Технології графічного процесінгу

(Масивно-паралельні обчислення на графічних прискорювачах
…
Massively Parallel Computing on Graphic Processing Units - GPUs)

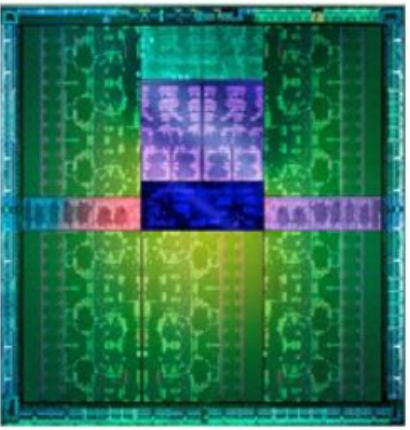
Lecture 6. CUDA Advanced Capabilities: Kepler vs. Fermi

Yuri G. Gordienko
(NTUU-KPI, 2021)

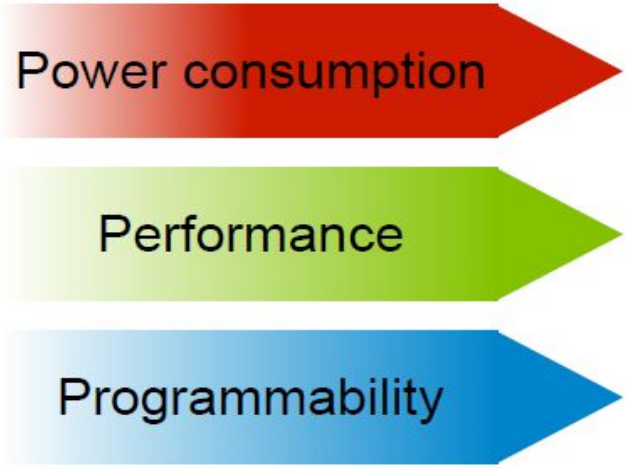
(on the basis of materials by NVIDIA, M.Ujaldon, etc.)

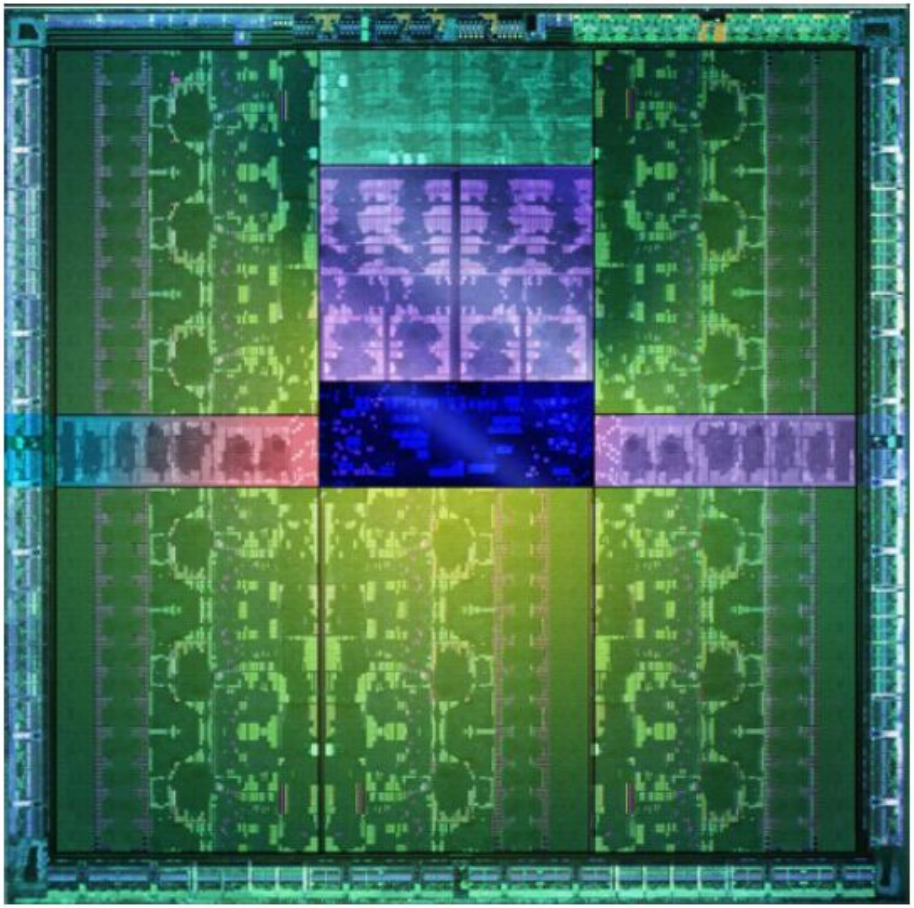
Kepler –> Overview

Kepler Architecture

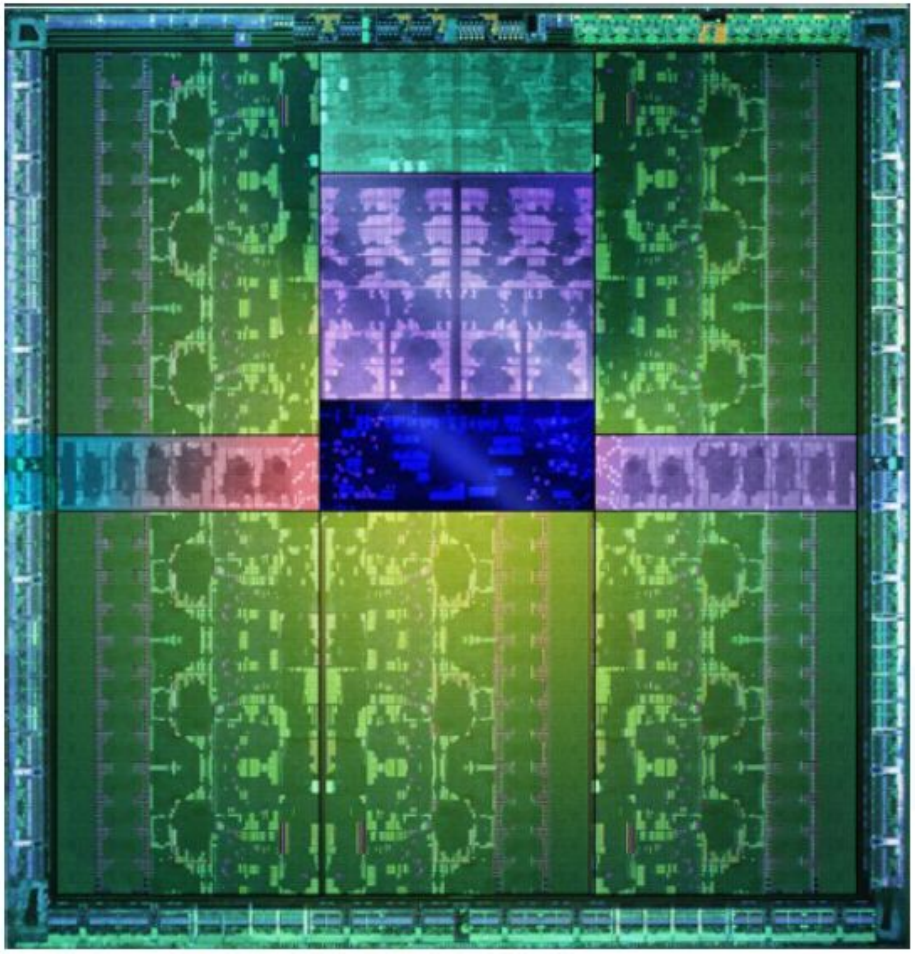


Three Main Benefits of Kepler Architecture





Three Main Benefits and their Tech-Basics of Kepler Architecture



SMX: A multiprocessor with more resources and less power.

Dynamic parallelism: The GPU is autonomous, can launch CUDA kernels.

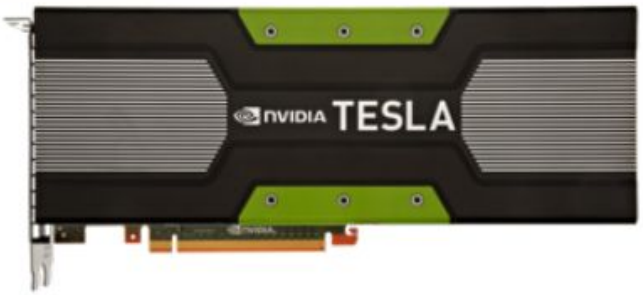
Hyper-Q: Multiple kernels can share the SMXs.

Summary of Improvements: Kepler (K40) versus Fermi (700, not 560!)

Resource	Kepler GK110 vs. Fermi GF100
Floating-point throughput	2-3x
Maximum number of blocks per SMX	2x
Maximum number of threads per SMX	1.3x
Register file bandwidth	2x
Register file capacity	2x
Shared memory bandwidth	2x
Shared memory capacity	1x
L2 bandwidth	2x
L2 cache capacity	2x

Summary on Commercial Implementations of Kepler: GTX Titan versus K40

- Designed for gamers:
  - Price is a priority (<500€).
  - Availability and popularity.
  - Little video memory (1-2 GB.).
  - Frequency slightly ahead.
  - Hyper-Q only for CUDA streams.
  - Perfect for developing code which can later run on a Tesla.



- Oriented to HPC:
  - Reliable (3 year warranty).
  - For cluster deployment.
  - More video memory (6-12 GB.).
  - Tested for endless run (24/7).
  - Hyper-Q for MPI.
  - GPUDirect (RDMA) and other features for GPU clusters.

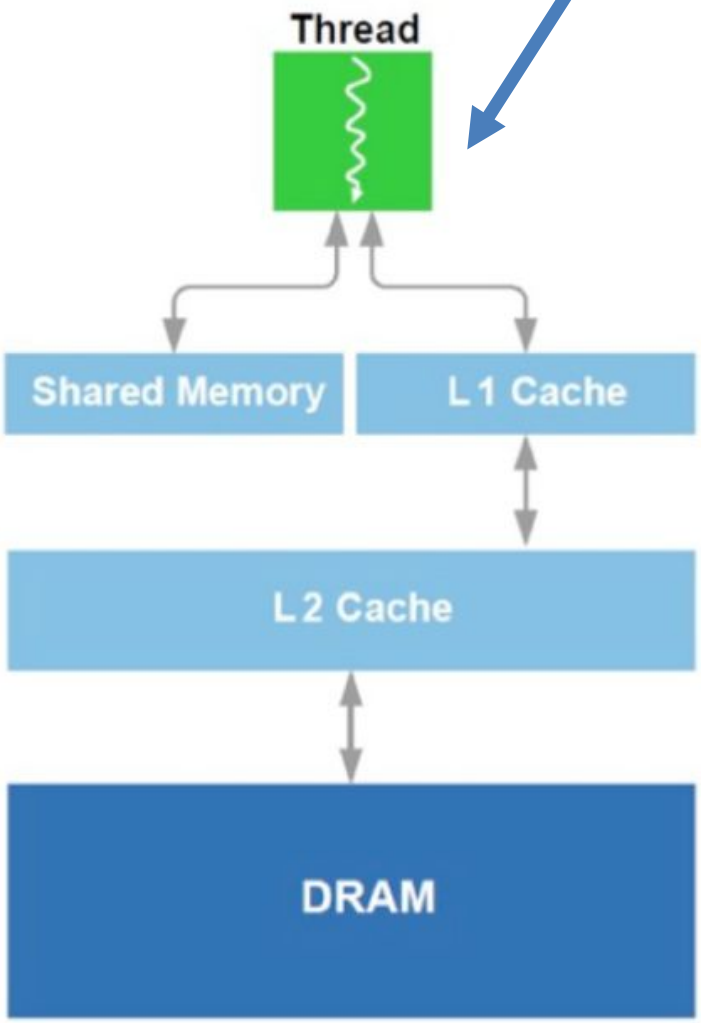
Kepler –> Memory

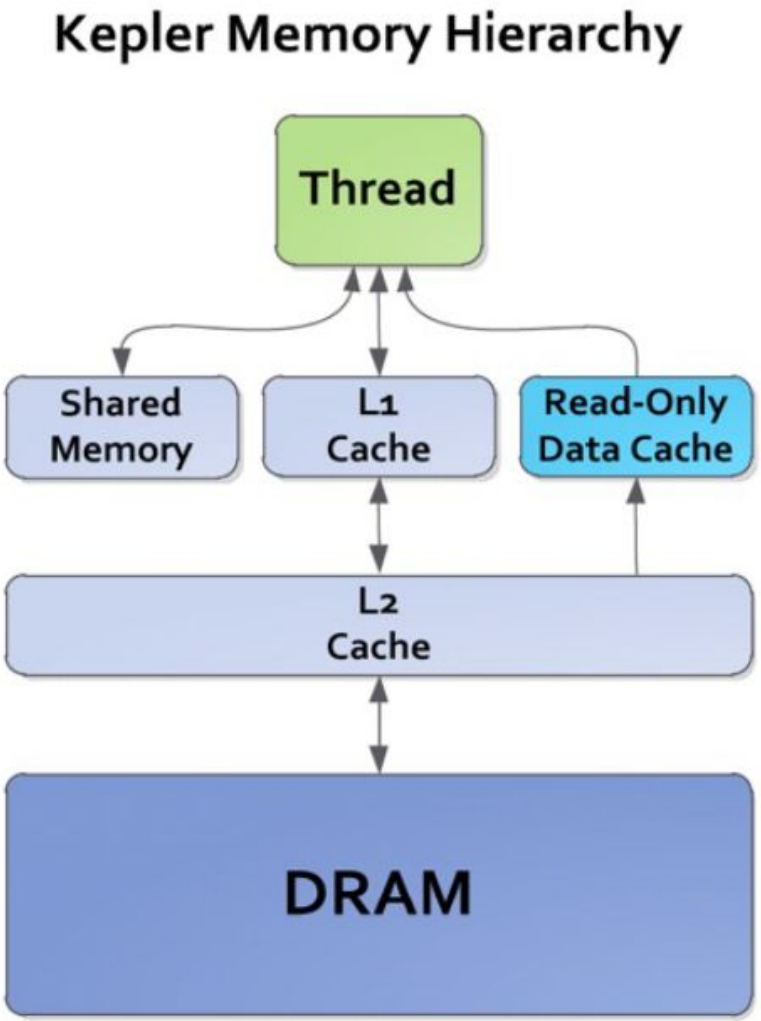
Summary of Memory Improvements: Kepler (K20-K40) vs. Fermi (M2075,M2090)

Tesla card	M2075	M2090	K20	K20X	K40
32-bit register file / multiprocessor	32768	32768	65536	65536	65536
L1 cache + shared memory size	64 KB.	64 KB.	64 KB.	64 KB.	64 KB.
Width of 32 shared memory banks	32 bits	32 bits	64 bits	64 bits	64 bits
SRAM clock freq. (same as GPU)	575 MHz	650 MHz	706 MHz	732 MHz	745,810,875 MHz
L1 and shared memory bandwidth	73.6 GB/s.	83.2 GB/s.	180.7 GB/s	187.3 GB/s	216.2 GB/s.
L2 cache size	768 KB.	768 KB.	1.25 MB.	1.5 MB.	1.5 MB.
L2 cache bandwidth (bytes/cycle)	384	384	1024	1024	1024
L2 on atomic ops. (shared address)	1/9 per clk	1/9 per clk	1 per clk	1 per clk	1 per clk
L2 on atomic ops. (indep. address)	24 per clk	24 per clk	64 per clk	64 per clk	64 per clk
DRAM memory width	384 bits	384 bits	320 bits	384 bits	384 bits
DRAM memory clock (MHz)	2x 1500	2x 1850	2x 2600	2x 2600	2 x 3000
DRAM bandwidth (ECC off)	144 GB/s.	177 GB/s.	208 GB/s.	250 GB/s.	288 GB/s.
DRAM memory size (all GDDR5)	6 GB.	6 GB.	5 GB.	6 GB.	12 GB.
External bus to connect to CPU	PCI-e 2.0	PCI-e 2.0	PCI-e 3.0	PCI-e 3.0	PCI-e 3.0

Difference in Memory Organization:

Fermi versus Kepler





Difference in Memory Organization:
Fermi versus Kepler

GPU generation	Fermi		Kepler		Limi-tation	Impact
Hardware model	GF100	GF104	GK104	GK110		
CUDA Compute Capability (CCC)	2.0	2.1	3.0	3.5		
Max. 32 bits registers / thread	63	63	63	255	SW.	Working set
32 bits registers / Multiprocessor	32 K	32 K	64 K	**64 K**	HW.	Working set
Shared memory / Multiprocessor	16-48KB	16-48KB	16-32-48KB	16-32-48 KB	HW.	Tile size
L1 cache / Multiprocessor	48-16KB	48-16KB	48-32-16KB	48-32-16 KB	HW.	Access speed
L2 cache / GPU	768 KB.	768 KB.	768 KB.	**1536 KB.**	HW.	Access speed

