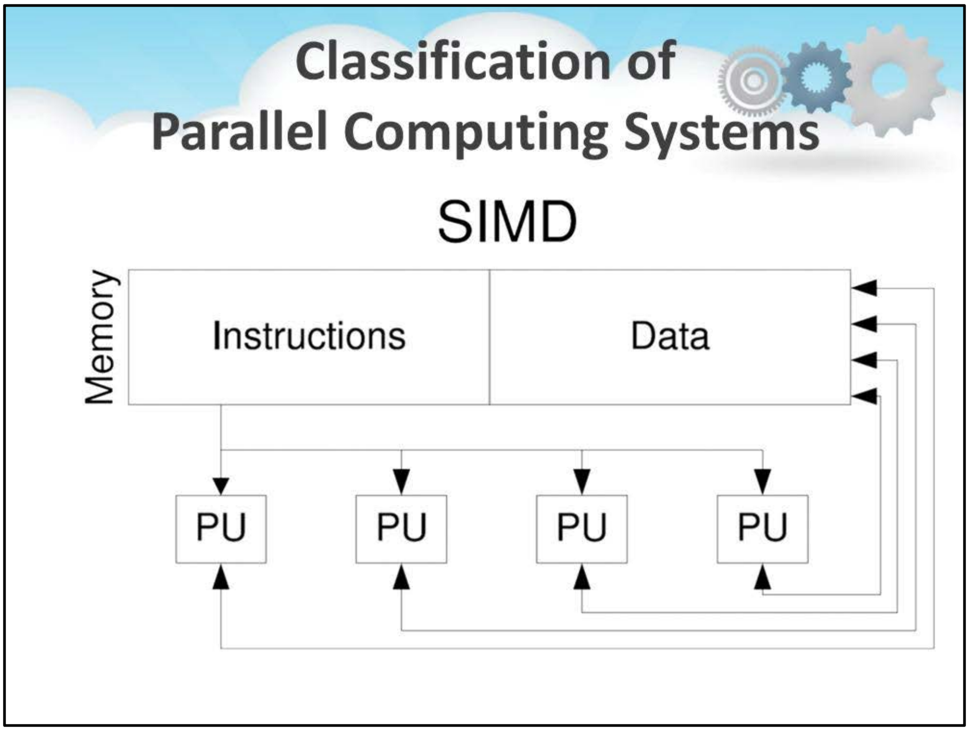
When introducing parallel processing,
it was therefore natural to let **multiple execution units** operate on different data (**multiple data** streams).
They were controlled by the same single control unit, that is, a single instruction stream.

A fundamental **limitation** of these **Single Instruction Multiple Data (SIMD)** architectures is that different PUs **cannot execute different instructions** and, at the same time, they are all **bound to one single instruction stream**.

Here you can see:

- its place in Flynn taxonomy
- its main characteristics
- and examples of implementation – in array processors, GPU, etc.

Let’s consider them in details on the next slide…



Classification of Parallel Computing Systems

SIMD

SIMD machines evolved in many variants.

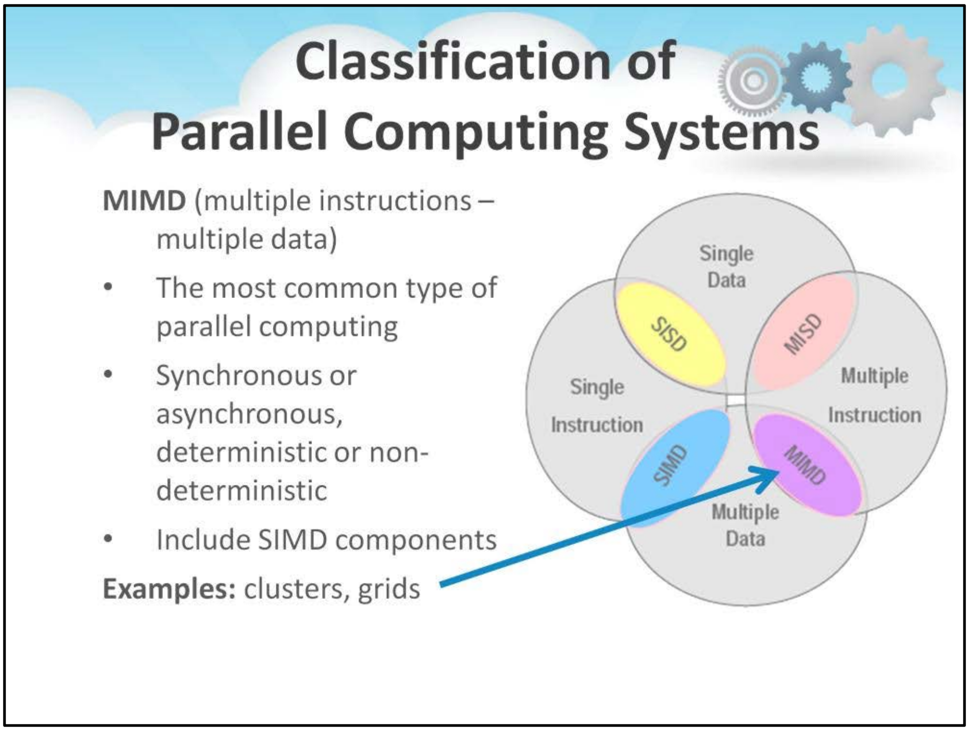
A main distinction is between

- SIMD with **shared memory** (as shown in this Figure) and
- SIMD computers with **distributed memory**.

In the latter variant (**distributed memory**),
the main memory is distributed to the different PUs.

The **advantage** of this architecture is that it is much **easier to implement** compared to multiple data streams to one shared memory.

The **disadvantage** is that it gives the need for some mechanism
such as **special instructions for communicating** between
the different PUs.



Classification of Parallel Computing Systems

MIMD (multiple instructions – multiple data)

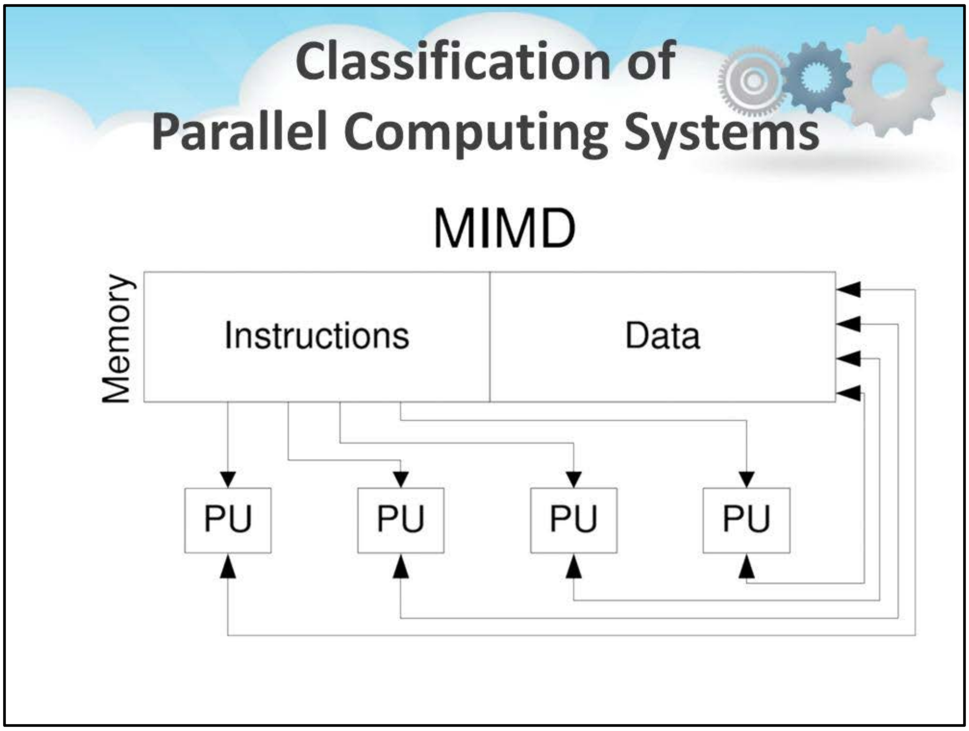
- The most common type of parallel computing
- Synchronous or asynchronous, deterministic or non-deterministic
- Include SIMD components

Examples: clusters, grids

Multiple Instruction Multiple Data (MIMD) category comprises
most contemporary parallel computer architectures,
and its inability to categorize these
has been a source for the proposal of **different alternative taxonomies**,
for example, described by Quinn in his book:

M. J. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill Book Company, New York, 1987.

But this question is out of scope of this course.



Classification of Parallel Computing Systems

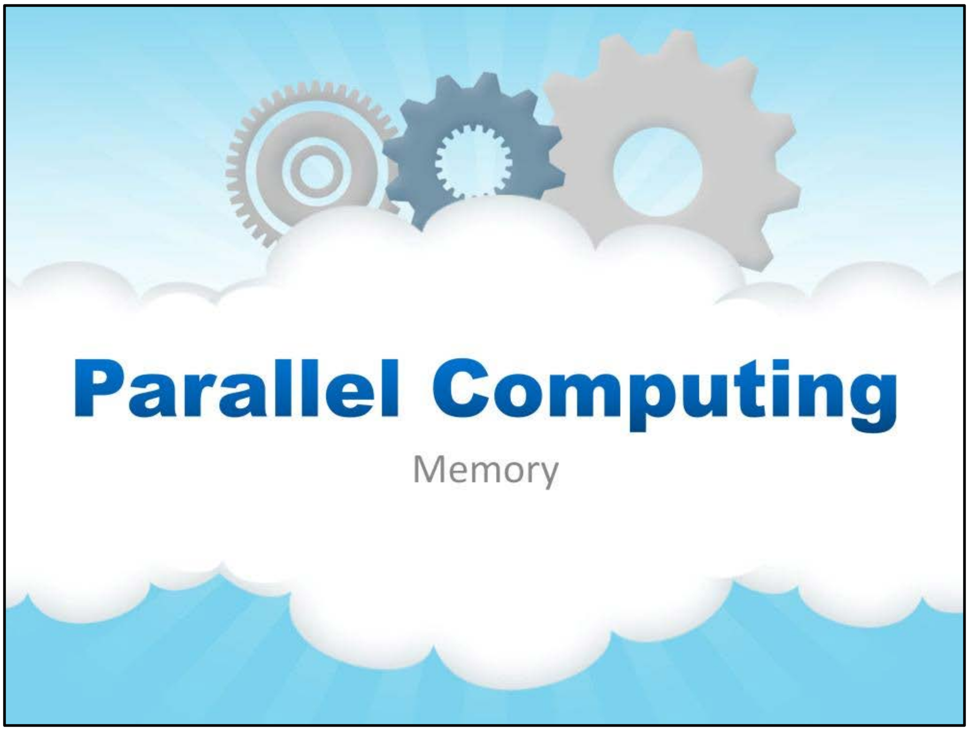
MIMD

In a MIMD computer every PU has its own control unit that reads, a separate stream of instructions dictating the execution in its PU.

Just as for SIMD machines,
a main subdivision of MIMD machines is into those having
- shared memory or
- distributed memory.

In the latter variant (**distributed memory**) each PU can have a local memory storing both instructions and data. This leads us to another main categorization of multiprocessors, – **shared memory** multiprocessors and **message passing** multiprocessors.

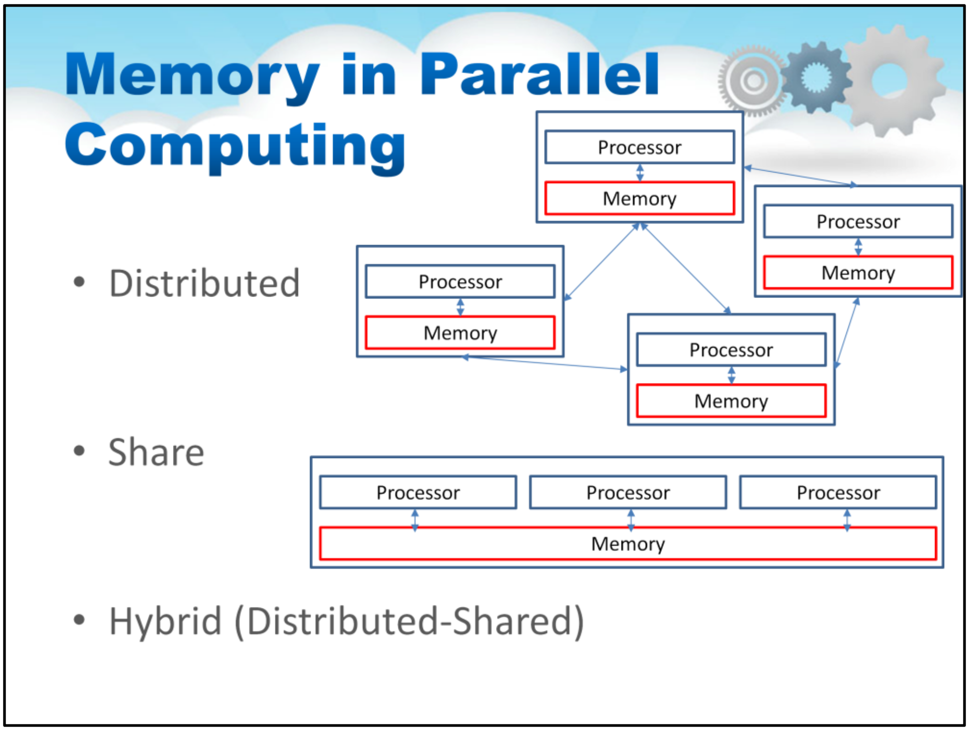
Let’s consider organization of memory in the NEXT SECTION…



Parallel Computing

Memory

Memory



Memory in Parallel Computing

- Distributed

Processor
Memory

Processor
Memory

Processor
Memory

Processor
Memory

- Share

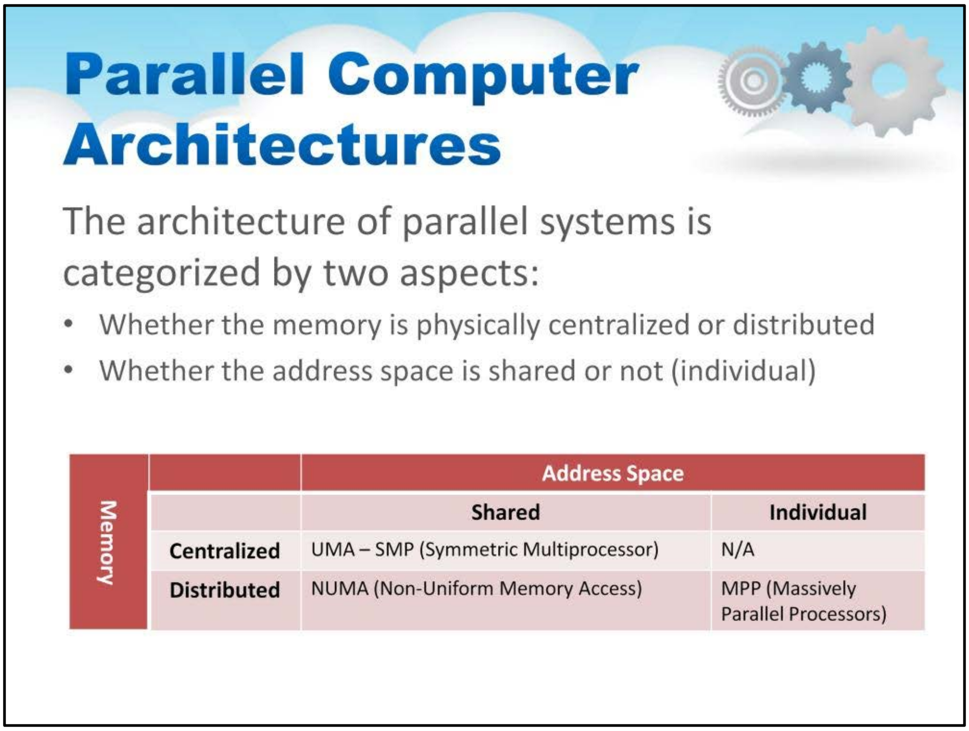
Processor
Processor
Processor
Memory

- Hybrid (Distributed-Shared)

Distributed Memory Architecture
Each processor has its own address space (local memory).
Changes done by each processor is not visible by others.

Shared Memory Architecture
Processors perform their operations independently,
but they share the same resources
since they have access to all the memory as global address space.

Hybrid (Distributed-Shared) Memory Architecture
It contains
- the shared memory components
and
- the distributed memory components.
It is used in most of today's fast and large parallel computers.



Parallel Computer Architectures

The architecture of parallel systems is categorized by two aspects:

- Whether the memory is physically centralized or distributed
- Whether the address space is shared or not (individual)

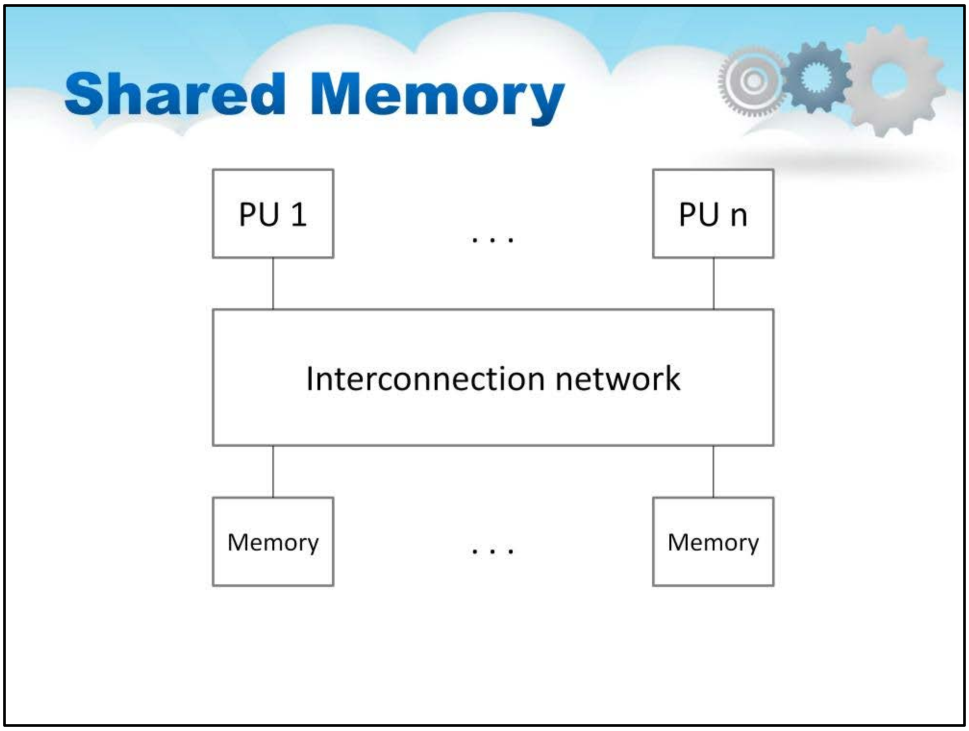
Memory		Address Space	
		Shared	Individual
	Centralized	UMA – SMP (Symmetric Multiprocessor)	N/A
	Distributed	NUMA (Non-Uniform Memory Access)	MPP (Massively Parallel Processors)

The architecture of parallel systems is categorized by two aspects:
- Whether the memory is physically centralized or distributed
- Whether the address space is shared or not

If
- memory is **centralized** AND address space is **shared** then we have **uniform memory access (UMA)** architecture;
- memory is **distributed** AND address space is **shared** then we have **non-uniform memory access (NUMA)** architecture;
- memory is **distributed** AND address space is **NOT shared** then we have **distributed memory (DM)** architecture.

Sometimes systems with
- **uniform memory access (UMA) architecture** are called **symmetric multiprocessors (SMP)**

and
- **distributed memory (DM) architecture** are called **massively parallel processors (MPP)**



Shared Memory

PU 1 . . . PU n

Interconnection network

Memory . . . Memory

Processors perform their operations independently,

but they **share the same memory resources**

since they have access to all the **memory as global address space**.

If a processor **changes the value in the memory**,

then it will be observed by all other processors.

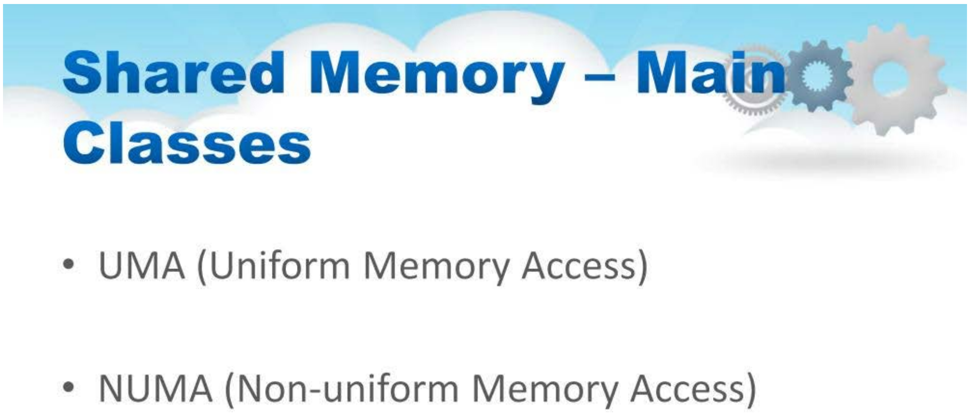
If many PUs try to access to a single memory module through a parallel interconnection

network, the memory could easily become a bottleneck. That is why,

it is common to use several physical memory modules **(memory banks)**

as shown in this Figure.

But this architecture is called a **centralized memory** system, because the modules (memory banks) are assembled as one subsystem that is equally accessible from all the processors.

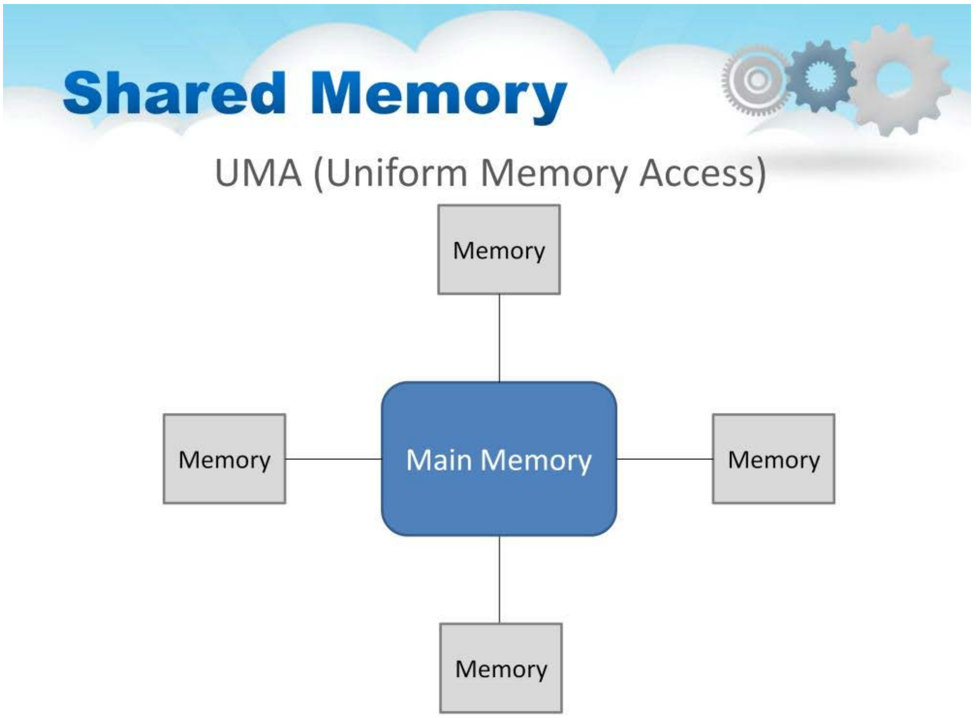


Shared Memory – Main Classes

- UMA (Uniform Memory Access)
- NUMA (Non-uniform Memory Access)

The following **main classes** (listed in this figure) should be considered in details:

- UMA (Uniform Memory Access)
- NUMA (Non-uniform Memory Access)



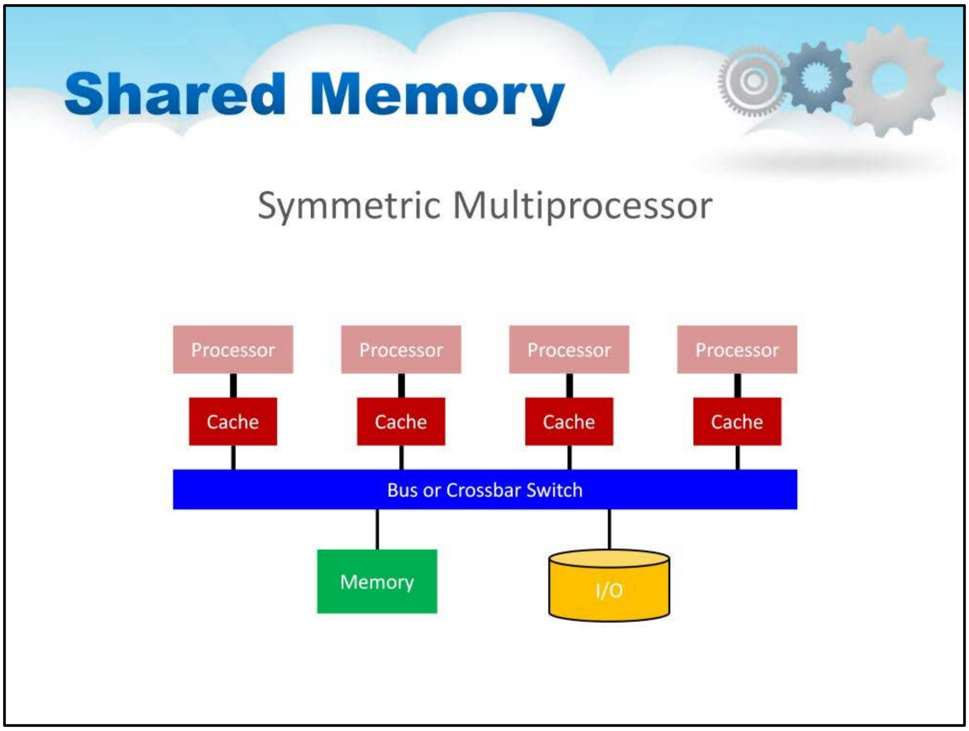
Shared Memory

UMA (Uniform Memory Access)

It gives **equal access** rights and access times to memory

Due to this uniformity of access,
these systems are often called **symmetric multiprocessors (SMP)**
or **uniform memory access (UMA)** architectures

Sometimes called **Cache Coherent UMA (CC-UMA)**

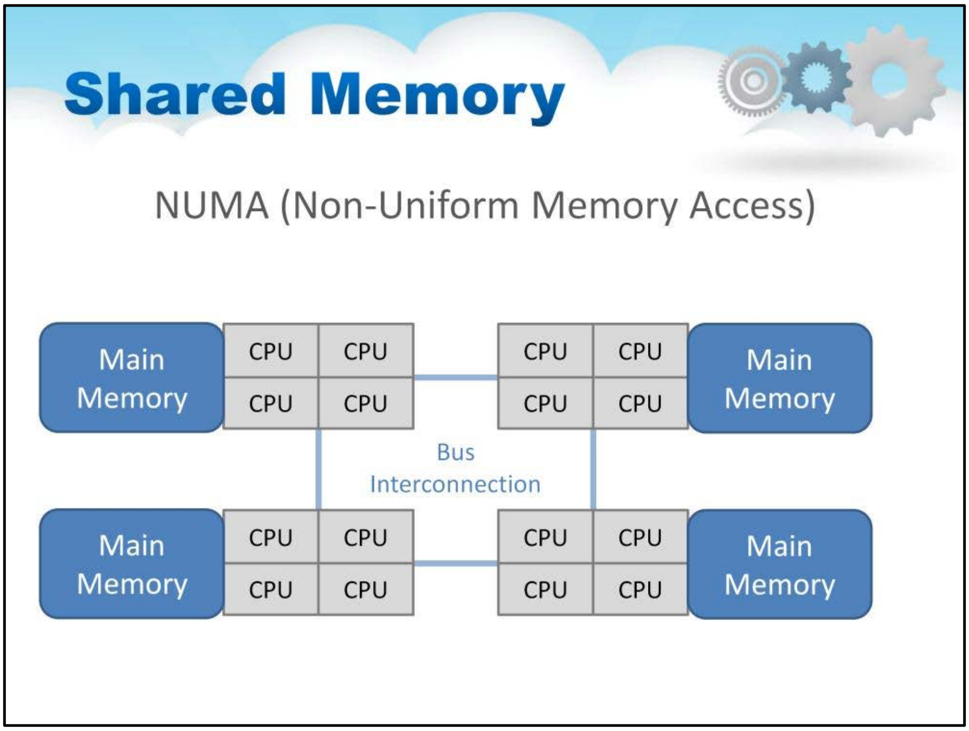


Shared Memory

Symmetric Multiprocessor

Symmetric Multiprocessor (SMP) architecture uses shared system resources that can be accessed equally from all processors.

Usually, a single OS controls the SMP machine and it schedules processes and threads on processors so that the load is balanced.



Shared Memory

NUMA (Non-Uniform Memory Access)

Main Memory

CPU CPU
CPU CPU

CPU CPU
CPU CPU

Main Memory

Bus Interconnection

Main Memory

CPU CPU
CPU CPU

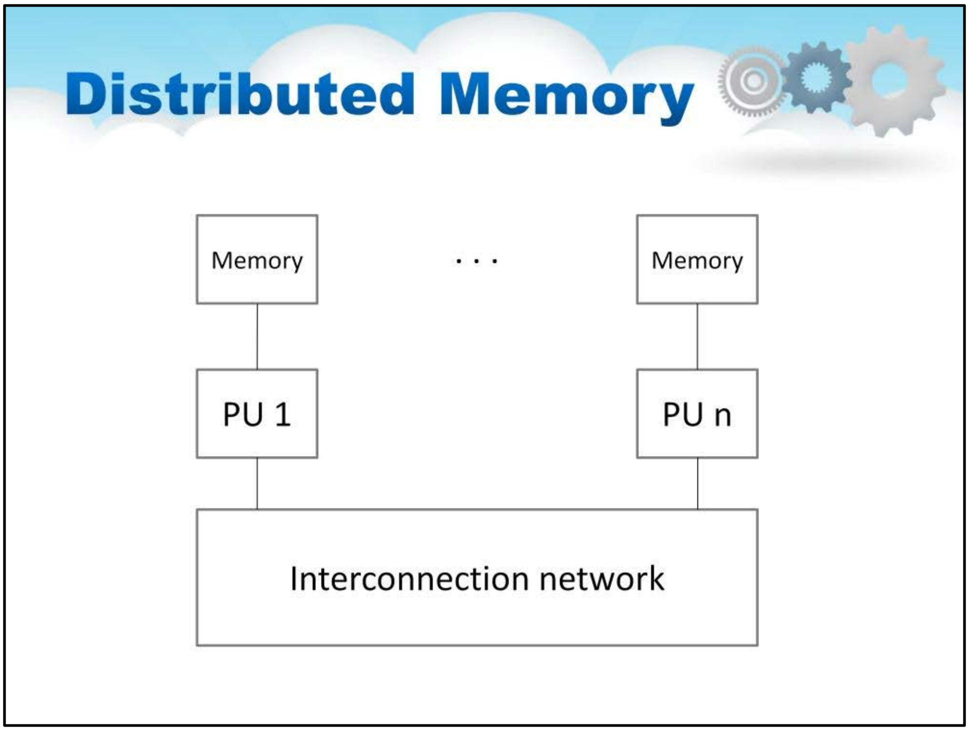
CPU CPU
CPU CPU

Main Memory

Usually physically linked 2 or more SMPs,
so they can access the memory of each other directly

Not all have equal access time to all memories

Memory access across the link is much slower



Distributed Memory

Memory

. . .

Memory

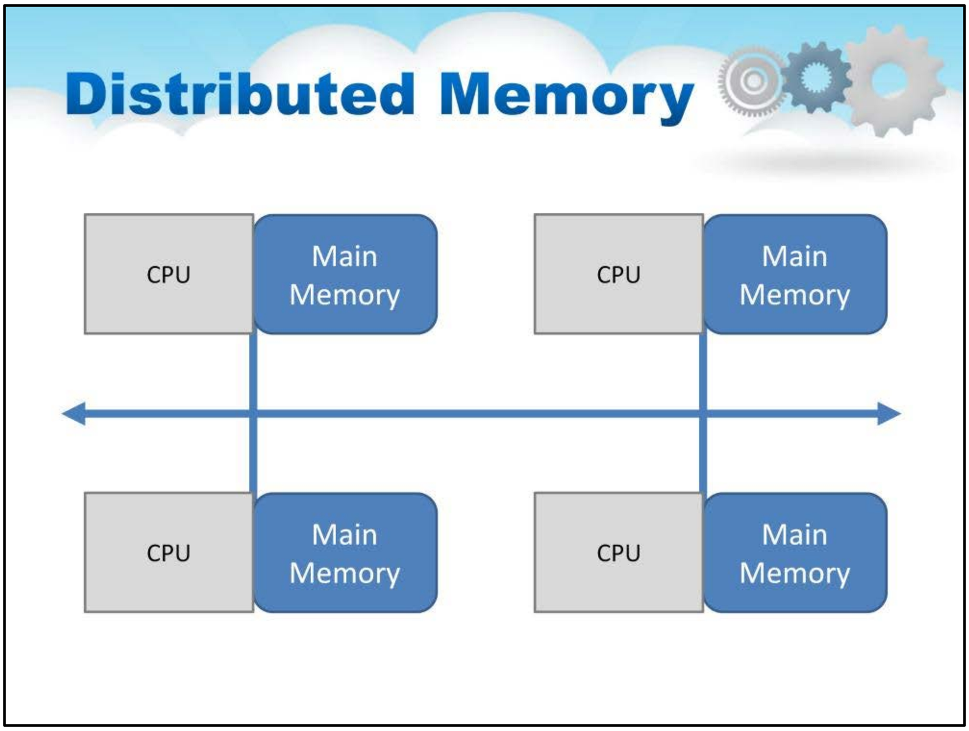
PU 1

PU n

Interconnection network

The main alternative to centralized memory is called **distributed memory** and is shown in this Figure.

The memory modules are located together with the processors.



Distributed Memory

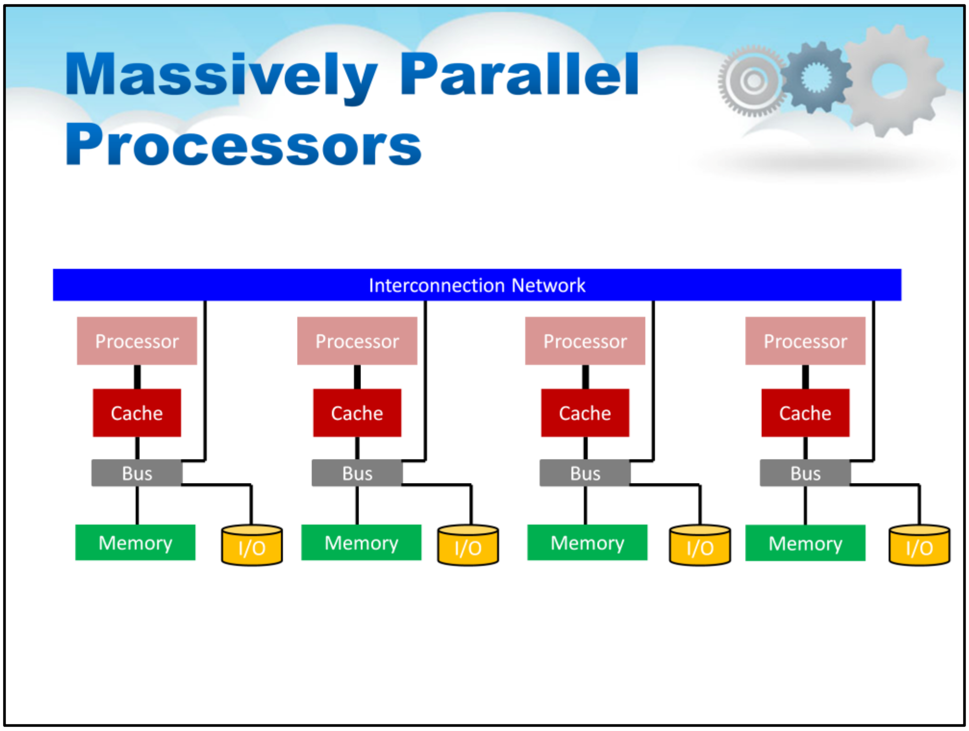
- Each CPU has its own local memory.

- Changes done by each processor are not visible by others.

- Processors are connected by network.

- The program must define a way to transfer data between processors.

Such systems with **distributed memory (DM) architectures** are called **massively parallel processors (MPP).**



Massively Parallel Processors

Interconnection Network

Processor

Cache

Bus

Memory

I/O

Processor

Cache

Bus

Memory

I/O

Processor

Cache

Bus

Memory

I/O

Processor

Cache

Bus

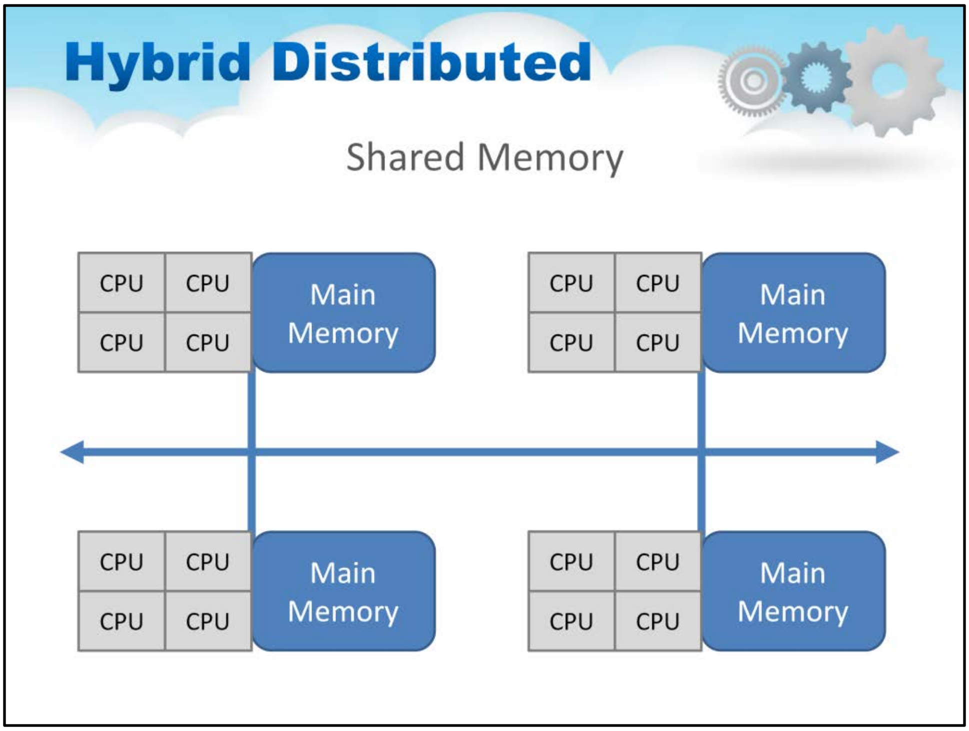
Memory

I/O

Massively Parallel Processors (MPP) architecture consists of multiple nodes.

Each node has its own processor, memory and I/O subsystem.

An independent OS runs at each node.



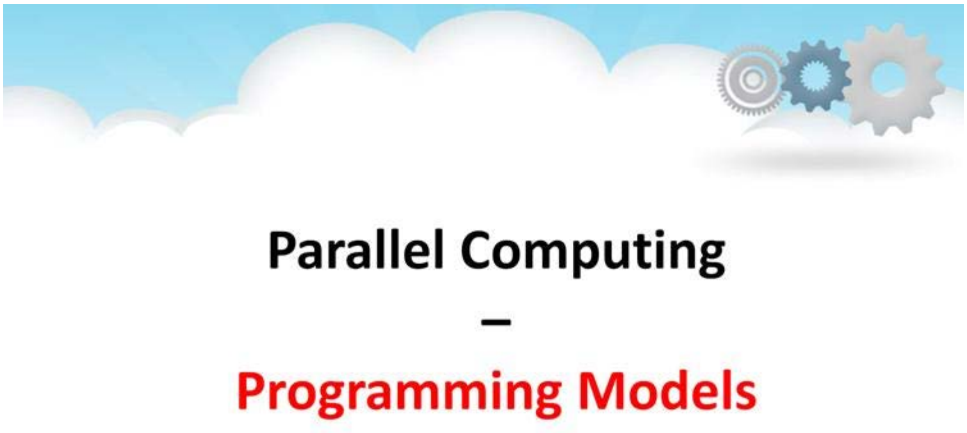
Hybrid Distributed

Shared Memory

Used in the most of the current parallel systems

The shared memory components are SMP nodes

The distributed memory component is a network

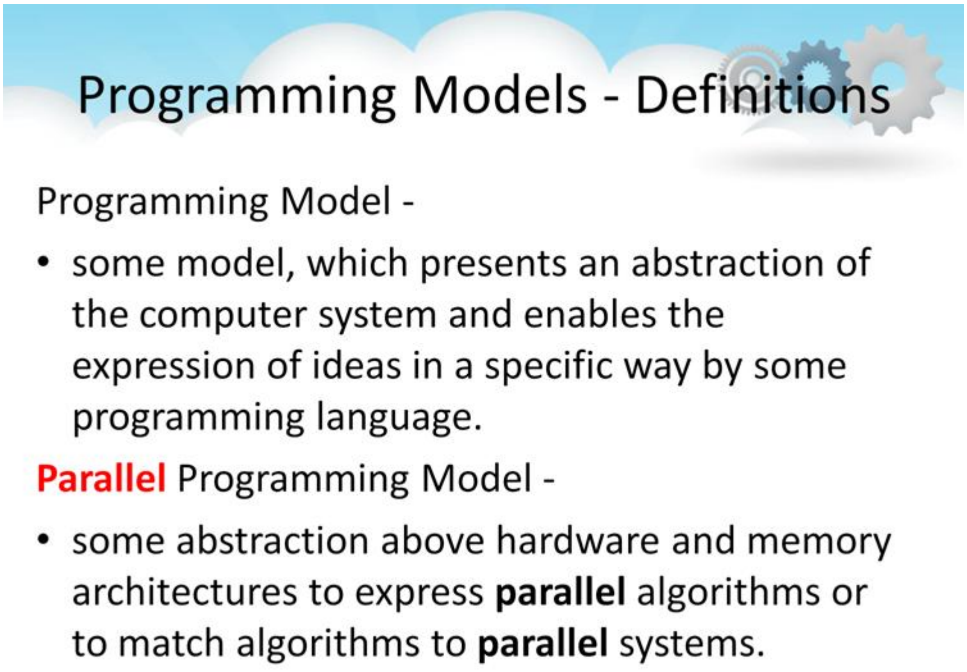


Parallel Computing

–

Programming Models

Programming Models



Programming Models - Definitions

Programming Model -

- some model, which presents an abstraction of the computer system and enables the expression of ideas in a specific way by some programming language.

Parallel Programming Model -

- some abstraction above hardware and memory architectures to express **parallel** algorithms or to match algorithms to **parallel** systems.

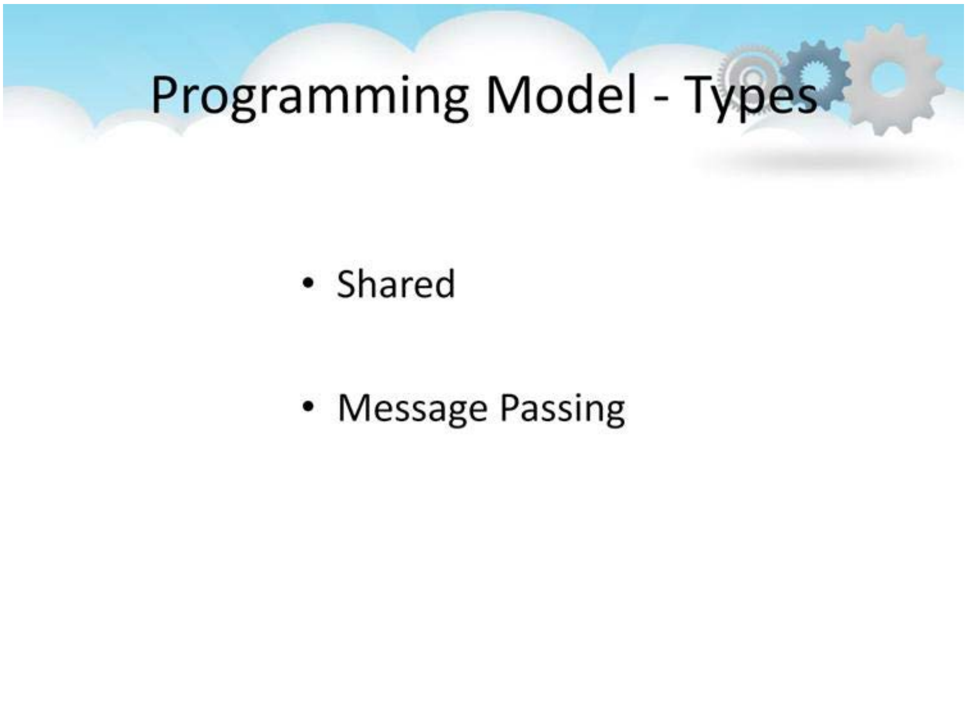
What is a programming model?
[Read 1st sentence from this slide]

What is a parallel programming model?
[Read 2nd sentence from this slide]

It determines **how easily** programmers can specify their algorithms into parallel unit of computations (i.e., tasks) that the hardware understands

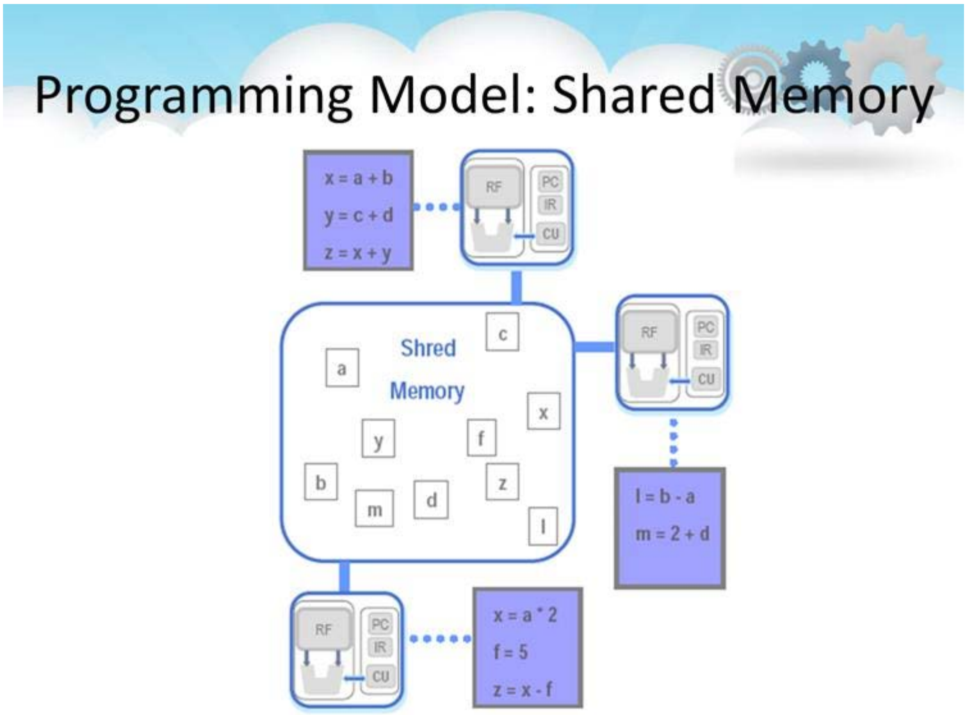
It determines **how efficiently** parallel tasks can be executed on the hardware

Main Goal: **utilize all** the processors of the underlying architecture (e.g., SMP, MPP, CMP) and **minimize the time** of your program



Programming Model - Types

- Shared
- Message Passing



Programming Model: Shared Memory

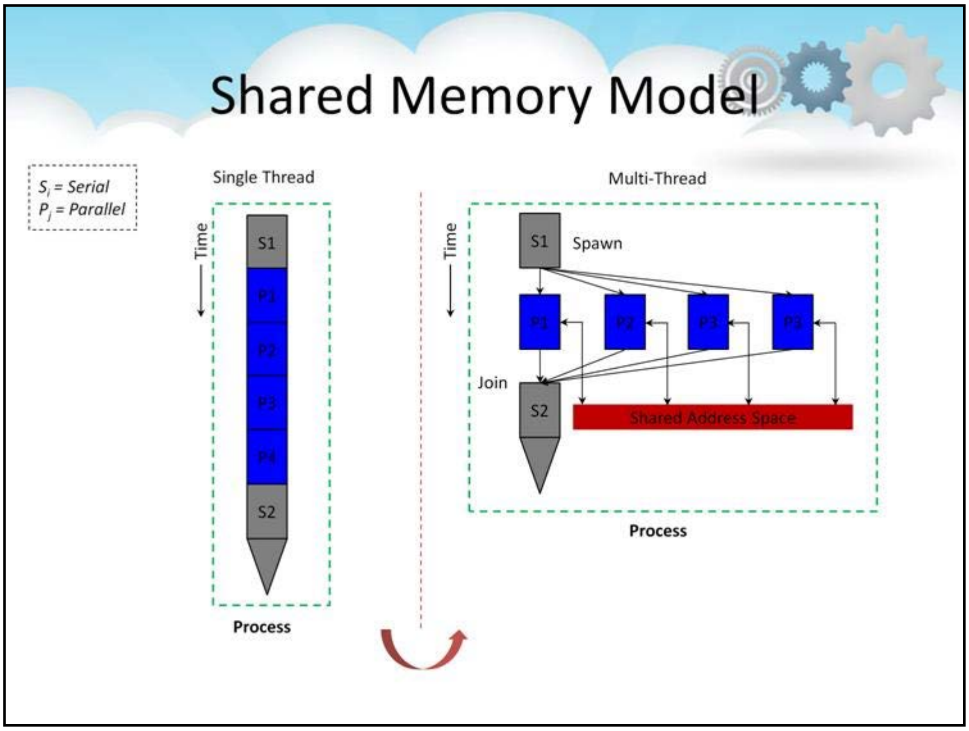
Processors read/write the variables stored in a shared address space asynchronously.

Access to the shared memory is controlled by some mechanisms (locks/semaphores)

Advantage:
Program development can be simplified,
because no process owns the data stored in memory.
Because all processors can access the same variables,
referencing data stored in memory is similar to traditional single processor programs.

-

Disadvantage:
it’s difficult to understand and manage data locality.
Data are saved locally to the processor,
that is working on it conserves memory access to this processor.
This causes bus traffic when multiple processors are trying to access the same data.



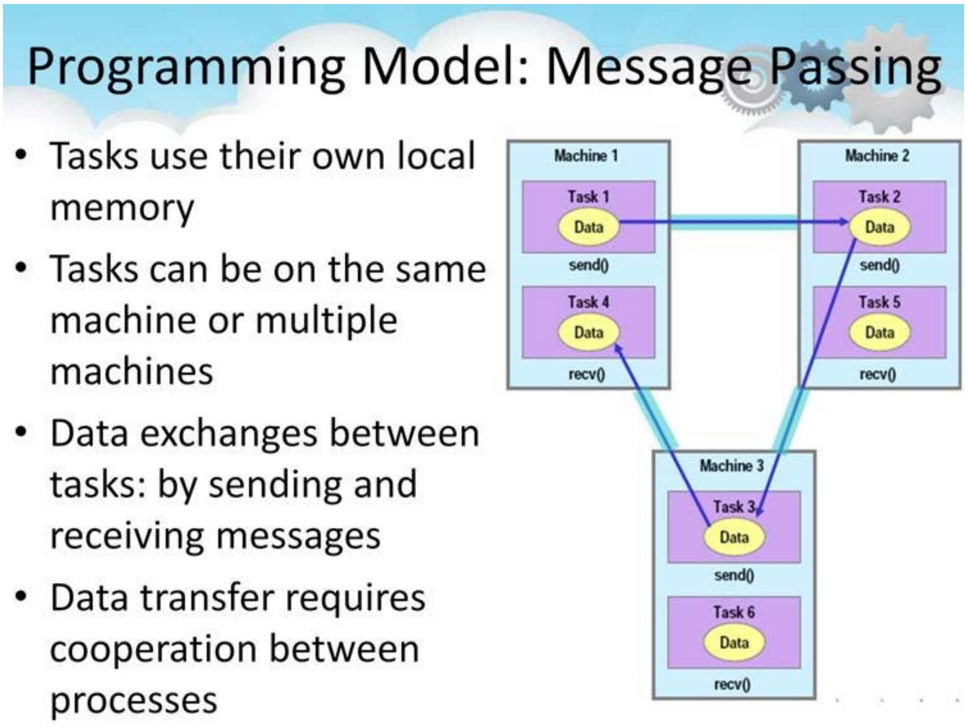
Shared Memory Model

In the shared memory programming model, the abstraction is that parallel tasks can access any location of the memory

Parallel tasks can communicate through reading and writing common memory locations

This is similar to threads from a single process which share a single address space

Multi-threaded programs (e.g., OpenMP programs) are the best fit with shared memory programming model



Programming Model: Message Passing

- Tasks use their own local memory
- Tasks can be on the same machine or multiple machines
- Data exchanges between tasks: by sending and receiving messages
- Data transfer requires cooperation between processes

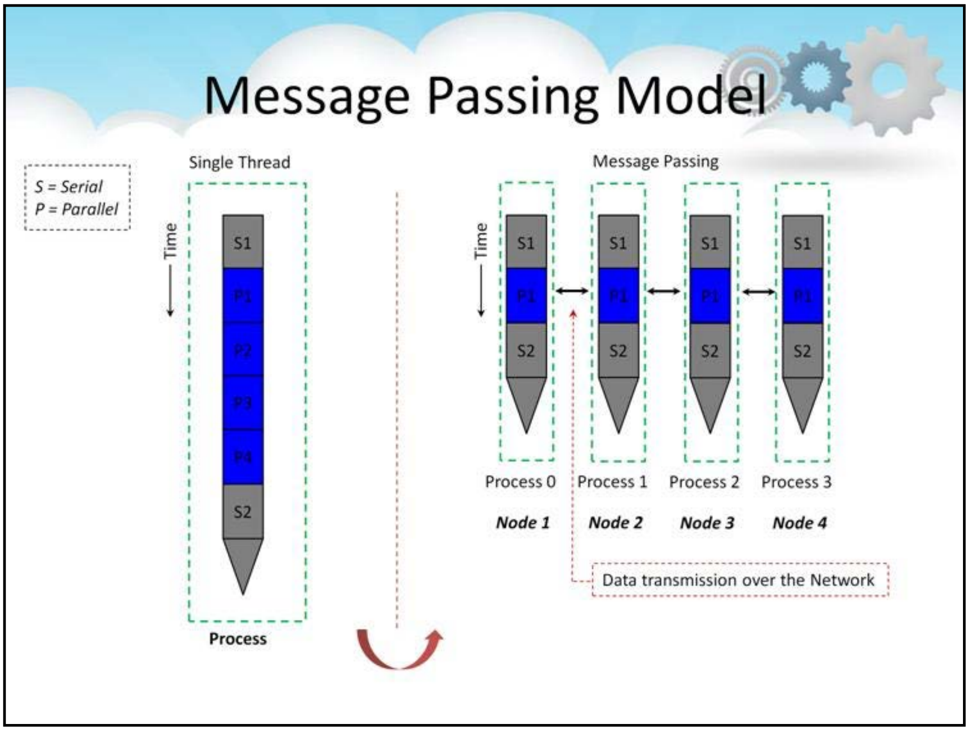
In message passing, parallel tasks have their own local memories

One task cannot access another task’s memory

Hence, to communicate data they have to rely on explicit messages sent to each other

This is similar to the abstraction of processes which do not share an address space

Message Passing Interface (MPI) programs are the best fit with the message passing programming model



Message Passing Model

S = Serial
P = Parallel

Single Thread

Time

S1
P1
P2
P3
P4
S2

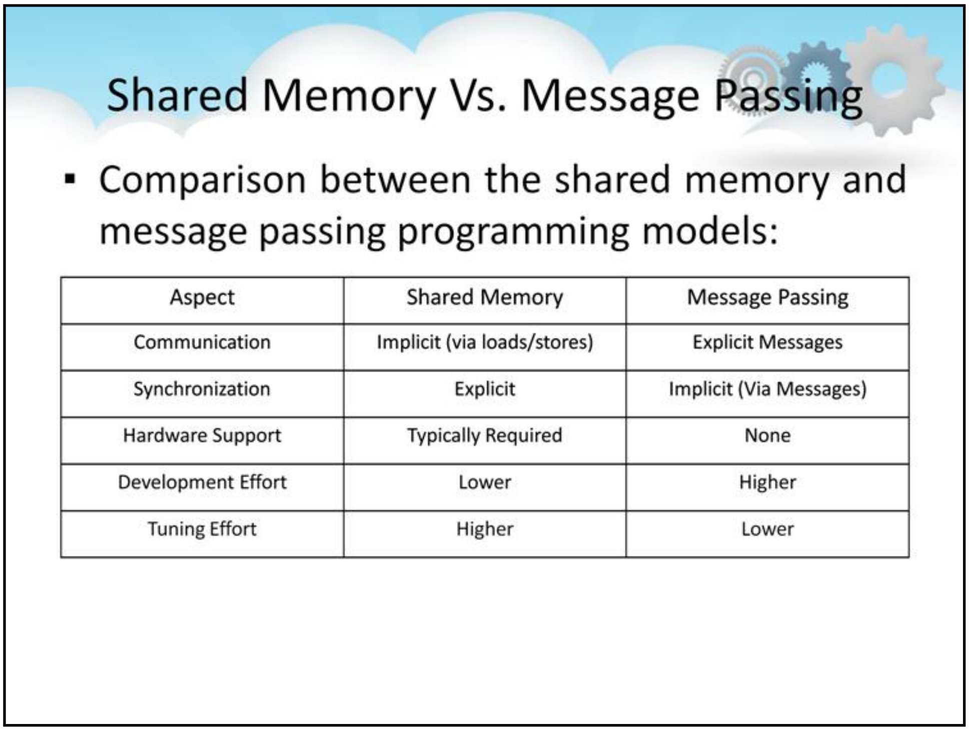
Process

Message Passing

Time

S1	S1	S1	S1
P1	P1	P1	P1
S2	S2	S2	S2
Process 0	Process 1	Process 2	Process 3
Node 1	***Node 2***	***Node 3***	***Node 4***

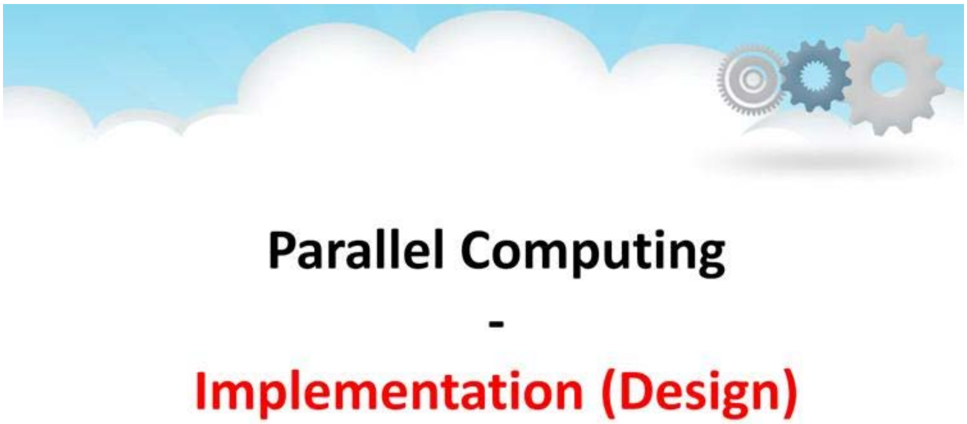
Data transmission over the Network



Shared Memory Vs. Message Passing

- Comparison between the shared memory and message passing programming models:

Aspect	Shared Memory	Message Passing
Communication	Implicit (via loads/stores)	Explicit Messages
Synchronization	Explicit	Implicit (Via Messages)
Hardware Support	Typically Required	None
Development Effort	Lower	Higher
Tuning Effort	Higher	Lower

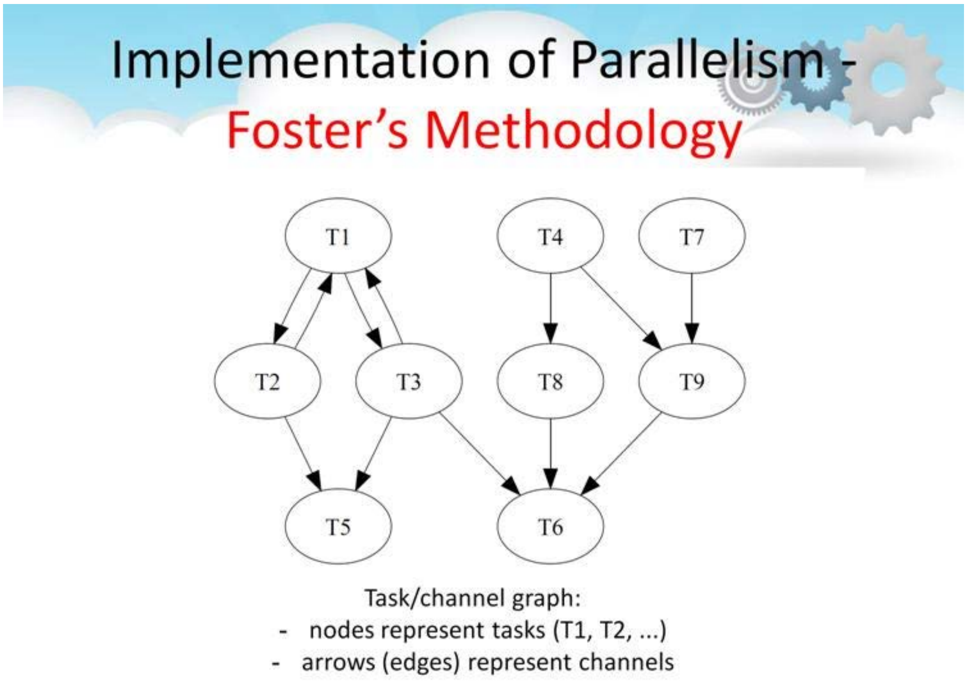


Parallel Computing

-

Implementation (Design)

Parallel Computing Design



Implementation of Parallelism - Foster's Methodology

Task/channel graph:

- nodes represent tasks (T1, T2, ...)
- arrows (edges) represent channels

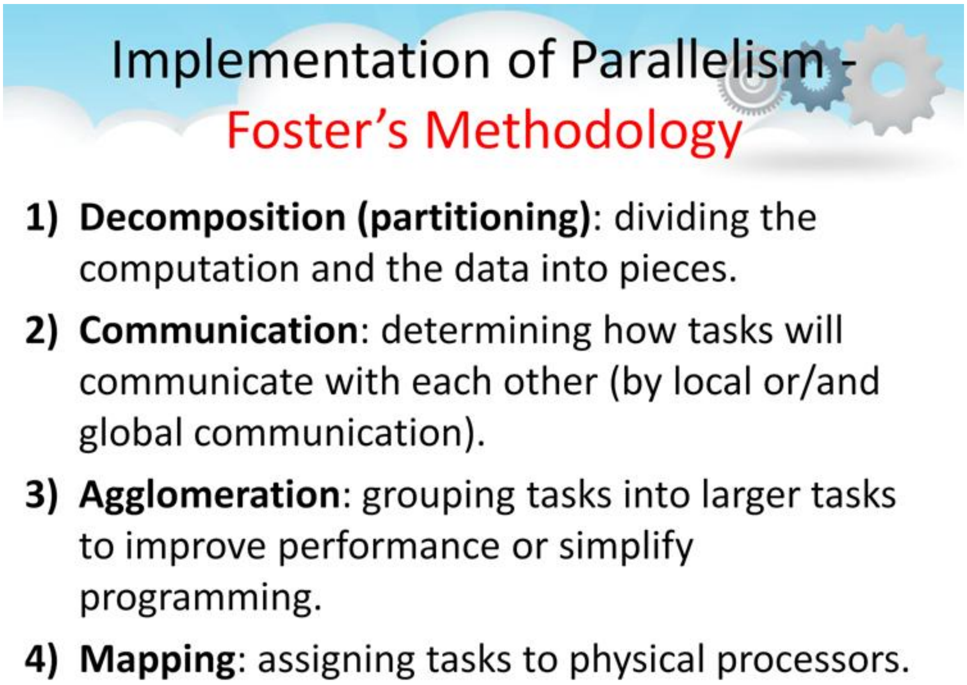
The task/channel methodology described here was proposed by Ian Foster.

In this methodology, a parallel program is viewed as a collection of tasks
that communicate by sending messages to each other through channels.

This Figure shows a task/channel representation of a hypothetical parallel program.

A task consists of an executable unit (think of it as a program)

A channel is a message queue
that connects one task's output port to another task's input port.



Implementation of Parallelism - Foster's Methodology

1) **Decomposition (partitioning)**: dividing the computation and the data into pieces.
2) **Communication**: determining how tasks will communicate with each other (by local or/and global communication).
3) **Agglomeration**: grouping tasks into larger tasks to improve performance or simplify programming.
4) **Mapping**: assigning tasks to physical processors.

Why we need this methodology?

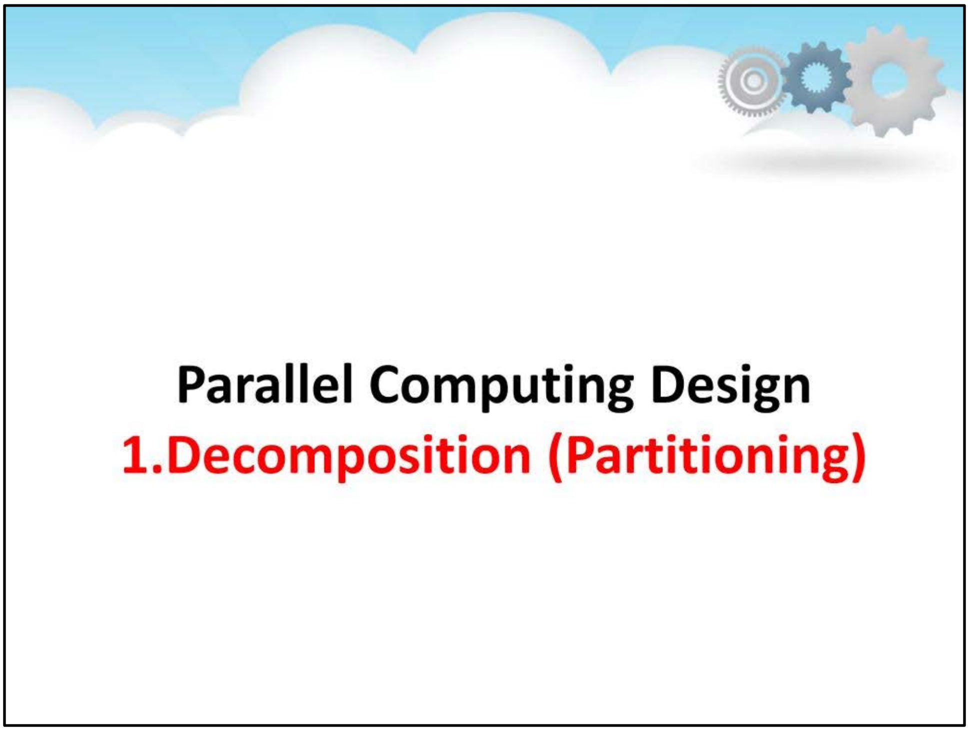
It is not easy to design a parallel program from scratch without some methodology.

It is far better to use a proven methodology that is general enough and that can be followed easily.

Otherwise you will not learn from the mistakes of others.

In 1995, Ian Foster proposed such a methodology, which has come to be called Foster's design methodology.

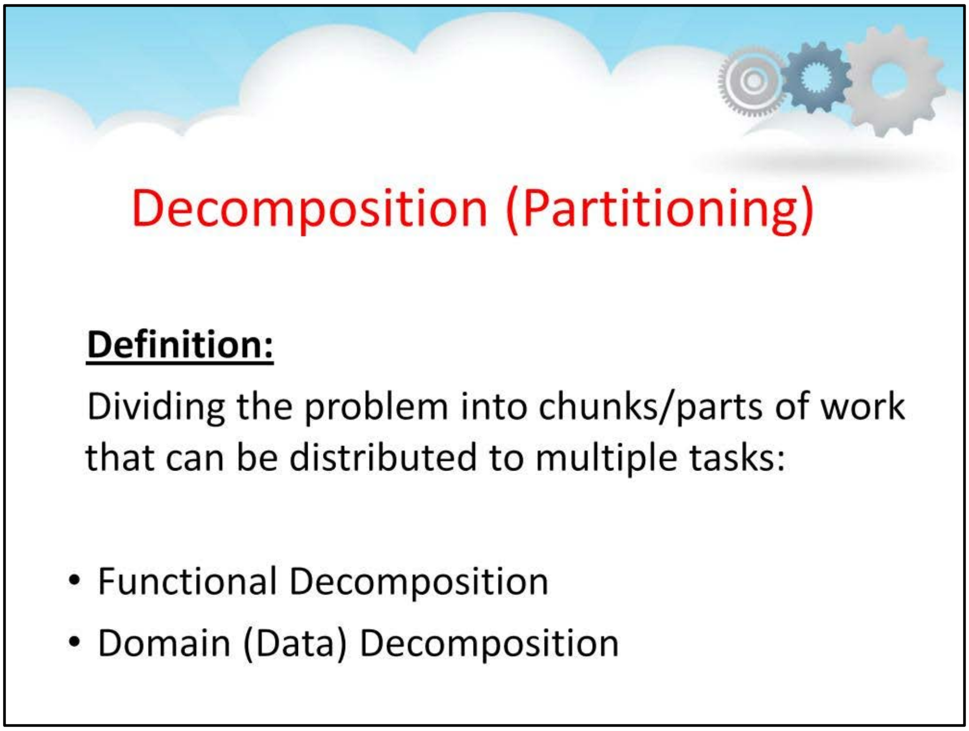
It is a four-stage design process as shown in this Figure with their brief descriptions.



Parallel Computing Design

1.Decomposition (Partitioning)

Decomposition

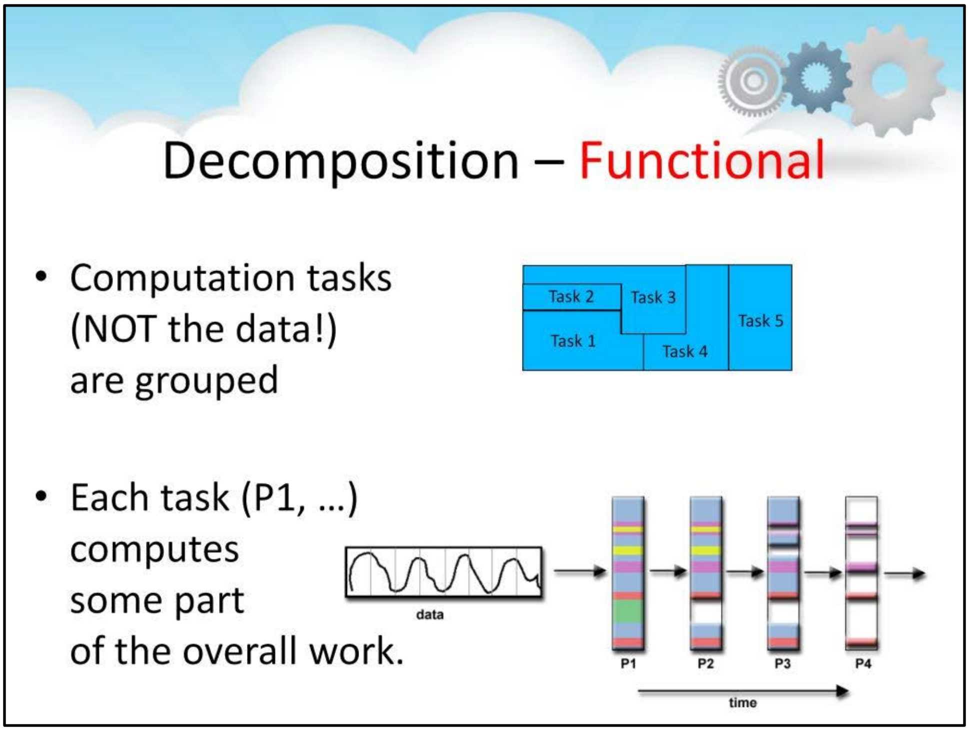


Decomposition (Partitioning)

Definition:

Dividing the problem into chunks/parts of work that can be distributed to multiple tasks:

- Functional Decomposition
- Domain (Data) Decomposition



Decomposition – Functional

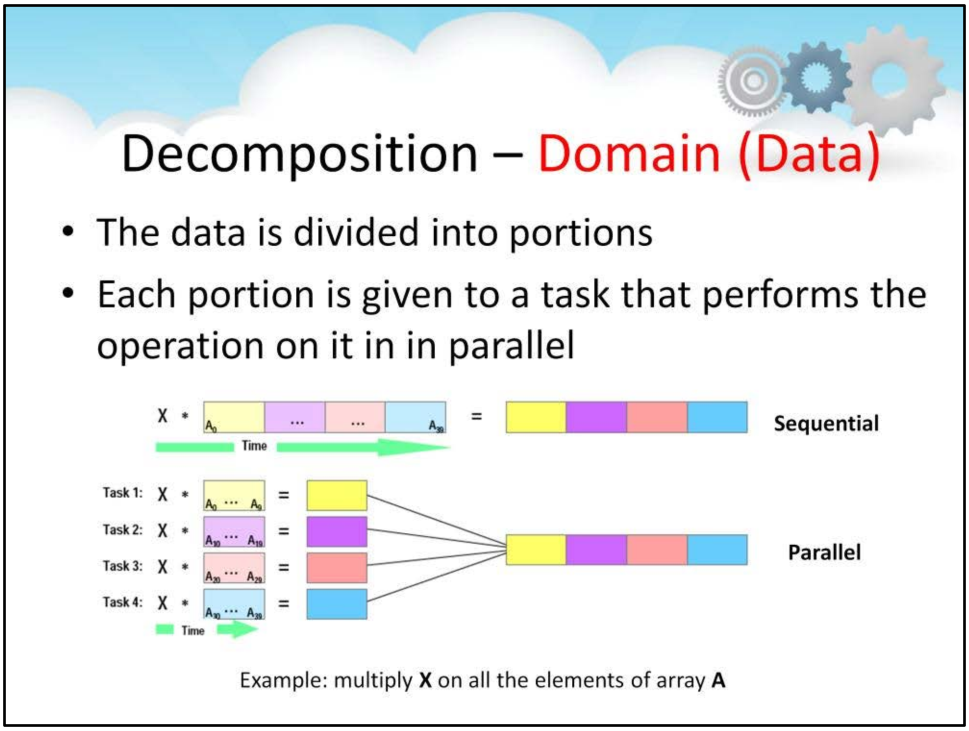
- Computation tasks (NOT the data!) are grouped
- Each task (P1, …) computes some part of the overall work.

Functional parallelism is possible when there are separate
operations that can be applied simultaneously, typically to different parts of the data set.

Functional decomposition is the paradigm in which primitive tasks are assigned
to separate functions, and then the data to which these functions can be applied is identified.

Here the focus is on the computation that is to be performed rather than on the data.
The problem is decomposed according to the work that must be done.
Each task then performs a portion of the overall work as shown here **in top image.**

Sometimes this results in a pipelined approach, as in an audio signal processing example,
as shown here **in bottom image** for digital signal processing.
The signal is passed through four different filters. Each filter is a separate process.
As the data is passed through a filter it is sent to the next one in the line.
This is so-called a **computational pipeline**.



Decomposition – Domain (Data)

- The data is divided into portions
- Each portion is given to a task that performs the operation on it in in parallel

X * A_0 … … A_{39} =
Time
Sequential
Task 1: X * A_0 … A_9 =
Task 2: X * A_{10} … A_{19} =
Task 3: X * A_{20} … A_{29} =
Task 4: X * A_{30} … A_{39} =
Time
Parallel

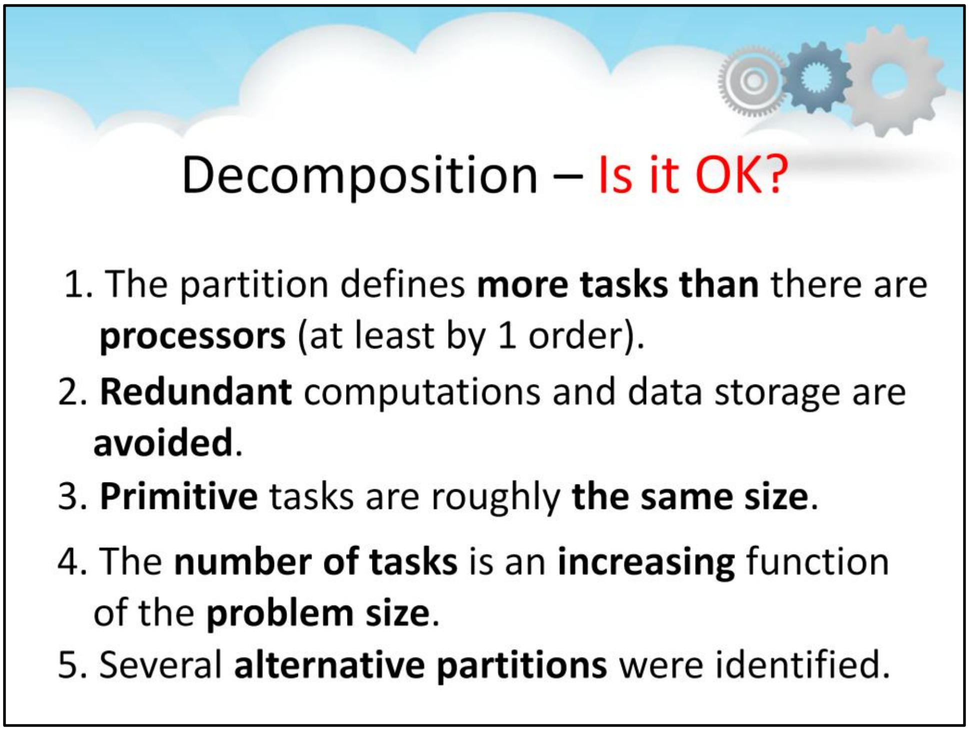
Example: multiply **X** on all the elements of array **A**

The same operation is being performed on different parts of the same data structure,
i.e. each processor performs the task of its part of the data.

• A data is divided into chunks, operations are performed on each chunk concurrently

• A set of tasks are carried out on the same data structure,
but on a different part of this structure

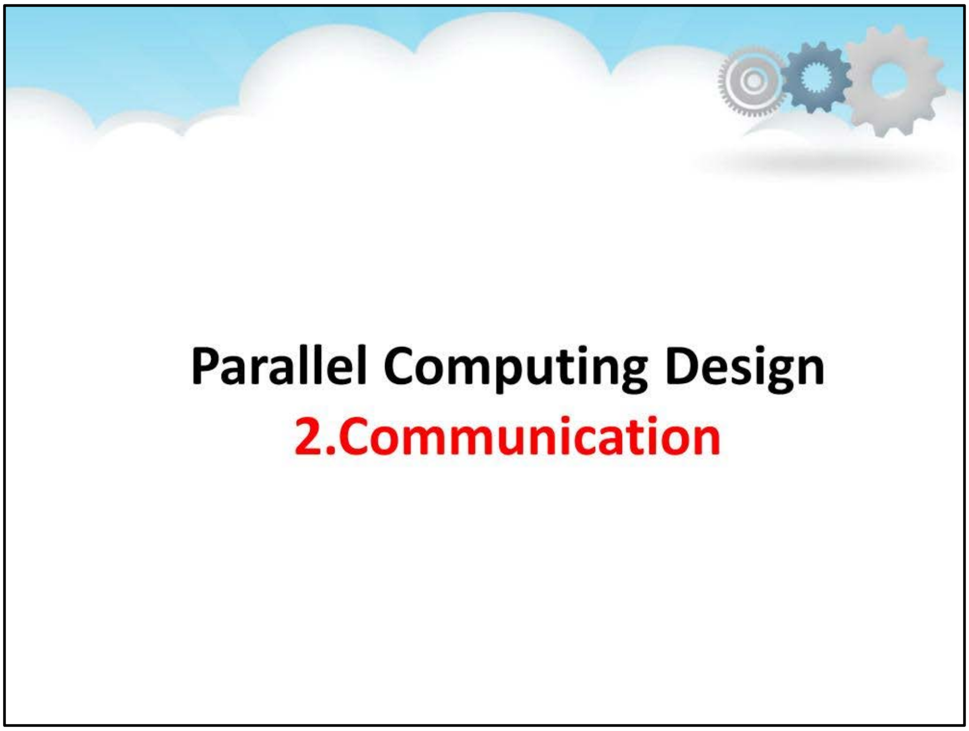
• Tasks on the same part of the data structure do the same operations on each instance of this data.



Decomposition – Is it OK?

1. The partition defines **more tasks than** there are **processors** (at least by 1 order).
2. **Redundant** computations and data storage are **avoided**.
3. **Primitive** tasks are roughly **the same size**.
4. The **number of tasks** is an **increasing** function of the **problem size**.
5. Several **alternative partitions** were identified.

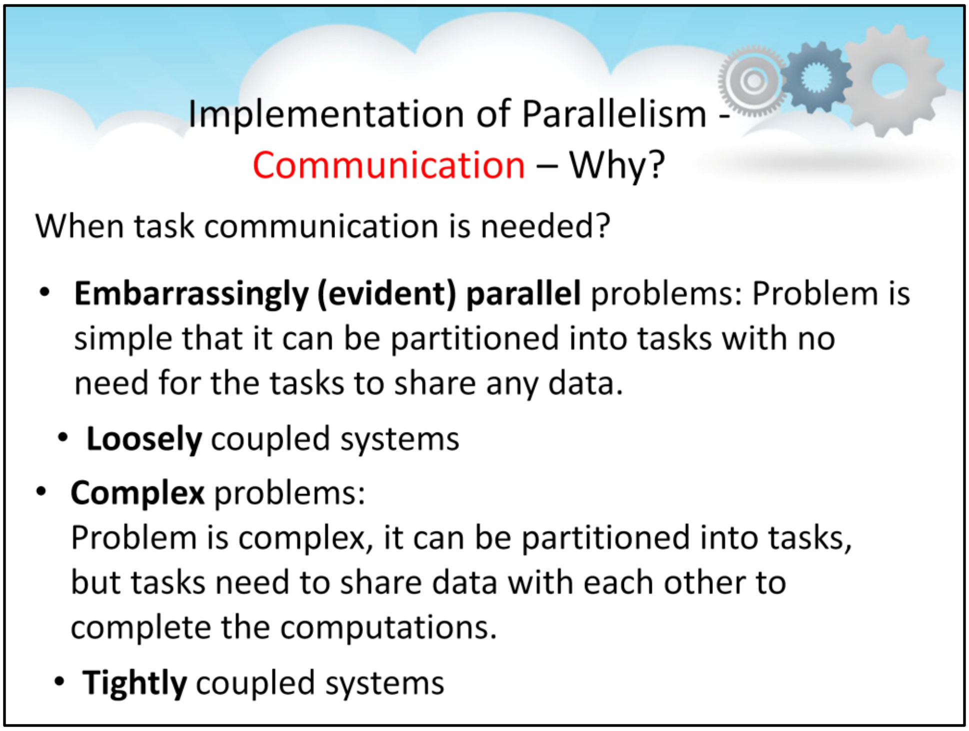
Foster provides a checklist for evaluating the partitioning stage.
Your partitioning should satisfy the following criteria as much as possible:
1.The partition defines at least an order of magnitude more tasks
than there are processors in your target computer.
If not, you have little flexibility in subsequent design stages.
2.Redundant computations and data storage are avoided as much as possible.
If not, the resulting algorithm may not be able to deal with large problems.
3.Primitive tasks are roughly the same size.
If not, it may be hard to allocate to each processor equal amounts of work,
and this will cause an overall decrease in performance.
4.The number of tasks is an increasing function of the problem size.
Ideally, an increase in problem size should increase the number of tasks rather
than the size of individual tasks. If this is not the case, your parallel algorithm
may not be able to solve larger problems when more processors are available.
5.You have identified several alternative partitions.
You can maximize flexibility in subsequent design stages by considering alternatives now.
Remember to investigate both domain and functional decompositions.



Parallel Computing Design

2.Communication

Communication

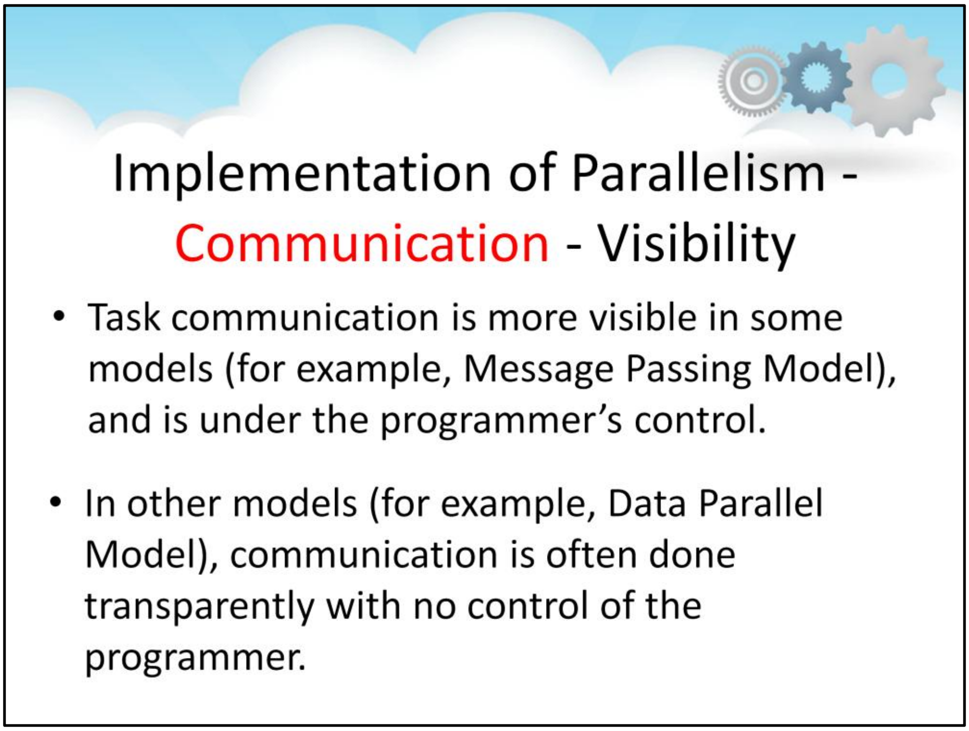


Implementation of Parallelism - Communication – Why?

When task communication is needed?

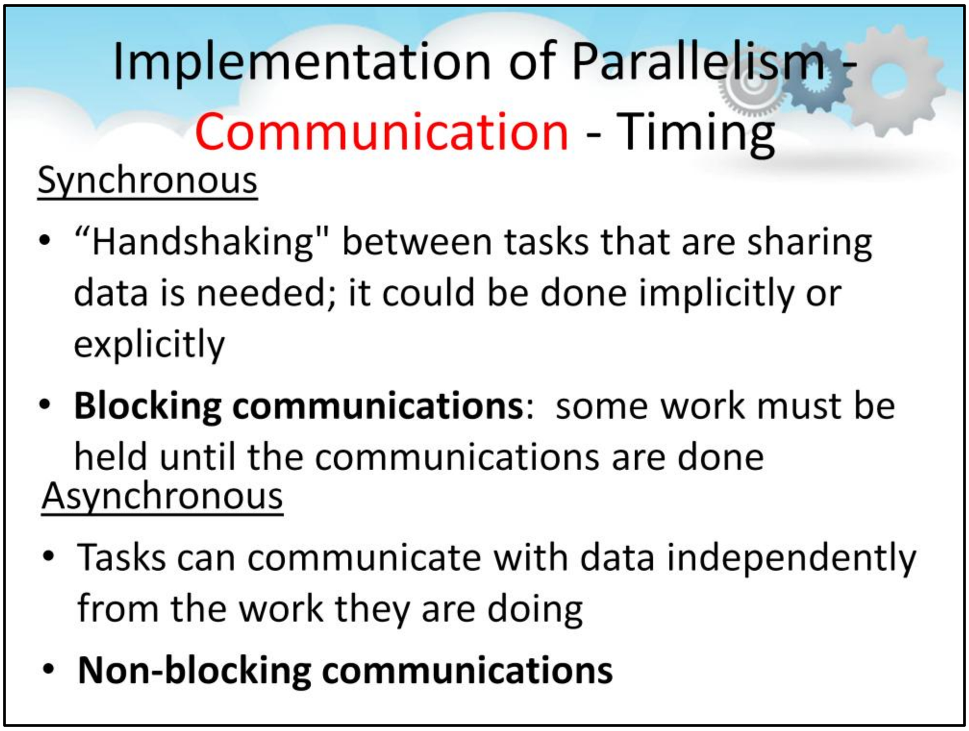
- **Embarrassingly (evident) parallel** problems: Problem is simple that it can be partitioned into tasks with no need for the tasks to share any data.
  - **Loosely** coupled systems
- **Complex** problems:
  Problem is complex, it can be partitioned into tasks, but tasks need to share data with each other to complete the computations.
  - **Tightly** coupled systems

When the entire computation is one sequential program,
all of the data is available to all parts of the program.
When that computation is divided up into independent tasks
that may execute in separate processors,
some of the data needed by a task may reside in its local memory,
but some of it may reside in that of other tasks.
As a result, these tasks may need to exchange data with one another.
This information flow is specified in the communication stage of the design.
This inter-task communication does not exist in a sequential program;
it is an artifact of the parallelization of the computation.
Therefore considered to be entirely **overhead**.
Overhead is just a term that means the price we pay to produce something,
but that is not directly a part of what we produce.
We want to minimize this overhead in our design;
therefore it is important to identify it.
We also want to design the program so
that delays caused by this communication are minimal.



Implementation of Parallelism - Communication - Visibility

- Task communication is more visible in some models (for example, Message Passing Model), and is under the programmer's control.
- In other models (for example, Data Parallel Model), communication is often done transparently with no control of the programmer.



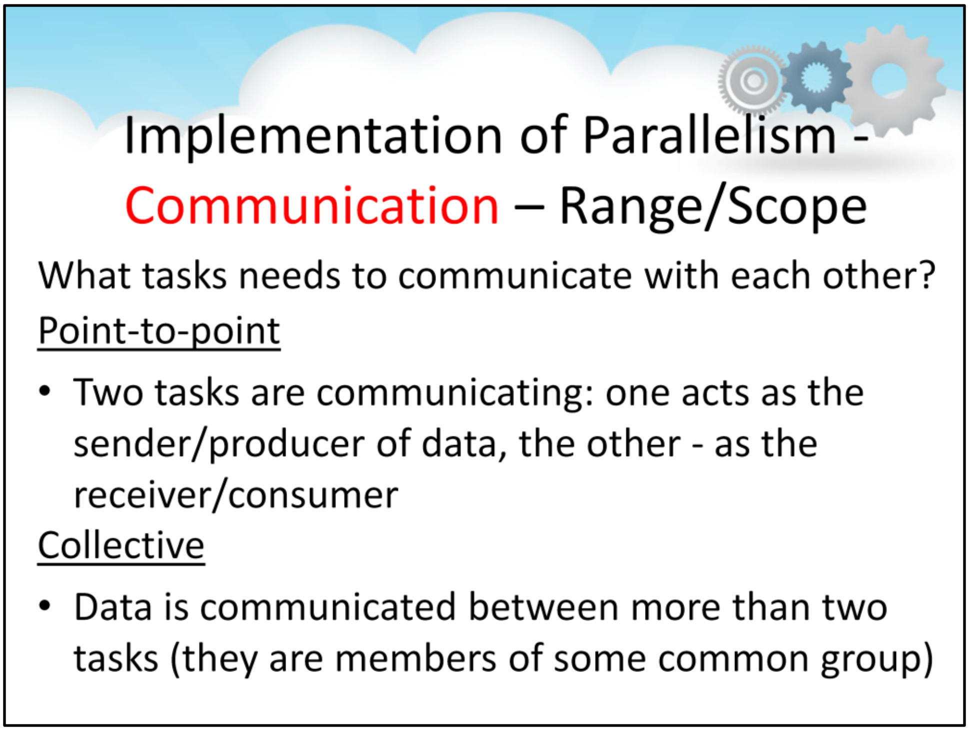
Implementation of Parallelism - Communication - Timing

Synchronous

- “Handshaking" between tasks that are sharing data is needed; it could be done implicitly or explicitly
- **Blocking communications**: some work must be held until the communications are done

Asynchronous

- Tasks can communicate with data independently from the work they are doing
- **Non-blocking communications**



Implementation of Parallelism - Communication – Range/Scope

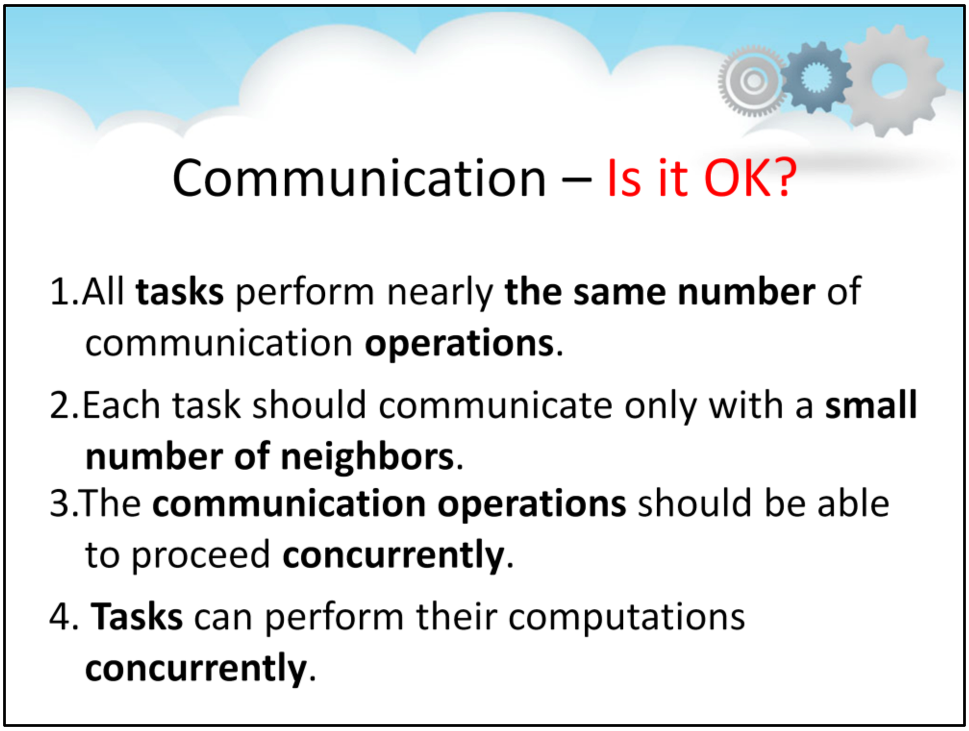
What tasks needs to communicate with each other?

Point-to-point

- Two tasks are communicating: one acts as the sender/producer of data, the other - as the receiver/consumer

Collective

- Data is communicated between more than two tasks (they are members of some common group)



Communication – Is it OK?

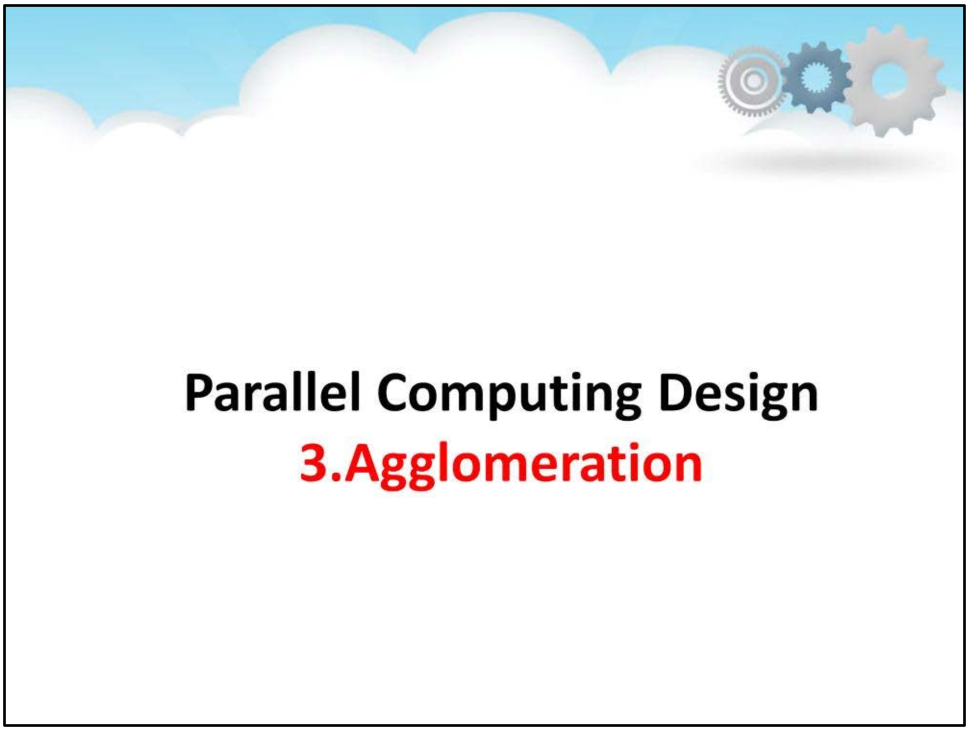
1.All **tasks** perform nearly **the same number** of communication **operations**.

2.Each task should communicate only with a **small number of neighbors**.

3.The **communication operations** should be able to proceed **concurrently**.

4. **Tasks** can perform their computations **concurrently**.

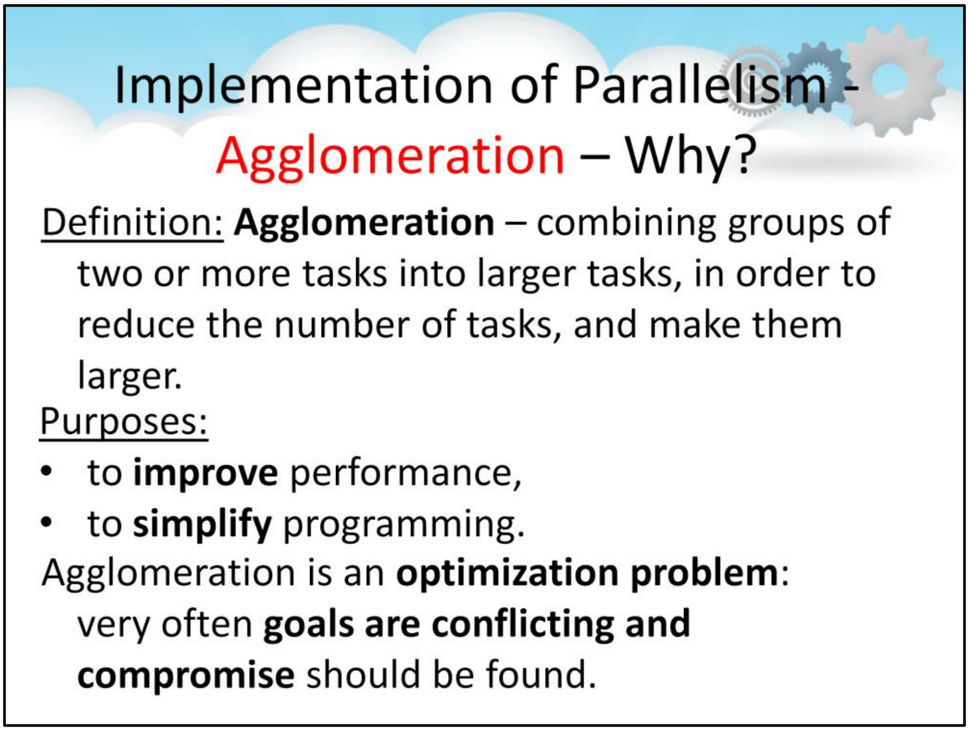
You should evaluate your design solution by these criteria:
1.All tasks perform about the same number of communication operations.
Unbalanced communication requirements suggest a design
that will not scale to a larger size instance of the problem.
2.Each task should communicate only with a small number of neighbors.
If each task must communicate with many other tasks,
it will add too much overhead.
3.The communication operations should be able to proceed concurrently.
If not, your algorithm is likely to be inefficient and non-scalable to a larger problem.
4. Tasks can perform their computations concurrently.
If not, your algorithm is likely to be inefficient and non-scalable to a larger problem.



Parallel Computing Design

3.Agglomeration

Agglomeration



Implementation of Parallelism - Agglomeration – Why?

Definition: **Agglomeration** – combining groups of two or more tasks into larger tasks, in order to reduce the number of tasks, and make them larger.

Purposes:

- to **improve** performance,
- to **simplify** programming.

Agglomeration is an **optimization problem**: very often **goals are conflicting and compromise** should be found.

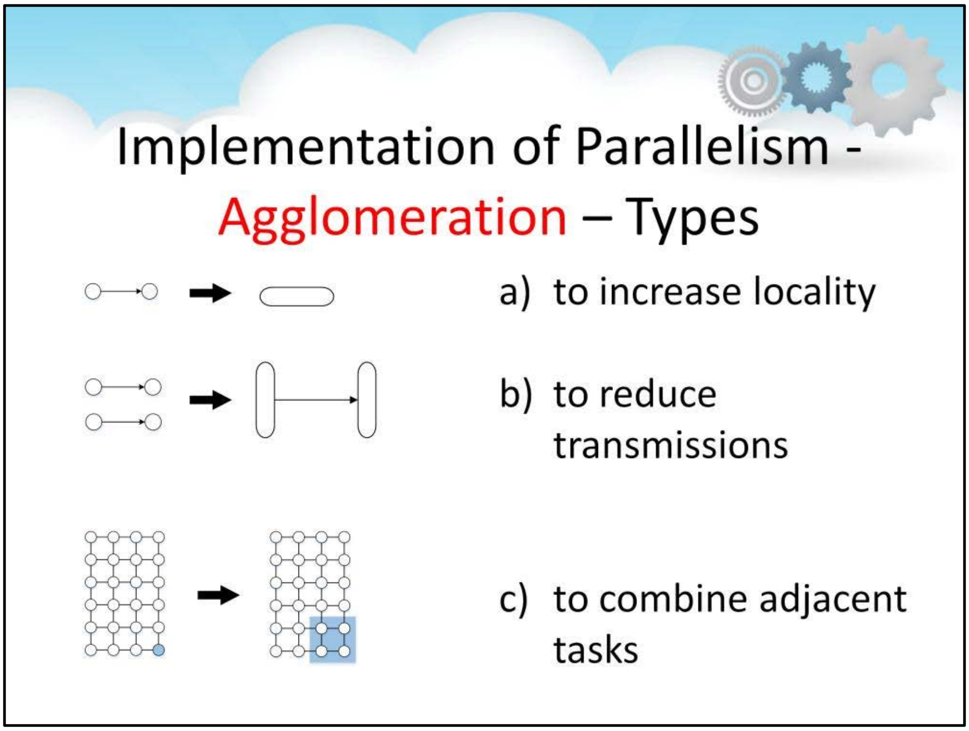
In the first two stages of the design process,
the computation is partitioned to maximize parallelism,
and communication between tasks is introduced so that tasks have the data they need.
The resulting algorithm is still an abstraction,
because it is not designed to execute on any particular parallel computer.
It may also be very inefficient, especially
if there are many more tasks than processors on the target computer.

This is because:

- Creating a task (process) typically uses overhead,
  and scheduling tasks on the processor uses overhead.
- The communication between tasks on the same processor adds artificial overhead,
  because if these tasks were combined into a single task,
  that communication would not exist.

Even if there are as many processors as tasks,
there may be inefficiencies in the design due to the inter-task communication.

In the third stage, agglomeration,
decisions made in the partitioning and communication phases are revisited.



Implementation of Parallelism - Agglomeration – Types

a) to increase locality

b) to reduce transmissions

c) to combine adjacent tasks

One way that performance is improved is by increase of granularity, that is by lowering communication overhead.

a) to **increase locality**

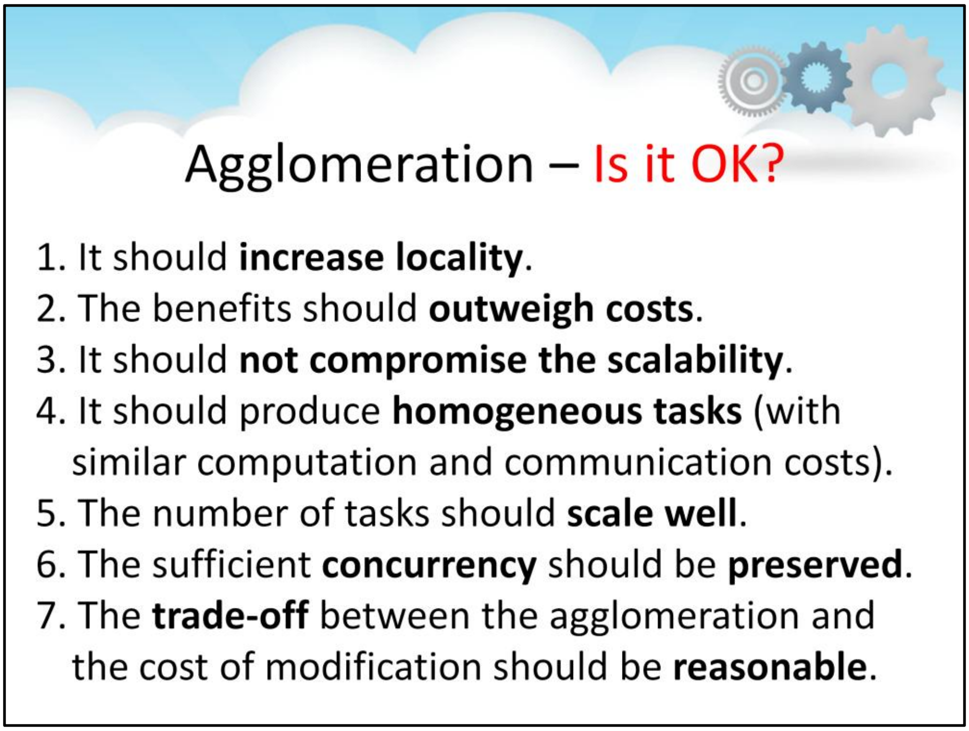
When two tasks that exchange data with each other are combined into a single task, the data that was exchanged through a channel is part of a single task and that channel and the overhead are removed. This is called increasing locality. Figure A shows how two tasks can be combined into one to increase locality.

b) **to reduce transmissions**

A second way to reduce communication overhead is by combining groups of tasks that all send and groups of tasks that all receive data from each other. The cost of sending a message has two components, the initial start-up time, called the latency, which is independent of how large the message is, and the transmission time, which is a function of the number of bytes sent. The transmission time is not reduced, but we cut the total latency in half. Figure B illustrates this type of agglomeration.

c) **to combine adjacent tasks**

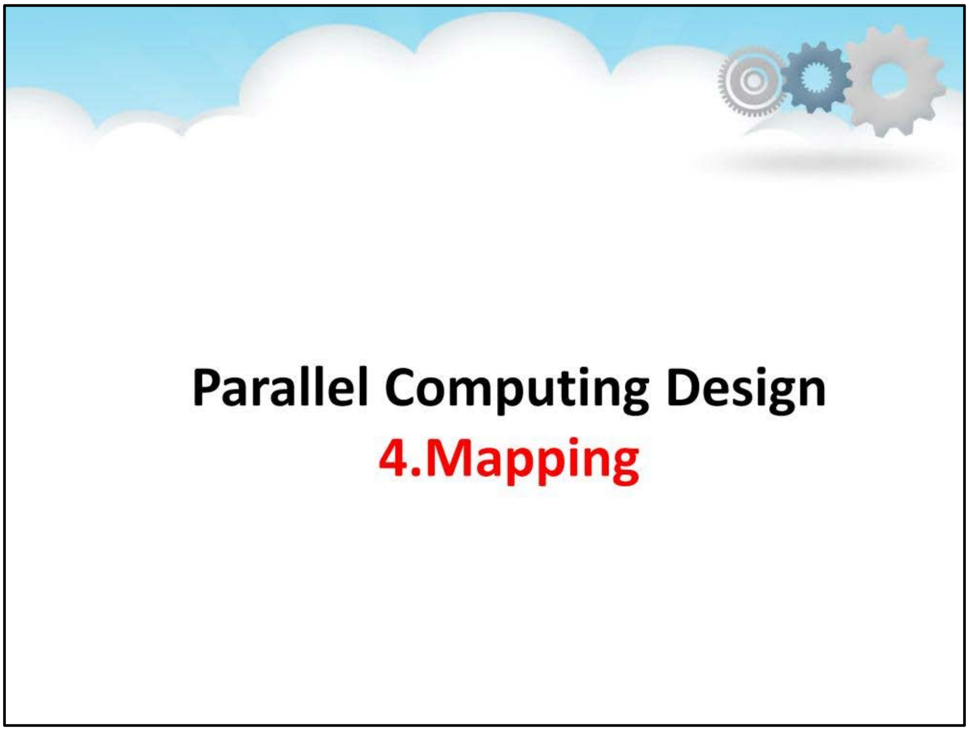
A third way to agglomerate is to combine adjacent tasks without reducing the dimensionality of the solution, but to increase granularity. Figure C shows how a 6*4 matrix that has been partitioned so that each task has one matrix element can be agglomerated by assigning four elements to each task. If the target architecture had only six processors, this might be a good solution.



Agglomeration – Is it OK?

1. It should **increase locality**.
2. The benefits should **outweigh costs**.
3. It should **not compromise the scalability**.
4. It should produce **homogeneous tasks** (with similar computation and communication costs).
5. The number of tasks should **scale well**.
6. The sufficient **concurrency** should be **preserved**.
7. The **trade-off** between the agglomeration and the cost of modification should be **reasonable**.

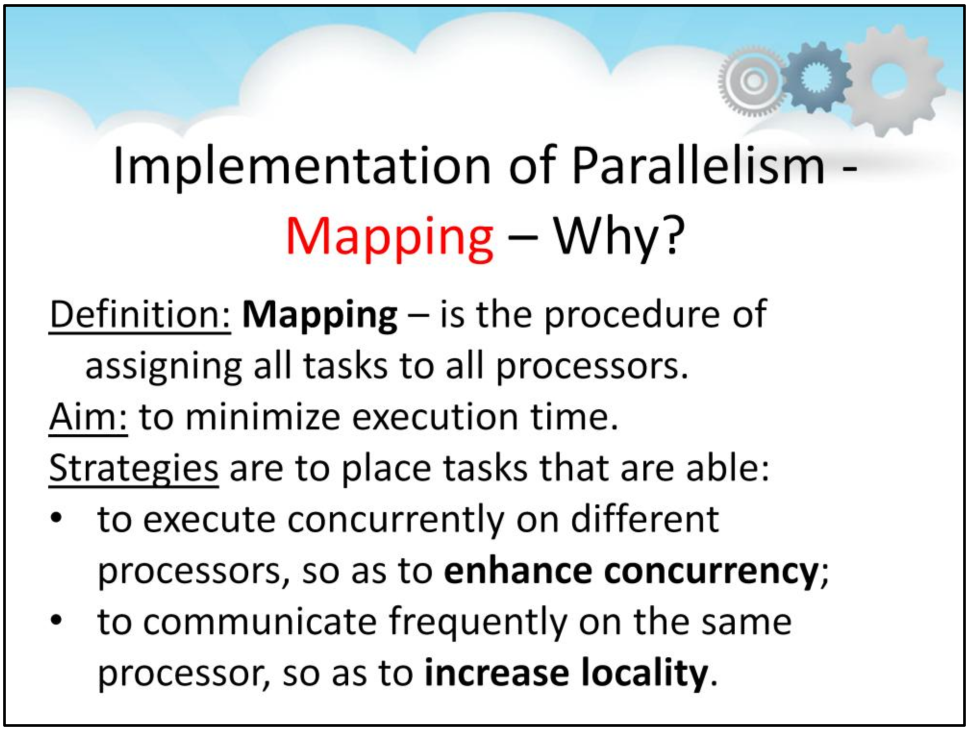
You should evaluate how well your agglomeration by the following criteria:
1. Agglomeration should reduce communication costs by increasing locality.
2. If agglomeration has replicated computation,
the benefits of this replication should outweigh its costs,
for a range of problem sizes and processor counts.
3. If agglomeration replicates data, it should not compromise the scalability
of the algorithm by restricting the range of problem sizes or processors that it can address.
4. Agglomeration should produce tasks with similar computation and communication costs.
5. The number of tasks can still scale with problem size.
6. There is sufficient concurrency for current and future target computers.
7. The trade-off between the chosen agglomeration and
the cost of modification to existing sequential code is reasonable.



Parallel Computing Design

4.Mapping

Mapping



Implementation of Parallelism - Mapping – Why?

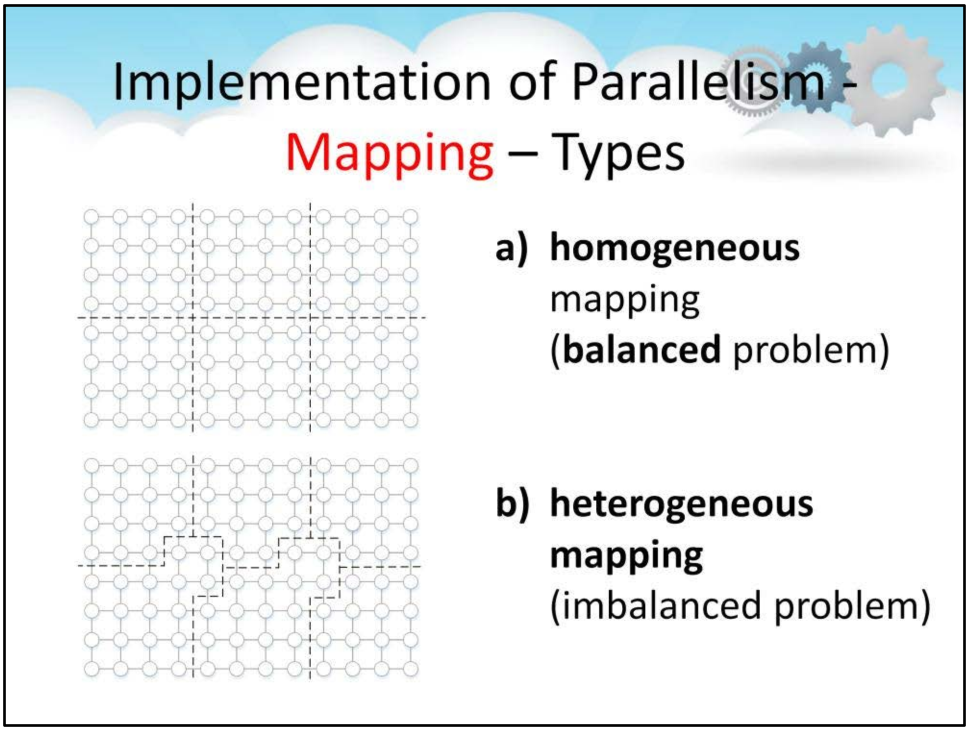
Definition: **Mapping** – is the procedure of assigning all tasks to all processors.

Aim: to minimize execution time.

Strategies are to place tasks that are able:

- to execute concurrently on different processors, so as to **enhance concurrency**;
- to communicate frequently on the same processor, so as to **increase locality**.

Mapping, the final stage of Foster's methodology, is the procedure of assigning each task to a processor. Of course, this mapping problem does not arise on uniprocessors or on shared-memory computers whose operating systems provide automatic task scheduling. We assume here that the target architecture is a distributed-memory parallel computer. The goal in developing mapping algorithms is to minimize total execution time. This is usually achieved by minimizing interprocessor communication and maximizing processor utilization. **Processor utilization** is the average percentage of time during which the computer's processors are actively executing code necessary to solve the problem. It is maximized when the computation is balanced evenly, because if there are processors that are idle while others are busy, then it would suggest that a different design might be able to off load the work being done by the busier ones onto the idle ones, making the total computation time smaller. This is not necessarily the case, but it is the general idea. **Two different strategies to minimize execution time are shown in Figure.** Unfortunately, sometimes these two strategies will be in conflict, in which case the design will involve trade-offs. In addition, resource limitations may restrict the number of tasks that can be placed on a single processor.



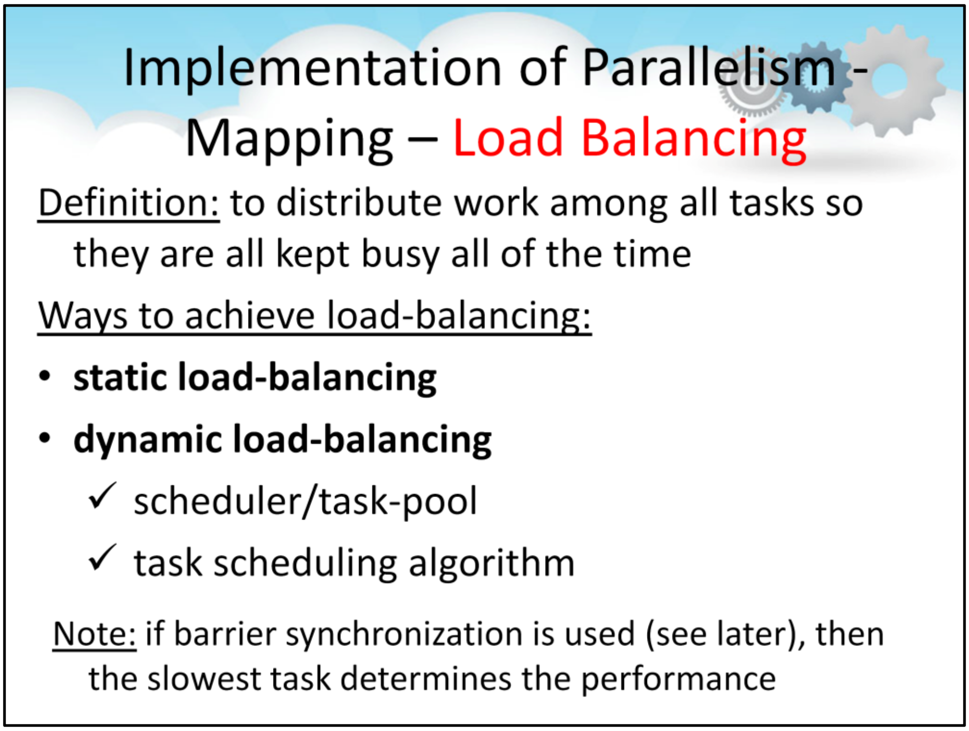
Implementation of Parallelism - Mapping – Types

a) **homogeneous** mapping (**balanced** problem)

b) **heterogeneous mapping** (imbalanced problem)

a) **homogeneous** mapping
usually after a domain decomposition,
there is a fixed number of tasks with similar size and similar computing steps.
A natural strategy is to divide the tasks among the processors in such a way
that each processor has the same number of tasks,
and such that adjacent tasks are assigned to the same processor **as shown in Figure A**.

b) **heterogeneous** mapping
sometimes, some of the tasks might have a different load than the others,
such as those perhaps in the corners of sub-grids.
This could create a **load imbalance**,
and a **heterogeneous mapping** might be optimal (such **as shown in Figure B**).



Implementation of Parallelism - Mapping – Load Balancing

<u>Definition:</u> to distribute work among all tasks so they are all kept busy all of the time

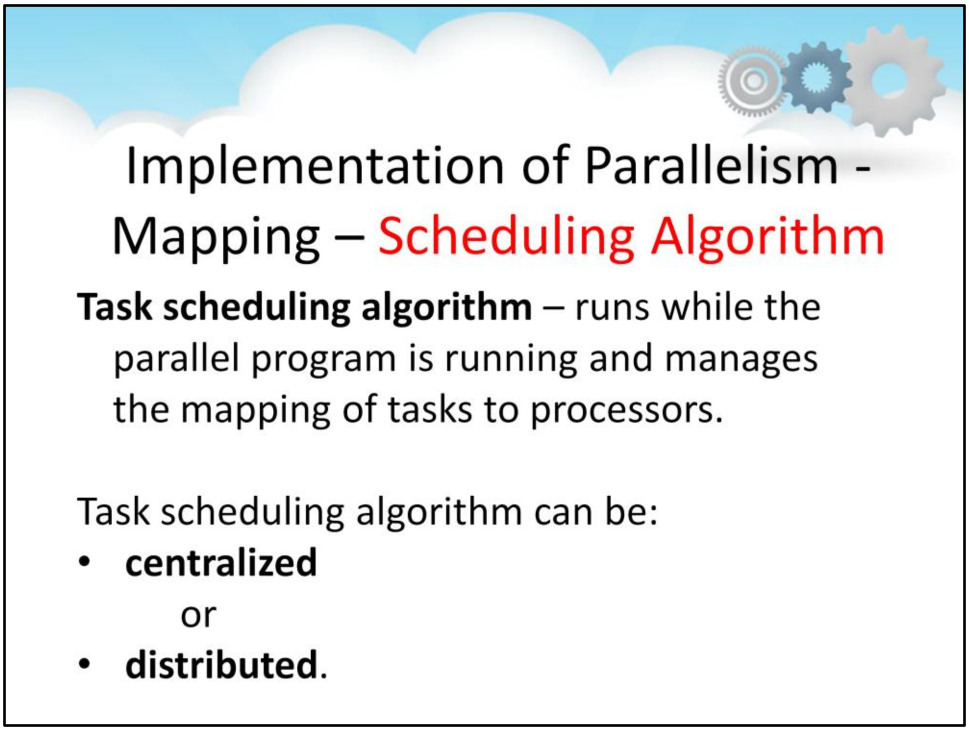
<u>Ways to achieve load-balancing:</u>

- **static load-balancing**
- **dynamic load-balancing**
  - ✓ scheduler/task-pool
  - ✓ task scheduling algorithm

<u>Note:</u> if barrier synchronization is used (see later), then the slowest task determines the performance

In the previous examples, the communication pattern was regular,
so the solutions were regular.

When the communication pattern is not regular, but is **known in advance**,
a **static load-balancing algorithm** can be used.
It can be applied at compile-time to determine a mapping strategy.
But sometimes the number of tasks is **NOT known in advance**,
or if it is, the communication requirements are not.
In either of these cases, **a dynamic load-balancing algorithm** must be used.
A dynamic load-balancing algorithm analyzes the set of running tasks
and generates a new mapping of tasks to processors.



Implementation of Parallelism - Mapping – Scheduling Algorithm

Task scheduling algorithm – runs while the parallel program is running and manages the mapping of tasks to processors.

Task scheduling algorithm can be:

- **centralized**

  or

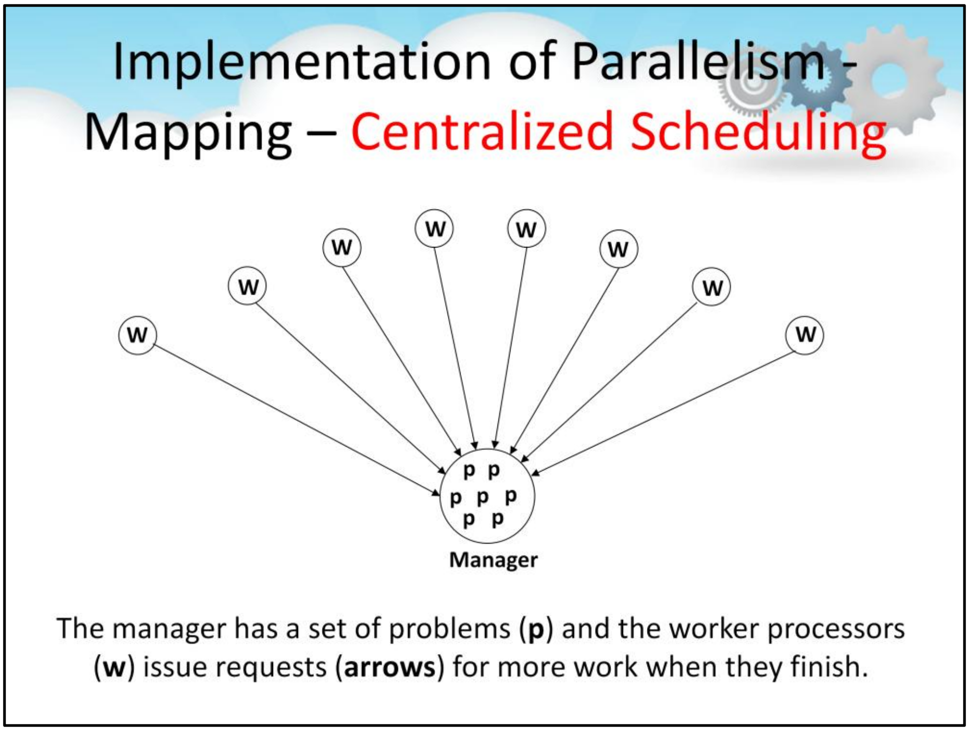
- **distributed**.

Sometimes the tasks are short-lived;
they are created on the fly to solve small problems and then they are destroyed,
and they do not communicate with each other.

In this case, a **task scheduling algorithm** runs while the parallel program is running
and manages the mapping of tasks to processors.

Such an algorithm can be
- centralized
or
- distributed.

In general, task scheduling algorithms can be used
when a functional decomposition yields many tasks, each with weak locality requirements.



Implementation of Parallelism - Mapping – Centralized Scheduling

The manager has a set of problems (**p**) and the worker processors (**w**) issue requests (**arrows**) for more work when they finish.

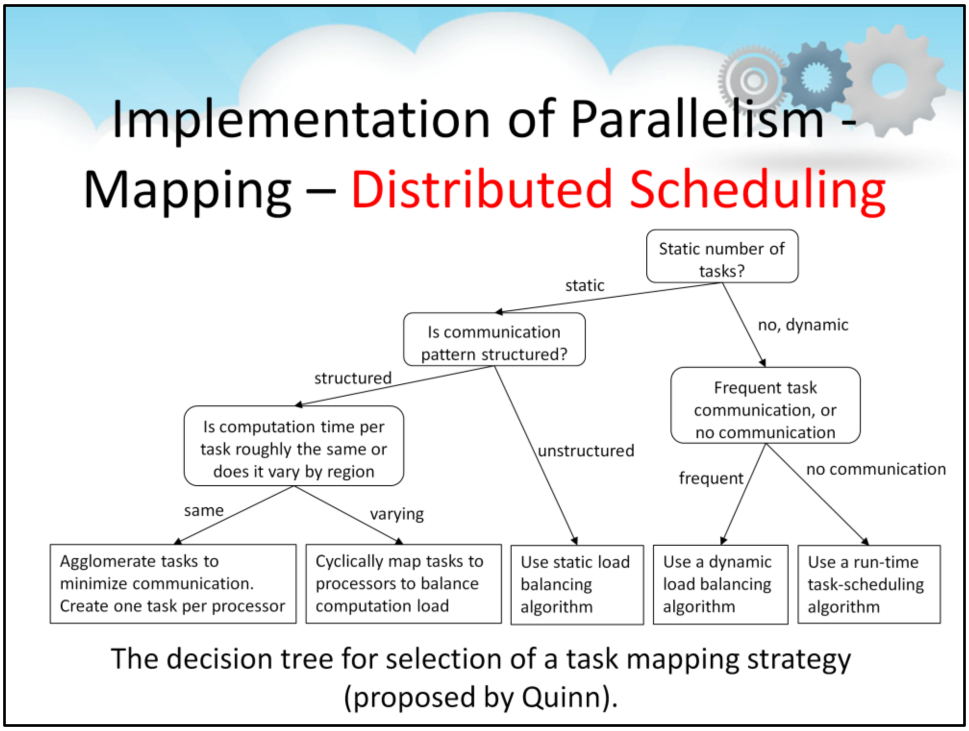
In a centralized task scheduling algorithm:

- one processor becomes a **manager**

and

- the remaining processors are used to run **worker** tasks.

A pool of problems is maintained, into which new problems are placed and from which problems are taken for allocation to the worker processors.
Each worker processor runs to solve a problem,
and when it is finished, it returns its results to the manager and requests a new problem.
This Figure illustrates the idea.



Implementation of Parallelism - Mapping – Distributed Scheduling

Static number of tasks?

static

no, dynamic

Is communication pattern structured?

structured

unstructured

Frequent task communication, or no communication

frequent

no communication

Is computation time per task roughly the same or does it vary by region

same

varying

Agglomerate tasks to minimize communication. Create one task per processor

Cyclically map tasks to processors to balance computation load

Use static load balancing algorithm

Use a dynamic load balancing algorithm

Use a run-time task-scheduling algorithm

The decision tree for selection of a task mapping strategy (proposed by Quinn).

One problem with the centralized scheduler is that the manager becomes a bottleneck.

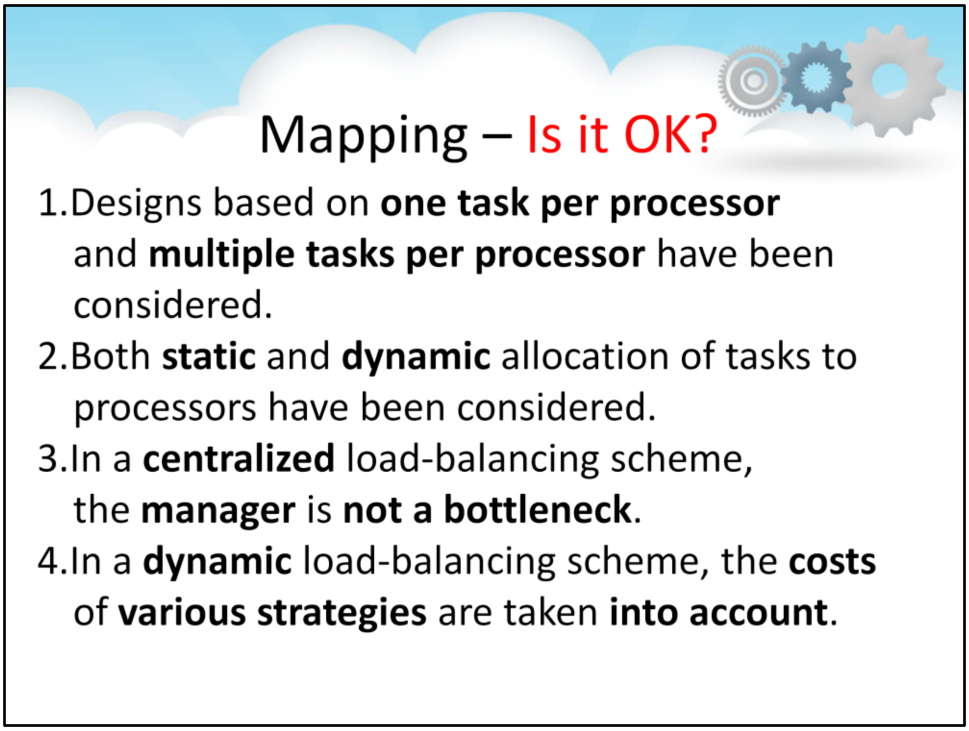
In a **distributed scheduling algorithm**, there is no manager.

Instead, a separate task pool is maintained on each processor,
and idle workers request problems from other processors.

The task pool is, in effect, a distributed data structure
that is accessed by the different tasks asynchronously.

This is a more scalable solution than a centralized one.

Quinn presents a **decision tree** (shown in this Figure)
for deciding on how to map tasks to processors.



Mapping – Is it OK?

1. Designs based on **one task per processor** and **multiple tasks per processor** have been considered.
2. Both **static** and **dynamic** allocation of tasks to processors have been considered.
3. In a **centralized** load-balancing scheme, the **manager** is **not a bottleneck**.
4. In a **dynamic** load-balancing scheme, the **costs** of **various strategies** are taken **into account**.

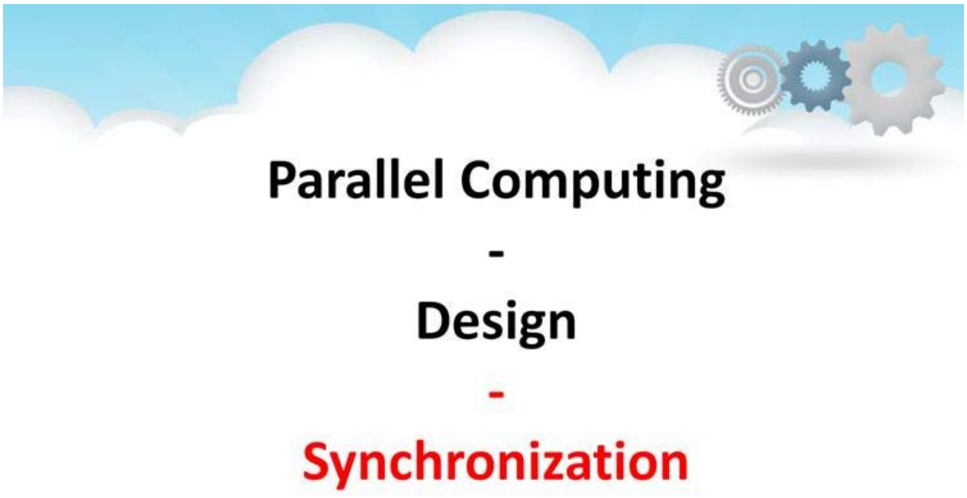
The following checklist can be used for informal evaluation of the mapping design:

1.Designs based on one task per processor
and multiple tasks per processor have been considered.

2.Both static and dynamic allocation of tasks to processors have been considered.

3.If a centralized load-balancing scheme is chosen,
you have made sure that the manager will not become a bottleneck.

4.If a dynamic load-balancing scheme is chosen,
you have evaluated the relative costs of different strategies,
and should account for the implementation costs in the analysis.



Parallel Computing

-

Design

-

Synchronization

Synchronization

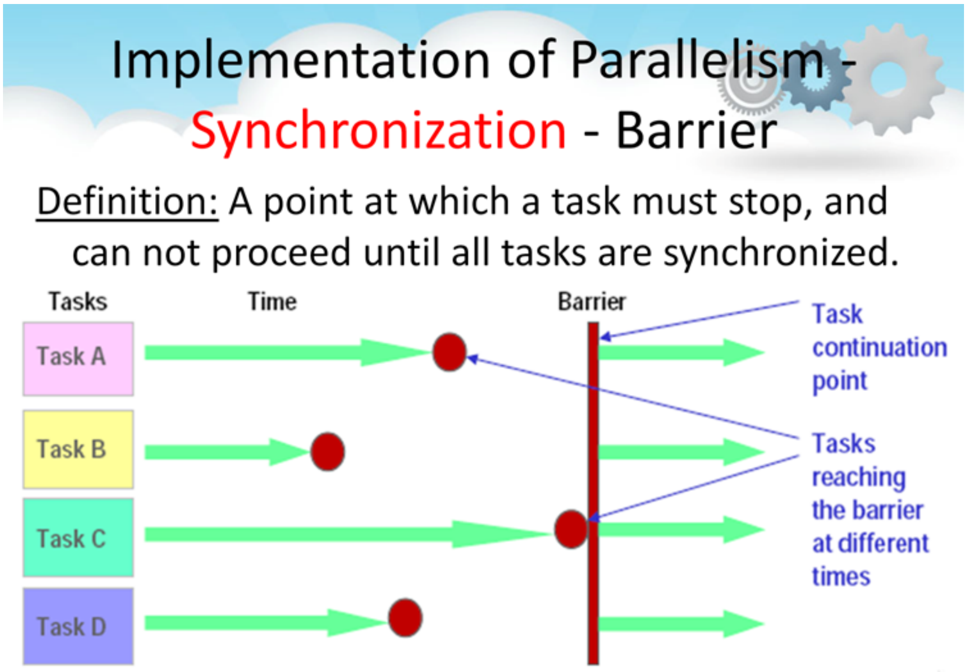
Implementation of Parallelism - Synchronization

Definition:

Reaching an agreement between simultaneous processes/tasks as to the sequence of steps to complete an action

Ways of Synchronization:

- Barriers
- Locks/ Semaphores
- Synchronous Communication Operations



Implementation of Parallelism - Synchronization - Barrier

Definition: A point at which a task must stop, and can not proceed until all tasks are synchronized.

Tasks

Time

Barrier

Task A

Task B

Task C

Task D

Task continuation point

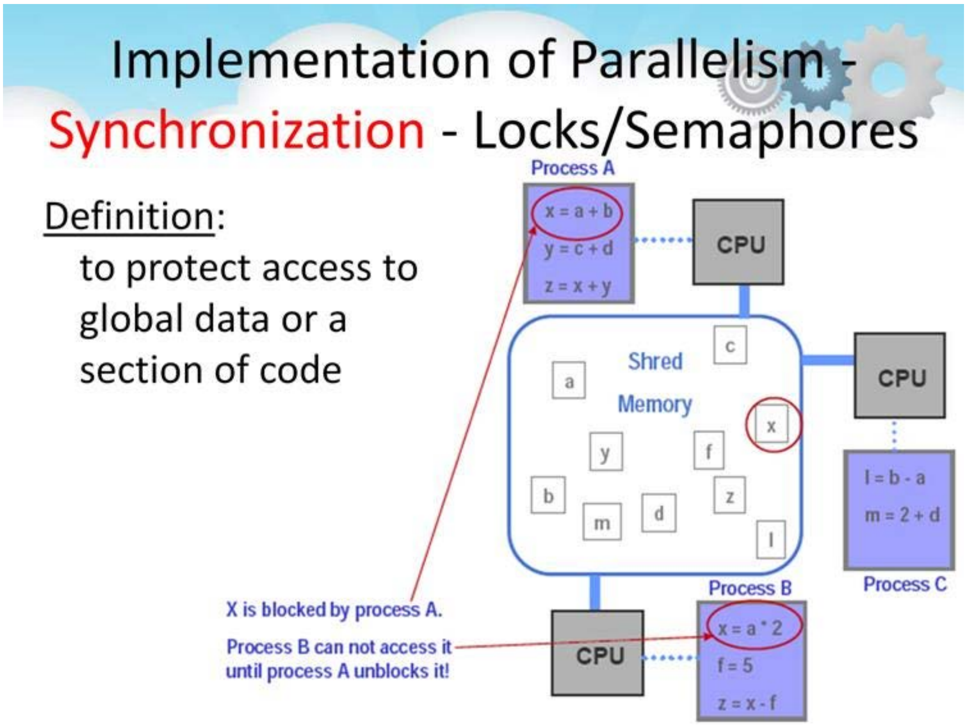
Tasks reaching the barrier at different times

Each task keeps working until reaching a barrier.

Then, it stops and keeps waiting for the last task to reach the barrier.

When last task reaches the barrier, all tasks are synchronized.

From this point, tasks continues their work.



Implementation of Parallelism - Synchronization - Locks/Semaphores

Definition:

to protect access to global data or a section of code

X is blocked by process A.

Process B can not access it until process A unblocks it!

Only one task at a time may access the lock/semaphore.

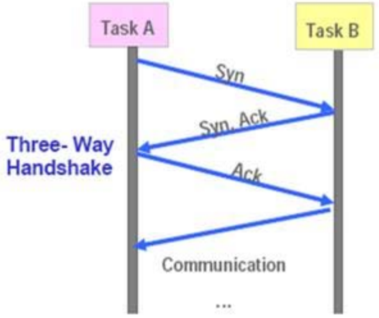
The first task accesses the lock sets it to be locked and releases it when it’s done with it.

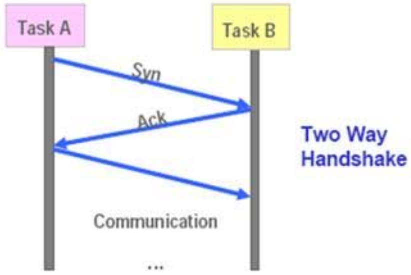
When other tasks try to access the lock they fail until the task that owns the lock releases it.

Can be blocking or non-blocking

Implementation of Parallelism - Synchronous Communication Oper-ns

Definition: Coordination is required between the task that is performing an operation and the other tasks performing the communication





Parallel Computing

-

Performance

Parallel Computing Performance

It is as important to know costs and benefits of parallelization.

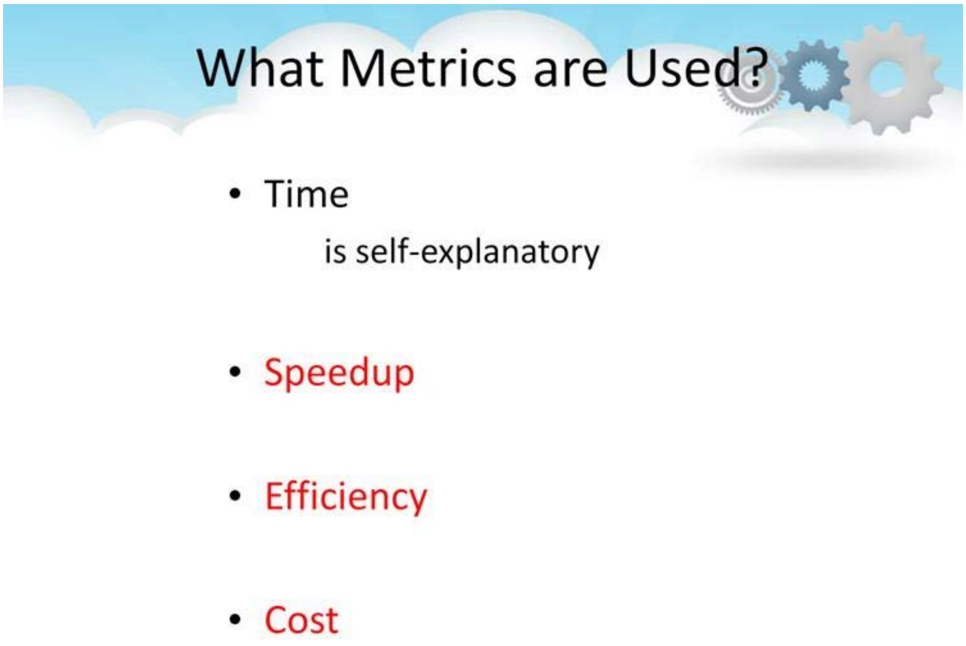
The purpose of this section is to consider the means (metrics):

- to determine how much faster a parallel algorithm performs than a sequential algorithm;

- to predict the performance of a parallel algorithm;

- to decide whether there are inherent limitations that prevent a parallel algorithm from performing, and what they are;

- to determine how efficient a parallel algorithm can be as the problem size and number of available processors increases.



What Metrics are Used?

- Time

  is self-explanatory

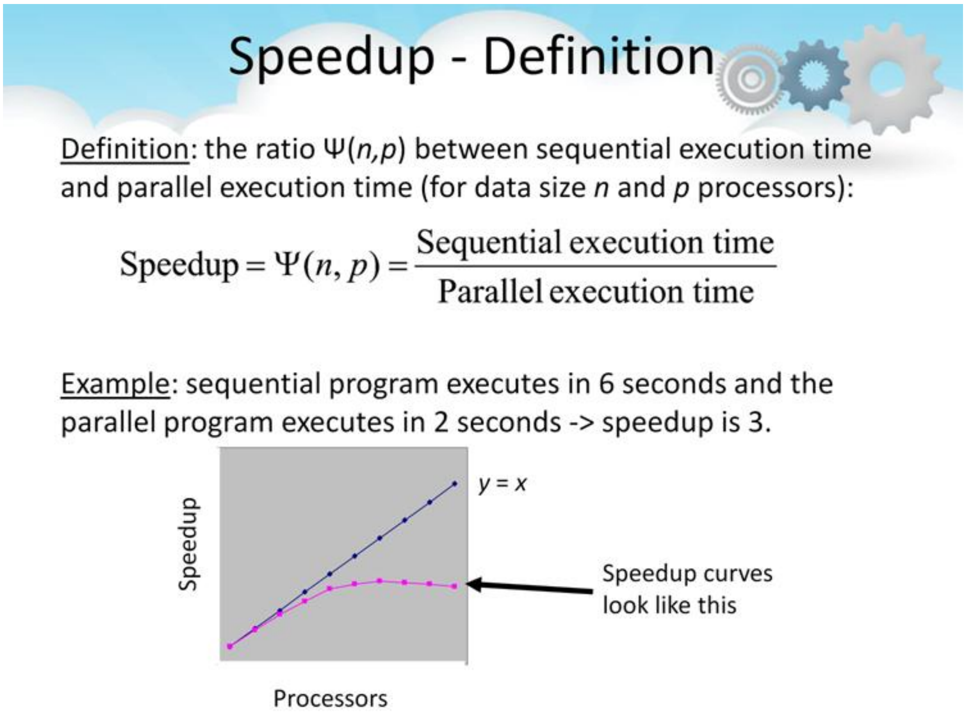
- Speedup

- Efficiency

- Cost

Metrics

The following metrics are usually used for estimation of performance of parallel programs…



Speedup - Definition

Definition: the ratio $\Psi(n,p)$ between sequential execution time and parallel execution time (for data size *n* and *p* processors):

$$\text{Speedup} = \Psi(n,p) = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

Example: sequential program executes in 6 seconds and the parallel program executes in 2 seconds -> speedup is 3.

Speedup curves look like this

We use the term “speedup” to indicate how much faster the parallel program is than the sequential program.

Question: What does it mean if speedup > 1?

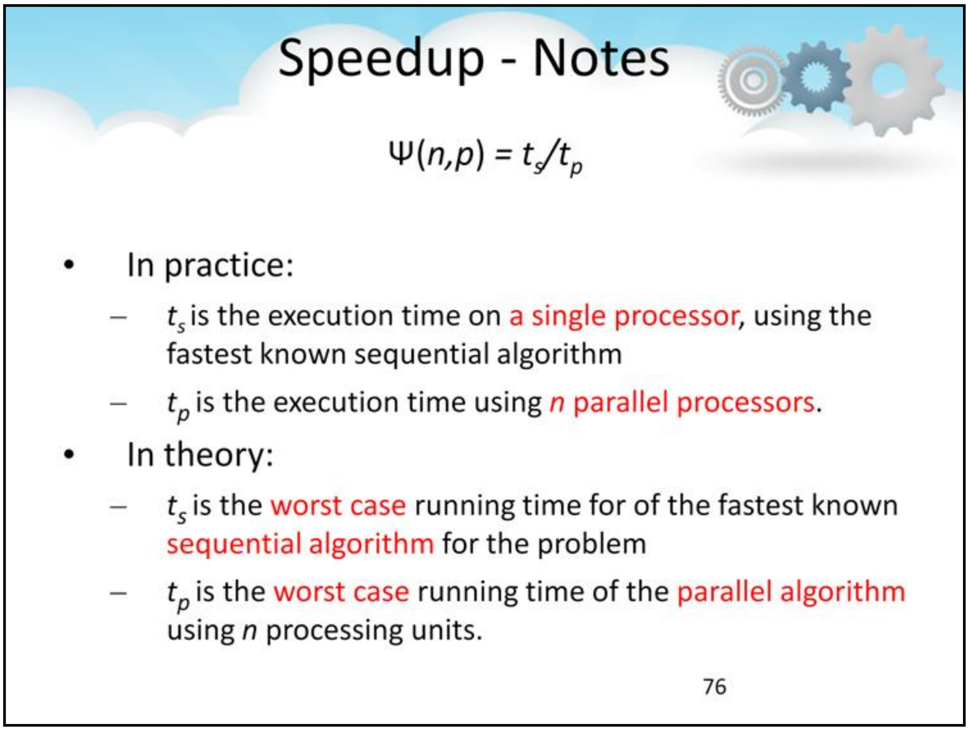
Answer: It means the parallel program is executing faster than the sequential program.

Question: What does it mean if speedup < 1?

Answer: It means the parallel program is executing slower than the sequential program.

Question: For a given problem size, speedup often increases as the number of processors increases, but it always “elbows out” and starts to decline. Why?

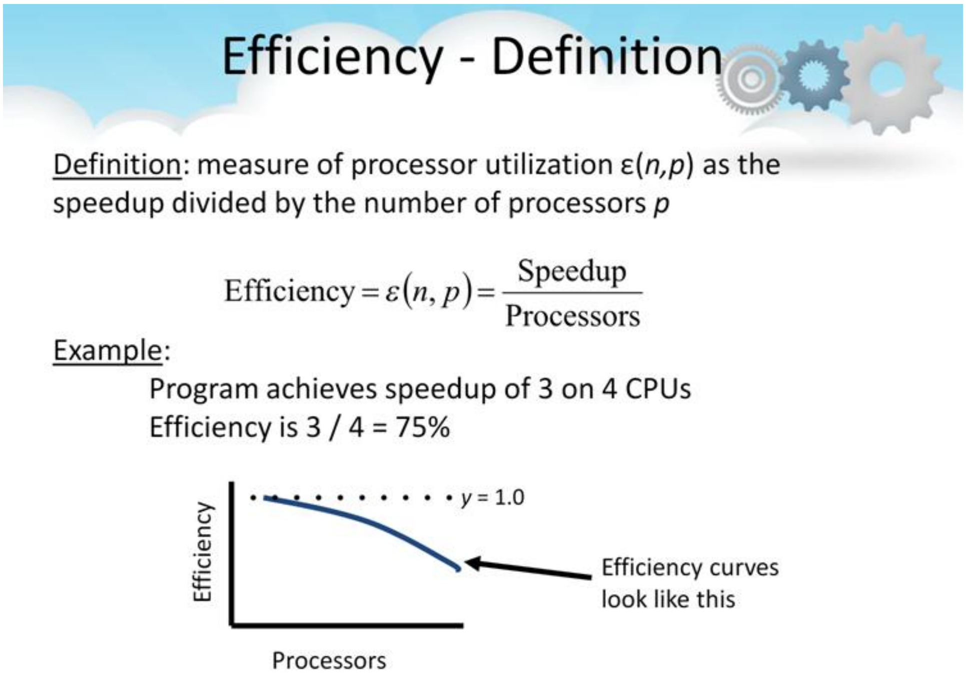
Answer: The overhead costs associated with creating, managing, and terminating parallel threads increases as the number of threads increases. The amount of work to be done by each thread keeps getting smaller as the number of threads increases. At some point the overhead time associated with a new thread is greater than the time savings achieved by adding a new thread. At that point the parallel execution time will increase, and the speedup curve will drop.



Speedup - Notes

$$\Psi(n,p) = t_s/t_p$$

- In practice:
  - t_s is the execution time on a single processor, using the fastest known sequential algorithm
  - t_p is the execution time using *n* parallel processors.
- In theory:
  - t_s is the worst case running time for of the fastest known sequential algorithm for the problem
  - t_p is the worst case running time of the parallel algorithm using *n* processing units.



Efficiency - Definition

Definition: measure of processor utilization ε(*n*,*p*) as the speedup divided by the number of processors *p*

$$\text{Efficiency} = \varepsilon(n, p) = \frac{\text{Speedup}}{\text{Processors}}$$

Example:

Program achieves speedup of 3 on 4 CPUs
Efficiency is 3 / 4 = 75%

We use the word “efficiency” to indicate how well we are using the available CPU resources. An efficiency of 100% means that every one of the processors in a parallel computation is spending all its time doing useful work, compared to the computational rate of the sequential program.

FOR A GIVEN PROBLEM SIZE, EFFICIENCY IS ALMOST ALWAYS A DECREASING FUNCTION OF THE NUMBER OF PROCESSORS USED TO SOLVE THE PROBLEM.

Question: Which is better---an efficiency of 80% on two processors, or an efficiency of 50% on four processors?

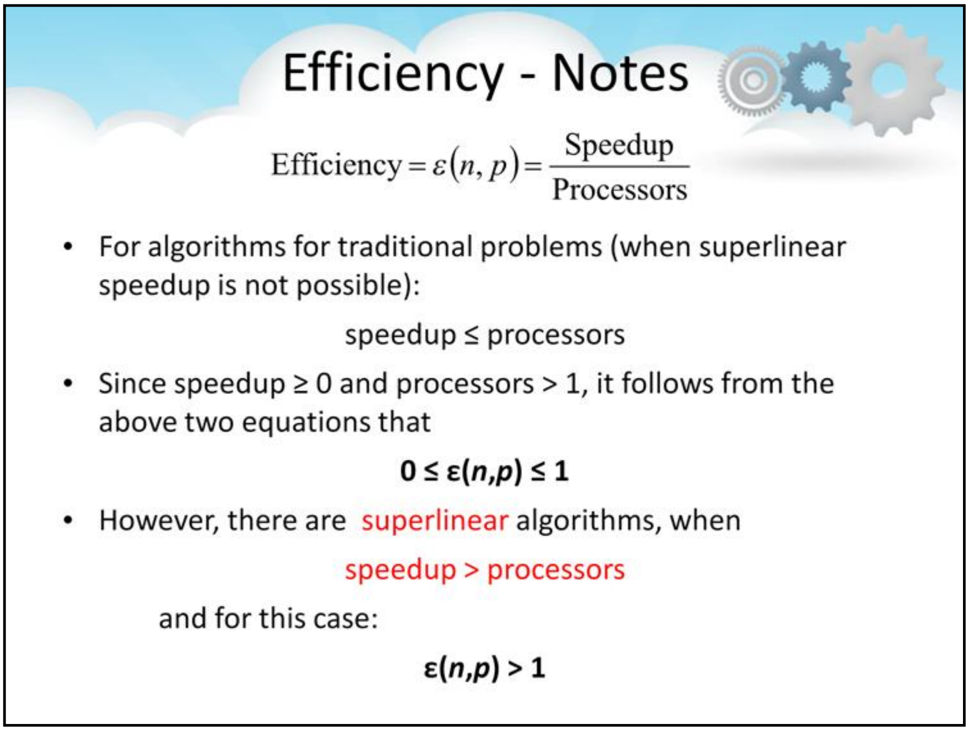
Answer: Efficiency = Speedup / Number of Processors

Speedup = Efficiency * Number of Processors ⇒

An efficiency of 80% on two processors means a speedup of 1.6.

An efficiency of 50% on four processors means a speedup of 2.0.

An efficiency of 50% on four processors is better because the speedup is higher.



Efficiency - Notes

$$\text{Efficiency} = \varepsilon(n, p) = \frac{\text{Speedup}}{\text{Processors}}$$

- For algorithms for traditional problems (when superlinear speedup is not possible):

  speedup ≤ processors

- Since speedup ≥ 0 and processors > 1, it follows from the above two equations that

  0 ≤ ε(*n*,*p*) ≤ 1

- However, there are superlinear algorithms, when

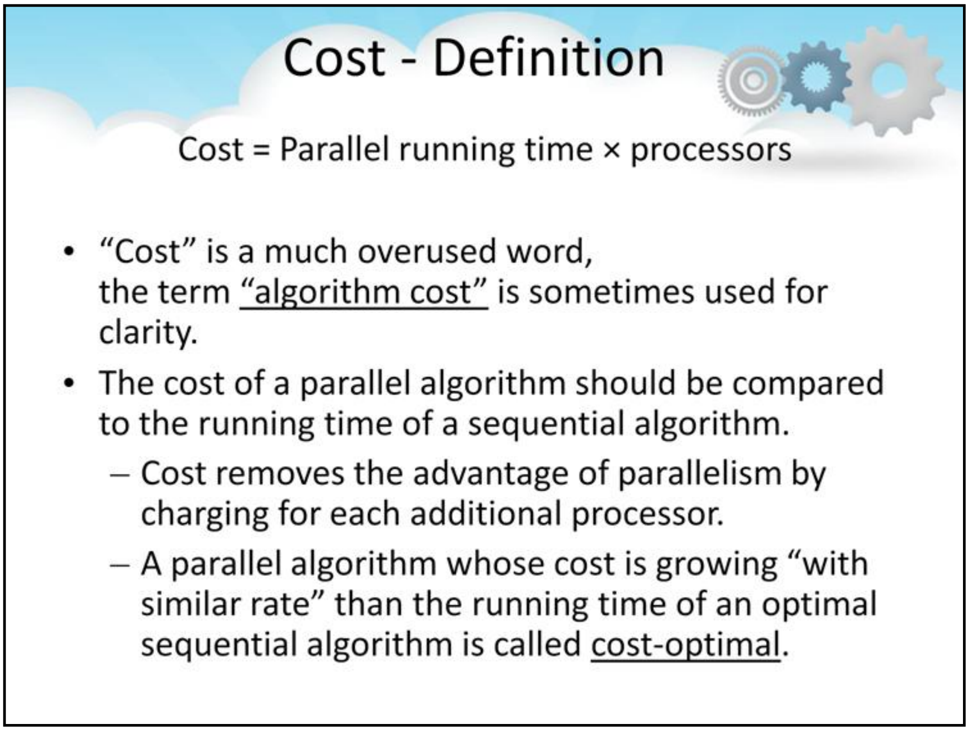
  speedup > processors

  and for this case:

  ε(*n*,*p*) > 1

This is true if the four processors have nothing else to do. But what if we have two jobs to run, and each job has an efficiency of 80% on two processors and an efficiency of 50% on four processors? In that case, we would make more efficient use of the processors if we ran both jobs simultaneously and gave each job two processors. That would increase the throughput of the system (i.e., the rate at which jobs are completed).

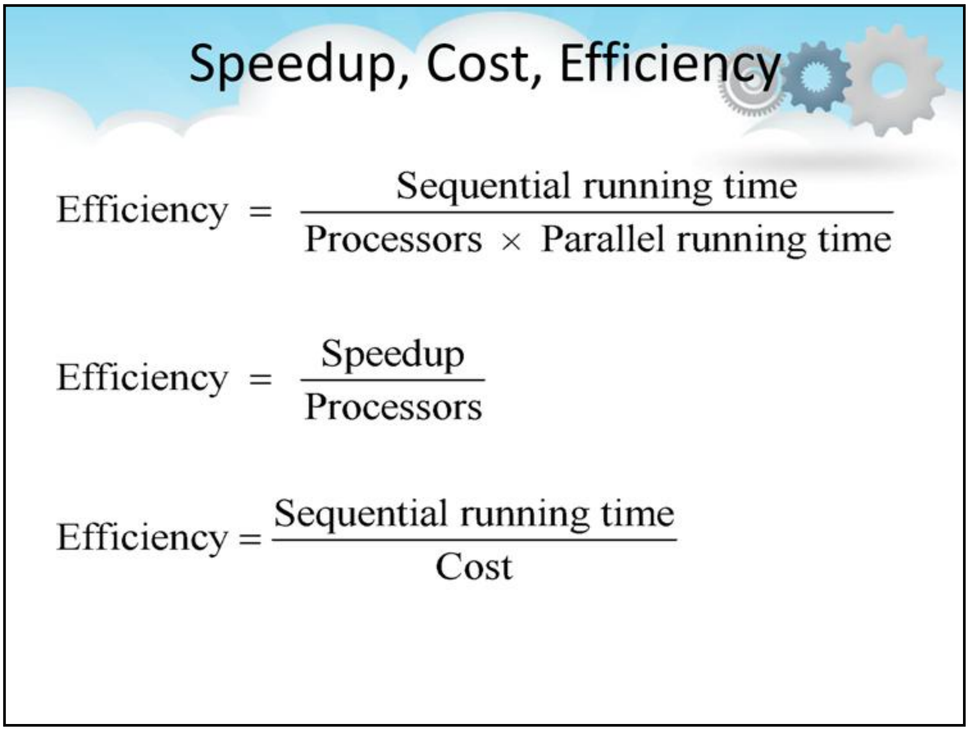
How does this relate to parallel processing? After all, we said at the beginning of the class that our focus is on getting particular tasks to complete faster. It's relevant if our problem has a potential task decomposition, and within each task decomposition there is a potential domain decomposition. It may be better to have two tasks executing simultaneously, each with two processors, then to first execute one task on four processors and then execute the second task on four processors.



Cost - Definition

Cost = Parallel running time × processors

- “Cost” is a much overused word,
  the term <u>“algorithm cost”</u> is sometimes used for clarity.
- The cost of a parallel algorithm should be compared to the running time of a sequential algorithm.
  - Cost removes the advantage of parallelism by charging for each additional processor.
  - A parallel algorithm whose cost is growing “with similar rate” than the running time of an optimal sequential algorithm is called <u>cost-optimal</u>.

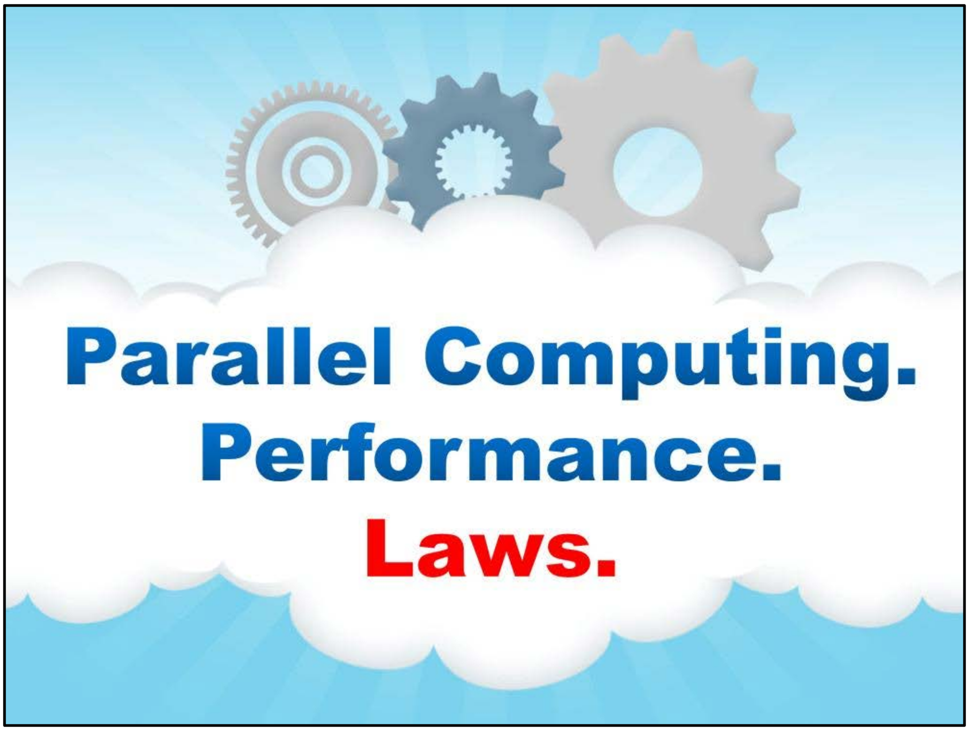


Speedup, Cost, Efficiency

$$\text{Efficiency} = \frac{\text{Sequential running time}}{\text{Processors} \times \text{Parallel running time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

$$\text{Efficiency} = \frac{\text{Sequential running time}}{\text{Cost}}$$



Parallel Computing.
Performance.
Laws.

Laws

It is as important to know costs and benefits of parallelization.

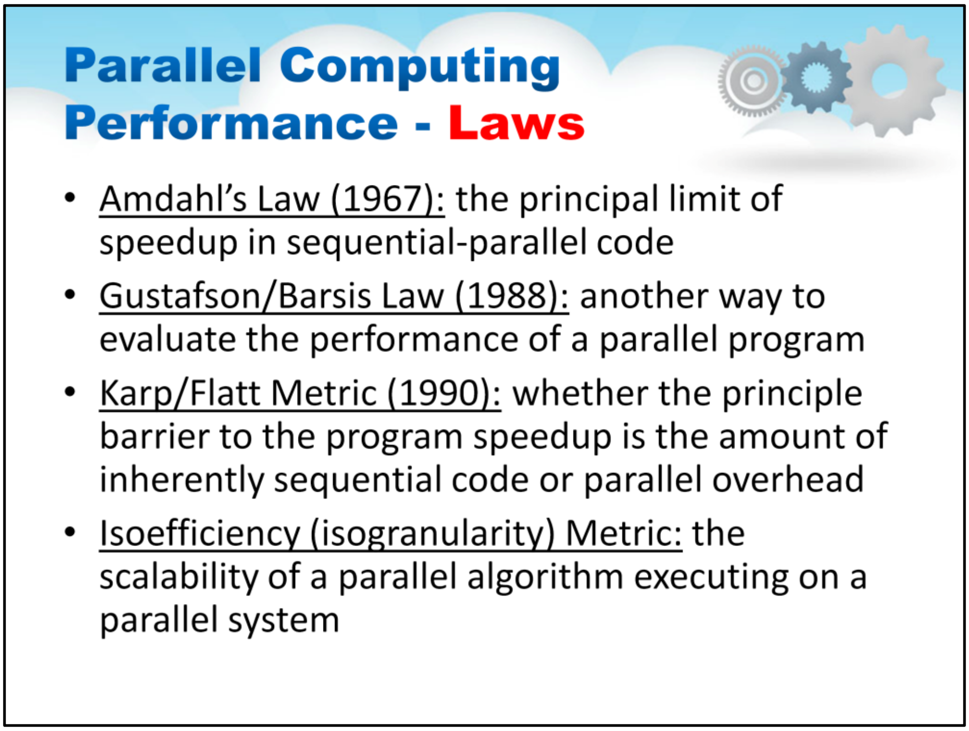
The purpose of this section is to consider the means (metrics):

- to determine how much faster a parallel algorithm performs than a sequential algorithm;

- to predict the performance of a parallel algorithm;

- to decide whether there are inherent limitations
  that prevent a parallel algorithm from performing, and what they are;

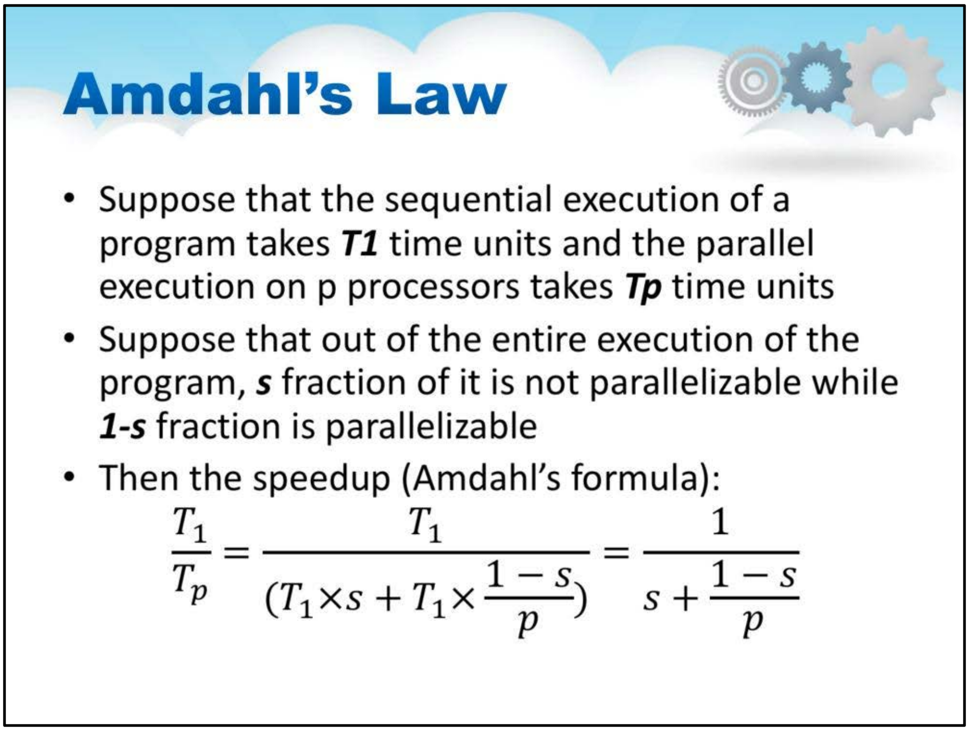
- to determine how efficient a parallel algorithm can be
  as the problem size and number of available processors increases.



Parallel Computing Performance - Laws

- Amdahl’s Law (1967): the principal limit of speedup in sequential-parallel code
- Gustafson/Barsis Law (1988): another way to evaluate the performance of a parallel program
- Karp/Flatt Metric (1990): whether the principle barrier to the program speedup is the amount of inherently sequential code or parallel overhead
- Isoefficiency (isogranularity) Metric: the scalability of a parallel algorithm executing on a parallel system

Today we will talk about Amdahl’s Law.

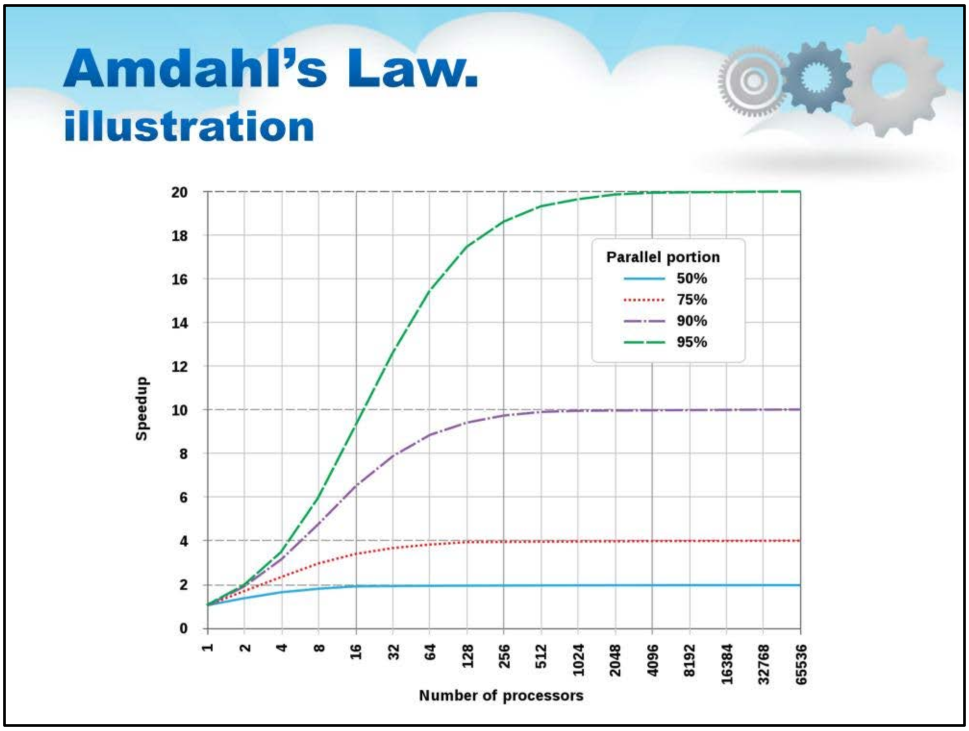


Amdahl's Law

- Suppose that the sequential execution of a program takes ***T1*** time units and the parallel execution on p processors takes ***Tp*** time units
- Suppose that out of the entire execution of the program, ***s*** fraction of it is not parallelizable while ***1-s*** fraction is parallelizable
- Then the speedup (Amdahl's formula):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$

Suppose that the sequential execution of a program takes T1 time units and the parallel execution on p processors takes Tp time units



Amdahl's Law.

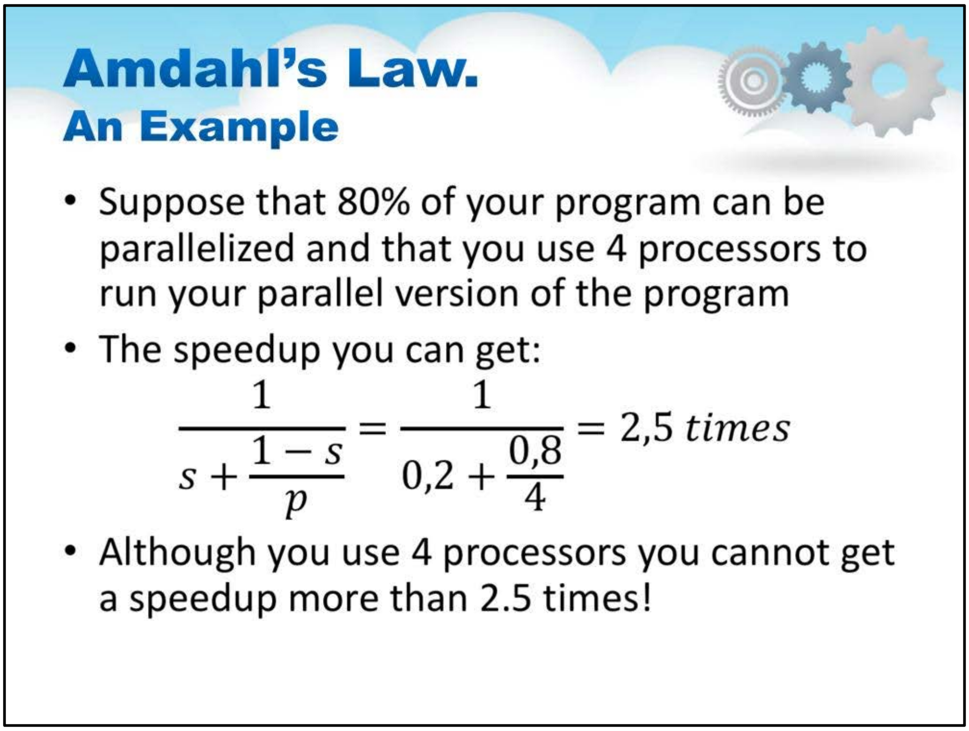
illustration

Amdahl's law is often used in parallel computing to predict the theoretical speedup when using multiple processors.

For example, if a program needs 20 hours using a single processor core, and a particular part of the program which takes one hour to execute cannot be parallelized, while the remaining 19 hours ($p = 0.95$) of execution time can be parallelized, then regardless of how many processors are devoted to a parallelized execution of this program, the minimum execution time cannot be less than that critical one hour.

Hence, the theoretical speedup is limited to at most 20 times ($1/(1 - p) = 20$).

For this reason, parallel computing with many processors is useful only for highly parallelizable programs.



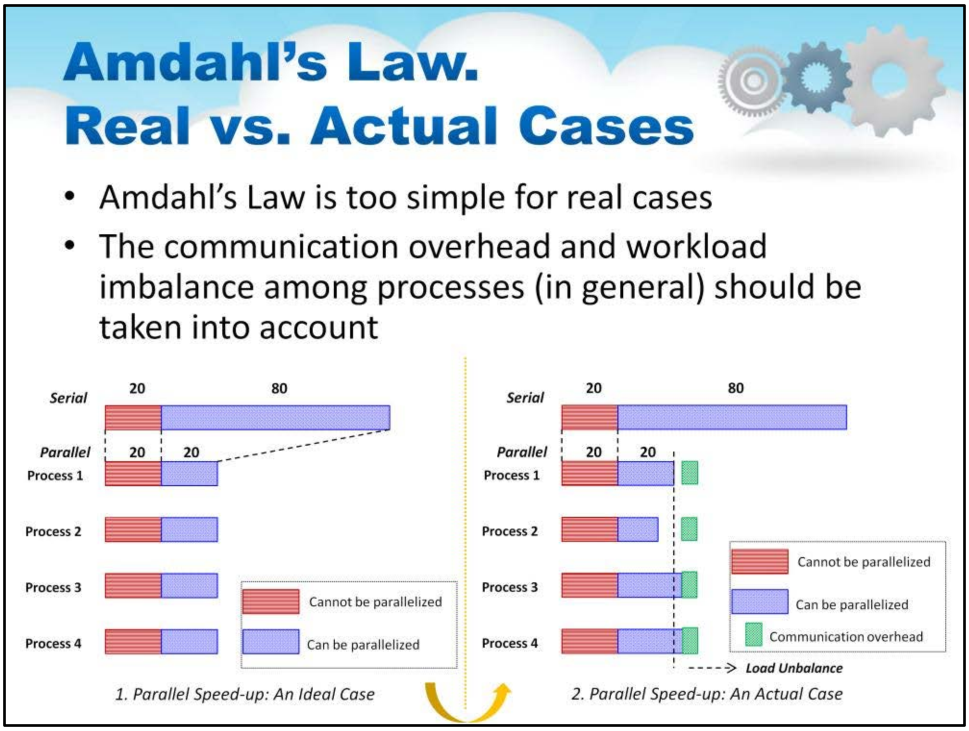
Amdahl's Law.
An Example

- Suppose that 80% of your program can be parallelized and that you use 4 processors to run your parallel version of the program
- The speedup you can get:

$$\frac{1}{s+\frac{1-s}{p}}=\frac{1}{0{,}2+\frac{0{,}8}{4}}=2{,}5\ times$$

- Although you use 4 processors you cannot get a speedup more than 2.5 times!

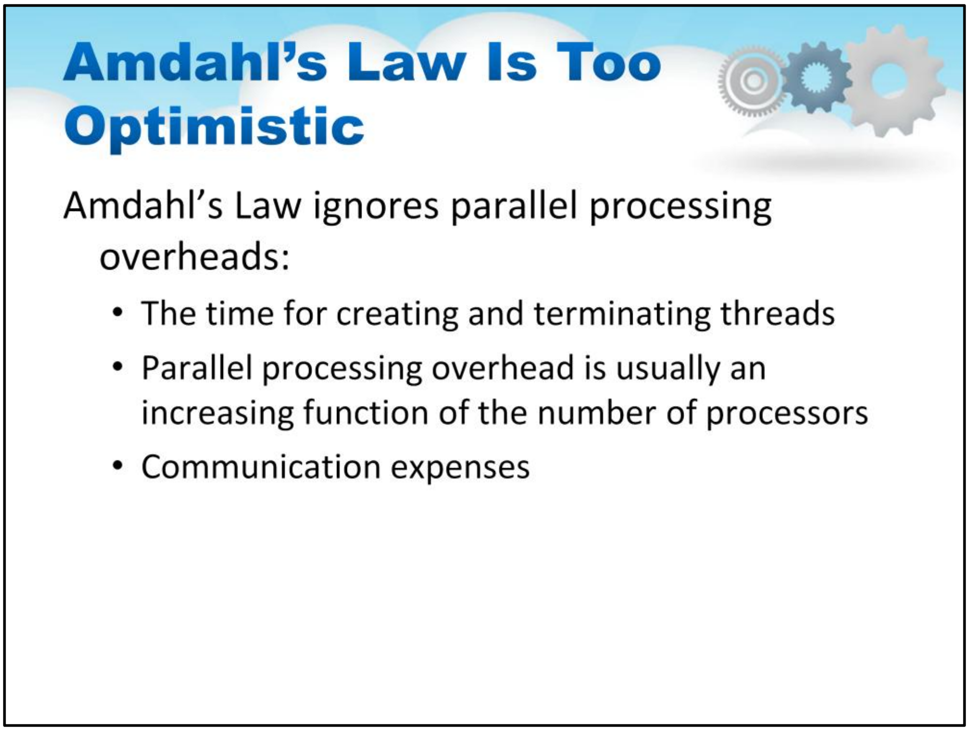
Suppose that 80% of your program can be parallelized and that you use 4 processors to run your parallel version of the program.
As a result you use 4 processors you cannot get a speedup more than 2.5 times!



Amdahl’s Law. Real vs. Actual Cases

- Amdahl’s Law is too simple for real cases
- The communication overhead and workload imbalance among processes (in general) should be taken into account

Amdahl’s Law is too simple for real cases. The communication overhead and workload imbalance among processes (in general) should be taken into account

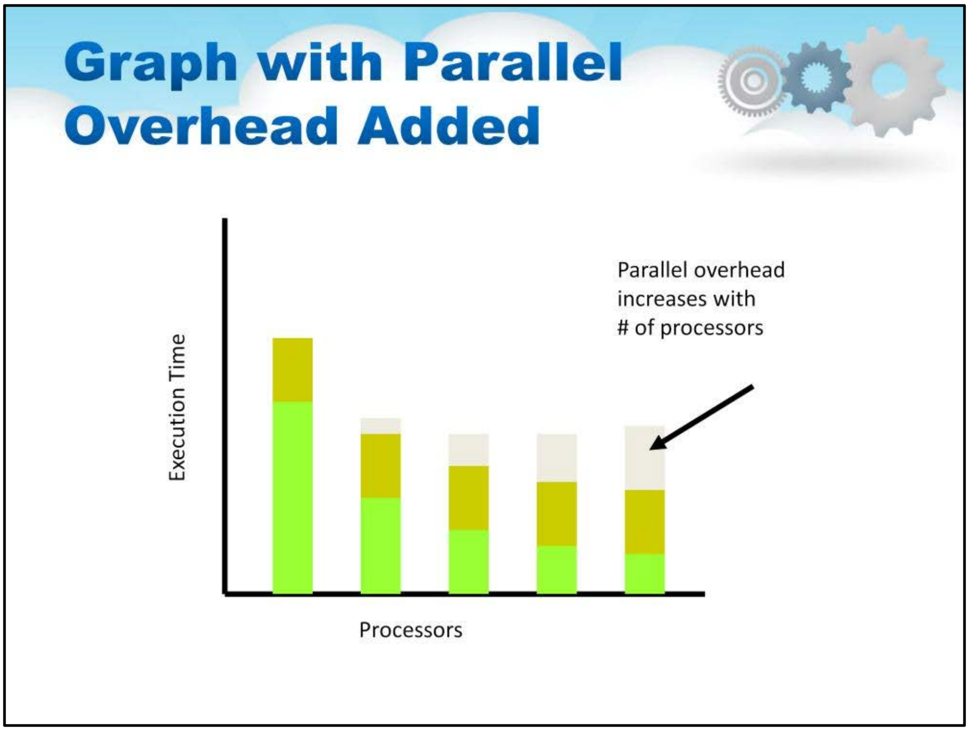


Amdahl’s Law Is Too Optimistic

Amdahl’s Law ignores parallel processing overheads:

- The time for creating and terminating threads
- Parallel processing overhead is usually an increasing function of the number of processors
- Communication expenses

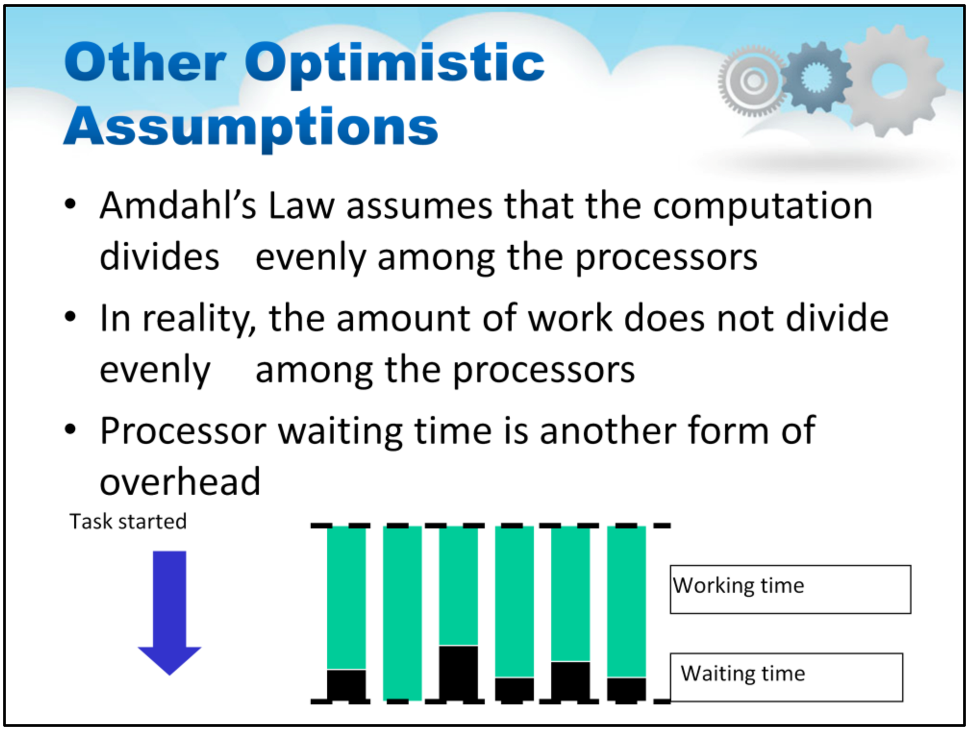
It’s rare that a parallel computation will actually achieve the speedup predicted by Amdahl’s Law.



Graph with Parallel Overhead Added

Parallel overhead increases with # of processors

Taking into account the parallel overhead reduces our expectations about how many processors can be profitably employed in speeding the computation.

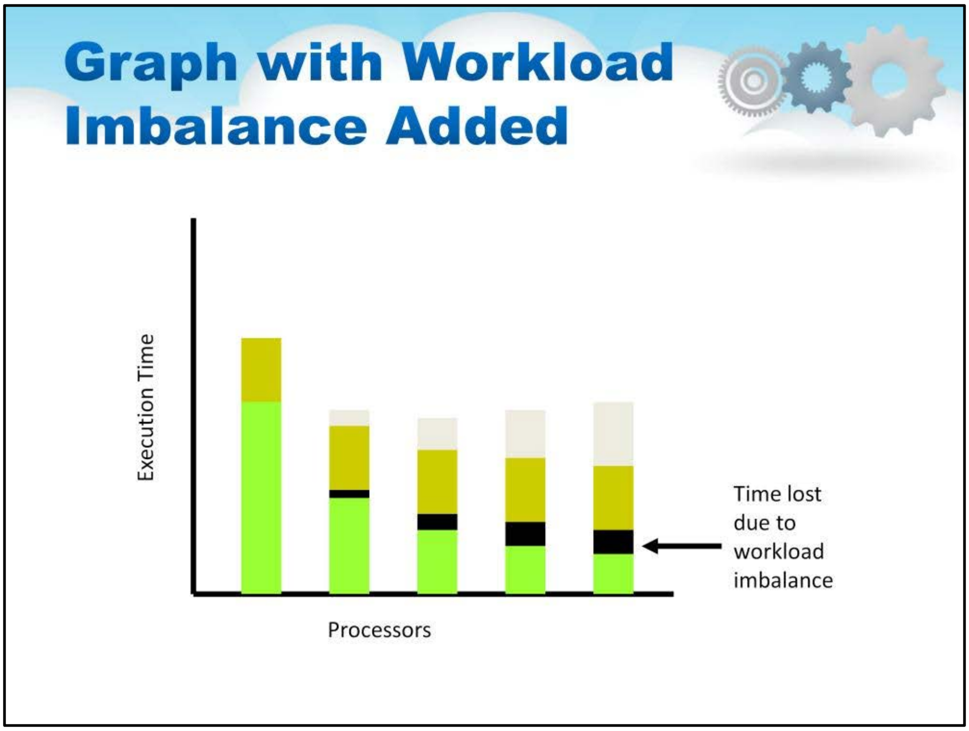


Other Optimistic Assumptions

- Amdahl’s Law assumes that the computation divides evenly among the processors
- In reality, the amount of work does not divide evenly among the processors
- Processor waiting time is another form of overhead

Task started
Working time
Waiting time

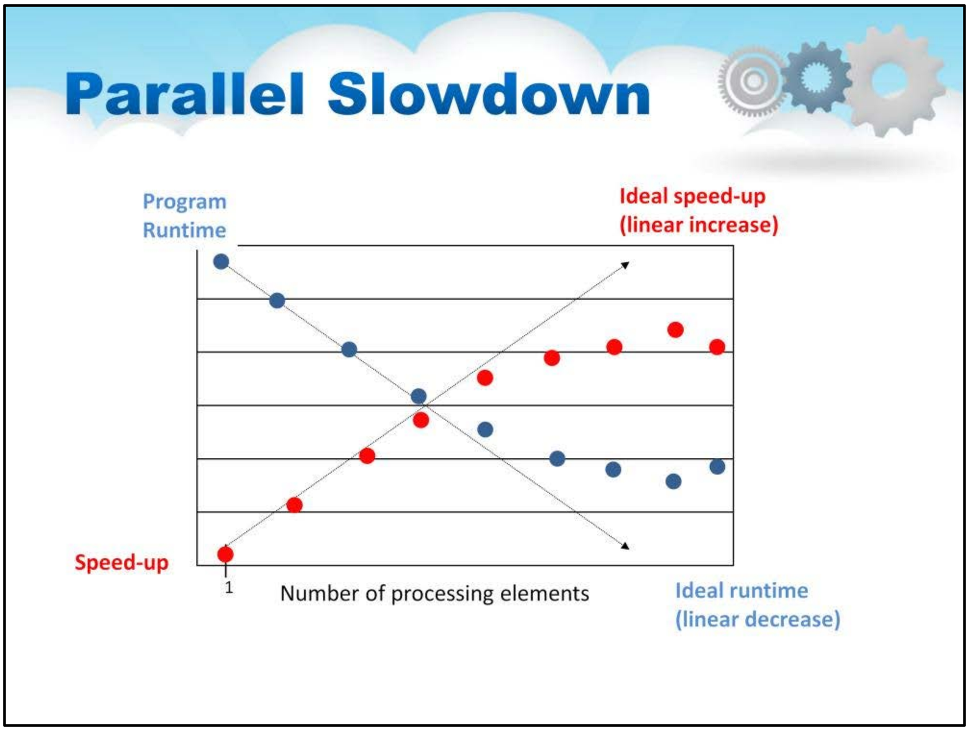
The task is completed only when the last processor finishes. When some processors finish before others, the time that the “early bird” processors spend waiting around for the last processor to finish is wasted. It is a form of overhead because a single processor never spends time waiting for another processor to finish.



Graph with Workload Imbalance Added

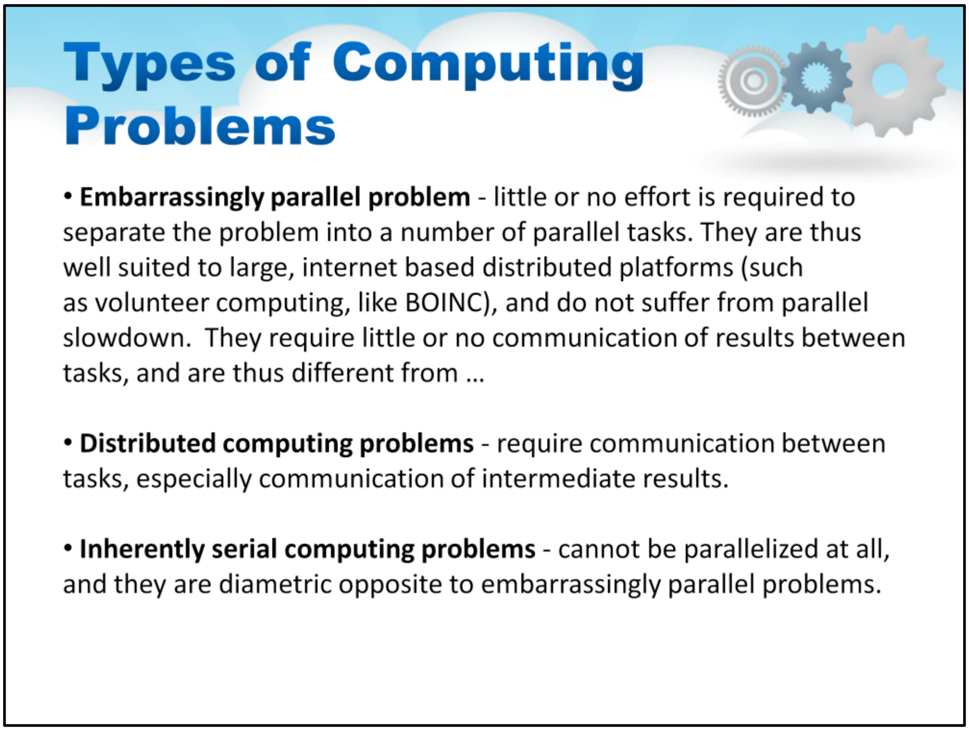
Time lost due to workload imbalance

When we add time lost due to workload imbalance, we see that the benefits of parallelization (in this hypothetical case) don’t extend beyond three processors. That’s a lot worse than the original graph implied.



Parallel Slowdown

blue - shows A diagram of the program runtime;
Red – shows program speed-up of a real-world program with sub-optimal parallelization.
The dashed lines indicate optimal parallelization–linear increase in speedup and linear decrease in program runtime.
Not: the runtime actually increases with more processors (and the speed-up likewise decreases) -> **this is parallel slowdown.**



Types of Computing Problems

- **Embarrassingly parallel problem** - little or no effort is required to separate the problem into a number of parallel tasks. They are thus well suited to large, internet based distributed platforms (such as volunteer computing, like BOINC), and do not suffer from parallel slowdown. They require little or no communication of results between tasks, and are thus different from …

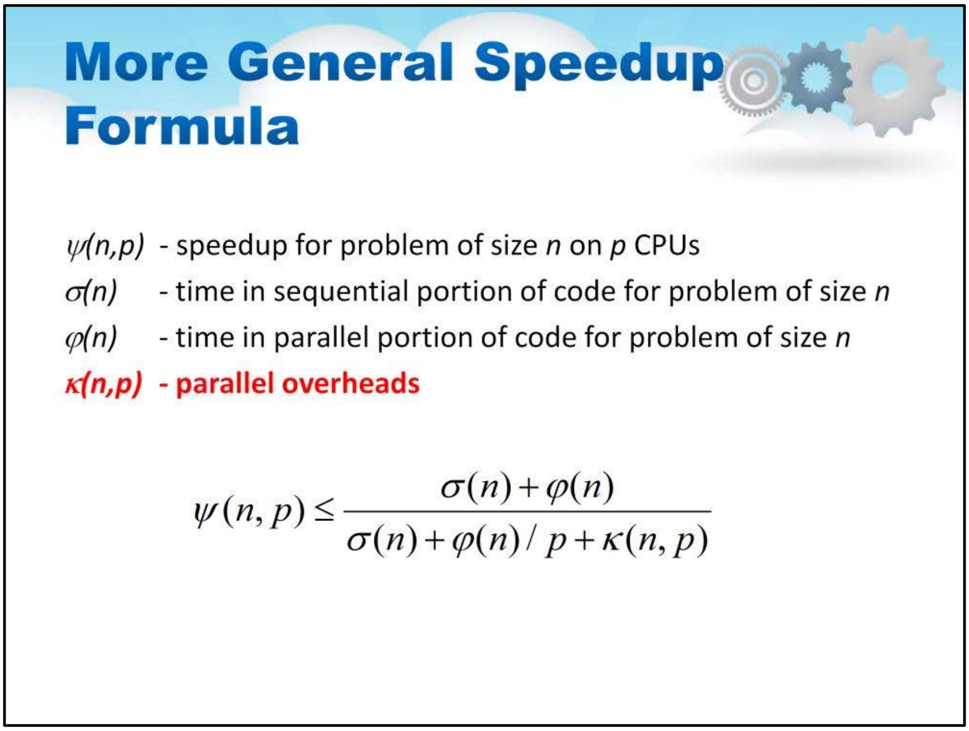
- **Distributed computing problems** - require communication between tasks, especially communication of intermediate results.

- **Inherently serial computing problems** - cannot be parallelized at all, and they are diametric opposite to embarrassingly parallel problems.

Embarrassingly parallel problem - little or no effort is required to separate the problem into a number of parallel tasks. They are thus well suited to large, internet based distributed platforms (such as volunteer computing, like BOINC), and do not suffer from parallel slowdown. They require little or no communication of results between tasks, and are thus different from …

Distributed computing problems - require communication between tasks, especially communication of intermediate results.

Inherently serial computing problems - cannot be parallelized at all, and they are diametric opposite to embarrassingly parallel problems



More General Speedup Formula

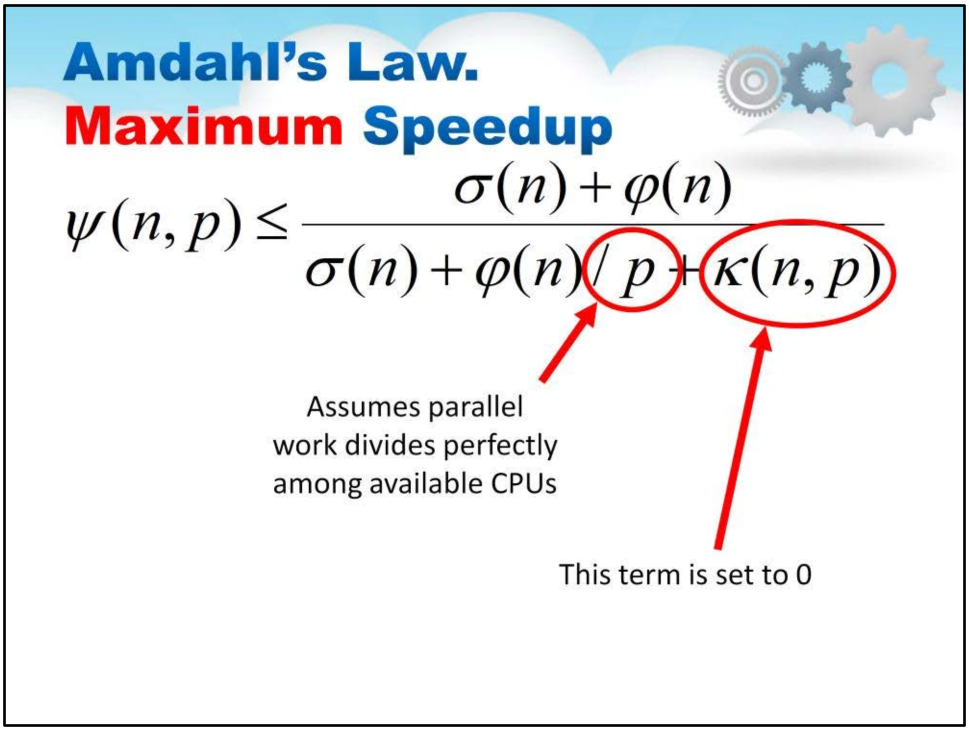
ψ(n,p) - speedup for problem of size *n* on *p* CPUs

σ(n) - time in sequential portion of code for problem of size *n*

φ(n) - time in parallel portion of code for problem of size *n*

***κ(n,p)* - parallel overheads**

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$



Amdahl's Law. Maximum Speedup

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

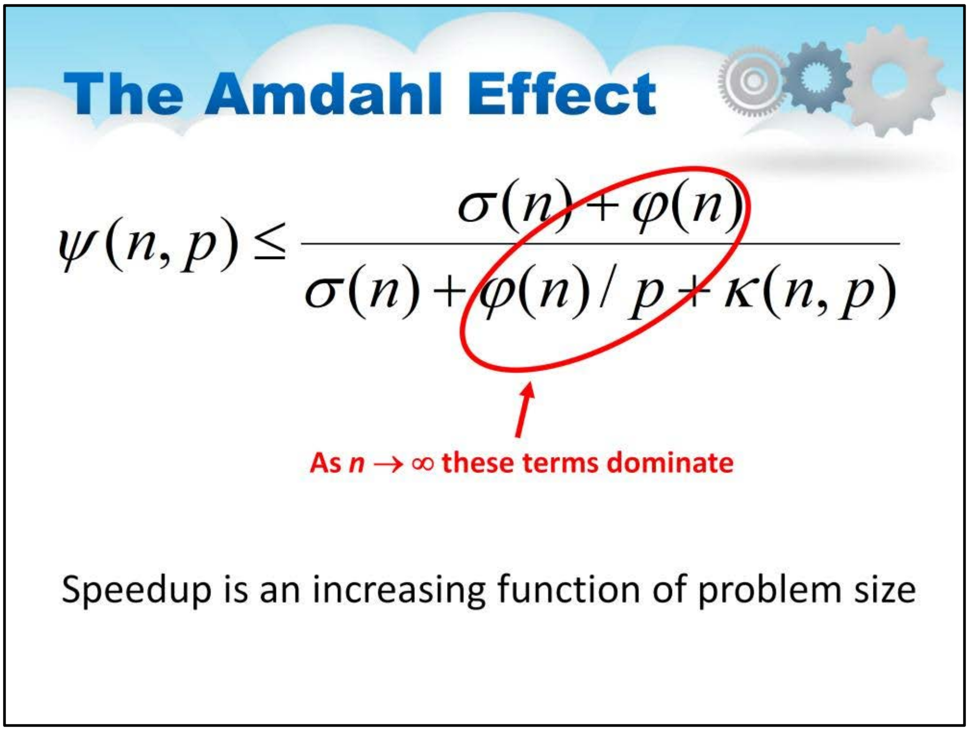
Assumes parallel work divides perfectly among available CPUs

This term is set to 0

The slide highlights the two reasons why we should view Amdahl’s Law as providing an upper bound on the speedup that can be achieved, rather than a realistic prediction.

Question: So why do we bother with Amdahl’s Law?

Answer: Because even an overly optimistic prediction can be useful. For example, suppose we benchmark a program and then use Amdahl’s Law to predict the speedup we would achieve if we made several key functions parallel. If Amdahl’s Law predicts a speedup of 1.20 (i.e., a 20% improvement in speed), we might decide it is not worth the effort.



The Amdahl Effect

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

As $n \to \infty$ these terms dominate

Speedup is an increasing function of problem size

In most parallel programs the functions “sigma(n)” and “kappa(n,p)” are in a lower complexity class than the function “phi(n)”. For example, in matrix multiplication, the I/O of the matrices is order “n” squared, whereas the complexity of the actual matrix multiplication is order “n” cubed.

So, as “n” gets larger, the “phi(n)” terms dominate, and “psi(n,p)” gets closer to “phi(n)/(phi(n)/p)”, or “p”.

Put another way, speedup is an increasing function of problem size. For example, when the problem size is larger, there may be more parallel operations per barrier synchronization. That makes the overhead of the barrier relatively lower.

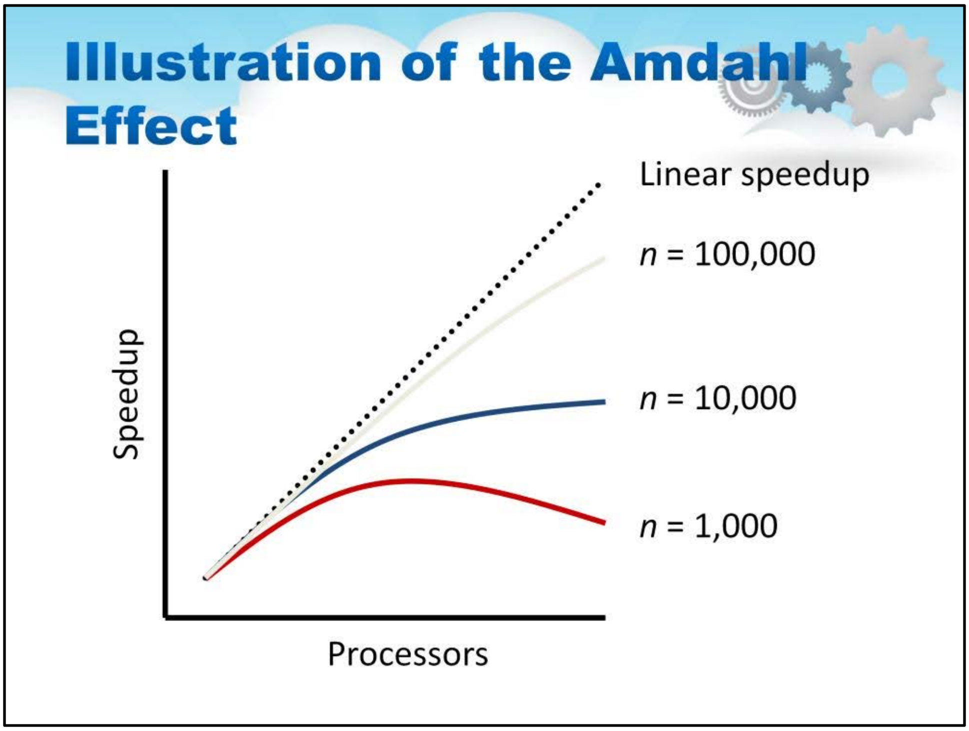
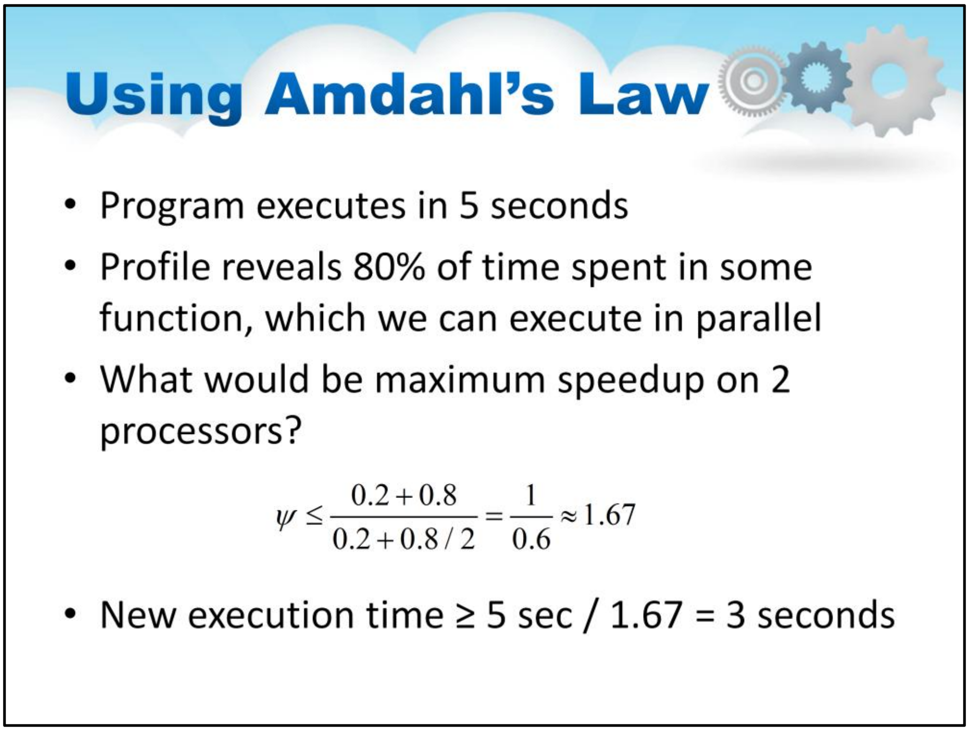


Illustration of the Amdahl Effect

For a given number of processors, as we increase the problem size, the relative amount of time spent doing useful work increases, raising the efficiency and the speedup of the parallel program.



Using Amdahl's Law

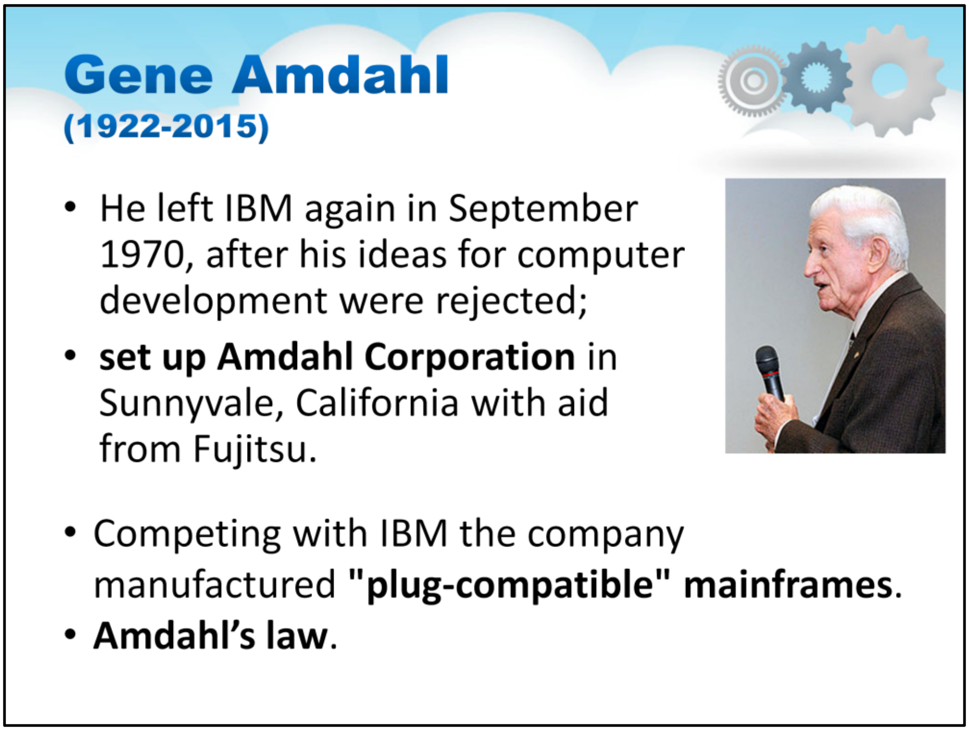
- Program executes in 5 seconds
- Profile reveals 80% of time spent in some function, which we can execute in parallel
- What would be maximum speedup on 2 processors?

$$\psi \leq \frac{0.2+0.8}{0.2+0.8/2} = \frac{1}{0.6} \approx 1.67$$

- New execution time ≥ 5 sec / 1.67 = 3 seconds

Question: If 25% of the program's time is spent in sequential code, what's the greatest speedup that can be achieved, regardless of the number of processors?

Answer: 4. That's because the limit, as "p" goes to infinity, of 1 / (0.25 + (1 - .25)/p) = 1 / 0.25 = 4.



Gene Amdahl

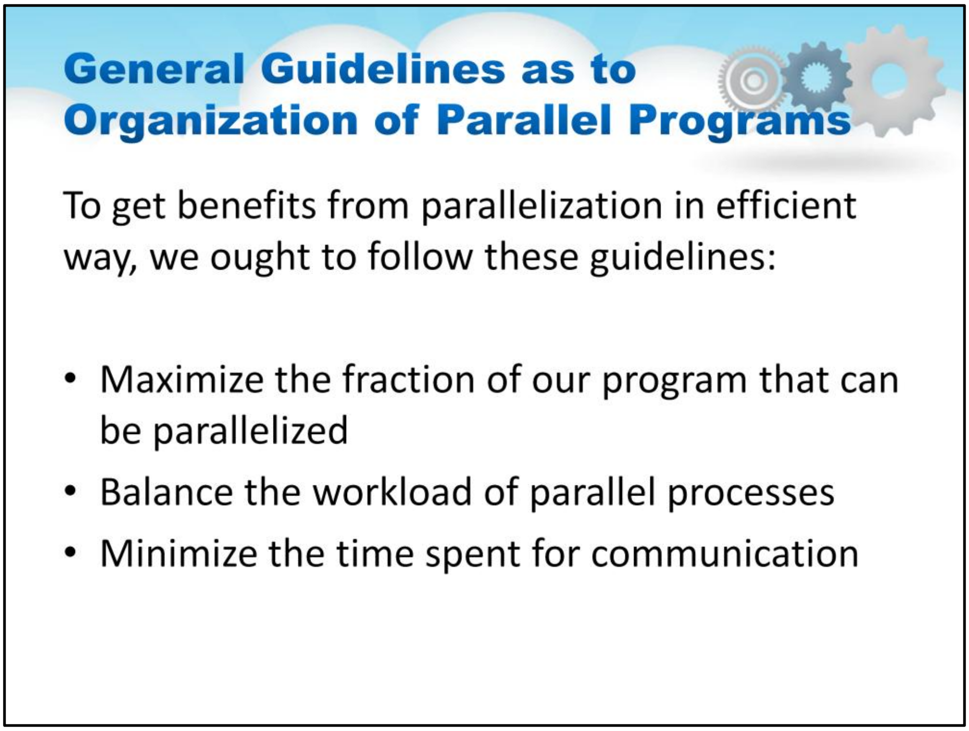
(1922-2015)

- He left IBM again in September 1970, after his ideas for computer development were rejected;
- **set up Amdahl Corporation** in Sunnyvale, California with aid from Fujitsu.
- Competing with IBM the company manufactured **"plug-compatible" mainframes**.
- **Amdahl's law**.

In the early 1960s Amdahl was the chief architect of the IBM System/360, which began the corporation's most profitable mainframe product line. This was the first time the term *architecture* was applied to a computer design.

The System/360 did not use parallel processing, but a rival computer, the ILIAC IV was a SIMD design with 64 processors.

In 1967, Gene Amdahl argued at the AFIPS Spring Joint Computer Conference that because the operating system of the ILIAC IV took 25 to 45 percent of the machine cycles, parallel programs could only achieve a speedup of 2 to 4.



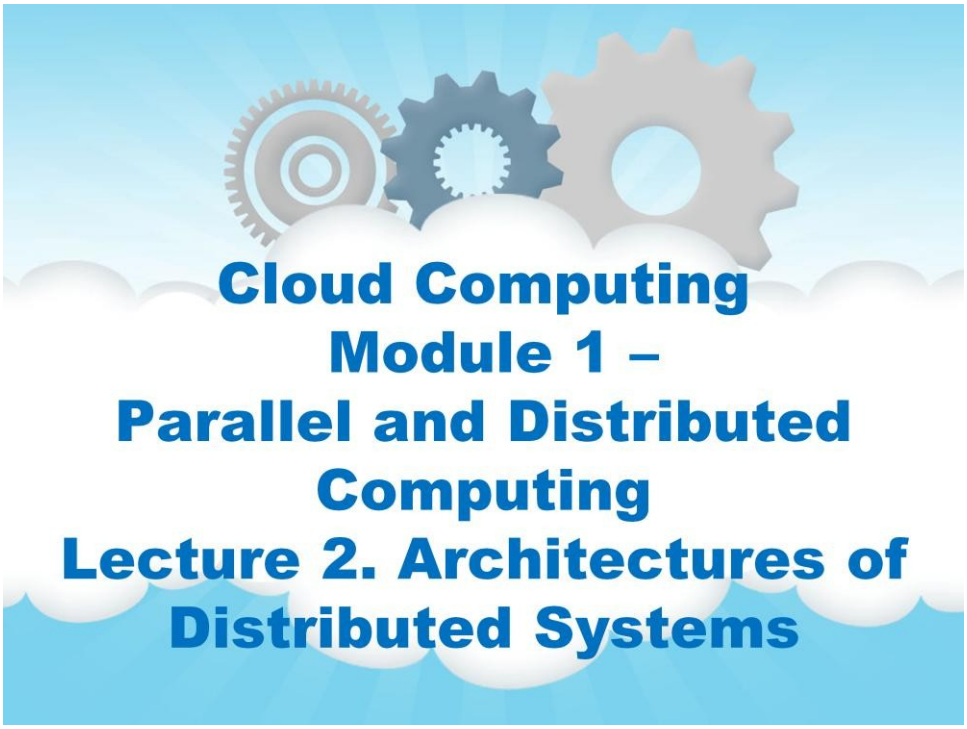
General Guidelines as to Organization of Parallel Programs

To get benefits from parallelization in efficient way, we ought to follow these guidelines:

- Maximize the fraction of our program that can be parallelized
- Balance the workload of parallel processes
- Minimize the time spent for communication

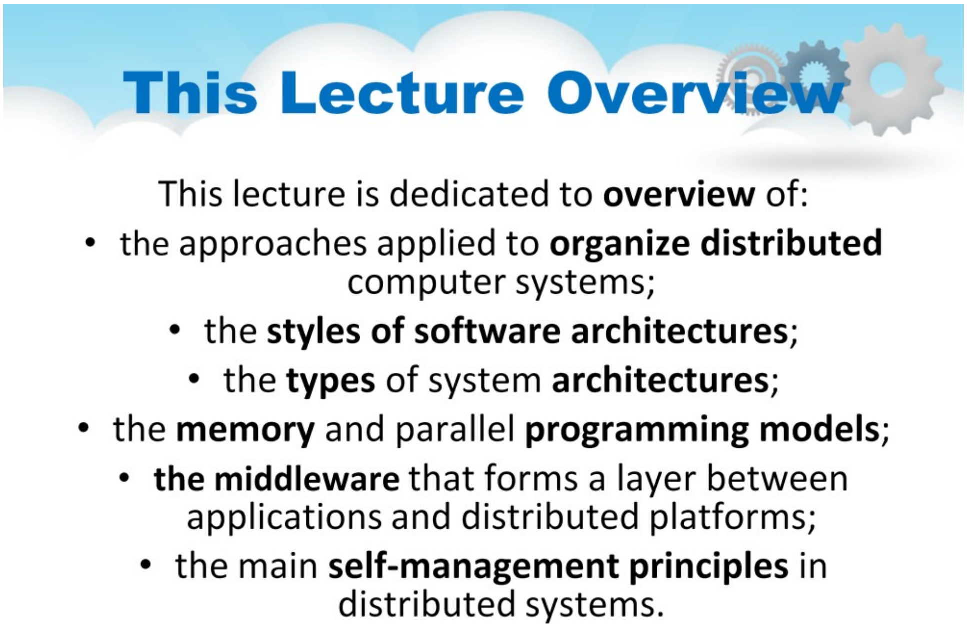
To get benefits from parallelization in efficient way, we ought to follow these guidelines:

Maximize the fraction of our program that can be parallelized
Balance the workload of parallel processes
Minimize the time spent for communication



Cloud Computing Module 1 – Parallel and Distributed Computing Lecture 2. Architectures of Distributed Systems

The title of this module is “Parallel and Distributed Computing”.
This is a lecture 2 “Architectures of Distributed Systems”.



This Lecture Overview

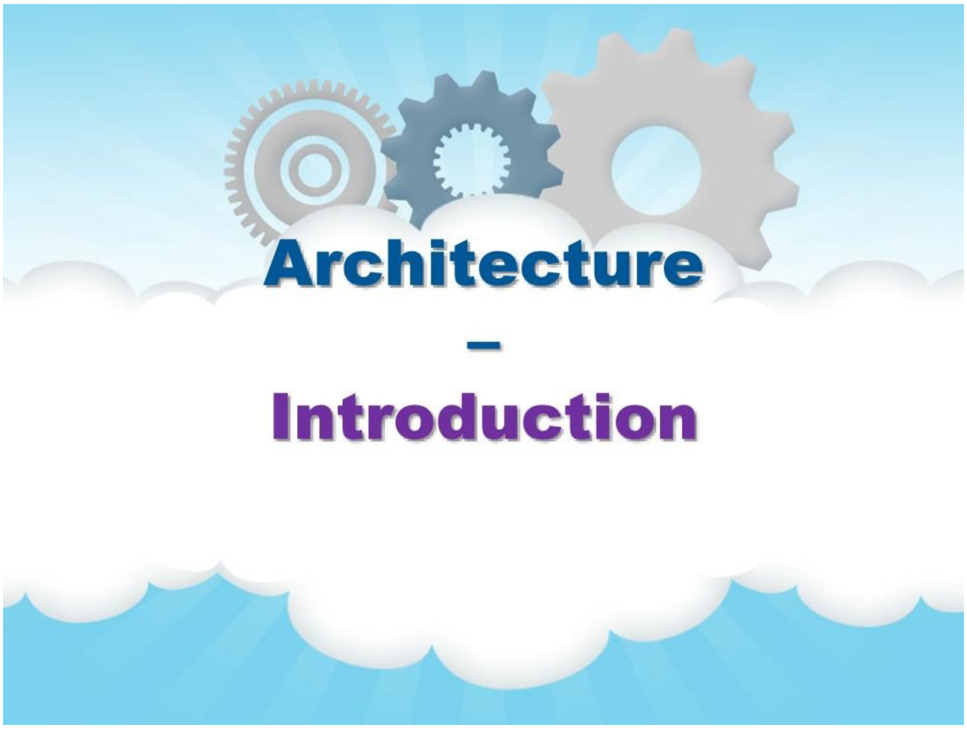
This lecture is dedicated to **overview** of:

- the approaches applied to **organize distributed** computer systems;
- the **styles of software architectures**;
- the **types** of system **architectures**;
- the **memory** and parallel **programming models**;
- **the middleware** that forms a layer between applications and distributed platforms;
- the main **self-management principles** in distributed systems.

Lecture 2. Architectures of Distributed Systems

This lecture is about:

- approaches used to organize distributed computer systems; software architectures styles;
- system architectures types;
- models of memory and parallel programming;
- Middleware, namely, software between applications and distributed platforms;
- and important self-management principles in distributed systems.

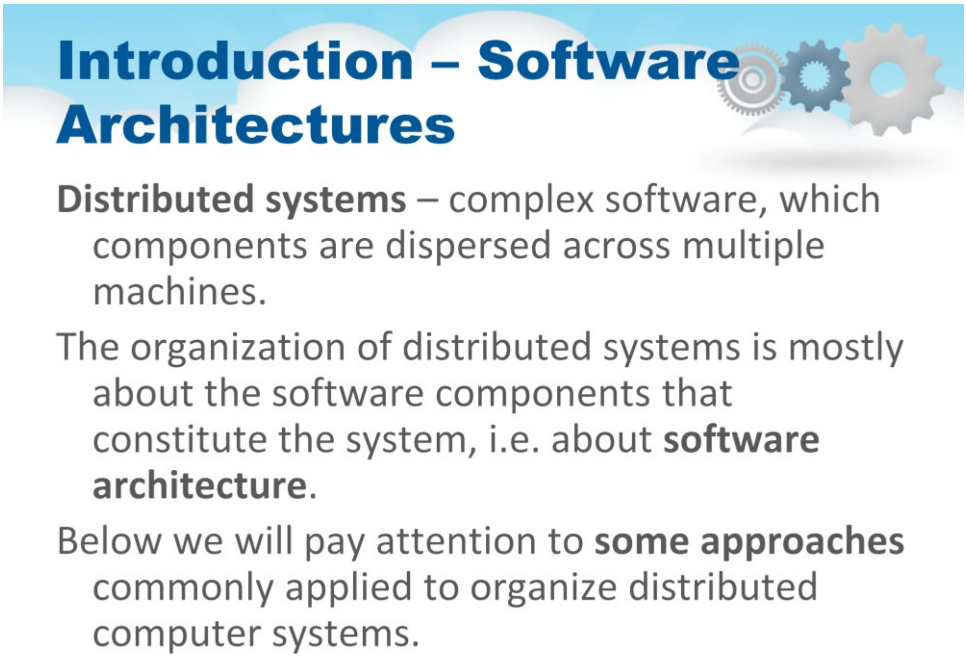


Architecture

Introduction

We start our discussion on architectures by first considering the logical organization of distributed systems into software components, also referred to as software architecture.

Research on software architectures has matured considerably and it is now commonly accepted that designing or adopting an architecture is crucial for the successful development of large systems.



Introduction – Software Architectures

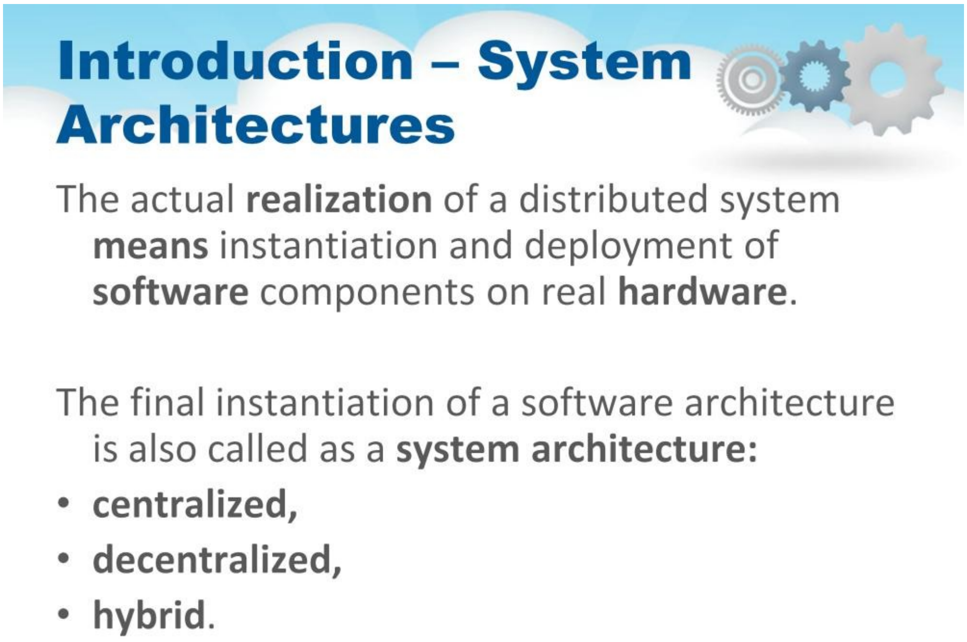
Distributed systems – complex software, which components are dispersed across multiple machines.

The organization of distributed systems is mostly about the software components that constitute the system, i.e. about **software architecture**.

Below we will pay attention to **some approaches** commonly applied to organize distributed computer systems.

Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines. To master their complexity, it is crucial that these systems are properly organized. There are different ways on how to view the organization of a distributed system, but an obvious one is to make a distinction between the logical organization of the collection of software components and on the other hand the actual physical realization.

The organization of distributed systems is mostly about the software components that constitute the system. These software architectures tell us how the various software components are to be organized and how they should interact. In this lecture we will pay attention to some commonly applied approaches toward organizing (distributed) computer systems.



Introduction – System Architectures

The actual **realization** of a distributed system **means** instantiation and deployment of **software** components on real **hardware**.

The final instantiation of a software architecture is also called as a **system architecture:**

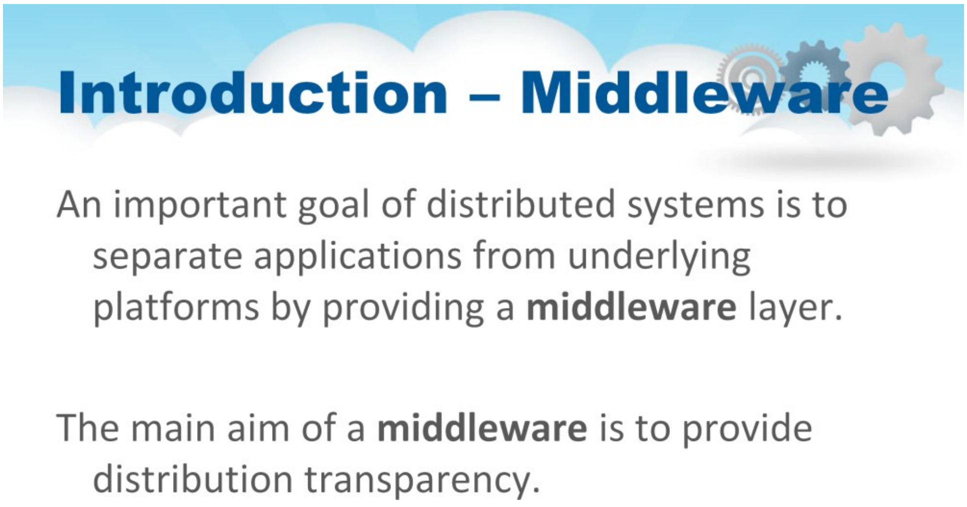
- **centralized,**
- **decentralized,**
- **hybrid**.

The actual realization of a distributed system requires that we instantiate and place software components on real hardware.
There are many different choices that can be made in doing so.
The final instantiation of a software architecture is also referred to as a system architecture.

In this lecture we will look into traditional **centralized** architectures in which a single server implements most of the software components (and thus functionality), while remote clients can access that server using simple communication means.

In addition, we consider **decentralized** architectures in which machines more or less play equal roles.

Also we consider as **hybrid** organizations.



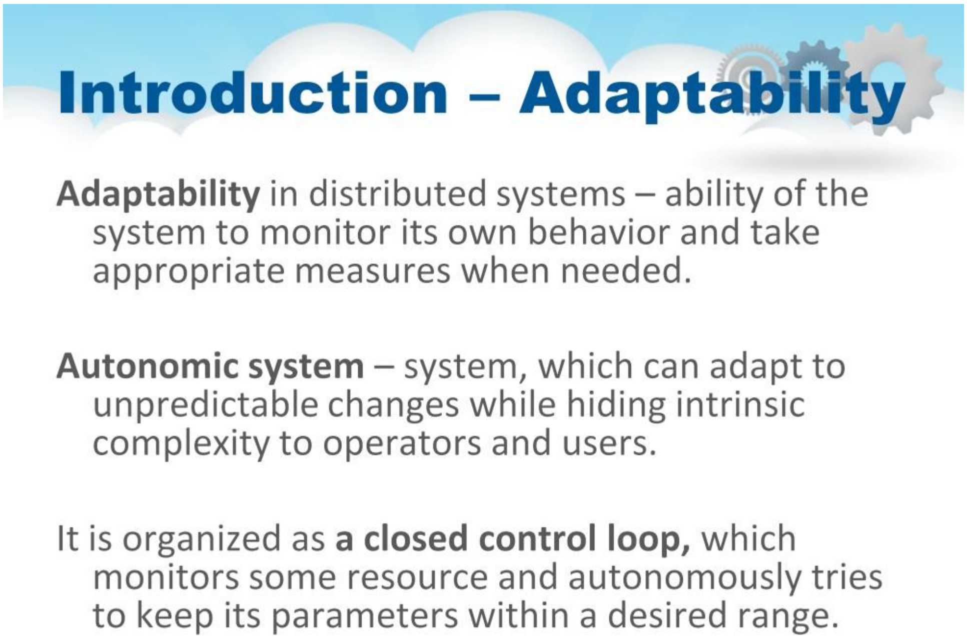
Introduction – Middleware

An important goal of distributed systems is to separate applications from underlying platforms by providing a **middleware** layer.

The main aim of a **middleware** is to provide distribution transparency.

As we explained in the previous lecture, an important goal of distributed systems is to separate applications from underlying platforms by providing a middleware layer.

Adopting such a layer is an important architectural decision, and its main purpose is to provide distribution transparency. However, trade-offs need to be made to achieve transparency, which has led to various techniques to make middleware adaptive. We discuss some of the more commonly applied ones in this lecture, as they affect the organization of the middleware itself.



Introduction – Adaptability

Adaptability in distributed systems – ability of the system to monitor its own behavior and take appropriate measures when needed.

Autonomic system – system, which can adapt to unpredictable changes while hiding intrinsic complexity to operators and users.

It is organized as **a closed control loop,** which monitors some resource and autonomously tries to keep its parameters within a desired range.

Adaptability in distributed systems can also be achieved by having the system monitor its own behavior and taking appropriate measures when needed.

This insight has led to a class of what are now referred to as **autonomic systems**, which are systems adapting to unpredictable changes while hiding intrinsic complexity to operators and users.

These distributed systems are frequently organized in the form of feedback **control loops**, which create an important architectural element during a system's design.

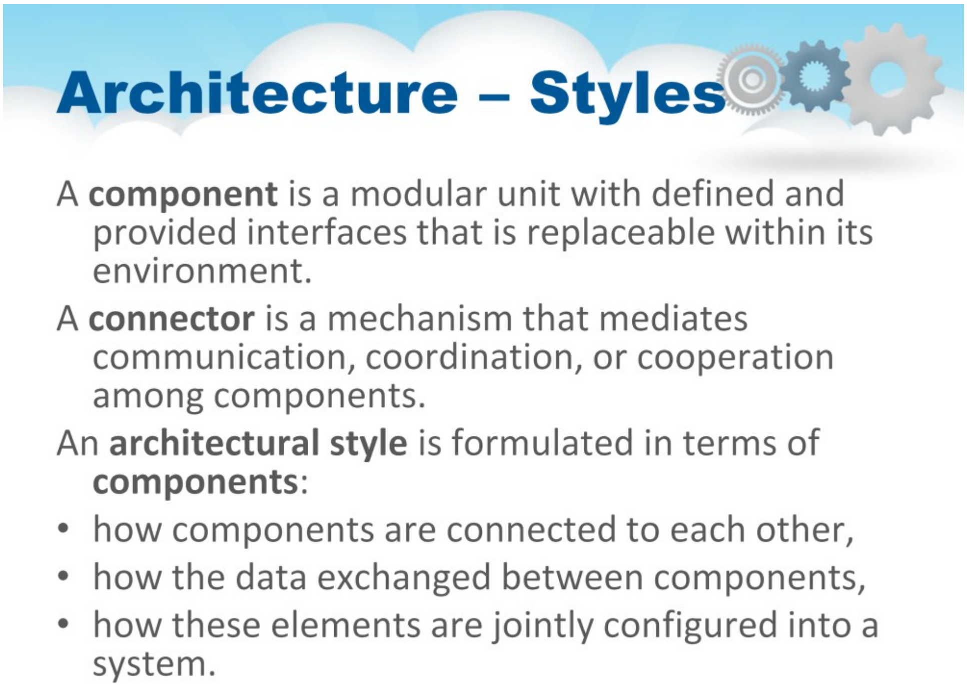
In this lecture, we devote a section to **autonomic distributed systems**.

Architecture – Styles

Styles

Let‘s start our discussion on architectures by first considering the logical organization of distributed systems into software components, also referred to as software architecture.

Research on software architectures has matured considerably and it is now commonly accepted that designing or adopting an architecture is crucial for the successful development of large systems.



Architecture – Styles

A **component** is a modular unit with defined and provided interfaces that is replaceable within its environment.

A **connector** is a mechanism that mediates communication, coordination, or cooperation among components.

An **architectural style** is formulated in terms of **components**:

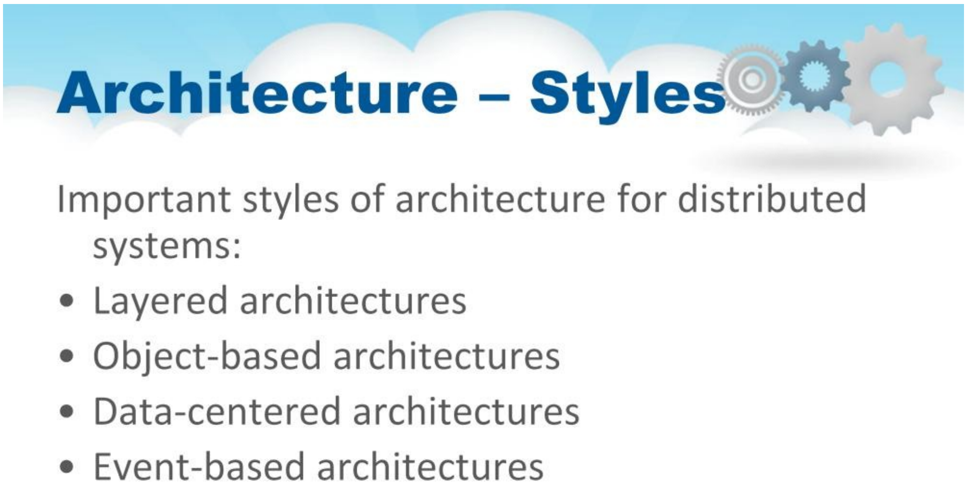
- how components are connected to each other,
- how the data exchanged between components,
- how these elements are jointly configured into a system.

The notion of an architectural style is important and based on components and connectors.

A component is a modular unit with well-defined required and provided interfaces that is replaceable within its environment. As we shall discuss below, the important issue about a component for distributed systems is that it can be replaced, provided we respect its interfaces.

A somewhat more difficult concept to grasp is that of a connector, which is generally described as a mechanism that mediates communication, coordination, or cooperation among components. For example, a connector can be formed by the facilities for (remote) procedure calls, message passing, or streaming data.

An architectural style is formulated in terms of components, the way that components are connected to each other, the data exchanged between components. and finally how these elements are jointly configured into a system.



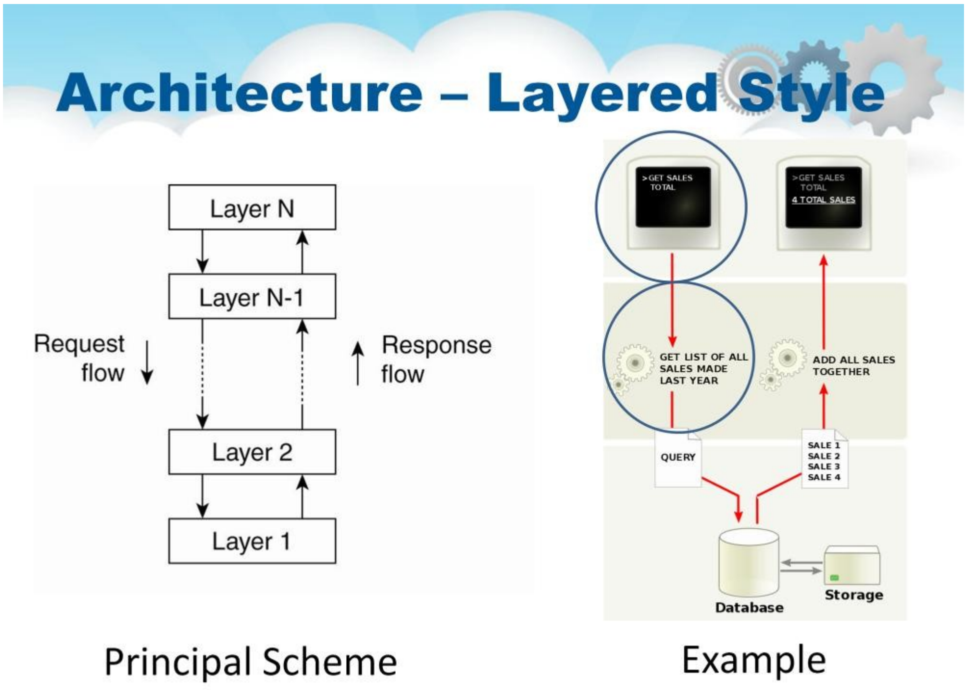
Architecture – Styles

Important styles of architecture for distributed systems:

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

Using components and connectors, we can come to various configurations, which, in turn have been classified into architectural styles.
Several styles have by now been identified, of which the most importantones for distributed systems are:

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures



Architecture – Layered Style

Principal Scheme

Example

[LEFT SIDE]

The basic idea for the **layered style** is simple:

components are organized in a layered fashion where a component at layer L_i is allowed to make requests to components at the underlying layer L_{i-1}, but not the other way around, as shown in this Figure.

This model has been widely accepted by the IT community. A key observation is that control generally flows from layer to layer: requests go down the hierarchy, but the results, in reverse, flow up.

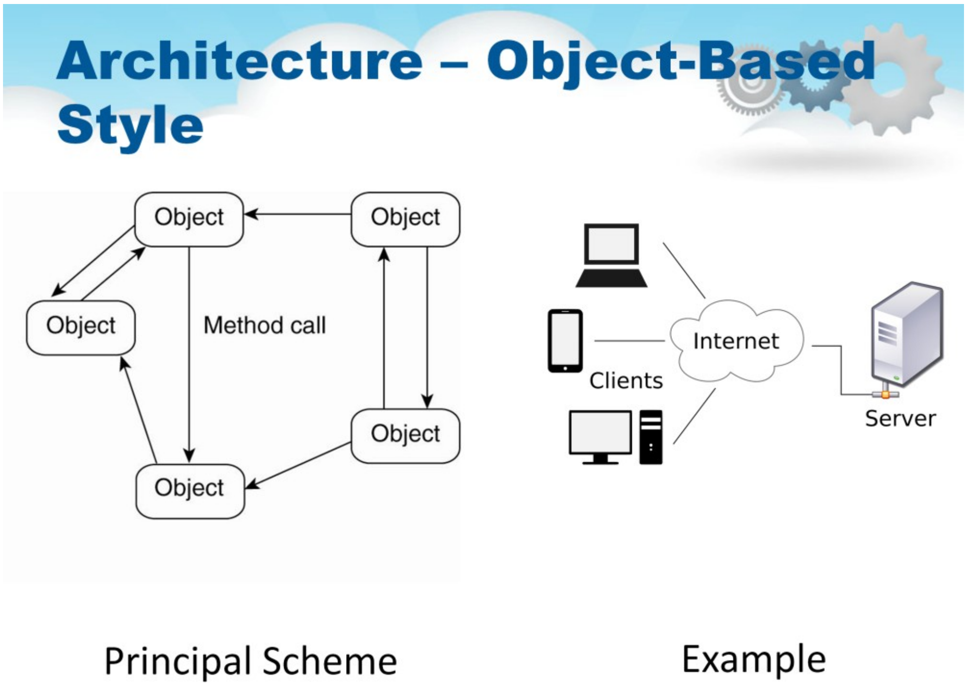
[RIGHT SIDE]

On the right side you can see the typical implementation of this architecture style. The **top-most** level (**Presentation Layer** in this example) creates the User Interface. Its main aim is to translate user requests to machine and, in reverse, to translate machine responses to user.

The **middle** level (**Logic Layer** here) coordinates the application, commands, makes evaluations, perform calculations and derive decisions.

And it delivers requests-responses and data between the other layers.

The bottom level (**Data Layer** here) contain the information in database or file system. After requests the information from this Data Layer is passed to the Logic Layer for processing and to the Presentation Layer for users.



Architecture – Object-Based Style

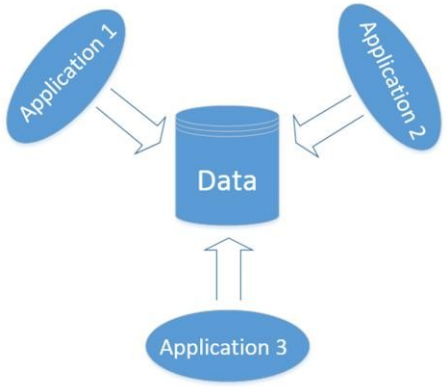
Principal Scheme

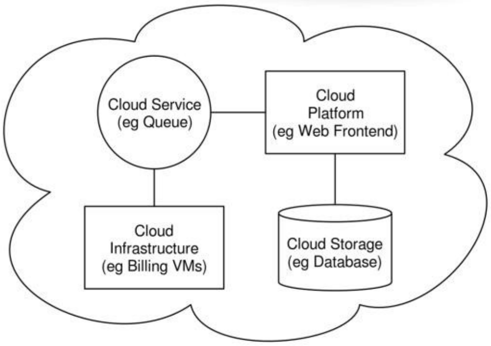
Example

[LEFT SIDE]
A more flexible organization is proposed in **object-based** architectures, which are illustrated in this Figure.
In essence, each object corresponds to what we have defined as a component,
and these components are connected through a (remote) procedure call mechanism.

[RIGHT SIDE]
This architecture style corresponds to the **client-server** system architecture (in details described below).
The layered and object-based architectures still form the most important styles for large software systems.
In this example of a computer network diagram, clients communicate with a server via the Internet.
A server runs one or more server programs which share their resources with clients.

Architecture – **Data-Centered Style**



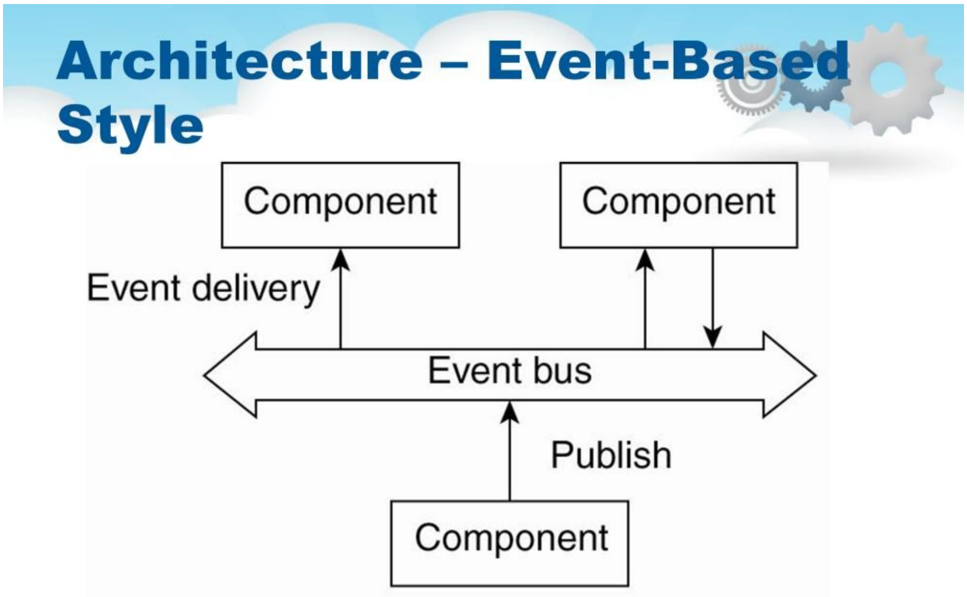

Cloud Service (eg Queue)
Cloud Platform (eg Web Frontend)
Cloud Infrastructure (eg Billing VMs)
Cloud Storage (eg Database)

Principal Scheme

Example

[LEFT SIDE]
Data-centered architectures views **data** as the **most valuable part** of the application, where processes communicate through a common (passive or active) repository of data.

[RIGHT SIDE]
In the context of distributed systems these architectures are as important as the layered and object-based architectures.
For example, many networked applications rely on a shared distributed file system in which virtually all communication takes place through files.
Likewise, cloud computing systems, which we discuss extensively later, are largely data-centric: processes communicate through the use of shared data services.



Architecture – Event-Based Style

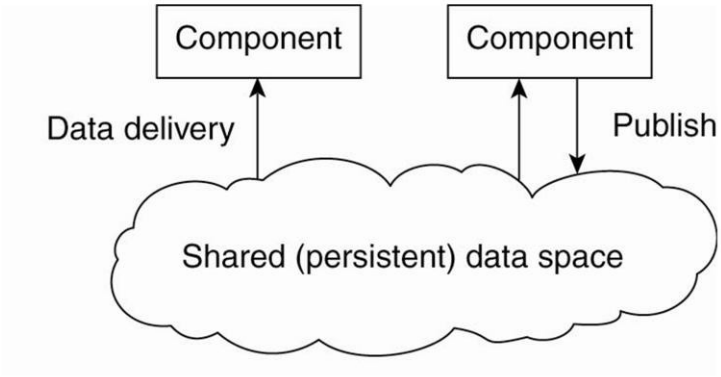
Principal Scheme

In event-based architectures, processes essentially communicate through the propagation of events, which optionally also carry data, as shown in this Figure.

For distributed systems, event propagation has generally been associated with what are known as publish/subscribe systems.
The basic idea is that processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.

The main advantage of event-based systems is that processes are loosely coupled.
In principle, they need not explicitly refer to each other.
This is also referred to as being **decoupled in space**, or **referentially decoupled**.

Architecture – Combined Style: Event-Based + Data-Centered



Principal Scheme

Event-based architectures can be combined with data-centered architectures, what is also known as shared data spaces.
The essence of shared data spaces is that processes are now also decoupled in time: they need not both be active when communication takes place.
Furthermore, many shared data spaces use a SQL-like interface to the shared repository in that sense that data can be accessed using a description rather than an explicit reference,
as is the case with files.

What makes these software architectures important for distributed systems is that they all aim at achieving (at a reasonable level) distribution transparency.
However, distribution transparency requires making trade-offs between performance, fault tolerance, ease-of-programming, and so on.
As there is no single solution that will meet the requirements for all possible distributed applications,
researchers have abandoned the idea that a single distributed system can be used to cover 90% of all possible cases.

System Architectures

System Architectures

System Architectures

- Centralized Architectures
- Decentralized Architectures
- Hybrid Architectures

Now that we have briefly discussed some common architectural styles, let us take a look at how many distributed systems are actually organized by considering where software components are placed.

Software components, their interaction, and their placement leads to creation of an instance of a software architecture, also called a system architecture.

We will discuss centralized and decentralized organizations, and various hybrid forms.

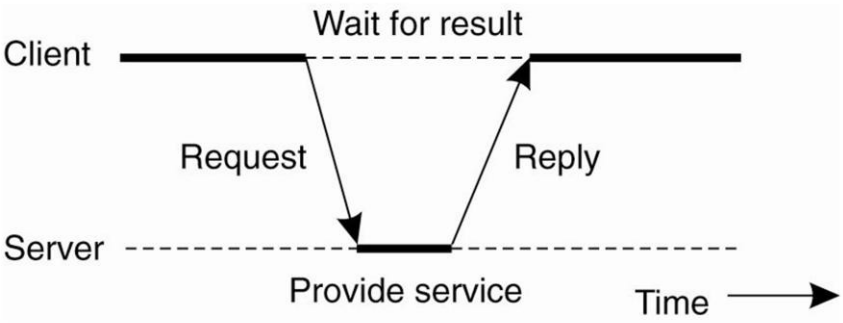
System Architectures –

Centralized Architectures

Centralized Architectures

Despite the lack of consensus on many distributed systems issues, there is one issue that many researchers and practitioners agree upon: thinking in terms of clients that request services from servers helps us understand and manage the complexity of distributed systems and that is a good thing.

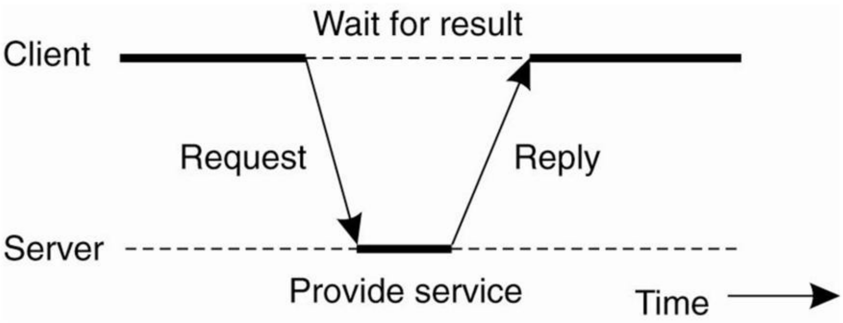
Centralized Architecture – Client-Server



General **client-server** interaction (or **request-reply** behavior) between a client and a server

In the basic client-server model, processes in a distributed system are divided into two (possibly overlapping) groups.
A **server** is a process implementing a specific service, for example, a file system service or a database service.
A **client** is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.
This client-server interaction, also known as **request-reply behavior** is shown in this Figure.

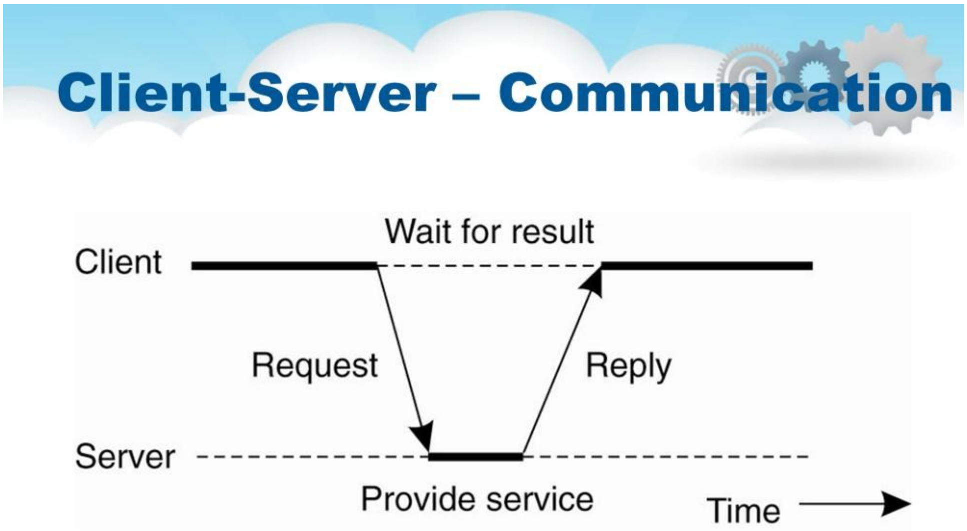
Centralized Architecture – Client-Server – Communication



Communication between a client and a server can be implemented by means of a simple connectionless protocol when the underlying network is fairly reliable as in many local-area networks.

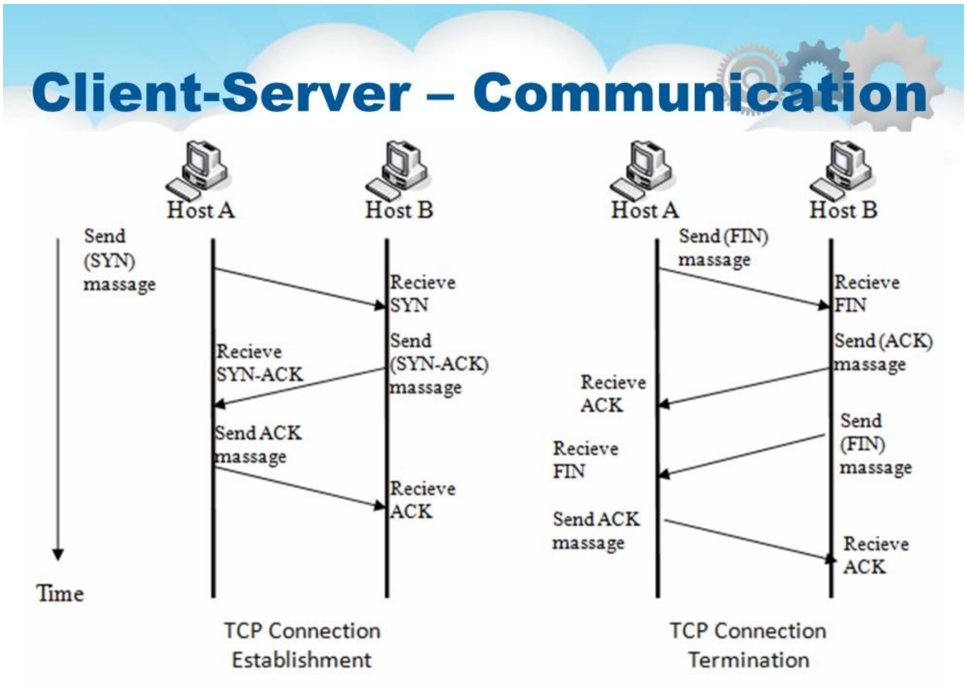
In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, along with the necessary input data. The message is then sent to the server.

The latter, in turn, will always wait for an incoming request, subsequently process it, and package the results in a reply message that is then sent to the client.



Client-Server – Communication

Using a connectionless protocol has the obvious advantage of being efficient. As long as messages do not get lost or corrupted, the request/reply protocol just sketched works fine. Unfortunately, making the protocol resistant to occasional transmission failures is not trivial. The only thing we can do is possibly let the client resend the request when no reply message comes in. The problem, however, is that the client cannot detect whether the original request message was lost, or that transmission of the reply failed. If the reply was lost, then resending a request may result in performing the operation twice. If the operation was something like "transfer $10,000 from my bank account," then clearly, it would have been better that we simply reported an error instead. On the other hand, if the operation was "tell me how much money I have left," it would be perfectly acceptable to resend the request. When an operation can be repeated multiple times without harm, it is said to be idempotent. Since some requests are idempotent and others are not it should be clear that there is no single solution for dealing with lost messages.



Client-Server – Communication

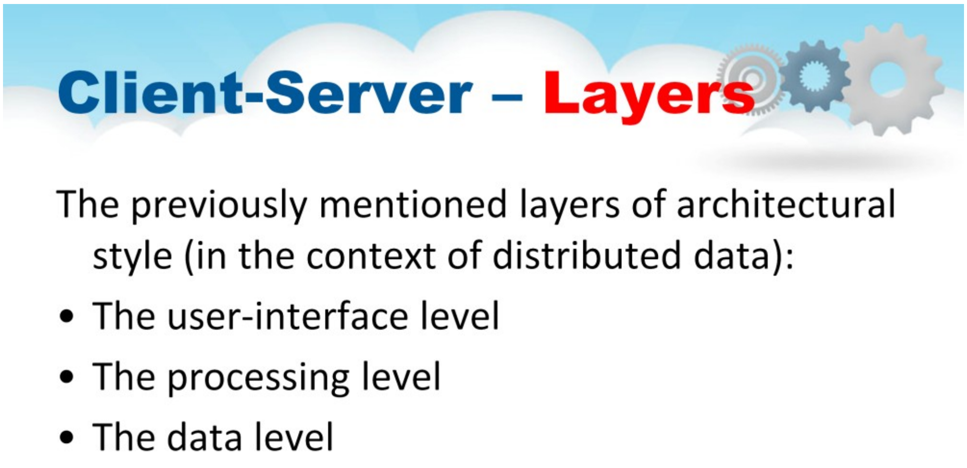
As an alternative, many client-server systems use a reliable connection-oriented protocol. Although this solution is not entirely appropriate in a local-area network due to relatively low performance, it works perfectly tine in wide-area systems in which communication is inherently unreliable. For example, virtually all Internet application protocols are based on reliable TCPI IP connections. In this case, whenever a client requests a service, it first sets up a connection to the server before sending the request. The server generally uses that same connection to send the reply message, after which the connection is torn down. The trouble is that setting up and tearing down a connection is relatively costly, especially when the request and reply messages are small.

[LEFT SIDE] Let’s consider connection-oriented communication in “three way handshake” **for TCP connection establishment**:

1. The first arrow is a request for synchronization (called SYN)
2. The second arrow is acknowledgment of synchronization (called SYN-ACK).
3. The third arrow is also acknowledgment to inform the receiver that the connection has been established (called ACK)

[RIGHT SIDE] And the similar sequence of actions **for TCP connection termination:**

1. The host A, who needs to terminate the connection, sends a special message with the FIN (finish) flag, indicating that it has finished sending the data.
2. The host B, who receives the FIN segment, does not terminate the connection but enters into a "passive close" (CLOSE_WAIT) state and sends the ACK for the FIN back to the host A. Now the host B enters into LAST_ACK state. At this point host B will no longer accept data from host A, but can continue transmit data to host A. If host B does not have any data to transmit to the host A it will also terminate the connection by sending FIN segment.
3. When the host A receives the last ACK from the host B, it enters into a (TIME_WAIT) state, and sends an ACK back to the host B.
4. Host B gets the ACK from the host A and closes the connection.



Client-Server – Layers

The previously mentioned layers of architectural style (in the context of distributed data):

- The user-interface level
- The processing level
- The data level

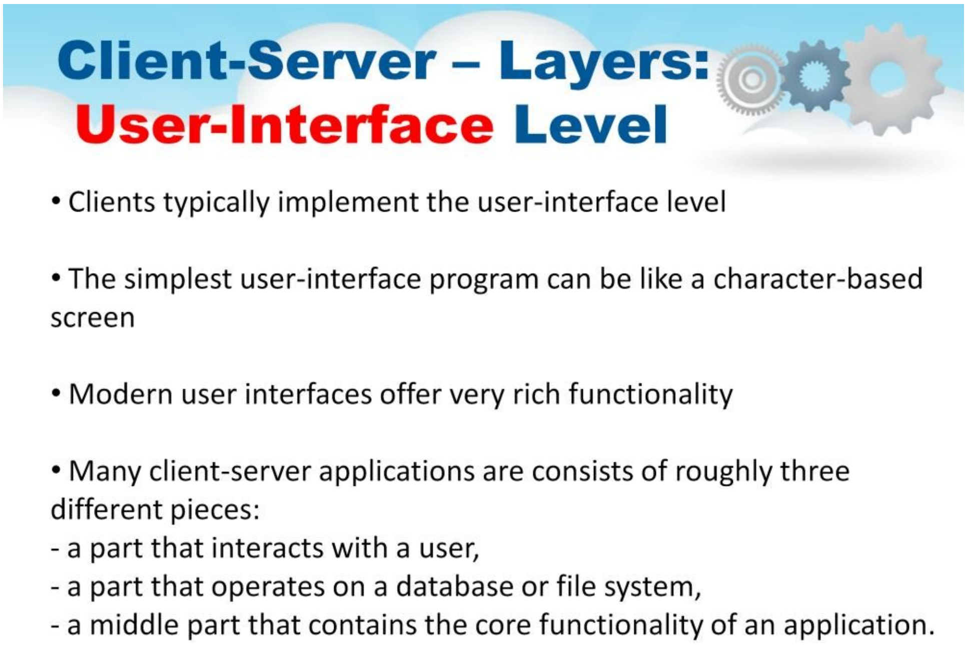
The client-server model has been subject to many debates and controversies over the years. One of the main issues was how to draw a clear distinction between a client and a server. Not surprisingly, there is often no clear distinction. For example, a server for a distributed database may continuously act as a client because it is forwarding requests to different file servers responsible for implementing the database tables. In such a case, the database server itself essentially does no more than process queries.

However, if client-server applications are targeted toward supporting user access to databases, then the following three levels can be considered:

1. The user-interface level - contains all that is necessary to directly interface with the user, such as display management.

2. The processing level - typically contains the applications.

3. The data level - manages the actual data that is processed by these applications for the users.



Client-Server – Layers: User-Interface Level

- Clients typically implement the user-interface level
- The simplest user-interface program can be like a character-based screen
- Modern user interfaces offer very rich functionality
- Many client-server applications are consists of roughly three different pieces:
  - a part that interacts with a user,
  - a part that operates on a database or file system,
  - a middle part that contains the core functionality of an application.

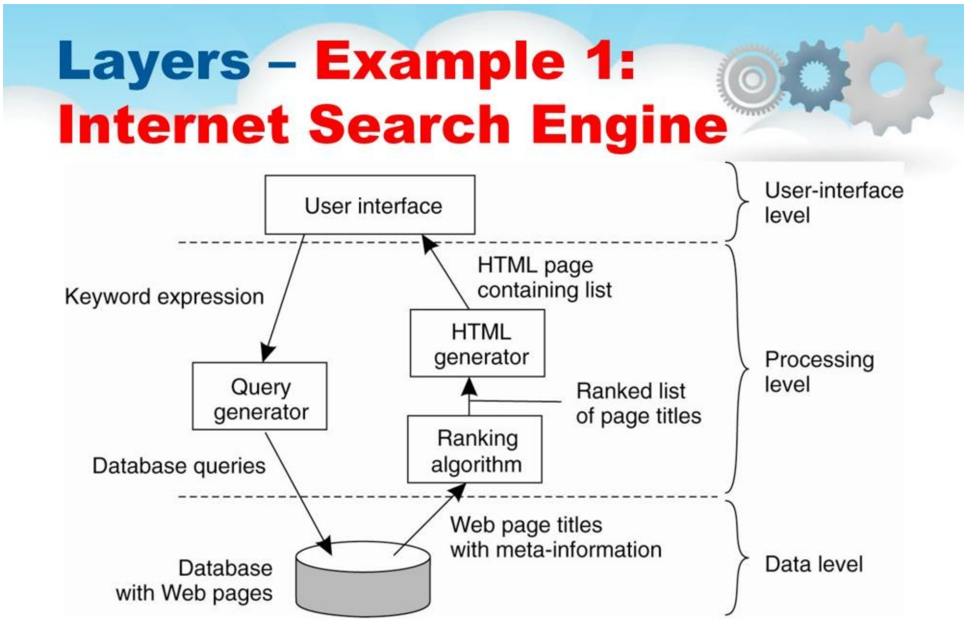
Clients typically implement the user-interface level. This level consists of the programs that allow end users to interact with applications. There is a considerable difference in how sophisticated user-interface programs are.

The simplest user-interface program is nothing more than a character-based screen. Such an interface has been typically used in mainframe environments. In those cases where the mainframe controls all interaction, including the keyboard and monitor, one can hardly speak of a client-server environment. However, in many cases, the user's terminal does some local processing such as echoing typed keystrokes, or supporting form-like interfaces in which a complete entry is to be edited before sending it to the main computer.

Nowadays, even in mainframe environments, we see more advanced user interfaces. Typically, the client machine offers at least a graphical display in which pop-up or pull-down menus are used, and of which many of the screen controls are handled through a mouse instead of the keyboard. Typical examples of such interfaces include the X-Windows interfaces as used in many UNIX environments, and earlier interfaces developed for MS-DOS PCs and Apple Macintoshes.

Modern user interfaces offer considerably more functionality by allowing applications to share a single graphical window, and to use that window to exchange data through user actions. For example, to delete a file, it is usually possible to move the icon representing that file to an icon representing a trash can. Likewise, many word processors allow a user to move text in a document to another position by using only the mouse.

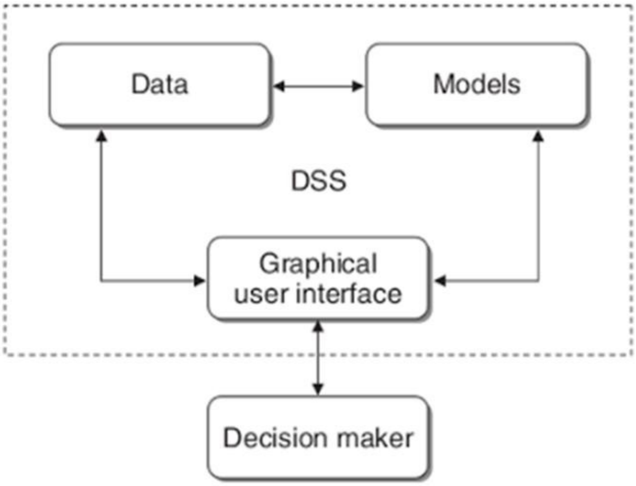
Many client-server applications can be constructed from roughly three different pieces: a part that handles interaction with a user, a part that operates on a database or file system, and a middle part that generally contains the core functionality of an application. This middle part is logically placed at the processing level. In contrast to user interfaces and databases, there are not many aspects common to the processing level. Therefore, we shall give several examples to make this level clearer.



Layers – Example 1: Internet Search Engine

As a first example, consider an Internet search engine. Ignoring all the animated banners, images, and other fancy window dressing, the user interface of a search engine is very simple: a user types in a string of keywords and is subsequently presented with a list of titles of Web pages. The back end is formed by a huge database of Web pages that have been prefetched and indexed. The core of the search engine is a program that transforms the user's string of keywords into one or more database queries. It subsequently ranks the results into a list, and transforms that list into a series of HTML pages. Within the client-server model, this information retrieval part is typically placed at the processing level.

Layers – Example 2: Decision Support System



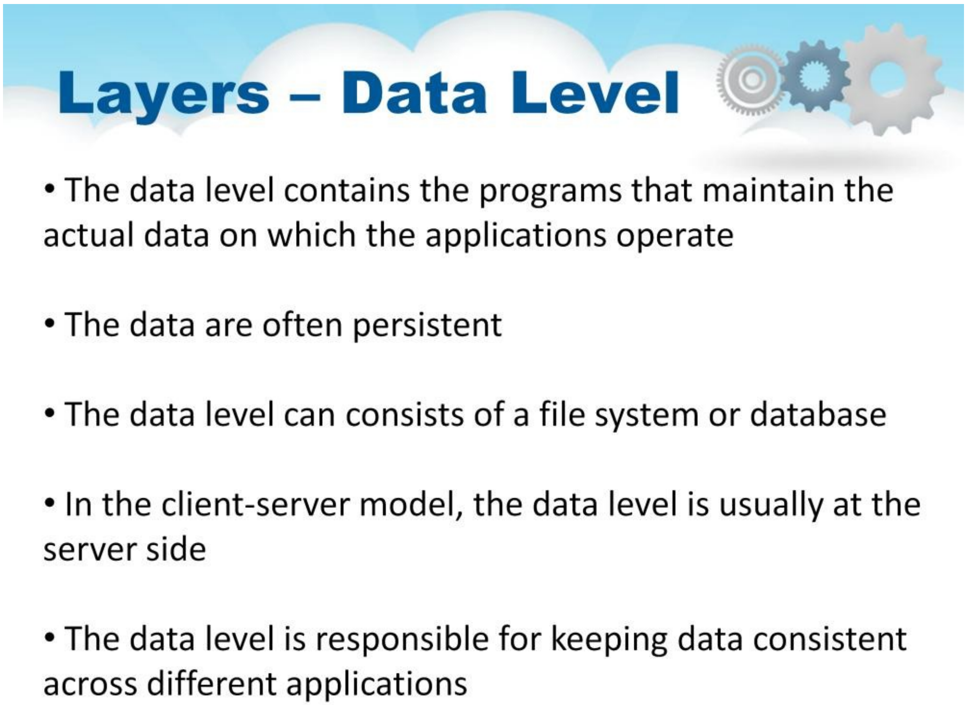
As a second example, consider a decision support system for a stock brokerage. Analogous to a search engine, such a system can be divided into a front end implementing the user interface, a back end for accessing a database with the financial data, and the analysis programs between these two. Analysis of financial data may require sophisticated methods and techniques from statistics and artificial intelligence. In some cases, the core of a financial decision support system may even need to be executed on high-performance computers in order to achieve the throughput and responsiveness that is expected from its users.

In general, Decision Support System developed by the three major components, namely database management, Base Model and Software System / User Interface.

1. Database Management.
   Is a subsystem of data organized in a database. Data that is a decision support system may come from outside and within the environment. For the purposes of SPK, the necessary data relevant to the problem to be solved through simulation.

2. Model Base.
   Is a model that represents the problem into a format quantitative (mathematical model as an example) as the basis of simulations or decision-making, including the purpose, related components, limitations exist (constraints), and related matters Other. Base Model enables decision makers to analyze as a whole by developing and comparing alternative solutions

3. User Interphase / Management Dialog.
   Sometimes referred to as a subsystem of dialogue, a merger between the two previous components, namely Database Management and Model Base incorporated in the third component (user interface), having previously presented in the form of computer models that understandable. User Interface display system output to the user and receive input from the user into the Decision Support System.



Layers – Data Level

- The data level contains the programs that maintain the actual data on which the applications operate
- The data are often persistent
- The data level can consists of a file system or database
- In the client-server model, the data level is usually at the server side
- The data level is responsible for keeping data consistent across different applications

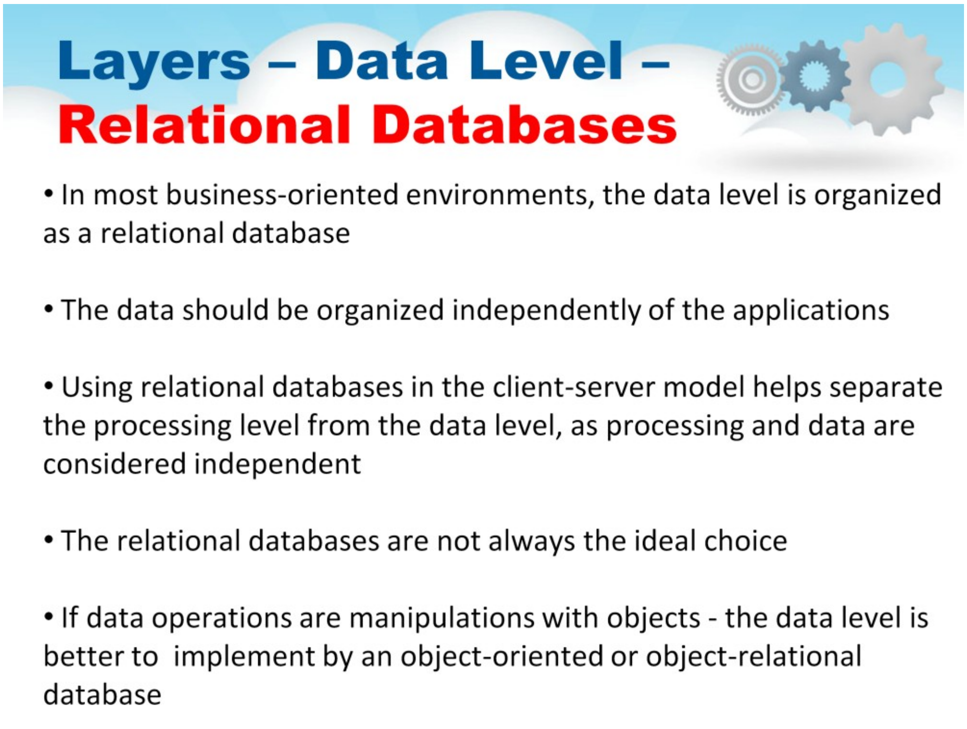
The data level in the client-server model contains the programs that maintain the actual data on which the applications operate.

An important property of this level is that data are often persistent, that is, even if no application is running, data will be stored somewhere for next use.

In its simplest form, the data level consists of a file system, but it is more common to use a full-fledged database.

In the client-server model, the data level is typically implemented at the server side.

Besides merely storing data, the data level is generally also responsible for keeping data consistent across different applications. When databases are being used, maintaining consistency means that metadata such as table descriptions, entry constraints and application-specific metadata are also stored at this level. For example, in the case of a bank, we may want to generate a notification when a customer's credit card debt reaches a certain value. This type of information can be maintained through a database trigger that activates a handler for that trigger at the appropriate moment.



Layers – Data Level – Relational Databases

- In most business-oriented environments, the data level is organized as a relational database
- The data should be organized independently of the applications
- Using relational databases in the client-server model helps separate the processing level from the data level, as processing and data are considered independent
- The relational databases are not always the ideal choice
- If data operations are manipulations with objects - the data level is better to implement by an object-oriented or object-relational database

In most business-oriented environments, the data level is organized as a relational database.

Data independence is crucial here. The data are organized independent of the applications in such a way that changes in that organization do not affect applications, and neither do the applications affect the data organization.

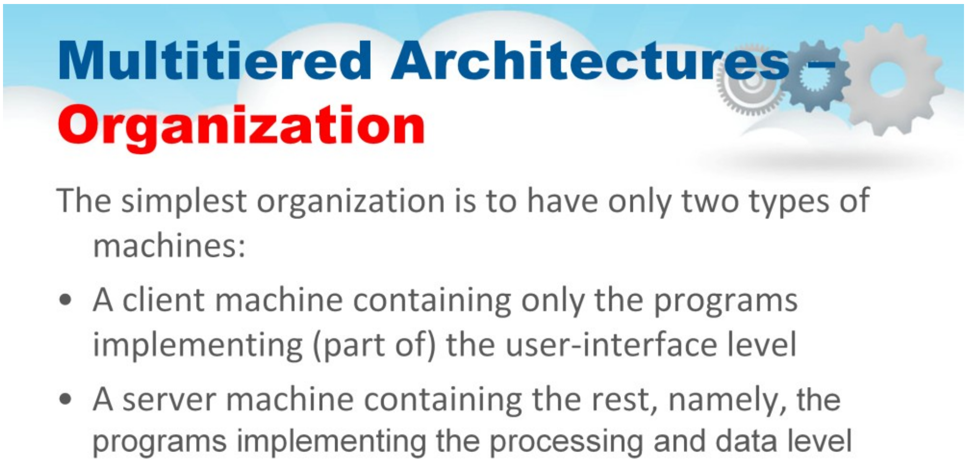
Using relational databases in the client-server model helps separate the processing level from the data level, as processing and data are considered independent.

However, relational databases are not always the ideal choice. A characteristic feature of many applications is that they operate on complex data types that are more easily modeled in terms of objects than in terms of relations. Examples of such data types range from simple polygons and circles to representations of aircraft designs, as is the case with computer-aided design (CAD) systems.

In those cases where data operations are more easily expressed in terms of object manipulations, it makes sense to implement the data level by means of an object-oriented or object-relational database. Notably the latter type has gained popularity as these databases build upon the widely dispersed relational data model, while offering the advantages that object-orientation gives.

System Architectures
–
Multitiered Architectures

Multitiered Architectures



Multitiered Architectures – Organization

The simplest organization is to have only two types of machines:

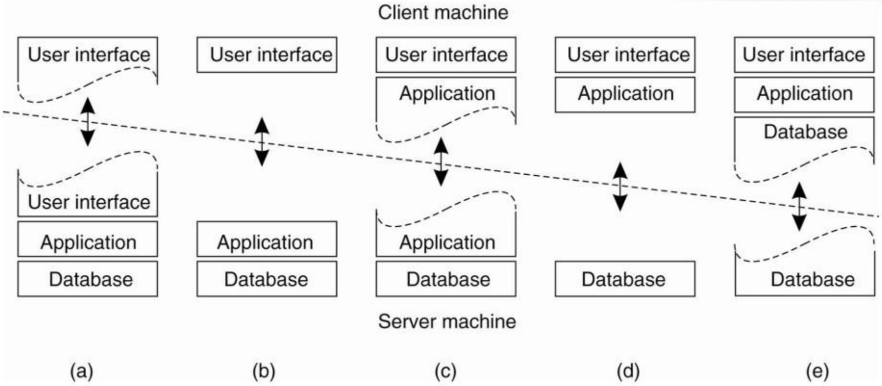
- A client machine containing only the programs implementing (part of) the user-interface level
- A server machine containing the rest, namely, the programs implementing the processing and data level

The distinction into three logical levels as discussed so far, suggests a number of possibilities for physically distributing a client-server application across several machines.

The simplest organization is to have only two types of machines:
1. A client machine containing only the programs implementing (part of) the user-interface level
2. A server machine containing the rest, that is the programs implementing the processing and data level

In this organization everything is handled by the server while the client is essentially no more than a dumb terminal, possibly with a pretty graphical interface. There are many other possibilities, of which we explore some of the more common ones in this section.

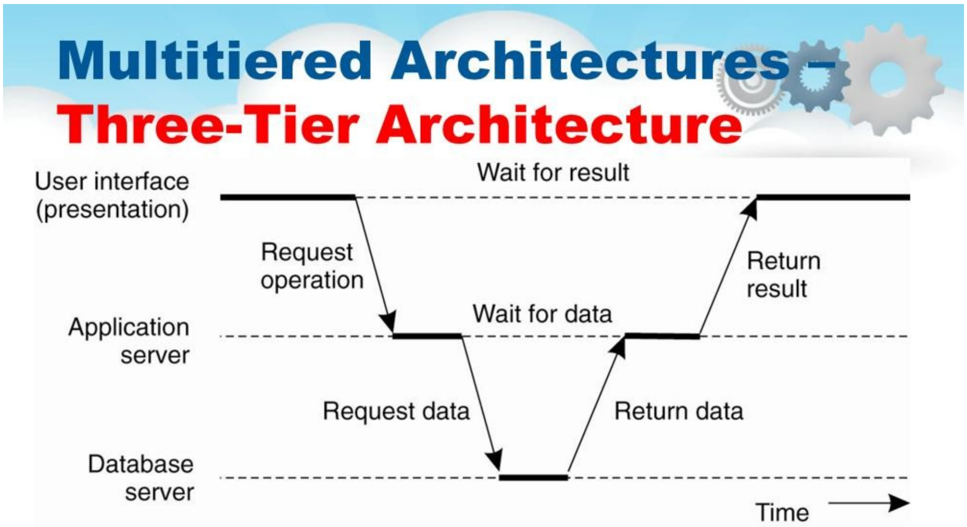
Multitiered Architectures – Two-Tier Architecture



One approach for organizing the clients and servers is to distribute the programs in the application layers of the previous section across different machines, as shown in this Figure. As a first step, we make a distinction between only two kinds of machines: client machines and server machines, leading to what is also referred to as a (physically) **two-tiered** architecture.

1) One possible organization is to have only the terminal-dependent part of the user interface on the client machine, as shown in Figure (a), and give the applications remote control over the presentation of their data.
2) An alternative is to place the entire user-interface software on the client side, as shown in Figure (b). In such cases, we essentially divide the application into a graphical front end, which communicates with the rest of the application (residing at the server) through an application-specific protocol. In this model, the front end (the client software) does no processing other than necessary for presenting the application's interface.
3) Continuing along this line of reasoning, we may also move part of the application to the front end, as shown in Figure (c). An example where this makes sense is where the application makes use of a form that needs to be filled in entirely before it can be processed. The front end can then check the correctness and consistency of the form, and where necessary interact with the user. Another example of the organization of Figure (c), is that of a word processor in which the basic editing functions execute on the client side where they operate on locally cached, or in-memory data. but where the advanced support tools such as checking the spelling and grammar execute on the server side.
4) In many client-server environments, the organizations shown in Figure (d) and Figure (e) are particularly popular. These organizations are used where the client machine is a PC or workstation, connected through a network to a distributed file system or database. Essentially, most of the application is running on the client machine, but all operations on files or database entries go to the server. For example, many banking applications run on an end- user's machine where the user prepares transactions and such. Once finished, the application contacts the database on the bank's server and uploads the transactions for further processing.
5) Figure (e) represents the situation where the client's local disk contains part of the data. For example, when browsing the Web, a client can gradually build a huge cache on local disk of most recent inspected Web pages.

We note that for a few years there has been a strong trend to move away from the configurations shown in Figure (d) and Figure (e) in those case that client software is placed at end-user machines. In these cases, most of the processing and data storage is handled at the server side. The reason for this is simple: although client machines do a lot, they are also more problematic to manage. Having more functionality on the client machine makes client-side software more prone to errors and more dependent on the client's underlying platform (i.e., operating system and resources). From a system's management perspective, having what are called fat clients is not optimal. Instead the thin clients as represented by the organizations shown in Figure (a)-(c) are much easier, perhaps at the cost of less sophisticated user interfaces and client-perceived performance.

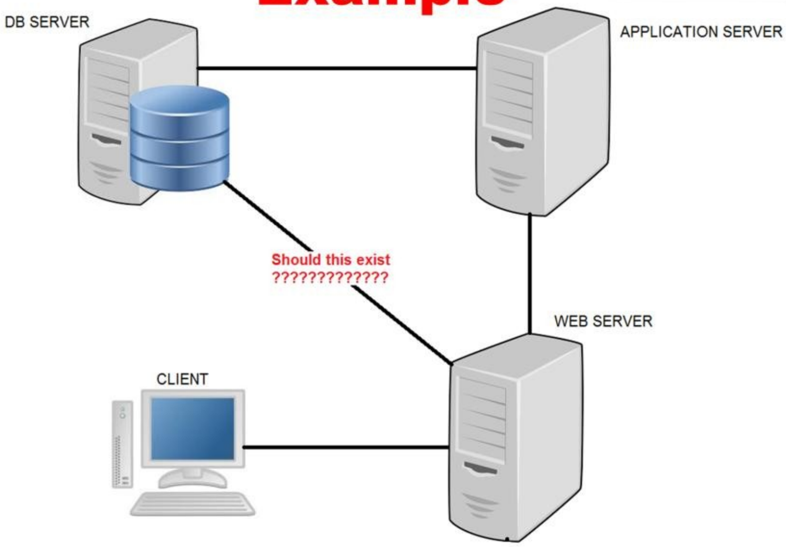


Multitiered Architectures – Three-Tier Architecture

Note that this trend does not imply that we no longer need distributed systems. On the contrary, what we are seeing is that server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines. In particular, when distinguishing only client and server machines as we have done so far, we miss the point that a server may sometimes need to act as a client, as shown in this Figure, leading to a (physically) **three-tiered** architecture.

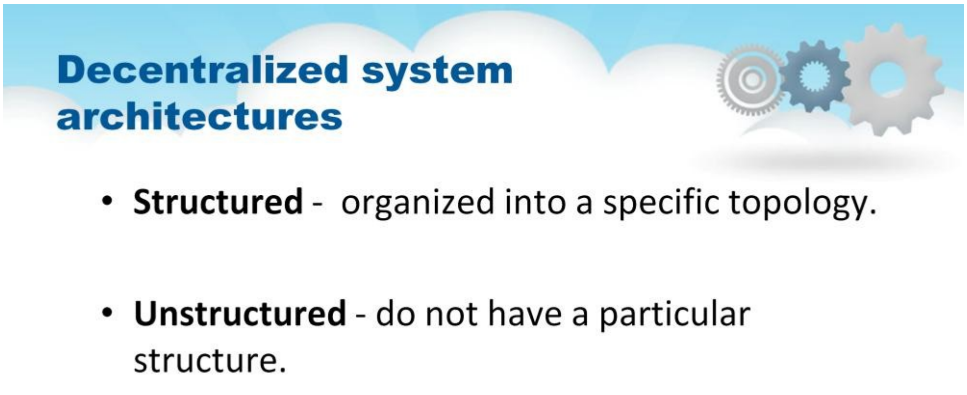
In this architecture, programs that form part of the processing level reside on a separate server, but may additionally be partly distributed across the client and server machines.
A typical example of where a three-tiered architecture is used is in transaction processing, where a separate process, called the transaction processing monitor, coordinates all transactions across possibly different data servers.

Multitiered Architectures – Three-Tier Architecture: Example



Another, but very different example where we often see a three-tiered architecture is in the organization of Web sites. In this case, a Web server acts as an entry point to a site, passing requests to an application server where the actual processing takes place. This application server, in tum, interacts with a database server. For example, an application server may be responsible for running the code to inspect the available inventory of some goods as offered by an electronic bookstore. To do so, it may need to interact with a database containing the raw inventory data.

Decentralized Architectures

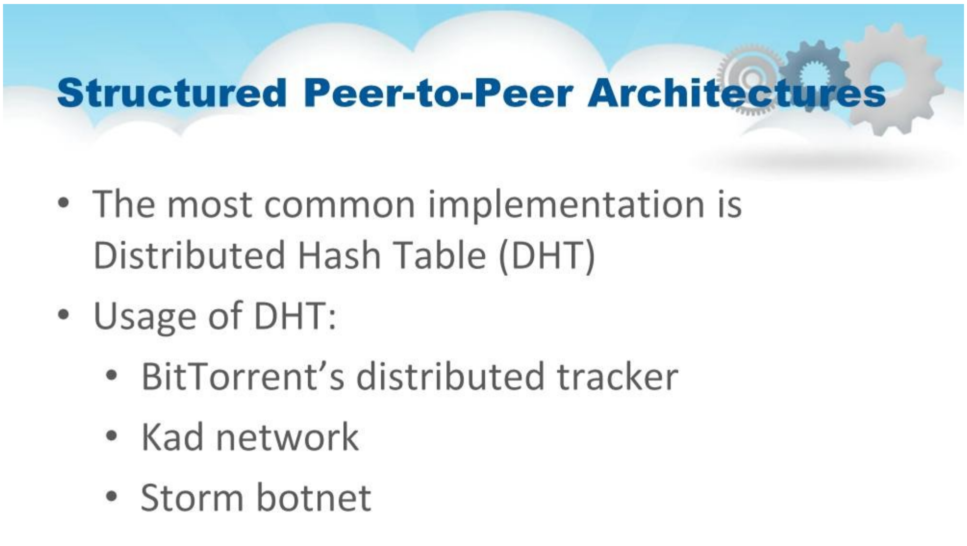


Decentralized system architectures

- **Structured** - organized into a specific topology.
- **Unstructured** - do not have a particular structure.

Structured - organized into a specific topology, and the protocol ensures that any node can efficiently search the network for a file/resource, even if the resource is extremely rare.

Unstructured - do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other.

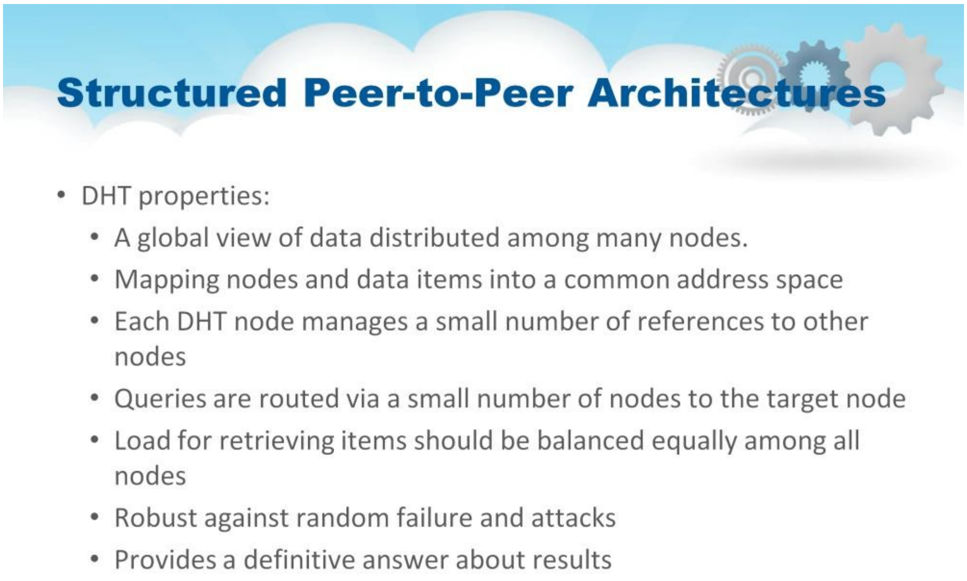


Structured Peer-to-Peer Architectures

- The most common implementation is Distributed Hash Table (DHT)
- Usage of DHT:
  - BitTorrent’s distributed tracker
  - Kad network
  - Storm botnet

In a structured peer-to-peer architecture, the overlay network is constructed using a deterministic procedure.

The most-used procedure is to organize the processes through a distributed hash table (DHT). In a DHT -based system, data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier. Likewise, nodes in the system are also assigned a random number from the same identifier space.



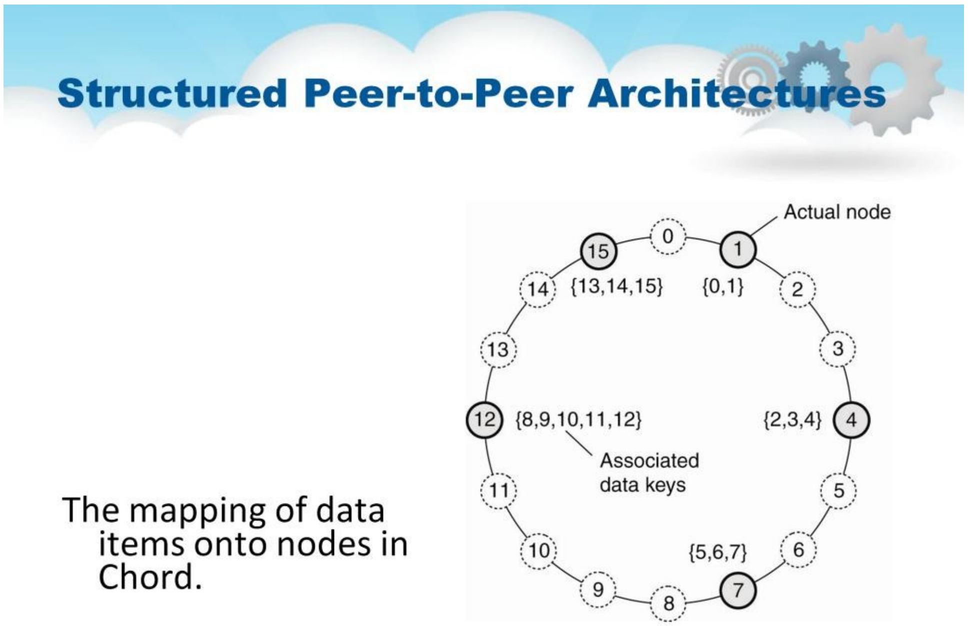
Structured Peer-to-Peer Architectures

- DHT properties:
  - A global view of data distributed among many nodes.
  - Mapping nodes and data items into a common address space
  - Each DHT node manages a small number of references to other nodes
  - Queries are routed via a small number of nodes to the target node
  - Load for retrieving items should be balanced equally among all nodes
  - Robust against random failure and attacks
  - Provides a definitive answer about results

Here some properties of DHT-based systems are listed.
- A global view of data distributed among many nodes.
- Mapping nodes and data items into a common address space
- Each DHT node manages a small number of references to other nodes
- Queries are routed via a small number of nodes to the target node
- Load for retrieving items should be balanced equally among all nodes
- Robust against random failure and attacks
- Provides a definitive answer about results

DHT-based system should implement an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node based on some distance metric.

Most importantly, when looking up a data item, the network address of the node responsible for that data item is returned. Effectively, this is accomplished by routing a request for a data item to the responsible node.

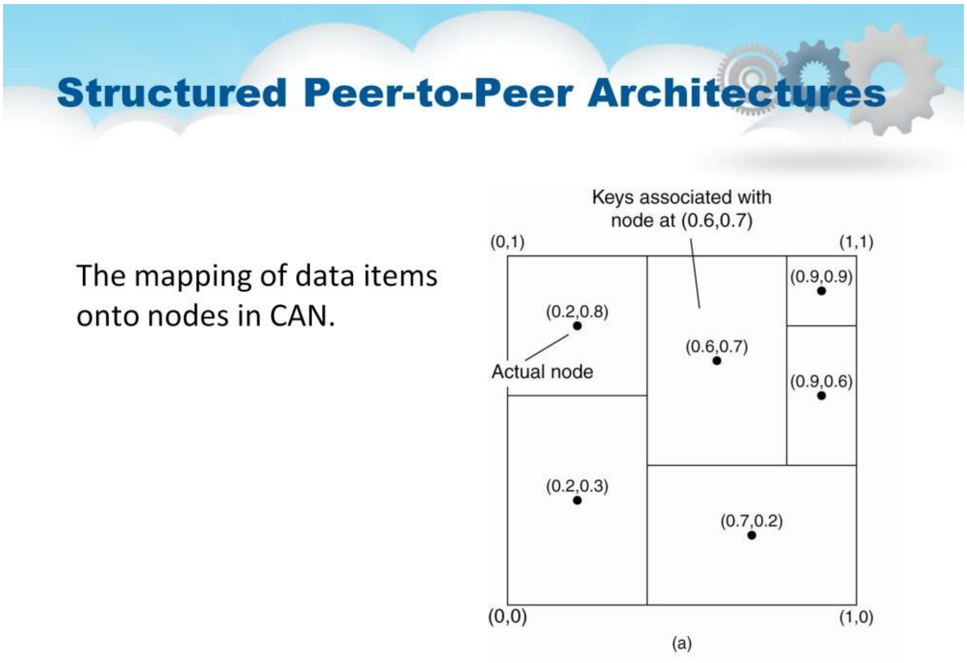


Structured Peer-to-Peer Architectures

The mapping of data items onto nodes in Chord.

For example, in the Chord system proposed by Stoica in 2003 the nodes are logically organized in a ring of nodes. This node is referred to as the successor of key k and denoted as succ(k).

To look up the data item, an application running on an arbitrary node would then call the function LOOKUP(k) which would subsequently return the network address of succ(k). Tnen, the application can contact the node to obtain a copy of the data item.



Structured Peer-to-Peer Architectures

The mapping of data items onto nodes in CAN.

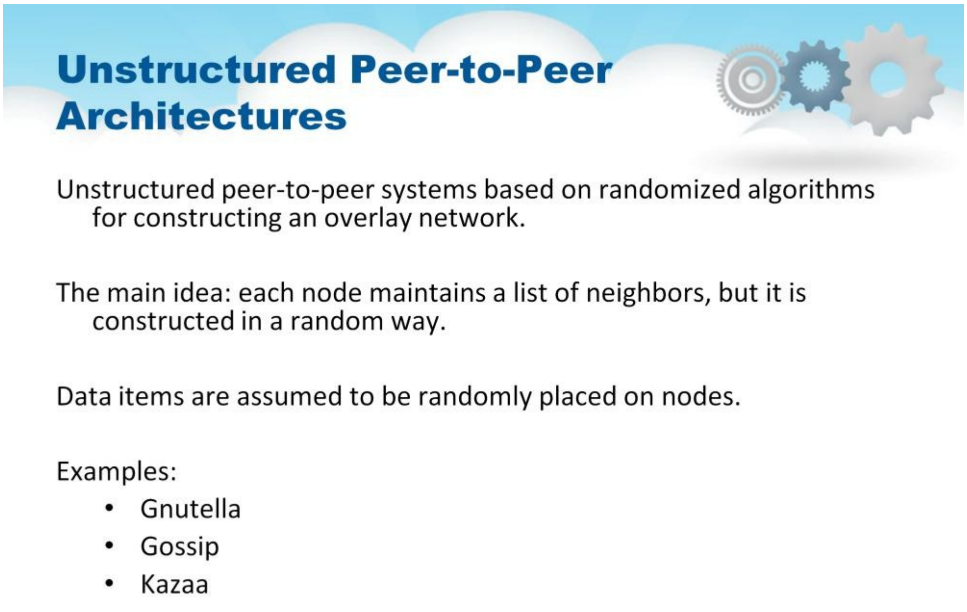
Another approaches are proposed in other DHT-based systems.

As an example, here Content Addressable Network (CAN) is shown, which was proposed by Ratnasamy in 2001.

CAN uses a d-dimensional Cartesian coordinate space, which is completely partitioned among all the nodes that participate in thesystem.

For simplicity let us consider only the 2-dimensional case.

This slide shows how the two-dimensional space is divided among six nodes. Each node has an associated region. Every data item in CAN will be assigned a unique point in this space, after which it is also clear which node is responsible for that data (ignoring data items that fall on the border of multiple regions, for which a deterministic assignment rule is used).



Unstructured Peer-to-Peer Architectures

Unstructured peer-to-peer systems based on randomized algorithms for constructing an overlay network.

The main idea: each node maintains a list of neighbors, but it is constructed in a random way.

Data items are assumed to be randomly placed on nodes.

Examples:

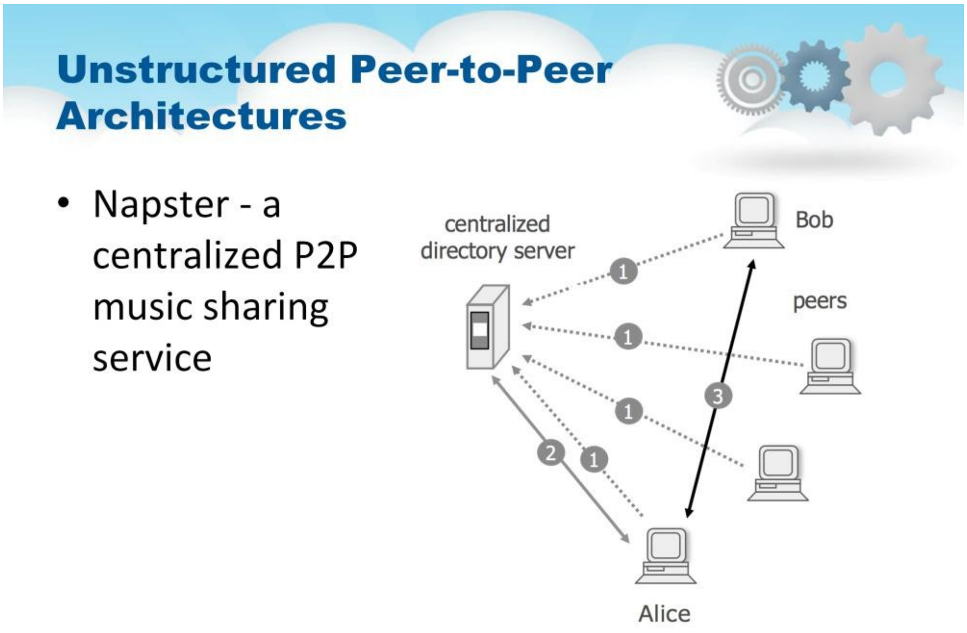
- Gnutella
- Gossip
- Kazaa

Unstructured peer-to-peer systems largely rely on randomized algorithms for constructing an overlay network.

The main idea is that each node maintains a list
of neighbors, but that this list is constructed in a more or less random way.

Data items are assumed to be randomly placed on nodes. As a consequence, when a node needs to locate a specific data item, the only thing it can effectively do is flood the network with a search query.

Unstructured networks utilize flooding and similar opportunistic techniques, such as random walks, expanding-ring, Time-to-Live (TTL) search, in order to locate peers that have interesting data items.

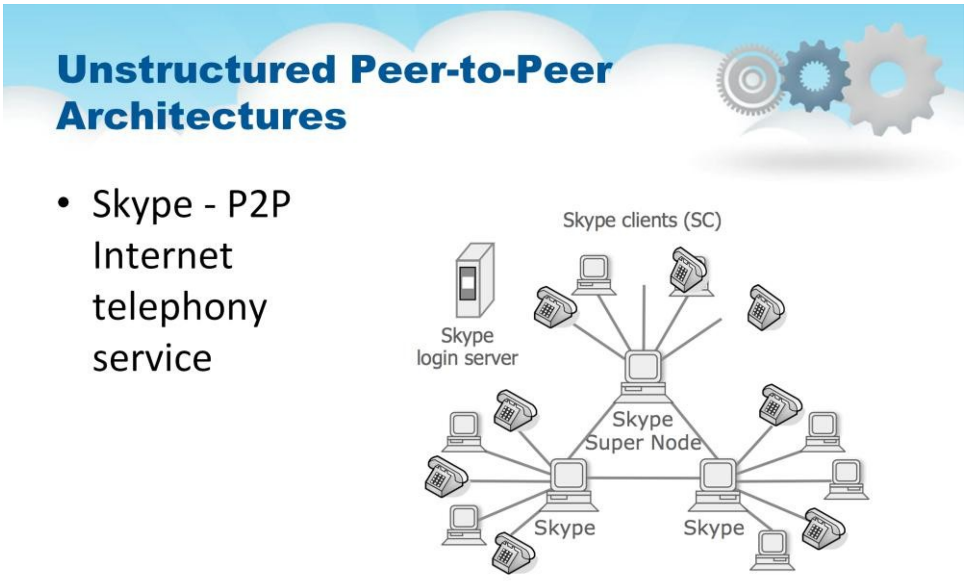


Unstructured Peer-to-Peer Architectures

- Napster - a centralized P2P music sharing service

Napster is the name for notoriously famous music-focused online services.
It was founded as a pioneering peer-to-peer (P2P) file sharing Internet service for sharing digital audio files, typically audio songs, encoded in MP3 format.
Here the principal scheme of Napster is shown.

It was shut down by court order.
Later companies and projects successfully followed its P2P file sharing example such as Gnutella, Freenet, Kazaa, BearShare, and many others.



Unstructured Peer-to-Peer Architectures

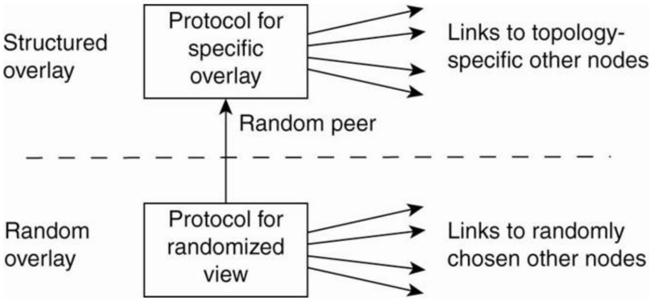
- Skype - P2P Internet telephony service

Skype uses a proprietary Internet telephony (VoIP) network called the Skype protocol.
Part of the Skype technology relies on the Global Index P2P protocol.
The main difference between Skype and standard VoIP clients is that Skype operates on a peer-to-peer model (originally based on the Kazaa software), rather than the more usual client–server model.
Here the principal scheme of Skype P2P model is shown.

Note: The very popular Session Initiation Protocol (SIP) model of VoIP is also peer-to-peer, but implementation generally requires registration with a server, as does Skype.

Note: On 20 June 2014, Microsoft announced the deprecation of the old Skype protocol. The new Skype protocol—Microsoft Notification Protocol 24.

Topology Management of Overlay Networks


Structured overlay
Protocol for specific overlay
Links to topology-specific other nodes
Random peer
Random overlay
Protocol for randomized view
Links to randomly chosen other nodes

A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

It is possible to construct and maintain specific topologies of overlay networks.

This topology management is achieved by adopting a two-layered approach, as shown in this slide.

The lowest layer constitutes an unstructured peer-to-peer system in which nodes periodically exchange entries of their partial views with the aim to maintain an accurate random graph.

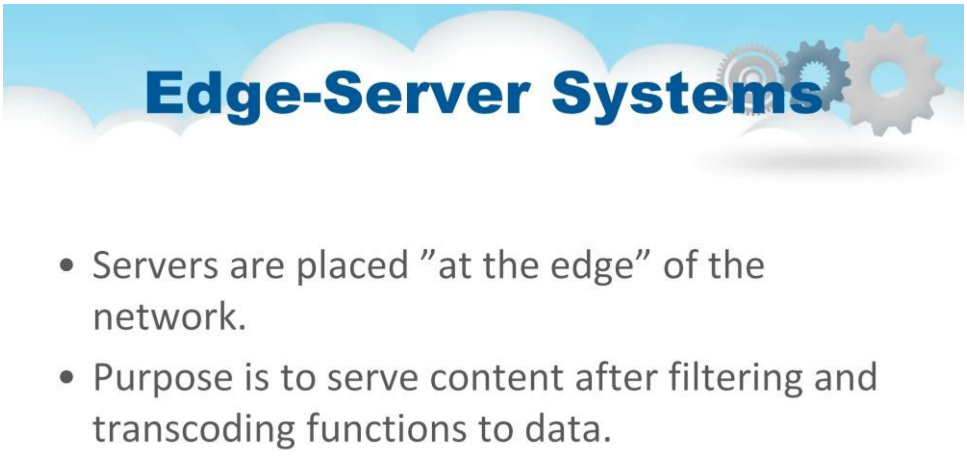
The lowest layer passes its partial view to the higher layer, where an additional selection of entries takes place. This then leads to a second list of neighbors corresponding to the desired topology.

System Architectures

Hybrid Architectures

Hybrid Architectures

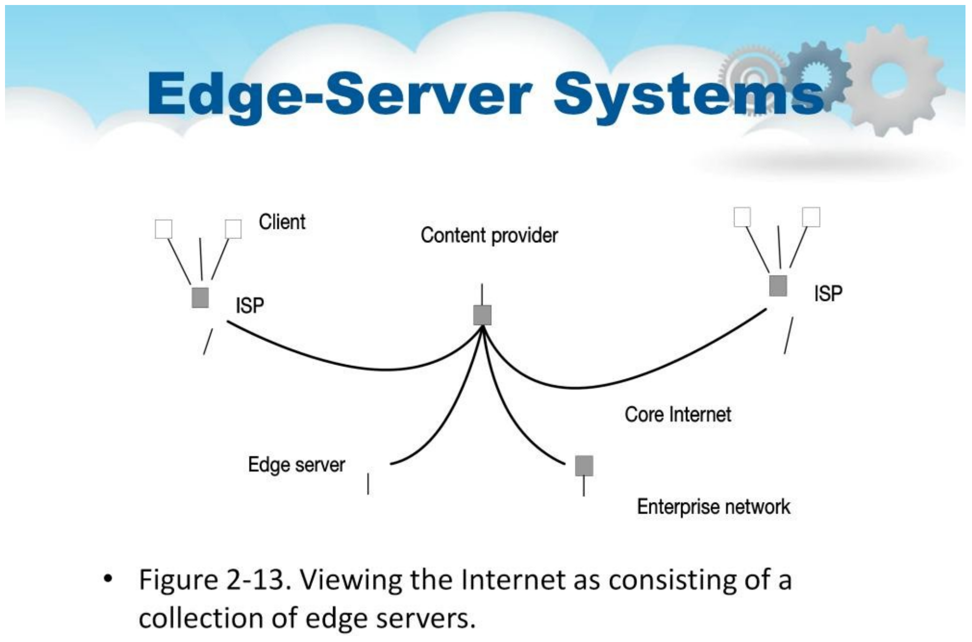
So far, we have focused on client-server architectures and a number of peer-to-peer architectures. Many distributed systems combine architectural features, as we already came across in super-peer networks. In this section we take a look at some specific classes of distributed systems in which client-server solutions are combined with decentralized architectures.



Edge-Server Systems

- Servers are placed ”at the edge” of the network.
- Purpose is to serve content after filtering and transcoding functions to data.

An important class of distributed systems that is organized according to a hybrid architecture is formed by edge-server systems. These systems are deployed on the Internet where servers are placed "at the edge" of the network.

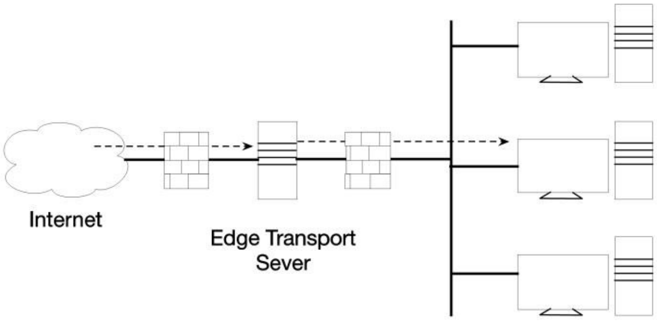


Edge-Server Systems

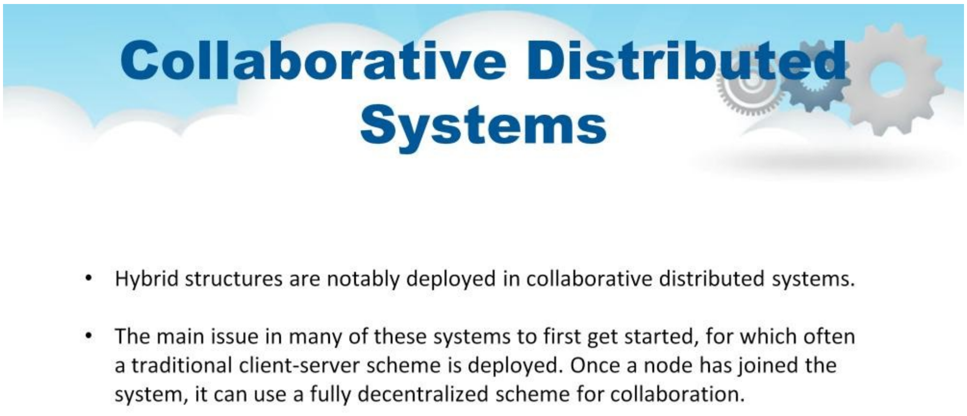
- Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

This edge is formed by the boundary between enterprise networks and the actual Internet, for example, as provided by an Internet Service Provider (ISP). Likewise, where end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet. This leads to a general organization as shown in this slide.

Edge-Server Systems



End users, or clients in general, connect to the Internet by means of an edge server. The edge server's main purpose is to serve content, possibly after applying filtering and transcoding functions. More interesting is the fact that a collection of edge servers can be used to optimize content and application distribution. The basic model is that for a specific organization, one edge server acts as an origin server from which all content originates. That server can use other edge servers for replicating Web pages and such.

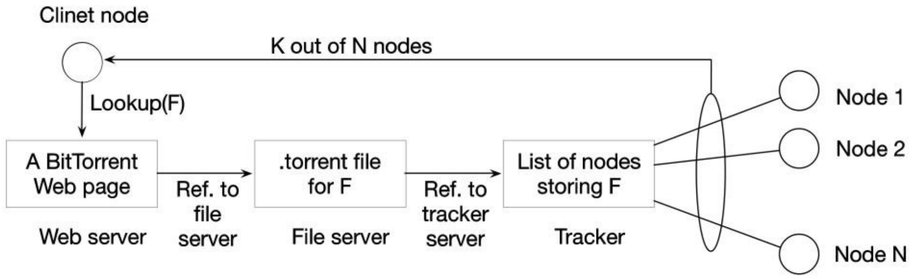


Collaborative Distributed Systems

- Hybrid structures are notably deployed in collaborative distributed systems.
- The main issue in many of these systems to first get started, for which often a traditional client-server scheme is deployed. Once a node has joined the system, it can use a fully decentralized scheme for collaboration.

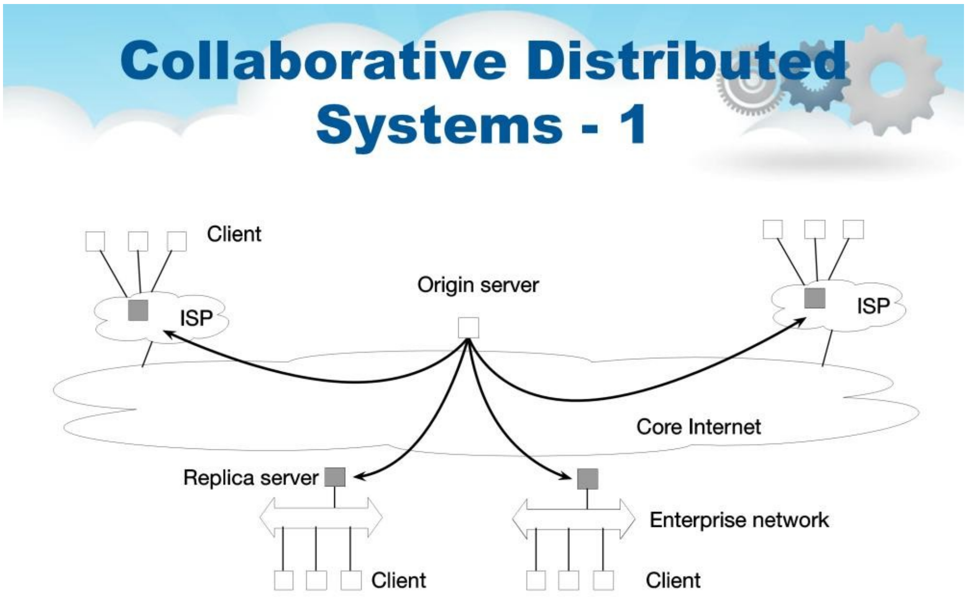
Hybrid structures are notably deployed in collaborative distributed systems. The main issue in many of these systems to first get started, for which often a traditional client-server scheme is deployed. Once a node has joined the system, it can use a fully decentralized scheme for collaboration.

Collaborative Distributed Systems (1)



The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

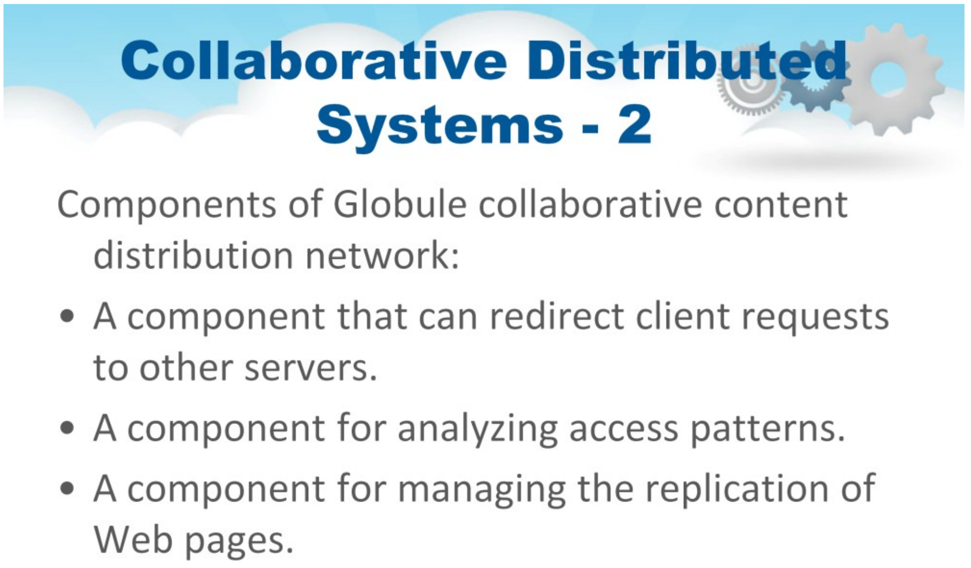
Let ‘s consider the BitTorrent file-sharing system.
BitTorrent is a peer-to-peer file downloading system. Its principal working is shown in this slide.
The basic idea is that when an end user is looking for a file, he downloads chunks of the file from other users until the downloaded chunks can be assembled together yielding the complete file. An important design goal was to ensure collaboration. In most file-sharing systems, a significant fraction of participants merely download files but otherwise contribute close to nothing. To this end, a file can be downloaded only when the downloading client is providing content to someone else.
To download a user needs to access a global directory, which is just one of a few well-known Web sites. Such a directory contains references to what are called .torrent files. A .torrent file contains the information that is needed to download a specific file. In particular, it refers to what is known as a tracker, which is a server that is keeping an accurate account of active nodes that have (chunks) of the requested file. An active node is one that is currently downloading another file.
Obviously, there will be many different trackers, although (there will generally be only a single tracker per file (or collection of files). Once the nodes have been identified from where chunks can be downloaded, the downloading node effectively becomes active.



Collaborative Distributed Systems - 1

Clearly, BitTorrent combines centralized with decentralized solutions. As it turns out, the bottleneck of the system is, not surprisingly, formed by the trackers. As another example, consider the Globule collaborative content distribution network propsoed by Pierre and van Steen in 2006.

Globule strongly resembles the edgeserver architecture mentioned above. In this case, instead of edge servers, end users (but also organizations) voluntarily provide enhanced Web servers that are capable of collaborating in the replication of Web pages.



Collaborative Distributed Systems - 2

Components of Globule collaborative content distribution network:

- A component that can redirect client requests to other servers.
- A component for analyzing access patterns.
- A component for managing the replication of Web pages.

Globule is a decentralized distributed system.
Requests for Web site are initially forwarded to server, at which point they may be redirected to one of the other servers. Distributed redirection is also supported.

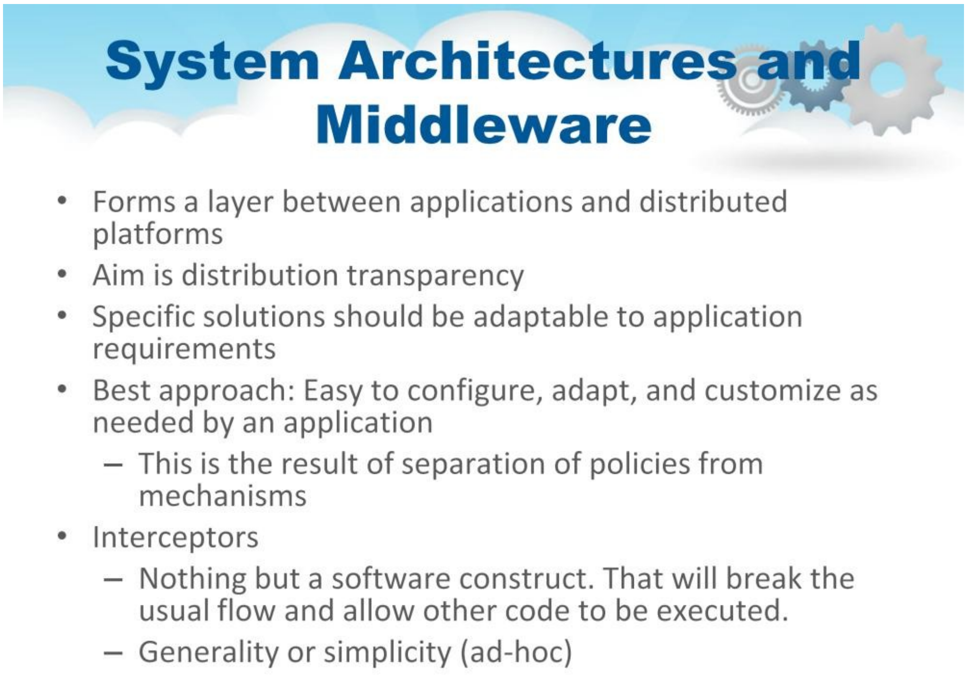
Globule also has a centralized component in the form of its broker. The broker is responsible for registering servers, and making these servers known to others. Servers communicate with the broker completely analogous to what one would expect in a client-server system. For reasons of availability, the broker can be replicated, but this type of replication is widely applied in order to achieve reliable client-server computing.

System Architectures

Middleware

Middleware

When considering the architectural issues we have discussed so far, a question that comes to mind is where middleware fits in. As we discussed before, middleware forms a layer between applications and distributed platforms. An important purpose is to provide a degree of distribution transparency, that is, to a certain extent hiding the distribution of-data, processing, and control from applications.



System Architectures and Middleware

- Forms a layer between applications and distributed platforms
- Aim is distribution transparency
- Specific solutions should be adaptable to application requirements
- Best approach: Easy to configure, adapt, and customize as needed by an application
  - This is the result of separation of policies from mechanisms
- Interceptors
  - Nothing but a software construct. That will break the usual flow and allow other code to be executed.
  - Generality or simplicity (ad-hoc)

Middleware systems actually follow a specific architectural style.

For example, many middleware solutions have adopted an object-based architectural style, such as CORBA. But others, like TIB/Rendezvous provide middleware that follows the event-based architectural style.

If middleware is molded according to a specific architectural style, then it has the benefit that designing applications may become simpler. However, an obvious drawback is that the middleware may no longer be optimal for what an application developer had in mind.

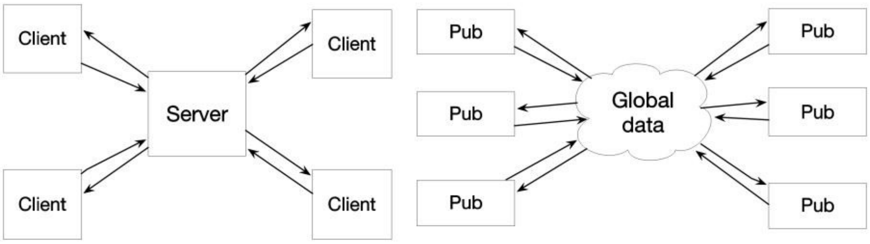
For example, some middleware can initially offer only objects that could be invoked by remote clients. Later, this form of interaction became too restrictive, so that other interaction patterns such as messaging were added. Obviously, adding new features can easily lead to awkward middleware solutions. In addition, middleware should provide distribution transparency, and specific solutions should be adaptable to application requirements.

One solution to this problem is to make several versions of a middleware system, where each version is tailored to a specific class of applications.

But the better approach is to make middleware systems such that they are easy to configure, adapt, and customize as needed by an application. As a result, systems are now being developed in which a stricter separation between policies and mechanisms is being made.

Let’s take a look at some of the commonly followed approaches.

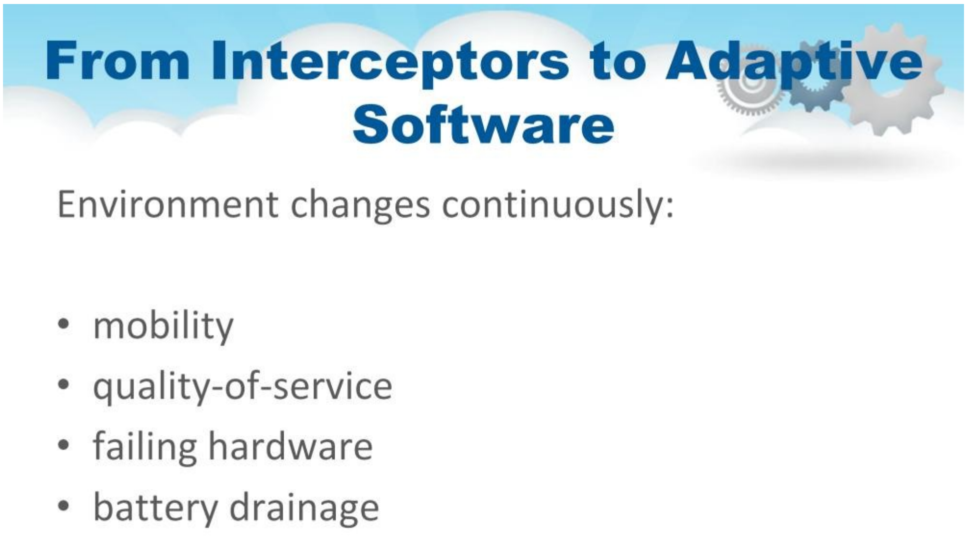
Interceptors



Examples: CORBA, Java RMI, COM+, web SVCs

Examples: TIBCO, Rendevous, JMS

Conceptually, an interceptor is nothing but a software construct that will break the usual flow of control and allow other (application specific) code to be executed (like it is shown in this slide). To make interceptors generic may require a substantial implementation effort, and it is unclear whether in such cases generality should be preferred over restricted applicability and simplicity. Also, in many cases having only limited interception facilities will improve management of the software and the distributed system as a whole.



From Interceptors to Adaptive Software

Environment changes continuously:

- mobility
- quality-of-service
- failing hardware
- battery drainage

What interceptors actually offer is a means to adapt the middleware. The need for adaptation comes from the fact that the environment in which distributed applications are executed changes continuously. Changes include those resulting from mobility, a strong variance in the quality-of-service of networks, failing hardware, and battery drainage, amongst others. Rather than making applications responsible for reacting to changes, this task is placed in the middleware. These strong influences from the environment have brought many designers of middleware to consider the construction of adaptive software.

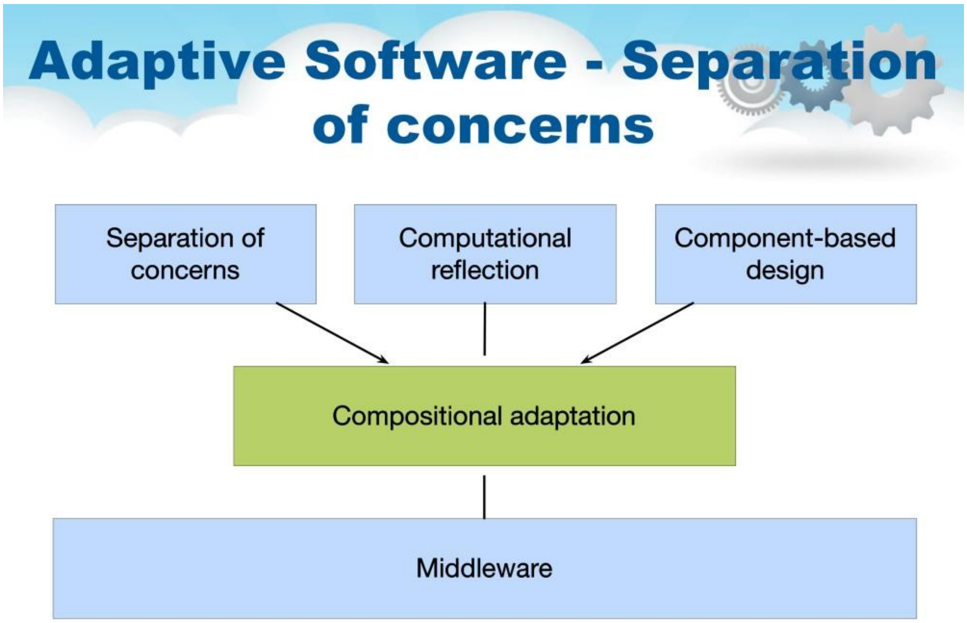
General Approaches to Adaptive Software

Three basic approaches to adaptive software:

- Separation of concerns
- Computational reflection
- Component-based design

However, adaptive software has not been as successful as anticipated. As many researchers and developers consider it to be an important aspect of modern distributed systems, let us briefly pay some attention to it.

The following three basic techniques are used for software adaptation:
- Separation of concerns
- Computational reflection
- Component-based design



Adaptive Software - Separation of concerns

Separation of concerns

Computational reflection

Component-based design

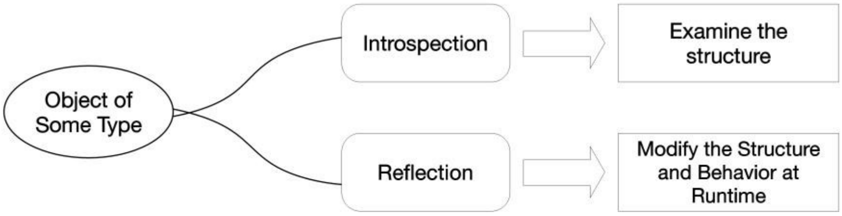
Compositional adaptation

Middleware

Separating Concerns
It relates to the traditional way of modularizing systems: separate the parts that implement functionality from those that take care of other things (known as extra functionalities) such as reliability, performance, security, etc.

But developing middleware for distributed applications is largely about handling extra functionalities independent from applications. The main problem is that we cannot easily separate these extra functionalities by means of modularization. For example, simply putting security into aseparate module is not going to work. Likewise, it is hard to imagine how fault tolerance can be isolated into a separate box and sold as an independent service. Separating and subsequently weaving these cross-cutting concerns into a (distributed) system is the major theme addressed by aspect-oriented software development. However, aspect orientation has not yet been successfully applied to developing large-scale distributed systems, and it can be expected that there is still a long way to go before it reaches thatstage.

Adaptive Software – Computational reflection



Object of Some Type

Introspection

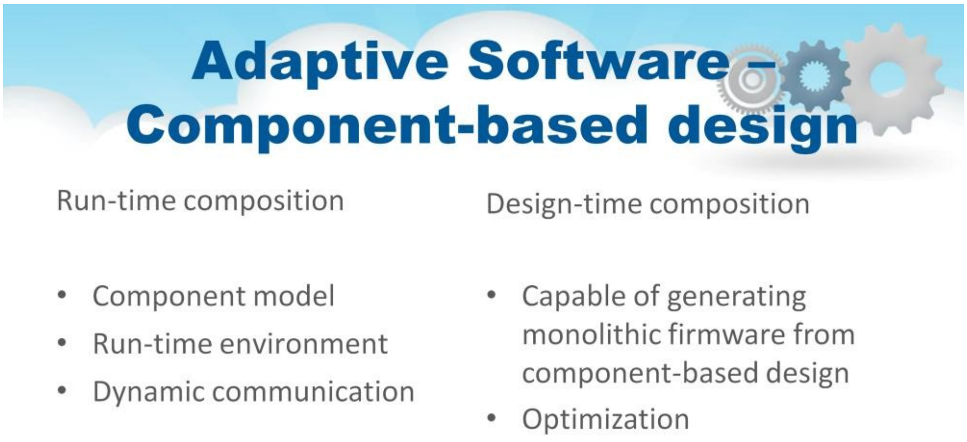
Examine the structure

Reflection

Modify the Structure and Behavior at Runtime

Computational Reflection

It refers to the ability of a program to inspect itself and, if necessary, adapt its behavior. Reflection has been built into programming languages, including Java, and offers a powerful facility for runtime modifications. In addition, some middleware systems provide the means to apply reflective techniques. However, just as in the case of aspect orientation, reflective middleware has yet to prove itself as a powerful tool to manage the complexity of large-scale distributed systems.



Adaptive Software – Component-based design

Run-time composition

- Component model
- Run-time environment
- Dynamic communication

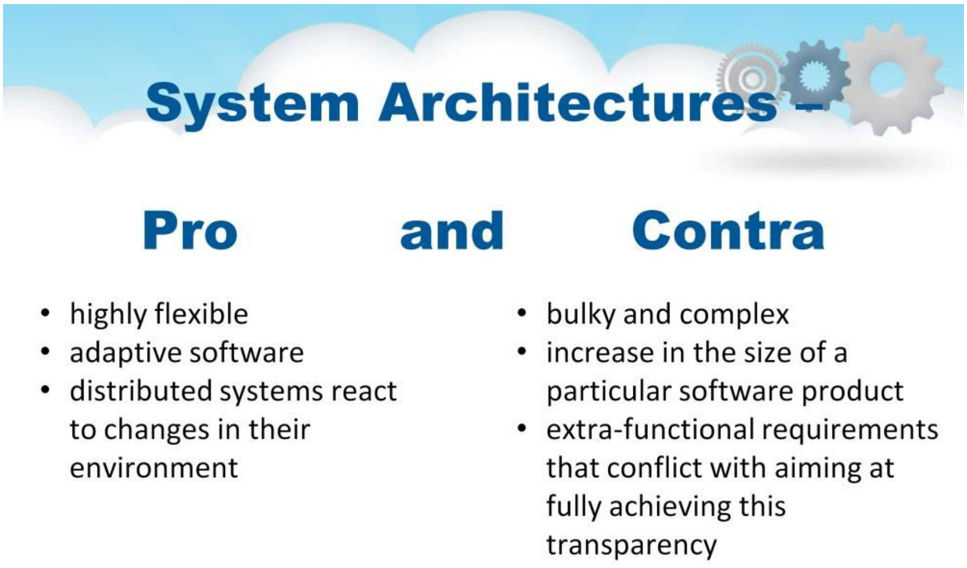
Design-time composition

- Capable of generating monolithic firmware from component-based design
- Optimization

Component-Based Design

It supports adaptation through composition. A system may either be configured statically at design time, or dynamically at runtime. The latter requires support for late binding, a technique that has been successfully applied in programming language environments, but also for operating systems where modules can be loaded and unloaded at will.

Research for automatic selection of the best implementation of a component during runtime is carried out now, but again, the process remains complex for distributed systems, especially when considering that replacement of one component requires knowning what the effect of that replacement on other components will be.



System Architectures –

Pro and Contra

Pro

- highly flexible
- adaptive software
- distributed systems react to changes in their environment

Contra

- bulky and complex
- increase in the size of a particular software product
- extra-functional requirements that conflict with aiming at fully achieving this transparency

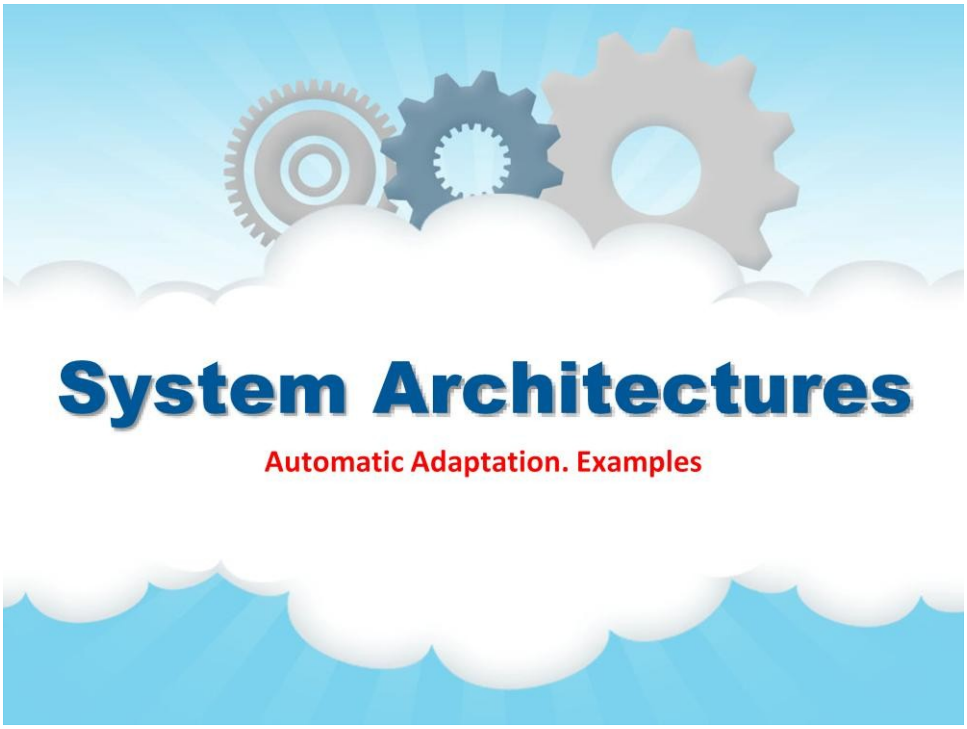
Here the comparison of advantages and disadvantages of software architecture for distributed systems are listed.

Software architectures for distributed systems (middleware) are bulky and complex. In large part, this bulkiness and complexity arises from the need to be general in the sense that distribution transparency needs to be provided. Applications have specific extra-functional requirements that conflict with aiming at fully achieving this transparency. These conflicting requirements for generality and specialization have resulted in middleware solutions that are highly flexible.

Now all large software systems are required to execute in a networked environment, and so the complexity of distributed systems becomes an inherent feature and result of attempts to make distribution transparent.

The underlying assumption is that we need adaptive software in the sense that the software should be allowed to change as the environment changes. However, one should question whether adapting to a changing environment is a good reason to adopt changing the software. Faulty hardware, security attacks, energy drainage, and so on, all seem to be environmental influences that can (and should) be anticipated by software.

The strongest, and certainly most valid, argument for supporting adaptive software is that many distributed systems cannot be shut down. This constraint calls for solutions to replace and upgrade components on the fly, but is not clear whether any of the solutions proposed above are the best ones to tackle this maintenance problem.



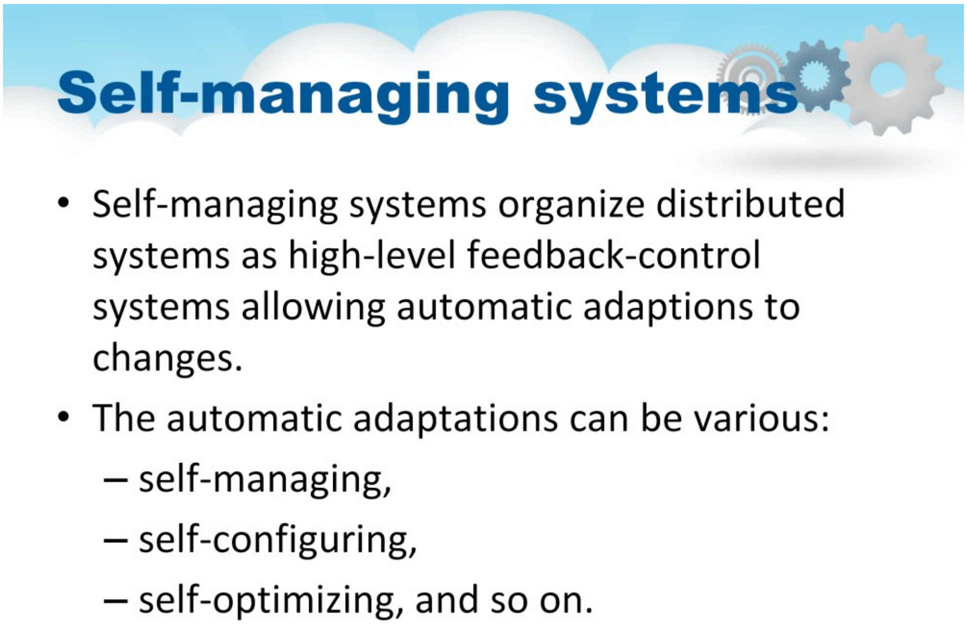
Automatic Adaptation

Distributed systems-and notably their associated middleware-need to provide general solutions toward shielding undesirable features inherent to networking so that they can support as many applications as possible. On the other hand, full distribution transparency is not what most applications actually want, resulting in application-specific solutions that need to be supported as well. We have argued that, for this reason, distributed systems should be adaptive, but notably when it comes to adapting their execution behavior and not the software components they comprise.

General Approaches to Adaptive Software

- What interceptors actually offer is a means to adapt the middleware.
- The need for adaptation comes from the fact that the environment in which distributed applications are executed changes continuously.
- Changes include those resulting from mobility, a strong variance in the quality-of-service of networks, failing hardware, and battery drainage, amongst others.
- Rather than making applications responsible for reacting to changes, this task is placed in the middleware.

What interceptors actually offer is a means to adapt the middleware.
The need for adaptation comes from the fact that the environment in which distributed applications are executed changes continuously.
Changes include those resulting from mobility, a strong variance in the quality-of-service of networks, failing hardware, and battery drainage, amongst others.
Rather than making applications responsible for reacting to changes, this task is placed in the middleware.



Self-managing systems

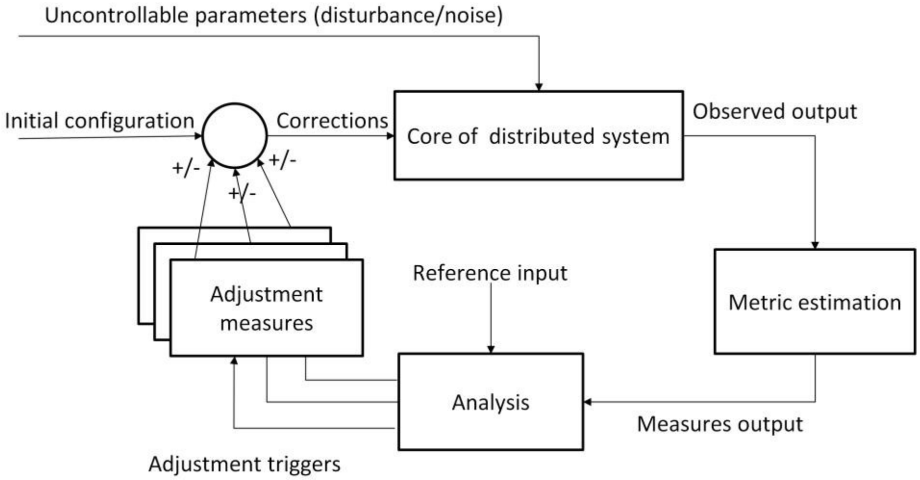
- Self-managing systems organize distributed systems as high-level feedback-control systems allowing automatic adaptions to changes.
- The automatic adaptations can be various:
  - self-managing,
  - self-configuring,
  - self-optimizing, and so on.

Distributed systems should be adaptive, but notably when it comes to adapting their execution behavior and not the software components they comprise. We need to organize the components of a distributed system such that monitoring and adjustments can be done while on the other hand we need to decide where the processes are to be executed that handle the adaptation

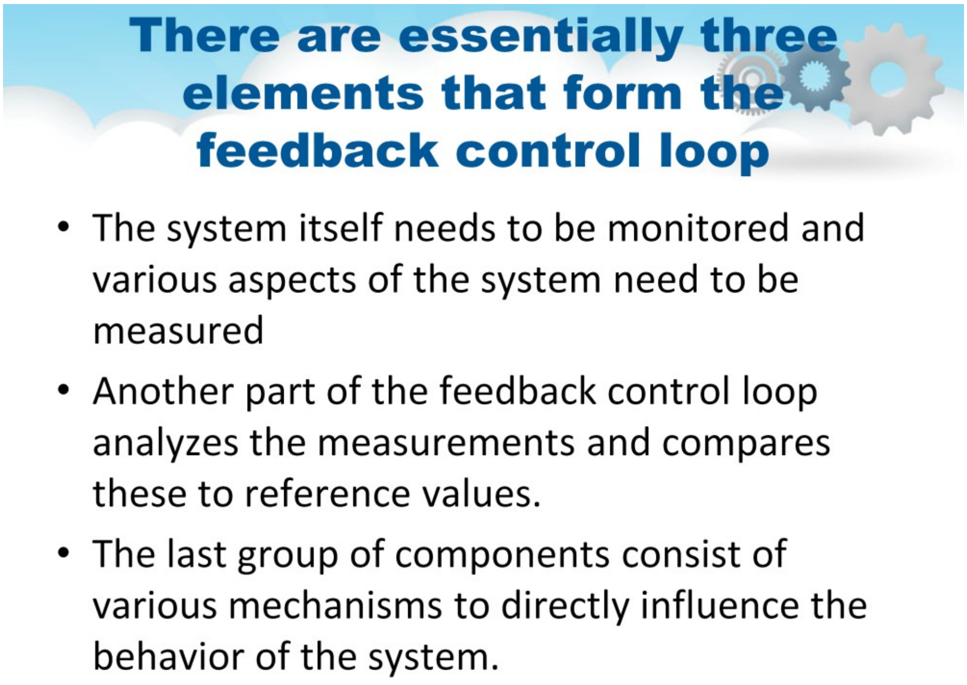
Self-managing systems autonomic computing or self-managing systems are organizing distributed systems as high-level feedback-control systems allowing automatic adaptations to changes.

The variety by which automatic adaptations are being captured are as follows:
self-managing,
self- healing,
self-configuring,
self-optimizing, and so on.

Feedback Control Model



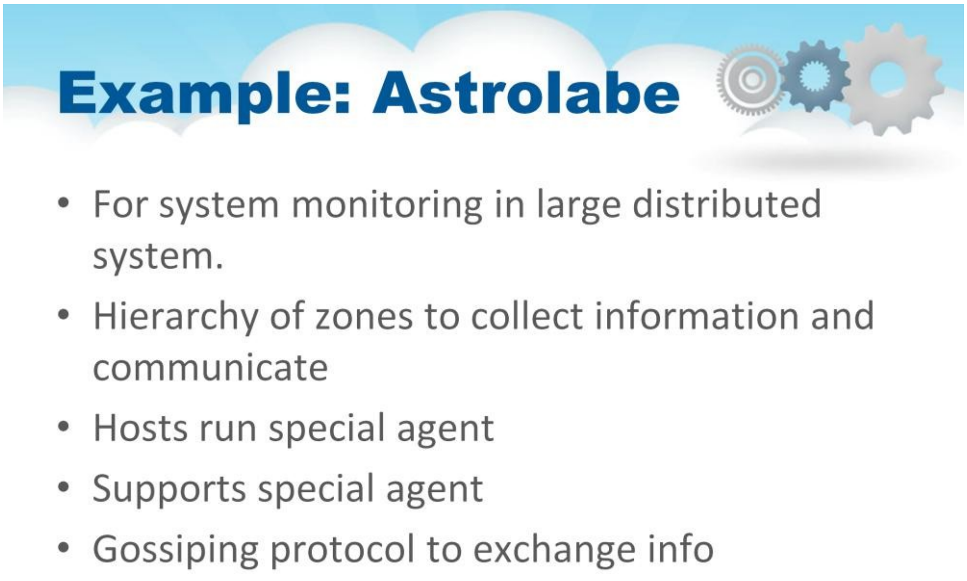
The core of a feedback control system is formed by the components that need to be managed. These components are assumed to be driven through controllable input parameters, but their behavior may be influenced by all kinds of uncontrollable input, also known as disturbance or noise input. Although disturbance will often come from the environment in which a distributed system is executing, it may well be the case that unanticipated component interaction causes unexpected behavior.



There are essentially three elements that form the feedback control loop

- The system itself needs to be monitored and various aspects of the system need to be measured
- Another part of the feedback control loop analyzes the measurements and compares these to reference values.
- The last group of components consist of various mechanisms to directly influence the behavior of the system.

There are essentially three elements that form the feedback control loop.
1 - First, the system itself needs to be monitored, which requires that various aspects of the system need to be measured. In many cases, measuring behavior is hard to organize. For example, round-trip delays in the Internet may vary wildly, and also depend on what exactly is being measured. In such cases, accurately estimating a delay may be difficult, and for these reasons, a feedback control loop generally contains a logical metric estimation component.
2 - Another part of the feedback control loop analyzes the measurements and compares these to reference values. This feedback analysis component forms the heart of the control loop, as it will contain the algorithms that decide on possible adaptations.
3 - The last group of components consist of various mechanisms to directly influence the behavior of the system. There can be many different mechanisms: placing replicas, changing scheduling priorities, switching services, moving data for reasons"of availability, redirecting requests to different servers, etc. The analysis component will need to be aware of these mechanisms and their (expected) effect on system behavior. Therefore, itwill trigger one or several mechanisms, to subsequently later observe the effect.

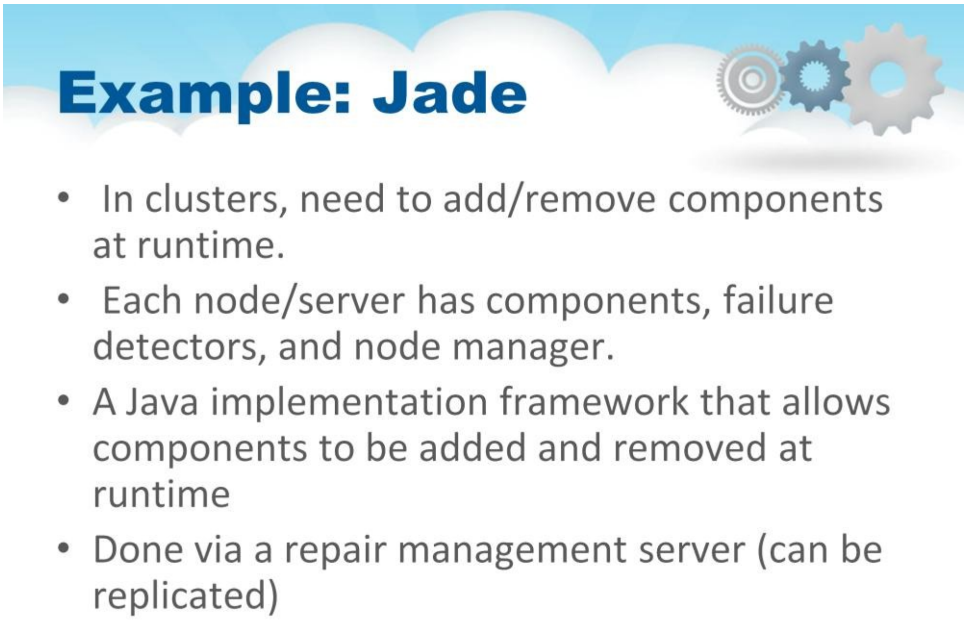


Example: Astrolabe

- For system monitoring in large distributed system.
- Hierarchy of zones to collect information and communicate
- Hosts run special agent
- Supports special agent
- Gossiping protocol to exchange info

For a example, let’s consider Astrolabe (proposed by Van Renesse in 2003), which is a system that can support general monitoring of very large distributed systems. In the context of self-managing systems, Astrolabe is to be positioned as a general tool for observing systems behavior. Its output can be used to feed into an analysis component for deciding on corrective actions. Astrolabe organizes a large collection of hosts into a hierarchy of zones. The lowest-level zones consist of just a single host, which are subsequently grouped into zones of increasing size. The top-level zone covers all hosts. Every host runs an Astrolabe process, called an agent, that collects information on the zones in which that host is contained. The agent also communicates with other agents with the aim to spread zone information across the entire system.

Each host maintains a set of attributes for collecting local information. For example, a host may keep track of specific files it stores, its resource usage, and so on. Only the attributes as maintained directly by hosts, that is, at the lowest level of the hierarchy are writable. Each zone can also have a collection of attributes, but the values of these attributes are computed from the values of lower level zones.



Example: Jade

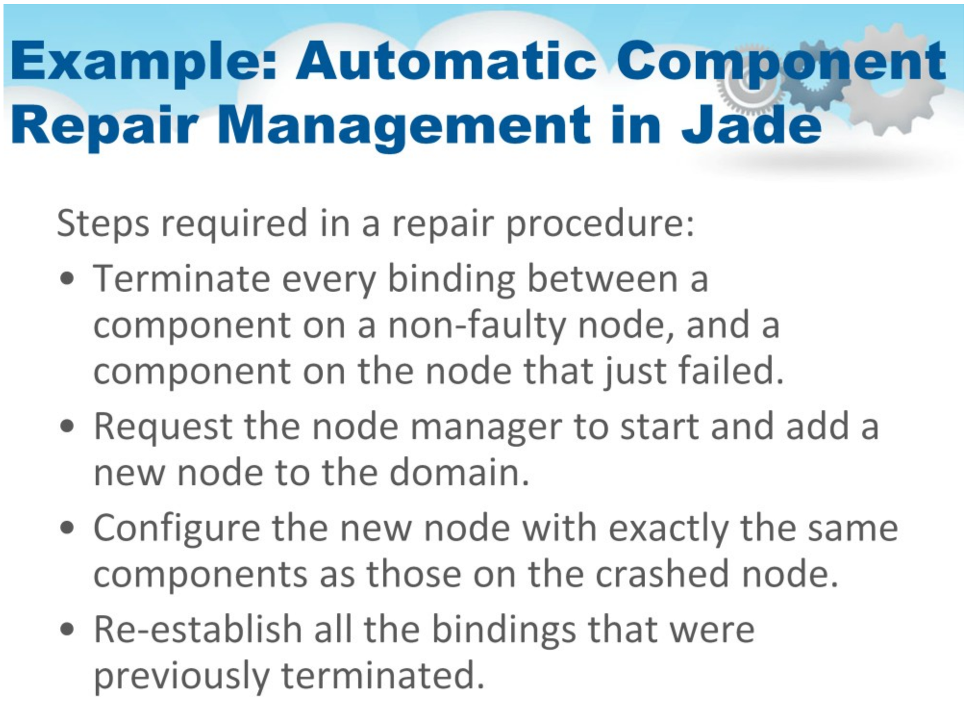
- In clusters, need to add/remove components at runtime.
- Each node/server has components, failure detectors, and node manager.
- A Java implementation framework that allows components to be added and removed at runtime
- Done via a repair management server (can be replicated)

When maintaining clusters of computers, each running sophisticated servers, it becomes important to alleviate management problems. One approach that can be applied to servers that are built using a component-based approach, is to detect component failures and have them automatically replaced. The Jade system follows this approach.

Jade is built on the Fractal component model, a Java implementation of a framework that allows components to be added and removed at runtime.

Jade uses the notion of a repair management domain. Such a domain consists of a number of nodes, where each node represents a server along with the components that are executed by that server. There is a separate node manager which is responsible for adding and removing nodes from the domain. The node manager may be replicated for assuring high availability.

Each node is equipped with failure detectors, which monitor the health of a node or one of its components and report any failures to the node manager. Typically, these detectors consider exceptional changes in the state of component, the usage of resources, and the actual failure of a component. Note that the latter may actually mean that a machine has crashed.



Example: Automatic Component Repair Management in Jade

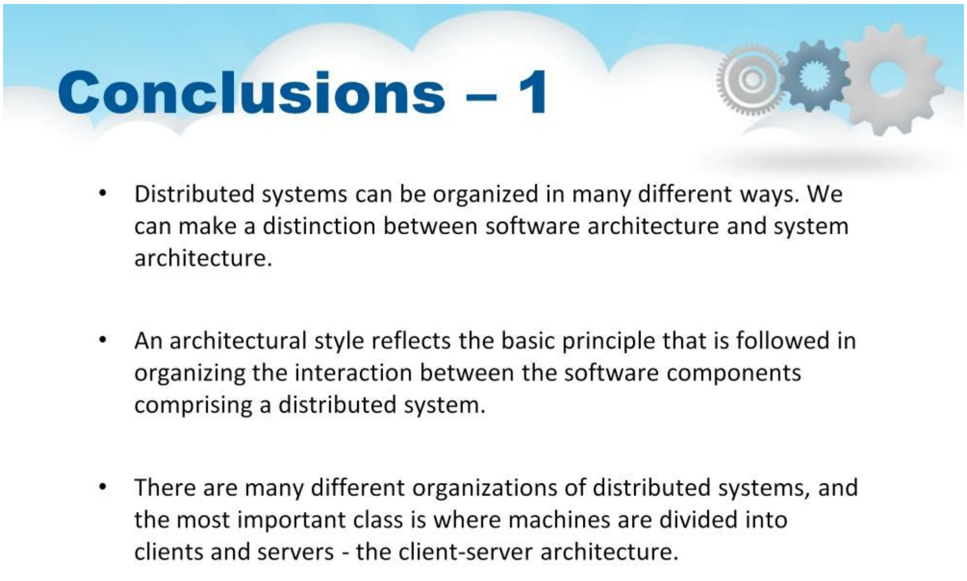
Steps required in a repair procedure:

- Terminate every binding between a component on a non-faulty node, and a component on the node that just failed.
- Request the node manager to start and add a new node to the domain.
- Configure the new node with exactly the same components as those on the crashed node.
- Re-establish all the bindings that were previously terminated.

When a failure has been detected, a repair procedure is started. Such a procedure is driven by a repair policy, partly executed by the node manager. Policies are stated explicitly and are carried out depending on the detected failure. For example, suppose a node failure has been detected. In that case, the repair policy may prescribe that the following steps are to be carried out:

1. Terminate every binding between a component on a nonfaulty node, and a component on the node that just failed.
2. Request the node manager to start and add a new node to the domain.
3. Configure the new node with exactly the same components as those on the crashed node.
4. Re-establish all the bindings that were previously terminated.

In this example, the repair policy is simple and will only work when no crucial data has been lost (the crashed components are said to be stateless).
The approach followed by Jade is an example of self-management: upon the detection of a failure, a repair policy is automatically executed to bring the system as a whole into a state in which it was before the crash. Being a component-based system, this automatic repair requires specific support to allow components to be added and removed at runtime. In general, turning legacy applications into selfmanaging systems is not possible.



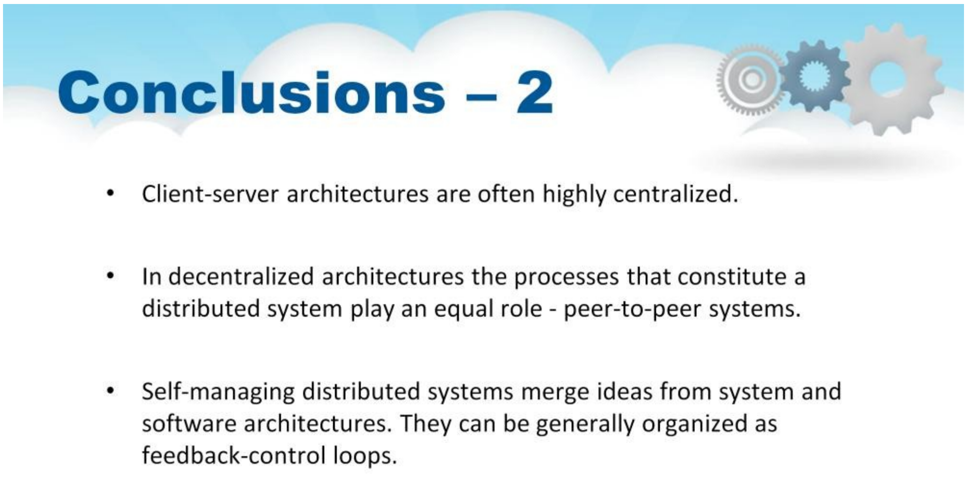
Conclusions – 1

- Distributed systems can be organized in many different ways. We can make a distinction between software architecture and system architecture.
- An architectural style reflects the basic principle that is followed in organizing the interaction between the software components comprising a distributed system.
- There are many different organizations of distributed systems, and the most important class is where machines are divided into clients and servers - the client-server architecture.

Distributed systems can be organized in many different ways. We can make a distinction between software architecture and system architecture. The latter considers where the components that constitute a distributed system are placed across the various machines. The former is more concerned about the logical organization of the software: how do components interact, it what ways can they be structured, how can they be made independent, and so on.

A key idea when talking about architectures is architectural style. A style reflects the basic principle that is followed in organizing the interaction between the software components comprising a distributed system. Important styles include layering, object orientation, event orientation, and data-space orientation.

There are many different organizations of distributed systems. An important class is where machines are divided into clients and servers. A client sends a request to a server, who will then produce a result that is returned to the client. The client-server architecture reflects the traditional way of modularizing software in which a module calls the functions available in another module. By placing different components on different machines, we obtain a natural physical distribution of functions across a collection of machines.



Conclusions – 2

- Client-server architectures are often highly centralized.
- In decentralized architectures the processes that constitute a distributed system play an equal role - peer-to-peer systems.
- Self-managing distributed systems merge ideas from system and software architectures. They can be generally organized as feedback-control loops.

Client-server architectures are often highly centralized.

In decentralized architectures we often see an equal role played by the processes that constitute a distributed system, also known as peer-to-peer systems. In peer-to-peer systems, the processes are organized into an overlay network, which is a logical network in which every process has a local list of other peers that it can communicate with. The overlay network can be structured, in which case deterministic schemes can be deployed for routing messages between processes. In unstructured networks, the list of peers is more or less random, implying that search algorithms need to be deployed for locating data or other processes.

As an alternative, self-managing distributed systems have been developed. These systems, to an extent, merge ideas from system and software architectures. Self-managing systems can be generally organized as feedback-control loops. Such loops contain a monitoring component by the behavior of the distributed system is measured, an analysis component to see whether anything needs to be adjusted, and a collection of various instruments for changing the behavior. Feedback -control loops can be integrated into distributed systems at numerous places.

Cloud Computing

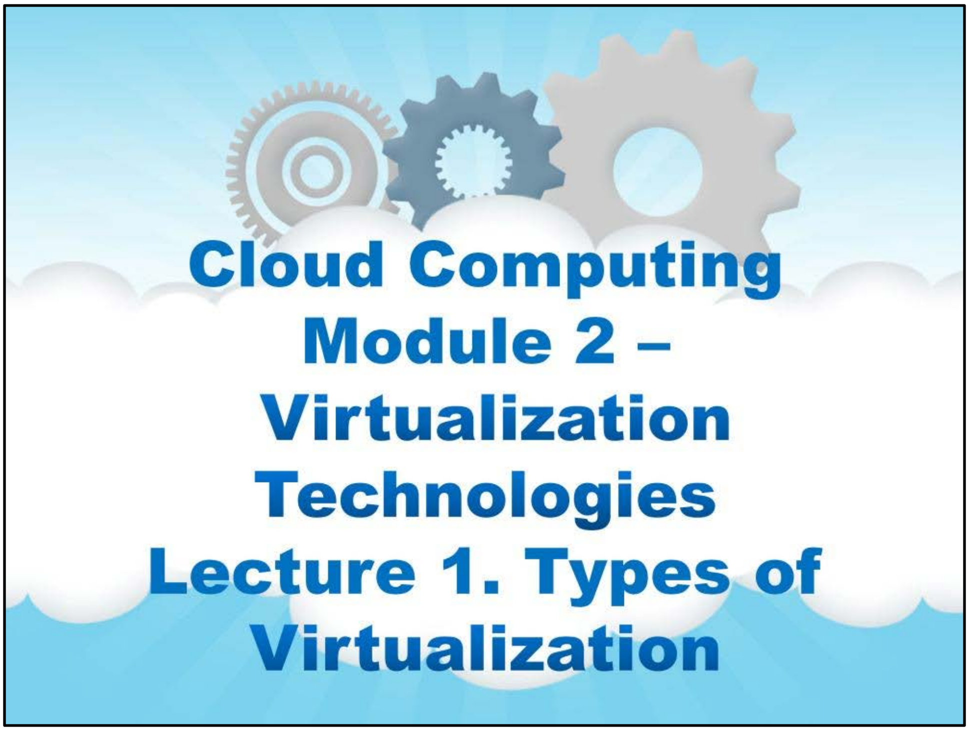
Lecture Manual

Volume 2

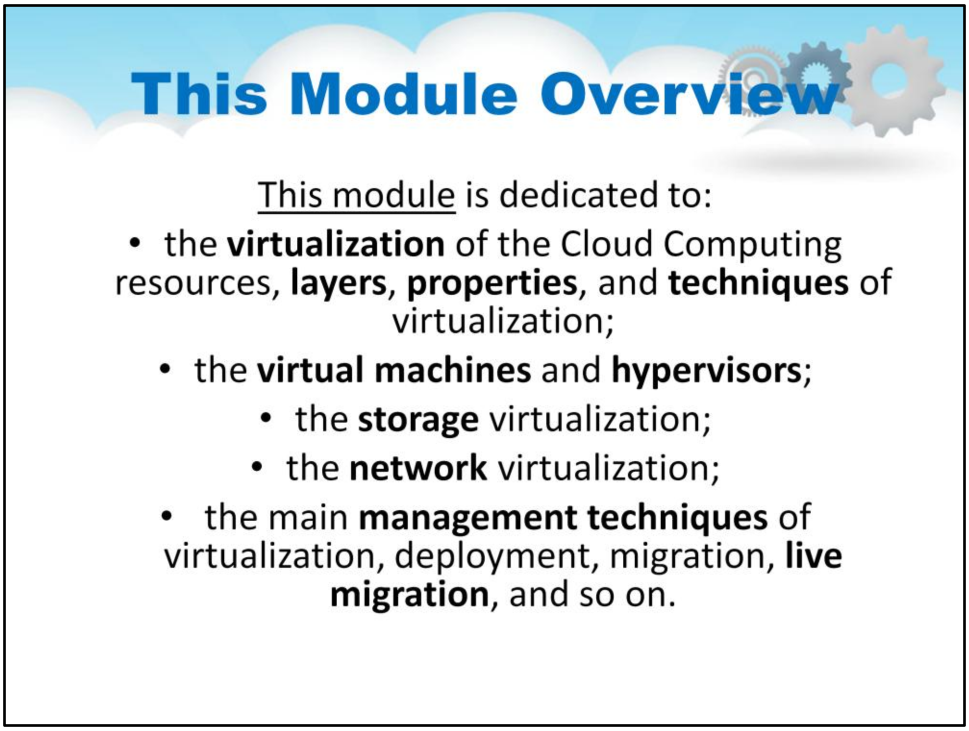
Module 2

Virtualization Technologies

Content



Cloud Computing Module 2 – Virtualization Technologies Lecture 1. Types of Virtualization



This Module Overview

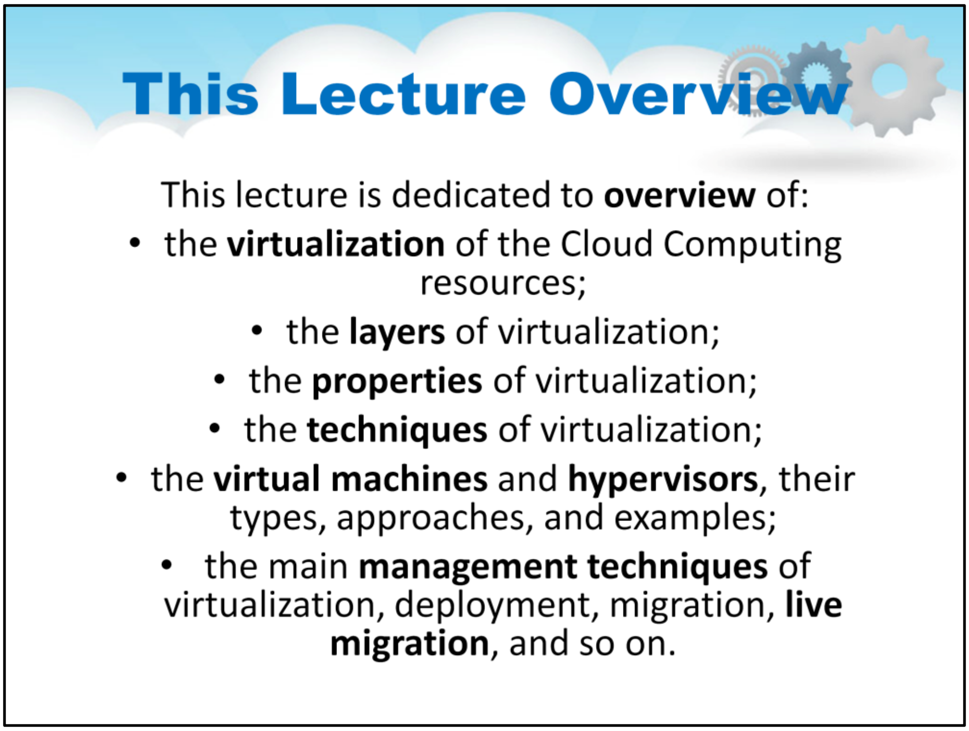
This module is dedicated to:

- the **virtualization** of the Cloud Computing resources, **layers**, **properties**, and **techniques** of virtualization;
- the **virtual machines** and **hypervisors**;
- the **storage** virtualization;
- the **network** virtualization;
- the main **management techniques** of virtualization, deployment, migration, **live migration**, and so on.

