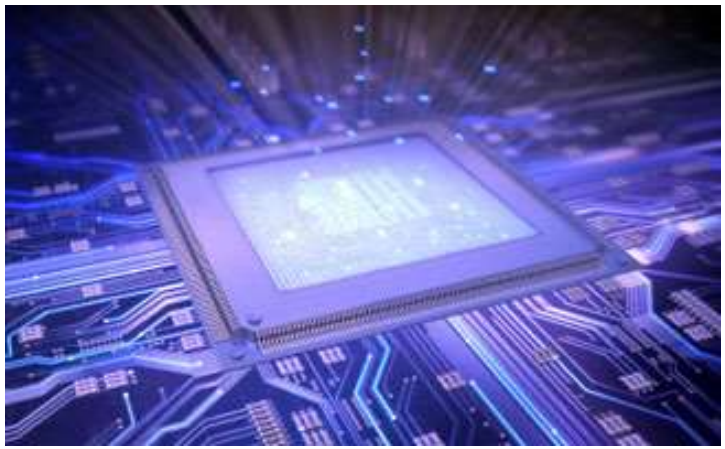
NYU
New York University

Parallel Computing

GPUs – What? Why? How?

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Three Main Goals for Current Architectures

- Maintain execution speed of old sequential programs
- Increase throughput of parallel programs
- Reduce execution time of moderately or non-GPU-friendly parallel programs.

Three Main Goals for Current Architectures

- Maintain execution speed of old sequential programs

CPU

- Increase throughput of parallel programs

GPU

- Reduce execution time of moderately or non-GPU-friendly parallel programs.