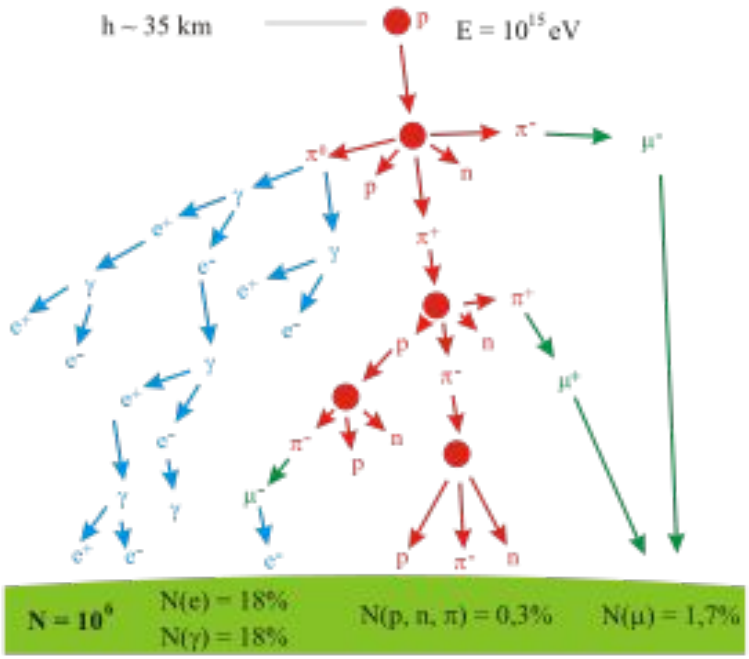
- All Fermi and Kepler models are endowed with:
  - ECC (Error Correction Code) in the video memory controller.
  - Address bus 64 bits wide.
  - Data bus 64 bits wide for each memory controller (few models include 4 controllers for 256 bits, most have 6 controllers for 384 bits)

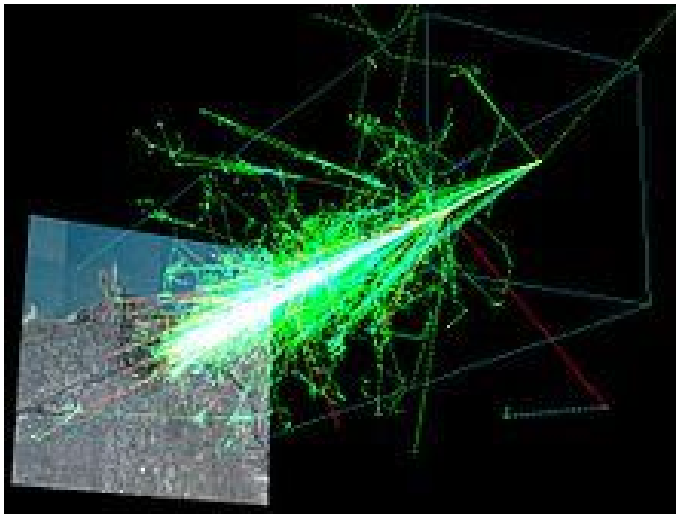
ECC – Error Correction Code: Problem

Electrical or magnetic interference inside computers can cause a single bit of dynamic random-access memory (DRAM) to flip to the opposite state.

Background radiation, chiefly neutrons from cosmic showers (secondary irradiation from high-energy cosmic particles) may change the contents of one or more memory cells or interfere with the circuitry used to read or write to them.

The error rates increase rapidly with altitude: for example, compared to the sea level, the rate of neutron flux is 3.5 times higher at 1.5 km and 300 times higher at 10–12 km (the cruising altitude of commercial airplanes).



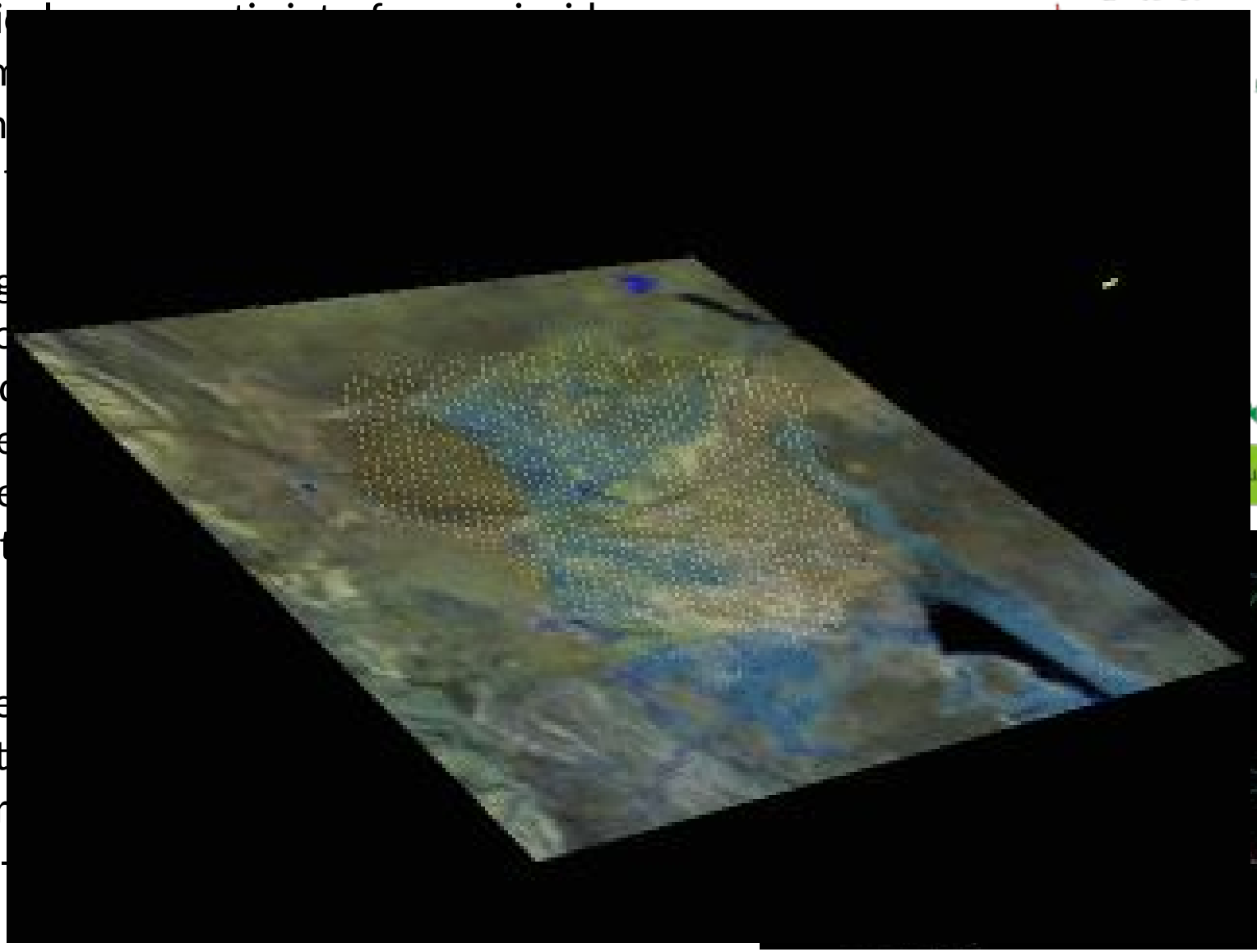


ECC – Error Correction Code: Problem

Electri
com
dynam

Backg
fro
irrac
particle
or more
circuit

The
altitude
level, t
higher
10-

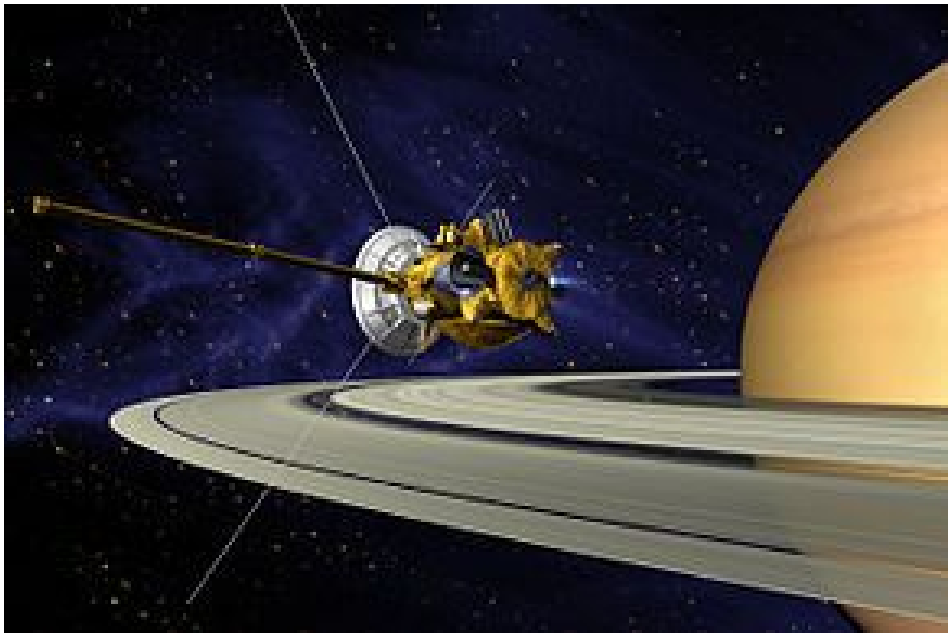


ECC – Error Correction Code: Counter-Measures

With ECC, some portion of the memory is used for ECC bits, so the **available user memory is reduced by 12.5%** (e.g. from 4 GB of total memory user can use ~3.5 GB of available memory only!)

Error-correcting code memory (ECC memory) is a type of computer data storage that can detect and correct the most common kinds of internal data corruption. ECC memory is used in most computers, where **data corruption cannot be tolerated**: for example, for avionics, financial, scientific, military, etc. purposes). Typically, ECC memory maintains a memory system immune to single-bit errors: **the data that is read from each word is always the same as the data that had been written to it**, even if one or more bits actually stored have been flipped to the wrong state. **Most non-ECC memory cannot detect errors** although some non-ECC memory with parity support allows detection but not correction

ECC – Error Correction Code: Example

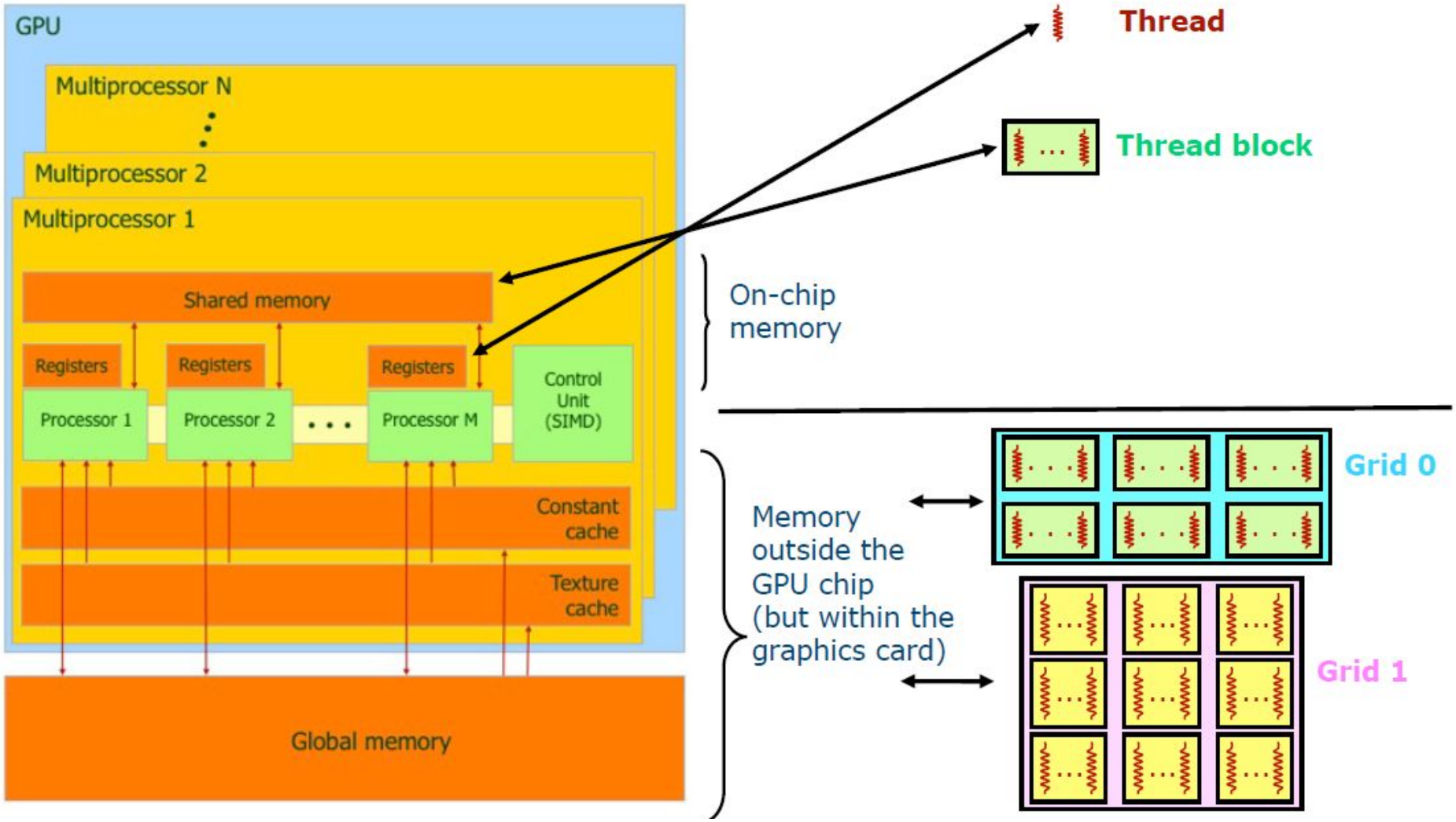


Cassini–Huygens (launched in 1997), contains two identical flight recorders, each with 2.5 gigabits of memory in the form of arrays of commercial DRAM chips. Spacecraft's engineering telemetry reports the number of (correctable) single-bit-per-word errors and (uncorrectable) double-bit-per-word errors. During the first 2.5 years of flight, the spacecraft reported a nearly constant single-bit error rate of about **280 errors per day. However, on November 6, 1997, the number of errors increased by more >4 times for a day.** This was attributed to a solar particle event that had been detected by the satellite GOES 9

Kepler –> SMX

(Streaming Multiprocessor Architecture)

General Organization



Difference in Organization:
From Tesla, and Fermi to Kepler

	Tesla		Fermi		Kepler			
Architecture	**G80**	**GT200**	**GF100**	**GF104**	**GK104 (K10)**	**GK110 (K20)**	**GK110 (K40)**	**GeForce GTX Titan Z**
Time frame	**2006-07**	**2008-09**	**2010**	**2011**	**2012**	**2013**	**2013-14**	**2014**
CUDA Compute Capability (CCC)	**1.0**	**1.2**	**2.0**	**2.1**	**3.0**	**3.5**	**3.5**	**3.5**
N (multiprocs.)	16	30	16	7	8	14	15	30
M (cores/multip.)	8	8	32	48	192	192	192	192
Number of cores	128	240	512	336	1536	2688	2880	5760