Module 2. Virtualization Technologies

This module is dedicated to:

- the **virtualization** of the Cloud Computing resources, **layers**, **properties**, and **techniques** of virtualization;
- the **virtual machines** and **hypervisors**;
- the **storage** virtualization;
- the **network** virtualization;
- the main **management techniques** of virtualization, deployment, migration, **live migration**, and so on.



This Lecture Overview

This lecture is dedicated to **overview** of:

- the **virtualization** of the Cloud Computing resources;
- the **layers** of virtualization;
- the **properties** of virtualization;
- the **techniques** of virtualization;
- the **virtual machines** and **hypervisors**, their types, approaches, and examples;
- the main **management techniques** of virtualization, deployment, migration, **live migration**, and so on.

Lecture 1. Types of Virtualization

This lecture is dedicated to **overview** of:

- the **virtualization** of the Cloud Computing resources;
- the **layers** of virtualization;
- the **properties** of virtualization;
- the **techniques** of virtualization;
- the **virtual machines** and **hypervisors**, their types, approaches, and examples;
- the main **management techniques** of virtualization, deployment, migration, **live migration**, and so on.

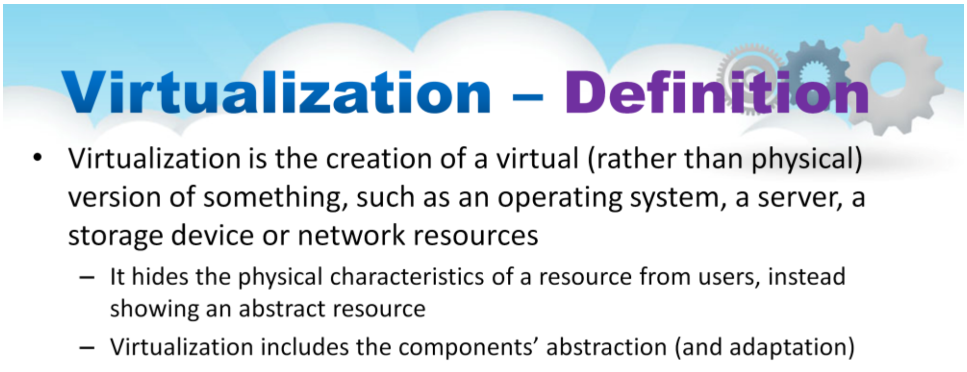
Virtualization Technologies

–

Overview

Virtualization Technologies

Overview



Virtualization – Definition

- Virtualization is the creation of a virtual (rather than physical) version of something, such as an operating system, a server, a storage device or network resources
  - It hides the physical characteristics of a resource from users, instead showing an abstract resource
  - Virtualization includes the components’ abstraction (and adaptation)

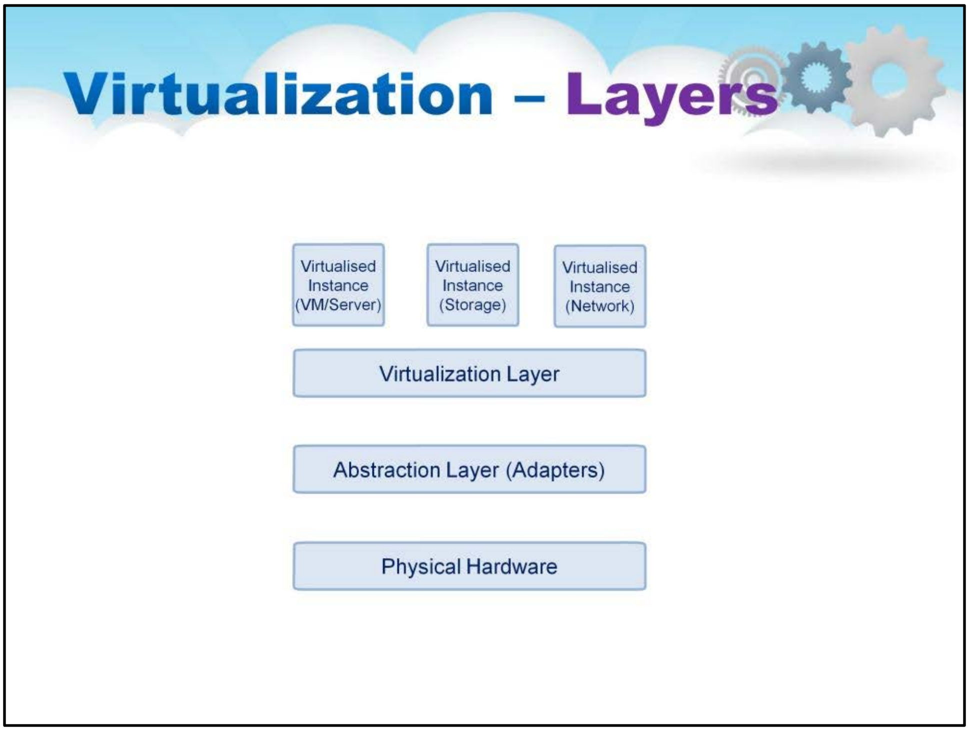
This slide speaks to the “abstract” or theoretical idea of virtualization.

Definition: Virtualization is the creation of a virtual (rather than physical) version of something, such as an operating system, a server, a storage device or network resources

- It hides the physical characteristics of a resource from users, instead showing an abstract resource
- Virtualization includes the components’ abstraction (and adaptation)

Either a physical thing can be virtualized, like a “disk drive”, or a more conceptual capability, like a “VLAN”. The slide speaks to how the virtualization layers relate to each other.

Note that sometimes specific hardware adapters are required, to model a particular type of processor, storage, or network.



Virtualization – Layers

Virtualised Instance (VM/Server)

Virtualised Instance (Storage)

Virtualised Instance (Network)

Virtualization Layer

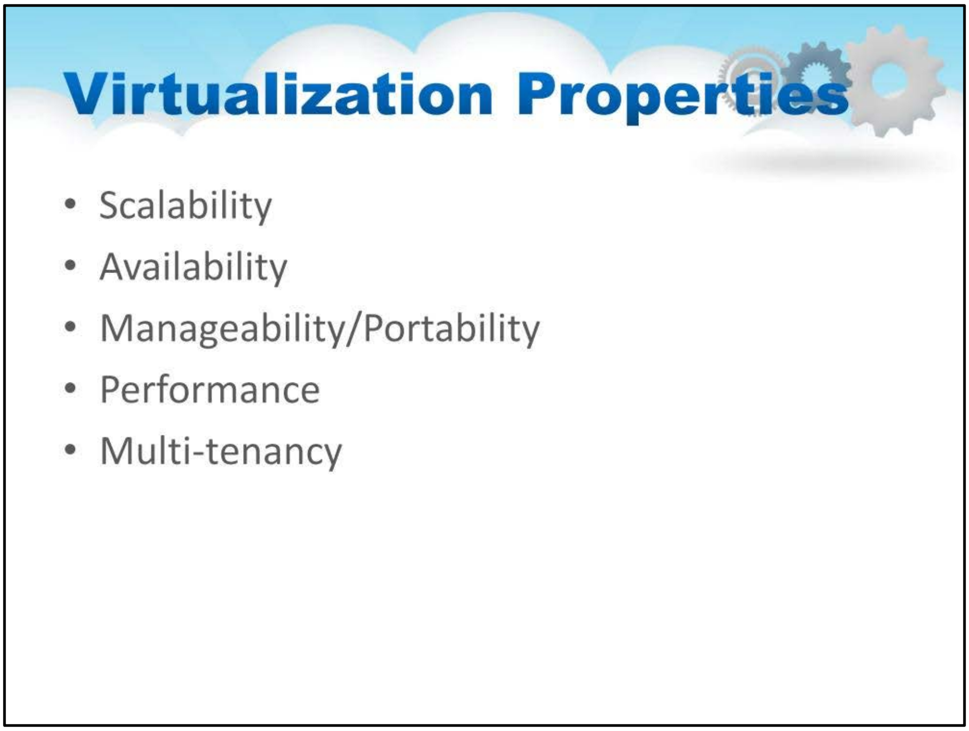
Abstraction Layer (Adapters)

Physical Hardware

Layers

General virtualization layers:

- Virtualized instances with configurable characteristics
- Virtualization layer, software virtualization implementation
- Abstraction layer, including hardware adaptors
- Physical resources: Various types of infrastructure resources



Virtualization Properties

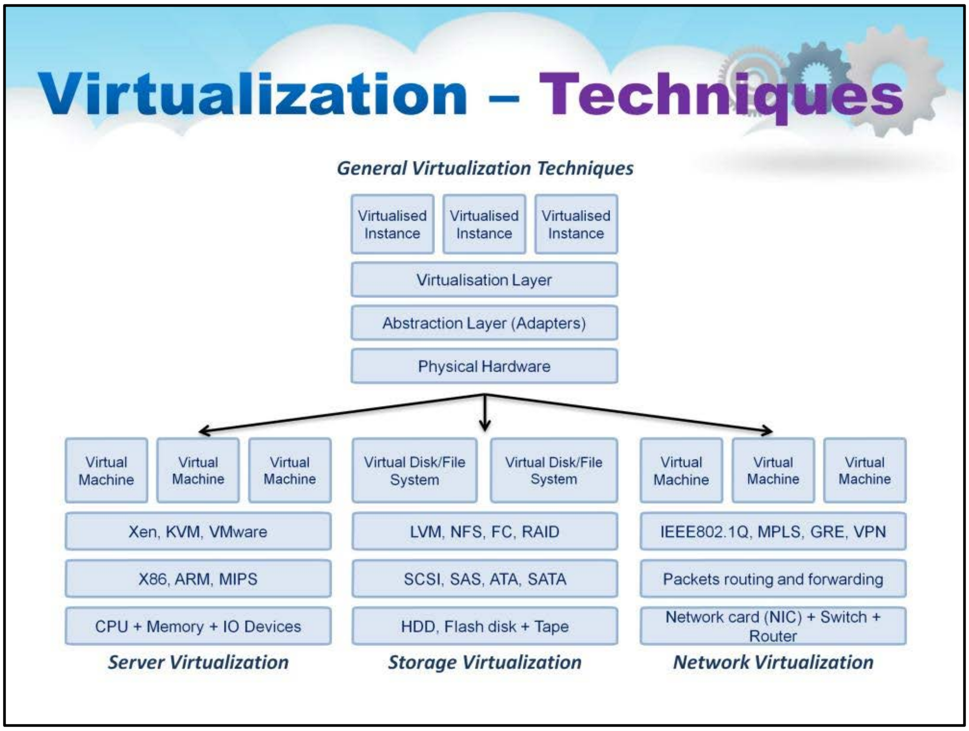
- Scalability
- Availability
- Manageability/Portability
- Performance
- Multi-tenancy

Virtualization is an essential technique for enabling cloud properties and important operations such as Live migration of VMs.

Live migration of virtual machines means that a virtual machine can be migrated from one physical machine to another in the run time with a small amount of performance down grade.

The following cloud properties are based on virtualization.

- Scalability - Virtual machine system scales automatically
- Availability - Fault tolerance to hardware and software failures
- Manageability/Portability - Automatic physical to virtual system transformation
- Performance - Dynamic virtual machine level load balancing
- Multi-tenancy - Tenants’ infrastructure and applications isolation



Virtualization – Techniques

General Virtualization Techniques

Virtualised Instance | Virtualised Instance | Virtualised Instance

Virtualisation Layer

Abstraction Layer (Adapters)

Physical Hardware

Virtual Machine | Virtual Machine | Virtual Machine

Xen, KVM, VMware

X86, ARM, MIPS

CPU + Memory + IO Devices

Server Virtualization

Virtual Disk/File System | Virtual Disk/File System

LVM, NFS, FC, RAID

SCSI, SAS, ATA, SATA

HDD, Flash disk + Tape

Storage Virtualization

Virtual Machine | Virtual Machine | Virtual Machine

IEEE802.1Q, MPLS, GRE, VPN

Packets routing and forwarding

Network card (NIC) + Switch + Router

Network Virtualization

Techniques

This slide illustrates Virtualization techniques.

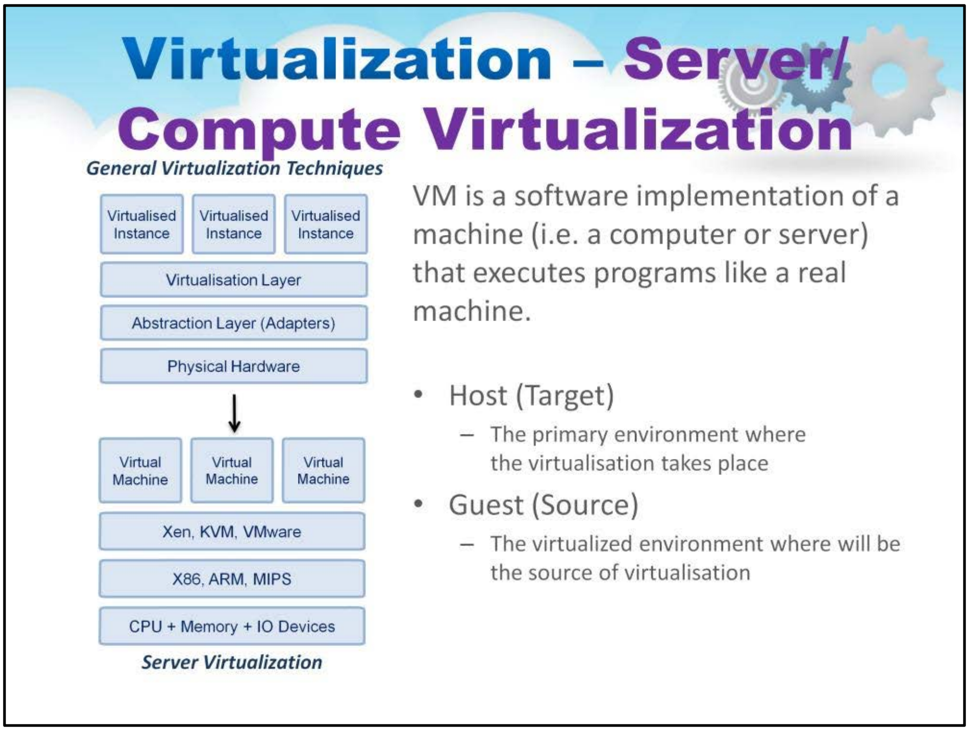
At the top the General Virtualization Technique is illustrated.

Underneath, one can see how the general technique is applied to several virtualization problems.

For Server virtualization, hypervisors are extensively used.
The hypervisors will be explained later.

For storage virtualization, many software techniques including volume management, file systems, and replication are applied.

For network virtualization, there are many different features including link aggregation, VPN, and also firewall, switching, routing, and application filtering and load balancing have virtual capability in many cloud implementations today.



Virtualization – Server/ Compute Virtualization

General Virtualization Techniques

Virtualised Instance | Virtualised Instance | Virtualised Instance

Virtualisation Layer

Abstraction Layer (Adapters)

Physical Hardware

↓

Virtual Machine | Virtual Machine | Virtual Machine

Xen, KVM, VMware

X86, ARM, MIPS

CPU + Memory + IO Devices

Server Virtualization

VM is a software implementation of a machine (i.e. a computer or server) that executes programs like a real machine.

- Host (Target)
  - – The primary environment where the virtualisation takes place
- Guest (Source)
  - – The virtualized environment where will be the source of virtualisation

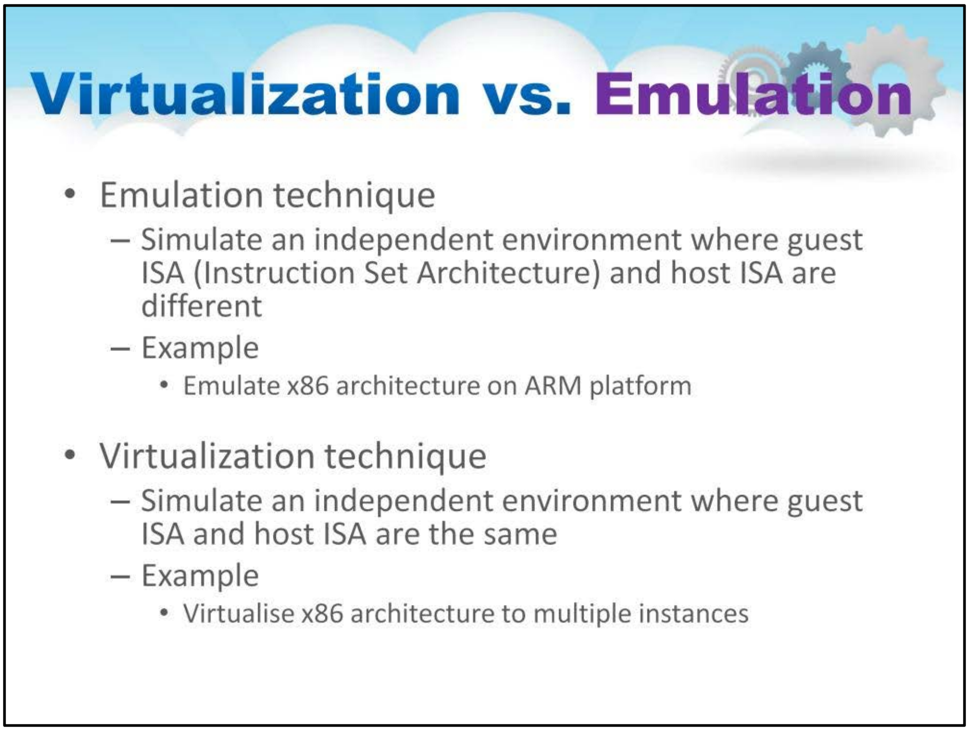
In this tutorial we look closer at the server or compute resource virtualisation. First , we will look at this situation in the generality.

The most important concept is the that the host is a particular physical infrastructure, right down to a processor type and specific I/O devices.

The guest is an operating system which is running on an “abstracted” hardware platform; this abstracted platform may not even be the same processor or I/O architecture as the host.

It is the job of the virtualization and abstraction layer to provide this intermediary function.

This is illustrated in the slide.



Virtualization vs. Emulation

- Emulation technique
  - Simulate an independent environment where guest ISA (Instruction Set Architecture) and host ISA are different
  - Example
    - Emulate x86 architecture on ARM platform
- Virtualization technique
  - Simulate an independent environment where guest ISA and host ISA are the same
  - Example
    - Virtualise x86 architecture to multiple instances

Emulation vs. Virtualisation

Example

Emulate x86 architecture on ARM platform

Example

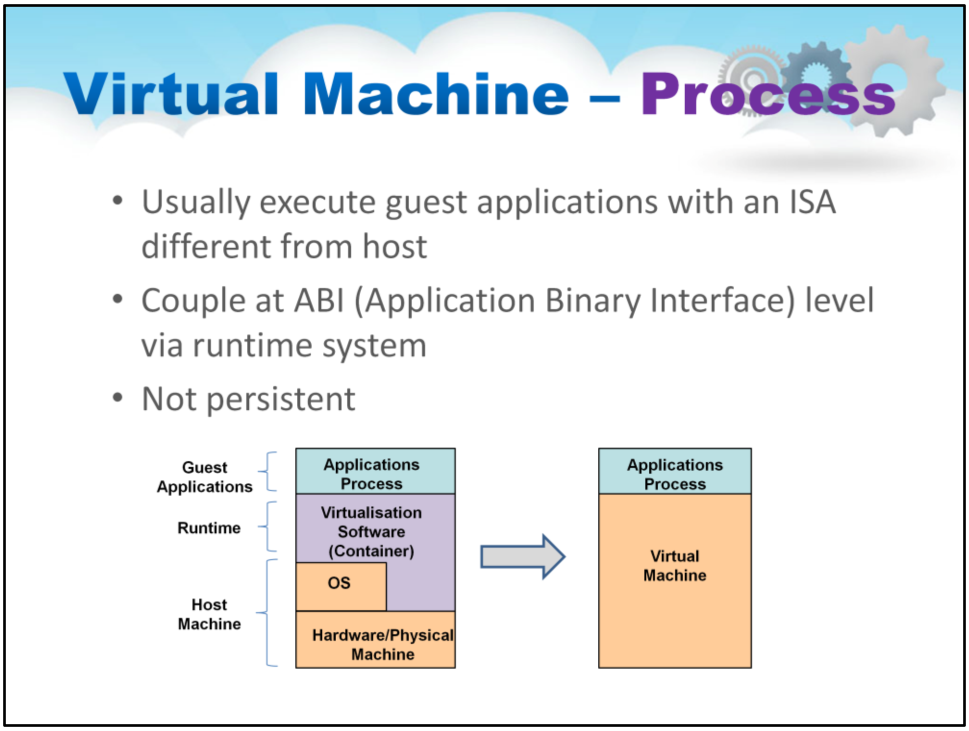
Virtualise x86 architecture to multiple instances

Virtualization Technologies

–

Virtual Machines

Virtual Machines

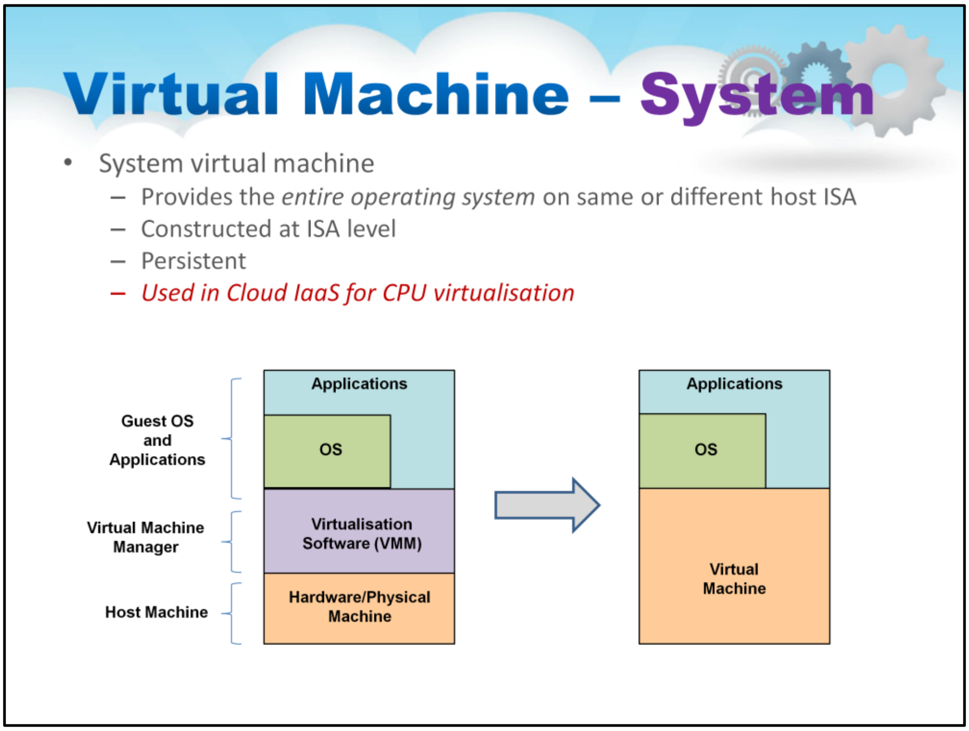


Virtual Machine – Process

- Usually execute guest applications with an ISA different from host
- Couple at ABI (Application Binary Interface) level via runtime system
- Not persistent

One approach for virtualization is to use a facility at the process-level in the Operating System. This is called “process virtual machine” which is not the same as a hypervisor technique (next slide), This is an OS technique and has some resurgence in interest lately.

A Process virtual machine, sometimes called an application virtual machine, runs as a normal application inside a host OS and supports a single process. It is created when that process is started and destroyed when it exits. Its purpose is to provide a platform-independent programming environment that abstracts away details of the underlying hardware or operating system, and allows a program to execute in the same way on any platform.



Virtual Machine – System

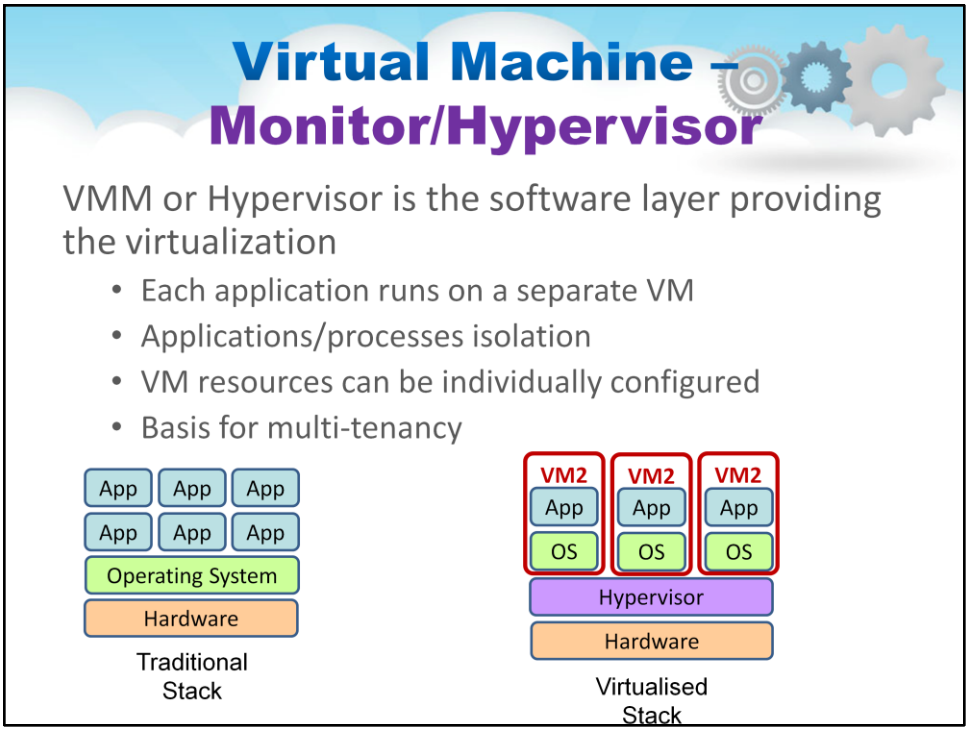
- System virtual machine
  - Provides the *entire operating system* on same or different host ISA
  - Constructed at ISA level
  - Persistent
  - *Used in Cloud IaaS for CPU virtualisation*

Guest OS and Applications
Virtual Machine Manager
Host Machine
Applications
OS
Virtualisation Software (VMM)
Hardware/Physical Machine
Applications
OS
Virtual Machine

The most common virtualization approach uses a Hypervisor and is called “System Virtual Machine”.

Provides the entire operating system on same or different host ISA
Constructed at ISA level
Persistent
Used in Cloud IaaS for CPU virtualisation

This is why we will concentrate on this approach form here out



Virtual Machine – Monitor/Hypervisor

VMM or Hypervisor is the software layer providing the virtualization

- Each application runs on a separate VM
- Applications/processes isolation
- VM resources can be individually configured
- Basis for multi-tenancy

App App App
App App App
Operating System
Hardware
Traditional Stack

VM2 VM2 VM2
App App App
OS OS OS
Hypervisor
Hardware
Virtualised Stack

To implement the System Layer Virtual Machine as described previously, a Virtual Machine Monitor is needed. This is more commonly called a Hypervisor.

The slide illustrates how a Hypervisor fits into the Operating System stack.

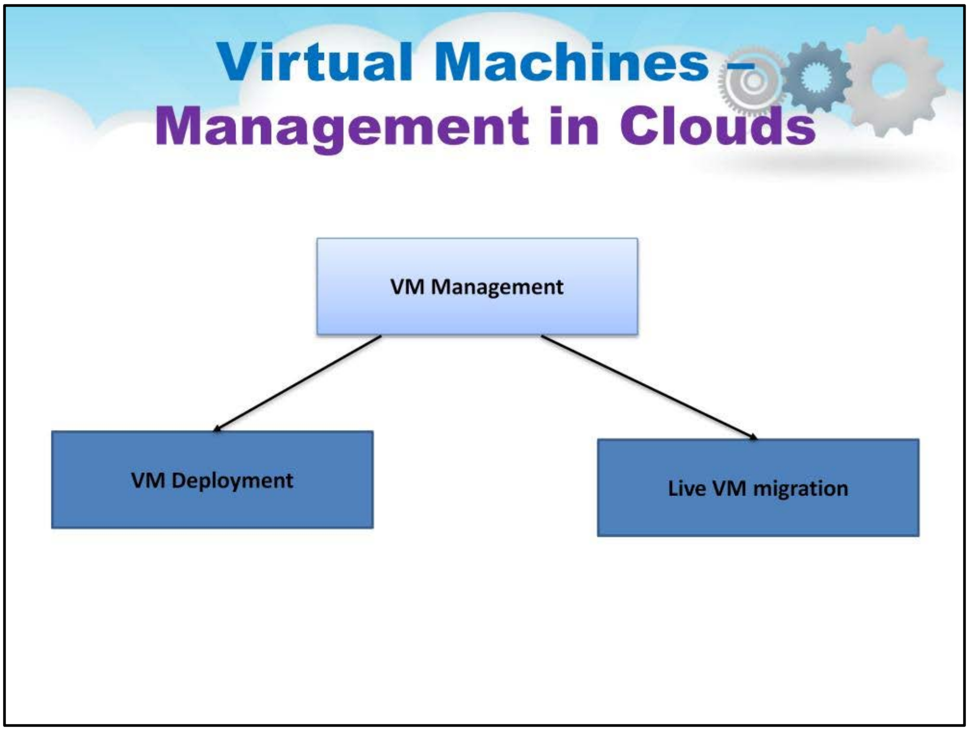
A hypervisor is a process that separates a computer's operating system and applications from the underlying physical hardware. Usually done as software although embedded hypervisors can be created for things like mobile devices.

The Hypervisor is the software layer providing the virtualization

- Each application runs on a separate VM
- Applications/processes isolation
- VM resources can be individually configured
- It is the Basis for multi-tenancy

Virtualization Technologies – Virtual Machines – Management in Clouds

Management in Clouds



Virtual Machines – Management in Clouds

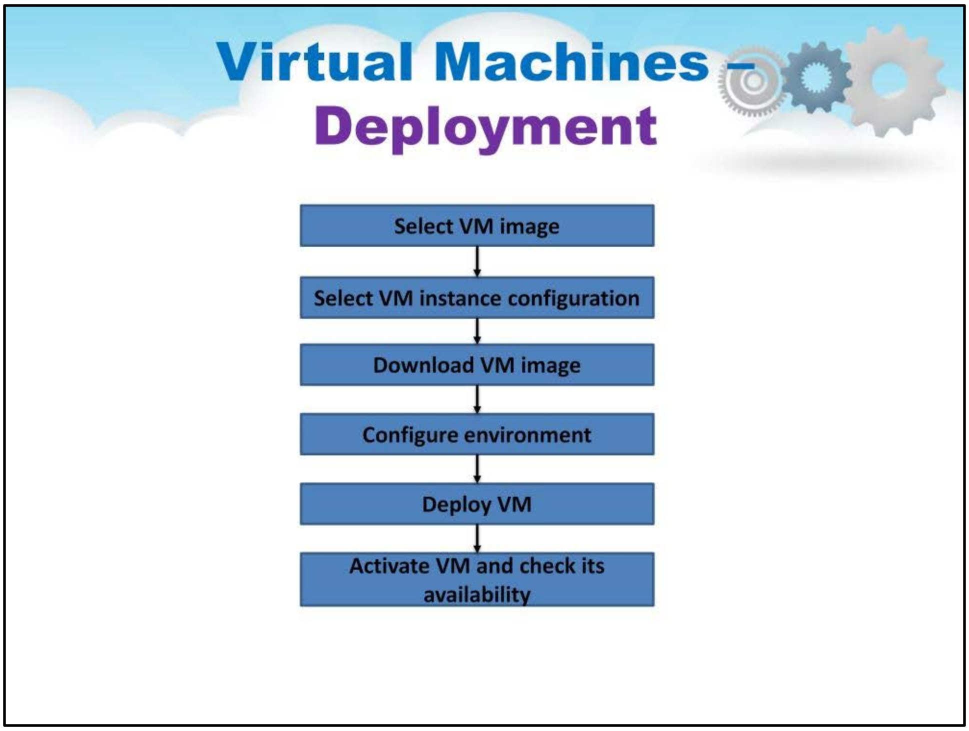
VM Management

VM Deployment

Live VM migration

Having a VMM/Hypervisor capability installed onto all of the machines inside a cluster is a terrific start. However one does not really have a cloud yet unitl automation is added, where the deployment of the VMs are controlled by the VMM in response to programmatic inputs. The VMM can also re-arrange, or start and stop VM’s to better utilize resources. Live VM migration is a technique which the VMM leverages to do its’ rearranging.

The overall automation is enabled by the software control of VM’s which a VMM/Hypervisor provides. Lets look into this important capability in depth.



Virtual Machines – Deployment

Select VM image

Select VM instance configuration

Download VM image

Configure environment

Deploy VM

Activate VM and check its availability

Select VM image - Use previously stored VM image or find on the VM marketplace typically available at cloud provider

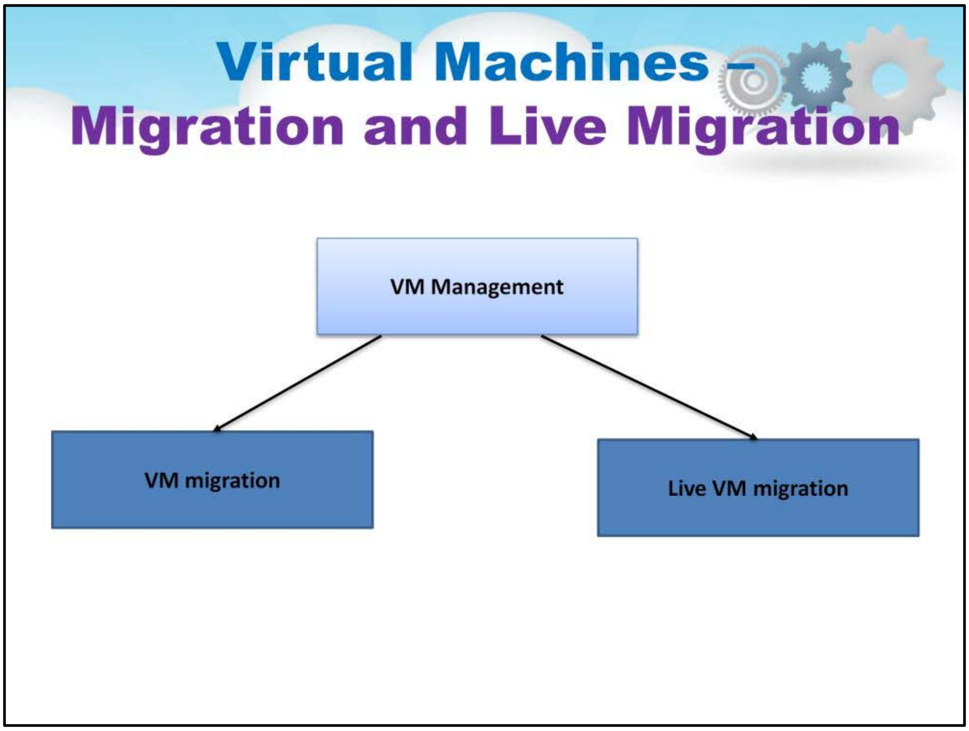
Select VM instance configuration - CPU type, # cores, memory, storage

Download VM image - to the selected server/datacenter location (e.g. select between AWS availability zone)

Configure environment - IP address (public or private); gateway, DNS, external storage, Load Balancer (optionally)

Deploy VM - using web tool or command line interface

Activate VM and check its availability - by accessing VM and/or in the hypervisor directory



Virtual Machines – Migration and Live Migration

VM Management

VM migration

Live VM migration

In the context of virtualization, where a guest simulation of an entire computer is actually merely a software virtual machine (VM) running on a host computer under a hypervisor, **migration** is the process by which a running virtual machine is moved from one physical host to another, with little or no disruption in service.

Live migration refers to the process of moving a running virtual machine or application between different physical machines without disconnecting the client or application. Memory, storage, and network connectivity of the virtual machine are transferred from the original guest machine to the destination.

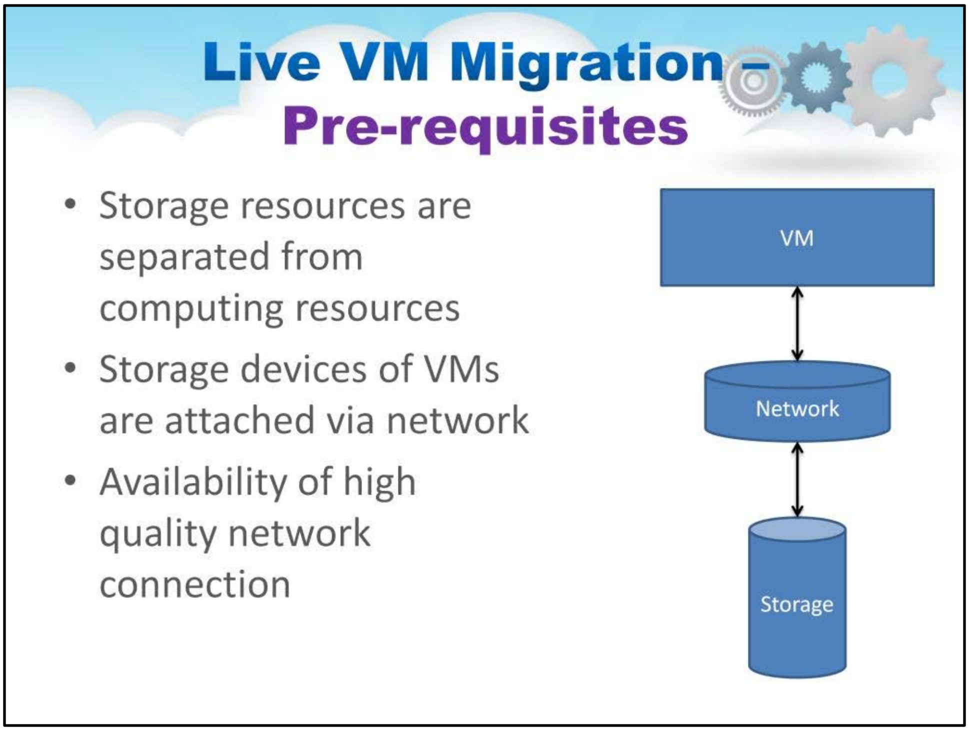
Virtualization Technologies

–

Virtual Machines –

Live VM Migration

Live VM Migration

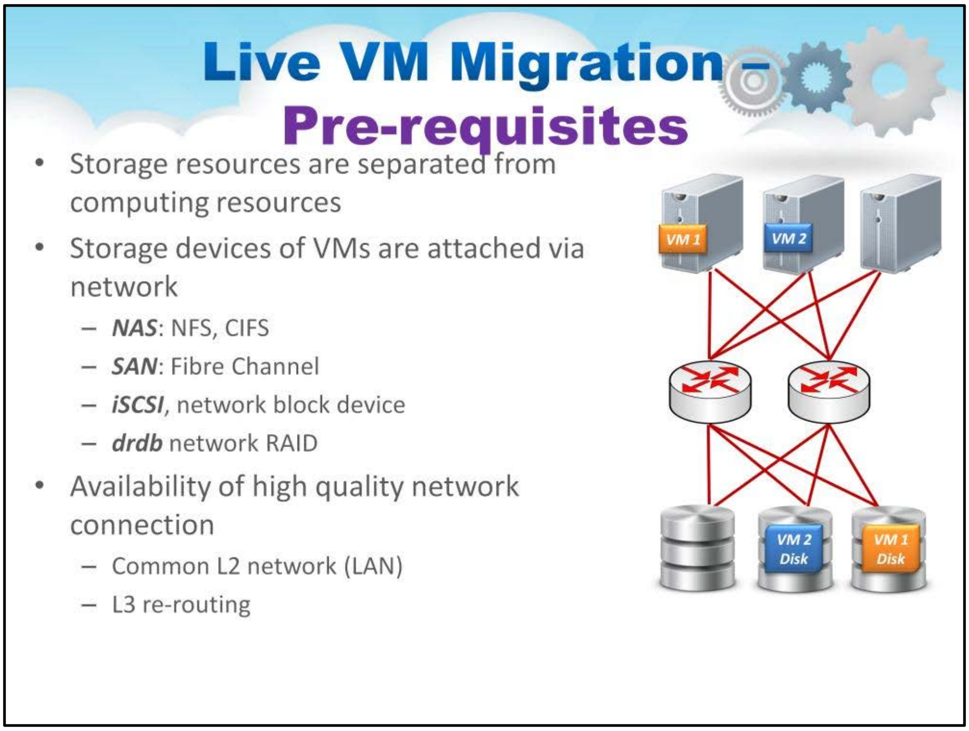


Live VM Migration – Pre-requisites

- Storage resources are separated from computing resources
- Storage devices of VMs are attached via network
- Availability of high quality network connection

VM
Network
Storage

Once the VM is running, it is the subject of further adjustment by the Cloud OS / VMM. There may be a utilization related optimization which calls for this running VM to be located on a different physical server.



Live VM Migration – Pre-requisites

- Storage resources are separated from computing resources
- Storage devices of VMs are attached via network
  - ***NAS***: NFS, CIFS
  - ***SAN***: Fibre Channel
  - ***iSCSI***, network block device
  - ***drdb*** network RAID
- Availability of high quality network connection
  - Common L2 network (LAN)
  - L3 re-routing

In order to move a VM with Live Migration, storage resources need to be separated from computing resources, so they can be re-mapped to the VM’s target location.

Storage devices of VMs are attached via network

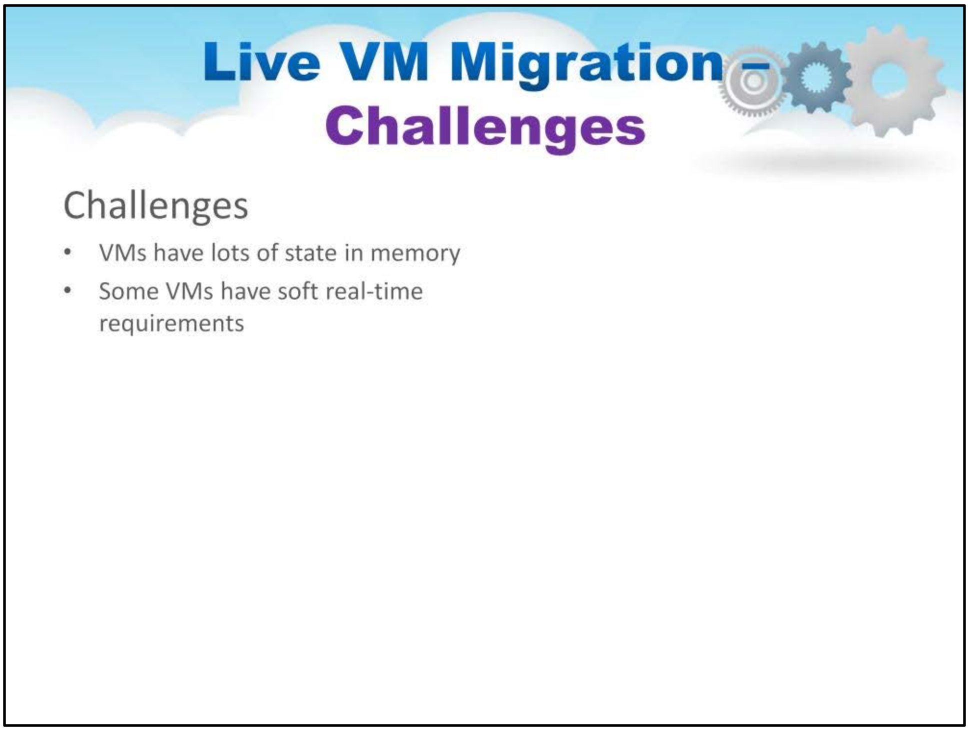
NAS: NFS, CIFS

SAN: Fibre Channel

iSCSI, network block device

drdb network RAID

And since we are doing memory copy (which is how Live Migration works) the network needs to support these transfer rates through switches and routers if needed.

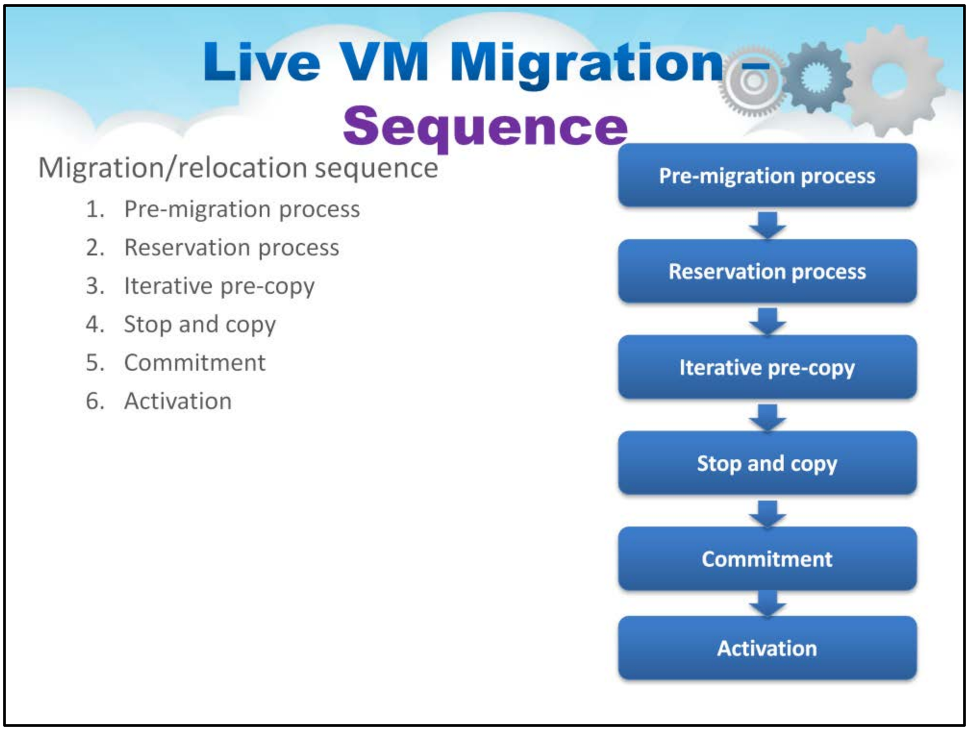


Live VM Migration – Challenges

Challenges

- VMs have lots of state in memory
- Some VMs have soft real-time requirements

Some applications have lots and lots of state, with large memory footprints and lots of transactional workloads, Live Migration can be a challenging race between the memory copy subsystem time slices and the application runtime time slices. For most applications, the algorithms work well and Live Migration is a smash hit



Live VM Migration – Sequence

Migration/relocation sequence

1. Pre-migration process
2. Reservation process
3. Iterative pre-copy
4. Stop and copy
5. Commitment
6. Activation

Pre-migration process

Reservation process

Iterative pre-copy

Stop and copy

Commitment

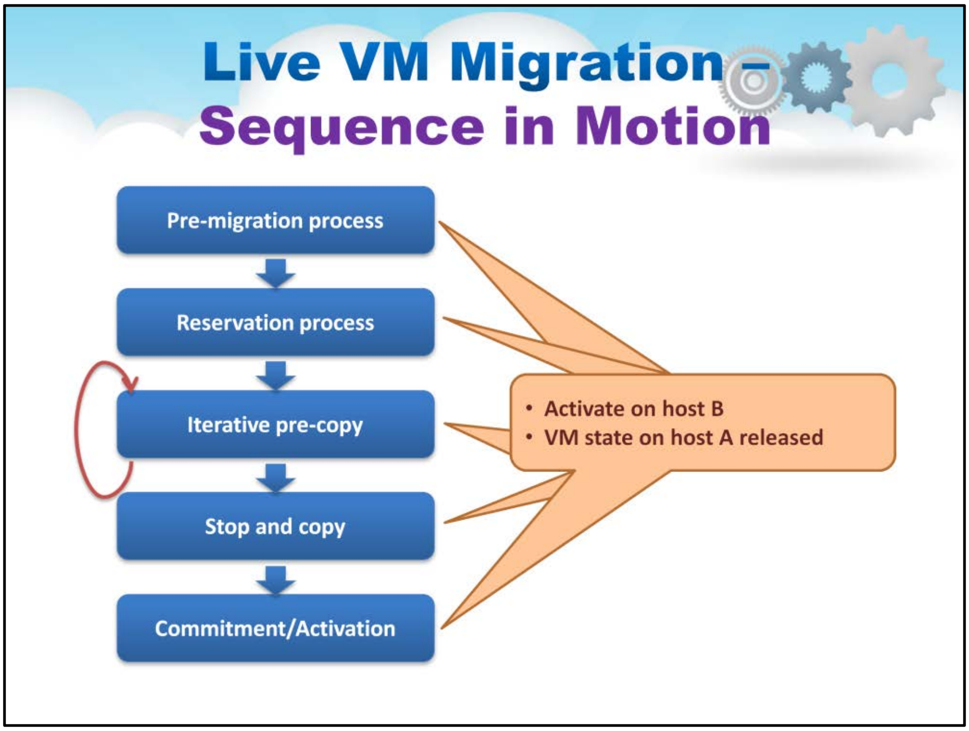
Activation

How does Live Migration actually work?

This is illustrated in the slide.

It is a specialized Migration/relocation sequence

1. Pre-migration process
2. Reservation process
3. Iterative pre-copy
4. Stop and copy
5. Commitment
6. Activation



Live VM Migration – Sequence in Motion

Pre-migration process

Reservation process

Iterative pre-copy

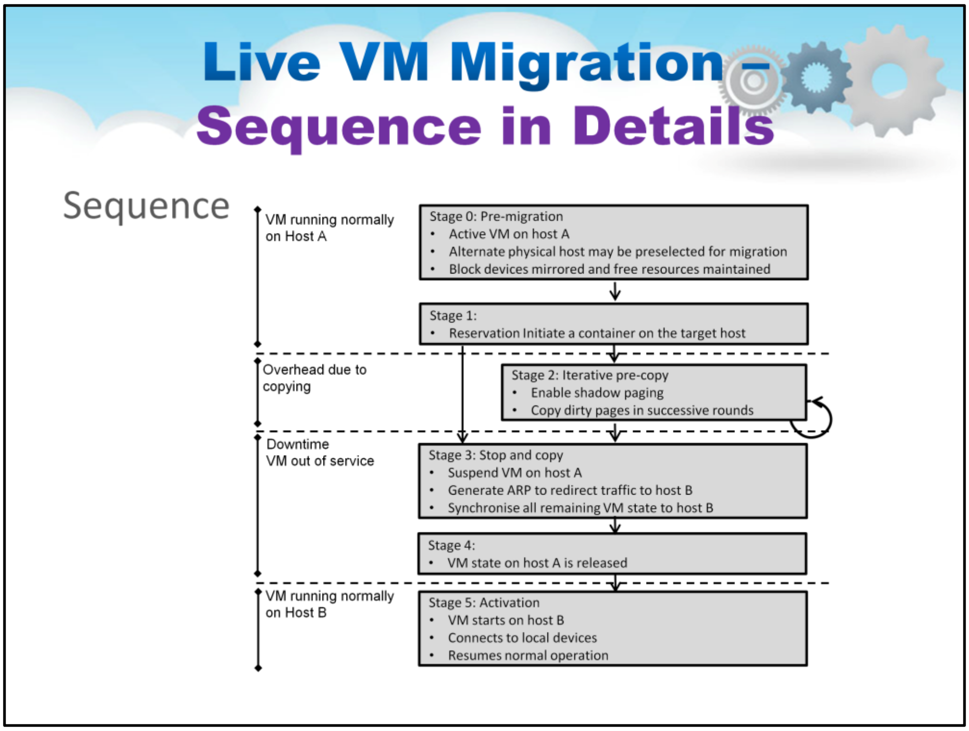
Stop and copy

Commitment/Activation

- Activate on host B
- VM state on host A released

This slide illustrate the whole process of migration/relocation sequence in animated way:

1. Pre-migration process
2. Reservation process
3. Iterative pre-copy
4. Stop and copy
5. Commitment/Activation



Live VM Migration – Sequence in Details

Sequence

VM running normally on Host A

Stage 0: Pre-migration
- Active VM on host A
- Alternate physical host may be preselected for migration
- Block devices mirrored and free resources maintained

Stage 1:
- Reservation Initiate a container on the target host

Overhead due to copying

Stage 2: Iterative pre-copy
- Enable shadow paging
- Copy dirty pages in successive rounds

Downtime VM out of service

Stage 3: Stop and copy
- Suspend VM on host A
- Generate ARP to redirect traffic to host B
- Synchronise all remaining VM state to host B

Stage 4:
- VM state on host A is released

VM running normally on Host B

Stage 5: Activation
- VM starts on host B
- Connects to local devices
- Resumes normal operation

This slide goes into even further detail on Live Migration.

Sequence is started at **Stage 1: Pre-migration**

- Active VM on host A
- Alternate physical host may be preselected for migration
- Block devices mirrored and free resources maintained

Then reservation process goes at **Stage 2: Reservation**

- Reservation Initiate a container on the target host

Note: During these two stages VM is running normally on Host A.

Then **Stage 3: Iterative pre-copy** begins:

- Enable shadow paging
- Copy dirty pages in successive rounds

Note: Overhead appears due to copying.

After this **Stage 4: Stop and copy** starts:

- Suspend VM on host A
- Generate ARP to redirect traffic to host B
- Synchronize all remaining VM state to host B

Then **Stage 5: Commitment** begins:

- VM state on host A is released

Note: Downtime VM out of service can be observed during Stages 4 and 5.

Finally, **Stage 6: Activation** starts:

- VM starts on host B
- Connects to local devices
- Resumes normal operation

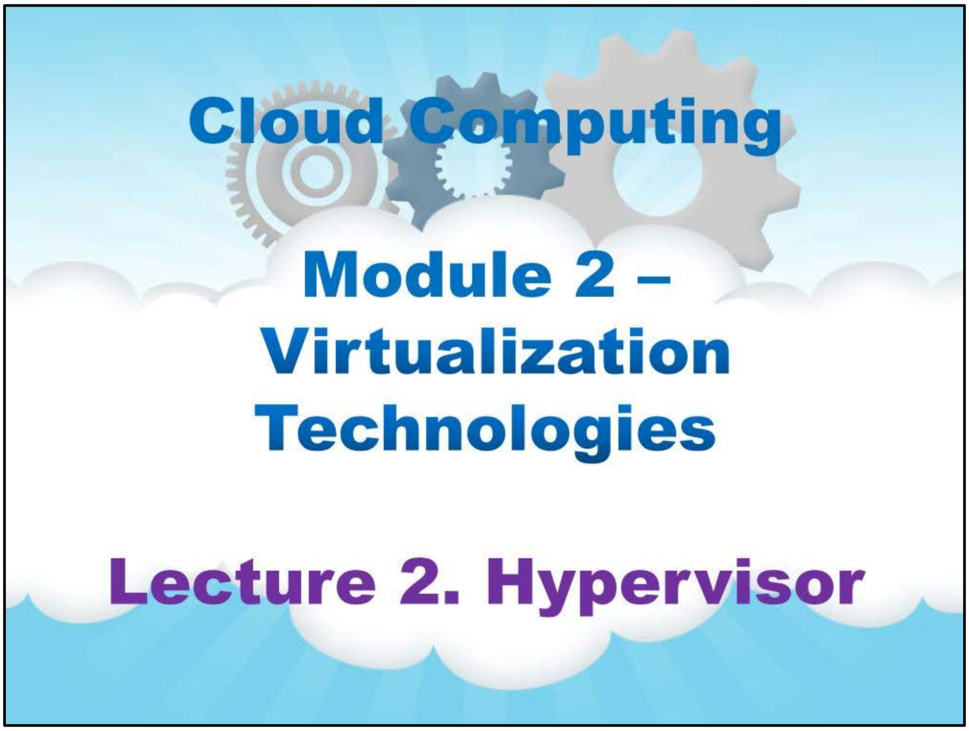
Note: now VM running normally on Host B

Summary and takeaway

- Cloud IaaS represents all generic Cloud Computing properties and is the most widely used cloud service type
- Cloud IaaS Architecture includes functionalities to virtualise physical resources
- There is a variety of Cloud IaaS platforms
- Virtualization is the major enabling technology for IaaS cloud
- VM migration is a function of the Cloud IaaS management software
- Historically first, current Amazon Web Services Cloud represents all generic IaaS cloud properties

Summary and take away

- Cloud IaaS represents all generic Cloud Computing properties and is the most widely used cloud service type
- Cloud IaaS Architecture includes functionalities to virtualize physical resources (Compute, Storage, Network), support provisioned on-demand cloud IaaS services deployment and management
- There is a variety of Cloud IaaS platforms
  Big Cloud IaaS Providers use their own proprietary platforms
- There is a variety of Open Source Cloud Management platforms
  OpenStack, OpenNebula, Eucalyptus, Nimbus are the most popular
- Virtualization is the major enabling technology for IaaS cloud
  Enables cloud scalability, availability, manageability and performance
- VM migration is a function of the Cloud IaaS management software
  Live VM migration is done in a few steps and can minimize services downtime
  In this lecture we discussed the Amazon Web Services Cloud and its main functionality
  Historically first, current AWS Cloud represents all the generic IaaS cloud properties.
  Understanding the basic AWS Cloud properties will provide a good basis for understanding other cloud platforms

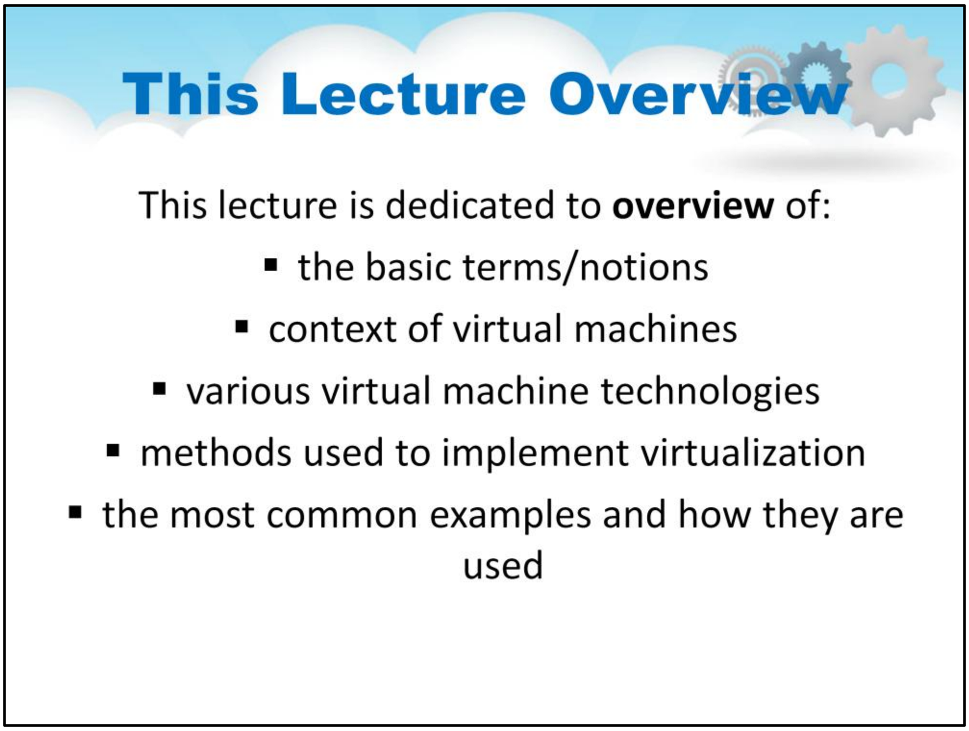


Cloud Computing

Module 2 – Virtualization Technologies

Lecture 2. Hypervisor

This is module 2 under title “Virtualization Technologies”.
This lecture 2 is about hypervisors.



This Lecture Overview

This lecture is dedicated to **overview** of:

- the basic terms/notions
- context of virtual machines
- various virtual machine technologies
- methods used to implement virtualization
- the most common examples and how they are used

Lecture 2. Hypervisor

This lecture is dedicated to **overview** of:

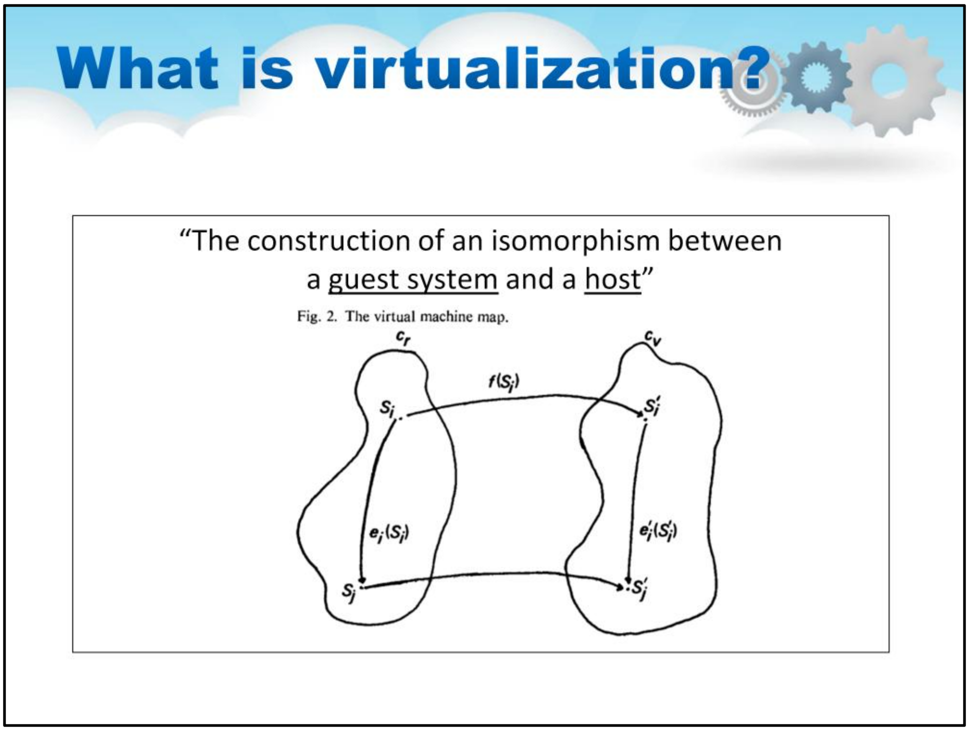
- the basic terms notions in the virtualization technologies (what is virtualization, definitions, components, scheme)
- the general context of virtual machines (what is virtualization, definitions, components, scheme)
- various virtual machine technologies (principal idea, motivation and usage, evolution)
- methods used to implement virtualization (definitions, pro and contra, classification)
- the most common examples and how they are used

Virtualization Technologies – Basic Notions

Basic Notions

This section is about the basic terms notions in the virtualization technologies:

- What is virtualization
- Definitions
- Components
- Scheme



What is virtualization?

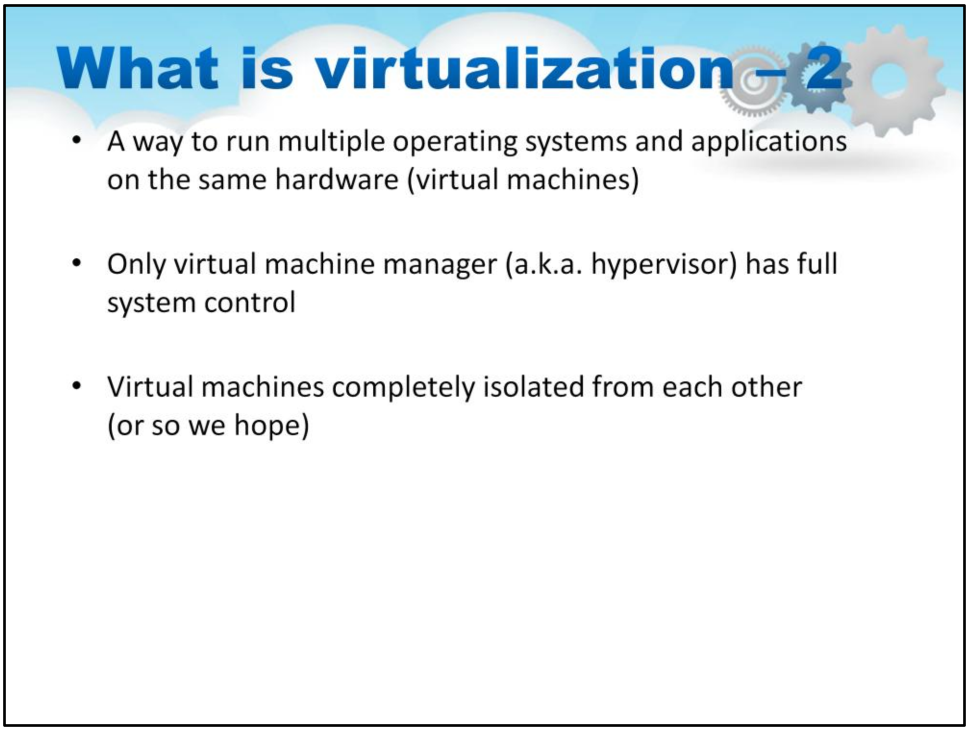
"The construction of an isomorphism between a guest system and a host"

Fig. 2. The virtual machine map.

Virtualization is creating a virtual version of something (hardware, operating system, application, network, memory, storage).

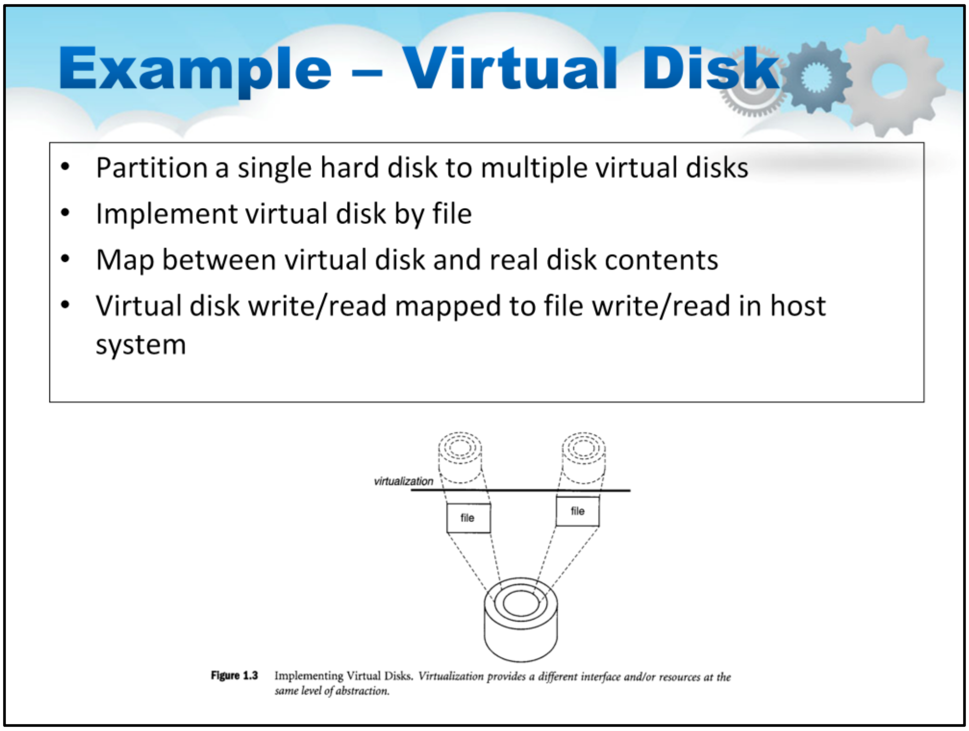
According to the work of Popek and Goldberg (in 1974), virtualization is a "The construction of an isomorphism between a guest system and a host"

See details in Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." Communications of the ACM 17.7 (1974): 412-421.



What is virtualization – 2

- A way to run multiple operating systems and applications on the same hardware (virtual machines)
- Only virtual machine manager (a.k.a. hypervisor) has full system control
- Virtual machines completely isolated from each other (or so we hope)



Example – Virtual Disk

- Partition a single hard disk to multiple virtual disks
- Implement virtual disk by file
- Map between virtual disk and real disk contents
- Virtual disk write/read mapped to file write/read in host system

virtualization
file
file

Figure 1.3 Implementing Virtual Disks. *Virtualization provides a different interface and/or resources at the same level of abstraction.*

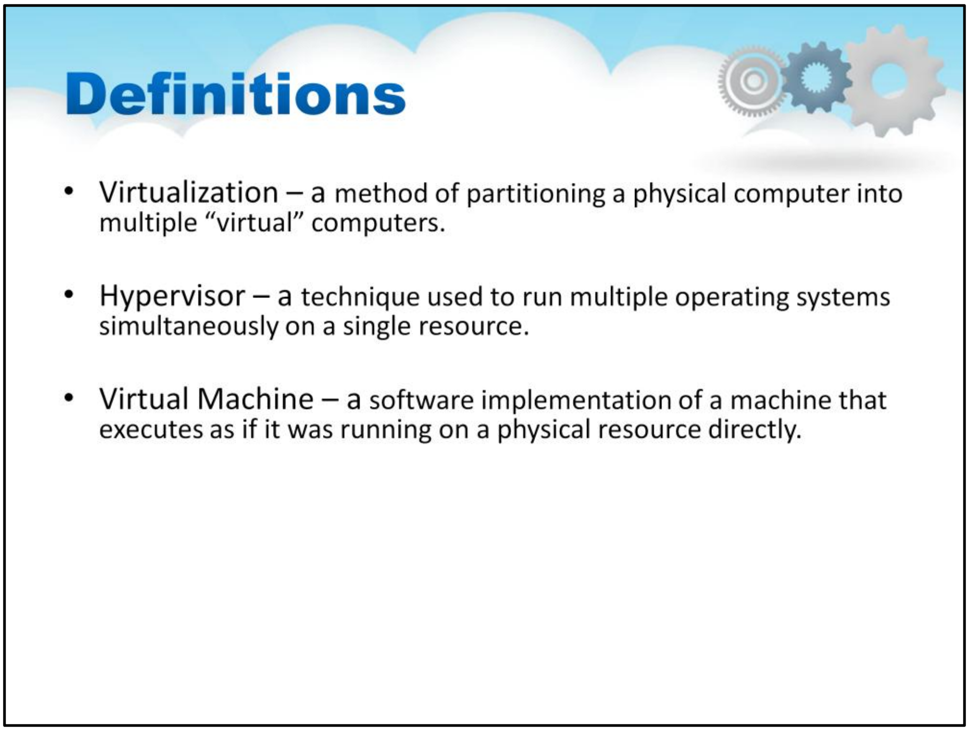
Partition a single hard disk to multiple virtual disks

Virtual disk has virtual tracks & sectors

Implement virtual disk by file

Map between virtual disk and real disk contents

Virtual disk write/read mapped to file write/read in host system



Definitions

- Virtualization – a method of partitioning a physical computer into multiple “virtual” computers.
- Hypervisor – a technique used to run multiple operating systems simultaneously on a single resource.
- Virtual Machine – a software implementation of a machine that executes as if it was running on a physical resource directly.

What is Virtualization?

It is a method of partitioning a physical computer into multiple “virtual” computers, each acting independently as if they were running directly on hardware.

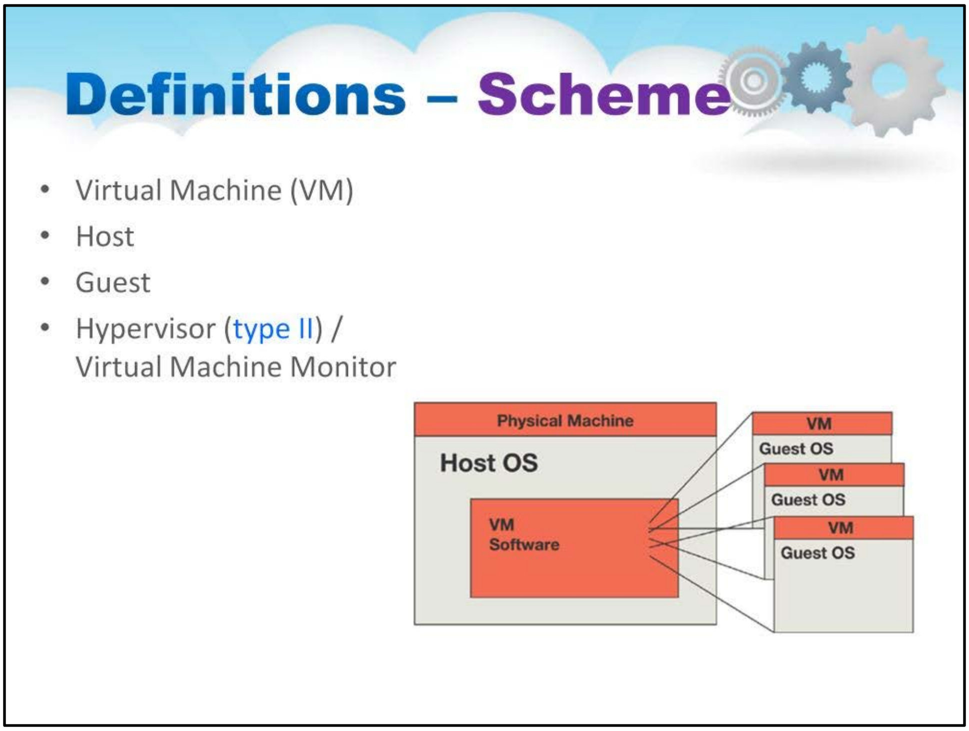
What is a Hypervisor?

It is a technique used to run multiple operating systems simultaneously on a single resource.

Also called a Virtual Machine Monitor (VMM).

What is a Virtual Machine?

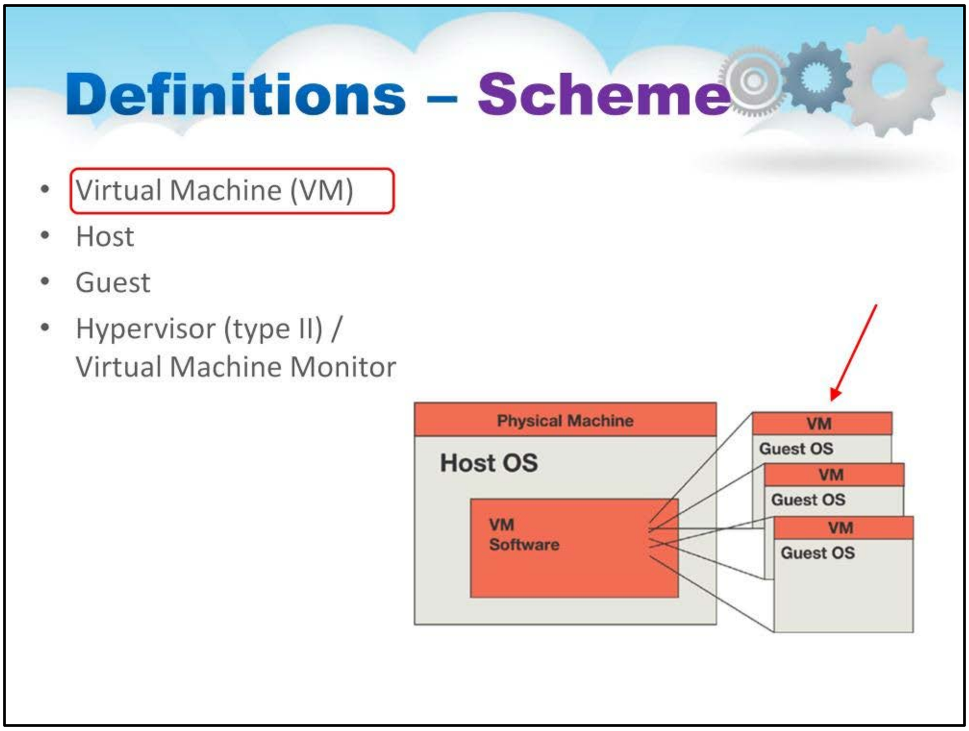
It is a software implementation of a machine that executes as if it was running on a physical resource directly.



Definitions – Scheme

- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) /
  Virtual Machine Monitor

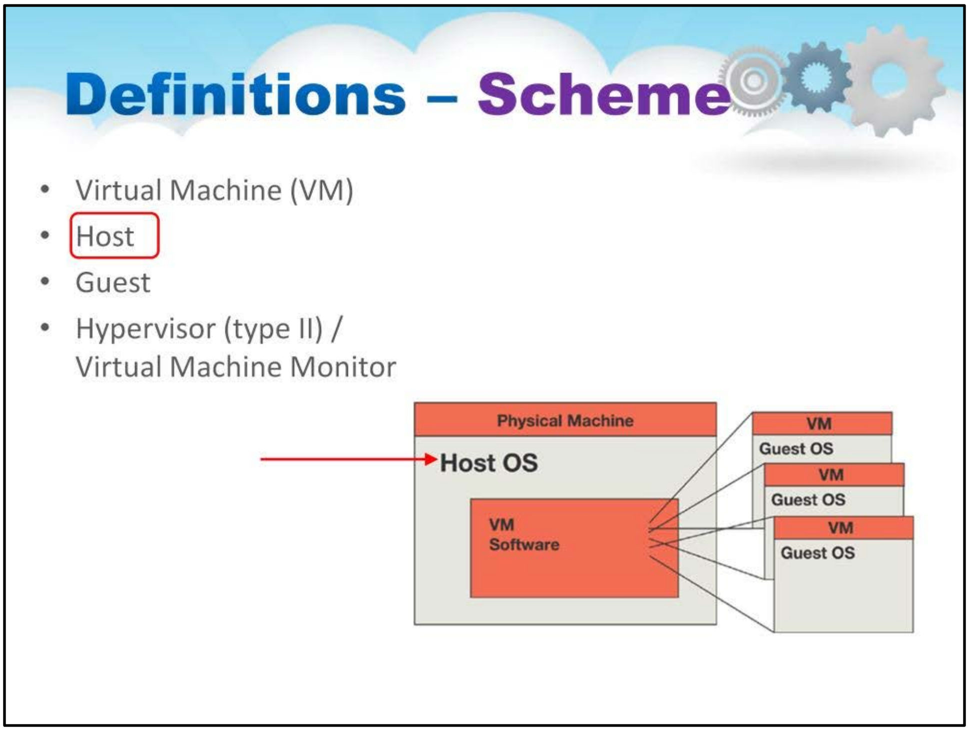
Let’s consider them in the common scheme…



Definitions – Scheme

- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) / Virtual Machine Monitor

Virtual Machine – a software implementation of a machine that executes as if it was running on a physical resource directly.



Definitions – Scheme

- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) / Virtual Machine Monitor

Physical Machine

Host OS

VM Software

VM

Guest OS

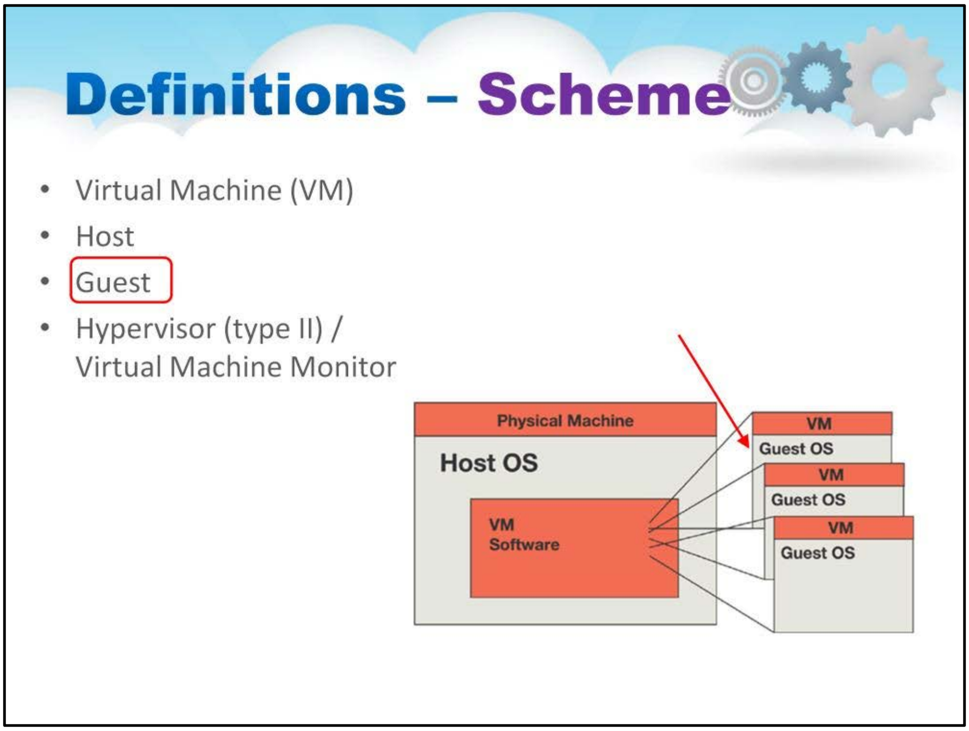
VM

Guest OS

VM

Guest OS

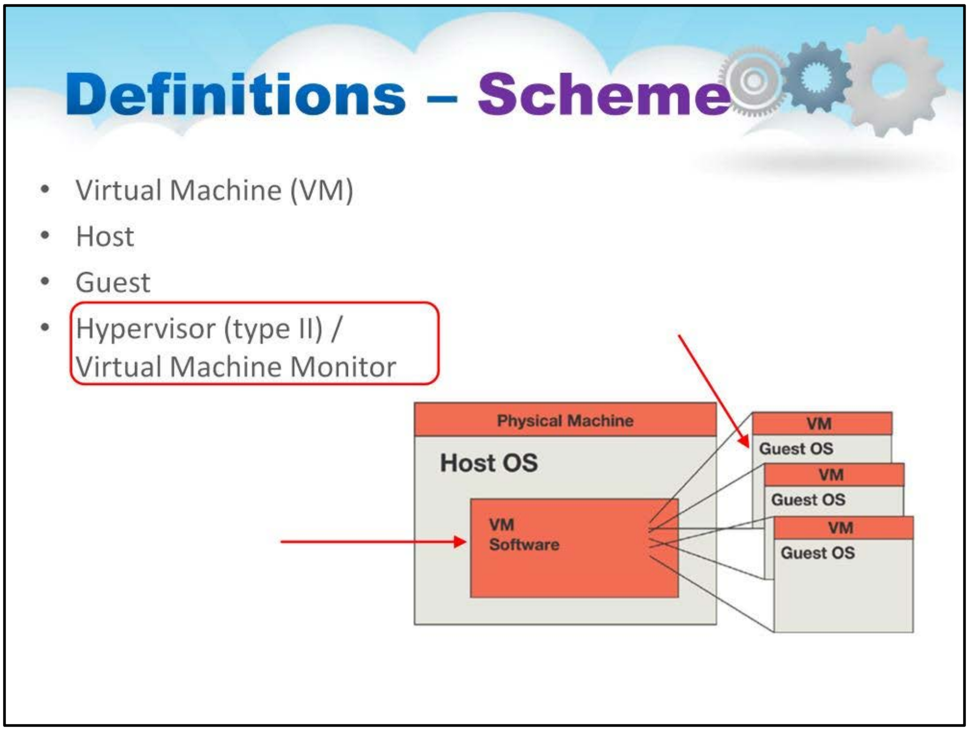
Host – underlying hardware system



Definitions – Scheme

- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) / Virtual Machine Monitor

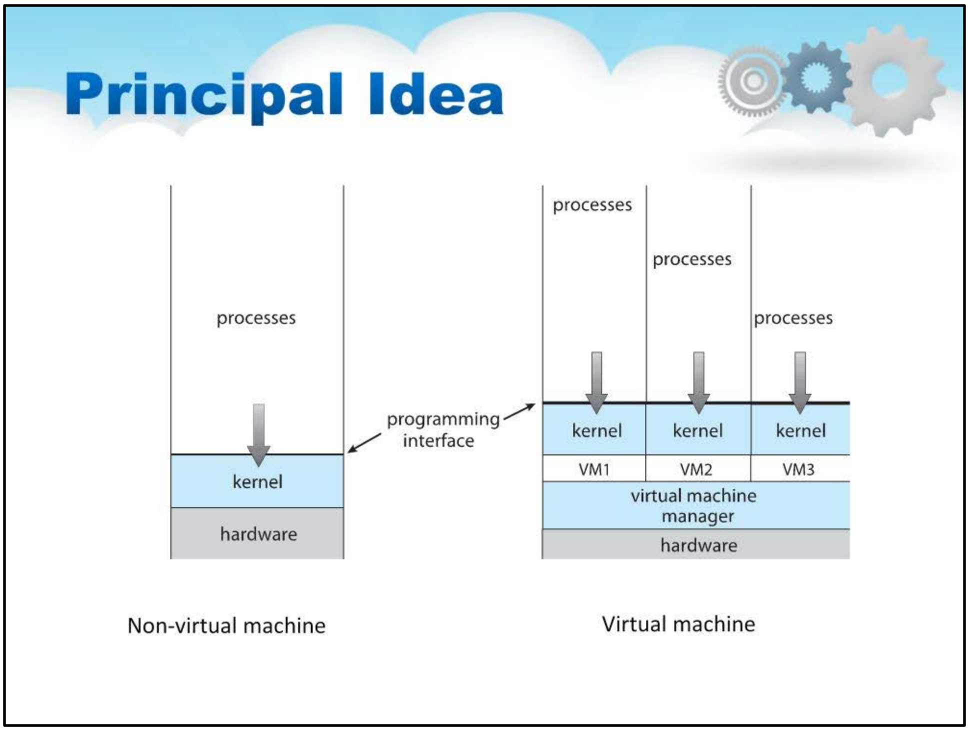
Guest – process provided with virtual copy of the host (usually an operating system).



Definitions – Scheme

- Virtual Machine (VM)
- Host
- Guest
- Hypervisor (type II) /
  Virtual Machine Monitor

Virtual machine manager/monitor (**VMM**) or **hypervisor** – creates and runs
virtual machines by providing interface that is identical to the host (except for the case of paravirtualization)



Principal Idea

Finally, fundamental idea is to abstract hardware of a single computer into several different execution environments

Similar to layered approach

But layer creates virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run

Several components

Host – underlying hardware system

Virtual machine manager (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is identical to the host

(Except in the case of paravirtualization)

Guest – process provided with virtual copy of the host

Usually an operating system

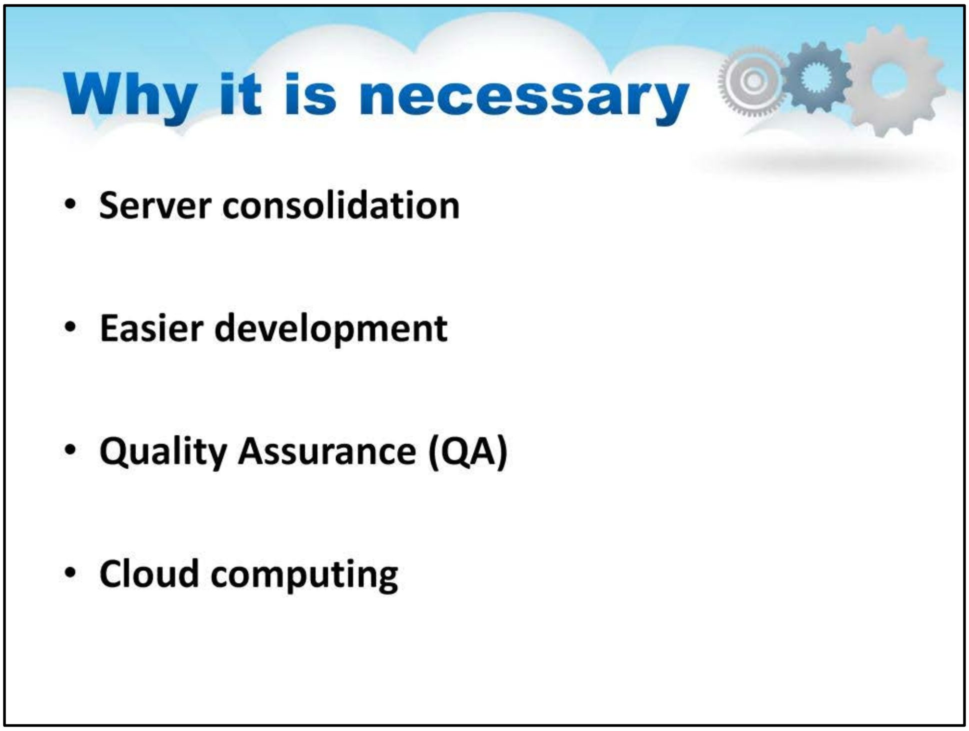
Single physical machine can run multiple operating systems concurrently, each in its own virtual machine.

Virtualization Technologies – Context

Context

This section is about Context of Virtualization Technologies:

- Principal Idea
- Motivation and Usage
- Evolution



Why it is necessary

- Server consolidation
- Easier development
- Quality Assurance (QA)
- Cloud computing

Server consolidation

Run a **web server** and a **mail server** on the **same physical server**

Easier development

Develop critical **operating system components** (file system, disk driver) without affecting **computer stability**

Quality Assurance (QA) as a way of preventing mistakes and defects in manufactured products and avoiding problems when delivering solutions or services:

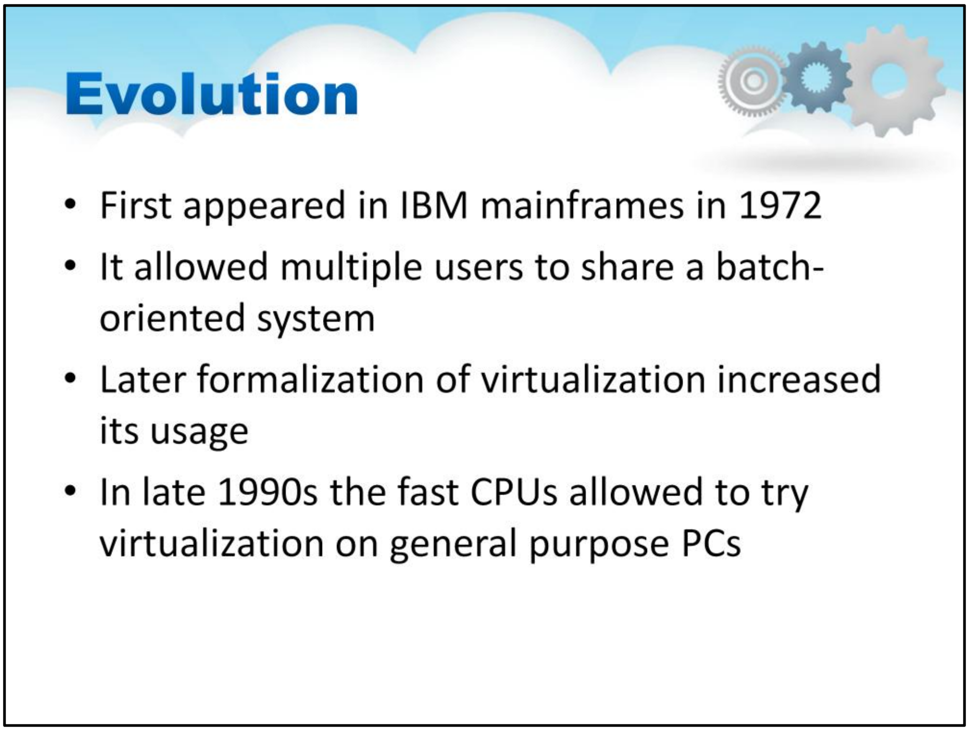
Testing a network product (e.g., a firewall) may require **tens of computers**

Try testing thoroughly a product at each pre-release milestone… and have a straight face when your boss shows you the **electricity bill**

Cloud computing

It is not the buzz-word anymore, but reality

It is necessary to sell computing power and storage



Evolution

- First appeared in IBM mainframes in 1972
- It allowed multiple users to share a batch-oriented system
- Later formalization of virtualization increased its usage
- In late 1990s the fast CPUs allowed to try virtualization on general purpose PCs

First appeared in IBM mainframes in 1972

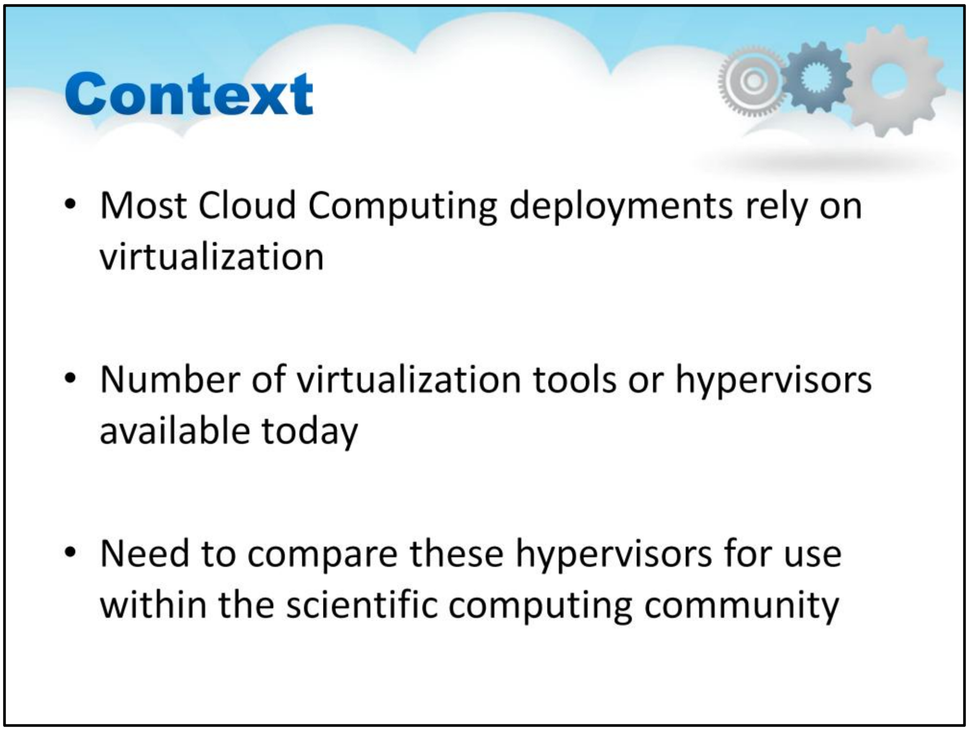
Allowed multiple users to share a batch-oriented system

Formal definition of virtualization helped move it beyond IBM

1. A VMM provides an environment for programs that is essentially identical to the original machine
2. Programs running within that environment show only minor performance decreases
3. The VMM is in complete control of system resources

In late 1990s CPUs fast enough for researchers to try virtualizing on general purpose PCs

Xen and **VMware** created technologies, still used today
Virtualization has expanded to many OSes, CPUs, VMMs



Context

- Most Cloud Computing deployments rely on virtualization
- Number of virtualization tools or hypervisors available today
- Need to compare these hypervisors for use within the scientific computing community

Most Cloud Computing deployments rely on virtualization.

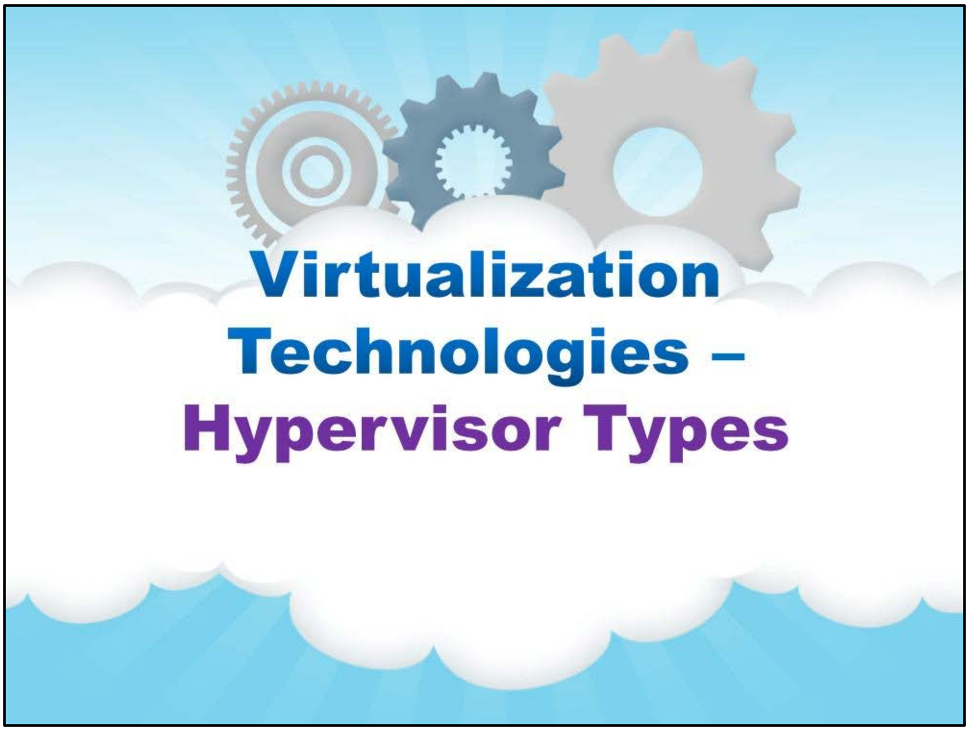
Amazon EC2, GoGrid, Azure, Rackspace Cloud …

Nimbus, Eucalyptus, OpenNebula, OpenStack …

Number of Virtualization tools or Hypervisors available today.

Xen, KVM, VMWare, Virtualbox, Hyper-V …

Need to compare these hypervisors for use within the scientific computing community.

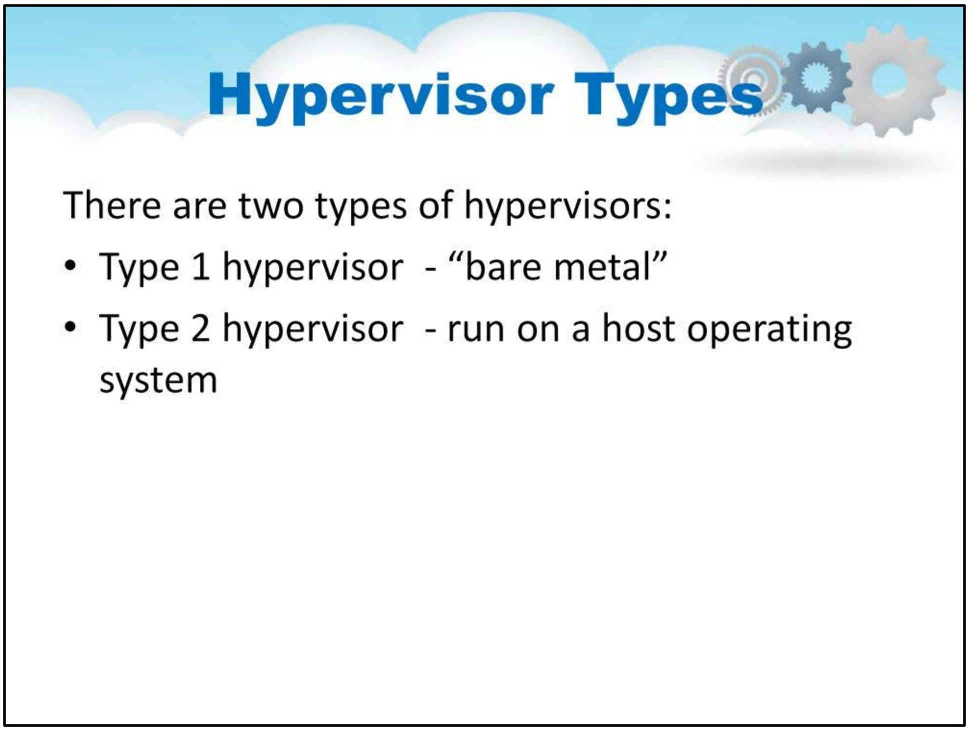


Virtualization Technologies – Hypervisor Types

Hypervisor Types

This section is about **Hypervisor Types**:

- Definitions
- Pro and Contra
- Classification



Hypervisor Types

There are two types of hypervisors:

- Type 1 hypervisor - “bare metal”
- Type 2 hypervisor - run on a host operating system

In 1973, Robert Goldberg classified hypervisors according to their proximity to hardware instructions. Goldberg 's “Type 1” hypervisor was defined as one that translated physical to virtual resources once. Goldberg 's “Type 2” hypervisor was one that made the resource translation twice.

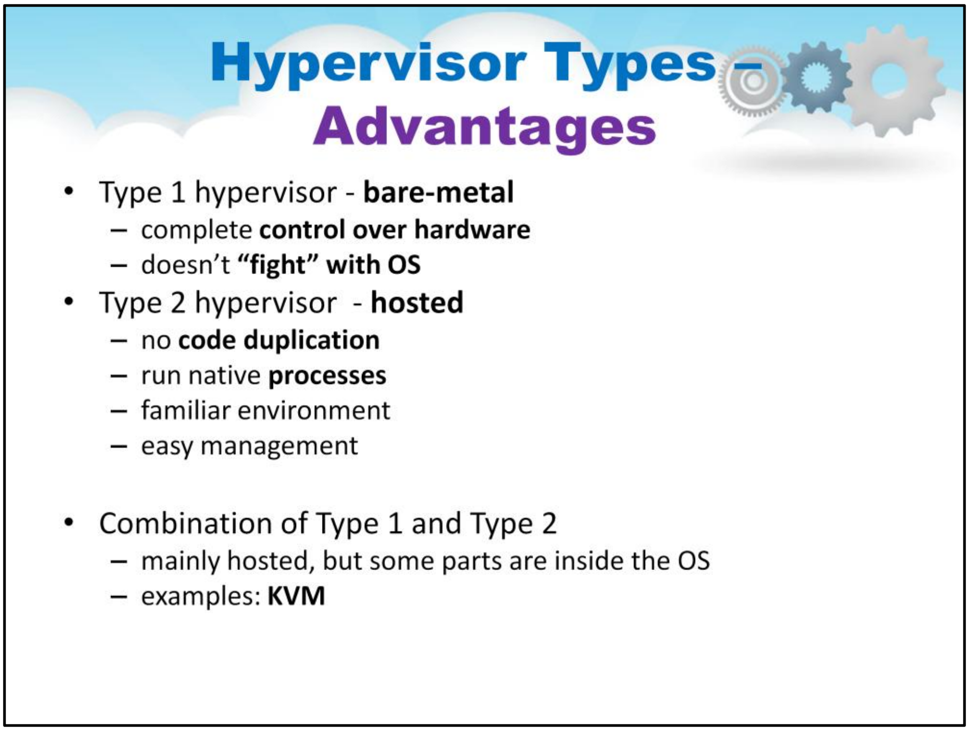
More recently, these definitions were extended to “bare metal” Type 1 hypervisors (running directly on the hardware), versus “hosted” Type 2 hypervisors (running within the operating system).

Later the discussion started: whether a traditional operating system (OS) environment is part of the hypervisor's resource management code, or even whether a traditional OS is visible to or hidden from the administrator.

Finally, the compromise is that there are two types of hypervisors:
Type 1 hypervisor: hypervisors run directly on the system hardware – A “bare metal” embedded hypervisor,
Type 2 hypervisor: hypervisors run on a host operating system that provides virtualization services, such as I/O device support and memory management.

Most hypervisors are based on *full virtualization* which means that they completely emulate all hardware devices to the virtual machines. Guest operating systems do not require any modification and behave as if they each have exclusive access to the entire system.

Full virtualization often includes performance drawbacks because complete emulation usually demands more processing resources (and more overhead) from the hypervisor. Xen is based on *paravirtualization*; it requires that the guest operating systems be modified to support the Xen operating environment. However, the user space applications and libraries do not require modification.



Hypervisor Types – Advantages

- Type 1 hypervisor - **bare-metal**
  - complete **control over hardware**
  - doesn’t **“fight” with OS**
- Type 2 hypervisor - **hosted**
  - no **code duplication**
  - run native **processes**
  - familiar environment
  - easy management
- Combination of Type 1 and Type 2
  - mainly hosted, but some parts are inside the OS
  - examples: **KVM**

Type 1 hypervisor - **bare-metal:**

It has complete **control over hardware**

It doesn’t have to “fight” with operating **system**

Type 2 hypervisor - **hosted:**

Avoid **code duplication**: need not code a **process scheduler**, **memory management** system – the **OS already does** that

Can run native **processes alongside** VMs

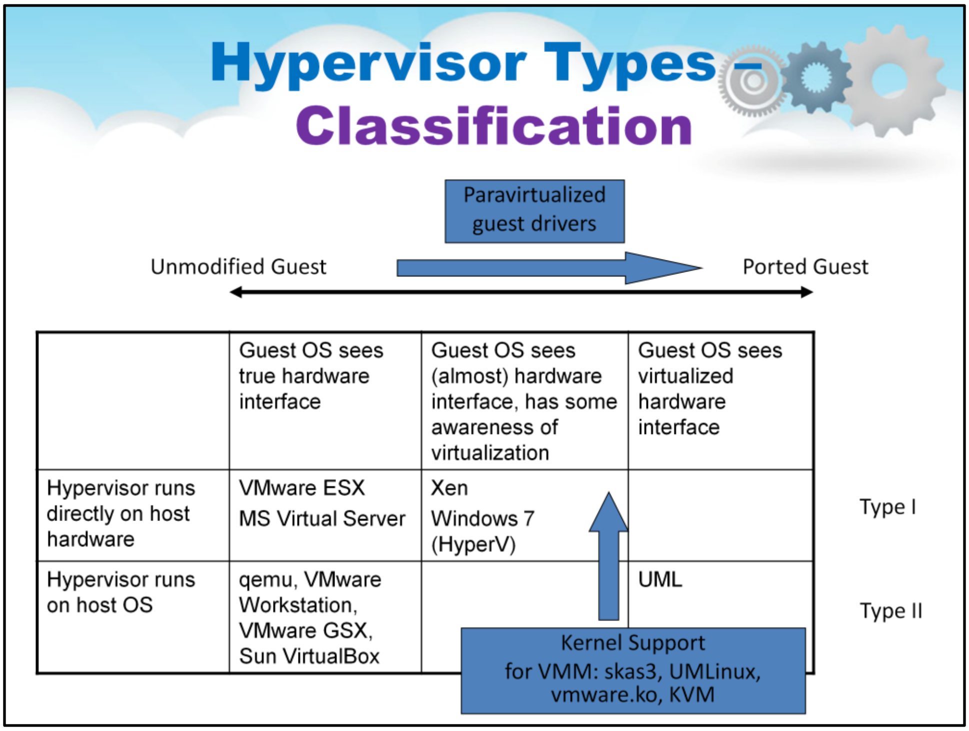
Familiar environment – **how much CPU** and **memory** does a VM take? Use **top**! How big is the **virtual disk**? **ls –l**

Easy management – stop a VM? Sure, just kill it!

A combination of Type 1 and Type 2:

Mostly hosted, but some parts are inside the OS kernel for performance reasons

E.g., **KVM**



Hypervisor Types – Classification

Paravirtualized guest drivers

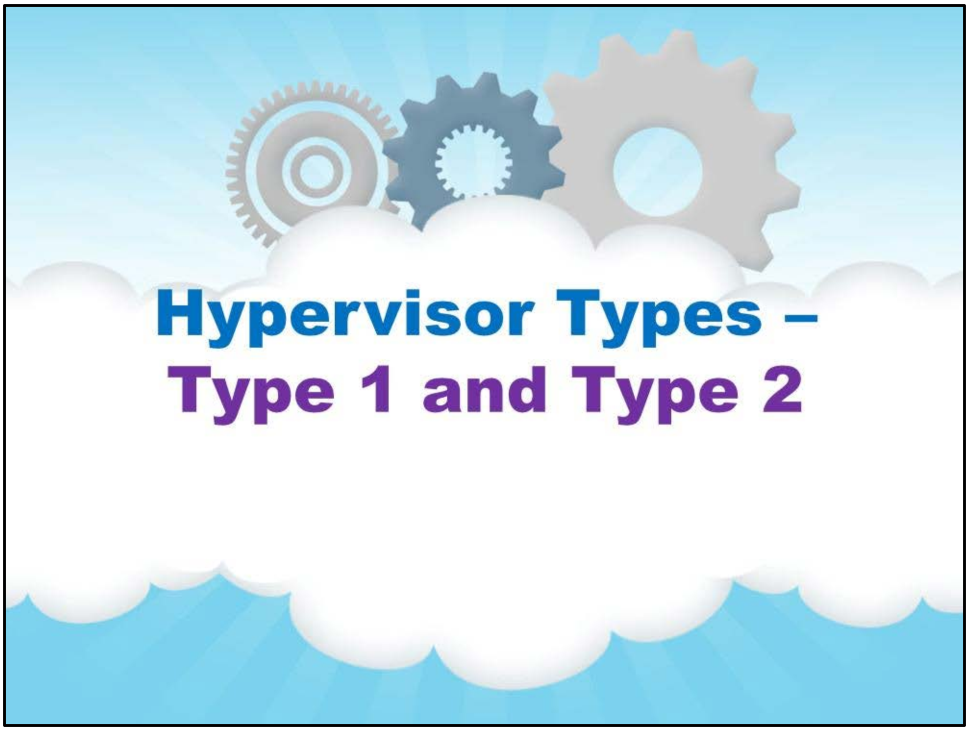
Unmodified Guest

Ported Guest

	Guest OS sees true hardware interface	Guest OS sees (almost) hardware interface, has some awareness of virtualization	Guest OS sees virtualized hardware interface	
Hypervisor runs directly on host hardware	VMware ESX MS Virtual Server	Xen Windows 7 (HyperV)		Type I
Hypervisor runs on host OS	qemu, VMware Workstation, VMware GSX, Sun VirtualBox		UML	Type II

Kernel Support for VMM: skas3, UMLinux, vmware.ko, KVM

Here the classification of hypervisors is summarized in the common comparison table with some examples of implementation.

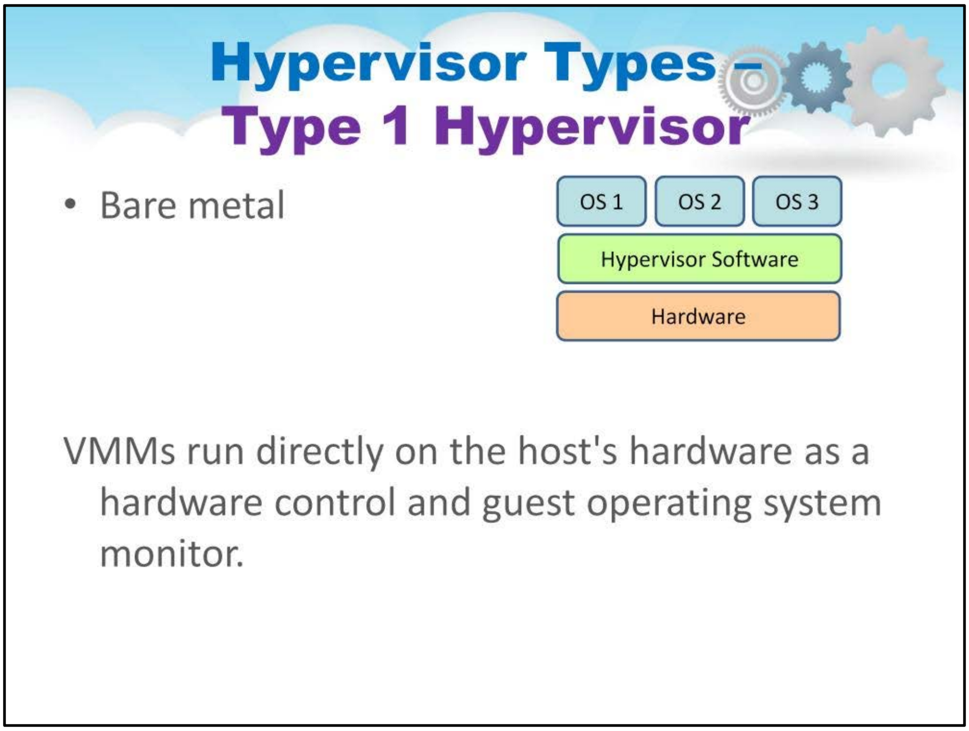


Hypervisor Types –
Type 1 and Type 2

Type 1 and Type 2 in Details

This section is dedicated to the more detailed description of Hypervisor Types 1 and 2:

- Structure
- Pro and Contra
- Examples:
  - ✓ Xen solution
  - ✓ KVM solution



Hypervisor Types – Type 1 Hypervisor

- Bare metal

OS 1 OS 2 OS 3

Hypervisor Software

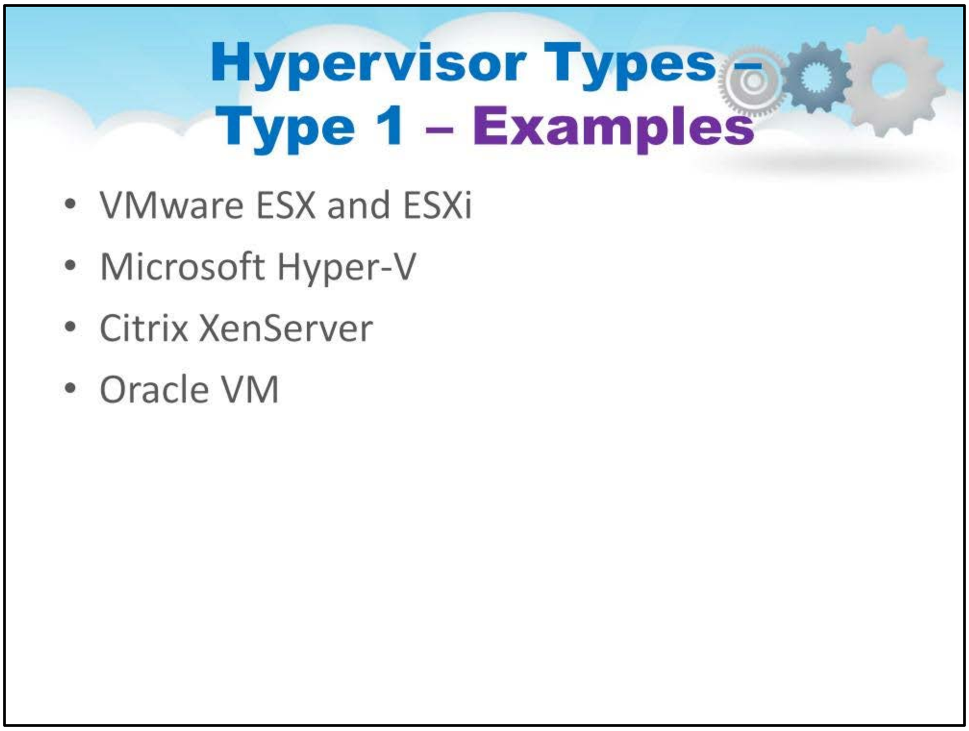
Hardware

VMMs run directly on the host's hardware as a hardware control and guest operating system monitor.

A Type 1 hypervisor has one less layer than a Type 2 hypervisor and sits directly on the host hardware (see Figure 2). Amazon’s well-known Elastic Compute Cloud (EC2) is a web service enabled through a Type 1 hypervisor. Using a Xen hypervisor on top of powerful servers, Amazon can offer “slices” of scalable compute power to its customers, allowing each slice to operate in a virtualized independent environment.

A Type 1 hypervisor may be thinner that its Type 2 counterpart, but it typically requires modifications to the guest OS. Thus, the embedded developer must often use a variant of the guest OS created specifically for the hypervisor.

Many Type 1 hypervisors rely on hardware than has virtualization capabilities. Intel’s VT, Freescale’s embedded hypervisor technology, and ARM’s TrustZone are all technologies that provide the foundation for virtualization solutions. As a result, some hardware vendors have joined the hypervisor movement and promote Type I hypervisors optimized for their hardware platforms. This approach offers an attractive alternative for systems that demand higher performance.



Hypervisor Types – Type 1 – Examples

- VMware ESX and ESXi
- Microsoft Hyper-V
- Citrix XenServer
- Oracle VM

1. VMware ESX and ESXi

These hypervisors offer advanced features and scalability, but require licensing, so the costs are higher. There are some lower-cost bundles that VMware offers and they can make hypervisor technology more affordable for small infrastructures.

VMware is the leader in the Type-1 hypervisors. Their vSphere/ESXi product is available in a free edition and 5 commercial editions.

2. Microsoft Hyper-V

The Microsoft hypervisor, Hyper-V doesn’t offer many of the advanced features that VMware’s products provide. However, with XenServer and vSphere, Hyper-V is one of the top 3 Type-1 hypervisors.

It was first released with Windows Server, but now Hyper-V has been greatly enhanced with Windows Server 2012 Hyper-V. Hyper-V is available in both a free edition (with no GUI and no virtualization rights) and 4 commercial editions – Foundations (OEM only), Essentials, Standard, and Datacenter. Hyper-V

3. Citrix XenServer

It began as an open source project.

The core hypervisor technology is free, but like VMware’s free ESXi, it has almost no advanced features.

Xen is a type-1 bare-metal hypervisor. Just as Red Hat Enterprise Virtualization uses KVM, Citrix uses Xen in the commercial XenServer.

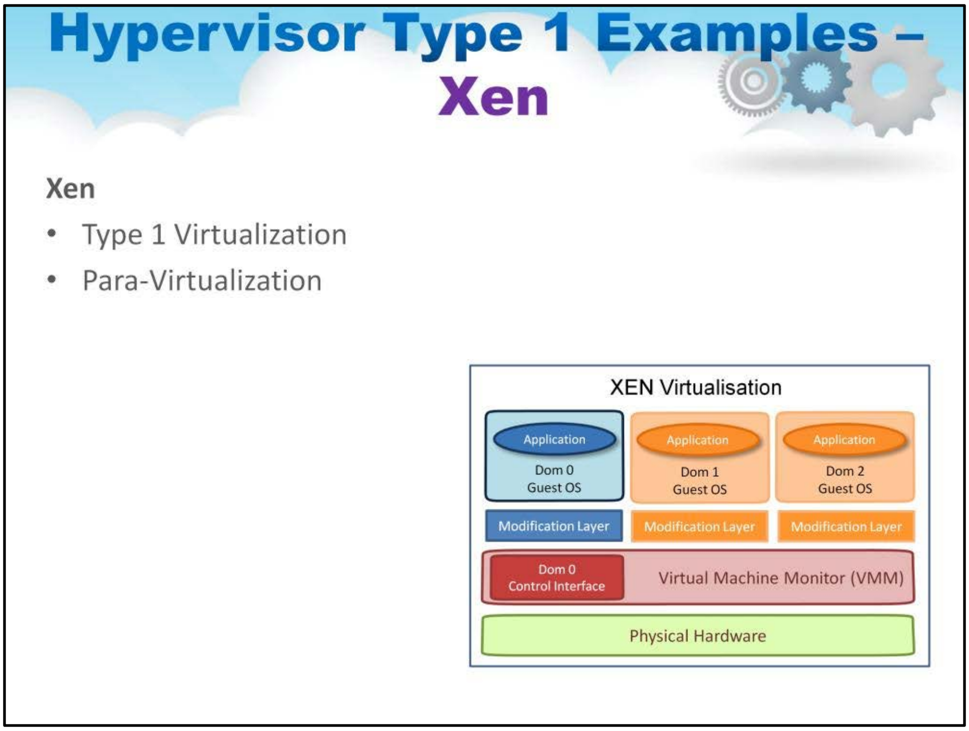
Today, the Xen open source projects and community are at Xen.org. Today, XenServer is a commercial type-1 hypervisor solution from Citrix, offered in 4 editions. Confusingly, Citrix has also branded their other proprietary solutions like XenApp and XenDesktop with the Xen name.

4. Oracle VM

The Oracle hypervisor is based on the open source Xen.

However, if you need hypervisor support and product updates, it will cost you.

Oracle VM lacks many of the advanced features found in other bare-metal virtualization hypervisors.



Hypervisor Type 1 Examples – Xen

Xen

- Type 1 Virtualization
- Para-Virtualization

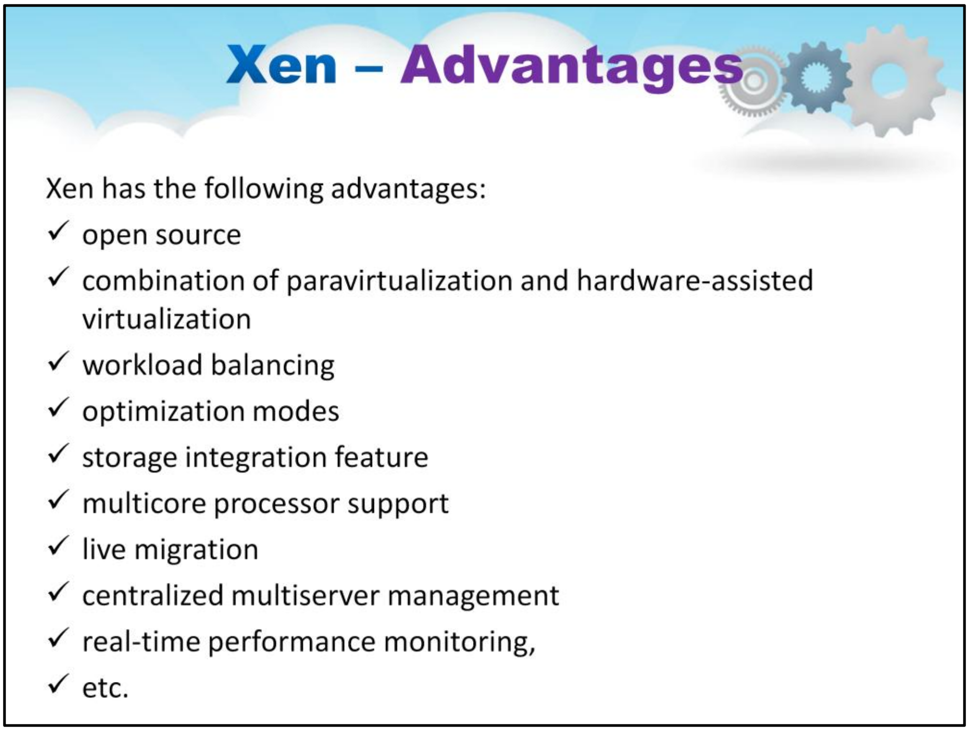
Here are some examples of Hypervisors, showing the variation in architectures.

The first hypervisor illustrated is XEN.

One can see it is Type 1 Virtualization where VMMs run directly on the host's hardware as a hardware control and guest operating system monitor.

Also it can be seen that it is a Para-Virtualization system where the VMM does not necessarily simulate hardware, but instead offers a special API that can only be used by the modified guest OS.
Paravirtualization is a virtualization technique that presents a software interface to the virtual machines that is similar to but not identical to that of the underlying hardware. The aim is to reduce the portion of the guest operating system's execution time that is spent performing operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment.

Xen can run several guest operating systems each running in its own virtual machine or domain. When Xen is first installed, it automatically creates the first domain, Domain 0 (or dom0).
Domain 0 is the management domain and is responsible for managing the system. It performs tasks like building additional domains (or virtual machines), managing the virtual devices for each virtual machine, suspending virtual machines, resuming virtual machines, and migrating virtual machines. Domain 0 runs a guest operating system and is responsible for the hardware devices.



Xen – Advantages

Xen has the following advantages:

- ✓ open source
- ✓ combination of paravirtualization and hardware-assisted virtualization
- ✓ workload balancing
- ✓ optimization modes
- ✓ storage integration feature
- ✓ multicore processor support
- ✓ live migration
- ✓ centralized multiserver management
- ✓ real-time performance monitoring,
- ✓ etc.

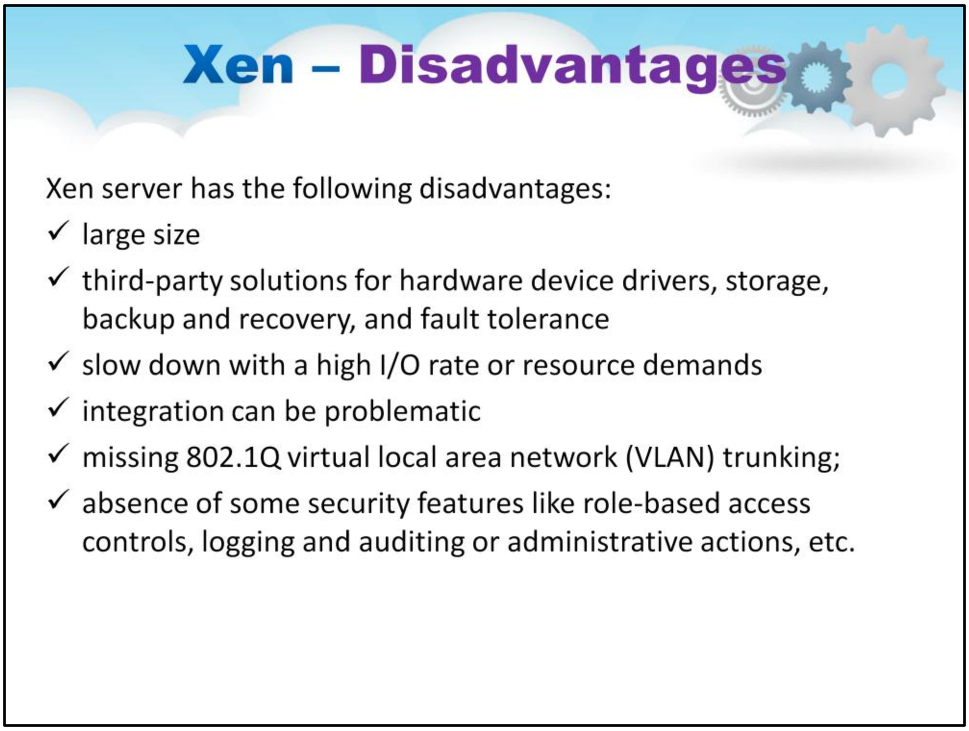
Xen has the following advantages:

✓ is built on the open source Xen hypervisor and uses a combination of paravirtualization and hardware-assisted virtualization. This collaboration between the OS and the virtualization platform enables the development of a simpler hypervisor that delivers highly optimized performance.

✓ provides sophisticated workload balancing that captures CPU, memory, disk I/O, and network I/O data; it offers two optimization modes: one for performance and another for density.

✓ takes advantage of a unique storage integration feature called the Citrix Storage Link. With it, the sysadmin can directly leverage features of arrays from such companies as HP, Dell Equal Logic, NetApp, EMC, and others.

✓ includes multicore processor support, live migration, physical-server-to-virtual-machine conversion (P2V) and virtual-to-virtual conversion (V2V) tools, centralized multiserver management, real-time performance monitoring, and speedy performance for Windows and Linux.



Xen – Disadvantages

Xen server has the following disadvantages:

✓ large size

✓ third-party solutions for hardware device drivers, storage, backup and recovery, and fault tolerance

✓ slow down with a high I/O rate or resource demands

✓ integration can be problematic

✓ missing 802.1Q virtual local area network (VLAN) trunking;

✓ absence of some security features like role-based access controls, logging and auditing or administrative actions, etc.

Xen server has the following disadvantages:

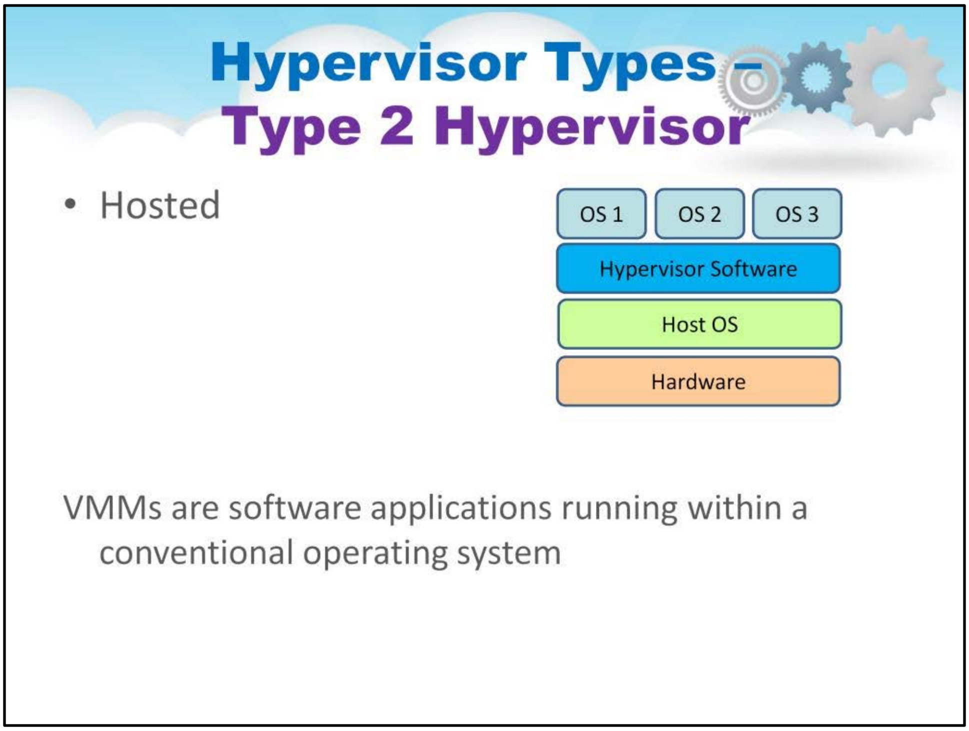
✓ it has a relatively large footprint

✓ it relies on third-party solutions for hardware device drivers, storage, backup and recovery, and fault tolerance

✓ it gets slow down with anything with a high I/O rate or anything that sucks up resources and starves other VMs

✓ its integration can be problematic; it could become a burden on your Linux kernel over time

✓ it is missing 802.1Q virtual local area network (VLAN) trunking; as for security, it doesn't offer directory services integration, role-based access controls, or security logging and auditing or administrative actions.



Hypervisor Types – Type 2 Hypervisor

- Hosted

OS 1 | OS 2 | OS 3

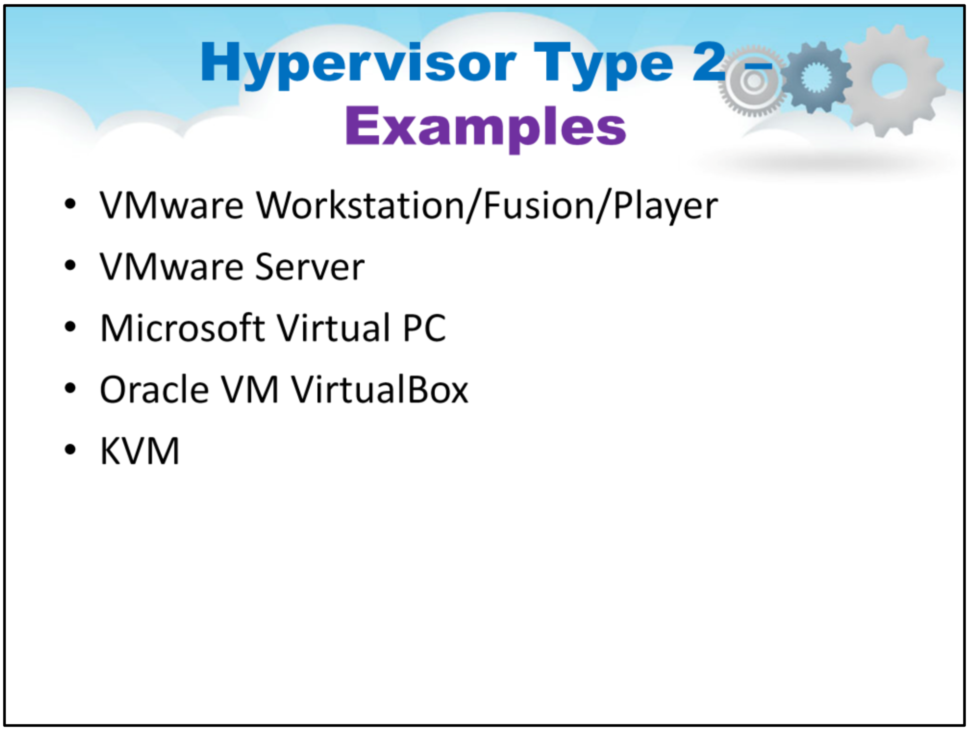
Hypervisor Software

Host OS

Hardware

VMMs are software applications running within a conventional operating system

Type 2 hypervisors are easy to use — they typically require no modification to the guest OS. In the case of VMWare, the Workstation software emulates the underlying hardware and provides a full set of virtual hardware resources (mapped by the hypervisor to physical resources) to the guest OS. If the guest OS can run on an x86 machine, it can run within Workstation without modifications. The convenience provided by this type of hypervisor comes with a trade-off: added processing overhead that can compromise real-time performance.



Hypervisor Type 2 – Examples

- VMware Workstation/Fusion/Player
- VMware Server
- Microsoft Virtual PC
- Oracle VM VirtualBox
- KVM

1. VMware Workstation/Fusion/Player

VMware Player is a free virtualization hypervisor. It is intended to run only one virtual machine (VM) and does not allow creating VMs. VMware Workstation is a more robust hypervisor with some advanced features, such as record-and-replay and VM snapshot support. VMware Workstation has three major use cases:

for running multiple different operating systems or versions of one OS on one desktop,

for developers that need sandbox environments and snapshots, or

for labs and demonstration purposes.

2. VMware Server

VMware Server is a free, hosted virtualization hypervisor that's very similar to the VMware Workstation.

3. Microsoft Virtual PC

This is the latest Microsoft's version of this hypervisor technology, Windows Virtual PC and runs only on Windows 7 and supports only Windows operating systems running on it.

4. Oracle VM VirtualBox

VirtualBox hypervisor technology provides reasonable performance and features if you want to virtualize on a budget. Despite being a free, hosted product with a very small footprint, VirtualBox shares many features with VMware vSphere and Microsoft Hyper-V.

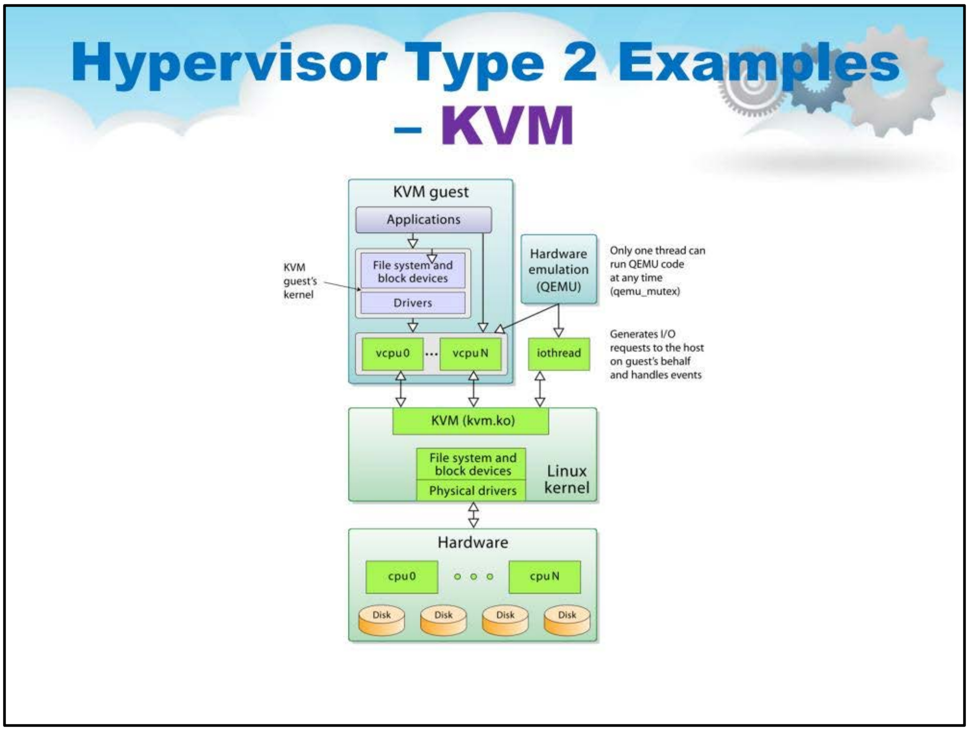
5. Red Hat Enterprise Virtualization

Red Hat's Kernel-based Virtual Machine (KVM) has qualities of both a hosted and a bare-metal virtualization hypervisor. It can turn the Linux kernel itself into a hypervisor so the VMs have direct access to the physical hardware.

KVM

This is a virtualization infrastructure for the Linux kernel. It supports native virtualization on processors with hardware virtualization extensions.

The open-source KVM (or Kernel-Based Virtual Machine) is a Linux-based type-1 hypervisor that can be added to most Linux operating systems including Ubuntu, Debian, SUSE, and Red Hat Enterprise Linux, but also Solaris, and Windows.



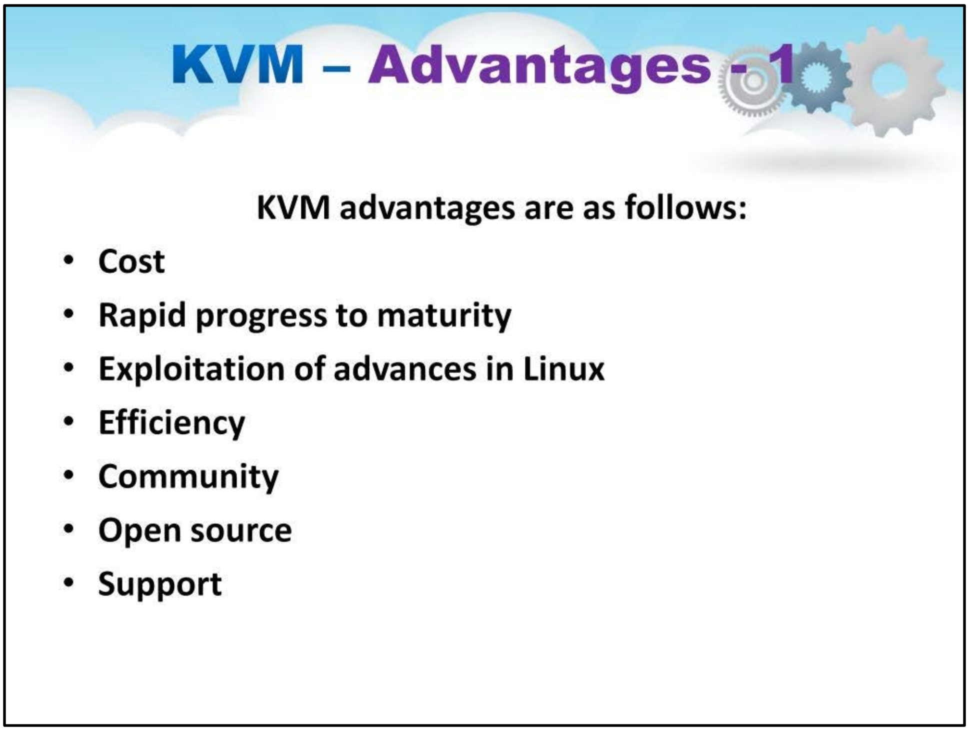
Hypervisor Type 2 Examples – KVM

Only one thread can run QEMU code at any time (qemu_mutex)

Generates I/O requests to the host on guest's behalf and handles events

KVM is an acronym of “Kernel based Virtual Machine”, and is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor.
The basic architecture for KVM is as follows.

The Kernel-based Virtual Machine (KVM) is a Type 2 hypervisor maintained by Qumranet, Inc.
KVM is based on the QEMU emulator and derives all its management tools from QEMU.
The main focus of KVM development is to use the x86 VT extensions, which allow virtual machines to make system calls.
KVM versions newer than KVM-62 have support for paravirtualized Linux guests.
KVM uses a set of Linux kernel modules to provide VT support.
KVM can run on a stock Linux kernel that is: (a) new enough and (b) has had the KVM modules built for it.
KVM is used with QEMU to emulate some peripherals, called QEMU-KVM.
KVM supports the QEMU Copy-on-write (QCOW) disk image format, allowing it to support a snapshot mode for its disk I/O operations.
In snapshot mode, all disk writes are directed to a temporary file, and changes are not persisted to the original disk image file.
Multiple VM's can be run from one disk image, somewhat mitigating the huge storage requirements associated with hosting a grid of VM's.
Destroying a virtual cluster is as simple as sending SIGKILL to each hypervisor and deleting the image from disk.
KVM supports the standard Linux TUN/TAP model for Ethernet bridging. By using this model, each VM gets its own networking resources, making it indistinguishable from a physical machine.



KVM – Advantages - 1

KVM advantages are as follows:

- Cost
- Rapid progress to maturity
- Exploitation of advances in Linux
- Efficiency
- Community
- Open source
- Support

KVM advantages are as follows:

Cost: Given its open source nature, KVM has a lower total cost of ownership.

Rapid progress to maturity: A community of experts continuously enhances KVM.

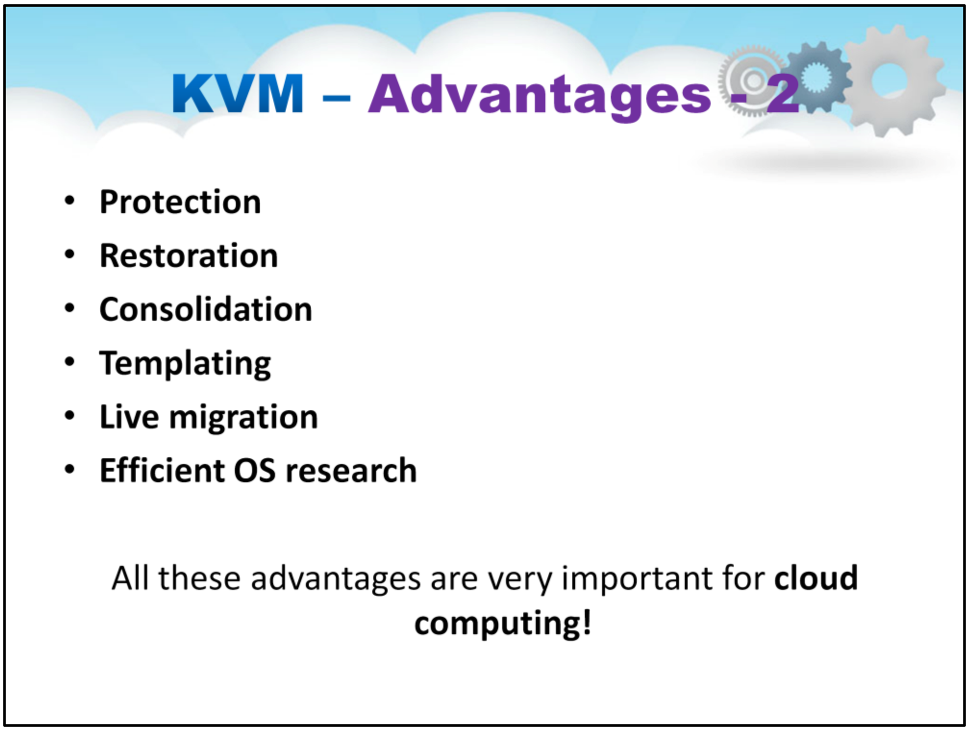
Exploitation of advances in Linux: KVM is built into Linux and benefits from the entire Linux community.

Efficiency: KVM takes advantage of modern hardware design to securely execute directly on the host CPU, and is engineered to perform well even in memory- and CPU-constrained environments.

Community: Customer feature requirements and security vulnerabilities are quickly addressed by very active and responsive community.

Open source: The code and its repository data are available, continuously inspected, and transparent in modification rationale throughout the product life cycle.

Support: some big players (like IBM) are the active KVM community members, and it is influential in setting KVM development priorities



KVM – Advantages - 2

- Protection
- Restoration
- Consolidation
- Templating
- Live migration
- Efficient OS research

All these advantages are very important for **cloud computing!**

Protection - host system protected from VMs, VMs protected from each other
A virus less likely to spread
Sharing is provided though via shared file system volume, network communication

Restoration - it is possible to freeze, **suspend**, running VM
Then can move or copy somewhere else and **resume**
To make snapshot of a given state, able to restore back to that state
Some VMMs allow multiple snapshots per VM
Clone by creating copy and running both original and copy

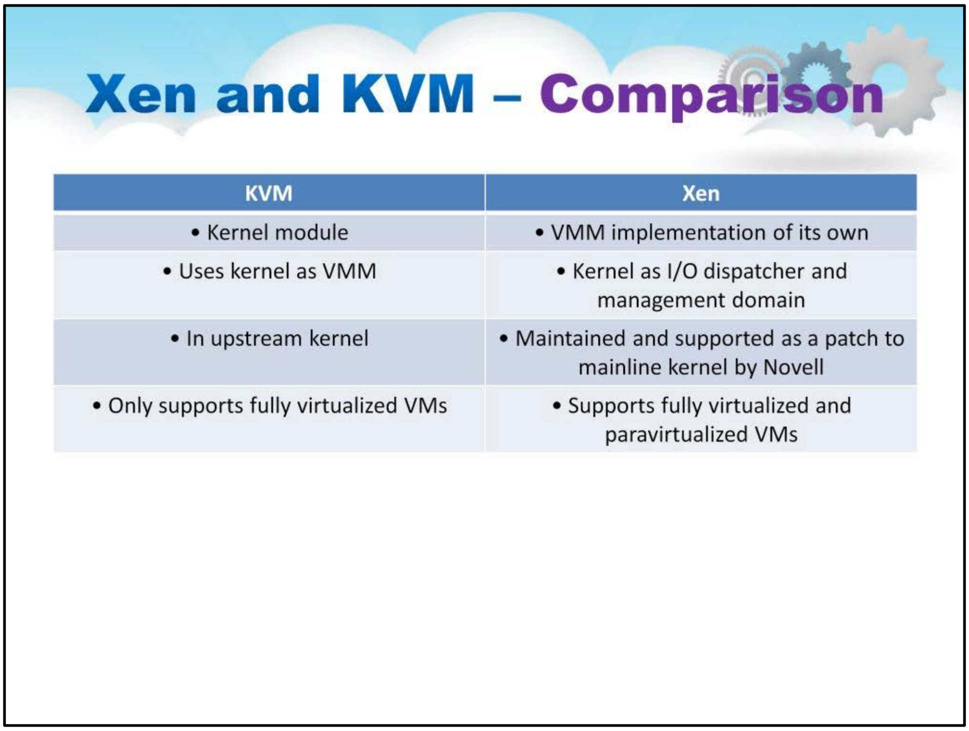
Consolidation – can run multiple, different OSes on a single machine
OS dev, app dev, …

Templating – to create an OS + application VM, provide it to customers, use it to create multiple instances of that combination

Live migration – move a running VM from one host to another!

Efficient OS research – allow for the better operating system development efficiency

All these advantages are very important for **cloud computing, because** using APIs programs can work in cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops!



Xen and KVM – Comparison

KVM	Xen
• Kernel module	• VMM implementation of its own
• Uses kernel as VMM	• Kernel as I/O dispatcher and management domain
• In upstream kernel	• Maintained and supported as a patch to mainline kernel by Novell
• Only supports fully virtualized VMs	• Supports fully virtualized and paravirtualized VMs

Let’s compare Xen or KVM – please, this table with some observations of their features.
Which hypervisor is best for your cloud: Xen or KVM?

Xen

Xen began around 2004 when there was no existing open source virtualization.
Xen, which is a Type I Hypervisor, sits at the single layer above bare metal. It’s like a stripped down operating system, and it uses a function called “pass-through” to connect directly to PCI devices like RAM/CPU/NICs.

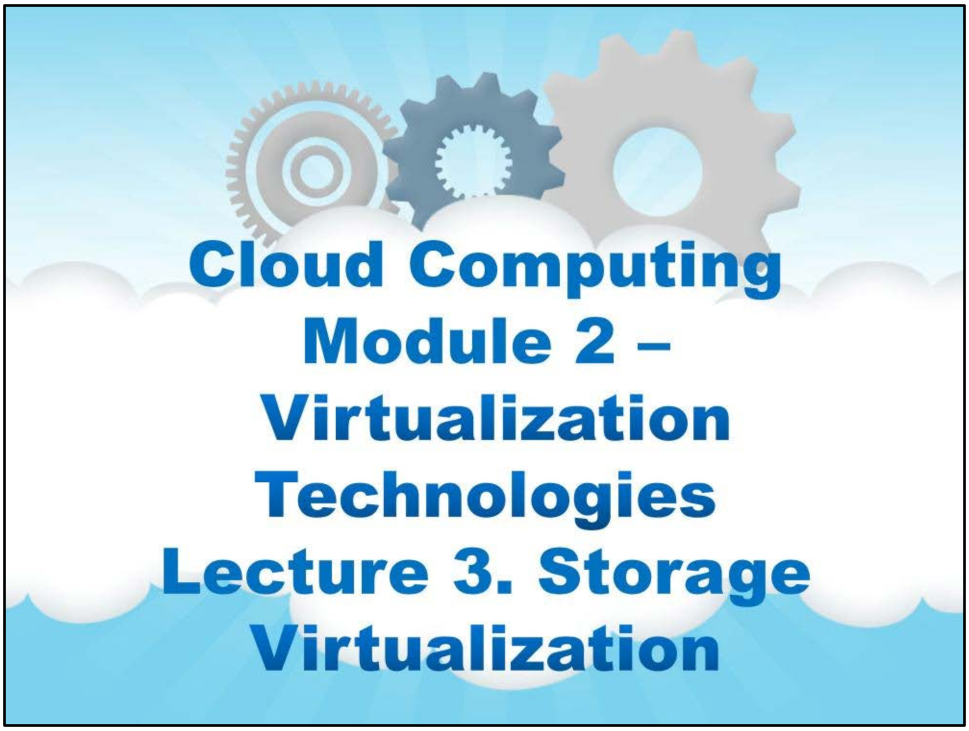
KVM

KVM (Kernel-based Virtual Machine) was originally written and released around 2007.
KVM, which is a Type II Hypervisor, sits one layer above the OS. If you have a bare metal machine, you need to install an OS, and then to install KVM.
In Linux it is installed in the form of a Kernel, and this kernel then turns the bare metal machine into a Hypervisor.

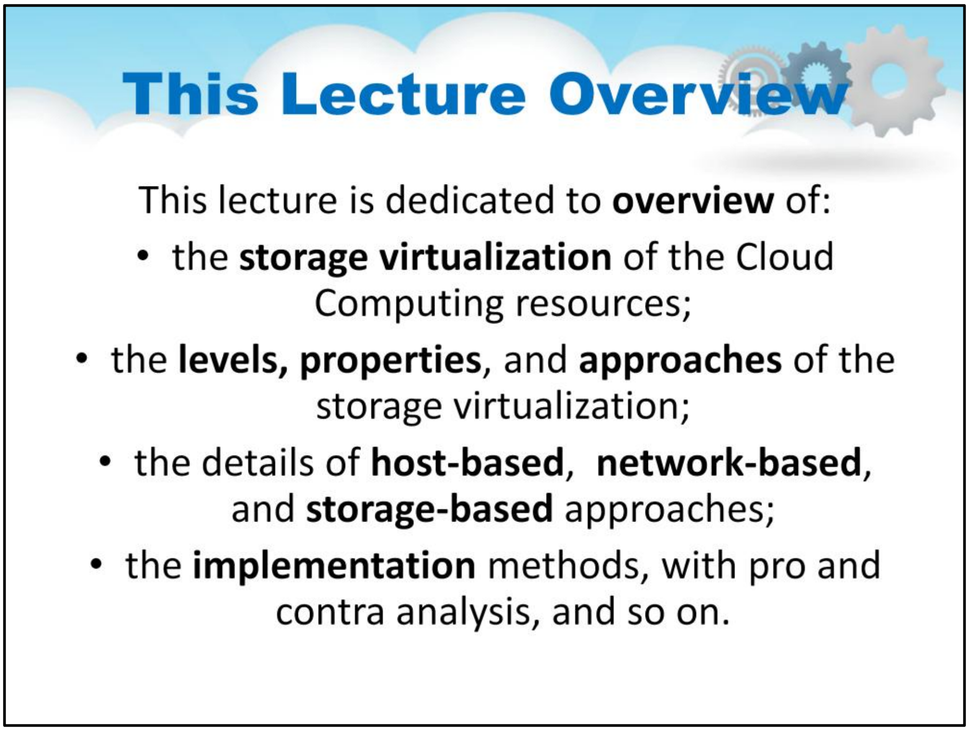
They are both open source, and both supported by large communities and large enterprises.
They both do the fundamental job of virtualization.
The differences are found in support for innovation, and in application integration, application awareness and performance.
Xen tends to be more stable. It is older and more mature. Xen offers near native drivers for OSs like Microsoft Windows, whereas KVM is weaker in those regards. However, KVM does very well with Linux.

The general advice:

If you’re dealing with a lot of Linux-based solutions, you’d go with KVM. This doesn’t mean Xen is bad. It’s just that chances are, you’ll find KVM will offer better performance with the Linux OS.
If you’re dealing with a lot of Windows-based solutions, you’d go with Xen, because there are more advantages to hosting on Xen due to the near native drivers.



Cloud Computing Module 2 – Virtualization Technologies Lecture 3. Storage Virtualization

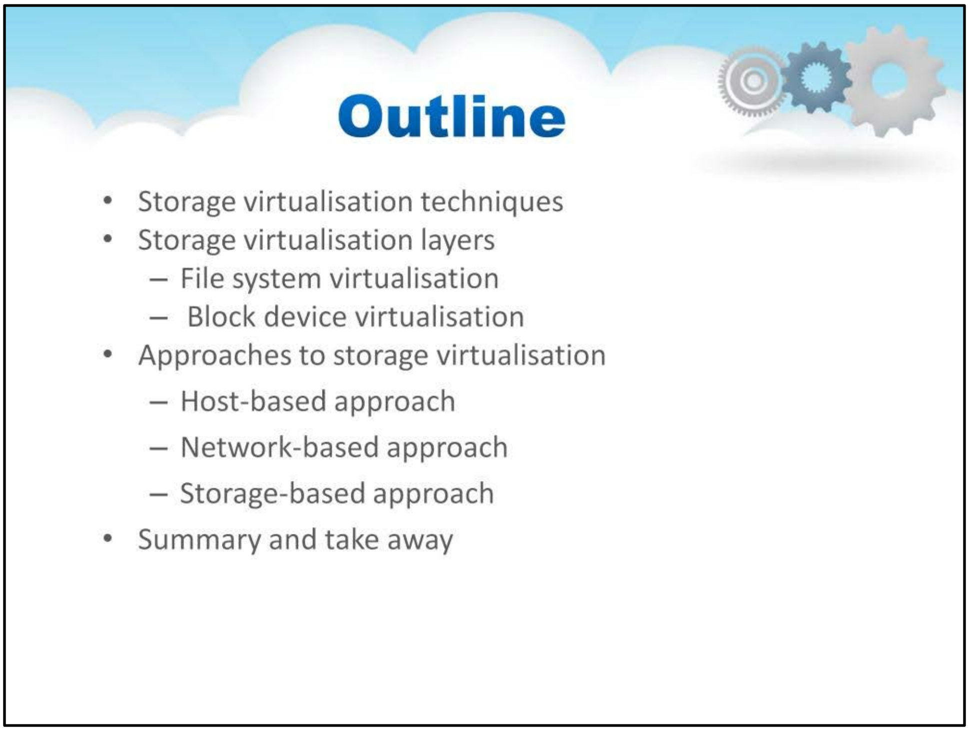


This Lecture Overview

This lecture is dedicated to **overview** of:

- the **storage virtualization** of the Cloud Computing resources;
- the **levels, properties**, and **approaches** of the storage virtualization;
- the details of **host-based**, **network-based**, and **storage-based** approaches;
- the **implementation** methods, with pro and contra analysis, and so on.

Lecture 3. Storage Virtualization



Outline

- Storage virtualisation techniques
- Storage virtualisation layers
  - File system virtualisation
  - Block device virtualisation
- Approaches to storage virtualisation
  - Host-based approach
  - Network-based approach
  - Storage-based approach
- Summary and take away

As you can see from the slide, we will be covering several areas relating to Storage Virtualisation in Cloud IaaS

Storage virtualisation techniques

Storage virtualisation layers

Approaches to storage virtualisation

Storage types in cloud

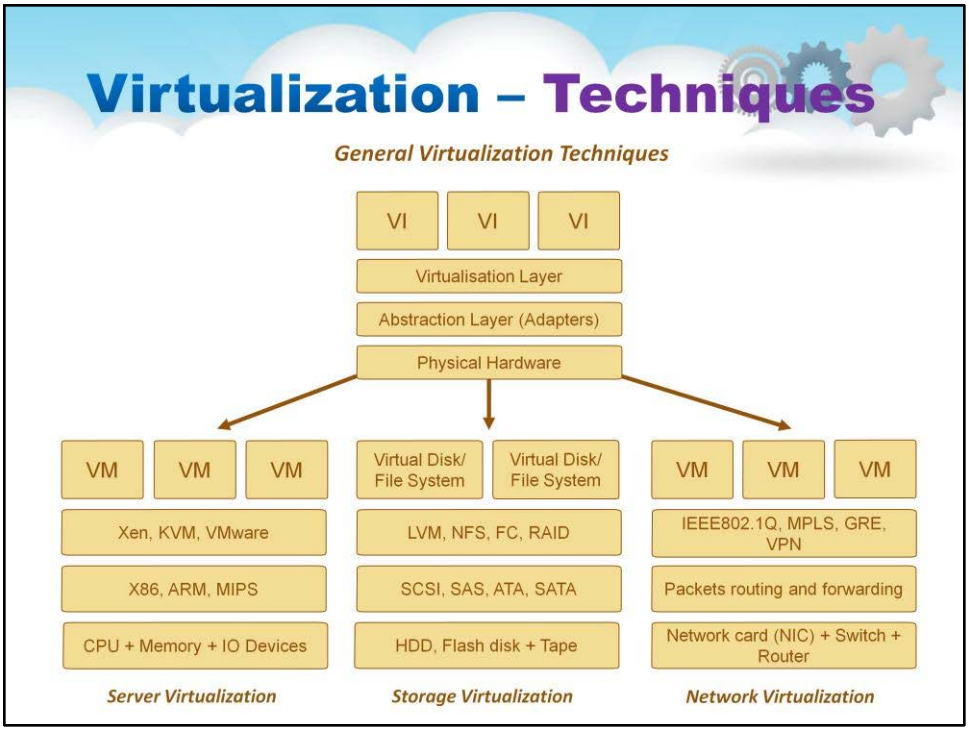
Virtualised file systems for cloud

Storage Virtualization

–

Overview

Overview



Virtualization – Techniques

General Virtualization Techniques

VI VI VI

Virtualisation Layer

Abstraction Layer (Adapters)

Physical Hardware

Server Virtualization	Storage Virtualization	Network Virtualization
VM VM VM	Virtual Disk/ File System Virtual Disk/ File System	VM VM VM
Xen, KVM, VMware	LVM, NFS, FC, RAID	IEEE802.1Q, MPLS, GRE, VPN
X86, ARM, MIPS	SCSI, SAS, ATA, SATA	Packets routing and forwarding
CPU + Memory + IO Devices	HDD, Flash disk + Tape	Network card (NIC) + Switch + Router

VI - Virtualised Instance
VM - Virtual Machine

This slide illustrates Virtualization techniques in Cloud IaaS

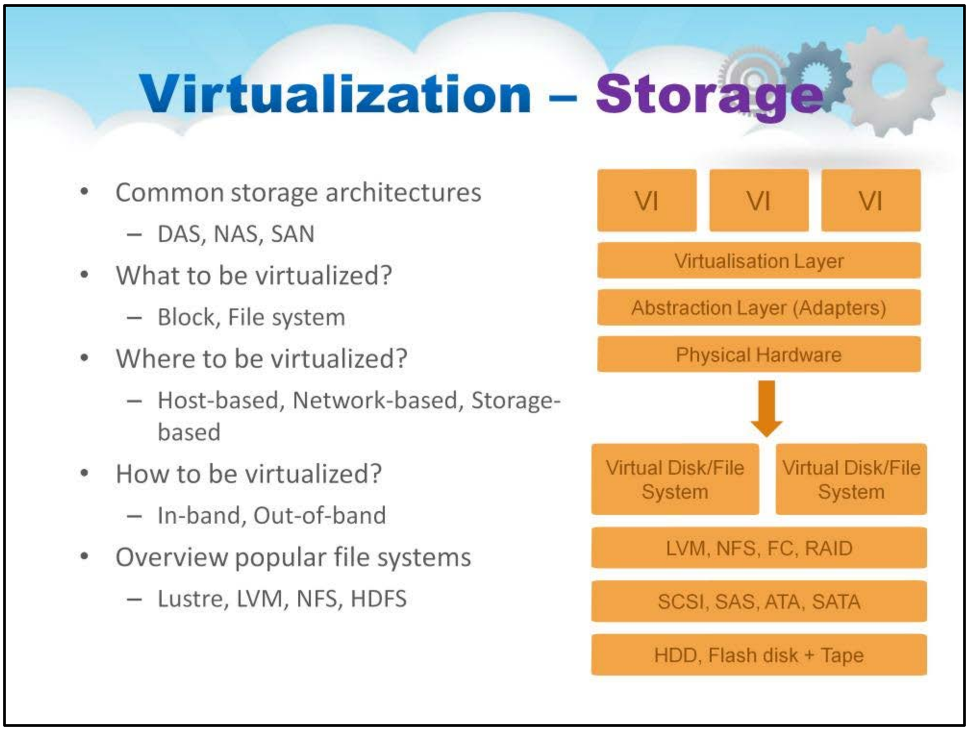
At the top the General Virtualization Technique is illustrated.

Underneath, one can visualize how the general technique is applied to several virtualization problems.

For Server virtualization, hypervisors are extensively used.

For storage virtualization, many software techniques including volume management, file systems, and replication are applied.

For network virtualization, there are many different features including link aggregation, VPN, and also firewall, switching, routing, and application filtering and load balancing have virtual capability in many cloud implementations today.



Virtualization – Storage

- Common storage architectures
  - DAS, NAS, SAN
- What to be virtualized?
  - Block, File system
- Where to be virtualized?
  - Host-based, Network-based, Storage-based
- How to be virtualized?
  - In-band, Out-of-band
- Overview popular file systems
  - Lustre, LVM, NFS, HDFS

VI VI VI

Virtualisation Layer

Abstraction Layer (Adapters)

Physical Hardware

Virtual Disk/File System

Virtual Disk/File System

LVM, NFS, FC, RAID

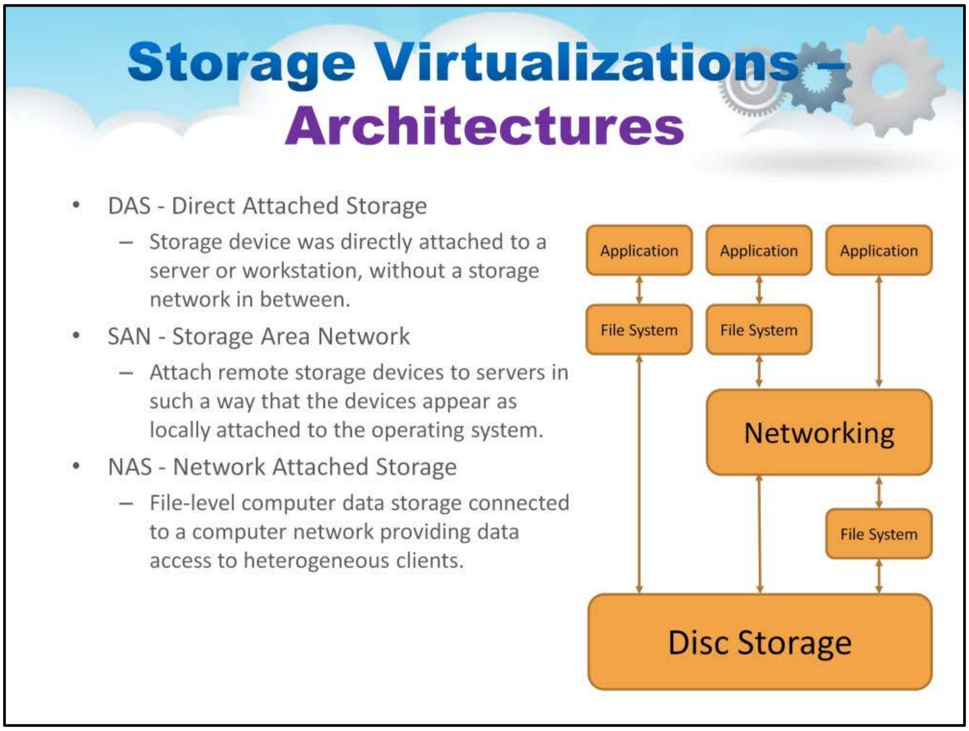
SCSI, SAS, ATA, SATA

HDD, Flash disk + Tape

VI - Virtualised Instance

As we have seen from previous lessons, virtualization of any kind of resource follows a common blueprint. The physical hardware is put under a special kind of software control which abstracts the physical layer, and presents to the user what “looks” like the actual resource but is actually a “virtual” instance of that resource.

Storage follows this blueprint. The hardware and the hardware interfaces are virtualized by a software layer as shown in the illustration, thus presenting virtual storage primitives (disks, file systems, etc) to the consumer.

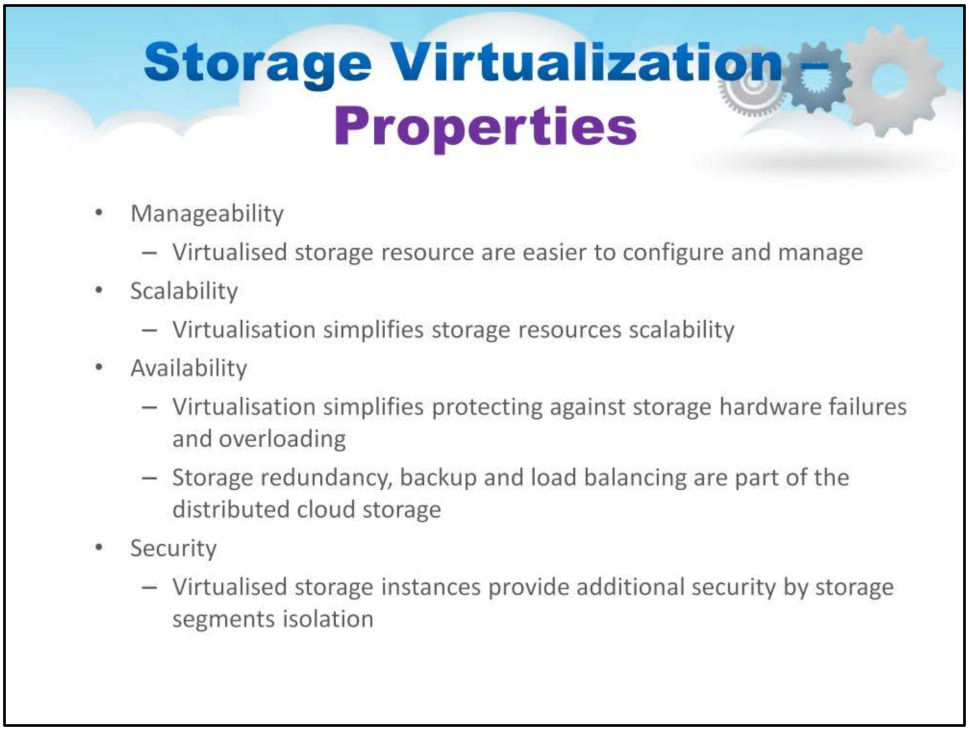


Storage Virtualizations – Architectures

- DAS - Direct Attached Storage
  - Storage device was directly attached to a server or workstation, without a storage network in between.
- SAN - Storage Area Network
  - Attach remote storage devices to servers in such a way that the devices appear as locally attached to the operating system.
- NAS - Network Attached Storage
  - File-level computer data storage connected to a computer network providing data access to heterogeneous clients.

In clouds, there are many options for implementing physical storage, because, the virtual storage interfaces can be kept to a small set, with the software providing common interfaces.

Once can see by the diagram, that the underlying architectures of these different schemes are quite different. The applications interface (legacy, non virtual) all look the same – basically an application accesses a file system. Under the hood, the connectivity of the components – and where the file system code actually runs – can be at any number of locations (as shown)



Storage Virtualization – Properties

- Manageability
  - Virtualised storage resource are easier to configure and manage
- Scalability
  - Virtualisation simplifies storage resources scalability
- Availability
  - Virtualisation simplifies protecting against storage hardware failures and overloading
  - Storage redundancy, backup and load balancing are part of the distributed cloud storage
- Security
  - Virtualised storage instances provide additional security by storage segments isolation

The software layer which is implementing the virtualized storage, can also enhance the storage model offered to beyond that which physical software can accomplish.

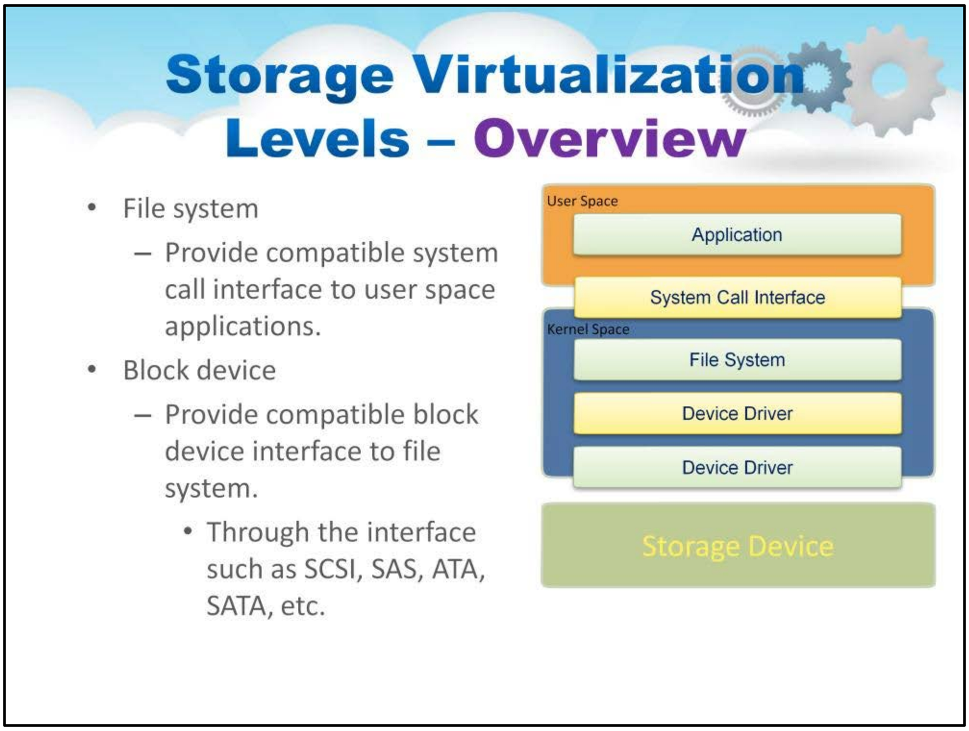
Storage virtualisation includes pooling multiple physical storage resources (in general heterogeneous), their abstraction and further logical composition or segmentation.

Storage Virtualization

–

Levels

Levels



Storage Virtualization Levels – Overview

- File system
  - Provide compatible system call interface to user space applications.
- Block device
  - Provide compatible block device interface to file system.
    - Through the interface such as SCSI, SAS, ATA, SATA, etc.

User Space

Application

System Call Interface

Kernel Space

File System

Device Driver

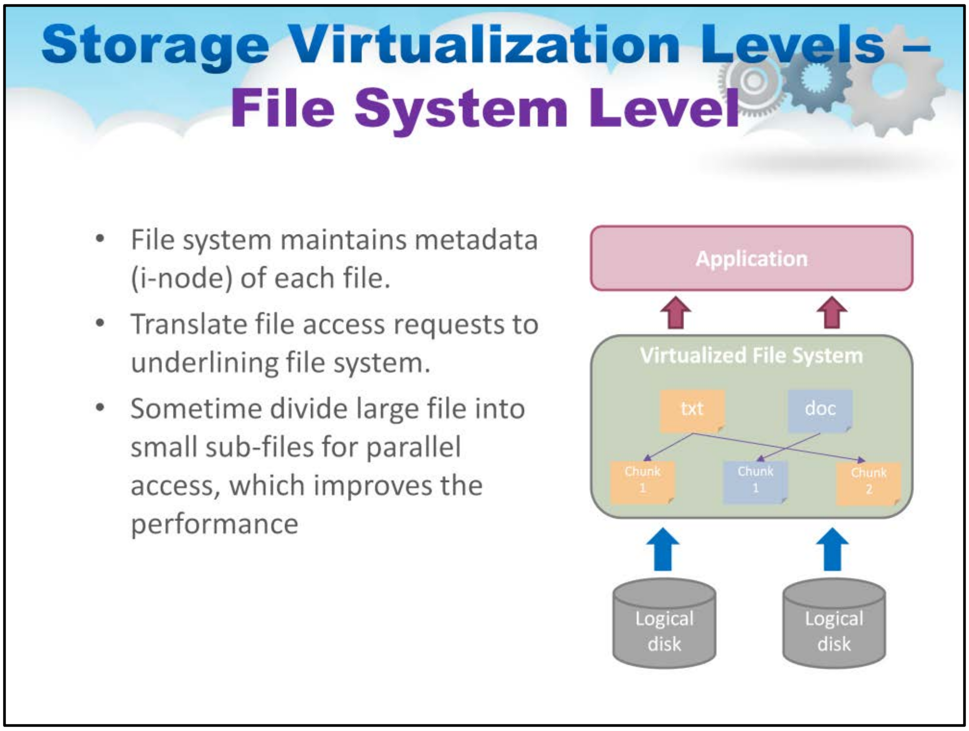
Device Driver

Storage Device

The illustration on this slide goes into more depth as to how the virtualization layers for storage work, and what kind of storage model they present.

The software layer for storage is in kernel space in the operating system, where it can intercept the disk and file system primitives and insert the capability of utilizing external, networked storage, and storage built from replicating, distributed drives.

The most common storage models are “file system” and “block device”. While the names imply the capability of the models, the Cloud OS may not provide the exact same capabilities as a standard lets say Linux file system or block device. We will discuss more on this later.



Storage Virtualization Levels – File System Level

- File system maintains metadata (i-node) of each file.
- Translate file access requests to underlining file system.
- Sometime divide large file into small sub-files for parallel access, which improves the performance

What is file system

- A file system is a software layer responsible for organizing and policing the creation, modification, and deletion of files
- File systems provide a hierarchical organization of files into directories and subdirectories
- The B-tree algorithm facilitates more rapid search and retrieval of files by name
- File system integrity is maintained through duplication of master tables, change logs, and immediate writes off file changes

Different file systems

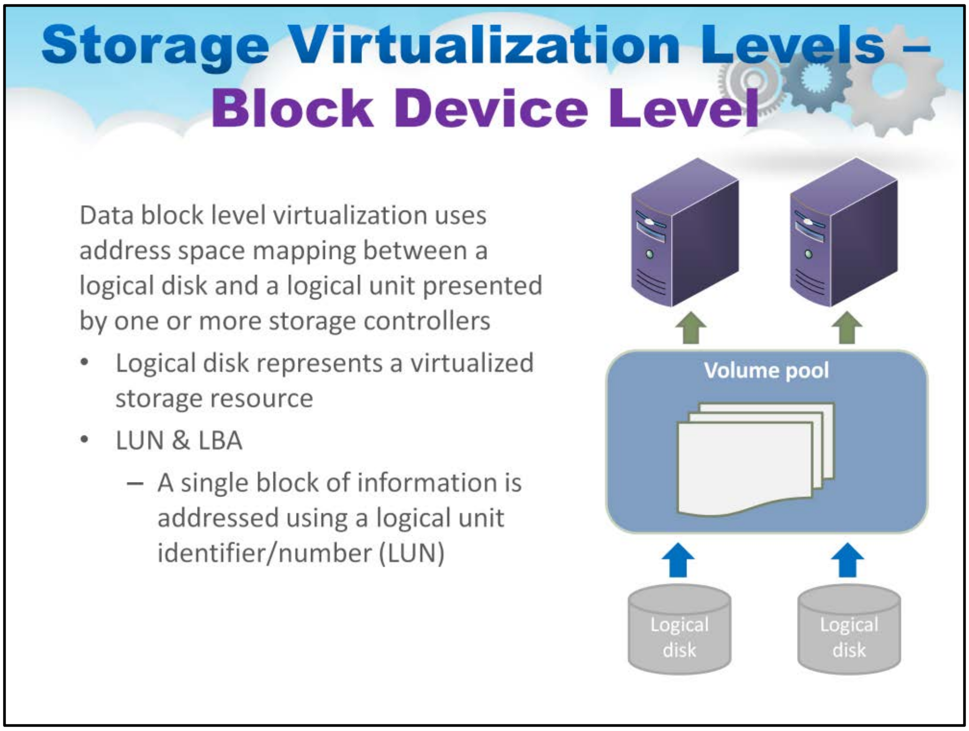
- In Unix, the super block contains information on the current state of the file system and its resources.
- In Windows NTFS, the master file table contains information on all file entries and status.

File Metadata

- The control information for file management is known as metadata.
- File metadata includes file attributes and pointers to the location of file data content.
- File metadata may be segregated from a file's data content.
- Metadata on file ownership and permissions is used in file access.
- File timestamp metadata facilitates automated processes such as backup and life cycle management.

Different file systems

- In Unix systems, file metadata is contained in the i-node structure.
- In Windows systems, file metadata is contained in records of file attributes.



Storage Virtualization Levels – Block Device Level

Data block level virtualization uses address space mapping between a logical disk and a logical unit presented by one or more storage controllers

- Logical disk represents a virtualized storage resource
- LUN & LBA
  - A single block of information is addressed using a logical unit identifier/number (LUN)

Volume pool

Logical disk

Logical disk

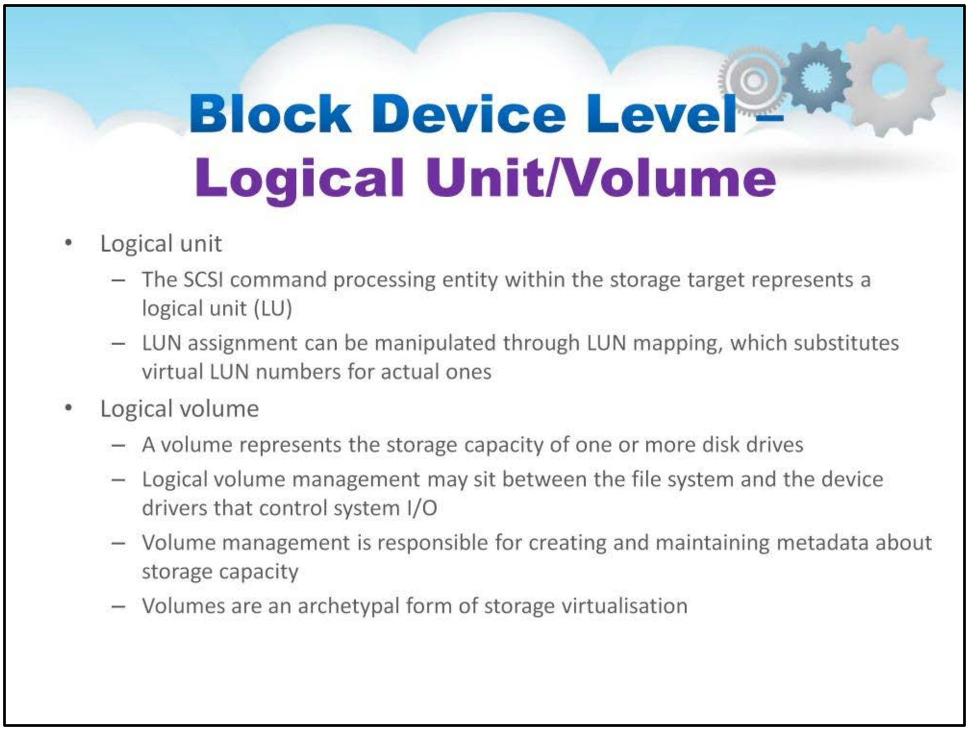
Block Device Level Virtualization is a low level technique which creates a “volume pool” from a collection of drives. It presents virtualized storage primitives called LUN

for Logical Unit Identifier, and an offset within that LUN, which

known as a Logical Block Address (LBA)

This is illustrated in the slide

A single block of information is addressed using a logical unit identifier/number (LUN) and an offset within that LUN, which known as a Logical Block Address (LBA)



Block Device Level – Logical Unit/Volume

- Logical unit
  - The SCSI command processing entity within the storage target represents a logical unit (LU)
  - LUN assignment can be manipulated through LUN mapping, which substitutes virtual LUN numbers for actual ones
- Logical volume
  - A volume represents the storage capacity of one or more disk drives
  - Logical volume management may sit between the file system and the device drivers that control system I/O
  - Volume management is responsible for creating and maintaining metadata about storage capacity
  - Volumes are an archetypal form of storage virtualisation

Block level data

The file system block

- The atomic unit of file system management is the file system block.
- A file's data may span multiple file system blocks.
- A file system block is composed of a consecutive range of disk block addresses.

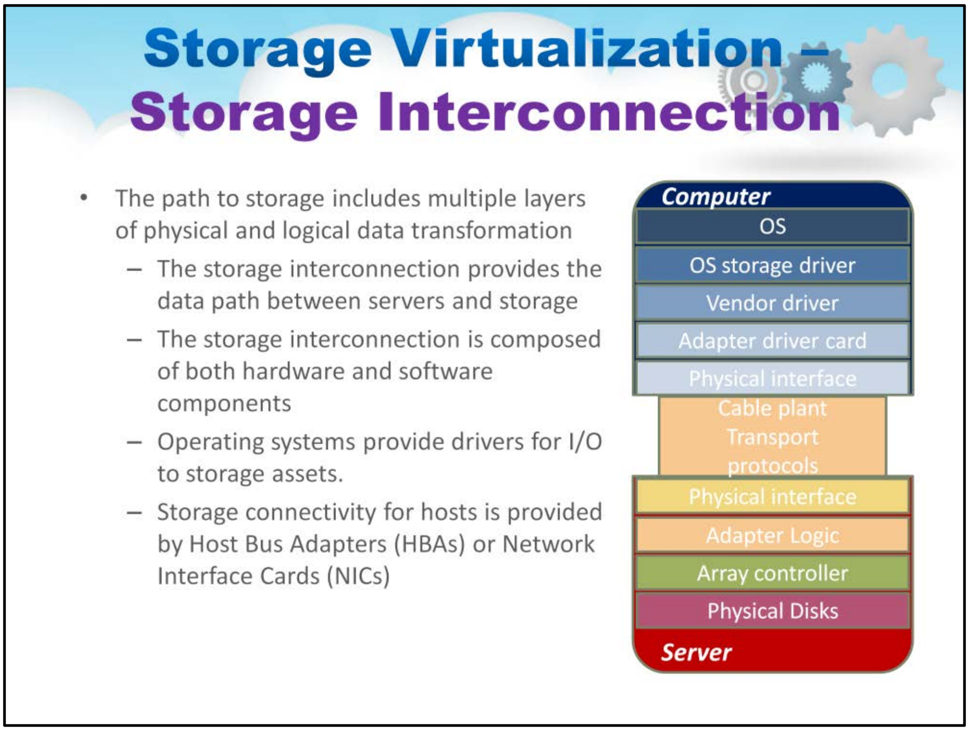
Data in disk

- Disk drives read and write data to media through cylinder, head, and sector geometry.
- Microcode on a disk translates between disk block numbers and cylinder/head/sector locations.
- This translation is an elementary form of virtualization.

Block device level interface: SCSI (*Small Computer System Interface*)

- The exchange of data blocks between the host system and storage is governed by the SCSI protocol.

The SCSI command processing entity within the storage target represents a logical unit (LU) and is assigned a logical unit number (LUN) for identification by the host platform



Storage Virtualization – Storage Interconnection

- The path to storage includes multiple layers of physical and logical data transformation
  - The storage interconnection provides the data path between servers and storage
  - The storage interconnection is composed of both hardware and software components
  - Operating systems provide drivers for I/O to storage assets.
  - Storage connectivity for hosts is provided by Host Bus Adapters (HBAs) or Network Interface Cards (NICs)

Computer

OS

OS storage driver

Vendor driver

Adapter driver card

Physical interface

Cable plant Transport protocols

Physical interface

Adapter Logic

Array controller

Physical Disks

Server

Drives are not always local to the server, and therefore a Storage Interconnection is utilized.

This illustration shows that the path to storage includes multiple layers of physical and logical data transformation

The storage interconnection provides the data path

between servers and storage

The storage interconnection is composed of both

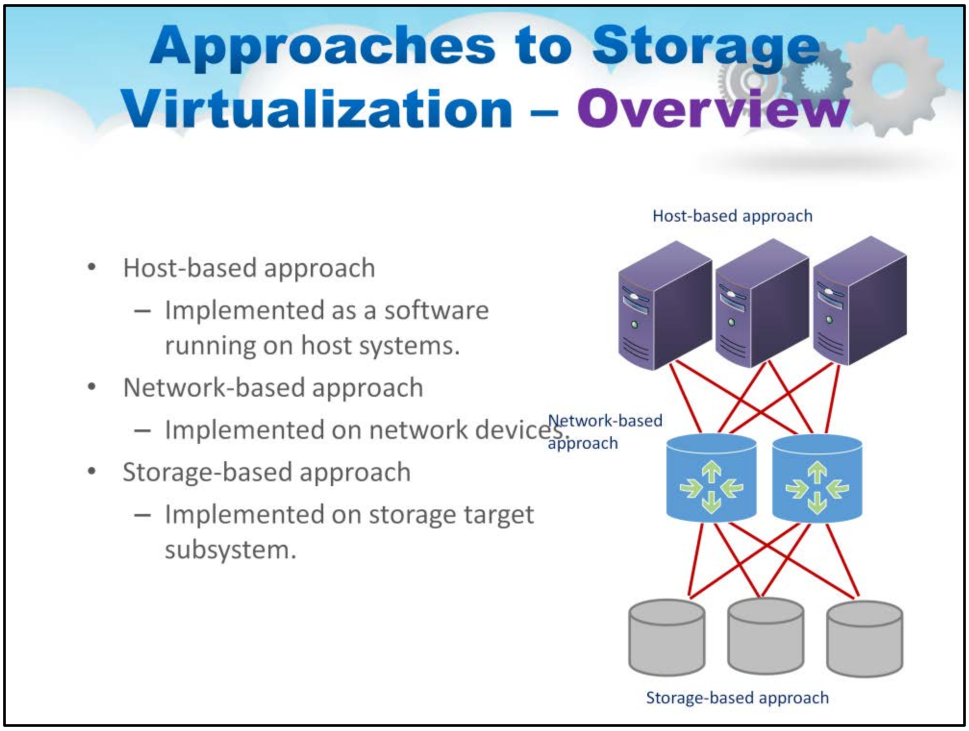
hardware and software components

Storage Virtualization

–

Approaches

Approaches



Approaches to Storage Virtualization – Overview

- Host-based approach
  - Implemented as a software running on host systems.
- Network-based approach
  - Implemented on network devices.
- Storage-based approach
  - Implemented on storage target subsystem.

Abstracting physical storage

Physical to virtual

The cylinder, head and sector geometry of individual disks is virtualized into logical block addresses (LBAs).

For storage networks, the physical storage system is identified by a network address / LUN pair.

Combining RAID and JBOD assets to create a virtualized mirror must accommodate performance differences.

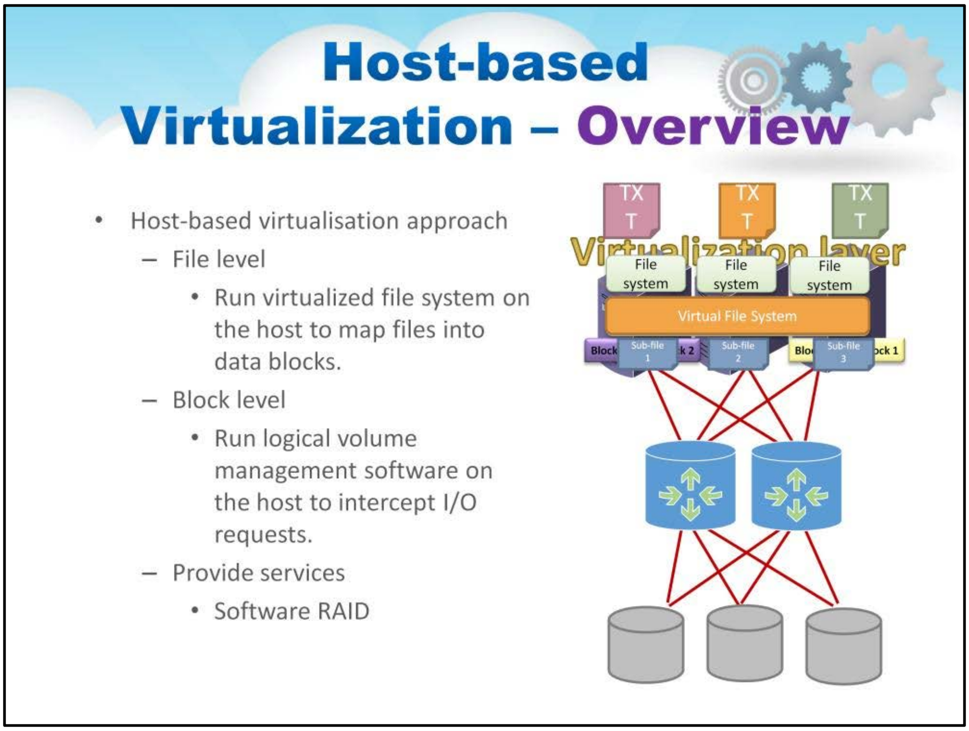
Metadata integrity

Storage metadata integrity requires redundancy for failover or load balancing.

Virtualization intelligence may need to interface with upper layer applications to ensure data consistency.

Storage Virtualization Approaches – Host-based

Host-based



Host-based Virtualization – Overview

- Host-based virtualisation approach
  - File level
    - Run virtualized file system on the host to map files into data blocks.
  - Block level
    - Run logical volume management software on the host to intercept I/O requests.
  - Provide services
    - Software RAID

Important issues

Storage metadata servers

Storage metadata may be shared by multiple servers.

Shared metadata enables a SAN file system view for multiple servers.

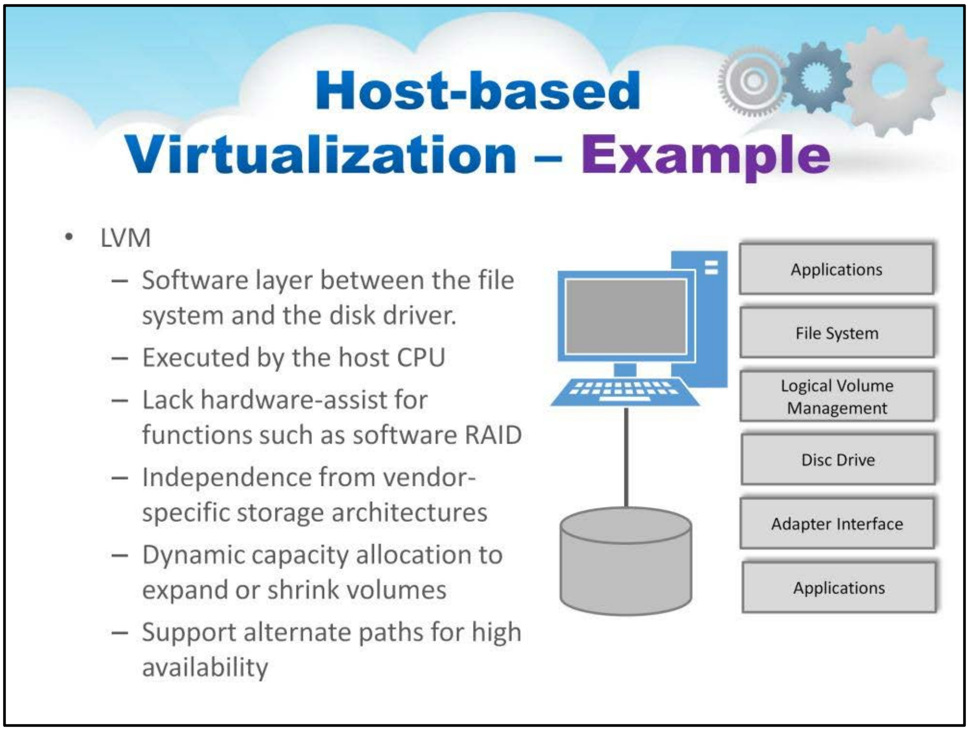
Provides virtual to real logical block address mapping for client.

A distributed SAN file system requires file locking mechanisms to preserve data integrity.

Host-based storage APIs

May be implemented by the operating system to provide a common interface to disparate virtualized resources.

Microsoft's virtual disk service (VDS) provides a management interface for dynamic generation of virtualized storage.



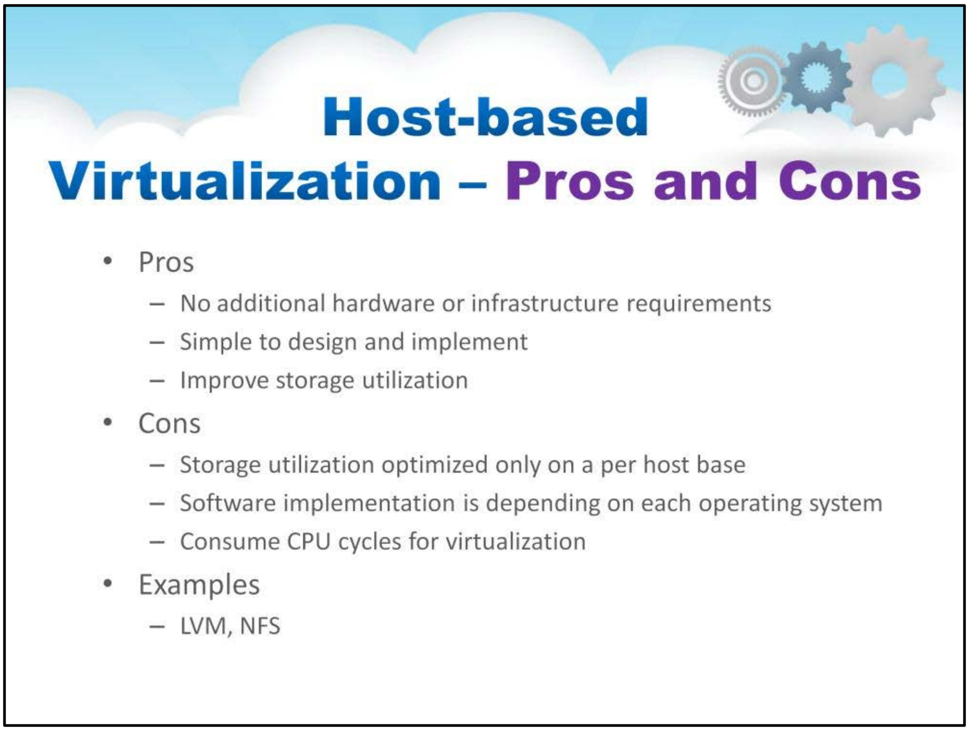
Host-based Virtualization – Example

- LVM
  - Software layer between the file system and the disk driver.
  - Executed by the host CPU
  - Lack hardware-assist for functions such as software RAID
  - Independence from vendor-specific storage architectures
  - Dynamic capacity allocation to expand or shrink volumes
  - Support alternate paths for high availability

Applications
File System
Logical Volume Management
Disc Drive
Adapter Interface
Applications

An additional layer of abstraction and control can be run on each host, and is called Logical Volume Manager (LVM). This code runs on the host and front-ends all kinds of back end storage resources.

The use cases and the architecture are shown on the slide.



Host-based Virtualization – Pros and Cons

- Pros
  - No additional hardware or infrastructure requirements
  - Simple to design and implement
  - Improve storage utilization
- Cons
  - Storage utilization optimized only on a per host base
  - Software implementation is depending on each operating system
  - Consume CPU cycles for virtualization
- Examples
  - LVM, NFS

Host based storage virtualization has gotten very popular in server machines because

No additional hardware or infrastructure

requirements

Simple to design and implement

Improve storage utilization

However

Storage utilization optimized only on a per host

base

Software implementation is depending on each

operating system

Consume CPU cycles for virtualization

As we all know, NFS is very popular, and it is a form of Host based storage virtualization

File level

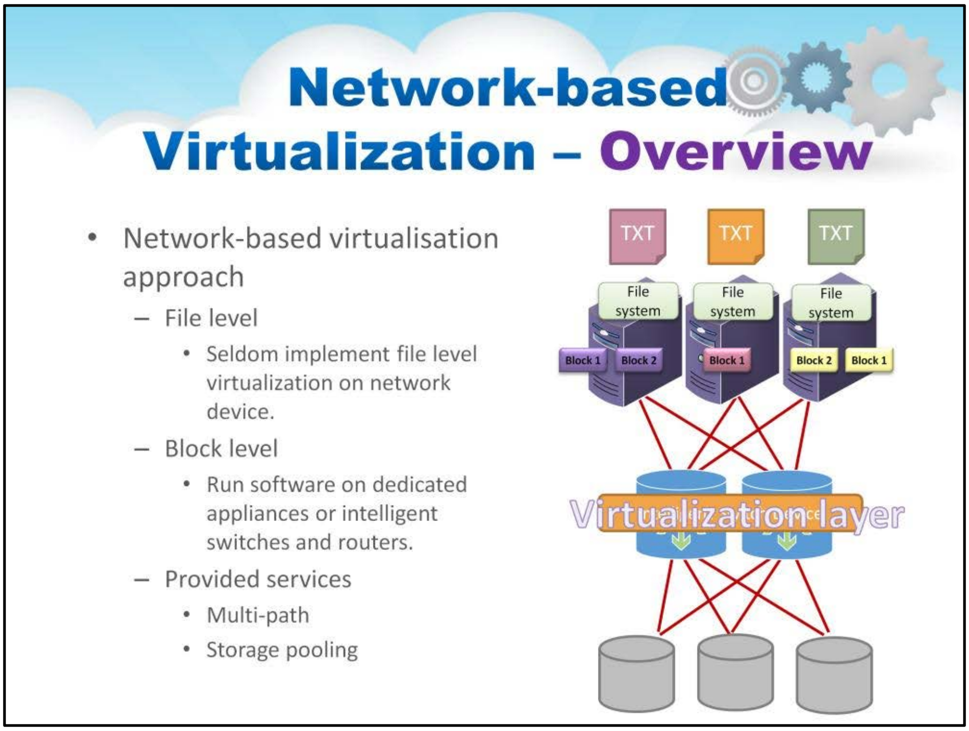
Run virtualized file system on the host to map files into data blocks, which distributed among several storage devices.

Block level

Run logical volume management software on the host to intercept I/O requests and redirect them to storage devices.

Storage Virtualization Approaches – Network-based

Network-based



Network-based Virtualization – Overview

- Network-based virtualisation approach
  - File level
    - Seldom implement file level virtualization on network device.
  - Block level
    - Run software on dedicated appliances or intelligent switches and routers.
  - Provided services
    - Multi-path
    - Storage pooling

Animation illustrates…

Fabric switch should provide

Connectivity for all storage transactions
Interoperability between disparate servers, operating systems, and target devices

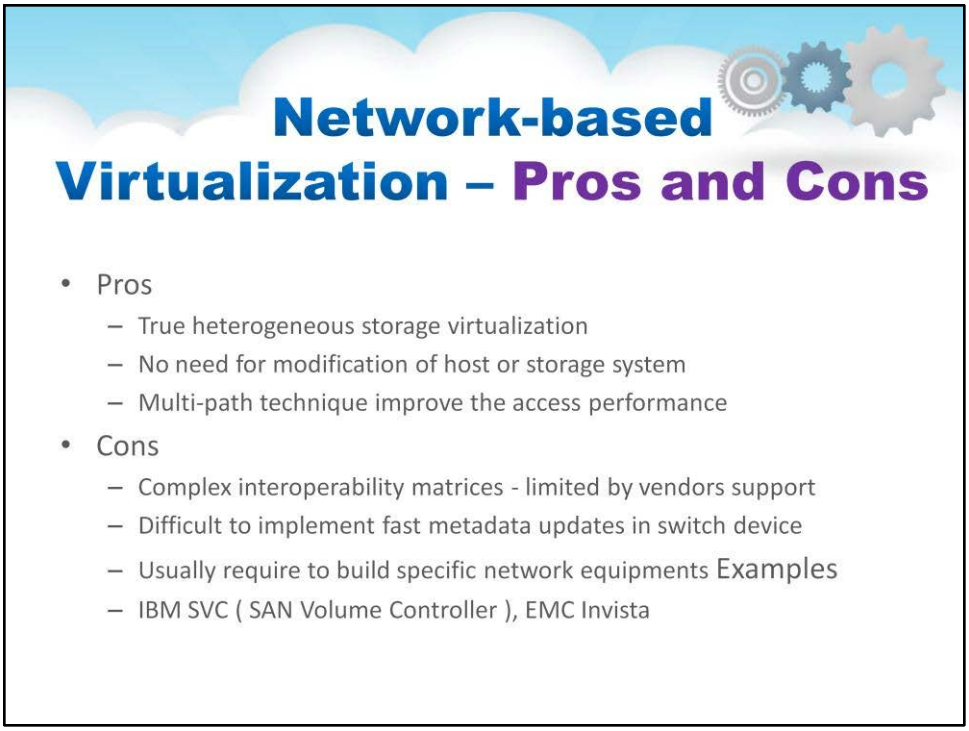
FAIS (Fabric Application Interface Standard)

Define a set of standard APIs to integrate applications and switches.

FAIS separates control information and data paths.

The control path processor (CPP) supports the FAIS APIs and upper layer storage virtualization application.

The data path controller (DPC) executes the virtualized SCSI I/Os under the management of one or more CPPs



Network-based Virtualization – Pros and Cons

- Pros
  - True heterogeneous storage virtualization
  - No need for modification of host or storage system
  - Multi-path technique improve the access performance
- Cons
  - Complex interoperability matrices - limited by vendors support
  - Difficult to implement fast metadata updates in switch device
  - Usually require to build specific network equipments Examples
  - IBM SVC (SAN Volume Controller), EMC Invista

Network-based Virtualization come with plus and minus factors as well

True heterogeneous storage virtualization

No need for modification of host or storage system

Multi-path technique improve the access performance

However

Complex interoperability matrices - limited by vendors support

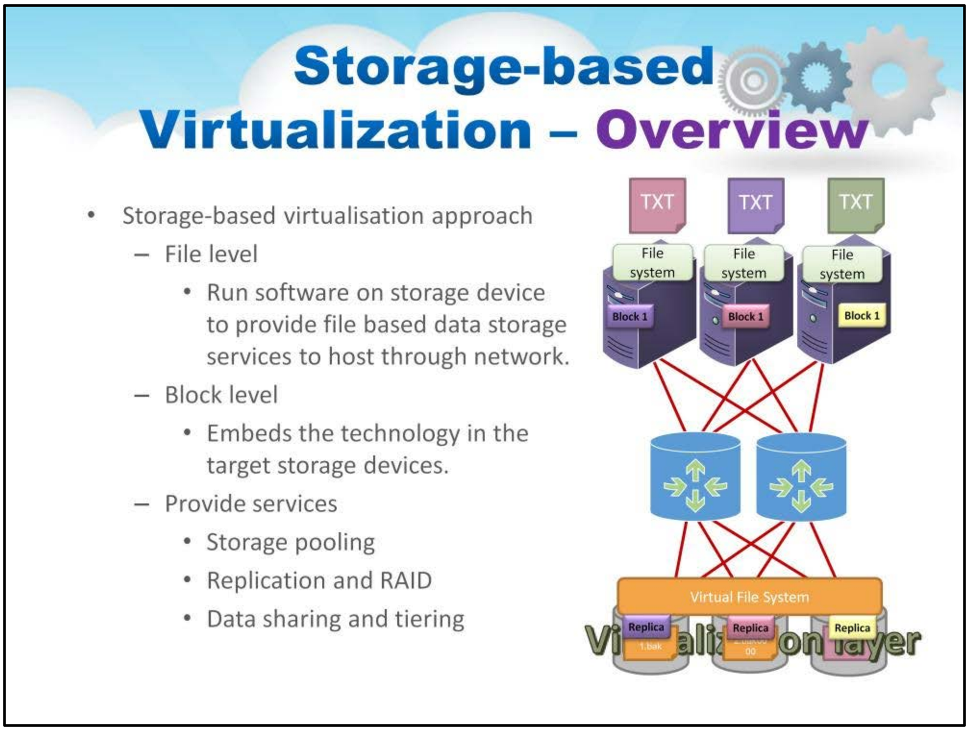
Difficult to implement fast metadata updates in switch device

Usually require to build specific network equipments (e.g., Fibre Channel)

IBM SVC (SAN Volume Controller) is an example

Storage Virtualization Approaches – Storage-based

Storage-based



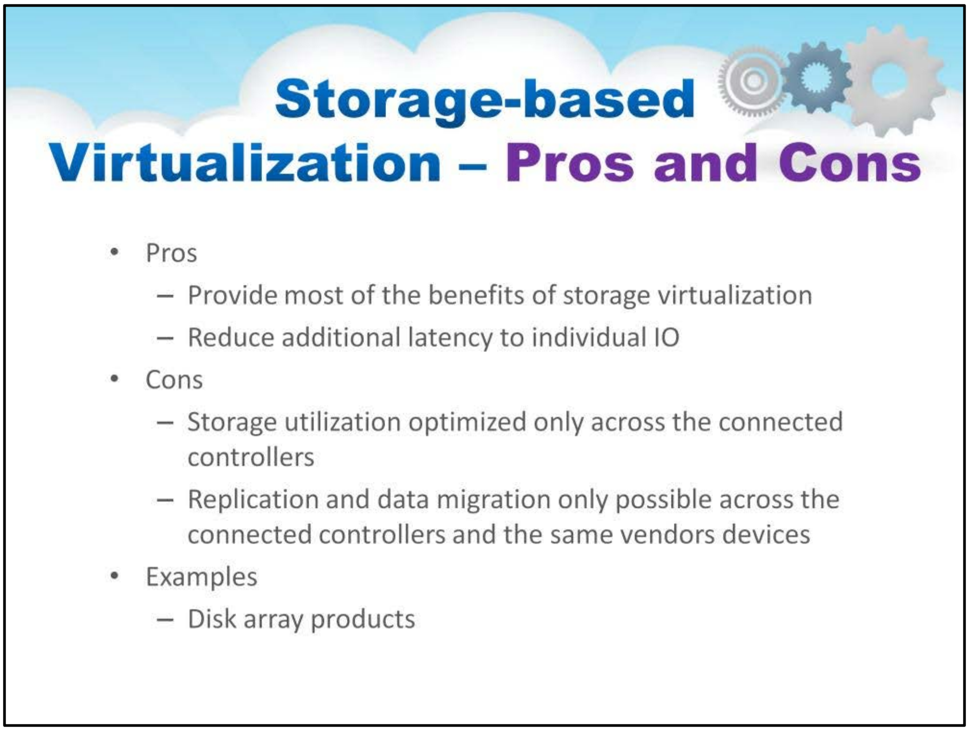
Storage-based Virtualization – Overview

- Storage-based virtualisation approach
  - File level
    - Run software on storage device to provide file based data storage services to host through network.
  - Block level
    - Embeds the technology in the target storage devices.
  - Provide services
    - Storage pooling
    - Replication and RAID
    - Data sharing and tiering

This animation illustrates how the different layers in a storage system perform their function.

In the first part of the animation one can see a mode where the underlying virtualized storage provides a virtual filesystem interface, The connected Operating Systems send the files there to get saved. The virtualized storage takes care of replicating the file across actual drives in the cloud for high durability.

In the second part of the animation, on can see the mode where the underlying virtualized storage presents and block level interface. Here, the application is running on an OS, which presents a local file system interface. The operating system deconstructs, through the file system code, the save into a series of blocks which need to be written. The blocks go to the virtualization layer in this case, which stores and replicates at the block level.



Storage-based Virtualization – Pros and Cons

- Pros
  - Provide most of the benefits of storage virtualization
  - Reduce additional latency to individual IO
- Cons
  - Storage utilization optimized only across the connected controllers
  - Replication and data migration only possible across the connected controllers and the same vendors devices
- Examples
  - Disk array products

Storage virtualization is extremely useful

On the one hand

Provide most of the benefits of storage virtualization
Reduce additional latency to individual IO

However

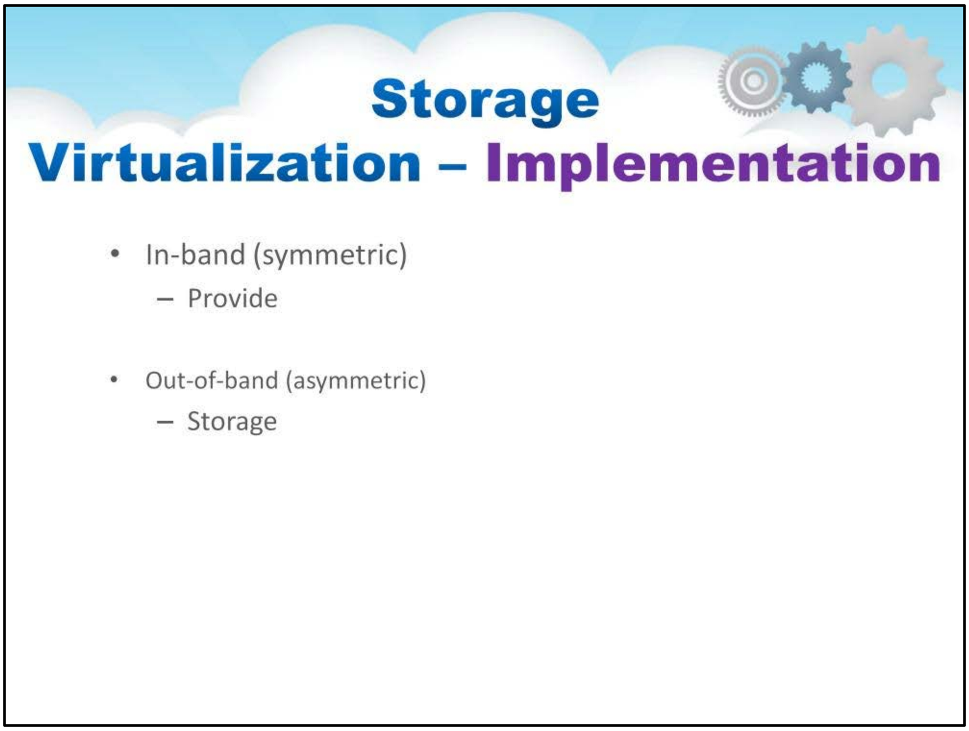
Storage utilization optimized only across the connected controllers
Replication and data migration only possible across the connected controllers and the same vendors devices

Storage Virtualization

–

Implementation Methods

Implementation Methods

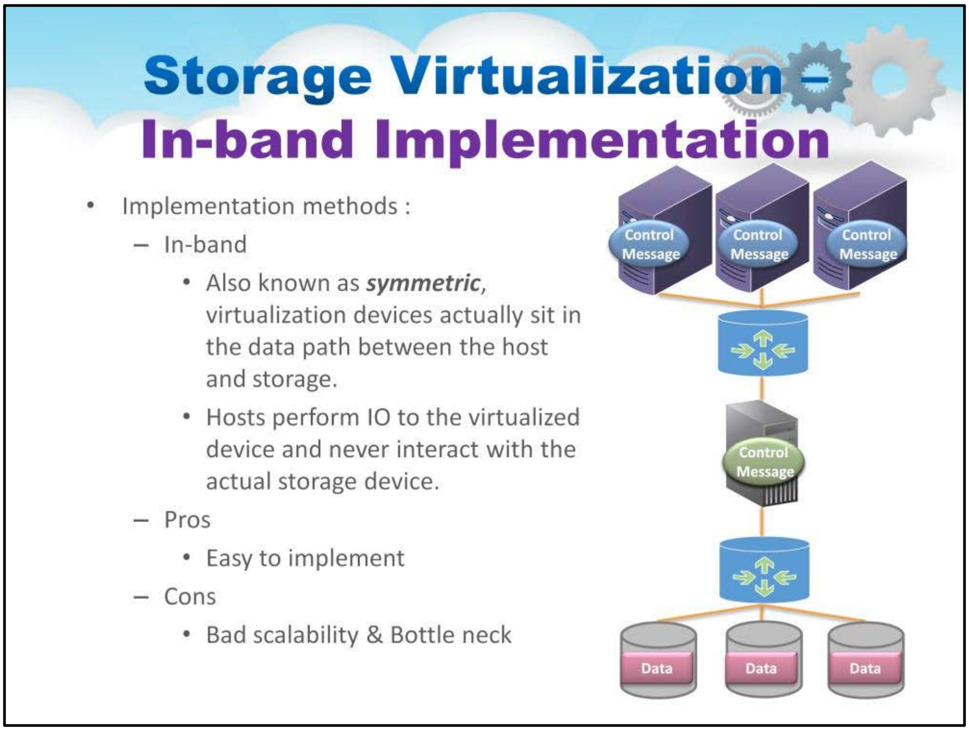


Storage Virtualization – Implementation

- In-band (symmetric)
  - Provide
- Out-of-band (asymmetric)
  - Storage

Storage virtualization can be implemented in a number of ways.

- In-Band virtualization - Also known as symmetric, virtualization devices actually sit in the data path between the host and storage.
- Out-of-Band virtualization - Also known as asymmetric, virtualization devices are sometimes called metadata servers.



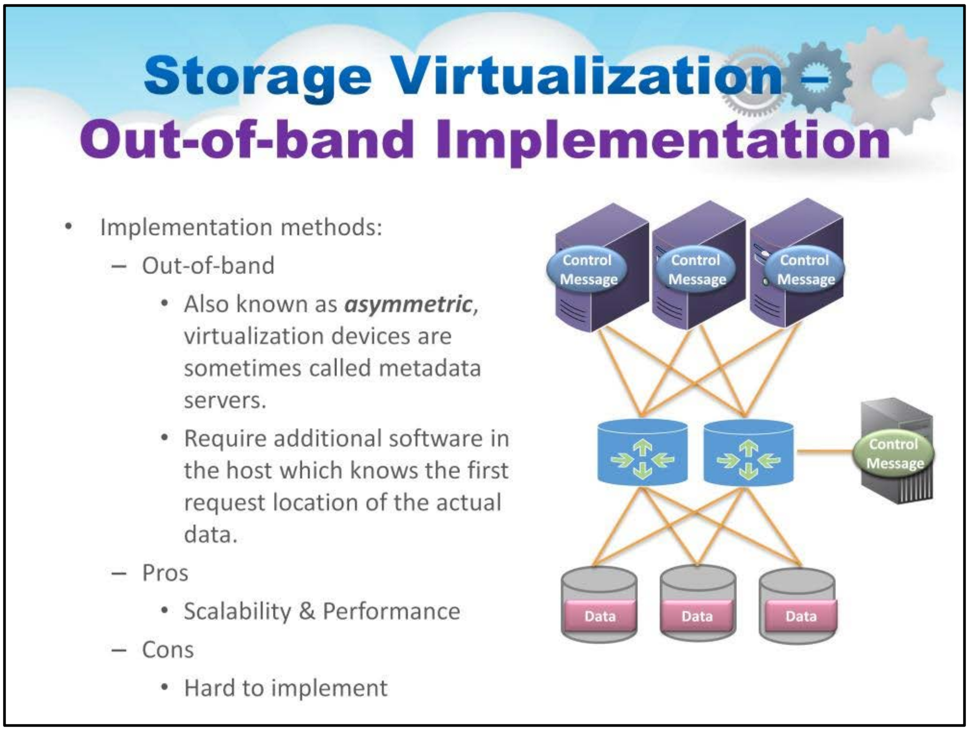
Storage Virtualization – In-band Implementation

- Implementation methods :
  - In-band
    - Also known as ***symmetric***, virtualization devices actually sit in the data path between the host and storage.
    - Hosts perform IO to the virtualized device and never interact with the actual storage device.
  - Pros
    - Easy to implement
  - Cons
    - Bad scalability & Bottle neck

The animation in this slide shows what is called In-Band virtualization, Also known as symmetric, virtualization devices actually sit in the data path between the host and storage.

Hosts perform IO to the virtualized device and never interact with the actual storage device.

While Easy to implement
It has Bad scalability & Bottle neck characteristics



Storage Virtualization – Out-of-band Implementation

- Implementation methods:
  - Out-of-band
    - Also known as ***asymmetric***, virtualization devices are sometimes called metadata servers.
    - Require additional software in the host which knows the first request location of the actual data.
  - Pros
    - Scalability & Performance
  - Cons
    - Hard to implement

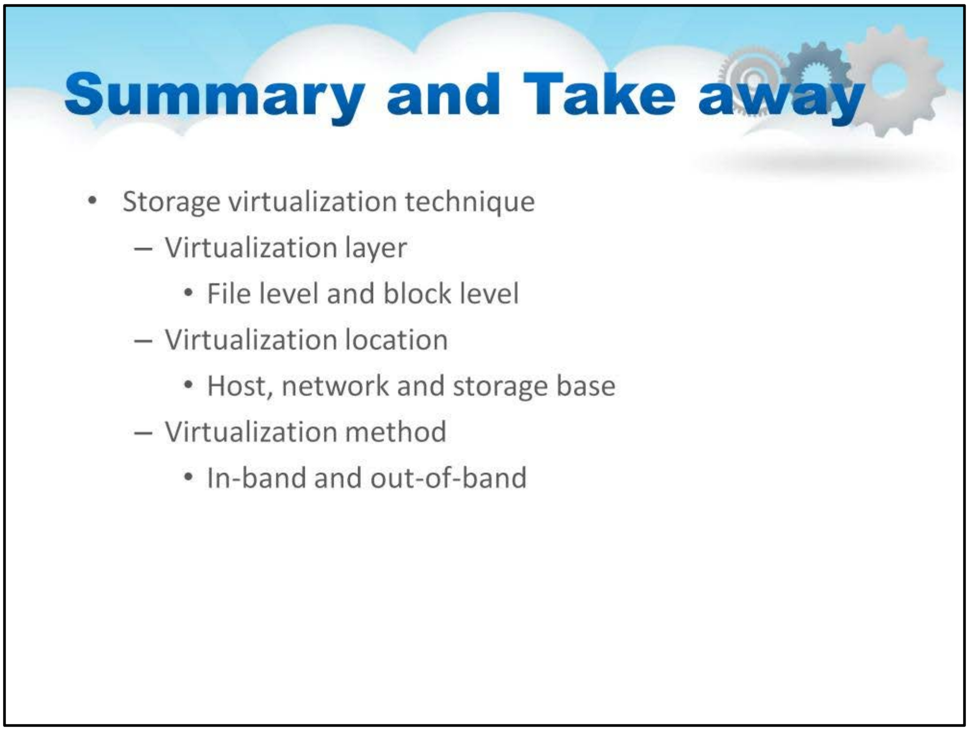
The animation in this illustration shows out of band virtualization

Also known as asymmetric, virtualization devices are sometimes called metadata servers.

It Requires additional software in the host which knows the first request location of the actual data.

While a good architecture for Scalability & Performance

It is Hard to implement



Summary and Take away

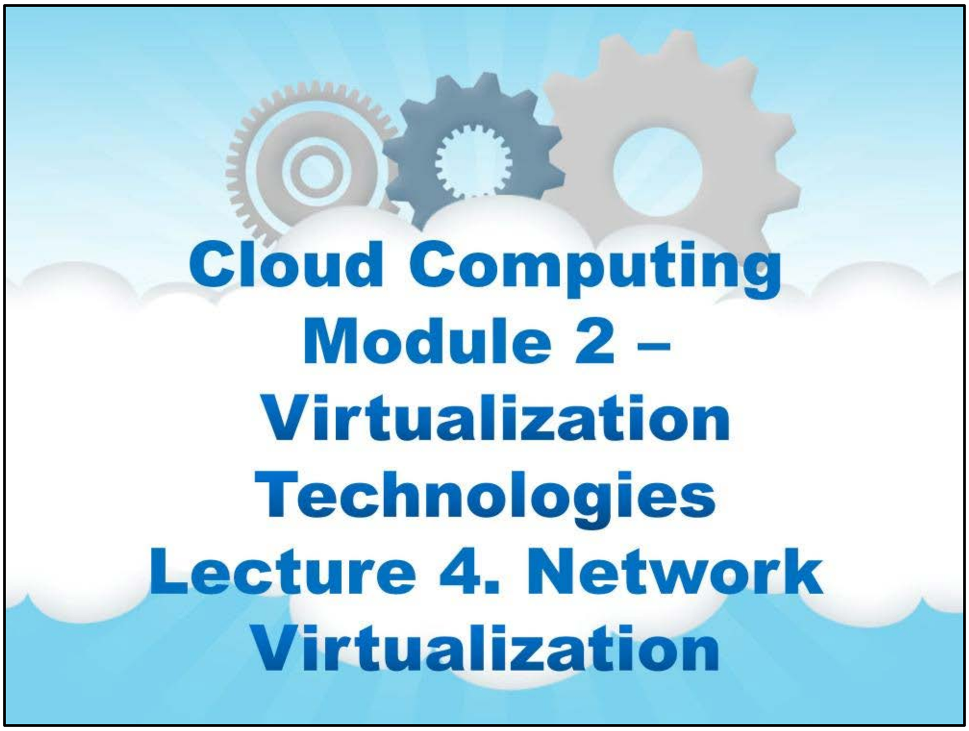
- Storage virtualization technique
  - Virtualization layer
    - File level and block level
  - Virtualization location
    - Host, network and storage base
  - Virtualization method
    - In-band and out-of-band

This tutorial has explored the various ways that Storage is virtualized and implemented for large scale, distributed systems, including Cloud.

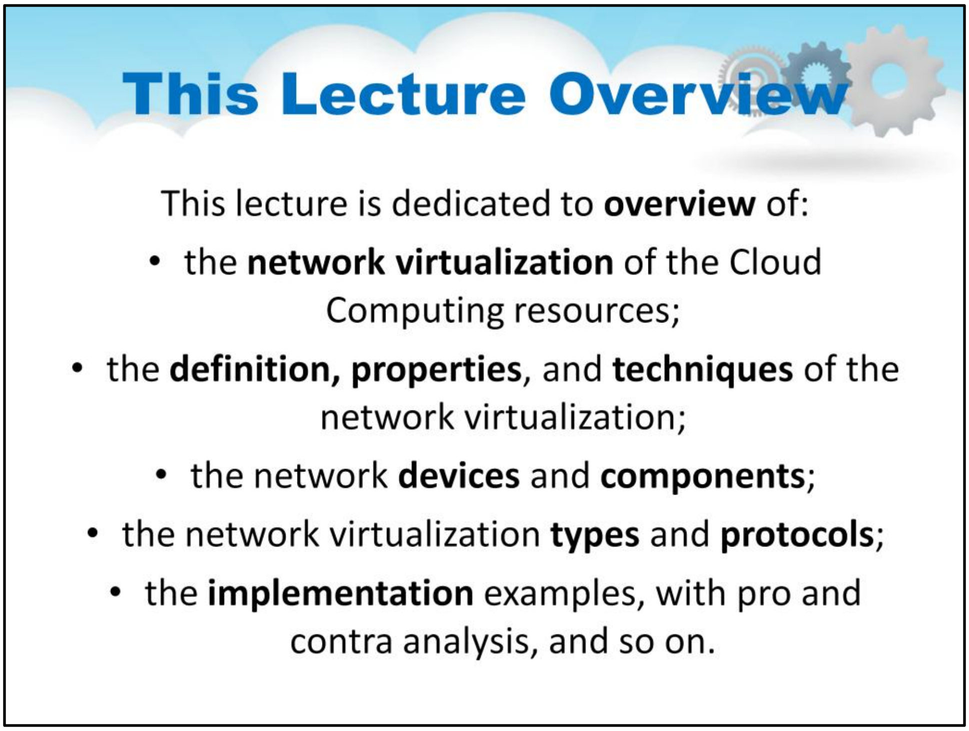
We explored the storage primitive which was virtualized, and saw that some systems concentrate on virtualizing files, and some systems concentrate on virtualizing blocks.

We saw that the virtualization function can run in a variety of places in the architecture. It can run in the host, in the network, or all the way back where the drives are.

We saw that virtualization can be placed “in band” of the storage operations, and for scale, is usually placed “out of band”.



Cloud Computing Module 2 – Virtualization Technologies Lecture 4. Network Virtualization

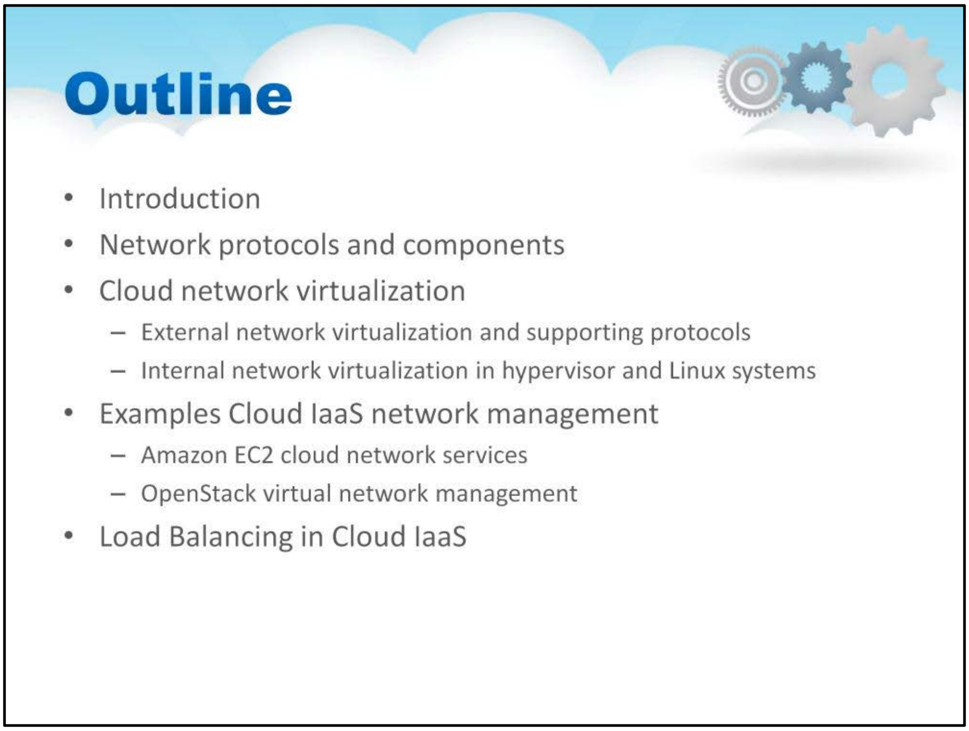


This Lecture Overview

This lecture is dedicated to **overview** of:

- the **network virtualization** of the Cloud Computing resources;
- the **definition, properties**, and **techniques** of the network virtualization;
- the network **devices** and **components**;
- the network virtualization **types** and **protocols**;
- the **implementation** examples, with pro and contra analysis, and so on.

Lecture 4. Network Virtualization



Outline

- Introduction
- Network protocols and components
- Cloud network virtualization
  - External network virtualization and supporting protocols
  - Internal network virtualization in hypervisor and Linux systems
- Examples Cloud IaaS network management
  - Amazon EC2 cloud network services
  - OpenStack virtual network management
- Load Balancing in Cloud IaaS

In this lesson we are going to dive deeper into Cloud Networking, in particular just as we have studied compute virtualization, and storage virtualization, we are going to study network virtualization, which is perhaps the most interesting and the most recent development of the three.

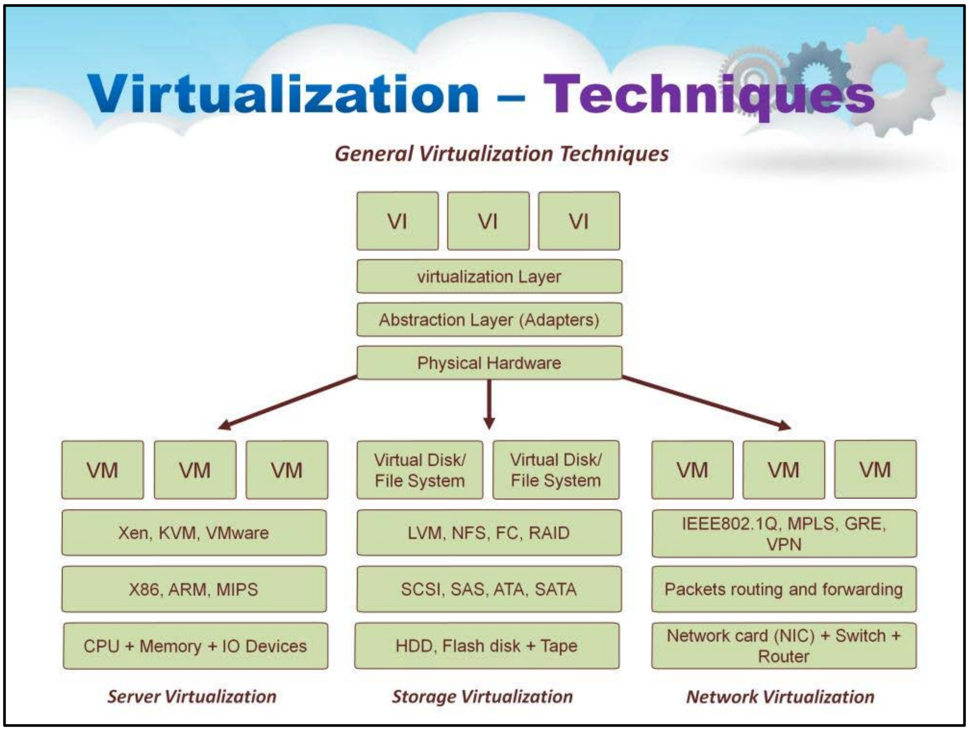
We will look at Network protocols and components and how they relate to the internal operating systems in the cloud.

We will study examples of network services in IaaS systems of AWS and OpenStack.

Finally we will cover the topic of load balancing, which is a special networking feature that is extremely useful in cloud computing.

Network Virtualization – Overview

Overview



Virtualization – Techniques

General Virtualization Techniques

VI VI VI

virtualization Layer

Abstraction Layer (Adapters)

Physical Hardware

VM VM VM

Xen, KVM, VMware

X86, ARM, MIPS

CPU + Memory + IO Devices

Server Virtualization

Virtual Disk/ File System

Virtual Disk/ File System

LVM, NFS, FC, RAID

SCSI, SAS, ATA, SATA

HDD, Flash disk + Tape

Storage Virtualization

VM VM VM

IEEE802.1Q, MPLS, GRE, VPN

Packets routing and forwarding

Network card (NIC) + Switch + Router

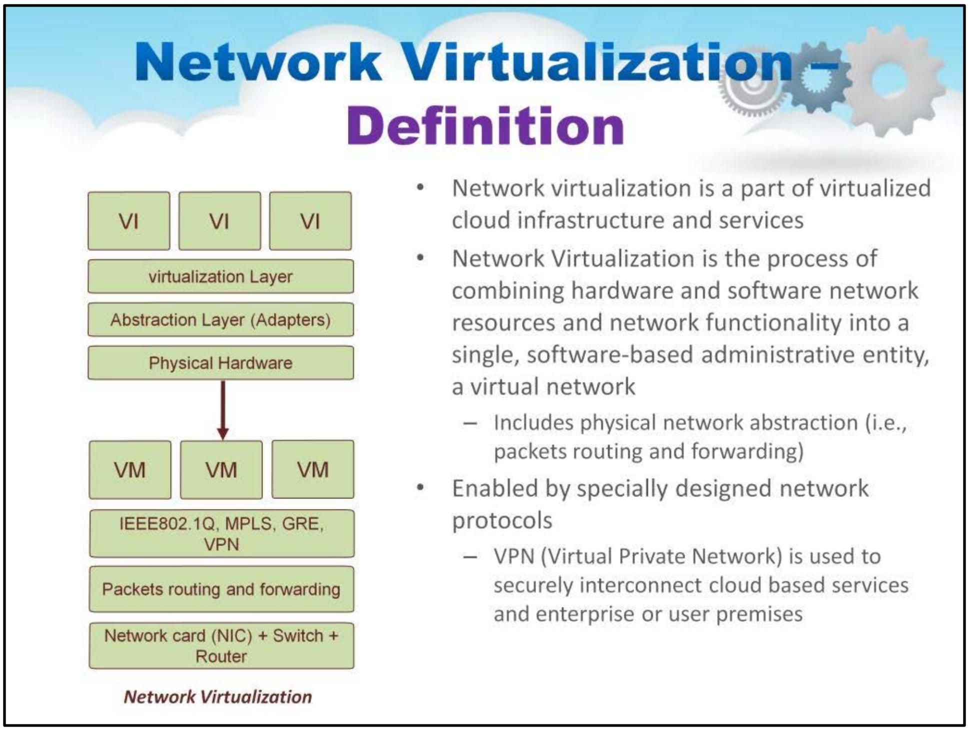
Network Virtualization

This diagram has been shown earlier to exemplify how the a generalized virtualization approach is applied to virtualization in specific areas.

In a previous lesson we covered in depth how server (or comoute) virtualization is typically enabled by a hypervisor as shown in the middle layer of the server virtualization part of the illustraton.

Also in a previous lesson we covered in depth how storage virtualization is enable. As shown in the middle layer of the storage virtualization part, this virtualization is typically implemented by filesystem software layers which can ru anywhere along the path to the drives, in other words, in the hosts, or across the network in processors where the drives are. The block devices themselves are also typically virtualized and merged using various techniques depending on how the drives are accessed.

Finally one can see from the illustration that there are analogous middle layers in the network virtualization case, which is the area we will now concentrate on.



Network Virtualization – Definition

VI
VI
VI
virtualization Layer
Abstraction Layer (Adapters)
Physical Hardware
VM
VM
VM
IEEE802.1Q, MPLS, GRE, VPN
Packets routing and forwarding
Network card (NIC) + Switch + Router

Network Virtualization

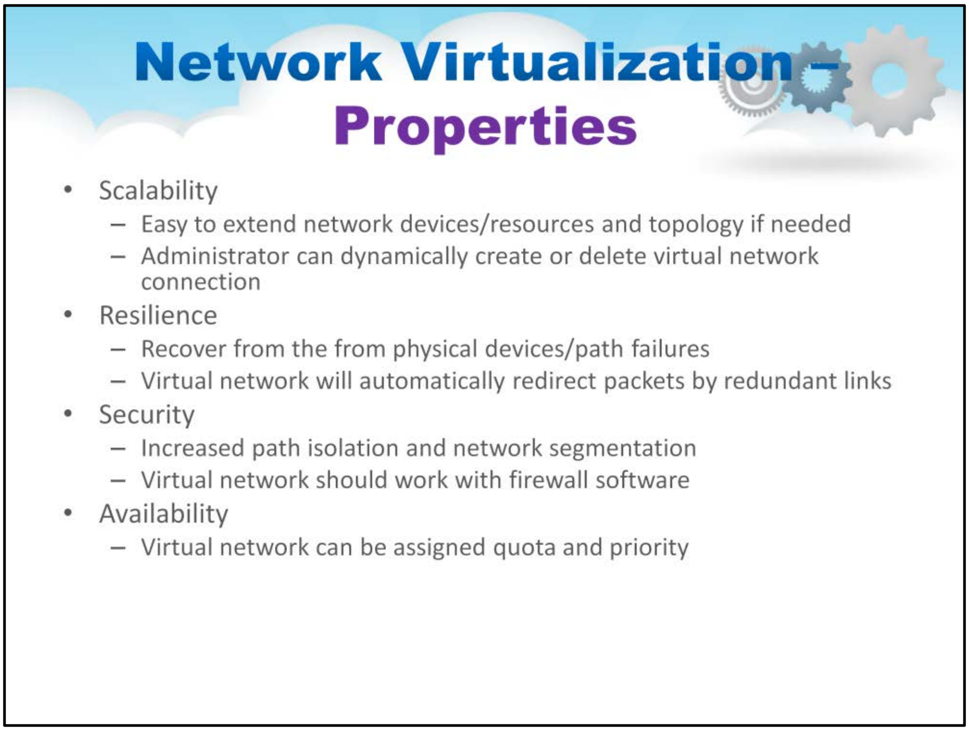
- Network virtualization is a part of virtualized cloud infrastructure and services
- Network Virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network
  - Includes physical network abstraction (i.e., packets routing and forwarding)
- Enabled by specially designed network protocols
  - VPN (Virtual Private Network) is used to securely interconnect cloud based services and enterprise or user premises

What is network virtualization? Network virtualization is a part of virtualized cloud infrastructure and services.

There is of course, always a physical network connecting all the physical parts of the cloud together. In fact, depending on how the cloud was actually constructed, there may be several networks connecting the physical parts of the cloud together, to provide for higher performance, better failure resiliency, or multiple physical paths. Across the cloud cluster, for example let’s say the physical cloud spans many, many racks, this/these network segments may be using Layer 2 in the rack and layer 3 from rack to rack; many techniques are possible. This physical layer is very important and requires traditional configuration of switches and routers.

Network virtualization creates a “logical” or virtual network on top of this physical network. Network virtualization uses the virtual switches that are present in the hypervisors to layer from VM to VM and they also layer on top of the physical network. Network Virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network

This virtual network is Enabled by specially designed network protocols, and by special software which controls or defines these virtual networks.,



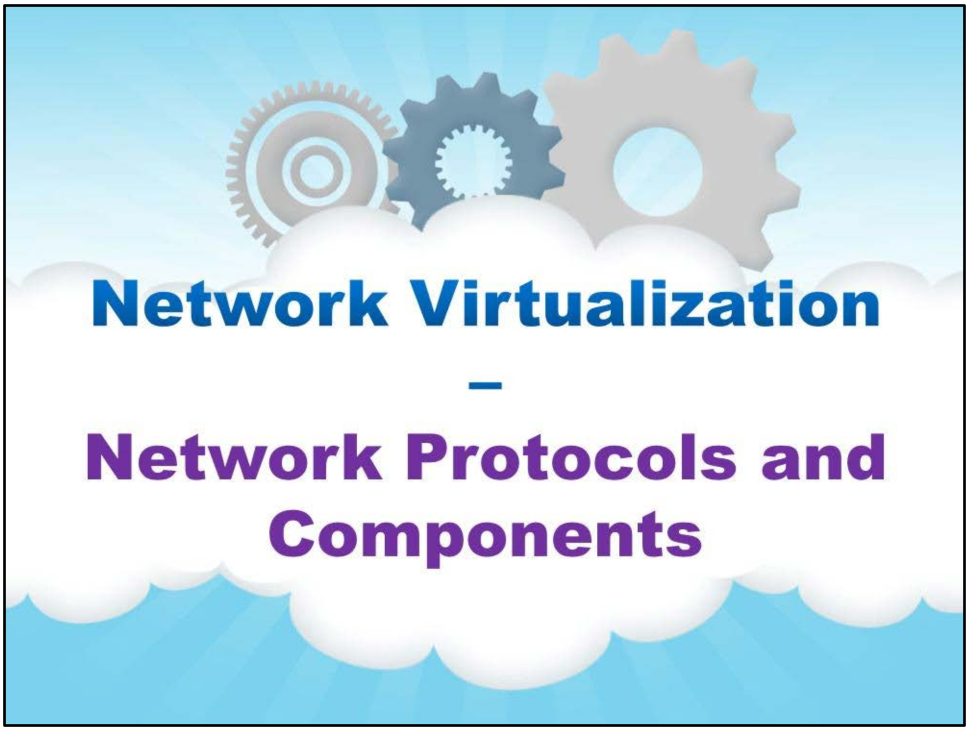
Network Virtualization – Properties

- Scalability
  - Easy to extend network devices/resources and topology if needed
  - Administrator can dynamically create or delete virtual network connection
- Resilience
  - Recover from the from physical devices/path failures
  - Virtual network will automatically redirect packets by redundant links
- Security
  - Increased path isolation and network segmentation
  - Virtual network should work with firewall software
- Availability
  - Virtual network can be assigned quota and priority

Now we will focus on this layer called the virtualized network. While we haven’t really explained how it works yet, conceptually we know it is software which sits on top of the physical network. It is distributed across the cloud and is made up of several software modules which talk to each other and also to the physical network below.

Given then that Network Virtualization is basically a distributed software system, if it is implemented using distributed computing principles such as being aware of state, using replication, asynchronous operation of components, and all the other usual guiding principles for building distributed systems, the Virtual Network will exhibit characteristics of a well designed distributed system.

As the slide indicates, Scalability, Resilience, Security, and Availability are all properties which the Virtual Network will exhibit by virtue of it’s distributed software based implementation.

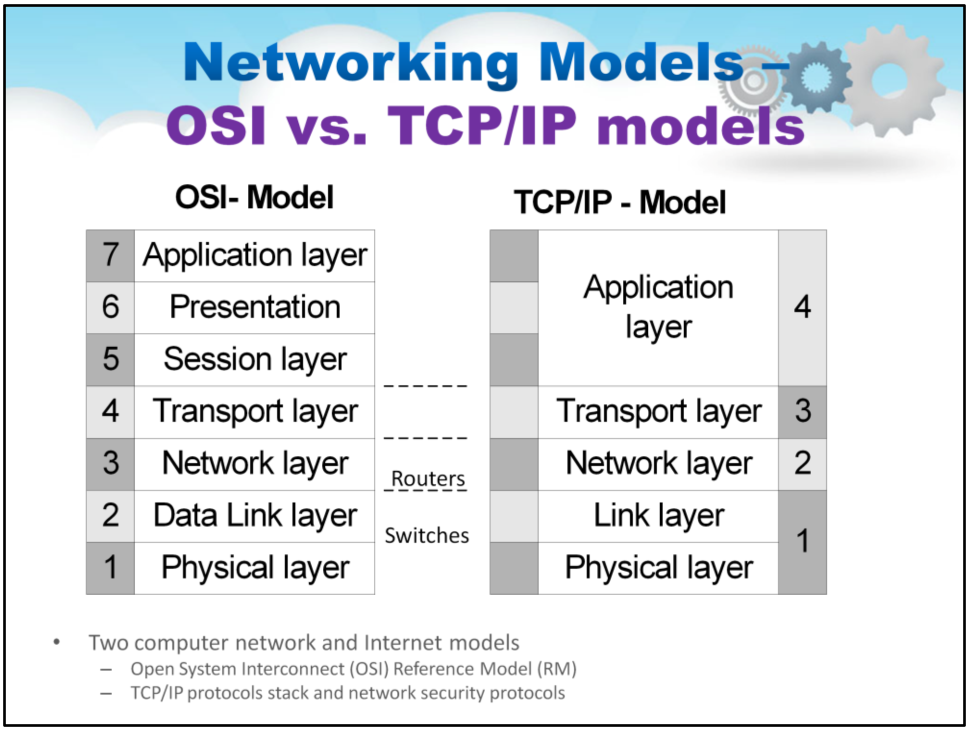


Network Virtualization

–

Network Protocols and Components

Network Protocols and Components



Networking Models – OSI vs. TCP/IP models

OSI- Model

7	Application layer
6	Presentation
5	Session layer
4	Transport layer
3	Network layer
2	Data Link layer
1	Physical layer

Routers

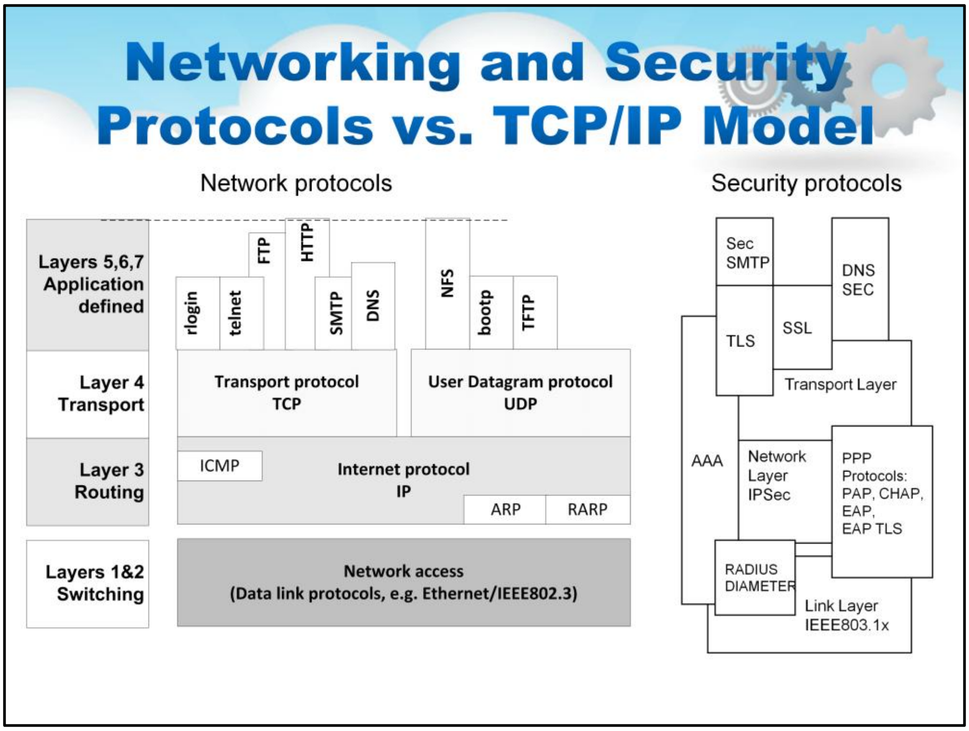
Switches

TCP/IP - Model

Application layer	4
Transport layer	3
Network layer	2
Link layer	1
Physical layer	

- Two computer network and Internet models
  - Open System Interconnect (OSI) Reference Model (RM)
  - TCP/IP protocols stack and network security protocols

As a reference, this slide provides an illustration to refresh yourself with the OSI networking model and the TCP/IP networking model. We will refer to the these layers on an ongoing basis



Networking and Security Protocols vs. TCP/IP Model

Network protocols

Layers 5,6,7 Application defined
rlogin
telnet
FTP
HTTP
SMTP
DNS
NFS
bootp
TFTP

Layer 4 Transport
Transport protocol TCP
User Datagram protocol UDP

Layer 3 Routing
ICMP
Internet protocol IP
ARP
RARP

Layers 1&2 Switching
Network access (Data link protocols, e.g. Ethernet/IEEE802.3)

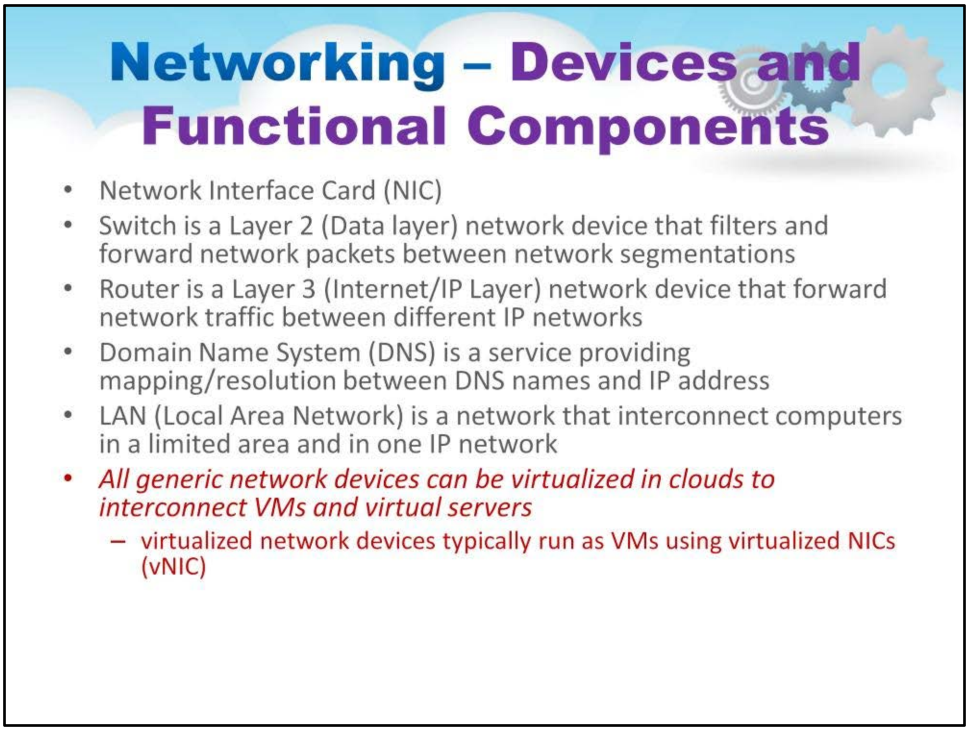
Security protocols

Sec SMTP
DNS SEC
TLS
SSL
Transport Layer
AAA
Network Layer IPSec
PPP Protocols: PAP, CHAP, EAP, EAP TLS
RADIUS DIAMETER
Link Layer IEEE803.1x

Also as a reference, this slide provides two illustrations.

The first illustration shows common IP based networking protocols and how they correspond to layers in the TCP/IP model.

The second illustration shows common security protocols and how they too correspond to layers in the TCP/IP model.



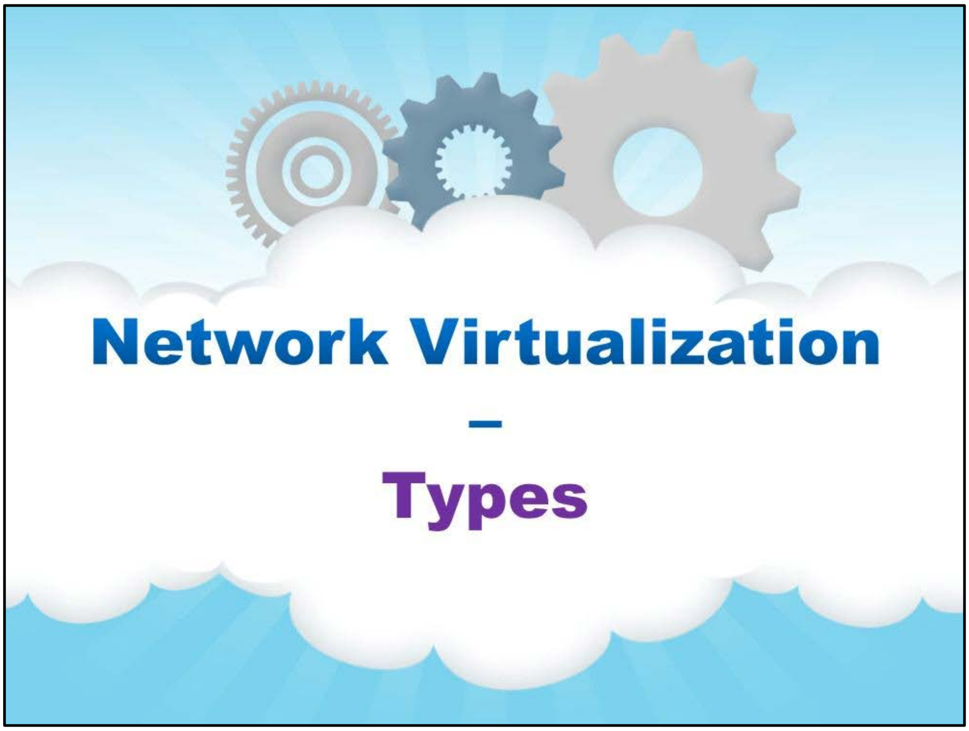
Networking – Devices and Functional Components

- Network Interface Card (NIC)
- Switch is a Layer 2 (Data layer) network device that filters and forward network packets between network segmentations
- Router is a Layer 3 (Internet/IP Layer) network device that forward network traffic between different IP networks
- Domain Name System (DNS) is a service providing mapping/resolution between DNS names and IP address
- LAN (Local Area Network) is a network that interconnect computers in a limited area and in one IP network
- *All generic network devices can be virtualized in clouds to interconnect VMs and virtual servers*
  - virtualized network devices typically run as VMs using virtualized NICs (vNIC)

As discussed earlier, there are physical network components, which are connected and configured. These are all sorts of networking devices, from Network Interface Cards (NIC) in the Servers, to Switches, to Routers, and also the infrastructure which the network depends on, such as a Domain Name System (DNS), and maybe an IP Address Management (IPAM) system (such as DHCP).

On top of this is a software layer which implements the Virtual Network. The virtual network includes the Virtual Switches and Virtual NICs (vNICs) which are part of the Hypervisor system. It also will contain virtual switches (vSwitch), virtual routers (vRouters), and even modules which supply DNA and IPAM at the virtualized network level.

In other words all generic network devices can be in a virtualized forms and used in clouds to interconnect VMs and virtual servers.

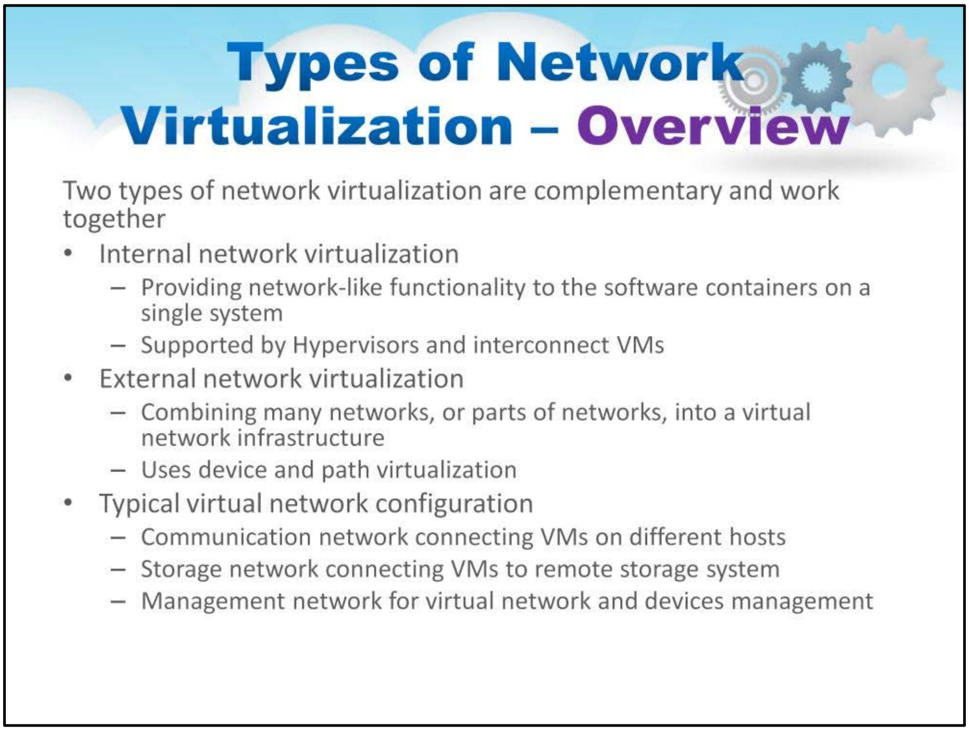


Network Virtualization

–

Types

Types



Types of Network Virtualization – Overview

Two types of network virtualization are complementary and work together

- Internal network virtualization
  - Providing network-like functionality to the software containers on a single system
  - Supported by Hypervisors and interconnect VMs
- External network virtualization
  - Combining many networks, or parts of networks, into a virtual network infrastructure
  - Uses device and path virtualization
- Typical virtual network configuration
  - Communication network connecting VMs on different hosts
  - Storage network connecting VMs to remote storage system
  - Management network for virtual network and devices management

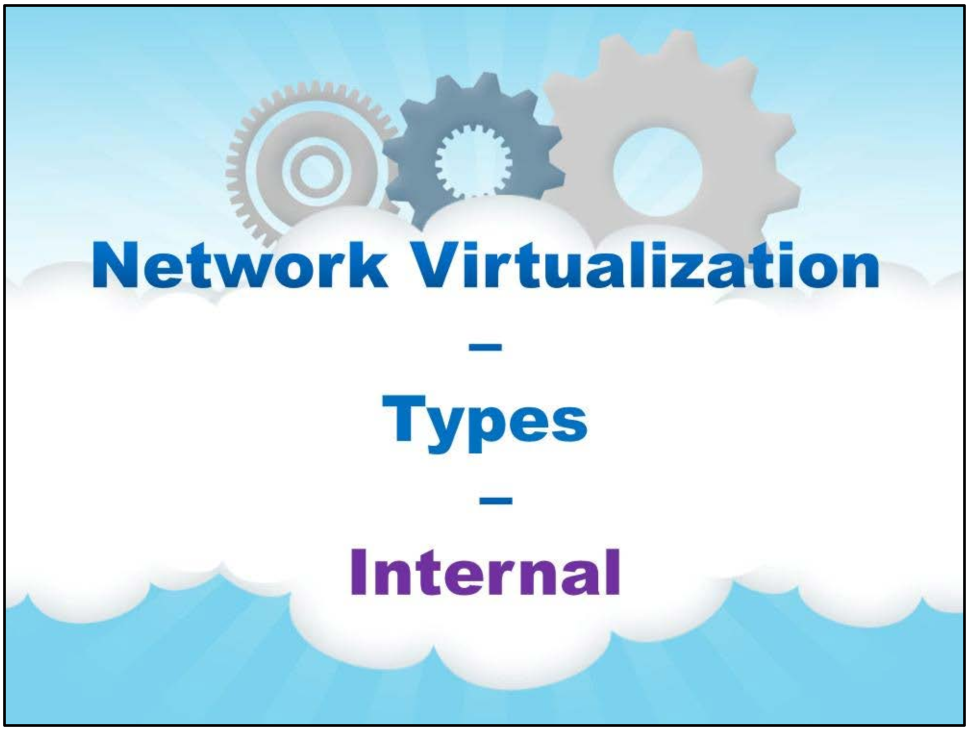
In the previous slide, we made a distinction between the Physical network and the Virtual Network.

Actually the virtual network has two types within it, as you might have already figured out.

The first type is called Internal network virtualization, it is the vNICs and vSwitches Supported by the Hypervisors and interconnect VMs. It is Providing network-like functionality to the software containers on a single system

The second type is called External network virtualization, it is where we go outside the individual server boundary, and Combine many networks, or parts of networks, into a virtual network infrastructure. This Uses device and path virtualization.

We will think about a Typical virtual network configuration as the slide details.



Network Virtualization

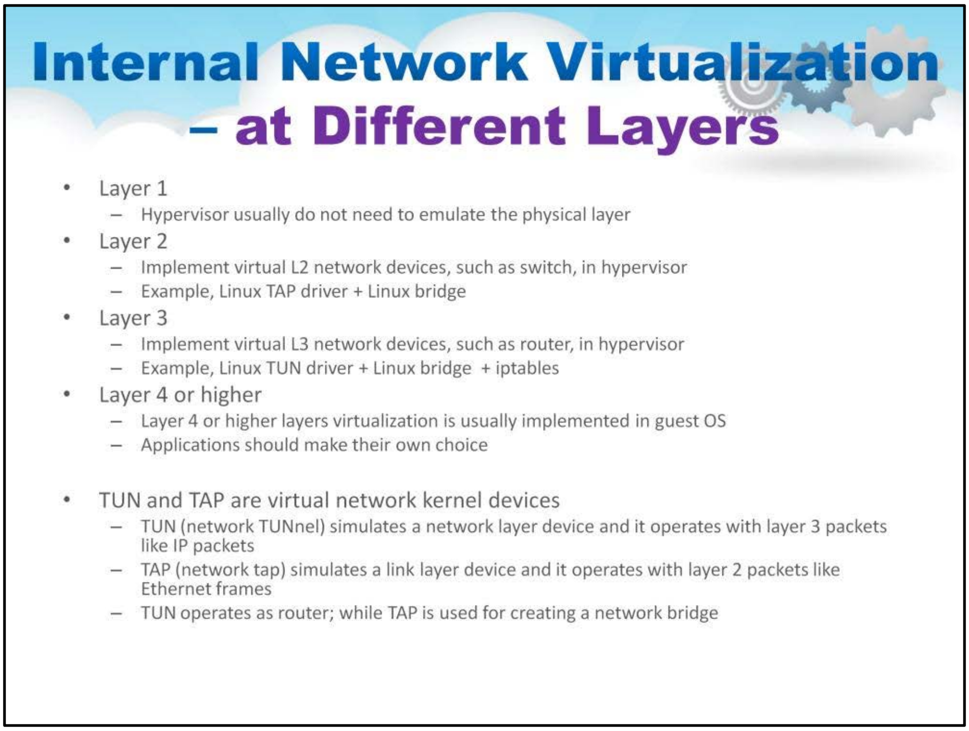
–

Types

–

Internal

Internal



Internal Network Virtualization – at Different Layers

- Layer 1
  - Hypervisor usually do not need to emulate the physical layer
- Layer 2
  - Implement virtual L2 network devices, such as switch, in hypervisor
  - Example, Linux TAP driver + Linux bridge
- Layer 3
  - Implement virtual L3 network devices, such as router, in hypervisor
  - Example, Linux TUN driver + Linux bridge + iptables
- Layer 4 or higher
  - Layer 4 or higher layers virtualization is usually implemented in guest OS
  - Applications should make their own choice

- TUN and TAP are virtual network kernel devices
  - TUN (network TUNnel) simulates a network layer device and it operates with layer 3 packets like IP packets
  - TAP (network tap) simulates a link layer device and it operates with layer 2 packets like Ethernet frames
  - TUN operates as router; while TAP is used for creating a network bridge

Let us now examine the Internal Network Virtualization at Different Layers.

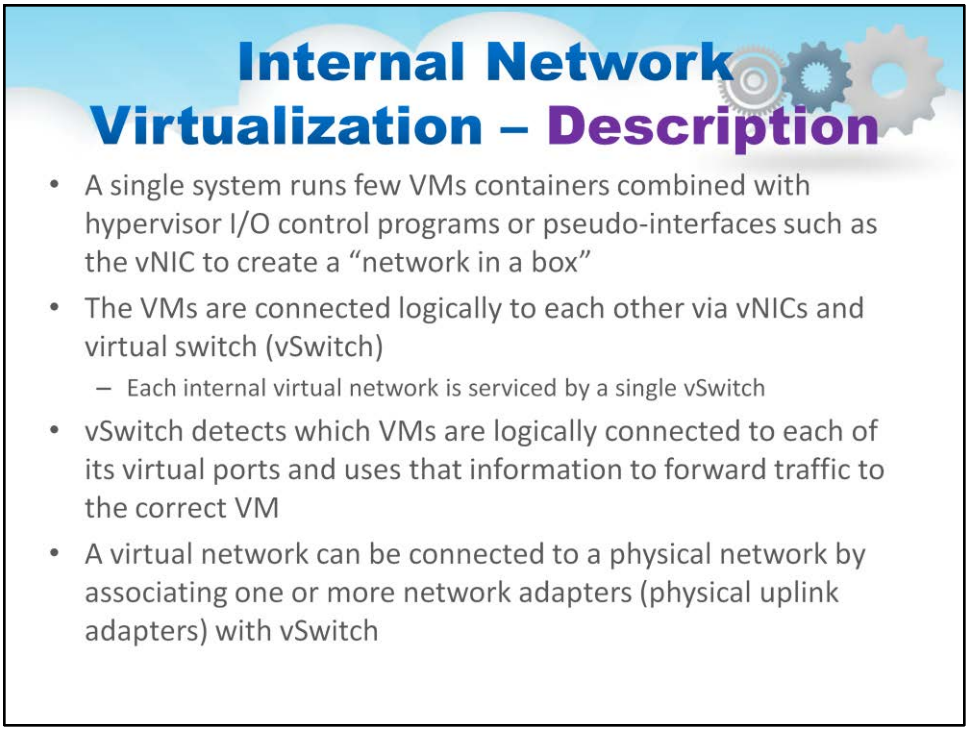
As the slide details, Layer 1 does not need to be emulated, as the physical layer is not emulated by the hypervisor, the drivers abstract that.

Layer 2 is where software is first exposed to network, this is where the vNIC exposes an Ethernet in the hypervisor, or where a vSwitch is used to do layer 2 bridging or switching.

Layer 3 Implements the virtual L3 network devices, such as router, in the hypervisor

Layer 4 or higher layers virtualization is usually implemented in guest OS

In Linux, one can see TUN and TAP virtual network kernel devices, as explained, TUN operates as router; while TAP is used for creating a network bridge



Internal Network Virtualization – Description

- A single system runs few VMs containers combined with hypervisor I/O control programs or pseudo-interfaces such as the vNIC to create a “network in a box”
- The VMs are connected logically to each other via vNICs and virtual switch (vSwitch)
  - Each internal virtual network is serviced by a single vSwitch
- vSwitch detects which VMs are logically connected to each of its virtual ports and uses that information to forward traffic to the correct VM
- A virtual network can be connected to a physical network by associating one or more network adapters (physical uplink adapters) with vSwitch

Now let us turn our detailed look towards Internal Network Virtualization

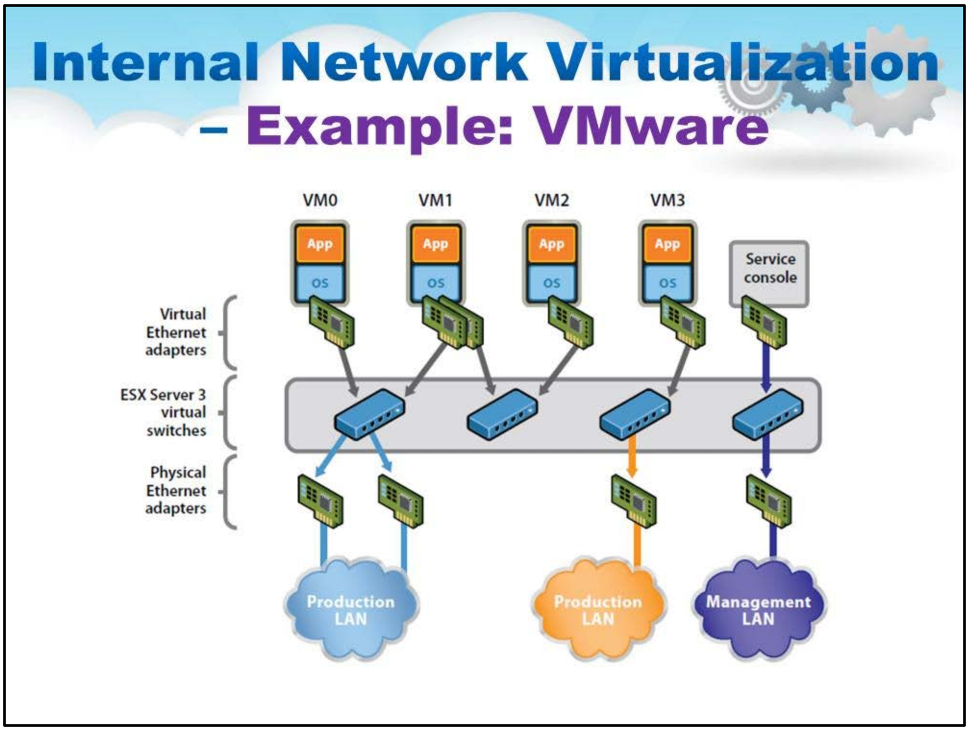
At the core of this, is that each single system which runs a few VMs containers is already combined with hypervisor I/O control programs or pseudo-interfaces such as the vNIC to create a “network in a box”

The VMs are connected logically to each other via vNICs and virtual switch (vSwitch)

Each internal virtual network is serviced by a single vSwitch

The vSwitch detects which VMs are logically connected to each of its virtual ports and uses that information to forward traffic to the correct VM

A virtual network can be connected to a physical network by associating one or more network adapters (physical uplink adapters) with vSwitch

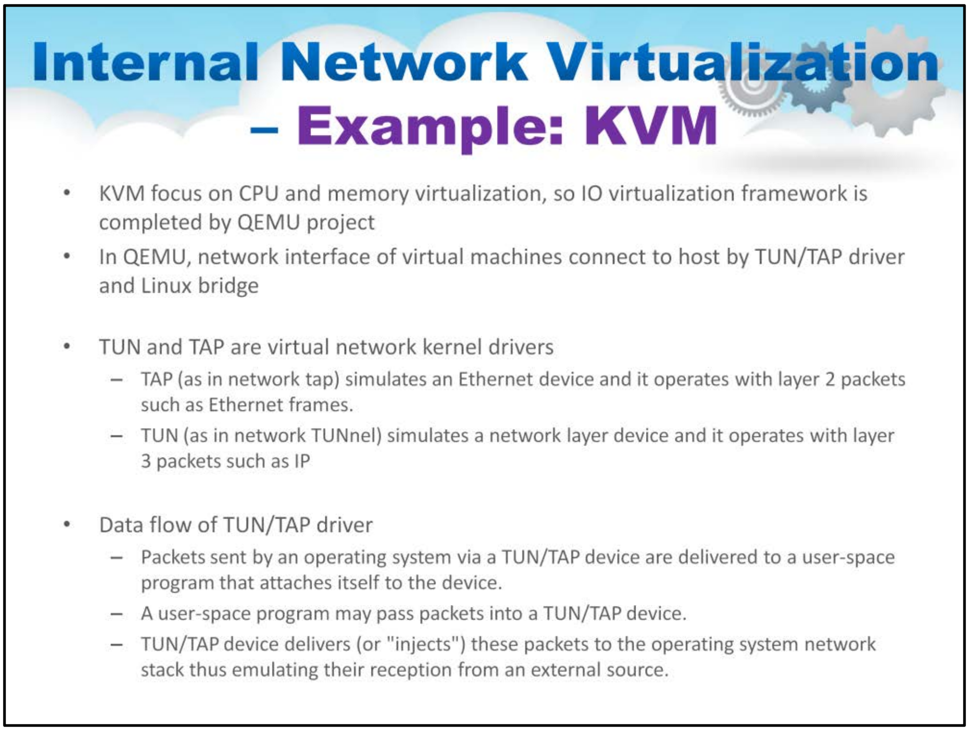


Internal Network Virtualization – Example: VMware

This slide illustrates nicely an example of Internal Network Virtualization from VMware.

One can see the vNICs and Virtual Switches

One Connects VMs to different networks via vSwitch and vNICs



Internal Network Virtualization – Example: KVM

- KVM focus on CPU and memory virtualization, so IO virtualization framework is completed by QEMU project
- In QEMU, network interface of virtual machines connect to host by TUN/TAP driver and Linux bridge

- TUN and TAP are virtual network kernel drivers
  - TAP (as in network tap) simulates an Ethernet device and it operates with layer 2 packets such as Ethernet frames.
  - TUN (as in network TUNnel) simulates a network layer device and it operates with layer 3 packets such as IP

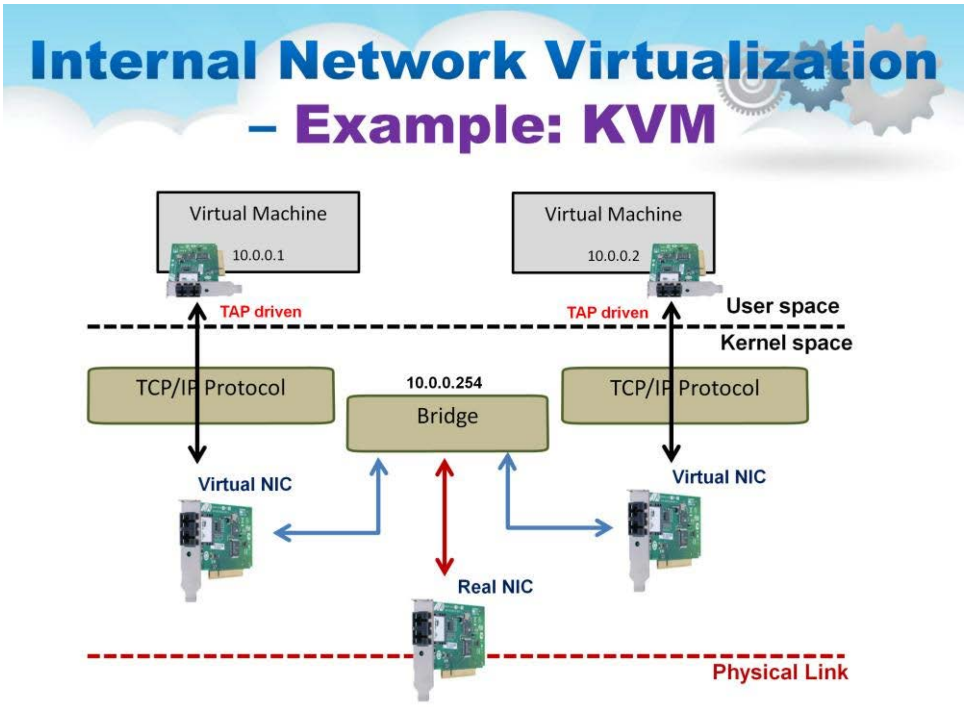
- Data flow of TUN/TAP driver
  - Packets sent by an operating system via a TUN/TAP device are delivered to a user-space program that attaches itself to the device.
  - A user-space program may pass packets into a TUN/TAP device.
  - TUN/TAP device delivers (or "injects") these packets to the operating system network stack thus emulating their reception from an external source.

Now lets look at Internal Network virtualization in KVM

KVM focuses on CPU and memory virtualization, so IO virtualization framework is completed by QEMU project

In QEMU, network interface of virtual machines connect to host by TUN/TAP driver and Linux bridge

The operation of TUN and TAP are described in the slide.



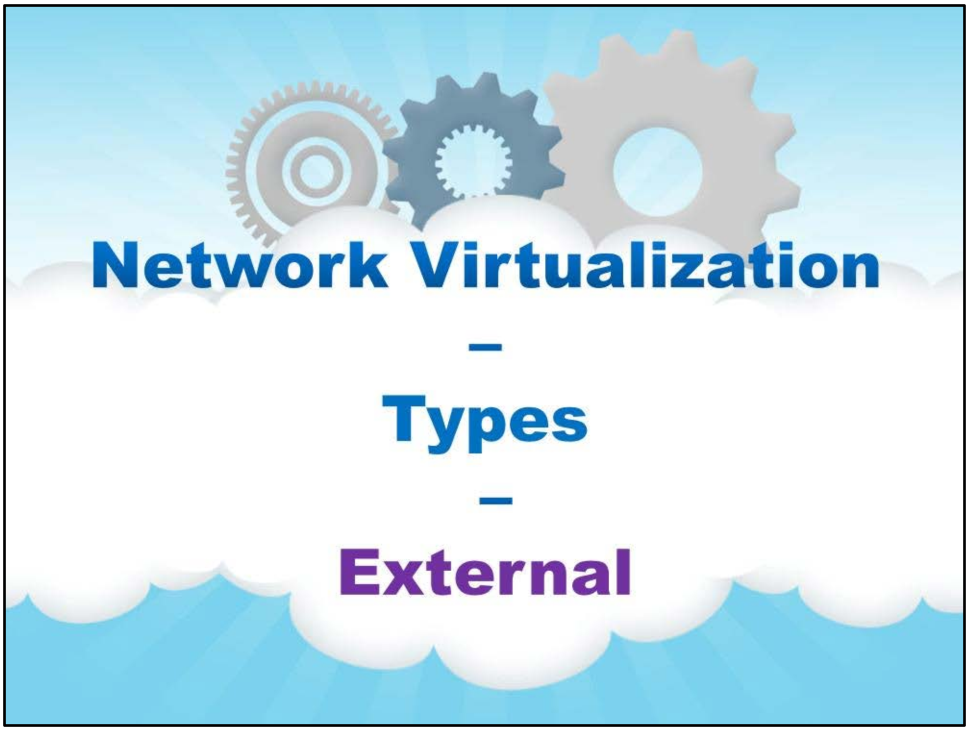
Internal Network Virtualization – Example: KVM

In the KVM/Linux bridge, as can bee seen in the illustration on this slide, the bridge is what is used to connect the vNICs with the real physical NIC,

Bridging is a forwarding technique used in packet-switched computer networks. Unlike routing, bridging makes no assumptions about where in a network a particular address is located.

Bridging depends on flooding and examination of source addresses in received packet headers to locate unknown devices.

Bridging connects multiple network segments at the data link layer (Layer 2) of the OSI model.



Network Virtualization

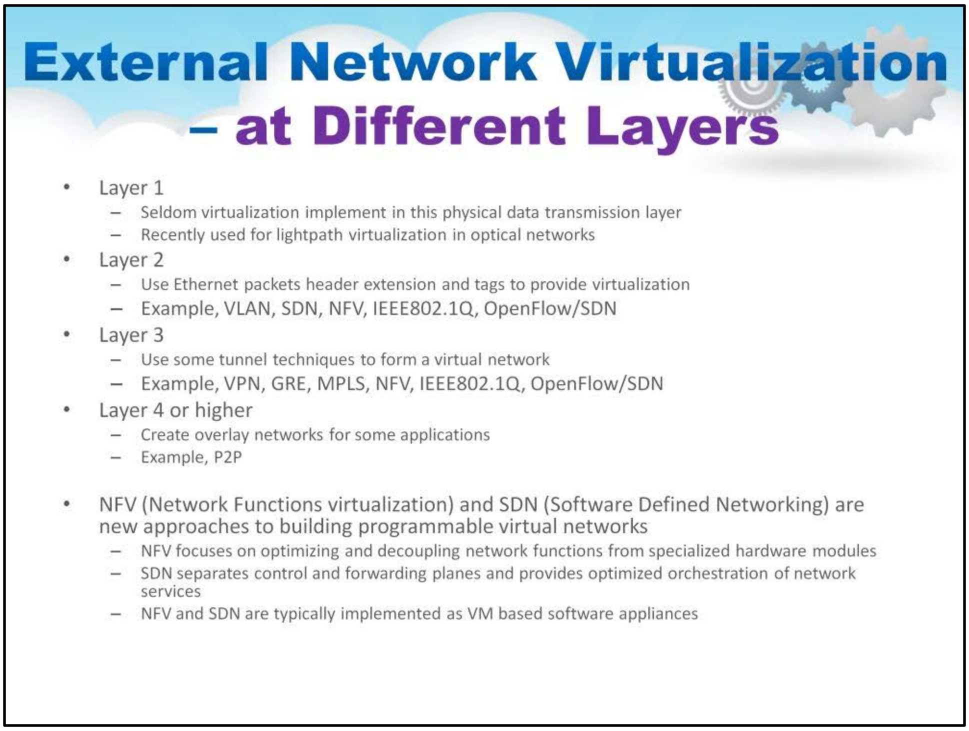
–

Types

–

External

External



External Network Virtualization – at Different Layers

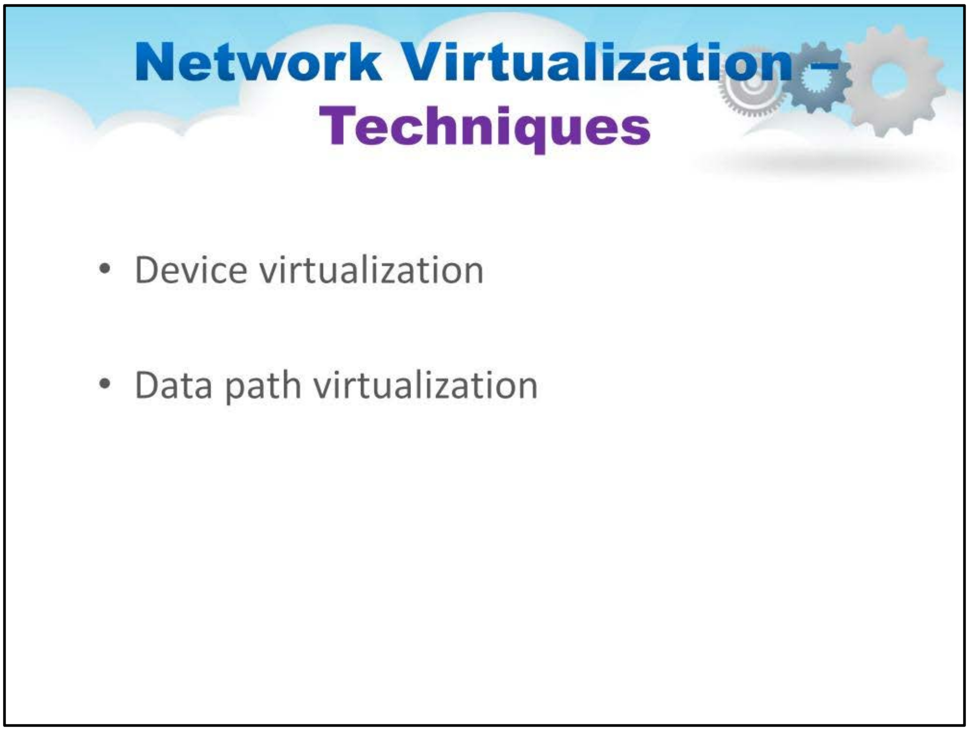
- Layer 1
  - Seldom virtualization implement in this physical data transmission layer
  - Recently used for lightpath virtualization in optical networks
- Layer 2
  - Use Ethernet packets header extension and tags to provide virtualization
  - Example, VLAN, SDN, NFV, IEEE802.1Q, OpenFlow/SDN
- Layer 3
  - Use some tunnel techniques to form a virtual network
  - Example, VPN, GRE, MPLS, NFV, IEEE802.1Q, OpenFlow/SDN
- Layer 4 or higher
  - Create overlay networks for some applications
  - Example, P2P
- NFV (Network Functions virtualization) and SDN (Software Defined Networking) are new approaches to building programmable virtual networks
  - NFV focuses on optimizing and decoupling network functions from specialized hardware modules
  - SDN separates control and forwarding planes and provides optimized orchestration of network services
  - NFV and SDN are typically implemented as VM based software appliances

No let us look at External Network Virtualization at Different Layers

Again, Layer 1 is rarely accessed by software Layer 2 has seen a lot of technology around virtualization lately, we will spend more time on Layer 2 virtualizations such as VLAN, SDN, NFV, IEEE802.1Q, OpenFlow/SDN.

Layer 2 techniques all use Use Ethernet packets header extensions and tags to transmit across the network that a virtualization is in place and some of these conventions have been standardized. This is an active area.

Layer 3 is in the same situation, lots of development usually using some tunnel techniques to form a virtual network -
Example, VPN, GRE, MPLS, NFV, IEEE802.1Q, OpenFlow/SDN.



Network Virtualization – Techniques

- Device virtualization
- Data path virtualization

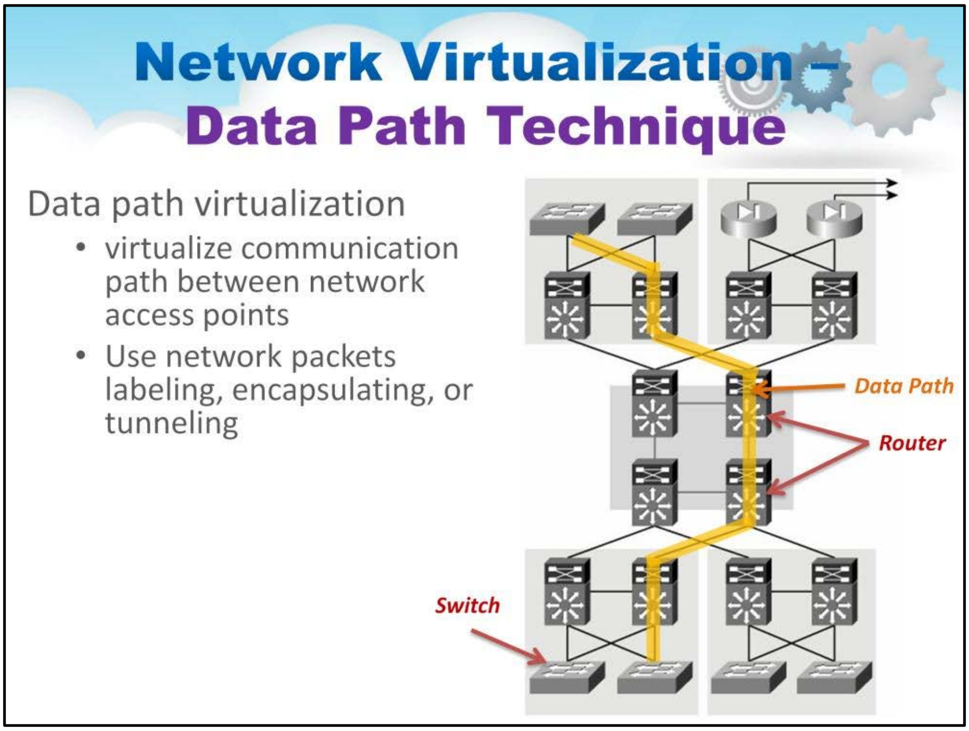
Network Virtualization usually utilizes two main techniques to get it’s job done, that is Device and Path virtualization

These place network virtualization in two places, one at the device, and one in the pathing of traffic,

• Device virtualization virtualizes physical devices in the network

• Data path virtualization virtualizes communication path between network access points

One can see from the illustration how a complete virtualized network can be constructed using Device and Path virtualization



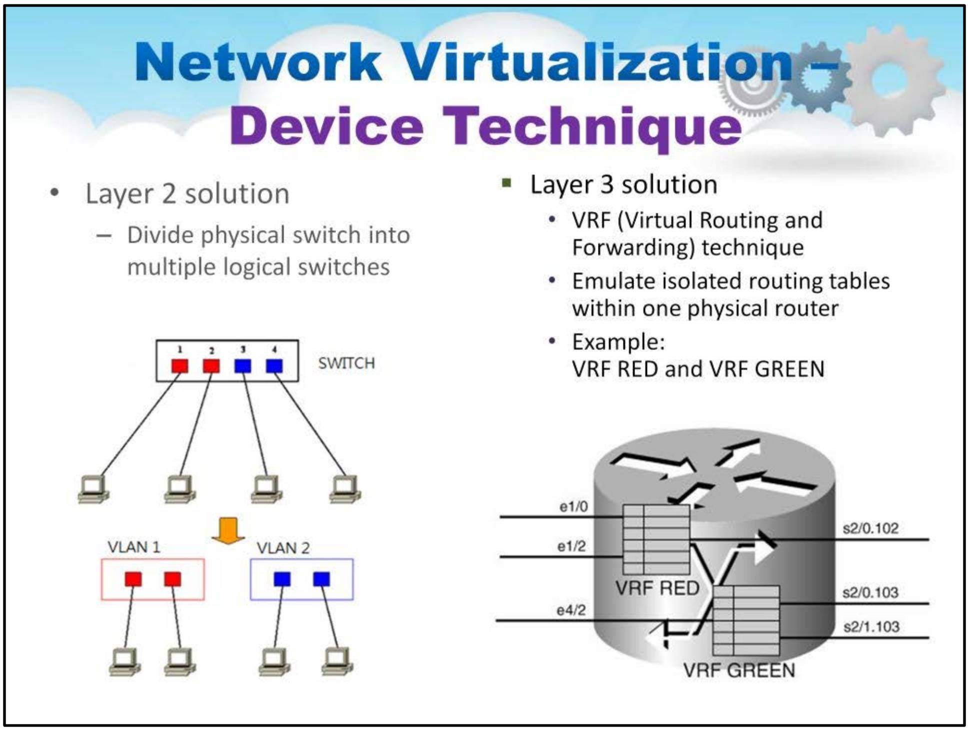
Network Virtualization – Data Path Technique

Data path virtualization

- virtualize communication path between network access points
- Use network packets labeling, encapsulating, or tunneling

Data path virtualization virtualizes communication path between network access points

One can see from the illustration how a complete virtualized network can be constructed using Device and Path virtualization



Network Virtualization – Device Technique

- Layer 2 solution
  - Divide physical switch into multiple logical switches

- Layer 3 solution
  - VRF (Virtual Routing and Forwarding) technique
  - Emulate isolated routing tables within one physical router
  - Example:
    VRF RED and VRF GREEN

Device virtualization is done differently for each type of network device to be constructed

A layer 2 (switch level) solution, creates numbers of switches, separating the traffic amongst the switches. In a software implementation they are multiple vSwitches, In a hardware device this is done by the ASICs, and is called VLAN for Virtual LAN

A layer 3 (router level) solution utilizes routing tables as one would suspect, making separate routing tables or more accurately routing domains for each Layer 3 network,

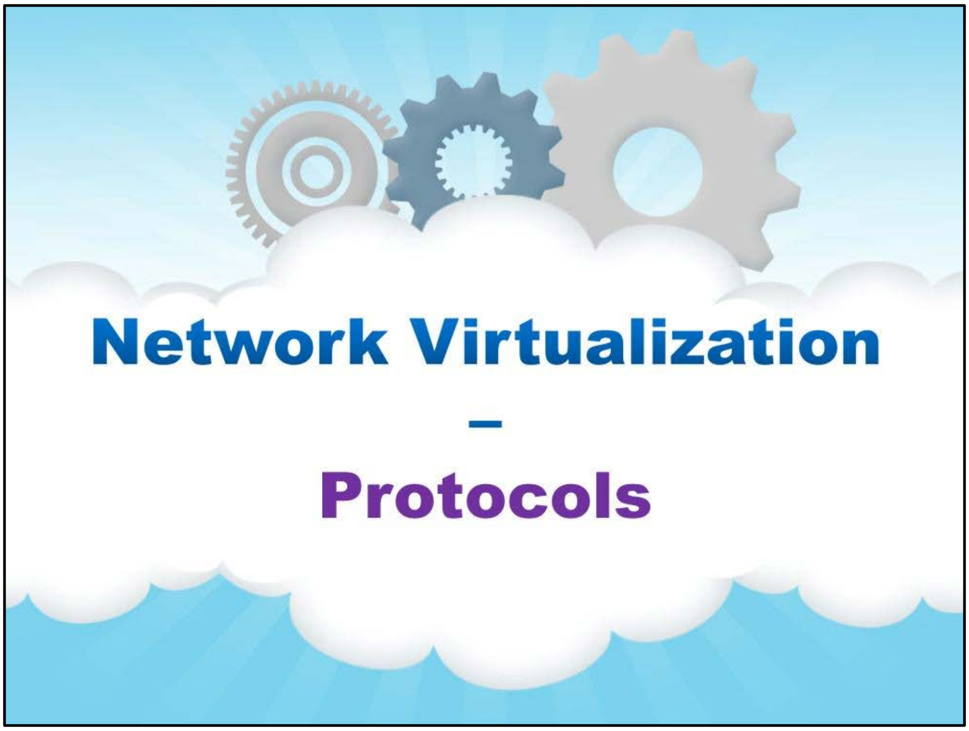
In a software implementation the VRF (Virtual Routing and

Forwarding) technique is used, in a hardware

device the router create separate layer 3 domains

within the one router (with separate routing

protocols, rout processors, and tables).

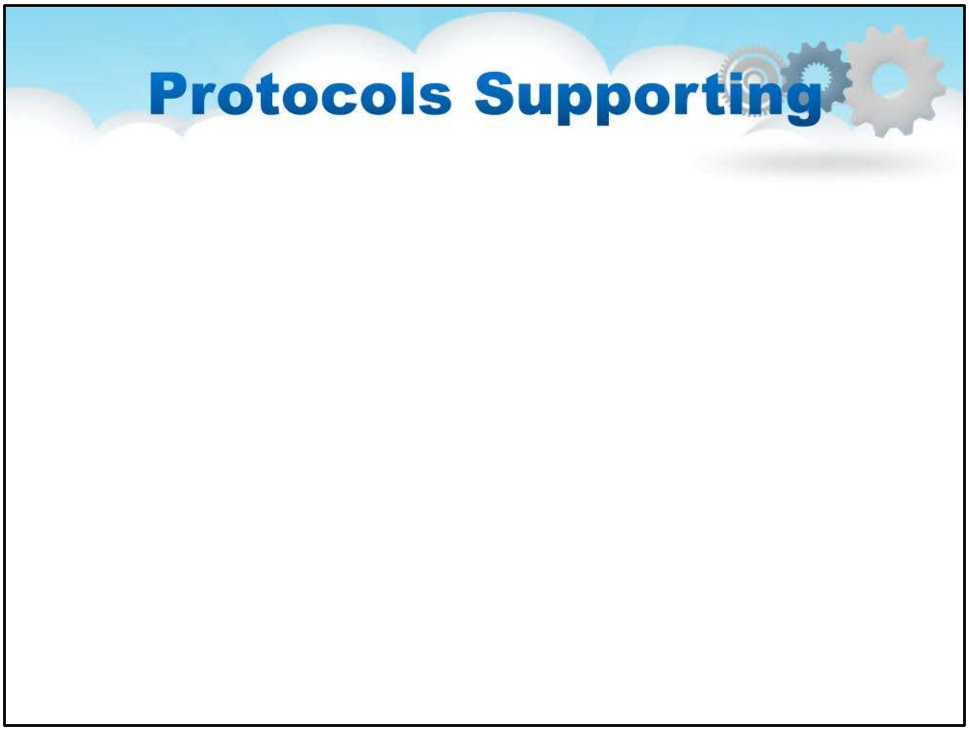


Network Virtualization

–

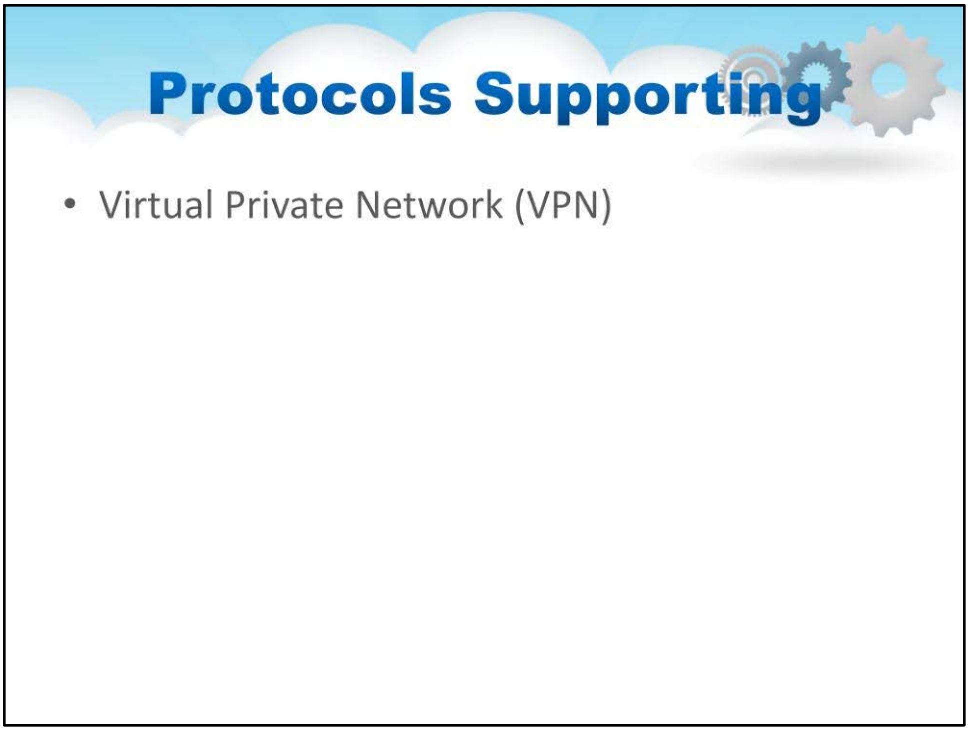
Protocols

Protocols



Protocols Supporting

Protocols Supporting Network Virtualization

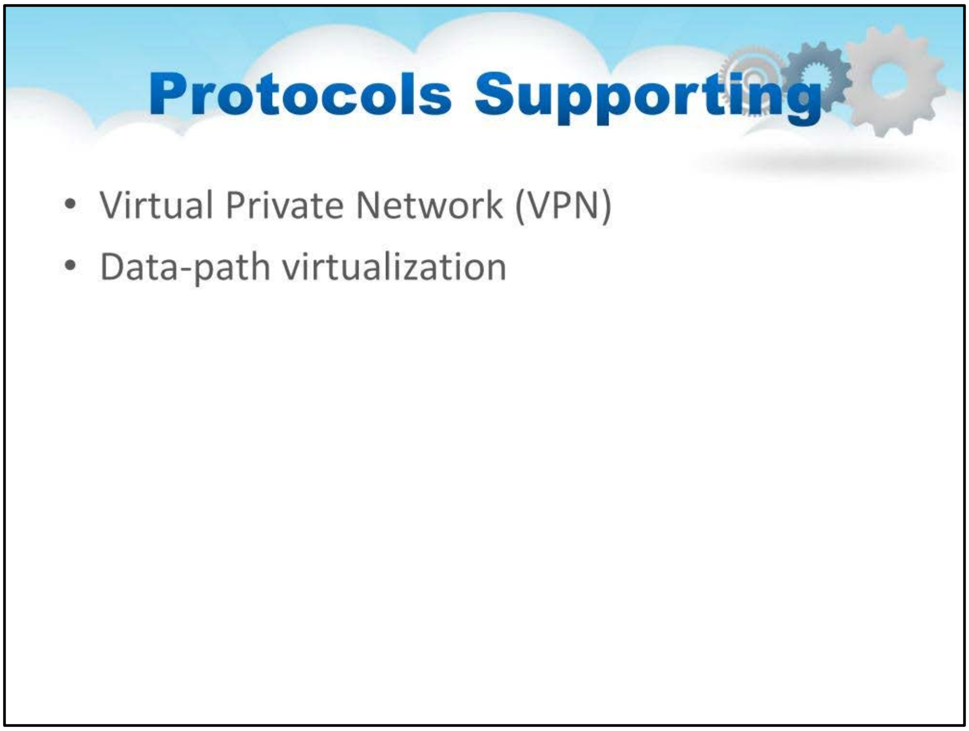


Protocols Supporting

- Virtual Private Network (VPN)

Virtual Private Network (VPN)

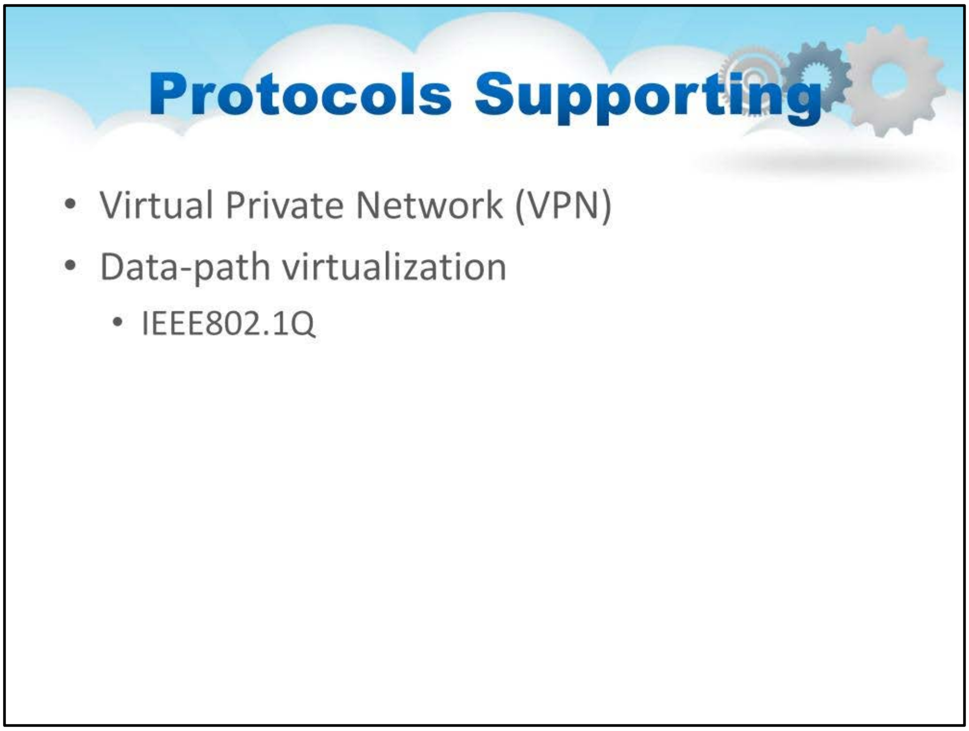
- **IPSec** based tunneling protocol interconnecting two network locations
- The most widely used solution for interconnecting remote offices and roaming users



Protocols Supporting

- Virtual Private Network (VPN)
- Data-path virtualization

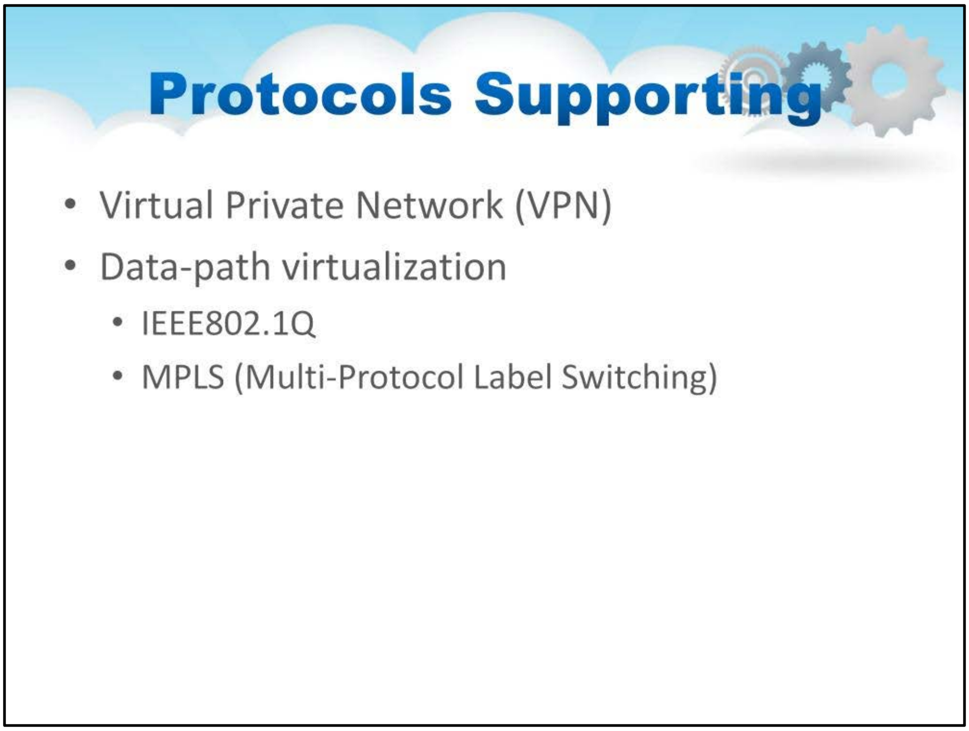
Data-path virtualization is realised based on special network protocols such as



Protocols Supporting

- Virtual Private Network (VPN)
- Data-path virtualization
  - IEEE802.1Q

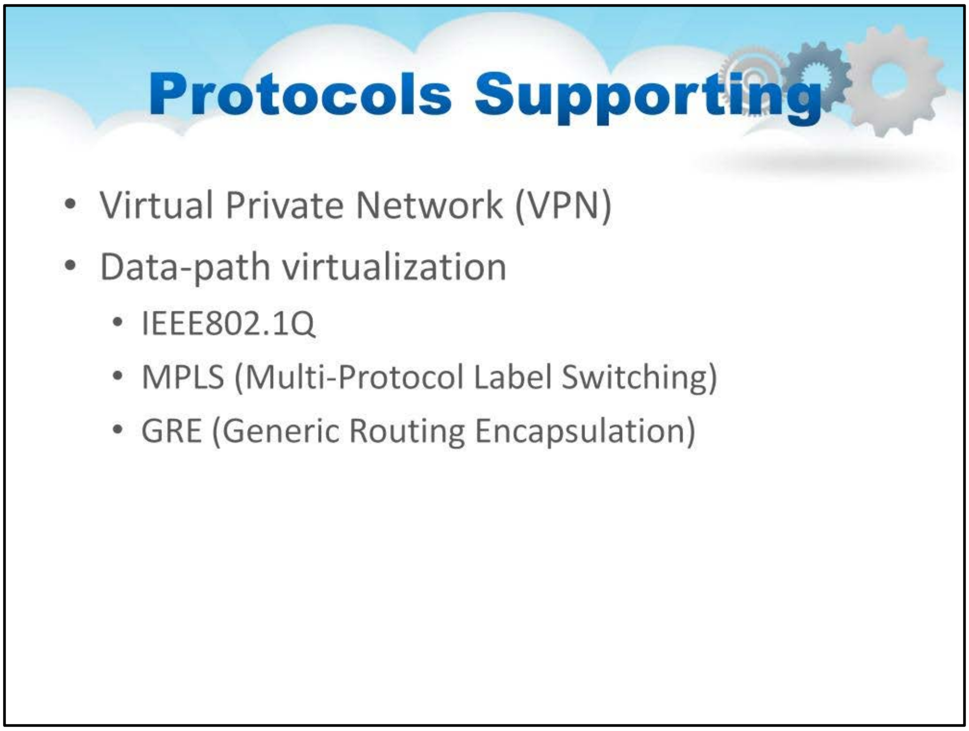
IEEE802.1Q is a Layer 2 virtualization protocol that implements hop to hop data-path virtualization and allows for creation of Virtual Local Area Networks.



Protocols Supporting

- Virtual Private Network (VPN)
- Data-path virtualization
  - IEEE802.1Q
  - MPLS (Multi-Protocol Label Switching)

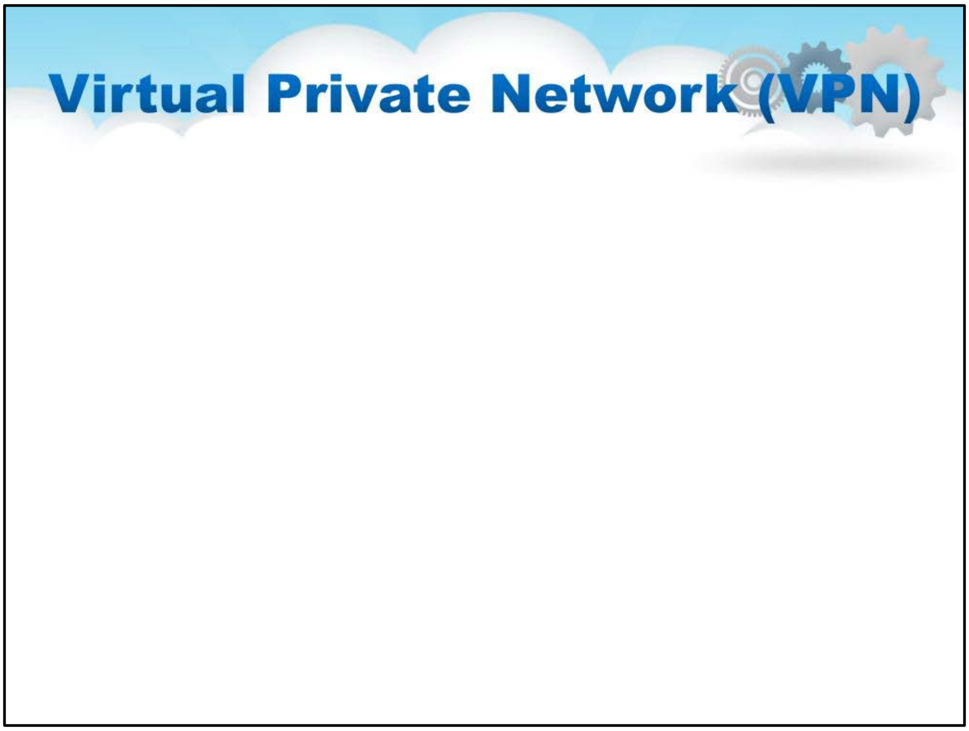
MPLS (Multi-Protocol Label Switching) is a Layer 3 network path virtualization protocol that operates at the level of network routers and switches;



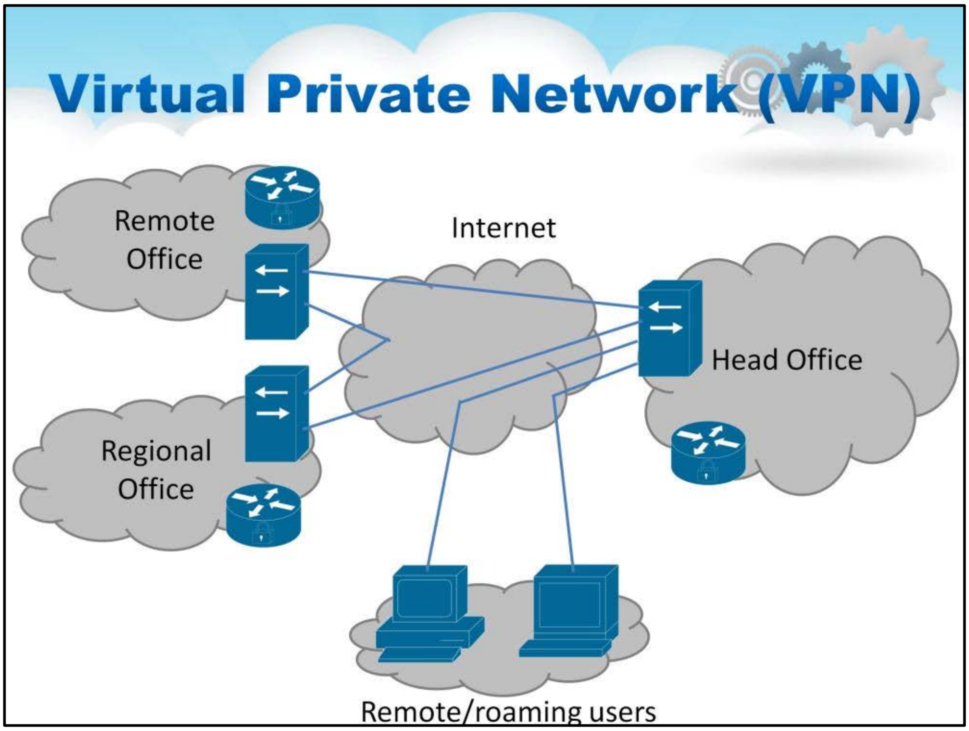
Protocols Supporting

- Virtual Private Network (VPN)
- Data-path virtualization
  - IEEE802.1Q
  - MPLS (Multi-Protocol Label Switching)
  - GRE (Generic Routing Encapsulation)

GRE (Generic Routing Encapsulation): implements virtualization among wide variety of networks with tunneling technique

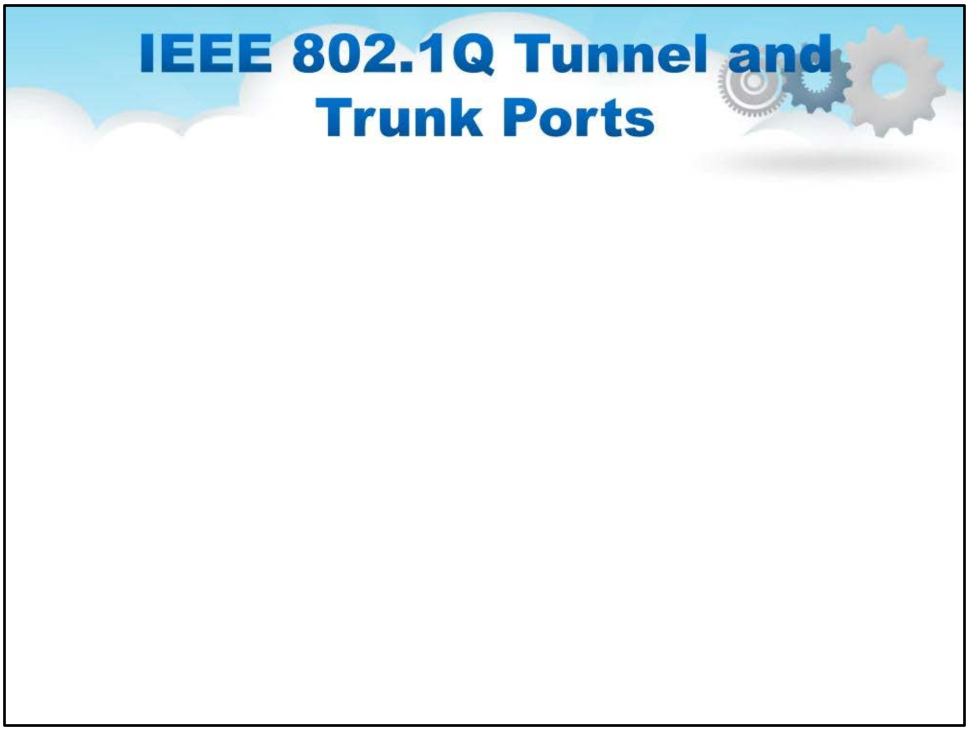


Virtual Private Network (VPN)

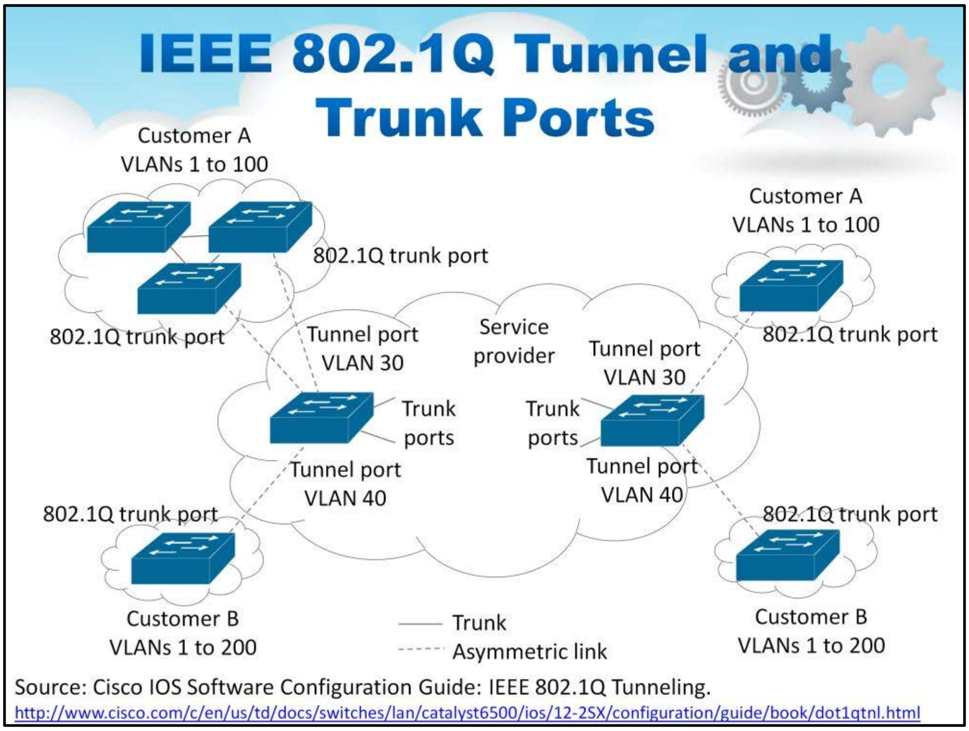


Virtual Private Network (VPN)

- Virtual Private Network (VPN) used to extend private network over public Internet to remote locations
  - Uses end-to-end secure IP protocol (IPSec)
- Originally VPN has one home/main office
- New technologies allow multi-homed VPN



IEEE 802.1Q Tunnel and Trunk Ports



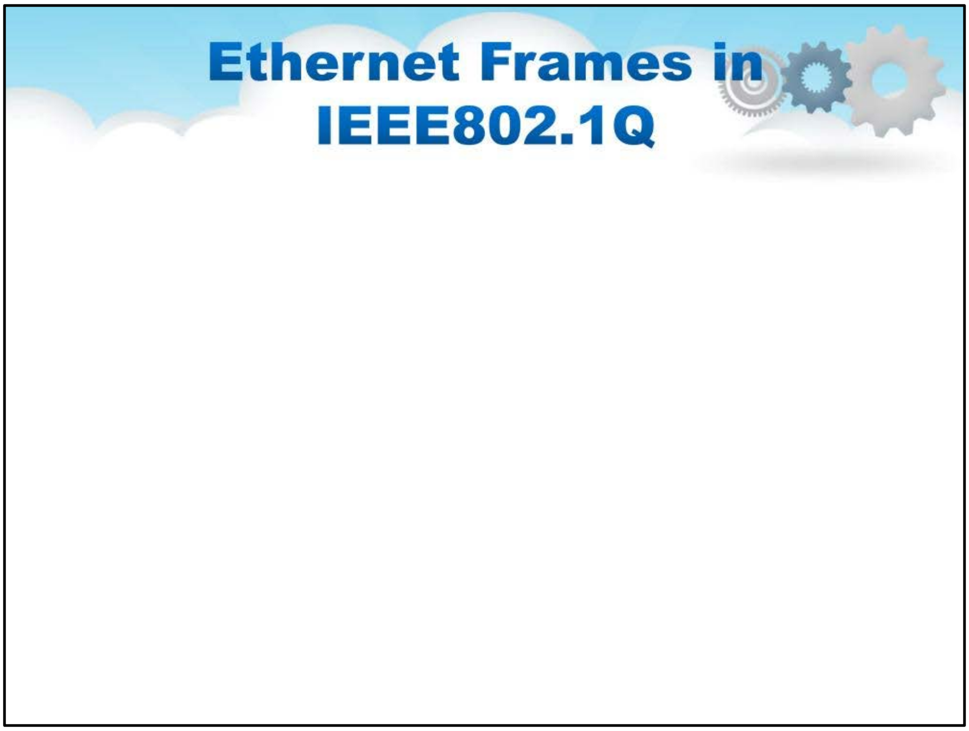
IEEE 802.1Q Tunnel and Trunk Ports

Source: Cisco IOS Software Configuration Guide: IEEE 802.1Q Tunneling.
http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dot1qtnl.html

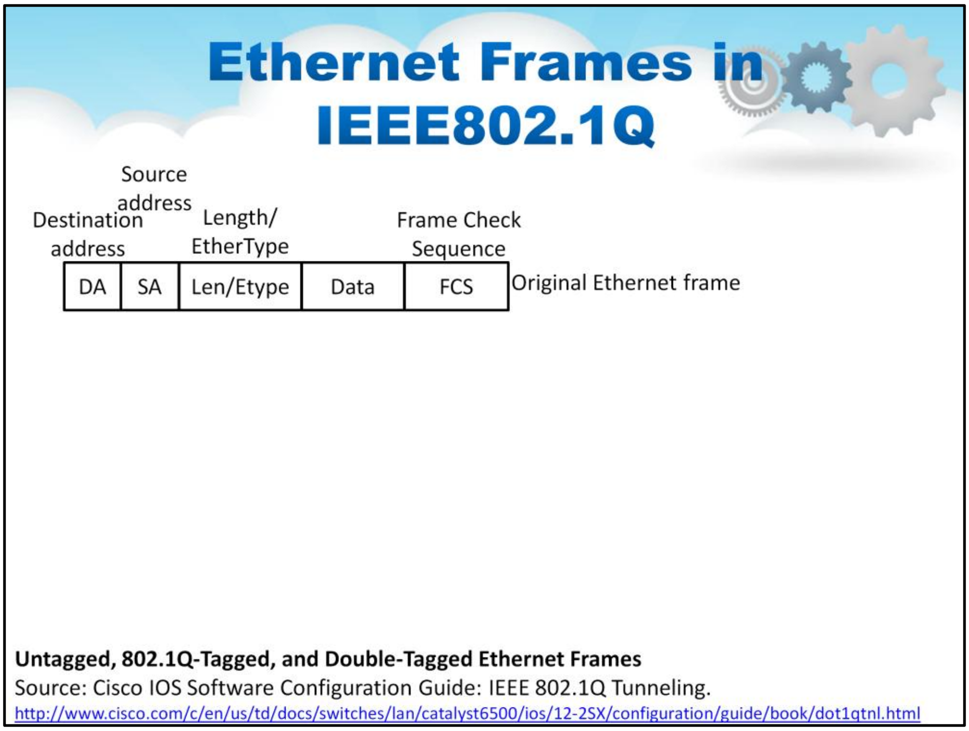
IEEE 802.1Q Tunnel Ports is in a Service-Provider network

- Multiple tunnel ports are supported

IEEE 802.1Q Trunk Port in Customer network



Ethernet Frames in IEEE802.1Q



Ethernet Frames in IEEE802.1Q

Destination address	Source address	Length/ EtherType		Frame Check Sequence	
DA	SA	Len/Etype	Data	FCS	Original Ethernet frame

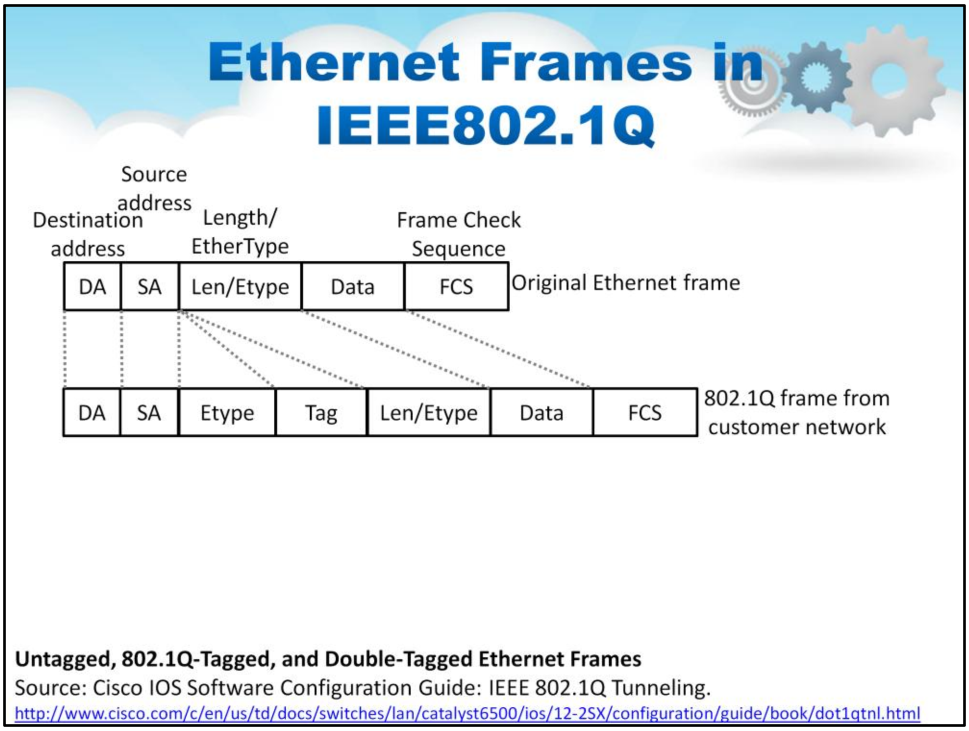
Untagged, 802.1Q-Tagged, and Double-Tagged Ethernet Frames

Source: Cisco IOS Software Configuration Guide: IEEE 802.1Q Tunneling.

http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dot1qtnl.html

IEEE802.1Q adds a 32-bit field between *MAC address* and *EtherTypes* field

- ETYPE(2B): Protocol identifier
- Dot1Q Tag(2B): VLAN number, Priority code



Ethernet Frames in IEEE802.1Q

Source address

Destination address

Length/ EtherType

Frame Check Sequence

DA	SA	Len/Etype	Data	FCS

Original Ethernet frame

DA	SA	Etype	Tag	Len/Etype	Data	FCS

802.1Q frame from customer network

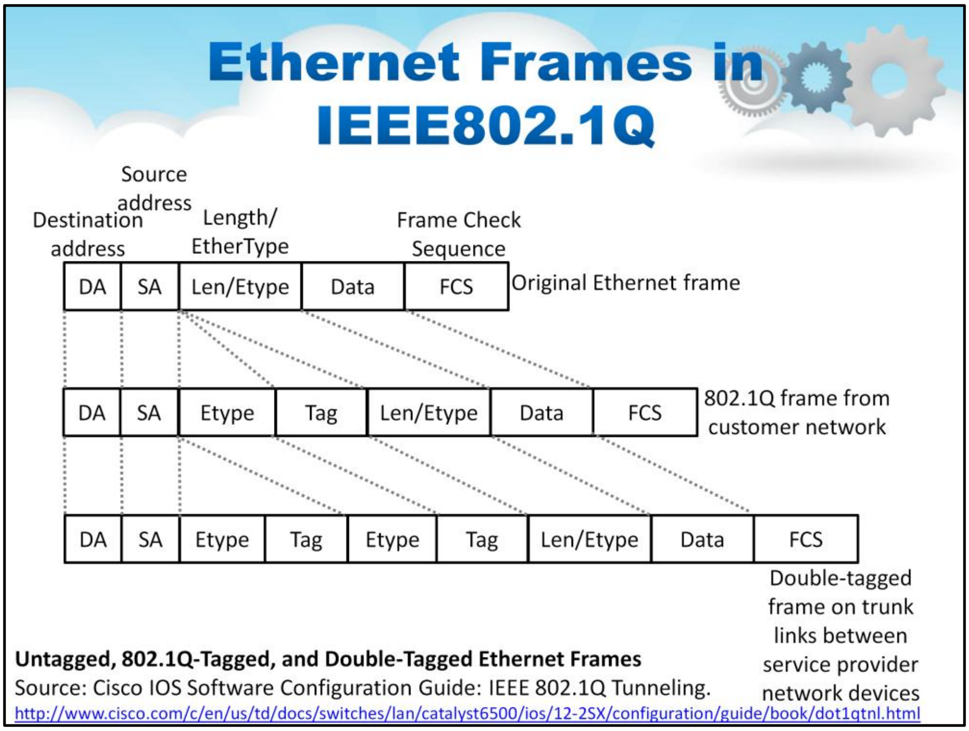
Untagged, 802.1Q-Tagged, and Double-Tagged Ethernet Frames

Source: Cisco IOS Software Configuration Guide: IEEE 802.1Q Tunneling.

http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dot1qtnl.html

IEEE802.1Q adds a 32-bit field between *MAC address* and *EtherTypes* field

- ETYPE(2B): Protocol identifier
- Dot1Q Tag(2B): VLAN number, Priority code



Ethernet Frames in IEEE802.1Q

DA | SA | Len/Etype | Data | FCS — Original Ethernet frame

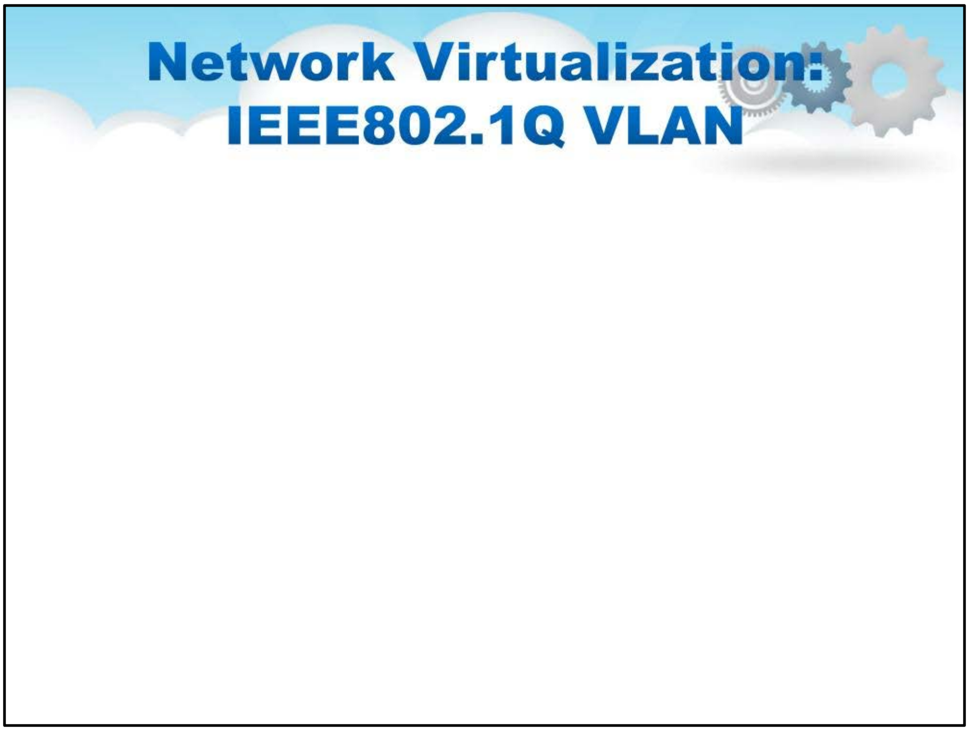
DA | SA | Etype | Tag | Len/Etype | Data | FCS — 802.1Q frame from customer network

DA | SA | Etype | Tag | Etype | Tag | Len/Etype | Data | FCS — Double-tagged frame on trunk links between service provider network devices

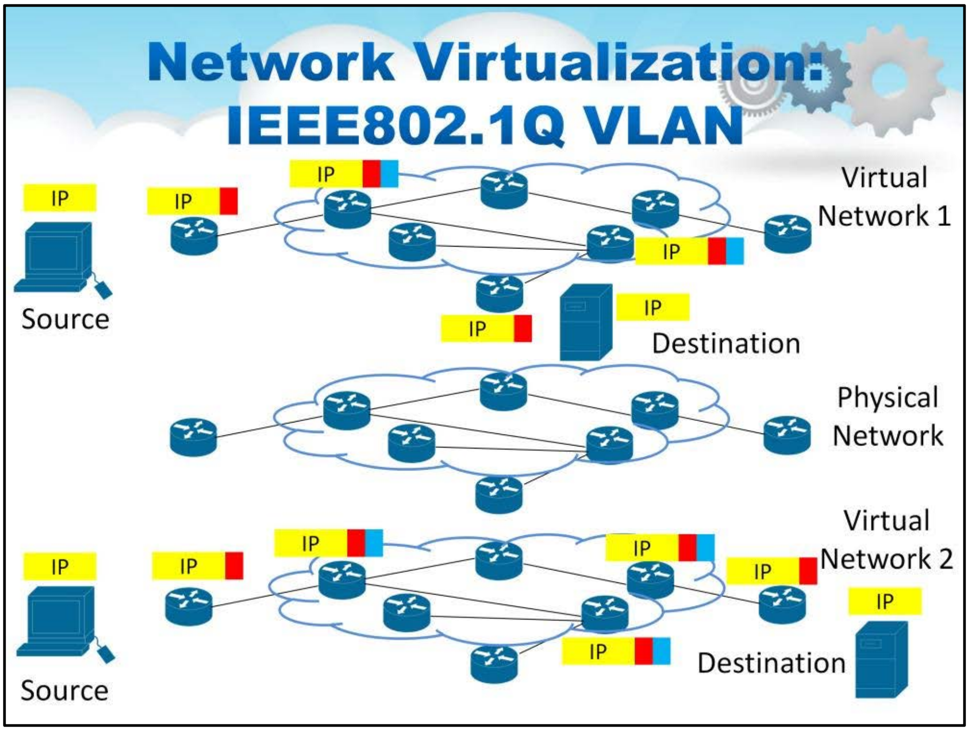
Untagged, 802.1Q-Tagged, and Double-Tagged Ethernet Frames
Source: Cisco IOS Software Configuration Guide: IEEE 802.1Q Tunneling.
http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dot1qtnl.html

IEEE802.1Q adds a 32-bit field between *MAC address* and *EtherTypes* field

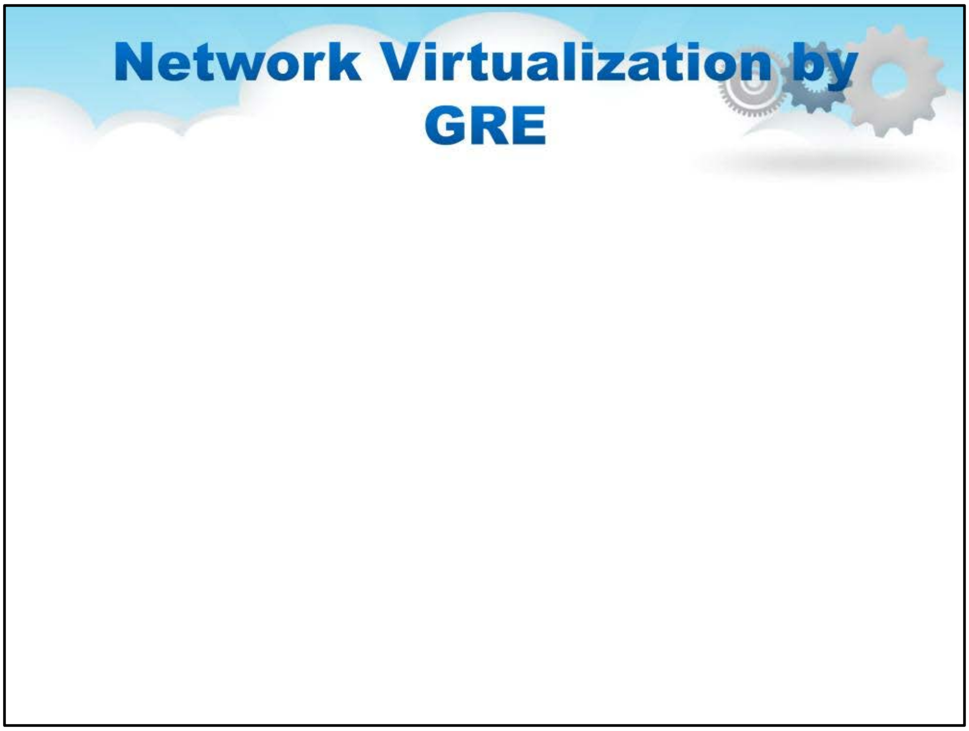
- ETYPE(2B): Protocol identifier
- Dot1Q Tag(2B): VLAN number, Priority code



Network Virtualization: IEEE802.1Q VLAN

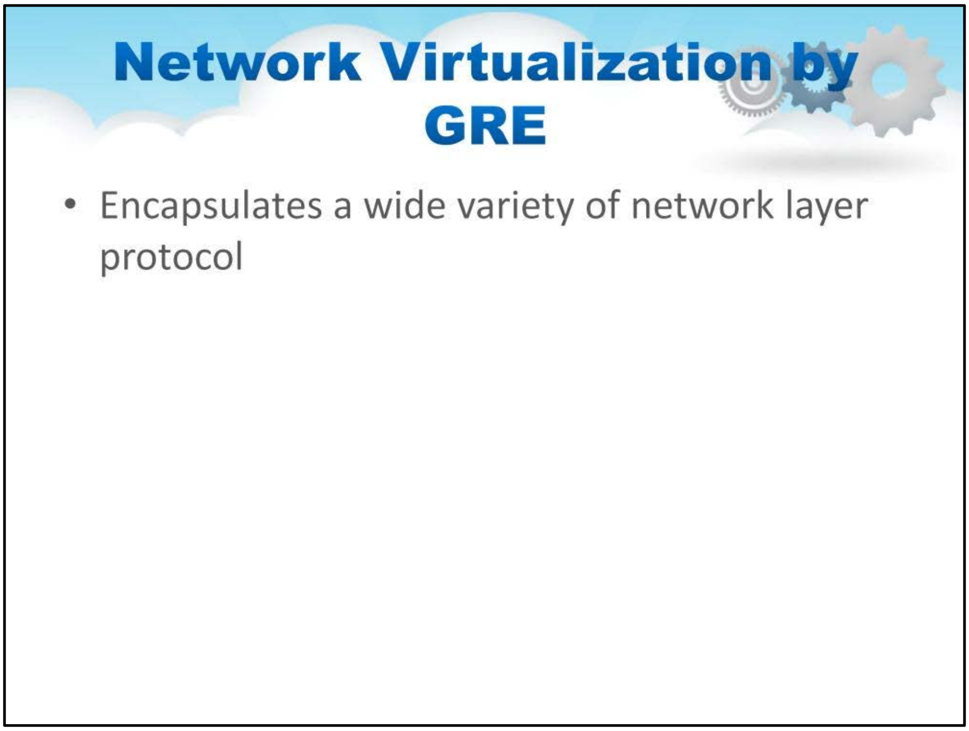


Network Virtualization: IEEE802.1Q VLAN



Network Virtualization by GRE

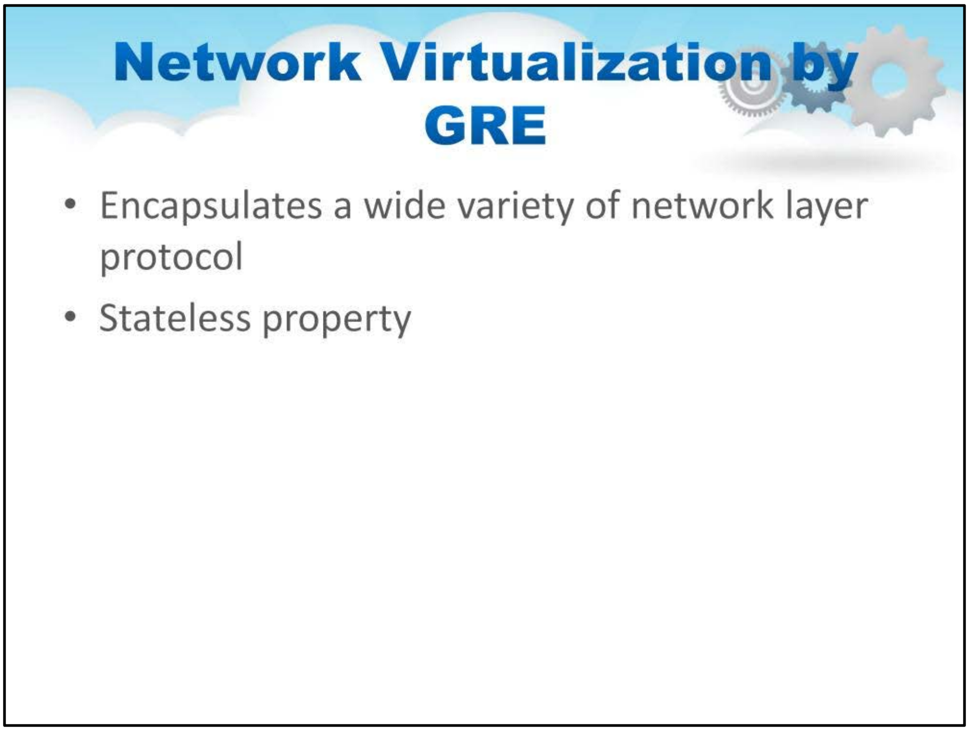
GRE (Generic Routing Encapsulation) is a tunnel protocol developed by Cisco



Network Virtualization by GRE

- Encapsulates a wide variety of network layer protocol

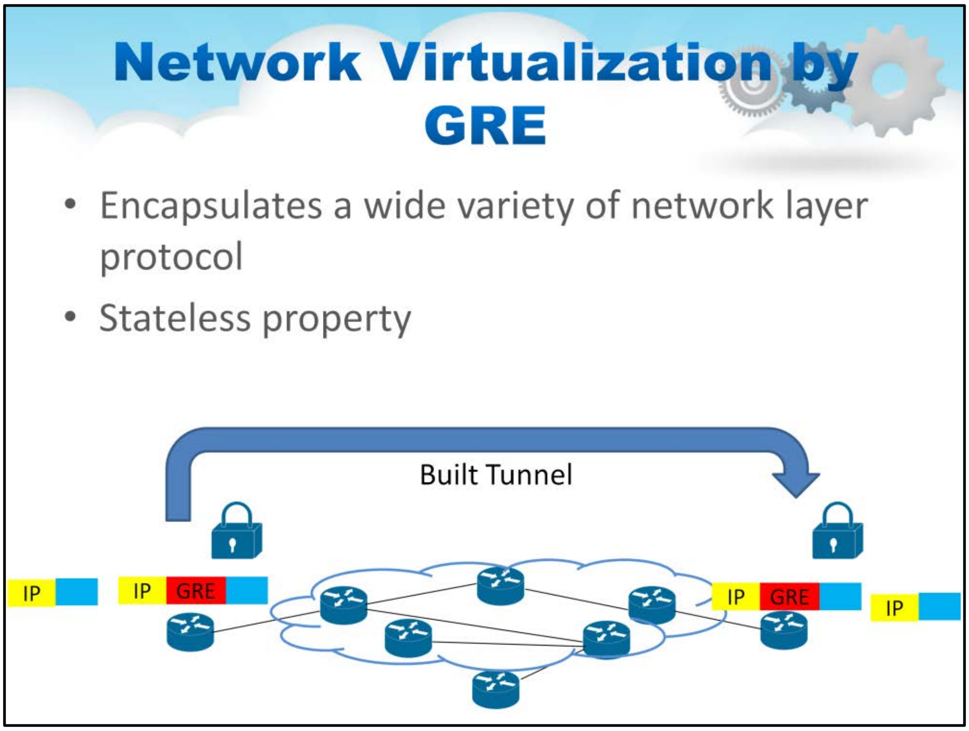
When combined with IPSec, allows for secure multi-homed VPN



Network Virtualization by GRE

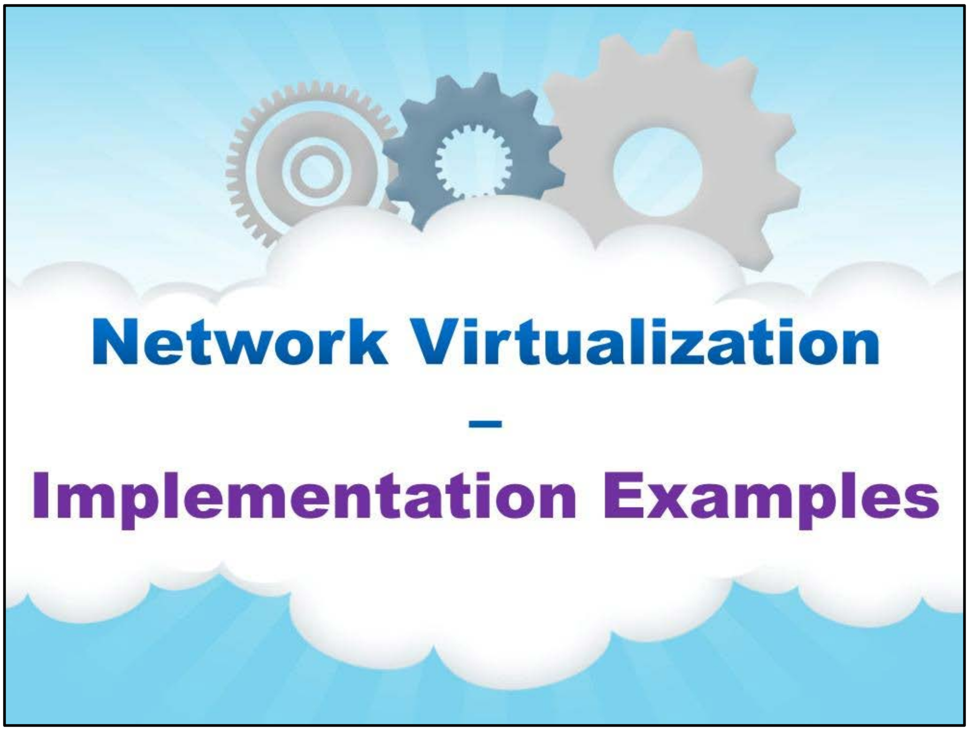
- Encapsulates a wide variety of network layer protocol
- Stateless property

This means end-point doesn't keep information about the state



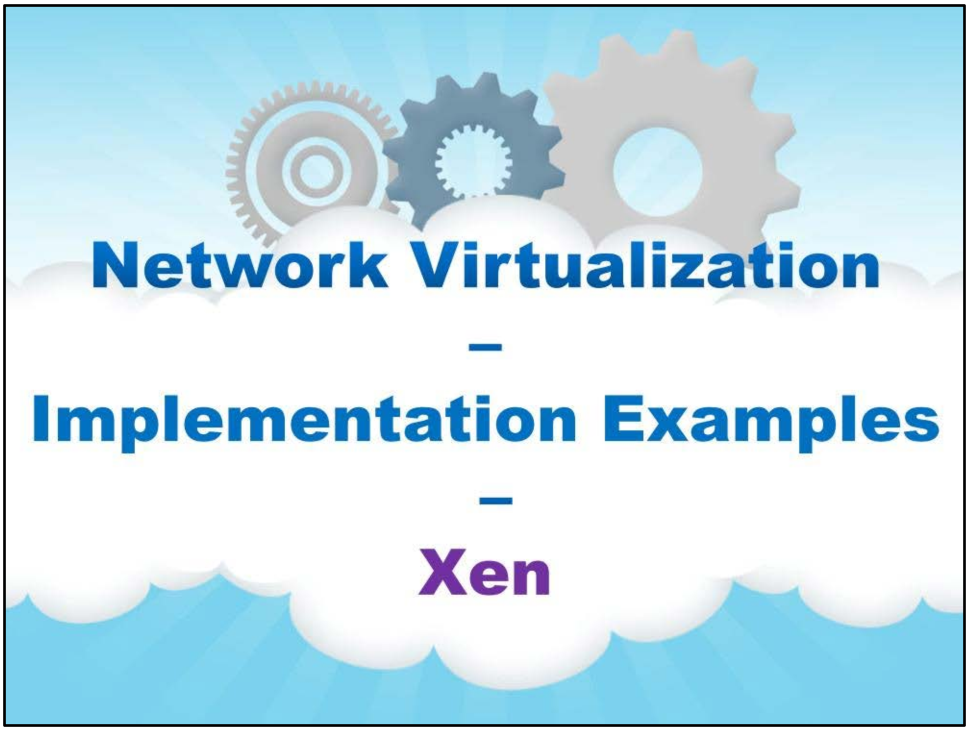
Network Virtualization by GRE

- Encapsulates a wide variety of network layer protocol
- Stateless property



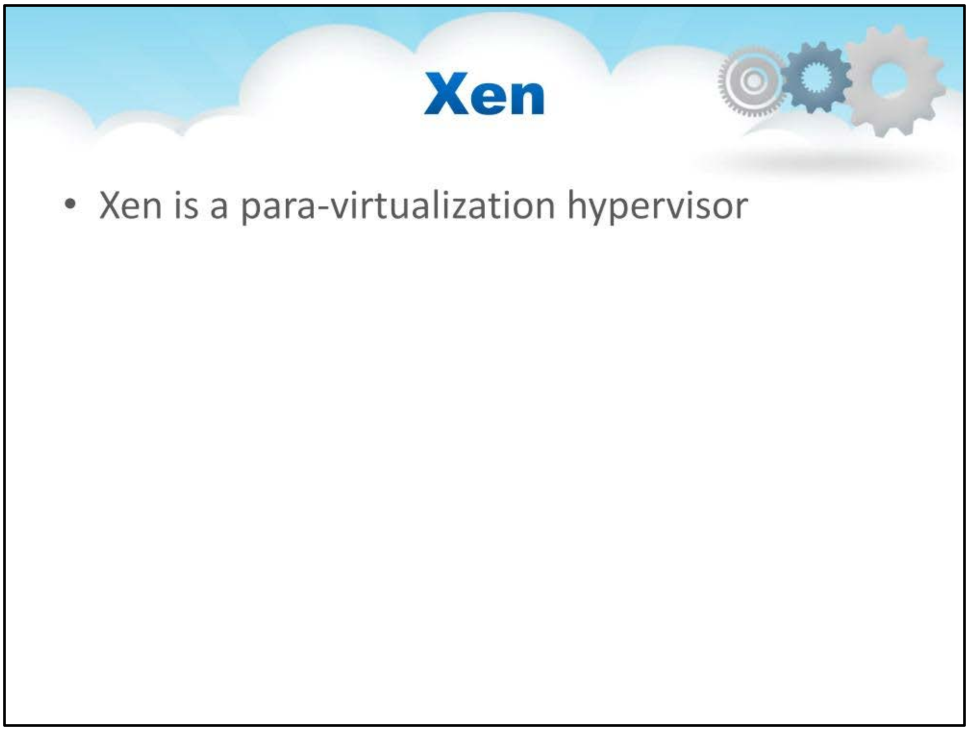
Network Virtualization – Implementation Examples

Implementation Examples



Network Virtualization
–
Implementation Examples
–
Xen

Xen



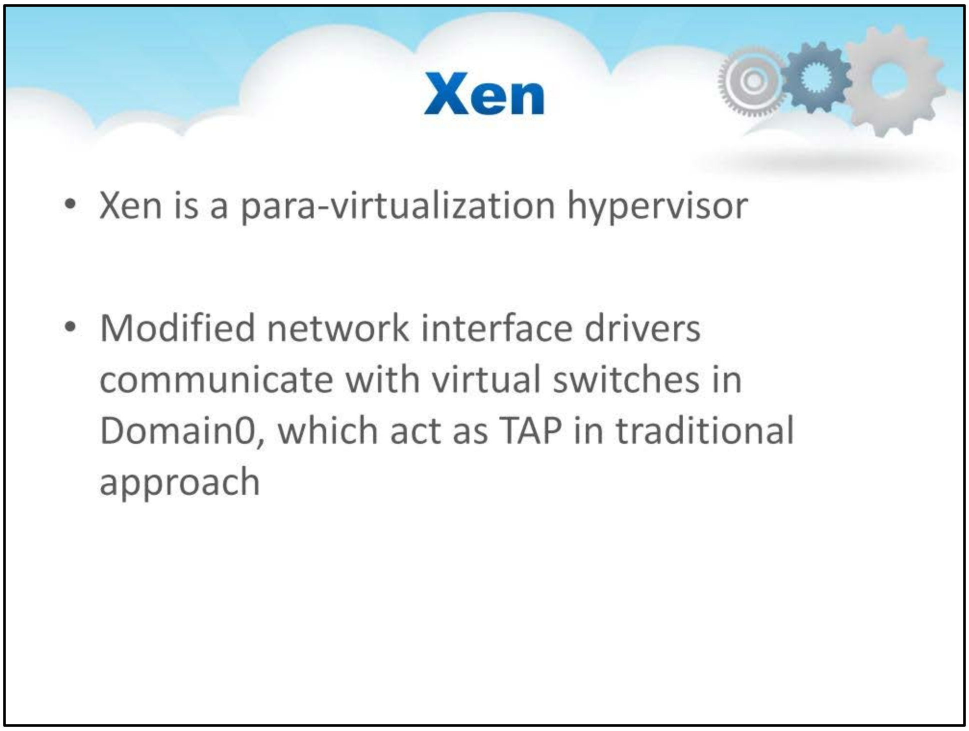
Xen

- Xen is a para-virtualization hypervisor

Next lets look at Xen. Xen is a para-virtualization hypervisor, so the guest OS uses modified network interface drivers

Modified network interface drivers communicate with virtual switches in Domain0, which act as TAP in traditional approach

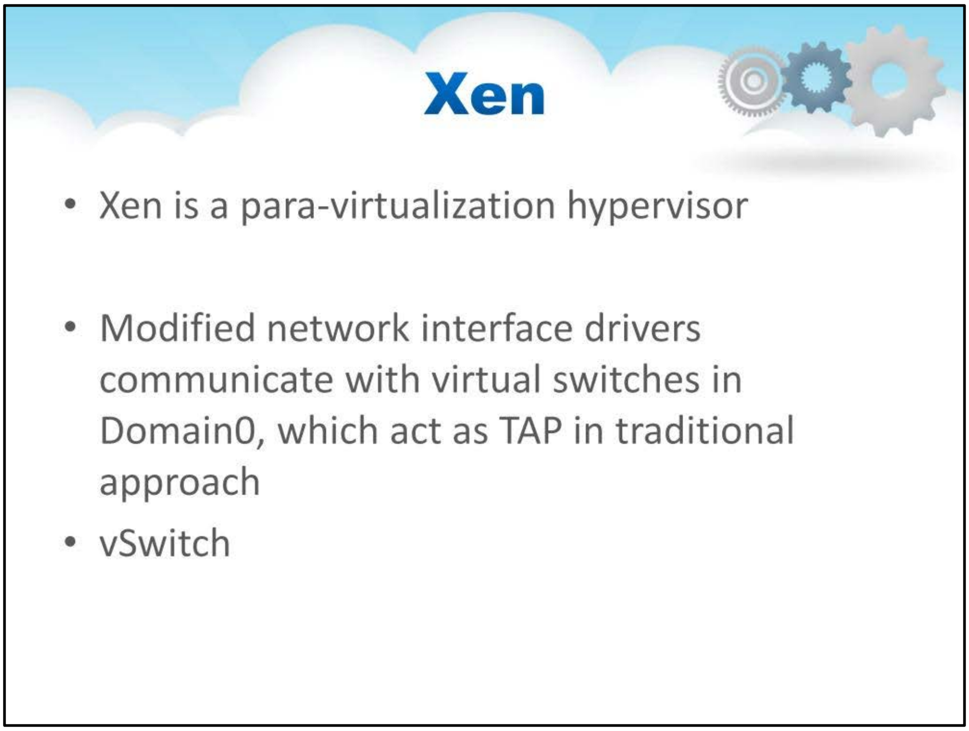
Xen is a para-virtualization hypervisor, so guest OS uses modified network interface drivers



Xen

- Xen is a para-virtualization hypervisor
- Modified network interface drivers communicate with virtual switches in Domain0, which act as TAP in traditional approach

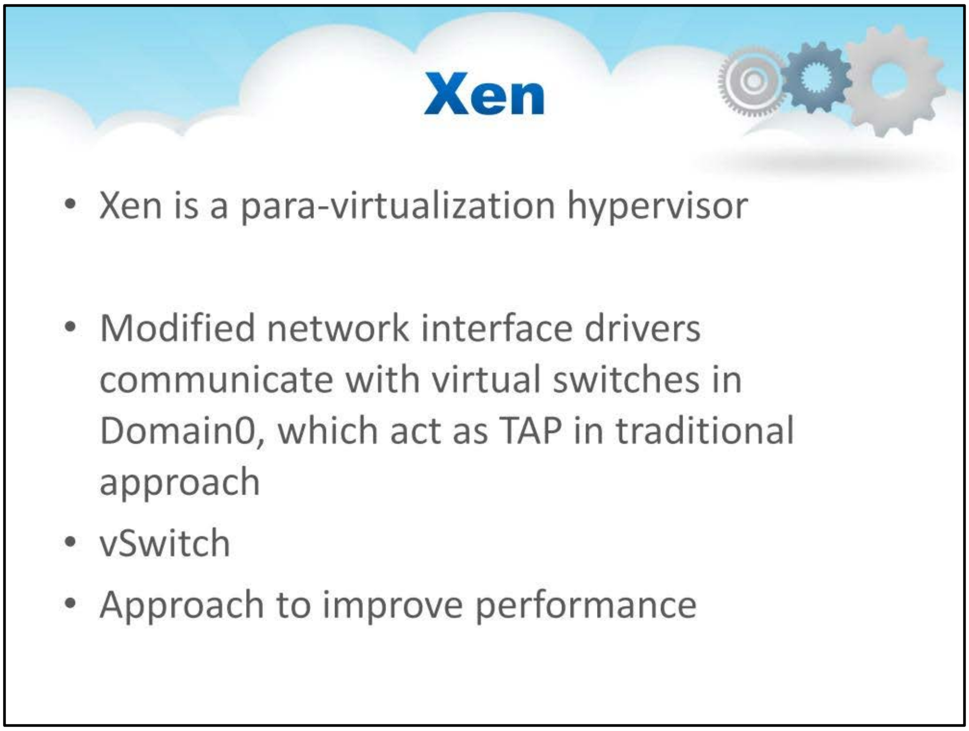
- Hypervisor remaps memory page for MMIO (Memory Mapped I/O)
- Whenever packets send, induce one context switch from guest to Domain 0 to drive real NIC



Xen

- Xen is a para-virtualization hypervisor
- Modified network interface drivers communicate with virtual switches in Domain0, which act as TAP in traditional approach
- vSwitch

vSwitch in Xen can be implemented by Linux bridge or work with other optimized bridge implementations

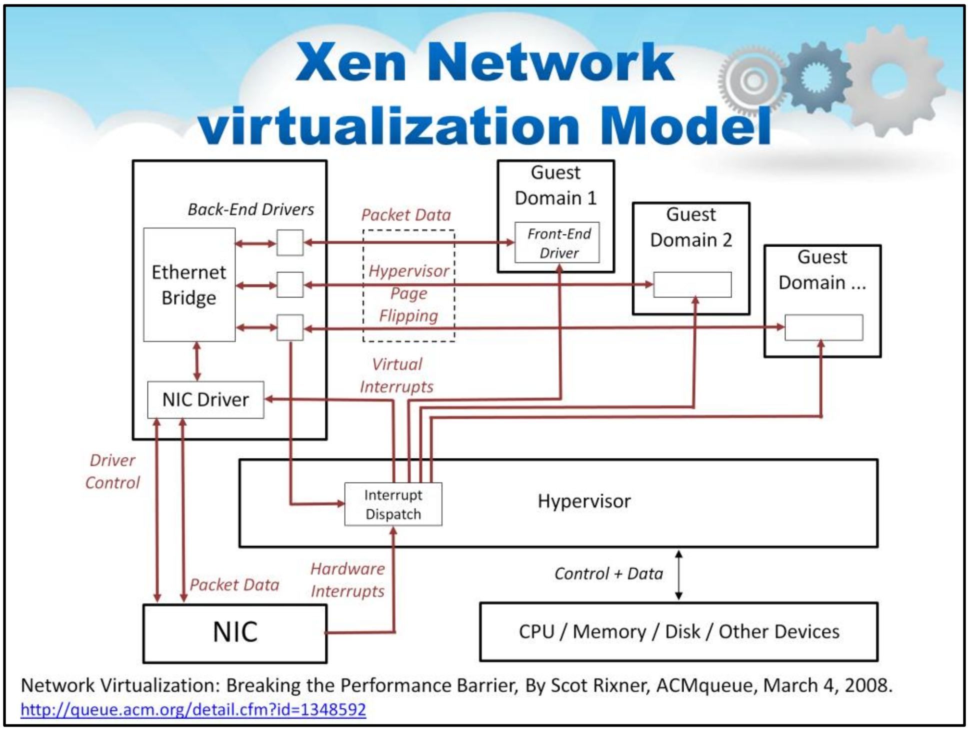


Xen

- Xen is a para-virtualization hypervisor

- Modified network interface drivers communicate with virtual switches in Domain0, which act as TAP in traditional approach
- vSwitch
- Approach to improve performance

CDNA (Concurrent Direct Network Access) hardware adapter



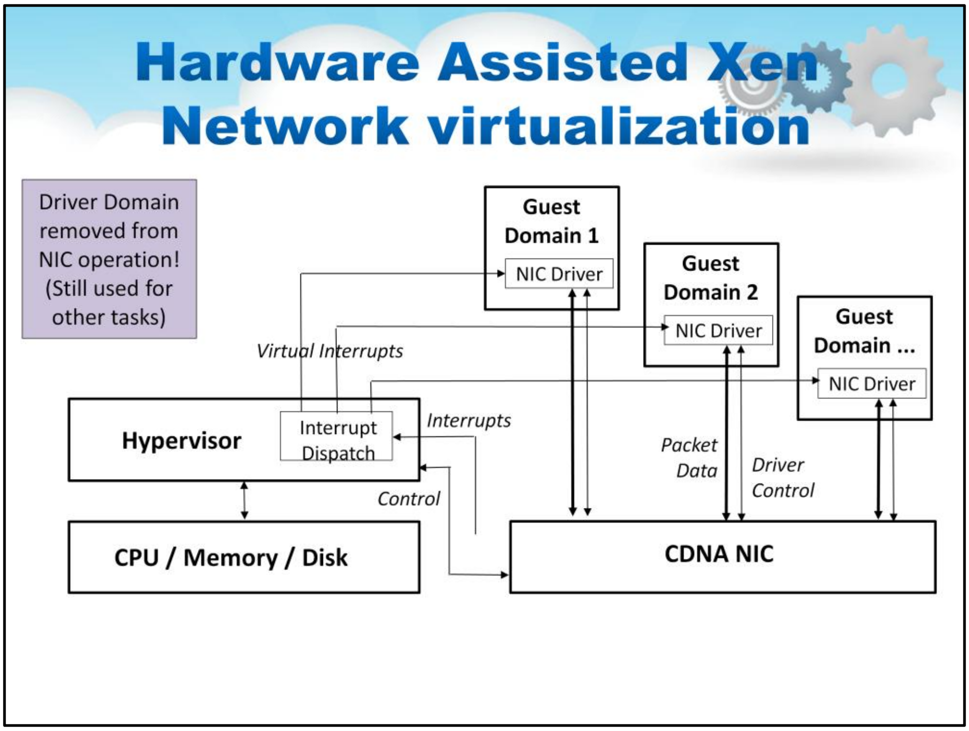
Xen Network virtualization Model

Network Virtualization: Breaking the Performance Barrier, By Scot Rixner, ACMqueue, March 4, 2008.
http://queue.acm.org/detail.cfm?id=1348592

This slide presents an illustration in detail of the Xen network virtualization model.

One can see that in each of the guest domains there is a customized driver which talks to the hypervisor, which then intermediates with the OS to bridge these to the physical NIC.

Most of this complication is due to the fact that the NIC was designed to be used with one OS at a time (the concept of multiple OS's controlling one NIC because of virtualization was not around at the time of the hardware design).



Hardware Assisted Xen Network virtualization

Driver Domain removed from NIC operation! (Still used for other tasks)

Guest Domain 1

NIC Driver

Guest Domain 2

NIC Driver

Guest Domain ...