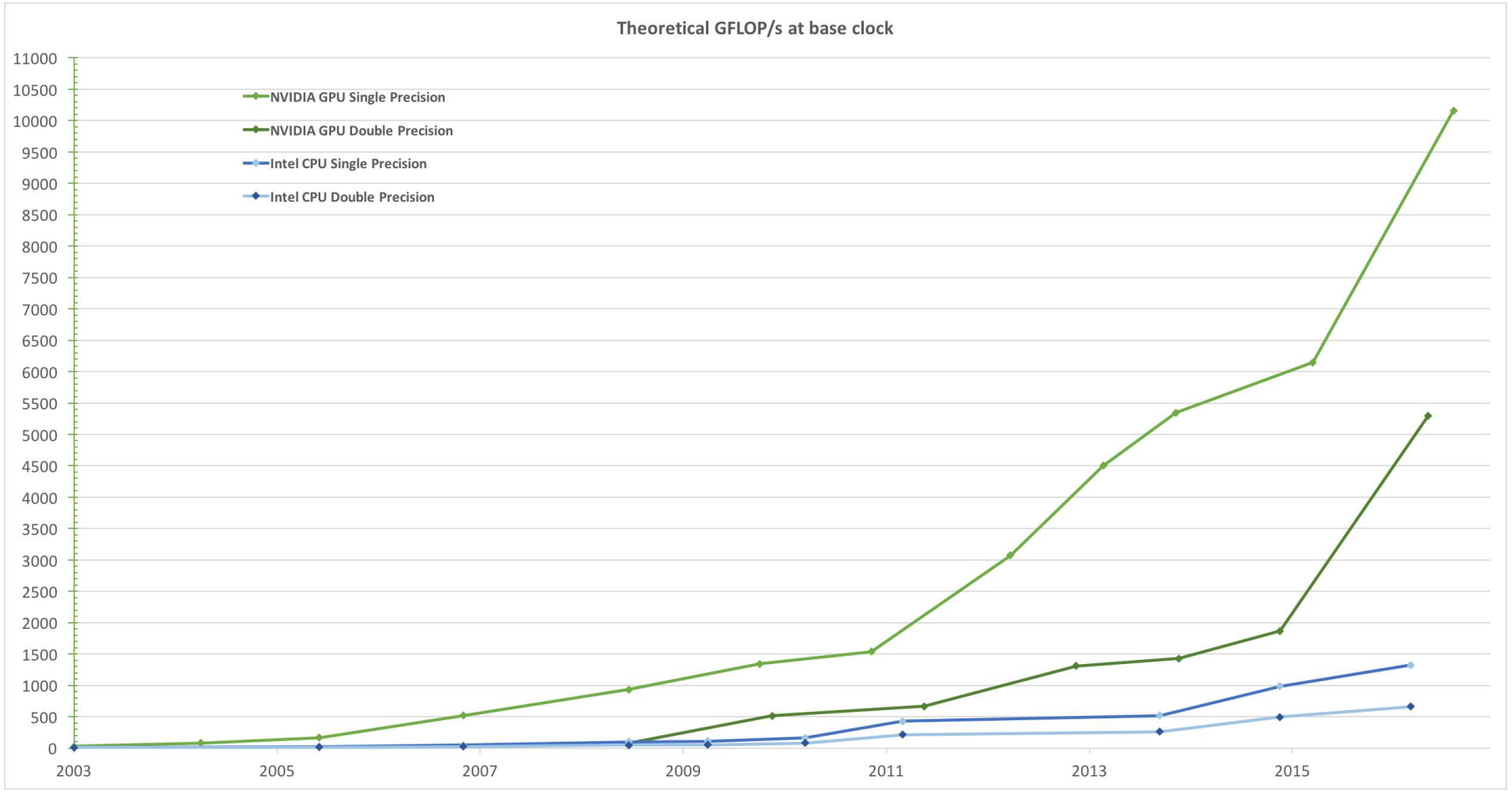
Multicore

Winning Applications Use Both CPU and GPU

- CPUs for sequential parts, or low or non-date-level parallelism, where latency matters
  - CPUs can be 10X+ faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  - GPUs can be 10X+ faster than CPUs for parallel GPU-friendly code

