New hardware features in Kepler, SMX and Tesla K40

GPGPU2: Advanced Methods for Computing with CUDA

Cape Town, April, 2014

Manuel Ujaldón

Computer Architecture Department. University of Malaga.

CUDA Fellow

NVIDIA

``... and if software people wants good machines, they must learn more about hardware to influence that way hardware designers ...''

David A. Patterson & John Hennessy

Organization and Computer Design

Mc-Graw-Hill (1995)

Chapter 9, page 569

Talk outline [63 slides]

1. Introducing the architecture [4 slides]
2. The memory [3]
3. The SMX cores [9]
4. How the SMX works: Front-end and back-end [21]
5. Functional enhancements [11]
   1. Dynamic parallelism [5]
   2. Hyper-Q [6]
6. A look to the future [15]
   1. Vectorization: The warp size [7]
   2. Stacked-DRAM: 3D memory on top of the GPU [4]
   3. Analysis based on the roofline model [4]

1. Introducing the architecture

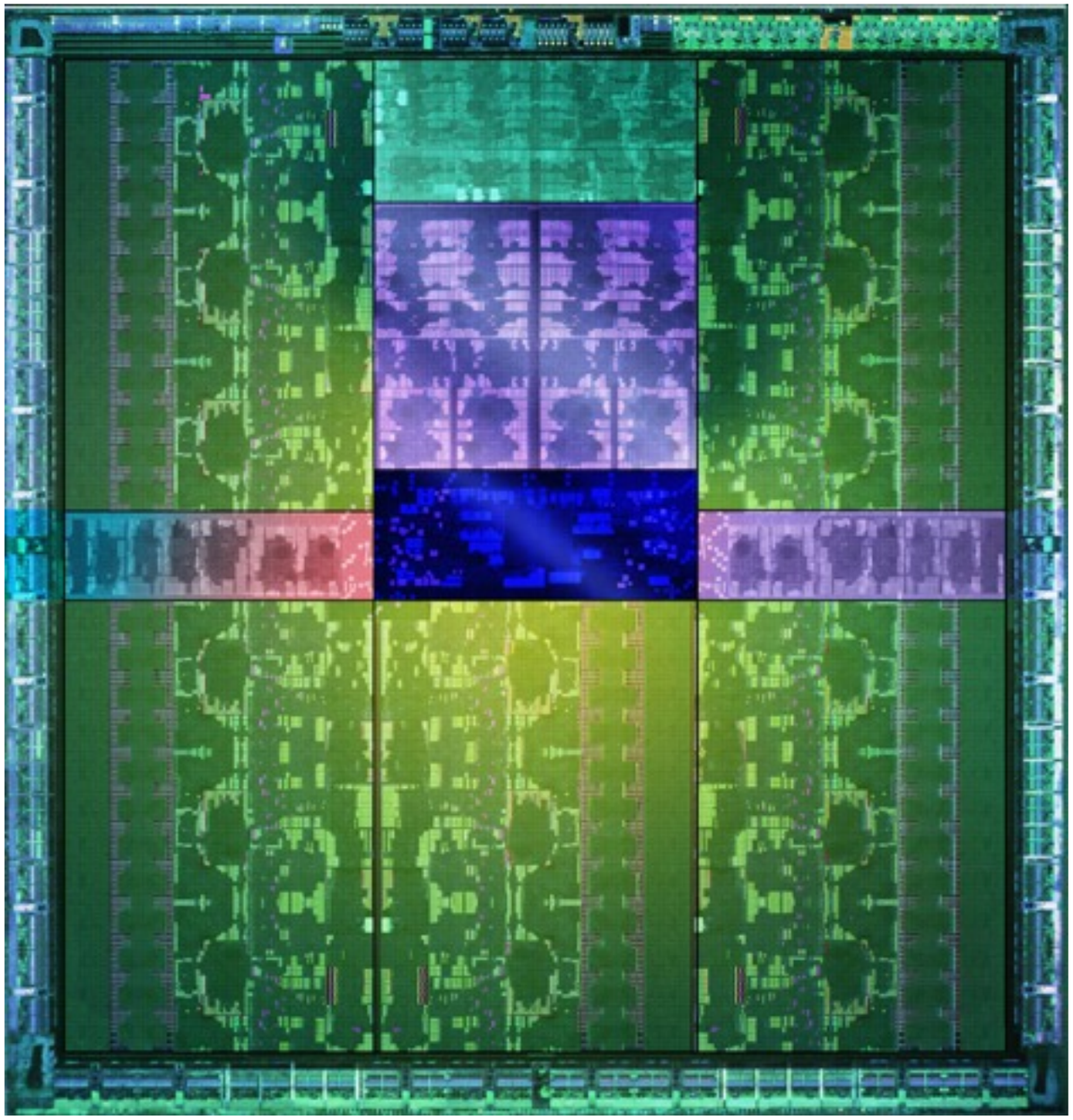
NVIDIA.

The three pillars of Kepler

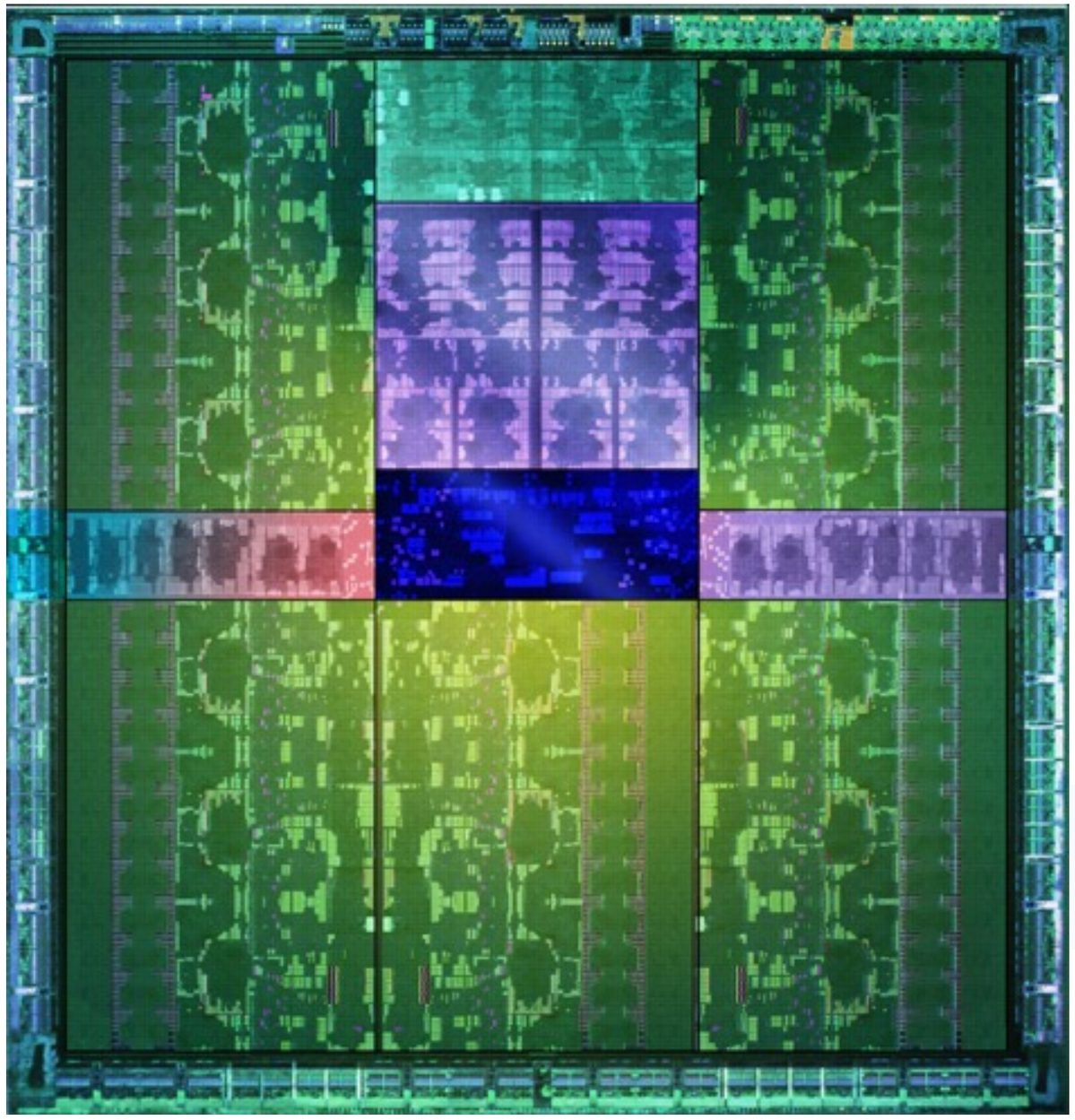
Power consumption

Performance

Programmability



And its three basic innovations



SMX: A multiprocessor with more resources and less power.

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

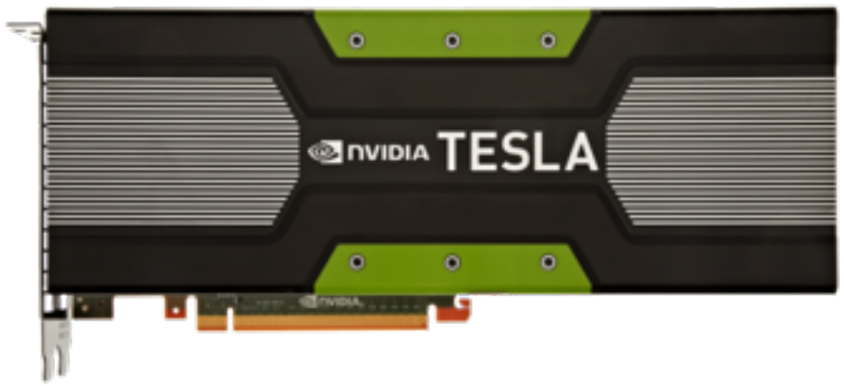
Hyper-Q: Multiple kernels can share the SMXs.

SMX Balance of Resources: Summary of improvements versus Fermi

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

Commercial models available for Kepler: GeForce vs. Tesla

- 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.

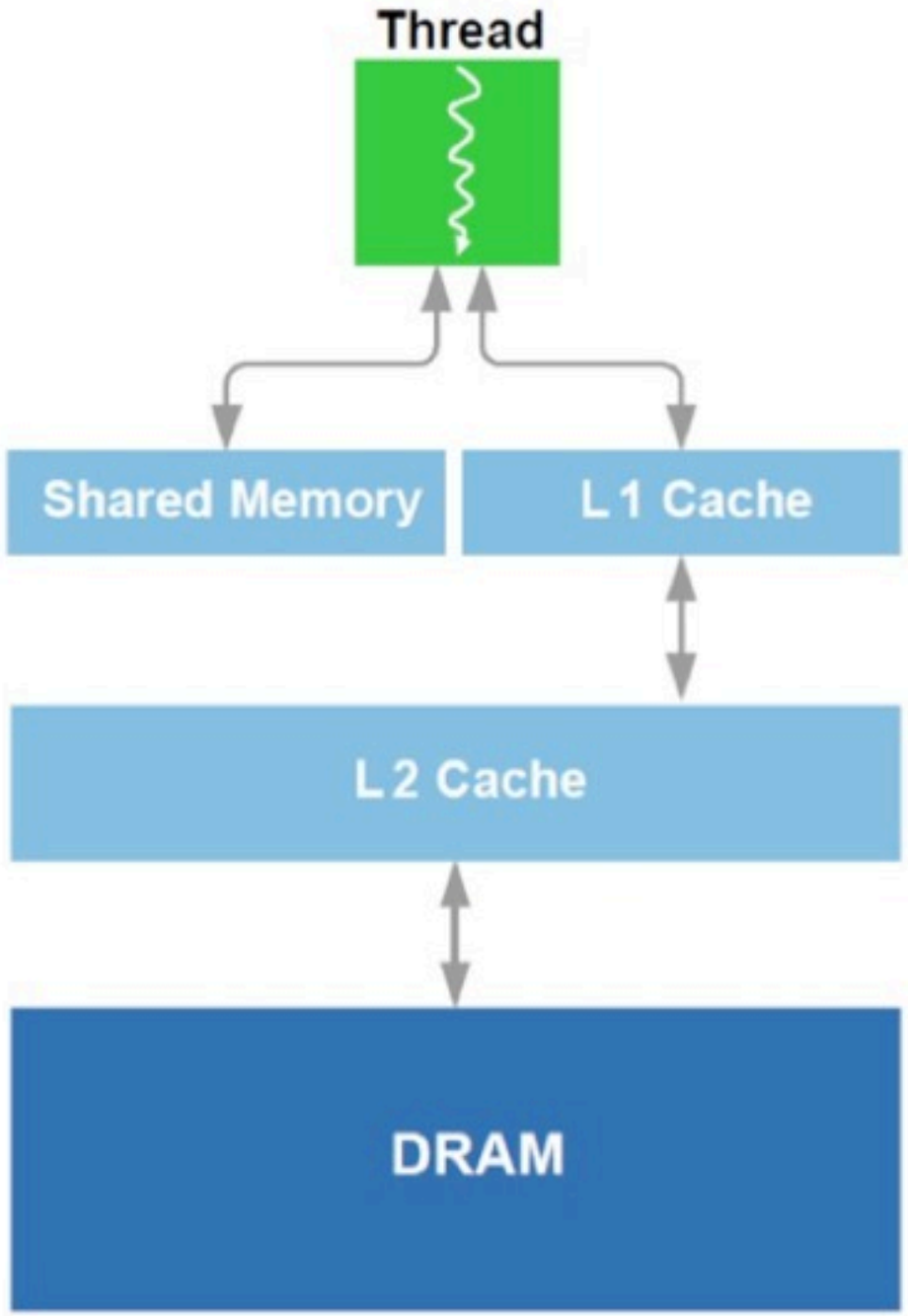
2. Memory

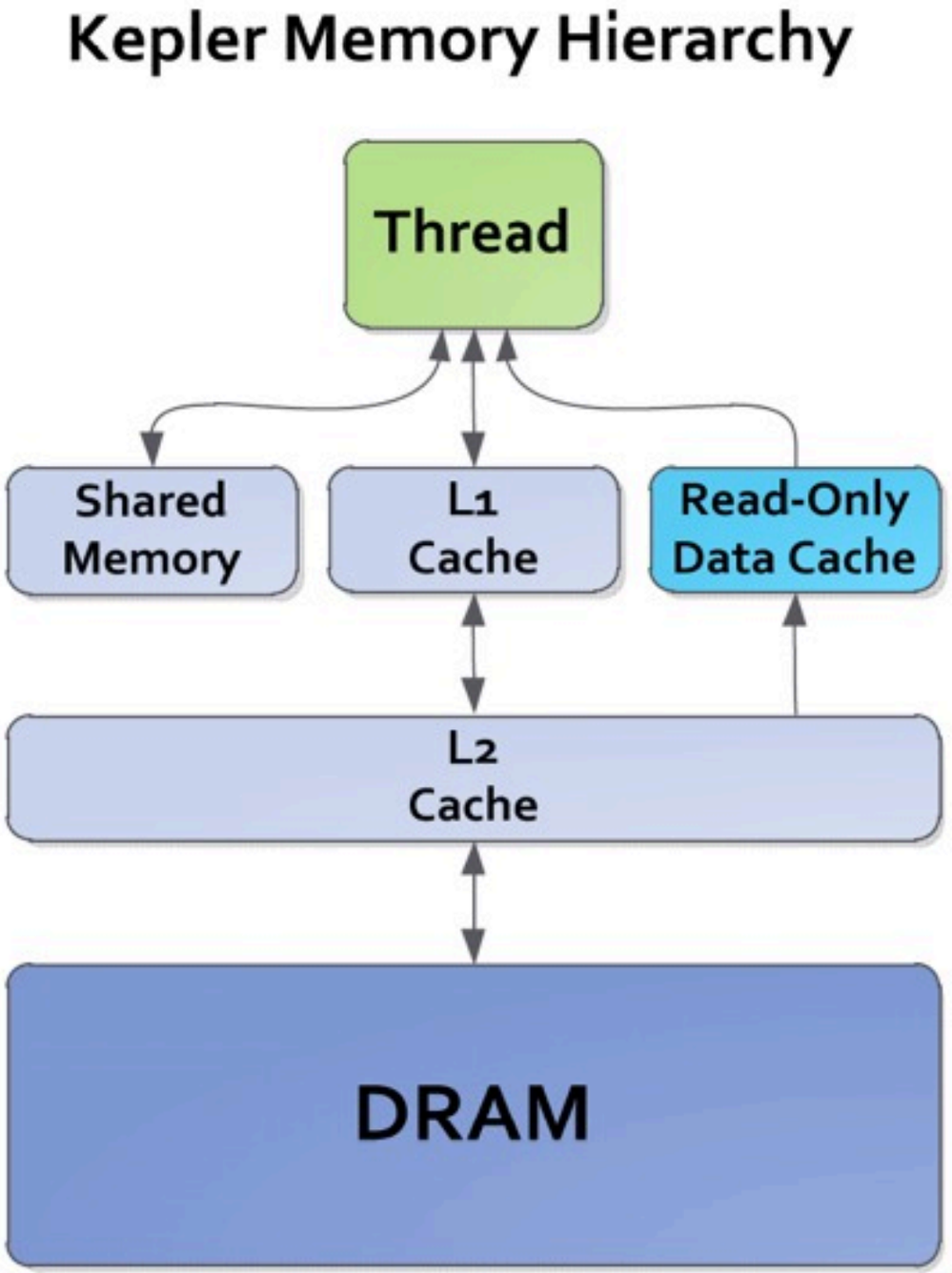
NVIDIA.

The memory in Tesla cards: Fermi vs. Kepler

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

Differences in memory hierarchy: Fermi vs. Kepler





The memory hierarchy in numbers

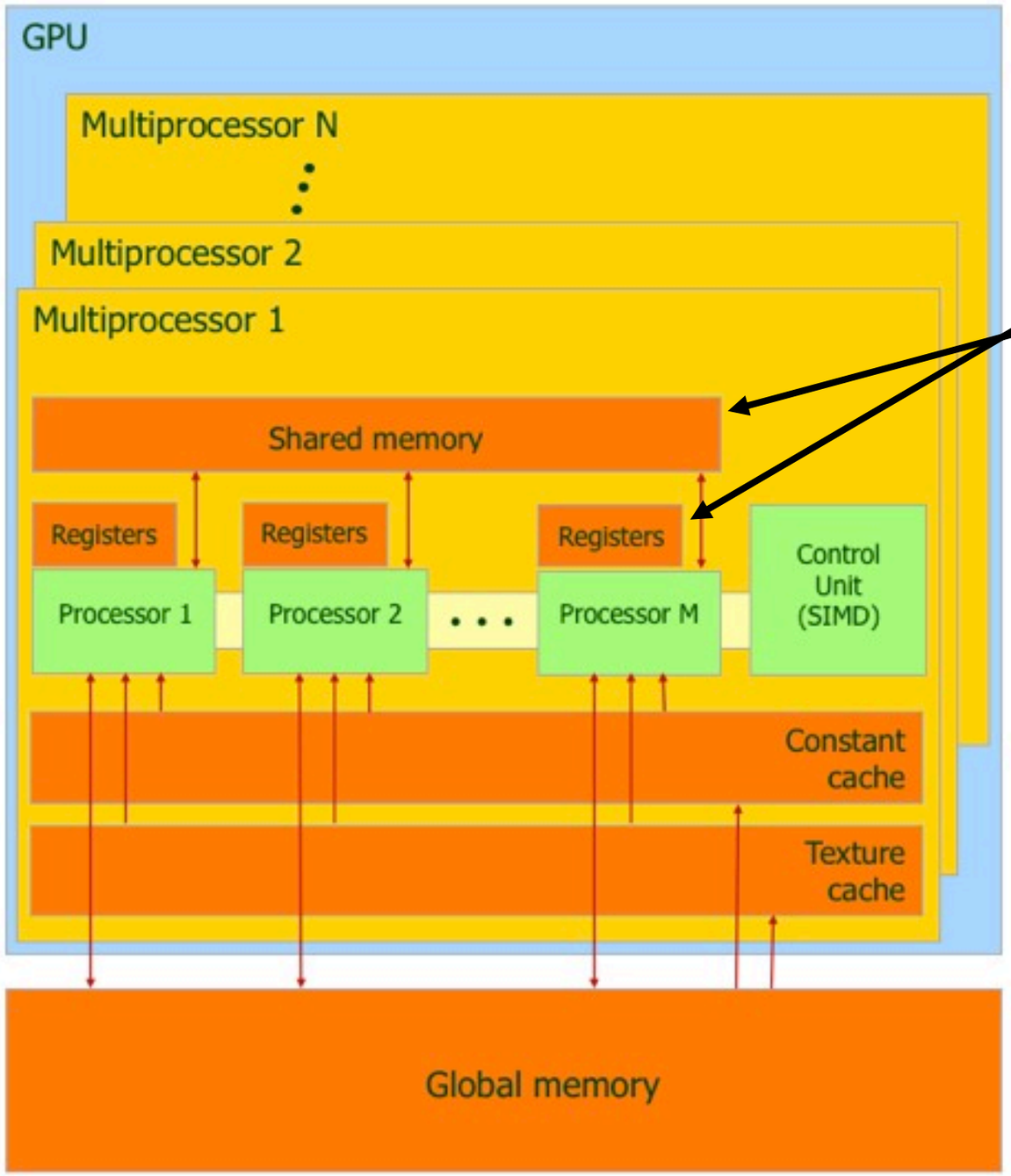
GPU generation	Fermi		Kepler		Limi-tation	Impact
Hardware model	GF100	GF104	GK104	GK110		
CUDA Compute Capability (CCC)	2.0	2.1	3.0	3.5		
Max. 32 bits registers / thread	63	63	63	255	SW.	Working set
32 bits registers / Multiprocessor	32 K	32 K	64 K	**64 K**	HW.	Working set
Shared memory / Multiprocessor	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)

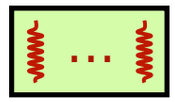
3. The SMX cores

NVIDIA

A brief reminder of what CUDA is about



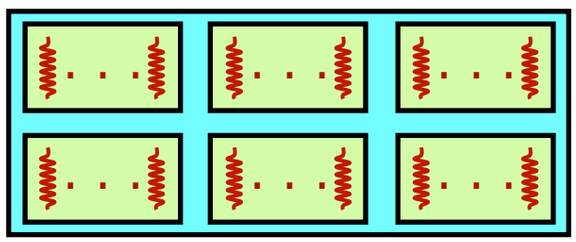
Thread



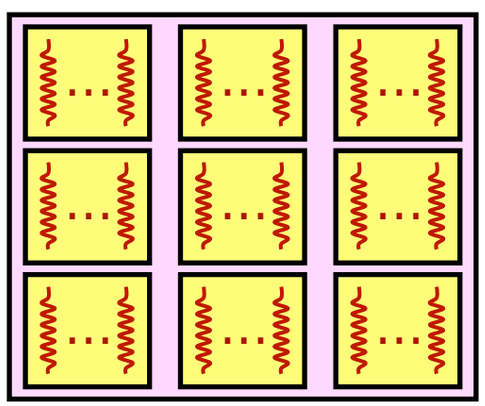
Thread block

On-chip memory

Memory outside the GPU chip (but within the graphics card)



