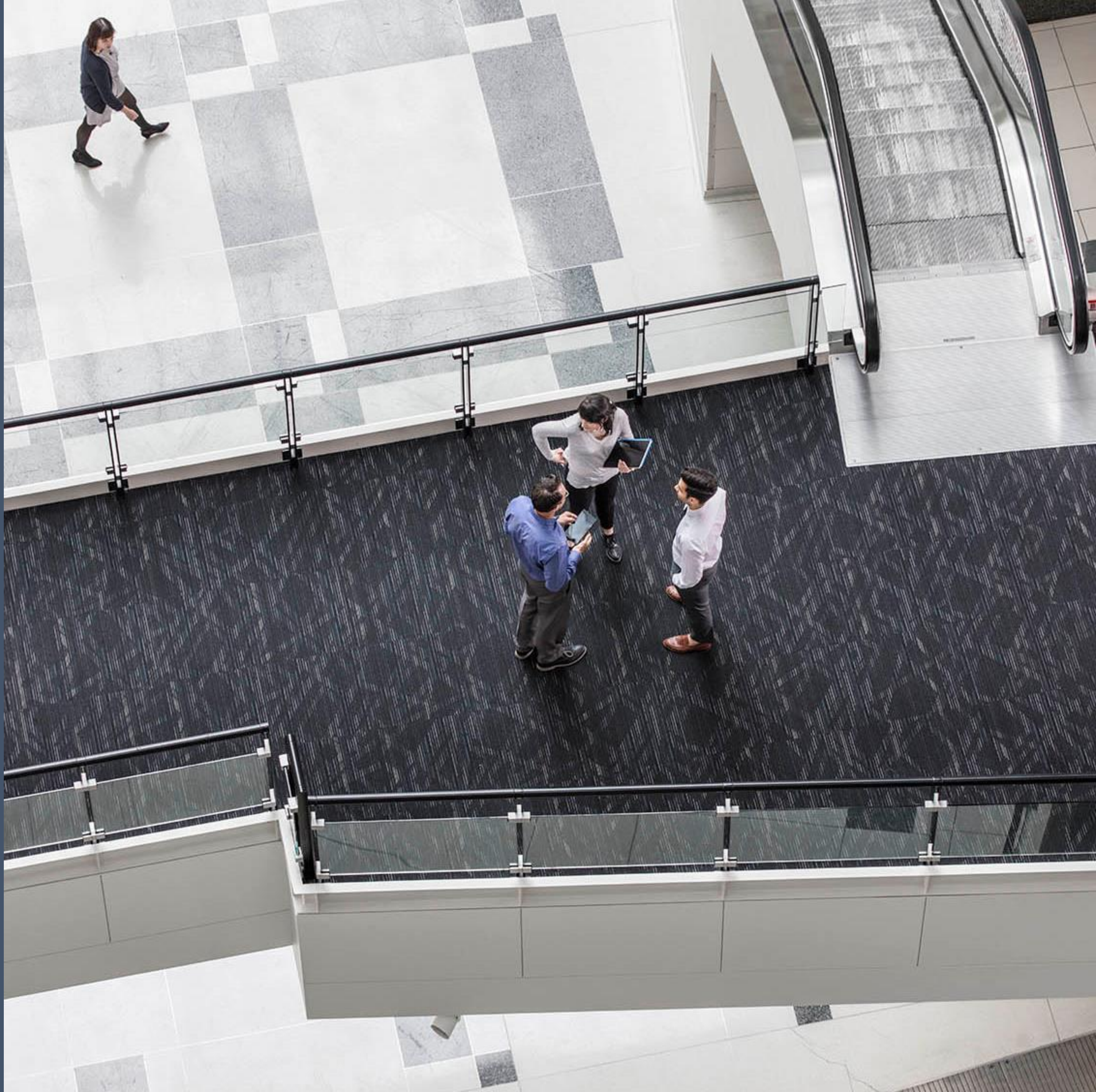
Microsoft

Hadoop 101

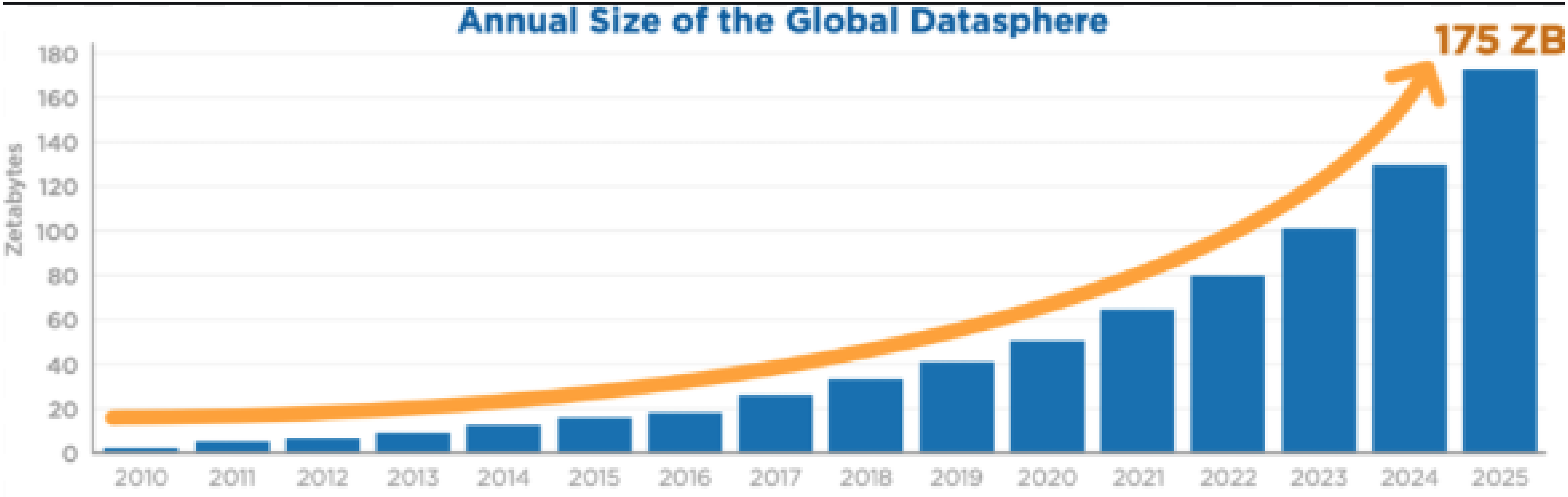
November 2020

Brolinskyi Sergii



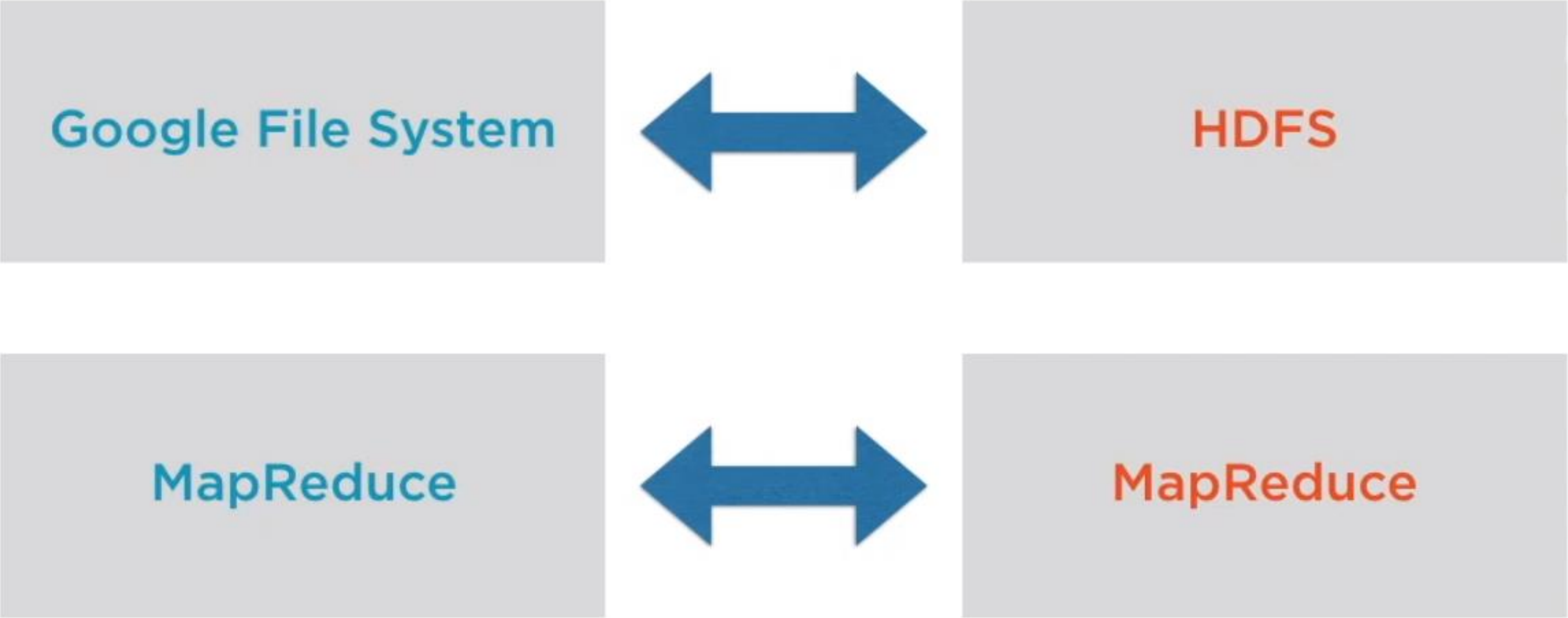
Plan of presentation

- How big is Big Data
- What do you need to do to be able to operate with the Big Data
- Hadoop origins
- What exactly is happening when you want to run a task on Hadoop
- Hadoop ecosystem/tech stack
- HDFS
- YARN
- Demo
- Summary


Annual Size of the Global Datasphere
175 ZB
Zetabytes
0
20
40
60
80
100
120
140
160
180
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025

Big Data System Requirements

Store	Process	Scale
Store massive amounts of data	Process it in a timely manner	Scale easily as data grows



What is Hadoop

Hadoop

HDFS

A file system to manage the storage of data

MapReduce

A framework to process data across multiple servers

Hadoop

HDFS	MapReduce	YARN
	A framework to define a data processing task	A framework to run the data processing task

What happens when you run a job on Hadoop

- User must define map and reduce task using the MapReduce API (those two that we've seen in the lecture 2)
- The job is triggered on the cluster with the help of the YARN
- YARN then figures out where and how to run the job, and store the results files in HDFS

Hadoop Ecosystem

Hive

HBase

Pig

Hadoop

Flume/Sqoop

Spark

Oozie

How to install Hadoop on Win 10

3 ways of installation:

- Standalone mode (no hdfs nor yarn, just checking the MapReduce logic)
- Pseudo-distributed mode (or a single-node mode) (advanced test and simulating an actual cluster)
- Fully distributed mode (the production mode)

How to install Hadoop on Win 10