NIC Driver

Virtual Interrupts

Hypervisor

Interrupt Dispatch

Interrupts

Control

Packet Data

Driver Control

CPU / Memory / Disk

CDNA NIC

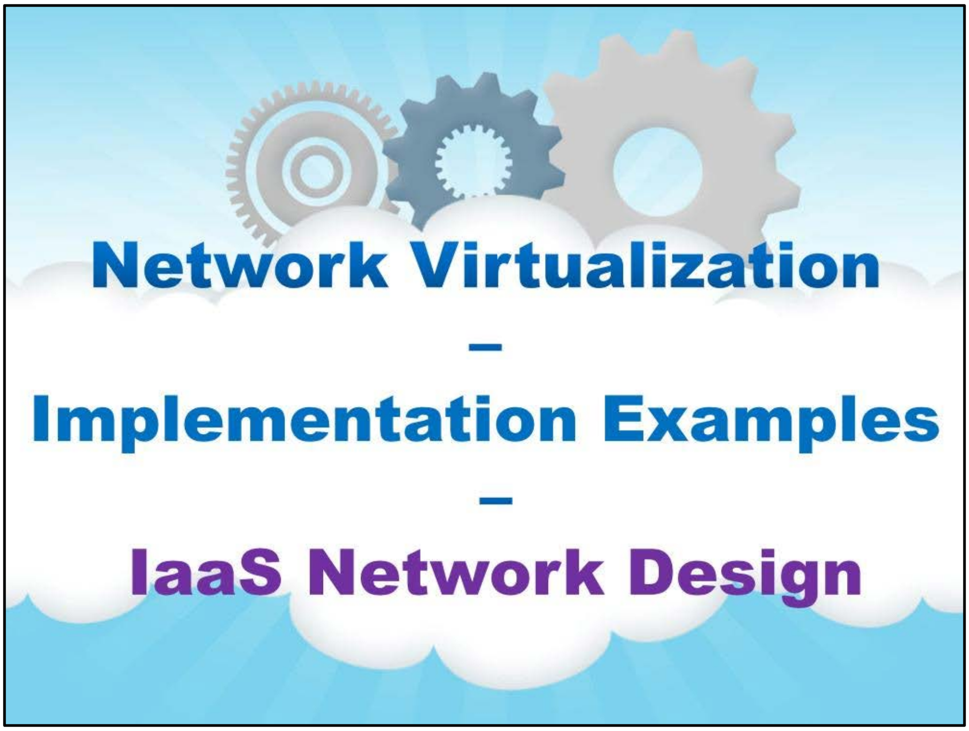
This slide contains an illustration which shows how Hardware Assisted Xen Network Virtualisation works,

The core of this is a re-architected NIC with supports multiple OS's driving it, this is called a CDNA (Concurrent Direct Network Access) hardware adapter

This Significantly improves performance by hardware, and Removes the driver domain from data and interrupts

CDNA (Concurrent Direct Network Access) hardware adapter

- Significantly improves performance by hardware
- Remove driver domain from data and interrupts
- Hypervisor only responsible for virtual interrupts and assigning context to guest OS



Network Virtualization

–

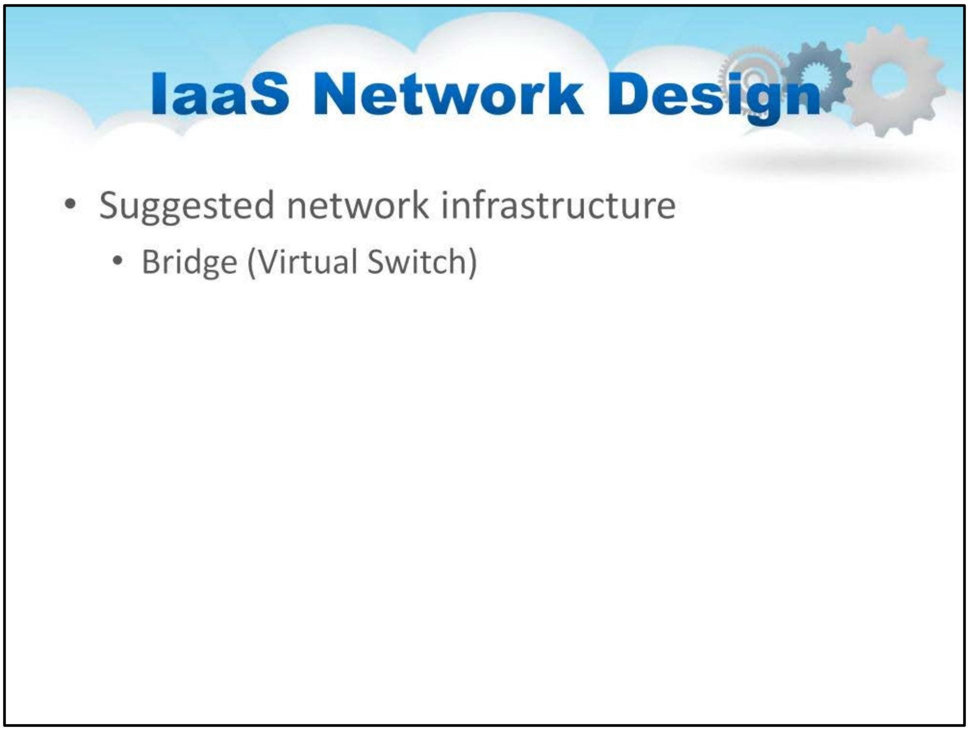
Implementation Examples

–

IaaS Network Design

Now let us put some of these concepts together and look at the networking design of an IaaS cloud.

IaaS Network Design



IaaS Network Design

- Suggested network infrastructure
  - Bridge (Virtual Switch)

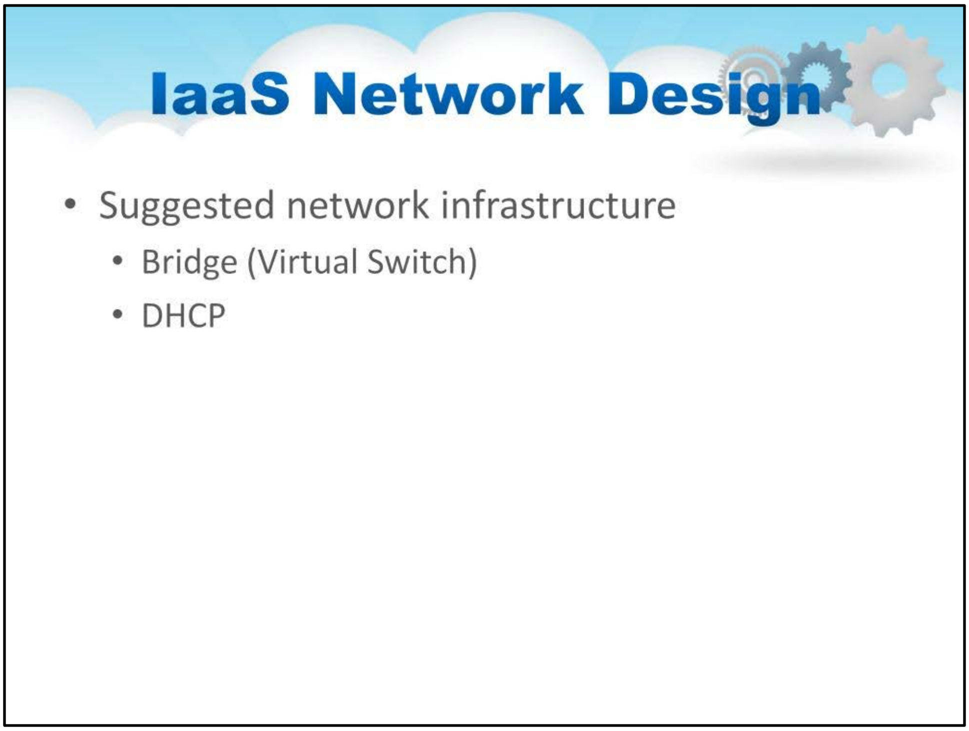
The Physical server has to connect to the physical network, and it does this with physical NICs. There is a bridge from this physical layer to the virtual layer in the hypervisor/at the Operating System level (vSwitch).

DHCP can be used as part if an IPAM (IP address management) strategy from the cloud to the outside to Map virtual MAC addresses of VMs to private IPs in the LAN

NAT Forwards the packages to public network (WAN)

Inside the cloud IPAM is handled by the cloud itself with it’s own IP/MAC mapping table

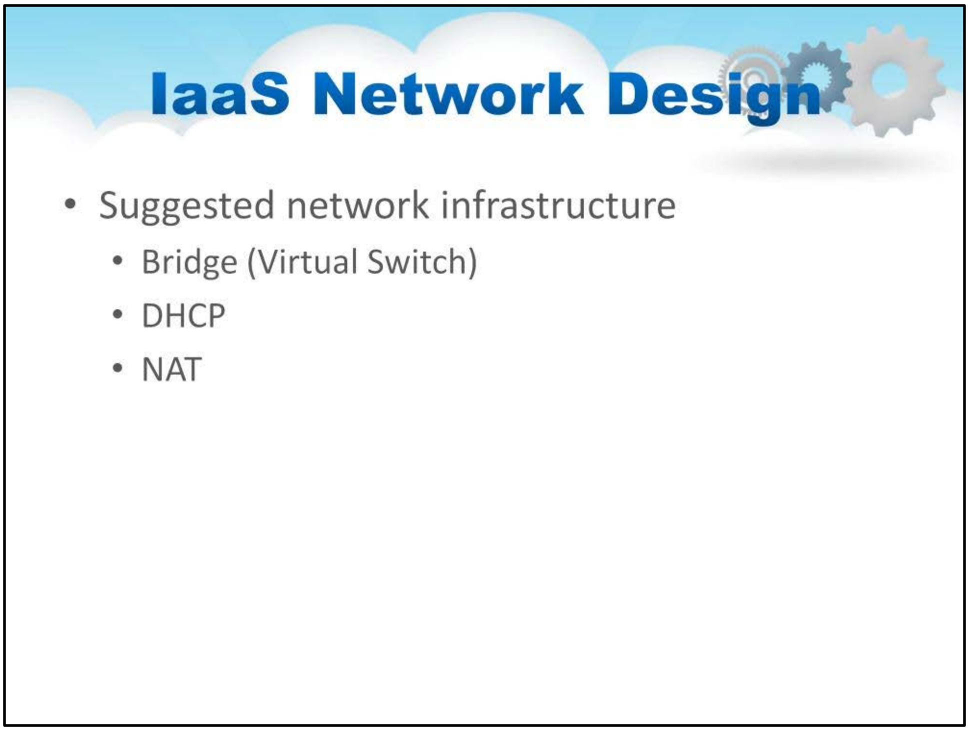
Make virtual machines on one node share physical NICs.



IaaS Network Design

- Suggested network infrastructure
  - Bridge (Virtual Switch)
  - DHCP

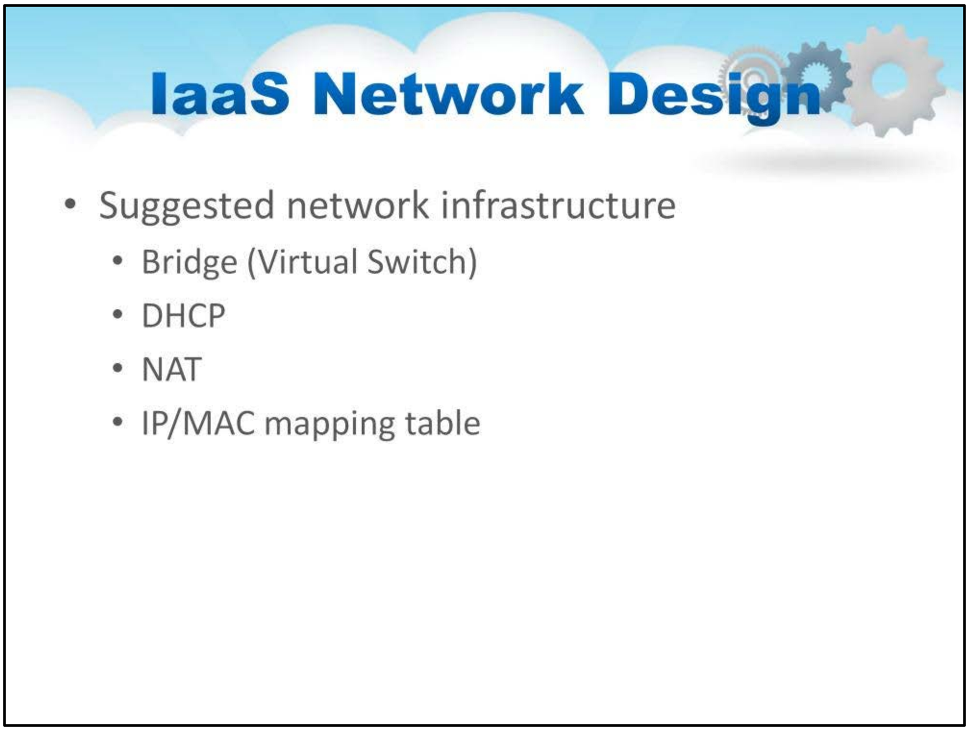
Map virtual MAC addresses of VMs to private IPs in the LAN.



IaaS Network Design

- Suggested network infrastructure
  - Bridge (Virtual Switch)
  - DHCP
  - NAT

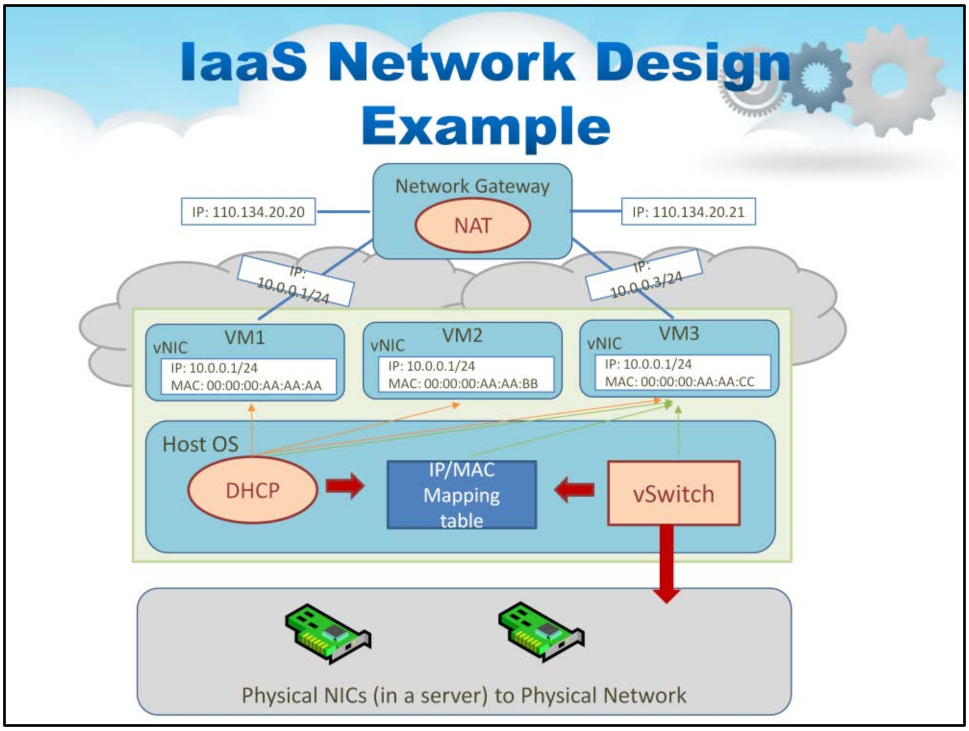
Forward the packages to public network (WAN).



IaaS Network Design

- Suggested network infrastructure
  - Bridge (Virtual Switch)
  - DHCP
  - NAT
  - IP/MAC mapping table

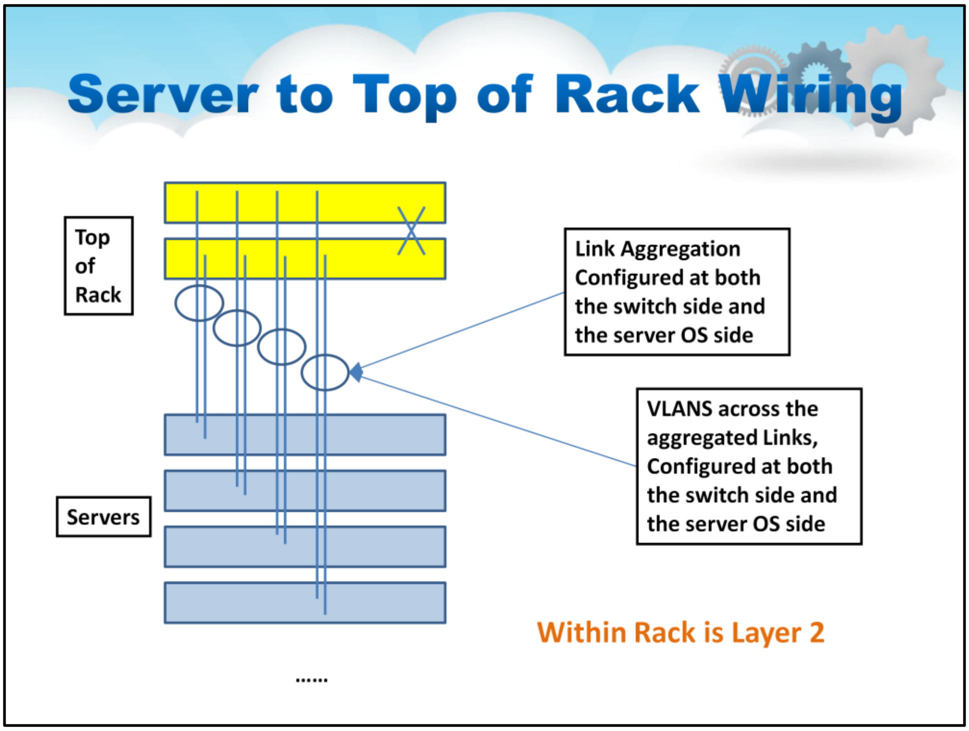
- IP addresses are assigned by the Cloud.
- MAC addresses are assigned by hypervisor.
- This mapping table is maintained by Cloud system.



IaaS Network Design Example

The figure on this slide expands on the previous assrtions and puts them into a diagram. It presents a hypothetical example of IaaS virtual network design. The three VMs are configured with vNICs and their internal IP addresses. They are connected to the local network via physical NICs and can access external or public Internet via Network Address Translation (NAT) protocol that translates an internal IP address into one of the external public IP addresses. More complex virtual network infrastructure can be created by deploying VM-based network devices or virtualising physical network devices. The following are the components of the created IaaS network infrastructure:

vSwitch: Make VMs on one node share physical NICs.
DHCP: Map virtual MAC addresses of VMs to private IPs in the LAN.
NAT: Forward the packages to the public network.
IP/MAC mapping table: maintains IP addresses assigned by CMS and MAC addresses are assigned by hypervisor. This mapping table is maintained by CMS.



Server to Top of Rack Wiring

Top of Rack

Servers

Link Aggregation Configured at both the switch side and the server OS side

VLANS across the aggregated Links, Configured at both the switch side and the server OS side

Within Rack is Layer 2

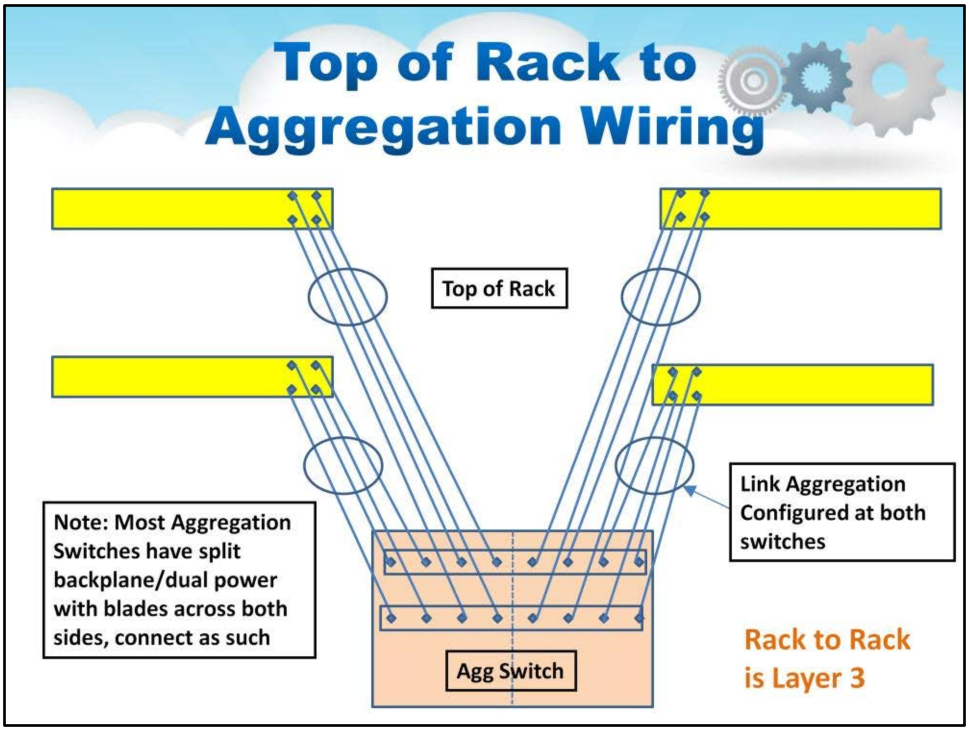
......

Now we can take the ideas from the previous single server environment and show how to construct a larger cloud

Basically the servers are stacked and they are connected to physical top of rack” network as indicated in the illustration in this slide

Note that the two physical NICs in each server (from the previous slide) are each connected to a different top of rack switch. This will increase reliability should a link or a switch fail. They are configured to use link aggregation to work as if they were “bonded” to get the use of the entire bandwidth that has been purchased.

Within the rack the top of rack switches use Layer 2 switching to connect all nodes.



Top of Rack to Aggregation Wiring

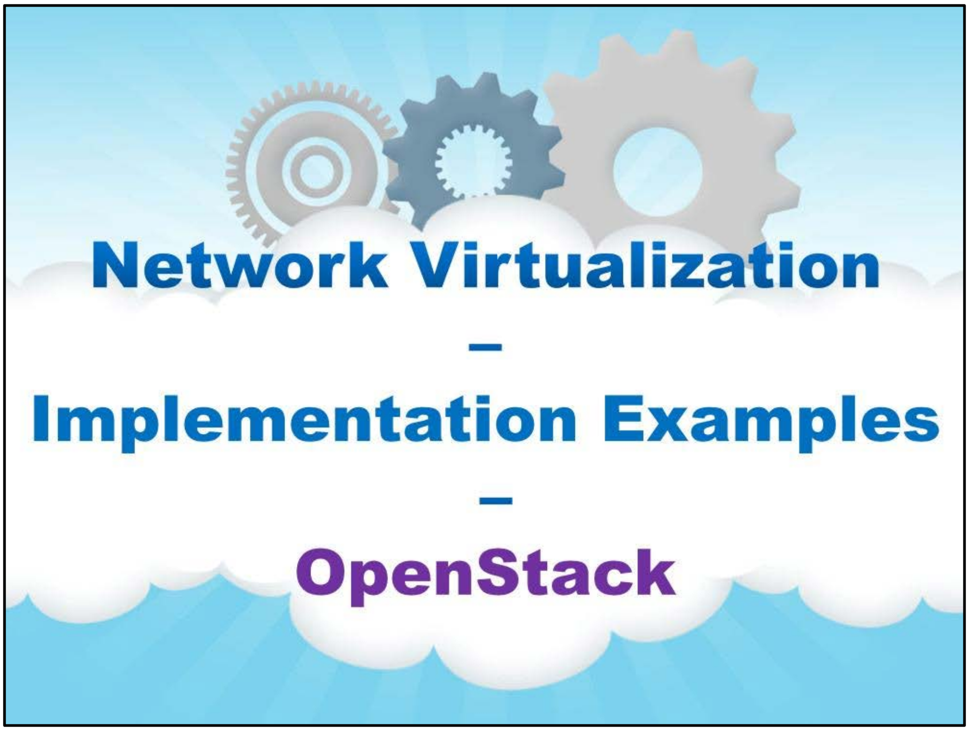
Note: Most Aggregation Switches have split backplane/dual power with blades across both sides, connect as such

Link Aggregation Configured at both switches

Rack to Rack is Layer 3

This slide illustrates how multiple racks are uplinked to an aggregation switch. The uplinks are L3 uplinks. In this example we show 4 uplinks per top of rack switch. Which allows for the hi availability approach of connecting each of the two top of rack switches to both separated backplane and separated blades on the aggregation switch.

Again link aggregation is used for maximum bandwidth and maximum redundancy.



Network Virtualization

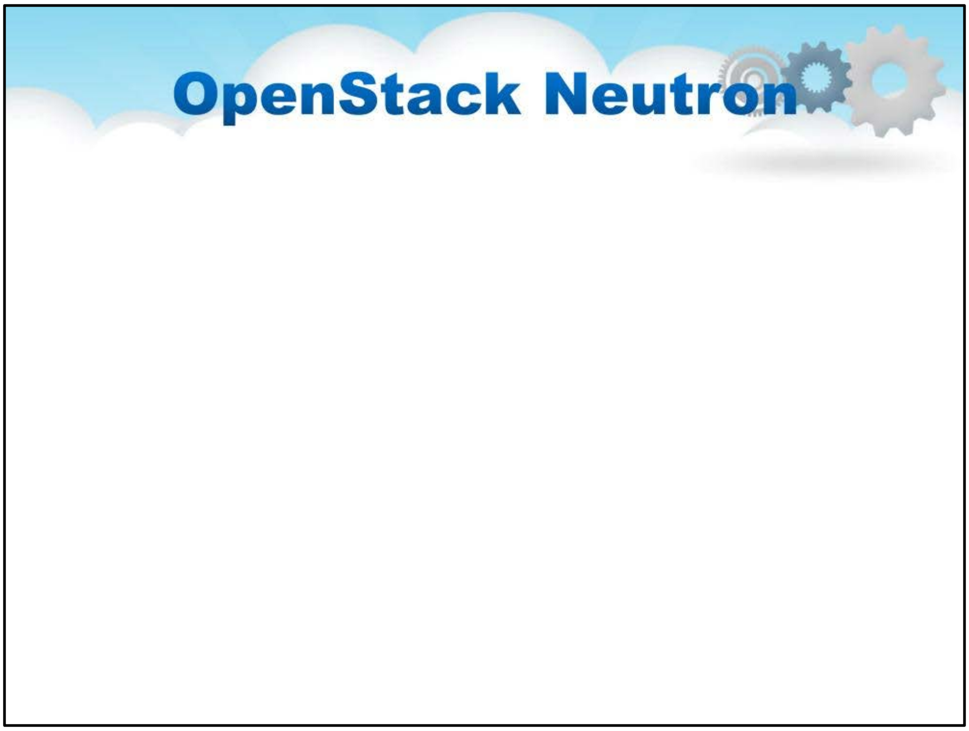
–

Implementation Examples

–

OpenStack

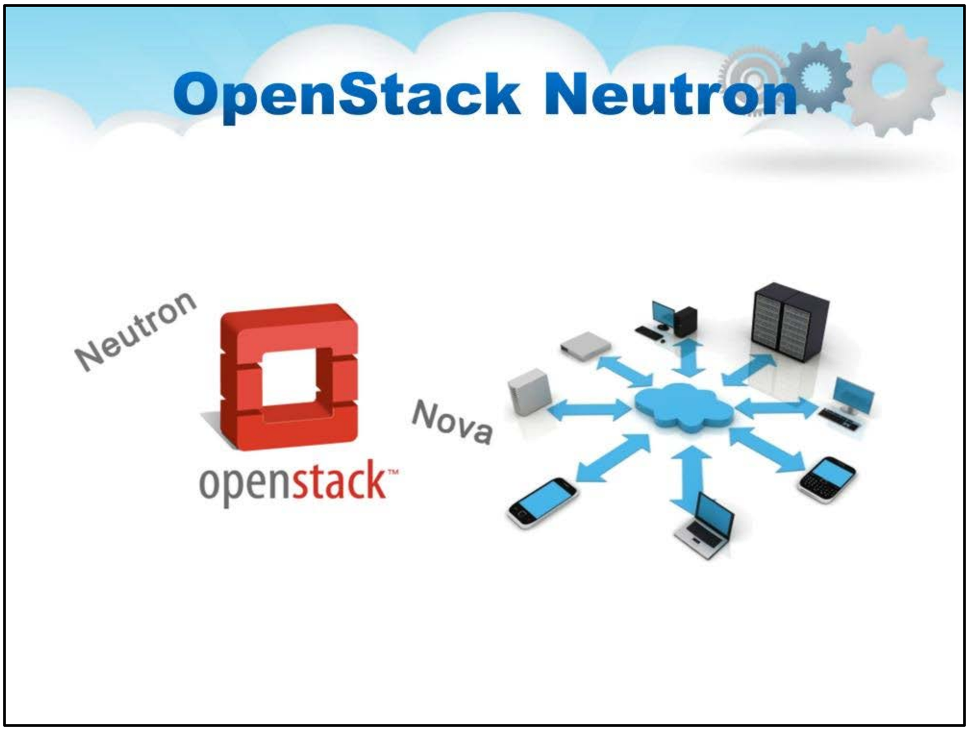
OpenStack



OpenStack Neutron

OpenStack Neutron is a network management component in OpenStack cloud platform

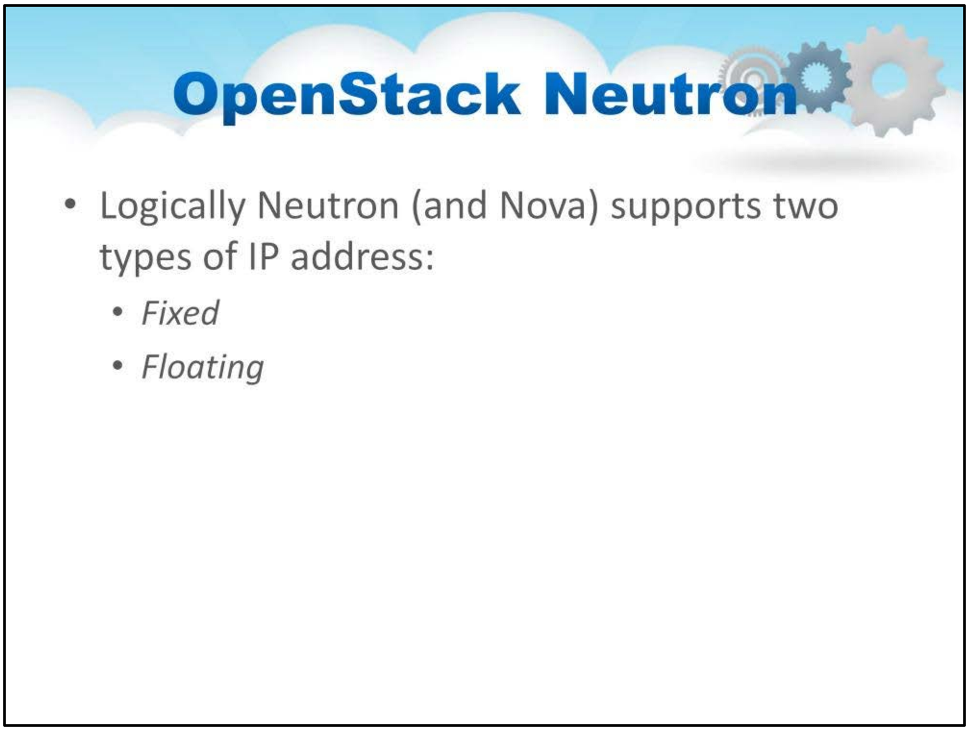
- Works together with the Network Controller in Nova (compute component)



OpenStack Neutron

Neutron network manager provides the following functionalities

- Provides API to build rich networking topologies, and configure advanced network policies
- Enable user developed plugins that introduce advanced network capabilities; e.g. NFV or SDN
- Allows building advanced network services that can be used in the tenant’s Openstack infrastructure

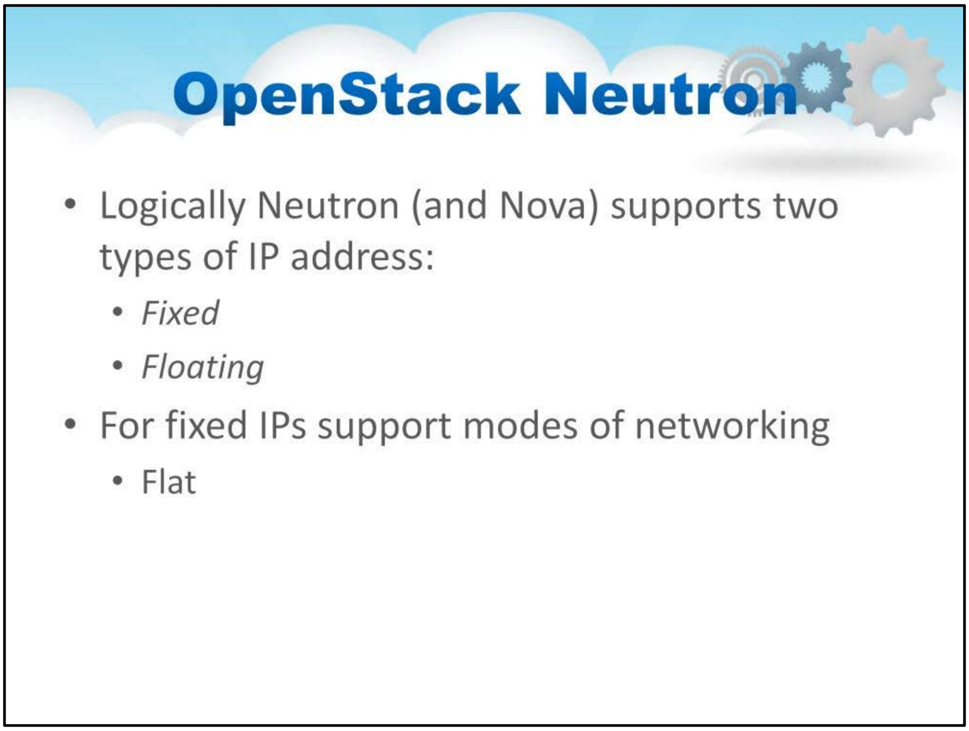


OpenStack Neutron

- Logically Neutron (and Nova) supports two types of IP address:
  - *Fixed*
  - *Floating*

Logically Neutron (and Nova) supports two types of IP address:

- *Fixed* which are associate with virtual machine instance at creation and remain associated till termination
- *Floating* which can be dynamically attached/detached to/from a running VM instance at run-time

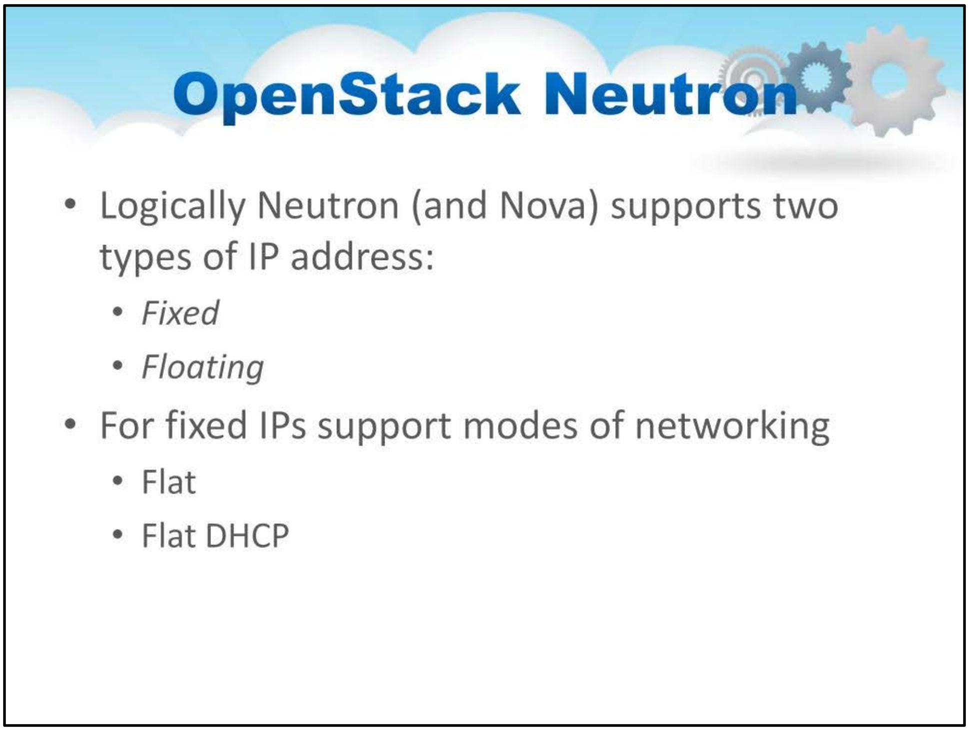


OpenStack Neutron

- Logically Neutron (and Nova) supports two types of IP address:
  - *Fixed*
  - *Floating*
- For fixed IPs support modes of networking
  - Flat

For fixed IPs, Neutron (and Nova) support following three modes of networking

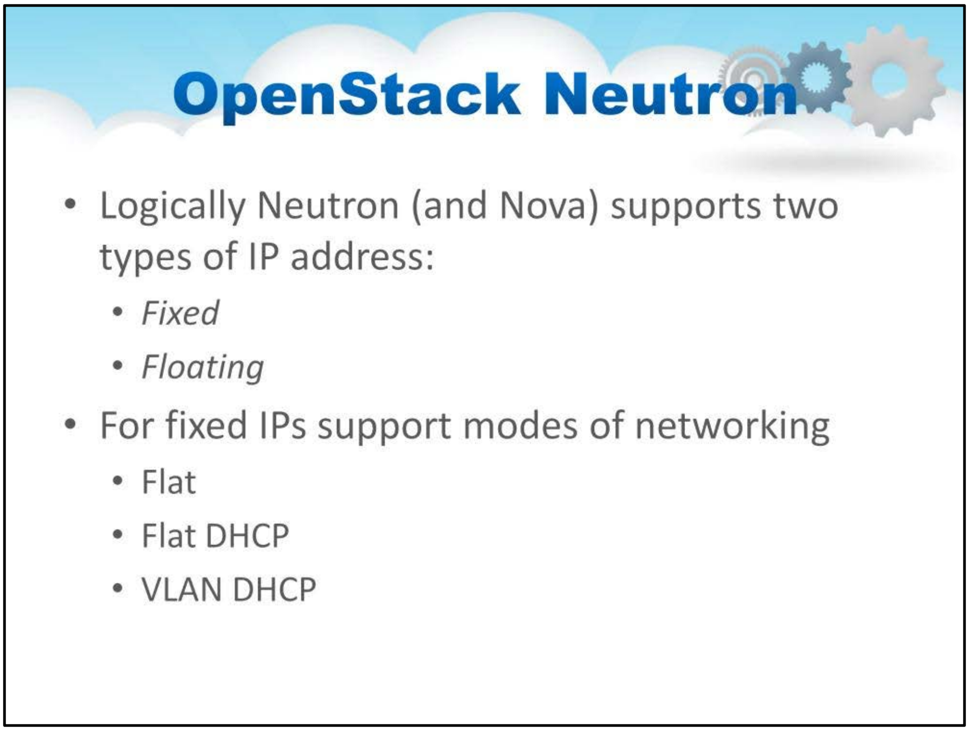
- *Flat* mode provides each VM instance with a fixed IP associated with a default network bridge that must be manually configured



OpenStack Neutron

- Logically Neutron (and Nova) supports two types of IP address:
  - *Fixed*
  - *Floating*
- For fixed IPs support modes of networking
  - Flat
  - Flat DHCP

Flat DHCP mode improves upon Flat mode by creating a DHCP server to provide fixed IPs to virtual machine instances

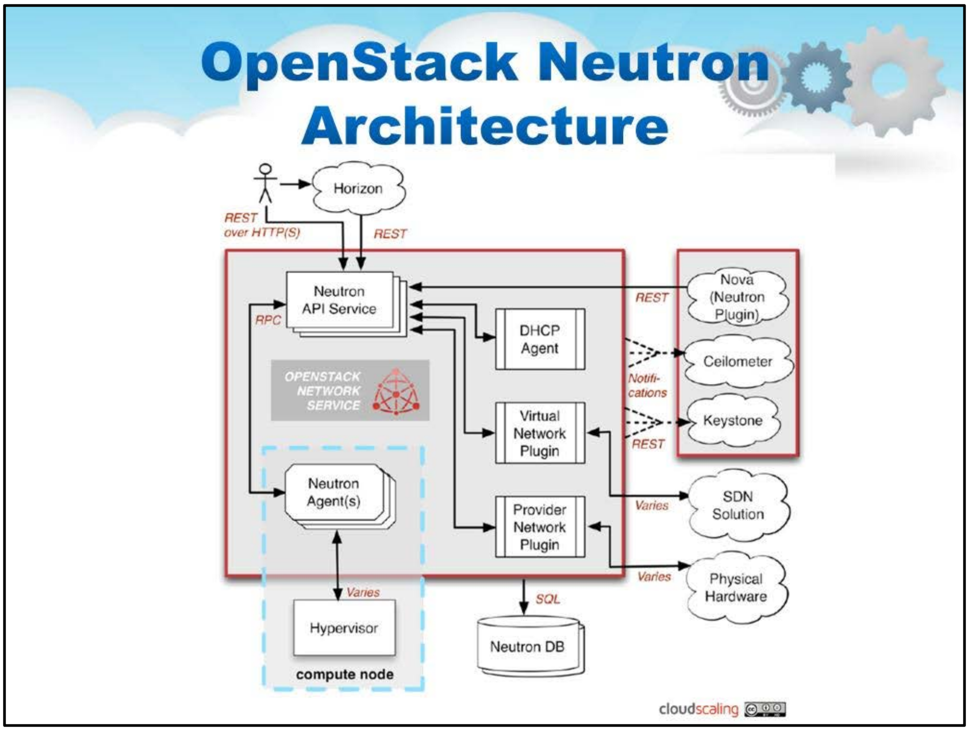


OpenStack Neutron

- Logically Neutron (and Nova) supports two types of IP address:
  - *Fixed*
  - *Floating*
- For fixed IPs support modes of networking
  - Flat
  - Flat DHCP
  - VLAN DHCP

VLAN DHCP mode is the default networking mode in which Nova creates a VLAN and
virtual bridge for each project

- VM instances in the project are allocated a private IP address from range of IPs.
- Users can access these instances by using a special VPN instance called 'cloudpipe' which uses a certificate and key to create a VPN (Virtual Private Network).



OpenStack Neutron Architecture

OpenStack Networking (Neutron, formerly Quantum) is a pluggable, scalable and API-driven system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

OpenStack Networking is a system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

OpenStack Neutron provides networking models for different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic. OpenStack Networking manages IP addresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically re routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure. Users can create their own networks, control traffic and connect servers and devices to one or more networks. Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale. OpenStack Networking has an extension framework allowing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.

Networking Capabilities

OpenStack provides flexible networking models to suit the needs of different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic.

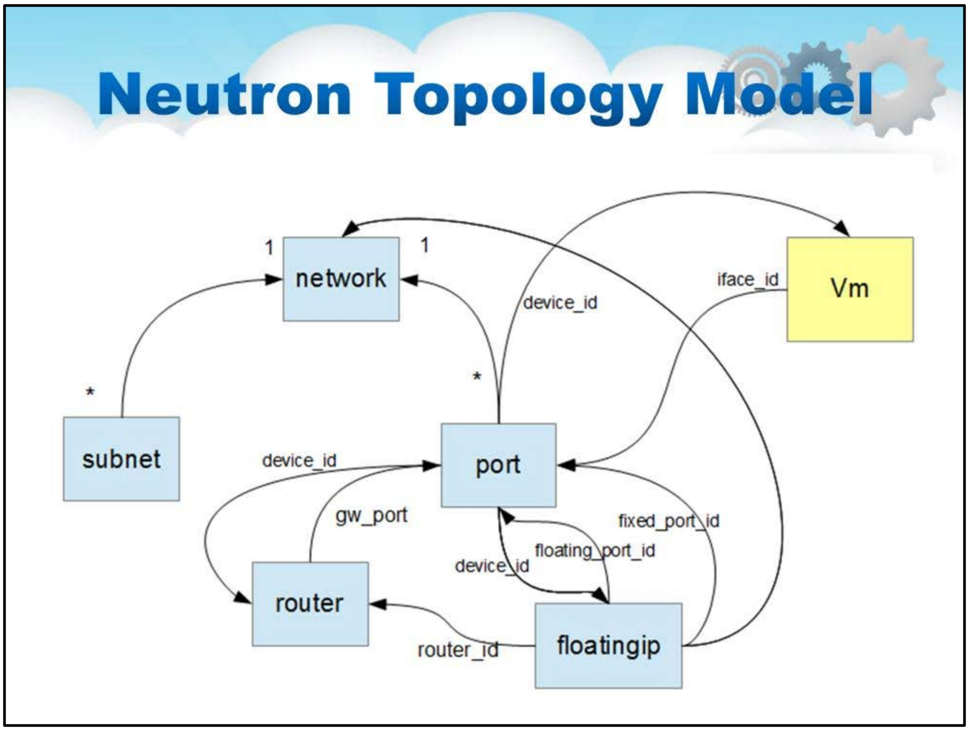
OpenStack Networking manages IP addresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically re-routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure.

Users can create their own networks, control traffic and connect servers and devices to one or more networks.

The pluggable backend architecture lets users take advantage of commodity gear or advanced networking services from supported vendors.

Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale.

OpenStack Networking has an extension framework allowing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.



Neutron Topology Model

Neutron provides "network connectivity as a service" between interface devices managed by other OpenStack services (most likely Nova). The service works by allowing users to create their own networks and then attach interfaces to them. Like many of the OpenStack services, Neutron is highly configurable due to its plug-in architecture. These plug-ins accommodate different networking equipment and software. As such, the architecture and deployment can vary dramatically.

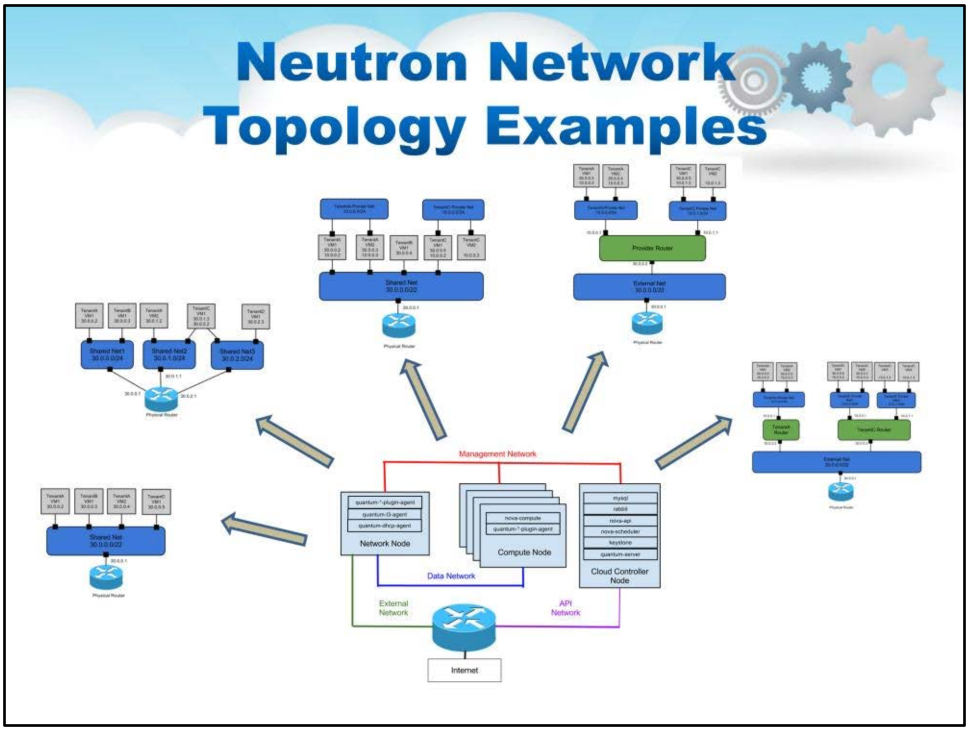
neutron-server accepts API requests and then routes them to the appropriate Neutron plug-in for action.

Neutron plug-ins and agents perform the actual actions such as plugging and unplugging ports, creating networks or subnets and IP addressing. These plug-ins and agents differ depending on the vendor and technologies used in the particular cloud. Neutron ships with plug-ins and agents for: Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, the Ryu Network Operating System, and VMware NSX.

The common agents are L3 (layer 3), DHCP (dynamic host IP addressing) and the specific plug-in agent.

Most Neutron installations will also make use of a messaging queue to route information between the neutron-server and various agents as well as a database to store networking state for particular plug-ins.

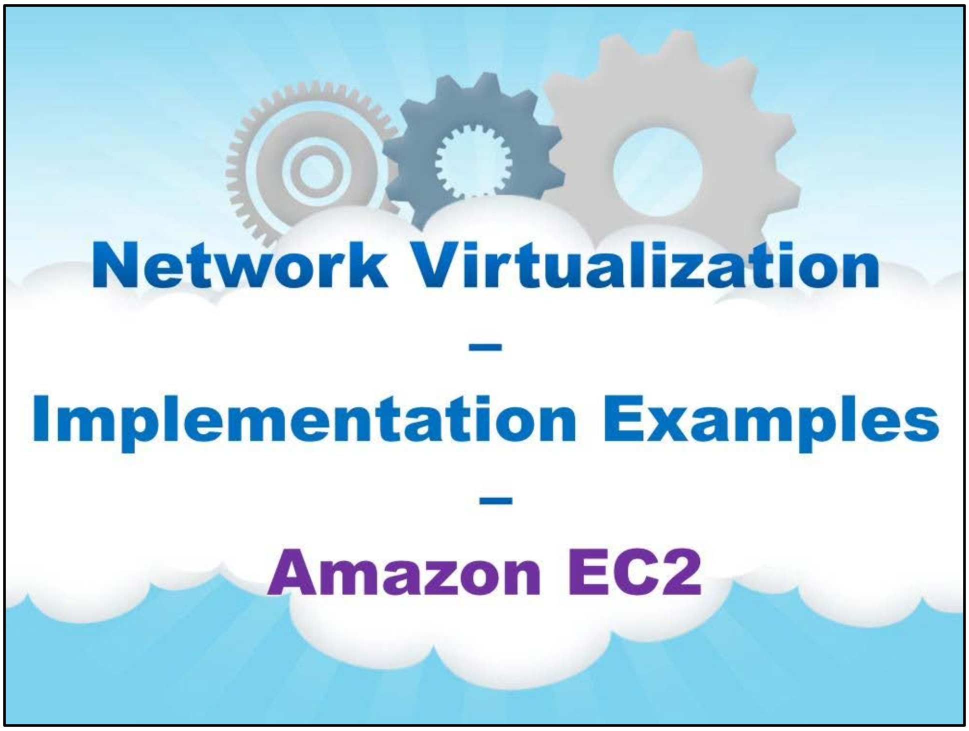
Neutron will interact mainly with Nova, where it will provide networks and connectivity for its instances.



Neutron Network Topology Examples

This slide illustrates how Neutron based software networking can be applied to result is many different networking topologies for the user

Note the ability to create virtual switch components as well as multiple network segments.



Network Virtualization

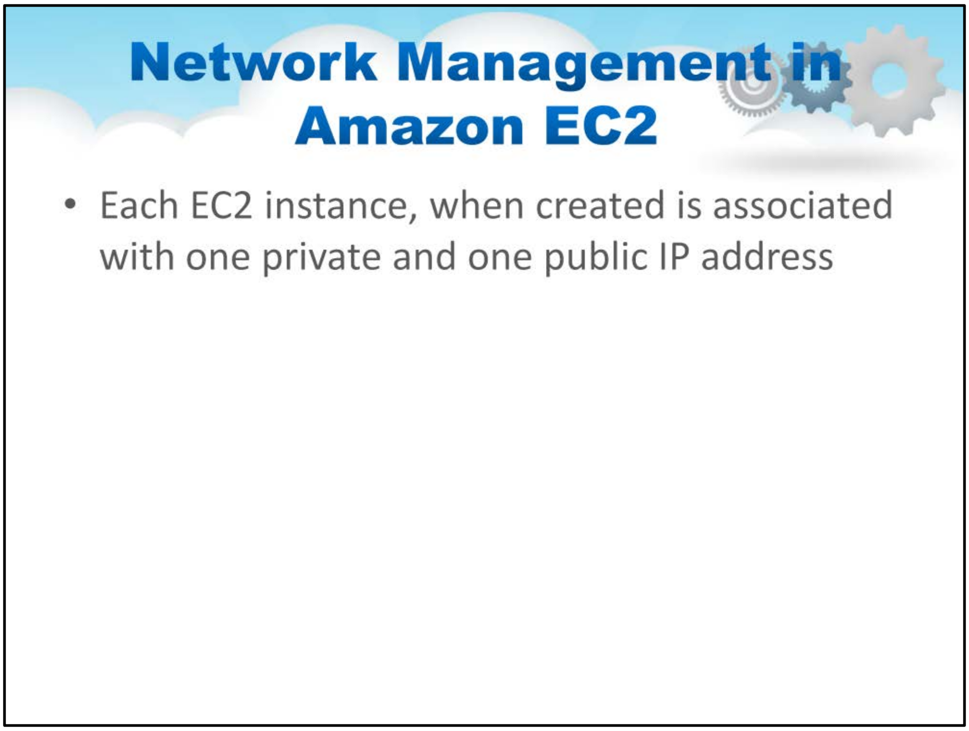
–

Implementation Examples

–

Amazon EC2

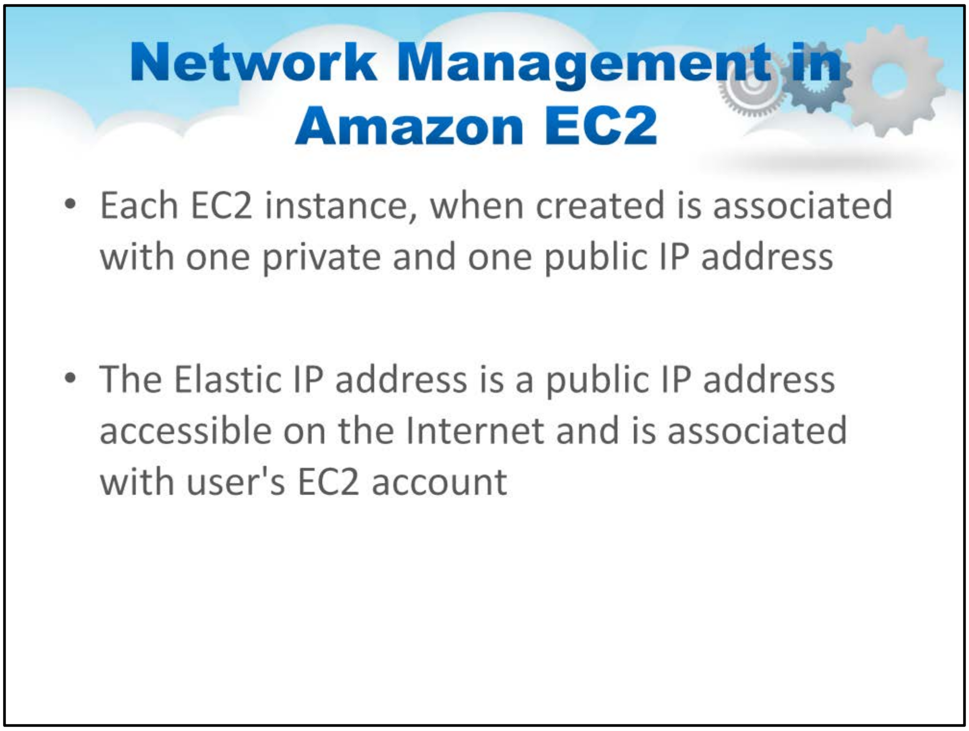
Amazon EC2



Network Management in Amazon EC2

- Each EC2 instance, when created is associated with one private and one public IP address

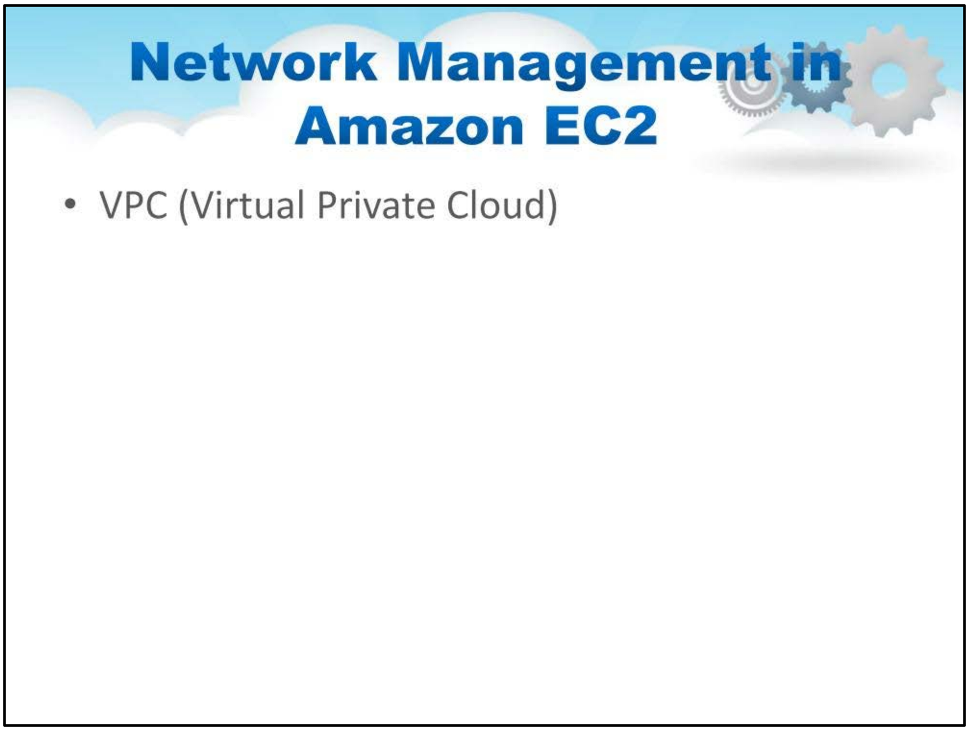
- Private IP address is used within the EC2 LAN (Local Area Network)
- Public IP address is used to access the user VM over the Internet
- Public IP address is advertised to the Internet and has a 1:1 NAT (Network Address Translation) mapping to the private IP address of the EC2 instance



Network Management in Amazon EC2

- Each EC2 instance, when created is associated with one private and one public IP address
- The Elastic IP address is a public IP address accessible on the Internet and is associated with user's EC2 account

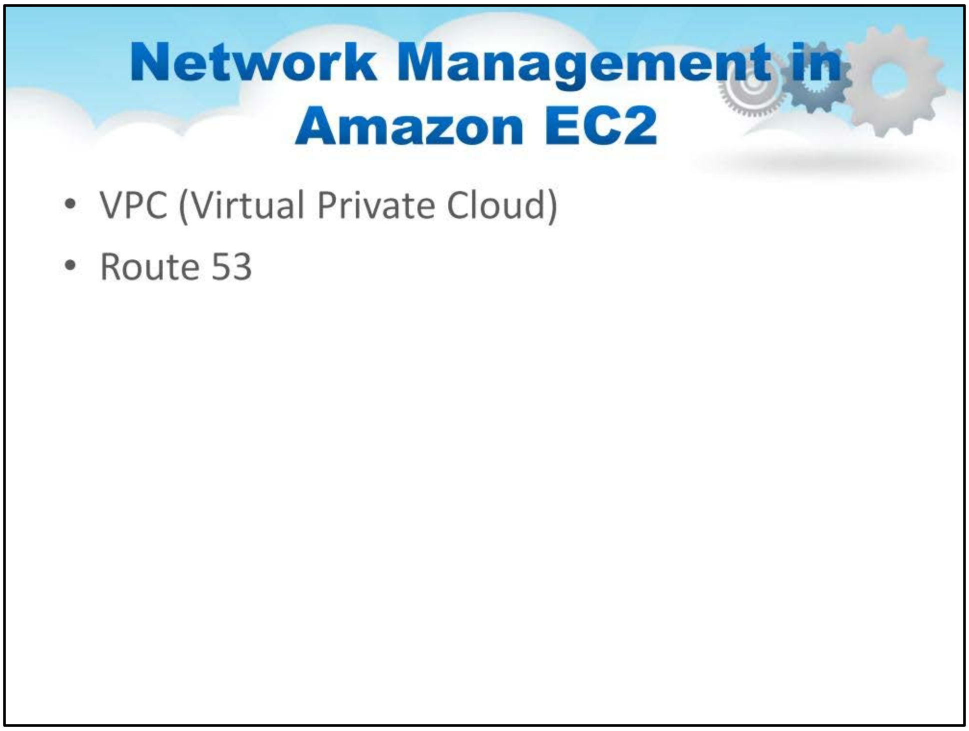
- A user can associate this IP address to any particular EC2 instance rented with that account
- The mapping can be changed dynamically to a different EC2 instance, in particular to a replacement instance in the event of a failure of the running VM



Network Management in Amazon EC2

- VPC (Virtual Private Cloud)

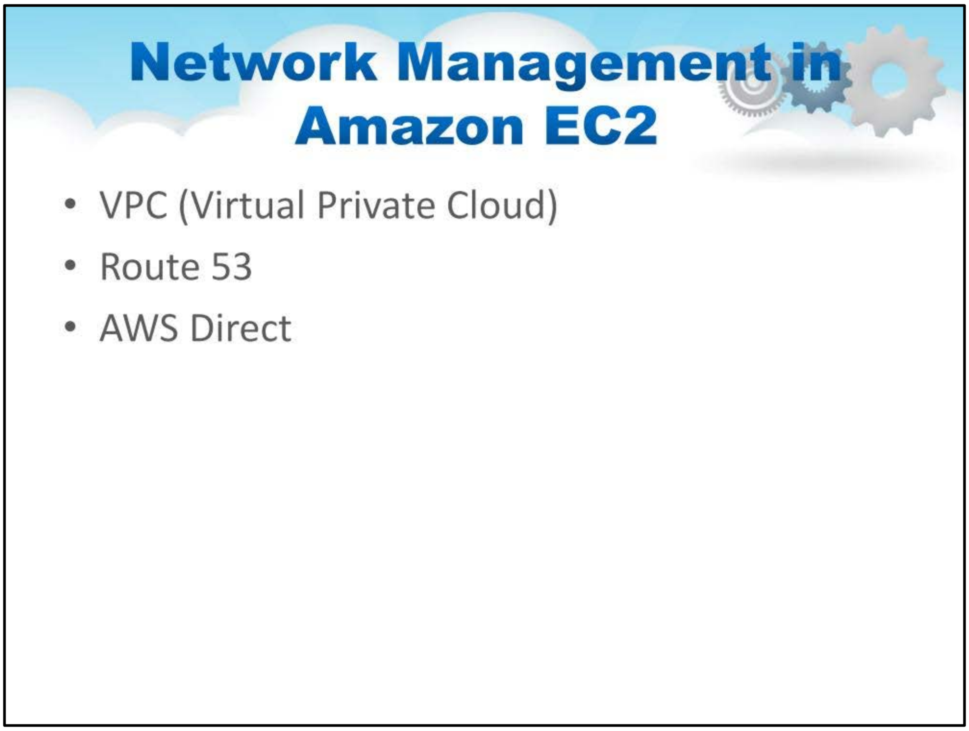
VPC (Virtual Private Cloud) allows organizations to use AWS resources along with their existing infrastructure connecting AWS VPC and organisation’s network via VPN



Network Management in Amazon EC2

- VPC (Virtual Private Cloud)
- Route 53

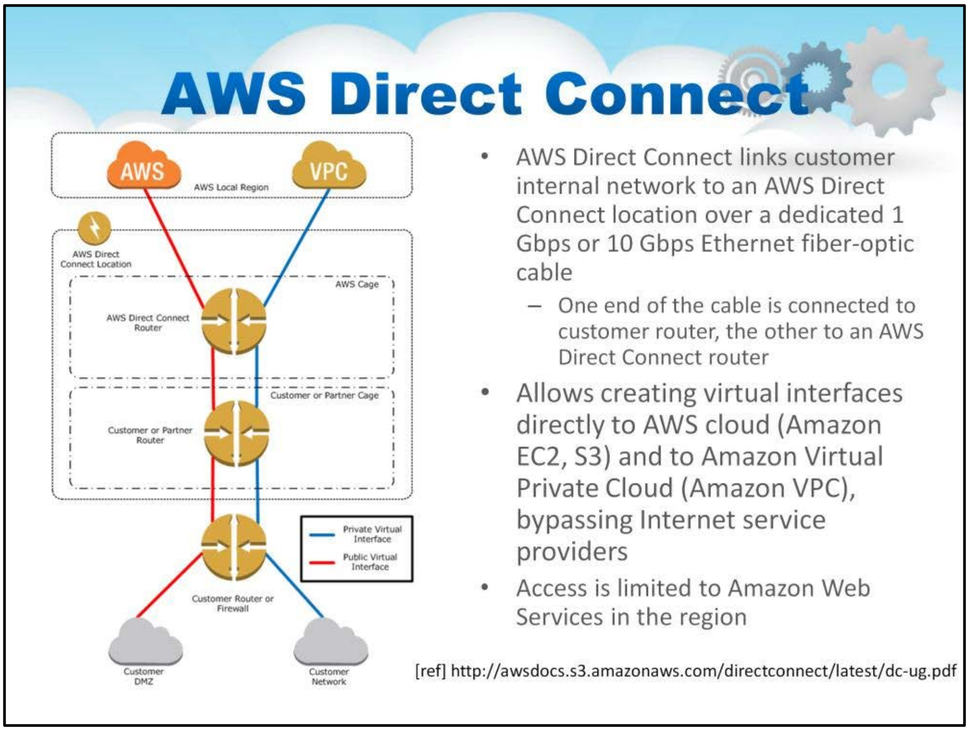
Amazon Route 53 is a DNS (Domain Name Server) service provided by Amazon to map EC2 instance IP addresses to a domain names.



Network Management in Amazon EC2

- VPC (Virtual Private Cloud)
- Route 53
- AWS Direct

AWS Direct connect for dedicated connectivity to AWS cloud



AWS Direct Connect

AWS
VPC
AWS Local Region
AWS Direct Connect Location
AWS Cage
AWS Direct Connect Router
Customer or Partner Cage
Customer or Partner Router
Private Virtual Interface
Public Virtual Interface
Customer Router or Firewall
Customer DMZ
Customer Network

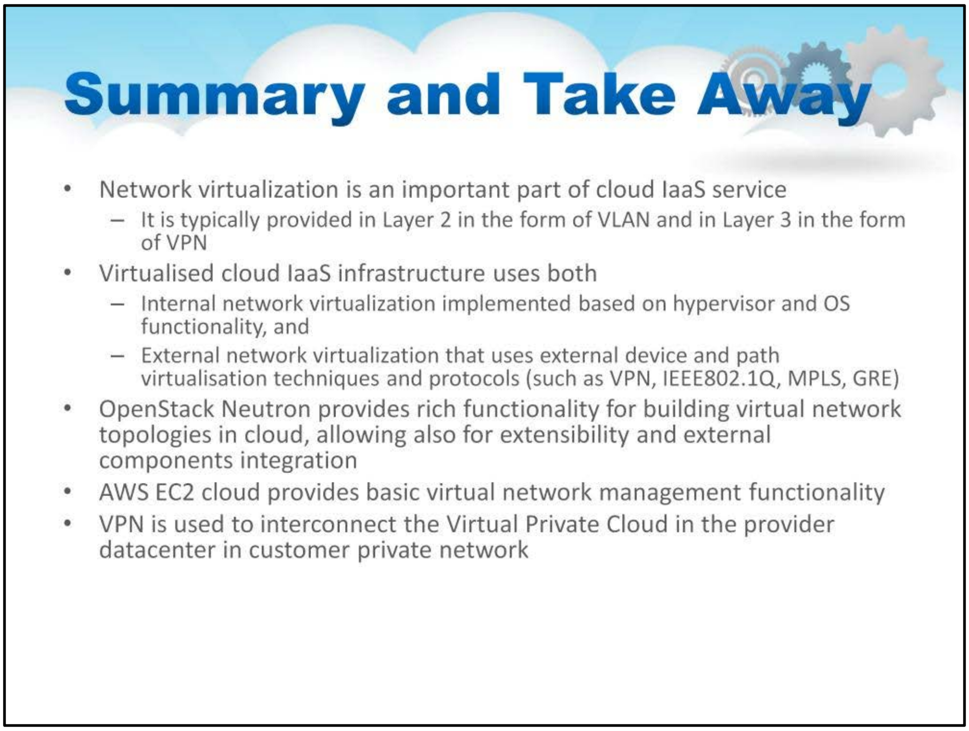
- AWS Direct Connect links customer internal network to an AWS Direct Connect location over a dedicated 1 Gbps or 10 Gbps Ethernet fiber-optic cable
  - One end of the cable is connected to customer router, the other to an AWS Direct Connect router
- Allows creating virtual interfaces directly to AWS cloud (Amazon EC2, S3) and to Amazon Virtual Private Cloud (Amazon VPC), bypassing Internet service providers
- Access is limited to Amazon Web Services in the region

[ref] http://awsdocs.s3.amazonaws.com/directconnect/latest/dc-ug.pdf

AWS offers yet another option for more intimately connecting to the AWS cloud. Let's assume that your own company datacentre or at least the part of your infrastructure which needs the overflow cloud-bursting" type of capability talked about in the previous slide, is really co-location space in a public datacentre. Lets further assume AWS has some of their infrastructure in exactly that same datacentre. This is not so coincidental because the brand-name mega datacenters in places like New York, Washington DC, Silicon Valley, Tokyo, and London are "the place to be" and those are places where AWS is as well. So if your syste is in the same datacetner as AWS why
not simply connect to them directly? That is what AWS direct connect is, One end of the cable is connected to customer router, the other to an AWS Direct Connect router. At an Ethernet level a VLAN is established between the two routers. Layer 3 is the connectivity which Direct Connect uses and it is like one ISP peering with another ISP, that is they both have Autonomous System (AS) numbers and use the Border Gateway Protocol (BGP) for Layer 3 IP routing. The public Interntet (or the rest of the in-datacenter IP exchange, for that matter) is completely bypassed.

AWS has Direct Connect in most of regions

CoreSite NY1 & NY2 in US East (Virginia)



Summary and Take Away

- Network virtualization is an important part of cloud IaaS service
  - It is typically provided in Layer 2 in the form of VLAN and in Layer 3 in the form of VPN
- Virtualised cloud IaaS infrastructure uses both
  - Internal network virtualization implemented based on hypervisor and OS functionality, and
  - External network virtualization that uses external device and path virtualisation techniques and protocols (such as VPN, IEEE802.1Q, MPLS, GRE)
- OpenStack Neutron provides rich functionality for building virtual network topologies in cloud, allowing also for extensibility and external components integration
- AWS EC2 cloud provides basic virtual network management functionality
- VPN is used to interconnect the Virtual Private Cloud in the provider datacenter in customer private network

In this Lesson, we have covered

Network virtualization is an important part of cloud IaaS service

It is typically provided in Layer 2 in the form of VLAN and in Layer 3 in the form of VPN

Virtualised cloud IaaS infrastructure uses both

Internal network virtualization implemented based on hypervisor and OS functionality, and

External network virtualization that uses external device and path virtualisation techniques and protocols (such as VPN, IEEE802.1Q, MPLS, GRE)

OpenStack Neutron provides rich functionality for building virtual network topologies in cloud, allowing also for extensibility and external components integration

AWS EC2 cloud provides basic virtual network management functionality

VPN is used to interconnect the Virtual Private Cloud in the provider datacenter in customer private network

Cloud Computing

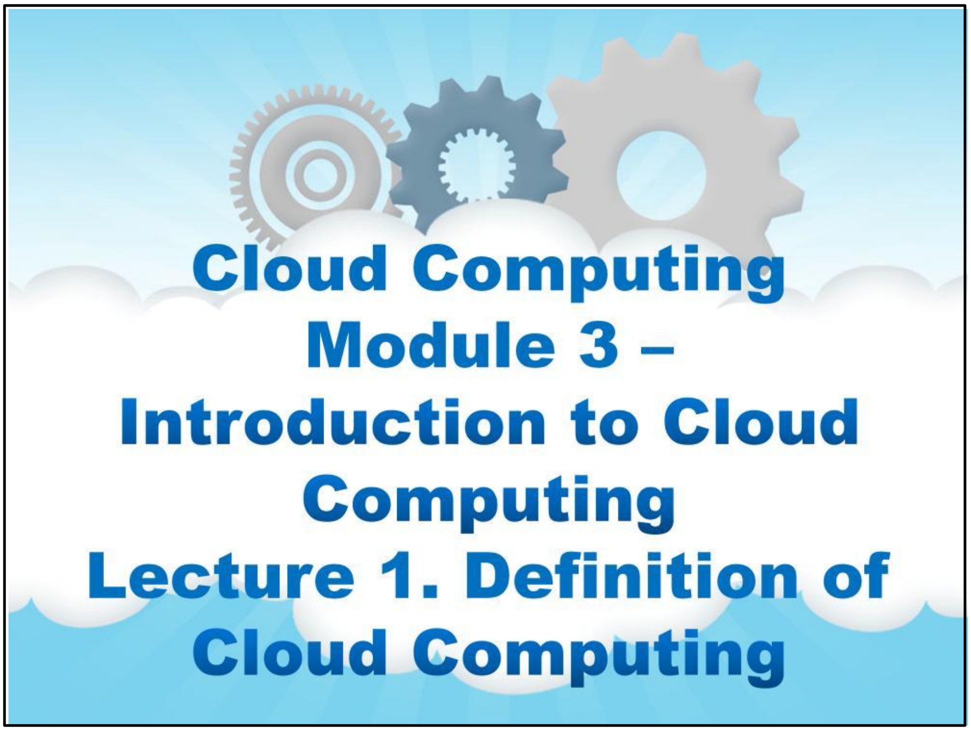
Lecture Manual

Volume 3

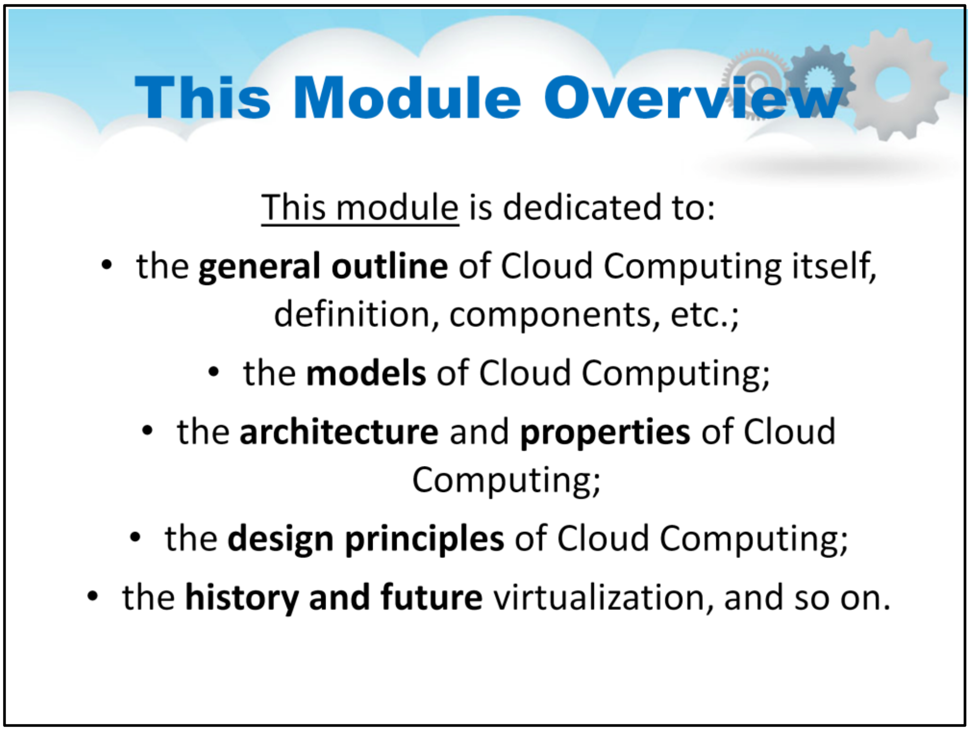
Module 3

Introduction to Cloud Computing

Content



Cloud Computing Module 3 – Introduction to Cloud Computing Lecture 1. Definition of Cloud Computing

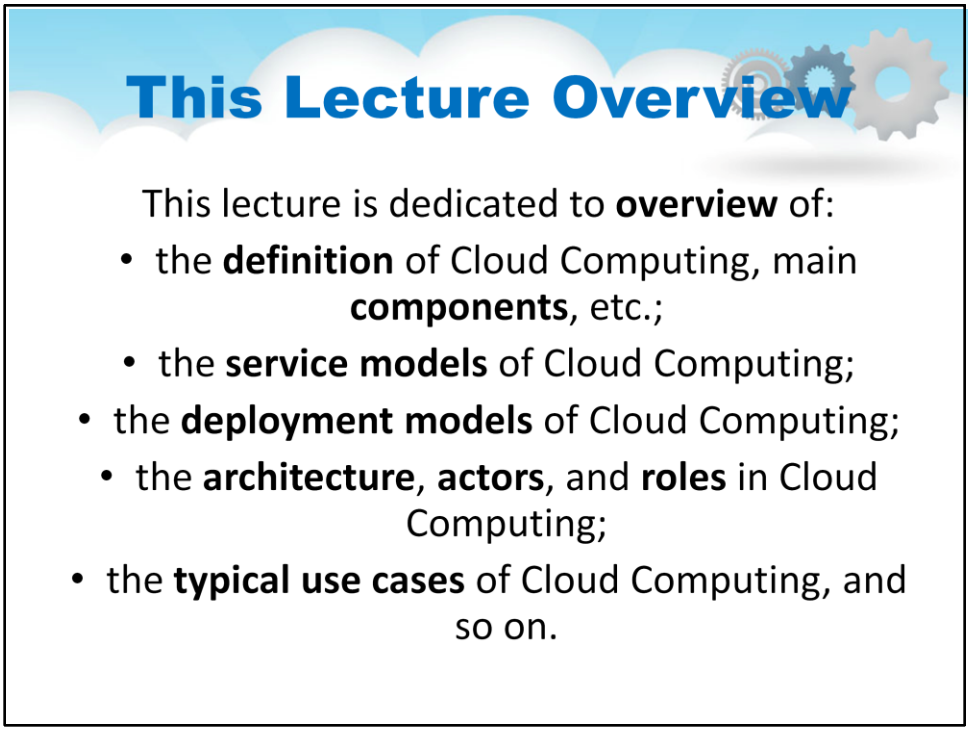


This Module Overview

This module is dedicated to:

- the **general outline** of Cloud Computing itself, definition, components, etc.;
- the **models** of Cloud Computing;
- the **architecture** and **properties** of Cloud Computing;
- the **design principles** of Cloud Computing;
- the **history and future** virtualization, and so on.

Module 3. Introduction to Cloud Computing



This Lecture Overview

This lecture is dedicated to **overview** of:

- the **definition** of Cloud Computing, main **components**, etc.;
- the **service models** of Cloud Computing;
- the **deployment models** of Cloud Computing;
- the **architecture**, **actors**, and **roles** in Cloud Computing;
- the **typical use cases** of Cloud Computing, and so on.

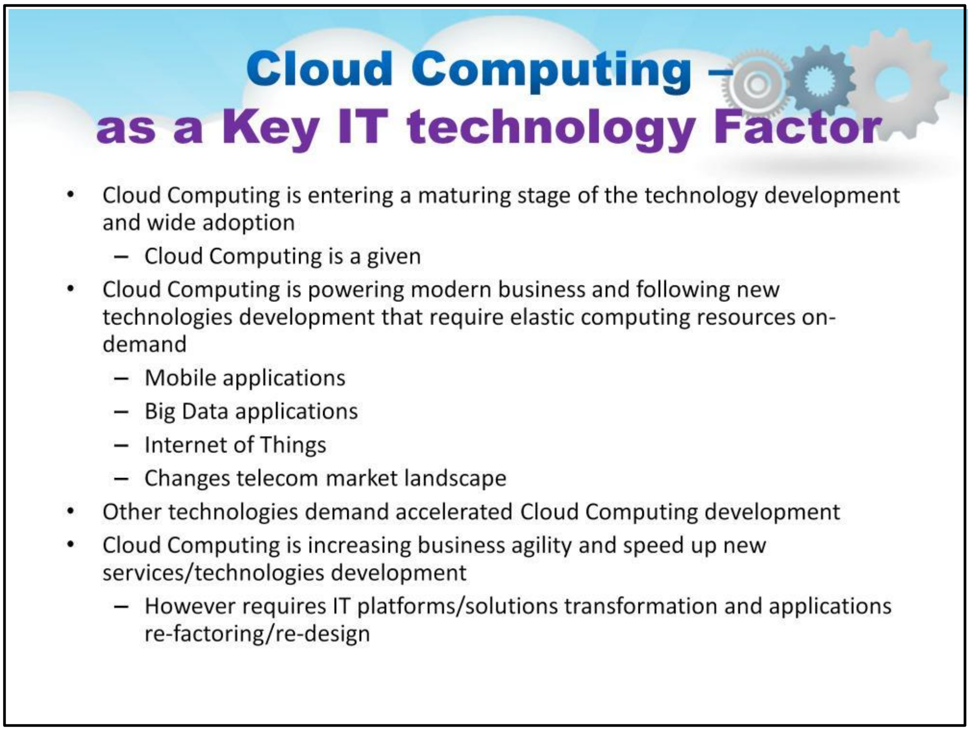
Lecture 1. Definition of Cloud Computing

Cloud Computing

–

Overview

Overview



Cloud Computing – as a Key IT technology Factor

- Cloud Computing is entering a maturing stage of the technology development and wide adoption
  - Cloud Computing is a given
- Cloud Computing is powering modern business and following new technologies development that require elastic computing resources on-demand
  - Mobile applications
  - Big Data applications
  - Internet of Things
  - Changes telecom market landscape
- Other technologies demand accelerated Cloud Computing development
- Cloud Computing is increasing business agility and speed up new services/technologies development
  - However requires IT platforms/solutions transformation and applications re-factoring/re-design

First of all, Cloud Computing is a key technology factor of Information Technology industry.

Cloud Computing is entering a maturing stage of the technology development and wide adoption

- Cloud Computing is already a given entity.

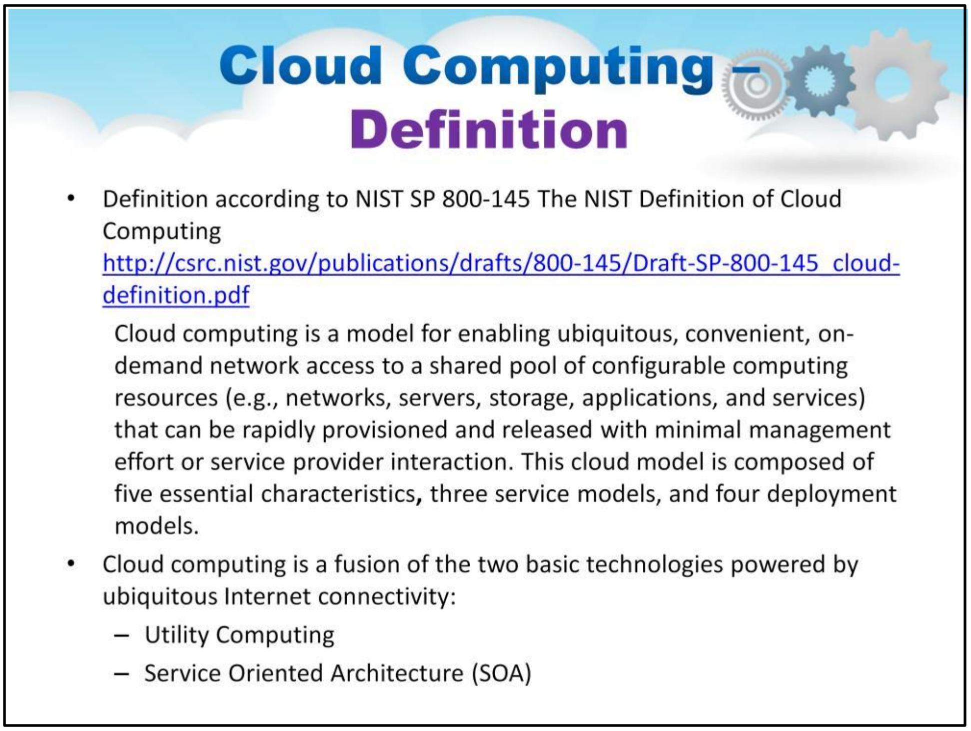
Cloud Computing is powering modern business and following new technologies development that require elastic computing resources on-demand

- Mobile applications
- Big Data applications
- Internet of Things
- Changes telecom market landscape

Other technologies demand accelerated Cloud Computing development

Cloud Computing is increasing business agility and speed up new services/technologies development

- However requires IT platforms/solutions transformation and applications re-factoring/re-design



Cloud Computing – Definition

- Definition according to NIST SP 800-145 The NIST Definition of Cloud Computing http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf

  Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.
- Cloud computing is a fusion of the two basic technologies powered by ubiquitous Internet connectivity:
  - Utility Computing
  - Service Oriented Architecture (SOA)

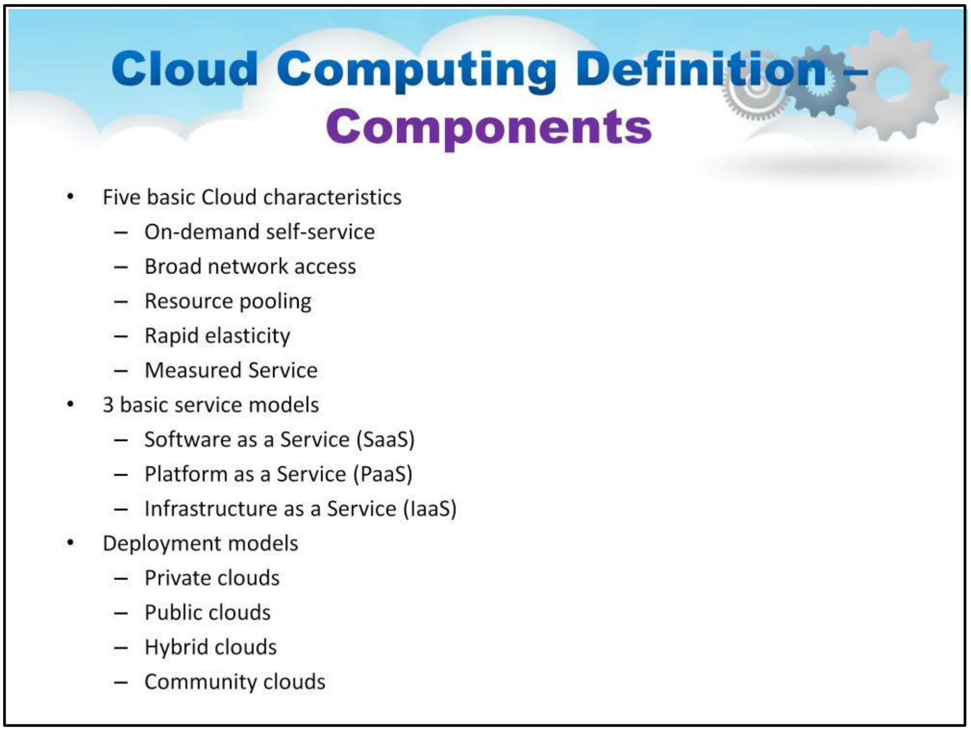
NIST Definition of Cloud Computing (according to NIST SP 800-145) (http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf)

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Cloud computing is a fusion of the two basic technologies powered by ubiquitous Internet connectivity:

- Utility Computing
- Service Oriented Architecture (SOA)



Cloud Computing Definition – Components

- Five basic Cloud characteristics
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
  - Measured Service
- 3 basic service models
  - Software as a Service (SaaS)
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)
- Deployment models
  - Private clouds
  - Public clouds
  - Hybrid clouds
  - Community clouds

This slide contains the list of the main components of Cloud Computing paradigm:

Five basic Cloud characteristics

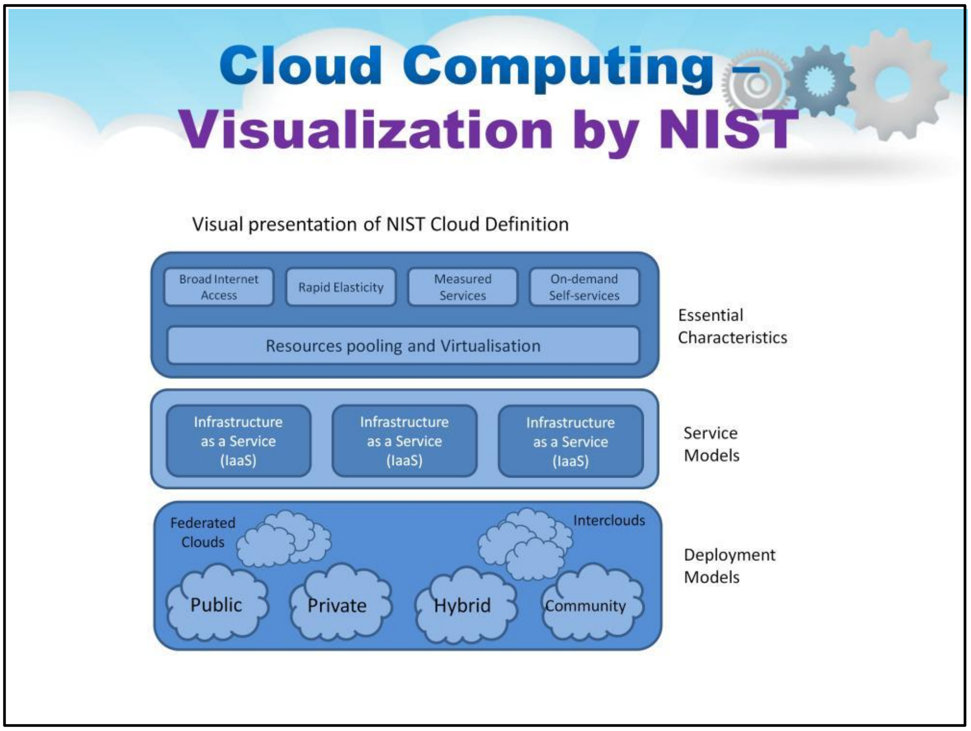
On-demand self-service
Broad network access
Resource pooling
Rapid elasticity
Measured Service

3 basic service models

Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)

Deployment models

Private clouds
Public clouds
Hybrid clouds
Community clouds



Cloud Computing – Visualization by NIST

Visual presentation of NIST Cloud Definition

Broad Internet Access

Rapid Elasticity

Measured Services

On-demand Self-services

Resources pooling and Virtualisation

Essential Characteristics

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS)

Service Models

Federated Clouds

Interclouds

Public

Private

Hybrid

Community

Deployment Models

This slide contains the visualization of Cloud Computing and its main components :

The essential Cloud characteristics

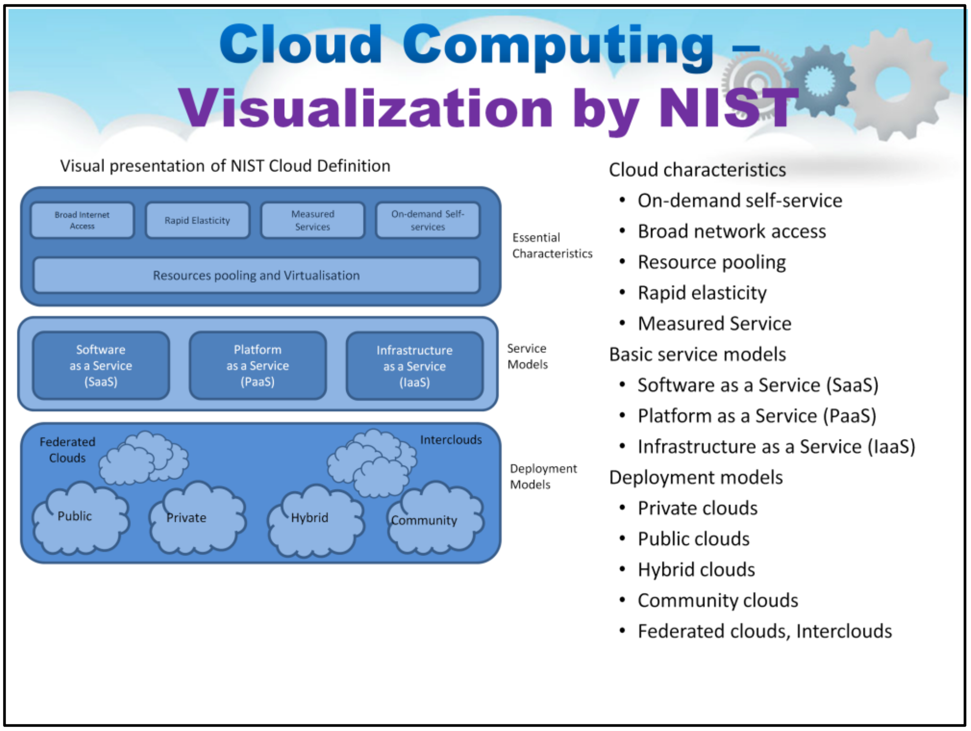
- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured Service

The main service models

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

The most popular deployment models

- Private clouds
- Public clouds
- Hybrid clouds
- Community clouds



Cloud Computing – Visualization by NIST

Visual presentation of NIST Cloud Definition

Cloud characteristics

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured Service

Basic service models

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Deployment models

- Private clouds
- Public clouds
- Hybrid clouds
- Community clouds
- Federated clouds, Interclouds

According to the point of view of NIST:
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics**,** three service models, and four deployment models.

Here you can see the cumulative view on
- all main components (in list from the right side)
and
- visualizations of relations between them i(on the scheme from the right side).

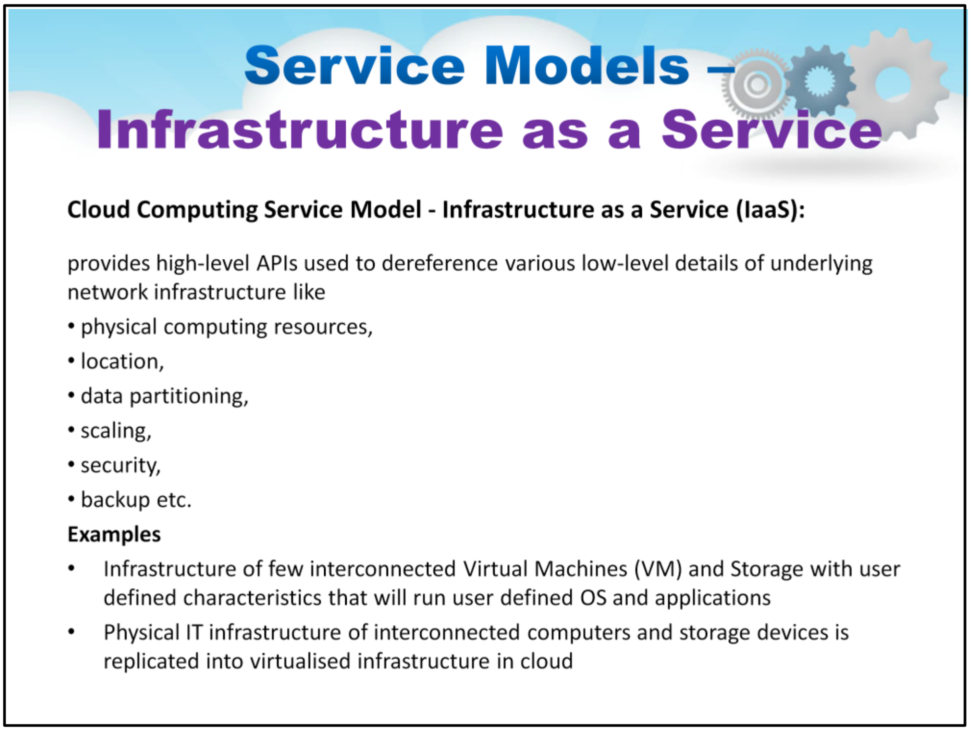
Cloud Computing

–

Service Models

Service Models

Now let's see consider the Service Models of Cloud Computing...



Service Models – Infrastructure as a Service

Cloud Computing Service Model - Infrastructure as a Service (IaaS):

provides high-level APIs used to dereference various low-level details of underlying network infrastructure like

- physical computing resources,
- location,
- data partitioning,
- scaling,
- security,
- backup etc.

Examples

- Infrastructure of few interconnected Virtual Machines (VM) and Storage with user defined characteristics that will run user defined OS and applications
- Physical IT infrastructure of interconnected computers and storage devices is replicated into virtualised infrastructure in cloud

Cloud Computing Service Model - Infrastructure as a Service (IaaS):
provides high-level APIs used to dereference various low-level details of underlying network infrastructure like

- physical computing resources,
- location,
- data partitioning,
- scaling,
- security,
- backup etc.

Examples

- Infrastructure of few interconnected Virtual Machines (VM) and Storage with user defined characteristics that will run user defined OS and applications
- Physical IT infrastructure of interconnected computers and storage devices is replicated into virtualised infrastructure in cloud

The capability provided to the consumer is to provision processing, storage, ***networks***, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and ***possibly limited control of select networking components (e.g., host firewalls)***.

The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Service Models – Platform as a Service

Cloud Computing Service Model – Platform as a Service (PaaS):

provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

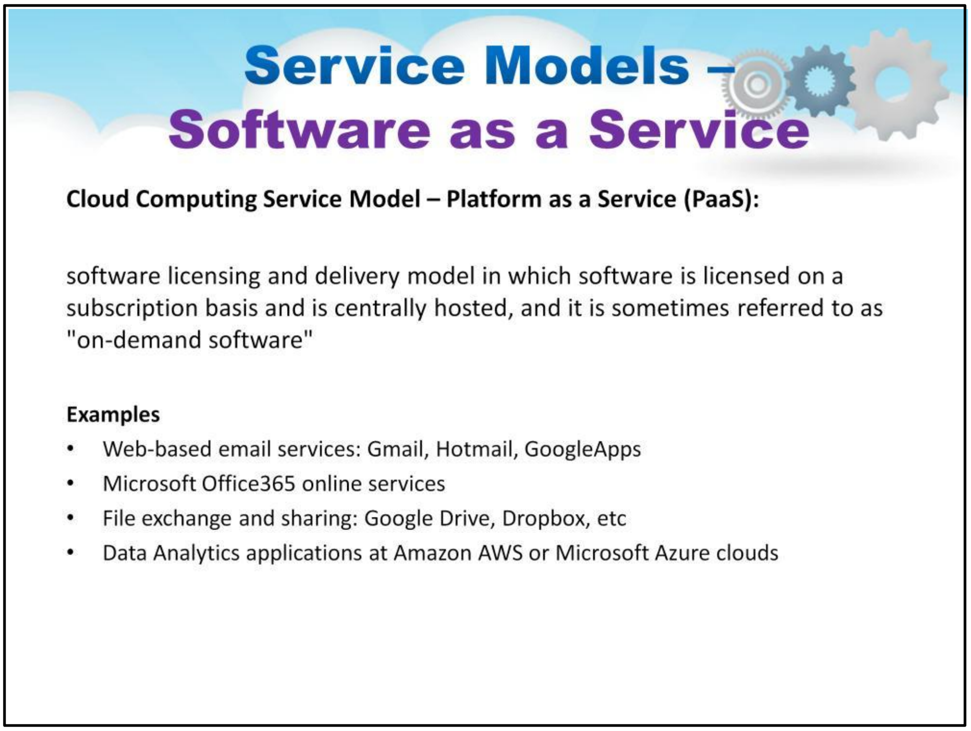
Examples

- Widely used enterprise management platform, such as CRM, provided for enterprise customer on-demand; it is easy scalable depending on workload
- Using PaaS cloud platform for new applications development and testing
- Using cloud business process management platform for running user business processes, e.g. Customer Relations Management (CRM) or Supply Chain Management (SCM)

Cloud Computing Service Model – Platform as a Service (PaaS):
provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.
Examples
Widely used enterprise management platform, such as CRM, provided for enterprise customer on-demand; it is easy scalable depending on workload
Using PaaS cloud platform for new applications development and testing
Using cloud business process management platform for running user business processes, e.g. Customer Relations Management (CRM) or Supply Chain Management (SCM)

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using *programming languages, libraries, services, and tools supported by the provider*. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has *control over the deployed applications and possibly configuration settings for the application-hosting environment*.
PaaS can be delivered in three ways:

- as a public cloud service from a provider, where the consumer controls software deployment with minimal configuration options, and the provider provides the networks, servers, storage, operating system (OS), middleware (e.g. Java runtime, .NET runtime, integration, etc.), database and other services to host the consumer's application.
- as a private service (software or appliance) inside the firewall.
- as software deployed on a public infrastructure as a service.



Service Models – Software as a Service

Cloud Computing Service Model – Platform as a Service (PaaS):

software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted, and it is sometimes referred to as "on-demand software"

Examples

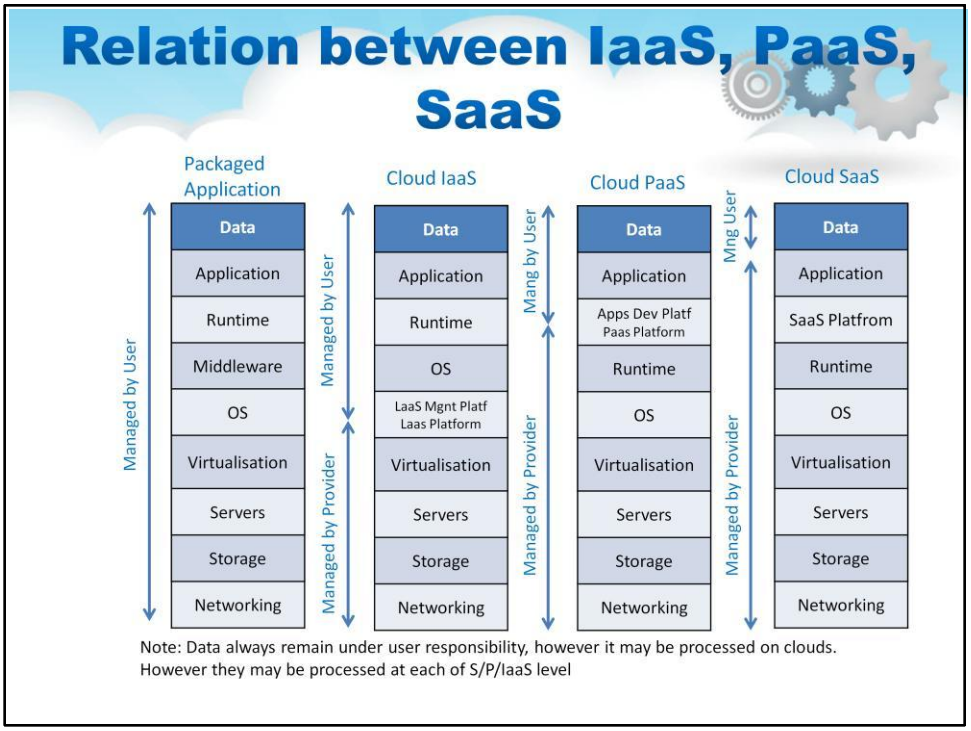
- Web-based email services: Gmail, Hotmail, GoogleApps
- Microsoft Office365 online services
- File exchange and sharing: Google Drive, Dropbox, etc
- Data Analytics applications at Amazon AWS or Microsoft Azure clouds

Cloud Computing Service Model – Platform as a Service (PaaS):
provides software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted, and it is sometimes referred to as "on-demand software"

Examples

- Web-based email services: Gmail, Hotmail, GoogleApps
- Microsoft Office365 online services
- File exchange and sharing: Google Drive, Dropbox, etc
- Data Analytics applications at Amazon AWS or Microsoft Azure clouds

The capability provided to the consumer is to *use the provider's applications running on a cloud infrastructure*. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of *limited user-specific application configuration settings*.



Relation between IaaS, PaaS, SaaS

Packaged Application	Cloud IaaS	Cloud PaaS	Cloud SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Runtime	Runtime	Apps Dev Platf Paas Platform	SaaS Platfrom
Middleware	OS	Runtime	Runtime
OS	LaaS Mgnt Platf Laas Platform	OS	OS
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Note: Data always remain under user responsibility, however it may be processed on clouds.
However they may be processed at each of S/P/IaaS level

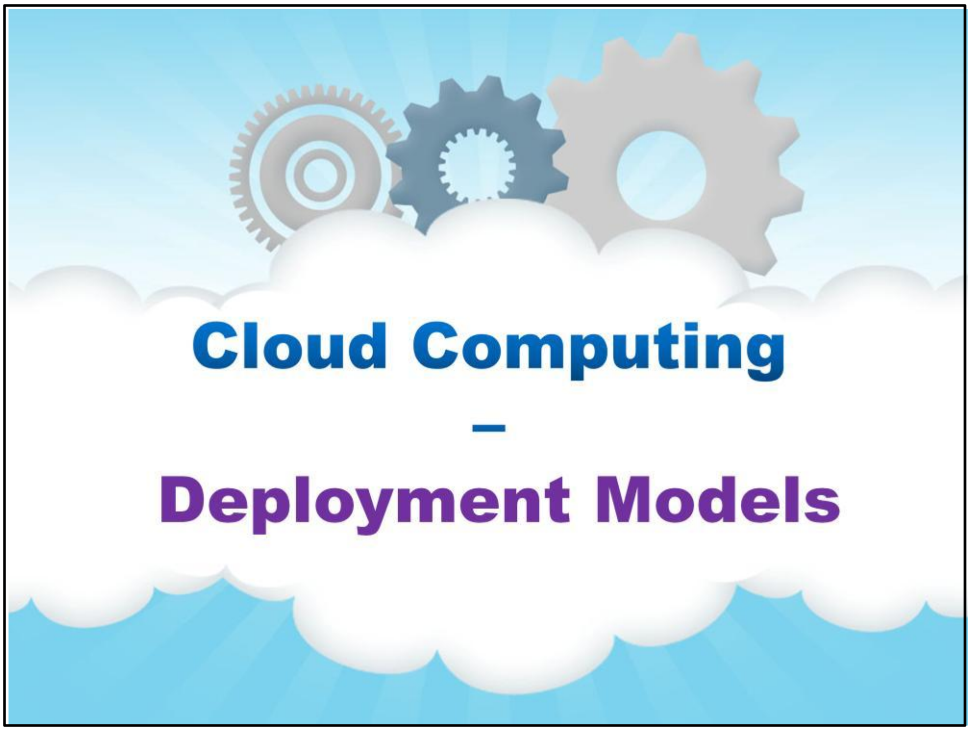
The use models for Cloud Computing are closely related.

For a traditional “physically installed” packaged application **(it is shown in the 1st column from the left side)**, the entire deployment stack is managed by the user, as can be seen in the first block of the illustration. The user is responsible for provisioning the complete server including networking and storage, and the Operating System software as well. Additionally, the user provisions whatever databases and middleware they need, any special runtime (Java or Ruby), and finally the application. The application data is also the responsibility (and the ownership of) the user.

The Cloud IaaS model **(it is shown in the 2nd column from the left side)** changes this significantly, alleviating a large portion of at least what would be considered “physical” infrastructure, this is shown in the second block of the illustration. The cloud management (sometimes called Cloud OS) software is shown in this diagram as “IaaS Mngt Platf”. The user obtains the hardware in a virtualized form from the IaaS CSP and worries only about the software stack, application, and data.

The Cloud PaaS model **(it is shown in the 3rd column from the left side)** further offers middleware, application runtime, and as much “software” infrastructure as possible, creating a convenient container for the application developer. While this requires the developer to restructure if not rewrite the application at the source code level to interface to the PaaS, this restricting will impart new scalability and availability strengths into the application.

The Cloud SaaS model **(it is shown in the 4th column from the left side)** is a consuming of just the application itself through a browser or web service.



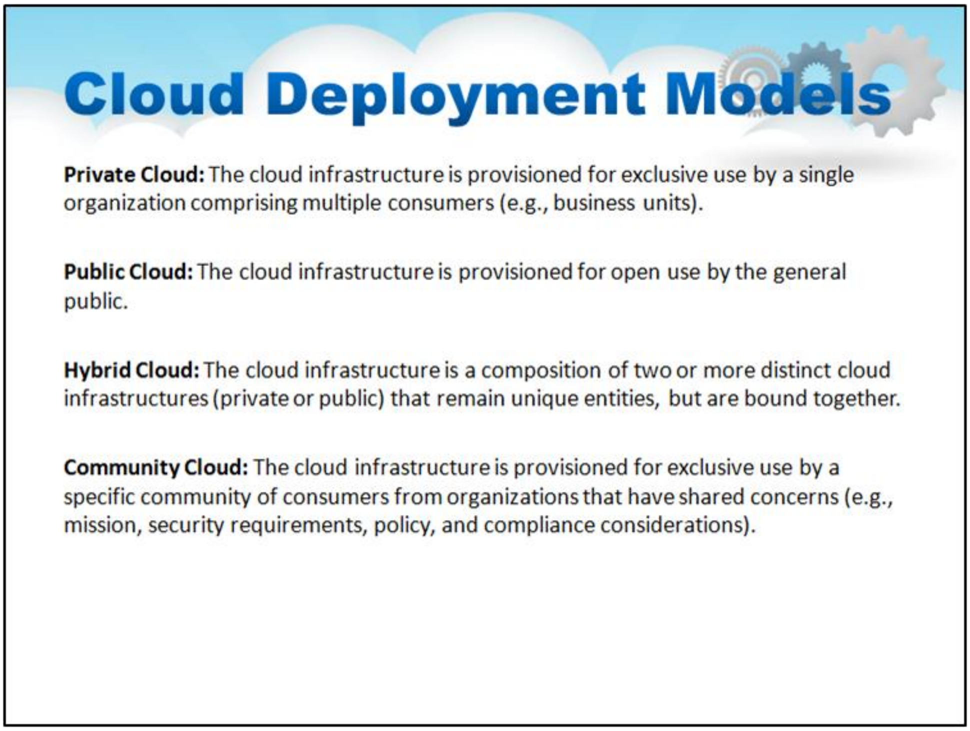
Cloud Computing

–

Deployment Models

Deployment Models

Now let's see consider the Deployment Models of Cloud Computing...



Cloud Deployment Models

Private Cloud: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units).

Public Cloud: The cloud infrastructure is provisioned for open use by the general public.

Hybrid Cloud: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private or public) that remain unique entities, but are bound together.

Community Cloud: The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations).

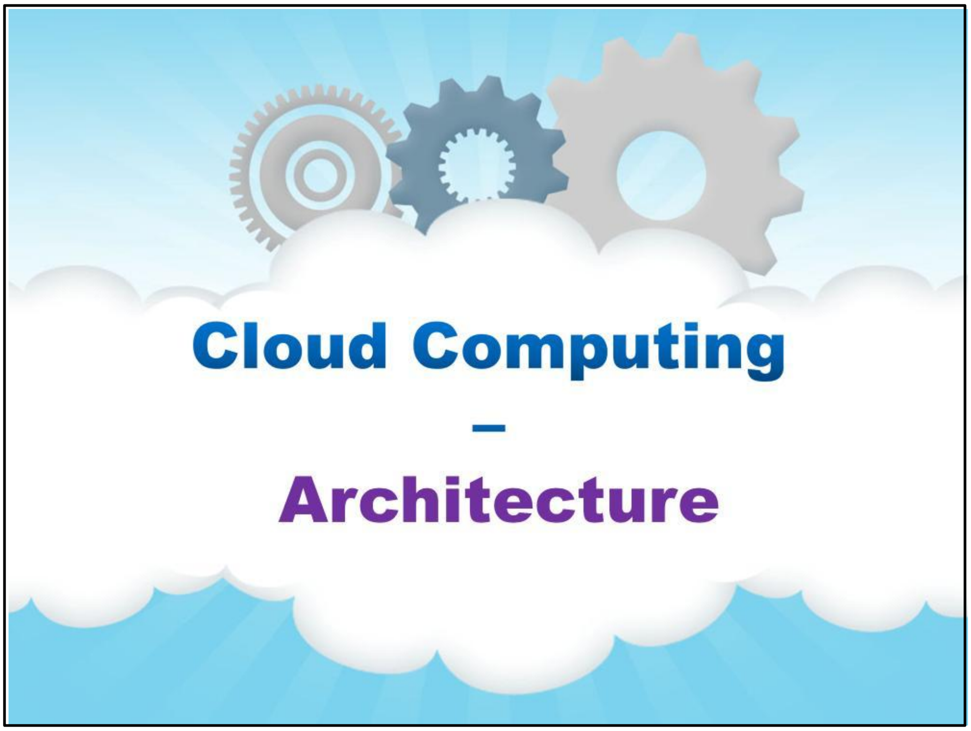
This slide contains the list of the main deployment models of Cloud Computing.

Private Cloud: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

Public Cloud: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

Hybrid Cloud: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Community Cloud: The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

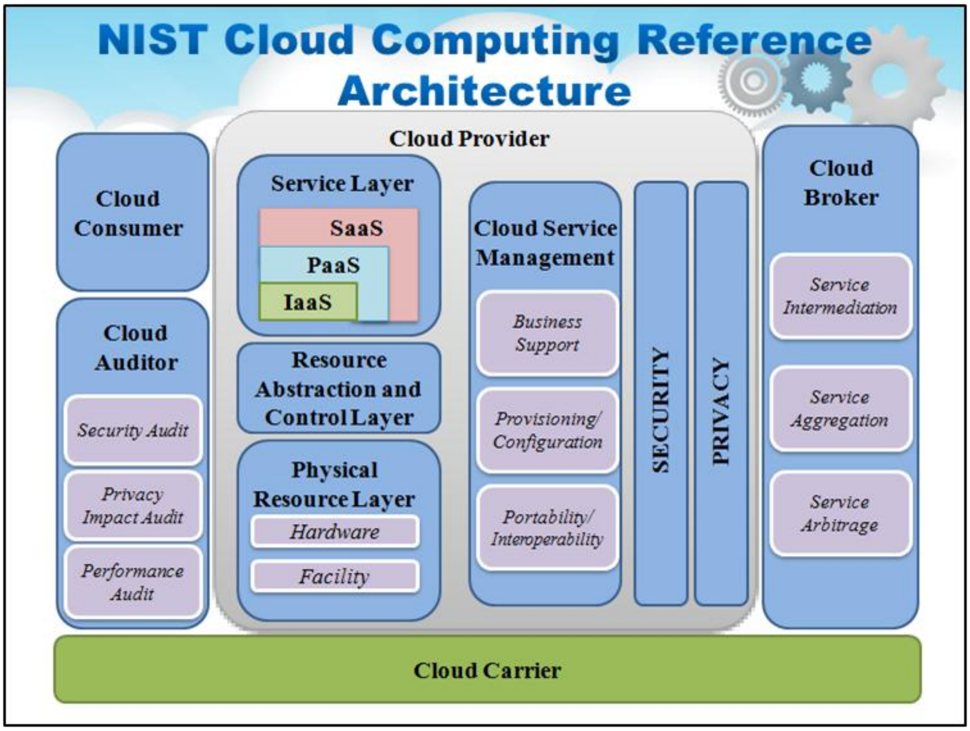


Cloud Computing

–

Architecture

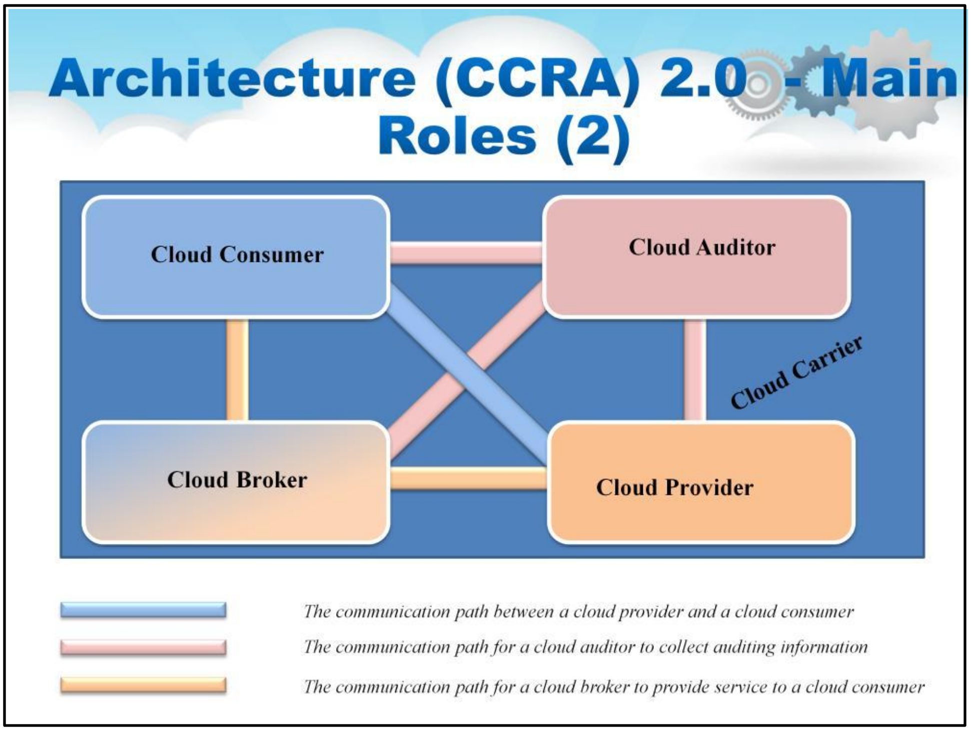
Architecture



Standardization has been very important from the beginning of the Cloud Computing development. It is important for both Cloud Services Providers (due to the scale of their infrastructure and facilities) and for cloud services consumers to allow their interoperability with other services.

Here we refer to the related standards by National Institute of Standards and Technology of USA (NIST) that define the Cloud Computing technology and Cloud Computing Reference Architecture.

NIST is active in fostering cloud computing practices that support interoperability, portability, and security requirements that are appropriate and achievable for important usage scenarios. Since first publication of the currently commonly accepted NIST Cloud definition in 2008, NIST is leading wide internationally recognized activity on defining conceptual and standard base in Cloud Computing, which has been resulted in publishing the following documents that create a solid base for cloud services development and offering: NIST SP 800-145, A NIST definition of cloud computing, and other formal documentation.



Architecture (CCRA) 2.0 - Main Roles (2)

Cloud Consumer

Cloud Auditor

Cloud Carrier

Cloud Broker

Cloud Provider

🟦 *The communication path between a cloud provider and a cloud consumer*

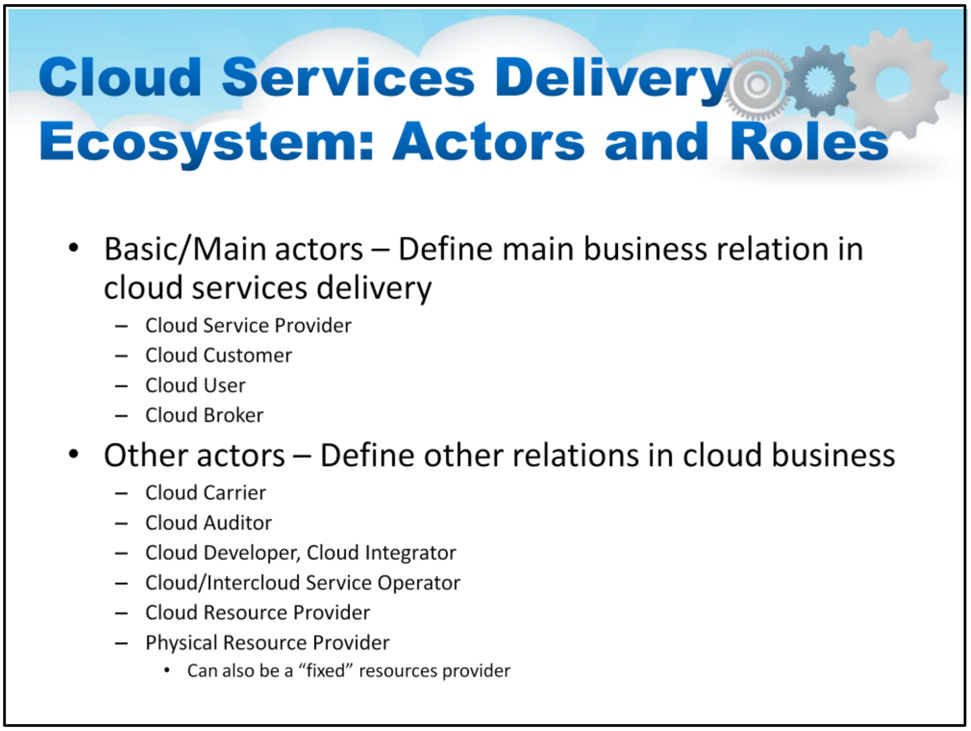
🟥 *The communication path for a cloud auditor to collect auditing information*

🟧 *The communication path for a cloud broker to provide service to a cloud consumer*

The slide presents a high level view of the NIST Cloud Computing Reference Architecture (CCRA), which identifies the major actors (Cloud Consumer, Cloud Service Provider, Cloud Auditor, Cloud Broker, and Cloud Carrier), their activities and functions in cloud computing. A cloud consumer may request cloud services from a cloud provider directly or via a cloud broker. A cloud auditor conducts independent audits and may contact the others to collect necessary information.

The proposed architecture is suitable for many purposes where network performance is not critical but needs to be extended with explicit network services provisioning and management when the cloud applications are critical to network latency like in case of enterprise applications, business transactions, crisis management, etc

NIST Cloud Computing Reference Architecture (CCRA) defines a number of stakeholders and actors which can be extended based on the basic of use cases analysis. The slide illustrates some of those on such a list and shows relationships of the stakeholders and actors.



Cloud Services Delivery Ecosystem: Actors and Roles

- Basic/Main actors – Define main business relation in cloud services delivery
  - Cloud Service Provider
  - Cloud Customer
  - Cloud User
  - Cloud Broker
- Other actors – Define other relations in cloud business
  - Cloud Carrier
  - Cloud Auditor
  - Cloud Developer, Cloud Integrator
  - Cloud/Intercloud Service Operator
  - Cloud Resource Provider
  - Physical Resource Provider
    - Can also be a “fixed” resources provider

Let’s consider the whole list of actors in Cloud Computing ecosystem of service delivery:

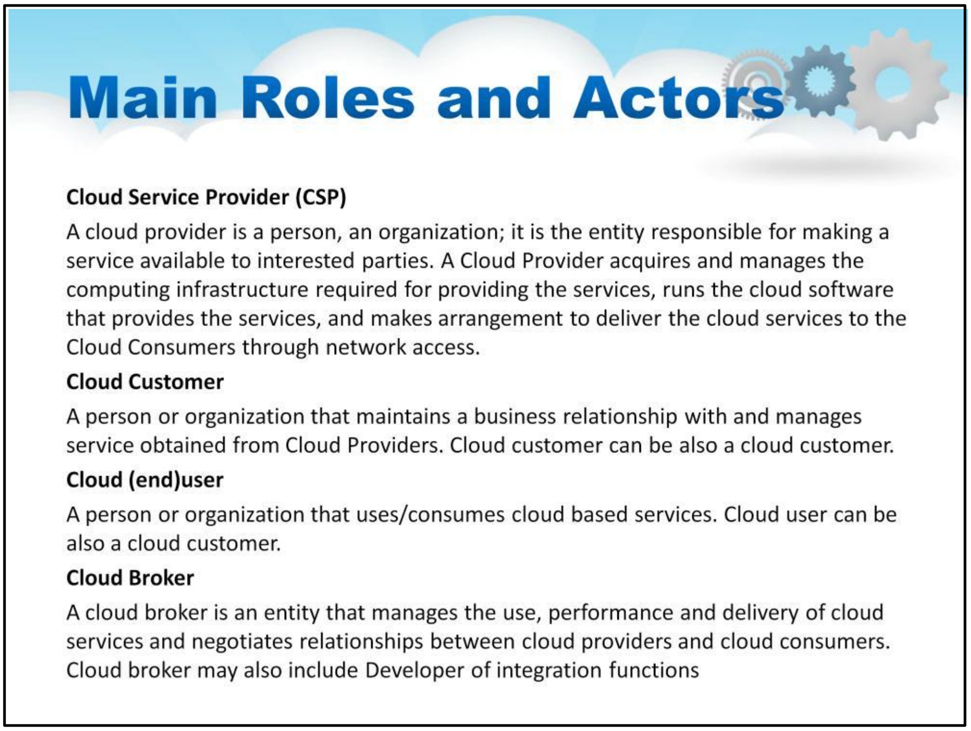
Basic/Main actors – Define main business relation in cloud services delivery

Cloud Service Provider
Cloud Customer
Cloud User
Cloud Broker

Other actors – Define other relations in cloud business

Cloud Carrier
Cloud Auditor
Cloud Developer, Cloud Integrator
Cloud/Intercloud Service Operator
Cloud Resource Provider
Physical Resource Provider

Can also be a “fixed” resources provider



Main Roles and Actors

Cloud Service Provider (CSP)

A cloud provider is a person, an organization; it is the entity responsible for making a service available to interested parties. A Cloud Provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangement to deliver the cloud services to the Cloud Consumers through network access.

Cloud Customer

A person or organization that maintains a business relationship with and manages service obtained from Cloud Providers. Cloud customer can be also a cloud customer.

Cloud (end)user

A person or organization that uses/consumes cloud based services. Cloud user can be also a cloud customer.

Cloud Broker

A cloud broker is an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers. Cloud broker may also include Developer of integration functions

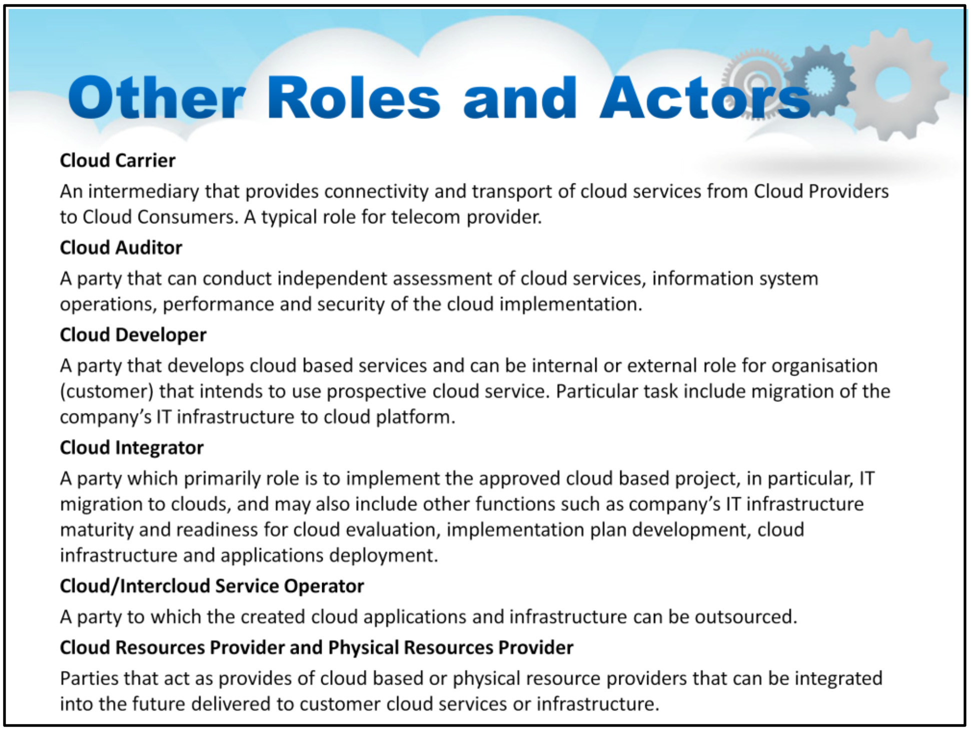
The main actors and their roles are as follows:

Cloud Service Provider (CSP)
A cloud provider is a person, an organization; it is the entity responsible for making a service available to interested parties. A Cloud Provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangement to deliver the cloud services to the Cloud Consumers through network access.

Cloud Customer
A person or organization that maintains a business relationship with and manages service obtained from Cloud Providers. Cloud customer can be also a cloud customer.

Cloud (end)user
A person or organization that uses/consumes cloud based services. Cloud user can be also a cloud customer.

Cloud Broker
A cloud broker is an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers. Cloud broker may also include Developer of integration functions



Other Roles and Actors

Cloud Carrier

An intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers. A typical role for telecom provider.

Cloud Auditor

A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.

Cloud Developer

A party that develops cloud based services and can be internal or external role for organisation (customer) that intends to use prospective cloud service. Particular task include migration of the company's IT infrastructure to cloud platform.

Cloud Integrator

A party which primarily role is to implement the approved cloud based project, in particular, IT migration to clouds, and may also include other functions such as company's IT infrastructure maturity and readiness for cloud evaluation, implementation plan development, cloud infrastructure and applications deployment.

Cloud/Intercloud Service Operator

A party to which the created cloud applications and infrastructure can be outsourced.

Cloud Resources Provider and Physical Resources Provider

Parties that act as provides of cloud based or physical resource providers that can be integrated into the future delivered to customer cloud services or infrastructure.

Other actors and roles include:

Cloud Carrier

An intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers. A typical role for telecom provider.

Cloud Auditor

A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.

Cloud Developer

A party that develops cloud based services and can be internal or external role for organisation (customer) that intends to use prospective cloud service. Particular task include migration of the company's IT infrastructure to cloud platform.

Cloud Integrator

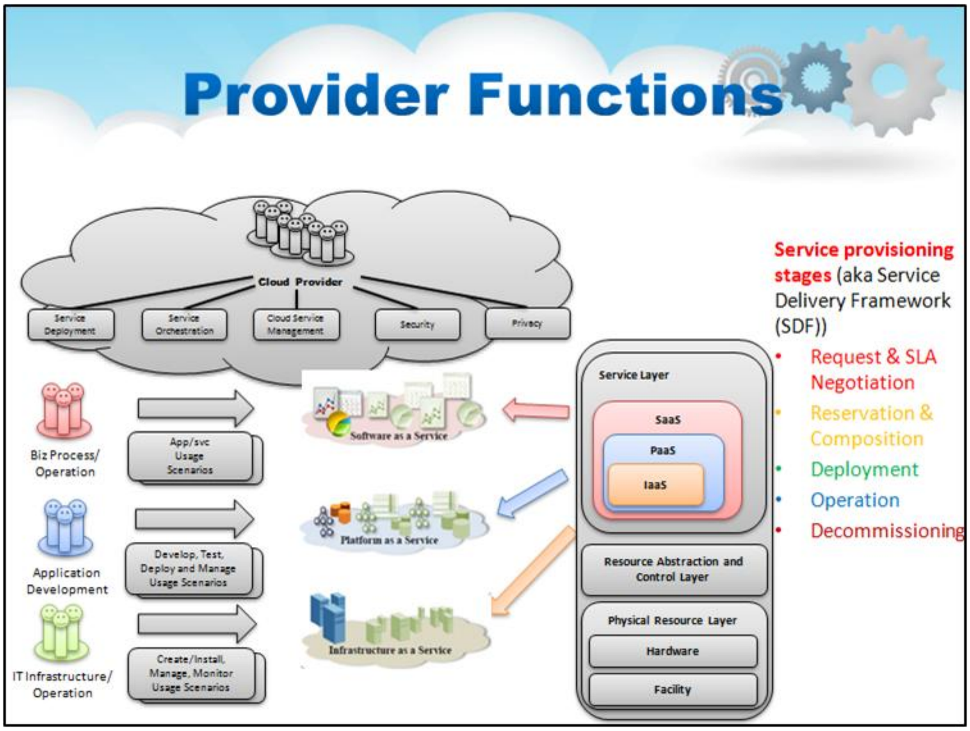
A party which primarily role is to implement the approved cloud based project, in particular, IT migration to clouds, and may also include other functions such as company's IT infrastructure maturity and readiness for cloud evaluation, implementation plan development, cloud infrastructure and applications deployment.

Cloud/Intercloud Service Operator

A party to which the created cloud applications and infrastructure can be outsourced.

Cloud Resources Provider and Physical Resources Provider

Parties that act as provides of cloud based or physical resource providers that can be integrated into the future delivered to customer cloud services or infrastructure.



Provider Functions

Cloud Provider

Service Deployment

Service Orchestration

Cloud Service Management

Security

Privacy

Biz Process/ Operation

App/svc Usage Scenarios

Application Development

Develop, Test, Deploy and Manage Usage Scenarios

IT Infrastructure/ Operation

Create/Install, Manage, Monitor Usage Scenarios

Software as a Service

Platform as a Service

Infrastructure as a Service

Service Layer

SaaS

PaaS

IaaS

Resource Abstraction and Control Layer

Physical Resource Layer

Hardware

Facility

Service provisioning stages (aka Service Delivery Framework (SDF))

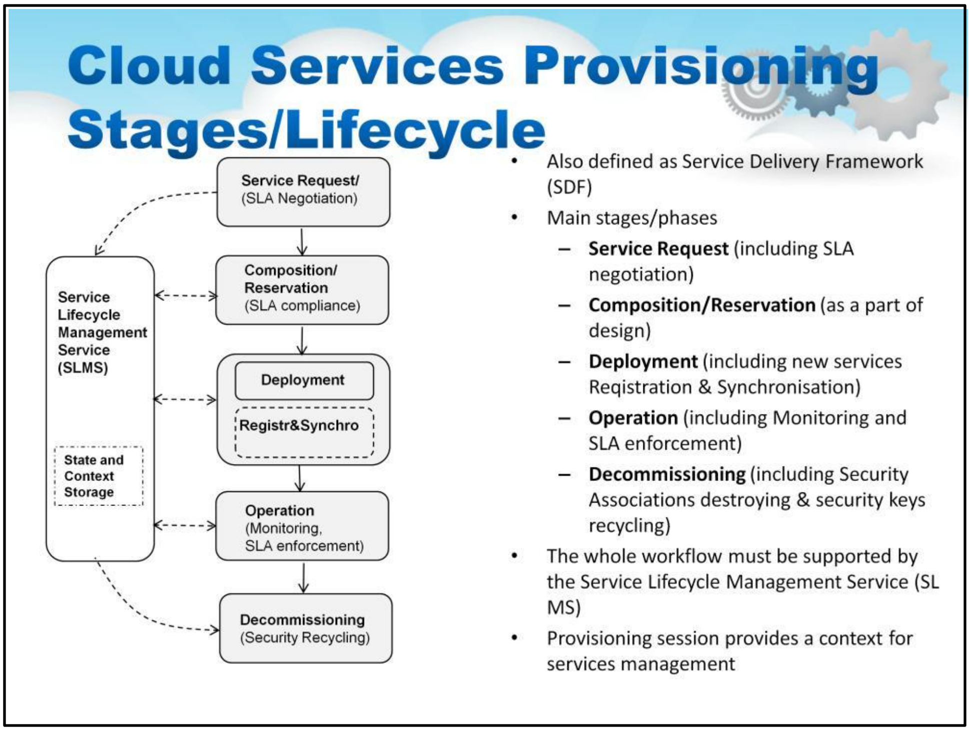
- Request & SLA Negotiation
- Reservation & Composition
- Deployment
- Operation
- Decommissioning

The Cloud Service Provider is best known for the visible functions it provides, that is enabling IaaS, PaaS, and SaaS capabilities.

Behind the scenes, there is a Service Delivery Framework (SDF) which provides the mechanics for the Service Provisioning.

The Service Provisioning: occurs in several stages: Request & SLA Negotiation Reservation & Composition Deployment Operation Decommissioning.

These are explained on the next slide.



Cloud Services Provisioning Stages/Lifecycle

- Also defined as Service Delivery Framework (SDF)
- Main stages/phases
  - **Service Request** (including SLA negotiation)
  - **Composition/Reservation** (as a part of design)
  - **Deployment** (including new services Reqistration & Synchronisation)
  - **Operation** (including Monitoring and SLA enforcement)
  - **Decommissioning** (including Security Associations destroying & security keys recycling)
- The whole workflow must be supported by the Service Lifecycle Management Service (SL MS)
- Provisioning session provides a context for services management

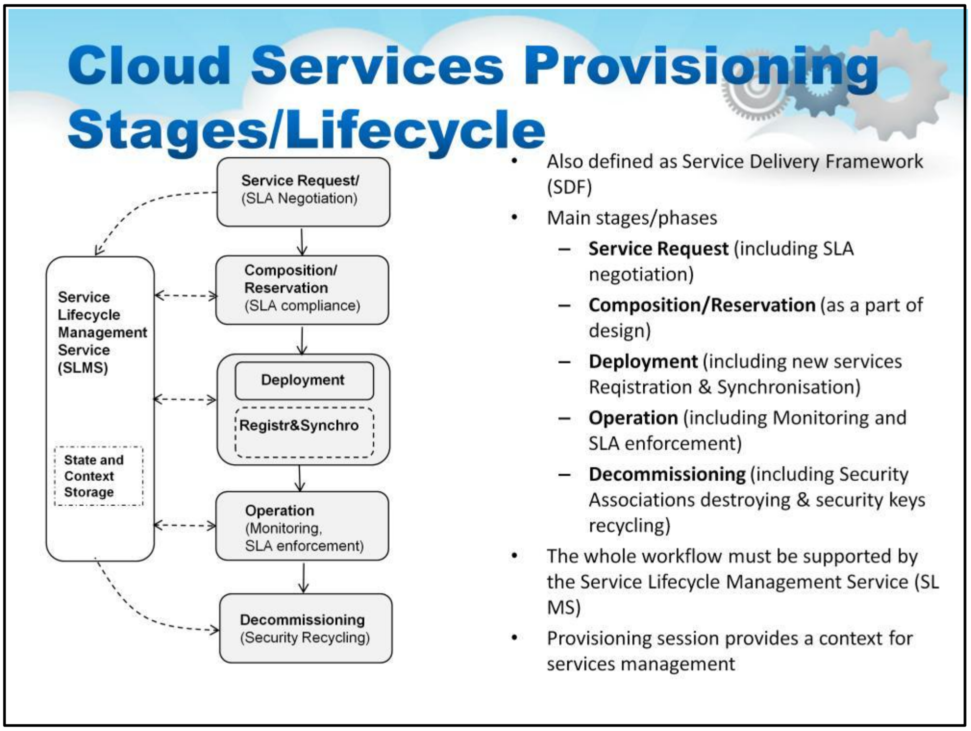
The Slide illustrates the main service provisioning or delivery stages that address specific requirements of the provisioned on-demand virtualized services:

Service Request Stage including service-level agreement (SLA) negotiation. The SLA can describe Quality of Service (QoS) and security requirements of the negotiated infrastructure service along with information that facilitates authentication of service requests from users. This stage also includes generation of the Global Reservation ID (GRI) that will serve as a provisioning session identifier and will bind all other stages and related security context.

Composition/Reservation Stage that also includes Reservation Session Binding with the GRI, which provides support for complex reservation processes in multi-domain multi-provider environments. This stage may require access control and SLA/policy enforcement.

Deployment Stage, including services Registration and Synchronization. The deployment stage begins after all component resources have been reserved and includes distribution of the common composed service context (including security context) and binding the reserved resources or services to the GRI as a common provisioning session ID.

The Registration and Synchronization stage (which are optional) specifically targets scenarios with provisioned service migration or re-planning.



Cloud Services Provisioning Stages/Lifecycle

Service Request/ (SLA Negotiation)

Composition/ Reservation (SLA compliance)

Deployment

Registr&Synchro

Operation (Monitoring, SLA enforcement)

Decommissioning (Security Recycling)

Service Lifecycle Management Service (SLMS)

State and Context Storage

- Also defined as Service Delivery Framework (SDF)
- Main stages/phases
  - **Service Request** (including SLA negotiation)
  - **Composition/Reservation** (as a part of design)
  - **Deployment** (including new services Reqistration & Synchronisation)
  - **Operation** (including Monitoring and SLA enforcement)
  - **Decommissioning** (including Security Associations destroying & security keys recycling)
- The whole workflow must be supported by the Service Lifecycle Management Service (SL MS)
- Provisioning session provides a context for services management

Operation Stage (including Monitoring). This is the main operational stage of the provisioned on-demand cloud services. Monitoring is an important functionality of this stage to ensure service availability and secure operation, including SLA enforcement.

Decommissioning Stage ensures that all sessions are terminated, data is cleaned up, and session security context is recycled. The decommissioning stage can also provide information to or initiate service usage accounting.

Two additional (sub-)stages can be initiated from the Operation stage, based on the running service or resources state:

Re-composition or Re-planning Stage should allow incremental infrastructure changes.

Recovery/Migration Stage can be initiated by the user or provider. This process can use MD-SLC to initiate a full or partial resource re-synchronization, it may also require re-composition.

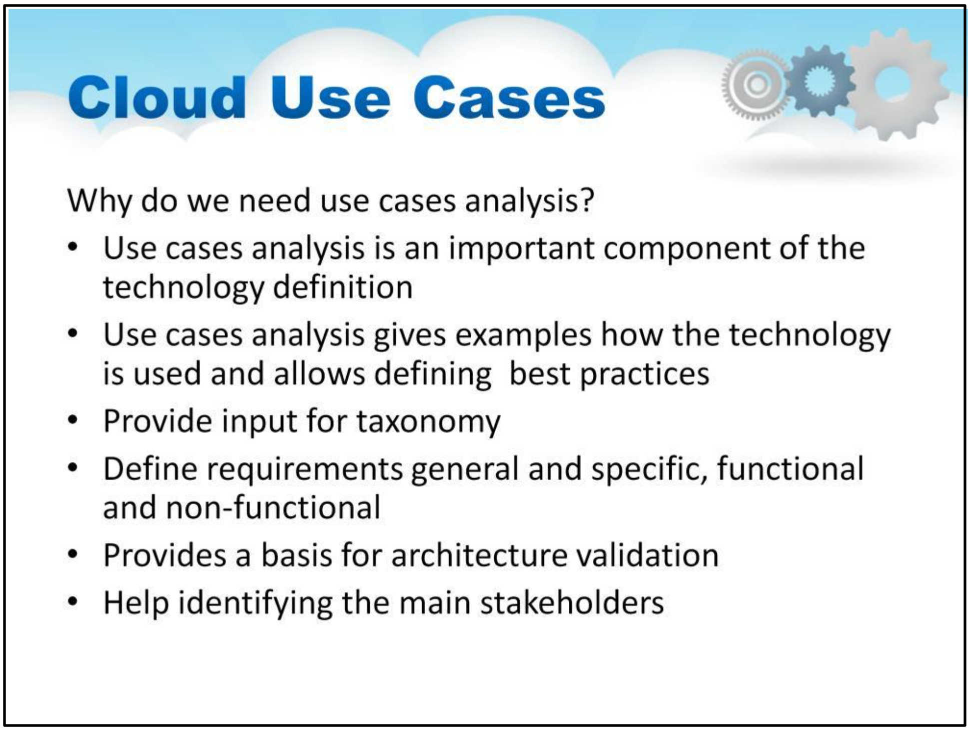
Implementation of the proposed SDF requires a special Service Lifecycle Metadata Repository (MD SLC) to support consistent services lifecycle management. MD SLC keeps the services metadata that include at least service state, service properties, and services configuration information. This functionality is a part of the cloud management software and cloud platform software.

Cloud Computing

–

Use Cases

Use Cases



Cloud Use Cases

Why do we need use cases analysis?

- Use cases analysis is an important component of the technology definition
- Use cases analysis gives examples how the technology is used and allows defining best practices
- Provide input for taxonomy
- Define requirements general and specific, functional and non-functional
- Provides a basis for architecture validation
- Help identifying the main stakeholders

Why do we need use cases analysis?

- Use cases analysis is an important component of the technology definition
- Use cases analysis gives examples how the technology is used and allows defining best practices
- Provide input for taxonomy
- Define requirements general and specific, functional and non-functional
- Provides a basis for architecture validation
- Help identifying the main stakeholders

We should not also exclude the analysis of use cases as valuable information for education and professional training. What we actually do in this course.

On the other hand, when planning for company’s IT infrastructure migration to clouds, the applicable cloud service and deployment model is selected based on a number of factors:

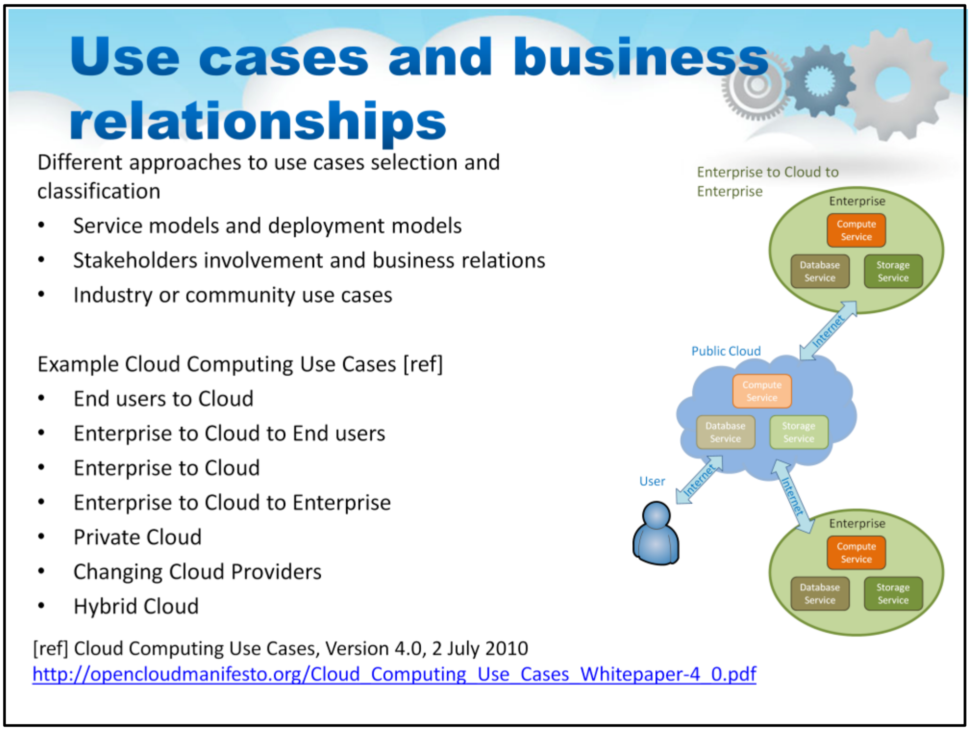
♦ Company, business and applications must have economical or business benefits from

♦ moving to clouds

♦ Besides purely technical, other business, organizational or staff factors must be considered

♦ Some (older) applications may need to be re-designed

♦ Transition period from in-house to cloud services takes time and must be carefully planned



Use cases and business relationships

Different approaches to use cases selection and classification

- Service models and deployment models
- Stakeholders involvement and business relations
- Industry or community use cases

Example Cloud Computing Use Cases [ref]

- End users to Cloud
- Enterprise to Cloud to End users
- Enterprise to Cloud
- Enterprise to Cloud to Enterprise
- Private Cloud
- Changing Cloud Providers
- Hybrid Cloud

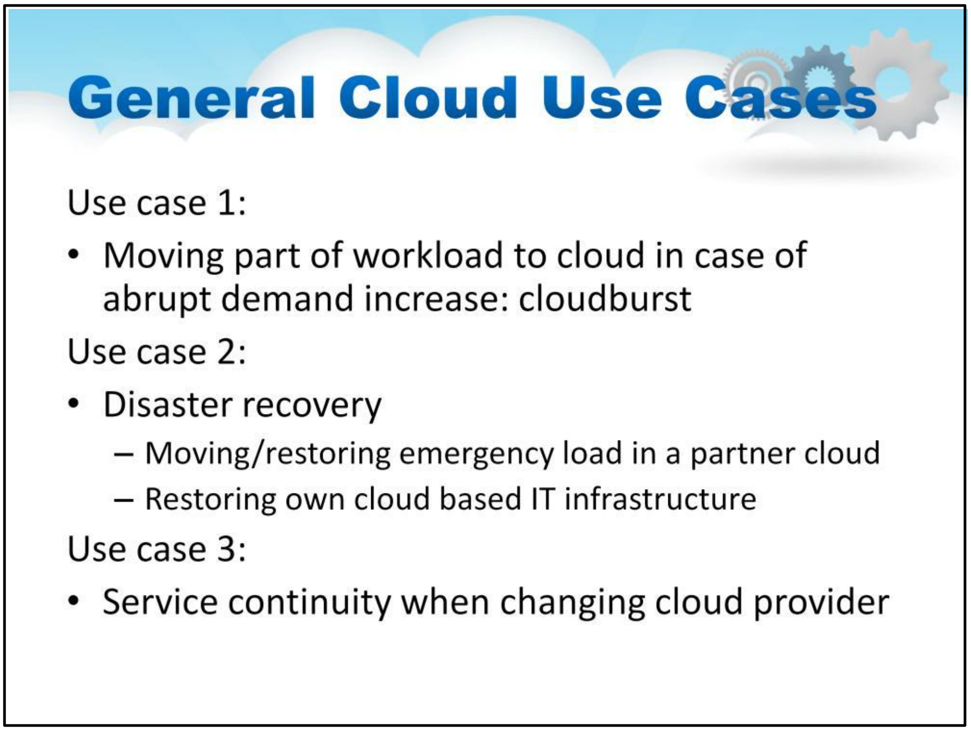
[ref] Cloud Computing Use Cases, Version 4.0, 2 July 2010
http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf

There are different approaches to selsction and classifiacation of use cases:

- to look at the variety of service models and deployment models;
- to enumerate possibilities based on stakeholder involvement and business relations;
- to consider and adopt common industry or community use cases.

A collaborative effort of by Cloud Computing researchers has come up with one way to characterize example use cases: End users to Cloud Enterprise to Cloud to End users Enterprise to Cloud Enterprise to Cloud to Enterprise Private Cloud Changing Cloud Providers Hybrid Cloud Something important to consider is, that these scenarios don't suddenly "happen", they are built out, or more accurately "grown into" because of a particular enterprise need. Think of the use case, where an enterprise wants to migrate part of its IT infrastructure to a Cloud. It is not sure which approach it needs yet. But this is the use case that all companies and enterprises face when they decide to move their IT infrastructure to clouds. The motivation to do this is to benefit from the functional cloud benefits described above as well as economical and business values.

As we pointed out, full cloud migration doesn't happen in one step. For the big organizations, it typically starts from implementing private cloud and moving local IT services to cloud based. This step will also lead to the whole IT maturity and its readiness to outsource some services to public cloud. This creates a hybrid cloud. And the next step will be to move operational IT infrastructure or some departments entirely to cloud. What are challenges and how to address them we will discuss in the subsequent use cases.



General Cloud Use Cases

Use case 1:

- Moving part of workload to cloud in case of abrupt demand increase: cloudburst

Use case 2:

- Disaster recovery
  - Moving/restoring emergency load in a partner cloud
  - Restoring own cloud based IT infrastructure

Use case 3:

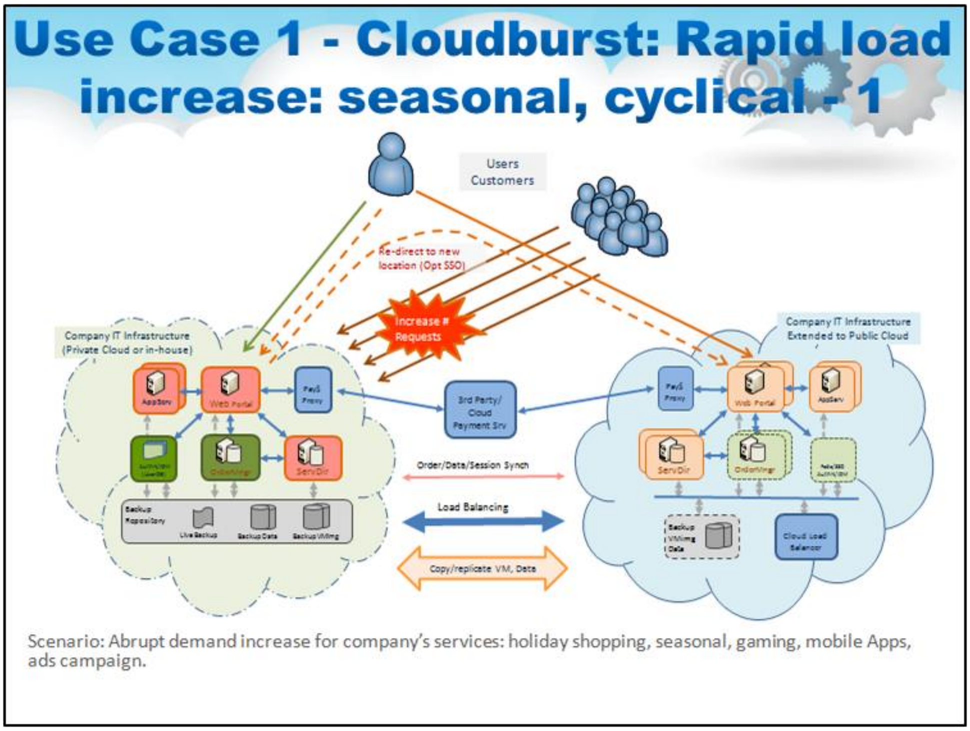
- Service continuity when changing cloud provider

We can identify the following general cloud use cases which we discuss in details below.

Use case 1:
Moving part of workload to cloud in case of abrupt demand increase: cloudburst

Use case 2:
Disaster recovery
Moving/restoring emergency load in a partner cloud
Restoring own cloud based IT infrastructure

Use case 3:
Service continuity when changing cloud provider



Use Case 1 - Cloudburst: Rapid load increase: seasonal, cyclical - 1

Scenario: Abrupt demand increase for company's services: holiday shopping, seasonal, gaming, mobile Apps, ads campaign.

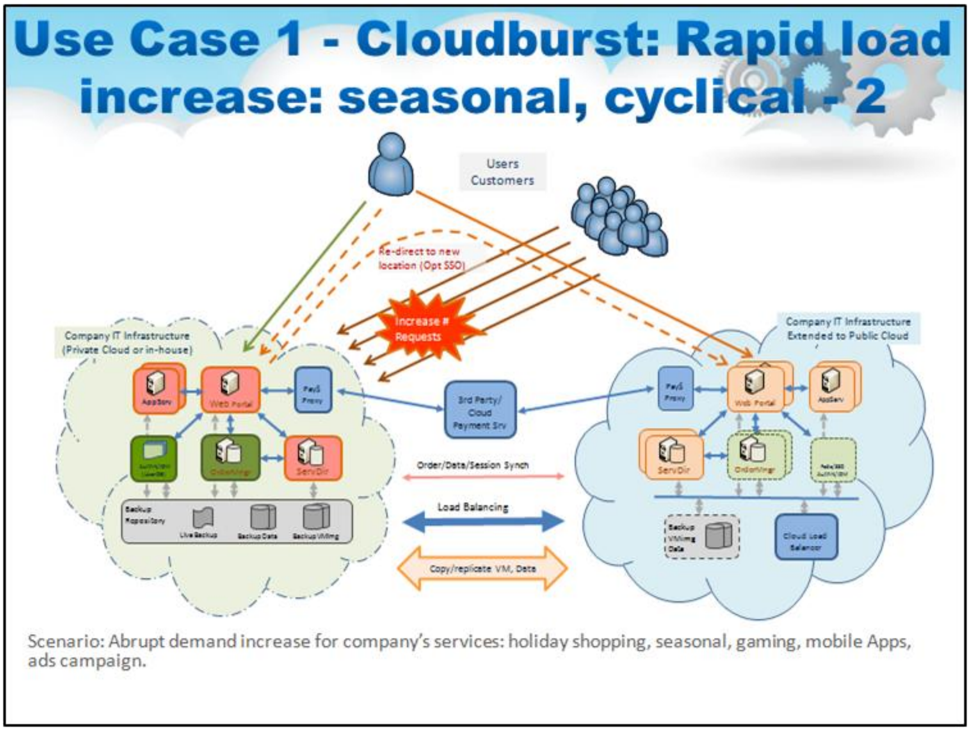
This is Use case 1: Extending services and capacities into public cloud in case of rapid demand increase (“cloudbursting” scenario).

This term “cloudbursting” is an imprecise term widely used by businesses to describe situations when workload is temporarily migrated to cloud, extending and replicating the private cloud resources and VMs (using the formula “buy the base, rent a spike”).

We will consider one “cloudbursting” scenario as it is one of key use cases for cloud computing that bring important advantages for use of cloud technologies by SME (also called SMB –Small and Medium Business).

The hypothetical SME is a startup with already running business but considering new product or service that will require building or outsourcing new IT infrastructure and resources. Main pre-conditions and requirements:

Untested/unpredictable workload -In particular for webshops, social sites, gaming and mobile applications. The business wants to deploy services and infrastructure “elastically” so they can be expanded and un-expanded to meet the actual demands. This allows for the easy extension (in case of success) and cheap failure (in case if service is not successful) .



Use Case 1 - Cloudburst: Rapid load increase: seasonal, cyclical - 2

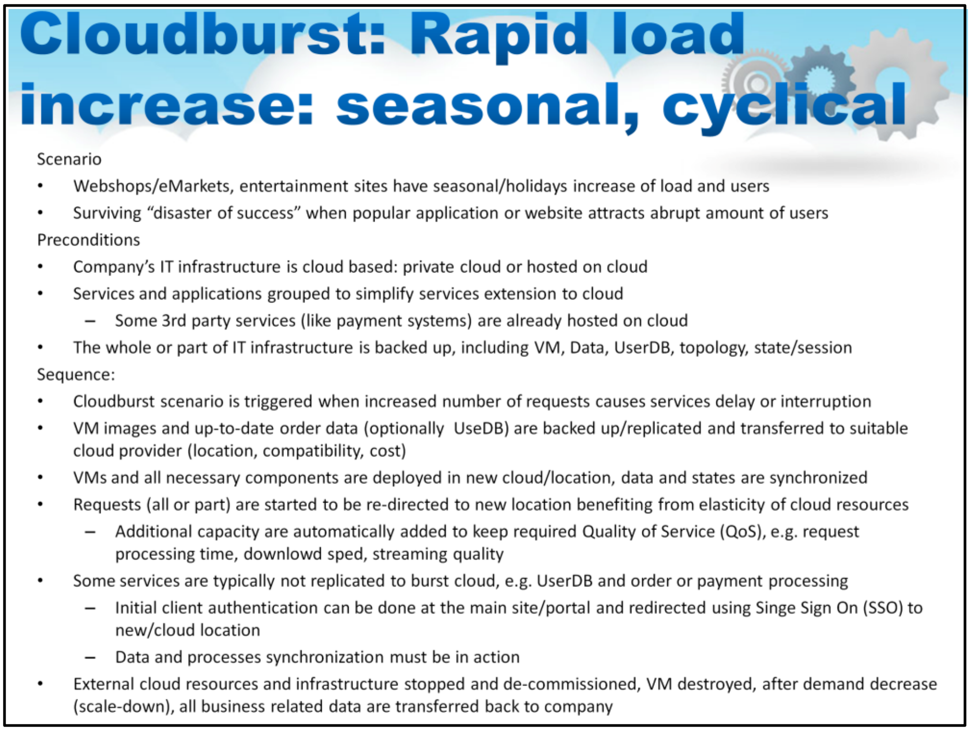
Scenario: Abrupt demand increase for company's services: holiday shopping, seasonal, gaming, mobile Apps, ads campaign.

Another driving force is handle an expected service expansion to different countries and geographical zones. The ability to bring up infrastructure else where addresses this need. One wants to have multiple infrastructures anyway, to provide for load balancing and latency minimization across geographies.

The SME use case should also address a potential situation called “High-profile success disaster” what happens when service or site popularity grows rapidly what can a cases with modern web and mobile applications. Known examples/stories include the BestBuy year 2012 holiday shopping service overloading, or Netflix service outage the same year 2012 after problems at AWS that hosts Netflix services. Netflix service was also irregularly available during Christmas holidays in 2013 in Europe. All these denial of service cases were caused by increased demand from customers.

The cloud based solution can effectively address the situations with the demand influx, however applications and services must be designed in a way to allow their easy extension, replication and relocation to external cloud provider infrastructure. It is important to repeat that not all services and operational procedures are suitable for moving to clouds, in particular those that deal with the sensitive data or require critical availability.

The diagram on the slide illustrates this use case. The company can be one of the type or running the following applications: webshop or e-market, entertainment or gaming application. Such applications are known for having seasonal or cyclical demand, and in case of success can attract abruptly increased amount of users.



Cloudburst: Rapid load increase: seasonal, cyclical

Scenario

- Webshops/eMarkets, entertainment sites have seasonal/holidays increase of load and users
- Surviving “disaster of success” when popular application or website attracts abrupt amount of users

Preconditions

- Company’s IT infrastructure is cloud based: private cloud or hosted on cloud
- Services and applications grouped to simplify services extension to cloud
  - Some 3rd party services (like payment systems) are already hosted on cloud
- The whole or part of IT infrastructure is backed up, including VM, Data, UserDB, topology, state/session

Sequence:

- Cloudburst scenario is triggered when increased number of requests causes services delay or interruption
- VM images and up-to-date order data (optionally UseDB) are backed up/replicated and transferred to suitable cloud provider (location, compatibility, cost)
- VMs and all necessary components are deployed in new cloud/location, data and states are synchronized
- Requests (all or part) are started to be re-directed to new location benefiting from elasticity of cloud resources
  - Additional capacity are automatically added to keep required Quality of Service (QoS), e.g. request processing time, downlowd sped, streaming quality
- Some services are typically not replicated to burst cloud, e.g. UserDB and order or payment processing
  - Initial client authentication can be done at the main site/portal and redirected using Singe Sign On (SSO) to new/cloud location
  - Data and processes synchronization must be in action
- External cloud resources and infrastructure stopped and de-commissioned, VM destroyed, after demand decrease (scale-down), all business related data are transferred back to company

To finalize let’s emphasize the crucial points of this Use Case:

Scenario

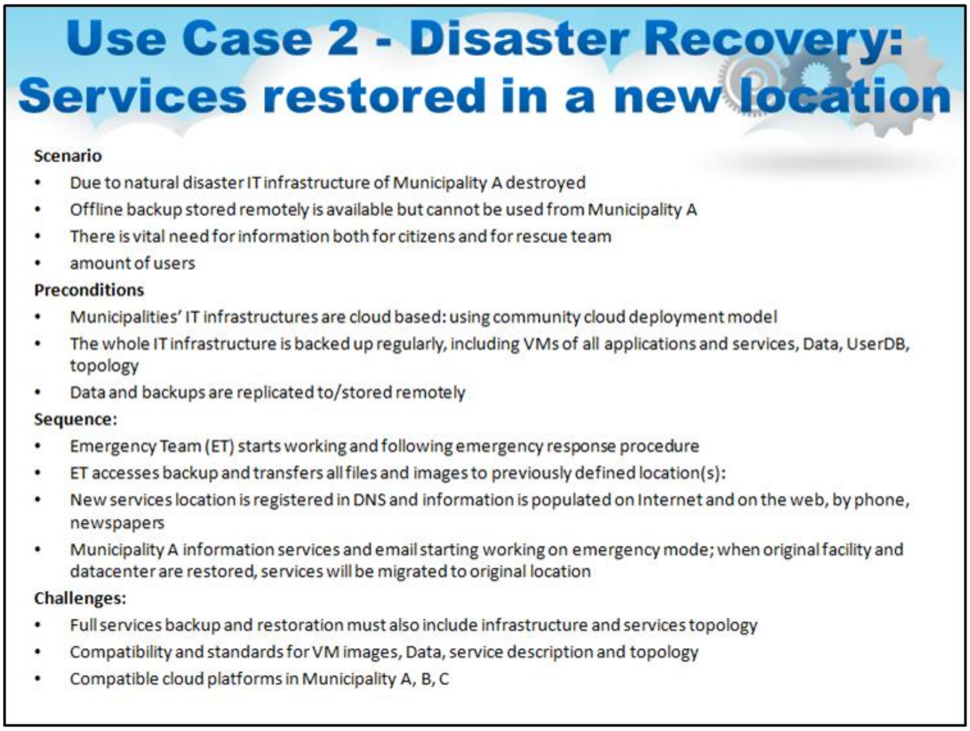
- Webshops/eMarkets, entertainment sites have seasonal/holidays increase of load and users
- Surviving “disaster of success” when popular website attracts abrupt amount of users

Preconditions

- Company’s IT infrastructure is cloud based: private cloud or hosted on cloud
- Services and applications grouped to simplify services extension to cloud
- Some 3rd party services (like payment systems) are already hosted on cloud
- The whole or part of IT infrastructure is backed up, including VM, Data, UserDB, topology, state/session

Sequence:

- Cloudburst scenario is triggered when increased number of requests causes services delay or interruption
- VM images and up-to-date order data (optionally UseDB) are backed up/replicated and transferred to suitable cloud provider (location, compatibility, cost)
- VMs and all necessary components are deployed in new cloud/location, data and states are synchronized
- Requests (all or part) are started to be re-directed to new location benefiting from elasticity of cloud resources
- Some services are typically not replicated to burst cloud, e.g. UserDB and order or payment processing
- External cloud resources and infrastructure stopped, VM destroyed, after demand decrease (scale-down), all business related data are transferred back to company.



Use Case 2 - Disaster Recovery: Services restored in a new location

Scenario

- Due to natural disaster IT infrastructure of Municipality A destroyed
- Offline backup stored remotely is available but cannot be used from Municipality A
- There is vital need for information both for citizens and for rescue team
- amount of users

Preconditions

- Municipalities' IT infrastructures are cloud based: using community cloud deployment model
- The whole IT infrastructure is backed up regularly, including VMs of all applications and services, Data, UserDB, topology
- Data and backups are replicated to/stored remotely

Sequence:

- Emergency Team (ET) starts working and following emergency response procedure
- ET accesses backup and transfers all files and images to previously defined location(s):
- New services location is registered in DNS and information is populated on Internet and on the web, by phone, newspapers
- Municipality A information services and email starting working on emergency mode; when original facility and datacenter are restored, services will be migrated to original location

Challenges:

- Full services backup and restoration must also include infrastructure and services topology
- Compatibility and standards for VM images, Data, service description and topology
- Compatible cloud platforms in Municipality A, B, C

Use case 2: Disaster recovery and large scale provider failure

Scenario

Due to natural disaster IT infrastructure of Municipality A destroyed

Offline backup stored remotely is available but cannot be used from Municipality A

There is vital need for information both for citizens and for rescue team amount of users

Preconditions

Municipalities' IT infrastructures are cloud based: using community cloud deployment model

The whole IT infrastructure is backed up regularly, including VMs of all applications and services, Data, UserDB, topology

Data and backups are replicated to/stored remotely

Sequence:

Emergency Team (ET) starts working and following emergency response procedure

ET accesses backup and transfers all files and images to previously defined location(s):

New services location is registered in DNS and information is populated on Internet and on the web, by phone, newspapers

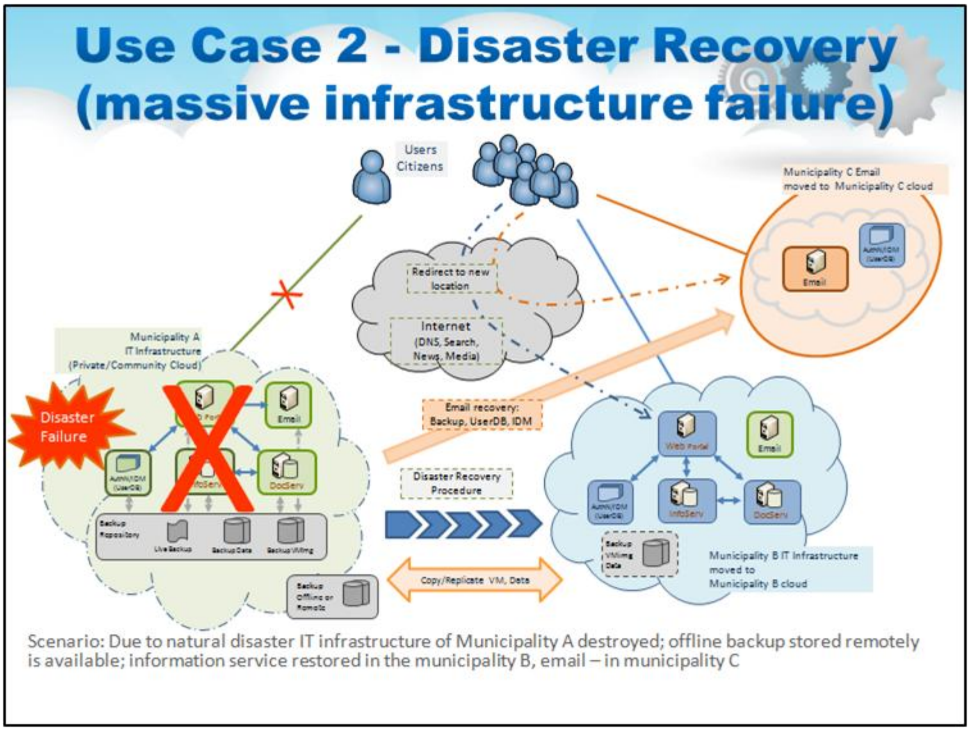
Municipality A information services and email starting working on emergency mode; when original facility and datacenter are restored, services will be migrated to original location

Challenges:

Full services backup and restoration must also include infrastructure and services topology

Compatibility and standards for VM images, Data, service description and topology

Compatible cloud platforms in Municipality A, B, C

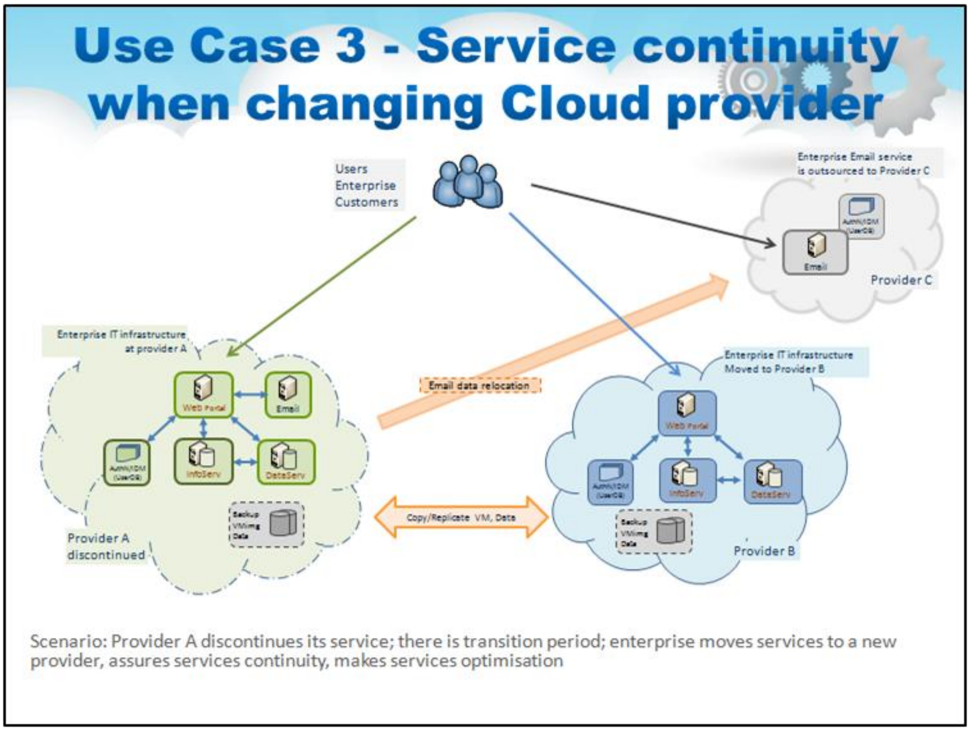


Use Case 2 - Disaster Recovery (massive infrastructure failure)

Scenario: Due to natural disaster IT infrastructure of Municipality A destroyed; offline backup stored remotely is available; information service restored in the municipality B, email – in municipality C

The following preconditions are suggested for this scenario to work successfully: Municipalities' IT infrastructures are cloud based, e.g. using community cloud deployment model The whole IT infrastructure is backed up regularly, including VMs of all applications and services, data, UserDB, and infrastructure topology Data and backups are replicated to/or stored remotely.

The success of the described here disaster recovery scenario depends on addressing the following challenges: Compatible cloud platforms in Municipality A, B, C Compatibility and common standards for VM images, data, and services description. Full services backup and restoration must also include infrastructure and services topology.



Use Case 3 - Service continuity when changing Cloud provider

Scenario: Provider A discontinues its service; there is transition period; enterprise moves services to a new provider, assures services continuity, makes services optimisation

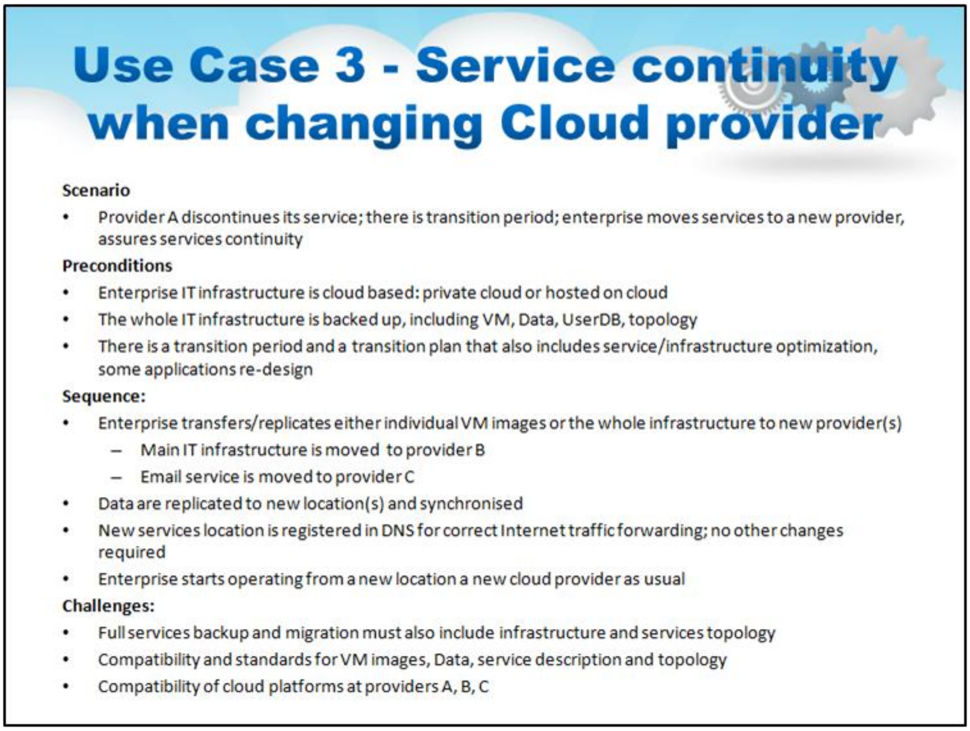
Use case 3: Service continuity when changing cloud provider.

This use case illustrates what are the main tasks and challenges when moving from one cloud service provider to another. Such situation may happen when the current provider discontinues its service or the customer decided to move to another provider because of a number of reasons, e.g. cost of services, available services, regulation requirements that may restrict location of the provider's data center.

Actually, the scenario with the service migration to another provider should be discussed when planning cloud technology implementation by enterprise, to avoid possible problems with the provider lock-in what is still typical in cloud business.

The slide illustrates the IT infrastructure migration scenario. The following steps describe the migration process: Enterprise transfers/replicates either individual VM images or the whole infrastructure to new provider(s), in our case.

Main IT infrastructure is moved to provider B. Email service is moved to provider C Data are replicated to new location(s) and synchronized. New services location is registered in DNS for correct Internet traffic forwarding; no other changes required Enterprise services start operating from the new cloud providers as usual.



Use Case 3 - Service continuity when changing Cloud provider

Scenario

- Provider A discontinues its service; there is transition period; enterprise moves services to a new provider, assures services continuity

Preconditions

- Enterprise IT infrastructure is cloud based: private cloud or hosted on cloud
- The whole IT infrastructure is backed up, including VM, Data, UserDB, topology
- There is a transition period and a transition plan that also includes service/infrastructure optimization, some applications re-design

Sequence:

- Enterprise transfers/replicates either individual VM images or the whole infrastructure to new provider(s)
  - Main IT infrastructure is moved to provider B
  - Email service is moved to provider C
- Data are replicated to new location(s) and synchronised
- New services location is registered in DNS for correct Internet traffic forwarding; no other changes required
- Enterprise starts operating from a new location a new cloud provider as usual

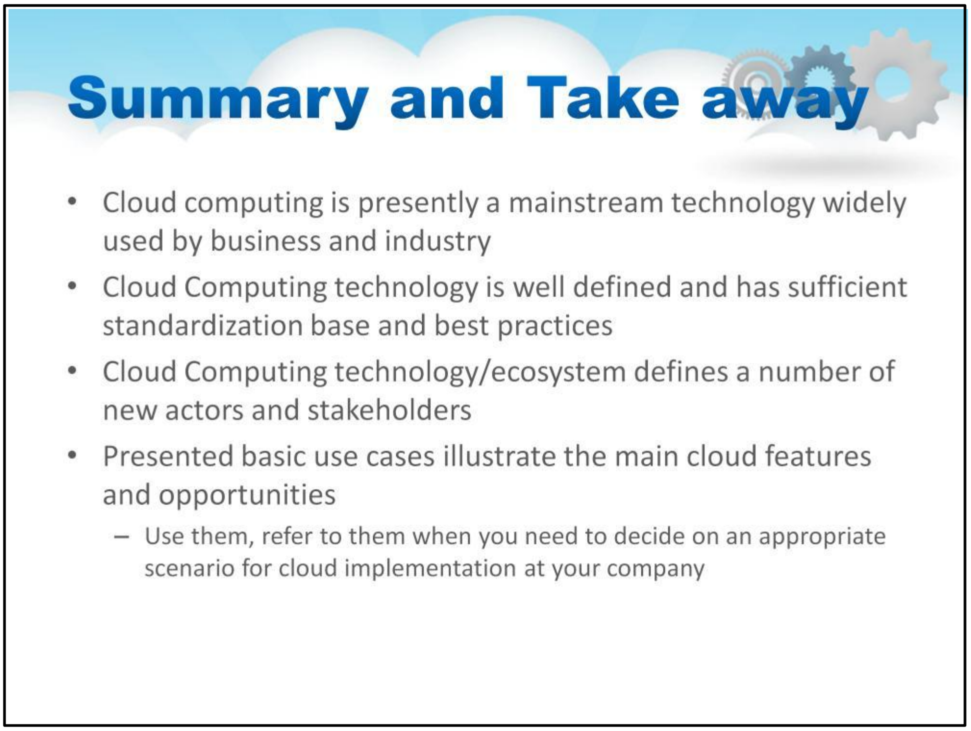
Challenges:

- Full services backup and migration must also include infrastructure and services topology
- Compatibility and standards for VM images, Data, service description and topology
- Compatibility of cloud platforms at providers A, B, C

To finalize let's emphasize the crucial points of this Use Case 3:

The migration process should be well planned and there will be a transition period. The following preconditions should assured: Enterprise IT infrastructure is cloud based: private cloud or hosted on cloud The whole IT infrastructure is backed up, including VM, Data, UserDB, infrastructure or services topology The transition plan may also include the services/infrastructure optimization, some applications re-design.

This use case has similar challenges as in our cases: Compatibility of cloud platforms at providers A, B, C Compatibility and standards for VM images, Data, service description and topology Full and up to date services backup, data synchronization at the moment of the service switch to a new location.



Summary and Take away

- Cloud computing is presently a mainstream technology widely used by business and industry
- Cloud Computing technology is well defined and has sufficient standardization base and best practices
- Cloud Computing technology/ecosystem defines a number of new actors and stakeholders
- Presented basic use cases illustrate the main cloud features and opportunities
  - Use them, refer to them when you need to decide on an appropriate scenario for cloud implementation at your company

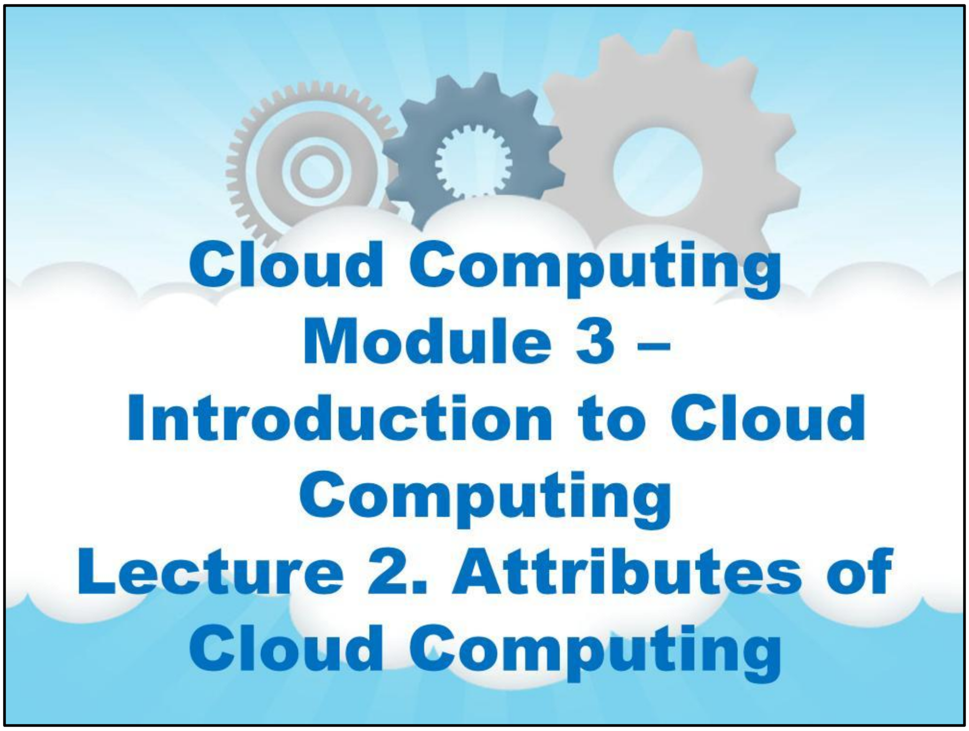
Cloud computing is presently a mainstream technology widely used by business and industry

Cloud Computing technology is well defined and has sufficient standardization base and best practices

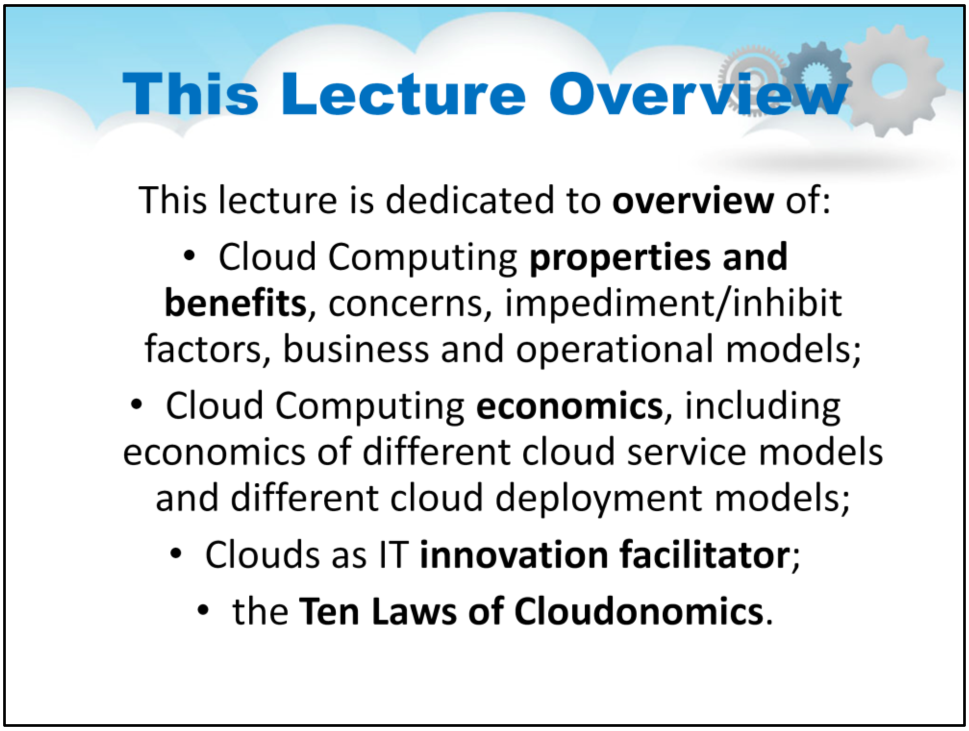
Cloud Computing technology/ecosystem defines a number of new actors and stakeholders

Presented basic use cases illustrate the main cloud features and opportunities

Use them, refer to them when you need to decide on an appropriate scenario for cloud implementation at your company



Cloud Computing Module 3 – Introduction to Cloud Computing Lecture 2. Attributes of Cloud Computing

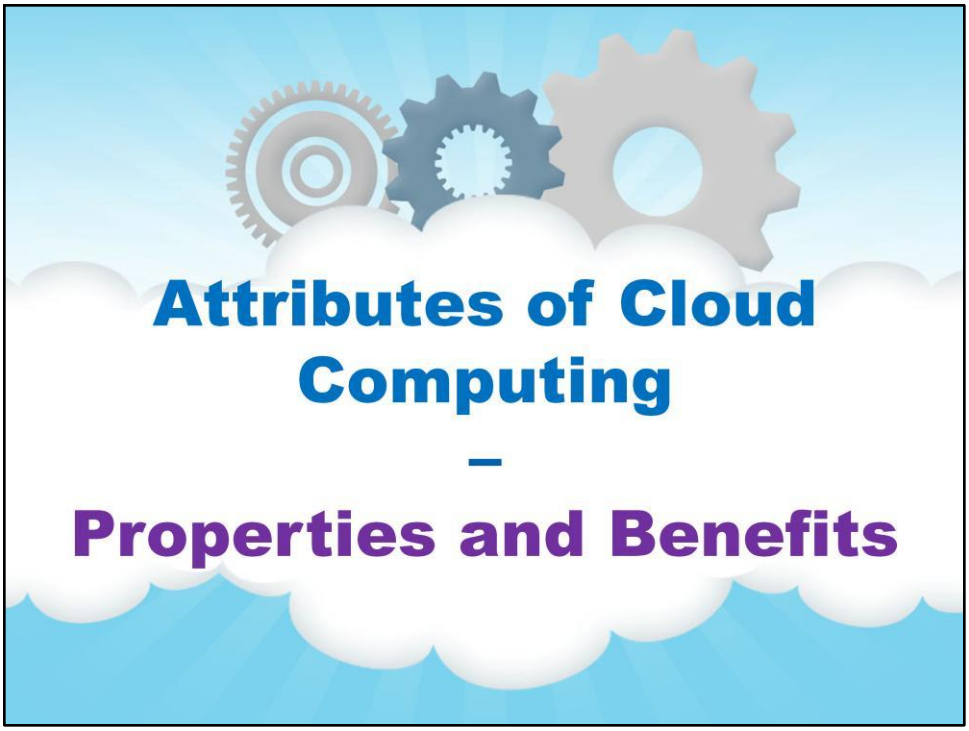


This Lecture Overview

This lecture is dedicated to **overview** of:

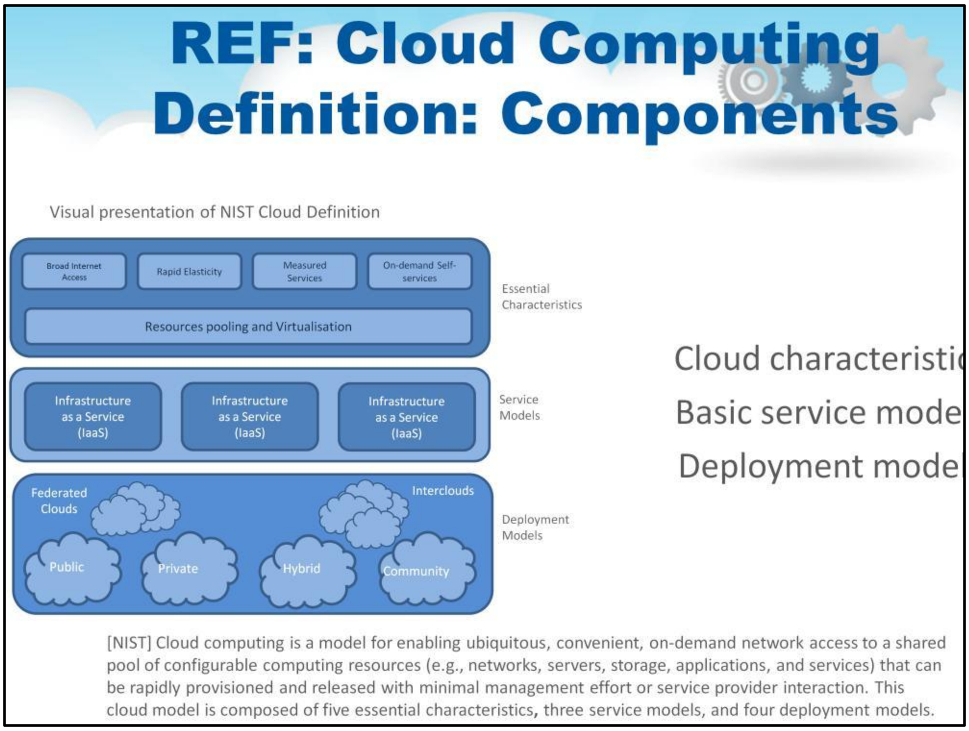
- Cloud Computing **properties and benefits**, concerns, impediment/inhibit factors, business and operational models;
- Cloud Computing **economics**, including economics of different cloud service models and different cloud deployment models;
- Clouds as IT **innovation facilitator**;
- the **Ten Laws of Cloudonomics**.

Lecture 2. Attributes of Cloud Computing



Attributes of Cloud Computing
–
Properties and Benefits

Properties and Benefits



REF: Cloud Computing Definition: Components

Visual presentation of NIST Cloud Definition

Broad Internet Access

Rapid Elasticity

Measured Services

On-demand Self-services

Resources pooling and Virtualisation

Essential Characteristics

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS)

Service Models

Federated Clouds

Interclouds

Public

Private

Hybrid

Community

Deployment Models

Cloud characteristic

Basic service mode

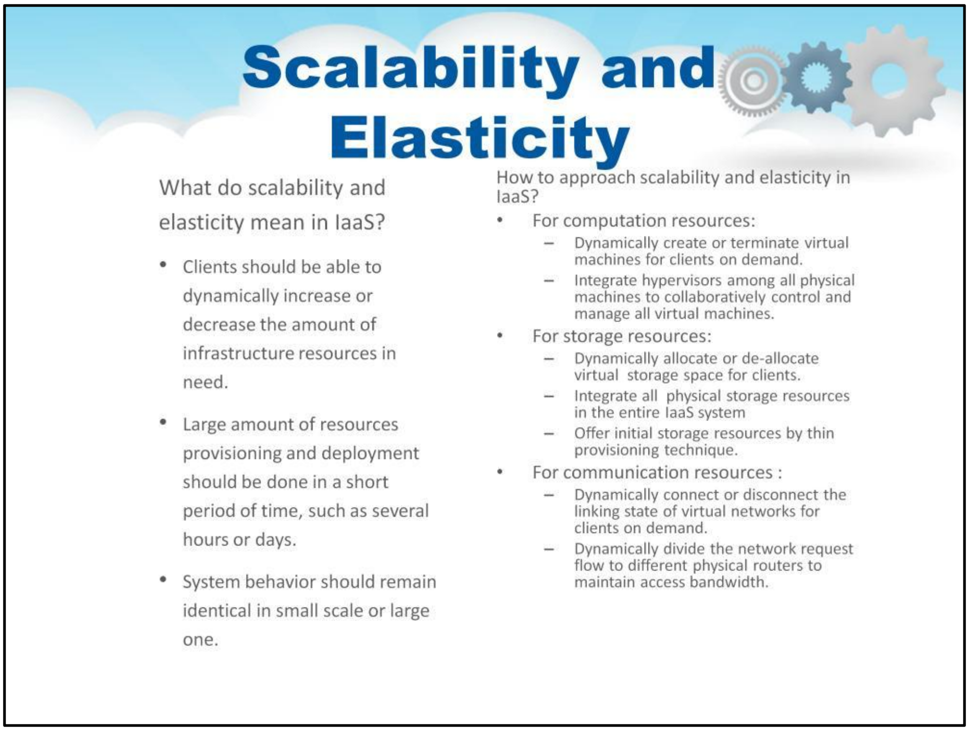
Deployment mode

[NIST] Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Cloud characteristics
- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured Service

Basic service models
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Deployment models
- Private clouds
- Public clouds
- Hybrid clouds
- Community clouds
- Federated clouds, Interclouds



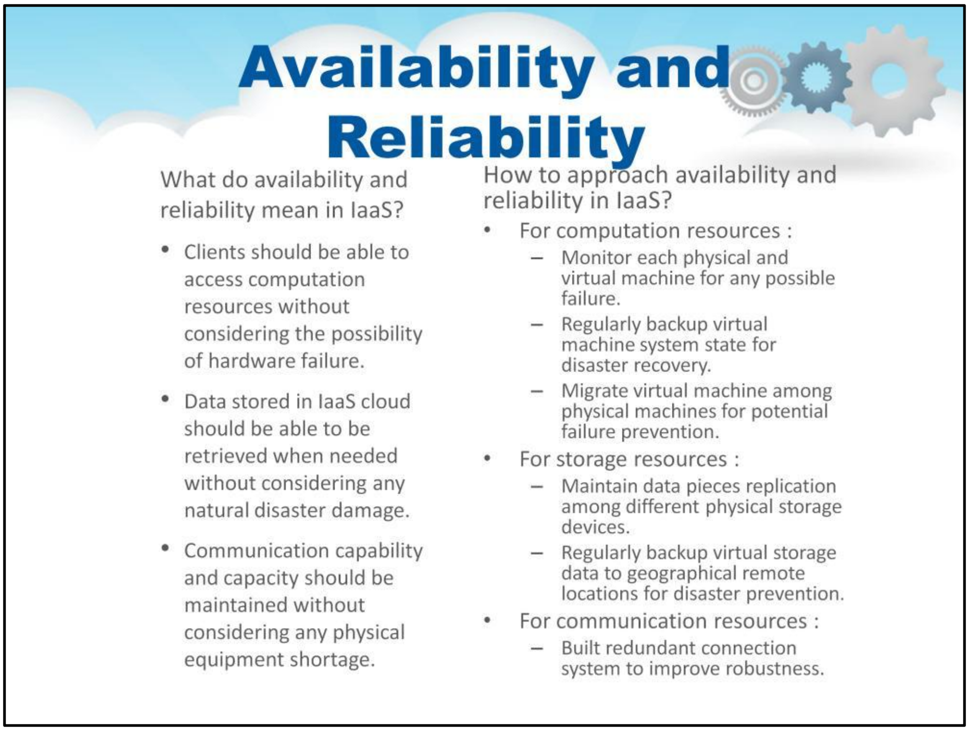
Scalability and Elasticity

What do scalability and elasticity mean in IaaS?

- Clients should be able to dynamically increase or decrease the amount of infrastructure resources in need.
- Large amount of resources provisioning and deployment should be done in a short period of time, such as several hours or days.
- System behavior should remain identical in small scale or large one.

How to approach scalability and elasticity in IaaS?

- For computation resources:
  - Dynamically create or terminate virtual machines for clients on demand.
  - Integrate hypervisors among all physical machines to collaboratively control and manage all virtual machines.
- For storage resources:
  - Dynamically allocate or de-allocate virtual storage space for clients.
  - Integrate all physical storage resources in the entire IaaS system
  - Offer initial storage resources by thin provisioning technique.
- For communication resources :
  - Dynamically connect or disconnect the linking state of virtual networks for clients on demand.
  - Dynamically divide the network request flow to different physical routers to maintain access bandwidth.



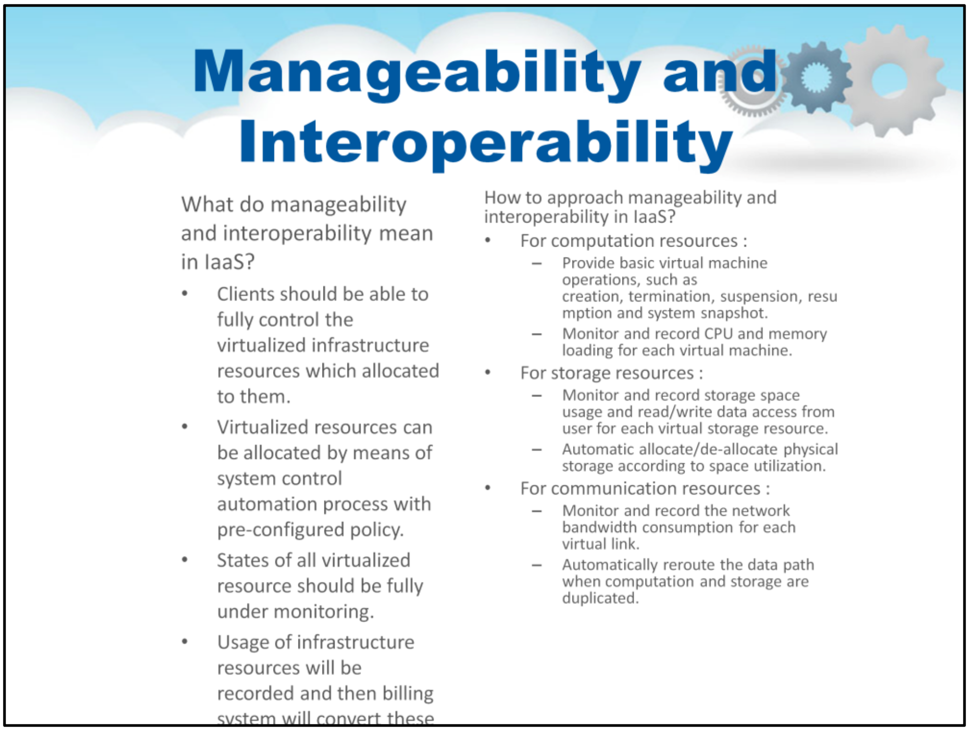
Availability and Reliability

What do availability and reliability mean in IaaS?

- Clients should be able to access computation resources without considering the possibility of hardware failure.
- Data stored in IaaS cloud should be able to be retrieved when needed without considering any natural disaster damage.
- Communication capability and capacity should be maintained without considering any physical equipment shortage.

How to approach availability and reliability in IaaS?

- For computation resources :
  - Monitor each physical and virtual machine for any possible failure.
  - Regularly backup virtual machine system state for disaster recovery.
  - Migrate virtual machine among physical machines for potential failure prevention.
- For storage resources :
  - Maintain data pieces replication among different physical storage devices.
  - Regularly backup virtual storage data to geographical remote locations for disaster prevention.
- For communication resources :
  - Built redundant connection system to improve robustness.



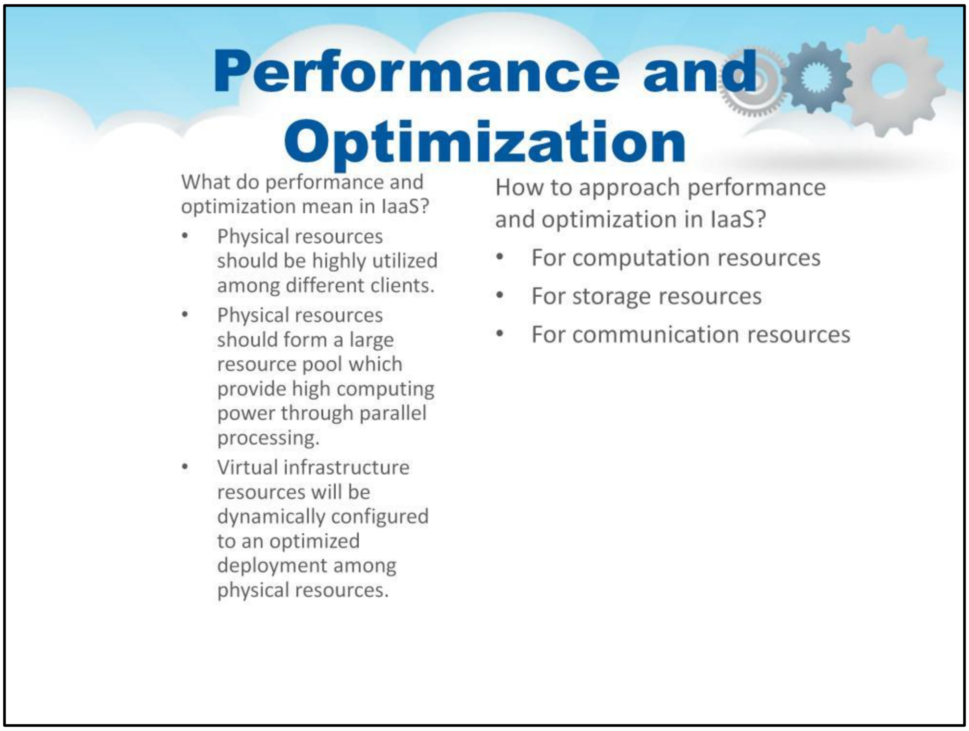
Manageability and Interoperability

What do manageability and interoperability mean in IaaS?

- Clients should be able to fully control the virtualized infrastructure resources which allocated to them.
- Virtualized resources can be allocated by means of system control automation process with pre-configured policy.
- States of all virtualized resource should be fully under monitoring.
- Usage of infrastructure resources will be recorded and then billing system will convert these

How to approach manageability and interoperability in IaaS?

- For computation resources :
  - Provide basic virtual machine operations, such as creation, termination, suspension, resu mption and system snapshot.
  - Monitor and record CPU and memory loading for each virtual machine.
- For storage resources :
  - Monitor and record storage space usage and read/write data access from user for each virtual storage resource.
  - Automatic allocate/de-allocate physical storage according to space utilization.
- For communication resources :
  - Monitor and record the network bandwidth consumption for each virtual link.
  - Automatically reroute the data path when computation and storage are duplicated.



Performance and Optimization

What do performance and optimization mean in IaaS?

- Physical resources should be highly utilized among different clients.
- Physical resources should form a large resource pool which provide high computing power through parallel processing.
- Virtual infrastructure resources will be dynamically configured to an optimized deployment among physical resources.

How to approach performance and optimization in IaaS?

- For computation resources
- For storage resources
- For communication resources

For computation resources :

Deploy virtual machine with load balancing consideration.

Live migrate virtual machines among physical ones to balance the system loading.

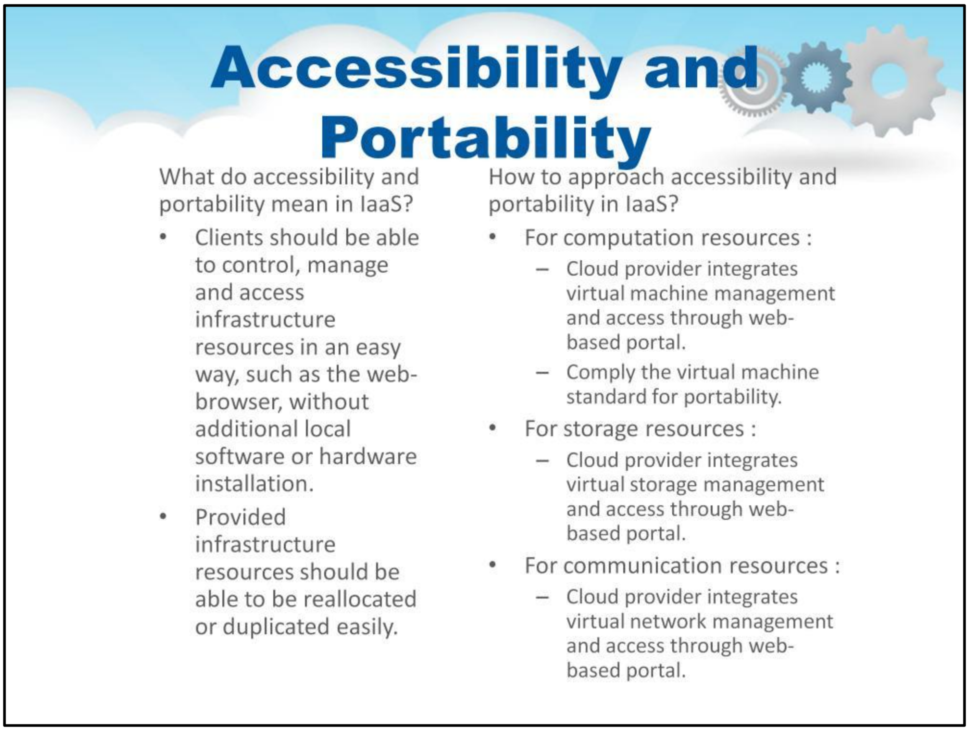
For storage resources :

Deploy virtual storage with hot spot access consideration.

Live migrate virtual storage among physical ones with different performance level.

For communication resources :

Consider network bandwidth loading when deploying virtual machines and storage.



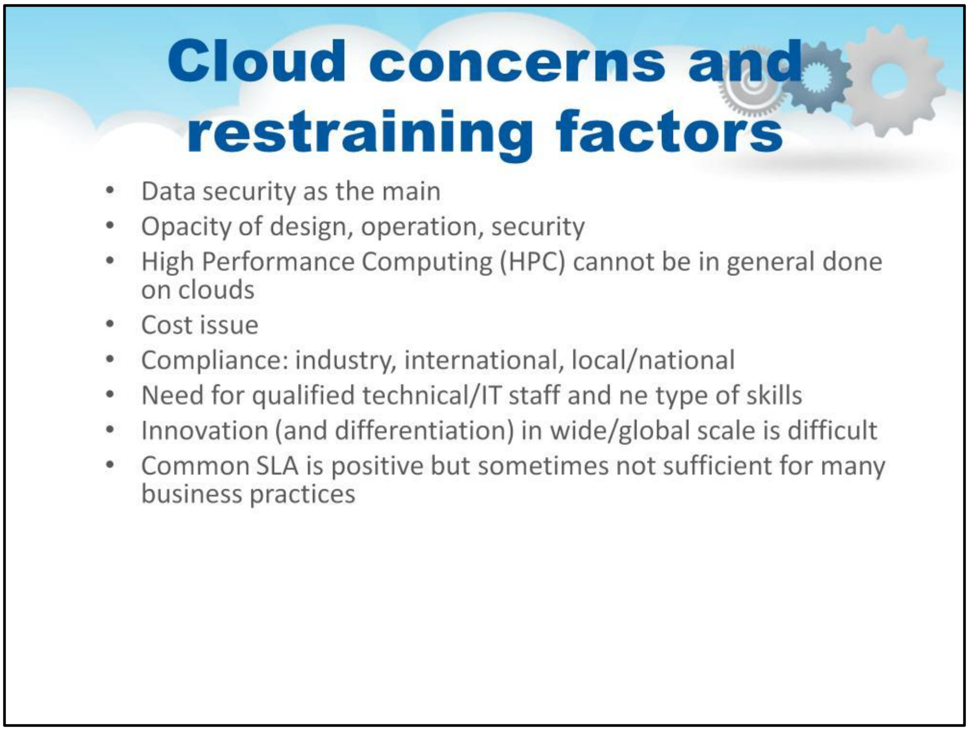
Accessibility and Portability

What do accessibility and portability mean in IaaS?

- Clients should be able to control, manage and access infrastructure resources in an easy way, such as the web-browser, without additional local software or hardware installation.
- Provided infrastructure resources should be able to be reallocated or duplicated easily.

How to approach accessibility and portability in IaaS?

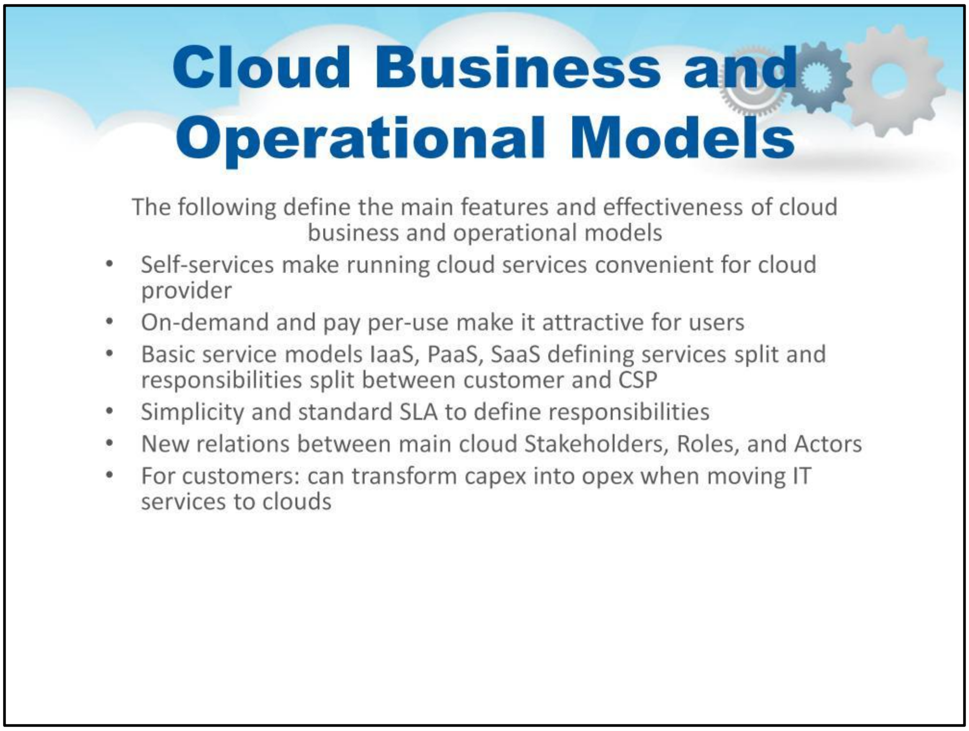
- For computation resources :
  - Cloud provider integrates virtual machine management and access through web-based portal.
  - Comply the virtual machine standard for portability.
- For storage resources :
  - Cloud provider integrates virtual storage management and access through web-based portal.
- For communication resources :
  - Cloud provider integrates virtual network management and access through web-based portal.



Cloud concerns and restraining factors

- Data security as the main
- Opacity of design, operation, security
- High Performance Computing (HPC) cannot be in general done on clouds
- Cost issue
- Compliance: industry, international, local/national
- Need for qualified technical/IT staff and ne type of skills
- Innovation (and differentiation) in wide/global scale is difficult
- Common SLA is positive but sometimes not sufficient for many business practices

- Data security as the main
  - Where my data?
  - What is happening with my data? Who has access to my data?
  - Cloud eDiscovery and data destroying
- Opacity of design, operation, security
  - Often referred to as “cloud curtain”
- High Performance Computing (HPC) cannot be in general done on clouds
  - Increasing factor due to Big Data advent
- Cost issue
  - Total Ownership Cost (TOC) is not always less comparing to traditional IT
- Innovation (and differentiation) in wide/global scale is difficult
  - Innovation pace in developing countries is significantly slower
- Common SLA is positive but sometimes not sufficient for many business practices
  - Some cloud concerns can be resolved by customised SLA but may involved signing NDA
    - Contact provider for modification and customized services



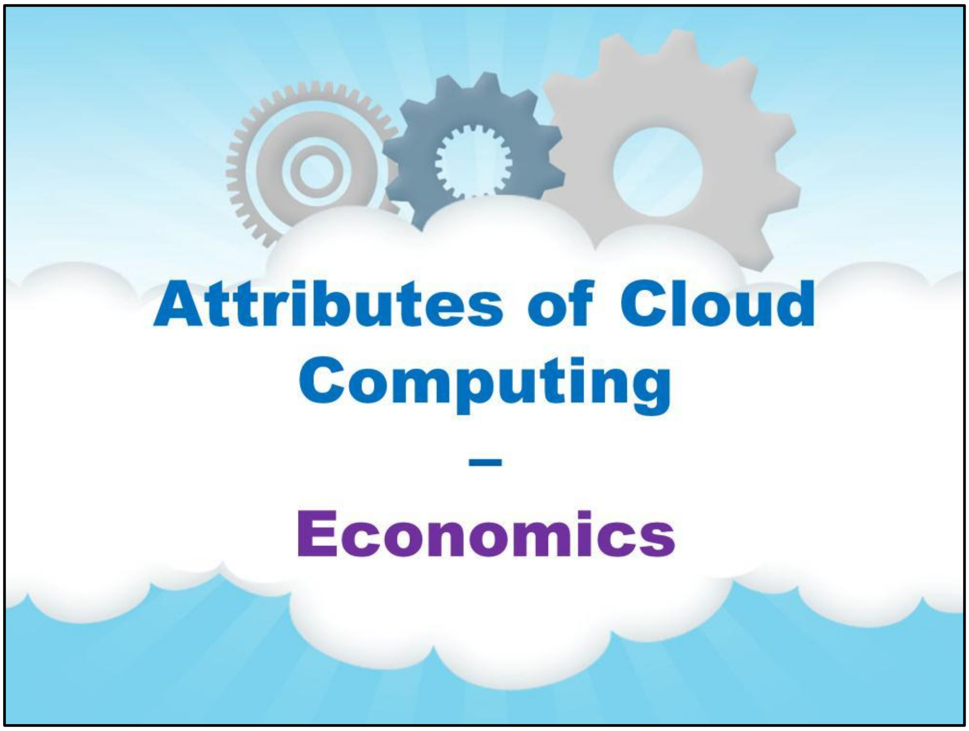
Cloud Business and Operational Models

The following define the main features and effectiveness of cloud business and operational models

- Self-services make running cloud services convenient for cloud provider
- On-demand and pay per-use make it attractive for users
- Basic service models IaaS, PaaS, SaaS defining services split and responsibilities split between customer and CSP
- Simplicity and standard SLA to define responsibilities
- New relations between main cloud Stakeholders, Roles, and Actors
- For customers: can transform capex into opex when moving IT services to clouds

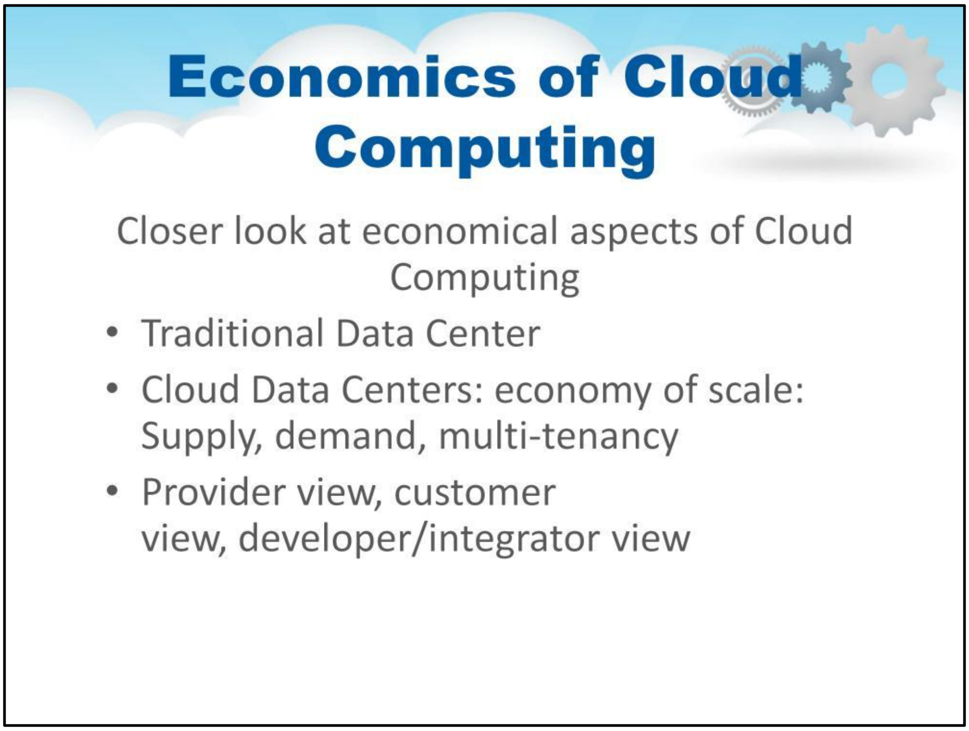
For customers: can transform capex into opex when moving IT services to clouds:

Plus decrease IT management, backup and IT staff costs



Attributes of Cloud Computing – Economics

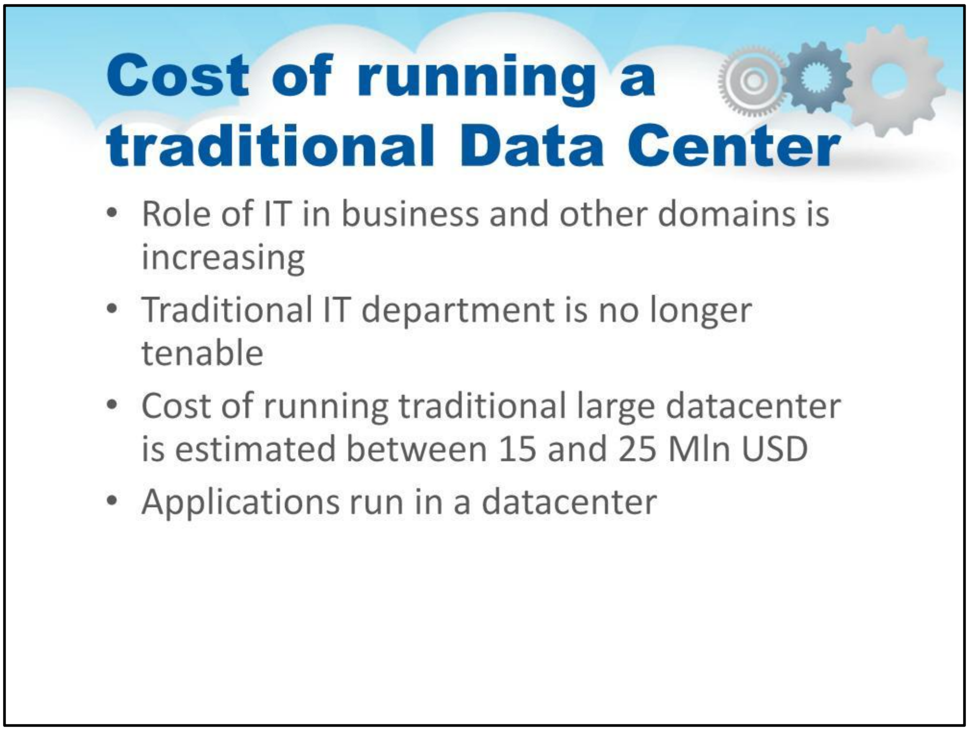
Economics



Economics of Cloud Computing

Closer look at economical aspects of Cloud Computing

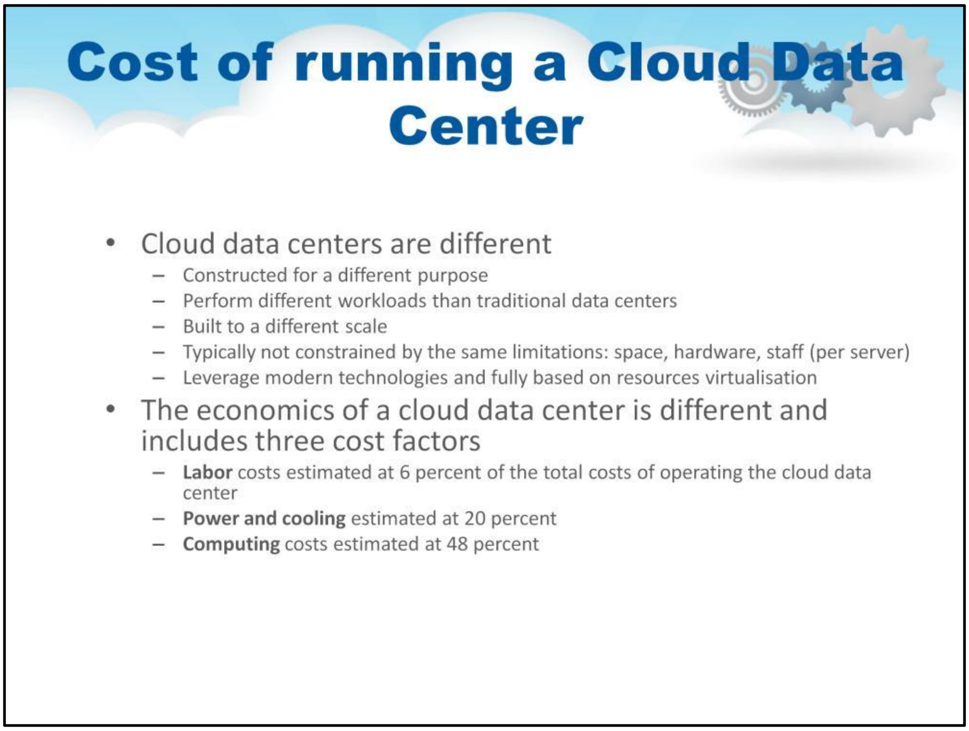
- Traditional Data Center
- Cloud Data Centers: economy of scale: Supply, demand, multi-tenancy
- Provider view, customer view, developer/integrator view



Cost of running a traditional Data Center

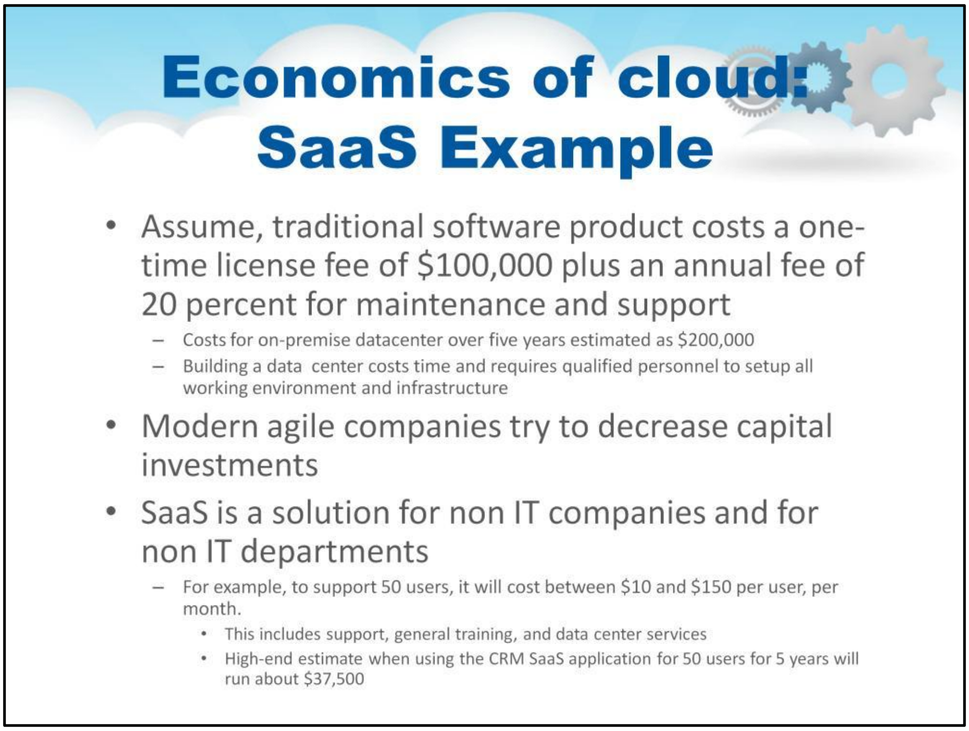
- Role of IT in business and other domains is increasing
- Traditional IT department is no longer tenable
- Cost of running traditional large datacenter is estimated between 15 and 25 Mln USD
- Applications run in a datacenter

- Role of IT in business and other domains is increasing
  - It facilitates changes in how work/business is done and increases business agility
- Traditional IT department is no longer tenable
  - It is difficult predict for longer term (as earlier) IT needs with current rush increase in digital applications as a primary method of doing business and interacting with customers
- Cost of running traditional large datacenter is estimated between 15 and 25 Mln USD
  - 42 percent: Hardware, software, disaster recovery arrangements, uninterrupted power supplies, and networking.
    - Costs are spread over time, amortized, because they are a combination of capital expenditures and regular payments.
  - 58 percent: Heating, air conditioning, property and sales taxes, and labor costs. (In fact, as much as 40 percent of annual costs are labor alone)
- Applications run in a datacenter
  - Most data centers run a lot of different applications and have a wide variety of workloads.
  - Many of the most important applications running in data centers are actually used by only a relatively few employees.
  - Some applications that run on older systems are already taken off the market (no longer sold) but are still necessary for business.



Cost of running a Cloud Data Center

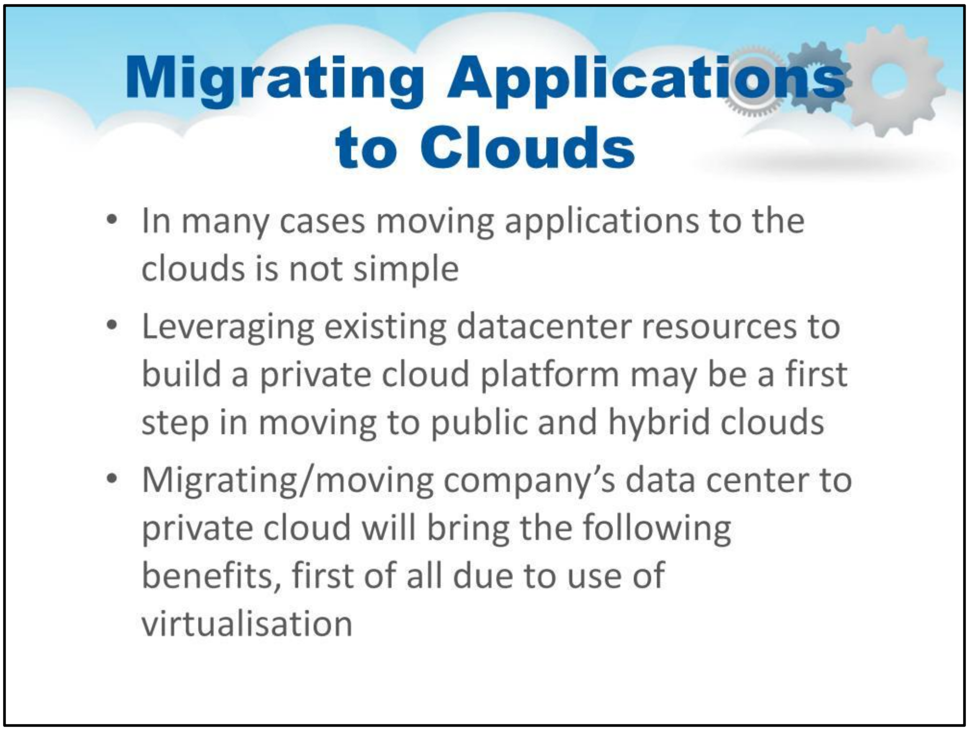
- Cloud data centers are different
  - Constructed for a different purpose
  - Perform different workloads than traditional data centers
  - Built to a different scale
  - Typically not constrained by the same limitations: space, hardware, staff (per server)
  - Leverage modern technologies and fully based on resources virtualisation
- The economics of a cloud data center is different and includes three cost factors
  - **Labor** costs estimated at 6 percent of the total costs of operating the cloud data center
  - **Power and cooling** estimated at 20 percent
  - **Computing** costs estimated at 48 percent



Economics of cloud: SaaS Example

- Assume, traditional software product costs a one-time license fee of $100,000 plus an annual fee of 20 percent for maintenance and support
  - Costs for on-premise datacenter over five years estimated as $200,000
  - Building a data center costs time and requires qualified personnel to setup all working environment and infrastructure
- Modern agile companies try to decrease capital investments
- SaaS is a solution for non IT companies and for non IT departments
  - For example, to support 50 users, it will cost between $10 and $150 per user, per month.
    - This includes support, general training, and data center services
    - High-end estimate when using the CRM SaaS application for 50 users for 5 years will run about $37,500

Please, for self-guide work, find the details in
"The Essentials of Services in Cloud Computing"
http://www.dummies.com/how-to/content/understanding-the-economics-of-saas-in-cloud-compu.html

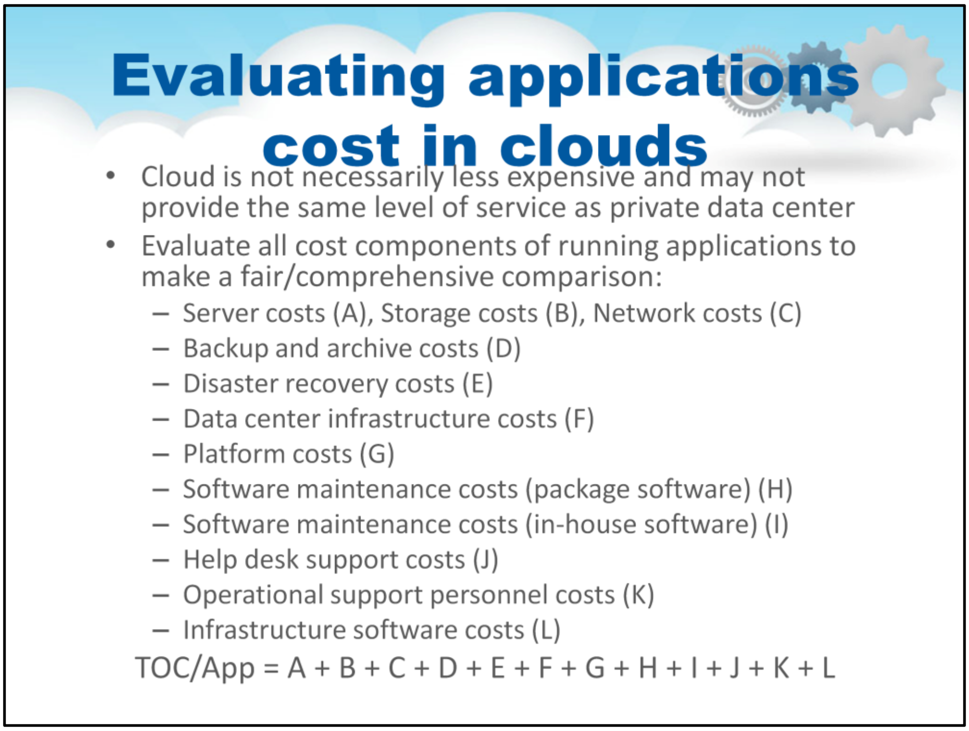


Migrating Applications to Clouds

- In many cases moving applications to the clouds is not simple
- Leveraging existing datacenter resources to build a private cloud platform may be a first step in moving to public and hybrid clouds
- Migrating/moving company's data center to private cloud will bring the following benefits, first of all due to use of virtualisation

- In many cases moving applications to the clouds is not simple
  - Some configuration changing work to be done
  - May require applications redesign and testing period
  - Estimate costs if application is not available for cloud
  - The same cost factors apply for both private and public clouds
  - Compliance can be another cost factor, in particular related to security and recovery procedure

Leveraging existing datacenter resources to build a private cloud platform may be a first step in moving to public and hybrid clouds Migrating/moving company's data center to private cloud will bring the following benefits, first of all due to use of virtualisation.



Evaluating applications cost in clouds

- Cloud is not necessarily less expensive and may not provide the same level of service as private data center
- Evaluate all cost components of running applications to make a fair/comprehensive comparison:
  - Server costs (A), Storage costs (B), Network costs (C)
  - Backup and archive costs (D)
  - Disaster recovery costs (E)
  - Data center infrastructure costs (F)
  - Platform costs (G)
  - Software maintenance costs (package software) (H)
  - Software maintenance costs (in-house software) (I)
  - Help desk support costs (J)
  - Operational support personnel costs (K)
  - Infrastructure software costs (L)

TOC/App = A + B + C + D + E + F + G + H + I + J + K + L

- Cloud is not necessarily less expensive and may not provide the same level of service as private data center
  - Own data center may have a service level agreement with a 99.999 percent uptime record.
  - Will the cloud provider offer that same level of service? Probably not.
  - The company have to weigh how critical the level of predictable uptime is to internal users and customers

Anticipate that the cloud won't necessarily be less expensive and it won't necessarily provide the same level of service as your data center.

Your own data center may have a service level agreement with a 99.999 percent uptime record. Will your cloud provider offer that same level of service? Probably not. You have to weigh how critical that level of predictable uptime is to your internal customers. Evaluate all cost components of running applications to make a fair/comprehensive comparison:

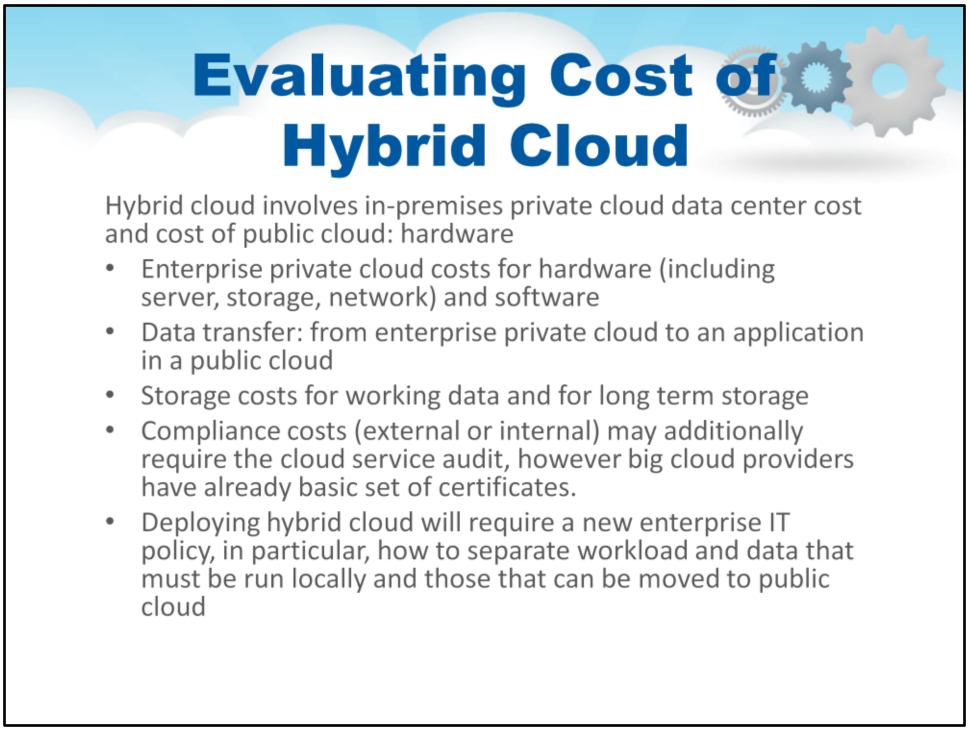
Server costs (A):

With this and all other hardware components, you're specifically interested in the total annual cost of ownership, which normally consists of the cost of hardware support plus some amortization cost for the purchase of the hardware.

Storage costs (B):

In situations where a storage area network (SAN) or network attached store(NAS) is used for an application, a proportional cost over the whole SAN or NAS needs to be determined, including management and support cost for the hardware.

Network costs (C):



Evaluating Cost of Hybrid Cloud

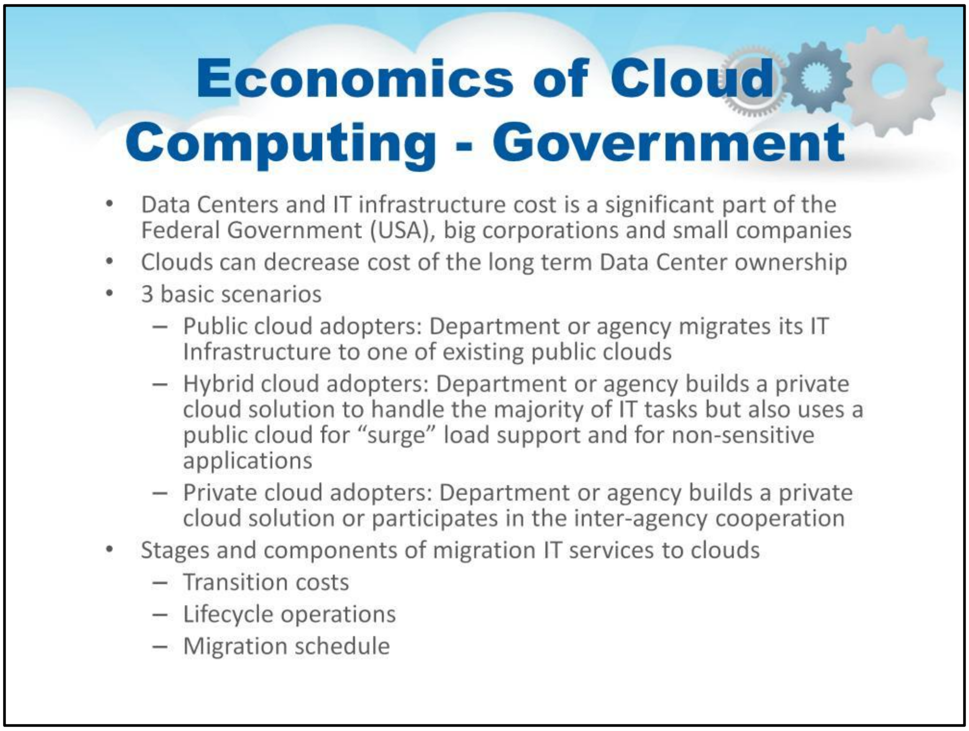
Hybrid cloud involves in-premises private cloud data center cost and cost of public cloud: hardware

- Enterprise private cloud costs for hardware (including server, storage, network) and software
- Data transfer: from enterprise private cloud to an application in a public cloud
- Storage costs for working data and for long term storage
- Compliance costs (external or internal) may additionally require the cloud service audit, however big cloud providers have already basic set of certificates.
- Deploying hybrid cloud will require a new enterprise IT policy, in particular, how to separate workload and data that must be run locally and those that can be moved to public cloud

- Enterprise private cloud costs for hardware (including server, storage, network) and software
  - In premises data center must also run cloud software • Only part of workload workloads will move into cloud
  - Customization costs for applications to work effectively on cloud
  - Integration costs between off-premises and on-premises applications
  - Operational support personnel costs will decrease for private cloud data center

- Compliance costs (external or internal) may additionally require the cloud service audit, however big cloud providers have already basic set of certificates.
  - Compliance requirements may be defined by several bodies. e.g. PCI DSS, HIPAA, SOX, company internal

- Deploying hybrid cloud will require a new enterprise IT policy, in particular, how to separate workload and data that must be run locally and those that can be moved to public cloud
  - Data security, safety and privacy must be managed differently for in-premises and in public cloud



Economics of Cloud Computing - Government

- Data Centers and IT infrastructure cost is a significant part of the Federal Government (USA), big corporations and small companies
- Clouds can decrease cost of the long term Data Center ownership
- 3 basic scenarios
  - Public cloud adopters: Department or agency migrates its IT Infrastructure to one of existing public clouds
  - Hybrid cloud adopters: Department or agency builds a private cloud solution to handle the majority of IT tasks but also uses a public cloud for “surge” load support and for non-sensitive applications
  - Private cloud adopters: Department or agency builds a private cloud solution or participates in the inter-agency cooperation
- Stages and components of migration IT services to clouds
  - Transition costs
  - Lifecycle operations
  - Migration schedule

- Data Centers and IT infrastructure cost is a significant part of the Federal Government (USA), big corporations and small companies
  - IT spending is increasing: $75.88B in FY10 (Fiscal Year 2010); $88B in FY11
  - Estimated as $518 Bln over 2013-2018 with CAGR 3% **)
  - Search for long term saving solution: pressure to cut federal IT budgets without losing efficiency and the implementation of new technologies: agility, “rent, not buy”

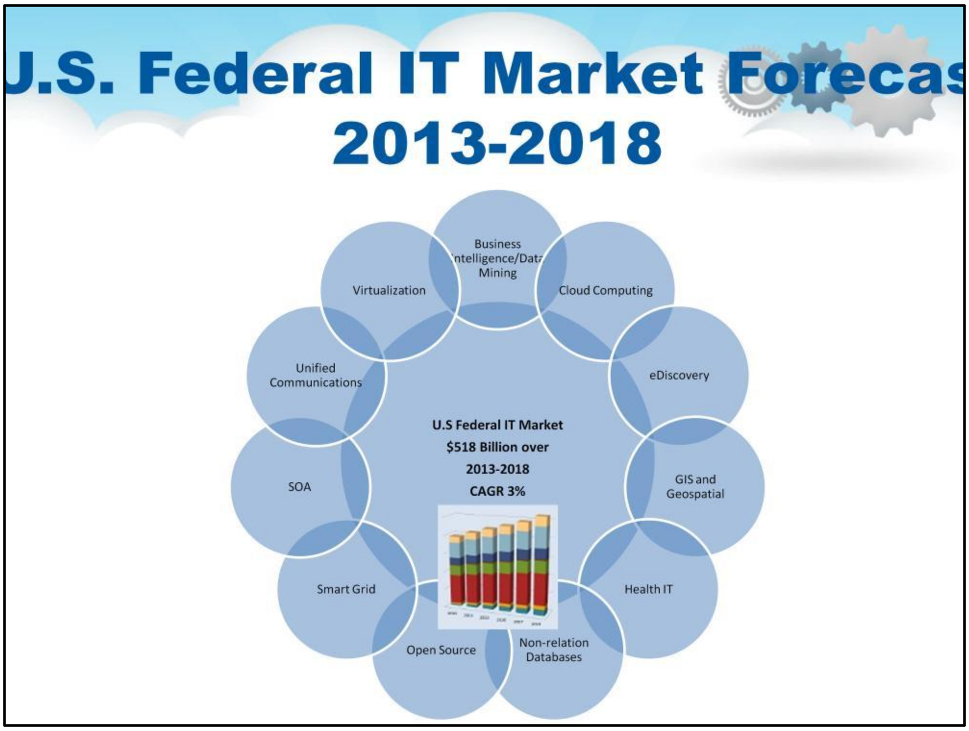
Clouds can decrease cost of the long term Data Center ownership

- Implementing and sustaining cloud environment over 13 year cycle up to two-thirds economy

- 3 basic scenarios
  - Public cloud adopters: Department or agency migrates its IT Infrastructure to one of existing public clouds
- No specific sensitive data, transition period extends 1-2 years
  - Hybrid cloud adopters: Department or agency builds a private cloud solution to handle the majority of IT tasks but also uses a public cloud for “surge” load support and for non-sensitive applications

    Estimated share between private and public cloud workload 75%/25%, private cloud is built on existing IT hardware, transition period extends 1-2 years
  - Private cloud adopters: Department or agency builds a private cloud solution or participates in the inter-agency cooperation

    Late adopters: Mission sensitive data, private built on existing IT hardware, transition period extends 1-2 years



.S. Federal IT Market Forecas 2013-2018

Business Intelligence/Data Mining

Cloud Computing

eDiscovery

GIS and Geospatial

Health IT

Non-relation Databases

Open Source

Smart Grid

SOA

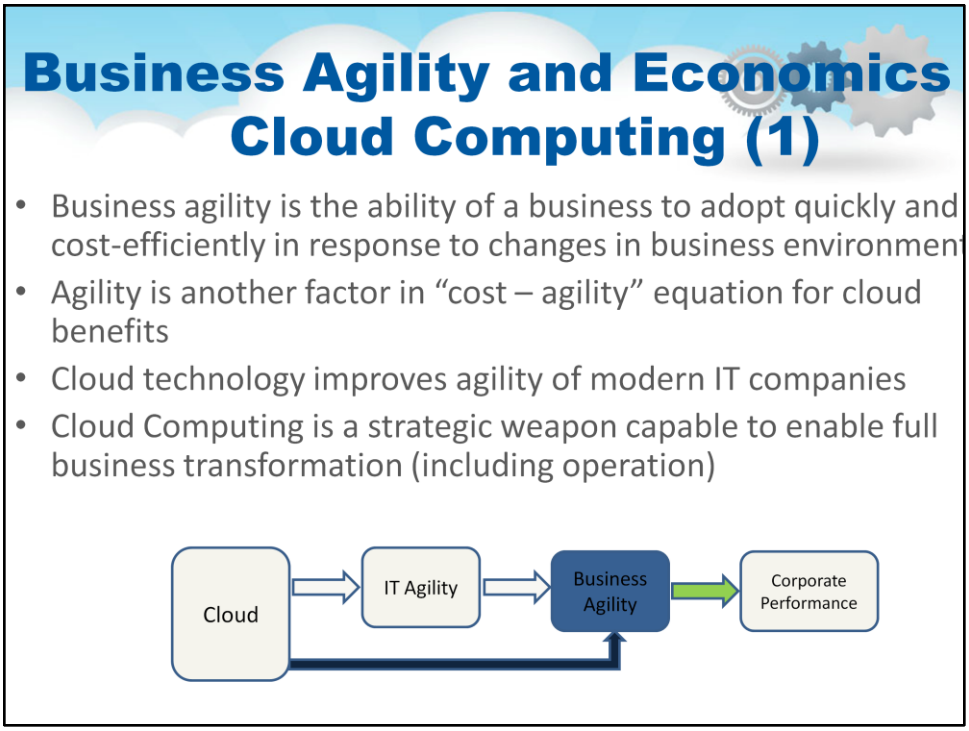
Unified Communications

Virtualization

U.S Federal IT Market
$518 Billion over
2013-2018
CAGR 3%

Fast growing sectors

- Business Intelligence
- Cloud Computing
- Cyber Security
- Deduplication
- eDiscovery
- GIS and Geospatial
- Health Information Technology (HIT)
- High Performance Computing
- Non-Relational Database Management Systems
- Open Source
- Smart Grid
- SOA
- Virtual Events
- Virtualization
- Wireless Voice and Data.



Business Agility and Economics Cloud Computing (1)

- Business agility is the ability of a business to adopt quickly and cost-efficiently in response to changes in business environment
- Agility is another factor in "cost – agility" equation for cloud benefits
- Cloud technology improves agility of modern IT companies
- Cloud Computing is a strategic weapon capable to enable full business transformation (including operation)

Cloud

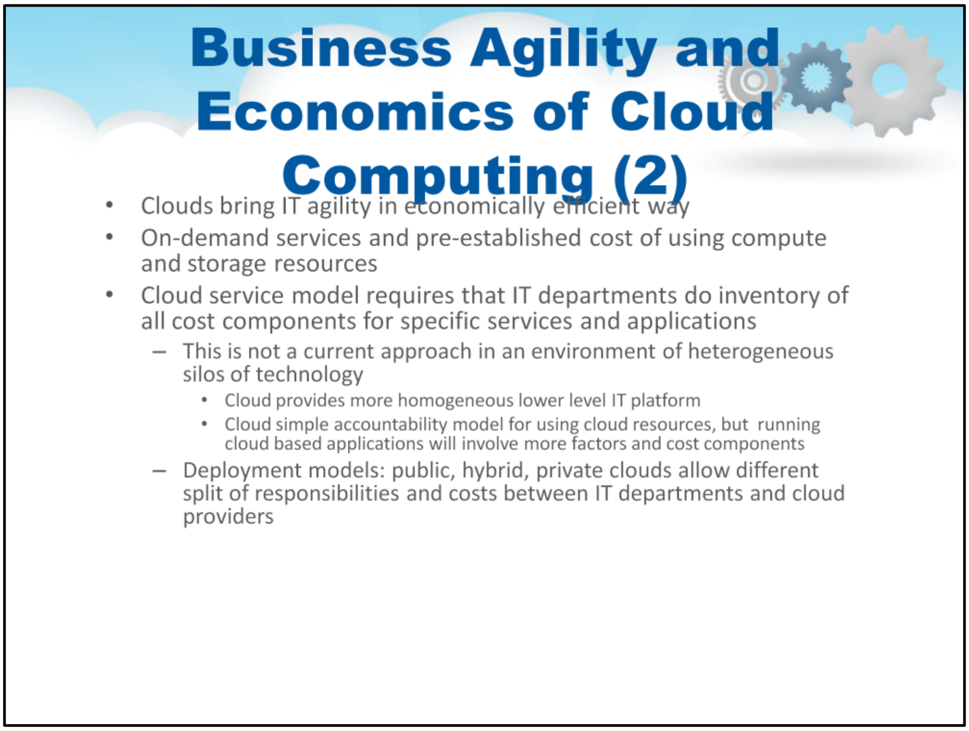
IT Agility

Business Agility

Corporate Performance

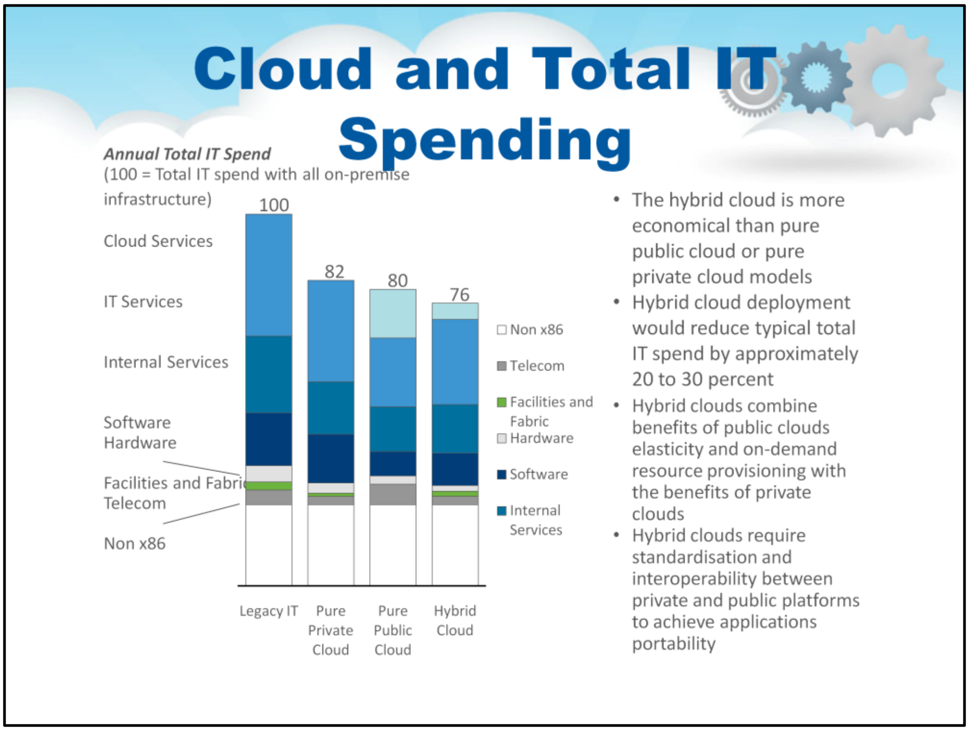
- Agility is another factor in "cost – agility" equation for cloud benefits
  - "How will cloud improve my company's competeveness?"
  - Not just "How much capital and operations expenses can cloud cut?"
- Cloud technology improves agility of modern IT companies
  - Strong link between IT agility and business agility
  - Continuous business transformation and technologies development is a current trend and a new economy reality
    - Requires faster responsiveness
  - (According to McKensey&Company) Benefits of agility includes faster revenue growth, greater and long lasting cost reduction, more effective risks management

Source: VMware Business white paper



Business Agility and Economics of Cloud Computing (2)

- Clouds bring IT agility in economically efficient way
- On-demand services and pre-established cost of using compute and storage resources
- Cloud service model requires that IT departments do inventory of all cost components for specific services and applications
  - This is not a current approach in an environment of heterogeneous silos of technology
    - Cloud provides more homogeneous lower level IT platform
    - Cloud simple accountability model for using cloud resources, but running cloud based applications will involve more factors and cost components
  - Deployment models: public, hybrid, private clouds allow different split of responsibilities and costs between IT departments and cloud providers

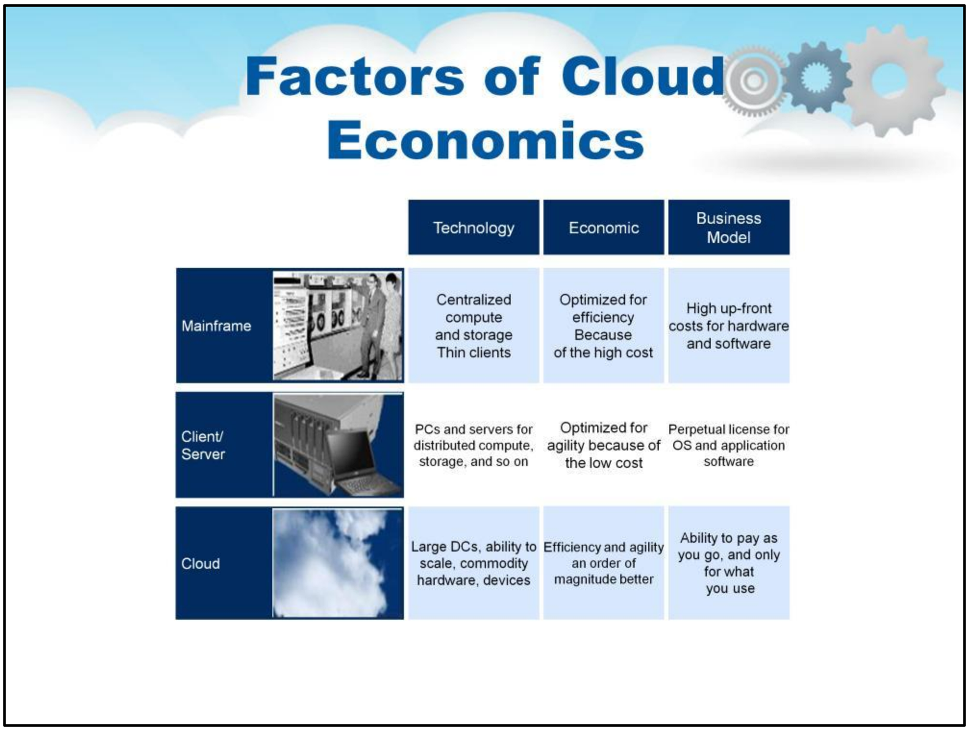


Cloud and Total IT Spending

- The hybrid cloud is more economical than pure public cloud or pure private cloud models
- Hybrid cloud deployment would reduce typical total IT spend by approximately 20 to 30 percent
- Hybrid clouds combine benefits of public clouds elasticity and on-demand resource provisioning with the benefits of private clouds
- Hybrid clouds require standardisation and interoperability between private and public platforms to achieve applications portability

- Hybrid clouds combine benefits of public clouds elasticity and on-demand resource provisioning with the benefits of private clouds
  - Legacy applications
  - Operational control
  - Sensitive applications and data
- Hybrid clouds require standardisation and interoperability between private and public platforms to achieve applications portability
  - Common platform
  - Common managegent
  - Security
  - Compliance

[ref] Business Agility and the True Economics of Cloud Computing. Business White Paper, VMware 2011.



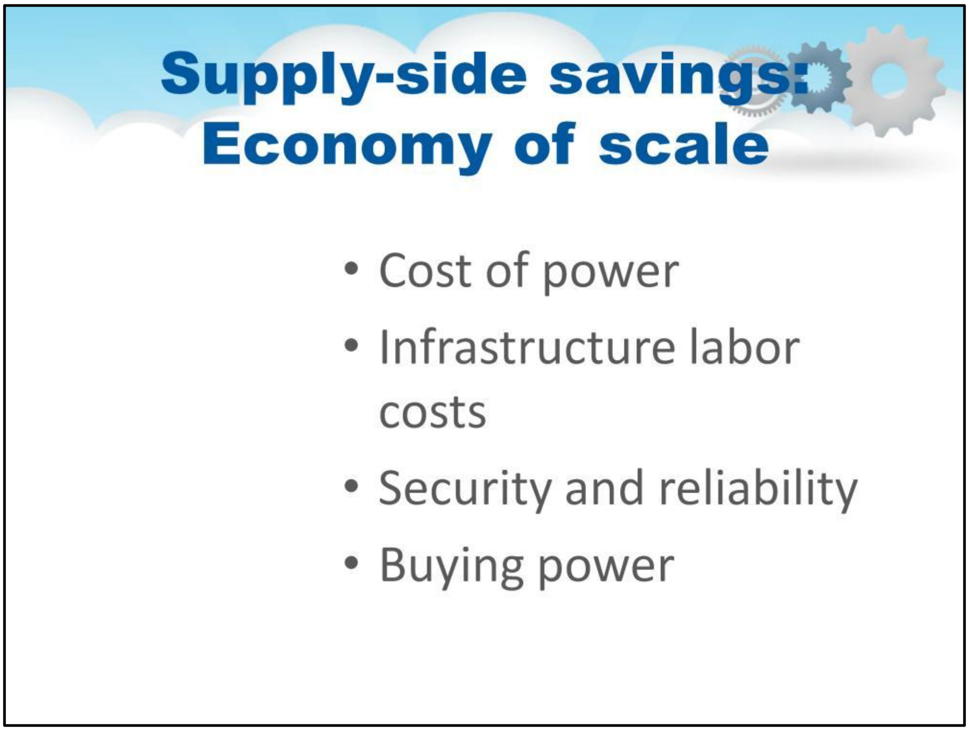
Factors of Cloud Economics

	Technology	Economic	Business Model
Mainframe	Centralized compute and storage Thin clients	Optimized for efficiency Because of the high cost	High up-front costs for hardware and software
Client/ Server	PCs and servers for distributed compute, storage, and so on	Optimized for agility because of the low cost	Perpetual license for OS and application software
Cloud	Large DCs, ability to scale, commodity hardware, devices	Efficiency and agility an order of magnitude better	Ability to pay as you go, and only for what you use

- Supply-side savings: Economy of scale
    - Large-scale data centers (DCs) lower costs per server.
- Demand-side aggregation: Aggregating demand for computing smooths overall variability, allowing server utilization rates to increase.
- Multi-tenancy efficiency: When changing to a multitenant application model, increasing the number of tenants (i.e., customers or users) lowers the application management and server cost per tenant.

The Economics of the Cloud.

Microsoft, 2010



Supply-side savings: Economy of scale

- Cost of power
- Infrastructure labor costs
- Security and reliability
- Buying power

Electricity cost is rapidly rising to become the largest element of total cost of ownership (TCO), currently representing 15%-20%.
Power Usage Effectiveness tends to be significantly lower in large facilities than in smaller ones.

- PUE = Total Facility Energy / It Facility Energy

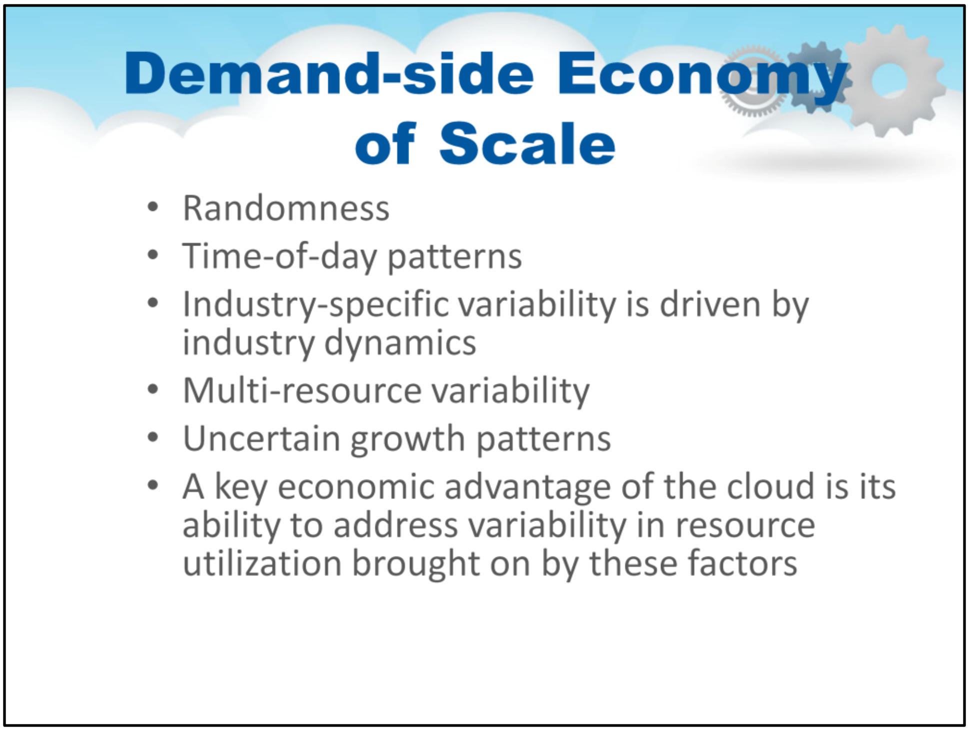
Cloud computing significantly lowers labor costs by automating many repetitive management tasks
A single system administrator can service approx 100+ servers in a traditional enterprise
In a cloud data center the same administrator can service thousands of servers
More time for higher value-add activities like building new capabilities and responding customer requests.

General IT facility security, reliability and compliance are common problems in traditional and cloud IT

- Often cited as a potential hurdle to public cloud adoption

Cloud again brings the economies of scale due to the largely fixed level of investment required to achieve operational security, reliability, and compliance.

- Large commercial cloud providers are able to bring deep expertise to address this problem, thus actually making cloud systems more secure and reliable.
- Modern hardware assisted virtualisation technologies allow for better tasks and tenants separation in a cloud multi-tenant mode

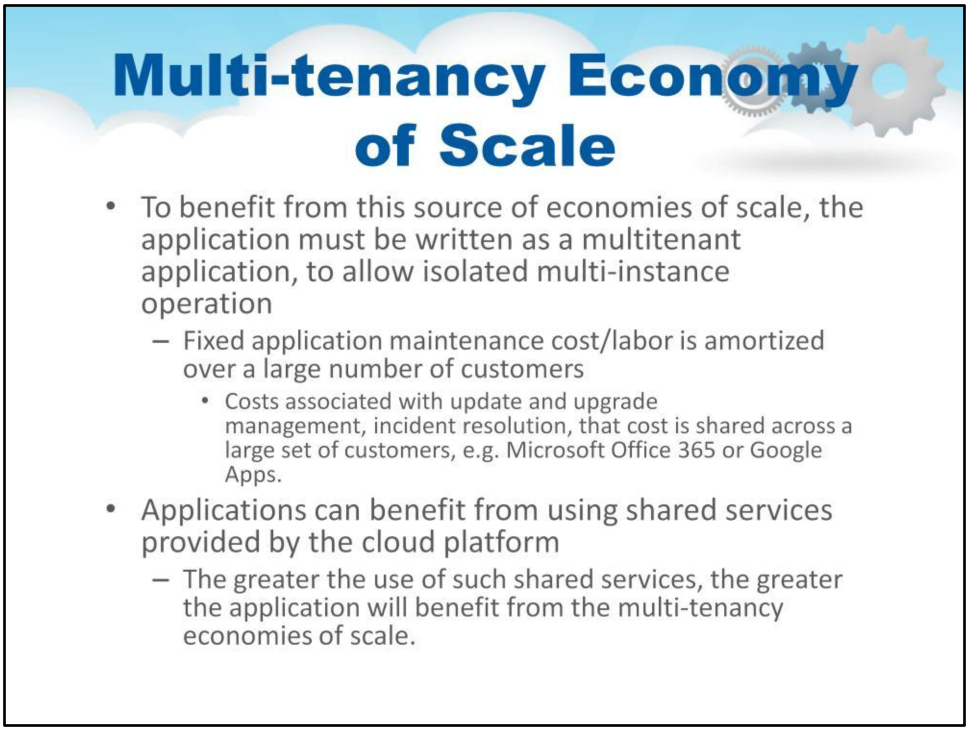


Demand-side Economy of Scale

- Randomness
- Time-of-day patterns
- Industry-specific variability is driven by industry dynamics
- Multi-resource variability
- Uncertain growth patterns
- A key economic advantage of the cloud is its ability to address variability in resource utilization brought on by these factors

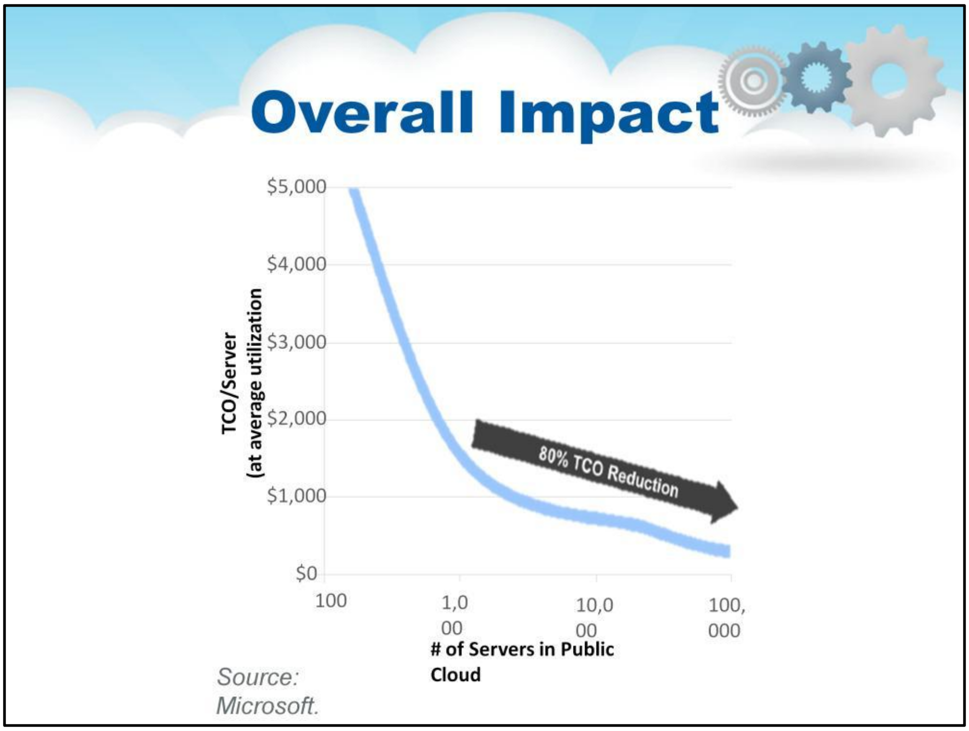
- Randomness:
  - End-user access patterns contain a certain degree of randomness, e.g checking email or accessing shared file storage.
  - To meet service level agreements, capacity buffers are built to account for a certain probability of services demand.
- Time-of-day patterns:
  - Daily recurring cycles in people's behavior: consumer services tend to peak in the evening, while workplace services tend to peak during the workday
  - Capacity has to be built to account for these daily peaks but can be optimized by load balancing between different load patterns
- Industry-specific variability is driven by industry dynamics:
  - Retail firms see a spike during the holiday shopping season while U.S. tax firms will see a peak before April 15.
- Multi-resource variability:
  - Compute, storage, and input/output (I/O) resources are generally bought in bundles: a server contains a certain amount of computing power (CPU), storage, and I/O (e.g., networking or disk access).
  - Some workloads like search use a lot of CPU but relatively little storage or I/O, while other workloads like email tend to use a lot of storage but little CPU.
- Uncertain growth patterns:
  - The difficulty of predicting future need for computing resources and the long lead time for bringing capacity online is another source of low utilization

A key economic advantage of the cloud is its ability to address variability in resource utilization brought on by these factors.



Multi-tenancy Economy of Scale

- To benefit from this source of economies of scale, the application must be written as a multitenant application, to allow isolated multi-instance operation
  - Fixed application maintenance cost/labor is amortized over a large number of customers
    - Costs associated with update and upgrade management, incident resolution, that cost is shared across a large set of customers, e.g. Microsoft Office 365 or Google Apps.
- Applications can benefit from using shared services provided by the cloud platform
  - The greater the use of such shared services, the greater the application will benefit from the multi-tenancy economies of scale.

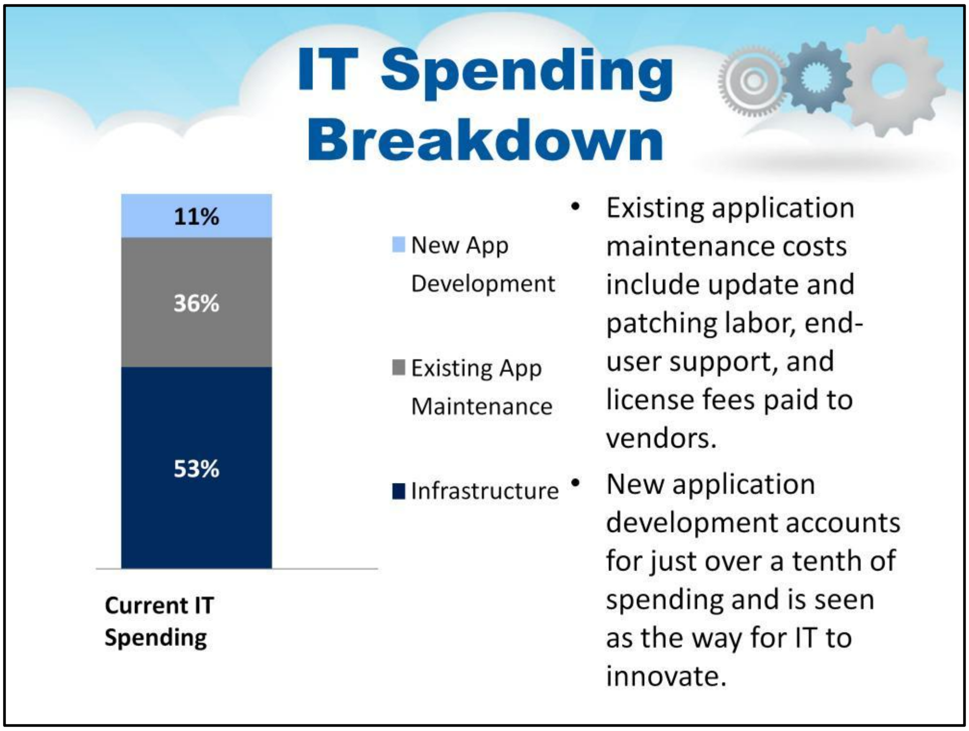


Overall Impact

Source:
Microsoft.

- The combination of supply-side economies of scale in server capacity (amortizing costs across more servers), demand-side aggregation of workloads (reducing variability), and the multi-tenant application model (amortizing costs across multiple customers) leads to powerful economies of scale.

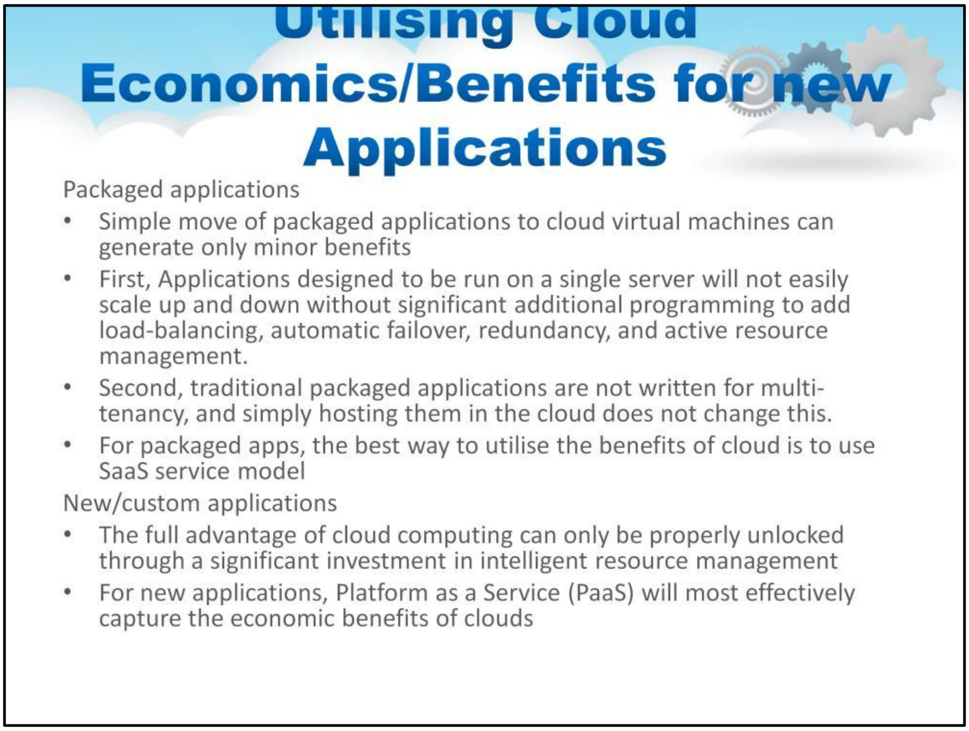
- The model indicates that a 100,000-server datacenter has an 80% lower total cost of ownership (TCO) compared to a 1,000-server datacenter.



IT Spending Breakdown

- Existing application maintenance costs include update and patching labor, end-user support, and license fees paid to vendors.
- New application development accounts for just over a tenth of spending and is seen as the way for IT to innovate.

- Existing application maintenance costs include update and patching labor, end-user support, and license fees paid to vendors.
  - They account for roughly a third of spending and are targeted by the multi-tenancy efficiency factor.
- New application development accounts for just over a tenth of spending and is seen as the way for IT to innovate.
  - The economic benefits of cloud computing will enable more effort and budget for innovation.



Utilising Cloud Economics/Benefits for new Applications

Packaged applications

- Simple move of packaged applications to cloud virtual machines can generate only minor benefits
- First, Applications designed to be run on a single server will not easily scale up and down without significant additional programming to add load-balancing, automatic failover, redundancy, and active resource management.
- Second, traditional packaged applications are not written for multi-tenancy, and simply hosting them in the cloud does not change this.
- For packaged apps, the best way to utilise the benefits of cloud is to use SaaS service model

New/custom applications

- The full advantage of cloud computing can only be properly unlocked through a significant investment in intelligent resource management
- For new applications, Platform as a Service (PaaS) will most effectively capture the economic benefits of clouds

- Simple move of packaged applications to cloud virtual machines can generate only minor benefits
- First, Applications designed to be run on a single server will not easily scale up and down without significant additional programming to add load-balancing, automatic failover, redundancy, and active resource management.
    - This limits the extent to which they are able to aggregate demand and increase server utilization.
    - Applications need to be (re-)designed/re-architected for effective use on clouds
- Second, traditional packaged applications are not written for multi-tenancy, and simply hosting them in the cloud does not change this.
- For packaged apps, the best way to utilise the benefits of cloud is to use SaaS service model
    - For example, Office365, which have been architected for scale-out and multi-tenancy to capture the full benefits of cloud platforms

New/custom applications

- The full advantage of cloud computing can only be properly unlocked through a
- significant investment in intelligent resource management
    - The resource manager must understand both the status of the resources (networking, storage, and compute) as well as the activity of the applications

For new applications, Platform as a Service (PaaS) will most effectively capture the economic benefits of clouds

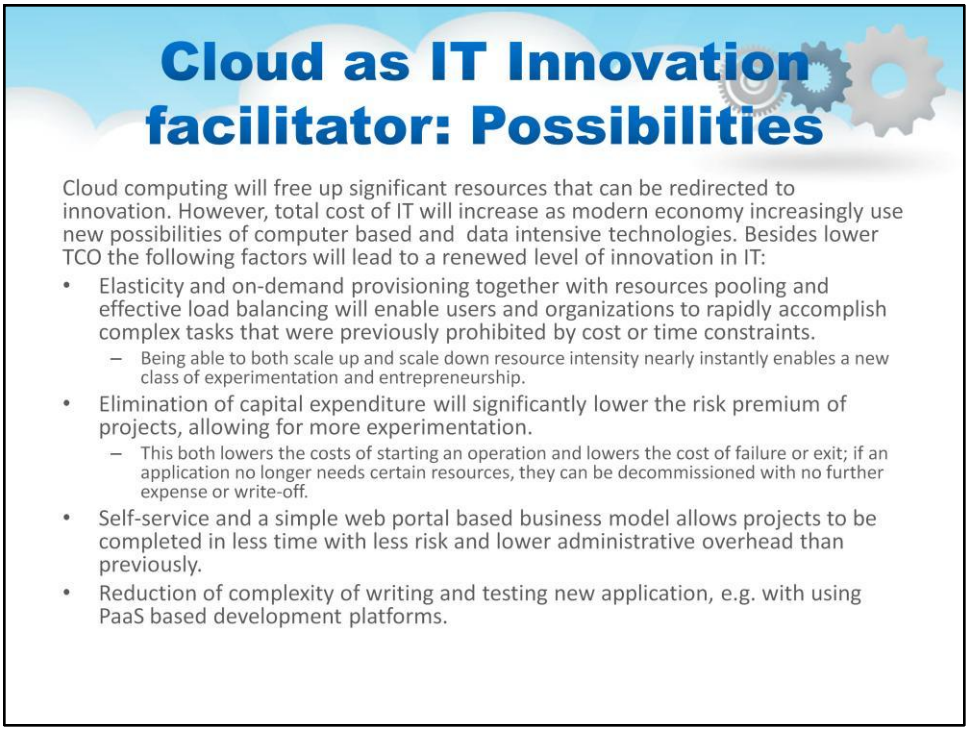
-
    - PaaS offers shared services, advanced management, and automation features that allow developers to focus directly on application logic rather than on engineering their application to scale.

Attributes of Cloud Computing

–

Clouds as IT Innovation Facilitator

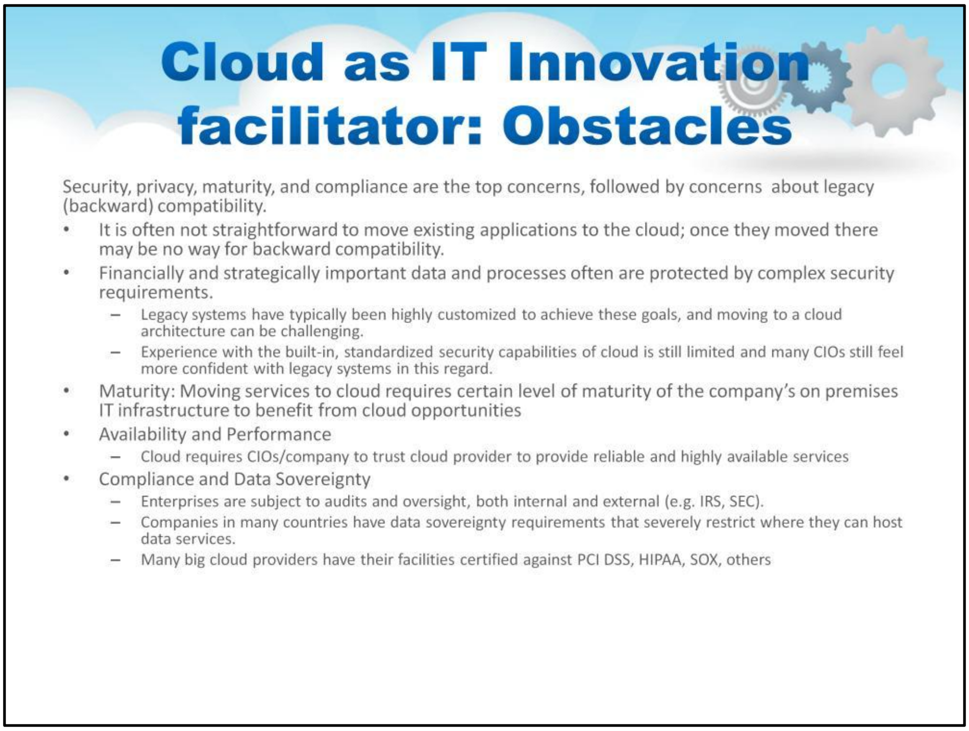
Clouds as IT Innovator Facilitator



Cloud as IT Innovation facilitator: Possibilities

Cloud computing will free up significant resources that can be redirected to innovation. However, total cost of IT will increase as modern economy increasingly use new possibilities of computer based and data intensive technologies. Besides lower TCO the following factors will lead to a renewed level of innovation in IT:

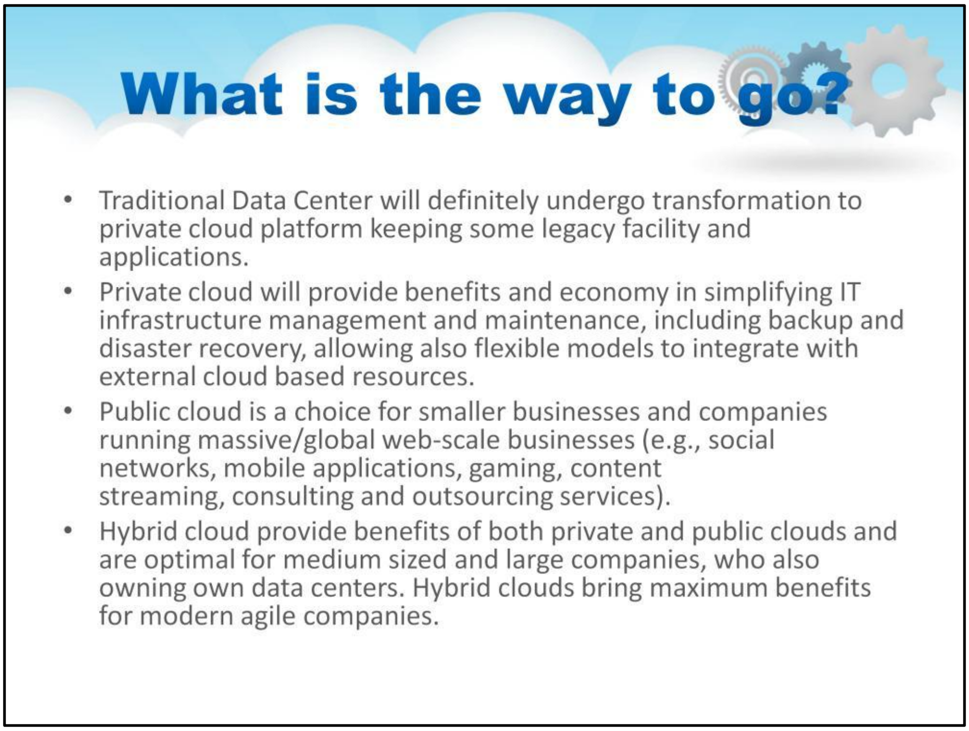
- Elasticity and on-demand provisioning together with resources pooling and effective load balancing will enable users and organizations to rapidly accomplish complex tasks that were previously prohibited by cost or time constraints.
  - Being able to both scale up and scale down resource intensity nearly instantly enables a new class of experimentation and entrepreneurship.
- Elimination of capital expenditure will significantly lower the risk premium of projects, allowing for more experimentation.
  - This both lowers the costs of starting an operation and lowers the cost of failure or exit; if an application no longer needs certain resources, they can be decommissioned with no further expense or write-off.
- Self-service and a simple web portal based business model allows projects to be completed in less time with less risk and lower administrative overhead than previously.
- Reduction of complexity of writing and testing new application, e.g. with using PaaS based development platforms.



Cloud as IT Innovation facilitator: Obstacles

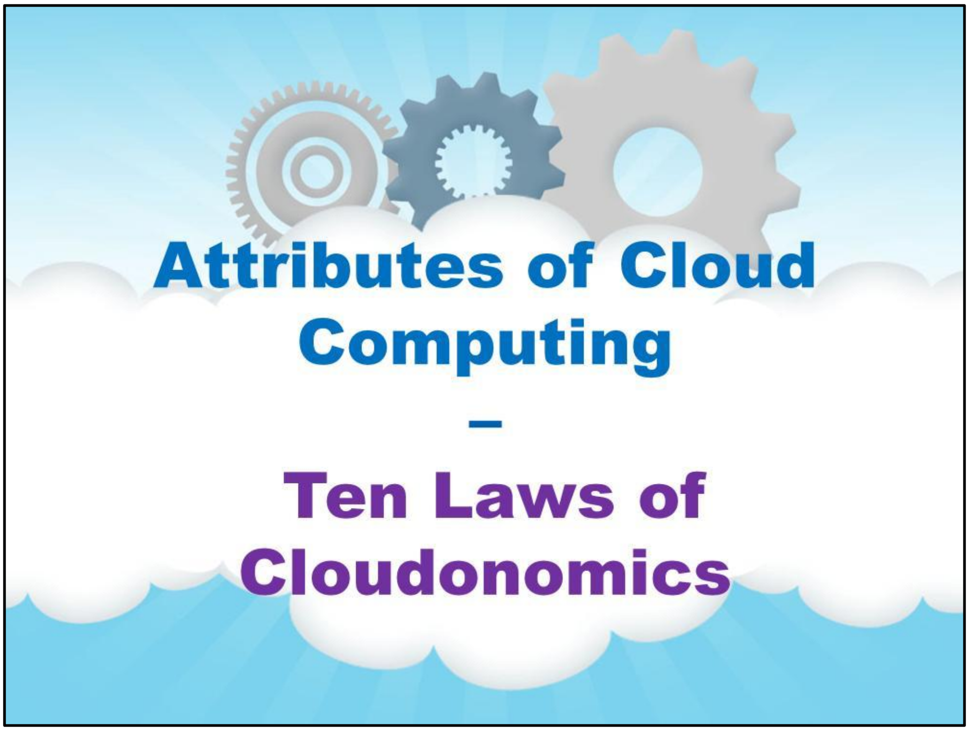
Security, privacy, maturity, and compliance are the top concerns, followed by concerns about legacy (backward) compatibility.

- It is often not straightforward to move existing applications to the cloud; once they moved there may be no way for backward compatibility.
- Financially and strategically important data and processes often are protected by complex security requirements.
  - Legacy systems have typically been highly customized to achieve these goals, and moving to a cloud architecture can be challenging.
  - Experience with the built-in, standardized security capabilities of cloud is still limited and many CIOs still feel more confident with legacy systems in this regard.
- Maturity: Moving services to cloud requires certain level of maturity of the company's on premises IT infrastructure to benefit from cloud opportunities
- Availability and Performance
  - Cloud requires CIOs/company to trust cloud provider to provide reliable and highly available services
- Compliance and Data Sovereignty
  - Enterprises are subject to audits and oversight, both internal and external (e.g. IRS, SEC).
  - Companies in many countries have data sovereignty requirements that severely restrict where they can host data services.
  - Many big cloud providers have their facilities certified against PCI DSS, HIPAA, SOX, others



What is the way to go?

- Traditional Data Center will definitely undergo transformation to private cloud platform keeping some legacy facility and applications.
- Private cloud will provide benefits and economy in simplifying IT infrastructure management and maintenance, including backup and disaster recovery, allowing also flexible models to integrate with external cloud based resources.
- Public cloud is a choice for smaller businesses and companies running massive/global web-scale businesses (e.g., social networks, mobile applications, gaming, content streaming, consulting and outsourcing services).
- Hybrid cloud provide benefits of both private and public clouds and are optimal for medium sized and large companies, who also owning own data centers. Hybrid clouds bring maximum benefits for modern agile companies.

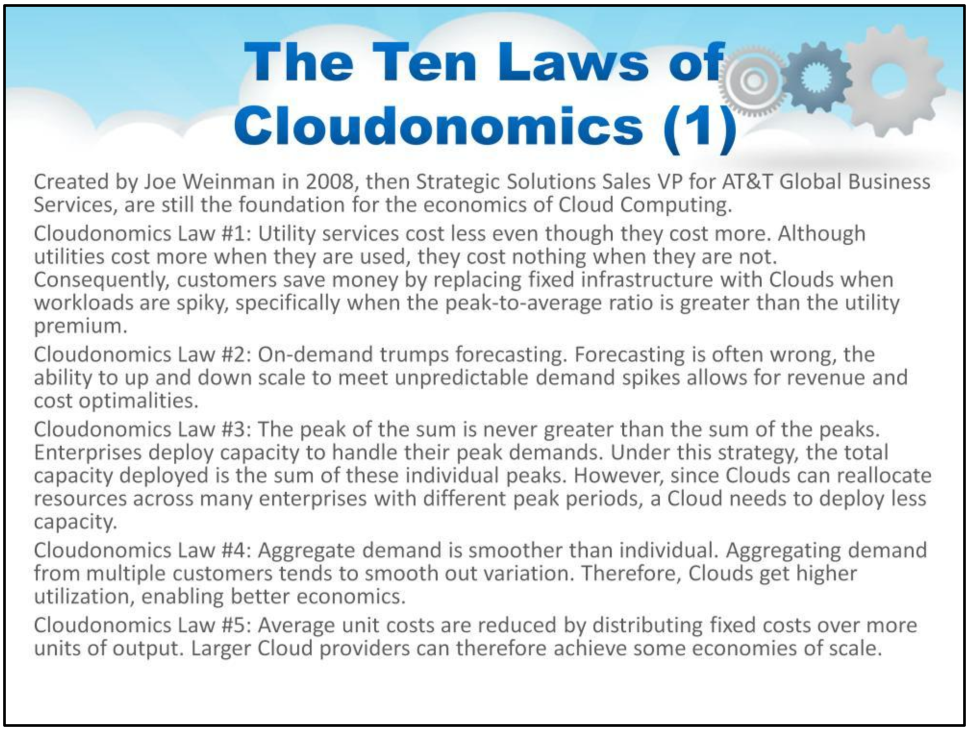


Attributes of Cloud Computing

–

Ten Laws of Cloudonomics

Ten Laws of Cloudonomics



The Ten Laws of Cloudonomics (1)

Created by Joe Weinman in 2008, then Strategic Solutions Sales VP for AT&T Global Business Services, are still the foundation for the economics of Cloud Computing.

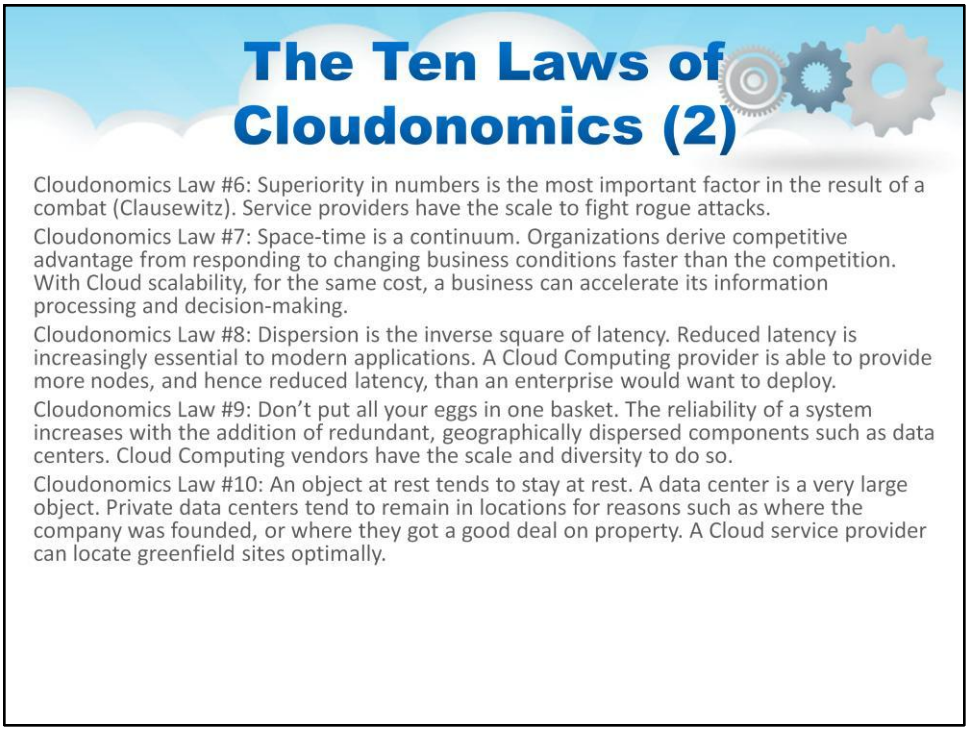
Cloudonomics Law #1: Utility services cost less even though they cost more. Although utilities cost more when they are used, they cost nothing when they are not. Consequently, customers save money by replacing fixed infrastructure with Clouds when workloads are spiky, specifically when the peak-to-average ratio is greater than the utility premium.

Cloudonomics Law #2: On-demand trumps forecasting. Forecasting is often wrong, the ability to up and down scale to meet unpredictable demand spikes allows for revenue and cost optimalities.

Cloudonomics Law #3: The peak of the sum is never greater than the sum of the peaks. Enterprises deploy capacity to handle their peak demands. Under this strategy, the total capacity deployed is the sum of these individual peaks. However, since Clouds can reallocate resources across many enterprises with different peak periods, a Cloud needs to deploy less capacity.

Cloudonomics Law #4: Aggregate demand is smoother than individual. Aggregating demand from multiple customers tends to smooth out variation. Therefore, Clouds get higher utilization, enabling better economics.

Cloudonomics Law #5: Average unit costs are reduced by distributing fixed costs over more units of output. Larger Cloud providers can therefore achieve some economies of scale.



The Ten Laws of Cloudonomics (2)

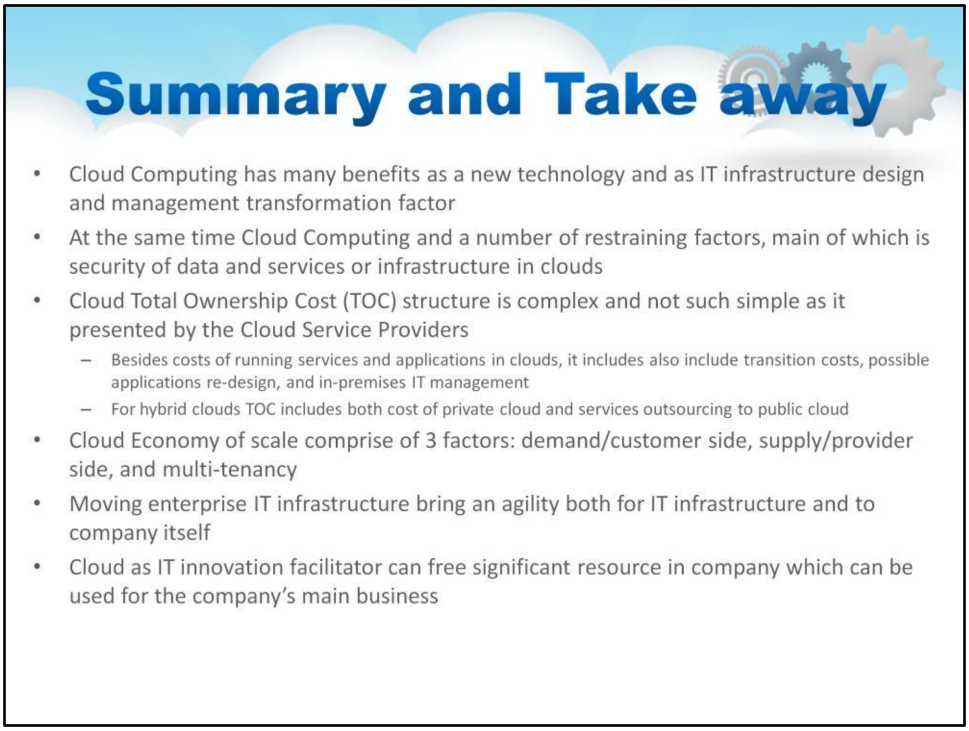
Cloudonomics Law #6: Superiority in numbers is the most important factor in the result of a combat (Clausewitz). Service providers have the scale to fight rogue attacks.

Cloudonomics Law #7: Space-time is a continuum. Organizations derive competitive advantage from responding to changing business conditions faster than the competition. With Cloud scalability, for the same cost, a business can accelerate its information processing and decision-making.

Cloudonomics Law #8: Dispersion is the inverse square of latency. Reduced latency is increasingly essential to modern applications. A Cloud Computing provider is able to provide more nodes, and hence reduced latency, than an enterprise would want to deploy.

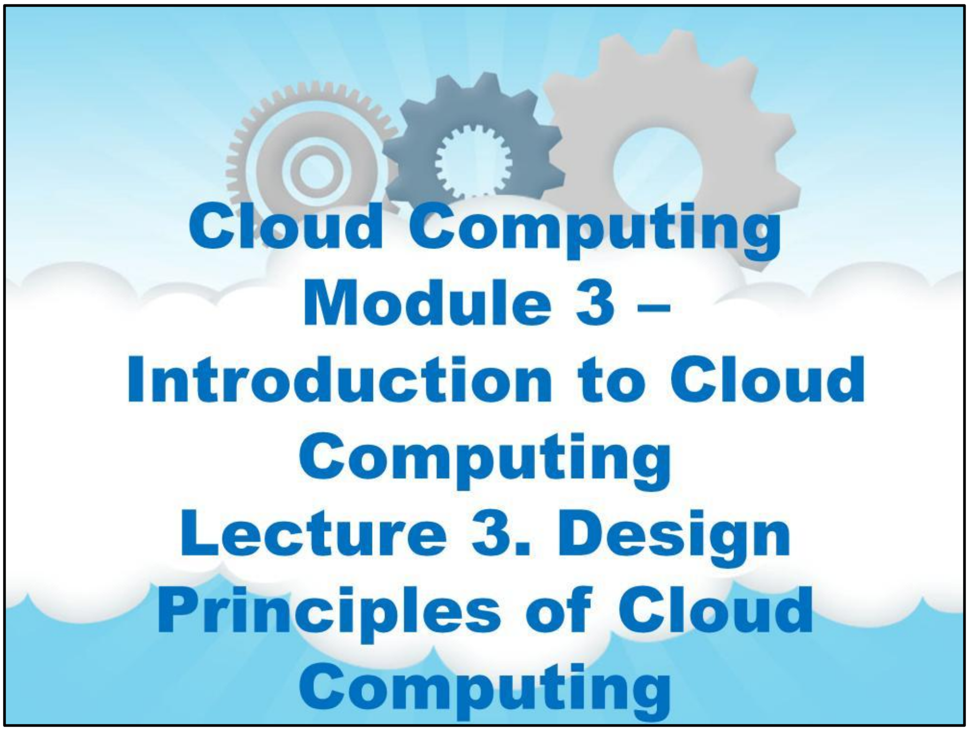
Cloudonomics Law #9: Don’t put all your eggs in one basket. The reliability of a system increases with the addition of redundant, geographically dispersed components such as data centers. Cloud Computing vendors have the scale and diversity to do so.

Cloudonomics Law #10: An object at rest tends to stay at rest. A data center is a very large object. Private data centers tend to remain in locations for reasons such as where the company was founded, or where they got a good deal on property. A Cloud service provider can locate greenfield sites optimally.

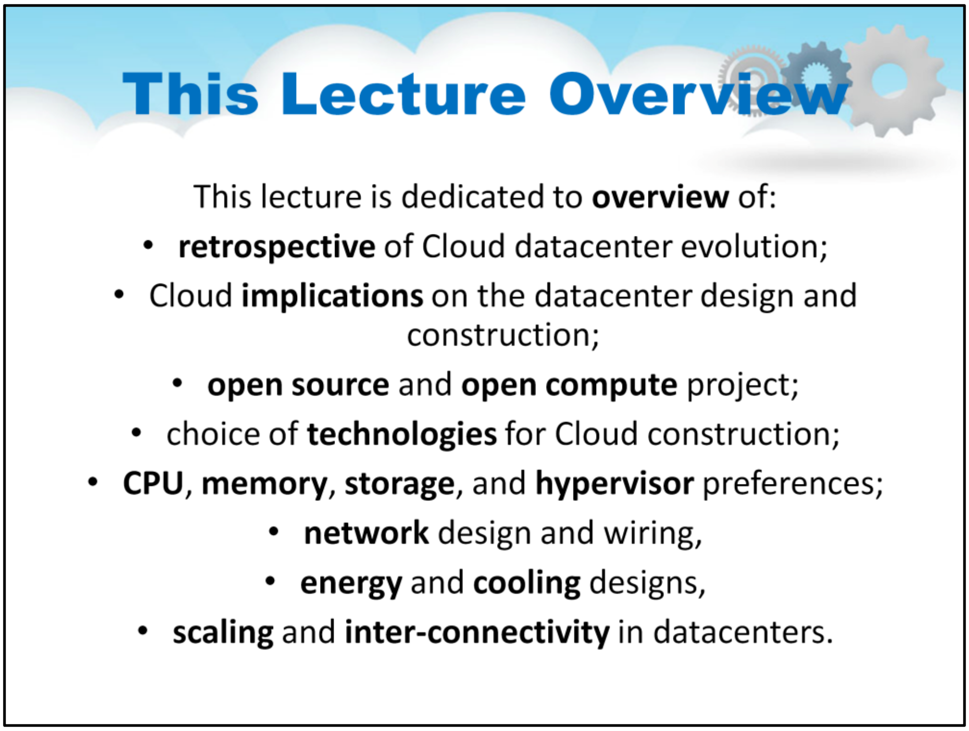


Summary and Take away

- Cloud Computing has many benefits as a new technology and as IT infrastructure design and management transformation factor
- At the same time Cloud Computing and a number of restraining factors, main of which is security of data and services or infrastructure in clouds
- Cloud Total Ownership Cost (TOC) structure is complex and not such simple as it presented by the Cloud Service Providers
  - Besides costs of running services and applications in clouds, it includes also include transition costs, possible applications re-design, and in-premises IT management
  - For hybrid clouds TOC includes both cost of private cloud and services outsourcing to public cloud
- Cloud Economy of scale comprise of 3 factors: demand/customer side, supply/provider side, and multi-tenancy
- Moving enterprise IT infrastructure bring an agility both for IT infrastructure and to company itself
- Cloud as IT innovation facilitator can free significant resource in company which can be used for the company's main business



Cloud Computing Module 3 – Introduction to Cloud Computing Lecture 3. Design Principles of Cloud Computing



This Lecture Overview

This lecture is dedicated to **overview** of:

- **retrospective** of Cloud datacenter evolution;
- Cloud **implications** on the datacenter design and construction;
- **open source** and **open compute** project;
- choice of **technologies** for Cloud construction;
- **CPU**, **memory**, **storage**, and **hypervisor** preferences;
- **network** design and wiring,
- **energy** and **cooling** designs,
- **scaling** and **inter-connectivity** in datacenters.

Lecture 3. Design Principles of Cloud Computing

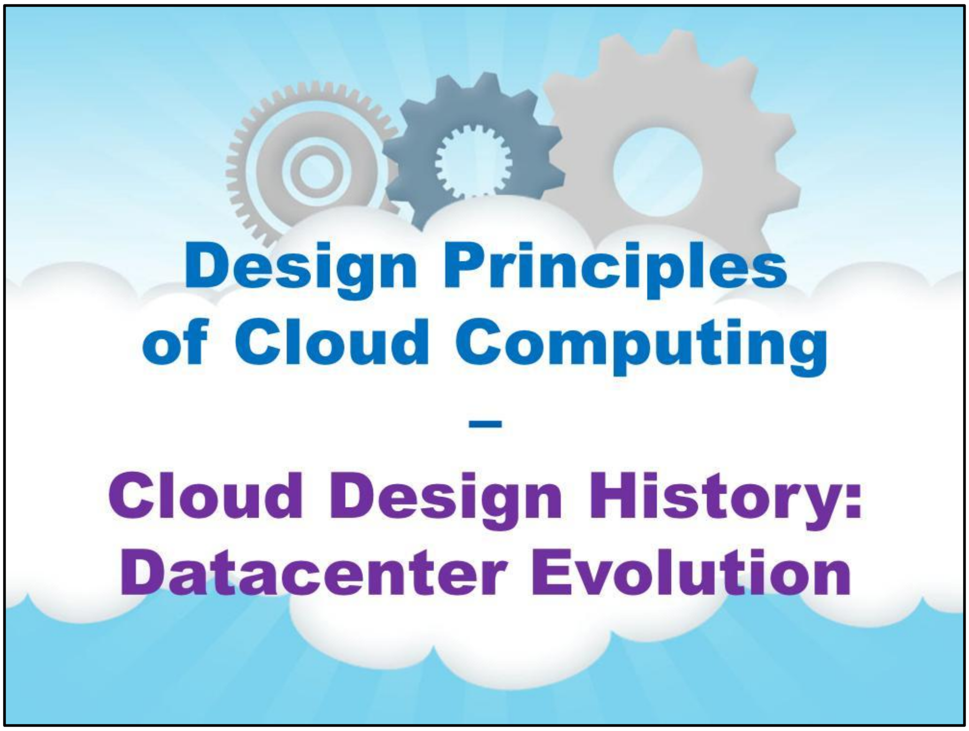
Outline

- Cloud Datacenter Evolution
- Cloud Design Principles
- Cloud Implication on the Datacenter Design and Construction
  - Open Source and Open Compute Project
- Technologies Choices for Cloud Construction
  - CPU and Hypervisor Preferences
  - Memory and Storage
  - Network Design and Wiring
- Energy and Cooling Designs
- Scale Out and Inter datacenter connectivity
- Wrap up and Summary

Outline

In this lecture we cover

Cloud Datacenter Evolution
Cloud Implication on the Datacenter Design and Construction
    Open Source and Open Compute Project
Technologies Choices for Cloud Construction
    CPU and Hypervisor Preferences
    Memory and Storage
    Network Design and Wiring
Energy and Cooling Designs
Scale Out and Inter datacenter connectivity
Wrap up and Summary

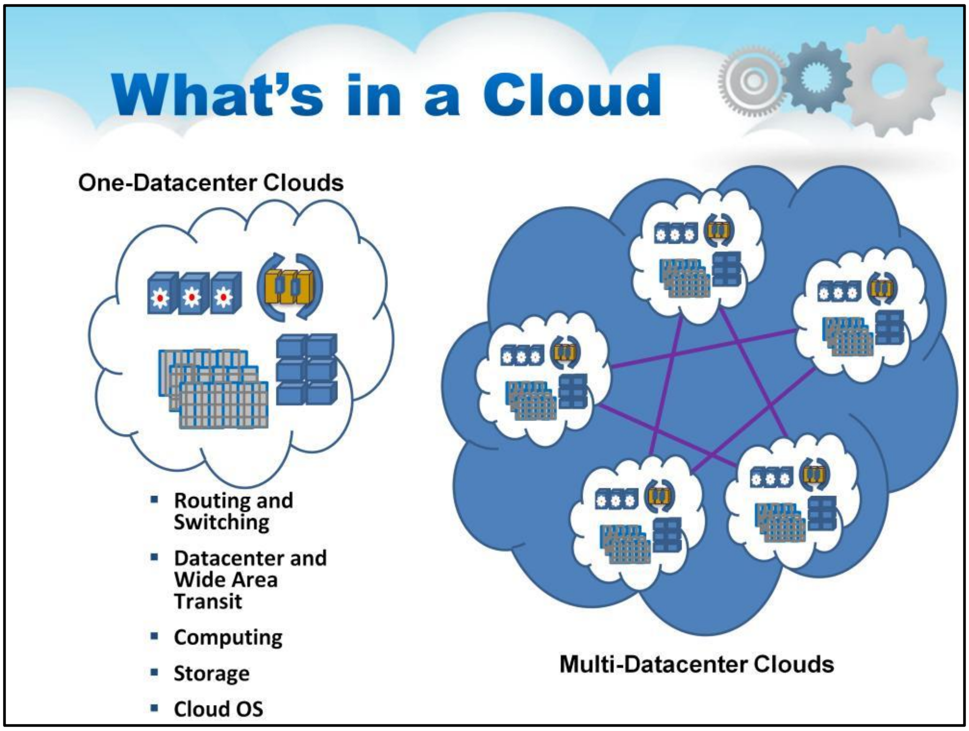


Design Principles of Cloud Computing

–

Cloud Design History: Datacenter Evolution

Cloud Design History



What’s in a Cloud

One-Datacenter Clouds

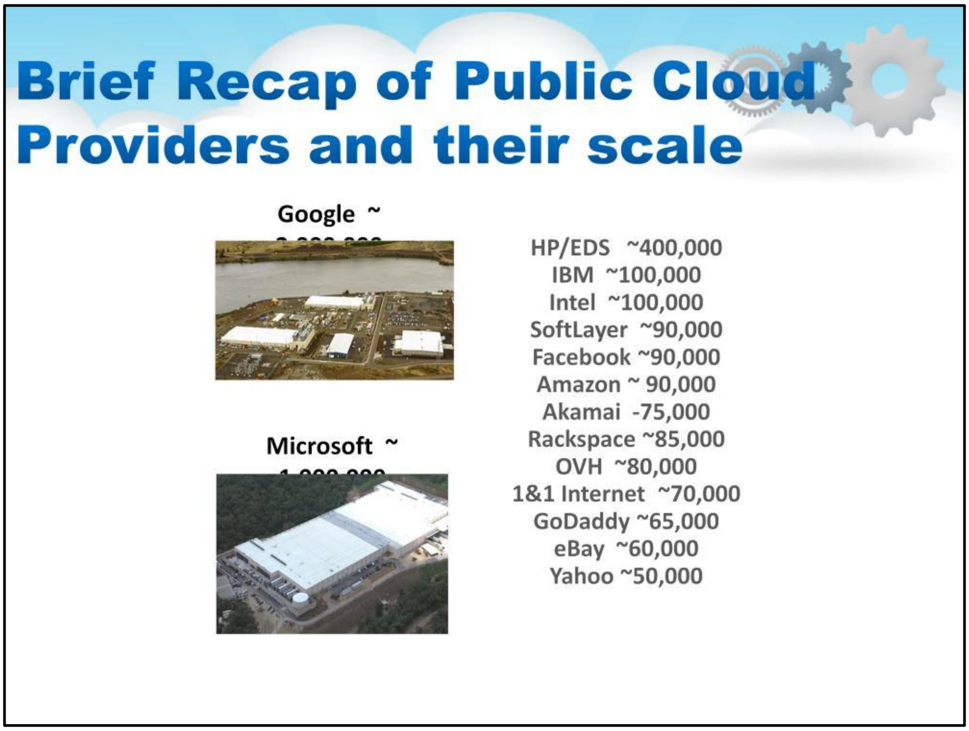
- Routing and Switching
- Datacenter and Wide Area Transit
- Computing
- Storage
- Cloud OS

Multi-Datacenter Clouds

What’s in a Cloud

Routing and Switching
Datacenter and Wide Area Transit
Computing
Storage
Cloud OS

One-Datacenter Clouds
Multi-Datacenter Clouds



Brief Recap of Public Cloud Providers and their scale

Google ~ 2,000,000

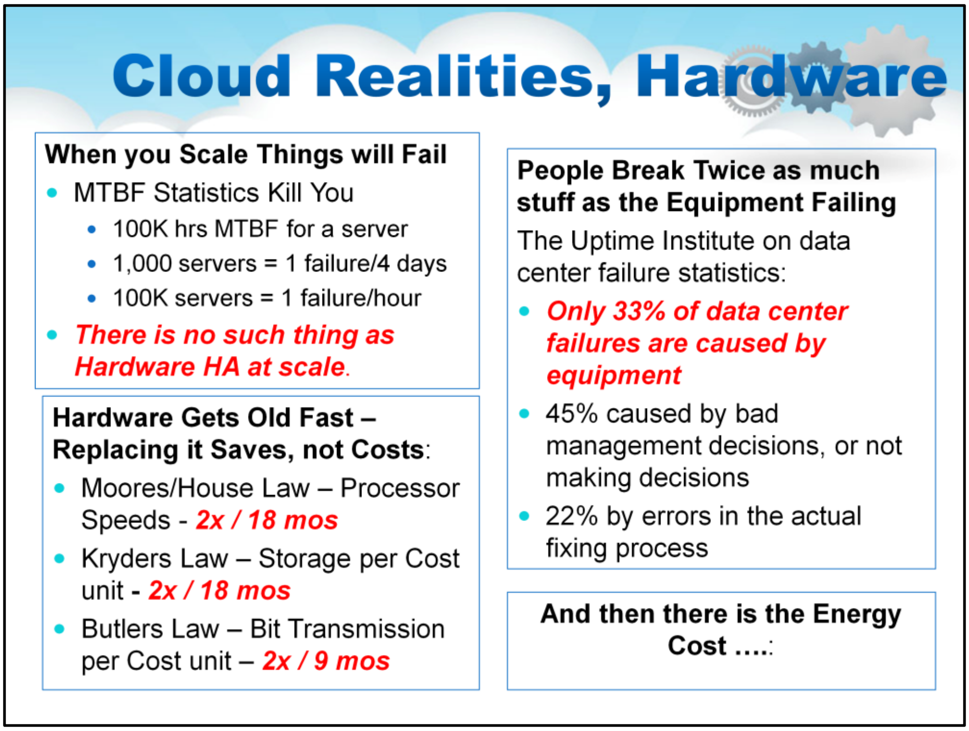
Microsoft ~ 1,000,000

HP/EDS ~400,000
IBM ~100,000
Intel ~100,000
SoftLayer ~90,000
Facebook ~90,000
Amazon ~ 90,000
Akamai -75,000
Rackspace ~85,000
OVH ~80,000
1&1 Internet ~70,000
GoDaddy ~65,000
eBay ~60,000
Yahoo ~50,000

Brief Recap of Public Cloud Providers and their scale

The largest public cloud providers are continuing their massive build-outs. While the physical scale of these datacenters is immediately obvious, the software and process accomplishments to manage this size infrastructure are no less impressive.

As can be seen these vendors have figure out how to run clouds across literally millions of servers. How do they do it?



Cloud Realities, Hardware

When you Scale Things will Fail

- MTBF Statistics Kill You
  - 100K hrs MTBF for a server
  - 1,000 servers = 1 failure/4 days
  - 100K servers = 1 failure/hour
- ***There is no such thing as Hardware HA at scale.***

Hardware Gets Old Fast – Replacing it Saves, not Costs:

- Moores/House Law – Processor Speeds - ***2x / 18 mos***
- Kryders Law – Storage per Cost unit **-** ***2x / 18 mos***
- Butlers Law – Bit Transmission per Cost unit – ***2x / 9 mos***

People Break Twice as much stuff as the Equipment Failing

The Uptime Institute on data center failure statistics:

- ***Only 33% of data center failures are caused by equipment***
- 45% caused by bad management decisions, or not making decisions
- 22% by errors in the actual fixing process

And then there is the Energy Cost ….:

Cloud Realities, Hardware

To understand this level of scaling, we need to consider some of the cold realities of hardware, statistics, and the dynamics of failure.

On the statistics side, the numbers are not on your side, as far as keeping things running, when you have large numbers at work.

For example consider how MTBF Statistics Kill You. If you have a server of 100K hrs MTBF for a server

When you have 1,000 servers = 1 failure/4 days

When you have 100K servers = 1 failure/hour

There is no such thing as Hardware HA at scale.

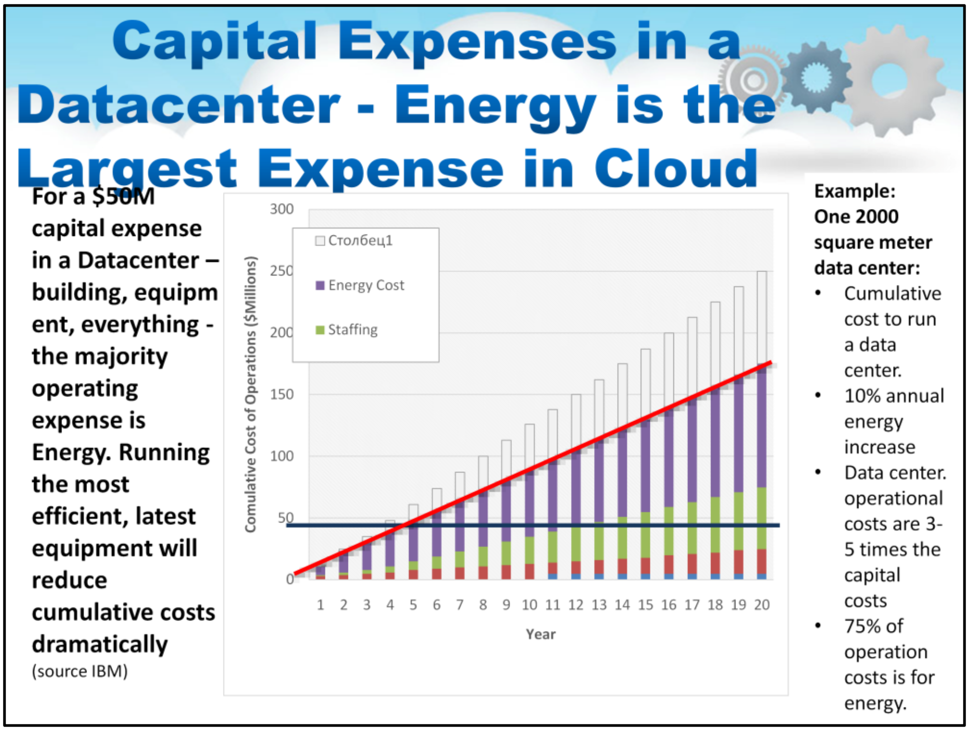
On the hardware side, consider the “efficiency of capital” in temrs of keeping servers and hardware around for even half of a depreciation cycle

An 18 month old server with be ½ as efficient use of capital as a new one (Moores Law)

An 18 mo old reserve of storage will cost ½ as much if purchased today instead of back then (Kryders Law)

An 18 mo old set of networking ports will be ½ the cost if purchased today (Butlers Law)

This means, Hardware Gets Old Fast – Replacing it Saves, not Costs! There is no reason to repair old, broken hardware, this is a very inefficient use of capital



Capital Expenses in a Datacenter - Energy is the Largest Expense in Cloud

For a $50M capital expense in a Datacenter – building, equipment, everything - the majority operating expense is Energy. Running the most efficient, latest equipment will reduce cumulative costs dramatically

(source IBM)

Example: One 2000 square meter data center:

- Cumulative cost to run a data center.
- 10% annual energy increase
- Data center. operational costs are 3-5 times the capital costs
- 75% of operation costs is for energy.

Energy – The Largest Expense in Cloud

This chart has in illustration showing that cumulatively, the majority operating expense is Energy

For a $50M capital expense in a Datacenter – building, equipment, everything - the majority operating expense is Energy. Running the most efficient, latest equipment will reduce cumulative costs dramatically

As one can see, cumulatively, cloud data center operational costs are 3-5 times the capital costs.

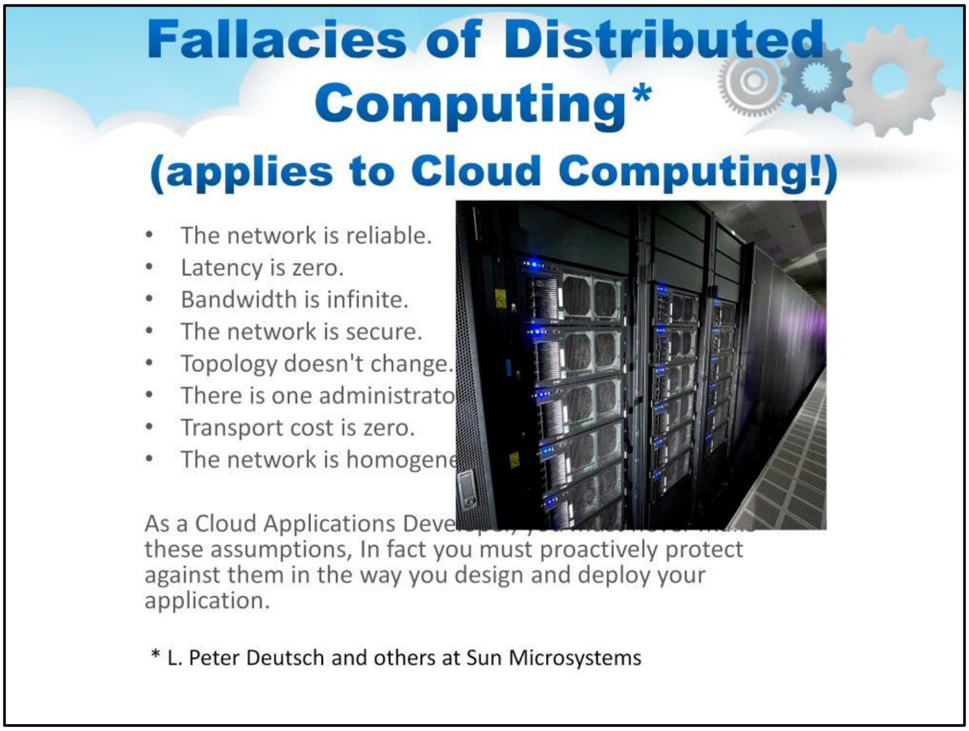
What this tells us is that the most efficient was to say money with a cloud datacenter is to use the most energy-dense configuration possible.

Design Principles of Cloud Computing

–

Overview

Overview



Fallacies of Distributed Computing*

(applies to Cloud Computing!)

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrato
- Transport cost is zero.
- The network is homogene

As a Cloud Applications Deve
these assumptions, In fact you must proactively protect against them in the way you design and deploy your application.

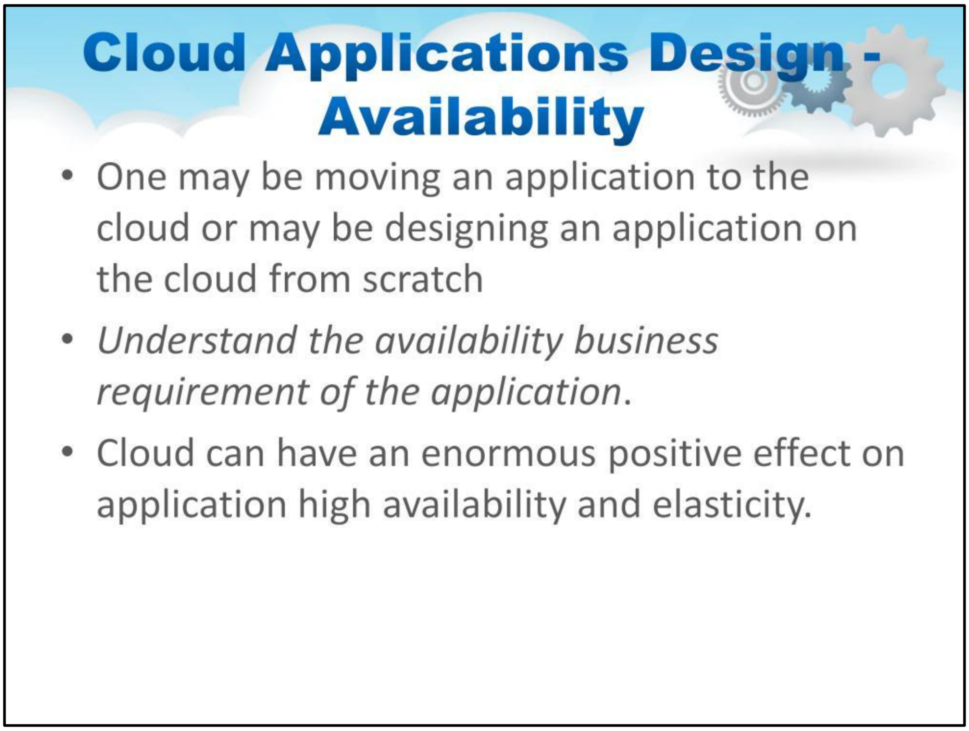
* L. Peter Deutsch and others at Sun Microsystems

The cloud is not a server. It is not a special collection of severs with just the right features for automatically making your application do magic things with scalability or availability,

It is a distributed computing system, and probably quite a large one. So the rules of distributed computing apply, What are these rules? When researcher first began to put applications on distributed computing platforms, the applications broke in several interesting ways. The researchers realized it was because they were making assumptions about the platform that simply were not true. Individually, these assumptions were often true for a development server or two. But in the world of a distributed system platform like the cloud, believing in any of these "fallacies of Distributed Computing" which we will now refer to as the "Fallacies of Cloud, will eventually prove fatal to your application.

The "Fallacies of Cloud" are dscribed here in this slide.

As a Cloud Applications Developer, you must never make these assumptions, In fact you must proactively protect against them in the way you design and deploy your



Cloud Applications Design - Availability

- One may be moving an application to the cloud or may be designing an application on the cloud from scratch
- *Understand the availability business requirement of the application.*
- Cloud can have an enormous positive effect on application high availability and elasticity.

Before jumping in to writing or moving the application, first one should consider some Cloud Applications Design on the subject of Availability. In this category we put both “Resilience to Failure” as well as “Ability to service as scale demand increases”.

Understand the availability business requirement of the application:

Is it absolutely critical that the application have essentially zero possible downtime in any situation?

In other words, are lives at stake, or is the very operation of the business, dependent on this application?

Or, is a 5 minute “restart” or “switchover” timeframe acceptable for the application?

Additionally, does the application experience widely varying load conditions, for example, at the end of each month does load on the application go from 1x to 100x or 1000x?

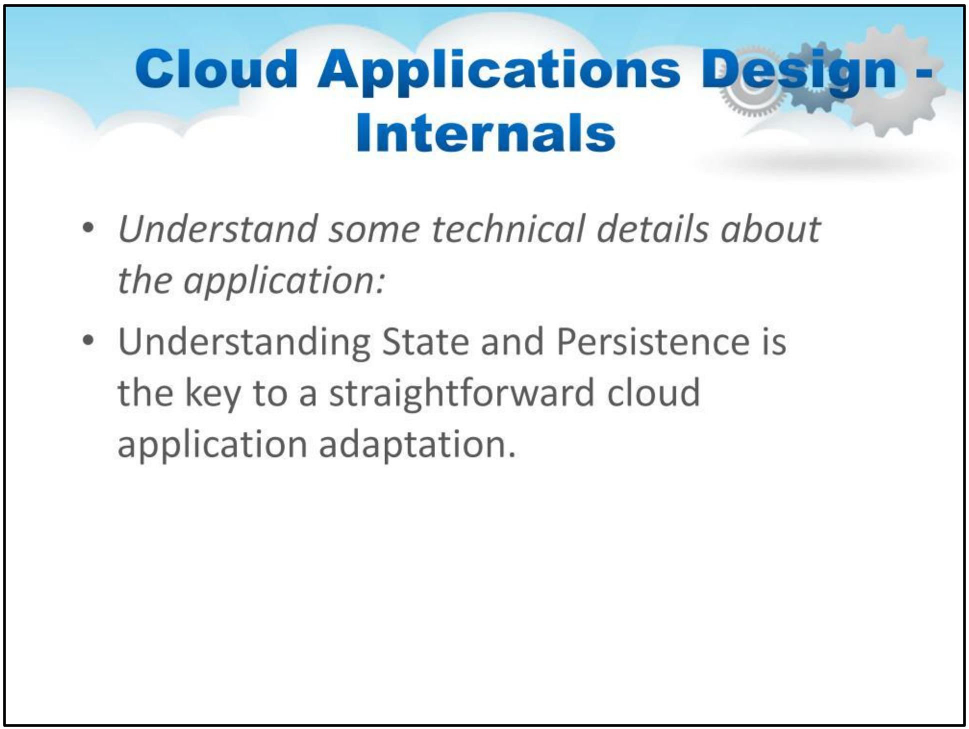
What is the business impact if the application is working, but for certain short periods, is working slowly, is this acceptable?

Cloud can have an enormous positive effect on application high availability and elasticity.

But these capabilities do not come automatically, they must designed-for and enabled/coded-for by the developer/deployer

To develop the right strategy on cloud, one must first understand the availability business requirement of the application:

Is it absolutely critical that the application have essentially zero possible downtime in any application?



Cloud Applications Design - Internals

- *Understand some technical details about the application:*
- Understanding State and Persistence is the key to a straightforward cloud application adaptation.

Next, one must consider the option of how to deploy the application on the cloud. It is not a matter of simply using the cloud as one large server – or a collection of servers – just like in the physical world.

First of all you will get hit with the result of one or more of the “fallacies of cloud computing”. Secondly you will get few if any of the benefits of cloud computing. So it is worth taking the time to Understand some technical details about the application The slide details several key questions about Cloud Applications design. The architecture of the application should be understood.

Understand some technical details about the application:

Are you writing this application brand new on latest infrastructure?

Are you re-platforming a relatively modern application running on latest version operating systems and middleware?

Do you have the source code?

Is that version supported by the application vendor?

How do the various parts of the application talk to each other?

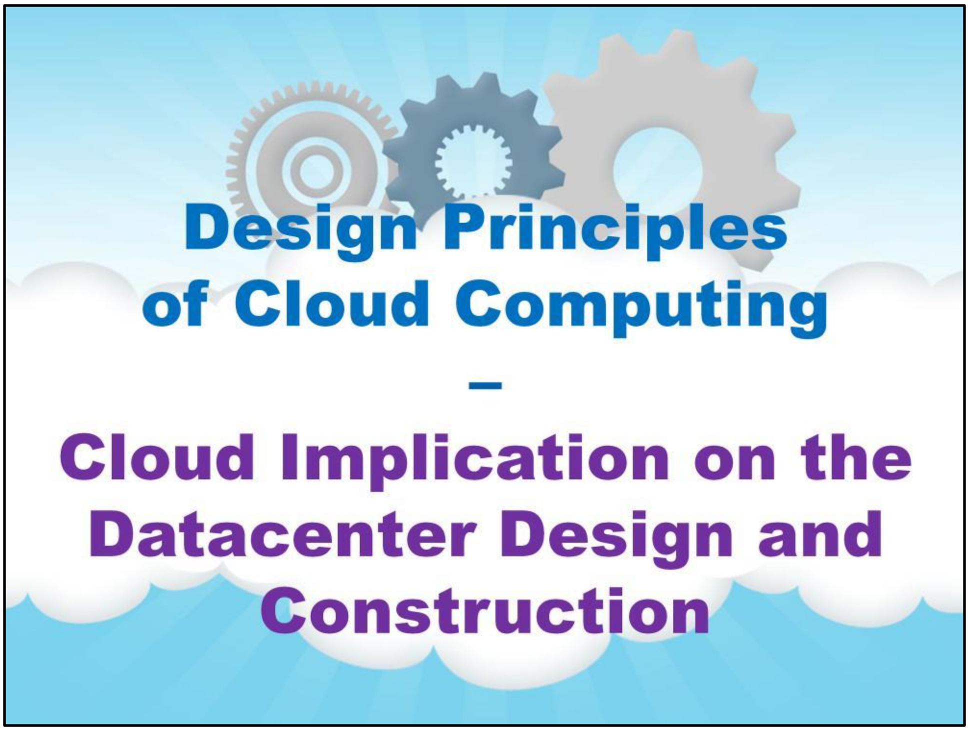
Is it an older application perhaps which had legacy platform requirements, and doesn’t (and won’t) support the latest infrastructure?

Does it use it’s own embedded database or is that externalized to a standard database component?

Is the application managing rapidly changing data which needs to constantly updating storage, or is it a tool which is accessing largely static (or occasionally updated) information?

How does it write to storage?

Understanding State and Persistence is the key to a straightforward cloud application.

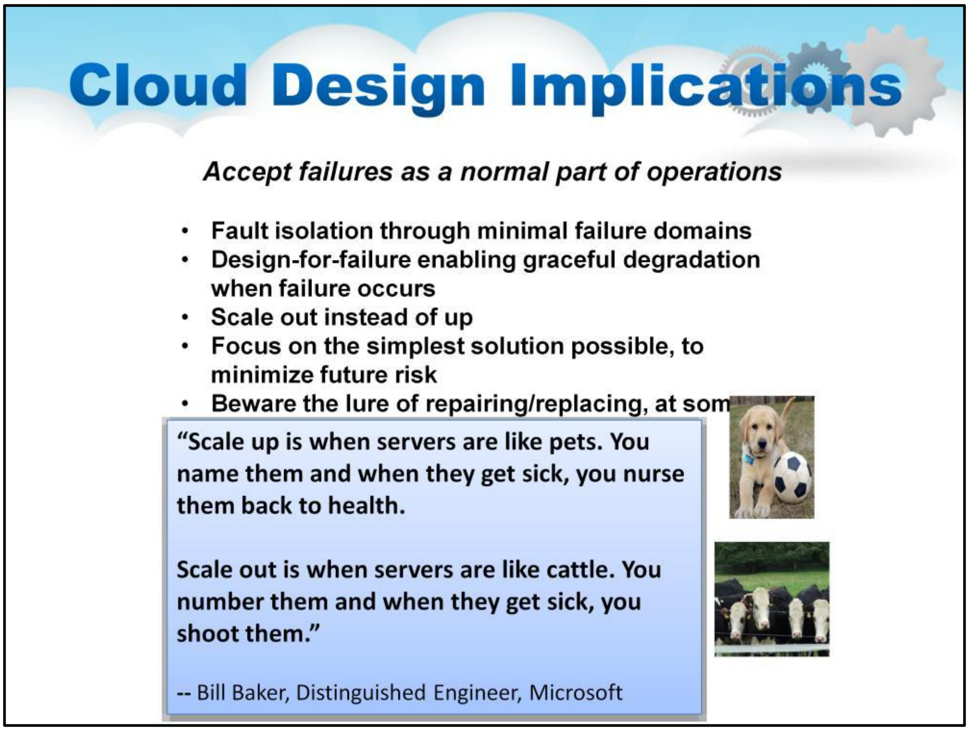


Design Principles of Cloud Computing

–

Cloud Implication on the Datacenter Design and Construction

Cloud Implication on the Datacenter Design and Construction



Cloud Design Implications

Accept failures as a normal part of operations

- Fault isolation through minimal failure domains
- Design-for-failure enabling graceful degradation when failure occurs
- Scale out instead of up
- Focus on the simplest solution possible, to minimize future risk
- Beware the lure of repairing/replacing, at som

"Scale up is when servers are like pets. You name them and when they get sick, you nurse them back to health.

Scale out is when servers are like cattle. You number them and when they get sick, you shoot them."

-- Bill Baker, Distinguished Engineer, Microsoft

Cloud Design Implications

What these statistics at scale show us is, that in Cloud Design, one must Accept failures as a normal part of operations

Heres a philosophy to illustrate the point

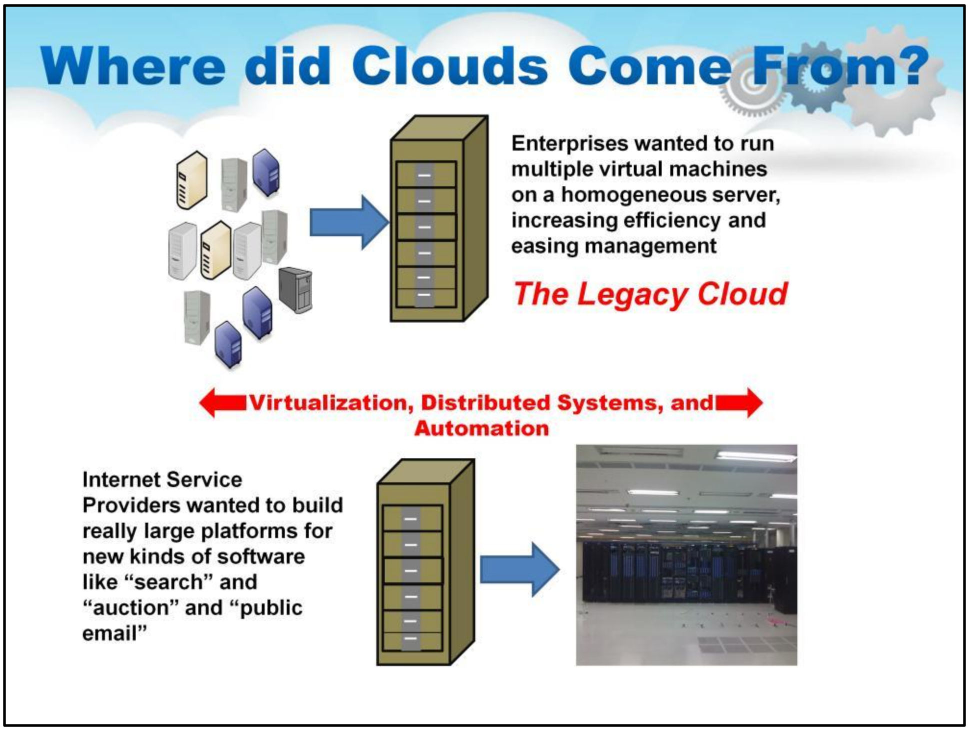
This rather famous quote which exists in many variations, can first be attributed to Bill Baker, Distinguished Engineer, Microsoft

"Scale up is when servers are like pets. You name them and when they get sick, you nurse them back to health.

Scale out is when servers are like cattle. You number them and when they get sick, you shoot them."

The implications are:

Fault isolation through minimal failure domains
Design-for-failure enabling graceful degradation when failure occurs
Scale out instead of up
Focus on the simplest solution possible, to minimize future risk
Beware the lure of repairing/replacing, at some scale, just shut it off
Upgrade as soon as CFO will let you



Where did Clouds Come From?

Enterprises wanted to run multiple virtual machines on a homogeneous server, increasing efficiency and easing management

The Legacy Cloud

Virtualization, Distributed Systems, and Automation

Internet Service Providers wanted to build really large platforms for new kinds of software like “search” and “auction” and “public email”

Where did Clouds Come From?

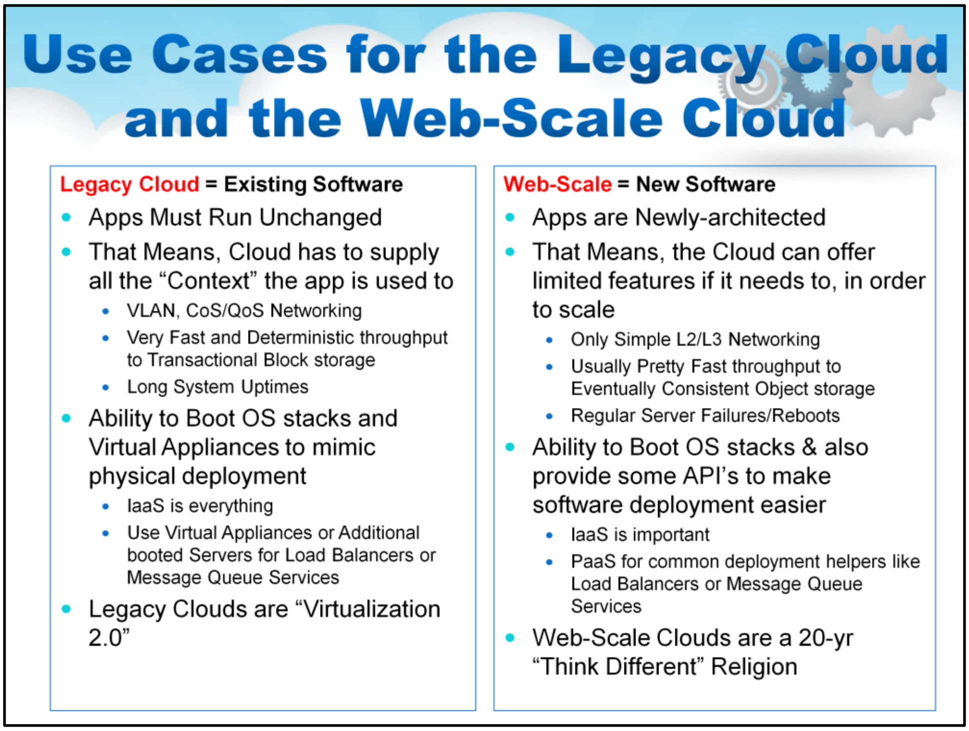
To understand how the perspectives on cloud building have emerged, one must consider where clouds came from.

One the one hand, Enterprises wanted to run multiple virtual machines on a homogeneous server, increasing efficiency and easing management

We call this The Legacy Cloud. These clouds do not contain the design principles to reach a large scale and to use capital efficiently

On the other hand, Internet Service Providers wanted to build really large platforms for new kinds of software like “search” and “auction” and “public email” (sound like Internet Apps you’ve used before?)

We call this The Web-Scale Cloud. These clouds DO contain the design principles to scale. We will look at these design principles in detail.



Use Cases for the Legacy Cloud and the Web-Scale Cloud

Legacy Cloud = Existing Software

- Apps Must Run Unchanged
- That Means, Cloud has to supply all the “Context” the app is used to
  - VLAN, CoS/QoS Networking
  - Very Fast and Deterministic throughput to Transactional Block storage
  - Long System Uptimes
- Ability to Boot OS stacks and Virtual Appliances to mimic physical deployment
  - IaaS is everything
  - Use Virtual Appliances or Additional booted Servers for Load Balancers or Message Queue Services
- Legacy Clouds are “Virtualization 2.0”

Web-Scale = New Software

- Apps are Newly-architected
- That Means, the Cloud can offer limited features if it needs to, in order to scale
  - Only Simple L2/L3 Networking
  - Usually Pretty Fast throughput to Eventually Consistent Object storage
  - Regular Server Failures/Reboots
- Ability to Boot OS stacks & also provide some API’s to make software deployment easier
  - IaaS is important
  - PaaS for common deployment helpers like Load Balancers or Message Queue Services
- Web-Scale Clouds are a 20-yr “Think Different” Religion

Use Cases for the Legacy Cloud and the Web-Scale Cloud

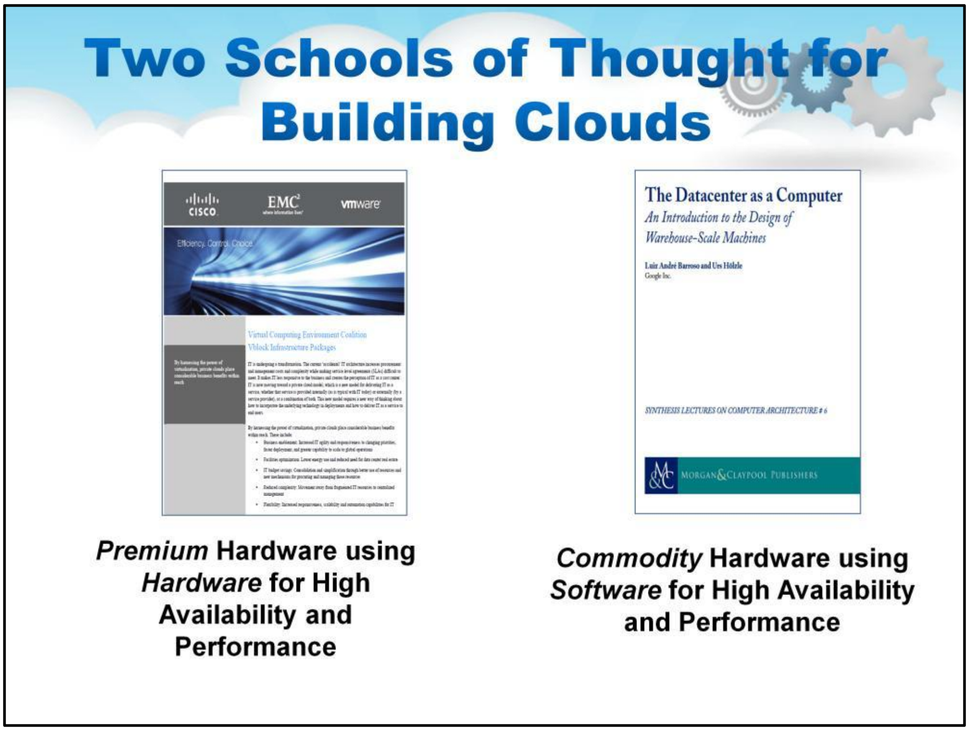
Legacy Cloud = Existing Software
Web-Scale = New Software

In Legacy cloud, Ability to Boot OS stacks and Virtual Appliances to mimic physical deployment
IaaS is everything
Use Virtual Appliances or Additional booted Servers for Load Balancers or Message Queue Services

In Web Scale cloud, Ability to Boot OS stacks & also provide some API’s to make software deployment easier
IaaS is important
PaaS for common deployment helpers like Load Balancers or Message Queue Services

Read the other areas of comparison

Conclusion
Legacy Clouds are “Virtualization 2.0”
Web-Scale Clouds are a 20-yr “Think Different” Religion



CISCO
EMC²
vmware
Efficiency. Control. Choice.
Virtual Computing Environment Coalition
Vblock Infrastructure Packages
The Datacenter as a Computer
An Introduction to the Design of Warehouse-Scale Machines
Luiz André Barroso and Urs Hölzle
Google Inc.
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE # 6
MORGAN&CLAYPOOL PUBLISHERS

Two Schools of Thought for Building Clouds

Premium **Hardware using *Hardware* for High Availability and Performance**

Commodity **Hardware using *Software* for High Availability and Performance**

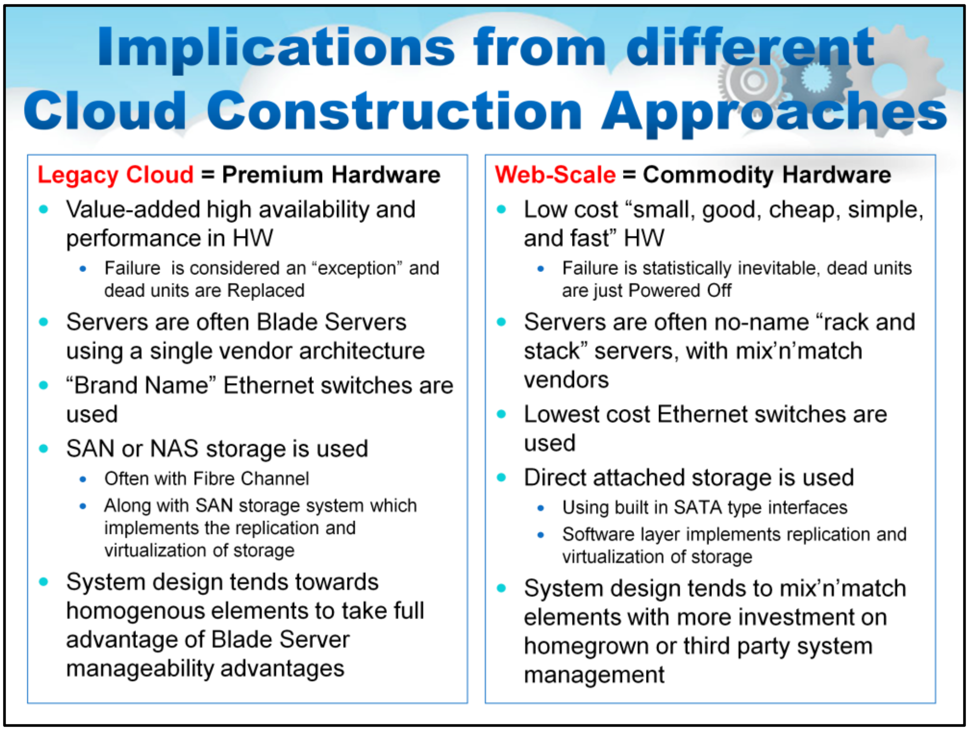
Two Schools of Thought for Building Clouds

There are Two Schools of Thought for Building Clouds, one for the Legacy Cloud, and one for the Web Scale cloud

For the Legacy Cloud, the classic blueprint is using Hardware for High Availability and Performance

For the Web Scale cloud, it’s using Software for High Availability and Performance

Literature for these two blueprints are illustrated in the slide



Implications from different Cloud Construction Approaches

Legacy Cloud = Premium Hardware

- Value-added high availability and performance in HW
  - Failure is considered an “exception” and dead units are Replaced
- Servers are often Blade Servers using a single vendor architecture
- “Brand Name” Ethernet switches are used
- SAN or NAS storage is used
  - Often with Fibre Channel
  - Along with SAN storage system which implements the replication and virtualization of storage
- System design tends towards homogenous elements to take full advantage of Blade Server manageability advantages

Web-Scale = Commodity Hardware

- Low cost “small, good, cheap, simple, and fast” HW
  - Failure is statistically inevitable, dead units are just Powered Off
- Servers are often no-name “rack and stack” servers, with mix’n’match vendors
- Lowest cost Ethernet switches are used
- Direct attached storage is used
  - Using built in SATA type interfaces
  - Software layer implements replication and virtualization of storage
- System design tends to mix’n’match elements with more investment on homegrown or third party system management

Implications from different Cloud Construction Approaches

Legacy Cloud = Premium Hardware
Web-Scale = Commodity Hardware

Diagram show many comparisons for example

Legacy Cloud, Value-added high availability and performance in HW
Failure is considered an “exception” and dead units are Replaced

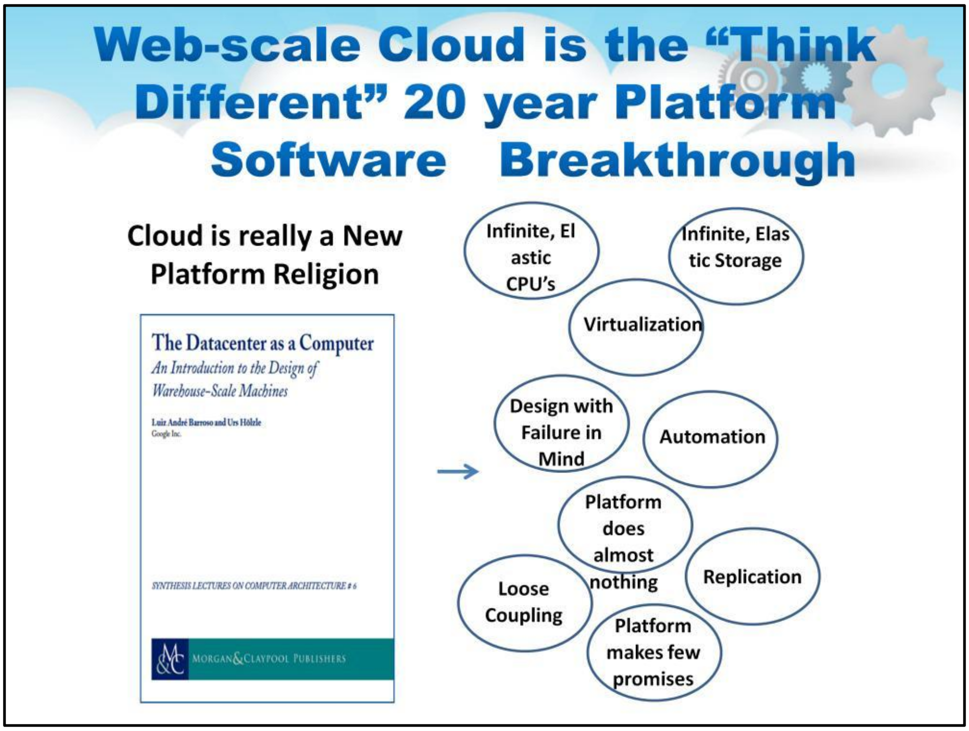
Web Scale cloud, Low cost “small, good, cheap, simple, and fast” HW
Failure is statistically inevitable, dead units are just Powered Off

Legacy Cloud, Servers are often Blade Servers using a single vendor architecture

Web Scale cloud, Servers are often no-name “rack and stack” servers, with mix’n’match vendors

Read the other areas of comparison

Conclusion
In Legacy Cloud System design tends towards homogenous elements
In Web Scale cloud, System design tends to mix’n’match elements



Web-scale Cloud is the “Think Different” 20 year Platform Software Breakthrough

Cloud is really a New Platform Religion

The Datacenter as a Computer

An Introduction to the Design of Warehouse-Scale Machines

Luiz André Barroso and Urs Hölzle
Google Inc.

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE # 6

MORGAN & CLAYPOOL PUBLISHERS

→

Infinite, El astic CPU’s

Infinite, Elas tic Storage

Virtualization

Design with Failure in Mind

Automation

Platform does almost nothing

Replication

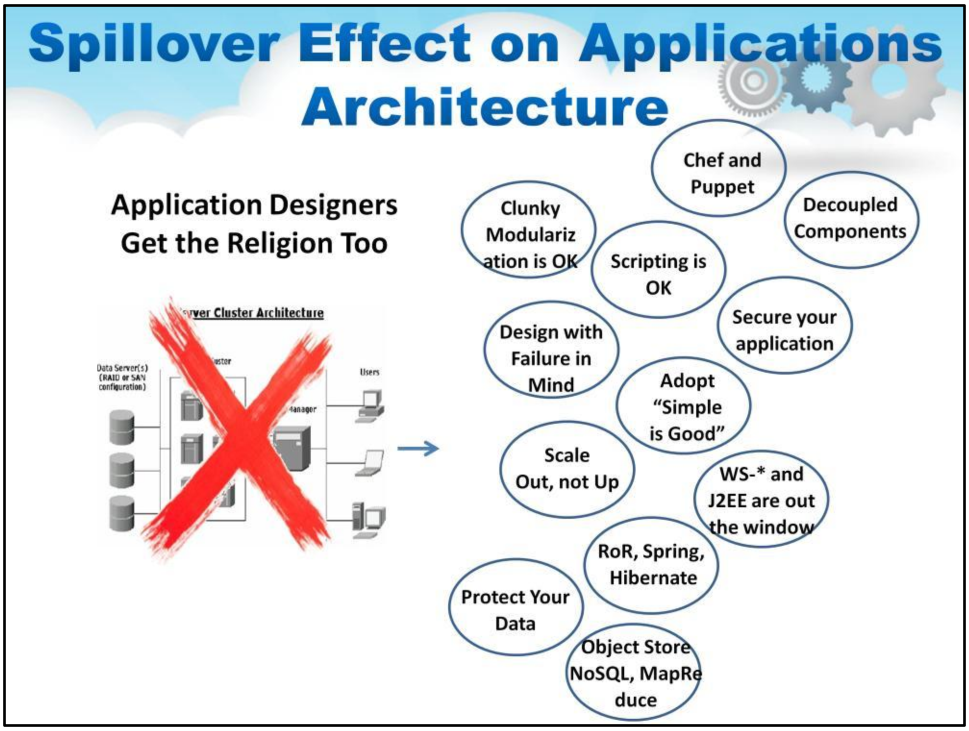
Loose Coupling

Platform makes few promises

Web-scale Cloud is the “Think Different” 20 year Platform Software Breakthrough

Cloud is really a New Platform Religion

Platform does almost nothing
Platform makes few promises
Virtualization
Automation
Infinite, Elastic Storage
Infinite, Elastic CPU’s



Spillover Effect on Applications Architecture

Application Designers Get the Religion Too

...rver Cluster Architecture
Data Server(s) (RAID or SAN configuration)
...uster
...anager
Users

Chef and Puppet

Clunky Modularization is OK

Decoupled Components

Scripting is OK

Secure your application

Design with Failure in Mind

Adopt "Simple is Good"

Scale Out, not Up

WS-* and J2EE are out the window

RoR, Spring, Hibernate

Protect Your Data

Object Store NoSQL, MapReduce

This has a Spillover Effect on Applications Architecture

One cannot bring the all the design patterns you know from enterprise multi-tier applications architecture or web architecture, while they are largely applicable many key areas are re-thought for cloud.

Applications must themselves now take care of: Scale Out, not Up, Adopt "Simple is Good", Design with Failure in Mind, and so on.

Key Technology Breakthroughs Enabling Cloud Building

No Frills Server w/Storage

Fast and Flat L2/L3 Networking

Cloud OS, Hypervisor

Hi Density, Green Datacenter

hadoop
amazon webservices
Cassandra
OPSCODE
Google Apps
Windows Azure
Microsoft's Cloud Services Platform
Xen
vmware
Hyper-V
KVM

Key Technology Breakthroughs enabling Cloud Building

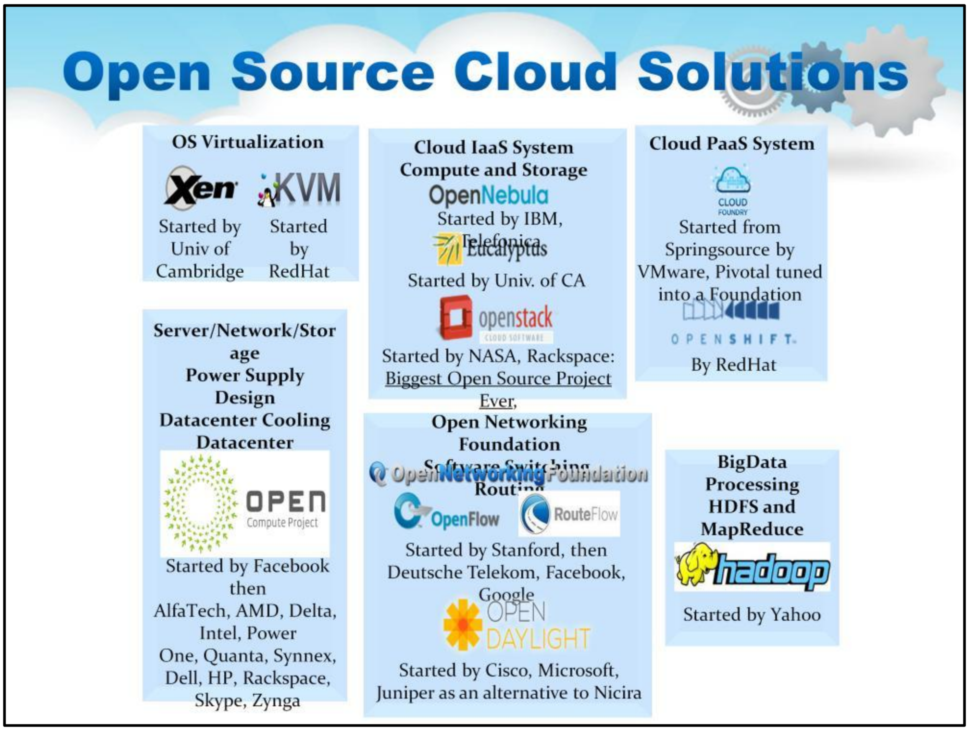
Certain technologies have aided in the construction of these large, web-scale clouds

As mentioned, commodity servers which are so-called "no frills" are emerging, minimizing fancy cabinets, blinking lights and extra USB ports – they are high memory footprint small servers with direct attached storage, perfect as cloud servers

Add to this networking which is easy to configure as a "Spine and Leaf" configuration for lots of top-of-rack switches to be conectedto each other either through an aggregation layer or directly as a spine.

Then, the data center itself has evolved to support very dense (high wattage) racks, and dense electricity

Of course, there are lots of open source elements which ae crucul n running all this. Many of the important modules are illustrated in the slide.



Open Source Cloud Solutions

OS Virtualization

Xen — Started by Univ of Cambridge

KVM — Started by RedHat

Server/Network/Storage Power Supply Design Datacenter Cooling Datacenter

OPEN Compute Project

Started by Facebook then AlfaTech, AMD, Delta, Intel, Power One, Quanta, Synnex, Dell, HP, Rackspace, Skype, Zynga

Cloud IaaS System Compute and Storage

OpenNebula — Started by IBM, Telefonica

Eucalyptus — Started by Univ. of CA

openstack CLOUD SOFTWARE — Started by NASA, Rackspace: Biggest Open Source Project Ever,

Open Networking Foundation Software Switching Routing

OpenNetworkingFoundation

OpenFlow RouteFlow

Started by Stanford, then Deutsche Telekom, Facebook, Google

OPEN DAYLIGHT

Started by Cisco, Microsoft, Juniper as an alternative to Nicira

Cloud PaaS System

CLOUD FOUNDRY — Started from Springsource by VMware, Pivotal tuned into a Foundation

OPENSHIFT — By RedHat

BigData Processing HDFS and MapReduce

hadoop — Started by Yahoo

Open Source Solutions for Cloud

Open Source has been extremely important in the technological advancement of Cloud. This chart illustrates all the areas when Open Source has been crucial. Amazingly in many areas there are more than one Open Source areas solving the problem.

On the hypervisor side we have both Xen and KVM as open source alternatives. These now come standard with Linux distributions.

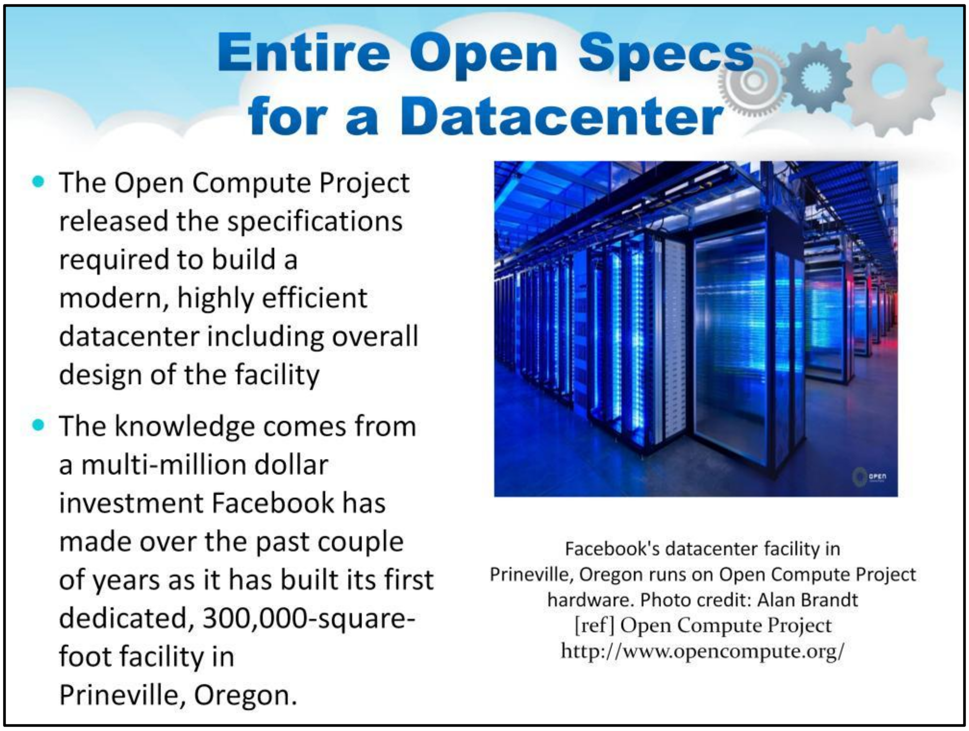
On the IaaS system, there is the original open source cloud system from Europe (OpenNebula) as well as OpenStack itself, the larget open source project in the history of mankind.

There are open source PaaS systems, from both Pivotal and RedHat.

There are lot of Big Data open source tools beginning with the Hadoop community and going on from there, with many projects in Apache.

Finally you will see some interesting initiatives, one called Open Compute project, is all about the physical parts of building a datacenter, from power and cooling and wiring techniques, to bare bones servers and storage modules. You will also see two organizations looking at Software Defined Networks (SDN) standards.

It is really useful to go to each of these sites and to take a look at each initiative. It is clear that Open Source is a very large driving factor in the technology going into cloud computing.



Entire Open Specs for a Datacenter

- The Open Compute Project released the specifications required to build a modern, highly efficient datacenter including overall design of the facility
- The knowledge comes from a multi-million dollar investment Facebook has made over the past couple of years as it has built its first dedicated, 300,000-square-foot facility in Prineville, Oregon.

Facebook's datacenter facility in Prineville, Oregon runs on Open Compute Project hardware. Photo credit: Alan Brandt
[ref] Open Compute Project
http://www.opencompute.org/

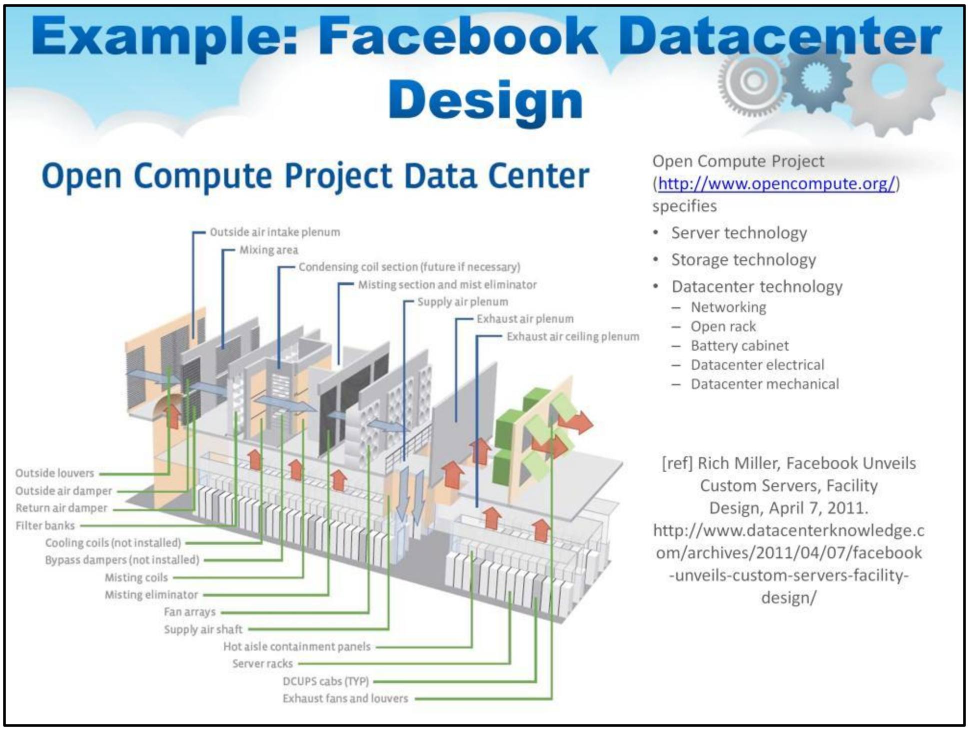
Entire Open Specs for a Datacenter

From the Open Compute Project (founded by Facebook in 2011), which released the specifications required to build a modern, highly efficient datacenter including overall design of the facility

The slide shows a photo of Facebook's datacenter facility in Prineville, Oregon which runs on Open Compute Project hardware.

The knowledge comes from a multi-million dollar investment Facebook has made over the past couple of years as it has built its first dedicated, 300,000-square-foot facility in Prineville, Oregon

Many other companies have since contributed to these specs.



