Master Informatics Eng.

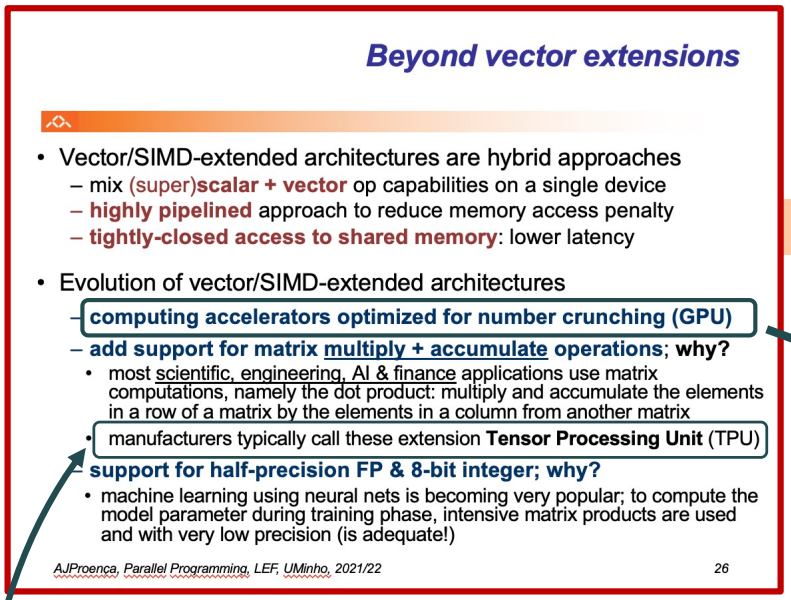
2021/22

A.J.Proença

Computing accelerators: GPU & CUDA

(most slides are borrowed)

Compute accelerators



Beyond vector extensions

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of vector/SIMD-extended architectures
  - **computing accelerators optimized for number crunching (GPU)**
  - **add support for matrix multiply + accumulate operations; why?**
    - most scientific, engineering, AI & finance applications use matrix computations, namely the dot product: multiply and accumulate the elements in a row of a matrix by the elements in a column from another matrix
  - manufacturers typically call these extension **Tensor Processing Unit** (TPU)
  - **support for half-precision FP & 8-bit integer; why?**
    - machine learning using neural nets is becoming very popular; to compute the model parameter during training phase, intensive matrix products are used and with very low precision (is adequate!)

AJProença, Parallel Programming, LEF, UMinho, 2021/22 26

Best accelerator for number crunching, namely intensive vector/matrix computing: **GPU**

Other common compute accelerators:

- DSP: Digital Signal Processor, mostly used in telecommunication equipments, from cell phones to radio systems and TVs
- TPU: Tensor Processing Units, optimized for operations with tensors (vector and n-dimensional matrices), popular in AI app's, namely in autonomous driving
- FPGA: Field Programmable Gate Arrays, reconfigurable h/w; can be configured in runtime to behave according to a given specification

Data Parallelism: SIMD CPU vs. GPU

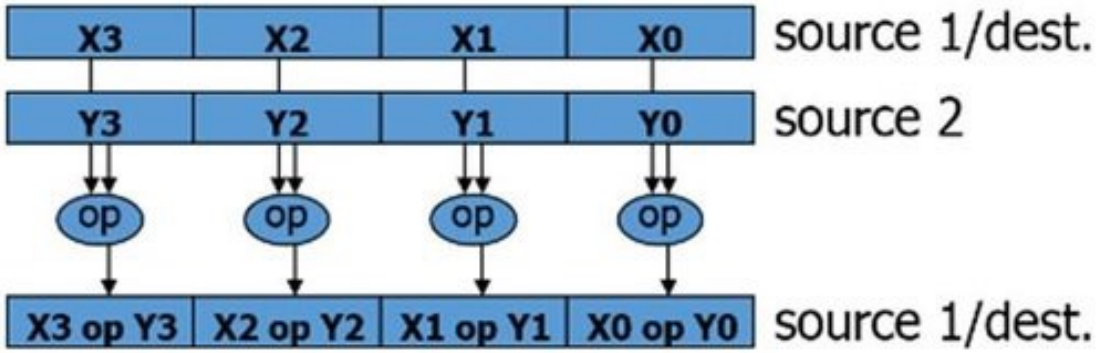
CPU

SIMD

1 instruction – multiple data

SSE2/3/4 – Neon – Altivec

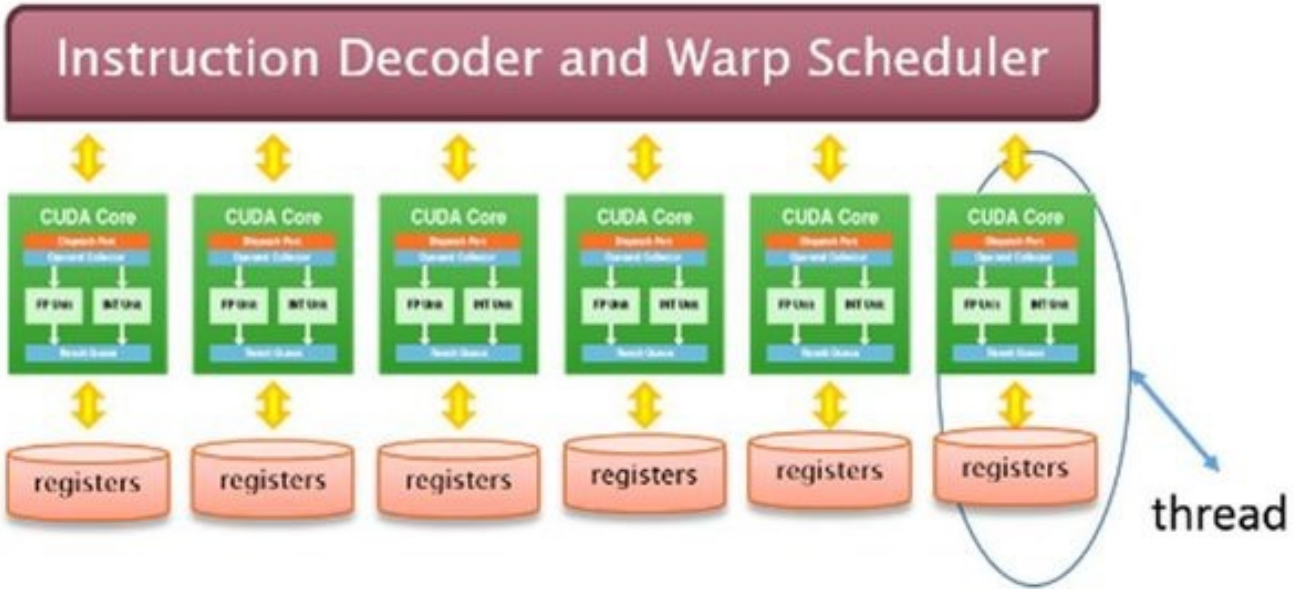
AVX – AVX2...



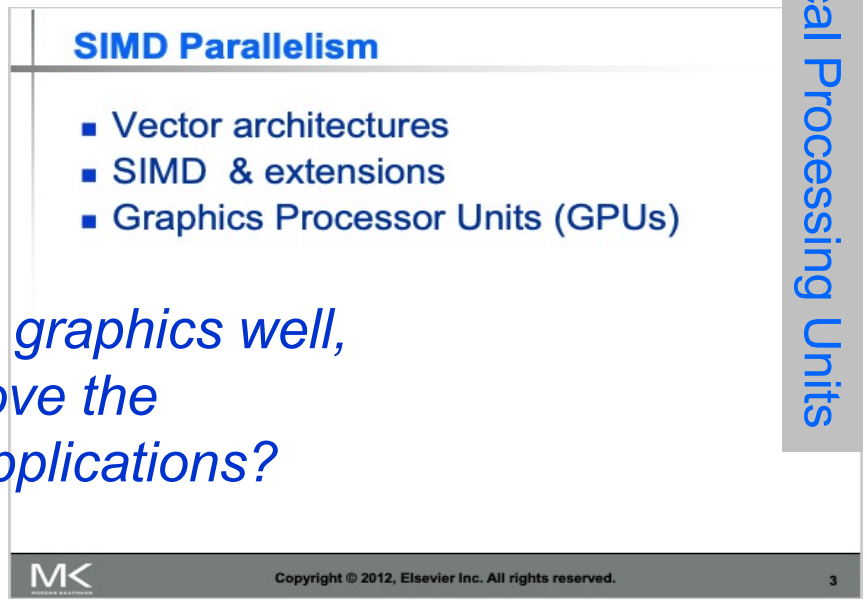
GPU

SIMT

1 instruction – multiple threads



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*"

cores/processing element in several computing devices

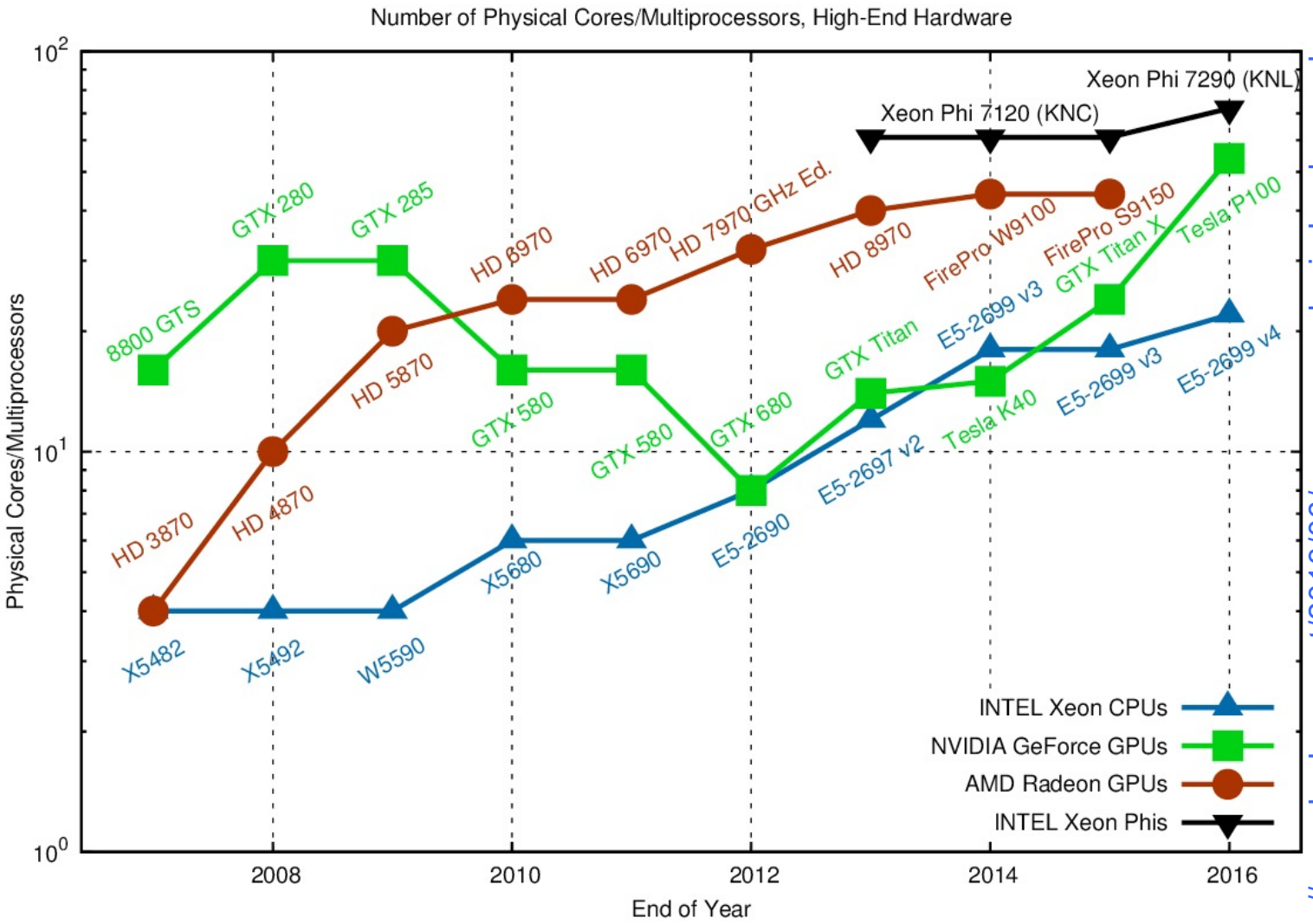
Key question: what is a **core**?

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

b) A SIMD processor?
CPU-type..