Grid 0



Grid 1

… and how the architecture scales up

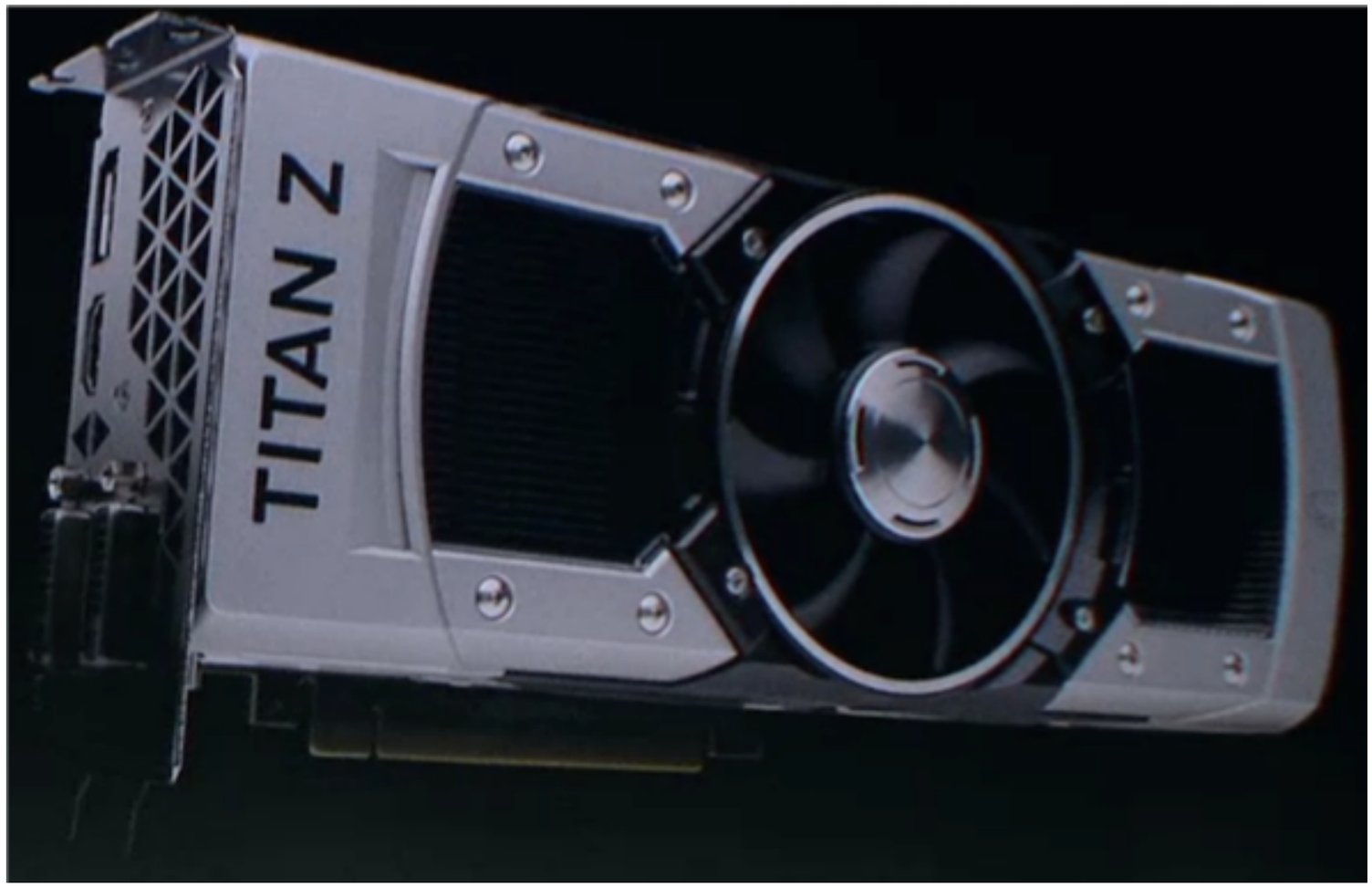
	Tesla		Fermi		Kepler			
Architecture	**G80**	**GT200**	**GF100**	**GF104**	**GK104 (K10)**	**GK110 (K20)**	**GK110 (K40)**	**GeForce GTX Titan Z**
Time frame	**2006-07**	**2008-09**	**2010**	**2011**	**2012**	**2013**	**2013-14**	**2014**
CUDA Compute Capability (CCC)	**1.0**	**1.2**	**2.0**	**2.1**	**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

Kepler in perspective: Hardware resources and peak 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

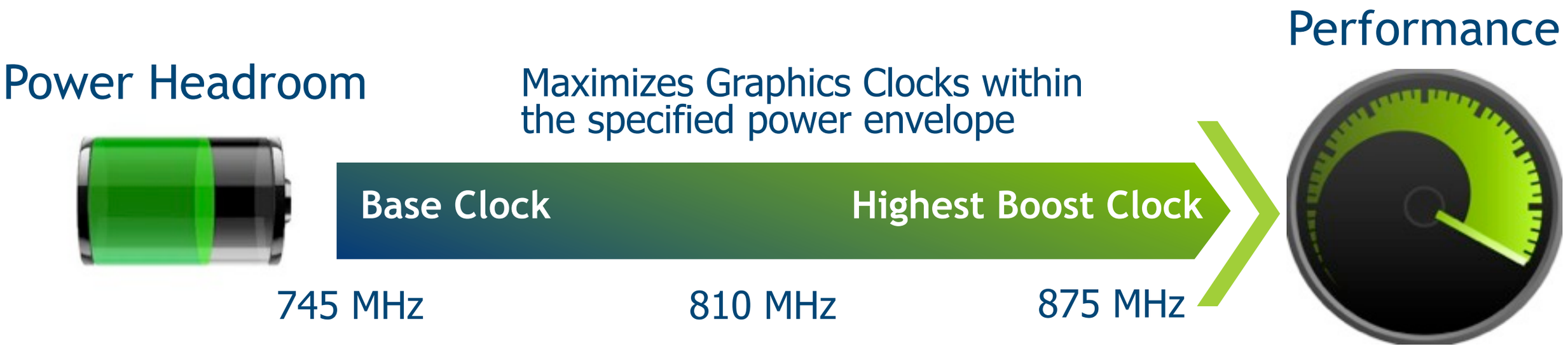
The new GeForce GTX Titan Z

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



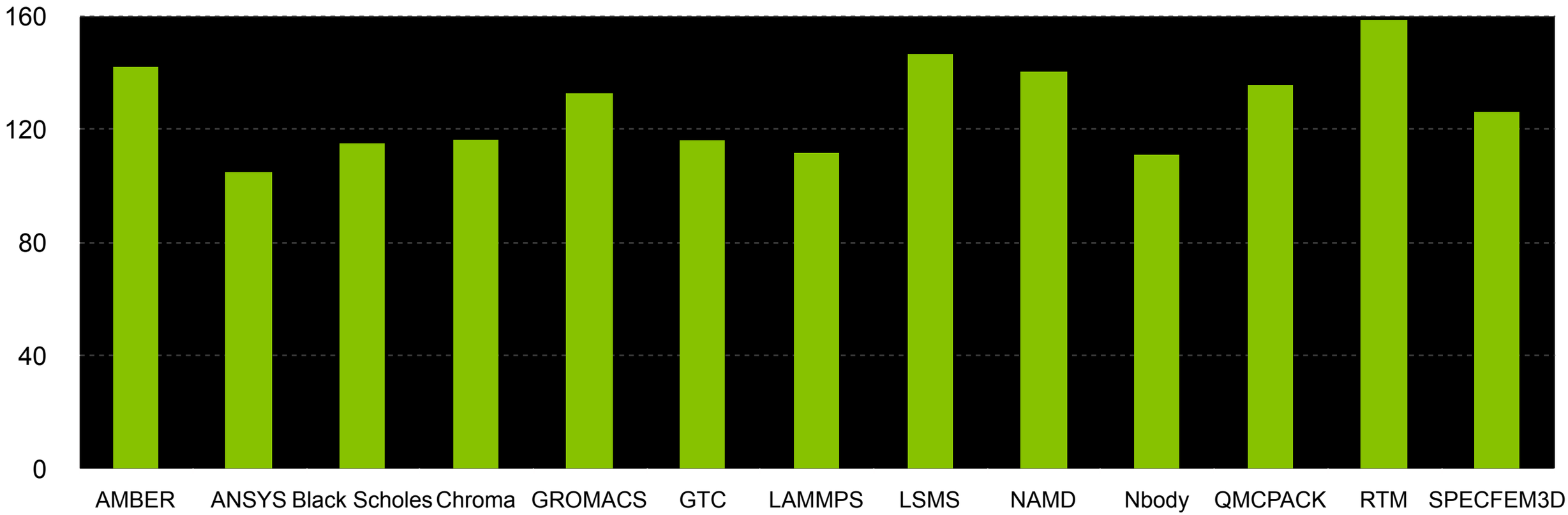
GPU Boost

- 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.



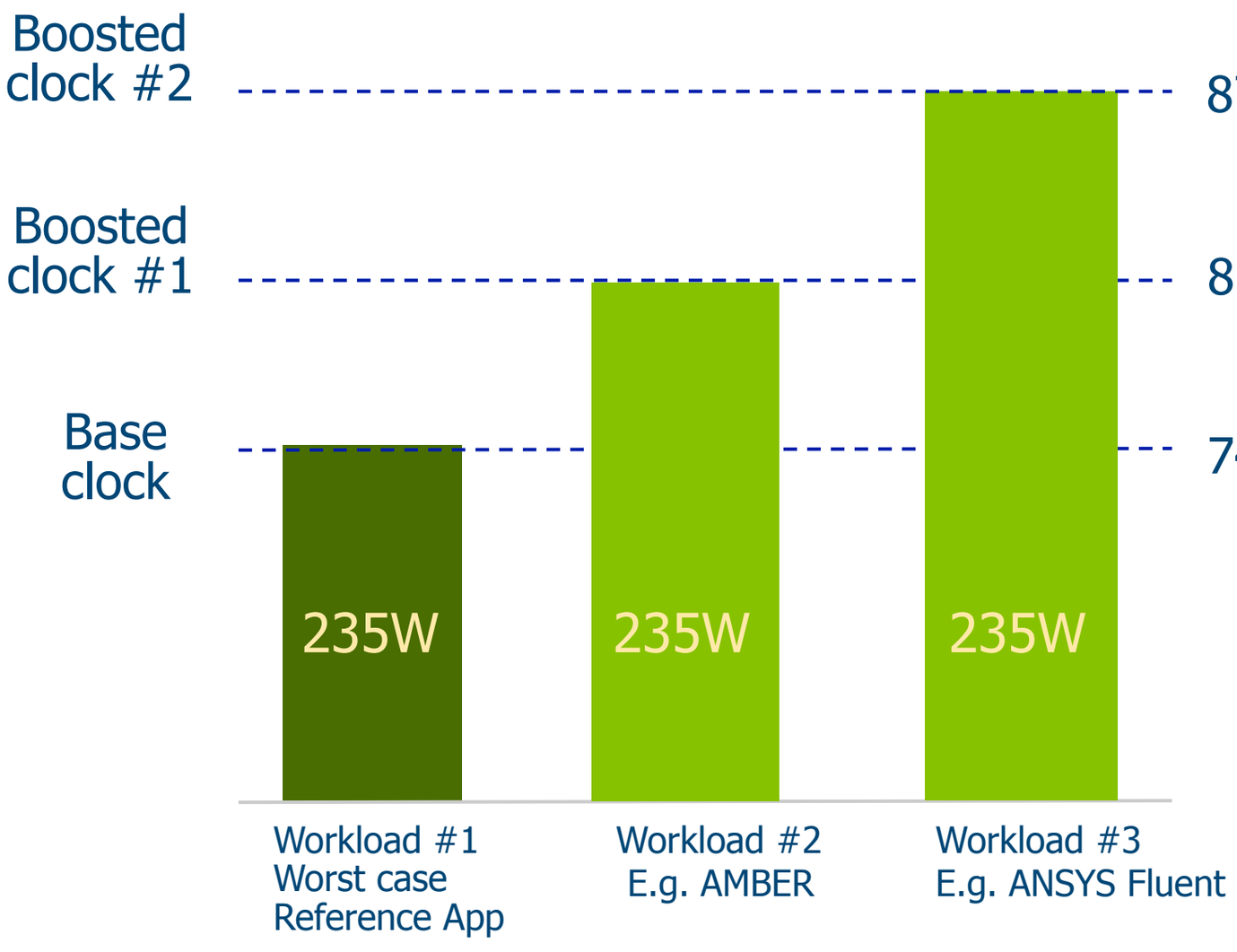
Every application has a different behaviour regarding power consumption

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



Those applications which are less power hungry can benefit from a higher clock rate

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

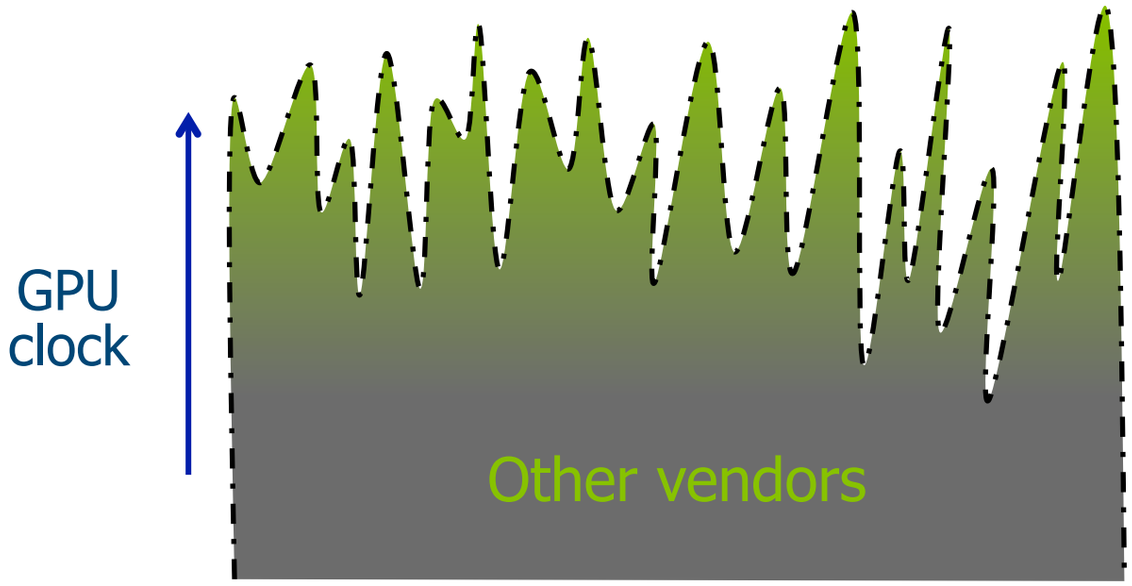


Up to 40% higher performance relative to Tesla K20X.

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

GPU Boost compared to other approaches

- 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

GPU Boost - List of commands

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

Example: Query the clock in use

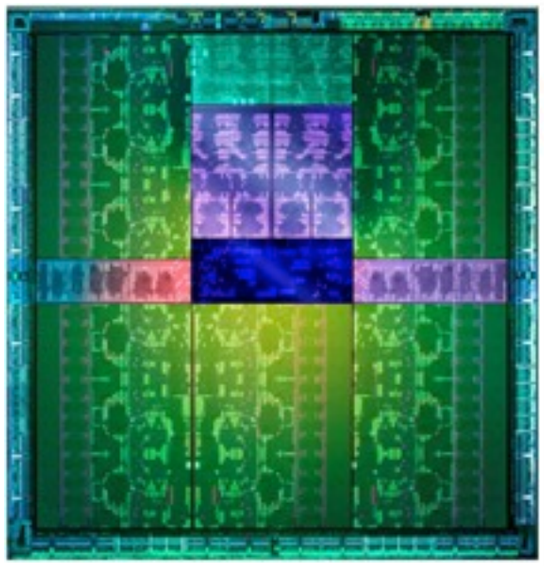
- `nvidia-smi -q -d CLOCK –id=0000:86:00.0`

```
==============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
```

4. How the SMX works: Front-end and back-end

NVIDIA

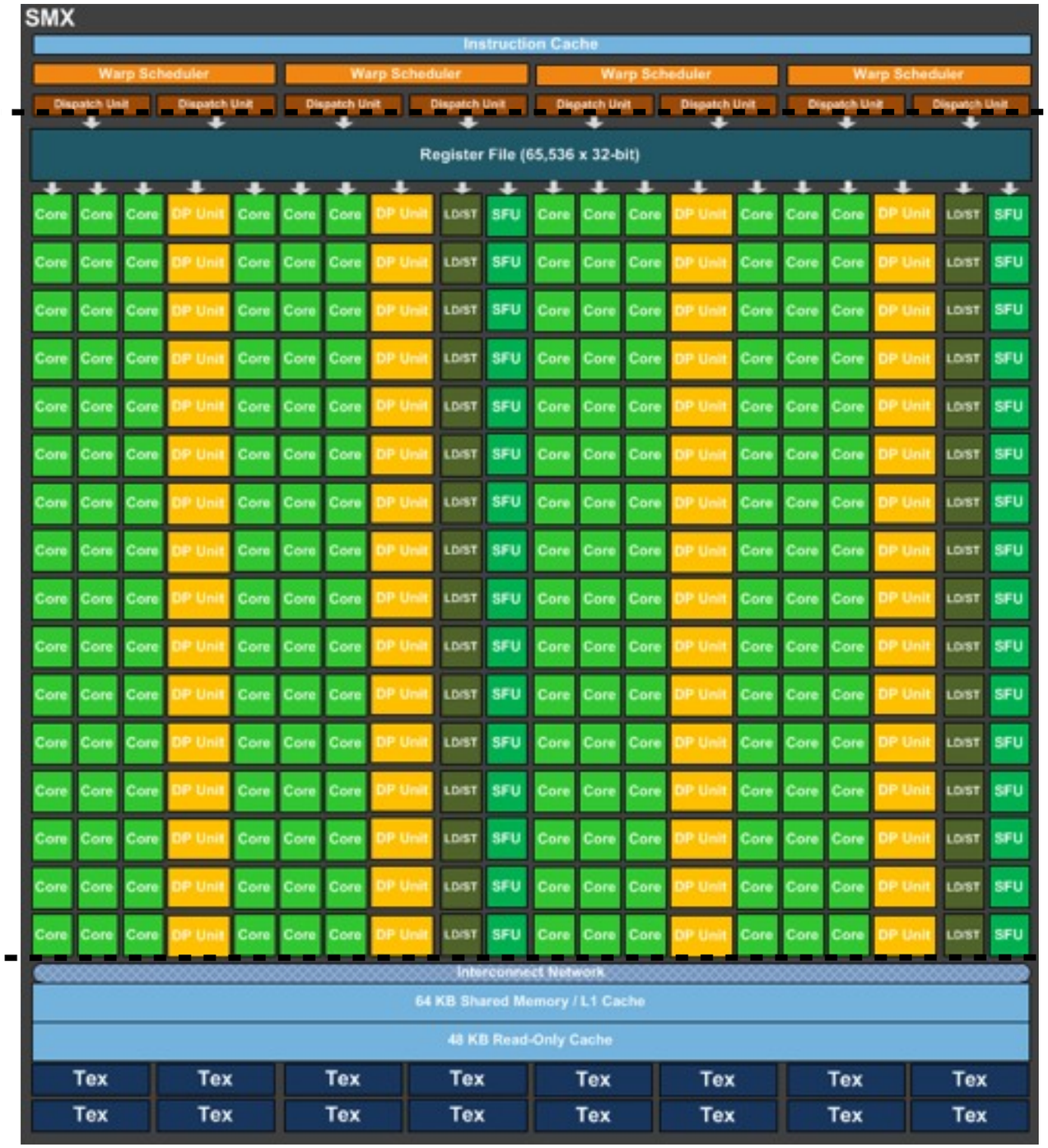
Kepler GK110: Physical layout of functional units for the Tesla K40 (endowed with 15 SMX)



The SMX multiprocessor

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

From SM multiprocessor in Fermi GF100 to SMX multiprocessor in Kepler GK110

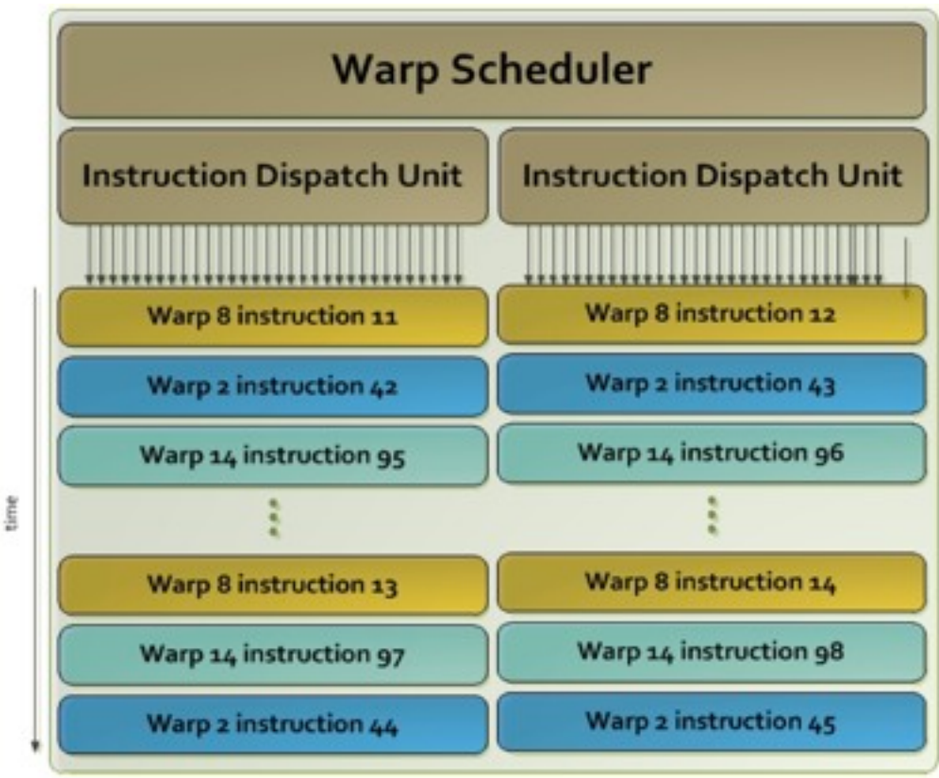
A comparison between instructions issue and execution (front-end vs. back-end)

	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.

The way GigaThread scheduling works

- Each grid provides a number of blocks, which are assigned to SMXs (up to 16 blocks per SMX in Kepler, 8 in Fermi).
- Blocks are split into warps (groups) of 32 threads.
- Warps are issued for each instruction in kernel threads (up to 64 active warps in Kepler, 48 in Fermi). Example:



Increasing concurrency and massive parallelism

GPU generation	Fermi		Kepler	
Hardware model	**GF100**	**GF104**	**GK104**	**GK110**
CUDA Compute Capability (CCC)	**2.0**	**2.1**	**3.0**	**3.5**
Number of threads / warp (warp size)	32	32	32	32
Max. number of warps / Multiprocessor	48	48	64	64
Max. number of blocks / Multiprocessor	8	8	16	16
Max. number of threads / Block	1024	1024	1024	1024
Max. number of threads / Multiprocessor	1536	1536	**2048**	**2048**

Crucial enhancements for hiding latencies

Max. concurrency on each SMX

Express as much parallelism as possible: SMXs (Kepler) are wider than SMs (Fermi)

Tetris (tile = warp_instr.):
- Issues 4 warp_instrs.
- Executes up to 10 warps = 320 threads.
- Warp_instrs. are symmetric and executed all in one cycle.