Example: Facebook Datacenter Design

Open Compute Project Data Center

Outside air intake plenum
Mixing area
Condensing coil section (future if necessary)
Misting section and mist eliminator
Supply air plenum
Exhaust air plenum
Exhaust air ceiling plenum
Outside louvers
Outside air damper
Return air damper
Filter banks
Cooling coils (not installed)
Bypass dampers (not installed)
Misting coils
Misting eliminator
Fan arrays
Supply air shaft
Hot aisle containment panels
Server racks
DCUPS cabs (TYP)
Exhaust fans and louvers

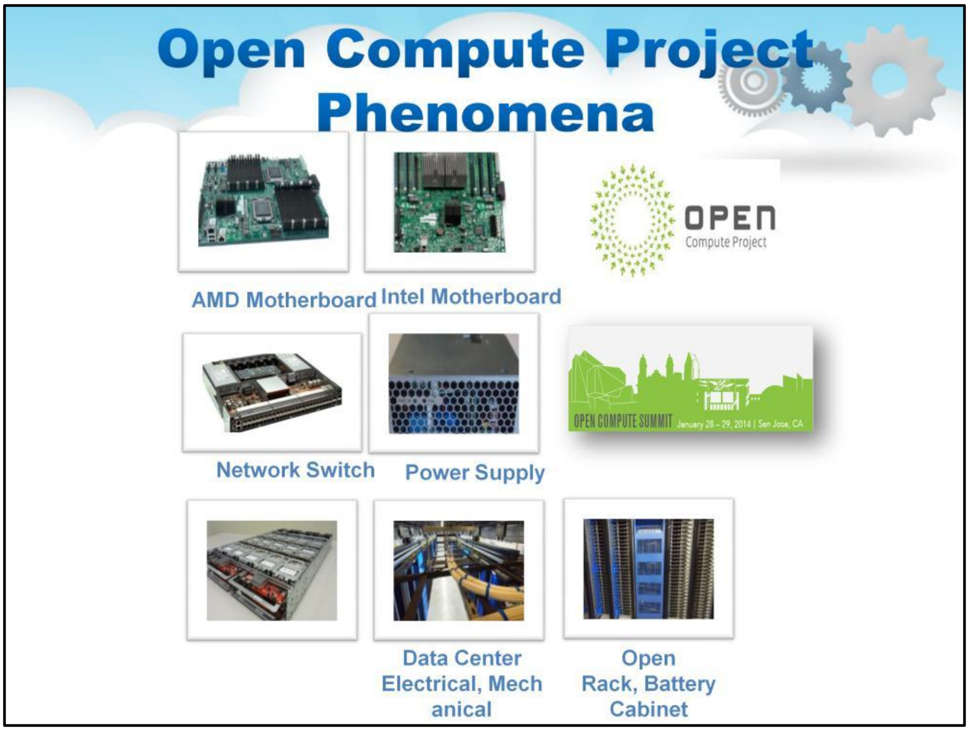
Open Compute Project (http://www.opencompute.org/) specifies

- Server technology
- Storage technology
- Datacenter technology
  - Networking
  - Open rack
  - Battery cabinet
  - Datacenter electrical
  - Datacenter mechanical

[ref] Rich Miller, Facebook Unveils Custom Servers, Facility Design, April 7, 2011. http://www.datacenterknowledge.com/archives/2011/04/07/facebook-unveils-custom-servers-facility-design/

Example

This slide contains an illustration showing details on the Facebook Datacenter design, with an emphasis on cooling design. Please study the diagram.



Open Compute Project Phenomena

AMD Motherboard

Intel Motherboard

Network Switch

Power Supply

Data Center Electrical, Mech anical

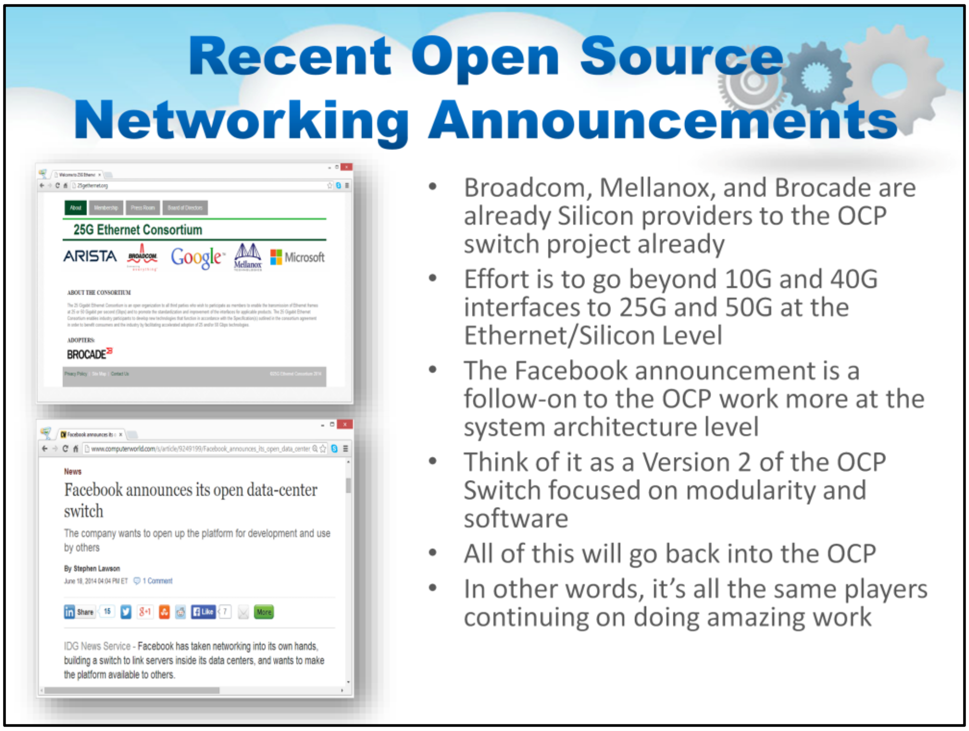
Open Rack, Battery Cabinet

Open Compute Project Phenomena

In the Open Compute Project there are actually many sub projects. The slide contains illustrations of the major projects.

You will see server (motherboard) specs, network switch specs, power supply specs, storage system specs, and specs for electrical, mechanical, even battery cabinets.

The Open Compute Project has it’s own conference now in Silicon Valley. It is a very interesting phenomena.



Recent Open Source Networking Announcements

25G Ethernet Consortium

ABOUT THE CONSORTIUM

The 25 Gigabit Ethernet Consortium is an open organization to all third parties who wish to participate as members to enable the transmission of Ethernet frames at 25 or 50 Gigabit per second (Gbps) and to promote the standardization and improvement of the interfaces for applicable products. The 25 Gigabit Ethernet Consortium enables industry participants to develop new technologies that function in accordance with the Specification(s) outlined in the consortium agreement in order to benefit consumers and the industry by facilitating accelerated adoption of 25 and/or 50 Gbps technologies.

ADOPTERS:

News

Facebook announces its open data-center switch

The company wants to open up the platform for development and use by others

By Stephen Lawson

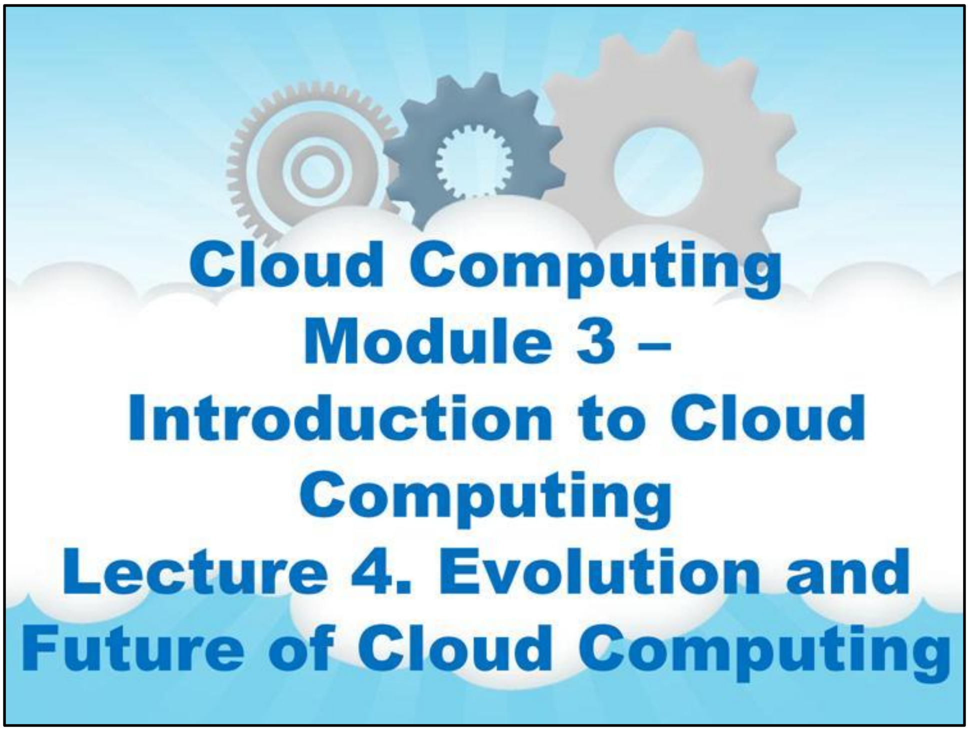
June 18, 2014 04:04 PM ET 1 Comment

IDG News Service - Facebook has taken networking into its own hands, building a switch to link servers inside its data centers, and wants to make the platform available to others.

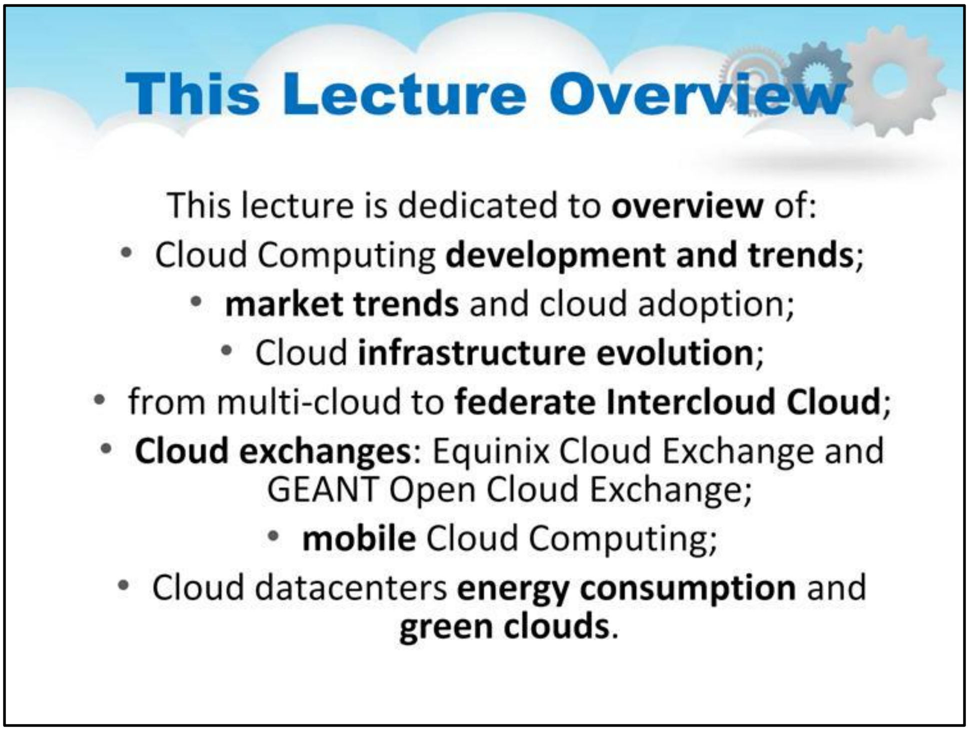
- Broadcom, Mellanox, and Brocade are already Silicon providers to the OCP switch project already
- Effort is to go beyond 10G and 40G interfaces to 25G and 50G at the Ethernet/Silicon Level
- The Facebook announcement is a follow-on to the OCP work more at the system architecture level
- Think of it as a Version 2 of the OCP Switch focused on modularity and software
- All of this will go back into the OCP
- In other words, it's all the same players continuing on doing amazing work

Recently, open source announcements regarding data center switches were made. At first these seemed confusing, especially because the announcements were made outside of the Open Compute project. On closer inspection, we see that the two announcements, as illustrated in the slide, were actually made by many of the same companies who are part of the Open Compute Project.

What all this means is that the era of needing proprietary networking equipment to access massive performance and scalability are over, commodity solutions are becoming extremely robust.



Cloud Computing Module 3 – Introduction to Cloud Computing Lecture 4. Evolution and Future of Cloud Computing



This Lecture Overview

This lecture is dedicated to **overview** of:

- Cloud Computing **development and trends**;
- **market trends** and cloud adoption;
- Cloud **infrastructure evolution**;
- from multi-cloud to **federate Intercloud Cloud**;
- **Cloud exchanges**: Equinix Cloud Exchange and GEANT Open Cloud Exchange;
- **mobile** Cloud Computing;
- Cloud datacenters **energy consumption** and **green clouds**.

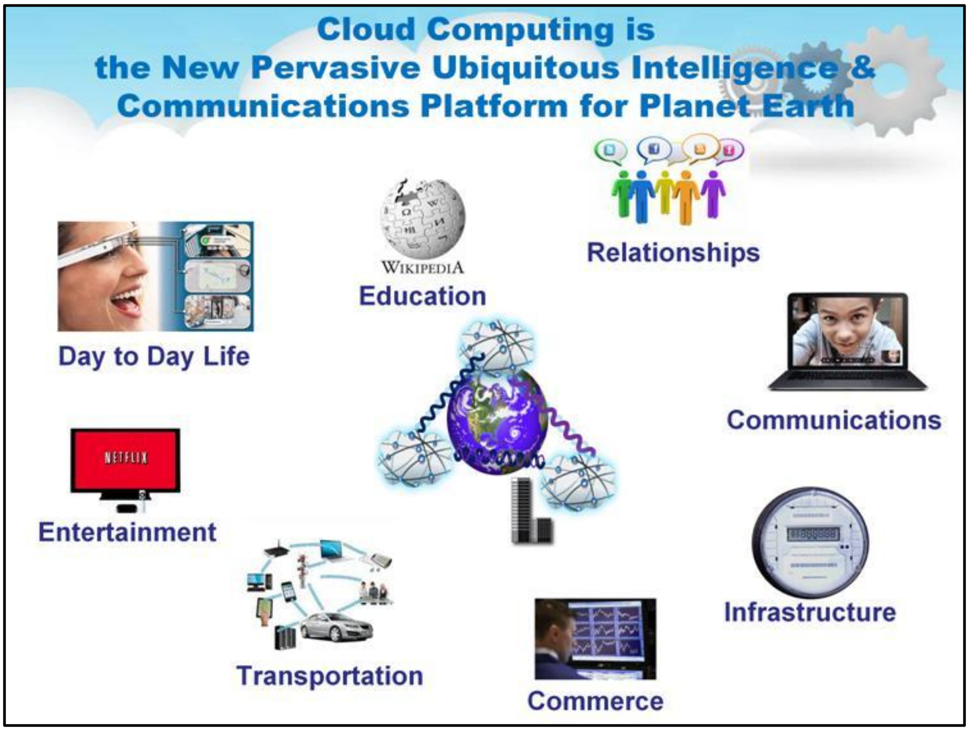
Lecture 4. Evolution and Future of Cloud Computing

Developments and Trends

–

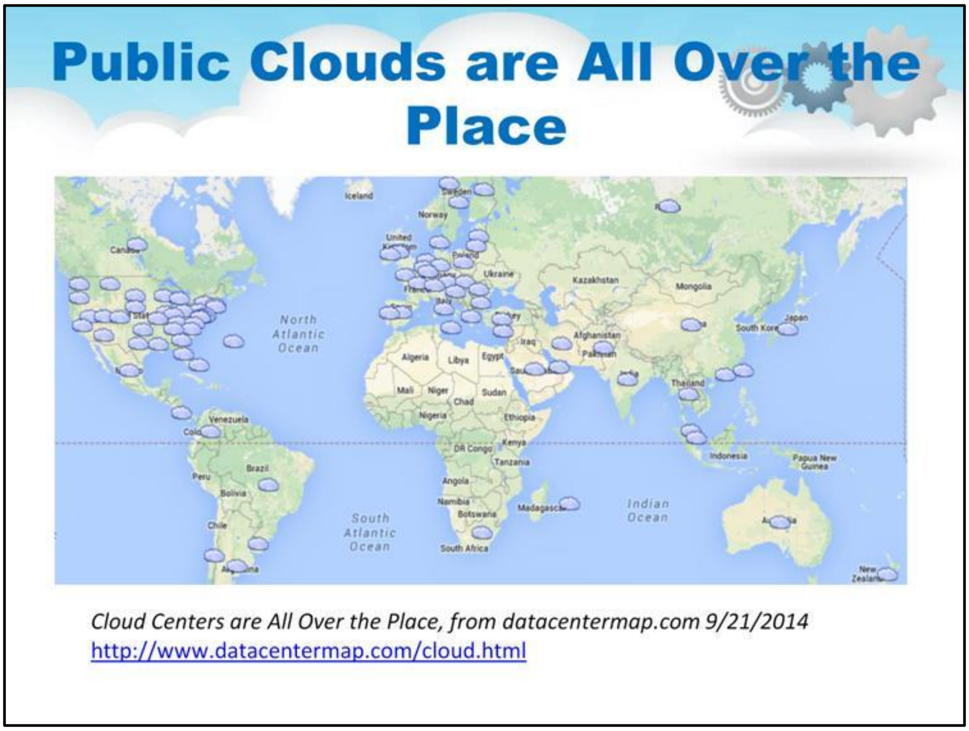
Overview

Overview



Cloud Computing is the New Pervasive Ubiquitous Intelligence & Communications Platform for Planet Earth

Cloud Computing is the New Pervasive Ubiquitous Intelligence & Communications Platform for Planet Earth
Education
Relationships
Communications
Internet of Things Infrastructure
Commerce
Transportation
Entertainment
Day to Day Life!

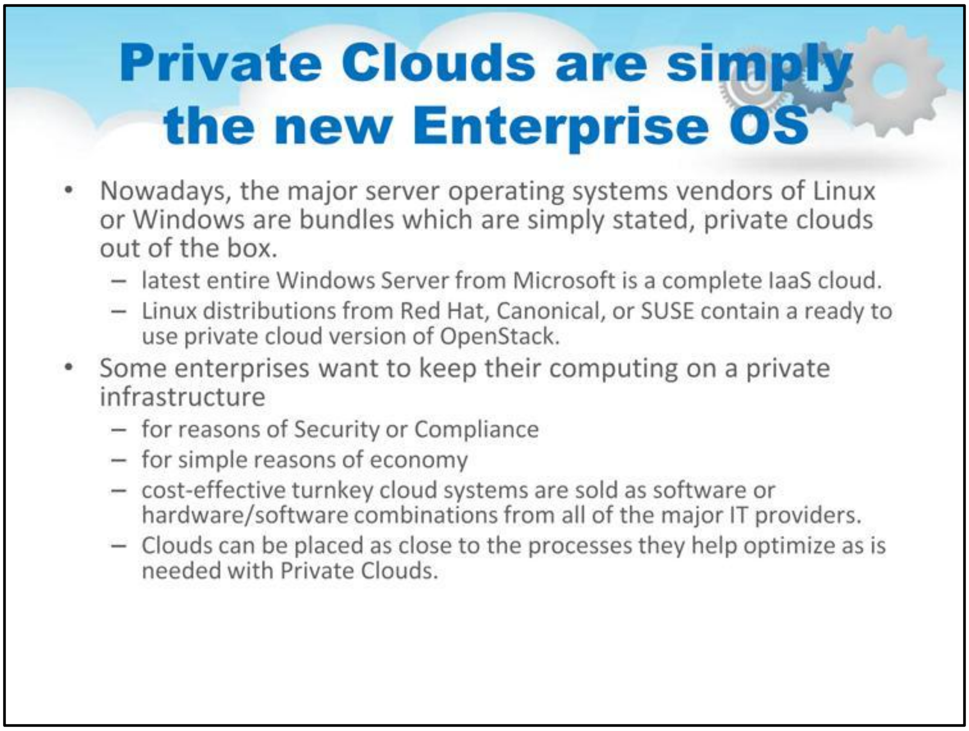


Public Clouds are All Over the Place

Cloud Centers are All Over the Place, from datacentermap.com 9/21/2014
http://www.datacentermap.com/cloud.html

Clouds are becoming more ubiquitous than one might think. Cloud Service Providers count from thousands to over a million servers per cloud. According to datacentermap.com, there are hundreds of such clouds now serving much of the civilized population of the world. They come from companies such as Google and Amazon and Microsoft and IBM and HP, and they also come from the local Telecom companies, in pretty much a the country of one’s choice, including Verizon, AT&T, NTT, Orange/FT, DT, BT, and so on. Many of these Telecoms will have their Clouds well integrated with their mobile networks, facilitating the use of Big and Fast Data capabilities for mobile apps as well as with M2M/IoT.

http://www.datacentermap.com/cloud.html

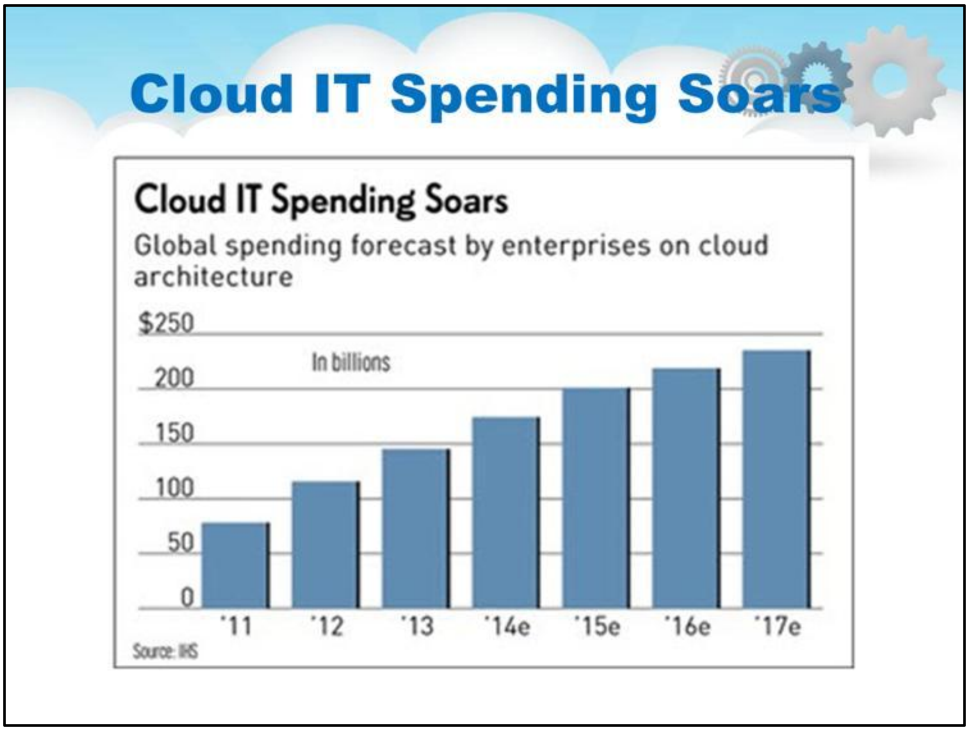


Private Clouds are simply the new Enterprise OS

- Nowadays, the major server operating systems vendors of Linux or Windows are bundles which are simply stated, private clouds out of the box.
  - latest entire Windows Server from Microsoft is a complete IaaS cloud.
  - Linux distributions from Red Hat, Canonical, or SUSE contain a ready to use private cloud version of OpenStack.
- Some enterprises want to keep their computing on a private infrastructure
  - for reasons of Security or Compliance
  - for simple reasons of economy
  - cost-effective turnkey cloud systems are sold as software or hardware/software combinations from all of the major IT providers.
  - Clouds can be placed as close to the processes they help optimize as is needed with Private Clouds.

Nowadays, the major server operating systems vendors of Linux or Windows are bundles which are simply stated, private clouds out of the box. Install the latest entire Windows Server from Microsoft and you will find it's a complete IaaS cloud. Likewise, the distributions of Linux from Red Hat, Canonical, or SUSE contain a ready to use private cloud version of OpenStack.

Some enterprises want to keep their computing on a private infrastructure for reasons of Security or Compliance, some for simple reasons of economy; cost-effective turnkey cloud systems are sold as software or hardware/software combinations from all of the major IT providers. Clouds can be placed as close to the processes they help optimize as is needed with Private Clouds.



Cloud IT Spending Soars

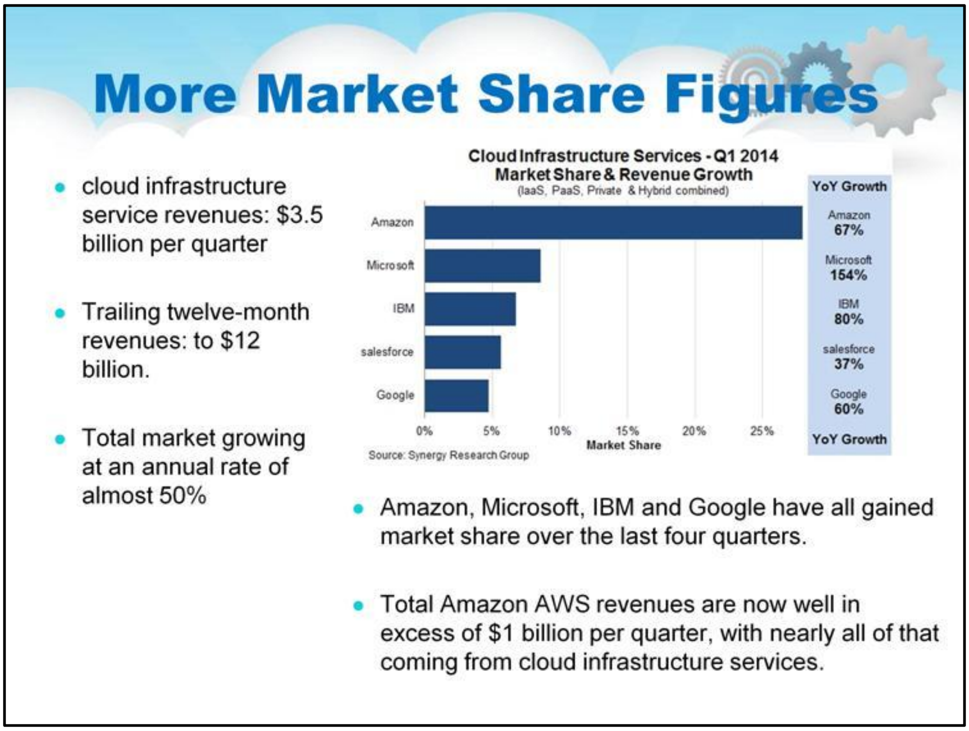
Cloud IT Spending Soars

Global spending forecast by enterprises on cloud architecture

Source: IHS

Cloud IT Spending Soars

The slide shows a graph following the overall spending on Cloud IT Globally
As one can see the worldwide investment in Cloud Computing will exceed USD $200B soon



More Market Share Figures

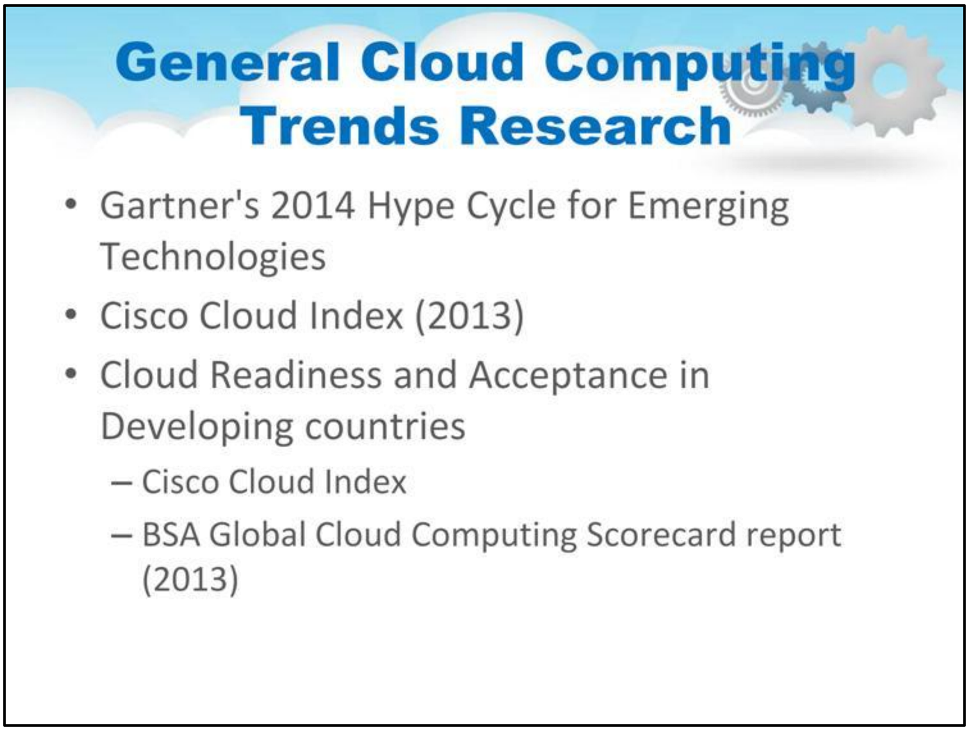
- cloud infrastructure service revenues: $3.5 billion per quarter
- Trailing twelve-month revenues: to $12 billion.
- Total market growing at an annual rate of almost 50%

Source: Synergy Research Group

- Amazon, Microsoft, IBM and Google have all gained market share over the last four quarters.
- Total Amazon AWS revenues are now well in excess of $1 billion per quarter, with nearly all of that coming from cloud infrastructure services.

Exploring More Market Share Figures

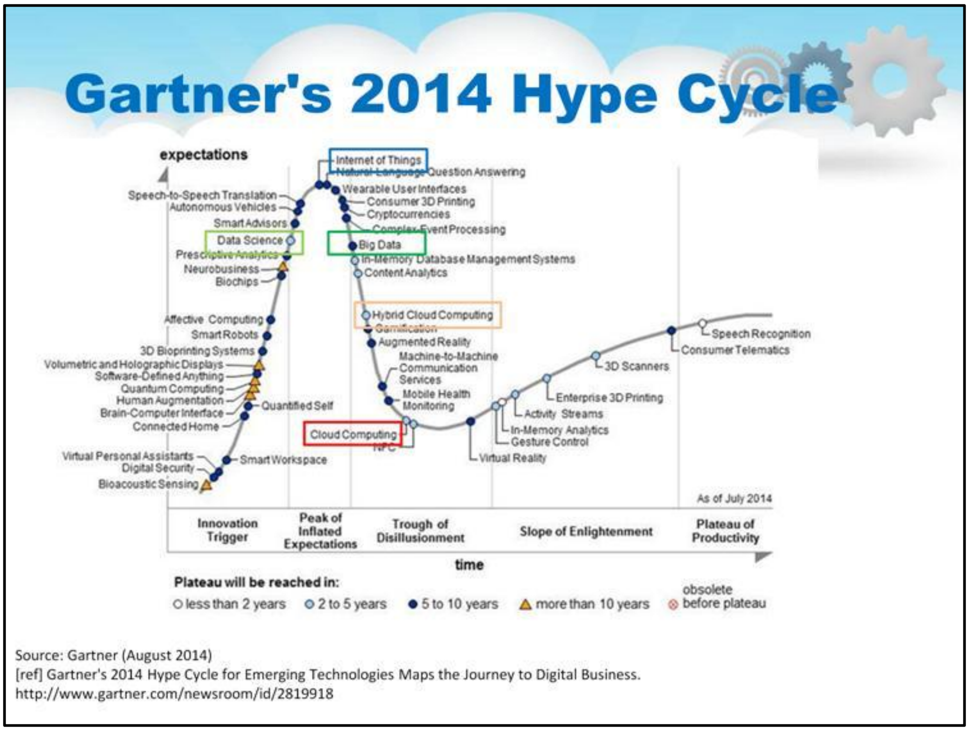
- Estimates that quarterly cloud infrastructure service revenues (including IaaS, PaaS and private & hybrid cloud) have reached $3.5 billion
- Trailing twelve-month revenues amounting to $12 billion.
- Total market growing at an annual rate of almost 50%
- Amazon, Microsoft, IBM and Google have all gained market share over the last four quarters.
- Total Amazon AWS revenues are now well in excess of $1 billion per quarter, with nearly all of that coming from cloud infrastructure services.



General Cloud Computing Trends Research

- Gartner's 2014 Hype Cycle for Emerging Technologies
- Cisco Cloud Index (2013)
- Cloud Readiness and Acceptance in Developing countries
  - Cisco Cloud Index
  - BSA Global Cloud Computing Scorecard report (2013)

Next we will look at several General Cloud Computing Trends Research

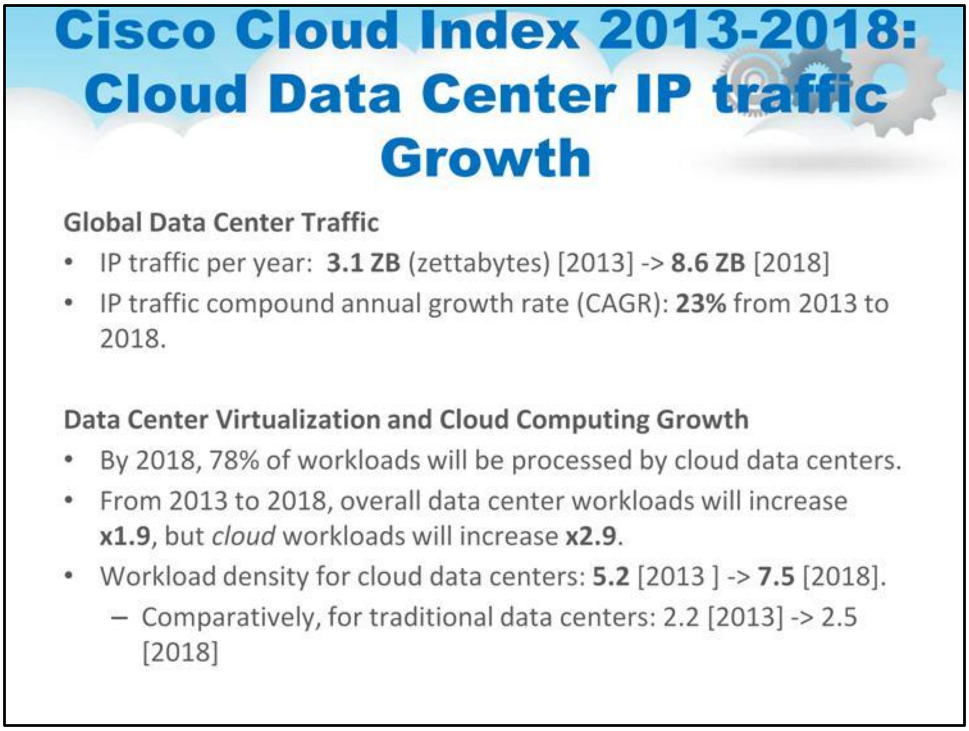


Gartner's 2014 Hype Cycle

Source: Gartner (August 2014)
[ref] Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business.
http://www.gartner.com/newsroom/id/2819918

Cloud Computing is approaching productivity area via massive implementation meets reality, followed with Hybrid Cloud Computing that adopts clouds into the transformed enterprise IT environment.
At the same time, it is obvious that Cloud Computing create a strong basis for many technologies currently been on rise and buzz words such as Big Data, Data Science .
Cloud Computing also provides a new rich platform for Internet of Thing (sometimes also called with broader term Internet of Everything).

Next slides and studies will explore different numerical estimations of the different aspects pf Cloud Computing development.



Cisco Cloud Index 2013-2018: Cloud Data Center IP traffic Growth

Global Data Center Traffic

- IP traffic per year: **3.1 ZB** (zettabytes) [2013] -> **8.6 ZB** [2018]
- IP traffic compound annual growth rate (CAGR): **23%** from 2013 to 2018.

Data Center Virtualization and Cloud Computing Growth

- By 2018, 78% of workloads will be processed by cloud data centers.
- From 2013 to 2018, overall data center workloads will increase **x1.9**, but *cloud* workloads will increase **x2.9**.
- Workload density for cloud data centers: **5.2** [2013] -> **7.5** [2018].
  - Comparatively, for traditional data centers: 2.2 [2013] -> 2.5 [2018]

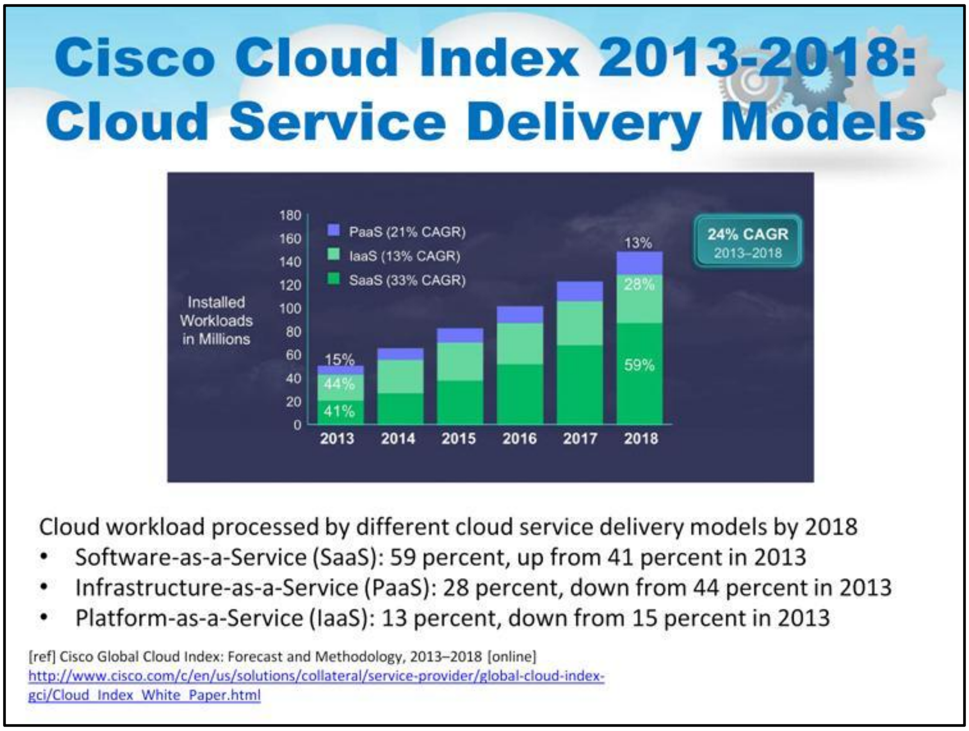
The Cisco Global Cloud Index (GCI) is an ongoing effort to forecast the growth of global data center and cloud-based IP traffic. The forecast includes trends associated with data center virtualization and cloud computing.
This document presents the details of the study and the methodology behind it.

Global Data Center Traffic

- Annual global data center IP traffic will reach 8.6 zettabytes (715 exabytes [EB] per month) by the end of 2018, up from 3.1 zettabytes (ZB) per year (255 EB per month) in 2013.
- Global data center IP traffic will nearly triple (2.8-fold) over the next 5 years. Overall, data center IP traffic will grow at a compound annual growth rate (CAGR) of 23 percent from 2013 to 2018.

Data Center Virtualization and Cloud Computing Growth

- By 2018, more than three quarters (78 percent) of workloads will be processed by cloud data centers; 22 percent will be processed by traditional data centers.
- Overall data center workloads will nearly double (1.9-fold) from 2013 to 2018; however, cloud workloads will nearly triple (2.9-fold) over the same period.
- The workload density (that is, workloads per physical server) for cloud data centers was 5.2 in 2013 and will grow to 7.5 by 2018. Comparatively, for traditional data centers, workload density was 2.2 in 2013 and will grow to 2.5 by 2018.



Cisco Cloud Index 2013-2018: Cloud Service Delivery Models

Cloud workload processed by different cloud service delivery models by 2018

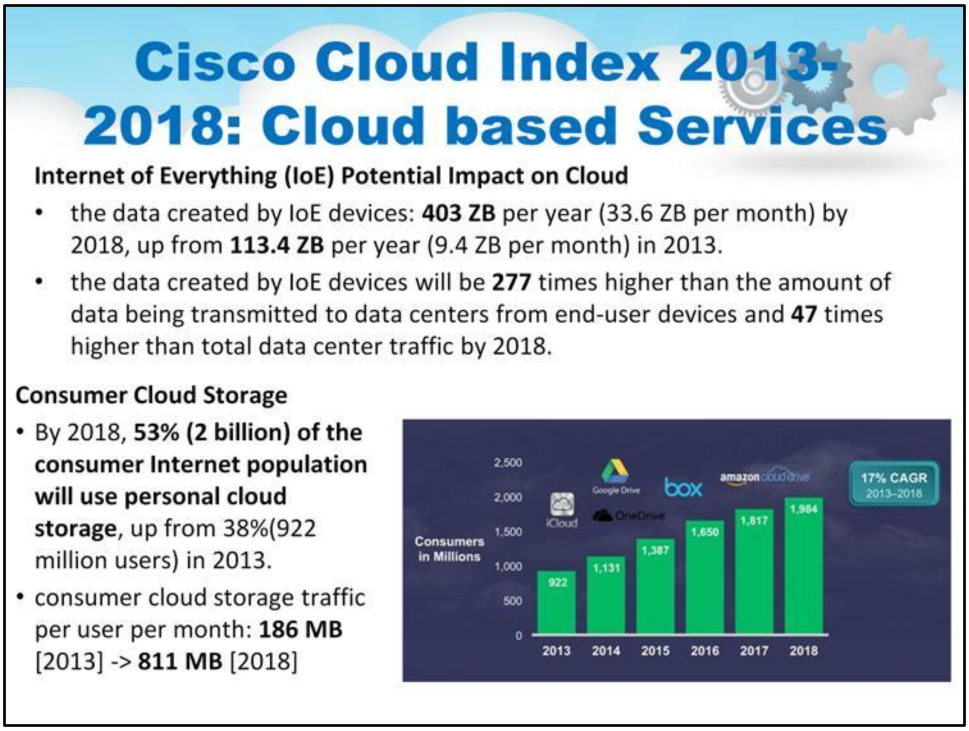
- Software-as-a-Service (SaaS): 59 percent, up from 41 percent in 2013
- Infrastructure-as-a-Service (PaaS): 28 percent, down from 44 percent in 2013
- Platform-as-a-Service (IaaS): 13 percent, down from 15 percent in 2013

[ref] Cisco Global Cloud Index: Forecast and Methodology, 2013–2018 [online] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

Important to notice changes in the cloud processed workload by different Cloud Service Delivery Models what will indicate financial market share

- By 2018, 59 percent of the total cloud workloads will be Software-as-a-Service (SaaS) workloads, up from 41 percent in 2013.
- By 2018, 28 percent of the total cloud workloads will be Infrastructure-as-a-Service (IaaS) workloads, down from 44 percent in 2013.
- By 2018, 13 percent of the total cloud workloads will be Platform-as-a-Service (PaaS) workloads, down from 15 percent in 2013.

The trend actually reflects typical cloud implementation and evolution at enterprises: From initial IaaS implementation through PaaS and common applications development platform to SaaS type operational services



Cisco Cloud Index 2013-2018: Cloud based Services

Internet of Everything (IoE) Potential Impact on Cloud

- the data created by IoE devices: **403 ZB** per year (33.6 ZB per month) by 2018, up from **113.4 ZB** per year (9.4 ZB per month) in 2013.
- the data created by IoE devices will be **277** times higher than the amount of data being transmitted to data centers from end-user devices and **47** times higher than total data center traffic by 2018.

Consumer Cloud Storage

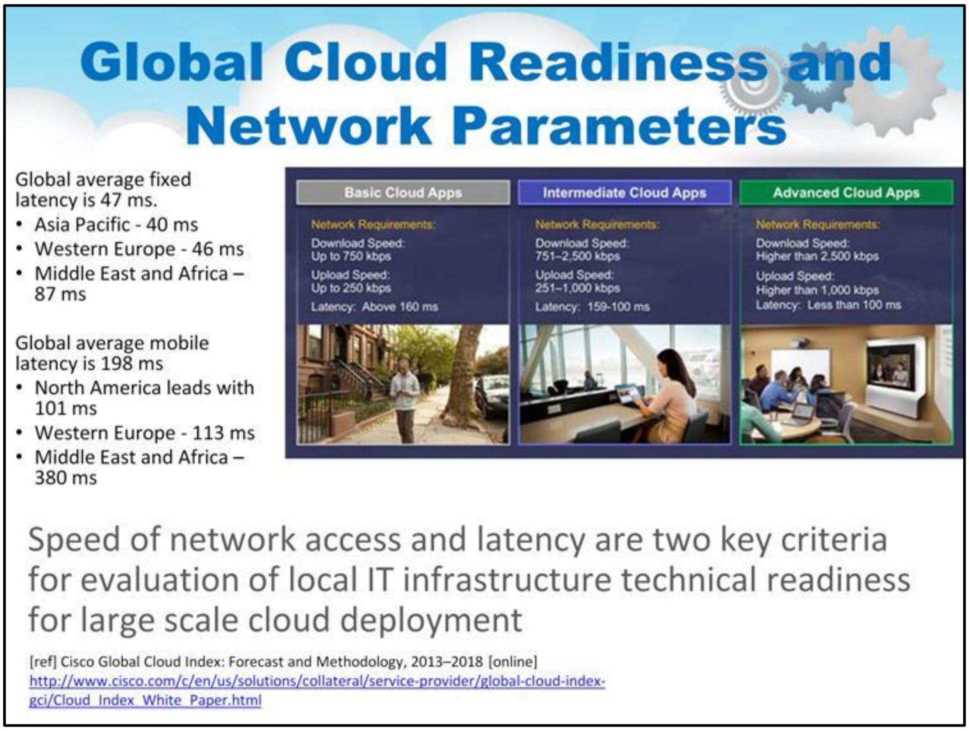
- By 2018, **53% (2 billion) of the consumer Internet population will use personal cloud storage**, up from 38%(922 million users) in 2013.
- consumer cloud storage traffic per user per month: **186 MB** [2013] -> **811 MB** [2018]

Internet of Everything (IoE) Potential Impact on Cloud

- Globally, the data created by IoE devices will reach 403 ZB per year (33.6 ZB per month) by 2018, up from 113.4 ZB per year (9.4 ZB per month) in 2013.
- Globally, the data created by IoE devices will be 277 times higher than the amount of data being transmitted to data centers from end-user devices and 47 times higher than total data center traffic by 2018.

Consumer Cloud Storage

- By 2018, 53 percent (2 billion) of the consumer Internet population will use personal cloud storage, up from 38 percent (922 million users) in 2013.
- Globally, consumer cloud storage traffic per user will be 811 megabytes per month by 2018, compared to 186 megabytes per month in 2013.



Global Cloud Readiness and Network Parameters

Global average fixed latency is 47 ms.

- Asia Pacific - 40 ms
- Western Europe - 46 ms
- Middle East and Africa – 87 ms

Global average mobile latency is 198 ms

- North America leads with 101 ms
- Western Europe - 113 ms
- Middle East and Africa – 380 ms

Basic Cloud Apps	Intermediate Cloud Apps	Advanced Cloud Apps
Network Requirements:	Network Requirements:	Network Requirements:
Download Speed: Up to 750 kbps	Download Speed: 751–2,500 kbps	Download Speed: Higher than 2,500 kbps
Upload Speed: Up to 250 kbps	Upload Speed: 251–1,000 kbps	Upload Speed: Higher than 1,000 kbps
Latency: Above 160 ms	Latency: 159-100 ms	Latency: Less than 100 ms

Speed of network access and latency are two key criteria for evaluation of local IT infrastructure technical readiness for large scale cloud deployment

[ref] Cisco Global Cloud Index: Forecast and Methodology, 2013–2018 [online] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

Network latency is crucial for running cloud based application because majority of reliable communication protocols require feedback (in particular TCP) and many cloud applications are interactive and near real time (such as voice recognition in smartphones, business applications or gaming)

The cloud-readiness study offers a regional view of the requirements for broadband and mobile networks to deliver next-generation cloud services

The study also explored the ability of each global region (Asia Pacific, Central and Eastern Europe, Latin America, Middle East and Africa, North America, and Western Europe) to support a sample set of basic, intermediate, and advanced business and consumer cloud applications.

Speed of network access and latency are two key criteria for evaluation of local IT infrastructure technical readiness for large scale cloud deployment

Cloud Readiness and Acceptance in Developing countries

The 2013 BSA Global Cloud Computing Scorecard report (http://cloudscorecard.bsa.org/2013/index.html)

- **24 countries** that are included together account for **80%**of the global information and communication technologies (ICT) market.

The economic growth and transformation from wide use of Cloud Computing require the proper policies in each of the seven areas:

- Ensuring privacy and data protection regulation
- Consistent security services and compliance with best practices in security
- Legal system and measure to protect against cybercrime
- Protecting intellectual property to support innovation and content flow crossborder and between cloud providers
- Ensuring unrestricted or simply regulated data move, harmonization of international regulations
- Existence of the necessary IT infrastructure: Cloud computing requires robust, ubiquitous, and affordable broadband access

Cloud Readiness and Acceptance in Developing countries

The next slide has an illustration which comes from The 2013 BSA Global Cloud Computing Scorecard

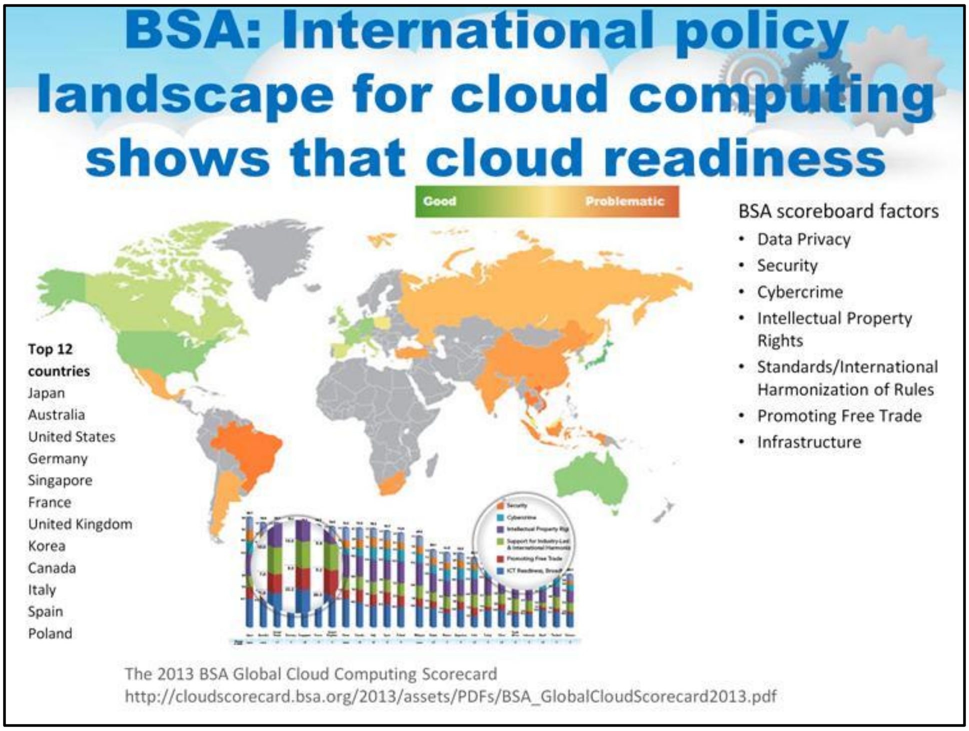
This report ranks countries' preparedness to support the growth of cloud computing

The 24 countries are included which together account for 80 percent of the global information and communication technologies (ICT) market.

The economic growth and transformation of both businesses and national economies from wide use of Cloud Computing require the proper policies in each of the seven areas:
Ensuring privacy and data protection regulation
Consistent security services and compliance with best practices in security
Legal system and measure to protect against cybercrime
Protecting intellectual property to support innovation and content flow crossborder and between cloud providers
Ensuring unrestricted or simply regulated data move, harmonization of international regulations
Existence of the necessary IT infrastructure: Cloud computing requires robust, ubiquitous, and affordable broadband access

Developments and Trends – Security and Privacy

Security and Privacy



BSA: International policy landscape for cloud computing shows that cloud readiness

BSA scoreboard factors

- Data Privacy
- Security
- Cybercrime
- Intellectual Property Rights
- Standards/International Harmonization of Rules
- Promoting Free Trade
- Infrastructure

Top 12 countries
Japan
Australia
United States
Germany
Singapore
France
United Kingdom
Korea
Canada
Italy
Spain
Poland

The 2013 BSA Global Cloud Computing Scorecard
http://cloudscorecard.bsa.org/2013/assets/PDFs/BSA_GlobalCloudScorecard2013.pdf

BSA scoreboard factors include

Data Privacy

Cloud users will fully accept and adopt cloud computing only if they are confident that private information stored in the cloud, wherever in the world, will not be used or disclosed by the cloud provider in unexpected ways.

Security

Consumers of cloud computing and other digital services (including both private-sector and government users) need assurance that cloud service providers understand and appropriately manage the security risks associated with storing their data and running their applications on cloud systems

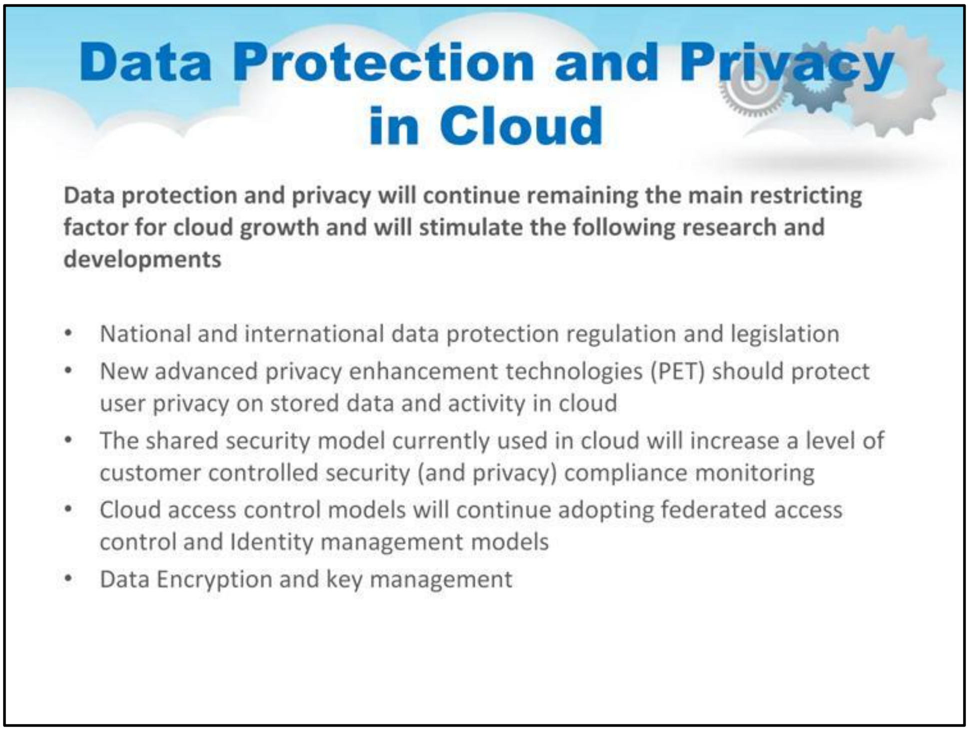
Cybercrime

Because cloud computing involves the aggregation of massive amounts of data in large data centers, it creates new and highly tempting targets. As criminals turn their attention to these vaults of information, it will become increasingly challenging to protect such data centers from both physical and cyberattacks.

Intellectual Property Rights

Providers of cloud computing and digital economy technologies and services, as with other highly innovative products, rely on a combination of patents, copyrights, trade secrets, and other forms of intellectual property protection.

Standards/International Harmonization of Rules



Data Protection and Privacy in Cloud

Data protection and privacy will continue remaining the main restricting factor for cloud growth and will stimulate the following research and developments

- National and international data protection regulation and legislation
- New advanced privacy enhancement technologies (PET) should protect user privacy on stored data and activity in cloud
- The shared security model currently used in cloud will increase a level of customer controlled security (and privacy) compliance monitoring
- Cloud access control models will continue adopting federated access control and Identity management models
- Data Encryption and key management

Data Protection and Privacy in Cloud

Data protection and privacy will continue remaining the main restricting factor for cloud growth and will stimulate the following research and developments

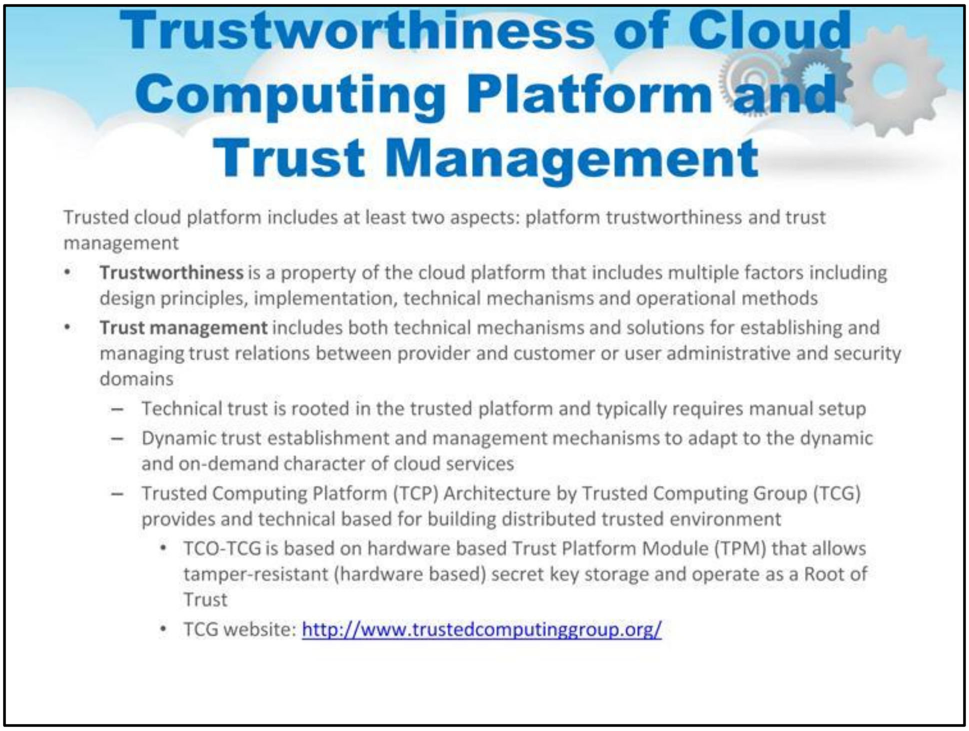
National and international data protection regulation and legislation that should be supported by corresponding certification and assessment procedures, services and bodies

New advanced privacy enhancement technologies (PET) should protect user privacy on stored data and activity in cloud
- Emergence of Big Data technologies will require new PET preventing user (re-)identification based on correlation between multiple sources of data

The shared security model currently used in cloud will increase a level of customer controlled security (and privacy) compliance monitoring

Cloud access control models will continue adopting federated access control and Identity management models
- While based on initially verified user identity the access control methods should not reveal user identity to cloud service providers and 3rd party services

Data Encryption and key management
- New encrypted data processing models such as homomorphic encryption
- New key management models



Trustworthiness of Cloud Computing Platform and Trust Management

Trusted cloud platform includes at least two aspects: platform trustworthiness and trust management

- **Trustworthiness** is a property of the cloud platform that includes multiple factors including design principles, implementation, technical mechanisms and operational methods
- **Trust management** includes both technical mechanisms and solutions for establishing and managing trust relations between provider and customer or user administrative and security domains
  - Technical trust is rooted in the trusted platform and typically requires manual setup
  - Dynamic trust establishment and management mechanisms to adapt to the dynamic and on-demand character of cloud services
  - Trusted Computing Platform (TCP) Architecture by Trusted Computing Group (TCG) provides and technical based for building distributed trusted environment
    - TCO-TCG is based on hardware based Trust Platform Module (TPM) that allows tamper-resistant (hardware based) secret key storage and operate as a Root of Trust
    - TCG website: http://www.trustedcomputinggroup.org/

Trustworthiness of Cloud Computing Platform and Trust Management

Trusted cloud platform includes at least two aspects: platform trustworthiness and trust management

Trustworthiness is a property of the cloud platform that includes multiple factors including design principles, implementation, technical mechanisms and operational methods

Trustworthiness by design is a key to create and trusted cloud platform

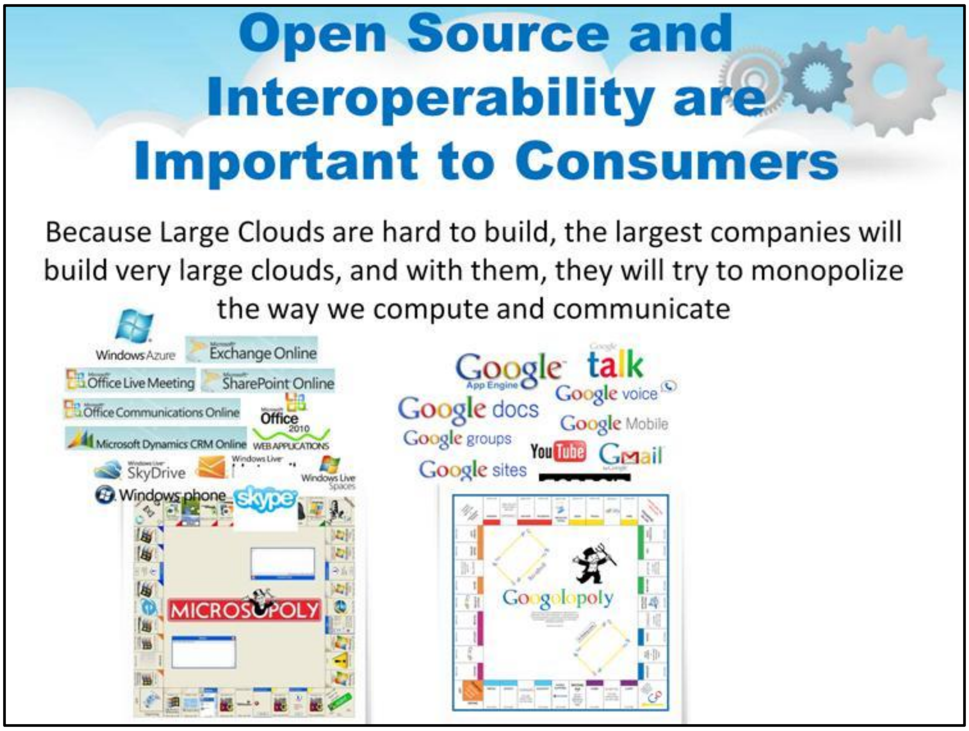
Cloud providers pay strong attention to all aspects of their platform to create a trusted environment for user services and data

Trust management include both technical mechanisms and solutions for establishing and managing trust relations between provider and customer or user administrative and security domains

Technical trust is rooted in the trusted platform and typically require manual setup

Cloud will stimulate further development of the dynamic trust establishment and management mechanisms to adopt to the dynamic and on-demand character of cloud services

Trusted Computing Platform (TCP) Architecture by Trusted Computing Group (TCG) provides and technical based for building distributed trusted environment

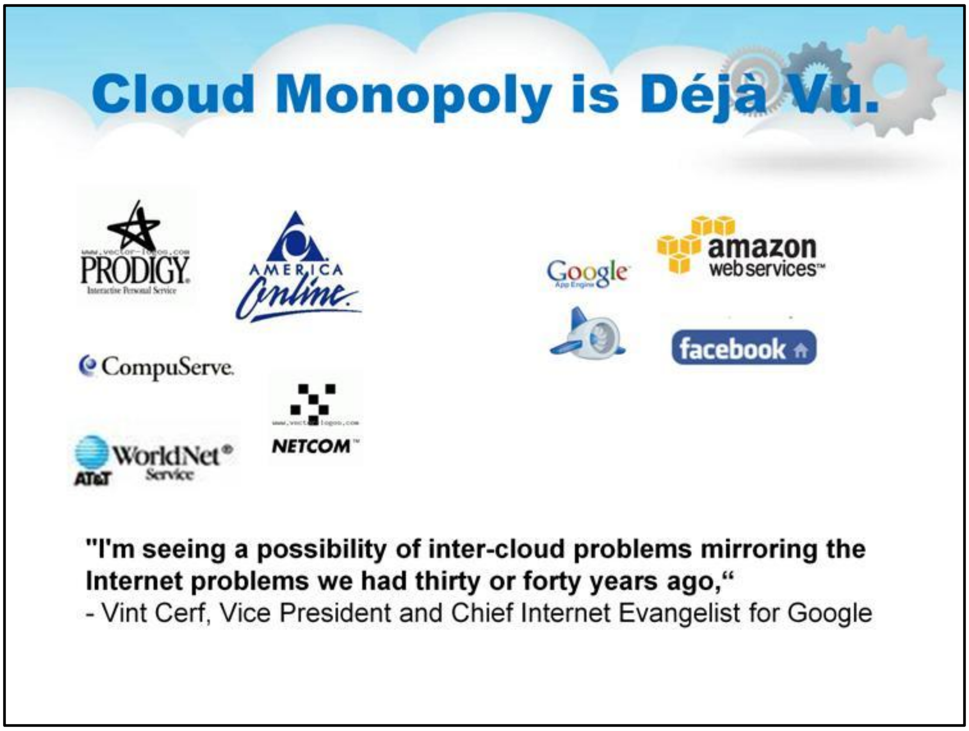


Open Source and Interoperability are Important to Consumers

Because Large Clouds are hard to build, the largest companies will build very large clouds, and with them, they will try to monopolize the way we compute and communicate

Open Source and Interoperability are Important to Consumers

Because Large Clouds are hard to build, the largest companies will build very large clouds, and with them, they will try to monopolize the way we compute and communicate



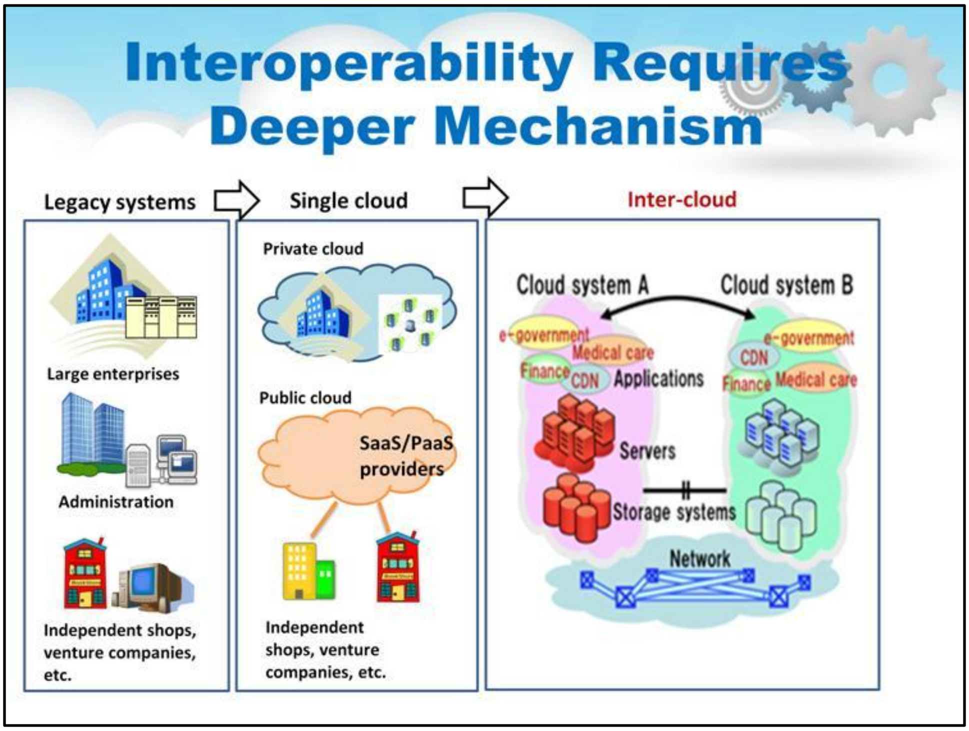
Cloud Monopoly is Déjà Vu.

PRODIGY
Interactive Personal Service
AMERICA Online
CompuServe
NETCOM
WorldNet Service
AT&T
Google App Engine
amazon web services
facebook

"I'm seeing a possibility of inter-cloud problems mirroring the Internet problems we had thirty or forty years ago,"
- Vint Cerf, Vice President and Chief Internet Evangelist for Google

(? Not sure if I interpreted the original text correctly)

In times before the Internet, large companies were dominating. Once open source and standards took over, the Internet started growing rapidly. Something similar will happen with Cloud technologies.



Interoperability Requires Deeper Mechanism

Legacy systems

Large enterprises

Administration

Independent shops, venture companies, etc.

Single cloud

Private cloud

Public cloud

SaaS/PaaS providers

Independent shops, venture companies, etc.

Inter-cloud

Cloud system A

Cloud system B

e-government

Medical care

Finance

CDN

Applications

Servers

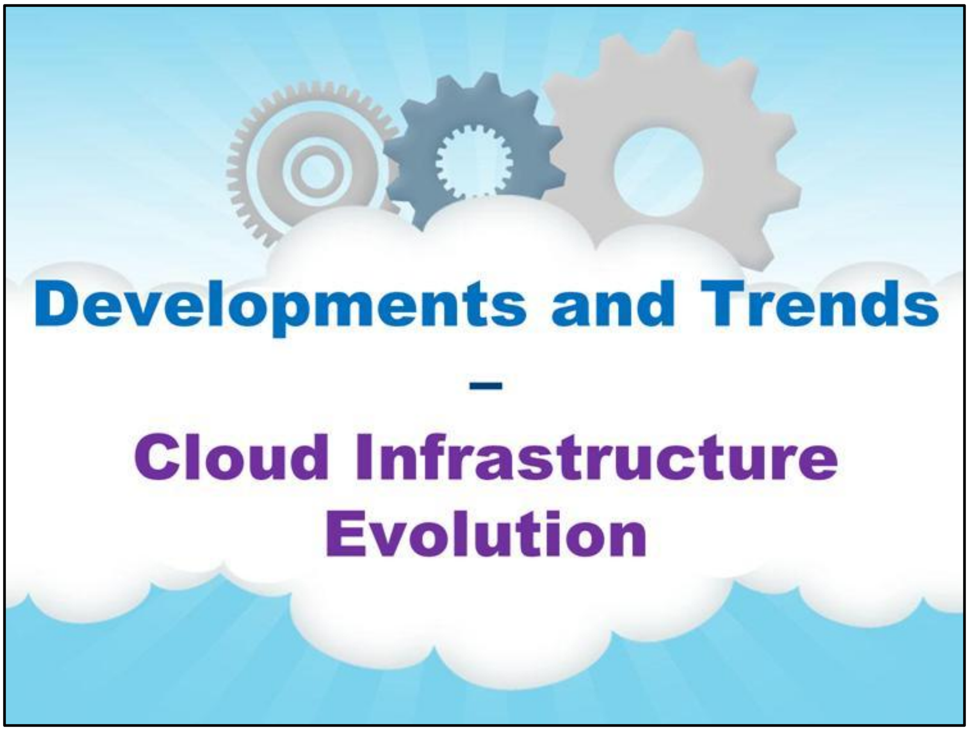
Storage systems

Network

This slide contains an illustration which shows major phases of evolution in IT. One can see that applications software used to be closely linked to a set of servers and storage and was usually installed on premise, in the site of the business.
Soon, computing platforms became more flexible, with virtualization and automation, so that applications could share a generalized cluster/platform, which we now call cloud.

The illustration shows that with Intercloud, businesses can run on a multiple datacenter platform, where multiple vendors clouds cooperate and interoperate to form one large distributed platform.

This illustration is courtesy of the Japan based industry association called the Global Intercloud Technology Forum. This group did pioneering work in defining the meaning and requirements for Intercloud.

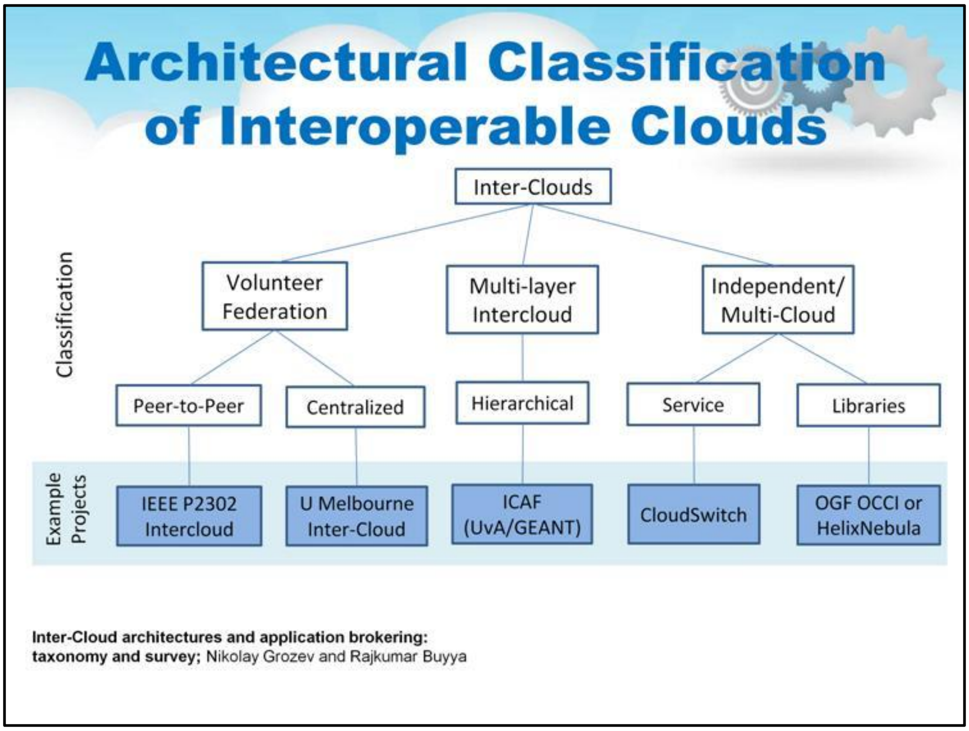


Developments and Trends

–

Cloud Infrastructure Evolution

Cloud Infrstructure Evolution



Architectural Classification of Interoperable Clouds

Inter-Clouds

Classification

Volunteer Federation

Multi-layer Intercloud

Independent/ Multi-Cloud

Peer-to-Peer

Centralized

Hierarchical

Service

Libraries

Example Projects

IEEE P2302 Intercloud

U Melbourne Inter-Cloud

ICAF (UvA/GEANT)

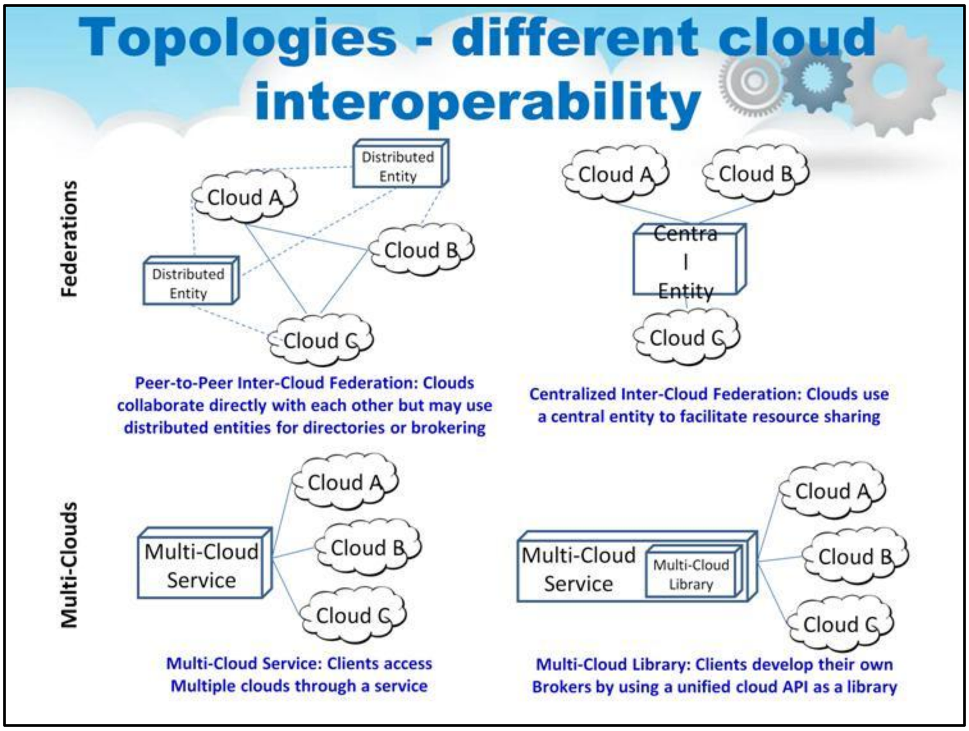
CloudSwitch

OGF OCCI or HelixNebula

Inter-Cloud architectures and application brokering: taxonomy and survey; Nikolay Grozev and Rajkumar Buyya

There are many ways clouds can interoperate. This slide uses an illustration from a research group from University of Melbourne, who polished a paper on the taxonomy of different cloud interoperability approaches. Please study the illustration.

Intercloud Architecture Framework (ICAF) is based on the multi-layer Cloud Services Model (CMS) and separates control, management, federation and operational aspects in Intercloud. Developed by UvA, currently implemented in the European project GEANT.



Topologies - different cloud interoperability

Federations

Distributed Entity

Cloud A

Cloud B

Distributed Entity

Cloud C

Peer-to-Peer Inter-Cloud Federation: Clouds collaborate directly with each other but may use distributed entities for directories or brokering

Cloud A

Cloud B

Centra l Entity

Cloud C

Centralized Inter-Cloud Federation: Clouds use a central entity to facilitate resource sharing

Multi-Clouds

Multi-Cloud Service

Cloud A

Cloud B

Cloud C

Multi-Cloud Service: Clients access Multiple clouds through a service

Multi-Cloud Service

Multi-Cloud Library

Cloud A

Cloud B

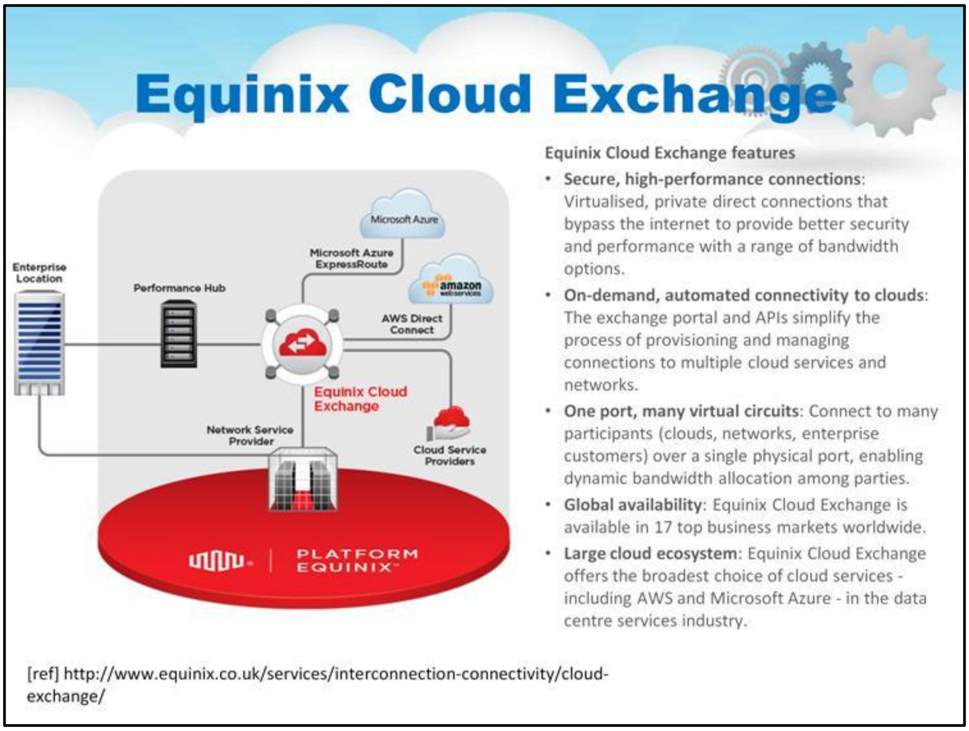
Cloud C

Multi-Cloud Library: Clients develop their own Brokers by using a unified cloud API as a library

It is easier to understand these different cloud interoperability approaches when one visualizes the resultant topologies. The illustration in this slide shows the same four categories from the previous slide but from a topology view.

It is easy to see that the multi-cloud approach will work for a user wanting to access a small number of clouds more or less manually and must "string together" the resources from each cloud themselves. This approach is ideal for scientists who want to access simple cloud resources (like VMs) to create large clusters for Hadoop or for Grid calculations. The responsibility for understanding the variations amongst the clouds falls with the user in this case.

It is easy to see that the Federations approach is a much more scalable and interconnected architecture. This approach is ideal for systems which might span multiple cloud providers, or span private and public clouds. It also places the responsibility of understanding the variations amongst the clouds with the infrastructure, not the end user. This is an important point which we will examine more later.



Equinix Cloud Exchange

Equinix Cloud Exchange features

- **Secure, high-performance connections**: Virtualised, private direct connections that bypass the internet to provide better security and performance with a range of bandwidth options.
- **On-demand, automated connectivity to clouds**: The exchange portal and APIs simplify the process of provisioning and managing connections to multiple cloud services and networks.
- **One port, many virtual circuits**: Connect to many participants (clouds, networks, enterprise customers) over a single physical port, enabling dynamic bandwidth allocation among parties.
- **Global availability**: Equinix Cloud Exchange is available in 17 top business markets worldwide.
- **Large cloud ecosystem**: Equinix Cloud Exchange offers the broadest choice of cloud services - including AWS and Microsoft Azure - in the data centre services industry.

[ref] http://www.equinix.co.uk/services/interconnection-connectivity/cloud-exchange/

Global cloud infrastructure continues to grow and creates its own global cloud ecosystem that includes global cloud provider infrastructures and exchange points similar to current Internet infrastructure

Operational since 2014 and has a strong growth

Equinix Cloud Exchange is an advanced interconnection solution that provides seamless, on-demand, and private connections to many clouds and many networks in major metro areas around the world. It makes the process of connecting to clouds as fast and flexible as the cloud itself. That’s why Equinix Cloud Exchange is the data centre industry’s most secure, scalable, and broadly adopted cloud connectivity solution.

Equinix Cloud Exchange features

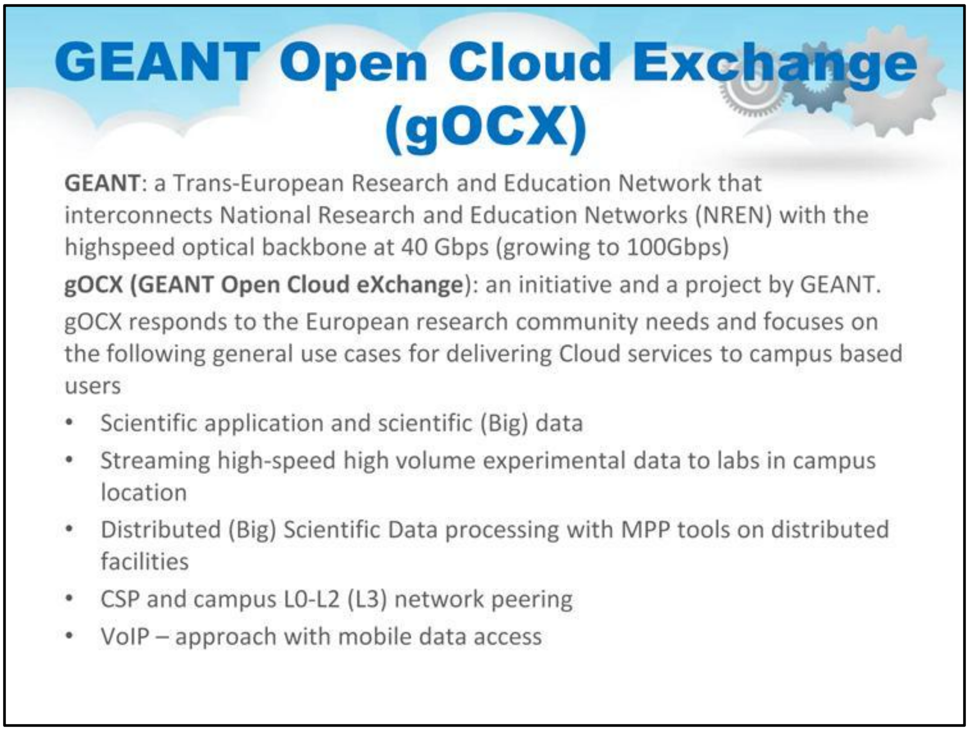
Secure, high-performance connections: Virtualised, private direct connections that bypass the internet to provide better security and performance with a range of bandwidth options.

On-demand, automated connectivity to clouds: The exchange portal and APIs simplify the process of provisioning and managing connections to multiple cloud services and networks.

One port, many virtual circuits: Connect to many participants (clouds, networks, enterprise customers) over a single physical port, enabling dynamic bandwidth allocation among parties.

Global availability: Equinix Cloud Exchange is available in 17 top business markets worldwide.

Large cloud ecosystem: Equinix Cloud Exchange offers the broadest choice of cloud services - including AWS and Microsoft Azure - in the data centre services industry.



GEANT Open Cloud Exchange (gOCX)

GEANT: a Trans-European Research and Education Network that interconnects National Research and Education Networks (NREN) with the highspeed optical backbone at 40 Gbps (growing to 100Gbps)

gOCX (GEANT Open Cloud eXchange): an initiative and a project by GEANT.

gOCX responds to the European research community needs and focuses on the following general use cases for delivering Cloud services to campus based users

- Scientific application and scientific (Big) data
- Streaming high-speed high volume experimental data to labs in campus location
- Distributed (Big) Scientific Data processing with MPP tools on distributed facilities
- CSP and campus L0-L2 (L3) network peering
- VoIP – approach with mobile data access

GEANT Open Cloud Exchange (gOCX) is an Interoperability project for Science. GEANT is the Trans-European highspeed network interconnecting European universities and research organisations (via National Research and Education Networks (NREN)) at current speed of 10 Gbps growing to 40 and 100 Gbps in the next few years.

gOCX responds to the European research community needs and focuses on the following general use cases for delivering Cloud services to campus based users:

Scientific application and scientific (Big) data

- LHC/HEP, genomics, astronomy, climate, video, etc. (+long tail science)

Streaming high-speed high volume experimental data to labs in campus location

- Direct links through campus network

Distributed (Big) Scientific Data processing with MPP tools on distributed facilities

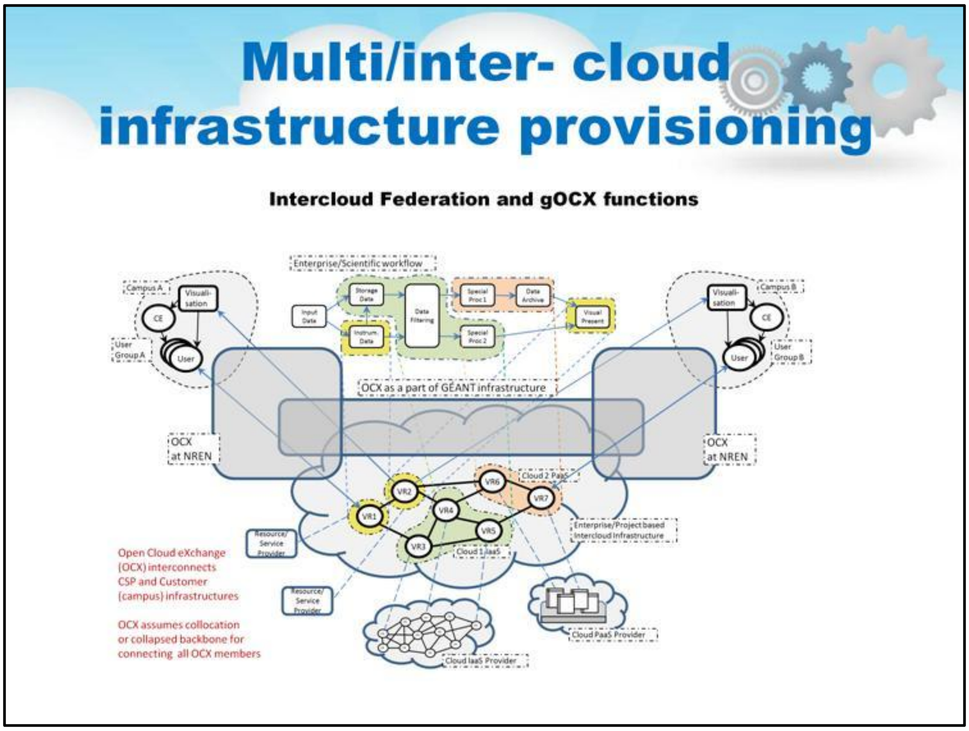
- Data distributed between few locations next to local datacenters

CSP and campus L0-L2 (L3) network peering

- Dark fiber with termination as campus network or as CSP network

VoIP – approach with mobile data access

- Support mobile access network (LTE) and tunnel access to campus network



Multi/inter- cloud infrastructure provisioning

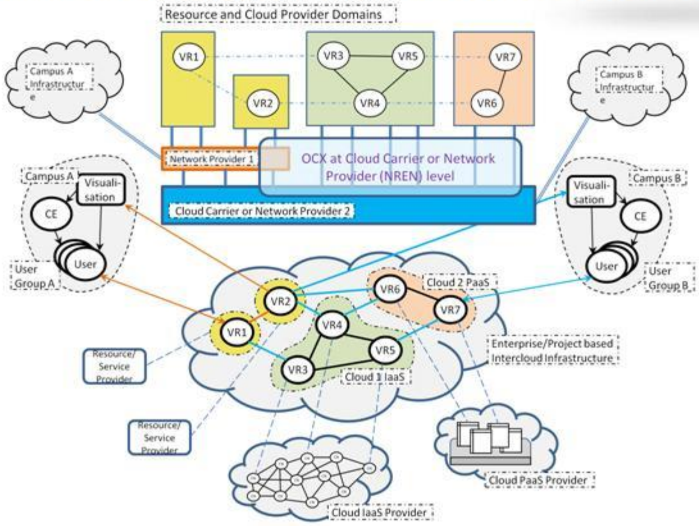
Intercloud Federation and gOCX functions

Open Cloud eXchange (OCX) interconnects CSP and Customer (campus) infrastructures

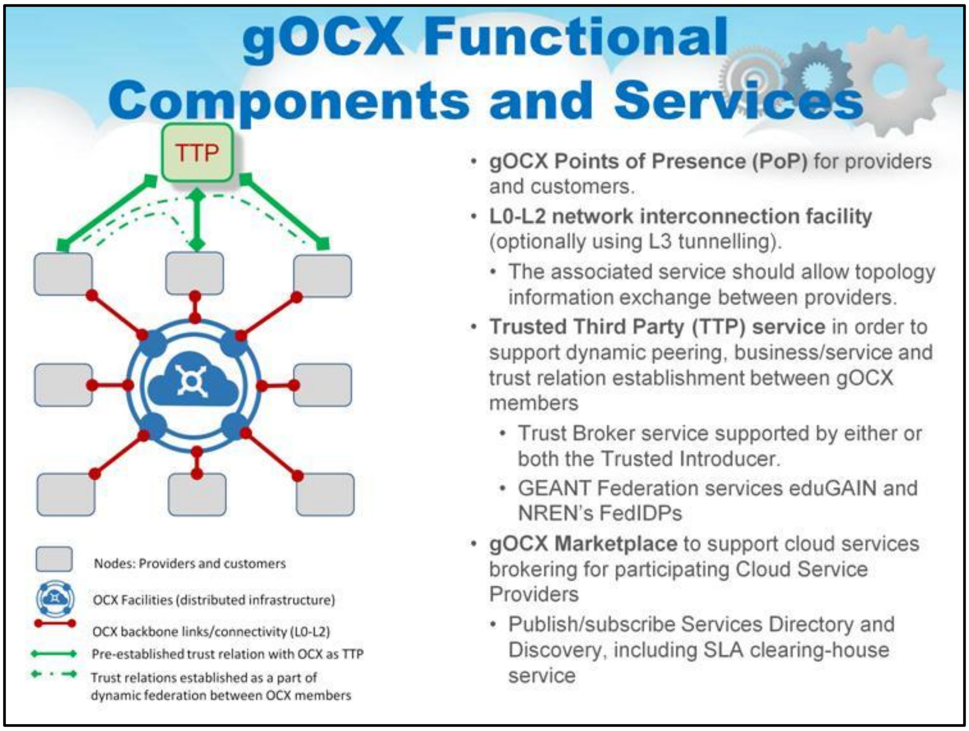
OCX assumes collocation or collapsed backbone for connecting all OCX members

OCX can fill a needed bridge between users and Cloud Service Providers.

gOCX as a Component of Intercloud Infrastructure


Resource and Cloud Provider Domains
VR1
VR2
VR3
VR4
VR5
VR6
VR7
Campus A Infrastructure
Campus B Infrastructure
Network Provider 1
OCX at Cloud Carrier or Network Provider (NREN) level
Cloud Carrier or Network Provider 2
Campus A
Visualisation
CE
User
User Group A
Campus B
User Group B
Cloud 2 PaaS
Cloud 1 IaaS
Enterprise/Project based Intercloud Infrastructure
Resource/Service Provider
Resource/Service Provider
Cloud PaaS Provider
Cloud IaaS Provider

The slide illustrates the gOCX system



gOCX Functional Components and Services

- **gOCX Points of Presence (PoP)** for providers and customers.
- **L0-L2 network interconnection facility** (optionally using L3 tunnelling).
  - The associated service should allow topology information exchange between providers.
- **Trusted Third Party (TTP) service** in order to support dynamic peering, business/service and trust relation establishment between gOCX members
  - Trust Broker service supported by either or both the Trusted Introducer.
  - GEANT Federation services eduGAIN and NREN's FedIDPs
- **gOCX Marketplace** to support cloud services brokering for participating Cloud Service Providers
  - Publish/subscribe Services Directory and Discovery, including SLA clearing-house service

□	Nodes: Providers and customers
◎	OCX Facilities (distributed infrastructure)
●—●	OCX backbone links/connectivity (L0-L2)
⟷	Pre-established trust relation with OCX as TTP
←·→	Trust relations established as a part of dynamic federation between OCX members

Architecturally and functionally the gOCX includes the following services and functional components

Architecturally and functionally the gOCX includes the following services and functional components

gOCX Points of Presence (PoP) for providers and customers.

L0-L2 network interconnection facility (optionally also connectivity using dedicated optical links and L3 tunneling).

Trusted Third Party (TTP) service in order to support dynamic
peering, business/service and trust relation establishment between gOCX members

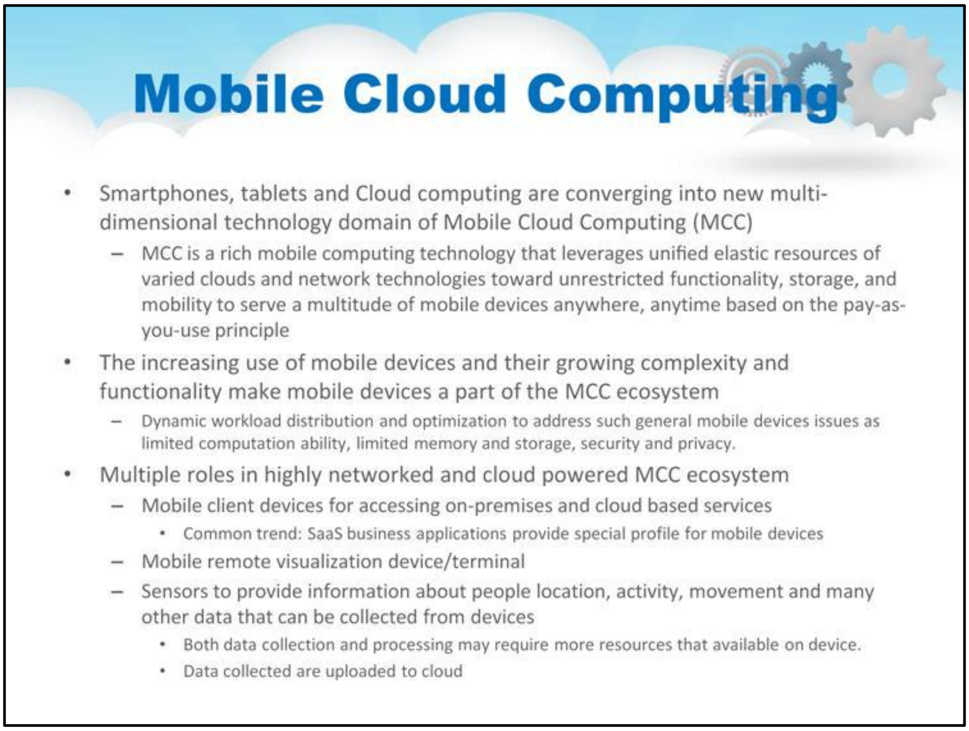
gOCX Marketplace to support cloud services brokering for participating Cloud Service Providers

Developments and Trends

–

Mobile Cloud Computing

Mobile Cloud Computing



Mobile Cloud Computing

- Smartphones, tablets and Cloud computing are converging into new multi-dimensional technology domain of Mobile Cloud Computing (MCC)
  - MCC is a rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime based on the pay-as-you-use principle
- The increasing use of mobile devices and their growing complexity and functionality make mobile devices a part of the MCC ecosystem
  - Dynamic workload distribution and optimization to address such general mobile devices issues as limited computation ability, limited memory and storage, security and privacy.
- Multiple roles in highly networked and cloud powered MCC ecosystem
  - Mobile client devices for accessing on-premises and cloud based services
    - Common trend: SaaS business applications provide special profile for mobile devices
  - Mobile remote visualization device/terminal
  - Sensors to provide information about people location, activity, movement and many other data that can be collected from devices
    - Both data collection and processing may require more resources that available on device.
    - Data collected are uploaded to cloud

Mobile cloud Computing (MCC) is another trend that will use and drive CC development.

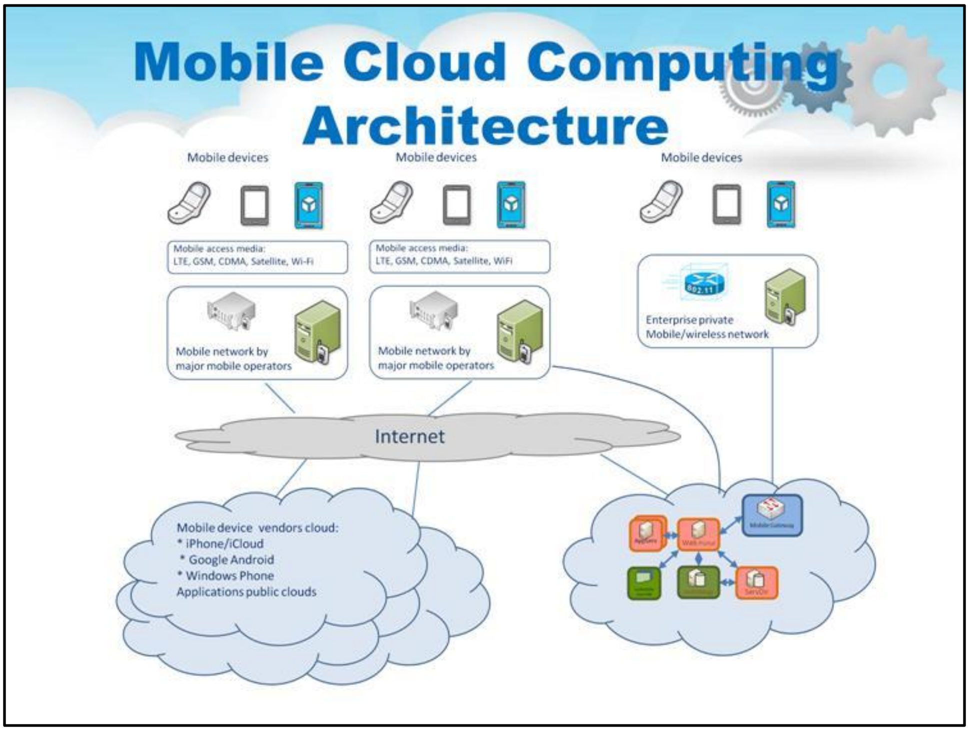
Smartphones, tablets and Cloud computing are converging into new multidimensional technology domain of Mobile Cloud Computing (MCC). MCC can be defined as "a rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms based on the pay-as-you-use principle“.

The increasing use of mobile devices and their growing complexity and functionality make mobile devices not just mobile clients or terminal to provide access to remote resources and computation power but also part of the MCC ecosystem that dynamically distribute and optimize workload to address such general mobile devices issues as limited computation ability, limited memory and storage, security and privacy.

Mobile devices may play multiple roles in our highly networked and computer/cloud powered world, in particular,

* Smartphones and tablets are increasingly used as mobile client devices for accessing on-premises and cloud based services with increasing functionality of the graphical user interface (GUI). Cloud based SaaS applications provide special profile for mobile devices.

* Smartphones and tablets can be used as mobile remote visualization



Mobile Cloud Computing Architecture

Mobile devices
Mobile devices
Mobile devices
Mobile access media:
LTE, GSM, CDMA, Satellite, Wi-Fi
Mobile access media:
LTE, GSM, CDMA, Satellite, WiFi
Mobile network by major mobile operators
Mobile network by major mobile operators
802.11
Enterprise private Mobile/wireless network
Internet
Mobile device vendors cloud:
* iPhone/iCloud
* Google Android
* Windows Phone
Applications public clouds
AppServ
Web Portal
Mobile Gateway
ServDir

Slide show general MCC architecture including mobile operator network, mobile device/OS vendor cloud, public cloud and enterprise mobile cloud gateway

illustrates a general MCC architecture that includes mobile operator network, mobile device/OS vendor cloud, public cloud and enterprise private cloud that may contain a special gateway to mobile operators' networks to allow optimized access for mobile devices. Important to mention that large mobile network operators as well as mobile device or mobile OS vendors are creating own clouds that are specifically targeted to support mobile applications and services. You can find many such smart applications from your provider or from the device vendor the obviously use cloud processing of computation intensive tasks such as building route in navigation applications (these kind of applications doesn't work without connection to data network and consequently to remote server).

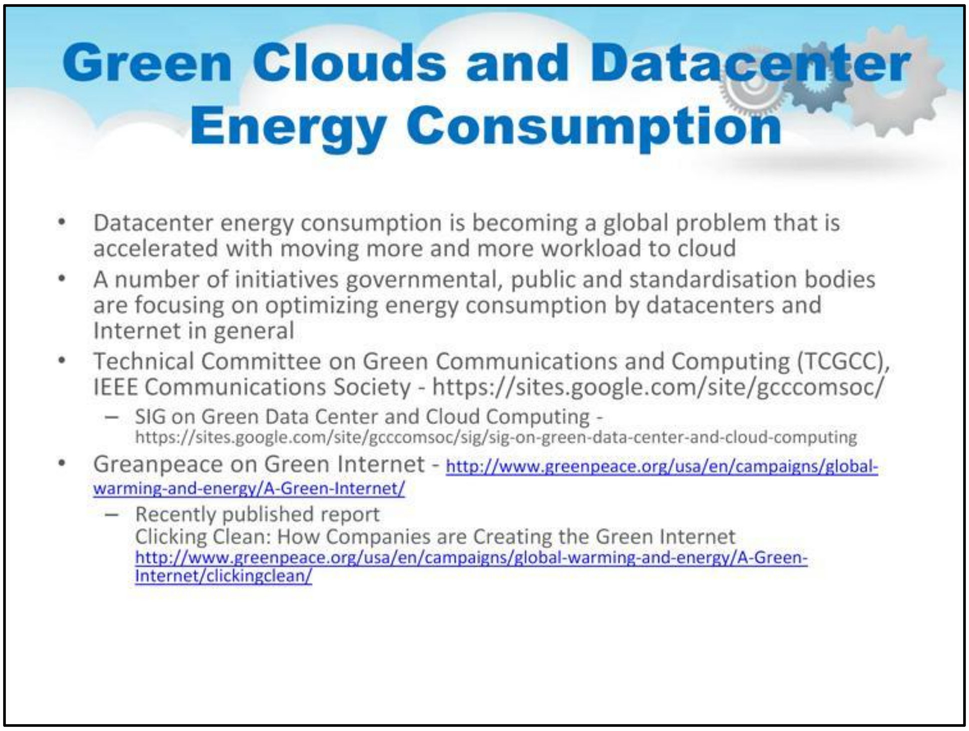
Supporting mobile applications with backend clouds requires distributed and global cloud infrastructure. Tracking the device movement with some continuous processes will require the application or VM migration between datacenters. Such migration needs to be context aware and location policies rising a number of technical and research questions.

Developments and Trends

–

Energy Consumption and Green Clouds

Energy Consumption and Green Clouds



Green Clouds and Datacenter Energy Consumption

- Datacenter energy consumption is becoming a global problem that is accelerated with moving more and more workload to cloud
- A number of initiatives governmental, public and standardisation bodies are focusing on optimizing energy consumption by datacenters and Internet in general
- Technical Committee on Green Communications and Computing (TCGCC), IEEE Communications Society - https://sites.google.com/site/gcccomsoc/
  - SIG on Green Data Center and Cloud Computing - https://sites.google.com/site/gcccomsoc/sig/sig-on-green-data-center-and-cloud-computing
- Greanpeace on Green Internet - http://www.greenpeace.org/usa/en/campaigns/global-warming-and-energy/A-Green-Internet/
  - Recently published report
    Clicking Clean: How Companies are Creating the Green Internet
    http://www.greenpeace.org/usa/en/campaigns/global-warming-and-energy/A-Green-Internet/clickingclean/

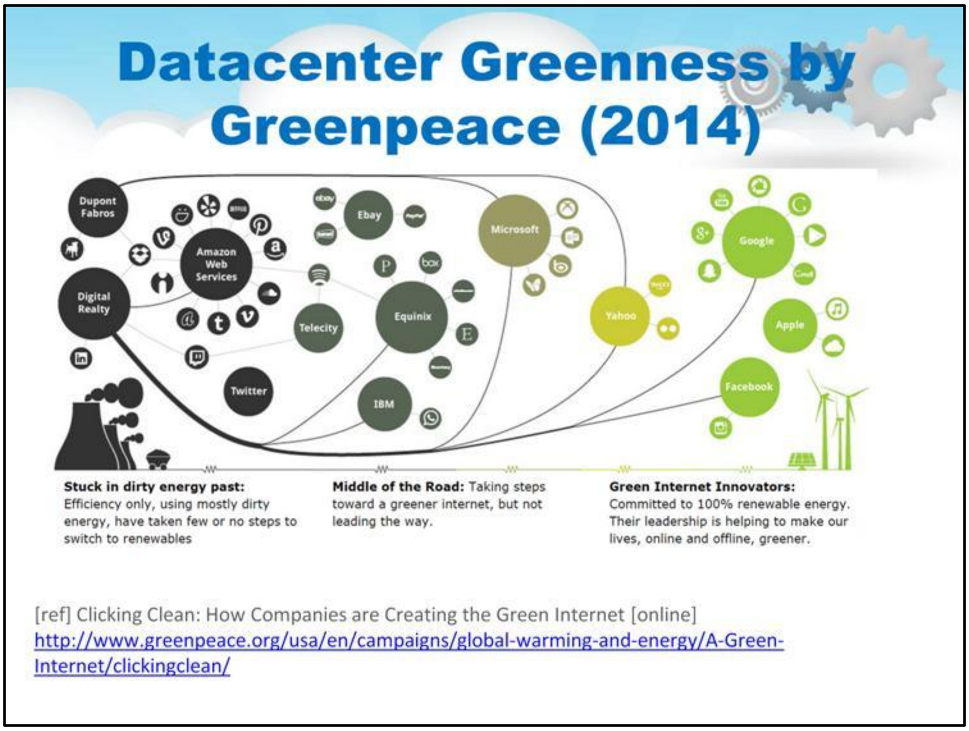
Green Clouds and Datacenter Energy Consumption

Datacenter energy consumption is becoming a global problem that is accelerated with moving more and more workload to cloud

A number of initiatives governmental, public and standardisation bodies are focusing on optimizing energy consumption by datacenters and Internet in general

Technical Committee on Green Communications and Computing (TCGCC), IEEE Communications Society

Greanpeace on Green Internet



Datacenter Greenness by Greenpeace (2014)

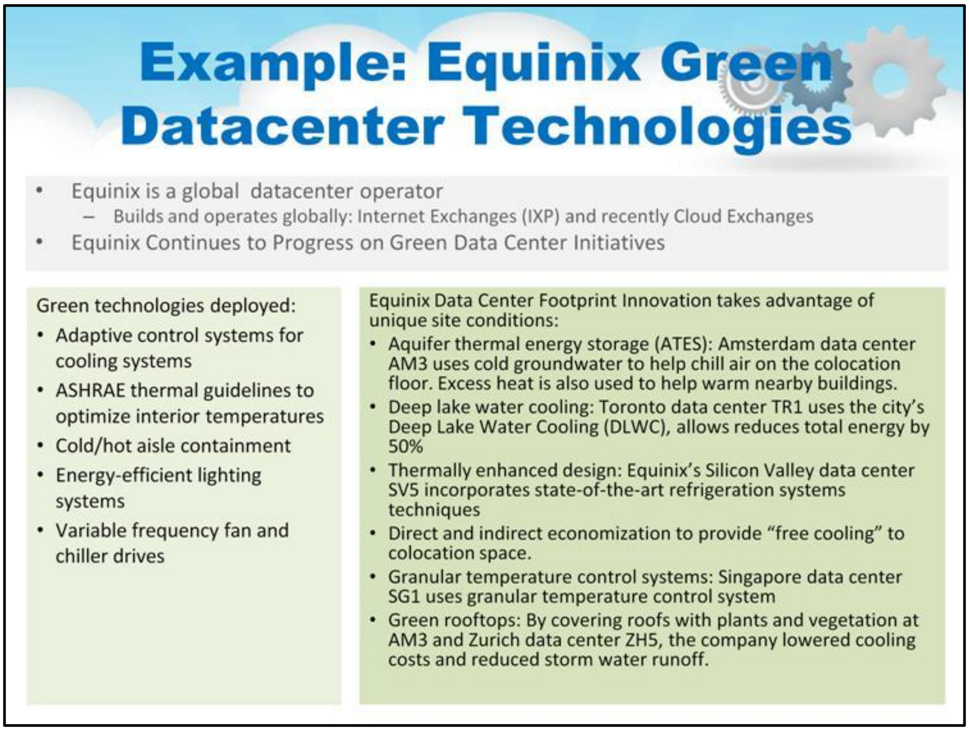
Stuck in dirty energy past: Efficiency only, using mostly dirty energy, have taken few or no steps to switch to renewables

Middle of the Road: Taking steps toward a greener internet, but not leading the way.

Green Internet Innovators: Committed to 100% renewable energy. Their leadership is helping to make our lives, online and offline, greener.

[ref] Clicking Clean: How Companies are Creating the Green Internet [online] http://www.greenpeace.org/usa/en/campaigns/global-warming-and-energy/A-Green-Internet/clickingclean/

The slide contains an illustration on Datacenter Greenness by Greenpeace (2014)



Example: Equinix Green Datacenter Technologies

- Equinix is a global datacenter operator
  - Builds and operates globally: Internet Exchanges (IXP) and recently Cloud Exchanges
- Equinix Continues to Progress on Green Data Center Initiatives

Green technologies deployed:

- Adaptive control systems for cooling systems
- ASHRAE thermal guidelines to optimize interior temperatures
- Cold/hot aisle containment
- Energy-efficient lighting systems
- Variable frequency fan and chiller drives

Equinix Data Center Footprint Innovation takes advantage of unique site conditions:

- Aquifer thermal energy storage (ATES): Amsterdam data center AM3 uses cold groundwater to help chill air on the colocation floor. Excess heat is also used to help warm nearby buildings.
- Deep lake water cooling: Toronto data center TR1 uses the city's Deep Lake Water Cooling (DLWC), allows reduces total energy by 50%
- Thermally enhanced design: Equinix's Silicon Valley data center SV5 incorporates state-of-the-art refrigeration systems techniques
- Direct and indirect economization to provide "free cooling" to colocation space.
- Granular temperature control systems: Singapore data center SG1 uses granular temperature control system
- Green rooftops: By covering roofs with plants and vegetation at AM3 and Zurich data center ZH5, the company lowered cooling costs and reduced storm water runoff.

Green technologies deployed by Equinix globally include:

Adaptive control systems that reduce power consumption and increase cooling capacity through active airflow management using intelligent, distributed sensors and innovative control policies.

ASHRAE thermal guidelines are used as reference in our newest facilities to optimize interior temperatures. This reduces power consumption for cooling, while maintaining a safe operating temperature for computing equipment.

Cold/hot aisle containment uses physical barriers to reduce the mixing of cold air in data center supply aisles with the hot air in their exhaust aisles. This results in lower energy consumption and more efficient cooling.

Energy-efficient lighting systems in our data centers use motion-activated controls to reduce energy consumption and ambient heat from operating lights.

Variable frequency drives are deployed in chillers, pumps and fans in our HVAC systems to save energy by automatically reducing a motor's speed and power draw to match lower system loads.

Data Center Footprint Innovation

When Equinix designs and builds new data centers, it reduces energy by taking advantage of unique site conditions:

Deep lake water cooling: Equinix's Toronto data center uses the city's Deep Lake Water Cooling (DLWC). This novel approach reduces total energy needs by 50 percent or more.

Summary and Takeaway

- Cloud is a fast developing area dating its first appearance in 2008 and currently (since 2015) in the maturity state
- Cloud computing drives many technologies transformation and provides a basis for new emerging technologies such as Big Data
  - And Big Data itself drives cloud development to respond to volume and velocity of data processing
- Shift in cloud service model from IaaS to SaaS indicates that there are growing opportunities for new business developments
  - Cloud can take over all routing functions of infrastructure management and allows innovators to focus on the key and smart ideas
- Mobile Cloud Computing that utilizes growing amount of personal handheld smart devices is utilizing global cloud infrastructure and will influence both infrastructure and applications development
- Cloud industry is working hard to decrease cloud datacenter consumption by implementing advanced “green” technologies
- There is a great opportunity for building a career in Cloud Computing and Big Data and this set of tutorials intends to provide a sufficient knowledge body from basic to advanced

In this lecture we covered:

- Cloud is a fast developing area dating its first appearance in 2008 and currently (since 2015) in the maturity state
- Cloud computing drives many technologies transformation and provides a basis for new emerging technologies such as Big Data
- And Big Data itself drives cloud development to respond to volume and velocity of data processing
- Shift in cloud service model from IaaS to SaaS indicates that there are growing opportunities for new business developments
- Cloud can take over all routing functions of infrastructure management and allows innovators to focus on the key and smart ideas
- Mobile Cloud Computing that utilizes growing amount of personal handheld smart devices is utilizing global cloud infrastructure and will influence both infrastructure and applications development
- Cloud industry is working hard to decrease cloud datacenter consumption by implementing advanced “green” technologies
- There is a great opportunity for building a career in Cloud Computing and Big Data and this set of materials intends to provide a sufficient knowledge body from basic to advanced level.

Cloud Computing

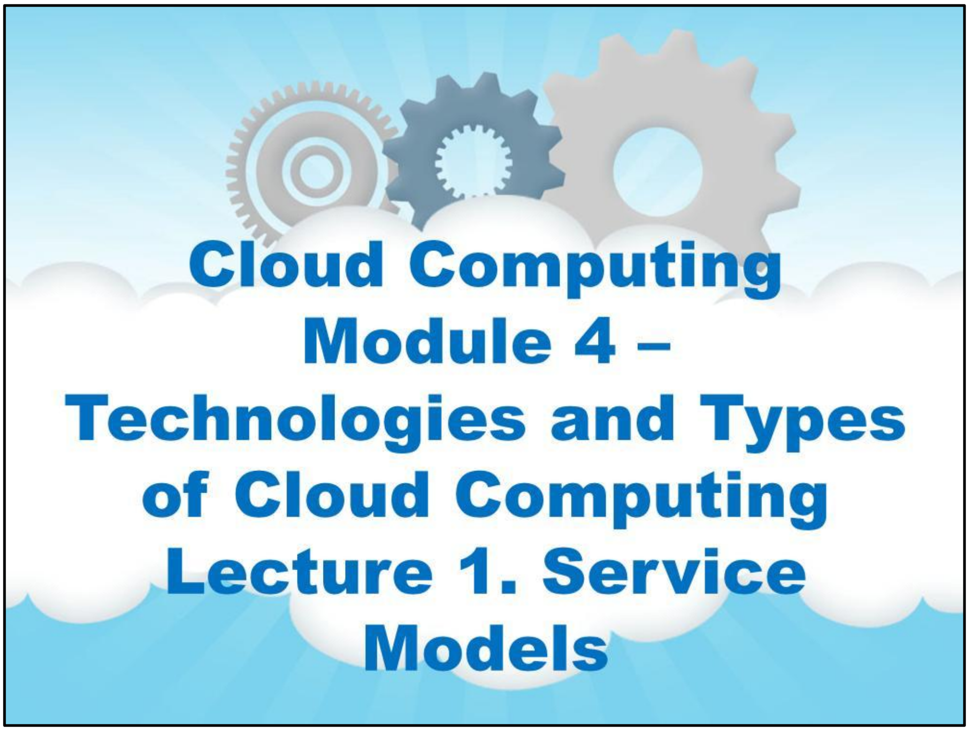
Lecture Manual

Volume 4

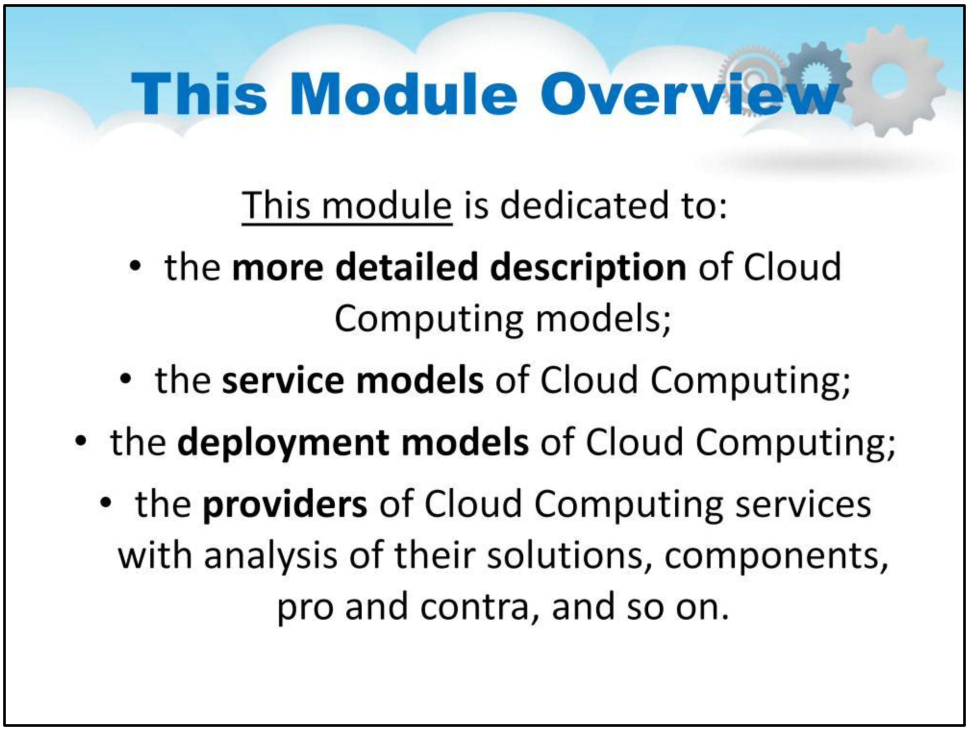
Module 4

Technologies and Types of Cloud Computing

Content



Cloud Computing Module 4 – Technologies and Types of Cloud Computing Lecture 1. Service Models

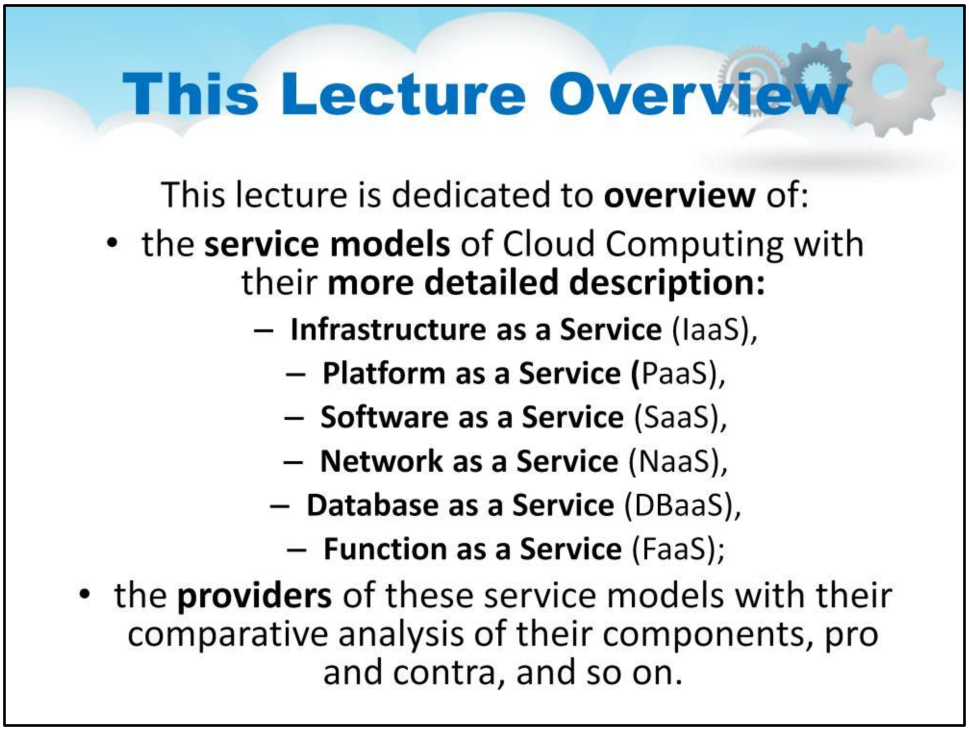


This Module Overview

This module is dedicated to:

- the **more detailed description** of Cloud Computing models;
- the **service models** of Cloud Computing;
- the **deployment models** of Cloud Computing;
- the **providers** of Cloud Computing services with analysis of their solutions, components, pro and contra, and so on.

Module 4. Technologies and Types of Cloud Computing



This Lecture Overview

This lecture is dedicated to **overview** of:

- the **service models** of Cloud Computing with their **more detailed description:**
  - **Infrastructure as a Service** (IaaS),
  - **Platform as a Service** (PaaS),
  - **Software as a Service** (SaaS),
  - **Network as a Service** (NaaS),
  - **Database as a Service** (DBaaS),
  - **Function as a Service** (FaaS);
- the **providers** of these service models with their comparative analysis of their components, pro and contra, and so on.

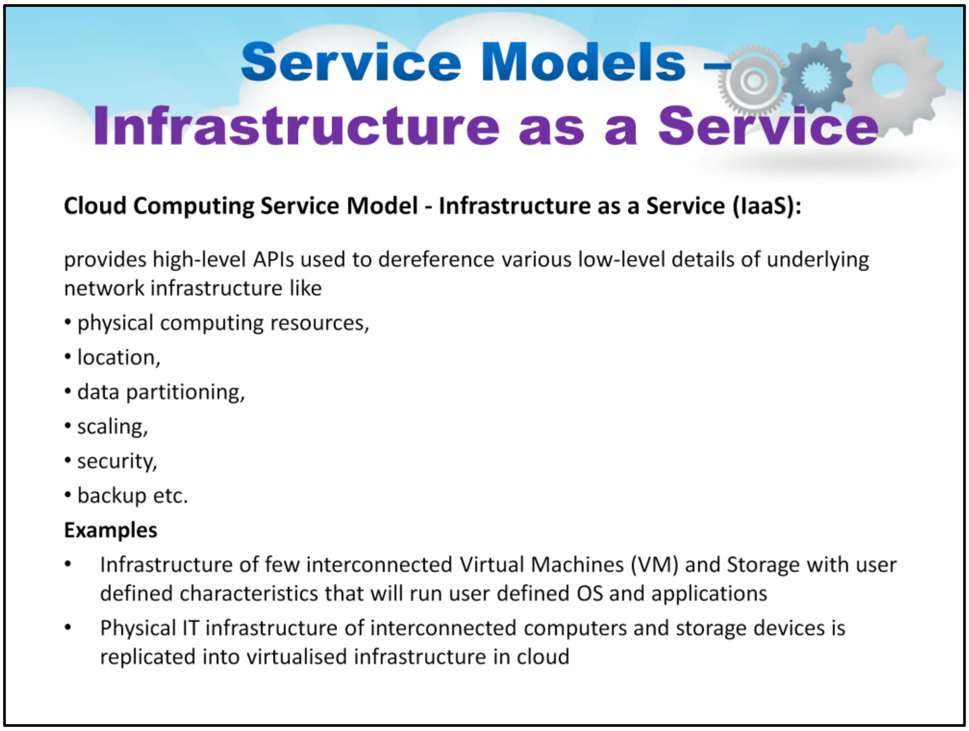
Lecture 1. Service Models

Cloud Service Models

–

Overview

Overview



Service Models – Infrastructure as a Service

Cloud Computing Service Model - Infrastructure as a Service (IaaS):

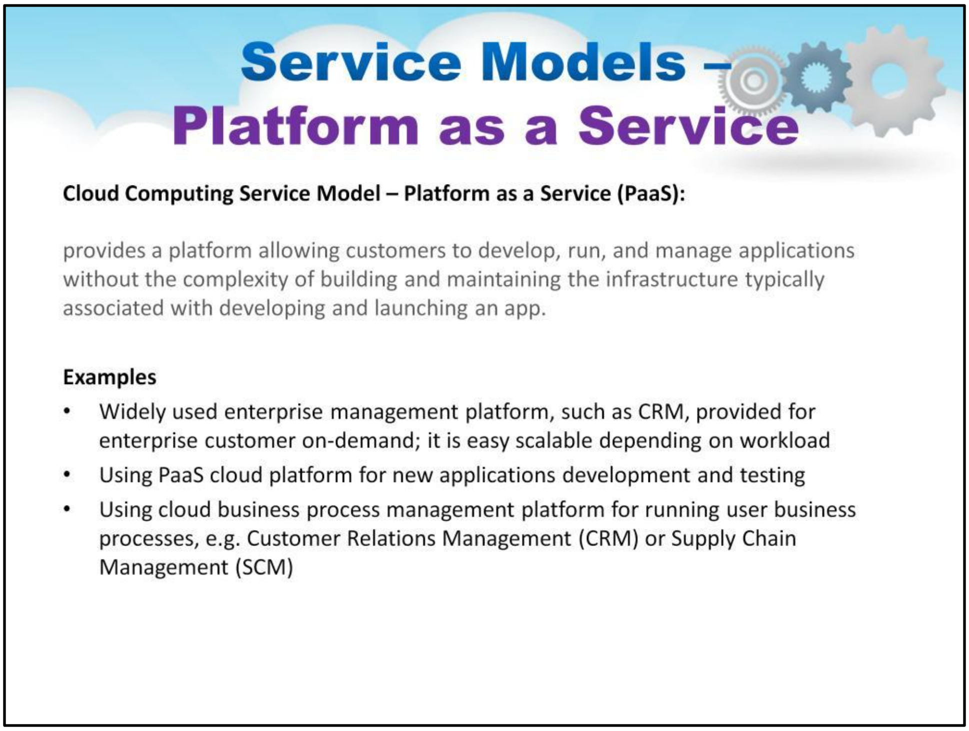
provides high-level APIs used to dereference various low-level details of underlying network infrastructure like

- physical computing resources,
- location,
- data partitioning,
- scaling,
- security,
- backup etc.

Examples

- Infrastructure of few interconnected Virtual Machines (VM) and Storage with user defined characteristics that will run user defined OS and applications
- Physical IT infrastructure of interconnected computers and storage devices is replicated into virtualised infrastructure in cloud

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).



Service Models – Platform as a Service

Cloud Computing Service Model – Platform as a Service (PaaS):

provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

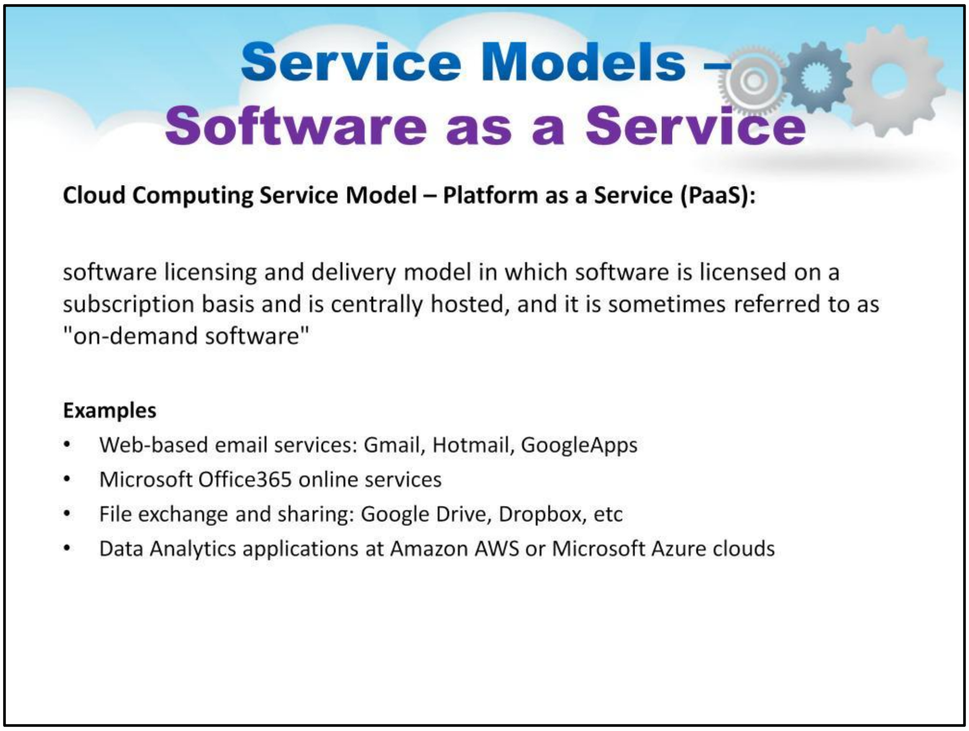
Examples

- Widely used enterprise management platform, such as CRM, provided for enterprise customer on-demand; it is easy scalable depending on workload
- Using PaaS cloud platform for new applications development and testing
- Using cloud business process management platform for running user business processes, e.g. Customer Relations Management (CRM) or Supply Chain Management (SCM)

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using *programming languages, libraries, services, and tools supported by the provider*. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has *control over the deployed applications and possibly configuration settings for the application-hosting environment*.

PaaS can be delivered in three ways:

- as a public cloud service from a provider, where the consumer controls software deployment with minimal configuration options, and the provider provides the networks, servers, storage, operating system (OS), middleware (e.g. Java runtime, .NET runtime, integration, etc.), database and other services to host the consumer's application.
- as a private service (software or appliance) inside the firewall.
- as software deployed on a public infrastructure as a service.



Service Models – Software as a Service

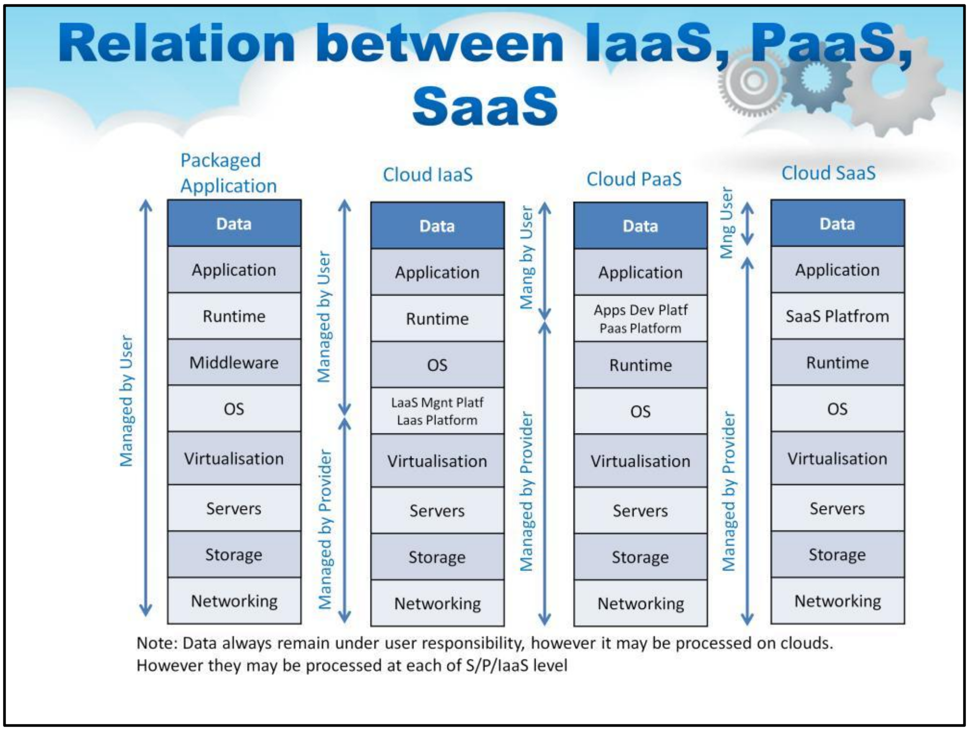
Cloud Computing Service Model – Platform as a Service (PaaS):

software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted, and it is sometimes referred to as "on-demand software"

Examples

- Web-based email services: Gmail, Hotmail, GoogleApps
- Microsoft Office365 online services
- File exchange and sharing: Google Drive, Dropbox, etc
- Data Analytics applications at Amazon AWS or Microsoft Azure clouds

The capability provided to the consumer is to use the provider's applications running *on a cloud infrastructure*. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of *limited user-specific application configuration settings*.



Relation between IaaS, PaaS, SaaS

Packaged Application	Cloud IaaS	Cloud PaaS	Cloud SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Runtime	Runtime	Apps Dev Platf Paas Platform	SaaS Platfrom
Middleware	OS	Runtime	Runtime
OS	LaaS Mgnt Platf Laas Platform	OS	OS
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Note: Data always remain under user responsibility, however it may be processed on clouds.
However they may be processed at each of S/P/IaaS level

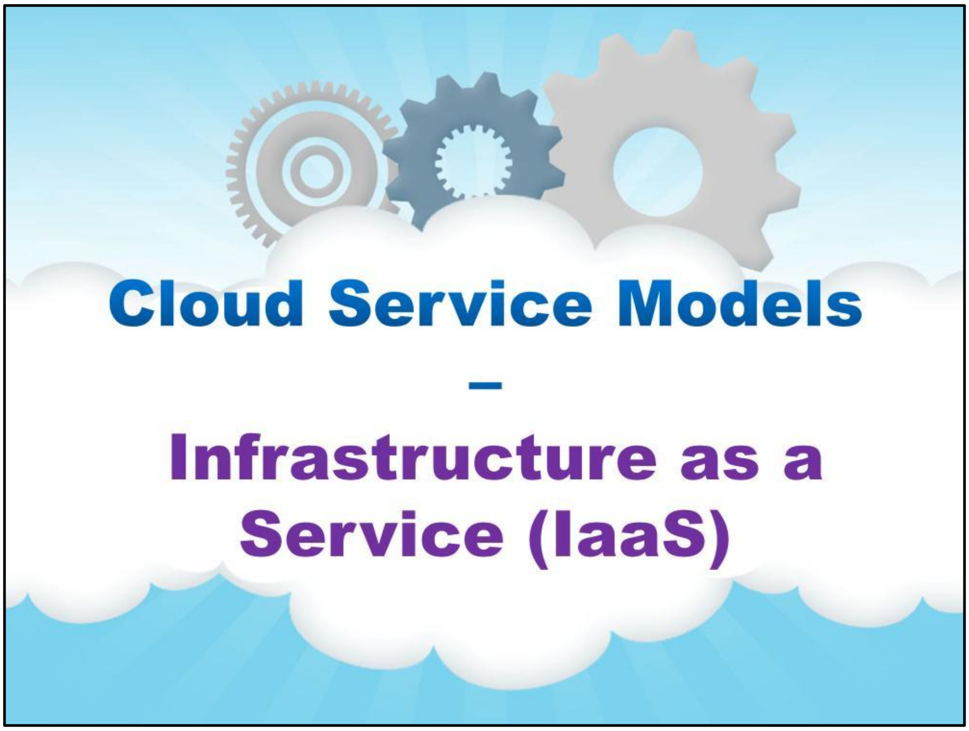
The use models for Cloud Computing are closely related.

For a traditional “physically installed” packaged application, the entire deployment stack is managed by the user, as can be seen in the first block of the illustration. The user is responsible for provisioning the complete server including networking and storage, and the Operating System software as well. Additionally, the user provisions whatever databases and middleware they need, any special runtime (Java or Ruby), and finally the application. The application data is also the responsibility (and the ownership of) the user.

The Cloud IaaS model changes this significantly, alleviating a large portion of at least what would be considered “physical” infrastructure, this is shown in the second block of the illustration. The cloud management (sometimes called Cloud OS) software is shown in this diagram as “IaaS Mngt Platf”. The user obtains the hardware in a virtualized form from the IaaS CSP and worries only about the software stack, application, and data.

The Cloud PaaS model further offers middleware, application runtime, and as much “software” infrastructure as possible, creating a convenient container for the application developer. While this requires the developer to restructure if not rewrite the application at the source code level to interface to the PaaS, this restricting will impart new scalability and availability strengths into the application.

The Cloud IaaS model is a consuming of just the application itself through a browser or web service.

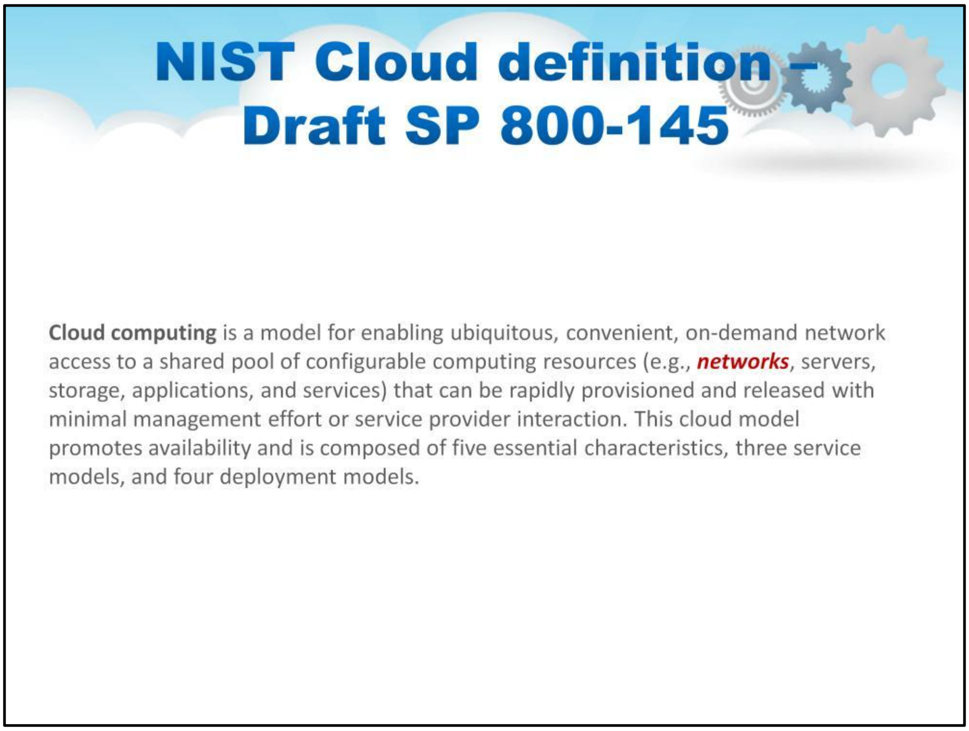


Cloud Service Models

–

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS)

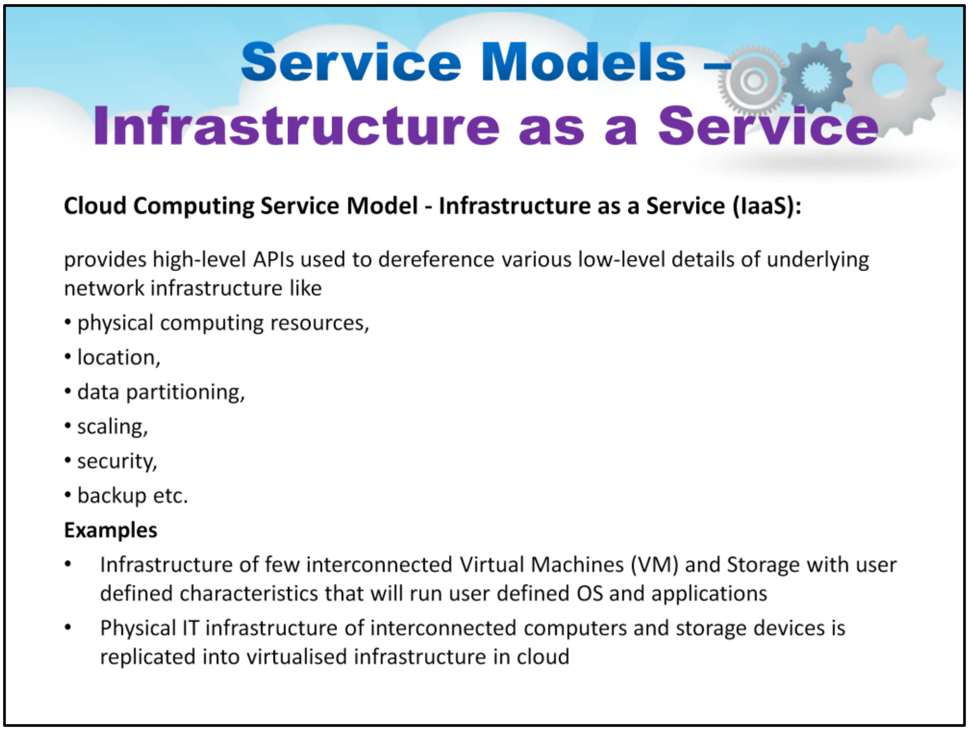


NIST Cloud definition – Draft SP 800-145

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., ***networks***, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

We will start the Lesson by reviewing the NIST Cloud Definition.

In particular, the NIST definition is very specific around Infrastructure as a Service, which we will first investigate.



Service Models – Infrastructure as a Service

Cloud Computing Service Model - Infrastructure as a Service (IaaS):

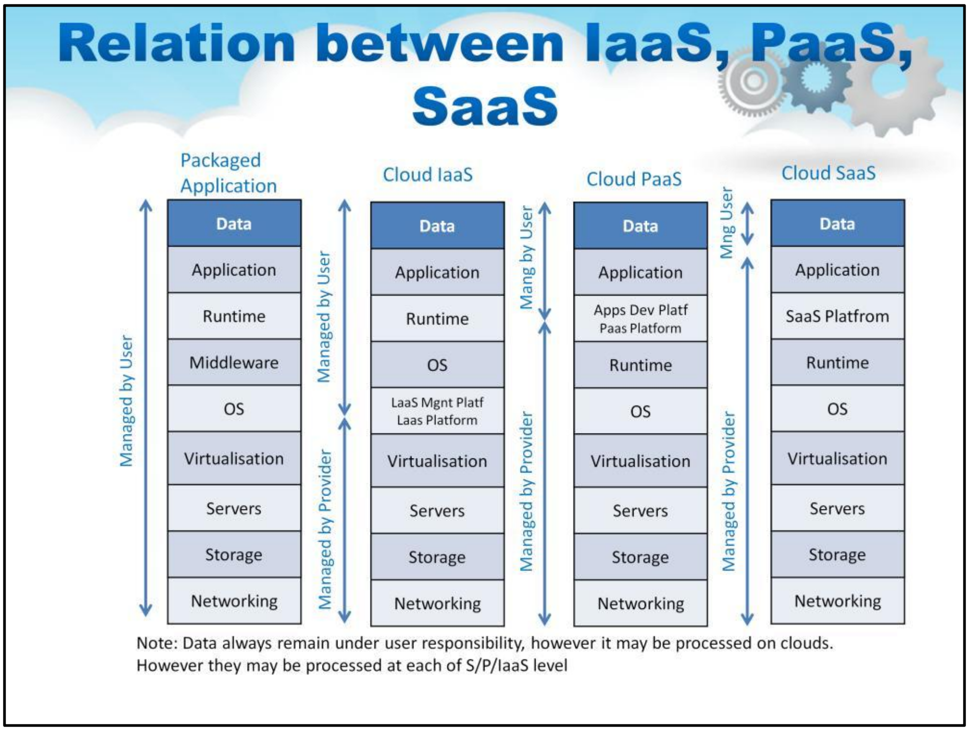
provides high-level APIs used to dereference various low-level details of underlying network infrastructure like

- physical computing resources,
- location,
- data partitioning,
- scaling,
- security,
- backup etc.

Examples

- Infrastructure of few interconnected Virtual Machines (VM) and Storage with user defined characteristics that will run user defined OS and applications
- Physical IT infrastructure of interconnected computers and storage devices is replicated into virtualised infrastructure in cloud

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).



Relation between IaaS, PaaS, SaaS

Packaged Application	Cloud IaaS	Cloud PaaS	Cloud SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Runtime	Runtime	Apps Dev Platf Paas Platform	SaaS Platfrom
Middleware	OS	Runtime	Runtime
OS	LaaS Mgnt Platf Laas Platform	OS	OS
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Note: Data always remain under user responsibility, however it may be processed on clouds. However they may be processed at each of S/P/IaaS level

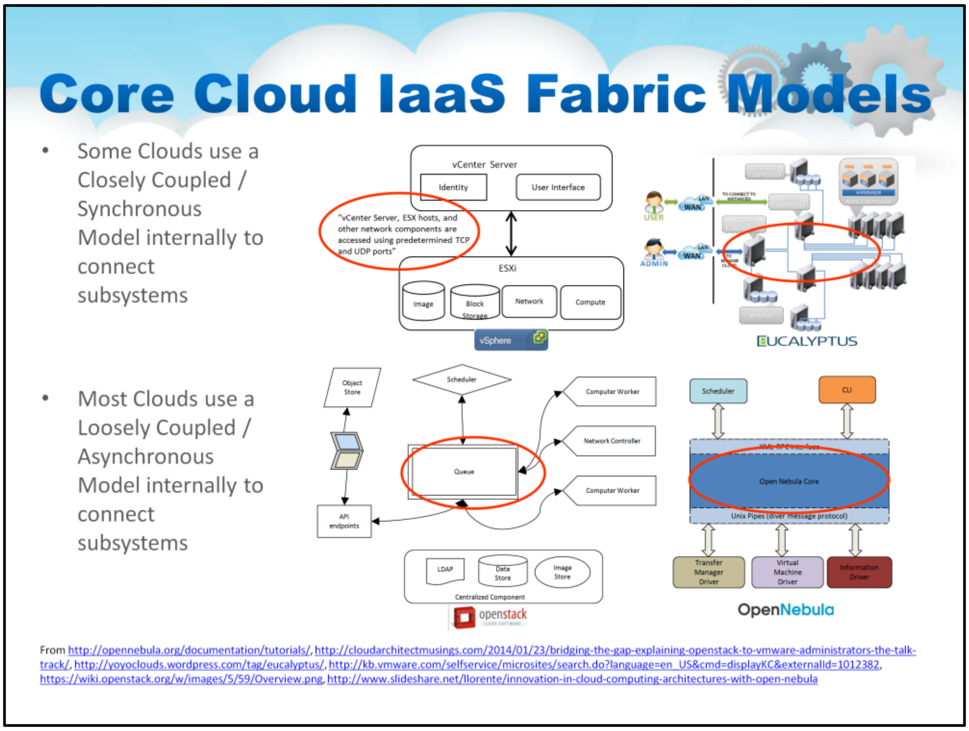
The use models for Cloud Computing are closely related. This illustration has been covered in an earlier Lesson, but to re-iterate.

For a traditional “physically installed” packaged application, the entire deployment stack is managed by the user, as can be seen in the first block of the illustration. The user is responsible for provisioning the complete server including networking and storage, and the Operating System software as well. Additionally, the user provisions whatever databases and middleware they need, any special runtime (Java or Ruby), and finally the application. The application data is also the responsibility (and the ownership of) the user.

The Cloud IaaS model changes this significantly, alleviating a large portion of at least what would be considered “physical” infrastructure, this is shown in the second block of the illustration. The cloud management (sometimes called Cloud OS) software is shown in this diagram as “IaaS Mngt Platf”. The user obtains the hardware in a virtualized form from the IaaS CSP and worries only about the software stack, application, and data.

The Cloud PaaS model further offers middleware, application runtime, and as much “software” infrastructure as possible, creating a convenient container for the application developer. While this requires the developer to restructure if not rewrite the application at the source code level to interface to the PaaS, this restricting will impart new scalability and availability strengths into the application.

The Cloud IaaS model is a consuming of just the application itself through a browser or web service.



Core Cloud IaaS Fabric Models

- Some Clouds use a Closely Coupled / Synchronous Model internally to connect subsystems

"vCenter Server, ESX hosts, and other network components are accessed using predetermined TCP and UDP ports"

- Most Clouds use a Loosely Coupled / Asynchronous Model internally to connect subsystems

From http://opennebula.org/documentation/tutorials/, http://cloudarchitectmusings.com/2014/01/23/bridging-the-gap-explaining-openstack-to-vmware-administrators-the-talk-track/, http://yoyoclouds.wordpress.com/tag/eucalyptus/, http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1012382, https://wiki.openstack.org/w/images/5/59/Overview.png, http://www.slideshare.net/llorente/innovation-in-cloud-computing-architectures-with-open-nebula

Internally, the technology inside the Cloud “Operating Systems” amongst the various alternatives is quite different. This has a direct impact on the focus or target use case for that system.

Generally speaking, one of the main goals of a CloudOS is to orchestrate all of the different services on the servers manipulating VMs, managing distributed storage, configuring network parameters, and so on. Therefore, the communications mechanism used by the CloudOS is like the “backbone” of the system and is a key architectural design element in the system.

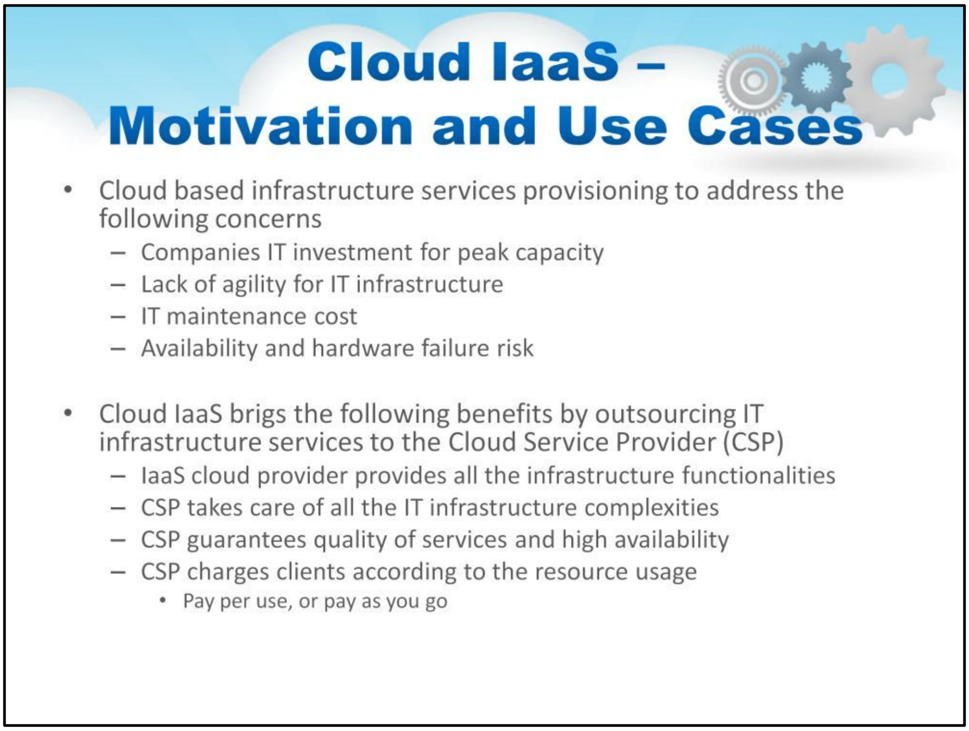
These communications mechanisms can be placed into two large categories: a Closely Coupled / Synchronous Model , versus a Loosely Coupled / Asynchronous Model.

A Closely Coupled / Synchronous Model is simpler architecturally. It is designed primarily to provide for excellent performance, visibility, and control characteristics. Usually it involves TCP/IP and Socket connections and perhaps Unicast or Multicast on top of that for communications. As a side effect of this architecture, scalability limits appear and also, single points of failure (the central orchestration element) appear. These architectures are excellent for smaller private clouds where large scale is not needed, and where tight control of running applications is needed (for example, older/legacy software running on the nodes).

As examples of a Closely Coupled / Synchronous Model - VMware vSphere and Eucalyptus, both use TCP/IP and Sockets as a fabric.

A Loosely Coupled / Asynchronous Model is more complicated architecturally. It is designed primarily to provide for excellent scalability and for high availability characteristics. Usually it involves a Message Queue system for communications. As a side effect of this architecture, there can be less granular control and less real-time control of the nodes. These architectures are excellent for larger clouds, especially the largest public clouds where scale and redundancy of controlling elements is required.

As examples of a Loosely Coupled / Asynchronous Model – OpenStack and OpenNebula both use a message queue mechanism as a fabric.



Cloud IaaS – Motivation and Use Cases

- Cloud based infrastructure services provisioning to address the following concerns
  - Companies IT investment for peak capacity
  - Lack of agility for IT infrastructure
  - IT maintenance cost
  - Availability and hardware failure risk
- Cloud IaaS brigs the following benefits by outsourcing IT infrastructure services to the Cloud Service Provider (CSP)
  - IaaS cloud provider provides all the infrastructure functionalities
  - CSP takes care of all the IT infrastructure complexities
  - CSP guarantees quality of services and high availability
  - CSP charges clients according to the resource usage
    - Pay per use, or pay as you go

How does one choose amongst these different approaches to using the cloud? There are many driving factors.

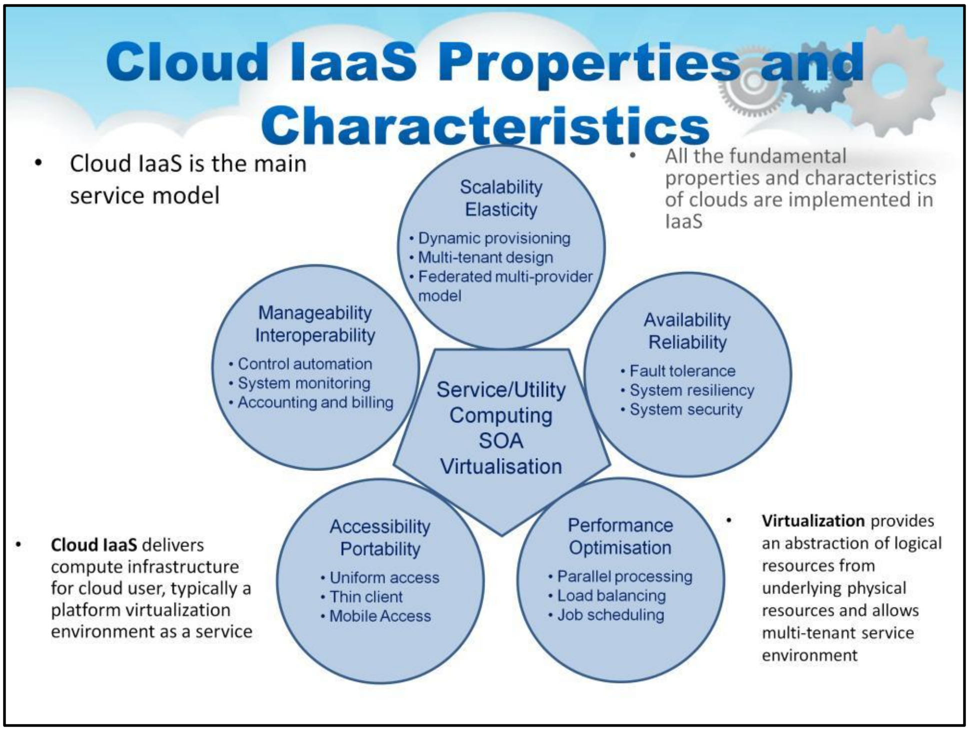
IaaS is in some ways and easier mode to adopt, especially if one is using commercial software, and not developing for PaaS within the IT shop.

This slide covers several of the reasons why companies look to IaaS. Some of these requirements are all expressed at the “physical deployment” conceptual levels, in other words “need more servers” type of concern; some are concerned with operational improvements such as agility or availability.

IaaS can take care of may of these benefits as long as the applications are moved to the IaaS following several best practices. To realize Cloud benefits one should detail the benefits sought from the Cloud Service Provider:

CSP provides substitute for our own physical infrastructure.
CSP takes care of all the IT infrastructure complexities
CSP guarantees quality of services and high availability
CSP charges clients according to the resource usage

Make sure an analysis is done to understand the expectation and impact of each of these on the department.



Cloud IaaS Properties and Characteristics

- Cloud IaaS is the main service model
- All the fundamental properties and characteristics of clouds are implemented in IaaS

Scalability
Elasticity
- Dynamic provisioning
- Multi-tenant design
- Federated multi-provider model

Manageability
Interoperability
- Control automation
- System monitoring
- Accounting and billing

Service/Utility
Computing
SOA
Virtualisation

Availability
Reliability
- Fault tolerance
- System resiliency
- System security

Accessibility
Portability
- Uniform access
- Thin client
- Mobile Access

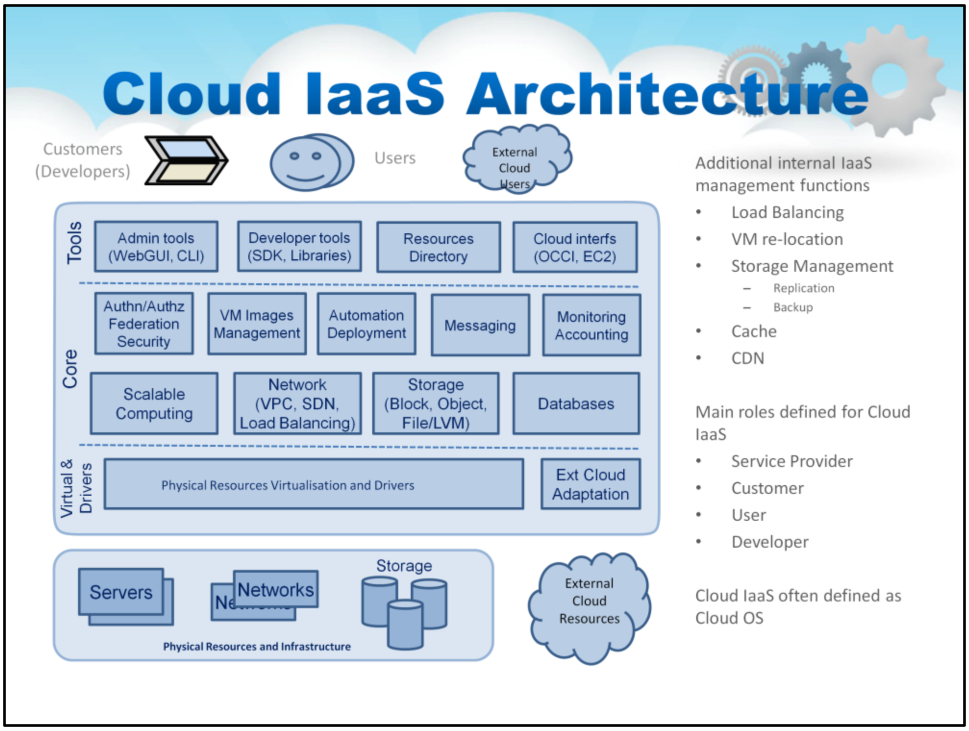
Performance
Optimisation
- Parallel processing
- Load balancing
- Job scheduling

- **Cloud IaaS** delivers compute infrastructure for cloud user, typically a platform virtualization environment as a service
- **Virtualization** provides an abstraction of logical resources from underlying physical resources and allows multi-tenant service environment

Cloud IaaS represents a complete set of cloud characteristics. It is more than simly using virtualization, or in running something outside of your datacenter. All the fundamental properties and characteristics of clouds are implemented in IaaS and this creates a synergy which enables new characteristics.

The Cloud combines the following technologies: Service/Utility Computing (automation), SOA, and Virtualisation, where virtualisation is the key technology to enable all cloud properties.

Cloud does not automatically provide new magical capabilities to an application unless it is deployed properly to take advantage of the new capability. For example, scalability and elasticity can be had on cloud but requires capabilities in the modules you are going to scale up and down, and also requires some control or automation code which needs to be customized for that application. Cloud enables this new capability but not automatically. We will look at Scalability / Elasticity, Availability / Reliability, Performance Optimization, Accessibility / Portability, and Manageability and Interoperability as we study more in IaaS, and learn how to enable the features in one’s application deployment.



Cloud IaaS Architecture

Customers (Developers)

Users

External Cloud Users

Tools

Admin tools (WebGUI, CLI)

Developer tools (SDK, Libraries)

Resources Directory

Cloud interfs (OCCI, EC2)

Core

Authn/Authz Federation Security

VM Images Management

Automation Deployment

Messaging

Monitoring Accounting

Scalable Computing

Network (VPC, SDN, Load Balancing)

Storage (Block, Object, File/LVM)

Databases

Virtual & Drivers

Physical Resources Virtualisation and Drivers

Ext Cloud Adaptation

Servers

Networks

Storage

Physical Resources and Infrastructure

External Cloud Resources

Additional internal IaaS management functions

- Load Balancing
- VM re-location
- Storage Management
  - Replication
  - Backup
- Cache
- CDN

Main roles defined for Cloud IaaS

- Service Provider
- Customer
- User
- Developer

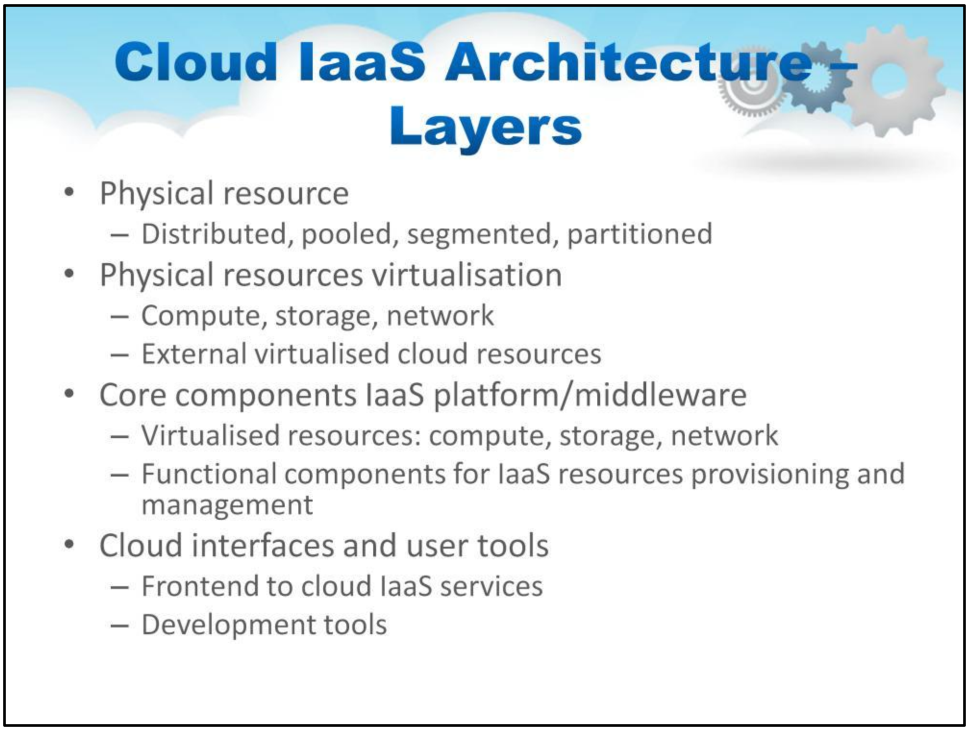
Cloud IaaS often defined as Cloud OS

Cloud IaaS is often defined as the Cloud OS (example OpenStack, Microsoft) as in general it provides all necessary functions for managing cloud resources and supporting applications.

External cloud resources can be different virtualised resources (servers, storage) or resources provided by other clouds.
External resources can use cloud APIs

Note difference between roles: customer and user
Customer is an entity ordering/buying cloud services. They install and run/administer cloud based applications or services. Like website. Developers are typically have a full control over cloud resources.

Users are entities using cloud based services like website users.



Cloud IaaS Architecture – Layers

- Physical resource
  - Distributed, pooled, segmented, partitioned
- Physical resources virtualisation
  - Compute, storage, network
  - External virtualised cloud resources
- Core components IaaS platform/middleware
  - Virtualised resources: compute, storage, network
  - Functional components for IaaS resources provisioning and management
- Cloud interfaces and user tools
  - Frontend to cloud IaaS services
  - Development tools

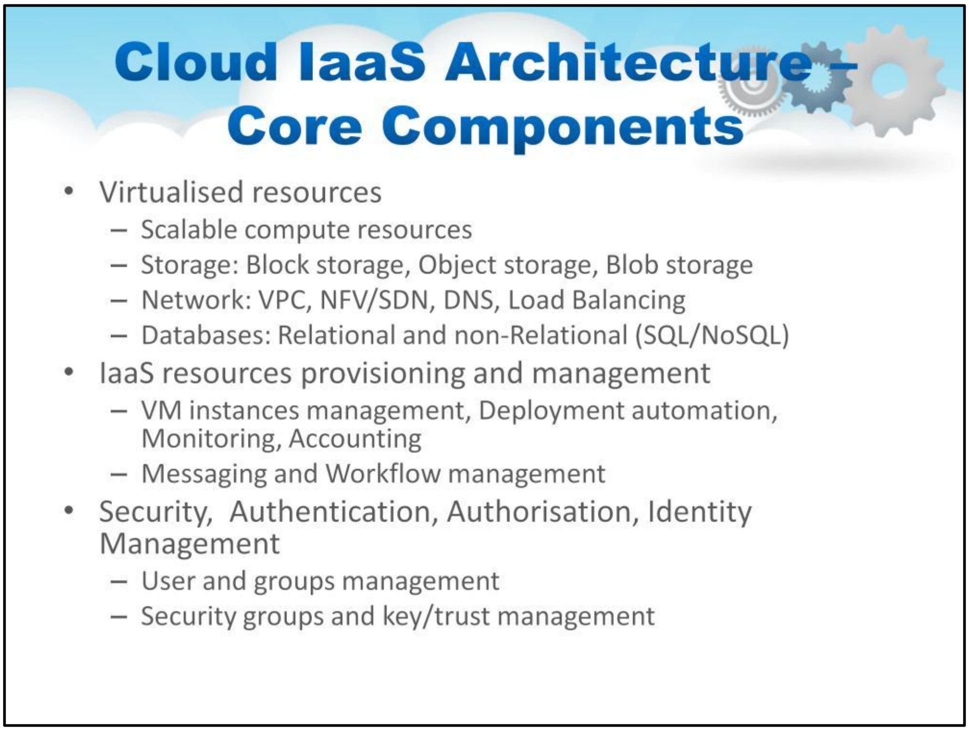
As can be seen in the previous illustration, Cloud IaaS has easy to discern architecture layers.

The Physical Resource layer is the hardware and firmware wired up in a particular topology with network and storage

The Virtualization layer virtualizes and places under software control the physical resource layer

Next running on the Virtualization are the cloud management components, they are the “Cloud OS” modules

Finally there are interfaces and tools which we ultimately go through to access the cloud internals



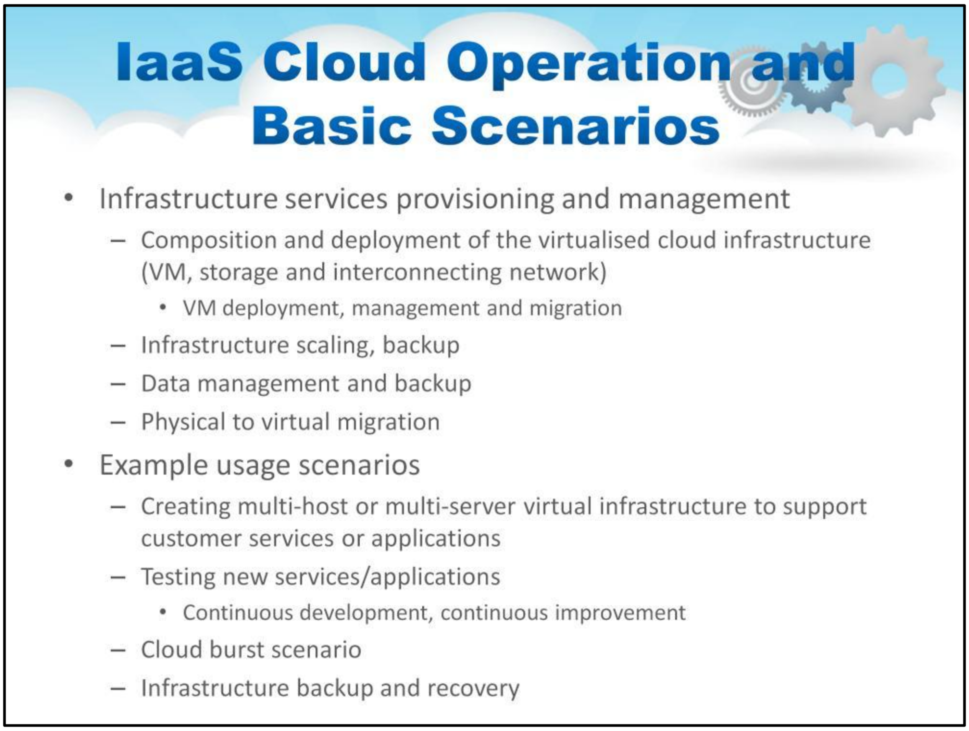
Cloud IaaS Architecture – Core Components

- Virtualised resources
  - Scalable compute resources
  - Storage: Block storage, Object storage, Blob storage
  - Network: VPC, NFV/SDN, DNS, Load Balancing
  - Databases: Relational and non-Relational (SQL/NoSQL)
- IaaS resources provisioning and management
  - VM instances management, Deployment automation, Monitoring, Accounting
  - Messaging and Workflow management
- Security, Authentication, Authorisation, Identity Management
  - User and groups management
  - Security groups and key/trust management

The Cloud OS has gotten to be very full featured and continues to grow in capability. His job was initially responsible for core IaaS functionality of virtualized resources and automating key processes in provisioning resources.

Now the Cloud OS also provides multiple storage models, software controlled networking with a variety of features, and support for many kinds of data storage (data base).

Other functions of the Cloud OS are detailed on the slide, where you can see mechanisms including Authentication, Key Mangement, Accounting, Workflow, and other stuff.



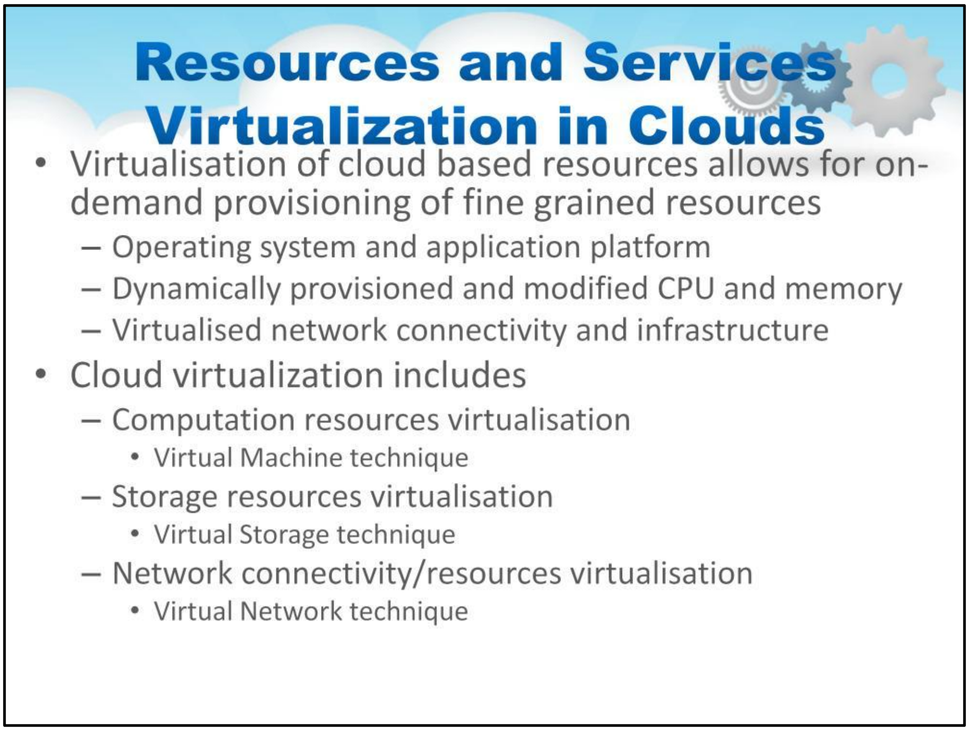
IaaS Cloud Operation and Basic Scenarios

- Infrastructure services provisioning and management
  - Composition and deployment of the virtualised cloud infrastructure (VM, storage and interconnecting network)
    - VM deployment, management and migration
  - Infrastructure scaling, backup
  - Data management and backup
  - Physical to virtual migration
- Example usage scenarios
  - Creating multi-host or multi-server virtual infrastructure to support customer services or applications
  - Testing new services/applications
    - Continuous development, continuous improvement
  - Cloud burst scenario
  - Infrastructure backup and recovery

Let us look at a basic IaaS cloud operation couple of scenarios.

The topics we will consider include
Composition and deployment of the VM’s we’ll need
Infrastructure scaling, backup
Data management and backup
And then Physical to virtual migration

From these basic steps we will “paper construct” a few common deployment scenarios
We will look at creating a multi-host system, how we do testing, how does Cloud Burst work, and what about backup and recovery



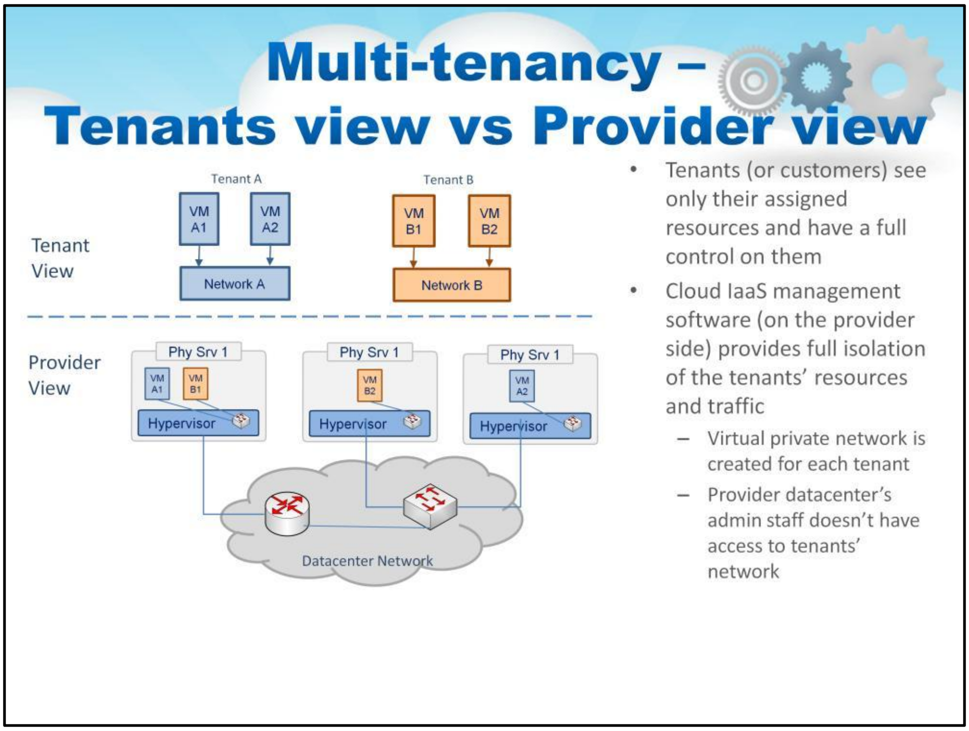
Resources and Services Virtualization in Clouds

- Virtualisation of cloud based resources allows for on-demand provisioning of fine grained resources
  - Operating system and application platform
  - Dynamically provisioned and modified CPU and memory
  - Virtualised network connectivity and infrastructure
- Cloud virtualization includes
  - Computation resources virtualisation
    - Virtual Machine technique
  - Storage resources virtualisation
    - Virtual Storage technique
  - Network connectivity/resources virtualisation
    - Virtual Network technique

To begin with, we concentrate on Resources and Services Virtualization in Clouds.

We are going to deploy the actual servers we need to run our applications. Through well defined interfaces to the cloud, for example web services or REST, the Cloud OS Dynamically provisions CPU and memory, arranges for Virtualised network connectivity and infrastructure.

Cloud virtualization includes
-Computation resources virtualization Virtual Machine technique
-Storage resources virtualization, Virtual Storage technique
- Network connectivity/resources virtualization, Virtual Network technique



Multi-tenancy – Tenants view vs Provider view

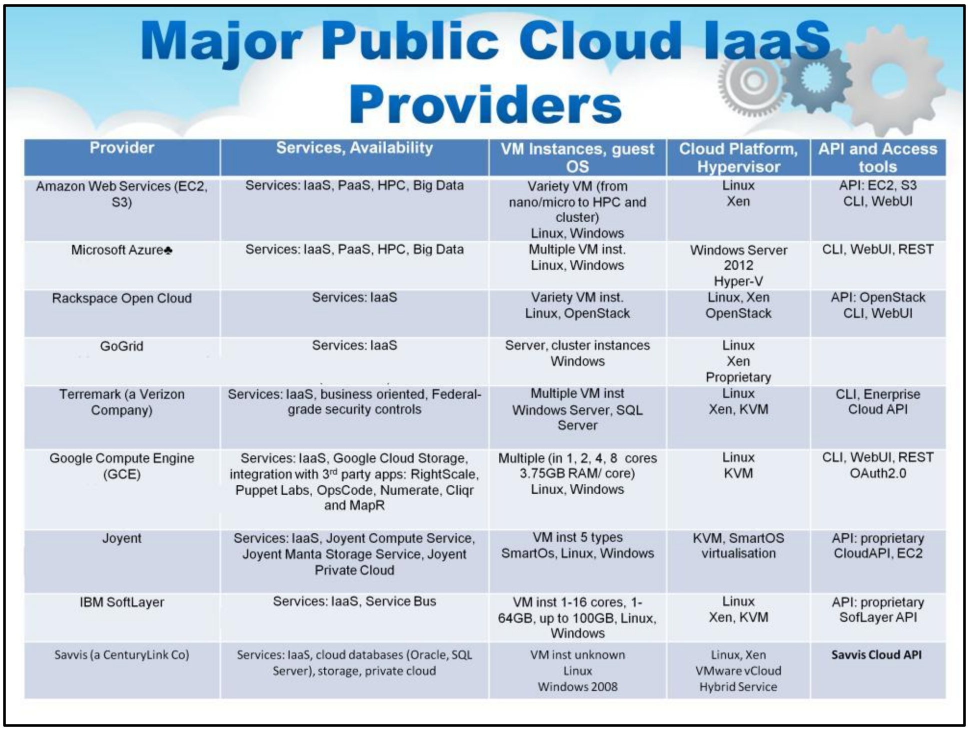
- Tenants (or customers) see only their assigned resources and have a full control on them
- Cloud IaaS management software (on the provider side) provides full isolation of the tenants' resources and traffic
  - Virtual private network is created for each tenant
  - Provider datacenter's admin staff doesn't have access to tenants' network

How does the Cloud actually implement the programmatic availability of VM's, served up to different tenants?

The illustration shows the "logical" view of Virtual Machines and Networks. Note the color coding.

The provider "serves up" the cloud resources using a variety of options, given the pool of physical servers it has in the datacenter.

This illustration shows the Physical Server 1 is running a VM for each tenant, sharing the machine resources across the two VMs. Other VM's are scattered about on other servers in the cloud.



Major Public Cloud IaaS Providers

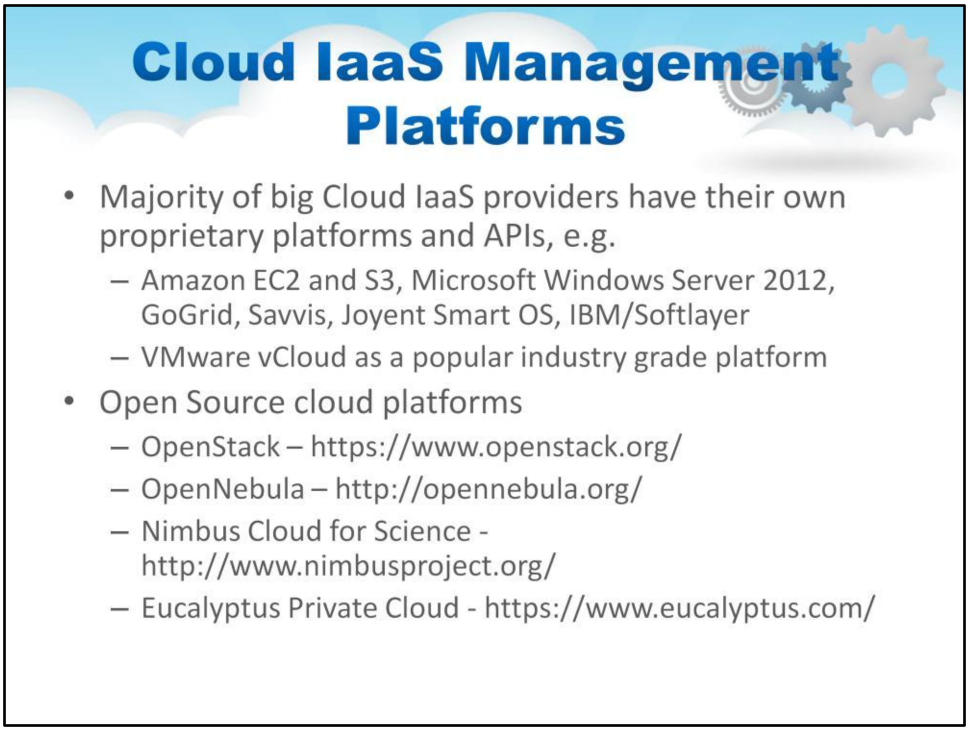
Provider	Services, Availability	VM Instances, guest OS	Cloud Platform, Hypervisor	API and Access tools
Amazon Web Services (EC2, S3)	Services: IaaS, PaaS, HPC, Big Data	Variety VM (from nano/micro to HPC and cluster) Linux, Windows	Linux Xen	API: EC2, S3 CLI, WebUI
Microsoft Azure♣	Services: IaaS, PaaS, HPC, Big Data	Multiple VM inst. Linux, Windows	Windows Server 2012 Hyper-V	CLI, WebUI, REST
Rackspace Open Cloud	Services: IaaS	Variety VM inst. Linux, OpenStack	Linux, Xen OpenStack	API: OpenStack CLI, WebUI
GoGrid	Services: IaaS	Server, cluster instances Windows	Linux Xen Proprietary	
Terremark (a Verizon Company)	Services: IaaS, business oriented, Federal-grade security controls	Multiple VM inst Windows Server, SQL Server	Linux Xen, KVM	CLI, Enerprise Cloud API
Google Compute Engine (GCE)	Services: IaaS, Google Cloud Storage, integration with 3rd party apps: RightScale, Puppet Labs, OpsCode, Numerate, Cliqr and MapR	Multiple (in 1, 2, 4, 8 cores 3.75GB RAM/ core) Linux, Windows	Linux KVM	CLI, WebUI, REST OAuth2.0
Joyent	Services: IaaS, Joyent Compute Service, Joyent Manta Storage Service, Joyent Private Cloud	VM inst 5 types SmartOs, Linux, Windows	KVM, SmartOS virtualisation	API: proprietary CloudAPI, EC2
IBM SoftLayer	Services: IaaS, Service Bus	VM inst 1-16 cores, 1-64GB, up to 100GB, Linux, Windows	Linux Xen, KVM	API: proprietary SofLayer API
Savvis (a CenturyLink Co)	Services: IaaS, cloud databases (Oracle, SQL Server), storage, private cloud	VM inst unknown Linux Windows 2008	Linux, Xen VMware vCloud Hybrid Service	**Savvis Cloud API**

IaaS cloud providers differ in platform, API, location but majority offer both Linux and Windows VM images. Some CSP use proprietary cloud management platforms, some also offer VMware cloud management platforms.

Large CSP offer services worldwide and have multiple availability zones.

(CSP1, 2104) IaaS Providers List: 2014 Comparison And Guide. [online] http://www.tomsitpro.com/articles/iaas-providers,1-1560.html

(CSP2, 2014) Joe Panettieri, Top 100 Cloud Services Providers (CSPs) List And Research [online] http://talkincloud.com/tc100



Cloud IaaS Management Platforms

- Majority of big Cloud IaaS providers have their own proprietary platforms and APIs, e.g.
  - Amazon EC2 and S3, Microsoft Windows Server 2012, GoGrid, Savvis, Joyent Smart OS, IBM/Softlayer
  - VMware vCloud as a popular industry grade platform
- Open Source cloud platforms
  - OpenStack – https://www.openstack.org/
  - OpenNebula – http://opennebula.org/
  - Nimbus Cloud for Science - http://www.nimbusproject.org/
  - Eucalyptus Private Cloud - https://www.eucalyptus.com/

The challenge for the developer is, that even IaaS API’s and conventions are different across cloud service providers.

The slide lists several Cloud Service Providers as well as Cloud OS software alternatives.

While similar in design, there are a variety of cloud platforms to choose from depending on what is driving your choice.

Cloud Service Models

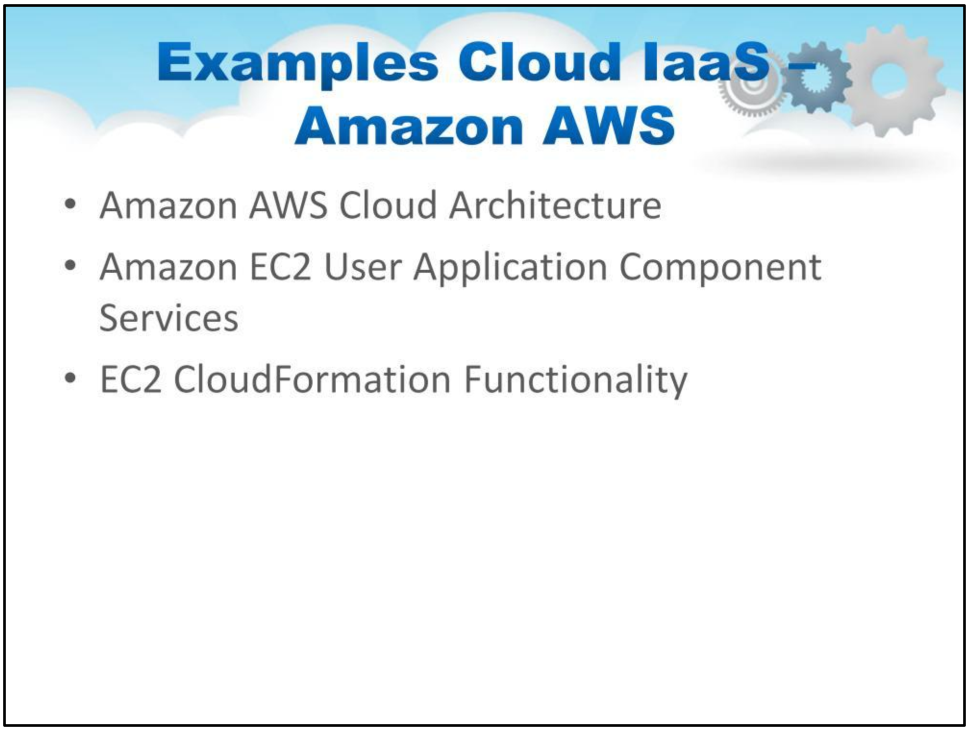
–

Infrastructure as a Service (IaaS)

–

Example: Amazon AWS

Example: Amazon AWS



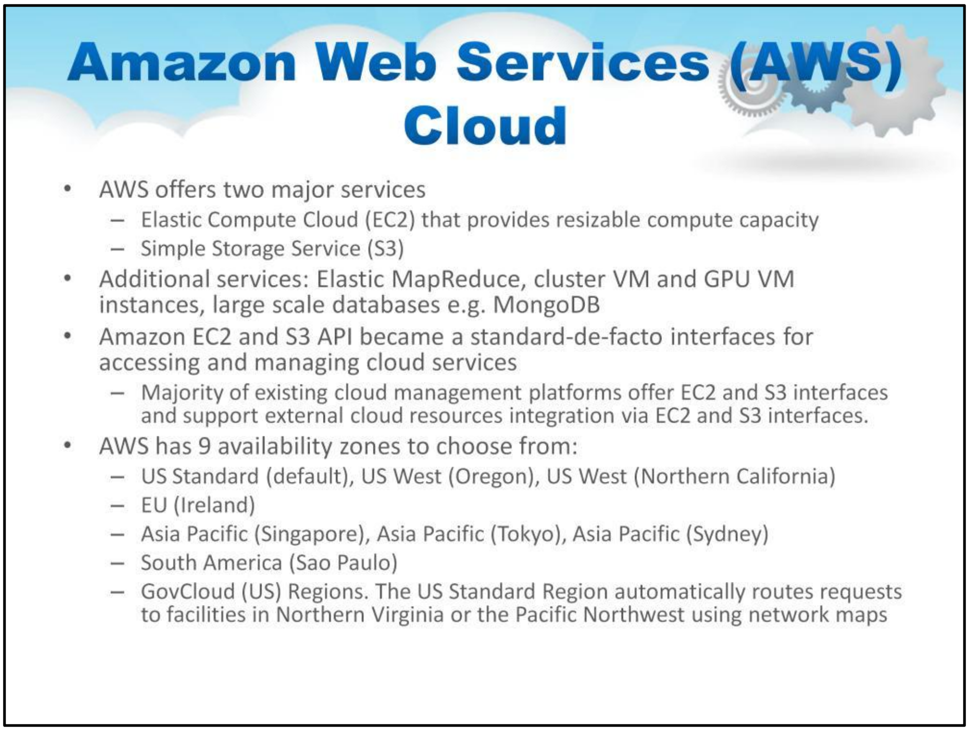
Examples Cloud IaaS – Amazon AWS

- Amazon AWS Cloud Architecture
- Amazon EC2 User Application Component Services
- EC2 CloudFormation Functionality

The most popular IaaS cloud is Amazon AWS. For that reason we will reference this platform in providing example solutions.

The AWS architecture contains many, many services. The first and most basic ones are "EC2" computing and the associated core parts that come with it (S3 and EBS storage for example).

Therefore let us investigate AWS more.



Amazon Web Services (AWS) Cloud

- AWS offers two major services
  - Elastic Compute Cloud (EC2) that provides resizable compute capacity
  - Simple Storage Service (S3)
- Additional services: Elastic MapReduce, cluster VM and GPU VM instances, large scale databases e.g. MongoDB
- Amazon EC2 and S3 API became a standard-de-facto interfaces for accessing and managing cloud services
  - Majority of existing cloud management platforms offer EC2 and S3 interfaces and support external cloud resources integration via EC2 and S3 interfaces.
- AWS has 9 availability zones to choose from:
  - US Standard (default), US West (Oregon), US West (Northern California)
  - EU (Ireland)
  - Asia Pacific (Singapore), Asia Pacific (Tokyo), Asia Pacific (Sydney)
  - South America (Sao Paulo)
  - GovCloud (US) Regions. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps

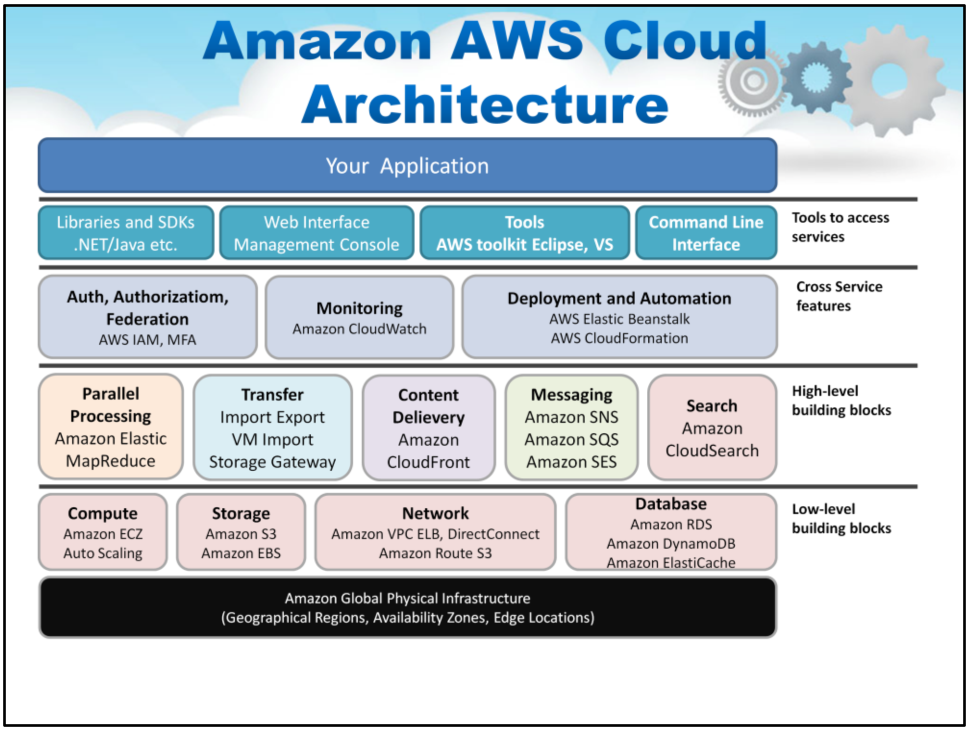
AWS offers two major services

Elastic Compute Cloud (EC2) that provides resizable compute capacity
Simple Storage Service (S3)

AWS also offer many many other services, For example, all kinds of databases are offered, many kinds of VM’s, and middleware components, all “as a service”.

Amazon EC2 and S3 API became a standard-de-facto interfaces for accessing and managing cloud services

AWS has 9 availability zones to choose from:



Amazon AWS Cloud Architecture

Your Application

Libraries and SDKs .NET/Java etc.

Web Interface Management Console

Tools AWS toolkit Eclipse, VS

Command Line Interface

Tools to access services

Auth, Authorizatiom, Federation AWS IAM, MFA

Monitoring Amazon CloudWatch

Deployment and Automation AWS Elastic Beanstalk AWS CloudFormation

Cross Service features

Parallel Processing Amazon Elastic MapReduce

Transfer Import Export VM Import Storage Gateway

Content Delievery Amazon CloudFront

Messaging Amazon SNS Amazon SQS Amazon SES

Search Amazon CloudSearch

High-level building blocks

Compute Amazon ECZ Auto Scaling

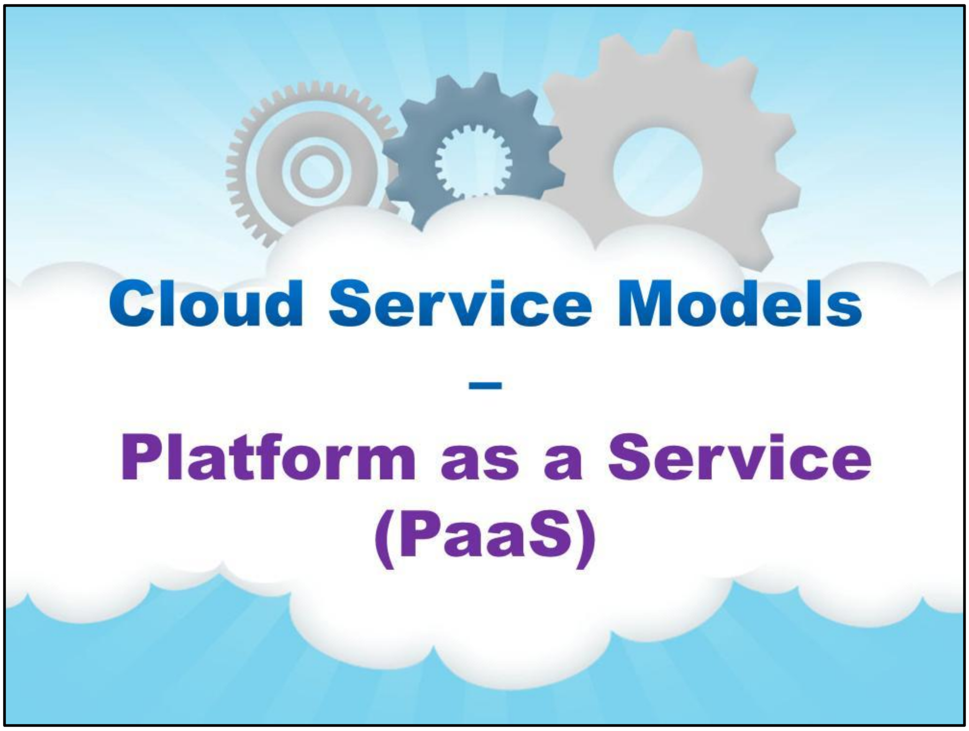
Storage Amazon S3 Amazon EBS

Network Amazon VPC ELB, DirectConnect Amazon Route S3

Database Amazon RDS Amazon DynamoDB Amazon ElastiCache

Low-level building blocks

Amazon Global Physical Infrastructure (Geographical Regions, Availability Zones, Edge Locations)

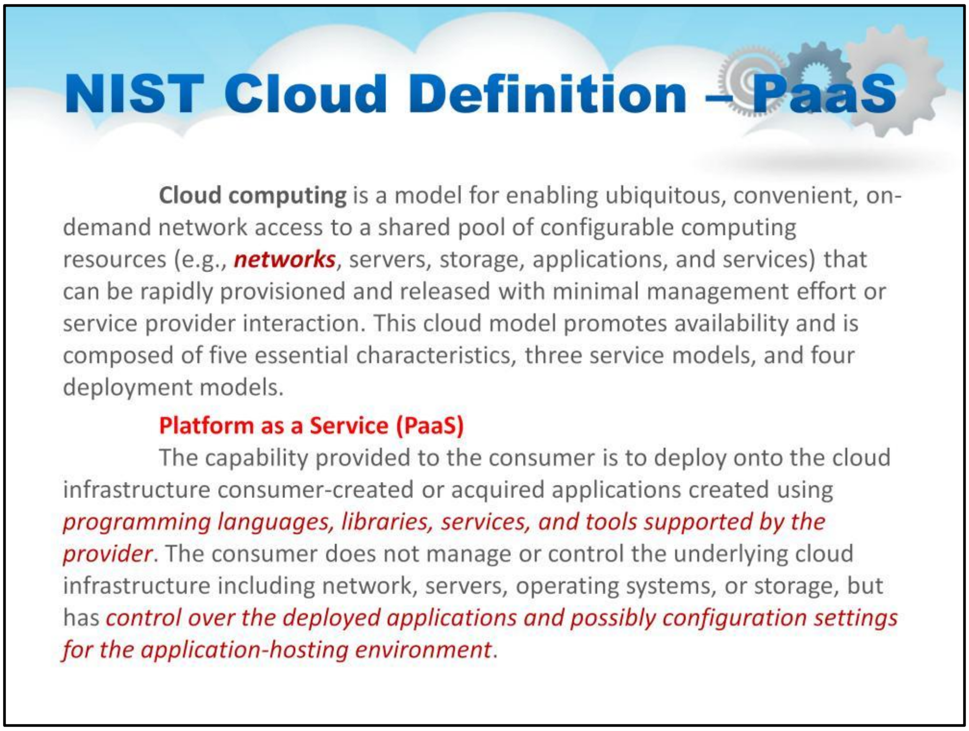


Cloud Service Models

–

Platform as a Service (PaaS)

Platform as a Service (PaaS)



NIST Cloud Definition – PaaS

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., ***networks***, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

Platform as a Service (PaaS)

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using *programming languages, libraries, services, and tools supported by the provider*. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has *control over the deployed applications and possibly configuration settings for the application-hosting environment*.

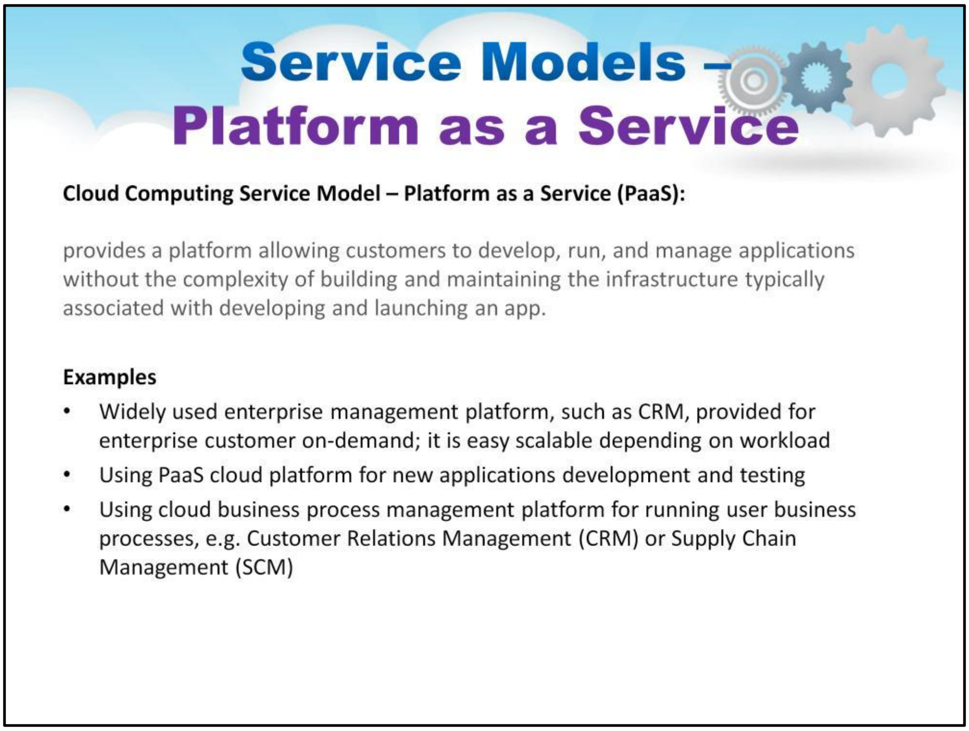
This slide quotes the NIST Cloud Computing definition of Platform as a Service.

First the slide repeats the NIST definition of Cloud Computing in general.

Note the emphasis on "ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources"

In the NIST definition for Platform as a Service (PaaS), the emphasis is on "programming languages, libraries, services, and tools supported by the provider. "

So PaaS is about an environment for developers!



Service Models – Platform as a Service

Cloud Computing Service Model – Platform as a Service (PaaS):

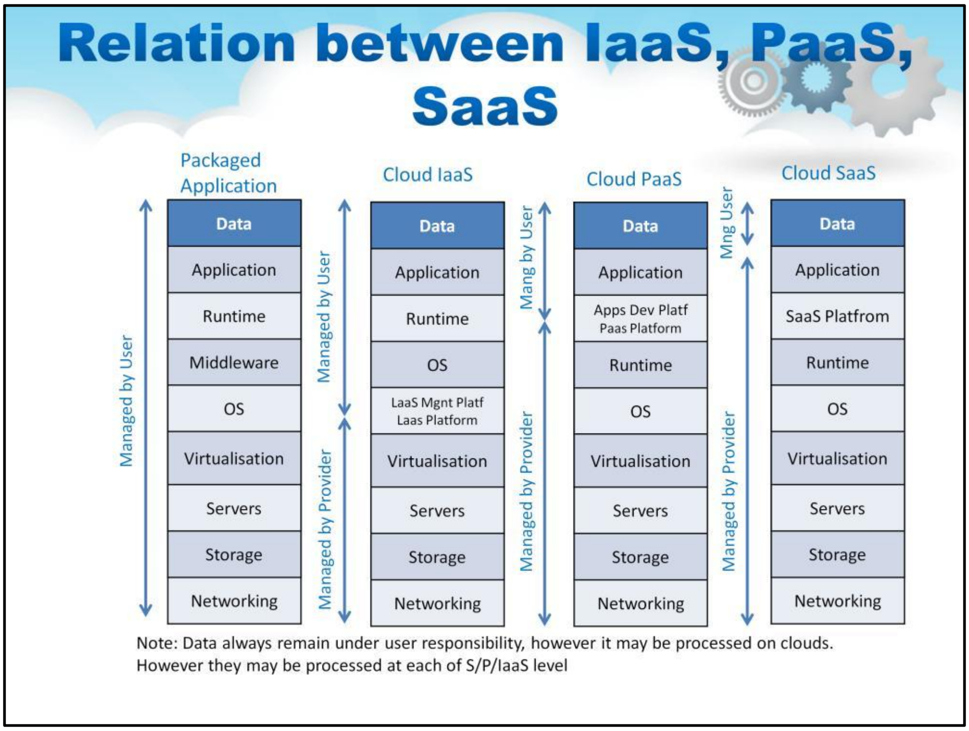
provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

Examples

- Widely used enterprise management platform, such as CRM, provided for enterprise customer on-demand; it is easy scalable depending on workload
- Using PaaS cloud platform for new applications development and testing
- Using cloud business process management platform for running user business processes, e.g. Customer Relations Management (CRM) or Supply Chain Management (SCM)

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using *programming languages, libraries, services, and tools supported by the provider*. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has *control over the deployed applications and possibly configuration settings for the application-hosting environment*.

PaaS can be delivered in three ways:
- as a public cloud service from a provider, where the consumer controls software deployment with minimal configuration options, and the provider provides the networks, servers, storage, operating system (OS), middleware (e.g. Java runtime, .NET runtime, integration, etc.), database and other services to host the consumer's application.
- as a private service (software or appliance) inside the firewall.
- as software deployed on a public infrastructure as a service.



Relation between IaaS, PaaS, SaaS

Packaged Application	Cloud IaaS	Cloud PaaS	Cloud SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Runtime	Runtime	Apps Dev Platf Paas Platform	SaaS Platfrom
Middleware	OS	Runtime	Runtime
OS	LaaS Mgnt Platf Laas Platform	OS	OS
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Note: Data always remain under user responsibility, however it may be processed on clouds. However they may be processed at each of S/P/IaaS level

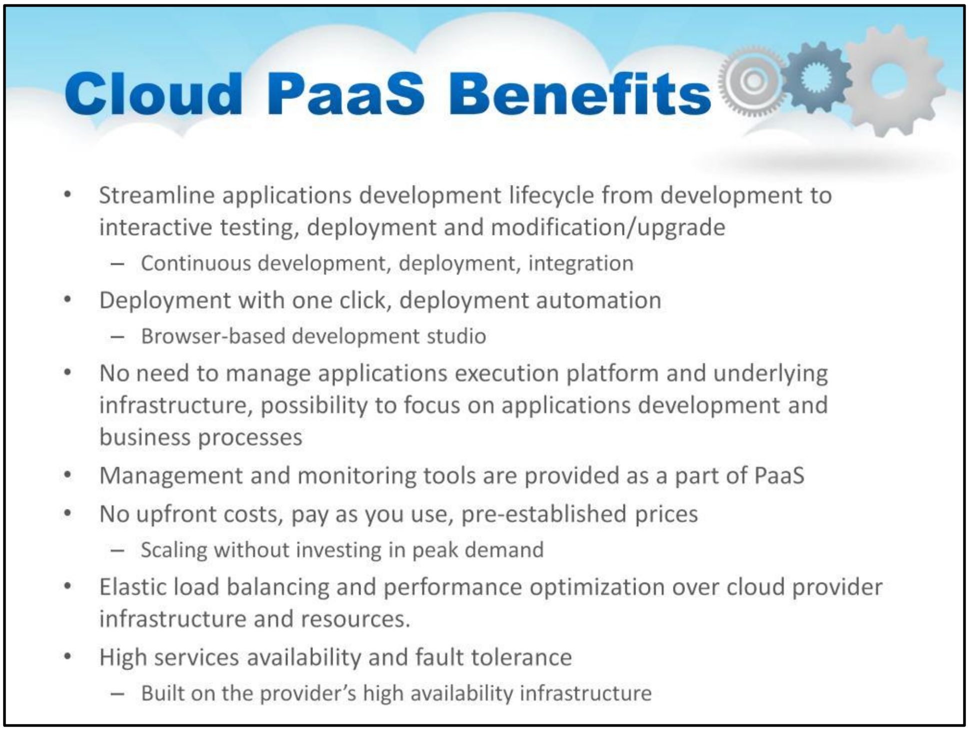
The use models for Cloud Computing are closely related. This illustration has been covered in an earlier Lesson, but to re-iterate.

For a traditional “physically installed” packaged application, the entire deployment stack is managed by the user, as can be seen in the first block of the illustration. The user is responsible for provisioning the complete server including networking and storage, and the Operating System software as well. Additionally, the user provisions whatever databases and middleware they need, any special runtime (Java or Ruby), and finally the application. The application data is also the responsibility (and the ownership of) the user.

The Cloud IaaS model changes this significantly, alleviating a large portion of at least what would be considered “physical” infrastructure, this is shown in the second block of the illustration. The cloud management (sometimes called Cloud OS) software is shown in this diagram as “IaaS Mngt Platf”. The user obtains the hardware in a virtualized form from the IaaS CSP and worries only about the software stack, application, and data.

The Cloud PaaS model further offers middleware, application runtime, and as much “software” infrastructure as possible, creating a convenient container for the application developer. While this requires the developer to restructure if not rewrite the application at the source code level to interface to the PaaS, this restricting will impart new scalability and availability strengths into the application.

The Cloud IaaS model is a consuming of just the application itself through a browser or web service.



Cloud PaaS Benefits

- Streamline applications development lifecycle from development to interactive testing, deployment and modification/upgrade
  - Continuous development, deployment, integration
- Deployment with one click, deployment automation
  - Browser-based development studio
- No need to manage applications execution platform and underlying infrastructure, possibility to focus on applications development and business processes
- Management and monitoring tools are provided as a part of PaaS
- No upfront costs, pay as you use, pre-established prices
  - Scaling without investing in peak demand
- Elastic load balancing and performance optimization over cloud provider infrastructure and resources.
- High services availability and fault tolerance
  - Built on the provider's high availability infrastructure

A PaaS provided by a Cloud provides a lot of benefits to the developer over classic middleware/programming suites.

It brings a Streamlined applications development lifecycle with Continuous development, deployment, integration

It provides Deployment with one click usually with a Browser based development studio

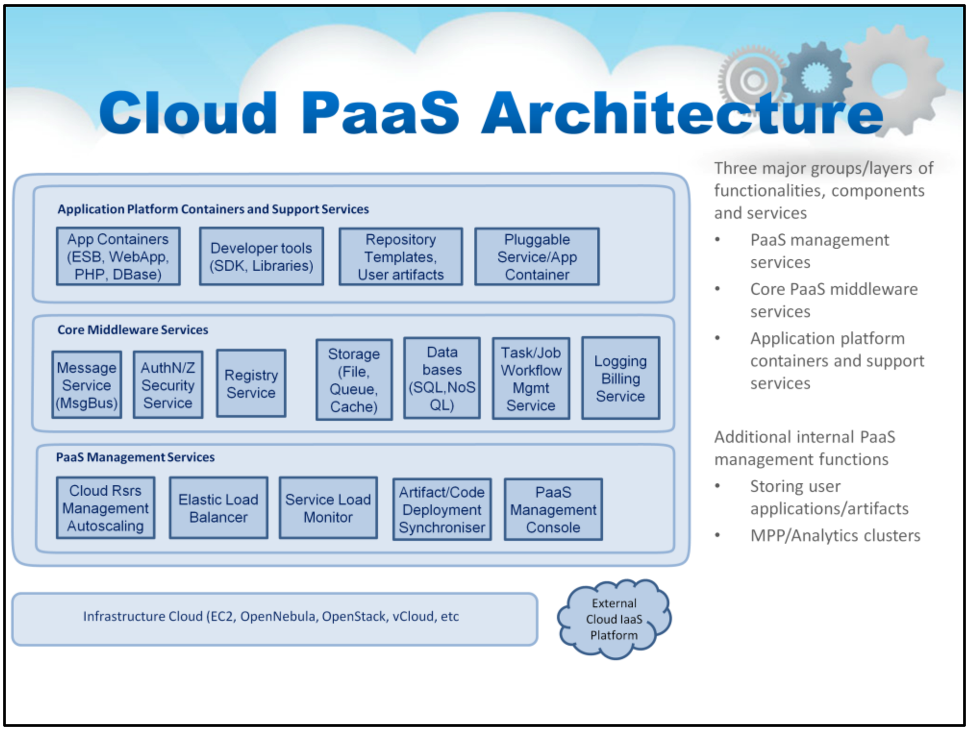
No need to manage applications execution platform and underlying infrastructure

Management and monitoring tools are provided

No upfront costs, pay as you use

More Automatic Elastic load balancing and performance optimization

High services availability



Cloud PaaS Architecture

Application Platform Containers and Support Services

App Containers (ESB, WebApp, PHP, DBase) | Developer tools (SDK, Libraries) | Repository Templates, User artifacts | Pluggable Service/App Container

Core Middleware Services

Message Service (MsgBus) | AuthN/Z Security Service | Registry Service | Storage (File, Queue, Cache) | Data bases (SQL,NoSQL) | Task/Job Workflow Mgmt Service | Logging Billing Service

PaaS Management Services

Cloud Rsrs Management Autoscaling | Elastic Load Balancer | Service Load Monitor | Artifact/Code Deployment Synchroniser | PaaS Management Console

Infrastructure Cloud (EC2, OpenNebula, OpenStack, vCloud, etc

External Cloud IaaS Platform

Three major groups/layers of functionalities, components and services

- PaaS management services
- Core PaaS middleware services
- Application platform containers and support services

Additional internal PaaS management functions

- Storing user applications/artifacts
- MPP/Analytics clusters

This slide contains an illustration of a Cloud PaaS Architecture

Of course, underneath a PaaS, there is an Infrastructure Cloud (EC2, OpenNebula, OpenStack, vCloud, etc. This can be local or it can be an External Cloud IaaS Platform

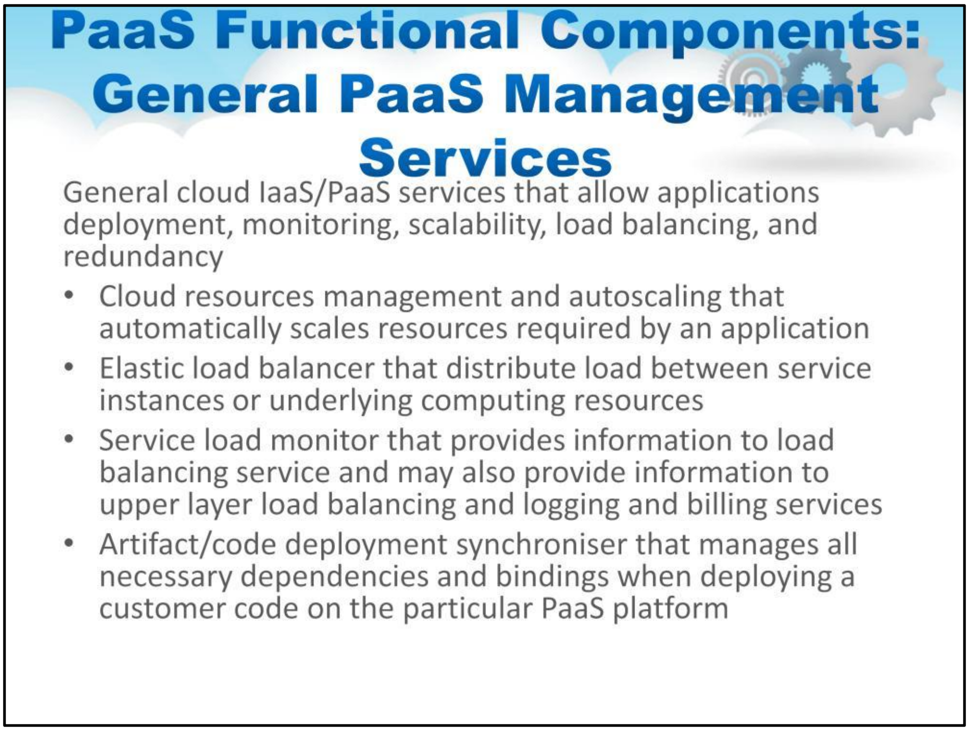
There ae Three major groups/layers of functionalities, components and services

1. PaaS management services such as Cloud Resources Management, Autoscaling, Elastic Load Balancer, Service Load Monitor, and Artifact/Code Deployment, and other subsystems depending on the PaaS variety

2. Core PaaS middleware services such as Authorization and Security Services, Storage (File, Queue, Cache), Data bases (SQL,NoSQL), Task/Job Workflow Mgmt Services, and other subsystems depending on the PaaS variety

3. Application platform containers and support services such as App Containers (ESB, WebApp, PHP, DBase), Developer tools (SDK, Libraries), Repository Templates, User artifacts, and other subsystems depending on the PaaS variety

Additional internal PaaS management functions
Storing user applications/artifacts
MPP/Analytics clusters



PaaS Functional Components: General PaaS Management Services

General cloud IaaS/PaaS services that allow applications deployment, monitoring, scalability, load balancing, and redundancy

- Cloud resources management and autoscaling that automatically scales resources required by an application
- Elastic load balancer that distribute load between service instances or underlying computing resources
- Service load monitor that provides information to load balancing service and may also provide information to upper layer load balancing and logging and billing services
- Artifact/code deployment synchroniser that manages all necessary dependencies and bindings when deploying a customer code on the particular PaaS platform

Let us look at these elements in more detail.

First, the PaaS Functional Components

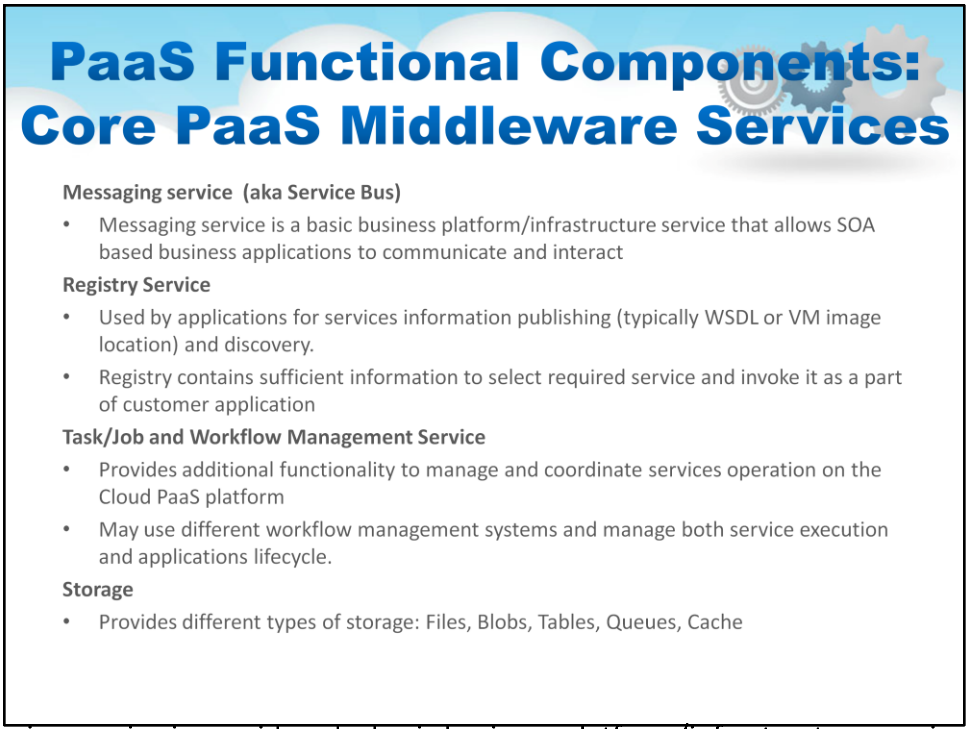
General cloud IaaS/PaaS services allow applications deployment, monitoring, scalability, load balancing, and redundancy

There is Cloud resources management and autoscaling that automatically scales resources required by an application

There is Elastic load balancer that distribute load between service instances or underlying computing resources

There is Service load monitor that provides information to load balancing service

And there is Artifact/code deployment synchroniser that manages all necessary dependencies and bindings when deploying a customer code



PaaS Functional Components: Core PaaS Middleware Services

Messaging service (aka Service Bus)

- Messaging service is a basic business platform/infrastructure service that allows SOA based business applications to communicate and interact

Registry Service

- Used by applications for services information publishing (typically WSDL or VM image location) and discovery.
- Registry contains sufficient information to select required service and invoke it as a part of customer application

Task/Job and Workflow Management Service

- Provides additional functionality to manage and coordinate services operation on the Cloud PaaS platform
- May use different workflow management systems and manage both service execution and applications lifecycle.

Storage

- Provides different types of storage: Files, Blobs, Tables, Queues, Cache

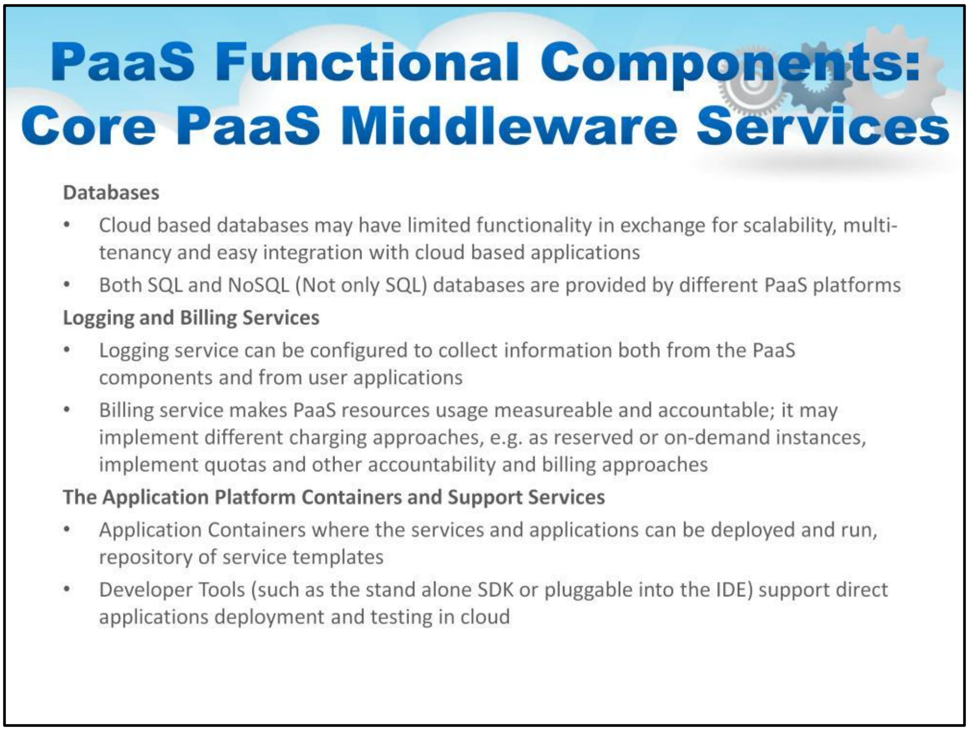
Messaging service is considered a basic business platform/infrastructure service that allows SOA based business applications to communicate and interact.
SOA is the major architecture for business services that use messages for communication between services and processes.

Messaging service has different implementations where the Enterprise Service Bus (ESB) is the most widely used and the standard-de-facto for enterprise SOA based applications. ESB has a number of Open Source and proprietary implementations. We will discuss in a separate tutorial the Azure Service Bus as a cloud based messaging service implementation by Azure.

Registry Service
Maintains the register of services available on the platform, one of the key component of the SOA based environment
Used by applications for services publishing and discovery. Registry contains sufficient information to select required service and invoke it as a part of customer application
The Registry may contain either a service interface description (e.g. WSDL) or reference to the service package or VM image.

Task/Job and Workflow Management Service
Provides additional functionality to manage and coordinate services operation on the Cloud PaaS platform
May use different workflow management systems and manage both service execution and applications lifecycle.

Storage
Provides different types of storage: Files, Blobs, Tables, Queues, Cache



PaaS Functional Components: Core PaaS Middleware Services

Databases

- Cloud based databases may have limited functionality in exchange for scalability, multi-tenancy and easy integration with cloud based applications
- Both SQL and NoSQL (Not only SQL) databases are provided by different PaaS platforms

Logging and Billing Services

- Logging service can be configured to collect information both from the PaaS components and from user applications
- Billing service makes PaaS resources usage measureable and accountable; it may implement different charging approaches, e.g. as reserved or on-demand instances, implement quotas and other accountability and billing approaches

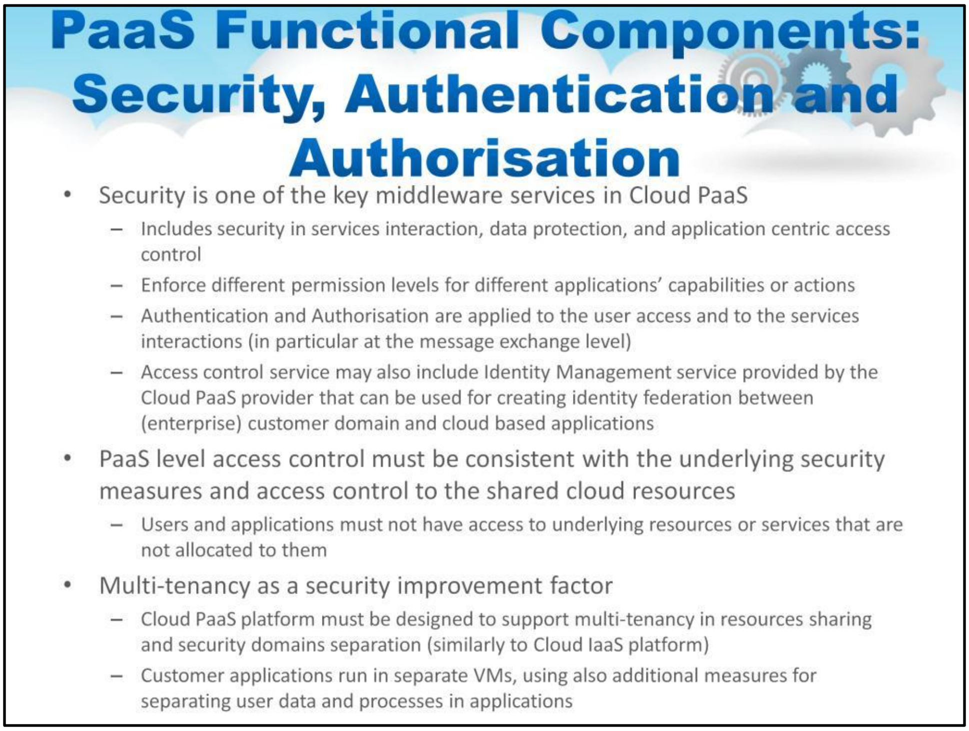
The Application Platform Containers and Support Services

- Application Containers where the services and applications can be deployed and run, repository of service templates
- Developer Tools (such as the stand alone SDK or pluggable into the IDE) support direct applications deployment and testing in cloud

Databases
Scalable Database services is one of the key application building blocks
Cloud based databases may have limited functionality in exchange for scalability, multi-tenancy and easy integration with cloud based applications. Both
SQL and NoSQL (Not only SQL) databases are provided by different PaaS platforms.

Logging and Billing Services
Logging service can be configured to collect information both from the PaaS components and from user applications
Billing service makes PaaS resources usage measureable and accountable; it may implement different charging approaches, e.g. as reserved or on-demand instances, implement quotas and other accountability and billing approaches.

The Application Platform Containers and Support Services
Includes Application Containers where the services and applications can be deployed and run, repository of service templates, and Developer Tools that include Software Development Kits (SDK) that in most cases can be part of the Integrated Development Environment (IDE) or pluggable to one of the popular IDEs such as Eclipse or Microsoft Visual Studio. Such SDK/IDEs support direct applications deployment and testing in cloud. Each Cloud PaaS provider typically supports one or several programming languages and corresponding application containers and SDK. It can also be targeted for different types of applications like websites, web applications, business process management, etc.



PaaS Functional Components: Security, Authentication and Authorisation

- Security is one of the key middleware services in Cloud PaaS
  - Includes security in services interaction, data protection, and application centric access control
  - Enforce different permission levels for different applications' capabilities or actions
  - Authentication and Authorisation are applied to the user access and to the services interactions (in particular at the message exchange level)
  - Access control service may also include Identity Management service provided by the Cloud PaaS provider that can be used for creating identity federation between (enterprise) customer domain and cloud based applications
- PaaS level access control must be consistent with the underlying security measures and access control to the shared cloud resources
  - Users and applications must not have access to underlying resources or services that are not allocated to them
- Multi-tenancy as a security improvement factor
  - Cloud PaaS platform must be designed to support multi-tenancy in resources sharing and security domains separation (similarly to Cloud IaaS platform)
  - Customer applications run in separate VMs, using also additional measures for separating user data and processes in applications

Security is one of key middleware services in Cloud PaaS

Includes security in services interaction, data protection, and application centric access control that allows enforcing different access/permissions level for different applications' capabilities or requested actions.

Authentication and Authorisation is applied to the services interactions (in particular at the message exchange level) and user access.

Access control service may also include Identity Management service provided by the Cloud PaaS provider that can be used for creating identity federation between (enterprise) customer domain and cloud based applications.

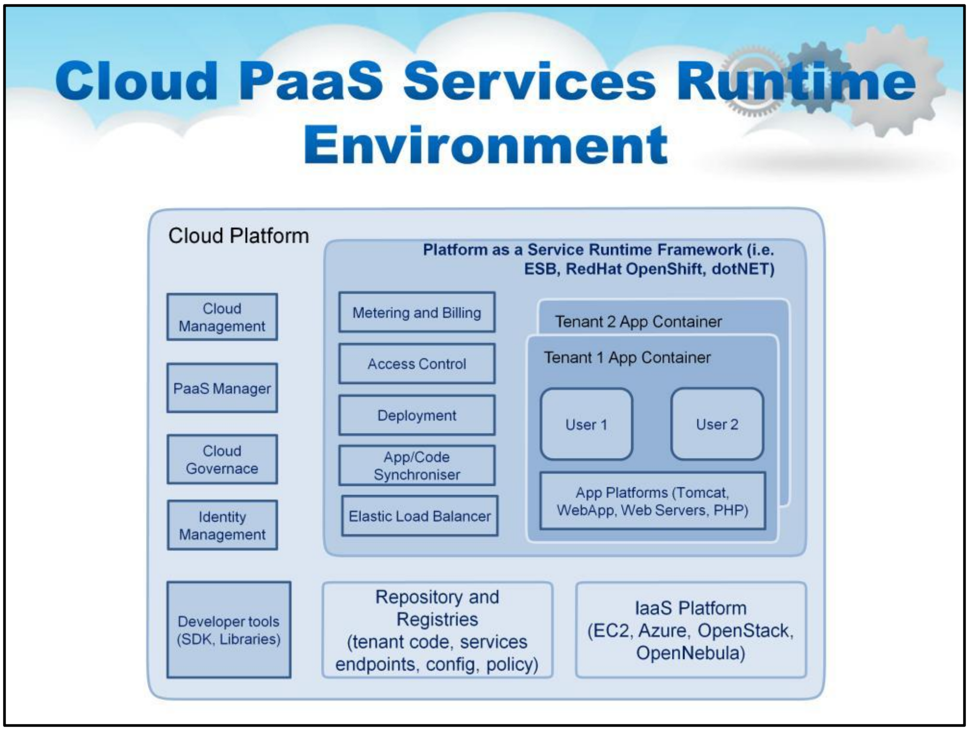
PaaS level access control must be consistent with the underlying security measures and access control to shared cloud resources.

Users and applications must not have access to resources or services that are not allocated to them or which permission level is higher than a particular user's access right.

Multi-tenancy as a security improvement factor

Cloud PaaS platform must be designed to support multi-tenancy in resources sharing and security domains separation, similarly to Cloud IaaS platform

Customer applications run in separate VMs, using also additional measures for separating application users' data and processes.



Cloud PaaS Services Runtime Environment

Cloud Platform

Cloud Management

PaaS Manager

Cloud Governace

Identity Management

Platform as a Service Runtime Framework (i.e. ESB, RedHat OpenShift, dotNET)

Metering and Billing

Access Control

Deployment

App/Code Synchroniser

Elastic Load Balancer

Tenant 2 App Container

Tenant 1 App Container

User 1

User 2

App Platforms (Tomcat, WebApp, Web Servers, PHP)

Developer tools (SDK, Libraries)

Repository and Registries (tenant code, services endpoints, config, policy)

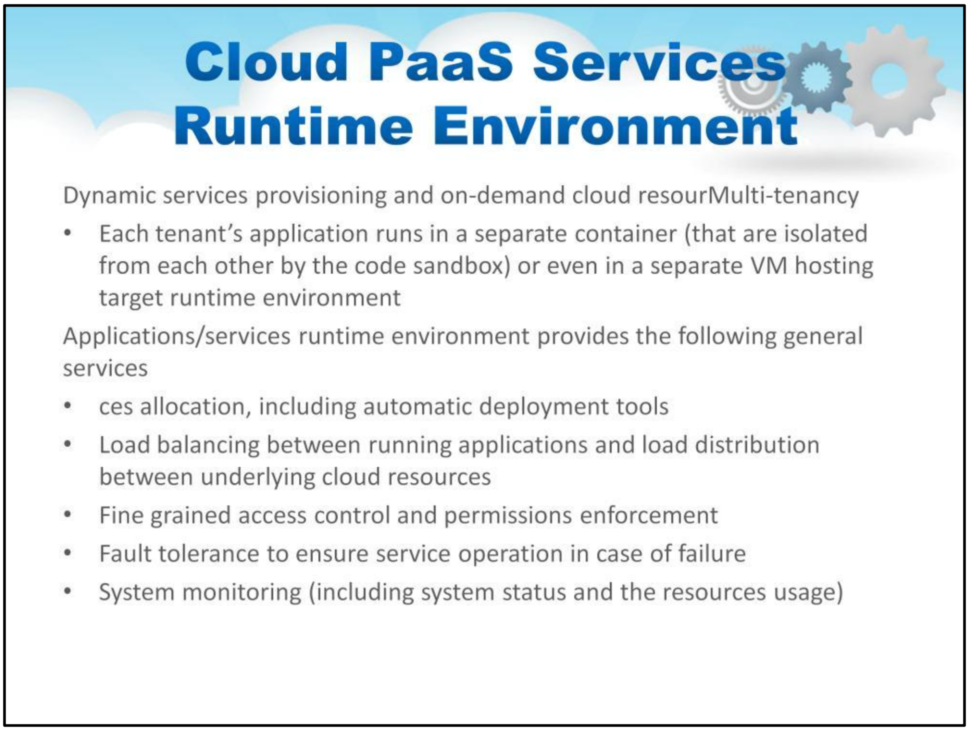
IaaS Platform (EC2, Azure, OpenStack, OpenNebula)

This slide contains an illustration of a Cloud PaaS Services Runtime Environment.

As can be seen, the IaaS platform is at the base. There are other facilties not visible to the programmer making the cloud platform work, such as cloud management, Governance, Identity etc,

Then there is a Platform as a Service Runtime Framework (i.e. Cloud Foundry, RedHat OpenShift, dotNET)

Note that n the PaaS runtime framework, the multi-tenant layer occurs at an Application Container level, not at the VM level. There is no visibility to VMs to users!



Cloud PaaS Services Runtime Environment

Dynamic services provisioning and on-demand cloud resourMulti-tenancy

- Each tenant’s application runs in a separate container (that are isolated from each other by the code sandbox) or even in a separate VM hosting target runtime environment

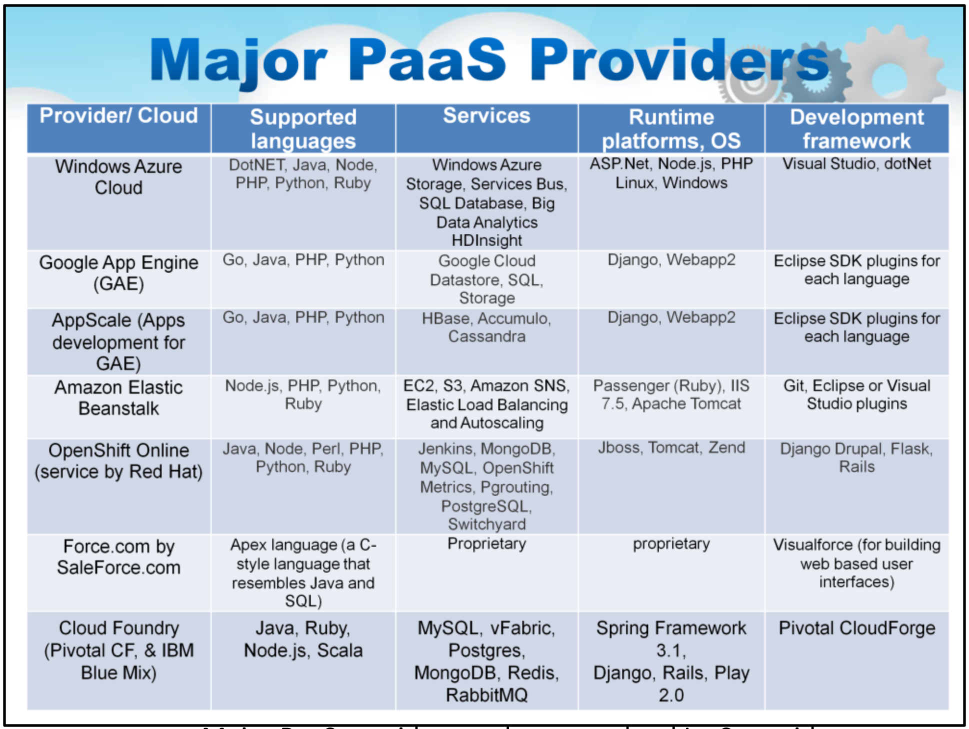
Applications/services runtime environment provides the following general services

- ces allocation, including automatic deployment tools
- Load balancing between running applications and load distribution between underlying cloud resources
- Fine grained access control and permissions enforcement
- Fault tolerance to ensure service operation in case of failure
- System monitoring (including system status and the resources usage)

As mentioned in the previous slide, each tenant’s application runs in a separate container (that are isolated from each other by the code sandbox) or even in a separate VM hosting target runtime environment

Applications/services runtime environment provides the following general services

Dynamic services provisioning including automatic deployment tools
Load balancing
Fine grained access control
Fault tolerance
System monitoring



Major PaaS Providers

Provider/ Cloud	Supported languages	Services	Runtime platforms, OS	Development framework
Windows Azure Cloud	DotNET, Java, Node, PHP, Python, Ruby	Windows Azure Storage, Services Bus, SQL Database, Big Data Analytics HDInsight	ASP.Net, Node.js, PHP Linux, Windows	Visual Studio, dotNet
Google App Engine (GAE)	Go, Java, PHP, Python	Google Cloud Datastore, SQL, Storage	Django, Webapp2	Eclipse SDK plugins for each language
AppScale (Apps development for GAE)	Go, Java, PHP, Python	HBase, Accumulo, Cassandra	Django, Webapp2	Eclipse SDK plugins for each language
Amazon Elastic Beanstalk	Node.js, PHP, Python, Ruby	EC2, S3, Amazon SNS, Elastic Load Balancing and Autoscaling	Passenger (Ruby), IIS 7.5, Apache Tomcat	Git, Eclipse or Visual Studio plugins
OpenShift Online (service by Red Hat)	Java, Node, Perl, PHP, Python, Ruby	Jenkins, MongoDB, MySQL, OpenShift Metrics, Pgrouting, PostgreSQL, Switchyard	Jboss, Tomcat, Zend	Django Drupal, Flask, Rails
Force.com by SaleForce.com	Apex language (a C-style language that resembles Java and SQL)	Proprietary	proprietary	Visualforce (for building web based user interfaces)
Cloud Foundry (Pivotal CF, & IBM Blue Mix)	Java, Ruby, Node.js, Scala	MySQL, vFabric, Postgres, MongoDB, Redis, RabbitMQ	Spring Framework 3.1, Django, Rails, Play 2.0	Pivotal CloudForge

There are not as many Major PaaS providers as there are cloud IaaS providers. Generally there are not too many mainstream programming environment “families”. There is at the highest level “.NET (Microsoft)” and “Java (or more generally speaking more open development environments)”.

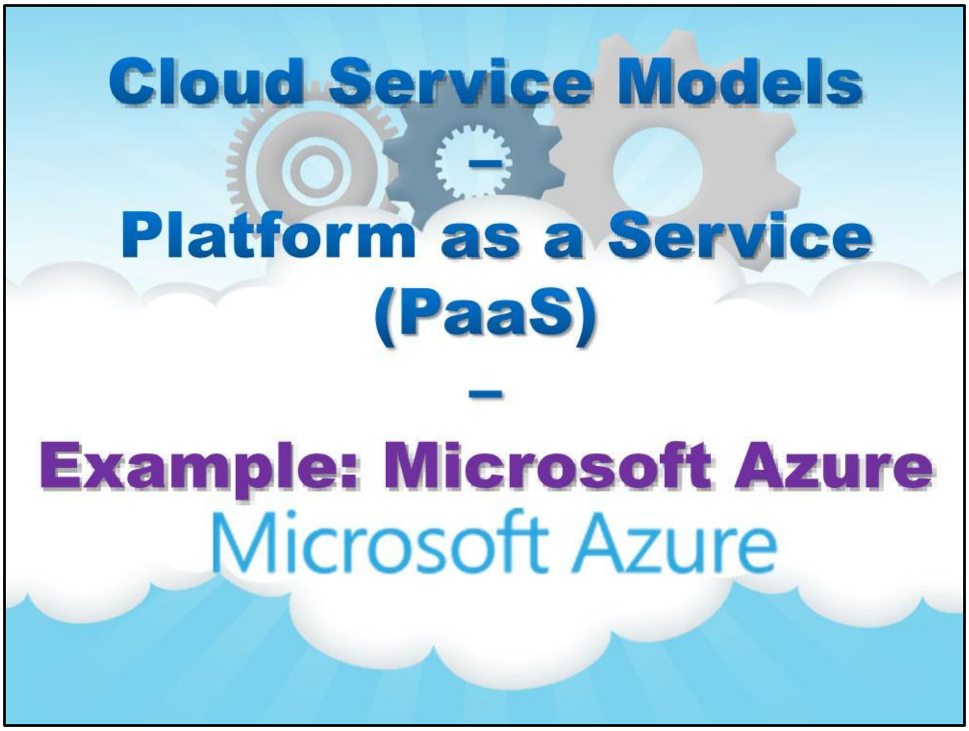
Microsoft was the unquestioned pioneer of PaaS. At the original launch of Windows Azure, there was no IaaS mode at all, it was a PaaS-pure-play offering. Microsoft was truly way ahead of the times here, believing that VM’s should never be manipulated by users, and that they could deliver a much better cloud and much more security and capability by exposing a pure code container environment. While Azure has added the IaaS capability, it remains one of the most well architected and complete PaaS environment. It is worth noting that Microsoft has excellent support for many languages including PHP, Ruby, and Java, and so thinking that Azure or .NET PaaS is strictly a C# environment is quite wrong.

Google also came out with a PaaS before they supported an IaaS offering, GAE was very very simple.

While people argue that AWS is not really a PaaS, this is a legacy perception now it contains all kinds of facilities to run code as indicated in the table

RedHat combined/re-adapted their middleware offerings into OpenShift but this has been less accepted by the community than Cloud Foundry (see below) as it seems to be very controlled by RedHat,

Salesforce,com has a “customization and extension” environment to it’s CRM applications and database which it calls Force,com.



Cloud Service Models

–

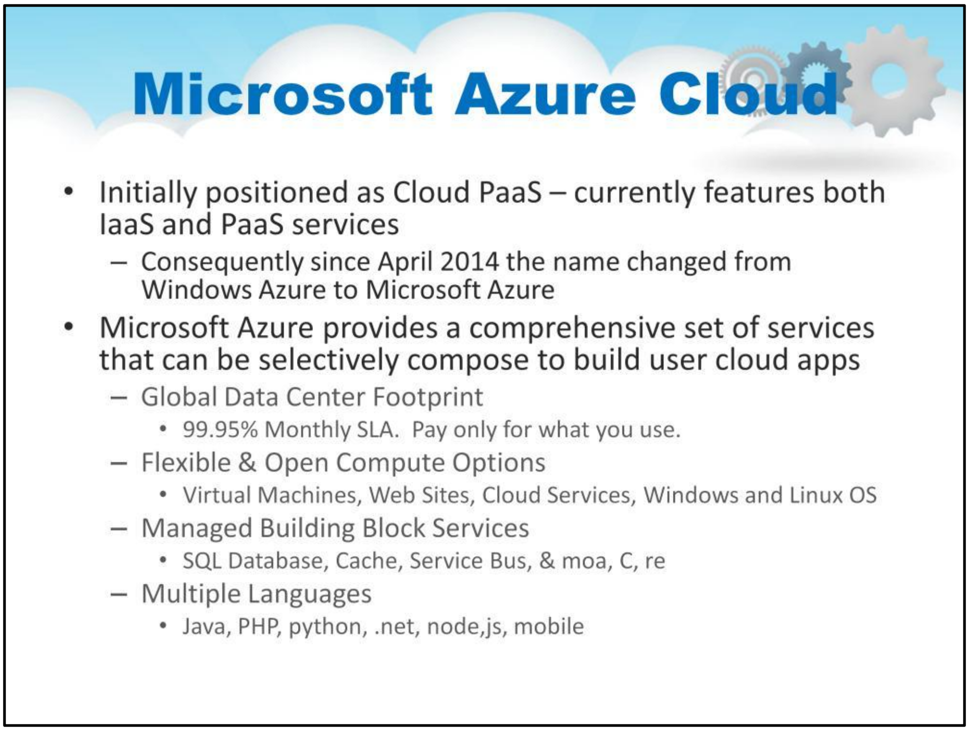
Platform as a Service (PaaS)

–

Example: Microsoft Azure

Microsoft Azure

Example: Microsoft Azure



Microsoft Azure Cloud

- Initially positioned as Cloud PaaS – currently features both IaaS and PaaS services
  - Consequently since April 2014 the name changed from Windows Azure to Microsoft Azure
- Microsoft Azure provides a comprehensive set of services that can be selectively compose to build user cloud apps
  - Global Data Center Footprint
    - 99.95% Monthly SLA. Pay only for what you use.
  - Flexible & Open Compute Options
    - Virtual Machines, Web Sites, Cloud Services, Windows and Linux OS
  - Managed Building Block Services
    - SQL Database, Cache, Service Bus, & moa, C, re
  - Multiple Languages
    - Java, PHP, python, .net, node,js, mobile

The Microsoft Azure Cloud was first released as Windows Azure and was strictly a PaaS platform aimed at providing a Cloud home to Windows developers. After adding IaaS abilities including the ability to run Linux VM’s, Azure has become a hue force in Cloud.

Microsoft Azure provides a comprehensive set of services that can be selectively compose to build user cloud apps

Global Data Center Footprint

99.95% Monthly SLA. Pay only for what you use.

Flexible & Open Compute Options

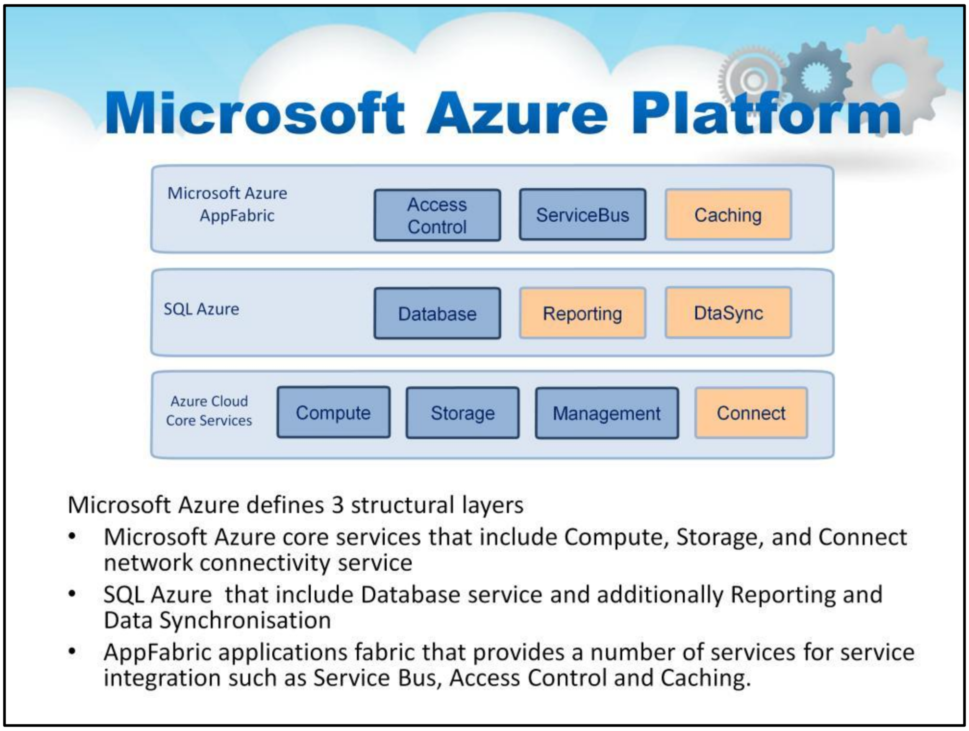
Virtual Machines, Web Sites, Cloud Services, Windows and Linux OS

Managed Building Block Services

SQL Database, Cache, Service Bus, & moa, C, re

Multiple Languages

Java, PHP, python, .net, node,js, mobile



Microsoft Azure Platform

Microsoft Azure AppFabric: Access Control, ServiceBus, Caching

SQL Azure: Database, Reporting, DtaSync

Azure Cloud Core Services: Compute, Storage, Management, Connect

Microsoft Azure defines 3 structural layers

- Microsoft Azure core services that include Compute, Storage, and Connect network connectivity service
- SQL Azure that include Database service and additionally Reporting and Data Synchronisation
- AppFabric applications fabric that provides a number of services for service integration such as Service Bus, Access Control and Caching.

The Azure platform is pretty much like other cloud platforms as to it's structure with the exception that the data and application services are very closely integrated with the underlying core cloud services.

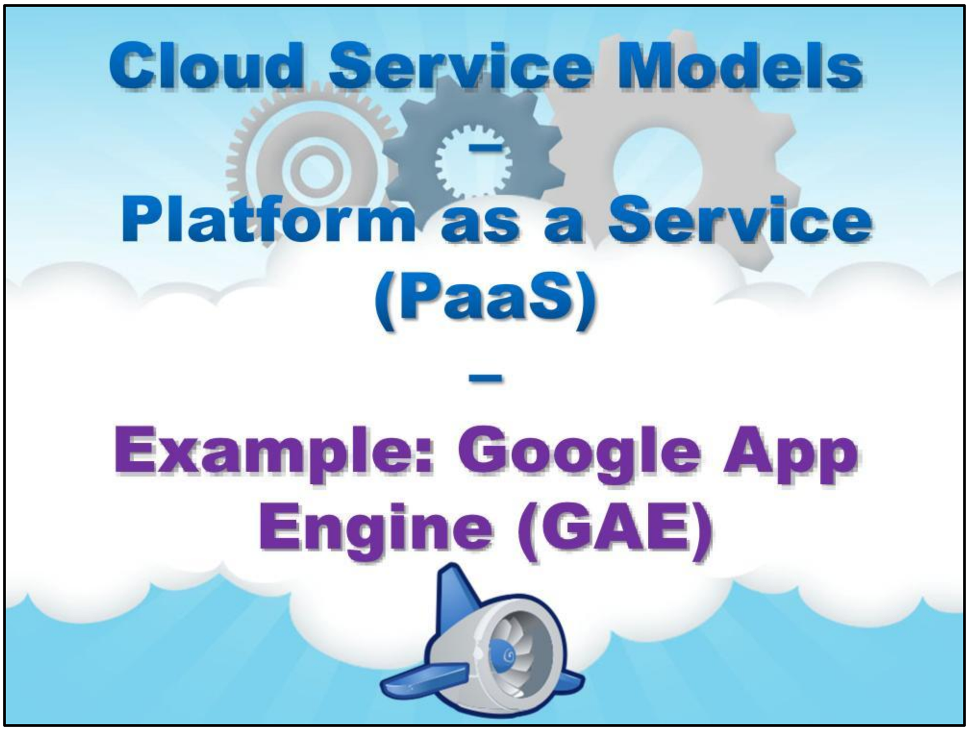
This is an artifact of how Azure ws a pure PaaS environment at the start

The slide illustrates the Microsoft Azure 3 structural layers

Microsoft Azure core services that include Compute, Storage, and Connect network connectivity service

SQL Azure that include Database service and additionally Reporting and Data Synchronisation

AppFabric applications fabric that provides a number of services for service integration such as Service Bus, Access Control and Caching.



Cloud Service Models

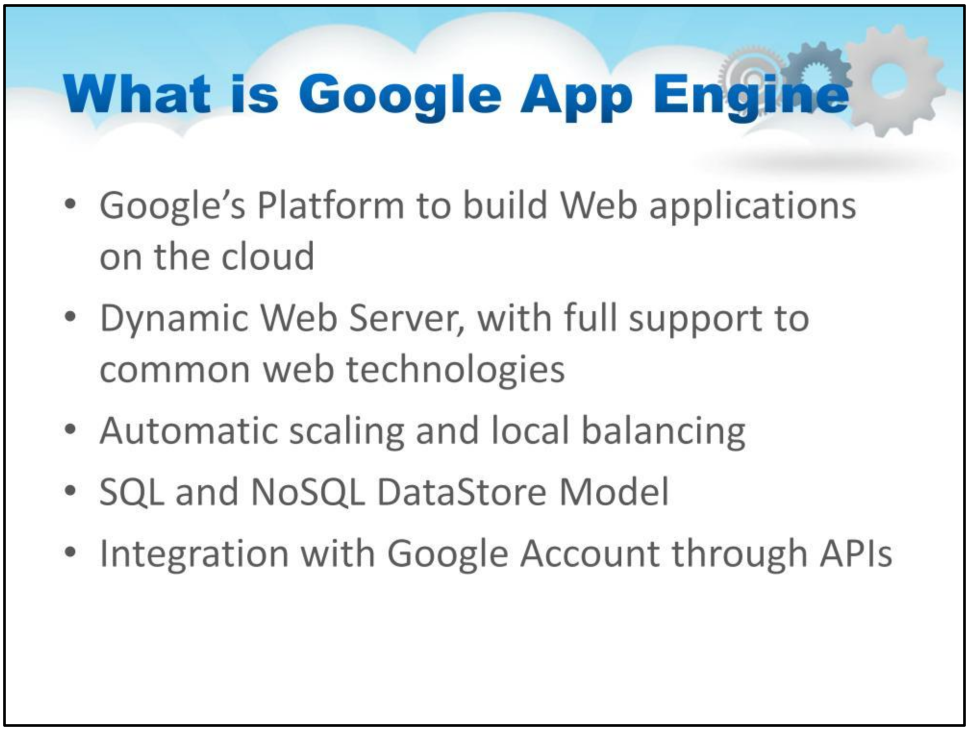
–

Platform as a Service (PaaS)

–

Example: Google App Engine (GAE)

Example: Google App Engine (GAE)



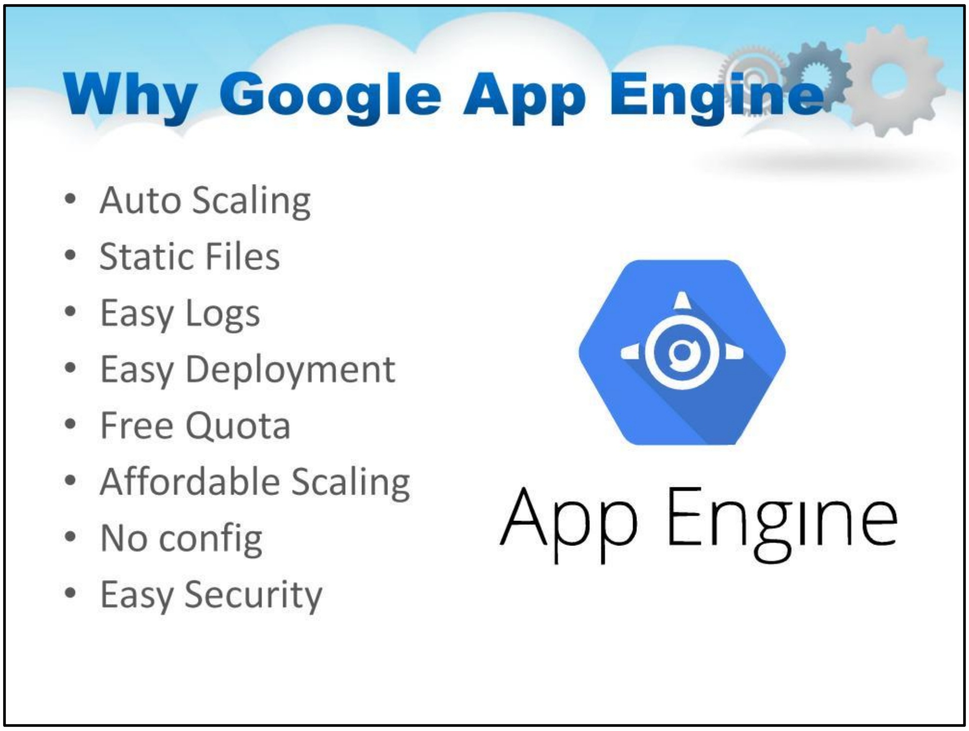
What is Google App Engine

- Google's Platform to build Web applications on the cloud
- Dynamic Web Server, with full support to common web technologies
- Automatic scaling and local balancing
- SQL and NoSQL DataStore Model
- Integration with Google Account through APIs

Google App Engine Google App Engine (often referred to as GAE or simply App Engine) is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers.

Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications— as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle the additional demand. Google App Engine is free up to a certain level of consumed resources. Fees are charged for additional storage, bandwidth, or instance hours required by the application.

It was first released as a preview version in April 2008, and came out of preview in September 2011.



Why Google App Engine

- Auto Scaling
- Static Files
- Easy Logs
- Easy Deployment
- Free Quota
- Affordable Scaling
- No config
- Easy Security

App Engine

Auto Scaling - No need to over provision

Static Files - Static files use Google’s CDN

Easy Logs - View logs in web console

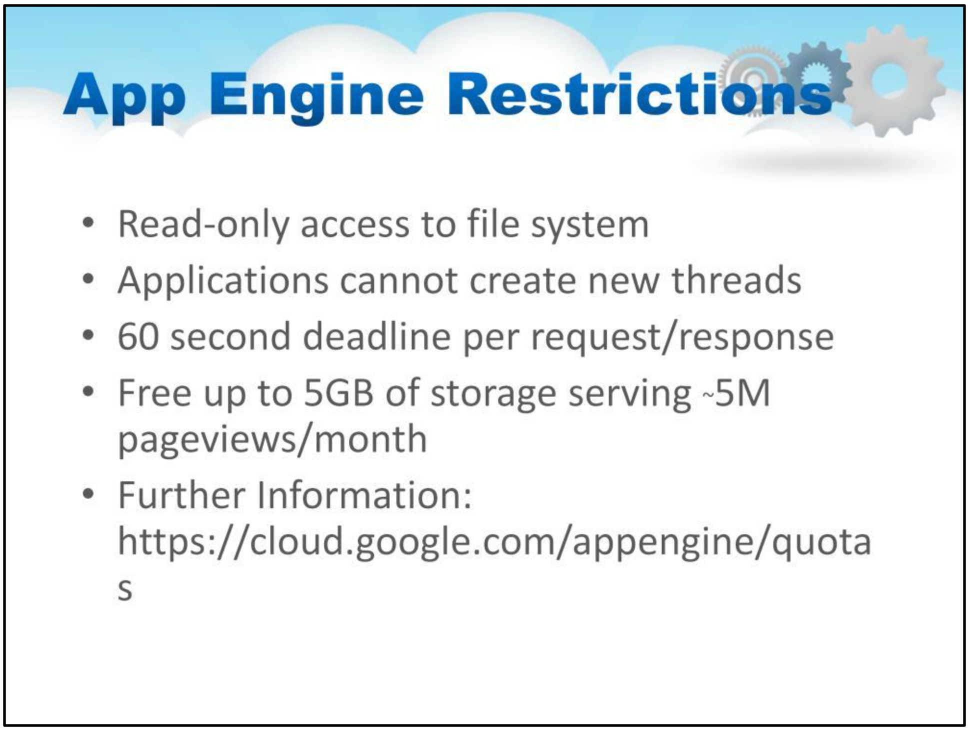
Easy Deployment - Literally 1-click deploy

Free Quota - 99% of apps will pay nothing

Affordable Scaling - Google App Engine is not cheap, but it has a very competitive cost structure considering the scalability and ease of use.

No config - No need to config OS or servers

Easy Security - The whole App Engine infrastructure is way different than “regular” hosting platforms. The application works in a small sandbox with very limited permissions. Because an app can’t write to the filesystem, it’s impossible to alter the application code from the application itself. This prevents most of the WordPress targeted attacks.



App Engine Restrictions

- Read-only access to file system
- Applications cannot create new threads
- 60 second deadline per request/response
- Free up to 5GB of storage serving ~5M pageviews/month
- Further Information: https://cloud.google.com/appengine/quotas

An App Engine application can consume resources up to certain quotas.

You can view the daily usage and quota consumption of App Engine resources for your project in the Google Cloud Platform Console Quota Details page.

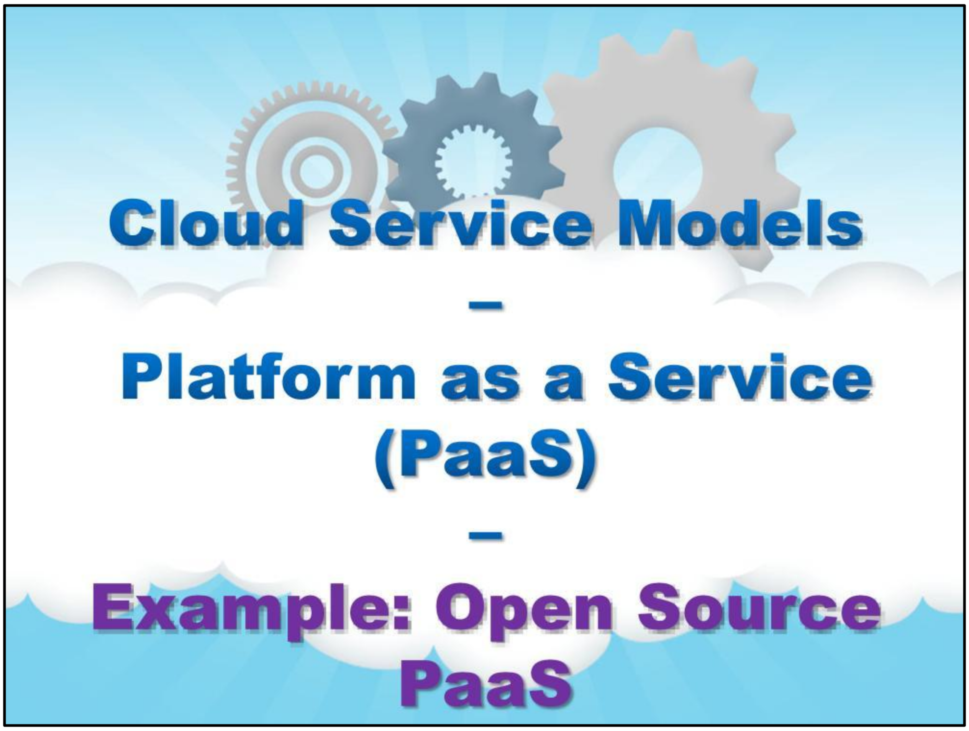
Billable limits and safety limits

App Engine has three kinds of quotas or limits:

Free quotas: Every application gets an amount of each resource for free. Free quotas can only be exceeded by paid applications, up to the application's spending limit or the safety limit, whichever applies first.

Spending limits: If you are the project owner and the billing administrator, you can set the spending limit to manage application costs in the Google Cloud Platform Console in the App Engine Settings. Spending limits might be exceeded slightly as the application is disabled.

Safety limits: Safety limits are set by Google to protect the integrity of the App Engine system. These quotas ensure that no single app can over-consume resources to the detriment of other apps. If you go above these limits you'll get an error whether you are paid or free.



Cloud Service Models

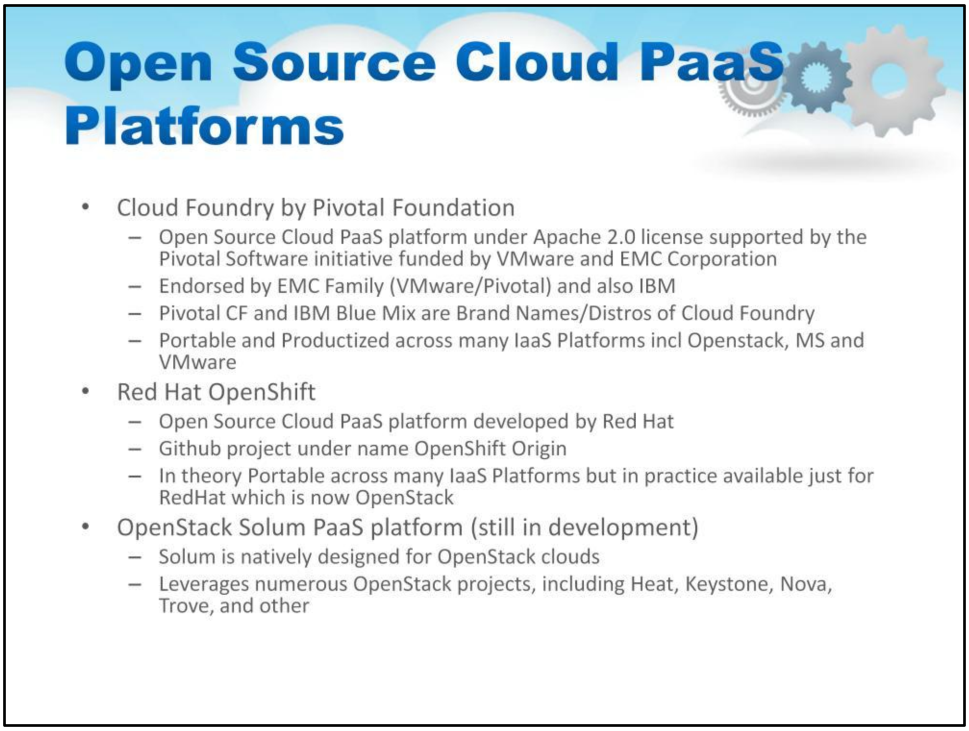
–

Platform as a Service (PaaS)

–

Example: Open Source PaaS

Example: Open Source PaaS



Open Source Cloud PaaS Platforms

- Cloud Foundry by Pivotal Foundation
  - Open Source Cloud PaaS platform under Apache 2.0 license supported by the Pivotal Software initiative funded by VMware and EMC Corporation
  - Endorsed by EMC Family (VMware/Pivotal) and also IBM
  - Pivotal CF and IBM Blue Mix are Brand Names/Distros of Cloud Foundry
  - Portable and Productized across many IaaS Platforms incl Openstack, MS and VMware
- Red Hat OpenShift
  - Open Source Cloud PaaS platform developed by Red Hat
  - Github project under name OpenShift Origin
  - In theory Portable across many IaaS Platforms but in practice available just for RedHat which is now OpenStack
- OpenStack Solum PaaS platform (still in development)
  - Solum is natively designed for OpenStack clouds
  - Leverages numerous OpenStack projects, including Heat, Keystone, Nova, Trove, and other

There is a “battle” going on regarding which Open Source PaaS platform will be most widely accepted

Cloud Foundry was initially assembled from Spring and Hibernate, and several components were added, developed both in Open Source and by VMware, and productized by the VMware/EMC spin out Pivotal. Pivotal launched a foundation created by VMware and EMC with initial endorsement by IBM. Pivotal CF and IBM Bluemix are Brand Names/Distros of Cloud Foundry. Immediately the Pivotal Foudation gained more support from business oriented IT companies such as IBM, HP and SAP. It appears Pivotal is becoming the “new J2EE”

RedHat had put together OpenShift, which came from it’s extensive middleware product line (all open source). While gaining initial momentum, OpenShift places RedHat at odds with it’s typically largest partner IBM. The future of OpenShift is in question.

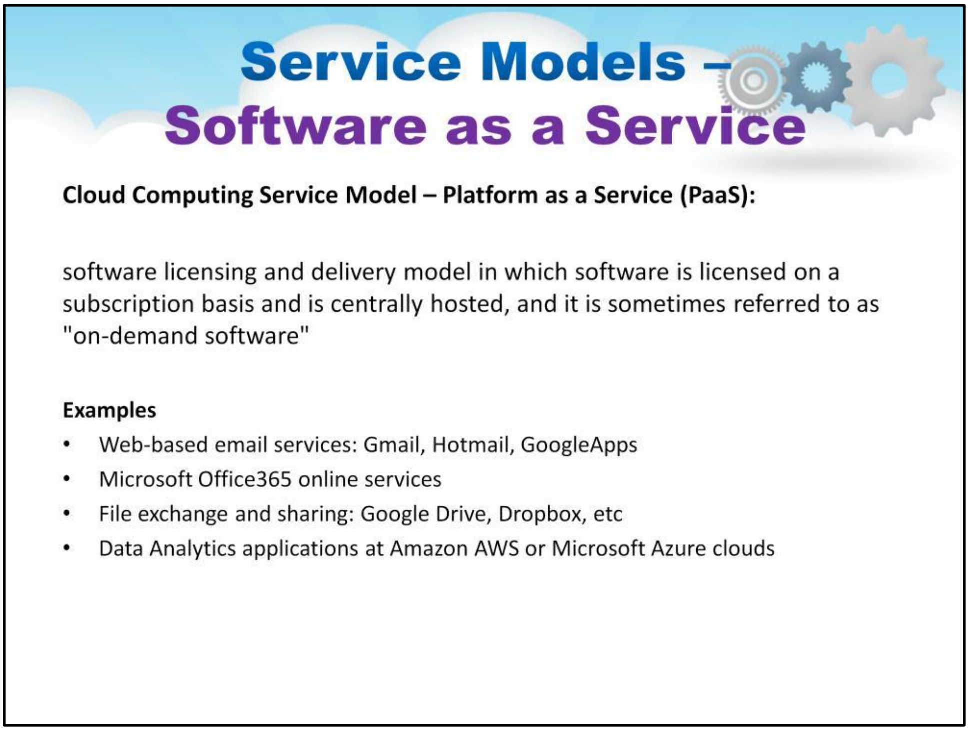
OpenStack Solum is still in development.

Cloud Service Models

–

Software as a Service (SaaS)

Software as a Service (SaaS)



Service Models – Software as a Service

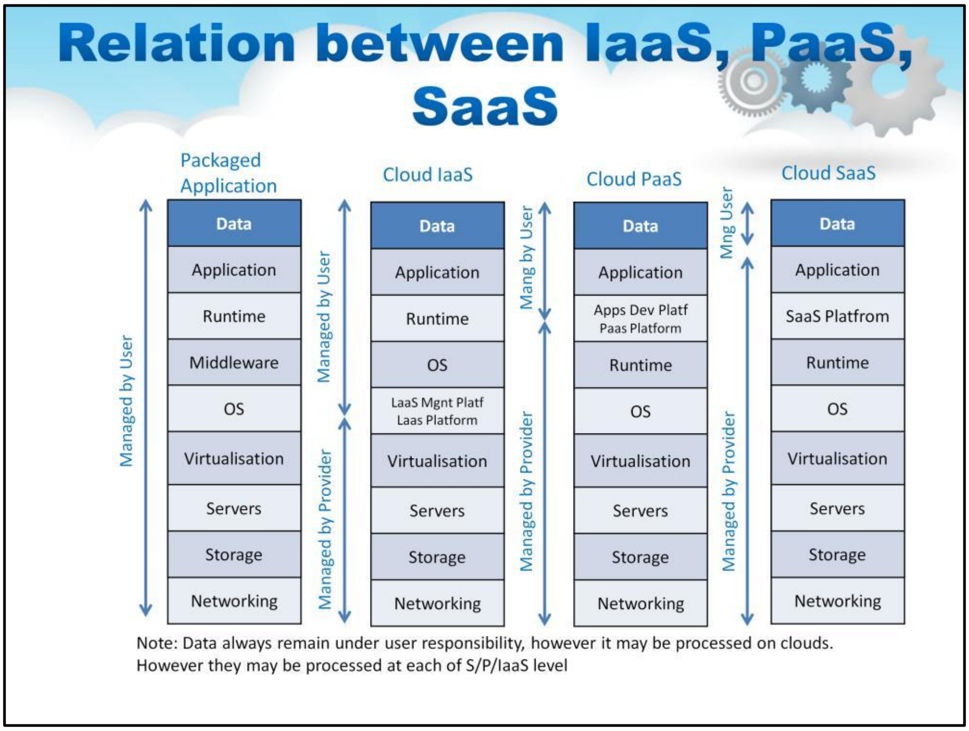
Cloud Computing Service Model – Platform as a Service (PaaS):

software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted, and it is sometimes referred to as "on-demand software"

Examples

- Web-based email services: Gmail, Hotmail, GoogleApps
- Microsoft Office365 online services
- File exchange and sharing: Google Drive, Dropbox, etc
- Data Analytics applications at Amazon AWS or Microsoft Azure clouds

The capability provided to the consumer is to use the provider’s applications running *on a cloud infrastructure*. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of *limited user-specific application configuration settings*.



Relation between IaaS, PaaS, SaaS

Packaged Application	Cloud IaaS	Cloud PaaS	Cloud SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Runtime	Runtime	Apps Dev Platf Paas Platform	SaaS Platfrom
Middleware	OS	Runtime	Runtime
OS	LaaS Mgnt Platf Laas Platform	OS	OS
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Note: Data always remain under user responsibility, however it may be processed on clouds. However they may be processed at each of S/P/IaaS level

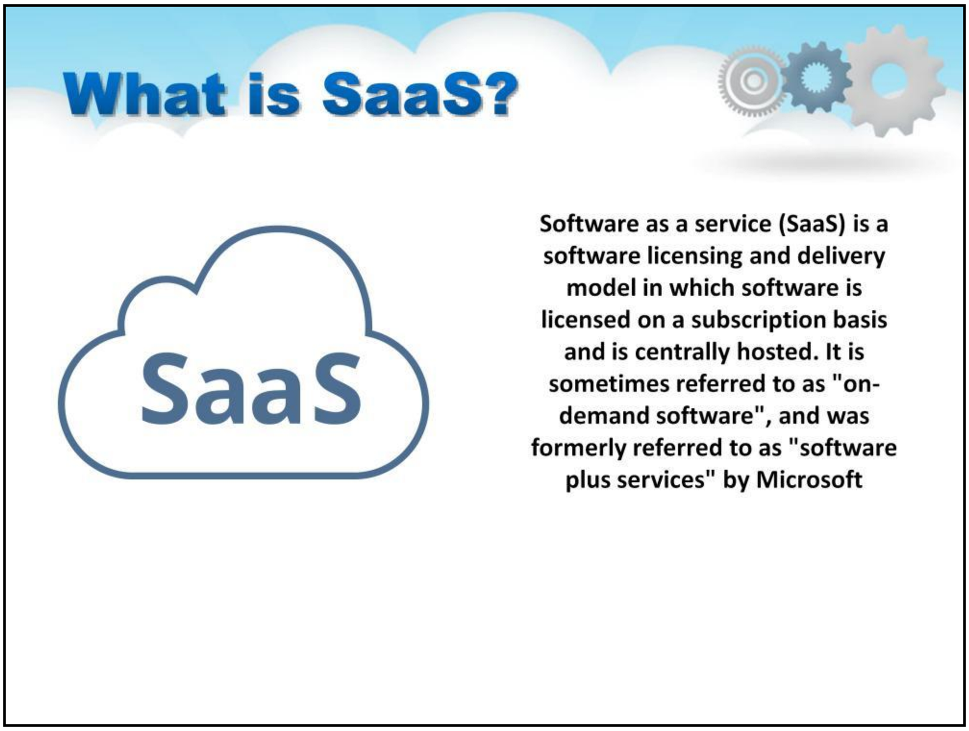
The use models for Cloud Computing are closely related. This illustration has been covered in an earlier Lesson, but to re-iterate.

For a traditional “physically installed” packaged application, the entire deployment stack is managed by the user, as can be seen in the first block of the illustration. The user is responsible for provisioning the complete server including networking and storage, and the Operating System software as well. Additionally, the user provisions whatever databases and middleware they need, any special runtime (Java or Ruby), and finally the application. The application data is also the responsibility (and the ownership of) the user.

The Cloud IaaS model changes this significantly, alleviating a large portion of at least what would be considered “physical” infrastructure, this is shown in the second block of the illustration. The cloud management (sometimes called Cloud OS) software is shown in this diagram as “IaaS Mngt Platf”. The user obtains the hardware in a virtualized form from the IaaS CSP and worries only about the software stack, application, and data.

The Cloud PaaS model further offers middleware, application runtime, and as much “software” infrastructure as possible, creating a convenient container for the application developer. While this requires the developer to restructure if not rewrite the application at the source code level to interface to the PaaS, this restricting will impart new scalability and availability strengths into the application.

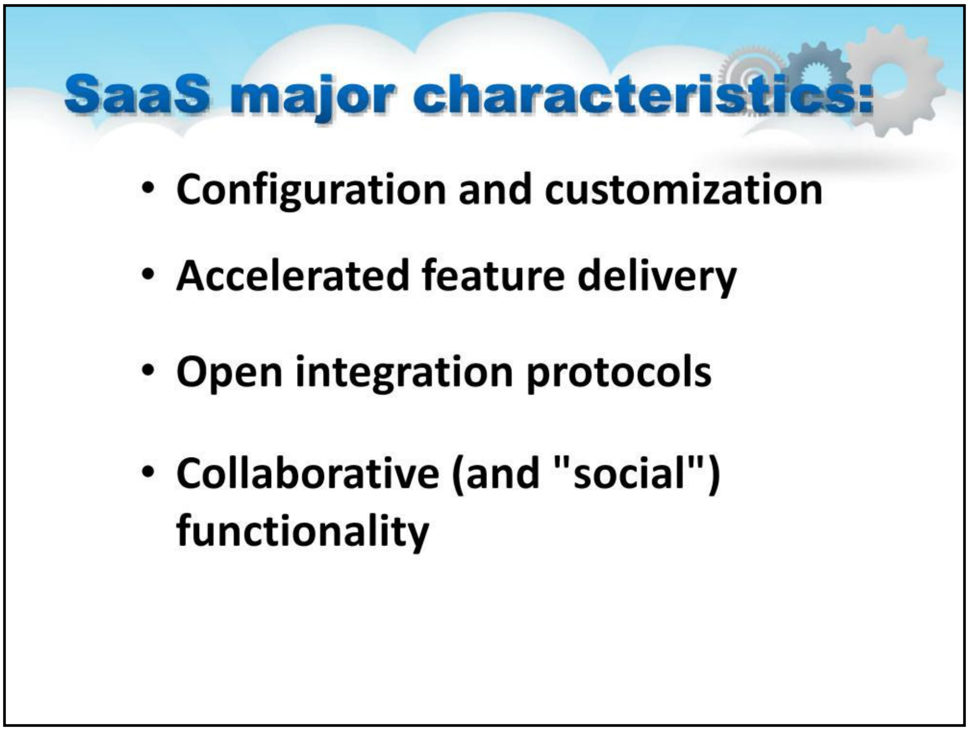
The Cloud IaaS model is a consuming of just the application itself through a browser or web service.



What is SaaS?

Software as a service (SaaS) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. It is sometimes referred to as "on-demand software", and was formerly referred to as "software plus services" by Microsoft

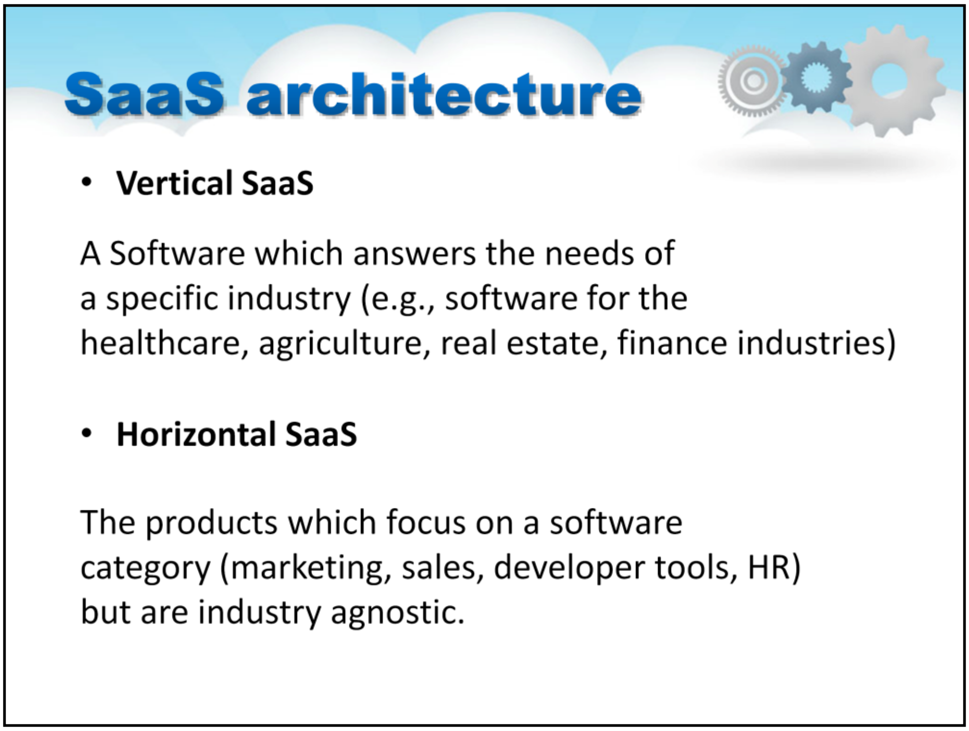
Software as a service (SaaS) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. It is sometimes referred to as "on-demand software", and was formerly referred to as "software plus services" by Microsoft. SaaS is typically accessed by users using a thin client via a web browser. SaaS has become a common delivery model for many business applications, including office software, messaging software, payroll processing software, DBMS software, management software, CAD software, development software, gamification, virtualization, accounting, collaboration, customer relationship management (CRM), Management Information Systems (MIS), enterprise resource planning (ERP), invoicing, human resource management (HRM), talent acquisition, learning management systems, content management (CM), and service desk management. SaaS has been incorporated into the strategy of nearly all leading enterprise software companies.



SaaS major characteristics:

- Configuration and customization
- Accelerated feature delivery
- Open integration protocols
- Collaborative (and "social") functionality

Configuration and customization
SaaS applications similarly support what is traditionally known as application configuration.
Accelerated feature delivery
SaaS applications are often updated more frequently than traditional software, in many cases on a weekly or monthly basis
Open integration protocols
Because SaaS applications cannot access a company's internal systems (databases or internal services), they predominantly offer integration protocols and application programming interfaces (APIs) that operate over a wide area network. Typically, these are protocols based on HTTP, REST and SOAP.
Collaborative (and "social") functionality
Inspired by the success of online social networks and other so-called web 2.0 functionality, many SaaS applications offer features that let their users collaborate and share information.



SaaS architecture

- **Vertical SaaS**

A Software which answers the needs of a specific industry (e.g., software for the healthcare, agriculture, real estate, finance industries)

- **Horizontal SaaS**

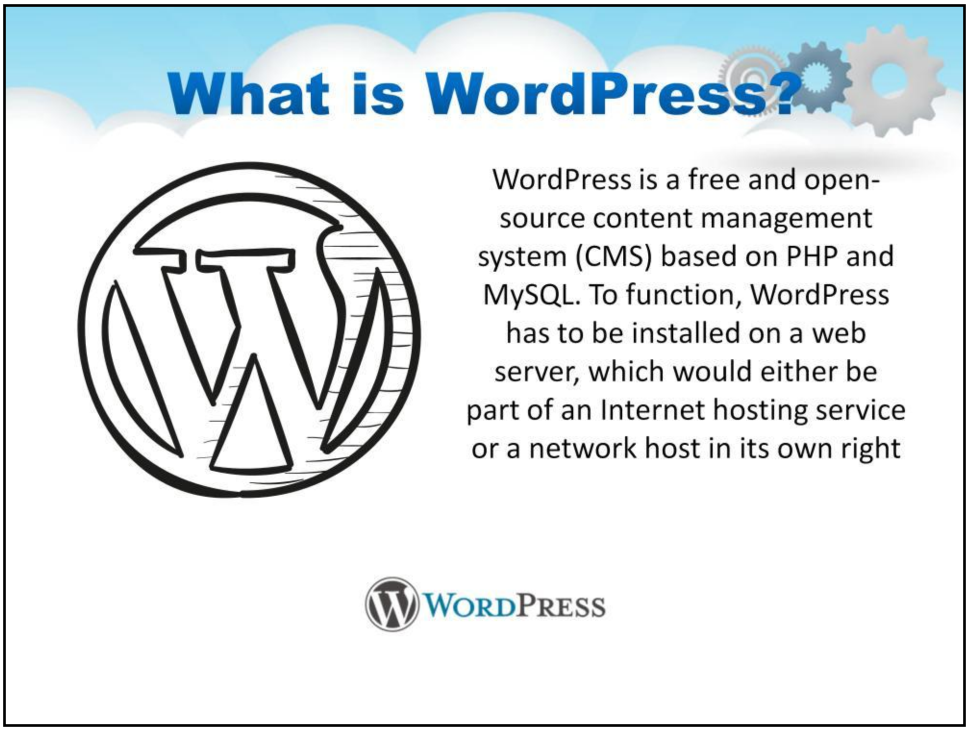
The products which focus on a software category (marketing, sales, developer tools, HR) but are industry agnostic.

The vast majority of SaaS solutions are based on a multitenant architecture. With this model, a single version of the application, with a single configuration (hardware, network, operating system), is used for all customers ("tenants"). To support scalability, the application is installed on multiple machines (called horizontal scaling). In some cases, a second version of the application is set up to offer a select group of customers access to pre-release versions of the applications (e.g., a beta version) for testing purposes. This is contrasted with traditional software, where multiple physical copies of the software — each potentially of a different version, with a potentially different configuration, and often customized — are installed across various customer sites. In this traditional model, each version of the application is based on a unique code

There are two main varieties of SaaS(current slide)

Cloud Service Models
–
Software as a Service (SaaS)
–
Example: WordPress

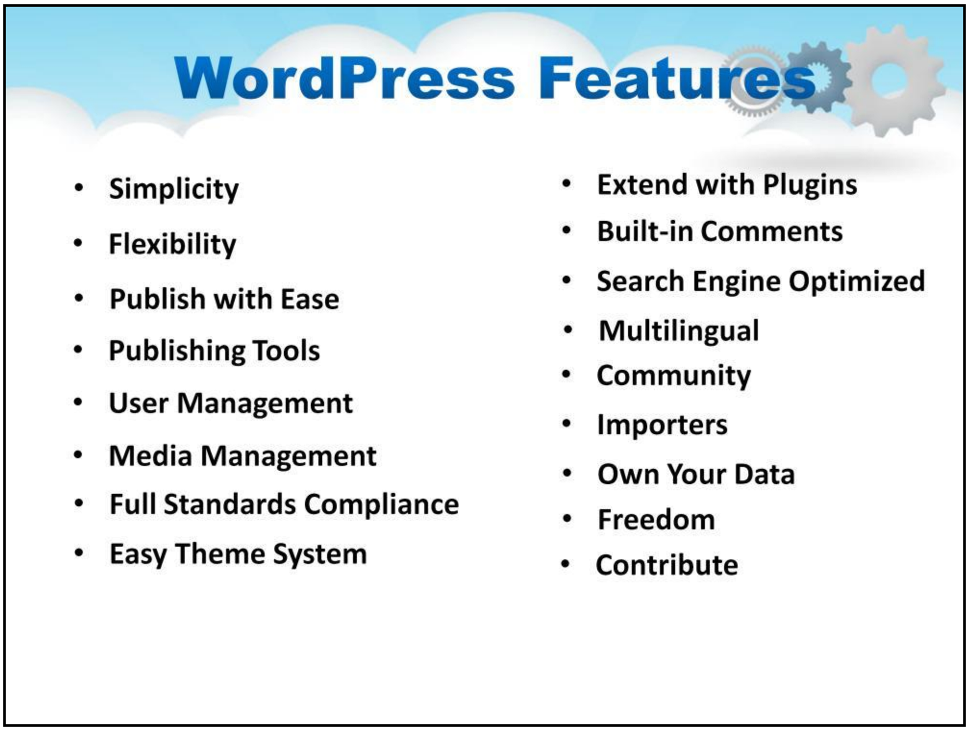
Example: WordPress



What is WordPress?

WordPress is a free and open-source content management system (CMS) based on PHP and MySQL. To function, WordPress has to be installed on a web server, which would either be part of an Internet hosting service or a network host in its own right

WordPress is a free and open-source content management system (CMS) based on PHP and MySQL. To function, WordPress has to be installed on a web server, which would either be part of an Internet hosting service or a network host in its own right. An example of the first scenario may be a service like WordPress.com, and the second case could be a computer running the software package WordPress.org. A local computer may be used for single-user testing and learning purposes. Features include a plugin architecture and a template system. WordPress is used by 30.6% of the top 10 million websites as of April 2018. As such, WordPress is the most popular website management or blogging system in use on the Web, supporting more than 60 million websites. WordPress has also been used for other application domains such as pervasive display systems (PDS).



WordPress Features

- Simplicity
- Flexibility
- Publish with Ease
- Publishing Tools
- User Management
- Media Management
- Full Standards Compliance
- Easy Theme System
- Extend with Plugins
- Built-in Comments
- Search Engine Optimized
- Multilingual
- Community
- Importers
- Own Your Data
- Freedom
- Contribute

Simplicity

Simplicity makes it possible for you to get online and get publishing, quickly. Nothing should get in the way of you getting your website up and your content out there.

Flexibility

With WordPress, you can create any type of website you want: a personal blog or website, a photoblog, a business website, a professional portfolio, a government website, a magazine or news website, an online community, even a network of websites. You can make your website beautiful with themes, and extend it with plugins. You can even build your very own application.

Publish with Ease

If you've ever created a document, you're already a whizz at creating content with WordPress. You can create Posts and Pages, format them easily, insert media, and with the click of a button your content is live and on the web.

Publishing Tools

WordPress makes it easy for you to manage your content. Create drafts, schedule publication, and look at your post revisions. Make your content public or private, and secure posts and pages with a password.

User Management

Not everyone requires the same access to your website. Administrators manage the site, editors work with content, authors and contributors write that content, and subscribers have a profile that they can manage. This lets you have a variety of contributors to your website, and let others simply be part of your community.

Media Management

They say a picture says a thousand words, which is why it's important for you to be able to quickly and easily upload images and media to WordPress. Drag and drop your media into the uploader to add it to your website. Add alt text, captions, and titles, and insert images and galleries into your content. We've even added a few image editing tools you can have fun with.

Full Standards Compliance

Every piece of WordPress generated code is in full compliance with the standards set by the W3C. This means that your website will work in today's browser, while maintaining forward compatibility with the next generation of browser. Your website is a beautiful thing, now and in the future.

Easy Theme System

WordPress comes bundled with two default themes, but if they aren't for you there's a theme directory with thousands of themes for you to create a beautiful website. None of those to your taste? Upload your own theme with the click of a button. It only takes a few seconds for you to give your website a complete makeover.

Extend with Plugins

WordPress comes packed full of features for every user, for every other feature there's a plugin directory with thousands of plugins. Add complex galleries, social networking, forums, social media widgets, spam protection, calendars, fine-tune controls for search engine optimization, and forms.

Built-in Comments

Your blog is your home, and comments provide a space for your friends and followers to engage with your content. WordPress's comment tools give you everything you need to be a forum for discussion and to moderate that discussion.

Search Engine Optimized

WordPress is optimized for search engines right out of the box. For more fine-grained SEO control, there are plenty of SEO plugins to take care of that for you.

Multilingual

WordPress is available in more than 70 languages. If you or the person you're building the website for would prefer to use WordPress in a language other than English, that's easy to do.

Easy Installation and Upgrades

WordPress has always been easy to install and upgrade. If you're happy using an FTP program, you can create a database, upload WordPress using FTP, and run the installer.

Own Your Data

Hosted services come and go. If you've ever used a service that disappeared, you know how traumatic that can be. If you've ever seen adverts appear on your website, you've probably been pretty annoyed. Using WordPress means no one has access to your content. Own your data, all of it — your website, your content, your data.

Freedom

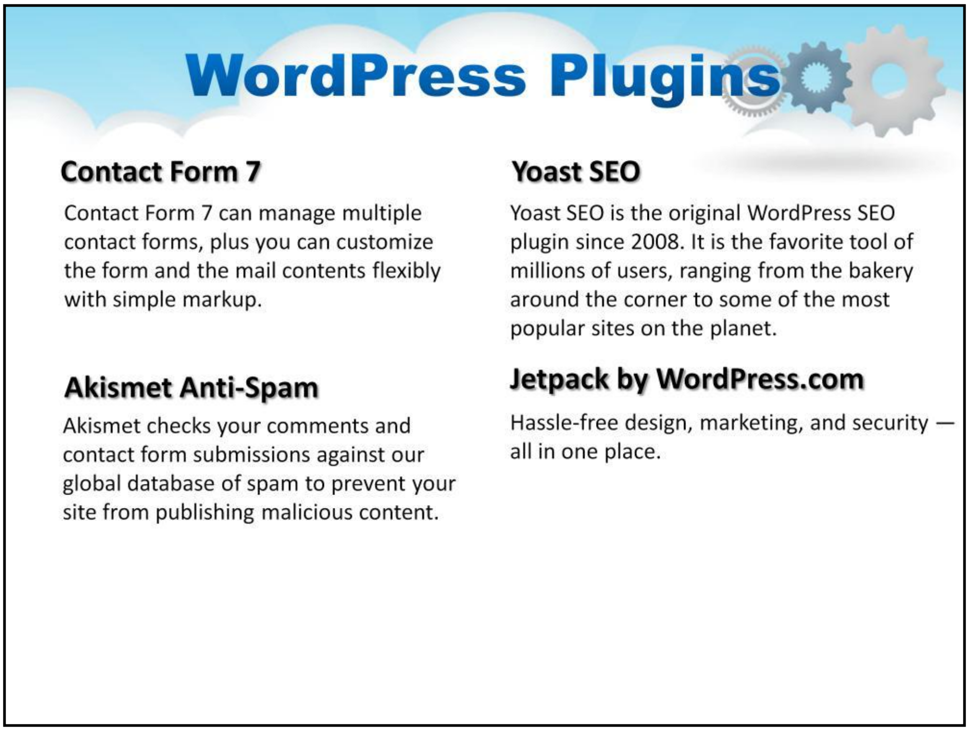
WordPress is licensed under the GPL which was created to protect your freedoms. You are free to use WordPress in any way you choose: install it, use it, modify it, distribute it. Software freedom is the foundation that WordPress is built on.

Community

As the most popular open source CMS on the web, WordPress has a vibrant and supportive community. Ask a question on the support forums and get help from a volunteer, attend a WordCamp or Meetup to learn more about WordPress, read blogs posts and tutorials about WordPress. Community is at the heart of WordPress, making it what it is today.

Contribute

You can be WordPress too! Help to build WordPress, answer questions on the support forums, write documentation, translate WordPress into your language, speak at a WordCamp, write about WordPress on your blog. Whatever your skill, we'd love to have you!



WordPress Plugins

Contact Form 7

Contact Form 7 can manage multiple contact forms, plus you can customize the form and the mail contents flexibly with simple markup.

Akismet Anti-Spam

Akismet checks your comments and contact form submissions against our global database of spam to prevent your site from publishing malicious content.

Yoast SEO

Yoast SEO is the original WordPress SEO plugin since 2008. It is the favorite tool of millions of users, ranging from the bakery around the corner to some of the most popular sites on the planet.

Jetpack by WordPress.com

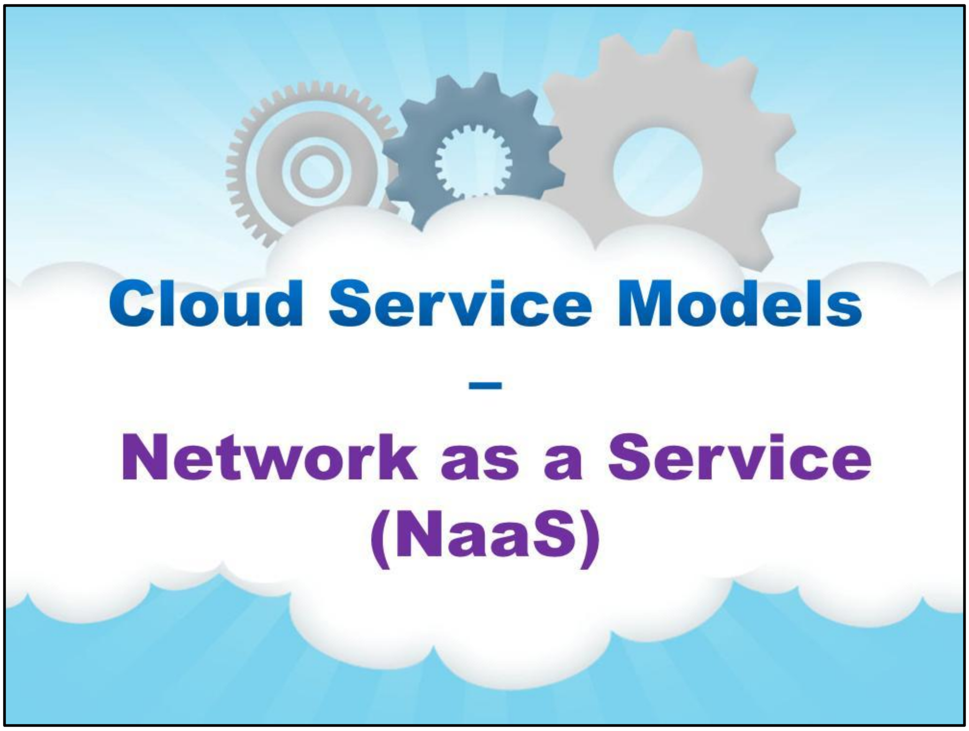
Hassle-free design, marketing, and security — all in one place.

Contact Form 7 can manage multiple contact forms, plus you can customize the form and the mail contents flexibly with simple markup. The form supports Ajax-powered submitting, CAPTCHA, Akismet spam filtering and so on.

Yoast SEO is the favorite tool of millions of users, ranging from the bakery around the corner to some of the most popular sites on the planet. With Yoast SEO, you get a solid toolset that helps you aim for that number one spot in the search results. Yoast: SEO for everyone. Yoast SEO does everything in its power to please both visitors and search engine spiders.

Akismet checks your comments and contact form submissions against our global database of spam to prevent your site from publishing malicious content. You can review the comment spam it catches on your blog's "Comments" admin screen. Major features in Akismet include: Automatically checks all comments and filters out the ones that look like spam. Each comment has a status history, so you can easily see which comments were caught or cleared by Akismet and which were spammed or unspammed by a moderator. URLs are shown in the comment body to reveal hidden or misleading links. Moderators can see the number of approved comments for each user. A discard feature that outright blocks the worst spam, saving you disk space and speeding up your site.

Jetpack by WordPress.com
Hassle-free design, marketing, and security — all in one place. Create and customize your WordPress site from start to finish. Jetpack helps you with: Hundreds of professional themes for any kind of site Intuitive and powerful customization tools Unlimited and high-speed image and video content delivery network Lazy image loading for a faster mobile experience Integration with the official WordPress mobile apps

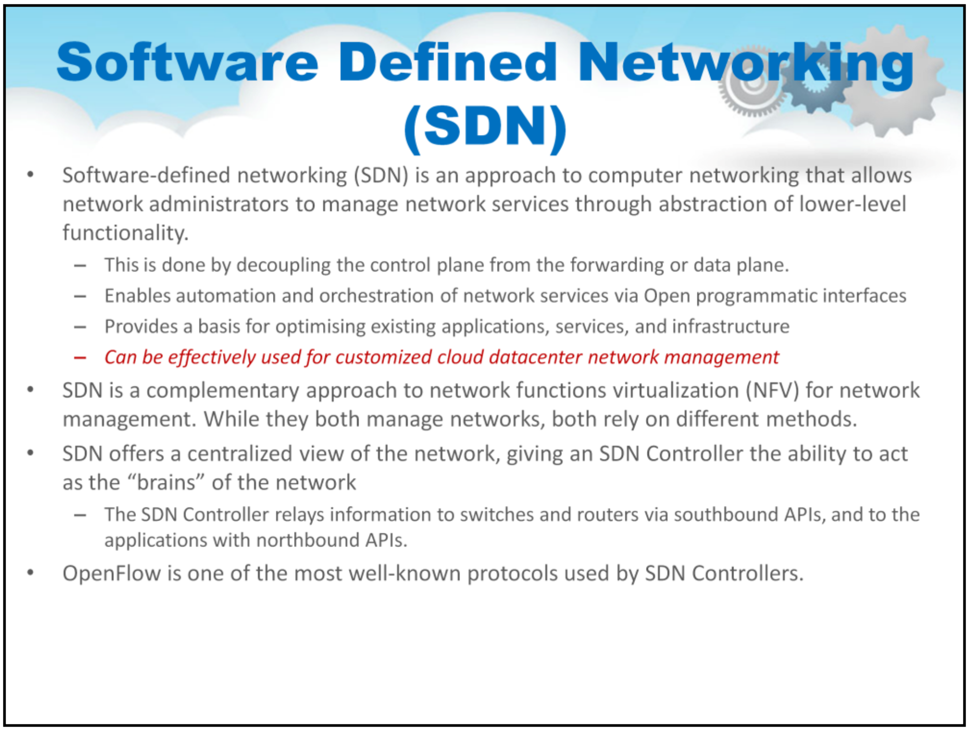


Cloud Service Models

–

Network as a Service (NaaS)

Network as a Service (NaaS)



Software Defined Networking (SDN)

- Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality.
  - This is done by decoupling the control plane from the forwarding or data plane.
  - Enables automation and orchestration of network services via Open programmatic interfaces
  - Provides a basis for optimising existing applications, services, and infrastructure
  - *Can be effectively used for customized cloud datacenter network management*
- SDN is a complementary approach to network functions virtualization (NFV) for network management. While they both manage networks, both rely on different methods.
- SDN offers a centralized view of the network, giving an SDN Controller the ability to act as the “brains” of the network
  - The SDN Controller relays information to switches and routers via southbound APIs, and to the applications with northbound APIs.
- OpenFlow is one of the most well-known protocols used by SDN Controllers.

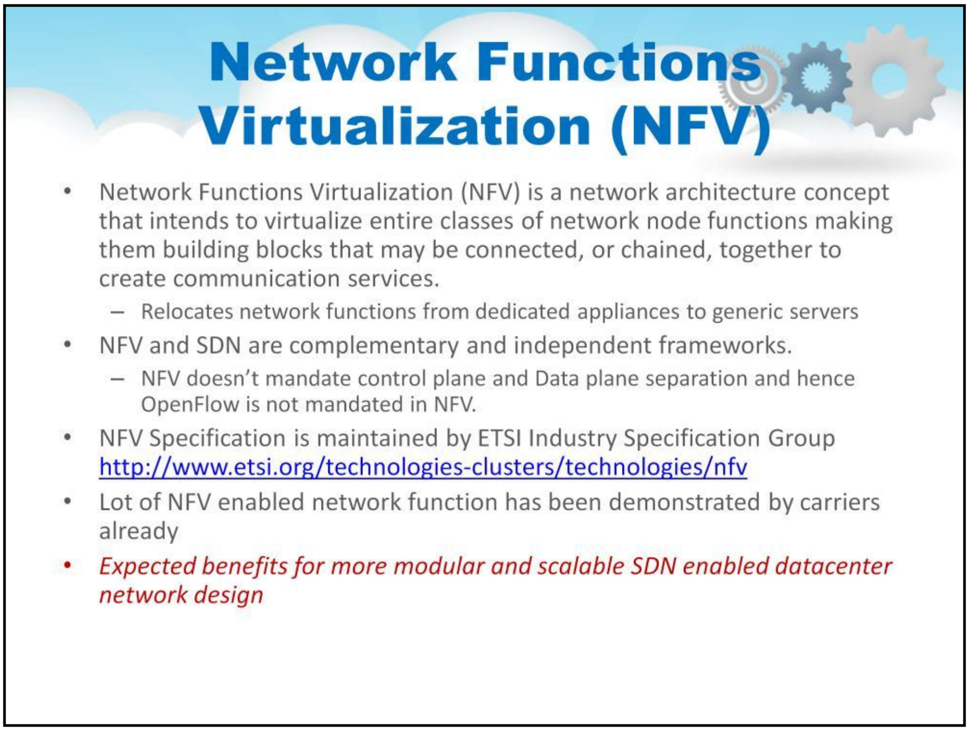
Software Defined Networking (SDN)

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality.

This is done by decoupling the control plane from the forwarding or data plane.
Enables automation and orchestration of network services via Open programmatic interfaces
Provides a basis for optimising existing applications, services, and infrastructure

SDN is a complementary approach to network functions virtualization (NFV) for network management. While they both manage networks, both rely on different methods.

SDN offers a centralized view of the network, giving an SDN Controller the ability to act as the “brains” of the network



Network Functions Virtualization (NFV)

- Network Functions Virtualization (NFV) is a network architecture concept that intends to virtualize entire classes of network node functions making them building blocks that may be connected, or chained, together to create communication services.
  - Relocates network functions from dedicated appliances to generic servers
- NFV and SDN are complementary and independent frameworks.
  - NFV doesn't mandate control plane and Data plane separation and hence OpenFlow is not mandated in NFV.
- NFV Specification is maintained by ETSI Industry Specification Group http://www.etsi.org/technologies-clusters/technologies/nfv
- Lot of NFV enabled network function has been demonstrated by carriers already
- *Expected benefits for more modular and scalable SDN enabled datacenter network design*

Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) is a network architecture concept that intends to virtualize entire classes of network node functions making them building blocks that may be connected, or chained, together to create communication services.

Relocates network functions from dedicated appliances to generic servers

Think of Firewalls, Load Balancers, WAN Optimization, and so on.

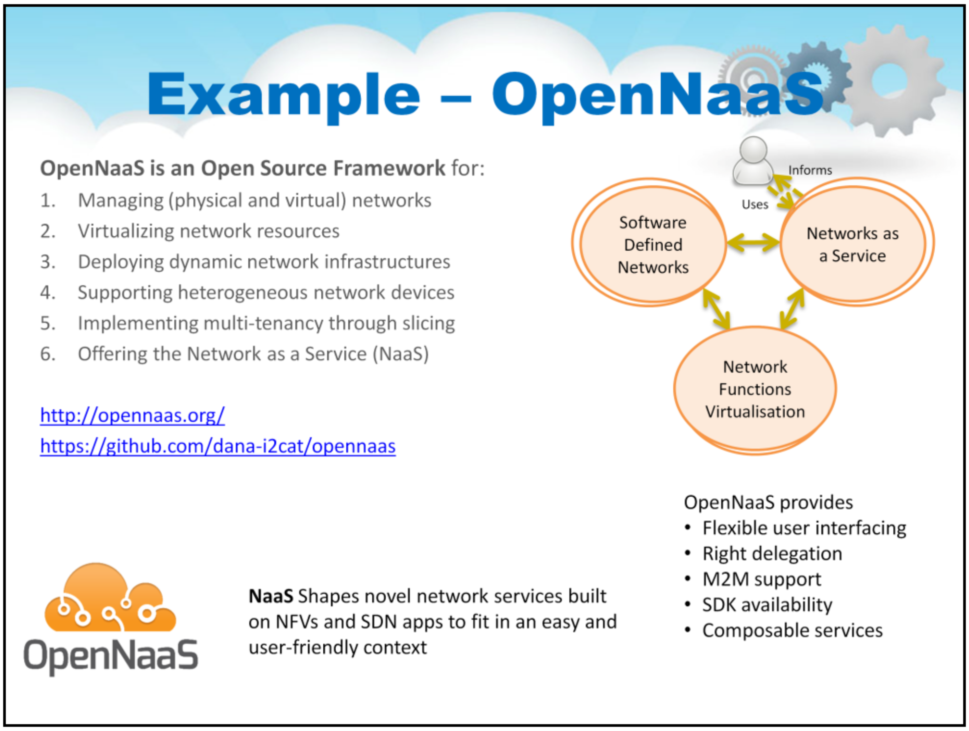
NFV also includes telecommunications functions such as IP/PBX, SMS/MMS, IP/TV, an other Telco functions usually found on dedicated appliances

NFV and SDN are complementary and independent frameworks.

NFV Specification is maintained by ETSI Industry Specification Group

Many Telecommunications functions comply with the ETSI spec when refactored into NFV

Many IT networking equipment product do not follow the ETSI spec



Example – OpenNaaS

OpenNaaS is an Open Source Framework for:

1. Managing (physical and virtual) networks
2. Virtualizing network resources
3. Deploying dynamic network infrastructures
4. Supporting heterogeneous network devices
5. Implementing multi-tenancy through slicing
6. Offering the Network as a Service (NaaS)

http://opennaas.org/
https://github.com/dana-i2cat/opennaas

Informs
Uses
Software Defined Networks
Networks as a Service
Network Functions Virtualisation

OpenNaaS provides
- Flexible user interfacing
- Right delegation
- M2M support
- SDK availability
- Composable services

OpenNaaS

NaaS Shapes novel network services built on NFVs and SDN apps to fit in an easy and user-friendly context

A closely related concept to SDN and to NFV is NaaS (“Network as a Service”)

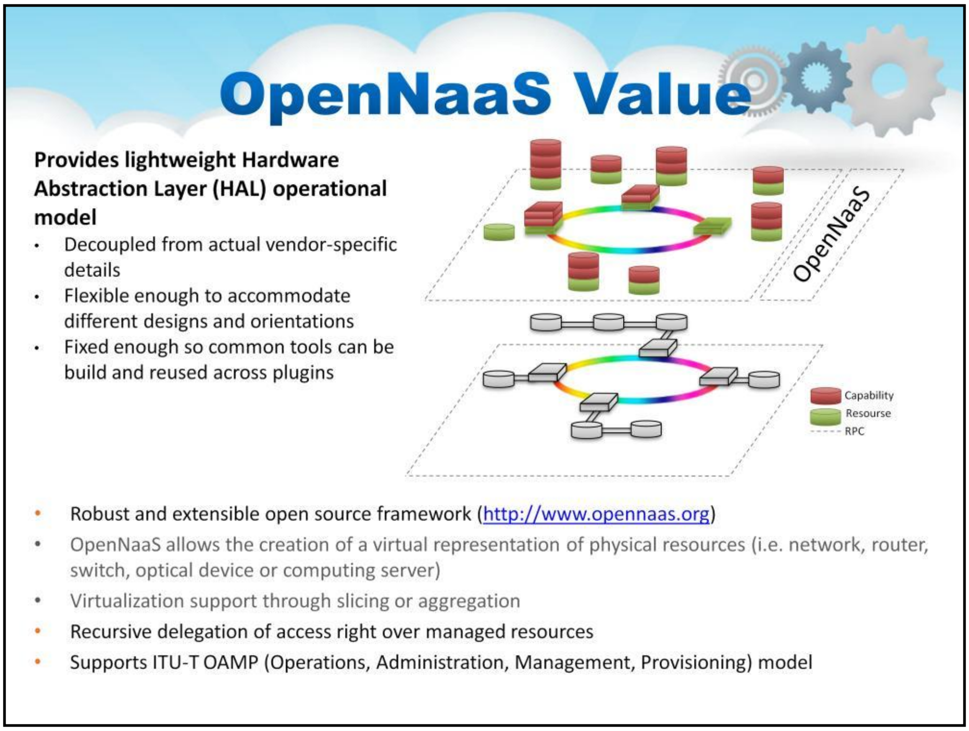
The slide contains an illustration showing this relationship.

NaaS Shapes novel network services built on VNFs and SDN apps to fit in an easy and user-friendly context

OpenNaaS is an organization promoting open source implementation of NaaS

OpenNaaS is an Open Source Framework for:

Managing (physical and virtual) networks
Virtualizing network resources
Deploying dynamic network infrastructures
Supporting heterogeneous network devices
Implementing multi-tenancy through slicing
Offering the Network as a Service (NaaS)



OpenNaaS Value

Provides lightweight Hardware Abstraction Layer (HAL) operational model

- Decoupled from actual vendor-specific details
- Flexible enough to accommodate different designs and orientations
- Fixed enough so common tools can be build and reused across plugins

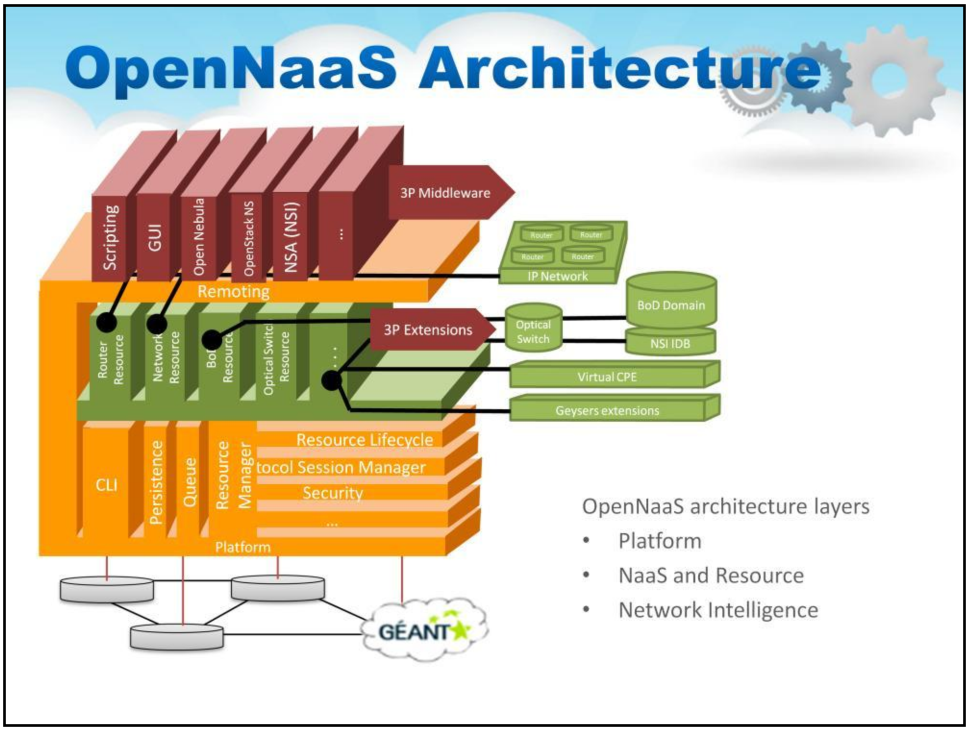
- Robust and extensible open source framework (http://www.opennaas.org)
- OpenNaaS allows the creation of a virtual representation of physical resources (i.e. network, router, switch, optical device or computing server)
- Virtualization support through slicing or aggregation
- Recursive delegation of access right over managed resources
- Supports ITU-T OAMP (Operations, Administration, Management, Provisioning) model

OpenNaaS Value

OpenNaaS Provides lightweight Hardware Abstraction Layer operational model

Decoupled from actual vendor-specific details
Flexible enough to accommodate different designs and orientations
Fixed enough so common tools can be build and reused across plugins

OpenNaaS allows the creation of a virtual representation of physical resources (i.e. network, router, switch, optical device or computing server)
Virtualization support through slicing or aggregation
Recursive delegation of access right over managed resources
Supports ITU-T OAMP (Operations, Administration, Management, Provisioning) model



OpenNaaS Architecture

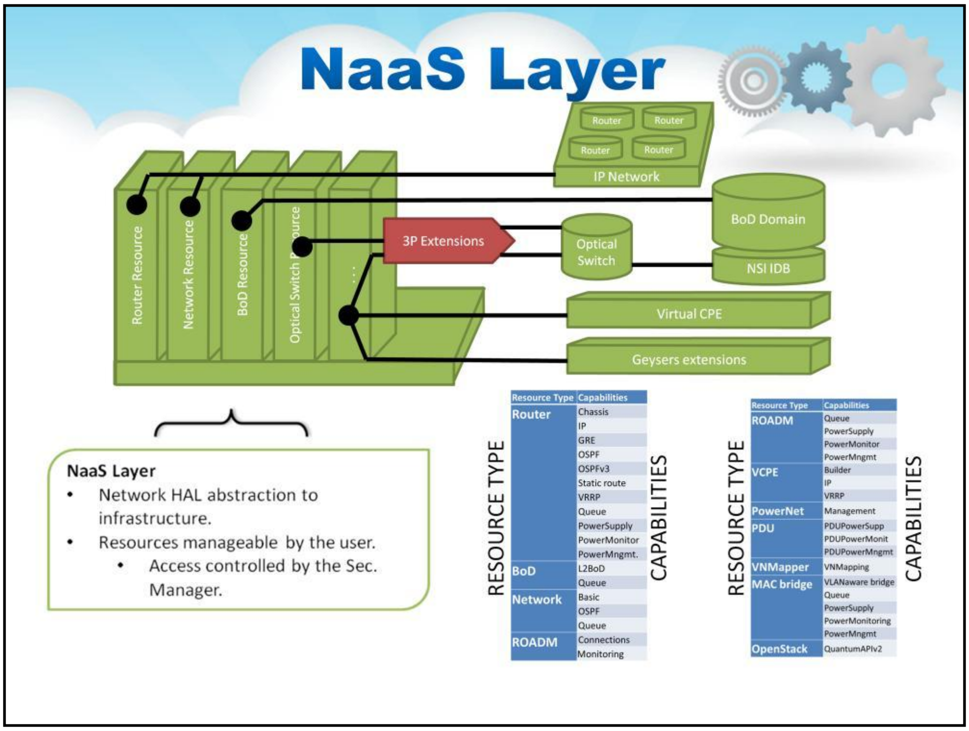
OpenNaaS architecture layers

- Platform
- NaaS and Resource
- Network Intelligence

The slide contains an illustration showing the OpenNaaS Architecture

Note the OpenNaaS architecture layers

Platform
NaaS and Resource
Network Intelligence



NaaS Layer

NaaS Layer

- Network HAL abstraction to infrastructure.
- Resources manageable by the user.
  - Access controlled by the Sec. Manager.

RESOURCE TYPE / CAPABILITIES

Resource Type	Capabilities
Router	Chassis
	IP
	GRE
	OSPF
	OSPFv3
	Static route
	VRRP
	Queue
	PowerSupply
	PowerMonitor
	PowerMngmt.
BoD	L2BoD
	Queue
Network	Basic
	OSPF
	Queue
ROADM	Connections
	Monitoring

RESOURCE TYPE / CAPABILITIES

Resource Type	Capabilities
ROADM	Queue
	PowerSupply
	PowerMonitor
	PowerMngmt
VCPE	Builder
	IP
	VRRP
PowerNet	Management
PDU	PDUPowerSupp
	PDUPowerMonit
	PDUPowerMngmt
VNMapper	VNMapping
MAC bridge	VLANaware bridge
	Queue
	PowerSupply
	PowerMonitoring
	PowerMngmt
OpenStack	QuantumAPIv2

NaaS in Detail

This layer is mainly composed by extensions leveraging abstract platform base components. It is here where support for specific resource types, capabilities, and devices are added.

Resource

Represents a manageable unit inside the NaaS concept

It can be a switch, a router, a link, a logical router, a network, etc…

Decomposed in:

A model.

An array of **Capabilities.**

Capability

An interface to a given **Resource** functionality.

For a router:

OSPF, IPv6, create/manage logical routers, etc…

This interface is, as the Model, abstracted and vendor neutral

Internally, **Capabilities** need a way to abstract implementation details of the devices, we called these **Actions**.

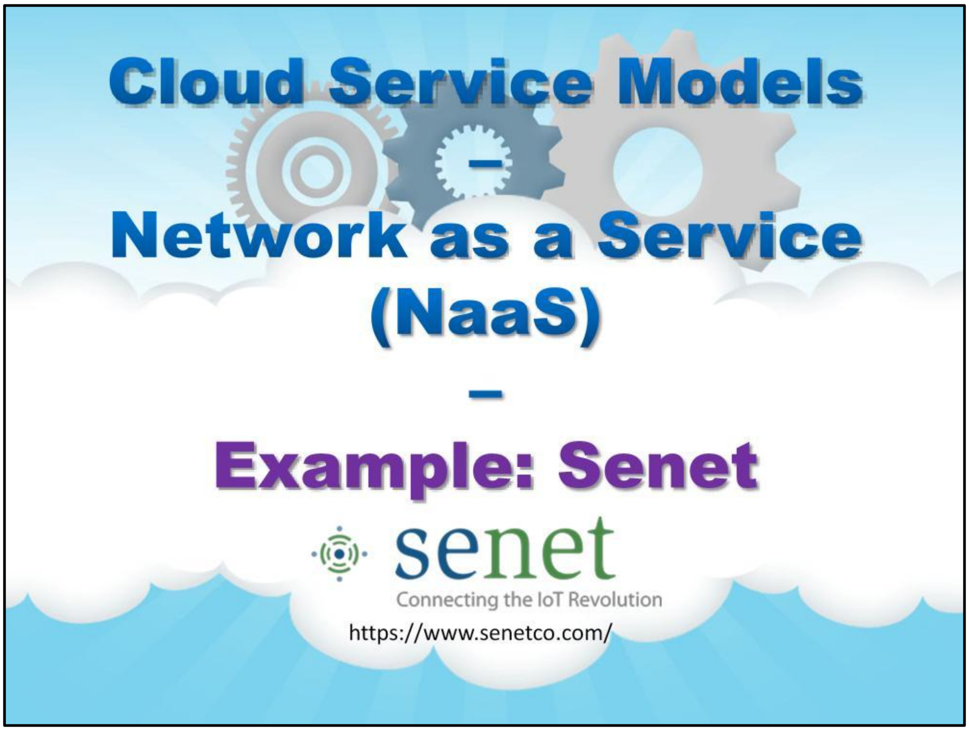
Actions

A vendor (and protocol) specific implementation of a configuration modification.

It can be Queued.

It can be undone (rollback).

The QueueManager is used to stack all **Actions** to be executed (supports rollback).



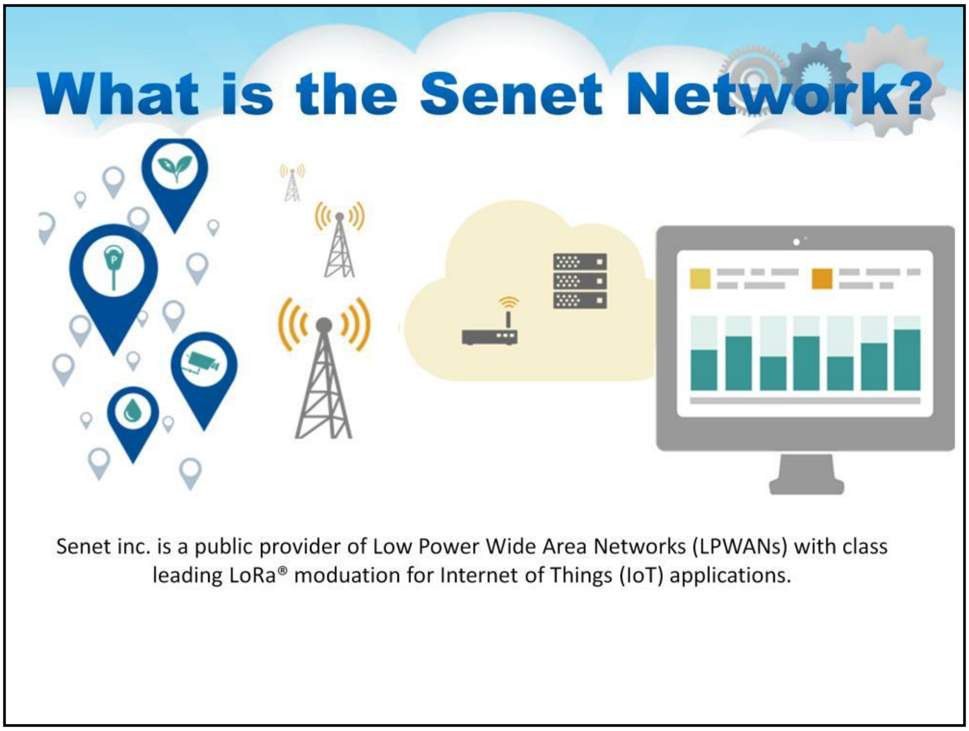
Cloud Service Models – Network as a Service (NaaS) – Example: Senet

senet

Connecting the IoT Revolution

https://www.senetco.com/

Example: Senet

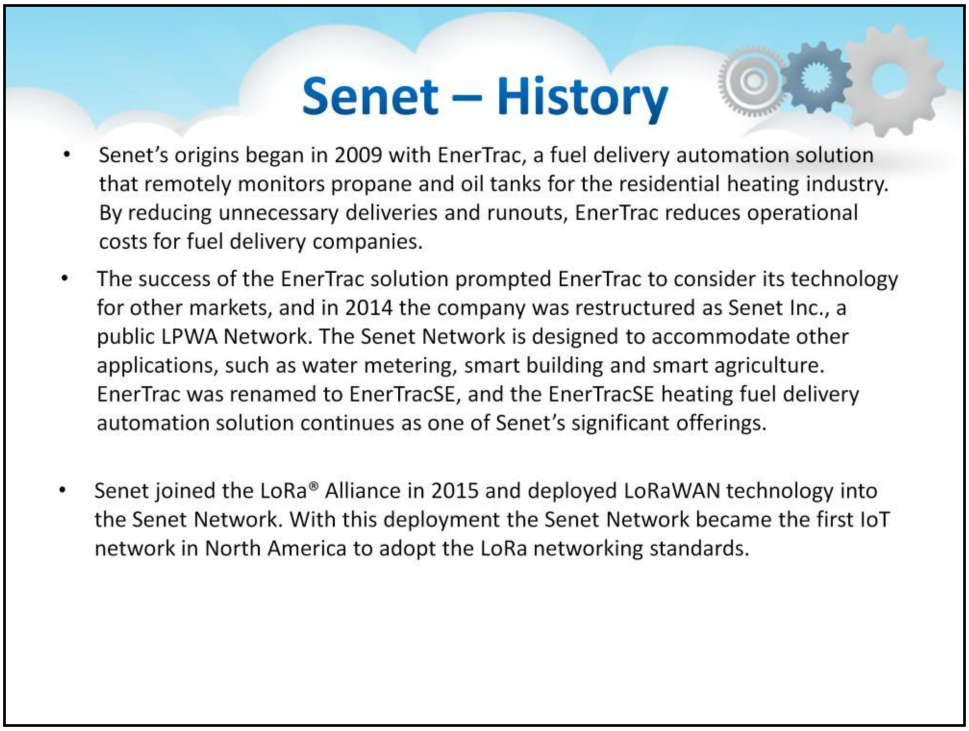


What is the Senet Network?

Senet inc. is a public provider of Low Power Wide Area Networks (LPWANs) with class leading LoRa® moduation for Internet of Things (IoT) applications.

Senet Inc. is an American Low Power Wide Area Network (LPWAN) provider for IoT/M2M applications. The Senet Network is described as "the first and only public provider of LPWA networks with class leading LoRa® modulation for IoT/M2M applications in North America”. Its platform is positioned to meet the needs of the growing “Internet of Things” (IoT) ecosystem.

The Senet Network adheres to the open global networking standards as proposed by the LoRa® Alliance, and enables low power, machine to machine (M2M) connectivity across wide geographic areas. It is designed to remove the barriers traditionally associated with managing remote devices such as reliability, range, battery life, bidirectional capability, mobility, and cost.



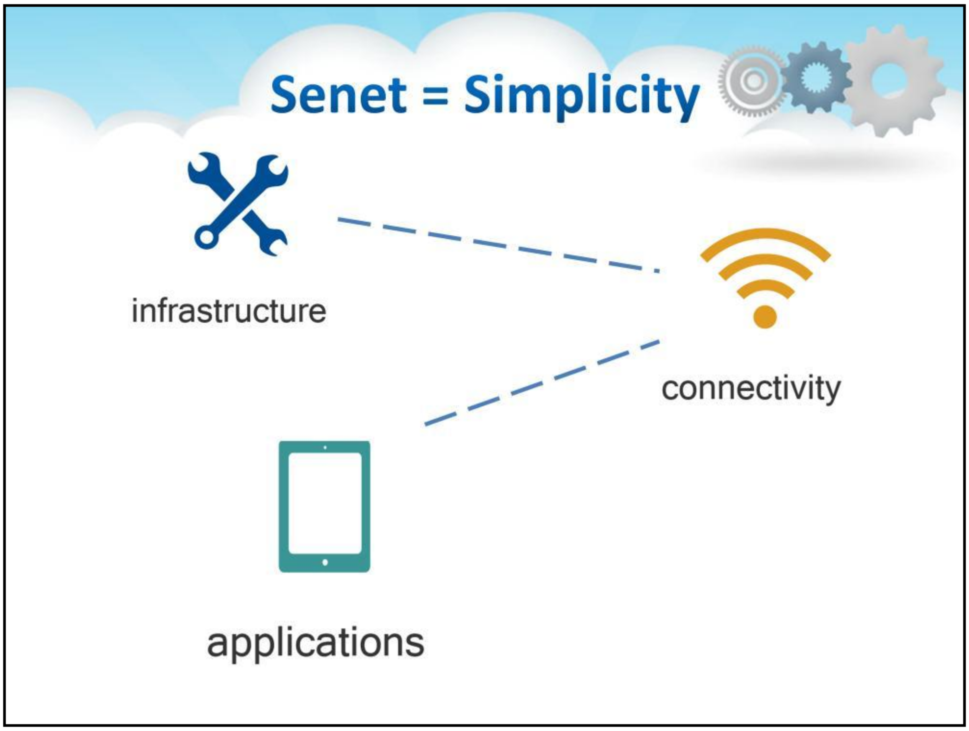
Senet – History

- Senet's origins began in 2009 with EnerTrac, a fuel delivery automation solution that remotely monitors propane and oil tanks for the residential heating industry. By reducing unnecessary deliveries and runouts, EnerTrac reduces operational costs for fuel delivery companies.
- The success of the EnerTrac solution prompted EnerTrac to consider its technology for other markets, and in 2014 the company was restructured as Senet Inc., a public LPWA Network. The Senet Network is designed to accommodate other applications, such as water metering, smart building and smart agriculture. EnerTrac was renamed to EnerTracSE, and the EnerTracSE heating fuel delivery automation solution continues as one of Senet's significant offerings.
- Senet joined the LoRa® Alliance in 2015 and deployed LoRaWAN technology into the Senet Network. With this deployment the Senet Network became the first IoT network in North America to adopt the LoRa networking standards.

Senet's origins began in 2009 with EnerTrac, a fuel delivery automation solution that remotely monitors propane and oil tanks for the residential heating industry. By reducing unnecessary deliveries and runouts, EnerTrac reduces operational costs for fuel delivery companies.

The success of the EnerTrac solution prompted EnerTrac to consider its technology for other markets, and in 2014 the company was restructured as Senet Inc., a public LPWA Network. The Senet Network is designed to accommodate other applications, such as water metering, smart building and smart agriculture. EnerTrac was renamed to EnerTracSE, and the EnerTracSE heating fuel delivery automation solution continues as one of Senet's significant offerings.

Senet joined the LoRa® Alliance in 2015 and deployed LoRaWAN technology into the Senet Network. With this deployment the Senet Network became the first IoT network in North America to adopt the LoRa networking standards.



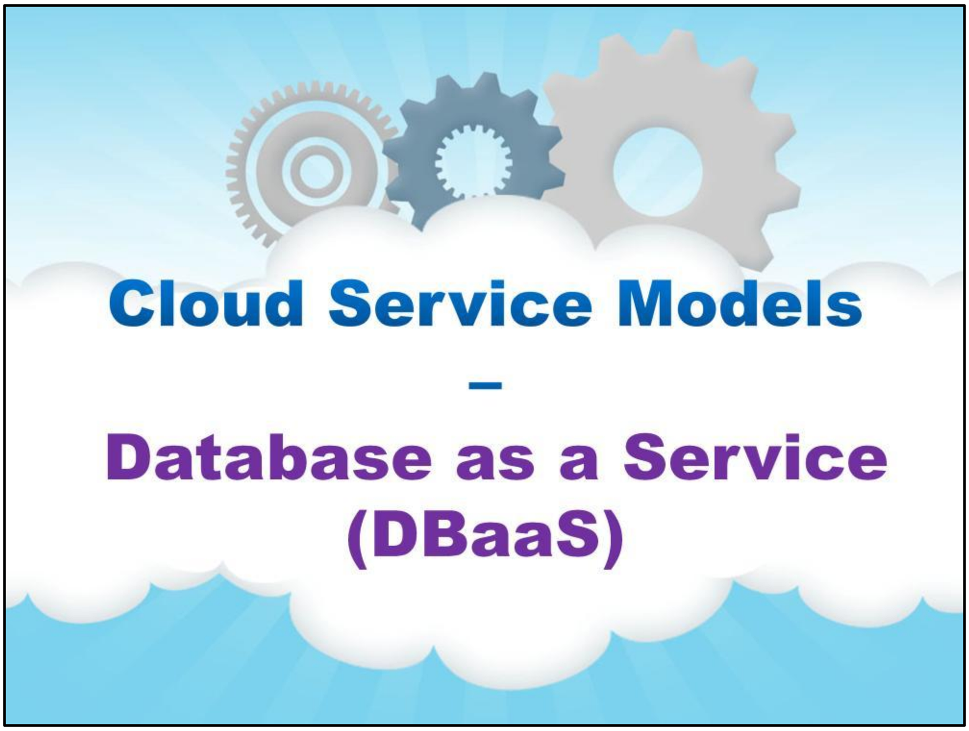
Senet = Simplicity

infrastructure

connectivity

applications

Until Senet, cost and complexity were barriers to IoT connectivity. Now, we make it easy for you to get connected with our complete, end-to-end networking and IoT platform services. All you have to do is subscribe. We’ll do the rest.

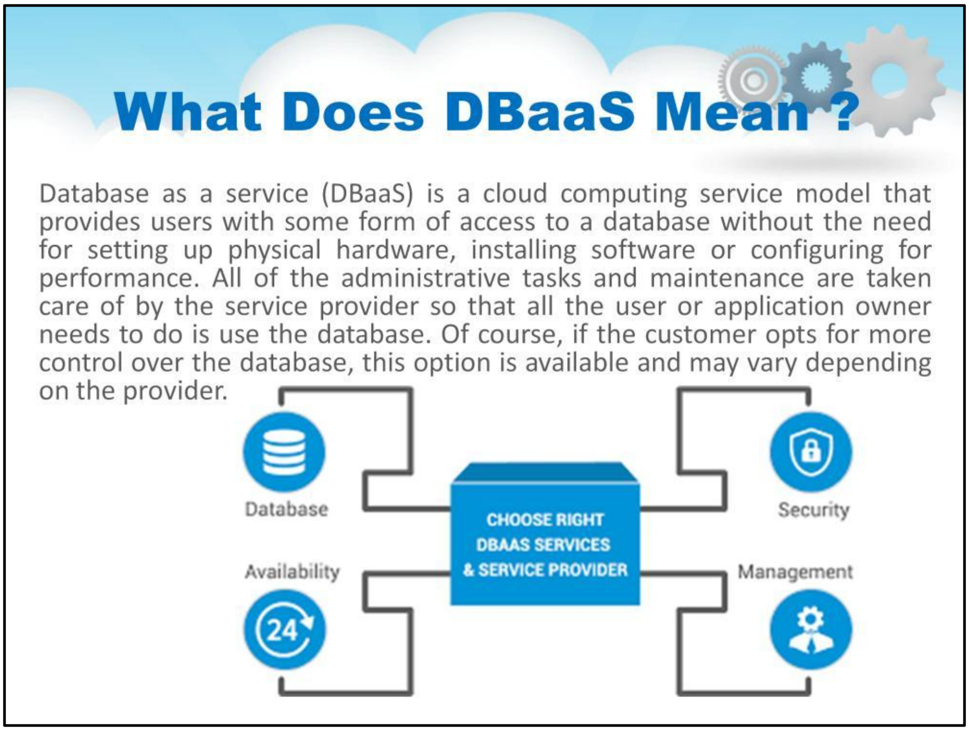


Cloud Service Models

–

Database as a Service (DBaaS)

Database as a Service (DaaS)



What Does DBaaS Mean ?

Database as a service (DBaaS) is a cloud computing service model that provides users with some form of access to a database without the need for setting up physical hardware, installing software or configuring for performance. All of the administrative tasks and maintenance are taken care of by the service provider so that all the user or application owner needs to do is use the database. Of course, if the customer opts for more control over the database, this option is available and may vary depending on the provider.

Database

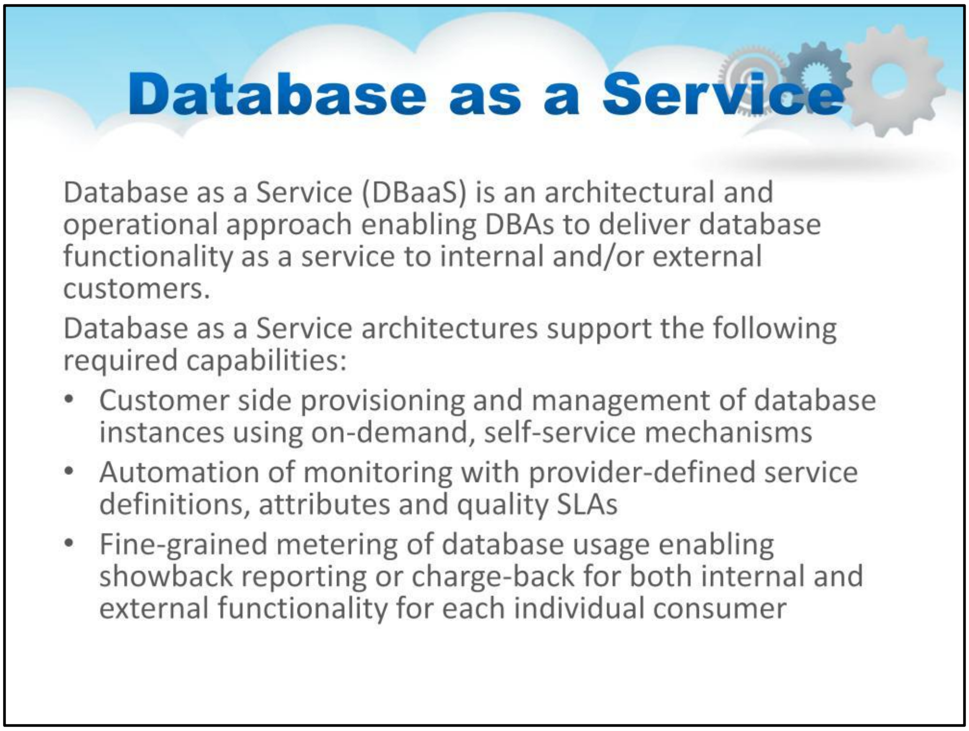
Security

CHOOSE RIGHT DBAAS SERVICES & SERVICE PROVIDER

Availability

Management

More information here:
https://searchcloudapplications.techtarget.com/definition/cloud-database

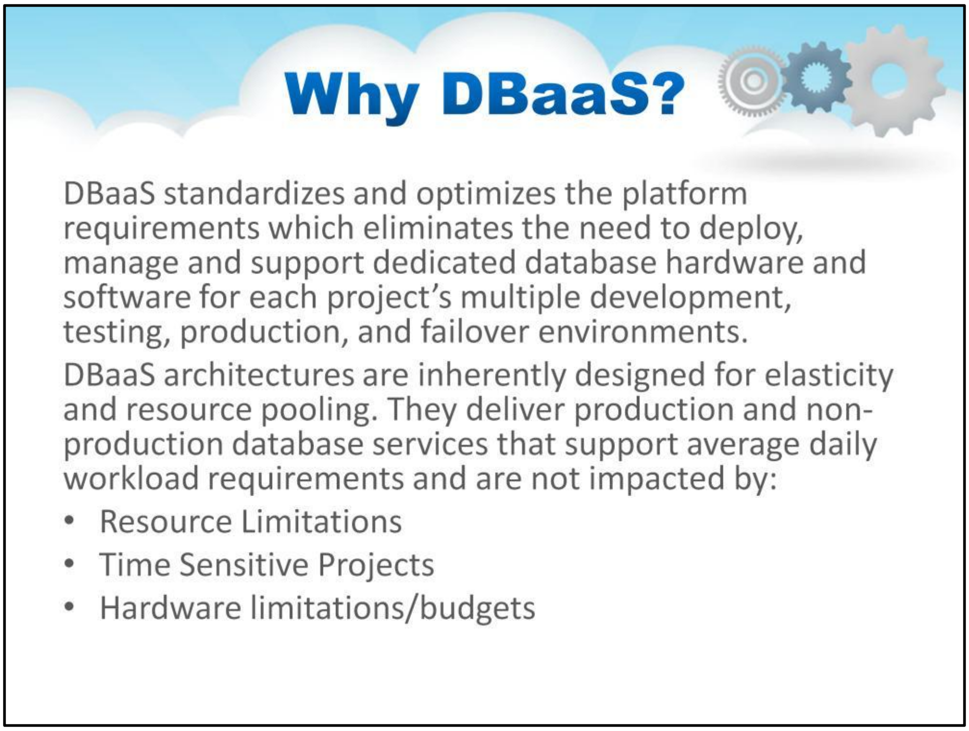


Database as a Service

Database as a Service (DBaaS) is an architectural and operational approach enabling DBAs to deliver database functionality as a service to internal and/or external customers.

Database as a Service architectures support the following required capabilities:

- Customer side provisioning and management of database instances using on-demand, self-service mechanisms
- Automation of monitoring with provider-defined service definitions, attributes and quality SLAs
- Fine-grained metering of database usage enabling showback reporting or charge-back for both internal and external functionality for each individual consumer

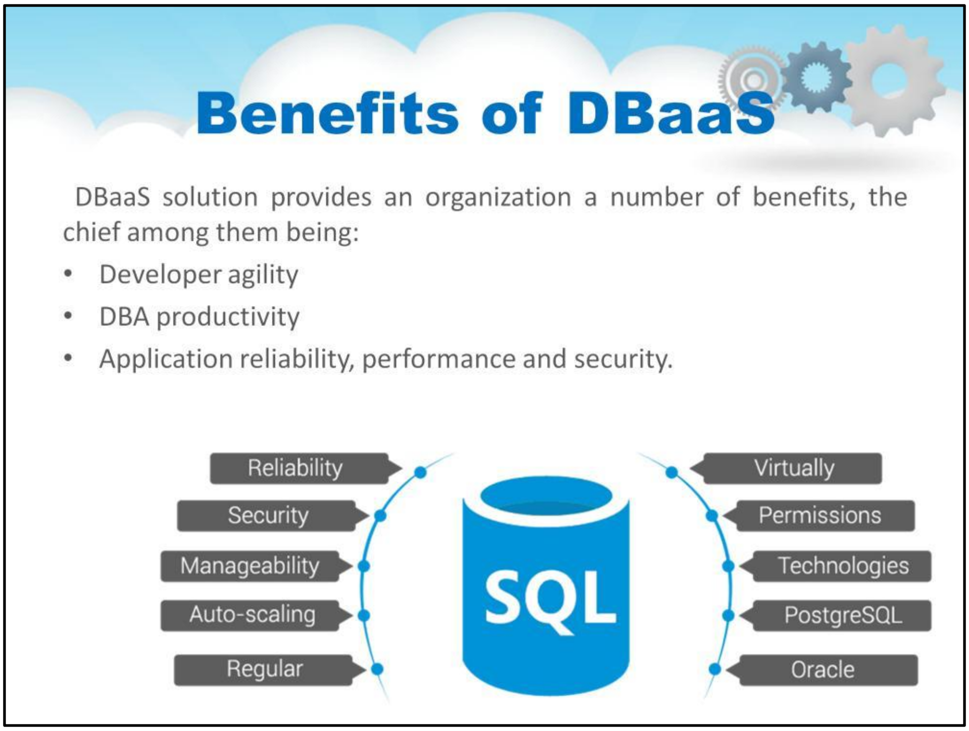


Why DBaaS?

DBaaS standardizes and optimizes the platform requirements which eliminates the need to deploy, manage and support dedicated database hardware and software for each project's multiple development, testing, production, and failover environments.

DBaaS architectures are inherently designed for elasticity and resource pooling. They deliver production and non-production database services that support average daily workload requirements and are not impacted by:

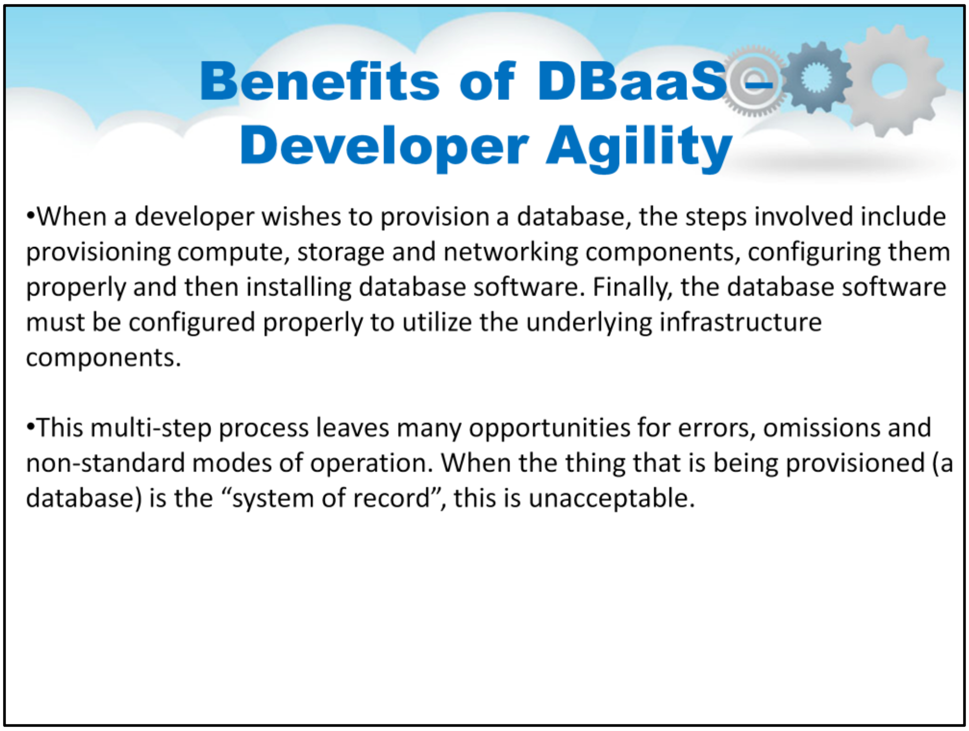
- Resource Limitations
- Time Sensitive Projects
- Hardware limitations/budgets



Benefits of DBaaS

DBaaS solution provides an organization a number of benefits, the chief among them being:

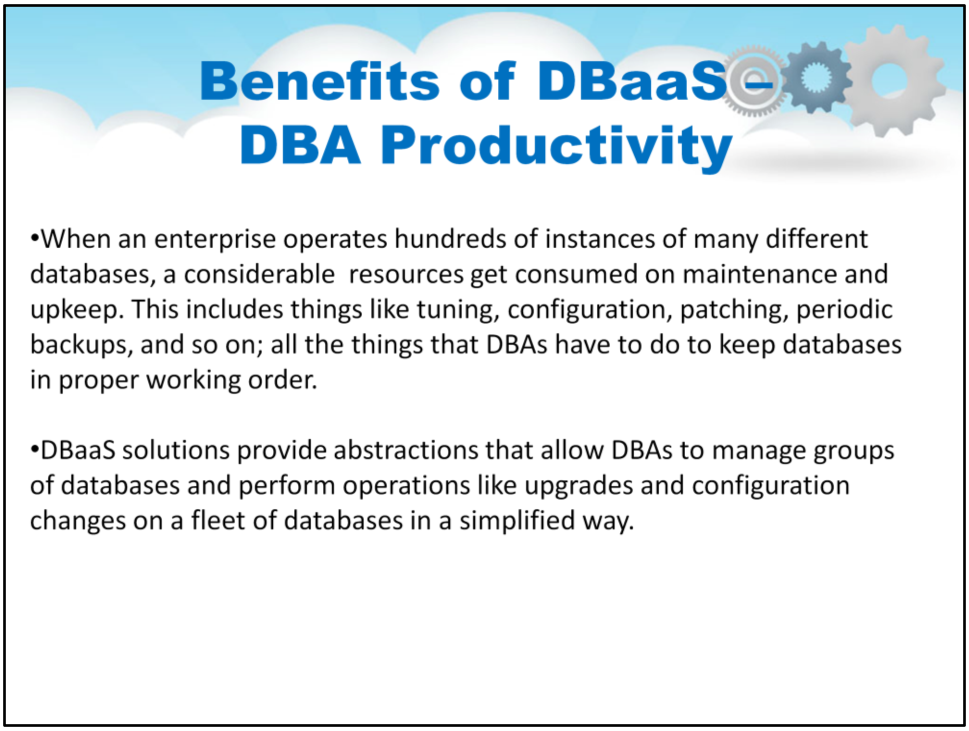
- Developer agility
- DBA productivity
- Application reliability, performance and security.



Benefits of DBaaS – Developer Agility

•When a developer wishes to provision a database, the steps involved include provisioning compute, storage and networking components, configuring them properly and then installing database software. Finally, the database software must be configured properly to utilize the underlying infrastructure components.

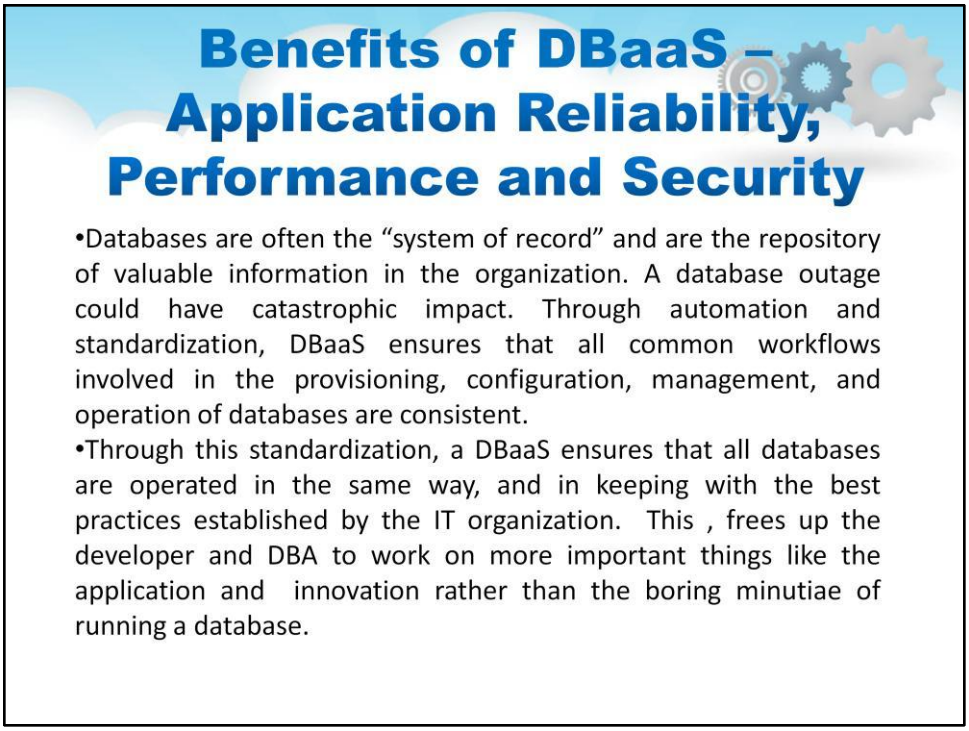
•This multi-step process leaves many opportunities for errors, omissions and non-standard modes of operation. When the thing that is being provisioned (a database) is the "system of record", this is unacceptable.



Benefits of DBaaS – DBA Productivity

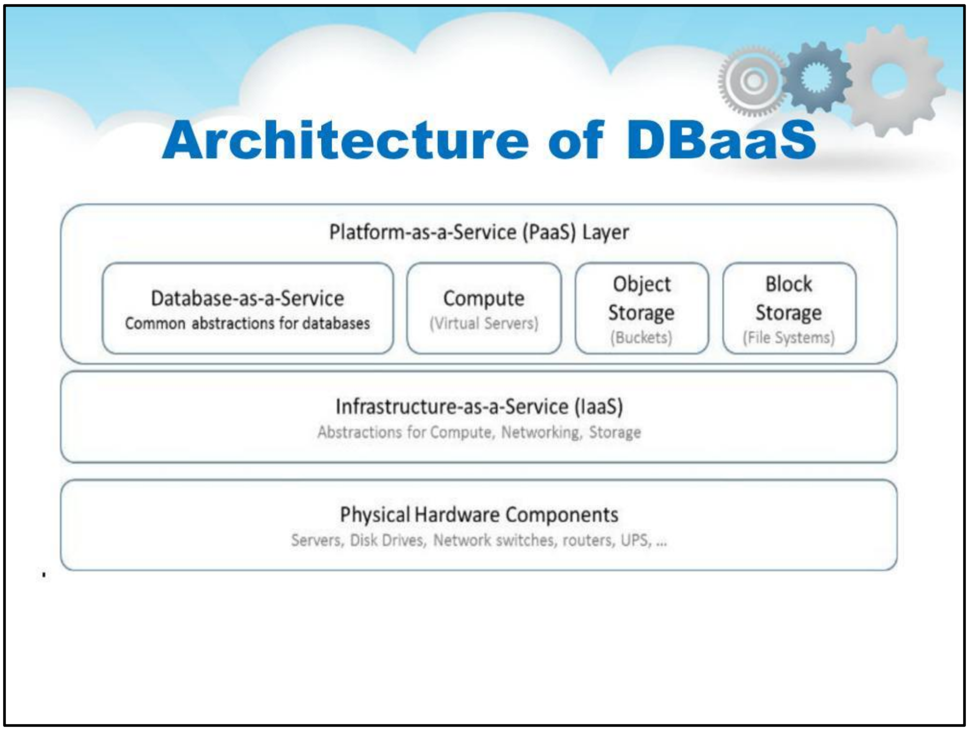
•When an enterprise operates hundreds of instances of many different databases, a considerable resources get consumed on maintenance and upkeep. This includes things like tuning, configuration, patching, periodic backups, and so on; all the things that DBAs have to do to keep databases in proper working order.

•DBaaS solutions provide abstractions that allow DBAs to manage groups of databases and perform operations like upgrades and configuration changes on a fleet of databases in a simplified way.



Benefits of DBaaS – Application Reliability, Performance and Security

- Databases are often the “system of record” and are the repository of valuable information in the organization. A database outage could have catastrophic impact. Through automation and standardization, DBaaS ensures that all common workflows involved in the provisioning, configuration, management, and operation of databases are consistent.
- Through this standardization, a DBaaS ensures that all databases are operated in the same way, and in keeping with the best practices established by the IT organization. This , frees up the developer and DBA to work on more important things like the application and innovation rather than the boring minutiae of running a database.



Architecture of DBaaS

Platform-as-a-Service (PaaS) Layer

Database-as-a-Service
Common abstractions for databases

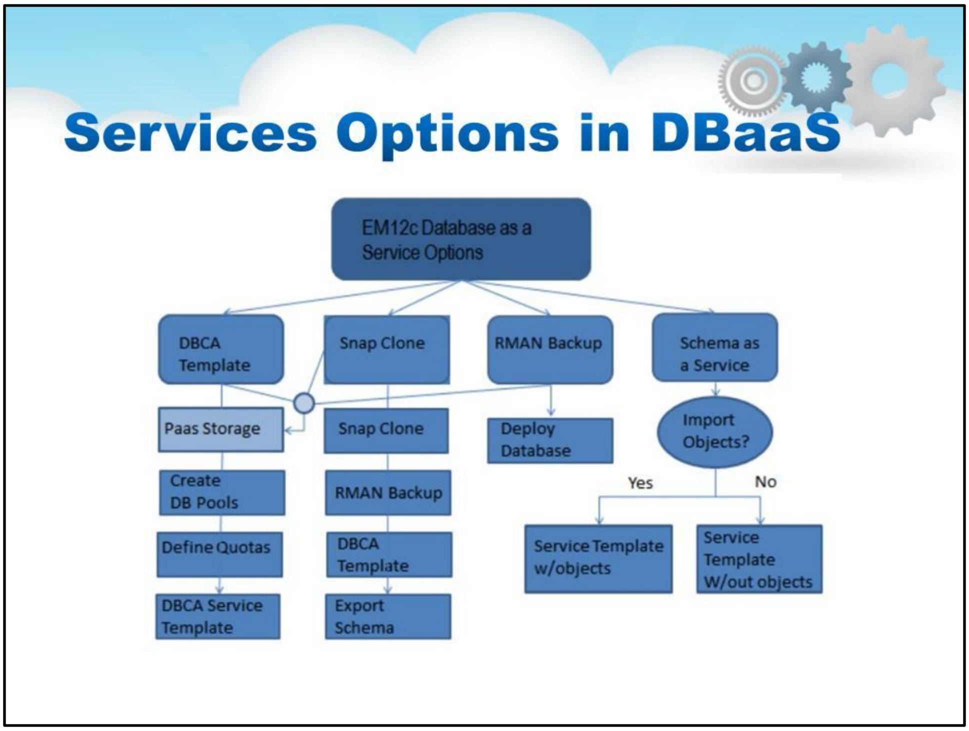
Compute
(Virtual Servers)

Object Storage
(Buckets)

Block Storage
(File Systems)

Infrastructure-as-a-Service (IaaS)
Abstractions for Compute, Networking, Storage

Physical Hardware Components
Servers, Disk Drives, Network switches, routers, UPS, ...



Services Options in DBaaS

EM12c Database as a Service Options

DBCA Template

Snap Clone

RMAN Backup

Schema as a Service

Paas Storage

Create DB Pools

Define Quotas

DBCA Service Template

Snap Clone

RMAN Backup

DBCA Template

Export Schema

Deploy Database

Import Objects?

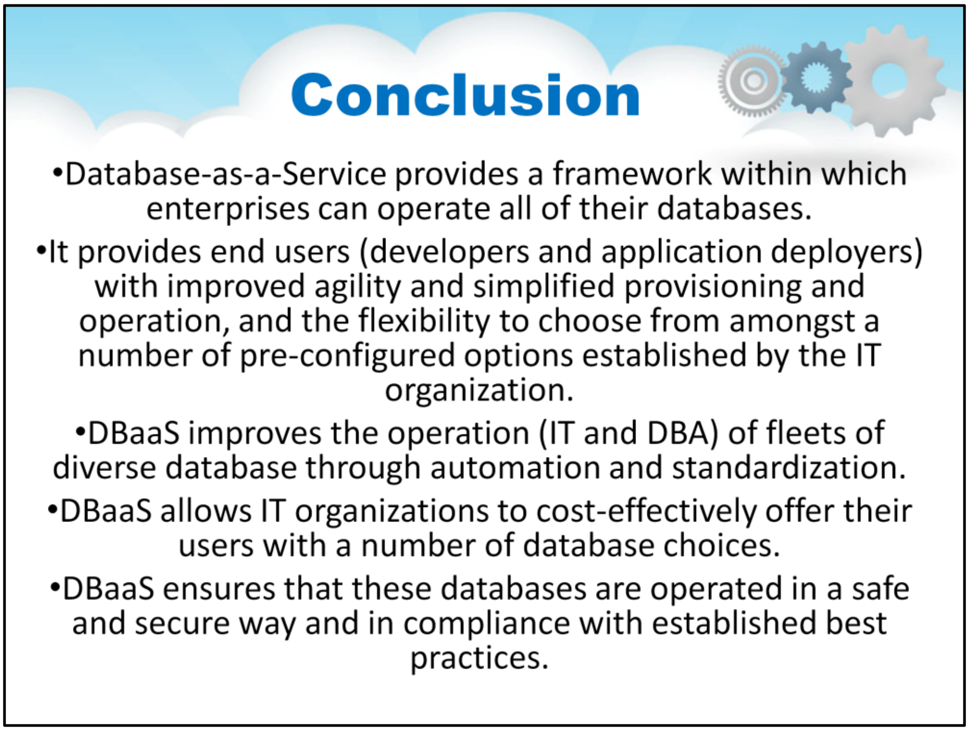
Yes

No

Service Template w/objects

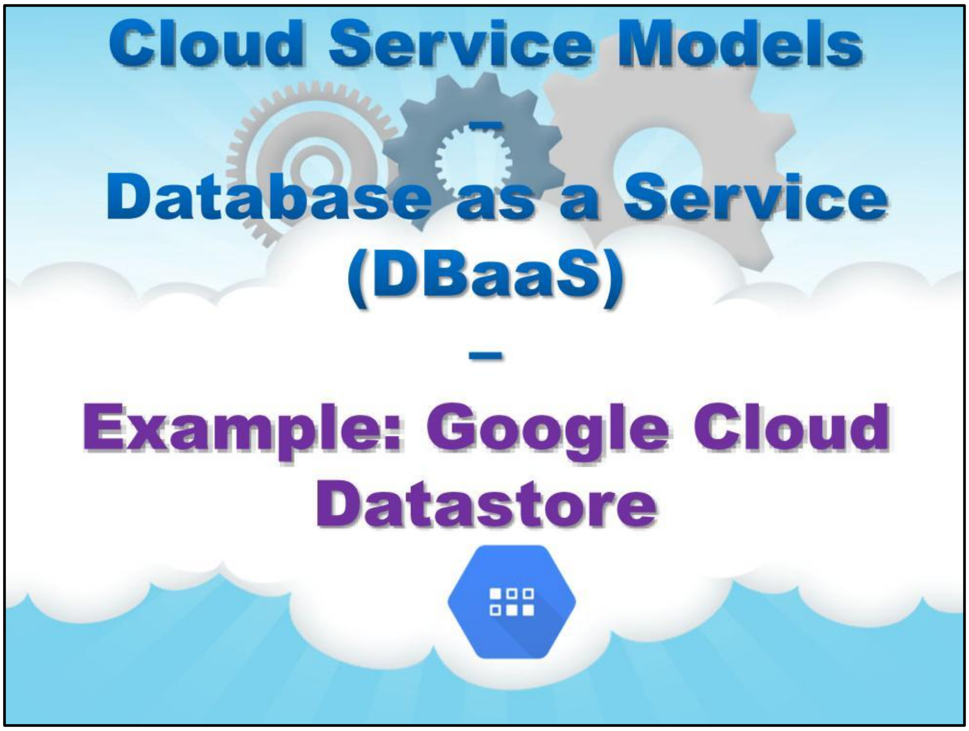
Service Template W/out objects

This slide shows main service options in DBaaS



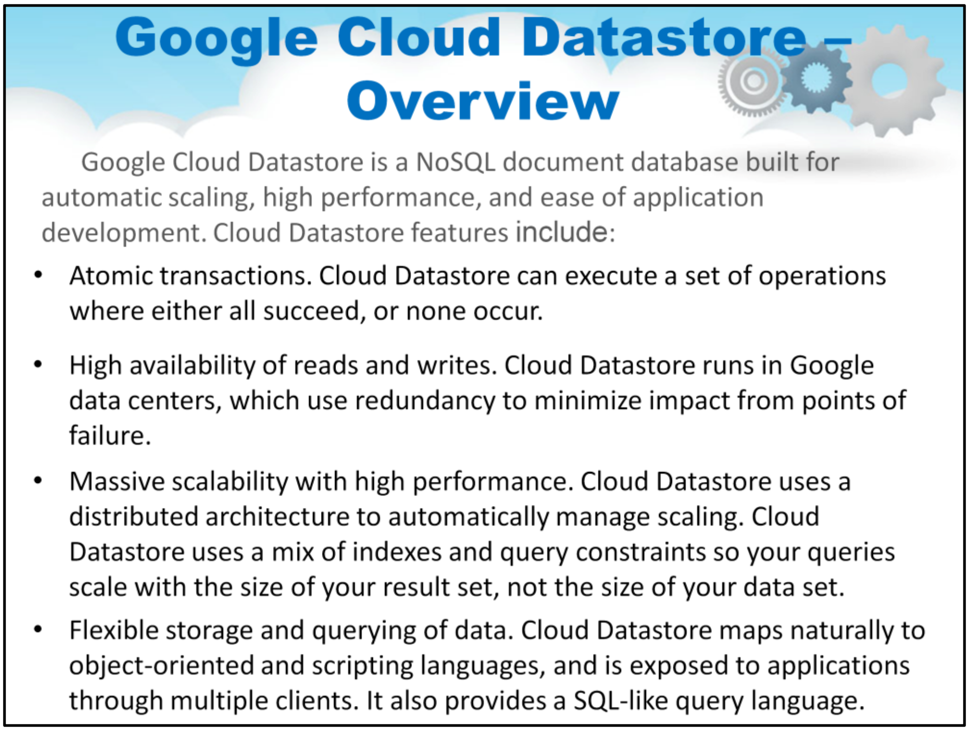
Conclusion

- Database-as-a-Service provides a framework within which enterprises can operate all of their databases.
- It provides end users (developers and application deployers) with improved agility and simplified provisioning and operation, and the flexibility to choose from amongst a number of pre-configured options established by the IT organization.
- DBaaS improves the operation (IT and DBA) of fleets of diverse database through automation and standardization.
- DBaaS allows IT organizations to cost-effectively offer their users with a number of database choices.
- DBaaS ensures that these databases are operated in a safe and secure way and in compliance with established best practices.