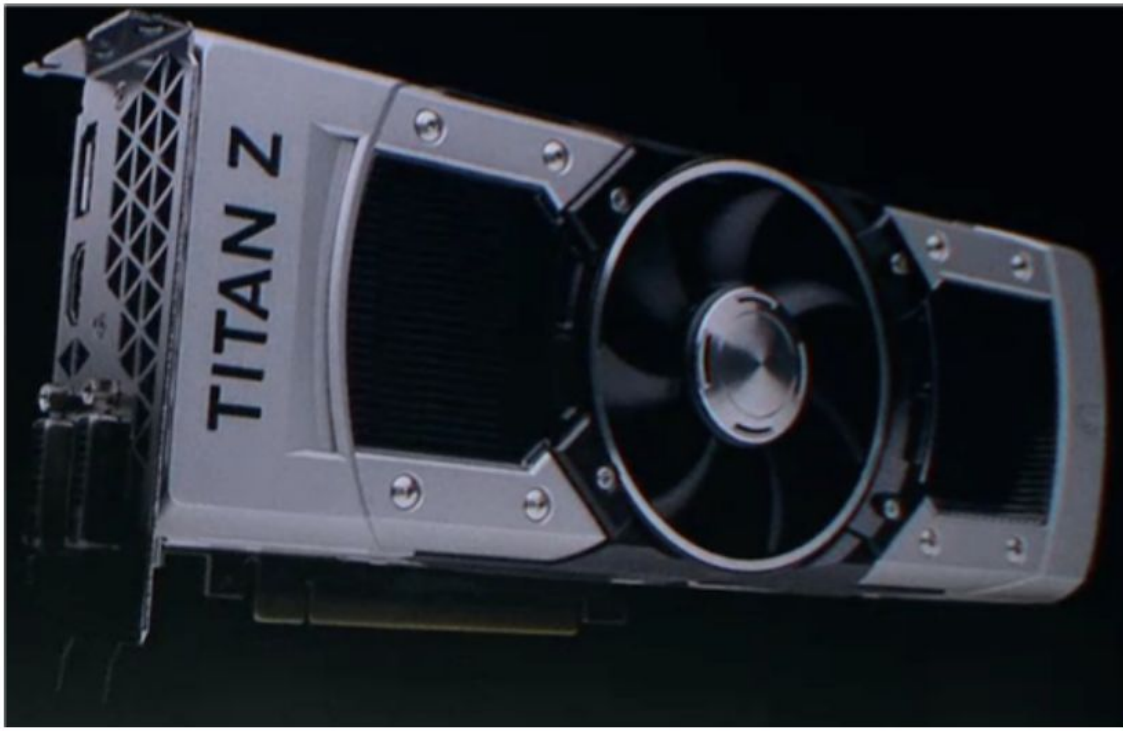
Difference in Organization and Performance

Tesla card (commercial model)	M2075	M2090	K20	K20X	K40
Similar GeForce model in cores	GTX 470	GTX 580	-	GTX Titan	GTX Titan Z (x2)
GPU generation (and CCC)	Fermi GF100 (2.0)		Kepler GK110 (3.5)		
Multiprocessors x (cores/multipr.)	14 x 32	16 x 32	13 x 192	14 x 192	15 x 192
Total number of cores	448	512	2496	2688	2880
Type of multiprocessor	SM		SMX with dynamic paralelism and HyperQ		
Transistors manufacturing process	40 nm.	40 nm.	28 nm.	28 nm.	28 nm.
GPU clock frequency (for graphics)	575 MHz	650 MHz	706 MHz	732 MHz	745,810,875 MHz
Core clock frequency (for GPGPU)	1150 MHz	1300 MHz	706 MHz	732 MHz	745,810,875 MHz
Number of single precision cores	448	512	2496	2688	2880
GFLOPS (peak single precision)	1030	1331	3520	3950	4290
Number of double precision cores	224	256	832	896	960
GFLOPS (peak double precision)	515	665	1170	1310	1680

What is Titan X?

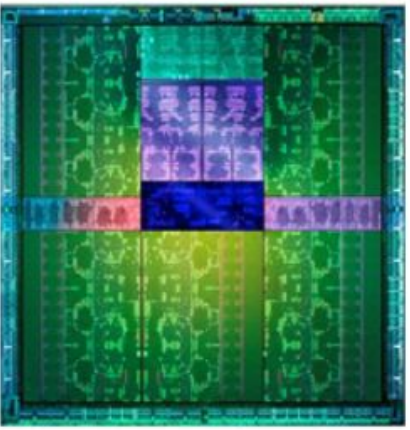
It's TWO K40

- 5760 cores (2x K40).
- Video memory: 12 Gbytes.
- Peak performance: 8 TeraFLOPS.
- Starting price: $2999.



Kepler –> SMX
(Streaming Multiprocessor Architecture)
at Work

Kepler GK110: Physical layout of functional of Tesla K40 (with 15 SMX)



SMX in Kepler GK110

Instruction scheduling and issuing in **warps**

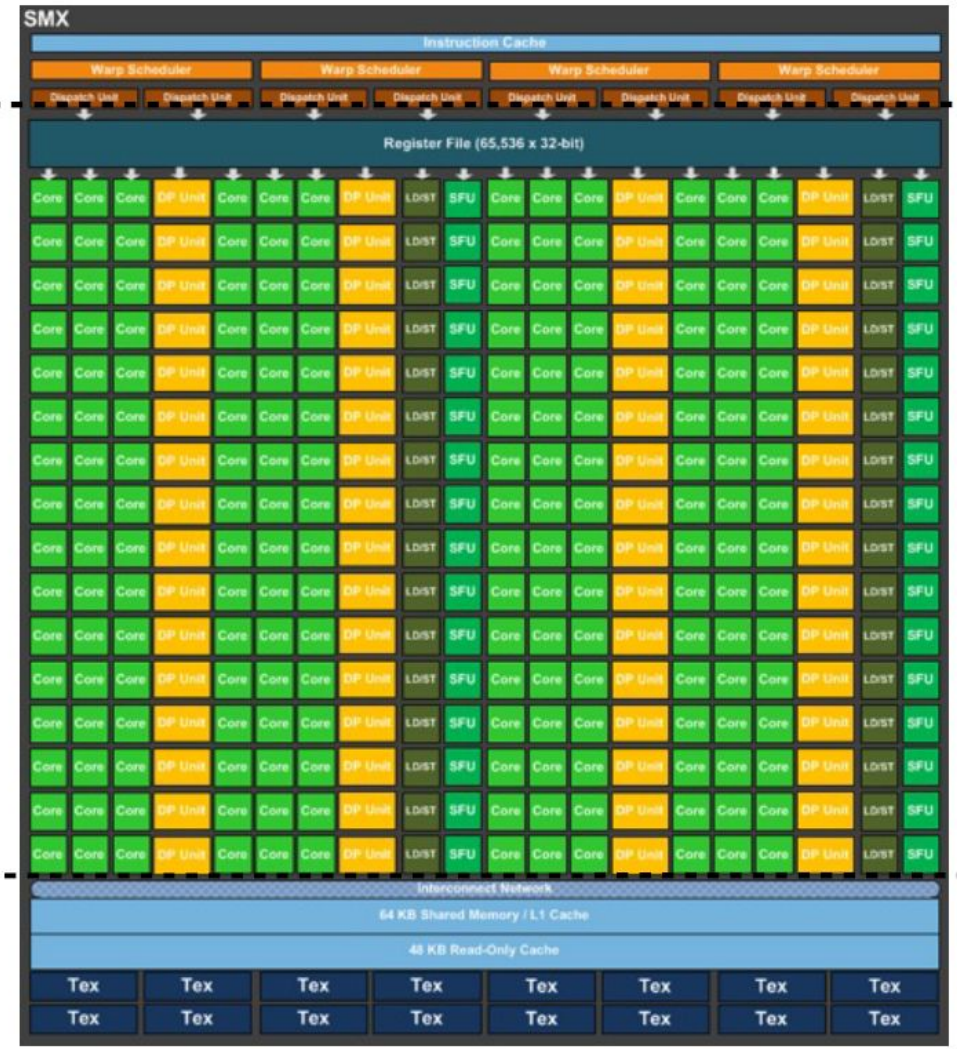
Front-end

Instructions execution.
512 functional units:
- 192 for ALUs.
- 192 for FPUs S.P.
- 64 for FPUs D.P.
- 32 for load/store.
- 32 for SFUs (log,sqrt, ...)

Back-end

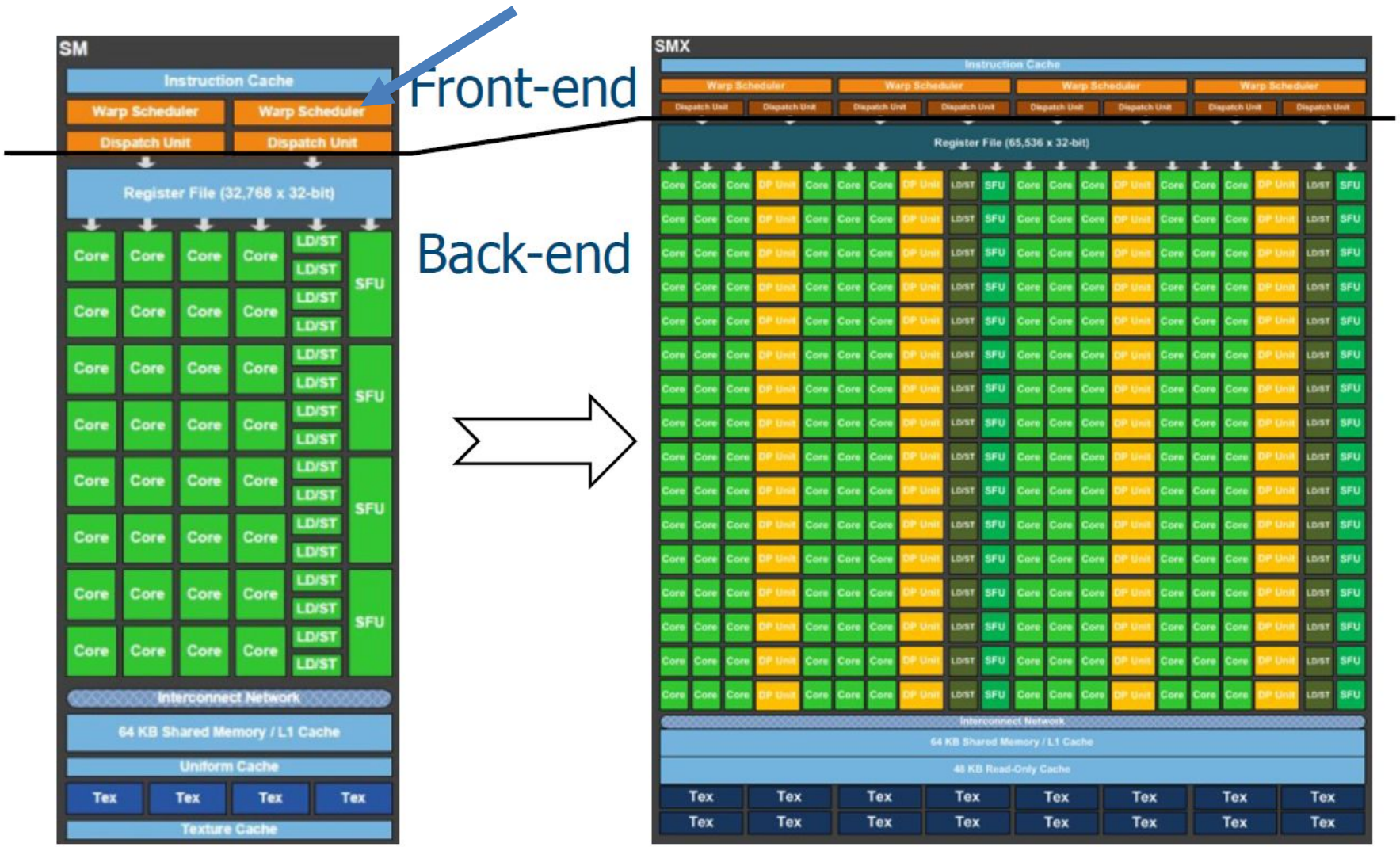
Memory access

Interface



Difference in SMX Organization:

