



## **Master Informatics Eng.**

2020/21

*A.J.Proença*

**Data Parallelism with GPUs** *(online)*

*(most slides are borrowed)*

# Graphics Processing Units

- Question to GPU architects:
  - *Given the hardware invested to do graphics well, how can we supplement it to improve the performance of a wider range of applications?*
- Key ideas:
  - Heterogeneous execution model
    - CPU is the *host*, GPU is the *device*
  - Develop a C-like programming language for GPU
  - Unify all forms of GPU parallelism as *CUDA\_threads*
  - Programming model follows SIMT:  
“*Single Instruction Multiple Thread*”

## SIMD Parallelism

- Vector architectures
- SIMD & extensions
- Graphics Processor Units (GPUs)



Copyright © 2012, Elsevier Inc. All rights reserved.

3

# # cores/processing elements in several computing devices



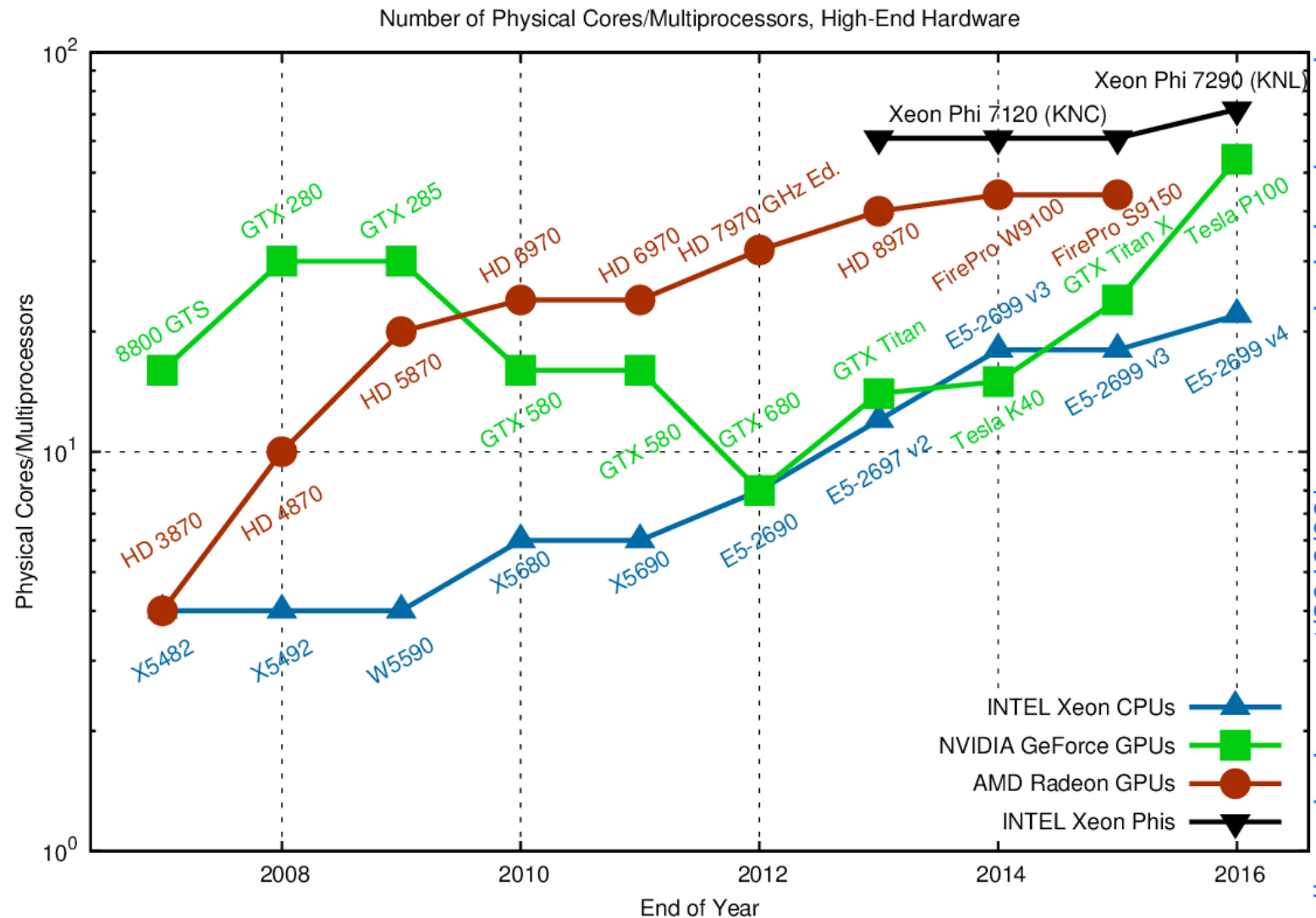
Key question:  
what is a **core**?

a) IU+FPU?  
*GPU-type...*

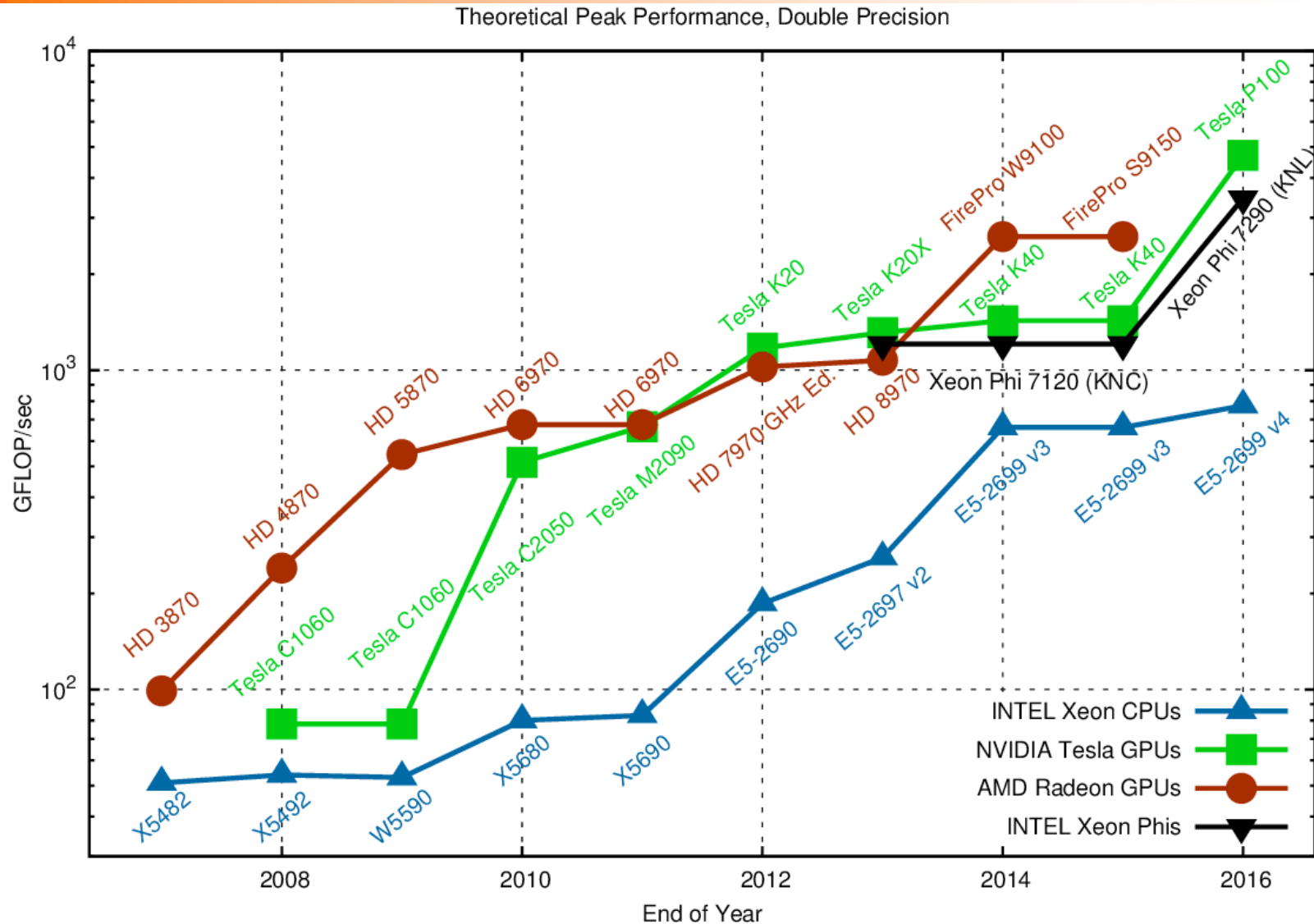
b) A SIMD  
processor?  
*CPU-type..*

This updated slide  
and in this course:  
- **b)**

Note: the web link  
with these plots was  
updated in Aug'16



# Theoretical peak performance in several computing devices (DP)

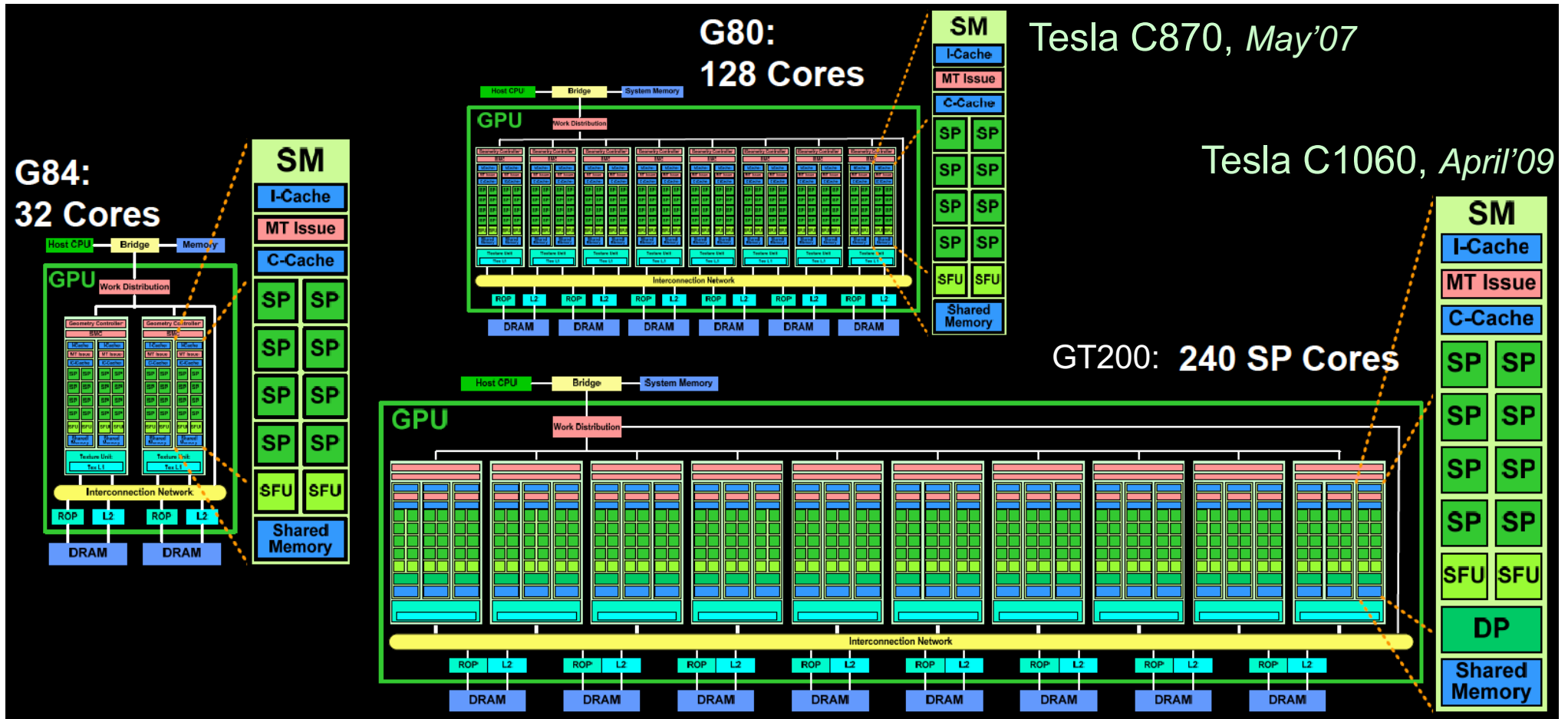




# NVIDIA GPU Architecture

- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files
- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