Cloud Service Models – Database as a Service (DBaaS) – Example: Google Cloud Datastore

Example: Google Cloud Datastore

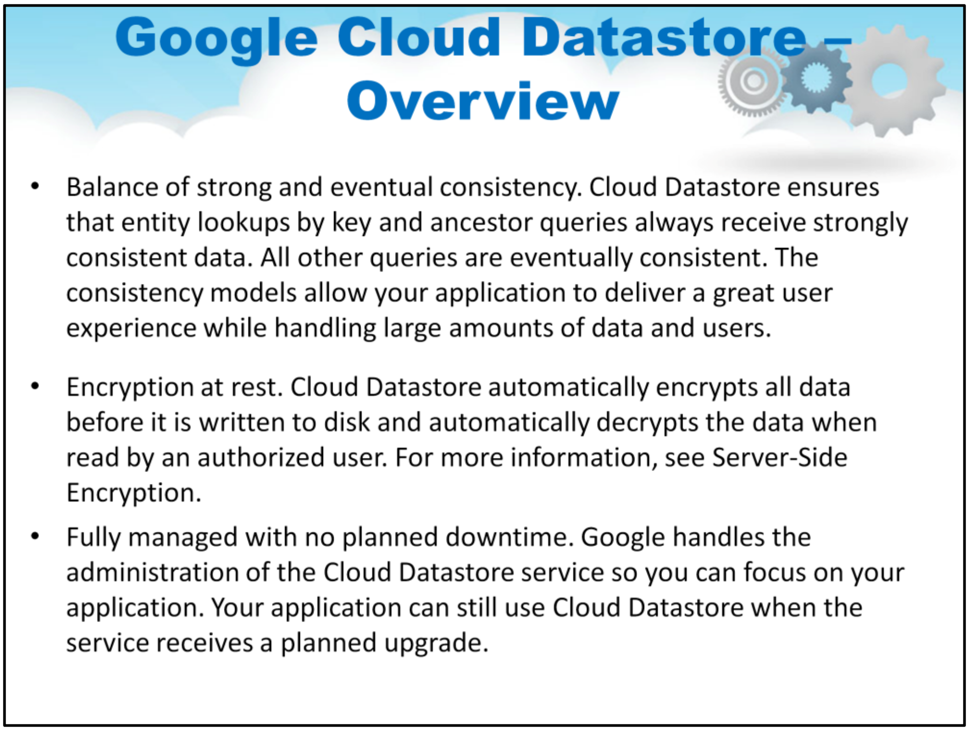


Google Cloud Datastore – Overview

Google Cloud Datastore is a NoSQL document database built for automatic scaling, high performance, and ease of application development. Cloud Datastore features include:

- Atomic transactions. Cloud Datastore can execute a set of operations where either all succeed, or none occur.
- High availability of reads and writes. Cloud Datastore runs in Google data centers, which use redundancy to minimize impact from points of failure.
- Massive scalability with high performance. Cloud Datastore uses a distributed architecture to automatically manage scaling. Cloud Datastore uses a mix of indexes and query constraints so your queries scale with the size of your result set, not the size of your data set.
- Flexible storage and querying of data. Cloud Datastore maps naturally to object-oriented and scripting languages, and is exposed to applications through multiple clients. It also provides a SQL-like query language.

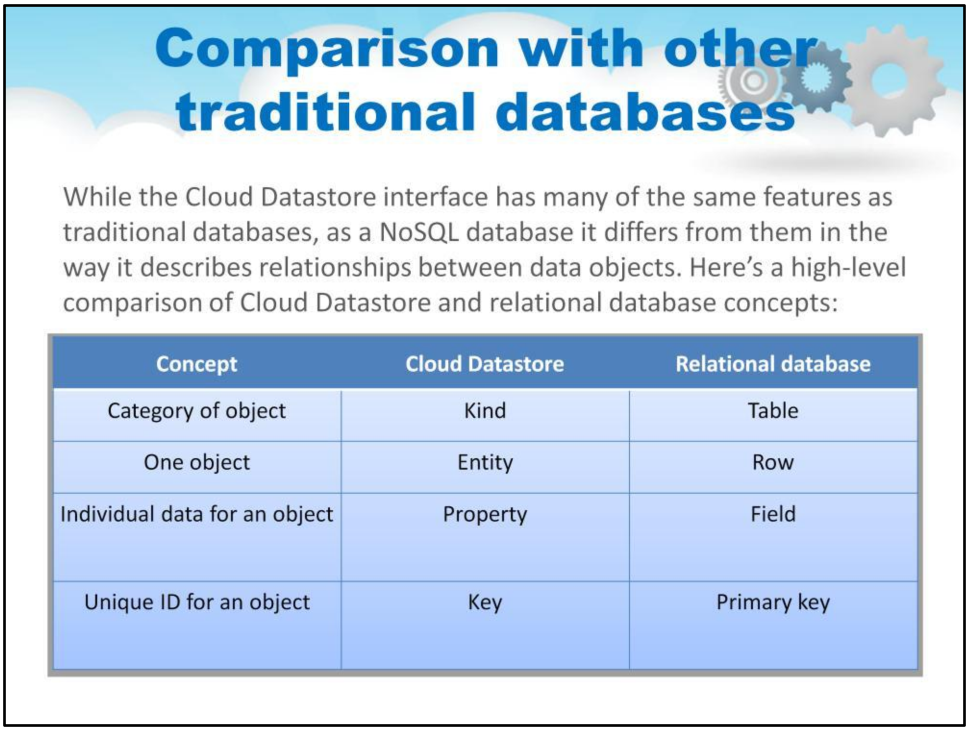
Official documentation: https://cloud.google.com/datastore/docs/concepts/overview



Google Cloud Datastore – Overview

- Balance of strong and eventual consistency. Cloud Datastore ensures that entity lookups by key and ancestor queries always receive strongly consistent data. All other queries are eventually consistent. The consistency models allow your application to deliver a great user experience while handling large amounts of data and users.
- Encryption at rest. Cloud Datastore automatically encrypts all data before it is written to disk and automatically decrypts the data when read by an authorized user. For more information, see Server-Side Encryption.
- Fully managed with no planned downtime. Google handles the administration of the Cloud Datastore service so you can focus on your application. Your application can still use Cloud Datastore when the service receives a planned upgrade.

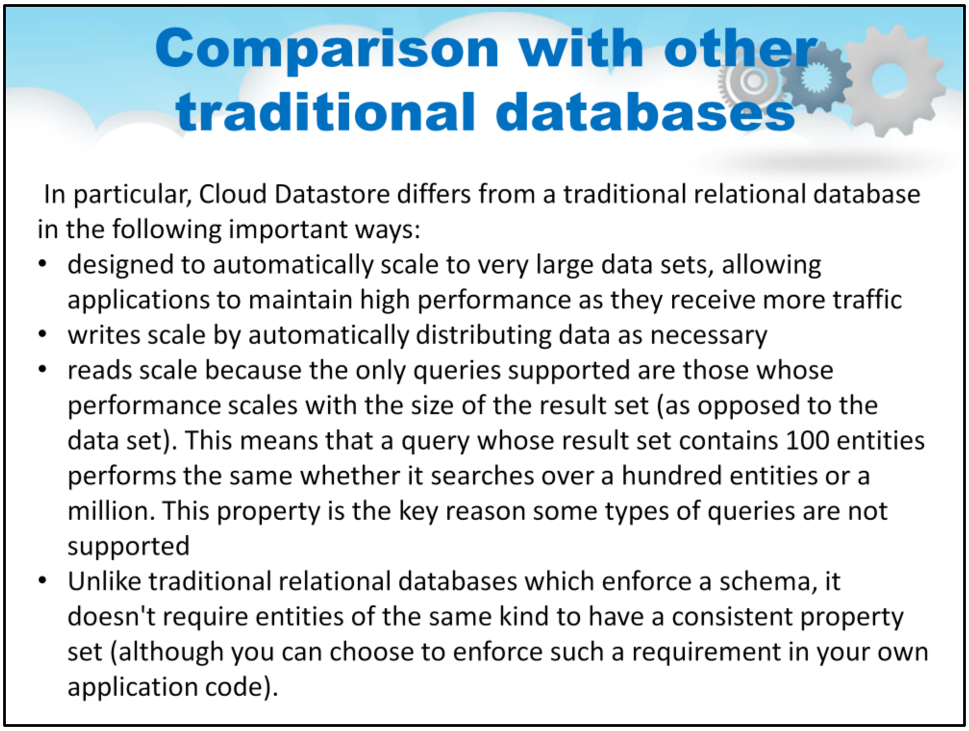
Official documentation: https://cloud.google.com/datastore/docs/concepts/overview



Comparison with other traditional databases

While the Cloud Datastore interface has many of the same features as traditional databases, as a NoSQL database it differs from them in the way it describes relationships between data objects. Here's a high-level comparison of Cloud Datastore and relational database concepts:

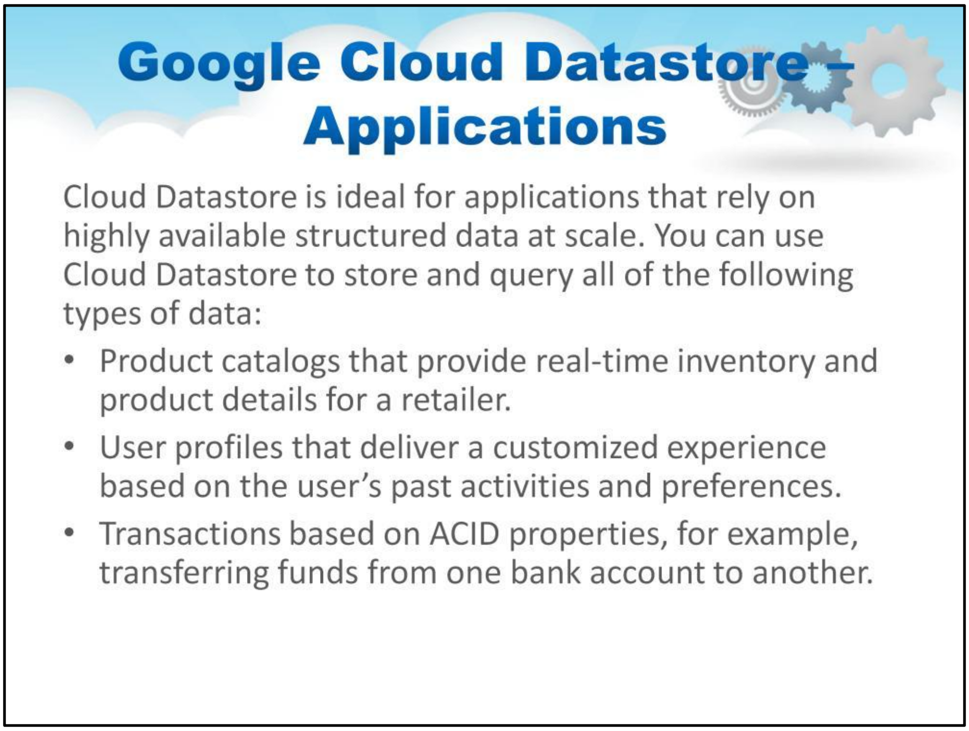
Concept	Cloud Datastore	Relational database
Category of object	Kind	Table
One object	Entity	Row
Individual data for an object	Property	Field
Unique ID for an object	Key	Primary key



Comparison with other traditional databases

In particular, Cloud Datastore differs from a traditional relational database in the following important ways:

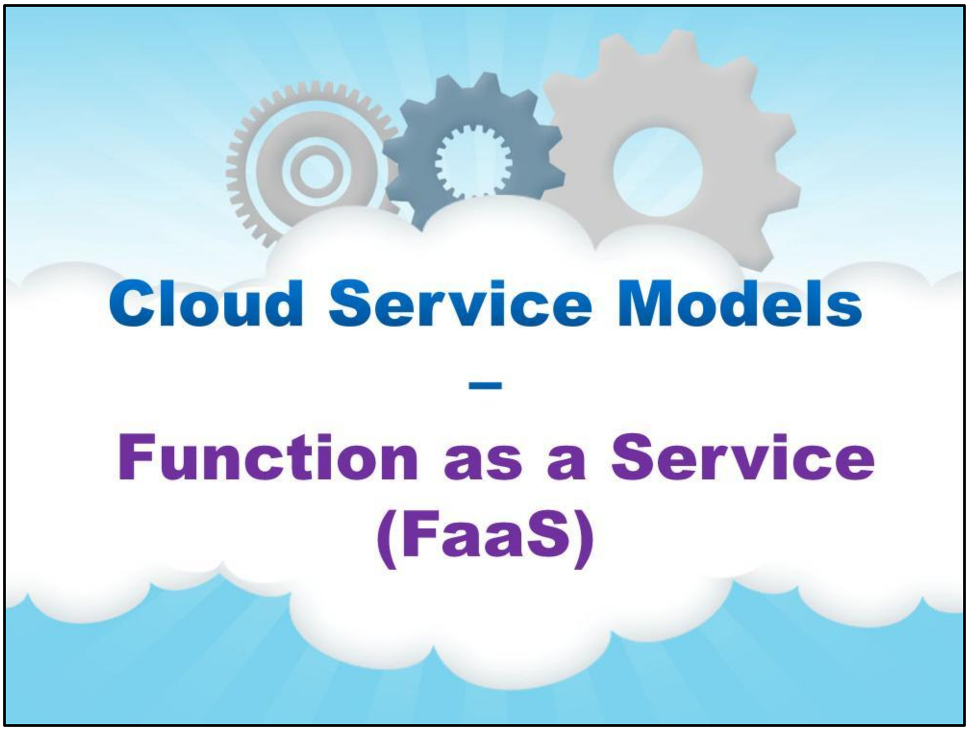
- designed to automatically scale to very large data sets, allowing applications to maintain high performance as they receive more traffic
- writes scale by automatically distributing data as necessary
- reads scale because the only queries supported are those whose performance scales with the size of the result set (as opposed to the data set). This means that a query whose result set contains 100 entities performs the same whether it searches over a hundred entities or a million. This property is the key reason some types of queries are not supported
- Unlike traditional relational databases which enforce a schema, it doesn't require entities of the same kind to have a consistent property set (although you can choose to enforce such a requirement in your own application code).



Google Cloud Datastore – Applications

Cloud Datastore is ideal for applications that rely on highly available structured data at scale. You can use Cloud Datastore to store and query all of the following types of data:

- Product catalogs that provide real-time inventory and product details for a retailer.
- User profiles that deliver a customized experience based on the user's past activities and preferences.
- Transactions based on ACID properties, for example, transferring funds from one bank account to another.



Cloud Service Models

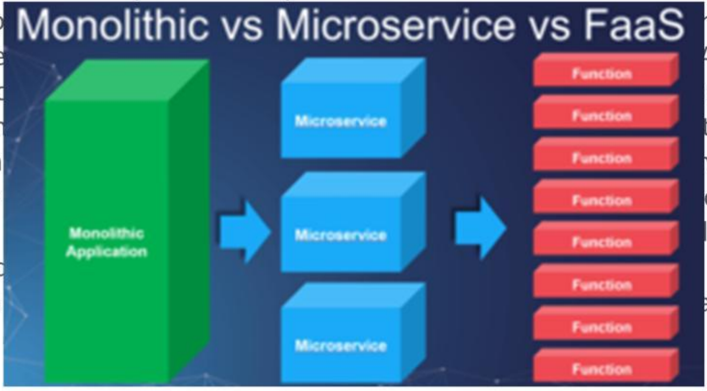
–

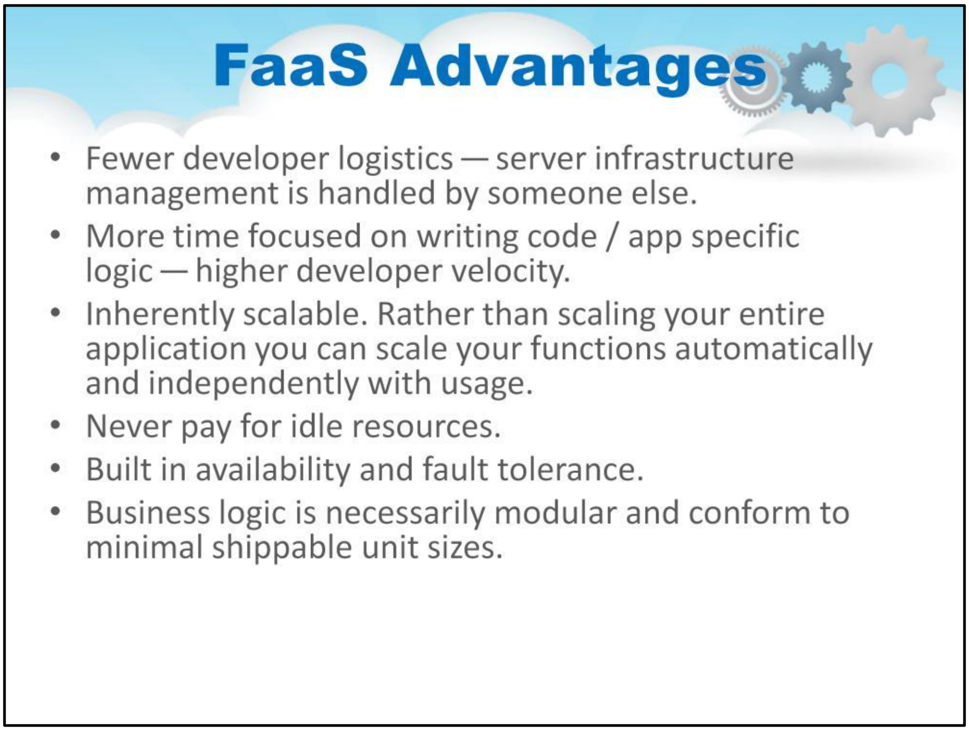
Function as a Service (FaaS)

Function as a Service (FaaS)

What is a purpose of FaaS?

FaaS is a relatively new concept that was first made available in 2014
by hoo
nbda,
Google
Azure
Functic
ream
allowin
thout
buildin
neans
is that
o the
cloud
lities!
Instead
load,
you ca
an be
scaled

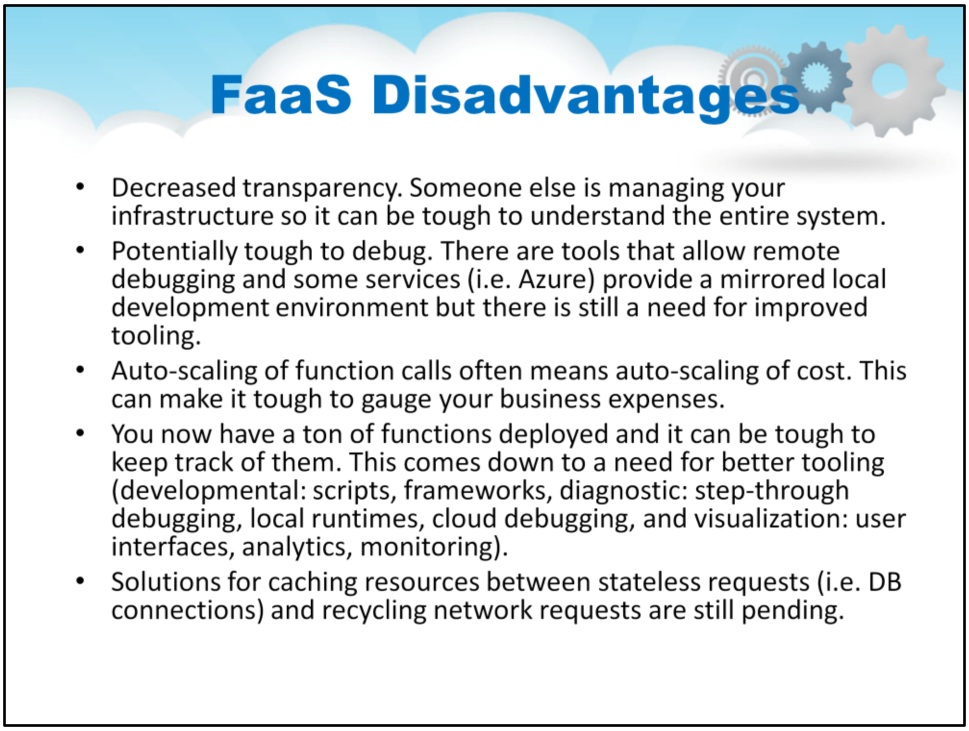




FaaS Advantages

- Fewer developer logistics — server infrastructure management is handled by someone else.
- More time focused on writing code / app specific logic — higher developer velocity.
- Inherently scalable. Rather than scaling your entire application you can scale your functions automatically and independently with usage.
- Never pay for idle resources.
- Built in availability and fault tolerance.
- Business logic is necessarily modular and conform to minimal shippable unit sizes.

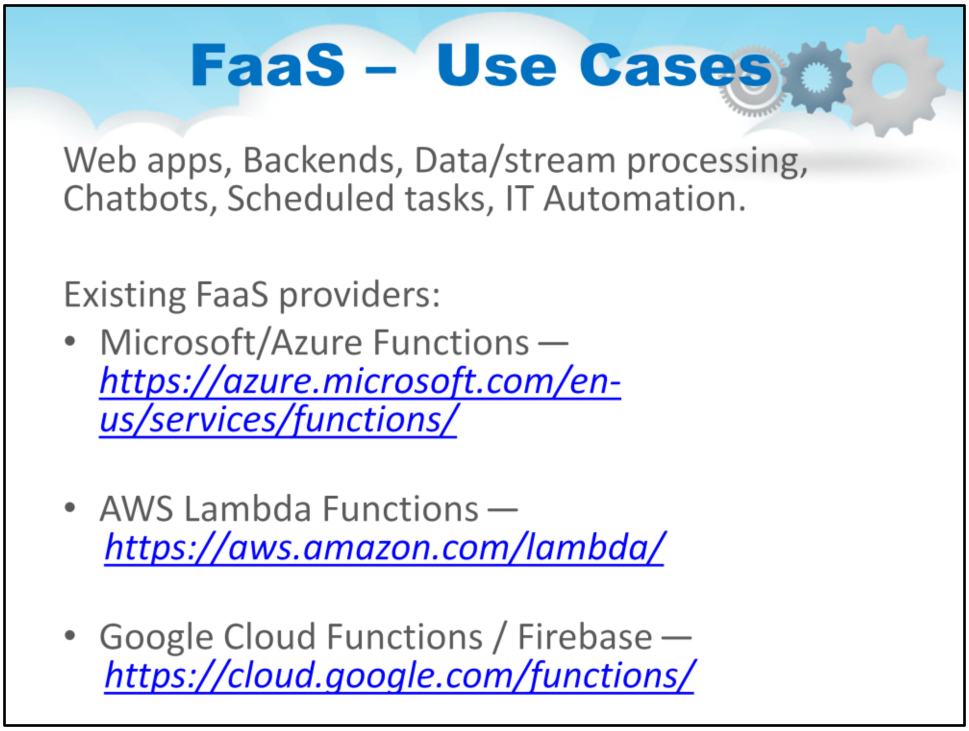
Great article, that describes what is exactly FaaS and what does it mean: https://stackify.com/function-as-a-service-serverless-architecture/



FaaS Disadvantages

- Decreased transparency. Someone else is managing your infrastructure so it can be tough to understand the entire system.
- Potentially tough to debug. There are tools that allow remote debugging and some services (i.e. Azure) provide a mirrored local development environment but there is still a need for improved tooling.
- Auto-scaling of function calls often means auto-scaling of cost. This can make it tough to gauge your business expenses.
- You now have a ton of functions deployed and it can be tough to keep track of them. This comes down to a need for better tooling (developmental: scripts, frameworks, diagnostic: step-through debugging, local runtimes, cloud debugging, and visualization: user interfaces, analytics, monitoring).
- Solutions for caching resources between stateless requests (i.e. DB connections) and recycling network requests are still pending.

Great article, that describes what is exactly FaaS and what does it mean:
https://stackify.com/function-as-a-service-serverless-architecture/



FaaS – Use Cases

Web apps, Backends, Data/stream processing, Chatbots, Scheduled tasks, IT Automation.

Existing FaaS providers:

- Microsoft/Azure Functions — https://azure.microsoft.com/en-us/services/functions/
- AWS Lambda Functions — https://aws.amazon.com/lambda/
- Google Cloud Functions / Firebase — https://cloud.google.com/functions/

Existing FaaS providers are as follows:

Microsoft/Azure Functions—https://azure.microsoft.com/en-us/services/functions/ proposes

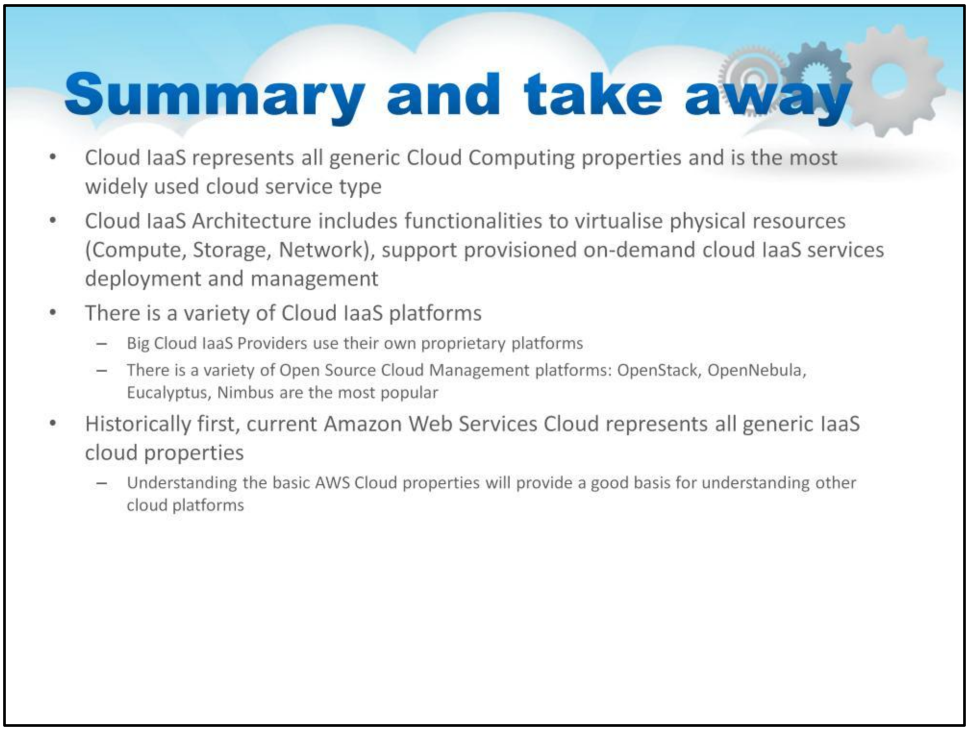
•Tooling, bindings & triggers
•Logic apps—visual designer with 25+ connectors and function orchestration
•Event grid
•Local debugging

AWS Lambda Functions—https://aws.amazon.com/lambda/ proposes

•Functions,
•APIs,
•Tables

Google Cloud Functions / Firebase—https://cloud.google.com/functions/ proposes

•Cloud storage,
•Cloud pub/sub,
•HTTPS,
•Stackdriver logs



Summary and take away

- Cloud IaaS represents all generic Cloud Computing properties and is the most widely used cloud service type
- Cloud IaaS Architecture includes functionalities to virtualise physical resources (Compute, Storage, Network), support provisioned on-demand cloud IaaS services deployment and management
- There is a variety of Cloud IaaS platforms
  - Big Cloud IaaS Providers use their own proprietary platforms
  - There is a variety of Open Source Cloud Management platforms: OpenStack, OpenNebula, Eucalyptus, Nimbus are the most popular
- Historically first, current Amazon Web Services Cloud represents all generic IaaS cloud properties
  - Understanding the basic AWS Cloud properties will provide a good basis for understanding other cloud platforms

Summary and take away

Cloud IaaS represents all generic Cloud Computing properties and is the most widely used cloud service type

Cloud IaaS Architecture includes functionalities to virtualise physical resources (Compute, Storage, Network), support provisioned on-demand cloud IaaS services deployment and management

There is a variety of Cloud IaaS platforms

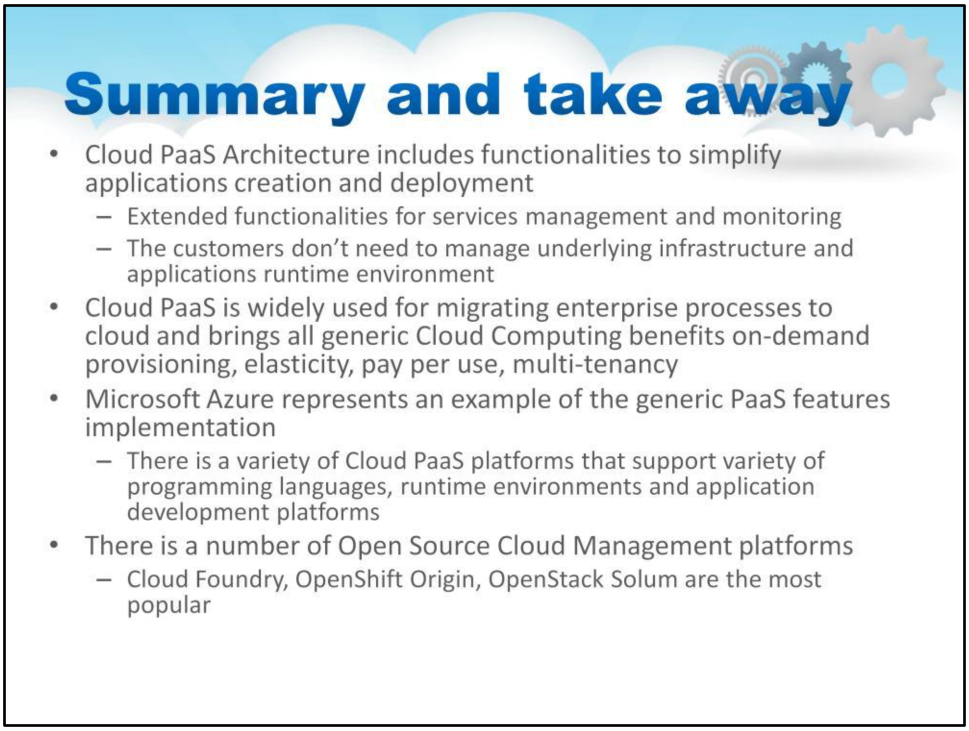
Big Cloud IaaS Providers use their own proprietary platforms

There is a variety of Open Source Cloud Management platforms

OpenStack, OpenNebula, Eucalyptus, Nimbus are the most popular

And finally, in this tutorial we discussed the Amazon Web Services Cloud and its main functionality

Historically first, current AWS Cloud represents all the generic IaaS cloud properties. Understanding the basic AWS Cloud properties will provide a good basis for understanding other cloud platforms



Summary and take away

- Cloud PaaS Architecture includes functionalities to simplify applications creation and deployment
  - Extended functionalities for services management and monitoring
  - The customers don't need to manage underlying infrastructure and applications runtime environment
- Cloud PaaS is widely used for migrating enterprise processes to cloud and brings all generic Cloud Computing benefits on-demand provisioning, elasticity, pay per use, multi-tenancy
- Microsoft Azure represents an example of the generic PaaS features implementation
  - There is a variety of Cloud PaaS platforms that support variety of programming languages, runtime environments and application development platforms
- There is a number of Open Source Cloud Management platforms
  - Cloud Foundry, OpenShift Origin, OpenStack Solum are the most popular

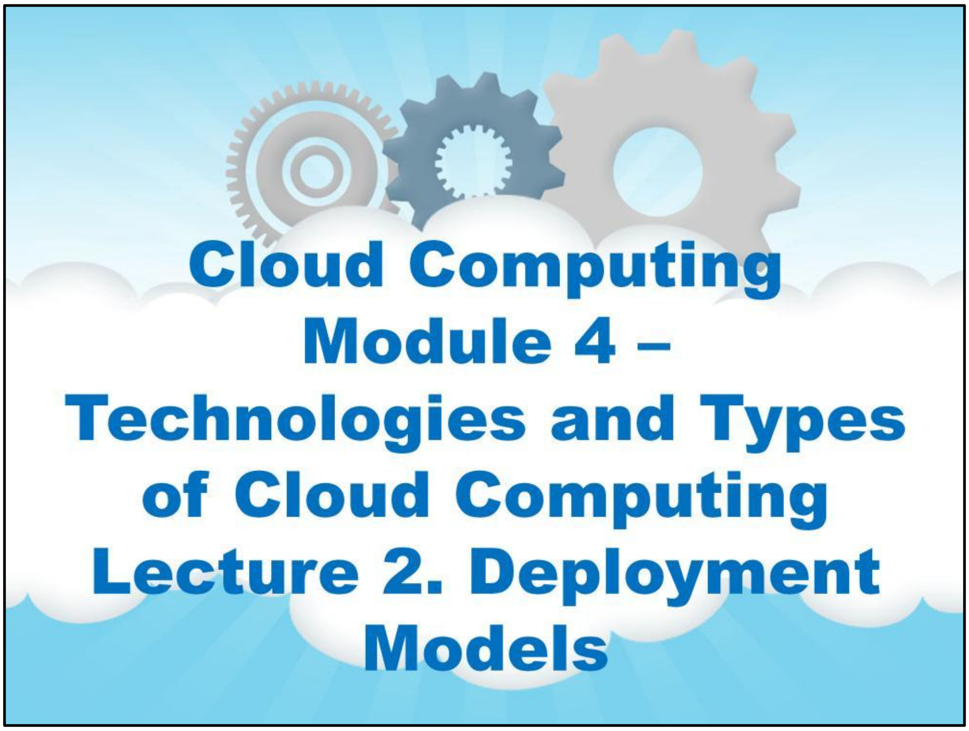
In this Lesson we covered

Cloud PaaS Architecture includes functionalities to simplify applications creation and deployment

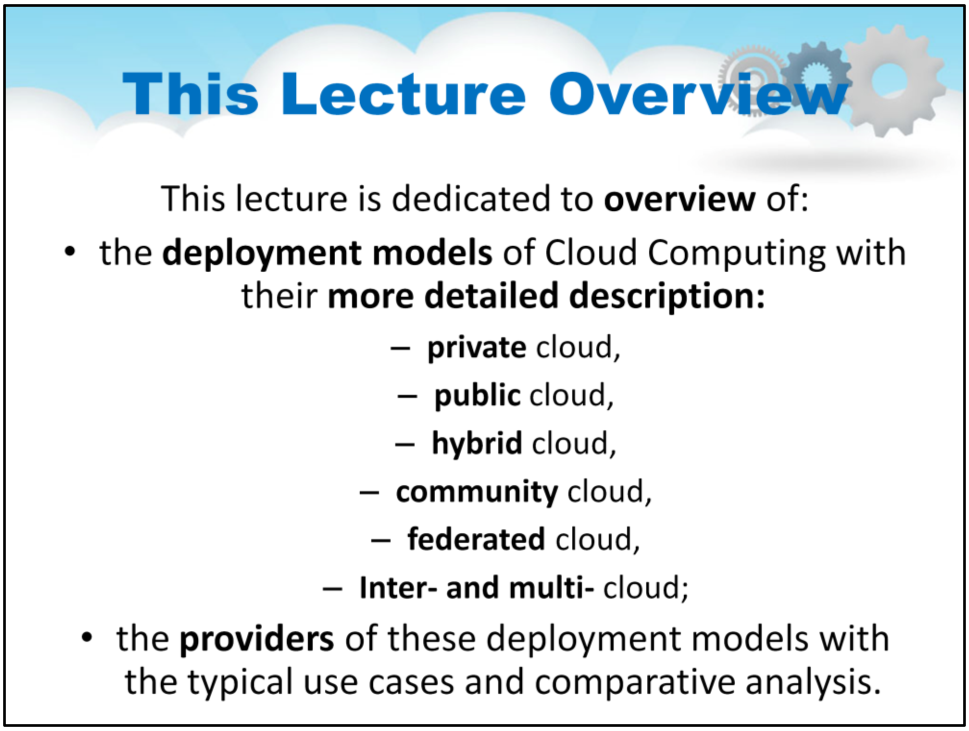
Cloud PaaS is widely used for migrating enterprise processes to cloud and brings all generic Cloud Computing benefits on-demand provisioning, elasticity, pay per use, multi-tenancy

Microsoft Azure represents an example of the generic PaaS features implementation

There is a number of Open Source Cloud Management platforms



Cloud Computing Module 4 – Technologies and Types of Cloud Computing Lecture 2. Deployment Models



This Lecture Overview

This lecture is dedicated to **overview** of:

- the **deployment models** of Cloud Computing with their **more detailed description:**
  - **private** cloud,
  - **public** cloud,
  - **hybrid** cloud,
  - **community** cloud,
  - **federated** cloud,
  - **Inter- and multi-** cloud;
- the **providers** of these deployment models with the typical use cases and comparative analysis.

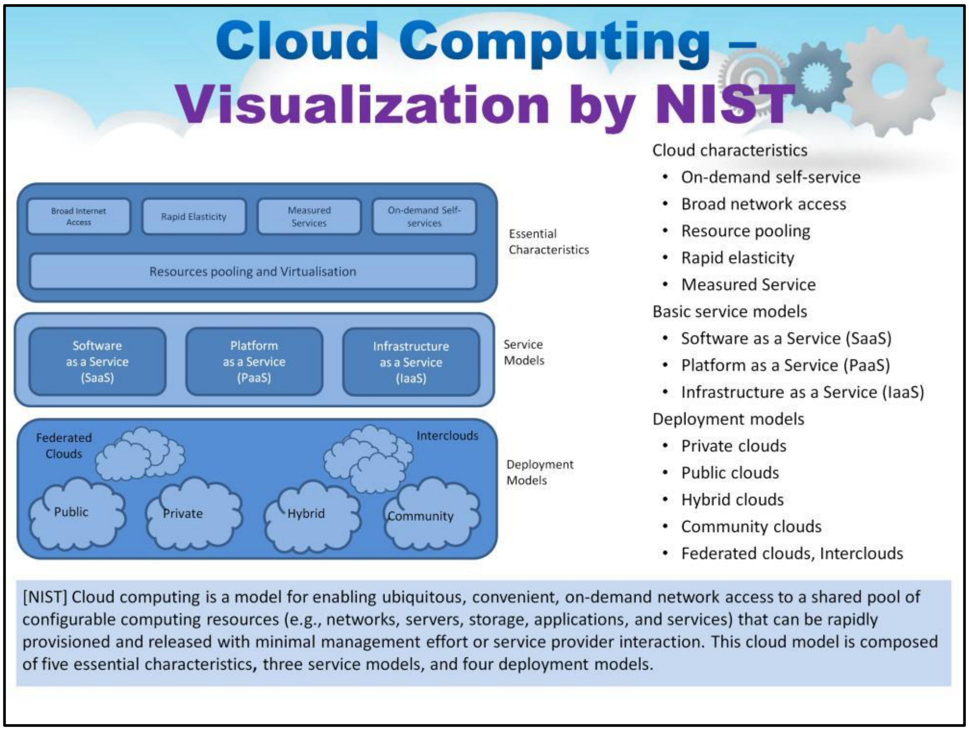
Lecture 2. Deployment Models

Cloud Deployment Models

–

Overview

Overview



Cloud Computing – Visualization by NIST

Broad Internet Access

Rapid Elasticity

Measured Services

On-demand Self-services

Resources pooling and Virtualisation

Essential Characteristics

Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

Service Models

Federated Clouds

Interclouds

Public

Private

Hybrid

Community

Deployment Models

Cloud characteristics

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured Service

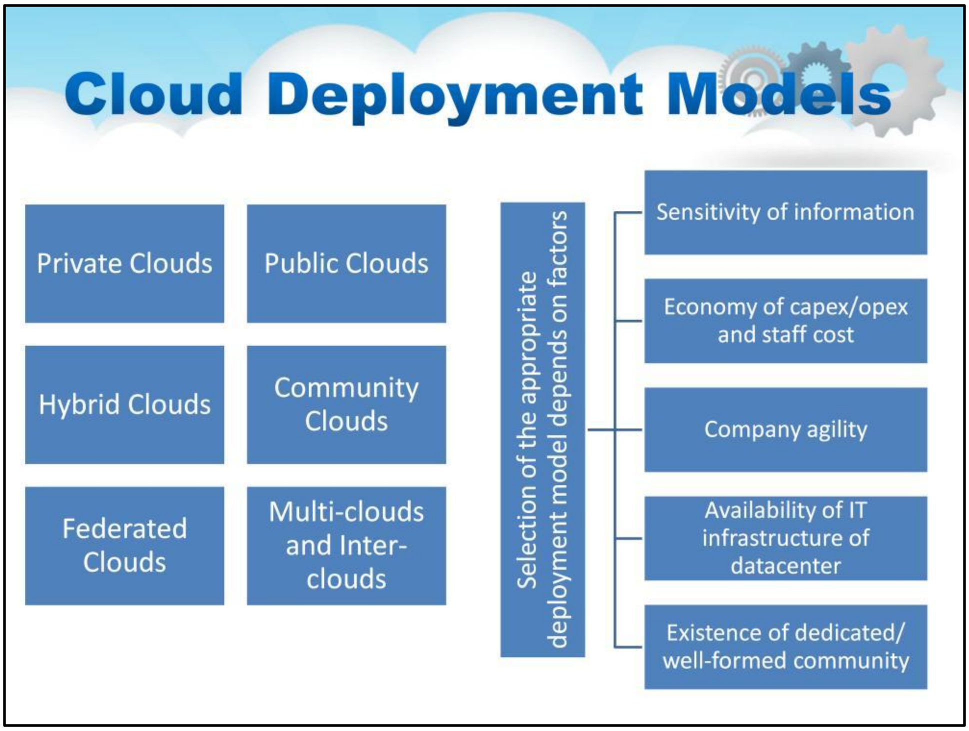
Basic service models

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Deployment models

- Private clouds
- Public clouds
- Hybrid clouds
- Community clouds
- Federated clouds, Interclouds

[NIST] Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.



Cloud Deployment Models

Private Clouds

Public Clouds

Hybrid Clouds

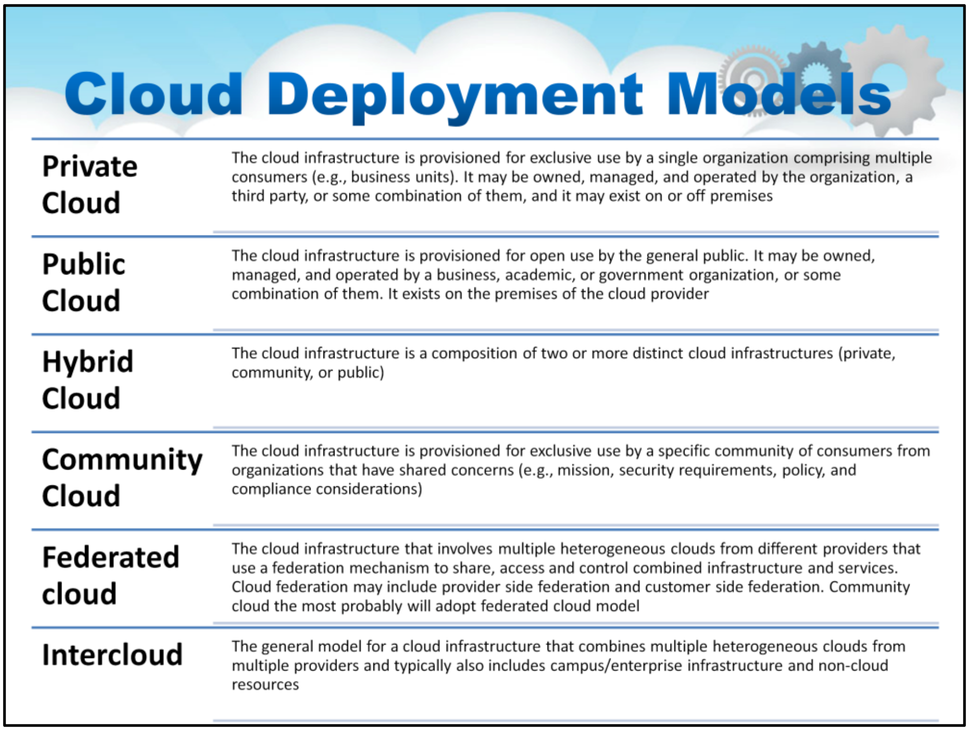
Community Clouds

Federated Clouds

Multi-clouds and Inter-clouds

Selection of the appropriate deployment model depends on factors

- Sensitivity of information
- Economy of capex/opex and staff cost
- Company agility
- Availability of IT infrastructure of datacenter
- Existence of dedicated/ well-formed community



Cloud Deployment Models

Private Cloud
The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises

Public Cloud
The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider

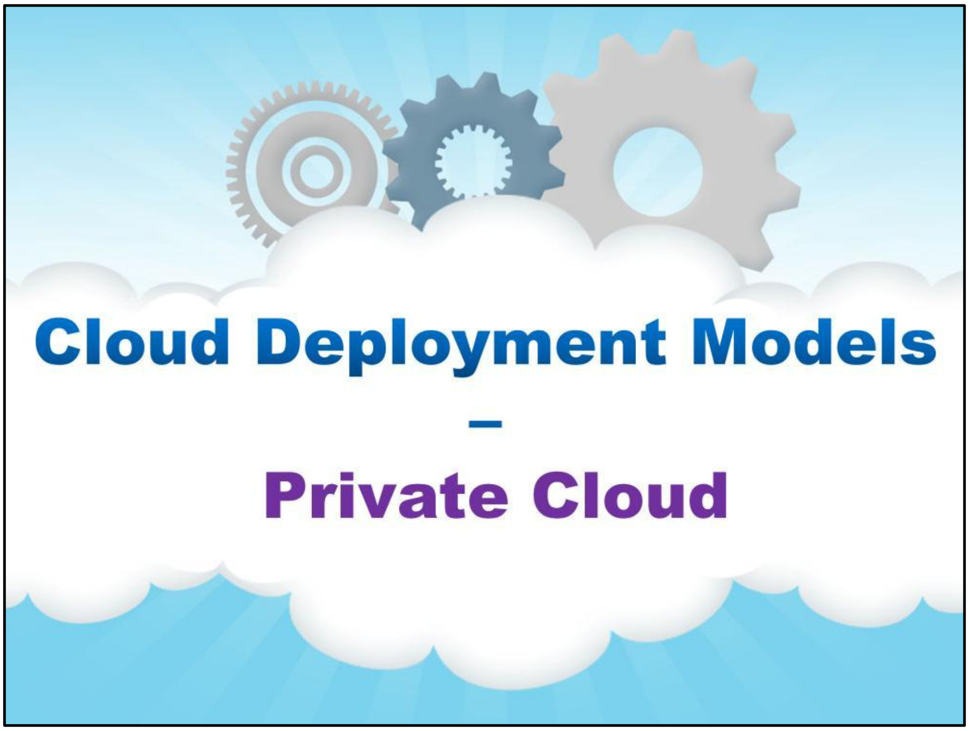
Hybrid Cloud
The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public)

Community Cloud
The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations)

Federated cloud
The cloud infrastructure that involves multiple heterogeneous clouds from different providers that use a federation mechanism to share, access and control combined infrastructure and services. Cloud federation may include provider side federation and customer side federation. Community cloud the most probably will adopt federated cloud model

Intercloud
The general model for a cloud infrastructure that combines multiple heterogeneous clouds from multiple providers and typically also includes campus/enterprise infrastructure and non-cloud resources

This slide illustrates the four Cloud IaaS deployment models which are defined in the NIST SP 800-145 standard: private, public, hybrid, community clouds. Two additional evolved recently: federated cloud and Intercloud.

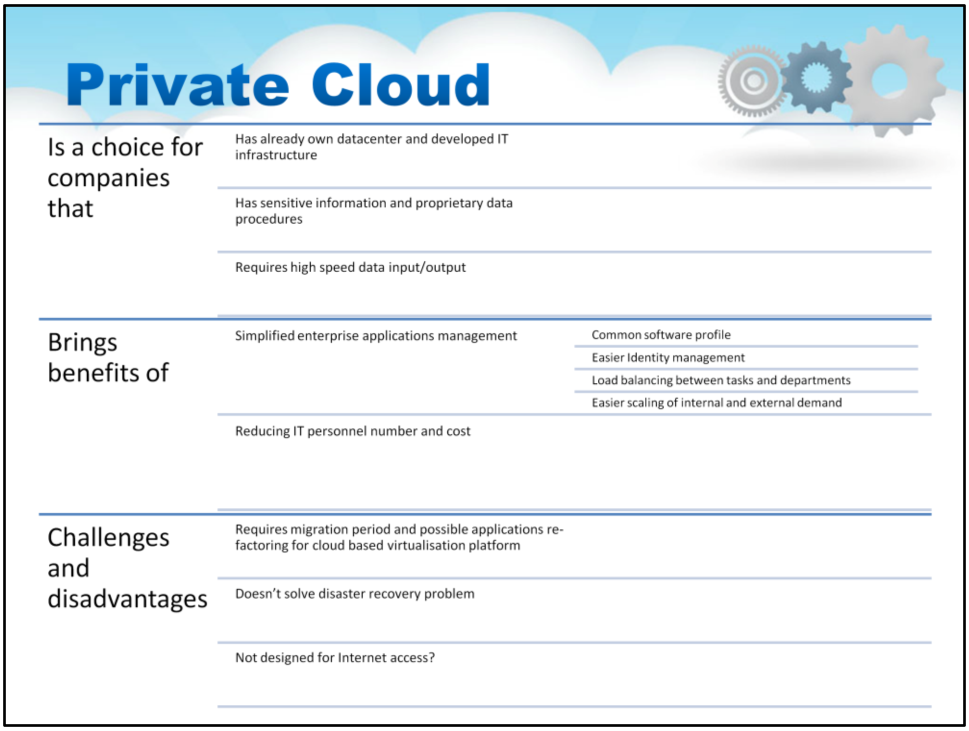


Cloud Deployment Models

–

Private Cloud

Private Cloud



Private Cloud

Is a choice for companies that

- Has already own datacenter and developed IT infrastructure
- Has sensitive information and proprietary data procedures
- Requires high speed data input/output

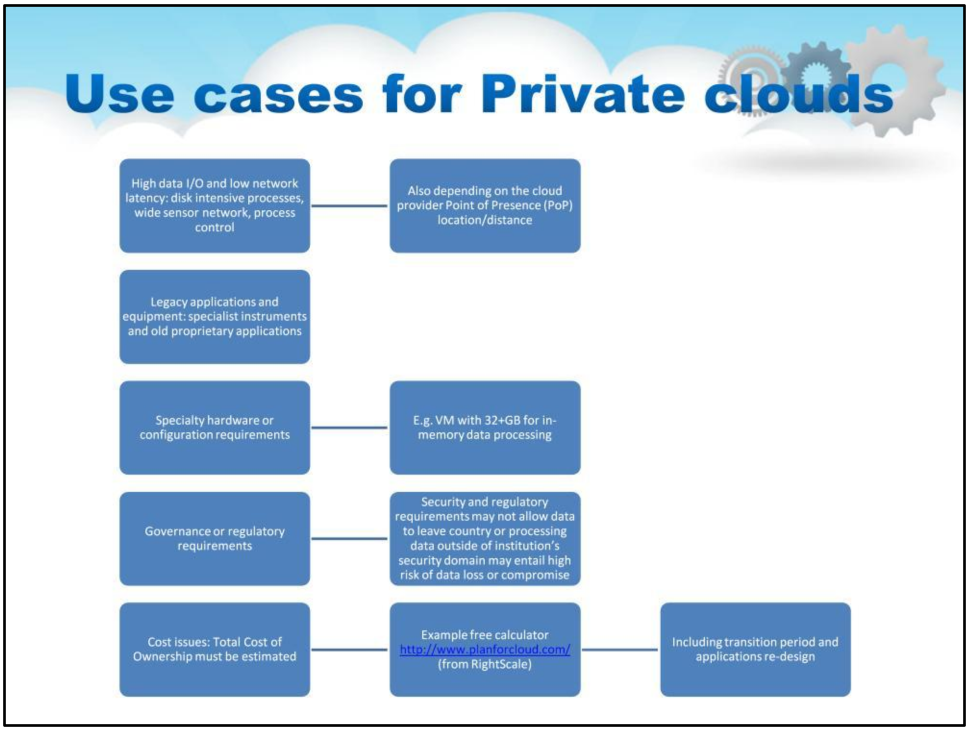
Brings benefits of

- Simplified enterprise applications management
  - Common software profile
  - Easier Identity management
  - Load balancing between tasks and departments
  - Easier scaling of internal and external demand
- Reducing IT personnel number and cost

Challenges and disadvantages

- Requires migration period and possible applications re-factoring for cloud based virtualisation platform
- Doesn’t solve disaster recovery problem
- Not designed for Internet access?

The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises



Use cases for Private clouds

High data I/O and low network latency: disk intensive processes, wide sensor network, process control

Also depending on the cloud provider Point of Presence (PoP) location/distance

Legacy applications and equipment: specialist instruments and old proprietary applications

Specialty hardware or configuration requirements

E.g. VM with 32+GB for in-memory data processing

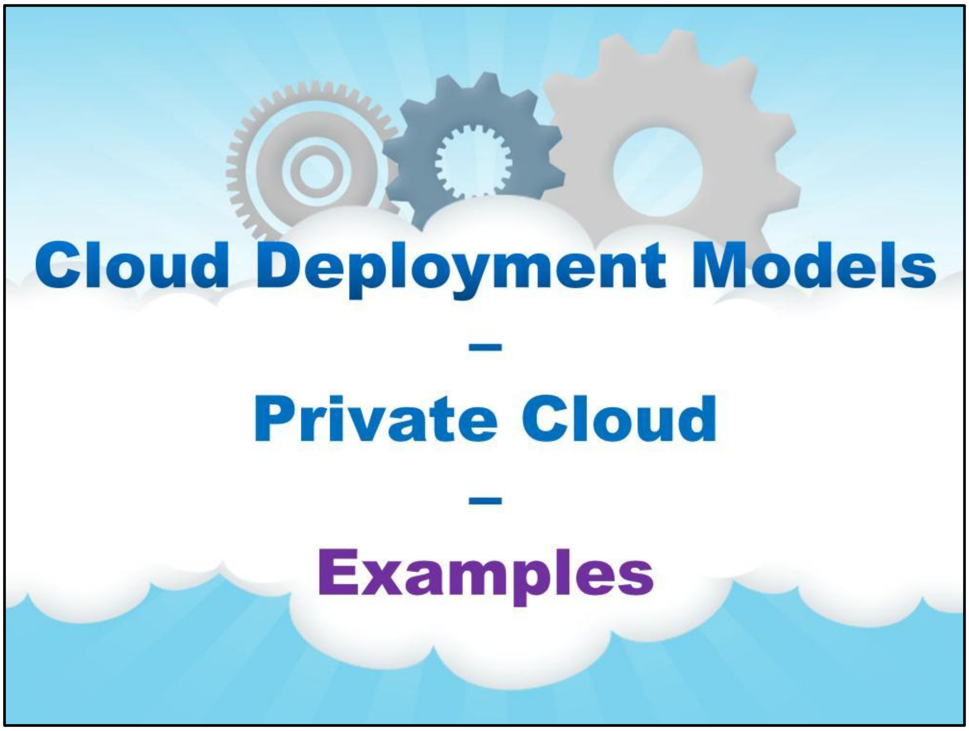
Governance or regulatory requirements

Security and regulatory requirements may not allow data to leave country or processing data outside of institution's security domain may entail high risk of data loss or compromise

Cost issues: Total Cost of Ownership must be estimated

Example free calculator http://www.planforcloud.com/ (from RightScale)

Including transition period and applications re-design



Cloud Deployment Models

–

Private Cloud

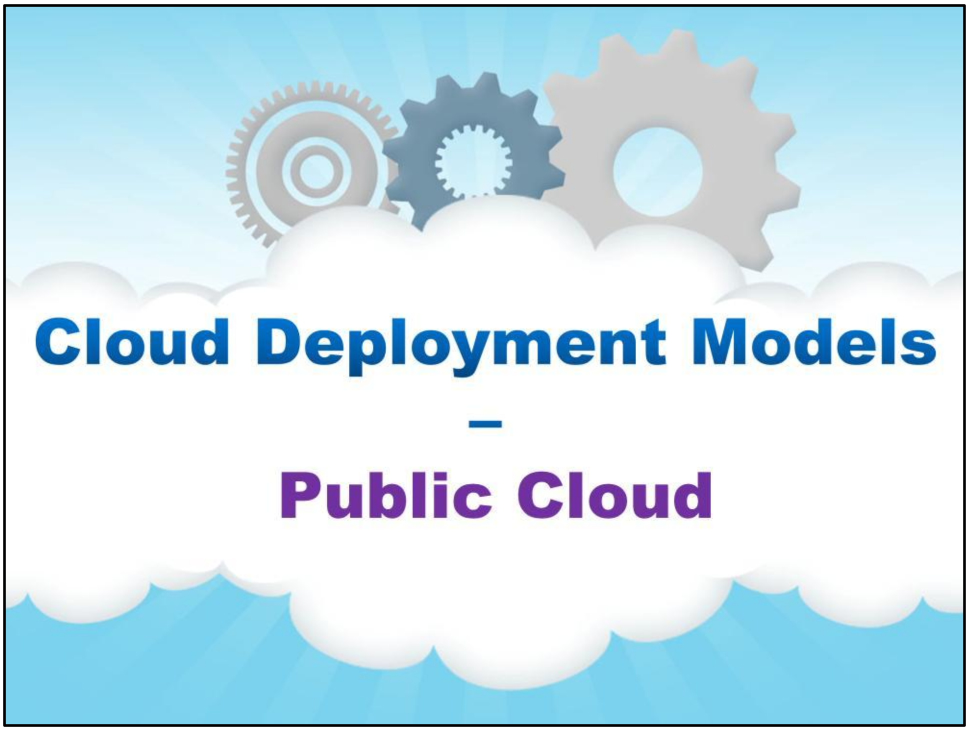
–

Examples

Examples

Examples

hp
NetIQ
ubuntu
ORACLE
amazon web services
Amazon VPC
CISCO
GOGRID
SUSE
We adapt. You succeed.
openstack
CLOUD SOFTWARE
apachecloudstack
open source cloud computing
DELL
Microsoft
rackspace
the open cloud company
CITRIX
Eucalyptus
vmware
nebula
Piston
PISTON CLOUD COMPUTING

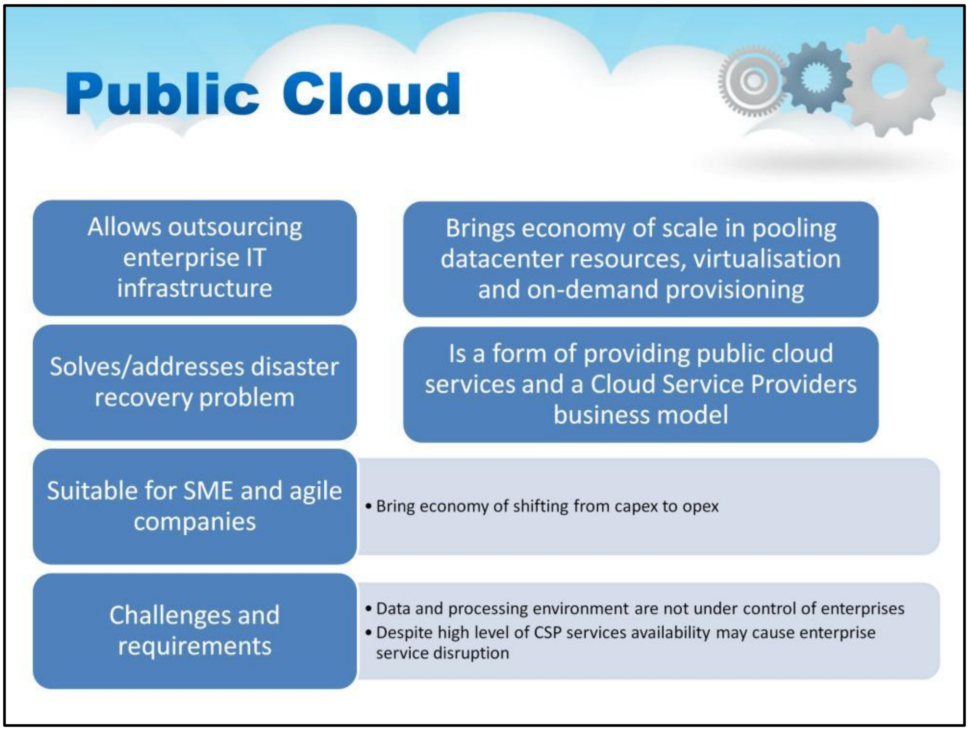


Cloud Deployment Models

–

Public Cloud

Public Cloud



Public Cloud

Allows outsourcing enterprise IT infrastructure

Brings economy of scale in pooling datacenter resources, virtualisation and on-demand provisioning

Solves/addresses disaster recovery problem

Is a form of providing public cloud services and a Cloud Service Providers business model

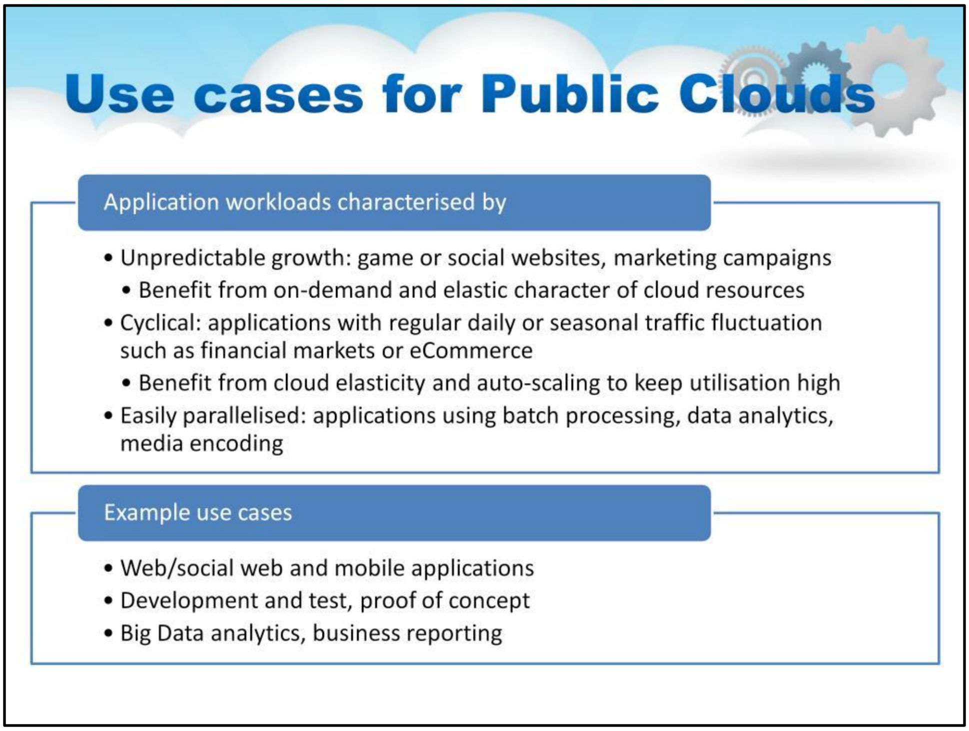
Suitable for SME and agile companies

- Bring economy of shifting from capex to opex

Challenges and requirements

- Data and processing environment are not under control of enterprises
- Despite high level of CSP services availability may cause enterprise service disruption

The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider



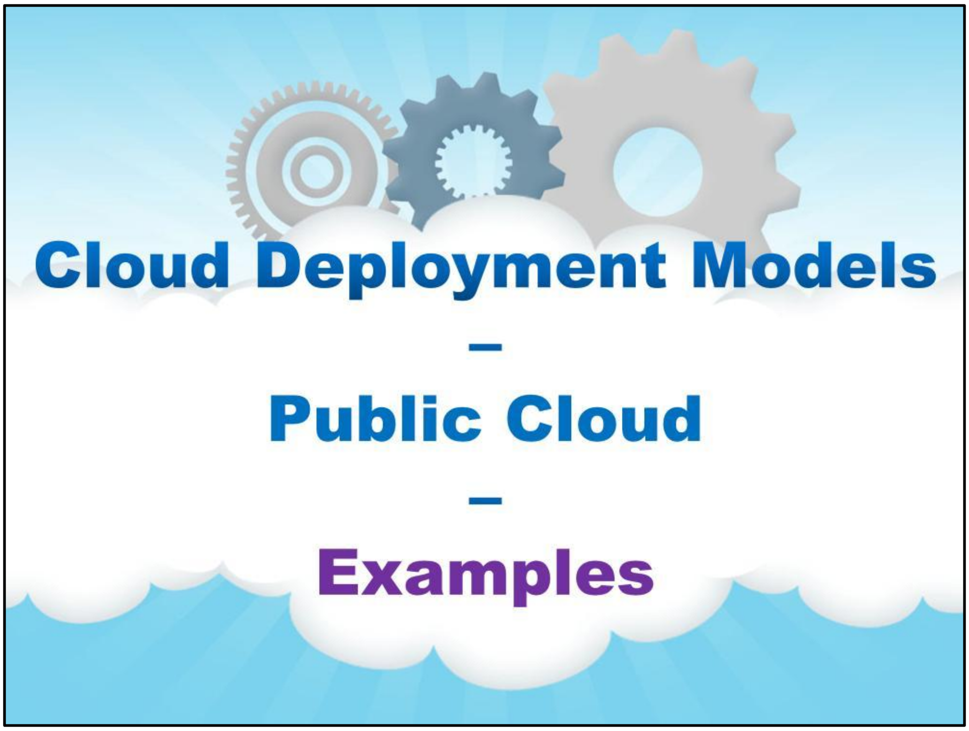
Use cases for Public Clouds

Application workloads characterised by

- Unpredictable growth: game or social websites, marketing campaigns
  - Benefit from on-demand and elastic character of cloud resources
- Cyclical: applications with regular daily or seasonal traffic fluctuation such as financial markets or eCommerce
  - Benefit from cloud elasticity and auto-scaling to keep utilisation high
- Easily parallelised: applications using batch processing, data analytics, media encoding

Example use cases

- Web/social web and mobile applications
- Development and test, proof of concept
- Big Data analytics, business reporting



Cloud Deployment Models

–

Public Cloud

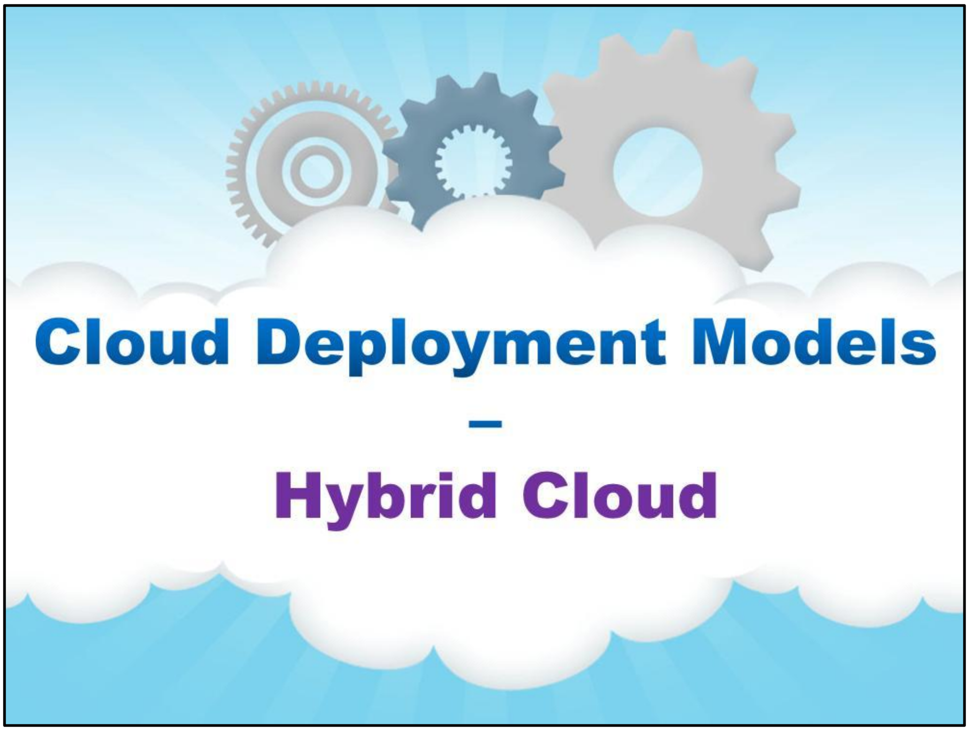
–

Examples

Examples

Examples

Microsoft Azure
IBM Cloud
Google Cloud Platform
amazon web services

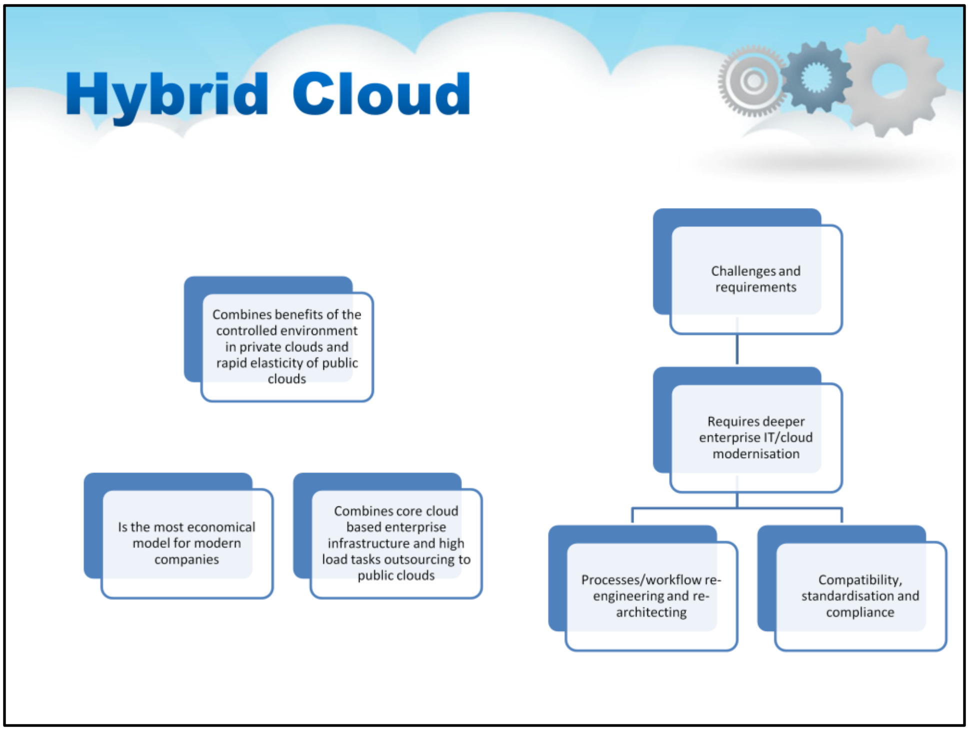


Cloud Deployment Models

–

Hybrid Cloud

Hybrid Cloud



Hybrid Cloud

Combines benefits of the controlled environment in private clouds and rapid elasticity of public clouds

Is the most economical model for modern companies

Combines core cloud based enterprise infrastructure and high load tasks outsourcing to public clouds

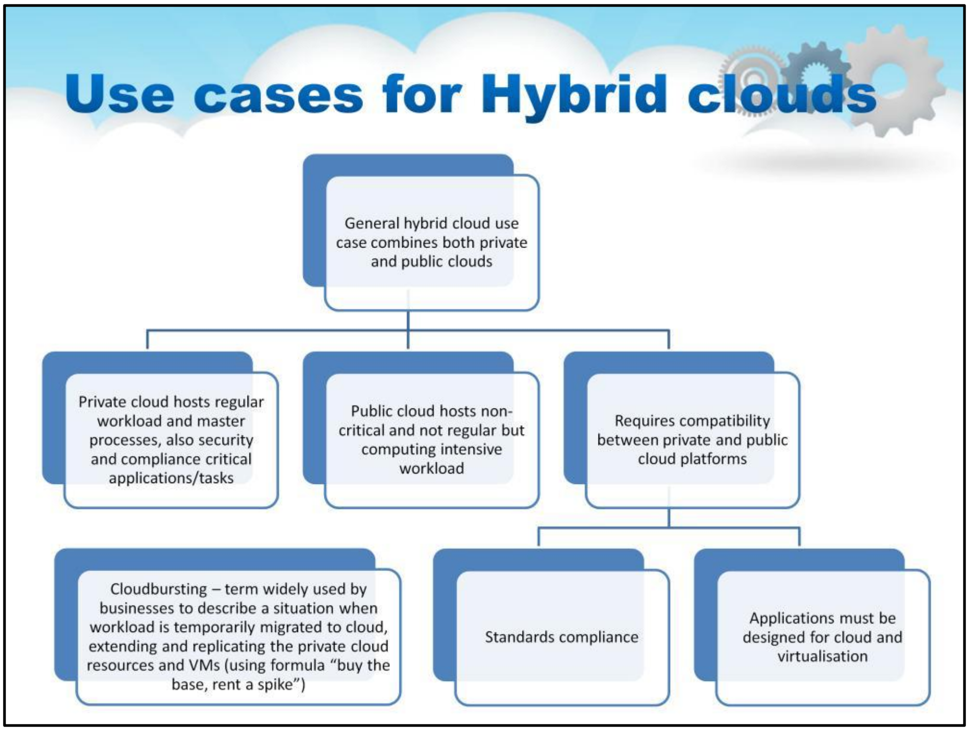
Challenges and requirements

Requires deeper enterprise IT/cloud modernisation

Processes/workflow re-engineering and re-architecting

Compatibility, standardisation and compliance

The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)



Use cases for Hybrid clouds

General hybrid cloud use case combines both private and public clouds

Private cloud hosts regular workload and master processes, also security and compliance critical applications/tasks

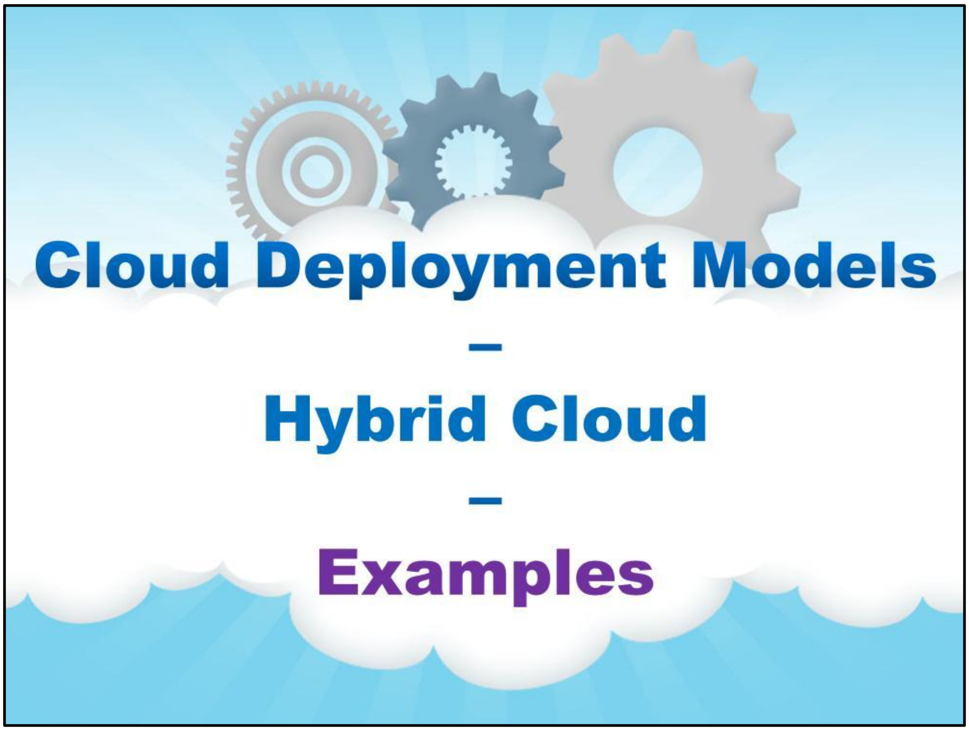
Public cloud hosts non-critical and not regular but computing intensive workload

Requires compatibility between private and public cloud platforms

Cloudbursting – term widely used by businesses to describe a situation when workload is temporarily migrated to cloud, extending and replicating the private cloud resources and VMs (using formula “buy the base, rent a spike”)

Standards compliance

Applications must be designed for cloud and virtualisation



Cloud Deployment Models

–

Hybrid Cloud

–

Examples

Examples

Examples

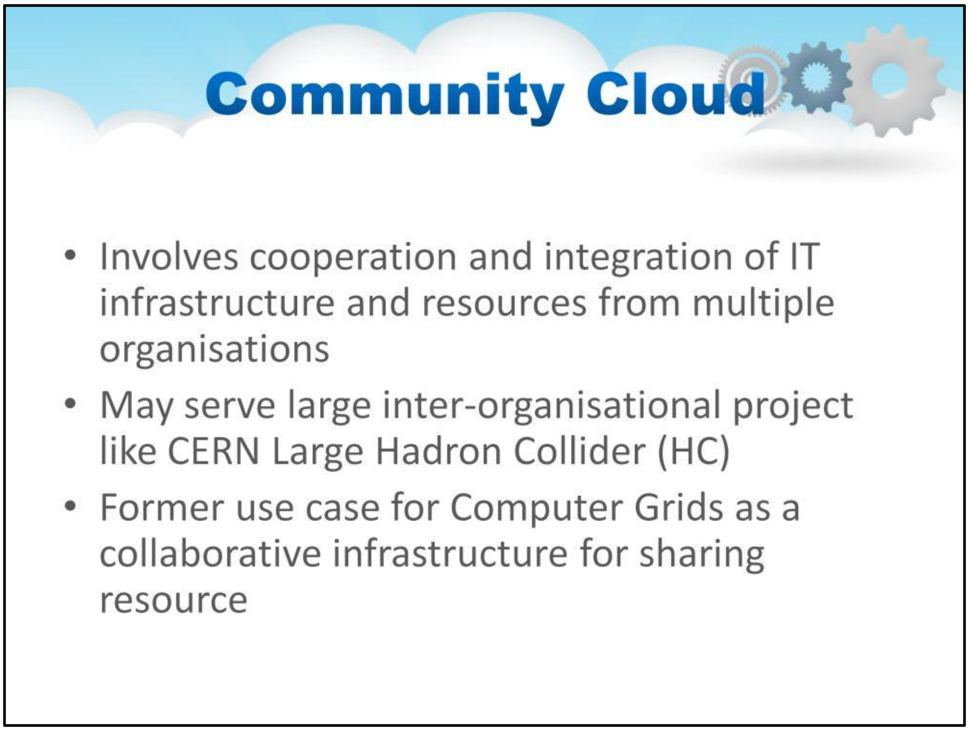
Microsoft
Hyper-V
DELL EMC
vmware
amazon
web services
rackspace

Cloud Deployment Models

–

Community Cloud

Community Cloud

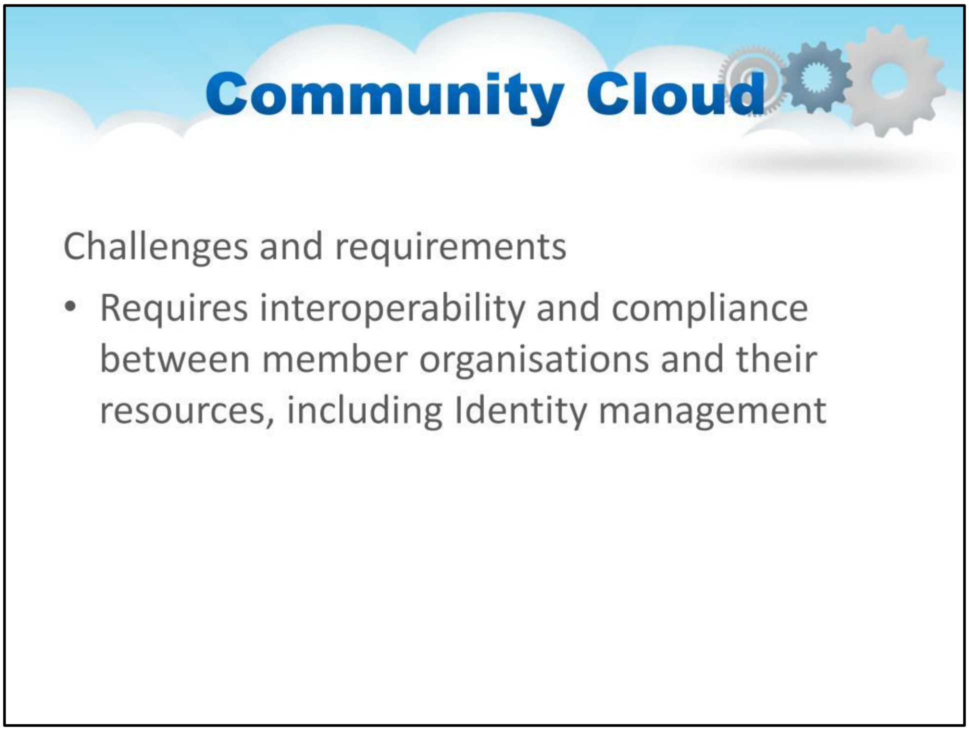


Community Cloud

- Involves cooperation and integration of IT infrastructure and resources from multiple organisations
- May serve large inter-organisational project like CERN Large Hadron Collider (HC)
- Former use case for Computer Grids as a collaborative infrastructure for sharing resource

[NIST SP800-145] The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

-Cloud platform simplifies creation and management of shared resources

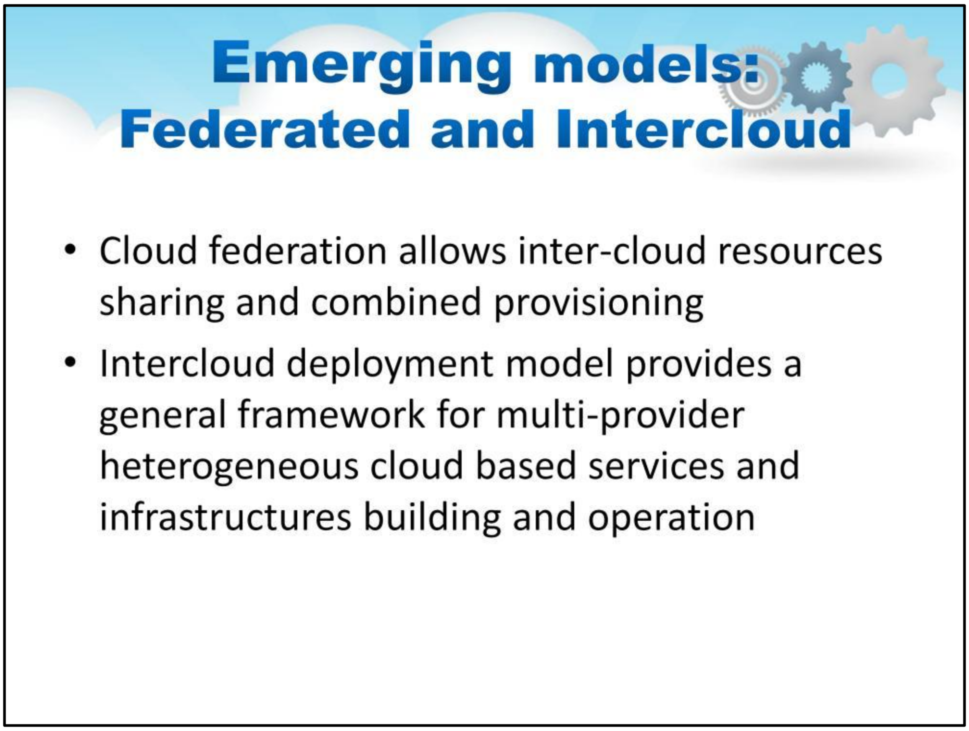


Community Cloud

Challenges and requirements

- Requires interoperability and compliance between member organisations and their resources, including Identity management

-Typically solved with Federated Clouds and Federated Identity management



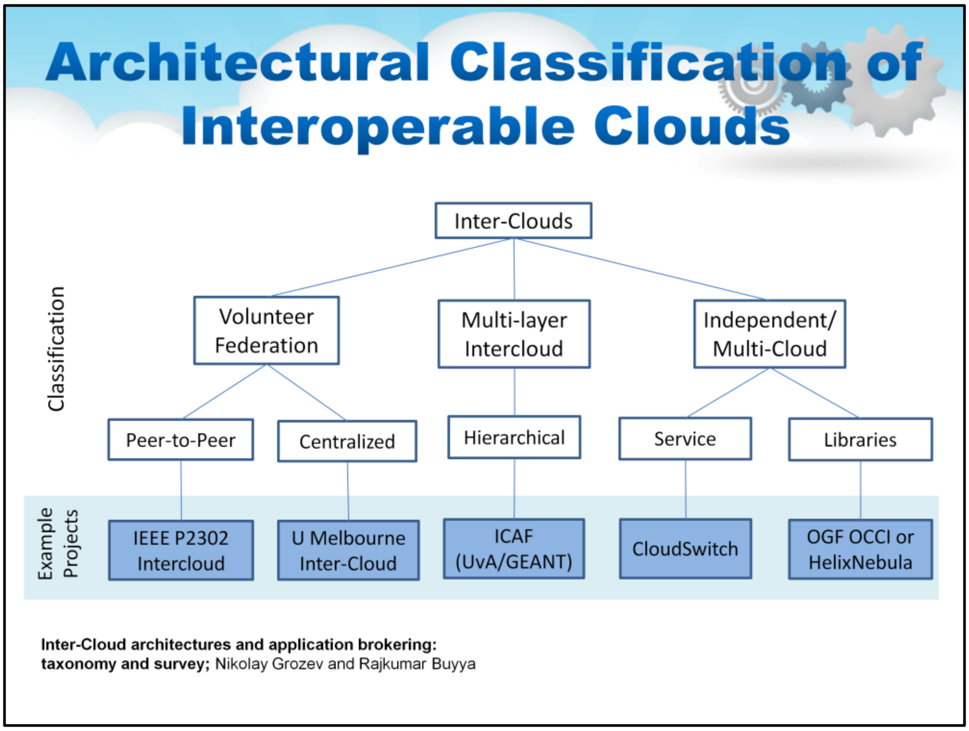
Emerging models: Federated and Intercloud

- Cloud federation allows inter-cloud resources sharing and combined provisioning
- Intercloud deployment model provides a general framework for multi-provider heterogeneous cloud based services and infrastructures building and operation

Two type of cloud federation

-Provider side federation for resources sharing and provisioning
-User/customer side federation that allows creation of multi-provider heterogeneous cloud infrastructures

-Allows hierarchical IaaS-PaaS-SaaS cloud integration
-Requires and is possible when using well defined cloud standards
-Should address issues: Control and management, intercloud federations management, operations issues



Architectural Classification of Interoperable Clouds

Classification

Inter-Clouds

Volunteer Federation

Multi-layer Intercloud

Independent/ Multi-Cloud

Peer-to-Peer

Centralized

Hierarchical

Service

Libraries

Example Projects

IEEE P2302 Intercloud

U Melbourne Inter-Cloud

ICAF (UvA/GEANT)

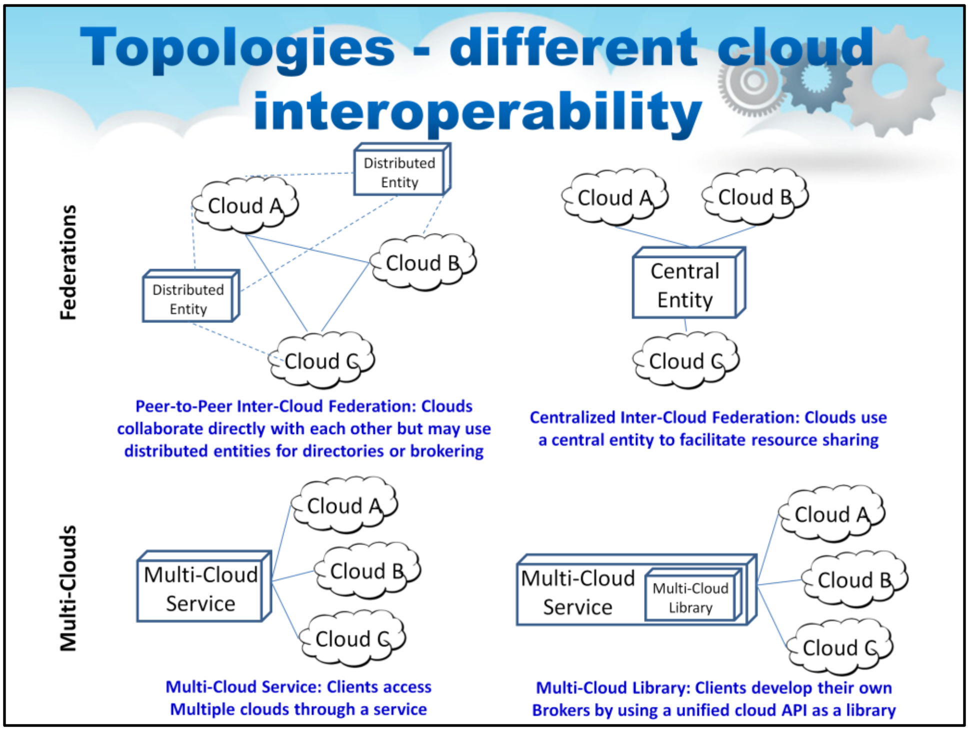
CloudSwitch

OGF OCCI or HelixNebula

Inter-Cloud architectures and application brokering: taxonomy and survey; Nikolay Grozev and Rajkumar Buyya

There are many ways clouds can interoperate. This slide uses an illustration from a research group from University of Melbourne, who polished a paper on the taxonomy of different cloud interoperability approaches. Please study the illustration.

Intercloud Architecture Framework (ICAF) is based on the multi-layer Cloud Services Model (CMS) and separates control, management, federation and operational aspects in Intercloud. Developed by UvA, currently implemented in the European project GEANT.



Topologies - different cloud interoperability

Peer-to-Peer Inter-Cloud Federation: Clouds collaborate directly with each other but may use distributed entities for directories or brokering

Centralized Inter-Cloud Federation: Clouds use a central entity to facilitate resource sharing

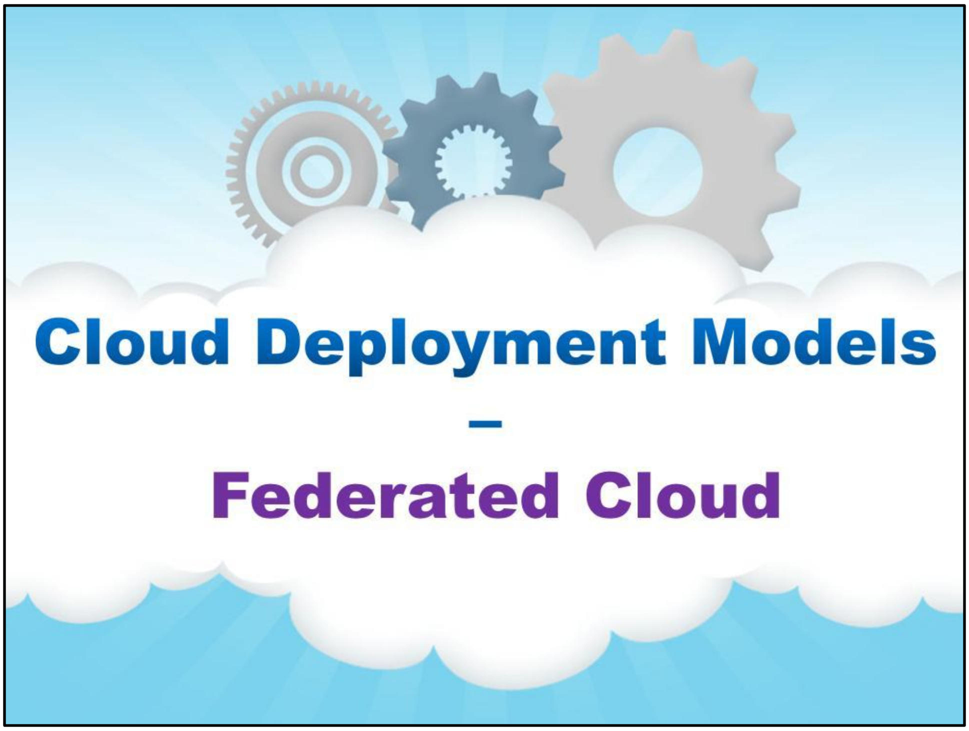
Multi-Cloud Service: Clients access Multiple clouds through a service

Multi-Cloud Library: Clients develop their own Brokers by using a unified cloud API as a library

It is easier to understand these different cloud interoperability approaches when one visualizes the resultant topologies. The illustration in this slide shows the same four categories from the previous slide but from a topology view.

It is easy to see that the multi-cloud approach will work for a user wanting to access a small number of clouds more or less manually and must “string together” the resources from each cloud themselves. This approach is ideal for scientists who want to access simple cloud resources (like VMs) to create large clusters for Hadoop or for Grid calculations. The responsibility for understanding the variations amongst the clouds falls with the user in this case.

It is easy to see that the Federations approach is a much more scalable and interconnected architecture. This approach is ideal for systems which might span multiple cloud providers, or span private and public clouds. It also places the responsibility of understanding the variations amongst the clouds with the infrastructure, not the end user. This is an important point which we will examine more later.

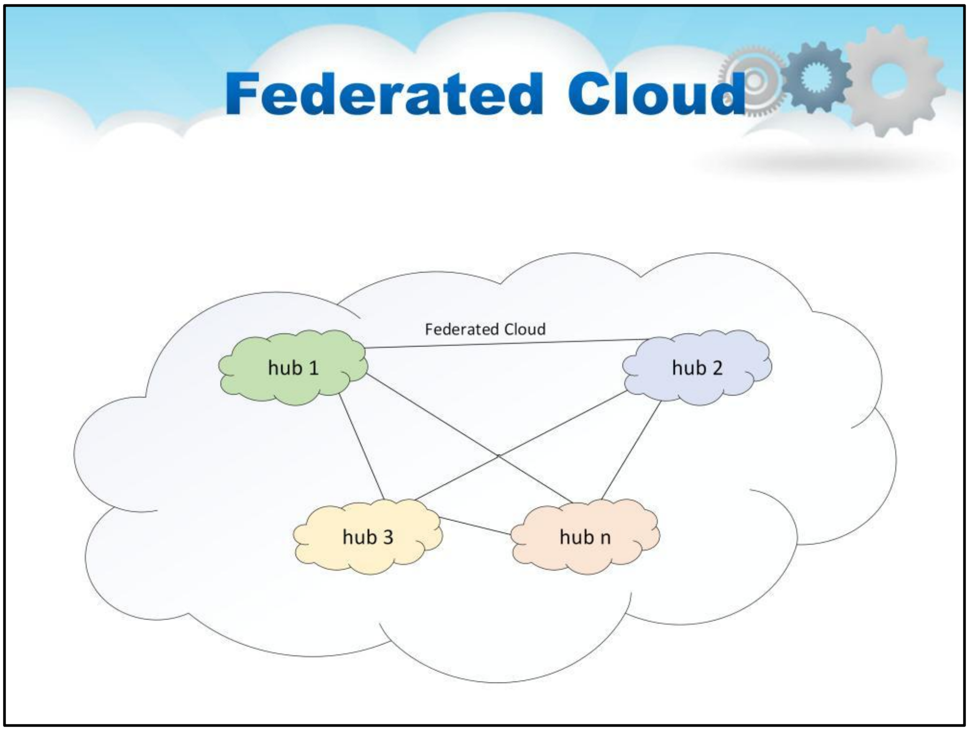


Cloud Deployment Models

–

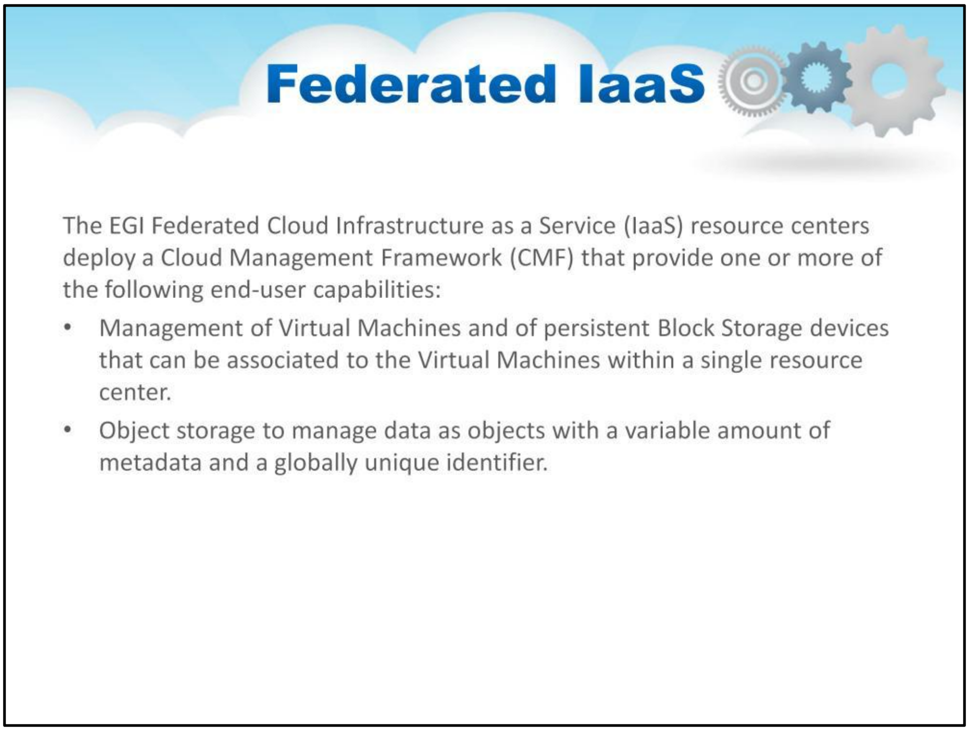
Federated Cloud

Federated Cloud



Federated Cloud

A federated cloud refers to a business model in which multiple external and internal cloud computing services deployed and managed to achieve a common goal.
A Federated cloud (also called cloud federation) is the deployment and management of external and internal cloud computing services to match business needs.
A federation is the union of several small parts that perform a common action.



Federated IaaS

The EGI Federated Cloud Infrastructure as a Service (IaaS) resource centers deploy a Cloud Management Framework (CMF) that provide one or more of the following end-user capabilities:

- Management of Virtual Machines and of persistent Block Storage devices that can be associated to the Virtual Machines within a single resource center.
- Object storage to manage data as objects with a variable amount of metadata and a globally unique identifier.

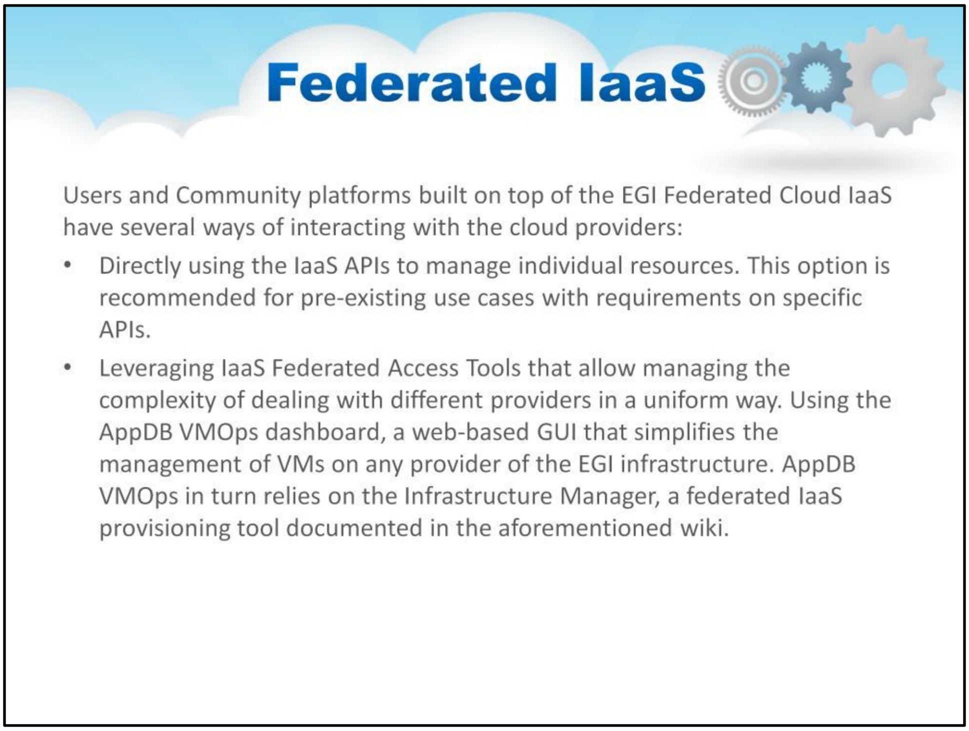
These end-user capabilities must be provided via community agreed APIs that can be integrated with the following EGI services:

AAI to provide Single Sign-On for authentication and authorization across the whole cloud federation.

Configuration Database, to record information about the topology of the e-infrastructure.

Accounting to collect, aggregate and display usage information.

Monitoring to perform federated service availability monitoring and reporting of the distributed cloud service endpoints, and to retrieve this information programmatically. Integration with monitoring is a passive activity of the resource center, the monitoring is performed using the end-user APIs with regular user credentials from EGI AAI.



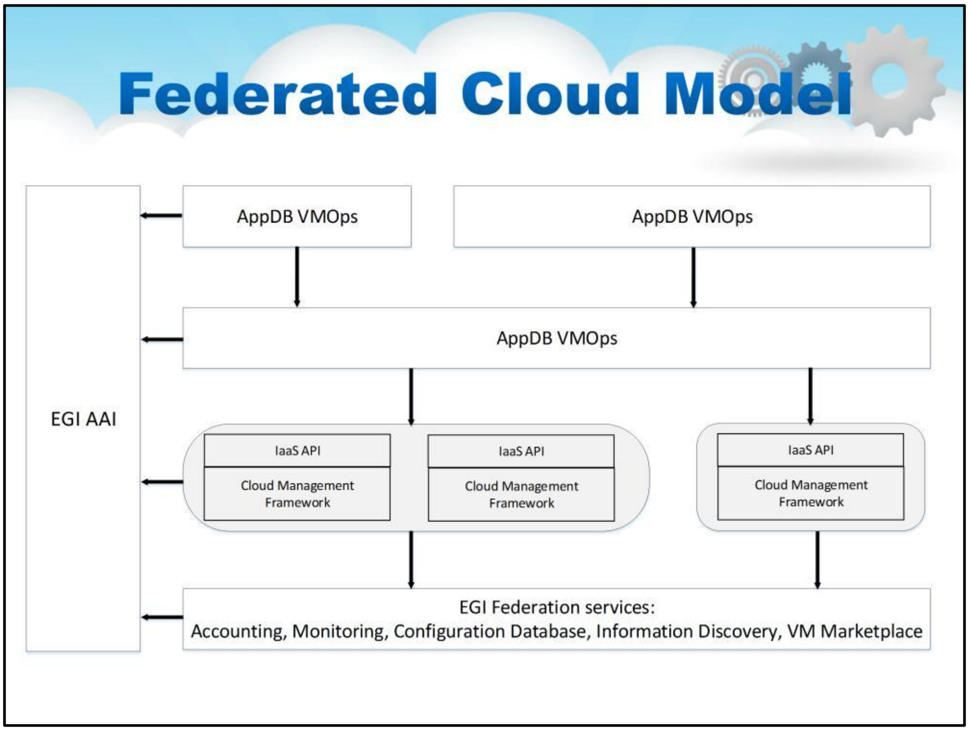
Federated IaaS

Users and Community platforms built on top of the EGI Federated Cloud IaaS have several ways of interacting with the cloud providers:

- Directly using the IaaS APIs to manage individual resources. This option is recommended for pre-existing use cases with requirements on specific APIs.
- Leveraging IaaS Federated Access Tools that allow managing the complexity of dealing with different providers in a uniform way. Using the AppDB VMOps dashboard, a web-based GUI that simplifies the management of VMs on any provider of the EGI infrastructure. AppDB VMOps in turn relies on the Infrastructure Manager, a federated IaaS provisioning tool documented in the aforementioned wiki.

These tools include

IaaS provisioning systems that allow to define infrastructure as code and manage and combine resources from different providers, thus enabling the portability of application deployments between them (e.g. IM or Terraform);
Cloud brokers, that provide matchmaking for workloads to available providers (e.g. the INDIGO-DataCloud Orchestrator); and
Cloud Management Software that provides a unified console for accessing resources and deploy workloads following a set of user-defined established policies (e.g. Scalr or RightScale)



Federated Cloud Model

EGI AAI

AppDB VMOps

AppDB VMOps

AppDB VMOps

IaaS API

Cloud Management Framework

IaaS API

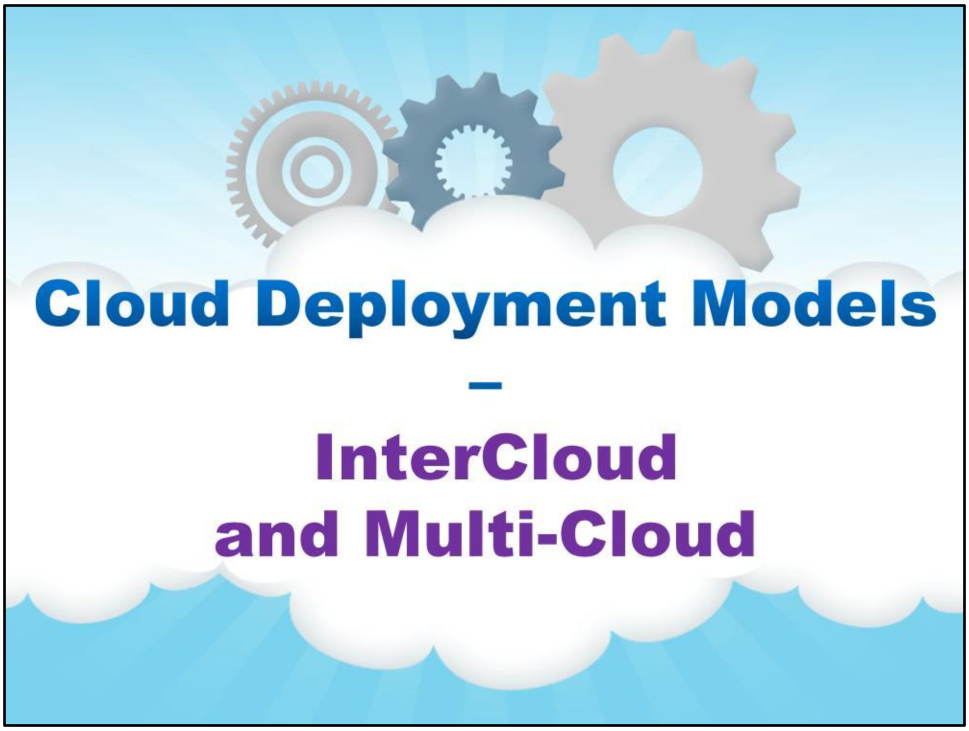
Cloud Management Framework

IaaS API

Cloud Management Framework

EGI Federation services:
Accounting, Monitoring, Configuration Database, Information Discovery, VM Marketplace

EGI does not mandate deploying any particular or specific Cloud Management Framework; it is the responsibility of the Resource Center to investigate, identify and deploy the solution that fits best their individual needs whilst ensuring that the offered services implement the required interfaces and domain languages of the federation realms they are member of.

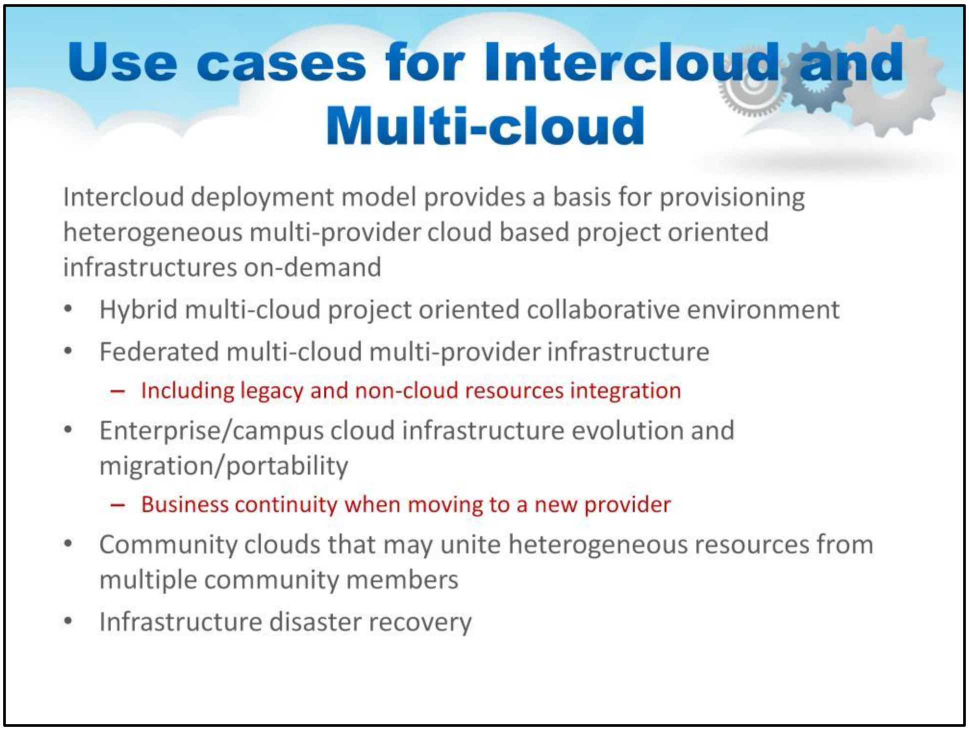


Cloud Deployment Models

–

InterCloud
and Multi-Cloud

Inter Cloud and Multi-Cloud

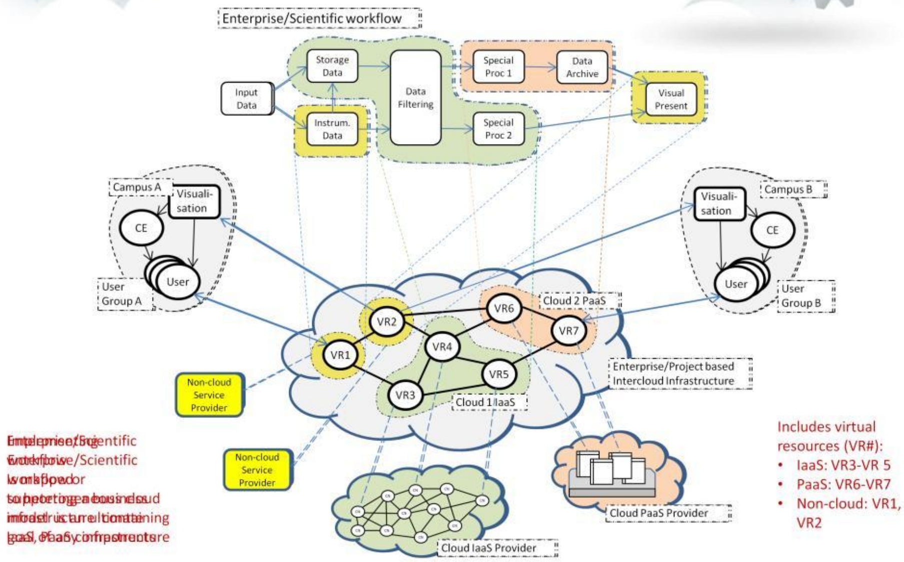


Use cases for Intercloud and Multi-cloud

Intercloud deployment model provides a basis for provisioning heterogeneous multi-provider cloud based project oriented infrastructures on-demand

- Hybrid multi-cloud project oriented collaborative environment
- Federated multi-cloud multi-provider infrastructure
  - Including legacy and non-cloud resources integration
- Enterprise/campus cloud infrastructure evolution and migration/portability
  - Business continuity when moving to a new provider
- Community clouds that may unite heterogeneous resources from multiple community members
- Infrastructure disaster recovery

Use case: Intercloud infrastructure provisioning: Workflow => Logical (Cloud) Infrastructure

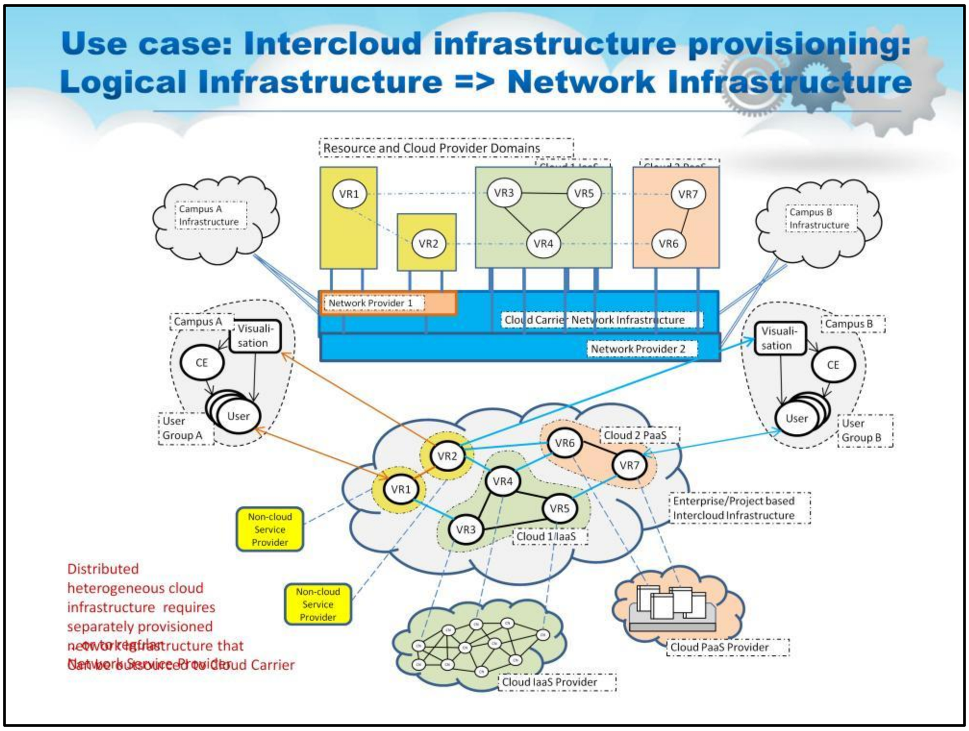


Enterprise/Scientific workflow is mapped to heterogeneous cloud infrastructure containing IaaS, PaaS components

Implementation of Enterprise/Scientific workflow for supporting a business [illegible]

Includes virtual resources (VR#):

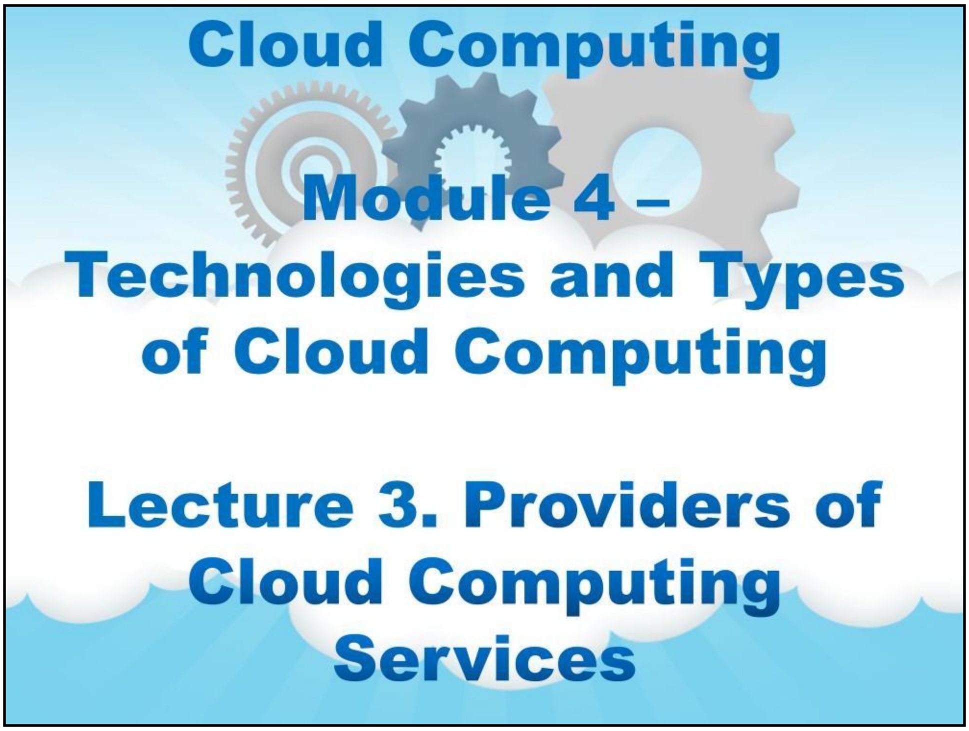
- IaaS: VR3-VR 5
- PaaS: VR6-VR7
- Non-cloud: VR1, VR2



Use case: Intercloud infrastructure provisioning: Logical Infrastructure => Network Infrastructure

Distributed heterogeneous cloud infrastructure requires separately provisioned network infrastructure that Can be outsourced to Cloud Carrier

– or to regular Network Service Provider



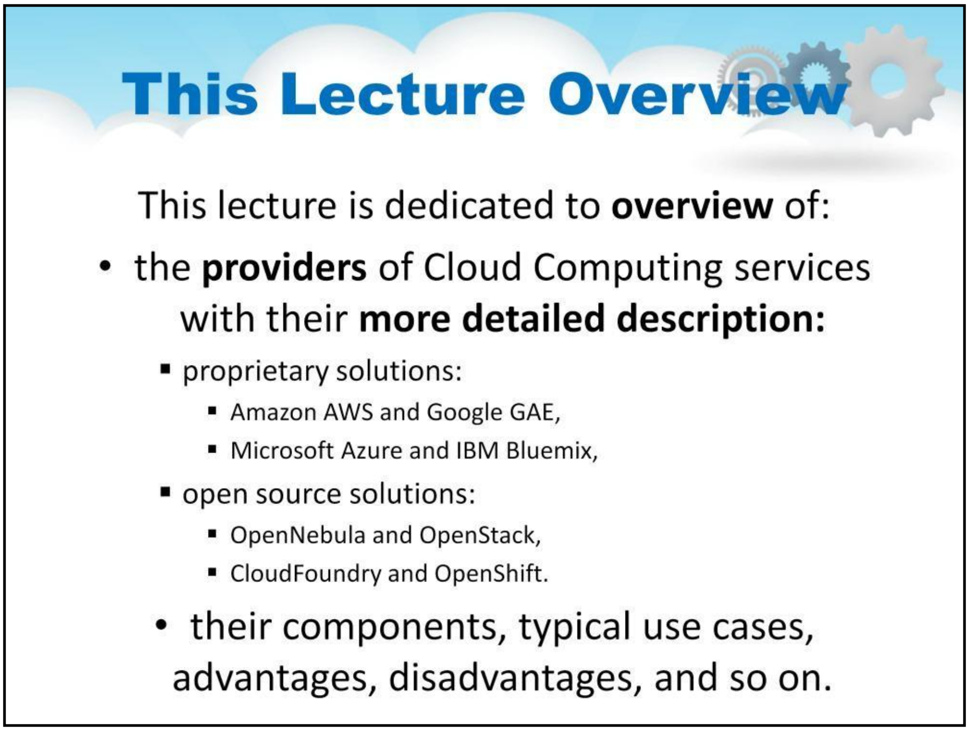
Cloud Computing

Module 4 – Technologies and Types of Cloud Computing

Lecture 3. Providers of Cloud Computing Services

The title of this module is: “Technologies and Types of Cloud Computing”.

This lecture 3 is about “Providers of Cloud Computing Services”.



This Lecture Overview

This lecture is dedicated to **overview** of:

- the **providers** of Cloud Computing services with their **more detailed description:**
  - proprietary solutions:
    - Amazon AWS and Google GAE,
    - Microsoft Azure and IBM Bluemix,
  - open source solutions:
    - OpenNebula and OpenStack,
    - CloudFoundry and OpenShift.
- their components, typical use cases, advantages, disadvantages, and so on.

Lecture 3. Providers of Cloud Computing Services

This lecture is about:

the **providers** of Cloud Computing services with the **more detailed description** of:

❑ proprietary (commercial) solutions:
- Amazon AWS and Google GAE,
- Microsoft Azure and IBM Bluemix,

❑ open source solutions:
- OpenNebula and OpenStack,
- CloudFoundry and OpenShift.

Here their components, typical use cases, advantages, disadvantages, etc. are described in details.

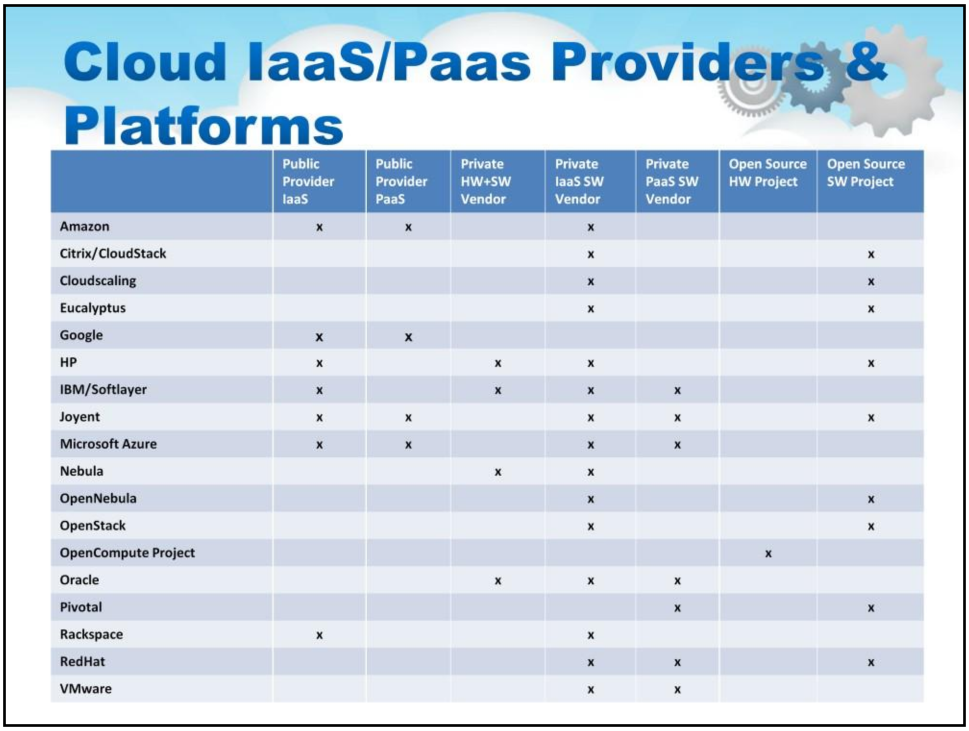
Cloud Service Providers

–

Overview

Overview

Let’s start from the overview of cloud service providers …



Cloud IaaS/Paas Providers & Platforms

	Public Provider IaaS	Public Provider PaaS	Private HW+SW Vendor	Private IaaS SW Vendor	Private PaaS SW Vendor	Open Source HW Project	Open Source SW Project
Amazon	x	x		x			
Citrix/CloudStack				x			x
Cloudscaling				x			x
Eucalyptus				x			x
Google	x	x					
HP	x		x	x			x
IBM/Softlayer	x		x	x	x		
Joyent	x	x		x	x		x
Microsoft Azure	x	x		x	x		
Nebula			x	x			
OpenNebula				x			x
OpenStack				x			x
OpenCompute Project						x	
Oracle			x	x	x		
Pivotal					x		x
Rackspace	x			x			
RedHat				x	x		x
VMware				x	x		

This chart lists the major players in the Cloud Computing technology space. While it is not full one thing that is evident is that there are a large number of major companies who are investing significant R&D in Cloud offerings.

The chart shows the different delivery ways (public cloud services, or private cloud software).

The chart also shows the layers of Cloud technology (Hardware, IaaS, or PaaS)

Finally the chart indicates whether the vendor makes available their offering as Open Source as a key strategy.

Note:
some SaaS providers with domain specific PaaS environments, such as Salesforce.com, or Netsuite, or Workday, have been excluded.

That is why the selection of the optimal technology direction can be complicated in this context!

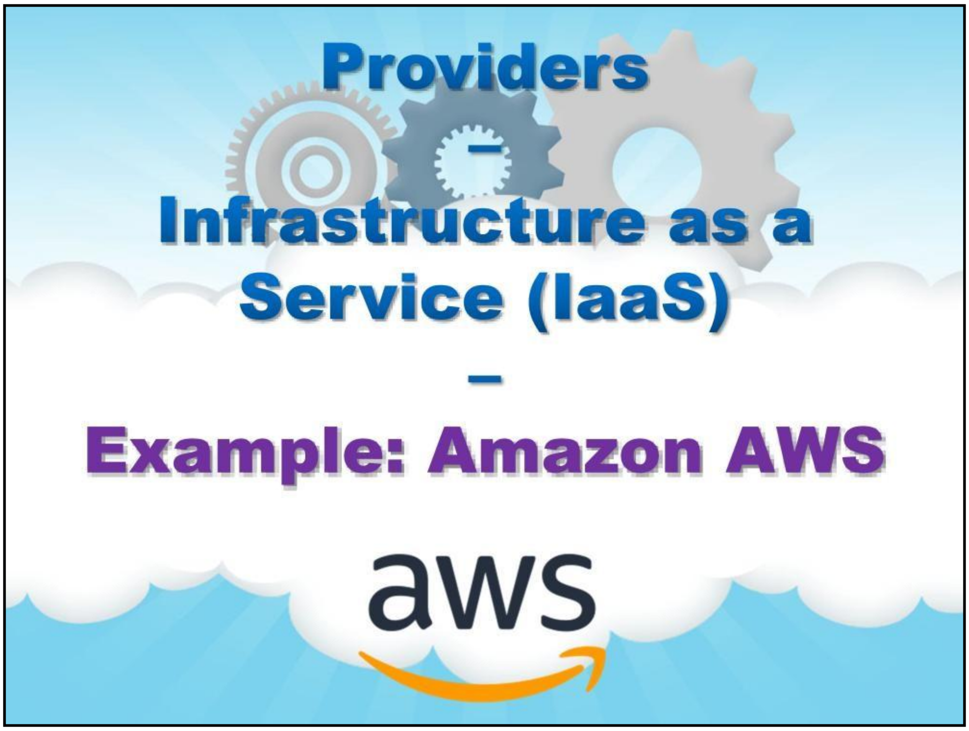
Cloud Service Providers

–

Proprietary Solutions

Proprietary Solutions

Let’s consider some commercial solutions…



Providers

–

Infrastructure as a Service (IaaS)

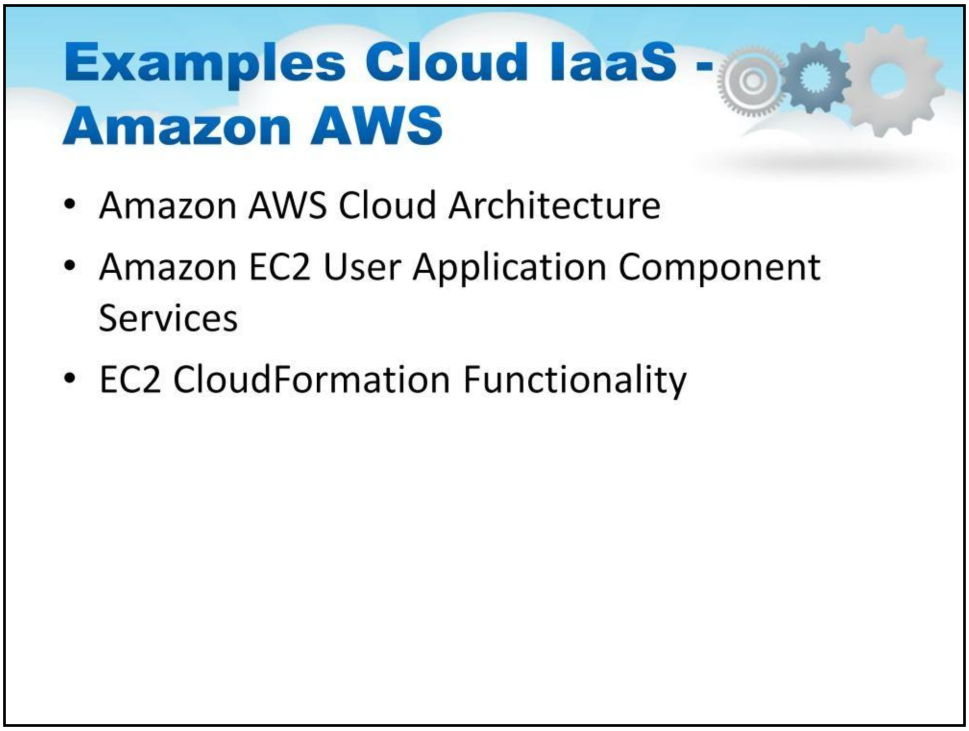
–

Example: Amazon AWS

aws

Example: Amazon AWS

The fist example is dedicated to Amazon AWS that represents an example of Infrastructure as a Service.



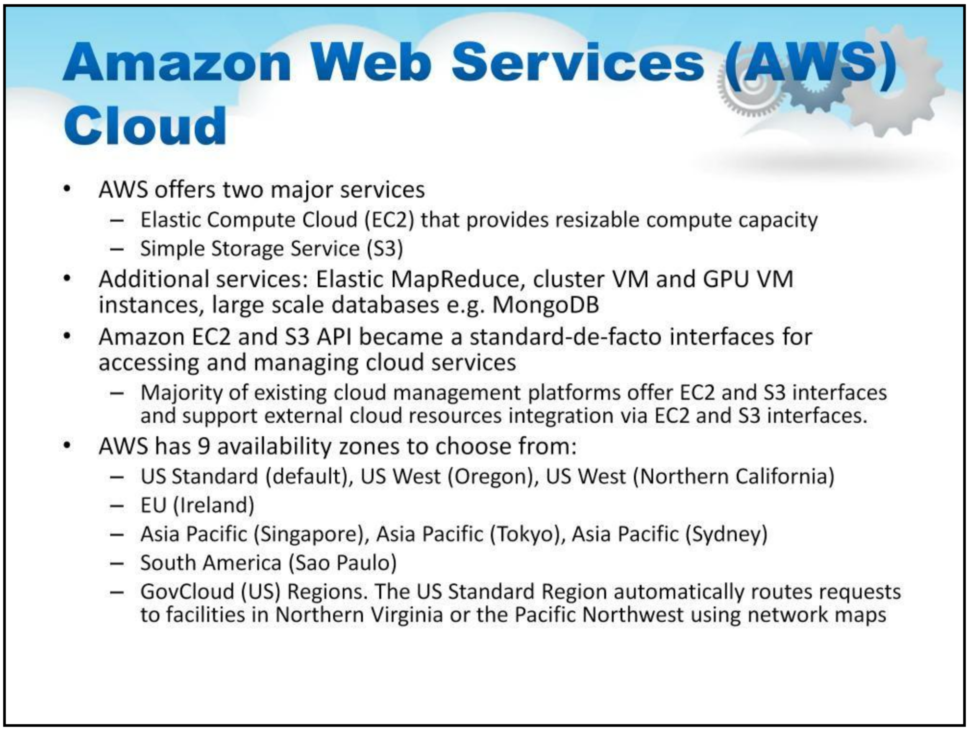
Examples Cloud IaaS - Amazon AWS

- Amazon AWS Cloud Architecture
- Amazon EC2 User Application Component Services
- EC2 CloudFormation Functionality

The most popular IaaS cloud is Amazon AWS. For that reason we will reference this platform in providing example solutions.

The AWS architecture contains many, many services. The first and most basic ones are “EC2” computing and the associated core parts that come with it (S3 and EBS storage for example).

Therefore let us investigate AWS more.



Amazon Web Services (AWS) Cloud

- AWS offers two major services
  - Elastic Compute Cloud (EC2) that provides resizable compute capacity
  - Simple Storage Service (S3)
- Additional services: Elastic MapReduce, cluster VM and GPU VM instances, large scale databases e.g. MongoDB
- Amazon EC2 and S3 API became a standard-de-facto interfaces for accessing and managing cloud services
  - Majority of existing cloud management platforms offer EC2 and S3 interfaces and support external cloud resources integration via EC2 and S3 interfaces.
- AWS has 9 availability zones to choose from:
  - US Standard (default), US West (Oregon), US West (Northern California)
  - EU (Ireland)
  - Asia Pacific (Singapore), Asia Pacific (Tokyo), Asia Pacific (Sydney)
  - South America (Sao Paulo)
  - GovCloud (US) Regions. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps

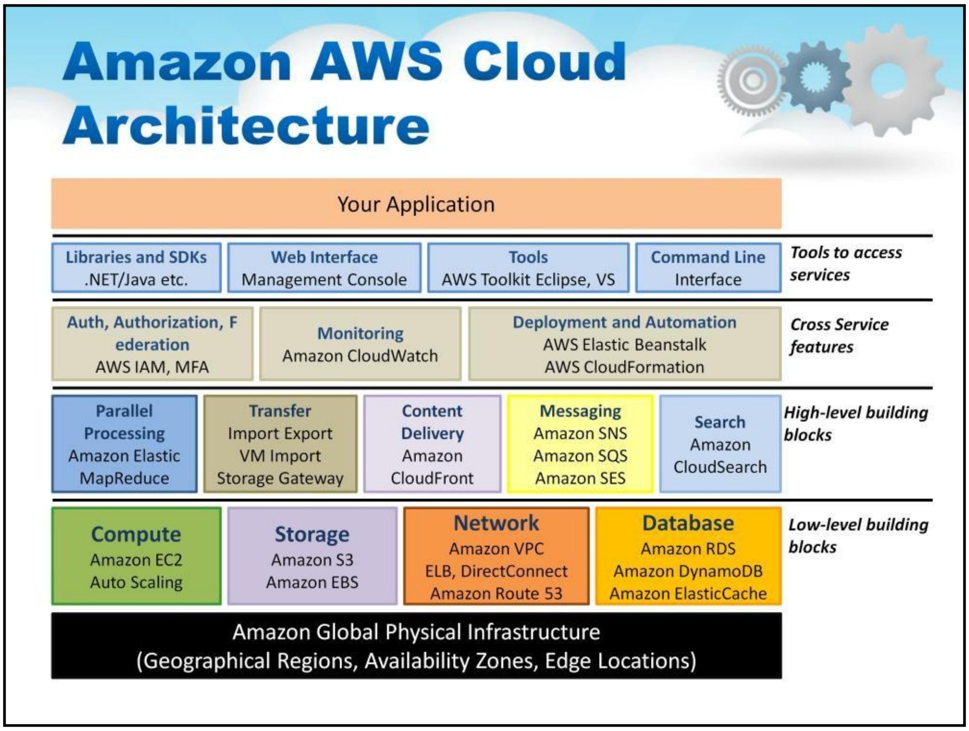
AWS offers two major services

Elastic Compute Cloud (EC2) that provides resizable compute capacity
Simple Storage Service (S3)

AWS also offer many many other services, For example, all kinds of databases are offered, many kinds of VM’s, and middleware components, all “as a service”.

Amazon EC2 and S3 API became a standard-de-facto interfaces for accessing and managing cloud services

AWS has several availability zones to choose from.



Amazon AWS Cloud Architecture

Your Application

Libraries and SDKs .NET/Java etc.
Web Interface Management Console
Tools AWS Toolkit Eclipse, VS
Command Line Interface
Tools to access services

Auth, Authorization, F ederation AWS IAM, MFA
Monitoring Amazon CloudWatch
Deployment and Automation AWS Elastic Beanstalk AWS CloudFormation
Cross Service features

Parallel Processing Amazon Elastic MapReduce
Transfer Import Export VM Import Storage Gateway
Content Delivery Amazon CloudFront
Messaging Amazon SNS Amazon SQS Amazon SES
Search Amazon CloudSearch
High-level building blocks

Compute Amazon EC2 Auto Scaling
Storage Amazon S3 Amazon EBS
Network Amazon VPC ELB, DirectConnect Amazon Route 53
Database Amazon RDS Amazon DynamoDB Amazon ElasticCache
Low-level building blocks

Amazon Global Physical Infrastructure
(Geographical Regions, Availability Zones, Edge Locations)

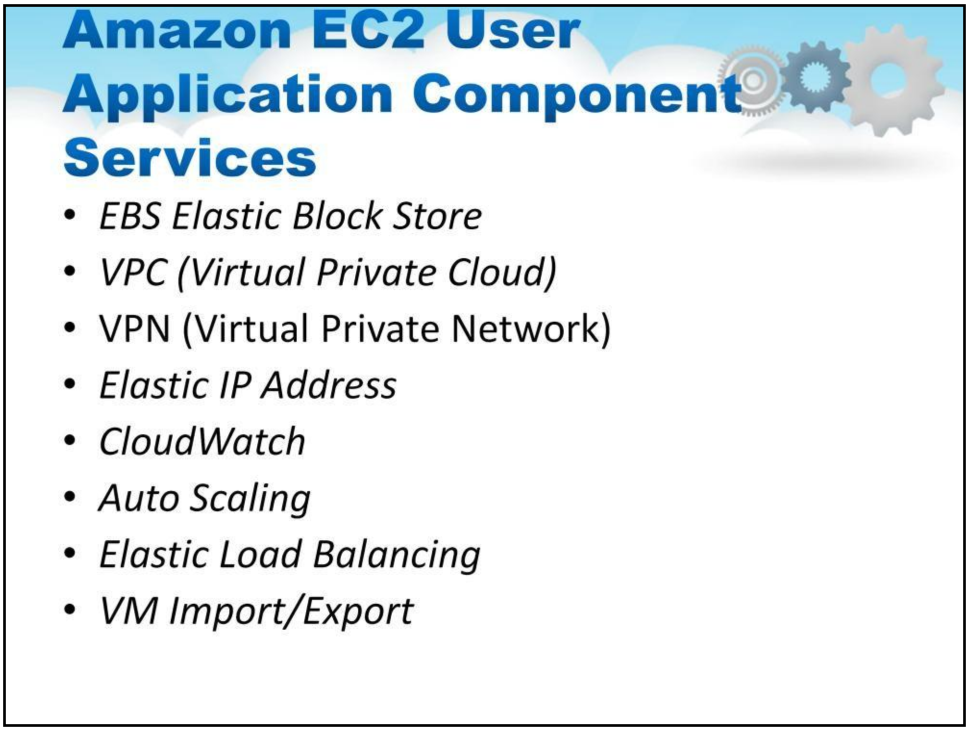
This is the “conceptual architecture” of AWS.
This illustration is a convenient way to get impression about layers and groups of the various AWS services.

Note:
The low level building blocks contain the core AWS functions as well as options in other core functions like DNS, networking, and cache.

The higher level building blocks are more recently constructed, and implement

higher level functions such as search and content delivery.
Cross service features include security related mechanisms like authentication, monitoring, and application control.

AWS lets one access it through the original low level commands and interfaces to tools supplied.



Amazon EC2 User Application Component Services

- *EBS Elastic Block Store*
- *VPC (Virtual Private Cloud)*
- VPN (Virtual Private Network)
- *Elastic IP Address*
- *CloudWatch*
- *Auto Scaling*
- *Elastic Load Balancing*
- *VM Import/Export*

This slide illustrates the AWS cloud architecture that can be viewed as a reference IaaS cloud implementation. It contains all the major functional components of the generic Cloud IaaS architecture.

Elastic Block Store (EBS) provides highly available and reliable storage volumes that are independent and persistent across the life of an VM/AMI

These volumes can be used as a boot partition for a particular instance or as a storage space with backend replication for long-term durability

Virtual Private Cloud (VPC) allows organizations to use AWS resources along with their existing infrastructure in a **VPN (Virtual Private Network)** to increase compute capabilities beyond the local resources.

Elastic IP Address is a persistent IP address which can be allocated to a particular user account and allows the user to dynamically assign it to any user VM.

Can be configured to switch to a replacement instance in the event of a failure of the running VM

CloudWatch is a service to monitor AWS resource utilization, operational performance, and overall demand patterns

Auto Scaling allows users to dynamically provision and relinquish resources used by their applications depending upon demand in run-time

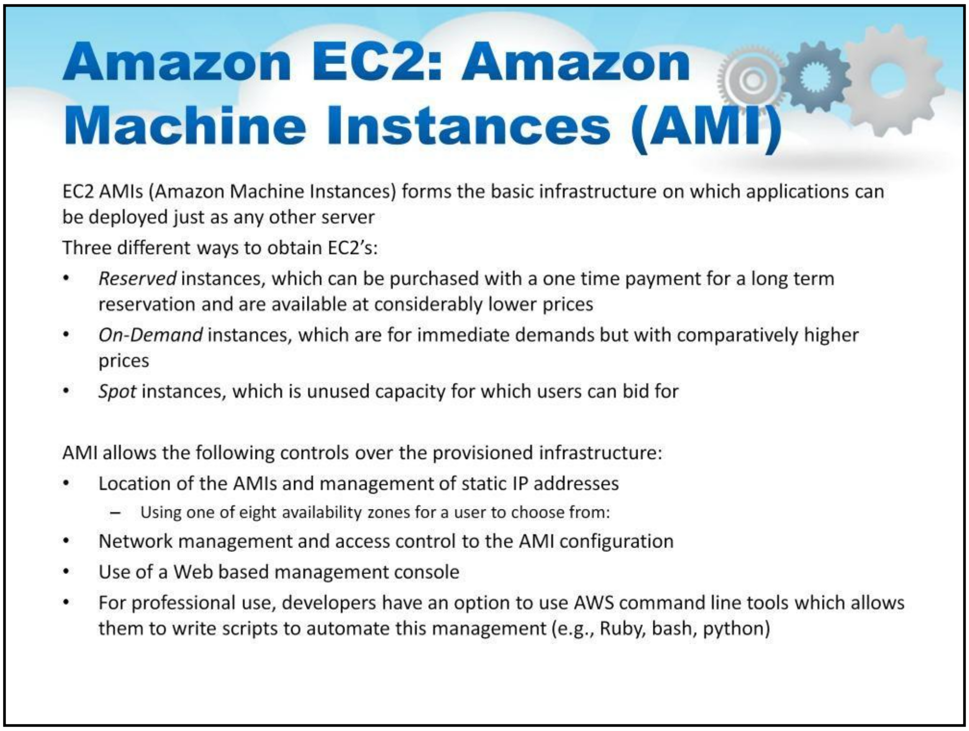
To handle sudden spikes in demand without the need to pre-over-provision resources

Elastic Load Balancing can balance load between multiple VMs located within a single availability zone or multiple zones

Can be used to create fault tolerance system that monitors health of each VM and distributes load between active instances

VM Import/Export is a service that provides functionality to upload custom VM images and store a live instance as an image

Saves efforts to create custom VM instance (including infrastructure components, applications, security and compliance features etc).



Amazon EC2: Amazon Machine Instances (AMI)

EC2 AMIs (Amazon Machine Instances) forms the basic infrastructure on which applications can be deployed just as any other server

Three different ways to obtain EC2's:

- *Reserved* instances, which can be purchased with a one time payment for a long term reservation and are available at considerably lower prices
- *On-Demand* instances, which are for immediate demands but with comparatively higher prices
- *Spot* instances, which is unused capacity for which users can bid for

AMI allows the following controls over the provisioned infrastructure:

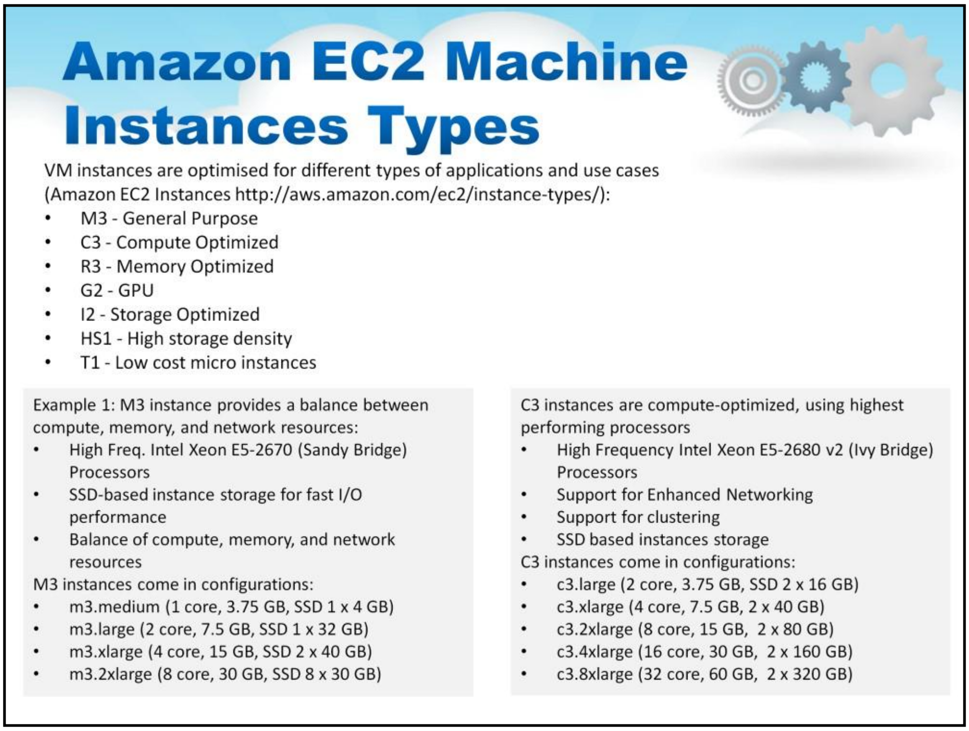
- Location of the AMIs and management of static IP addresses
  - Using one of eight availability zones for a user to choose from:
- Network management and access control to the AMI configuration
- Use of a Web based management console
- For professional use, developers have an option to use AWS command line tools which allows them to write scripts to automate this management (e.g., Ruby, bash, python)

Let us now look more closely at EC2.

EC2 AMIs (Amazon Machine Instances) forms the basic infrastructure on which applications can be deployed just as any other server

The slide lists three different ways to obtain EC2's. Just to be clear, once you get a certain size EC2, they are all the "same", there is no inherent difference between a "small" obtain through Spot or Reserved.

AMIs allow to use several controls over the provisioned infrastructure.
The slide lists these controls. There are supposed to be a minimum of them to "maximize simplicity" in use.



Amazon EC2 Machine Instances Types

VM instances are optimised for different types of applications and use cases (Amazon EC2 Instances http://aws.amazon.com/ec2/instance-types/):

- M3 - General Purpose
- C3 - Compute Optimized
- R3 - Memory Optimized
- G2 - GPU
- I2 - Storage Optimized
- HS1 - High storage density
- T1 - Low cost micro instances

Example 1: M3 instance provides a balance between compute, memory, and network resources:

- High Freq. Intel Xeon E5-2670 (Sandy Bridge) Processors
- SSD-based instance storage for fast I/O performance
- Balance of compute, memory, and network resources

M3 instances come in configurations:

- m3.medium (1 core, 3.75 GB, SSD 1 x 4 GB)
- m3.large (2 core, 7.5 GB, SSD 1 x 32 GB)
- m3.xlarge (4 core, 15 GB, SSD 2 x 40 GB)
- m3.2xlarge (8 core, 30 GB, SSD 8 x 30 GB)

C3 instances are compute-optimized, using highest performing processors

- High Frequency Intel Xeon E5-2680 v2 (Ivy Bridge) Processors
- Support for Enhanced Networking
- Support for clustering
- SSD based instances storage

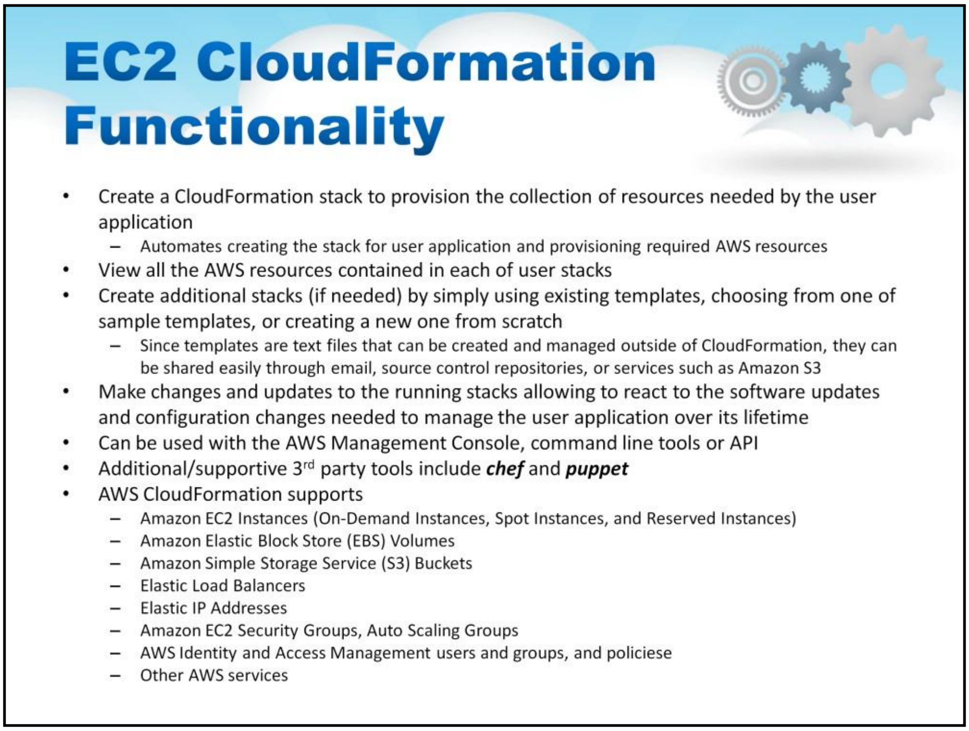
C3 instances come in configurations:

- c3.large (2 core, 3.75 GB, SSD 2 x 16 GB)
- c3.xlarge (4 core, 7.5 GB, 2 x 40 GB)
- c3.2xlarge (8 core, 15 GB, 2 x 80 GB)
- c3.4xlarge (16 core, 30 GB, 2 x 160 GB)
- c3.8xlarge (32 core, 60 GB, 2 x 320 GB)

Amazon has many machine types.
These are the “types” and sizes of EC2 attributes one can “order”.

Note: the naming rules allow to understand configuration behind the names.

Note: there is no “standard Elastic Compute Unit” or other real measurement of capacity. While the different machine types are quite explicit in what they consist of, any notion of derived performance or standardized benchmark is not to be seen.



EC2 CloudFormation Functionality

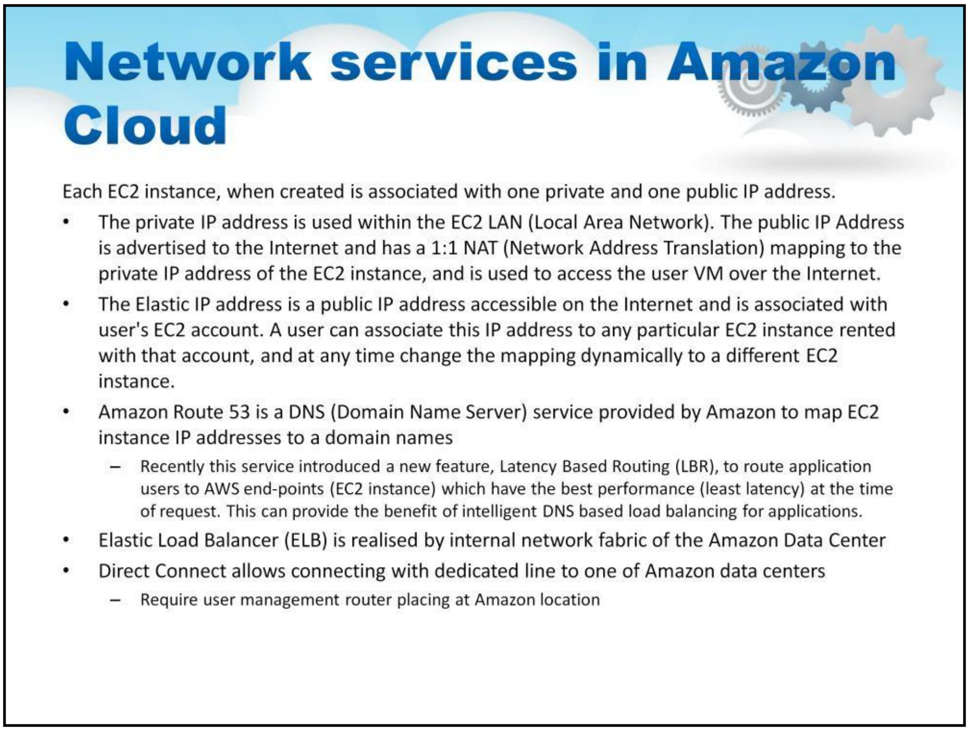
- Create a CloudFormation stack to provision the collection of resources needed by the user application
  - Automates creating the stack for user application and provisioning required AWS resources
- View all the AWS resources contained in each of user stacks
- Create additional stacks (if needed) by simply using existing templates, choosing from one of sample templates, or creating a new one from scratch
  - Since templates are text files that can be created and managed outside of CloudFormation, they can be shared easily through email, source control repositories, or services such as Amazon S3
- Make changes and updates to the running stacks allowing to react to the software updates and configuration changes needed to manage the user application over its lifetime
- Can be used with the AWS Management Console, command line tools or API
- Additional/supportive 3rd party tools include ***chef*** and ***puppet***
- AWS CloudFormation supports
  - Amazon EC2 Instances (On-Demand Instances, Spot Instances, and Reserved Instances)
  - Amazon Elastic Block Store (EBS) Volumes
  - Amazon Simple Storage Service (S3) Buckets
  - Elastic Load Balancers
  - Elastic IP Addresses
  - Amazon EC2 Security Groups, Auto Scaling Groups
  - AWS Identity and Access Management users and groups, and policiese
  - Other AWS services

CloudFormation is system (or tool really) which AWS provides as a option, to help with the automation of provisioning. It takes a specification on the resources needed for each element in the collection of elements that make up the application.

This slide details some of the features of Cloud Formation.

It is a very handy toolset for creating and maintaining AWS deployments. It is proprietary to Amazon.

While there are some efforts (OpenStack “heat” project) to re-create a compatible system as a portable, Open Source implementation, the developer should realize that this toolset is not available anywhere else but AWS. For this reason many developers use Chef or Puppet as much as possible and then interface that to CloudFormation.



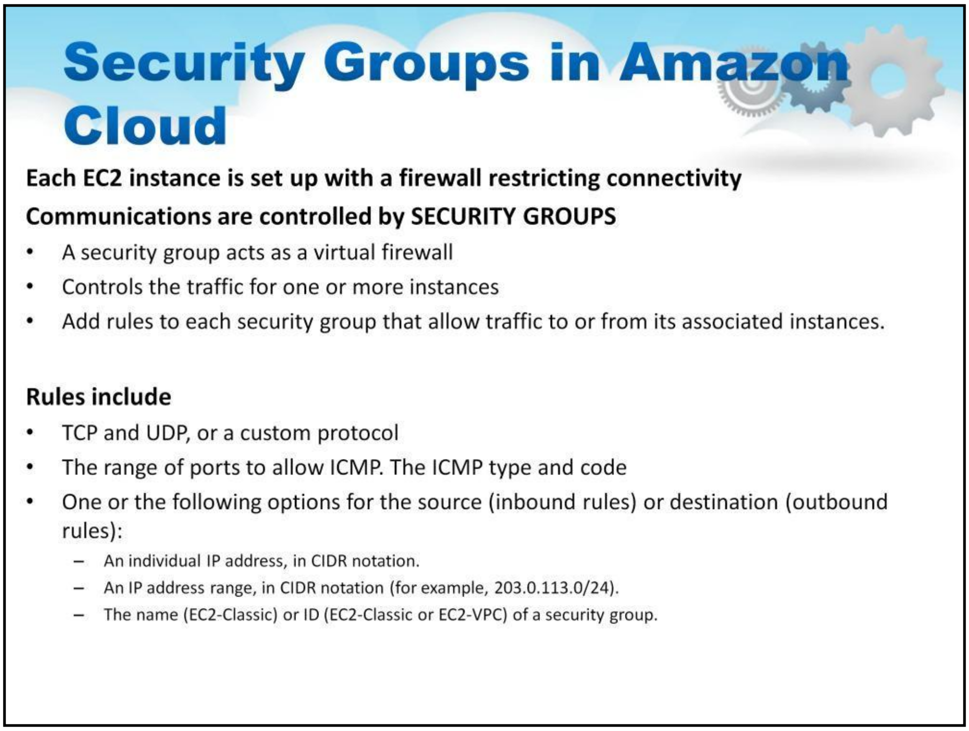
Network services in Amazon Cloud

Each EC2 instance, when created is associated with one private and one public IP address.

- The private IP address is used within the EC2 LAN (Local Area Network). The public IP Address is advertised to the Internet and has a 1:1 NAT (Network Address Translation) mapping to the private IP address of the EC2 instance, and is used to access the user VM over the Internet.
- The Elastic IP address is a public IP address accessible on the Internet and is associated with user's EC2 account. A user can associate this IP address to any particular EC2 instance rented with that account, and at any time change the mapping dynamically to a different EC2 instance.
- Amazon Route 53 is a DNS (Domain Name Server) service provided by Amazon to map EC2 instance IP addresses to a domain names
  - Recently this service introduced a new feature, Latency Based Routing (LBR), to route application users to AWS end-points (EC2 instance) which have the best performance (least latency) at the time of request. This can provide the benefit of intelligent DNS based load balancing for applications.
- Elastic Load Balancer (ELB) is realised by internal network fabric of the Amazon Data Center
- Direct Connect allows connecting with dedicated line to one of Amazon data centers
  - Require user management router placing at Amazon location

Every IaaS system has a specific “personality” about how it works and one of the main aspects of this personality” is how networking is set up for a VM.
AWS uses a very specific networking set up which lacks many networking features (like Multicast or VLAN).

This slide details the networking context of AWS.



Security Groups in Amazon Cloud

Each EC2 instance is set up with a firewall restricting connectivity

Communications are controlled by SECURITY GROUPS

- A security group acts as a virtual firewall
- Controls the traffic for one or more instances
- Add rules to each security group that allow traffic to or from its associated instances.

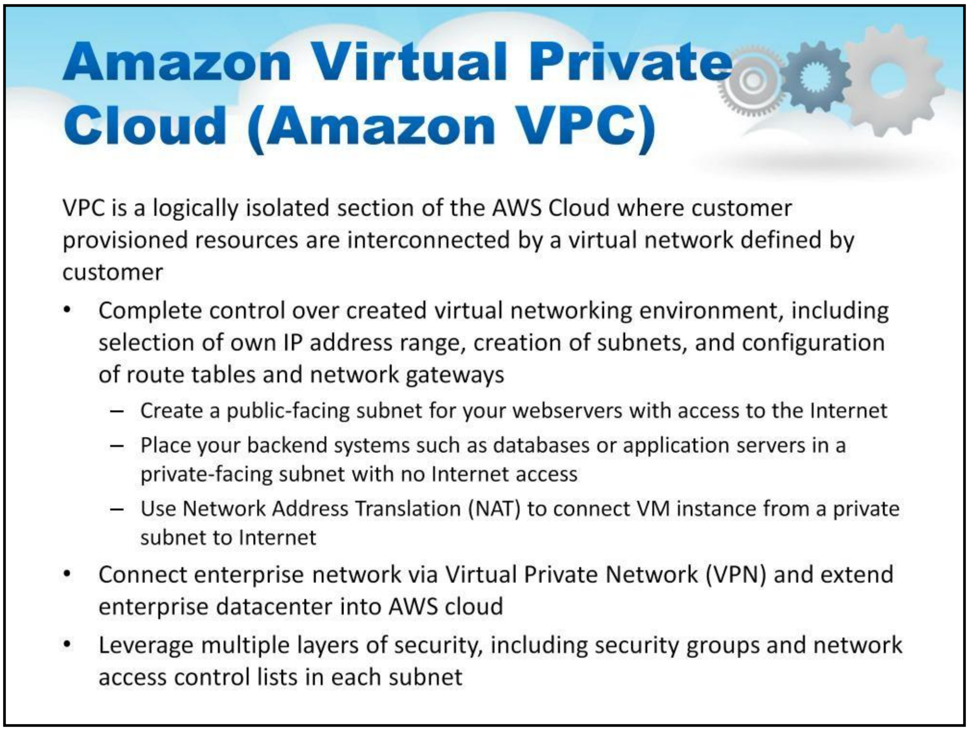
Rules include

- TCP and UDP, or a custom protocol
- The range of ports to allow ICMP. The ICMP type and code
- One or the following options for the source (inbound rules) or destination (outbound rules):
  - An individual IP address, in CIDR notation.
  - An IP address range, in CIDR notation (for example, 203.0.113.0/24).
  - The name (EC2-Classic) or ID (EC2-Classic or EC2-VPC) of a security group.

The rules of a security group control the inbound traffic that's allowed to reach the instances that are associated with the security group and the outbound traffic that's allowed to leave them.

By default, security groups allow all outbound traffic.
- You can add and remove rules at any time.
- Your changes are automatically applied to the instances associated with the security group after a short period.
- You can either edit an existing rule in a security group, or delete it and add a new rule.
- You can copy the rules from an existing security group to a new security group.
- You can't change the outbound rules for EC2-Classic.

Security group rules are always permissive; you can't create rules that deny access. For each rule, you specify the following:
- The protocol to allow (such as TCP, UDP, or ICMP): TCP and UDP, or a custom protocol.
- The range of ports to allow ICMP, ICMP type and code
- One or the following options for the source (inbound rules) or destination (outbound rules) as detailed on this slide.



Amazon Virtual Private Cloud (Amazon VPC)

VPC is a logically isolated section of the AWS Cloud where customer provisioned resources are interconnected by a virtual network defined by customer

- Complete control over created virtual networking environment, including selection of own IP address range, creation of subnets, and configuration of route tables and network gateways
  - Create a public-facing subnet for your webservers with access to the Internet
  - Place your backend systems such as databases or application servers in a private-facing subnet with no Internet access
  - Use Network Address Translation (NAT) to connect VM instance from a private subnet to Internet
- Connect enterprise network via Virtual Private Network (VPN) and extend enterprise datacenter into AWS cloud
- Leverage multiple layers of security, including security groups and network access control lists in each subnet

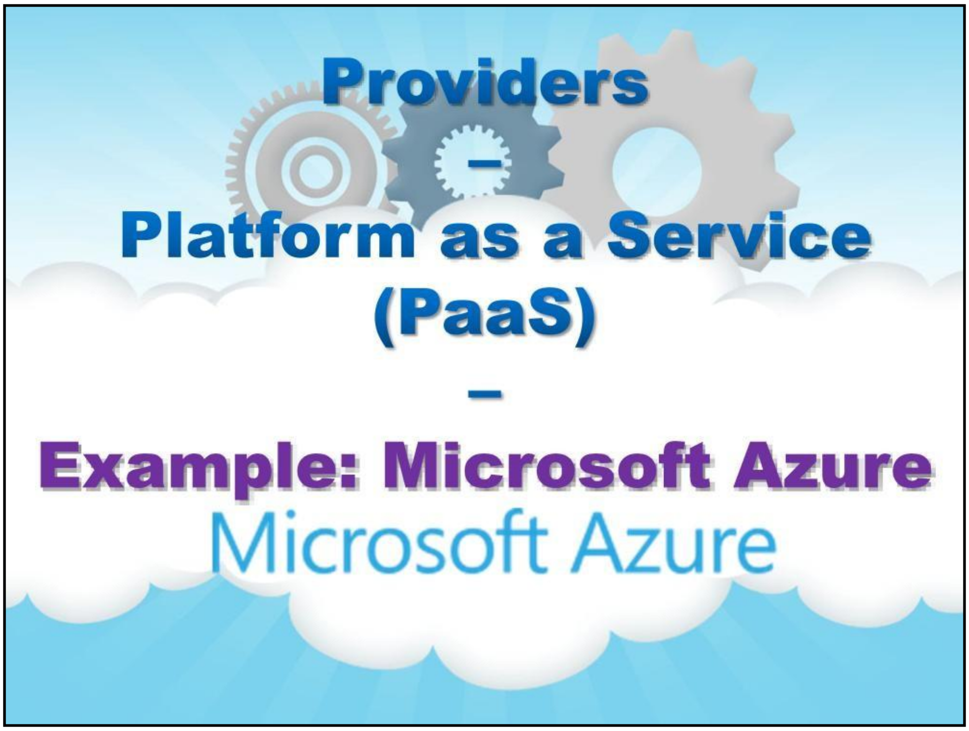
Amazon Virtual Private Cloud (Amazon VPC) is a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network defined by you.

VPC allows **complete control over created virtual networking environment**, including selection of own IP address range, creation of subnets, and configuration of route tables and network gateways.

> For example. You can create a public-facing subnet for your webservers with access to the Internet. Then you can create private-facing subnet where you can place your backend systems such as databases or application servers with no Internet access.

VPC allows **connect enterprise network via Virtual Private Network (VPN)** and extend enterprise datacenter into AWS cloud

VPC allows **to leverage multiple layers of security** including security groups and network access control lists in each subnet.



Providers

–

Platform as a Service (PaaS)

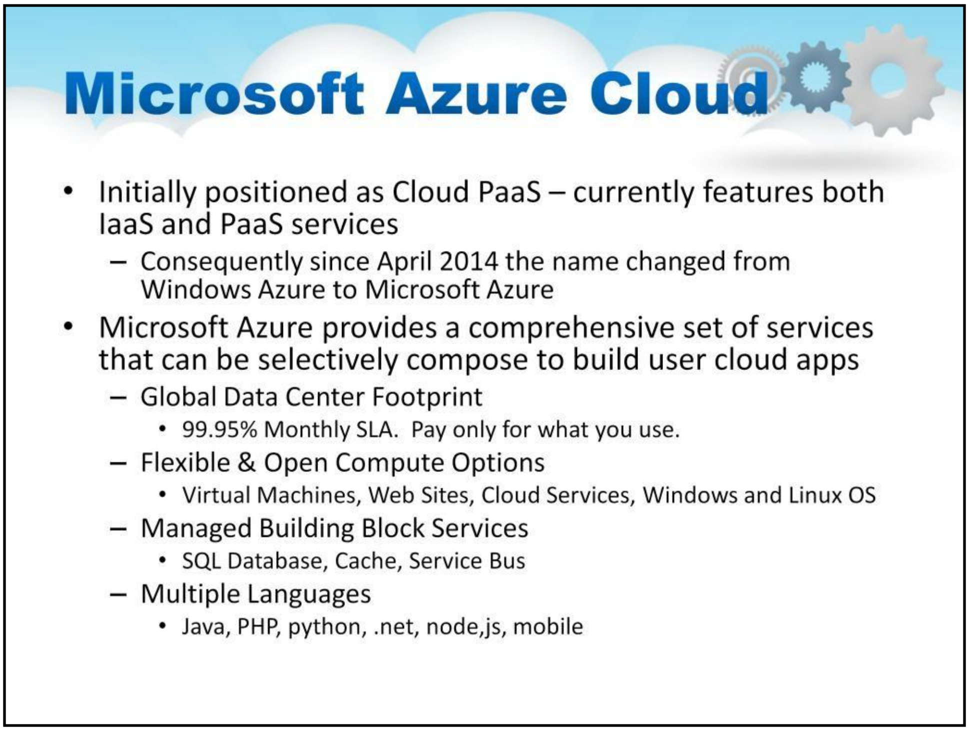
–

Example: Microsoft Azure

Microsoft Azure

Example: Microsoft Azure

The second example is dedicated to Microsoft Azure that represents an example of Platform as a Service (PaaS).



Microsoft Azure Cloud

- Initially positioned as Cloud PaaS – currently features both IaaS and PaaS services
  - Consequently since April 2014 the name changed from Windows Azure to Microsoft Azure
- Microsoft Azure provides a comprehensive set of services that can be selectively compose to build user cloud apps
  - Global Data Center Footprint
    - 99.95% Monthly SLA. Pay only for what you use.
  - Flexible & Open Compute Options
    - Virtual Machines, Web Sites, Cloud Services, Windows and Linux OS
  - Managed Building Block Services
    - SQL Database, Cache, Service Bus
  - Multiple Languages
    - Java, PHP, python, .net, node,js, mobile

The Microsoft Azure Cloud was first released as Windows Azure and was strictly a PaaS platform aimed at providing Cloud services to Windows developers. After adding IaaS abilities including the ability to run Linux VM's, Azure has become an important player in Cloud.

Microsoft Azure provides a comprehensive set of services that can be selectively compose to build user cloud applications, for example:

Global Data Center Footprint

99.95% Monthly SLA. Pay only for what you use.

Flexible & Open Compute Options

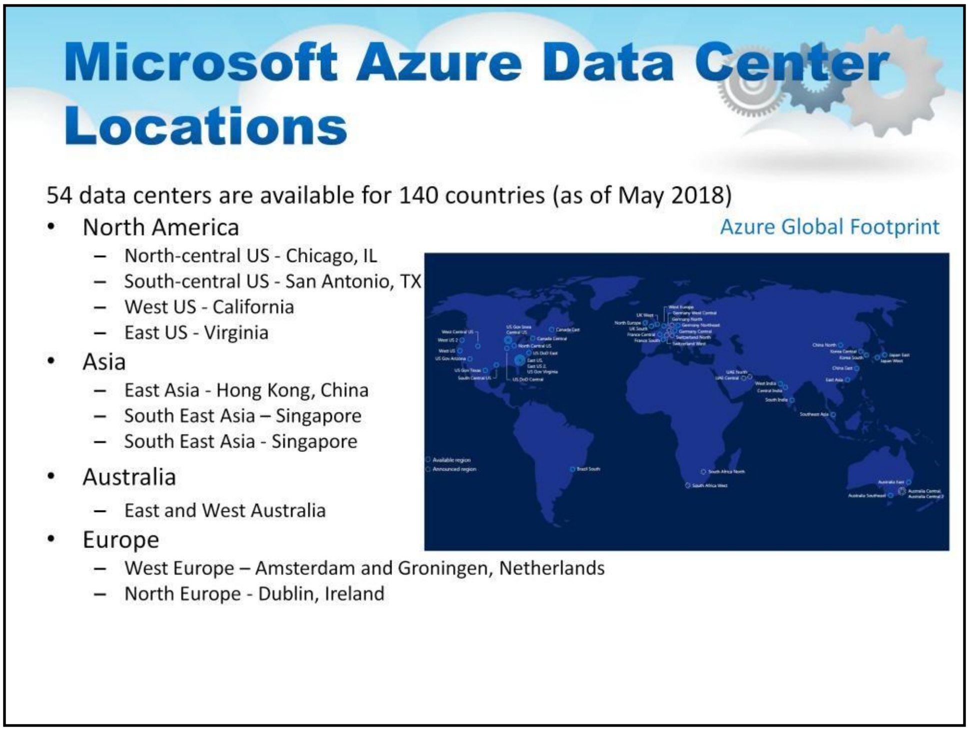
Virtual Machines, Web Sites, Cloud Services, Windows and Linux OS

Managed Building Block Services

SQL Database, Cache, Service Bus

Multiple Languages

Java, PHP, python, .net, node,js, mobile



Microsoft Azure Data Center Locations

54 data centers are available for 140 countries (as of May 2018)

- North America
  - North-central US - Chicago, IL
  - South-central US - San Antonio, TX
  - West US - California
  - East US - Virginia
- Asia
  - East Asia - Hong Kong, China
  - South East Asia – Singapore
  - South East Asia - Singapore
- Australia
  - East and West Australia
- Europe
  - West Europe – Amsterdam and Groningen, Netherlands
  - North Europe - Dublin, Ireland

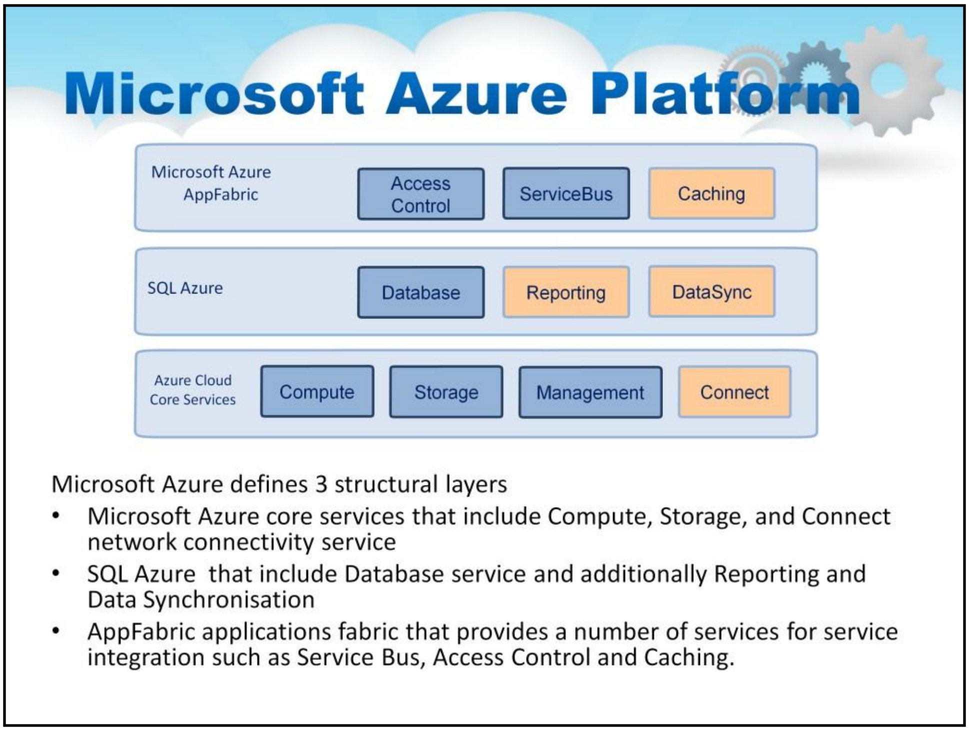
Azure Global Footprint

Microsoft has one of the three largest footprints of computing counting all the cloud vendors (Microsoft, Google, and Amazon).

54 massive datacenters are installed now worldwide and available in 140 countries.

Microsoft delivers many services from their Cloud, ranging from Azure, to Xbox Live, to Outlook.com, to Windows Update, and every SaaS application inbetween.

So they really have invested in a global infrastructure.



Microsoft Azure Platform

Microsoft Azure AppFabric: Access Control | ServiceBus | Caching

SQL Azure: Database | Reporting | DataSync

Azure Cloud Core Services: Compute | Storage | Management | Connect

Microsoft Azure defines 3 structural layers

- Microsoft Azure core services that include Compute, Storage, and Connect network connectivity service
- SQL Azure that include Database service and additionally Reporting and Data Synchronisation
- AppFabric applications fabric that provides a number of services for service integration such as Service Bus, Access Control and Caching.

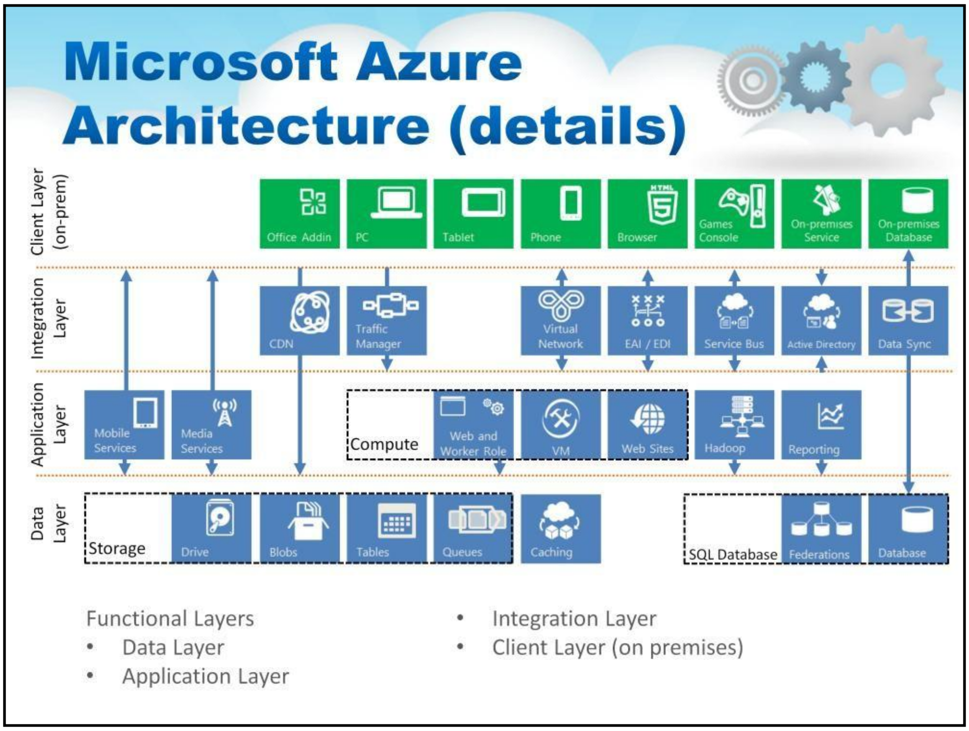
The Azure platform is pretty much like other cloud platforms as to it’s structure with the exception that the data and application services are very closely integrated with the underlying core cloud services.

The slide illustrates the Microsoft Azure 3 structural layers :

Microsoft Azure core services includes Compute, Storage, and Connect network connectivity service.

SQL Azure includes Database service and additionally Reporting and Data Synchronisation.

Microsoft AppFabric applications provides a number of services for service integration such as Service Bus, Access Control and Caching.



Microsoft Azure Architecture (details)

Client Layer (on-prem)

Office Addin · PC · Tablet · Phone · Browser · Games Console · On-premises Service · On-premises Database

Integration Layer

CDN · Traffic Manager · Virtual Network · EAI / EDI · Service Bus · Active Directory · Data Sync

Application Layer

Mobile Services · Media Services · Compute · Web and Worker Role · VM · Web Sites · Hadoop · Reporting

Data Layer

Storage · Drive · Blobs · Tables · Queues · Caching · SQL Database · Federations · Database

Functional Layers

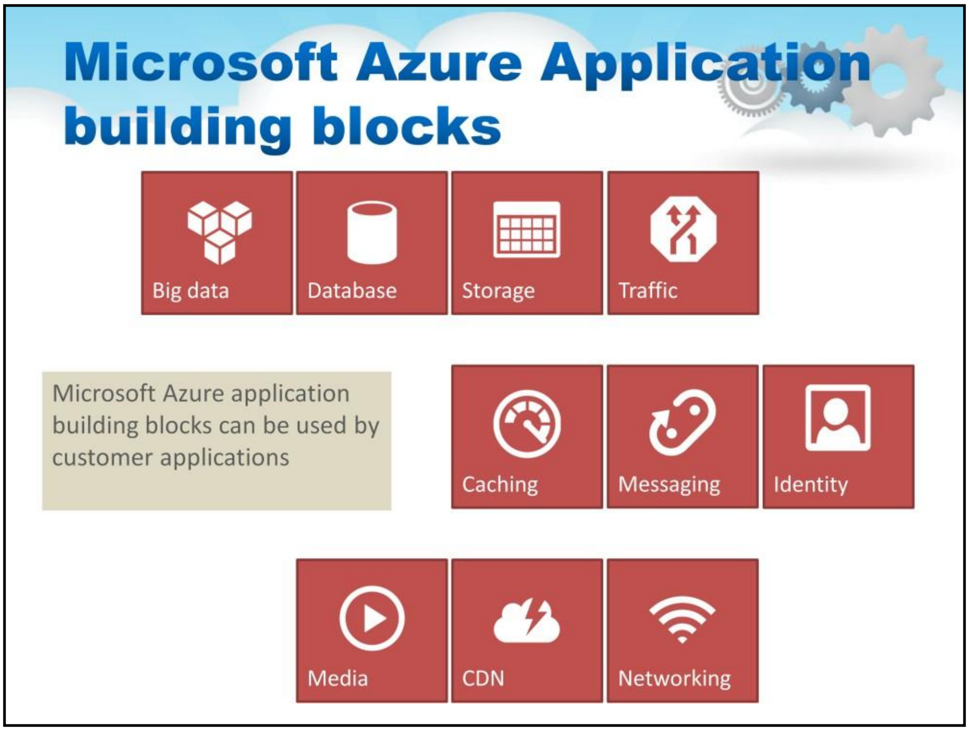
- Data Layer
- Application Layer
- Integration Layer
- Client Layer (on premises)

This slide illustrates the “Marketecture”, that is Market-Architecture, of Microsoft Azure.

It can be seen that it is organized into several Functional Layers:

❑ Data Layer
❑ Application Layer
❑ Integration Layer
❑ Client Layer (on premises)

As you can see from this picture, each layer contains a considerable amount of functional capabilities.



Microsoft Azure Application building blocks

Microsoft Azure application building blocks can be used by customer applications

One of the other advantages of Microsoft Azure is possibility to use a number of Application Building Blocks.

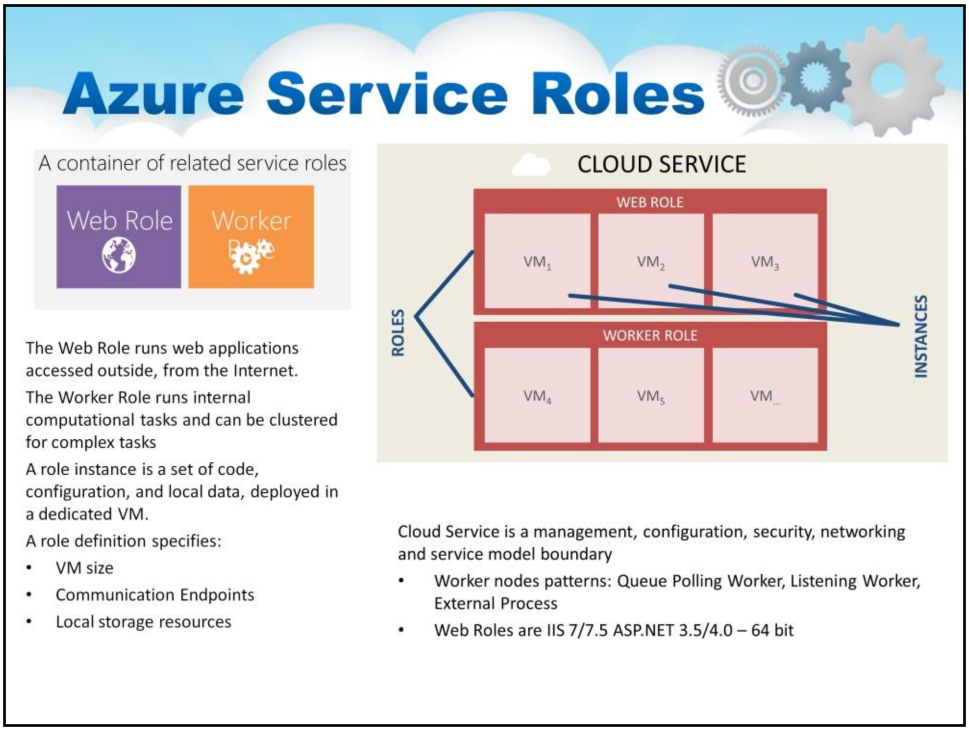
Application Building Blocks are managed services that Microsoft runs to provide a lot of value so the developer can avoid standing up the infrastructure for common capabilities.

The developer can stand up VMs and put anything wanted there.

In many cases the developer will find that Microsoft have built in services that are directly delivered or that are delivered by their partners.

A developer can use any of these services with a VM, with a Web Site, or with a Cloud Service – so there is flexibility in how one will consume them.

The slide illustrates variety of these Application building blocks.



Azure Service Roles

The Web Role runs web applications accessed outside, from the Internet.

The Worker Role runs internal computational tasks and can be clustered for complex tasks

A role instance is a set of code, configuration, and local data, deployed in a dedicated VM.

A role definition specifies:

- VM size
- Communication Endpoints
- Local storage resources

Cloud Service is a management, configuration, security, networking and service model boundary

- Worker nodes patterns: Queue Polling Worker, Listening Worker, External Process
- Web Roles are IIS 7/7.5 ASP.NET 3.5/4.0 – 64 bit

A key concept in the Azure PaaS environment is the Service Role.

Service Roles mostly determine what kind of traffic is directed towards that service,

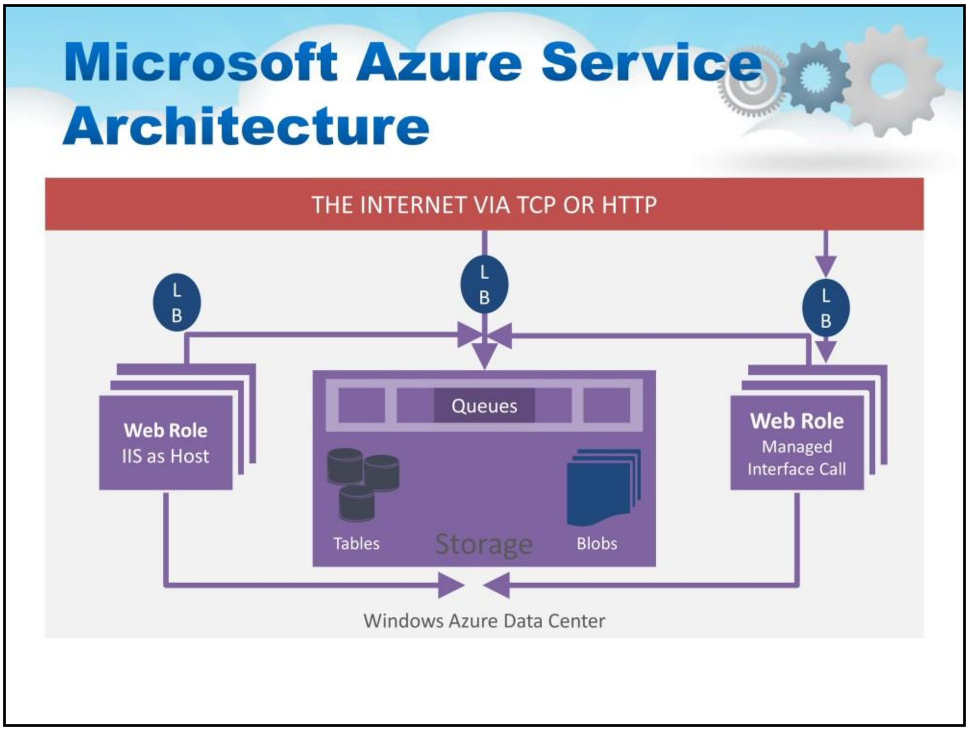
For example, **Web Role** runs web applications accessed outside, from the Internet. All outside connectivity goes through Web Service Role entities. Load Balancing (inherent in Azure) spreads incoming traffic across Web Service Role entities.

But the other **Worker Role** runs internal computational tasks and can be clustered for complex tasks. Connectivity of Worker Role entities is limited to internal traffic only (to/from other Worker Role entities, or to/from Web Role entities)

A **role instance** is a set of code, configuration, and local data, deployed in a dedicated VM.
A role definition specifies:

- VM size
- Communication Endpoints
- Local storage resources

Cloud Service is a management, configuration, security, networking and service model boundary – consisting of a group of Service Roles.



Microsoft Azure Service Architecture

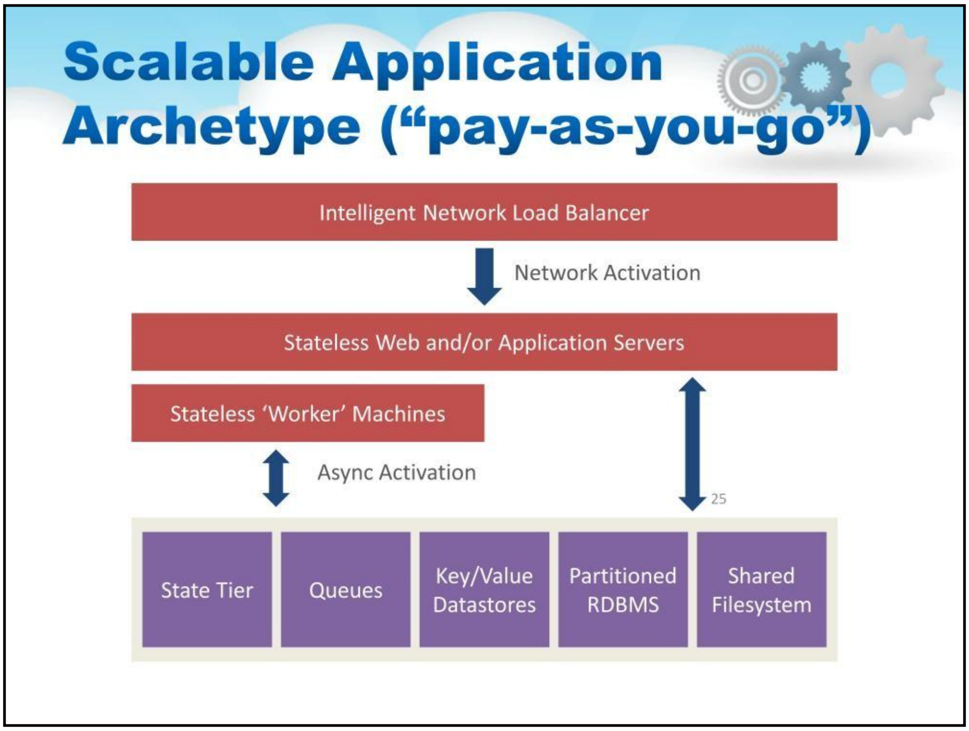
THE INTERNET VIA TCP OR HTTP

This slide includes an illustration of the high-scale view on the Microsoft Azure Service Architecture:

Key points here are

- all external connections come through a load balancer (LB)
- inter-role communication (notice there is no load balancer) and TCP ports directly to Worker Roles (or Web Roles)
- the storage is used to communicate asynchronously and reliably via queues for a lot of options
- inter-role communication fills in when you need direct synchronous communication.

The load balancers are a key to Windows Azure.



Scalable Application Archetype (“pay-as-you-go”)

Intelligent Network Load Balancer

Network Activation

Stateless Web and/or Application Servers

Stateless ‘Worker’ Machines

Async Activation

25

State Tier

Queues

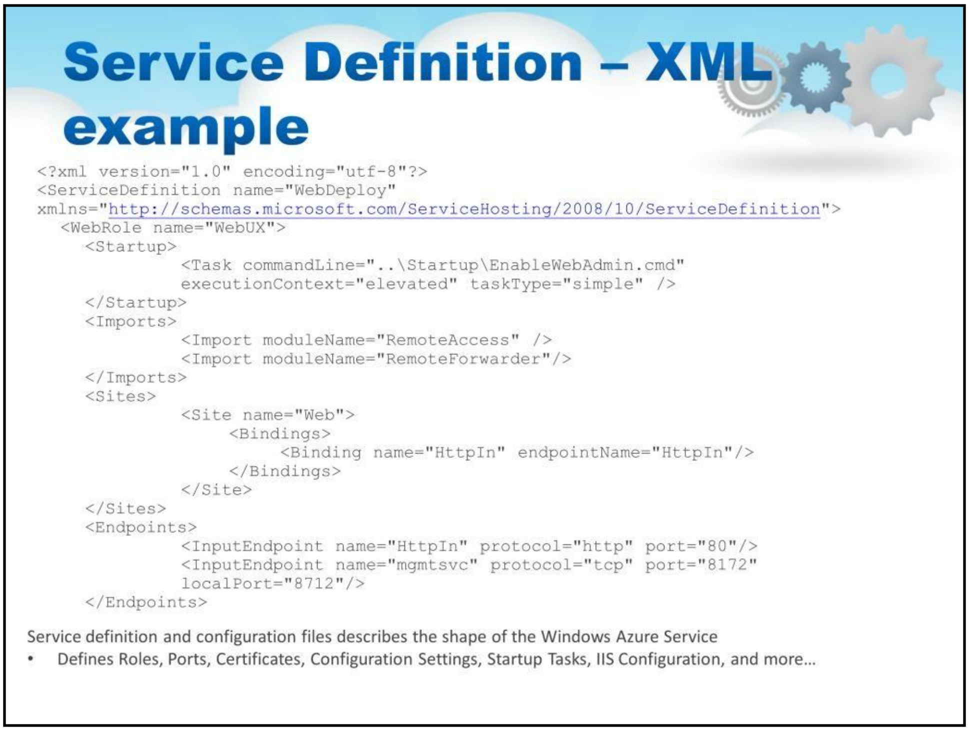
Key/Value Datastores

Partitioned RDBMS

Shared Filesystem

This slide contains an illustration of a scale-out application

- High scale applications often follow this sort of an pattern
- Inbound connectivity comes through a load balancer
  - Requests are routed in a round robin way
  - Load balancer is typically aware of the state of the web servers
- There are one or more tiers or groups of stateless web or app servers
  - By stateless we mean that they do not hold state between client requests
  - Stateless means that simple load balancing works – no need for sticky sessions
  - Stateless means that the failure of a web server does not cause major issues for application- it is simply removed from the load balancer
- A stateful or storage tier
  - This will generally involve some sort of scale out approach for large apps
  - Often using partitioned databases
  - Often some sort of queuing mechanism
- Applications will often perform processing in the background.
  - Improves response time for users
  - Allows load peaks to be buffered in queues



Service Definition – XML example

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="WebDeploy"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="WebUX">
    <Startup>
            <Task commandLine="..\Startup\EnableWebAdmin.cmd"
            executionContext="elevated" taskType="simple" />
    </Startup>
    <Imports>
            <Import moduleName="RemoteAccess" />
            <Import moduleName="RemoteForwarder"/>
    </Imports>
    <Sites>
            <Site name="Web">
                <Bindings>
                     <Binding name="HttpIn" endpointName="HttpIn"/>
                </Bindings>
            </Site>
    </Sites>
    <Endpoints>
            <InputEndpoint name="HttpIn" protocol="http" port="80"/>
            <InputEndpoint name="mgmtsvc" protocol="tcp" port="8172"
            localPort="8712"/>
    </Endpoints>
```

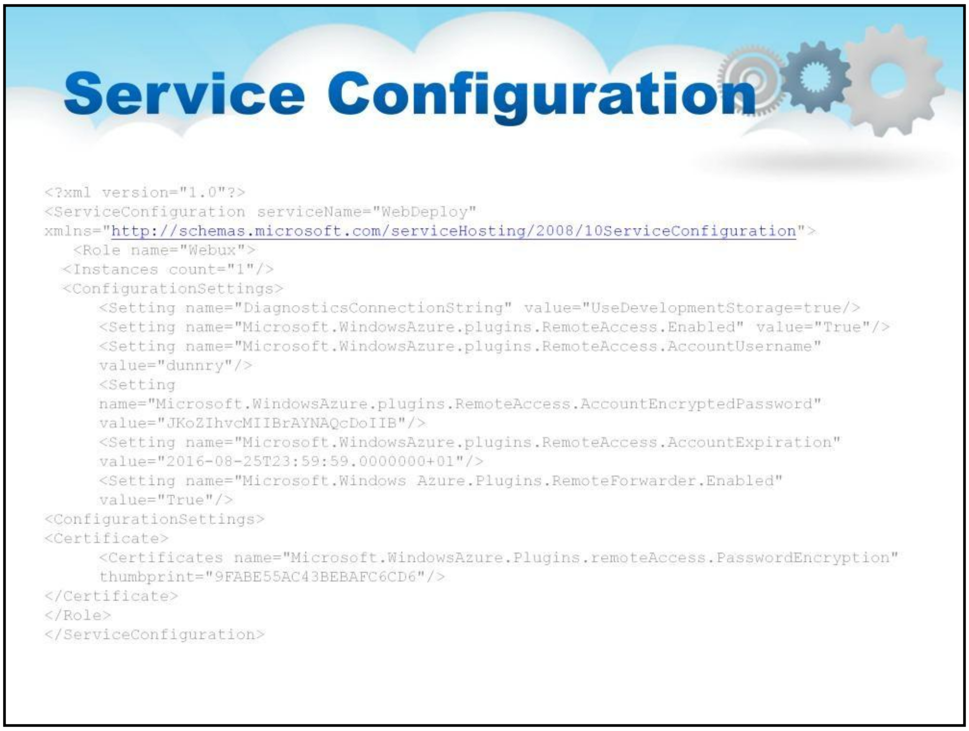
Service definition and configuration files describes the shape of the Windows Azure Service

- Defines Roles, Ports, Certificates, Configuration Settings, Startup Tasks, IIS Configuration, and more…

Microsoft Azure Service components and configuration are defined in a form of XML document.

Services Definition
• describes the shape of the Microsoft Azure Service
• defines Roles, Ports, Certificates, Configuration Settings, Startup Tasks, IIS Configuration, and more…

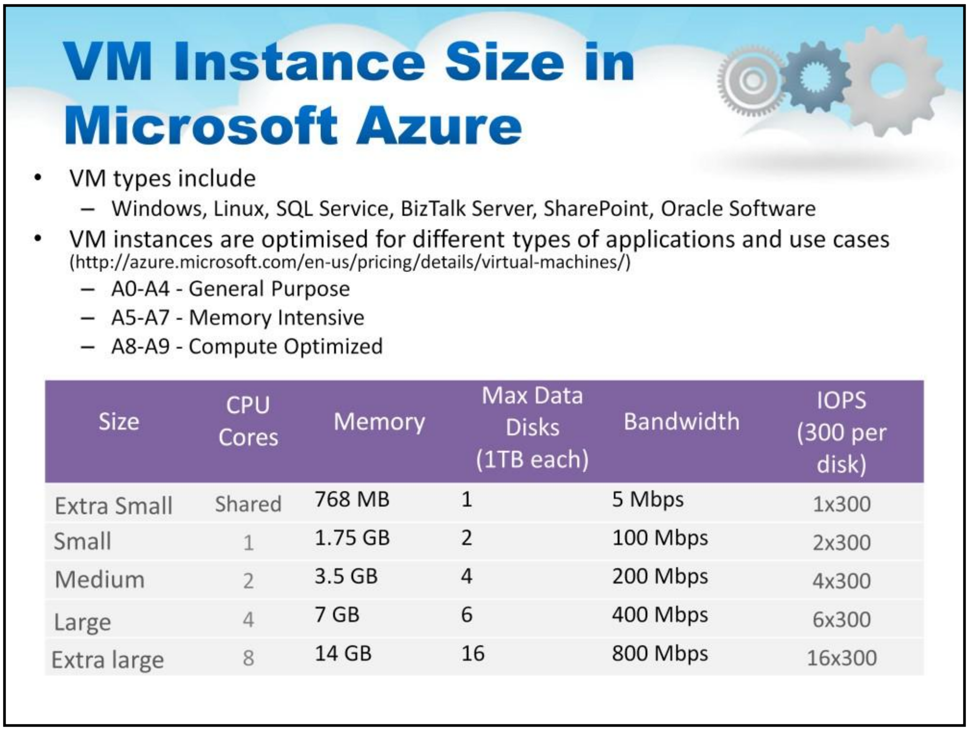
XML example of a Service Definition is shown in this slide.



Service Configuration

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="WebDeploy"
xmlns="http://schemas.microsoft.com/serviceHosting/2008/10ServiceConfiguration">
   <Role name="Webux">
  <Instances count="1"/>
  <ConfigurationSettings>
      <Setting name="DiagnosticsConnectionString" value="UseDevelopmentStorage=true/>
      <Setting name="Microsoft.WindowsAzure.plugins.RemoteAccess.Enabled" value="True"/>
      <Setting name="Microsoft.WindowsAzure.plugins.RemoteAccess.AccountUsername"
      value="dunnry"/>
      <Setting
      name="Microsoft.WindowsAzure.plugins.RemoteAccess.AccountEncryptedPassword"
      value="JKoZIhvcMIIBrAYNAQcDoIIB"/>
      <Setting name="Microsoft.WindowsAzure.plugins.RemoteAccess.AccountExpiration"
      value="2016-08-25T23:59:59.0000000+01"/>
      <Setting name="Microsoft.Windows Azure.Plugins.RemoteForwarder.Enabled"
      value="True"/>
<ConfigurationSettings>
<Certificate>
      <Certificates name="Microsoft.WindowsAzure.Plugins.remoteAccess.PasswordEncryption"
      thumbprint="9FABE55AC43BEBAFC6CD6"/>
</Certificate>
</Role>
</ServiceConfiguration>
```

In this example the service configuration file defines account name and access related information: username, encrypted password, and related Certificate fingerprint.



VM Instance Size in Microsoft Azure

- VM types include
  - Windows, Linux, SQL Service, BizTalk Server, SharePoint, Oracle Software
- VM instances are optimised for different types of applications and use cases (http://azure.microsoft.com/en-us/pricing/details/virtual-machines/)
  - A0-A4 - General Purpose
  - A5-A7 - Memory Intensive
  - A8-A9 - Compute Optimized

Size	CPU Cores	Memory	Max Data Disks (1TB each)	Bandwidth	IOPS (300 per disk)
Extra Small	Shared	768 MB	1	5 Mbps	1x300
Small	1	1.75 GB	2	100 Mbps	2x300
Medium	2	3.5 GB	4	200 Mbps	4x300
Large	4	7 GB	6	400 Mbps	6x300
Extra large	8	14 GB	16	800 Mbps	16x300

This slide contains a table aimed at understanding how and why to change the VM Size for a Windows Azure role.

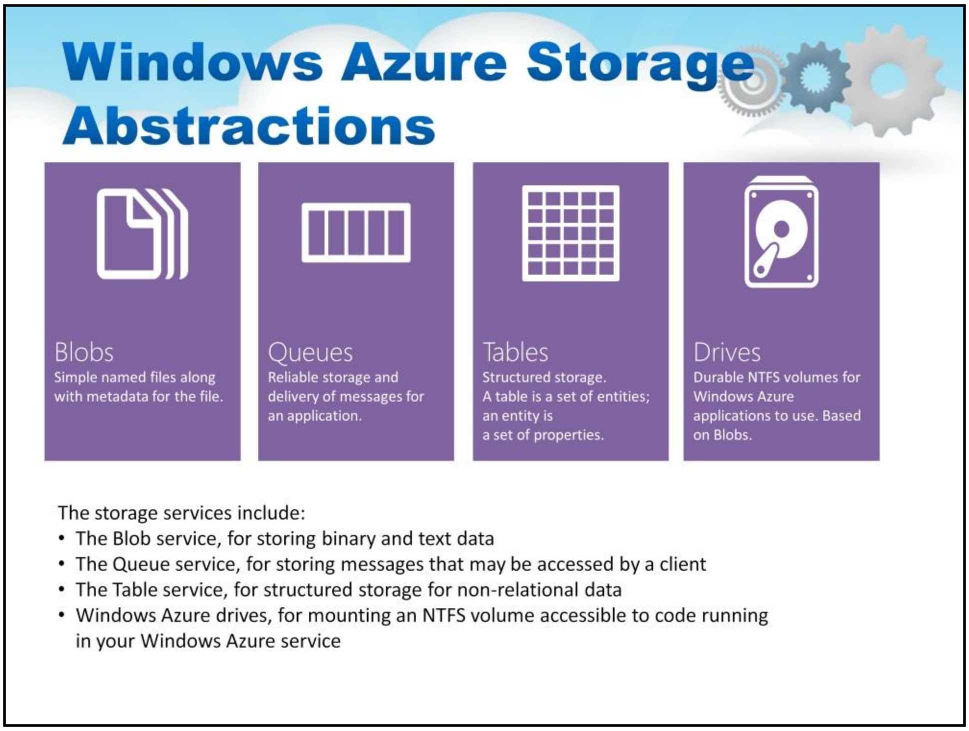
When you create your service model, you can specify the size of the virtual machine (VM) to which to deploy instances of your role, depending on its resource requirements.

The size of the VM determines

- the number of CPU cores
- the memory capacity
- the local file system size allocated to a running instance

Each physical machine in Windows Azure contains 8 processor cores. You need to specify an XL instance to reserve an entire machine

- Network is shared but burstable
- Can burst beyond your 1/8th allocation when using a small VM
- May be limited to just your allocation
- For guaranteed high network throughput use an XLVM



Windows Azure Storage Abstractions

Blobs
Simple named files along with metadata for the file.

Queues
Reliable storage and delivery of messages for an application.

Tables
Structured storage.
A table is a set of entities; an entity is a set of properties.

Drives
Durable NTFS volumes for Windows Azure applications to use. Based on Blobs.

The storage services include:
- The Blob service, for storing binary and text data
- The Queue service, for storing messages that may be accessed by a client
- The Table service, for structured storage for non-relational data
- Windows Azure drives, for mounting an NTFS volume accessible to code running in your Windows Azure service

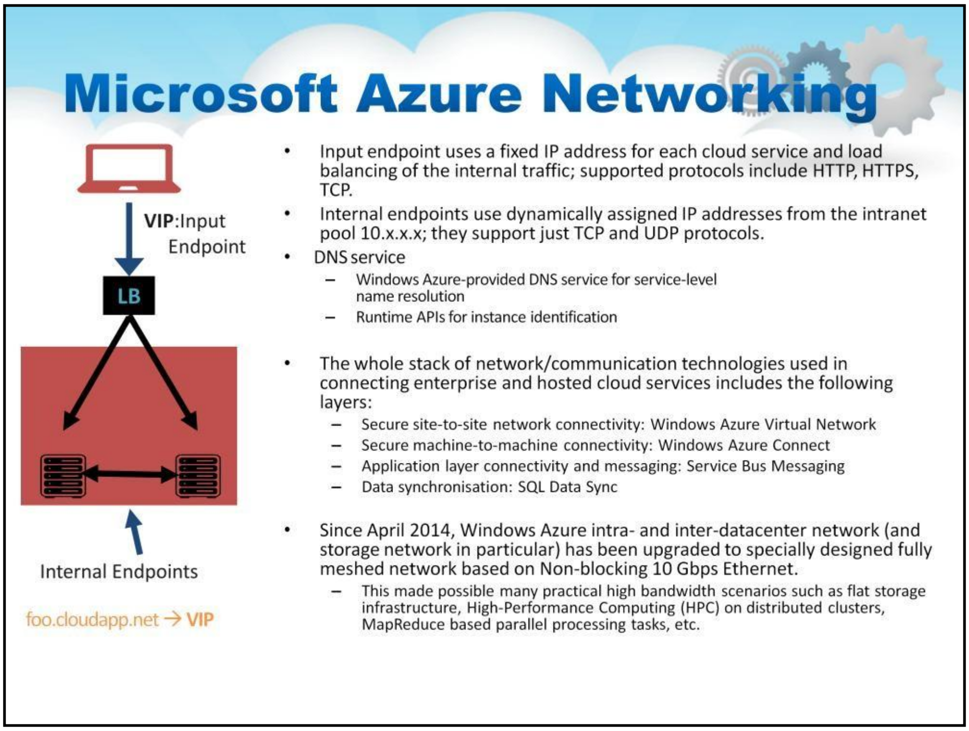
The illustration on this slide addresses Windows Azure Storage Abstractions and allows to understand each of the storage types at a high level.

The Windows Azure storage services provide storage for binary and text data, messages, and structured data in Windows Azure.

The storage services include:

- The Blob service, for storing binary and text data
- The Queue service, for storing messages that may be accessed by a client
- The Table service, for structured storage for non-relational data
- Windows Azure drives, for mounting an NTFS volume accessible to code running in your Windows Azure service

For developers, access to the Blob, Queue, and Table services is available via the Windows Azure Managed Library and the Windows Azure storage services REST API.



Microsoft Azure Networking

VIP:Input Endpoint

LB

Internal Endpoints

foo.cloudapp.net → VIP

- Input endpoint uses a fixed IP address for each cloud service and load balancing of the internal traffic; supported protocols include HTTP, HTTPS, TCP.
- Internal endpoints use dynamically assigned IP addresses from the intranet pool 10.x.x.x; they support just TCP and UDP protocols.
- DNS service
  - Windows Azure-provided DNS service for service-level name resolution
  - Runtime APIs for instance identification
- The whole stack of network/communication technologies used in connecting enterprise and hosted cloud services includes the following layers:
  - Secure site-to-site network connectivity: Windows Azure Virtual Network
  - Secure machine-to-machine connectivity: Windows Azure Connect
  - Application layer connectivity and messaging: Service Bus Messaging
  - Data synchronisation: SQL Data Sync
- Since April 2014, Windows Azure intra- and inter-datacenter network (and storage network in particular) has been upgraded to specially designed fully meshed network based on Non-blocking 10 Gbps Ethernet.
  - This made possible many practical high bandwidth scenarios such as flat storage infrastructure, High-Performance Computing (HPC) on distributed clusters, MapReduce based parallel processing tasks, etc.

Now we will look a the networking included with a Cloud Service

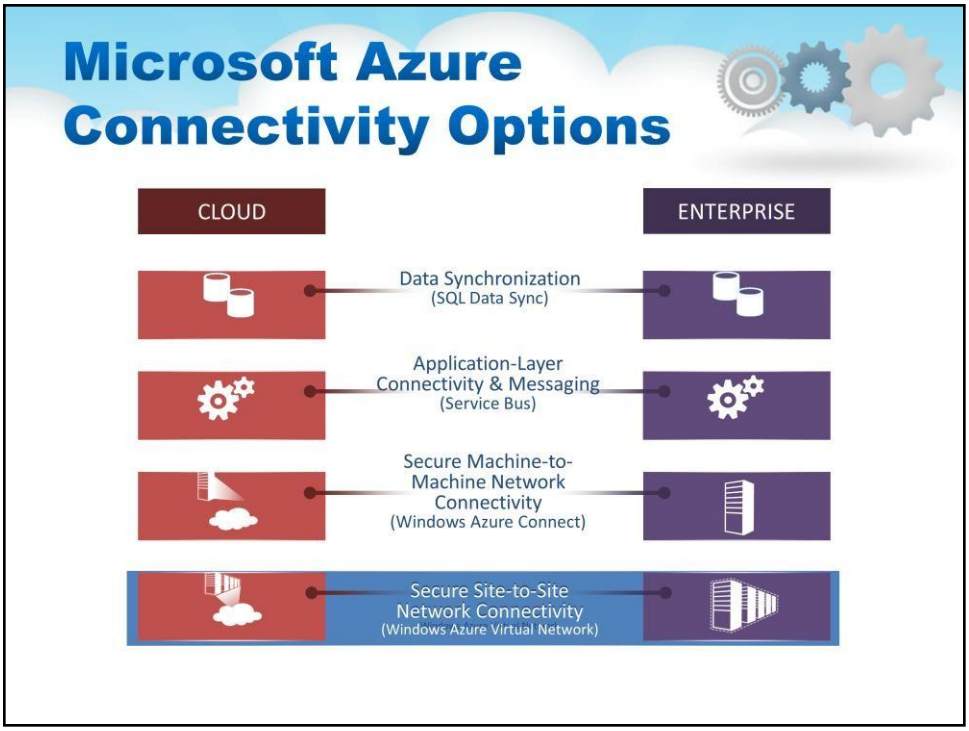
1. Cloud Service gets a Virtual IP assigned for a deployment slot
   - No ports opened up by default
   - Need to define endpoints to open up ports
2. Input endpoint is a port
   - It is load balanced
   - Mapped across all role instances
   - Port mapipng is supported
3. Internal endpoint enables inter-role-instance communication
   - Ports for inter-vm communication are closed by default
   - Need to define an internal endpoint for communication
   - Internal endpoints can be port ranges
4. DNS resolution
   - Only service-level name resolution is supported
   - Need to use runtime APIs for instance name resolution.

The whole stack of network/communication technologies used in connecting enterprise and hosted cloud services includes the following layers:

- Secure site-to-site network connectivity: Windows Azure Virtual Network
- Secure machine-to-machine connectivity: Windows Azure Connect
- Application layer connectivity and messaging: Service Bus Messaging
- Data synchronisation: SQL Data Sync

Since April 2014, Windows Azure intra- and inter-datacenter network (and storage network in particular) has been upgraded to specially designed fully meshed network based on Non-blocking 10 Gbps Ethernet.

This made possible many practical high bandwidth scenarios such as flat storage infrastructure, High-Performance Computing (HPC) on distributed clusters, MapReduce based parallel processing tasks, etc.



Microsoft Azure Connectivity Options

CLOUD

ENTERPRISE

Data Synchronization
(SQL Data Sync)

Application-Layer
Connectivity & Messaging
(Service Bus)

Secure Machine-to-
Machine Network
Connectivity
(Windows Azure Connect)

Secure Site-to-Site
Network Connectivity
(Windows Azure Virtual Network)

This slide illustrates how the Microsoft stack is structured to provide connectivity between on-premise and cloud.

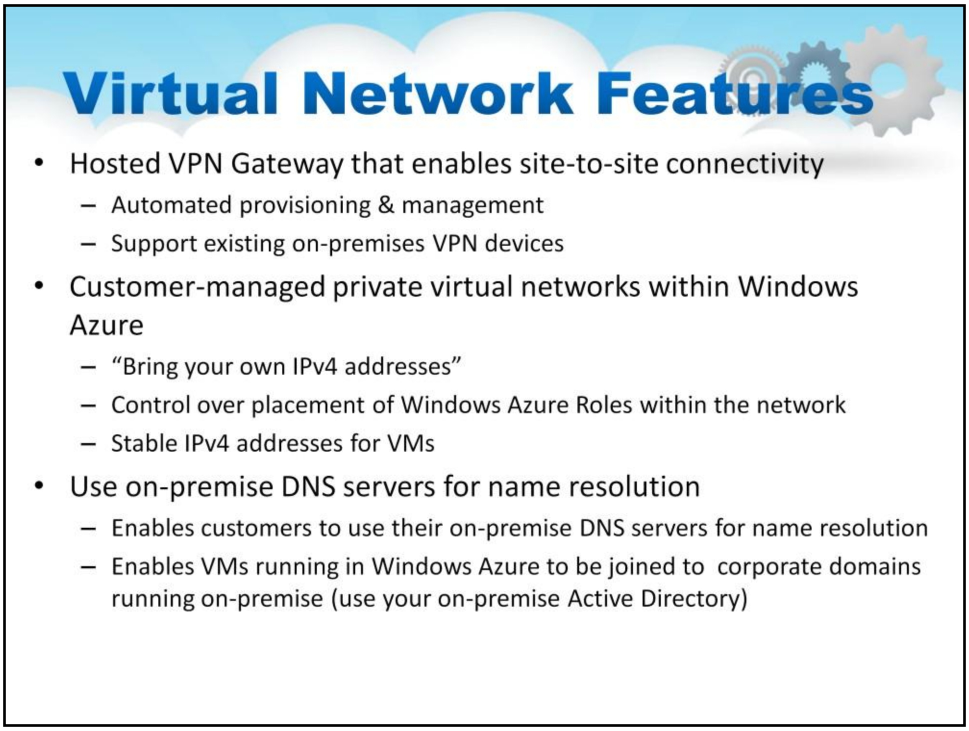
There are several facilities which can be used depending on the connectivity objective

At the lowest level, a VPN based connectivity can be achieved using Azure Virtual Network. This is analogous to AWS Virtual Private Cloud, it is based on IPSEC tunneling.

Azure Connect is more specific, connecting server (VM) to server as illustrated

There are many application level connectivity option but the most common implements a reliable message transport (pub/sub) Service Bus (MS MQ protocol)

Finally, a new feature in SQL Server called SQL Sever Always On synchronizes AQL Server instances across a network.



Virtual Network Features

- Hosted VPN Gateway that enables site-to-site connectivity
  - Automated provisioning & management
  - Support existing on-premises VPN devices
- Customer-managed private virtual networks within Windows Azure
  - "Bring your own IPv4 addresses"
  - Control over placement of Windows Azure Roles within the network
  - Stable IPv4 addresses for VMs
- Use on-premise DNS servers for name resolution
  - Enables customers to use their on-premise DNS servers for name resolution
  - Enables VMs running in Windows Azure to be joined to corporate domains running on-premise (use your on-premise Active Directory)

This slide describes Virtual Network Features.

VNET requires Hosted VPN Gateway that enables site-to-site connectivity

•Automated provisioning & management

•Support existing on-premises VPN devices

It provides for Customer-managed private virtual networks within Windows Azure

•"Bring your own IPv4 addresses"

•Control over placement of Windows Azure Roles within the network

•Stable IPv4 addresses for VMs

•Only provides for IP Addresses in a Virtualized Network

•It also allows one to Carve out IP subnets with a Virtualized Network

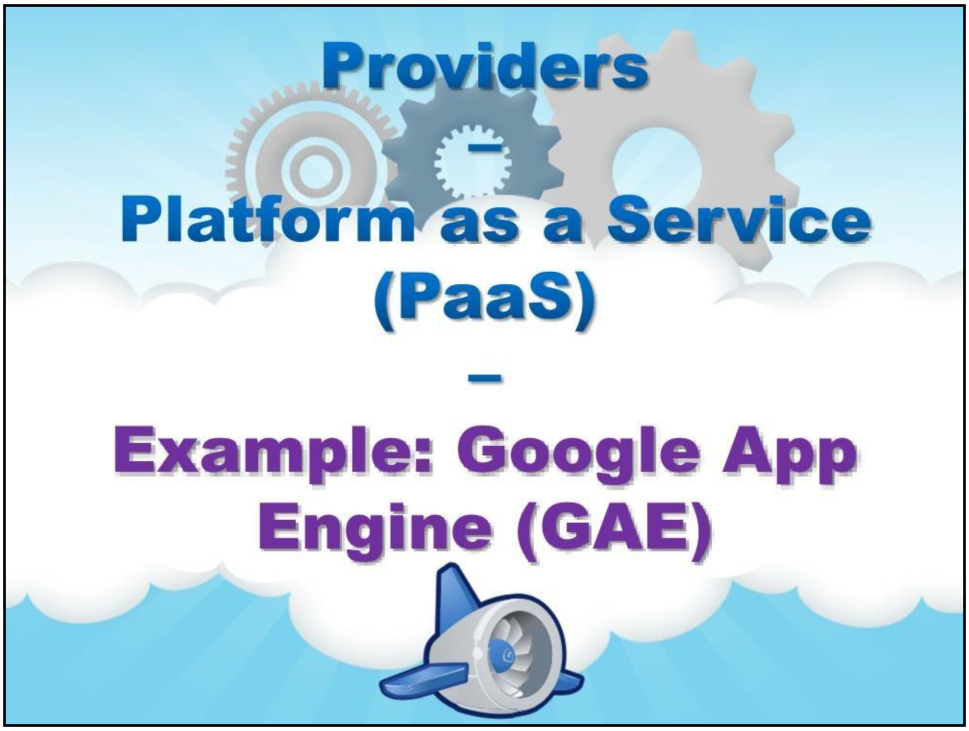
•Cavetas: overlapping subnets are not allowed, and the IP address stays with the VM for it's lifetime

This allows customers to use on-premise DNS servers for name resolution

•Enables customers to use their on-premise DNS servers for name resolution

•Enables VMs running in Windows Azure to be joined to corporate domains running on-premise (use your on-premise Active Directory)

Microsoft manages the gateway at the customer site, which is software, and runs in active / passive mode for high availability.



Providers

–

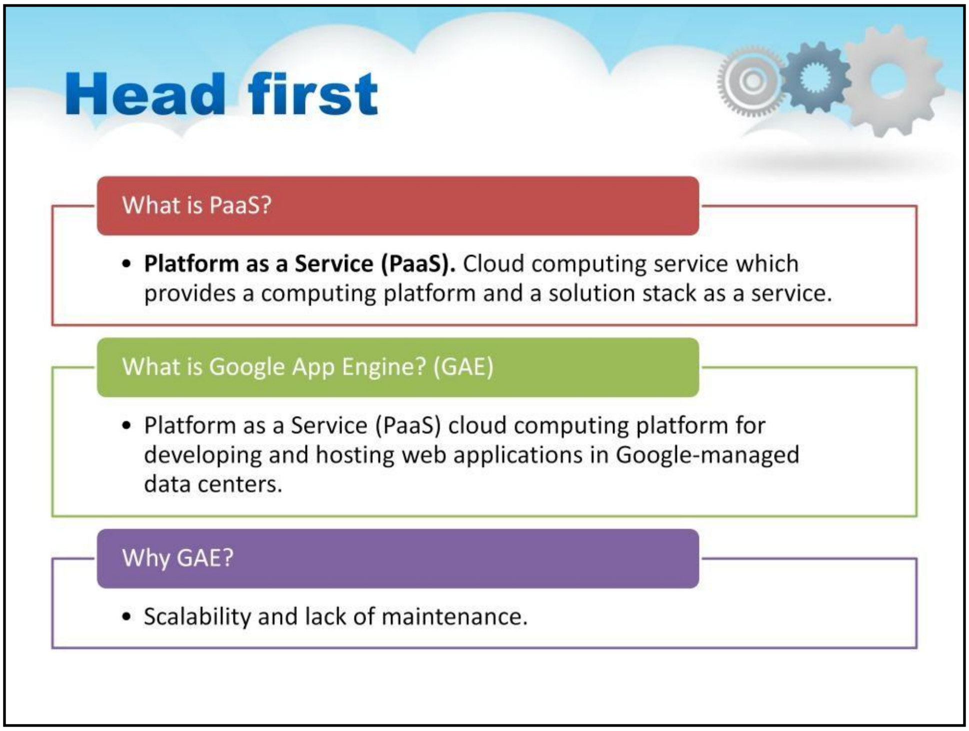
Platform as a Service (PaaS)

–

Example: Google App Engine (GAE)

Example: Google App Engine (GAE)

The second example is dedicated to Google App Engine (GAE) that represents an example of Platform as a Service (PaaS).



Head first

What is PaaS?

- **Platform as a Service (PaaS).** Cloud computing service which provides a computing platform and a solution stack as a service.

What is Google App Engine? (GAE)

- Platform as a Service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers.

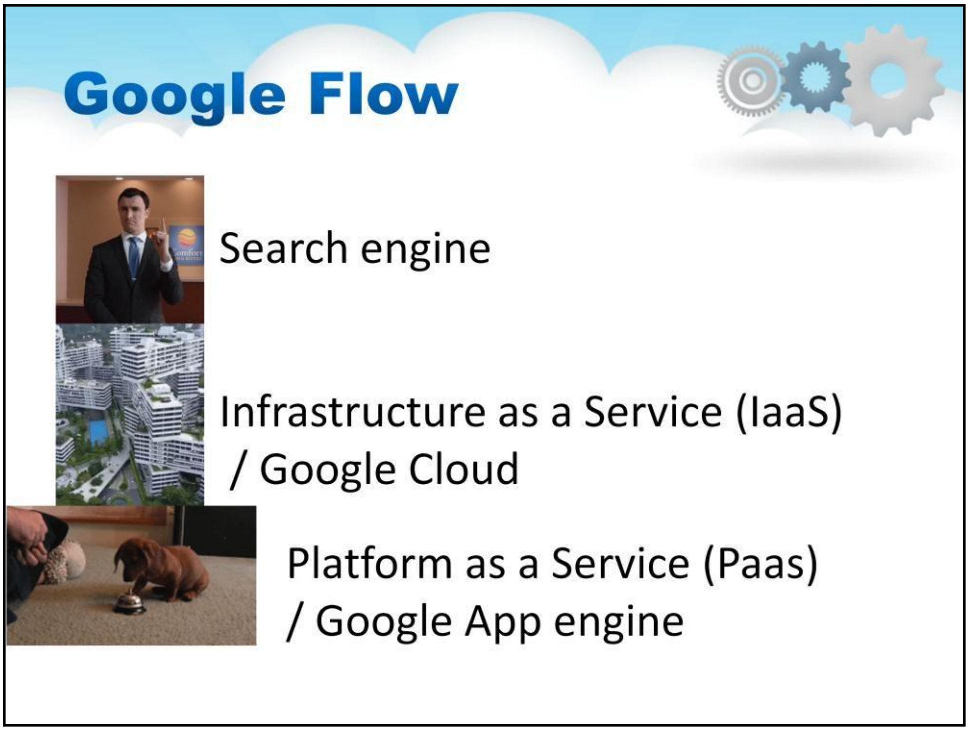
Why GAE?

- Scalability and lack of maintenance.

Normal Cloud Servers (like Amazon AWS) give you a virtual server in the cloud. You get an abstraction of a physical machine, and you have to do everything else yourself. To build a website here, you would need to install an OS (like Linux), then install a webserver (like Apache), then add in an interpreter for your preferred language in the webserver (like modpython for Python), and install a database (like MySQL). Of course, if you want to use multiple servers or multiple databases or multiple webserver for scalability, you're completely on your own. And, you pay per server per month.

One level higher than this are players like Heroku or Webfaction. They have the OS installed, they have the Webserver installed, they have the interpreter installed, and they have the Database installed. You just need to write your app, and wire all the above pieces together. As before, if you want to use multiple servers or multiple databases or multiple webserver for scalability, you're completely on your own. Usually, in such shared hosts, you pay for the "number of instances" that you have running of any application, and the size of each (in terms of memory or bandwidth consumed).

Google AppEngine is one level higher than this. Here, you are not exposed to the webserver or the database. You just get a platform where you start writing Python or Java code (or a whole bunch of other languages are now supported), and there is a data-storage API which you use directly without worrying about which database it is stored in. And here, you pay separately each resource consumed (like memory, bandwidth, GB of storage, MB of data in the database, number of emails sent, etc).



Google Flow

Search engine

Infrastructure as a Service (IaaS) / Google Cloud

Platform as a Service (Paas) / Google App engine

As we all know, Google is at the forefront of what you may call the Internet revolution.

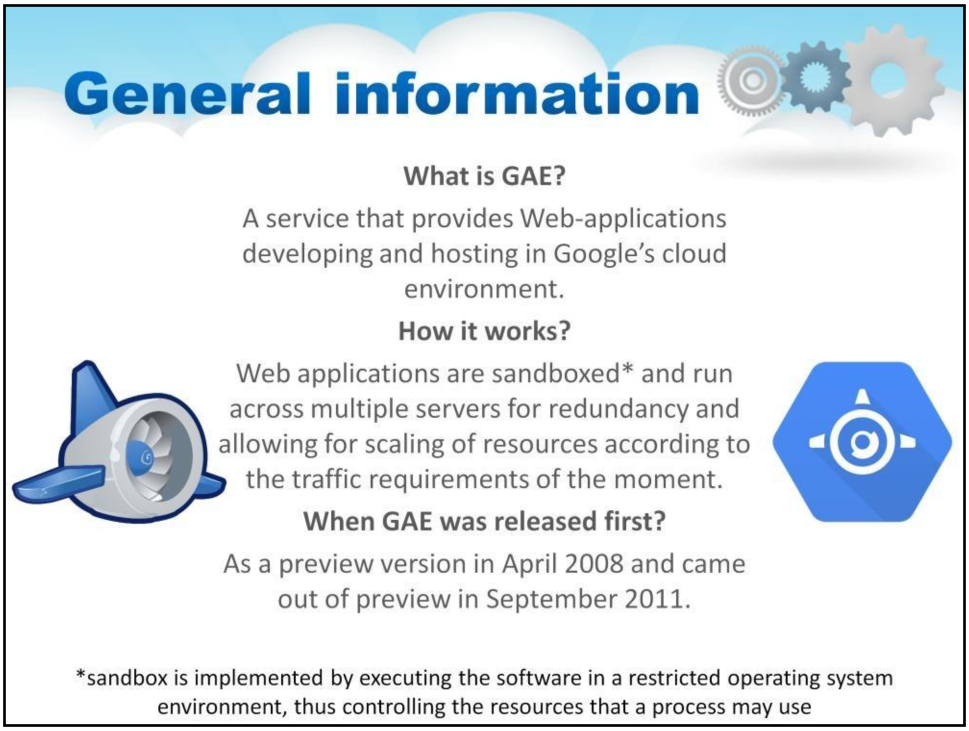
Besides being the number 1 search engine, the company is also leveraging Cloud and has come up with different products to help developers launch new scalable web and mobile apps.

Amongst its various Cloud based products, Google app engine has become quite popular.

The app engine is a Cloud based platform, is quite comprehensive and combines infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). The app engine supports the delivery, testing and development of software on demand in a Cloud computing environment that supports millions of users and is highly scalable.

The company extends its platform and infrastructure to the Cloud through its app engine. It presents the platform to those who want to develop SaaS solutions at competitive costs.

It is a platform-as-a service (PaaS) Cloud computing platform that is fully managed and uses in-built services to run your apps. You can start development almost instantly after downloading the software development kit (SDK). You can go on to the developer's guide right away when you click on the language you wish to develop your app in.



General information

What is GAE?

A service that provides Web-applications developing and hosting in Google's cloud environment.

How it works?

Web applications are sandboxed* and run across multiple servers for redundancy and allowing for scaling of resources according to the traffic requirements of the moment.

When GAE was released first?

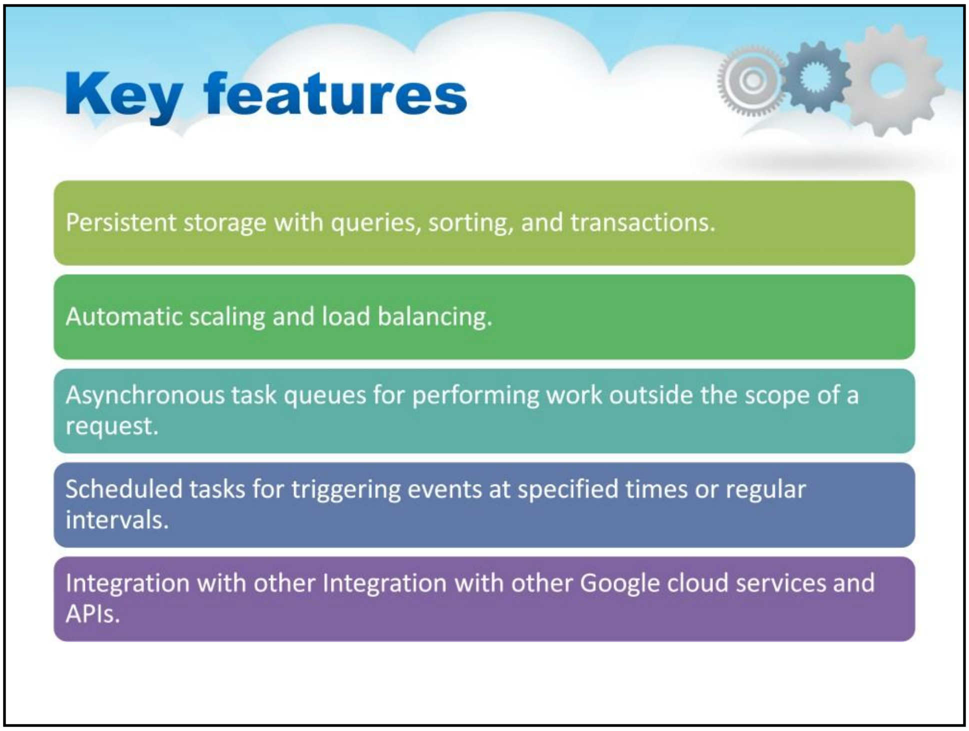
As a preview version in April 2008 and came out of preview in September 2011.

*sandbox is implemented by executing the software in a restricted operating system environment, thus controlling the resources that a process may use

Google App Engine (often referred to as GAE) is a web framework and cloud computing platform for developing and hosting web applications in Google-managed data centers.

Applications are sandboxed and run across multiple servers.

GAE was first released as a preview version in April 2008 and came out of preview in September 2011.



Key features

Persistent storage with queries, sorting, and transactions.

Automatic scaling and load balancing.

Asynchronous task queues for performing work outside the scope of a request.

Scheduled tasks for triggering events at specified times or regular intervals.

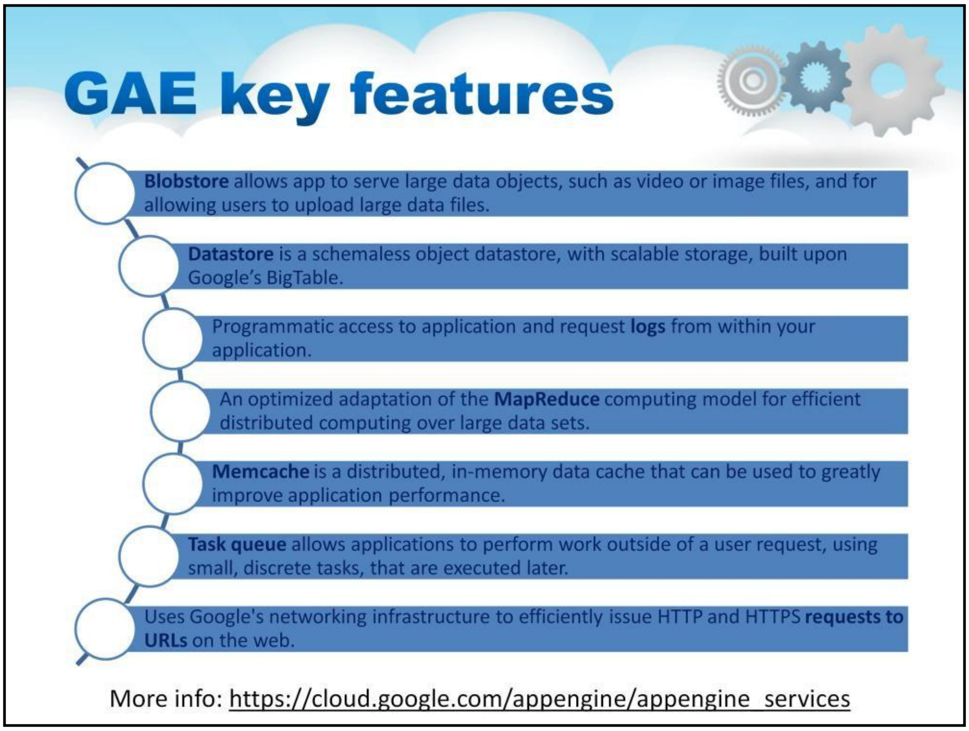
Integration with other Integration with other Google cloud services and APIs.

The App Engine standard environment is based on container instances running on Google's infrastructure. Containers are preconfigured with one of several available runtimes (Java 7, Java 8, Python 2.7, Go and PHP). Each runtime also includes libraries that support App Engine Standard APIs. For many applications, the standard environment runtimes and libraries might be all you need.

The App Engine standard environment makes it easy to build and deploy an application that runs reliably even under heavy load and with large amounts of data. It includes the following features which are shown in this figure:

✓Persistent storage with queries, sorting, and transactions.
✓Automatic scaling and load balancing.
✓Asynchronous task queues for performing work outside the scope of a request.
✓Scheduled tasks for triggering events at specified times or regular intervals.
✓Integration with other Google cloud services and APIs.

Applications run in a secure, sandboxed environment, allowing App Engine standard environment to distribute requests across multiple servers, and scaling servers to meet traffic demands. Your application runs within its own secure, reliable environment that is independent of the hardware, operating system, or physical location of the server.



GAE key features

- **Blobstore** allows app to serve large data objects, such as video or image files, and for allowing users to upload large data files.
- **Datastore** is a schemaless object datastore, with scalable storage, built upon Google's BigTable.
- Programmatic access to application and request **logs** from within your application.
- An optimized adaptation of the **MapReduce** computing model for efficient distributed computing over large data sets.
- **Memcache** is a distributed, in-memory data cache that can be used to greatly improve application performance.
- **Task queue** allows applications to perform work outside of a user request, using small, discrete tasks, that are executed later.
- Uses Google's networking infrastructure to efficiently issue HTTP and HTTPS **requests to URLs** on the web.

More info: https://cloud.google.com/appengine/appengine_services

On this slide you can see a list of GAE key features.

Data storage. Google App Engine storage consists of DataStore and BlobStore. BlobStore serve data objects larger than allowable for Datastore.

Logs. App Engine standard environment maintains two kinds of logs: application logs contain arbitrary messages with a timestamp and log level, request logs contain entries for each request handled by your app.

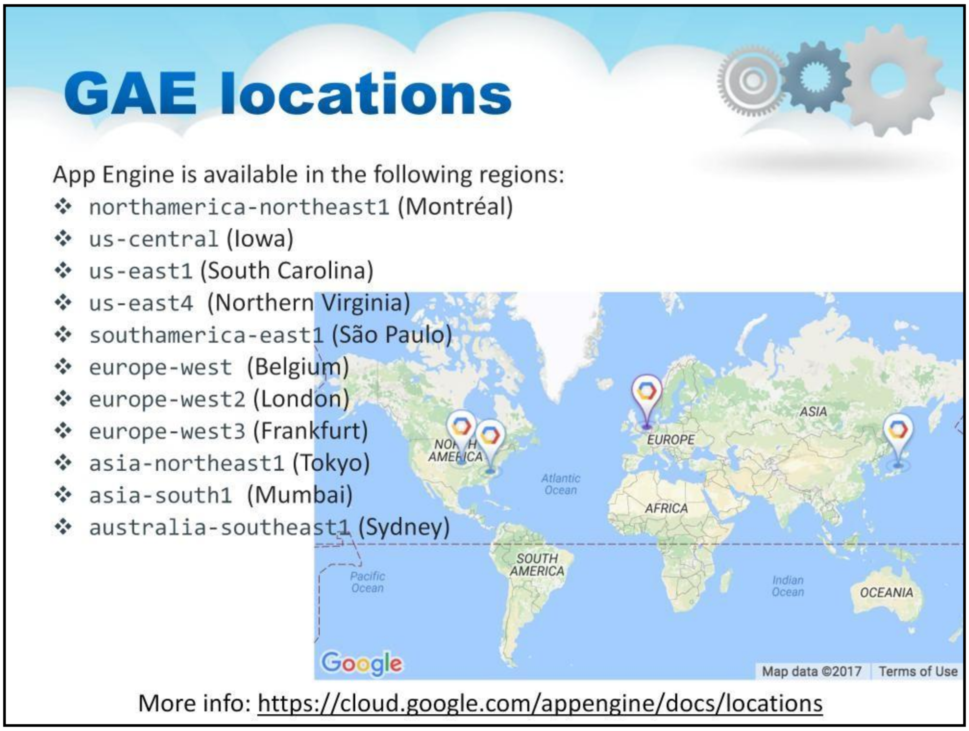
MapReduce. MapReduce is an open source library that is built on top of App Engine services. It provides a programming model for large-scale distributed data processing, automatic parallelization and distribution within existing codebase and other useful features. MapReduce is available on GitHub for Java and Python. It is described in the separate module and lectures.

Memcache. App Engine standard environment supports two classes of the memcache service: shared memcache provides cache capacity on a best-effort basis and is subject to the overall demand of all applications served by App Engine; dedicated memcache provides a fixed cache capacity assigned exclusively to client's application.

Task Queue. If some application needs to execute some background work, it can use the Task Queue API to organize that work into small, discrete units, called tasks. The app adds tasks to task queues to be executed later.

Communication. App Engine standard environment applications can communicate with other applications or access other resources on the web by fetching URLs.

More information can be obtained by the reference shown here.



GAE locations

App Engine is available in the following regions:

- northamerica-northeast1 (Montréal)
- us-central (Iowa)
- us-east1 (South Carolina)
- us-east4 (Northern Virginia)
- southamerica-east1 (São Paulo)
- europe-west (Belgium)
- europe-west2 (London)
- europe-west3 (Frankfurt)
- asia-northeast1 (Tokyo)
- asia-south1 (Mumbai)
- australia-southeast1 (Sydney)

More info: https://cloud.google.com/appengine/docs/locations

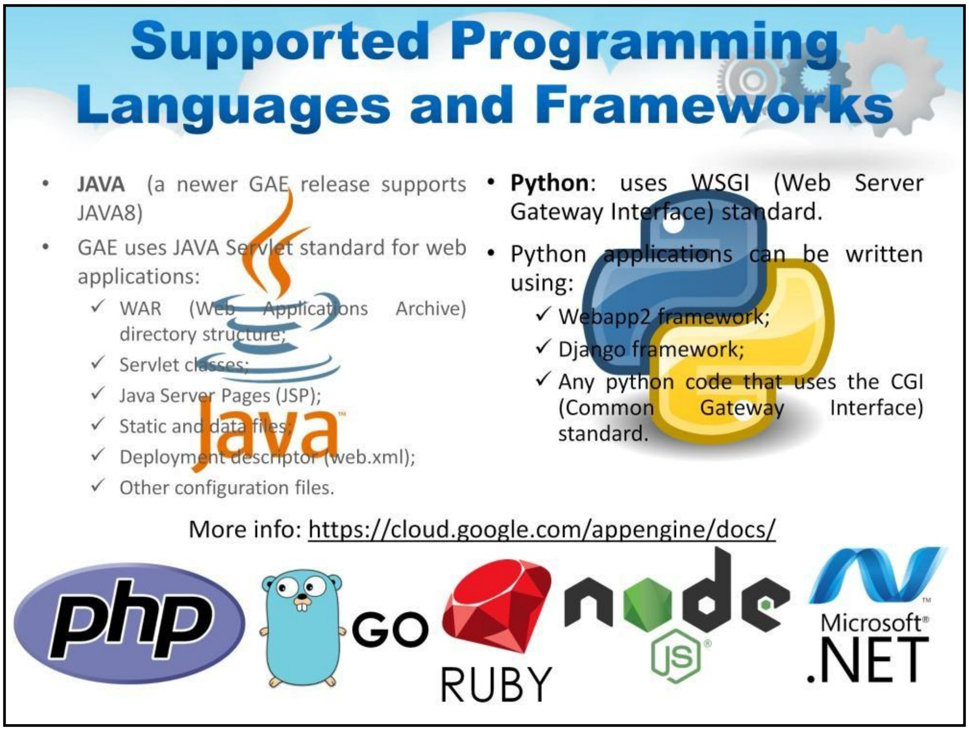
App Engine is regional, which means the infrastructure that runs your apps is located in a specific region and is managed by Google to be redundantly available across all the zones within that region.

Client can choose where to locate applications to meet latency, availability and durability requirements.

Regions are independent geographic areas that consist of zones.

Google Cloud Platform services are available in locations across North America, South America, Europe, Asia, and Australia.

More information can be obtained by the reference shown here.



Supported Programming Languages and Frameworks

- **JAVA** (a newer GAE release supports JAVA8)
- GAE uses JAVA Servlet standard for web applications:
  - ✓ WAR (Web Applications Archive) directory structure;
  - ✓ Servlet classes;
  - ✓ Java Server Pages (JSP);
  - ✓ Static and data files;
  - ✓ Deployment descriptor (web.xml);
  - ✓ Other configuration files.
- **Python**: uses WSGI (Web Server Gateway Interface) standard.
- Python applications can be written using:
  - ✓ Webapp2 framework;
  - ✓ Django framework;
  - ✓ Any python code that uses the CGI (Common Gateway Interface) standard.

More info: https://cloud.google.com/appengine/docs/

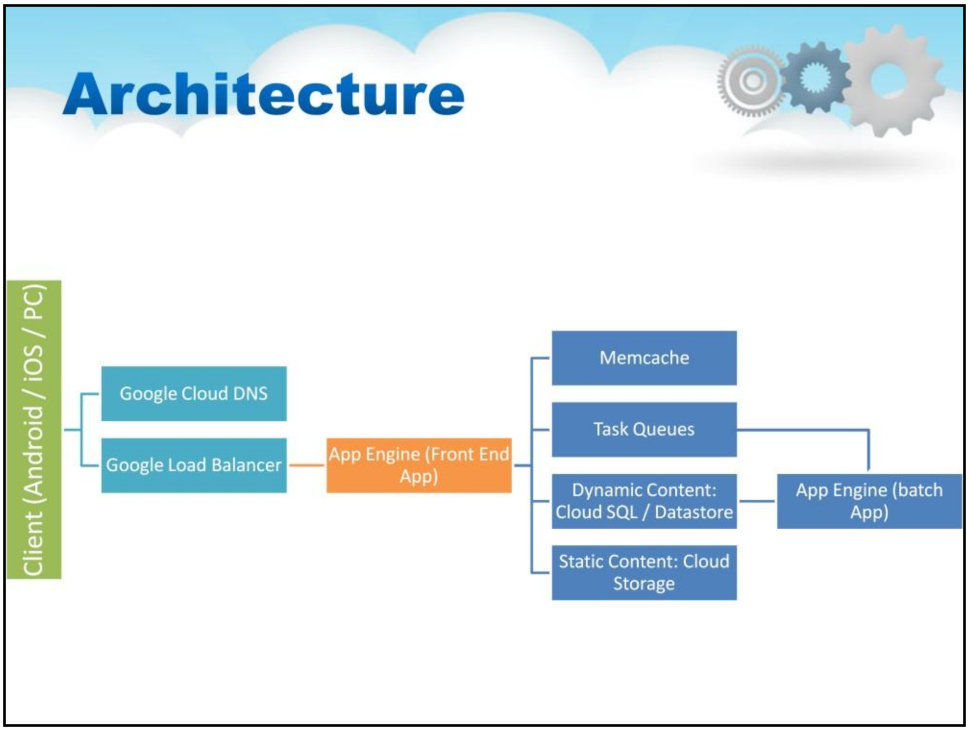
For GAE, supported programming languages include Java (and, by extension, other JVM languages such as Kotlin, Groovy, JRuby, Scala, Clojure), Python, PHP, Go and Ruby. Node.js and .NET Framework are also available in the flexible environment.

Google App Engine supports many Java standards and frameworks. Core to this is the servlet 2.5 technology using the open-source Jetty Web Server along with accompanying technologies such as JSP. A newer release of App Engine Standard Java in Beta supports Java8, Servlet 3.1 and Jetty9.

Python web frameworks that run on Google App Engine include Django, CherryPy, Pyramid, Flask, web2py and webapp2. Any Python framework that supports the WSGI using the CGI adapter can be used to create an application; the framework can be uploaded with the developed application. Third-party libraries written in pure Python may also be uploaded.

Developers can use Go, Java, PHP or Python to write an app engine application. They can develop and test an app locally using the SDK containing tools for deploying apps. Every language has its own SDK and runtime.

More information can be obtained by the reference shown here.



Architecture

Client (Android / iOS / PC)

Google Cloud DNS

Google Load Balancer

App Engine (Front End App)

Memcache

Task Queues

Dynamic Content: Cloud SQL / Datastore

Static Content: Cloud Storage

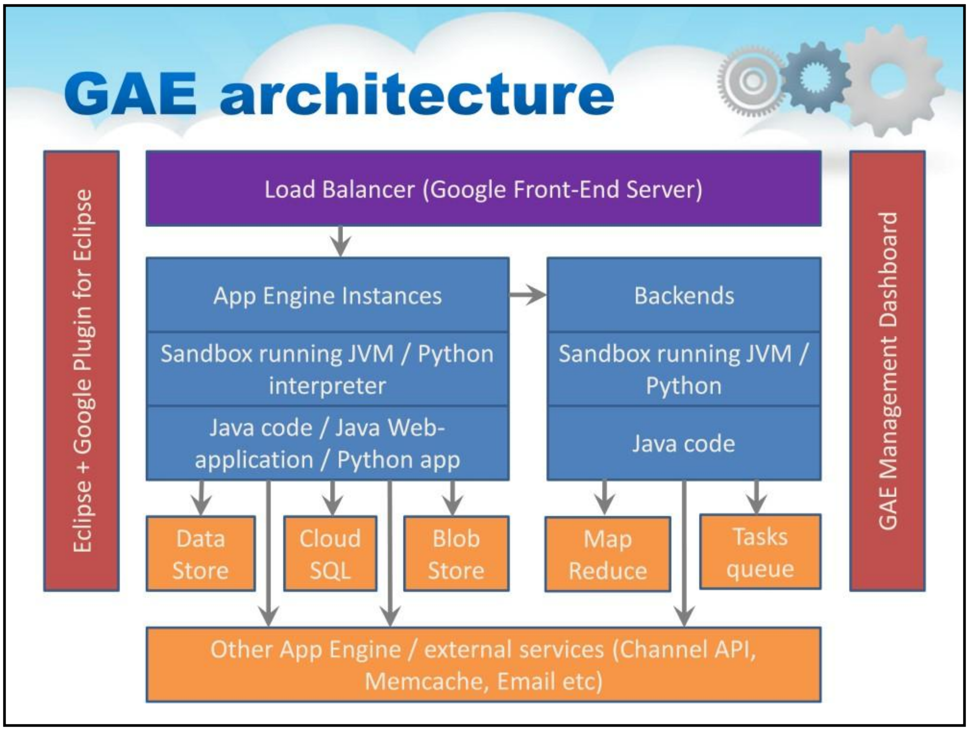
App Engine (batch App)

App Engine is Google's PaaS platform and robust development environment.
The SDK for App Engine supports development and deployment of the application to the cloud.
App Engine supports multiple application versions, which enables easy rollout of new application features and traffic splitting to support A/B testing.

The Memcache and Task Queue services are integrated in the App Engine standard environment. Memcache is an in-memory cache shared across the App Engine instances. This provides extremely high speed access to information cached by the web server (e.g. authentication or account information).

Task Queues provide a mechanism to offload longer running tasks to backend servers, freeing the front end servers to service new user requests.

Finally, App Engine features a built-in load balancer (provided by the Google Load Balancer) which provides transparent Layer 3 and Layer 7 load balancing to applications.



GAE architecture

Load Balancer (Google Front-End Server)

App Engine Instances

Sandbox running JVM / Python interpreter

Java code / Java Web-application / Python app

Backends

Sandbox running JVM / Python

Java code

Data Store

Cloud SQL

Blob Store

Map Reduce

Tasks queue

Other App Engine / external services (Channel API, Memcache, Email etc)

Eclipse + Google Plugin for Eclipse

GAE Management Dashboard

This slide describes how GAE works.

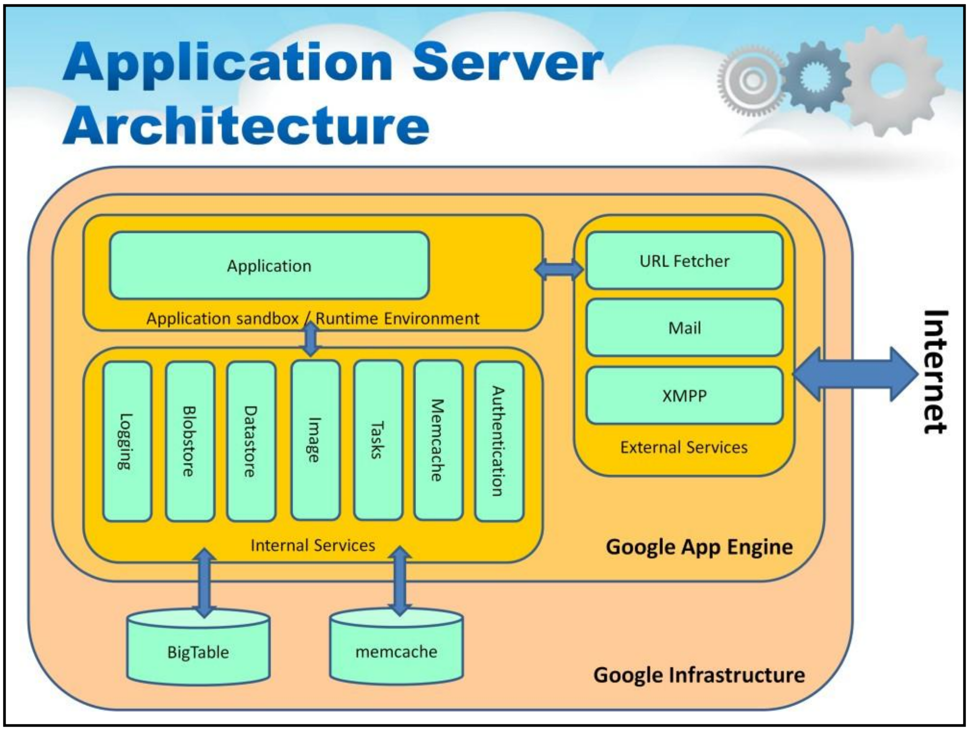
Front-End Server balancing the application between an App Engine Instances and their backends.

Running apps accessing the storages as Datastore, Blobstore and Cloud SQL.

Running code operating with MapReduce library and Tasks queues.

Both apps and code can use other services such as memcache, email etc.

Developers control the processes through the IDE plugin and GAE Management dashboard.



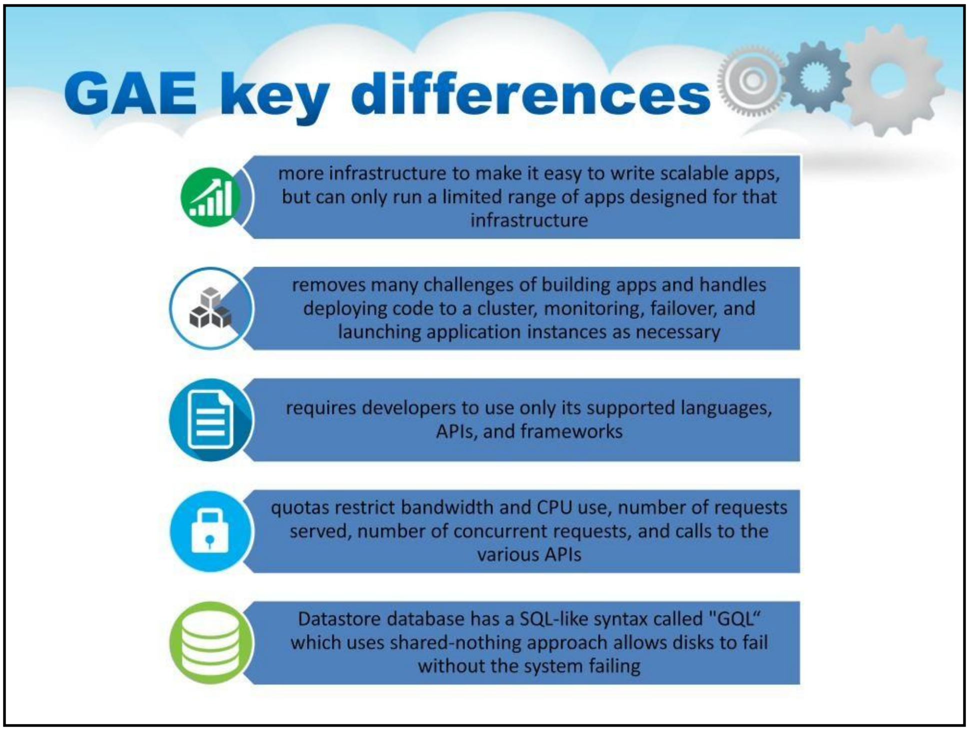
This slide describes how Application Server works.

Application is running on server in a sandbox.

Google App Engine connecting running app with internal and external services.

Internal services are using Google infrastructure implements in Memcache and BigTable technologies.

External services provides connection to the Internet.

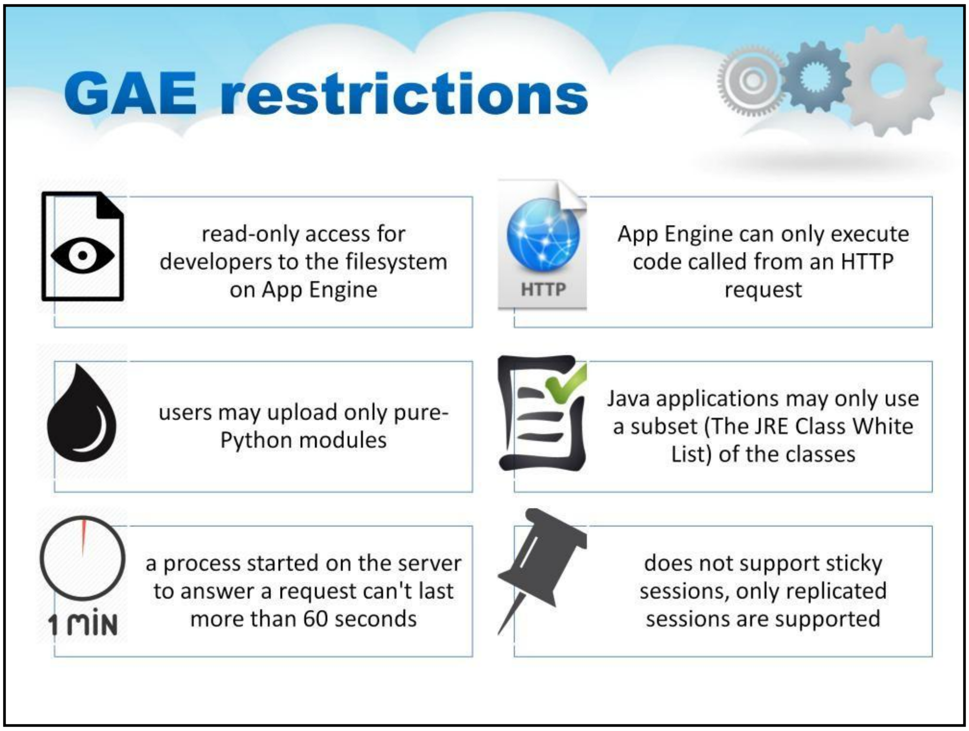


GAE key differences

- more infrastructure to make it easy to write scalable apps, but can only run a limited range of apps designed for that infrastructure
- removes many challenges of building apps and handles deploying code to a cluster, monitoring, failover, and launching application instances as necessary
- requires developers to use only its supported languages, APIs, and frameworks
- quotas restrict bandwidth and CPU use, number of requests served, number of concurrent requests, and calls to the various APIs
- Datastore database has a SQL-like syntax called "GQL“ which uses shared-nothing approach allows disks to fail without the system failing

The key differences of Google App Engine from other providers are as follows:

- Compared to other scalable hosting services such as Amazon EC2, GAE provides more infrastructure to make it easy to write scalable applications, but can only run a limited range of applications designed for that infrastructure.
- App Engine's infrastructure removes many of the system administration and development challenges of building applications to scale to hundreds of requests per second and beyond. Google handles deploying code to a cluster, monitoring, failover, and launching application instances as necessary.
- While other services let users install and configure nearly any *NIX compatible software, App Engine requires developers to use only its supported languages, APIs, and frameworks. Google Cloud SQL can be used for App Engine applications requiring a relational MySQL compatible database backend.
- Per-day and per-minute quotas restrict bandwidth and CPU use, number of requests served, number of concurrent requests, and calls to the various APIs, and individual requests are terminated if they take more than 60 seconds or return more than 32MB of data.
- Google App Engine's integrated Google Cloud Datastore database has a SQL-like syntax called "GQL". GQL does not support the Join statement. Instead, one-to-many and many-to-many relationships can be accomplished using ReferenceProperty(). This shared-nothing approach allows disks to fail without the system failing.



GAE restrictions

read-only access for developers to the filesystem on App Engine

App Engine can only execute code called from an HTTP request

users may upload only pure-Python modules

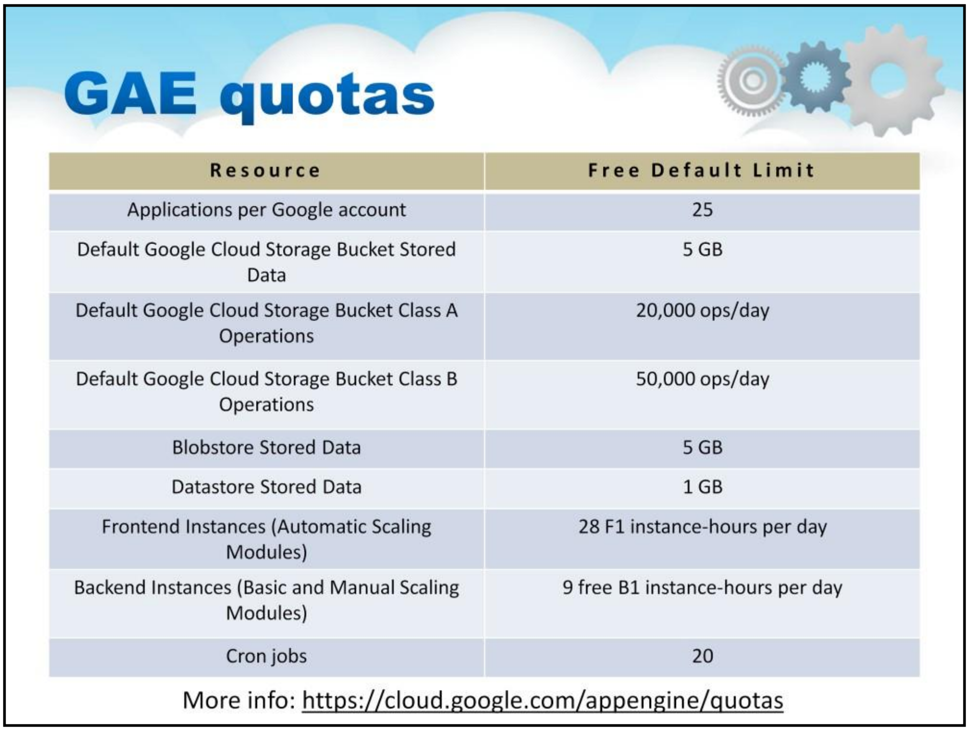
Java applications may only use a subset (The JRE Class White List) of the classes

a process started on the server to answer a request can't last more than 60 seconds

does not support sticky sessions, only replicated sessions are supported

Developers should take into account the following restrictions:

- Developers have read-only access to the filesystem on App Engine. Applications can use only virtual filesystems, like GAE-filestore.
- App Engine can only execute code called from an HTTP request (scheduled background tasks allow for self calling HTTP requests).
- Users may upload arbitrary Python modules, but only if they are pure-Python; C modules are not supported.
- Java applications may only use a subset (listed in JRE Class White List) of the classes from the JRE standard edition. This restriction does not exist with the App Engine Standard Java8 runtime.
- A process started on the server to answer a request can't last more than 60 seconds (with the 1.4.0 release, this restriction does not apply to background jobs anymore).
- Does not support sticky sessions (a.k.a. session affinity), only replicated sessions are supported including limitation of the amount of data being serialized and time for session serialization.



GAE quotas

Resource	Free Default Limit
Applications per Google account	25
Default Google Cloud Storage Bucket Stored Data	5 GB
Default Google Cloud Storage Bucket Class A Operations	20,000 ops/day
Default Google Cloud Storage Bucket Class B Operations	50,000 ops/day
Blobstore Stored Data	5 GB
Datastore Stored Data	1 GB
Frontend Instances (Automatic Scaling Modules)	28 F1 instance-hours per day
Backend Instances (Basic and Manual Scaling Modules)	9 free B1 instance-hours per day
Cron jobs	20

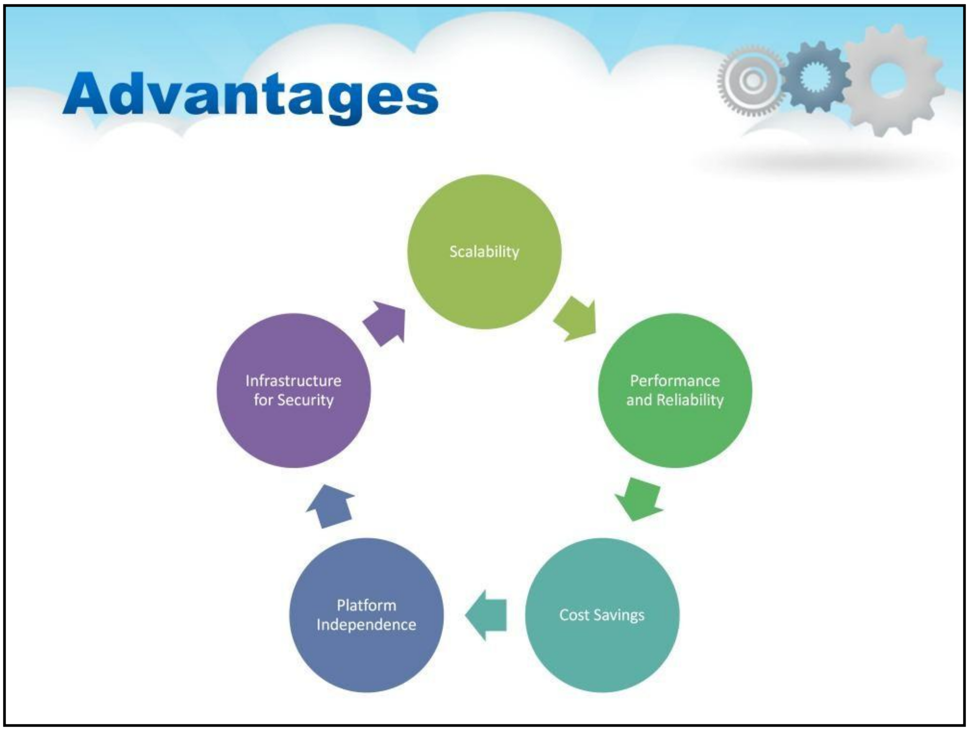
More info: https://cloud.google.com/appengine/quotas

Google App Engine is free up to a certain level of consumed resources.

On this slide you can see a table with resources and corresponding free default limits.

Fees are charged for additional storage, bandwidth, or instance hours required by the application etc.

More information can be obtained by the reference shown here.



Advantages

This slide describe the numerous **advantages** of the app engine:

Infrastructure for Security

Internet infrastructure that Google has is probably the most secure in the world. You can be sure that your app will be available to users worldwide at all times since Google has several hundred servers globally. Google's security and privacy policies are applicable to the apps developed using Google's infrastructure.

Scalability

For any app's success, this is among the deciding factors. Google creates its own apps using GFS, Big Table and other such technologies. You only have to write the code for the app and Google looks after the testing on account of the automatic scaling feature that the app engine has. Regardless of the amount of data or number of users that your app stores, the app engine can meet your needs by scaling up or down as required.

Performance and Reliability

In the past 15 years, the company has created new benchmarks based on its services' and products' performance. The app engine provides the same reliability and performance as any other Google product.

Cost Savings

You don't have to hire engineers to manage your servers or to do that yourself. You can invest the money saved in to other parts of your business.

Disadvantages

You're limited to in language: Java, PHP, Go and Python

You're limited to using AppEngine's services (queues, search, memcache, logging etc). If you need anything beyond that - it'll be hard.

You're limited to appngine's runtime

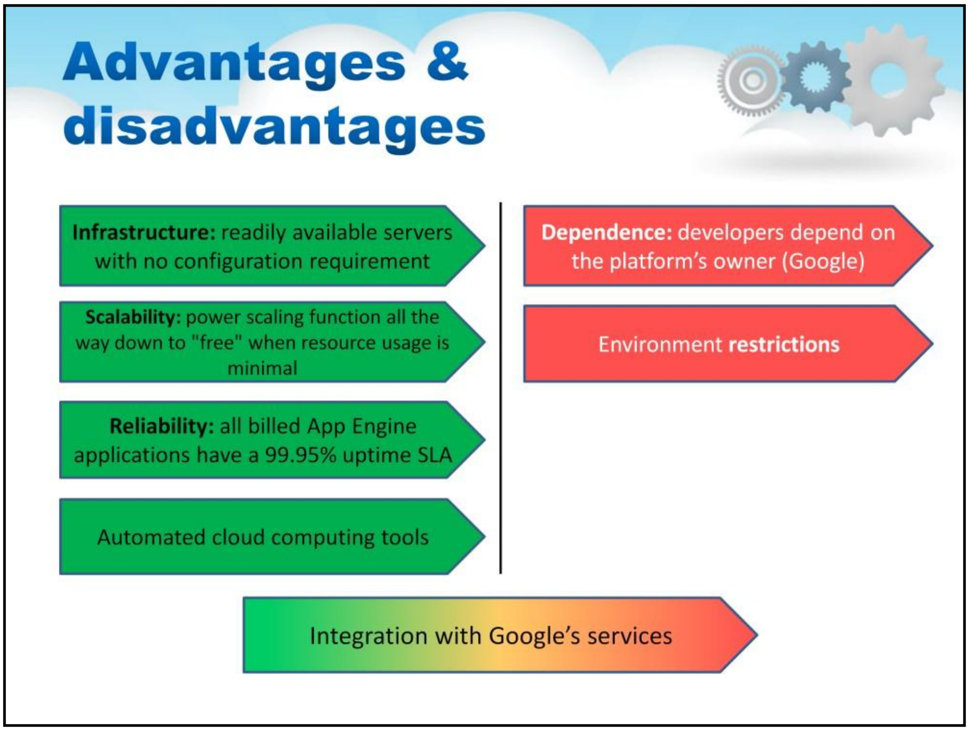
This slide describe some **disadvantages** of the app engine:

You're limited to languages: Java, PHP, Go and Python

You're limited to using AppEngine's services (queues, search, memcache, logging etc). If you need anything beyond that - it'll be hard.

You're limited to AppEngine's runtime (for example: in Python you can't have any 3rd party dependencies that are in C... image processing, etc - out of the question)

The bottom line is that you're limited - you have the set of services AppEngine provide and you're locked into those services. Anything beyond that will require some hacking to circumvent its limitations.



Advantages & disadvantages

Infrastructure: readily available servers with no configuration requirement

Scalability: power scaling function all the way down to "free" when resource usage is minimal

Reliability: all billed App Engine applications have a 99.95% uptime SLA

Automated cloud computing tools

Dependence: developers depend on the platform's owner (Google)

Environment **restrictions**

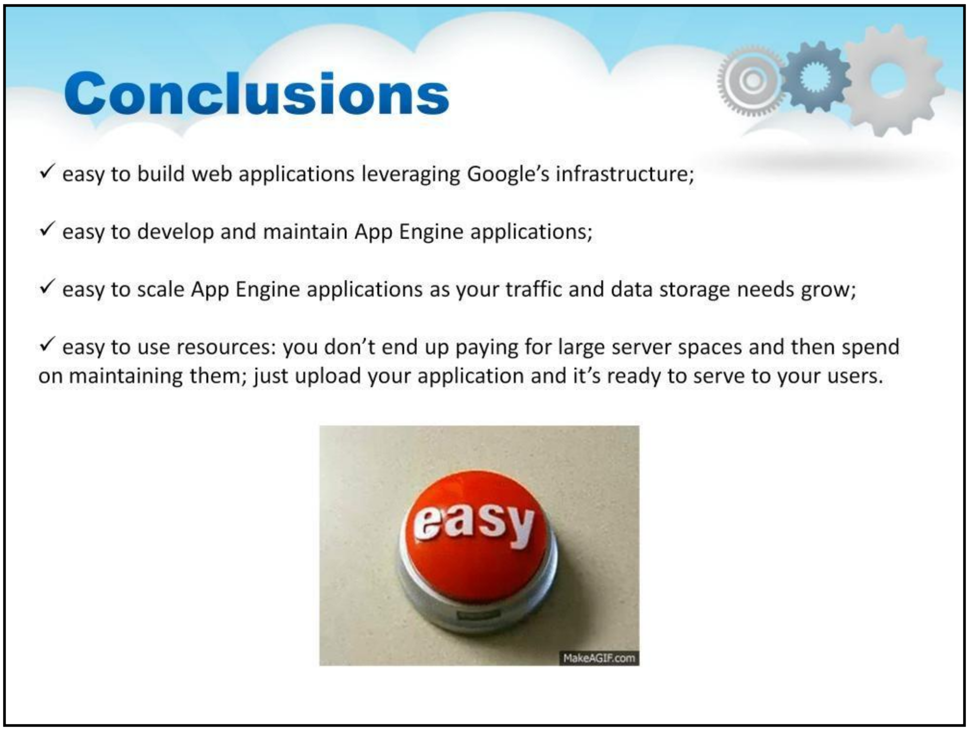
Integration with Google's services

Finally, lets compare the advantages and disadvantages of Google App Engine Usage.

The **advantages** include ready infrastructure, scalability, reliability and wide range of tools.

The **disadvantages** include dependence on Google platform and restrictions, described earlier.

Integration with Google's service may be advantage and disadvantage at the same time.



Conclusions

✓ easy to build web applications leveraging Google's infrastructure;

✓ easy to develop and maintain App Engine applications;

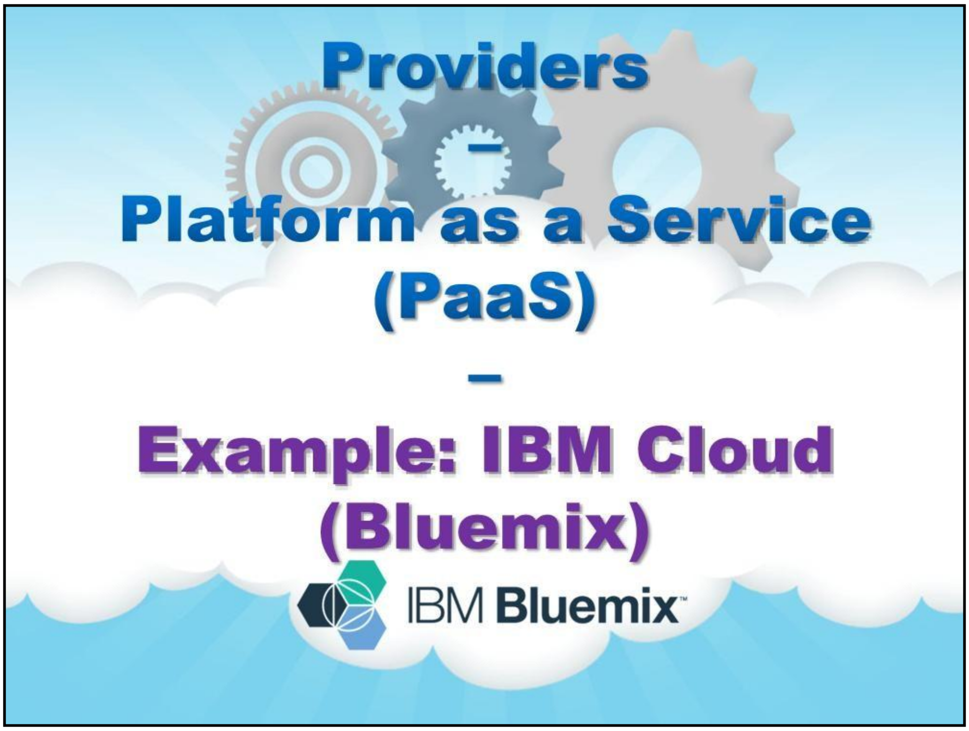
✓ easy to scale App Engine applications as your traffic and data storage needs grow;

✓ easy to use resources: you don't end up paying for large server spaces and then spend on maintaining them; just upload your application and it's ready to serve to your users.

Google App Engine enables you to build web applications for your business leveraging Google's infrastructure.

App Engine applications are easy to develop, maintain, and can scale as your traffic and data storage needs grow.

With App Engine, you don't end up paying for large server spaces and then spend on resources maintaining them. You just upload your application, and it's ready to serve to your users. Rest is taken care by Google Cloud.



Providers

–

Platform as a Service (PaaS)

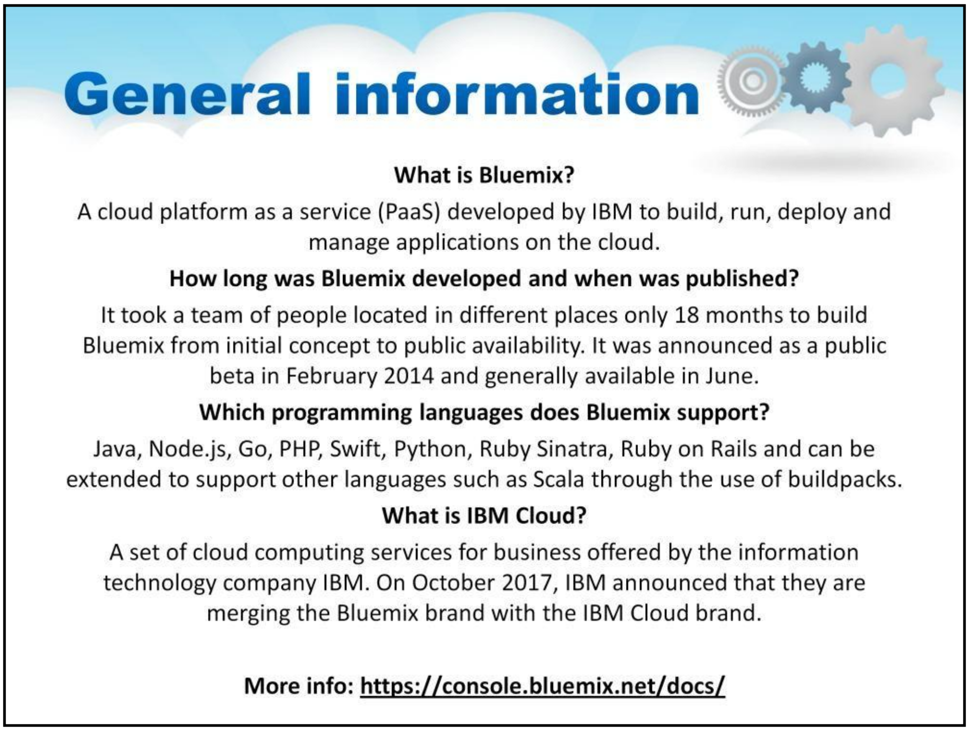
–

Example: IBM Cloud (Bluemix)

IBM Bluemix™

Example: IBM Cloud (Bluemix)

The next example is dedicated to IBM Cloud (the former name was IBM Bluemix) that represents an example of Platform as a Service (PaaS).



General information

What is Bluemix?

A cloud platform as a service (PaaS) developed by IBM to build, run, deploy and manage applications on the cloud.

How long was Bluemix developed and when was published?

It took a team of people located in different places only 18 months to build Bluemix from initial concept to public availability. It was announced as a public beta in February 2014 and generally available in June.

Which programming languages does Bluemix support?

Java, Node.js, Go, PHP, Swift, Python, Ruby Sinatra, Ruby on Rails and can be extended to support other languages such as Scala through the use of buildpacks.

What is IBM Cloud?

A set of cloud computing services for business offered by the information technology company IBM. On October 2017, IBM announced that they are merging the Bluemix brand with the IBM Cloud brand.

More info: https://console.bluemix.net/docs/

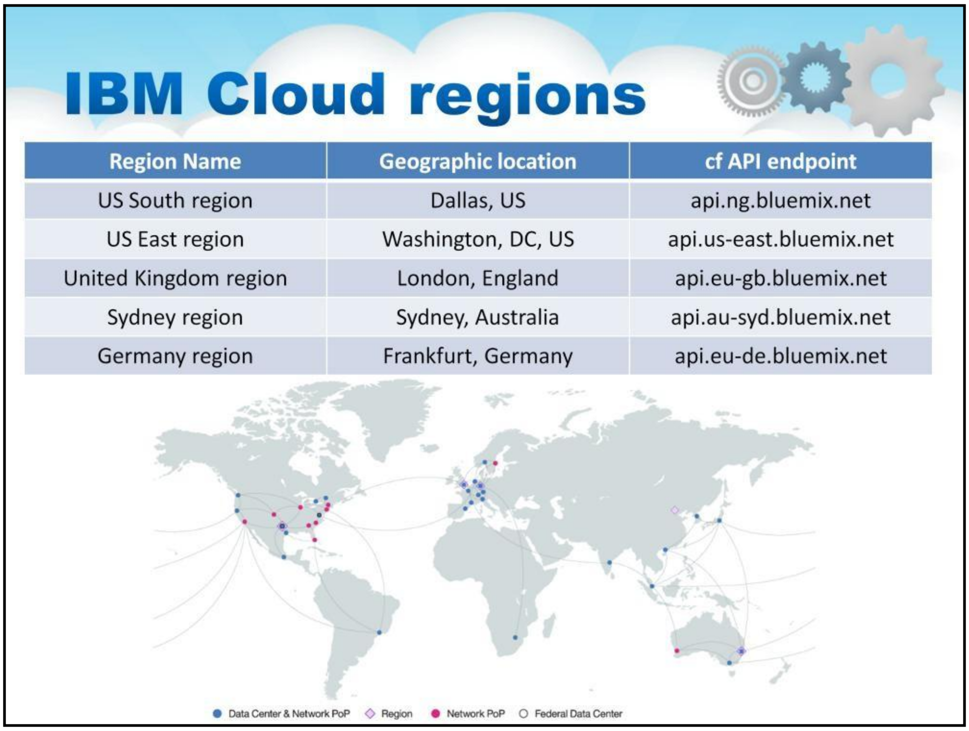
IBM Bluemix is a cloud platform as a service (PaaS) developed by IBM to build, run, deploy and manage applications on the cloud.

It took a team of people located in different places only 18 months to build Bluemix. It was announced as a public beta in February 2014 and generally available in June.

Bluemix supports Java, Node.js, Go, PHP, Swift, Python, Ruby Sinatra, Ruby on Rails and can be extended to support other languages such as Scala through the use of buildpacks.

On October 2017, IBM announced that they are merging the Bluemix brand with the IBM Cloud brand.
That is why we will investigate the IBM Cloud docs for Bluemix.

More information can be obtained by the reference shown here.

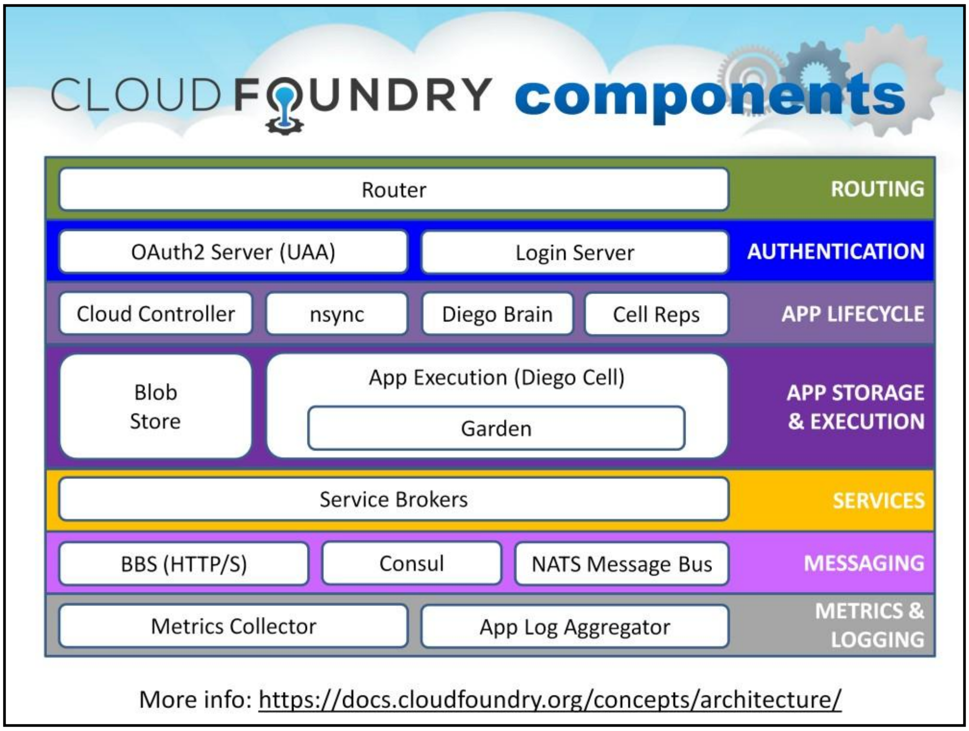


IBM Cloud regions

Region Name	Geographic location	cf API endpoint
US South region	Dallas, US	api.ng.bluemix.net
US East region	Washington, DC, US	api.us-east.bluemix.net
United Kingdom region	London, England	api.eu-gb.bluemix.net
Sydney region	Sydney, Australia	api.au-syd.bluemix.net
Germany region	Frankfurt, Germany	api.eu-de.bluemix.net

IBM Cloud offers nearly 60 data centers into 6 regions and 18 availability zones globally. around the world to help you meet the needs of your global business.

Its possible to deploy apps to different regions for latency or security consideration, with the option to deploy to one region or across multiple regions.



CLOUD FOUNDRY components

Router				ROUTING
OAuth2 Server (UAA)	Login Server			AUTHENTICATION
Cloud Controller	nsync	Diego Brain	Cell Reps	APP LIFECYCLE
Blob Store	App Execution (Diego Cell) Garden			APP STORAGE & EXECUTION
Service Brokers				SERVICES
BBS (HTTP/S)	Consul	NATS Message Bus		MESSAGING
Metrics Collector	App Log Aggregator			METRICS & LOGGING

More info: https://docs.cloudfoundry.org/concepts/architecture/

IBM Bluemix is a PaaS offering based on the Cloud Foundry open source project. Cloud Foundry platform consists of the next components.

> **Note**: here Diego is the container management system, and Diego Cell directly manages and maintains Tasks and Requests. Diego Brain distribute Tasks and Requests to Diego Cells.

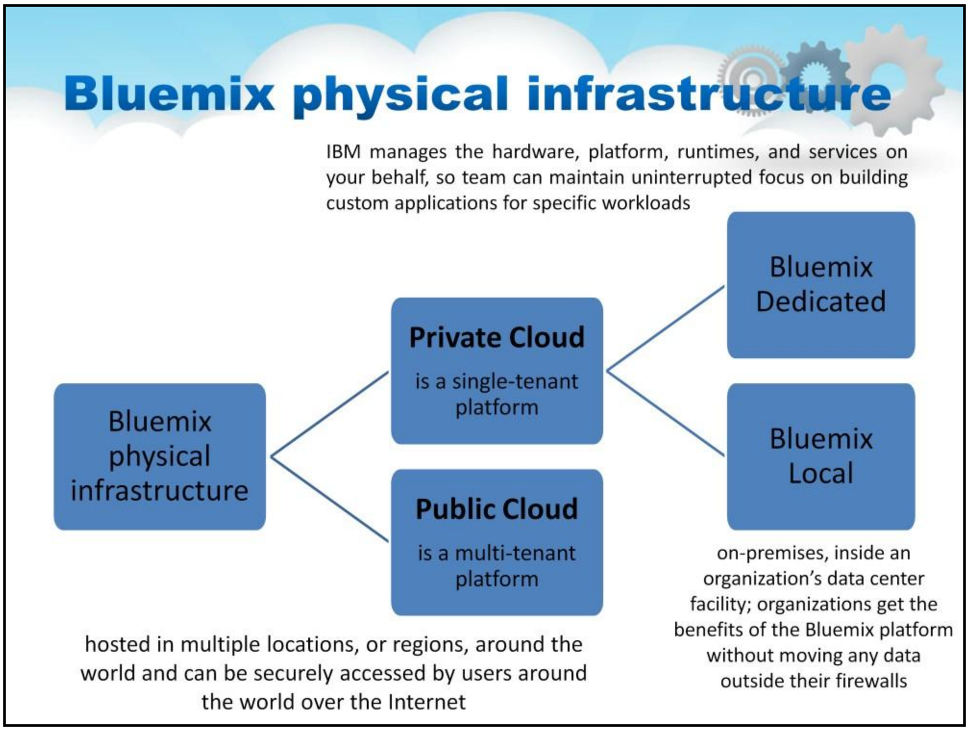
Router routes incoming traffic to the appropriate component, either a Cloud Controller component or a hosted application running on a Diego Cell.
OAuth2 server (User Account and Authentication server **-** UAA) and Login Server

work together to provide identity management.
Cloud Controller then directs the Diego Brain through the CC-Bridge components to coordinate individual Diego cells to stage and run applications.
Blobstore is a repository for large binary files.

Garden containers. Application instances, application tasks, and staging tasks all run as Garden containers on the Diego Cell VMs.
Service broker is responsible for providing the particular service instance.
Diego's Bulletin Board System (BBS) stores frequently updated and disposable data.
NATS Message Bus uses the NATS protocol to broadcast the latest routing tables to the routers.

Consul server stores longer-lived control data.

The **metrics collector** gathers metrics and statistics from the components.
The **App Log Aggregator** streams application logs to developers.



Bluemix physical infrastructure

IBM manages the hardware, platform, runtimes, and services on your behalf, so team can maintain uninterrupted focus on building custom applications for specific workloads

Bluemix physical infrastructure

Private Cloud

is a single-tenant platform

Public Cloud

is a multi-tenant platform

Bluemix Dedicated

Bluemix Local

on-premises, inside an organization's data center facility; organizations get the benefits of the Bluemix platform without moving any data outside their firewalls

hosted in multiple locations, or regions, around the world and can be securely accessed by users around the world over the Internet

Increasingly the most pragmatic and practical mode for enterprises is a **hybrid cloud strategy**.

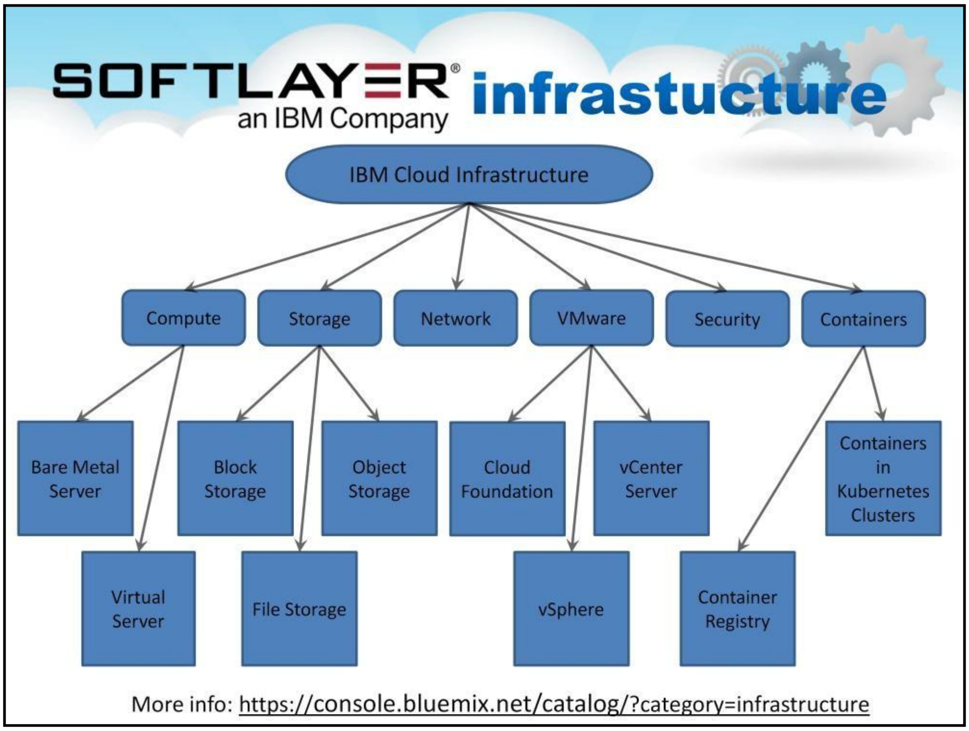
This allows them to leverage data and IT assets that already exist and bring new cloud services as well as moving some existing functionality to the cloud.

To support this IBM offers public and private cloud-based platforms for building, running and managing applications.

Public cloud is multi-tenant, and hosted in a vendor's data center.

Private cloud is single-tenant, and/or hosted in a corporate data center.

Allow for further flexibility, Bluemix environments can be deployed in a single-tenant private cloud environment (**Bluemix Dedicated**)
or
private on-premises in a customer data center (**Bluemix Local**).



SOFTLAYER® an IBM Company infrastucture

IBM Cloud Infrastructure

Compute

Storage

Network

VMware

Security

Containers

Bare Metal Server

Virtual Server

Block Storage

File Storage

Object Storage

Cloud Foundation

vSphere

vCenter Server

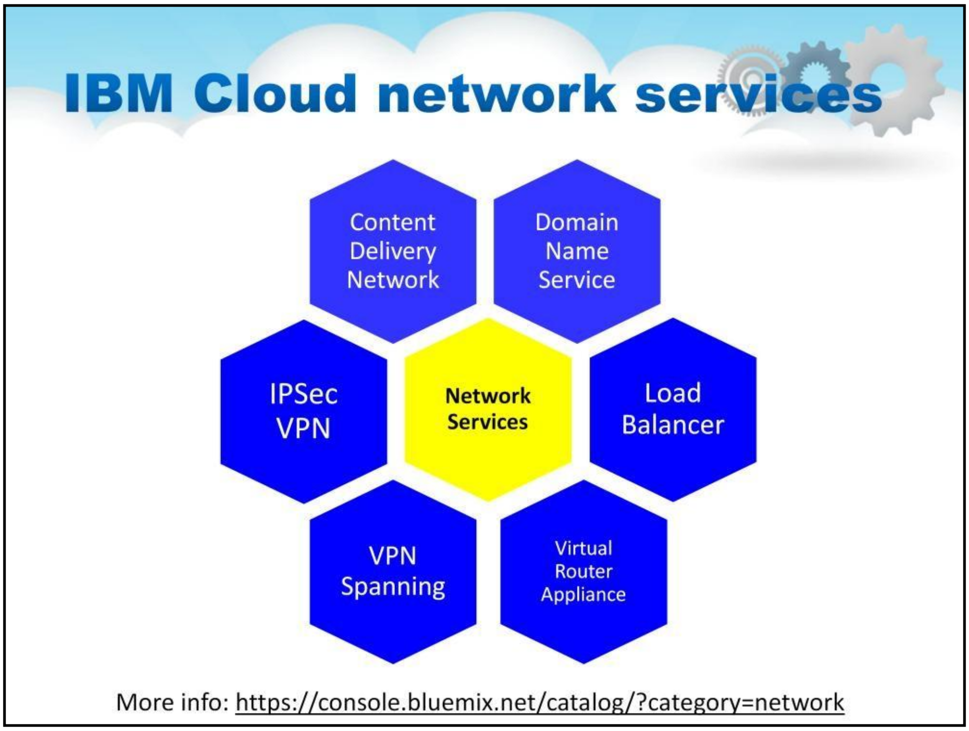
Container Registry

Containers in Kubernetes Clusters

More info: https://console.bluemix.net/catalog/?category=infrastructure

Bluemix runs on the SoftLayer infrastructure. On this slide you can see its main components.
Bare metal servers provide the raw horsepower demanded for processor-intensive and disk I/O-intensive workloads.
Virtual Servers can be deployed in a matter of minutes from virtual server images in the geographic region that makes sense for particular workloads.
Block Storage is a persistent iSCSI based storage with high-powered performance and capacity up to 12TB.
File Storage is a fast and flexible NFS-based file storage with capacity options from 20GB to 12TB.
Object Storage is a highly scalable cloud storage service, designed for storing, managing and accessing data via self-service portal and RESTful APIs.
VMware **Cloud Foundation** on IBM Cloud brings together VMware vSphere, vSAN, and NSX into a natively-integrated stack of virtual compute, virtual storage, and virtual networking built on IBM Cloud compute dedicated infrastructure.
VMware **vSphere** on IBM Cloud provides maximum flexibility to build IBM-hosted environment using VMware-compatible hardware and the right set of VMware components that fit business needs and expertise.
VMware **vCenter** Server on IBM Cloud is a hosted private cloud that delivers the VMware vSphere stack as a service.
Kubernetes cluster lets securely manage the resources needed to quickly deploy, update, and scale applications.
Container registry can be detached from IBM Cloud Kubernetes Service in order to store and distribute Docker images via a stateless, highly scalable server side application.

Available network services will be described on the next slide.



IBM Cloud network services

More info: https://console.bluemix.net/catalog/?category=network

Content Delivery Network service distributes content where it is needed.

The first time content is requested, it's pulled from the host server to the network and stays there for other users to access it.

Domain name service. IBM Cloud offers domain registration services complete with dedicated support staff, knowledgeable customer service, and reasonable prices, all delivered over a secure network.

Virtual Private Netwrok (VPN). It access is designed to allow users to remotely manage all servers and services associated with their account over our private network.

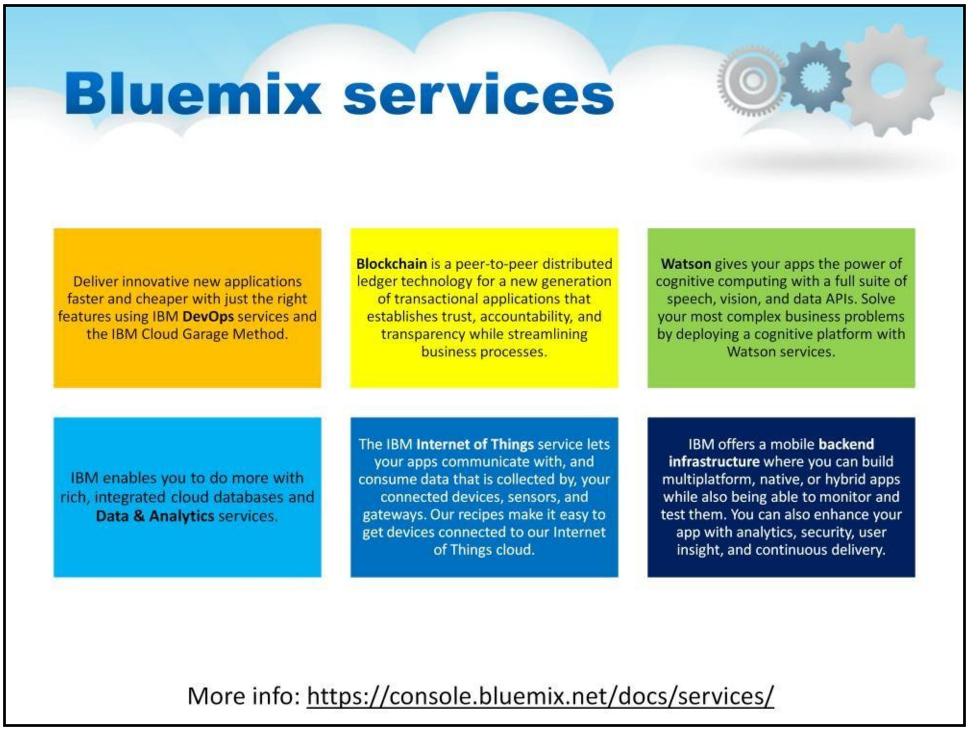
Load balancer service distribute traffic among application servers residing locally within data center.

VPN Spanning. It connects all private network VLANs on an account, allowing devices on separate VLANs to communicate with each other.

Virtual Router Appliance (VRA) provides the latest Vyatta Network Operating System

for x86 bare metal servers.
Its is possible to create virtual routers, firewalls, and VPNs that fit unique application requirements.

More information can be obtained by the reference shown here.



Bluemix services

Deliver innovative new applications faster and cheaper with just the right features using IBM **DevOps** services and the IBM Cloud Garage Method.

Blockchain is a peer-to-peer distributed ledger technology for a new generation of transactional applications that establishes trust, accountability, and transparency while streamlining business processes.

Watson gives your apps the power of cognitive computing with a full suite of speech, vision, and data APIs. Solve your most complex business problems by deploying a cognitive platform with Watson services.

IBM enables you to do more with rich, integrated cloud databases and **Data & Analytics** services.

The IBM **Internet of Things** service lets your apps communicate with, and consume data that is collected by, your connected devices, sensors, and gateways. Our recipes make it easy to get devices connected to our Internet of Things cloud.

IBM offers a mobile **backend infrastructure** where you can build multiplatform, native, or hybrid apps while also being able to monitor and test them. You can also enhance your app with analytics, security, user insight, and continuous delivery.

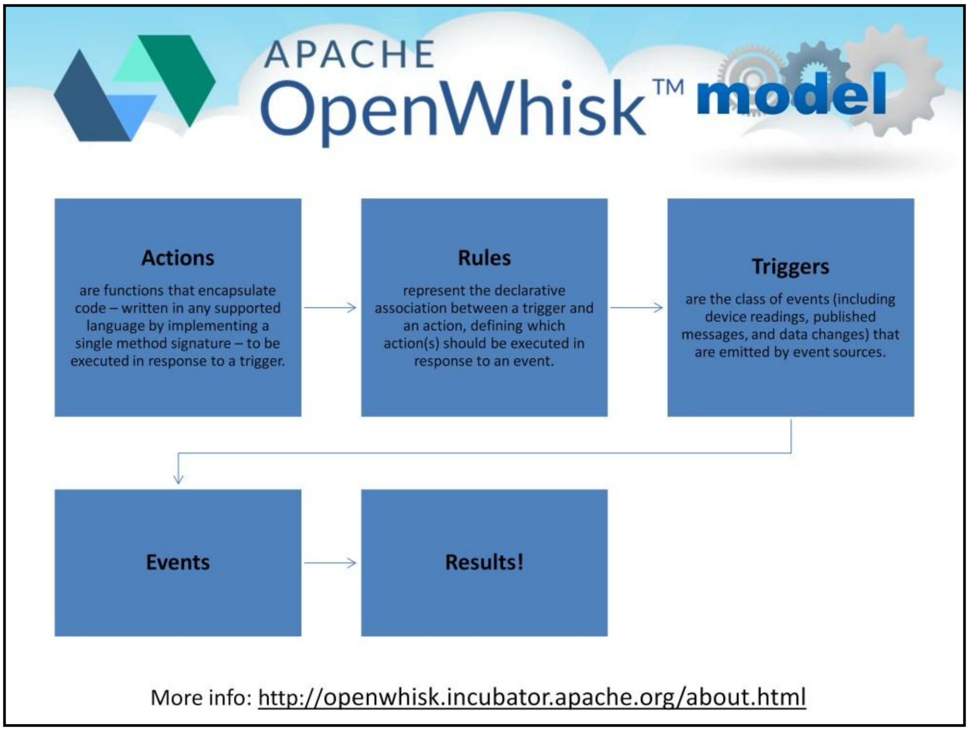
More info: https://console.bluemix.net/docs/services/

The IBM Cloud dashboard provides access to the IBM Cloud services available from IBM and third-party providers.

These include Watson, Internet of Things, Data & Analytics, Mobile, and DevOps services.

On this slide you can see a short descriptions for each service.

More information can be obtained by the reference shown here.



APACHE OpenWhisk™ model

Actions

are functions that encapsulate code – written in any supported language by implementing a single method signature – to be executed in response to a trigger.

Rules

represent the declarative association between a trigger and an action, defining which action(s) should be executed in response to an event.

Triggers

are the class of events (including device readings, published messages, and data changes) that are emitted by event sources.

Events

Results!

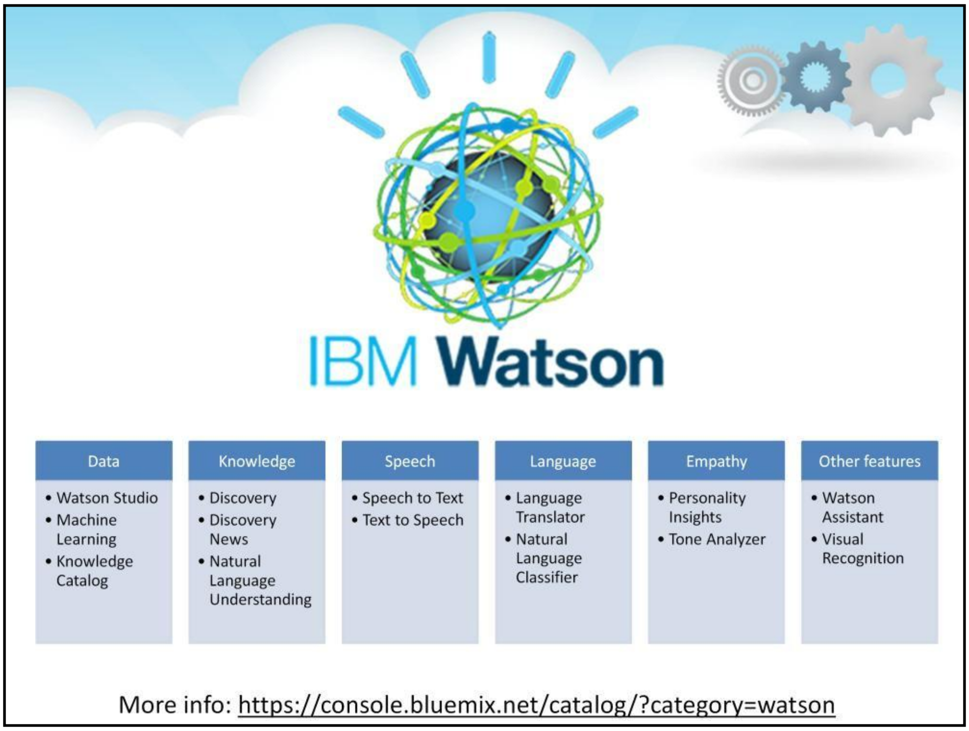
More info: http://openwhisk.incubator.apache.org/about.html

IBM Bluemix includes IBM's Function as a Service (FaaS) system, or Serverless computing offering, that is built using open source from the Apache OpenWhisk incubator project largely credited to IBM for seeding.

OpenWhisk is an open source, distributed serverless computing platform able to execute application logic (**Actions**) in response to events (**Triggers**) from external sources (**Feeds**) or HTTP requests governed by conditional logic (**Rules**).

It provides a programming environment supported by a REST API-based Command Line Interface (CLI) along with tooling to support packaging and catalog services.

More information can be obtained by the reference shown here.



IBM Watson

Data	Knowledge	Speech	Language	Empathy	Other features
• Watson Studio • Machine Learning • Knowledge Catalog	• Discovery • Discovery News • Natural Language Understanding	• Speech to Text • Text to Speech	• Language Translator • Natural Language Classifier	• Personality Insights • Tone Analyzer	• Watson Assistant • Visual Recognition

More info: https://console.bluemix.net/catalog/?category=watson

Watson is an IBM Cloud service for cognitive computing with the following components:

IBM Watson Studio accelerates the machine and deep learning workflows required to infuse AI into business to drive innovation.

IBM Watson Knowledge Catalog powers intelligent, self-service discovery of data, models and more, activating them for artificial intelligence, machine learning and deep learning.

Watson Discovery feature does a cognitive search and content analytics to identify patterns, trends and actionable insights to drive better decision-making.

Discovery News means discovering trends and patterns in sentiment with aggregate analysis in order to see new perspectives on how news unfolds across the globe.

Natural language understanding. It allows to analyze text to extract meta-data from content such as concepts, entities, keywords, categories, emotion, relations, semantic roles.

Speech to Text service converts the human voice into the written word.

Text to Speech service processes text and natural language to generate synthesized audio output complete with appropriate cadence and intonation.

Watson Language Translator allows to dynamically translate text and instantly publish content in multiple languages.

Natural Language Classifier service applies cognitive computing techniques to return the best matching classes for a sentence or phrase.

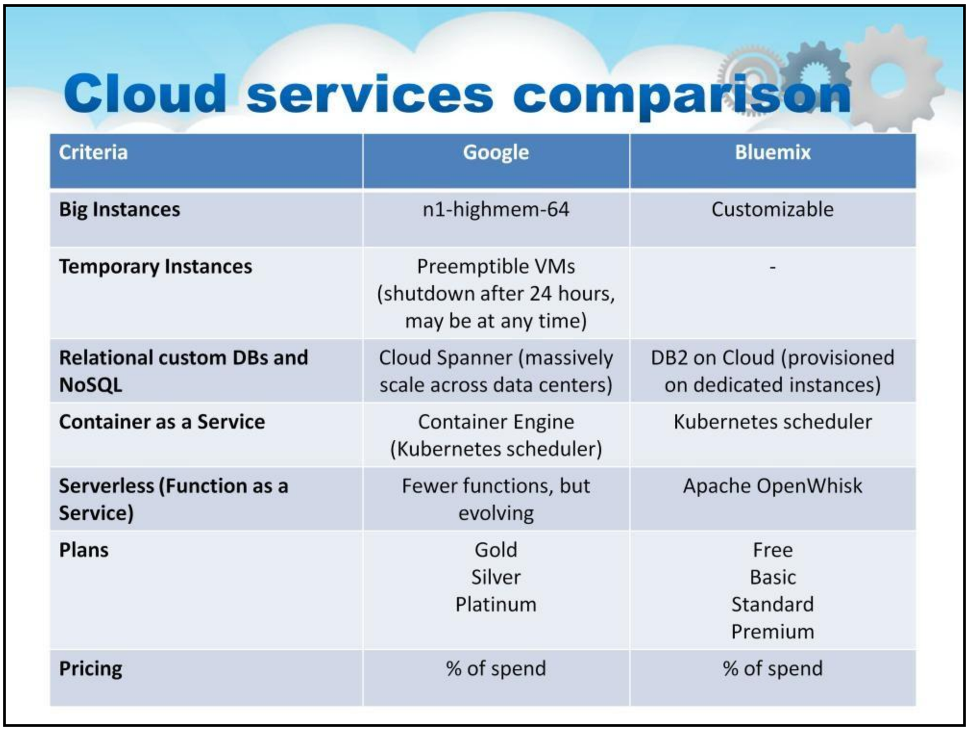
Personality Insights derives insights from transactional and social media data to identify psychological traits which determine purchase decisions, intent and behavioral traits.

Tone Analyzer leverages cognitive linguistic analysis to identify a variety of tones at both the sentence and document level.

Watson Assistant provides a natural language interface to automate interactions with users.

Visual Recognition allows to analyze images for scenes, objects, faces, and other content.

More information can be obtained by the reference shown here.



Cloud services comparison

Criteria	Google	Bluemix
Big Instances	n1-highmem-64	Customizable
Temporary Instances	Preemptible VMs (shutdown after 24 hours, may be at any time)	-
Relational custom DBs and NoSQL	Cloud Spanner (massively scale across data centers)	DB2 on Cloud (provisioned on dedicated instances)
Container as a Service	Container Engine (Kubernetes scheduler)	Kubernetes scheduler
Serverless (Function as a Service)	Fewer functions, but evolving	Apache OpenWhisk
Plans	Gold Silver Platinum	Free Basic Standard Premium
Pricing	% of spend	% of spend

On the last slide you can see differences of Google and IBM Clouds in some key points.

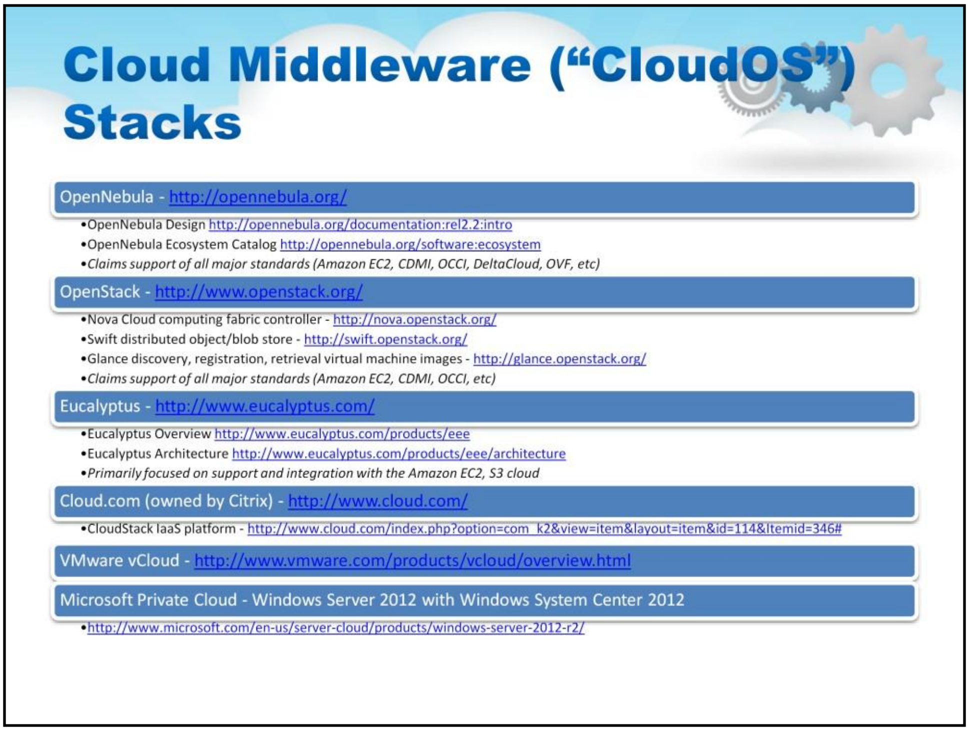
Cloud Service Providers

–

Open Source Solutions

Open Source Solutions

Let’s consider some Open Source solutions…



Cloud Middleware ("CloudOS") Stacks

OpenNebula - http://opennebula.org/

- OpenNebula Design http://opennebula.org/documentation:rel2.2:intro
- OpenNebula Ecosystem Catalog http://opennebula.org/software:ecosystem
- *Claims support of all major standards (Amazon EC2, CDMI, OCCI, DeltaCloud, OVF, etc)*

OpenStack - http://www.openstack.org/

- Nova Cloud computing fabric controller - http://nova.openstack.org/
- Swift distributed object/blob store - http://swift.openstack.org/
- Glance discovery, registration, retrieval virtual machine images - http://glance.openstack.org/
- *Claims support of all major standards (Amazon EC2, CDMI, OCCI, etc)*

Eucalyptus - http://www.eucalyptus.com/

- Eucalyptus Overview http://www.eucalyptus.com/products/eee
- Eucalyptus Architecture http://www.eucalyptus.com/products/eee/architecture
- *Primarily focused on support and integration with the Amazon EC2, S3 cloud*

Cloud.com (owned by Citrix) - http://www.cloud.com/

- CloudStack IaaS platform - http://www.cloud.com/index.php?option=com_k2&view=item&layout=item&id=114&Itemid=346#

VMware vCloud - http://www.vmware.com/products/vcloud/overview.html

Microsoft Private Cloud - Windows Server 2012 with Windows System Center 2012

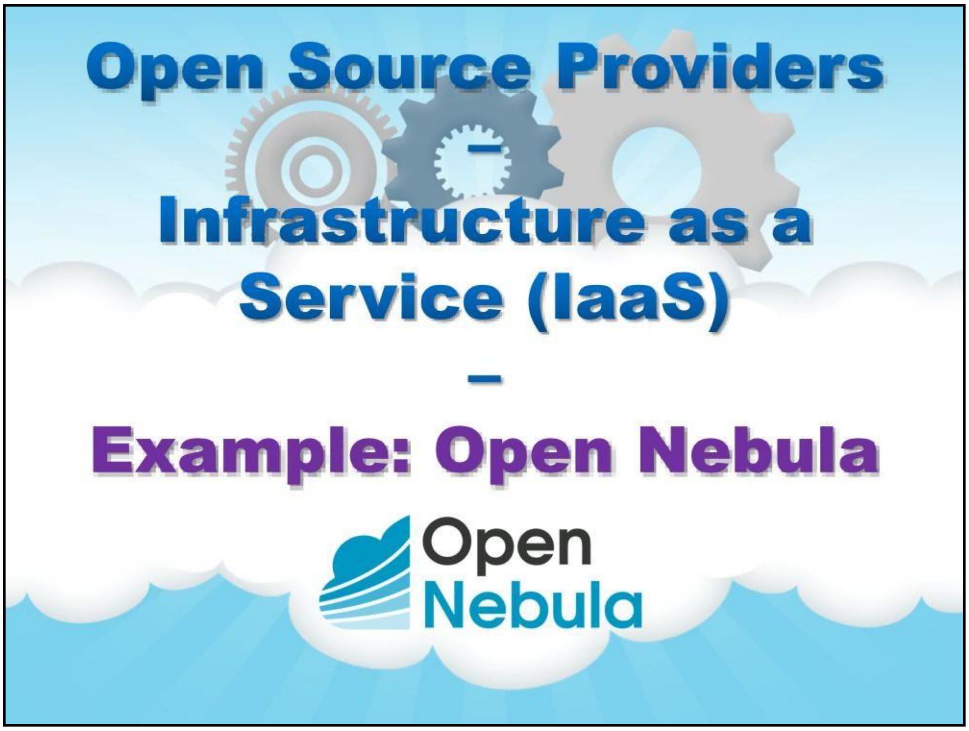
- http://www.microsoft.com/en-us/server-cloud/products/windows-server-2012-r2/

There are several different Cloud Middleware stacks.

These "stacks" are often called a "CloudOS", even though they are actually Middleware.

Each of the servers in a Cloud is actually running an Operating System such as Linux or Windows, and includes a virtualization component such as Xen, KVM, ESX, or Hyper-V.

In this part of the lecture we start from OpenStack…



Open Source Providers

–

Infrastructure as a Service (IaaS)

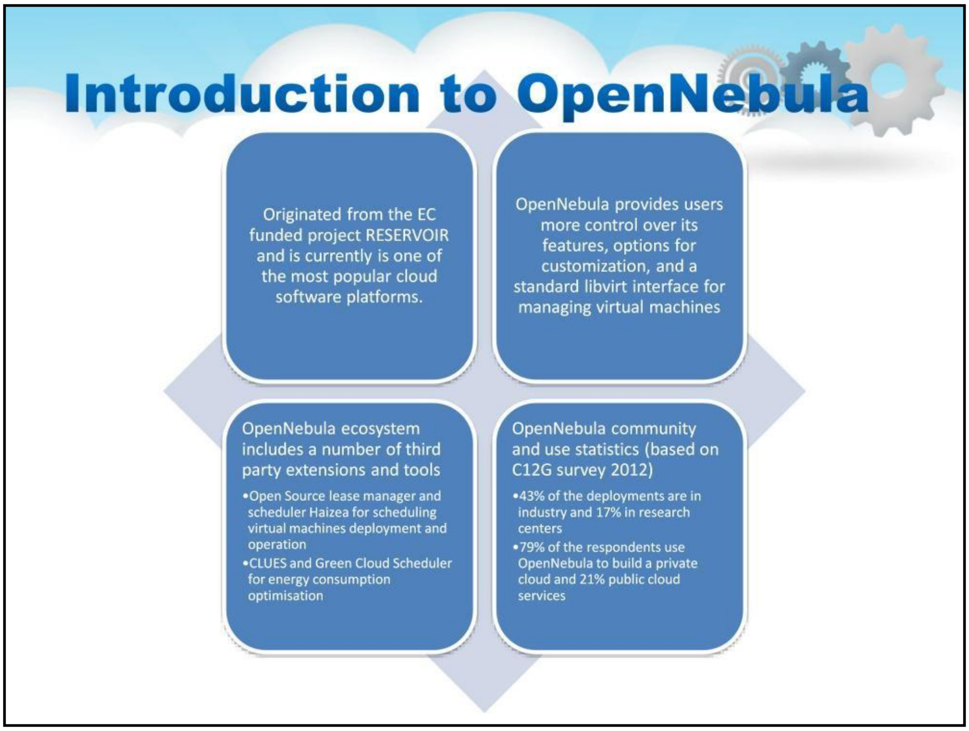
–

Example: Open Nebula

Open
Nebula

Example: OpenNebula

The first example is dedicated to OpenNebula that represents an example of Infrastructure as a Service (IaaS).



Introduction to OpenNebula

Originated from the EC funded project RESERVOIR and is currently is one of the most popular cloud software platforms.

OpenNebula provides users more control over its features, options for customization, and a standard libvirt interface for managing virtual machines

OpenNebula ecosystem includes a number of third party extensions and tools

- Open Source lease manager and scheduler Haizea for scheduling virtual machines deployment and operation
- CLUES and Green Cloud Scheduler for energy consumption optimisation

OpenNebula community and use statistics (based on C12G survey 2012)

- 43% of the deployments are in industry and 17% in research centers
- 79% of the respondents use OpenNebula to build a private cloud and 21% public cloud services

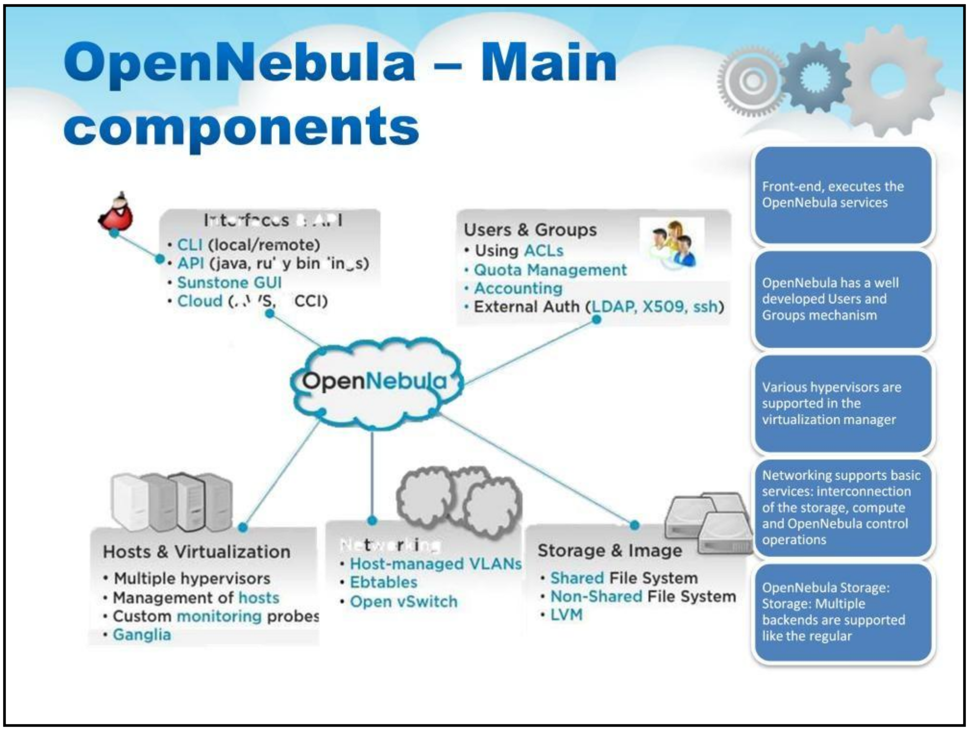
OpenNebula was first established as a research project back in 2005 from the EC funded project RESERVOIR. Since its first public release of software in March 2008, it has evolved through open-source releases and now operates as an open source project.

> The first version 1.0 was released in 2008, and since then, the project is being sponsored by the Distributed Systems Architecture Research Group (http://dsa-research.org) at the Universidad Computense de Madrid.

The OpenNebula technology has matured thanks to an active and engaged community of users and developers. Its software was downloaded several thousands times per month from the site. Besides an exponential growth in its number of users, different projects, research groups and companies have built new virtualization and cloud components to complement and to enhance the functionality provided by this widely used open-source toolkit for cloud computing.

In March 2010, the main authors of OpenNebula founded C12G Labs to provide the value-added professional services that many enterprise IT shops require for internal adoption and to allow the OpenNebula project to not be tied exclusively to public financing, contributing to its long-term sustainability. In September 2013, OpenNebula organized its first ever community conference, which included presentations by leading organizations worldwide.

OpenNebula provides users more control over its features, options for customization, and a standard libvirt interface for managing virtual machines. Uses NFS for storing disk images. OpenNebula has a scheduler which can do the basic tasks of scheduling virtual machines.



OpenNebula – Main components

Interfaces & API
- CLI (local/remote)
- API (java, ruby bindings)
- Sunstone GUI
- Cloud (AWS, OCCI)

Users & Groups
- Using ACLs
- Quota Management
- Accounting
- External Auth (LDAP, X509, ssh)

Hosts & Virtualization
- Multiple hypervisors
- Management of hosts
- Custom monitoring probes
- Ganglia

Networking
- Host-managed VLANs
- Ebtables
- Open vSwitch

Storage & Image
- Shared File System
- Non-Shared File System
- LVM

Front-end, executes the OpenNebula services

OpenNebula has a well developed Users and Groups mechanism

Various hypervisors are supported in the virtualization manager

Networking supports basic services: interconnection of the storage, compute and OpenNebula control operations

OpenNebula Storage: Storage: Multiple backends are supported like the regular

OpenNebula features advanced multi-tenancy with powerful users and groups management, fine-grained resource allocation, and resource quota management to track and limit computing, storage and networking utilization.

User Management: OpenNebula can validate users using its own internal user database based on passwords, or external mechanisms, like ssh, x509, ldap or Active Directory **OpenNebula Virtualization:** Various hypervisors are supported in the virtualization manager, with the ability to control the complete lifecycle of Virtual Machines and multiple hypervisors in the same cloud infrastructure.

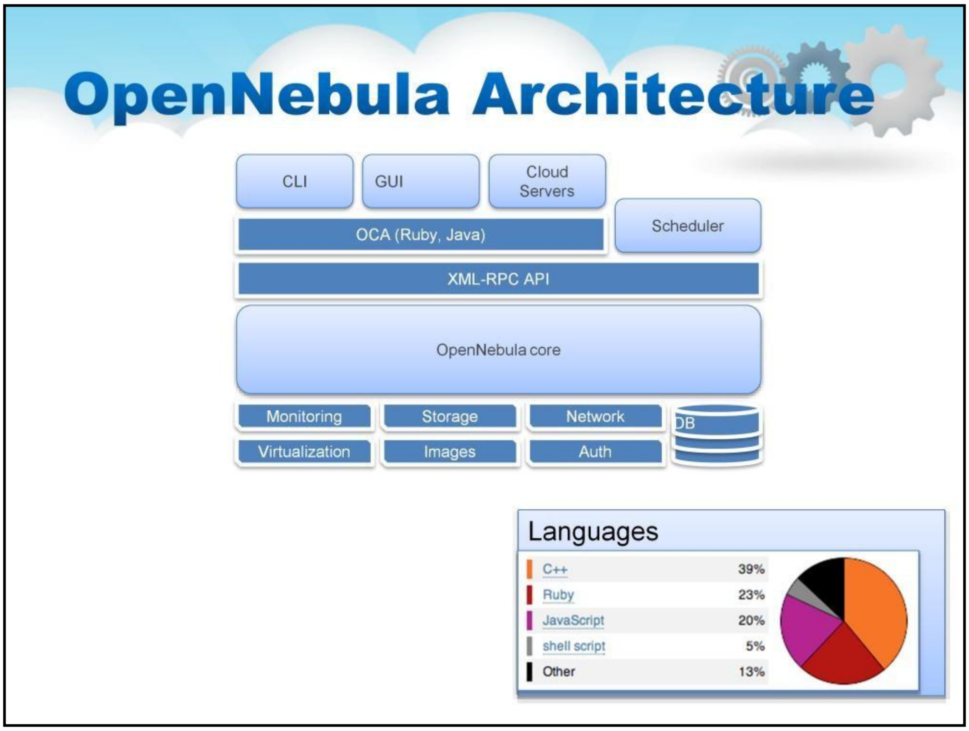
Virtualization: Several hypervisor technologies are fully supported, like Xen, KVM and VMware. **Monitoring:** OpenNebula provides its own customized monitoring system and also integrates with data center monitoring tools like Ganglia.

OpenNebula Hosts: The host manager provides complete functionality for the management of the physical hosts in the cloud.

OpenNebula has strong Networking. Dynamic creation of Clusters as pools of hosts that share datastores and virtual networks for load balancing, high availability, and high performance computing.

OpenNebula Networking: An easily adaptable and customizable network subsystem is present in OpenNebula in order to better integrate with the specific network requirements of existing data centers and to allow full isolation between virtual machines that composes a virtualised service.

OpenNebula Storage: Multiple backends are supported like the regular (shared or not) filesystem datastore supporting popular distributed file systems like NFS, Lustre, GlusterFS,; the VMware datastore specialized for the VMware hypervisor, iSCSI/LVM datastore to store disk images in a block device form; and Ceph for distributed block device.



OpenNebula Architecture

CLI

GUI

Cloud Servers

OCA (Ruby, Java)

Scheduler

XML-RPC API

OpenNebula core

Monitoring

Storage

Network

DB

Virtualization

Images

Auth

Languages

C++	39%
Ruby	23%
JavaScript	20%
shell script	5%
Other	13%

This slide contains the general view on the OpenNebula architecture with the following main components:

1.CLI - command-line interface

2.GUI – graphical user interface

3.CloudServer

4. OCA - OpenNebula Cloud API

5. Sheduler

6.XML_RPC API

7. OpenNebula core

8. Monitoring services

9. Storage services

10. Network services

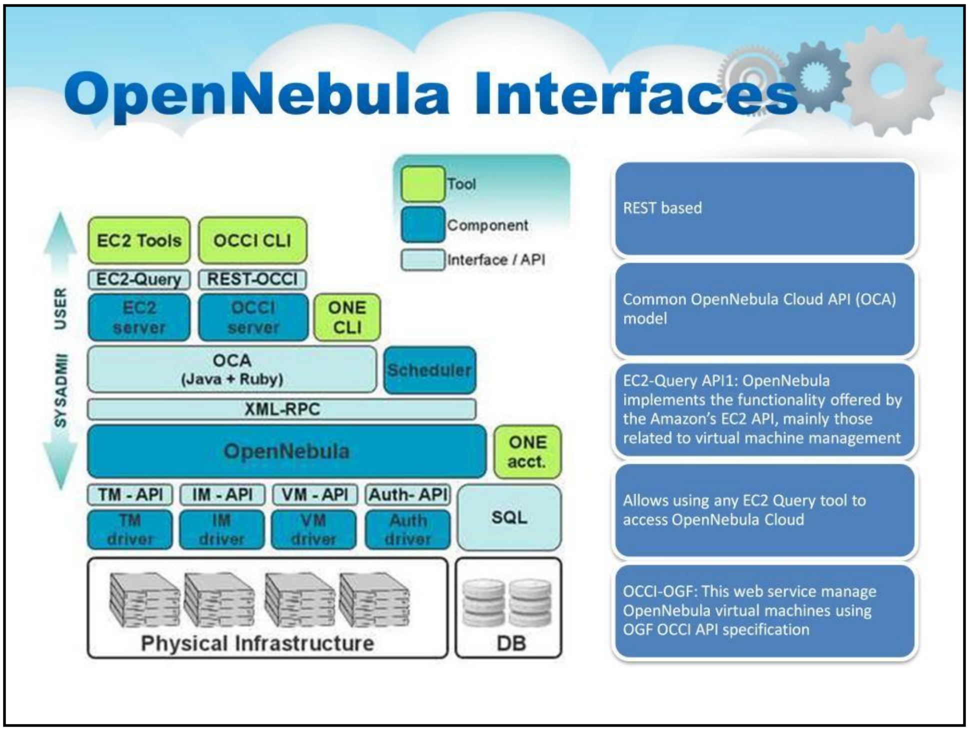
11. Virtualization services

12. Images

13.Authentication services

14.DB – database services

Here the supported languages are also shown with the percentage of their users worldwide.



OpenNebula Interfaces

Tool
Component
Interface / API

USER
SYSADMIN

EC2 Tools
OCCI CLI
EC2-Query
REST-OCCI
EC2 server
OCCI server
ONE CLI
OCA (Java + Ruby)
Scheduler
XML-RPC
OpenNebula
ONE acct.
TM - API
IM - API
VM - API
Auth- API
TM driver
IM driver
VM driver
Auth driver
SQL
Physical Infrastructure
DB

REST based

Common OpenNebula Cloud API (OCA) model

EC2-Query API1: OpenNebula implements the functionality offered by the Amazon's EC2 API, mainly those related to virtual machine management

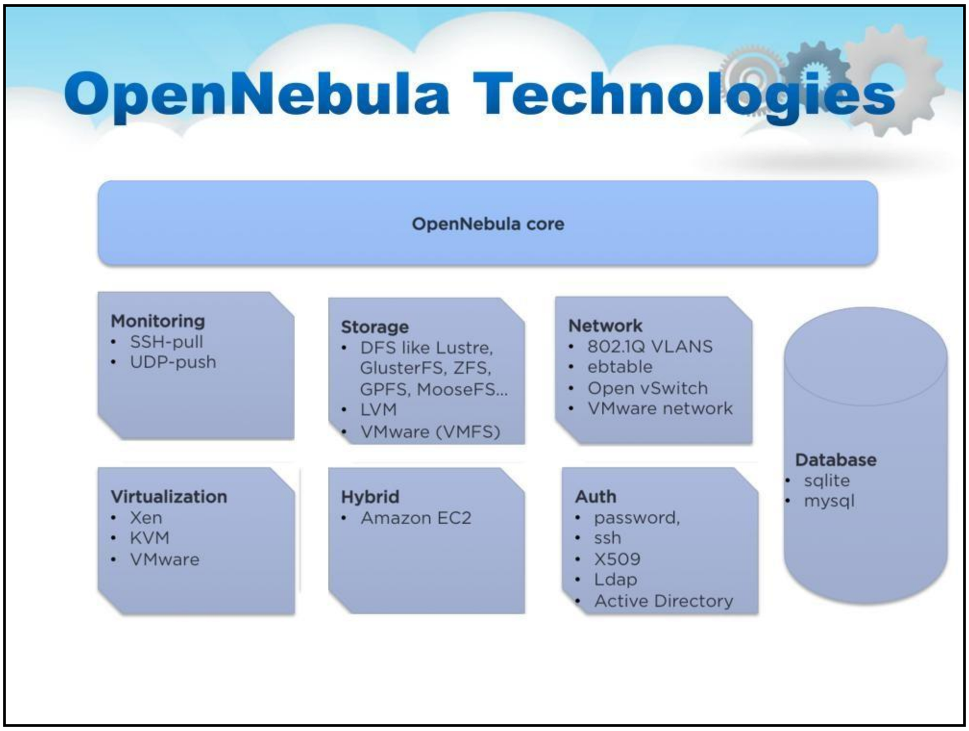
Allows using any EC2 Query tool to access OpenNebula Cloud

OCCI-OGF: This web service manage OpenNebula virtual machines using OGF OCCI API specification

Cloud interfaces enable you to manage virtual machines, networks and images through a simple and easy-to-use REST API. The Cloud interfaces hide most of the complexity of a Cloud and are specially suited for end-users. OpenNebula implements two different interfaces, namely:

❑ EC2-Query API. OpenNebula implements the functionality offered by the Amazon's EC2 API, mainly those related to virtual machine management. In this way, you can use any EC2 Query tool to access your OpenNebula Cloud.

❑ OCCI-OGF. This web service enables you to launch and manage virtual machines in your OpenNebula installation using the latest draft of the OGF OCCI API specification.



OpenNebula Technologies

OpenNebula core

Monitoring
- SSH-pull
- UDP-push

Storage
- DFS like Lustre, GlusterFS, ZFS, GPFS, MooseFS...
- LVM
- VMware (VMFS)

Network
- 802.1Q VLANS
- ebtable
- Open vSwitch
- VMware network

Database
- sqlite
- mysql

Virtualization
- Xen
- KVM
- VMware

Hybrid
- Amazon EC2

Auth
- password,
- ssh
- X509
- Ldap
- Active Directory

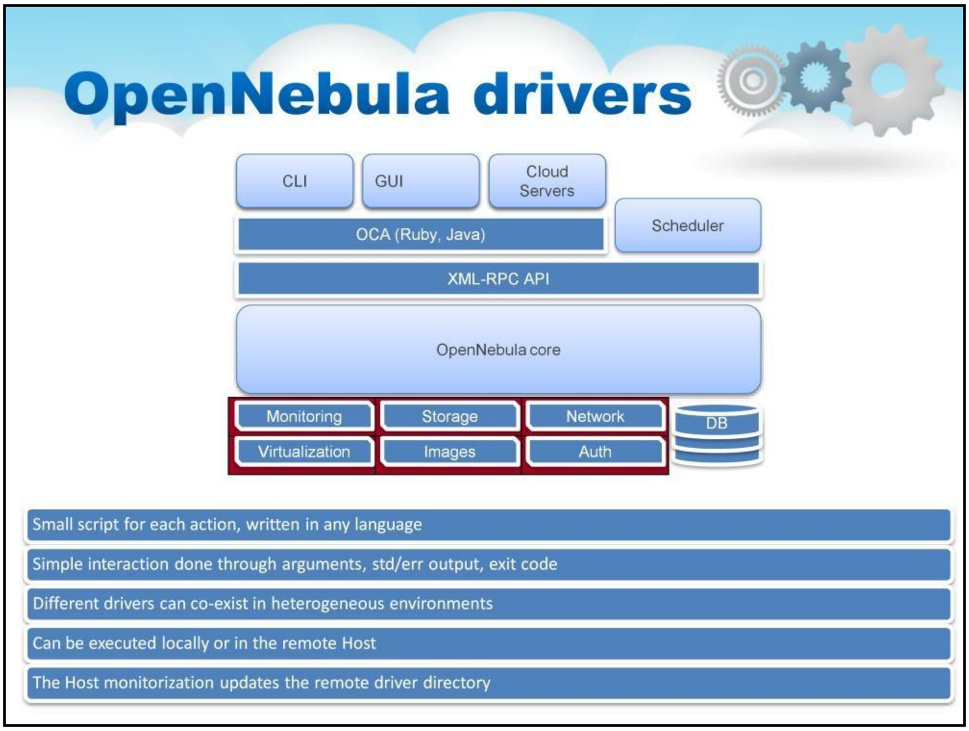
This slide describes OpenNebula core technologies behind its components.

Advanced Control and Monitoring of Virtual Infrastructure

Image Repository Subsystem with catalog and complete functionality for VM image management. Template Repository Subsystem with catalog and complete functionality for VM template management. Full control of VM instance life-cycle and complete functionality for VM instance management. Advanced functionality for VM dynamic management like system and disk snapshotting, capacity resizing, or NIC hotplugging. Programmable VM operations, so allowing users to schedule actions
Storage Subsystem with support for multiple data stores to balance I/O operations between storage servers, or to define different SLA policies (e.g. backup) and performance features for different VM types or users. It supports any backend configuration with different datastore types.

Flexible Network Subsystem.

Hypervisor agnostic Virtualization with broad hypervisor support (Xen, KVM and VMware), centralized management of environments with multiple hypervisors, and support for multiple hypervisors within the same physical box.
Hybrid Cloud Computing and Cloudbursting. Extension of the local private infrastructure with resources from remote clouds.
Authorization framework. Secure and efficient Users and Groups Subsystem for authentication and authorization of requests with complete functionality for user management.
Advanced Multi-tenancy with Group Management. Administrators can groups users into organizations that can represent different projects, division.
Database. Highly scalable database back-end with support for MySQL and SQLite



OpenNebula drivers

CLI

GUI

Cloud Servers

OCA (Ruby, Java)

Scheduler

XML-RPC API

OpenNebula core

Monitoring

Storage

Network

DB

Virtualization

Images

Auth

- Small script for each action, written in any language
- Simple interaction done through arguments, std/err output, exit code
- Different drivers can co-exist in heterogeneous environments
- Can be executed locally or in the remote Host
- The Host monitorization updates the remote driver directory

This slide describes OpenNebula Drivers Interfaces.

The interactions between OpenNebula and the Cloud infrastructure are performed by specific drivers each one addressing a particular area:

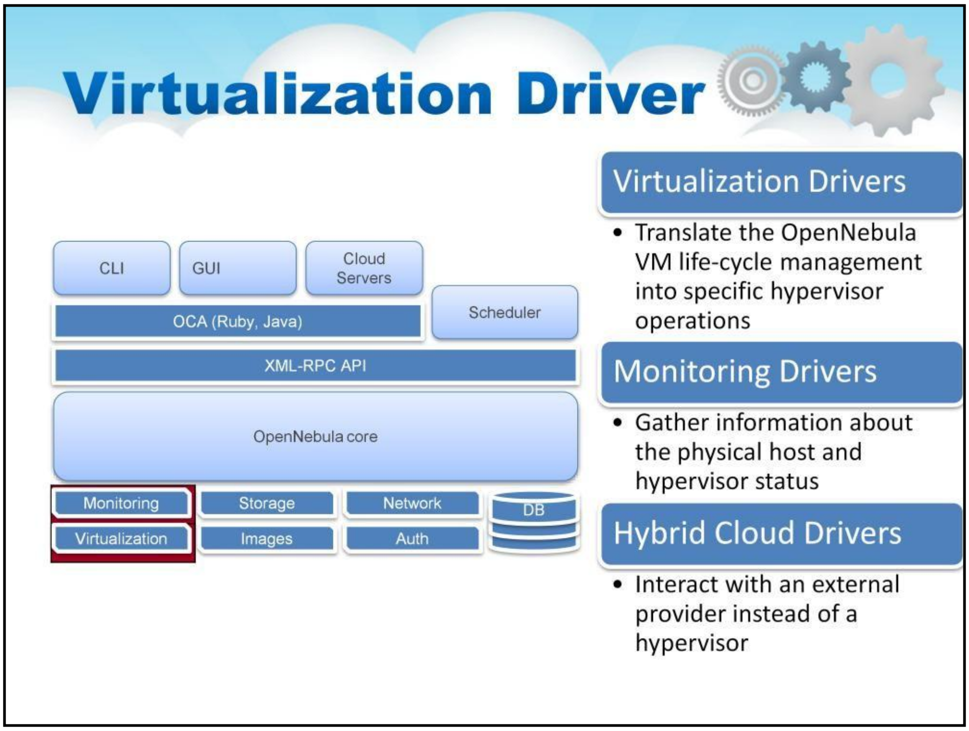
Storage. The OpenNebula core issue abstract storage operations (e.g. clone or delete) that are implemented by specific programs that can be replaced or modified to interface special storage backends and file-systems.

Virtualization. The interaction with the hypervisors are also implemented with custom programs to boot, stop or migrate a virtual machine. This allows you to specialize each VM operation so to perform custom operations.

Monitoring. Monitoring information is also gathered by external probes. You can add additional probes to include custom monitoring metrics that can be later used to allocate virtual machines or for accounting purposes

Authorization. OpenNebula can be also configured to use an external program to authorize and authenticate user requests. In this way, you can implement any access policy to Cloud resources.

DataBase. OpenNebula saves its state and lots of accounting information in a persistent data-base. OpenNebula can use MySQL or SQLite database that can be easily interfaced with any of DB tool.



Virtualization Driver

Virtualization Drivers

- Translate the OpenNebula VM life-cycle management into specific hypervisor operations

Monitoring Drivers

- Gather information about the physical host and hypervisor status

Hybrid Cloud Drivers

- Interact with an external provider instead of a hypervisor

In OpenNebula the hptional integration with datacenter monitoring tools like Ganglia is possible.

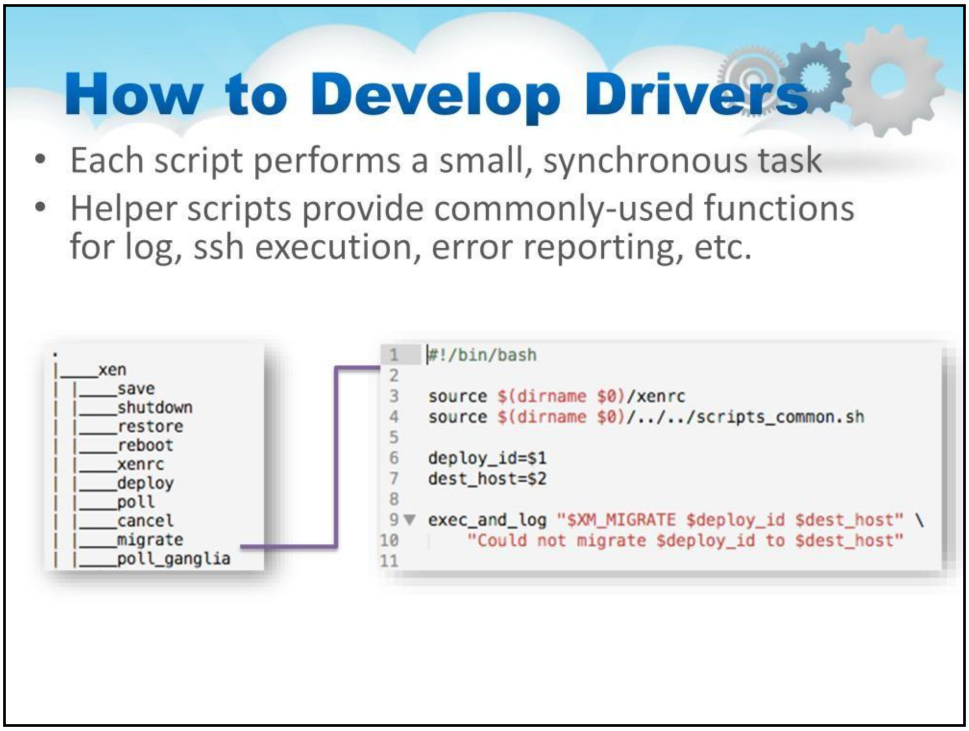
Note: Ganglia is a scalable, distributed monitoring tool for high-performance computing systems, clusters and networks.

If you already have ganglia deployed in your cluster you can use the ganglia drivers provided by OpenNebula to get information about hosts and virtual machines from it. These drivers should make the monitoring more performant in a big installation as they don't use ssh connections to the nodes to get the information. On the other side they require more work on the system administrator as ganglia should be properly configured and cron jobs must be installed on the nodes to provide virtual machine information to ganglia system.

The component that deals with the hypervisor to create, manage and get information about virtual machine objects is called Virtual Machine Manager (VMM for short).

The following drivers for the main hypervisors are already written:

❑ KVM Driver 4.0 - KVM (Kernel-based Virtual Machine) is a complete virtualization technique for Linux. It offers full virtualization, where each Virtual Machine interacts with its own virtualized hardware.
❑ Xen Driver 4.0 - The XEN hypervisor offers a powerful, efficient and secure feature set for virtualization of x86, IA64, PowerPC and other CPU architectures. It delivers both paravirtualization and full virtualization.
❑ VMware Drivers 4.0 - The VMware Drivers enable the management of an OpenNebula cloud based on VMware ESX and/or VMware Server hypervisors. They use feature a simple configuration process that will leverage the stability, performance and feature set of any existing VMware based OpenNebula cloud.



How to Develop Drivers

- Each script performs a small, synchronous task
- Helper scripts provide commonly-used functions for log, ssh execution, error reporting, etc.

```
.
|____xen
| |____save
| |____shutdown
| |____restore
| |____reboot
| |____xenrc
| |____deploy
| |____poll
| |____cancel
| |____migrate
| |____poll_ganglia
```

```
#!/bin/bash

source $(dirname $0)/xenrc
source $(dirname $0)/../../scripts_common.sh

deploy_id=$1
dest_host=$2

exec_and_log "$XM_MIGRATE $deploy_id $dest_host" \
    "Could not migrate $deploy_id to $dest_host"

```

This slide contains description of how the Virtualization drivers are written.

This component has the following two parts:
• the first one resides in the core and holds most of the general functionality common to all the drivers (and some specific),
• the second is the driver that is the one able to translate basic VMM actions to the hypervisor.

The possible actions are:
DEPLOY - Tells the hypervisor to create a new VM
SHUTDOWN - Sends shutdown signal to a VM
CANCEL - Destroys a VM
SAVE - Saves the state of a VM (suspend)
RESTORE - Restores a VM to a previous saved state
MIGRATE - Performs live migration of a VM
POLL - Gets information about a VM
MIGRATE - Performs live migration of a VM

Every action should have an executable program (mainly scripts) located in the remote directory that performs the desired action.

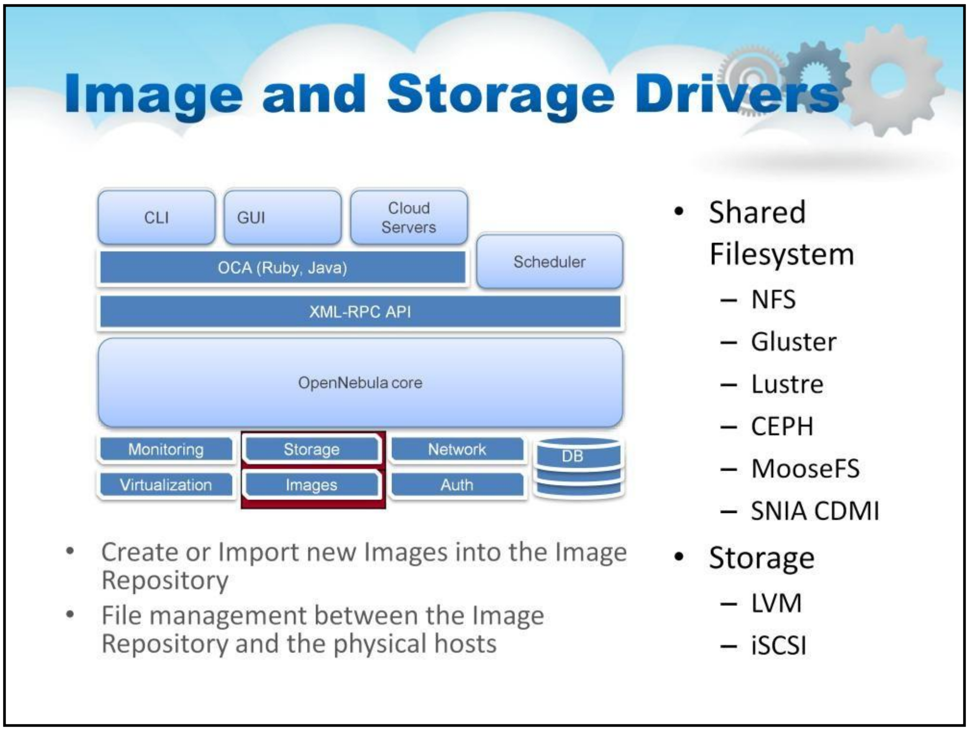


Image and Storage Drivers

- Create or Import new Images into the Image Repository
- File management between the Image Repository and the physical hosts

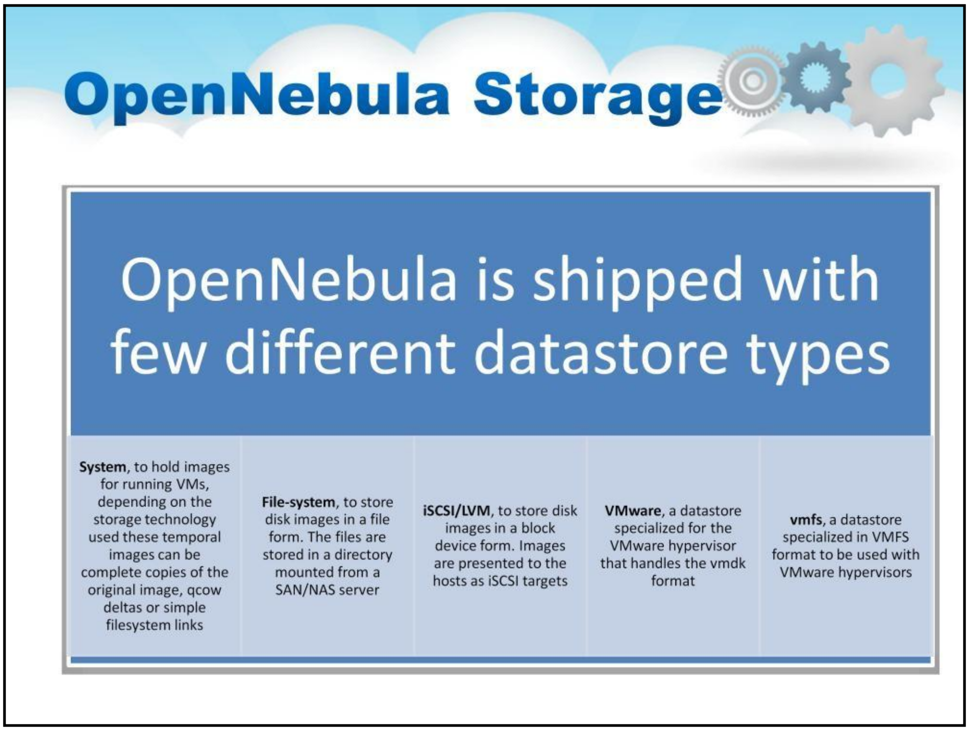
- Shared Filesystem
  - NFS
  - Gluster
  - Lustre
  - CEPH
  - MooseFS
  - SNIA CDMI
- Storage
  - LVM
  - iSCSI

The Storage subsystem is highly modular.

These drivers are separated into two logical sets:

DS: Datastore drivers. They serve the purpose of managing images: register, delete, and create empty datablocks.
TM: Transfer Manager drivers. They manage images associated to instantiated VMs.

The Disk images registered in a datastore are transferred to the hosts by the transfer manager (TM) drivers. These drivers are specialized pieces of software that perform low-level storage operations (e.g. setting up iSCSI targets or file movements).



OpenNebula Storage

OpenNebula is shipped with few different datastore types

System, to hold images for running VMs, depending on the storage technology used these temporal images can be complete copies of the original image, qcow deltas or simple filesystem links

File-system, to store disk images in a file form. The files are stored in a directory mounted from a SAN/NAS server

iSCSI/LVM, to store disk images in a block device form. Images are presented to the hosts as iSCSI targets

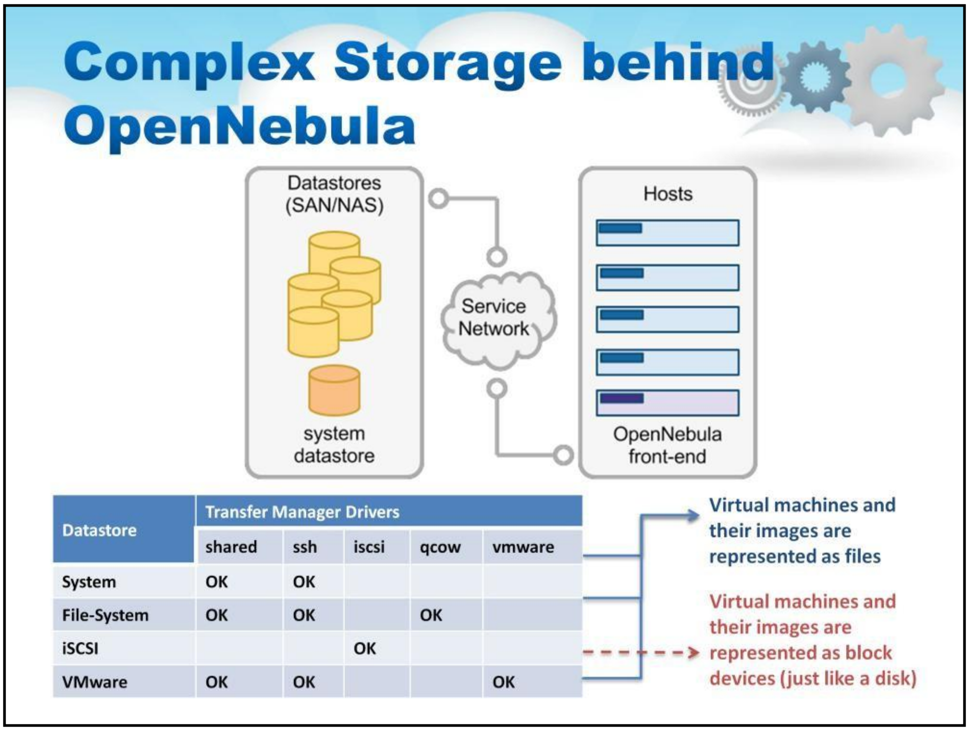
VMware, a datastore specialized for the VMware hypervisor that handles the vmdk format

vmfs, a datastore specialized in VMFS format to be used with VMware hypervisors

The transfer mechanism is defined for each datastore. In this way a single host can simultaneously access multiple datastores that uses different transfer drivers. Note that the hosts must be configured to properly access each data-store type (e.g. mount FS shares).

OpenNebula is shipped with different datastore types:

1. System, to hold images for running VMs, depending on the storage technology used.
2. File-system, to store disk images in a file form.
3. iSCSI/LVM, to store disk images in a block device form.
4. VMware, a datastore specialized for the VMware hypervisor.
5. vmfs, a datastore specialized in VMFS format to be used with VMware hypervisors.
6. Ceph, to store disk images using Ceph block devices.
7. Files, this is a special datastore used to store plain files and not disk images. The plain files can be used as kernels, ramdisks or context files.



Complex Storage behind OpenNebula

Datastores (SAN/NAS)

system datastore

Service Network

Hosts

OpenNebula front-end

Datastore	Transfer Manager Drivers				
	shared	ssh	iscsi	qcow	vmware
System	OK	OK			
File-System	OK	OK		OK	
iSCSI			OK		
VMware	OK	OK			OK

Virtual machines and their images are represented as files

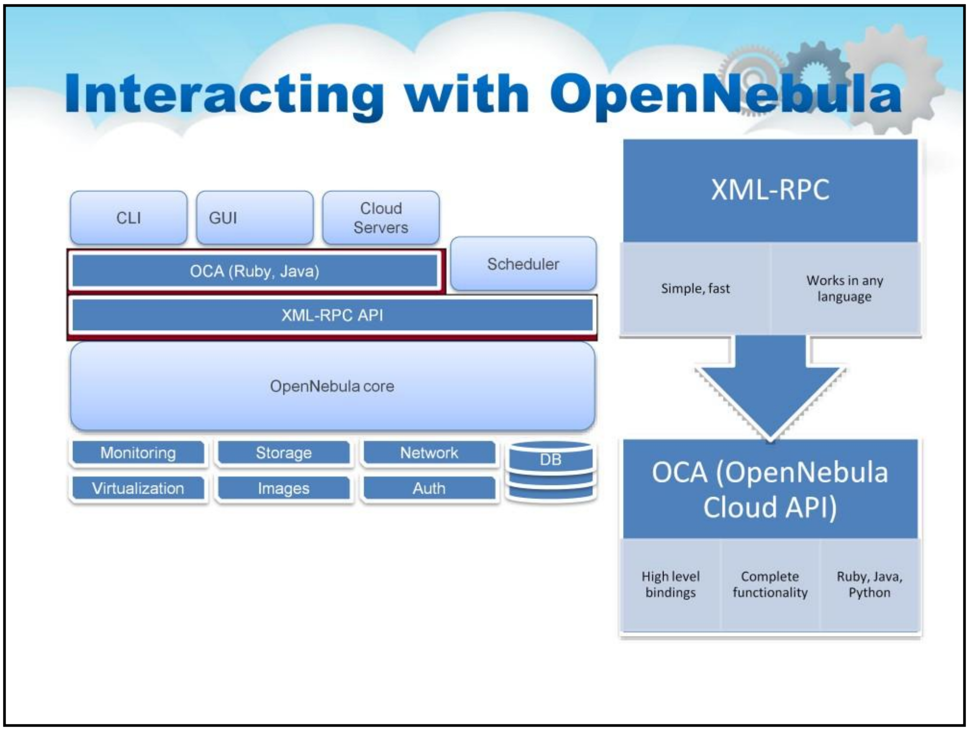
Virtual machines and their images are represented as block devices (just like a disk)

Datastore is any storage medium used to store disk images for VMs, previous versions of OpenNebula refer to this concept as Image Repository.

An OpenNebula installation can have multiple datastores of several types to store disk images. OpenNebula also uses a special datastore, the *system datastore*, to hold images of running VMs. You can take advantage of the multiple datastore features of OpenNebula to better scale the storage for your VMs, in particular:

Balancing I/O operations between storage servers
Different VM types or users can use datastores with different performance features
Different SLA policies (e.g. backup) can be applied to different VM types or users
Easily add new storage to the cloud

There are some limitations and features depending on the transfer mechanism you choose for your system and image datastores (check each datastore guide for more information). The table summarizes the valid combinations of Datastore and transfer drivers.

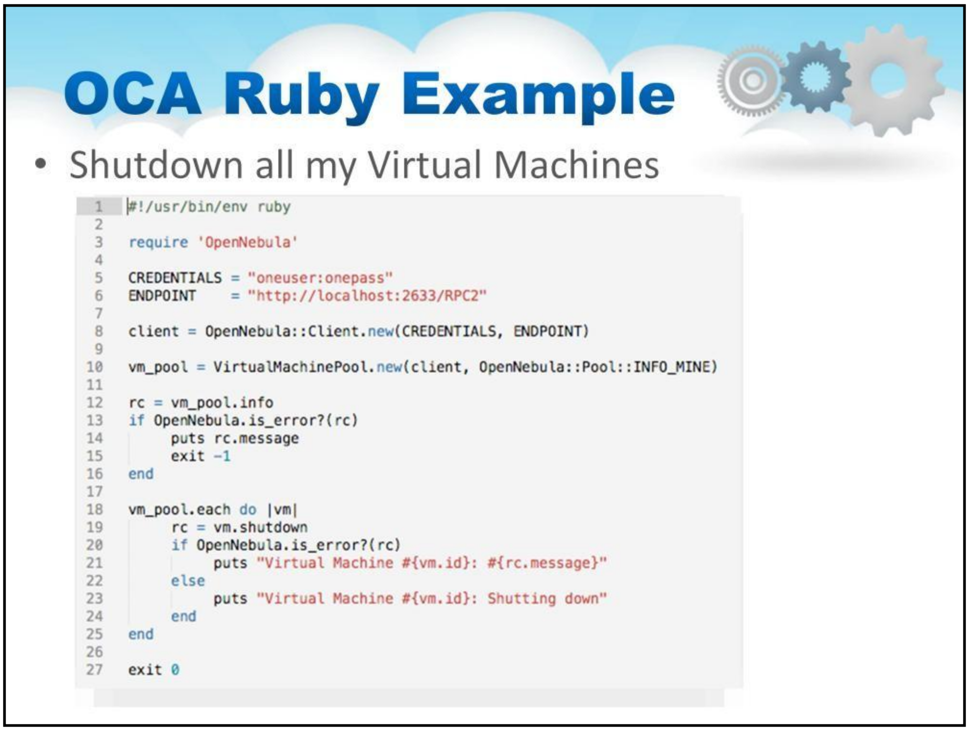


Interacting with OpenNebula

This slide describes OpenNebula Interfaces

XML-RPC interface
It is the primary interface for OpenNebula, and it exposes all the functionality to interface the OpenNebula daemon. Through the XML-RPC interface you can control and manage any OpenNebula resource, including virtual machines, networks, images, users, hosts and clusters.
XML-RPC interface should be used if you are developing specialized libraries for Cloud applications or you need a low-level interface with the OpenNebula core.

OpenNebula Cloud API (OCA)
It provides a simplified and convenient way to interface the OpenNebula core. The OCA interfaces exposes the same functionality as that of the XML-RPC interface. OpenNebula includes two language bindings for OCA: Ruby and JAVA.
OCA interface should be used if you are developing advanced IaaS tools that need full access to the OpenNebula functionality.