Single-node mode: 2 JVM processes will run

- HDFS for storage
- YARN for managing tasks

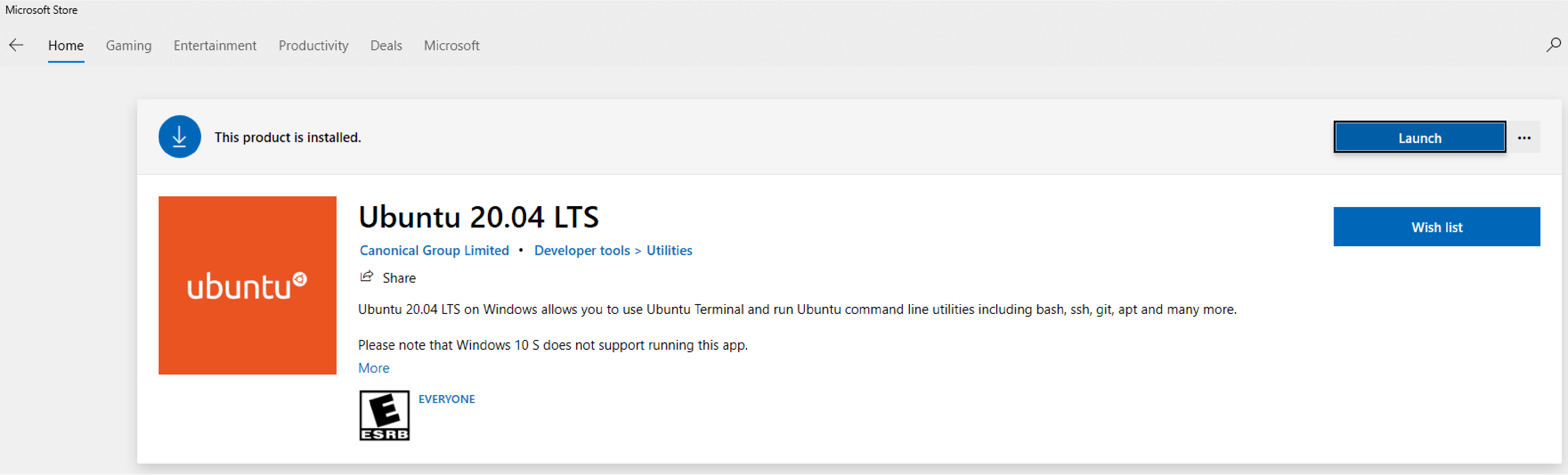
WSL – Windows subsystem for Linux

Enable the feature

Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\sebrolin> Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

Install Ubuntu



Run some code

```
# Install Java and ssh
sudo apt-get update
java -version # if not found than install is needed
sudo apt-get install openjdk-11-jre-headless
sudo apt-get install openjdk-11-jdk
sudo apt-get install ssh
```

Download and unzip Hadoop

```
wget https://miroir.univ-lorraine.fr/apache/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
mkdir ~/hadoop
tar -xvzf hadoop-3.3.0.tar.gz -C ~/hadoop
cd ~/hadoop/hadoop-3.3.0/
```

Setup ssh in a passphrase less mode

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

Format the namenode

```
bin/hdfs namenode -format
```

Configuration of a single node

Files that should be modified according to the setup doc:

- ~/.bashrc
- etc/hadoop/hadoop-env.sh
- etc/hadoop/core-site.xml
- etc/hadoop/hdfs-site.xml
- etc/hadoop/mapred-site.xml
- etc/hadoop/yarn-site.xml

Follow next guides to better help:

- https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
- https://kontext.tech/column/hadoop/445/install-hadoop-330-on-windows-10-using-wsl

Configuration of a single node

sebrolin@MININT-EA70061: ~/hadoop/hadoop-3.3.0

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
             <name>dfs.replication</name>
             <value>1</value>
        </property>
</configuration>
```

Run the hadoop

```
sbin/start-dfs.sh
jps # JPS stands for Java Virtual Machine Process Status Tool
sbin/start-yarn.sh
jps
```