# Early NVidia GPU Computing Modules

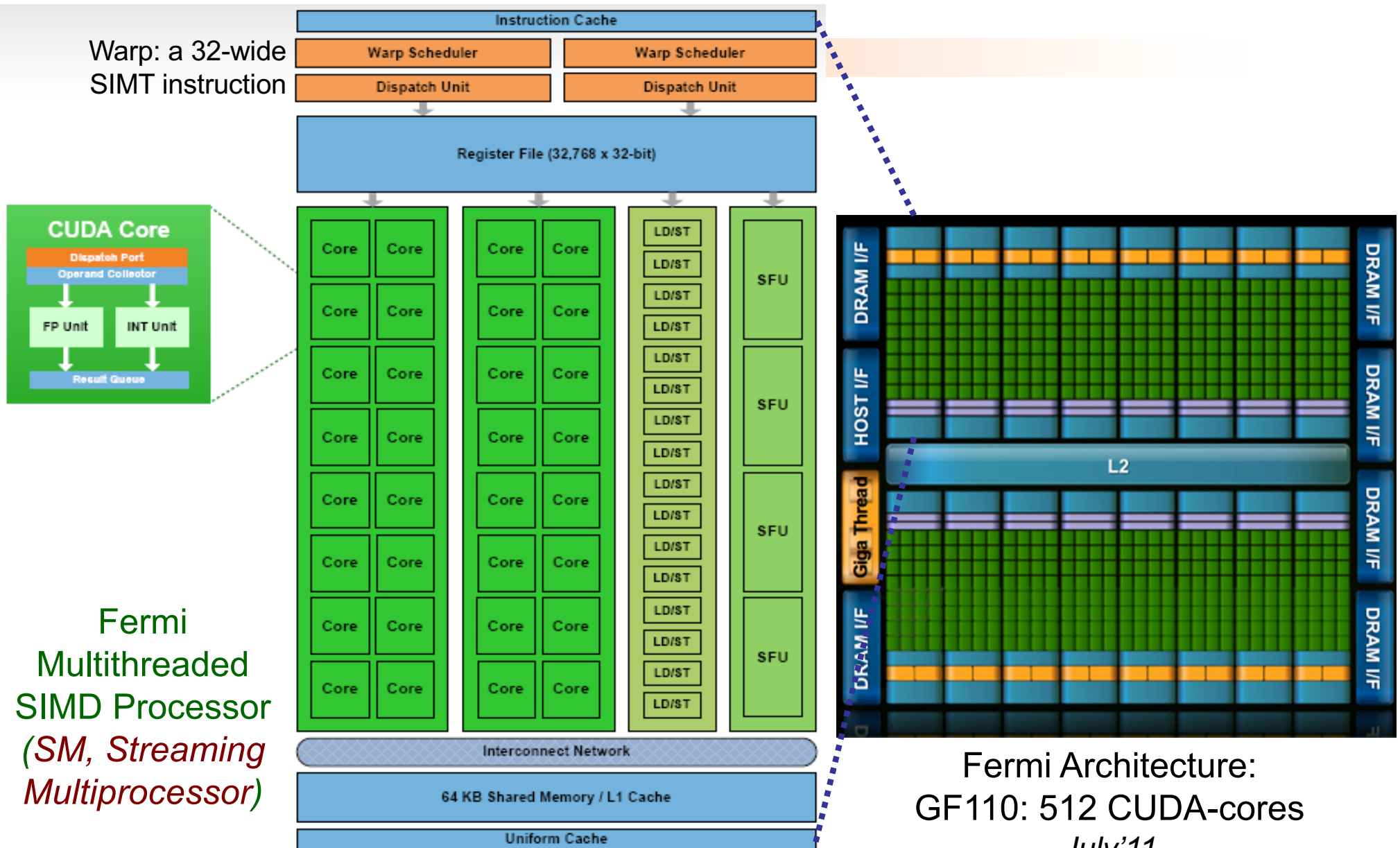


# NVIDIA GPU Memory Structures

- Each SIMD Lane has private section of **off-chip DRAM**
  - “Private memory” (*Local Memory*)
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor (*SM*) also has local memory (*Shared Memory*)
  - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors (*SM*) is GPU Memory, **off-chip DRAM** (*Global Memory*)
  - Host can read and write GPU memory



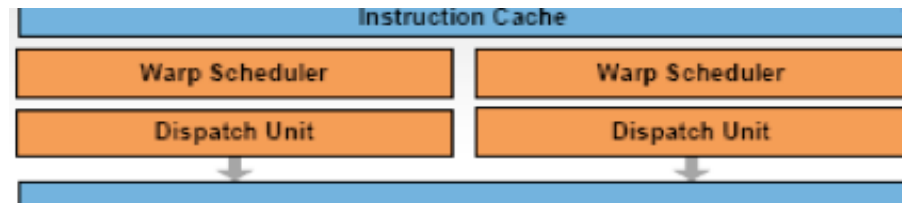
# The NVidia Fermi architecture



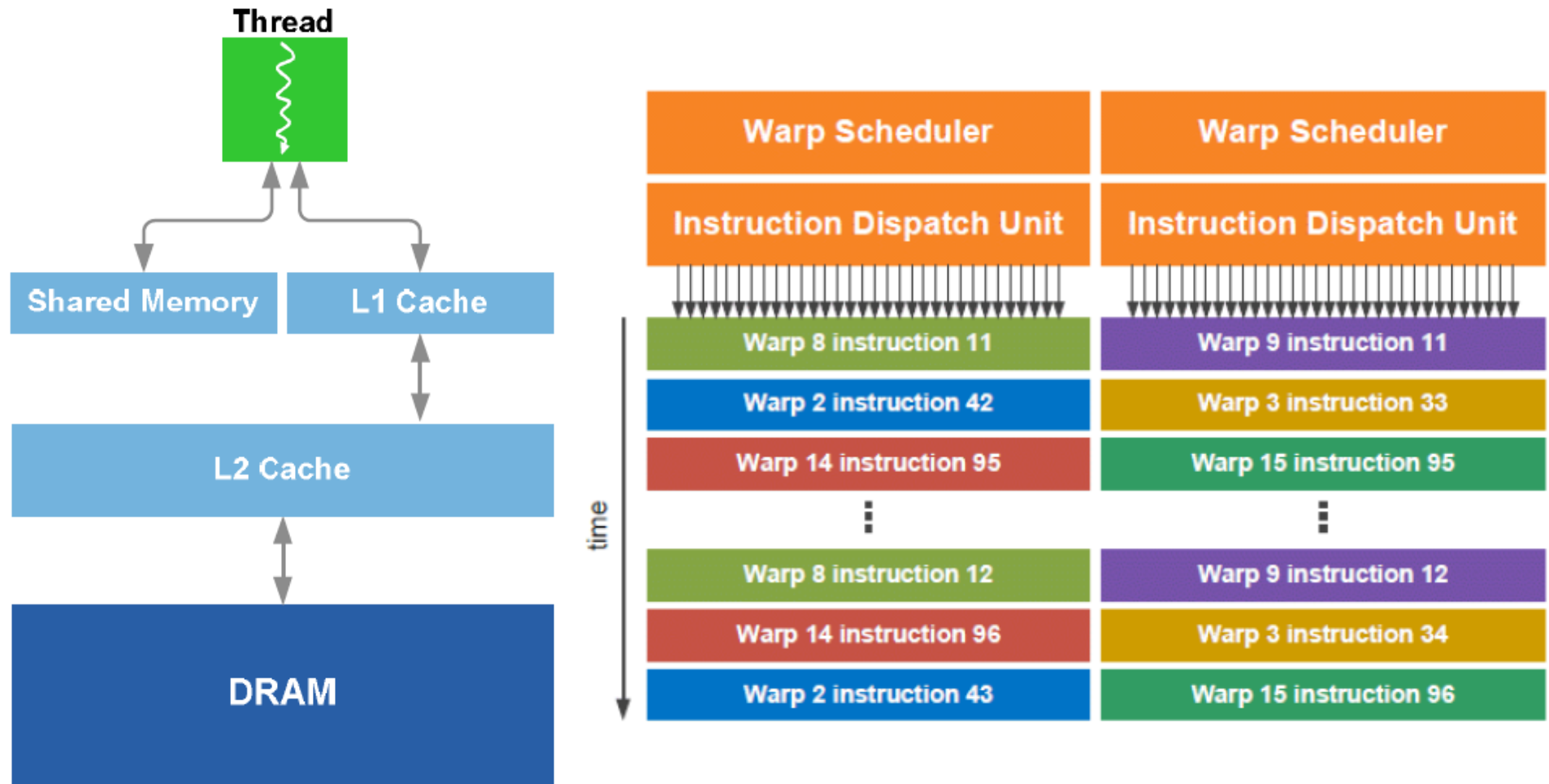
Fermi Architecture:  
GF110: 512 CUDA-cores  
July'11

# Fermi Architecture Innovations

- Each SIMD processor has
  - Two SIMD thread schedulers, two instruction dispatch units
  - 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 4 special function units
  - Thus, two threads of SIMD instructions are scheduled every two clock cycles
- Fast double precision
- Caches for GPU memory (16/64KiB\_L1/SM and global 768KiB\_L2)
- 64-bit addressing and unified address space
- Error correcting codes
- Faster context switching
- Faster atomic instructions



# Fermi: Multithreading and Memory Hierarchy



# ***TOP500 list in November 2010: 3 systems in the top4 use Fermi GPUs***



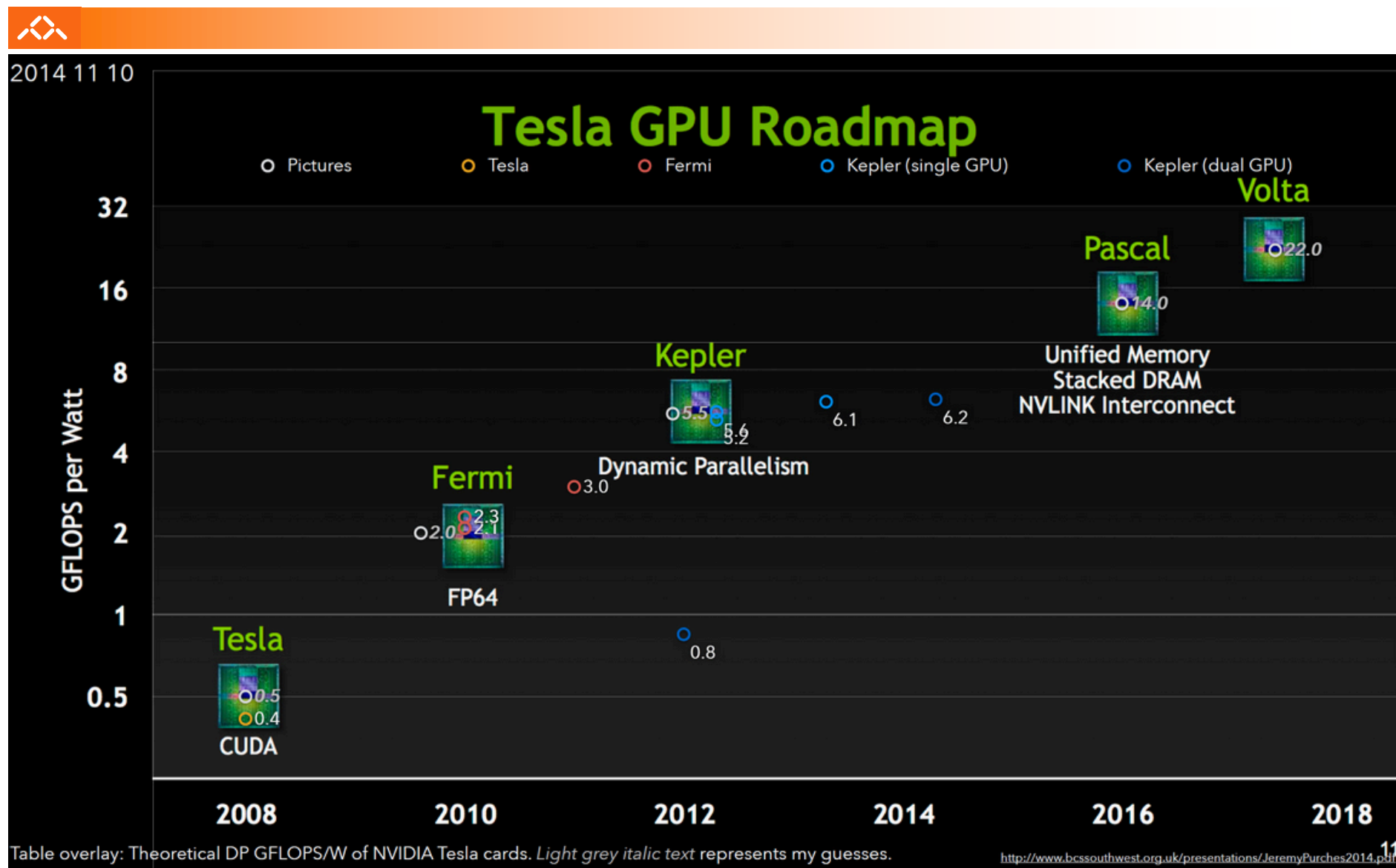
## **HIGHLIGHTS: NOVEMBER 2010**

- The Chinese Tianhe-1A system is the new No. 1 on the TOP500 and clearly in the lead with 2.57 petaflop/s performance.
- No. 3 is also a Chinese system called Nebulae, built from a Dawning TC3600 Blade system with Intel X5650 processors and NVIDIA Tesla C2050 GPUs
- There are seven petaflop/s systems in the TOP10
- The U.S. is tops in petaflop/s with three systems performing at the petaflop/s level
- The two Chinese systems and the new Japanese Tsubame 2.0 system at No. 4 are all using NVIDIA GPUs to accelerate computation and a total of 28 systems on the list are using GPU technology.