Fermi versus Kepler

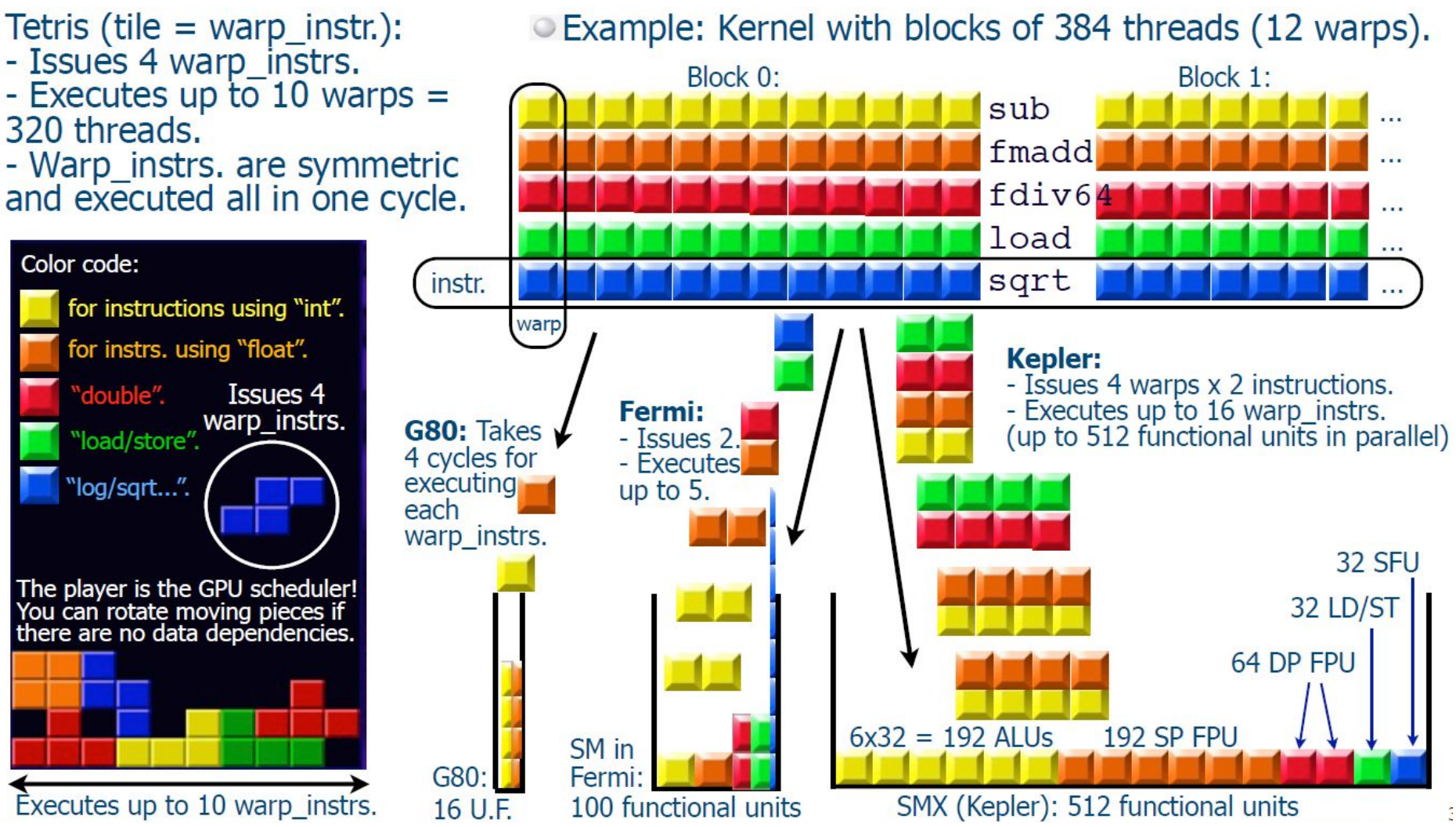


Difference in SMX Organization:
Fermi versus Kepler

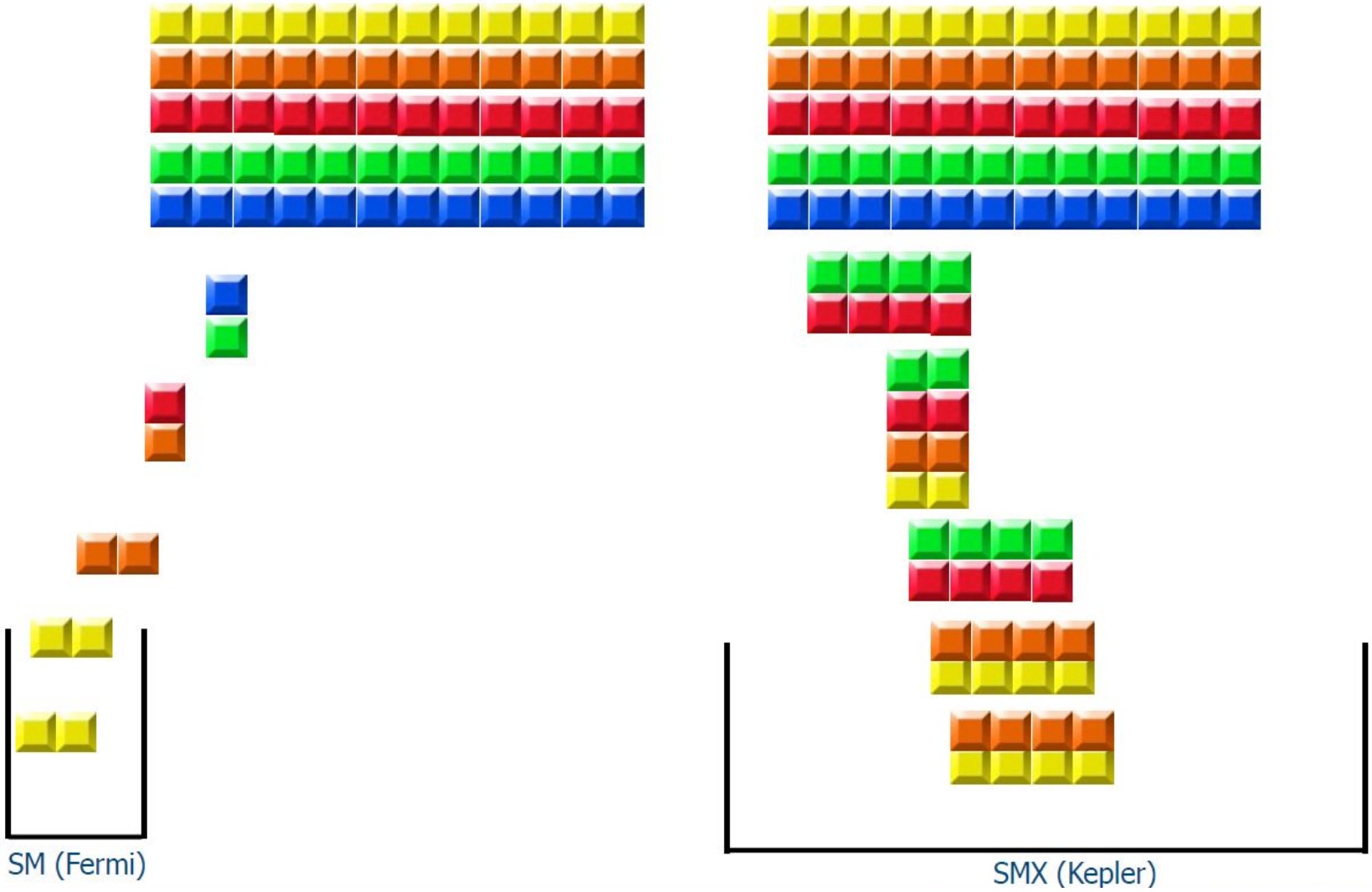
	SM-SMX fetch & issue (front-end)	SM-SMX execution (back-end)
Fermi (GF100)	Can issue 2 warps, 1 instruction each. Total: Up to **2 warps per cycle**. Active warps: 48 on each SM, chosen from up to 8 blocks. In GTX580: 16 * 48 = 768 active warps.	32 cores [1 warp] for "int" and "float". 16 cores for "double" [1/2 warp]. 16 load/store units [1/2 warp]. 4 special function units [1/8 warp]. A total of up to **5 concurrent warps**.
Kepler (GK110)	Can issue 4 warps, 2 instructions each. Total: Up to **8 warps per cycle**. Active warps: 64 on each SMX, chosen from up to 16 blocks. In K40: 15 * 64 = 960 active warps.	192 cores [6 warps] for "int" and "float". 64 cores for "double" [2 warps]. 32 load/store units [1 warp]. 32 special function units [1 warp]. A total of up to **16 concurrent warps**.

- In Kepler, each SMX can issue 8 warp-instructions per cycle, but due to resources and dependencies limitations:
  - 7 is the sustainable peak.
  - 4-5 is a good amount for instruction-limited codes.
  - <4 in memory- or latency-bound codes.

Difference in Parallelism Width per SMX:
G80 versus Fermi versus Kepler

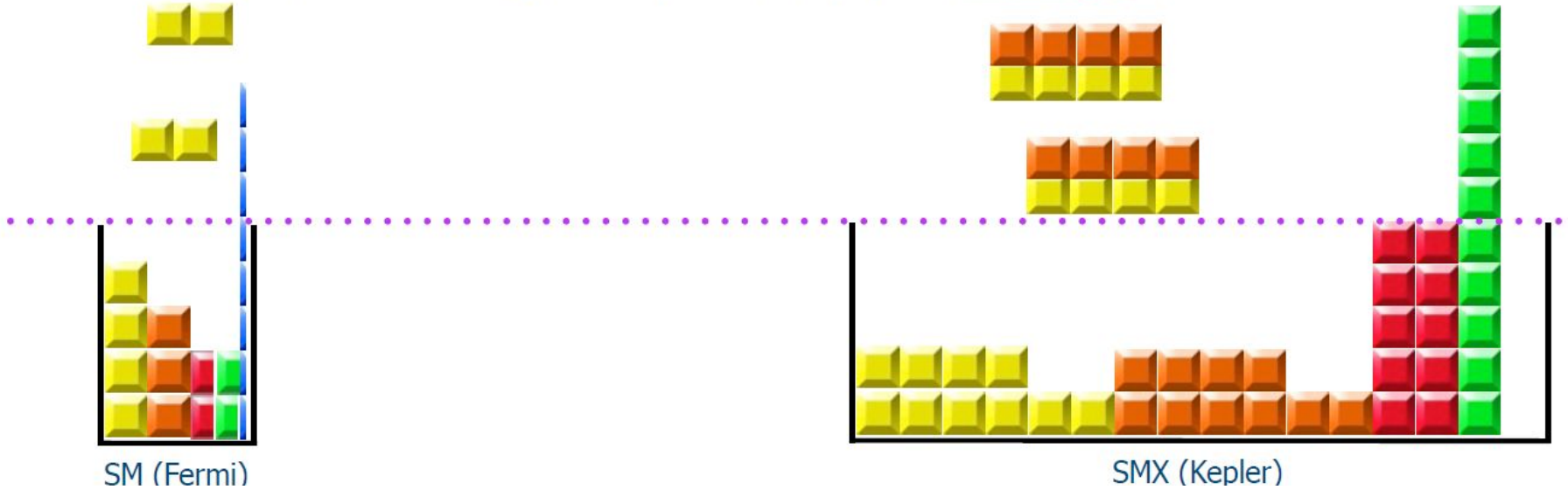


How warps are Scheduled: Fermi versus Kepler



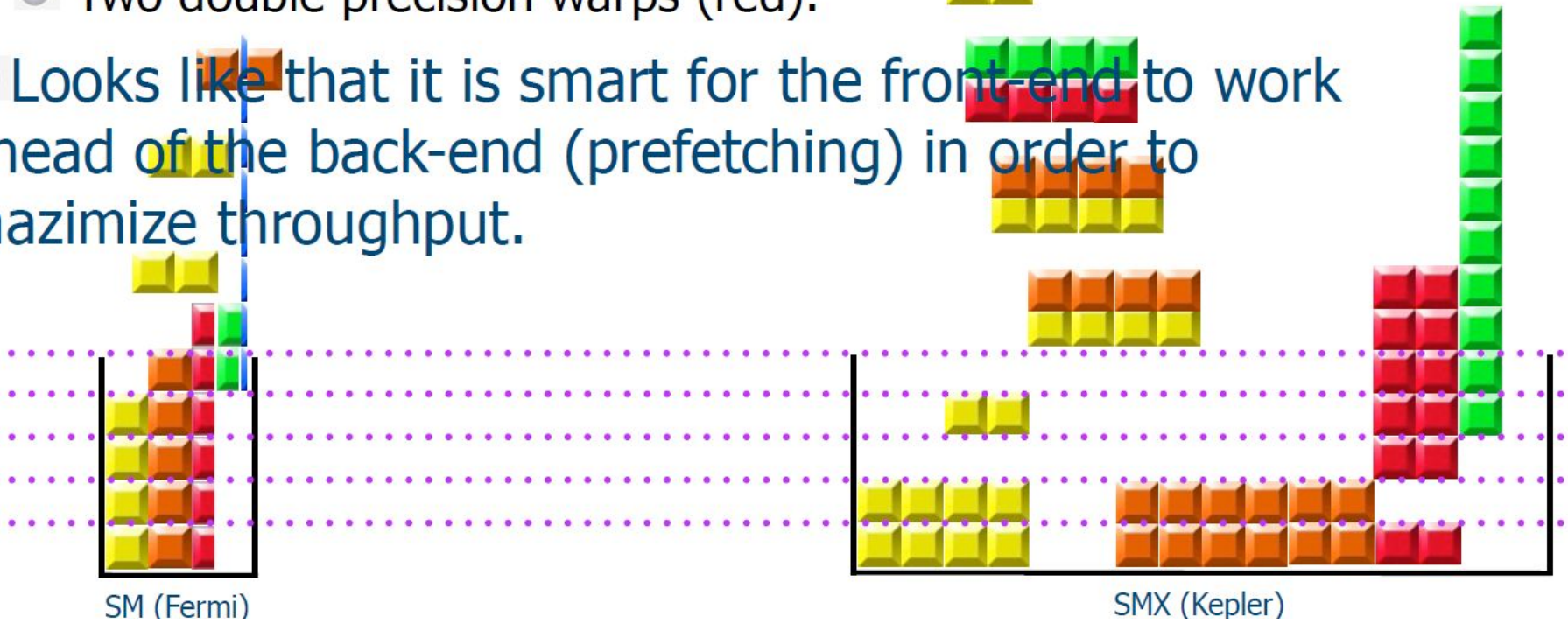
How warps are Issued: Fermi versus Kepler

- In the 5 cycles shown, we could have executed all this work
  - In Fermi, there is a deficit in SFUs (blue), whereas in Kepler, the deficit goes to load/store units (green).
  - Kepler balances double precision (red) and has a good surplus in "int" and "float" computations, an evidence that real codes have more presence of orange and, overall, yellow instructions.



How warps are Executed: Fermi versus Kepler

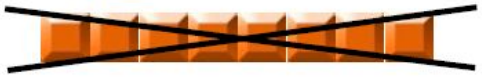
- Let us assume that when we start the execution there are few warps pending to be executed:
  - Two single precision warps (orange).
  - Two double precision warps (red).
- Looks like that it is smart for the front-end to work ahead of the back-end (prefetching) in order to mazimize throughput.

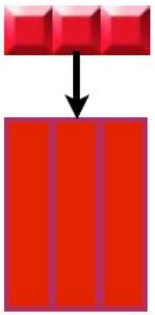


Comments as to this “Tetris” analogy: Fermi versus Kepler

- In Fermi, red tiles are not allowed to be combined with others.

- In Kepler, we cannot take 8 warp_instrs. horizontally, bricks must have a minimum height of 2.





- Instructions have different latency, so those consuming more than one cycle (i.e. double precision floating-point) should expand vertically.
- In case the warp suffers from divergencies, it will consume two cycles, not one. We can extend it vertically like in the previous case.
- Real codes have a mayority of yellow tiles (“int” predominates).
- Some bricks are incomplete, because the warp scheduler cannot find a 4x2 structure free of dependencies.

- Bricks can assemble tiles which are not contiguous.

Overheads (latency) as to Warps

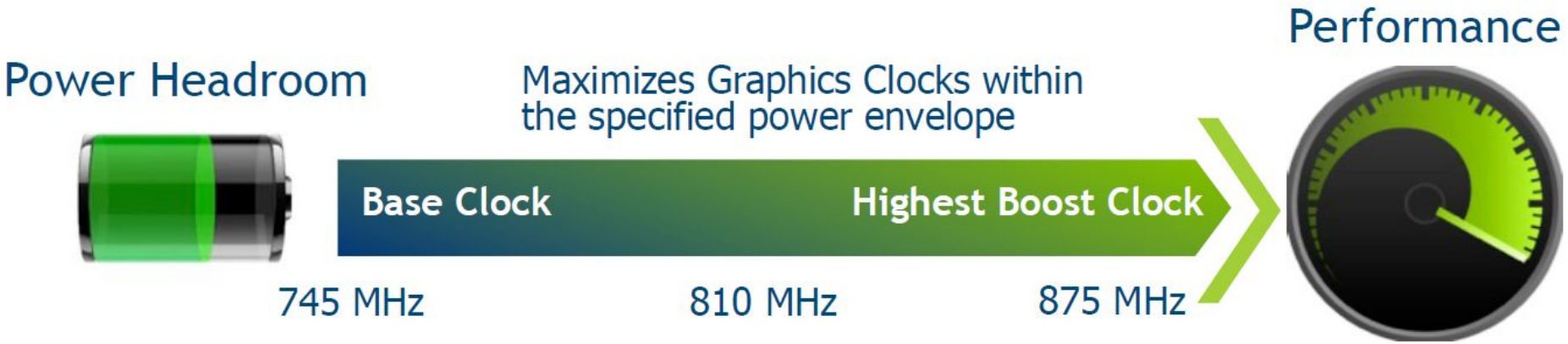
- Even if all tiles be executed in one cycle, warps duration would not be that one. The time elapsed by a warp within the GPU is the addition of three:
  - Scheduling time.
  - Issuing time.
  - Execution time.
- Scheduling/execution are quite regular, but issuing is not: It depends on tiles piled up at the bottom of the bucket (reserve stations). That is what explains the variance of its duration.

Kepler –>

New Feature: GPU Boost

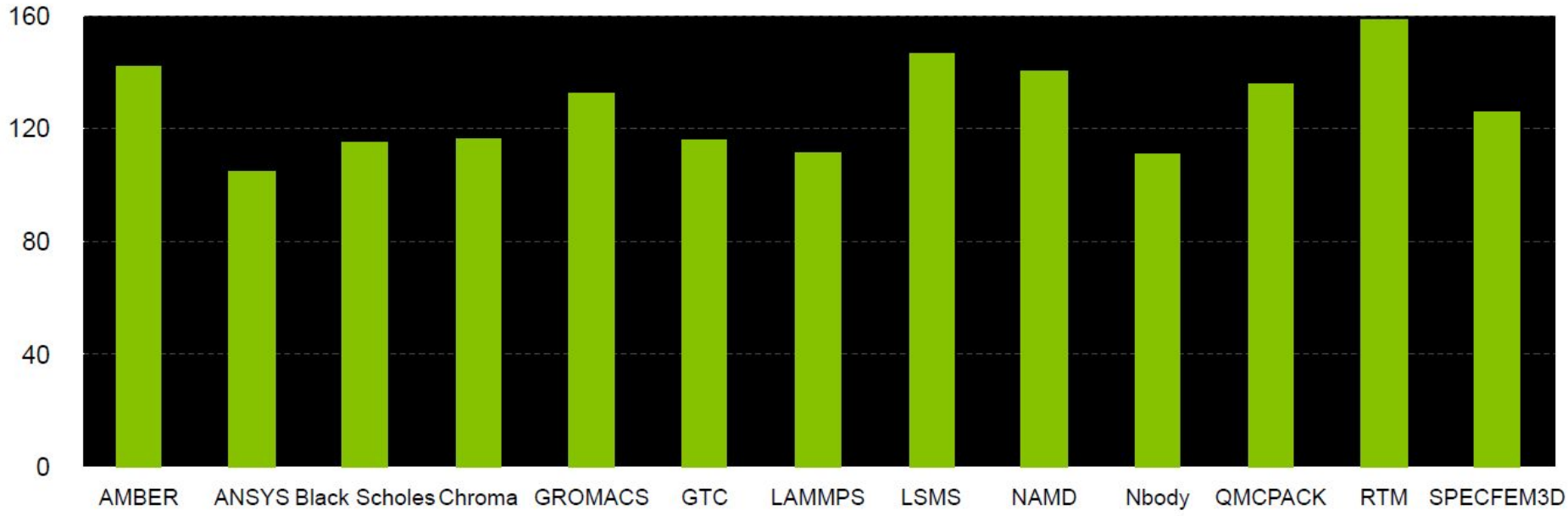
New Feature: GPU Boost

- Allows to speed-up the GPU clock up to 17% if the power required by an application is low.
- The base clock will be restored if we exceed 235 W.
- We can set up a persistent mode which keep values permanently, or another one for a single run.



GPU Boost
Why it is important?

- Here we see the average power (watts) on a Tesla K20X for a set of popular applications within the HPC field:



GPU Boost
How it is organized?

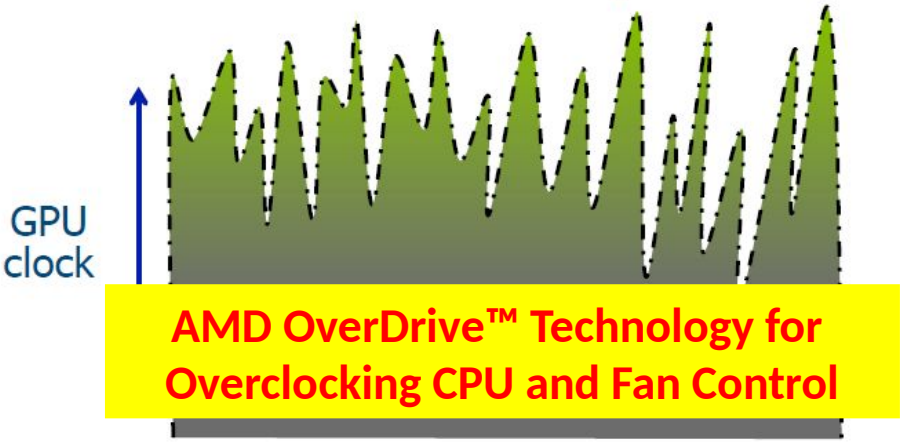
- For the Tesla K40 case, 3 clocks are defined, 8.7% apart.