```
sebrolin@MININT-EA70061:~/hadoop/hadoop-3.3.0$ jps
4165 SecondaryNameNode
4598 NodeManager
3895 DataNode
4457 ResourceManager
8953 Jps
3756 NameNode
```

Hadoop cluster

- Cluster
  - About
  - Nodes
  - Node Labels
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed
11	0	0	11

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	
1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type
Capacity Scheduler	[memory-mb (unit=Mi), vcores]

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority
application_1604133744741_0014	sebrolin	grep-sort	MAPREDUCE		default	0
application_1604133744741_0013	sebrolin	grep-search	MAPREDUCE		default	0
application_1604133744741_0012	sebrolin	grep-sort	MAPREDUCE		default	0
application_1604133744741_0011	sebrolin	grep-search	MAPREDUCE		default	0

Hadoop data node

Hadoop   Overview   Utilities

DataNode on MININT-EA70061.europe.corp.microsoft.com:9866

Cluster ID:	CID-3f7df37e-3028-42ca-a6de-594160fb1d0e
Version:	3.3.0, raa96f1871bfd858f9bac59cf2a81ec470da649af

Block Pools

Namenode Address	Block Pool ID	Actor State	Last Heartbeat	Last Block Report	Last Block Report Size (Max Size)
localhost:9000	BP-911422811-127.0.1.1-1604133558936	RUNNING	2s	4 hours	21.17 KB (128 MB)

Volume Information

Directory	StorageType	Capacity Used	Capacity Left	Capacity Reserved	Reserved Space for Replicas	Blocks
/tmp/hadoop-sebrolin/dfs/data	DISK	65.06 MB	157.76 GB	0 B	0 B	2181

HDFS

- HDFS – is Hadoop distributed file system
- - All files are immutable, you can't change them
- - Data is stored in a semi-structured form
- - It is still a file system

HDFS

- Built on commodity hardware
- Highly fault tolerant, hardware failure is the norm
- Suited to batch processing - data access has high throughput rather than low latency
- Supports very large data sets

HDFS

Manage file storage across multiple disks

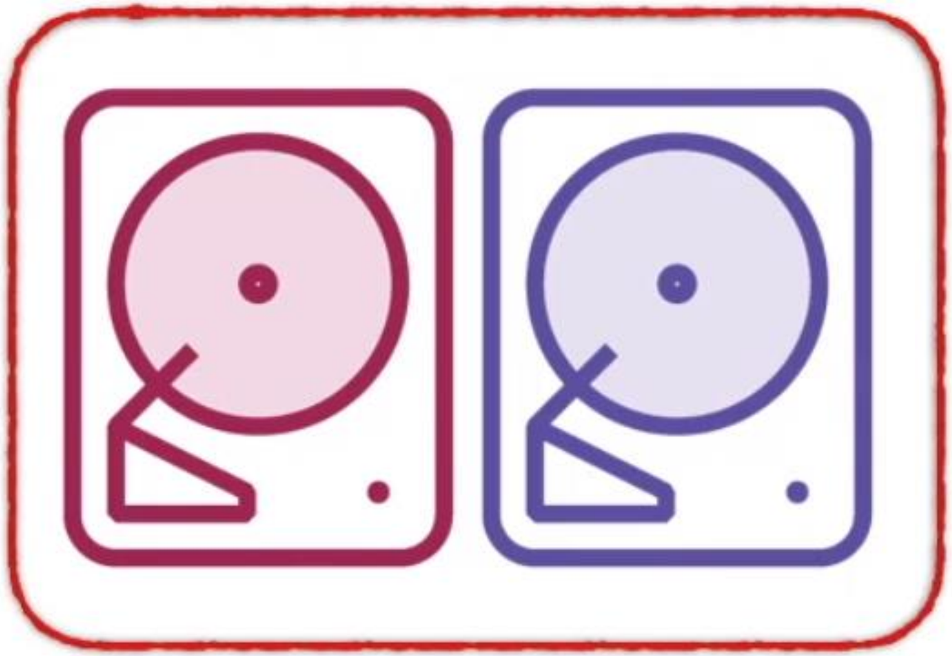
HDFS

1 node is the master node

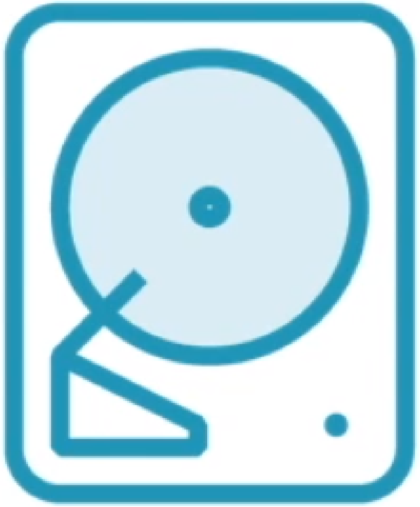
HDFS

Name node

Data nodes

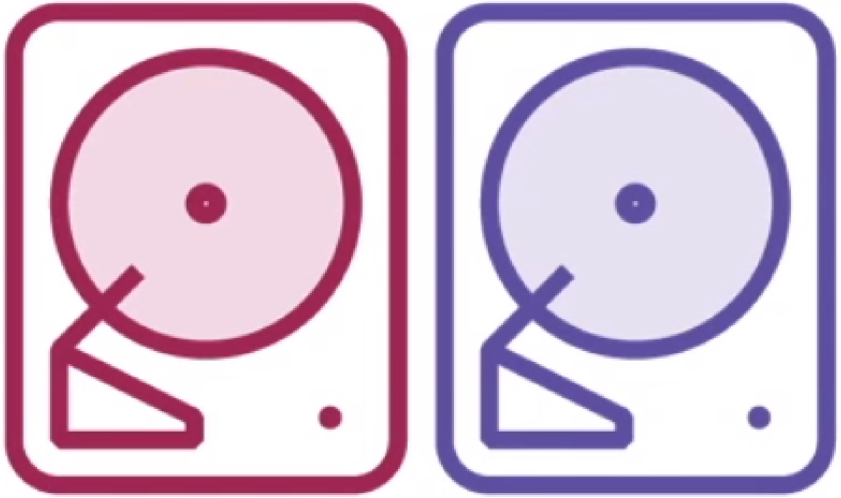


Name node



The name node is the table of contents

Data nodes



The data nodes hold the actual text in each page

Name node

Manages the overall file system

Stores

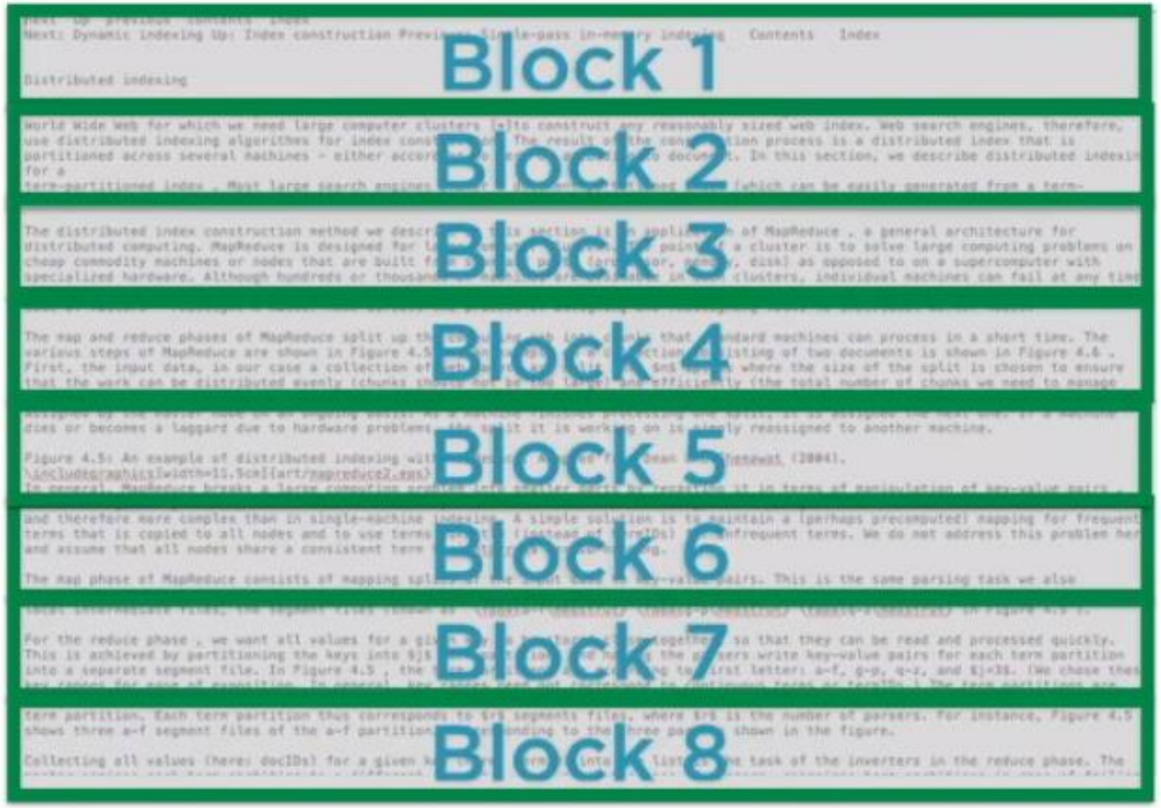
- The directory structure
- Metadata of the files

Storing a File in HDFS

```
next  up  previous  contents  index
Next: Dynamic indexing Up: Index construction Previous: Single-pass in-memory indexing   Contents   Index


Distributed indexing

Collections are often so large that we cannot perform index construction efficiently on a single machine. This is particularly true of the
World Wide Web for which we need large computer clusters [*]to construct any reasonably sized web index. Web search engines, therefore,
use distributed indexing algorithms for index construction. The result of the construction process is a distributed index that is
partitioned across several machines - either according to term or according to document. In this section, we describe distributed indexing
for a
term-partitioned index . Most large search engines prefer a document-partitioned index (which can be easily generated from a term-
partitioned index). We discuss this topic further in Section 20.3 (page [*]).