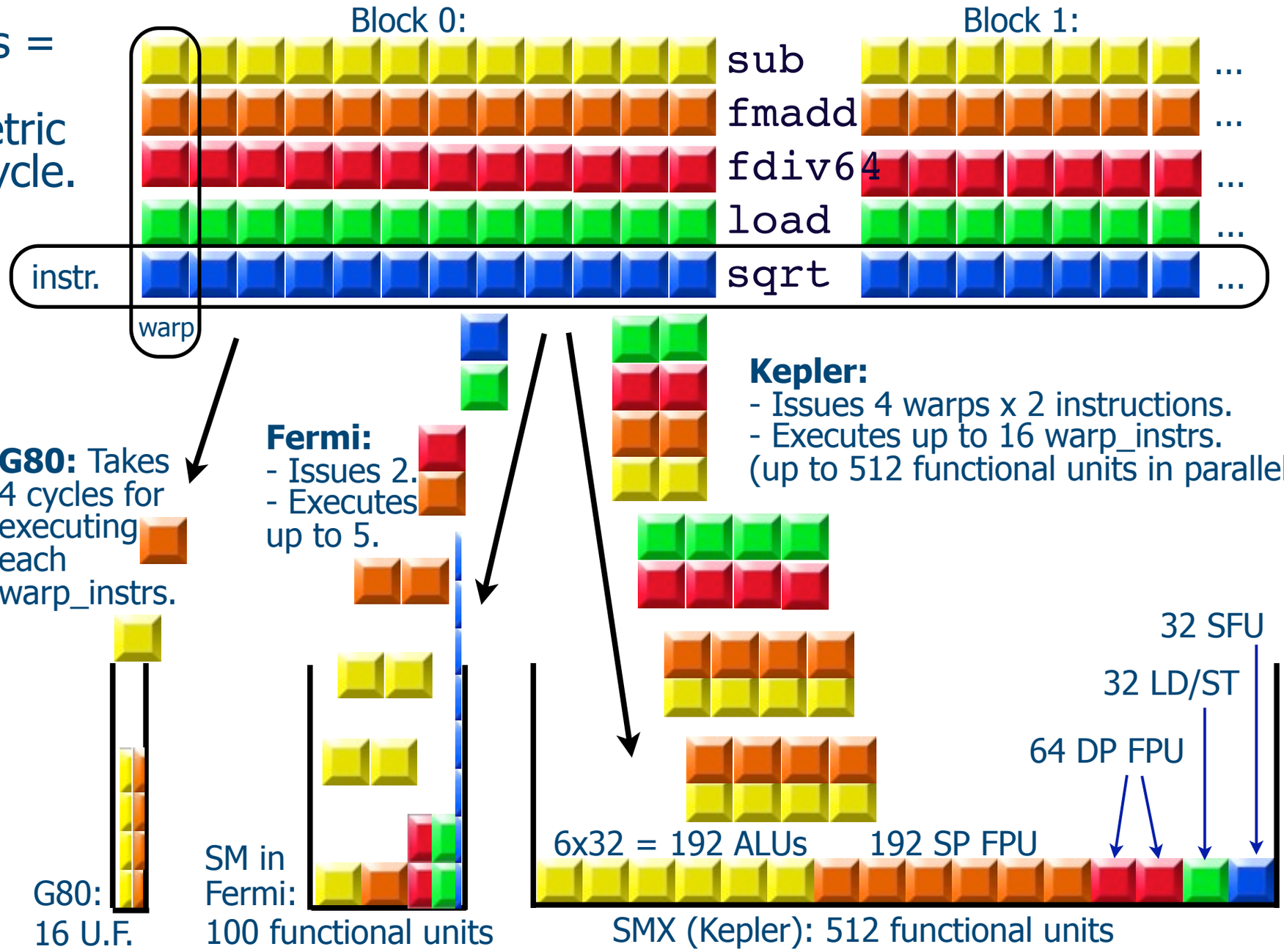
Color code:
- for instructions using "int".
- for instrs. using "float".
- "double".
- "load/store".
- "log/sqrt...".

Issues 4 warp_instrs.

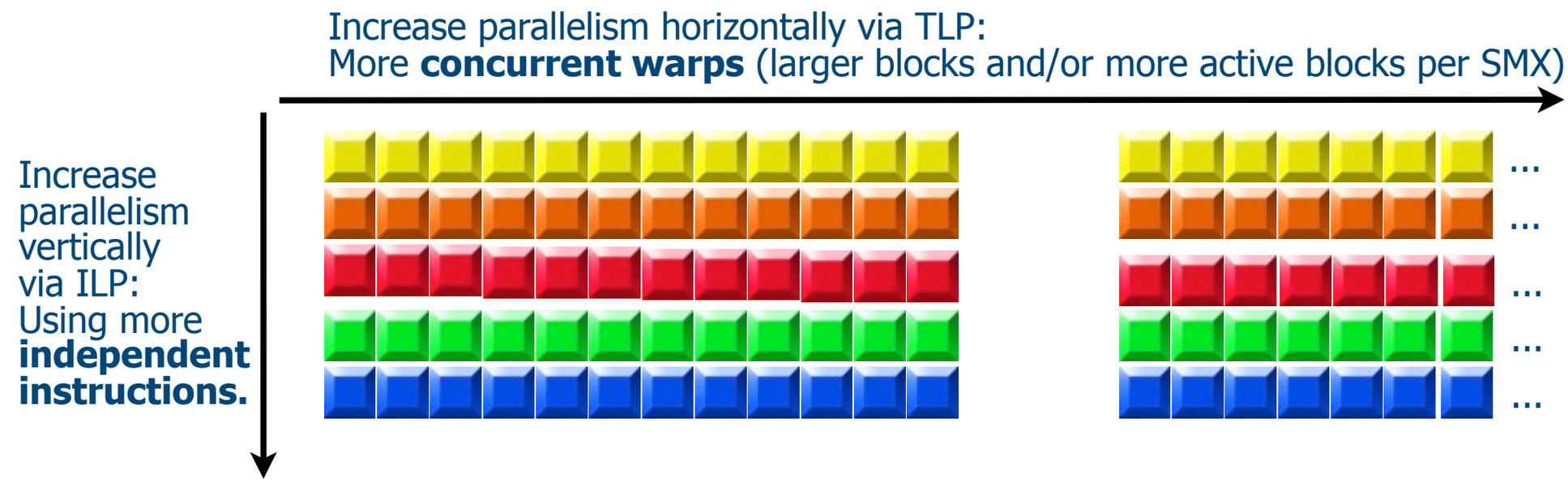
The player is the GPU scheduler! You can rotate moving pieces if there are no data dependencies.

Executes up to 10 warp_instrs.

Example: Kernel with blocks of 384 threads (12 warps).

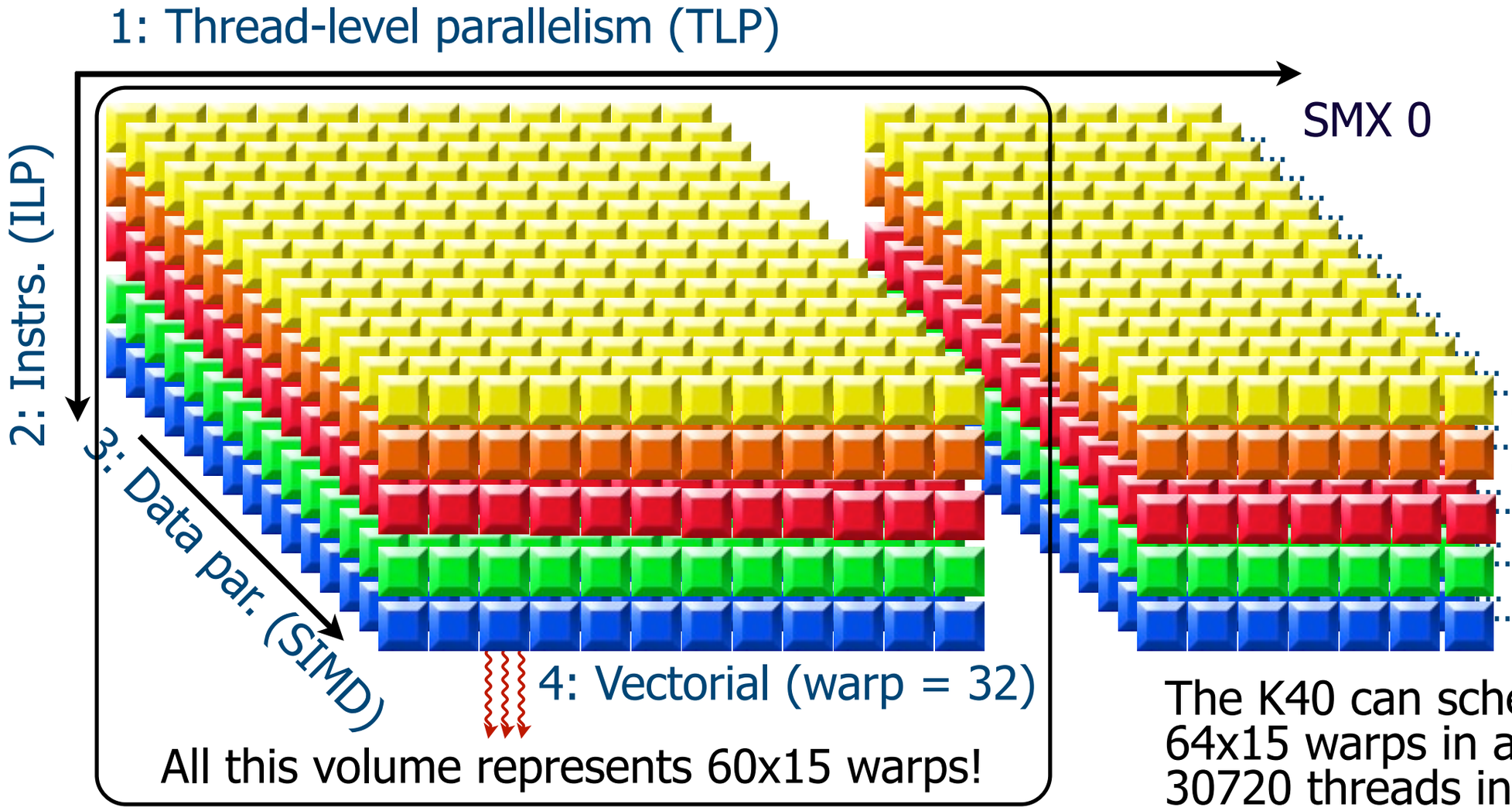


Thread Level Parallelism (TLP) and Instruction Level Parallelism (ILP)



- SMXs can leverage available ILP interchangeably with TLP:
  - It is much better at this than Fermi.
- Sometimes is easier to increase ILP than TLP (for example, a small loop unrolling):
  - # of threads may be limited by algorithm or HW limits.
- We need ILP for attaining a high IPC (Instrs. Per Cycle).

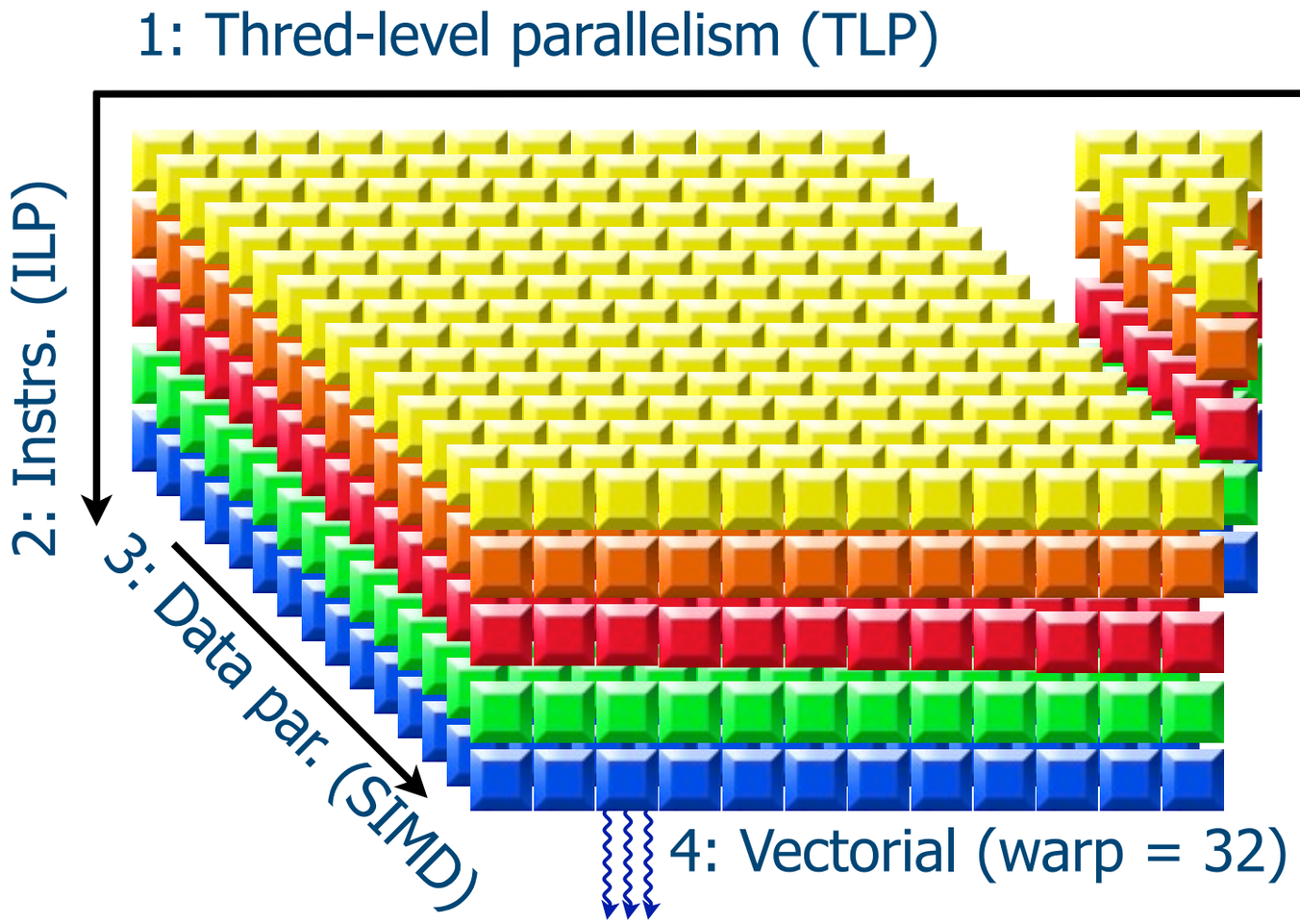
Kepler GPUs can hold together **all** forms of parallelism. Example: K40.

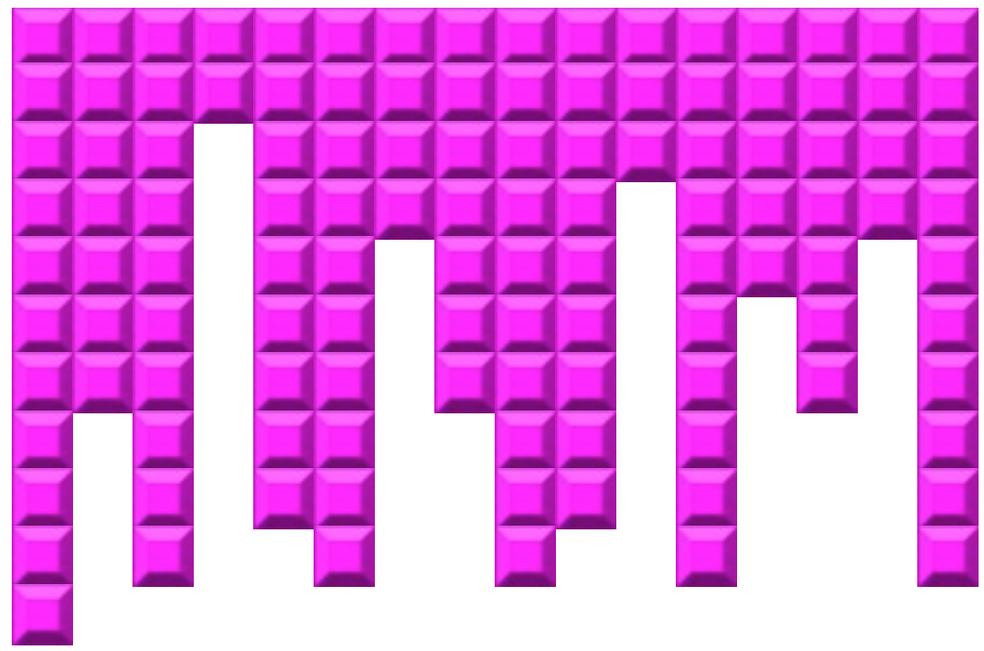


The K40 can schedule up to 64x15 warps in a single cycle: 30720 threads in 1.14 ns.

- Imagine a 3D tetris with 15 boxes and up to 64 pieces falling down simultaneously on each of them, because that is the way K40 works when all parallelism is deployed.

A quick introduction to our hands-on

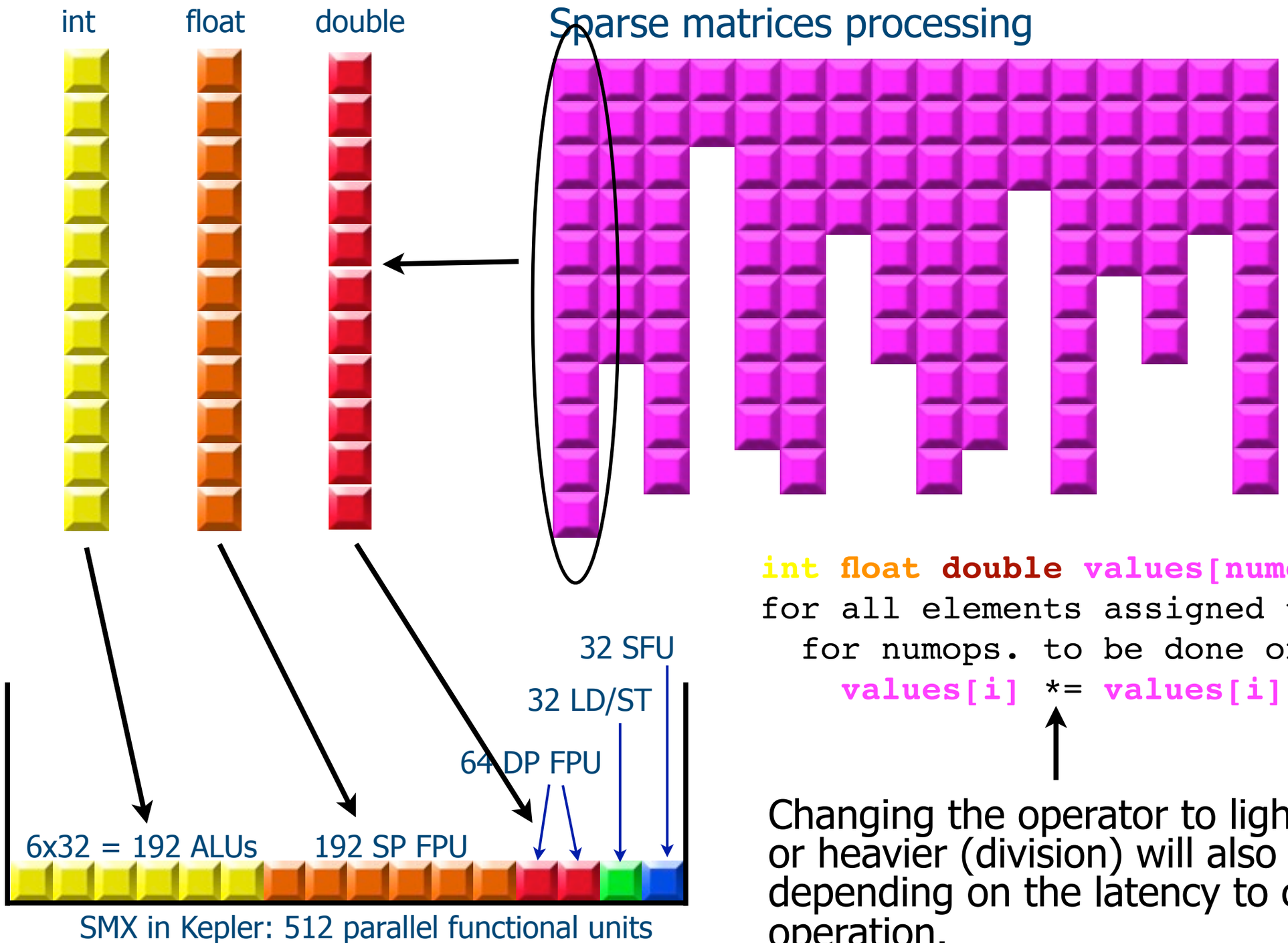




Our code traverses the whole matrix, performing operations independently on each element.

- Base strategy:
  - We launch a CUDA kernel for each matrix column.
  - Each kernel will have the lowest number of blocks.
  - Each kernel will have the largest number of warps.

A quick introduction to our hands-on (2)




```
int float double values[numelements];
for all elements assigned to each thread:
  for numops. to be done on each element
    values[i] *= values[i];
```

Changing the operator to lighter (addition) or heavier (division) will also have an impact depending on the latency to carry out that operation.

Case study: Zernike moments

GPU resources	ALU	32-bits FPU	64-bits FPU	Load/store	SFU
Fermi	32%	32%	16%	16%	4%
Kepler	37.5%	37.5%	12.5%	6.25%	6.25%
Kernel for Zernike	54%	21%	0%	25%	0%
Better	Kepler	Fermi	Kepler	Fermi	Fermi

- Fermi is more balanced in this case.
- With the resources distribution in Kepler, the execution of integer arithmetic improves, but the floating-point arithmetic and the load/store worsens. All the others are not used.

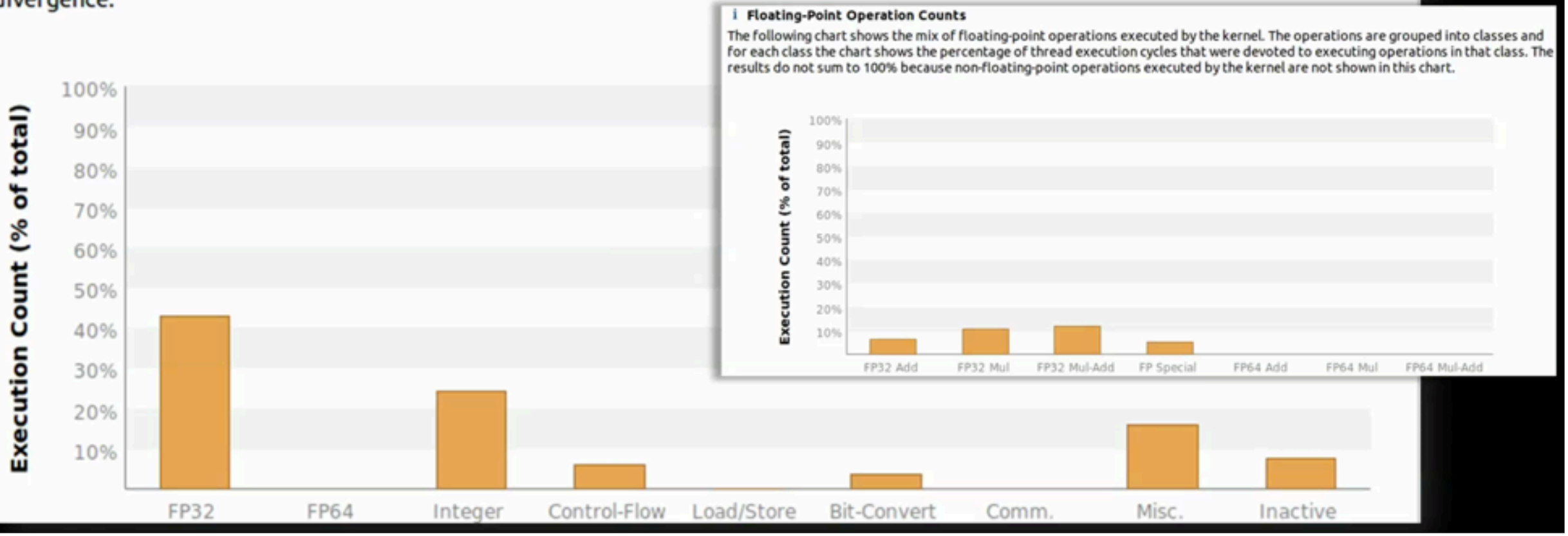
Use the CUDA Visual Profiler to know how good your application adapts to resources

Detailed Instruction Mix Visualization

Visual Profiler and NSight EE

Instruction Execution Counts

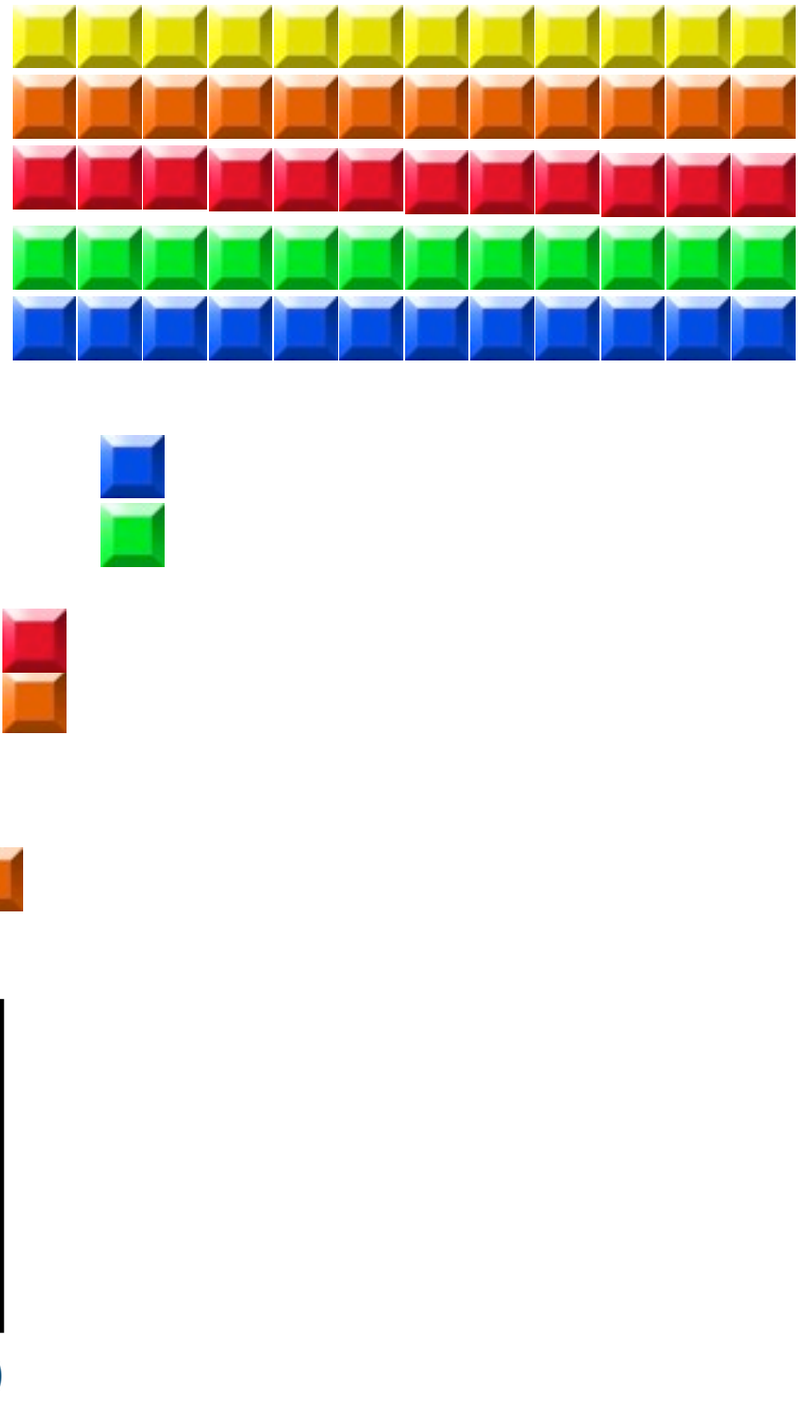
The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing instructions in that class. The "Inactive" result shows the thread executions that did not execute any instruction because the thread was predicated or inactive due to divergence.