Up to 40% higher performance relative to Tesla K20X.

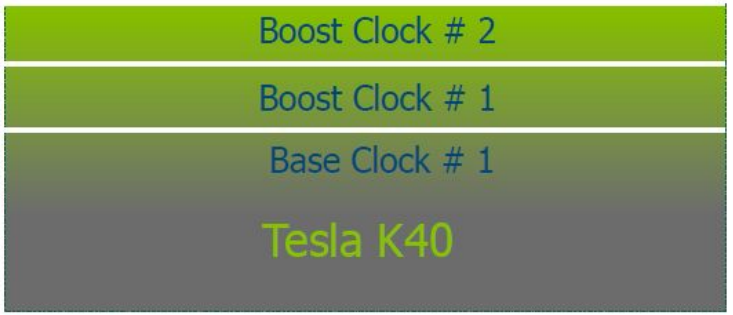
And not only GFLOPS are improved, but also effective memory bandwidth.

Comparison with Competitors

- It is better a stationary state for the frequency to avoid thermal stress and improve reliability.



Automatic clock switching



Deterministic Clocks

	Other vendors	Tesla K40
Default	Boost	Base
Preset options	Lock to base clock	3 levels: Base, Boost1 o Boost2
Boost interface	Control panel	Shell command: `nv-smi`
Target duration for boosts	Roughly 50% of run-time	100% of workload run time

Commands to Control Boost

Command	Effect
nvidia-smi -q -d SUPPORTED_CLOCKS	**View the clocks supported by our GPU**
nvidia-smi -ac <MEM clock, Graphics clock>	**Set one of the supported clocks**
nvidia-smi -pm 1	Enables persistent mode: The clock settings are preserved after restarting the system or driver
nvidia-smi -pm 0	Enables non-persistent mode: Clock settings revert to base clocks after restarting the system or driver
nvidia-smi -q -d CLOCK	**Query the clock in use**
nvidia-smi -rac	Reset clocks back to the base clock
nvidia-smi -acp 0	Allow non-root users to change clock rates

New Advance: GPU Boost - Example

- `nvidia-smi -q -d CLOCK –id=0000:86:00.0`

```
==============NVSMI LOG==============

Timestamp                           : Wed Jan 29 13:35:58 2014
Driver Version                      : 319.37

Attached GPUs                       : 5
GPU 0000:86:00.0
    Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 3004 MHz
    Applications Clocks
        Graphics                    : 875 MHz
        Memory                      : 3004 MHz
    Default Applications Clocks
        Graphics                    : 745 MHz
        Memory                      : 3004 MHz
    Max Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 3004 MHz
```

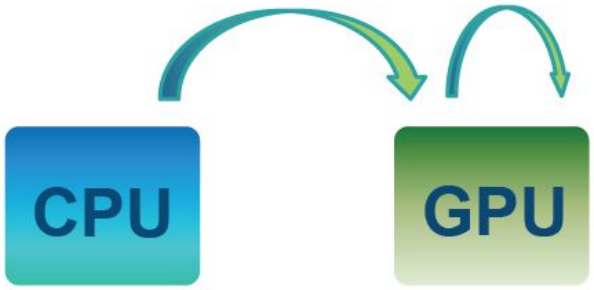
Kepler –>

New Feature: Dynamic Parallelism

What is Dynamic Parallelism?

- The ability to launch new grids from the GPU:
  - Dynamically: Based on run-time data.
  - Simultaneously: From multiple threads at once.
  - Independently: Each thread can launch a different grid.

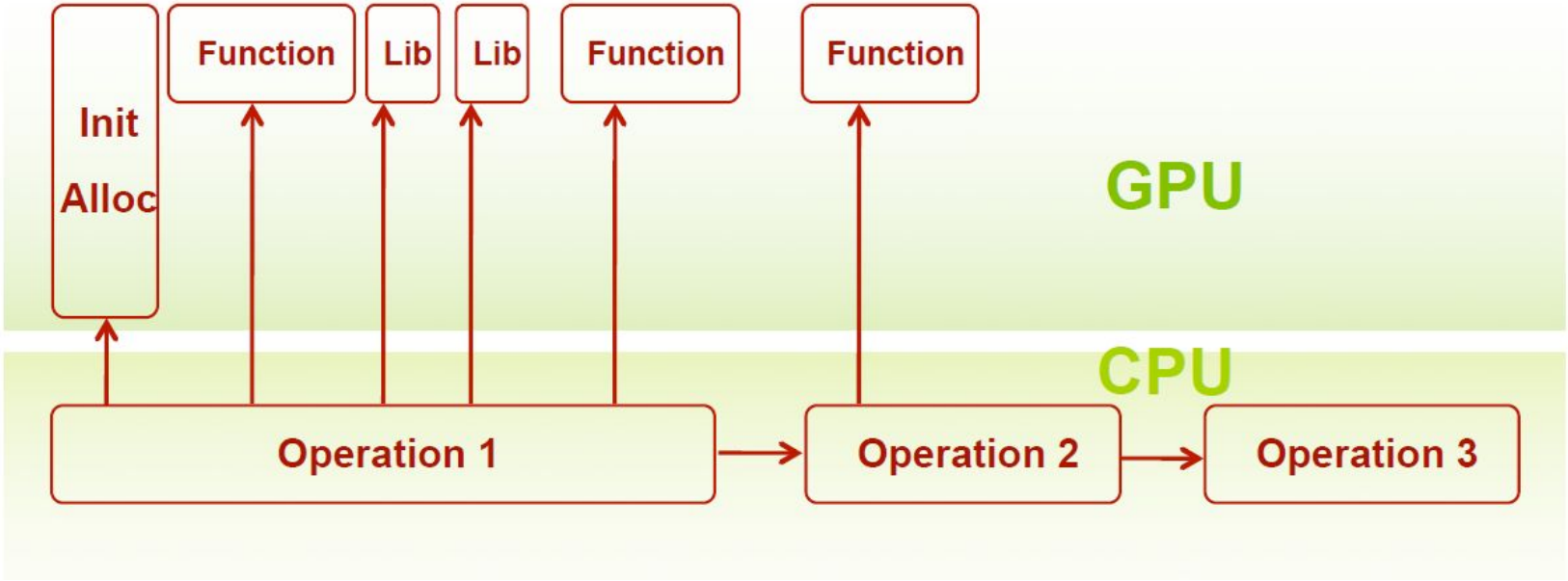
Fermi: Only CPU
can generate GPU work.



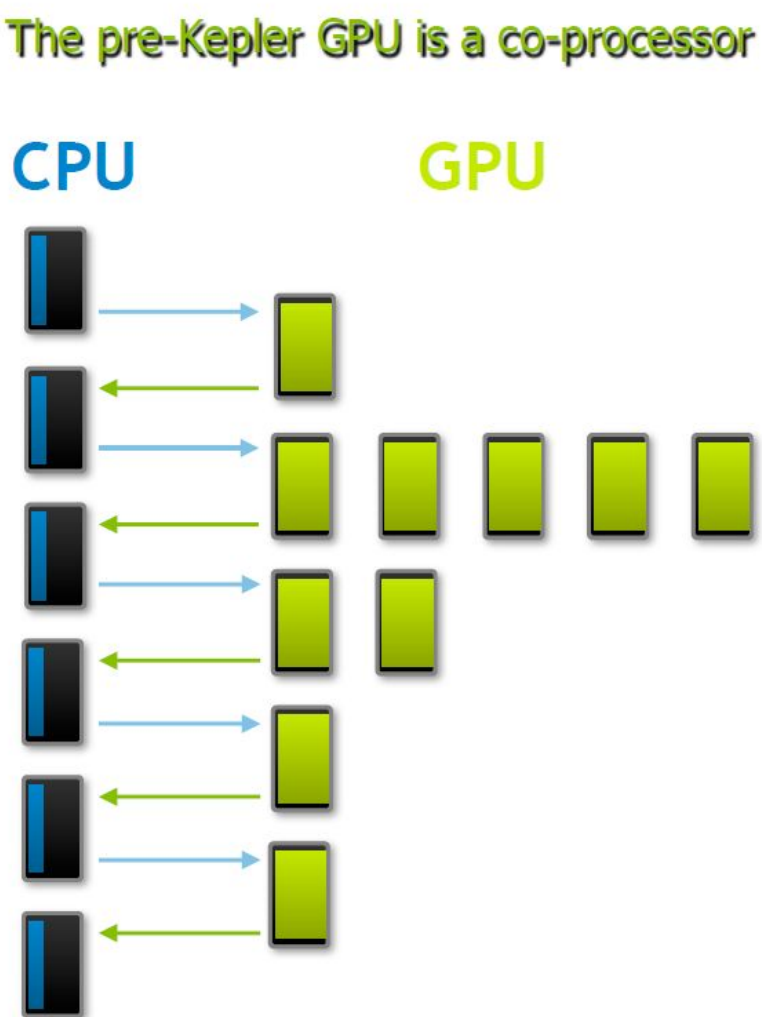
Kepler: GPU can generate work for itself.

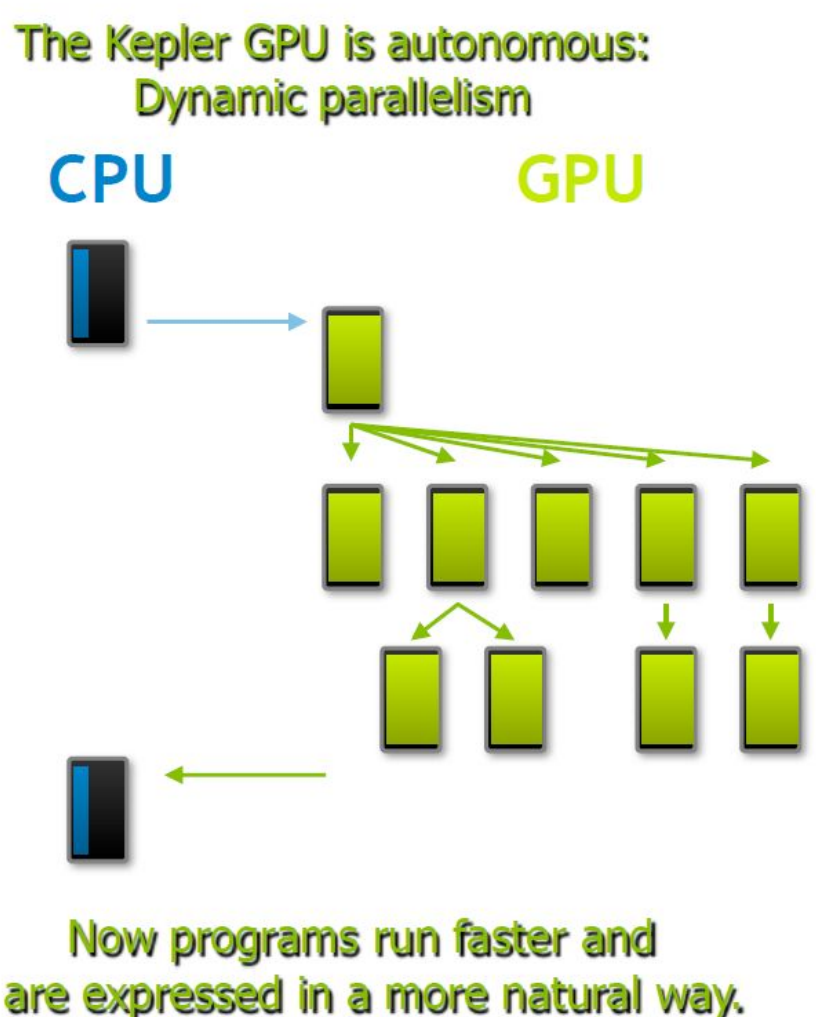
How GPU Worked before Kepler?

- High data bandwidth for communications:
  - External: More than 10 GB/s (PCI-express 3).
  - Internal: More than 100 GB/s (GDDR5 video memory and 384 bits, which is like a six channel CPU architecture).

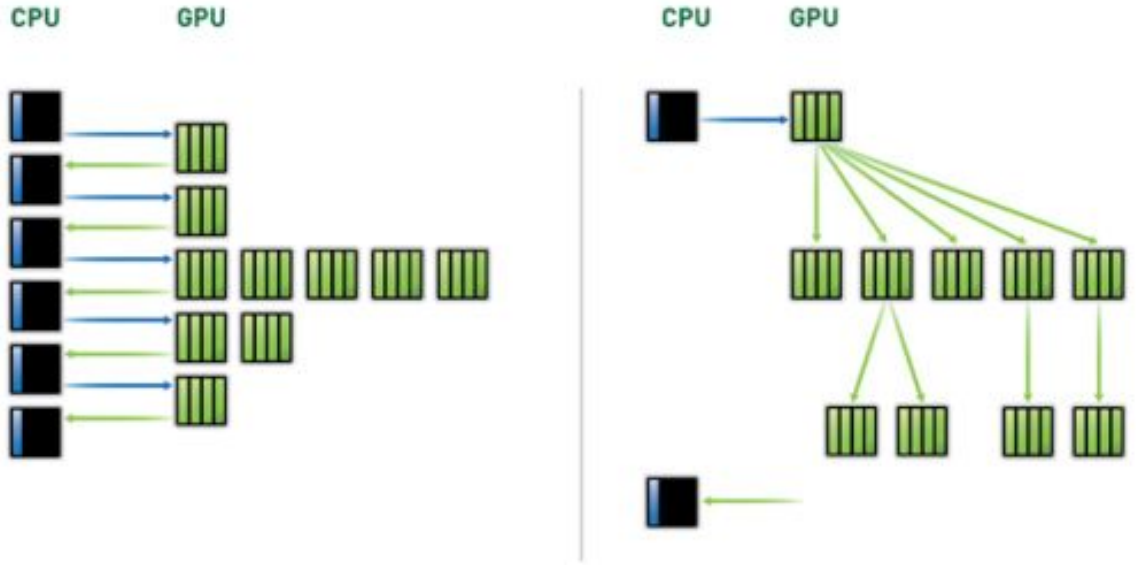


Difference in Parallelization:
Fermi versus Kepler





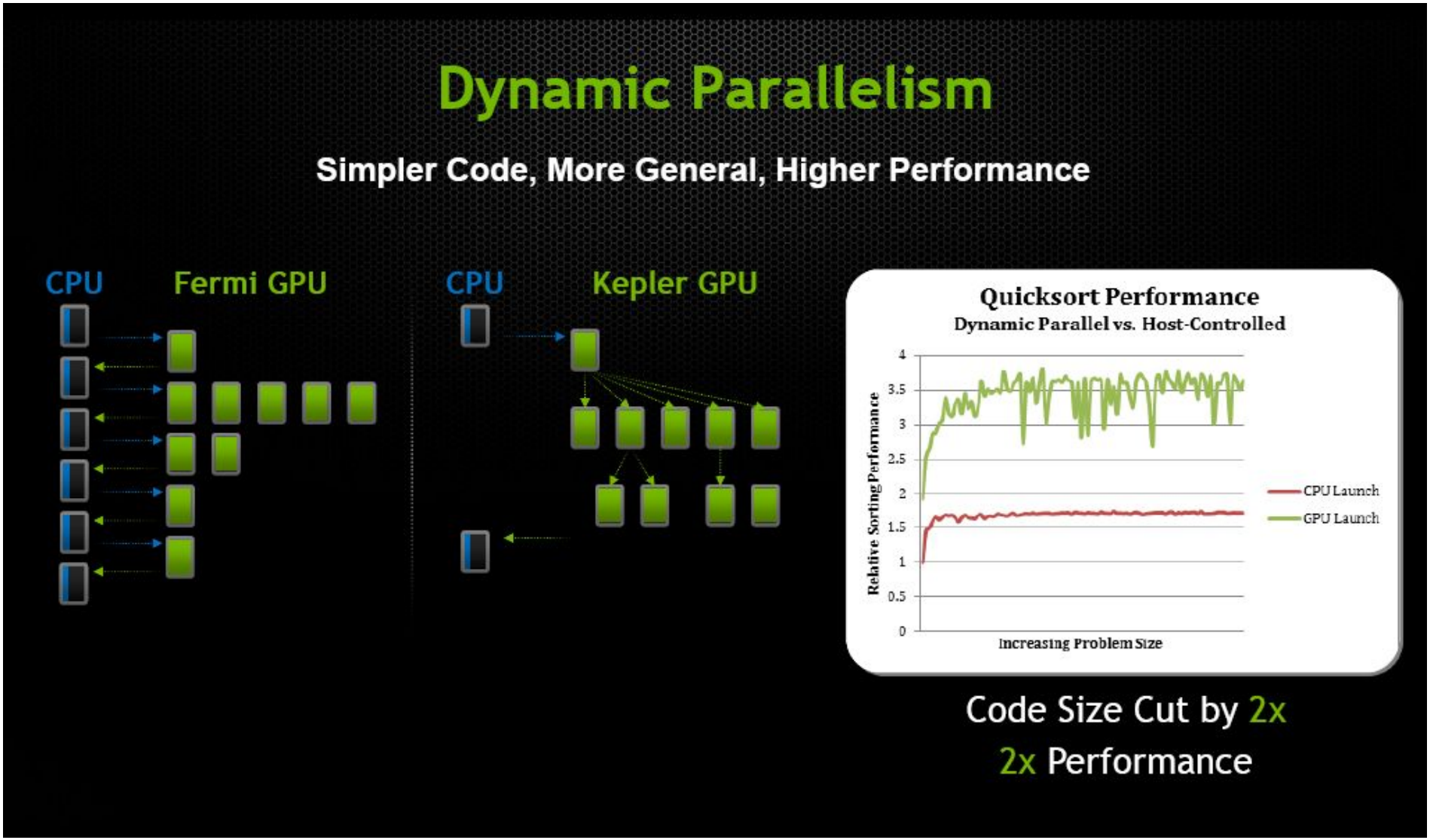
Dynamic Parallelism: How it Works?