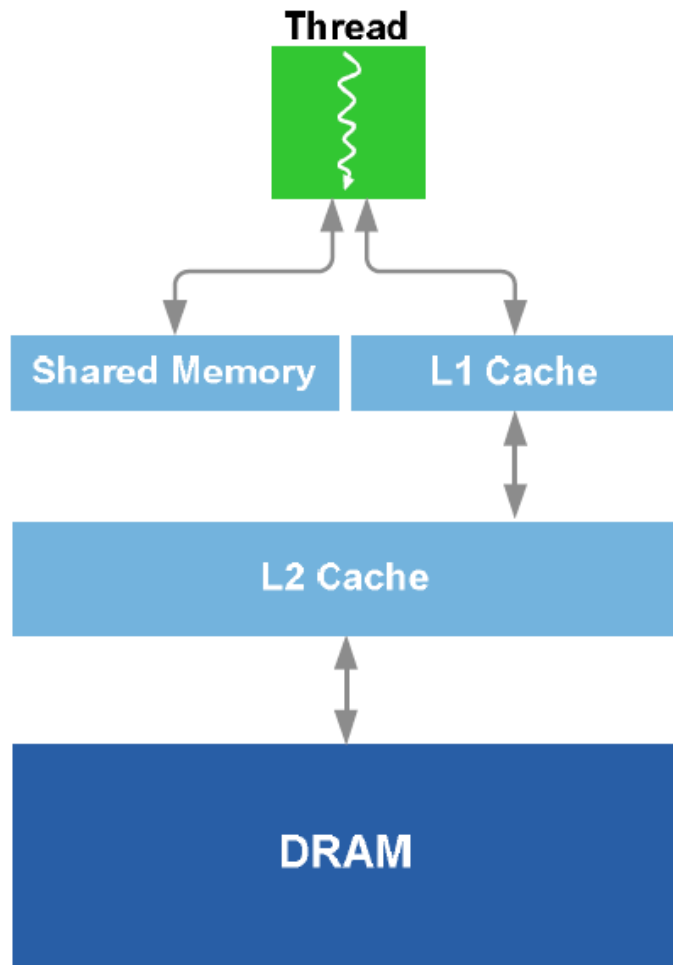
# Families in NVidia Tesla GPUs

(up to 2018)

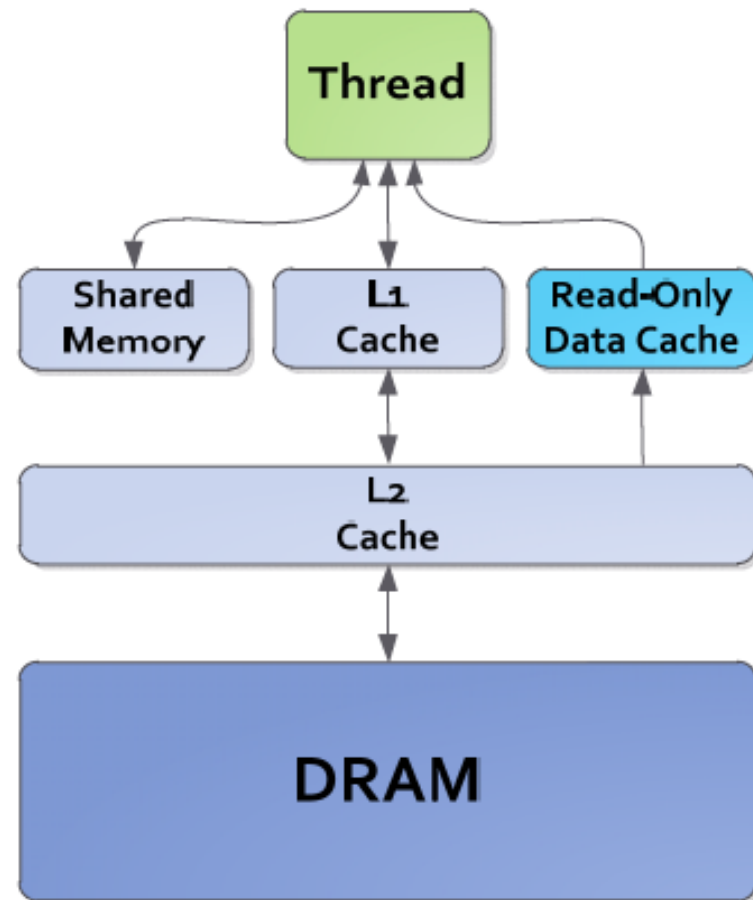




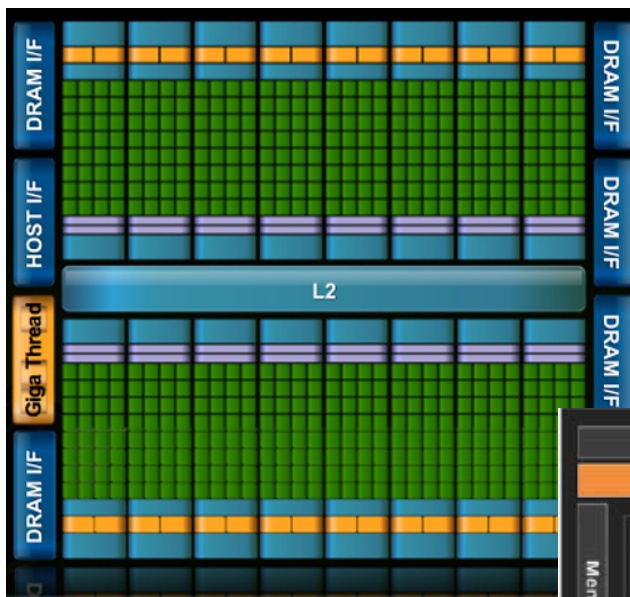
# From Fermi into Kepler: the Memory Hierarchy



Kepler Memory Hierarchy



# From the GF110 to the GK110 Kepler Architecture



Fermi:  
16 SM  
512 CUDA-cores  
July'11



Kepler:  
15 SMX  
2880 CUDA-cores  
October'13

# From Fermi to Kepler core: SM and the SMX Architecture



Fermi SM

SMX:  
192 CUDA-cores

Ratio DPunit : SPunit → 1 : 3

AJProença, Parallel Computing, MiEI, UMinho, 2011



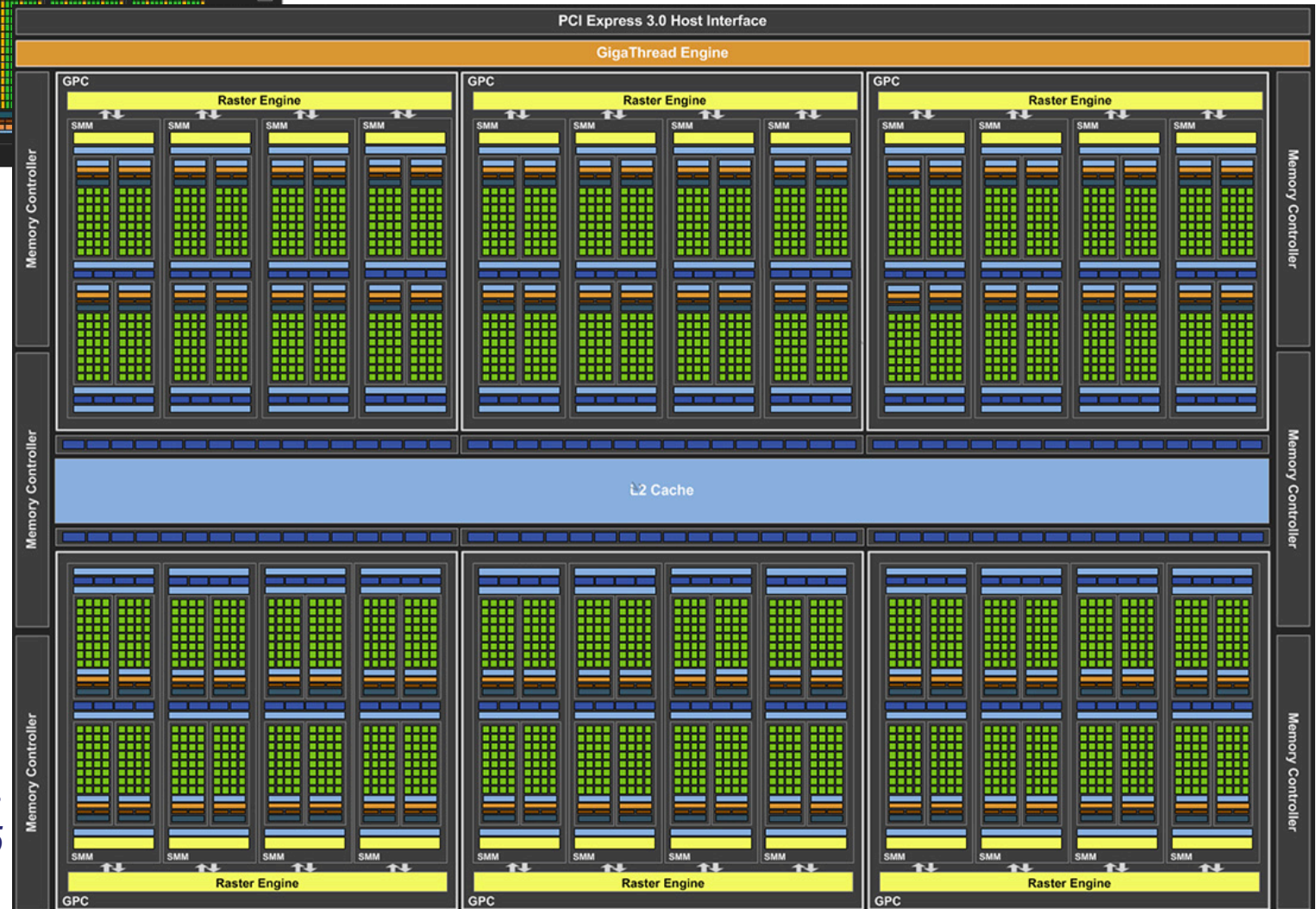


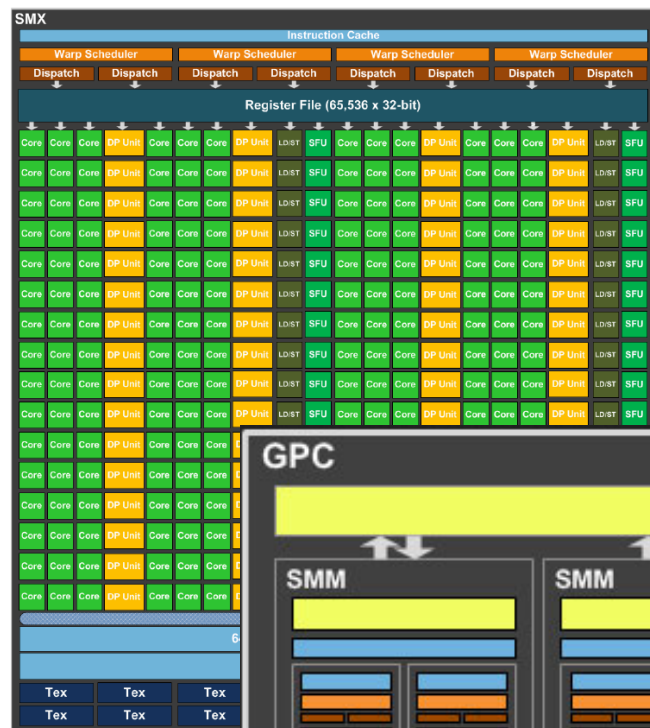
# From the GK110 to the GM200 Maxwell Architecture