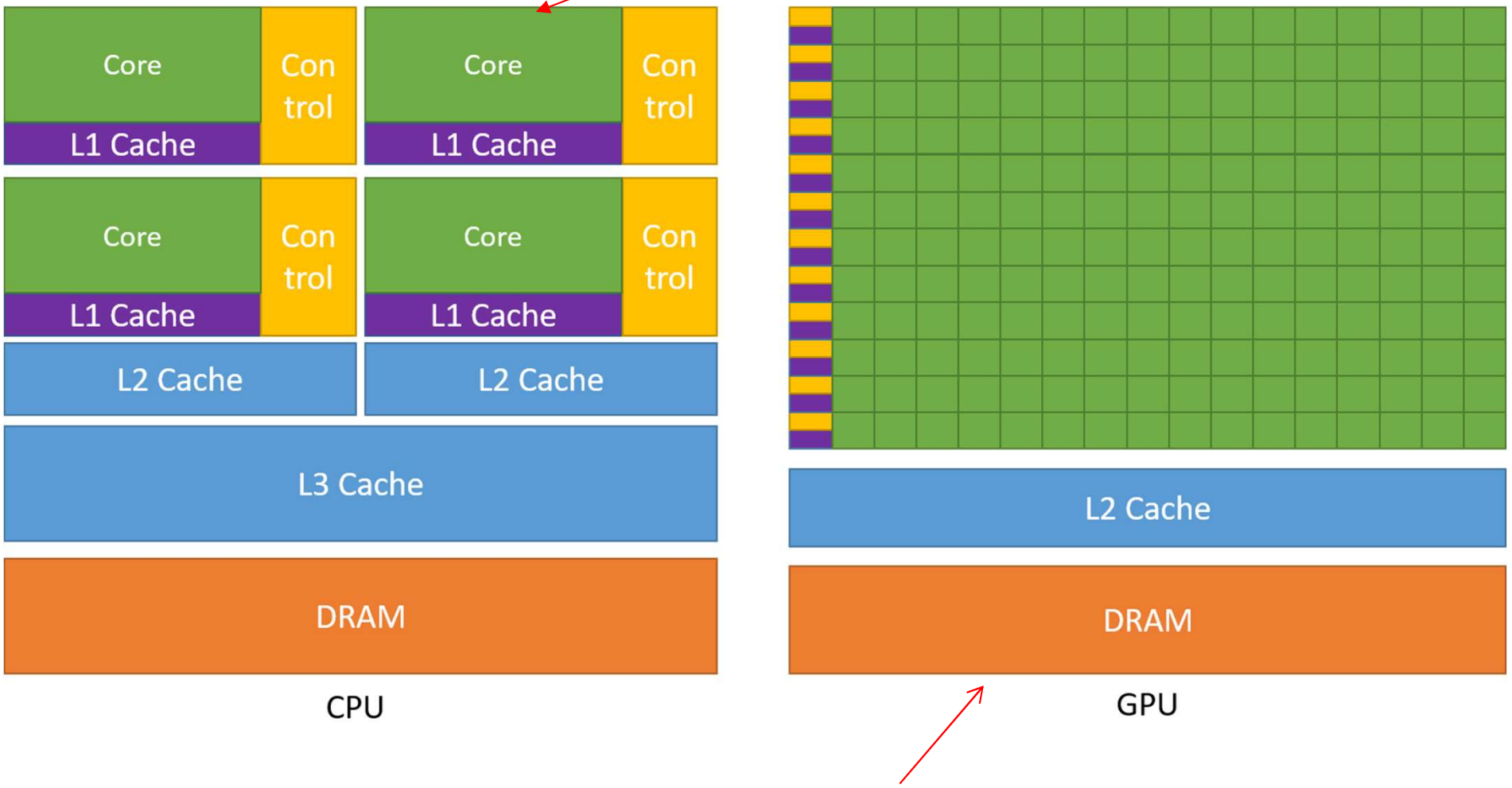
Source: NVIDIA GPU teaching kit

Performance



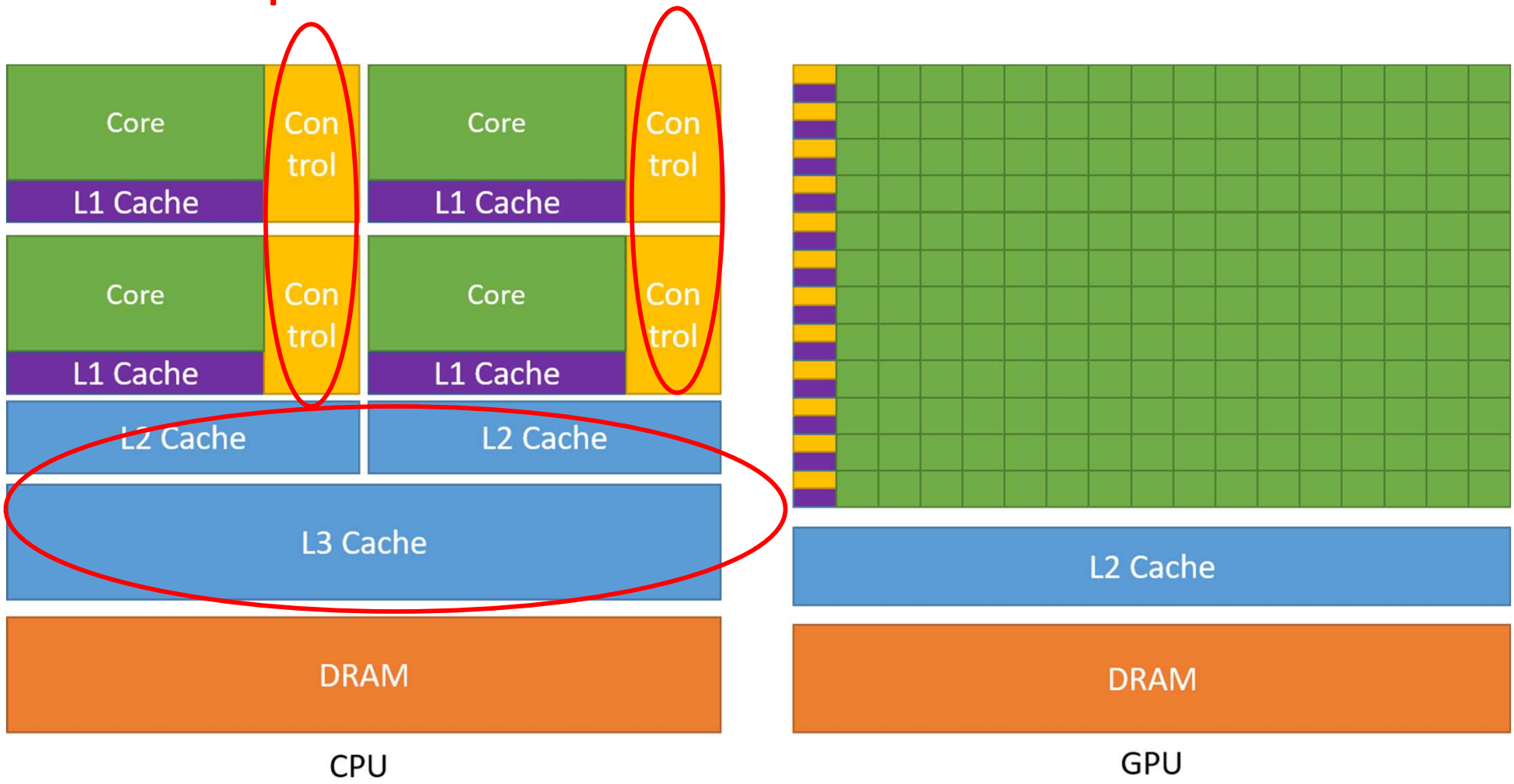
Source: NVIDIA CUDA C Programming Guide