Fermi: data transits CPU->GPU across the PCIe bus, a single CUDA kernel executes on the GPU, and then data returns across the bus back to the CPU. This round-robin data traffic occurs for each and every kernel launch. Unfortunately the PCIe bus is slow, much slower than CPU and GPU speeds, and so we try to avoid CPU->GPU data transfers whenever possible. **PCIe bus can become a severe bottleneck for many types of algorithms.**

Kepler: data transits CPU->GPU through the PCIe bus, **but then with Dynamic Parallelism one kernel can spawn one to several additional kernels without the need for data transfers back to the CPU**. All of the kernels and their associated datasets remain on the GPU, thus avoiding the need for PCIe bus traffic. When some required synchronization point is reached, the GPU kernels finally terminate and data transfers to the CPU take place.

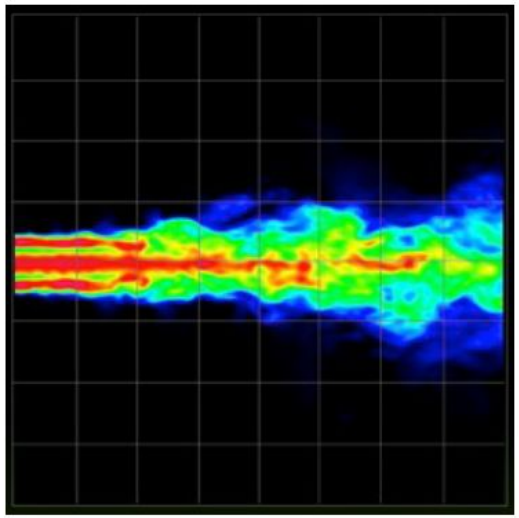
Dynamic Parallelism - Speedup



Why Dynamic Parallelism is Important?
Example 1: Finite Element Method

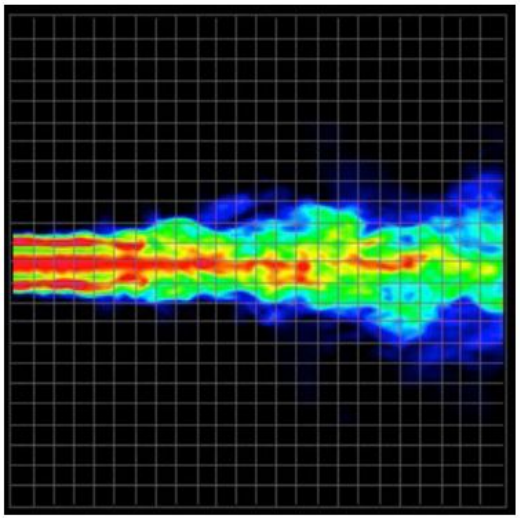
- Assign resources dynamically according to real-time demand, making easier the computation of irregular problems on GPU.
- It broadens the application scope where it can be useful.

Coarse grid



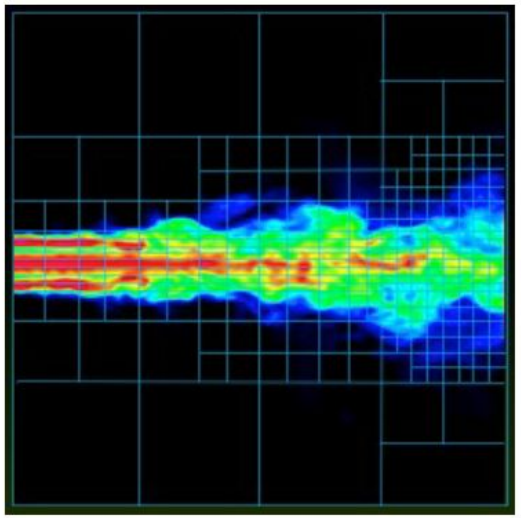
Higher performance, lower accuracy

Fine grid



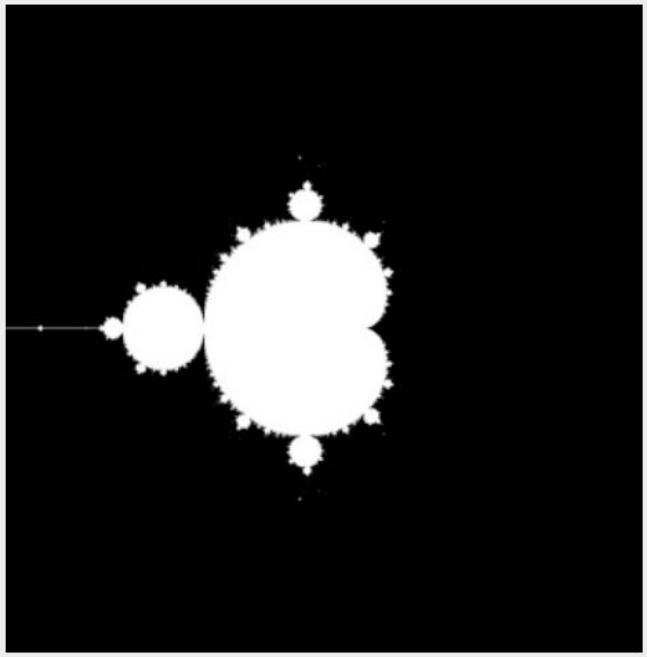
Lower performance, higher accuracy

Dynamic grid

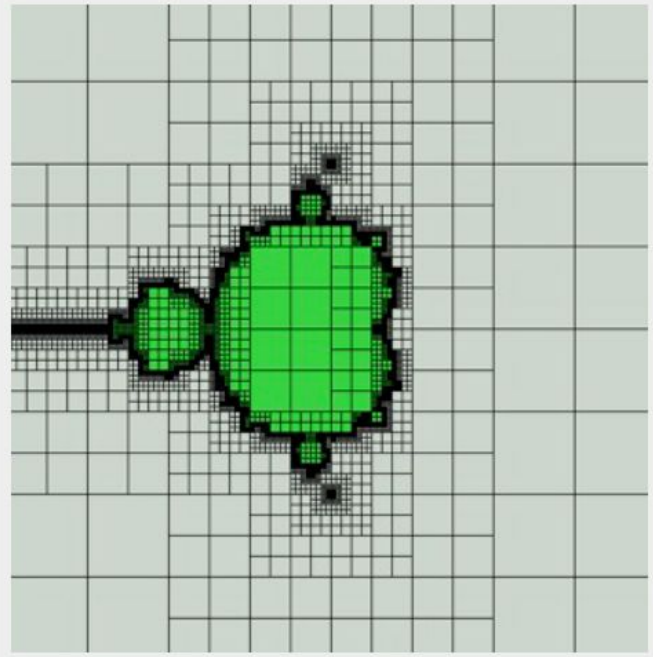


Target performance where accuracy is required

Why Dynamic Parallelism is Important?
Example 2: Fractal Simulation



Computational power allocated to regions of interest



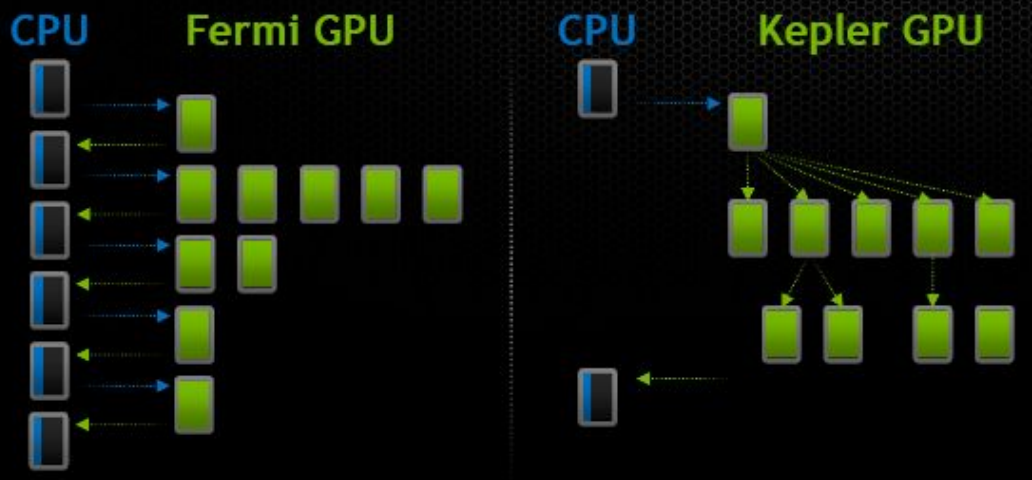
CUDA until 2012:
- The CPU launches kernels regularly.
- All pixels are treated the same.

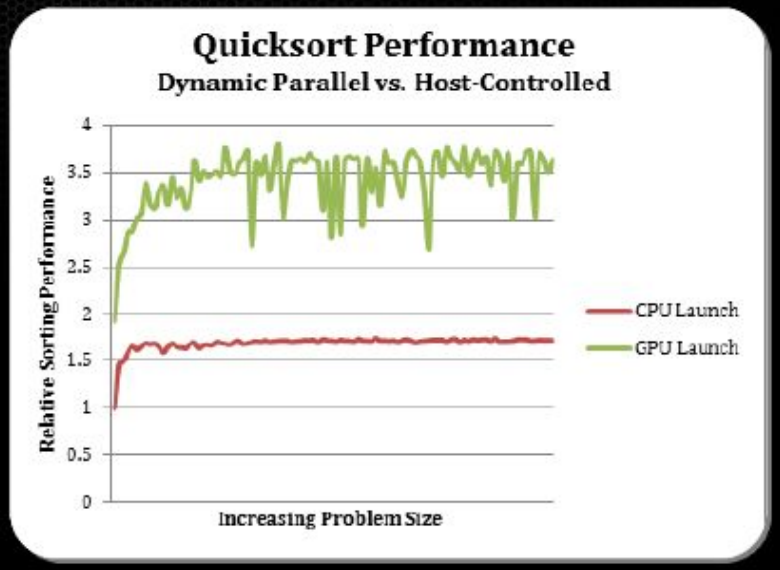
CUDA on Kepler:
- The GPU launches a different number of kernels/blocks for each computational region.

Dynamic Parallelism: Speedup

Dynamic Parallelism

Simpler Code, More General, Higher Performance





Code Size Cut by 2x

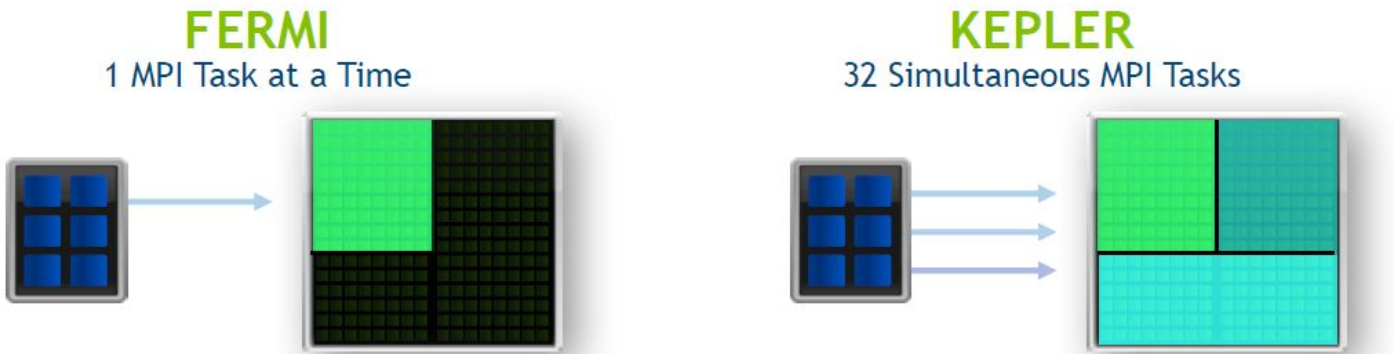
2x Performance

Kepler –>

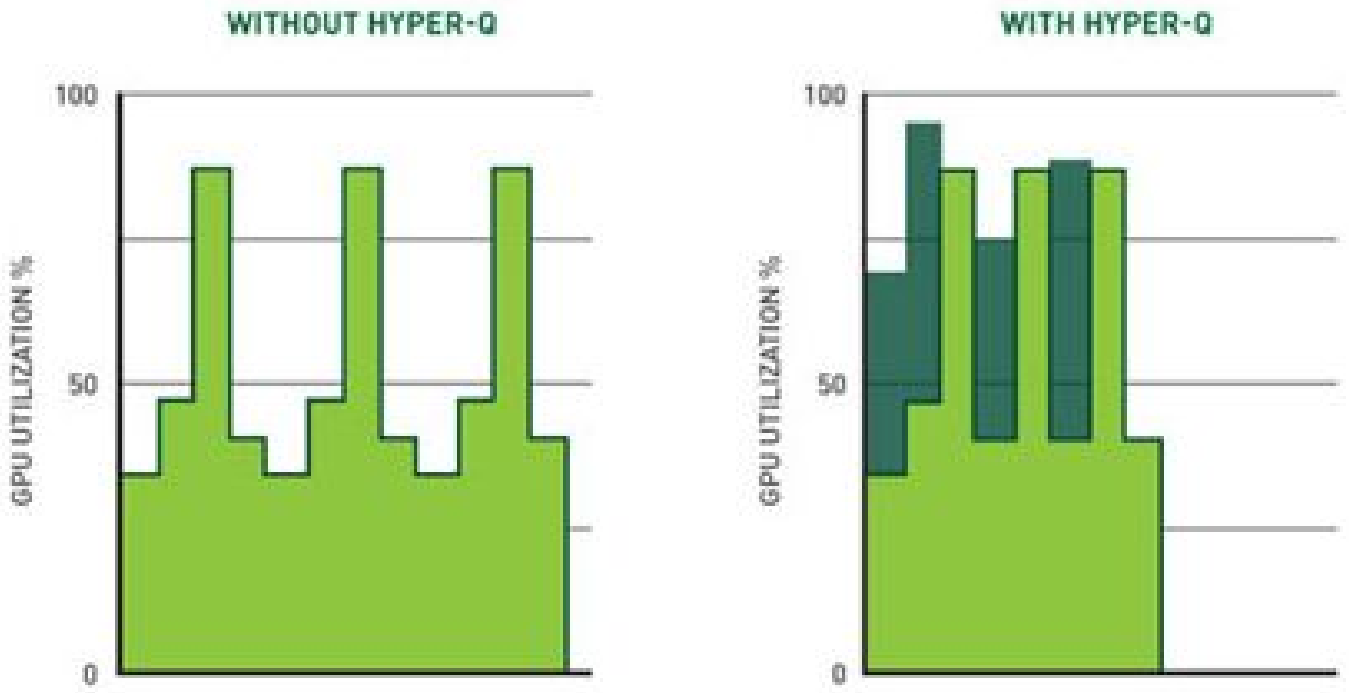
New Feature: Hyper-Q

Hyper-Q: What is it?

- In Fermi, several CPU processes can send thread blocks to the same GPU, but a kernel cannot start its execution until the previous one has finished.
- In Kepler, we can execute simultaneously up to 32 kernels launched from different:
  - MPI processes, CPU threads (POSIX threads) or CUDA streams.
- This increments the % of temporal occupancy on the GPU.



Hyper-Q: Higher GPU Utilization



Hyper-Q thus increases the utilization and efficiency of GPU workloads, which in turn decreases CPU idle time.
For well-designed algorithms Hyper-Q could result in a **32X increase** in software application performance.