Kepler:  
15 SMX  
2880 CUDA-cores  
October'13

Maxwell:  
3072 CUDA-cores  
November'15

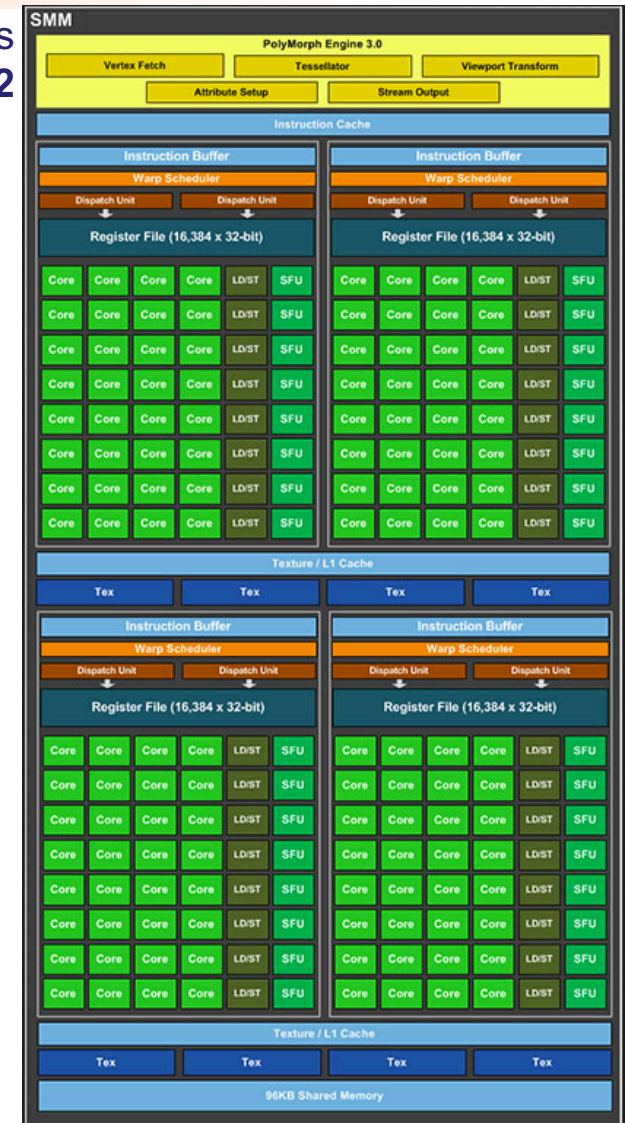
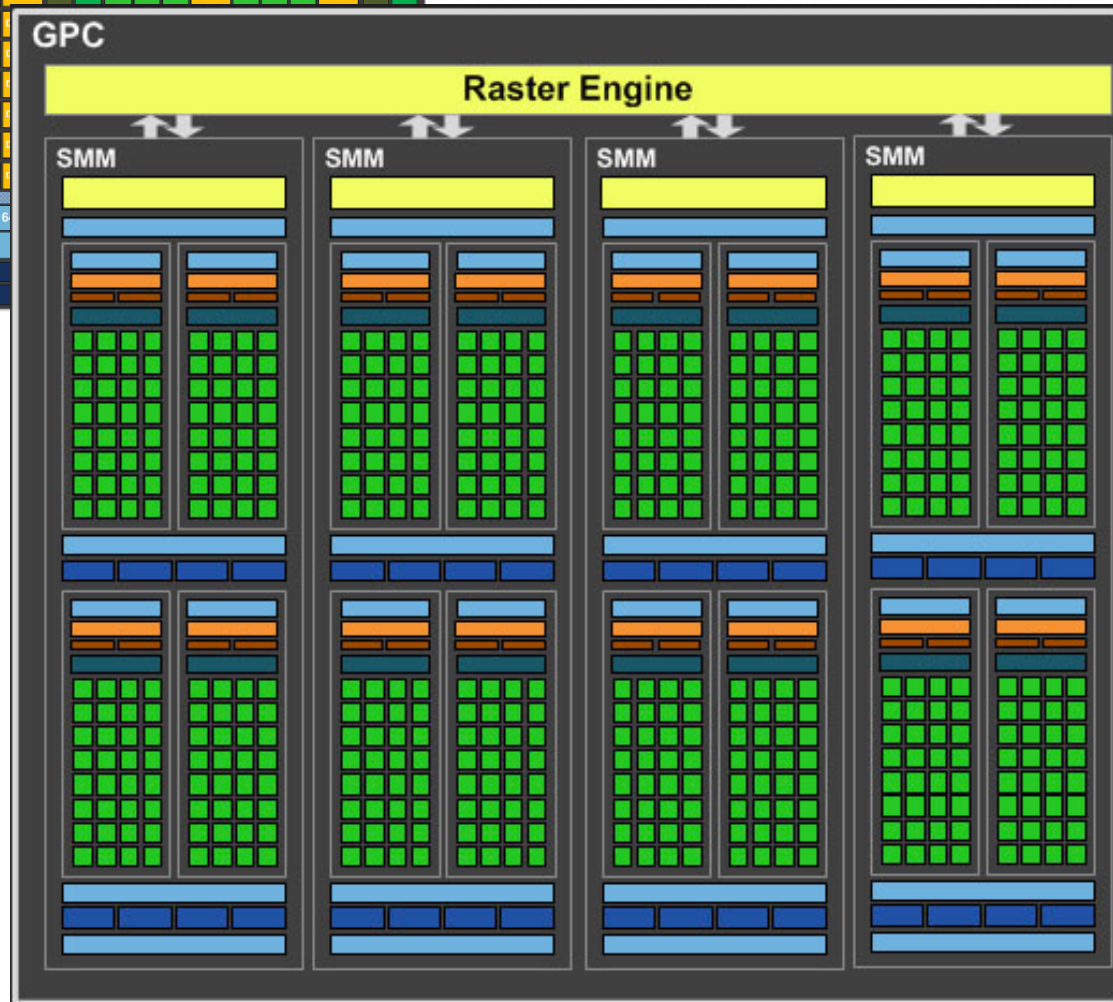




Kepler SMX

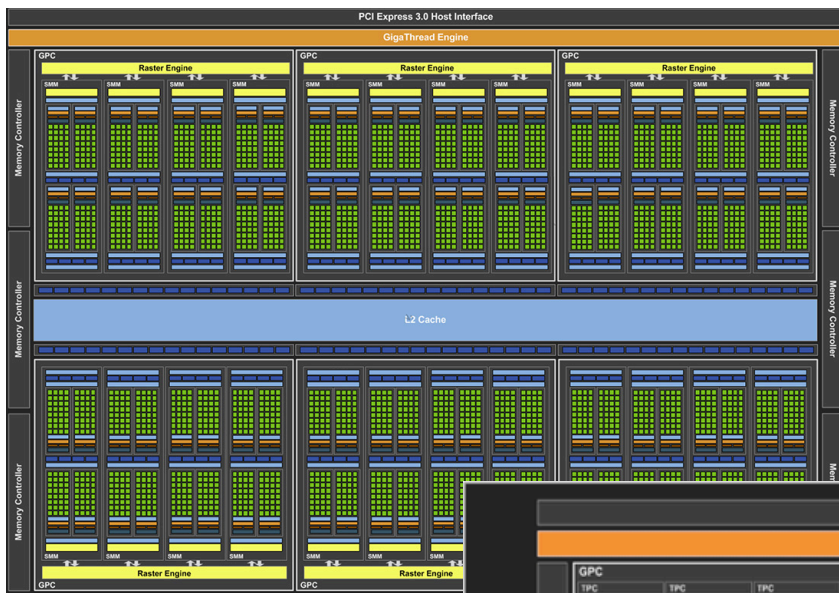
# The move from Kepler to Maxwell : from 15 SMXs to 48 SMMs in 6 GPCs