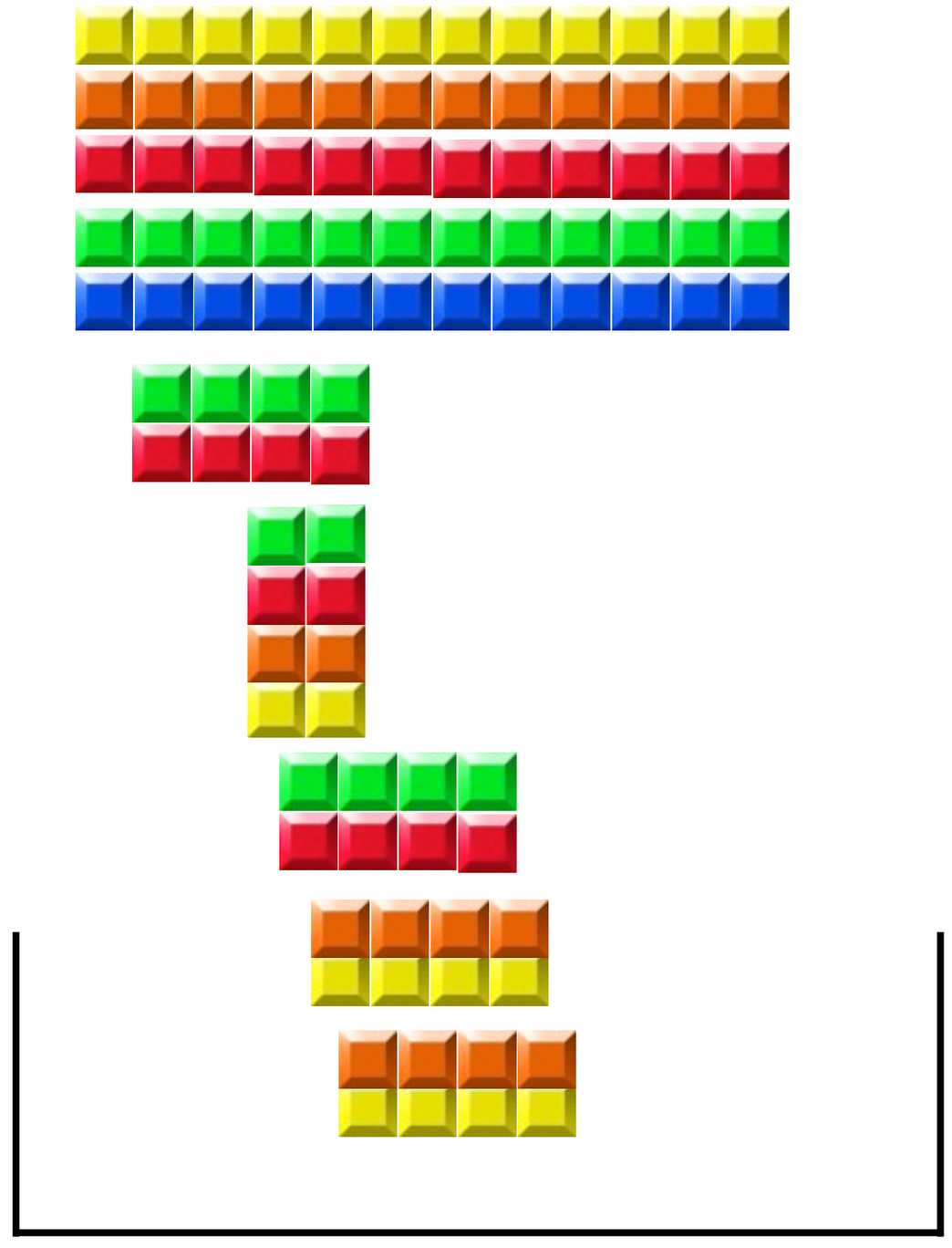
Floating-Point Operation Counts

The following chart shows the mix of floating-point operations executed by the kernel. The operations are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing operations in that class. The results do not sum to 100% because non-floating-point operations executed by the kernel are not shown in this chart.

The way the GPU front-end works: (1) How warps are scheduled



SM (Fermi)

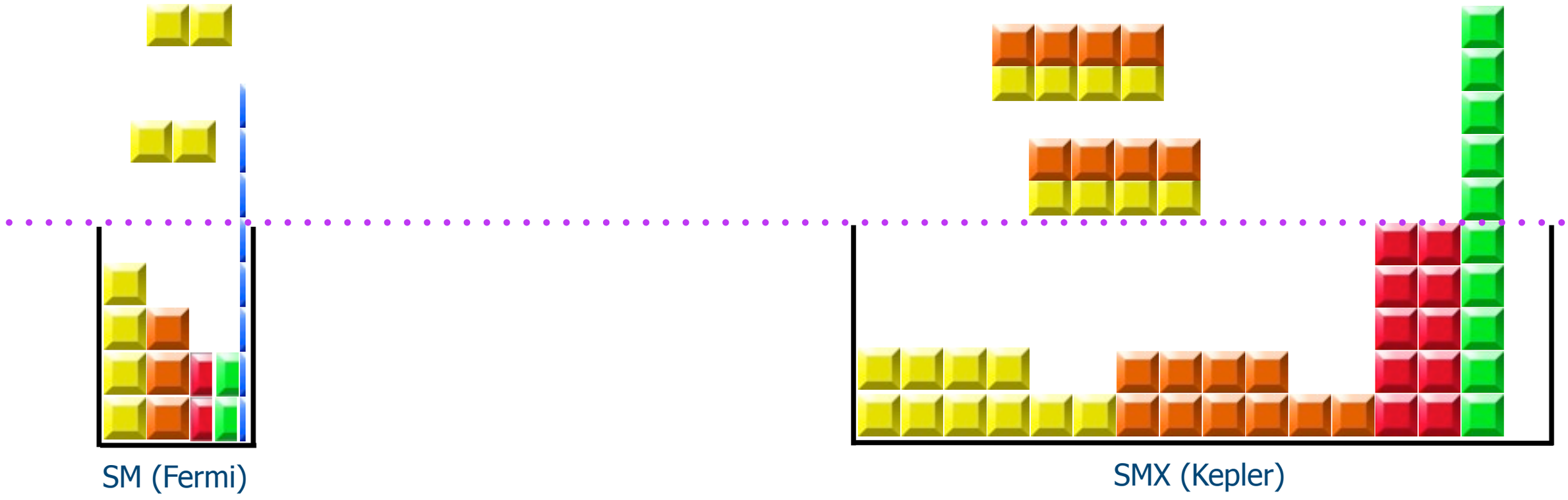


SMX (Kepler)

38

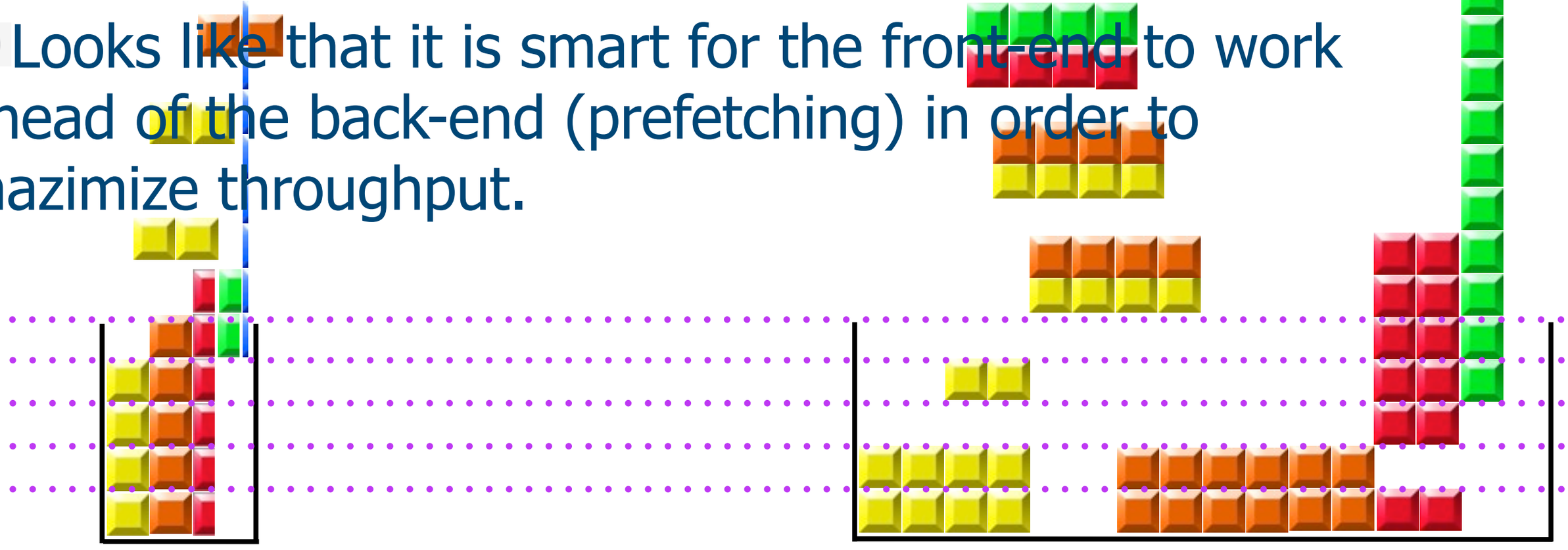
The interface between front-end & back-end: (2) How warps are issued

- 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.



The way the GPU back-end works: (3) Warps execution

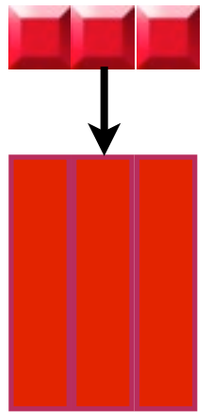
- Let us assume that when we start the execution there are few warps pending to be executed:
  - Two single precision warps (orange).
  - Two double precision warps (red).
- Looks like that it is smart for the front-end to work ahead of the back-end (prefetching) in order to mazimize throughput.



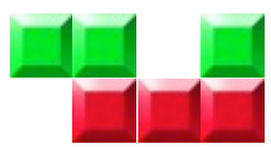
Some remarks about the "tetris" model

- 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.



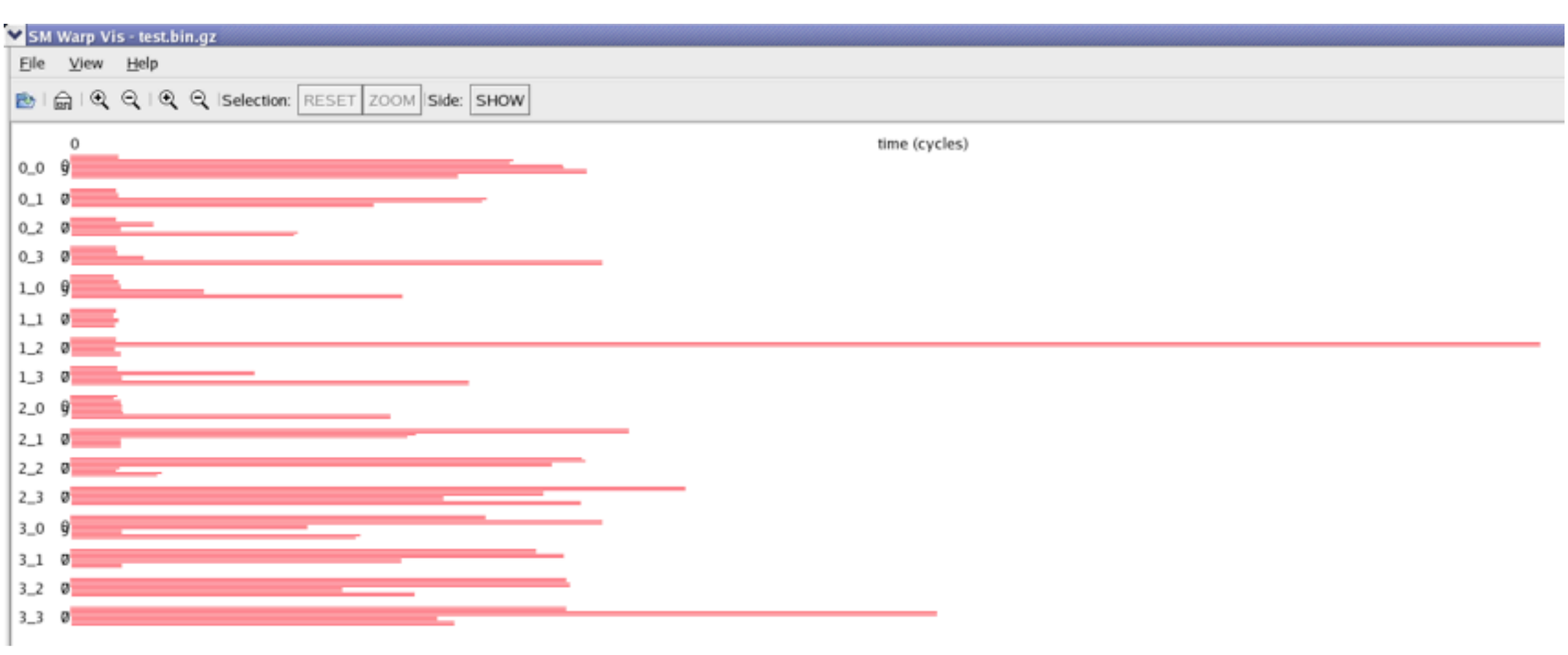
- Bricks can assemble tiles which are not contiguous.

Warps latency

- 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.

The warps behaviour teaches us that the GPU is not a regular processor at all

- Unpredictable factors at run-time pose a challenge for the workload balance among multiprocessors. Here is an example of the variance for the last 8 warps executed on each multiprocessor of a G80 GPU:



43

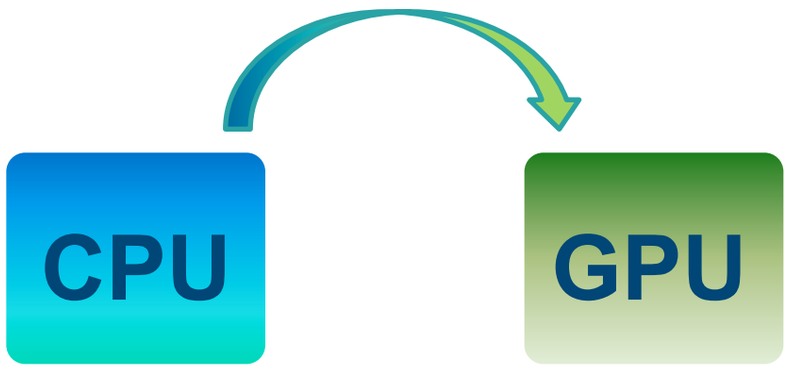
5. Functional improvements

NVIDIA

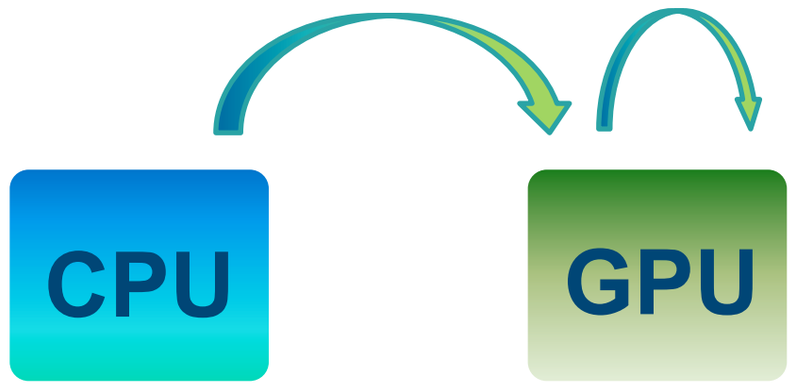
5.1. Dynamic parallelism

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.



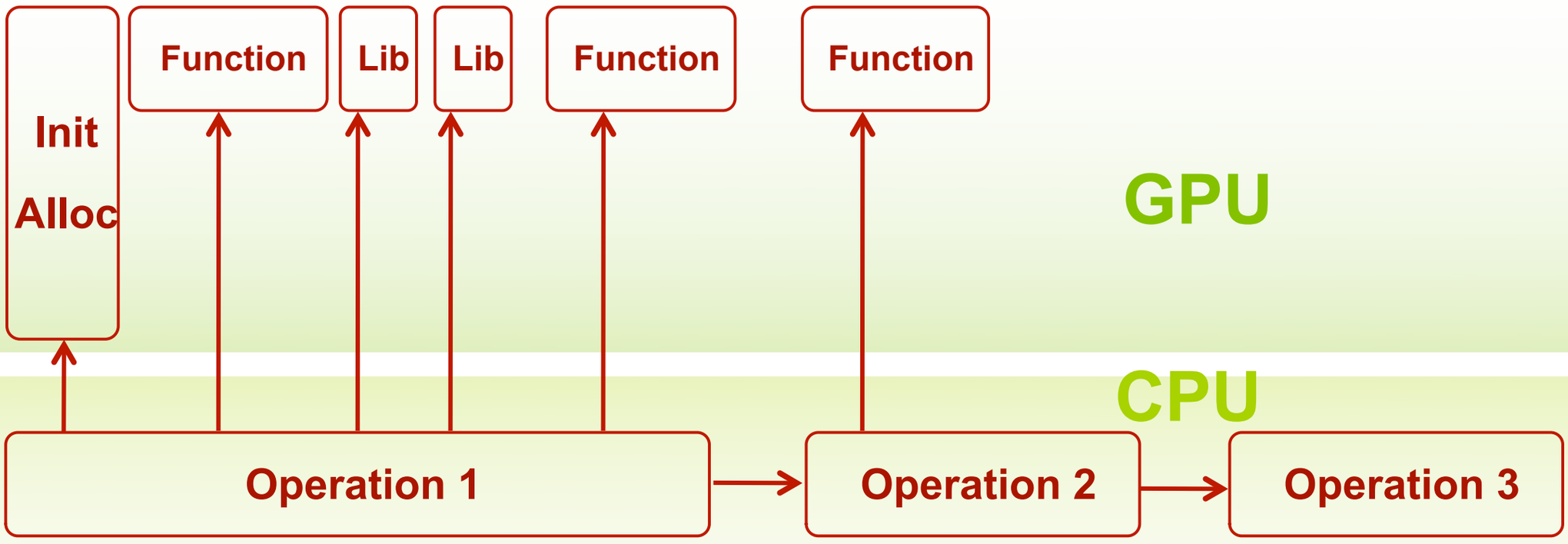
Fermi: Only CPU
can generate GPU work.



Kepler: GPU can
generate work for itself.

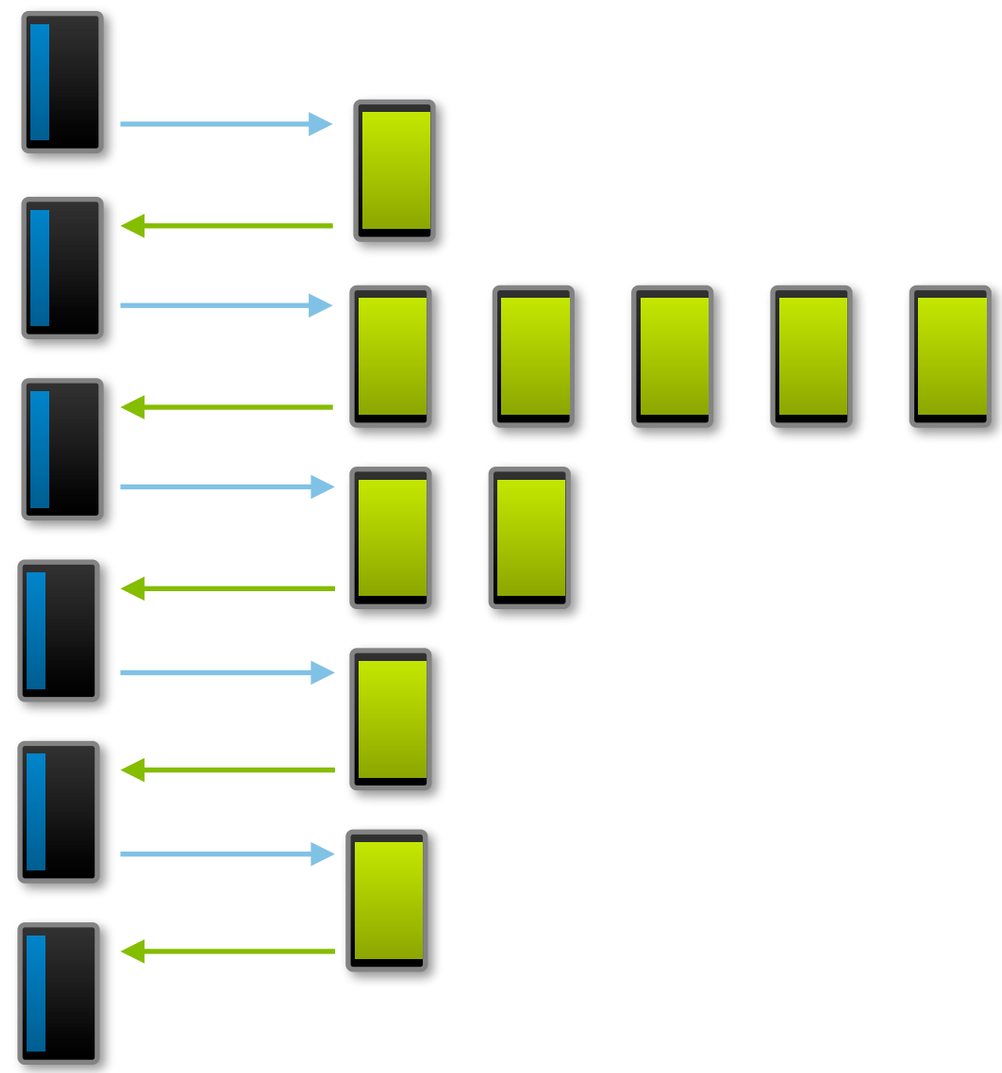
The way we did things in the pre-Kepler era: The GPU was a slave for the CPU

- 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).