The distributed index construction method we describe in this section is an application of MapReduce , a general architecture for
distributed computing. MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on
cheap commodity machines or nodes that are built from standard parts (processor, memory, disk) as opposed to on a supercomputer with
specialized hardware. Although hundreds or thousands of machines are available in such clusters, individual machines can fail at any time.
One requirement for robust distributed indexing is, therefore, that we divide the work up into chunks that we can easily assign and - in
case of failure - reassign. A master node directs the process of assigning and reassigning tasks to individual worker nodes.

The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time. The
various steps of MapReduce are shown in Figure 4.5 and an example on a collection consisting of two documents is shown in Figure 4.6 .
First, the input data, in our case a collection of web pages, are split into $n$ splits where the size of the split is chosen to ensure
that the work can be distributed evenly (chunks should not be too large) and efficiently (the total number of chunks we need to manage
should not be too large); 16 or 64 MB are good sizes in distributed indexing. Splits are not preassigned to machines, but are instead
assigned by the master node on an ongoing basis: As a machine finishes processing one split, it is assigned the next one. If a machine
dies or becomes a laggard due to hardware problems, the split it is working on is simply reassigned to another machine.

Figure 4.5: An example of distributed indexing with MapReduce. Adapted from Dean and Ghemawat (2004).
\includegraphics[width=11.5cm]{art/mapreduce2.eps}
In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of key-value pairs .
For indexing, a key-value pair has the form (termID,docID). In distributed indexing, the mapping from terms to termIDs is also distributed
and therefore more complex than in single-machine indexing. A simple solution is to maintain a (perhaps precomputed) mapping for frequent
terms that is copied to all nodes and to use terms directly (instead of termIDs) for infrequent terms. We do not address this problem here
and assume that all nodes share a consistent term $\rightarrow$ termID mapping.

The map phase of MapReduce consists of mapping splits of the input data to key-value pairs. This is the same parsing task we also
encountered in BSBI and SPIMI, and we therefore call the machines that execute the map phase parsers . Each parser writes its output to
local intermediate files, the segment files (shown as  \fbox{a-f\medstrut} \fbox{g-p\medstrut} \fbox{q-z\medstrut} in Figure 4.5 ).

For the reduce phase , we want all values for a given key to be stored close together, so that they can be read and processed quickly.
This is achieved by partitioning the keys into $j$ term partitions and having the parsers write key-value pairs for each term partition
into a separate segment file. In Figure 4.5 , the term partitions are according to first letter: a-f, g-p, q-z, and $j=3$. (We chose these
key ranges for ease of exposition. In general, key ranges need not correspond to contiguous terms or termIDs.) The term partitions are
defined by the person who operates the indexing system (Exercise 4.6 ). The parsers then write corresponding segment files, one for each
term partition. Each term partition thus corresponds to $r$ segments files, where $r$ is the number of parsers. For instance, Figure 4.5
shows three a-f segment files of the a-f partition, corresponding to the three parsers shown in the figure.