SMM: 128 CUDA-cores  
Ratio DPunit : SPunit → 1 : 32

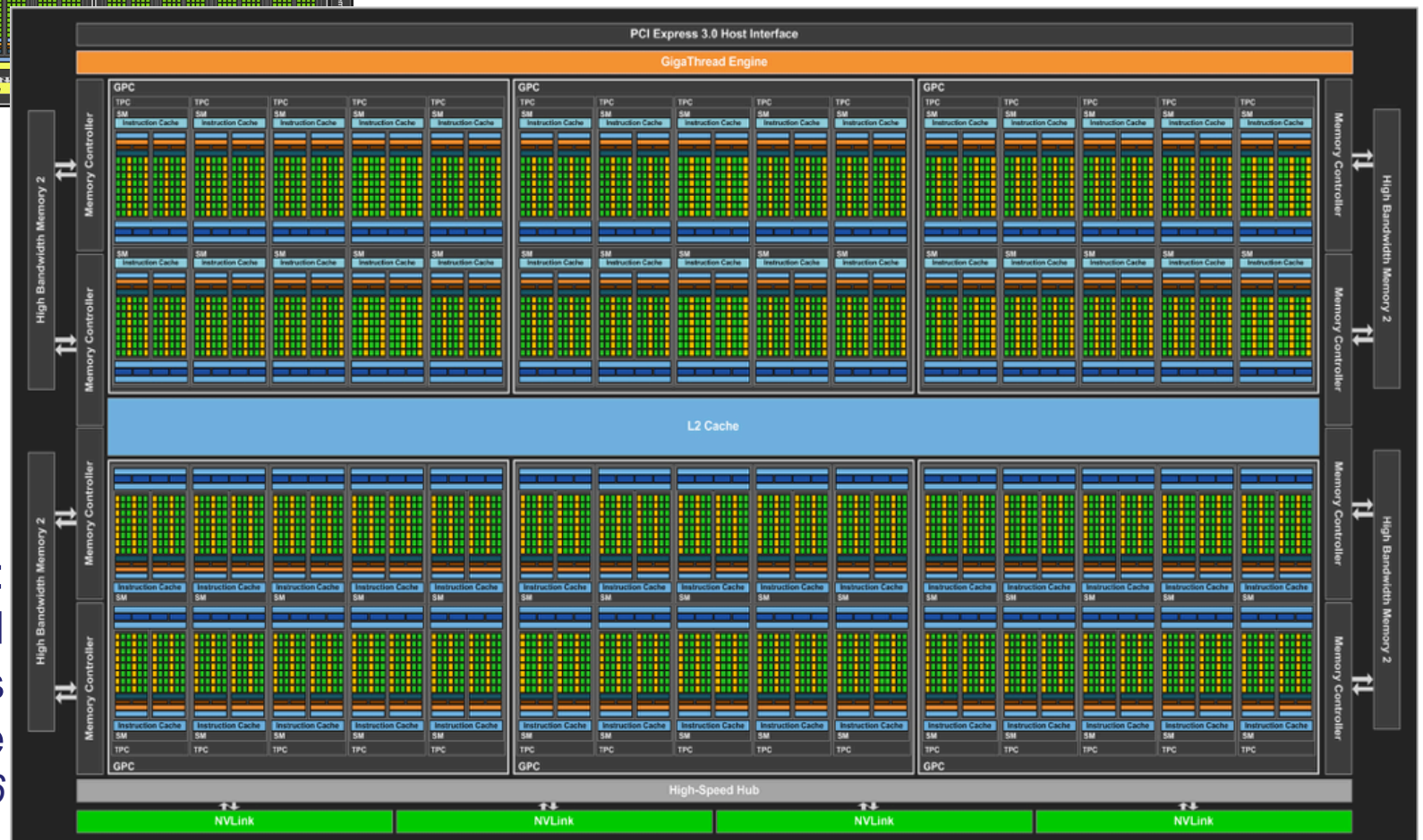




# From the M200 to the GP100 Pascal Architecture

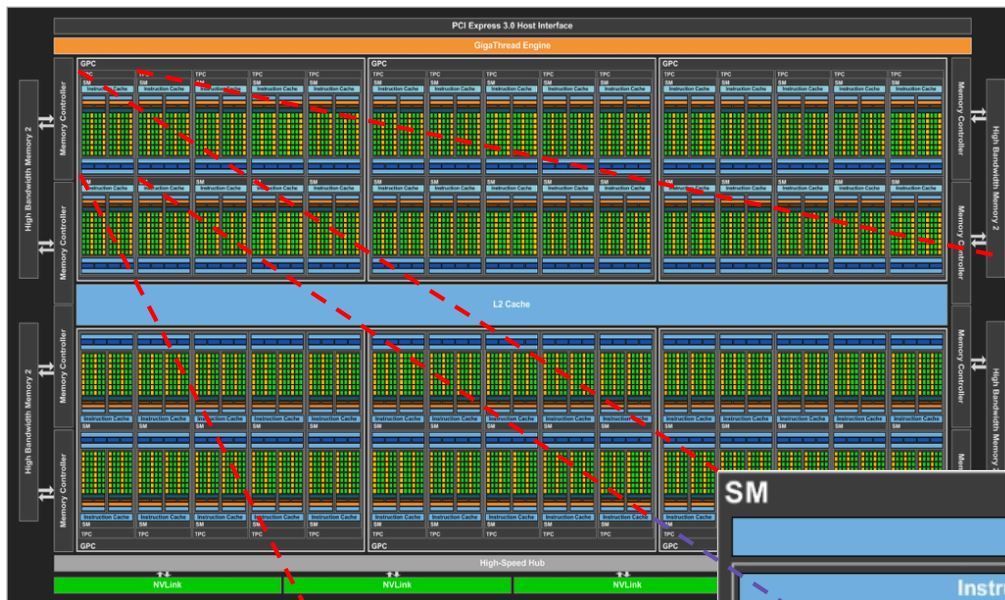


Maxwell:  
48 SMM  
3072 CUDA-cores  
November'15



Pascal:  
60 SM  
3840 CUDA-cores  
4 HBM on-package  
September'16

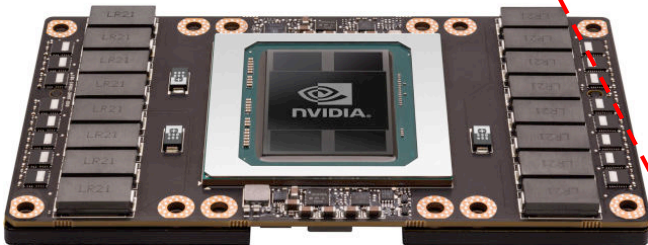
# Pascal Architecture: 6x GPCs, 60 SMs



Pascal SM:  
64 CUDA-cores

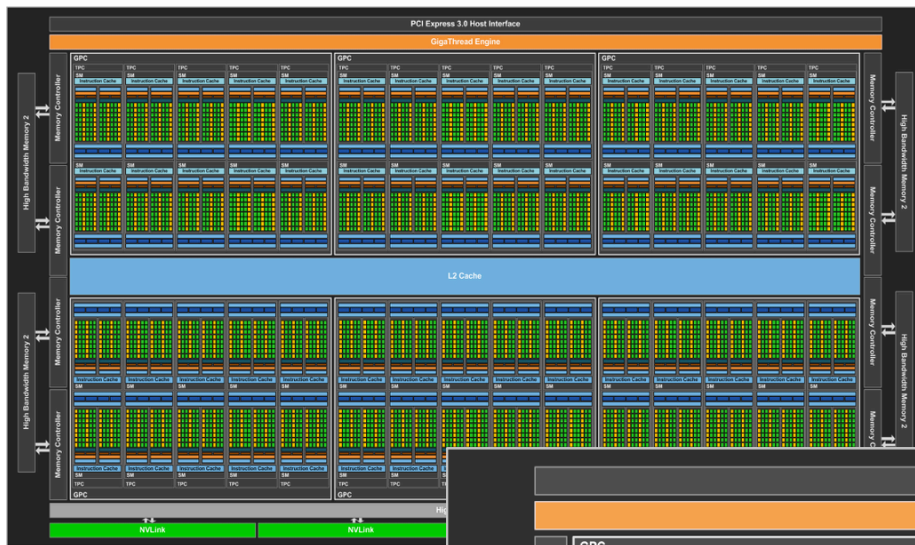
Ratio DPunit : SPunit  $\rightarrow 1 : 2$

Pascal P100 w/ 16GB HBM2





# From the GP100 to the GV100 Volta Architecture



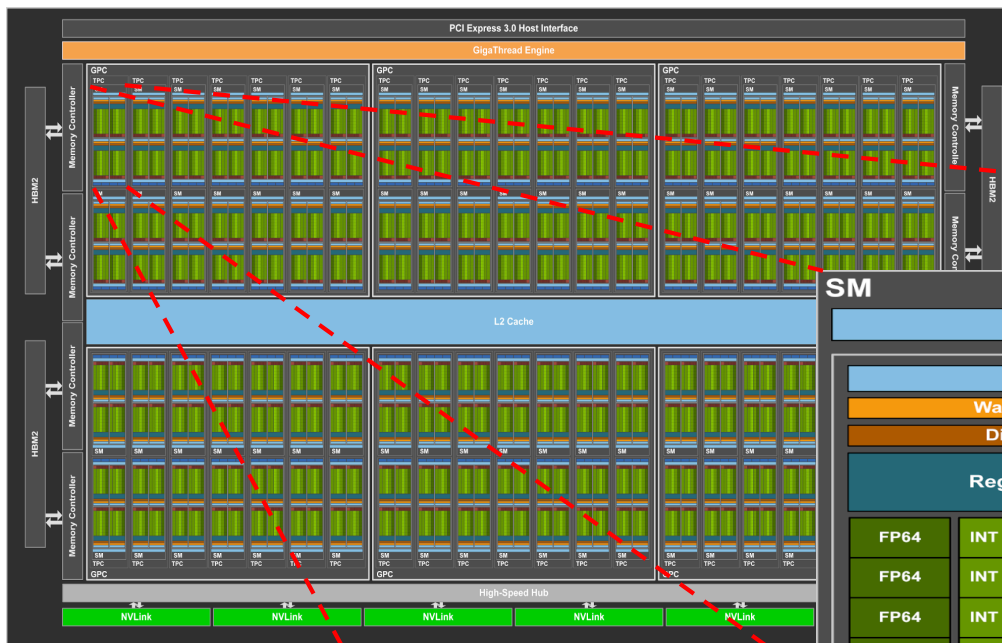
Pascal:  
60 SM  
3840 CUDA-cores  
November'15



Volta:  
84 SM  
5120 CUDA-cores  
HBM on-package  
June'17



# Volta Architecture: 6x GPCs, 84 SMs



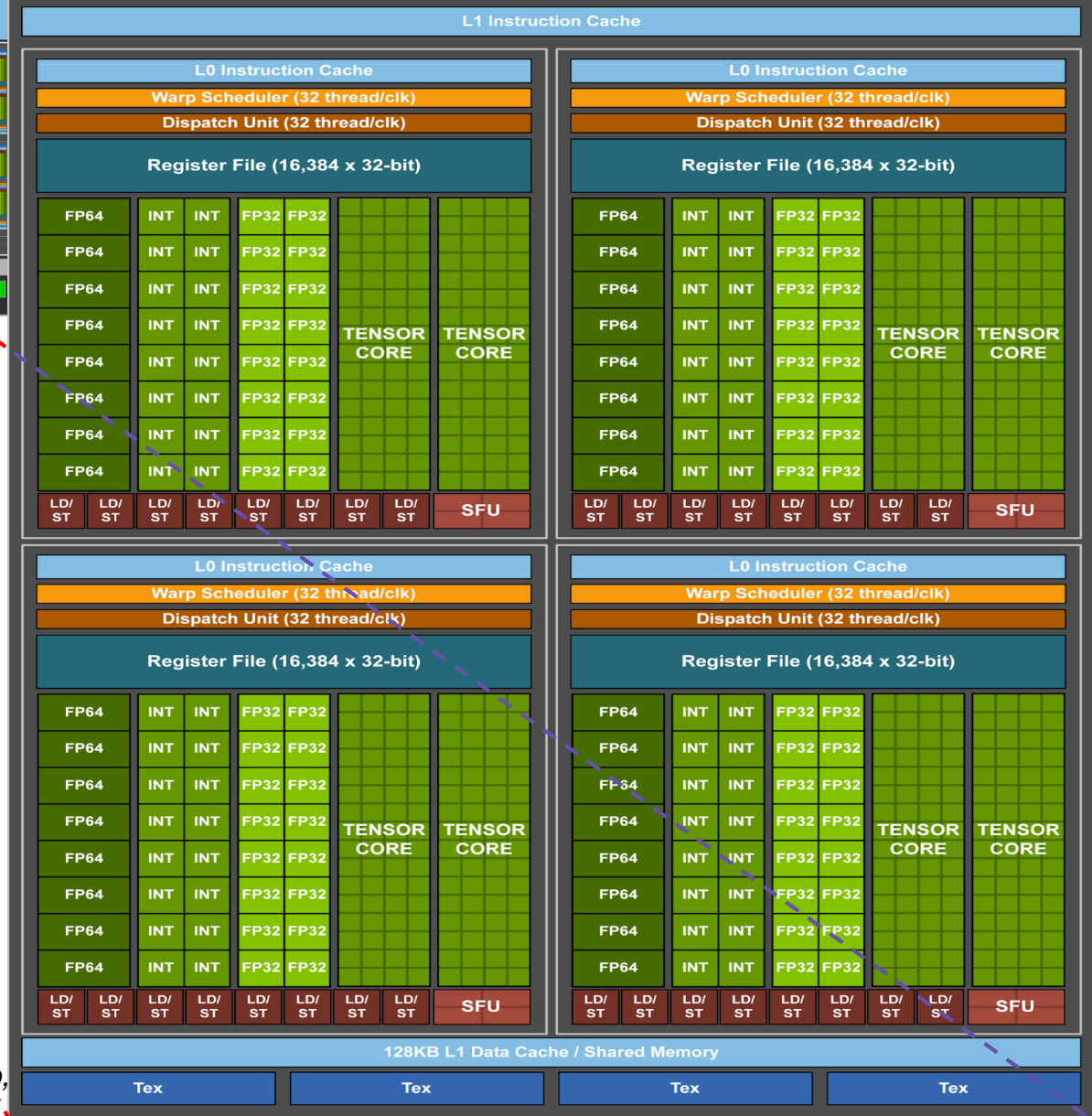
SM

Volta SM:  
64 CUDA-cores  
**New:** 8 Tensor-cores  
Ratio DPunit : SPunit  $\rightarrow$  1 : 2

Volta V100 w/ 16GiB HBM2



AJProença, Parallel Computing, MiEI, UMinho,

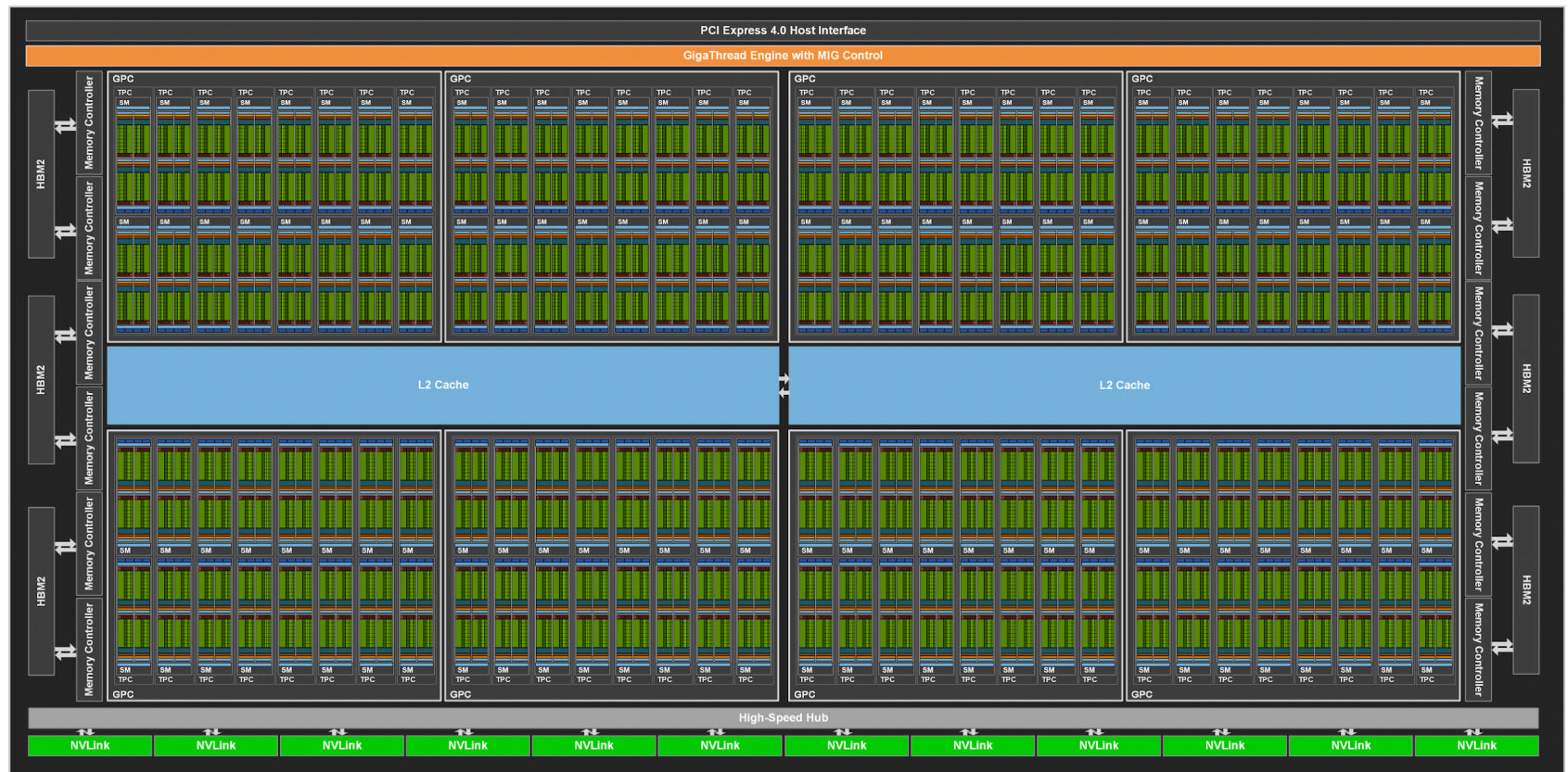
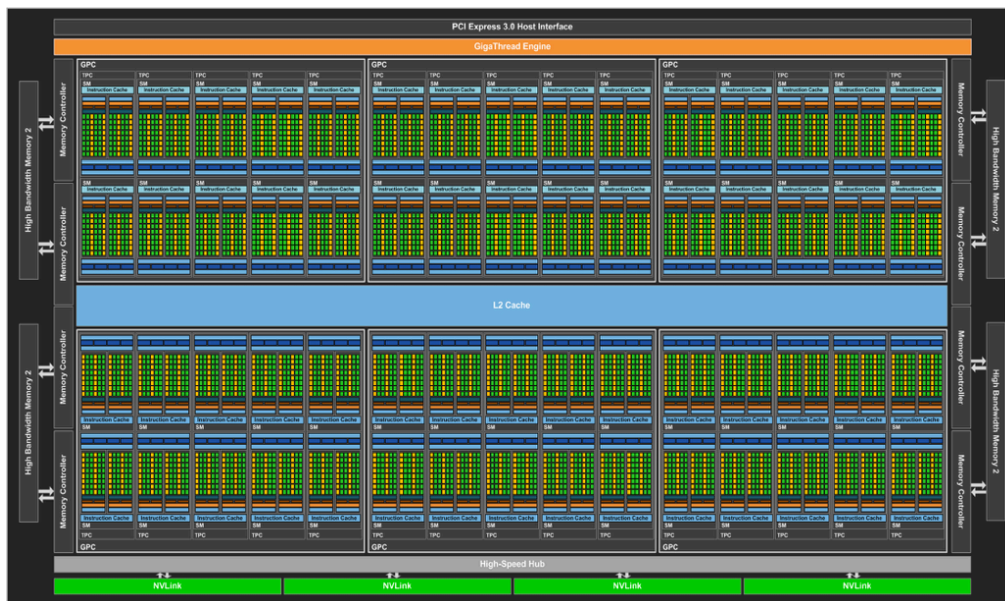