GPUs have way more processing power than multicore



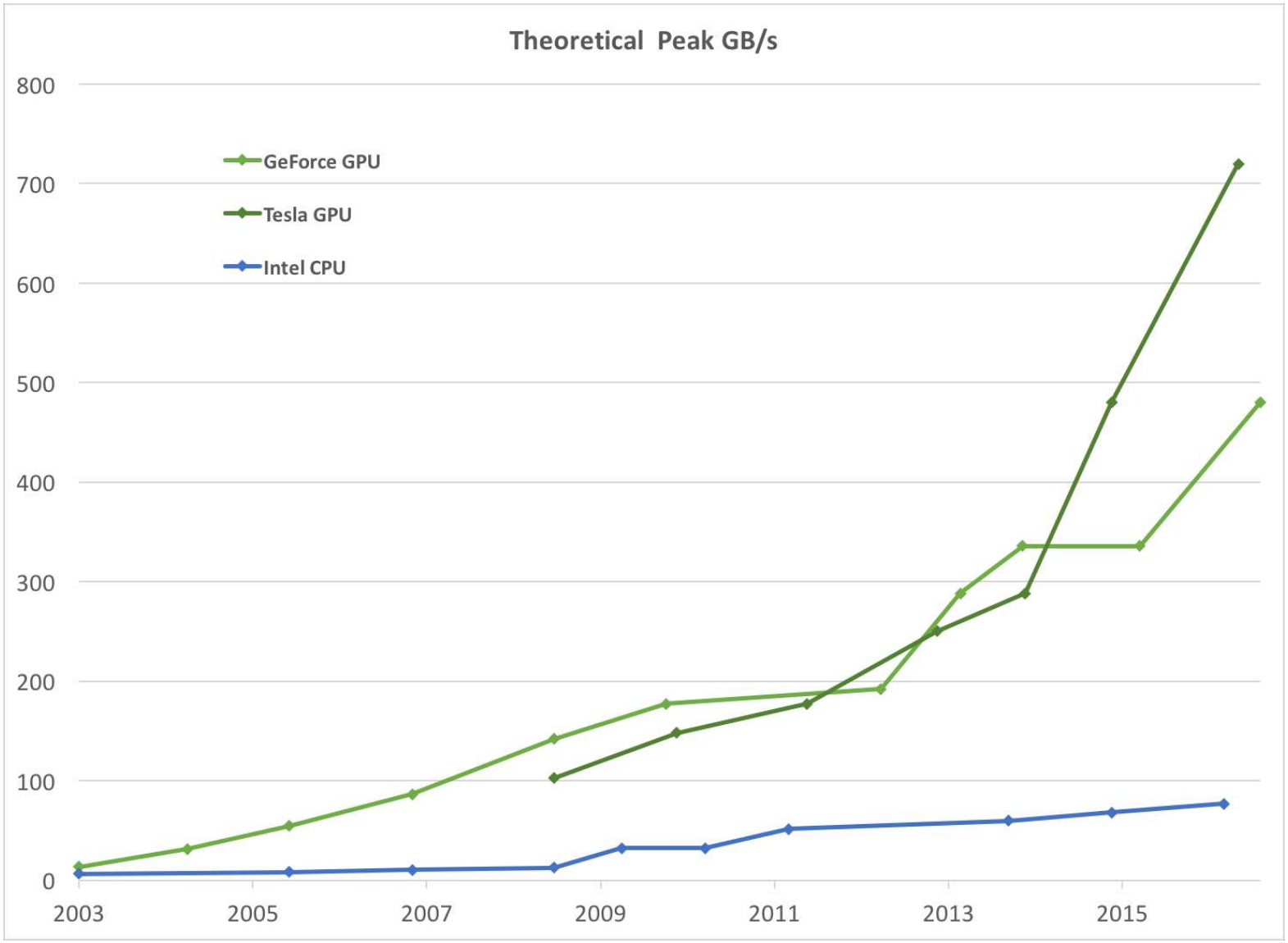
Almost 10x the bandwidth of multicore memory