OCA Ruby Example

- Shutdown all my Virtual Machines

```
#!/usr/bin/env ruby

require 'OpenNebula'

CREDENTIALS = "oneuser:onepass"
ENDPOINT    = "http://localhost:2633/RPC2"

client = OpenNebula::Client.new(CREDENTIALS, ENDPOINT)

vm_pool = VirtualMachinePool.new(client, OpenNebula::Pool::INFO_MINE)

rc = vm_pool.info
if OpenNebula.is_error?(rc)
    puts rc.message
    exit -1
end

vm_pool.each do |vm|
    rc = vm.shutdown
    if OpenNebula.is_error?(rc)
        puts "Virtual Machine #{vm.id}: #{rc.message}"
    else
        puts "Virtual Machine #{vm.id}: Shutting down"
    end
end

exit 0
```

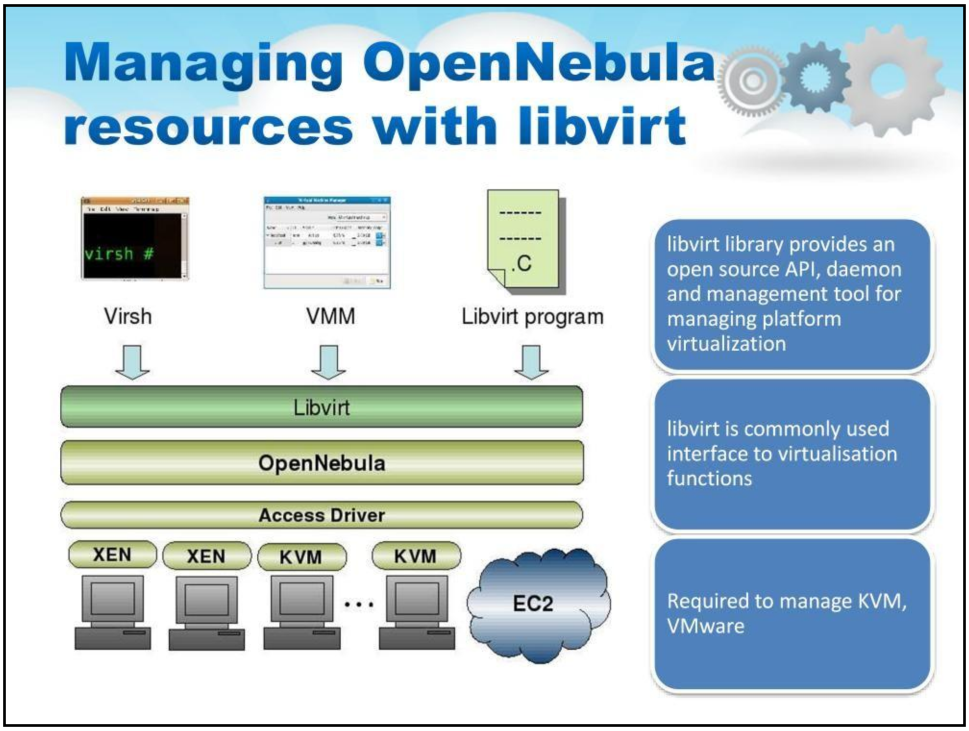
The OpenNebula Cloud API Specification for Ruby has been designed as a wrapper for the XML-RPC methods, with some basic helpers.

Here the Code Sample is presented to perform shutdown for all the VMs of the Pool. The code flow would be as follows:

❑ Create a new Client, setting up the authorization string. You only need to create it once.

❑ Get the VirtualMachine pool that contains the VirtualMachines owned by this User.

❑ You can perform “actions” over these objects right away, like myVNet.delete();. In this example all the VirtualMachines will be shut down.



Managing OpenNebula resources with libvirt

libvirt library provides an open source API, daemon and management tool for managing platform virtualization

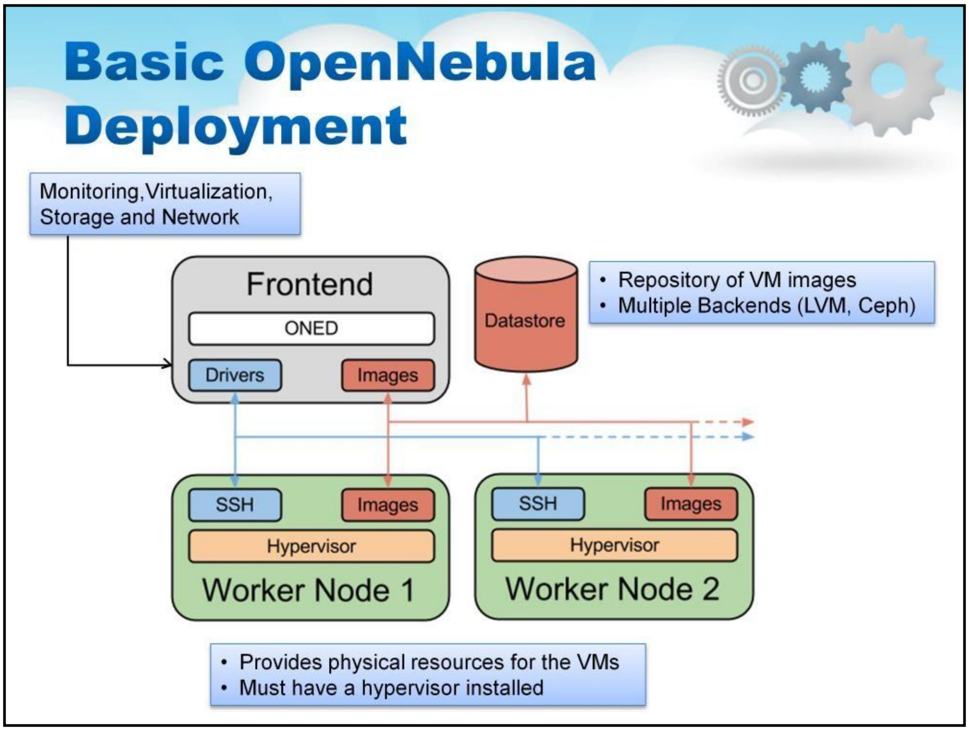
libvirt is commonly used interface to virtualisation functions

Required to manage KVM, VMware

libvirt virtualization API

It can be used as interface for private cloud computing. libvirt version includes OpenNebula driver that provides a libvirt interface to a distributed virtual infrastructure consisting of local resources running VMware, KVM or Xen; and Cloud resources on Amazon EC2 or ElasticHosts.

libvirt is evolving into a very rich and widely used interface to manage the virtualization capabilities of a server, including virtual network, storage and domain management. So, libvirt can be a very effective administration interface for a private cloud exposing a complete set of VM and physical node operations. In this way, libvirt + OpenNebula provides a powerful abstraction for your private cloud



Basic OpenNebula Deployment

Monitoring,Virtualization, Storage and Network

- Repository of VM images
- Multiple Backends (LVM, Ceph)

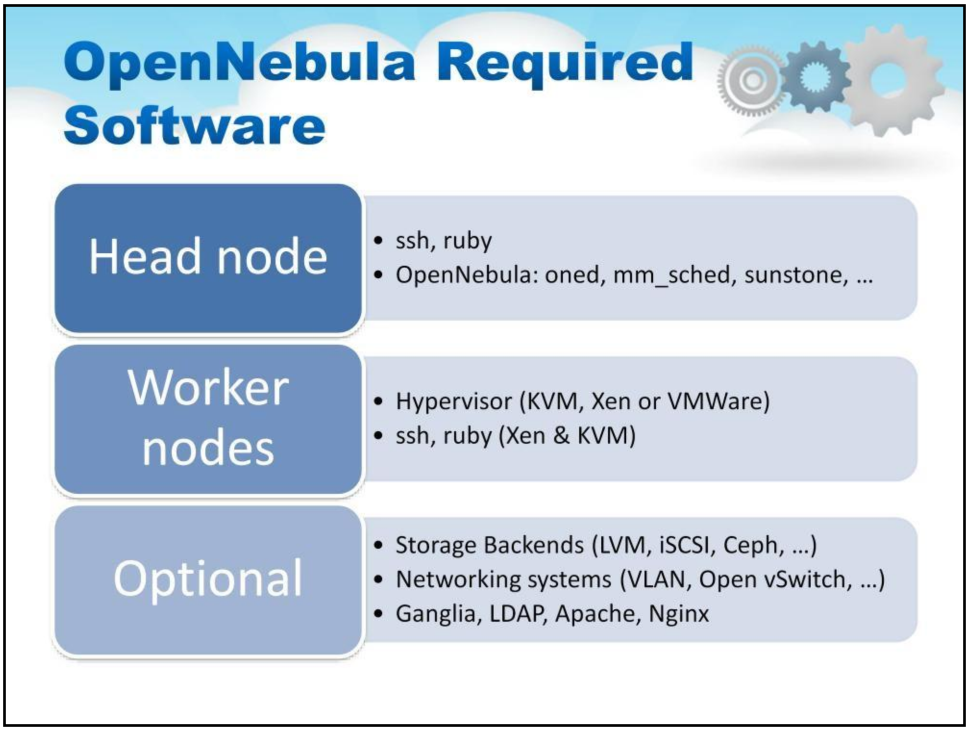
- Provides physical resources for the VMs
- Must have a hypervisor installed

Deployment of OpenNebula

OpenNebula assumes that your physical infrastructure adopts a classical cluster-like architecture with a front-end, and a set of hosts where Virtual Machines (VM) will be executed. There is at least one physical network joining all the hosts with the front-end. The basic components of an OpenNebula system are:

- Front-end, executes the OpenNebula services.
- Hosts, or Noes, hypervisor-enabled hosts that provide the resources needed by the VMs.
- Datastores hold the base images of the VMs.
- Service Network, physical network used to support basic services: interconnection of the storage servers and OpenNebula control operations
- VM Networks physical network that will support VLAN for the VMs.

Front-End. The machine that holds the OpenNebula installation is called the front-end. This machine needs to have access to the storage Datastores (e.g. directly mount or network), and network connectivity to each host. OpenNebula services include:
Hosts. The hosts are the physical machines that will run the VMs. During the installation you will have to configure the OpenNebula administrative account to be able to ssh to the hosts, and depending on your hypervisor you will have to allow this account to execute commands with root privileges or make it part of a given group.
Storage. OpenNebula uses Datastores to handle the VM disk Images. VM Images are registered, or created (empty volumes) in a Datastore. In general, each Datastore has to be accessible through the front-end using any suitable technology.



OpenNebula Required Software

Head node
- ssh, ruby
- OpenNebula: oned, mm_sched, sunstone, …

Worker nodes
- Hypervisor (KVM, Xen or VMWare)
- ssh, ruby (Xen & KVM)

Optional
- Storage Backends (LVM, iSCSI, Ceph, …)
- Networking systems (VLAN, Open vSwitch, …)
- Ganglia, LDAP, Apache, Nginx

Here some software is listed that is required for OpenNebula:

Head node

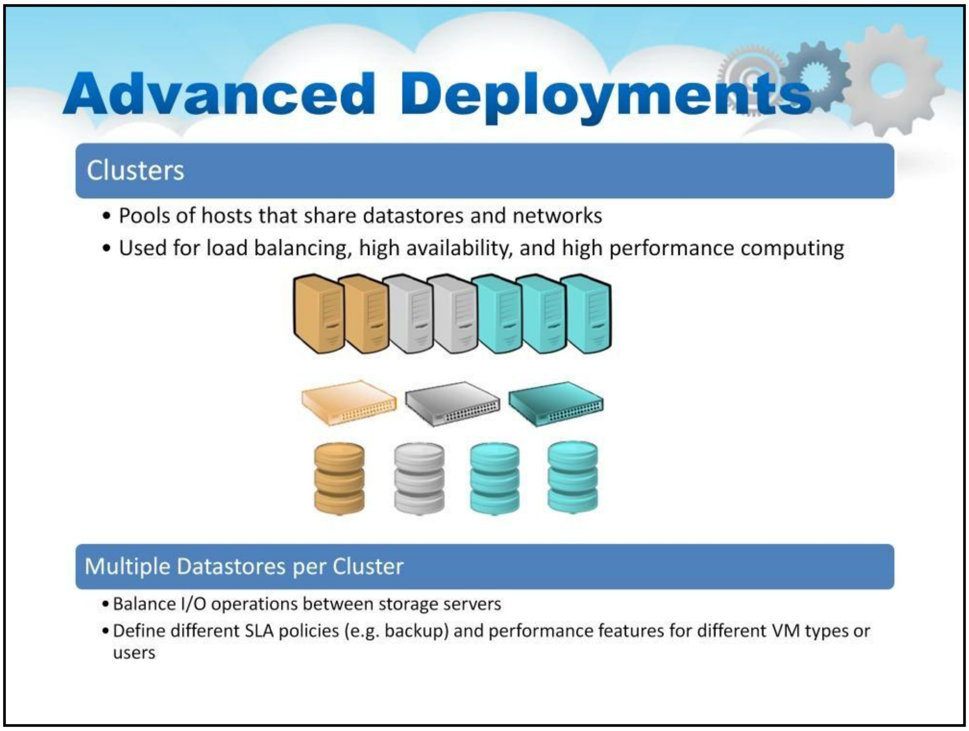
- ssh, ruby
- OpenNebula: oned, mm_sched, sunstone, …

Worker nodes

- Hypervisor (like KVM, Xen or VMWare)
- ssh, ruby (for Xen & KVM)

Optional

- Storage Backends (like LVM, iSCSI, Ceph, …)
- Networking systems (like VLAN, Open vSwitch, …)
- Ganglia, LDAP, Apache, Nginx



Advanced Deployments

Clusters

- Pools of hosts that share datastores and networks
- Used for load balancing, high availability, and high performance computing

Multiple Datastores per Cluster

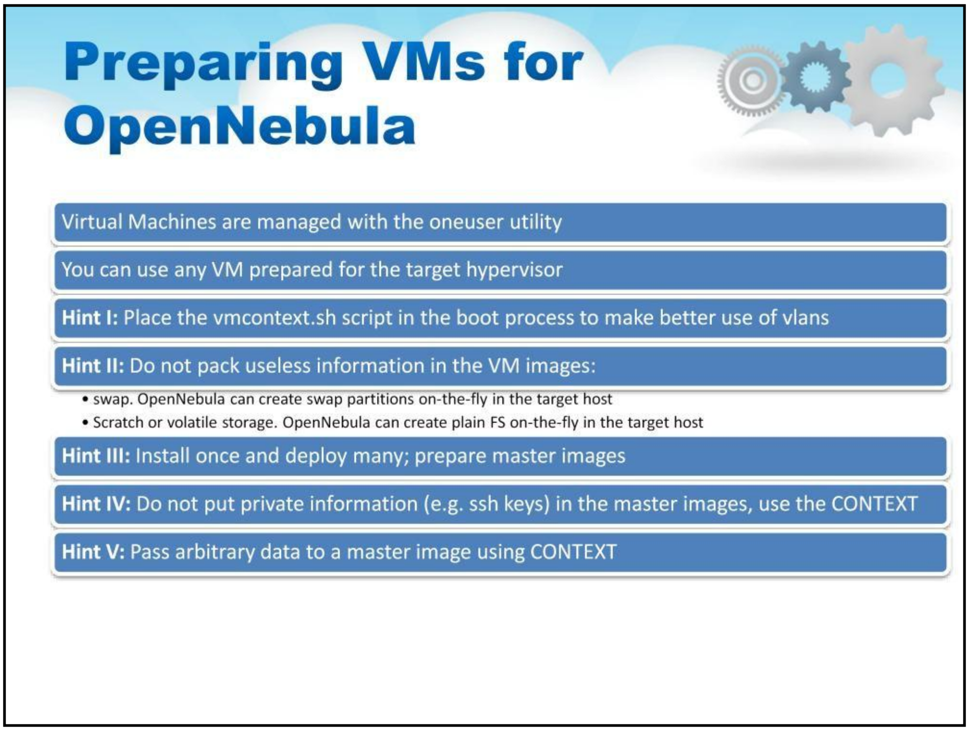
- Balance I/O operations between storage servers
- Define different SLA policies (e.g. backup) and performance features for different VM types or users

Clusters are pools of hosts that share datastores and virtual networks. Clusters are used for load balancing, high availability, and high performance computing. Hosts can be grouped in Clusters.

Hosts can be created directly in a Cluster. Hosts can be in only one Cluster at a time.

Datastores and Virtual Networks can be added to one Cluster. This means that any Host in that Cluster is properly configured to run VMs using Images from the Datastores, or is using leases from the Virtual Networks.
For instance, if you have several Hosts configured to use the iSCSI datastore drivers and Open vSwitch networks, you would group them in the same Cluster.

System Datastore for a Cluster. You can associate an specific System Datastore to a cluster to improve its performance (e.g. balance VM I/O between different servers) or to use different System Datastore types (e.g. shared and ssh). To use a specific System Datastore with your cluster, instead of the default one, just create it, and associate it just like any other datastore.



Preparing VMs for OpenNebula

- Virtual Machines are managed with the oneuser utility
- You can use any VM prepared for the target hypervisor
- **Hint I:** Place the vmcontext.sh script in the boot process to make better use of vlans
- **Hint II:** Do not pack useless information in the VM images:
  - swap. OpenNebula can create swap partitions on-the-fly in the target host
  - Scratch or volatile storage. OpenNebula can create plain FS on-the-fly in the target host
- **Hint III:** Install once and deploy many; prepare master images
- **Hint IV:** Do not put private information (e.g. ssh keys) in the master images, use the CONTEXT
- **Hint V:** Pass arbitrary data to a master image using CONTEXT

The Storage system allows OpenNebula administrators and users to set up images, which can be operative systems or data, to be used in Virtual Machines easily. These images can be used by several Virtual Machines simultaneously, and also shared with other users.

There are different types of images:

OS: An OS image contains a working operative system. Every VM template must define one DISK referring to an image of this type.

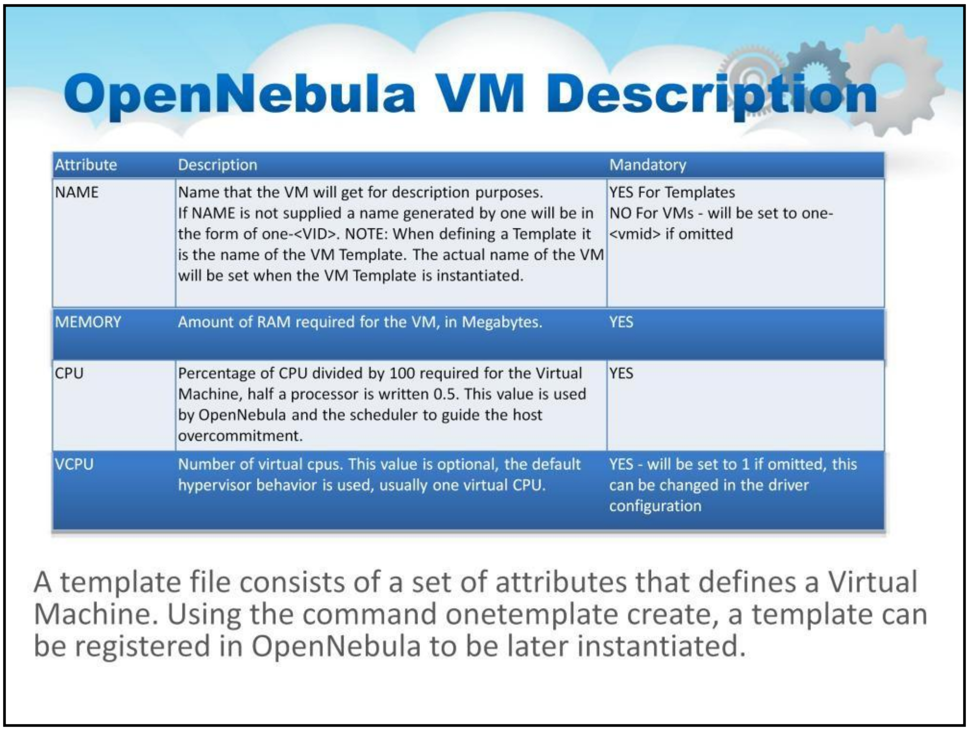
CDROM: This images are readonly data. Only one image of this type can be used in each VM template.

DATABLOCK: A datablock image is a storage for data, which can be accessed and modified from different Virtual Machines. This images can be created from previous existing data, or as an empty drive.

KERNEL: A plain file to be used as kernel. Note that KERNEL file images can be registered only in File Datastores.

RAMDISK: A plain file to be used as ramdisk Note that RAMDISK file images can be registered only in File Datastores.

CONTEXT: A plain file to be included in the context CD-ROM. Note that CONTEXT file images can be registered only in File Datastores.



OpenNebula VM Description

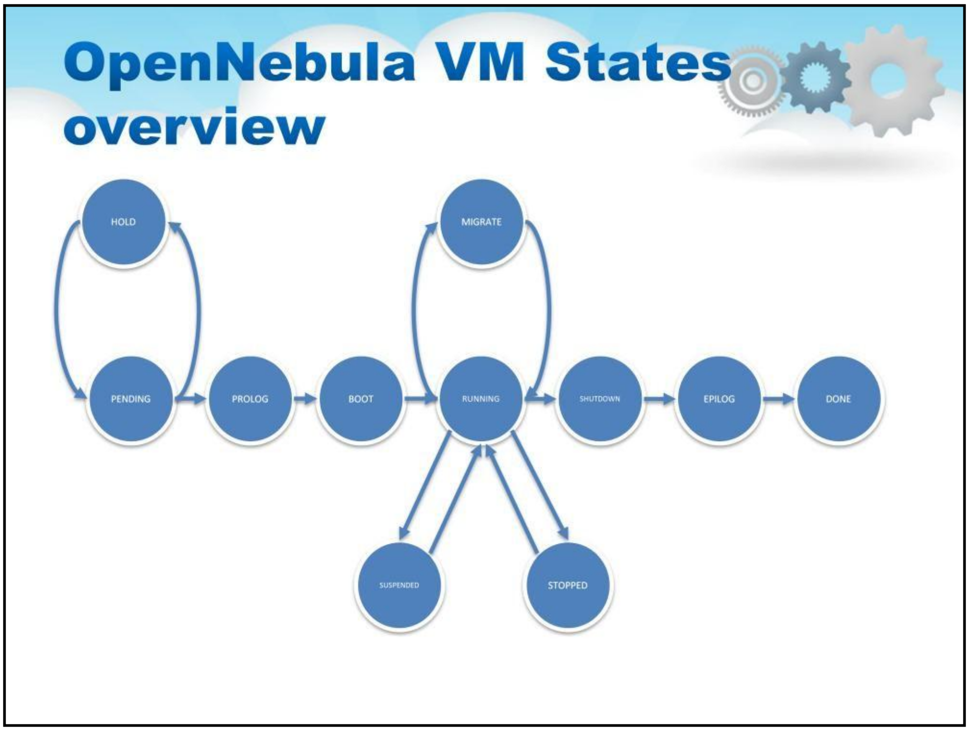
Attribute	Description	Mandatory
NAME	Name that the VM will get for description purposes. If NAME is not supplied a name generated by one will be in the form of one-<VID>. NOTE: When defining a Template it is the name of the VM Template. The actual name of the VM will be set when the VM Template is instantiated.	YES For Templates NO For VMs - will be set to one-<vmid> if omitted
MEMORY	Amount of RAM required for the VM, in Megabytes.	YES
CPU	Percentage of CPU divided by 100 required for the Virtual Machine, half a processor is written 0.5. This value is used by OpenNebula and the scheduler to guide the host overcommitment.	YES
VCPU	Number of virtual cpus. This value is optional, the default hypervisor behavior is used, usually one virtual CPU.	YES - will be set to 1 if omitted, this can be changed in the driver configuration

A template file consists of a set of attributes that defines a Virtual Machine. Using the command onetemplate create, a template can be registered in OpenNebula to be later instantiated.

This slide contains explanations for description of OpenNebula Virtula Machine.

A template file consists of a set of attributes that defines a Virtual Machine.

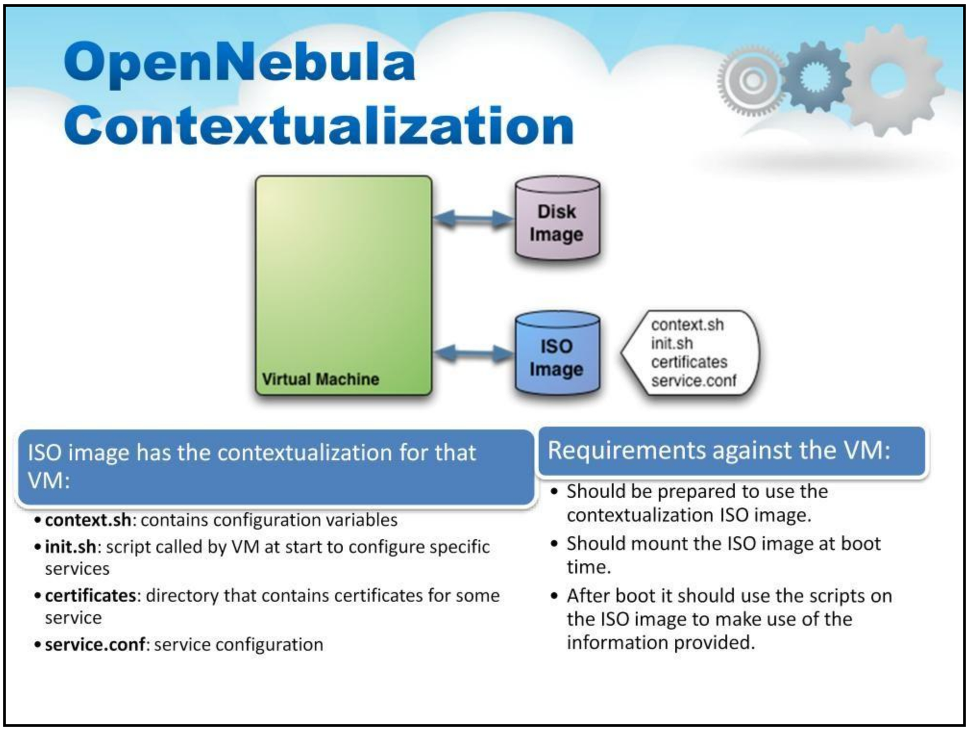
For compatibility with previous versions, you can also create a new Virtual Machine directly from a template file.



OpenNebula VM States overview

One of the most difficult aspects of effectively utilizing any Cloud computing infrastructure is the need to adequately understand the life-cycle that a Cloud-managed virtual machine will progress through.

This is principal scheme of such life-cycle.



OpenNebula Contextualization

Disk Image

ISO Image

Virtual Machine

context.sh
init.sh
certificates
service.conf

ISO image has the contextualization for that VM:

- **context.sh**: contains configuration variables
- **init.sh**: script called by VM at start to configure specific services
- **certificates**: directory that contains certificates for some service
- **service.conf**: service configuration

Requirements against the VM:

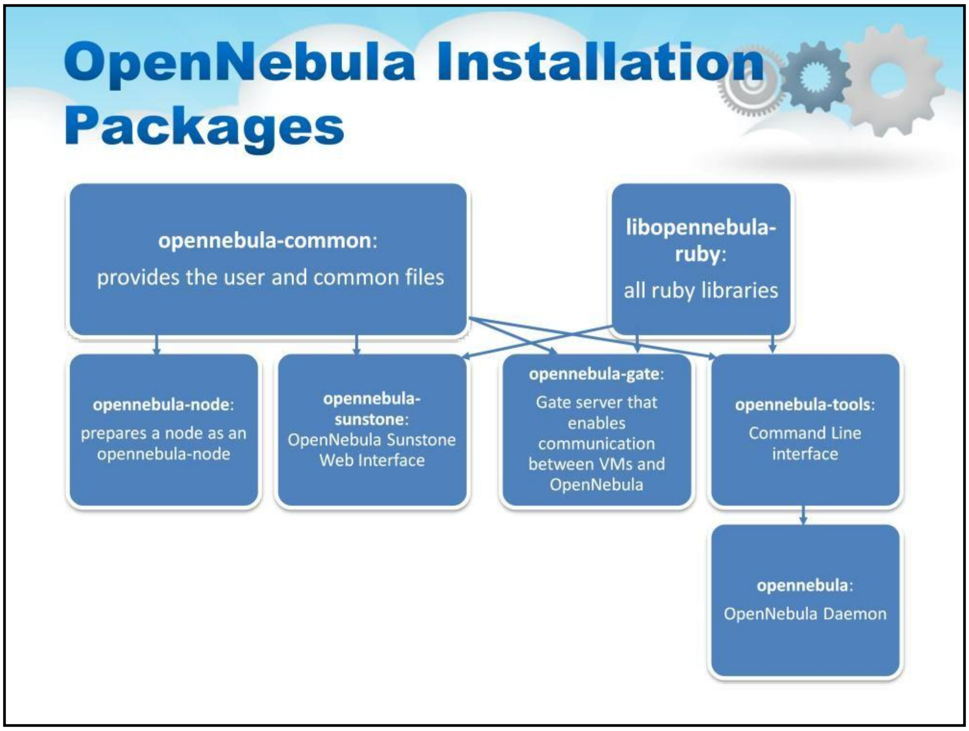
- Should be prepared to use the contextualization ISO image.
- Should mount the ISO image at boot time.
- After boot it should use the scripts on the ISO image to make use of the information provided.

There are two contextualization mechanisms available in OpenNebula:
•the automatic IP assignment, and
•the more generic way to give any file and configuration parameters.

Automatic IP Assignment. With OpenNebula you can derive the IP address assigned to the VM from the MAC address using the MAC_PREFFIX:IP rule. In order to achieve this there are context scripts for Debian, Ubuntu, CentOS and openSUSE based systems.

Generic Contextualization. The method is provided to give configuration parameters to a newly started virtual machine using an ISO image (OVF recommendation). This method is network agnostic so it can be used also to configure network interfaces. In the VM description file you can specify the contents of the iso file (files and directories), tell the device the ISO image will be accessible and specify the configuration parameters that will be written to a file for later use inside the virtual machine.

In this example we see a Virtual Machine with two associated disks. The Disk Image holds the filesystem where the Operating System will run from. The ISO image has the contextualization for that VM.



OpenNebula Installation Packages

opennebula-common: provides the user and common files

libopennebula-ruby: all ruby libraries

opennebula-node: prepares a node as an opennebula-node

opennebula-sunstone: OpenNebula Sunstone Web Interface

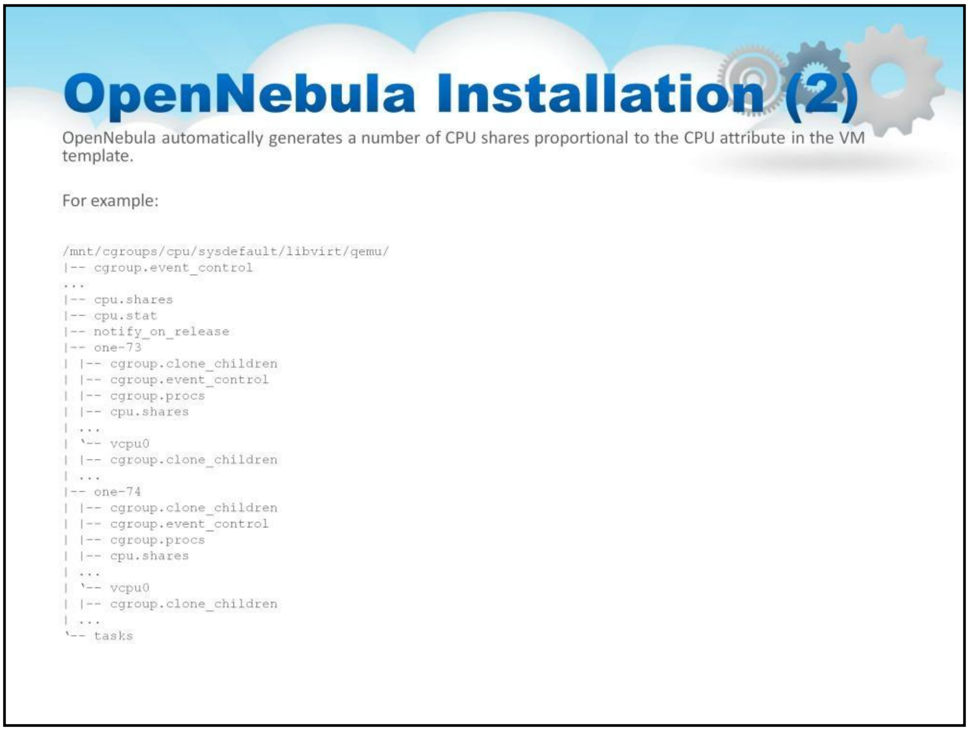
opennebula-gate: Gate server that enables communication between VMs and OpenNebula

opennebula-tools: Command Line interface

opennebula: OpenNebula Daemon

OpenNebula installation packages include the following parts:

opennebula-common: provides the user and common files
libopennebula-ruby: all ruby libraries
opennebula-node: prepares a node as an opennebula-node
opennebula-sunstone: OpenNebula Sunstone Web Interface
opennebula-tools: Command Line interface
opennebula-gate: Gate server that enables communication between VMs and OpenNebula
opennebula-flow: Manages services and elasticity
opennebula: OpenNebula Daemon



OpenNebula Installation (2)

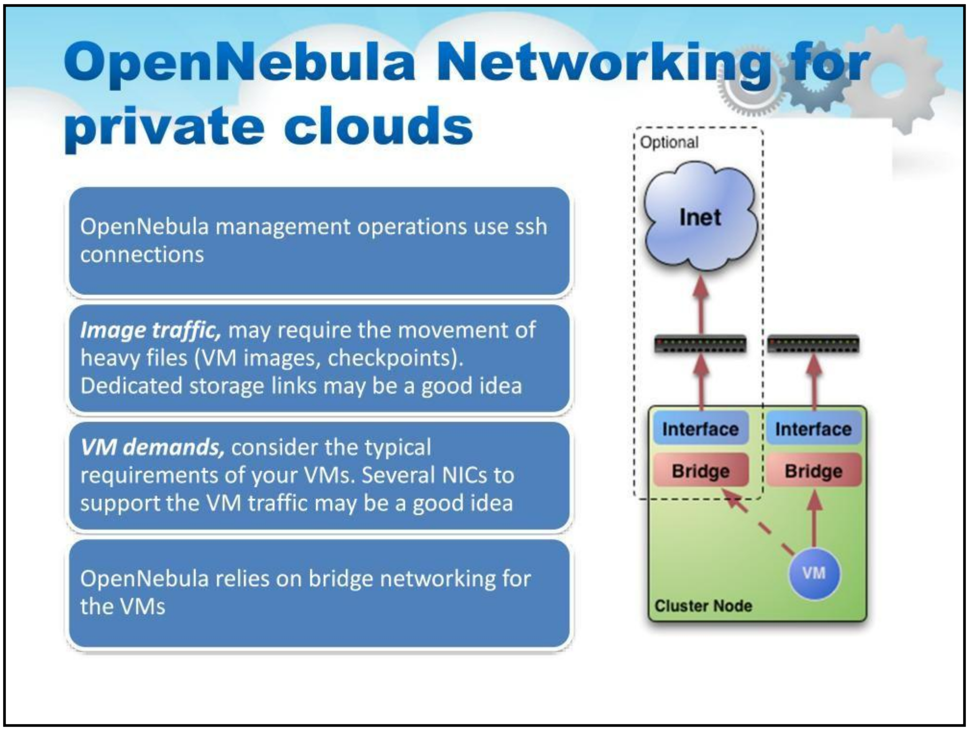
OpenNebula automatically generates a number of CPU shares proportional to the CPU attribute in the VM template.

For example:

```
/mnt/cgroups/cpu/sysdefault/libvirt/qemu/
|-- cgroup.event_control
...
|-- cpu.shares
|-- cpu.stat
|-- notify_on_release
|-- one-73
| |-- cgroup.clone_children
| |-- cgroup.event_control
| |-- cgroup.procs
| |-- cpu.shares
| ...
| `-- vcpu0
| |-- cgroup.clone_children
| ...
|-- one-74
| |-- cgroup.clone_children
| |-- cgroup.event_control
| |-- cgroup.procs
| |-- cpu.shares
| ...
| `-- vcpu0
| |-- cgroup.clone_children
| ...
`-- tasks
```

OpenNebula automatically generates a number of CPU shares proportional to the CPU attribute in the VM template.

For example, consider a host running 2 VMs (73 and 74) with the various shares of CPUs for them.
If everything is properly configured you should see:

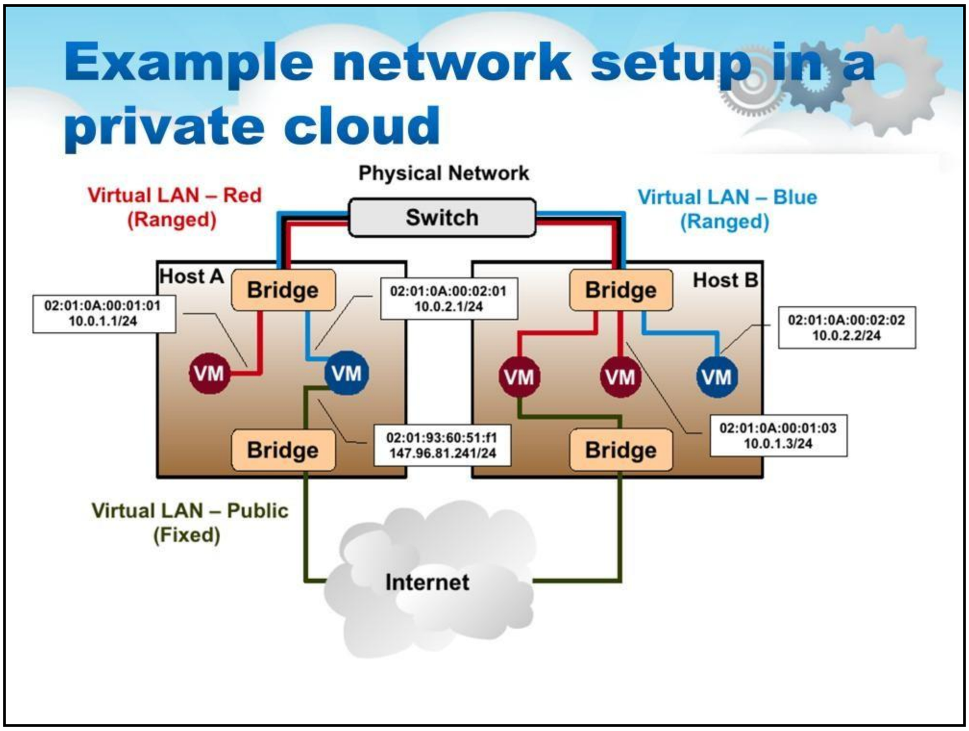


OpenNebula Networking for private clouds

- OpenNebula management operations use ssh connections
- ***Image traffic,*** may require the movement of heavy files (VM images, checkpoints). Dedicated storage links may be a good idea
- ***VM demands,*** consider the typical requirements of your VMs. Several NICs to support the VM traffic may be a good idea
- OpenNebula relies on bridge networking for the VMs

When a new Virtual Machine is launched, OpenNebula will connect its network interfaces (defined in the NIC section of the template) to the bridge or physical device specified in the Virtual Network definition. This will allow the VM to have access to different networks, public or private. This functionality is provided through Virtual Network Manager drivers.

To make an effective use of your VM deployments you'll probably need to make one or more physical networks accessible to them. For example, a typical host with two physical networks, one for public IP addresses (attached to eth0 NIC) and the other for private virtual LANs (NIC eth1) should have two bridges (as the illustration shows).



Example network setup in a private cloud

OpenNebula administrator may associate one of the following drivers to each Host:

dummy: Default driver that doesn't perform any network operation. Firewalling rules are also ignored.

fw: Firewall rules are applied, but networking isolation is ignored.

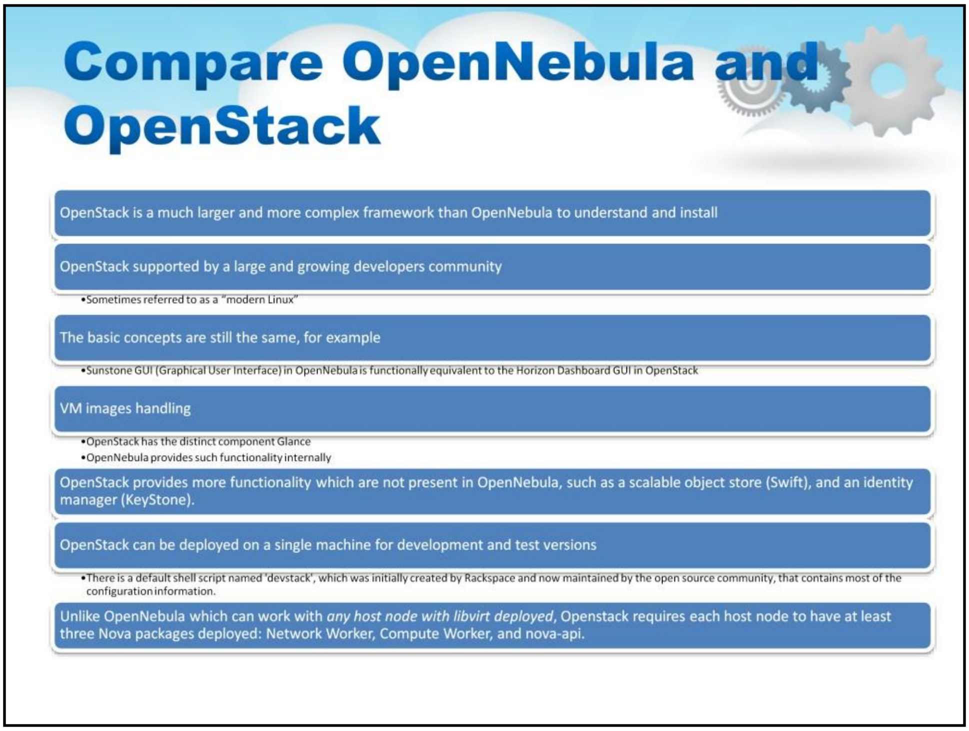
802.1Q: restrict network access through VLAN tagging, which also requires support from the hardware switches.

ebtables: restrict network access through Ebtables rules. No special hardware configuration required.

ovswitch: restrict network access with Open vSwitch Virtual Switch.

VMware: uses the VMware networking infrastructure to provide an isolated and 802.1Q compatible network for VMs launched with the VMware hypervisor.
Note that some of these drivers also create the bridging device in the hosts.

The network is needed by the OpenNebula front-end daemons to access the hosts to manage and monitor the hypervisors; and move image files. It is highly recommended to install a dedicated network for this purpose. To offer network connectivity to the VMs across the different hosts, the default configuration connects the virtual machine network interface to a bridge in the physical host.



Compare OpenNebula and OpenStack

- OpenStack is a much larger and more complex framework than OpenNebula to understand and install
- OpenStack supported by a large and growing developers community
  - •Sometimes referred to as a "modern Linux"
- The basic concepts are still the same, for example
  - •Sunstone GUI (Graphical User Interface) in OpenNebula is functionally equivalent to the Horizon Dashboard GUI in OpenStack
- VM images handling
  - •OpenStack has the distinct component Glance
  - •OpenNebula provides such functionality internally
- OpenStack provides more functionality which are not present in OpenNebula, such as a scalable object store (Swift), and an identity manager (KeyStone).
- OpenStack can be deployed on a single machine for development and test versions
  - •There is a default shell script named 'devstack', which was initially created by Rackspace and now maintained by the open source community, that contains most of the configuration information.
- Unlike OpenNebula which can work with *any host node with libvirt deployed*, Openstack requires each host node to have at least three Nova packages deployed: Network Worker, Compute Worker, and nova-api.

Now let's compare OpenNebula and OpenStack which will be considered in the next section (and left for the self-guided work).

OpenStack is a much larger and more complex framework than OpenNebula to understand and install.

OpenStack supported by a large and growing developers community

Sometimes referred to as a "modern Linux"

The basic concepts are still the same, for example

Sunstone GUI (Graphical User Interface) in OpenNebula is functionally equivalent to the Horizon Dashboard GUI in OpenStack

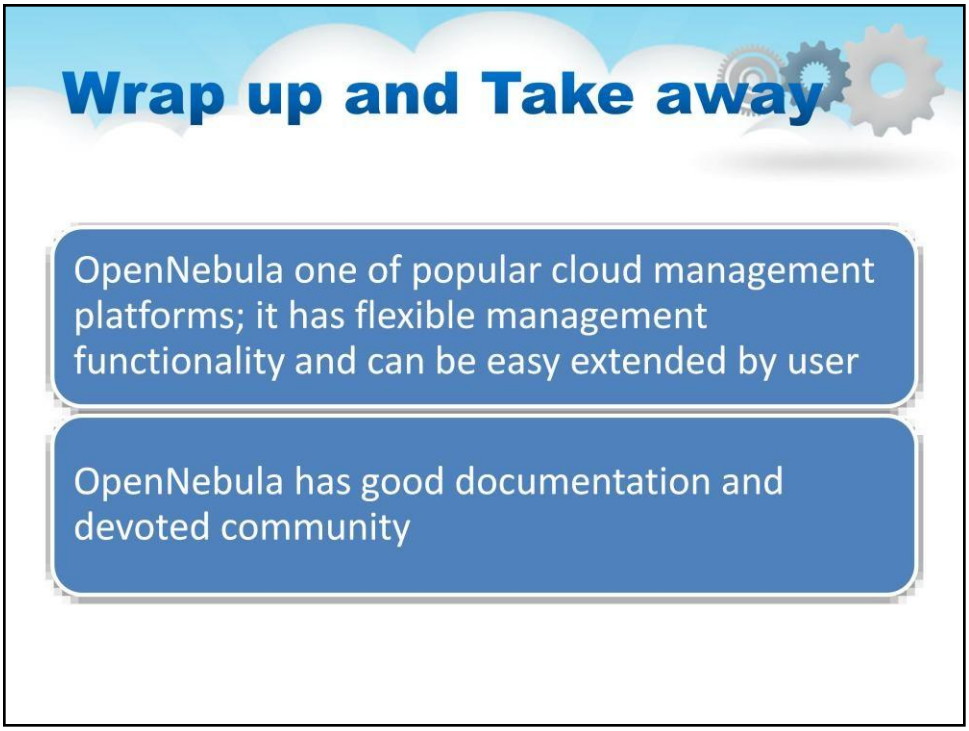
VM images handling

OpenStack has the distinct component Glance

OpenNebula provides such functionality internally

OpenStack provides more functionality which are not present in OpenNebula, such as a scalable object store (Swift), and an identity manager (KeyStone).

OpenStack can be deployed on a single machine for development and test versions



Wrap up and Take away

OpenNebula one of popular cloud management platforms; it has flexible management functionality and can be easy extended by user

OpenNebula has good documentation and devoted community

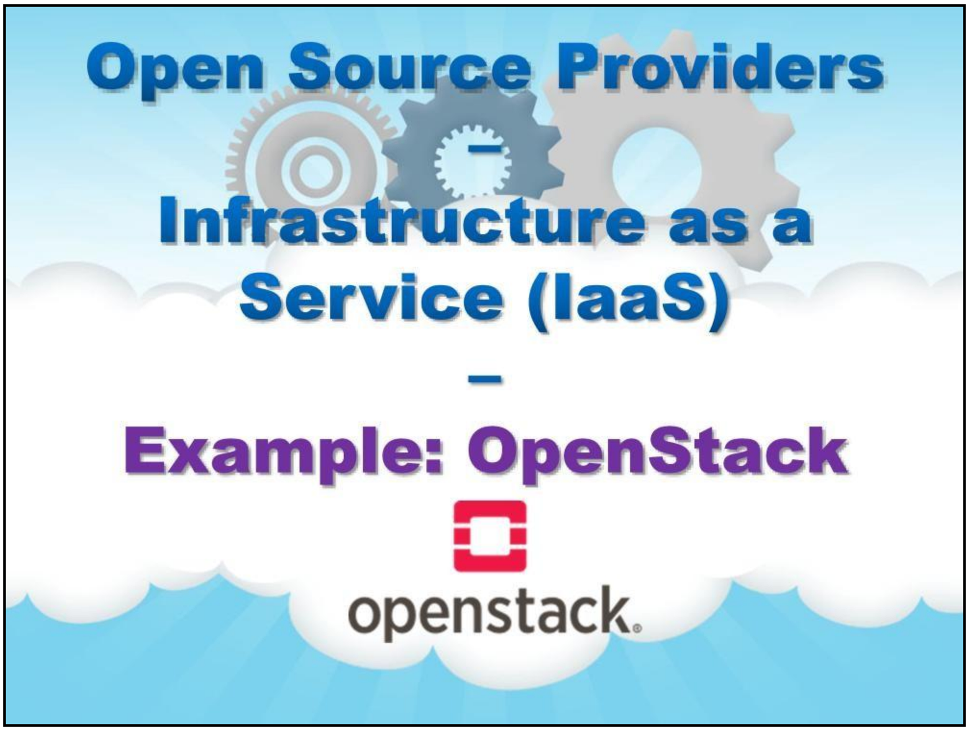
The following final notes should be made as to OpenNebula:

❑ OpenNebula is one of popular cloud management platforms

❑ OpenNebula has flexible management functionality

❑ OpenNebula can be easily extended by user

❑ OpenNebula has good documentation and devoted community



Open Source Providers

–

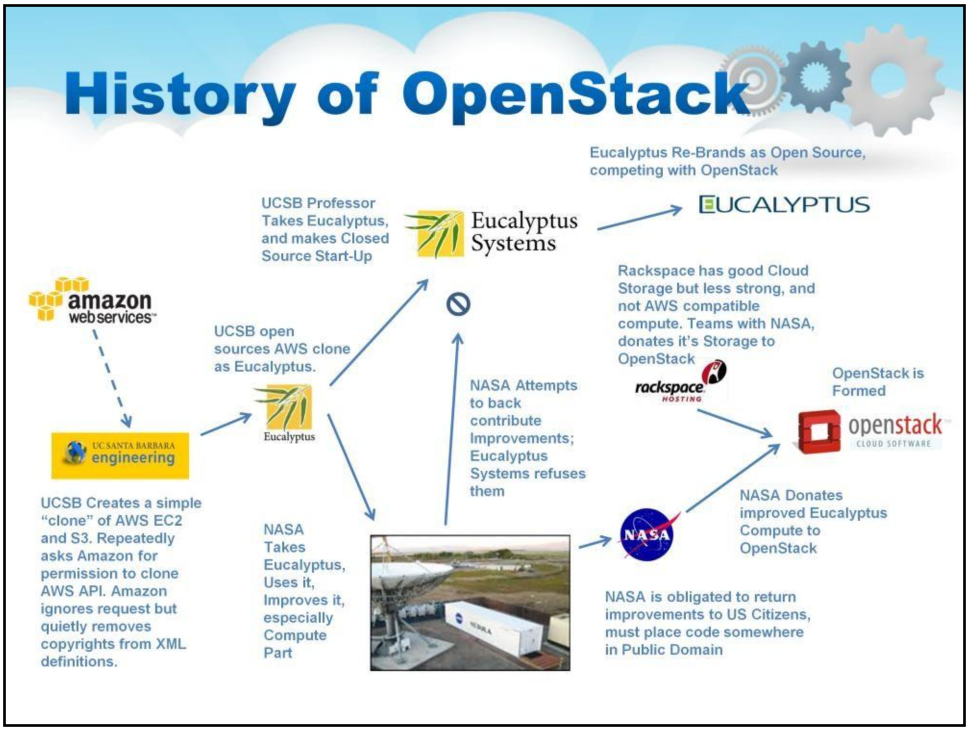
Infrastructure as a Service (IaaS)

–

Example: OpenStack

Example: OpenStack

This section and following ones are left for the self-guided work.



History of OpenStack

Eucalyptus Re-Brands as Open Source, competing with OpenStack

UCSB Professor Takes Eucalyptus, and makes Closed Source Start-Up

Rackspace has good Cloud Storage but less strong, and not AWS compatible compute. Teams with NASA, donates it's Storage to OpenStack

UCSB open sources AWS clone as Eucalyptus.

OpenStack is Formed

NASA Attempts to back contribute Improvements; Eucalyptus Systems refuses them

NASA Donates improved Eucalyptus Compute to OpenStack

UCSB Creates a simple "clone" of AWS EC2 and S3. Repeatedly asks Amazon for permission to clone AWS API. Amazon ignores request but quietly removes copyrights from XML definitions.

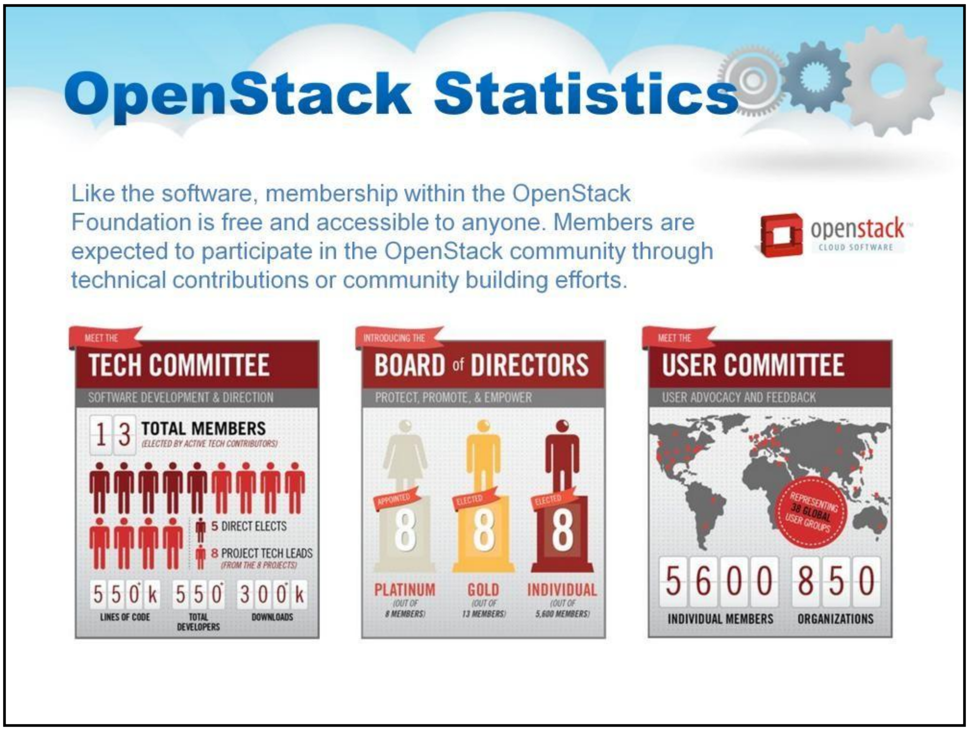
NASA Takes Eucalyptus, Uses it, Improves it, especially Compute Part

NASA is obligated to return improvements to US Citizens, must place code somewhere in Public Domain

OpenStack came about in a very round-about fashion, as the illustration portrays. This is actually a little known story.

The inspiration for OpenStack came from the University of California, Santa Barbara Eucalyptus Project, which NASA has done significant work on. However, NASA was unable to give their changes back to the Eucalyptus project, as it had become a spinoff company already and was pursuing a "closed source" strategy. As NASA was required by law to make the improvements available somehow (they were owned by the citizens of the United States, as NASA is funded with taxpayer money), when Rackspace was approached to host a repository, they decided to throw in their code as well and create OpenStack as we largely know it today.

In July 2010 Rackspace Hosting and NASA jointly launched an open-source cloud-software initiative known as OpenStack. The OpenStack project intended to help organizations which offer cloud-computing services running on standard hardware. The first official release, code-named Austin, appeared four months later, with plans to release regular updates of the software every few months. The early code came from the NASA Nebula platform and from the Rackspace Cloud Files platform. In July 2011, Ubuntu Linux developers adopted OpenStack.



OpenStack Statistics

Like the software, membership within the OpenStack Foundation is free and accessible to anyone. Members are expected to participate in the OpenStack community through technical contributions or community building efforts.

openstack
CLOUD SOFTWARE
MEET THE
TECH COMMITTEE
SOFTWARE DEVELOPMENT & DIRECTION
13 TOTAL MEMBERS
(ELECTED BY ACTIVE TECH CONTRIBUTORS)
5 DIRECT ELECTS
8 PROJECT TECH LEADS
(FROM THE 8 PROJECTS)
550k LINES OF CODE
550 TOTAL DEVELOPERS
300k DOWNLOADS
INTRODUCING THE
BOARD of DIRECTORS
PROTECT, PROMOTE, & EMPOWER
APPOINTED 8 PLATINUM (OUT OF 8 MEMBERS)
ELECTED 8 GOLD (OUT OF 13 MEMBERS)
ELECTED 8 INDIVIDUAL (OUT OF 5,600 MEMBERS)
MEET THE
USER COMMITTEE
USER ADVOCACY AND FEEDBACK
REPRESENTING 38 GLOBAL USER GROUPS
5600 INDIVIDUAL MEMBERS
850 ORGANIZATIONS

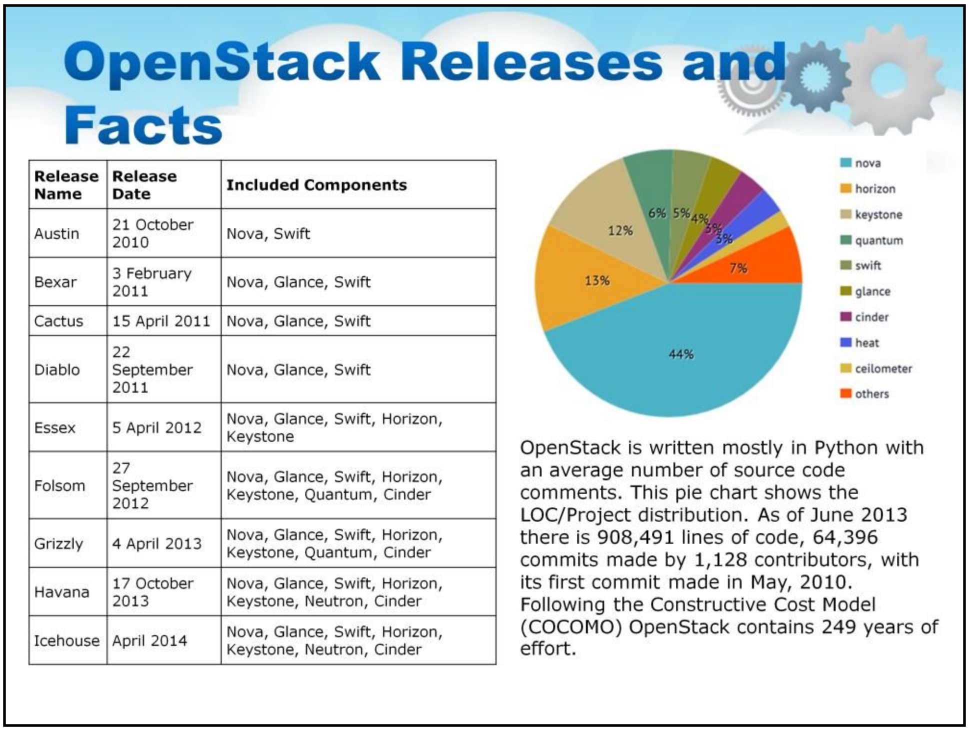
The OpenStack Foundation, established September 2012, is an independent body providing shared resources to help achieve the OpenStack Mission by protecting, empowering, and promoting OpenStack software and the community around it. This includes users, developers and the entire ecosystem.

Founded by Rackspace Hosting and NASA, OpenStack has grown to be a global software community of developers collaborating on a standard and massively scalable open source cloud operating system.

The OpenStack Foundation promotes the development, distribution and adoption of the OpenStack cloud operating system. As the independent home for OpenStack, the Foundation has already attracted more than 7,000 individual members from 100 countries and 850 different organizations. It has also secured more than $10 million in funding.

All of the code for OpenStack is freely available under the Apache 2.0 license.

Some OpenStack users include: PayPal / eBay, NASA, CERN, Yahoo! Rackspace Cloud HP Public Cloud, Mercado, Libre.com, AT&T, KT (formerly Korea Telecom), Deutsche Telekom, Wikimedia Labs, Hostalia of Telef nica Group, SUSE Cloud solution, Red Hat OpenShift PaaS solution Zadara Storage, etc.



OpenStack Releases and Facts

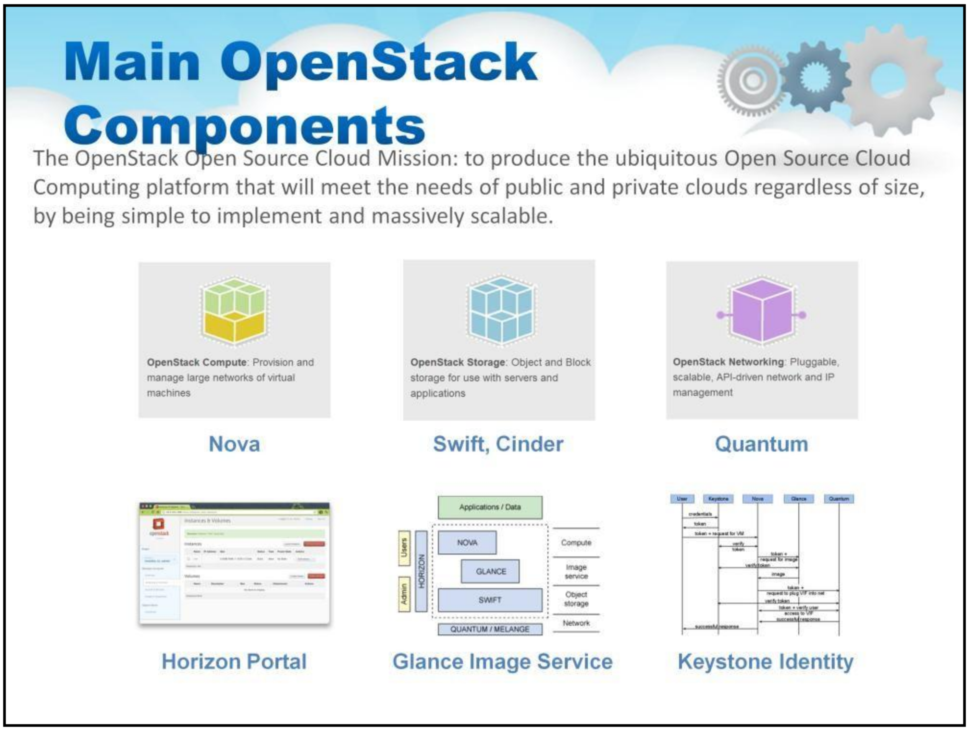
Release Name	Release Date	Included Components
Austin	21 October 2010	Nova, Swift
Bexar	3 February 2011	Nova, Glance, Swift
Cactus	15 April 2011	Nova, Glance, Swift
Diablo	22 September 2011	Nova, Glance, Swift
Essex	5 April 2012	Nova, Glance, Swift, Horizon, Keystone
Folsom	27 September 2012	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder
Grizzly	4 April 2013	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder
Havana	17 October 2013	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder
Icehouse	April 2014	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder

OpenStack is written mostly in Python with an average number of source code comments. This pie chart shows the LOC/Project distribution. As of June 2013 there is 908,491 lines of code, 64,396 commits made by 1,128 contributors, with its first commit made in May, 2010. Following the Constructive Cost Model (COCOMO) OpenStack contains 249 years of effort.

OpenStack is based on a coordinated 6-month release cycle with frequent development milestones.

The creation of OpenStack took an estimated 249 years of effort (COCOMO model).

In a nutshell, OpenStack has:
64,396 commits made by 1,128 contributors, with its first commit made in May, 2010.
908,491 lines of code. OpenStack is written mostly in Python with an average number of source code comments.
A code base with a long source history.
Increasing Y-O-Y commits.
A very large development team comprised of people from around the world.



Main OpenStack Components

The OpenStack Open Source Cloud Mission: to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable.

OpenStack Compute: Provision and manage large networks of virtual machines

Nova

OpenStack Storage: Object and Block storage for use with servers and applications

Swift, Cinder

OpenStack Networking: Pluggable, scalable, API-driven network and IP management

Quantum

Horizon Portal

Glance Image Service

Keystone Identity

OpenStack has a modular architecture with various code names for its components. OpenStack has several shared services that span the three pillars of compute, storage and networking, making it easier to implement and operate your cloud. These services - including identity, image management and a web interface - integrate the OpenStack components with each other as well as external systems to provide a unified experience for users as they interact with different cloud resources.

These are

Nova: OpenStack Compute: Provision and manage large networks of virtual machines

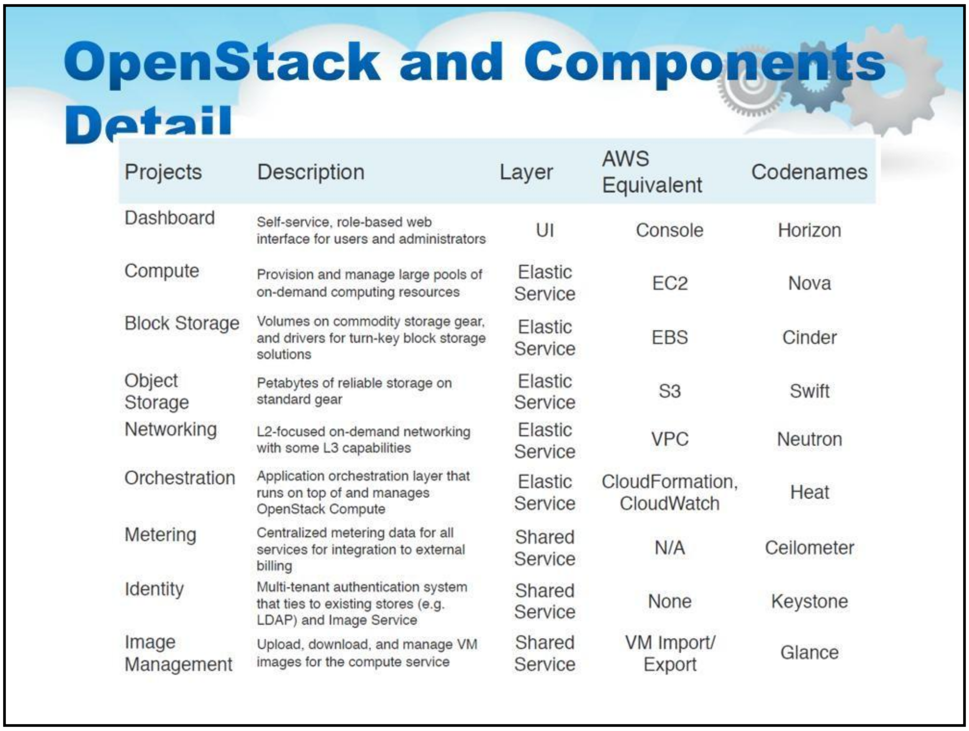
Swift and Cinder: OpenStack Storage: Object and Block storage for use with servers and applications

Neutron: OpenStack Networking: Pluggable, scalable, API-driven network and IP management

Horizon : A Dashboard

Keystone: Identity Services

Glance: Image Services



OpenStack and Components Detail

Projects	Description	Layer	AWS Equivalent	Codenames
Dashboard	Self-service, role-based web interface for users and administrators	UI	Console	Horizon
Compute	Provision and manage large pools of on-demand computing resources	Elastic Service	EC2	Nova
Block Storage	Volumes on commodity storage gear, and drivers for turn-key block storage solutions	Elastic Service	EBS	Cinder
Object Storage	Petabytes of reliable storage on standard gear	Elastic Service	S3	Swift
Networking	L2-focused on-demand networking with some L3 capabilities	Elastic Service	VPC	Neutron
Orchestration	Application orchestration layer that runs on top of and manages OpenStack Compute	Elastic Service	CloudFormation, CloudWatch	Heat
Metering	Centralized metering data for all services for integration to external billing	Shared Service	N/A	Ceilometer
Identity	Multi-tenant authentication system that ties to existing stores (e.g. LDAP) and Image Service	Shared Service	None	Keystone
Image Management	Upload, download, and manage VM images for the compute service	Shared Service	VM Import/ Export	Glance

Dashboard(Horizon) provides administrators and users a graphical interface to access, provision and automate cloud-based resources.

Compute (Nova). The OpenStack cloud operating system enables enterprises and service providers to offer on-demand computing resources, by provisioning and managing large networks of virtual machines. Compute resources are accessible via APIs for developers building cloud applications and via web interfaces for administrators and users. The compute architecture is designed to scale horizontally on standard hardware.

Block Storage(Cinder) provides persistent block level storage devices for use with OpenStack compute instances. The block storage system manages the creation, attaching and detaching of the block devices to servers.

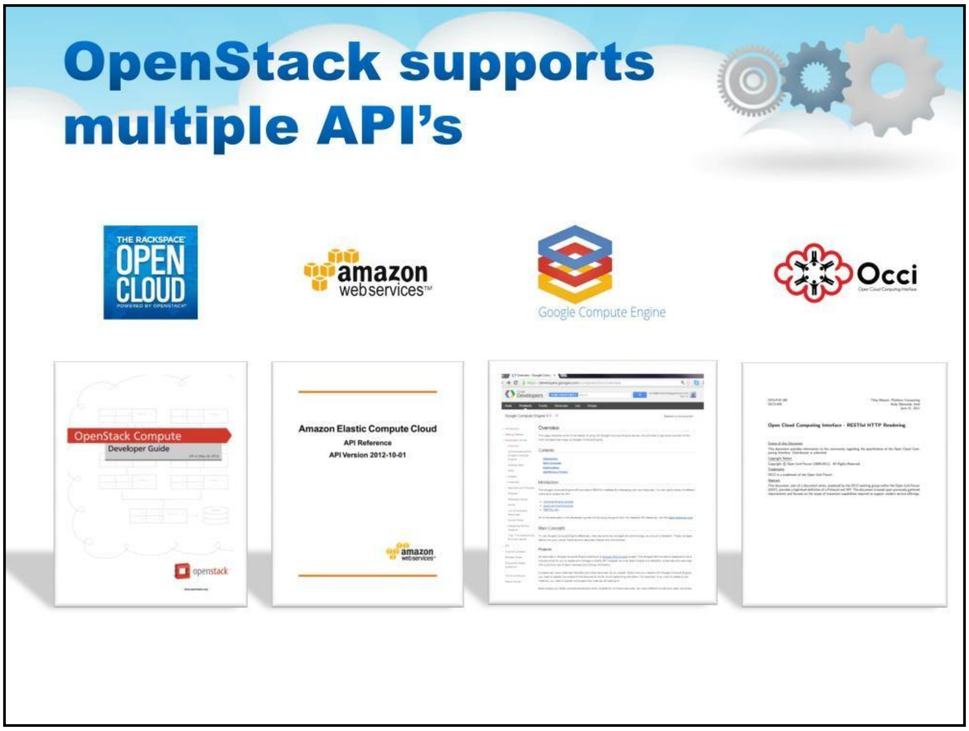
Object Storage(Swift). In addition to traditional enterprise-class storage technology, many organizations now have a variety of storage needs with varying performance and price requirements. OpenStack has support for both Object Storage and Block Storage, with many deployment options for each depending on the use case.

Networking(Neutron). Today's data center networks contain more devices than ever before. From servers, network equipment, storage systems and security appliances, many of which are further divided into virtual machines and virtual networks. The number of IP addresses, routing configurations and security rules can quickly grow into the millions. Traditional network management techniques fall short of providing a truly scalable, automated approach to managing these next-generation networks. At the same time, users expect more control and flexibility with quicker provisioning.

Identity Service(Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP.

Image Service(Glance)

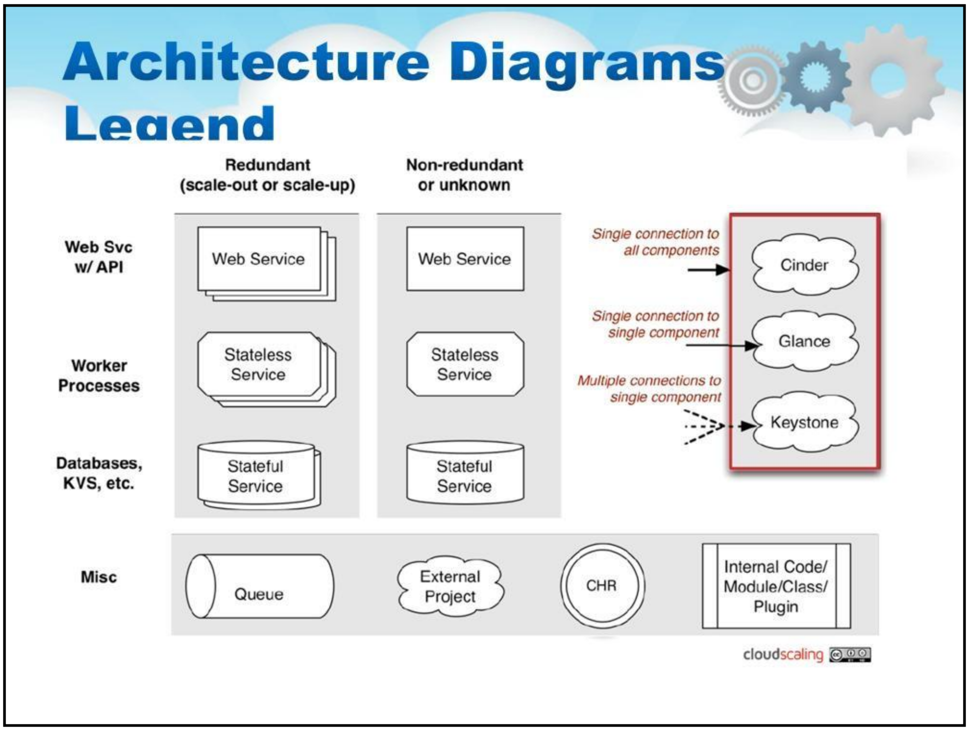
Image Service (Glance) provides discovery, registration and delivery services for disk and server images.



OpenStack supports multiple API's

THE RACKSPACE OPEN CLOUD POWERED BY OPENSTACK
amazon web services
Google Compute Engine
Occi Open Cloud Computing Interface
OpenStack Compute Developer Guide
openstack
Amazon Elastic Compute Cloud
API Reference
API Version 2012-10-01
amazon webservices
Open Cloud Computing Interface - RESTful HTTP Rendering

OpenStack APIs are compatible with Amazon EC2 and Amazon S3 and thus client applications written for Amazon Web Services can be used with OpenStack with minimal porting effort.



Architecture Diagrams Legend

Redundant (scale-out or scale-up)

Non-redundant or unknown

Web Svc w/ API

Web Service

Web Service

Worker Processes

Stateless Service

Stateless Service

Databases, KVS, etc.

Stateful Service

Stateful Service

Single connection to all components

Single connection to single component

Multiple connections to single component

Cinder

Glance

Keystone

Misc

Queue

External Project

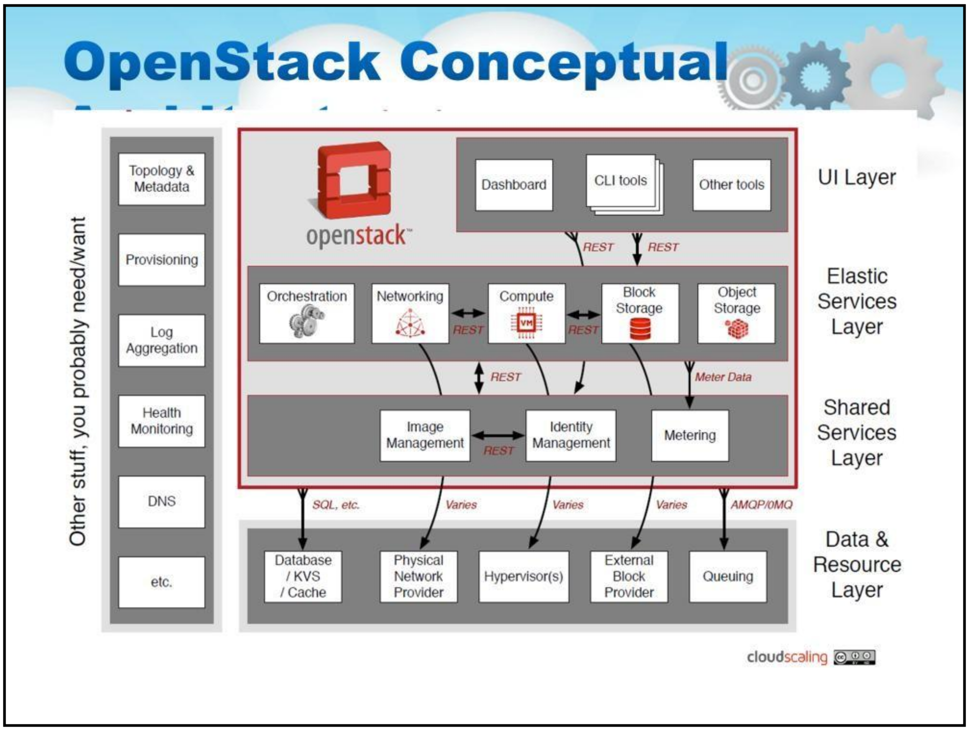
CHR

Internal Code/ Module/Class/ Plugin

cloudscaling

The subsequent slides contain architectural diagrams of OpenStack internals.

Very specific conventions are used to indicate the type of element. Please refer to this diagram when studying the following diagrams.

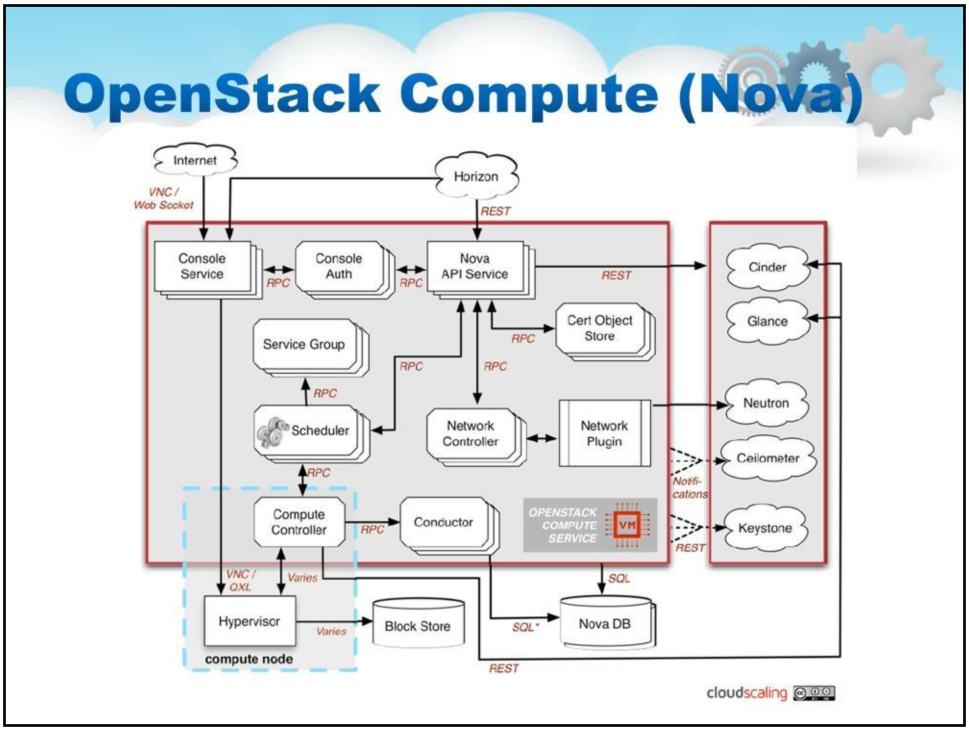


OpenStack Conceptual

Conceptual Architecture

The OpenStack project as a whole is designed to deliver a massively scalable cloud operating system. To achieve this, each of the constituent services are designed to work together to provide a complete Infrastructure-as-a-Service (IaaS). This integration is facilitated through public application programming interfaces (APIs) that each service offers (and in turn can consume). While these APIs allow each of the services to use another service, it also allows an implementer to switch out any service as long as they maintain the API. These are (mostly) the same APIs that are available to end users of the cloud.

Dashboard ("Horizon") provides a web front end to the other OpenStack services
Compute ("Nova") stores and retrieves virtual disks ("images") and associated metadata in Image ("Glance")
Network ("Neutron") provides virtual networking for Compute.
Block Storage ("Cinder") provides storage volumes for Compute.
Image ("Glance") can store the actual virtual disk files in the Object Store("Swift")
All the services authenticate with Identity ("Keystone")
End users can interact through a common web interface (Horizon) or directly to each service through their API
All services authenticate through a common source (facilitated through keystone)
Individual services interact with each other through their public APIs (except where privileged administrator commands are necessary)



OpenStack Compute (Nova)

OpenStack Compute (Nova) is a cloud computing fabric controller (the main part of an IaaS system). It is written in Python and uses many external libraries such as Eventlet (for concurrent programming), Kombu (for AMQP communication), and SQLAlchemy (for database access). Nova's architecture is designed to scale horizontally on standard hardware with no proprietary hardware or software requirements and provide the ability to integrate with legacy systems and third party technologies. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high- performance computing (HPC) configurations. KVM and XenServer are available choices for hypervisor technology, together with Hyper-V and Linux container technology such as LXC. In addition to different hypervisors, OpenStack runs on ARM.

Popular Use Cases:
Service providers offering an IaaS compute platform or services higher up the stack IT departments acting as cloud service providers for business units and project teams Processing big data with tools like Hadoop

Scaling compute up and down to meet demand for web resources and applications High-performance computing (HPC) environments processing diverse and intensive workloads

Nova is the most complicated and distributed component of OpenStack. A large number of processes cooperate to turn end user API requests into running virtual machines. Below is a list of these processes and their functions:

nova-api accepts and responds to end user compute API calls. It supports OpenStack Compute API, Amazon's EC2 API and a special Admin API (for privileged users to perform administrative actions). It also initiates most of the orchestration activities (such as running an instance) as well as enforces some policy (mostly quota checks).

The nova-compute process is primarily a worker daemon that creates and terminates virtual machine instances via hypervisor's APIs (XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, etc.). The process by which it does so is fairly complex but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database.

nova-volume manages the creation, attaching and detaching of z volumes to compute instances (similar functionality to Amazon’s Elastic Block Storage). It can use volumes from a variety of providers such as iSCSI or Rados Block Device in Ceph. A new OpenStack project, Cinder, will eventually replace nova-volume functionality. In the Folsom release, nova-volume and the Block Storage service will have similar functionality.

The nova-network worker daemon is very similar to nova-compute and nova-volume. It accepts networking tasks from the queue and then performs tasks to manipulate the network (such as setting up bridging interfaces or changing iptables rules). This functionality is being migrated to Neutron, a separate OpenStack project. In the Folsom release, much of the functionality will be duplicated between nova-network and Neutron.

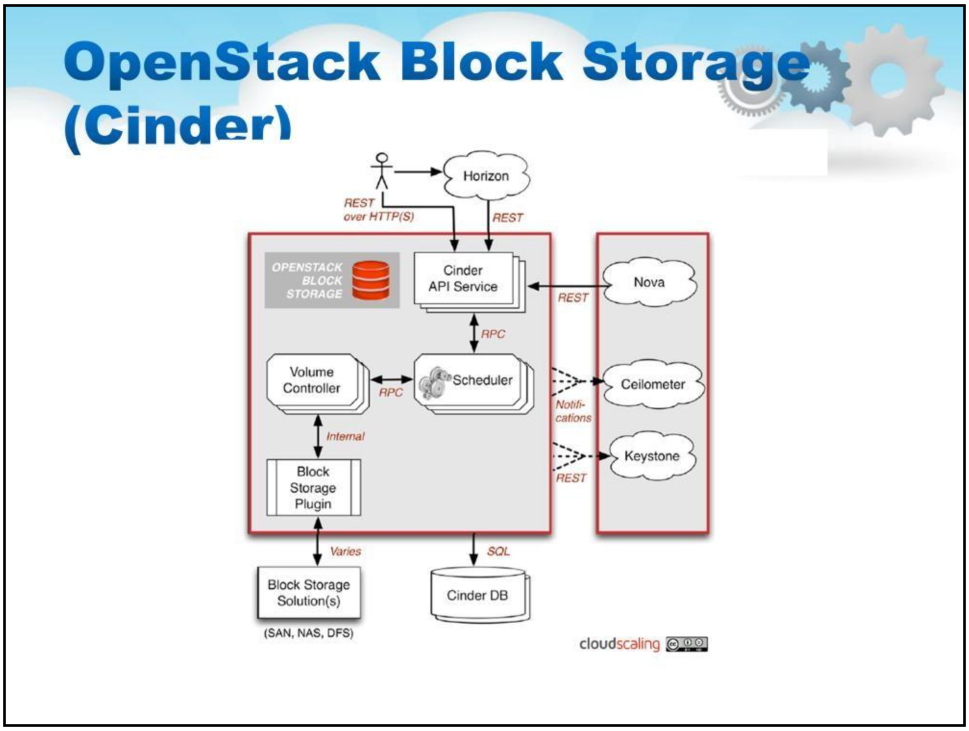
The nova-schedule process is conceptually the simplest piece of code in OpenStack Nova: it takes a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on).

The queue provides a central hub for passing messages between daemons. This is usually implemented with RabbitMQ today, but could be any AMQP message queue (such as Apache Qpid). New to the Folsom release is support for Zero MQ.

The SQL database stores most of the build-time and runtime state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects. Theoretically, OpenStack Nova can support any database supported by SQL-Alchemy but the only databases currently being widely used are SQLite3 (only appropriate for test and development work), MySQL and PostgreSQL.

Nova also provides console services to allow end users to access their virtual instance's console through a proxy. This involves several daemons (nova-console, nova- novncproxy and nova-consoleauth).

Nova interacts with many other OpenStack services: Keystone for authentication, Glance for images and Horizon for web interface. The Glance interactions are central. The API process can upload and query Glance while nova-compute will download images for use in launching images.



OpenStack Block Storage (Cinder)

Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs. In addition to local Linux server storage, it can use storage platforms including Ceph, CloudByte, Coraid, EMC (VMAX and VNX), GlusterFS, IBM Storage (Storwize family, SAN Volume Controller, and XIV Storage System), Linux LIO, NetApp, Nexenta, Scality, SolidFire and HP (Store Virtual and StoreServ 3Par families). Block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems, or providing a server with access to raw block level storage. Snapshot management provides powerful functionality for backing up data stored on block storage volumes. Snapshots can be restored or used to create a new block storage volume.

A few points on OpenStack Block Storage:

OpenStack provides persistent block level storage devices for use with OpenStack compute instances.

The block storage system manages the creation, attaching and detaching of the block devices to servers. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs.

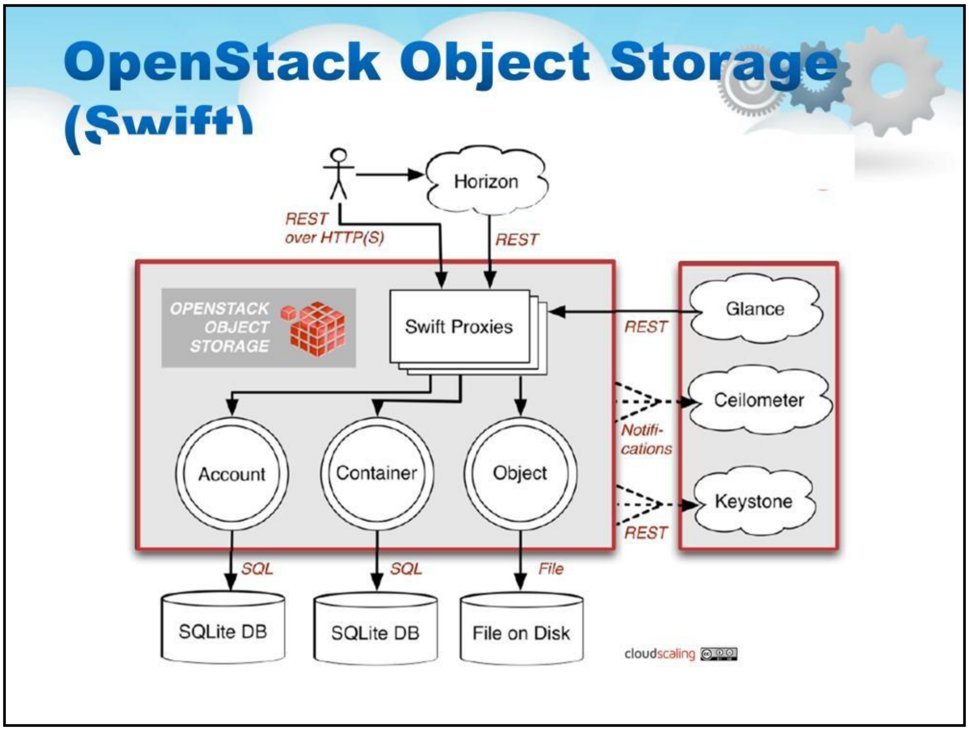
In addition to using simple Linux server storage, it has unified storage support for numerous storage platforms including Ceph, NetApp, Nexenta, SolidFire, and Zadara.

Block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems, or providing a server with access to raw block level storage.

Snapshot management provides powerful functionality for backing up data stored on block storage volumes. Snapshots can be restored or used to create a new block storage volume.

Cinder separates out the persistent block storage functionality that was previously part of OpenStack Compute (in the form of nova-volume) into its own service. The OpenStack Block Storage API allows for manipulation of volumes, volume types (similar to compute flavors) and volume snapshots.
cinder-api accepts API requests and routes them to cinder-volume for action.

cinder-volume acts upon the requests by reading or writing to the Cinder database to maintain state, interacting with other processes (like cinder-scheduler) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture. Currently, there are drivers for IBM, SolidFire, NetApp, Nexenta, Zadara, linux iSCSI and other storage providers.
Much like nova-scheduler, the cinder-scheduler daemon picks the optimal block storage provider node to create the volume on.
Cinder deployments will also make use of a messaging queue to route information between the cinder processes as well as a database to store volume state.
Like Neutron, Cinder will mainly interact with Nova, providing volumes for its instances.



OpenStack Object Storage (Swift)

OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used.

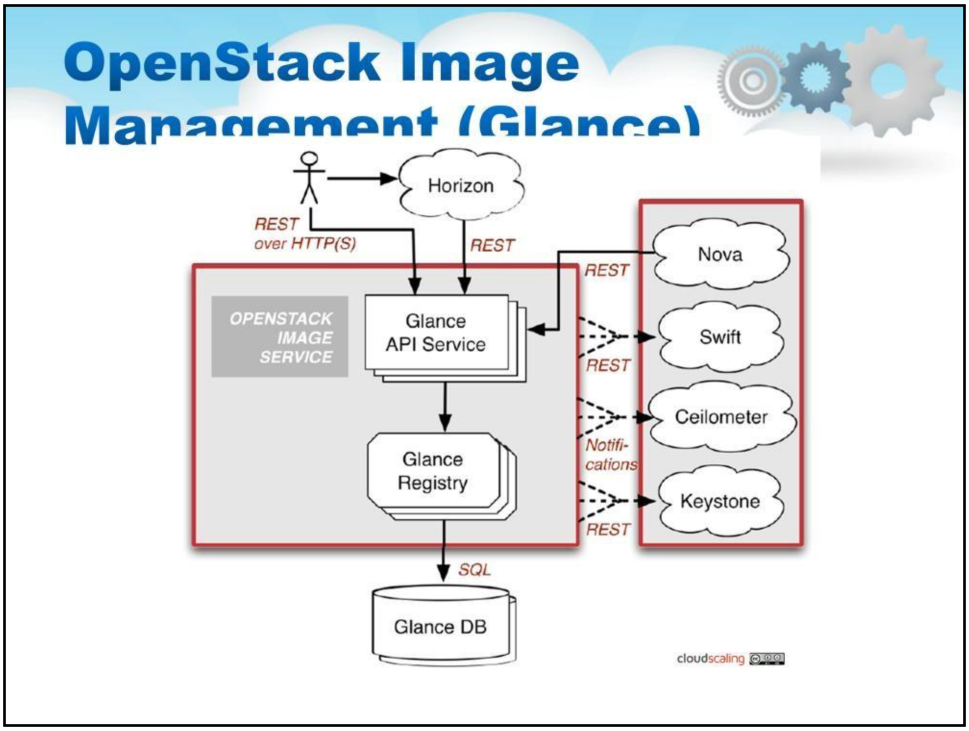
Object Storage is ideal for cost effective, scale-out storage. It provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving and data retention. Block Storage allows block devices to be exposed and connected to compute instances for expanded storage, better performance and integration with enterprise storage platforms, such as NetApp, Nexenta and SolidFire.
A few details on OpenStack's Object Storage
OpenStack provides redundant, scalable object storage using clusters of standardized servers capable of storing petabytes of data

Object Storage is not a traditional file system, but rather a distributed storage system for static data such as virtual machine images, photo storage, email storage, backups and archives. Having no central "brain" or master point of control provides greater scalability, redundancy and durability.

Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster.

Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.
The swift architecture is very distributed to prevent any single point of failure as well as to scale horizontally.
It includes the following components:

Proxy server (swift-proxy-server) accepts incoming requests via the OpenStack Object API or just raw HTTP. It accepts files to upload, modifications to metadata or container creation. In addition, it will also serve files or container listing to web browsers. The proxy server may utilize an optional cache (usually deployed with memcache) to improve performance.
Account servers manage accounts defined with the object storage service.
Container servers manage a mapping of containers (i.e folders) within the object store service.
Object servers manage actual objects (i.e. files) on the storage nodes.
There are also a number of periodic processes which run to perform housekeeping tasks on the large data store. The most important of these is the replication services, which ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters and reapers.
Authentication is handled through configurable WSGI middleware (which will usually be Keystone).



OpenStack Image Management (Glance)

OpenStack Image Service (Glance) provides discovery, registration and delivery services for disk and server images. Stored images can be used as a template. They can also be used to store and catalog an unlimited number of backups. The Image Service can store disk and server images in a variety of back-ends, including OpenStack Object Storage. The Image Service API provides a standard REST interface for querying information about disk images and lets clients stream the images to new servers.

Capabilities of the Image Service include:

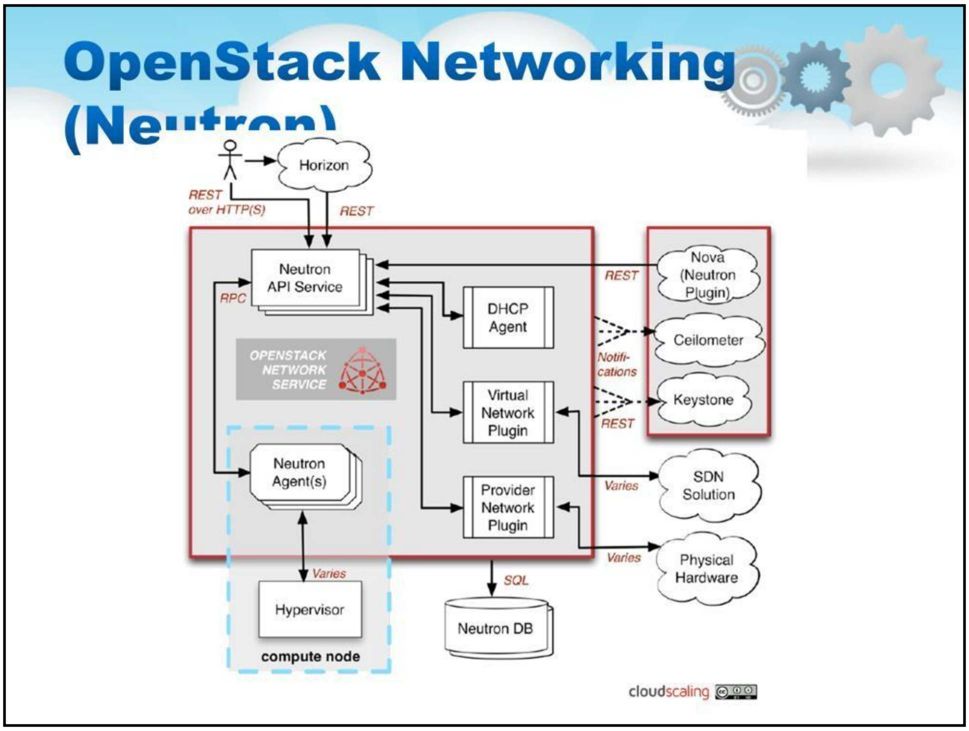
- Administrators can create base templates from which their users can start new compute instances Users can choose from available images, or create their own from existing servers
- Snapshots can also be stored in the Image Service so that virtual machines can be backed up quickly
- A multi-format image registry, the image service allows uploads of private and public images in a variety of formats, including:
  - ✓ Raw
  - ✓ Machine (kernel/ramdisk outside of image, also known as AMI) VHD (Hyper-V)
  - ✓ VDI (VirtualBox) qcow2 (Qemu/KVM) VMDK (VMWare)
  - ✓ OVF (VMWare, others)

The Glance architecture has stayed relatively stable since the Cactus release. The biggest architectural change has been the addition of authentication, which was added in the Diablo release. Just as a quick reminder, Glance has four main parts to it:

1) glance-api accepts Image API calls for image discovery, image retrieval and image storage.
2) glance-registry stores, processes and retrieves metadata about images (size, type, etc.).
3) A database to store the image metadata. Like Nova, you can choose your database depending on your preference (but most people use MySQL or SQLite).
4) A storage repository for the actual image files. In the diagram above, Swift is shown as the image repository, but this is configurable. In addition to Swift, Glance supports normal filesystems, RADOS block devices, Amazon S3 and HTTP. Be aware that some of these choices are limited to read-only usage.

There are also a number of periodic processes which run on Glance to support caching. The most important of these is the replication services, which ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters and reapers.

As you can see from the diagram in the Conceptual Architecture section, Glance serves a central role to the overall IaaS picture. It accepts API requests for images (or image metadata) from end users or Nova components and can store its disk files in the object storage service, Swift.



OpenStack Networking (Neutron)

OpenStack Networking (Neutron, formerly Quantum) is a pluggable, scalable and API-driven system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

OpenStack Networking is a system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

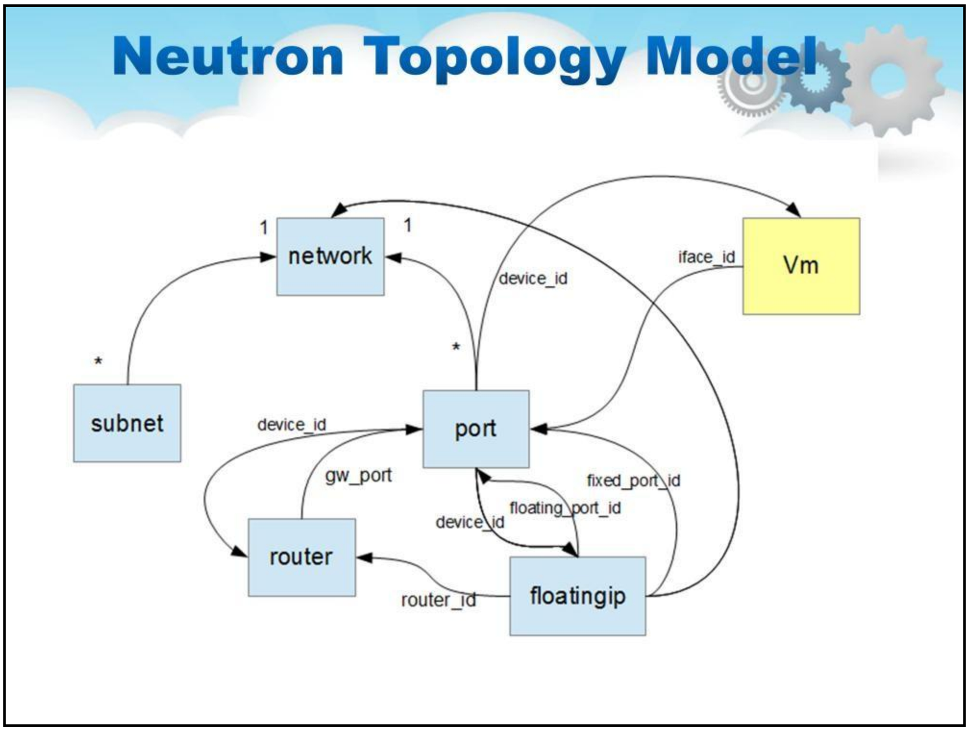
OpenStack Neutron provides networking models for different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic. OpenStack Networking manages IP addresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically re routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure. Users can create their own networks, control traffic and connect servers and devices to one or more networks. Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale. OpenStack Networking has an extension framework allowing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.
Networking Capabilities

OpenStack provides flexible networking models to suit the needs of different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic.

OpenStack Networking manages IP addresses, allowing for dedicated static IPs or DHCP.
Floating IPs allow traffic to be dynamically re-routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure.
Users can create their own networks, control traffic and connect servers and devices to one or more networks.
The pluggable backend architecture lets users take advantage of commodity gear or advanced networking services from supported vendors.

Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale.

OpenStack Networking has an extension framework allowing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.

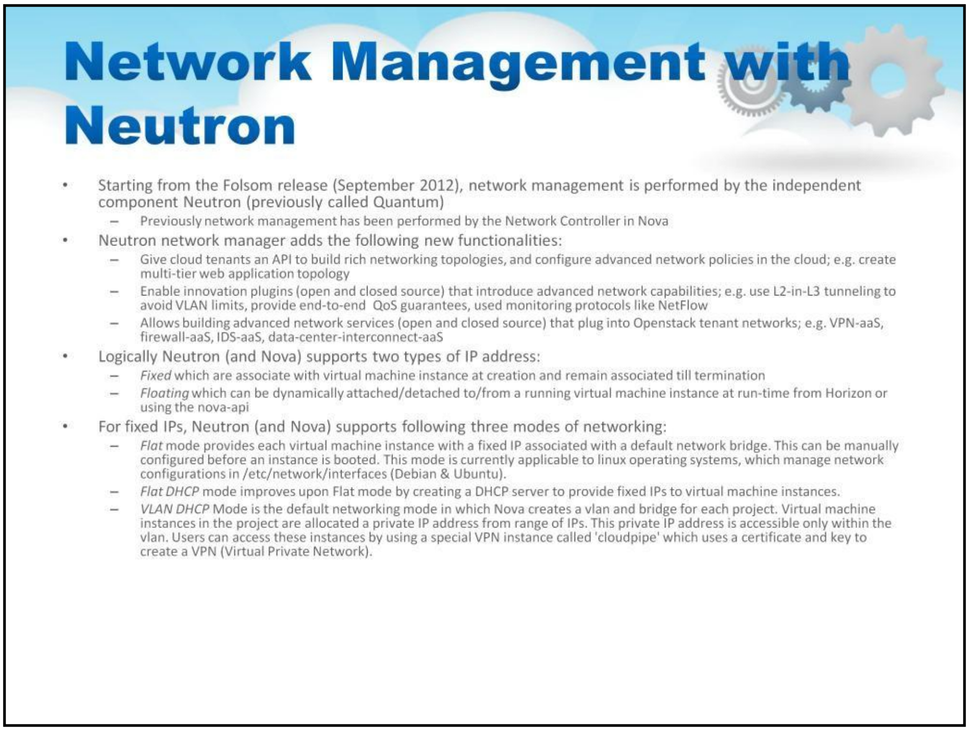


Neutron Topology Model

Neutron provides "network connectivity as a service" between interface devices managed by othe neutron-server accepts API requests and then routes them to the appropriate Neutron plug-in for Neutron plug-ins and agents perform the actual actions such as plugging and unplugging ports, cre
The common agents are L3 (layer 3), DHCP (dynamic host IP addressing) and the specific plug-in ag

Most Neutron installations will also make use of a messaging queue to route information between

Neutron will interact mainly with Nova, where it will provide networks and connectivity for its insta



Network Management with Neutron

- Starting from the Folsom release (September 2012), network management is performed by the independent component Neutron (previously called Quantum)
  - Previously network management has been performed by the Network Controller in Nova
- Neutron network manager adds the following new functionalities:
  - Give cloud tenants an API to build rich networking topologies, and configure advanced network policies in the cloud; e.g. create multi-tier web application topology
  - Enable innovation plugins (open and closed source) that introduce advanced network capabilities; e.g. use L2-in-L3 tunneling to avoid VLAN limits, provide end-to-end QoS guarantees, used monitoring protocols like NetFlow
  - Allows building advanced network services (open and closed source) that plug into Openstack tenant networks; e.g. VPN-aaS, firewall-aaS, IDS-aaS, data-center-interconnect-aaS
- Logically Neutron (and Nova) supports two types of IP address:
  - *Fixed* which are associate with virtual machine instance at creation and remain associated till termination
  - *Floating* which can be dynamically attached/detached to/from a running virtual machine instance at run-time from Horizon or using the nova-api
- For fixed IPs, Neutron (and Nova) supports following three modes of networking:
  - *Flat* mode provides each virtual machine instance with a fixed IP associated with a default network bridge. This can be manually configured before an instance is booted. This mode is currently applicable to linux operating systems, which manage network configurations in /etc/network/interfaces (Debian & Ubuntu).
  - *Flat DHCP* mode improves upon Flat mode by creating a DHCP server to provide fixed IPs to virtual machine instances.
  - *VLAN DHCP* Mode is the default networking mode in which Nova creates a vlan and bridge for each project. Virtual machine instances in the project are allocated a private IP address from range of IPs. This private IP address is accessible only within the vlan. Users can access these instances by using a special VPN instance called 'cloudpipe' which uses a certificate and key to create a VPN (Virtual Private Network).

Starting from the Folsom release (September 2012), network management is performed by the independent component Neutron (previously called Quantum)

Previously network management has been performed by the Network Controller in Nova

Neutron network manager adds the following new functionalities:

Give cloud tenants an API to build rich networking topologies, and configure advanced network policies in the cloud; e.g. create multi-tier web application topology

Enable innovation plugins (open and closed source) that introduce advanced network capabilities; e.g. use L2-in-L3 tunneling to avoid VLAN limits, provide end-to-end QoS guarantees, used monitoring protocols like NetFlow

Allows building advanced network services (open and closed source) that plug into Openstack tenant networks; e.g. VPN-aaS, firewall-aaS, IDS-aaS, data-center-interconnect-aaS

Logically Neutron (and Nova) supports two types of IP address:

Fixed which are associate with virtual machine instance at creation and remain associated till termination

Floating which can be dynamically attached/detached to/from a running virtual machine instance at run-time from Horizon or using the nova-api

For fixed IPs, Neutron (and Nova) supports following three modes of networking:

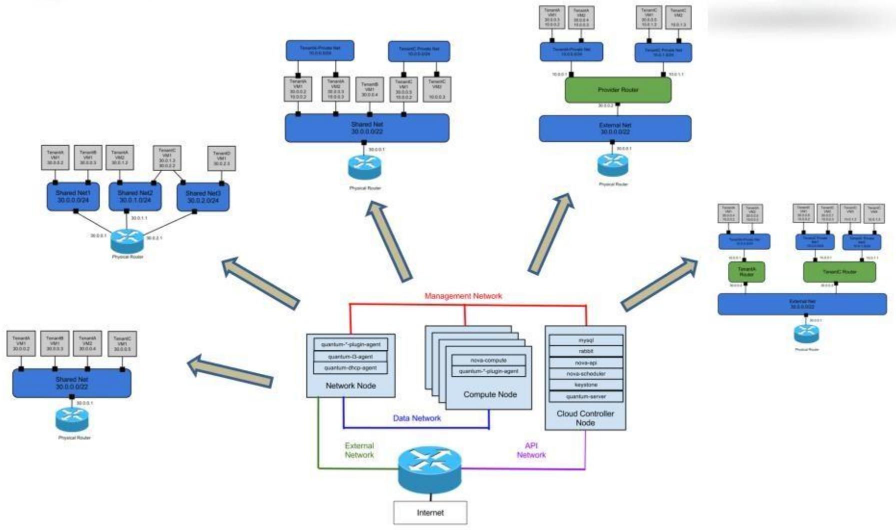
Flat mode provides each virtual machine instance with a fixed IP associated with a default network bridge. This can be manually configured before an instance is booted. This mode is currently applicable to linux operating systems, which manage network configurations in

/etc/network/interfaces (Debian & Ubuntu).

Flat DHCP mode improves upon Flat mode by creating a DHCP server to provide fixed IPs to virtual machine instances.

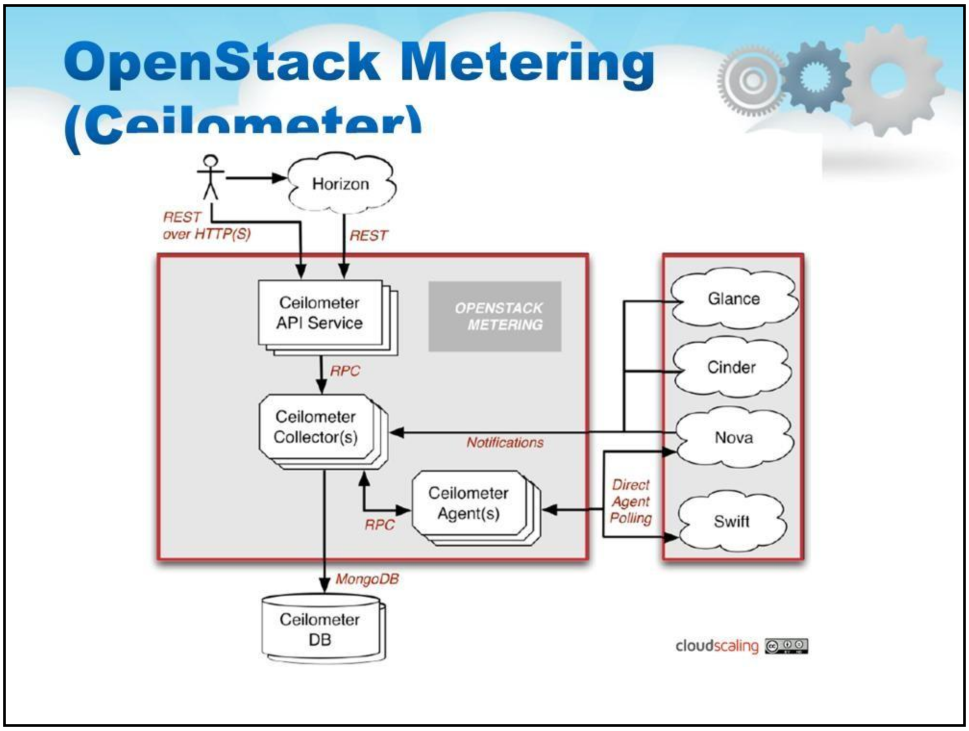
VLAN DHCP Mode is the default networking mode in which Nova creates a vlan and bridge for each project. Virtual machine instances in the project are allocated a private IP address from range of IPs. This private IP address is accessible only within the vlan. Users can access these instances by using a special VPN instance called 'cloudpipe' which uses a certificate and key to create a VPN (Virtual Private Network).

Neutron Networking Example



This slide illustrates how Neutron based software networking can be applied to result is many different networking topologies for the user

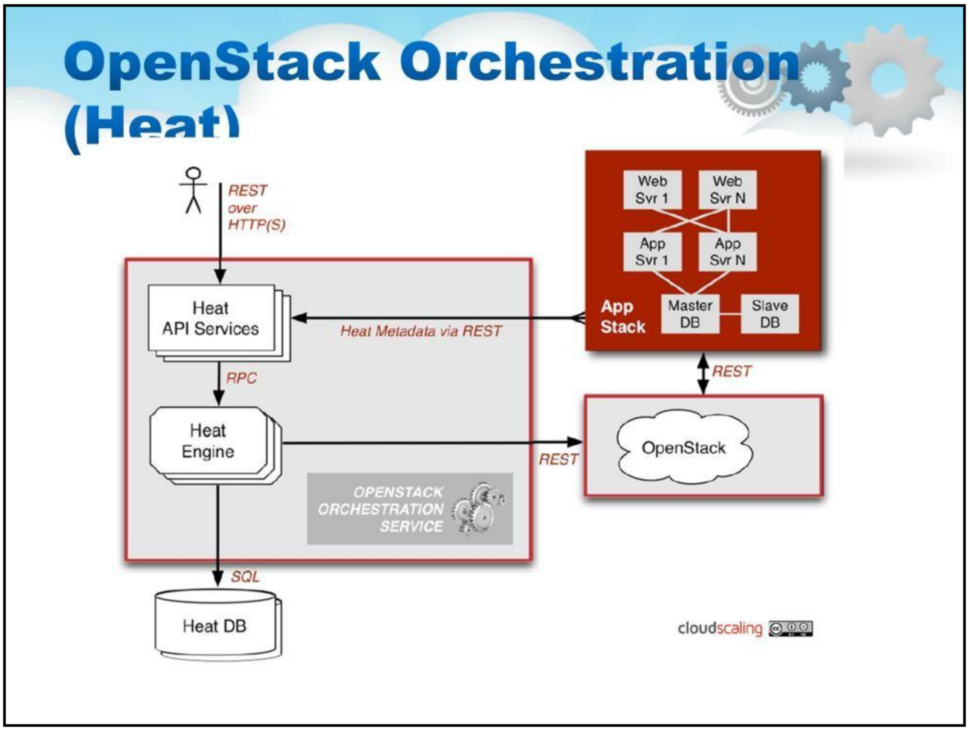
Note the ability to create virtual switch components as well as multiple network segments.



OpenStack Metering (Ceilometer)

The Ceilometer project aims to become the infrastructure to collect measurements within OpenStack so that no two agents would need to be written to collect the same data. Its primary targets are monitoring and metering, but the framework should be easily expandable to collect for other needs. To that effect, Ceilometer should be able to share collected data with a variety of consumers.

An agent runs on each OpenStack node (Bare Metal machine) and harvests the data locally
If a meter is available from the existing OpenStack component it should be used
A standalone ceilometer agent implements the meters that are not yet available from the existing OpenStack components
A storage daemon communicates with the agents to collect their data and aggregate them
The agents collecting data are authenticated to avoid pollution of the metering service
The data is sent from agents to the storage daemon via a trusted messaging system (RabbitMQ?)
The data / messages exchanged between agents and the storage daemon use a common messages format
The content of the storage is made available thru a REST API providing aggregation
The message queue is separate from other queues (such as the nova queue)
The messages in queue are signed and non repudiable



OpenStack Orchestration (Heat)

Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat also endeavours to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack.

Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API.

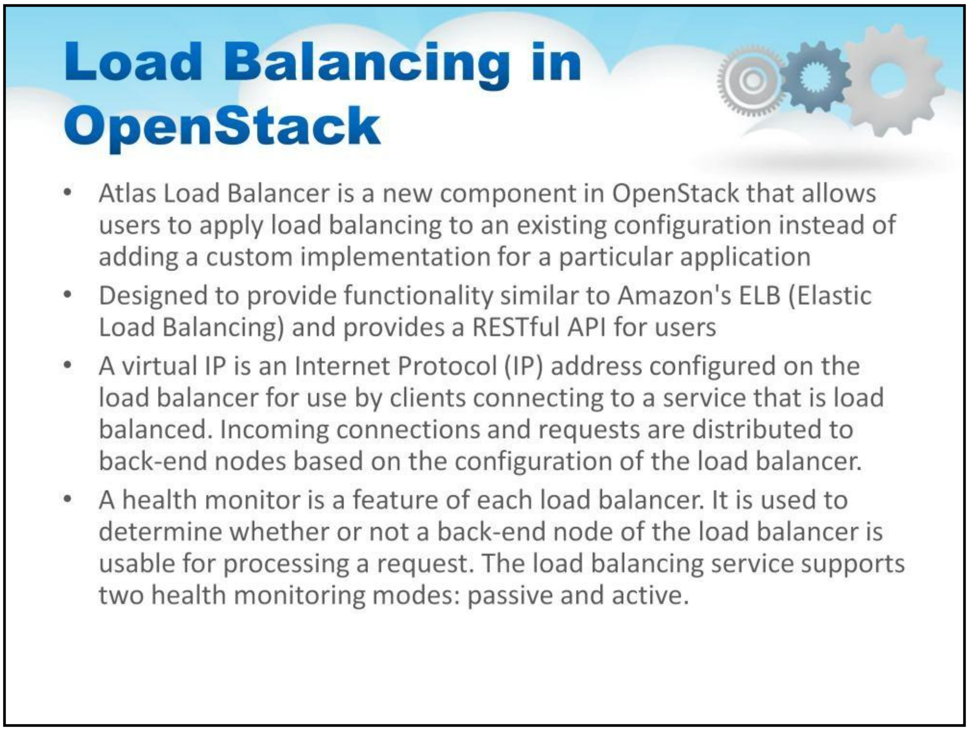
A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, &c. Infrastructure resources that can be described include: servers, floating ips, volumes, security groups, users, etc.

Heat also provides an autoscaling service that integrates with Ceilometer, so you can include a scaling group as a resource in a template.

Templates can also specify the relationships between resources (e.g. this volume is connected to this server). This enables Heat to call out to the OpenStack APIs to create all of your infrastructure in the correct order to completely launch your application.

Heat manages the whole lifecycle of the application - when you need to change your infrastructure, simply modify the template and use it to update your existing stack. Heat knows how to make the necessary changes. It will delete all of the resources when you are finished with the application, too.

Heat primarily manages infrastructure, but the templates integrate well with software configuration management tools such as Puppet and Chef. The Heat team is working on providing even better integration between infrastructure and software.



Load Balancing in OpenStack

- Atlas Load Balancer is a new component in OpenStack that allows users to apply load balancing to an existing configuration instead of adding a custom implementation for a particular application
- Designed to provide functionality similar to Amazon's ELB (Elastic Load Balancing) and provides a RESTful API for users
- A virtual IP is an Internet Protocol (IP) address configured on the load balancer for use by clients connecting to a service that is load balanced. Incoming connections and requests are distributed to back-end nodes based on the configuration of the load balancer.
- A health monitor is a feature of each load balancer. It is used to determine whether or not a back-end node of the load balancer is usable for processing a request. The load balancing service supports two health monitoring modes: passive and active.

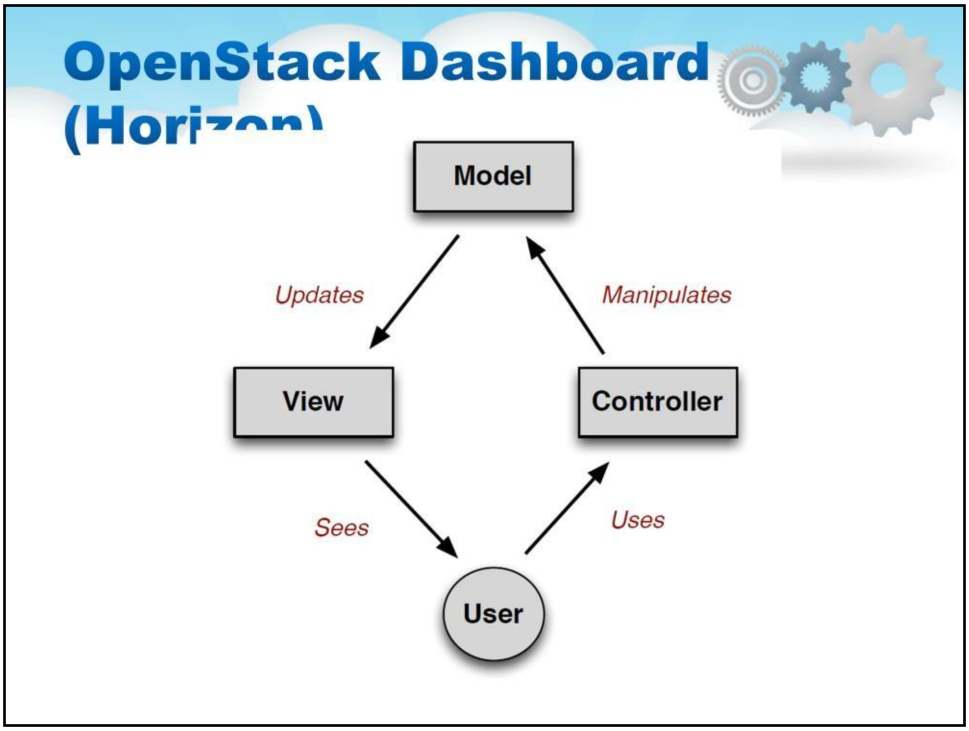
A load balancer is a logical device. It is used to distribute workloads between multiple back-end systems or services called nodes, based on the criteria defined as part of its configuration.

Atlas Load Balancer is a new component in OpenStack that allows users to apply load balancing to an existing configuration instead of adding a custom implementation for a particular application

Designed to provide functionality similar to Amazon's ELB (Elastic Load Balancing) and provides a RESTful API for users

A virtual IP is an Internet Protocol (IP) address configured on the load balancer for use by clients connecting to a service that is load balanced. Incoming connections and requests are distributed to back-end nodes based on the configuration of the load balancer.

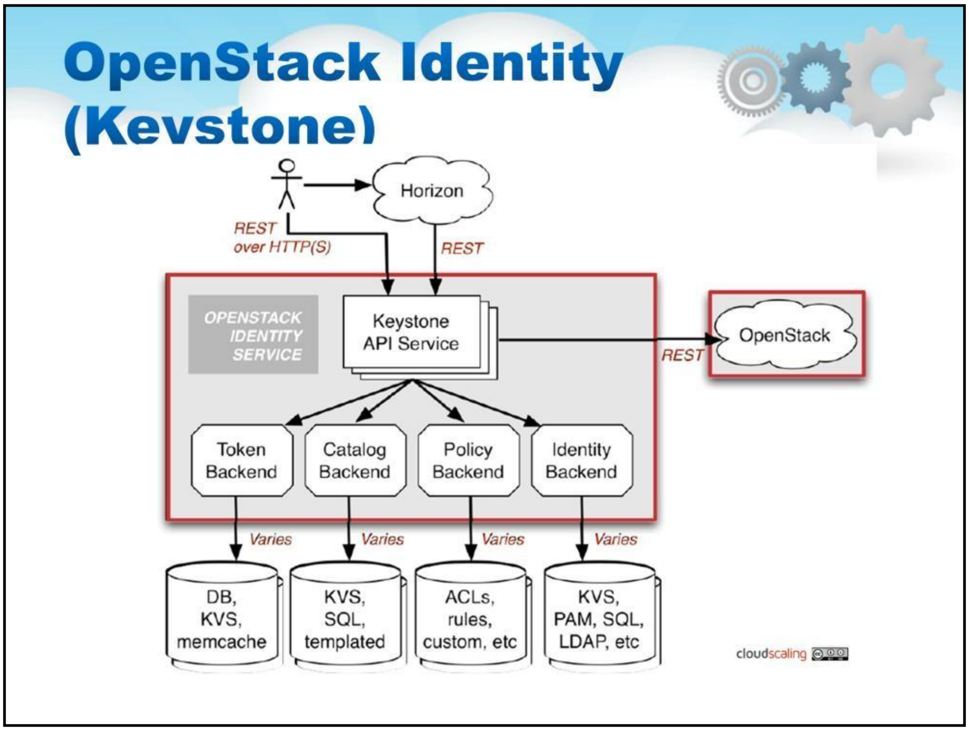
A health monitor is a feature of each load balancer. It is used to determine whether or not a back-end node of the load balancer is usable for processing a request. The load balancing service supports two health monitoring modes: passive and active.



OpenStack Dashboard (Horizon)

The design allows for third party products and services, such as billing, monitoring and additional management tools. Service providers and other commercial vendors can customize the dashboard with their own brand.

The dashboard is just one way to interact with OpenStack resources. Developers can automate access or build tools to manage their resources using the native OpenStack API or the EC2 compatibility API.



OpenStack Identity (Keystone)

OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems, and Amazon Web Services log in credentials such as those used for EC2.

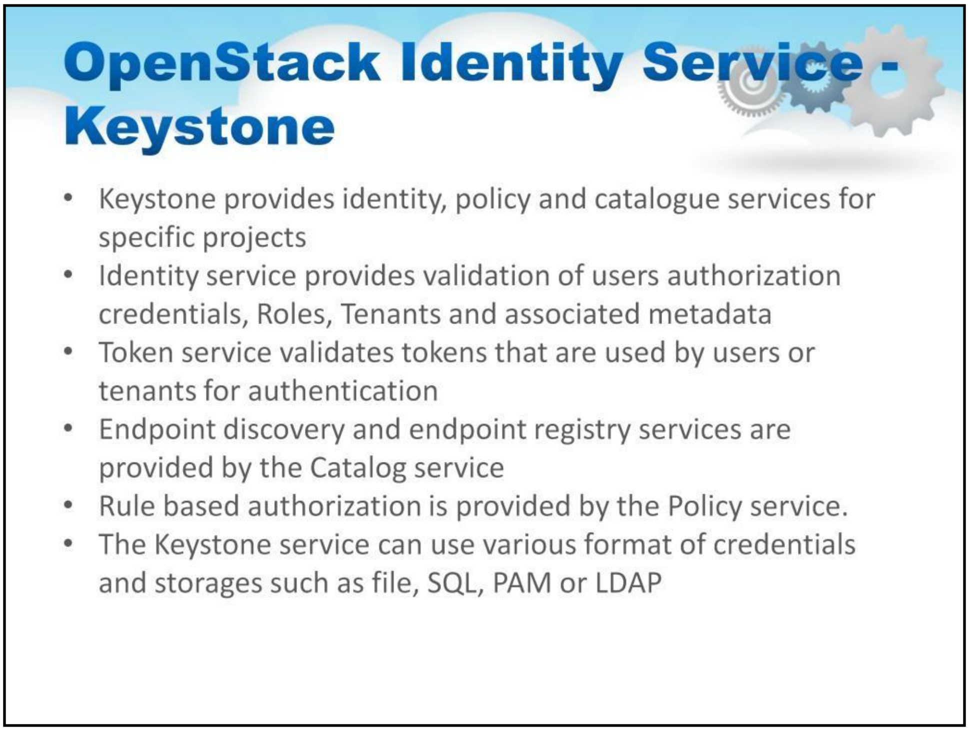
Additionally, the catalog provides a query-able list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can programmatically determine which resources they can access.

The OpenStack Identity Service enables administrators to:

Configure centralized policies across users and systems
Create users and tenants and define permissions for compute, storage, and networking resources by using role-based access control (RBAC) features
Integrate with an existing directory, like LDAP, to provide a single source of authentication across the enterprise

The OpenStack Identity Service enables users to:

List the services to which they have access
Make API requests
Log into the web dashboard to create resources owned by their account



OpenStack Identity Service - Keystone

- Keystone provides identity, policy and catalogue services for specific projects
- Identity service provides validation of users authorization credentials, Roles, Tenants and associated metadata
- Token service validates tokens that are used by users or tenants for authentication
- Endpoint discovery and endpoint registry services are provided by the Catalog service
- Rule based authorization is provided by the Policy service.
- The Keystone service can use various format of credentials and storages such as file, SQL, PAM or LDAP

Keystone provides a single point of integration for OpenStack policy, catalog, token and authentication.

Keystone handles API requests as well as providing configurable catalog, policy, token and identity services.

Identity service provides validation of users authorization credentials, Roles, Tenants and associated metadata

Token service validates tokens that are used by users or tenants for authentication

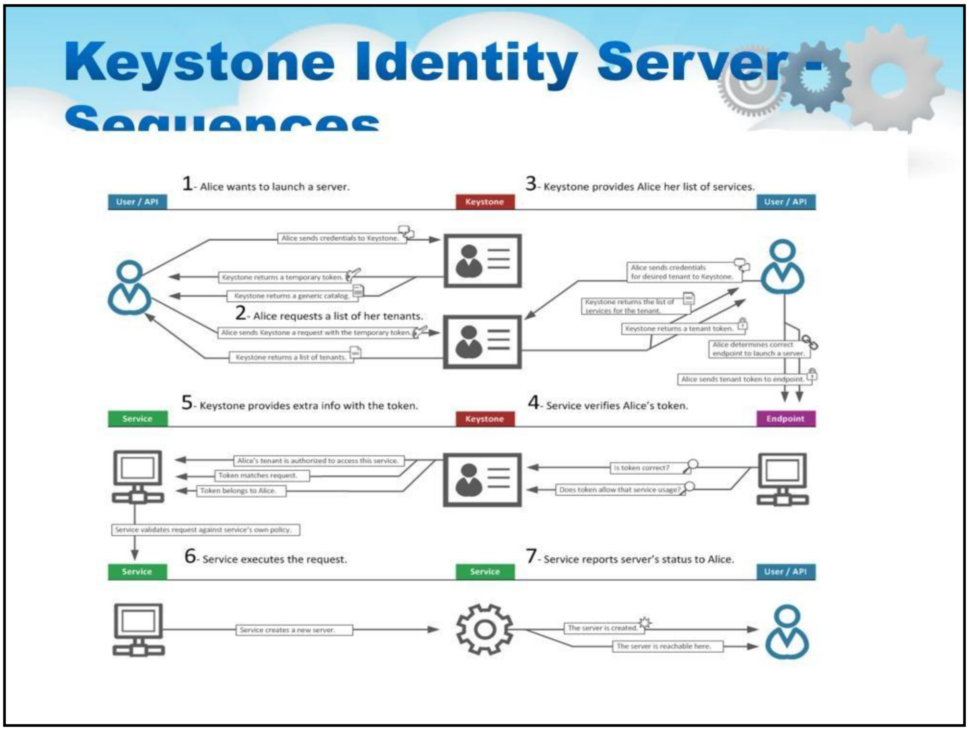
Endpoint discovery and endpoint registry services are provided by the Catalog service

Rule based authorization is provided by the Policy service.

The Keystone service can use various format of credentials and storages such as file, SQL, PAM or LDAP

Each Keystone function has a pluggable backend which allows different ways to use the particular service. Most support standard backends like LDAP or SQL, as well as Key Value Stores (KVS).

Most people will use this as a point of customization for their current authentication services.



Keystone Identity Server - Sequences

1- Alice wants to launch a server.

Alice sends credentials to Keystone.

Keystone returns a temporary token.

Keystone returns a generic catalog.

2- Alice requests a list of her tenants.

Alice sends Keystone a request with the temporary token.

Keystone returns a list of tenants.

3- Keystone provides Alice her list of services.

Alice sends credentials for desired tenant to Keystone.

Keystone returns the list of services for the tenant.

Keystone returns a tenant token.

Alice determines correct endpoint to launch a server.

Alice sends tenant token to endpoint.

4- Service verifies Alice's token.

Is token correct?

Does token allow that service usage?

5- Keystone provides extra info with the token.

Alice's tenant is authorized to access this service.

Token matches request.

Token belongs to Alice.

Service validates request against service's own policy.

6- Service executes the request.

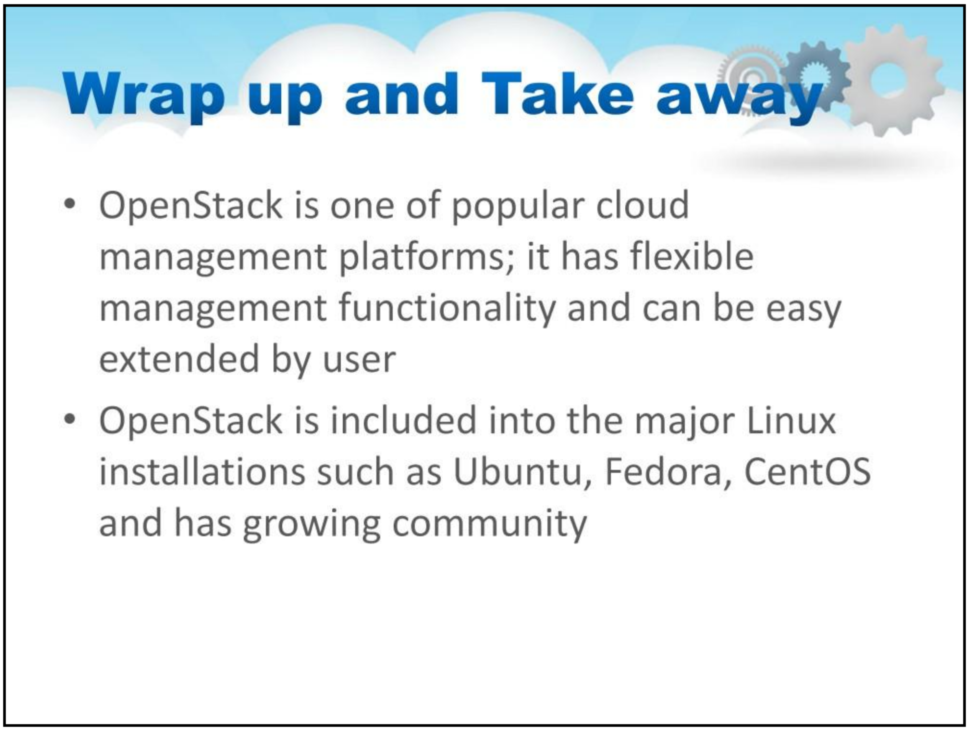
Service creates a new server.

7- Service reports server's status to Alice.

The server is created.

The server is reachable here.

The diagram in this slide shows the Sequences for how Keystone is used.

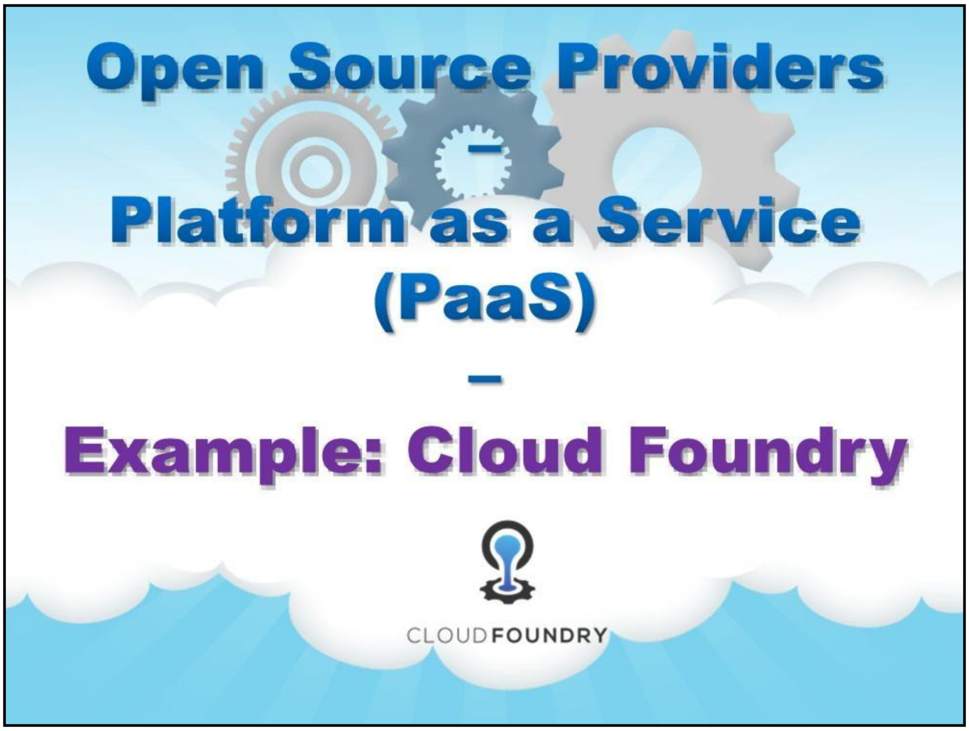


Wrap up and Take away

- OpenStack is one of popular cloud management platforms; it has flexible management functionality and can be easy extended by user
- OpenStack is included into the major Linux installations such as Ubuntu, Fedora, CentOS and has growing community

OpenStack is one of popular cloud management platforms; it has flexible management functionality and can be easy extended by user

OpenStack is included into the major Linux installations such as Ubuntu, Fedora, CentOS and has growing community



Open Source Providers

–

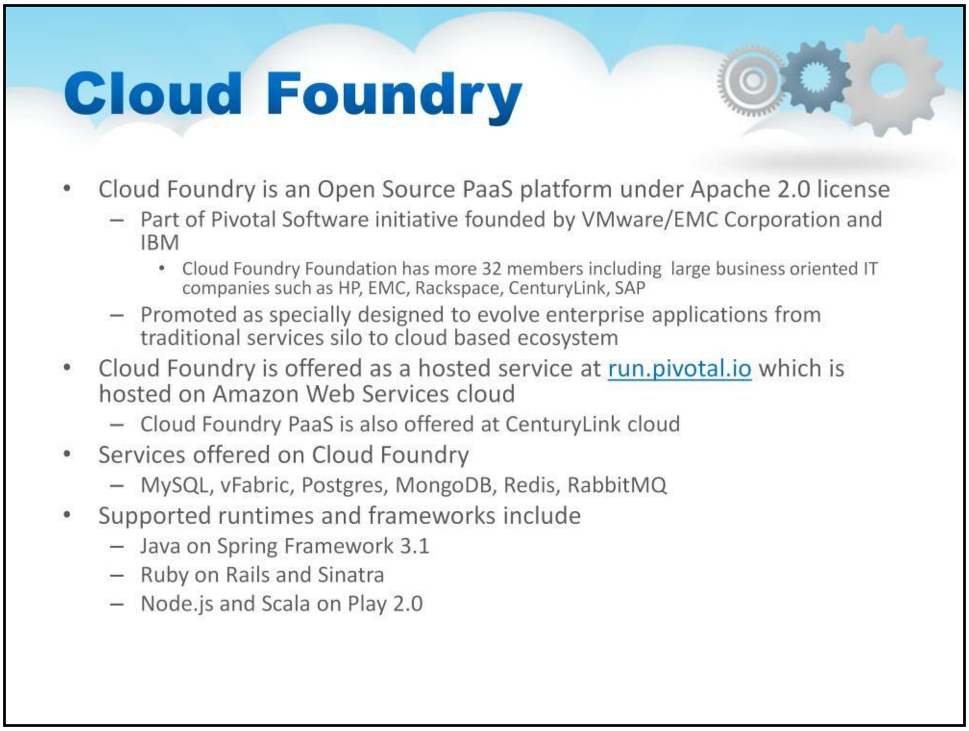
Platform as a Service (PaaS)

–

Example: Cloud Foundry

CLOUDFOUNDRY

Example: Cloud Fondry



Cloud Foundry

- Cloud Foundry is an Open Source PaaS platform under Apache 2.0 license
  - Part of Pivotal Software initiative founded by VMware/EMC Corporation and IBM
    - Cloud Foundry Foundation has more 32 members including large business oriented IT companies such as HP, EMC, Rackspace, CenturyLink, SAP
  - Promoted as specially designed to evolve enterprise applications from traditional services silo to cloud based ecosystem
- Cloud Foundry is offered as a hosted service at run.pivotal.io which is hosted on Amazon Web Services cloud
  - Cloud Foundry PaaS is also offered at CenturyLink cloud
- Services offered on Cloud Foundry
  - MySQL, vFabric, Postgres, MongoDB, Redis, RabbitMQ
- Supported runtimes and frameworks include
  - Java on Spring Framework 3.1
  - Ruby on Rails and Sinatra
  - Node.js and Scala on Play 2.0

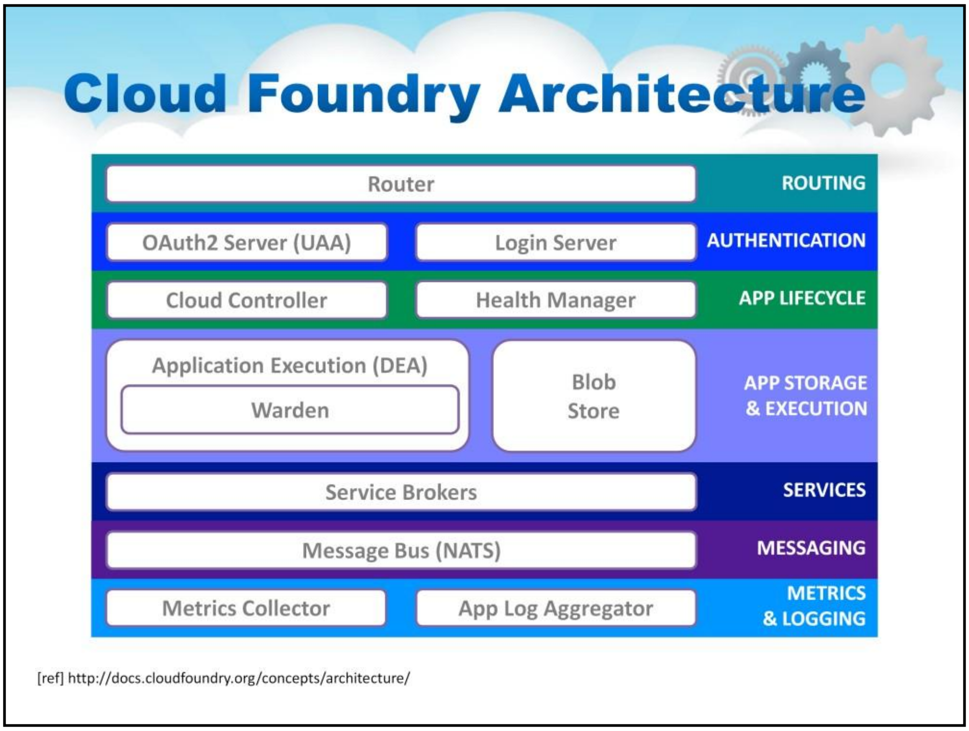
Cloud Foundry is an Open Source PaaS platform under Apache 2.0 license, Apache 2.0 is the least restrictive license. Developed code can be used for both commercial and non-commercial purposes. Part of Pivotal Software initiative founded by VMware/EMC Corporation and IBM Cloud Foundry Foundation has more 32 members including large business oriented IT companies such as HP, EMC, Rackspace, CenturyLink, SAP
Promoted as specially designed to evolve enterprise applications from traditional services silo to cloud based ecosystem

Services offered on Cloud Foundry:

- ✓ MySQL - The open source relational database
- ✓ Postgres - Relational database based on PostgreSQL
- ✓ MongoDB - scalable, open, document-based database
- ✓ Redis - open key-value data structure server
- ✓ RabbitMQ - Reliable, scalable, and portable messaging for applications

Supported runtimes and frameworks include:

- ✓ Java on Spring Framework 3.1
- ✓ Ruby on Rails and Sinatra
- ✓ Node.js and Scala on Play 2.0



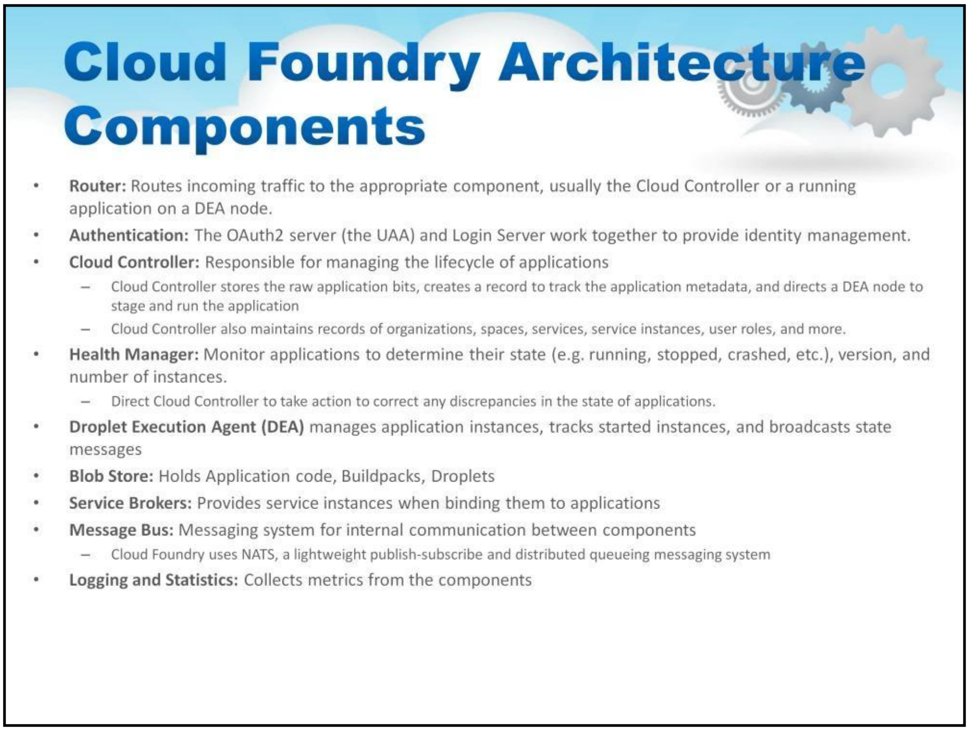
Cloud Foundry Architecture

Router		ROUTING
OAuth2 Server (UAA)	Login Server	AUTHENTICATION
Cloud Controller	Health Manager	APP LIFECYCLE
Application Execution (DEA) Warden	Blob Store	APP STORAGE & EXECUTION
Service Brokers		SERVICES
Message Bus (NATS)		MESSAGING
Metrics Collector	App Log Aggregator	METRICS & LOGGING

[ref] http://docs.cloudfoundry.org/concepts/architecture/

This slide shows an illustration of the “Marketechture” of Cloud Foundry.

The next slide will over each area specifically.



Cloud Foundry Architecture Components

- **Router:** Routes incoming traffic to the appropriate component, usually the Cloud Controller or a running application on a DEA node.
- **Authentication:** The OAuth2 server (the UAA) and Login Server work together to provide identity management.
- **Cloud Controller:** Responsible for managing the lifecycle of applications
  - Cloud Controller stores the raw application bits, creates a record to track the application metadata, and directs a DEA node to stage and run the application
  - Cloud Controller also maintains records of organizations, spaces, services, service instances, user roles, and more.
- **Health Manager:** Monitor applications to determine their state (e.g. running, stopped, crashed, etc.), version, and number of instances.
  - Direct Cloud Controller to take action to correct any discrepancies in the state of applications.
- **Droplet Execution Agent (DEA)** manages application instances, tracks started instances, and broadcasts state messages
- **Blob Store:** Holds Application code, Buildpacks, Droplets
- **Service Brokers:** Provides service instances when binding them to applications
- **Message Bus:** Messaging system for internal communication between components
  - Cloud Foundry uses NATS, a lightweight publish-subscribe and distributed queueing messaging system
- **Logging and Statistics:** Collects metrics from the components

Router: Routes incoming traffic to the appropriate component, usually the Cloud Controller or a running application on a DEA node.

Authentication: The OAuth2 server (the UAA) and Login Server work together to provide identity management.

Cloud Controller: Responsible for managing the lifecycle of applications

Cloud Controller stores the raw application bits, creates a record to track the application metadata, and directs a DEA node to stage and run the application

Cloud Controller also maintains records of organizations, spaces, services, service instances, user roles, and more.

Health Manager: Monitor applications to determine their state (e.g. running, stopped, crashed, etc.), version, and number of instances.

Direct Cloud Controller to take action to correct any discrepancies in the state of applications.

Droplet Execution Agent (DEA) manages application instances, tracks started instances, and broadcasts state messages.

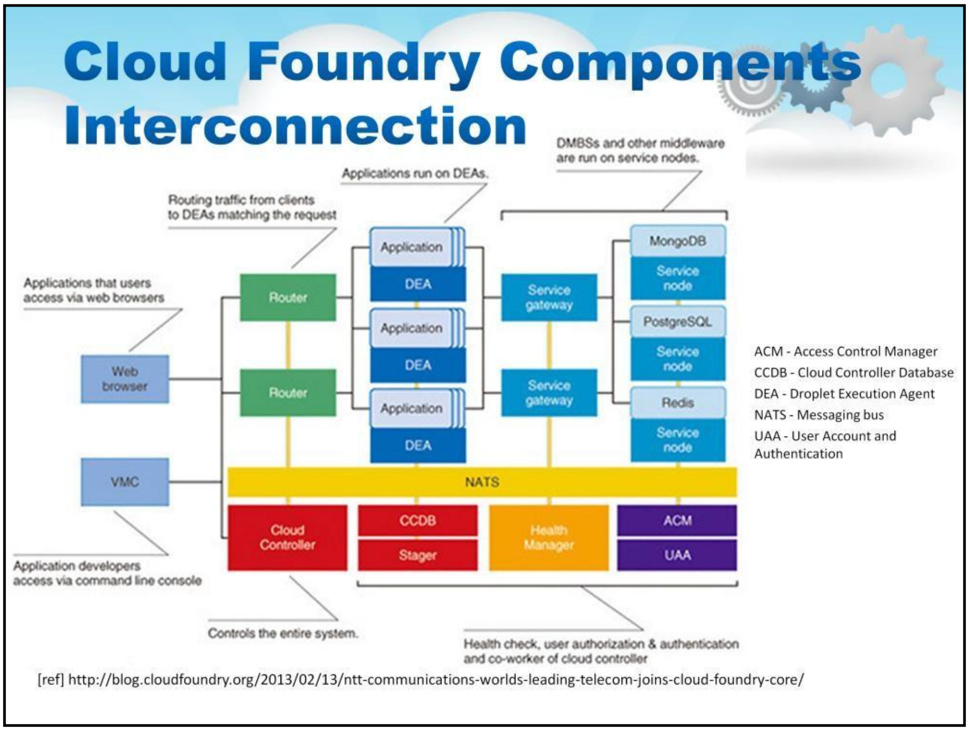
Blob Store: Holds Application code, Buildpacks, Droplets

Service Brokers: Provides service instances when binding them to applications

Message Bus: Messaging system for internal communication between components

Cloud Foundry uses NATS, a lightweight publish-subscribe and distributed queueing messaging system

Logging and Statistics: Collects metrics from the components



Cloud Foundry Components Interconnection

DMBSs and other middleware are run on service nodes.

Applications run on DEAs.

Routing traffic from clients to DEAs matching the request

Applications that users access via web browsers

ACM - Access Control Manager
CCDB - Cloud Controller Database
DEA - Droplet Execution Agent
NATS - Messaging bus
UAA - User Account and Authentication

Application developers access via command line console

Controls the entire system.

Health check, user authorization & authentication and co-worker of cloud controller

[ref] http://blog.cloudfoundry.org/2013/02/13/ntt-communications-worlds-leading-telecom-joins-cloud-foundry-core/

This slides illustrates a more technically accurate structure of Cloud Foundry,

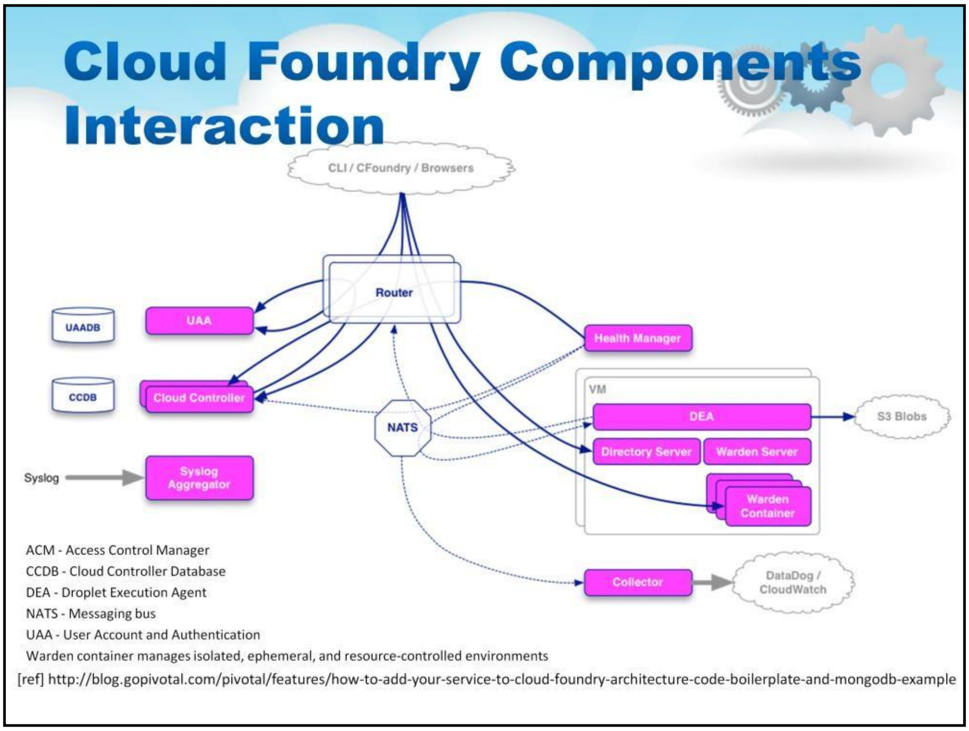
In the center one sees the Droplet Execution Engine, which runs the notion of an "application" in Cloud Foundry.

In the front of these applications are software routers which move traffic from the internet (the browser, or web services) to the Applications.

On the other side, the back end of the applications, as the applications need services such as database or storage, the services gateway connects the Application and DEA to the right type of service.

The message bus NATS orchestrates these different tiers to talk to each other. NATS is driven by the cloud controller which controls the whole system. Other modules include management, authorization and authentication, and other modules typical to an application server environment,

Note the inherently distributed architecture of Cloud Foundry. It is not like the tightly coupled Remote Procedure Call based multi-tier application servers of years past. The idea that all the parts are connected a transactional message queue – the same technique which the clouds use to hold themselves together at the IaaS level – is designed for the distributed environment. Also perfect for clouds, is the replication of the DEAs and therefore the applications. This scales with the sale-out cloud model.



Cloud Foundry Components Interaction

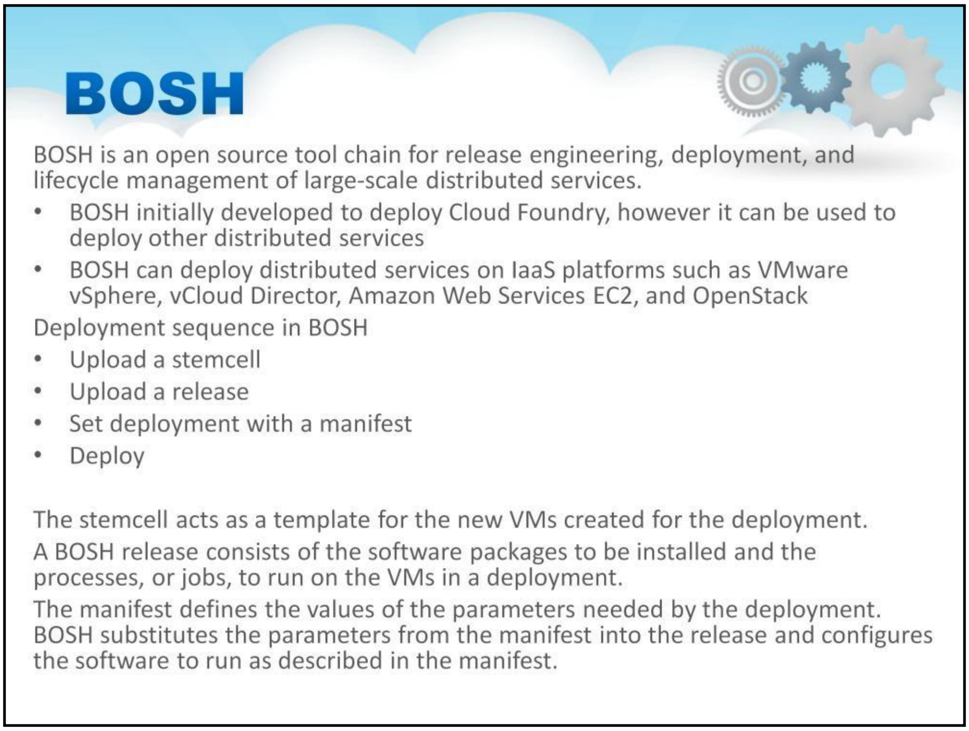
ACM - Access Control Manager
CCDB - Cloud Controller Database
DEA - Droplet Execution Agent
NATS - Messaging bus
UAA - User Account and Authentication
Warden container manages isolated, ephemeral, and resource-controlled environments

[ref] http://blog.gopivotal.com/pivotal/features/how-to-add-your-service-to-cloud-foundry-architecture-code-boilerplate-and-mongodb-example

The illustration in this slide explodes the data flow inside of Cloud Foundry. Solid lines indicate traffic of the application, eg, HTTP. Dotted lines indicate control messages traffic, eg, from NATS.

One can see the central role that the Router plays in application traffic, All application traffic is directed by the Router.

The business end of Cloud Foundry is the VM where the Drople Execution Engine runs, Each VM runs a Droplet Execution Agent and multiple Warden service containers.
The Warden container manages isolated, ephemeral, and resource-controlled environments.



BOSH

BOSH is an open source tool chain for release engineering, deployment, and lifecycle management of large-scale distributed services.

- BOSH initially developed to deploy Cloud Foundry, however it can be used to deploy other distributed services
- BOSH can deploy distributed services on IaaS platforms such as VMware vSphere, vCloud Director, Amazon Web Services EC2, and OpenStack

Deployment sequence in BOSH

- Upload a stemcell
- Upload a release
- Set deployment with a manifest
- Deploy

The stemcell acts as a template for the new VMs created for the deployment.

A BOSH release consists of the software packages to be installed and the processes, or jobs, to run on the VMs in a deployment.

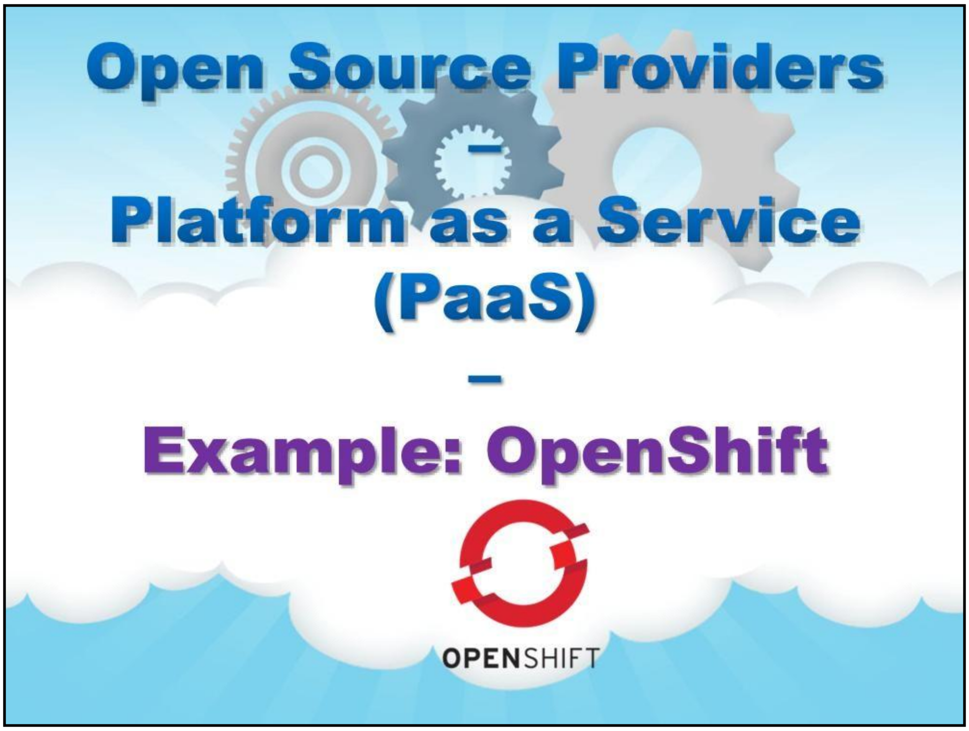
The manifest defines the values of the parameters needed by the deployment. BOSH substitutes the parameters from the manifest into the release and configures the software to run as described in the manifest.

NOSH is a tool which was created to build Cloud Foundry itself, In normal situations, Cloud Foundry application developers will see little if any BOSH.

Developers of services and addons to Cloud Foundry will use BOSH,

While BOSH can be applied to build other application frameworks, being that it is portable to pretty much any IaaS platform, it has not seen any real use outside of Cloud Foundry development

The rest of this slide details how BOSH is used to create and in the runtime of Cloud Foundry.



Open Source Providers

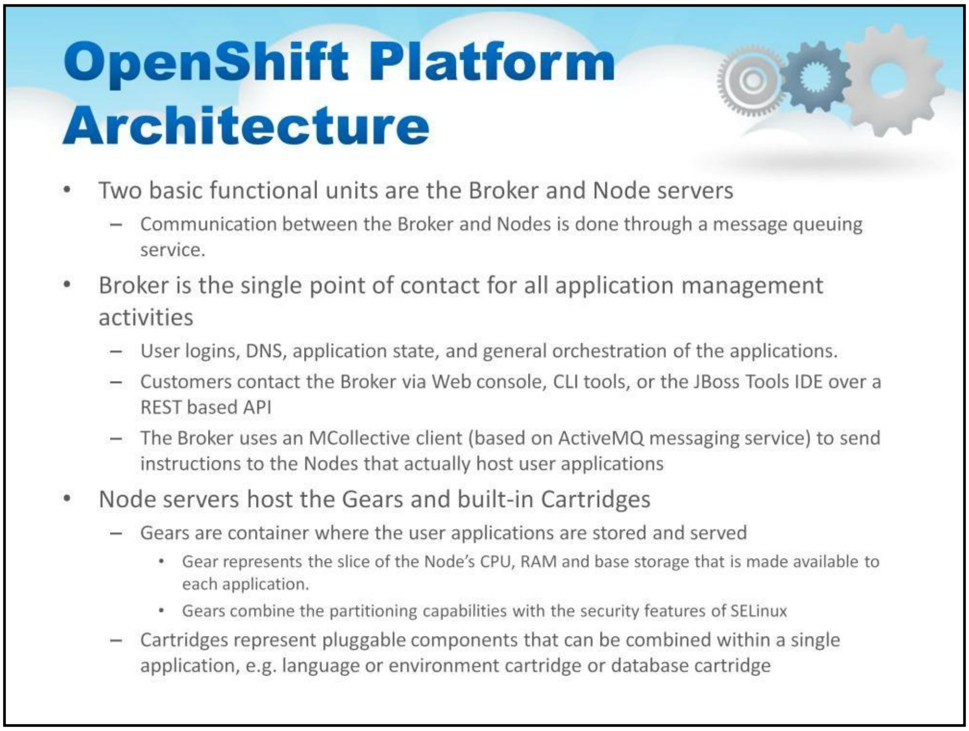
–

Platform as a Service (PaaS)

–

Example: OpenShift

Example: OpenShift



OpenShift Platform Architecture

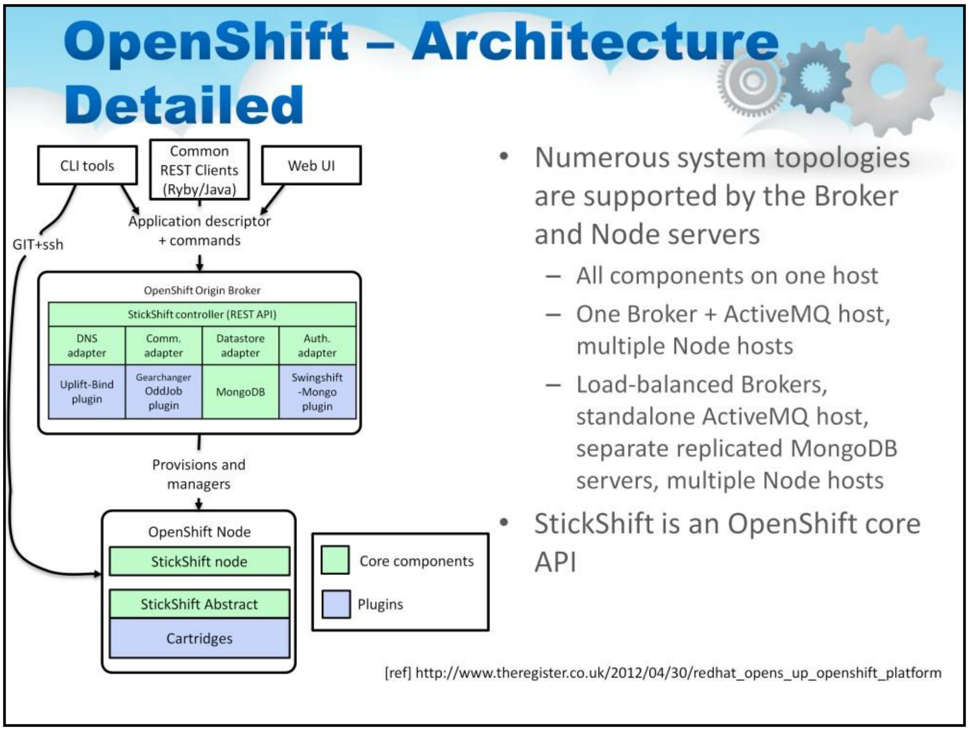
- Two basic functional units are the Broker and Node servers
  - Communication between the Broker and Nodes is done through a message queuing service.
- Broker is the single point of contact for all application management activities
  - User logins, DNS, application state, and general orchestration of the applications.
  - Customers contact the Broker via Web console, CLI tools, or the JBoss Tools IDE over a REST based API
  - The Broker uses an MCollective client (based on ActiveMQ messaging service) to send instructions to the Nodes that actually host user applications
- Node servers host the Gears and built-in Cartridges
  - Gears are container where the user applications are stored and served
    - Gear represents the slice of the Node's CPU, RAM and base storage that is made available to each application.
    - Gears combine the partitioning capabilities with the security features of SELinux
  - Cartridges represent pluggable components that can be combined within a single application, e.g. language or environment cartridge or database cartridge

Now we turn our attentions to understanding RedHat OpenShift

OpenShift has two basic functional units - the Broker and Node servers. Communication between the Broker and Nodes is done through a message queuing service just as with CloudFoundry.

The Broker is the single point of contact for all application management activities, including User logins, DNS, application state, and general orchestration of the applications. Customers contact the Broker via the Web console, CLI tools, or the JBoss Tools IDE over a REST based API. The Broker uses an MCollective client (based on ActiveMQ messaging service) to send instructions to the Nodes that actually host user applications

Node servers host the Gears and built-in Cartridges. Gears are container where the user applications are stored and served, A Gear represents the slice of the Node's CPU, RAM and base storage that is made available to each application. Gears combine the partitioning capabilities with the security features of SELinux. Cartridges represent pluggable components that can be combined within a single application, e.g. language or environment cartridge or database cartridge



OpenShift – Architecture Detailed

- Numerous system topologies are supported by the Broker and Node servers
  - All components on one host
  - One Broker + ActiveMQ host, multiple Node hosts
  - Load-balanced Brokers, standalone ActiveMQ host, separate replicated MongoDB servers, multiple Node hosts
- StickShift is an OpenShift core API

[ref] http://www.theregister.co.uk/2012/04/30/redhat_opens_up_openshift_platform

This slide has a detailed architecture illustration, showing the components of OpenShift.

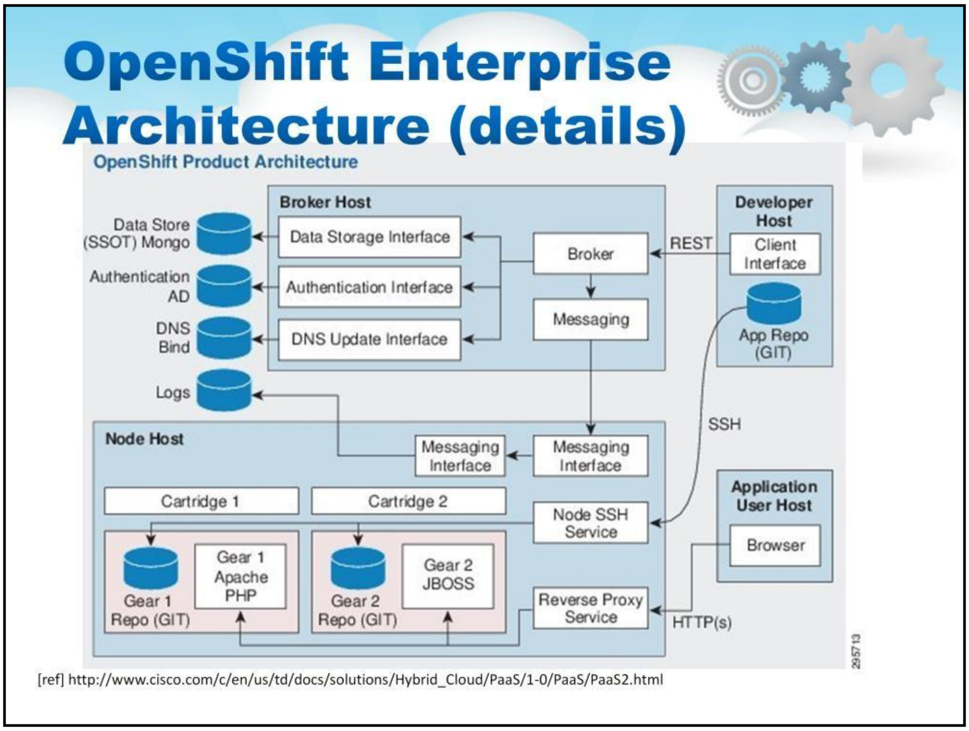
In the most straightforward topology, OpenShift is divided as shown, where the Origin Broker is on one node, and the OpenSHift node with the Cartridge is on another node.

However, Numerous system topologies are supported by the Broker and Node servers

All components on one host

One Broker + ActiveMQ host, multiple Node hosts Load-balanced Brokers, standalone ActiveMQ host, separate replicated MongoDB servers, multiple Node hosts

StickShift is an OpenShift core API internally; one would use Stickshift to add cartridges to the system.



OpenShift Enterprise Architecture (details)

[ref] http://www.cisco.com/c/en/us/td/docs/solutions/Hybrid_Cloud/PaaS/1-0/PaaS/PaaS2.html

This slide shows more detailed architecture of the OpenShift based application hosting, development and management environment.

In particular it shows User browser based client, Developer host, Broker Host and Node Host components.

Node that a single node host (on a VM) can host many gears and cartridges,

Cloud Computing

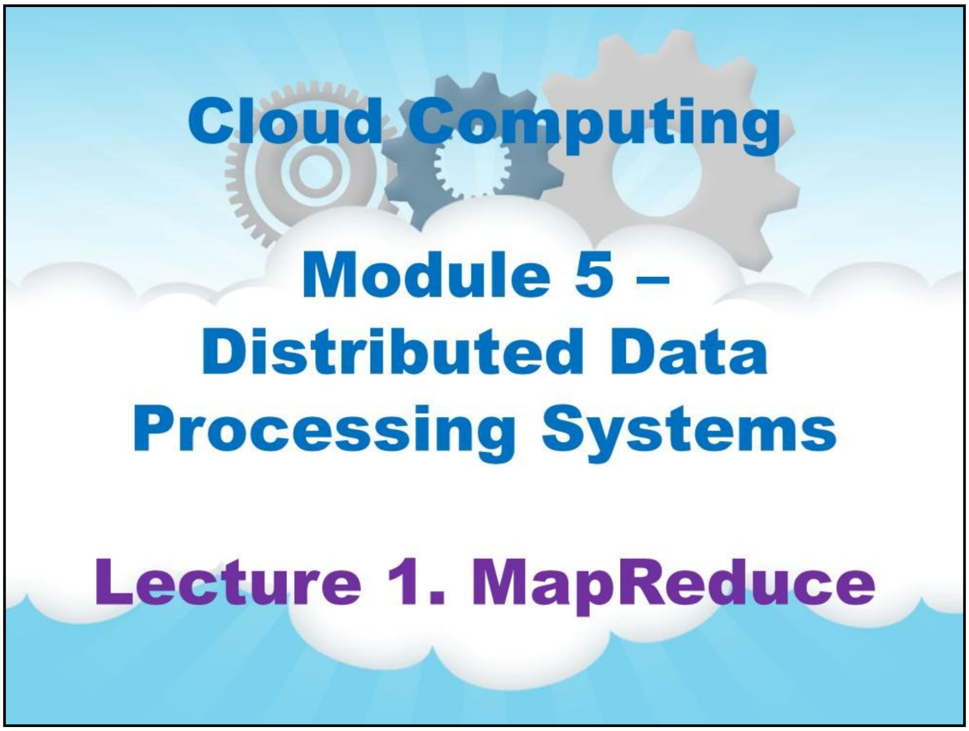
Lecture Manual

Volume 5

Module 5

Distributed Data Processing Systems in Cloud Computing

Content



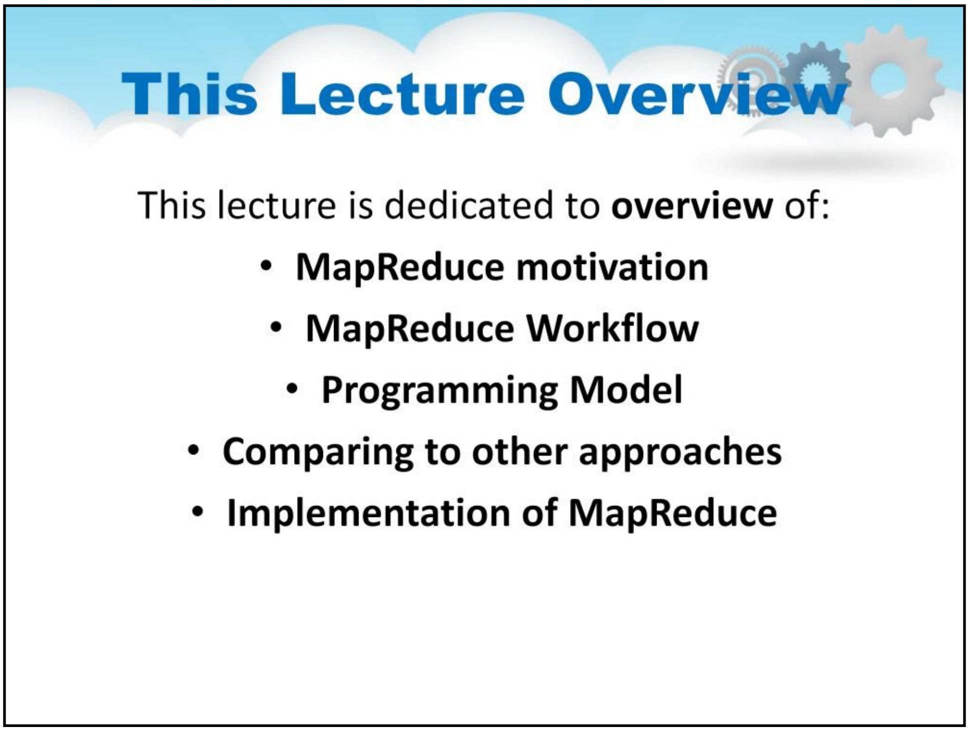
Cloud Computing

Module 5 – Distributed Data Processing Systems

Lecture 1. MapReduce

In this lecture we will consider a model of distributed processing of large data sets, called MapReduce.

Frameworks that implement this model provide good horizontal scaling, and hide from a developer many technical details related to the parallelization, scheduling, data transfer, error handling and recovery from hardware failures.



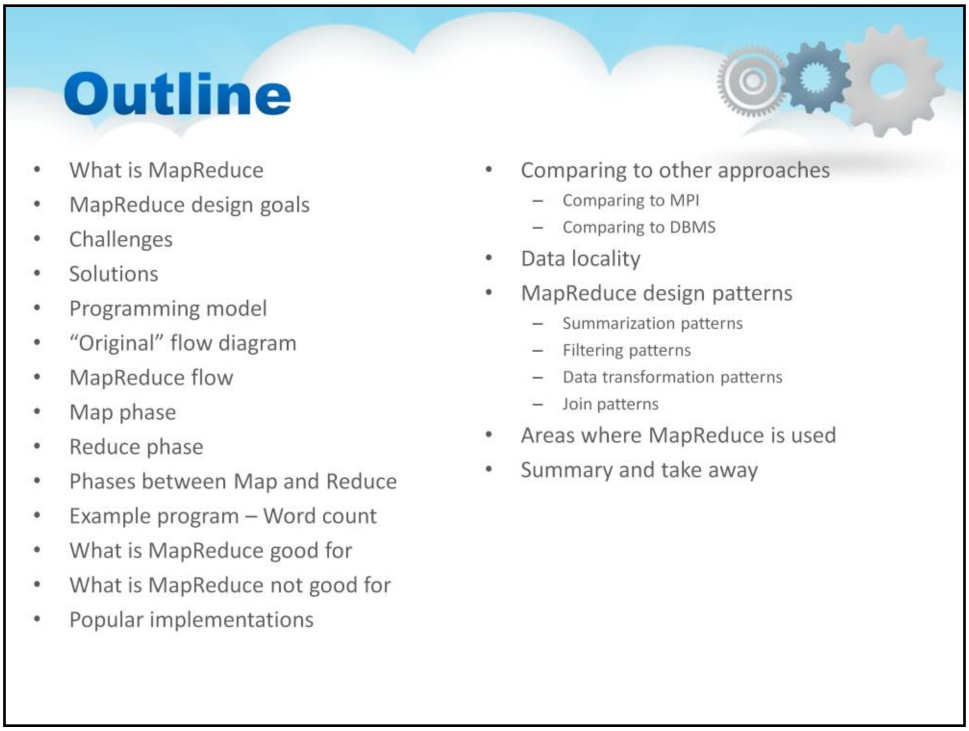
This Lecture Overview

This lecture is dedicated to **overview** of:

- **MapReduce motivation**
- **MapReduce Workflow**
- **Programming Model**
- **Comparing to other approaches**
- **Implementation of MapReduce**

Lecture 1. MapReduce

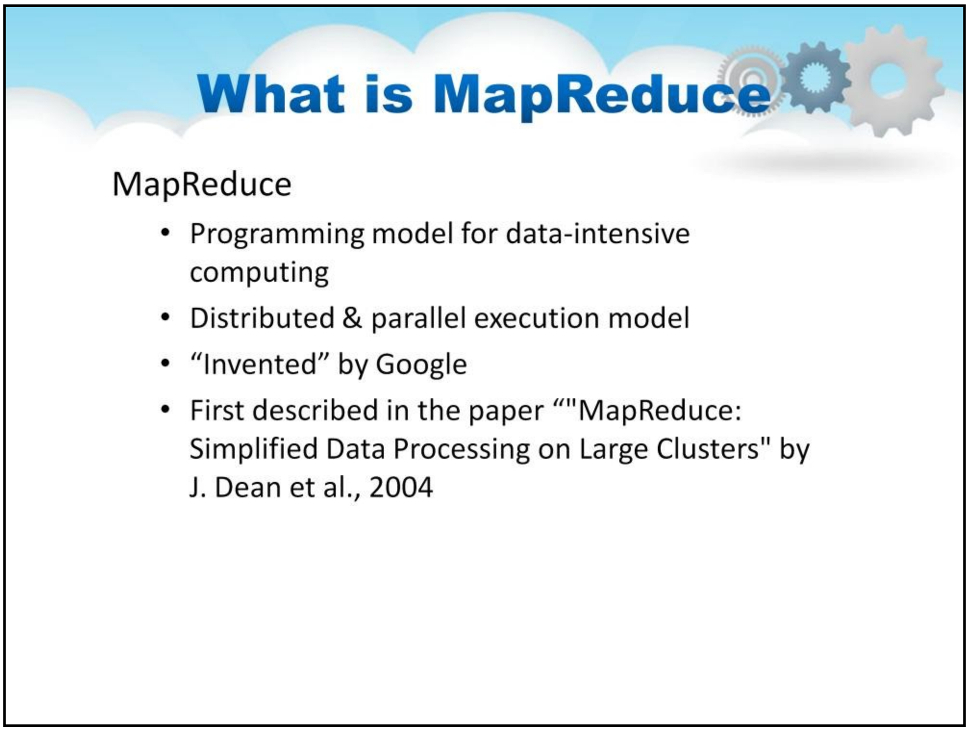
This lecture is dedicated to overview of:
- MapReduce motivation
- MapReduce Workflow
- Programming Model
- MapReduce compared to other approaches of large-scale data processing
- Most popular MapReduce implementations



Outline

- What is MapReduce
- MapReduce design goals
- Challenges
- Solutions
- Programming model
- “Original” flow diagram
- MapReduce flow
- Map phase
- Reduce phase
- Phases between Map and Reduce
- Example program – Word count
- What is MapReduce good for
- What is MapReduce not good for
- Popular implementations
- Comparing to other approaches
  - Comparing to MPI
  - Comparing to DBMS
- Data locality
- MapReduce design patterns
  - Summarization patterns
  - Filtering patterns
  - Data transformation patterns
  - Join patterns
- Areas where MapReduce is used
- Summary and take away

What is MapReduce
MapReduce design goals
Challenges
Solutions
Programming model
“Original” flow diagram
MapReduce flow
Map phase
Reduce phase
Phases between Map and Reduce
Example program – Word count
What is MapReduce good for
What is MapReduce not good for
Popular implementations
Comparing to other approaches
    Comparing to MPI
    Comparing to DBMS
Data locality
MapReduce design patterns
    Summarization patterns
    Filtering patterns
    Data transformation patterns
    Join patterns
Areas where MapReduce is used
Summary and take away



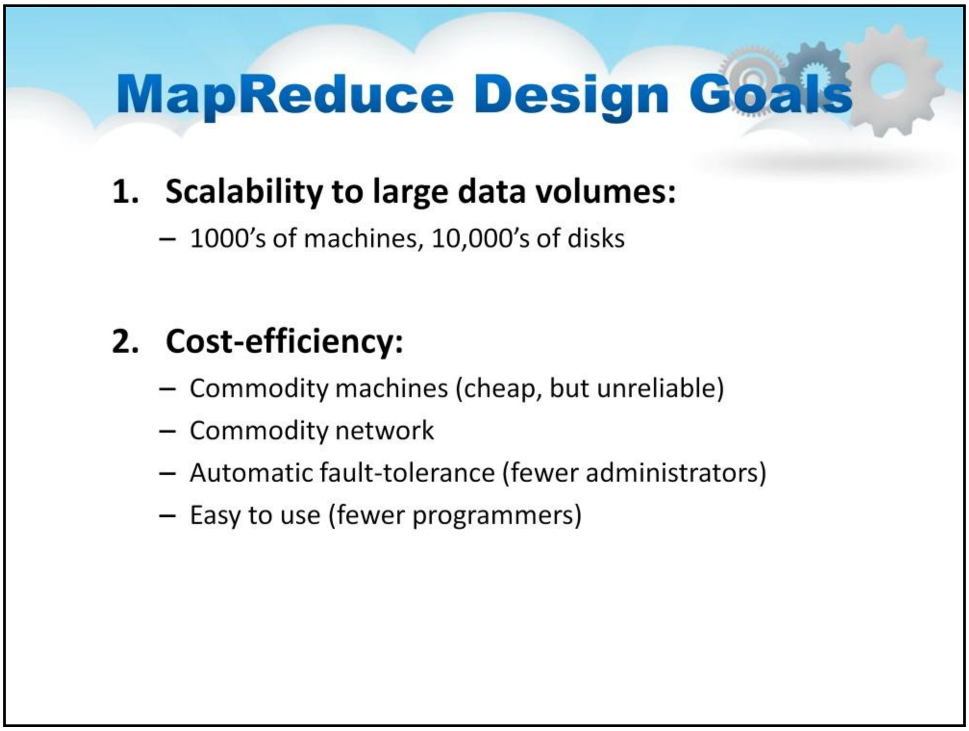
What is MapReduce

MapReduce

- Programming model for data-intensive computing
- Distributed & parallel execution model
- “Invented” by Google
- First described in the paper “"MapReduce: Simplified Data Processing on Large Clusters" by J. Dean et al., 2004

Overview

MapReduce is a programming model and an associated implementation for processing large data sets.
MapReduce originated from functional programming and was introduced by Google
in a paper called “MapReduce: Simplified Data Processing on Large Clusters.”
One of Google’s first challenges was to figure out how to index the exploding volume of content on the web.
To solve this, Google “invented” a new style of data processing known as MapReduce to manage large-scale data processing across large clusters of commodity servers.
Google’s MapReduce implementation is a proprietary solution and has not yet been released to the public.



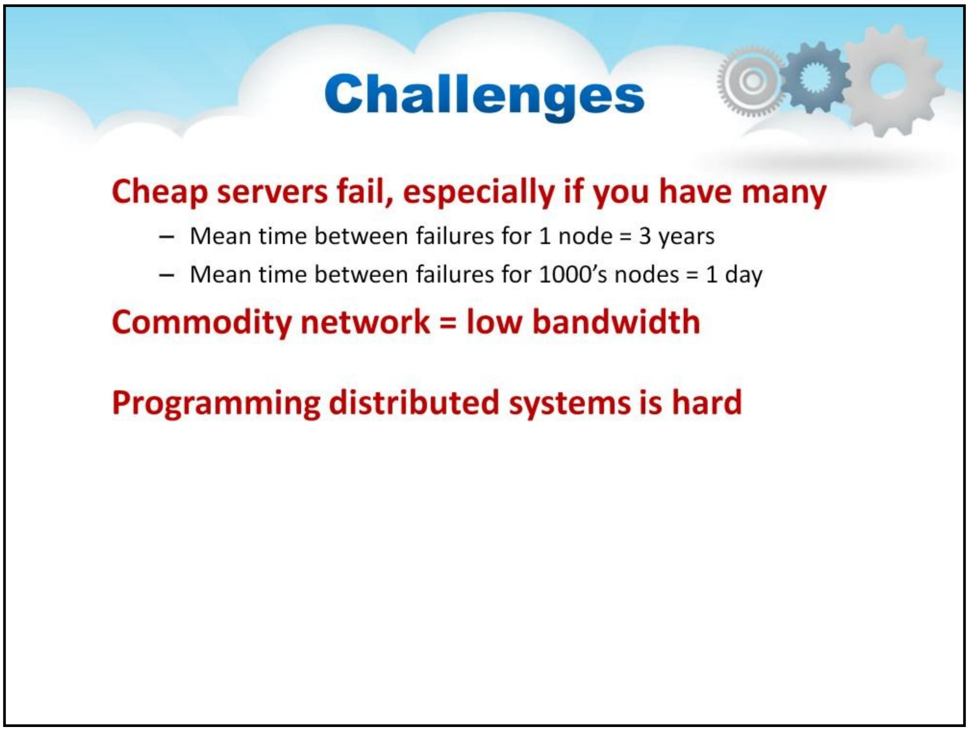
MapReduce Design Goals

1. **Scalability to large data volumes:**
   - 1000's of machines, 10,000's of disks
2. **Cost-efficiency:**
   - Commodity machines (cheap, but unreliable)
   - Commodity network
   - Automatic fault-tolerance (fewer administrators)
   - Easy to use (fewer programmers)

The main design goals were:

First**, scalability to large data volumes**.
This is not about a cluster of several, or several tens of computers. It's about
thousands and more computers with disks.

Second, **cost-efficiency**.
Commodity computers and network equipment should be used. It means that hardware failures is rule rather than exception.
So, at the software level, there should be built-in failover and recovery functions. Some abstraction should hide from the developer complex technical details related to parallelization and scheduling, inter-machine communications, handling hardware failures.
Those specific knowledge should not be required.
This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.



Challenges

Cheap servers fail, especially if you have many

- Mean time between failures for 1 node = 3 years
- Mean time between failures for 1000’s nodes = 1 day

Commodity network = low bandwidth

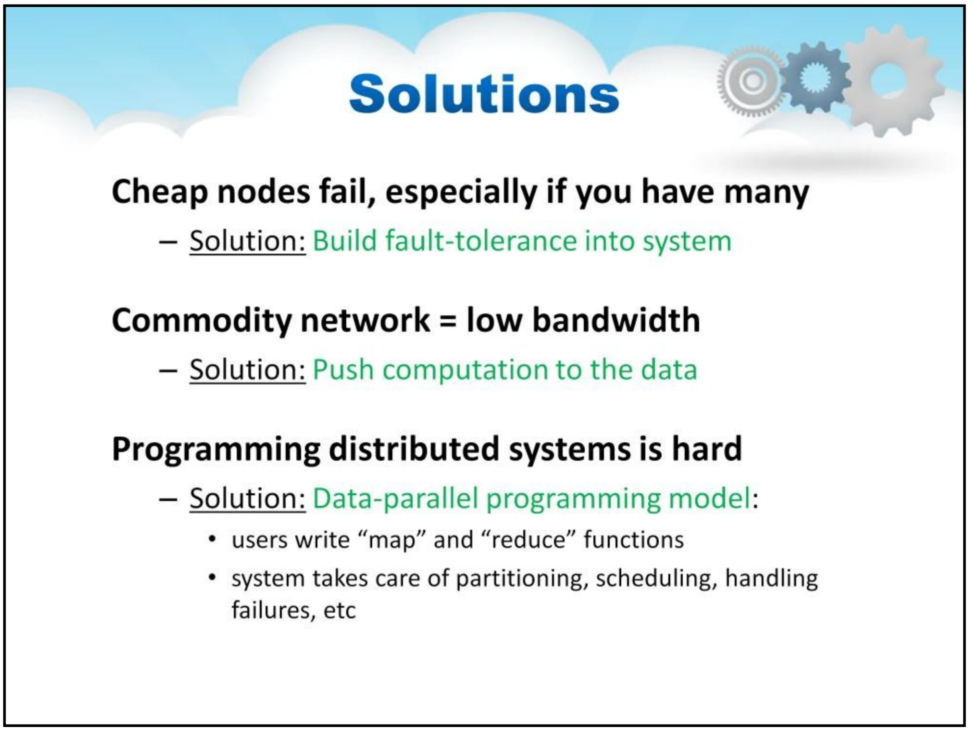
Programming distributed systems is hard

Consider the challenges that can be encountered when developing such a system.

First challenge is a mean time between failures. For one node is three years, but for thousand of nodes is about one day, because any of nodes can fail and the whole system will fail.

Second challenge is a commodity network. If there is no special high-speed interconnect between nodes, the bandwidth is low and the latency is high.
Intensive data exchange between processes on different nodes will reduce the overall system performance.

Third, programming distributed systems is hard. It require specific knowledge.



Solutions

Cheap nodes fail, especially if you have many

- Solution: Build fault-tolerance into system

Commodity network = low bandwidth

- Solution: Push computation to the data

Programming distributed systems is hard

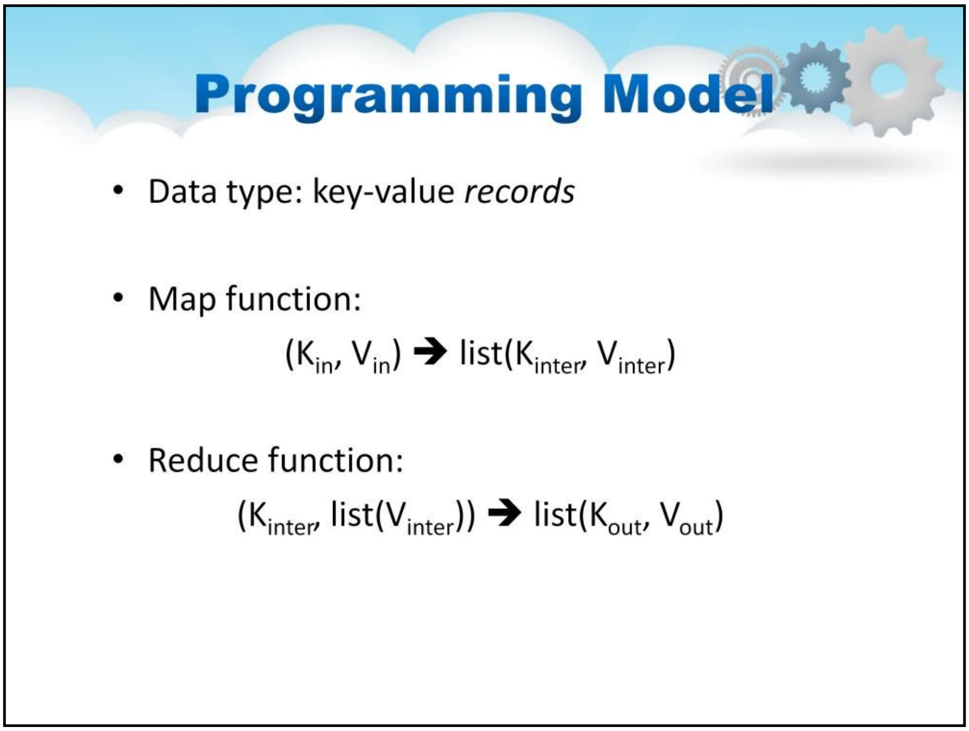
- Solution: Data-parallel programming model:
  - users write "map" and "reduce" functions
  - system takes care of partitioning, scheduling, handling failures, etc

Consider possible solutions

First challenge is a mean time between failures.
The solution is to build fault-tolerance into the system.

Second problem is a commodity network.
The solution is to push computation to the data.

Third, programming distributed systems is hard.
The solution is to create a simple programming model. Here we have only two functions- Map and Reduce. Framework will take care of the rest.



Programming Model

- Data type: key-value *records*
- Map function:

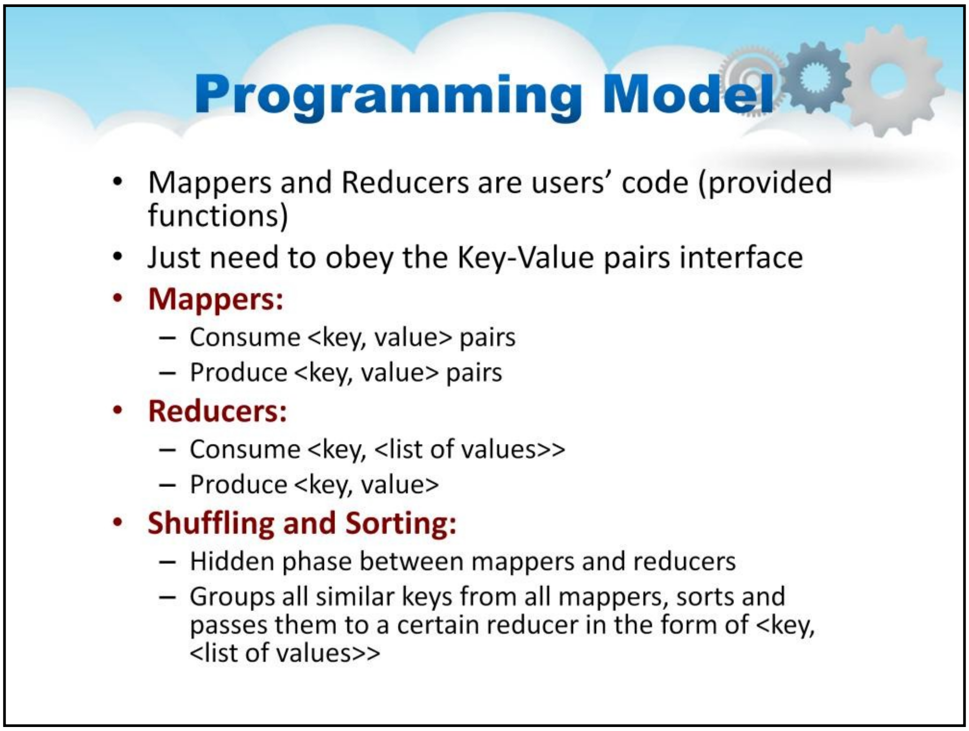
$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

- Reduce function:

$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Programming Model

In general form the model looks like this:
Map function maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records.
Reduce function reduces a set of intermediate values which share a key to a smaller set of values.



Programming Model

- Mappers and Reducers are users' code (provided functions)
- Just need to obey the Key-Value pairs interface
- **Mappers:**
  - Consume <key, value> pairs
  - Produce <key, value> pairs
- **Reducers:**
  - Consume <key, <list of values>>
  - Produce <key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>

FUNCTIONAL PROGRAMMING CONCEPTS

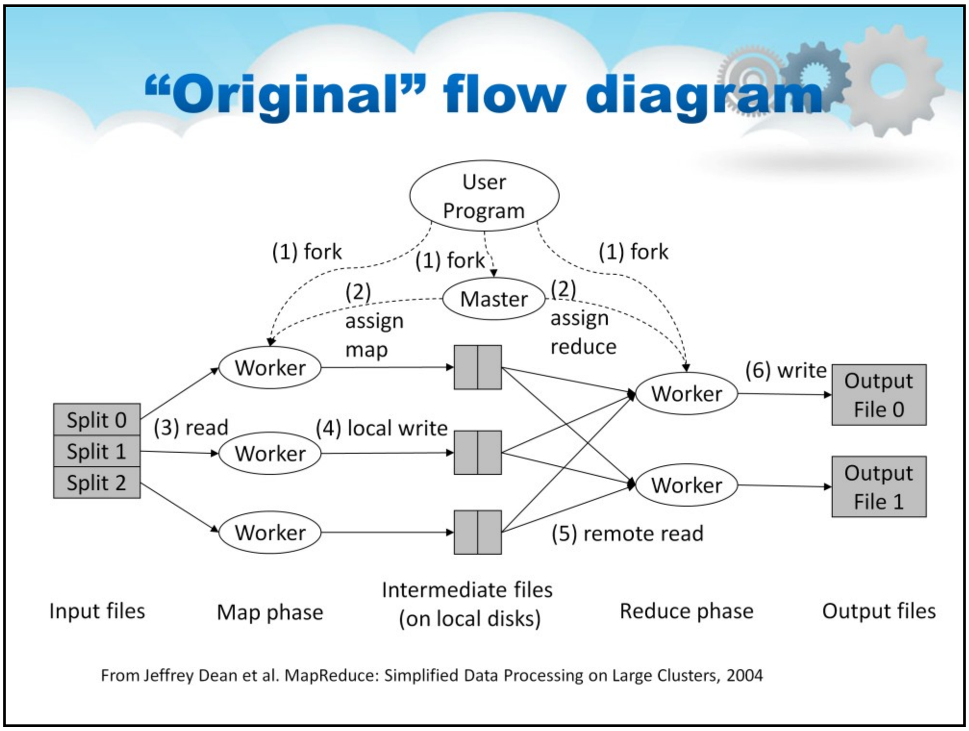
MapReduce programs are designed to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines.

This model would not scale to large clusters (hundreds or thousands of nodes) if the

components were allowed to share data arbitrarily.

The communication overhead required to keep the data on the nodes synchronized at all times would prevent the system from performing reliably or efficiently at large scale.

Instead, all data elements in MapReduce are **immutable**, meaning that they cannot be updated. If in a mapping task you change an input (key, value) pair, it does not get reflected back in the input files; communication occurs only by generating new output (key, value) pairs which are then forwarded by the system into the next phase of execution.



“Original” flow diagram

User Program

(1) fork

(1) fork

(1) fork

Master

(2) assign map

(2) assign reduce

Split 0

Split 1

Split 2

(3) read

Worker

Worker

Worker

(4) local write

(5) remote read

Worker

Worker

(6) write

Output File 0

Output File 1

Input files

Map phase

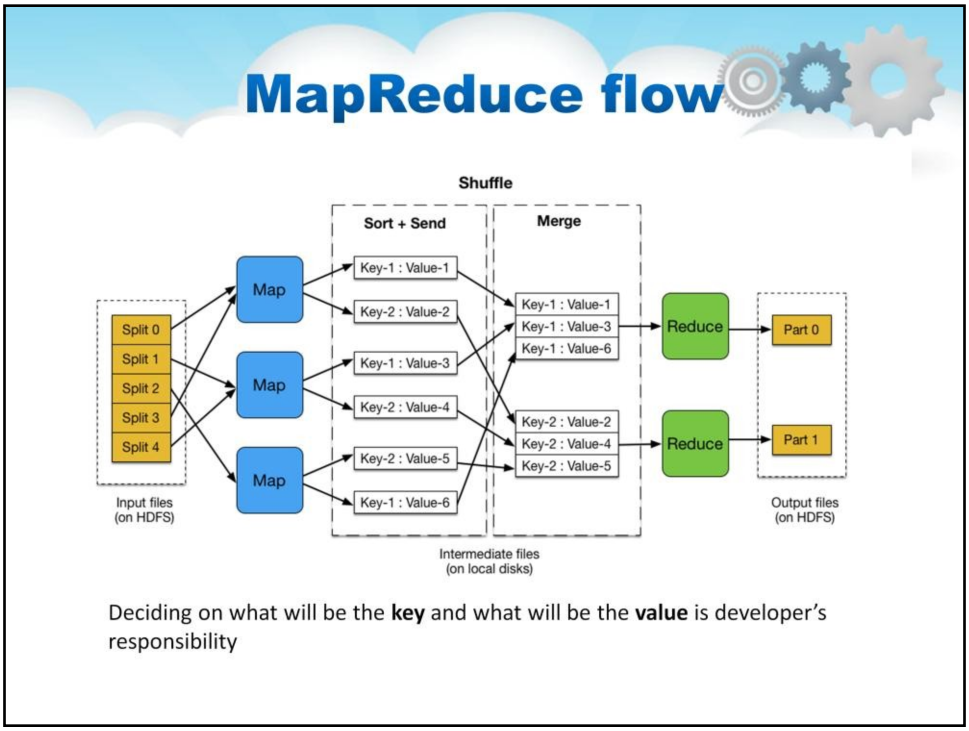
Intermediate files (on local disks)

Reduce phase

Output files

From Jeffrey Dean et al. MapReduce: Simplified Data Processing on Large Clusters, 2004

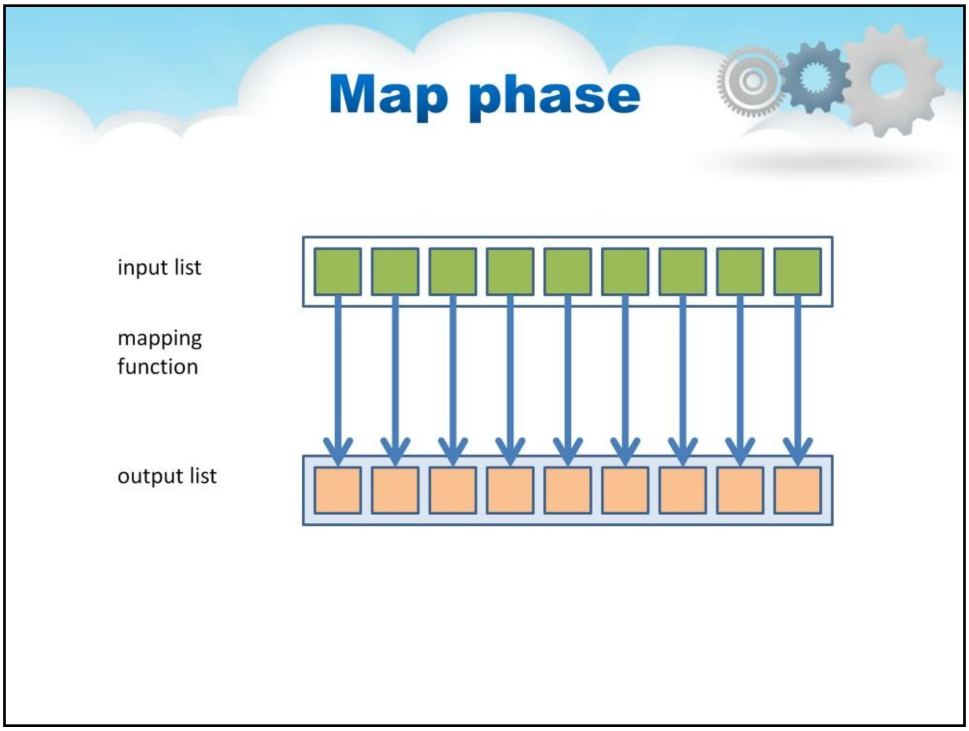
This is the original flow diagram from the paper "MapReduce: Simplified Data Processing on Large Clusters" by Jeffrey Dean, published in 2014.



MapReduce flow

Deciding on what will be the **key** and what will be the **value** is developer's responsibility

Conceptually, MapReduce programs transform lists of input data elements into lists of output data elements.
A MapReduce program will do this twice, using two different list processing
idioms: ***map***, and ***reduce***. These terms are taken from list processing languages such
as LISP.



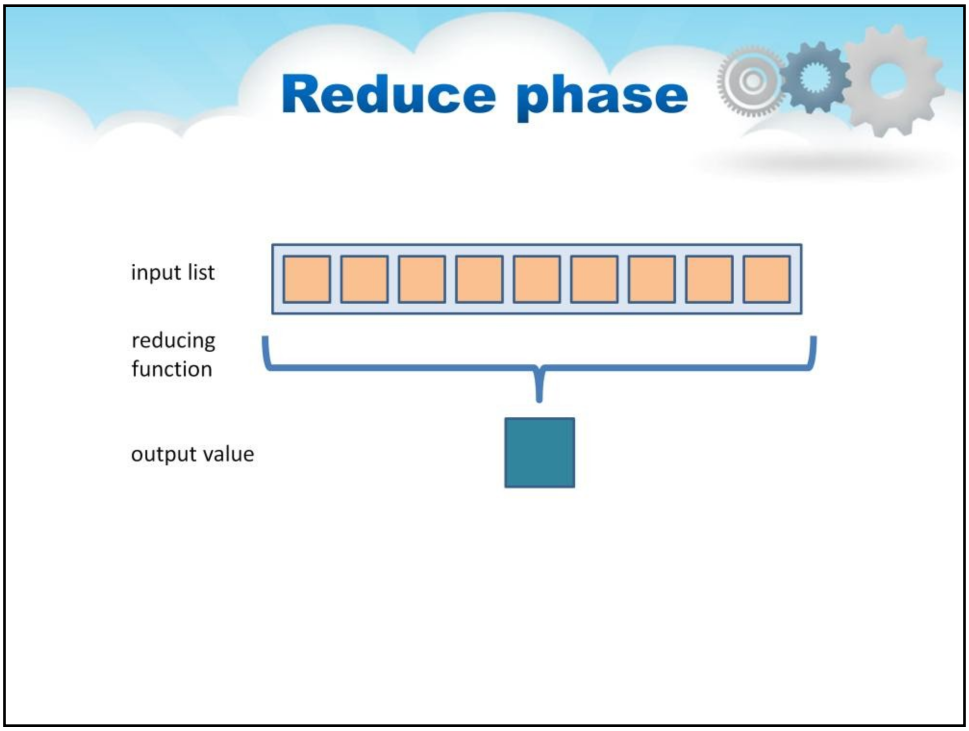
Map phase

input list

mapping
function

output list

The first phase of a MapReduce program is called ***mapping***. A list of data elements are provided, one at a time, to a function called the ***Mapper***, which transforms each element individually to an output data element.
Mapping creates a new output list by applying a function to individual elements of an
input list.
As an example of the utility of map: Suppose you had a function toUpper(str) which returns an uppercase version of its input string. You could use this function
with map to turn a list of strings into a list of uppercase strings.
Note that we are not modifying the input string here: we are returning a new string that will form part of a new output list.



Reduce phase

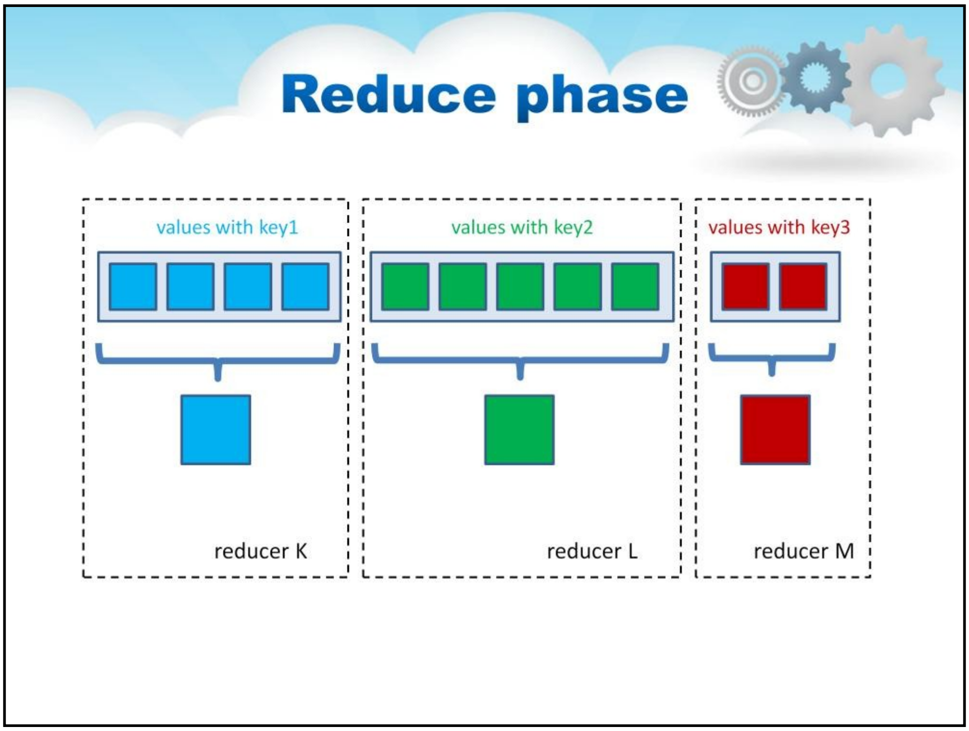
input list

reducing function

output value

Reducing lets you aggregate values together. A ***reducer*** function receives an iterator of input values from an input list. It then combines these values together, returning a single output value.

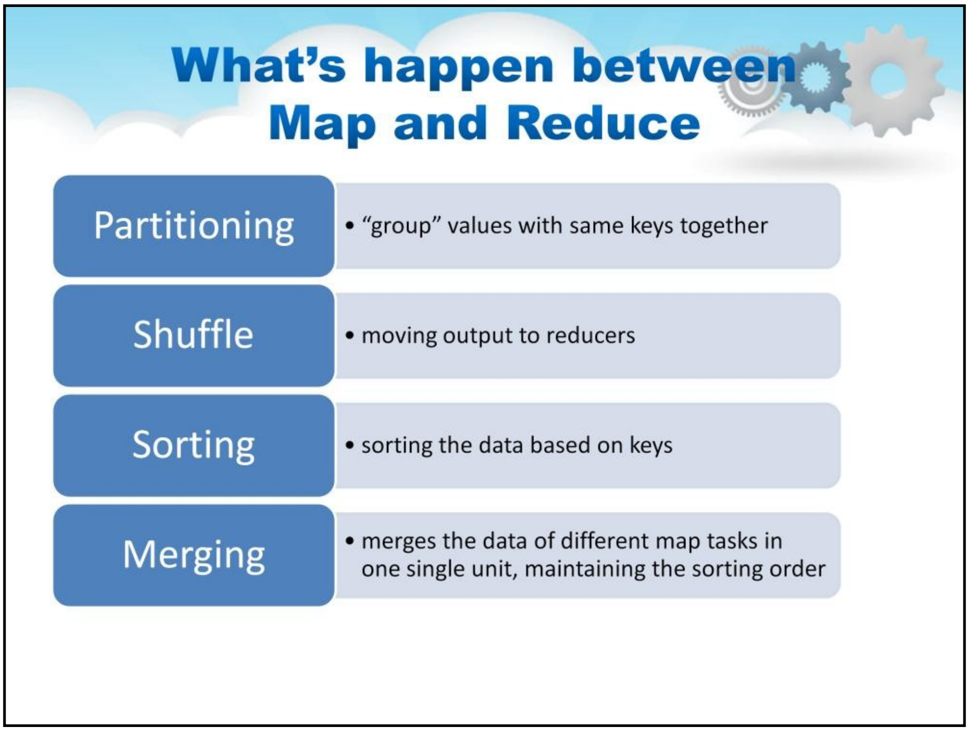
Reducing is often used to produce "summary" data, turning a large volume of data into a smaller summary of itself. For example, "+" can be used as a reducing function, to return the sum of a list of input values.



Reduce phase

The MapReduce framework takes these concepts and uses them to process large volumes of information. A MapReduce program has two components: one that implements the mapper, and another that implements the reducer. The Mapper and Reducer idioms described above are extended slightly to work in this environment, but the basic principles are the same.

Keys and values: In MapReduce, no value stands on its own. Every value has a ***key*** associated with it. Keys identify related values.
The mapping and reducing functions receive not just values, but (key, value) pairs. The output of each of these functions is the same: both a key and a value must be emitted to the next list in the data flow.
Keys divide the reduce space: A reducing function turns a large list of values into one (or a few) output values. In MapReduce, all of the output values are not usually reduced together. All of the values *with the same key* are presented to a single reducer together. This is performed independently of any reduce operations occurring on other lists of values, with different keys attached.
On the diagram, different colors represent different keys. All values with the same key are presented to a single reduce task.



What’s happen between Map and Reduce

Partitioning
- “group” values with same keys together

Shuffle
- moving output to reducers

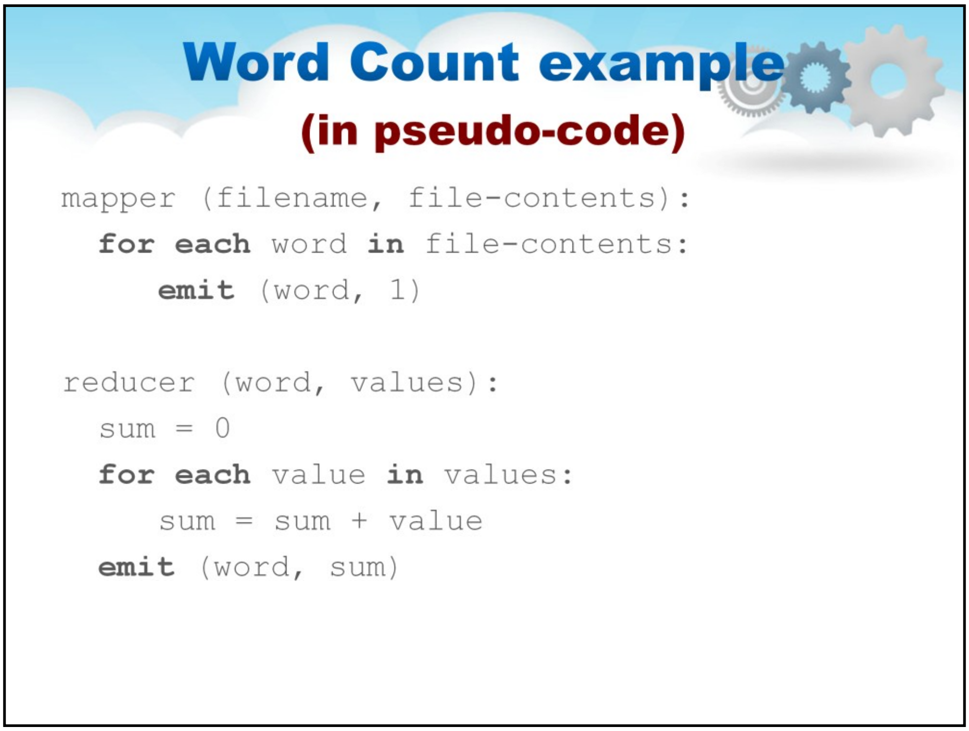
Sorting
- sorting the data based on keys

Merging
- merges the data of different map tasks in one single unit, maintaining the sorting order

Partition & Shuffle: After the first map tasks have completed, the nodes may still be performing several more map tasks each.
But they also begin exchanging the intermediate outputs from the map tasks to where they are required by the reducers.
This process of moving map outputs to the reducers is known as ***shuffling***.
A different subset of the intermediate key space is assigned to each reduce node; these subsets (known as "partitions") are the inputs to the reduce tasks.
Each map task may emit (key, value) pairs to any partition; all values for the same key are always reduced together regardless of which mapper is its origin.
Therefore, the map nodes must all agree on where to send the different pieces of the intermediate data.
The Partitioner determines which partition a given (key, value) pair will go to. The default partitioner computes a hash value for the key and assigns the partition based on this result.

Sorting
It is just sorting the data based on keys

Merging:
This happens on reducer side. Reducer can get data from multiple map tasks and through merging it merges the data of different map tasks in one single unit, maintaining the sorting order.



Word Count example

(in pseudo-code)

```
mapper (filename, file-contents):
  for each word in file-contents:
     emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
     sum = sum + value
  emit (word, sum)
```

Example

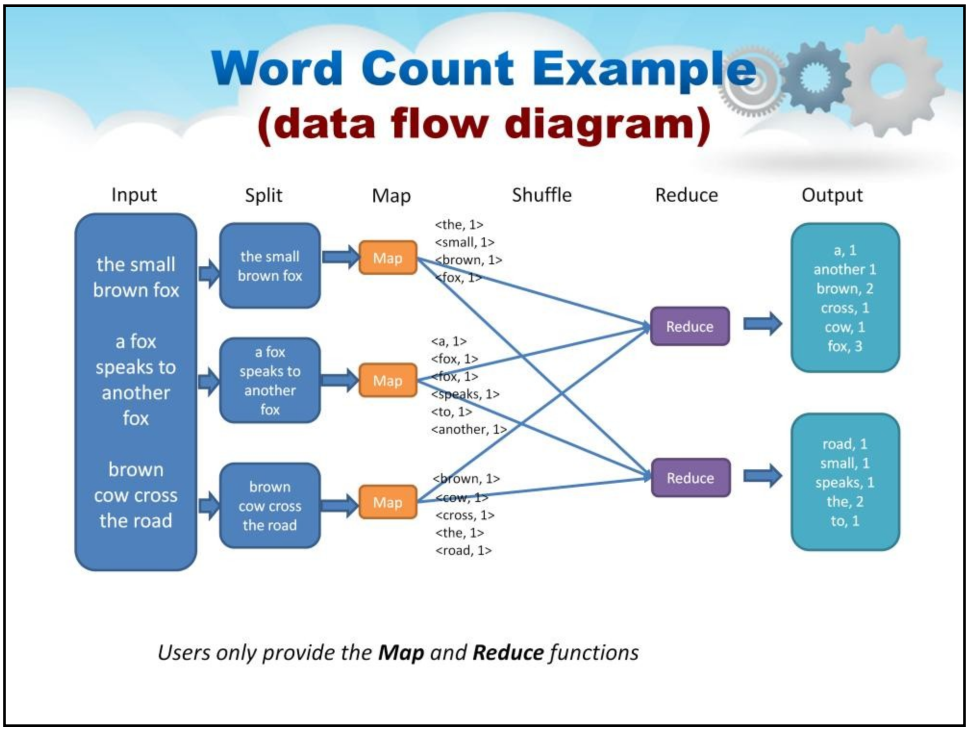
Now look at the example program in pseudo-code. It counts word occurrences in a text.

Several instances of the **mapper** function are created on the different machines in our cluster. Each instance receives a different input file (it is assumed that we have many such files).

The mappers output (word, 1) pairs which are then forwarded to the reducers. Several instances of the reducer method are also instantiated on the different machines.

Each reducer is responsible for processing the list of values associated with a different word.

The list of values will be a list of 1's; the reducer sums up those ones into a final count associated with a single word. The reducer then emits the final (word, count) output which is written to an output file.



Word Count Example
(data flow diagram)

Input | Split | Map | Shuffle | Reduce | Output

the small brown fox

a fox speaks to another fox

brown cow cross the road

the small brown fox → Map → <the, 1> <small, 1> <brown, 1> <fox, 1>

a fox speaks to another fox → Map → <a, 1> <fox, 1> <fox, 1> <speaks, 1> <to, 1> <another, 1>

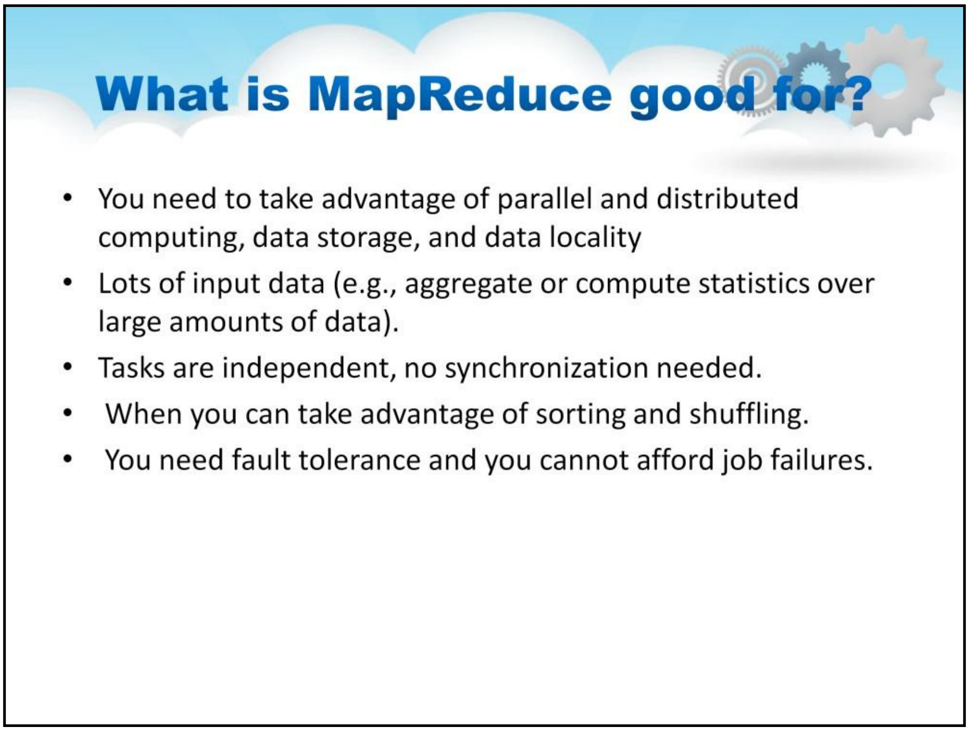
brown cow cross the road → Map → <brown, 1> <cow, 1> <cross, 1> <the, 1> <road, 1>

Reduce → a, 1 another 1 brown, 2 cross, 1 cow, 1 fox, 3

Reduce → road, 1 small, 1 speaks, 1 the, 2 to, 1

*Users only provide the **Map** and **Reduce** functions*

Let’s understand each of the stages depicted in the above diagram.
Input: This is the input data to be processed.
Split: Framework splits the incoming data into smaller pieces called "splits".
Map: In this step, MapReduce processes each split according to the logic defined in map() function. One mapper works on one split at a time. Each mapper is treated as a task.
Shuffle & Sort: In this step, outputs from all the mappers is shuffled, sorted to put them in order, and grouped before sending them to the next step.
Reduce: This step is used to aggregate the outputs of mappers using the reduce() function. Output of reducer is sent to the next and final step. Each reducer is treated as a task.
Output: Finally the output of reduce step is written to a file.



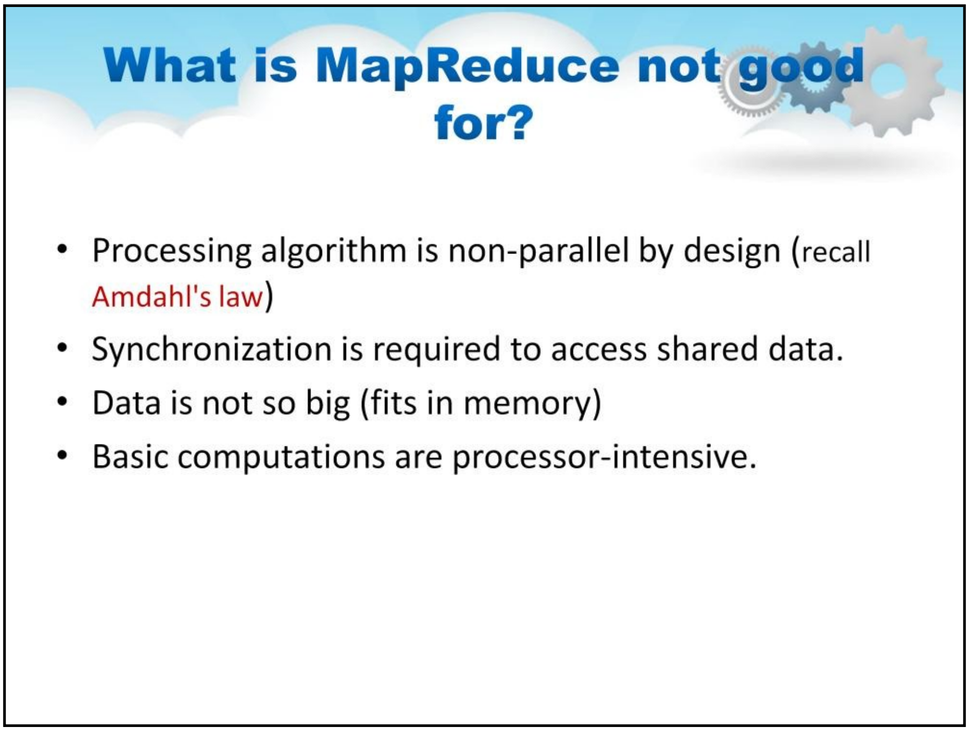
What is MapReduce good for?

- You need to take advantage of parallel and distributed computing, data storage, and data locality
- Lots of input data (e.g., aggregate or compute statistics over large amounts of data).
- Tasks are independent, no synchronization needed.
- When you can take advantage of sorting and shuffling.
- You need fault tolerance and you cannot afford job failures.

Usage

There are many cases where MapReduce is appropriate, such as:

- When you have to handle lots of input data (e.g., aggregate or compute statistics over large amounts of data).
- When you need to take advantage of parallel and distributed computing, data storage, and data locality.
- When you can do many tasks independently without synchronization.
- When you can take advantage of sorting and shuffling.
- When you need fault tolerance and you cannot afford job failures.



What is MapReduce not good for?

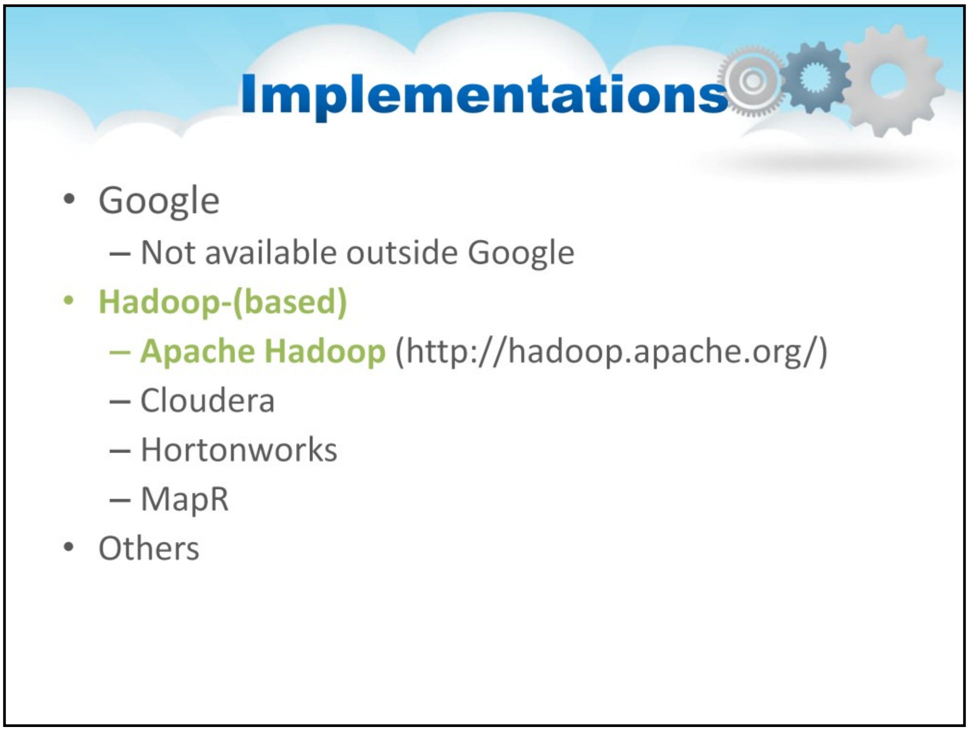
- Processing algorithm is non-parallel by design (recall Amdahl's law)
- Synchronization is required to access shared data.
- Data is not so big (fits in memory)
- Basic computations are processor-intensive.

Is MapReduce good for everything? The simple answer is no. When we have big data, if we can partition it and each partition can be processed independently, then we can start to think about MapReduce algorithms. But here are other scenarios where MapReduce should not be used:

- If the computation of a value depends on previously computed values. One good example is the Fibonacci series, where each value is a summation of the previous two values:

F(k + 2) = F(k + 1) + F(k)

- If the data set is small enough to be computed on a single machine.
- If synchronization is required to access shared data.
- If all of your input data fits in memory.
- If one operation depends on other operations.
- If basic computations are processor-intensive.



Implementations

- Google
  - Not available outside Google
- **Hadoop-(based)**
  - **Apache Hadoop** (http://hadoop.apache.org/)
  - Cloudera
  - Hortonworks
  - MapR
- Others

Google’s MapReduce implementation isn’t available outside Google. Other well-known implementations are based on Apache Hadoop.

Cloudera, founded in 2008, was the first company to develop and distribute Apache Hadoop-based software and still has the largest user base with most number of clients.

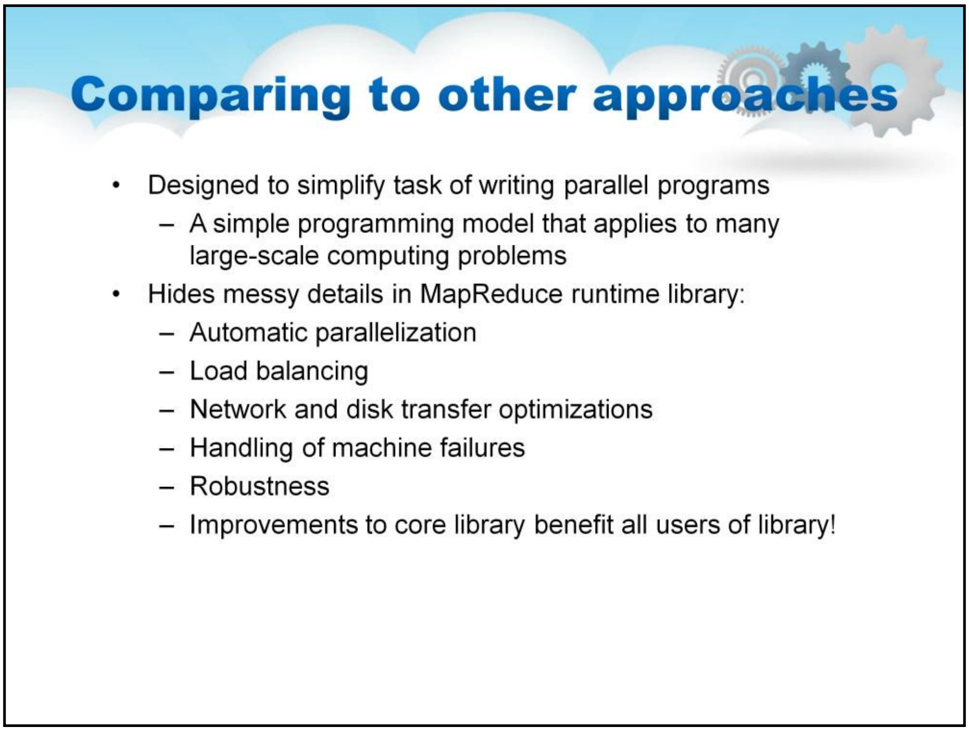
Although the core of the distribution is based on Apache Hadoop, it also provides a proprietary services and tools.

Hortonworks, founded in 2011, has quickly emerged as one of the leading vendors of Hadoop.

The distribution provides open source platform based on Apache Hadoop for analysing, storing and managing big data.

Hortonworks is the only commercial vendor to distribute complete open source Apache Hadoop without additional proprietary software.

MapR replaces HDFS component (will be described later) and instead uses its own proprietary file system, called MapR FS.



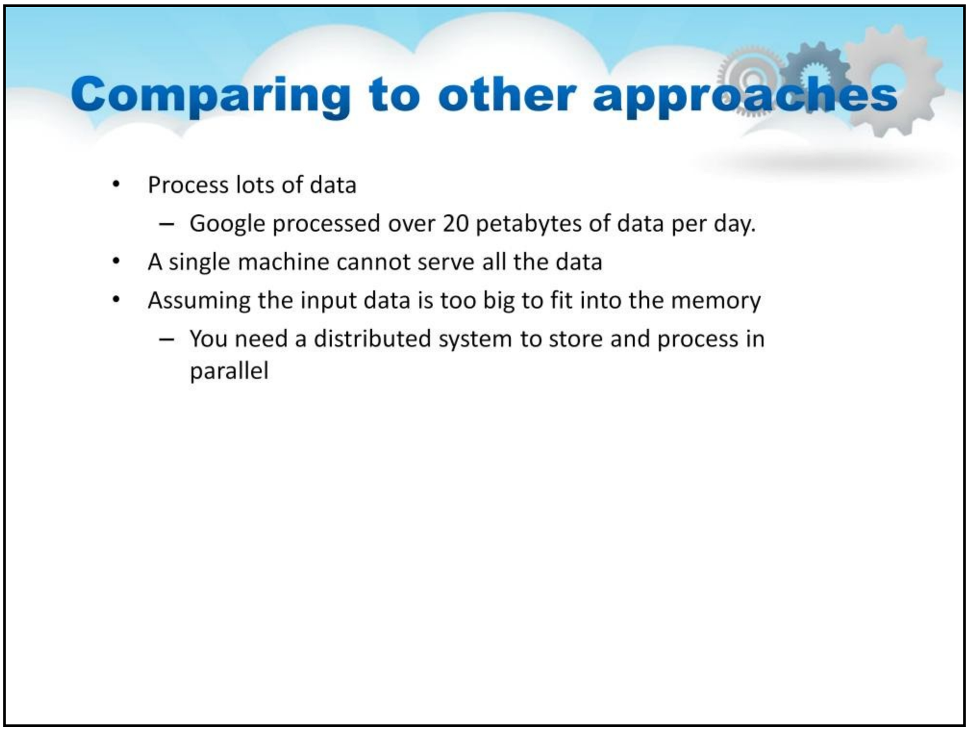
Comparing to other approaches

- Designed to simplify task of writing parallel programs
  - A simple programming model that applies to many large-scale computing problems
- Hides messy details in MapReduce runtime library:
  - Automatic parallelization
  - Load balancing
  - Network and disk transfer optimizations
  - Handling of machine failures
  - Robustness
  - Improvements to core library benefit all users of library!

Comparison

Inspired by functional programming, the computing paradigm of MapReduce is simple and easy to understand.
Because of automatic parallelization, no explicit handling of data transfer and synchronization in programs, and no deadlock, this model is very attractive. Meanwhile it is sufficient for simple computations mentioned earlier. In contrast, the learning curve of much more comprehensive MPI is steep.

Fault tolerance. Because hardware failures are common in the large clusters of commodity PCs, MapReduce puts a lot of emphasis on fault tolerance. In MPI, it is mainly application developers’ job to make program fault tolerance.

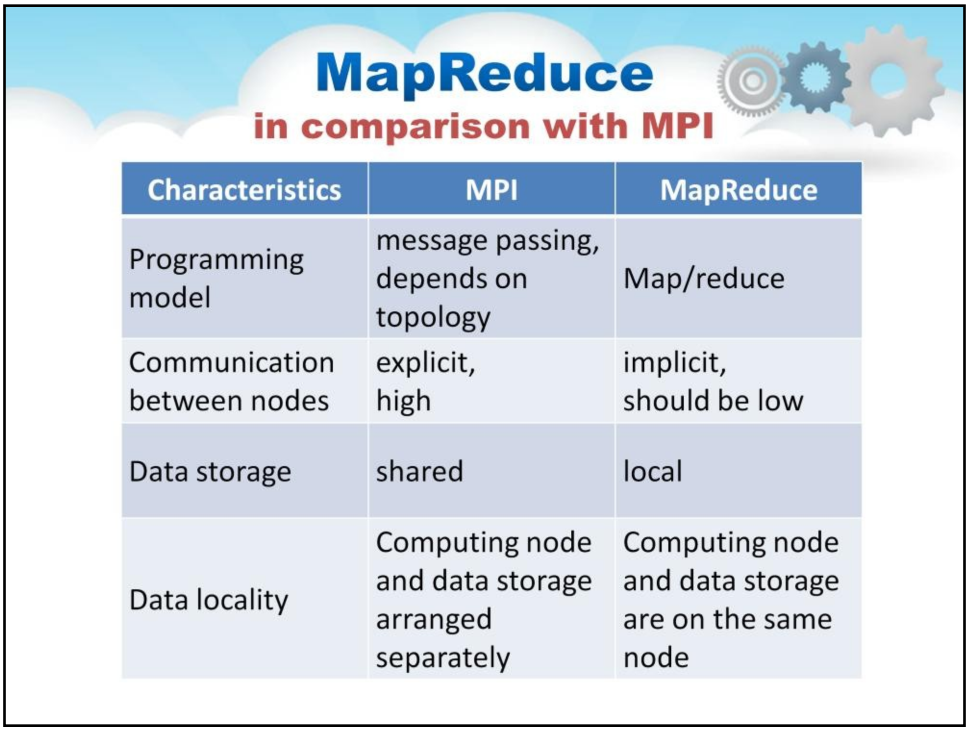


Comparing to other approaches

- Process lots of data
  - Google processed over 20 petabytes of data per day.
- A single machine cannot serve all the data
- Assuming the input data is too big to fit into the memory
  - You need a distributed system to store and process in parallel

Assuming the input data is too big to fit into the memory (combined from all nodes), MapReduce employs a data flow model, which also provides a simple I/O interface to access large amount of data in distributed file system.

It also exploits data locality for efficiency. In most cases, we don’t need to worry about I/O at all. Although it sounds trivial, it is actually an important improvement for big data analytics.



MapReduce

in comparison with MPI

Characteristics	MPI	MapReduce
Programming model	message passing, depends on topology	Map/reduce
Communication between nodes	explicit, high	implicit, should be low
Data storage	shared	local
Data locality	Computing node and data storage arranged separately	Computing node and data storage are on the same node

MapReduce and MPI

Consider some differences between MPI and MapReduce.

In the MPI programming model, a computation comprises one or more processes that communicate by calling library routines to send and receive messages to other processes

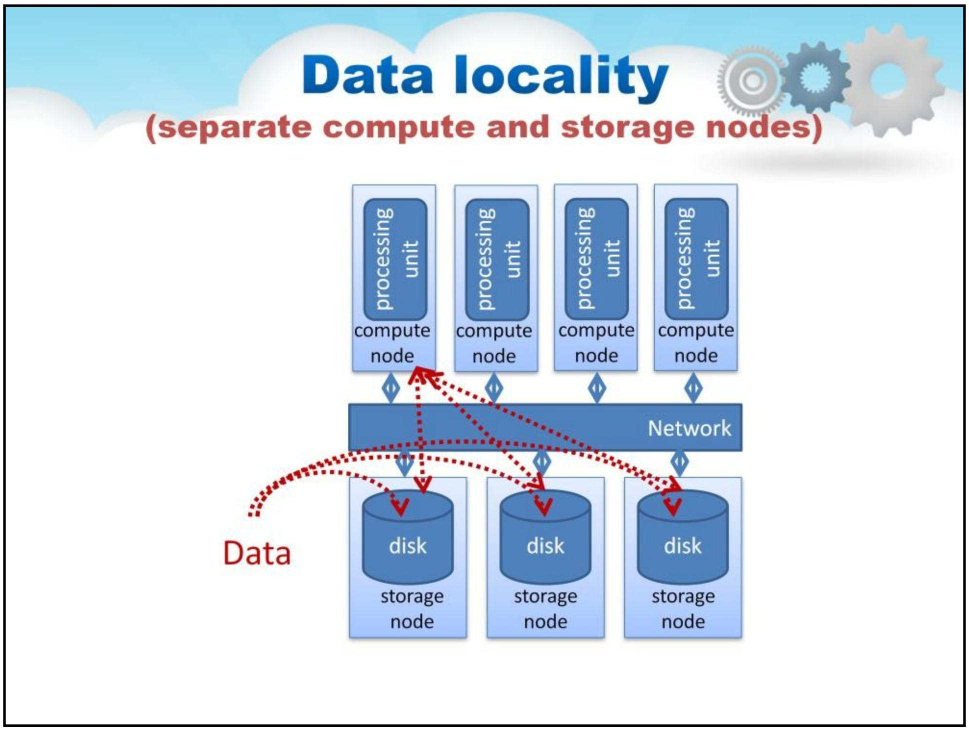
MapReduce model based on two functins – map and reduce.

MPI allows explicit communication between processes at runtime. A developer decides, when and where to send the message.

In MapReduce, communication is implicit. It is assumed that data exchange occurs only at certain stages of execution.

In MPI, usually all nodes of the cluster are connected to the shared storage. A computing node and a data storage are different nodes.

In contrast, in MapReduce typically the compute nodes and the storage nodes are the same.

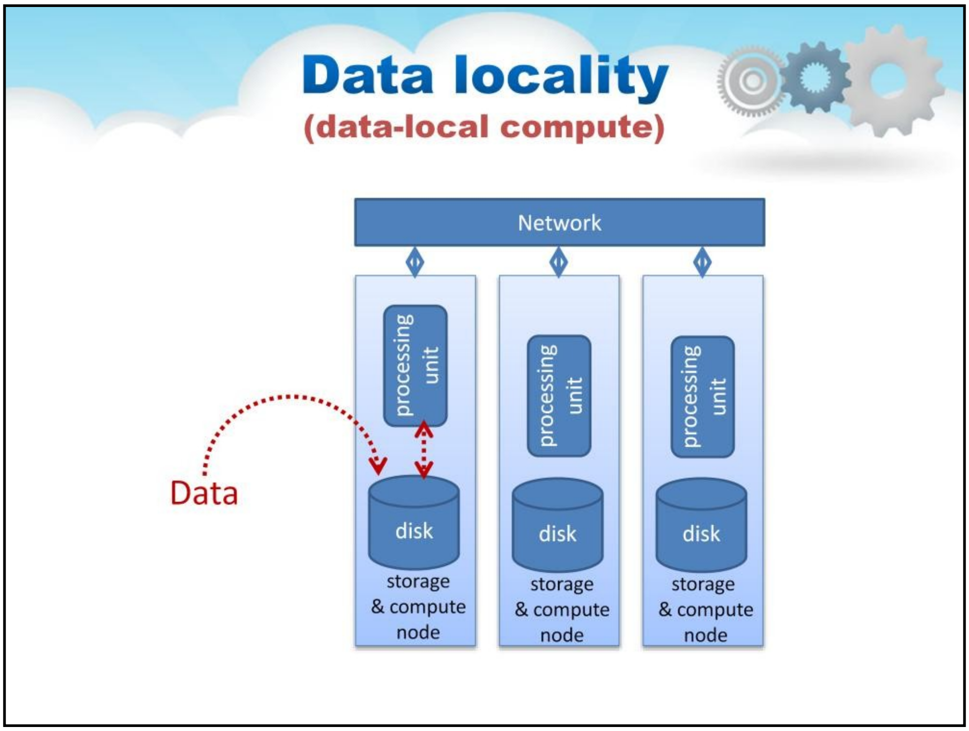


Data locality

(separate compute and storage nodes)

In many traditional data processing systems, storage (where the data is at rest) and compute (where the data gets processed) have been kept separate.

In these systems, data would be moved over a very fast network to computers that would then process it. Moving data over a network is really expensive in terms of time.



Data locality

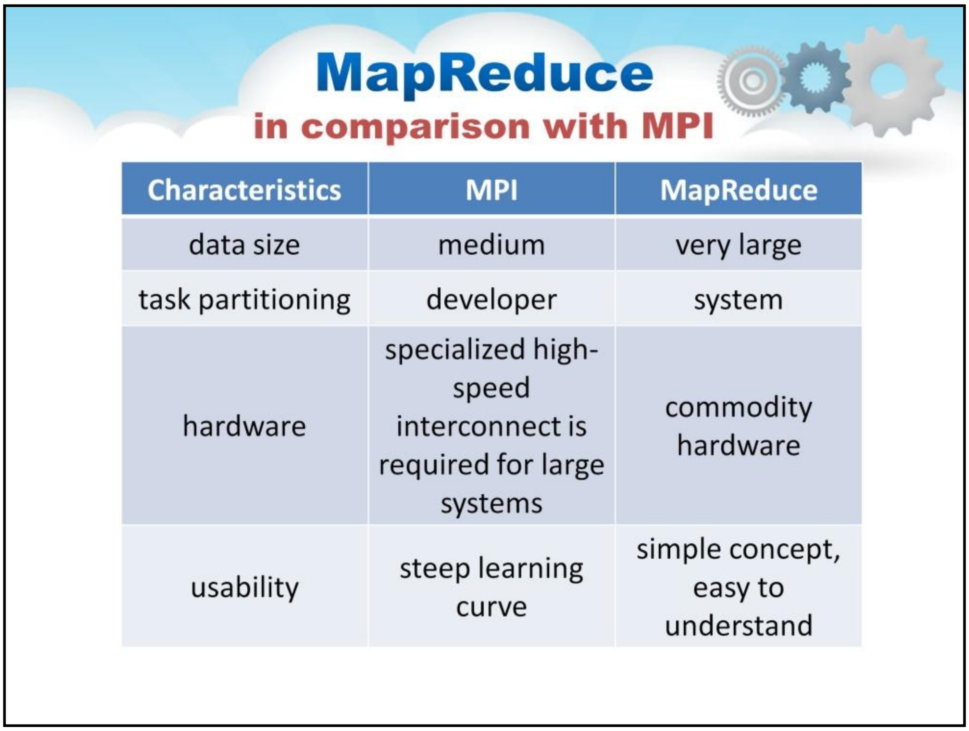
(data-local compute)

Data locality seems like a really straightforward idea, but the concept of not moving data over the network and processing it in place is actually pretty novel.

MapReduce frameworks, were among the first general-purpose and widely available
systems that took that approach.

It turns out that shipping the instructions for computation (i.e., a compiled program containing code for map and reduce functions, which is relatively small) over the network is much faster than shipping a petabyte of data to where the program is.

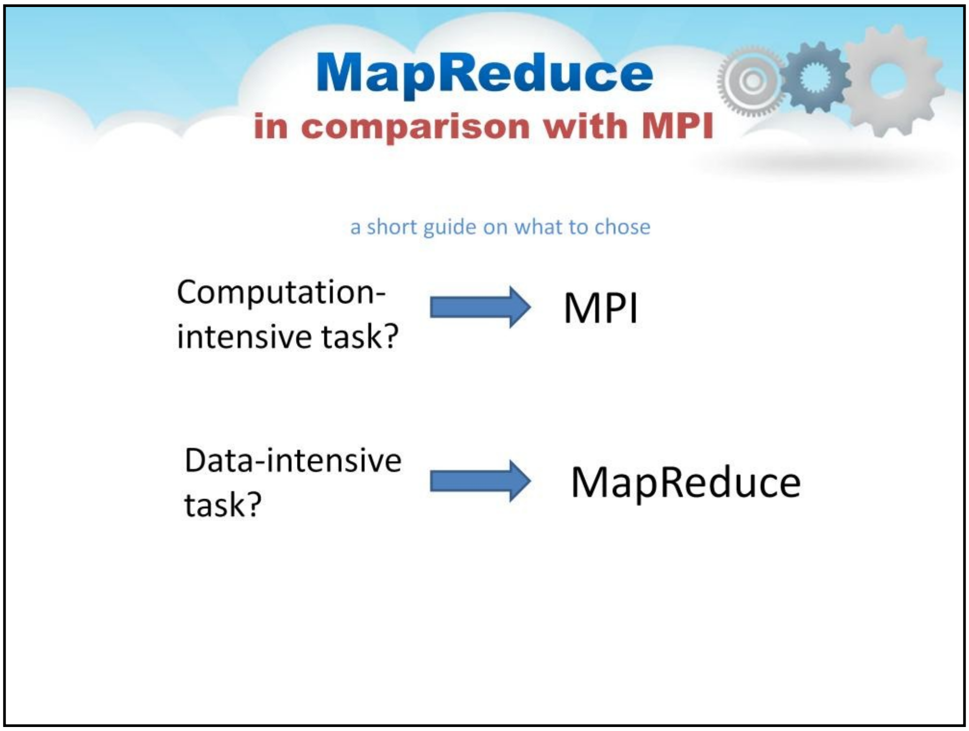
Data locality fundamentally removes the network as a bottleneck for linear scalability.



MapReduce
in comparison with MPI

Characteristics	MPI	MapReduce
data size	medium	very large
task partitioning	developer	system
hardware	specialized high-speed interconnect is required for large systems	commodity hardware
usability	steep learning curve	simple concept, easy to understand

Usually, the MapReduce is used to process huge amounts of data.
In MPI, task partitioning is a developer’s responsibility. In MapReduce, partitioning is automatic.
A large-scale MPI cluster won’t work well without high-speed interconnect. Map-reduce is easier to learn, while MPI is distinctly more complex with lots of functions.



MapReduce

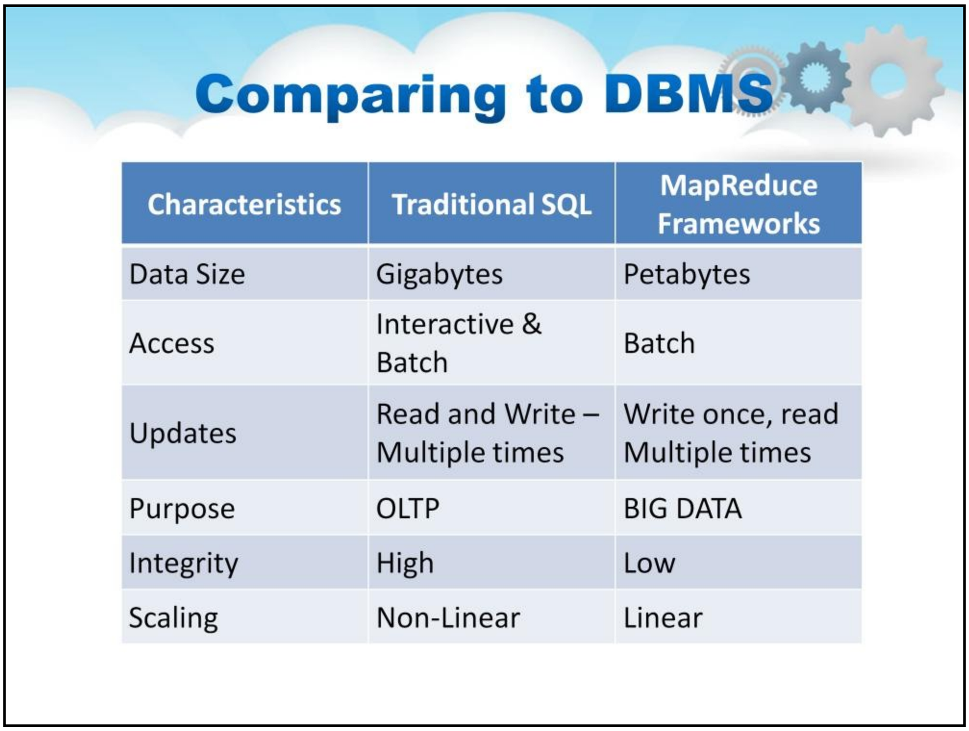
in comparison with MPI

a short guide on what to chose

Computation-intensive task? → MPI

Data-intensive task? → MapReduce

Summing up the above, map-reduce is more suitable for data-intensive task, while MPI is more appropriate for computation-intensive task.



Comparing to DBMS

Characteristics	Traditional SQL	MapReduce Frameworks
Data Size	Gigabytes	Petabytes
Access	Interactive & Batch	Batch
Updates	Read and Write – Multiple times	Write once, read Multiple times
Purpose	OLTP	BIG DATA
Integrity	High	Low
Scaling	Non-Linear	Linear

MapReduce and DBMS

First difference is a data size. A database management systems store and process gigabytes of data, while MapReduce deals with petabytes.

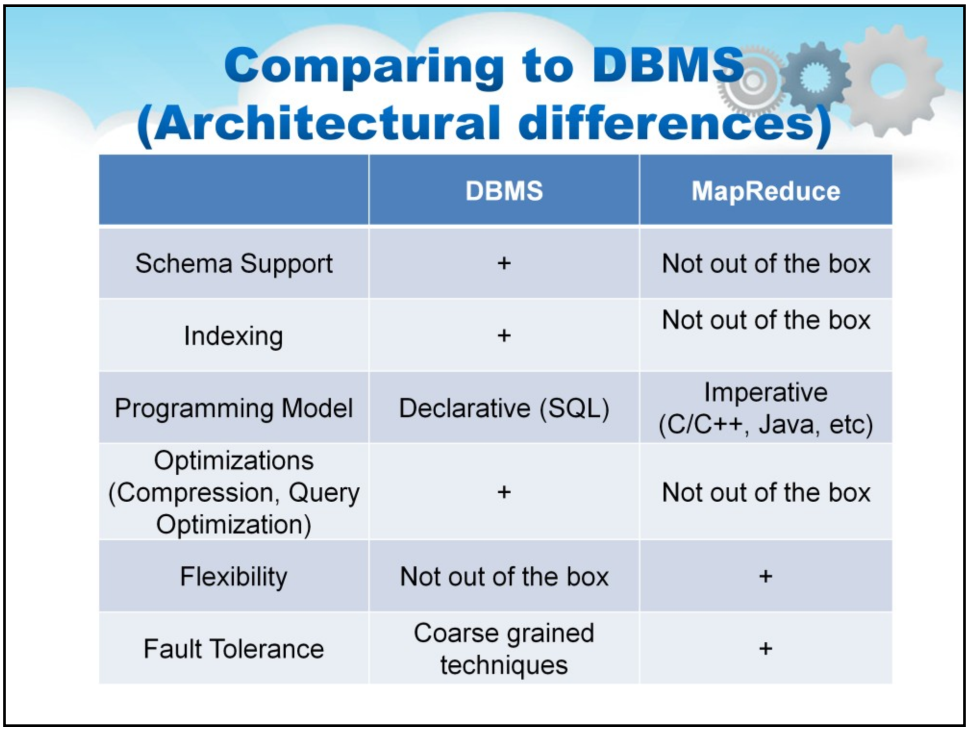
Second, traditional Database Management Systems allows both interactive and batch access, while MapReduce framework – only batch access.

Write, read and modify the same data multiple types are ordinary actions for DBMS, while MapReduce follows WORM principle (write once, read many times)

Next, DBMS is suitable for OLTP (online transaction processing), and MapReduce frameworks is good for BigData.

In DBMS, data integrity is high. Recall ACID (atomicity, consistency, isolation, and durability).

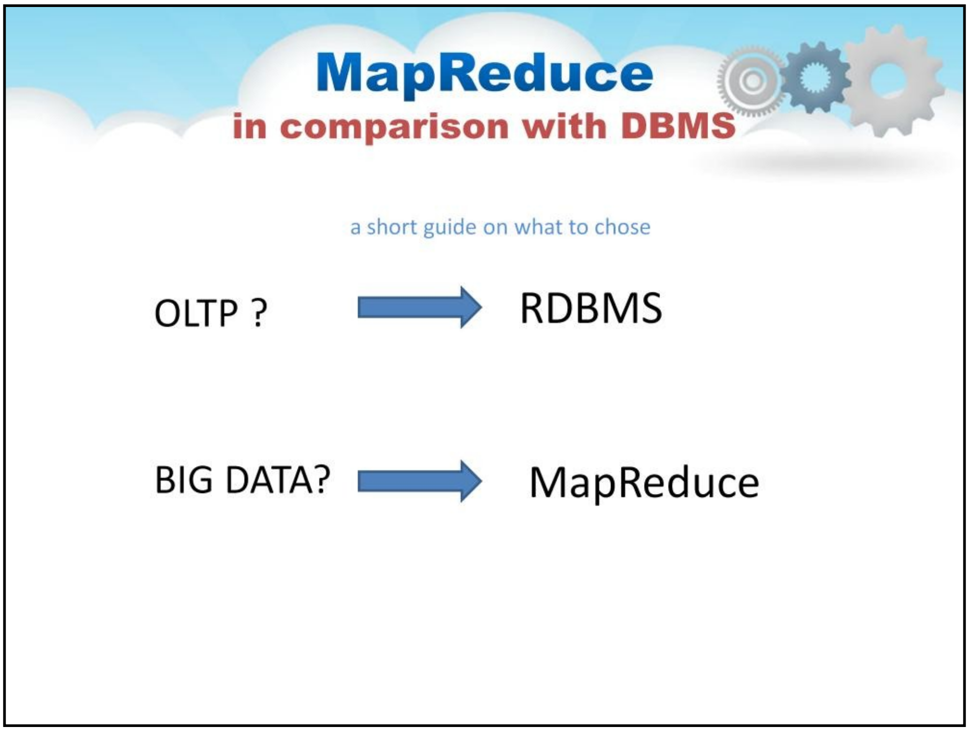
Scaling is an advantage of MapReduce. In the ideal case, MapReduce framework provides linear scaling.



Comparing to DBMS (Architectural differences)

	DBMS	MapReduce
Schema Support	+	Not out of the box
Indexing	+	Not out of the box
Programming Model	Declarative (SQL)	Imperative (C/C++, Java, etc)
Optimizations (Compression, Query Optimization)	+	Not out of the box
Flexibility	Not out of the box	+
Fault Tolerance	Coarse grained techniques	+

Schema-on -rite has been the standard for many years in relational databases. Before any data is written in the database, the structure of that data is strictly defined.
For MapReduce, schema is not defined when data is stored. When someone is ready
to use that data, they define what pieces are essential to their purpose. DBMS support indexing out of the box, while MapReduce frameworks not. Programming model is Declarative in DBMS (SQL language is used).
In MapReduce, programming model is imperative. You should define map and reduce functions (for example, Java code).



MapReduce
in comparison with DBMS

a short guide on what to chose

OLTP ? → RDBMS

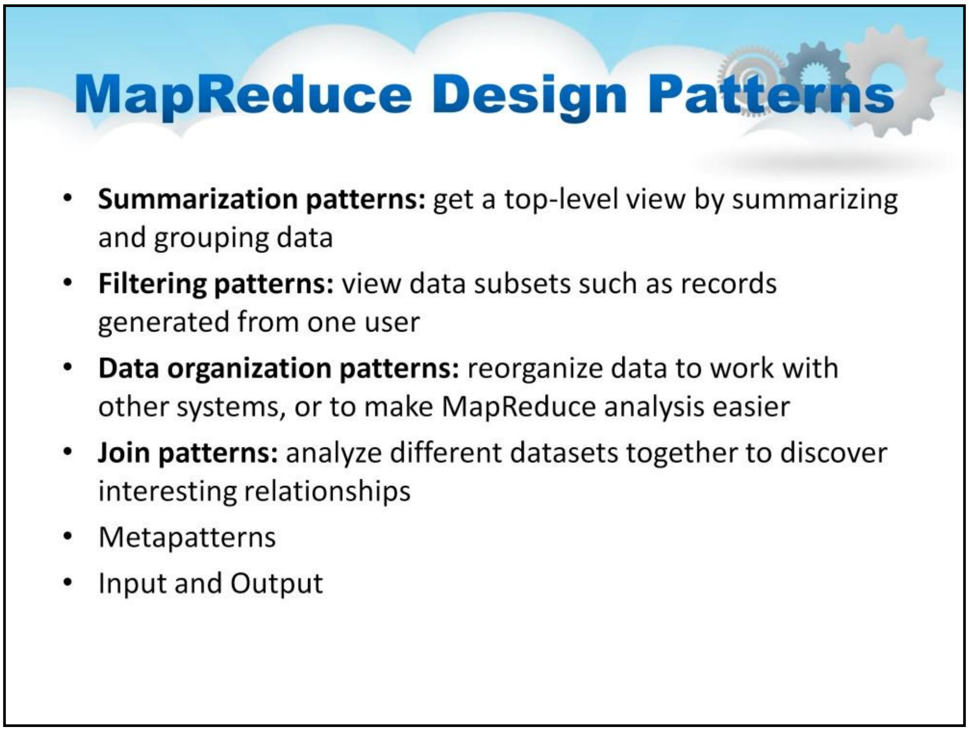
BIG DATA? → MapReduce

Summing up the above, Database Management System is more suitable for Online Transaction Processing, while MapReduce is good for Big Data Analysis.

MapReduce
Design Patterns

Design Patterns

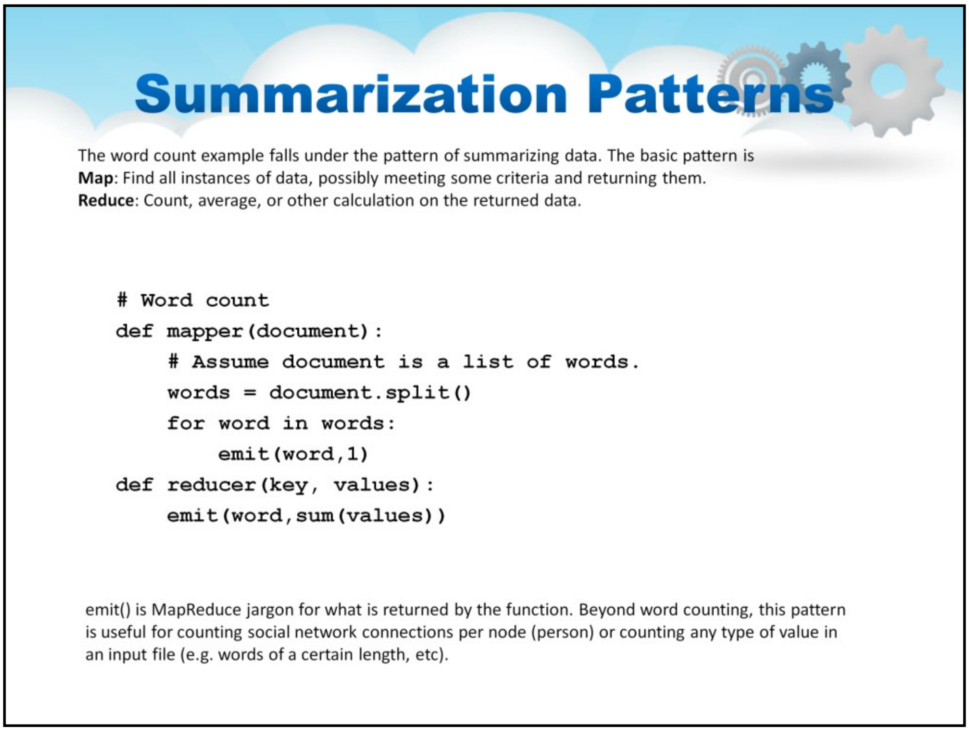
The MapReduce paradigm is simple and powerful, but it does not provide a general solution how to solve any problem in a big data field.
A design pattern (in general) is a reusable solution to a commonly occurring problem. So let's see how to solve some common types of tasks within the MapReduce paradigm.



MapReduce Design Patterns

- **Summarization patterns:** get a top-level view by summarizing and grouping data
- **Filtering patterns:** view data subsets such as records generated from one user
- **Data organization patterns:** reorganize data to work with other systems, or to make MapReduce analysis easier
- **Join patterns:** analyze different datasets together to discover interesting relationships
- Metapatterns
- Input and Output

Patterns can be divided into such categories:
Summarization patterns: get a top-level view by summarizing and grouping data
Filtering patterns: view data subsets such as records generated from one user
Data organization patterns: reorganize data to work with other systems, or to make
MapReduce analysis easier
Join patterns: analyze different datasets together to discover interesting relationships There are also **meta-patterns** and **input/output** patterns.



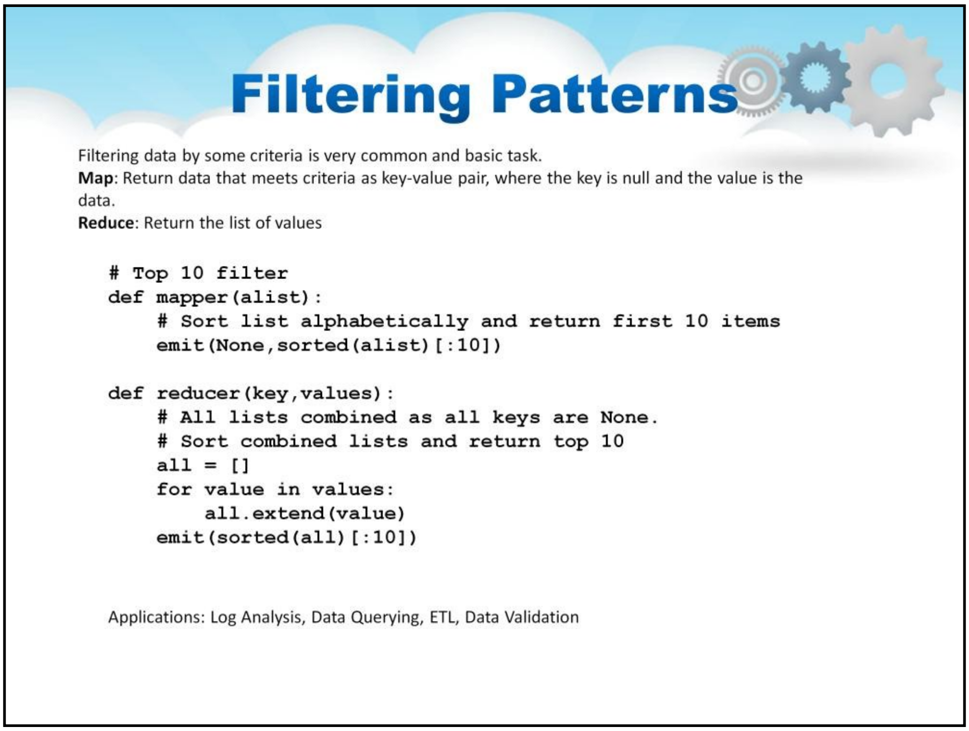
Summarization Patterns

The word count example falls under the pattern of summarizing data. The basic pattern is
Map: Find all instances of data, possibly meeting some criteria and returning them.
Reduce: Count, average, or other calculation on the returned data.

```
# Word count
def mapper(document):
    # Assume document is a list of words.
    words = document.split()
    for word in words:
        emit(word,1)
def reducer(key, values):
    emit(word,sum(values))
```

emit() is MapReduce jargon for what is returned by the function. Beyond word counting, this pattern is useful for counting social network connections per node (person) or counting any type of value in an input file (e.g. words of a certain length, etc).

Summarization Patterns.
The word count example (described earlier) falls under the pattern of
summarizing
data. The basic pattern is:
Map: Find all instances of data, possibly meeting some criteria and returning them.
Reduce: Count, average, or other calculation on the returned data.
Beyond word counting, this pattern is useful for counting social network connections per person, or counting any type of value in an input file (e.g. words of a certain length, etc).



Filtering Patterns

Filtering data by some criteria is very common and basic task.
Map: Return data that meets criteria as key-value pair, where the key is null and the value is the data.
Reduce: Return the list of values

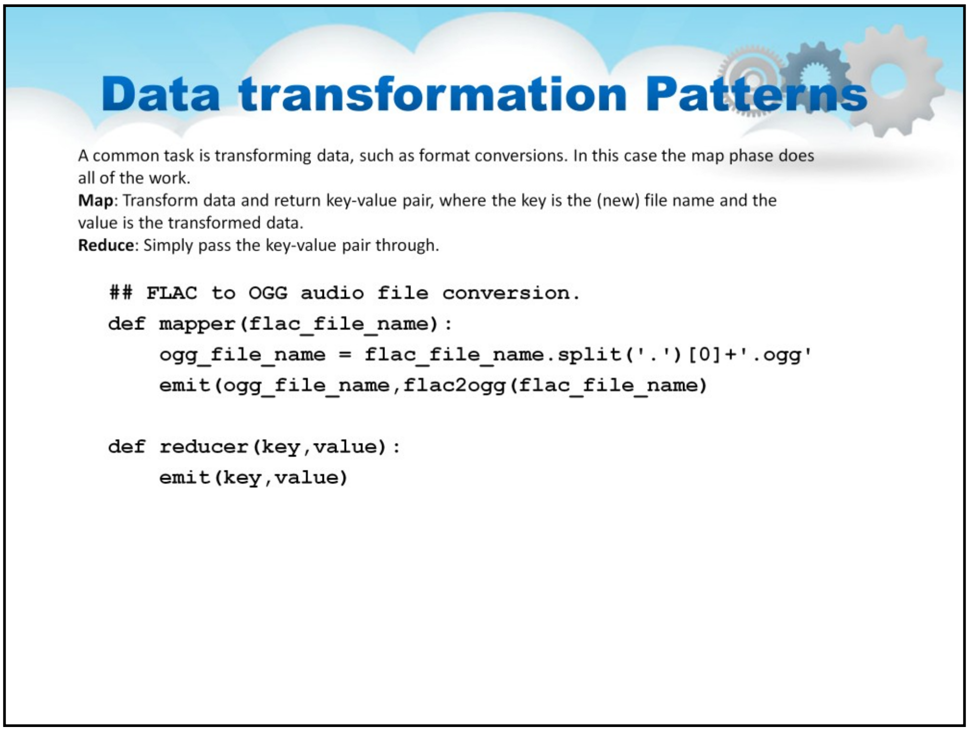
```
# Top 10 filter
def mapper(alist):
    # Sort list alphabetically and return first 10 items
    emit(None,sorted(alist)[:10])

def reducer(key,values):
    # All lists combined as all keys are None.
    # Sort combined lists and return top 10
    all = []
    for value in values:
        all.extend(value)
    emit(sorted(all)[:10])
```

Applications: Log Analysis, Data Querying, ETL, Data Validation

Filtering patterns
Filtering data by some criteria is very common and basic task.
Map: Return data that meets criteria as key-value pair, where the key is null and the
value is the data.
Reduce: Return the list of values

Example applications are: Log Analysis, Data Querying, ETL, Data Validation



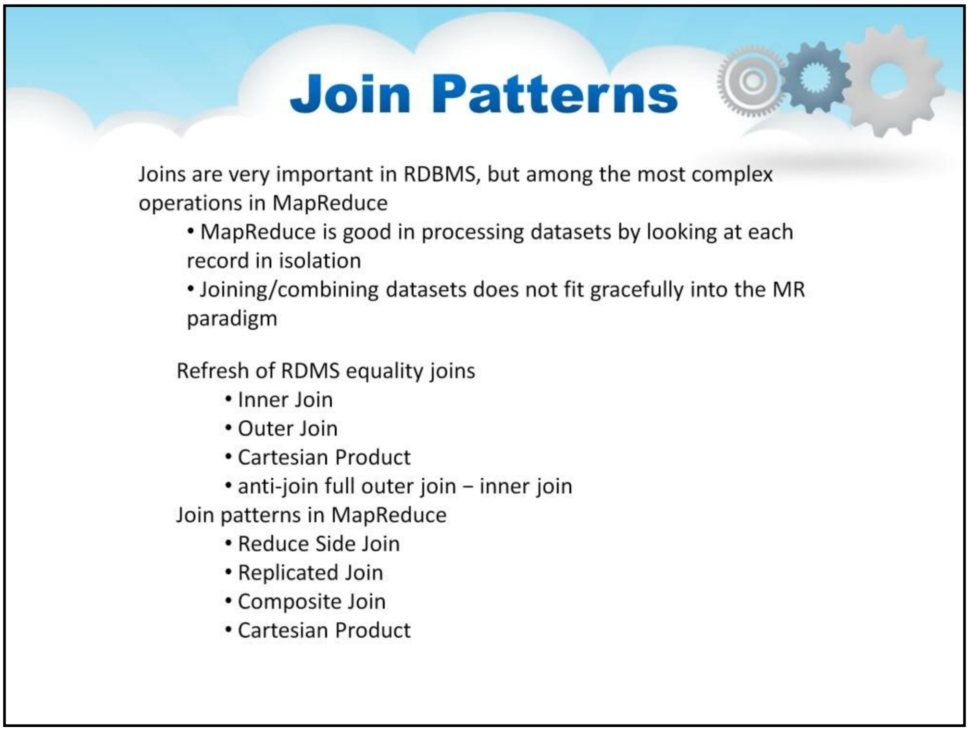
Data transformation Patterns

A common task is transforming data, such as format conversions. In this case the map phase does all of the work.
Map: Transform data and return key-value pair, where the key is the (new) file name and the value is the transformed data.
Reduce: Simply pass the key-value pair through.

```
## FLAC to OGG audio file conversion.
def mapper(flac_file_name):
    ogg_file_name = flac_file_name.split('.')[0]+'.ogg'
    emit(ogg_file_name,flac2ogg(flac_file_name)

def reducer(key,value):
    emit(key,value)
```

A common task is transforming data, such as format conversions. In this case the map phase does all of the work.
Map: Transform data and return key-value pair, where the key is the (new) file name
and the value is the transformed data.
Reduce: Simply pass the key-value pair through.



Join Patterns

Joins are very important in RDBMS, but among the most complex operations in MapReduce

- MapReduce is good in processing datasets by looking at each record in isolation
- Joining/combining datasets does not fit gracefully into the MR paradigm

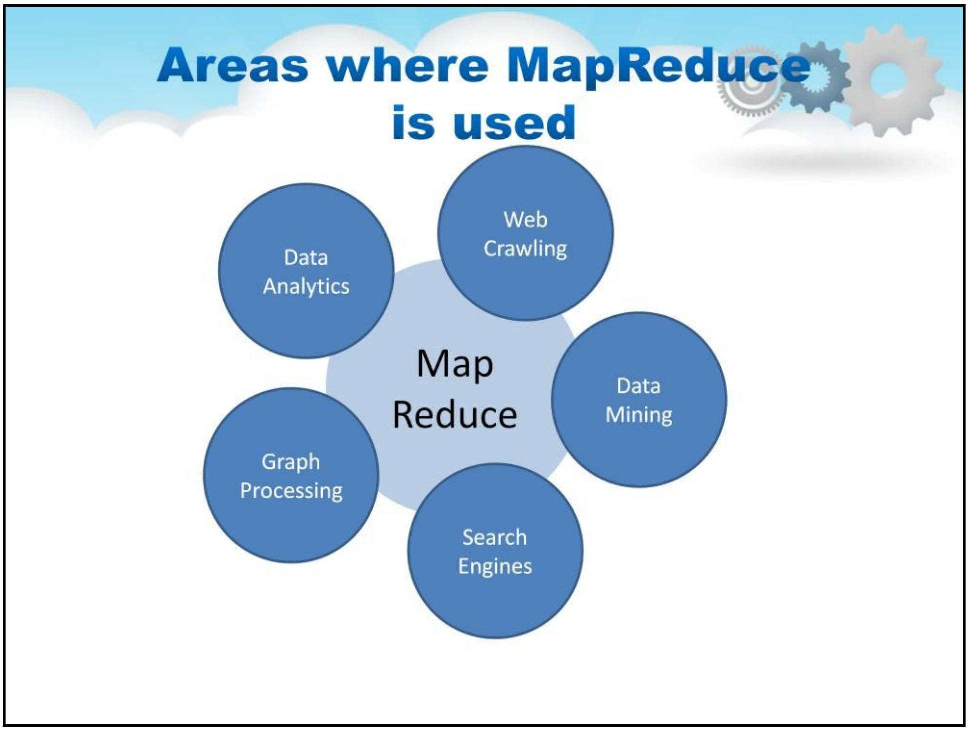
Refresh of RDMS equality joins

- Inner Join
- Outer Join
- Cartesian Product
- anti-join full outer join – inner join

Join patterns in MapReduce

- Reduce Side Join
- Replicated Join
- Composite Join
- Cartesian Product

Joins are very important in Relational Database Management Systems, but among the most complex operations in MapReduce .
MapReduce is good in processing datasets by looking at each record in isolation, but
joining/combining datasets does not fit gracefully into the MapReduce paradigm.
A detailed review of this topic is beyond the scope of this course.
For more details please refer, for example, this book
“MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems”



Areas where MapReduce is used

Map Reduce

Data Analytics

Web Crawling

Data Mining

Search Engines

Graph Processing

Applications

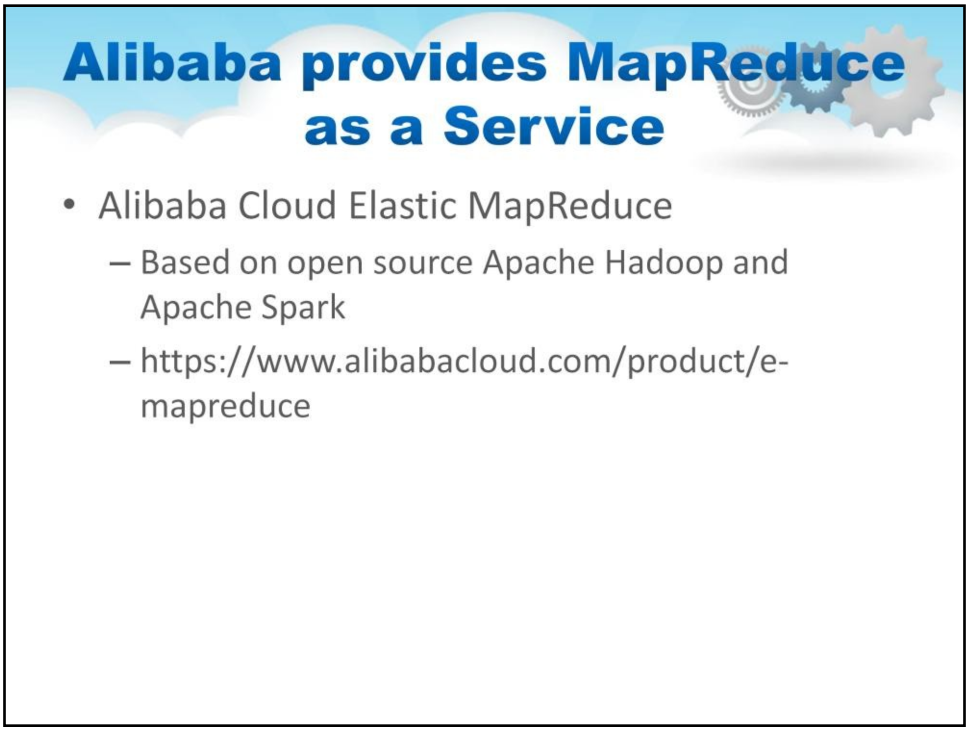
Data Analytics:
MapReduce is one of the most preferred solution for data analytics. Now most common data analytics include things like calculating and analyzing unique clicks or visitors on a website, finding top visited/searched products per month on an e-commerce website.

Crawling of data:
There are services and projects which crawl the data on the internet for example applications which crawl twitter data and to find some facts or reach to some conclusions. These kind of application need to deal with a large amount of data. MapReduce is a good solution for crawling and processing this kind of data.

Data Mining:
Data Mining is another kind of problem where MapReduce is used. For example, we need to mine a large amount of data (lets say the whole archives of a newspaper company) and person some kind of text classification or clustering on those archives. Again here also if the amount of data is large which can't be processed on a single system, MapReduce is used on top of Hadoop Cluster.

Search Engines:
Google was the company which developed MapReduce first and then based on the Google's research paper, open source version called Hadoop MapReduce was developed. Most probably Google developed MapReduce for text-indexing. Now obviously, Google or any such company need to retrieve information from the large amount of data/documents they have and the results should be quick. So these kind of data need to be indexed to improve the performance. MapReduce is used for creating inverted indexes of those documents etc. Inverted Index is a data structure which stores the map from the content to its location in the file/document. MapReduce is used for creating such inverted indexes.

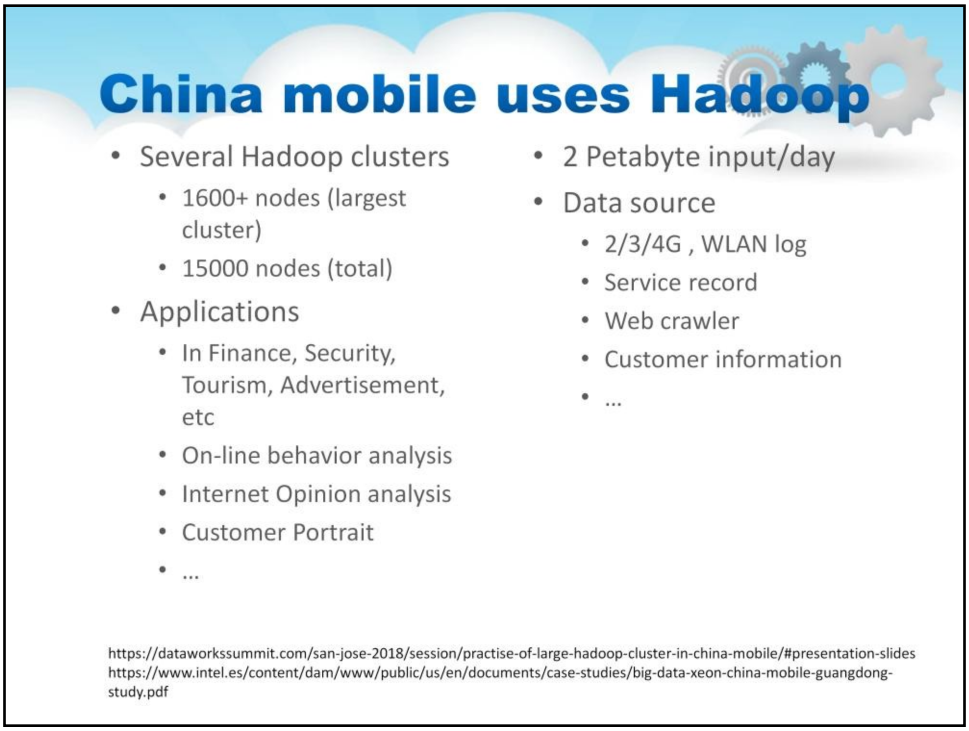
Graph Processing:
Lets assume there is a network of entities and relationships between them. Now we need to calculate a state of each entity on the basis of properties of the neighboring entities. This problem is also termed as Graph Processing or Graph Analysis which is most of the times used in Social Network analysis. MapReduce can be used for this type of graph processing also as social network graph tends to be quite big



Alibaba provides MapReduce as a Service

- Alibaba Cloud Elastic MapReduce
  - Based on open source Apache Hadoop and Apache Spark
  - https://www.alibabacloud.com/product/e-mapreduce

Alibaba Cloud Elastic MapReduce (E-MapReduce) is a big data processing solution to quickly process huge amounts of data.
Based on open source Apache Hadoop and Apache Spark, E-MapReduce flexibly manages your big data use cases such as trend analysis, data warehousing, and analysis of continuously streaming data.
E-MapReduce simplifies big data processing, making it easy, fast, scalable and cost- effective for you to provision distributed Hadoop clusters and process your data.
This helps you to streamline your business through better decisions based on massive data analysis completed in real time.

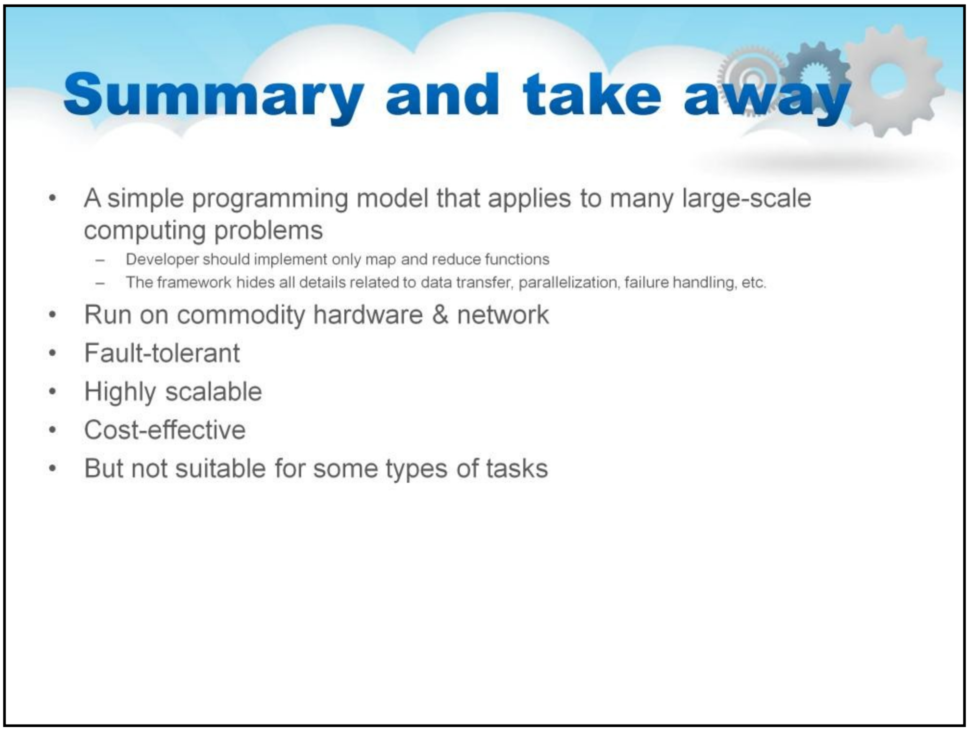


China mobile uses Hadoop

- Several Hadoop clusters
  - 1600+ nodes (largest cluster)
  - 15000 nodes (total)
- Applications
  - In Finance, Security, Tourism, Advertisement, etc
  - On-line behavior analysis
  - Internet Opinion analysis
  - Customer Portrait
  - …
- 2 Petabyte input/day
- Data source
  - 2/3/4G , WLAN log
  - Service record
  - Web crawler
  - Customer information
  - …

https://dataworkssummit.com/san-jose-2018/session/practise-of-large-hadoop-cluster-in-china-mobile/#presentation-slides
https://www.intel.es/content/dam/www/public/us/en/documents/case-studies/big-data-xeon-china-mobile-guangdong-study.pdf

Some reports and conference materials mention that China Mobile deployed a large Hadoop cluster.
While deployment, the company's engineers encountered some challenges and
successfully overcame it.
In the near future the number of cluster nodes will increase to 14,000.

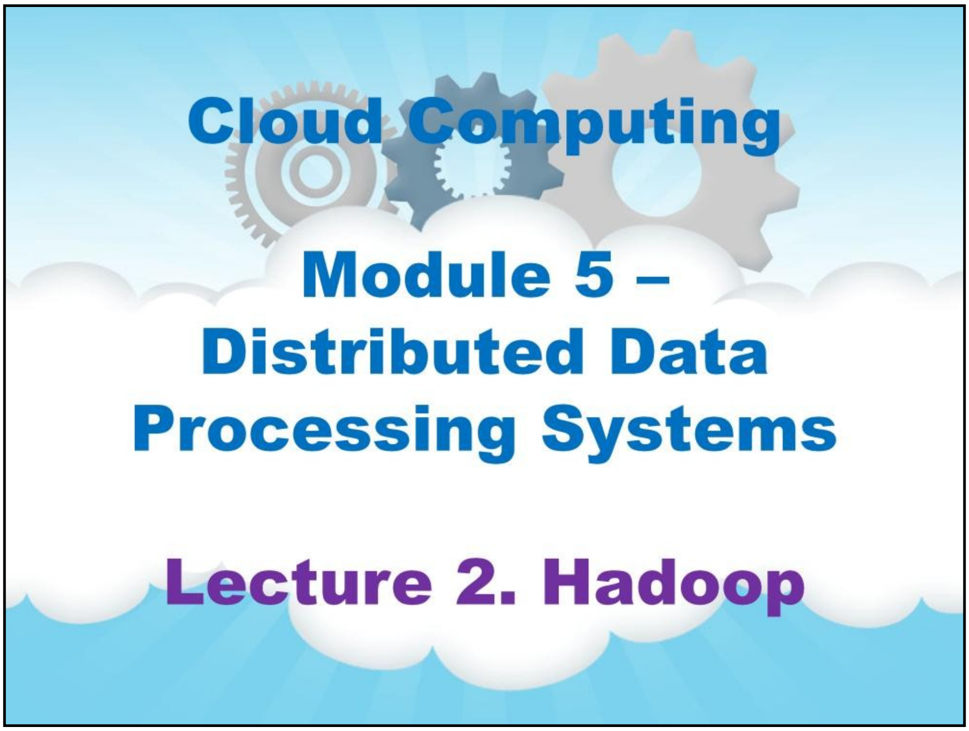


Summary and take away

- A simple programming model that applies to many large-scale computing problems
  - Developer should implement only map and reduce functions
  - The framework hides all details related to data transfer, parallelization, failure handling, etc.
- Run on commodity hardware & network
- Fault-tolerant
- Highly scalable
- Cost-effective
- But not suitable for some types of tasks

In this Lesson we covered

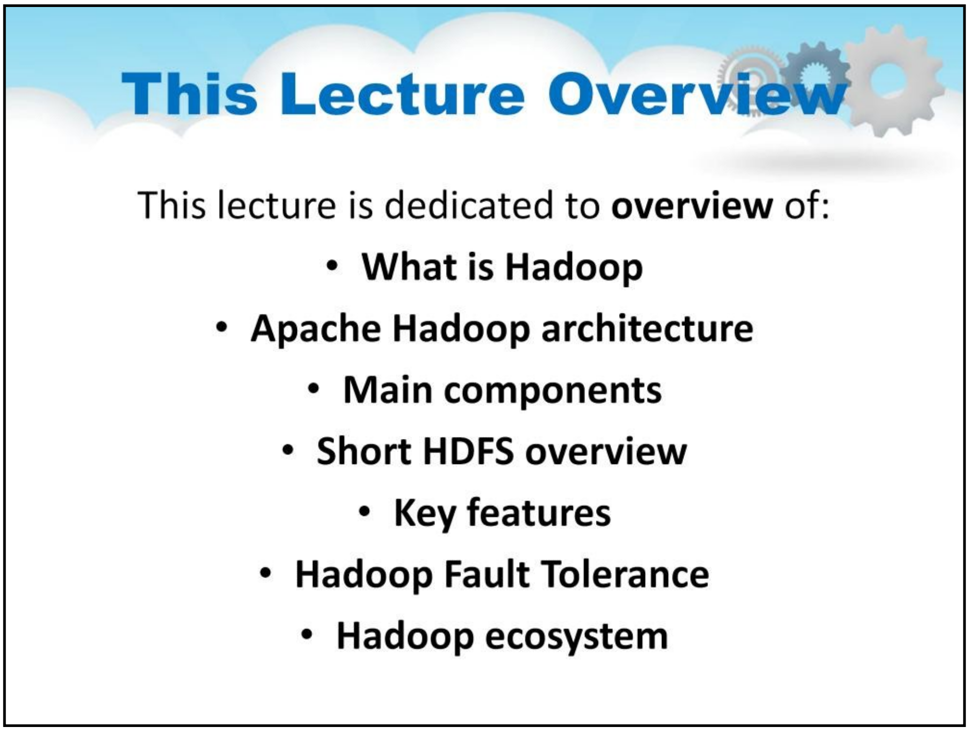
MapReduce is a simple programming model and an associated implementation for
processing and generating large data-sets.
Developer should implement only two functions- Map and Reduce.
Modern MapReduce frameworks is highly scalable (up to 10000 nodes and more), fault-tolerant, can run on commodity (cheap) hardware.
MapReduce is not a “silver bullet”. There are some cases where MapReduce doesn’t work very well.



Cloud Computing

Module 5 – Distributed Data Processing Systems

Lecture 2. Hadoop



This Lecture Overview

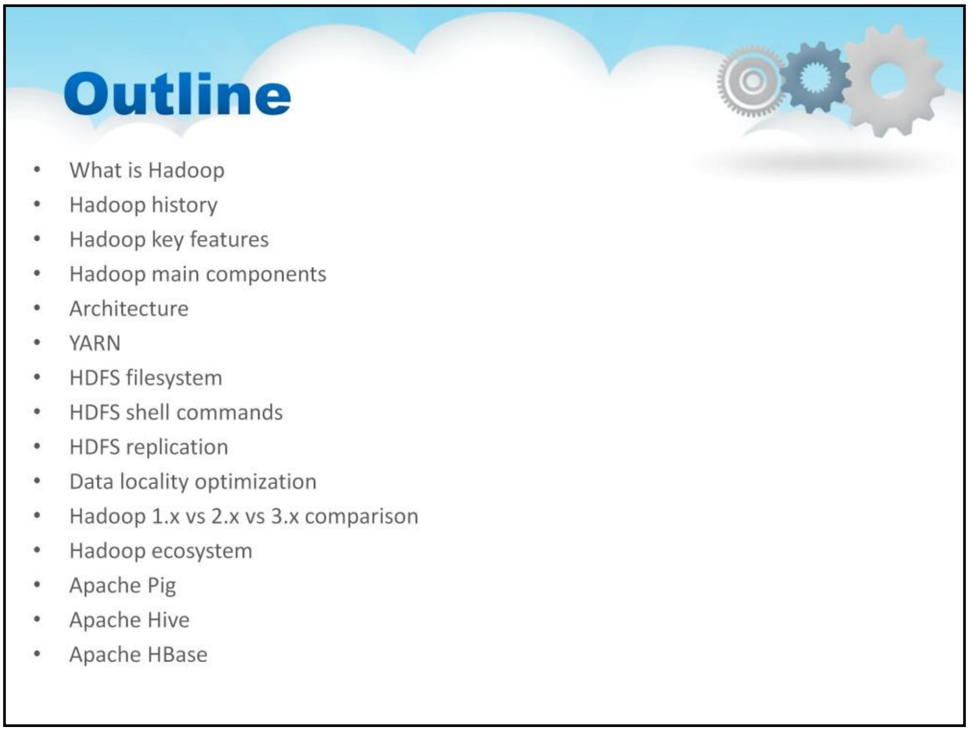
This lecture is dedicated to **overview** of:

- **What is Hadoop**
- **Apache Hadoop architecture**
- **Main components**
- **Short HDFS overview**
- **Key features**
- **Hadoop Fault Tolerance**
- **Hadoop ecosystem**

Lecture 2. Hadoop

This lecture is dedicated to overview of:

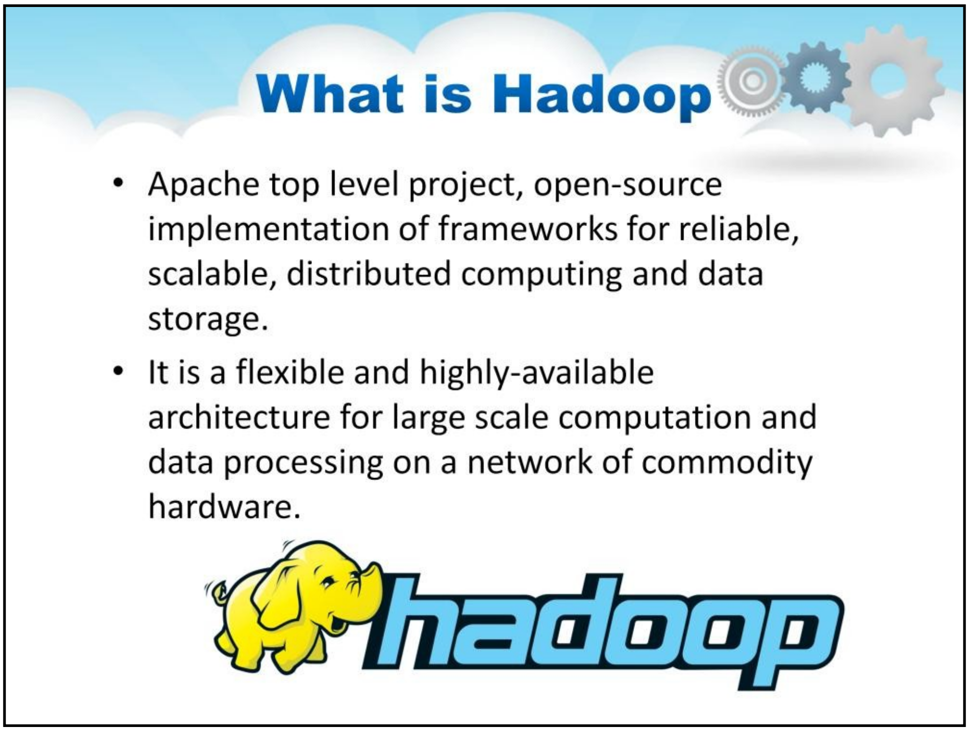
- What is Hadoop
- Apache Hadoop architecture
- Main components
- Short HDFS overview
- Key features
- Hadoop Fault Tolerance
- Hadoop ecosystem



Outline

- What is Hadoop
- Hadoop history
- Hadoop key features
- Hadoop main components
- Architecture
- YARN
- HDFS filesystem
- HDFS shell commands
- HDFS replication
- Data locality optimization
- Hadoop 1.x vs 2.x vs 3.x comparison
- Hadoop ecosystem
- Apache Pig
- Apache Hive
- Apache HBase

Outline
What is Hadoop
Hadoop history
Hadoop key
features
Hadoop main
components
Architecture
YARN
HDFS filesystem
HDFS shell
commands HDFS
replication
Data locality optimization
Hadoop 1.x vs 2.x vs 3.x
comparison Hadoop
ecosystem
Apache Pig
Apache
Hive
Apache
HBase

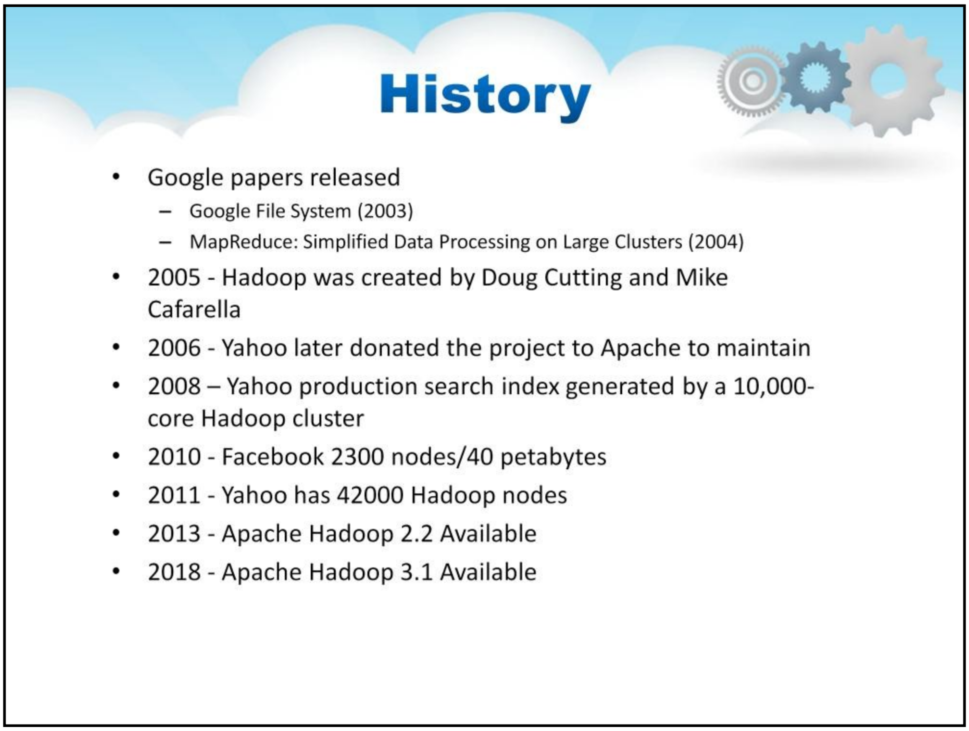


What is Hadoop

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

Overview

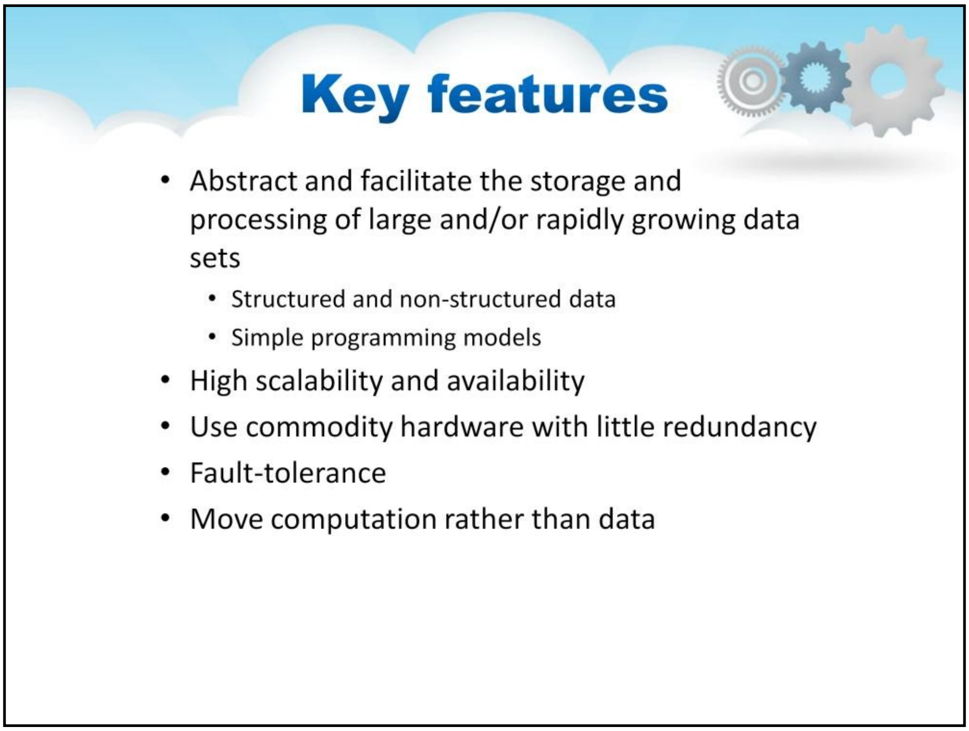
Apache Hadoop is a top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.



History

- Google papers released
  - Google File System (2003)
  - MapReduce: Simplified Data Processing on Large Clusters (2004)
- 2005 - Hadoop was created by Doug Cutting and Mike Cafarella
- 2006 - Yahoo later donated the project to Apache to maintain
- 2008 – Yahoo production search index generated by a 10,000-core Hadoop cluster
- 2010 - Facebook 2300 nodes/40 petabytes
- 2011 - Yahoo has 42000 Hadoop nodes
- 2013 - Apache Hadoop 2.2 Available
- 2018 - Apache Hadoop 3.1 Available

Doug Cutting and Mike Cafarella, the founders of Hadoop, were inspired by Google's GFS and MapReduce architecture.
Google introduced GFS (Google File System) system for storing huge size of data sets in a distributed manner in a cluster of commodity hardware's and MapReduce the technology for processing the data sets present in these distributed systems.
In 2004, Google published its GFS and MapReduce white papers. Doug Cutting and Mike Cafarella, who were then working in Nutch, got inspired by Google's technology and started to build their own search engine called Nutch, on top of the Google's file system and MapReduce technology.
Hadoop was originally developed to support distribution for the Nutch search engine project.
And in 2006, Yahoo! hired Doug Cutting and introduced Hadoop framework to the world named the project after his son's toy elephant.
Now, with the help of this framework, many organizations have started analyzing huge data sets which remained unresolved for many decades.

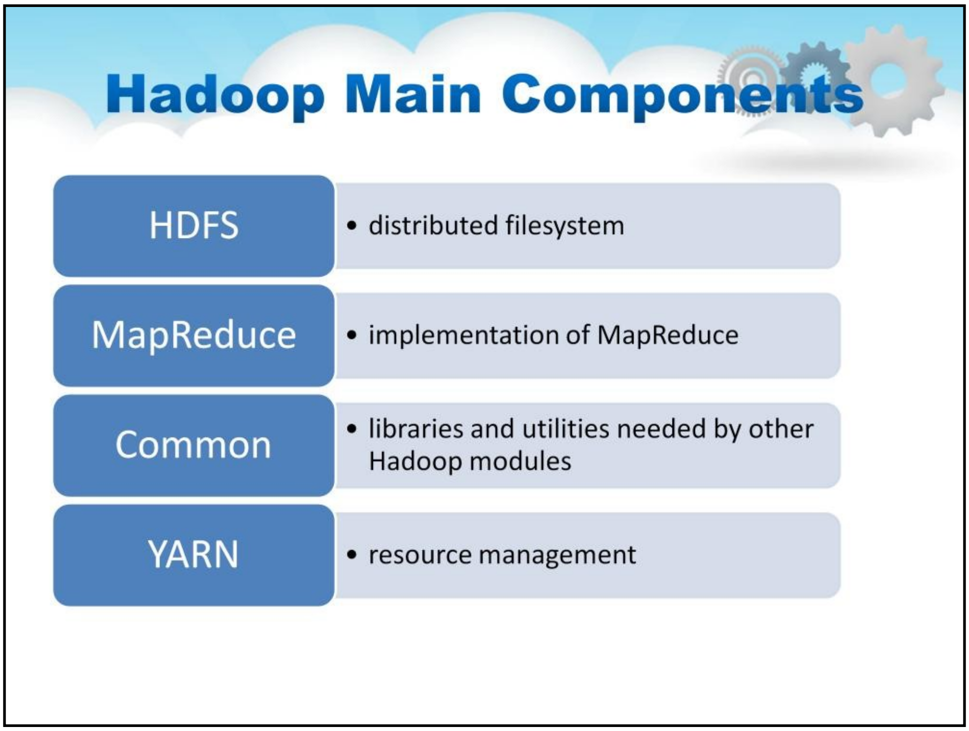


Key features

- Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
  - Structured and non-structured data
  - Simple programming models
- High scalability and availability
- Use commodity hardware with little redundancy
- Fault-tolerance
- Move computation rather than data

Key Apache Hadoop features

- Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
  - Structured and non-structured data
  - Simple programming models
- High scalability and availability
- Use commodity (cheap) hardware with little redundancy
- Fault-tolerance
- Move computation rather than data

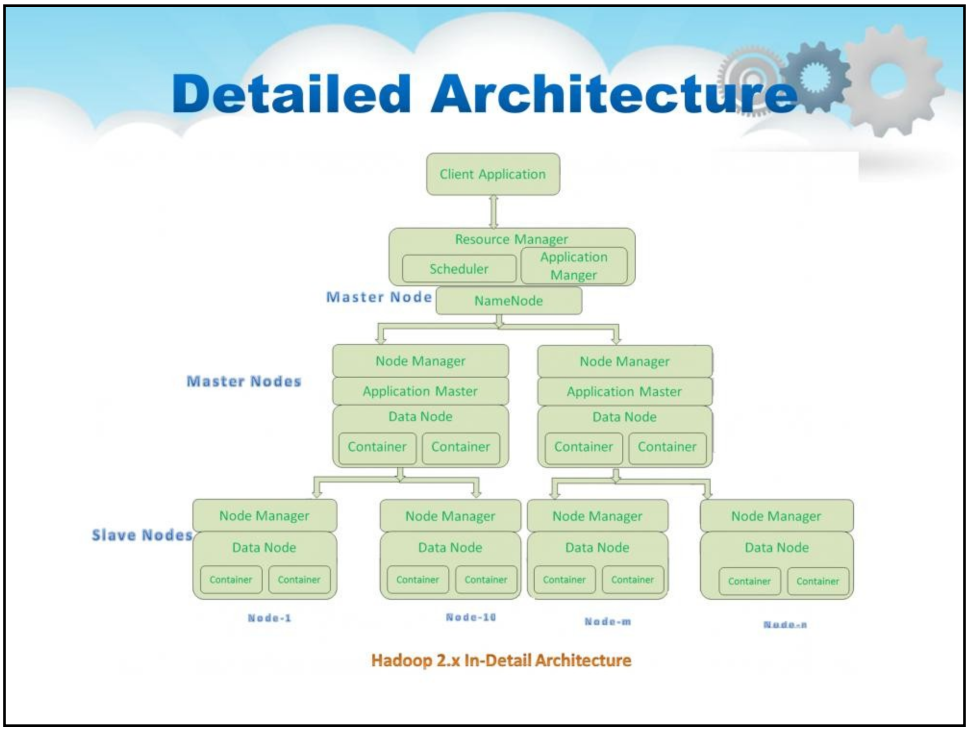


Hadoop Main Components

HDFS	• distributed filesystem
MapReduce	• implementation of MapReduce
Common	• libraries and utilities needed by other Hadoop modules
YARN	• resource management

Components of Hadoop
The core components in the first iteration of Hadoop were MapReduce, the Hadoop Distributed File System (HDFS) and Hadoop Common, a set of shared utilities and libraries. As its name indicates, MapReduce uses map and reduce functions to split processing jobs into multiple tasks that run at the cluster nodes where data is stored and then to combine what the tasks produce into a coherent set of results. MapReduce initially functioned as both Hadoop's processing engine and cluster resource manager, which tied HDFS directly to it and limited users to running MapReduce batch applications.
That changed in Hadoop 2.0, which became generally available in October 2013 when version 2.2.0 was released. It introduced Apache Hadoop YARN, a new cluster resource management and job scheduling technology that took over those functions from MapReduce. YARN -- short for Yet Another Resource Negotiator but typically referred to by the acronym alone -- ended the strict reliance on MapReduce and opened up Hadoop to other processing engines and various applications besides batch jobs.

The Hadoop 2.0 series of releases also added high availability (HA) and federation features for HDFS, support for running Hadoop clusters on Microsoft Windows servers and other capabilities designed to expand the distributed processing framework's versatility for big data management and analytics.
Hadoop 3.0.0 was the next major version of Hadoop. Released by Apache in December 2017, it didn't expand Hadoop's set of core components. However, it added a YARN Federation feature designed to enable YARN to support tens of thousands of nodes or more in a single cluster, up from a previous 10,000-node limit. The new version also included support for GPUs and erasure coding, an alternative to data replication that requires significantly less storage space.



Detailed Architecture

Hadoop 2.x In-Detail Architecture

Architecture

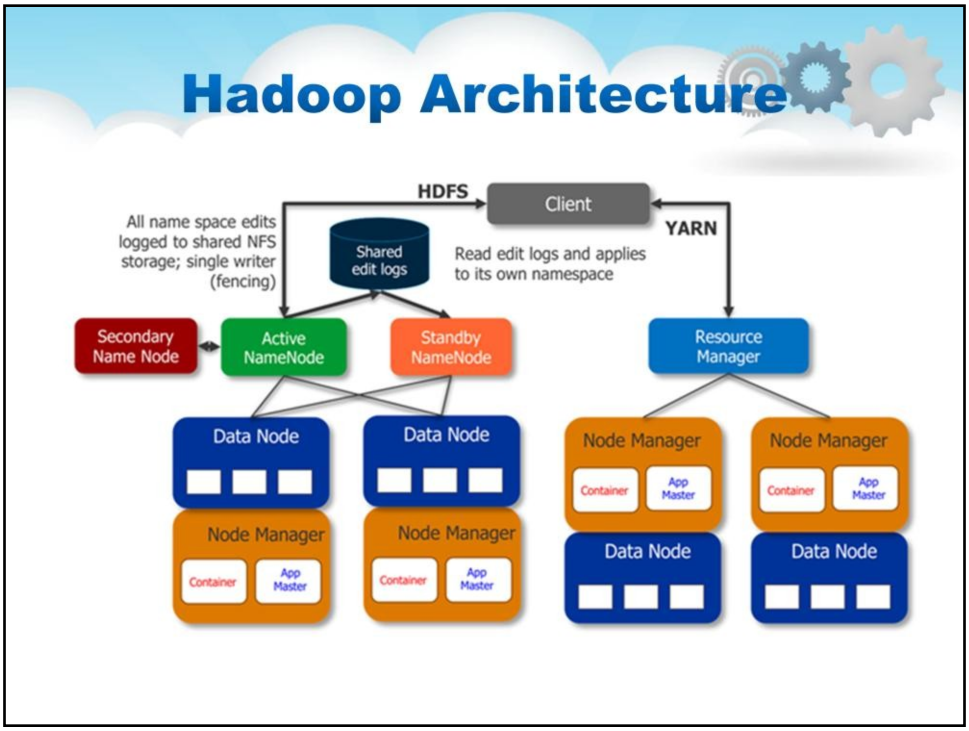
Resource Manager is a Per-Cluster Level Component. Resource Manager is divided into two components: Scheduler and Application Manager.
Resource Manager's Scheduler is responsible to schedule required resources to Applications (that is Per- Application Master).
It does only scheduling, and it does care about monitoring or tracking of those Applications.
Application Master is a per-application level component. It is responsible for:

- Managing assigned Application Life cycle.
- It interacts with both Resource Manager's Scheduler and Node Manager
- It interacts with Scheduler to acquire required resources.
- It interacts with Node Manager to execute assigned tasks and monitor those task's status.

Node Manager is a Per-Node Level component, and is responsible for:

- Managing the life-cycle of the Container.
- Monitoring each Container's Resources utilization.

Each Master Node or Slave Node contains set of **Containers**.
Container is a portion of Memory in HDFS (Either Name Node or Data Node).



Hadoop Architecture

In Hadoop 2.0, the Job Tracker in YARN mainly depends on 3 important components

1. Resource Manager Component:

This component is considered as the negotiator of all the resources in the cluster. Resource Manager is further categorized into an Application Manager that will manage all the user jobs with the cluster and a pluggable scheduler. This is a relentless YARN service that is designed for receiving and running the applications on the Hadoop Cluster. In Hadoop 2.0, a MapReduce job will be considered as an application.

2. Node Manager Component:

This is the job history server component of YARN which will furnish the information about all the completed jobs. The NM keeps a track of all the users' jobs and their workflow on any particular given node.

3. Application Master Component (aka User Job Life Cycle Manager):

This is the component where the job actually resides and the Application Master component is responsible for managing each and every Map Reduce job and is concluded once the job completes processing.

RM-Resource Manager

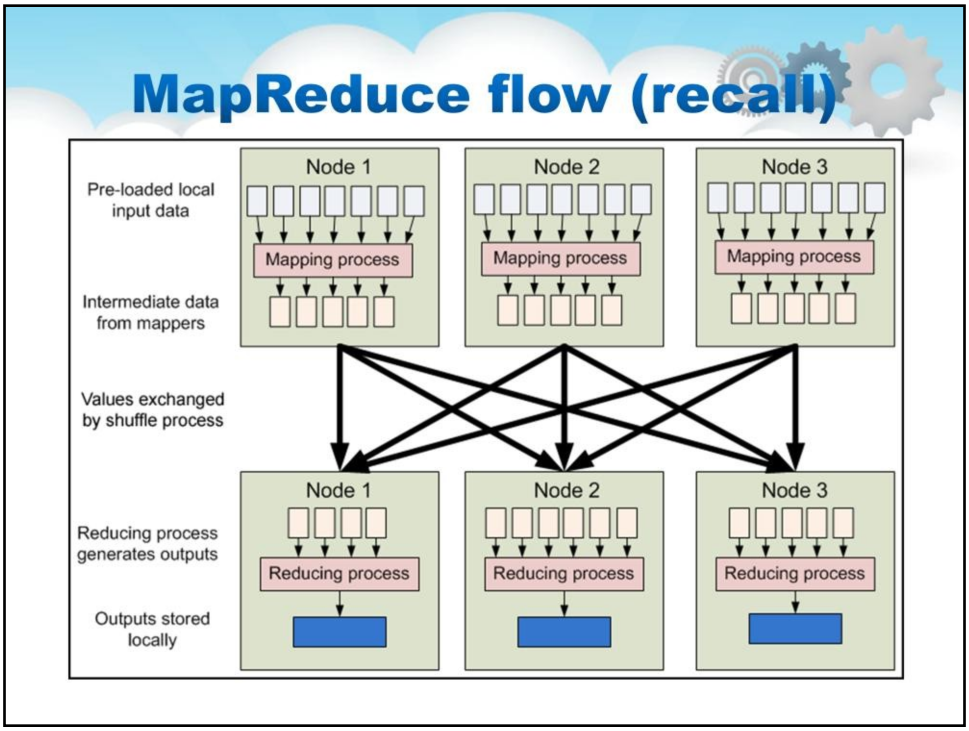
1. It is the global resource scheduler
2. It runs on the Master Node of the Cluster
3. It is responsible for negotiating the resources of the system amongst the competing applications.
4. It keeps a track on the heartbeats from the Node Manager

NM-Node Manager

1.Node Manager communicates with the resource manager. 2.It runs on the Slave Nodes of the Cluster

AM-Application Master

1.There is one AM per application which is application specific or framework specific. 2.The AM runs in Containers that are created by the resource manager on request.



MapReduce flow (recall)

Pre-loaded local input data

Intermediate data from mappers

Values exchanged by shuffle process

Reducing process generates outputs

Outputs stored locally

Node 1

Mapping process

Node 2

Mapping process

Node 3

Mapping process

Node 1

Reducing process

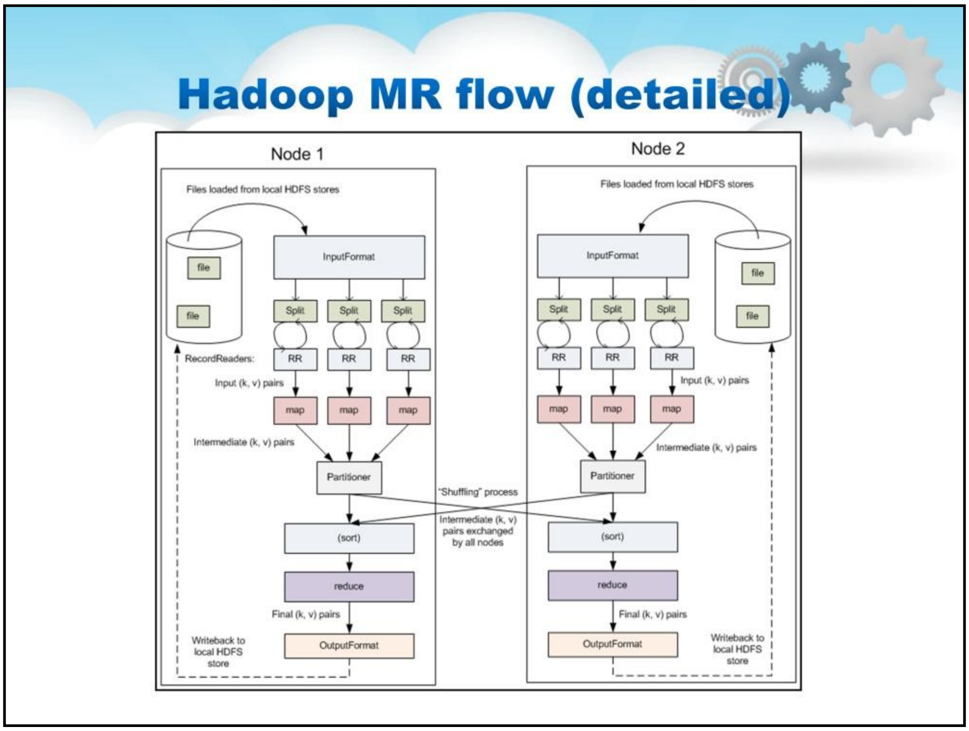
Node 2

Reducing process

Node 3

Reducing process

In the previous lecture we discussed the MapReduce programming model.
Let’s recall the main points of MapReduce dataflow.
Hadoop limits the amount of communication which can be performed by the processes, as each individual record is processed by a task in isolation from one another.
While this sounds like a major limitation at first, it makes the whole framework much more reliable. Hadoop will not run just any program and distribute it across a cluster.
Programs must be written to conform to a particular programming model, named "MapReduce." In MapReduce, records are processed in isolation by tasks called *Mappers*.
The output from the Mappers is then brought together into a second set of tasks called *Reducers*, where results from different mappers can be merged together.



Hadoop MR flow (detailed)

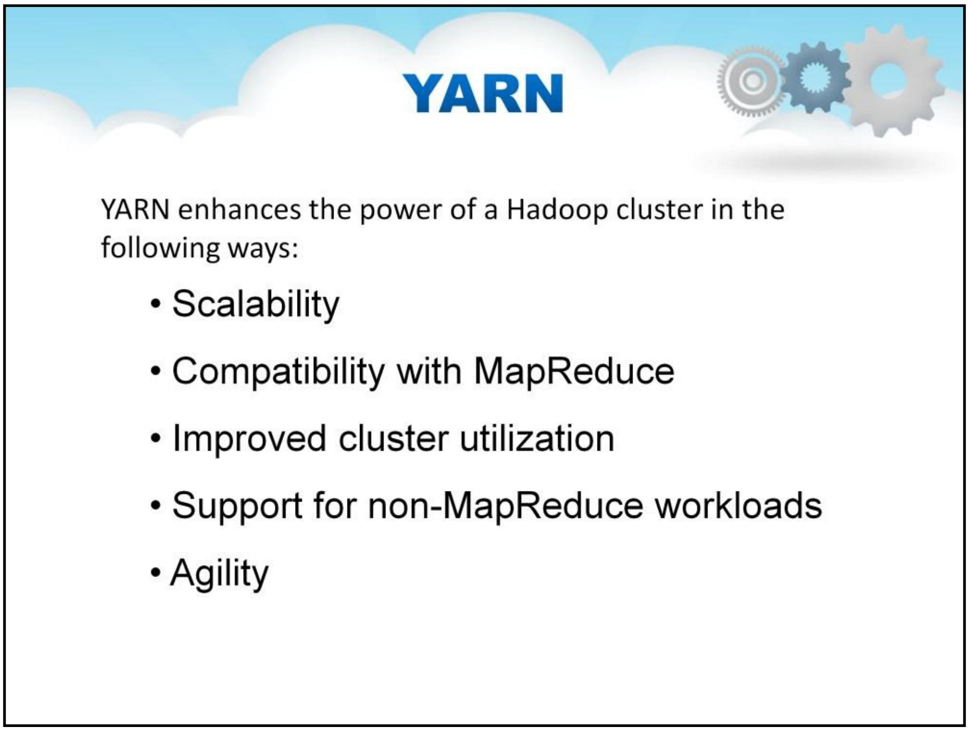
You can see components of the example "Word Count" application on this detailed diagram.
Input files: Here is the data for a task is initially stored (typically in HDFS).
InputFormat defines how input files are split and read. Several InputFormats are provided with Hadoop. **InputSplits**: describes a unit of work that comprises a single map task in a MapReduce program. Map tasks may read a whole file or just only part of a file.
The **InputSplit** defines a slice of work, but does not describe how to access it. The **RecordReader** actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper.
The **Mapper** performs user-defined work of the Map phase. Given a key and a value, the map() method emits (key, value) pair(s) which are forwarded to the **Reducers**.
After the first map tasks have completed, the nodes may still be performing several more map tasks each. But they also begin exchanging the intermediate outputs from the map tasks to where they are required by the reducers. This process of moving map outputs to the reducers is known as **shuffling**.
Each reduce task is responsible for reducing the values associated with several intermediate keys. The set of intermediate keys on a single node is automatically sorted by Hadoop before they are presented to the **Reducer**. **Reducer** is an instance of user-provided code that performs the Reduce phase.
The way how (key, value) pairs they are written is defined by the **OutputFormat**.
The OutputFormat functions is like the InputFormat class described earlier.
The **OutputFormat** class is a factory for **RecordWriter** objects; these are used to write the individual records to the files as defined by the **OutputFormat**.
The **output files** written by the Reducers are then stored in HDFS.

YARN

overview

YARN

Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework. One of Apache Hadoop's core components, YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.



YARN

YARN enhances the power of a Hadoop cluster in the following ways:

- Scalability
- Compatibility with MapReduce
- Improved cluster utilization
- Support for non-MapReduce workloads
- Agility

What YARN does:
YARN enhances the power of a Hadoop compute cluster in the following ways:

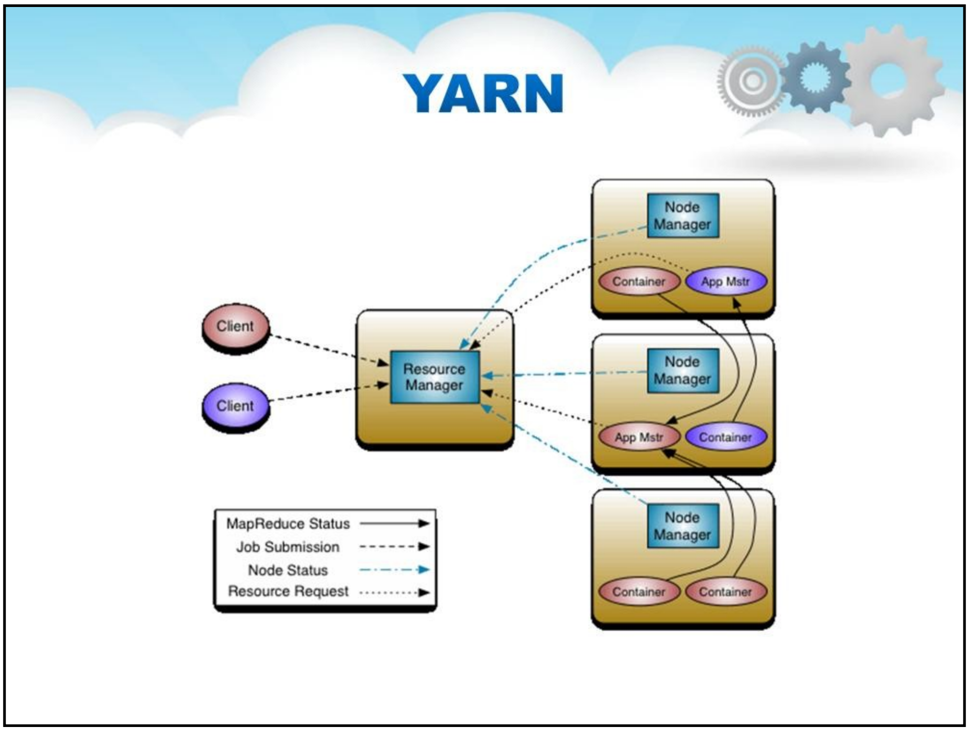
Scalability: The processing power in data centers continues to grow quickly. Because YARN ResourceManager focuses exclusively on scheduling, it can manage those larger clusters much more easily.

Compatibility with MapReduce: Existing MapReduce applications and users can run on top of YARN without disruption to their existing processes.

Improved cluster utilization: The ResourceManager is a pure scheduler that optimizes cluster utilization according to criteria such as capacity guarantees, fairness, and SLAs. Also, unlike before, there are no named map and reduce slots, which helps to better utilize cluster resources.

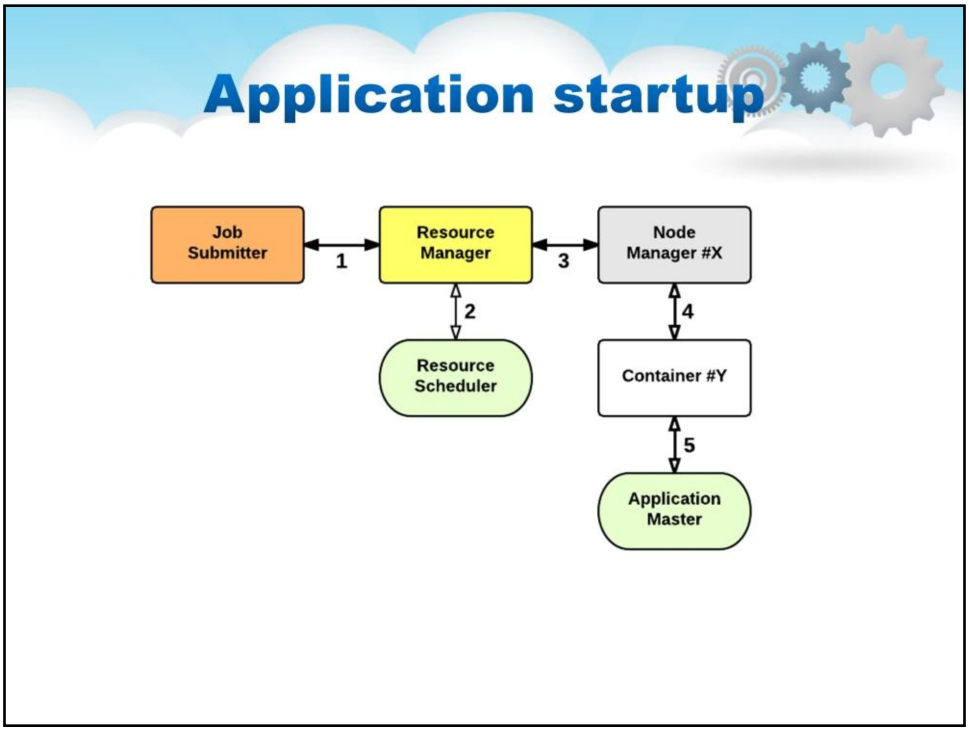
Support for **workloads other than MapReduce**: Additional programming models such as graph processing and iterative modeling are now possible for data processing.

Agility: With MapReduce becoming a user-land library, it can evolve independently of the underlying resource manager layer and in a much more agile manner.



YARN

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons.
The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs.
The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.
The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks. The ResourceManager has two main components: Scheduler and ApplicationsManager.
The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource *Container* which incorporates elements such as memory, cpu, disk, network etc.
The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the CapacityScheduler and the FairScheduler would be some examples of plug-ins.
The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.
MapReduce in hadoop-2.x maintains API compatibility with previous stable release (hadoop-1.x). This means that all MapReduce jobs should still run unchanged on top of YARN with just a recompile.



Application startup

In YARN, there are at least three actors:

- the **Job Submitter** (the client)
- the **Resource Manager** (the master)
- the **Node Manager** (the slave)

The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the **Application Master**

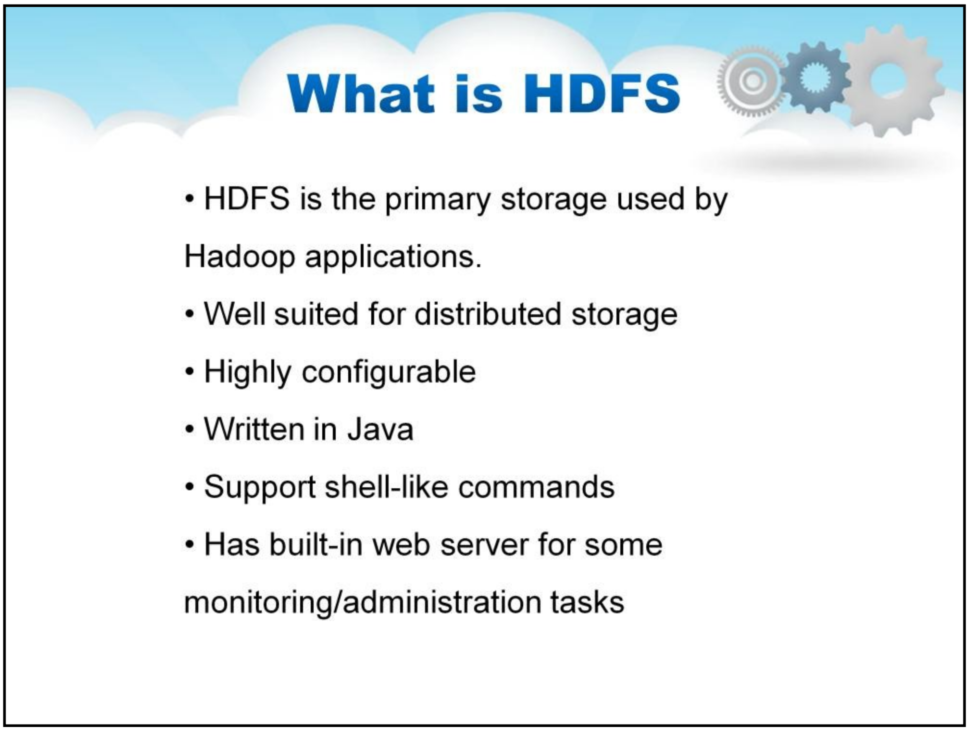
The Resource Manager is a single point of failure in YARN. Using Application Masters, YARN is spreading over the cluster the metadata related to running applications. This reduces the load of the Resource Manager and makes it fast recoverable.

HDFS

overview

HDFS

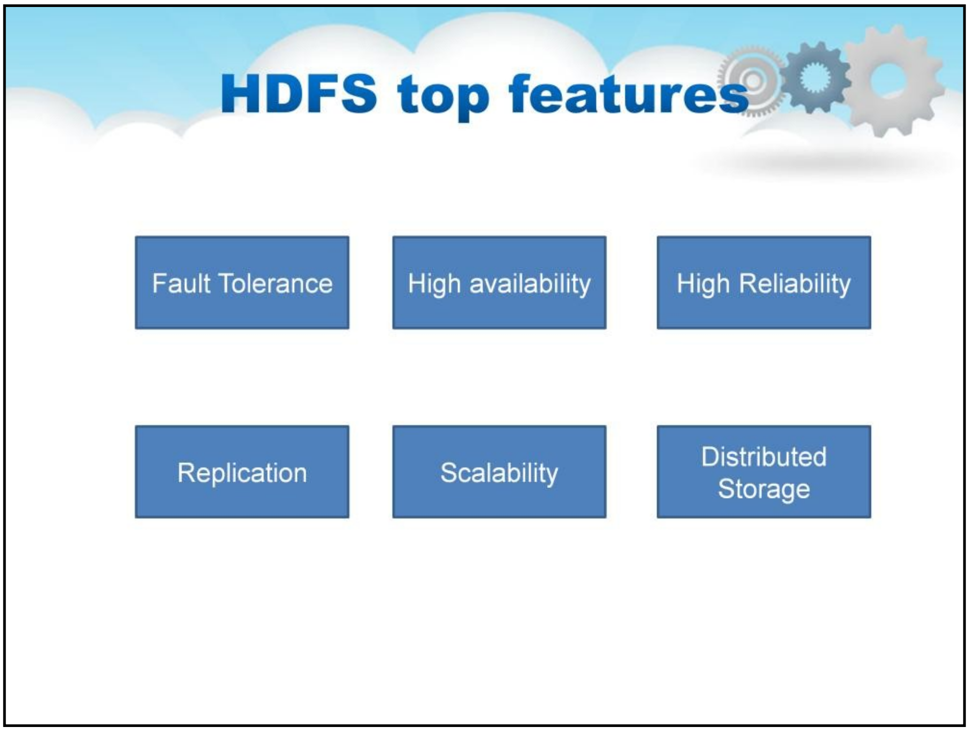
The HDFS is the storage system of the Hadoop framework.
It is a distributed file system that can conveniently run on commodity hardware for processing unstructured data.



What is HDFS

- HDFS is the primary storage used by Hadoop applications.
- Well suited for distributed storage
- Highly configurable
- Written in Java
- Support shell-like commands
- Has built-in web server for some monitoring/administration tasks

HDFS is the primary distributed storage used by Hadoop applications.
HDFS is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand.
HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
Hadoop is written in Java and is supported on all major platforms. Hadoop supports shell-like commands to interact with HDFS directly.
Have built in web servers that makes it easy to check current status of the cluster.



HDFS top features

Fault Tolerance

High availability

High Reliability

Replication

Scalability

Distributed Storage

Top HDFS features

1. Fault Tolerance

The fault tolerance in Hadoop HDFS is the working strength of a system in unfavorable conditions. It is highly fault-tolerant. Hadoop framework divides data into blocks. After that creates multiple copies of blocks on different machines in the cluster. So, when any machine in the cluster goes down, then a client can easily access their data from the other machine which contains the same copy of data blocks.

2. High Availability

Hadoop HDFS is a highly available file system. In HDFS, data gets replicated among the nodes in the Hadoop cluster by creating a replica of the blocks on the other slaves present in HDFS cluster. So, whenever a user wants to access this data, they can access their data from the slaves which contain its blocks. At the time of unfavorable situations like a failure of a node, a user can easily access their data from the other nodes. Because duplicate copies of blocks are present on the other nodes in the HDFS cluster.

3. High Reliability

HDFS provides reliable data storage. It can store data in the range of 100s of petabytes. HDFS stores data reliably on a cluster. It divides the data into blocks. Hadoop framework stores these blocks on nodes present in HDFS cluster. HDFS stores data reliably by creating a replica of each and every block present in the cluster. Hence provides fault tolerance facility. If the node in the cluster containing data goes down, then a user can easily access that data from the other nodes. HDFS by default creates 3 replicas of each block containing data present in the nodes. So, data is quickly available to the users. Hence user does not face the problem of data loss. Thus, HDFS is highly reliable.

4. Replication

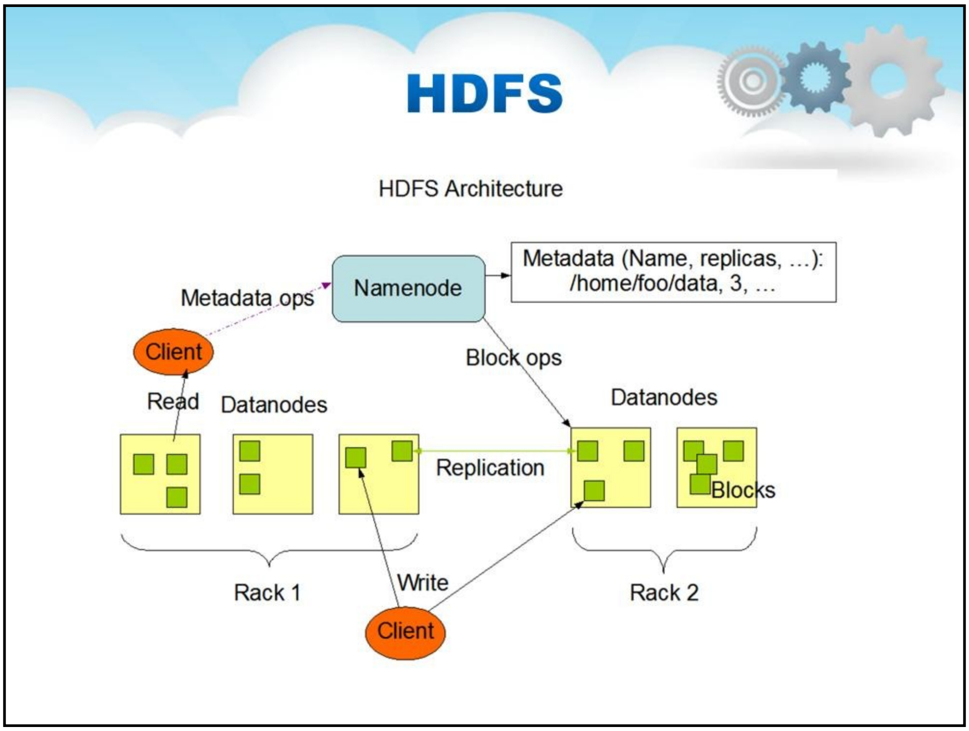
Data Replication is unique features of HDFS. Replication solves the problem of data loss in an unfavorable condition like hardware failure, crashing of nodes etc. HDFS maintain the process of replication at regular interval of time. HDFS also keeps creating replicas of user data on different machine present in the cluster. So, when any node goes down, the user can access the data from other machines. Thus, there is no possibility of losing of user data.

5. Scalability

Hadoop HDFS stores data on multiple nodes in the cluster. So, whenever requirements increase you can scale the cluster. Two scalability mechanisms are available in HDFS: Vertical and Horizontal Scalability.

6. Distributed Storage

All the features in HDFS are achieved via distributed storage and replication. HDFS store data in a distributed manner across the nodes. In Hadoop, data is divided into blocks and stored on the nodes present in the HDFS cluster. After that HDFS create the replica of each and every block and store on other nodes. When a single machine in the cluster gets crashed we can easily access our data from the other nodes which contain its replica.



HDFS

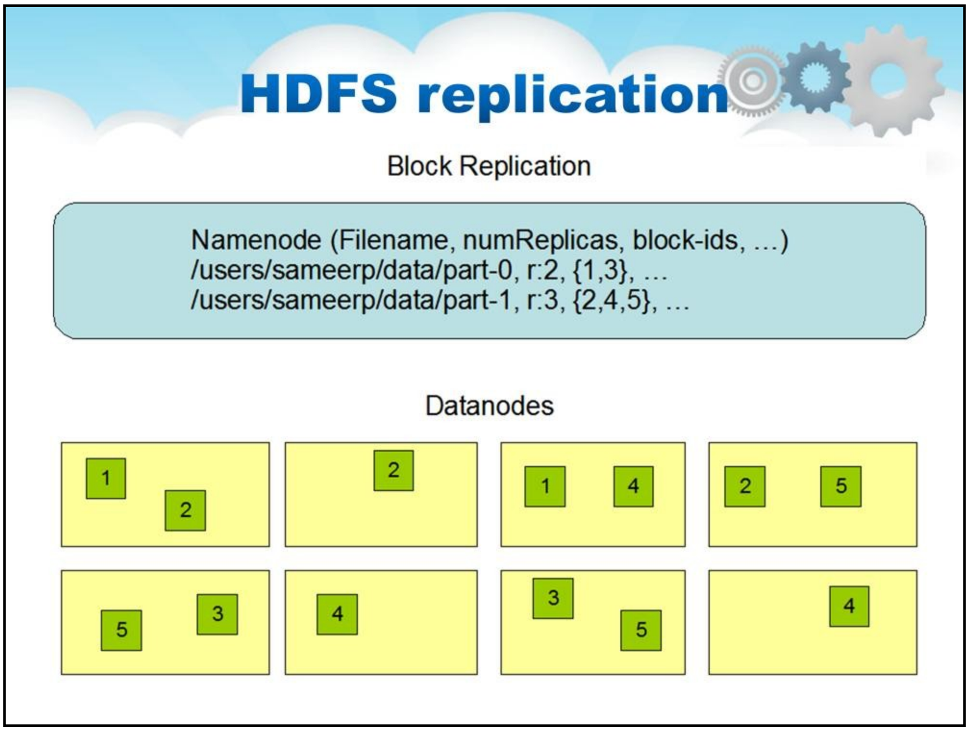
HDFS Architecture

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.



HDFS replication

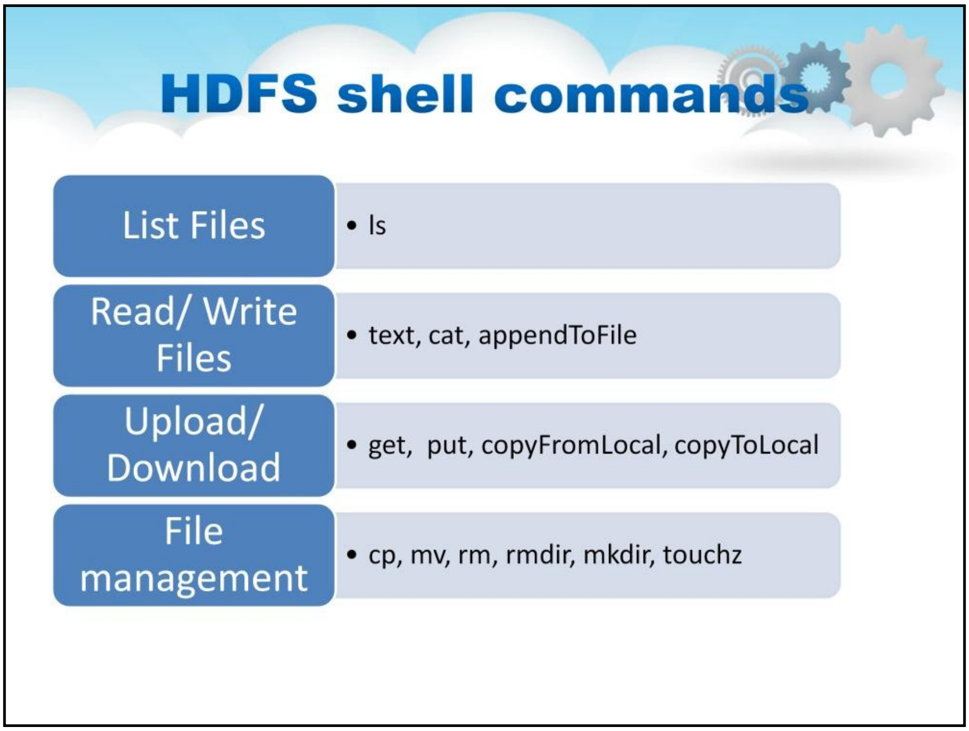
Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes

The File System Namespace
HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS supports user quotas and access permissions. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.
The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

Data Replication
HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
All blocks in a file except the last block are the same size, while users can start a new block without filling out the last block to the configured block size after the support for variable length block was added to append and hsync. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once (except for appends and truncates) and have strictly one writer at any time.
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.



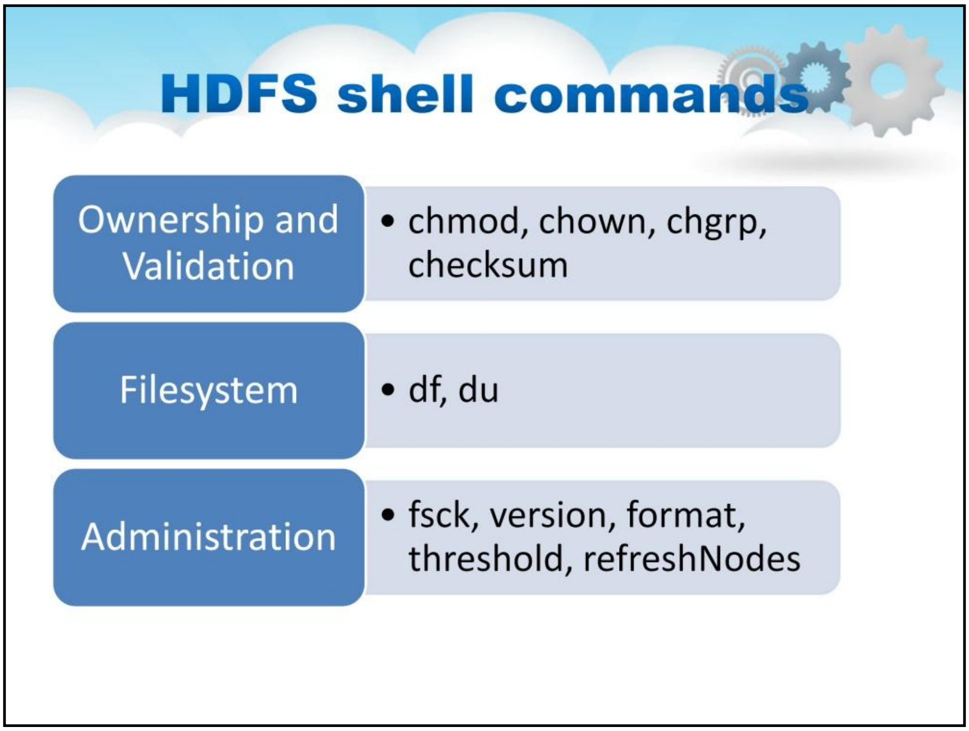
HDFS shell commands

List Files	• ls
Read/ Write Files	• text, cat, appendToFile
Upload/ Download	• get, put, copyFromLocal, copyToLocal
File management	• cp, mv, rm, rmdir, mkdir, touchz

The File System shell includes various shell-like commands that directly interact with the Hadoop Distributed File System (HDFS) as well as other file systems that Hadoop supports.

All commands fall into such categories:

- List Files (for example, **ls** command)
- Read/Write Files (**text**, **cat**, **appendToFile**)
- Upload/Download (**get**, **put**, **copyFromLocal**, **copyToLocal**)
- File management (**cp**, **mv**, etc)



HDFS shell commands

Ownership and Validation

- chmod, chown, chgrp, checksum

Filesystem

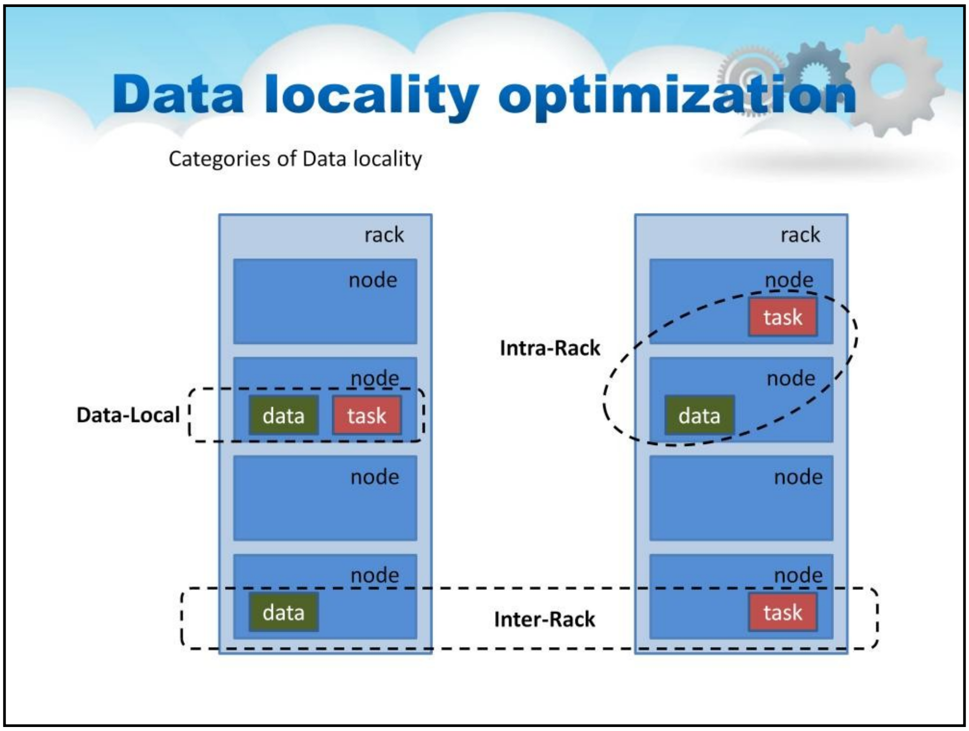
- df, du

Administration

- fsck, version, format, threshold, refreshNodes

- Ownership and Validation commands (such as **chmod**, **chown**, etc)
- Filesystem commands (**du** and **df**)
- Administration commands (**fsck**, **format**, etc)

Commands and how to use them will be discussed in details in the Lab



Data locality optimization

Categories of Data locality

Data Locality

In Hadoop it means moving computation close to data rather than moving data towards computation. Hadoop stores data in **HDFS**, which splits files into blocks and distribute among various data nodes. When a mapReduce job is submitted, it is divided into map jobs and reduce jobs. A Map job is assigned to a datanode according to the availability of the data, ie it assigns the task to a datanode which is closer to or stores the data on its local disk. Data locality refers the process of placing computation near to data , which helps in high throughput and faster execution of data.

Categories of Data locality**:**

1. Data Local

If a map task is executing on a node which has the input block to be processed, its called data local.

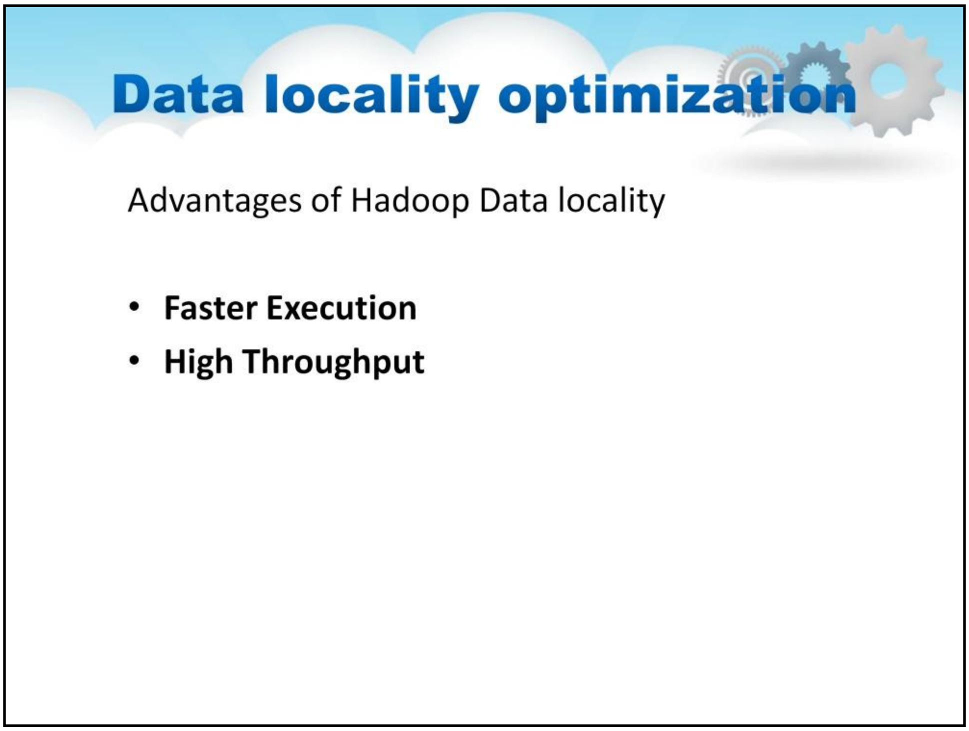
2. Intra- Rack

Its always not possible to run map task on the same node where data is located due to network constraints. In that case, mapper runs on another machine, but on the same rack. So the data need to be moved between the nodes for execution.

3. Inter-Rack

In certain cases Intra- Rack local is also not possible. In such cases, the mapper will execute from a different rack.In order to execute the mapper, the data need to be copied from the node which stores the data to the node which is executing the mapper between the racks.

Map jobs read data from the input blocks and generate intermediate results. Since map jobs work on blocks from HDFS and are data-parallel, data locality is important for better performance and faster execution of the map jobs.



Data locality optimization

Advantages of Hadoop Data locality

- **Faster Execution**
- **High Throughput**

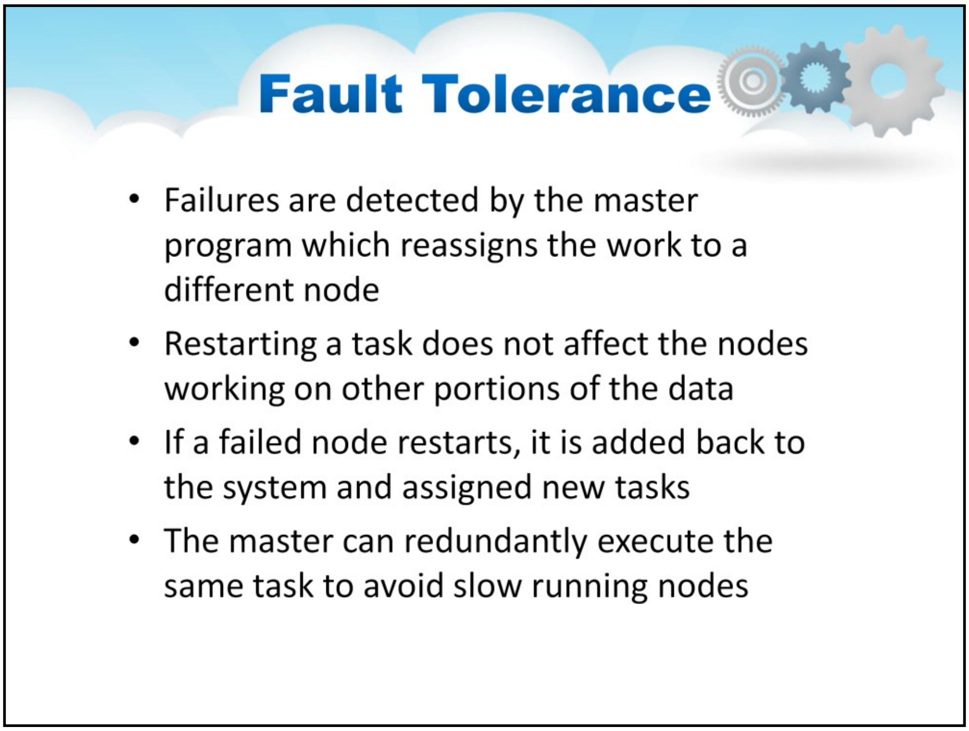
There are two benefits of data Locality in MapReduce. Let’s discuss them one by one-

Faster Execution

In data locality, the program is moved to the node where data resides instead of moving large data to the node, this makes Hadoop faster. Because the size of the program is always lesser than the size of data, so moving data is a bottleneck of network transfer.

High Throughput

Data locality increases the overall throughput of the system.

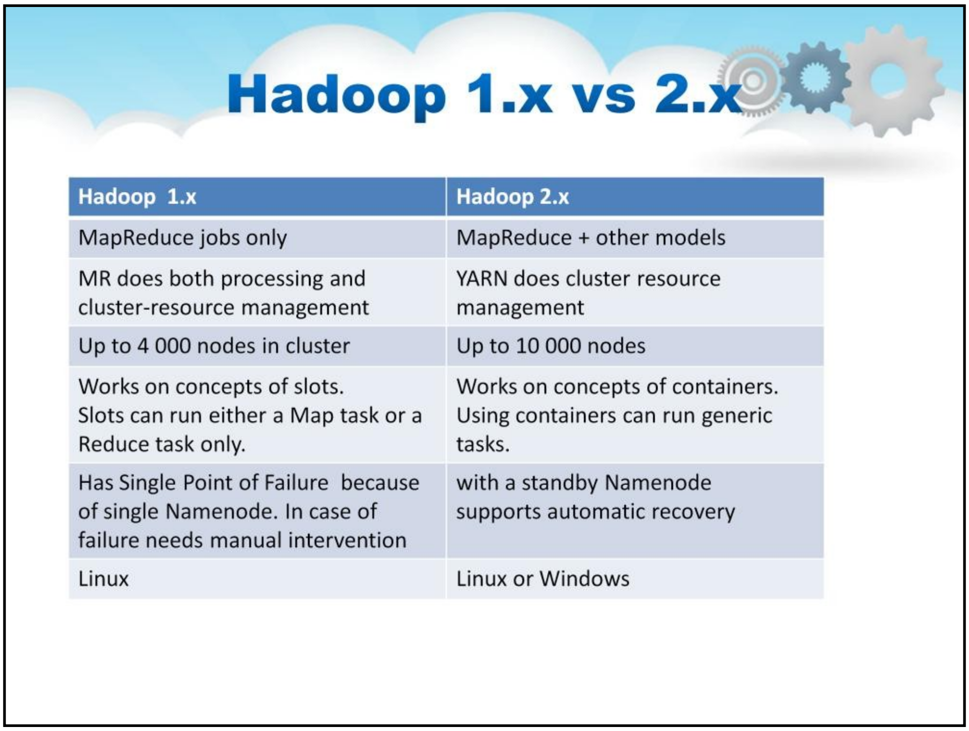


Fault Tolerance

- Failures are detected by the master program which reassigns the work to a different node
- Restarting a task does not affect the nodes working on other portions of the data
- If a failed node restarts, it is added back to the system and assigned new tasks
- The master can redundantly execute the same task to avoid slow running nodes

Fault Tolerance

Failures are detected by the master program which reassigns the work to a different node. Restarting a task does not affect the nodes working on other portions of the data.
If a failed node restarts, it is added back to the
system and assigned new tasks. The master can
redundantly execute the same task to avoid slow
running nodes.

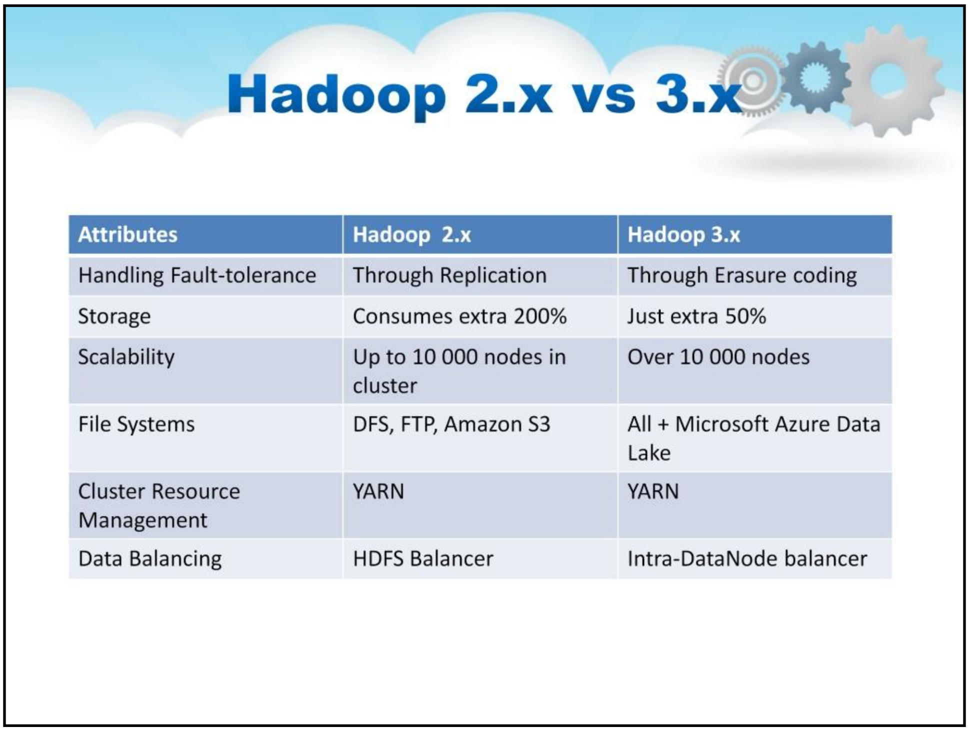


Hadoop 1.x vs 2.x

Hadoop 1.x	Hadoop 2.x
MapReduce jobs only	MapReduce + other models
MR does both processing and cluster-resource management	YARN does cluster resource management
Up to 4 000 nodes in cluster	Up to 10 000 nodes
Works on concepts of slots. Slots can run either a Map task or a Reduce task only.	Works on concepts of containers. Using containers can run generic tasks.
Has Single Point of Failure because of single Namenode. In case of failure needs manual intervention	with a standby Namenode supports automatic recovery
Linux	Linux or Windows

Version Differences

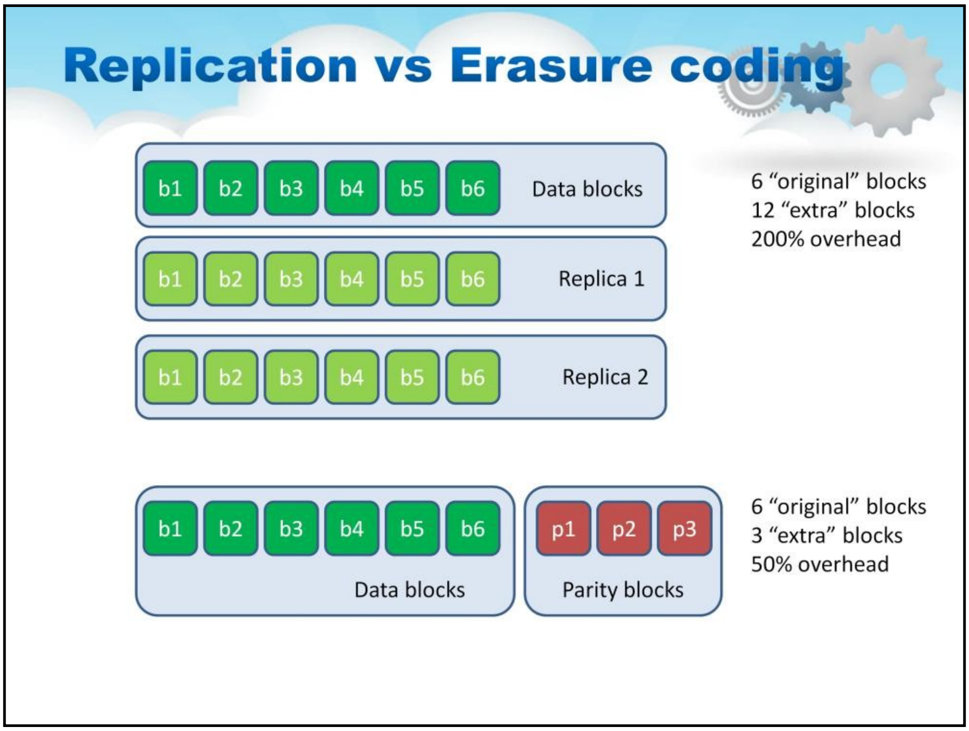
Here is Hadoop 1.x comparison with Hadoop 2.x table.
The main differences are that Hadoop 2 uses YARN for resource management, support both MapReduce and non- MapReduce jobs, more scalable (up to 10000 nodes), and more fault tolerant. It also can run on Windows.



Hadoop 2.x vs 3.x

Attributes	Hadoop 2.x	Hadoop 3.x
Handling Fault-tolerance	Through Replication	Through Erasure coding
Storage	Consumes extra 200%	Just extra 50%
Scalability	Up to 10 000 nodes in cluster	Over 10 000 nodes
File Systems	DFS, FTP, Amazon S3	All + Microsoft Azure Data Lake
Cluster Resource Management	YARN	YARN
Data Balancing	HDFS Balancer	Intra-DataNode balancer

Comparing to 2.x, Hadoop 3.x requires less storage space for the same data (through erasure coding and not duplication), more scalable, and has intra-DataNode balancer.



Replication vs Erasure coding

b1 b2 b3 b4 b5 b6 Data blocks

b1 b2 b3 b4 b5 b6 Replica 1

b1 b2 b3 b4 b5 b6 Replica 2

6 “original” blocks
12 “extra” blocks
200% overhead

b1 b2 b3 b4 b5 b6 Data blocks

p1 p2 p3 Parity blocks

6 “original” blocks
3 “extra” blocks
50% overhead

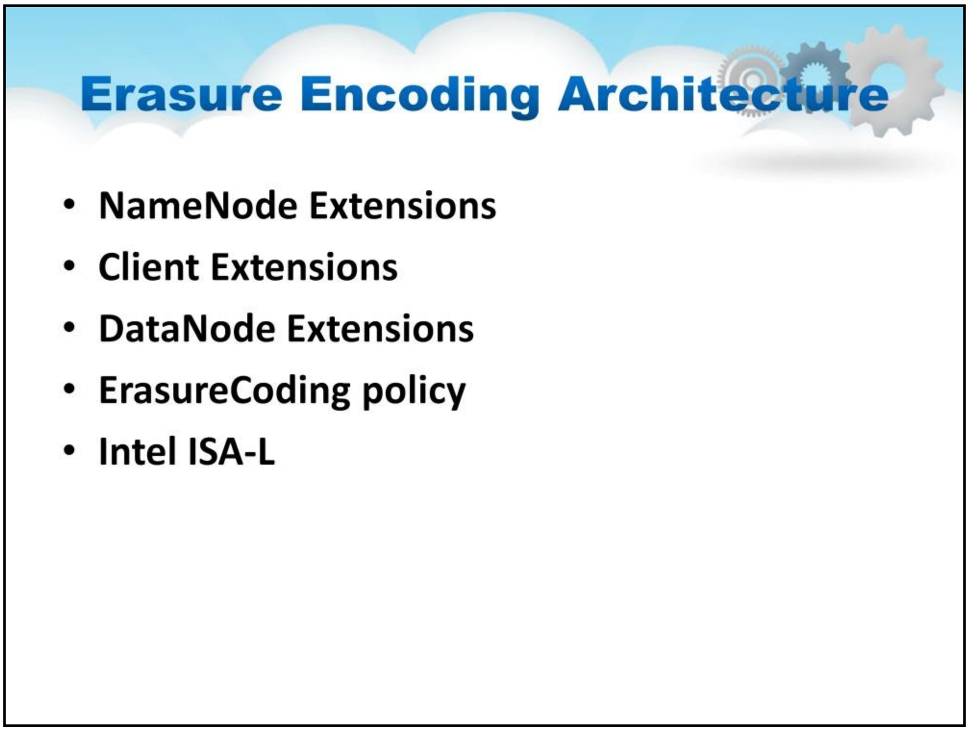
Replication

Replication is expensive – the default 3x replication scheme in HDFS has 200% overhead in storage space and other resources (e.g., network bandwidth). However, for warm and cold datasets with relatively low I/O activities, additional block replicas are rarely accessed during normal operations, but still consume the same amount of resources as the first replica.