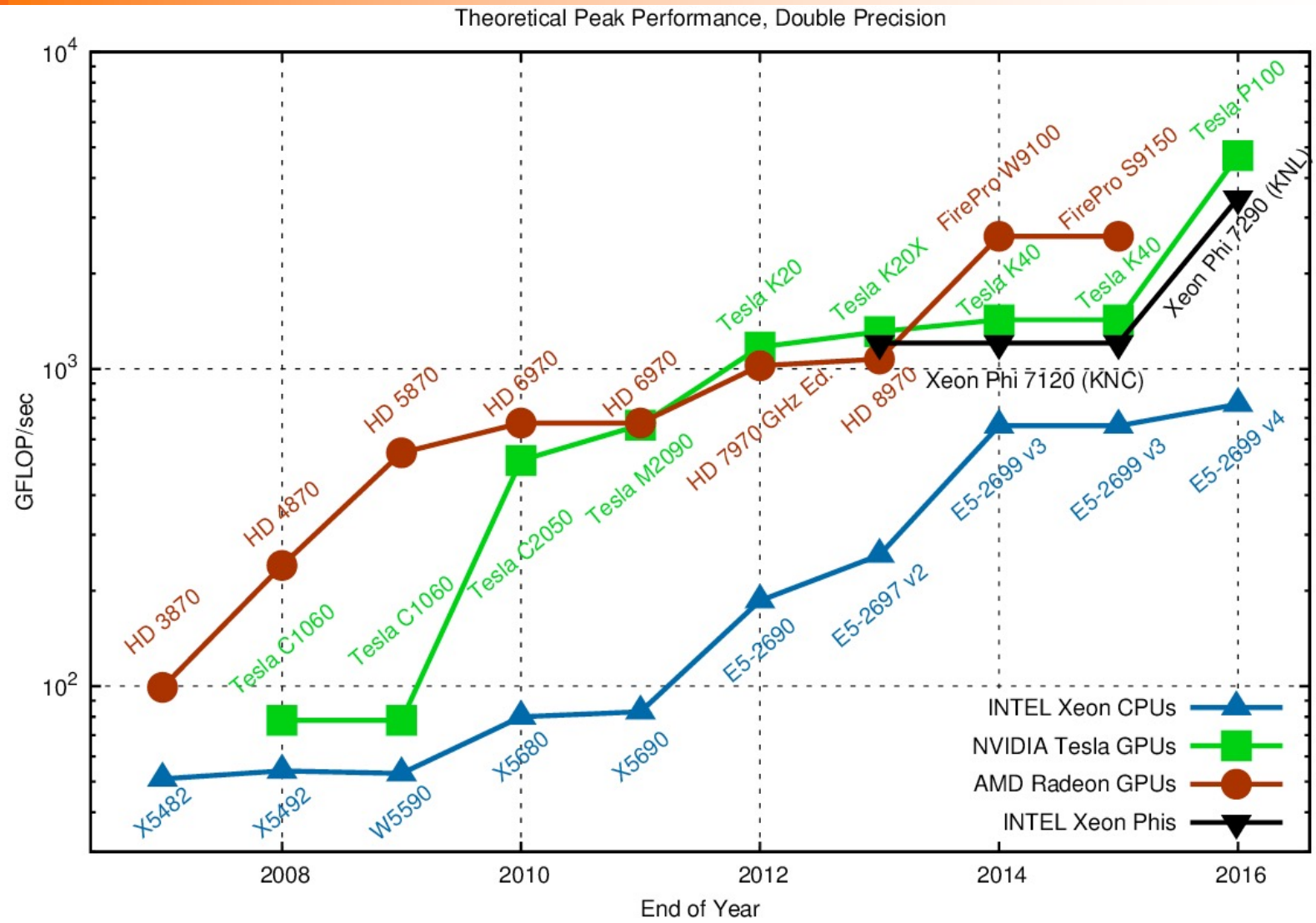
This updated slide and in this course:
- b)

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



http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Theoretical peak performance in several computing devices (DP)



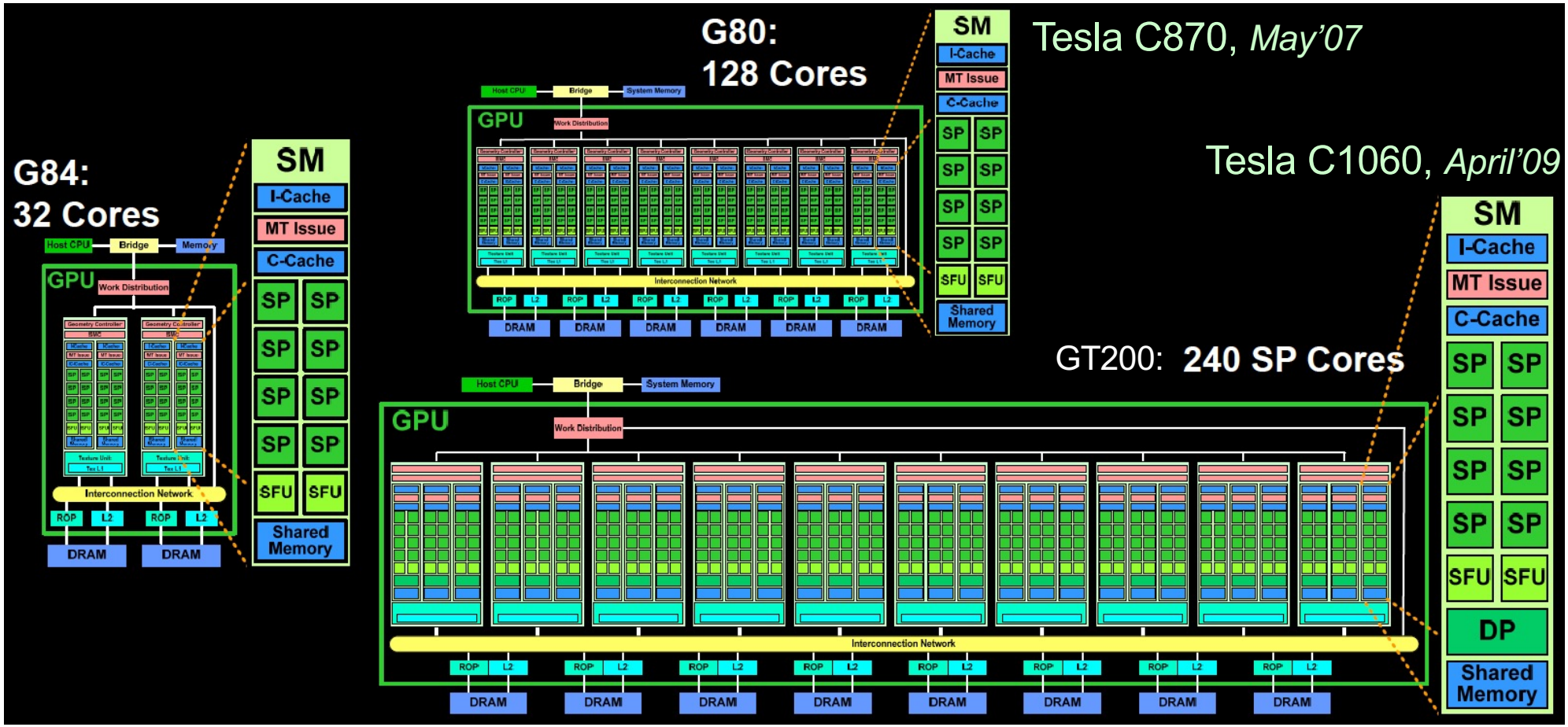
http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

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

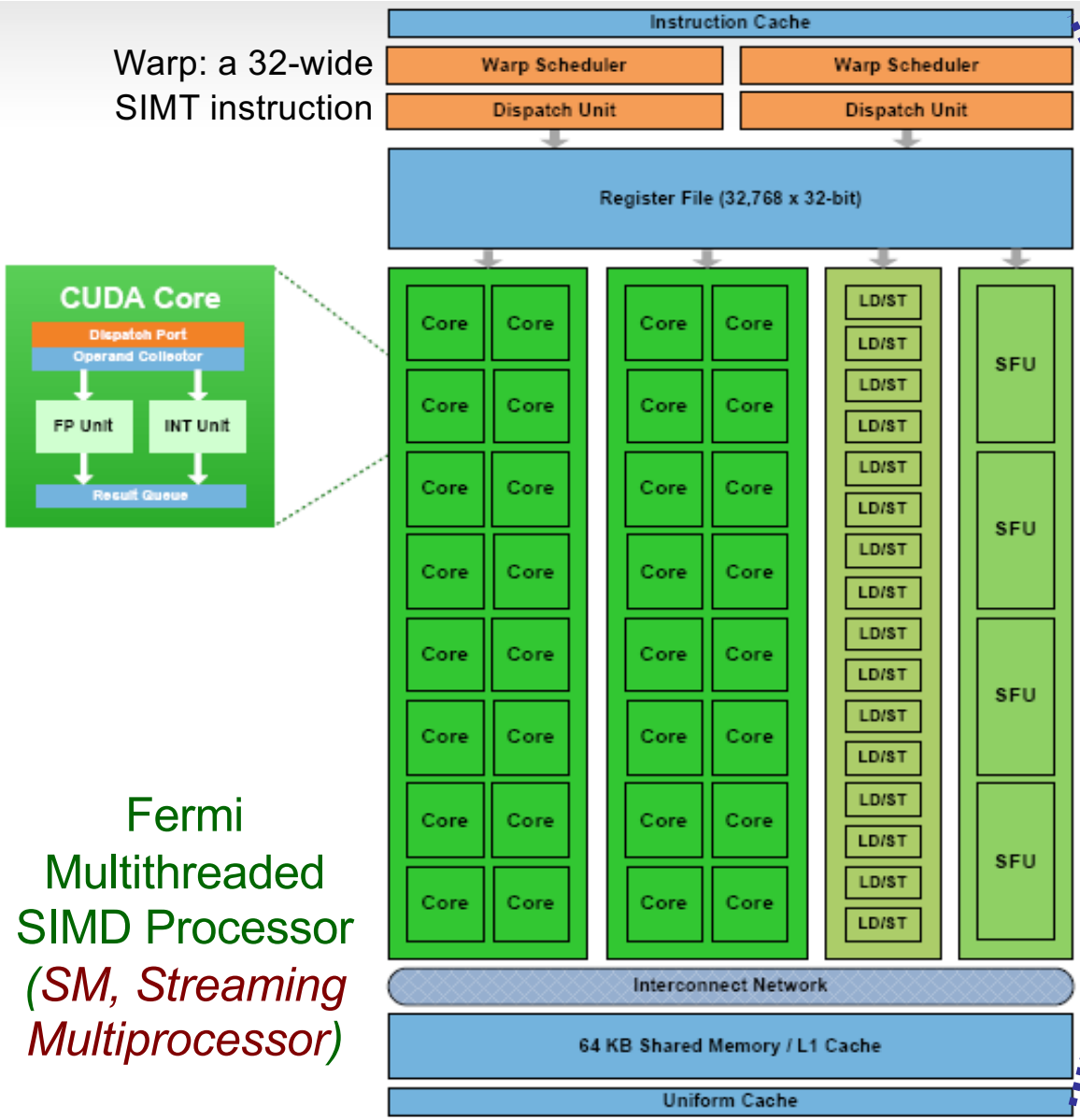


NVIDIA GPU Memory Structures

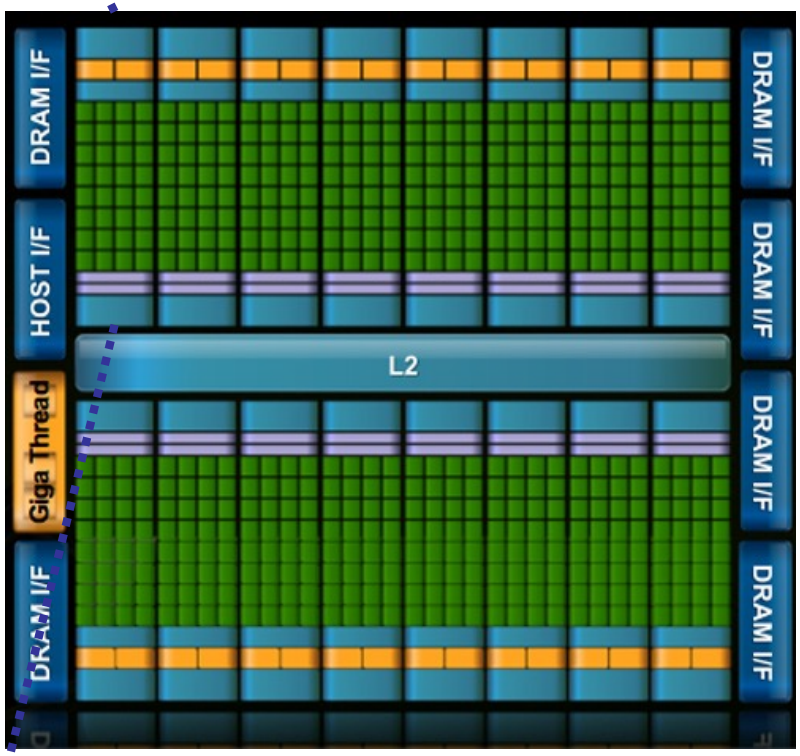
- Each SIMD Lane has private section of **off-chip DRAM**
  - "Private memory" *(Local Memory)*
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor *(SM)* also has local memory *(Shared Memory)*
  - Shared by SIMD lanes / threads within a block
- Memory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

The NVidia Fermi architecture

Warp: a 32-wide SIMT instruction



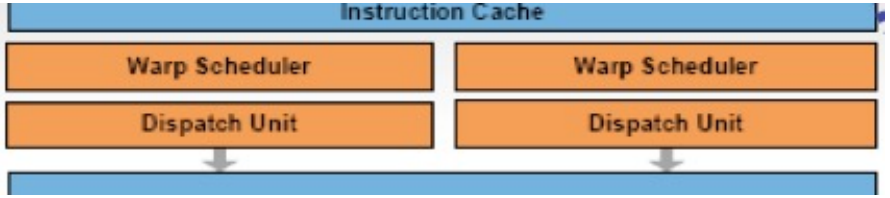
Fermi Multithreaded SIMD Processor *(SM, Streaming Multiprocessor)*