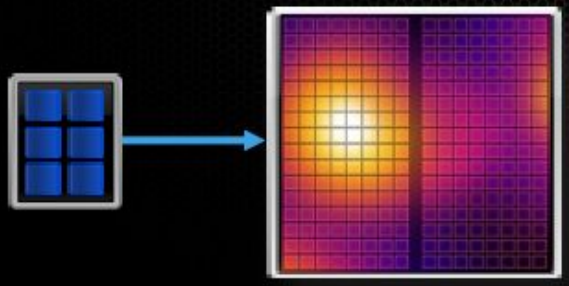
Hyper-Q: Speedup

Hyper-Q

Easier Speedup of Legacy MPI Apps

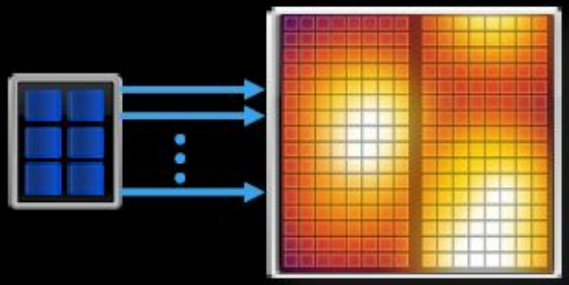
FERMI

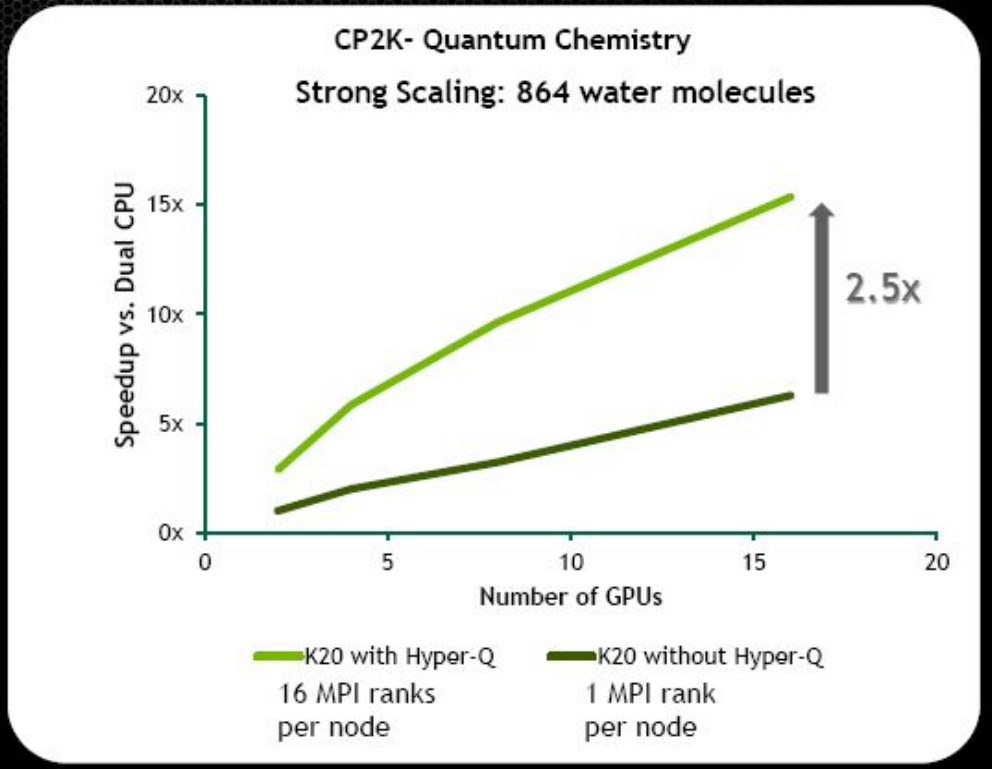
1 Work Queue



KEPLER

32 Concurrent Work Queues





Hyper-Q: How it Works?

```
__global__ kernel_A(pars) {body} // Same for B...Z
cudaStream_t stream_1, stream_2, stream_3;
...
cudaStreamCreatewithFlags(&stream_1, ...);
cudaStreamCreatewithFlags(&stream_2, ...);
cudaStreamCreatewithFlags(&stream_3, ...);
...
kernel_A <<< dimgridA, dimblockA, 0, stream_1 >>> (pars);
kernel_B <<< dimgridB, dimblockB, 0, stream_1 >>> (pars);
kernel_C <<< dimgridC, dimblockC, 0, stream_1 >>> (pars);
...
kernel_P <<< dimgridP, dimblockP, 0, stream_2 >>> (pars);
kernel_Q <<< dimgridQ, dimblockQ, 0, stream_2 >>> (pars);
kernel_R <<< dimgridR, dimblockR, 0, stream_2 >>> (pars);
...
kernel_X <<< dimgridX, dimblockX, 0, stream_3 >>> (pars);
kernel_Y <<< dimgridY, dimblockY, 0, stream_3 >>> (pars);
kernel_Z <<< dimgridZ, dimblockZ, 0, stream_3 >>> (pars);
```

stream 1 ↓

stream 2 ↓

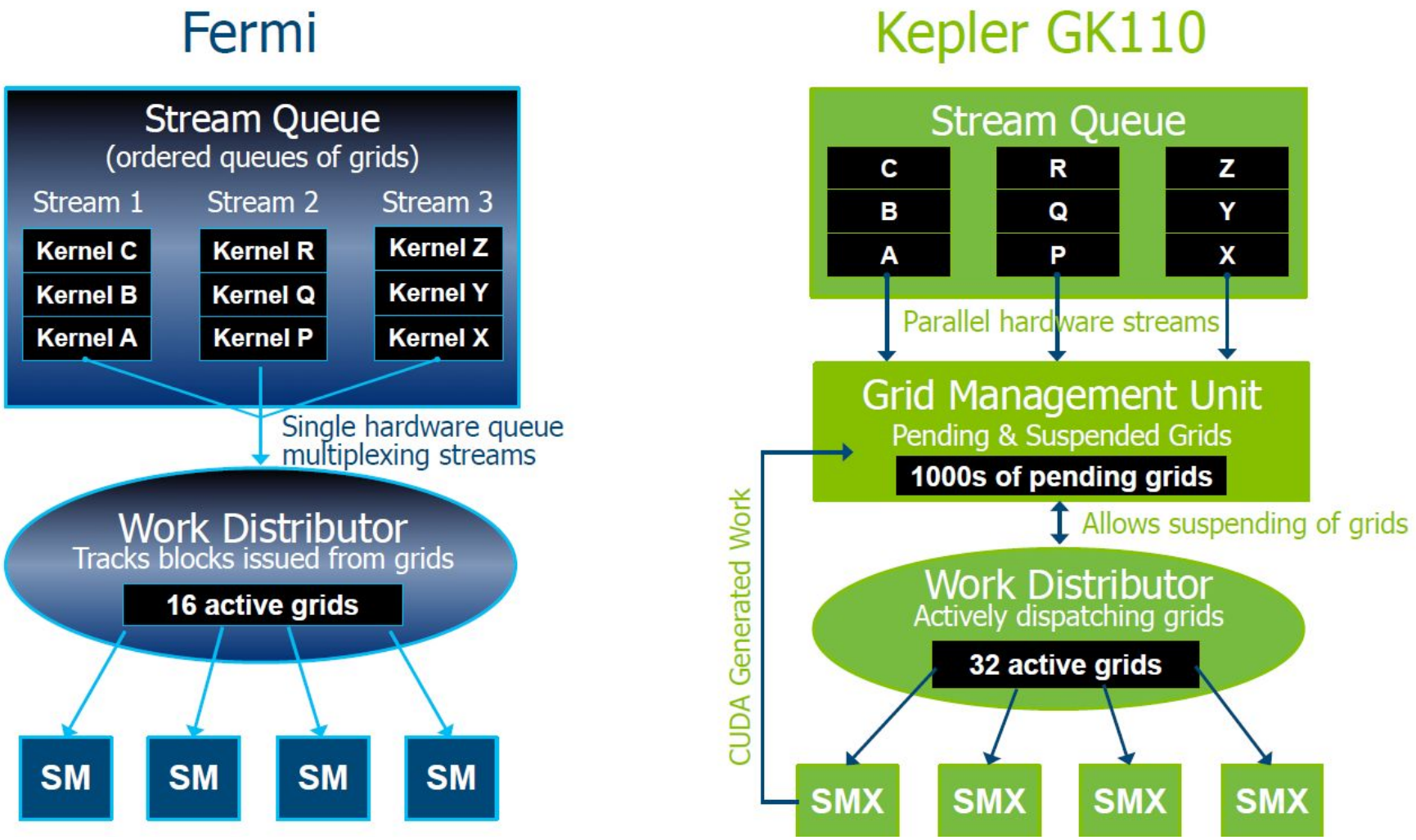
stream 3 ↓

stream_1
kernel_A
kernel_B
kernel_C

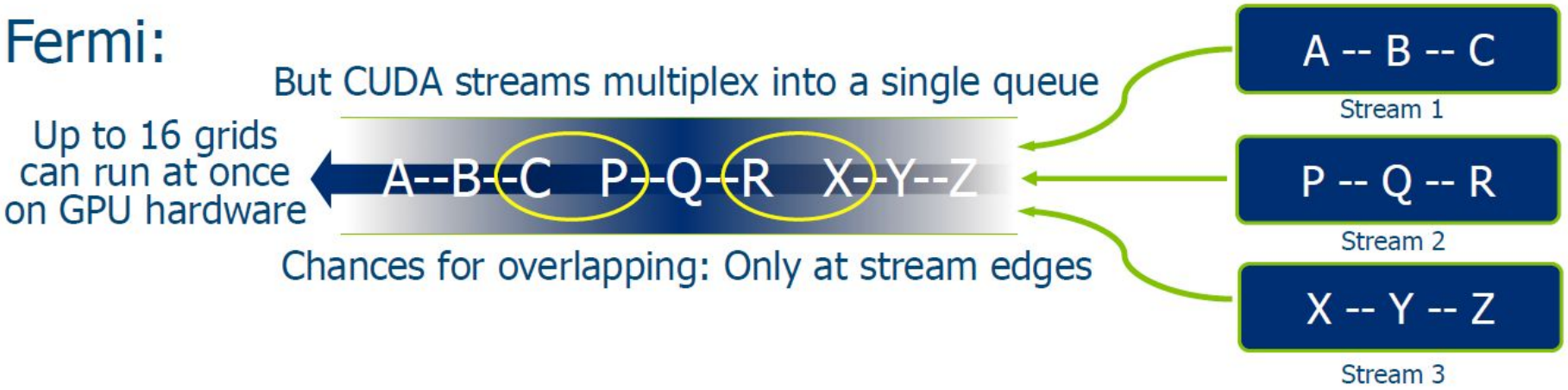
stream_2
kernel_P
kernel_Q
kernel_R

stream_3
kernel_X
kernel_Y
kernel_Z

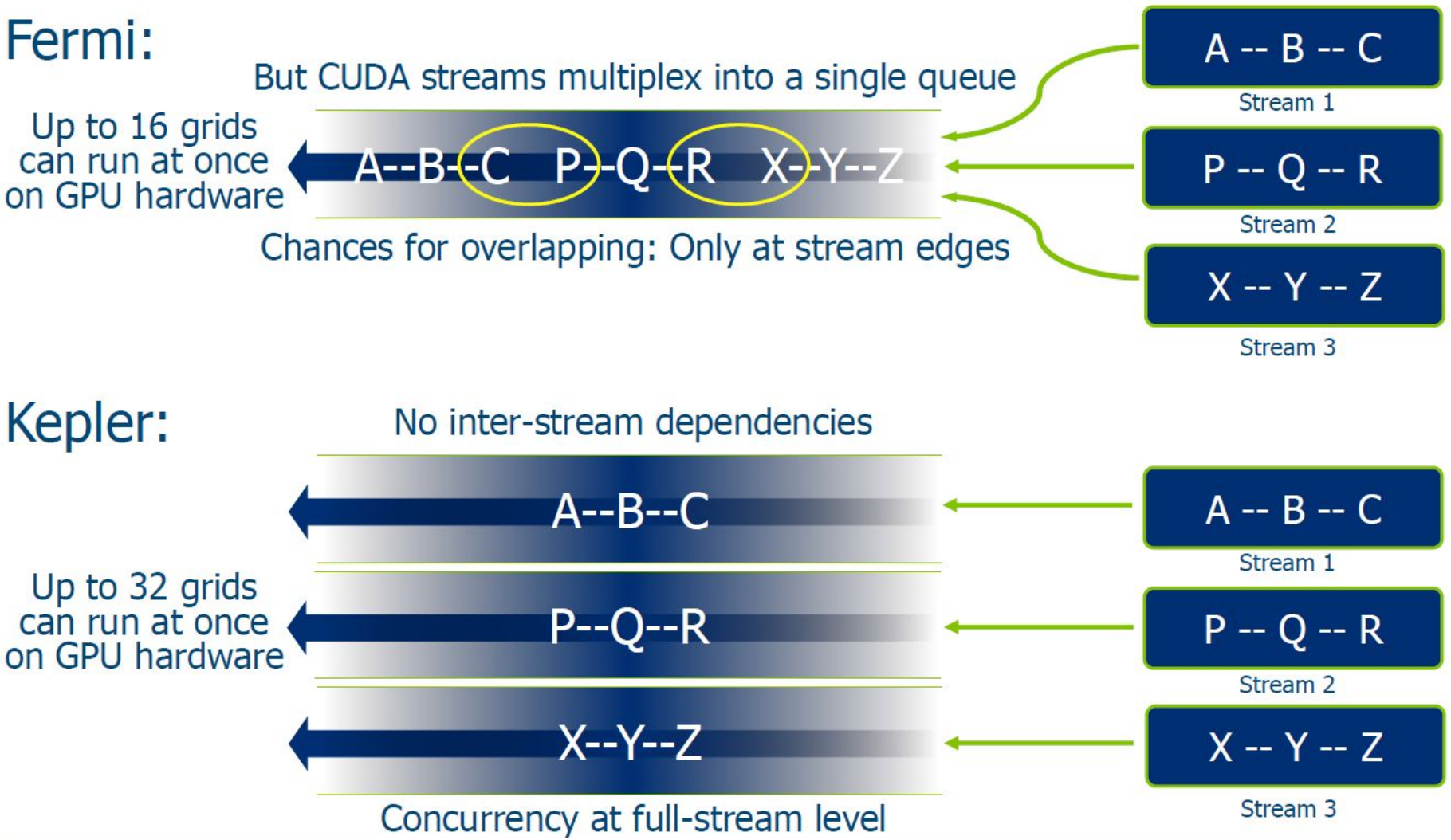
Hyper-Q: How it Works?