Collecting all values (here: docIDs) for a given key (here: termID) into one list is the task of the inverters in the reduce phase. The
```

A large text file

Storing a File in HDFS



Break the data into blocks

Different length files are treated the same way

Storage is simplified

Unit for replication and fault tolerance

Storing a File in HDFS

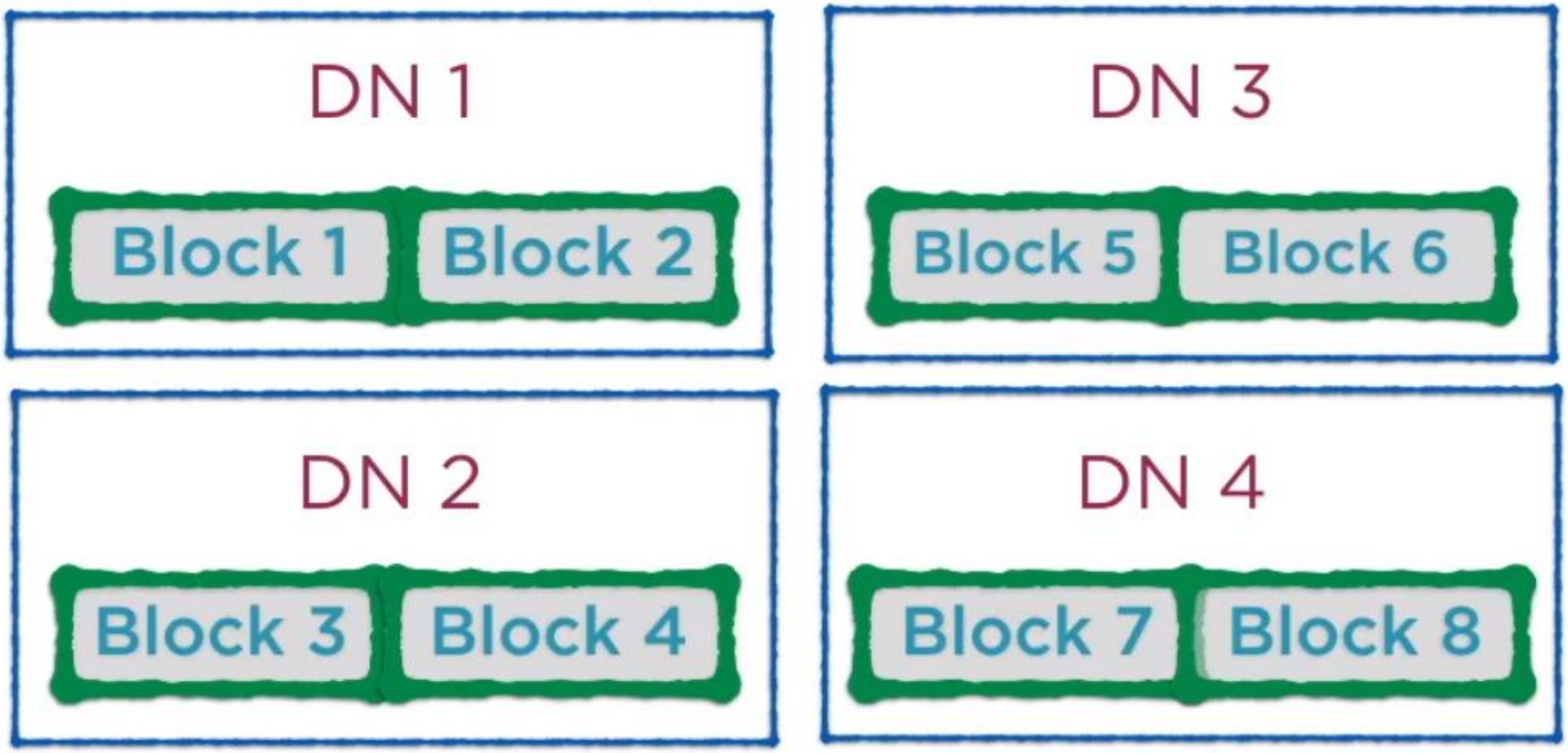
size 128 MB

Block size is a trade off

Reduces parallelism

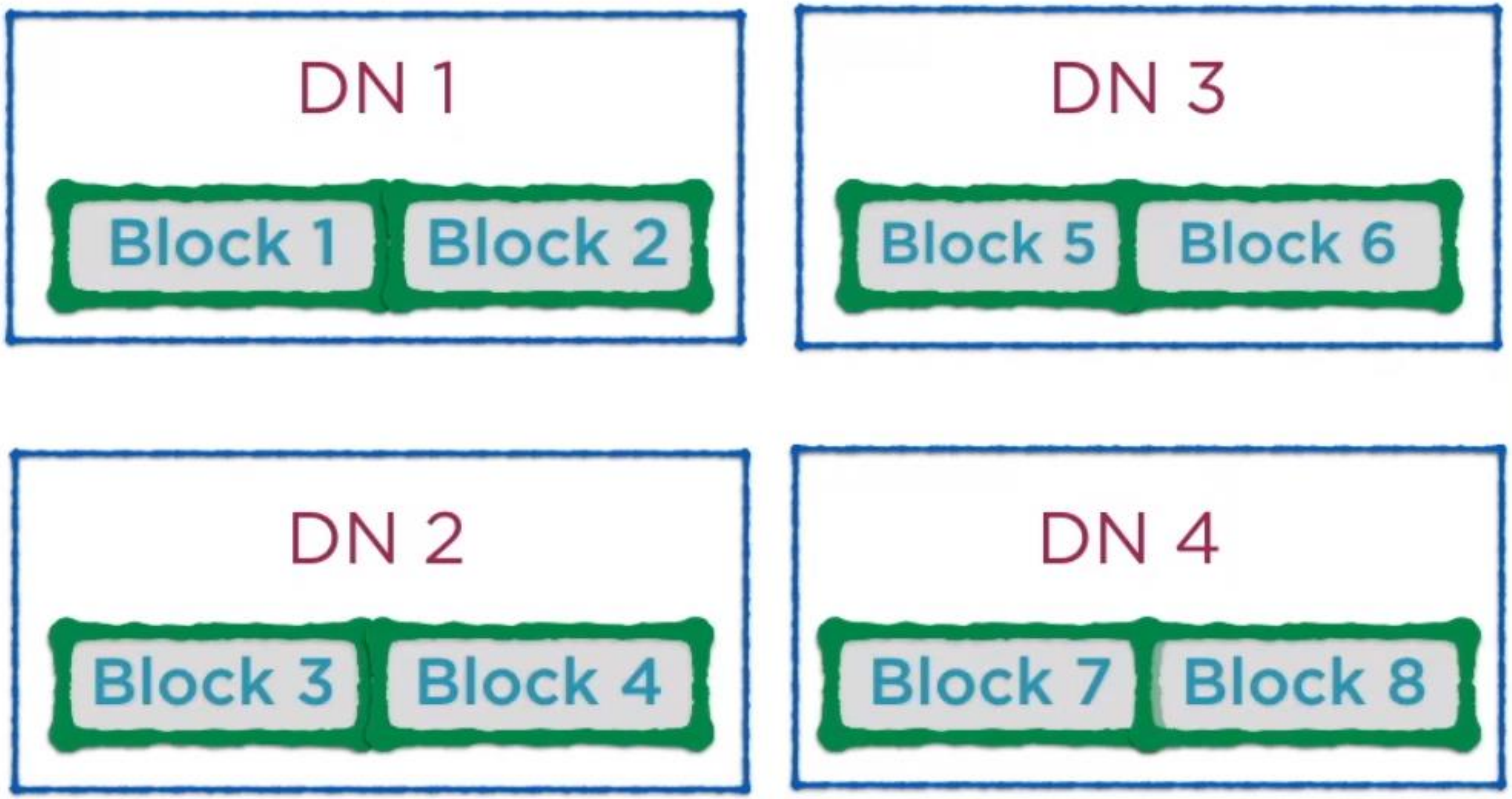
Increases overhead

Storing a File in HDFS



Each node contains a partition or a split of data

Storing a File in HDFS



Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

Reading a File in HDFS

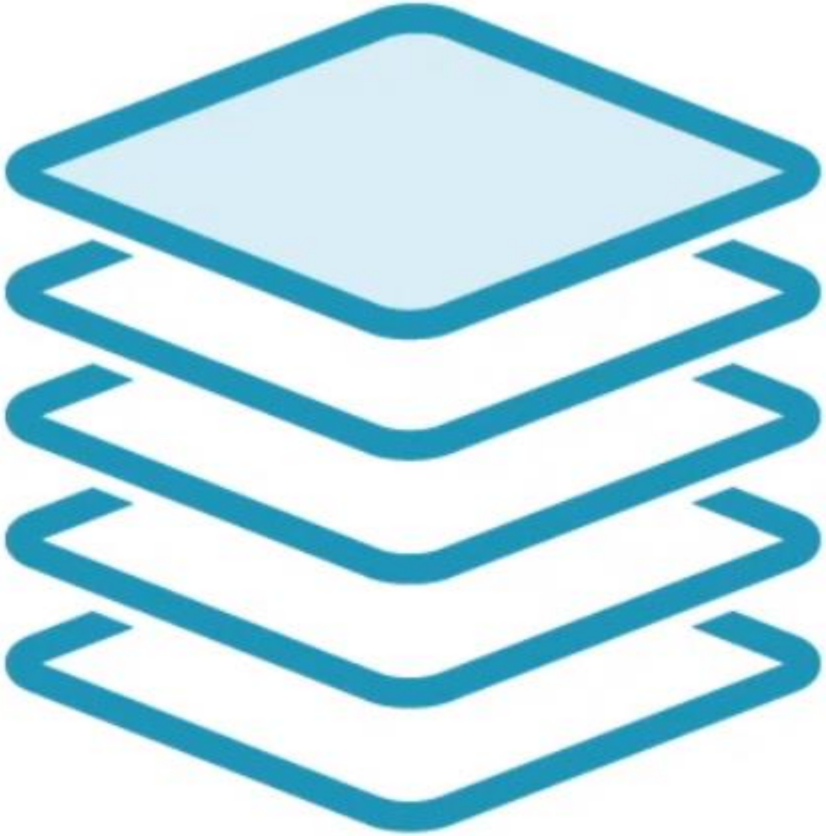
1. Use metadata in the name node to look up block locations
2. Read the blocks from respective locations

Challenges of Distributed Storage

Failure management in the data nodes