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

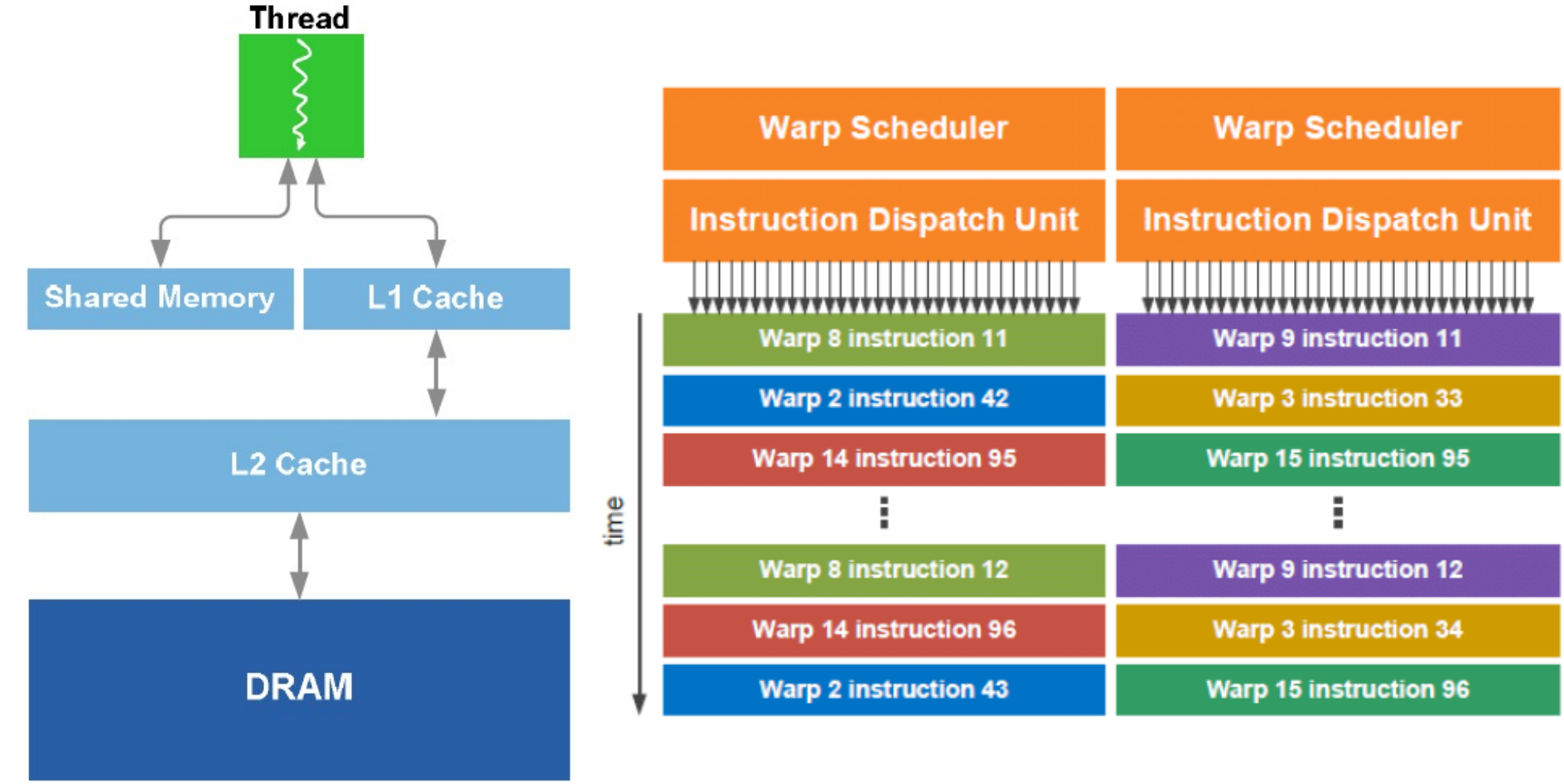
Fermi Architecture Innovations

- 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



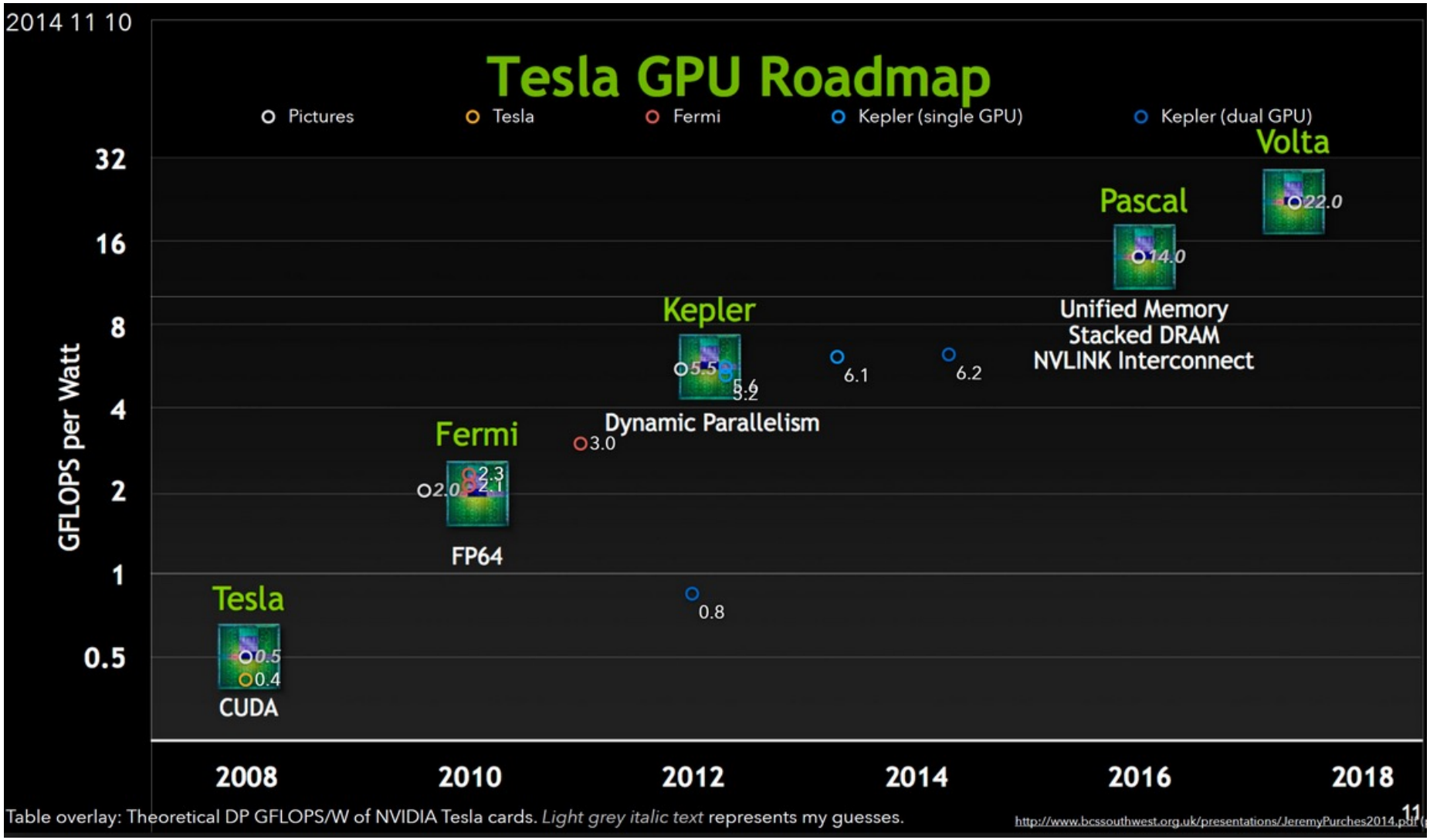
TOP500 list in November 2010: 3 systems in the top4 use Fermi GPUs

HIGHLIGHTS: NOVEMBER 2010

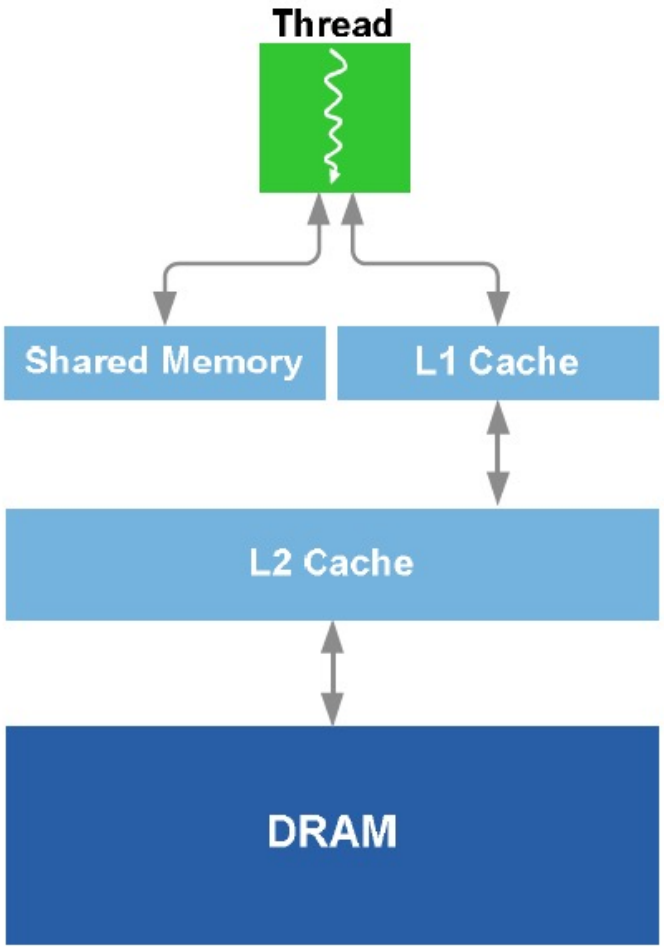
- The Chinese Tianhe-1A system is the new No. 1 on the TOP500 and clearly in the lead with 2.57 petaflop/s performance.
- No. 3 is also a Chinese system called Nebulae, built from a Dawning TC3600 Blade system with Intel X5650 processors and NVIDIA Tesla C2050 GPUs
- There are seven petaflop/s systems 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.

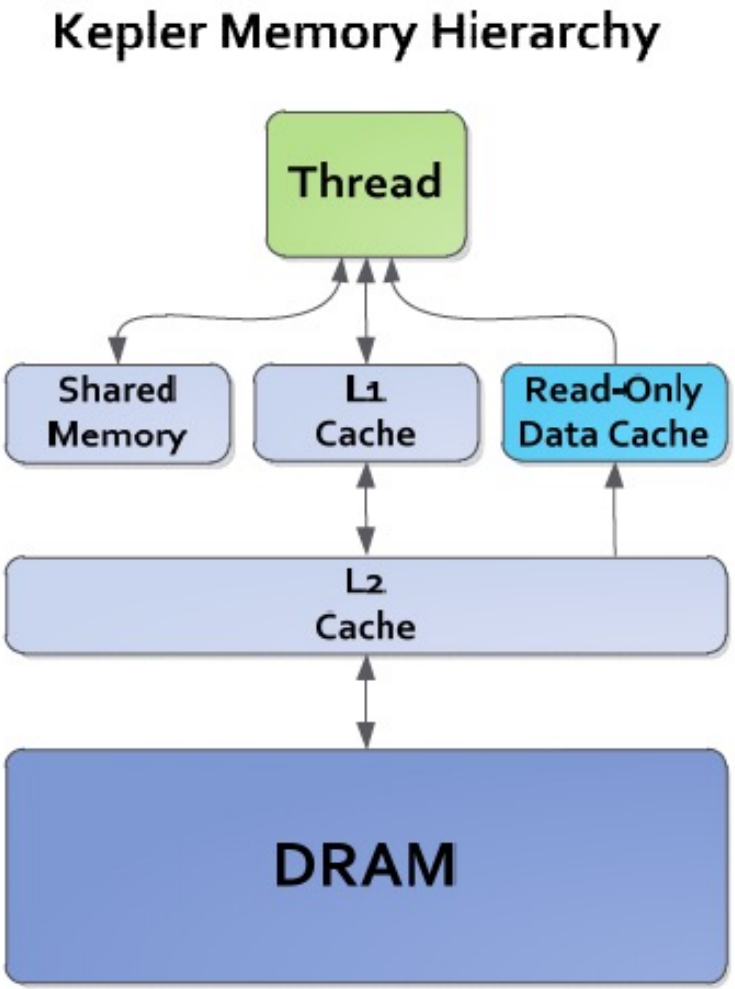
Families in NVidia Tesla GPUs

(up to 2018)

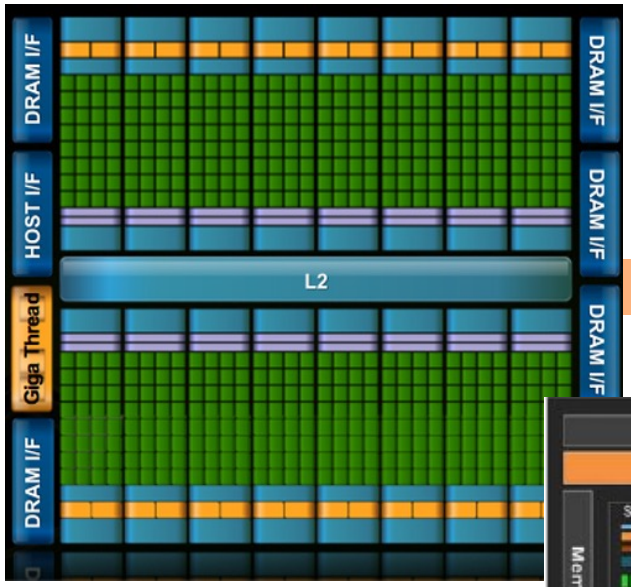


From Fermi into Kepler: the Memory Hierarchy





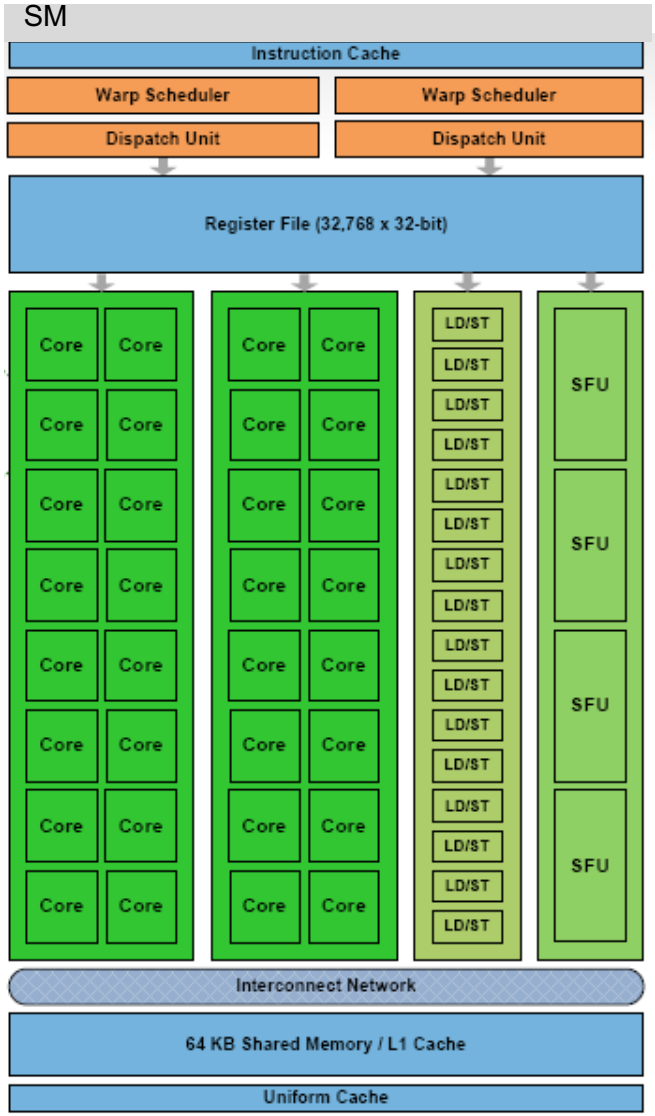
From the GF110 to the GK110 Kepler Architecture