CPU is optimized for sequential code performance



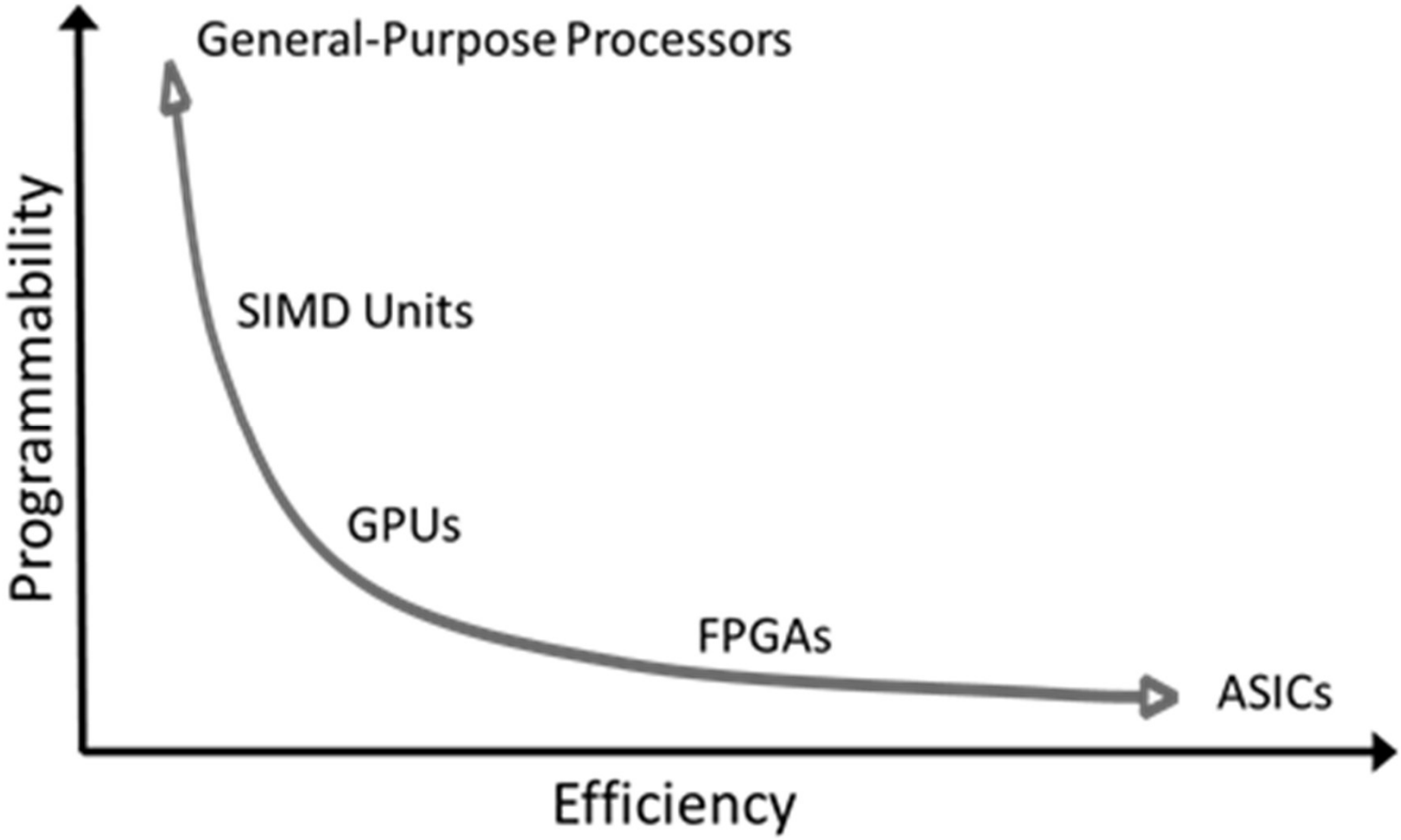
Source: NVIDIA C++ Programming Guide

Memory Bandwidth

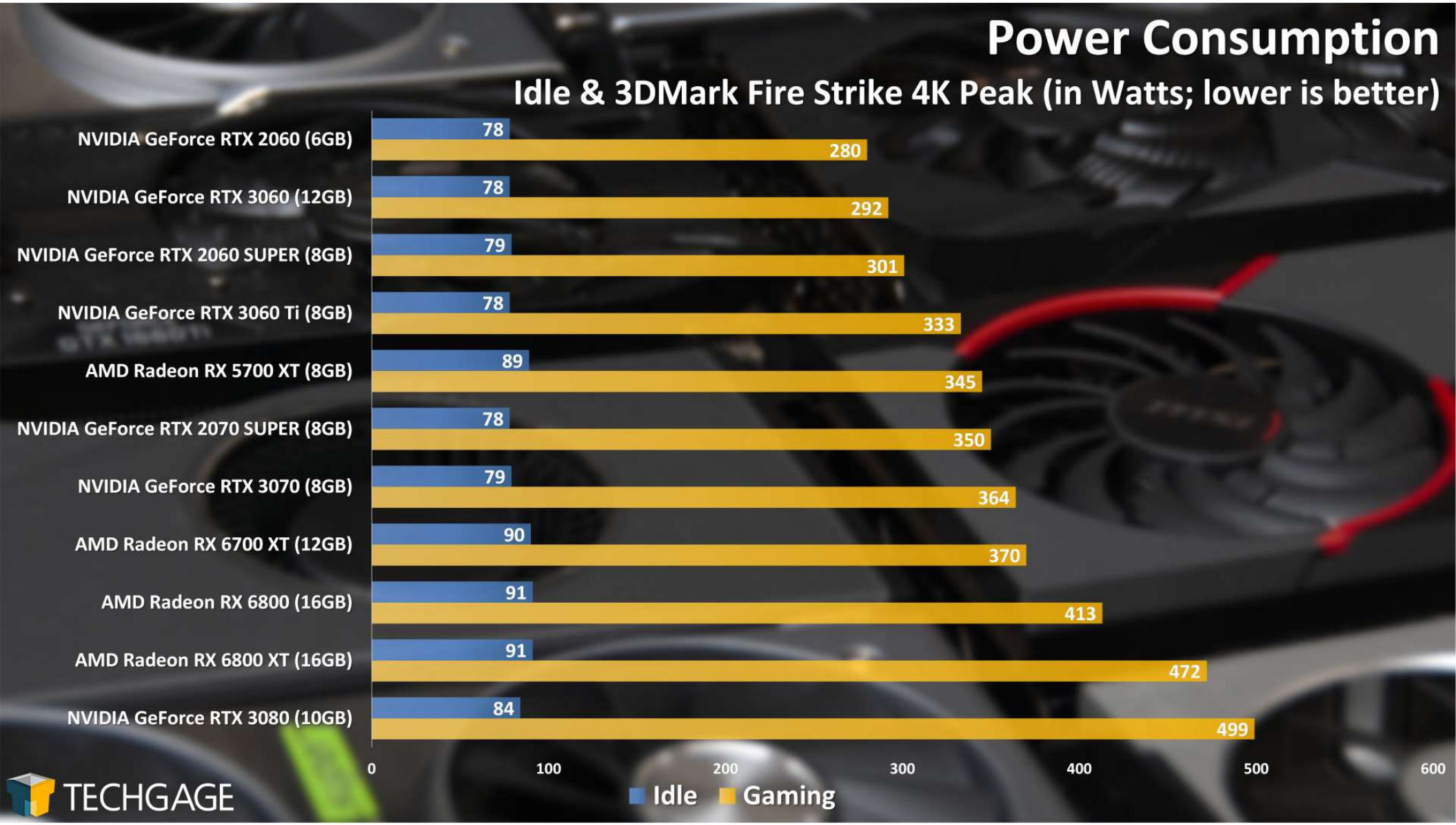


Source: NVIDIA CUDA C Programming Guide

Where do GPU stand among other chips?