Therefore, a natural improvement is to use Erasure Coding (EC) in place of replication, which provides the same level of fault-tolerance with much less storage space. In typical Erasure Coding (EC) setups, the storage overhead is no more than 50%. Replication factor of an EC file is meaningless. It is always 1 and cannot be changed via -setrep command.

In storage systems, the most notable usage of EC is Redundant Array of Inexpensive Disks (RAID). RAID implements EC through striping, which divides logically sequential data (such as a file) into smaller units (such as bit, byte, or block) and stores consecutive units on different disks. In the rest of this guide this unit of striping distribution is termed a striping cell (or cell). For each stripe of original data cells, a certain number of parity cells are calculated and stored – the process of which is called encoding. The error on any striping cell can be recovered through decoding calculation based on surviving data and parity cells.

Integrating EC with HDFS can improve storage efficiency while still providing similar data durability as traditional replication-based HDFS deployments. As an example, a 3x replicated file with 6 blocks will consume 6*3 = 18 blocks of disk space. But with EC (6 data, 3 parity) deployment, it will only consume 9 blocks of disk space.



Erasure Encoding Architecture

- NameNode Extensions
- Client Extensions
- DataNode Extensions
- ErasureCoding policy
- Intel ISA-L

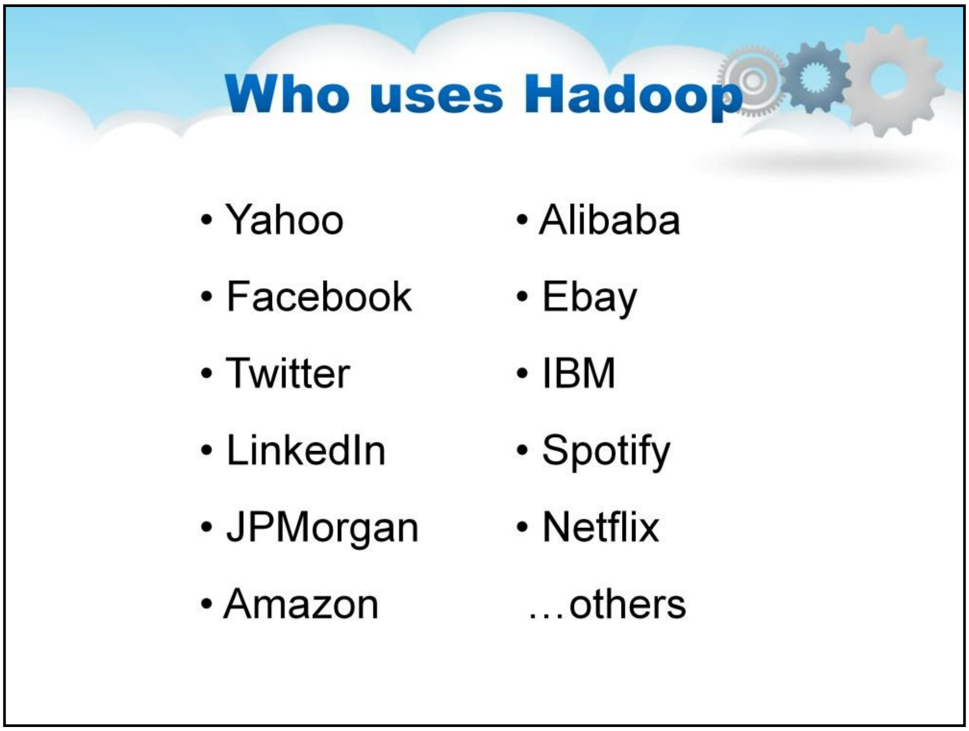
HDFS Erasure Encoding Architecture

NameNode Extensions – The HDFS files are striped into block groups, which have a certain number of internal blocks. Now to reduce NameNode memory consumption from these additional blocks, a new hierarchical ***block naming protocol*** was introduced. The ID of a block group can be deduced from the ID of any of its internal blocks. This allows management at the level of the block group rather than the block.

Client Extensions – After implementing Erasure Encoding in HDFS, NameNode works on block group level & the client read and write paths were enhanced to work on multiple internal blocks in a block group in *parallel*. On the output/write path, *DFSStripedOutputStream* manages a set of data streamers, one for each DataNode storing an internal block in the current block group. A coordinator takes charge of operations on the entire block group, including ending the current block group, allocating a new block group, etc.On the input/read path, *DFSStripedInputStream* translates a requested logical byte range of data as ranges into internal blocks stored on DataNodes. It then issues read requests in parallel. Upon failures, it issues additional read requests for decoding.

DataNode Extensions – The DataNode runs an additional ErasureCodingWorker (ECWorker) task for background recovery of failed erasure coded blocks. Failed EC blocks are detected by the NameNode, which then chooses a DataNode to do the recovery work. Reconstruction performs three key tasks:Read the data from source nodes and reads only the minimum number of input blocks & parity blocks for reconstruction.New data and parity blocks are decoded from the input data. All missing data and parity blocks are decoded together.Once decoding is finished, the recovered blocks are transferred to target DataNodes.

ErasureCoding policy – To accommodate heterogeneous workloads, we allow files and directories in an HDFS cluster to have different replication and EC policies. Information about encoding & decoding files is encapsulated in an ErasureCodingPolicy class. It contains 2 pieces of information, i.e. the *ECSchema & the size of a stripping cell.* **Intel ISA-L** Intel ISA-L stands for Intel Intelligent Storage Acceleration Library. ISA-L is an open-source collection of optimized low-level functions designed for storage applications. It includes fast block Reed-Solomon type erasure codes optimized for Intel AVX and AVX2 instruction sets. HDFS erasure coding can leverage ISA-L to accelerate encoding and decoding calculation. ISA-L supports most major operating systems, including Linux and Windows. ISA-L is not enabled by default. See the instructions below for how to enable ISA-L



Who uses Hadoop

- Yahoo
- Facebook
- Twitter
- LinkedIn
- JPMorgan
- Amazon
- Alibaba
- Ebay
- IBM
- Spotify
- Netflix

…others

Usage

Yahoo: Used for scaling tests.

Facebook: Used as a source for reporting and machine learning.

Twitter: To store and process tweets, log files using LZO which is a portable lossless data compression library written in ANSI C. It is fast and also helps release CPU for other tasks.

LinkedIn: LinkedIn's data flows through Hadoop clusters. User activity, server metrics, images, transaction logs stored in HDFS are used by data analysts for business analytics like discovering people whom you may know.

JPMorgan: Analytics on the transactions of the customers.

Amazon: Data processing by analyzing the customer reviews and requirements.

Adobe: Social services to structured data storage.

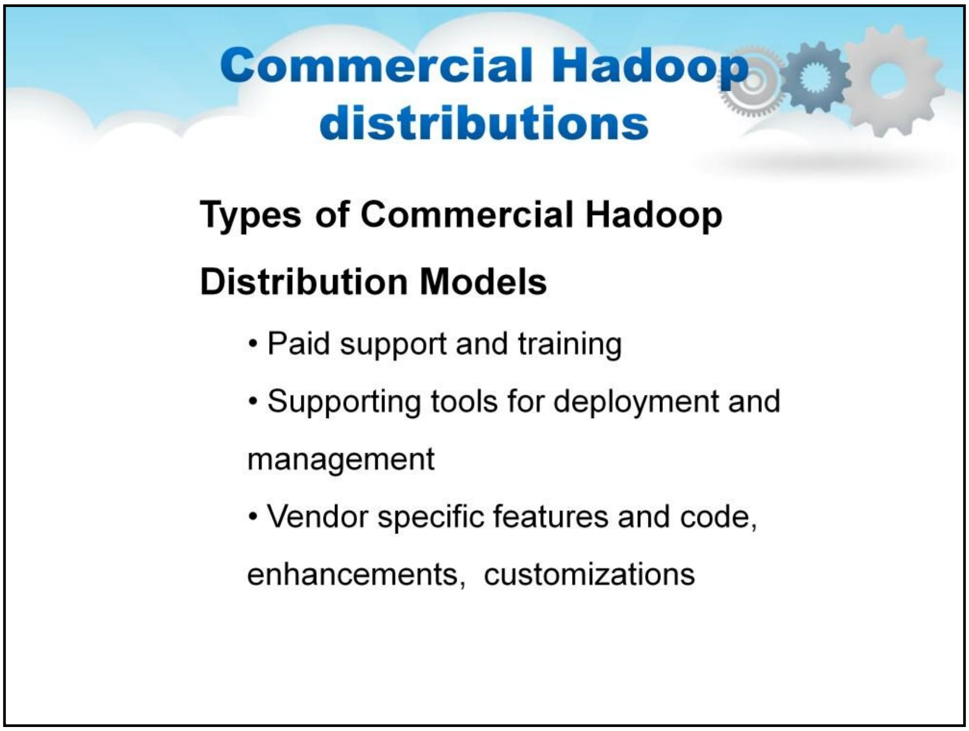
Ebay: With 300+ million users browsing more than 350 million products listed on their website, eBay has one of the largest Hadoop clusters in the industry that is run prominently on MapReduce Jobs. Hadoop is used by eBay for Search Optimization and Research.

Netflix: For decision making.

Aol: Targets machines and dual processors.

Alibaba: Analyzes vertical search engine.

IBM: Client projects in finance, telecom and retail,
Machine learning with Watson Analytics.
Infosys: Client projects in finance, telecom and retail.
TCS: Client projects in finance, telecom and retail.
Spotify: Used for content generation and data aggregation.



Commercial Hadoop distributions

Types of Commercial Hadoop Distribution Models

- Paid support and training
- Supporting tools for deployment and management
- Vendor specific features and code, enhancements, customizations

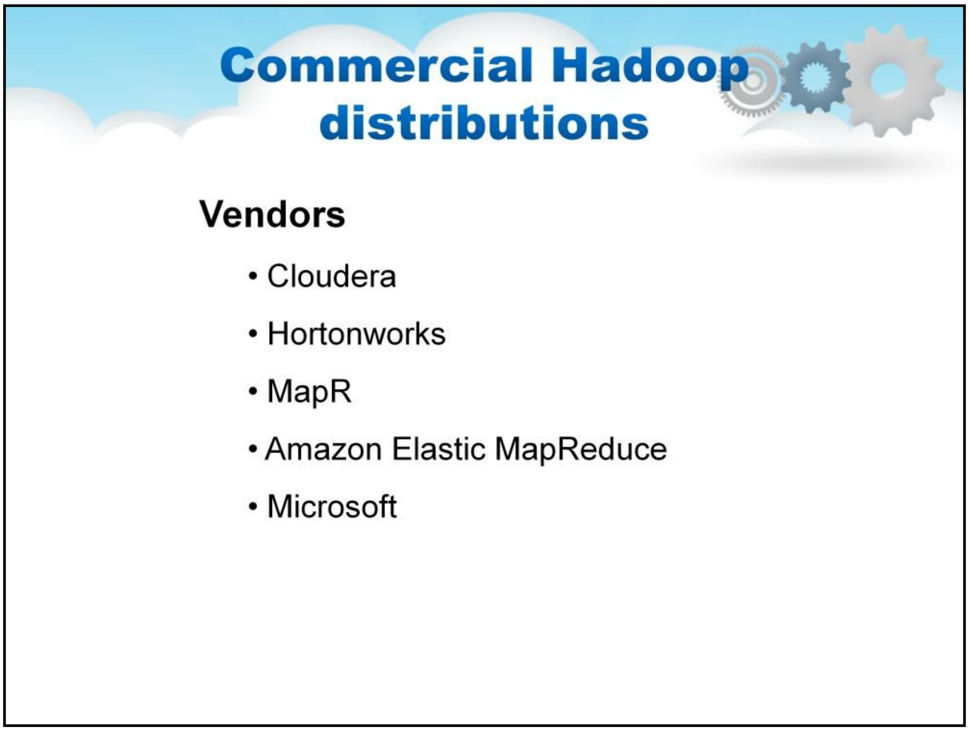
Types of Commercial Hadoop Distribution Models

To fulfill the need of enterprises to deploy Hadoop for taming Big Data, several companies came up with commercial distribution models for Hadoop. Commercial Hadoop distributions are majorly categorized in three primary kinds. They are as follows:

Distributions that provide paid support and training for the Apache Hadoop (e.g. Cloudera, HortonWorks, MapR, IBM, etc.).

Distributions that offer a set of supporting tools for deployment and management of Apache Hadoop as an alternative (e.g. Cloudera, HortonWorks, MapR).

Distributions that enable adding vendor specific features and code, paid enhancements, to enhance or customize the Apache Hadoop deployment and align it to the business needs (e.g. Cloudera, HortonWorks, MapR, IBM, etc.) Now the big question is, how do you choose a Hadoop distribution from the numerous options that are available in the market? Let’s take a look at some of the criteria that may guide you to choose the suitable Hadoop distribution for you.



Commercial Hadoop distributions

Vendors

- Cloudera
- Hortonworks
- MapR
- Amazon Elastic MapReduce
- Microsoft

Cloudera

The US-based software and solutions provider for Apache Hadoop technology and the first vendor to offer Hadoop as a package, Cloudera, still continues to be a market leader in Hadoop distributions. Cloudera's CDH that comprise all the open source components targets enterprise-class deployments and is one of the most popular commercial Hadoop distributions.

Known for its innovations, Cloudera was the first to offer SQL-for-Hadoop with its Impala query engine. Other additions of Cloudera include security, user interface, and interfaces for integration with third party applications. Cloudera supports its distribution through the Cloudera Enterprise subscription service.

Hortonworks

Hortonworks develops and supports Apache Hadoop for the distributed processing of large data sets across computer clusters. The Hortonworks Data Platform (HDP) is an entirely open source platform designed to maneuver data from many sources and formats. The platform includes various Hadoop technology such as the Hadoop Distributed File System, MapReduce, Zookeeper, HBase, Pig, and Hive, and additional components.

Hortonworks is known for making acquisitions of other companies with useful code and releasing the code into the open source community. A new trend towards consolidation in the market has resulted in the growing popularity of Hortonworks' product. Recently, both Amazon and IBM has started offering Hortonworks as options on their own platforms alongside their own Hadoop distributions. HDP also serves as the core of the Open Data Platform Initiative, a group aiming to simplify and standardize specifications in the Big Data ecosphere.

MapR

Rather than a managed service provider like Amazon and Microsoft, MapR is a platform-focused Hadoop solutions provider, just like Hortonworks and Cloudera. MapR integrates its own database system, known as MapR-DB, while offering Hadoop distribution services. MapR-DB is claimed to be four to seven times faster than the stock Hadoop database, HBase, running on other distributions. Thanks to its speed, MapR is often seen as a preferred choice for large Big Data projects.

Amazon Elastic MapReduce

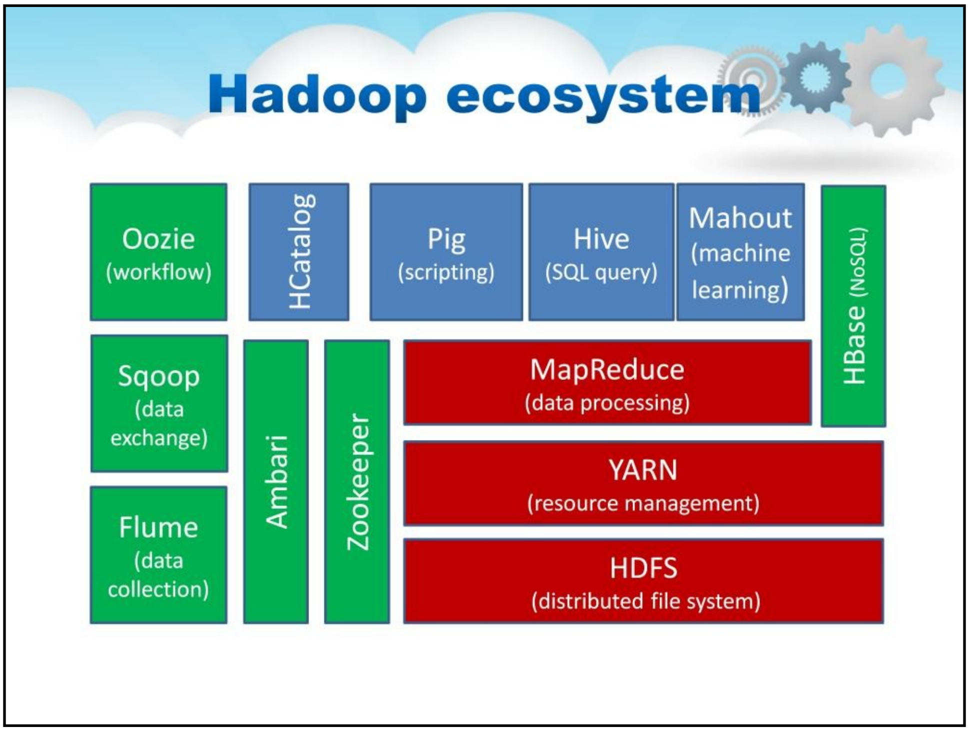
Amazon offers a pay-as-you-go model in a cloud-only platform. It provides Hadoop-as-a-Service platform through its Amazon Web Services arm. The key advantage of the pay-as-you-go model is the scalability. This model allows you to scale up or down as demands change. Amazon Elastic MapReduce also seamlessly connects with Amazon's other cloud services infrastructure such as Amazon S3 and DynamoDB for storage, EC2 for cloud processing, and AWS IoT for collecting data from Internet of Things-enabled devices.

Microsoft

Microsoft also offers a cloud-only service in the form of Azure HDInsight platform that offers managed installations of several open source Hadoop distributions, including Cloudera, Hortonworks, and MapR. HDInsight integrates different Hadoop distributions with its own Azure Data Lake platform to provide a complete solution for cloud-based storage and analytics. Additionally, HDInsights provides Hive, Spark, Kafka, and Storm cloud services along with its own cloud security framework.

Opting for the right Hadoop Distribution entirely depends on the obstacles and problems an organization is facing in implementing Hadoop in the enterprise. Each commercial Hadoop distribution has its own pros and cons.

Therefore, it is imperative to consider the risk and cost along with the additional value offered by each Hadoop distribution, for the distribution to prove beneficial for your business needs.



Hadoop ecosystem

Oozie (workflow)

HCatalog

Pig (scripting)

Hive (SQL query)

Mahout (machine learning)

HBase (NoSQL)

Sqoop (data exchange)

Ambari

Zookeeper

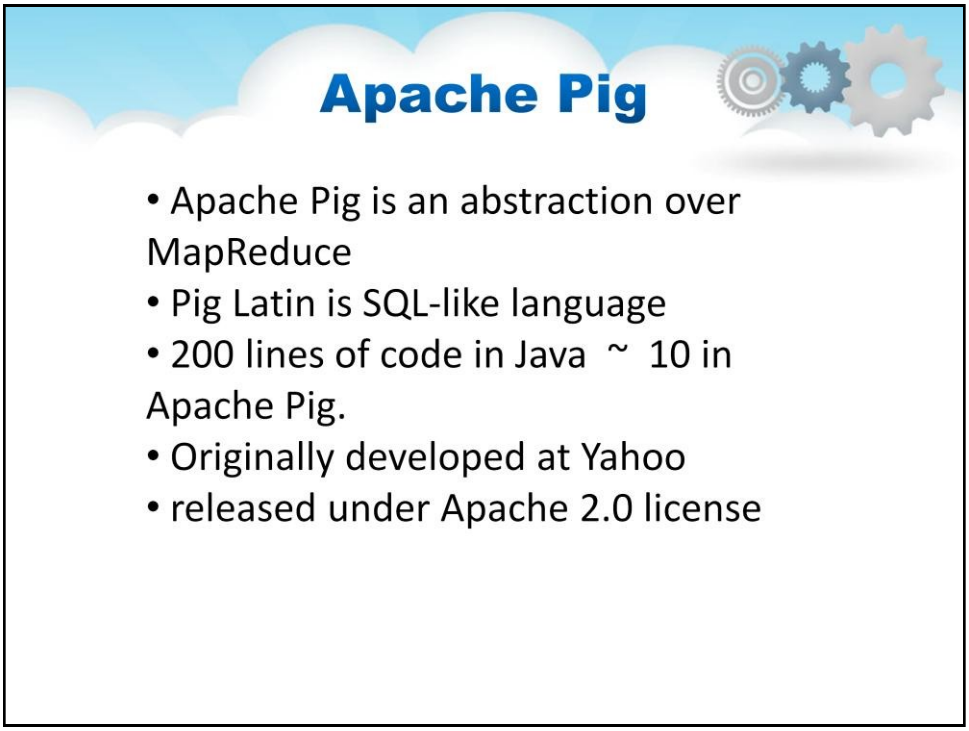
MapReduce (data processing)

YARN (resource management)

Flume (data collection)

HDFS (distributed file system)

Hadoop HDFS – Distributed storage layer for Hadoop.
Yarn Hadoop – Resource management layer
introduced in Hadoop 2.x. Hadoop Map-Reduce –
Parallel processing layer for Hadoop.
HBase – It is a column-oriented database that runs on top of HDFS. It is a NoSQL database which does not understand the structured query. For sparse data set, it suits well.
Hive – Apache Hive is a data warehousing infrastructure based on Hadoop
and it enables easy data summarization, using SQL queries.
Pig – It is a top-level scripting language. As we use it with Hadoop. Pig enables writing complex data processing without Java programming.
Flume – It is a reliable system for efficiently collecting large amounts of log data
from many different sources in real-time.
Sqoop – It is a tool design to transport huge volumes of data between Hadoop and RDBMS.
Oozie – It is a Java Web application uses to schedule Apache Hadoop jobs. It combines multiple jobs sequentially into one logical unit of work.
Zookeeper – A centralized service for maintaining configuration information,
naming, providing distributed synchronization, and providing group services.
Mahout – A library of scalable machine-learning algorithms, implemented on top of Apache Hadoop and using the MapReduce paradigm



Apache Pig

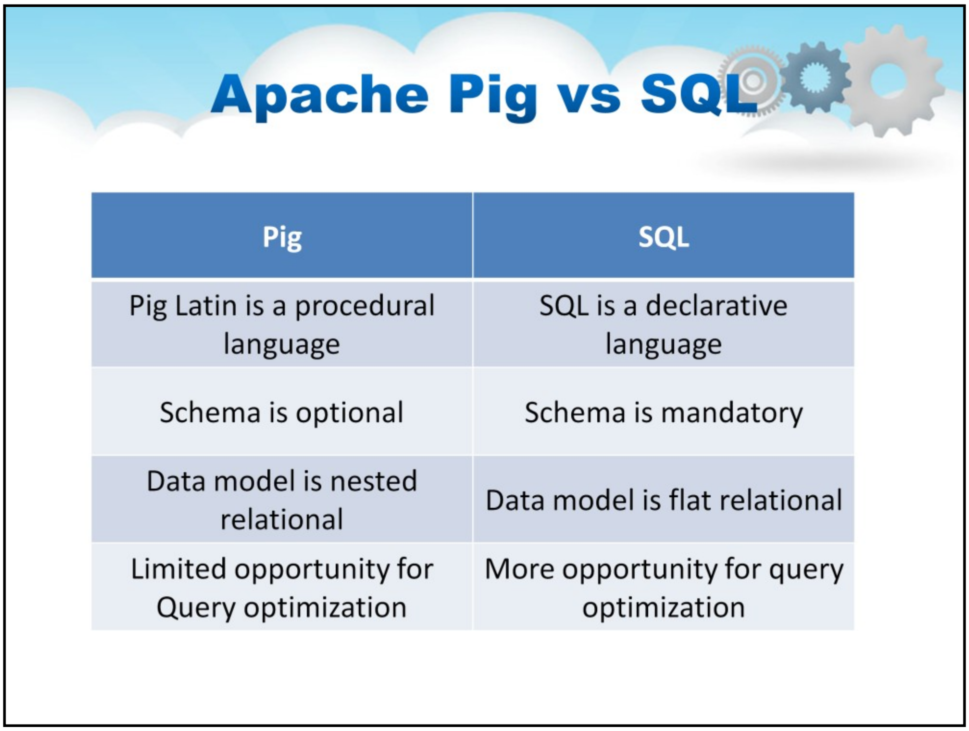
- Apache Pig is an abstraction over MapReduce
- Pig Latin is SQL-like language
- 200 lines of code in Java ~ 10 in Apache Pig.
- Originally developed at Yahoo
- released under Apache 2.0 license

Apache Pig

Apache Pig is a platform for analyzing large data sets, is an abstraction over MapReduce. To write data analysis programs, Pig provides a high-level language known as **Pig Latin**.

Pig Latin lets you specify a sequence of data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. Pig comes with many built-in functions but you can also create your own user-defined functions to do special-purpose processing.

Pig Latin programs run in a distributed fashion on a cluster (programs are complied into Map/Reduce jobs and executed using Hadoop).



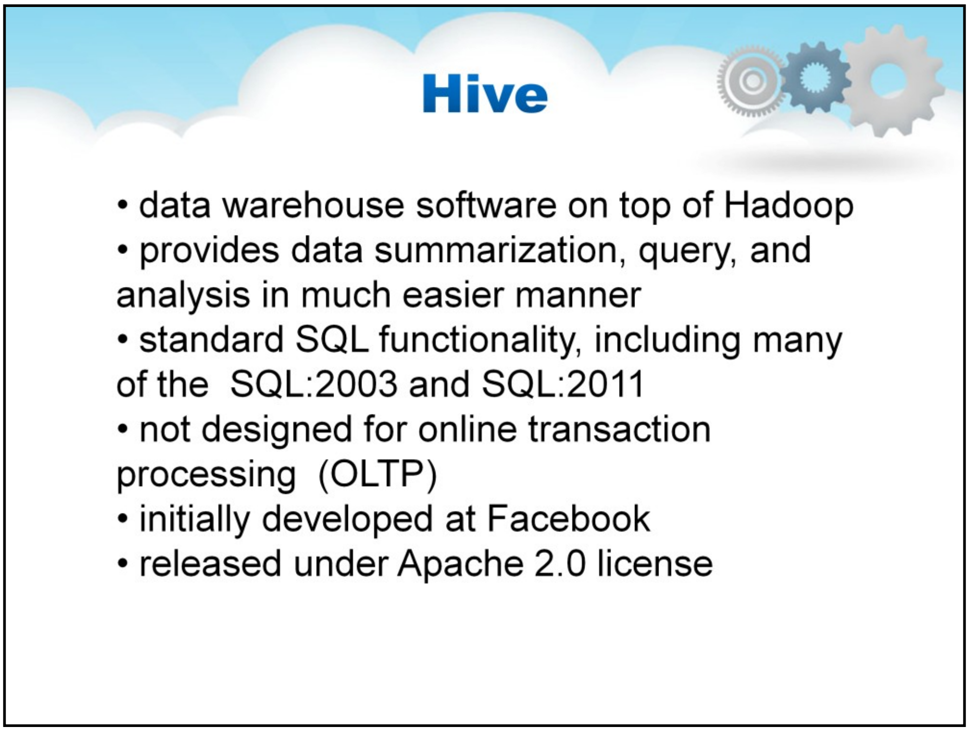
Apache Pig vs SQL

Pig	SQL
Pig Latin is a procedural language	SQL is a declarative language
Schema is optional	Schema is mandatory
Data model is nested relational	Data model is flat relational
Limited opportunity for Query optimization	More opportunity for query optimization

Apache Pig is a platform for analyzing large data sets, is an abstraction over MapReduce. To write data analysis programs, Pig provides a high-level language known as **Pig Latin**.

Pig Latin lets you specify a sequence of data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. Pig comes with many built-in functions but you can also create your own user-defined functions to do special-purpose processing.

Pig Latin programs run in a distributed fashion on a cluster (programs are complied into Map/Reduce jobs and executed using Hadoop).



Hive

- data warehouse software on top of Hadoop
- provides data summarization, query, and analysis in much easier manner
- standard SQL functionality, including many of the SQL:2003 and SQL:2011
- not designed for online transaction processing (OLTP)
- initially developed at Facebook
- released under Apache 2.0 license

Apache Hive

The Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax.

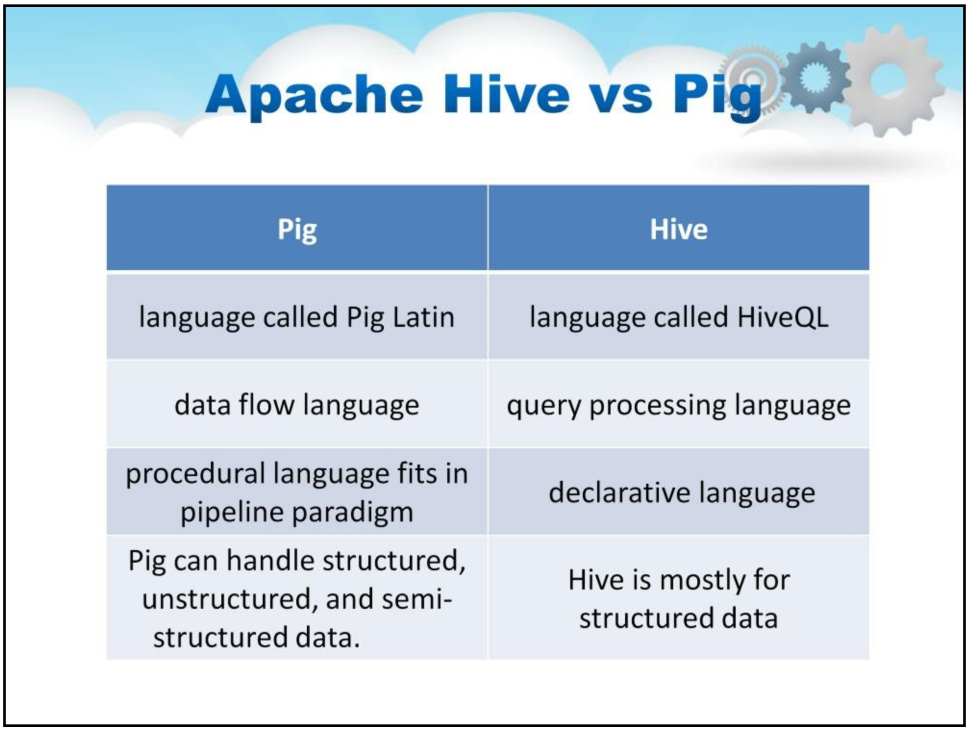
provides data summarization, query, and analysis in much easier manner supports external tables which make it possible to process data without actually storing in HDFS.

Using HiveQL doesn't require any knowledge of programming language, Knowledge of basic SQL query is enough. Hive provides standard SQL functionality, including many of the later SQL:2003 and SQL:2011 features for analytics.

Hive's SQL can also be extended with user code via user defined functions

Hive is not designed for online transaction processing (OLTP) workloads. It is best used for traditional data warehousing tasks.

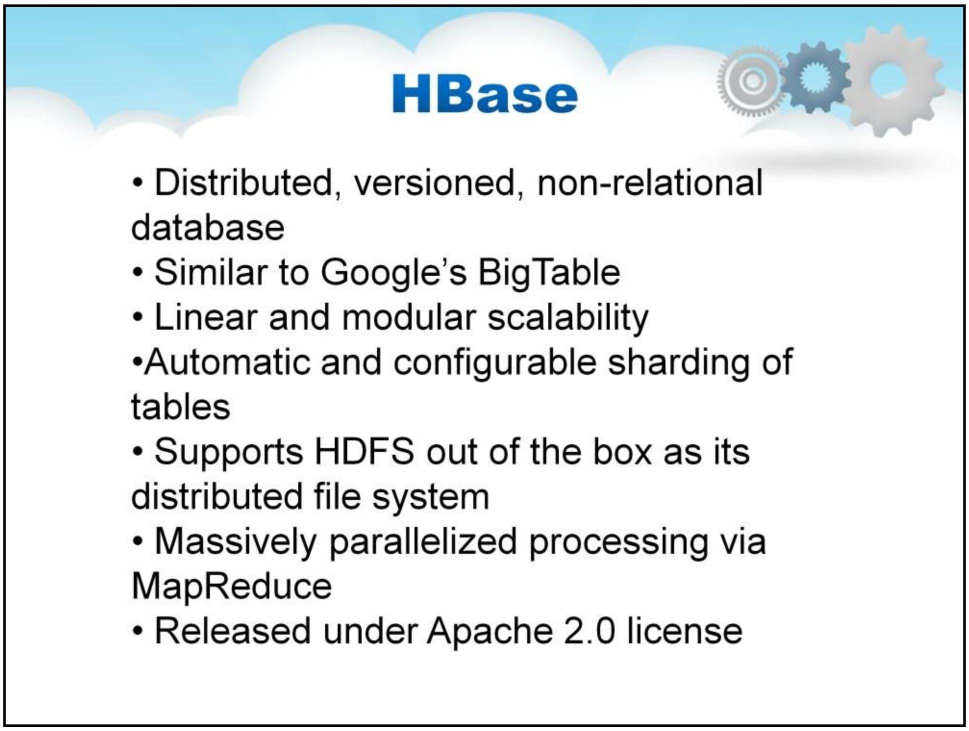
Hive is designed to maximize scalability (scale out with more machines added dynamically to the Hadoop cluster), performance, extensibility, fault-tolerance, and loose-coupling with its input formats.



Apache Hive vs Pig

Pig	Hive
language called Pig Latin	language called HiveQL
data flow language	query processing language
procedural language fits in pipeline paradigm	declarative language
Pig can handle structured, unstructured, and semi-structured data.	Hive is mostly for structured data

Here is a Apache Pig vs Hive comparison table.
Apache Pig uses a language called Pig Latin. It was originally created at Yahoo. Hive uses a language called HiveQL. It was originally created at Facebook.
Pig Latin is a data flow language. HiveQL is a query processing language.
Pig Latin is a procedural language and it fits in pipeline paradigm. HiveQL is a declarative language.
Apache Pig can handle structured, unstructured, and semi-structured data. Hive is mostly for structured data.

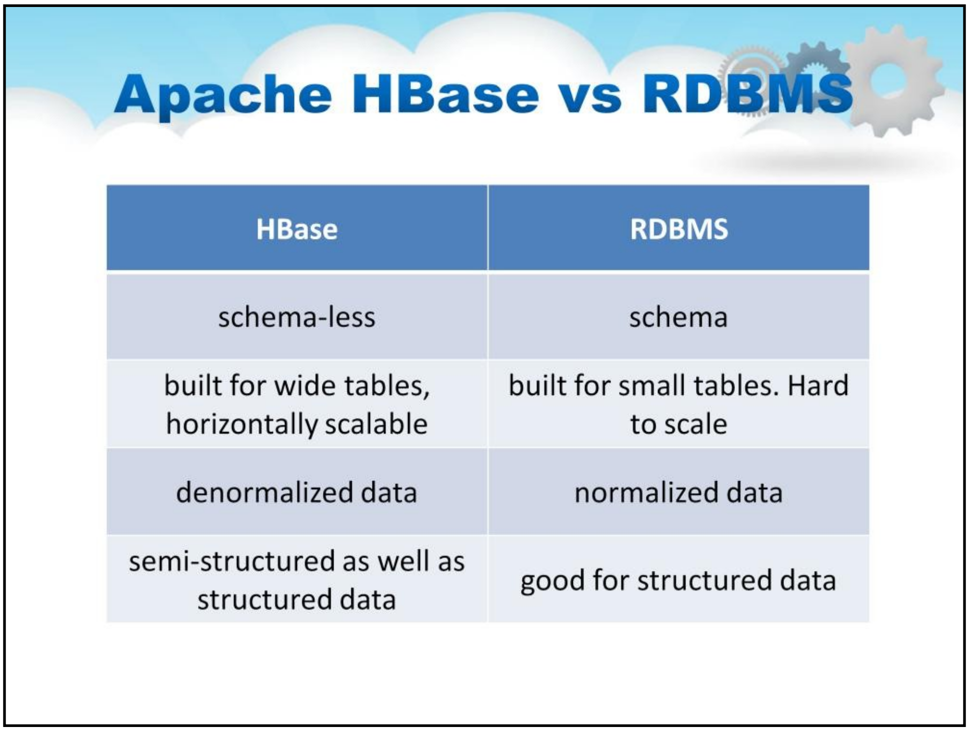


HBase

- Distributed, versioned, non-relational database
- Similar to Google’s BigTable
- Linear and modular scalability
- Automatic and configurable sharding of tables
- Supports HDFS out of the box as its distributed file system
- Massively parallelized processing via MapReduce
- Released under Apache 2.0 license

HBase

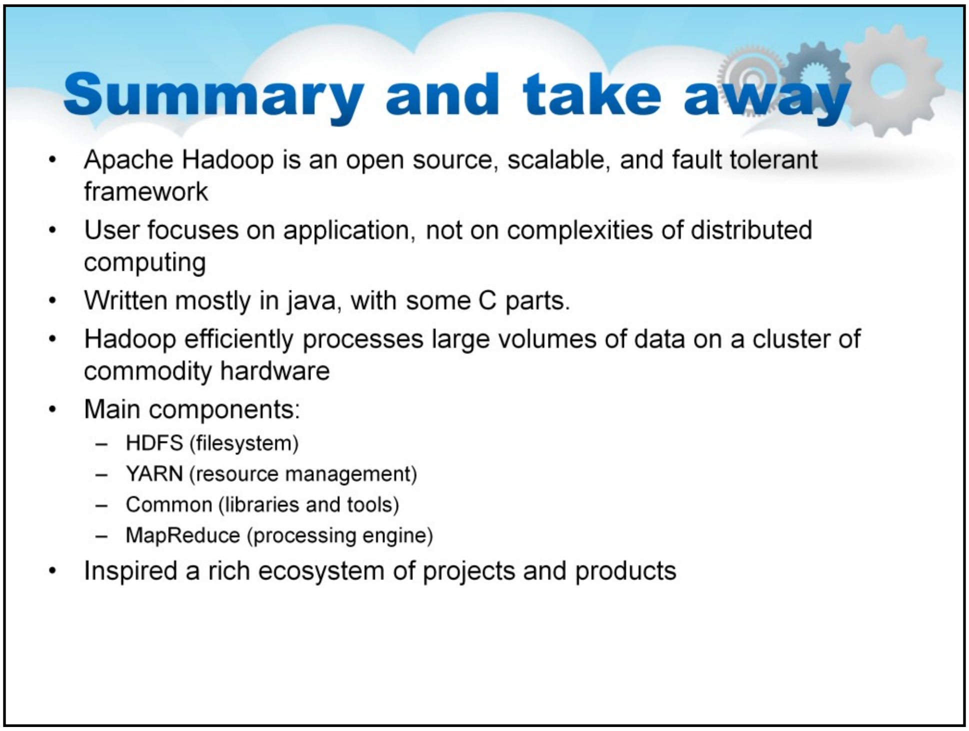
HBase is an open-source, distributed, versioned, non-relational database modeled after Google's Bigtable paper. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.
Linear and modular scalability.
Automatic and configurable sharding of tables
Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.



Apache HBase vs RDBMS

HBase	RDBMS
schema-less	schema
built for wide tables, horizontally scalable	built for small tables. Hard to scale
denormalized data	normalized data
semi-structured as well as structured data	good for structured data

Here is a HBase vs RDBMS comparison table.
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families. An RDBMS is governed by its schema, which describes the whole structure of tables.
HBase is built for wide tables and horizontally scalable. RDBMS is thin and built for small tables. Hard to scale. No transactions are there in HBase, but most of RDBMS are transactional.
HBase has de-normalized data, but RDBMS have normalized data.
HBase is good for semi-structured as well as structured data, and RDBMS is good for structured data.



Summary and take away

- Apache Hadoop is an open source, scalable, and fault tolerant framework
- User focuses on application, not on complexities of distributed computing
- Written mostly in java, with some C parts.
- Hadoop efficiently processes large volumes of data on a cluster of commodity hardware
- Main components:
  - HDFS (filesystem)
  - YARN (resource management)
  - Common (libraries and tools)
  - MapReduce (processing engine)
- Inspired a rich ecosystem of projects and products

Apache Hadoop is an open source, scalable,
and fault tolerant framework
Hadoop is mainly written in java. Some parts of it is written in C.
User focuses on application, not on complexities of
distributed computing
Hadoop efficiently processes large volumes of data on a cluster of
commodity hardware Main components:

HDFS (filesystem)
YARN (resource management) Common
(libraries and tools) MapReduce (processing
engine)

Apache Hadoop has inspired a rich ecosystem of projects and products
that benefit everyone interested in Big Data

Cloud Computing

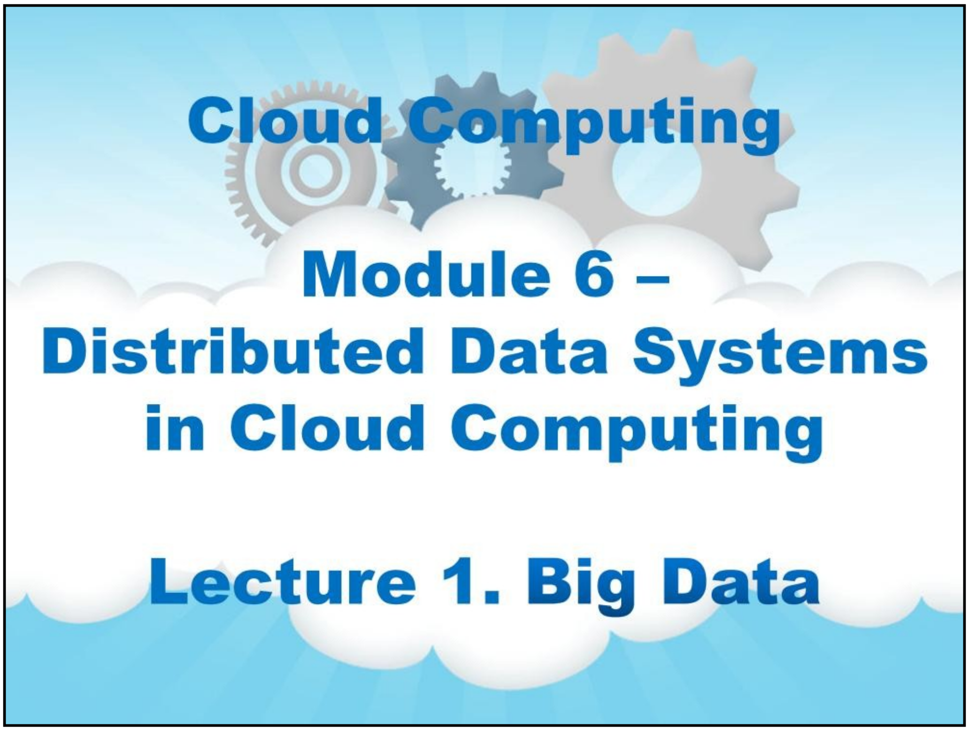
Lecture Manual

Volume 6

Module 6

Distributed Data Systems in Cloud Computing

Content

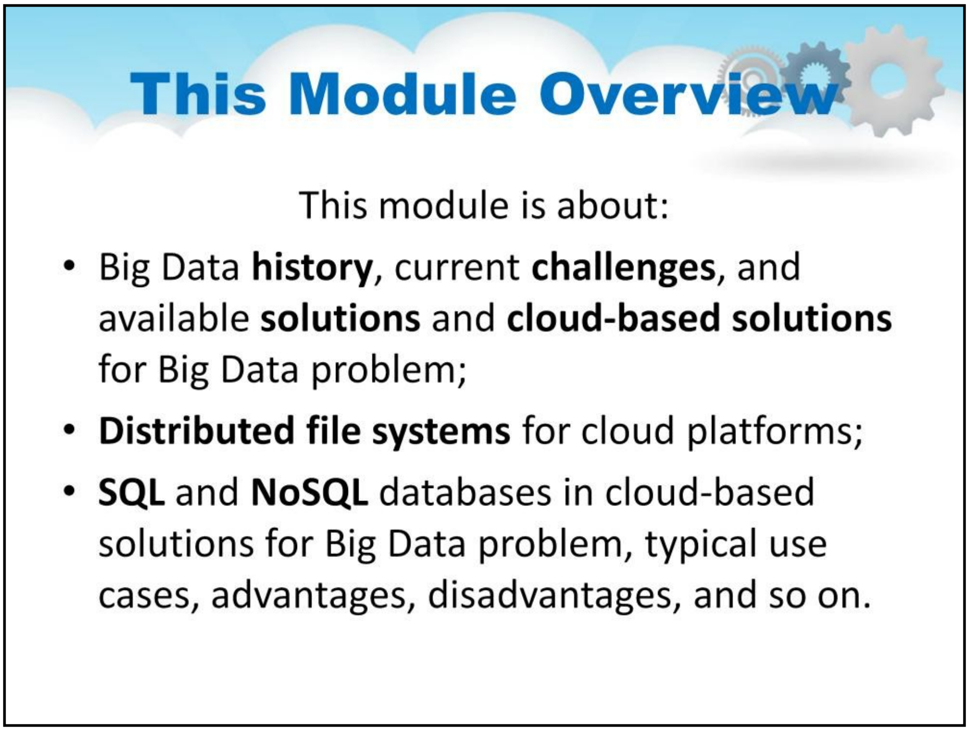


Cloud Computing

Module 6 – Distributed Data Systems in Cloud Computing

Lecture 1. Big Data

The title of this module is “Distributed Data Systems in Cloud Computing” This is the lecture number 1 about the problem of Big Data and its context.



This Module Overview

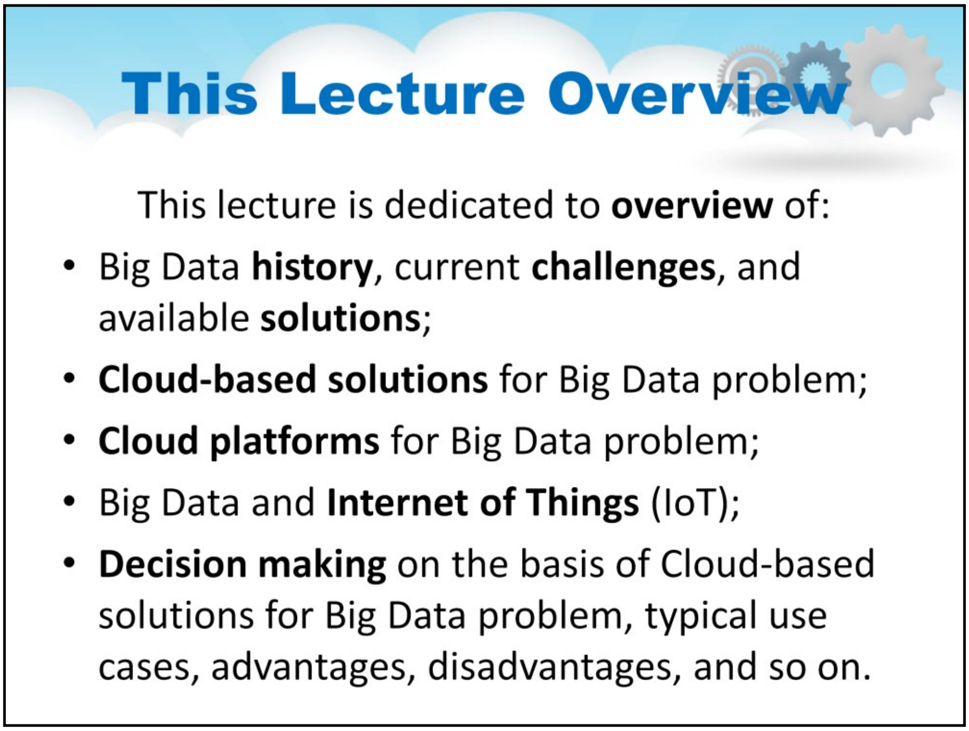
This module is about:

- Big Data **history**, current **challenges**, and available **solutions** and **cloud-based solutions** for Big Data problem;
- **Distributed file systems** for cloud platforms;
- **SQL** and **NoSQL** databases in cloud-based solutions for Big Data problem, typical use cases, advantages, disadvantages, and so on.

Module 6. Distributed Data Systems in Cloud Computing

This module is about:

• Big Data **history**, current **challenges**, and available **solutions** and **cloud-based solutions** for Big Data problem;
• **Distributed file systems** for cloud platforms;
• **SQL** and **NoSQL** databases in cloud-based solutions for Big Data problem, typical use cases, advantages, disadvantages, and so on.



This Lecture Overview

This lecture is dedicated to **overview** of:

- Big Data **history**, current **challenges**, and available **solutions**;
- **Cloud-based solutions** for Big Data problem;
- **Cloud platforms** for Big Data problem;
- Big Data and **Internet of Things** (IoT);
- **Decision making** on the basis of Cloud-based solutions for Big Data problem, typical use cases, advantages, disadvantages, and so on.

Lecture 1. Big Data

This lecture is dedicated to **overview** of:

- Big Data **history**, current **challenges**, and available **solutions**;
- **Cloud-based solutions** for Big Data problem;
- **Cloud platforms** for Big Data problem;
- Big Data and **Internet of Things** (IoT);
- **Decision making** on the basis of Cloud-based solutions for Big Data problem, typical use cases, advantages, disadvantages, and so on.

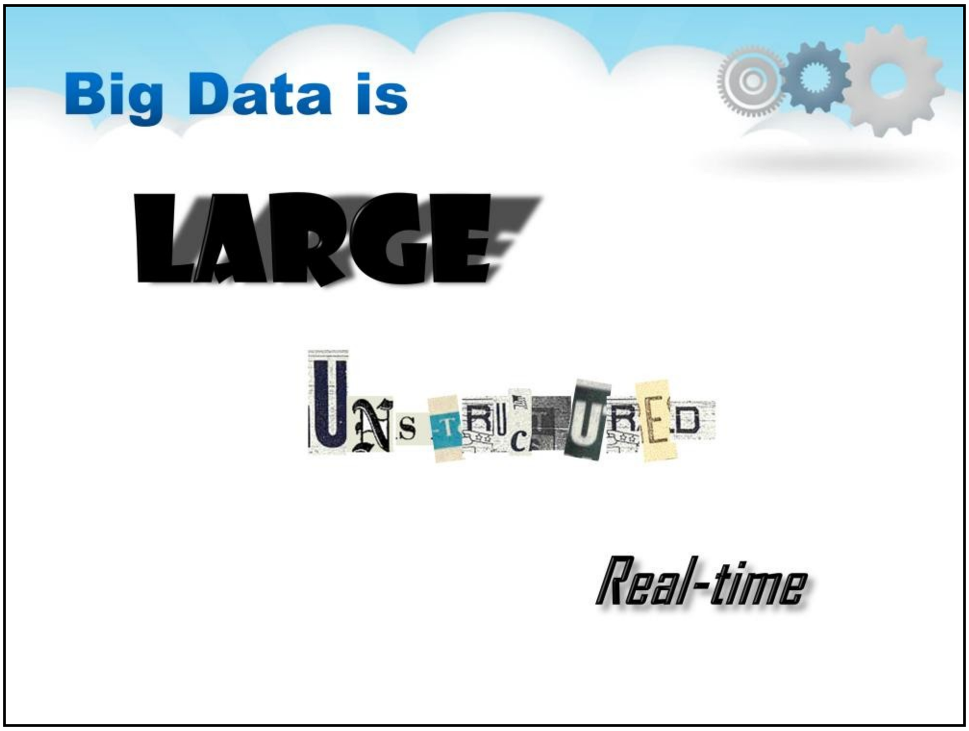
Big Data

–

Overview

Overview

Let’s start from the general view on the problem of Big Data…



Big Data is

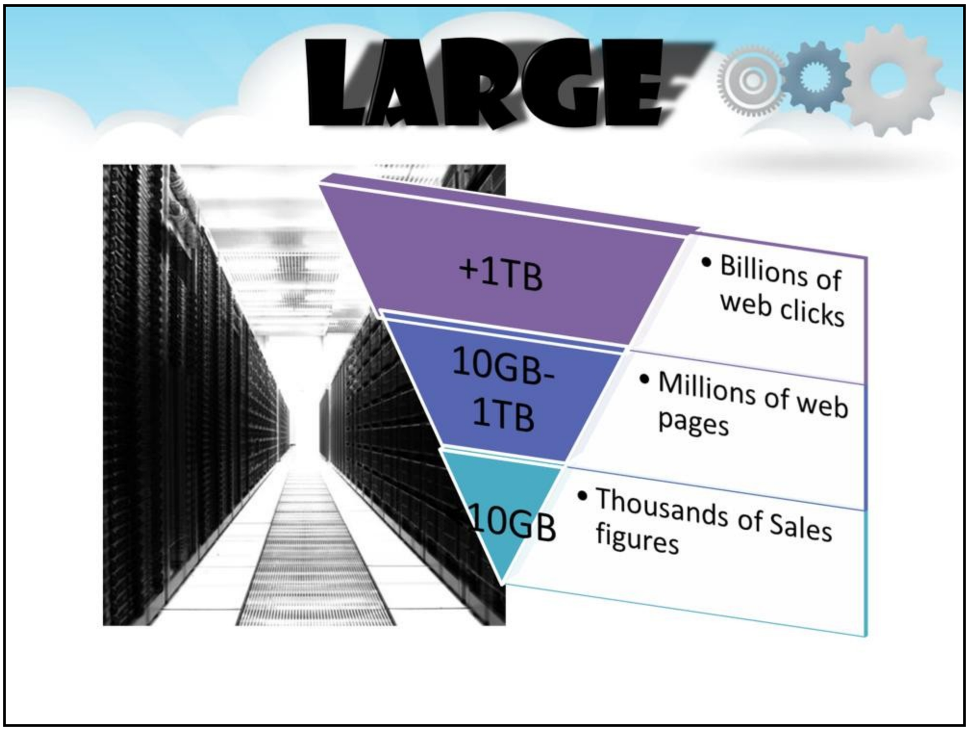
LARGE

UNSTRUCTURED

Real-time

Big Data is usually characterized as some bulk of data which is:

- Large
- Unstructured
- Real-Time



LARGE

+1TB

- Billions of web clicks

10GB-1TB

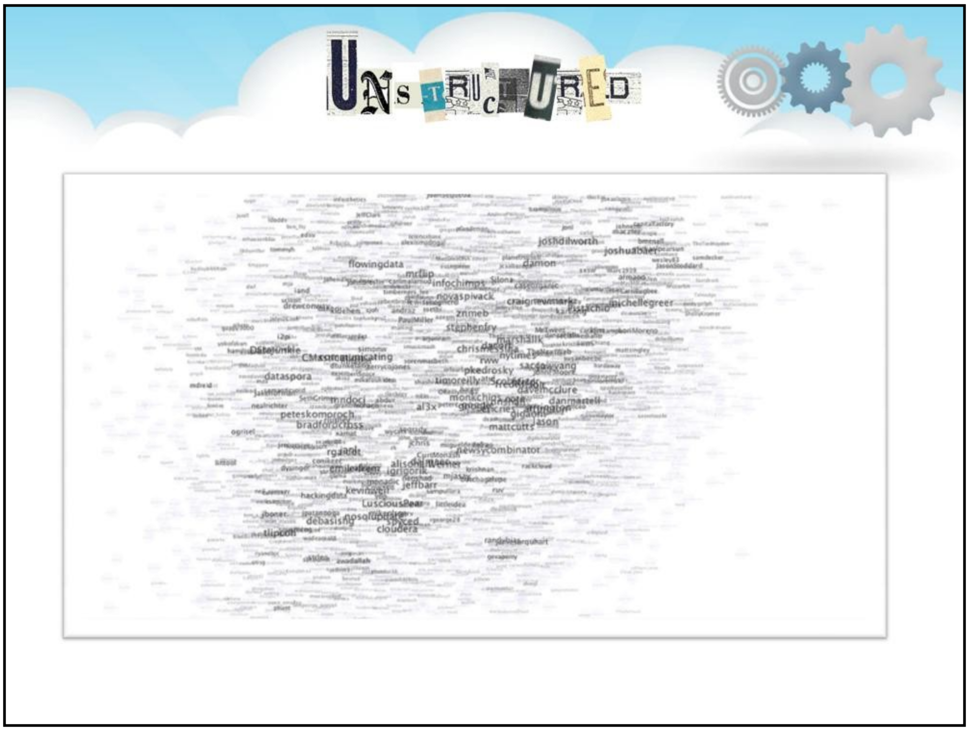
- Millions of web pages

10GB

- Thousands of Sales figures

Big Data has transformed from:

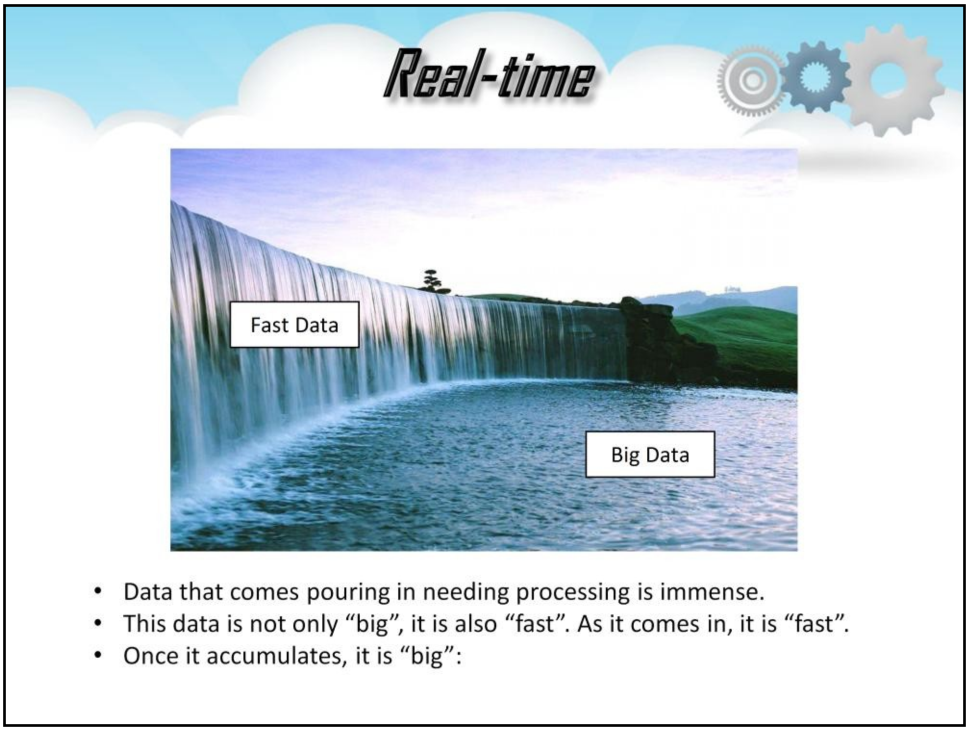
- Thousands: Databases with thousands of sales figures
- Millions: Millions of Web Pages
- Billions: of Web Clicks



UNSTRUCTURED

Big Data is Unstructured

The slide shows an illustration of web content, which stored and presented in Human Consumable forms, which us unstructured as far as the machines are concerned



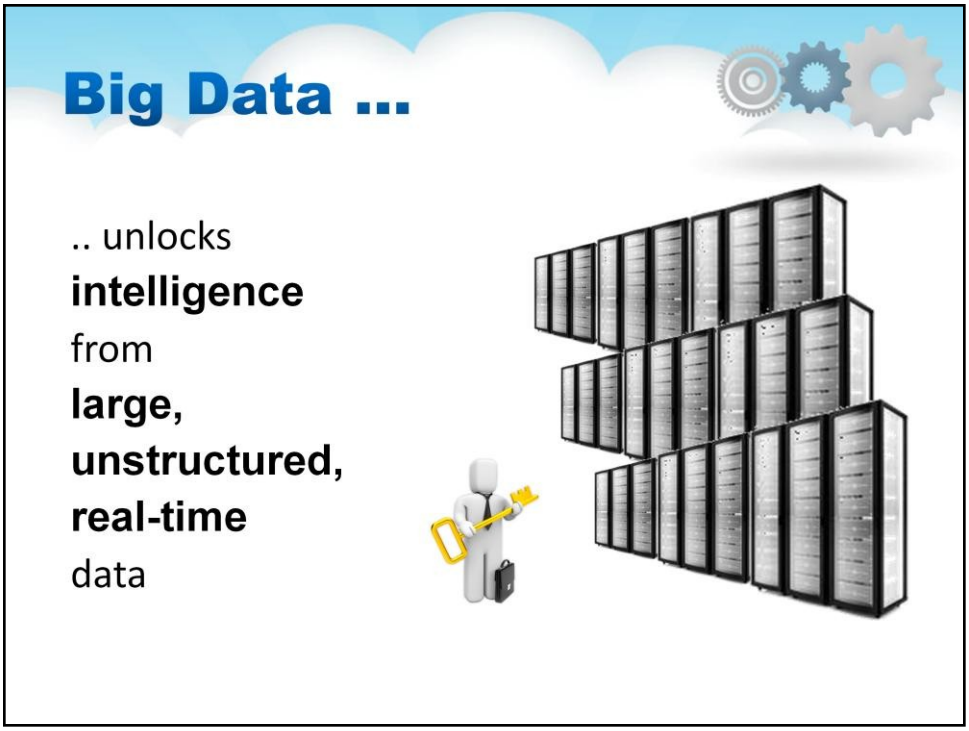
Real-time

Fast Data
Big Data

- Data that comes pouring in needing processing is immense.
- This data is not only “big”, it is also “fast”. As it comes in, it is “fast”.
- Once it accumulates, it is “big”:

Big Data is Real-Time

Fast Data can be information generated by software or hardware. It can be log files, a rapidly changing in-memory data set, sensor data from the Internet of Things (IoT), geospatial data from populations of mobile applications, messages from Email, Twitter, or SMS, weather or stock market information, or intelligence information from the battlefield.



Big Data ...

.. unlocks

intelligence

from

large,

unstructured,

real-time

data

What is Big Data?

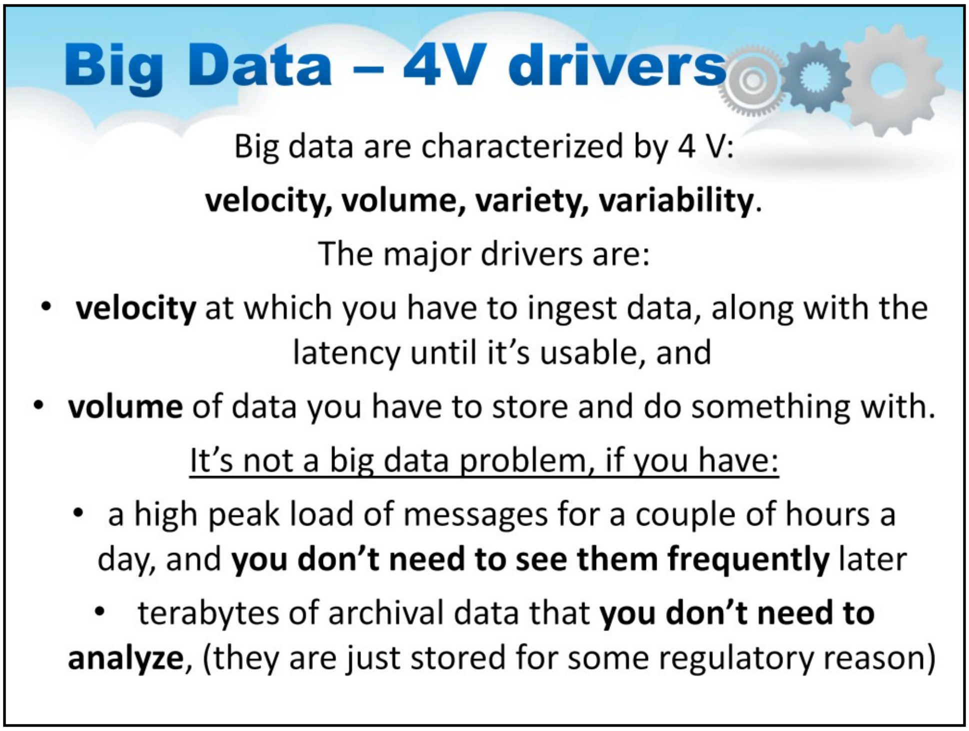
Big Data unlocks

intelligence

from

large,
unstructured,
real-time

data



Big Data – 4V drivers

Big data are characterized by 4 V:

velocity, volume, variety, variability.

The major drivers are:

- **velocity** at which you have to ingest data, along with the latency until it's usable, and
- **volume** of data you have to store and do something with.

<u>It's not a big data problem, if you have:</u>

- a high peak load of messages for a couple of hours a day, and **you don't need to see them frequently** later
- terabytes of archival data that **you don't need to analyze**, (they are just stored for some regulatory reason)

As it was outlined in the introductory lecture 0, the current problem of Big Data related with processing of the high volume of data generated by Internet of People, Internet of Things, and Internet of Everything.

Big Data are characterized by **4 V:**

velocity, volume, variety, variability.

The major drivers are:

- **velocity** at which you have to ingest data, along with the latency until it's usable, and
- **volume** of data you have to store and do something with.

<u>But it's not a big data problem, if you have:</u>

- a high peak load of messages for a couple of hours a day, and **you don't need to see them frequently** later
- terabytes of archival data that **you don't need to analyze**, (they are just stored for some regulatory reason)

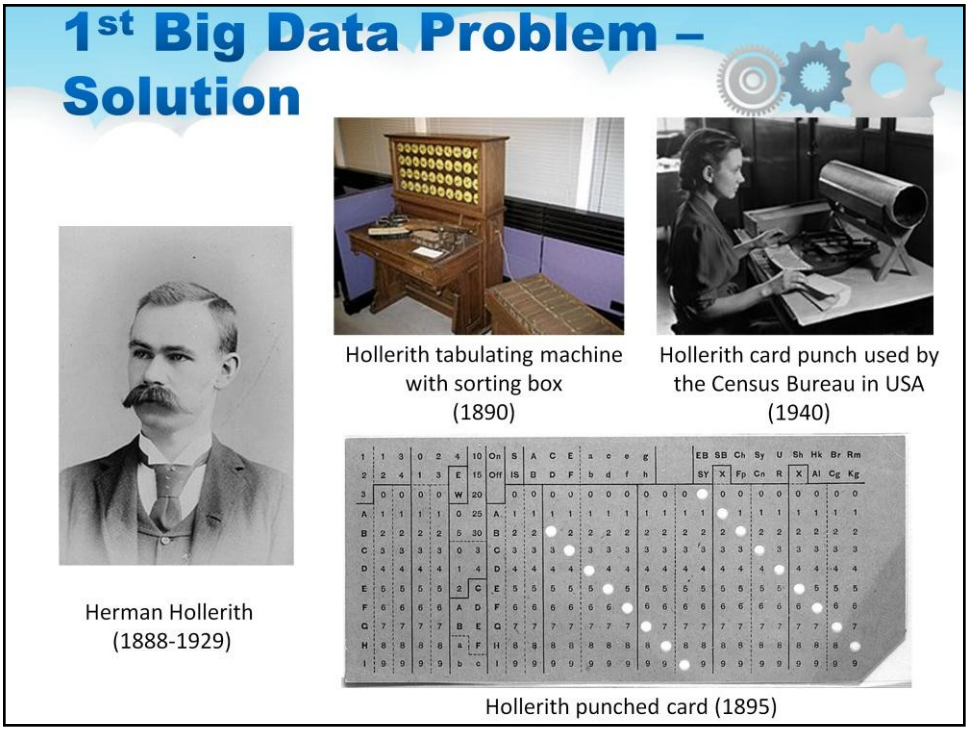
Big Data

–

History

History

Let’s consider the history and evolution of the Big Data problem…



1st Big Data Problem – Solution

Herman Hollerith
(1888-1929)

Hollerith tabulating machine with sorting box
(1890)

Hollerith card punch used by the Census Bureau in USA
(1940)

Hollerith punched card (1895)

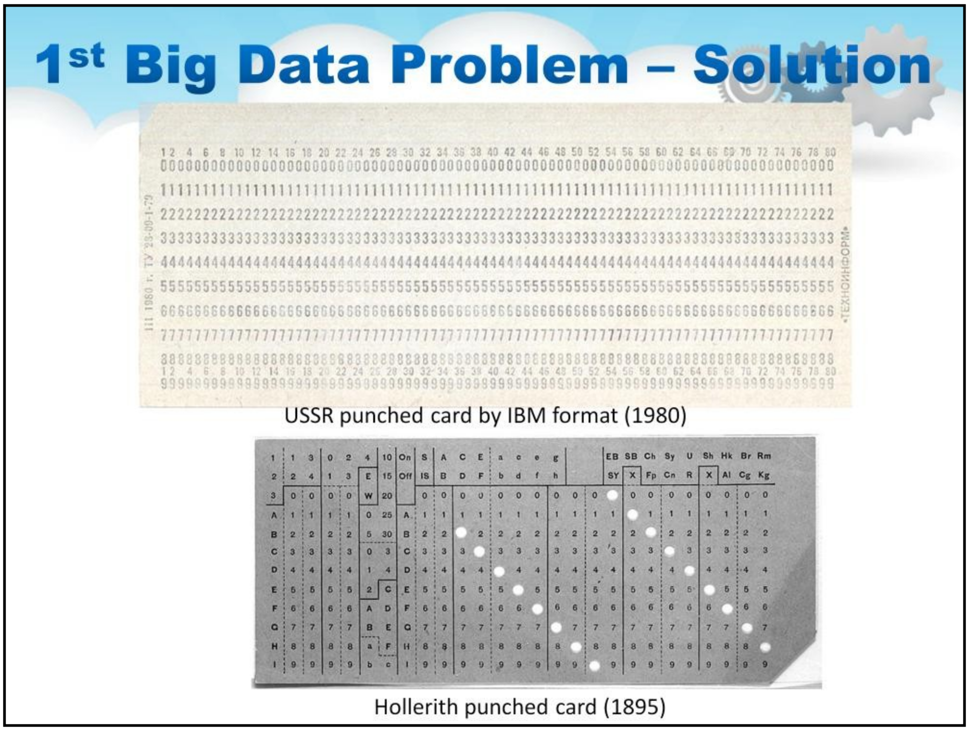
Let’s recall the introductory lecture 0, where we considered the first big data problem.

It occurred in the 1880s.

In the late 1800s, the processing of the U.S. census was beginning to take close to 10 years. The census runs **every 10 years** and the population, and thus the amount of information was increasing — **problem**!

In 1886, Herman Hollerith started a business to rent machines that could read and tabulate census data on punch cards. The 1890 census took <2 years to complete and handled a larger population (62 million people) and more data points than the 1880 census.

Later Hollerith’s business merged with three others to form what became IBM!



1st Big Data Problem – Solution

USSR punched card by IBM format (1980)

Hollerith punched card (1895)

In 1886, Herman Hollerith started a business to rent machines that could read and tabulate census data on punch cards. The 1890 census took <2 years to complete and handled a larger population (62 million people) and more data points than the 1880 census.

Later Hollerith's business merged with three others to form what became IBM!

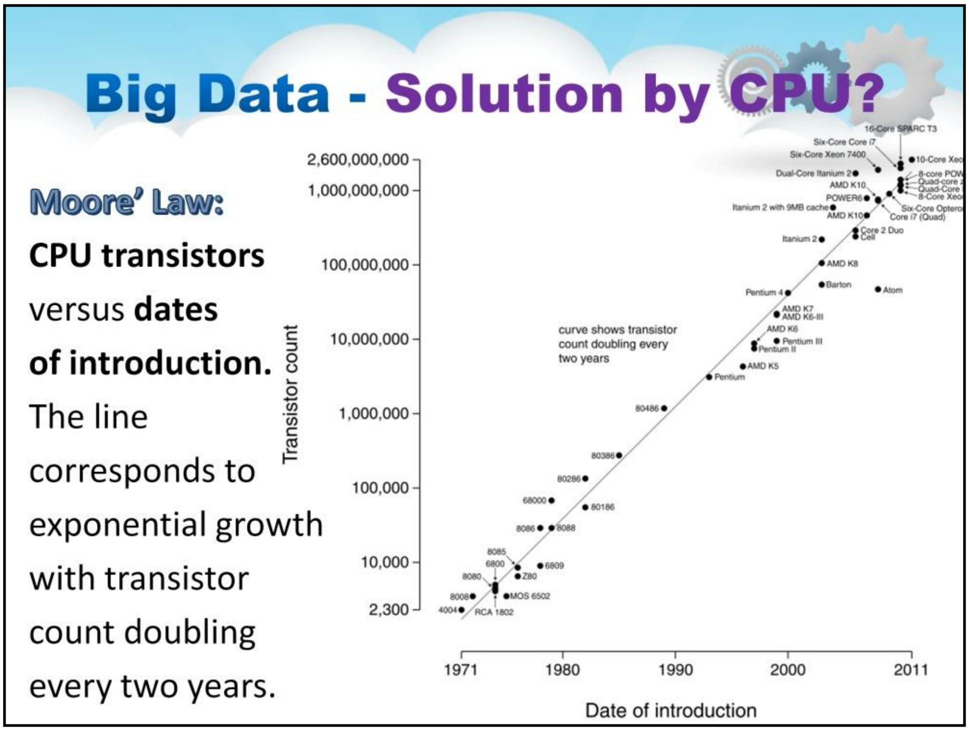
Big Data

–

Current Challenges

Current Challenges

Now, let’s recall the most crucial contemporary challenges…



Big Data - Solution by CPU?

Moore' Law:

CPU transistors versus **dates of introduction.** The line corresponds to exponential growth with transistor count doubling every two years.

Here you can see Moore' Law:

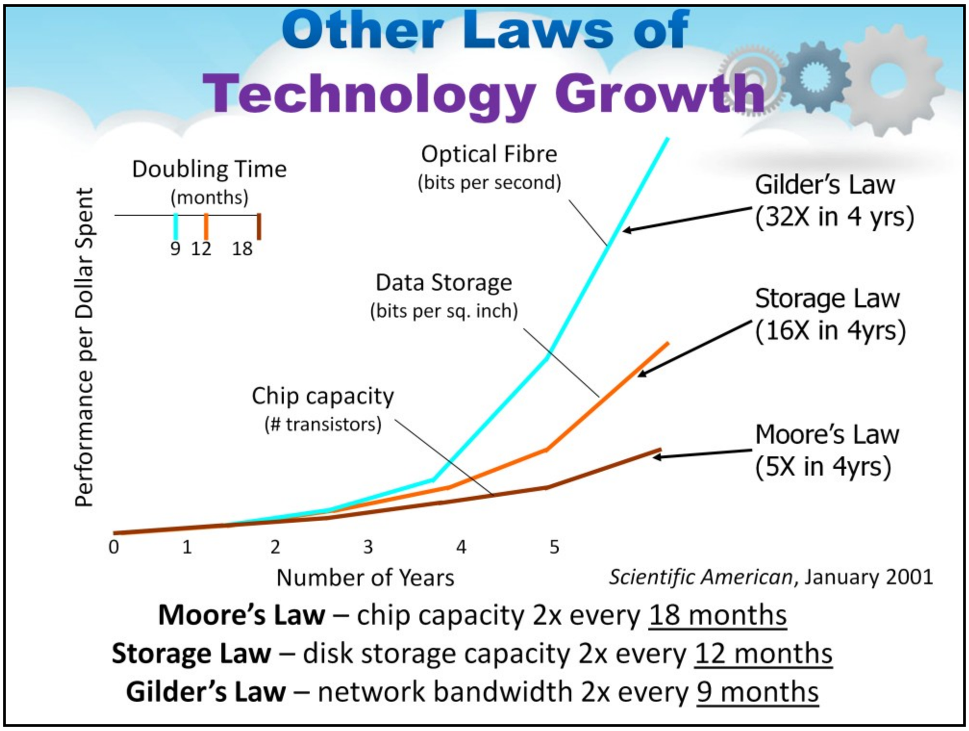
Dependence of the number of transistors inside CPU versus **dates of introduction.**

The line corresponds to exponential growth with transistor count doubling every two years.

You can see that this dependence is saturated during the last years due to physical limitations on the size of elements.

Gordon Moore said in 2005:
"In terms of size *of transistors+ you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far—but that's as far out as we've ever been able to see. We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions."

But now the current development of single Central Processing Unit (CPU) computing is not enough to response to the current challenges.



Other Laws of Technology Growth

Scientific American, January 2001

Moore's Law – chip capacity 2x every 18 months

Storage Law – disk storage capacity 2x every 12 months

Gilder's Law – network bandwidth 2x every 9 months

Now there are several other formulations of laws for characterization of technology growth:

Moore's Law – Individual computers double in processing power every 18 months

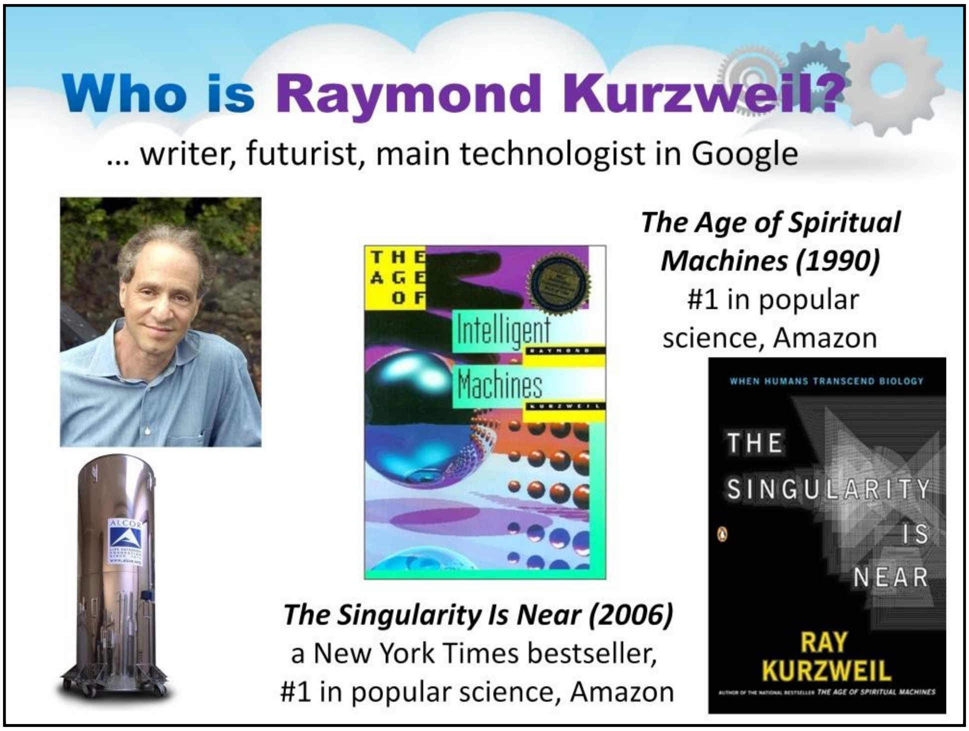
Storage Law – disk storage capacity doubles every 12 Months

Gilder's Law – Network bandwidth doubles every 9 months (but is harder to install)

This exponential growth profoundly changes the landscape of Information technology

More and more data created (because sensors are smaller and processors are cheaper) and stored (because disks are cheaper and more reliable than tape etc) and accessed remotely (because networking is cheaper).

Note that the time needed to fill available storage in increasing (i.e. the speed of writing data to disk is not increasing as quickly as the storage capacity)



Who is Raymond Kurzweil?

… writer, futurist, main technologist in Google

The Age of Spiritual Machines (1990)
#1 in popular science, Amazon

The Singularity Is Near (2006)
a New York Times bestseller,
#1 in popular science, Amazon

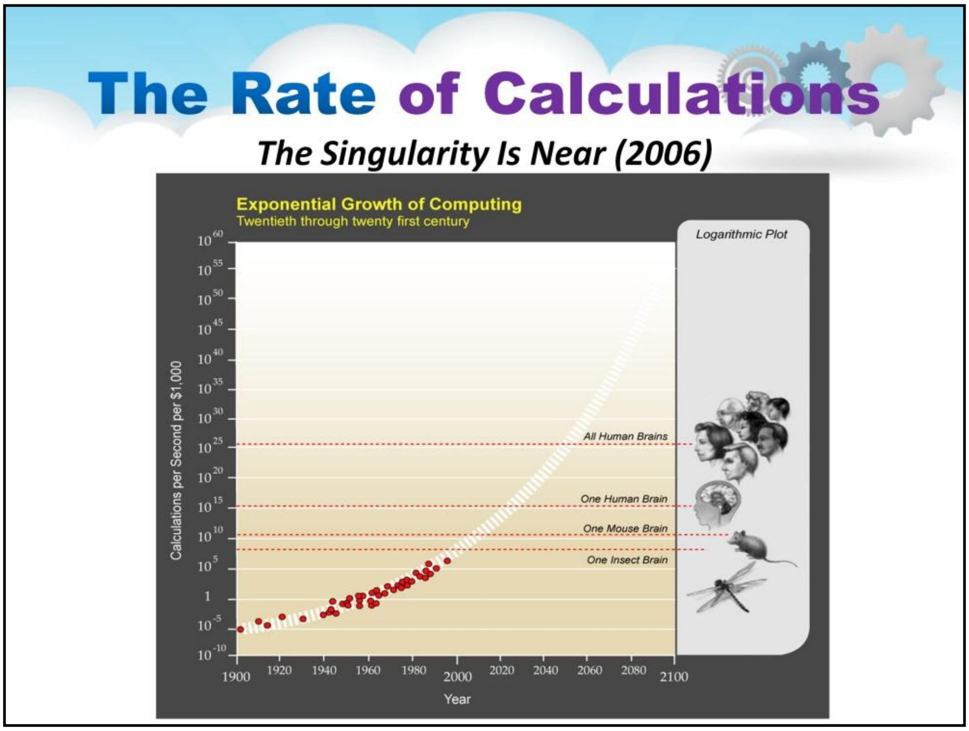
THE AGE OF Intelligent Machines
RAYMOND KURZWEIL
ALCOR
WHEN HUMANS TRANSCEND BIOLOGY
THE SINGULARITY IS NEAR
RAY KURZWEIL
AUTHOR OF THE NATIONAL BESTSELLER THE AGE OF SPIRITUAL MACHINES

Raymond Kurzweil is an American author, computer scientist, inventor and futurist.

Aside from futurism, he is involved in fields such as optical character recognition (OCR), text-to-speech synthesis, speech recognition technology, and electronic keyboard instruments.

He has written books on health, artificial intelligence (AI), singularity.

Kurzweil's first book, The Age of Intelligent Machines, presented his ideas about the future in 1990. Kurzweil extrapolated trends in the improvement of computer chess software performance to predict that computers would beat the best human players "by the year 2000". In May 1997, chess World Champion Garry Kasparov was defeated by IBM's Deep Blue computer in a well-publicized chess match.



The Rate of Calculations

The Singularity Is Near (2006)

Kurzweil suggests that this exponential technological growth is counter-intuitive to the way our brains perceive the world—since our brains were biologically inherited from humans living in a world that was linear and local—and, as a consequence, he claims it has encouraged great skepticism in his future projections.

In the context of the before mentioned challenges and Big Data problems, the need of the new paradigms for high-performance computing is very high and important.

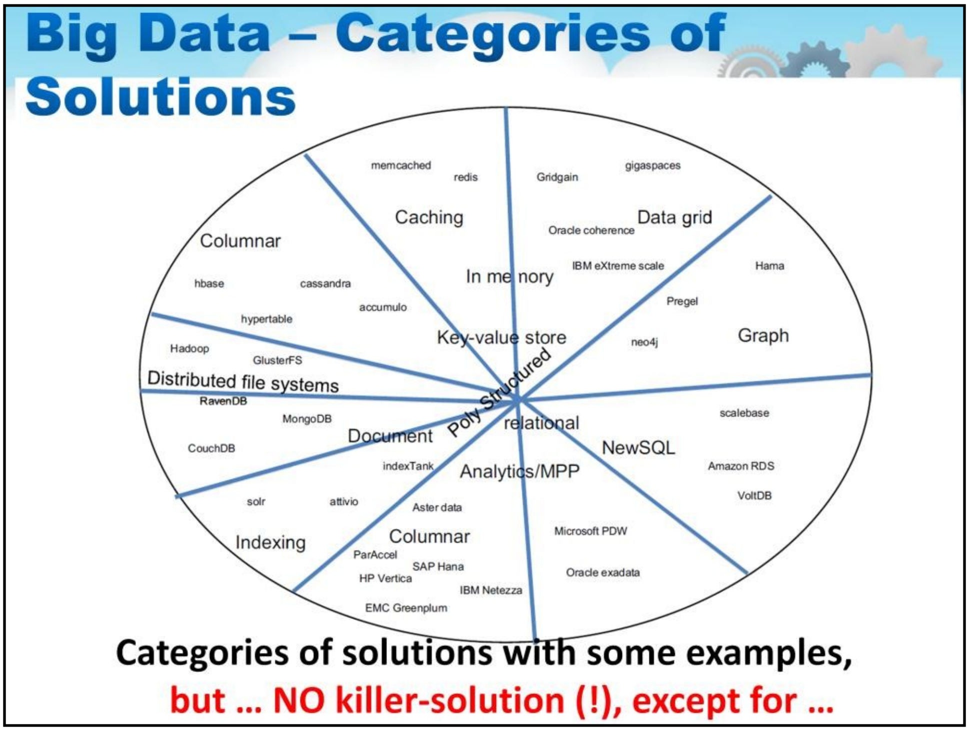
Big Data

–

Solutions?

Solutions

And what solutions can be proposed for Big Data problem…



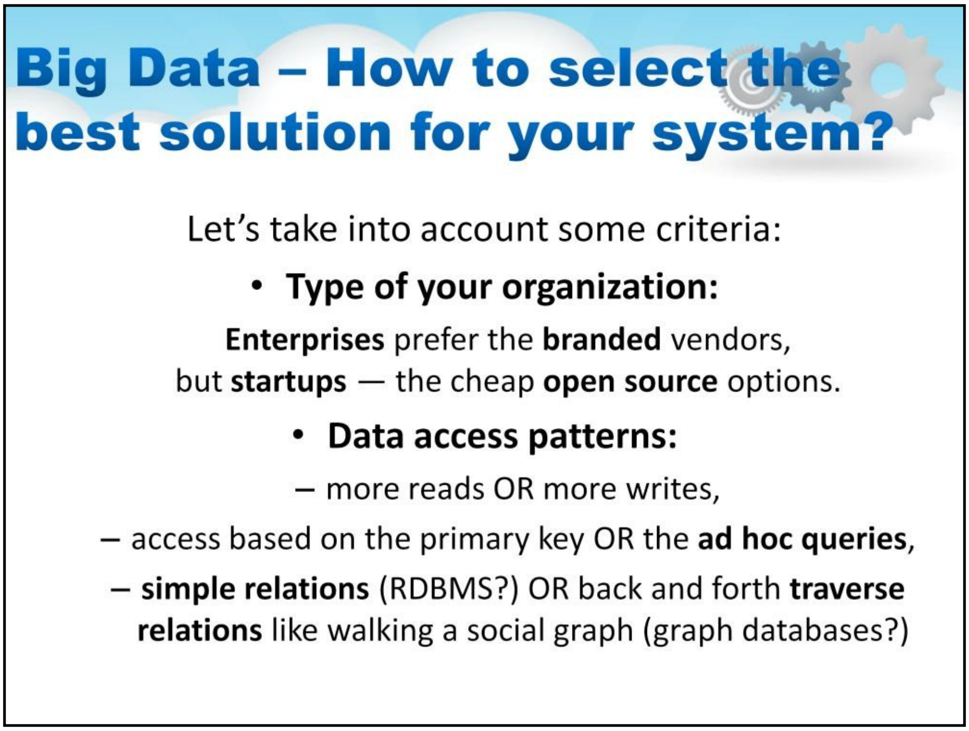
Big Data – Categories of Solutions

Categories of solutions with some examples,

but … NO killer-solution (!), except for …

This slide contains the map of numerous available tools and means for solving Bog data problem.

The fast conclusion is that we have not SINGLE solution for this problem.



Big Data – How to select the best solution for your system?

Let's take into account some criteria:

- **Type of your organization:**

Enterprises prefer the **branded** vendors,
but **startups** — the cheap **open source** options.

- **Data access patterns:**

– more reads OR more writes,

– access based on the primary key OR the **ad hoc queries**,

– **simple relations** (RDBMS?) OR back and forth **traverse relations** like walking a social graph (graph databases?)

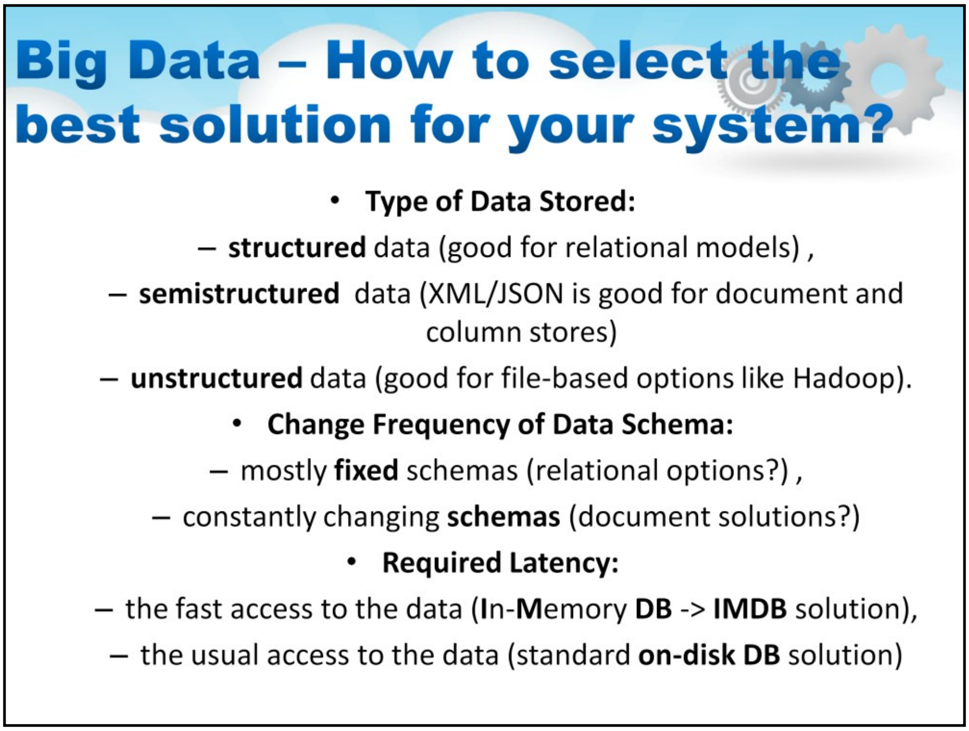
To select the best solution for your system we need to take into account the following criteria:

• **Type of your organization:**
Enterprises prefer the **branded** vendors,

but **startups** — the cheap **open source** options.

• **Data access patterns:**
more reads OR more writes,
access based on the primary key OR the **ad hoc queries**,

simple relations (RDBMS?) OR back and forth **traverse relations** like walking a social graph (graph databases?)



Big Data – How to select the best solution for your system?

- **Type of Data Stored:**
  - **structured** data (good for relational models) ,
  - **semistructured** data (XML/JSON is good for document and column stores)
  - **unstructured** data (good for file-based options like Hadoop).
- **Change Frequency of Data Schema:**
  - mostly **fixed** schemas (relational options?) ,
  - constantly changing **schemas** (document solutions?)
- **Required Latency:**
  - the fast access to the data (**I**n-**M**emory **DB** -> **IMDB** solution),
  - the usual access to the data (standard **on-disk DB** solution)

(continued from the previous slide)

- **Type of Data Stored:**

  structured data (good for relational models) ,

  semistructured data (XML/JSON is good for document and column stores)

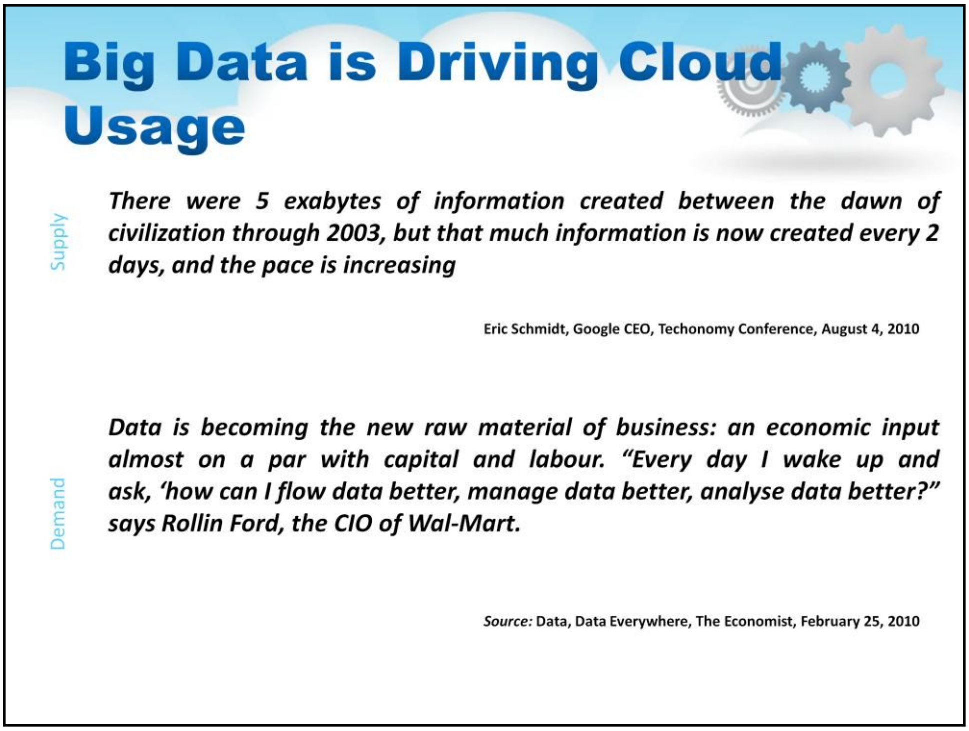
  unstructured data (good for file-based options like Hadoop).

- **Change Frequency of Data Schema:**

  mostly **fixed** schemas (relational options?) ,
  constantly changing **schemas** (document solutions?)

- **Required Latency:**

  the fast access to the data (**I**n-**M**emory **DB** -> **IMDB** solution),
  the usual access to the data (standard **on-disk DB** solution)



Big Data is Driving Cloud Usage

Supply

There were 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing

Eric Schmidt, Google CEO, Techonomy Conference, August 4, 2010

Demand

Data is becoming the new raw material of business: an economic input almost on a par with capital and labour. "Every day I wake up and ask, 'how can I flow data better, manage data better, analyse data better?" says Rollin Ford, the CIO of Wal-Mart.

Source: Data, Data Everywhere, The Economist, February 25, 2010

Big Data is Driving Cloud Usage

Over the last decade two profound trends have emerged in computing. Enterprises and Telecoms are transforming their IT infrastructures from collections of loosely integrated applications, each running on their own servers and with their own databases, into strategic, flexible, high performance platforms able to store every bit of information the enterprise has access to, with enough on-demand computing to provide for capabilities matching the imagination of the most creative CIO.

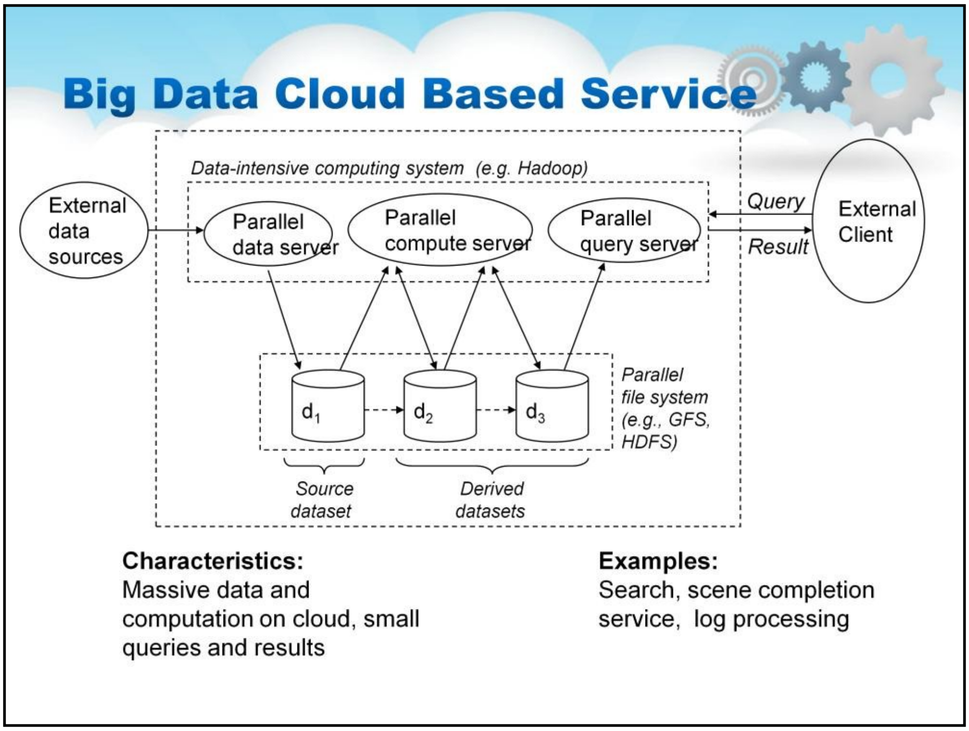
Big Data

–

Cloud-Based Solutions

Cloud-Based Solutions

Let’s take into account the available cloud-based solutions…



Big Data Cloud Based Service

Data-intensive computing system (e.g. Hadoop)

External data sources

Parallel data server

Parallel compute server

Parallel query server

Query

Result

External Client

d_1

d_2

d_3

Parallel file system (e.g., GFS, HDFS)

Source dataset

Derived datasets

Characteristics:
Massive data and computation on cloud, small queries and results

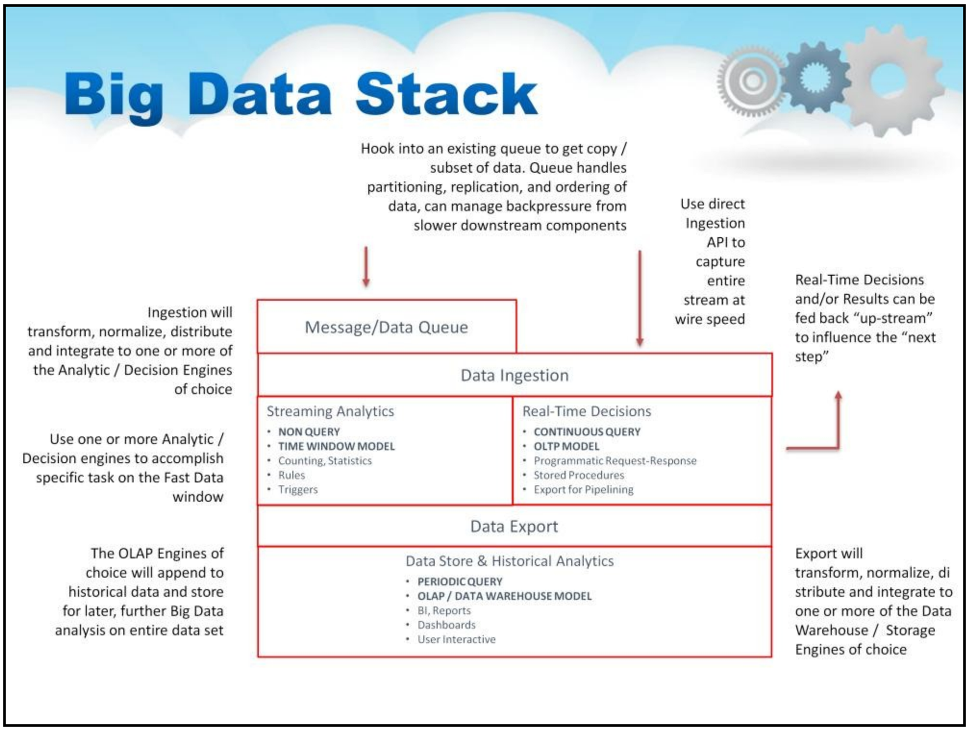
Examples:
Search, scene completion service, log processing

This slide illustrates how a Big Data Cloud Based Service works

Examples are Search, scene completion service, log processing

Characteristics are Massive data and computation on cloud, but driven by small queries and returning small results

The way Big Data is processed on a cloud is using a data pipeline, with a pipeline management system such as Hadoop.



Big Data Stack

Hook into an existing queue to get copy / subset of data. Queue handles partitioning, replication, and ordering of data, can manage backpressure from slower downstream components

Use direct Ingestion API to capture entire stream at wire speed

Ingestion will transform, normalize, distribute and integrate to one or more of the Analytic / Decision Engines of choice

Message/Data Queue

Data Ingestion

Real-Time Decisions and/or Results can be fed back "up-stream" to influence the "next step"

Use one or more Analytic / Decision engines to accomplish specific task on the Fast Data window

Streaming Analytics

- **NON QUERY**
- **TIME WINDOW MODEL**
- Counting, Statistics
- Rules
- Triggers

Real-Time Decisions

- **CONTINUOUS QUERY**
- **OLTP MODEL**
- Programmatic Request-Response
- Stored Procedures
- Export for Pipelining

Data Export

The OLAP Engines of choice will append to historical data and store for later, further Big Data analysis on entire data set

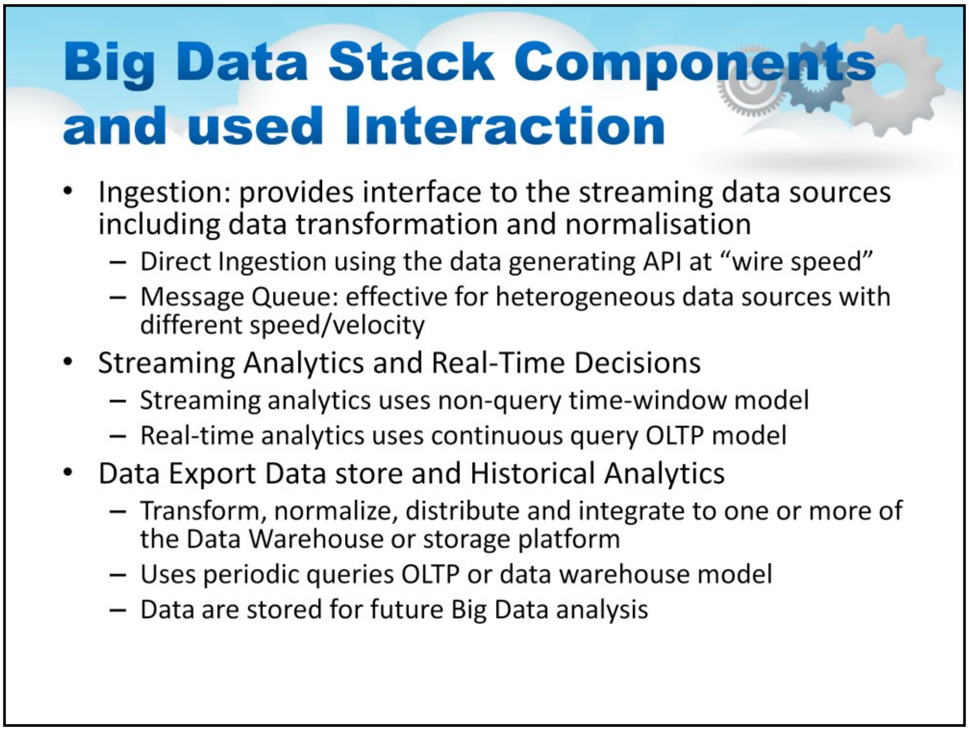
Data Store & Historical Analytics

- **PERIODIC QUERY**
- **OLAP / DATA WAREHOUSE MODEL**
- BI, Reports
- Dashboards
- User Interactive

Export will transform, normalize, di stribute and integrate to one or more of the Data Warehouse / Storage Engines of choice

The Big Data Stack

This "Fast Data/Big Data Stack" includes ingestion, analysis, and storage. But there are several arrangements and combinations used for different use cases.



Big Data Stack Components and used Interaction

- Ingestion: provides interface to the streaming data sources including data transformation and normalisation
  - Direct Ingestion using the data generating API at “wire speed”
  - Message Queue: effective for heterogeneous data sources with different speed/velocity
- Streaming Analytics and Real-Time Decisions
  - Streaming analytics uses non-query time-window model
  - Real-time analytics uses continuous query OLTP model
- Data Export Data store and Historical Analytics
  - Transform, normalize, distribute and integrate to one or more of the Data Warehouse or storage platform
  - Uses periodic queries OLTP or data warehouse model
  - Data are stored for future Big Data analysis

YD: This is combined notes from few slides

In the Fast and Big data stack, ingestion is the first stage. The job of ingestion is to interface to the streaming data sources, and as needed to transform and normalize the data. Then, ingestion will potentially partition the data stream, distributing to one or more of the Analytic/Decision Engines of choice.

There are two choices for ingestion. The first choice is to use “Direct Ingestion” if possible, where a straight-through code module can hook directly into the data generating API, capturing directly the entire stream at the speed that the API and the network will run, e.g., at “wire speed”. In this case the Analytic/Decision Engines will have a Direct Ingestion “adapter” and with some amount of coding the Analytic/Decision Engines can handle streams of data from an API pipeline and not have to stage or cache any of the data on disk.

If access to the data generating API is not available, usually the alternative is that the data is being served up in a Message Queue. In this case, an ingestion system is needed to hook into that existing queue to get copy/subset of data. The ingestion queue handles partitioning, replication, and ordering of data, can manage backpressure from slower downstream components.

Once the data in Ingested, it is ultimately handed to one or more Analytic/Decision engines to accomplish specific task on the data streaming in. At this point the data is Fast Data and the challenge for the Analytic/Decision engines is to consume the velocity of the data stream.

Real-Time Decisions provide enough analysis in time for Decisions to be fed back “up-stream” and influence the “next step” of the processing. Real-Time Decisions are doing a lot of work, in that they consume the velocity of the data stream and at the same time are usually processing complex logic, all in time to complete the Real-Time Decision feedback loop. Real-Time Decisions have the following characteristics:

Continuous Query mode
OLTP Analytics Model

Programmatic Request-Response
Stored Procedures

Export for Pipelining

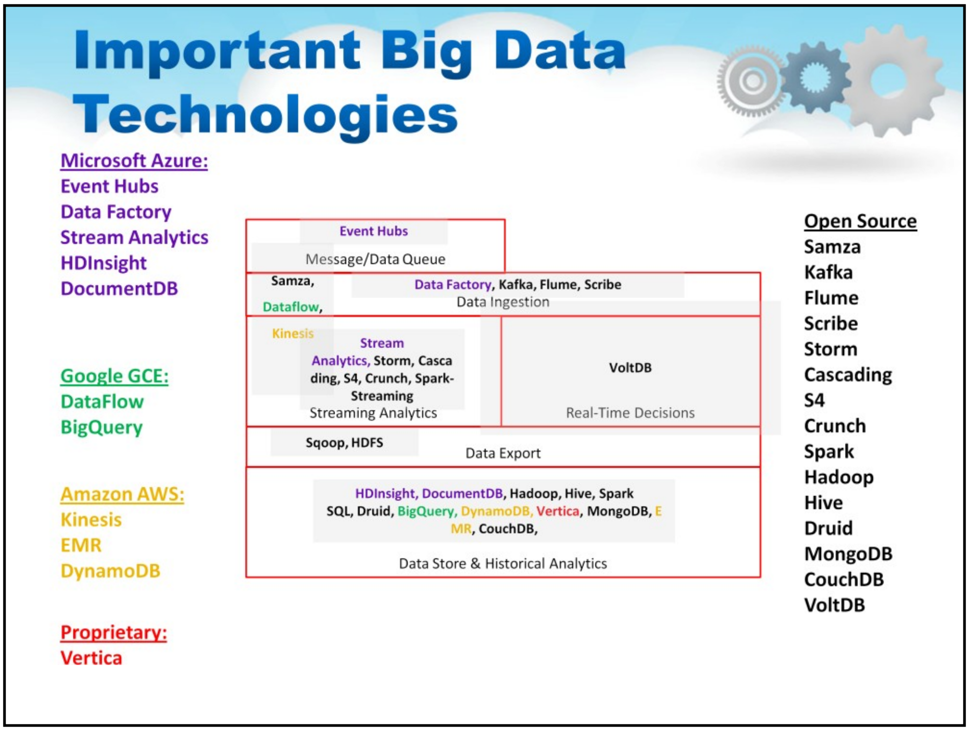
Streaming Analytics still need to consume the velocity of the data stream but are processing less complex logic, where the applications don’t require Real-Time feedback into the applications. Streaming Analytics have the following characteristics:

Non-Query mode

Time Window Processing
Counting, Statistics

Rules
Triggers

Once the Fast Data Analytics are completed, the data can continue to move along in the pipeline for later processing. Usually the Streaming Analytics relied on an Ingestion queue which is simply continued along to an Export stage. For Real-Time Decisions, which are processing the Fast Data in a Continuous Query mode, an Export function is needed to transform and distribute the data to the Big Data Warehouse/Storage (OLAP) Engines of choice.



Important Big Data Technologies

Microsoft Azure:
Event Hubs
Data Factory
Stream Analytics
HDInsight
DocumentDB

Google GCE:
DataFlow
BigQuery

Amazon AWS:
Kinesis
EMR
DynamoDB

Proprietary:
Vertica

Event Hubs
Message/Data Queue

Samza,
Dataflow,
Data Factory, Kafka, Flume, Scribe
Data Ingestion

Kinesis
Stream Analytics, Storm, Cascading, S4, Crunch, Spark-Streaming
Streaming Analytics

VoltDB
Real-Time Decisions

Sqoop, HDFS
Data Export

HDInsight, DocumentDB, Hadoop, Hive, Spark SQL, Druid, BigQuery, DynamoDB, Vertica, MongoDB, EMR, CouchDB,
Data Store & Historical Analytics

Open Source
Samza
Kafka
Flume
Scribe
Storm
Cascading
S4
Crunch
Spark
Hadoop
Hive
Druid
MongoDB
CouchDB
VoltDB

This slide illustrates important technologies in the Big Data Space.
Note some are from vendors, some are open source
Note also some span functional areas

Big Data

–

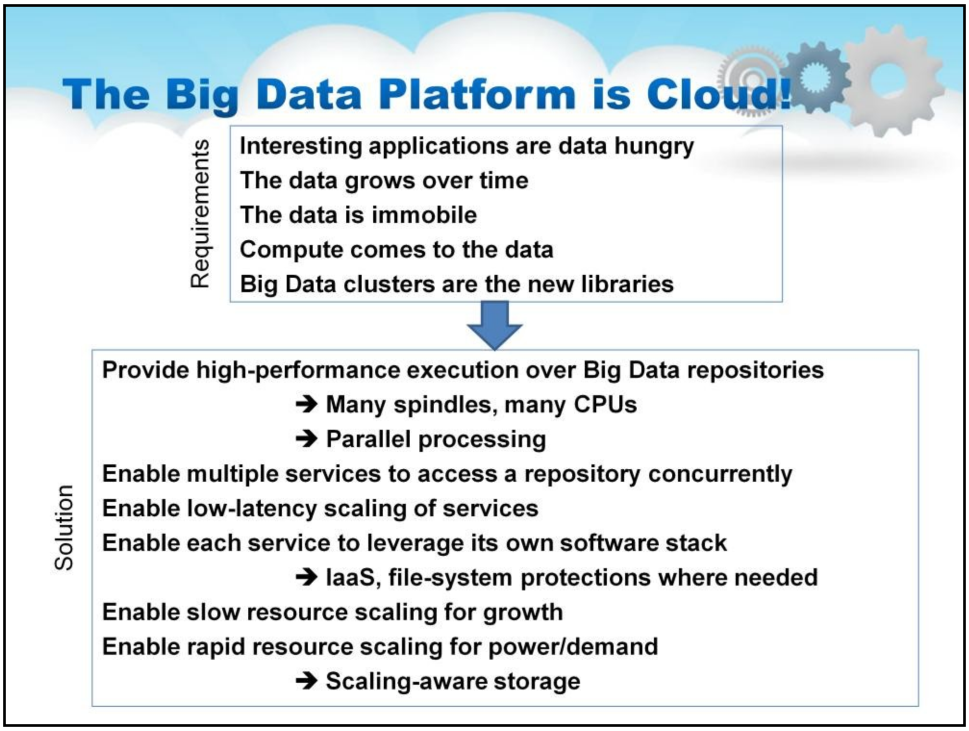
Cloud-Based Solutions

–

Cloud Platform

Cloud Platform

Now we go to the cloud platform for Big Data problem…



The Big Data Platform is Cloud!

Requirements

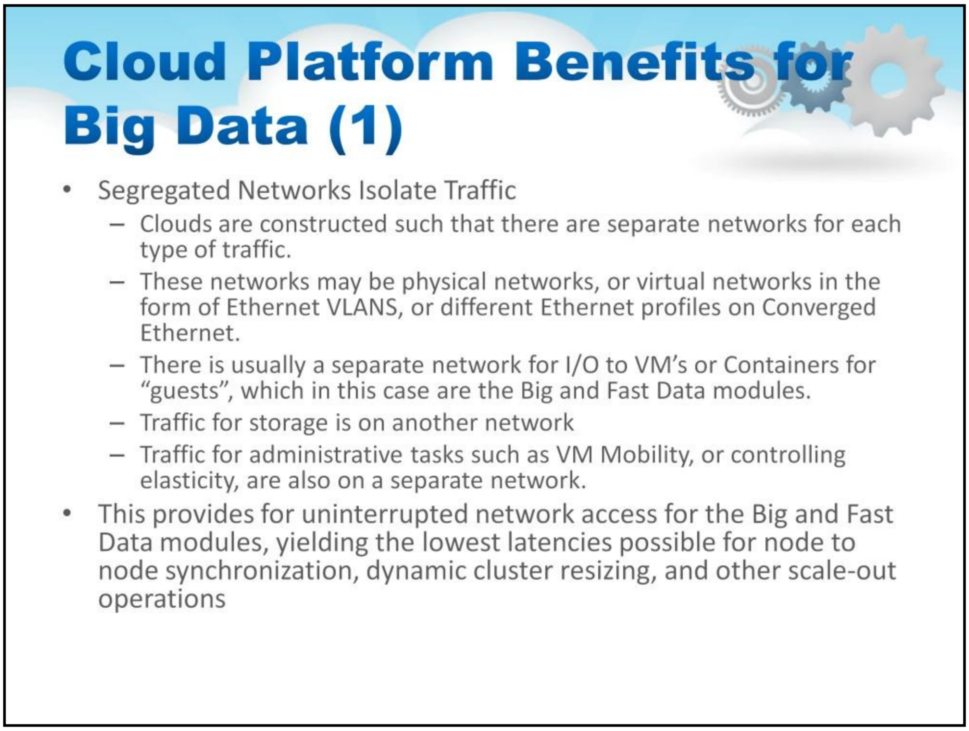
Interesting applications are data hungry
The data grows over time
The data is immobile
Compute comes to the data
Big Data clusters are the new libraries

Solution

Provide high-performance execution over Big Data repositories
➔ Many spindles, many CPUs
➔ Parallel processing
Enable multiple services to access a repository concurrently
Enable low-latency scaling of services
Enable each service to leverage its own software stack
➔ IaaS, file-system protections where needed
Enable slow resource scaling for growth
Enable rapid resource scaling for power/demand
➔ Scaling-aware storage

The Big Data Platform is Cloud!

Cloud Computing has emerged as a ubiquitous, essentially infinite computing platform serving the needs for companies and smart applications to process data – no matter how fast it comes in and no matter how large it gets. Clouds are designed to have incredible capacity to move data, to dynamically apply processing power to the data, and to store that data. We have learned that looking at the data in its entire lifecycle – as it arrives, and as it accumulates – provides powerful insights. The scale-out philosophy around which Clouds are designed allows for parallel, highly scalable mechanisms for launching networking, computing, and storage elements to handle all aspects of this processing. As the Internet has expanded into a seamless world of Desktop Browsers and Mobile Devices for which smart applications are available for every kind of task, the Cloud has become the engine which accomplishes all the massive processing behind these apps. A person's location, history, preferences, and "what's trending" are being leveraged to provide a brand new real-time on-line experience, all powered by the Cloud.

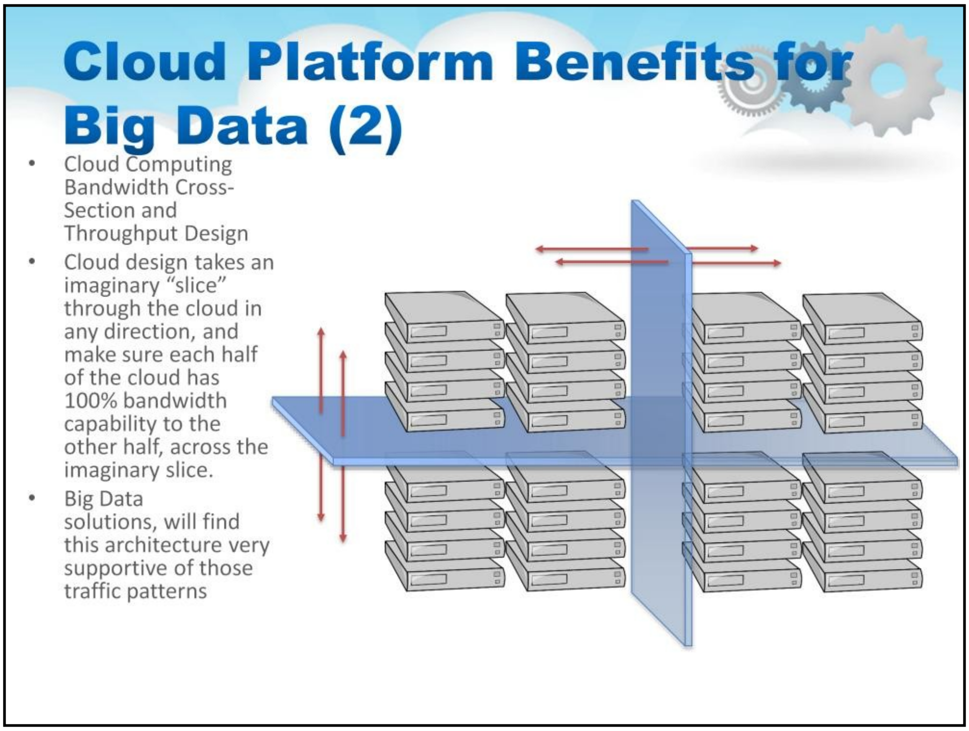


Cloud Platform Benefits for Big Data (1)

- Segregated Networks Isolate Traffic
  - Clouds are constructed such that there are separate networks for each type of traffic.
  - These networks may be physical networks, or virtual networks in the form of Ethernet VLANS, or different Ethernet profiles on Converged Ethernet.
  - There is usually a separate network for I/O to VM’s or Containers for “guests”, which in this case are the Big and Fast Data modules.
  - Traffic for storage is on another network
  - Traffic for administrative tasks such as VM Mobility, or controlling elasticity, are also on a separate network.
- This provides for uninterrupted network access for the Big and Fast Data modules, yielding the lowest latencies possible for node to node synchronization, dynamic cluster resizing, and other scale-out operations

In these following sections, then, some of the specific internal structures of Cloud will be examined, and examined to understand how synergistic there platform attributes are for Big and Fast Data solutions running on Cloud.

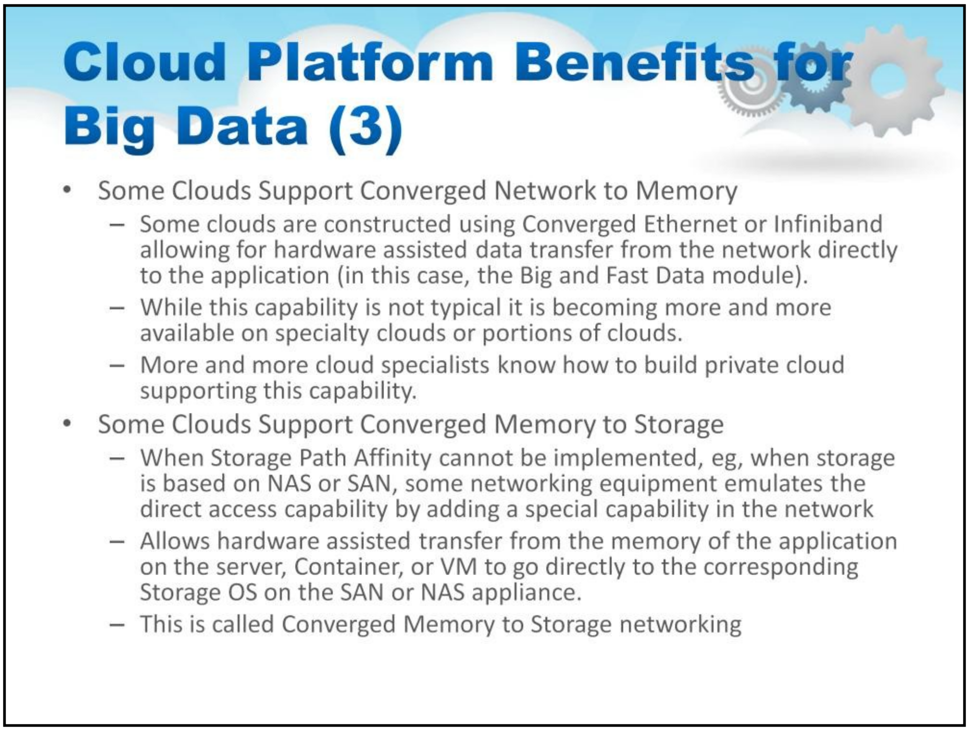
These comments apply to the way Public or Private clouds are constructed. One might only find some of the more esoteric features mentioned here in special “High Performance Computing (“HPC”) clouds, or sections of clouds (as a Public cloud example), or one might purposefully architect as such acquiring the particular software and hardware which enables these capabilities (as a Private cloud example).



Cloud Platform Benefits for Big Data (2)

- Cloud Computing Bandwidth Cross-Section and Throughput Design
- Cloud design takes an imaginary "slice" through the cloud in any direction, and make sure each half of the cloud has 100% bandwidth capability to the other half, across the imaginary slice.
- Big Data solutions, will find this architecture very supportive of those traffic patterns

Cloud instances are basically racks of servers connected together in an efficient manner. The figure below shows a Cloud instance with 4 racks of servers, 8 servers in each rack. Each server will contain multiple CPU chips and each CPU chip will contain multiple cores, each core able to run multiple threads. When building a Cloud instance, one must be careful not to oversubscribe the network in "upstream" aggregation points. One technique which is done is to take an imaginary "slice" through the cloud in any direction, and make sure each half of the cloud has 100% bandwidth capability to the other half, across the imaginary slice. As a result of this analysis, typical enterprise Top of Rack/Aggregation types of architectures have been replaced by "Leaf and Spine" architectures. The net is that Big and Fast Data solutions which scale-out and need to have inter-node communication, will find this architecture very supportive of those traffic patterns.



Cloud Platform Benefits for Big Data (3)

- Some Clouds Support Converged Network to Memory
  - Some clouds are constructed using Converged Ethernet or Infiniband allowing for hardware assisted data transfer from the network directly to the application (in this case, the Big and Fast Data module).
  - While this capability is not typical it is becoming more and more available on specialty clouds or portions of clouds.
  - More and more cloud specialists know how to build private cloud supporting this capability.
- Some Clouds Support Converged Memory to Storage
  - When Storage Path Affinity cannot be implemented, eg, when storage is based on NAS or SAN, some networking equipment emulates the direct access capability by adding a special capability in the network
  - Allows hardware assisted transfer from the memory of the application on the server, Container, or VM to go directly to the corresponding Storage OS on the SAN or NAS appliance.
  - This is called Converged Memory to Storage networking

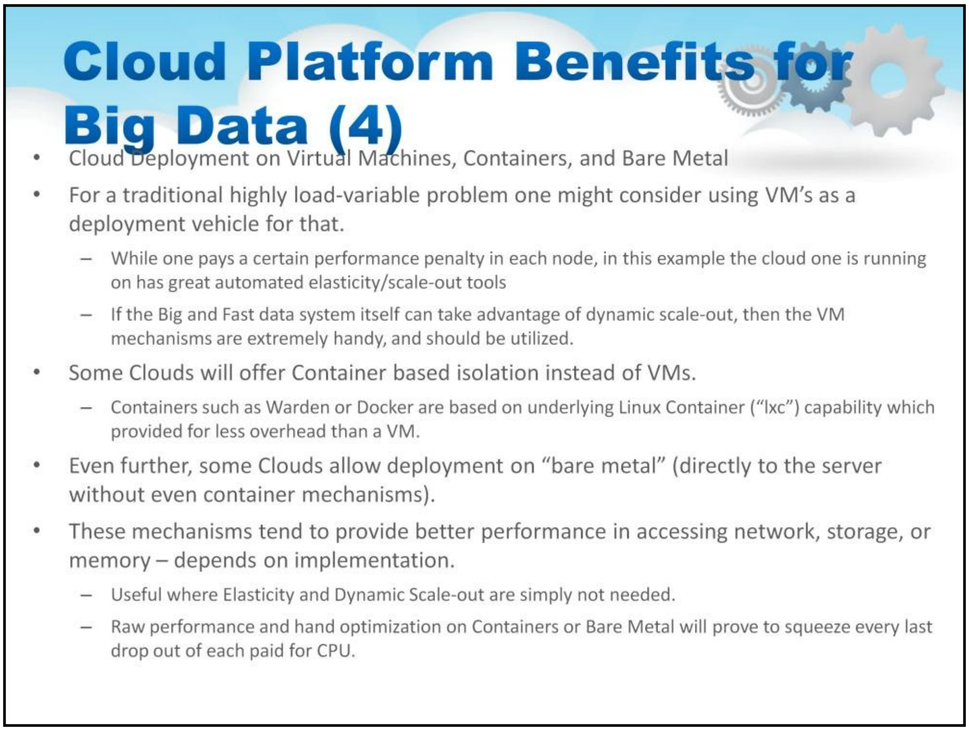
Some Clouds Support Converged Network to Memory

While looking at Cloud networking models, one can find certain clouds which use a special network capability designed to increase node to node networking drastically. Like most operating system drivers, standard network drivers are kernel functions in the operating systems of the cloud servers, which typically copy data from the network into the VM's or the Containers running the application modules. Some clouds are constructed using Converged Ethernet or Infiniband allowing for hardware assisted data transfer from the network directly to the application (in this case, the Big and Fast Data module). While this capability is not typical it is becoming more and more available on specialty clouds or portions of clouds.

More and more cloud specialists know how to build private cloud supporting this capability. It is a real "turbocharge" for certain classes of distributed applications. Big and Fast Data will be amongst the first to take advantage of this capability as it becomes more common.

Some Clouds Support Converged Memory to Storage

When Storage Path Affinity cannot be implemented, e.g., when storage is based on NAS or SAN, some networking equipment emulates the direct access capability by adding a special capability in the network allowing from hardware assisted transfer from the memory of the application on the server, Container, or VM to go directly to the corresponding Storage OS on the SAN or NAS appliance. This is called Converged Memory to Storage networking and is yet another of the esoteric optimizations found on cloud which are perfect for Big and Fast Data problems.



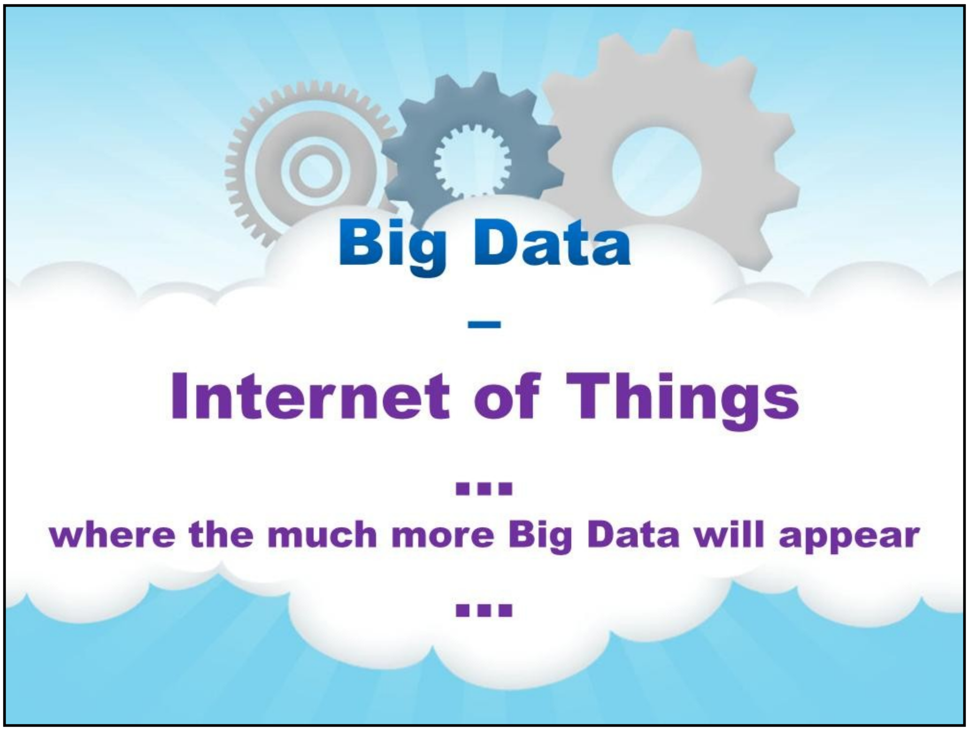
Cloud Platform Benefits for Big Data (4)

- Cloud Deployment on Virtual Machines, Containers, and Bare Metal
- For a traditional highly load-variable problem one might consider using VM’s as a deployment vehicle for that.
  - While one pays a certain performance penalty in each node, in this example the cloud one is running on has great automated elasticity/scale-out tools
  - If the Big and Fast data system itself can take advantage of dynamic scale-out, then the VM mechanisms are extremely handy, and should be utilized.
- Some Clouds will offer Container based isolation instead of VMs.
  - Containers such as Warden or Docker are based on underlying Linux Container (“lxc”) capability which provided for less overhead than a VM.
- Even further, some Clouds allow deployment on “bare metal” (directly to the server without even container mechanisms).
- These mechanisms tend to provide better performance in accessing network, storage, or memory – depends on implementation.
  - Useful where Elasticity and Dynamic Scale-out are simply not needed.
  - Raw performance and hand optimization on Containers or Bare Metal will prove to squeeze every last drop out of each paid for CPU.

Cloud Deployment Models – Virtual Machines, Containers, and Bare Metal

Whenever application performance is an issue, one needs to consider trade-offs in cost, manageability, and specific performance (latencies for example), across the various deployment options one has. For example, consider a highly load-variable problem. One might consider using VM’s as a deployment vehicle for that. While one pays a certain performance penalty in each node, in this example the cloud one is running on has great automated elasticity/scale-out tools, and if the Big and Fast data system itself can take advantage of dynamic scale-out, then the VM mechanisms are extremely handy, and should be utilized.

Some Clouds will offer Container based isolation instead of VMs. Containers such as Warden or Docker are based on underlying Linux Container (“lxc”) capability which provided for less overhead than a VM. Even further, some Clouds allow deployment on “bare metal” (directly to the server without even container mechanisms). These mechanisms tend to provide better performance in accessing network, storage, or memory – depends on implementation. Consider a rather static throughput and velocity Big and Fast Data problem set that may not need any infrastructure variability – Elasticity and Dynamic Scale-out are simply not needed. Therefore extensive automation based on VM’s may not be needed either. Raw performance and hand optimization on Containers or Bare Metal will prove to squeeze every last drop out of each paid for CPU.



Big Data

–

Internet of Things

…

where the much more Big Data will appear

…

Internet of Things

The next REALLY Big aspect is Internet of Things as a source much Bigger Data than ever…

What is IoT-solution?

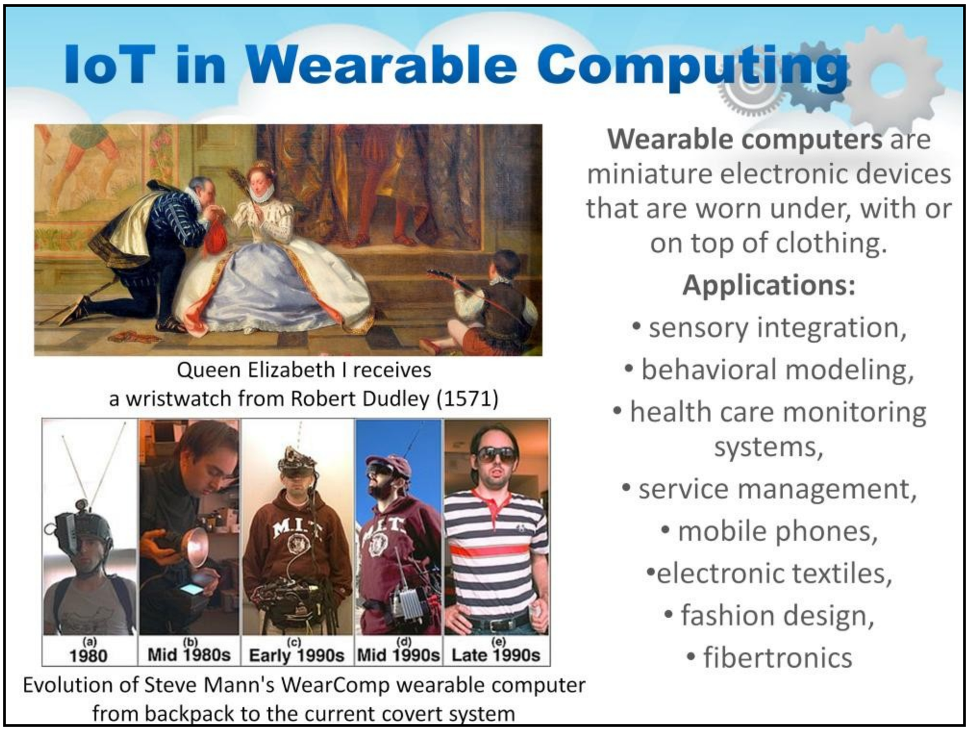
IoT is designed **to get data and perform actions about anything from anywhere at any time**

IoT (Internet of Things) is a set of interconnected devices that generate and exchange data from observations, facts, and other data, making it available to anyone. IoT solutions are designed to make our knowledge of the world around us more timely and relevant by making it possible to get data about anything from anywhere at any time. It is clear there is potential for the number of IOT devices to exceed the human population of the planet.

IOT solution is simply a set of devices designed to produce, consume, or present data about some event or series of events or observations. This can include devices that generate data such as a sensor, devices that combine data to deduce something, devices or services designed to tabulate and store the data, and devices or systems designed to present the data. Any or all of these may be connected to the Internet.

IOT solutions may include one or all of these qualities whether they are combined in a single device such as a web camera, use a sensor package and monitoring unit such as a weather station, or use a complex system of dedicated sensors, aggregators, data storage, and presentation such as a complete home automation system.

This popular picture shows a futuristic picture of all devices everywhere connected to the Internet via either databases, data collectors or integrators, display services, and even other devices.



IoT in Wearable Computing

Queen Elizabeth I receives
a wristwatch from Robert Dudley (1571)

(a) 1980
(b) Mid 1980s
(c) Early 1990s
(d) Mid 1990s
(e) Late 1990s

Evolution of Steve Mann's WearComp wearable computer
from backpack to the current covert system

Wearable computers are miniature electronic devices that are worn under, with or on top of clothing.

Applications:

- sensory integration,
- behavioral modeling,
- health care monitoring systems,
- service management,
- mobile phones,
- electronic textiles,
- fashion design,
- fibertronics

Gathering and understanding all that data is complicated by how the data is stored or more appropriately retrieved.

For example, what would happen if every device in your home, car, office, and so on, were to produce data?

Add in the view of the increasing interest to wearable sensors and similar devices and you’ve got the potential to generate more data than any human can manage or even decipher.

Let's observe the short history of wearable computing on this slide.

The modern IoT Example: Steve Mann - Wearable Pioneer

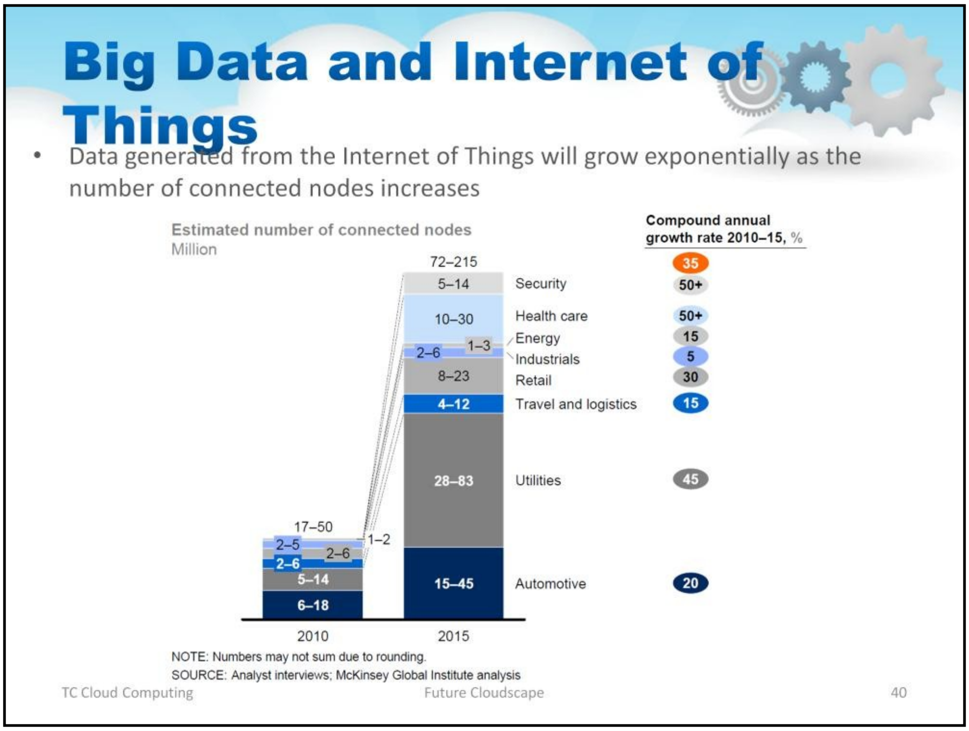
Steve Mann with 3 of his inventions:
- **EyeTap Digital Eye Glass**,
- **Smartwatch**,
- **SWIM (Sequential Wave Imprinting Machine)**

Phenomenological Augmented Reality visualizing radio waves from smartphone.

Steve Mann (1962)
- PhD in Media Arts (1997);
- seed of Wearable Computing group in MIT;
- now professor at the University of Toronto.

He is the "father of wearable computing".
He created the 1st general-purpose wearable computer, in contrast to wearable devices that perform 1 specific function like time-keeping (e.g. wristwatch)

This is Steve Mann and the short synopsis of his activities as a pioneer in wearable computing and the multimedia data produced by wearable gadgets.



Big Data and Internet of Things

- Data generated from the Internet of Things will grow exponentially as the number of connected nodes increases

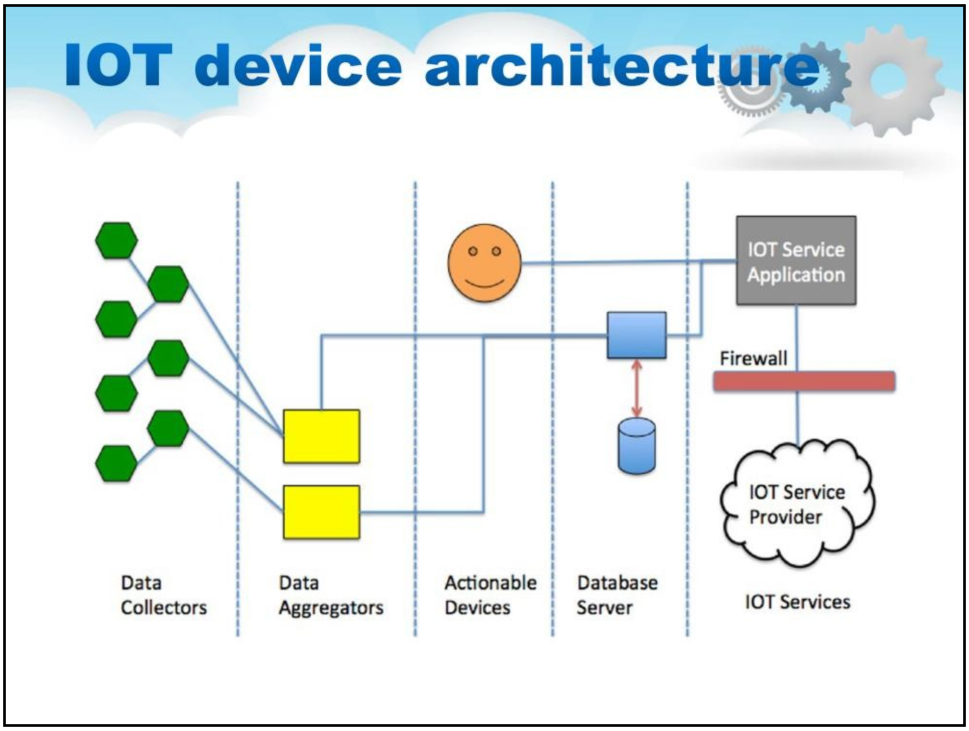
NOTE: Numbers may not sum due to rounding.

SOURCE: Analyst interviews; McKinsey Global Institute analysis

TC Cloud Computing Future Cloudscape 40

Big Data is further accelerated by the Internet of Things

The slide illustrates how Data generated from the Internet of Things will grow exponentially as the number of connected nodes increases



IOT device architecture

IOT Service Application

Firewall

IOT Service Provider

Data Collectors

Data Aggregators

Actionable Devices

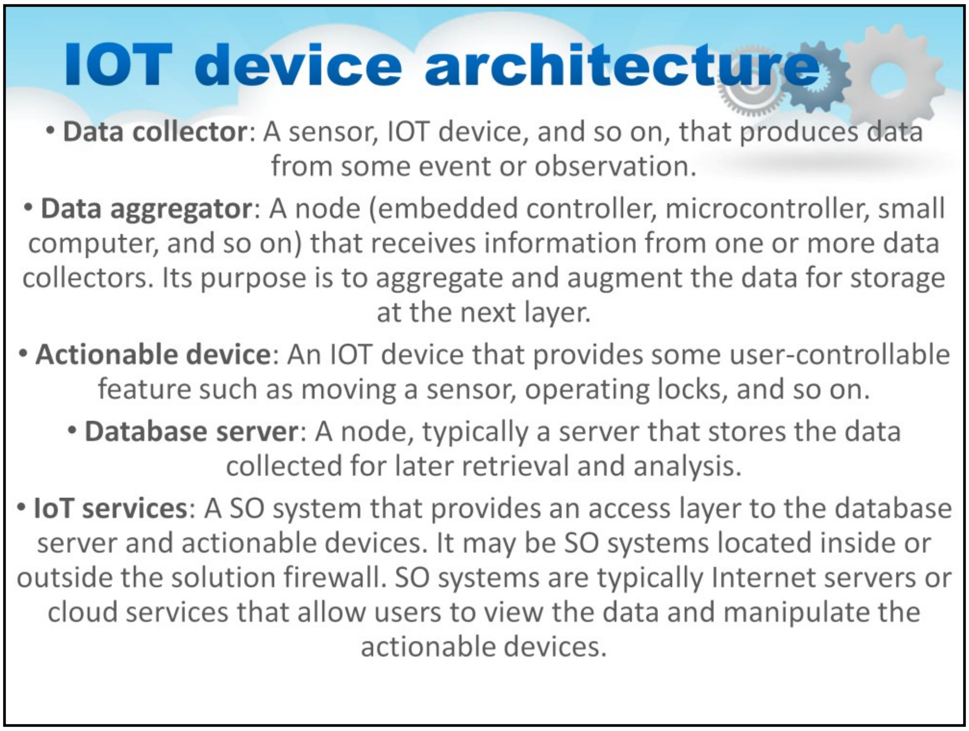
Database Server

IOT Services

On the left side of this slide are the IOT devices. These could be a simple sensor, an entire sensor network, a device with one or more sensors, an embedded microcontroller solution with sensors, a more sophisticated microprocessor-based solution, or even a device such as a microwave oven, alarm clock, or television. These devices are the more sophisticated devices with built-in networking capabilities.

At the center is a user accessing the data via the Internet or a cloud service. Naturally, this can be any type of device such as a
laptop, desktop, tablet, phone, watch, or other smart device or appliance (including another IOT device).

On the next slide let’s consider the roles in IoT device architecture presented at the bottom of this slide.



IOT device architecture

- **Data collector**: A sensor, IOT device, and so on, that produces data from some event or observation.
- **Data aggregator**: A node (embedded controller, microcontroller, small computer, and so on) that receives information from one or more data collectors. Its purpose is to aggregate and augment the data for storage at the next layer.
- **Actionable device**: An IOT device that provides some user-controllable feature such as moving a sensor, operating locks, and so on.
- **Database server**: A node, typically a server that stores the data collected for later retrieval and analysis.
- **IoT services**: A SO system that provides an access layer to the database server and actionable devices. It may be SO systems located inside or outside the solution firewall. SO systems are typically Internet servers or cloud services that allow users to view the data and manipulate the actionable devices.

Let’s consider the roles in IoT device architecture presented at the previous slide…

Big Data

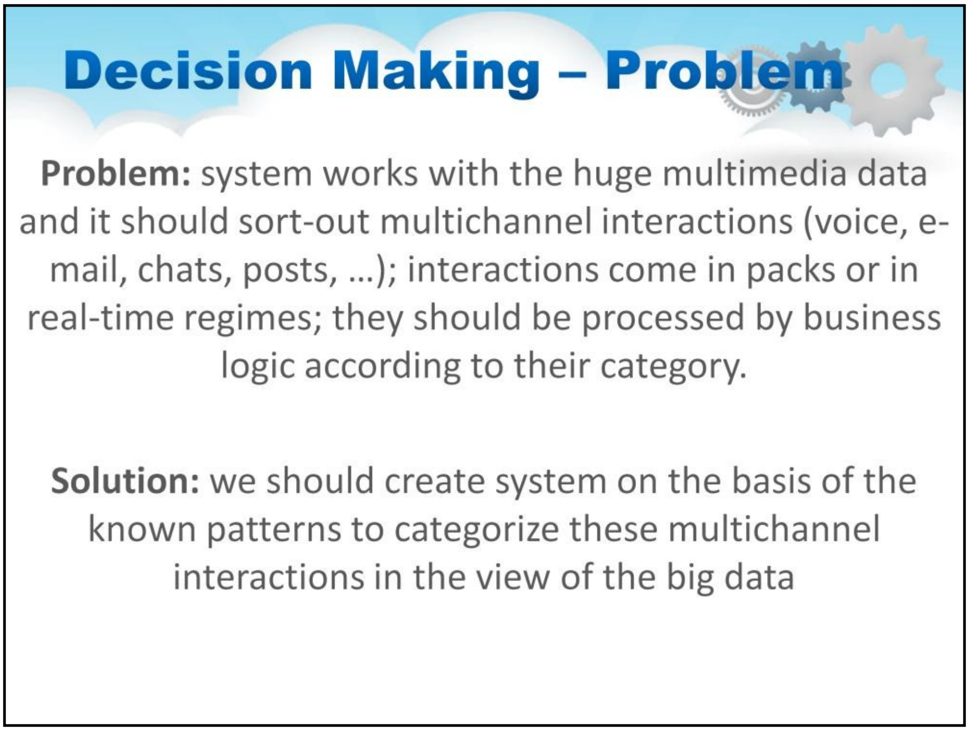
–

Decision Making

on the basis of Big Data

Decision Making

Now let’s consider how Big Data processing can help us in the real life…



Decision Making – Problem

Problem: system works with the huge multimedia data and it should sort-out multichannel interactions (voice, e-mail, chats, posts, …); interactions come in packs or in real-time regimes; they should be processed by business logic according to their category.

Solution: we should create system on the basis of the known patterns to categorize these multichannel interactions in the view of the big data

From this problem formulation:

> **Problem:** system works with the huge multimedia data and it should sort-out multichannel interactions (voice, e-mail, chats, posts, …); interactions come in packs or in real-time regimes; they should be processed by business logic according to their category.

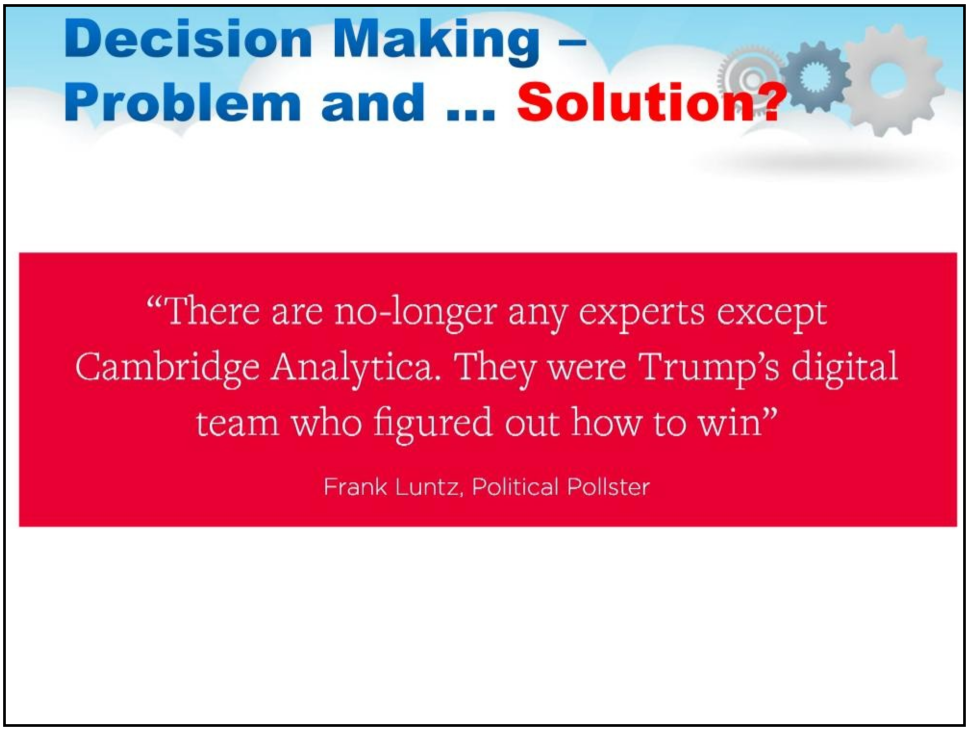
we should find some solution which can be formalized as follows:

> **Solution:** we should create system on the basis of the known patterns to categorize these multichannel interactions in the view of the big data

In this connection, **machine learning** can be used, which is a field of computer science that uses statistical techniques to give computer systems the ability to "learn“ with data, without being explicitly programmed.

The ‘machine learning’ term was proposed by Arthur Samuel in 1959.
It started from the study of pattern recognition and computational learning theory in artificial intelligence.

Now it relates to research on algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.



Decision Making – Problem and ... Solution?

> “There are no-longer any experts except Cambridge Analytica. They were Trump’s digital team who figured out how to win”
>
> Frank Luntz, Political Pollster

Let’s consider the latest example of Cambridge Analytica involvement in Big data processing during elections in USA.

Cambridge Analytica Ltd (CA) was a British political consulting firm which combined data mining, data brokerage, and data analysis (machine learning) with strategic communication during the electoral processes.

The personal data of approximately 87 million Facebook users were acquired via the 270,000 Facebook users who used a Facebook app called "This Is Your Digital Life”.

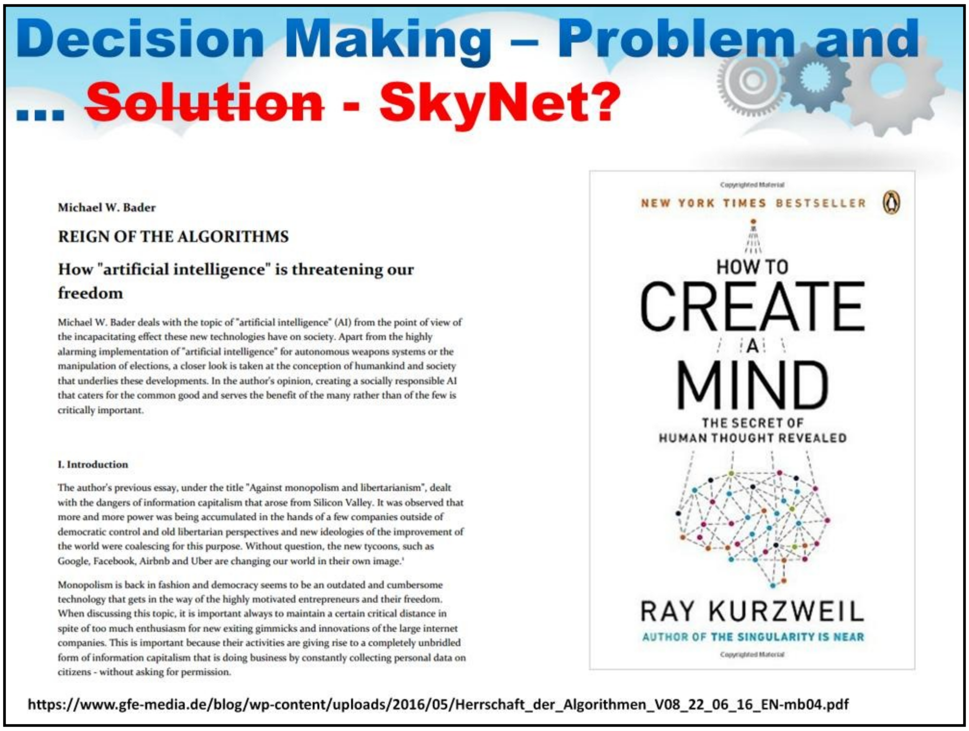
Decision Making – Problem and ... Solution?

http://www.spiegel.de

They developed a profiling system using general online data, Facebook-likes, and smartphone data. They showed that with a limited number of "likes", people can be analyzed better than friends or relatives can do and that individual psychological targeting is a powerful tool to influence people.

For each political client, CA could narrow voter segments from 32 different personality styles it attributes to every adult in the United States. The personality data would inform the tone of the language used in ad messages or voter contact scripts, while additional data is used to determine voters' stances on particular issues.

CA worked for Donald Trump's presidential campaign, and CA had influenced Trump's 2016 US presidential campaign - actually helps Trump to win to surprise of many political scientists.



Decision Making – Problem and ... ~~Solution~~ - SkyNet?

Michael W. Bader

REIGN OF THE ALGORITHMS

How "artificial intelligence" is threatening our freedom

Michael W. Bader deals with the topic of "artificial intelligence" (AI) from the point of view of the incapacitating effect these new technologies have on society. Apart from the highly alarming implementation of "artificial intelligence" for autonomous weapons systems or the manipulation of elections, a closer look is taken at the conception of humankind and society that underlies these developments. In the author's opinion, creating a socially responsible AI that caters for the common good and serves the benefit of the many rather than of the few is critically important.

I. Introduction

The author's previous essay, under the title "Against monopolism and libertarianism", dealt with the dangers of information capitalism that arose from Silicon Valley. It was observed that more and more power was being accumulated in the hands of a few companies outside of democratic control and old libertarian perspectives and new ideologies of the improvement of the world were coalescing for this purpose. Without question, the new tycoons, such as Google, Facebook, Airbnb and Uber are changing our world in their own image.'

Monopolism is back in fashion and democracy seems to be an outdated and cumbersome technology that gets in the way of the highly motivated entrepreneurs and their freedom. When discussing this topic, it is important always to maintain a certain critical distance in spite of too much enthusiasm for new exiting gimmicks and innovations of the large internet companies. This is important because their activities are giving rise to a completely unbridled form of information capitalism that is doing business by constantly collecting personal data on citizens - without asking for permission.

https://www.gfe-media.de/blog/wp-content/uploads/2016/05/Herrschaft_der_Algorithmen_V08_22_06_16_EN-mb04.pdf

In 2012 the inventor and futurist Ray Kurzweil (he was mentioned in the beginning of this

lecture) published his book “How to Create a Mind: The Secret of Human Thought Revealed” about brains, both human and artificial. It became a New York Times Best Seller.[

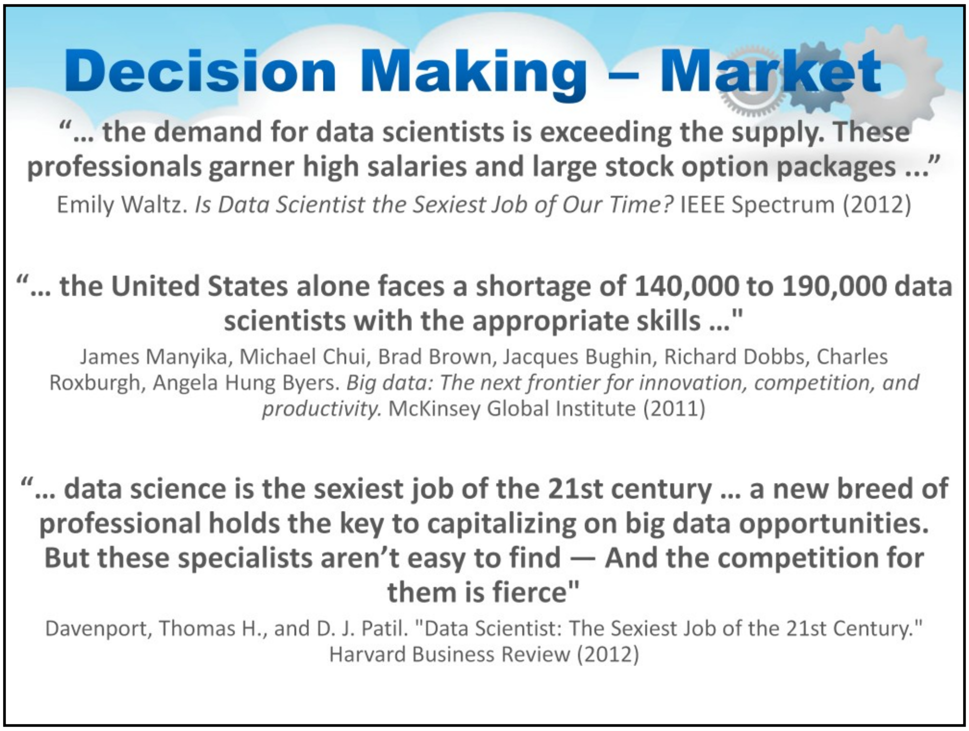
Kurzweil suggests that the brain contains a hierarchy of pattern recognizers and the brain is a "recursive probabilistic fractal" whose line of code is represented within the 30-100 million bytes of compressed code in the genome.

Kurzweil then explains that a computer version of this design could be used to create an artificial intelligence more capable than the human brain. It would employ techniques such as hidden Markov models and genetic algorithms, strategies Kurzweil used successfully in his years as a commercial developer of speech recognition software.

Artificial brains will require massive computational power (which Cloud Computing can provide at the moment), so Kurzweil reviews his law of accelerating returns which explains how the compounding effects of exponential growth will deliver the necessary hardware in only a few decades.

Remember SkyNet from Terminator movie? This fictional artificial intelligence (AI) system features centrally in the movie and serves as the true main antagonist.

Maybe we are facing the pre-conditions of real Skynet as the possible threat that a sufficiently advanced AI could pose to humanity. Elon Musk has previously mentioned Skynet when referring to such a threat.



Decision Making – Market

"… the demand for data scientists is exceeding the supply. These professionals garner high salaries and large stock option packages ..."

Emily Waltz. *Is Data Scientist the Sexiest Job of Our Time?* IEEE Spectrum (2012)

"… the United States alone faces a shortage of 140,000 to 190,000 data scientists with the appropriate skills …"

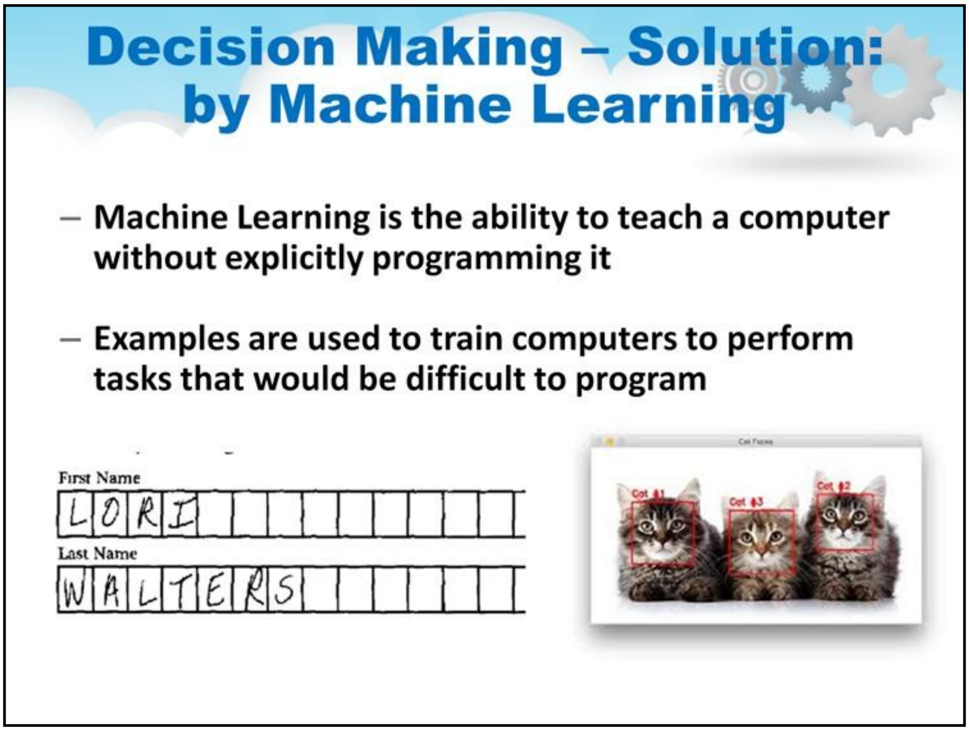
James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers. *Big data: The next frontier for innovation, competition, and productivity.* McKinsey Global Institute (2011)

"… data science is the sexiest job of the 21st century … a new breed of professional holds the key to capitalizing on big data opportunities. But these specialists aren't easy to find — And the competition for them is fierce"

Davenport, Thomas H., and D. J. Patil. "Data Scientist: The Sexiest Job of the 21st Century." Harvard Business Review (2012)

The potential market for such Decision Making systems is HUGE!

Let's consider shortly their machine learning basics on the next slides…



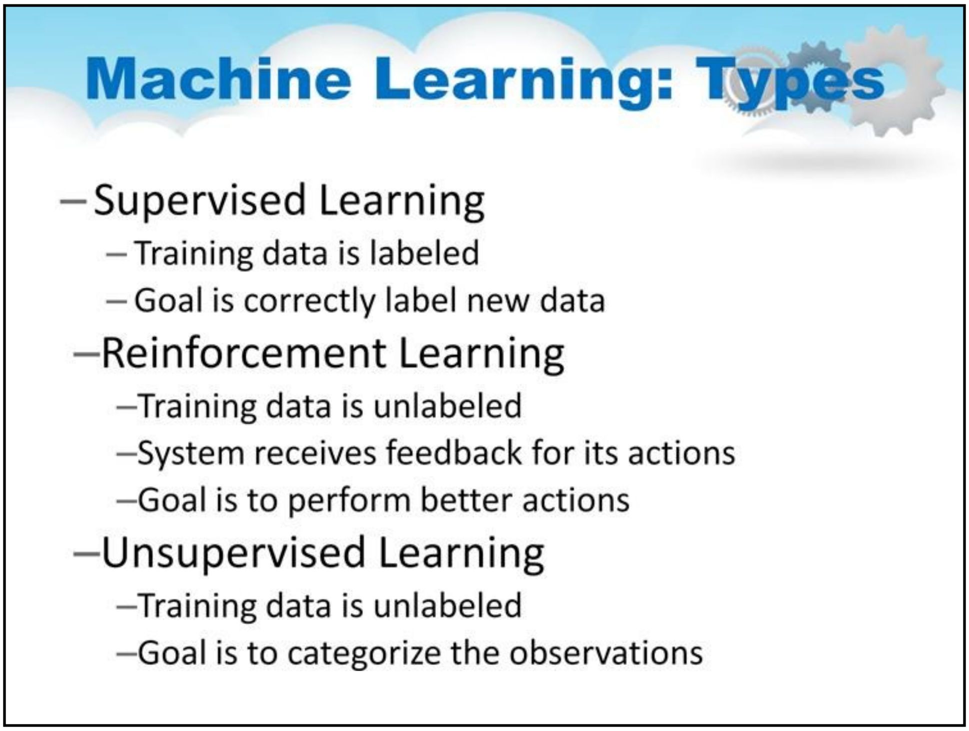
Decision Making – Solution: by Machine Learning

- Machine Learning is the ability to teach a computer without explicitly programming it
- Examples are used to train computers to perform tasks that would be difficult to program

First Name
LORI

Last Name
WALTERS

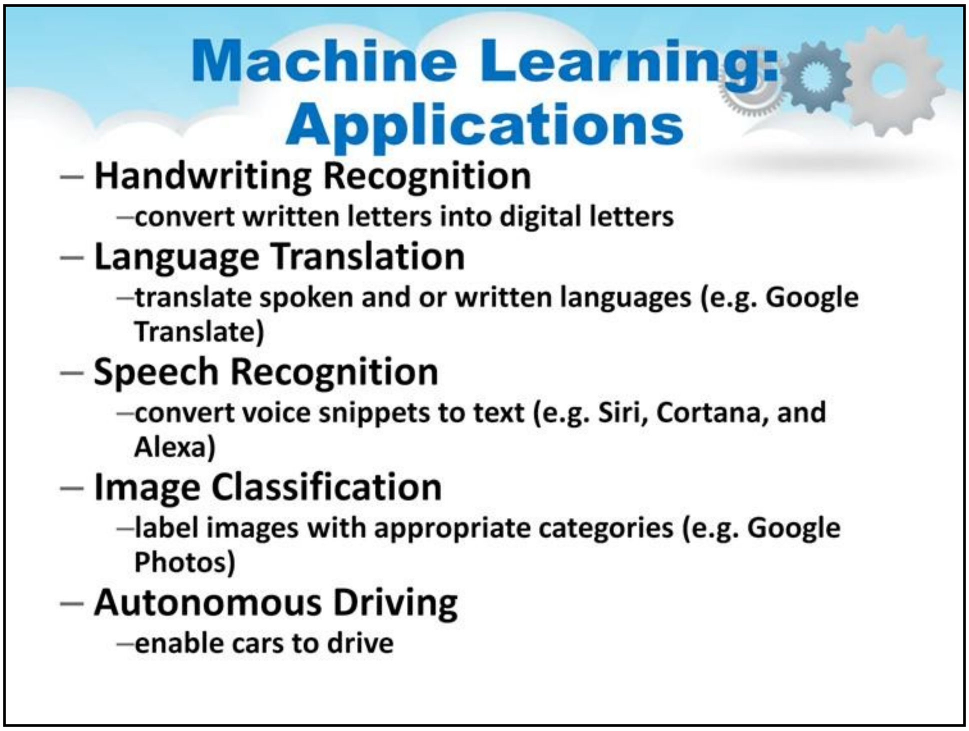
- Machine Learning relies on large quantities of data to train computers (OCR and detecting objects in images)
- Computers can learn complex tasks that would require immense effort to program
- Machine Learning is limited by the availability of examples and computational resources
- With sufficient examples and computational power, computers can learn nearly any task
- Models are the core component Machine Learning
- A model is a method for using known examples to predict or produce inferences on new data.
- Robots are exposed to noisy and dynamic environments, so programming a robot to handle all situations is infeasible
- Therefore, engineers often use Machine Learning to teach robots to perform tasks, much like the way human children learn.



Machine Learning: Types

- Supervised Learning
  - Training data is labeled
  - Goal is correctly label new data
- Reinforcement Learning
  - Training data is unlabeled
  - System receives feedback for its actions
  - Goal is to perform better actions
- Unsupervised Learning
  - Training data is unlabeled
  - Goal is to categorize the observations

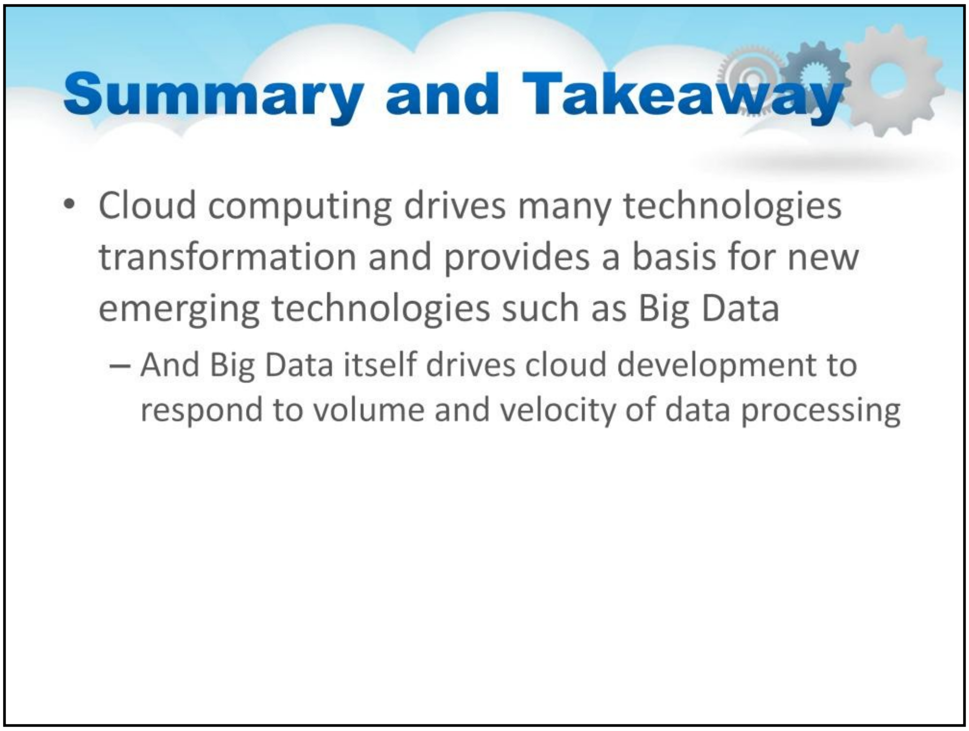
- Supervised Learning is the most common type of machine learning.
- An example of supervised learning is classifying emails as SPAM.
- The training data is emails that are labeled as SPAM or HAM.
- A model is then created that captures the relationship between email contents and the email label.
- The model can then predict the category for new emails.
- Reinforcement learning is commonly used in robotics because there is usually not labeled data
- An example of reinforcement learning is teaching a robot to climb stairs.
- The robot is “rewarded” for each step that it ascends, so it learns which actions are beneficial
- Unsupervised learning is used in data mining to discover insights about unlabeled data
- An example of unsupervised learning is grouping flowers based on their characteristics without knowing the flower species



Machine Learning: Applications

- Handwriting Recognition
  - convert written letters into digital letters
- Language Translation
  - translate spoken and or written languages (e.g. Google Translate)
- Speech Recognition
  - convert voice snippets to text (e.g. Siri, Cortana, and Alexa)
- Image Classification
  - label images with appropriate categories (e.g. Google Photos)
- Autonomous Driving
  - enable cars to drive

- In recent years, Machine Learning has yielded impressive results in diverse disciplines.
- There are many mobile apps available that let users take photos of handwritten characters and convert them to digital text.
- Most translation software now uses machine learning to understand language translation since there is often not a one-to-one correspondence between words in different languages
- Prior to machine learning speech recognition was frustrating and inaccurate, now machine learning enables robust speech recognition on a variety of devices
- Consumers applications like Google Photos and Apple Photos automatically group photos by the people or places in them to make searching simple and intuitive
- It would be impractical to program a car to handle every situation that could occur, but machine learning has enabled cars to self-driving cars to learn from their experiences



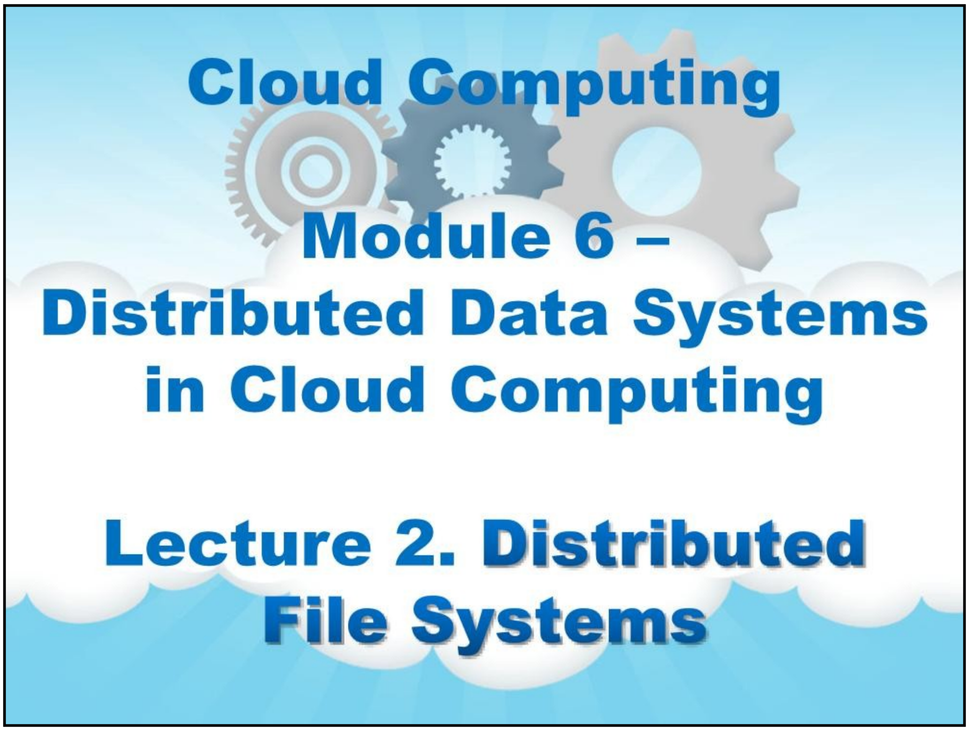
Summary and Takeaway

- Cloud computing drives many technologies transformation and provides a basis for new emerging technologies such as Big Data
  - And Big Data itself drives cloud development to respond to volume and velocity of data processing

In this lecture we covered:

Cloud computing drives many technologies transformation and provides a basis for new emerging technologies such as Big Data

And Big Data itself drives cloud development to respond to volume and velocity of data processing



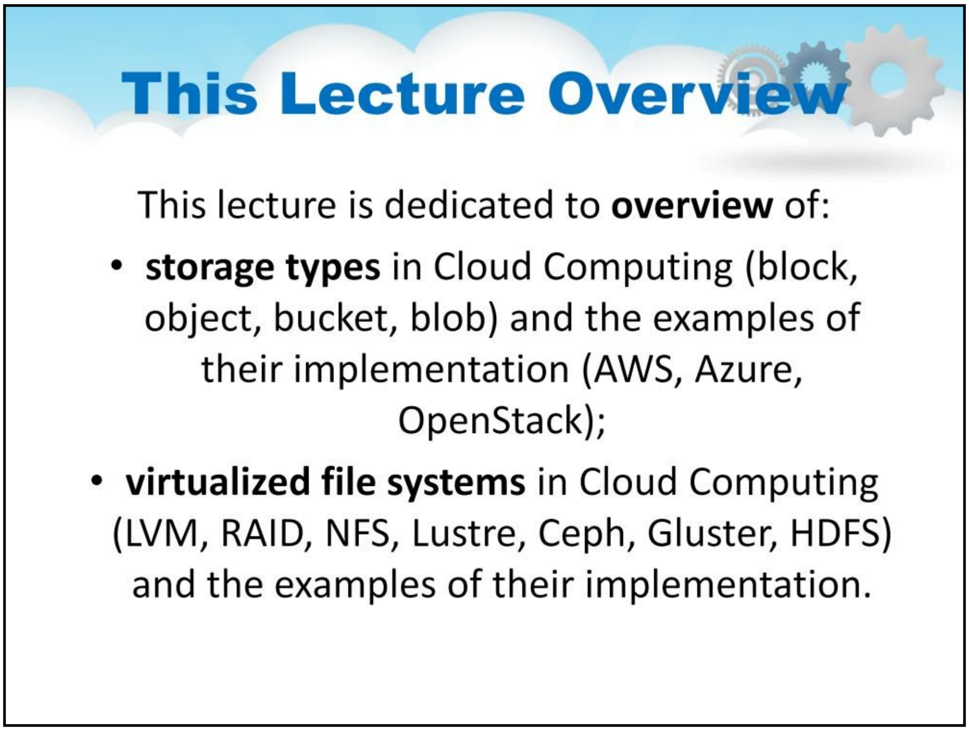
Cloud Computing

Module 6 –
Distributed Data Systems
in Cloud Computing

Lecture 2. Distributed
File Systems

The title of this module is “Distributed data Systems in Cloud Computing”.

This is Lecture 2 about Distributed File Systems.



This Lecture Overview

This lecture is dedicated to **overview** of:

- **storage types** in Cloud Computing (block, object, bucket, blob) and the examples of their implementation (AWS, Azure, OpenStack);
- **virtualized file systems** in Cloud Computing (LVM, RAID, NFS, Lustre, Ceph, Gluster, HDFS) and the examples of their implementation.

Lecture 2. Distributed File Systems

This lecture describes the following main components of Distributed File Systems:

- storage types in Cloud Computing

- block,
- object,
- bucket,
- blob

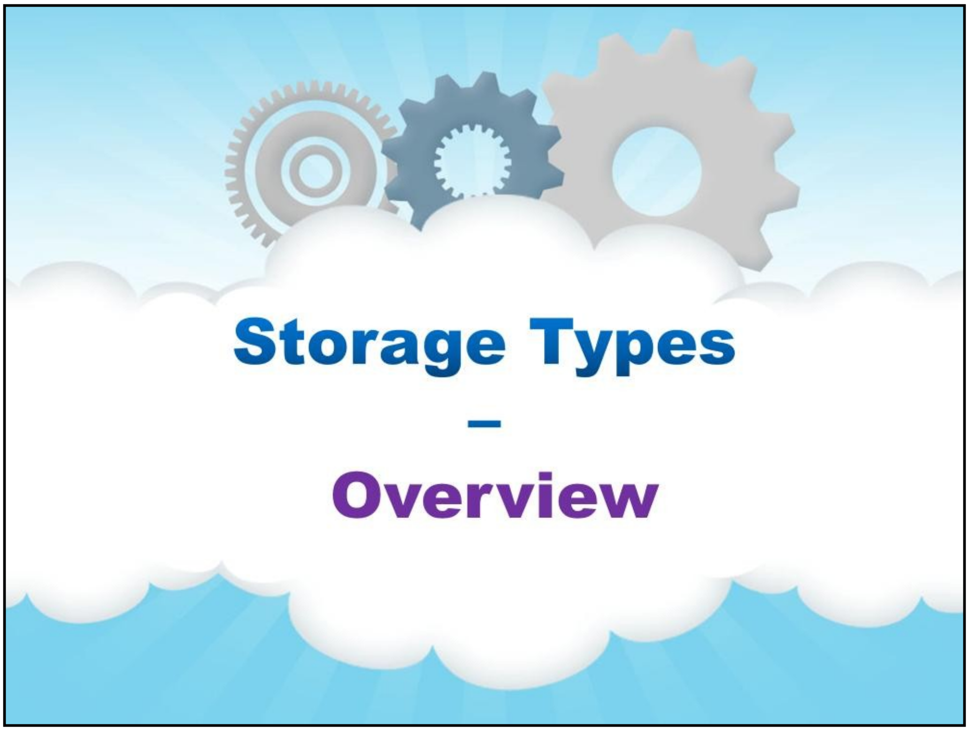
with examples of their implementation:

- AWS,
- Azure,
- OpenStack

- virtualized file systems in Cloud Computing:

- LVM,
- RAID,
- NFS,
- Lustre,
- Ceph,
- Gluster,
- HDFS

with examples of their implementation.



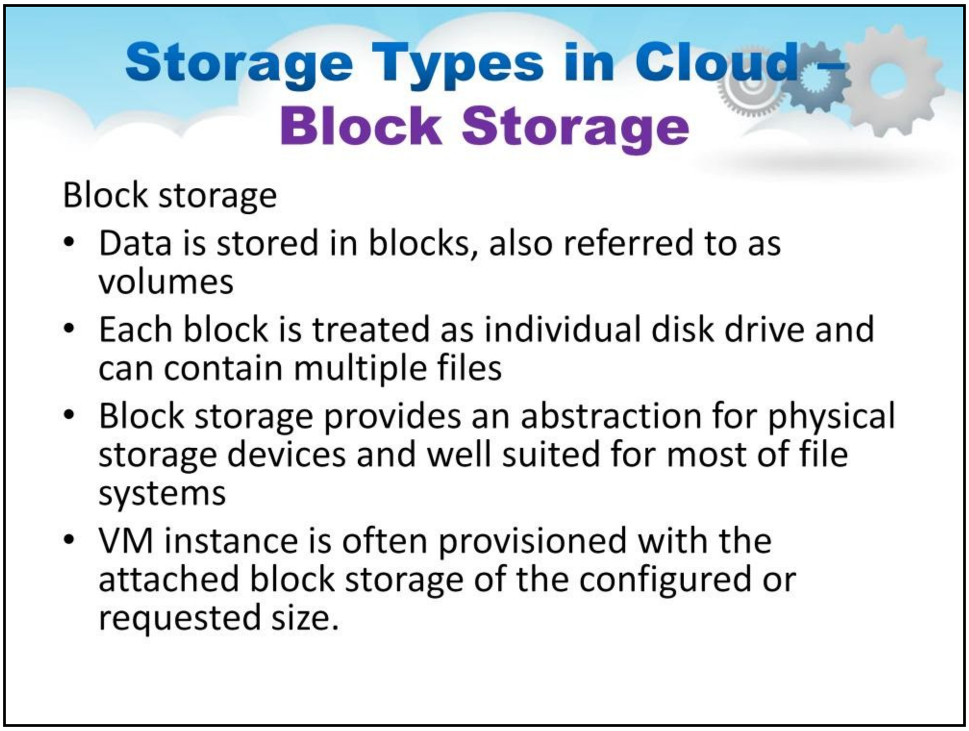
Storage Types

–

Overview

Storage Types

Overview



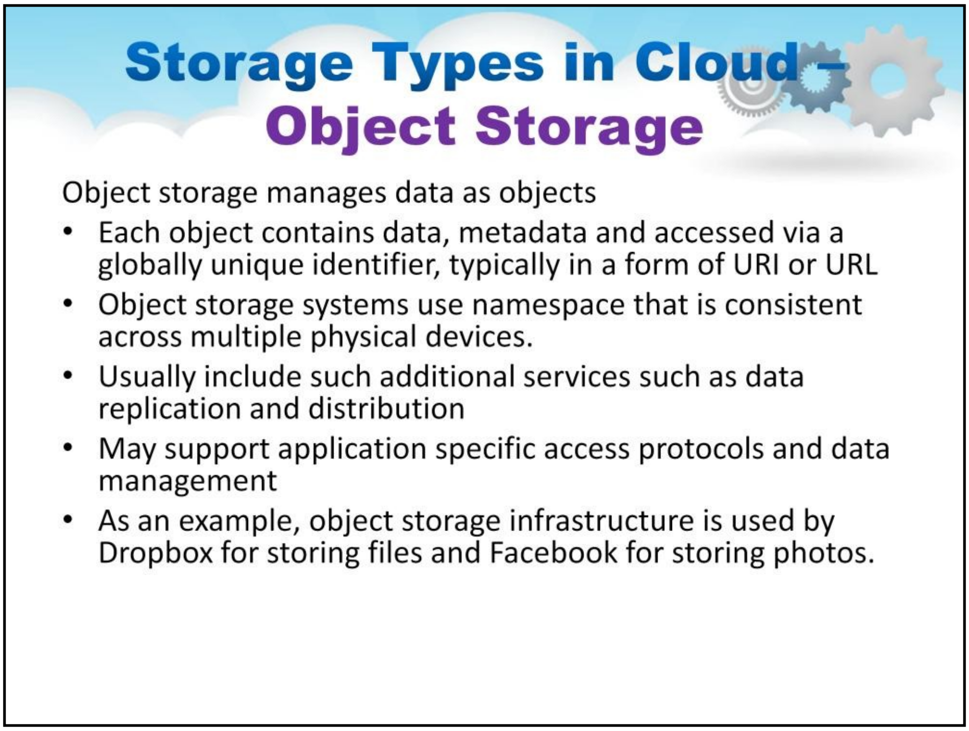
Storage Types in Cloud – Block Storage

Block storage

- Data is stored in blocks, also referred to as volumes
- Each block is treated as individual disk drive and can contain multiple files
- Block storage provides an abstraction for physical storage devices and well suited for most of file systems
- VM instance is often provisioned with the attached block storage of the configured or requested size.

Block storage

Block storage is a type of data storage where data is stored in blocks, also referred to as volumes. Each block is treated as individual disk drive and can contain multiple files. In this way, block storage provides a good abstraction for physical storage devices and well suited for most of file systems. In cloud, VM instance is often provisioned with the attached block storage of the configured or requested size.

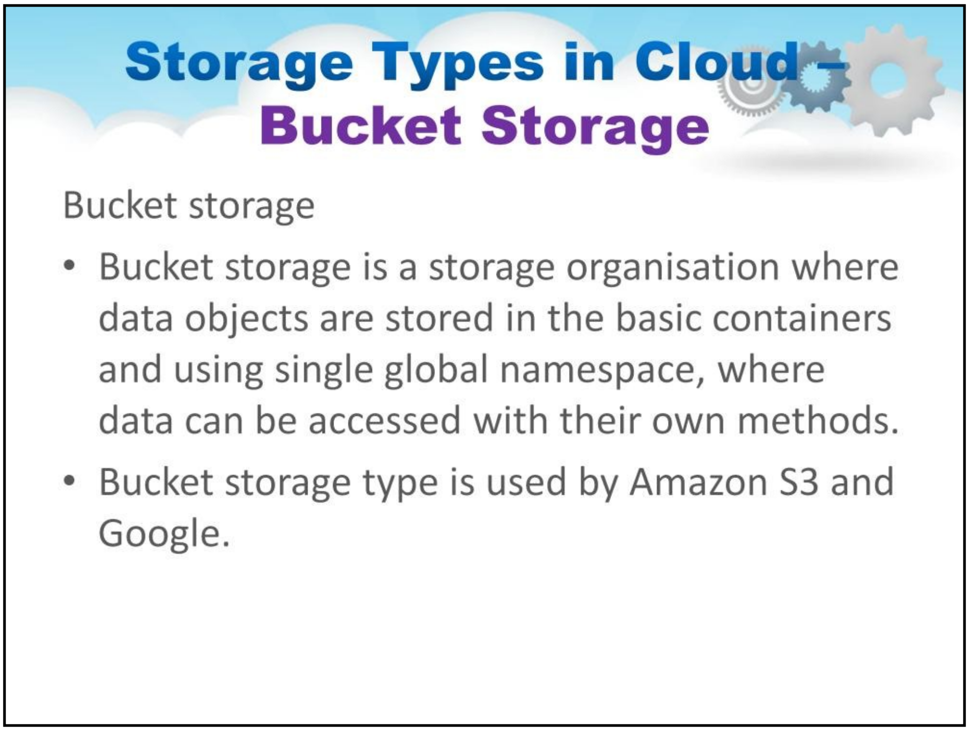


Storage Types in Cloud – Object Storage

Object storage manages data as objects

- Each object contains data, metadata and accessed via a globally unique identifier, typically in a form of URI or URL
- Object storage systems use namespace that is consistent across multiple physical devices.
- Usually include such additional services such as data replication and distribution
- May support application specific access protocols and data management
- As an example, object storage infrastructure is used by Dropbox for storing files and Facebook for storing photos.

Object storage
Storage architecture that manages data as objects. Each object contains data, metadata and accessed via a globally unique identifier, typically in a form of URI or URL. Object storage systems use namespace that is consistent across multiple physical devices. Object storage systems usually include such additional services as data replication and distribution, and may also support application specific access protocols and data management. As an example, object storage infrastructure is used by Dropbox for storing files and Facebook for storing photos.



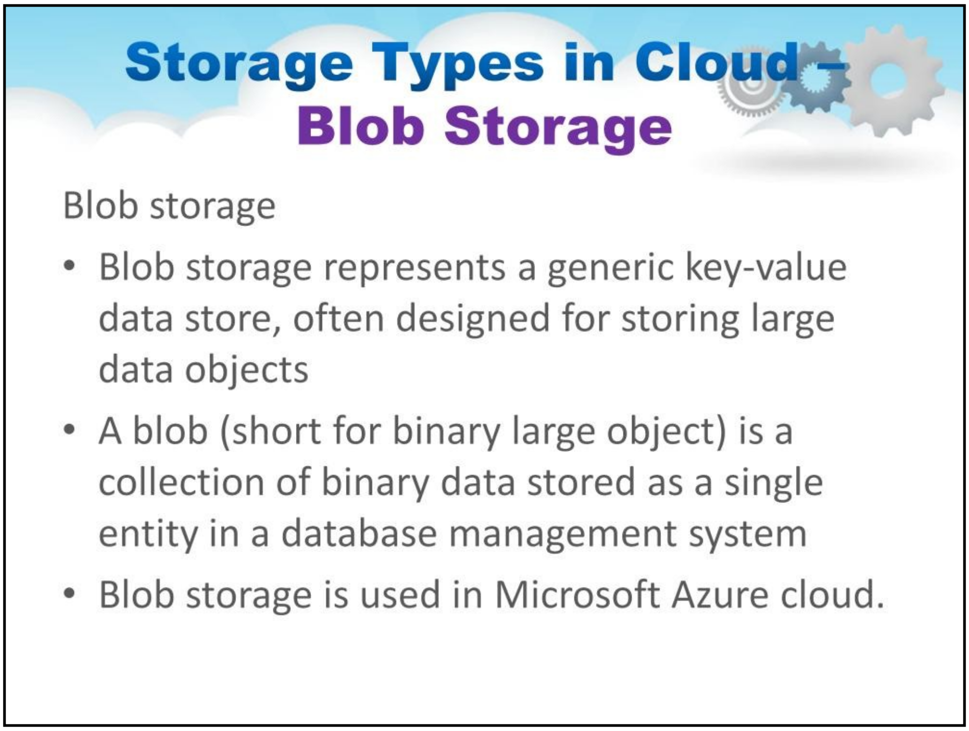
Storage Types in Cloud – Bucket Storage

Bucket storage

- Bucket storage is a storage organisation where data objects are stored in the basic containers and using single global namespace, where data can be accessed with their own methods.
- Bucket storage type is used by Amazon S3 and Google.

Bucket storage

Bucket storage is a storage organization where data objects are stored in the basic containers and using single global namespace, where data can be accessed with their own methods. Bucket storage type is used by Amazon S3 and Google.

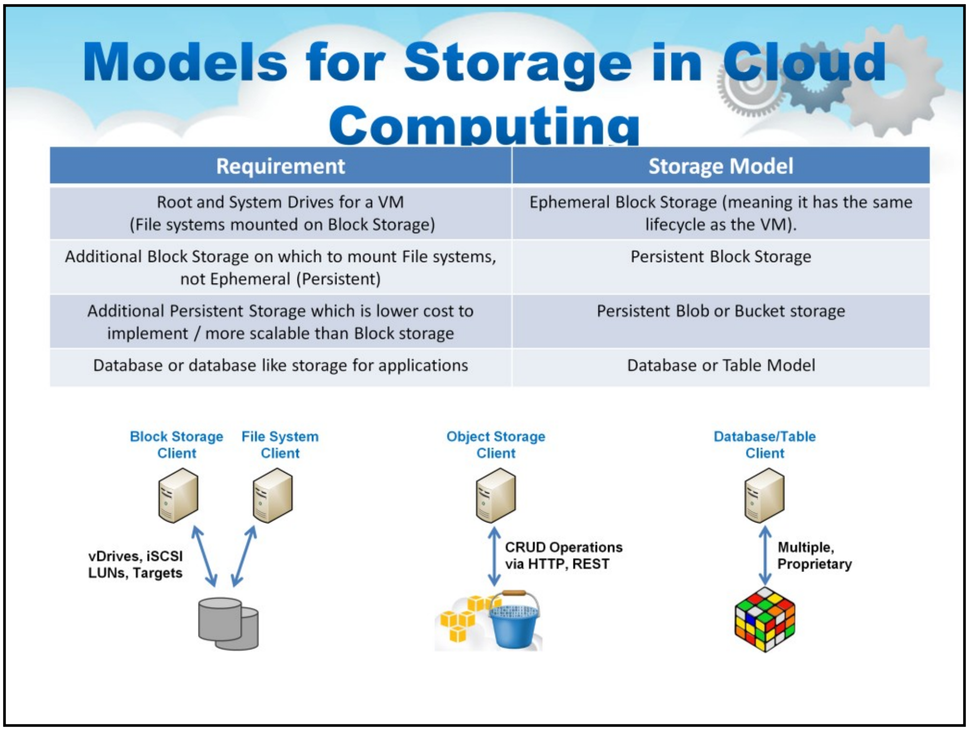


Storage Types in Cloud – Blob Storage

Blob storage

- Blob storage represents a generic key-value data store, often designed for storing large data objects
- A blob (short for binary large object) is a collection of binary data stored as a single entity in a database management system
- Blob storage is used in Microsoft Azure cloud.

Blob storage
Blob storage represents a generic key-value data store, often designed for storing large data objects. A blob (short for binary large object) is a collection of binary data stored as a single entity in a database management system. Blob storage is used in Microsoft Azure cloud.



Models for Storage in Cloud Computing

Requirement	Storage Model
Root and System Drives for a VM (File systems mounted on Block Storage)	Ephemeral Block Storage (meaning it has the same lifecycle as the VM).
Additional Block Storage on which to mount File systems, not Ephemeral (Persistent)	Persistent Block Storage
Additional Persistent Storage which is lower cost to implement / more scalable than Block storage	Persistent Blob or Bucket storage
Database or database like storage for applications	Database or Table Model

Block Storage Client

File System Client

vDrives, iSCSI LUNs, Targets

Object Storage Client

CRUD Operations via HTTP, REST

Database/Table Client

Multiple, Proprietary

Cloud Computing includes several storage models. Some of these come out of necessity, for running existing software. There are also a large number of storage models which have emerged on Cloud which are totally new, becoming possible because of the architecture of Cloud, or becoming needed (where there was not a need before) because of the unprecedently large data volumes found on Cloud, which simple didn't exist before.

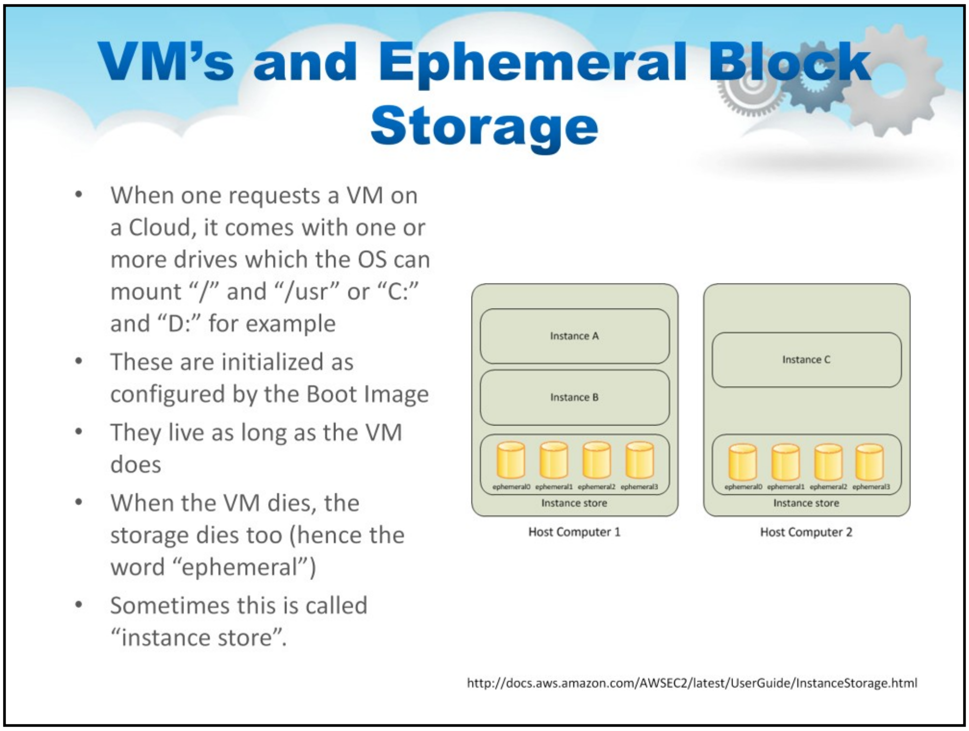
The first type of storage comes from the need for Root and System Drives for a VM. These are File systems mounted on Block Storage. These need to be delivered along with a created VM and conversely don't really need to exist after a VM disappears. Therefore this type of storage is called Ephemeral Block Storage (meaning it has the same lifecycle as the VM).

The second type is Additional Block Storage on which to mount File systems, not Ephemeral (Persistent). This block storage will stick around in the cloud independent of any VMs. Presumably VMs, once created, will mount these drives and have access to them that way. The structure of block storage must be maintained by the actual storage implementation. In general block storage is implemented by simply attaching to existing SAN or NAS storage devices which always provide a block interface. Or it could be implemented in software by the CloudOS as we will see below.

The next type of storage really emerged on the cloud following the model of a tape archive. In the legacy use of tapes, one could have multiple "archives", one after the other, on the tape. The archive had no structure to it, it was a sequence of bits. One could directly place a disk image onto a tape archive for example. Or a "tar" file (which compares to a modern day ZIP file in some ways) could be placed onto a tape archive (in fact, tar stands for Tape ARchive). Tape archives were "buckets of bits" on a sequential magnetic media. On the cloud, we have

this same notion which we call object, or BLOB, or bucket storage. Object storage is implemented on the cloud in clever ways to provide for replication of data for high availability. It is often the lowest cost storage option on a cloud.

Finally, many applications need highly structured storage, like a database, for their applications. While many applications use RDBMs systems with very complicated SQL queries, many application use very simple databases, without complex joins or stored procedures, they need some simple way to do column or table based lookup. As we will see later, implementing databases on a distributed architecture yields some specific challenges and to answer these challenges using different trade-offs, there are many different kinds of database and database-like choices.



VM's and Ephemeral Block Storage

- When one requests a VM on a Cloud, it comes with one or more drives which the OS can mount "/" and "/usr" or "C:" and "D:" for example
- These are initialized as configured by the Boot Image
- They live as long as the VM does
- When the VM dies, the storage dies too (hence the word "ephemeral")
- Sometimes this is called "instance store".

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html

The fist cloud storage model is Ephemeral Block Storage.

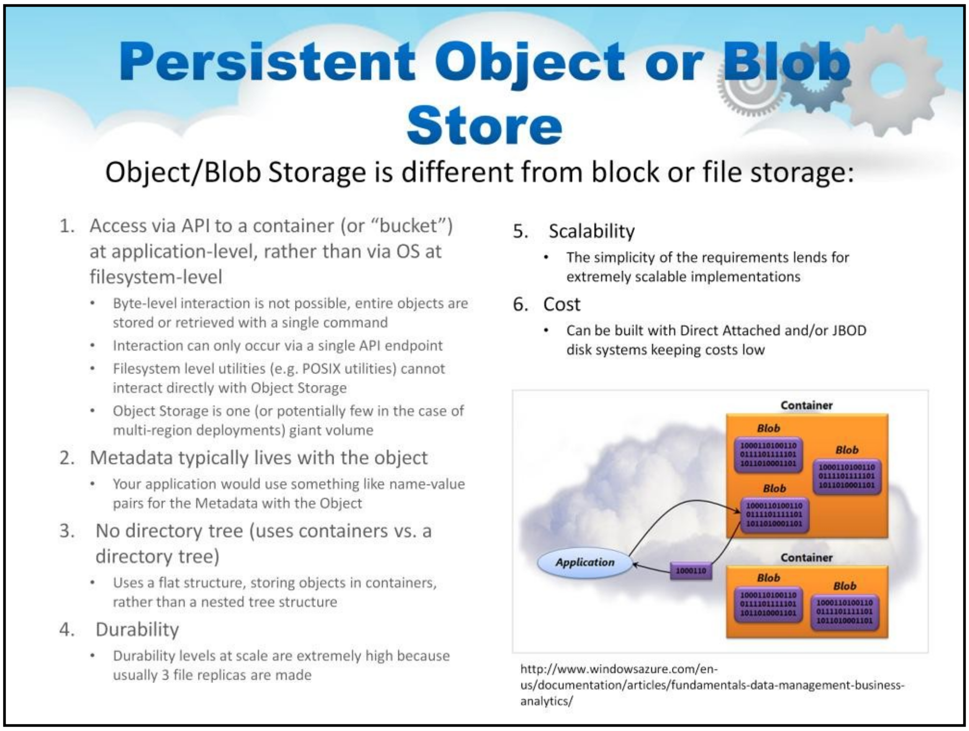
When one requests a VM on a Cloud, it comes with one or more drives which the OS can mount "/" and "/usr" or "C:" and "D:" for example

These are initialized as configured by the Boot Image

They live as long as the VM does

When the VM dies, the storage dies too (hence the word "ephemeral")

Sometimes this is called "instance store".



Persistent Object or Blob Store

Object/Blob Storage is different from block or file storage:

1. Access via API to a container (or "bucket") at application-level, rather than via OS at filesystem-level
   - Byte-level interaction is not possible, entire objects are stored or retrieved with a single command
   - Interaction can only occur via a single API endpoint
   - Filesystem level utilities (e.g. POSIX utilities) cannot interact directly with Object Storage
   - Object Storage is one (or potentially few in the case of multi-region deployments) giant volume
2. Metadata typically lives with the object
   - Your application would use something like name-value pairs for the Metadata with the Object
3. No directory tree (uses containers vs. a directory tree)
   - Uses a flat structure, storing objects in containers, rather than a nested tree structure
4. Durability
   - Durability levels at scale are extremely high because usually 3 file replicas are made
5. Scalability
   - The simplicity of the requirements lends for extremely scalable implementations
6. Cost
   - Can be built with Direct Attached and/or JBOD disk systems keeping costs low

http://www.windowsazure.com/en-us/documentation/articles/fundamentals-data-management-business-analytics/

Persistent Object or Blob Store is different from block or file storage.

Again, think about it like what we use to call a “file” or “archive” on a tape. This is a stream of bit between archive markers. You can stream from the tape or to the tape, you can’t seek around in within an archive.

Access to Object Storage is via API to a container (or “bucket”) at application-level, rather than via OS at filesystem-level

Byte-level interaction is not possible, entire objects are stored or retrieved with a single command

Filesystem level utilities (e.g. POSIX utilities) cannot interact directly with Object Storage

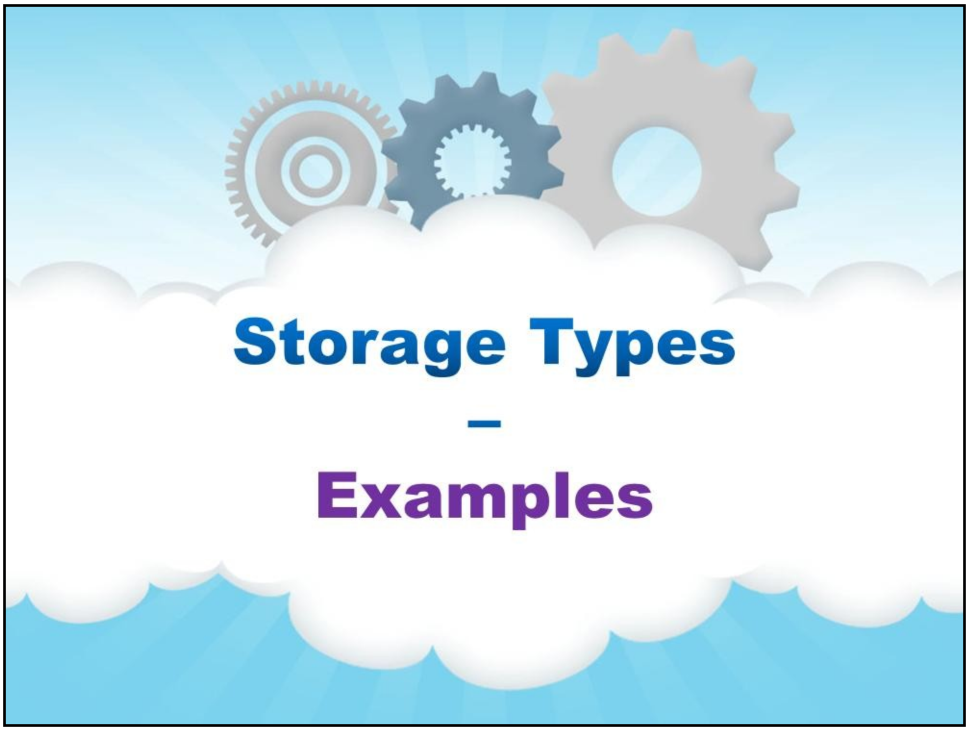
Object Storage is one (or potentially few in the case of multi-region deployments) giant volume, Metadata typically lives with the object

Your application would use something like name-value pairs for the Metadata with the Object

In object storage, there is no structure, no directory tree. It Uses a flat structure, storing objects in containers, rather than a nested tree structure

Durability levels at scale are extremely high because usually 3 file replicas are made

The simplicity of the requirements lends for extremely scalable implementations with low cost drives and pure software implementation, keeping costs low

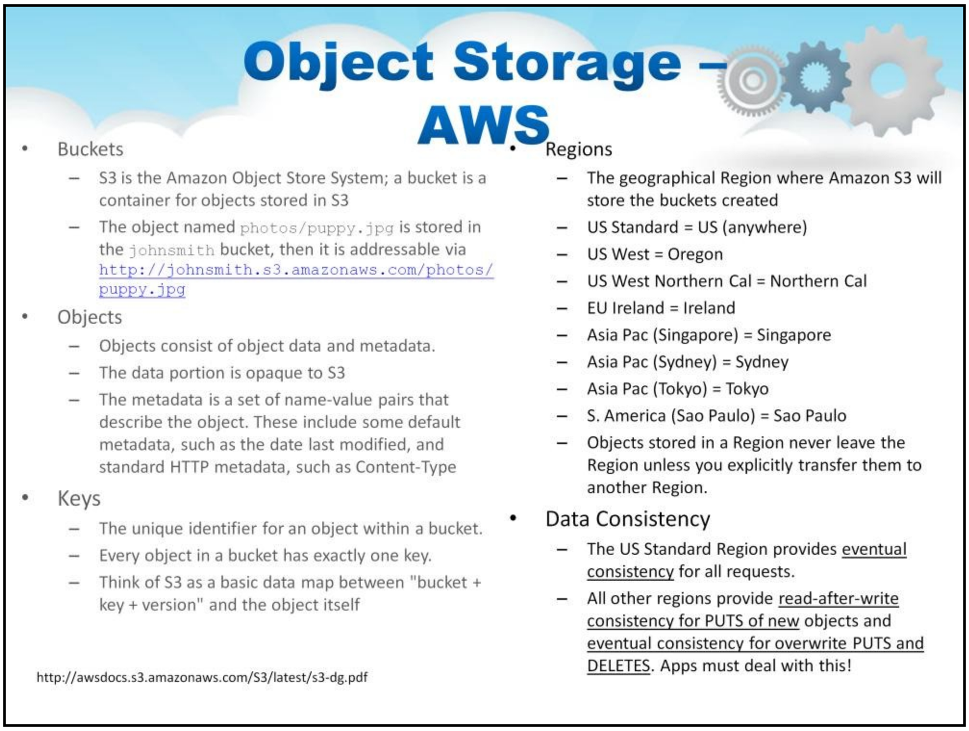


Storage Types

–

Examples

Examples



Object Storage – AWS

- Buckets
  - S3 is the Amazon Object Store System; a bucket is a container for objects stored in S3
  - The object named `photos/puppy.jpg` is stored in the `johnsmith` bucket, then it is addressable via `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`
- Objects
  - Objects consist of object data and metadata.
  - The data portion is opaque to S3
  - The metadata is a set of name-value pairs that describe the object. These include some default metadata, such as the date last modified, and standard HTTP metadata, such as Content-Type
- Keys
  - The unique identifier for an object within a bucket.
  - Every object in a bucket has exactly one key.
  - Think of S3 as a basic data map between "bucket + key + version" and the object itself

http://awsdocs.s3.amazonaws.com/S3/latest/s3-dg.pdf

- Regions
  - The geographical Region where Amazon S3 will store the buckets created
  - US Standard = US (anywhere)
  - US West = Oregon
  - US West Northern Cal = Northern Cal
  - EU Ireland = Ireland
  - Asia Pac (Singapore) = Singapore
  - Asia Pac (Sydney) = Sydney
  - Asia Pac (Tokyo) = Tokyo
  - S. America (Sao Paulo) = Sao Paulo
  - Objects stored in a Region never leave the Region unless you explicitly transfer them to another Region.
- Data Consistency
  - The US Standard Region provides eventual consistency for all requests.
  - All other regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and DELETES. Apps must deal with this!

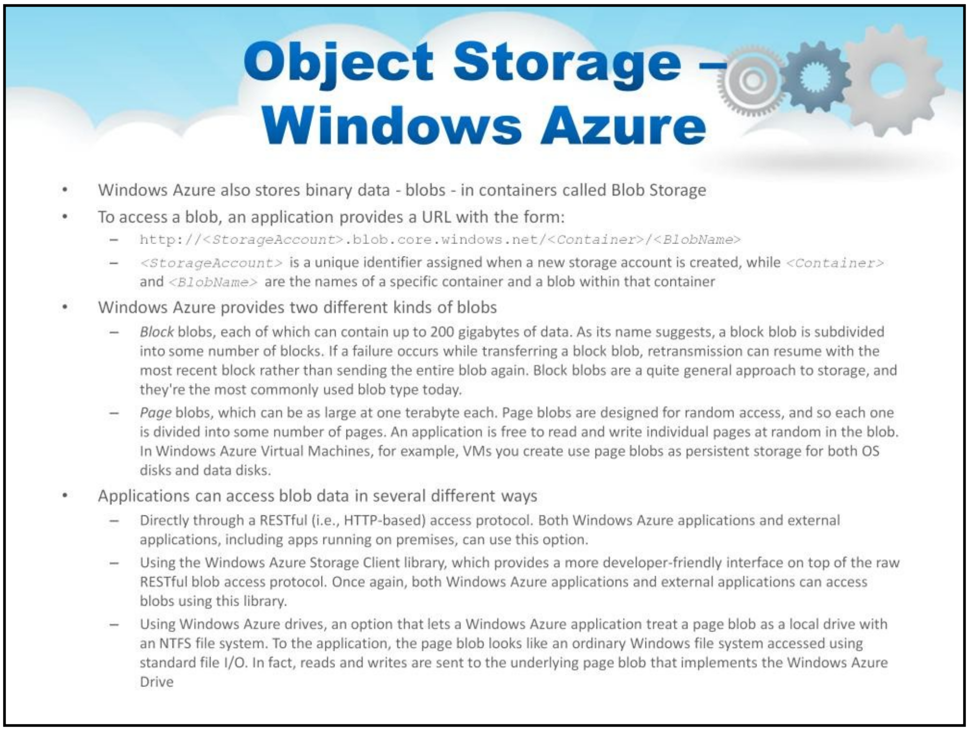
Let's examine the characteristics of the Object Storage implementation in Amazon AWS.

S3 is the name of Amazon’s Object Store System; a bucket is a container for objects stored in S3. The object named photos/puppy.jpg is stored in the johnsmith bucket, then it is addressable via http://johnsmith.s3.amazonaws.com/photos/puppy.jpg

Objects consist of object data and metadata. The data portion is opaque to S3
The metadata is a set of name-value pairs that describe the object. These include some default metadata, such as the date last modified, and standard HTTP metadata, such as Content-Type

Keys are the unique identifier for an object within a bucket.
Every object in a bucket has exactly one key.
Think of S3 as a basic data map between "bucket + key + version" and the object itself

With object storage, one has to know what they are doing! Different Regions of AWS behave differently with Object Storage, as the slide details.



Object Storage – Windows Azure

- Windows Azure also stores binary data - blobs - in containers called Blob Storage
- To access a blob, an application provides a URL with the form:
  - `http://<StorageAccount>.blob.core.windows.net/<Container>/<BlobName>`
  - `<StorageAccount>` is a unique identifier assigned when a new storage account is created, while `<Container>` and `<BlobName>` are the names of a specific container and a blob within that container
- Windows Azure provides two different kinds of blobs
  - *Block* blobs, each of which can contain up to 200 gigabytes of data. As its name suggests, a block blob is subdivided into some number of blocks. If a failure occurs while transferring a block blob, retransmission can resume with the most recent block rather than sending the entire blob again. Block blobs are a quite general approach to storage, and they're the most commonly used blob type today.
  - *Page* blobs, which can be as large at one terabyte each. Page blobs are designed for random access, and so each one is divided into some number of pages. An application is free to read and write individual pages at random in the blob. In Windows Azure Virtual Machines, for example, VMs you create use page blobs as persistent storage for both OS disks and data disks.
- Applications can access blob data in several different ways
  - Directly through a RESTful (i.e., HTTP-based) access protocol. Both Windows Azure applications and external applications, including apps running on premises, can use this option.
  - Using the Windows Azure Storage Client library, which provides a more developer-friendly interface on top of the raw RESTful blob access protocol. Once again, both Windows Azure applications and external applications can access blobs using this library.
  - Using Windows Azure drives, an option that lets a Windows Azure application treat a page blob as a local drive with an NTFS file system. To the application, the page blob looks like an ordinary Windows file system accessed using standard file I/O. In fact, reads and writes are sent to the underlying page blob that implements the Windows Azure Drive

Windows Azure also stores binary data - blobs - in containers called Blob Storage

Windows Azure provides two different kinds of blobs

Block blobs, each of which can contain up to 200 gigabytes of data. As its name suggests, a block blob is subdivided into some number of blocks. If a failure occurs while transferring a block blob, retransmission can resume with the most recent block rather than sending the entire blob again. Block blobs are a quite general approach to storage, and they're the most commonly used blob type today.

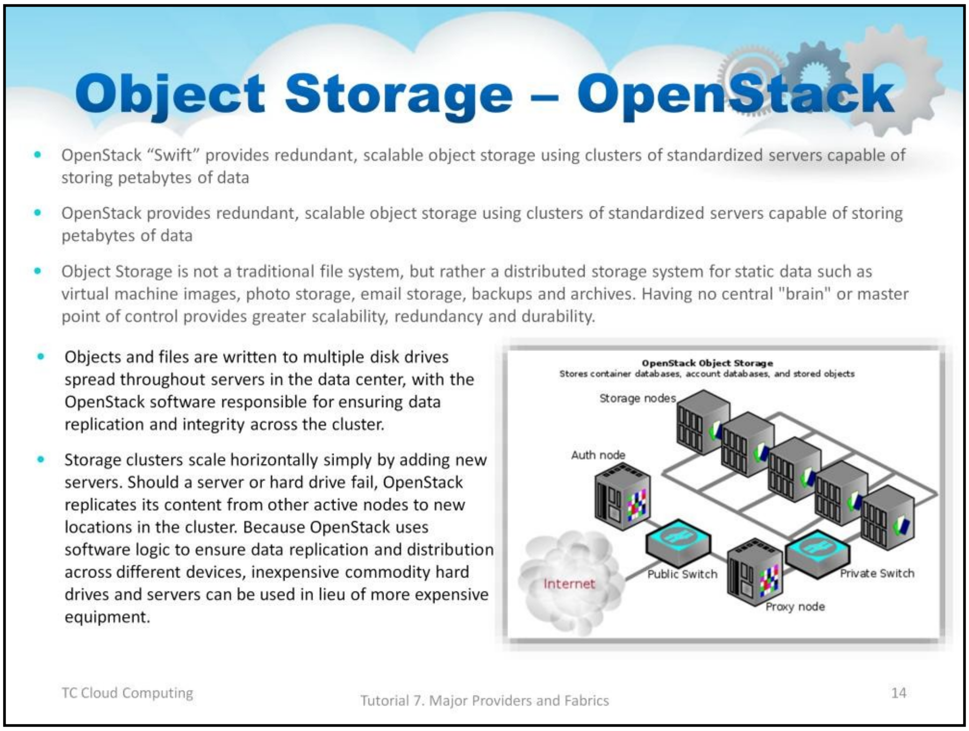
Page blobs, which can be as large at one terabyte each. Page blobs are designed for random access, and so each one is divided into some number of pages. An application is free to read and write individual pages at random in the blob. In Windows Azure Virtual Machines, for example, VMs you create use page blobs as persistent storage for both OS disks and data disks.

Applications can access blob data in several different ways

Directly through a RESTful (i.e., HTTP-based) access protocol. Both Windows Azure applications and external applications, including apps running on premises, can use this option.

Using the Windows Azure Storage Client library, which provides a more developer-friendly interface on top of the raw RESTful blob access protocol. Once again, both Windows Azure applications and external applications can access blobs using this library.

Using Windows Azure drives, an option that lets a Windows Azure application treat a page blob as a local drive with an NTFS file system. To the application, the page blob looks like an ordinary Windows file system accessed using standard file I/O. In fact, reads and writes are sent to the underlying page blob that implements the Windows Azure Drive



Object Storage – OpenStack

- OpenStack “Swift” provides redundant, scalable object storage using clusters of standardized servers capable of storing petabytes of data
- OpenStack provides redundant, scalable object storage using clusters of standardized servers capable of storing petabytes of data
- Object Storage is not a traditional file system, but rather a distributed storage system for static data such as virtual machine images, photo storage, email storage, backups and archives. Having no central "brain" or master point of control provides greater scalability, redundancy and durability.
- Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster.
- Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.

TC Cloud Computing

Tutorial 7. Major Providers and Fabrics

14

The OpenStack Object Store project, known as Swift, offers cloud storage software to store and retrieve lots of data with a simple API. It's built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound.

Swift provides redundant, scalable object storage using clusters of standardized servers capable of storing petabytes of data

Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster.

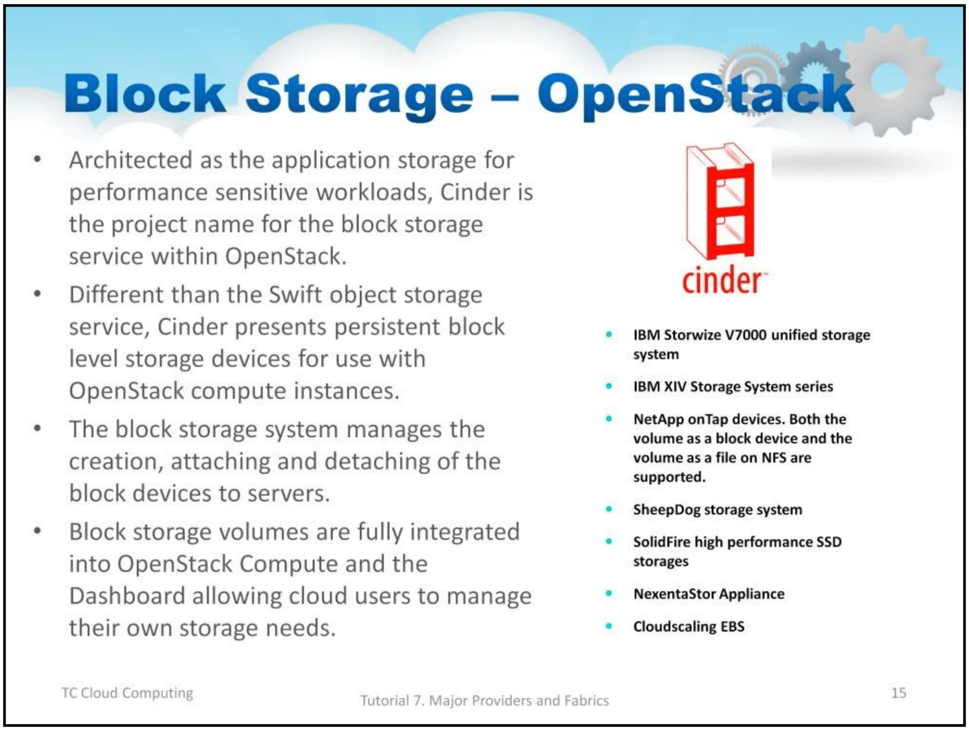
Swift has many features for both end users and sysadmins running the system.

Versioned writes

CORS
ACLs

Arbitrarily large objects
Static website hosting
Signed, expiring URLs
Custom metadata

Bulk operations
Multi-range requests



Block Storage – OpenStack

- Architected as the application storage for performance sensitive workloads, Cinder is the project name for the block storage service within OpenStack.
- Different than the Swift object storage service, Cinder presents persistent block level storage devices for use with OpenStack compute instances.
- The block storage system manages the creation, attaching and detaching of the block devices to servers.
- Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing cloud users to manage their own storage needs.

cinder

- IBM Storwize V7000 unified storage system
- IBM XIV Storage System series
- NetApp onTap devices. Both the volume as a block device and the volume as a file on NFS are supported.
- SheepDog storage system
- SolidFire high performance SSD storages
- NexentaStor Appliance
- Cloudscaling EBS

TC Cloud Computing

Tutorial 7. Major Providers and Fabrics

15

Cinder is a Block Storage service for OpenStack. It's designed to allow the use of either a reference implementation (LVM) to present storage resources to end users that can be consumed by the OpenStack Compute Project (Nova).

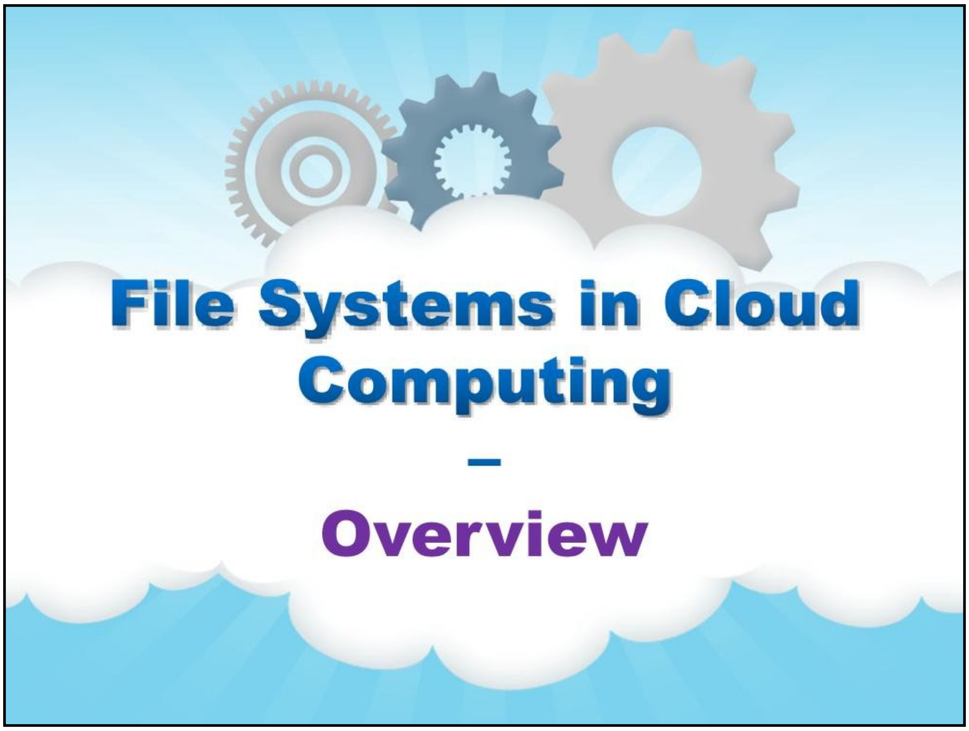
The short description of Cinder is that it virtualizes pools of block storage devices and provides end users with a self service API to request and consume those resources wit

Architected as the application storage for performance sensitive workloads, Cinder is the project name for the block storage service within OpenStack.

Different than the Swift object storage service, Cinder presents persistent block level storage devices for use with OpenStack compute instances.

The block storage system manages the creation, attaching and detaching of the block devices to servers.

Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing cloud users to manage their own storage needs.



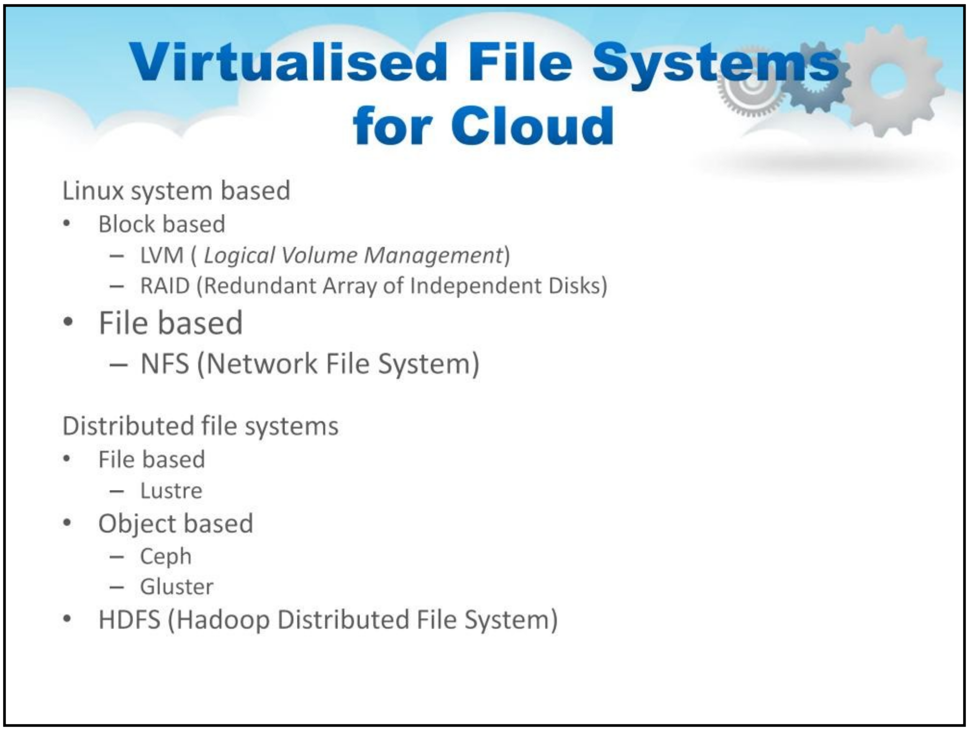
File Systems in Cloud Computing

–

Overview

File Systems in Cloud Computing

Overview



Virtualised File Systems for Cloud

Linux system based

- Block based
  - LVM (*Logical Volume Management*)
  - RAID (Redundant Array of Independent Disks)
- File based
  - NFS (Network File System)

Distributed file systems

- File based
  - Lustre
- Object based
  - Ceph
  - Gluster
- HDFS (Hadoop Distributed File System)

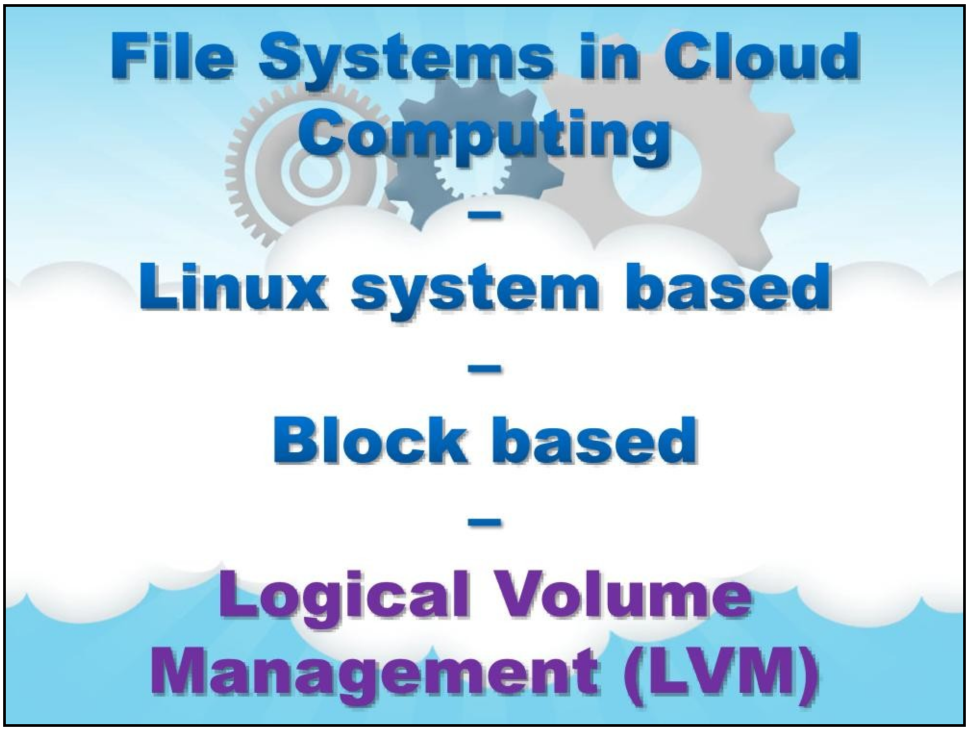
We will look closer at some popular file systems for cloud.
Each of them has a number of benefits when implemented in specific cloud environment.

Most applications need filesystems because servers have filesystems. Most filesystems need block storage upon which to mount. Therefore it is no surprise that filesystems with underlying block storage are popular cloud storage options.

A simple approach is to use server or NAS based technologies extended directly to the cloud for the block filesystems. That is tie a number of drives together, and maybe access across the network. As the slide lists, LVM, RAID, and NFS are all examples of virtualized filesystems on block storage commonly found in smaller clouds.

Larger clouds utilize distributed file systems, which have a high degree of redundancy across the cloud they are serving. As the slide shows, they can be file or object based and there are many examples of both popular in cloud implementations.

HDFS (Hadoop Distributed File System) is specifically designed for large scale data processing on massively parallel clusters. Can be used in cloud for high performance data input/output, in particular for CDN (Content Distribution Network)



File Systems in Cloud Computing

–

Linux system based

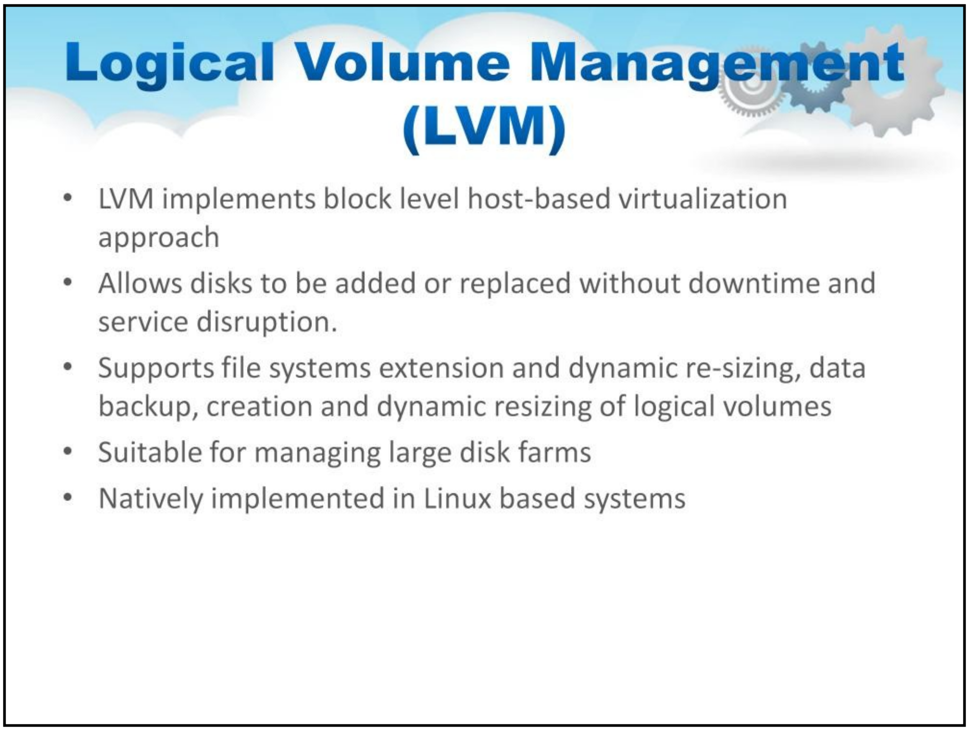
–

Block based

–

Logical Volume Management (LVM)

LVM



Logical Volume Management (LVM)

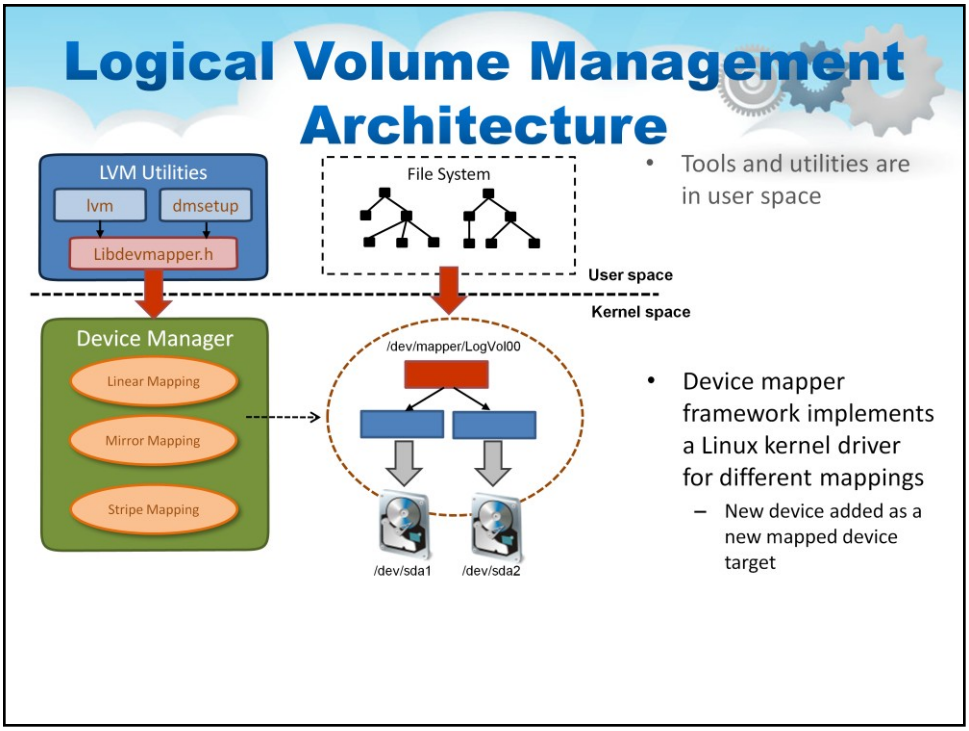
- LVM implements block level host-based virtualization approach
- Allows disks to be added or replaced without downtime and service disruption.
- Supports file systems extension and dynamic re-sizing, data backup, creation and dynamic resizing of logical volumes
- Suitable for managing large disk farms
- Natively implemented in Linux based systems

Logical Volume Manager is very popular because it is commonly found in Linux. It implements block level host-based virtualization approach

Allows disks to be added or replaced without downtime and service disruption.

Supports file systems extension and dynamic re-sizing, data backup, creation and dynamic resizing of logical volumes

Suitable for managing large disk farms



Logical Volume Management Architecture

LVM Utilities

lvm

dmsetup

Libdevmapper.h

File System

User space

Kernel space

Device Manager

Linear Mapping

Mirror Mapping

Stripe Mapping

/dev/mapper/LogVol00

/dev/sda1

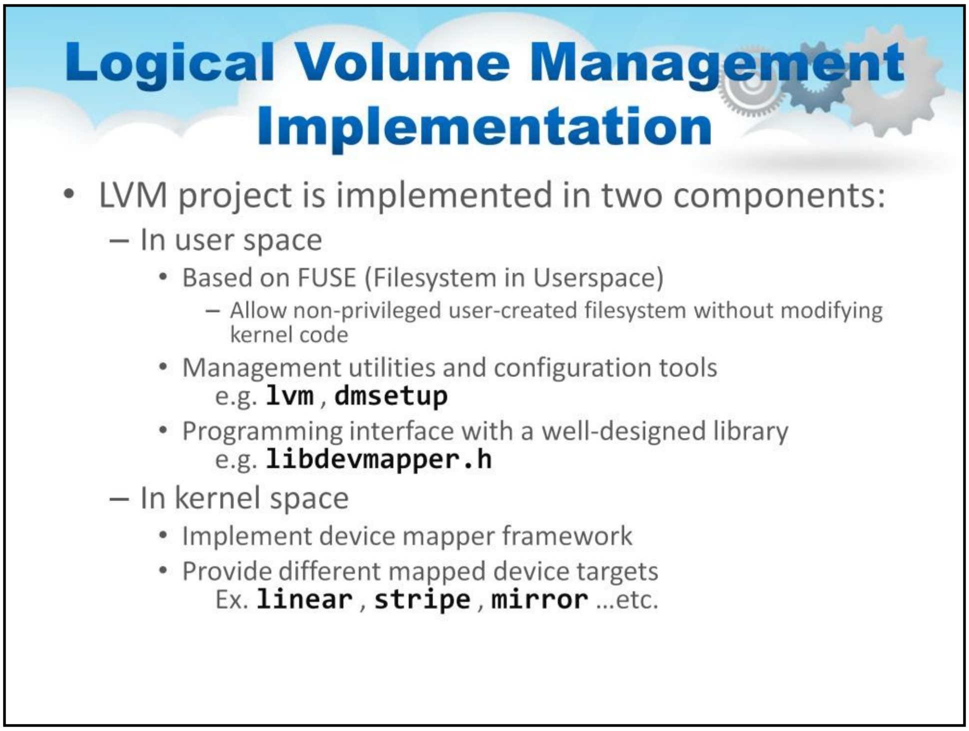
/dev/sda2

- Tools and utilities are in user space
- Device mapper framework implements a Linux kernel driver for different mappings
  - New device added as a new mapped device target

This slide illustrates the Logical Volume Management Architecture

Tools and utilities are in user space

Device mapper framework implements a Linux kernel driver for different mappings



Logical Volume Management Implementation

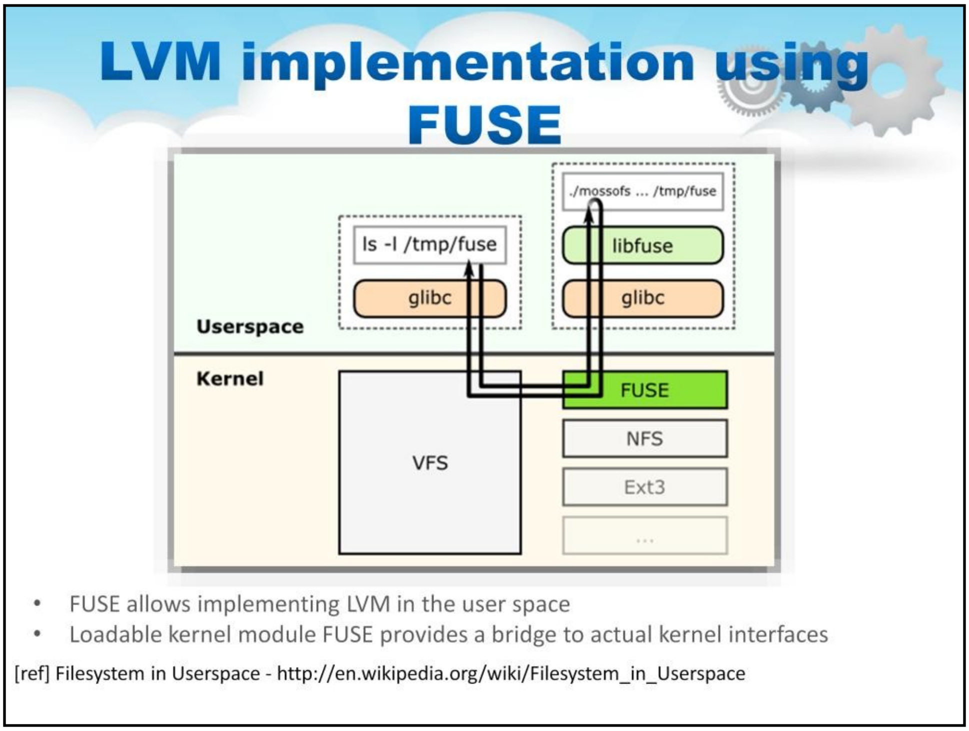
- LVM project is implemented in two components:
  - In user space
    - Based on FUSE (Filesystem in Userspace)
      - Allow non-privileged user-created filesystem without modifying kernel code
    - Management utilities and configuration tools e.g. **`lvm`** , **`dmsetup`**
    - Programming interface with a well-designed library e.g. **`libdevmapper.h`**
  - In kernel space
    - Implement device mapper framework
    - Provide different mapped device targets Ex. **`linear`** , **`stripe`** , **`mirror`** ...etc.

Logical Volume Management Implementation

LVM project is implemented in two components:

In user space Based on FUSE (Filesystem in Userspace)

In kernel space which Implements device mapper framework



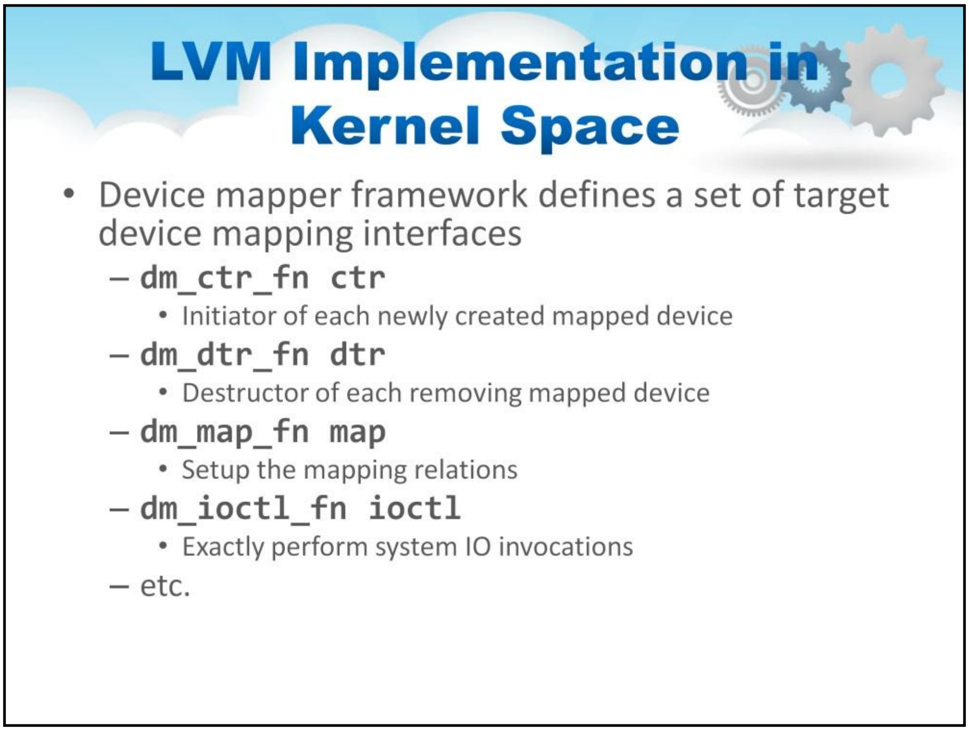
LVM implementation using FUSE

- FUSE allows implementing LVM in the user space
- Loadable kernel module FUSE provides a bridge to actual kernel interfaces

[ref] Filesystem in Userspace - http://en.wikipedia.org/wiki/Filesystem_in_Userspace

This slide diagram the Logical Volume Manager using Filesystem in User Space

As can be seen by the diagram, the key is a that the Loadable kernel module FUSE provides a bridge to actual kernel interfaces

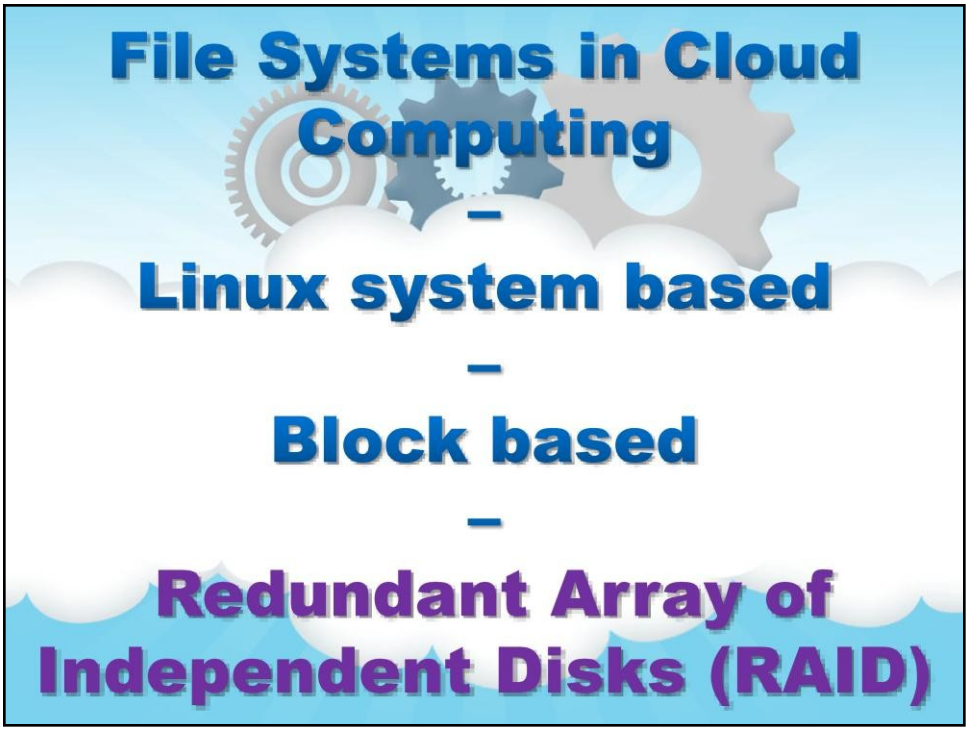


LVM Implementation in Kernel Space

- Device mapper framework defines a set of target device mapping interfaces
  - **`dm_ctr_fn ctr`**
    - Initiator of each newly created mapped device
  - **`dm_dtr_fn dtr`**
    - Destructor of each removing mapped device
  - **`dm_map_fn map`**
    - Setup the mapping relations
  - **`dm_ioctl_fn ioctl`**
    - Exactly perform system IO invocations
  - etc.

For performance reasons, there is an implementation available of LVM in kernel space

This slide speaks to the system calls when implemented this way



File Systems in Cloud Computing

–

Linux system based

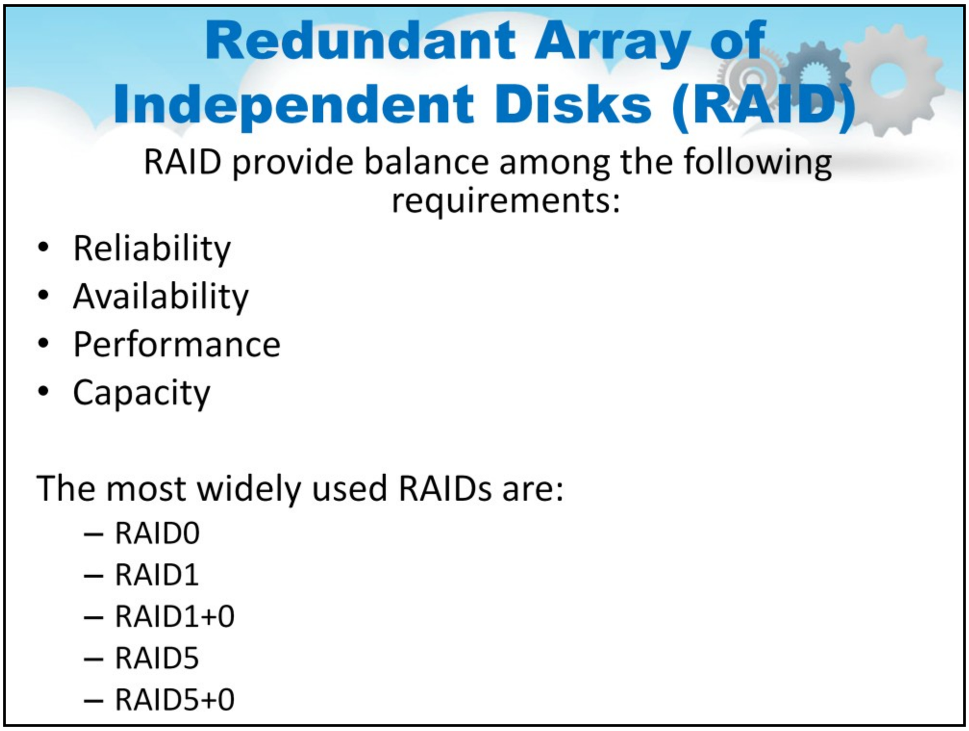
–

Block based

–

Redundant Array of Independent Disks (RAID)

RAID



Redundant Array of Independent Disks (RAID)

RAID provide balance among the following requirements:

- Reliability
- Availability
- Performance
- Capacity

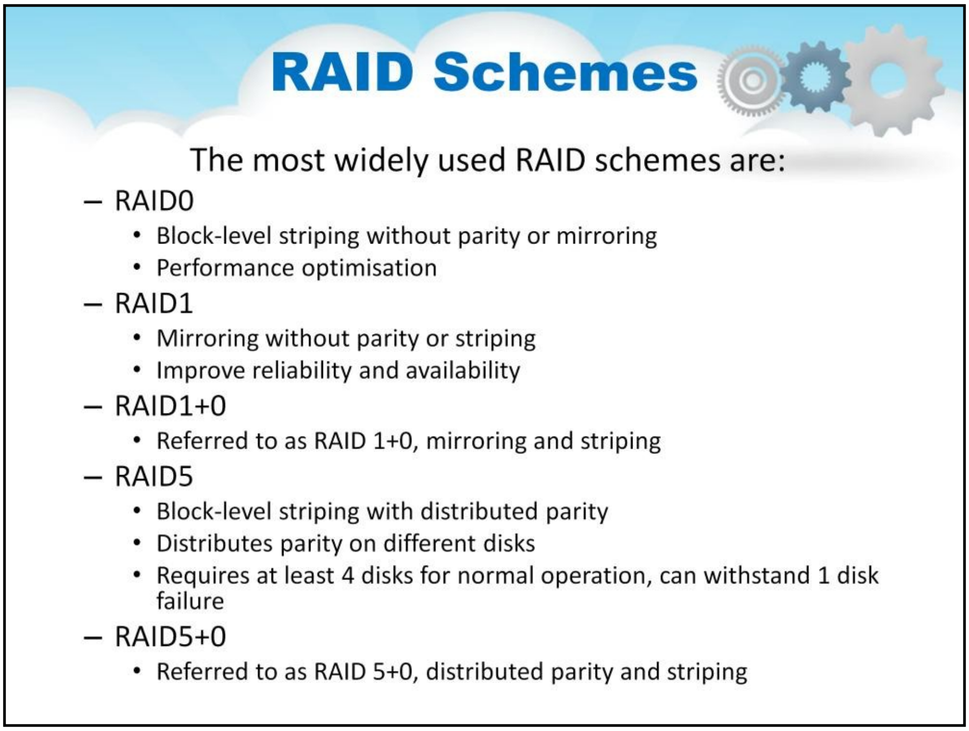
The most widely used RAIDs are:

- RAID0
- RAID1
- RAID1+0
- RAID5
- RAID5+0

Another common scheme for virtualizing storage is RAID (Redundant Array of Independent Disks)

RAID is a software layer which groups together disks and implements various levels of replication and distribution of data across the drives.

RAID schemes provide different balance between the key goals: Reliability, Availability, Performance, Capacity



RAID Schemes

The most widely used RAID schemes are:

- RAID0
  - Block-level striping without parity or mirroring
  - Performance optimisation
- RAID1
  - Mirroring without parity or striping
  - Improve reliability and availability
- RAID1+0
  - Referred to as RAID 1+0, mirroring and striping
- RAID5
  - Block-level striping with distributed parity
  - Distributes parity on different disks
  - Requires at least 4 disks for normal operation, can withstand 1 disk failure
- RAID5+0
  - Referred to as RAID 5+0, distributed parity and striping

The slide describes the differences in common RAID schemes:

RAID0

Block-level striping without parity or mirroring

Performance optimization

RAID1

Mirroring without parity or striping

Improve reliability and availability

RAID1+0

Referred to as RAID 1+0, mirroring and striping

RAID5

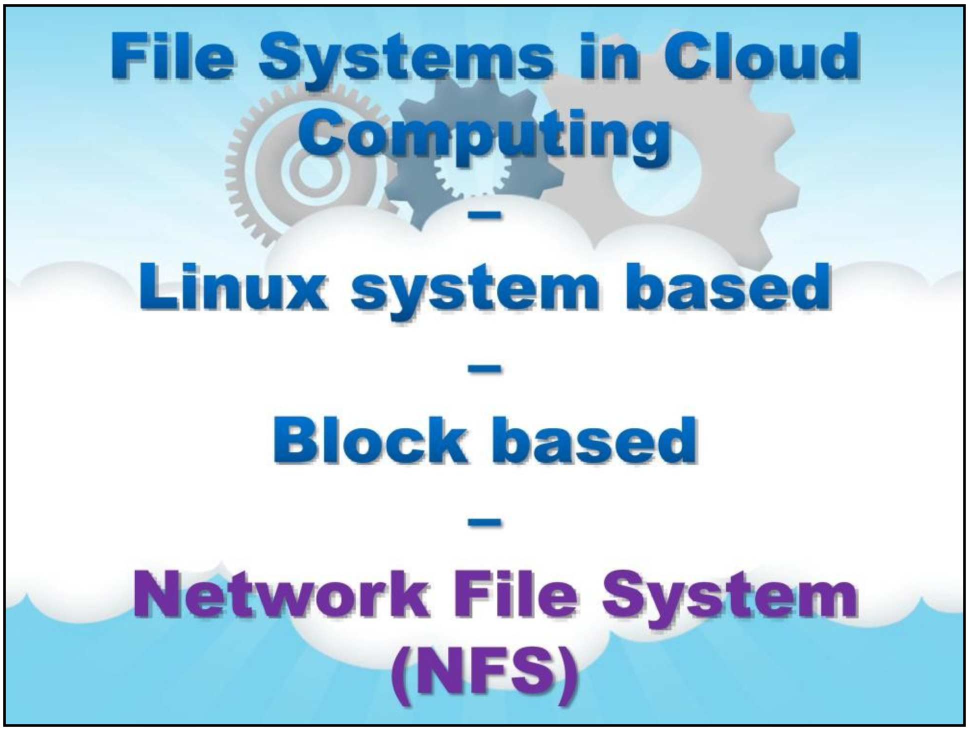
Block-level striping with distributed parity

Distributes parity on different disks

Requires at least 4 disks for normal operation, can withstand 1 disk failure

RAID5+0

Referred to as RAID 5+0, distributed parity and striping



File Systems in Cloud Computing

–

Linux system based

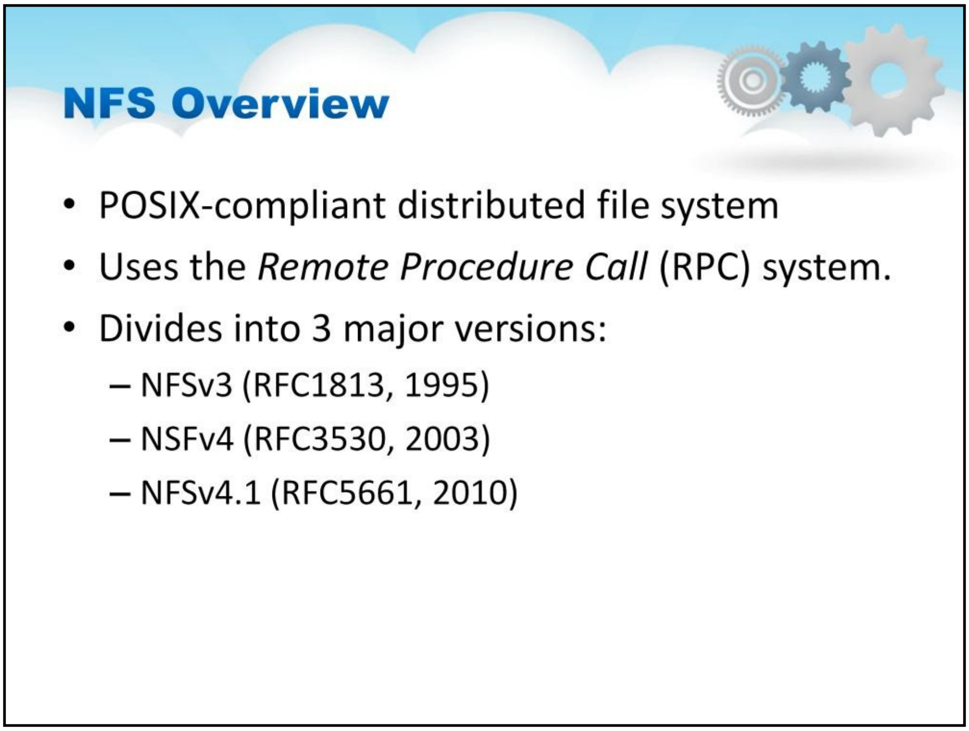
–

Block based

–

Network File System (NFS)

NFS



NFS Overview

- POSIX-compliant distributed file system
- Uses the *Remote Procedure Call* (RPC) system.
- Divides into 3 major versions:
  - NFSv3 (RFC1813, 1995)
  - NSFv4 (RFC3530, 2003)
  - NFSv4.1 (RFC5661, 2010)

The Network File System is an open standard defined in RFCs.

NFS is a common module in Linux kernel.

NFSv3 provides:

Support for 64-bit file sizes and offsets, to handle files larger than 2GB Support TCP protocol and asynchronous writes on the server

NSFv4 and NFSv4.1 provide:

Performance improvements, mandates strong security, and introduces a stateful protocol

Supports clustered server deployments including scalable parallel access to files distributed among multiple servers (pNFS extension)

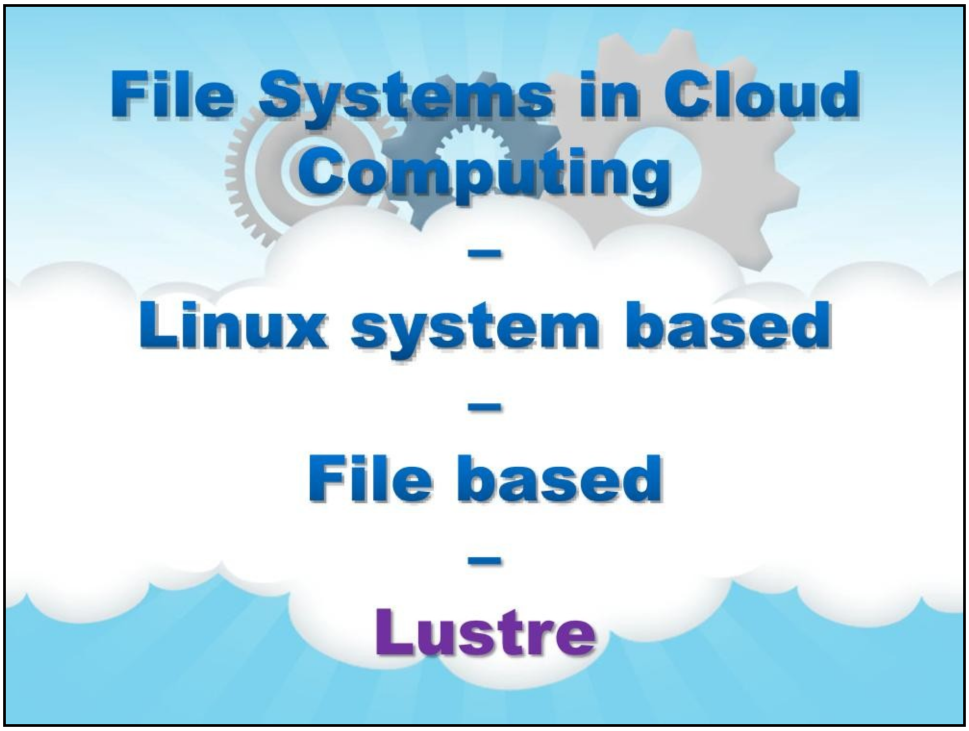
Filesystems have characteristic which application developers have come to depend on. For example, when the “write” call returns from the kernel to the user application, the user

application assumes that the data is actually written, or at least that a subsequent read of the same data will return what was just written. These behavioral assumptions are part of the POSIX specification.

When clusters of disks are used, and when filesystems are exported across the network, it becomes challenging to live up to all of the POSIX filesystem requirements. Network filesystems which had correct behavior became very popular. The SUN Network File System (NFS) was one of the first and most reliable POSIX-compliant distributed file systems.

As the slide lists, NFS is specified by a number of RFCs and the protocols use to implement NFS are well known and understood. Over the years NFS has become a “go to” network filesystem.

Notably, NFS has been extended to support clustered server deployments including scalable parallel access to files distributed among multiple servers (pNFS extension). This technology is a key part of for example the IBM private cloud implementation.



File Systems in Cloud Computing

–

Linux system based

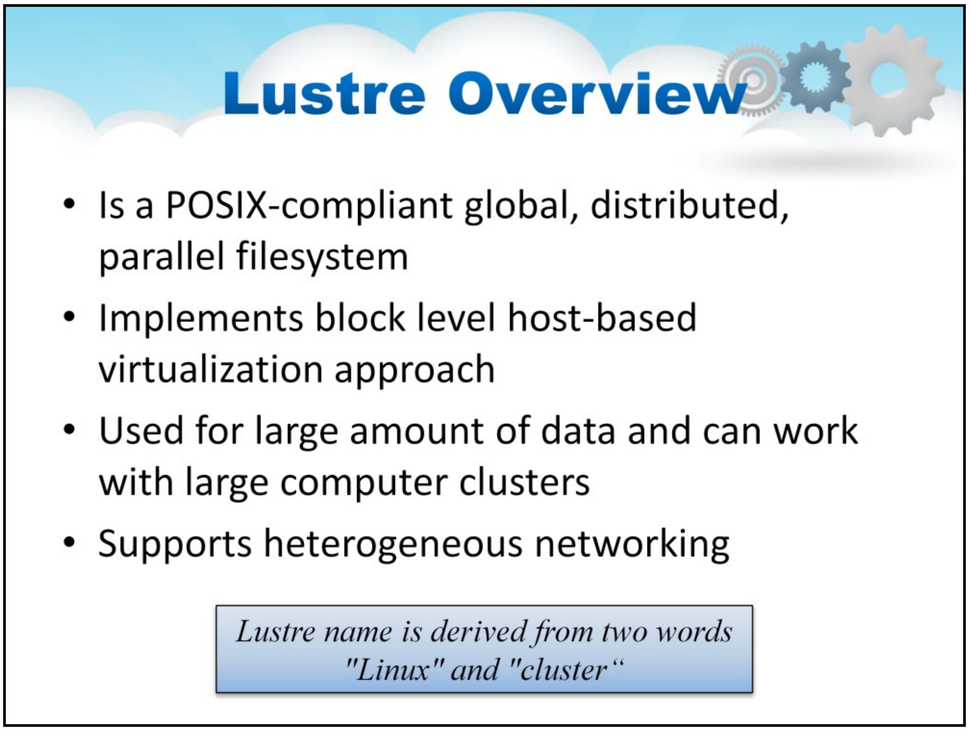
–

File based

–

Lustre

Lustre

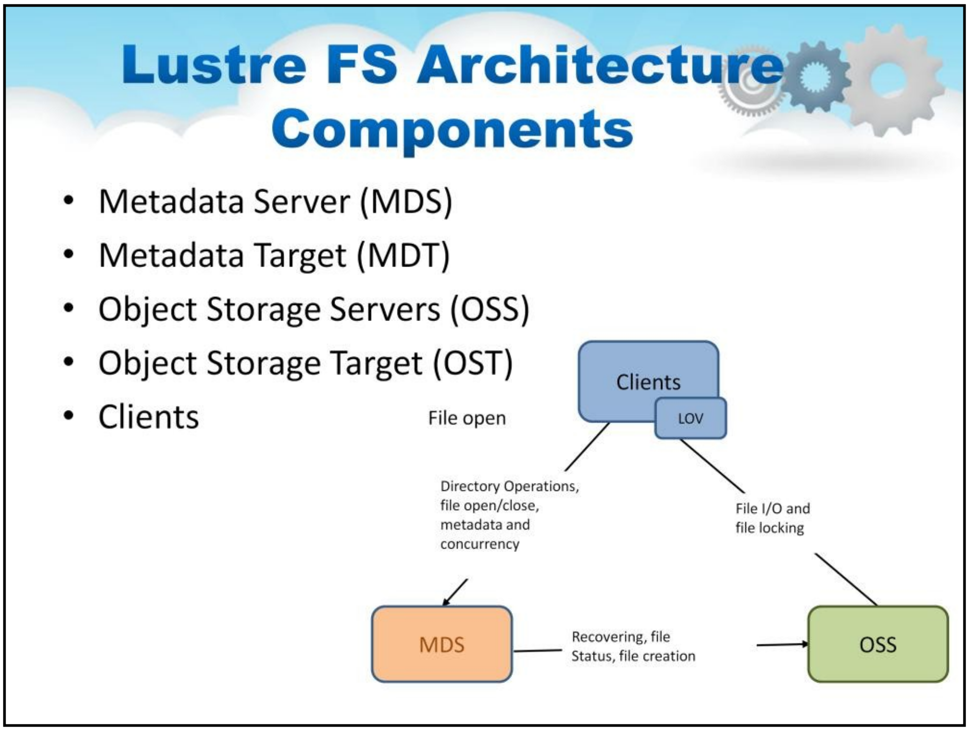


Lustre Overview

- Is a POSIX-compliant global, distributed, parallel filesystem
- Implements block level host-based virtualization approach
- Used for large amount of data and can work with large computer clusters
- Supports heterogeneous networking

Lustre name is derived from two words "Linux" and "cluster“

Lustre is a type of parallel distributed file system used for large amount of data and can work with large computer clusters.
Lustre name is derived from two words "Linux" and "cluster".
Lustre is often used as a file system for supercomputers and multi-site computer clusters.
Lustre storage cluster may contain thousands of nodes and Petabytes of storage volume.
Lustre architecture includes three main components: metadata servers that stores filesystem information (files and directories) as well as access rights, object storage servers, and clients that access and use data.
Lustre uses unified namespace compatible with POSIX semantics.



Lustre FS Architecture Components

- Metadata Server (MDS)
- Metadata Target (MDT)
- Object Storage Servers (OSS)
- Object Storage Target (OST)
- Clients

File open

Clients

LOV

Directory Operations, file open/close, metadata and concurrency

File I/O and file locking

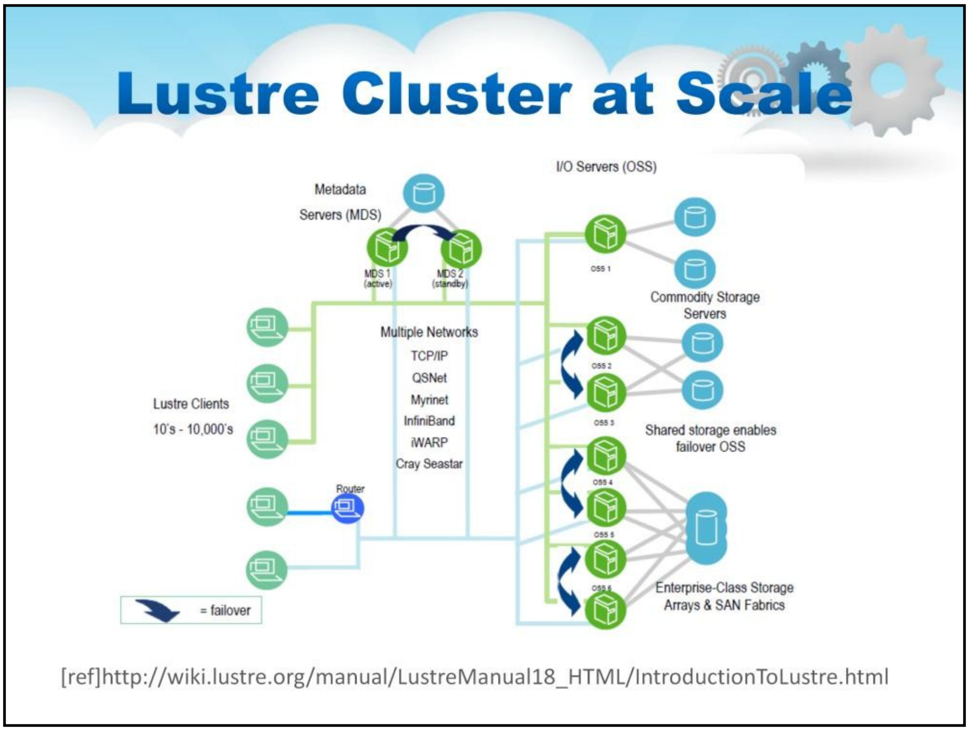
MDS

Recovering, file Status, file creation

OSS

The MDS server makes metadata stored in one or more MDTs.
The MDT stores metadata (such as filenames, permissions) on an MDS.
The OSS provides file I/O service, and network request handling for one or more local OSTs
The OST stores file data as data objects on one or more OSSs
A groups of OSS are wrapped into Logical Object Volume (LOV)

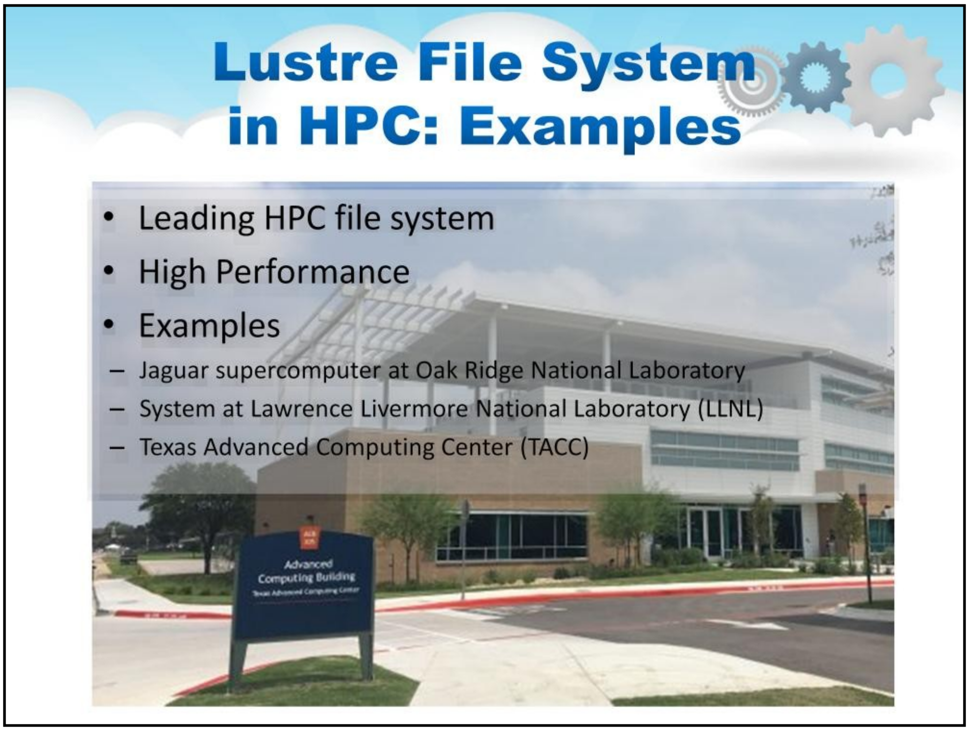
The main file system components inside of Lustre are described in this slide.
Note one of the most significant elements of the design is the notion of many Object Storage Servers. This lends to the scalability of the design.



Lustre Cluster at Scale

[ref]http://wiki.lustre.org/manual/LustreManual18_HTML/IntroductionToLustre.html

This slide illustrates the scalability design introduced in the previous slide.
In general, highly available and high scalability concepts are both used in large deployments. Here a Lustre deployment at scale is illustrated showing multiple networks between clients and the Lustre cluster, and also the number of I/O Servers (paired as fail over groups).



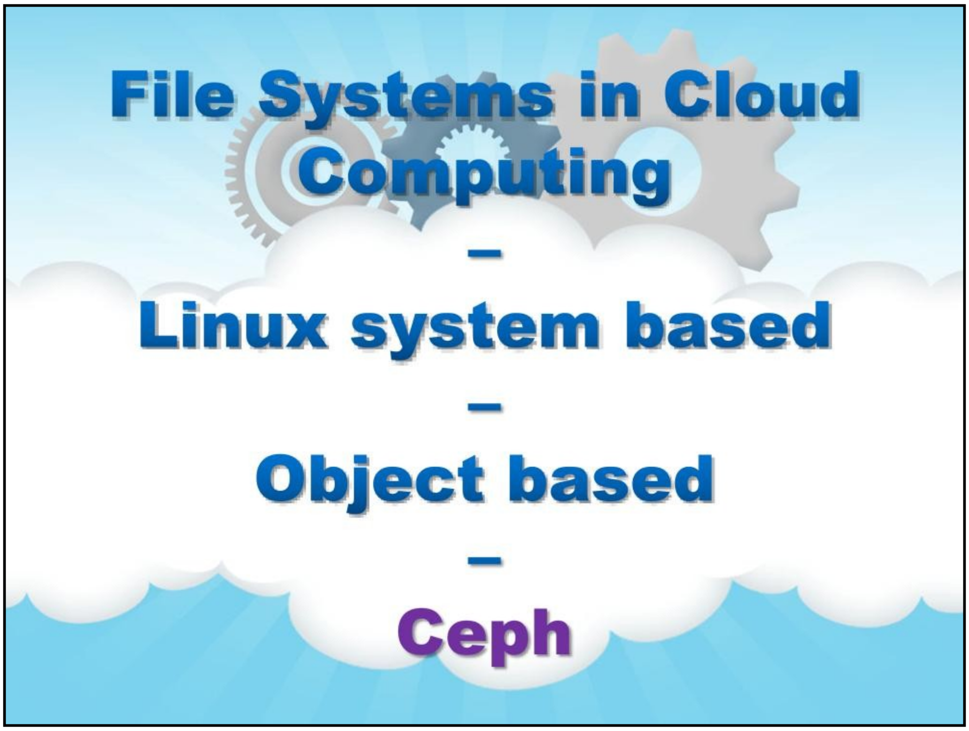
Lustre File System in HPC: Examples

- Leading HPC file system
- High Performance
- Examples
  - Jaguar supercomputer at Oak Ridge National Laboratory
  - System at Lawrence Livermore National Laboratory (LLNL)
  - Texas Advanced Computing Center (TACC)

Advanced
Computing Building
Texas Advanced Computing Center

1. 7 of Top 10
Over 40% of Top100
Demonstrated scalability
2. 190 GB/sec IO
26,000 clients
Systems with over 1,000 nodes

The high performance computing community has been working on pushing the limits of performance and scalability and Lustre has achieved popularity within that community. Lustre has significant momentum in the HPC community and is actually the lading distributed filesystem for those systems, as the slide details.
You can see impressive scale-out and high performance numbers achieved.



File Systems in Cloud Computing

–

Linux system based

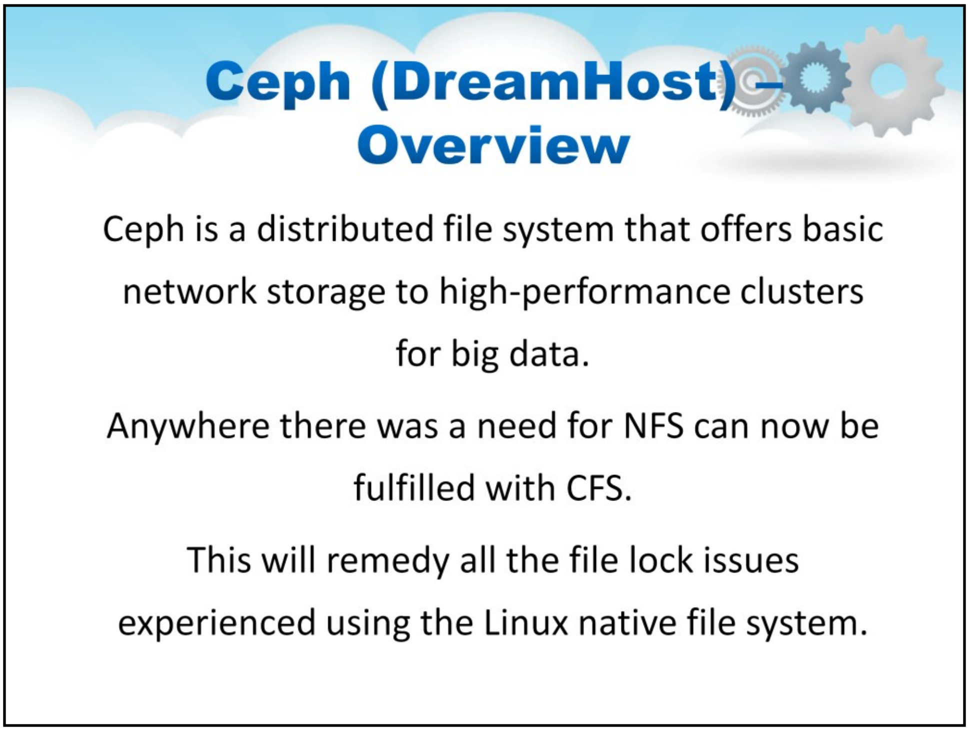
–

Object based

–

Ceph

Ceph

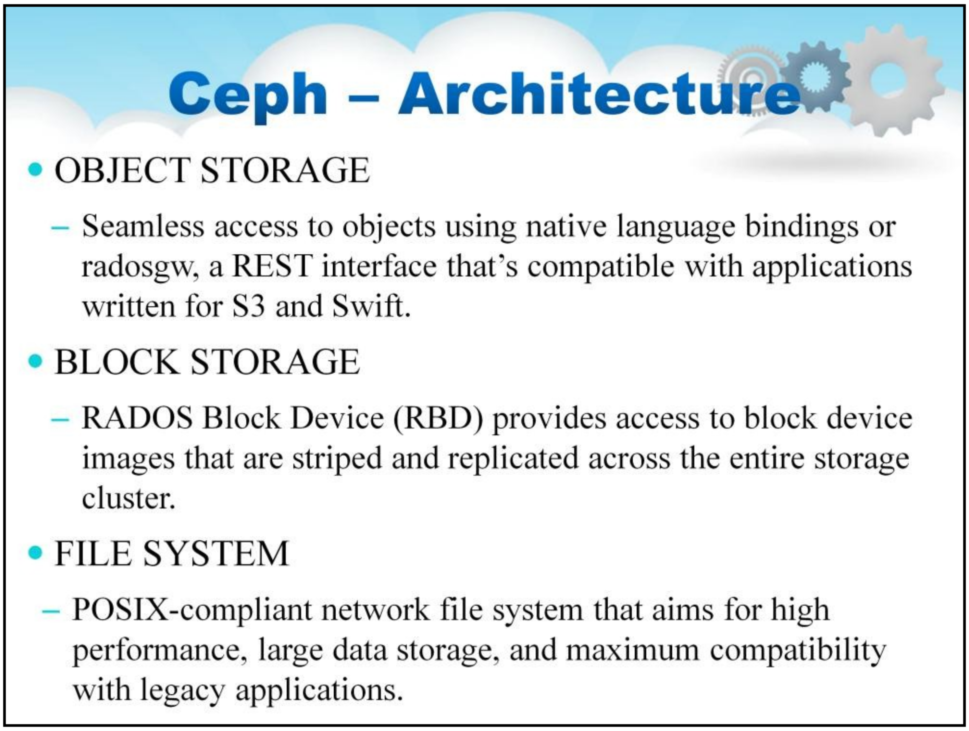


Ceph (DreamHost) – Overview

Ceph is a distributed file system that offers basic network storage to high-performance clusters for big data.

Anywhere there was a need for NFS can now be fulfilled with CFS.

This will remedy all the file lock issues experienced using the Linux native file system.



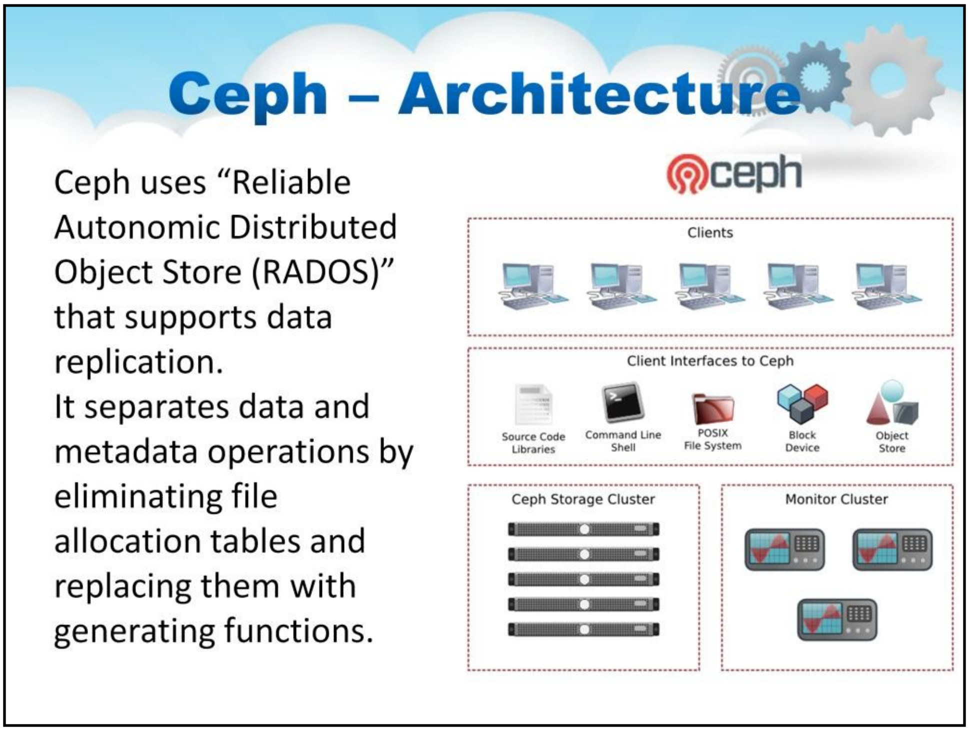
Ceph – Architecture

- OBJECT STORAGE
  - Seamless access to objects using native language bindings or radosgw, a REST interface that's compatible with applications written for S3 and Swift.
- BLOCK STORAGE
  - RADOS Block Device (RBD) provides access to block device images that are striped and replicated across the entire storage cluster.
- FILE SYSTEM
  - POSIX-compliant network file system that aims for high performance, large data storage, and maximum compatibility with legacy applications.

Ceph Object Storage and/or Ceph Block Device services are modules to be integrated into Cloud Platforms, deploy a Ceph Filesystem or use Ceph for another purpose, all Ceph Storage Cluster deployments begin with setting up each Ceph Node, your network and the Ceph Storage Cluster.

A Ceph Storage Cluster requires at least one Ceph Monitor and at least two Ceph OSD Daemons. The Ceph Metadata Server is essential when running Ceph Filesystem clients.

Ceph stores a client's data as objects within storage pools. Using the CRUSH algorithm, Ceph calculates which placement group should contain the object, and further calculates which Ceph OSD Daemon should store the placement group. The CRUSH algorithm enables the Ceph Storage Cluster to scale, rebalance, and recover dynamically.



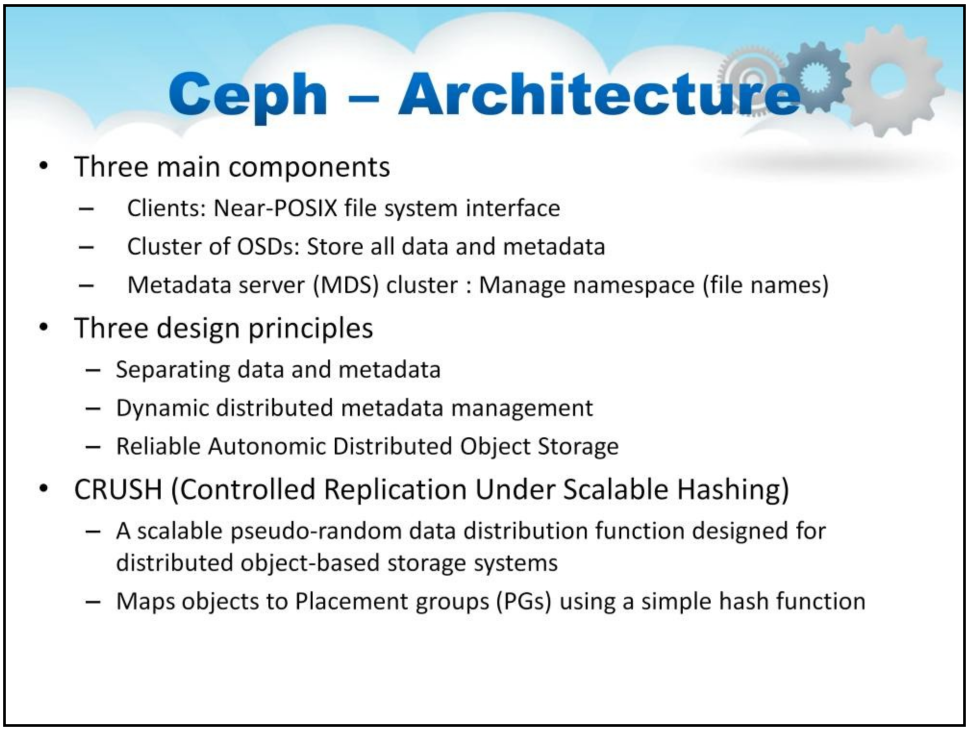
Ceph – Architecture

Ceph uses "Reliable Autonomic Distributed Object Store (RADOS)" that supports data replication.
It separates data and metadata operations by eliminating file allocation tables and replacing them with generating functions.

1. Designed to integrate object, block and file storage servers from a single distributed computer cluster
2. Ceph maps file names and directories across RADOS cluster
Ceph block storage can be directly mounted to a VM and provides automatic data replication.
3. Ceph utilizes a highly adaptive distributed metadata cluster, to improve scalability

It was acquired by Red Hat in 2014

Ceph is an example of fully distributed storage architecture (not having central management) and the file system designed to integrate object, block and file storage servers from a single distributed computer cluster.
Ceph distributed object storage is built around the Reliable Autonomic Distributed Object Store (RADOS) that support data replication. Ceph block storage can be directly mounted to a VM and provides automatic data replication across the storage cluster.
Ceph file system runs on top of the object or block storage and maps file names and directories across RADOS cluster.



Ceph – Architecture

- Three main components
  - Clients: Near-POSIX file system interface
  - Cluster of OSDs: Store all data and metadata
  - Metadata server (MDS) cluster : Manage namespace (file names)
- Three design principles
  - Separating data and metadata
  - Dynamic distributed metadata management
  - Reliable Autonomic Distributed Object Storage
- CRUSH (Controlled Replication Under Scalable Hashing)
  - A scalable pseudo-random data distribution function designed for distributed object-based storage systems
  - Maps objects to Placement groups (PGs) using a simple hash function

Ceph has three components

Clients: Near-POSIX file system interface

Cluster of OSDs: Store all data and metadata, Uses CRUSH function to assigns objects to storage devices

Metadata server (MDS) cluster : Manage namespace (file names)

It is designed for high availability and scalability using key design patterns:

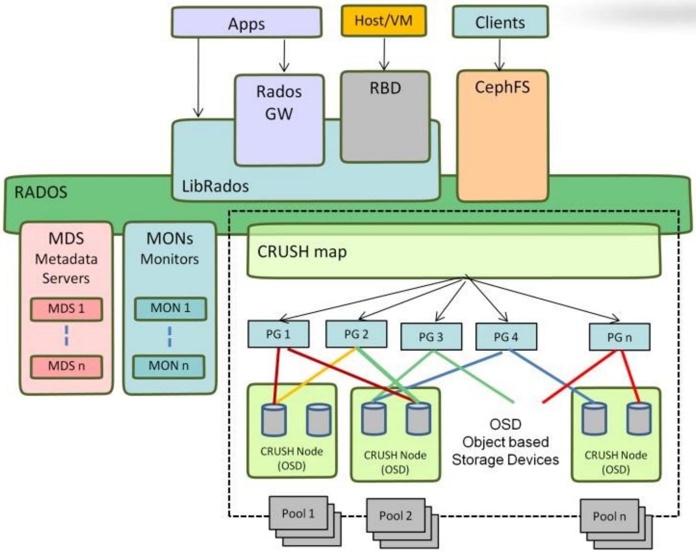
Separating data and metadata

Dynamic distributed metadata management

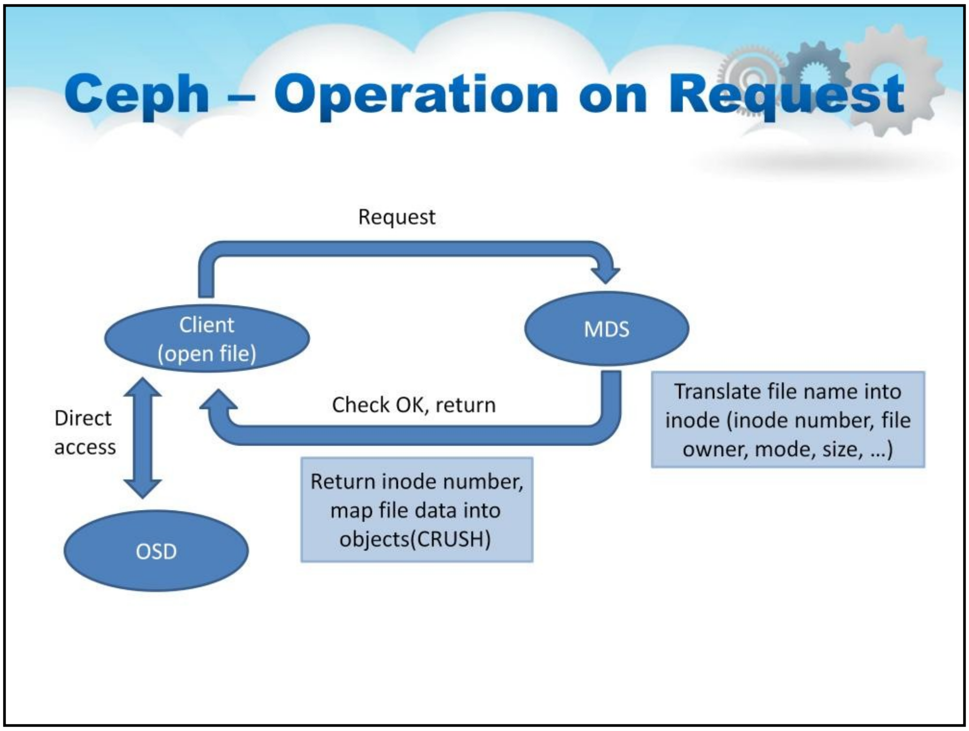
Reliable Autonomic Distributed Object Storage

PGs are assigned to OSDs by CRUSH

Ceph – Architecture



Ceph separates data and metadata operations
Data/file request includes request to MDS to obtain file components/nodes location and metadata



Ceph – Operation on Request

Request

Client (open file)

MDS

Translate file name into inode (inode number, file owner, mode, size, …)

Check OK, return

Direct access

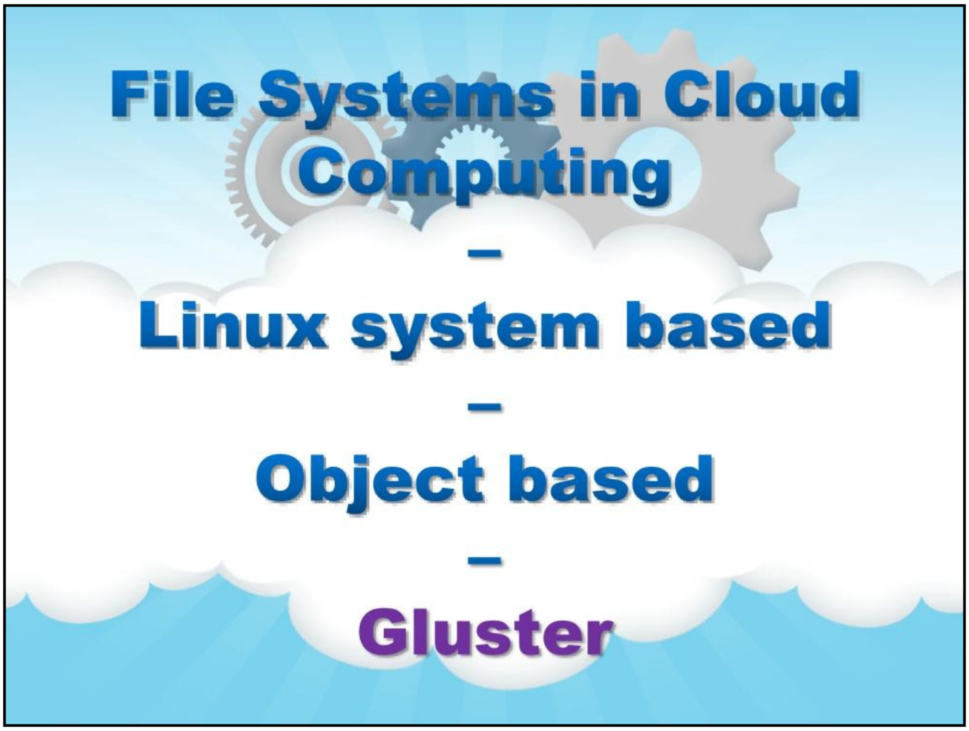
OSD

Return inode number, map file data into objects(CRUSH)

Client synchronization

- If multiple clients (readers and writers) use the same file, the MDS will revoke any previously read and write requests until acknowledged by OSD
- Forces clients synchronization

Ceph uses an effective client synchronization model.
The client makes a request to the Metadata Server which translates the file name into inode (inode number, file owner, mode, size, …)
Then the CRUSH (Controlled Replication Under Scalable Hashing) module goes to work.
CRUSH is A scalable pseudo-random data distribution function designed for distributed object-based storage systems
Maps objects to Placement groups (PGs) using a simple hash function
It returns inode number, map file data into objects.
The client then accesses the Object Storage Device, as can be seen by the illustration.



File Systems in Cloud Computing

–

Linux system based

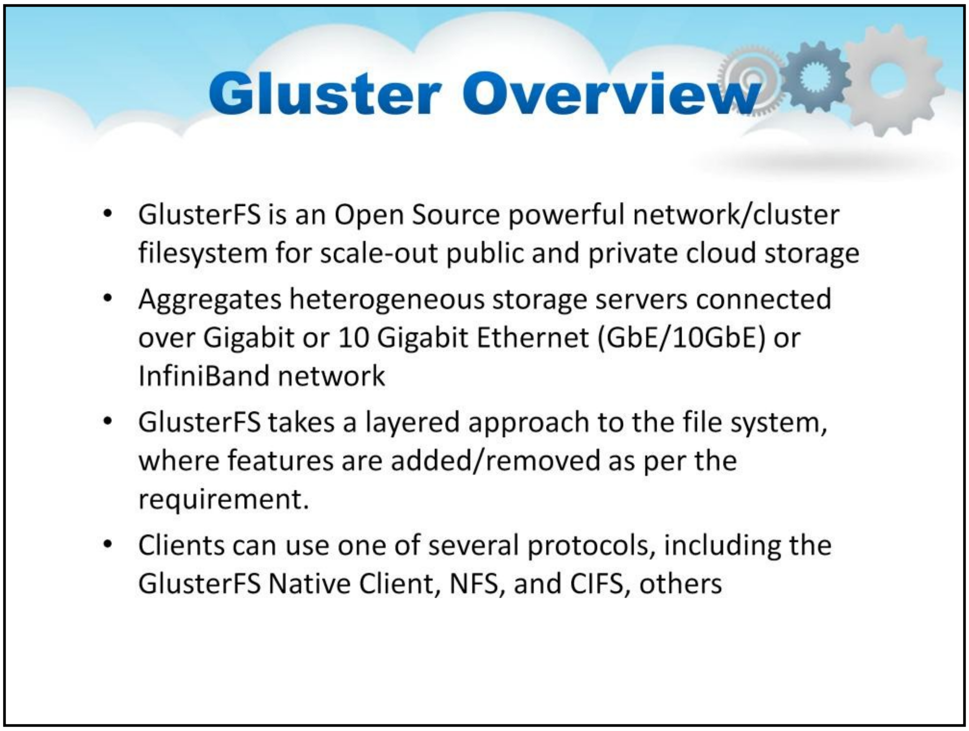
–

Object based

–

Gluster

Gluster



Gluster Overview

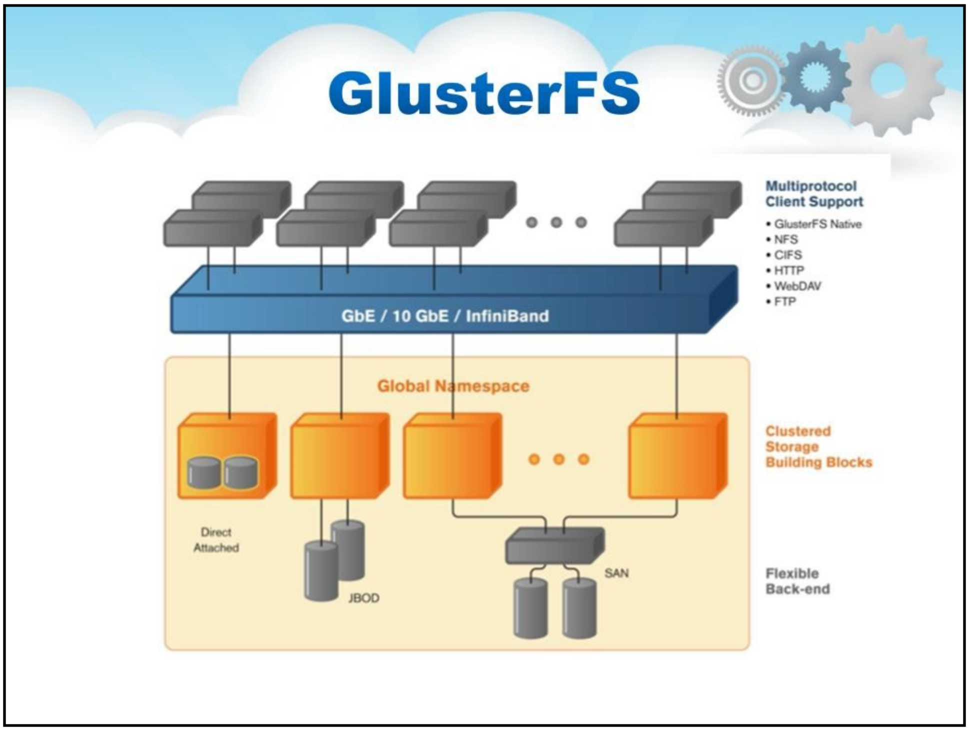
- GlusterFS is an Open Source powerful network/cluster filesystem for scale-out public and private cloud storage
- Aggregates heterogeneous storage servers connected over Gigabit or 10 Gigabit Ethernet (GbE/10GbE) or InfiniBand network
- GlusterFS takes a layered approach to the file system, where features are added/removed as per the requirement.
- Clients can use one of several protocols, including the GlusterFS Native Client, NFS, and CIFS, others

1. Provides simple functionality and leaves all file management functionality to clients
2. Can create a global namespace from a set of clustered storage building blocks that include direct attached storage, JBOD (Just a Bunch of Disks), and SAN fabrics
3. Written in user space and uses FUSE to hook itself with VFS (Virtual File System) layer

Gluster uses existing disk file systems like ext3, ext4, xfs, etc. to store the data
Scale up to petabytes of storage which is available under a single mount point
Acquired by Red Hat in Fall 2011

Gluster storage and files system is an Open Source platform for scale-out public and private cloud storage. Similar to Lustre, the Gluster name is derived from two words “GNU” and “cluster”. Gluster aggregates heterogeneous storage server connected over Ethernet or Infiniband network.

The Gluster file system provides simple functionality and leave all file management functionality to clients.



GlusterFS

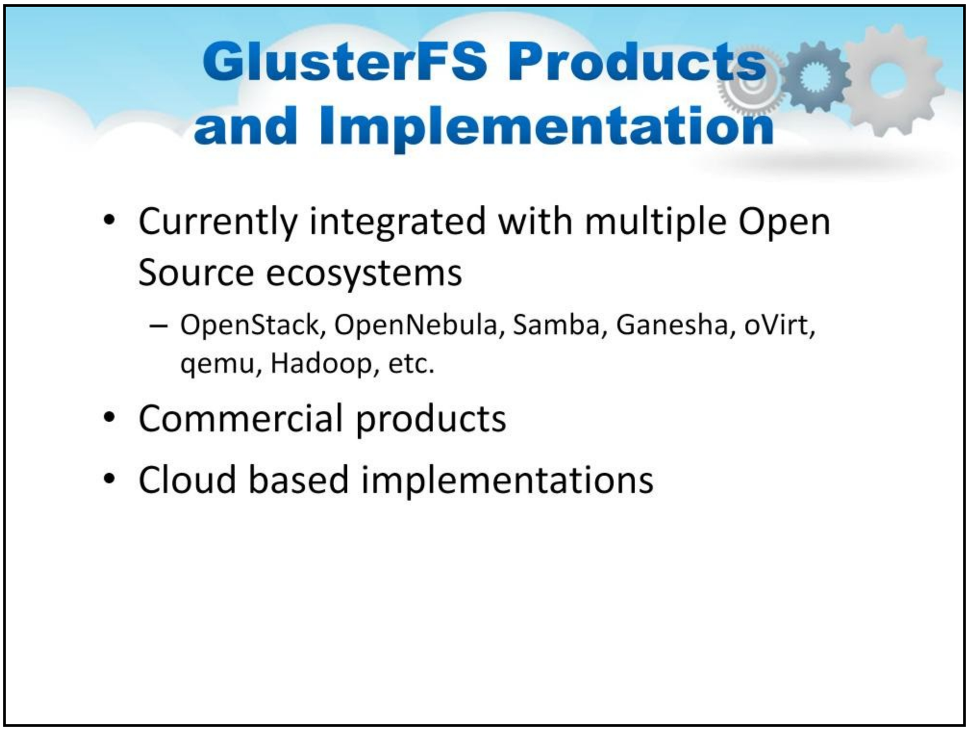
Multiprotocol Client Support

- GlusterFS Native
- NFS
- CIFS
- HTTP
- WebDAV
- FTP

Clustered Storage Building Blocks

Flexible Back-end

- GlusterFS is the distributed file system for commodity hardware
- Each server plus attached commodity storage (configured as DAS, JBOD, or SAN) is considered to be a node.
- Capacity is scaled by adding additional nodes or adding additional storage to each node.
- Performance is increased by deploying storage among more nodes.
- High availability is achieved by replicating data n-way between nodes.



GlusterFS Products and Implementation

- Currently integrated with multiple Open Source ecosystems
  - OpenStack, OpenNebula, Samba, Ganesha, oVirt, qemu, Hadoop, etc.
- Commercial products
- Cloud based implementations

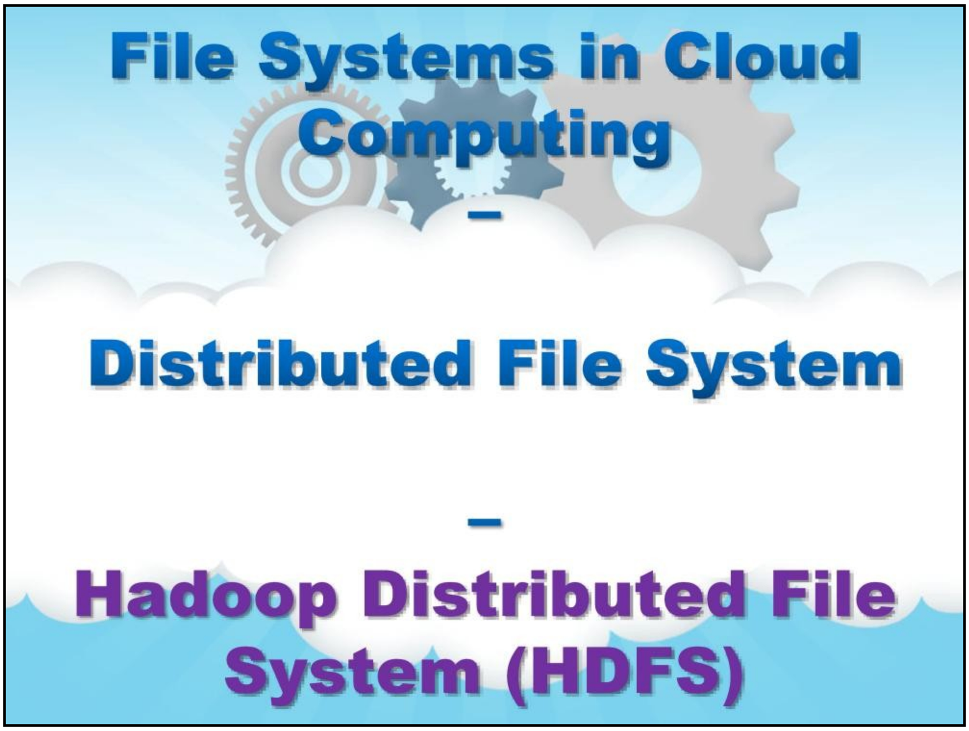
Gluster is a scale-out network-attached storage file system. It has found applications including cloud computing, streaming media services, and content delivery networks. GlusterFS was developed originally by Gluster, Inc., then by Red Hat, Inc., after their purchase of Gluster in 2011.

Think of Gluster as an alternative to Swift and Cinder in OpenStack.

GlusterFS aggregates various storage servers over Ethernet or Infiniband interconnect into one large parallel network file system.

Gluster stores data as files and folders, and uses tokens to identify the location of a file within the cluster. Tokens, which are stored as extended attributes of a file, are themselves distributed across directories thereby enhancing load balancing while avoiding the need for a dedicated metadata server. When a client accesses a file, Gluster translates the requested file name to a token and access the files directly.

Gluster Filesystem enables you to configure the cluster to replicate files across storage devices, thereby high availability to files and data in your storage environment.



File Systems in Cloud Computing

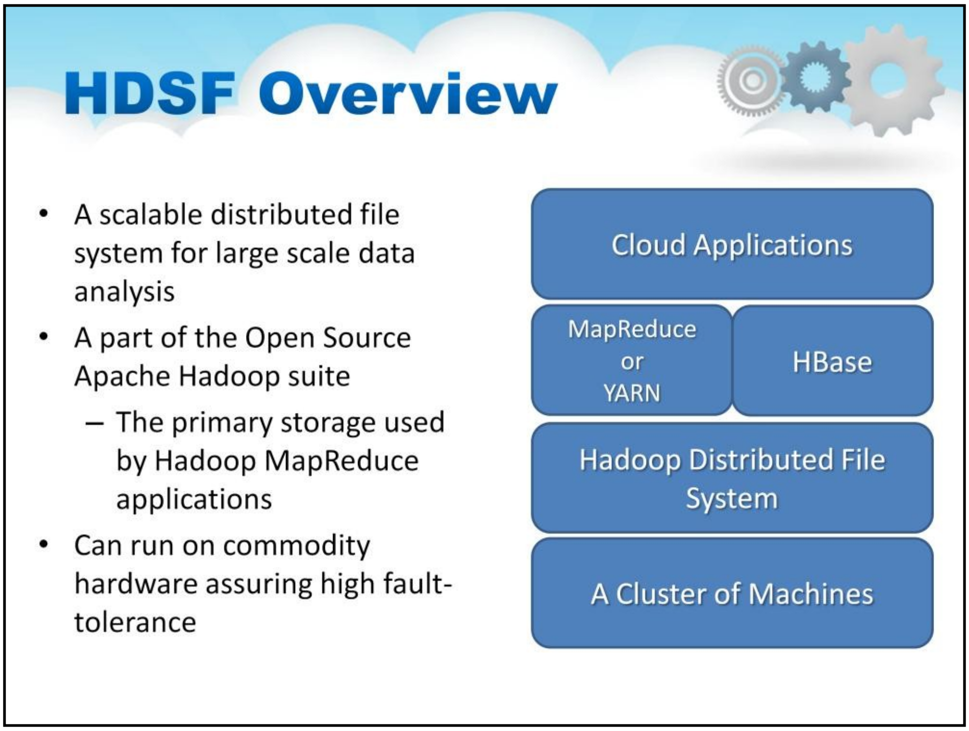
–

Distributed File System

–

Hadoop Distributed File System (HDFS)

HDFS



HDSF Overview

- A scalable distributed file system for large scale data analysis
- A part of the Open Source Apache Hadoop suite
  - The primary storage used by Hadoop MapReduce applications
- Can run on commodity hardware assuring high fault-tolerance

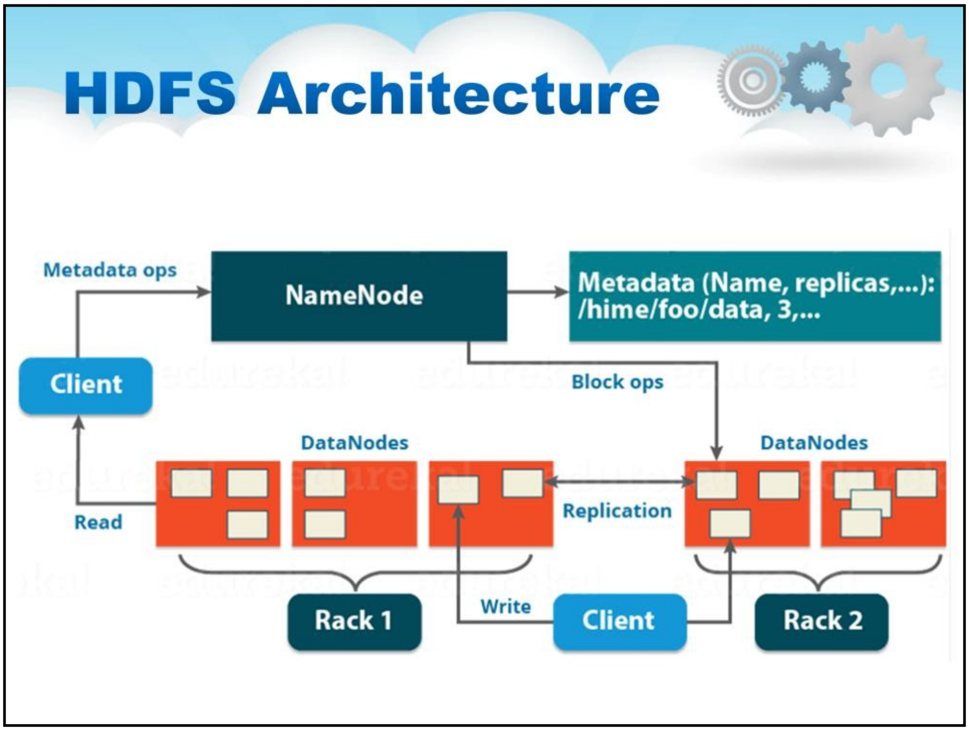
The Hadoop Distributed File System (HDFS) is a very different sort of “filesystem” optimized for a specific class of applications, those are Map Reduce and similar “Big Data” systems like “no-SQL” databases.

It is a scalable distributed file system for large scale data analysis

A part of the Open Source Apache Hadoop suite

The primary storage used by Hadoop MapReduce applications

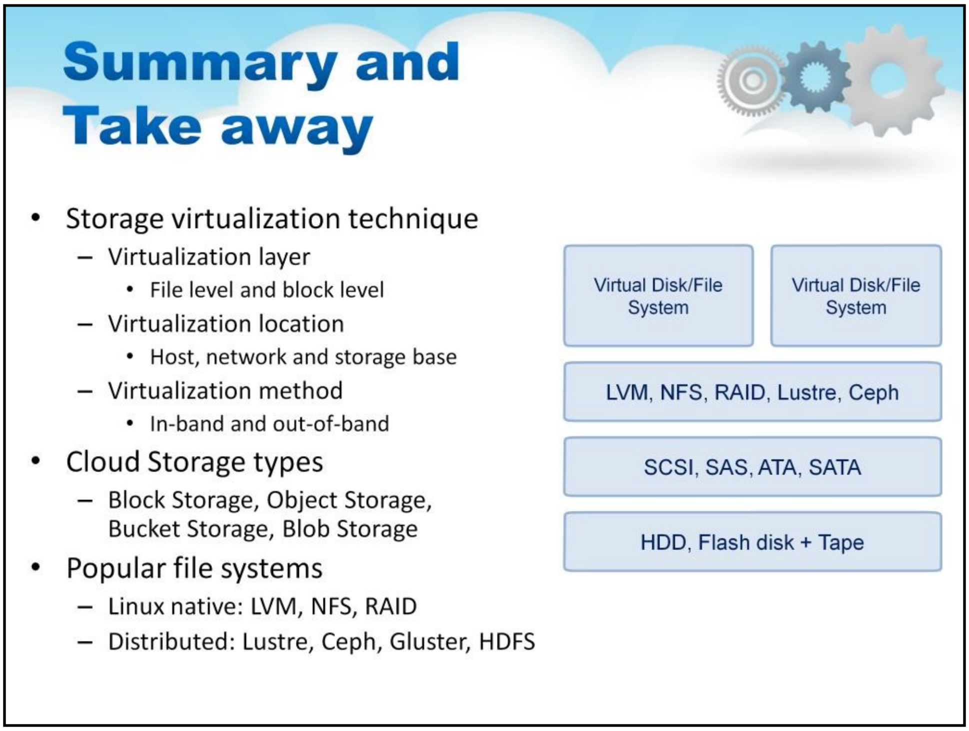
Can run on commodity hardware assuring high fault-tolerant



HDFS Architecture

File content is split into blocks (default 128MB, 3 replica).
NameNode maintains the namespace tree and the mapping of file blocks to DataNodes.
Files and directories are represented on the NameNode by *inodes* (permissions, modification and access times, namespace and disk space quotas*).*
Namespace is a hierarchy of files and directories.

HDFS cluster consists of a single master node/server that runs NameNode and multiple DataNodes, usually one per physical node in the Hadoop cluster. User data are stored in the files, externally they are exposed through namespace managed by the NameNode. To access a file, a user client needs to request a file location or metadata from the NameNode, and after that it can send read or write request to the DataNode directly. DataNodes create data blocks and do replication based on instructions from NameNode.



Summary and Take away

- Storage virtualization technique
  - Virtualization layer
    - File level and block level
  - Virtualization location
    - Host, network and storage base
  - Virtualization method
    - In-band and out-of-band
- Cloud Storage types
  - Block Storage, Object Storage, Bucket Storage, Blob Storage
- Popular file systems
  - Linux native: LVM, NFS, RAID
  - Distributed: Lustre, Ceph, Gluster, HDFS

Virtual Disk/File System

Virtual Disk/File System

LVM, NFS, RAID, Lustre, Ceph

SCSI, SAS, ATA, SATA

HDD, Flash disk + Tape

This lecture has explored the various ways that Storage is virtualized and implemented for large scale, distributed systems, including Cloud.

We explored the storage primitive which was virtualized, and saw that some systems concentrate on virtualizing files, and some systems concentrate on virtualizing blocks.

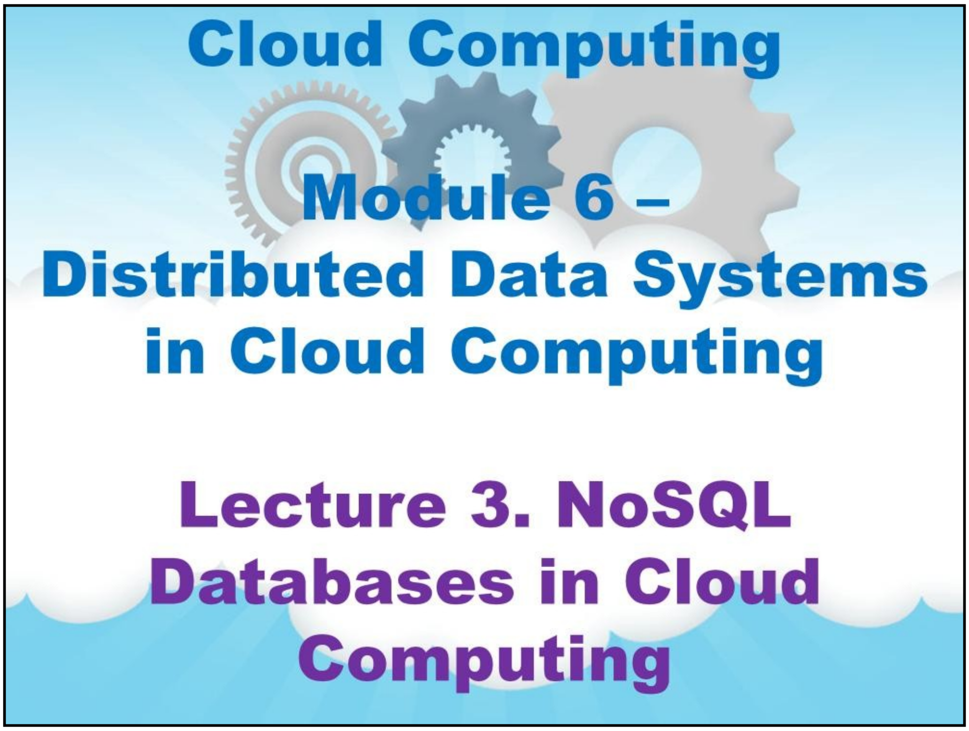
We saw that the virtualization function can run in a variety of places in the architecture. It can run in the host, in the network, or all the way back where the drives are.

We saw that virtualization can be placed “in band” of the storage operations, and for scale, is usually placed “out of band”.

There are many types of storage primitives which, after all the virtualization has occurred, end up getting exposed to applications. Objects (buckets, blobs), blocks, or file systems

There are many ways to layer the file system in leveraging the virtualization and replication in a cluster. The capability can be close to the operating system suchas LVM or across the network like NFS.

Finally, we took a hard look at several of the New file systems optimized for managing heterogeneous cloud storage farms

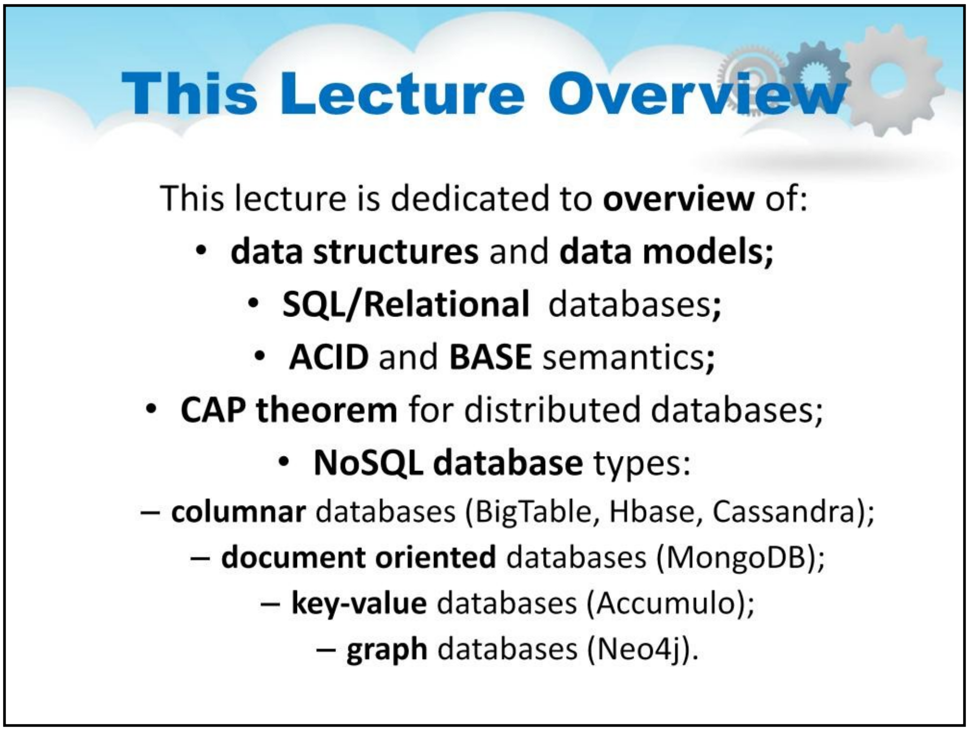


Cloud Computing

Module 6 –
Distributed Data Systems
in Cloud Computing

Lecture 3. NoSQL
Databases in Cloud
Computing

This Lecture 3 is about NoSQL Databases in Cloud Computing.



This Lecture Overview

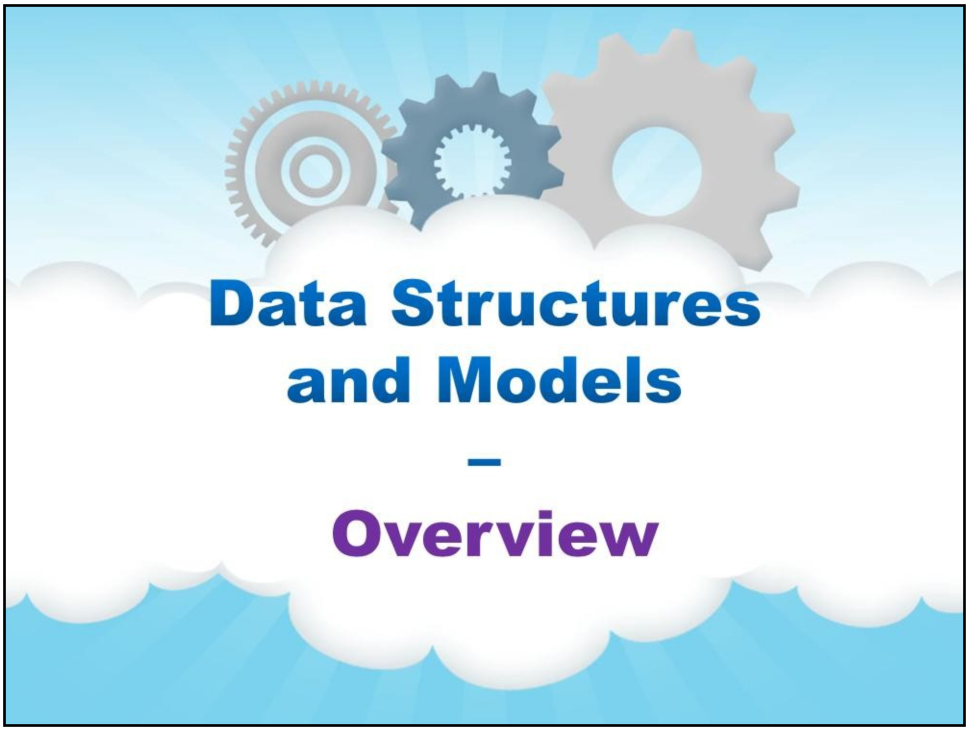
This lecture is dedicated to **overview** of:

- **data structures** and **data models;**
- **SQL/Relational** databases**;**
- **ACID** and **BASE** semantics**;**
- **CAP theorem** for distributed databases;
- **NoSQL database** types:
  - **columnar** databases (BigTable, Hbase, Cassandra);
  - **document oriented** databases (MongoDB);
  - **key-value** databases (Accumulo);
  - **graph** databases (Neo4j).

Lecture 3. NoSQL Databases in Cloud Computing

This lecture is dedicated to:

- **data structures** and **data models;**
- **SQL/Relational** databases**;**
- **ACID** and **BASE** semantics**;**
- **CAP theorem** for distributed databases;
- **NoSQL database** types:
  - **columnar** databases (BigTable, Hbase, Cassandra);
  - **document oriented** databases (MongoDB);
  - **key-value** databases (Accumulo);
  - **graph** databases (Neo4j).



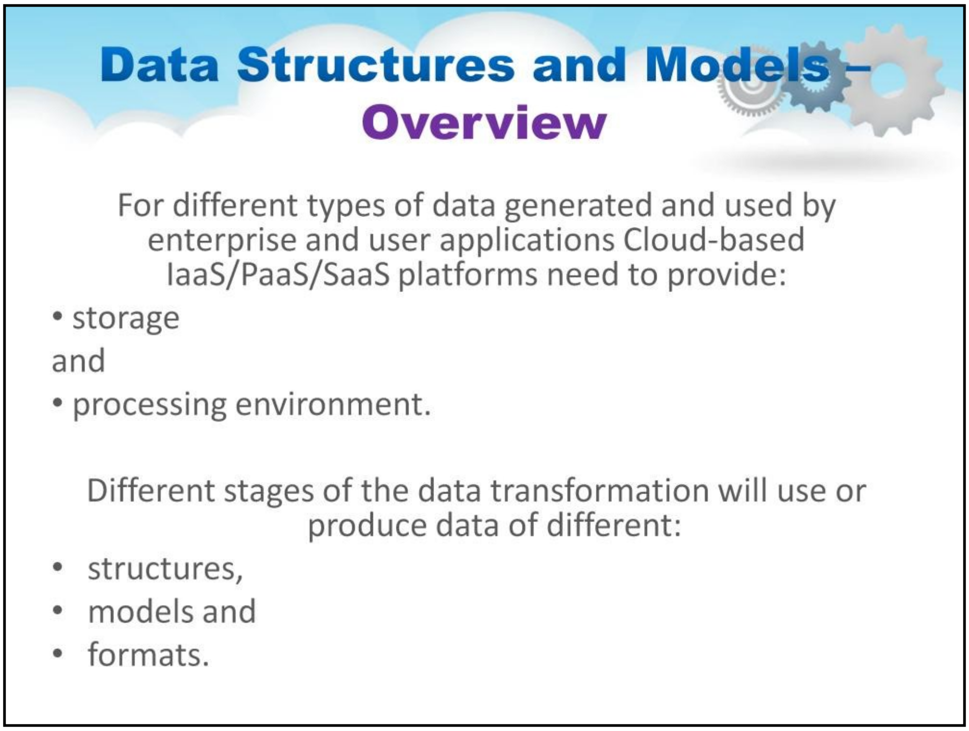
Data Structures and Models

–

Overview

Overview

Let’s start from overview of data Structures and Models…



Data Structures and Models – Overview

For different types of data generated and used by enterprise and user applications Cloud-based IaaS/PaaS/SaaS platforms need to provide:

- storage

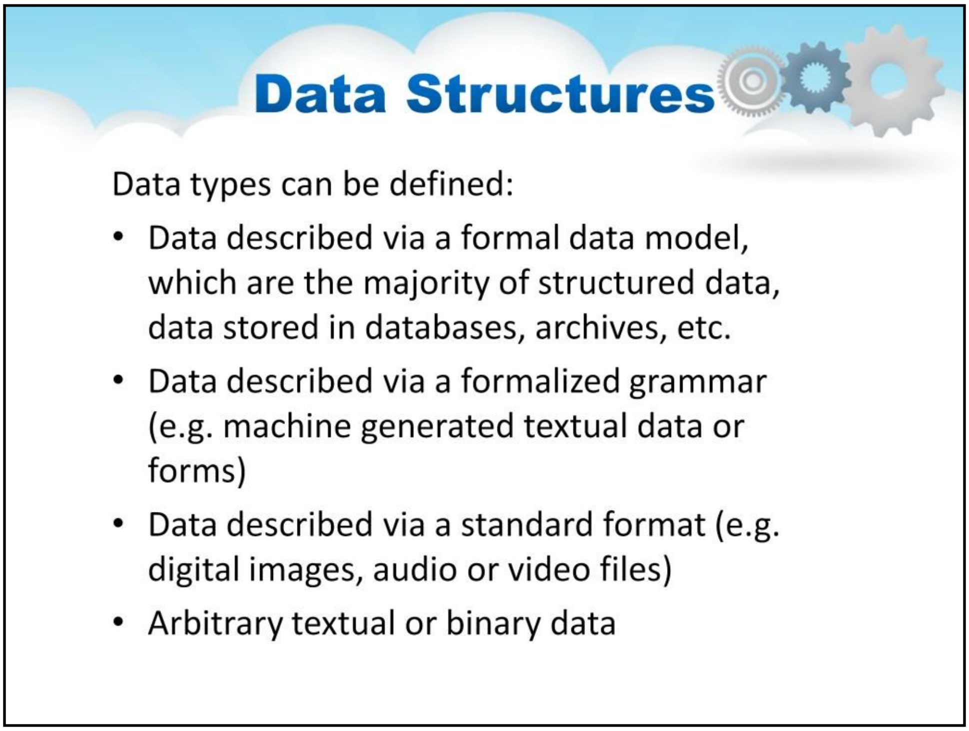
and

- processing environment.

Different stages of the data transformation will use or produce data of different:

- structures,
- models and
- formats.

Big Data systems and applications will use and/or produce different data types that are defined by their origin or target use. In its own turn, different stages of the data transformation will also use or produce data of different structures, models and formats.



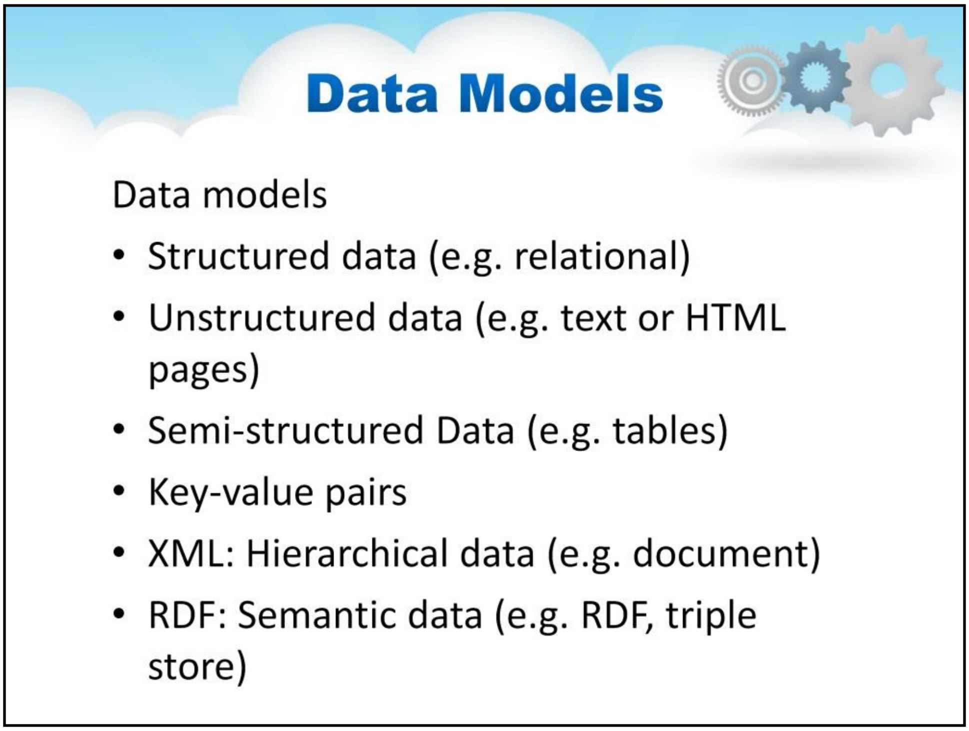
Data Structures

Data types can be defined:

- Data described via a formal data model, which are the majority of structured data, data stored in databases, archives, etc.
- Data described via a formalized grammar (e.g. machine generated textual data or forms)
- Data described via a standard format (e.g. digital images, audio or video files)
- Arbitrary textual or binary data

The following data types can be defined:

- data described via a formal data model, which are the majority of structured data, data stored in databases, archives, etc.
- data described via a formalized grammar (e.g. machine generated textual data or forms)
- data described via a standard format (many examples of digital images, audio or video files, also formatted binary data)
- arbitrary textual or binary data



Data Models

Data models

- Structured data (e.g. relational)
- Unstructured data (e.g. text or HTML pages)
- Semi-structured Data (e.g. tables)
- Key-value pairs
- XML: Hierarchical data (e.g. document)
- RDF: Semantic data (e.g. RDF, triple store)

The following slides will explain examples of various data models with

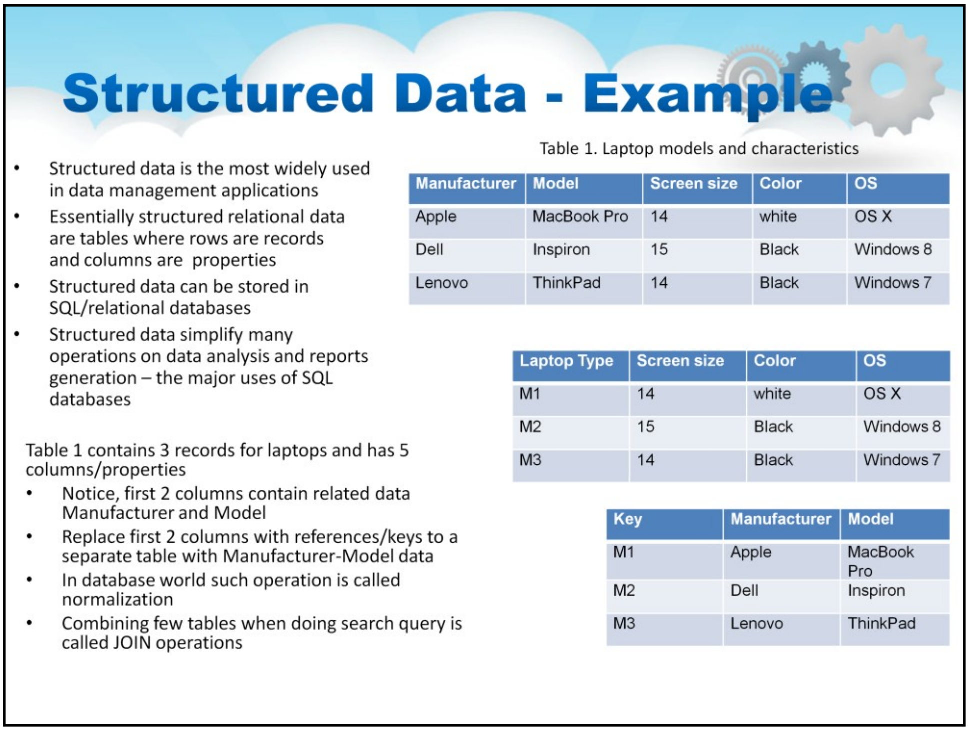
- Structured data (e.g. relational)
- Unstructured data (e.g. text or HTML pages)
- Semi-structured Data (e.g. tables)
- Key-value pairs
- XML: Hierarchical data (e.g. document)
- RDF: Semantic data (e.g. RDF, triple store)

In the following we will look at some examples

Here just to mention about unstructured data which example is textual data

Although textual data are widely used everywhere, first of all, for intelligent reporting and communication between humans in natural language, they have almost no explicit structure. It is also possible that machine generated text has some formal grammar but it is rather common that text created by humans will contain many grammatical irregularities.

Web pages or HTML documents represent another example of unstructured text-type data. Currently query operations over textual data are quite simple, an example of which are Search Engine operations. You can use simple Boolean search control commands in the search form (e.g. check advanced Google search for this) but at the end that simply searches in the huge but still flat index database.



Structured Data - Example

- Structured data is the most widely used in data management applications
- Essentially structured relational data are tables where rows are records and columns are properties
- Structured data can be stored in SQL/relational databases
- Structured data simplify many operations on data analysis and reports generation – the major uses of SQL databases

Table 1 contains 3 records for laptops and has 5 columns/properties

- Notice, first 2 columns contain related data Manufacturer and Model
- Replace first 2 columns with references/keys to a separate table with Manufacturer-Model data
- In database world such operation is called normalization
- Combining few tables when doing search query is called JOIN operations

Table 1. Laptop models and characteristics

Manufacturer	Model	Screen size	Color	OS
Apple	MacBook Pro	14	white	OS X
Dell	Inspiron	15	Black	Windows 8
Lenovo	ThinkPad	14	Black	Windows 7

Laptop Type	Screen size	Color	OS
M1	14	white	OS X
M2	15	Black	Windows 8
M3	14	Black	Windows 7

Key	Manufacturer	Model
M1	Apple	MacBook Pro
M2	Dell	Inspiron
M3	Lenovo	ThinkPad

Now let us look deeply at the way data is structured (or not)

Now to look at Structured Data

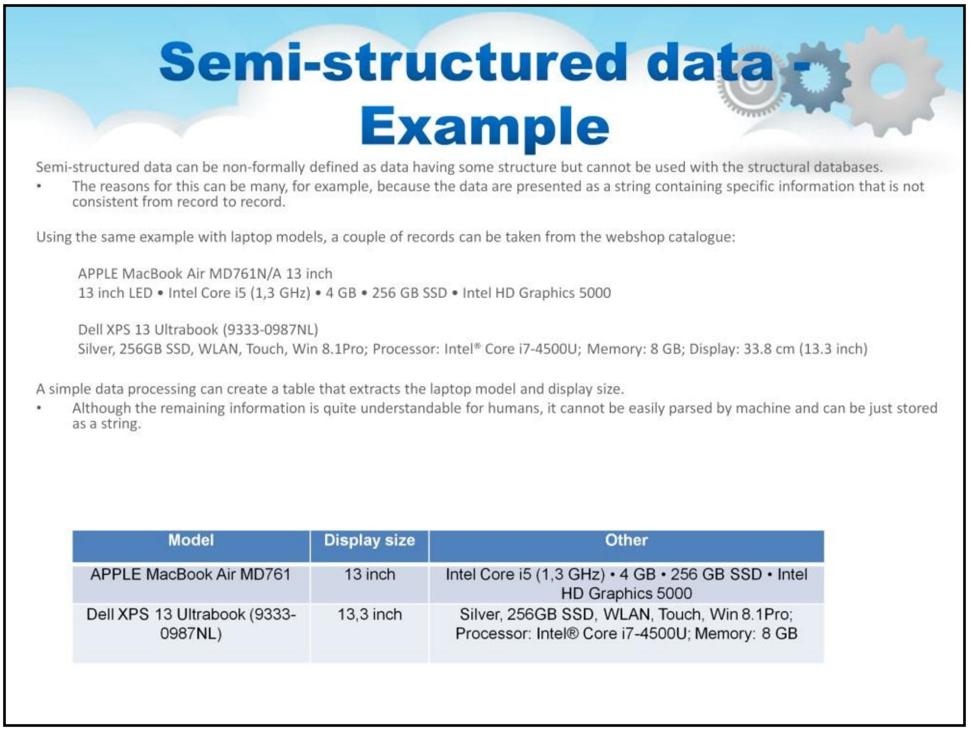
Structured data is the most widely used in data management applications.

Essentially structured relational data are tables where rows are records and columns are properties

Structured data can be stored in SQL/relational databases

Structured data simplify many operations on data analysis and reports generation – the major uses of SQL databases.

The slide illustrates Structured Data. Not how the data fits perfectly into tables. Note that there is one table which “joins” the other two tables together, via a key. These are all very common types of techniques in Structured Data.



Semi-structured data - Example

Semi-structured data can be non-formally defined as data having some structure but cannot be used with the structural databases.

- The reasons for this can be many, for example, because the data are presented as a string containing specific information that is not consistent from record to record.

Using the same example with laptop models, a couple of records can be taken from the webshop catalogue:

APPLE MacBook Air MD761N/A 13 inch
13 inch LED • Intel Core i5 (1,3 GHz) • 4 GB • 256 GB SSD • Intel HD Graphics 5000

Dell XPS 13 Ultrabook (9333-0987NL)
Silver, 256GB SSD, WLAN, Touch, Win 8.1Pro; Processor: Intel® Core i7-4500U; Memory: 8 GB; Display: 33.8 cm (13.3 inch)

A simple data processing can create a table that extracts the laptop model and display size.

- Although the remaining information is quite understandable for humans, it cannot be easily parsed by machine and can be just stored as a string.

Model	Display size	Other
APPLE MacBook Air MD761	13 inch	Intel Core i5 (1,3 GHz) • 4 GB • 256 GB SSD • Intel HD Graphics 5000
Dell XPS 13 Ultrabook (9333-0987NL)	13,3 inch	Silver, 256GB SSD, WLAN, Touch, Win 8.1Pro; Processor: Intel® Core i7-4500U; Memory: 8 GB

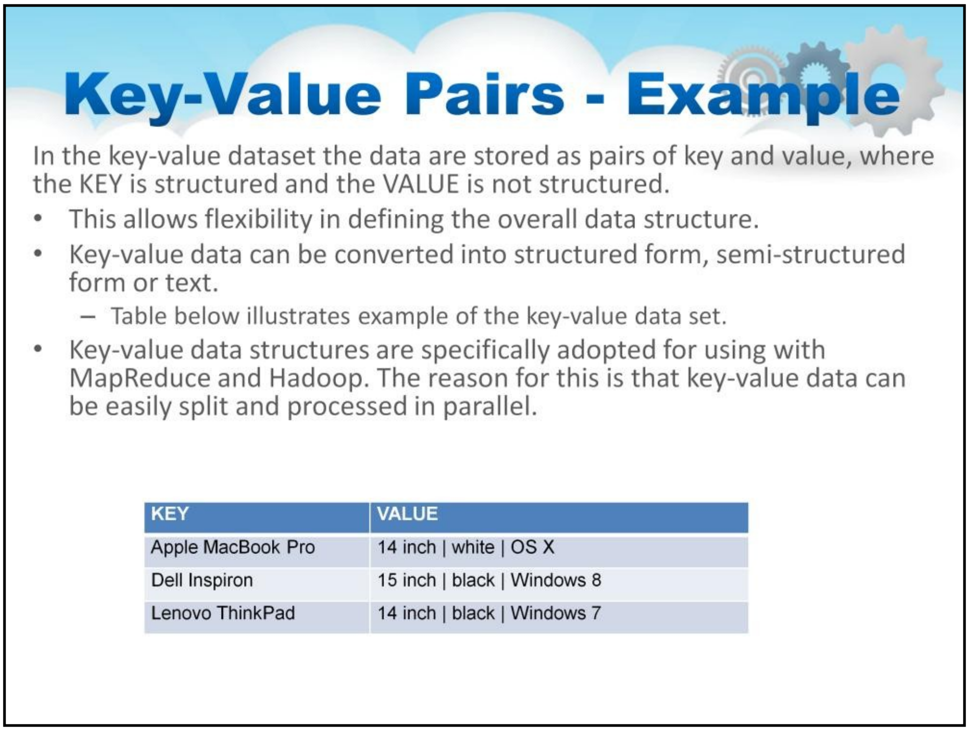
Next we will look at Semi-structured data.

Think of semi-structured data defined as data having some structure but cannot be used with the structural databases.

The reasons for this can be many. The slide presents an example where the semi-structured data source has a structure where there are two parts, that is apparent to any software. The first part is intended to be a product model, that is followed by second part which is meant to a description. The “two part” aspect is the “structure” in this example.

The “semi-structured” part is that the text content – the model and the description – are text of “regular language” – while they are perfectly human readable, they don’t have enough structure to be machine readable.

While easily fitted into the table, the data is not as useful as it might be.



Key-Value Pairs - Example

In the key-value dataset the data are stored as pairs of key and value, where the KEY is structured and the VALUE is not structured.

- This allows flexibility in defining the overall data structure.
- Key-value data can be converted into structured form, semi-structured form or text.
  - Table below illustrates example of the key-value data set.
- Key-value data structures are specifically adopted for using with MapReduce and Hadoop. The reason for this is that key-value data can be easily split and processed in parallel.

KEY	VALUE
Apple MacBook Pro	14 inch | white | OS X
Dell Inspiron	15 inch | black | Windows 8
Lenovo ThinkPad	14 inch | black | Windows 7

One often hears about data being structured in “Key Value Pairs”. We will define that.

In the key-value dataset the data are stored as pairs of key and value, where the KEY is structured and the VALUE is not structured.

This allows flexibility in defining the overall data structure.

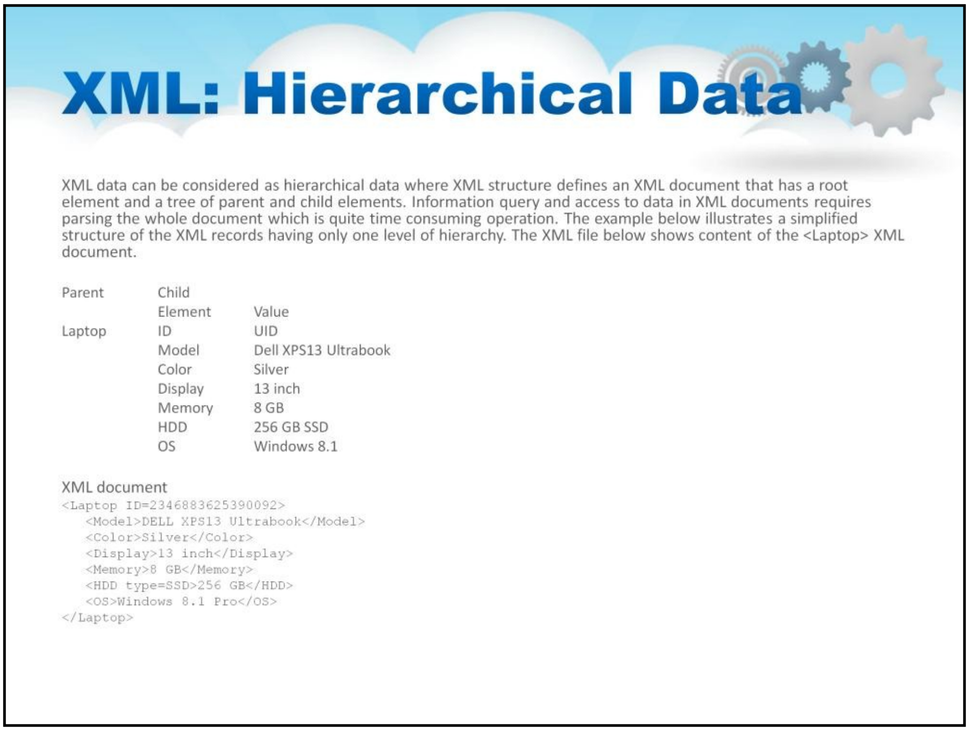
Key-value data can be converted into structured form, semi-structured form or text.

The Table in the slide illustrates an example of a key-value data set.

One can see this is improvement over the semi-structured data set, in that the first part of the data has been put in a rigid format, where it can be used as a “key”, that is matched up with a corresponding set of data which may have a little or a lot of structure. The point is the Key makes it easy to organize this data specifically to divide it up and work on figuring out the data set which the keys are paired with.

Key-value data structures are specifically adopted for using with MapReduce and Hadoop. The reason for this is that key-value data can be easily split and processed in parallel.

Please, see the more detailed Lecture on MapReduce and Hadoop. For now it should be logical and a take-away that the key-value structure leads itself to some kind of divide and conquer algorithm, of which one of the most popular is called MapReduce.



XML: Hierarchical Data

XML data can be considered as hierarchical data where XML structure defines an XML document that has a root element and a tree of parent and child elements. Information query and access to data in XML documents requires parsing the whole document which is quite time consuming operation. The example below illustrates a simplified structure of the XML records having only one level of hierarchy. The XML file below shows content of the <Laptop> XML document.

Parent	Child	
	Element	Value
Laptop	ID	UID
	Model	Dell XPS13 Ultrabook
	Color	Silver
	Display	13 inch
	Memory	8 GB
	HDD	256 GB SSD
	OS	Windows 8.1

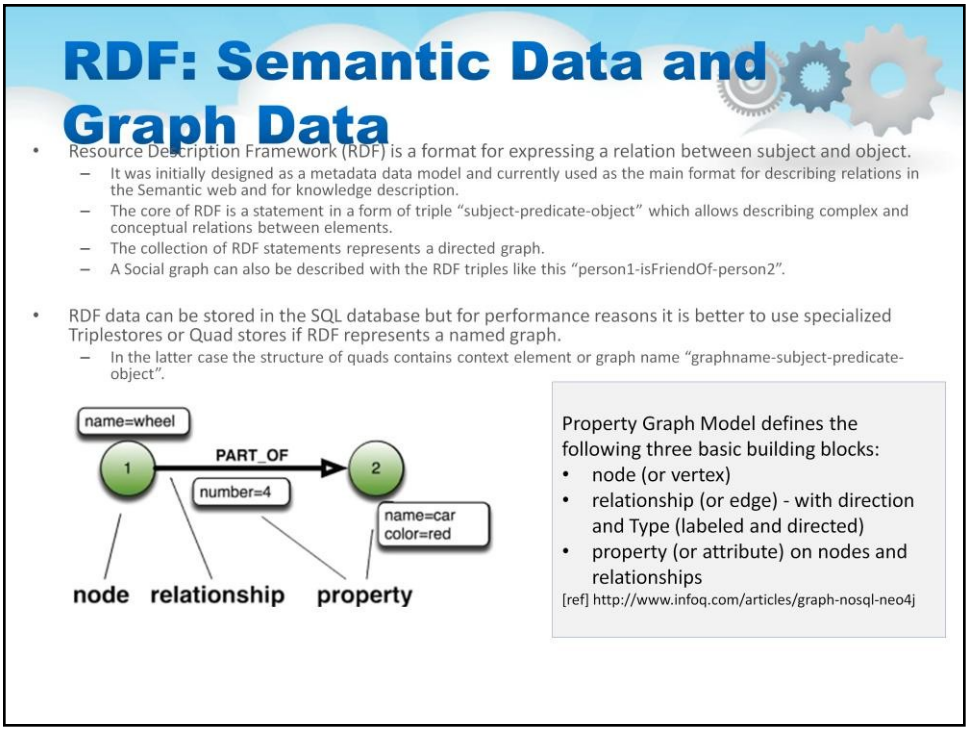
XML document

```
<Laptop ID=2346883625390092>
   <Model>DELL XPS13 Ultrabook</Model>
   <Color>Silver</Color>
   <Display>13 inch</Display>
   <Memory>8 GB</Memory>
   <HDD type=SSD>256 GB</HDD>
   <OS>Windows 8.1 Pro</OS>
</Laptop>
```

XML (eXtensible Markup Language) defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The format of the XML Structure is defined by another file, called the XML Schema. Once the Schema is know, then one can read or write an XML document conforming to that Schema. The XML Schema which defines the data model and is referenced in the document can be a separate file, or can be a part of the document itself in its own section.

For computers, XML is a very un-compact form for structuring data, XML documents add a lot of overhead to data encoded using XML. If one puts binary data in an XML structure – anything from a compressed photo to a Java object – it will be encoded using text characters and become very large and unwieldy. Modules which encode and decode XML, and serialize/de-serialize XML for transmission and reception, respectively, while not complicated can require a large budget of computing and memory resources.

For humans, while XML is theoretically readable and writeable, it is not very human friendly, and rarely is XML interacted with directly, using tools as a layer presenting a much more useable interface are very common.



RDF: Semantic Data and Graph Data

- Resource Description Framework (RDF) is a format for expressing a relation between subject and object.
  - It was initially designed as a metadata data model and currently used as the main format for describing relations in the Semantic web and for knowledge description.
  - The core of RDF is a statement in a form of triple “subject-predicate-object” which allows describing complex and conceptual relations between elements.
  - The collection of RDF statements represents a directed graph.
  - A Social graph can also be described with the RDF triples like this “person1-isFriendOf-person2”.

- RDF data can be stored in the SQL database but for performance reasons it is better to use specialized Triplestores or Quad stores if RDF represents a named graph.
  - In the latter case the structure of quads contains context element or graph name “graphname-subject-predicate-object”.

Property Graph Model defines the following three basic building blocks:

- node (or vertex)
- relationship (or edge) - with direction and Type (labeled and directed)
- property (or attribute) on nodes and relationships

[ref] http://www.infoq.com/articles/graph-nosql-neo4j

A more sophisticate data structure is called “Semantic Data”. Semantic Data is represented in a format called Resource Description Framework (RDF).

RDF is a format for expressing a relation between subject and object.

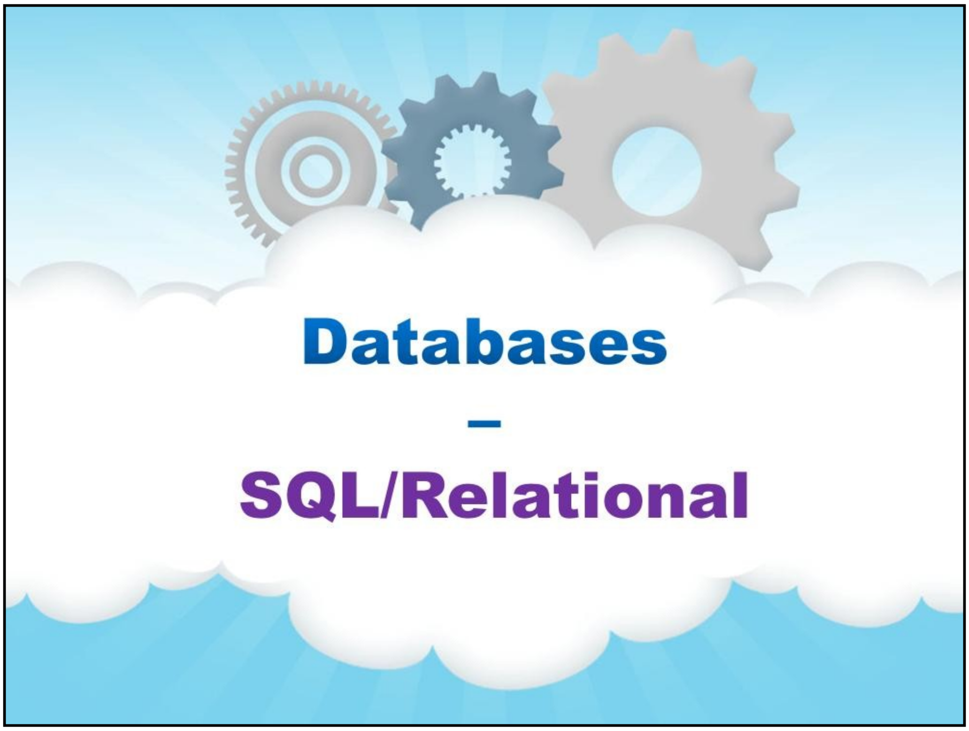
It was initially designed as a metadata data model and currently used as the main format for describing relations in the Semantic web and for knowledge description.

The core of RDF is a statement in a form of triple “subject-predicate-object” which allows describing complex and conceptual relations between elements.

The collection of RDF statements represents a directed graph.

A Social graph can also be described with the RDF triples like this “person1-isFriendOf-person2”.

These subject-predicate-object and graph-based collections of subject-predicate-object entities add considerable information into the data description. Many studies have been done in the Semantic Web project, that the information added into the data description using these relationships enables one to utilize the information as a knowledge-base, not just a data-base. Questions such as “which of these belongs together” and “ which thing is most similar to this other thing”. One can understand more of what the data *means*.



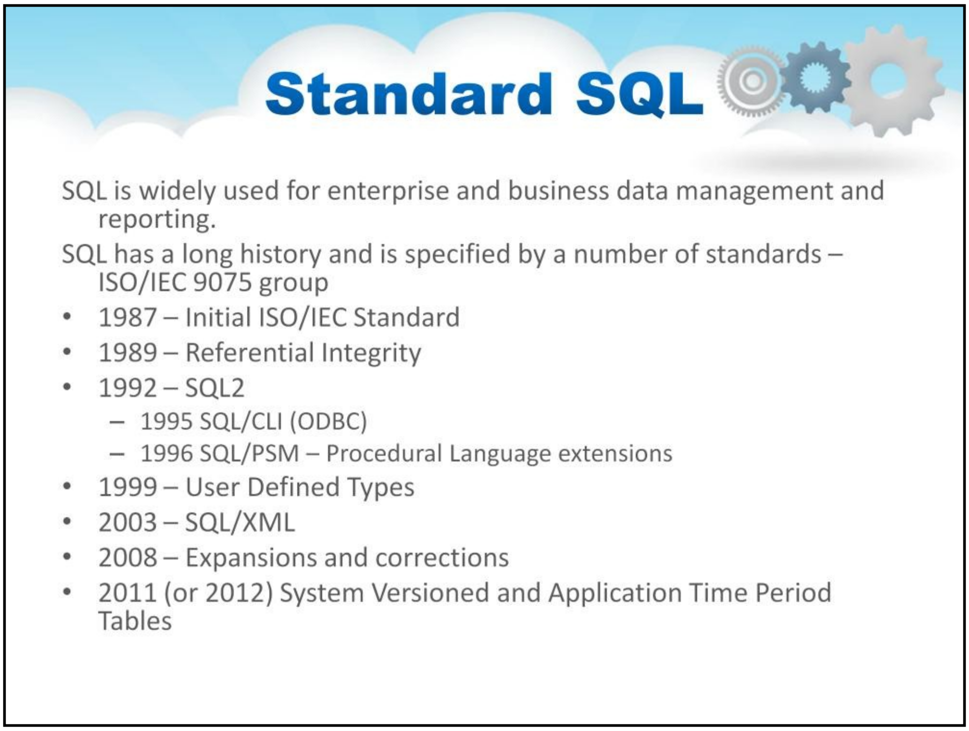
Databases

–

SQL/Relational

Databases

SQL/Relational



Standard SQL

SQL is widely used for enterprise and business data management and reporting.

SQL has a long history and is specified by a number of standards – ISO/IEC 9075 group

- 1987 – Initial ISO/IEC Standard
- 1989 – Referential Integrity
- 1992 – SQL2
  - 1995 SQL/CLI (ODBC)
  - 1996 SQL/PSM – Procedural Language extensions
- 1999 – User Defined Types
- 2003 – SQL/XML
- 2008 – Expansions and corrections
- 2011 (or 2012) System Versioned and Application Time Period Tables

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database.

According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.

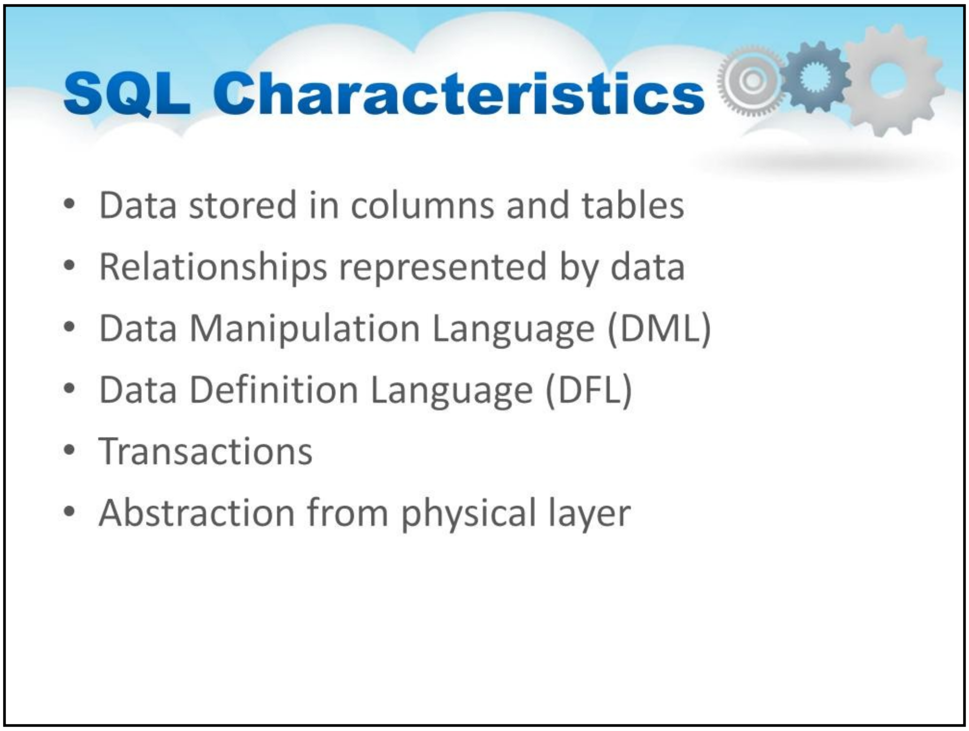
In the end it is a special-purpose programming language

SQL inherently relies on being able to interact with the data in a relational way. There are many commands such as JOIN, SELECT, UPDATE, DELETE, INSERT, WHERE which assume there are relations in the data (or database).

SQL is widely used for enterprise and business data management and reporting.

SQL has a long history and is specified by a number of standards

The slide show many of the milestones of SQL.



SQL Characteristics

- Data stored in columns and tables
- Relationships represented by data
- Data Manipulation Language (DML)
- Data Definition Language (DFL)
- Transactions
- Abstraction from physical layer

An SQL Database is quite complex.

Data Manipulation Language (DML)

In SQL, the data manipulation language comprises the SQL-data change statements, which modify stored data but not the schema or database objects.

Data manipulated with Select, Insert, Update, & Delete statements

Data Aggregation

Compound statements

Functions and Procedures

Explicit transaction control

Data Definition Language (DFL)

Manipulation of persistent database objects, e.g., tables or stored procedures, via the SQL schema statements, rather than the data stored within them.

Schema defined at the start like: Create Table (Column1 Datatype1, Column2 Datatype 2, …)

Constraints to define and enforce relationships

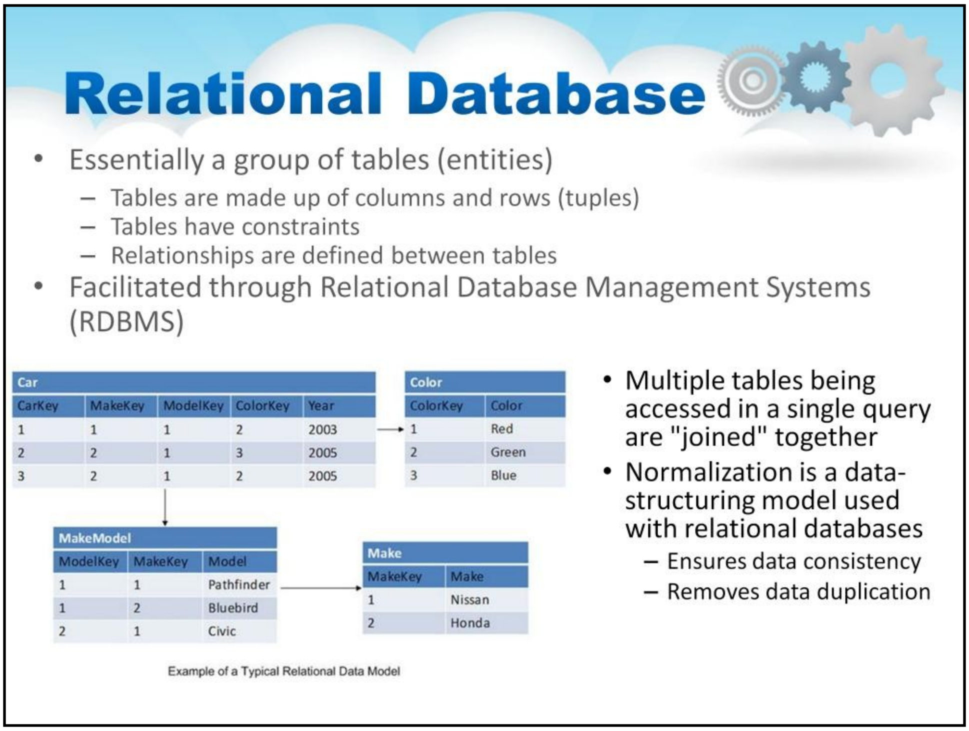
Primary Key

Foreign Key

Etc.