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

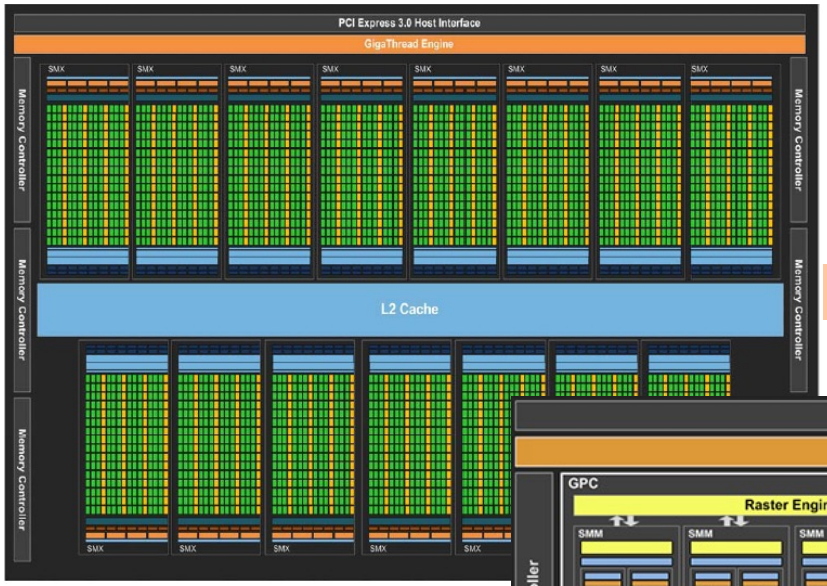


Fermi SM

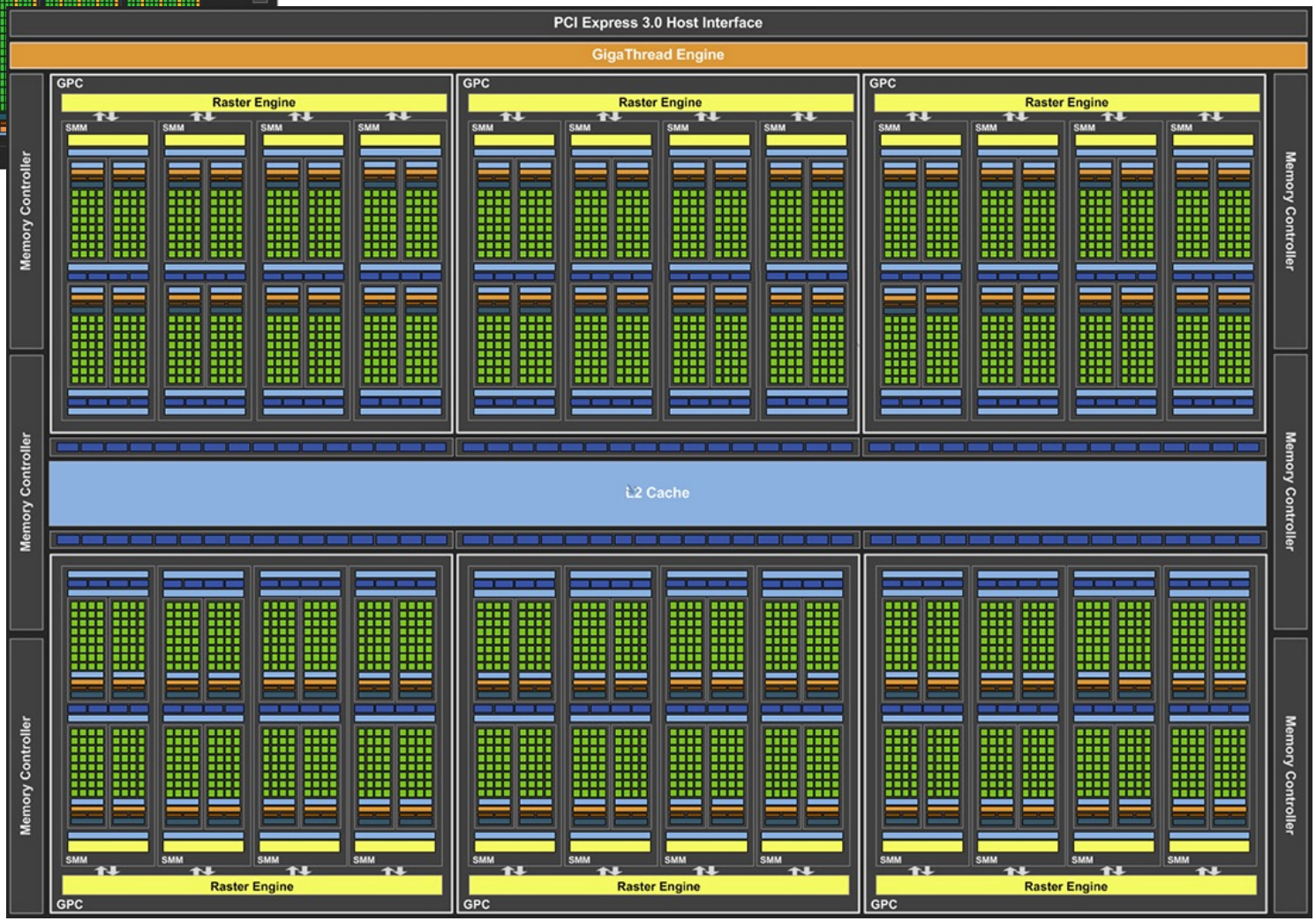
SMX:
192 CUDA-cores

Ratio **DP**unit : **SP**unit —> **1 : 3**

From the GK110 to the GM200 Maxwell Architecture

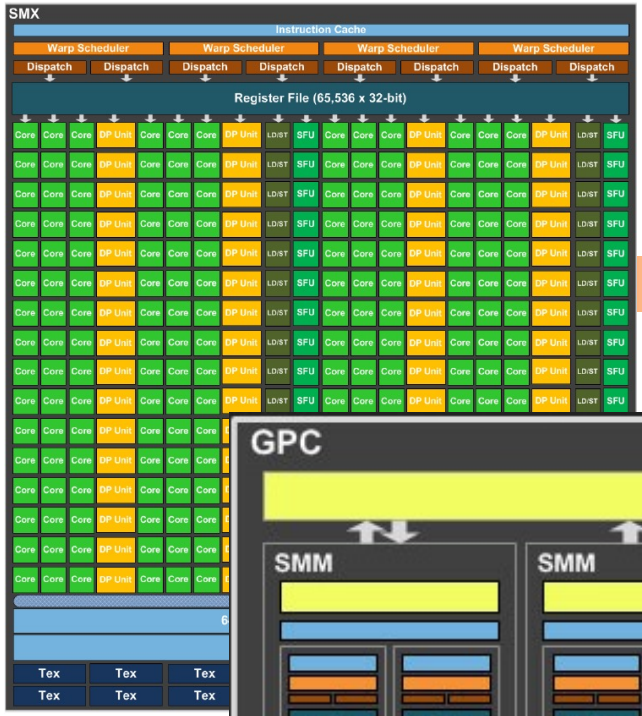


Kepler:
15 SMX
2880 CUDA-cores
October'13



Maxwell:
3072 CUDA-cores
November'15

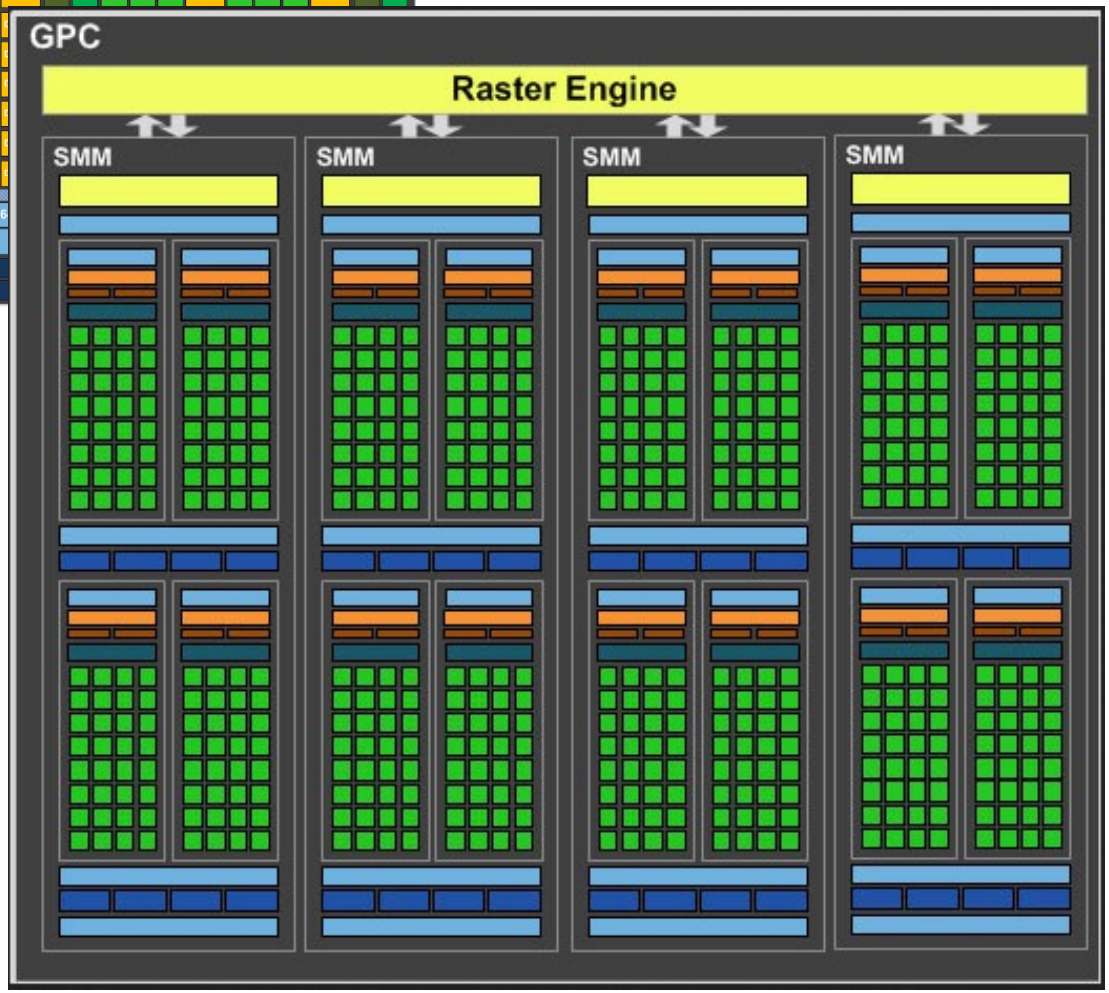
The move from Kepler to Maxwell : from 15 SMXs to 48 SMMs in 6 GPCs