Failure management for the name node

Managing Failures in Data Nodes



Define a replication factor

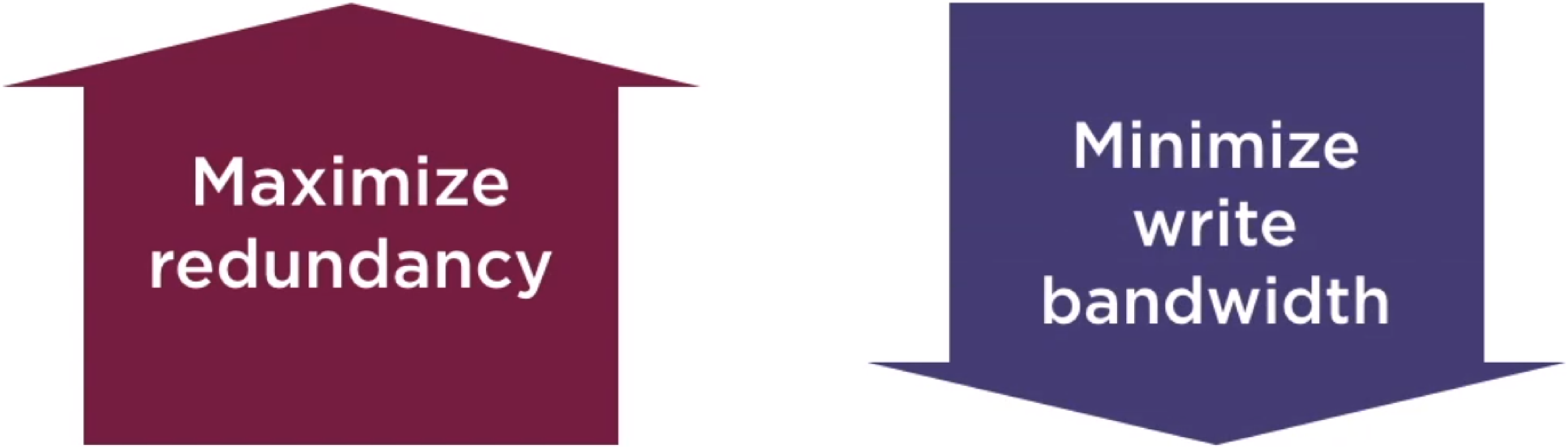
Replication

The replica locations are also stored in the name node

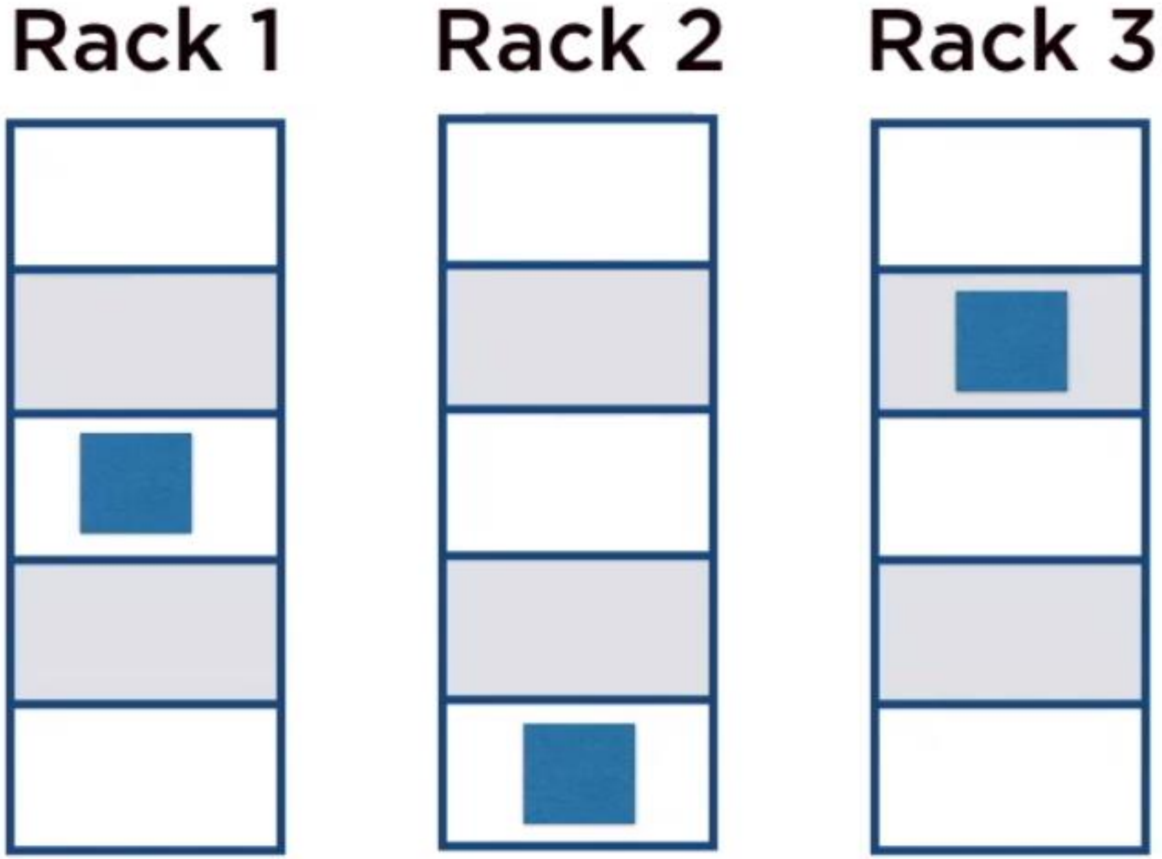
Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3
File 1	Block 1	DN 2

Choosing Replica Locations

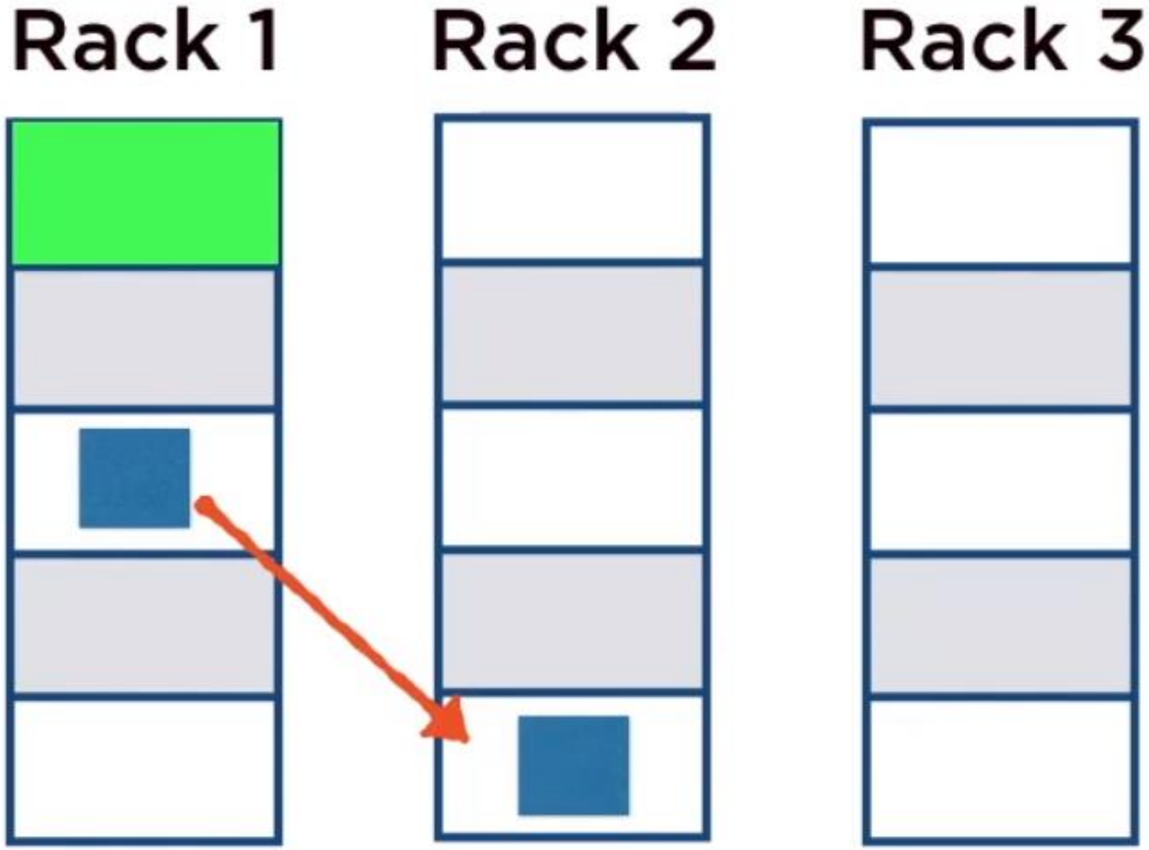


Maximize redundancy



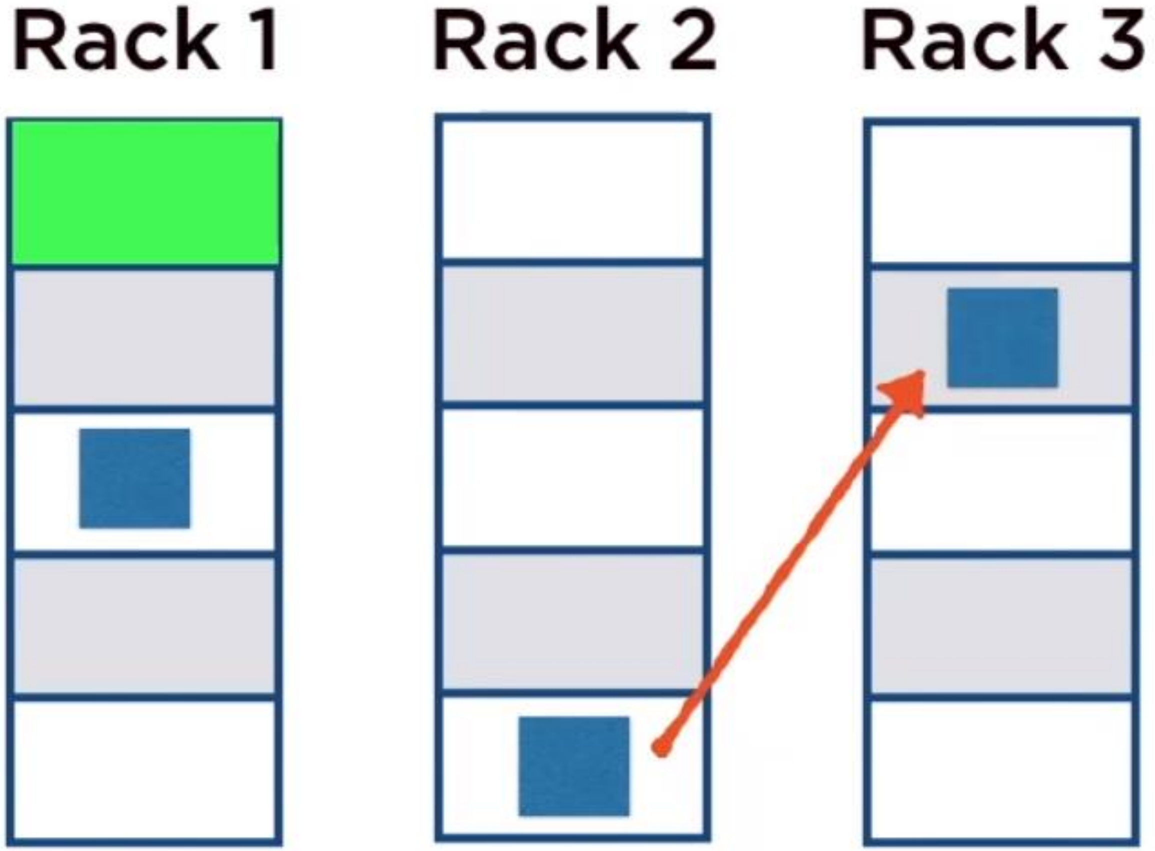
Store replicas “far away”
i.e. on different nodes

Minimize write bandwidth



Data is forwarded from here to the next replica location

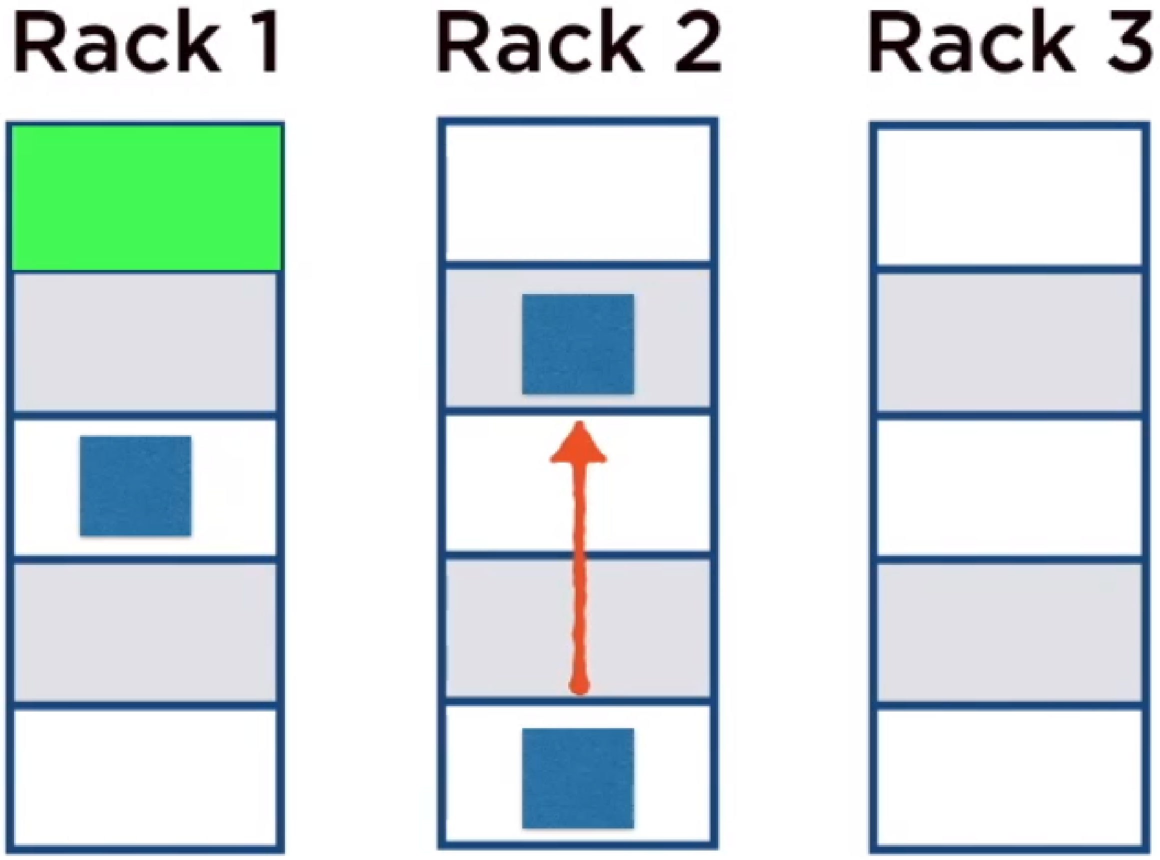
Minimize write bandwidth



Forwarded further to the next replica location

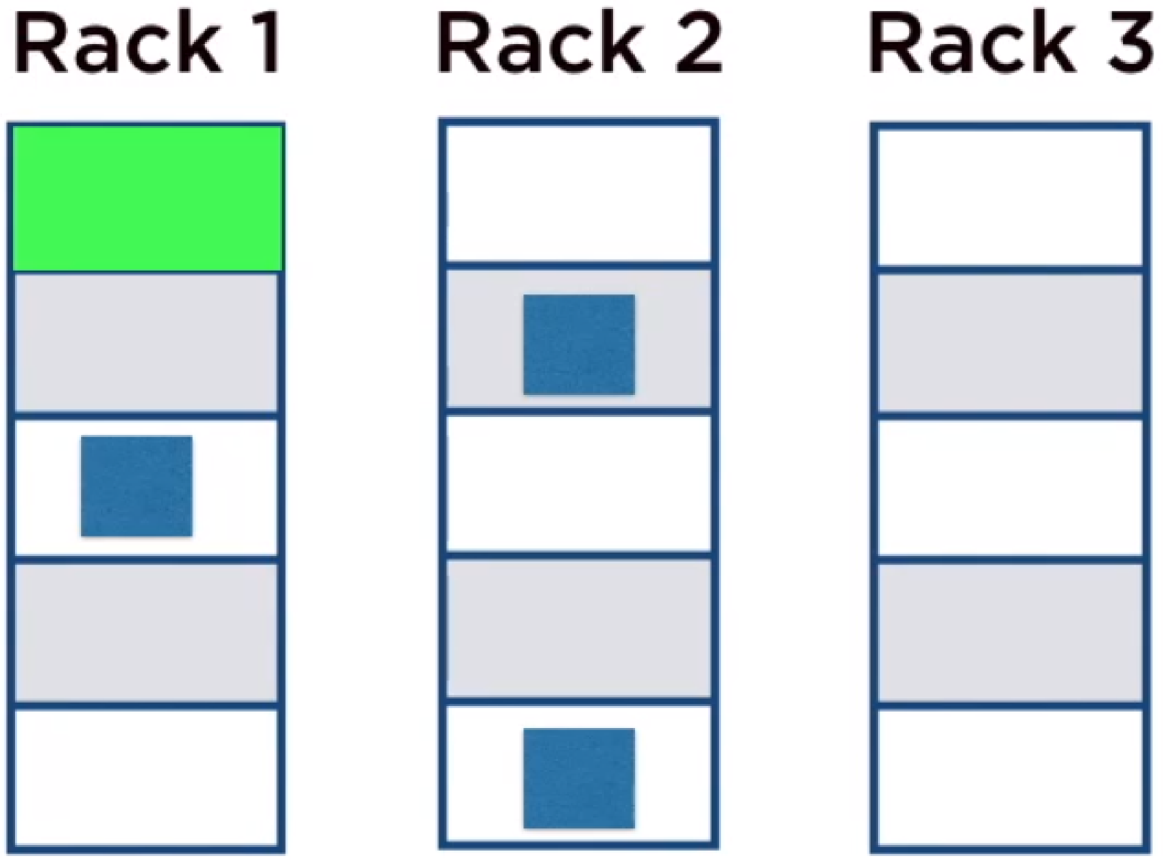
Default Hadoop Replication Strategy

Third replica is on the same rack as the second but on different nodes



Default Hadoop Replication Strategy