# From GV 100 to Ampere: up to 8 GPC, 128 SMs total

Ampere: NVidia GA100  
8192 FP32 CUDA Cores  
512 3<sup>rd</sup> generation Tensor Cores  
6 HBM2, 12 512-bit mem controllers  
May'20

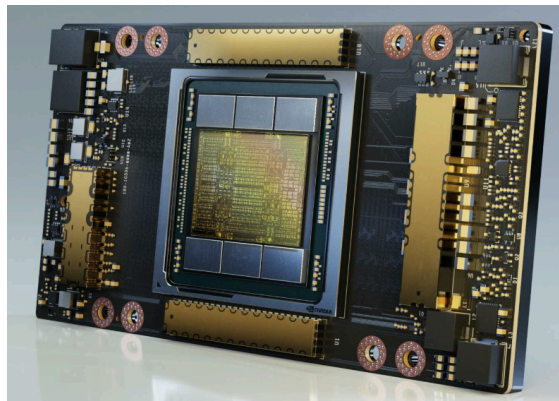
Volta:  
84 SM  
3584 CUDA-cores  
November'15

Ampere:  
GA100  
for graphics  
w/ 8 GPC  
A100  
for HPC & AI  
w/ 7 GPC





# Ampere Architecture



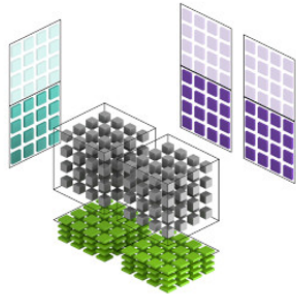
Ampere SM:  
 64x FP32 CUDA Cores/SM  
 32x FP64 CUDA Cores/SM  
 4x 3<sup>rd</sup> generation Tensor Cores  
 Tensor Cores support  
 FP64, FP32, TF32, FP16, BF16, INT8...  
 1024 dense FP16/FP32 FMA op's/cycle

AJProença, Parallel Computing, MiEI, UMinho, 2022

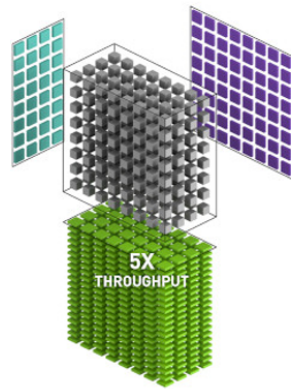
# Tensor cores in Ampere



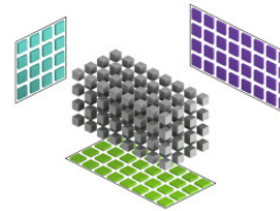
NVIDIA V100 Tensor Core FP16



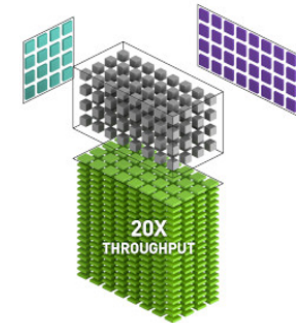
NVIDIA A100 Tensor Core FP16 with Sparsity



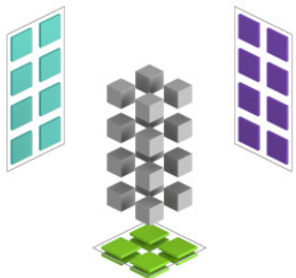
NVIDIA V100 FP32



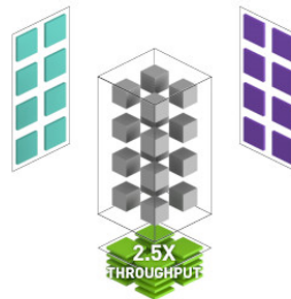
NVIDIA A100 Tensor Core TF32 with Sparsity



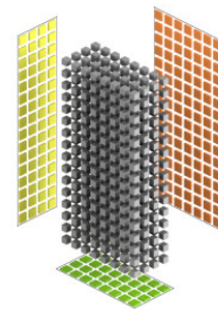
NVIDIA V100 FP64



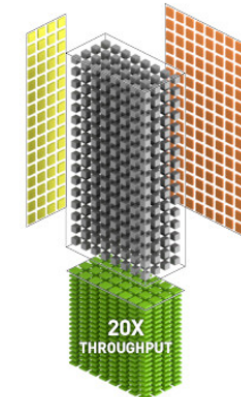
NVIDIA A100 Tensor Core FP64



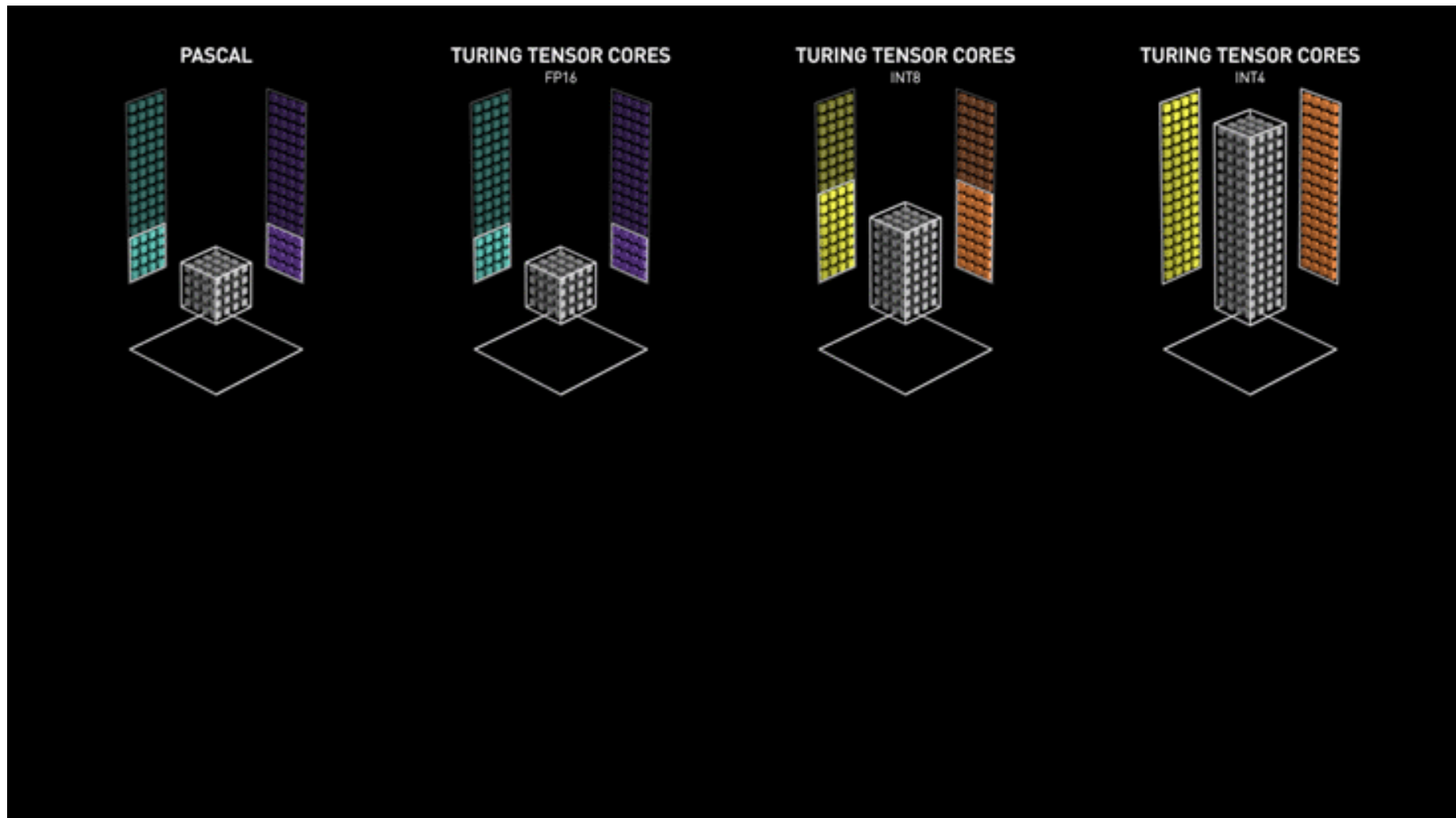
NVIDIA V100 INT8



NVIDIA A100 Tensor Core INT8 with Sparsity



# *Pascal vs. Turing tensor cores (animation)*



Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOP/s*	5.04	6.8	10.6	15.7
Peak FP64 TFLOP/s*	1.68	.21	5.3	7.8
Peak Tensor Core TFLOP/s*	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB	20480 KB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm <sup>2</sup>	601 mm <sup>2</sup>	610 mm <sup>2</sup>	815 mm <sup>2</sup>
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

# Tesla accelerators: evolution

## Ampere SYSTEM SPECIFICATIONS (PEAK PERFORMANCE)

	NVIDIA A100 for NVIDIA HGX™	NVIDIA A100 for PCIe
GPU Architecture	NVIDIA Ampere	
Double-Precision Performance	FP64: 9.7 TFLOPS FP64 Tensor Core: 19.5 TFLOPS	
Single-Precision Performance	FP32: 19.5 TFLOPS Tensor Float 32 (TF32): 156 TFLOPS   312 TFLOPS*	
Half-Precision Performance	312 TFLOPS   624 TFLOPS*	
Bfloat16	312 TFLOPS   624 TFLOPS*	
Integer Performance	INT8: 624 TOPS   1,248 TOPS* INT4: 1,248 TOPS   2,496 TOPS*	
GPU Memory	40 GB HBM2	
Memory Bandwidth	1.6 TB/sec	



# Tesla evolution (1)

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOP/s*	5.04	6.8	10.6	15.7
Peak FP64 TFLOP/s*	1.68	.21	5.3	7.8
Peak Tensor Core TFLOP/s*	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB	20480 KB

<https://devblogs.nvidia.com/parallelforall/inside-volta/>

## Tesla evolution (2)



Nvidia Datacenter GPU	Nvidia Tesla V100	Nvidia A100
GPU codename	GV100	GA100
GPU architecture	Volta	Ampere
Launch date	May 2017	May 2020
GPU process	TSMC 12nm	TSMC 7nm
Die size	815mm <sup>2</sup>	826mm <sup>2</sup>
Transistor Count	21.1 billion	54 billion
FP64 CUDA cores	2,560	3,456
FP32 CUDA cores	5,120	6,912
Tensor Cores	640	432
Streaming Multiprocessors	80	108
Peak FP64	7.8 teraflops	9.7 teraflops
Peak FP64 Tensor Core	-	19.5 teraflops
Peak FP32	15.7 teraflops	19.5 teraflops
Peak FP32 Tensor Core	-	156 teraflops/312 teraflops*
Peak BFLOAT16 Tensor Core	-	312 teraflops/624 teraflops*
Peak FP16 Tensor Core	-	312 teraflops/624 teraflops*
Peak INT8 Tensor Core	-	624 teraflops/1,248 TOPS*
Peak INT4 Tensor Core	-	1,248 TOPS/2,496 TOPS*
Mixed-precision Tensor Core	125 teraflops	312 teraflops/624 teraflops*
Max TDP	300 watts	400 watts