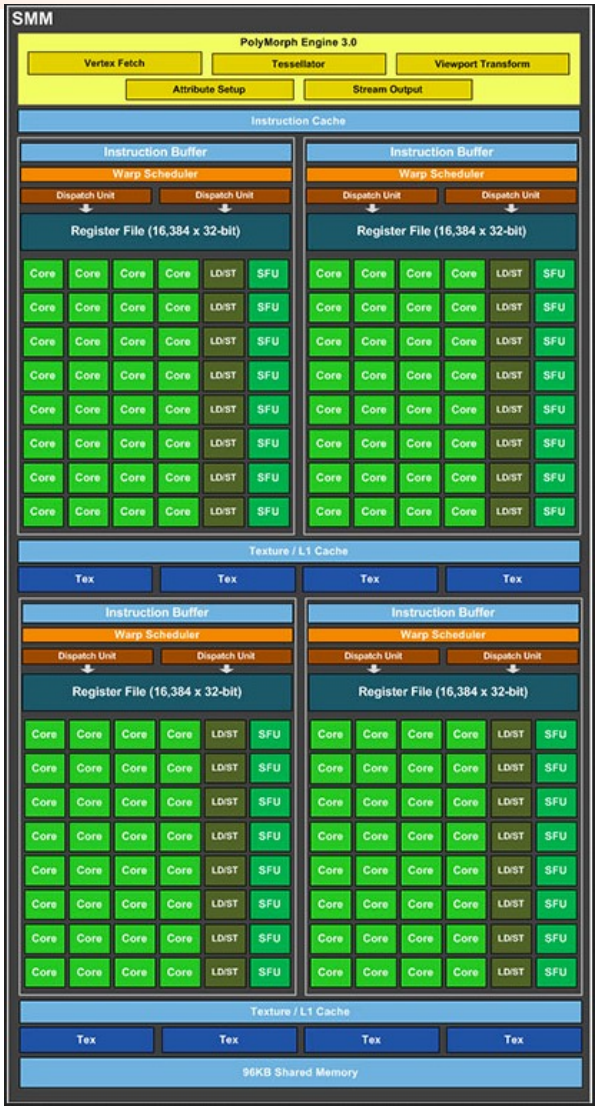


Kepler SMX

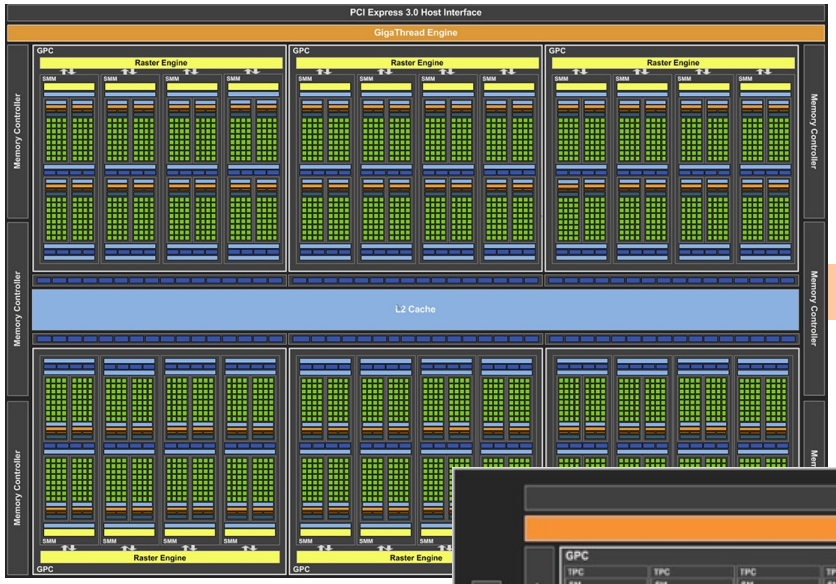
SMM: 128 CUDA-cores

Ratio **DP**unit : **SP**unit —> **1 : 32**





From the M200 to the GP100 Pascal Architecture



Maxwell:
48 SMM
3072 CUDA-cores
November'15

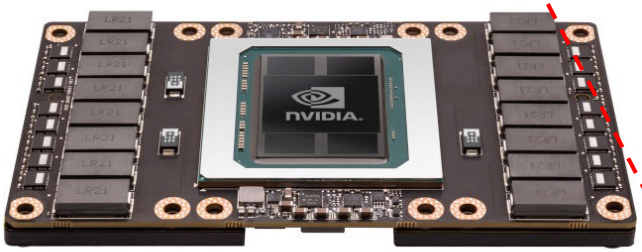
Pascal:
60 SM
3840 CUDA-cores
4 HBM on-package
September'16

Pascal Architecture: 6x GPCs, 60 SMs

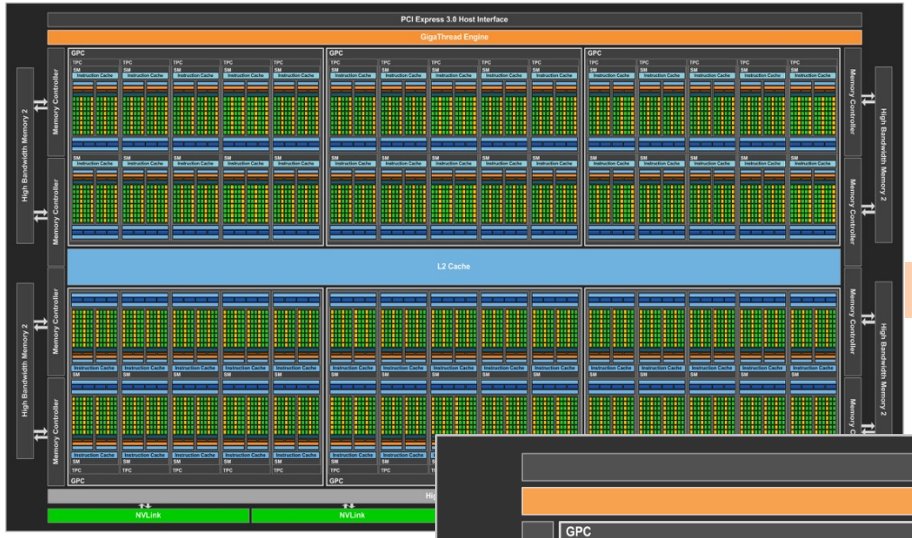
Pascal SM:
64 CUDA-cores

Ratio **DP**unit : **SP**unit —> **1 : 2**

Pascal P100 w/ 16GB HBM2



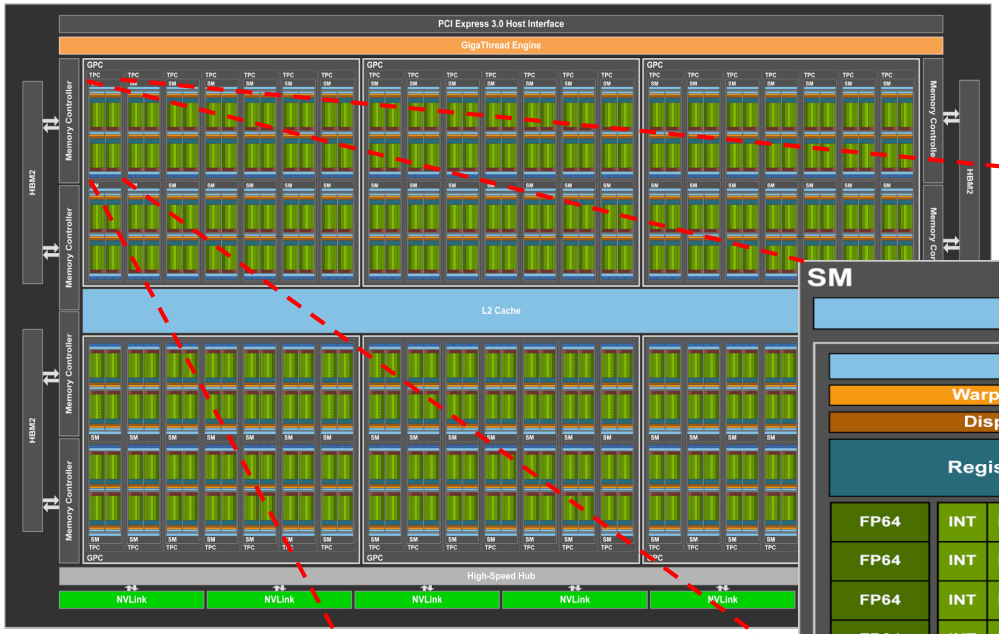
From the GP100 to the GV100 Volta Architecture



Pascal:
60 SM
3840 CUDA-cores
November'15

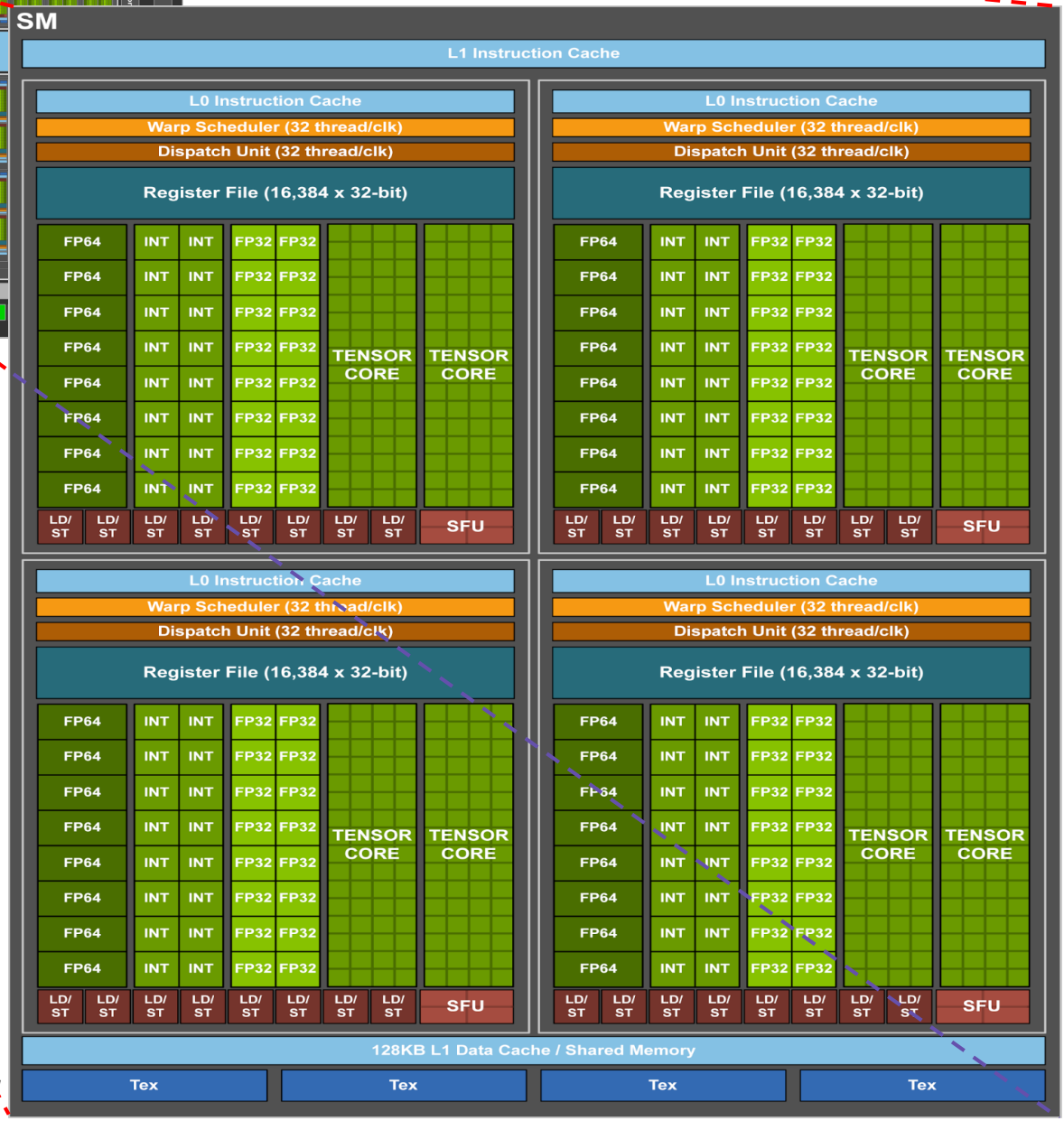
Volta:
84 SM
5120 CUDA-cores
HBM on-package
June'17