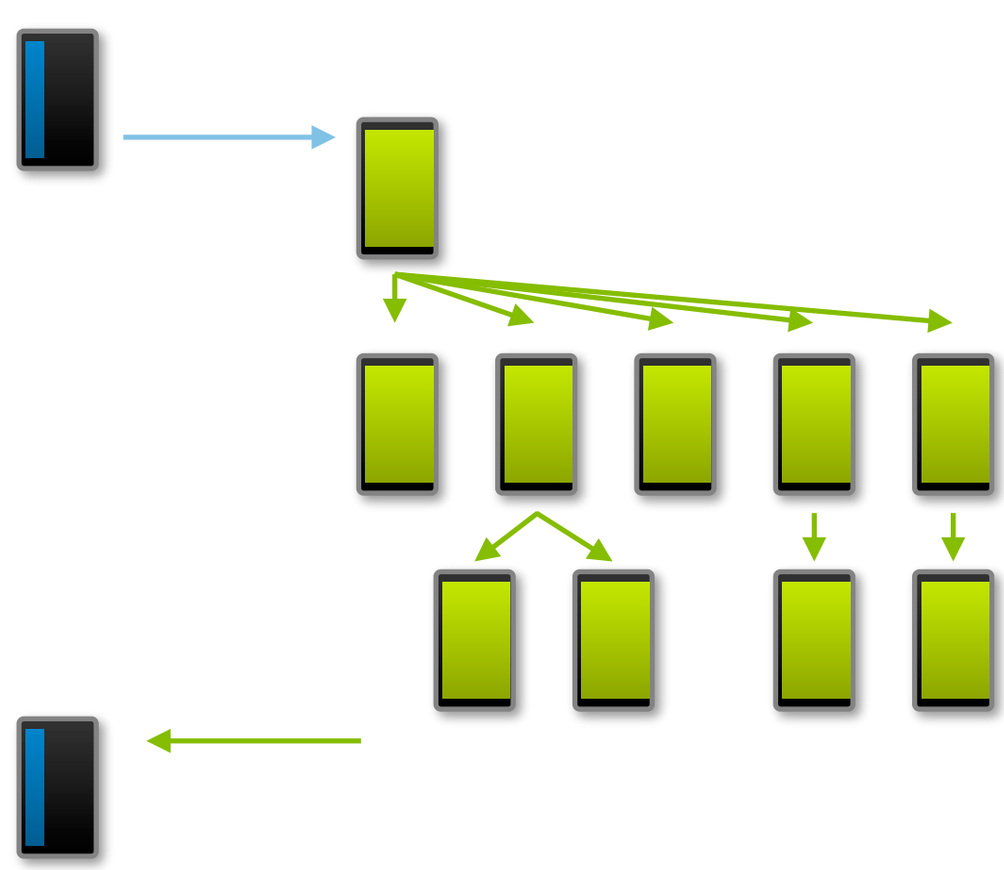


The way we do things in Kepler: GPUs launch their own kernels

The pre-Kepler GPU is a co-processor



The Kepler GPU is autonomous: Dynamic parallelism

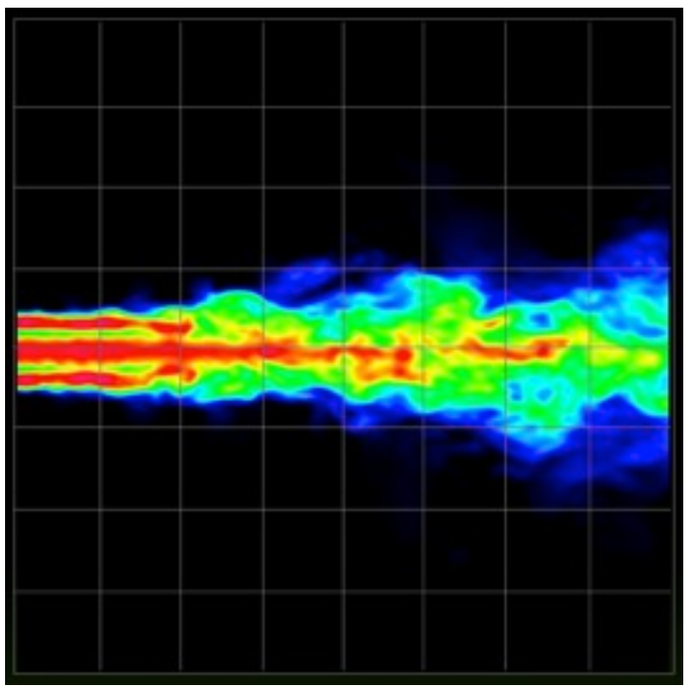


Now programs run faster and are expressed in a more natural way.

Example 1: Dynamic work generation

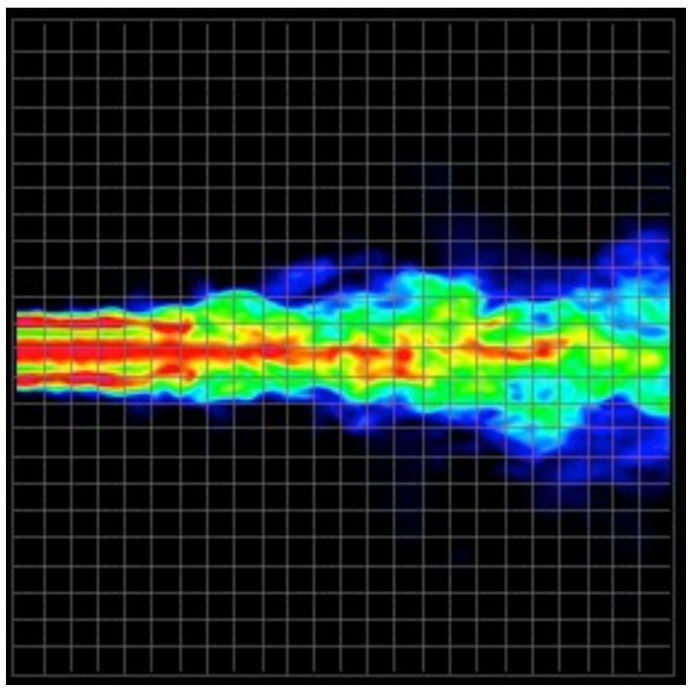
- Assign resources dynamically according to real-time demand, making easier the computation of irregular problems on GPU.
- It broadens the application scope where it can be useful.

Coarse grid



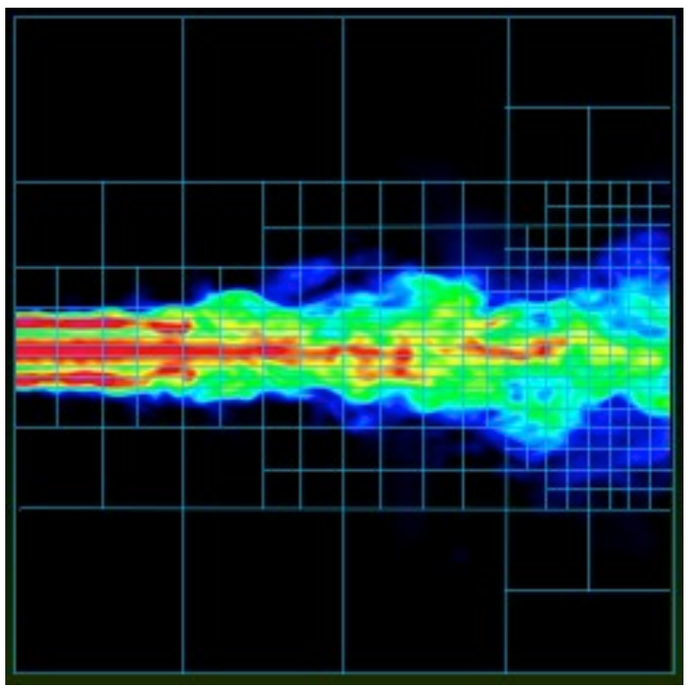
Higher performance, lower accuracy

Fine grid



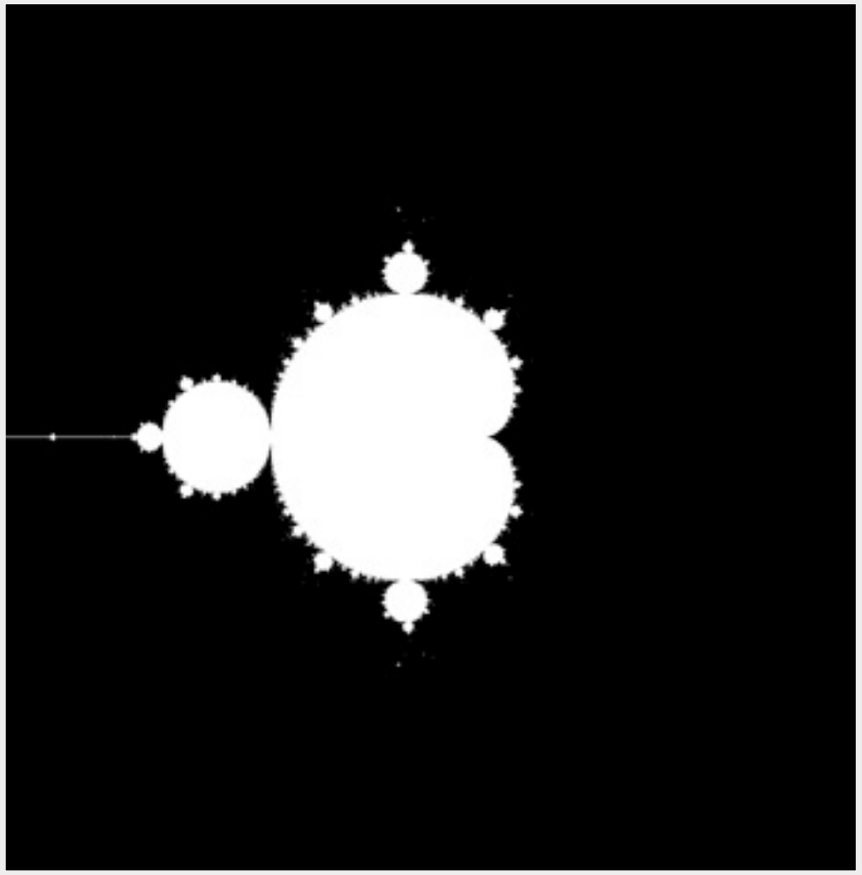
Lower performance, higher accuracy

Dynamic grid

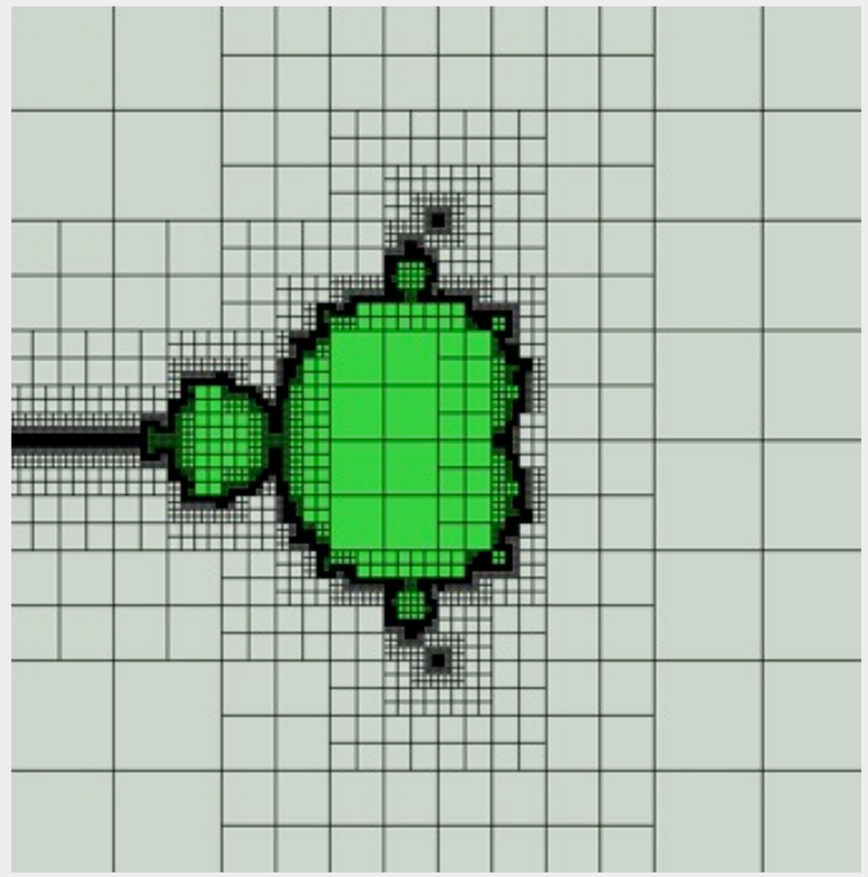


Target performance where accuracy is required

Example 2: Deploying parallelism based on level of detail



Computational power allocated to regions of interest



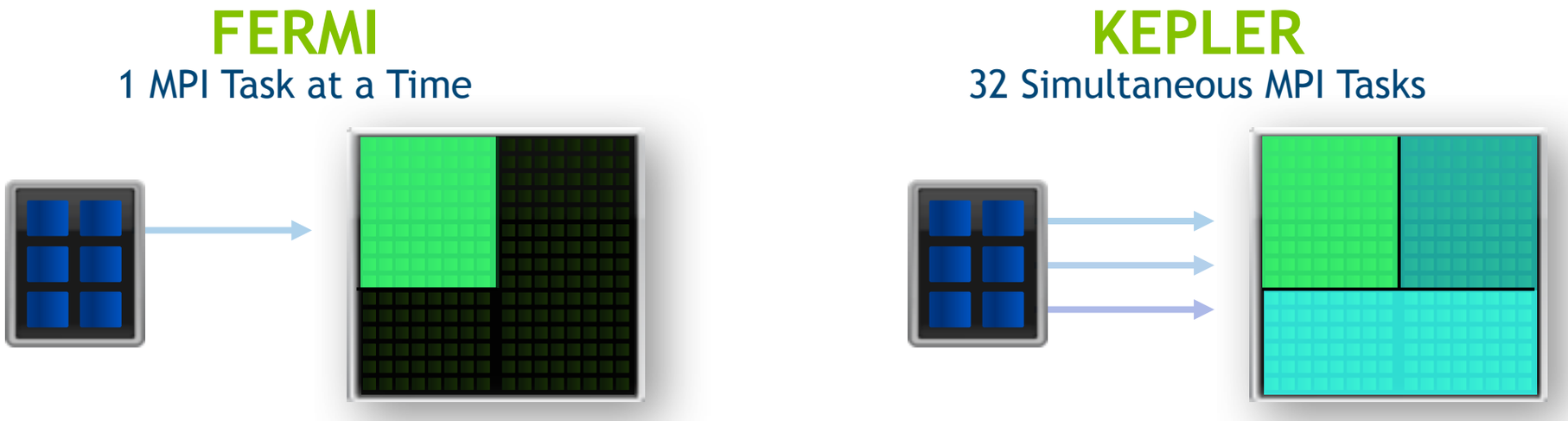
CUDA until 2012:
- The CPU launches kernels regularly.
- All pixels are treated the same.

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

5.2. Hyper-Q

Hyper-Q

- 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.



An example:
3 streams, each composed of 3 kernels

```
__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, ...);
...
```

stream 1

```
kernel_A <<< dimgridA, dimblockA, 0, stream_1 >>> (pars);
kernel_B <<< dimgridB, dimblockB, 0, stream_1 >>> (pars);
kernel_C <<< dimgridC, dimblockC, 0, stream_1 >>> (pars);
```

stream 2

```
...
kernel_P <<< dimgridP, dimblockP, 0, stream_2 >>> (pars);
kernel_Q <<< dimgridQ, dimblockQ, 0, stream_2 >>> (pars);
kernel_R <<< dimgridR, dimblockR, 0, stream_2 >>> (pars);
```

stream 3

```
...
kernel_X <<< dimgridX, dimblockX, 0, stream_3 >>> (pars);
kernel_Y <<< dimgridY, dimblockY, 0, stream_3 >>> (pars);
kernel_Z <<< dimgridZ, dimblockZ, 0, stream_3 >>> (pars);
```

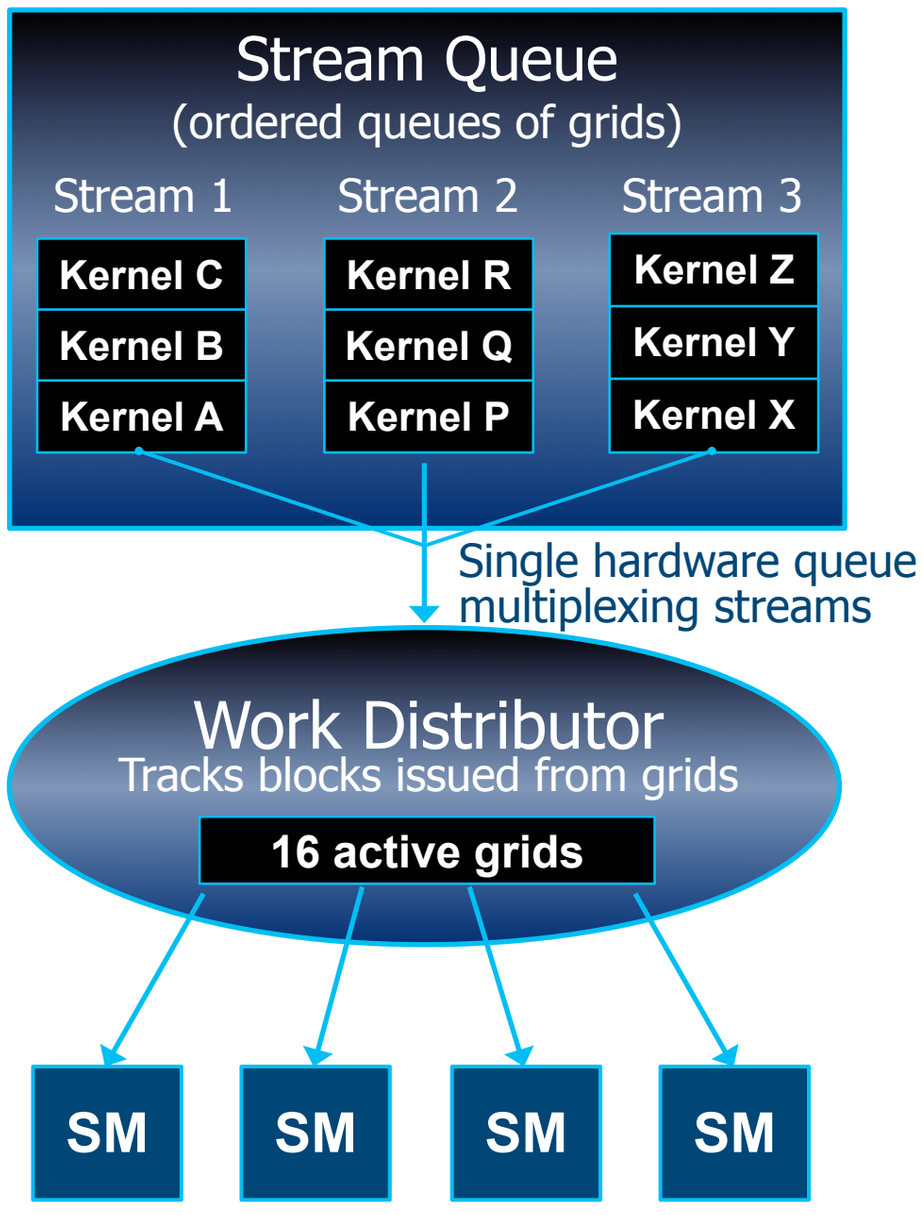
stream_1
kernel_A
kernel_B
kernel_C

stream_2
kernel_P
kernel_Q
kernel_R

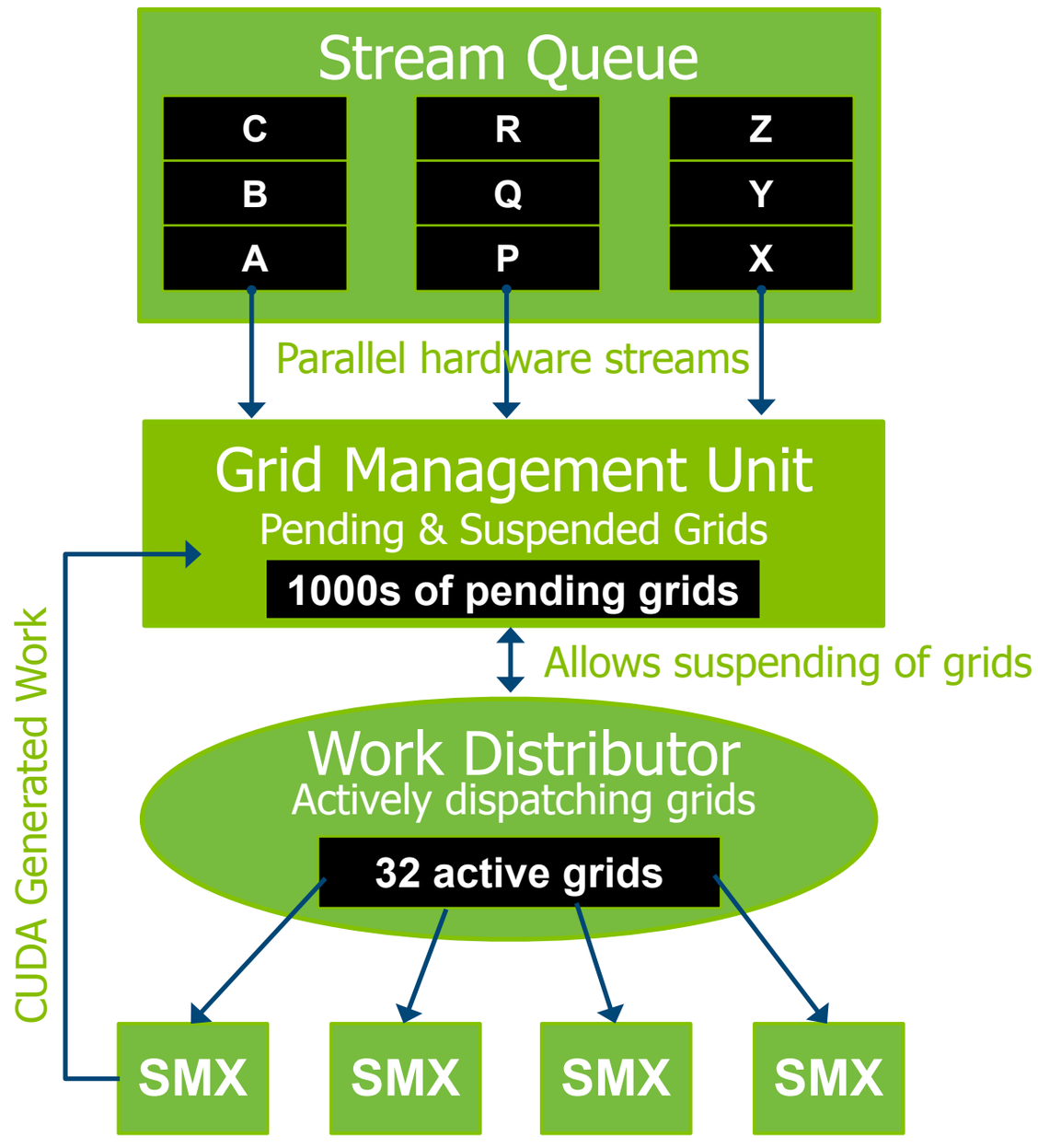
stream_3
kernel_X
kernel_Y
kernel_Z

Grid management unit: Fermi vs. Kepler

Fermi



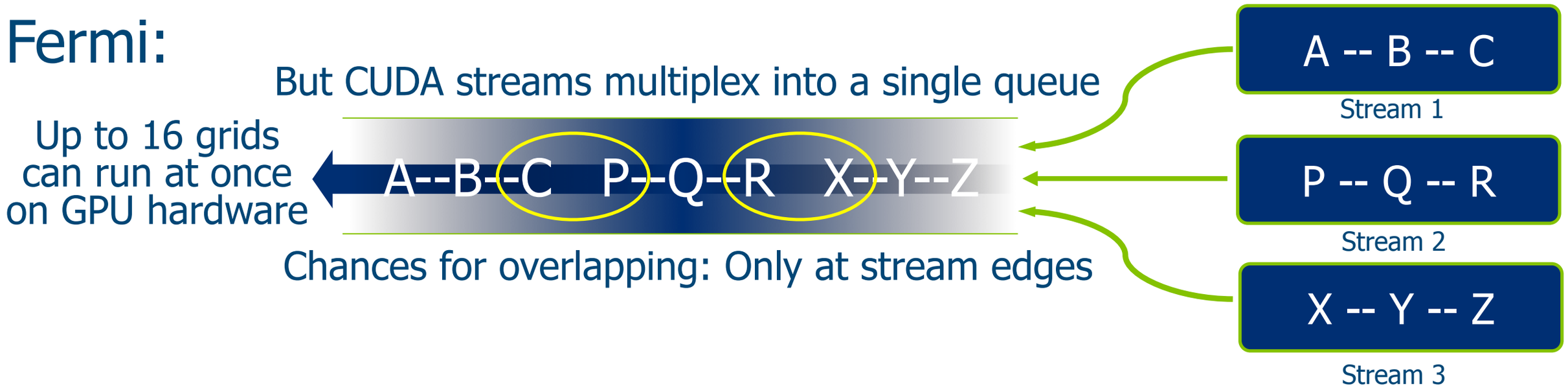
Kepler GK110



54

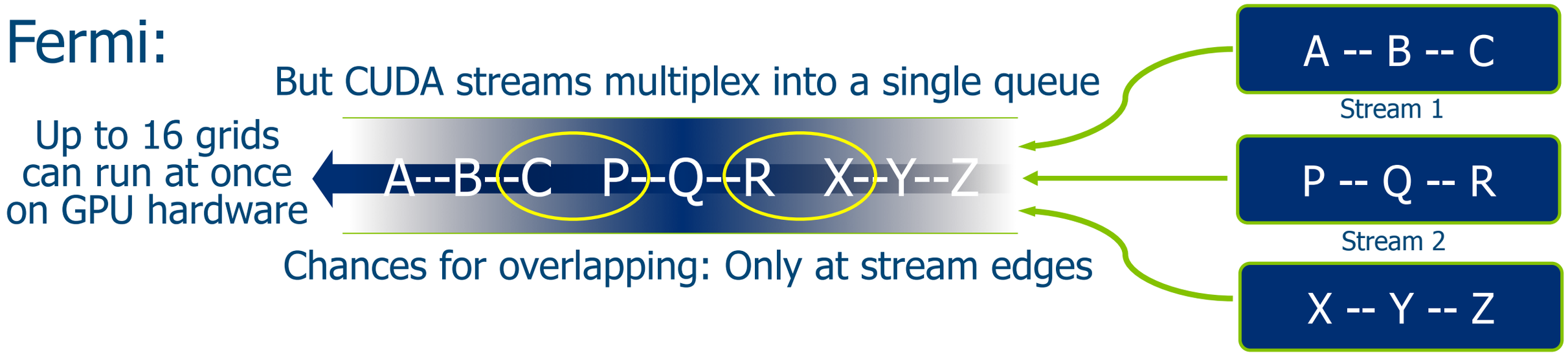
The relation between software and hardware queues