Reduces inter-rack traffic and improves write performance



Setting the Replication Factor

`dfs.replication`

3

This is the default in fully-distributed mode

Setting the Replication Factor

`dfs.replication`

1

The pseudo-distributed mode has just one node so the replication factor cannot be >1

Name Node Failures

Block locations are not persistent

i.e. they are stored in memory

block caching

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

Name Node Failures

If the name node fails

File-Block Location mapping is lost!

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

Managing Name Node Failures

Metadata Files

Secondary Name Node

Metadata Files

`fsimage`

`edits`

Two files that store the filesystem metadata

YARN

Yet Another Resource Negotiator

YARN

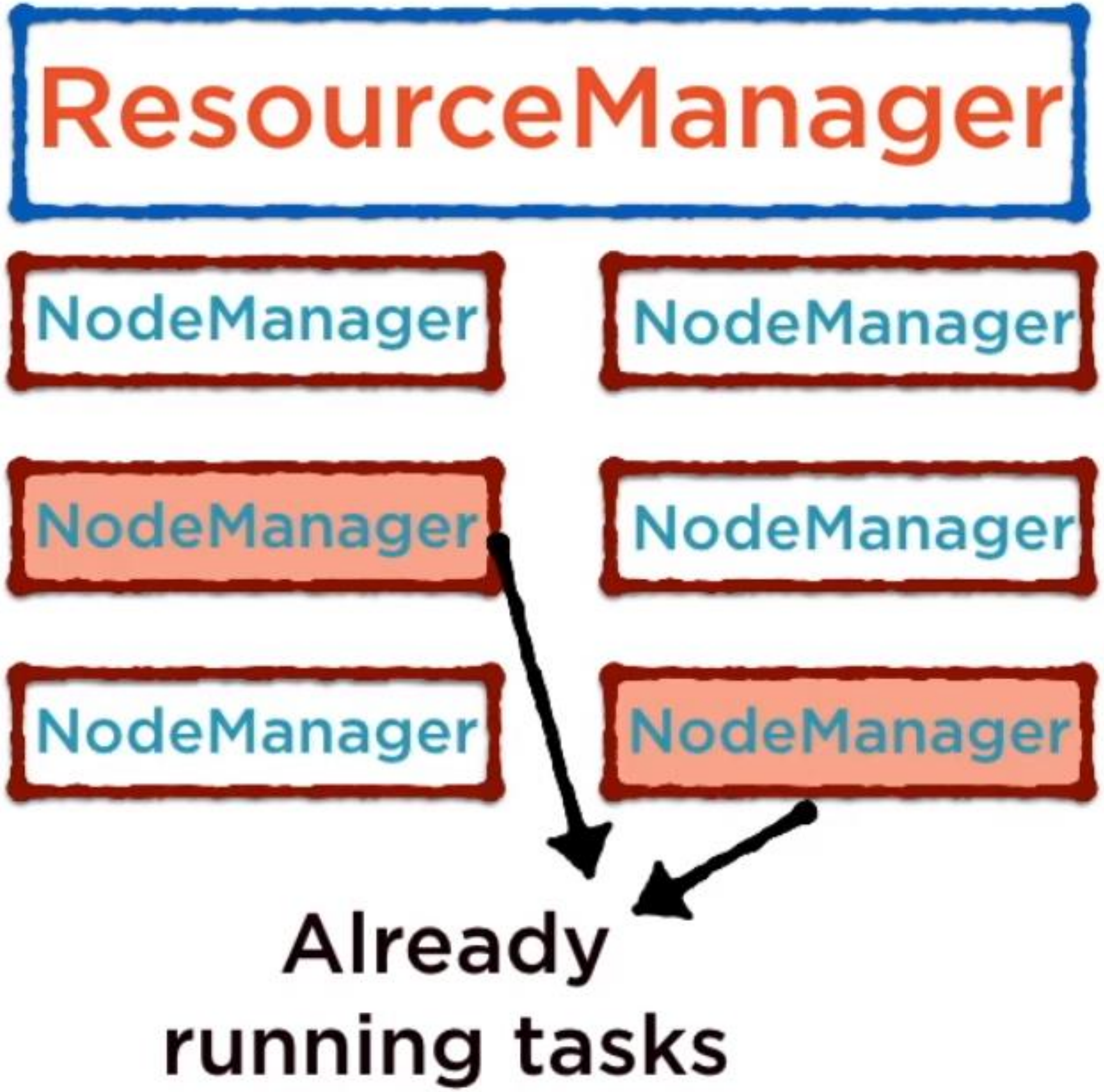
Co-ordinates tasks running on the cluster