Triggers to respond to Insert, Update , & Delete

Stored Modules



Relational Database

- Essentially a group of tables (entities)
  - Tables are made up of columns and rows (tuples)
  - Tables have constraints
  - Relationships are defined between tables
- Facilitated through Relational Database Management Systems (RDBMS)

Car

CarKey	MakeKey	ModelKey	ColorKey	Year
1	1	1	2	2003
2	2	1	3	2005
3	2	1	2	2005

Color

ColorKey	Color
1	Red
2	Green
3	Blue

MakeModel

ModelKey	MakeKey	Model
1	1	Pathfinder
1	2	Bluebird
2	1	Civic

Make

MakeKey	Make
1	Nissan
2	Honda

Example of a Typical Relational Data Model

- Multiple tables being accessed in a single query are "joined" together
- Normalization is a data-structuring model used with relational databases
  - Ensures data consistency
  - Removes data duplication

Traditionally the database has been a system which one programs with SQL and which supports transactions.

The underlying architecture which makes this possible is a Relational data structure. Relational supports the what the constructs (like SELECT and JOIN and WHERE) need to be implemented.

Relational is a very structured group of tables.

Tables (also called entities) are made up of columns and rows (tuples)

Tables have constraints

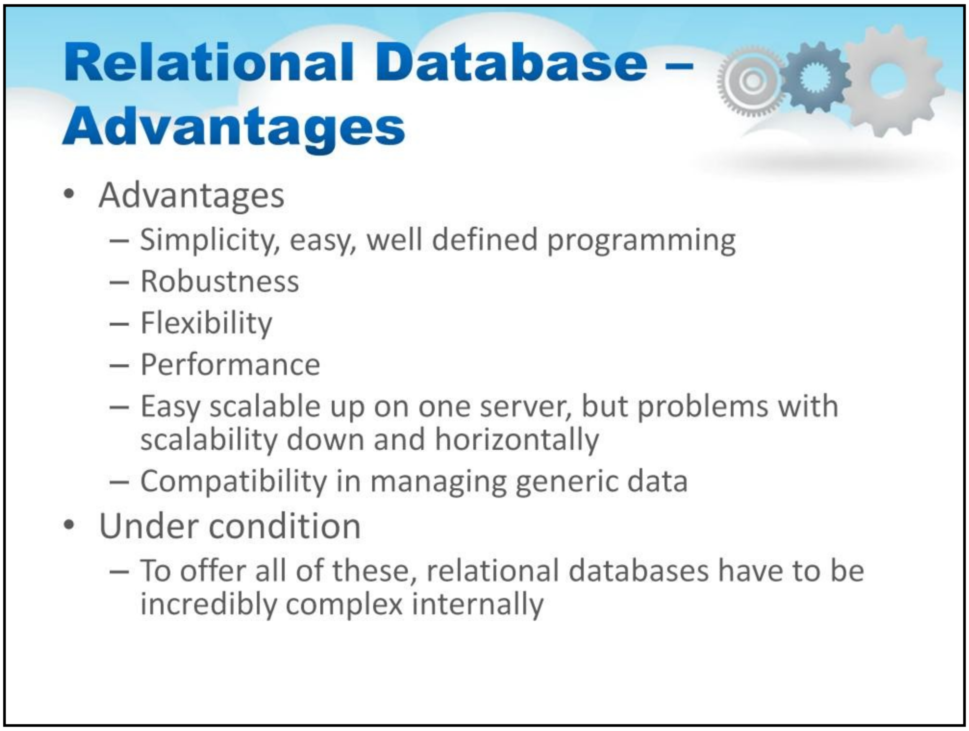
Relationships are defined between tables

This is illustrated in the Slide.

Multiple tables being accessed in a single query are "joined" together

Normalization is a data-structuring model used with relational databases

Ensures data consistency - Removes data duplication



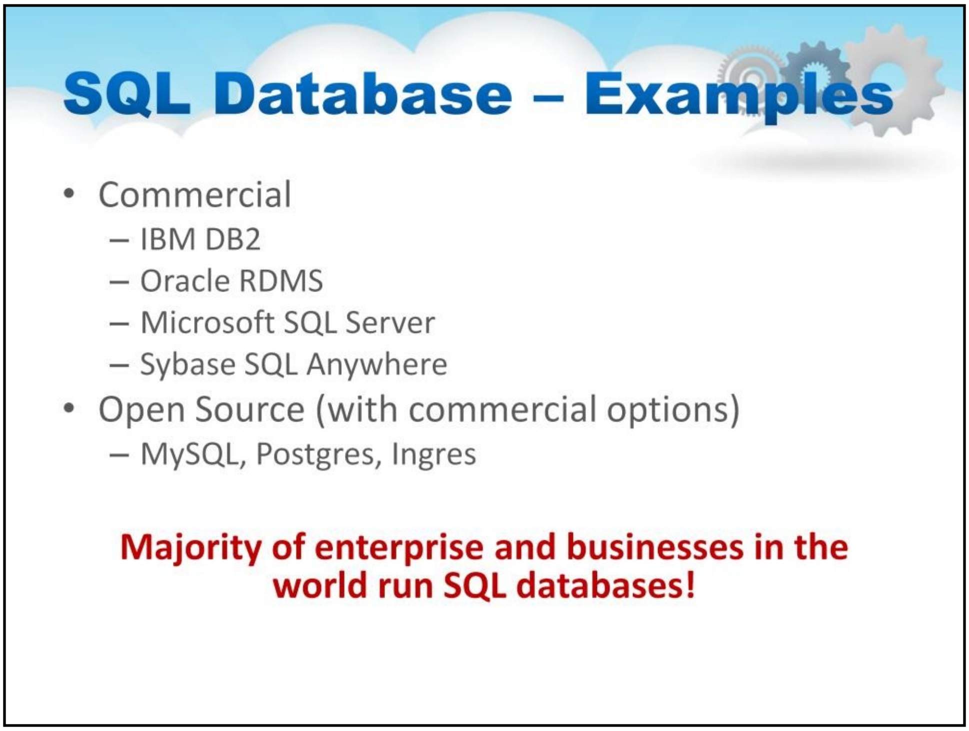
Relational Database – Advantages

- Advantages
  - Simplicity, easy, well defined programming
  - Robustness
  - Flexibility
  - Performance
  - Easy scalable up on one server, but problems with scalability down and horizontally
  - Compatibility in managing generic data
- Under condition
  - To offer all of these, relational databases have to be incredibly complex internally

The Relational Database has many Advantages:

•Simplicity, easy, well defined programming
•Robustness
•Flexibility
•Performance
•Easy scalable up on one server, but problems with scalability down and horizontally
•Compatibility in managing generic data

However! To offer all of these, relational databases have to be incredibly complex internally



SQL Database – Examples

- Commercial
  - IBM DB2
  - Oracle RDMS
  - Microsoft SQL Server
  - Sybase SQL Anywhere
- Open Source (with commercial options)
  - MySQL, Postgres, Ingres

Majority of enterprise and businesses in the world run SQL databases!

It is not hard to find SQL Database Examples

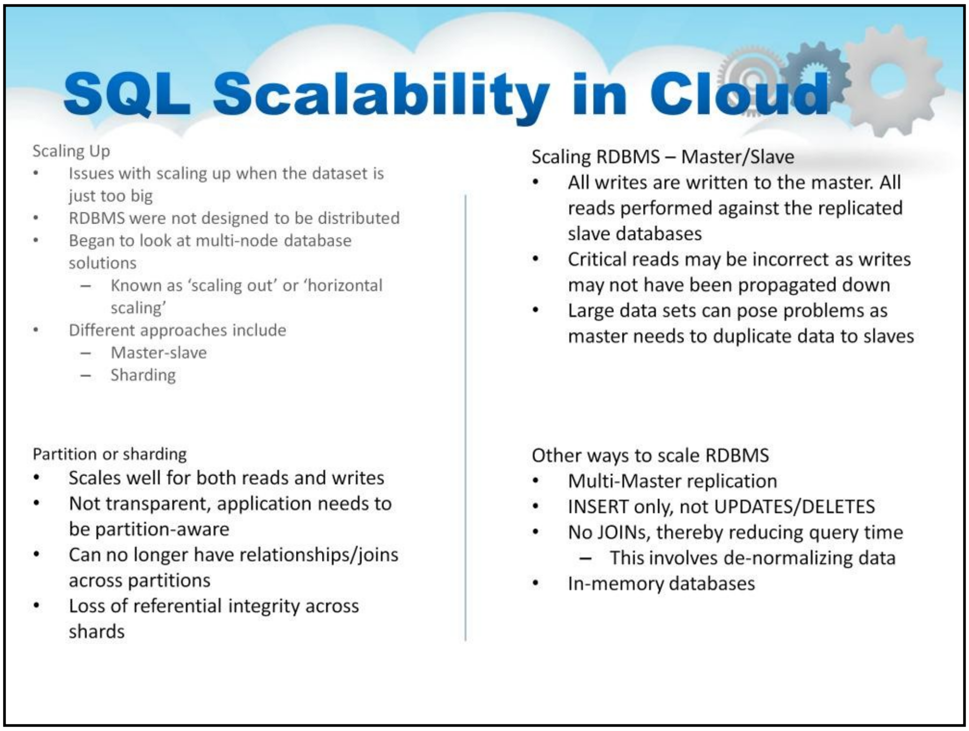
Commercial

•IBM DB2
•Oracle RDMS
•Microsoft SQL Server
•Sybase SQL Anywhere

Open Source (with commercial options)

•MySQL,
•Postgres,
Ingres

Majority of enterprise and businesses in the world run SQL databases!



SQL Scalability in Cloud

Scaling Up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
  - Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include
  - Master-slave
  - Sharding

Partition or sharding

- Scales well for both reads and writes
- Not transparent, application needs to be partition-aware
- Can no longer have relationships/joins across partitions
- Loss of referential integrity across shards

Scaling RDBMS – Master/Slave

- All writes are written to the master. All reads performed against the replicated slave databases
- Critical reads may be incorrect as writes may not have been propagated down
- Large data sets can pose problems as master needs to duplicate data to slaves

Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

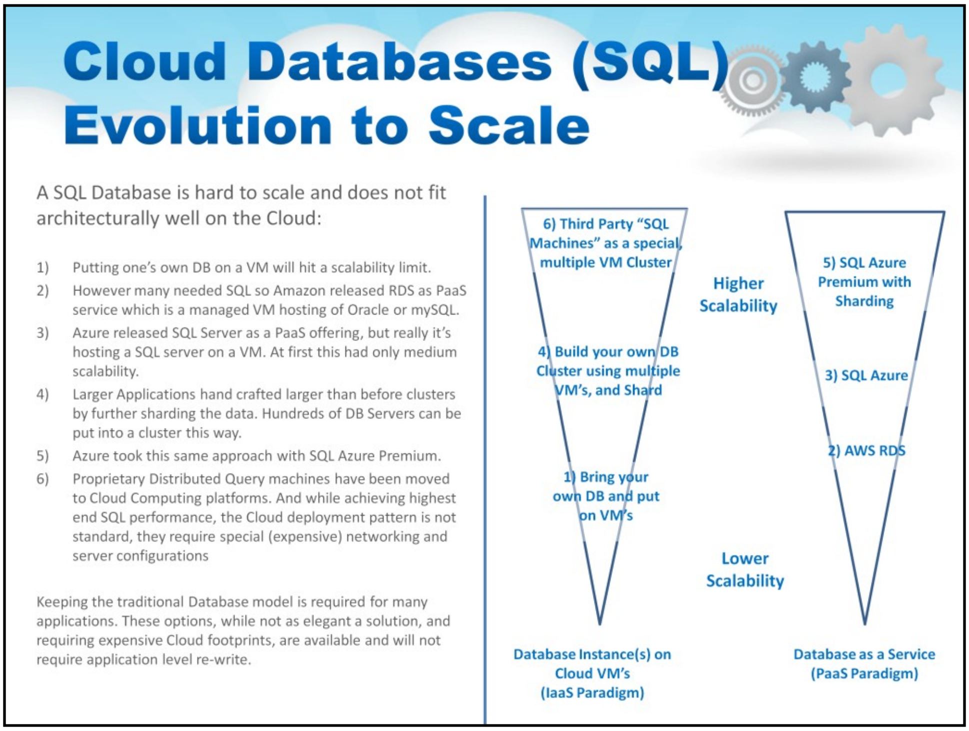
-> Best way to provide ACID and a rich query model is to have the dataset on a single machine.

-> However, there are limits to scaling up (Vertical Scaling).

-> Past a certain point, an organization will find it is cheaper and more feasible to scale out (horizontal scaling) by adding smaller, more inexpensive (relatively) servers rather than investing in a single larger server.

-> A number of different approaches to scaling out (Horizontal Scaling).

-> DBAs began to look at master-slave and sharding as a strategy to overcome some of these issues.



Cloud Databases (SQL) Evolution to Scale

A SQL Database is hard to scale and does not fit architecturally well on the Cloud:

1) Putting one's own DB on a VM will hit a scalability limit.
2) However many needed SQL so Amazon released RDS as PaaS service which is a managed VM hosting of Oracle or mySQL.
3) Azure released SQL Server as a PaaS offering, but really it's hosting a SQL server on a VM. At first this had only medium scalability.
4) Larger Applications hand crafted larger than before clusters by further sharding the data. Hundreds of DB Servers can be put into a cluster this way.
5) Azure took this same approach with SQL Azure Premium.
6) Proprietary Distributed Query machines have been moved to Cloud Computing platforms. And while achieving highest end SQL performance, the Cloud deployment pattern is not standard, they require special (expensive) networking and server configurations

Keeping the traditional Database model is required for many applications. These options, while not as elegant a solution, and requiring expensive Cloud footprints, are available and will not require application level re-write.

6) Third Party "SQL Machines" as a special, multiple VM Cluster

4) Build your own DB Cluster using multiple VM's, and Shard

1) Bring your own DB and put on VM's

Database Instance(s) on Cloud VM's (IaaS Paradigm)

Higher Scalability

Lower Scalability

5) SQL Azure Premium with Sharding

3) SQL Azure

2) AWS RDS

Database as a Service (PaaS Paradigm)

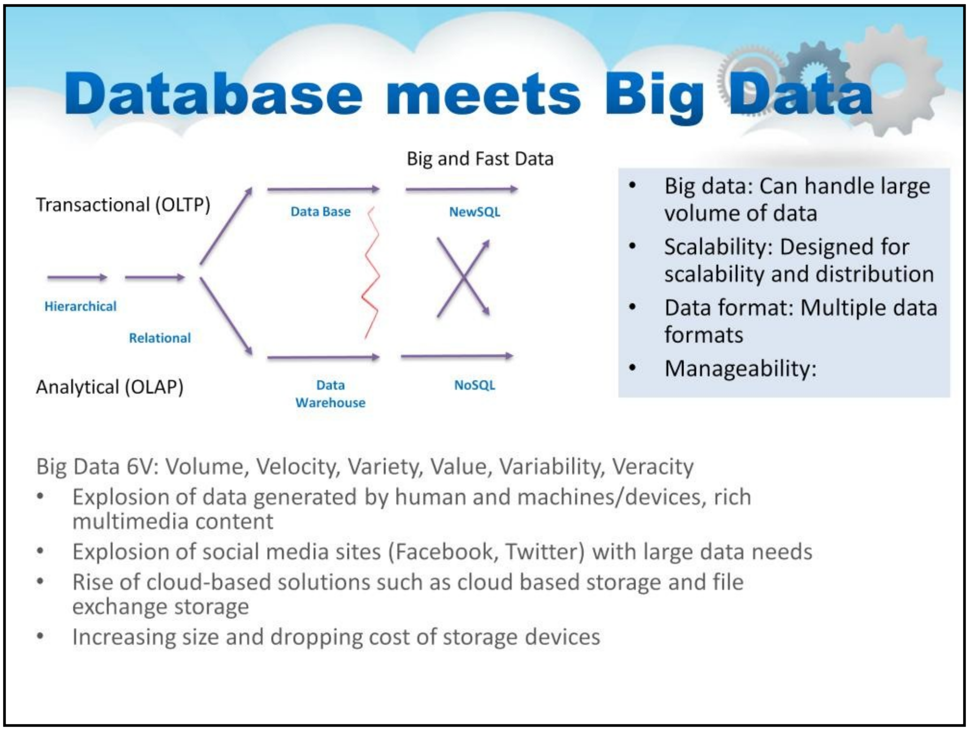
A SQL Database is hard to scale and does not fit architecturally well on the Cloud

This diagram investigates two different scenarios. One of Database Instance(s) on Cloud VM's (IaaS Paradigm), and one of Database as a Service (PaaS Paradigm).

As one goes from requirements of lower scalability to higher scalability, classic DBMS systems run into challenging problems.

This chart indicates exactly what people do when their scalability needs increases, and still try to use the DBMS profile

Essentially, the field of Big Data was coined for when Transaction Systems, and Data Warehouses, ran into troubles with large data and a processing environment set up across a distributed system, like the cloud.



Database meets Big Data

Big and Fast Data

Transactional (OLTP)

Data Base

NewSQL

Hierarchical

Relational

Analytical (OLAP)

Data Warehouse

NoSQL

- Big data: Can handle large volume of data
- Scalability: Designed for scalability and distribution
- Data format: Multiple data formats
- Manageability:

Big Data 6V: Volume, Velocity, Variety, Value, Variability, Veracity

- Explosion of data generated by human and machines/devices, rich multimedia content
- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as cloud based storage and file exchange storage
- Increasing size and dropping cost of storage devices

This slide illustrates the Evolution of Databases, and introduces a simple way to think about OLTP and OLAP and what is happening with Databases on the Cloud

Where the meanings are:

OLTP – Online Transactional Processing
OLAP – Online Analytic Processing

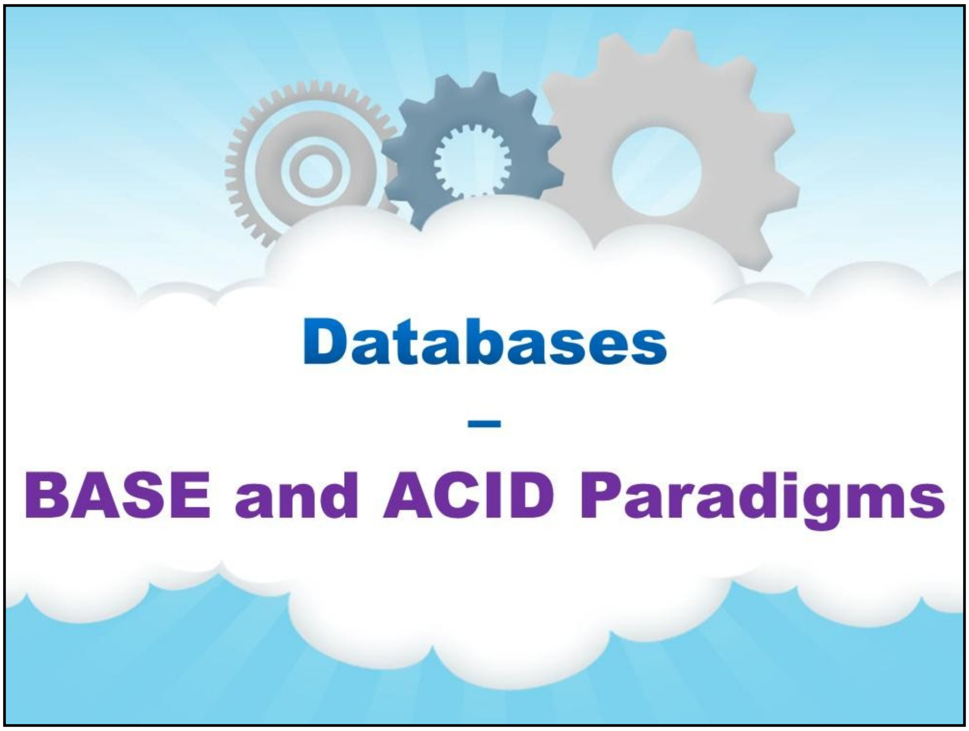
-> Databases could be made to grow by slicing up (or “sharding”) datasets.

-> However, the people with the largest datasets (terrabyte/petabyte) began to realize that sharding was putting a bandage on their issues. The more aggressive thought leaders (Google, Facebook, Twitter) began to explore alternative ways to store data. This became especially true in 2008/2009.

-> These datasets have high read/write rates.

-> With the advent of Amazon S3, a large respected vendor made the statement that maybe it was okay to look at alternative storage solutions other that relational.

-> All of the NoSQL options with the exception of Amazon S3 (Amazon Dynamo) are open-source solutions. This provides a low-cost entry point to ‘kick the tires’.

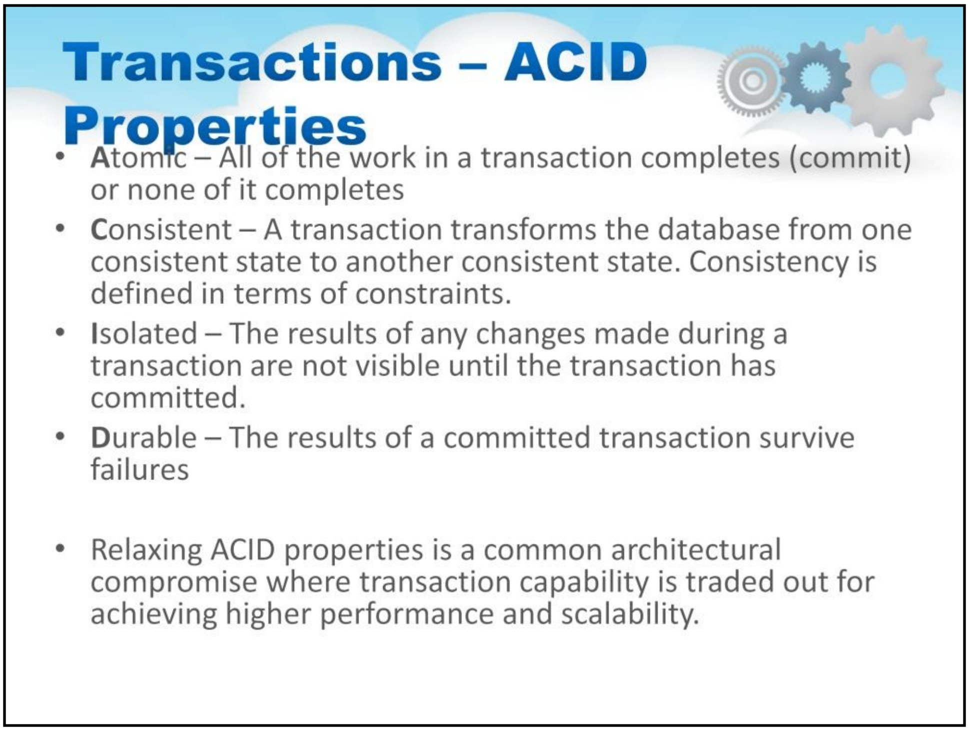


Databases

–

BASE and ACID Paradigms

BASE and ACID Paradigms



Transactions – ACID Properties

- **A**tomic – All of the work in a transaction completes (commit) or none of it completes
- **C**onsistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- **I**solated – The results of any changes made during a transaction are not visible until the transaction has committed.
- **D**urable – The results of a committed transaction survive failures

- Relaxing ACID properties is a common architectural compromise where transaction capability is traded out for achieving higher performance and scalability.

Most Databases support the notion of a transaction. Saying that, many of the “NoSQL Databases” don’t support transactions, across the NoSQL cluster they may have an “eventual consistency” model.

So what does it mean to support a Transaction?

Transactions exhibit ACID Properties where ACID is an acronym as follows:

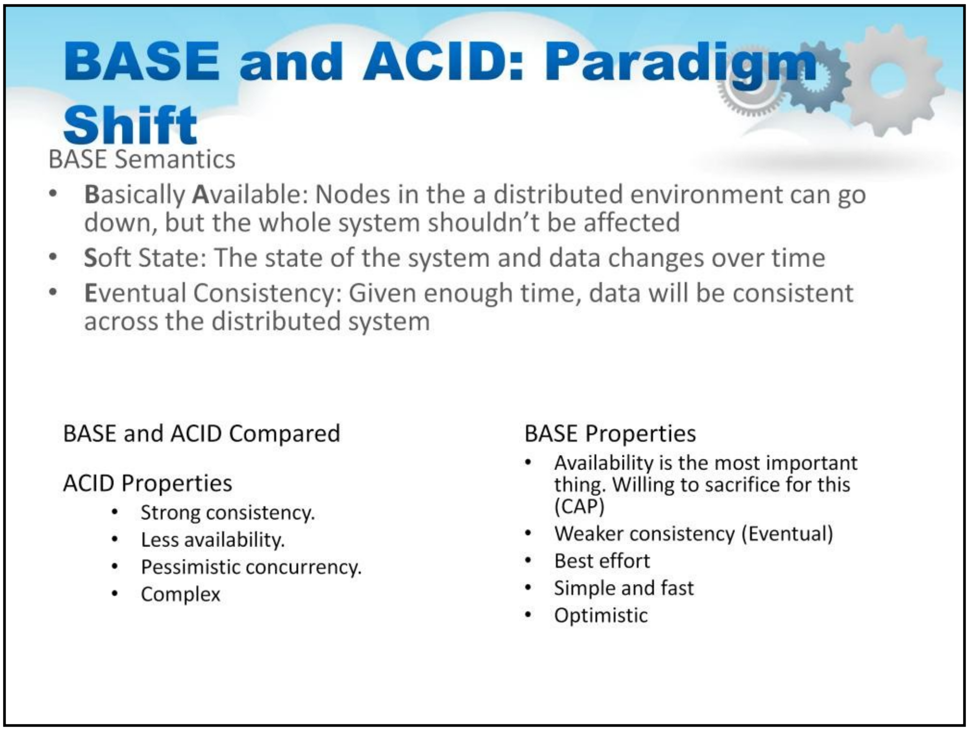
Atomic – All of the work in a transaction completes (commit) or none of it completes

Consistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.

Isolated – The results of any changes made during a transaction are not visible until the transaction has committed.

Durable – The results of a committed transaction survive failures

Relaxing ACID properties is a way some systems compromise transactions to achieve higher performance and scalability.



BASE and ACID: Paradigm Shift

BASE Semantics

- **B**asically **A**vailable: Nodes in the a distributed environment can go down, but the whole system shouldn’t be affected
- **S**oft State: The state of the system and data changes over time
- **E**ventual Consistency: Given enough time, data will be consistent across the distributed system

BASE and ACID Compared

ACID Properties

- Strong consistency.
- Less availability.
- Pessimistic concurrency.
- Complex

BASE Properties

- Availability is the most important thing. Willing to sacrifice for this (CAP)
- Weaker consistency (Eventual)
- Best effort
- Simple and fast
- Optimistic

Now to look more closely at the BASE and ACID Paradigm Shift
We studied ACID properties for implementing transactions. Not accidentally named as an opposite to ACID, the acronym BASE is defined as:

- **B**asically **A**vailable: Nodes in the a distributed environment can go down, but the whole system shouldn’t be affected
- **S**oft State: The state of the system and data changes over time
- **E**ventual Consistency: Given enough time, data will be consistent across the distributed system

It is meant to be an “different behavior” of a database to allow for more performance, scale, geographic distribution of data, and so on

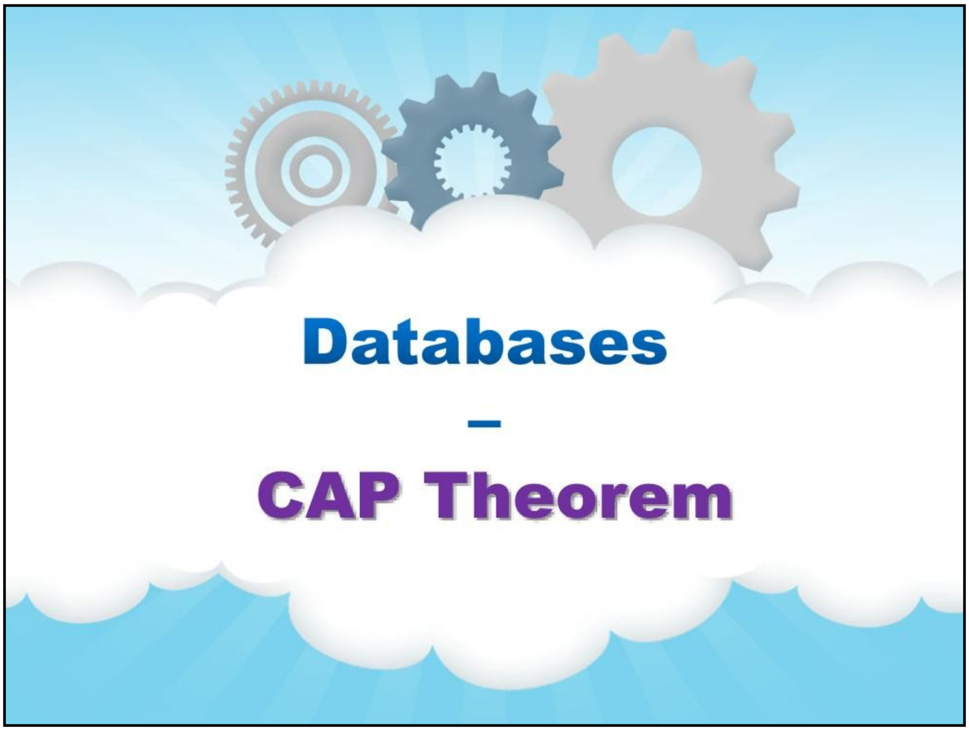
Let’s compare BASE and ACID

ACID is characterized by the following properties:

- Strong consistency.
- Less availability.
- Pessimistic concurrency.
- Complex

BASE is characterized by the other properties:

- Availability is the most important thing
- Weaker consistency (Eventual)
- Best effort
- Simple and fast
- Optimistic

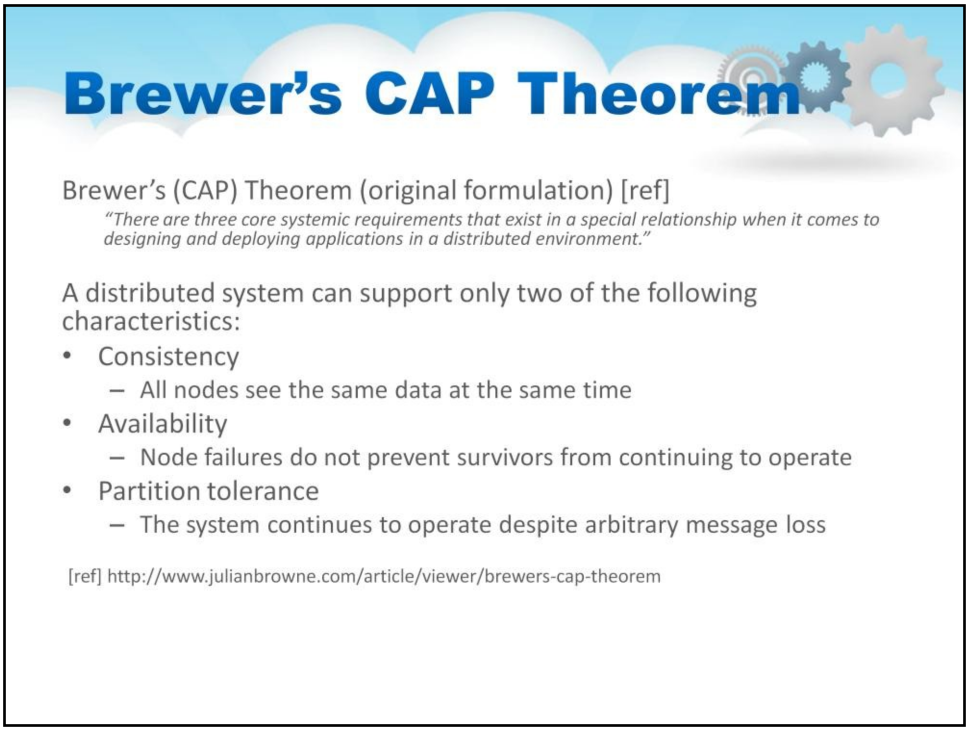


Databases

–

CAP Theorem

CAP Theorem



Brewer's CAP Theorem

Brewer's (CAP) Theorem (original formulation) [ref]

> *"There are three core systemic requirements that exist in a special relationship when it comes to designing and deploying applications in a distributed environment."*

A distributed system can support only two of the following characteristics:

- Consistency
  - All nodes see the same data at the same time
- Availability
  - Node failures do not prevent survivors from continuing to operate
- Partition tolerance
  - The system continues to operate despite arbitrary message loss

[ref] http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

Why is it hard to implement things like Transactions on Databases in the Cloud? Because a Cloud is a distributed system,.

A Professor named Brewer was studying that area of implementing distributed databases and published the CAP Theorem

He asserted that

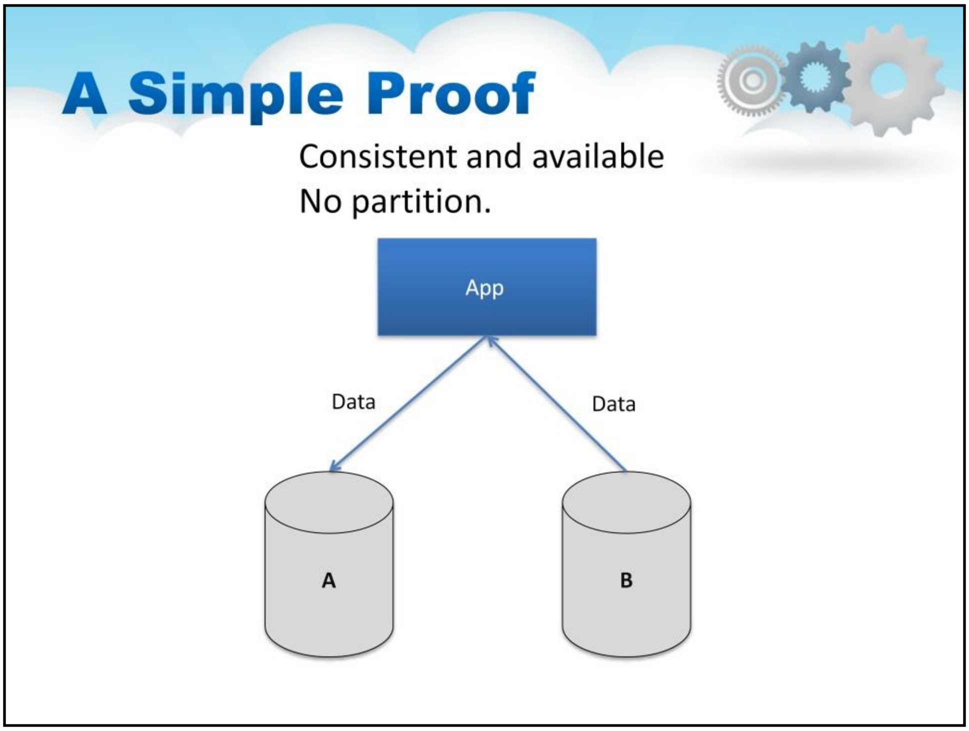
"There are three core systemic requirements that exist in a special relationship when it comes to designing and deploying applications in a distributed environment."

He then asserted that

A distributed system can support only two of the following characteristics (in any one design):

Consistency
    All nodes see the same data at the same time
Availability
    Node failures do not prevent survivors from continuing to operate
Partition tolerance
    The system continues to operate despite arbitrary message loss

The acronym for this is CAP



A Simple Proof

Consistent and available
No partition.

App

Data

Data

A

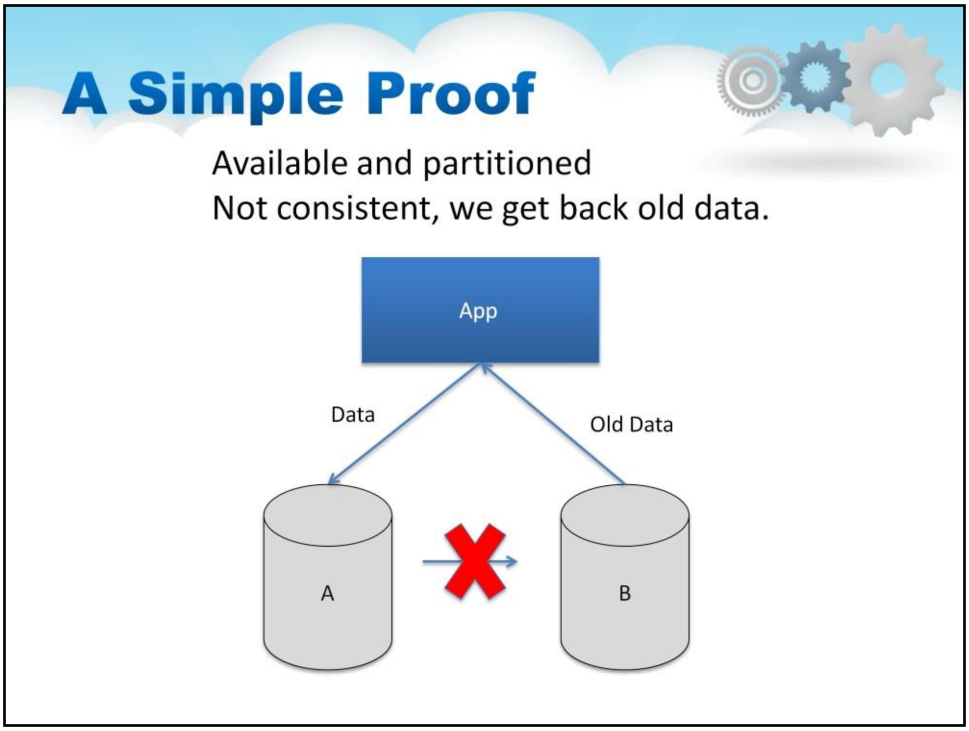
B

Let us step through some examples as illustrated in the slide.

First we look at an app connected to data that resides in two storage services. All data is in both places.

Because the data is online and connected to the app in both places, the app sees data consistent and available no matter which storage service fulfils the read or write actions

I do not need to partition the data, since it is all online to my app.



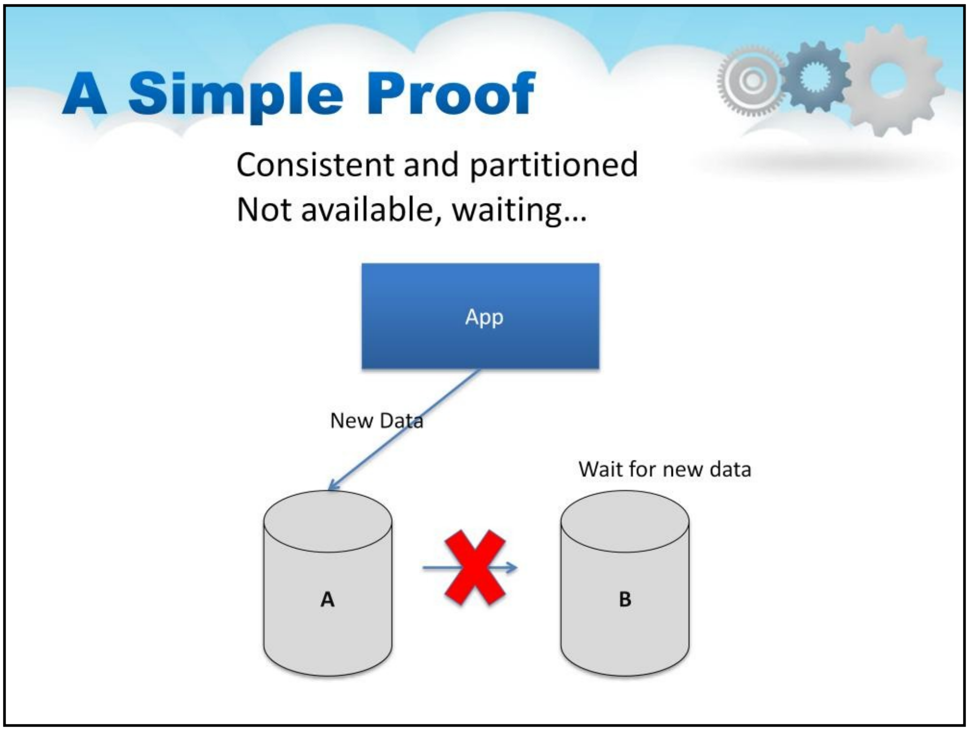
A Simple Proof

Available and partitioned

Not consistent, we get back old data.

In this example, a partition is made, some of the data is old now. If one accesses and is served by the service which accidentally has old data, that’s what one will get – old data

We tried to be available and partitioned, but we get not consistent out of that.

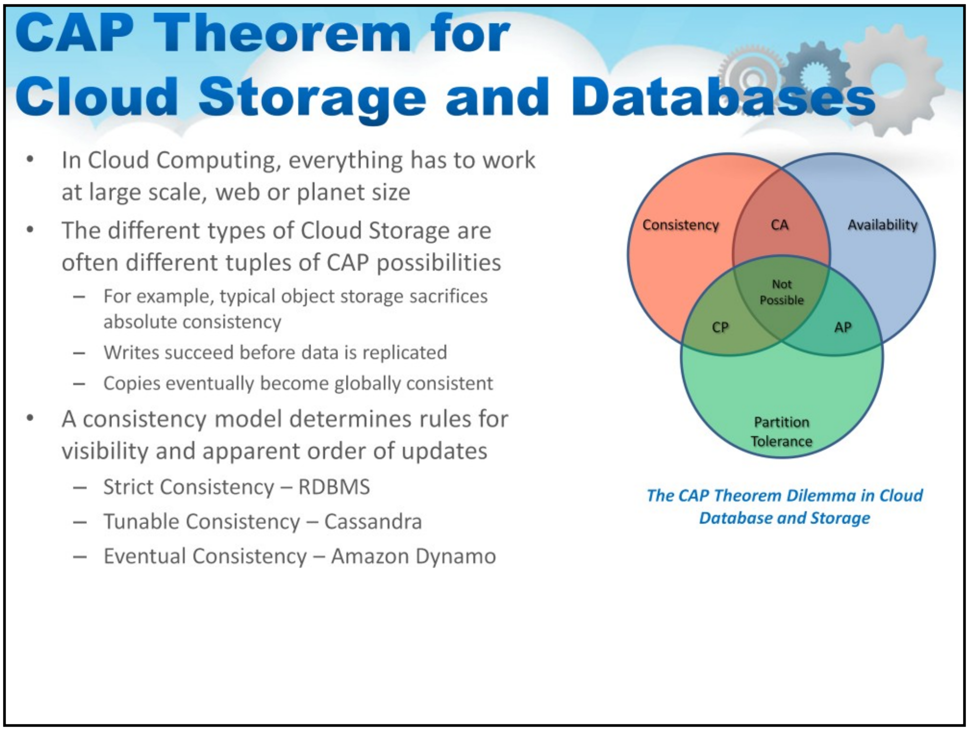


A Simple Proof

Consistent and partitioned
Not available, waiting…

In this case, as illustrated in the slide, if one insists on consistency one has to wait for all the storage services to synchronize. This means during the synchronization process, the data is not available (e.g., one is waiting).

It is now easy to see why consistency, partitioning, and availability are related.



CAP Theorem for Cloud Storage and Databases

- In Cloud Computing, everything has to work at large scale, web or planet size
- The different types of Cloud Storage are often different tuples of CAP possibilities
  - For example, typical object storage sacrifices absolute consistency
  - Writes succeed before data is replicated
  - Copies eventually become globally consistent
- A consistency model determines rules for visibility and apparent order of updates
  - Strict Consistency – RDBMS
  - Tunable Consistency – Cassandra
  - Eventual Consistency – Amazon Dynamo

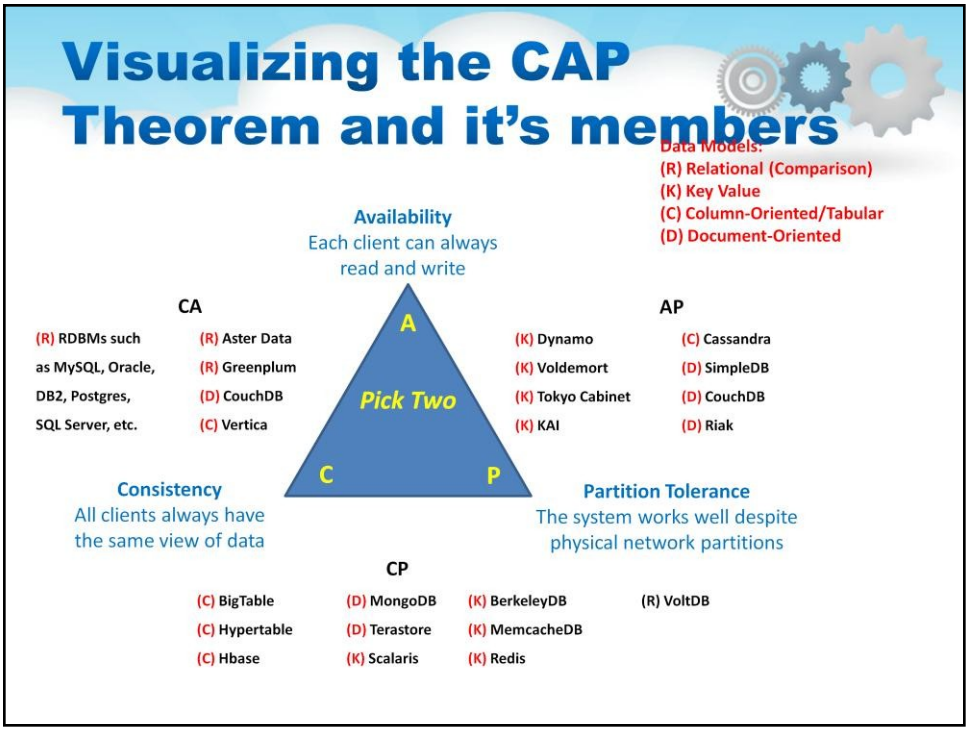
The CAP Theorem Dilemma in Cloud Database and Storage

This is the essence of the challenge that databases have on cloud platforms with large datasets. It is called the Distributed (Cloud) Databases Theorem, or more precisely the CAP Theorem.

It states that In Cloud Computing, everything has to work at planet sized, large scale. Areas which seem to be at odds with each other in a large database deployment are:
•Consistency: all clients should see the current data regardless of updates or deletes
•Availability: the system continues to operate as expected even with node failures
•Partition Tolerance: the system continues to operate as expected despite network or message system failures

Many developers were wondering how to fix this problem.
• Partitionability: divide nodes into small groups that can see other groups, but they can't see everyone.
• Consistency: write a value and then you read the value you get the same value back. In a partitioned system there are windows where that's not true.
• Availability: may not always be able to write or read. The system will say you can't write because it wants to keep the system consistent. To scale you have to partition, so you are left with choosing either high consistency or high availability for a particular system. You must find the right overlap of availability and consistency.

Choose a specific approach based on the needs of the service.



Visualizing the CAP Theorem and it's members

Data Models:
(R) Relational (Comparison)
(K) Key Value
(C) Column-Oriented/Tabular
(D) Document-Oriented

Availability
Each client can always read and write

CA

(R) RDBMs such as MySQL, Oracle, DB2, Postgres, SQL Server, etc.

(R) Aster Data
(R) Greenplum
(D) CouchDB
(C) Vertica

AP

(K) Dynamo
(K) Voldemort
(K) Tokyo Cabinet
(K) KAI

(C) Cassandra
(D) SimpleDB
(D) CouchDB
(D) Riak

Consistency
All clients always have the same view of data

Partition Tolerance
The system works well despite physical network partitions

CP

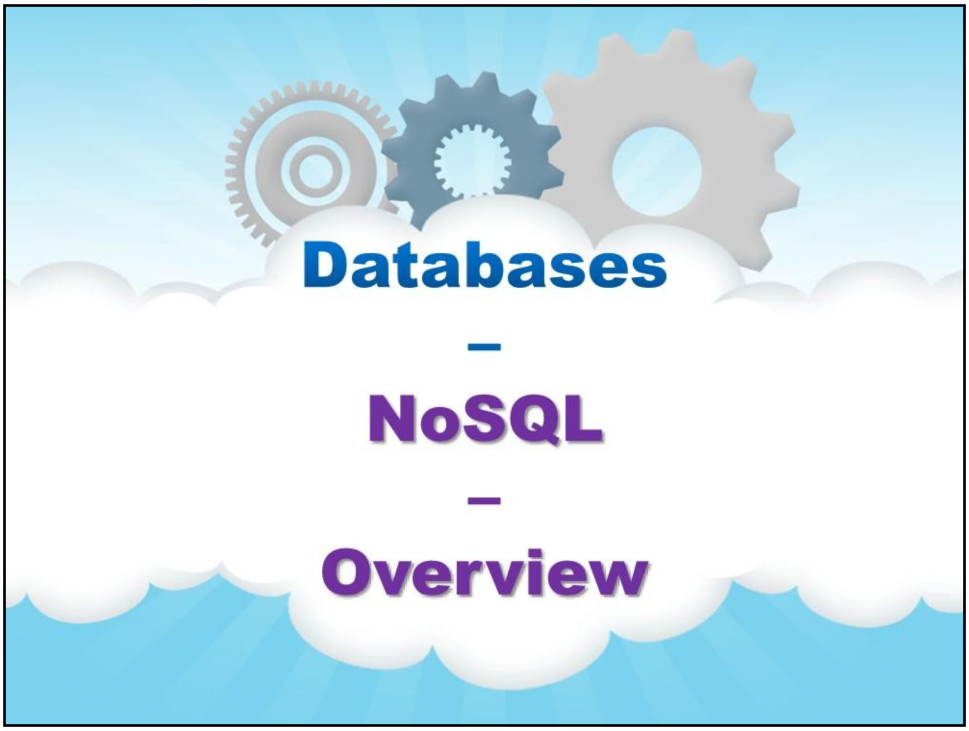
(C) BigTable
(C) Hypertable
(C) Hbase

(D) MongoDB
(D) Terastore
(K) Scalaris

(K) BerkeleyDB
(K) MemcacheDB
(K) Redis

(R) VoltDB

The diagram on this slide illustrates the CAP Theorem visibly. It also categorizes commercial and open source implementations into one of the three CAP Theorem categories.



Databases

–

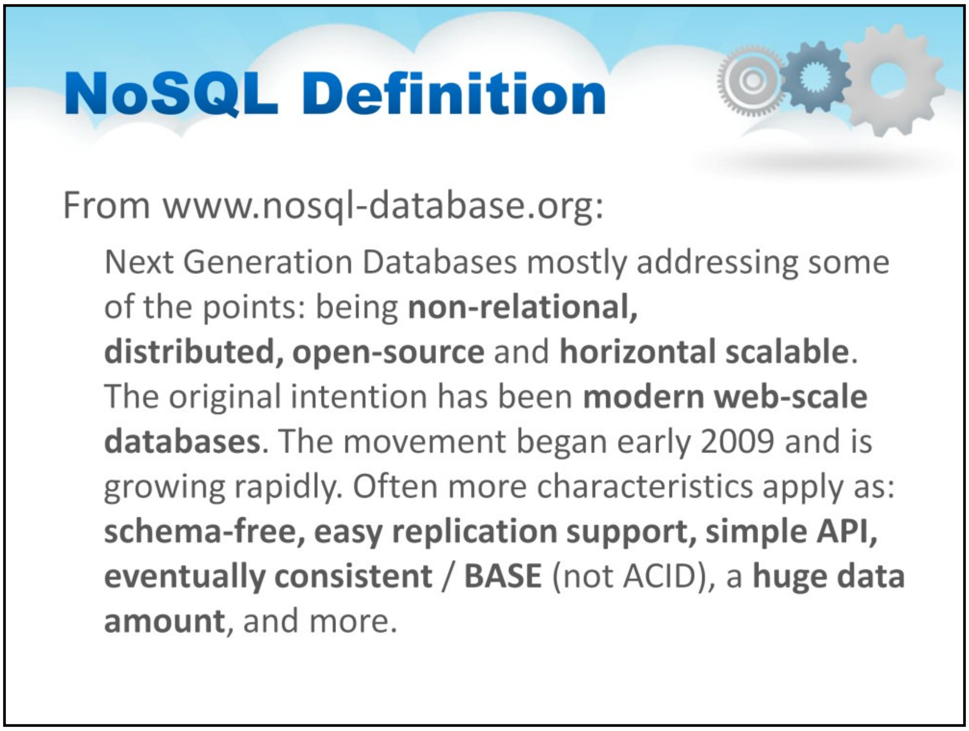
NoSQL

–

Overview

NoSQL

Overview



NoSQL Definition

From www.nosql-database.org:

> Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontal scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge data amount**, and more.

NoSQL Definition

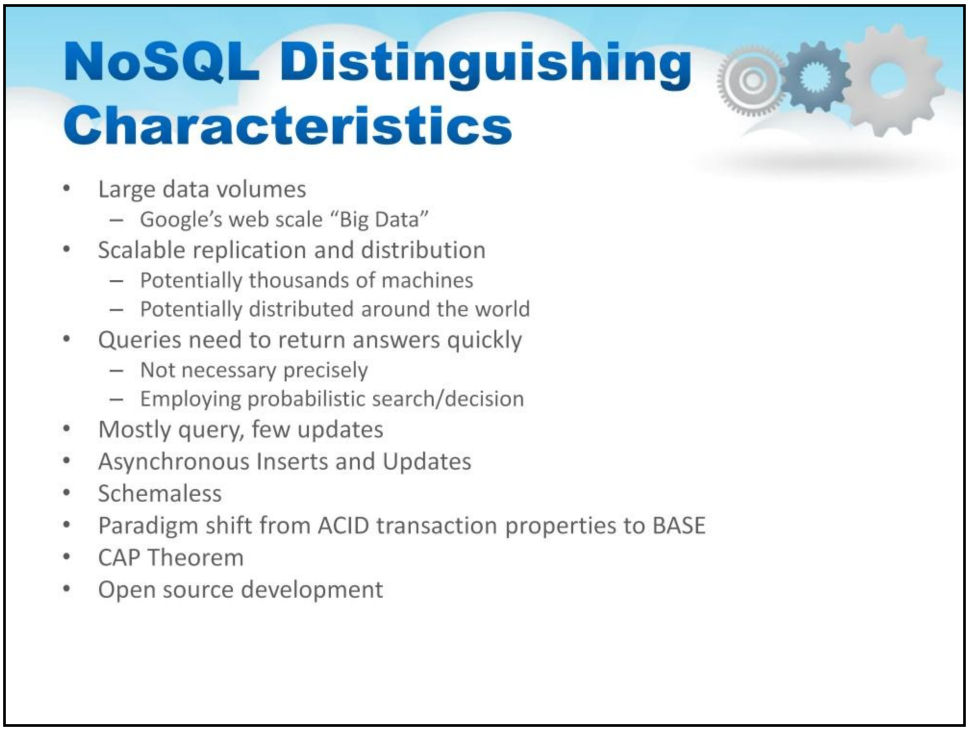
Next Generation Databases mostly addressing some of the points:

being **non-relational, distributed, open-source** and **horizontal scalable**.

The original intention has been **modern web-scale databases**.

The movement began early 2009 and is growing rapidly.

Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge data amount**, and more.



NoSQL Distinguishing Characteristics

- Large data volumes
  - Google’s web scale “Big Data”
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
  - Not necessary precisely
  - Employing probabilistic search/decision
- Mostly query, few updates
- Asynchronous Inserts and Updates
- Schemaless
- Paradigm shift from ACID transaction properties to BASE
- CAP Theorem
- Open source development

Another way to look at NoSQL is by Distinguishing Characteristics

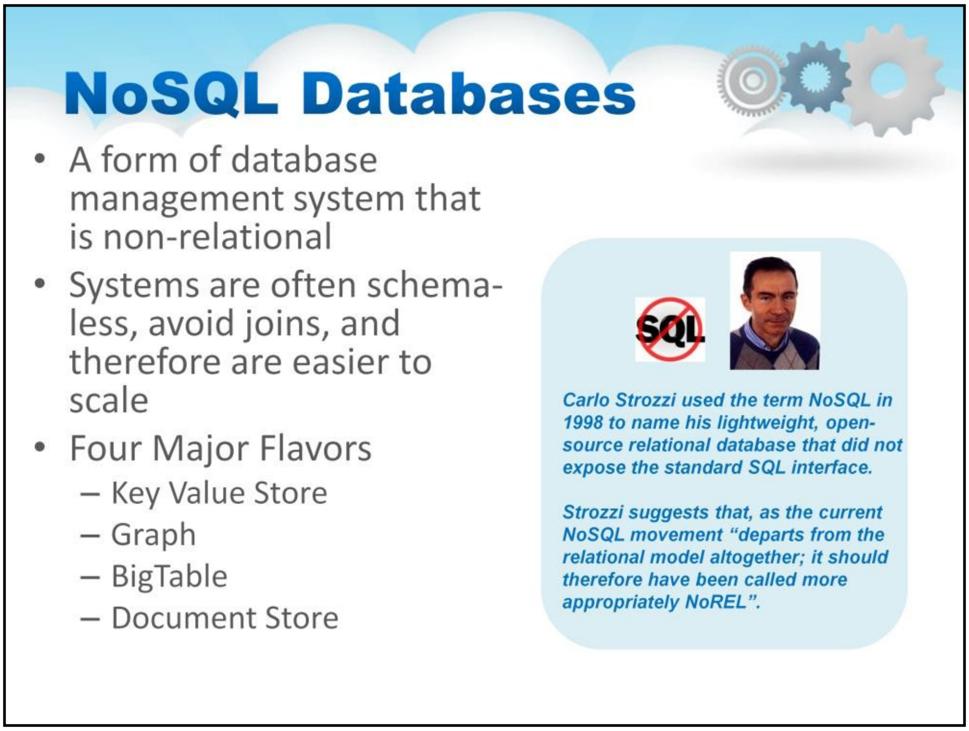
Large data volumes
    Google’s web scale “Big Data”

Scalable replication and distribution
    Potentially thousands of machines
    Potentially distributed around the world

Queries need to return answers quickly
    Not necessary precisely
    Employing probabilistic search/decision

Mostly query, few updates
Asynchronous Inserts and Updates
Schemaless
Paradigm shift from ACID transaction properties to BASE
CAP Theorem
Open source development

We will cover several of these subject in more detail next



NoSQL Databases

- A form of database management system that is non-relational
- Systems are often schema-less, avoid joins, and therefore are easier to scale
- Four Major Flavors
  - Key Value Store
  - Graph
  - BigTable
  - Document Store

Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface.

Strozzi suggests that, as the current NoSQL movement "departs from the relational model altogether; it should therefore have been called more appropriately NoREL".

You will note many of the databases in the CAP Theorem quantification are called "NoSQL Databases".

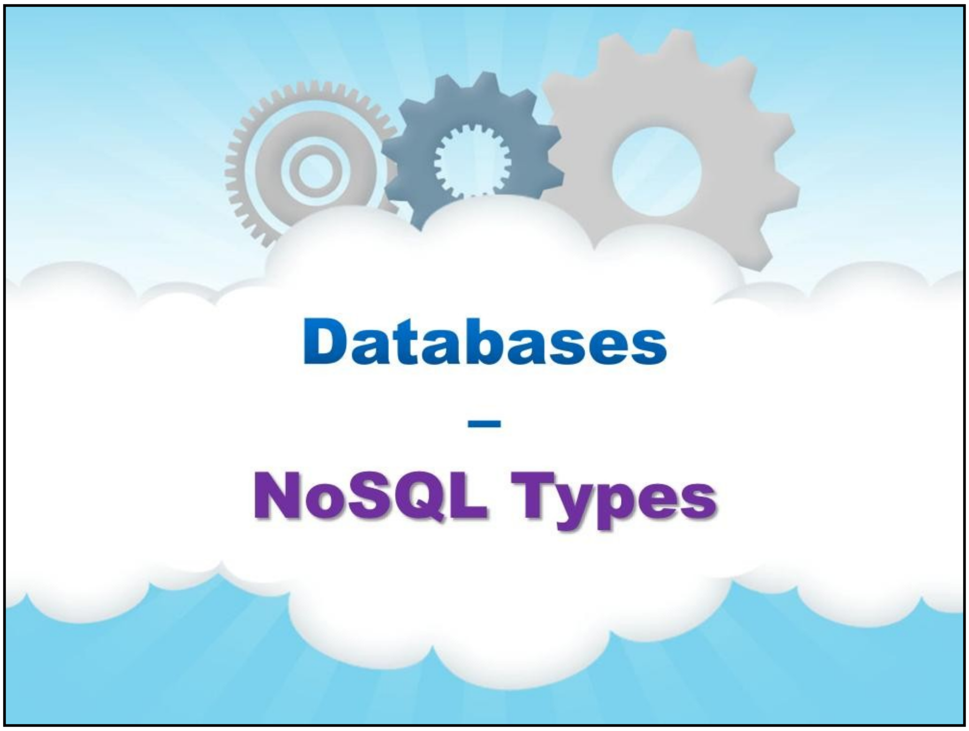
This curious term refers to A form of database management system that is non-relational. These Systems are often schema-less, avoid joins, and therefore are easier to scale.

As a bit of history, Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface.

Strozzi suggests that, as the current NoSQL movement "departs from the relational model altogether; it should therefore have been called more appropriately NoREL".

There are Four Major Flavors

Key Value Store
Graph
BigTable
Document Store

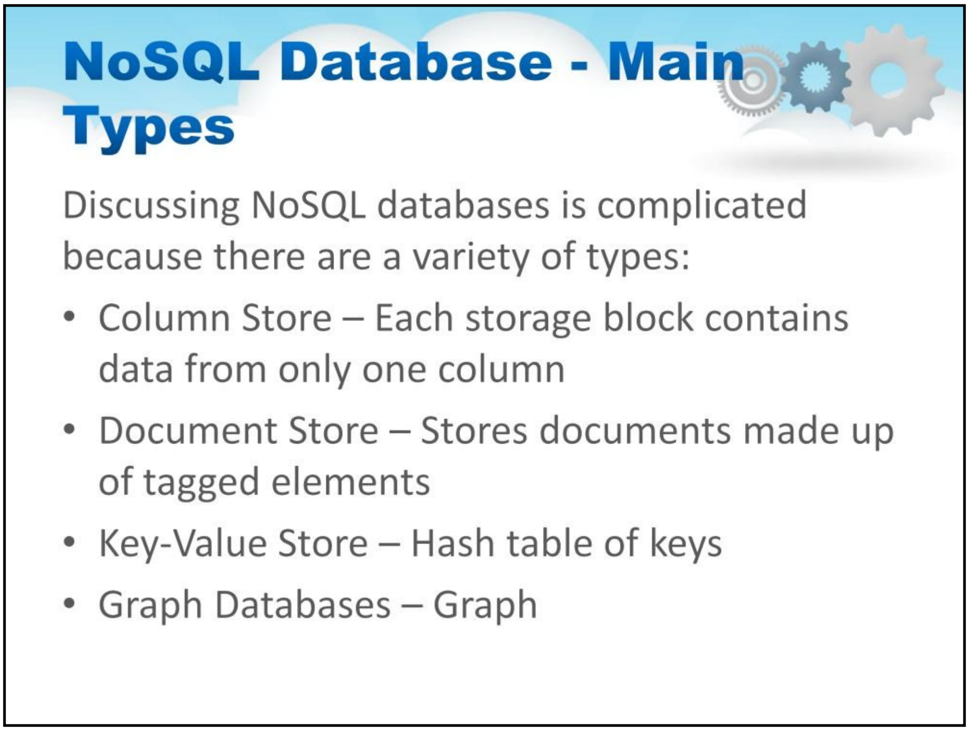


Databases

–

NoSQL Types

NoSQL Types



NoSQL Database - Main Types

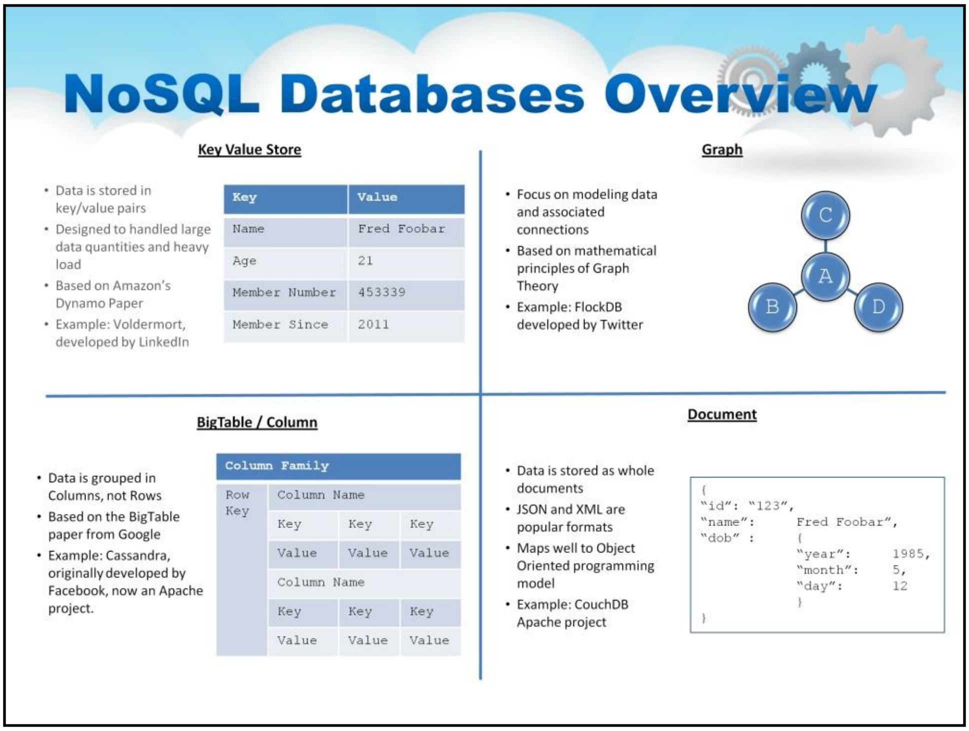
Discussing NoSQL databases is complicated because there are a variety of types:

- Column Store – Each storage block contains data from only one column
- Document Store – Stores documents made up of tagged elements
- Key-Value Store – Hash table of keys
- Graph Databases – Graph

Next we will dive into NoSQL Databases – first we look at major types

Discussing NoSQL databases is complicated because there are a variety of types:

Column Store – Each storage block contains data from only one column
Document Store – Stores documents made up of tagged elements
Key-Value Store – Hash table of keys
Graph Databases – Graph



NoSQL Databases Overview

Key Value Store

- Data is stored in key/value pairs
- Designed to handled large data quantities and heavy load
- Based on Amazon's Dynamo Paper
- Example: Voldermort, developed by LinkedIn

Key	Value
Name	Fred Foobar
Age	21
Member Number	453339
Member Since	2011

Graph

- Focus on modeling data and associated connections
- Based on mathematical principles of Graph Theory
- Example: FlockDB developed by Twitter

BigTable / Column

- Data is grouped in Columns, not Rows
- Based on the BigTable paper from Google
- Example: Cassandra, originally developed by Facebook, now an Apache project.

Column Family			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Document

- Data is stored as whole documents
- JSON and XML are popular formats
- Maps well to Object Oriented programming model
- Example: CouchDB Apache project

```
{
"id": "123",
"name":      Fred Foobar",
"dob" :      {
             "year":     1985,
             "month":    5,
             "day":      12
             }
}
```

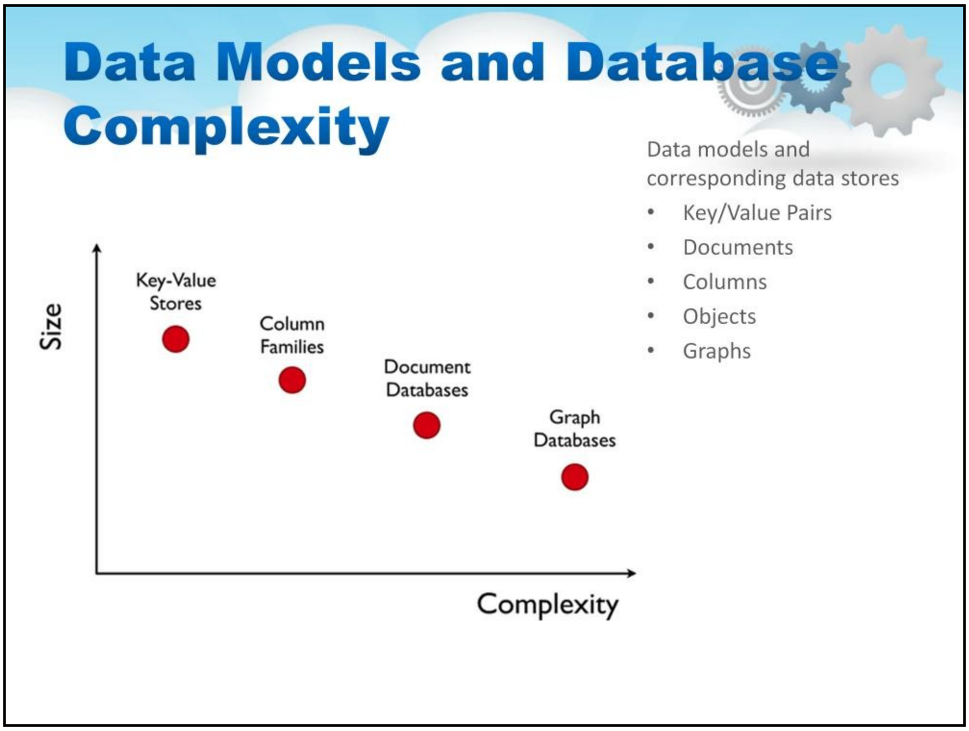
This slide shows the four major types of noRel databases along with how the data structures inside them turn out looking.

In Key Value Store Data is stored in key/value pairs
Designed to handled large data quantities and heavy load
Based on Amazon's Dynamo Paper
Example: Voldermort, developed by LinkedIn

In Graph the Focus is on modeling data and associated connections
Based on mathematical principles of Graph Theory
Example: FlockDB developed by Twitter

In BigTable / Column Data is grouped in Columns, not Rows
Based on the BigTable paper from Google
Example: Cassandra, originally developed by Facebook, now an Apache project.

Finally in Document Data is stored as whole documents
JSON and XML are popular formats
Maps well to Object Oriented programming model
Example: CouchDB Apache project



Data Models and Database Complexity

Data models and corresponding data stores

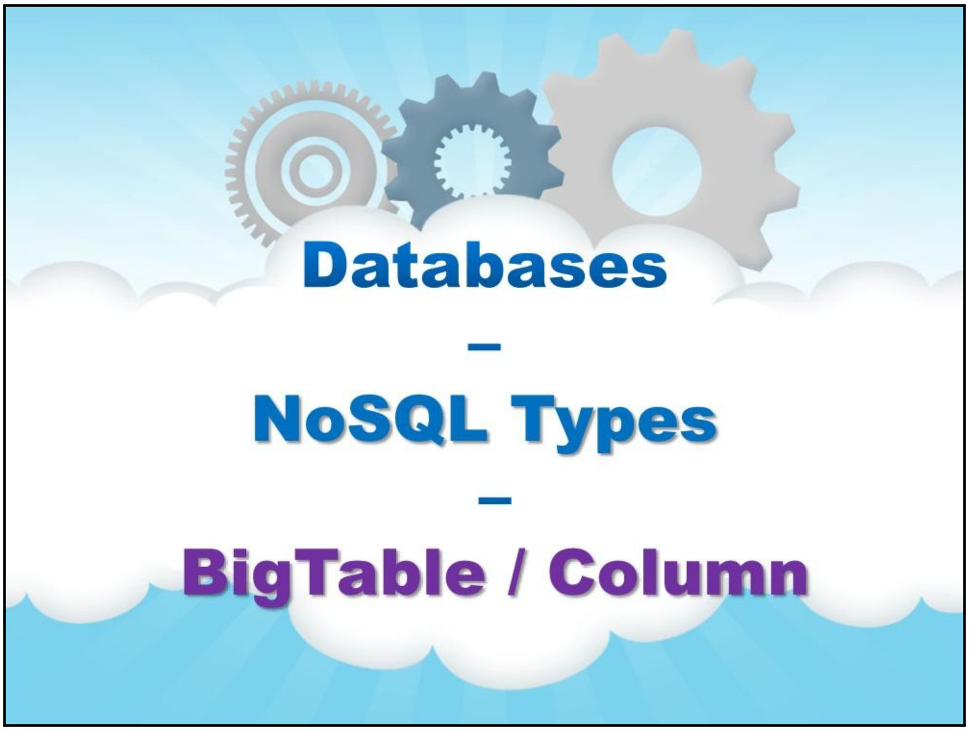
- Key/Value Pairs
- Documents
- Columns
- Objects
- Graphs

One way to classify these different approaches is illustrated on the slide.

The different system each have a slightly different data size and complexity tradeoff.

Which means, the more complex of an analysis one wants to do on a data set, the smaller the data set will be that that data base can understand.

Each technique fits a certain tradeoff point – this is illustrated.



Databases

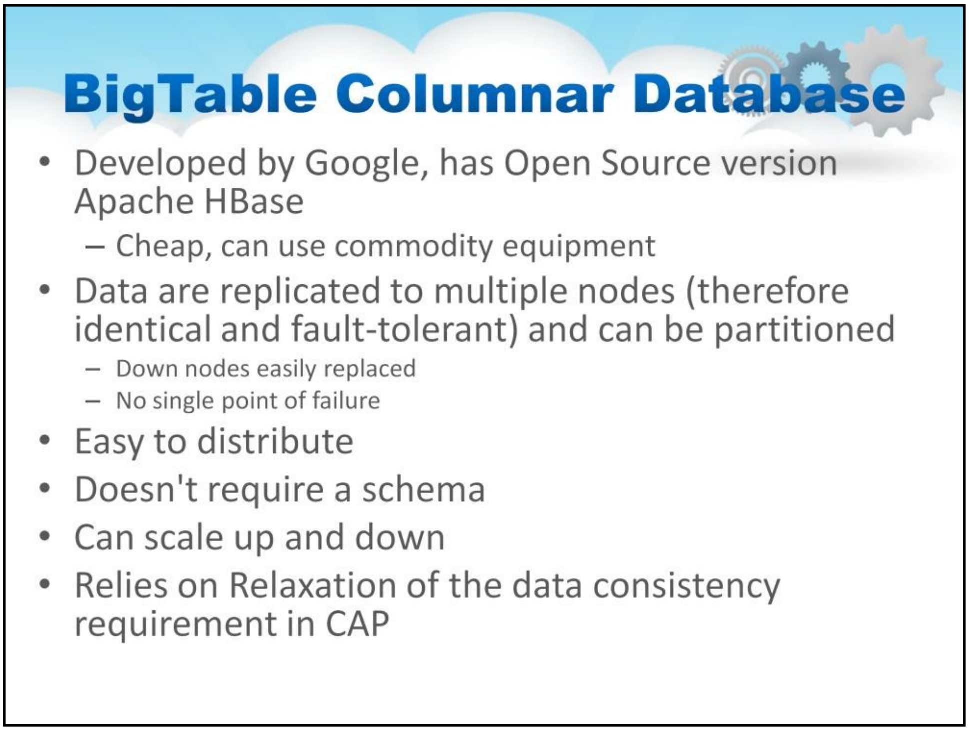
–

NoSQL Types

–

BigTable / Column

BigTable / Column



BigTable Columnar Database

- Developed by Google, has Open Source version Apache HBase
  - Cheap, can use commodity equipment
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Doesn't require a schema
- Can scale up and down
- Relies on Relaxation of the data consistency requirement in CAP

First we will focus on one of the first NoSQL solutions, based n early work done by Google. It is called BigTable.

There is an Open Source version of this called Apache HBase

Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned

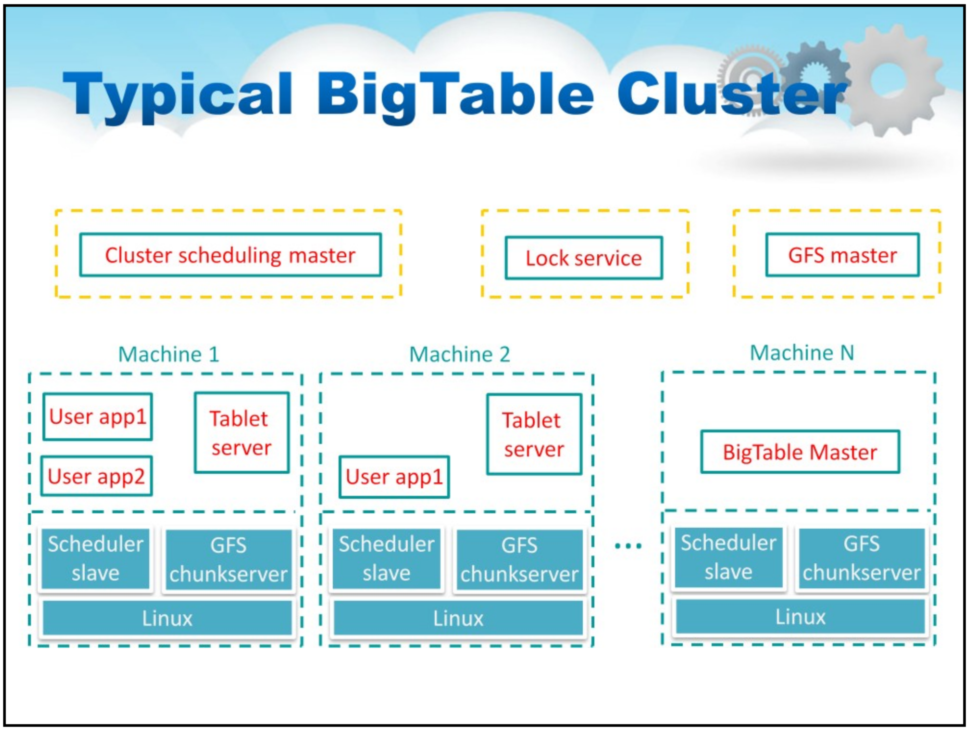
Down nodes easily replaced
No single point of failure

BigTable is Easy to distribute
Doesn’t require a schema
And Can scale up and down

BigTable Relies on Relaxation of the data consistency requirement in CAP

-> As the data is written, the latest version is on at least one node. The data is then versioned/replicated to other nodes within the system.

-> Eventually, the same version is on all nodes.



Typical BigTable Cluster

Cluster scheduling master

Lock service

GFS master

Machine 1

User app1

User app2

Tablet server

Scheduler slave

GFS chunkserver

Linux

Machine 2

Tablet server

User app1

Scheduler slave

GFS chunkserver

Linux

…

Machine N

BigTable Master

Scheduler slave

GFS chunkserver

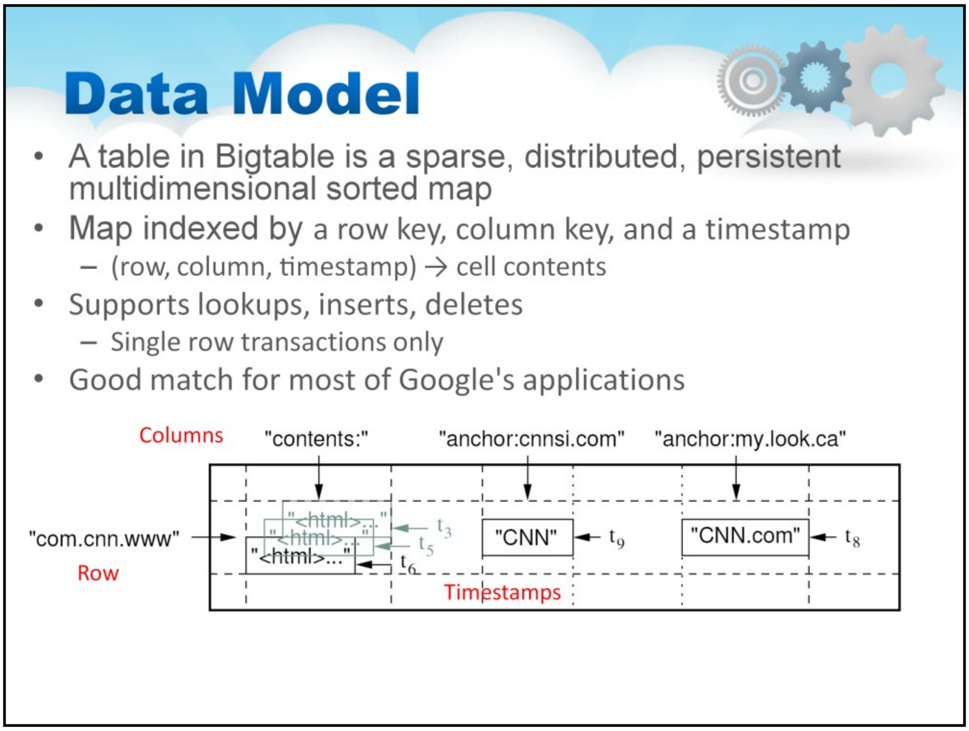
Linux

Here is an example of how BigTable is implemented in a Cluster. Please see the illustration in the slide.

Note that there are one or more control nodes (or machines, or VMs) running Scheduling, Locking, and the File System

The other N number of nodes (or machines, or VM's) are set up to work with the scheduler and the Filesystem, each working on a part ("chunk") of the data

Tablets are explained a little later.



Data Model

- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
  - (row, column, timestamp) → cell contents
- Supports lookups, inserts, deletes
  - Single row transactions only
- Good match for most of Google's applications

A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map

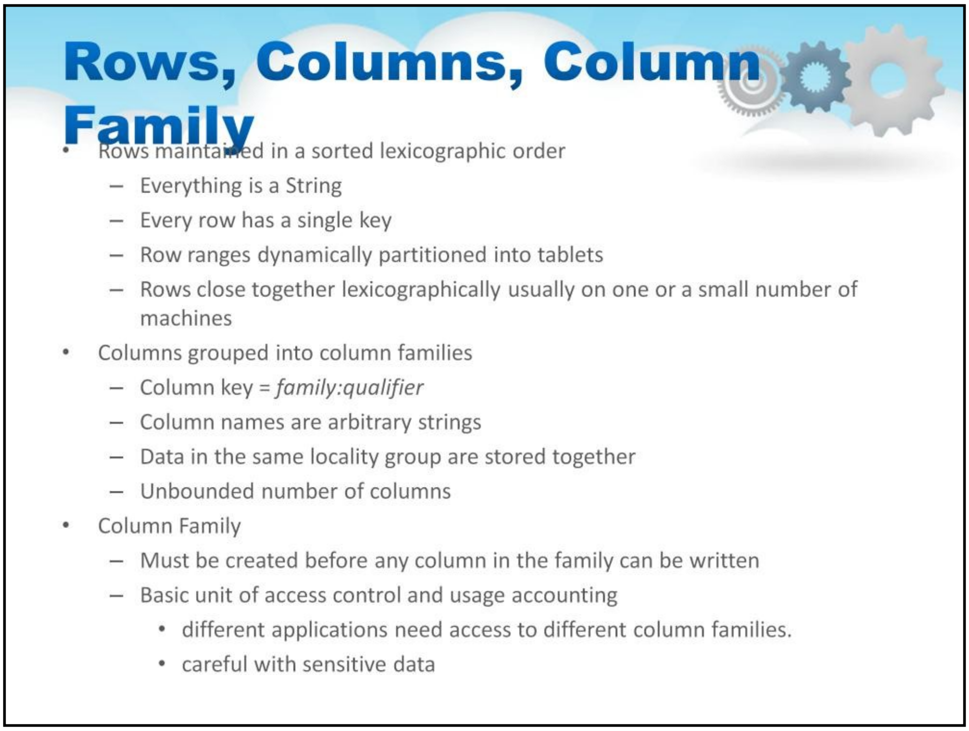
This Map indexed by a row key, column key, and a timestamp

Timestamps are Used to store different versions of data in a cell

New writes default to current time, but timestamps for writes can also be set explicitly by clients

There are several Lookup options. Examples are:

“Return most recent K values”
“Return all values in timestamp range (or all values)”



Rows, Columns, Column Family

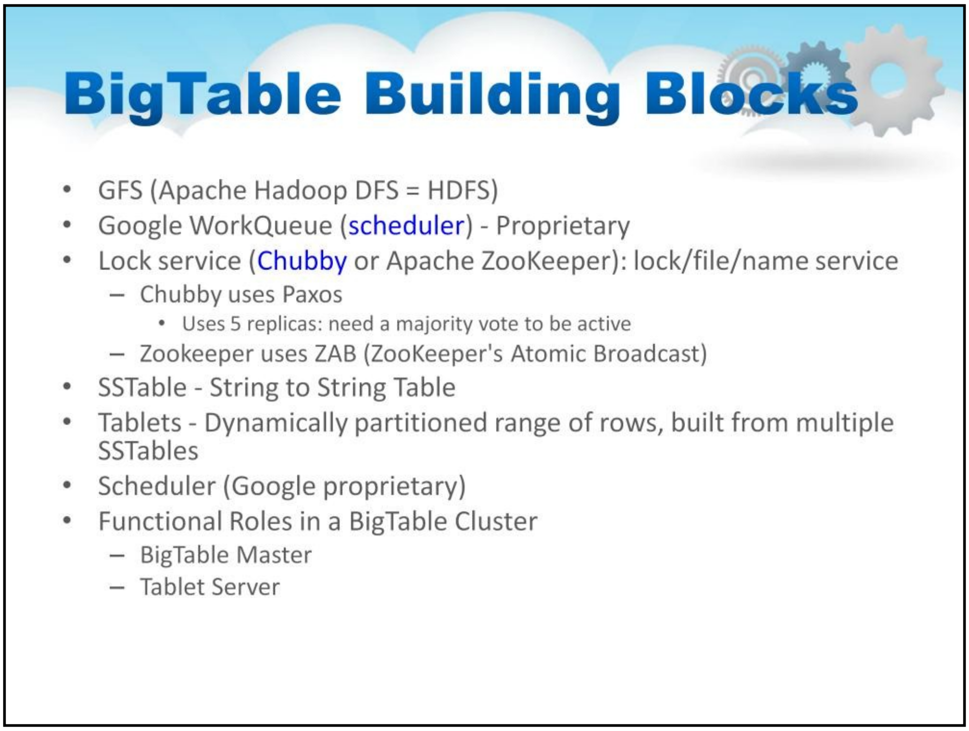
- Rows maintained in a sorted lexicographic order
  - Everything is a String
  - Every row has a single key
  - Row ranges dynamically partitioned into tablets
  - Rows close together lexicographically usually on one or a small number of machines
- Columns grouped into column families
  - Column key = *family:qualifier*
  - Column names are arbitrary strings
  - Data in the same locality group are stored together
  - Unbounded number of columns
- Column Family
  - Must be created before any column in the family can be written
  - Basic unit of access control and usage accounting
    - different applications need access to different column families.
    - careful with sensitive data

In BigTable, Rows, Columns, Column Family have special structure

Rows are maintained in a sorted lexicographic order
  Everything is a String
  Every row has a single key
  Row ranges dynamically partitioned into tablets
  Rows close together lexicographically usually on one or a small number of machines

Columns are grouped into column families
  Column key = *family:qualifier*
  Column names are arbitrary strings
  Data in the same locality group are stored together
  Unbounded number of columns

Column Families Must be created before any column in the family can be written
  They are the Basic unit of access control and usage accounting



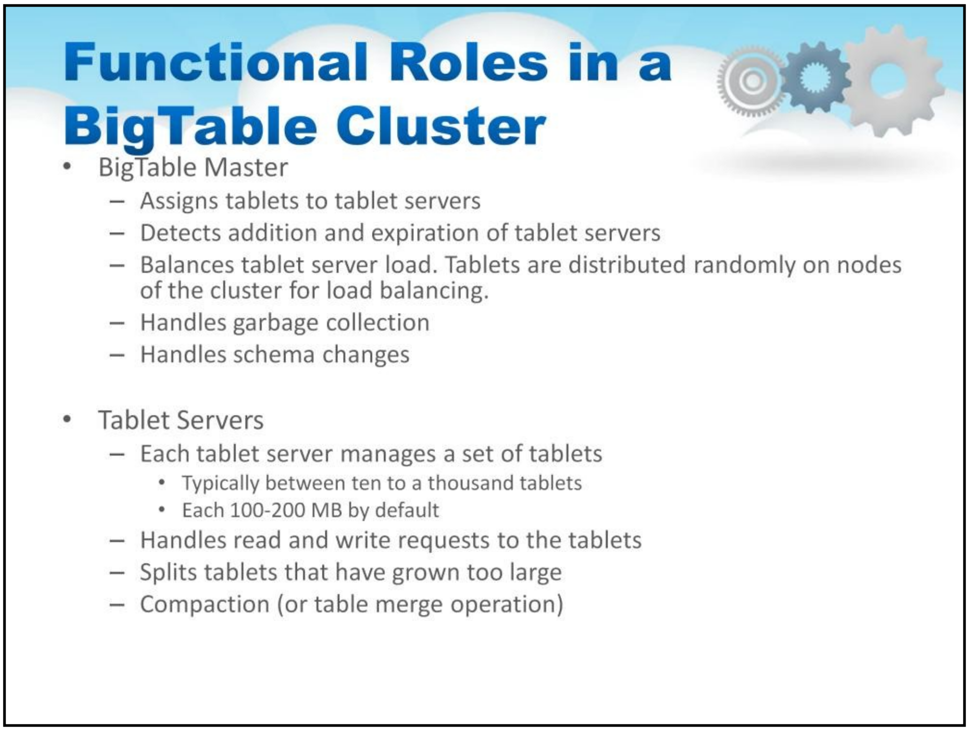
BigTable Building Blocks

- GFS (Apache Hadoop DFS = HDFS)
- Google WorkQueue (scheduler) - Proprietary
- Lock service (Chubby or Apache ZooKeeper): lock/file/name service
  - Chubby uses Paxos
    - Uses 5 replicas: need a majority vote to be active
  - Zookeeper uses ZAB (ZooKeeper's Atomic Broadcast)
- SSTable - String to String Table
- Tablets - Dynamically partitioned range of rows, built from multiple SSTables
- Scheduler (Google proprietary)
- Functional Roles in a BigTable Cluster
  - BigTable Master
  - Tablet Server

BigTable requires a collection of modules to form a complete solution

The building blocks are

A filesystem – Google GFS (Apache Hadoop DFS = HDFS)
A Work Queue - Google WorkQueue (scheduler) - Proprietary
A Lock service (Chubby or Apache ZooKeeper): lock/file/name service
Chubby uses Paxos
Uses 5 replicas: need a majority vote to be active
Zookeeper uses ZAB (ZooKeeper's Atomic Broadcast)
An SSTable - String to String Table
Tablets - Dynamically partitioned range of rows, built from multiple SSTables
A Scheduler (Google proprietary)



Functional Roles in a BigTable Cluster

- BigTable Master
  - Assigns tablets to tablet servers
  - Detects addition and expiration of tablet servers
  - Balances tablet server load. Tablets are distributed randomly on nodes of the cluster for load balancing.
  - Handles garbage collection
  - Handles schema changes
- Tablet Servers
  - Each tablet server manages a set of tablets
    - Typically between ten to a thousand tablets
    - Each 100-200 MB by default
  - Handles read and write requests to the tablets
  - Splits tablets that have grown too large
  - Compaction (or table merge operation)

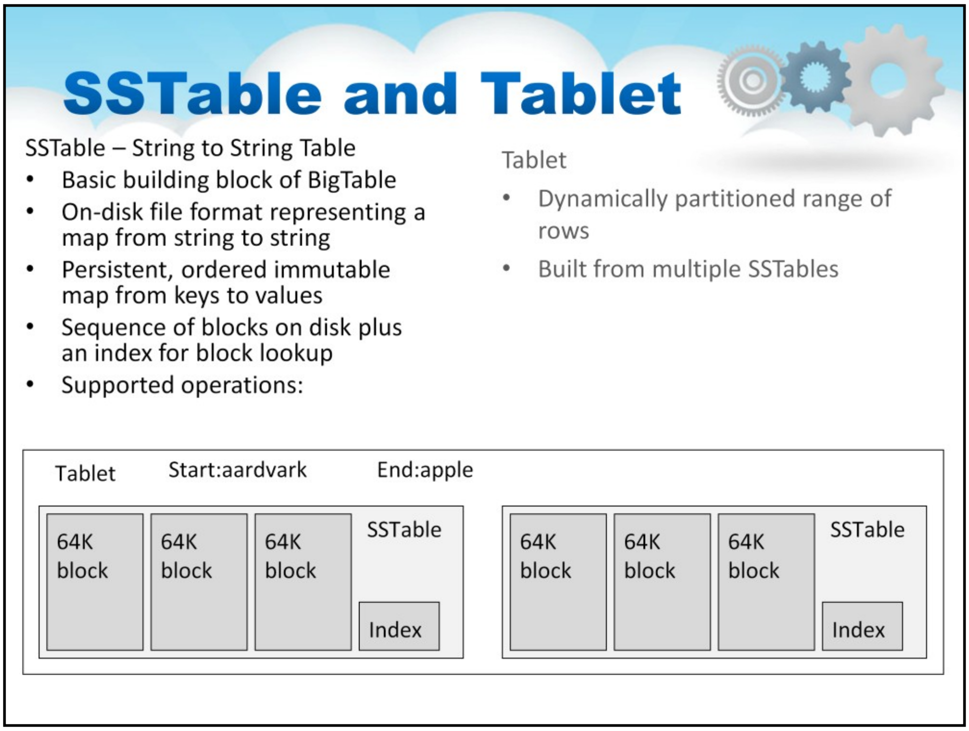
The Functional Roles in a BigTable Cluster are as follows:

The BigTable Master

Assigns tablets to tablet servers
Detects addition and expiration of tablet servers
Balances tablet server load. Tablets are distributed randomly on nodes of the cluster for load balancing.
Handles garbage collection
Handles schema changes

The Tablet Servers

Each tablet server manages a set of tablets
Typically between ten to a thousand tablets
Each 100-200 MB by default
Handles read and write requests to the tablets
Splits tablets that have grown too large
Compaction (or table merge operation)



SSTable and Tablet

SSTable – String to String Table

- Basic building block of BigTable
- On-disk file format representing a map from string to string
- Persistent, ordered immutable map from keys to values
- Sequence of blocks on disk plus an index for block lookup
- Supported operations:

Tablet

- Dynamically partitioned range of rows
- Built from multiple SSTables

Tablet Start:aardvark End:apple

64K block 64K block 64K block SSTable Index

64K block 64K block 64K block SSTable Index

A Closer Look at the SSTable and the Tablet

SSTable – String to String Table Basic building block of BigTable

On-disk file format representing a map from string to string Persistent, ordered immutable map from keys to values

Stored in GFS

Sequence of blocks on disk plus an index for block lookup Can be completely mapped into memory

Supported operations:

- Look up value associated with key
- Iterate key/value pairs within a key range

Tablet

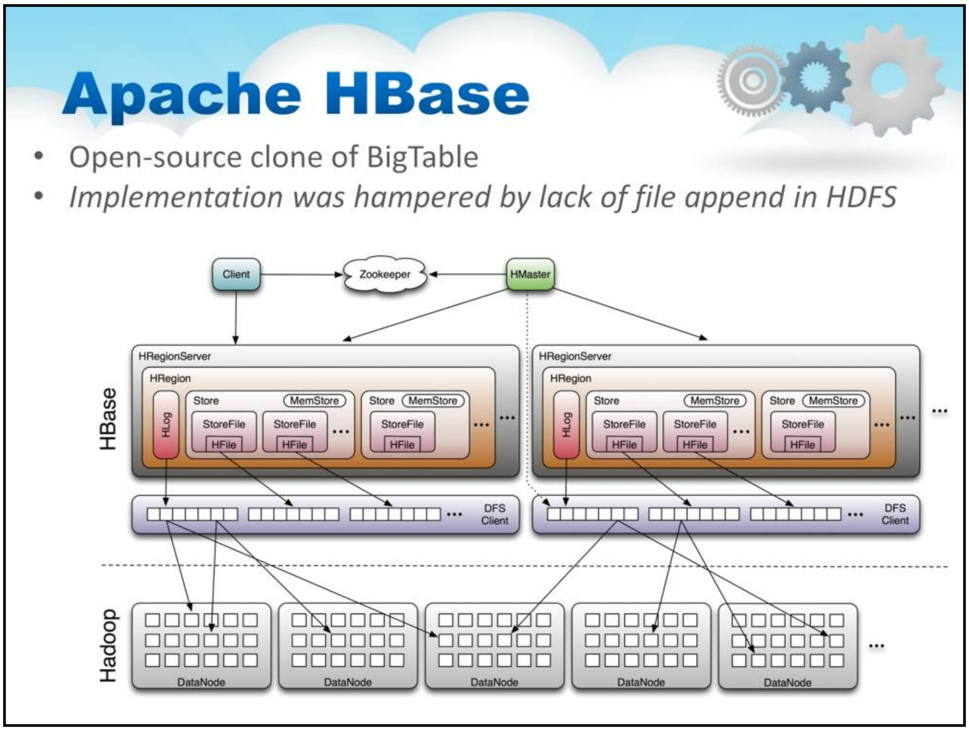
Dynamically partitioned range of rows Built from multiple SSTables

Distributed over tablet servers Unit of load balancing

Tablets split and merge automatically based on size and load or manually

Clients can choose row keys to achieve locality

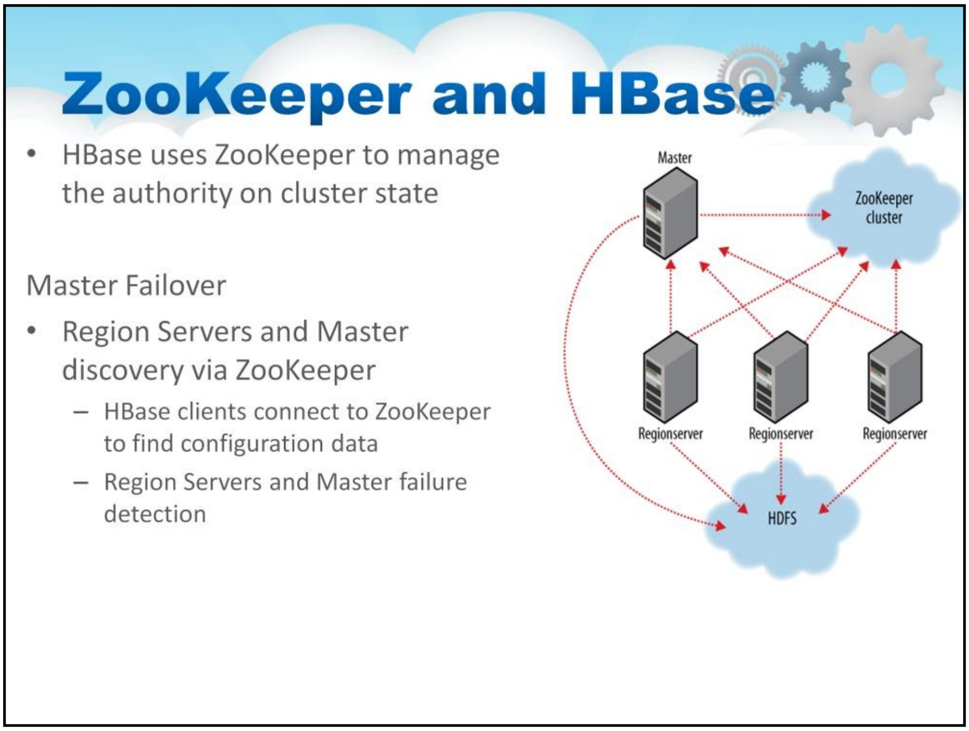
The slide has a detailed illustration which shows ore details on these components as well as how they fit together.



Apache HBase

- Open-source clone of BigTable
- *Implementation was hampered by lack of file append in HDFS*

As Mentioned, Apache Hbase is Open-source clone of BigTable
Initially, it’s Implementation was hampered by lack of file append in HDFS
Workarounds have been posted to this issue.



ZooKeeper and HBase

- HBase uses ZooKeeper to manage the authority on cluster state

Master Failover

- Region Servers and Master discovery via ZooKeeper
  - HBase clients connect to ZooKeeper to find configuration data
  - Region Servers and Master failure detection

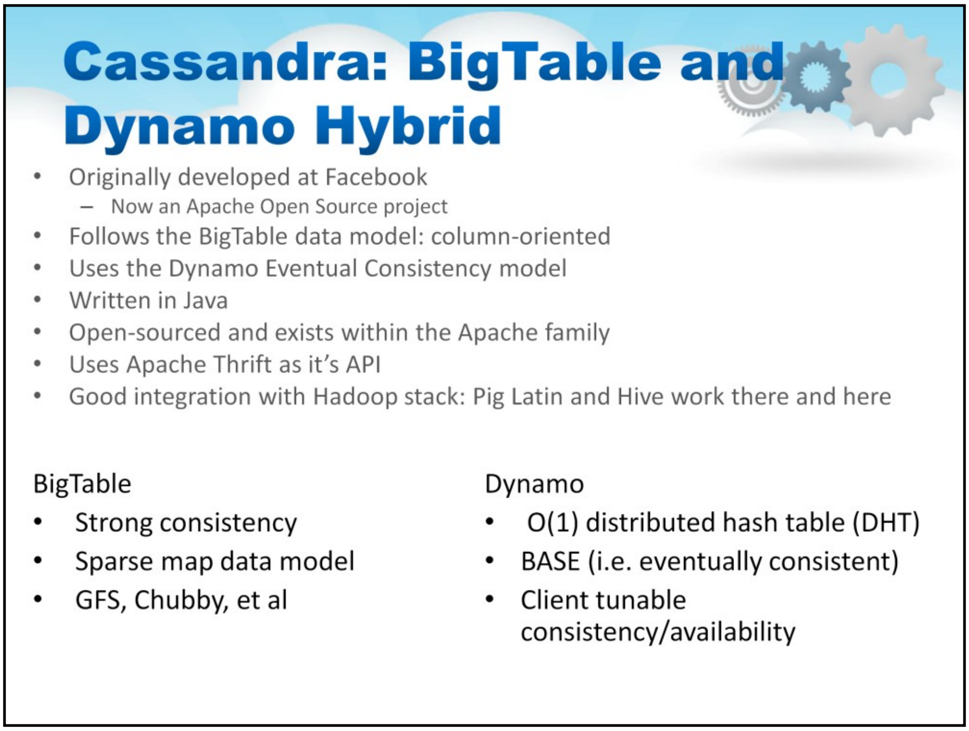
ZooKeeper is a tool for clustering of Hbase

The slide has an illustration of a ZooKeeper Cluster showing Region Servers and HDFS.

HBase uses ZooKeeper to manage the authority on cluster state

The Region Servers and Master discovery are managed by ZooKeeper

HBase clients connect to ZooKeeper to find configuration data



Cassandra: BigTable and Dynamo Hybrid

- Originally developed at Facebook
  - Now an Apache Open Source project
- Follows the BigTable data model: column-oriented
- Uses the Dynamo Eventual Consistency model
- Written in Java
- Open-sourced and exists within the Apache family
- Uses Apache Thrift as it's API
- Good integration with Hadoop stack: Pig Latin and Hive work there and here

BigTable

- Strong consistency
- Sparse map data model
- GFS, Chubby, et al

Dynamo

- O(1) distributed hash table (DHT)
- BASE (i.e. eventually consistent)
- Client tunable consistency/availability

Cassandra gained very large popularity as an implementation of BigTable

It utilizes a DHT based system much like the Dynamo system Amazon describes that controls many of the modules of AWS

Cassandra was Originally developed at Facebook

Now it is an Apache Open Source project

It Follows the BigTable data model: column-oriented

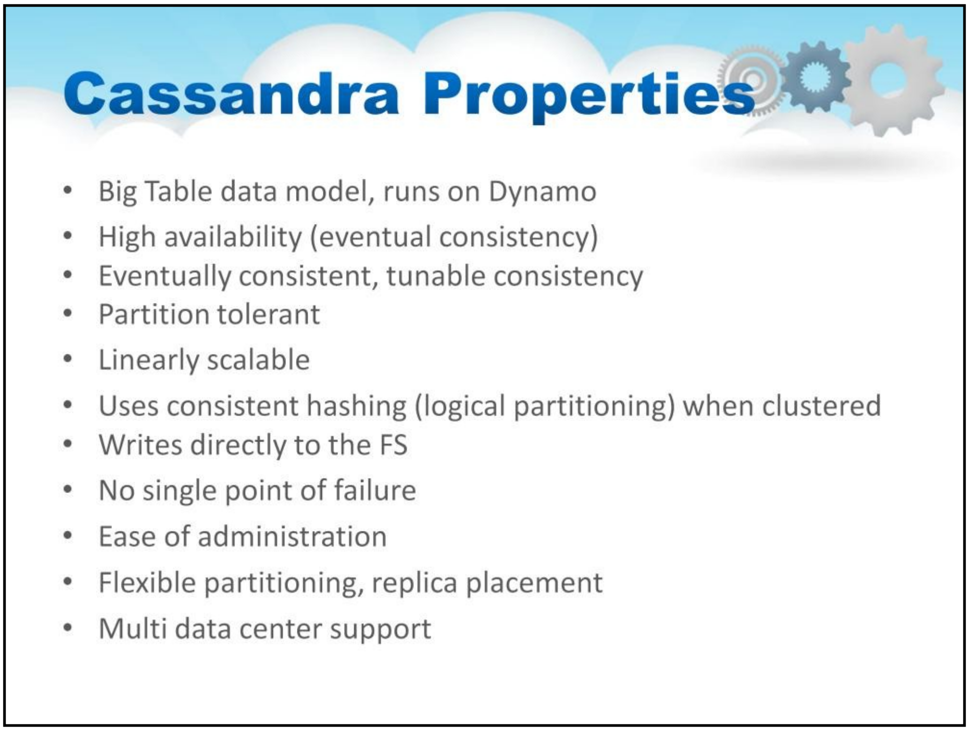
It Uses the Dynamo Eventual Consistency model

It is very portable, Written in Java

It is Open-sourced and exists within the Apache family

Cassandra Uses Apache Thrift as it's API

Additionally, it has Good integration with Hadoop stack: Pig Latin and Hive work there and here



Cassandra Properties

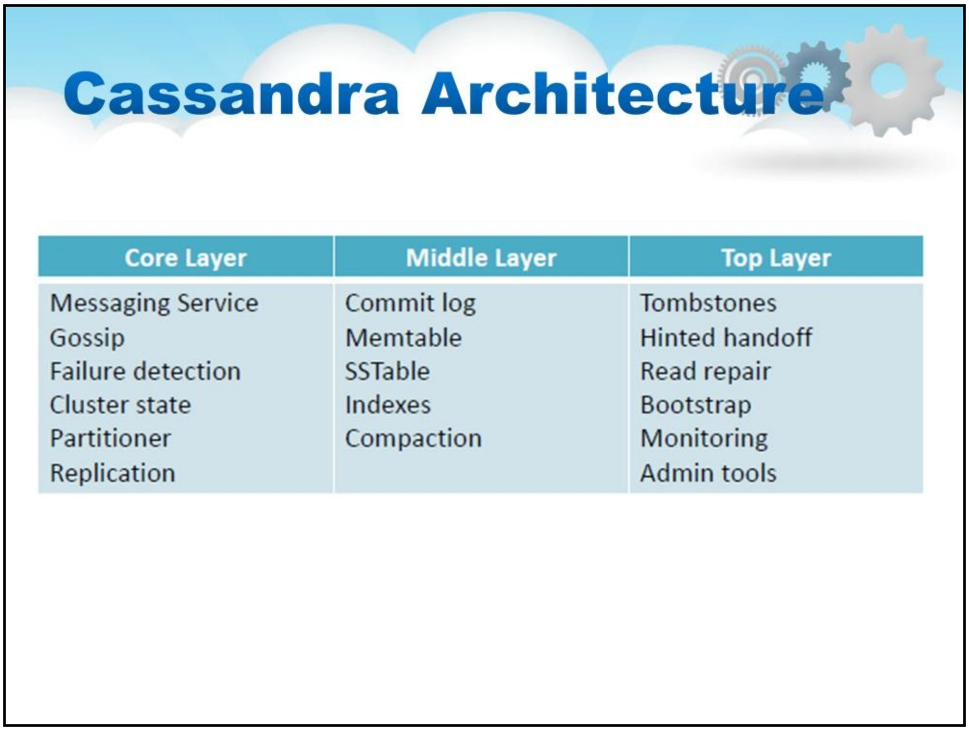
- Big Table data model, runs on Dynamo
- High availability (eventual consistency)
- Eventually consistent, tunable consistency
- Partition tolerant
- Linearly scalable
- Uses consistent hashing (logical partitioning) when clustered
- Writes directly to the FS
- No single point of failure
- Ease of administration
- Flexible partitioning, replica placement
- Multi data center support

Cassandra Properties in more detail

Cassandra is very capable. It is based on the Big Table data model

It implements:

High availability (eventual consistency)
Eventually consistent, tunable consistency
Partition tolerant
Linearly scalable
Uses consistent hashing (logical partitioning) when clustered
Writes directly to the FS
No single point of failure
Flexible partitioning, replica placement
Multi data center support



Cassandra Architecture

Core Layer	Middle Layer	Top Layer
Messaging Service	Commit log	Tombstones
Gossip	Memtable	Hinted handoff
Failure detection	SSTable	Read repair
Cluster state	Indexes	Bootstrap
Partitioner	Compaction	Monitoring
Replication		Admin tools

The Cassandra architecture can be described as follows:

The top layer is designed to allow efficient, consistent reads and writes using a simple API. The Cassandra API is made up of simple getter and setter methods and has no reference to the database distributed nature.

Another element in the top layer is Hinted hand-off. This occurs when a node goes down - the successor node becomes a coordinator (temporarily) with some information (`hint') about the failed node.

The middle layer contains functions for handling the data being written into the database. Compaction tries to combine keys and columns to increase the performance of the system. The different ways of storing data such as Memtable and SSTable are also handled here.

The core layer deals with the distributed nature of the database, and contains functions for communication between nodes, the state of the cluster as a whole (including failure detection) and replication between nodes.

Deeper look at HintedHandoff

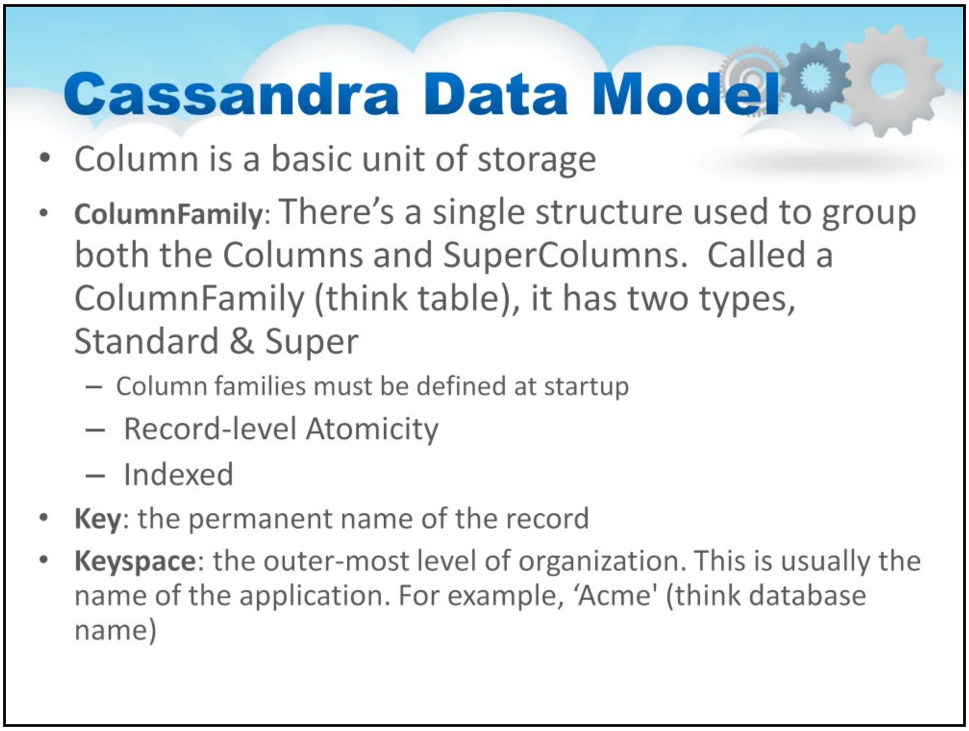
Cassandra Hinted hand off is an optimization technique for data write on replicas
When a write is made and a replica node for the key is down

Cassandra write a hint to a live replica node

That replica node will remind the downed node of changes once it is back on line

> HintedHandoff reduce write latency when a replica is temporarily down
> HintedHandoff provides high write availability at the cost of consistency
>
> A hinted write does NOT count towards ConsistencyLevel requirements for ONE, QUORUM, or ALL

If no replica nodes are alive for this key and ConsistencyLevel.ANY was specified, the coordinating node will write the hint locally



Cassandra Data Model

- Column is a basic unit of storage
- **ColumnFamily**: There’s a single structure used to group both the Columns and SuperColumns. Called a ColumnFamily (think table), it has two types, Standard & Super
  - Column families must be defined at startup
  - Record-level Atomicity
  - Indexed
- **Key**: the permanent name of the record
- **Keyspace**: the outer-most level of organization. This is usually the name of the application. For example, ‘Acme' (think database name)

Now to look at the Cassandra Data Model

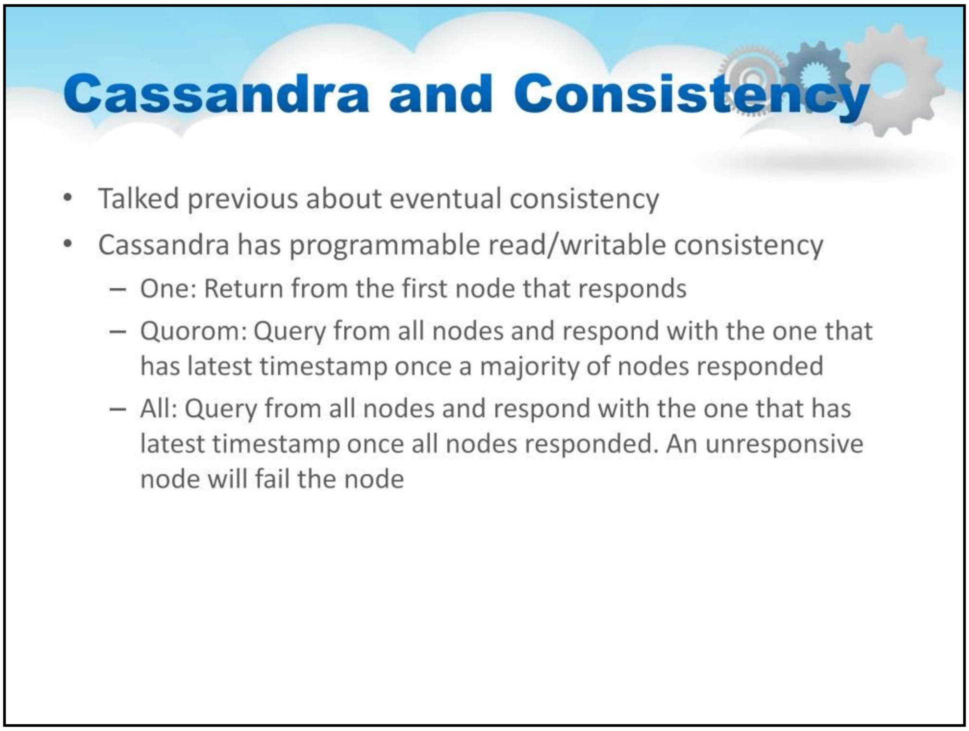
The **Column** is a basic unit of storage
There’s a single structure used to group both the Columns and SuperColumns. Called a ColumnFamily (think table), it has two types, Standard & Super

The **Key**: is the permanent name of the record

The **Keyspace**: is the outer-most level of organization.
This is usually the name of the application. For example, ‘Acme' (think database name)

Things to know about Cassandra:

-> Keys have different numbers of columns, so the database can scale in an irregular way.
-> Simple and Super: super columns are columns within columns.
-> Refer to date by **keyspace**, a **column family**, a **key**, an *optional* **super column**, and a **column**.



Cassandra and Consistency

- Talked previous about eventual consistency
- Cassandra has programmable read/writable consistency
  - One: Return from the first node that responds
  - Quorom: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded
  - All: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node

Cassandra and Consistency

A previous covered eventual consistency which is the mode Cassandra operates in.

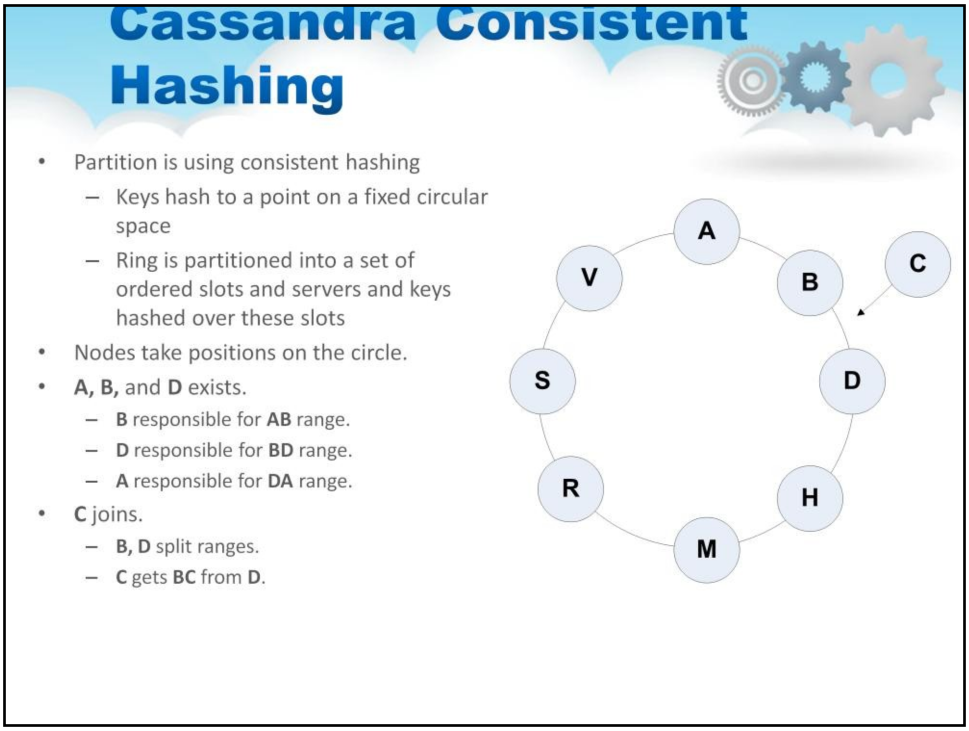
Cassandra has programmable read/writable consistency

“One”: Return from the first node that responds

“Quorom”: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded

“All”: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node

-> http://www.slideshare.net/julesbravo/cassandra-3125809
-> Have a good simple benchmark to show to demonstrate difference between Cassandra and MySQL



Cassandra Consistent Hashing

- Partition is using consistent hashing
  - Keys hash to a point on a fixed circular space
  - Ring is partitioned into a set of ordered slots and servers and keys hashed over these slots
- Nodes take positions on the circle.
- **A, B,** and **D** exists.
  - **B** responsible for **AB** range.
  - **D** responsible for **BD** range.
  - **A** responsible for **DA** range.
- **C** joins.
  - **B, D** split ranges.
  - **C** gets **BC** from **D**.

Cassandra Scales because of its Consistent Hashing architecture (Dynamo DHT Style)

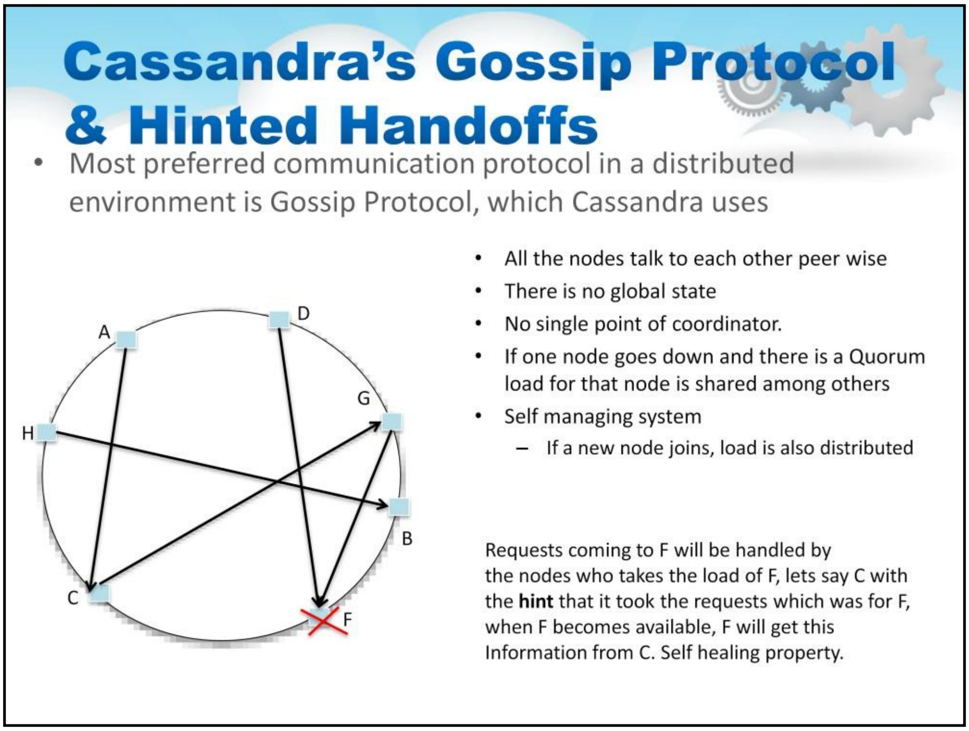
Partition uses consistent hashing

Keys hash to a point on a fixed circular space

The Ring is partitioned into a set of ordered slots and servers and keys hashed over these slots

Nodes take positions on the circle.

The slide has an illustration and an example of the circular space ranges.



Cassandra's Gossip Protocol & Hinted Handoffs

- Most preferred communication protocol in a distributed environment is Gossip Protocol, which Cassandra uses

- All the nodes talk to each other peer wise
- There is no global state
- No single point of coordinator.
- If one node goes down and there is a Quorum load for that node is shared among others
- Self managing system
  - If a new node joins, load is also distributed

Requests coming to F will be handled by the nodes who takes the load of F, lets say C with the **hint** that it took the requests which was for F, when F becomes available, F will get this Information from C. Self healing property.

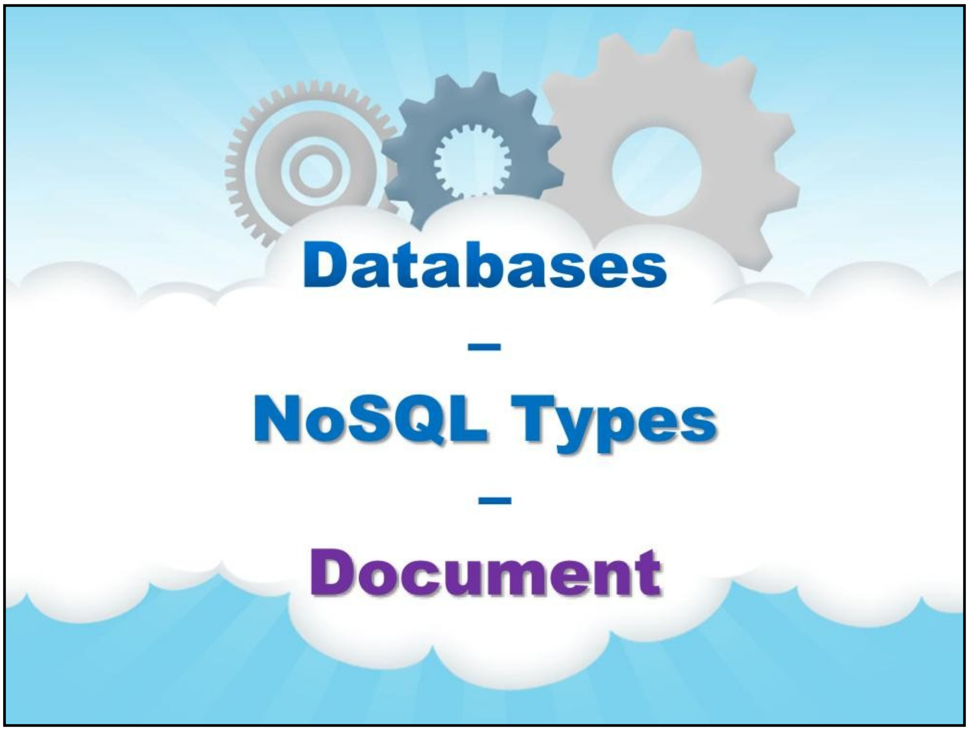
The Gossip Protocol is a method to resolve this communication chaos

Cassandra uses this Most preferred communication protocol in a distributed environment

In Gossip:

All the nodes talk to each other peer wise
There is no global state
No single point of coordinator.
If one node goes down and there is a Quorum load for that node is shared among others
It is a Self managing system
If a new node joins, load is also distributed

The slide has an illustration which shows the Protocol, and also provides an example of how it works.



Databases

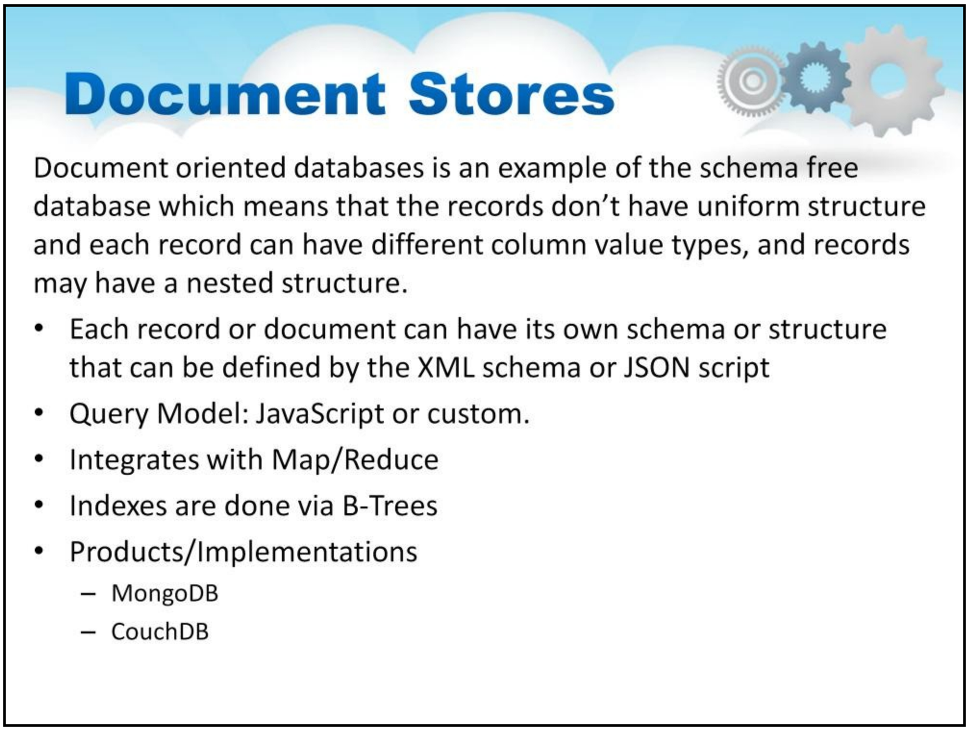
–

NoSQL Types

–

Document

Document



Document Stores

Document oriented databases is an example of the schema free database which means that the records don't have uniform structure and each record can have different column value types, and records may have a nested structure.

- Each record or document can have its own schema or structure that can be defined by the XML schema or JSON script
- Query Model: JavaScript or custom.
- Integrates with Map/Reduce
- Indexes are done via B-Trees
- Products/Implementations
  - MongoDB
  - CouchDB

Document oriented databases are another architecture of database.

They are an example of the schema free database
which means that the records don't have uniform structure
and each record can have different column value types,
and records may have a nested structure.

Each record or document can have its own schema or structure
that can be defined by the XML schema or JSON script

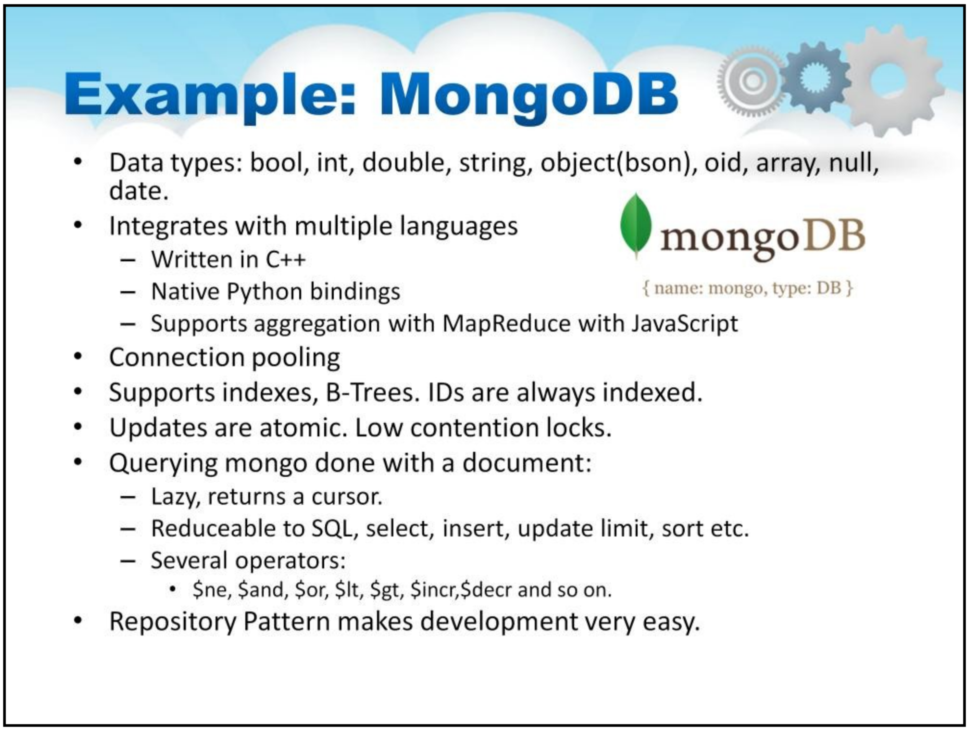
Query Model: JavaScript or custom.

Integrates with Map/Reduce

Indexes are done via B-Trees

Products/Implementations

MongoDB
CouchDB



Example: MongoDB

- Data types: bool, int, double, string, object(bson), oid, array, null, date.
- Integrates with multiple languages
  - Written in C++
  - Native Python bindings
  - Supports aggregation with MapReduce with JavaScript
- Connection pooling
- Supports indexes, B-Trees. IDs are always indexed.
- Updates are atomic. Low contention locks.
- Querying mongo done with a document:
  - Lazy, returns a cursor.
  - Reduceable to SQL, select, insert, update limit, sort etc.
  - Several operators:
    - $ne, $and, $or, $lt, $gt, $incr,$decr and so on.
- Repository Pattern makes development very easy.

To look at detail about MongoDB

It supports many Data types: bool, int, double, string, object(bson), oid, array, null, date.

It Integrates with multiple languages

Written in C++

Native Python bindings

Supports aggregation with MapReduce with JavaScript

It implements Connection pooling

It Supports indexes, B-Trees. IDs are always indexed.

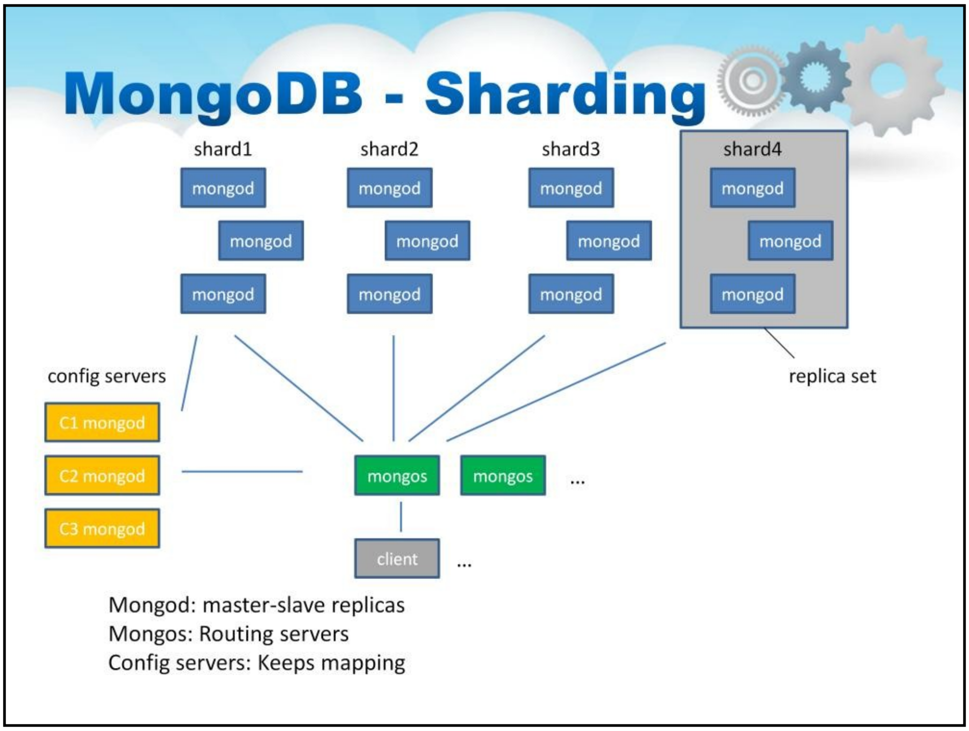
MongoDB Updates are atomic. There are Low contention locks.

Querying mongo done with a document:

It is a Lazy query meaning that the results are not returned, a cursor at the result location is.

Somewhat Reduceable to SQL subset, select, insert, update limit, sort etc.

Joins are not there as is typical of NoSQL systems.



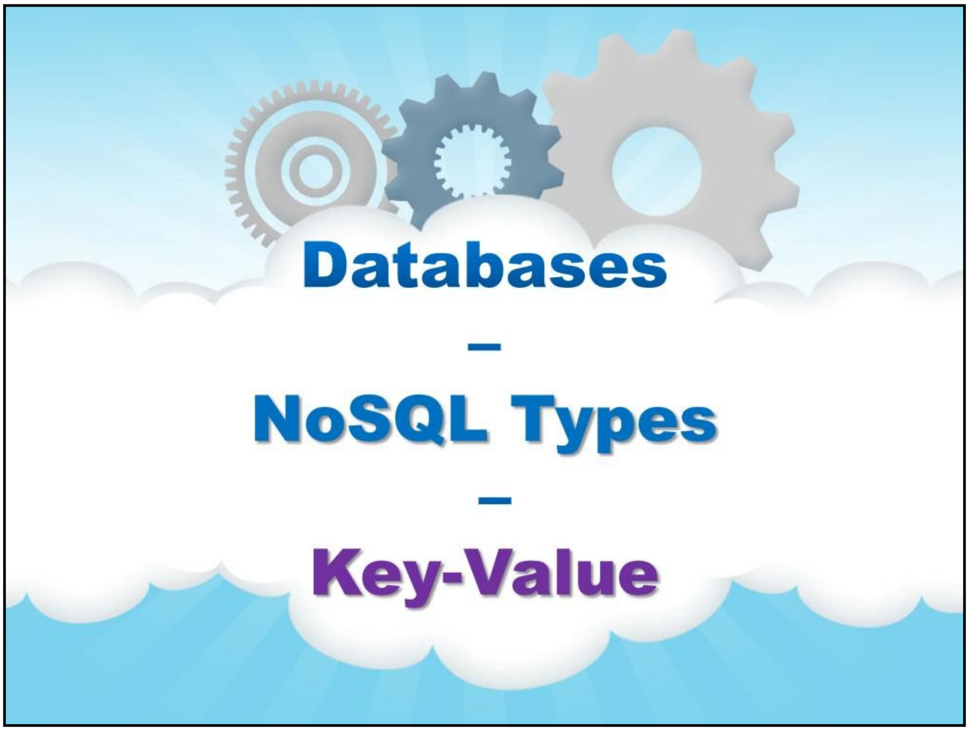
MongoDB - Sharding

Mongod: master-slave replicas
Mongos: Routing servers
Config servers: Keeps mapping

MongoDB implements sharding.

A MongoDB sharded cluster consists of the following components:
•shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
•mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
•config servers: Config servers store metadata and configuration settings forthe cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

The slide contains an illustration which shows how this work.



Databases

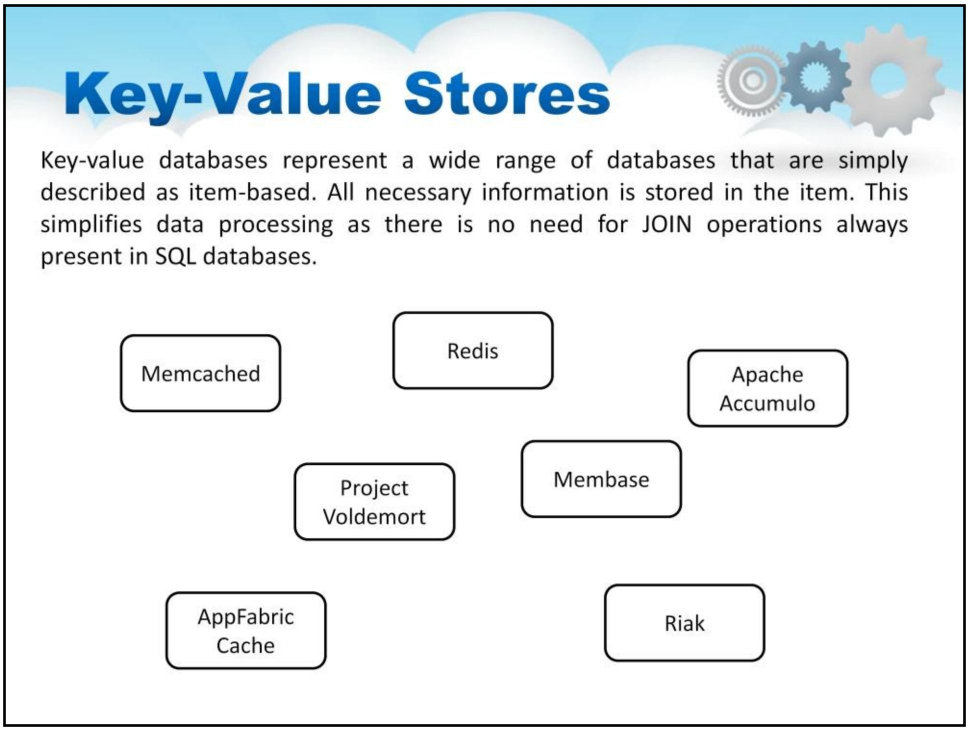
–

NoSQL Types

–

Key-Value

Key-Value



Key-Value Stores

Key-value databases represent a wide range of databases that are simply described as item-based. All necessary information is stored in the item. This simplifies data processing as there is no need for JOIN operations always present in SQL databases.

Memcached

Redis

Apache Accumulo

Project Voldemort

Membase

AppFabric Cache

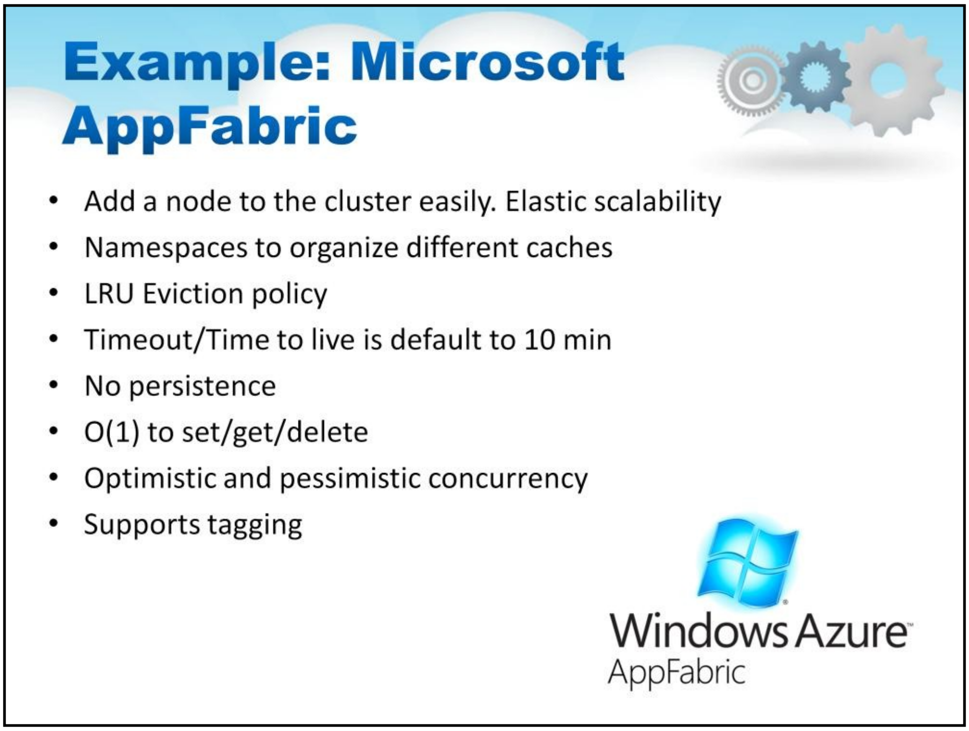
Riak

Next we look at Key-Value Stores

Key-value databases represent a wide range of databases that are simply described as item-based. All necessary information is stored in the item. This simplifies data processing as there is no need for JOIN operations always present in SQL databases.

Examples are:

Memcached – Key value stores
Membase – Memcached with persistence and improved consistent hashing
AppFabric Cache – Multi region Cache
Redis – Data structure server
Riak – Based on Amazon's Dynamo
Project Voldemort – eventual consistent key value stores, auto scaling
Apache Accumulo



Example: Microsoft AppFabric

- Add a node to the cluster easily. Elastic scalability
- Namespaces to organize different caches
- LRU Eviction policy
- Timeout/Time to live is default to 10 min
- No persistence
- O(1) to set/get/delete
- Optimistic and pessimistic concurrency
- Supports tagging

Windows Azure
AppFabric

Another Example: Microsoft AppFabric

As similar to memcached:

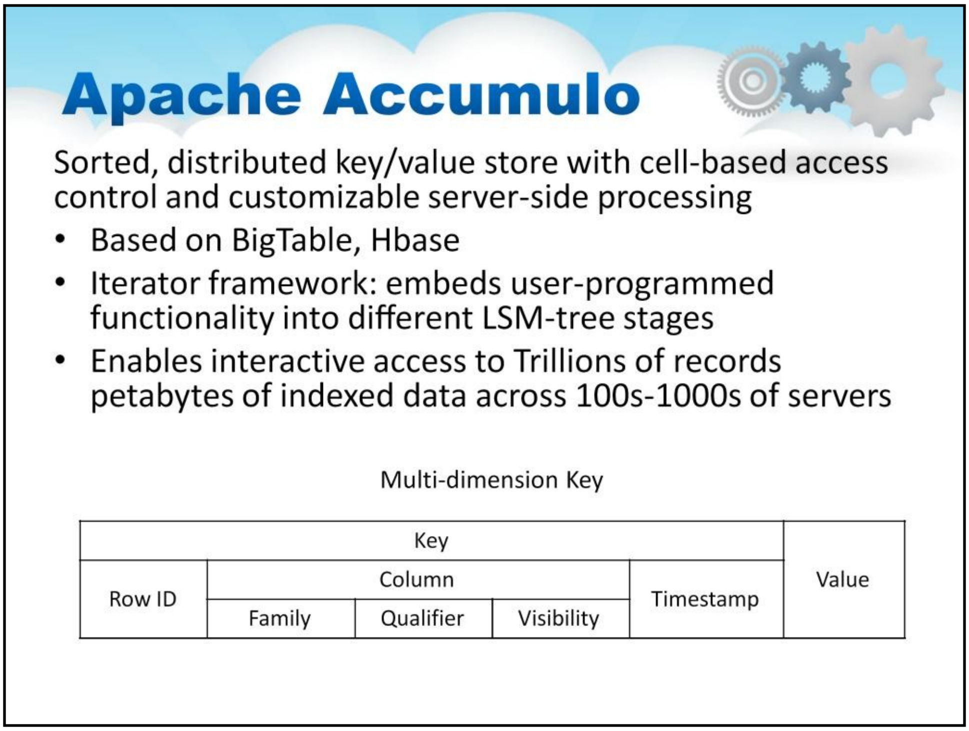
Add a node to the cluster easily. Elastic scalability

Namespaces to organize different caches

LRU Eviction policy

Timeout/Time to live is default to 10 min

No persistence



Apache Accumulo

Sorted, distributed key/value store with cell-based access control and customizable server-side processing

- Based on BigTable, Hbase
- Iterator framework: embeds user-programmed functionality into different LSM-tree stages
- Enables interactive access to Trillions of records petabytes of indexed data across 100s-1000s of servers

Multi-dimension Key

Key					Value
Row ID	Column			Timestamp	
	Family	Qualifier	Visibility		

Apache Accumulo is another key value store example,

Apache Accumulo is a Sorted, distributed key/value store with cell-based access control and customizable server-side processing

Developed by NSA (National Security Agency), also open source

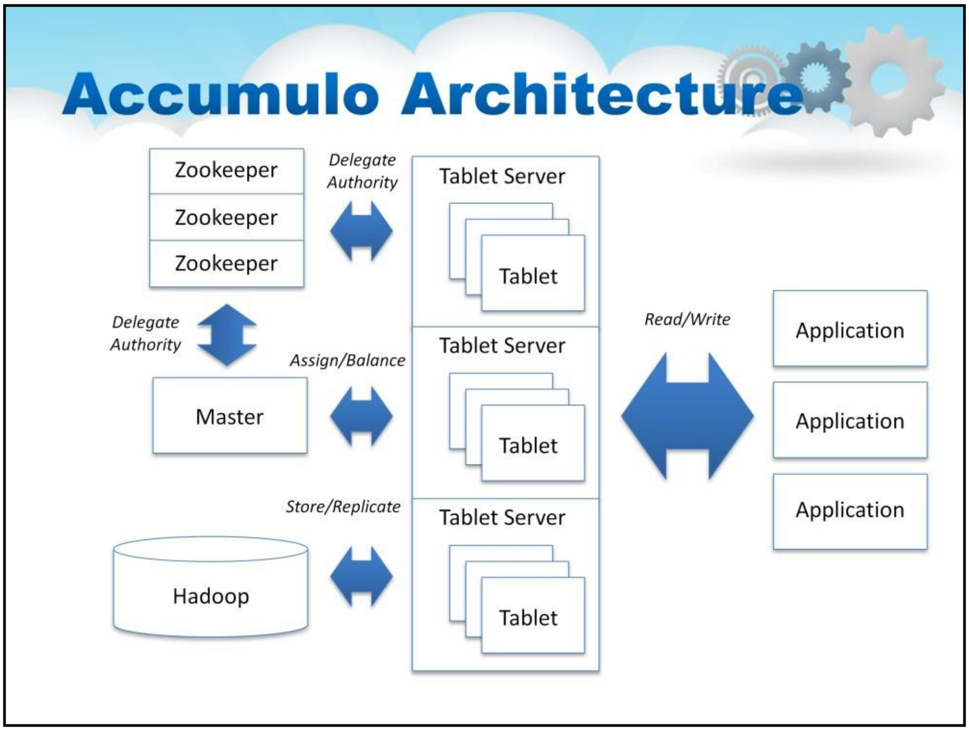
Based on BigTable, Hbase

Iterator framework: embeds user-programmed functionality into different LSM-tree stages

For example, iterators can operate during minor compactions by using the memstore data as input to generate on-disk store files comprised of some transformation of the input such as statistics or additional indices

Enables interactive access to Trillions of records petabytes of indexed data across 100s-1000s of servers

The slide contains an illustration which shows the way key, columns, and rows are organized in Accumulo.



Accumulo Architecture

Zookeeper

Zookeeper

Zookeeper

Delegate Authority

Delegate Authority

Master

Assign/Balance

Store/Replicate

Hadoop

Tablet Server

Tablet

Tablet Server

Tablet

Tablet Server

Tablet

Read/Write

Application

Application

Application

The slide contains an illustration which shows the architecture in Accumulo

Accumulo’s architecture consists of the following components:

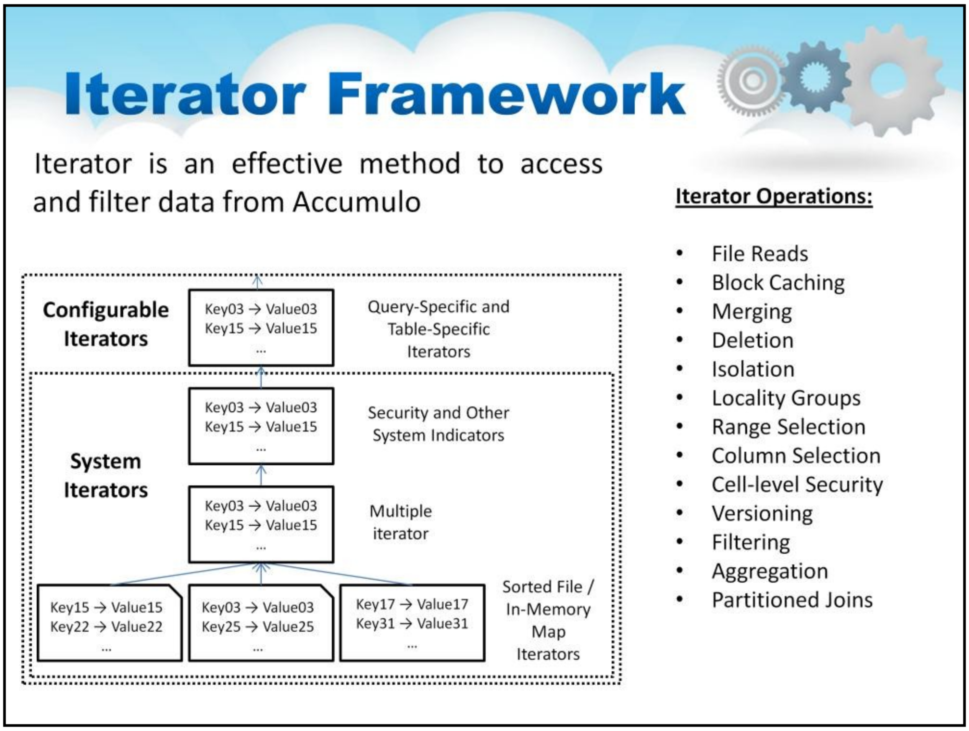
Tablets: Partitions of tables consisting of sorted key/value pairs.

Tablet servers: Manage the tablets, including receiving writes from clients, persisting writes to a write- ahead log, sorting new key-value pairs in memory, periodically fl using sorted key-value pairs to new files in HDFS and responding to reads from clients. During a read the tablet servers provide a merge-sorted view of all keys and values from the files it has created and the sorted in-memory store.

Master: Responsible for detecting and responding to tablet server failure. The Master tries to balance the load across Tablet Servers by assigning the tablets carefully and instructing Tablet Servers to migrate the tablets when necessary. The Master ensures each tablet is assigned to exactly one Tablet Server, and handles many miscellaneous database administration requests. The master also coordinates startup, graceful shutdown and recovery of write-ahead logs when the tablet servers fail.

ZooKeeper: Distributed locking mechanism with no single point of failure. Zookeeper is responsible for maintaining configuration information, naming, and providing distributed synchronization. (http://www.sqrrl.com/whitepaper/)

At the heart of Accumulo is the Tablet mechanism, which simultaneously optimizes for low latency between random writes and sorted reads (real-time query support) and efficient use of disk-based storage.



Iterator Framework

Iterator is an effective method to access and filter data from Accumulo

Configurable Iterators

Key03 → Value03
Key15 → Value15
…

Query-Specific and Table-Specific Iterators

System Iterators

Key03 → Value03
Key15 → Value15
…

Security and Other System Indicators

Key03 → Value03
Key15 → Value15
…

Multiple iterator

Key15 → Value15
Key22 → Value22
…

Key03 → Value03
Key25 → Value25
…

Key17 → Value17
Key31 → Value31
…

Sorted File / In-Memory Map Iterators

Iterator Operations:

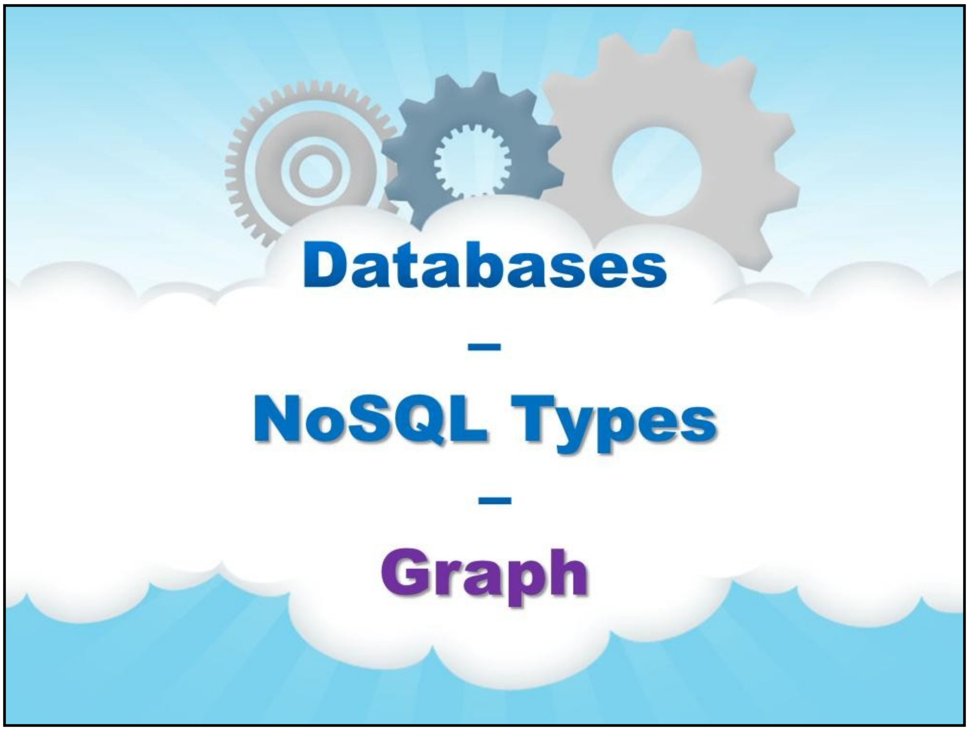
- File Reads
- Block Caching
- Merging
- Deletion
- Isolation
- Locality Groups
- Range Selection
- Column Selection
- Cell-level Security
- Versioning
- Filtering
- Aggregation
- Partitioned Joins

Accumulo supports more advanced data processing than simply keeping keys sorted and performing efficient lookups.

Analytics can be developed by using MapReduce and Iterators in conjunction with Accumulo tables.

Iterator is an effective method to access and filter data from Accumulo

The slide contains an illustration which shows the Iterator Framework in Accumulo



Databases

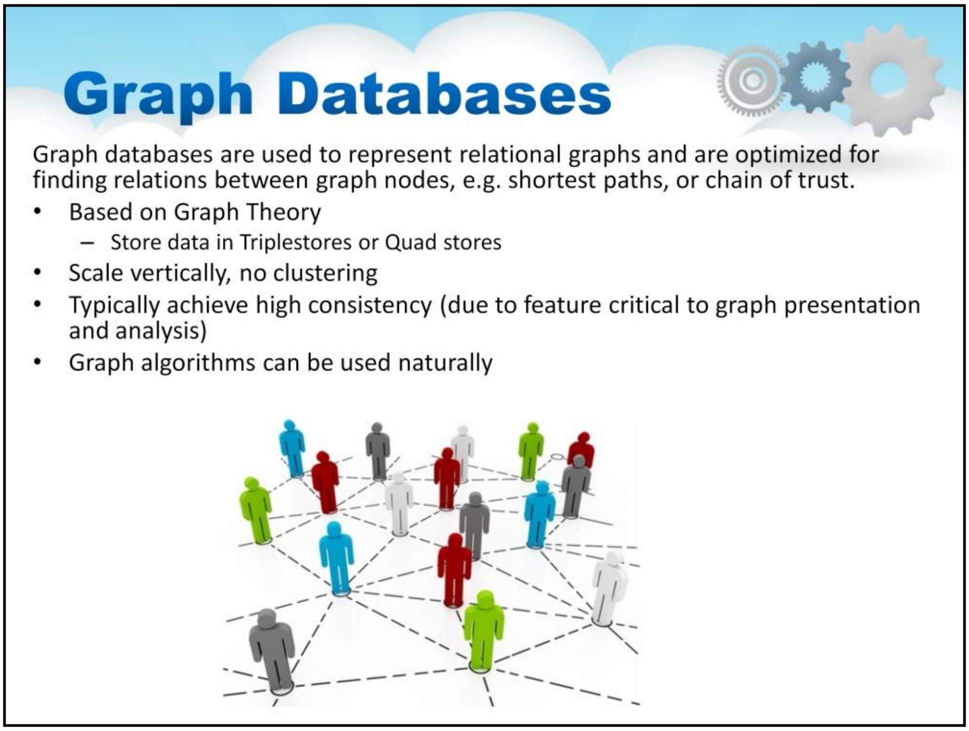
–

NoSQL Types

–

Graph

Graph



Graph Databases

Graph databases are used to represent relational graphs and are optimized for finding relations between graph nodes, e.g. shortest paths, or chain of trust.

- Based on Graph Theory
  - Store data in Triplestores or Quad stores
- Scale vertically, no clustering
- Typically achieve high consistency (due to feature critical to graph presentation and analysis)
- Graph algorithms can be used naturally

Graph Databases

Graph databases are used to represent relational graphs and are optimized for finding relations between graph nodes, e.g. shortest paths, or chain of trust.

Based on Graph Theory

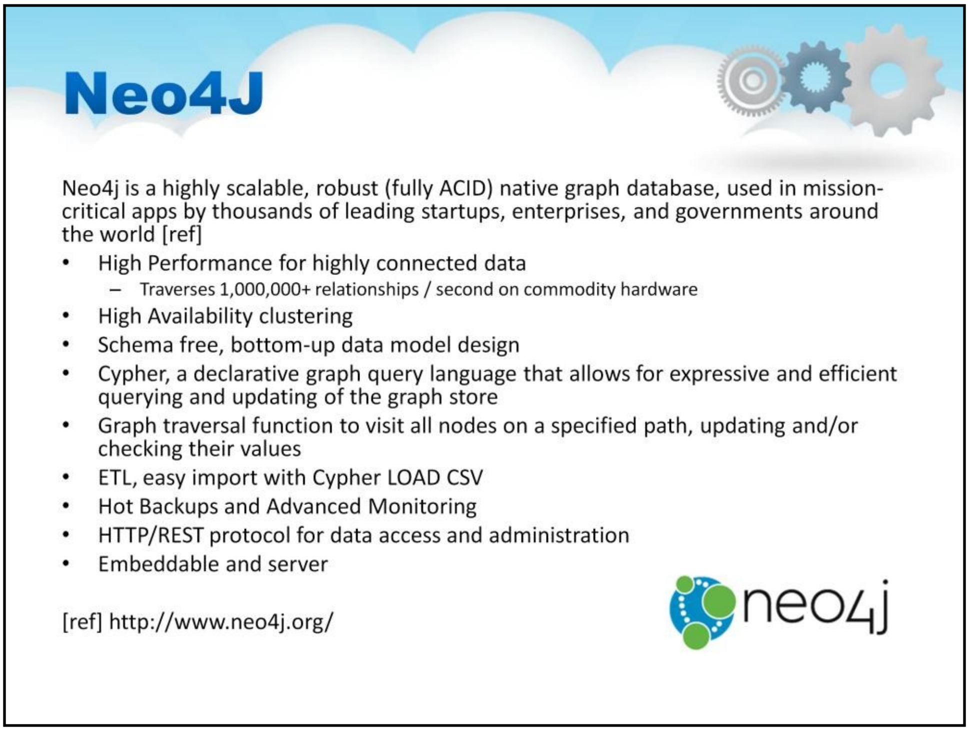
Store data in Triplestores or Quad stores

Scale vertically, no clustering

Typically achieve high consistency (due to feature critical to graph presentation and analysis)

Graph algorithms can be used naturally

For example, great for Social Networks



Neo4J

Neo4j is a highly scalable, robust (fully ACID) native graph database, used in mission-critical apps by thousands of leading startups, enterprises, and governments around the world [ref]

- High Performance for highly connected data
  - Traverses 1,000,000+ relationships / second on commodity hardware
- High Availability clustering
- Schema free, bottom-up data model design
- Cypher, a declarative graph query language that allows for expressive and efficient querying and updating of the graph store
- Graph traversal function to visit all nodes on a specified path, updating and/or checking their values
- ETL, easy import with Cypher LOAD CSV
- Hot Backups and Advanced Monitoring
- HTTP/REST protocol for data access and administration
- Embeddable and server

[ref] http://www.neo4j.org/

An example of a graph database is Neo4J

Neo4j is a highly scalable, robust (fully ACID) native graph database

High Performance for highly connected data

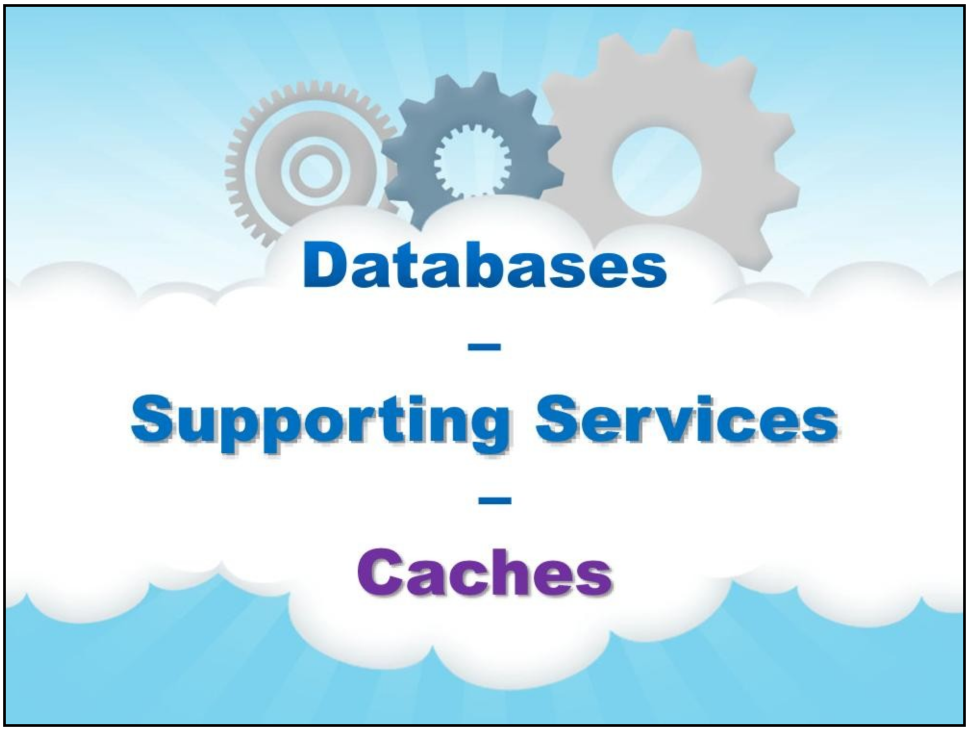
Traverses 1,000,000+ relationships / second on commodity hardware

High Availability clustering

Schema free, bottom-up data model design

Cypher, a declarative graph query language that allows for expressive and efficient querying and updating of the graph store

Graph traversal function to visit all nodes on a specified path, updating and/or checking their values



Databases

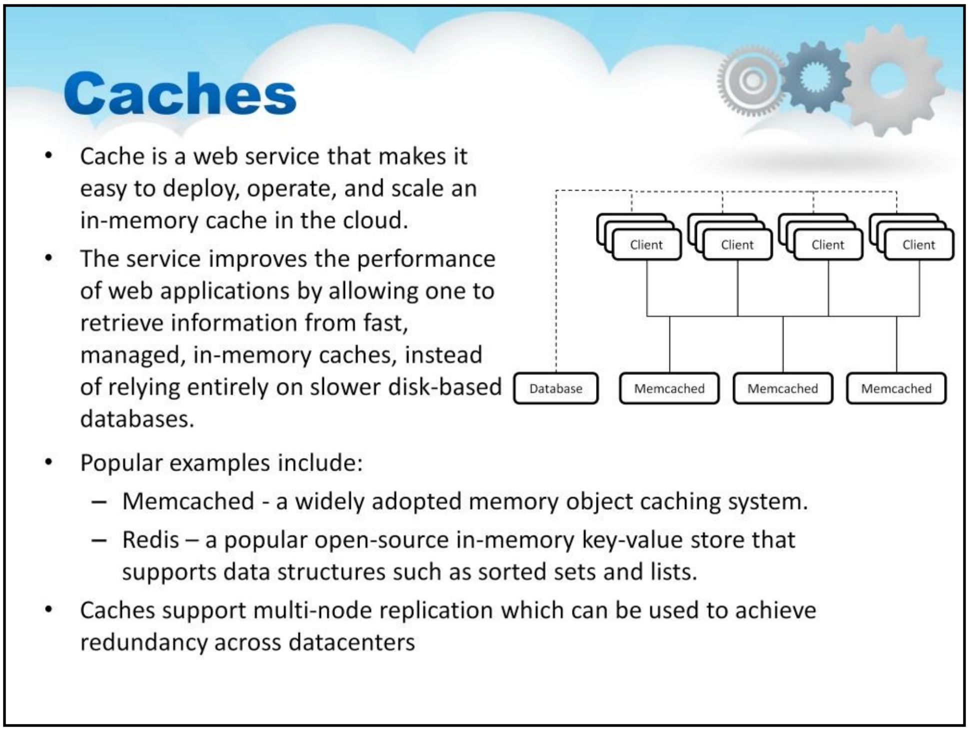
–

Supporting Services

–

Caches

Caches



Caches

- Cache is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud.
- The service improves the performance of web applications by allowing one to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.
- Popular examples include:
  - Memcached - a widely adopted memory object caching system.
  - Redis – a popular open-source in-memory key-value store that supports data structures such as sorted sets and lists.
- Caches support multi-node replication which can be used to achieve redundancy across datacenters

Another important storage tool in Cloud, which is actually very close architecturally to the Database, is the Cache.

The Cache is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud.

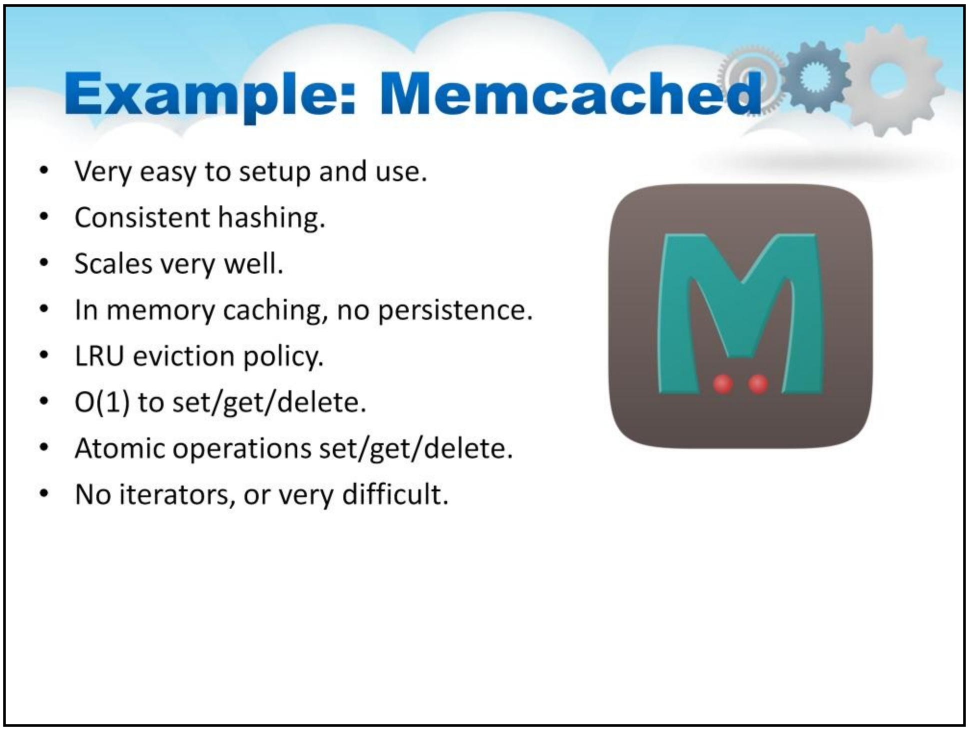
The service improves the performance of web applications by allowing one to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.

Popular examples include:

Memcached - a widely adopted memory object caching system.

Redis – a popular open-source in-memory key-value store that supports data structures such as sorted sets and lists.

Caches support multi-node replication which can be used to achieve redundancy across datacenters



Example: Memcached

- Very easy to setup and use.
- Consistent hashing.
- Scales very well.
- In memory caching, no persistence.
- LRU eviction policy.
- O(1) to set/get/delete.
- Atomic operations set/get/delete.
- No iterators, or very difficult.

Memcahced is very popular and comes as a preconfigured service on many clouds.

Memcached is:

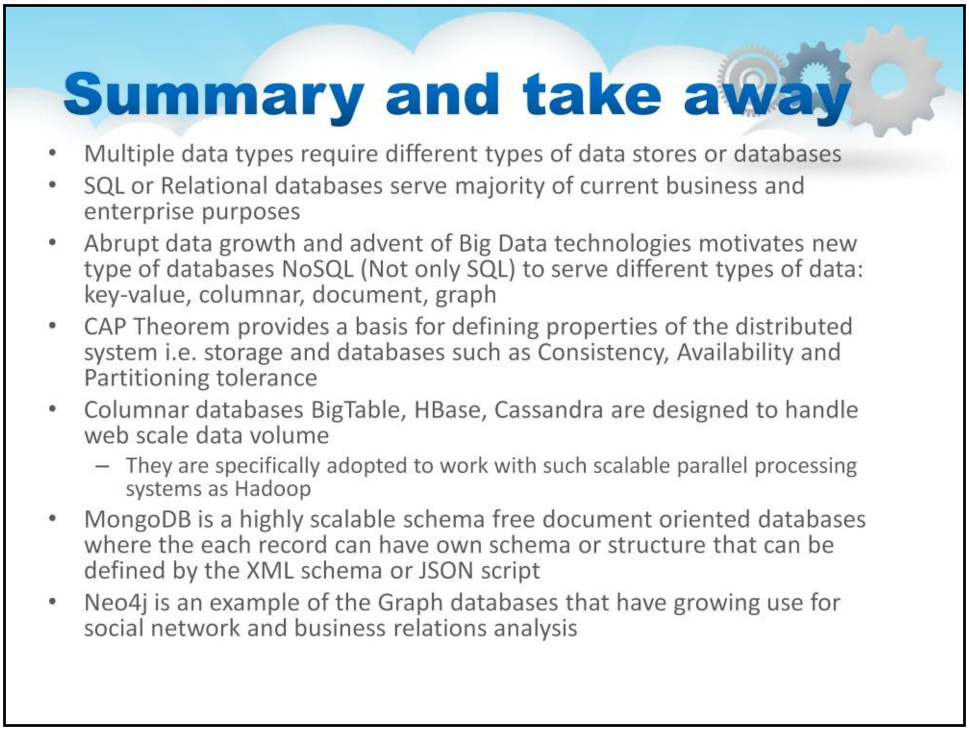
Very easy to setup and use.

Uses Consistent hashing.

Scales very well.

Implements In memory caching, no persistence.

LRU eviction policy



Summary and take away

- Multiple data types require different types of data stores or databases
- SQL or Relational databases serve majority of current business and enterprise purposes
- Abrupt data growth and advent of Big Data technologies motivates new type of databases NoSQL (Not only SQL) to serve different types of data: key-value, columnar, document, graph
- CAP Theorem provides a basis for defining properties of the distributed system i.e. storage and databases such as Consistency, Availability and Partitioning tolerance
- Columnar databases BigTable, HBase, Cassandra are designed to handle web scale data volume
  - They are specifically adopted to work with such scalable parallel processing systems as Hadoop
- MongoDB is a highly scalable schema free document oriented databases where the each record can have own schema or structure that can be defined by the XML schema or JSON script
- Neo4j is an example of the Graph databases that have growing use for social network and business relations analysis

In this lecture we covered the following aspects:

Multiple data types require different types of data stores or databases

SQL or Relational databases serve majority of current business and enterprise purposes

Abrupt data growth and advent of Big Data technologies motivates new type of databases NoSQL (Not only SQL) to serve different types of data: key-value, columnar, document, graph

CAP Theorem provides a basis for defining properties of the distributed system i.e. storage and databases such as Consistency, Availability and Partitioning tolerance

Columnar databases BigTable, HBase, Cassandra are designed to handle web scale data volume. They are specifically adopted to work with such scalable parallel processing systems as Hadoop

MongoDB is a highly scalable schema free document oriented databases where the each record can have own schema or structure that can be defined by the XML schema or JSON script

Neo4j is an example of the Graph databases that have growing use for social network and business relations analysis

Cloud Computing

Lecture Manual

Volume 7

Module 7

Cloud Computing security

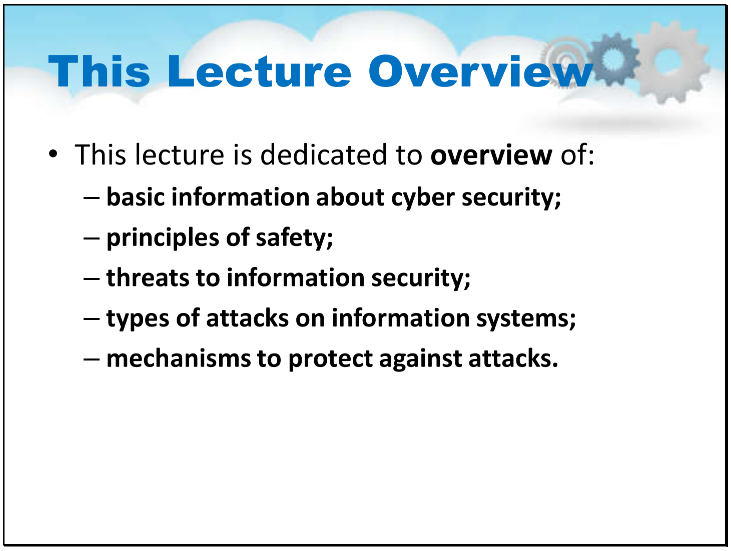
Content

Cloud Computing

Module – Cloud Computing security

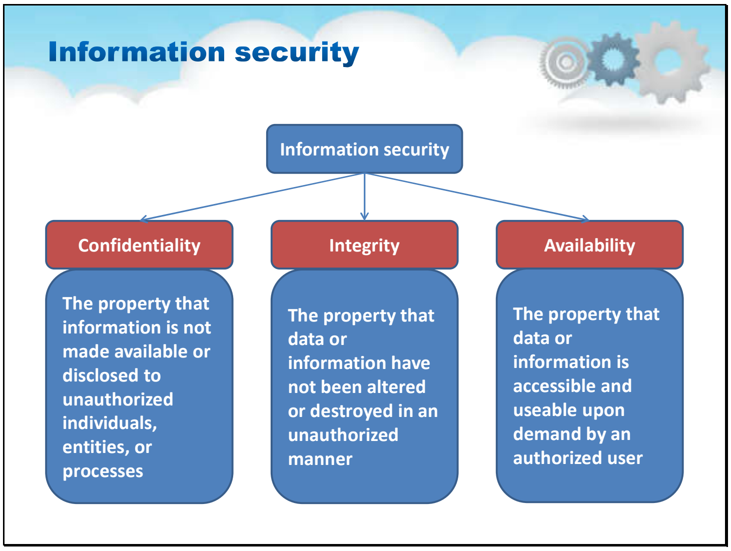
Lecture 1. Threats and risks of using cloud computing

Lecture 1.Threats and risks of using cloud computing



This Lecture Overview

- This lecture is dedicated to **overview** of:
  - **basic information about cyber security;**
  - **principles of safety;**
  - **threats to information security;**
  - **types of attacks on information systems;**
  - **mechanisms to protect against attacks.**



Information security

Information security

Confidentiality

The property that information is not made available or disclosed to unauthorized individuals, entities, or processes

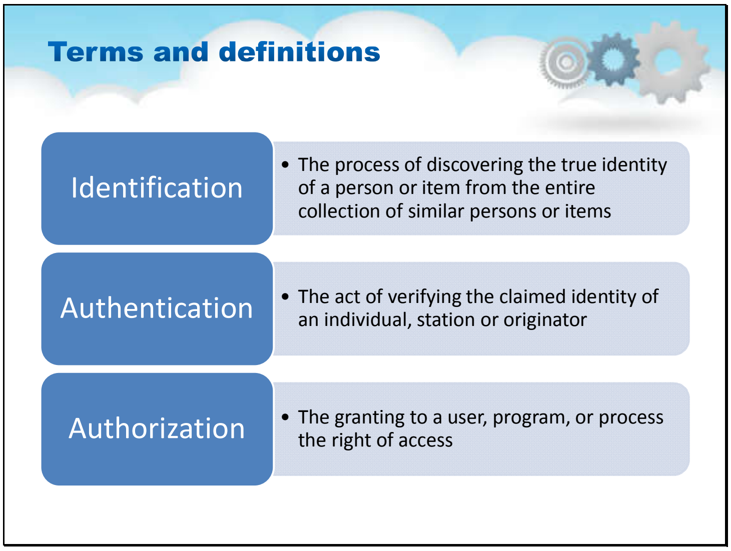
Integrity

The property that data or information have not been altered or destroyed in an unauthorized manner

Availability

The property that data or information is accessible and useable upon demand by an authorized user

Basic information about cyber security



Terms and definitions

Identification

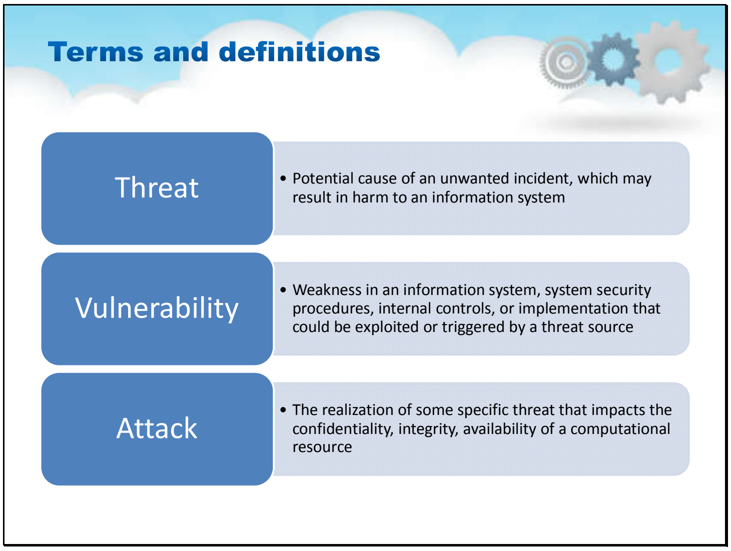
- The process of discovering the true identity of a person or item from the entire collection of similar persons or items

Authentication

- The act of verifying the claimed identity of an individual, station or originator

Authorization

- The granting to a user, program, or process the right of access



Terms and definitions

Threat

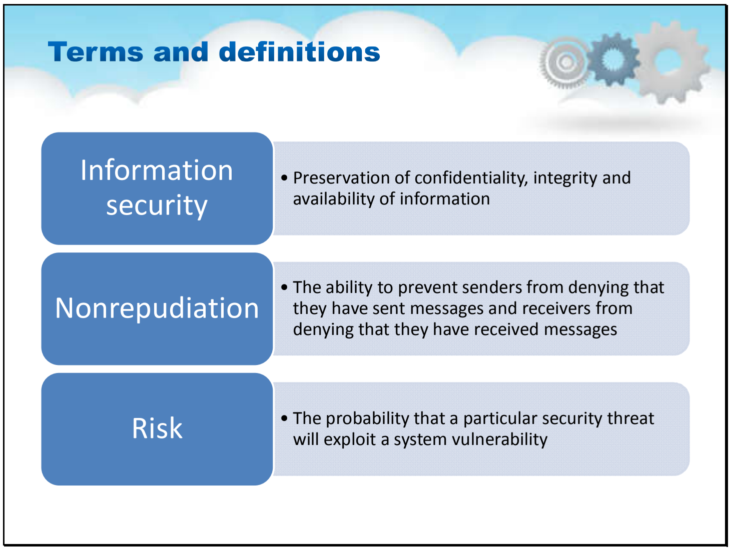
- Potential cause of an unwanted incident, which may result in harm to an information system

Vulnerability

- Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source

Attack

- The realization of some specific threat that impacts the confidentiality, integrity, availability of a computational resource



Terms and definitions

Information security

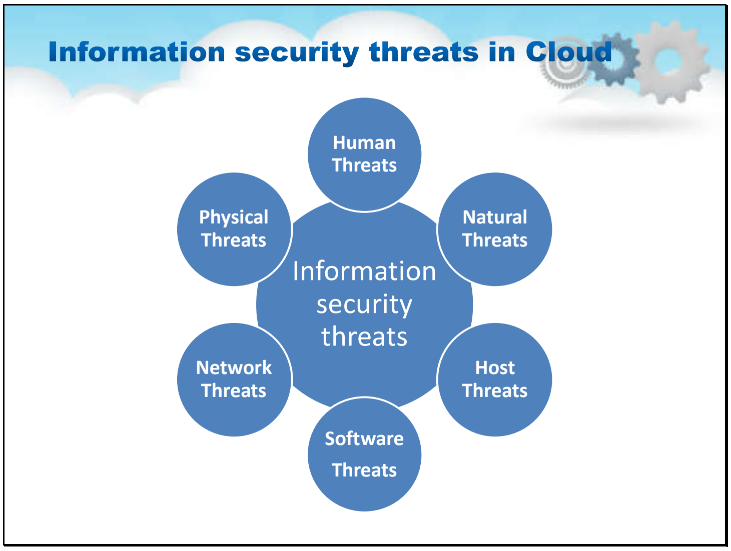
- Preservation of confidentiality, integrity and availability of information

Nonrepudiation

- The ability to prevent senders from denying that they have sent messages and receivers from denying that they have received messages

Risk

- The probability that a particular security threat will exploit a system vulnerability



Information security threats in Cloud

Information security threats

Human Threats

Natural Threats

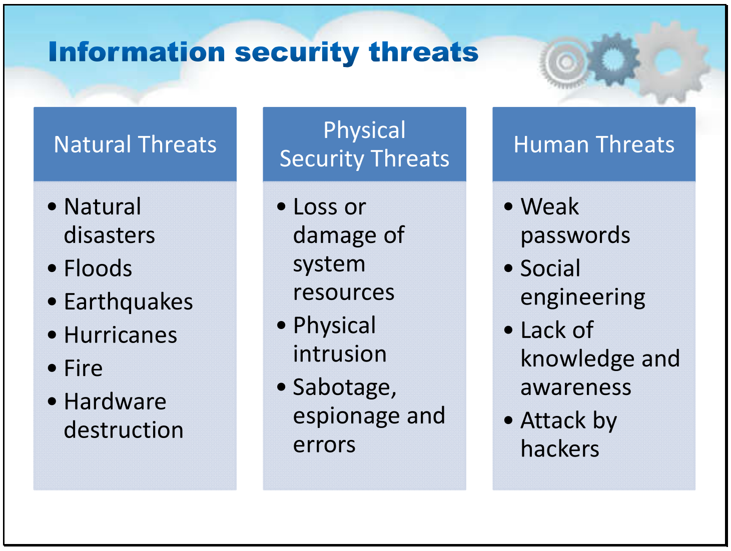
Host Threats

Software Threats

Network Threats

Physical Threats

Threats to information security



Information security threats

Natural Threats

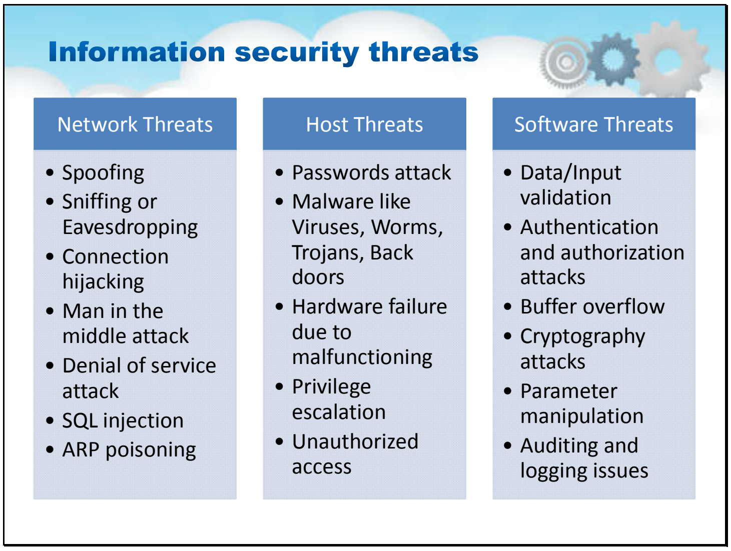
- Natural disasters
- Floods
- Earthquakes
- Hurricanes
- Fire
- Hardware destruction

Physical Security Threats

- Loss or damage of system resources
- Physical intrusion
- Sabotage, espionage and errors

Human Threats

- Weak passwords
- Social engineering
- Lack of knowledge and awareness
- Attack by hackers



Information security threats

Network Threats

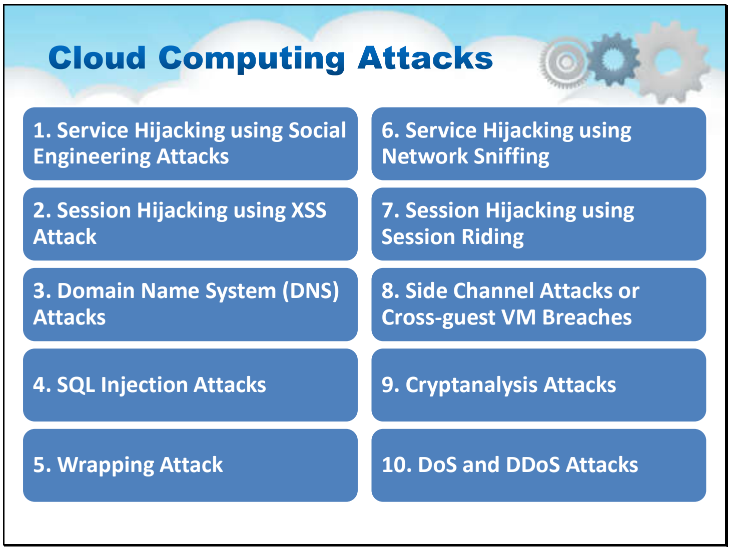
- Spoofing
- Sniffing or Eavesdropping
- Connection hijacking
- Man in the middle attack
- Denial of service attack
- SQL injection
- ARP poisoning

Host Threats

- Passwords attack
- Malware like Viruses, Worms, Trojans, Back doors
- Hardware failure due to malfunctioning
- Privilege escalation
- Unauthorized access

Software Threats

- Data/Input validation
- Authentication and authorization attacks
- Buffer overflow
- Cryptography attacks
- Parameter manipulation
- Auditing and logging issues



Cloud Computing Attacks

1. Service Hijacking using Social Engineering Attacks
2. Session Hijacking using XSS Attack
3. Domain Name System (DNS) Attacks
4. SQL Injection Attacks
5. Wrapping Attack
6. Service Hijacking using Network Sniffing
7. Session Hijacking using Session Riding
8. Side Channel Attacks or Cross-guest VM Breaches
9. Cryptanalysis Attacks
10. DoS and DDoS Attacks

Cloud Computing Attacks

Service Hijacking using Social Engineering Attacks

Using Social Engineering techniques, the attack may attempt to guess the password. Social Engineering attacks result in unauthorized access exposing sensitive information according to the privilege level of the compromised user.

Service Hijacking using Network Sniffing

Using Packet Sniffing tools by placing himself in the network, an attacker can capture sensitive information such as passwords, session ID, cookies, and another web servicerelated information such as UDDI, SOAP, and WSDL

Session Hijacking using XSS Attack

By launching Cross-Site Scripting (XSS), the attacker can steal cookies by injecting malicious code into the website.

Session Hijacking using Session Riding

Session Riding is intended for session hijacking. An attacker may exploit it by attempting cross-site request forgery. The attacker uses currently active session and rides on it by executing the requests such as modification of data, erasing data, online transactions and password change by tracking the user to click on a malicious link.

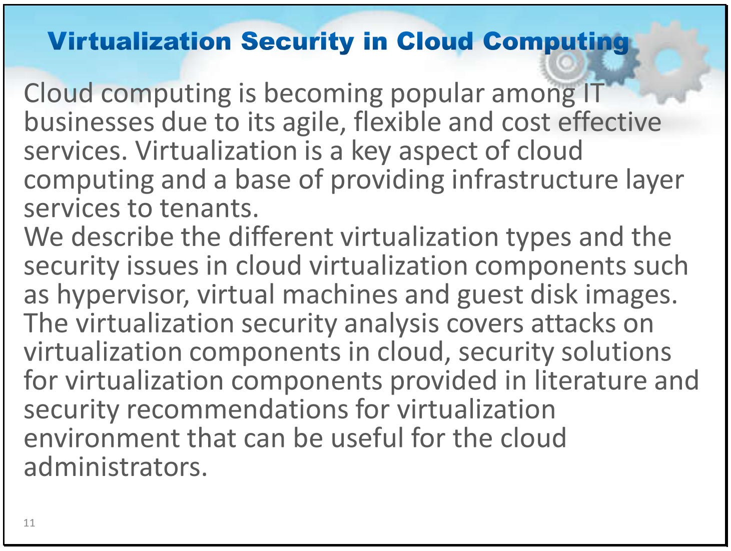
Domain Name System (DNS) Attacks

Domain Name System (DNS) attacks include DNS Poisoning, Cybersquatting, Domain hijacking and Domain Snipping. An attacker may attempt to spoof by poisoning the DNS server or cache to obtain credentials of internal users. Domain Hijacking involves stealing cloud service domain name. Similarly, through Phishing scams, users can be redirected to a fake website.

Side Channel Attacks or Cross-guest VM Breaches

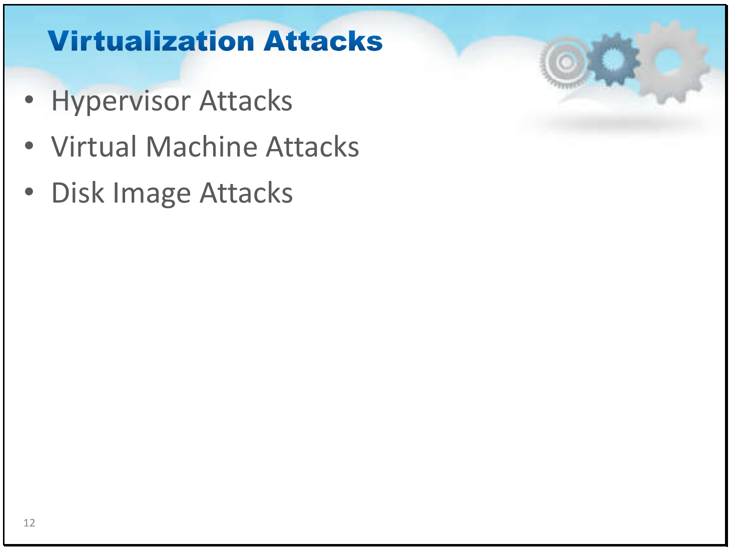
Side Channel Attacks or Cross-Guest VM Breach is an attack which requires the deployment of a malicious virtual machine on the same host. For example, a physical host is hosting a virtual machine that is offering the cloud services hence the target of an attacker. The attacker will install a malicious virtual machine on the same host to take advantage of sharing resources of the same host such as processor cache, cryptographic keys, etc. Installation can be done by a malicious insider, or an attacker by impersonating a legitimate user.

Similarly, there are other attackers that are discussed earlier are also vulnerable to Cloud Computing such as SQL Injection attack (injecting malicious SQL statements to extract information), Cryptanalysis Attacks (weak or obsolete encryption) Wrapping Attack (duplicating the body of message), Denial-of-Service (DoS) and Distributed Denial-ofService (DDoS) Attacks.



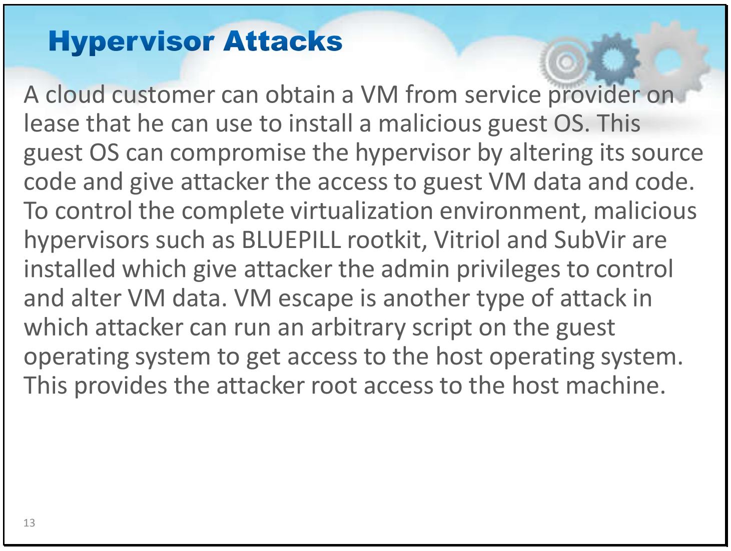
Virtualization Security in Cloud Computing

Cloud computing is becoming popular among IT businesses due to its agile, flexible and cost effective services. Virtualization is a key aspect of cloud computing and a base of providing infrastructure layer services to tenants.
We describe the different virtualization types and the security issues in cloud virtualization components such as hypervisor, virtual machines and guest disk images.
The virtualization security analysis covers attacks on virtualization components in cloud, security solutions for virtualization components provided in literature and security recommendations for virtualization environment that can be useful for the cloud administrators.



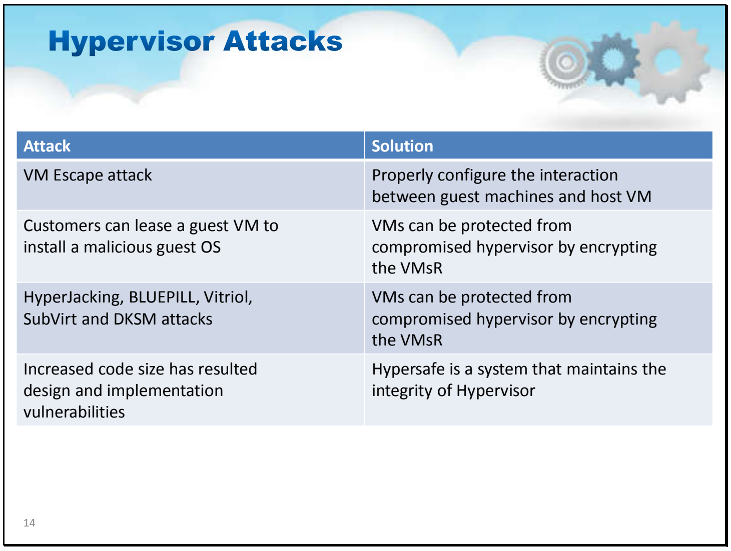
Virtualization Attacks

- Hypervisor Attacks
- Virtual Machine Attacks
- Disk Image Attacks



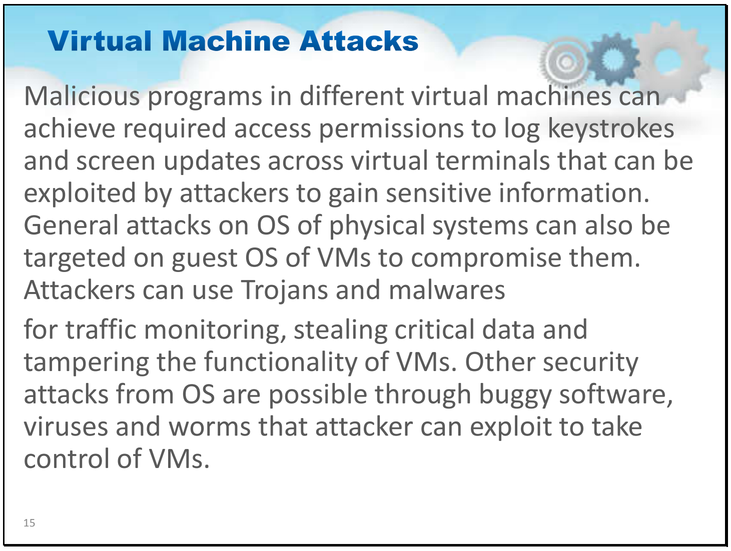
Hypervisor Attacks

A cloud customer can obtain a VM from service provider on lease that he can use to install a malicious guest OS. This guest OS can compromise the hypervisor by altering its source code and give attacker the access to guest VM data and code. To control the complete virtualization environment, malicious hypervisors such as BLUEPILL rootkit, Vitriol and SubVir are installed which give attacker the admin privileges to control and alter VM data. VM escape is another type of attack in which attacker can run an arbitrary script on the guest operating system to get access to the host operating system. This provides the attacker root access to the host machine.



Hypervisor Attacks

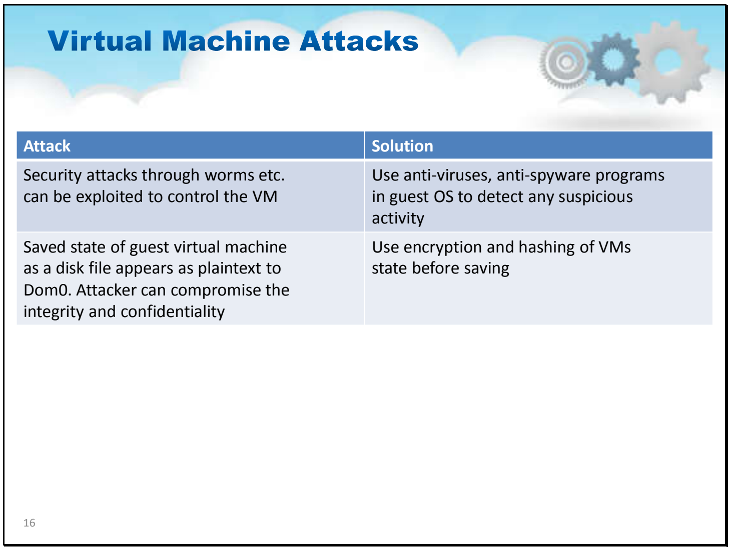
Attack	Solution
VM Escape attack	Properly configure the interaction between guest machines and host VM
Customers can lease a guest VM to install a malicious guest OS	VMs can be protected from compromised hypervisor by encrypting the VMsR
HyperJacking, BLUEPILL, Vitriol, SubVirt and DKSM attacks	VMs can be protected from compromised hypervisor by encrypting the VMsR
Increased code size has resulted design and implementation vulnerabilities	Hypersafe is a system that maintains the integrity of Hypervisor



Virtual Machine Attacks

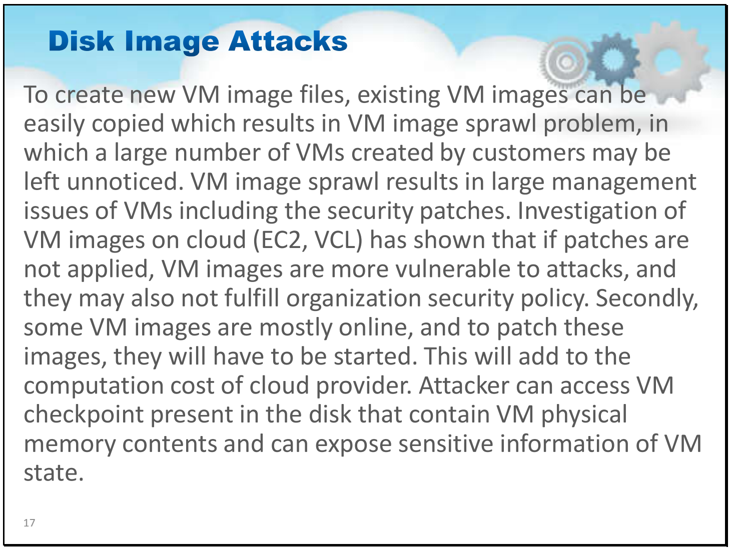
Malicious programs in different virtual machines can achieve required access permissions to log keystrokes and screen updates across virtual terminals that can be exploited by attackers to gain sensitive information. General attacks on OS of physical systems can also be targeted on guest OS of VMs to compromise them. Attackers can use Trojans and malwares

for traffic monitoring, stealing critical data and tampering the functionality of VMs. Other security attacks from OS are possible through buggy software, viruses and worms that attacker can exploit to take control of VMs.



Virtual Machine Attacks

Attack	Solution
Security attacks through worms etc. can be exploited to control the VM	Use anti-viruses, anti-spyware programs in guest OS to detect any suspicious activity
Saved state of guest virtual machine as a disk file appears as plaintext to Dom0. Attacker can compromise the integrity and confidentiality	Use encryption and hashing of VMs state before saving

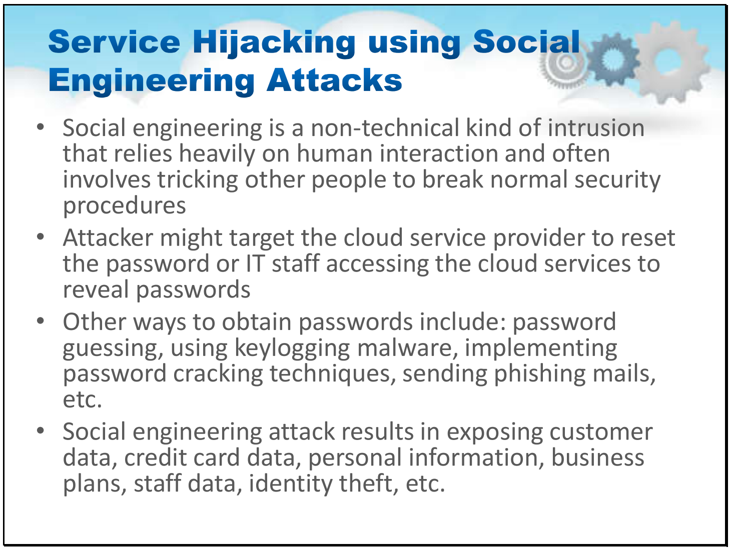


Disk Image Attacks

To create new VM image files, existing VM images can be easily copied which results in VM image sprawl problem, in which a large number of VMs created by customers may be left unnoticed. VM image sprawl results in large management issues of VMs including the security patches. Investigation of VM images on cloud (EC2, VCL) has shown that if patches are not applied, VM images are more vulnerable to attacks, and they may also not fulfill organization security policy. Secondly, some VM images are mostly online, and to patch these images, they will have to be started. This will add to the computation cost of cloud provider. Attacker can access VM checkpoint present in the disk that contain VM physical memory contents and can expose sensitive information of VM state.

Disk Image Attacks

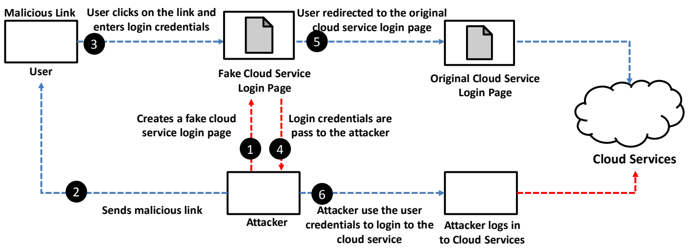
Attack	Solution
VM checkpoint attacks	Checkpoint attacks can be prevented by encrypting the checkpoints or using SPARCR
Unnecessary images can result in a security compromise	Organizations using virtualization must have a policy that manages issues of unnecessary images



Service Hijacking using Social Engineering Attacks

- Social engineering is a non-technical kind of intrusion that relies heavily on human interaction and often involves tricking other people to break normal security procedures
- Attacker might target the cloud service provider to reset the password or IT staff accessing the cloud services to reveal passwords
- Other ways to obtain passwords include: password guessing, using keylogging malware, implementing password cracking techniques, sending phishing mails, etc.
- Social engineering attack results in exposing customer data, credit card data, personal information, business plans, staff data, identity theft, etc.

Service Hijacking using Social Engineering Attacks



Malicious Link

User

3 User clicks on the link and enters login credentials

Fake Cloud Service Login Page

5 User redirected to the original cloud service login page

Original Cloud Service Login Page

Cloud Services

1 Creates a fake cloud service login page

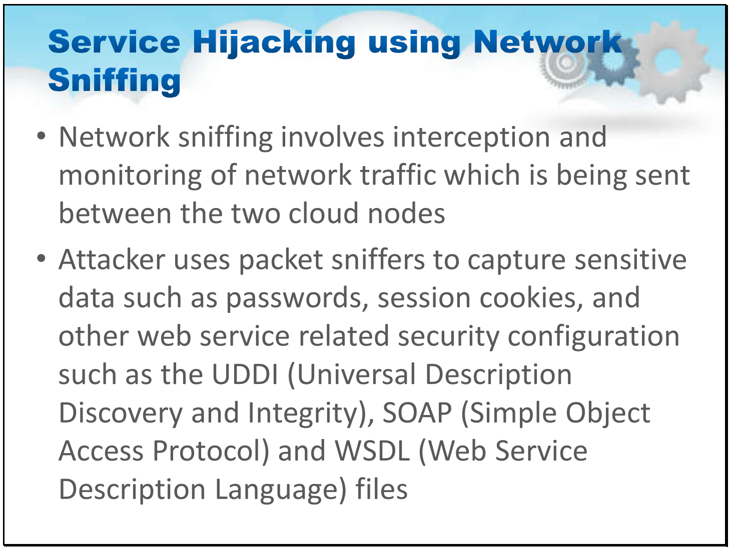
4 Login credentials are pass to the attacker

2 Sends malicious link

Attacker

6 Attacker use the user credentials to login to the cloud service

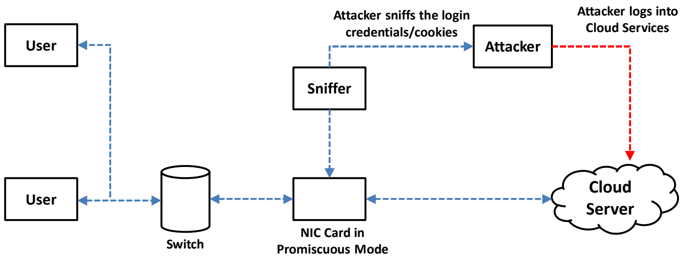
Attacker logs in to Cloud Services

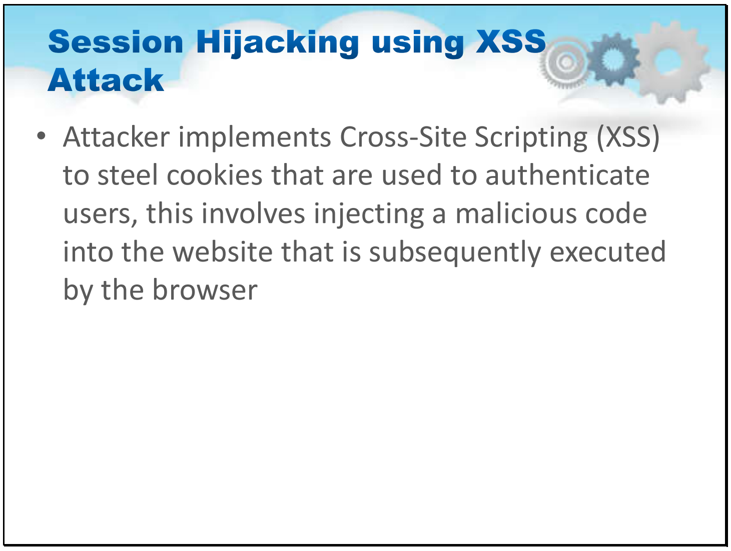


Service Hijacking using Network Sniffing

- Network sniffing involves interception and monitoring of network traffic which is being sent between the two cloud nodes
- Attacker uses packet sniffers to capture sensitive data such as passwords, session cookies, and other web service related security configuration such as the UDDI (Universal Description Discovery and Integrity), SOAP (Simple Object Access Protocol) and WSDL (Web Service Description Language) files

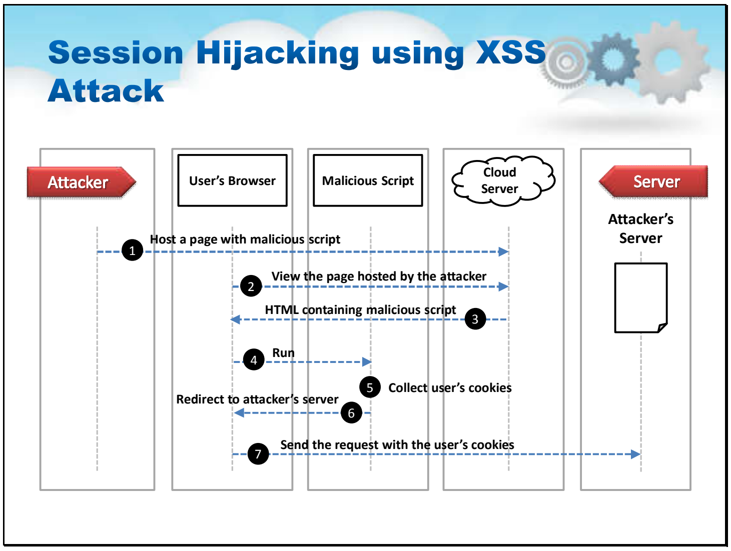
Service Hijacking using Network Sniffing


User
User
Switch
Attacker sniffs the login credentials/cookies
Sniffer
NIC Card in Promiscuous Mode
Attacker
Attacker logs into Cloud Services
Cloud Server



Session Hijacking using XSS Attack

- Attacker implements Cross-Site Scripting (XSS) to steel cookies that are used to authenticate users, this involves injecting a malicious code into the website that is subsequently executed by the browser



Session Hijacking using XSS Attack

Attacker

User's Browser

Malicious Script

Cloud Server

Server

Attacker's Server

1 Host a page with malicious script

2 View the page hosted by the attacker

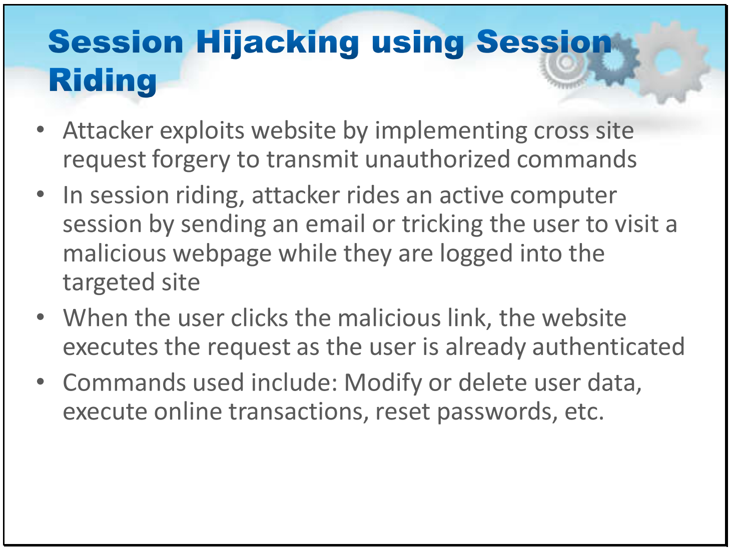
3 HTML containing malicious script

4 Run

5 Collect user's cookies

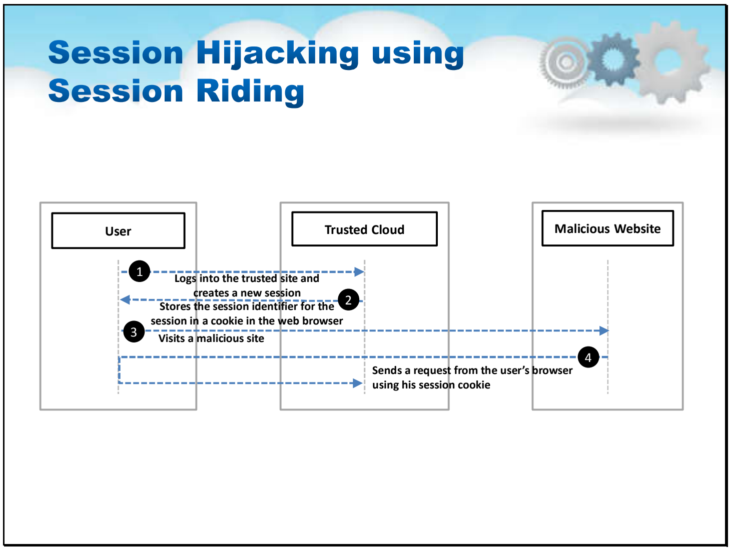
6 Redirect to attacker's server

7 Send the request with the user's cookies



Session Hijacking using Session Riding

- Attacker exploits website by implementing cross site request forgery to transmit unauthorized commands
- In session riding, attacker rides an active computer session by sending an email or tricking the user to visit a malicious webpage while they are logged into the targeted site
- When the user clicks the malicious link, the website executes the request as the user is already authenticated
- Commands used include: Modify or delete user data, execute online transactions, reset passwords, etc.



Session Hijacking using Session Riding

User

Trusted Cloud

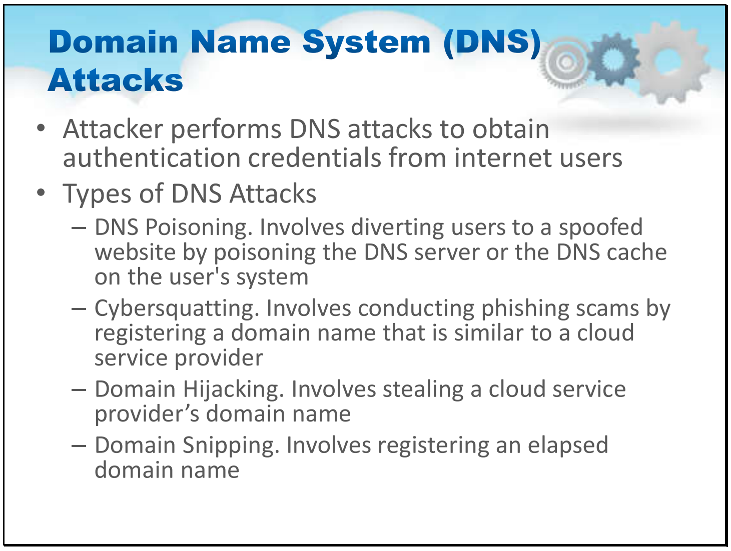
Malicious Website

1 Logs into the trusted site and creates a new session

2 Stores the session identifier for the session in a cookie in the web browser

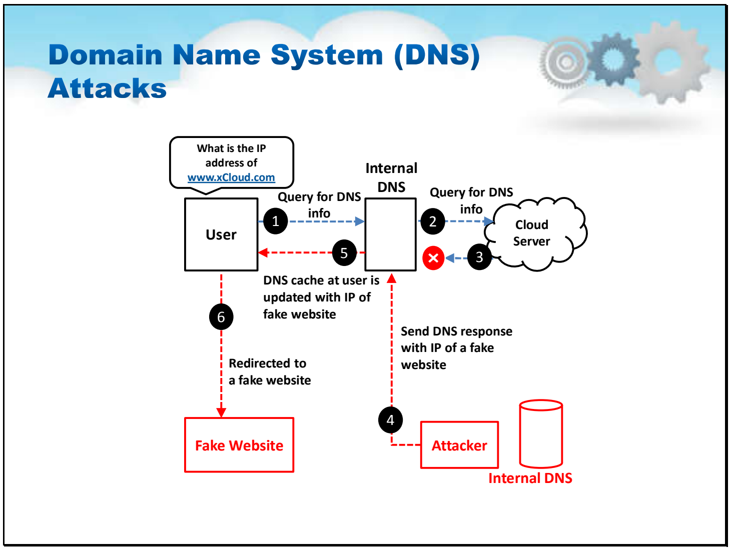
3 Visits a malicious site

4 Sends a request from the user’s browser using his session cookie



Domain Name System (DNS) Attacks

- Attacker performs DNS attacks to obtain authentication credentials from internet users
- Types of DNS Attacks
  - DNS Poisoning. Involves diverting users to a spoofed website by poisoning the DNS server or the DNS cache on the user's system
  - Cybersquatting. Involves conducting phishing scams by registering a domain name that is similar to a cloud service provider
  - Domain Hijacking. Involves stealing a cloud service provider’s domain name
  - Domain Snipping. Involves registering an elapsed domain name



Domain Name System (DNS) Attacks

What is the IP address of www.xCloud.com

User

Internal DNS

Cloud Server

1 Query for DNS info

2 Query for DNS info

3

4 Send DNS response with IP of a fake website

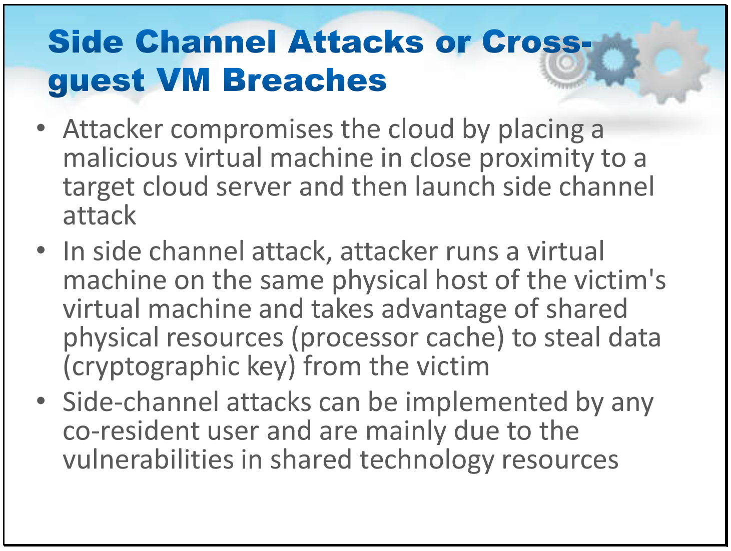
5 DNS cache at user is updated with IP of fake website

6 Redirected to a fake website

Fake Website

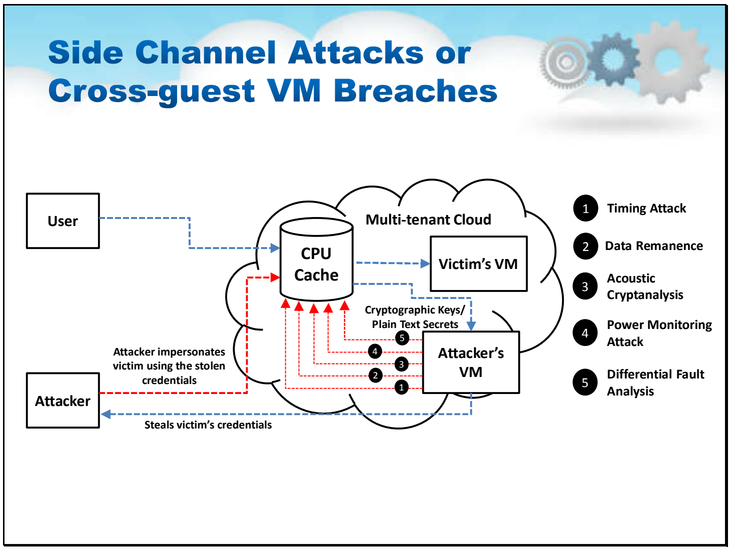
Attacker

Internal DNS



Side Channel Attacks or Cross-guest VM Breaches

- Attacker compromises the cloud by placing a malicious virtual machine in close proximity to a target cloud server and then launch side channel attack
- In side channel attack, attacker runs a virtual machine on the same physical host of the victim's virtual machine and takes advantage of shared physical resources (processor cache) to steal data (cryptographic key) from the victim
- Side-channel attacks can be implemented by any co-resident user and are mainly due to the vulnerabilities in shared technology resources



Side Channel Attacks or Cross-guest VM Breaches

User

Multi-tenant Cloud

CPU Cache

Victim's VM

Cryptographic Keys/ Plain Text Secrets

Attacker's VM

5 4 3 2 1

Attacker impersonates victim using the stolen credentials

Attacker

Steals victim's credentials

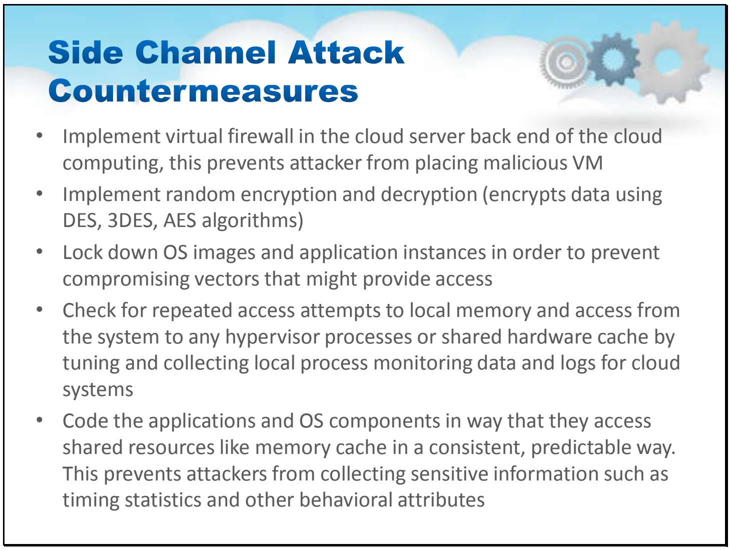
1 Timing Attack

2 Data Remanence

3 Acoustic Cryptanalysis

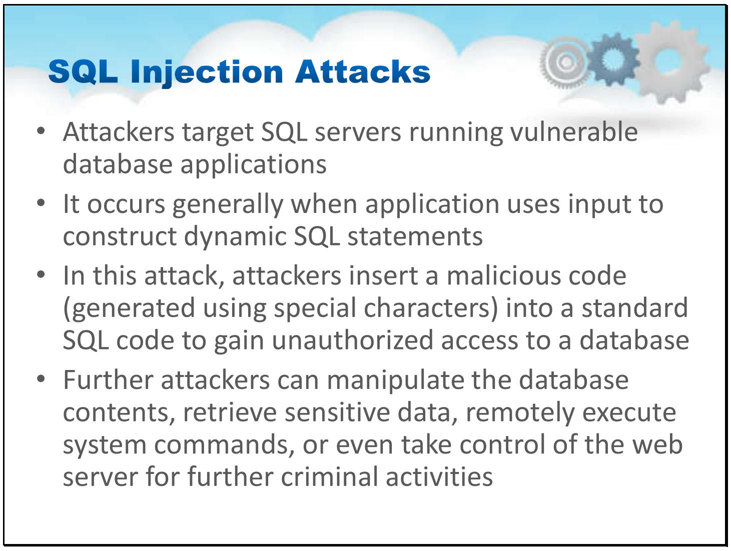
4 Power Monitoring Attack

5 Differential Fault Analysis



Side Channel Attack Countermeasures

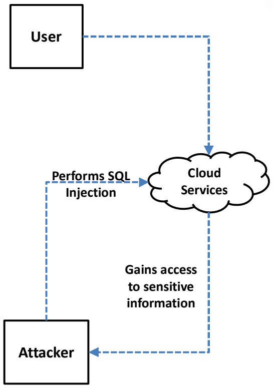
- Implement virtual firewall in the cloud server back end of the cloud computing, this prevents attacker from placing malicious VM
- Implement random encryption and decryption (encrypts data using DES, 3DES, AES algorithms)
- Lock down OS images and application instances in order to prevent compromising vectors that might provide access
- Check for repeated access attempts to local memory and access from the system to any hypervisor processes or shared hardware cache by tuning and collecting local process monitoring data and logs for cloud systems
- Code the applications and OS components in way that they access shared resources like memory cache in a consistent, predictable way. This prevents attackers from collecting sensitive information such as timing statistics and other behavioral attributes

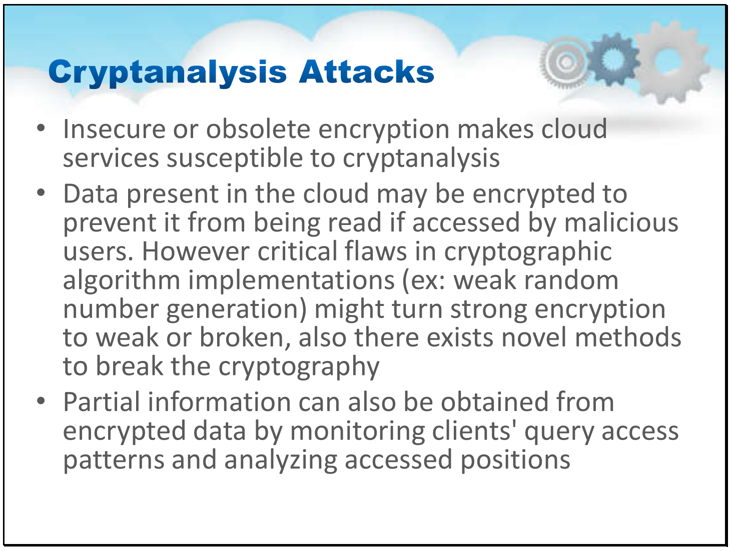


SQL Injection Attacks

- Attackers target SQL servers running vulnerable database applications
- It occurs generally when application uses input to construct dynamic SQL statements
- In this attack, attackers insert a malicious code (generated using special characters) into a standard SQL code to gain unauthorized access to a database
- Further attackers can manipulate the database contents, retrieve sensitive data, remotely execute system commands, or even take control of the web server for further criminal activities

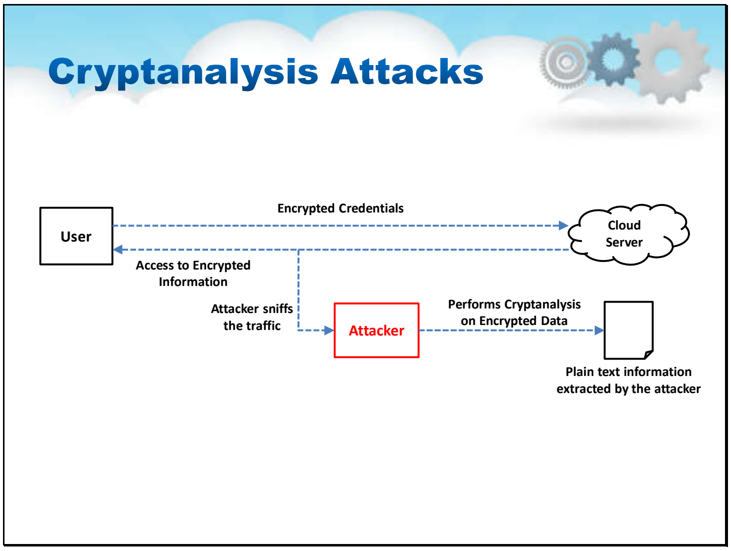
SQL Injection Attacks





Cryptanalysis Attacks

- Insecure or obsolete encryption makes cloud services susceptible to cryptanalysis
- Data present in the cloud may be encrypted to prevent it from being read if accessed by malicious users. However critical flaws in cryptographic algorithm implementations (ex: weak random number generation) might turn strong encryption to weak or broken, also there exists novel methods to break the cryptography
- Partial information can also be obtained from encrypted data by monitoring clients' query access patterns and analyzing accessed positions



Cryptanalysis Attacks

User

Encrypted Credentials

Cloud Server

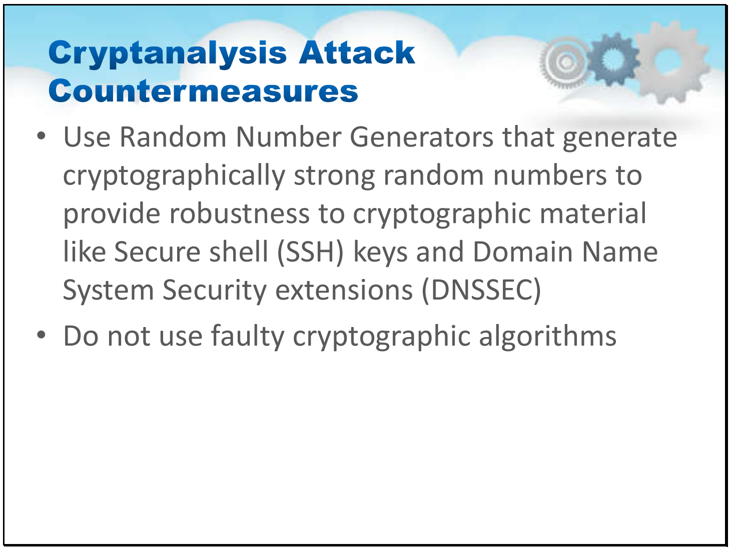
Access to Encrypted Information

Attacker sniffs the traffic

Attacker

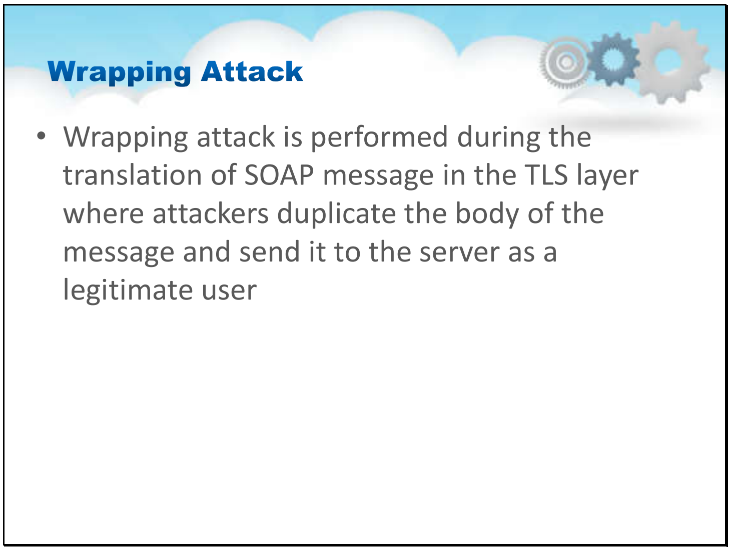
Performs Cryptanalysis on Encrypted Data

Plain text information extracted by the attacker



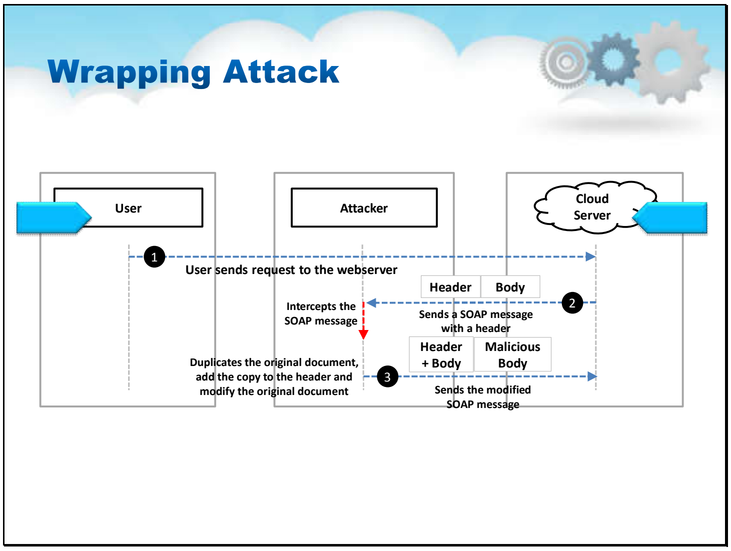
Cryptanalysis Attack Countermeasures

- Use Random Number Generators that generate cryptographically strong random numbers to provide robustness to cryptographic material like Secure shell (SSH) keys and Domain Name System Security extensions (DNSSEC)
- Do not use faulty cryptographic algorithms



Wrapping Attack

- Wrapping attack is performed during the translation of SOAP message in the TLS layer where attackers duplicate the body of the message and send it to the server as a legitimate user



Wrapping Attack

User

Attacker

Cloud Server

1 User sends request to the webserver

Header | Body

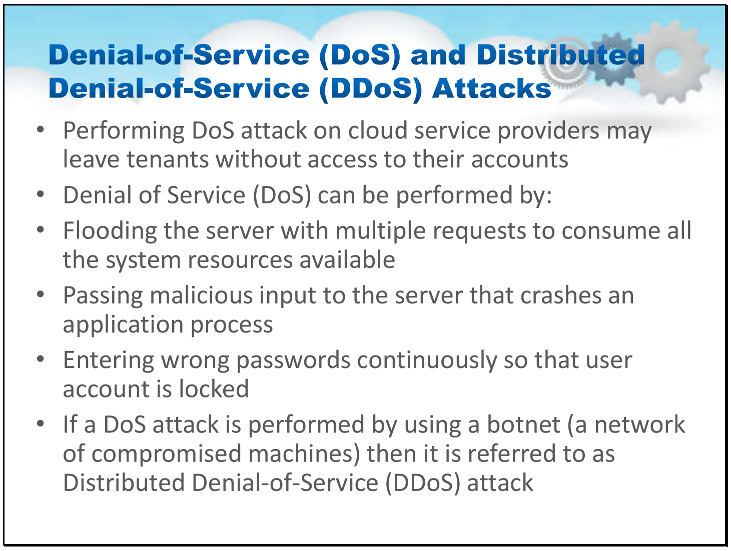
2 Sends a SOAP message with a header

Intercepts the SOAP message

Header + Body | Malicious Body

3 Duplicates the original document, add the copy to the header and modify the original document

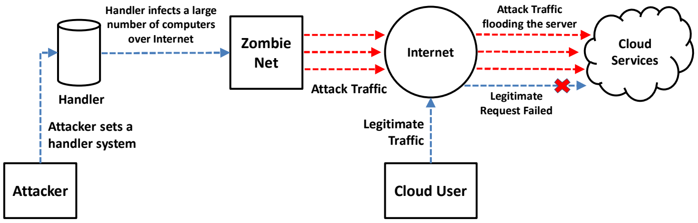
Sends the modified SOAP message

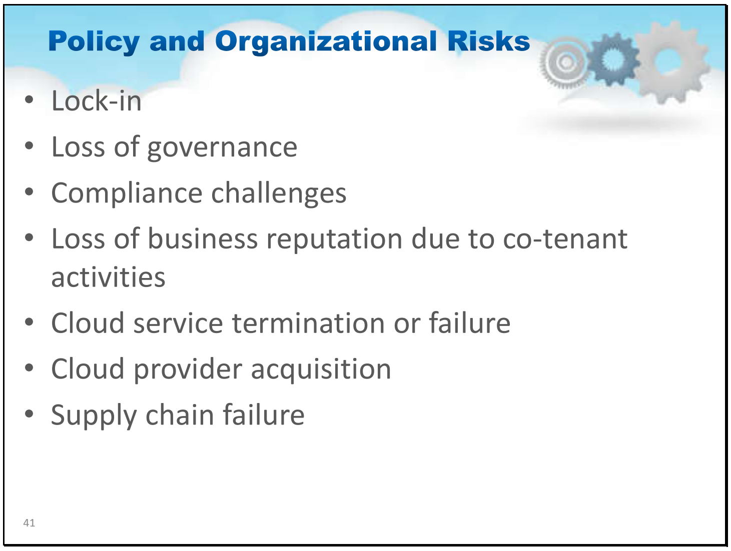


Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks

- Performing DoS attack on cloud service providers may leave tenants without access to their accounts
- Denial of Service (DoS) can be performed by:
- Flooding the server with multiple requests to consume all the system resources available
- Passing malicious input to the server that crashes an application process
- Entering wrong passwords continuously so that user account is locked
- If a DoS attack is performed by using a botnet (a network of compromised machines) then it is referred to as Distributed Denial-of-Service (DDoS) attack

Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks


Handler infects a large number of computers over Internet
Zombie Net
Internet
Attack Traffic flooding the server
Cloud Services
Handler
Attack Traffic
Legitimate Request Failed
Attacker sets a handler system
Legitimate Traffic
Attacker
Cloud User



Policy and Organizational Risks

- Lock-in
- Loss of governance
- Compliance challenges
- Loss of business reputation due to co-tenant activities
- Cloud service termination or failure
- Cloud provider acquisition
- Supply chain failure

Lock-In
The potential dependency on a particular cloud provider, depending on the provider’ s commitments, may lead to a catastrophic business failure should the cloud provider go bankrupt or the content and application migration path to another provider is too costly. There is little or no incentive for cloud providers to make migrating to another provider easy if not contractually bound to do so.

Loss of Governance
By using cloud infrastructures, the client necessarily cedes control to the cloud provider on a number of issues which may affect security. This could have a potentially severe impact on the organization’ s strategy and therefore on the capacity to meet its mission and goals. The loss of control and governance could lead to the impossibility of complying with the security requirements, a lack of confidentiality, integrity and availability of data, and a deterioration of performance and quality of service, not to mention the introduction of compliance challenges.

Compliance Challenges
Certain companies migrating to the cloud might have the need to meet certain industry standards or regulatory requirements, such as the Payment Card Industry Data

Security Standard (PCI DSS). Migrating to the cloud could compromise these business needs if the cloud provider cannot provide evidence of their own compliance to the relevant requirements or if the provider does not permit audits by the customer.

Loss of Business Reputation Due To Co-Tenant Activities

Resource sharing can give rise to problems when the shared resources’ reputation becomes tainted by a bad neighbor’ s activities. This would also include that certain measures are taken to mitigate, such as internet protocol (IP) address blocking and equipment confiscation.

Cloud Service Termination or Failure

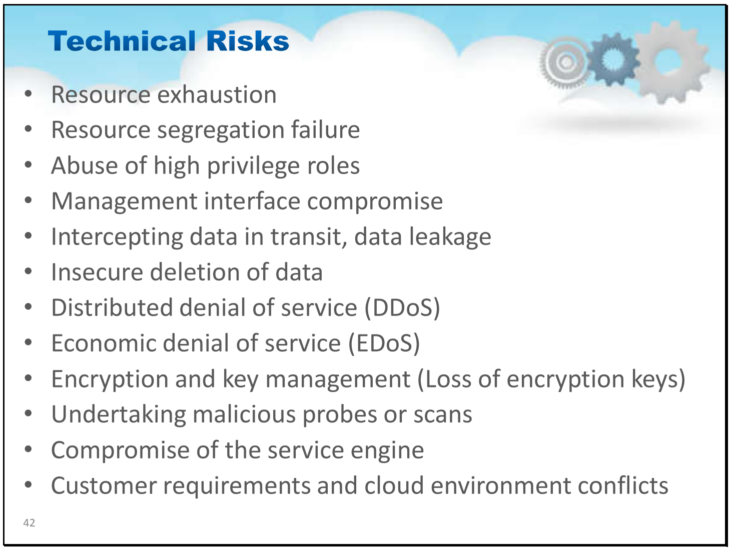
If the cloud provider faces the risk of going out of business due to financial, legal, or other reasons, the customer could suffer from loss or deterioration of service delivery performance, and quality of service, as well as a loss of investment .

Cloud Provider Acquisition

The acquisition of the cloud provider could increase the possibility of a strategic change and may put previous agreements at risk. This could make it impossible to comply with existing security requirements. The final impact could be damaging for crucial assets, such as the organization’ s reputation, customer or patient trust, and employee loyalty and experience.

Supply Chain Failure

A cloud computing provider can outsource certain specialized tasks of its infrastructure to third parties. In such a situation the level of security of the cloud provider may depend on the level of security of each one of the links and the level of dependency of the cloud provider on the third party. In general, a lack of transparency in the contract can be a problem for the whole system.



Technical Risks

- Resource exhaustion
- Resource segregation failure
- Abuse of high privilege roles
- Management interface compromise
- Intercepting data in transit, data leakage
- Insecure deletion of data
- Distributed denial of service (DDoS)
- Economic denial of service (EDoS)
- Encryption and key management (Loss of encryption keys)
- Undertaking malicious probes or scans
- Compromise of the service engine
- Customer requirements and cloud environment conflicts

Resource Exhaustion
Inaccurate modeling of customer demands by the cloud provider can lead to service unavailability, access control compromise, and economic and reputation losses due to resource exhaustion. The customer takes a level of calculated risk in allocating all the resources of a cloud service, because resources are allocated according to statistical projections.

Resource Segre gat ion Failure
This class of risks includes the failure of mechanisms separating storage, memory, routing, and even reputation between different tenants of the shared infrastructure (guest-hopping attacks, SQL injection attacks exposing multiple customers' data, and side-channel attacks). The likelihood of this incident scenario depends on the cloud model adopted by the customer. It is less likely to occur for private cloud customers compared to public cloud customers.

Abuse of High Privilege Roles
The malicious activities of an insider could potentially have an impact on the confidentiality, integrity, and availability of all kinds of data, IP, services, and therefore indirectly on the organization's reputation, customer trust,

and the experiences of employees. This can be considered especially important in the case of cloud computing due to the fact that cloud architectures necessitate certain roles , which are extremely high- risk. Examples of such roles include the cloud provider' s system administrators and auditors and managed security service providers dealing with intrusion detection report s and incident response.

Management Interface Compromise

The customer management interfaces of public cloud providers are Inter net accessible and mediate access to larger sets of resources (than traditional hosting providers) . They also pose an increased risk especially when combined with remote access and web browser vulnerabilities.

Intercepting Data in Transit, Data Leakage

Cloud computing, being a distributed architecture, implies more data in trans it than traditional infrastructures. Sniffing, spoofing, man-in-the-middle, side channel , and replay attacks should be considered as possible threat sources.

Insecure Deletion of Data

Whenever a provider is changed, resources are scaled down , physical hardware is reallocated, and data may be available beyond the life time specified in the security policy. Where true data wiping is required, special procedures must be followed and this may not be supported by the cloud provider.

Distributed Denial of Service

A common method of attack involves saturating the target environment with external communications requests, such that it cannot respond to legitimate traffic, or responds so slowly as to be rendered effectively unavailable. This can result in financial and economic losses.

Economic Denial of Service

EDoS destroys economic resources; the worst-case scenario would be the bankruptcy of the customer or a serious economic imp act. The following scenarios are possible: An attacker can use an account and uses the customer' s resources for his own gain or in order to damage the customer economically. The customer has not set effective limits on the use of paid resources and experiences unexpected loads on these resources. An attacker can use a public channel to deplete the customer’ s metered resources. For example, where the customer pays per HTTP request, a DDoS attack can have this effect .

Encrypt ion and Key Management (Loss of Encrypt ion Keys)

This risk includes the disclosure of secret keys (SSL, file encryption, customer private keys) or passwords to malicious parties. It also includes the loss or corruption of those keys, or their unauthorized use for authentication and nonrepudiation (digital signature).

Undertaking Malicious Probes or Scans

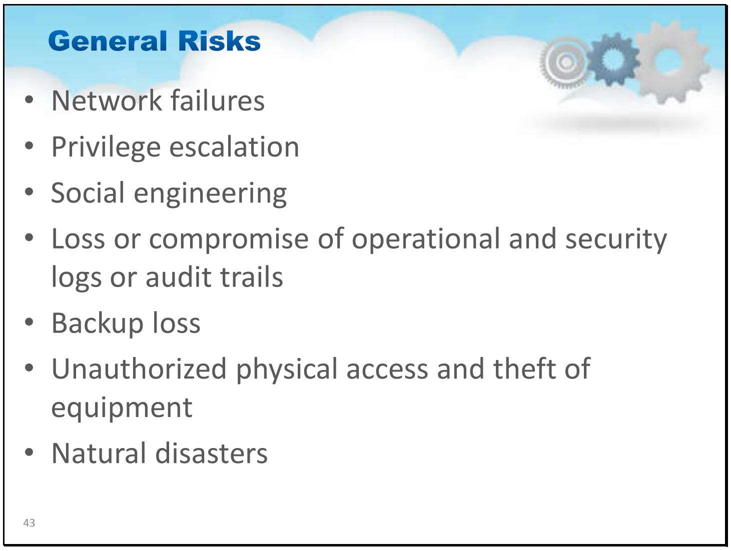
Malicious probes or scanning, as well as network mapping, are indirect threats to the assets being considered. They can be used to collect information in the context o f a hacking attempt . A possible impact could be a loss of confidentiality, integrity, and availability of service and data.

Compromise of the Service Engine

Each cloud architecture relies on a highly specialize d platform and the service engine . The service engine sit s above the physical hardware resources and manages customer resources at different levels o f abstraction. For example, in IaaS clouds this software component can be the hypervisor. Like any other software layer , the service engine code can have vulnerabilities and is prone to attacks or unexpected failure. Cloud providers must set out a clear segregation of responsibilities that articulates the minim um actions customers must undertake.

Customer Requirements and Cloud Environment Conflicts

Cloud providers must set out a clear segregation of responsibilities that articulates the minimum actions customers must undertake. The failure of the customers to properly secure their environments may pose a vulnerability to the cloud platform if the cloud provider has not taken the necessary steps to provide isolation. Cloud providers should further articulate their isolation mechanisms and provide best practice guidelines to assist customers to secure their resources.



General Risks

- Network failures
- Privilege escalation
- Social engineering
- Loss or compromise of operational and security logs or audit trails
- Backup loss
- Unauthorized physical access and theft of equipment
- Natural disasters

43

Network Failures
This risk is one of the highest risks since it directly affects service delivery. It exists due to network misconfiguration, system vulnerabilities, lack of resource isolation, and poor or untested business continuity (BC) and disaster recovery (DR) plans. Network traffic modification can also be a risk for a customer and cloud provider if provisioning isn't done properly or there are no traffic encryption or vulnerability assessments.

Privilege Escalation
Although there is a low probability of exploitation, privilege escalation can cause loss of customer data, and access control. A malicious entity can therefore take control of large portions of the cloud platform. The risk manifests itself due to authentication, authorization, and other access control vulnerabilities, hypervisor vulnerabilities (cloudbursting), and misconfiguration.

Social Engineering
This risk is one of the most disregarded since most technical staff focus on the nonhuman aspects of their platforms. The exploitation of this risk has caused loss of reputation for cloud service providers, such as Amazon and Apple, due to the publicity of the events. This risk can be

easily be minimized by security awareness training, proper user provisioning, resource isolation, data encryption, and proper physical security procedures.

Loss or Compromise of Operational and Security Logs or Audit Trails

Operational logs can be vulnerable due to lack of policy or poor procedures for logs collection. This would also include retention, access management vulnerabilities, user deprovisioning vulnerabilities, lack of forensic readiness, and OS vulnerabilities.

Back up Loss

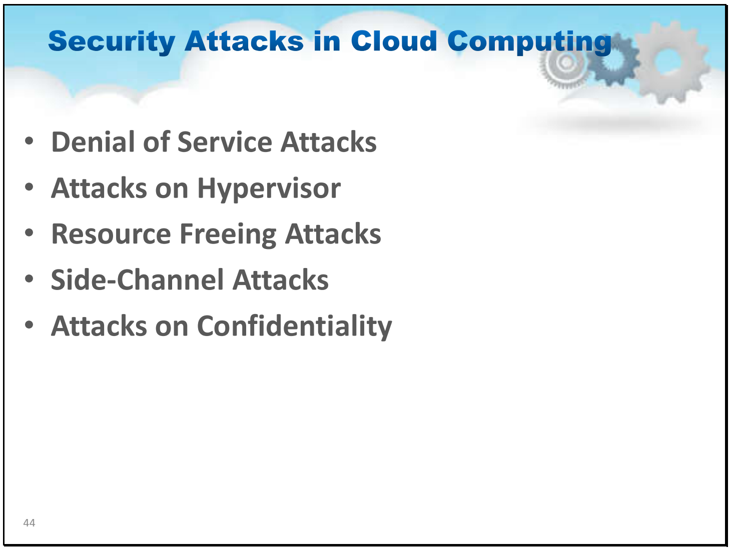
This high impact risk affects company reputation, all backed up data, and service delivery. It also occurs due to inadequate physical security procedures, access management vulnerabilities, and user deprovisioning vulnerabilities.

Unauthorized Physical Access and Theft of Equipment

The probability of malicious actors gaining access to a physical location is very low, but in the event of such occurrence, the impact to the cloud provider and its customers is very high. It can affect company reputation, and data hosted on premises and the security risk it brings is due to inadequate physical security procedures.

Natural Disasters

This risk is often ignored but can have a high impact on the businesses involved in the event of its occurrence. If a business has a poor or untested continuity and DR plan or lacks one, their reputation, data, and service delivery can be severely compromised.



Security Attacks in Cloud Computing

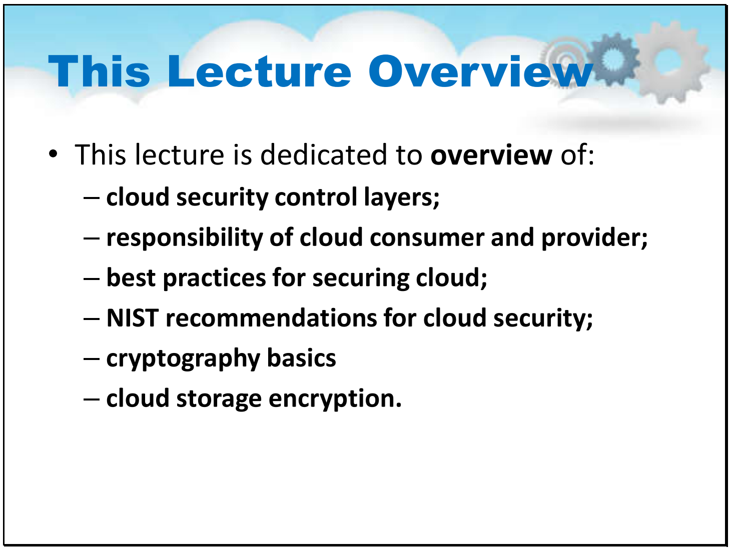
- Denial of Service Attacks
- Attacks on Hypervisor
- Resource Freeing Attacks
- Side-Channel Attacks
- Attacks on Confidentiality

Cloud Computing

Module – Cloud Computing security

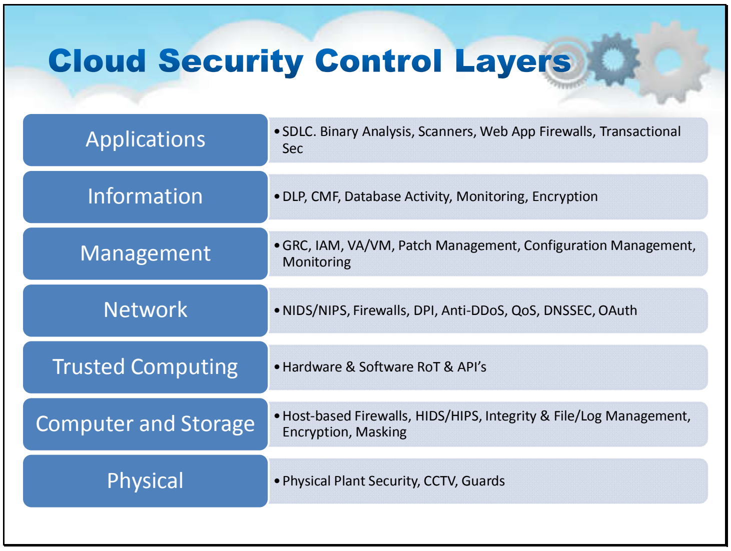
Lecture 2. Cloud Computing Security Considerations

Lecture 2.Cloud Computing Security Considerations



This Lecture Overview

- This lecture is dedicated to **overview** of:
  - – **cloud security control layers;**
  - – **responsibility of cloud consumer and provider;**
  - – **best practices for securing cloud;**
  - – **NIST recommendations for cloud security;**
  - – **cryptography basics**
  - – **cloud storage encryption.**



Cloud Security Control Layers

Applications	• SDLC. Binary Analysis, Scanners, Web App Firewalls, Transactional Sec
Information	• DLP, CMF, Database Activity, Monitoring, Encryption
Management	• GRC, IAM, VA/VM, Patch Management, Configuration Management, Monitoring
Network	• NIDS/NIPS, Firewalls, DPI, Anti-DDoS, QoS, DNSSEC, OAuth
Trusted Computing	• Hardware & Software RoT & API's
Computer and Storage	• Host-based Firewalls, HIDS/HIPS, Integrity & File/Log Management, Encryption, Masking
Physical	• Physical Plant Security, CCTV, Guards

Cloud security control layers

Cloud Security Control Layers

Application Layer

There are several security mechanisms, devices, and policies that provide support at different cloud security controls layers. At the Application layer, Web application firewalls are deployed to filter the traffic and observe the behavior of traffic. Similarly, Systems Development Life Cycle (SDLC), Binary Code Analysis, Transactional Security provide security for online transactions, and script analysis, etc.

Information

In Cloud Computing, to provide confidentiality and integrity of information that is being communicated between client and server, different policies are configured to monitor any data loss. These policies include Data Loss Prevention (DLP) and Content Management Framework (CMF). Data Loss Prevention (DLP) is the feature which offers to prevent the leakage of information to outside the network. Traditionally this information may include company or organizations confidential information, proprietary, financial and other secret information. Data Loss Prevention feature also ensures the enforcement of compliance with the rules and regulations using Data Loss Prevention policies to prevent the user from intentionally or unintentionally sending this confidential information.

Management

Security of Cloud Computing regarding management is performed by different approaches such as Governance, Risk Management, and Compliance (GRC), Identity and Access Management (IAM), Patch and Configuration management. These approaches help to control the secure access to the resources and manage them.

Network layer

There are some solutions available to secure the network layer in cloud computing such as the deployment of Next-Generation IDS/IPS devices, Next-Generation Firewalls, DNSSec, Anti-DDoS, OAuth and Deep Packet Inspection (DPI), etc. Next-Generation Intrusion Prevention System, known as NGIPS, is one of the efficiently-proactive components in the Integrated Threat Security Solution. NGIPS provide stronger security layer with deep visibility, enhanced security intelligence and advanced protection against emerging threat to secure complex infrastructures of networks.

Trusted Computing

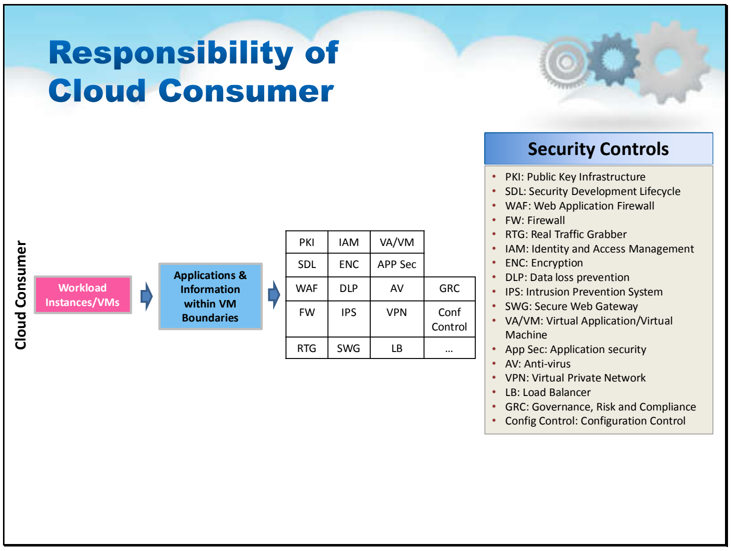
The root of Trust (RoT) is established by validating each component of hardware and software from the end entity up to the root certificate. It is intended to ensure that only trusted software and hardware can be used while still retaining flexibility.

Computer and Storage

Computing and Storage in cloud computing can be secured by implementing Host-based Intrusion Detection or Prevention Systems HIDS/HIPS. Configuring Integrity check, File system monitoring and Log File Analysis, Connection Analysis, Kernel Level detection, Encrypting the storage, etc. Host-based IPS/IDS is normally deployed for the protection of specific host machine, and it works closely with the Operating System Kernel of the host machine. It creates a filtering layer and filters out any malicious application call to the OS.

Physical Security

Physical Security is always required on priority to secure anything. As it is also the first layer OSI model, if the device is not physically secured, any sort of security configuration will not be effective. Physical security includes protection against man-made attacks such as theft, damage, unauthorized physical access as well as environmental impact such as rain, dust, power failure, fire, etc.



Responsibility of Cloud Consumer

PKI	IAM	VA/VM	
SDL	ENC	APP Sec	
WAF	DLP	AV	GRC
FW	IPS	VPN	Conf Control
RTG	SWG	LB	...

Security Controls

- PKI: Public Key Infrastructure
- SDL: Security Development Lifecycle
- WAF: Web Application Firewall
- FW: Firewall
- RTG: Real Traffic Grabber
- IAM: Identity and Access Management
- ENC: Encryption
- DLP: Data loss prevention
- IPS: Intrusion Prevention System
- SWG: Secure Web Gateway
- VA/VM: Virtual Application/Virtual Machine
- App Sec: Application security
- AV: Anti-virus
- VPN: Virtual Private Network
- LB: Load Balancer
- GRC: Governance, Risk and Compliance
- Config Control: Configuration Control

Responsibility of cloud consumer and provider

Responsibilities of a cloud service consumer include to meet the following security
controls:

 Public Key Infrastructure (PKI).

 Security Development Life Cycle (SDLC).

 Web Application Firewall (WAF).

 Firewall

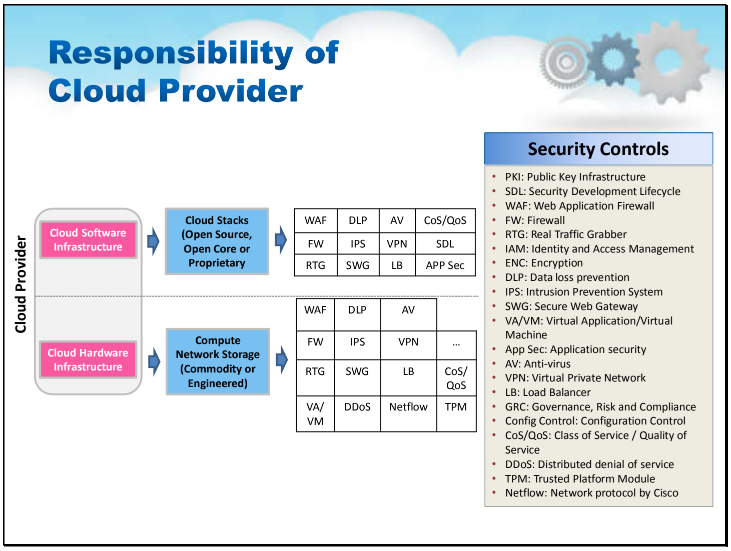
 Encryption.

 Intrusion Prevention Systems

 Secure Web Gateway

 Application Security

 Virtual Private Network (VPN) and others.



Responsibility of Cloud Provider

Cloud Provider
Cloud Software Infrastructure
Cloud Stacks (Open Source, Open Core or Proprietary
Cloud Hardware Infrastructure
Compute Network Storage (Commodity or Engineered)

WAF	DLP	AV	CoS/QoS
FW	IPS	VPN	SDL
RTG	SWG	LB	APP Sec

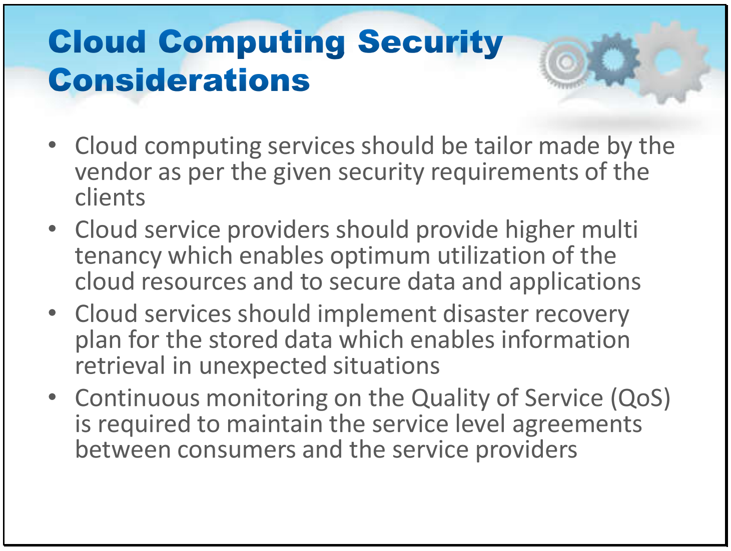
WAF	DLP	AV	
FW	IPS	VPN	...
RTG	SWG	LB	CoS/ QoS
VA/ VM	DDoS	Netflow	TPM

Security Controls

- PKI: Public Key Infrastructure
- SDL: Security Development Lifecycle
- WAF: Web Application Firewall
- FW: Firewall
- RTG: Real Traffic Grabber
- IAM: Identity and Access Management
- ENC: Encryption
- DLP: Data loss prevention
- IPS: Intrusion Prevention System
- SWG: Secure Web Gateway
- VA/VM: Virtual Application/Virtual Machine
- App Sec: Application security
- AV: Anti-virus
- VPN: Virtual Private Network
- LB: Load Balancer
- GRC: Governance, Risk and Compliance
- Config Control: Configuration Control
- CoS/QoS: Class of Service / Quality of Service
- DDoS: Distributed denial of service
- TPM: Trusted Platform Module
- Netflow: Network protocol by Cisco

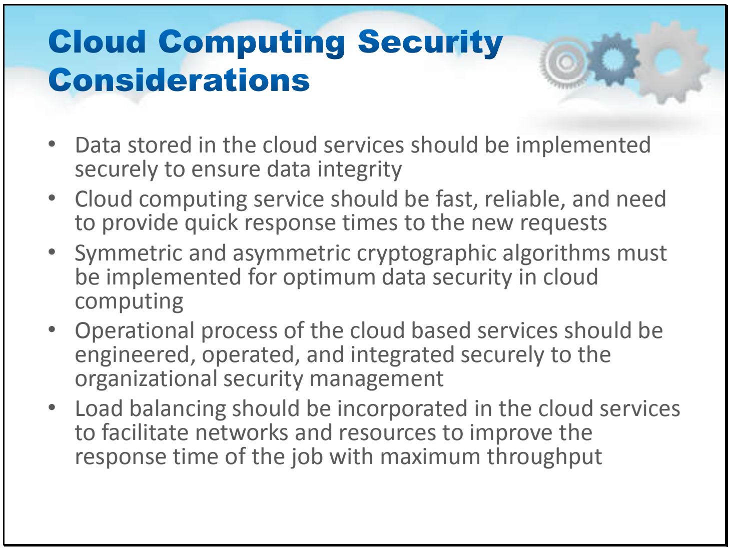
Cloud Service Provider
Responsibilities of a cloud service provider include to meet the following security controls:

- Web Application Firewall (WAF).
- Real Traffic Grabber (RTG)
- Firewall
- Data Loss Prevention (DLP)
- Intrusion Prevention Systems
- Secure Web Gateway (SWG)
- Application Security (App Sec)
- Virtual Private Network (VPN)
- Load Balancer
- CoS/QoS
- Trusted Platform Module
- Netflow and others.



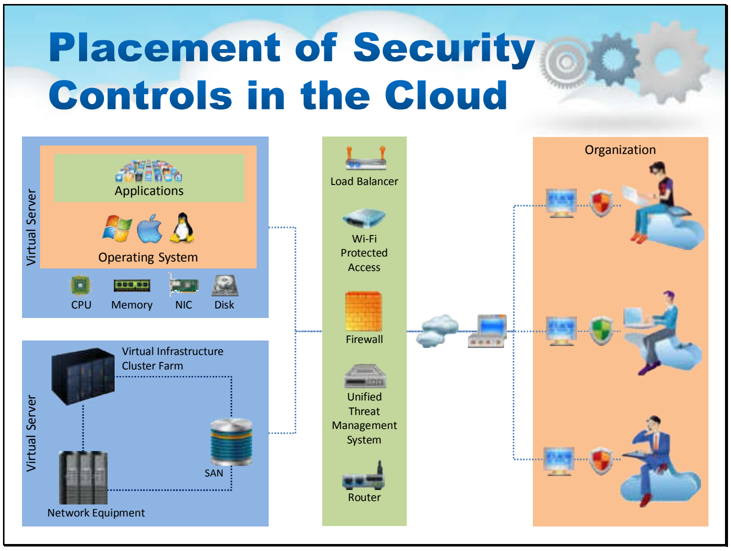
Cloud Computing Security Considerations

- Cloud computing services should be tailor made by the vendor as per the given security requirements of the clients
- Cloud service providers should provide higher multi tenancy which enables optimum utilization of the cloud resources and to secure data and applications
- Cloud services should implement disaster recovery plan for the stored data which enables information retrieval in unexpected situations
- Continuous monitoring on the Quality of Service (QoS) is required to maintain the service level agreements between consumers and the service providers



Cloud Computing Security Considerations

- Data stored in the cloud services should be implemented securely to ensure data integrity
- Cloud computing service should be fast, reliable, and need to provide quick response times to the new requests
- Symmetric and asymmetric cryptographic algorithms must be implemented for optimum data security in cloud computing
- Operational process of the cloud based services should be engineered, operated, and integrated securely to the organizational security management
- Load balancing should be incorporated in the cloud services to facilitate networks and resources to improve the response time of the job with maximum throughput



Placement of Security Controls in the Cloud

Virtual Server

Applications

Operating System

CPU

Memory

NIC

Disk

Virtual Server

Virtual Infrastructure Cluster Farm

SAN

Network Equipment

Load Balancer

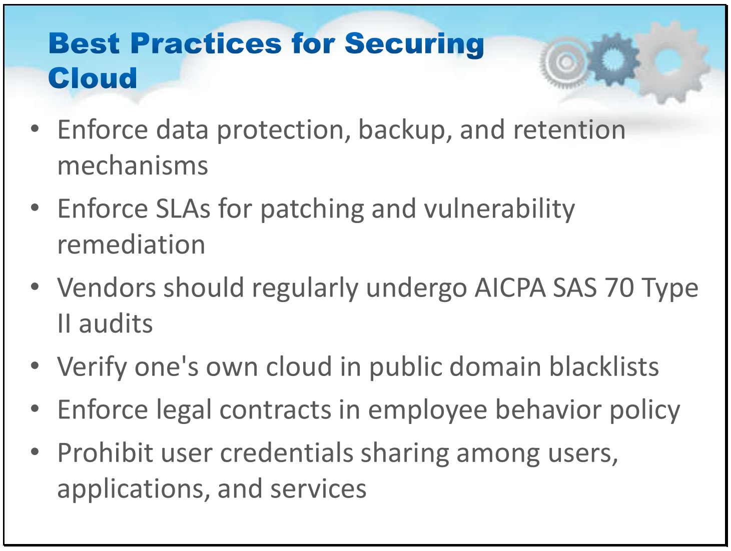
Wi-Fi Protected Access

Firewall

Unified Threat Management System

Router

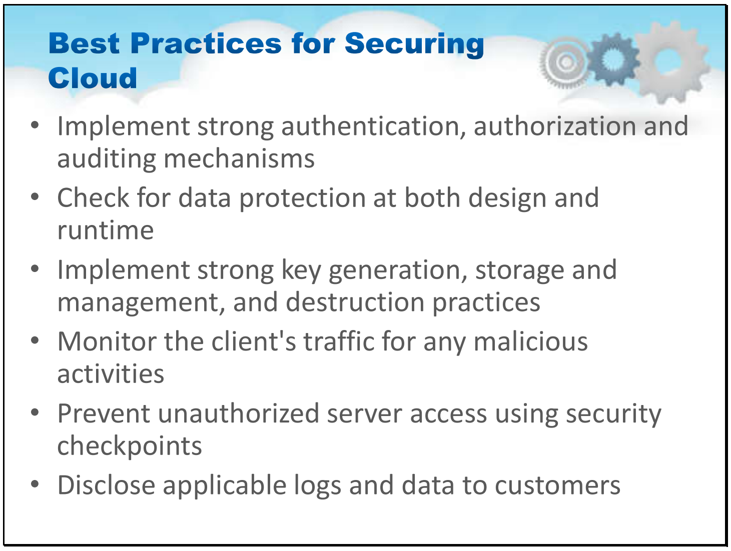
Organization



Best Practices for Securing Cloud

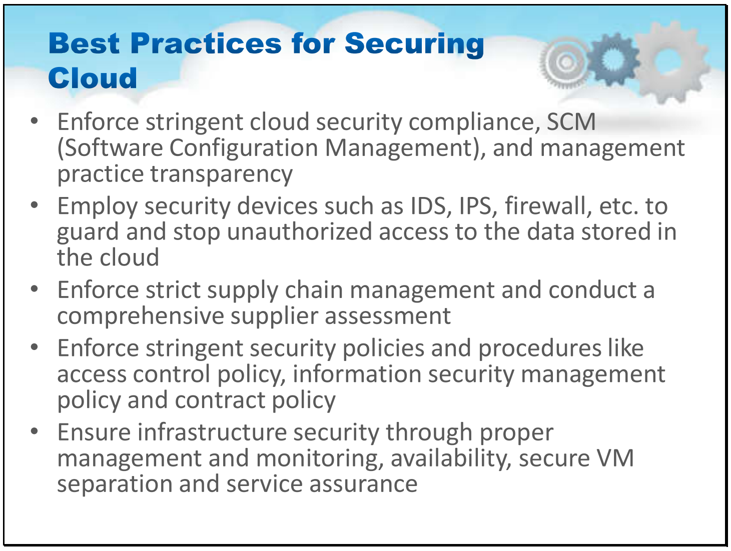
- Enforce data protection, backup, and retention mechanisms
- Enforce SLAs for patching and vulnerability remediation
- Vendors should regularly undergo AICPA SAS 70 Type II audits
- Verify one's own cloud in public domain blacklists
- Enforce legal contracts in employee behavior policy
- Prohibit user credentials sharing among users, applications, and services

Best practices for securing cloud



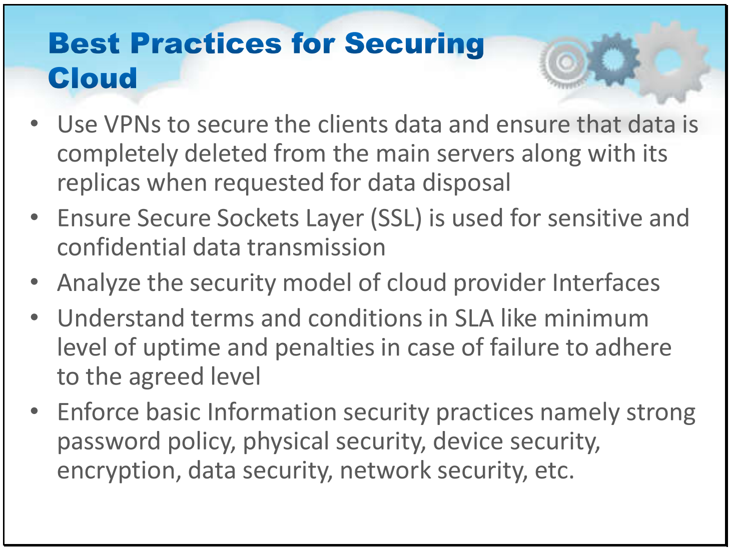
Best Practices for Securing Cloud

- Implement strong authentication, authorization and auditing mechanisms
- Check for data protection at both design and runtime
- Implement strong key generation, storage and management, and destruction practices
- Monitor the client's traffic for any malicious activities
- Prevent unauthorized server access using security checkpoints
- Disclose applicable logs and data to customers



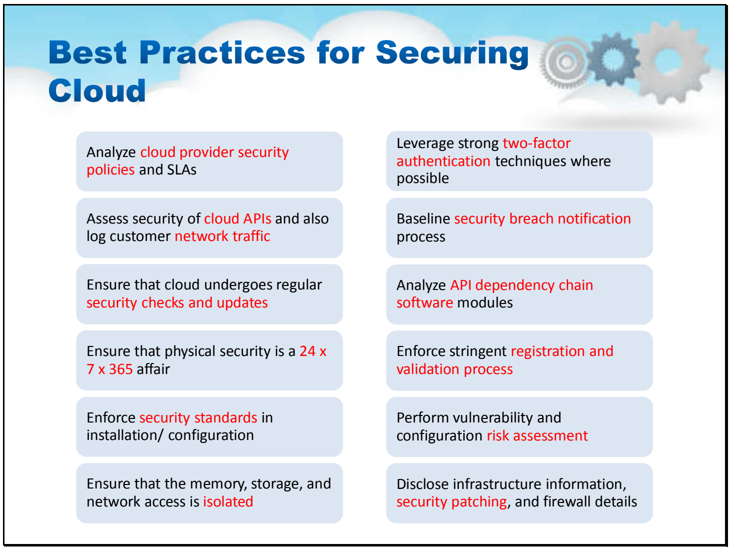
Best Practices for Securing Cloud

- Enforce stringent cloud security compliance, SCM (Software Configuration Management), and management practice transparency
- Employ security devices such as IDS, IPS, firewall, etc. to guard and stop unauthorized access to the data stored in the cloud
- Enforce strict supply chain management and conduct a comprehensive supplier assessment
- Enforce stringent security policies and procedures like access control policy, information security management policy and contract policy
- Ensure infrastructure security through proper management and monitoring, availability, secure VM separation and service assurance



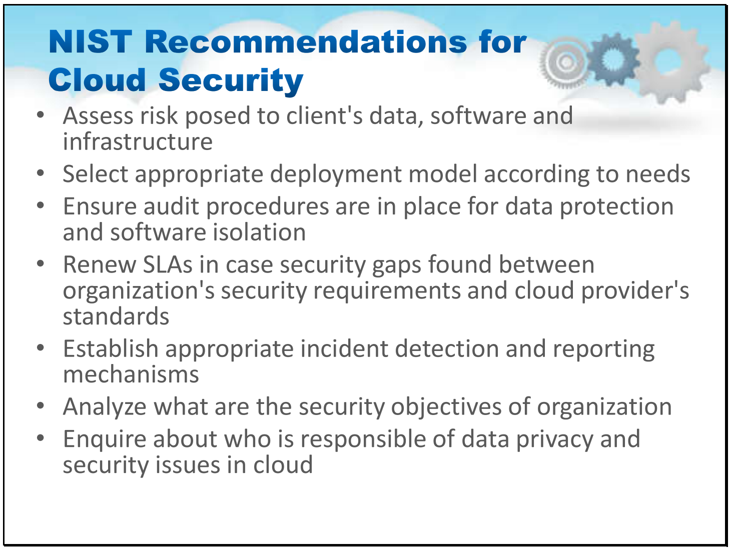
Best Practices for Securing Cloud

- Use VPNs to secure the clients data and ensure that data is completely deleted from the main servers along with its replicas when requested for data disposal
- Ensure Secure Sockets Layer (SSL) is used for sensitive and confidential data transmission
- Analyze the security model of cloud provider Interfaces
- Understand terms and conditions in SLA like minimum level of uptime and penalties in case of failure to adhere to the agreed level
- Enforce basic Information security practices namely strong password policy, physical security, device security, encryption, data security, network security, etc.



Best Practices for Securing Cloud

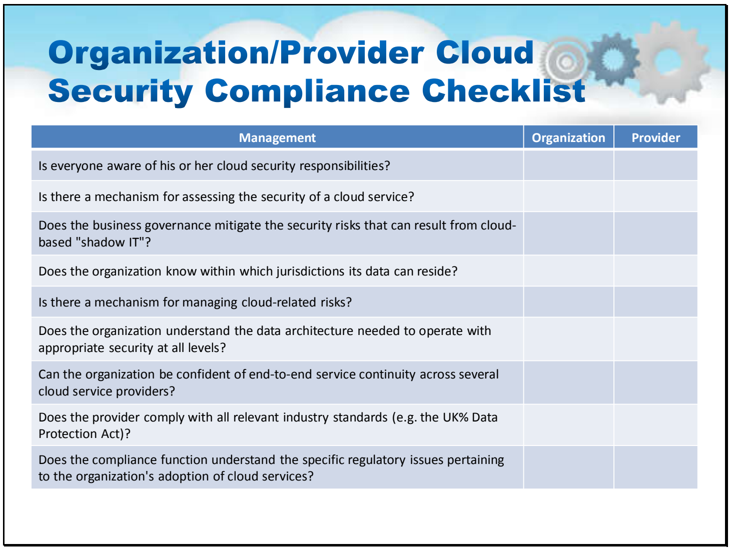
- Analyze cloud provider security policies and SLAs
- Assess security of cloud APIs and also log customer network traffic
- Ensure that cloud undergoes regular security checks and updates
- Ensure that physical security is a 24 x 7 x 365 affair
- Enforce security standards in installation/ configuration
- Ensure that the memory, storage, and network access is isolated
- Leverage strong two-factor authentication techniques where possible
- Baseline security breach notification process
- Analyze API dependency chain software modules
- Enforce stringent registration and validation process
- Perform vulnerability and configuration risk assessment
- Disclose infrastructure information, security patching, and firewall details



NIST Recommendations for Cloud Security

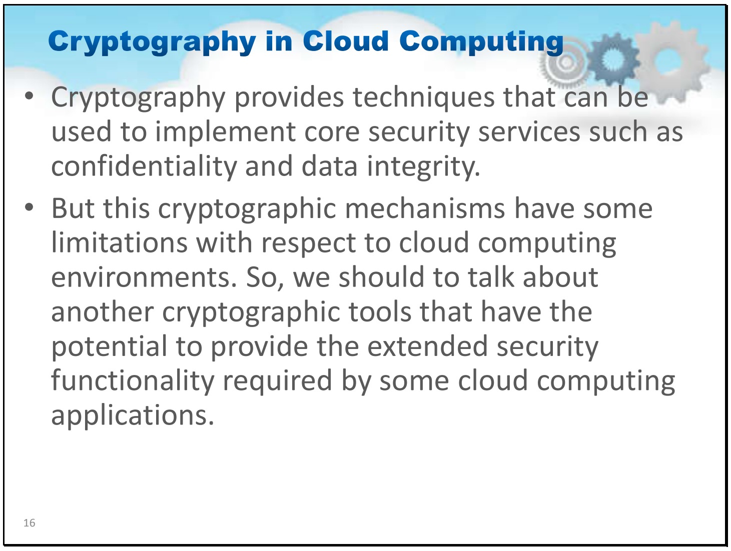
- Assess risk posed to client's data, software and infrastructure
- Select appropriate deployment model according to needs
- Ensure audit procedures are in place for data protection and software isolation
- Renew SLAs in case security gaps found between organization's security requirements and cloud provider's standards
- Establish appropriate incident detection and reporting mechanisms
- Analyze what are the security objectives of organization
- Enquire about who is responsible of data privacy and security issues in cloud

NIST recommendations for cloud security



Organization/Provider Cloud Security Compliance Checklist

Management	Organization	Provider
Is everyone aware of his or her cloud security responsibilities?		
Is there a mechanism for assessing the security of a cloud service?		
Does the business governance mitigate the security risks that can result from cloud-based "shadow IT"?		
Does the organization know within which jurisdictions its data can reside?		
Is there a mechanism for managing cloud-related risks?		
Does the organization understand the data architecture needed to operate with appropriate security at all levels?		
Can the organization be confident of end-to-end service continuity across several cloud service providers?		
Does the provider comply with all relevant industry standards (e.g. the UK% Data Protection Act)?		
Does the compliance function understand the specific regulatory issues pertaining to the organization's adoption of cloud services?		



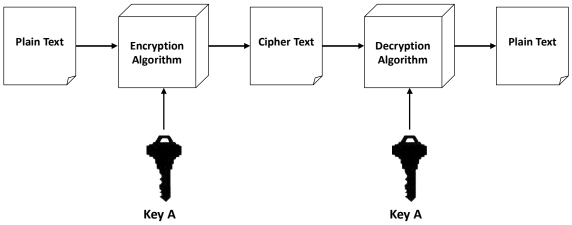
Cryptography in Cloud Computing

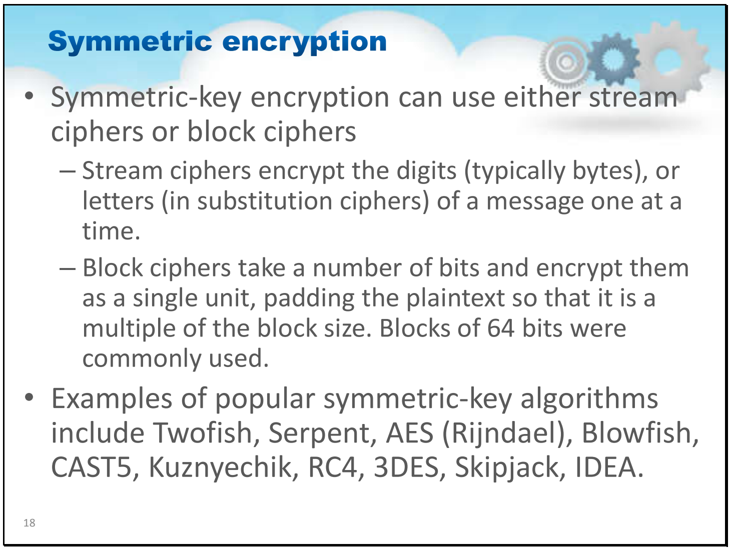
- Cryptography provides techniques that can be used to implement core security services such as confidentiality and data integrity.
- But this cryptographic mechanisms have some limitations with respect to cloud computing environments. So, we should to talk about another cryptographic tools that have the potential to provide the extended security functionality required by some cloud computing applications.

Cryptography basics

Symmetric encryption

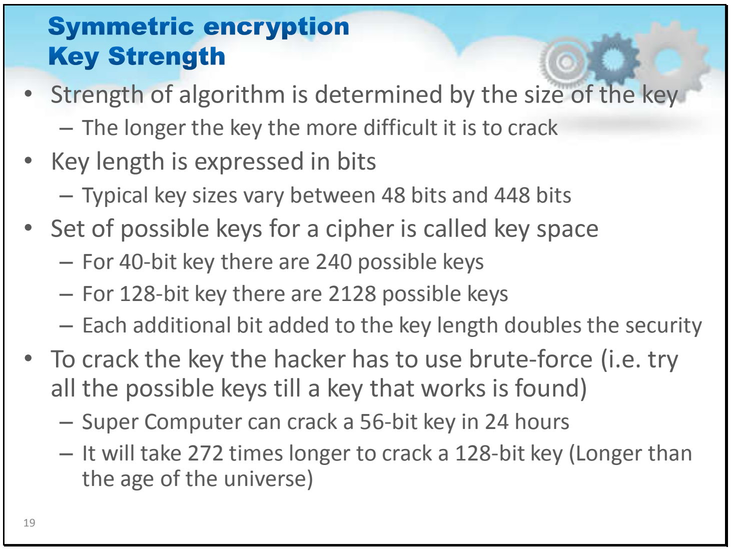
- Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys, in practice, represent a shared secret between two or more parties. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption.





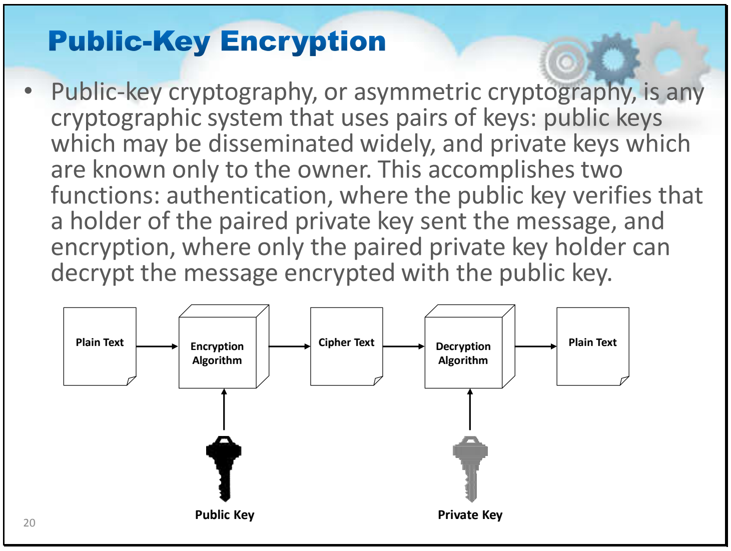
Symmetric encryption

- Symmetric-key encryption can use either stream ciphers or block ciphers
  - Stream ciphers encrypt the digits (typically bytes), or letters (in substitution ciphers) of a message one at a time.
  - Block ciphers take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. Blocks of 64 bits were commonly used.
- Examples of popular symmetric-key algorithms include Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, Kuznyechik, RC4, 3DES, Skipjack, IDEA.



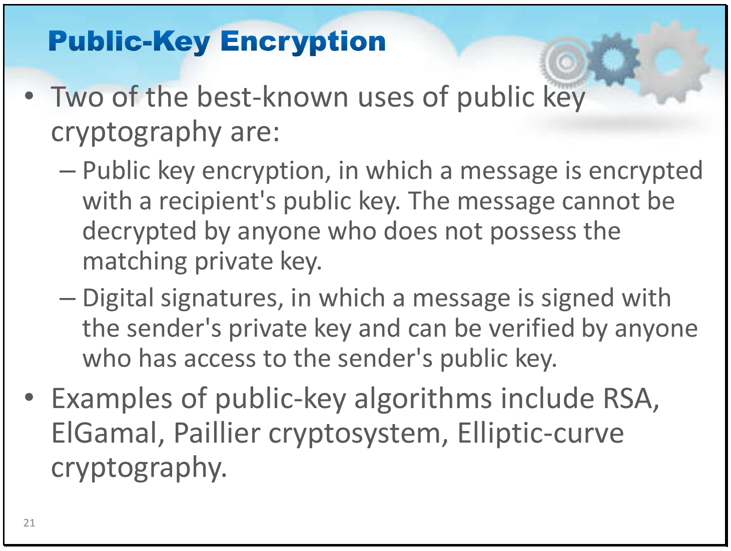
Symmetric encryption
Key Strength

- Strength of algorithm is determined by the size of the key
  - The longer the key the more difficult it is to crack
- Key length is expressed in bits
  - Typical key sizes vary between 48 bits and 448 bits
- Set of possible keys for a cipher is called key space
  - For 40-bit key there are 240 possible keys
  - For 128-bit key there are 2128 possible keys
  - Each additional bit added to the key length doubles the security
- To crack the key the hacker has to use brute-force (i.e. try all the possible keys till a key that works is found)
  - Super Computer can crack a 56-bit key in 24 hours
  - It will take 272 times longer to crack a 128-bit key (Longer than the age of the universe)



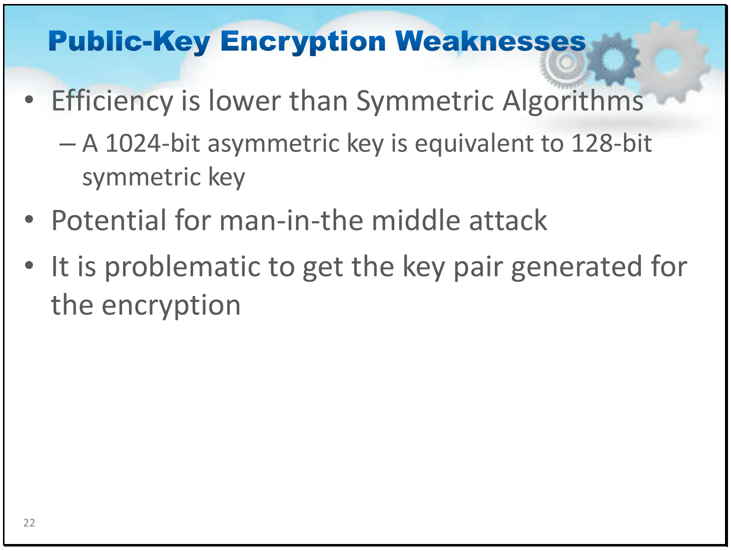
Public-Key Encryption

- Public-key cryptography, or asymmetric cryptography, is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, where the public key verifies that a holder of the paired private key sent the message, and encryption, where only the paired private key holder can decrypt the message encrypted with the public key.



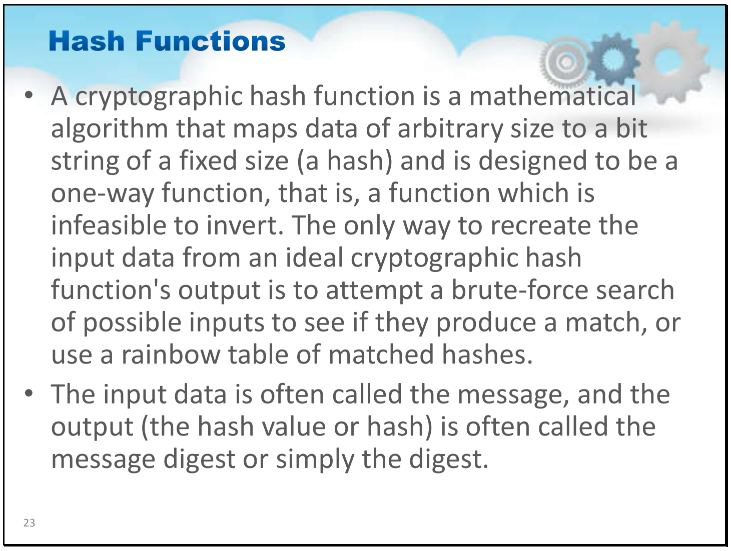
Public-Key Encryption

- Two of the best-known uses of public key cryptography are:
  - Public key encryption, in which a message is encrypted with a recipient's public key. The message cannot be decrypted by anyone who does not possess the matching private key.
  - Digital signatures, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key.
- Examples of public-key algorithms include RSA, ElGamal, Paillier cryptosystem, Elliptic-curve cryptography.



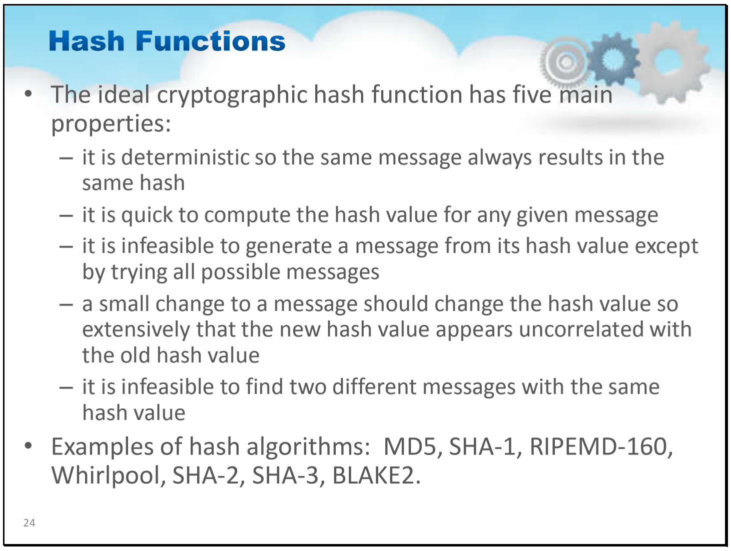
Public-Key Encryption Weaknesses

- Efficiency is lower than Symmetric Algorithms
  - A 1024-bit asymmetric key is equivalent to 128-bit symmetric key
- Potential for man-in-the middle attack
- It is problematic to get the key pair generated for the encryption



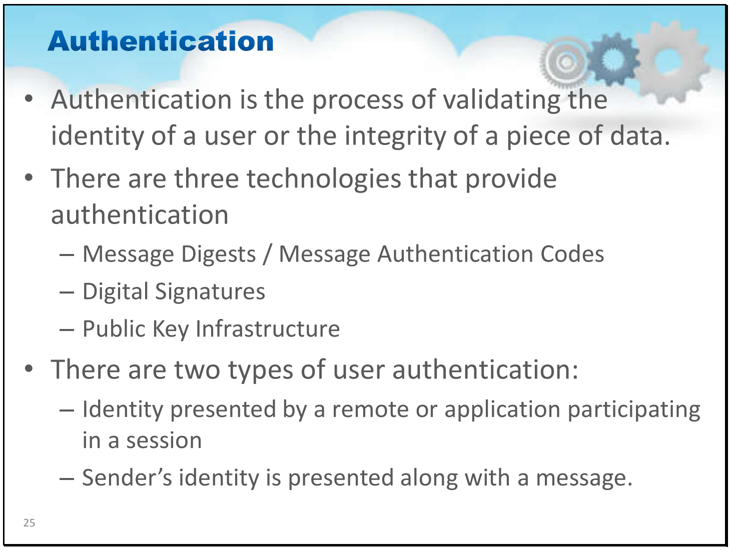
Hash Functions

- A cryptographic hash function is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash) and is designed to be a one-way function, that is, a function which is infeasible to invert. The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes.
- The input data is often called the message, and the output (the hash value or hash) is often called the message digest or simply the digest.



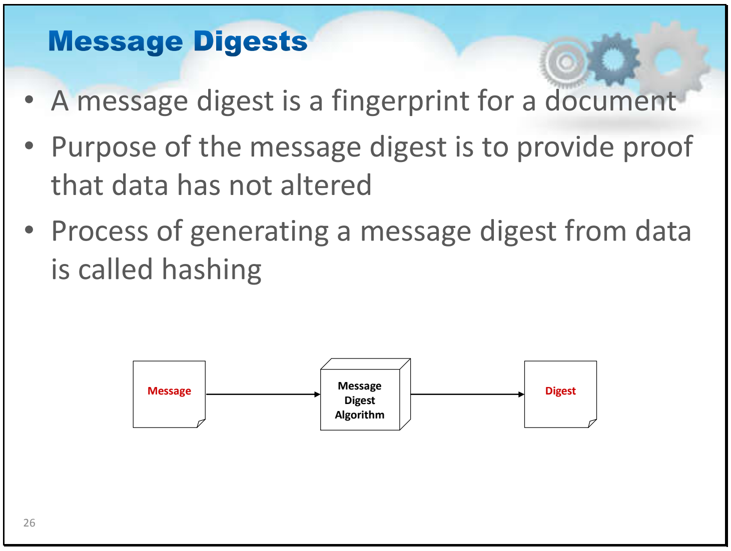
Hash Functions

- The ideal cryptographic hash function has five main properties:
  - it is deterministic so the same message always results in the same hash
  - it is quick to compute the hash value for any given message
  - it is infeasible to generate a message from its hash value except by trying all possible messages
  - a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
  - it is infeasible to find two different messages with the same hash value
- Examples of hash algorithms: MD5, SHA-1, RIPEMD-160, Whirlpool, SHA-2, SHA-3, BLAKE2.



Authentication

- Authentication is the process of validating the identity of a user or the integrity of a piece of data.
- There are three technologies that provide authentication
  - Message Digests / Message Authentication Codes
  - Digital Signatures
  - Public Key Infrastructure
- There are two types of user authentication:
  - Identity presented by a remote or application participating in a session
  - Sender’s identity is presented along with a message.



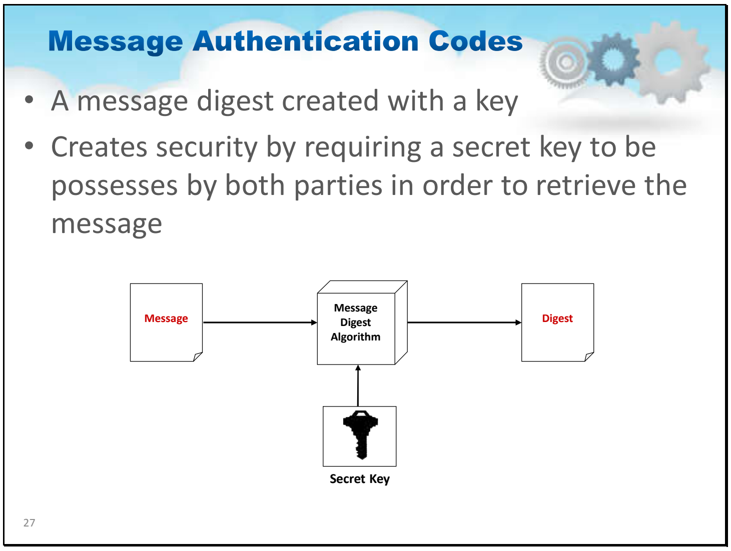
Message Digests

- A message digest is a fingerprint for a document
- Purpose of the message digest is to provide proof that data has not altered
- Process of generating a message digest from data is called hashing

Message

Message Digest Algorithm

Digest



Message Authentication Codes

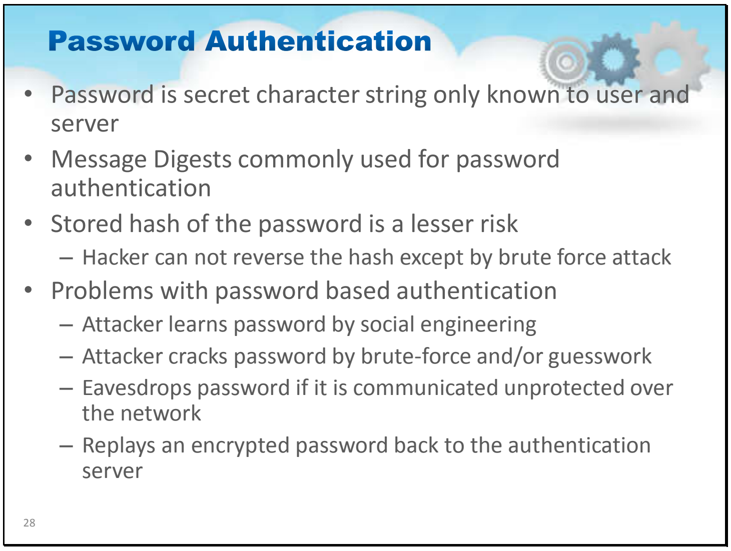
- A message digest created with a key
- Creates security by requiring a secret key to be possesses by both parties in order to retrieve the message

Message

Message Digest Algorithm

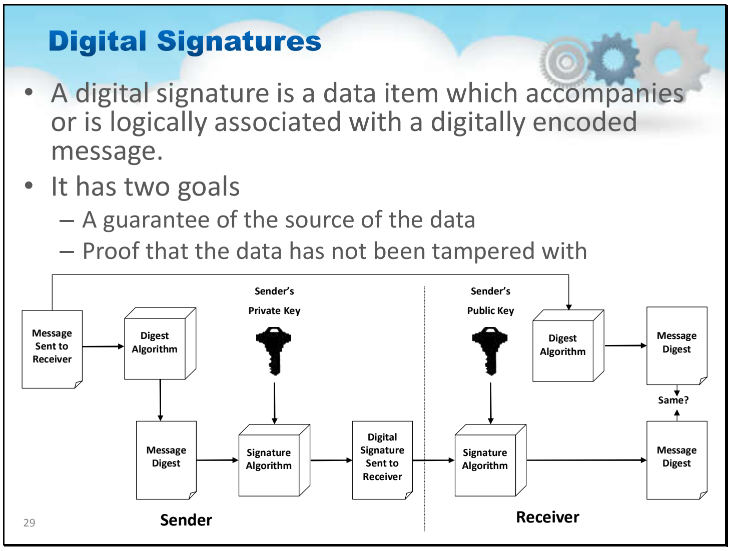
Digest

Secret Key



Password Authentication

- Password is secret character string only known to user and server
- Message Digests commonly used for password authentication
- Stored hash of the password is a lesser risk
  - Hacker can not reverse the hash except by brute force attack
- Problems with password based authentication
  - Attacker learns password by social engineering
  - Attacker cracks password by brute-force and/or guesswork
  - Eavesdrops password if it is communicated unprotected over the network
  - Replays an encrypted password back to the authentication server



Digital Signatures

- A digital signature is a data item which accompanies or is logically associated with a digitally encoded message.
- It has two goals
  - A guarantee of the source of the data
  - Proof that the data has not been tampered with

Message Sent to Receiver

Digest Algorithm

Message Digest

Sender's Private Key

Signature Algorithm

Digital Signature Sent to Receiver

Sender's Public Key

Signature Algorithm

Digest Algorithm

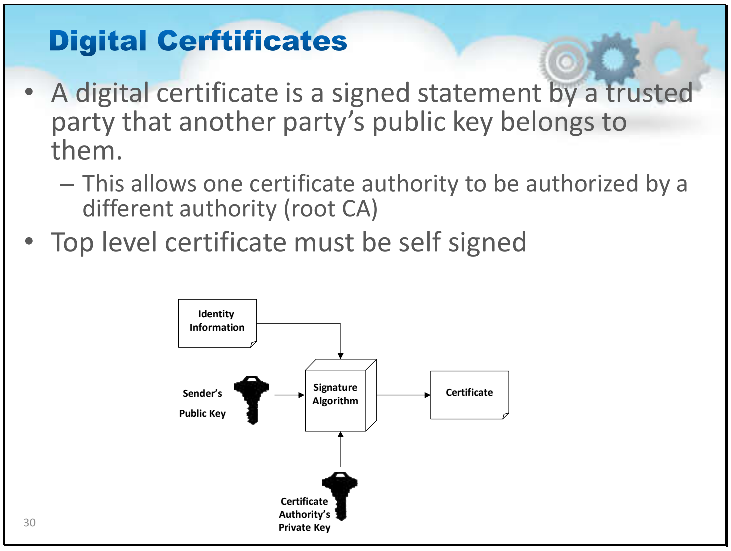
Message Digest

Same?

Message Digest

Sender

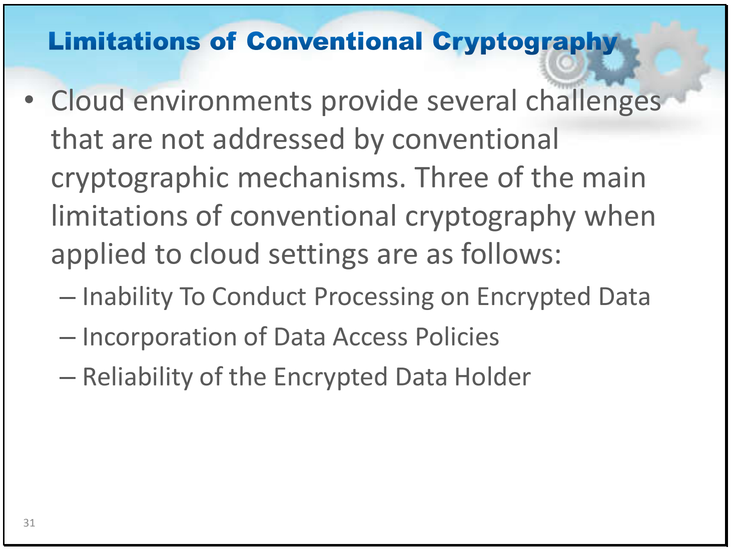
Receiver



Digital Cerftificates

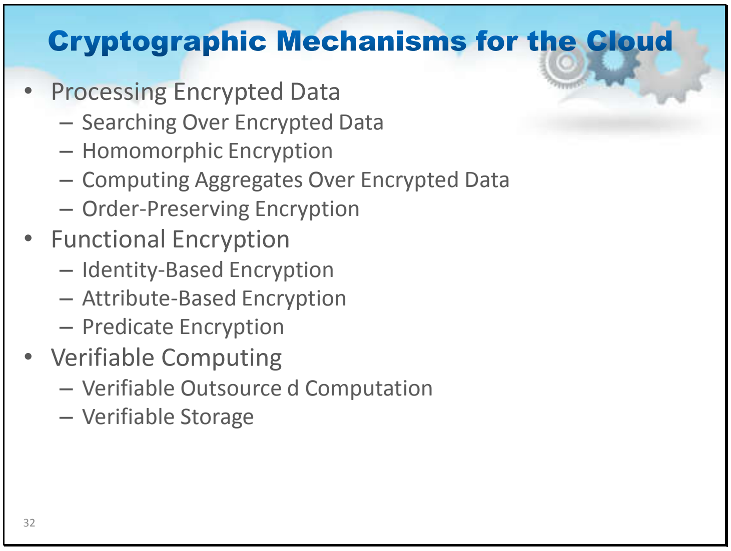
- A digital certificate is a signed statement by a trusted party that another party’s public key belongs to them.
  - – This allows one certificate authority to be authorized by a different authority (root CA)
- Top level certificate must be self signed

Identity Information
Sender’s Public Key
Signature Algorithm
Certificate
Certificate Authority’s Private Key



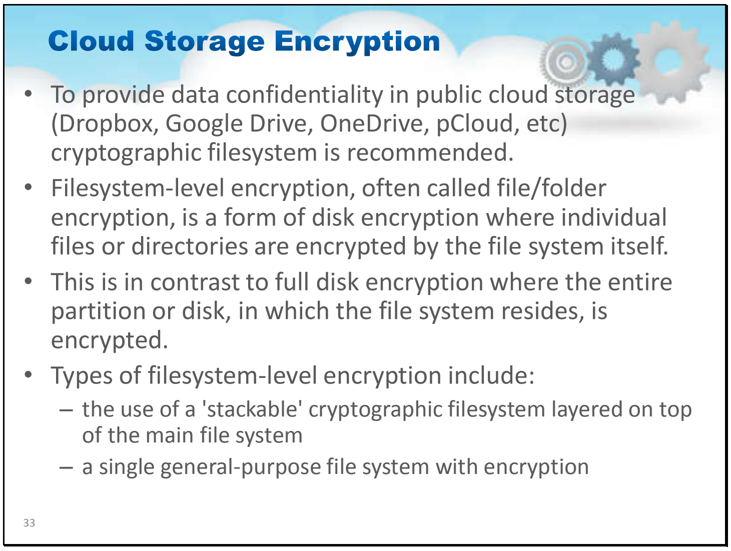
Limitations of Conventional Cryptography

- Cloud environments provide several challenges that are not addressed by conventional cryptographic mechanisms. Three of the main limitations of conventional cryptography when applied to cloud settings are as follows:
  - Inability To Conduct Processing on Encrypted Data
  - Incorporation of Data Access Policies
  - Reliability of the Encrypted Data Holder



Cryptographic Mechanisms for the Cloud

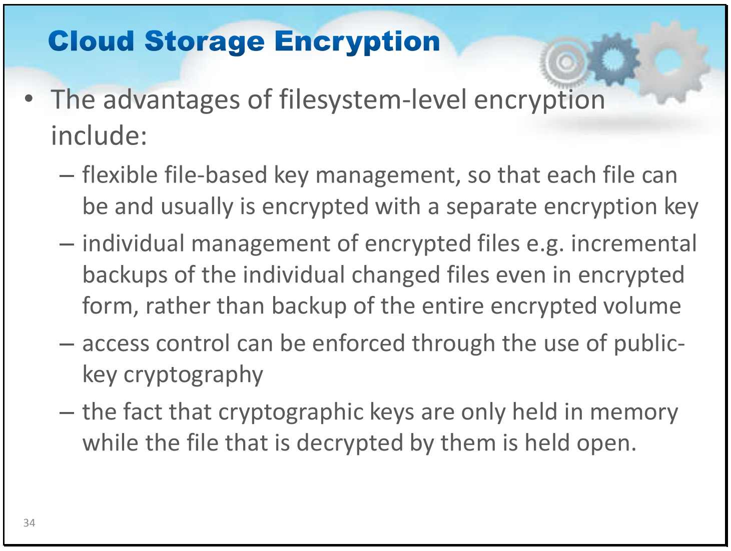
- Processing Encrypted Data
  - Searching Over Encrypted Data
  - Homomorphic Encryption
  - Computing Aggregates Over Encrypted Data
  - Order-Preserving Encryption
- Functional Encryption
  - Identity-Based Encryption
  - Attribute-Based Encryption
  - Predicate Encryption
- Verifiable Computing
  - Verifiable Outsource d Computation
  - Verifiable Storage



Cloud Storage Encryption

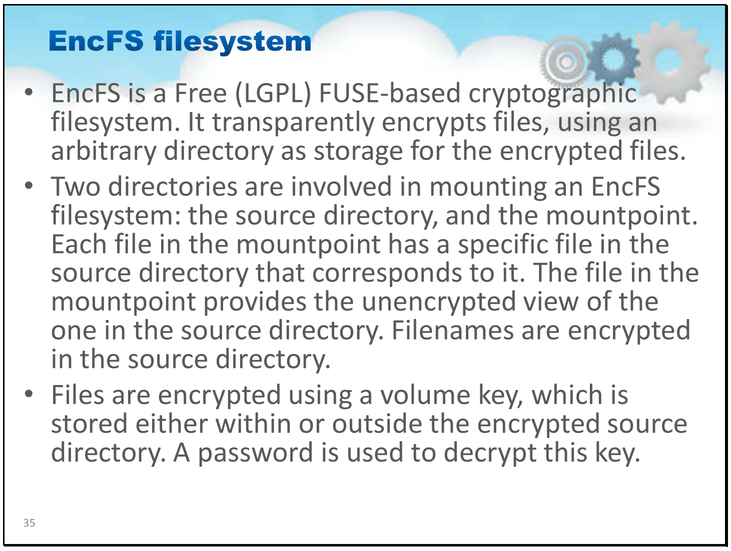
- To provide data confidentiality in public cloud storage (Dropbox, Google Drive, OneDrive, pCloud, etc) cryptographic filesystem is recommended.
- Filesystem-level encryption, often called file/folder encryption, is a form of disk encryption where individual files or directories are encrypted by the file system itself.
- This is in contrast to full disk encryption where the entire partition or disk, in which the file system resides, is encrypted.
- Types of filesystem-level encryption include:
  - the use of a 'stackable' cryptographic filesystem layered on top of the main file system
  - a single general-purpose file system with encryption

Cloud storage encryption



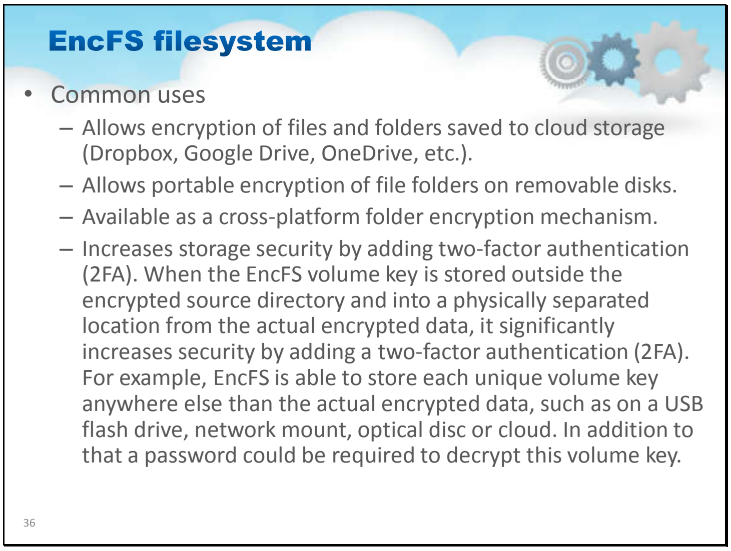
Cloud Storage Encryption

- The advantages of filesystem-level encryption include:
  - flexible file-based key management, so that each file can be and usually is encrypted with a separate encryption key
  - individual management of encrypted files e.g. incremental backups of the individual changed files even in encrypted form, rather than backup of the entire encrypted volume
  - access control can be enforced through the use of public-key cryptography
  - the fact that cryptographic keys are only held in memory while the file that is decrypted by them is held open.



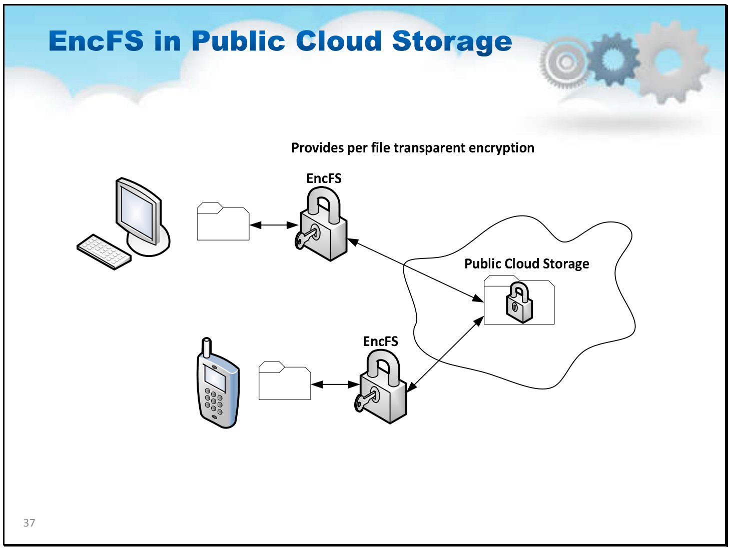
EncFS filesystem

- EncFS is a Free (LGPL) FUSE-based cryptographic filesystem. It transparently encrypts files, using an arbitrary directory as storage for the encrypted files.
- Two directories are involved in mounting an EncFS filesystem: the source directory, and the mountpoint. Each file in the mountpoint has a specific file in the source directory that corresponds to it. The file in the mountpoint provides the unencrypted view of the one in the source directory. Filenames are encrypted in the source directory.
- Files are encrypted using a volume key, which is stored either within or outside the encrypted source directory. A password is used to decrypt this key.



EncFS filesystem

- Common uses
  - Allows encryption of files and folders saved to cloud storage (Dropbox, Google Drive, OneDrive, etc.).
  - Allows portable encryption of file folders on removable disks.
  - Available as a cross-platform folder encryption mechanism.
  - Increases storage security by adding two-factor authentication (2FA). When the EncFS volume key is stored outside the encrypted source directory and into a physically separated location from the actual encrypted data, it significantly increases security by adding a two-factor authentication (2FA). For example, EncFS is able to store each unique volume key anywhere else than the actual encrypted data, such as on a USB flash drive, network mount, optical disc or cloud. In addition to that a password could be required to decrypt this volume key.



EncFS in Public Cloud Storage

Provides per file transparent encryption

EncFS

Public Cloud Storage

EncFS

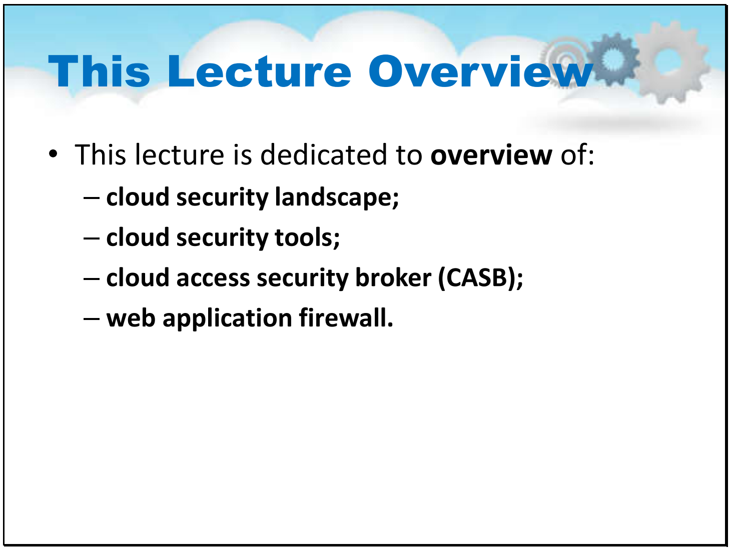
EncFS provides transparent encryption of each file when data is transferred to the cloud and decryption when data is returned. This function can be used to ensure confidentiality in the public cloud storage.

Cloud Computing

Module – Cloud Computing security

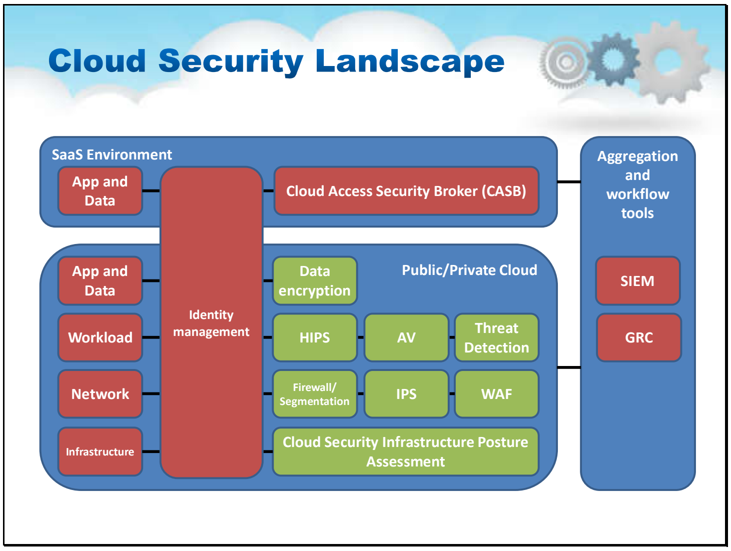
Lecture 3. Security Audit in Cloud Computing

Lecture 3.Security Audit in Cloud Computing



This Lecture Overview

- This lecture is dedicated to **overview** of:
  - **cloud security landscape;**
  - **cloud security tools;**
  - **cloud access security broker (CASB);**
  - **web application firewall.**

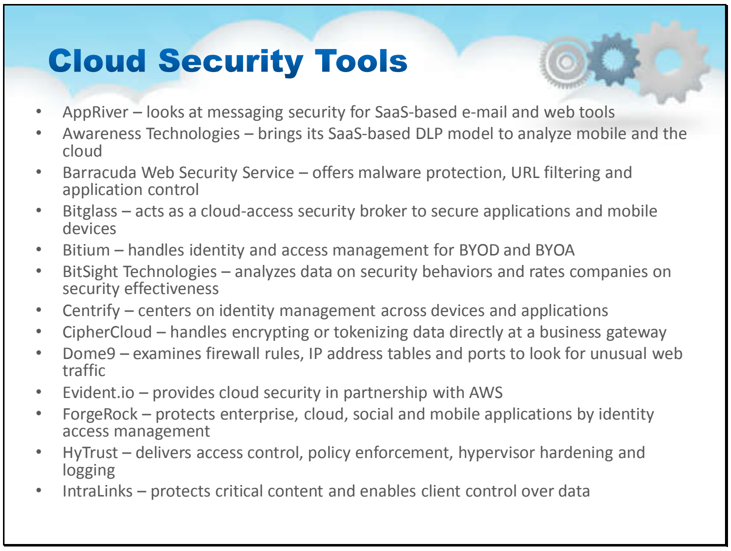


Cloud Security Landscape

Cloud security landscape

Effective cloud security has enough layers that even experienced security professionals at times have trouble keeping up with the multiple components of a robust cloud security strategy – and it does require a strategy.

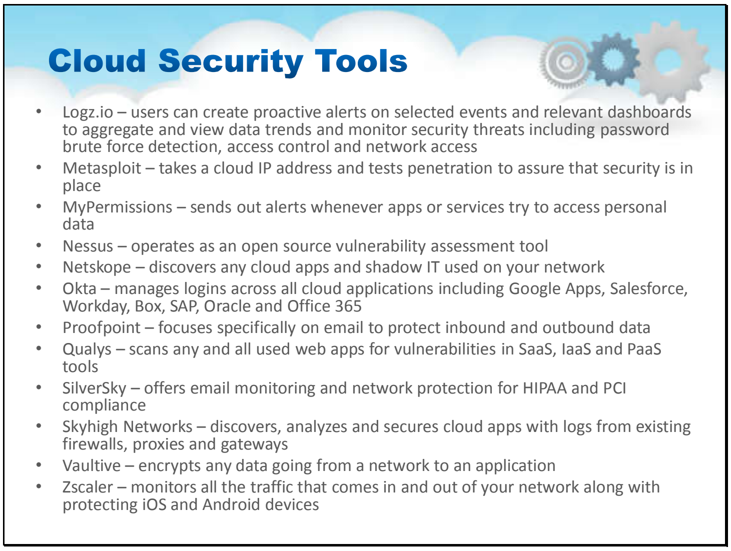
Unfortunately, we repeatedly see organizations with approaches that are strong in the areas they cover themselves, but miss one (or multiple) layers that expose organizations to serious threats. Owing to this, we've laid out a diagram that highlights the primary layers and vendors in the cloud security landscape. These range from well-known components like antivirus and firewalls – to critical new solutions such as cloud infrastructure security posture assessment (CISPA) tools – effective cloud security strategies:
- Adopt the shared security model detailed by the major IaaS providers
- Ensure every layer is accounted for



Cloud Security Tools

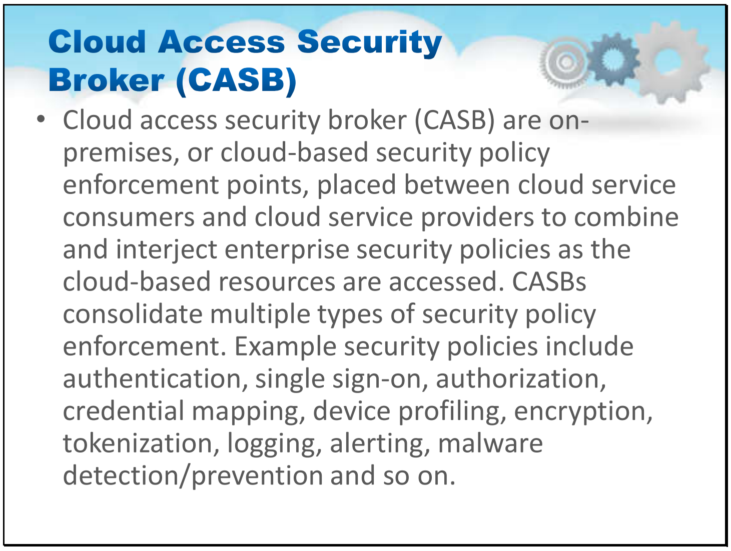
- AppRiver – looks at messaging security for SaaS-based e-mail and web tools
- Awareness Technologies – brings its SaaS-based DLP model to analyze mobile and the cloud
- Barracuda Web Security Service – offers malware protection, URL filtering and application control
- Bitglass – acts as a cloud-access security broker to secure applications and mobile devices
- Bitium – handles identity and access management for BYOD and BYOA
- BitSight Technologies – analyzes data on security behaviors and rates companies on security effectiveness
- Centrify – centers on identity management across devices and applications
- CipherCloud – handles encrypting or tokenizing data directly at a business gateway
- Dome9 – examines firewall rules, IP address tables and ports to look for unusual web traffic
- Evident.io – provides cloud security in partnership with AWS
- ForgeRock – protects enterprise, cloud, social and mobile applications by identity access management
- HyTrust – delivers access control, policy enforcement, hypervisor hardening and logging
- IntraLinks – protects critical content and enables client control over data

Cloud security tools



Cloud Security Tools

- Logz.io – users can create proactive alerts on selected events and relevant dashboards to aggregate and view data trends and monitor security threats including password brute force detection, access control and network access
- Metasploit – takes a cloud IP address and tests penetration to assure that security is in place
- MyPermissions – sends out alerts whenever apps or services try to access personal data
- Nessus – operates as an open source vulnerability assessment tool
- Netskope – discovers any cloud apps and shadow IT used on your network
- Okta – manages logins across all cloud applications including Google Apps, Salesforce, Workday, Box, SAP, Oracle and Office 365
- Proofpoint – focuses specifically on email to protect inbound and outbound data
- Qualys – scans any and all used web apps for vulnerabilities in SaaS, IaaS and PaaS tools
- SilverSky – offers email monitoring and network protection for HIPAA and PCI compliance
- Skyhigh Networks – discovers, analyzes and secures cloud apps with logs from existing firewalls, proxies and gateways
- Vaultive – encrypts any data going from a network to an application
- Zscaler – monitors all the traffic that comes in and out of your network along with protecting iOS and Android devices

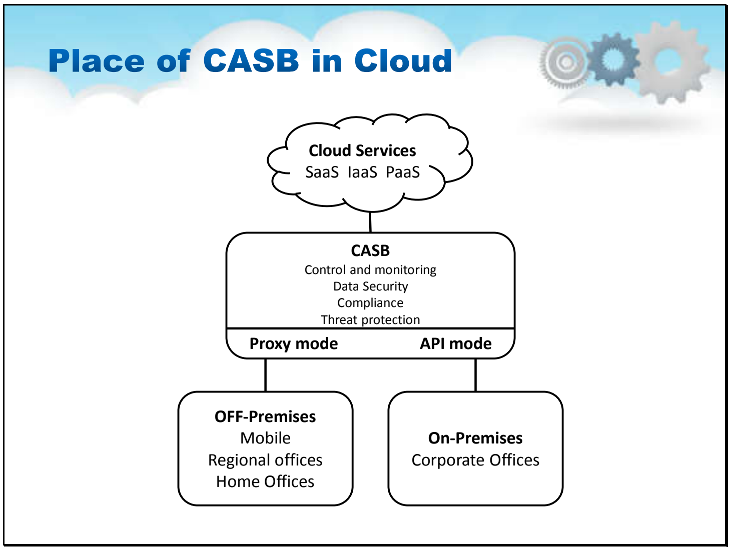


Cloud Access Security Broker (CASB)

- Cloud access security broker (CASB) are on-premises, or cloud-based security policy enforcement points, placed between cloud service consumers and cloud service providers to combine and interject enterprise security policies as the cloud-based resources are accessed. CASBs consolidate multiple types of security policy enforcement. Example security policies include authentication, single sign-on, authorization, credential mapping, device profiling, encryption, tokenization, logging, alerting, malware detection/prevention and so on.

Cloud access security broker (CASB)

Cloud Access Security Broker (CASB) software has emerged to help IT get its arms around the full cloud security situation. CASBs are security policy enforcement points between cloud service users and one or more cloud service providers. They can reside on the enterprise's premises, or a cloud provider can host them. Either way, CASBs provide information security professionals with a critical control point for the secure and compliant use of cloud services across multiple cloud providers. They enforce the many layers of an enterprise's security policies as users, devices, and other cloud entities attempt to access cloud resources.



Place of CASB in Cloud

Cloud Services
SaaS IaaS PaaS

CASB
Control and monitoring
Data Security
Compliance
Threat protection

Proxy mode
API mode

OFF-Premises
Mobile
Regional offices
Home Offices

On-Premises
Corporate Offices

A CASB provides enterprises with a critical control point for the secure use of cloud services across multiple cloud providers. Software as a service (SaaS) apps are becoming pervasive in enterprises, which exacerbates the frustration of security teams looking for visibility and control of those apps.

CASB sales have soared as cloud security concerns have grown, especially the use of "Shadow IT" cloud services that IT security teams don't know about.

CASB solutions fill many of the security gaps in individual cloud services and allow information security professionals to do it across cloud services, including infrastructure as a service (IaaS) and platform as a service (PaaS) providers. As such, CASBs address a critical enterprise requirement to set policy, monitor behavior, and manage risk across the entire set of enterprise cloud services being consumed.

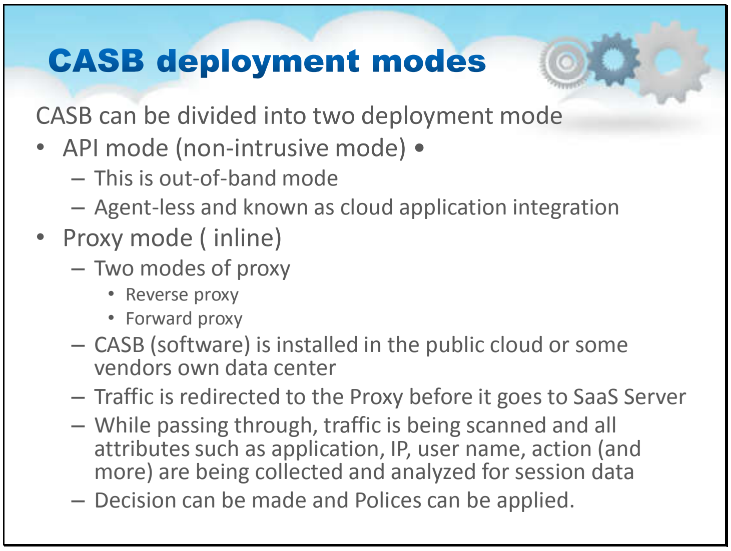
What is CASB

A CASB can consolidate multiple types of security policy enforcement. Examples of security policies enforced by a CASB include authentication, single sign on, authorization, credential mapping, device profiling, encryption, tokenization, logging, alerting, and malware detection and prevention.

A CASB vendor also gives enterprises visibility into authorized and non-authorized cloud usage. It can intercept and monitor data traffic between the corporate network and cloud platform, assist with compliance issues, offer data security policy enforcement, and prevent unauthorized devices, users, and apps from accessing cloud services.

In the all-important area of data security, a CASB provider enforces corporate data security policies to prevent unwanted activity based on data classification, data discovery, and user activity monitoring. Policies are applied through controls, such as audits, alerts, blocking, quarantine, deletion, and encryption, at the field and file level in cloud hosting services.

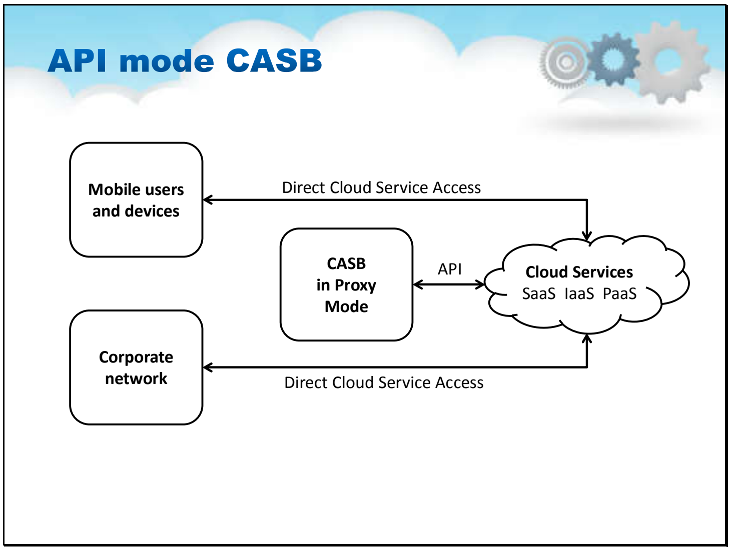
CASB solutions include control and monitoring, risk and compliance management, threat protection, and cloud data security.



CASB deployment modes

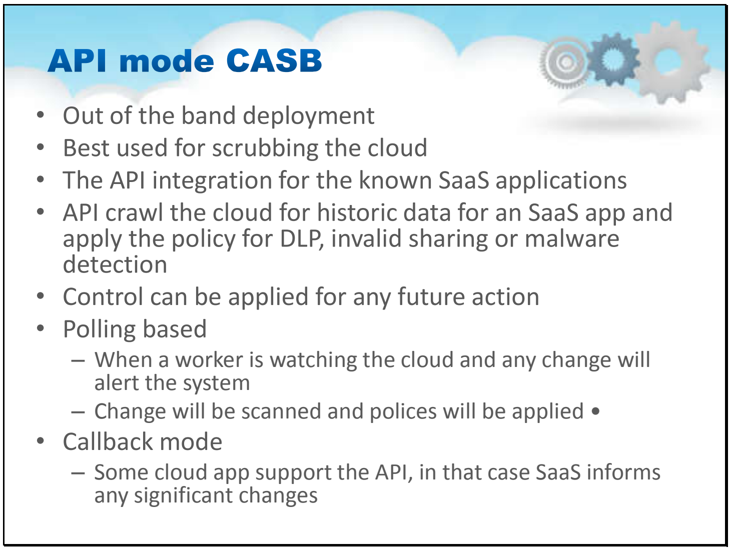
CASB can be divided into two deployment mode

- API mode (non-intrusive mode) •
  - This is out-of-band mode
  - Agent-less and known as cloud application integration
- Proxy mode (inline)
  - Two modes of proxy
    - Reverse proxy
    - Forward proxy
  - CASB (software) is installed in the public cloud or some vendors own data center
  - Traffic is redirected to the Proxy before it goes to SaaS Server
  - While passing through, traffic is being scanned and all attributes such as application, IP, user name, action (and more) are being collected and analyzed for session data
  - Decision can be made and Polices can be applied.



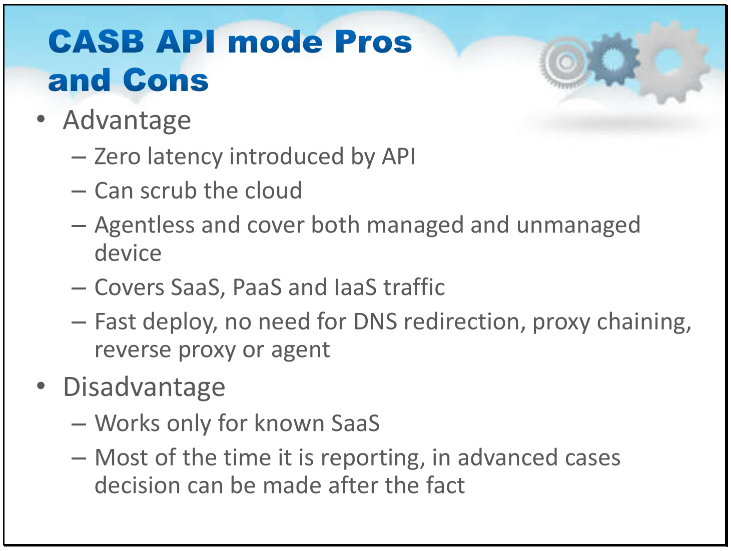
API mode CASB

An API-based CASB is an Out-of-Band solution that does not follow the same network path as data. Since the solution integrates directly with cloud services, API-based solutions have no performance degradation, and they secure both managed and unmanaged traffic across Saas, IaaS, and PaaS cloud services.



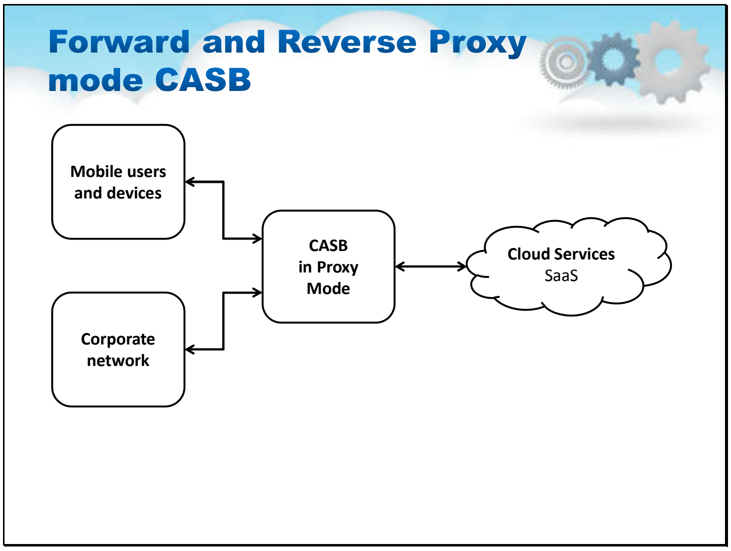
API mode CASB

- Out of the band deployment
- Best used for scrubbing the cloud
- The API integration for the known SaaS applications
- API crawl the cloud for historic data for an SaaS app and apply the policy for DLP, invalid sharing or malware detection
- Control can be applied for any future action
- Polling based
  - When a worker is watching the cloud and any change will alert the system
  - Change will be scanned and polices will be applied •
- Callback mode
  - Some cloud app support the API, in that case SaaS informs any significant changes



CASB API mode Pros and Cons

- Advantage
  - Zero latency introduced by API
  - Can scrub the cloud
  - Agentless and cover both managed and unmanaged device
  - Covers SaaS, PaaS and IaaS traffic
  - Fast deploy, no need for DNS redirection, proxy chaining, reverse proxy or agent
- Disadvantage
  - Works only for known SaaS
  - Most of the time it is reporting, in advanced cases decision can be made after the fact



Forward and Reverse Proxy mode CASB

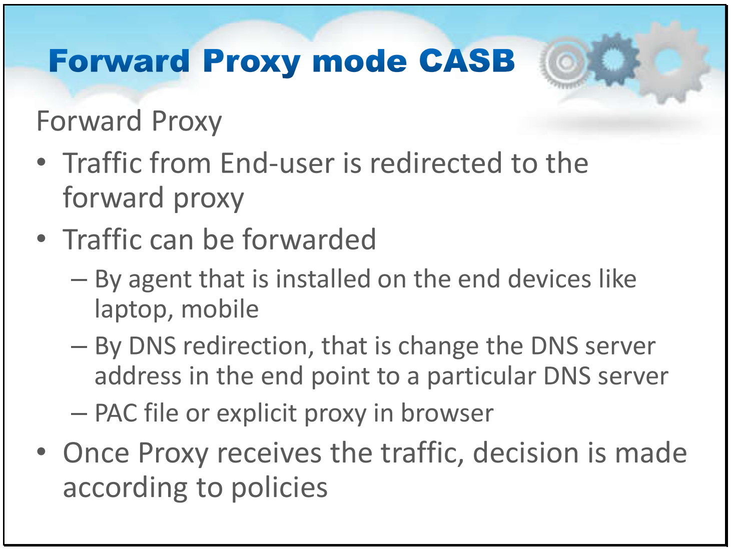
Mobile users and devices

Corporate network

CASB in Proxy Mode

Cloud Services
SaaS

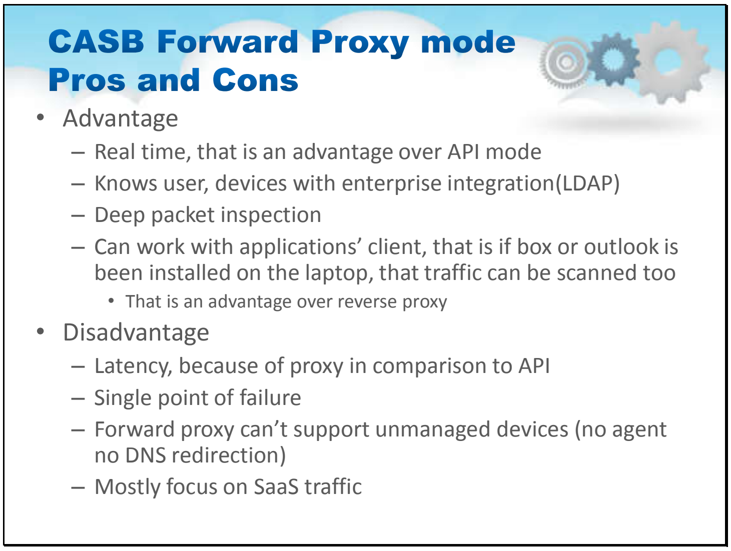
An in-line proxy solution checks and filters known users and devices through a single gateway. Because all traffic flows through a single checkpoint, the proxy can take security action in real-time. Unfortunately, the single checkpoint also means that it slows network performance, and only secures known users. Further, proxy-based solutions only secure SaaS cloud services, leaving IaaS and PaaS clouds vulnerable.



Forward Proxy mode CASB

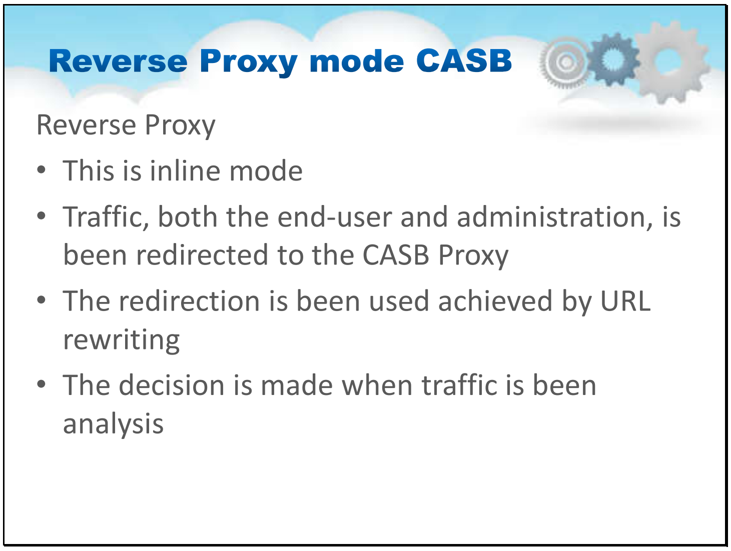
Forward Proxy

- Traffic from End-user is redirected to the forward proxy
- Traffic can be forwarded
  - By agent that is installed on the end devices like laptop, mobile
  - By DNS redirection, that is change the DNS server address in the end point to a particular DNS server
  - PAC file or explicit proxy in browser
- Once Proxy receives the traffic, decision is made according to policies



CASB Forward Proxy mode Pros and Cons

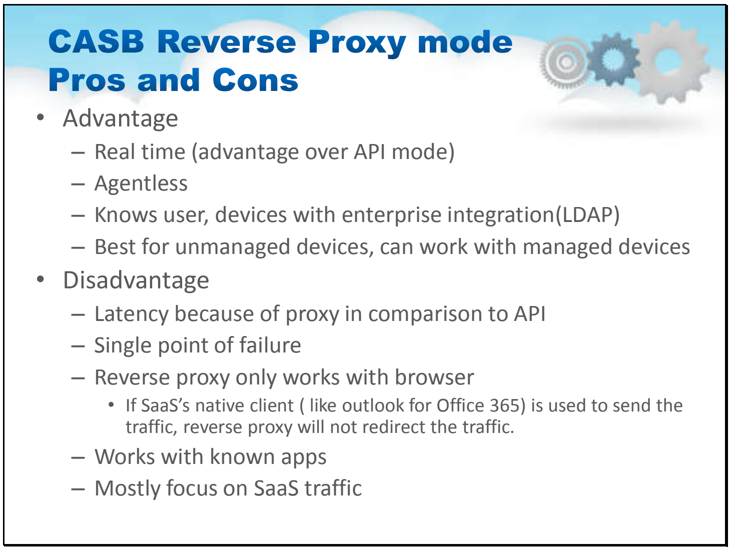
- Advantage
  - Real time, that is an advantage over API mode
  - Knows user, devices with enterprise integration(LDAP)
  - Deep packet inspection
  - Can work with applications’ client, that is if box or outlook is been installed on the laptop, that traffic can be scanned too
    - That is an advantage over reverse proxy
- Disadvantage
  - Latency, because of proxy in comparison to API
  - Single point of failure
  - Forward proxy can’t support unmanaged devices (no agent no DNS redirection)
  - Mostly focus on SaaS traffic



Reverse Proxy mode CASB

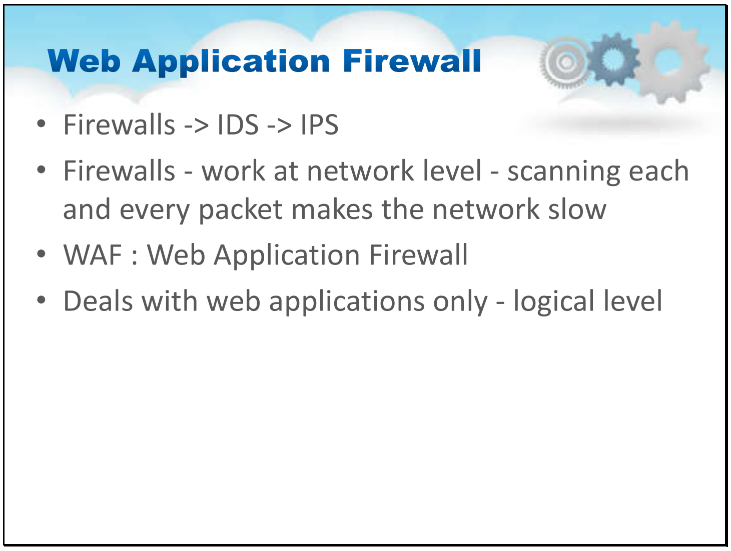
Reverse Proxy

- This is inline mode
- Traffic, both the end-user and administration, is been redirected to the CASB Proxy
- The redirection is been used achieved by URL rewriting
- The decision is made when traffic is been analysis



CASB Reverse Proxy mode Pros and Cons

- Advantage
  - Real time (advantage over API mode)
  - Agentless
  - Knows user, devices with enterprise integration(LDAP)
  - Best for unmanaged devices, can work with managed devices
- Disadvantage
  - Latency because of proxy in comparison to API
  - Single point of failure
  - Reverse proxy only works with browser
    - If SaaS's native client (like outlook for Office 365) is used to send the traffic, reverse proxy will not redirect the traffic.
  - Works with known apps
  - Mostly focus on SaaS traffic

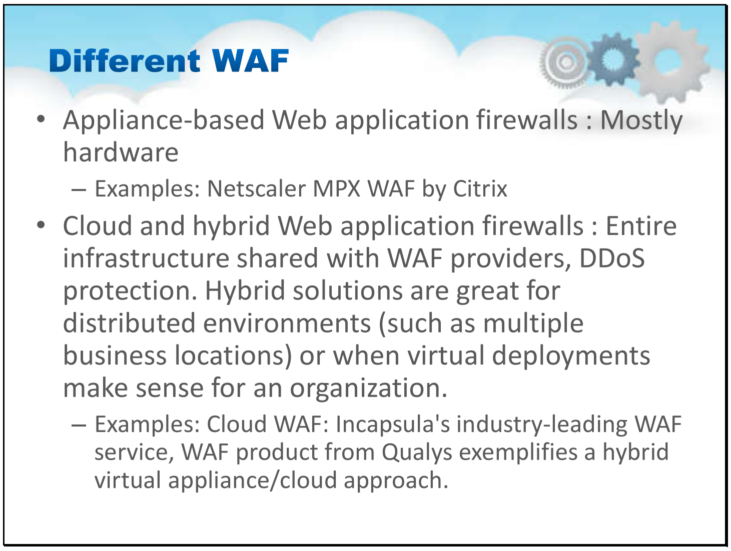


Web Application Firewall

- Firewalls -> IDS -> IPS
- Firewalls - work at network level - scanning each and every packet makes the network slow
- WAF : Web Application Firewall
- Deals with web applications only - logical level

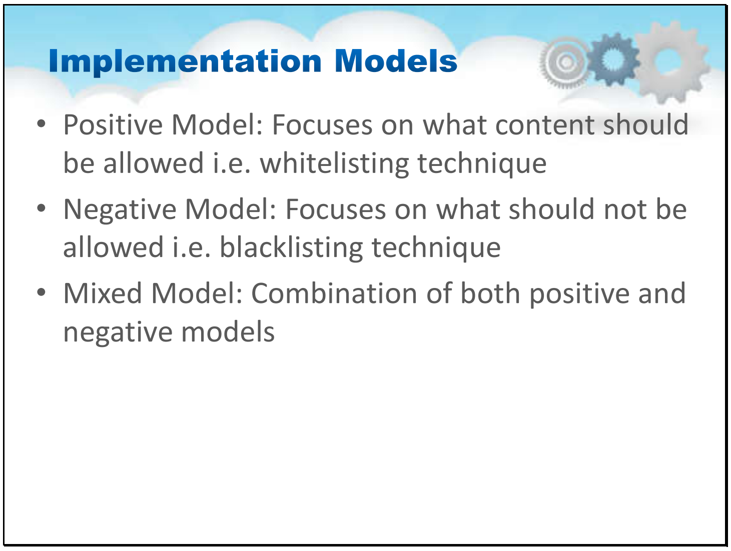
Web application firewall

A web application firewall (or WAF) filters, monitors, and blocks HTTP traffic to and from a web application. A WAF is differentiated from a regular firewall in that a WAF is able to filter the content of specific web applications while regular firewalls serve as a safety gate between servers. By inspecting HTTP traffic, it can prevent attacks stemming from web application security flaws, such as SQL injection, cross-site scripting (XSS), file inclusion, and security misconfigurations.



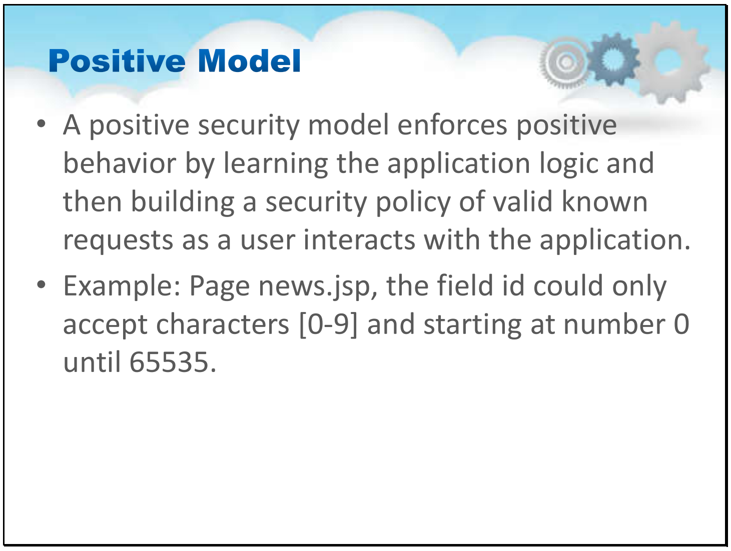
Different WAF

- Appliance-based Web application firewalls : Mostly hardware
  - Examples: Netscaler MPX WAF by Citrix
- Cloud and hybrid Web application firewalls : Entire infrastructure shared with WAF providers, DDoS protection. Hybrid solutions are great for distributed environments (such as multiple business locations) or when virtual deployments make sense for an organization.
  - Examples: Cloud WAF: Incapsula's industry-leading WAF service, WAF product from Qualys exemplifies a hybrid virtual appliance/cloud approach.



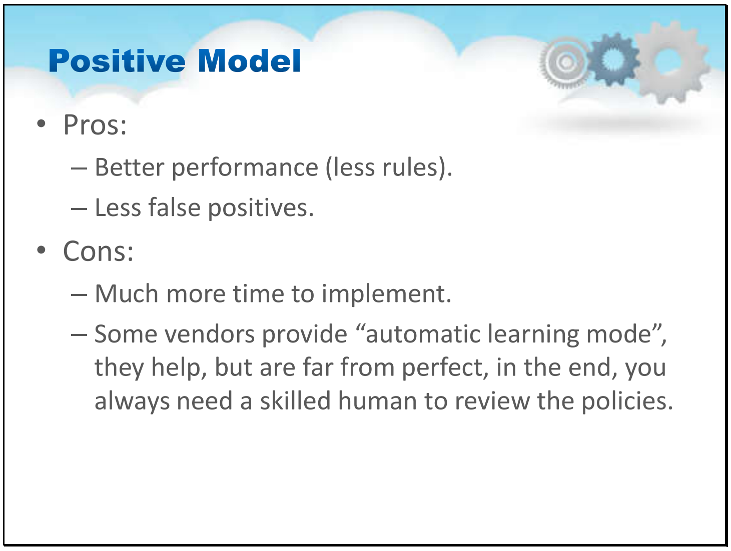
Implementation Models

- Positive Model: Focuses on what content should be allowed i.e. whitelisting technique
- Negative Model: Focuses on what should not be allowed i.e. blacklisting technique
- Mixed Model: Combination of both positive and negative models



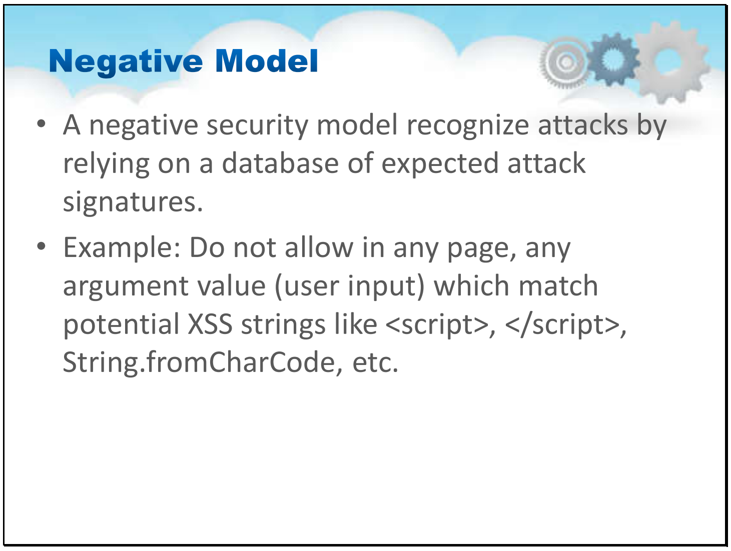
Positive Model

- A positive security model enforces positive behavior by learning the application logic and then building a security policy of valid known requests as a user interacts with the application.
- Example: Page news.jsp, the field id could only accept characters [0-9] and starting at number 0 until 65535.



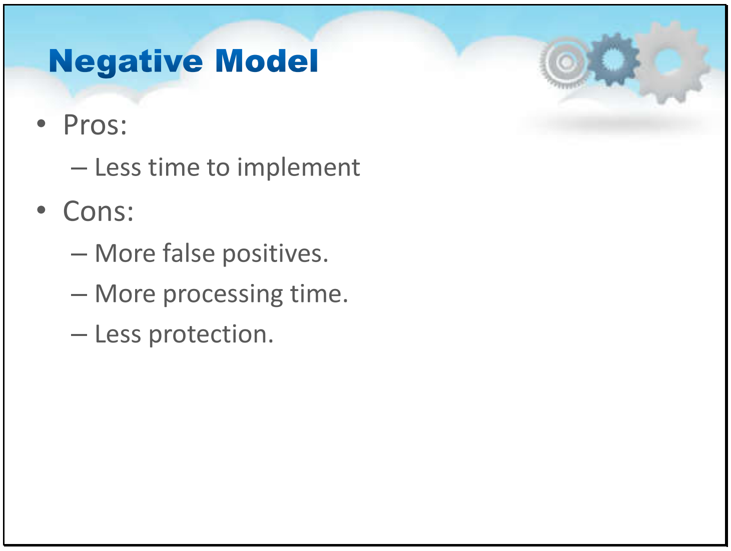
Positive Model

- Pros:
  - – Better performance (less rules).
  - – Less false positives.
- Cons:
  - – Much more time to implement.
  - – Some vendors provide “automatic learning mode”, they help, but are far from perfect, in the end, you always need a skilled human to review the policies.



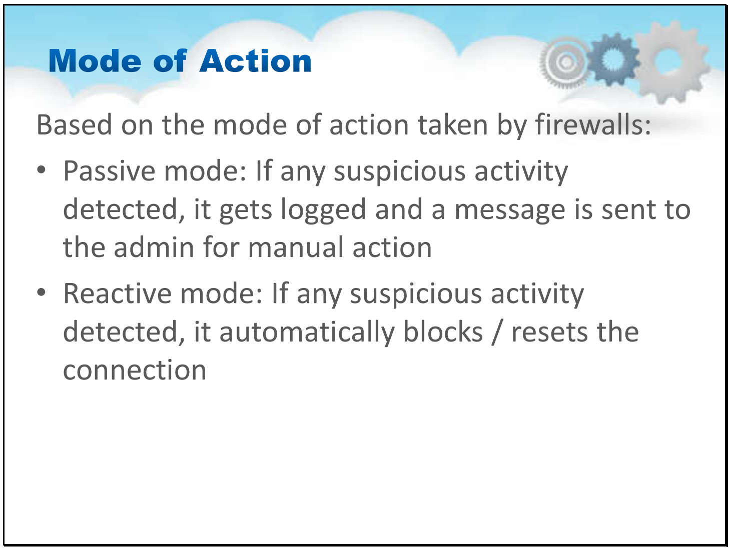
Negative Model

- A negative security model recognize attacks by relying on a database of expected attack signatures.
- Example: Do not allow in any page, any argument value (user input) which match potential XSS strings like <script>, </script>, String.fromCharCode, etc.



Negative Model

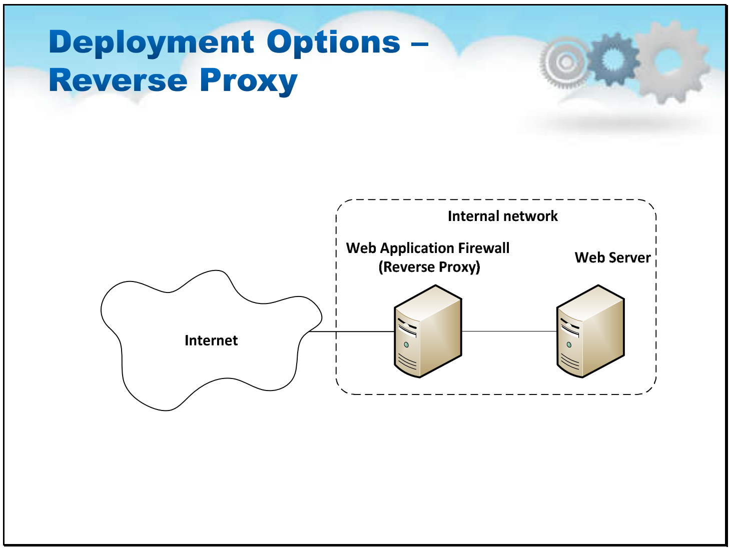
- Pros:
  - Less time to implement
- Cons:
  - More false positives.
  - More processing time.
  - Less protection.



Mode of Action

Based on the mode of action taken by firewalls:

- Passive mode: If any suspicious activity detected, it gets logged and a message is sent to the admin for manual action
- Reactive mode: If any suspicious activity detected, it automatically blocks / resets the connection



Deployment Options – Reverse Proxy

Internet

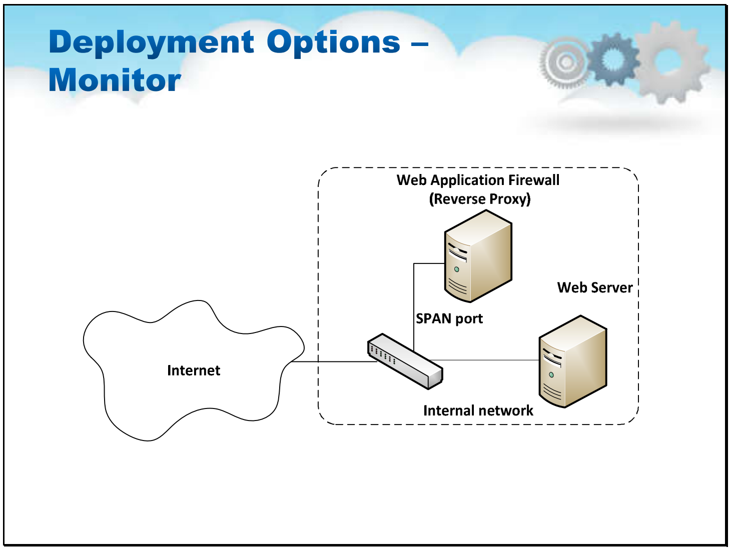
Internal network

Web Application Firewall (Reverse Proxy)

Web Server

In computer networks, a reverse proxy is a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client, appearing as if they originated from the proxy server itself. Unlike a forward proxy, which is an intermediary for its associated clients to contact any server, a reverse proxy is an intermediary for its associated servers to be contacted by any client.

Quite often, popular web servers use reverse-proxying functionality, shielding application frameworks of weaker HTTP capabilities.

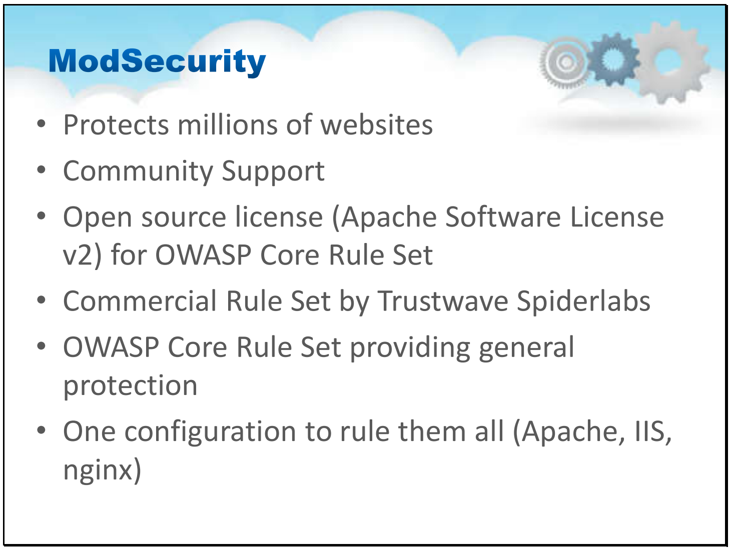


Deployment Options – Monitor

WAF in this deployment mode provide only monitor functions, it uses SPAN port on network switch for get all traffic.

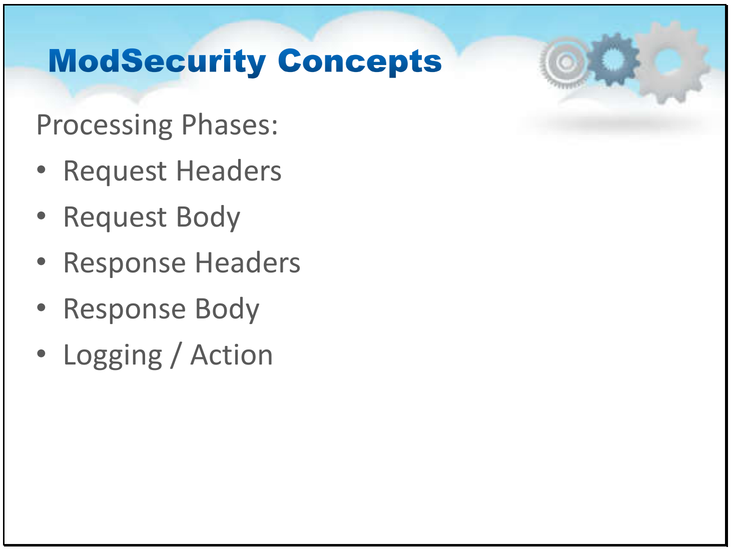
Port Mirroring, also known as SPAN (Switched Port Analyzer), is a method of monitoring network traffic. With port mirroring enabled, the switch sends a copy of all network packets seen on one port (or an entire VLAN) to another port, where the packet can be analyzed.

Port Mirroring function is supported by almost all enterprise-class switches (managed switches).



ModSecurity

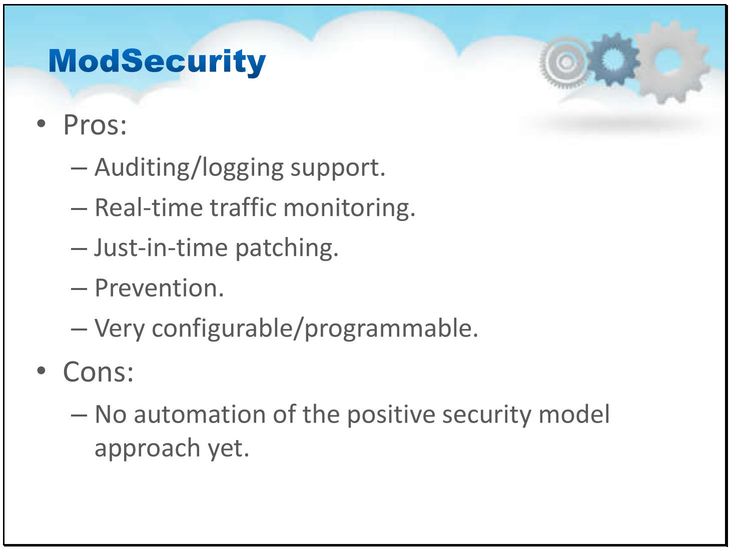
- Protects millions of websites
- Community Support
- Open source license (Apache Software License v2) for OWASP Core Rule Set
- Commercial Rule Set by Trustwave Spiderlabs
- OWASP Core Rule Set providing general protection
- One configuration to rule them all (Apache, IIS, nginx)



ModSecurity Concepts

Processing Phases:

- Request Headers
- Request Body
- Response Headers
- Response Body
- Logging / Action



ModSecurity

- Pros:
  - Auditing/logging support.
  - Real-time traffic monitoring.
  - Just-in-time patching.
  - Prevention.
  - Very configurable/programmable.
- Cons:
  - No automation of the positive security model approach yet.