Volta Architecture: 6x GPCs, 84 SMs

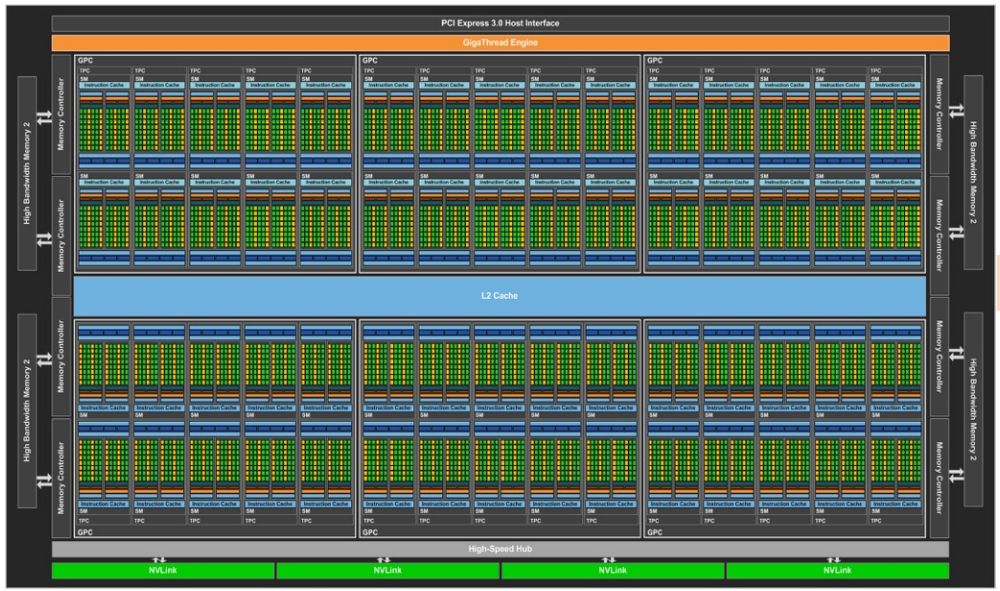


Volta SM:
64 CUDA-cores
New: 8 Tensor-cores

Ratio **DP**unit : **SP**unit —> **1 : 2**



Volta V100 w/ 16GiB HBM2



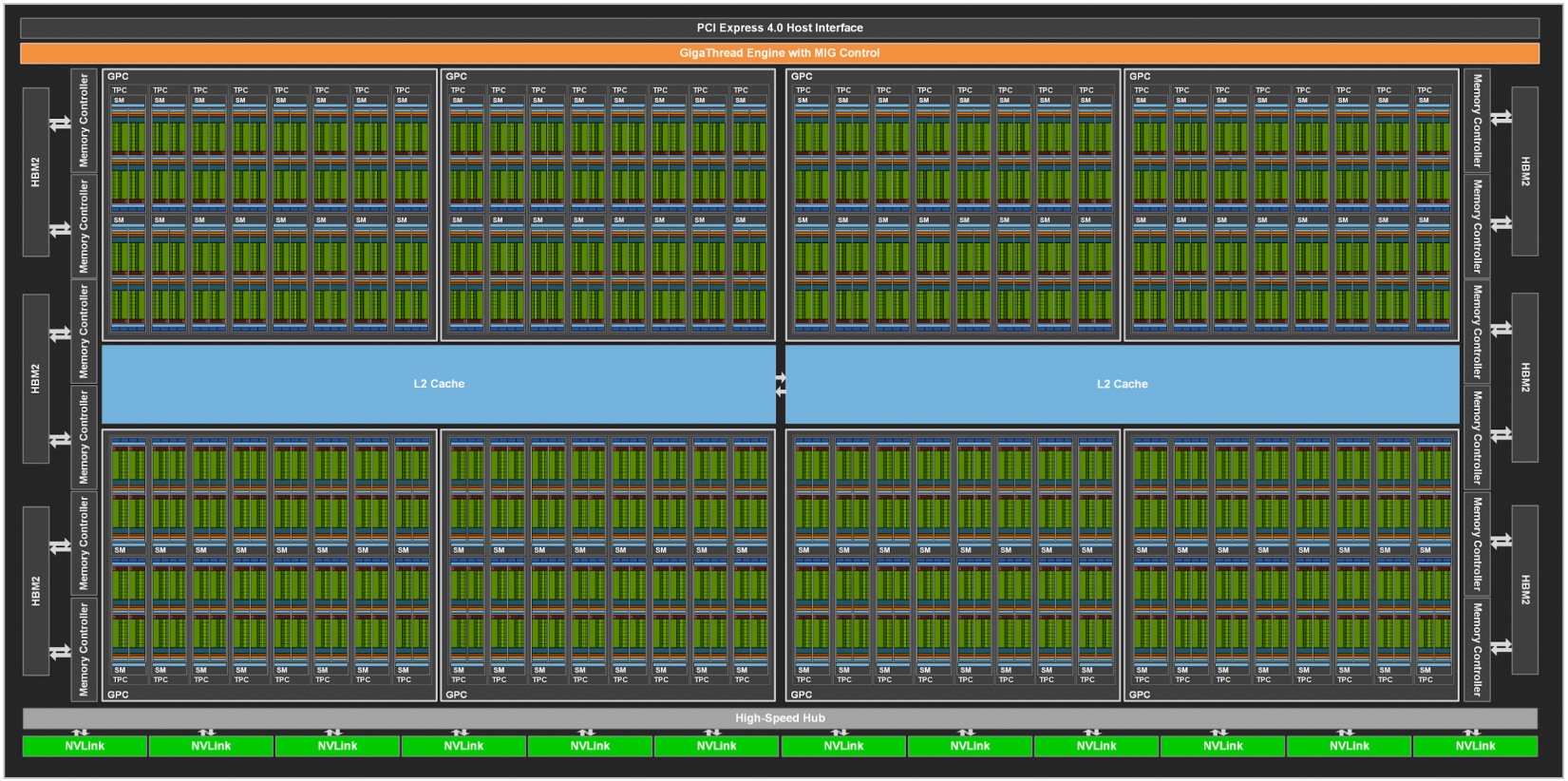
From GV 100 to Ampere: up to 8 GPC, 128 SMs total

Ampere: NVidia GA100
8192 FP32 CUDA Cores
512 3rd generation Tensor Cores
5 HBM2e, 10 512-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

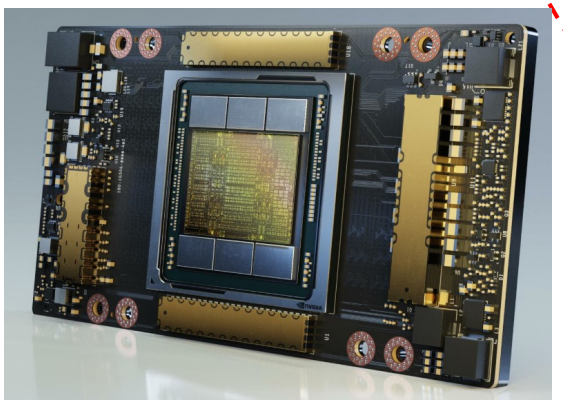
32x FP64 CUDA Cores/SM

4x 3rd generation Tensor Cores

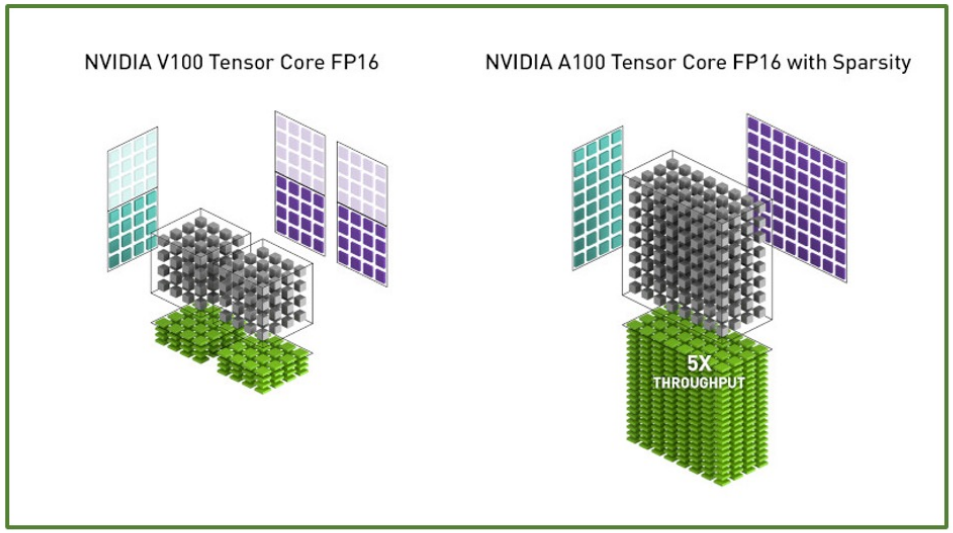
Tensor Cores support

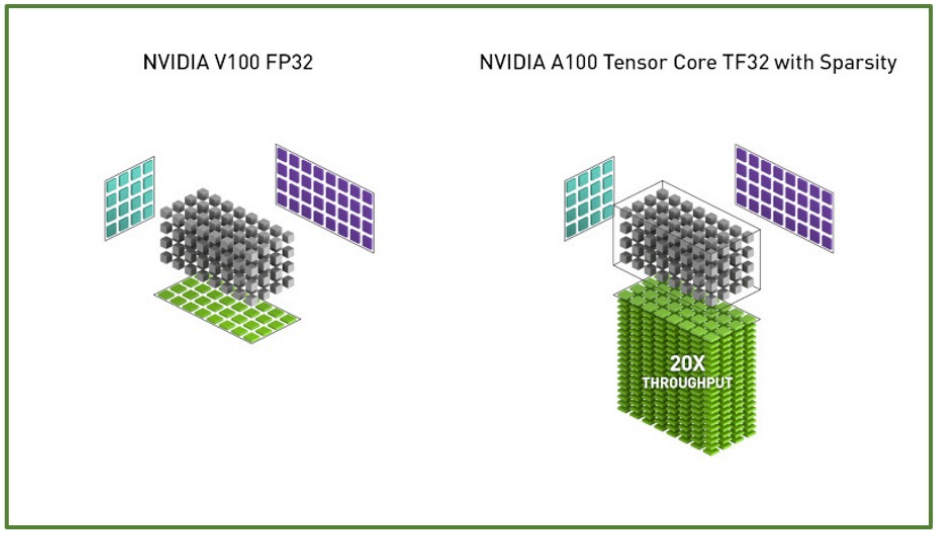
FP64, FP32, TF32, FP16, BF16, INT8…

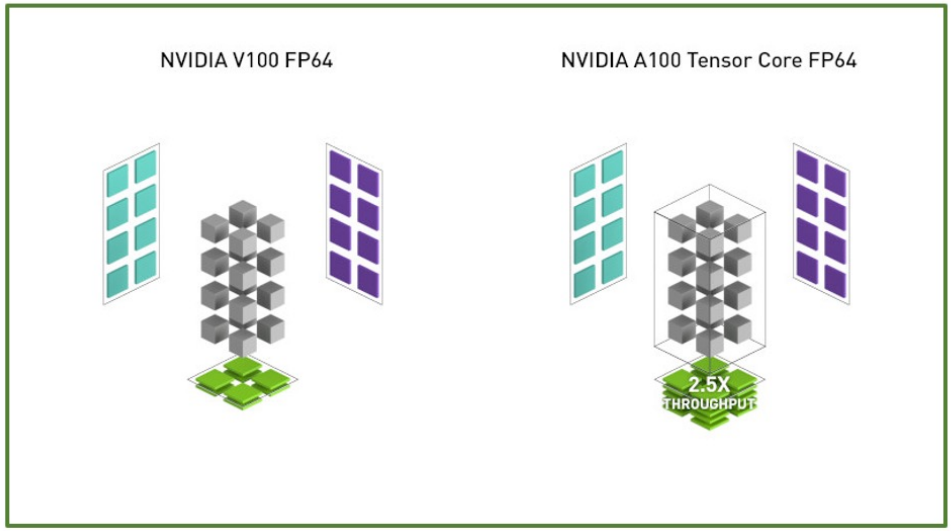
1024 dense FP16/FP32 FMA op's/cycle

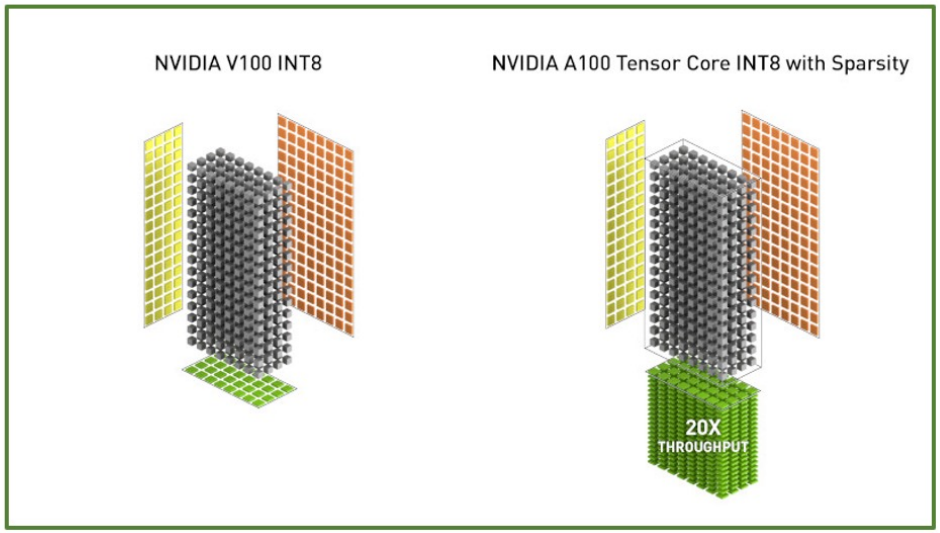


Tensor cores in Ampere

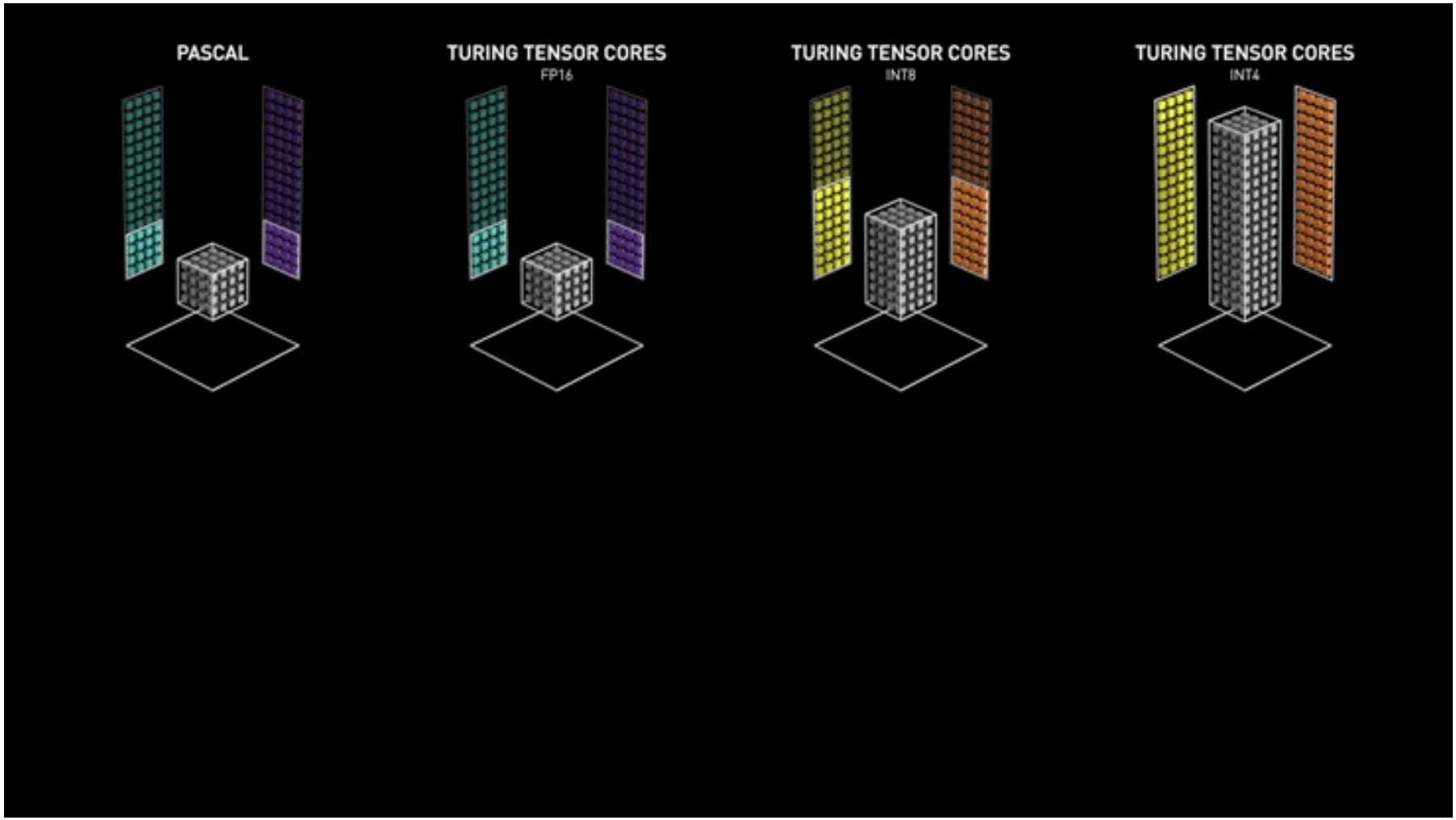








Pascal vs. Turing tensor cores (animation)