Fermi:

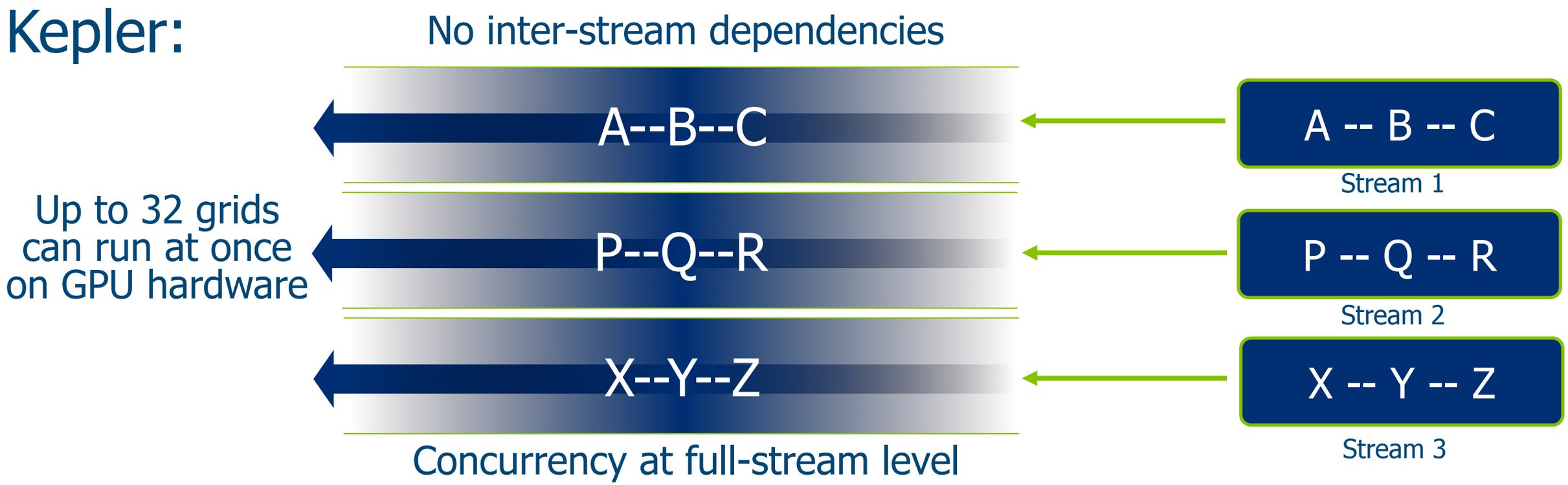


The relation between software and hardware queues

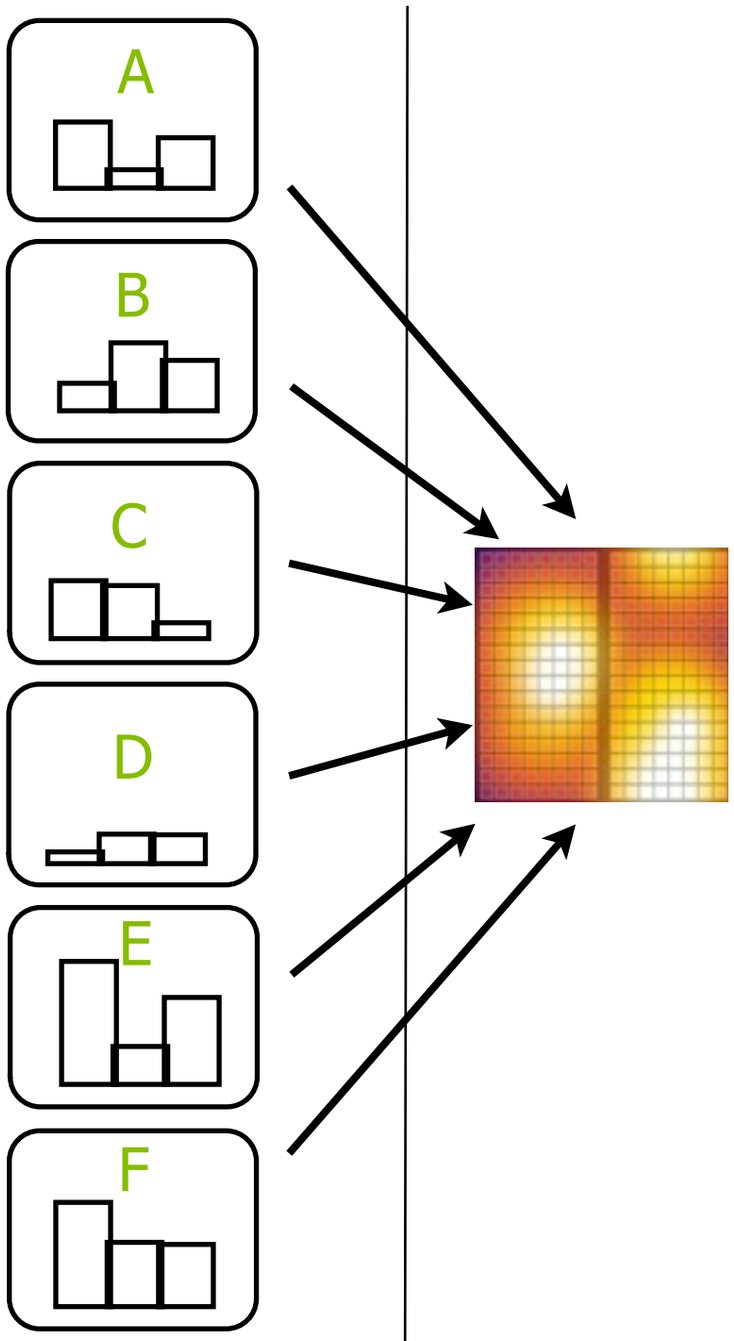
Fermi:

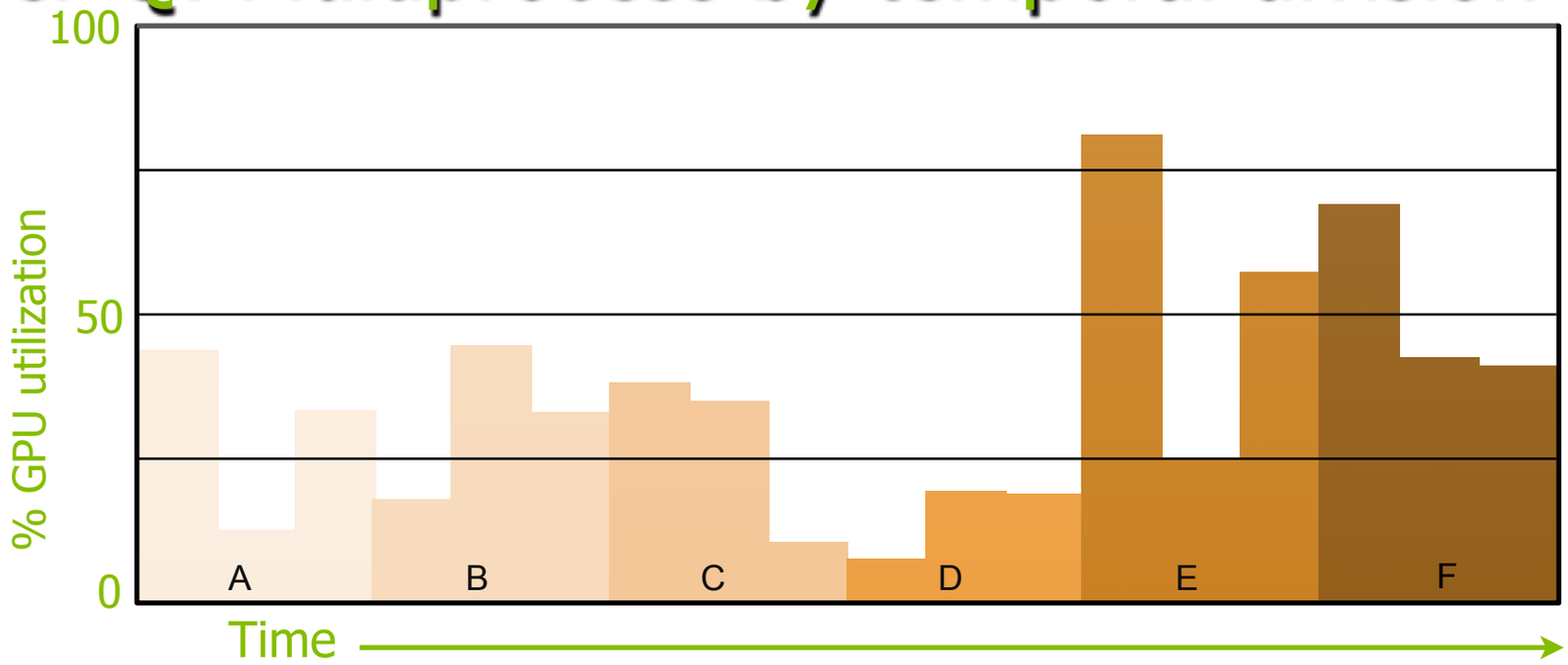


Kepler:



Without Hyper-Q: Multiprocess by temporal division

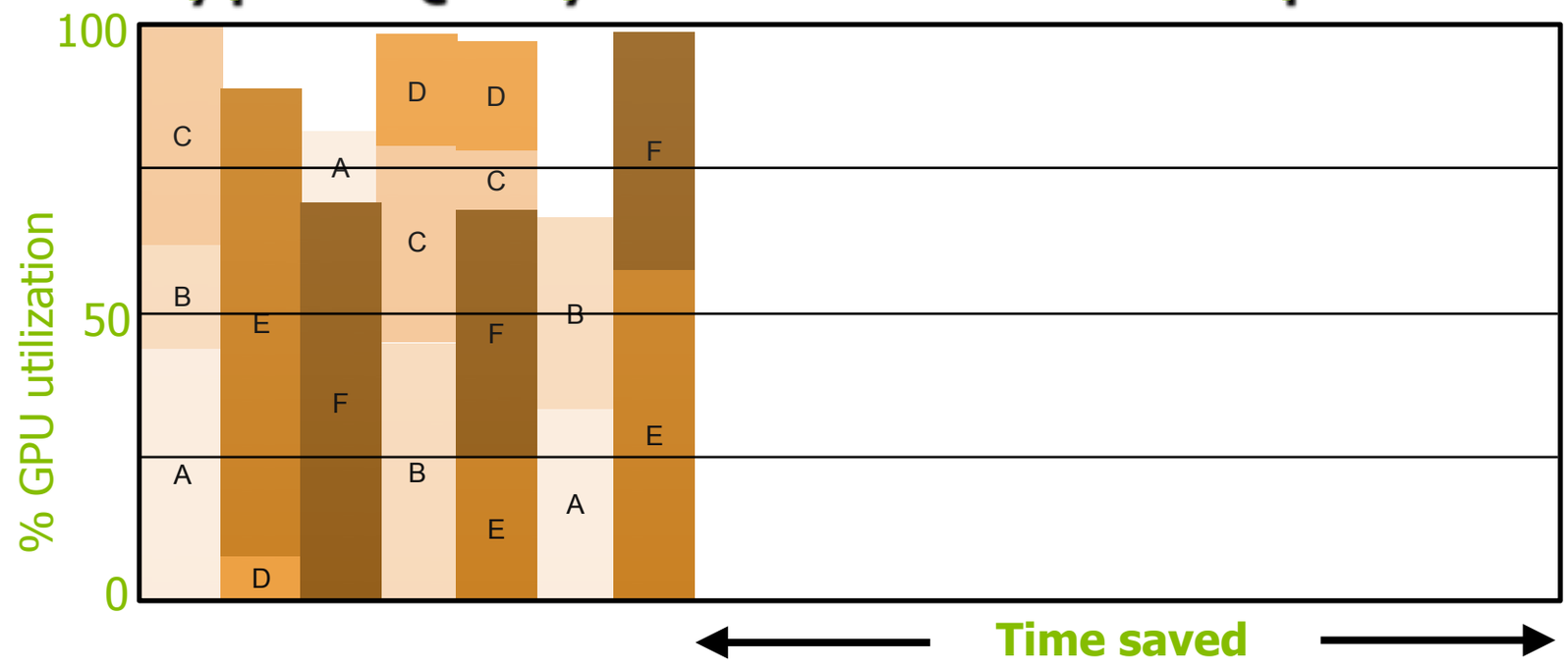




CPU processes...

...mapped on GPU

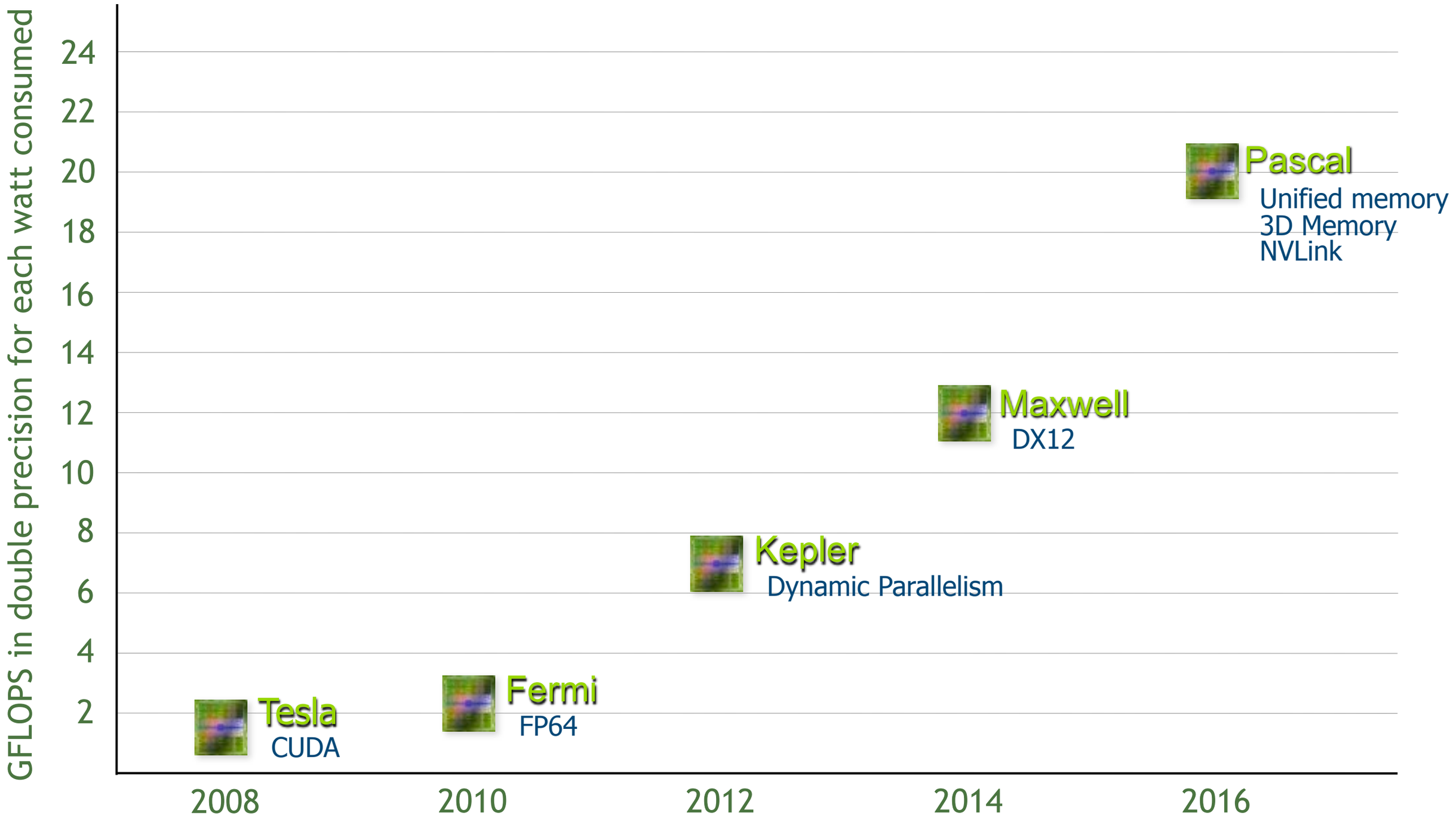
With Hyper-Q: Symultaneous multiprocess



6. A look-ahead to next generations

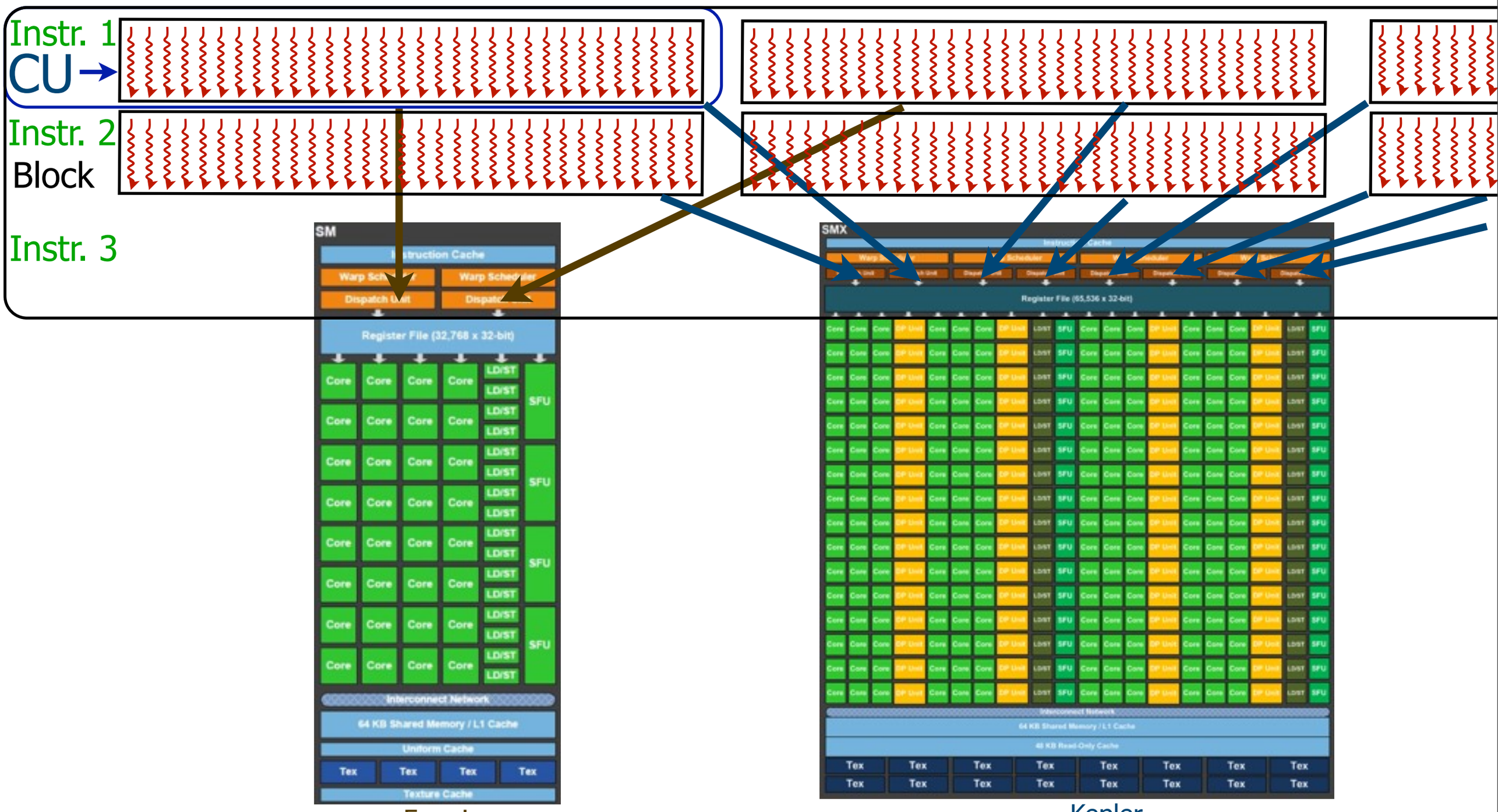
NVIDIA.