Problem With GPUs: Power



Source: https://techgage.com/article/1080p-1440-gaming-geforce-radeon-performance/3/

What are GPUs good for?

Regularity + Massive Parallelism







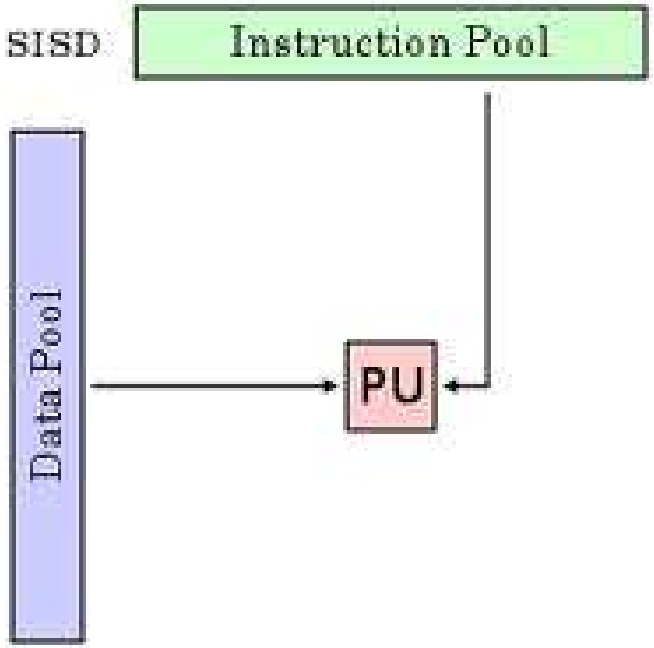

Is Any Application Suitable for GPU?