Tesla accelerators: evolution

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

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 mm²	601 mm²	610 mm²	815 mm²
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

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	815mm2	826mm2
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

*Effective TOPS / TFLOPS using the new Sparsity feature

https://www.hpcwire.com/2020/05/14/nvidias-ampere-a100-gpu-2-5x-the-hpc-20x-the-training/

The CUDA programming model

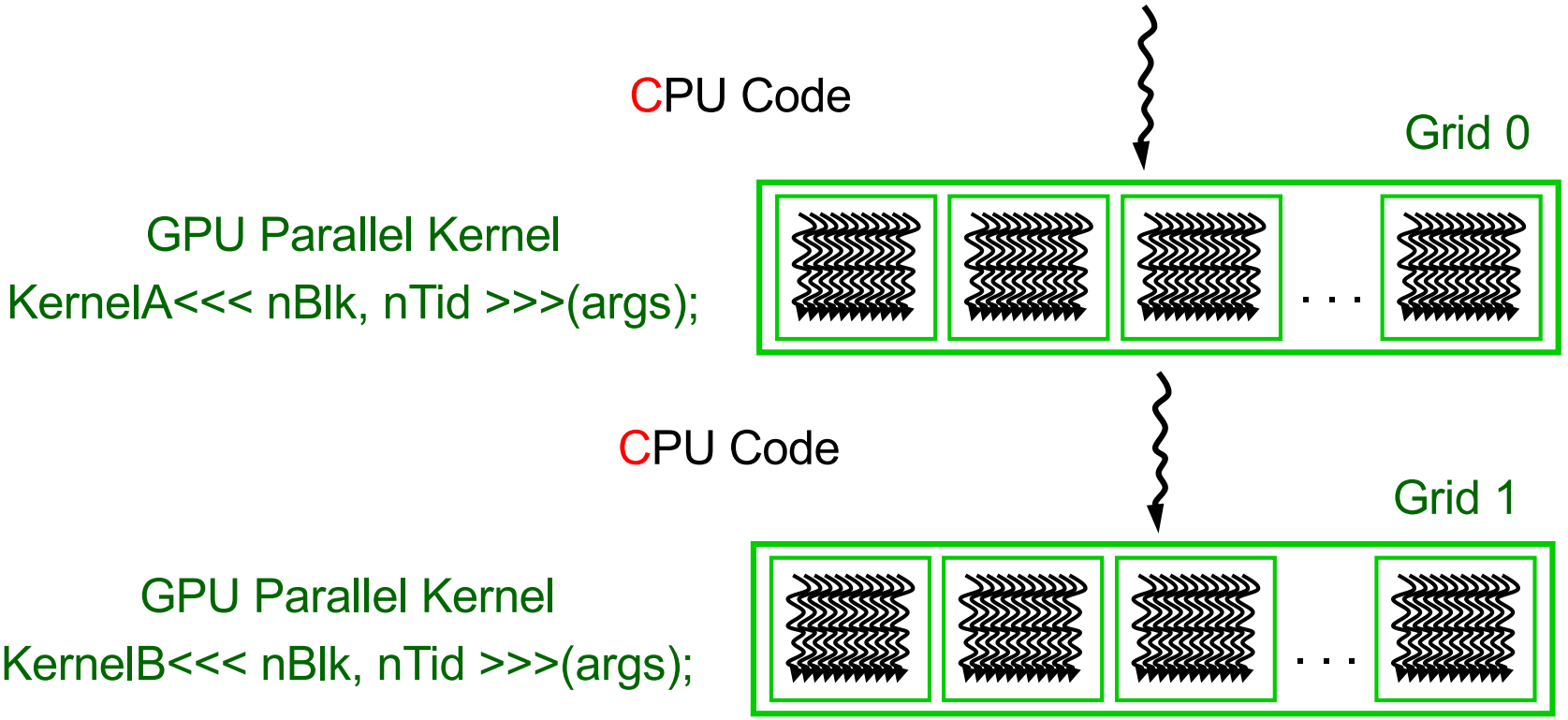
- ***C**ompute **U**nified **D**evice **A**rchitecture*
- CUDA is a recent programming model, designed for
  - a multicore CPU ***host*** coupled to a many-core ***device***, where
  - *devices* have wide SIMD/SIMT parallelism, and
  - the *host* and the *device* do not share memory
- CUDA provides:
  - 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 CPU 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 CPU threads
  - GPU threads are extremely lightweight
    - very little creation overhead, requires LARGE register bank
  - GPU needs 1000s of threads for full efficiency
    - multi-core CPU needs only a few

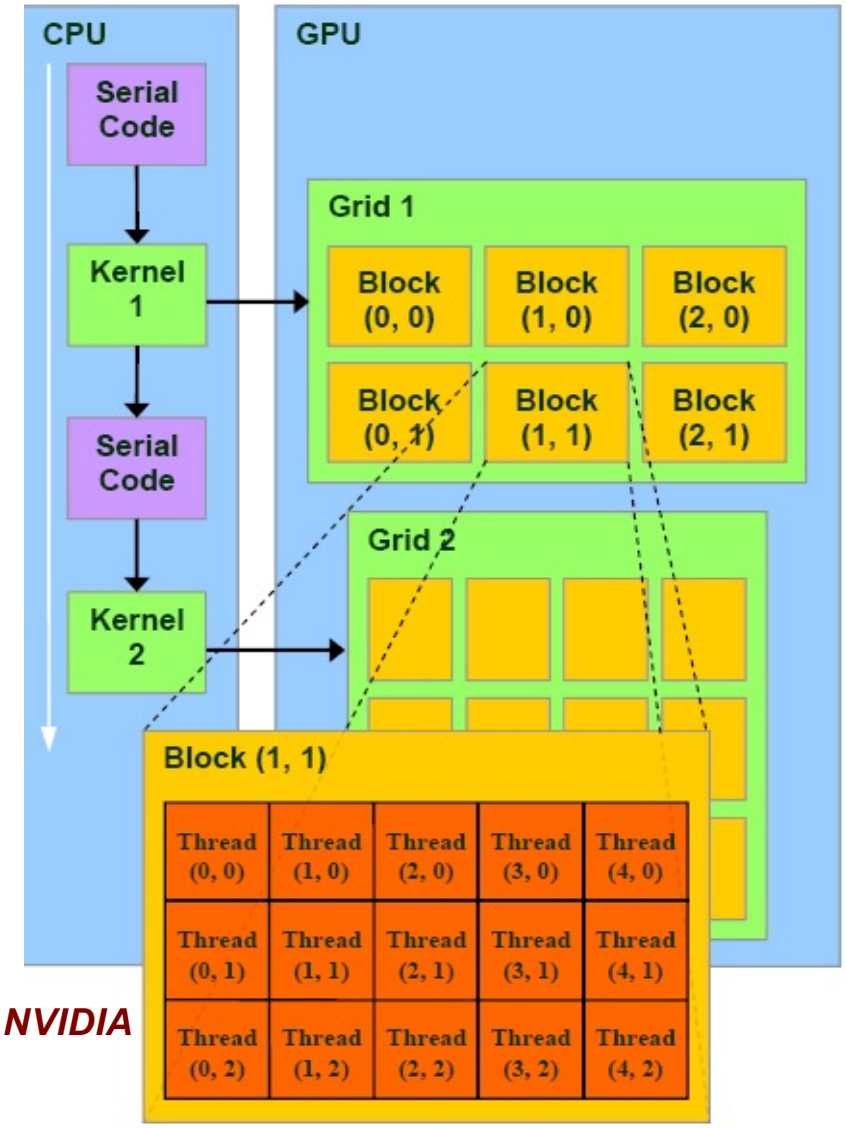
CUDA basic model: Single-Program Multiple-Data (SPMD)

- CUDA integrated CPU + GPU application C program
  - Serial C code executes on CPU
  - Parallel Kernel C code executes on GPU thread blocks



Programming Model: SPMD + SIMT/SIMD

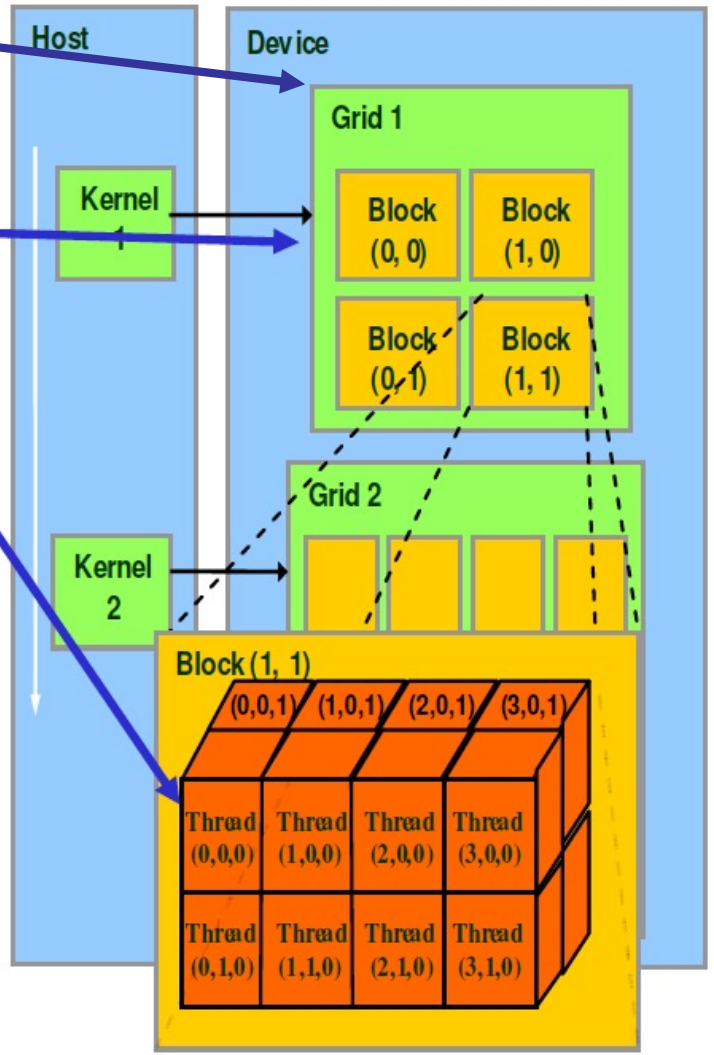
- Hierarchy
  - Device => Grids
  - Grid => Blocks
  - Block => Warps
  - Warp => Threads
- Single kernel runs on multiple blocks (SPMD)
- Threads within a warp are executed in a lock-step way called single-instruction multiple-thread (SIMT)
- Single instruction are executed on multiple threads (SIMD)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