AJProença, \*Effective TOPS / TFLOPS using the new Sparsity feature



# The CUDA programming model



- *Compute Unified Device Architecture*
- CUDA is a recent programming model, designed for
  - a multicore **CPU** *host* coupled to a many-core *device*, where
  - *devices* have wide SIMD/SIMT parallelism, and
  - the *host* and the *device* do not share memory
- CUDA provides:
  - a thread abstraction to deal with SIMD
  - synchr. & data sharing between small groups of threads
- CUDA programs are written in C with extensions
- OpenCL inspired by CUDA, but hw & sw vendor neutral
  - programming model essentially identical

# CUDA Devices and Threads

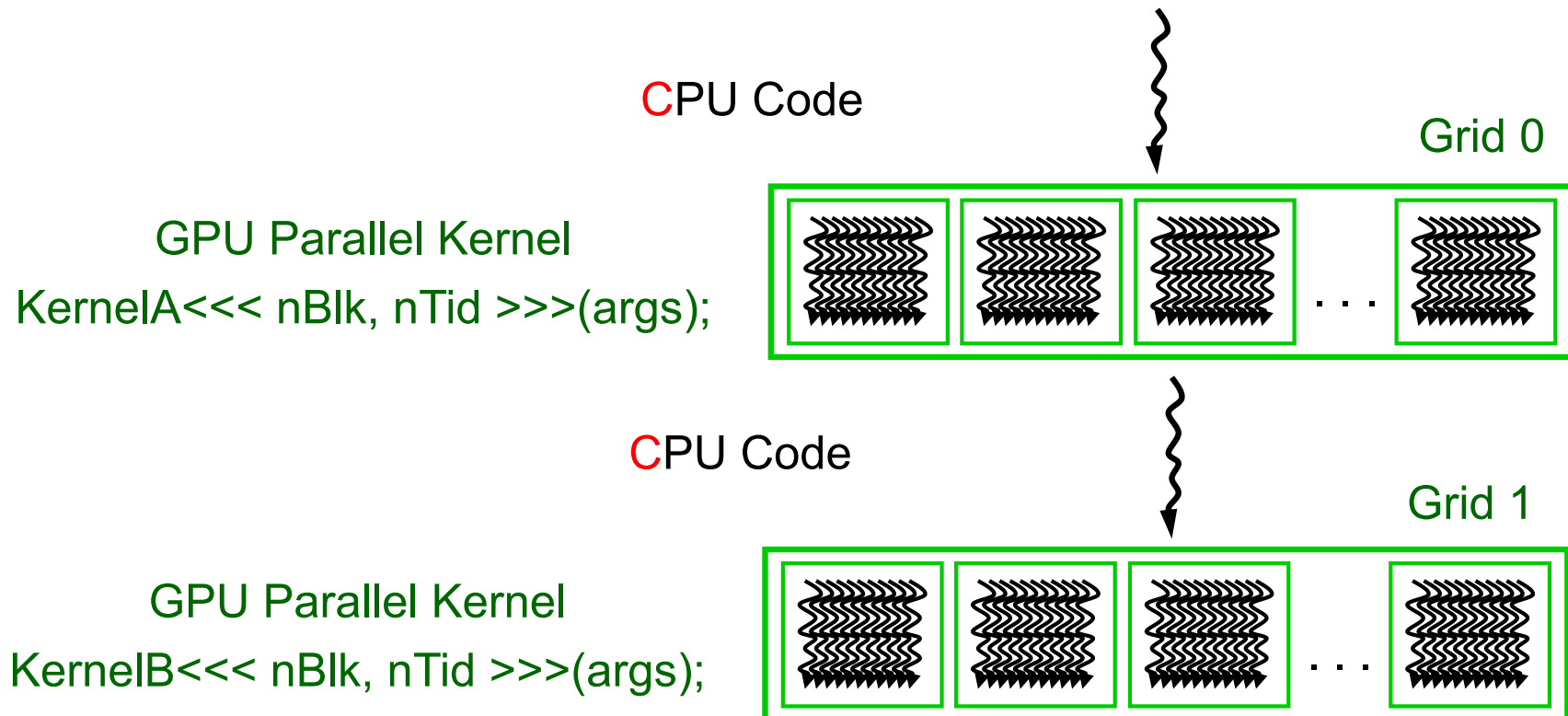


- A compute **device**
  - is a coprocessor to the **C**PU or **host**
  - has its own DRAM (**device memory**)
  - runs many **threads in parallel**
  - is typically a **GPU** but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device **kernels** which run on many threads - **SIMT**
- Differences between GPU and **C**PU threads
  - GPU threads are extremely lightweight
    - very little creation overhead, **requires LARGE register bank**
  - GPU needs 1000s of threads for full efficiency
    - multi-core **C**PU needs only a few

# CUDA basic model: Single-Program Multiple-Data (SPMD)



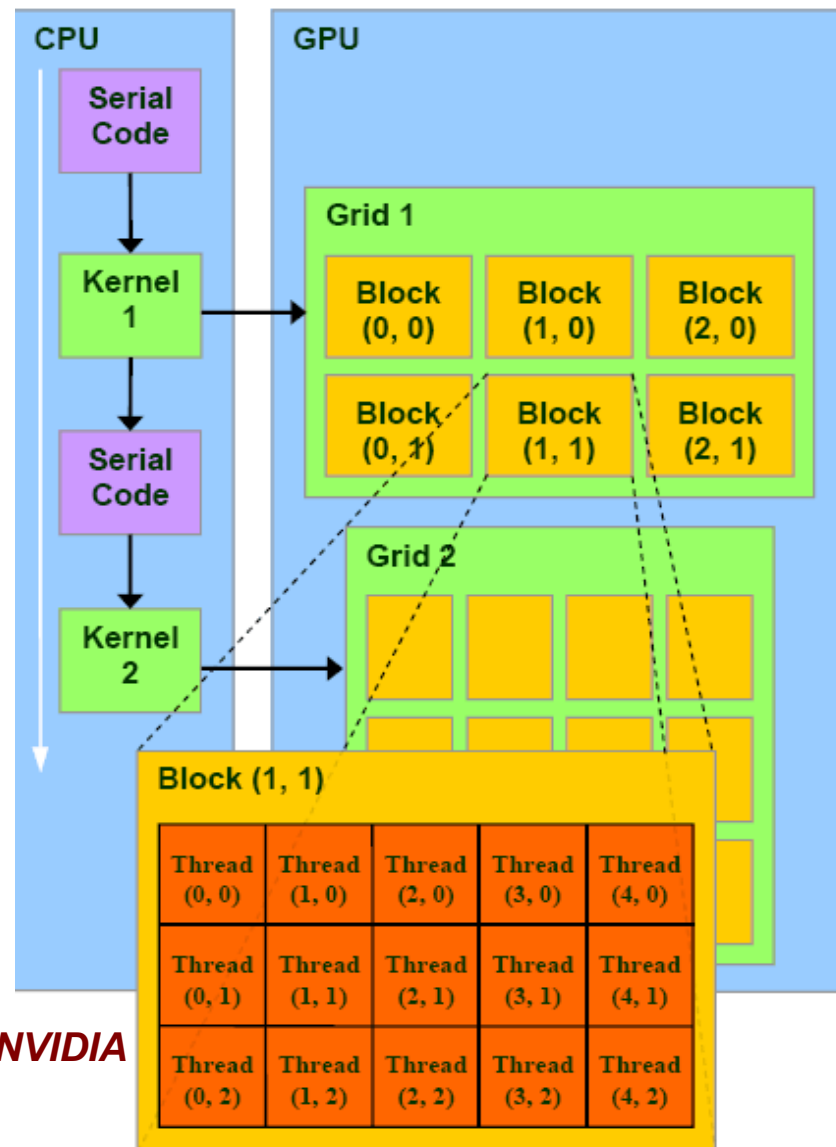
- CUDA integrated **C**PU + GPU application C program
  - Serial C code executes on **C**PU
  - Parallel **K**ernel C code executes on GPU **t**hread **b**locks



# Programming Model: SPMD + SIMT/SIMD



- Hierarchy
  - Device => Grids
  - Grid => Blocks
  - Block => Warps
  - Warp => Threads
- Single kernel runs on multiple blocks (SPMD)
- Threads within a warp are executed in a lock-step way called single-instruction multiple-thread (SIMT)
- Single instruction are executed on multiple threads (SIMD)
  - Warp size defines SIMD granularity (32 threads)
- Synchronization within a block uses shared memory

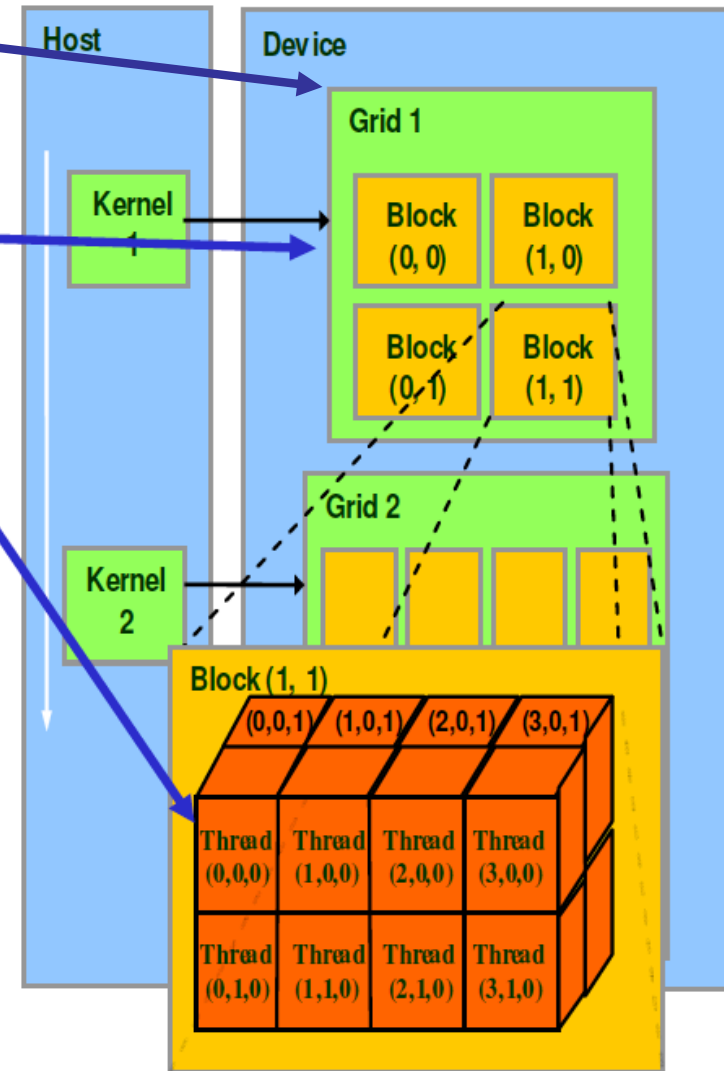


*Courtesy NVIDIA*

# The Computational Grid: Block IDs and Thread IDs



- A **kernel** runs on a **computational grid of thread blocks**
  - Threads share global memory
- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- A thread block is a batch of threads that can cooperate by:
  - Sync their execution w/ barrier
  - Efficiently sharing data through a low latency shared memory
  - Two threads from two different blocks cannot cooperate