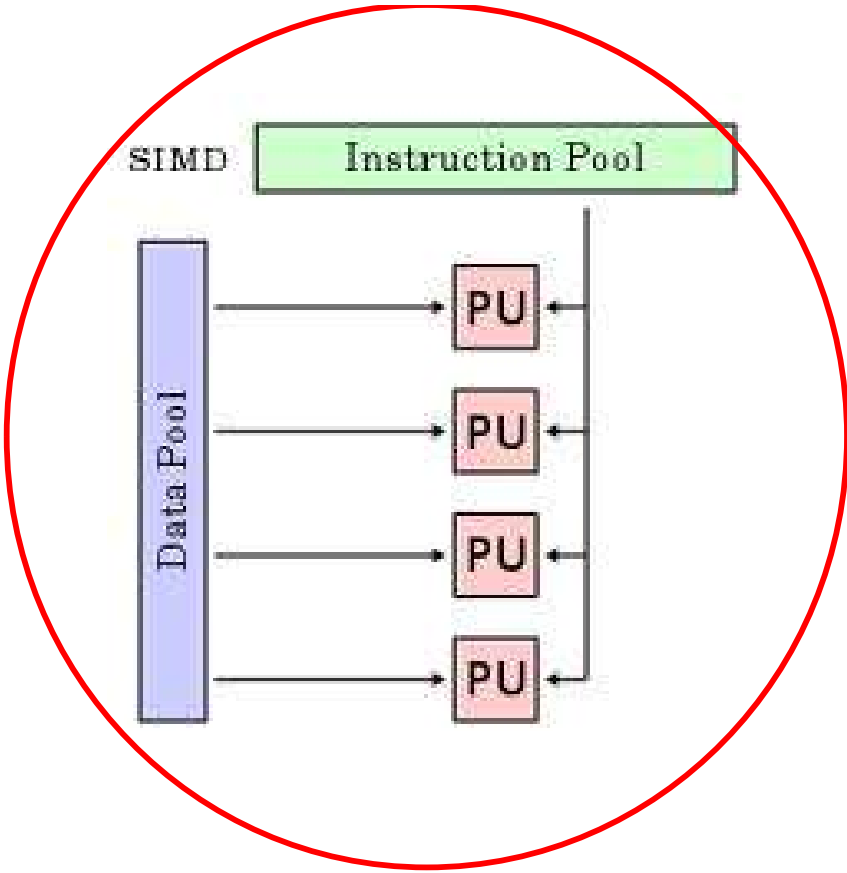
- No!
- You will get the best performance from GPU if your application is:
  - Computation intensive
  - Many independent computations
  - Many similar computations
  - Problem size is big enough

Let's Remember Flynn Classification

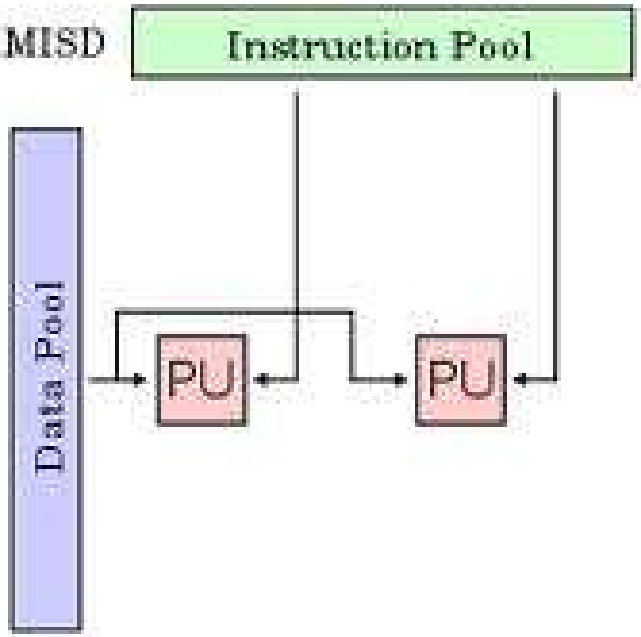
- A taxonomy of computer architecture
- Proposed by Micheal Flynn in 1966
- It is based two things:
  - Instructions
  - Data

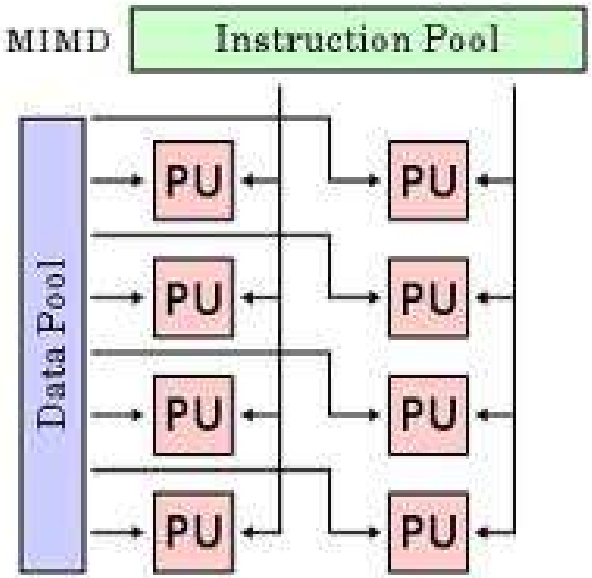
	Single instruction	Multiple instruction
Single data	SISD	MISD
Multiple data	SIMD	MIMD