Overview of CUDA hardware generations



6.1. The warp size

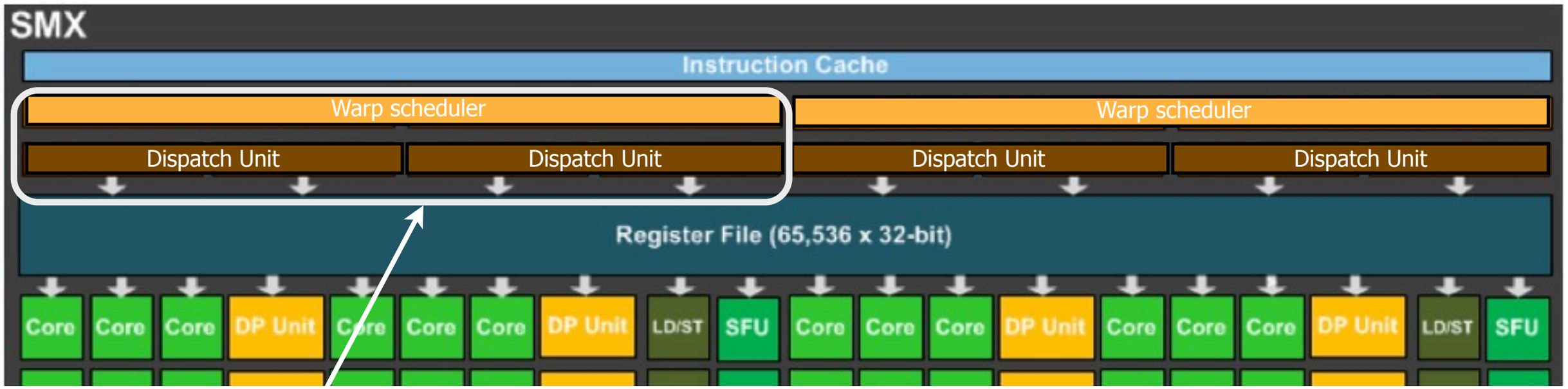
The way each multiprocessor swallows SIMD instructions

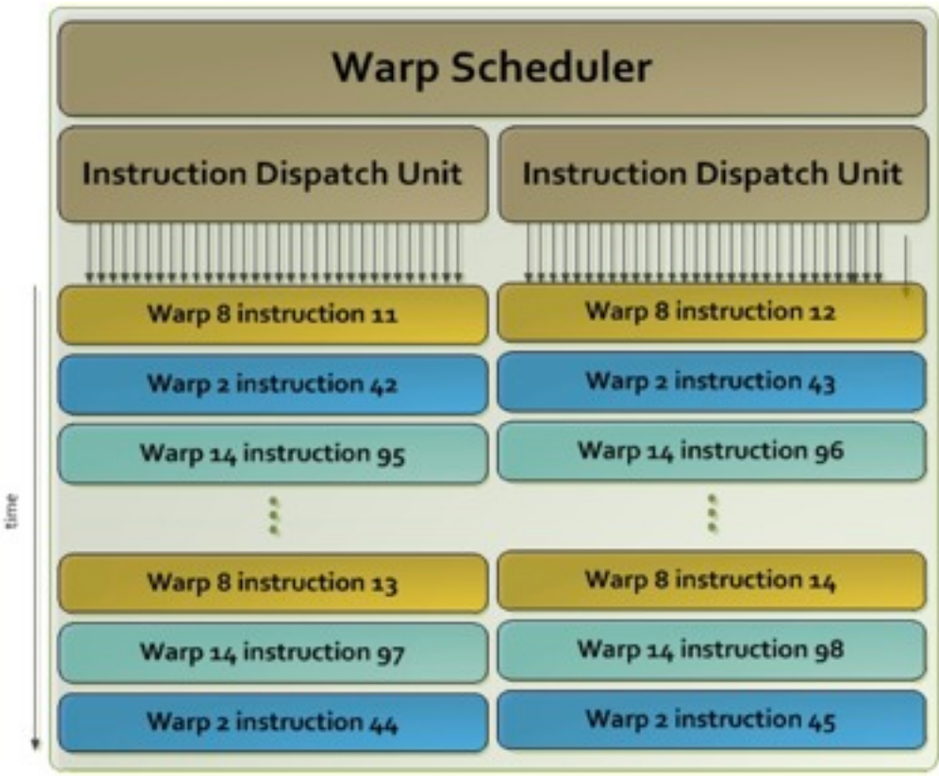


Fermi

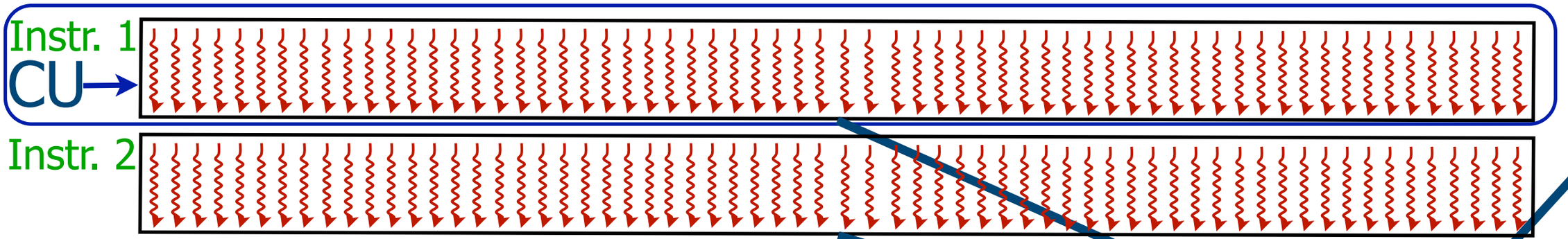
Kepler

A hypothetical GPU front-end with the warp size increased to 64

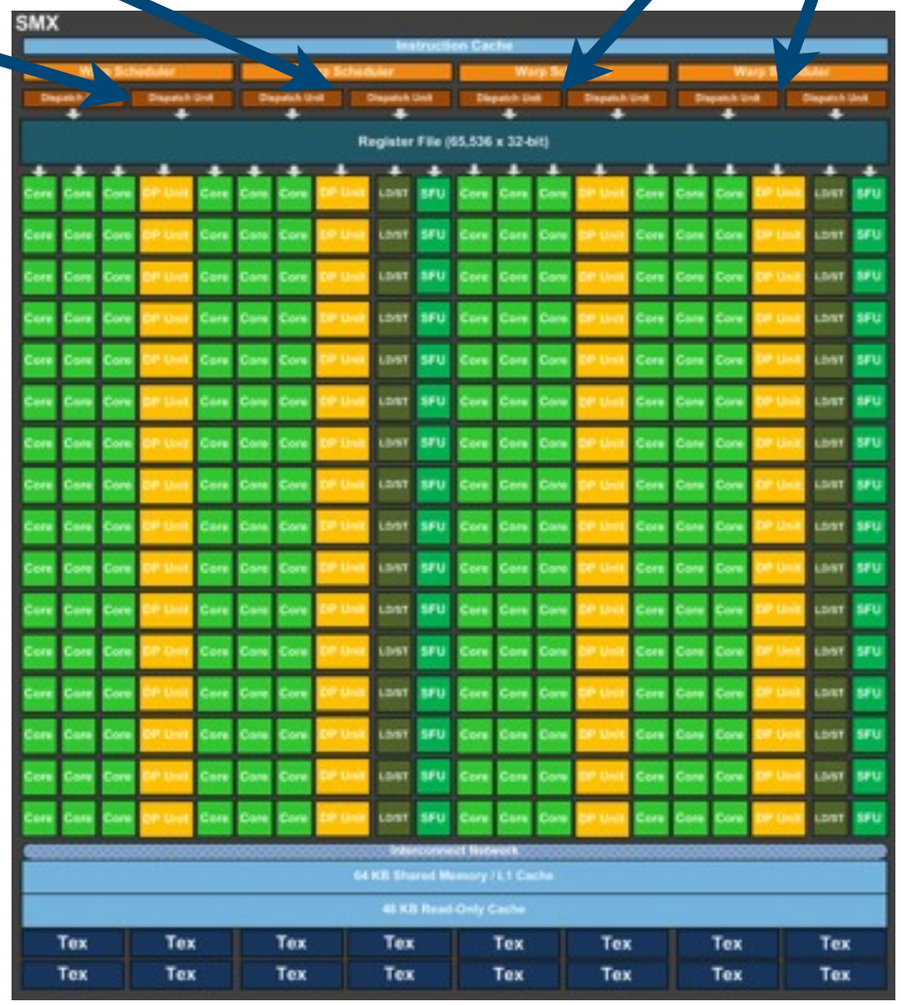




The way each multiprocessor would swallow SIMD instructions using a warp size of 64

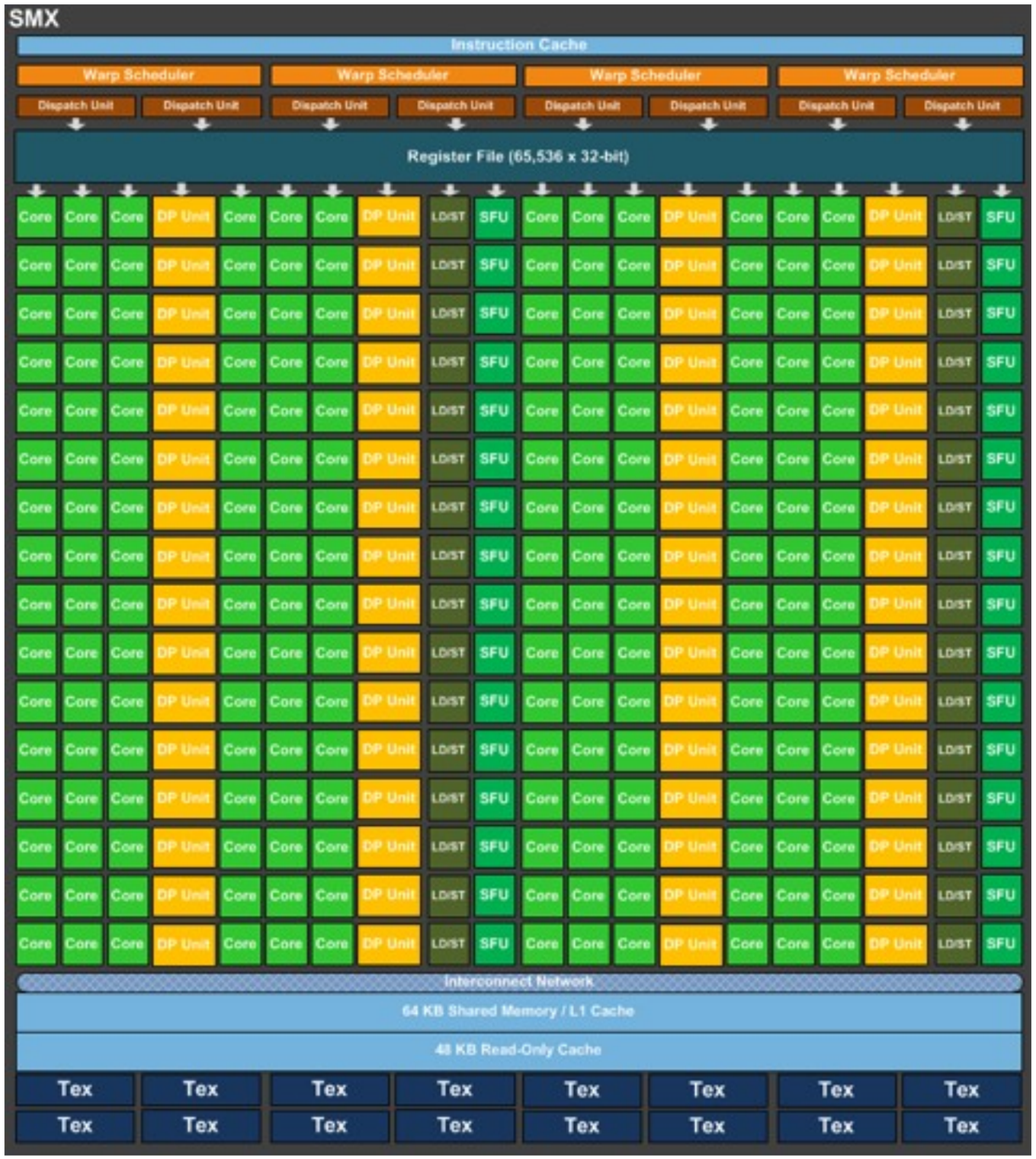


- The cost for the control unit is just the half.
- The penalty due to data dependencies is potentially lower, and the hardware is more simple.
- The penalty due to control dependencies is **higher**.



Kepler

The GPU back-end: Transforming the SMX for a warp size of 64



Functional Unit	#	warp size = 32	warp size = 64
int/fp32 Core	192	6	3
fp64 DP Unit	64	2	1
load/store LD/ST	32	1	1/2
SFU SFU	32	1	1/2

- The deficit lies in load/store and SFUs, but they were facing a tougher constraint during the Fermi generation, and they were able to recover from that.

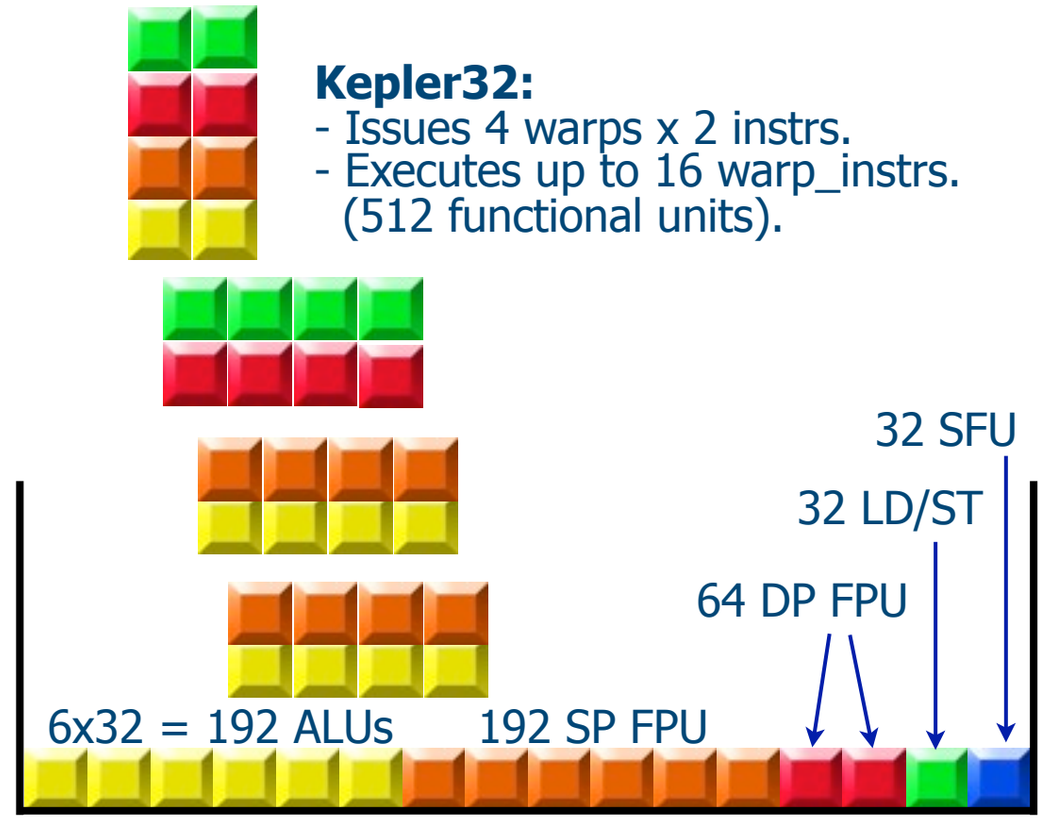
Other facts promoting the warp size to 64

- **Shared memory:** Concurrency attained through banks, and they were already increased from 16 (pre-Fermi) to 32.
- **Device memory:** Higher data bandwidth is required, but that is not the problem in the DDR saga (latency is).
- **Branching:** Techniques minimizing penalties on divergent branches are more mature and ready to face the challenge.
- **Scalability** in the number of cores: Simplicity in the control unit would allow to increase cores of every kind.
- Nvidia is anticipating this move with a warning.
- Other vendors are moving in the same direction:
  - Graphics Core Next (GCN) from AMD is a 4 x 16-wide vector SIMD.

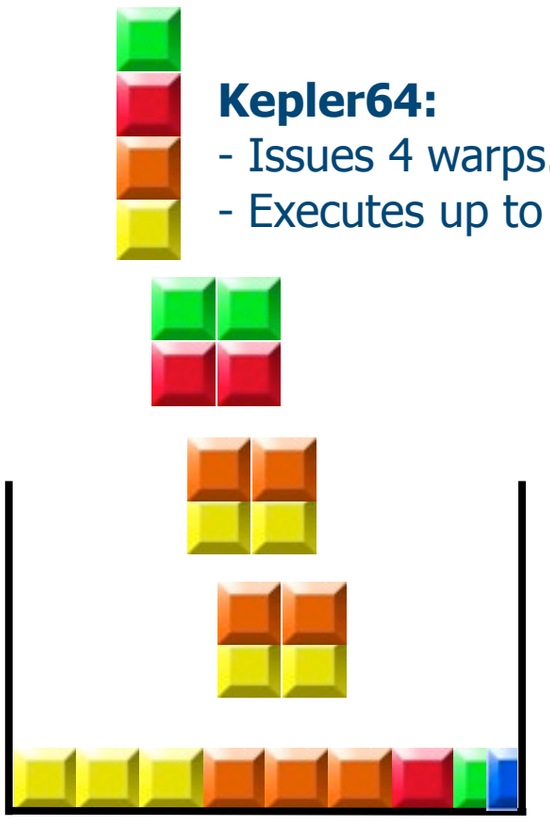
To benefit from this technological change

- Make blocks bigger:
  - Less than 64 threads per block is forbidden.
  - 256 would now be the minimum required.
  - 384 gains momentum.
- Pay more attention to warp divergencies.
- Advantageous for regular computations. Sophistication of hardware scheduler (Hyper-Q, dynamic parallelism) lifts irregular applications.

How it would be Kepler with a warp size of 64

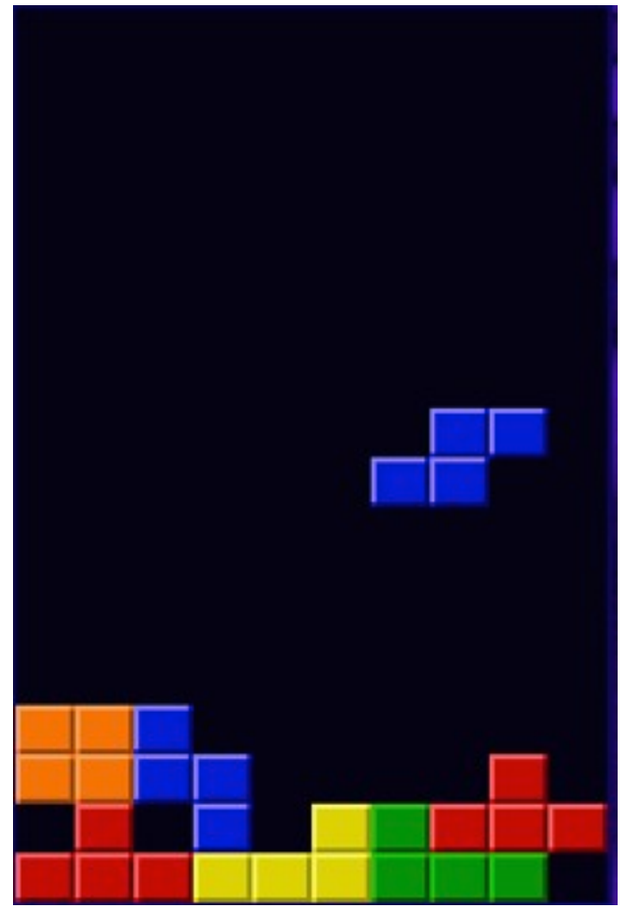


Kepler32:
- Issues 4 warps x 2 instrs.
- Executes up to 16 warp_instrs. (512 functional units).



Kepler64:
- Issues 4 warps.
- Executes up to 8.

SMX in Kepler: 512 parallel functional units

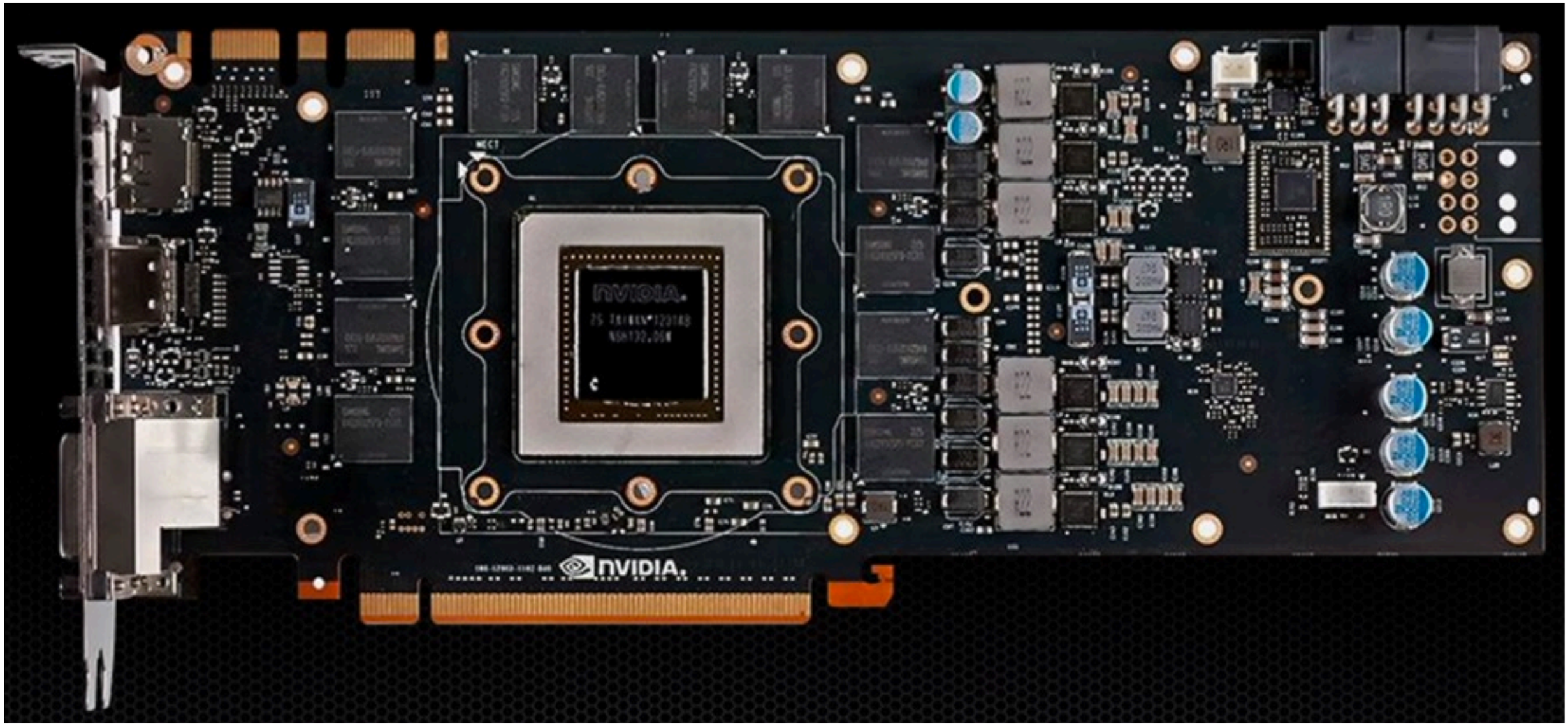


- If we take for granted that Nvidia uses to "complete" to a warps enteros las Unidades Funcionales en la siguiente generación, verde y azul aumentarían, y el parecido de Kepler64 con el Tetris del video-juego sería asombroso.