## C with CUDA Extensions: C with a few keywords

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

*Standard C Code*

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<nblocks, 256>>>(n, 2.0, x, y);
```

*Parallel C Code*

NVIDIA Confidential

# Terminology (and in NVidia)



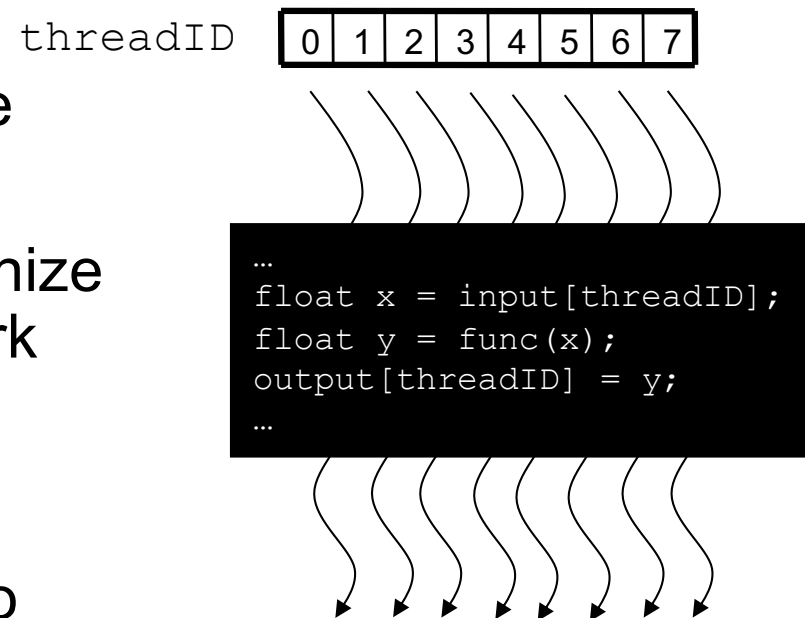
- Threads of SIMD instructions (**warps**)
  - Each has its own IP (up to 48/64 per SIMD processor, Fermi/Kepler)
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Threads are organized into blocks & executed in groups of 32 threads (**thread block**)
    - Blocks are organized into a grid
- The thread block scheduler schedules blocks to SIMD processors (**Streaming Multiprocessors**)
- Within each SIMD processor:
  - 32 SIMD lanes (**thread processors**)
  - Wide and shallow compared to vector processors

# CUDA Thread Block



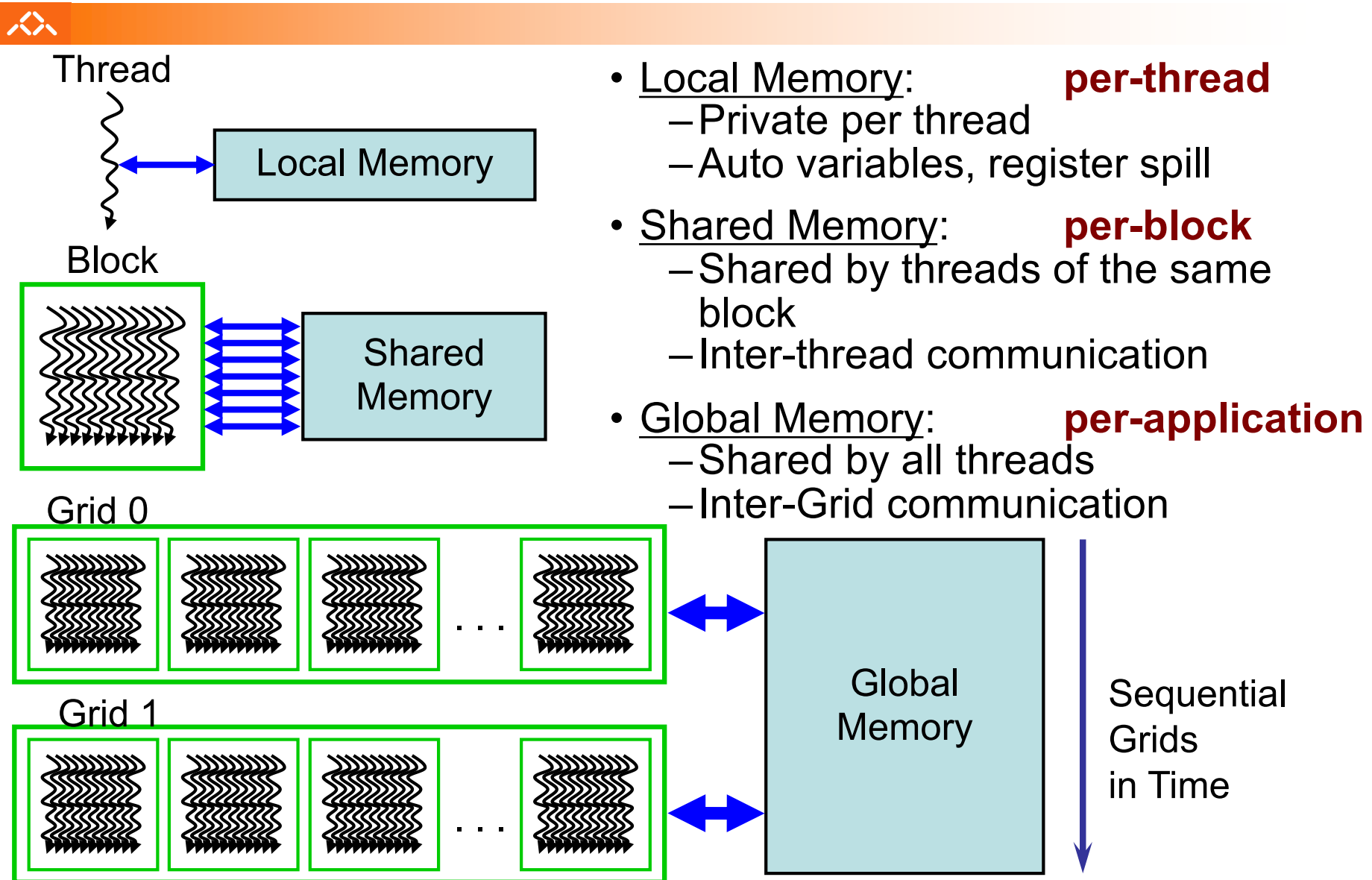
- Programmer declares (Thread) Block:
  - Block size 1 to **512** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have **thread id** numbers within Block
- Thread program uses **thread id** to select work and address shared data

## CUDA Thread Block





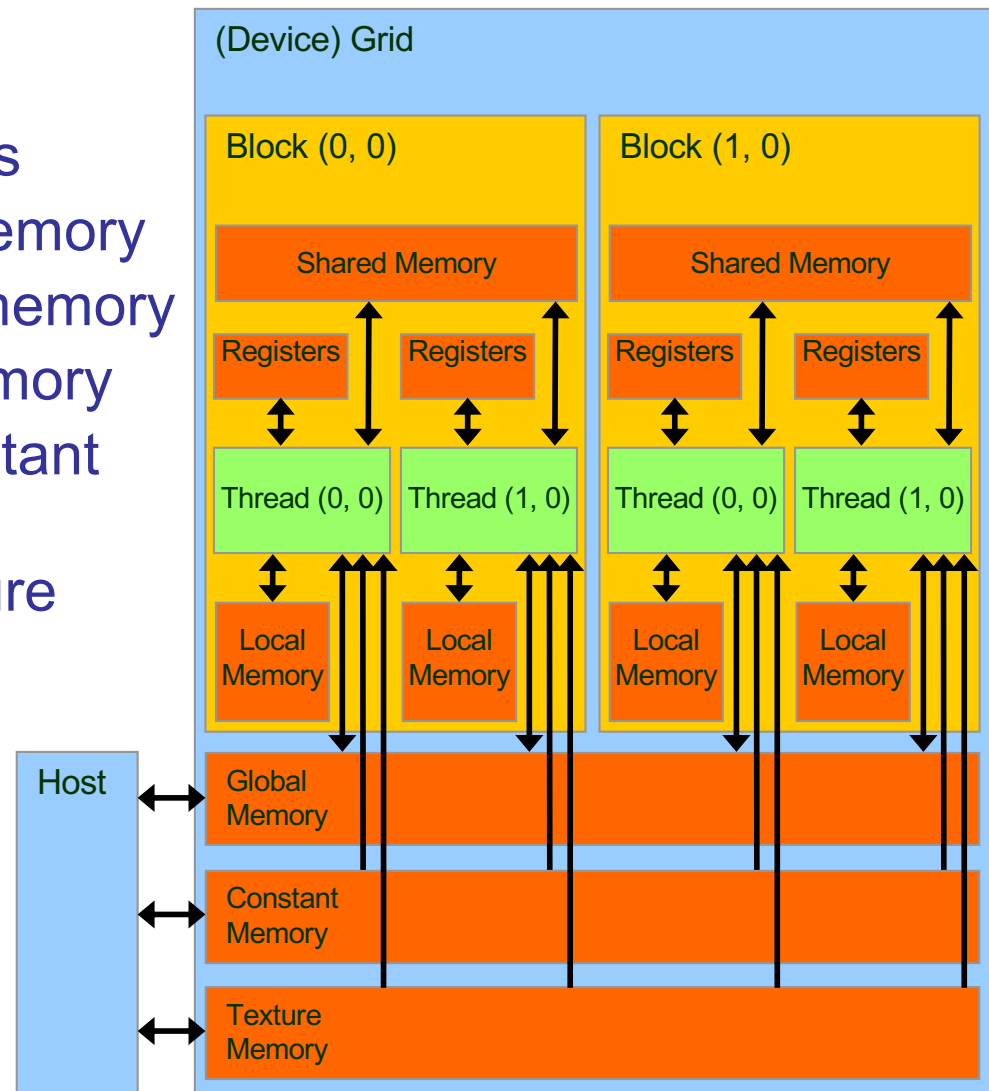
# Parallel Memory Sharing



# CUDA Memory Model Overview



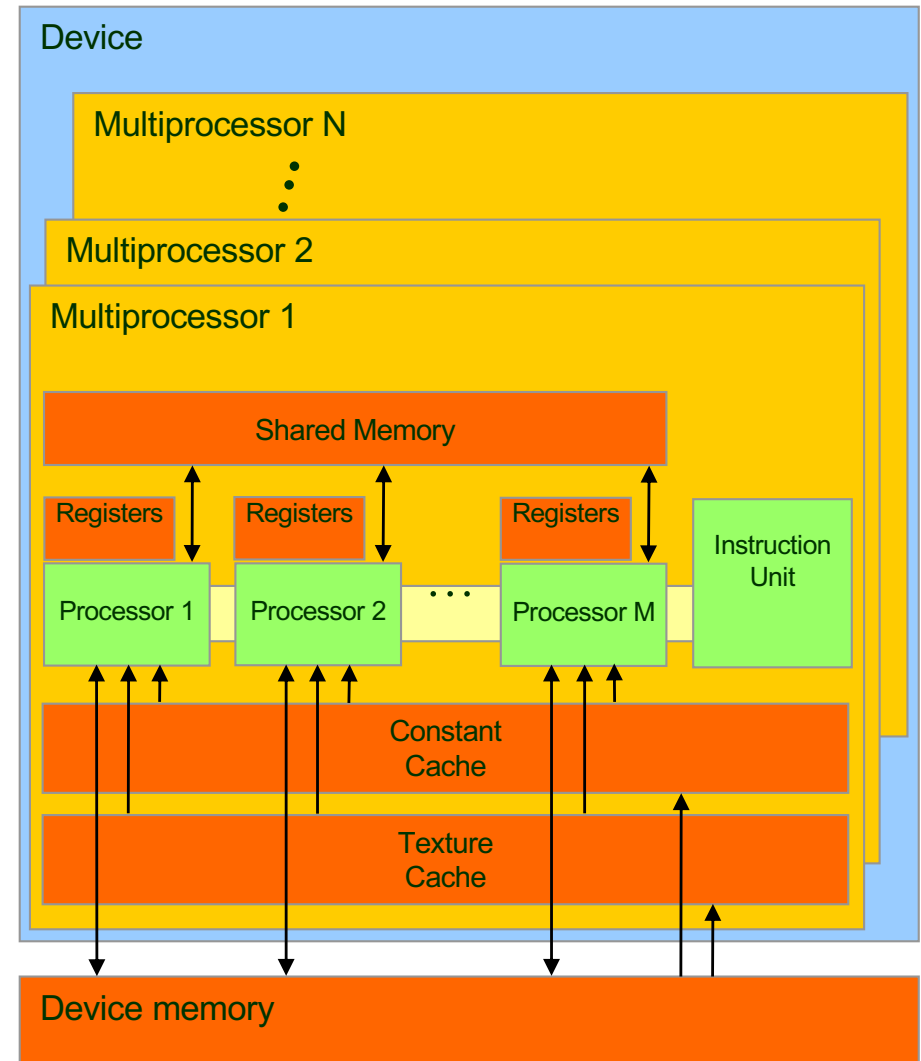
- Each thread can:
  - R/W per-thread **registers**
  - R/W per-thread **local memory**
  - R/W per-block **shared memory**
  - R/W per-grid **global memory**
  - Read only per-grid **constant memory**
  - Read only per-grid **texture memory**
- The host can R/W global, constant, and texture memories



# Hardware Implementation: Memory Architecture



- Device memory (DRAM)
  - Slow (2~300 cycles)
  - Local, global, constant, and texture memory
- On-chip memory
  - Fast (1 cycle)
  - Registers, shared memory, constant/texture cache



Courtesy NVIDIA

# Terminology: CUDA and OpenCL



## CUDA and OpenCL