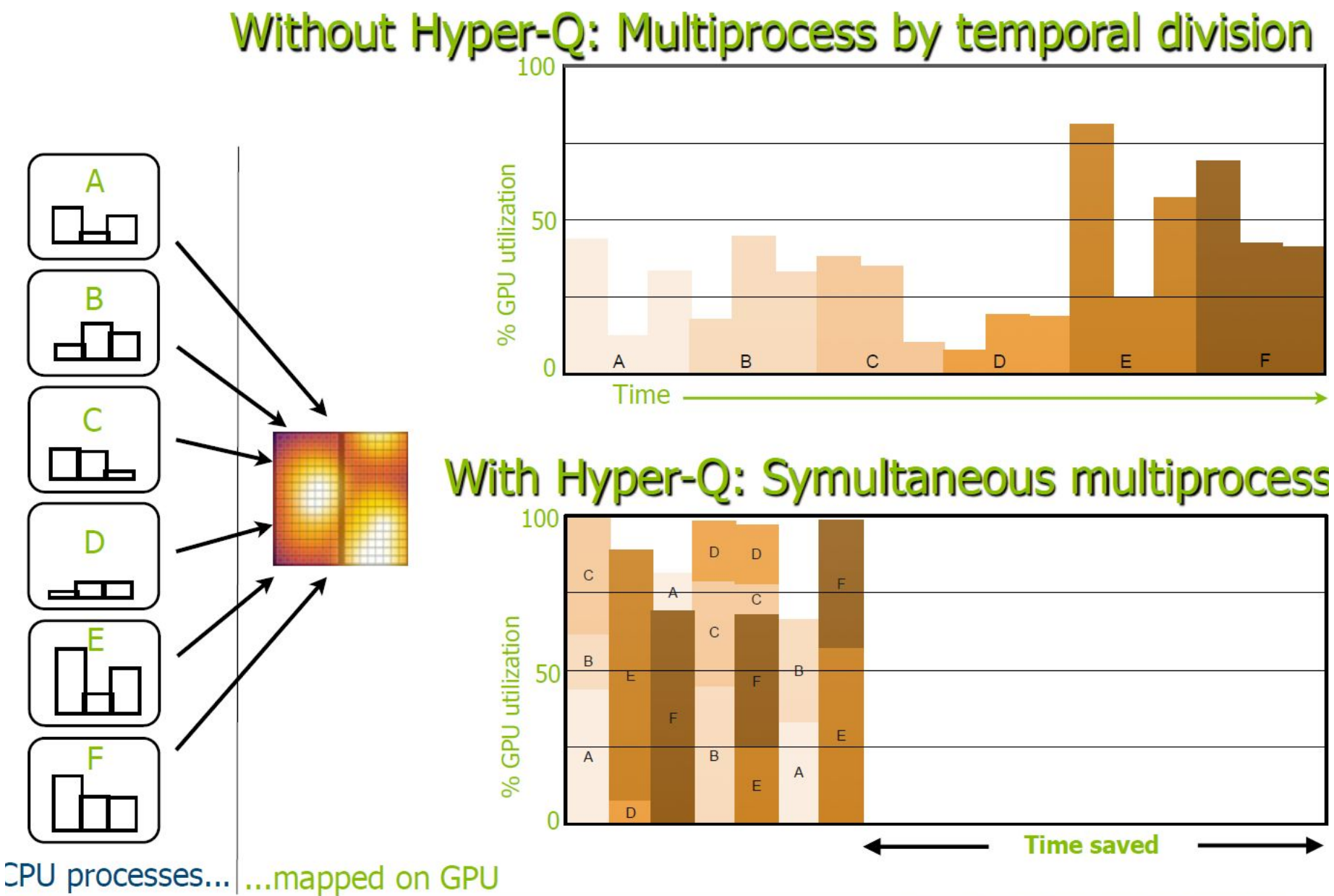
Hyper-Q:
Software and Hardware Queues



Hyper-Q: Software and Hardware Queues

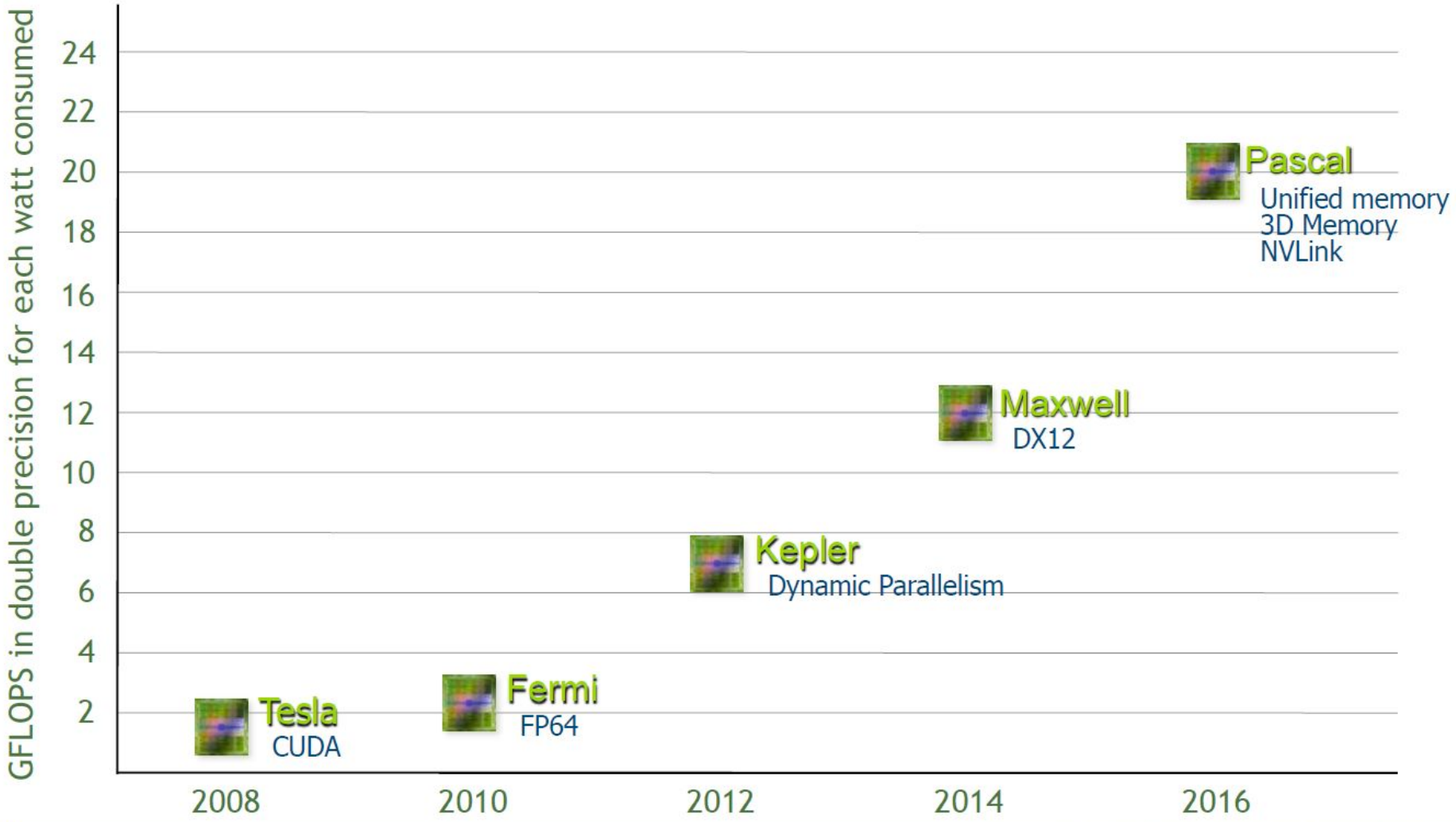


Hyper-Q: How it Works?

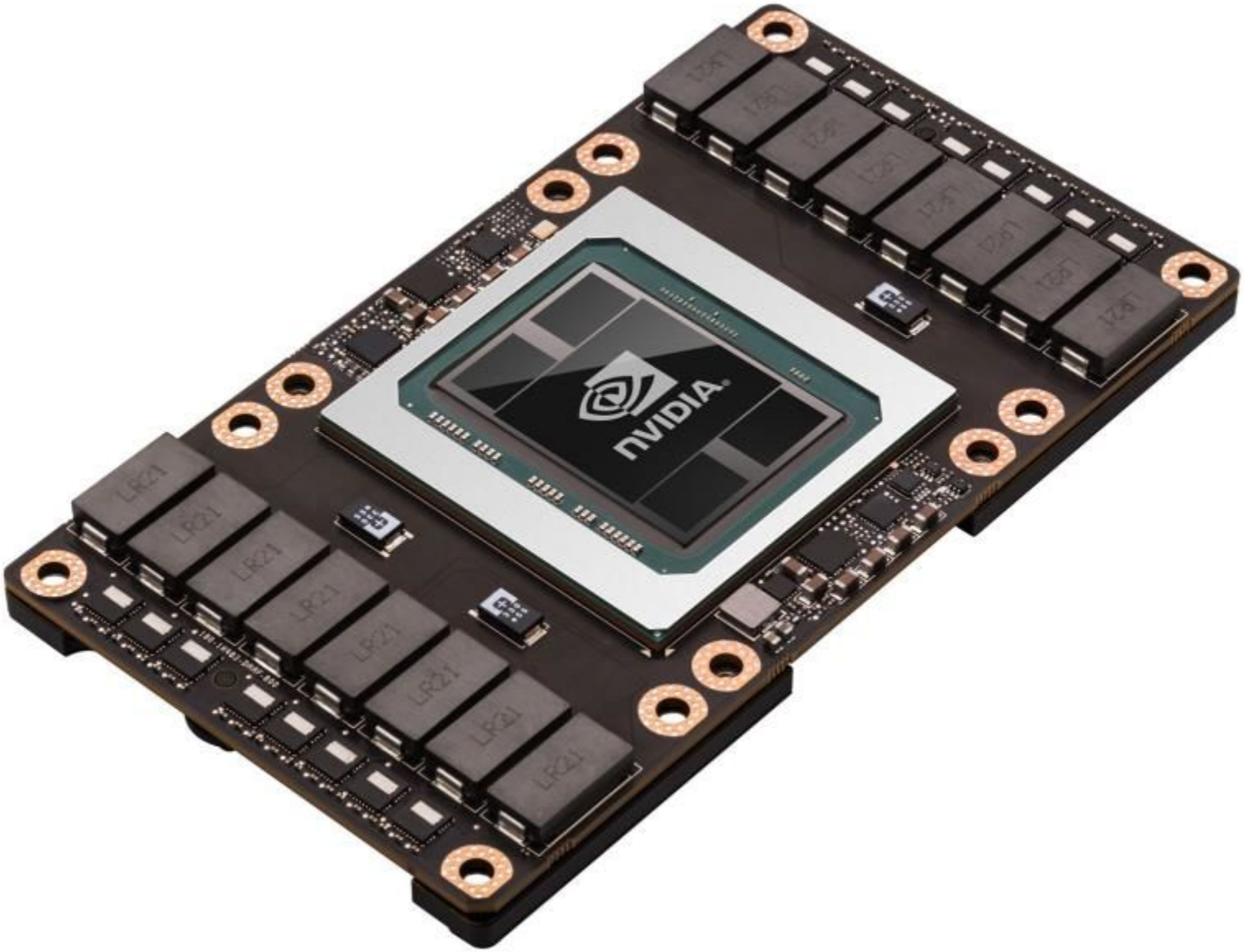


From Kepler –>
to New Generations:
Maxwell,
Pascal,
…

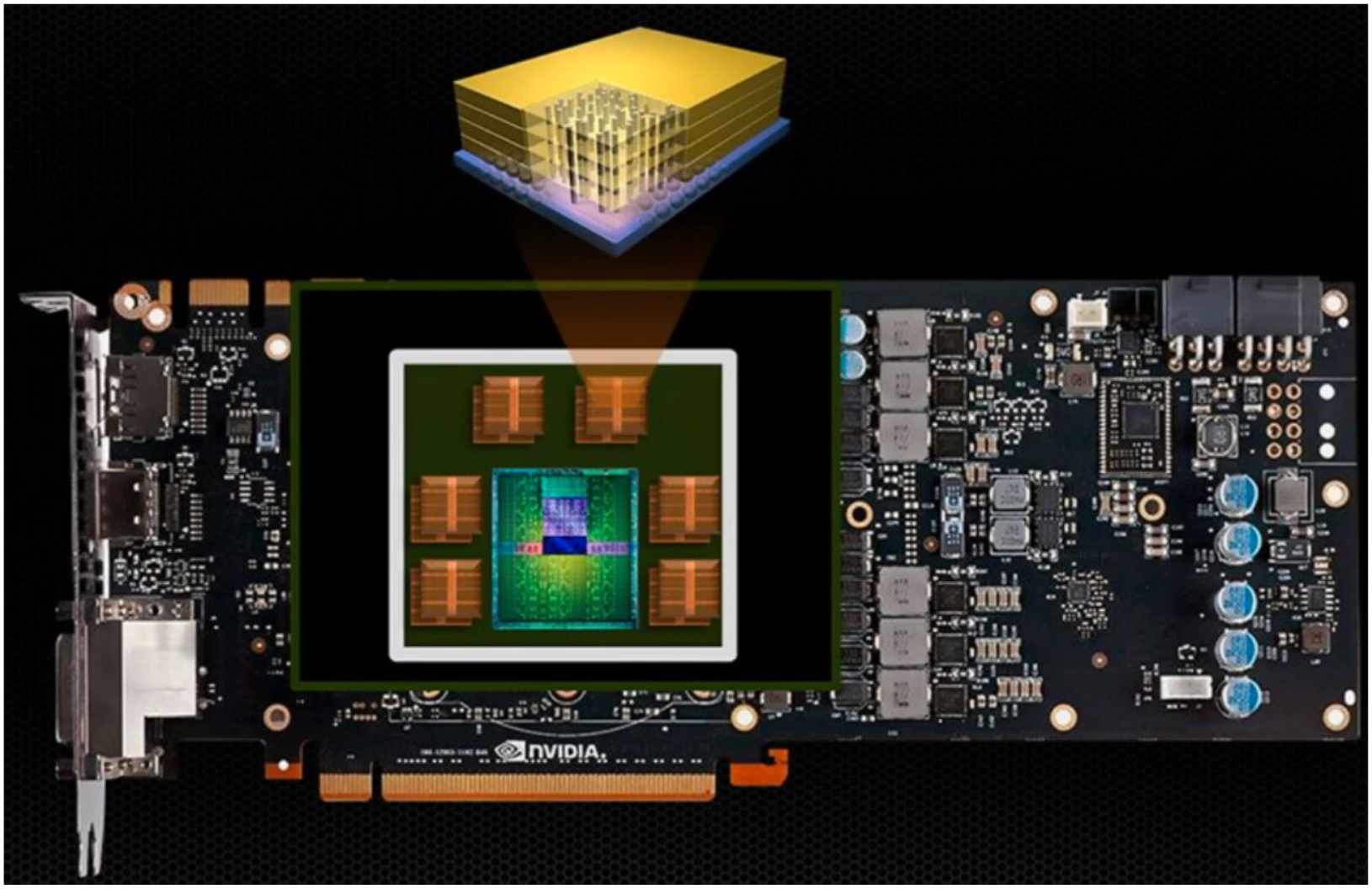
… to New Generations: Maxwell, Pascal, …



Pascal: What is New in it?

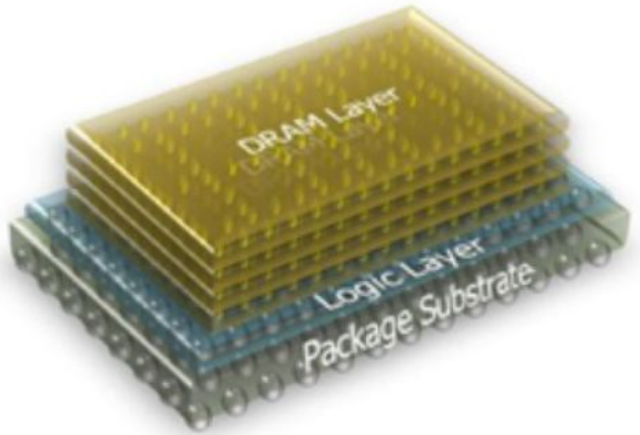


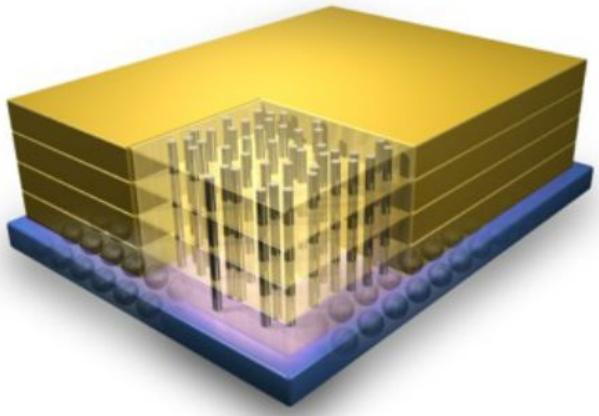
Pascal: to Stacked (3D) RAM …



Pascal: to Stacked (3D) RAM …

- DRAM cells are organized in **vaults**, which take borrowed the interleaved memory arrays from already existing DRAM chips.
- A logic controller is placed at the base of the DRAM **layers**, with data matrices on top.
- The assembly is connected with through-silicon vias, **TSVs**, which traverse vertically the stack using pitches between 4 and 50 um.
  - For a pitch of 10 um., a 1024-bit bus (16 memory channels) requires a **die size** of 0.32 mm2, which barely represents 0.2% of a CPU die (160 mm2).
  - Vertical **latency** to traverse the height of a Stacked DRAM endowed with 20 layers is only **12 picosecs**.
- The final step is advanced package assembly of vaults, layers and TSVs. This prevents parasitic capacitances which reduce signal speed and increase power required to switch.





Pascal: to Stacked (3D) RAM …

- On a CPU system (PC with a 4-channel motherboard, 256 bits):
  - [2013] DDR3 @ 4 GHz (2x 2000 MHz): 128 Gbytes/s.
  - [2014] A CPU with HMC 1.0 (first generation): 320 Gbytes/s. on each dir.
  - [2015] A CPU with HMC 2.0 (second generation): 448 Gbytes/s.
- On a GPU system (384-bits wide graphics card):
  - [2013] A GPU with GDDR5 @ 7 GHz (2x 3500 MHz): 336 Gbytes/s.
  - [2014] A GPU with 12 chips of 32 bits manuf. using near memory HMC 1.0 would reach **480 Gbytes/s.** (6 channels HMC 1.0 @ 80 GB/s. each).
  - [2015] A GPU using HMC 2.0 (112 GB/s.) would reach **672 Gbytes/s., which doubles the bandwidth with respect to the most advanced GDDR technology in 2013.**