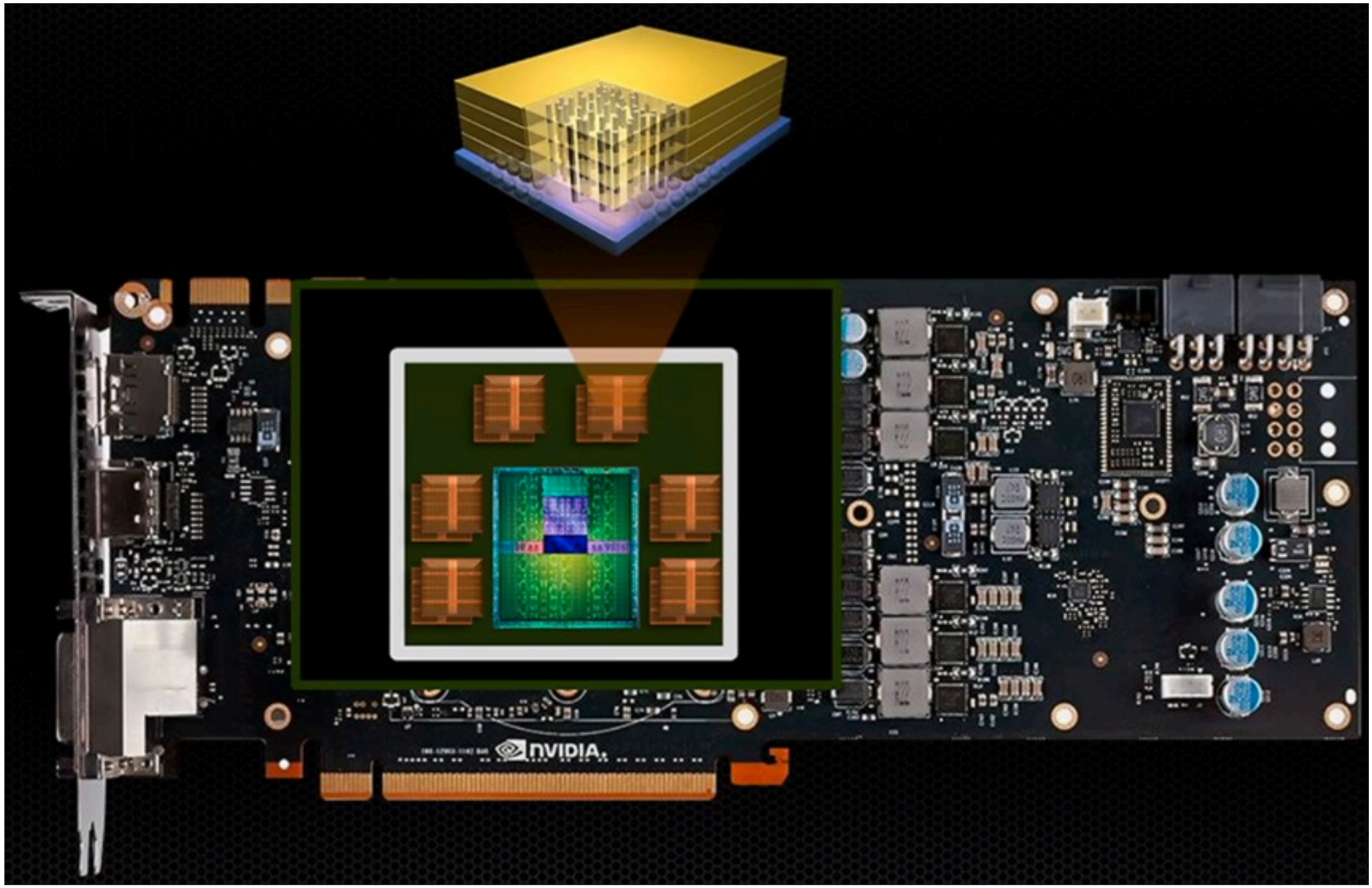
6.2. Stacked (3D) DRAM

NVIDIA

A 2013 graphics card:
Kepler GPU with GDDR5 video memory

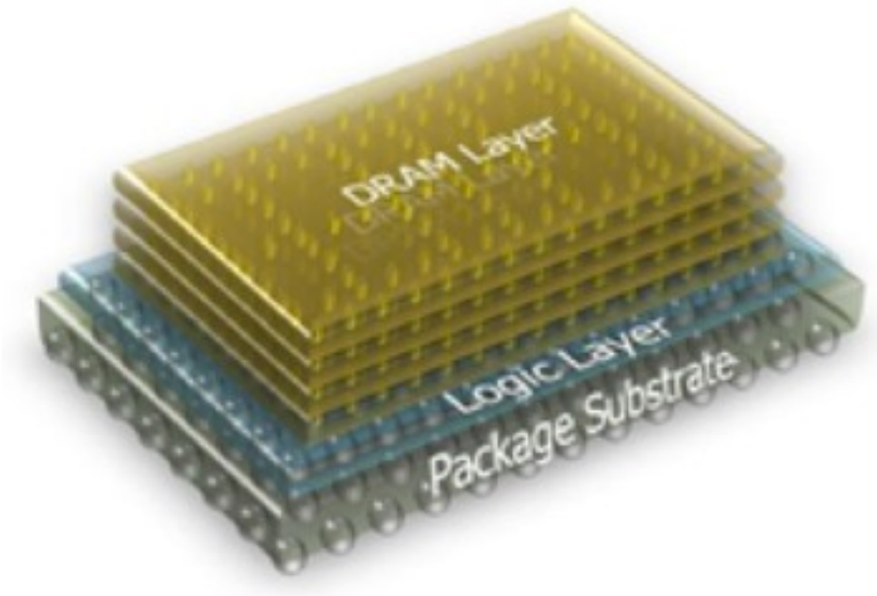


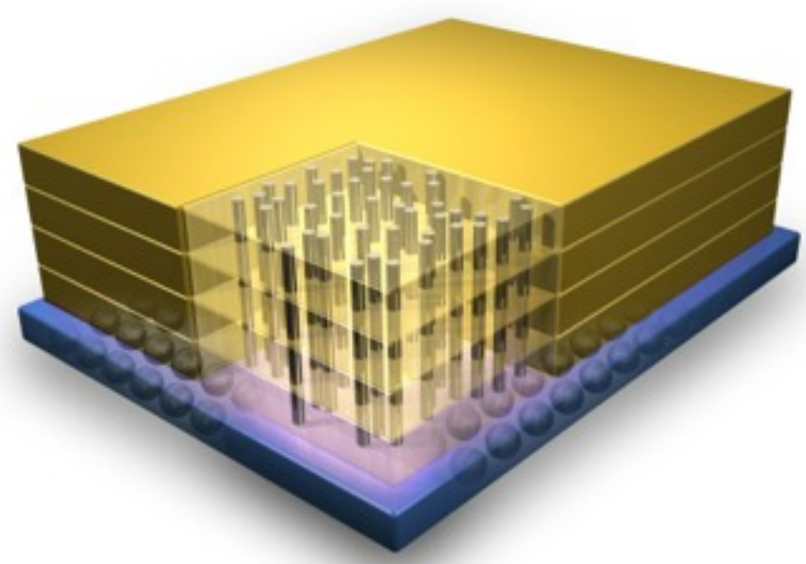
A 2017 graphics card: Pascal GPU with Stacked DRAM



Details on silicon integration

- 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.





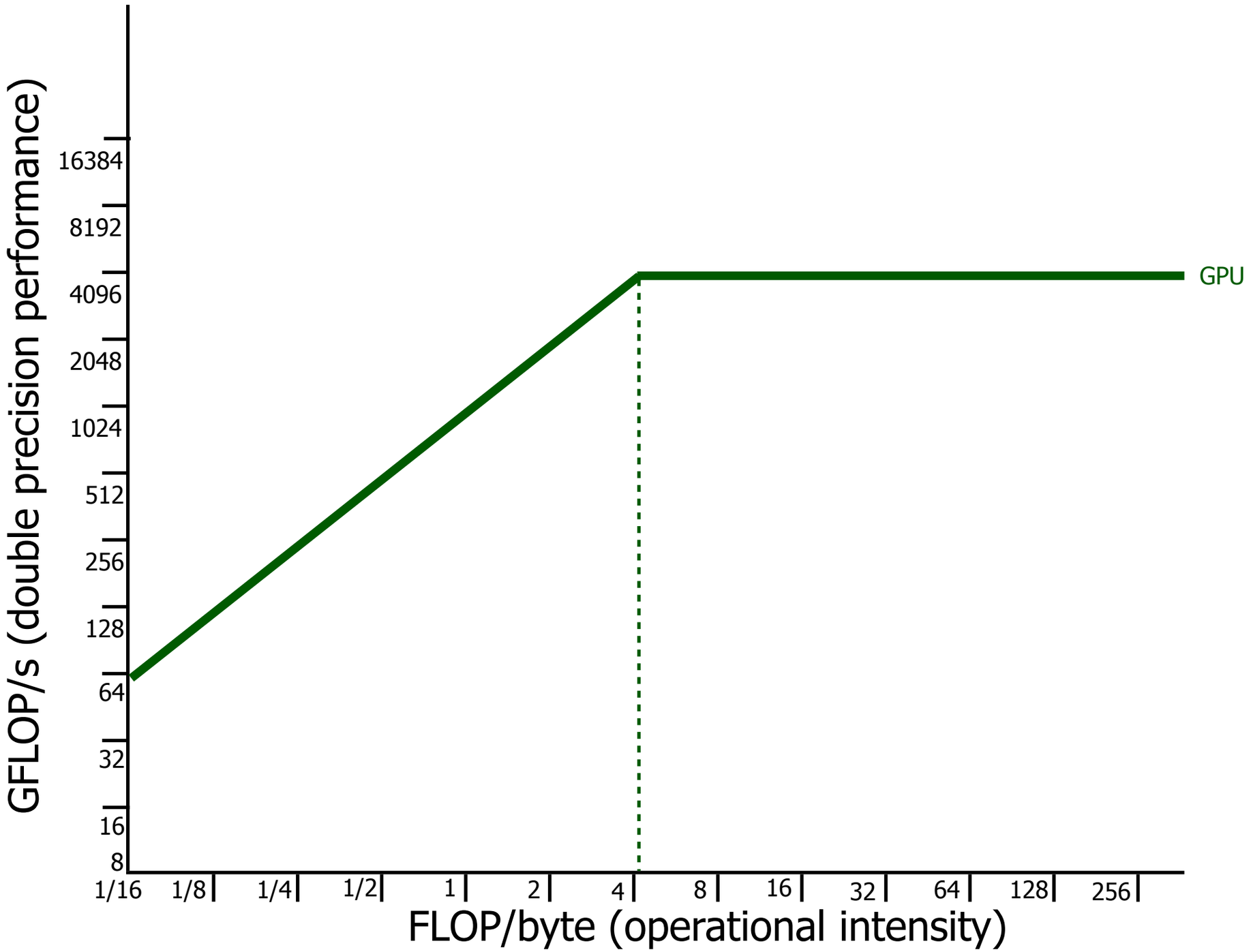
A comparative in bandwidth with existing technologies

- 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.**

(*) Taking the bandwidth estimations given by HMCC 1.0 y 2.0 (20 and 28 GB/s. respectively on each 16-bit link for each direction). Nvidia already confirmed in GTC'13 data bandwidths around 1 TB/s. for its Pascal GPU.

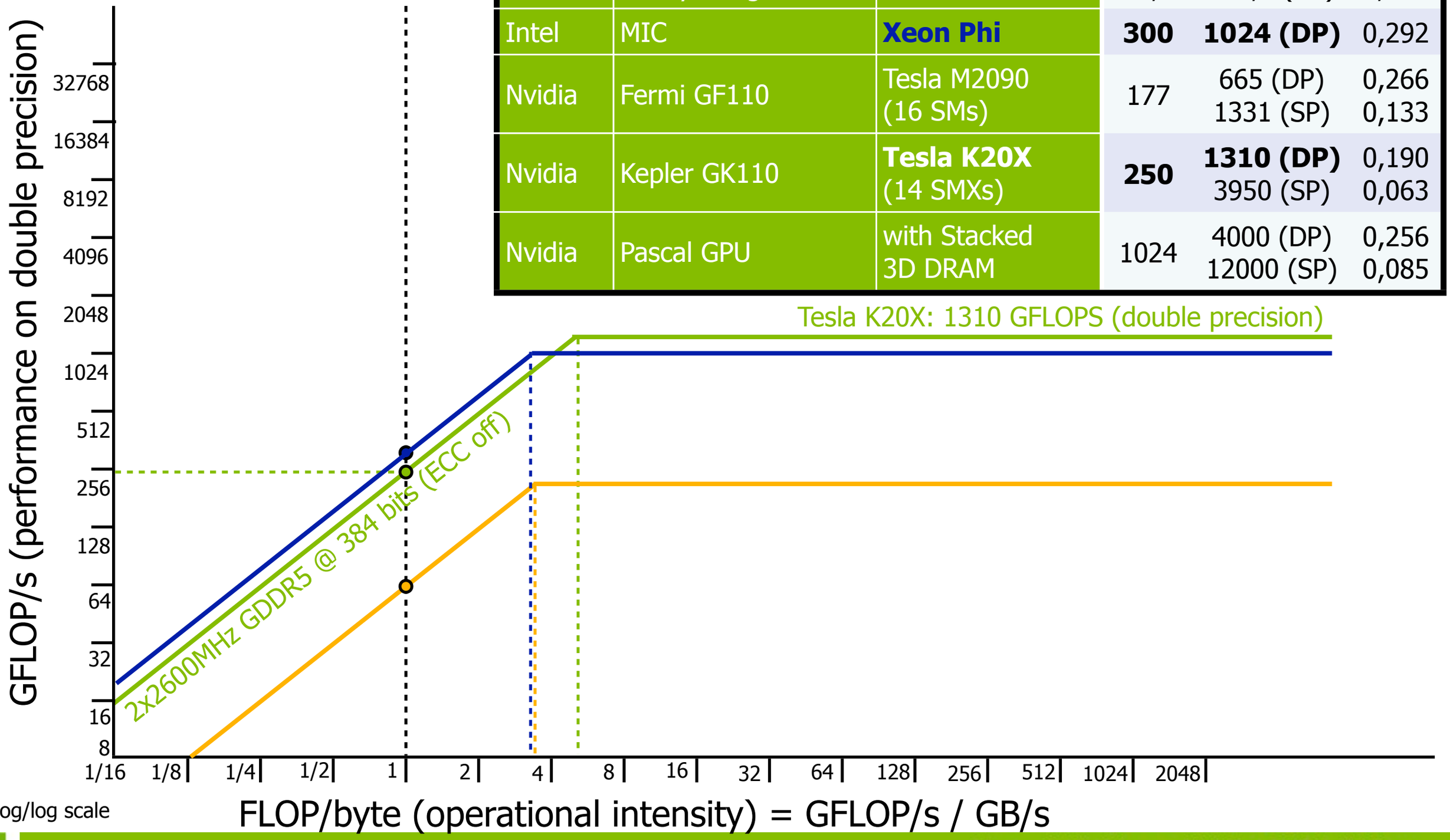
6.3. Analysis based on the roofline model

Impact on GPUs: Analysis based on the roofline model

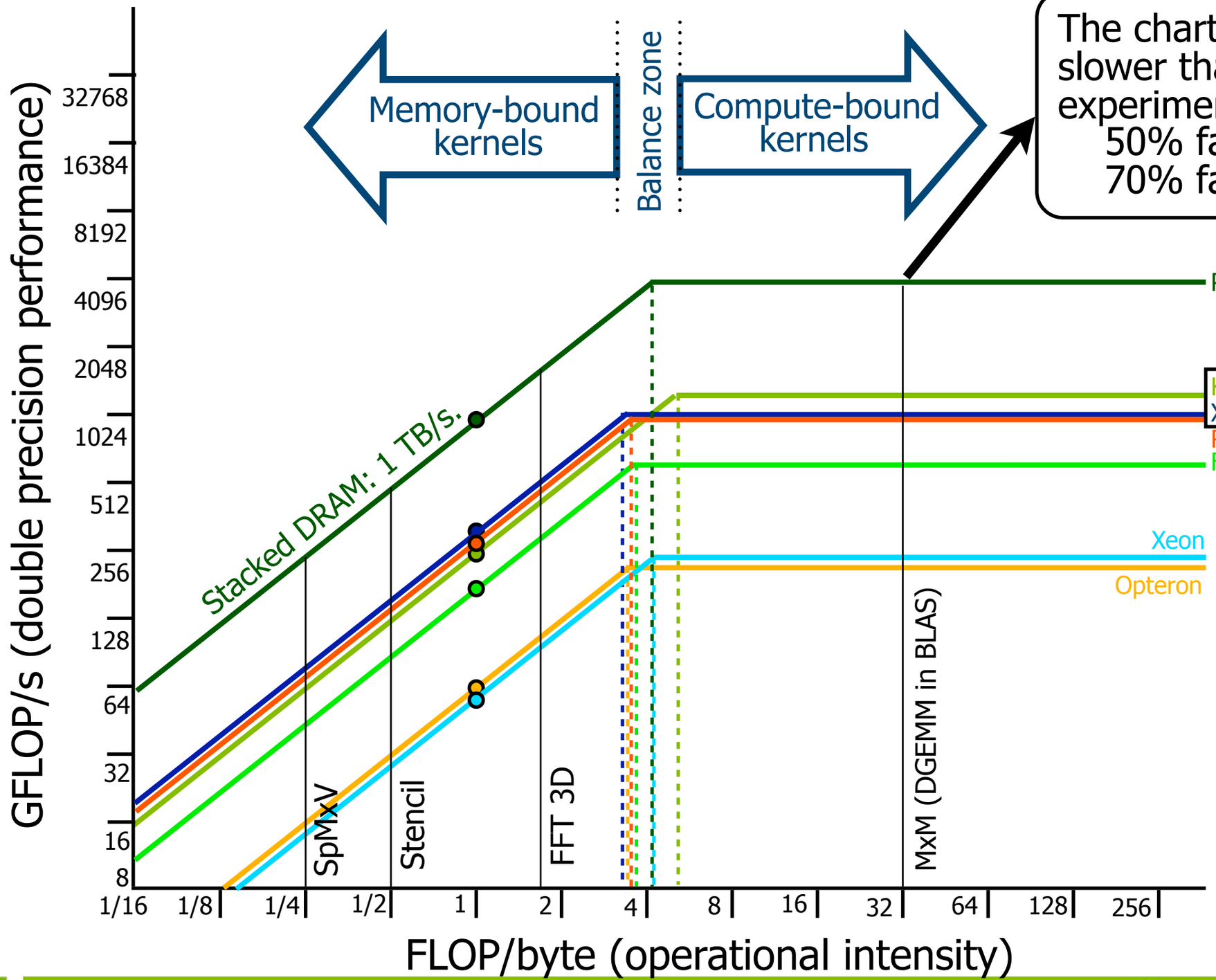


Platforms to compare

Vendor	Microarchitecture	Model	GB/s.	GFLOP/s.	Byte/ FLOP
AMD	Bulldozer	**Opteron 6284**	**59,7**	**217,6 (DP)**	0,235
AMD	Souther Islands	Radeon HD7970	288	1010 (DP)	0,285
Intel	Sandy Bridge	Xeon E5-2690	51,2	243,2 (DP)	0,211
Intel	MIC	**Xeon Phi**	**300**	**1024 (DP)**	0,292
Nvidia	Fermi GF110	Tesla M2090 (16 SMs)	177	665 (DP) 1331 (SP)	0,266 0,133
Nvidia	Kepler GK110	**Tesla K20X** (14 SMXs)	**250**	**1310 (DP)** 3950 (SP)	0,190 0,063
Nvidia	Pascal GPU	with Stacked 3D DRAM	1024	4000 (DP) 12000 (SP)	0,256 0,085



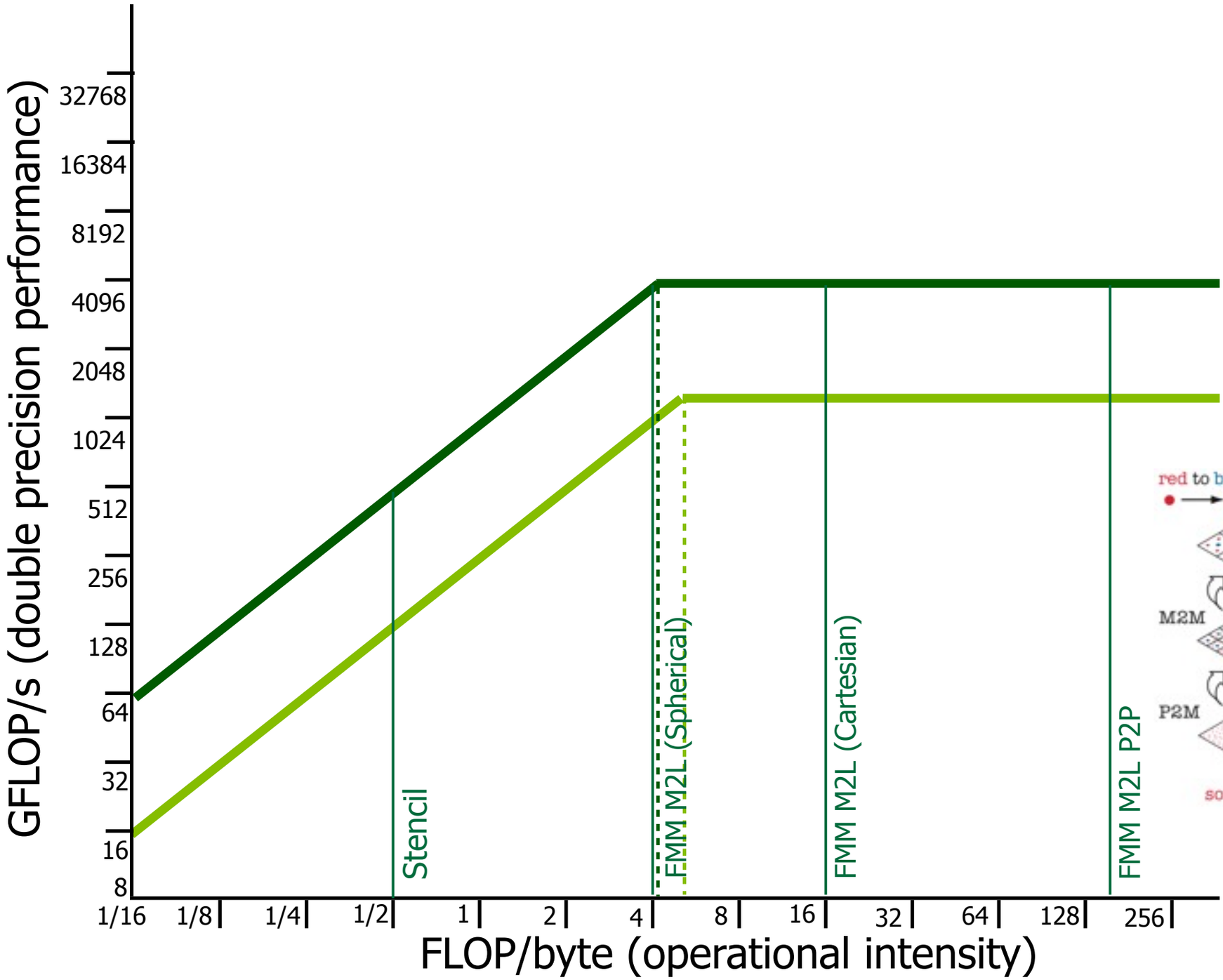
The Roofline model: Hardware vs. Software

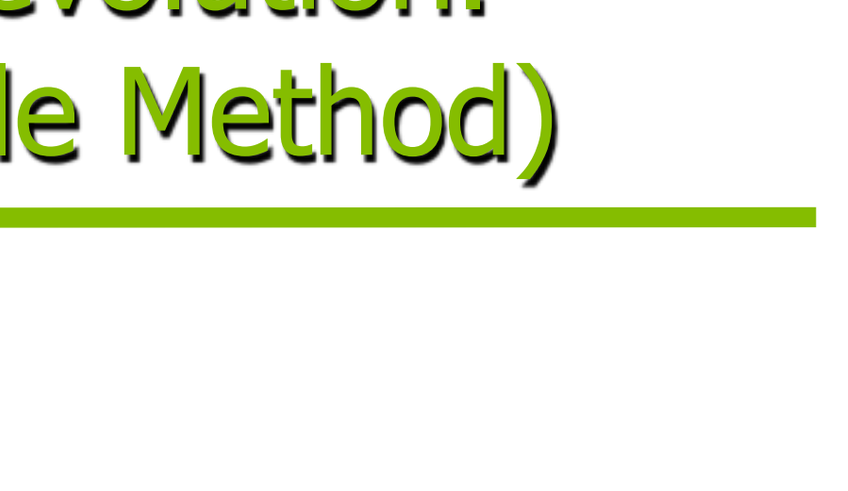


The chart places Xeon Phi 225 as 30% slower than K20X on DGEMM, but our experimental runs say that K20X is:
50% faster in double precision.
70% faster in single precision.

Processor	GB/s.	GFLOP/s.	B/FLOP
Opteron	60	217 (DP)	0,235
Radeon	288	1010 (DP)	0,285
Xeon	51	243 (DP)	0,211
Xeon Phi	300	1024 (DP)	0,292
Fermi	177	665 (DP) 1331 (SP)	0,266 0,133
Kepler	250	1310 (DP) 3950 (SP)	0,190 0,063
Pascal	1024	4000 (DP) 12000 (SP)	0,256 0,085

The Roofline model: Software evolution. Case study: FMM (Fast Multipole Method)





Concluding remarks

- Kepler represents the architectural design for 2013-2014, ready to host thousands of cores on a single die.
- Deploys all types of parallelism: Task (threads), instruction (pipelines), data (SIMD) and vectorial (warps).
- Enhances power consumption and programmability, improving CUDA for irregular and dynamic applications.
- The GPU is more autonomous, but at the same time allows more interaction with the CPU.
- The memory hierarchy improves significantly, as well as the connection among GPUs.
- SMX-DRAM interconnect will be crucial in future designs.

Thanks for coming!

- You can always reach me in Spain at the Computer Architecture Department of the University of Malaga:
  - e-mail: ujaldon@uma.es
  - Phone: +34 952 13 28 24.
  - Web page: http://manuel.ujaldon.es (english/spanish versions available).
- Or, more specifically on GPUs, visit my web page as Nvidia CUDA Fellow:
  - http://research.nvidia.com/users/manuel-ujaldon