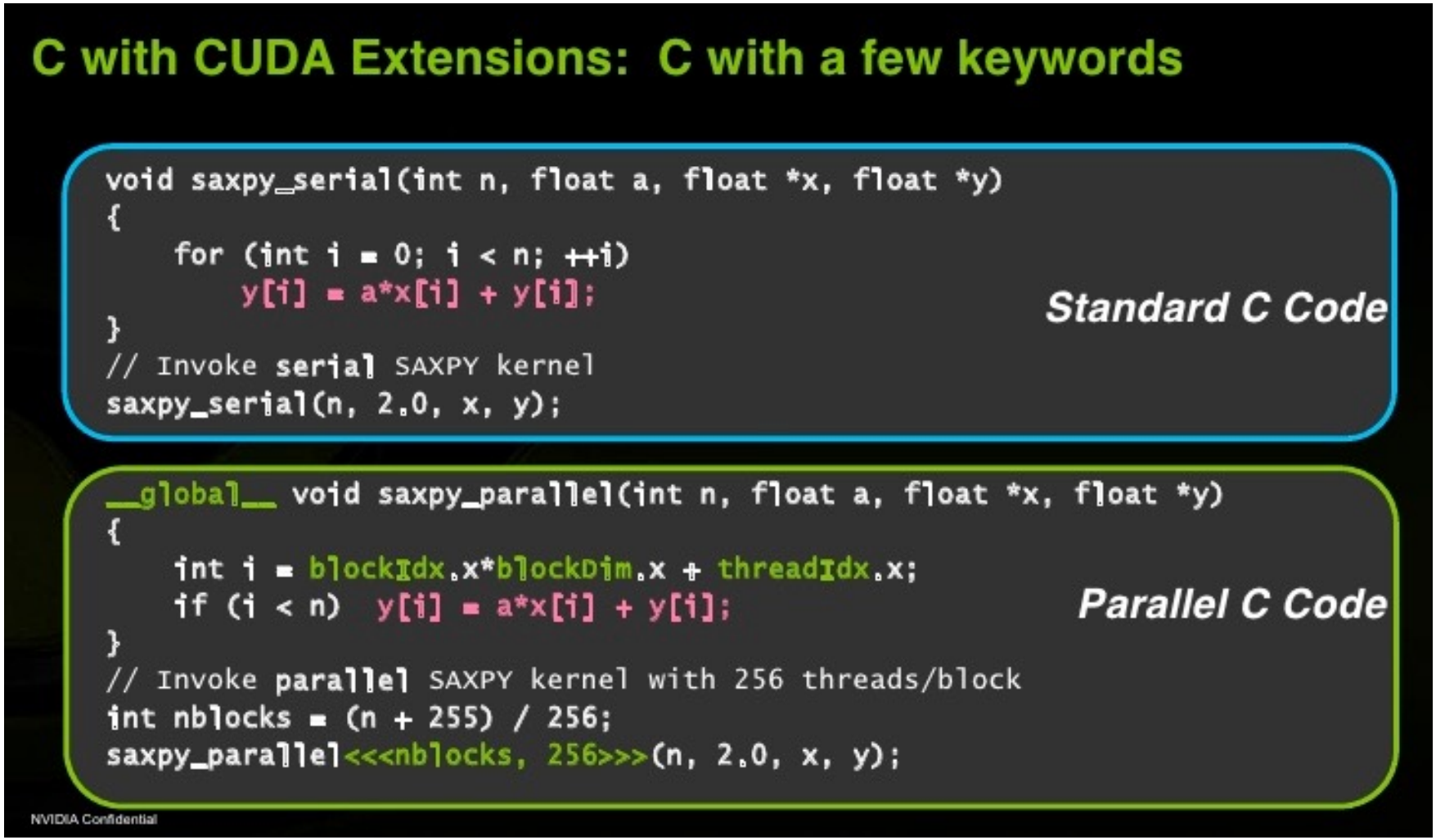
Courtesy NVIDIA

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



Code example

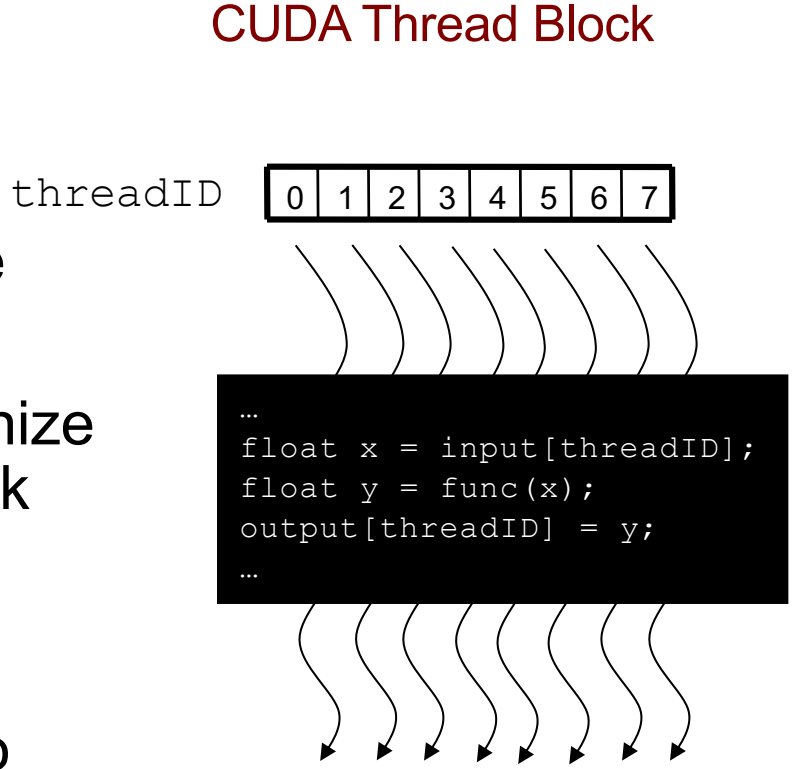


Terminology (and in NVidia)

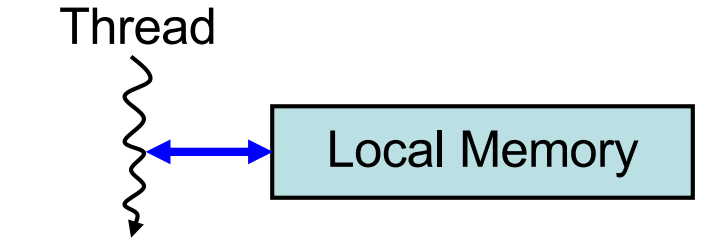
- Threads of SIMD instructions *(**warps**)*
  - Each has its own IP (up to 48/64 per SIMD processor, Fermi/Kepler)
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Threads are organized into blocks & executed in groups of 32 threads *(**thread block**)*
    - 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

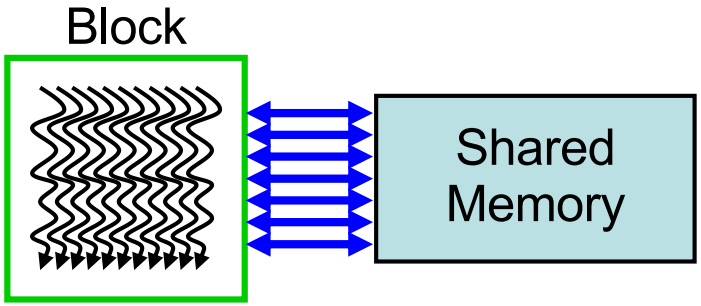
CUDA Thread Block

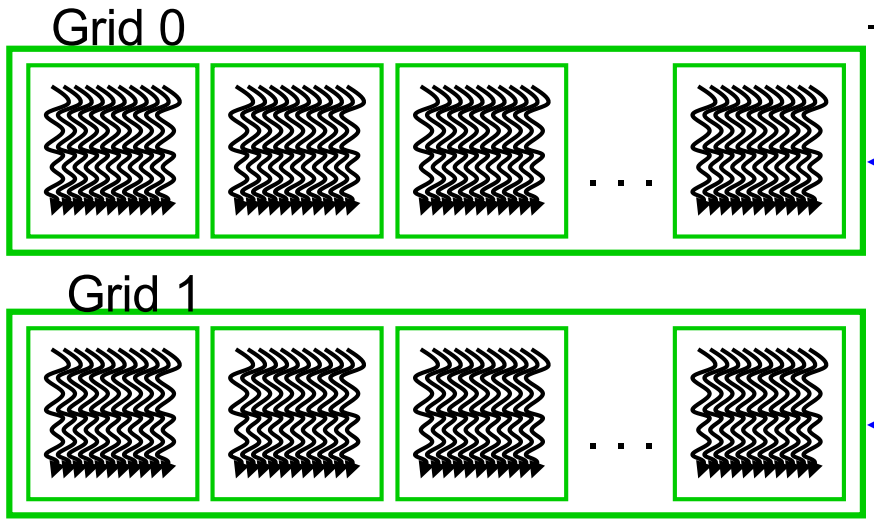
- Programmer declares (Thread) Block:
  - Block size 1 to **512** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have thread id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



Parallel Memory Sharing



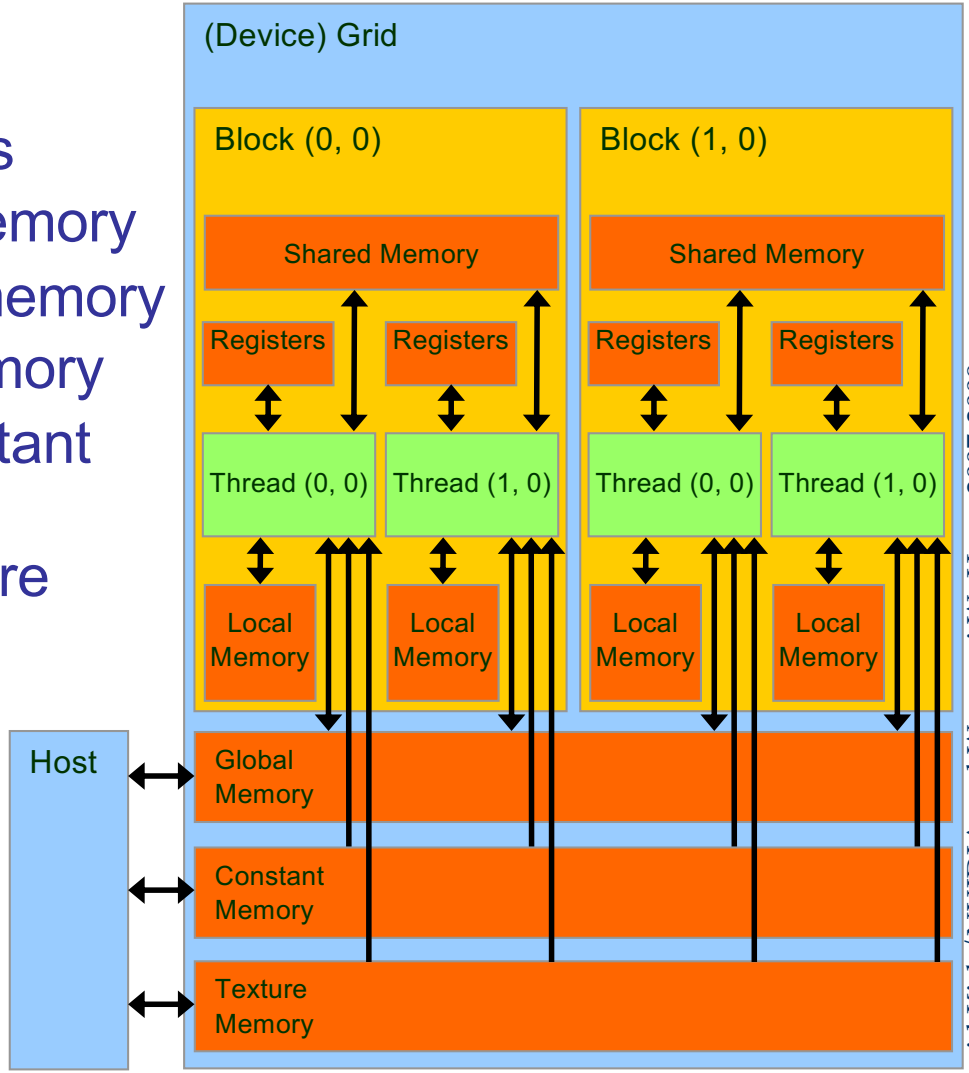




- Local Memory: **per-thread**
  - Private per thread
  - Auto variables, register spill
- Shared Memory: **per-block**
  - Shared by threads of the same block
  - Inter-thread communication
- Global Memory: **per-application**
  - Shared by all threads
  - Inter-Grid communication

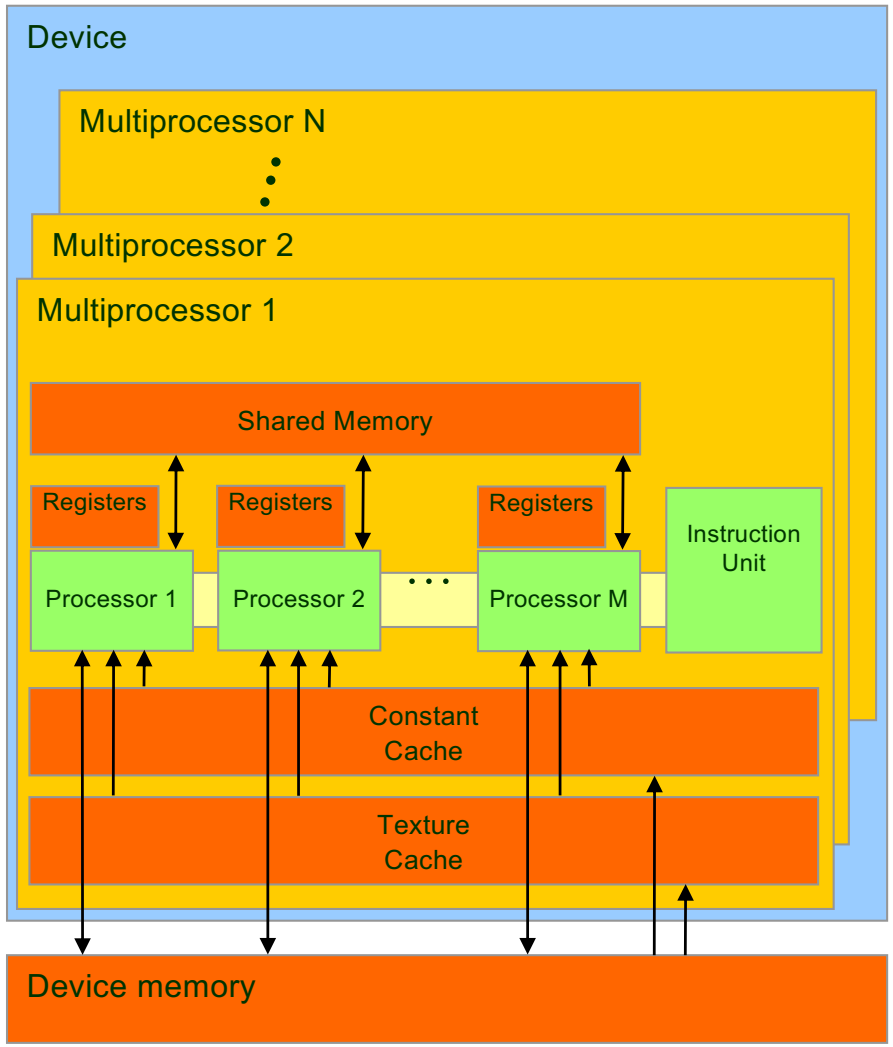
CUDA Memory Model Overview

- 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