Assigns new nodes in case of failure

YARN

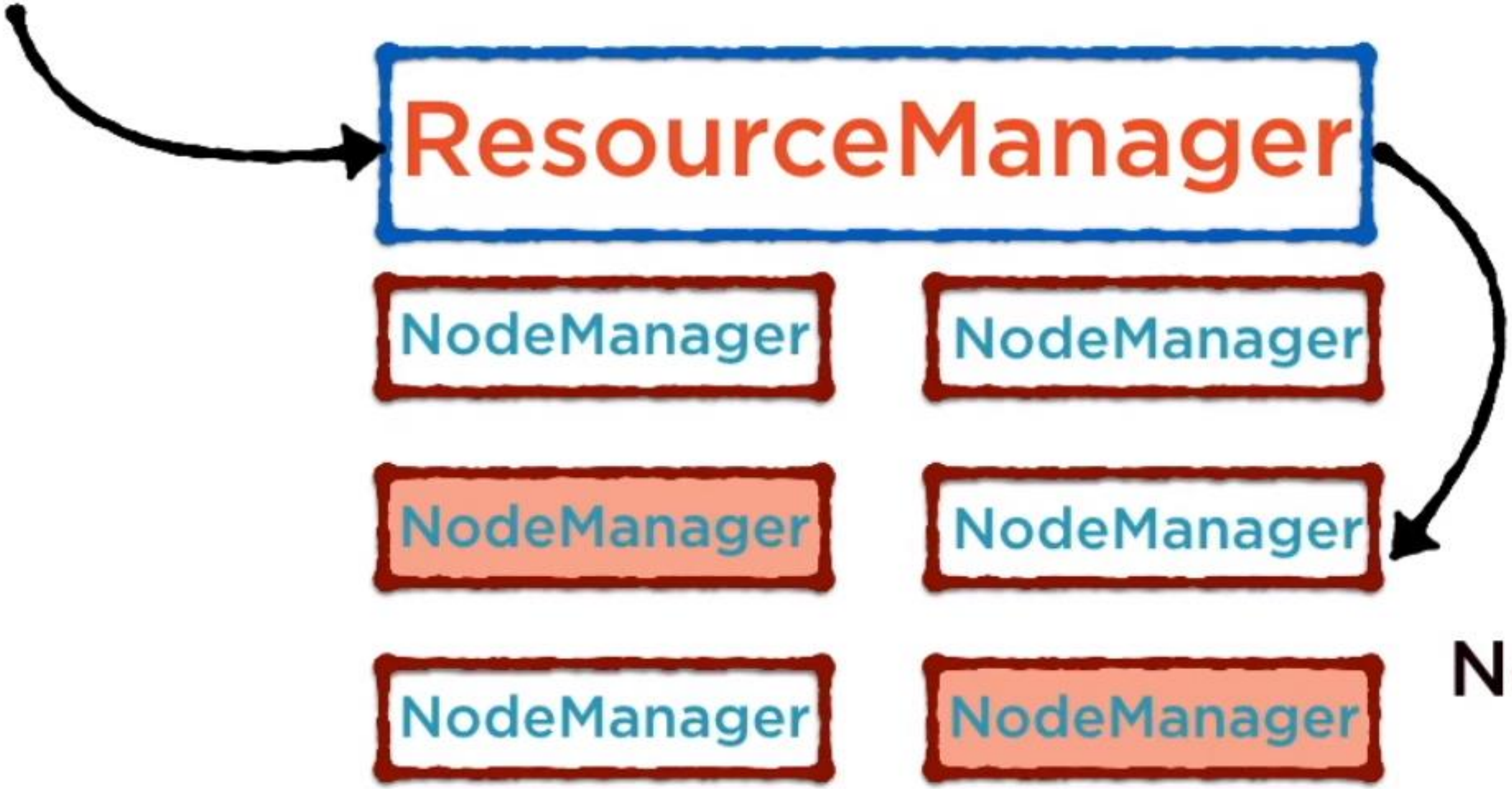
Resource Manager	Node Manager
Runs on a single master node	Run on all other nodes
Schedules tasks across nodes	Manages tasks on the individual node

YARN


ResourceManager
NodeManager
NodeManager
NodeManager
NodeManager
NodeManager
NodeManager
Already
running tasks

Job

Submitting a Job



Find a NodeManager with free capacity

Application Master Process

This is the logical unit for resources the process needs - memory, CPU etc

YARN schedulers

- FIFO scheduler
- Capacity scheduler
- Fair scheduler

Summary

It is complicated to always think about the parallel data processing and manually define the rules of how it should be done, so frameworks add a level of abstraction so you would only need to think about what work should be done.

HDFS, MapReduce and YARN are the building blocks of any Hadoop application

Microsoft