PU = Processing Unit

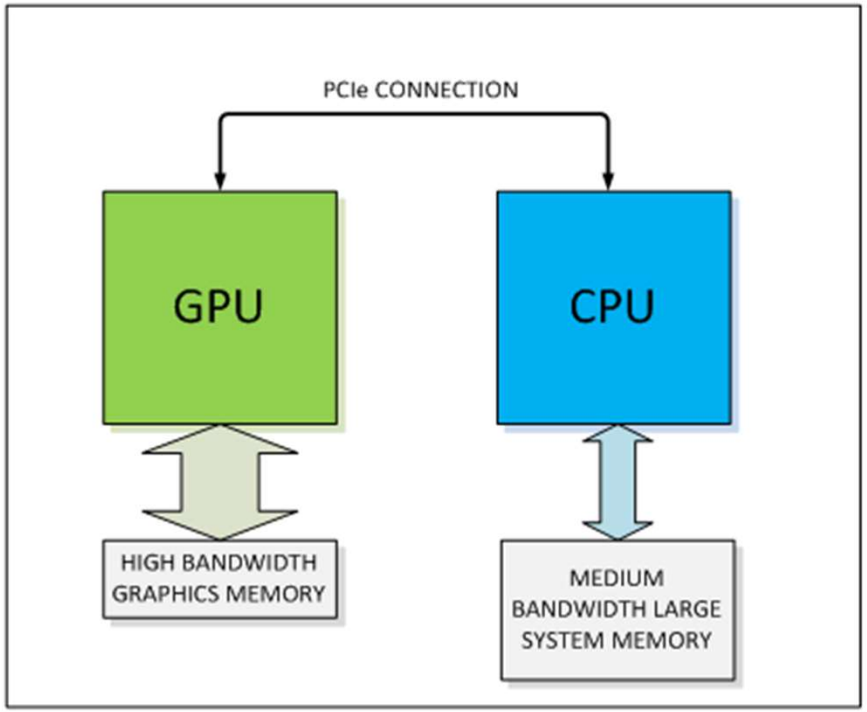


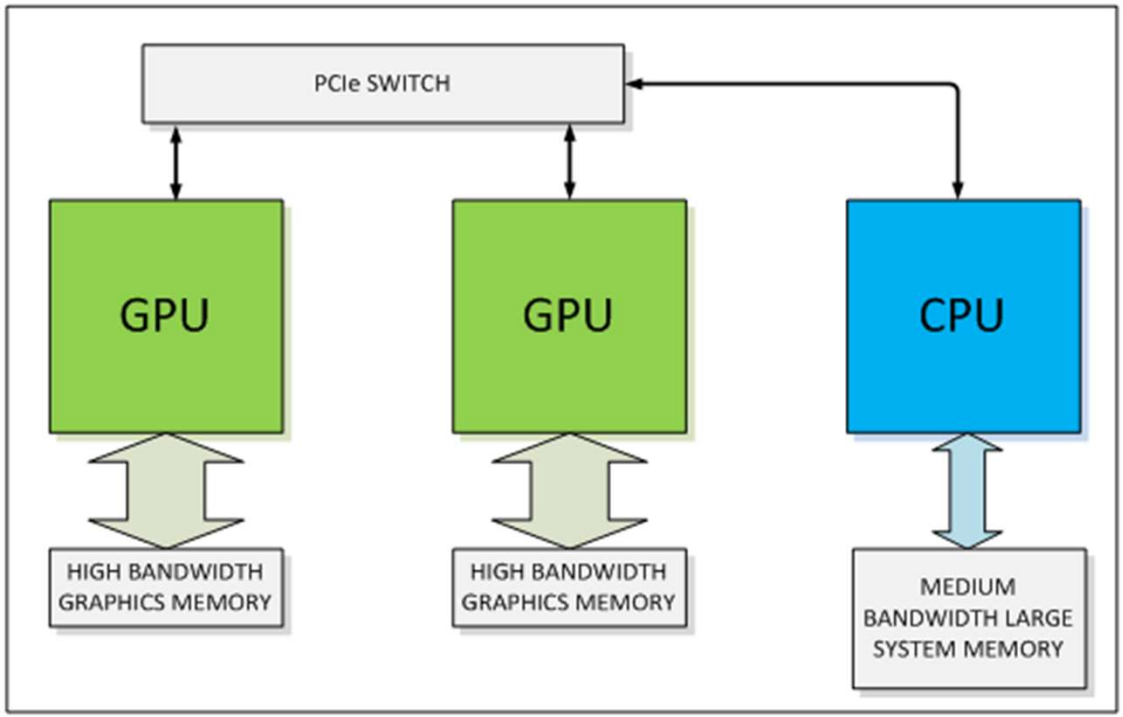


Problems Faced by GPUs

- Not enough parallelism
- Under-utilization
- Memory access
- Bandwidth to multicore memory
  - GPUs need to get the data from multicore memory and put them in GPU memory before processing them.

Let's Take A Closer Look: The Hardware

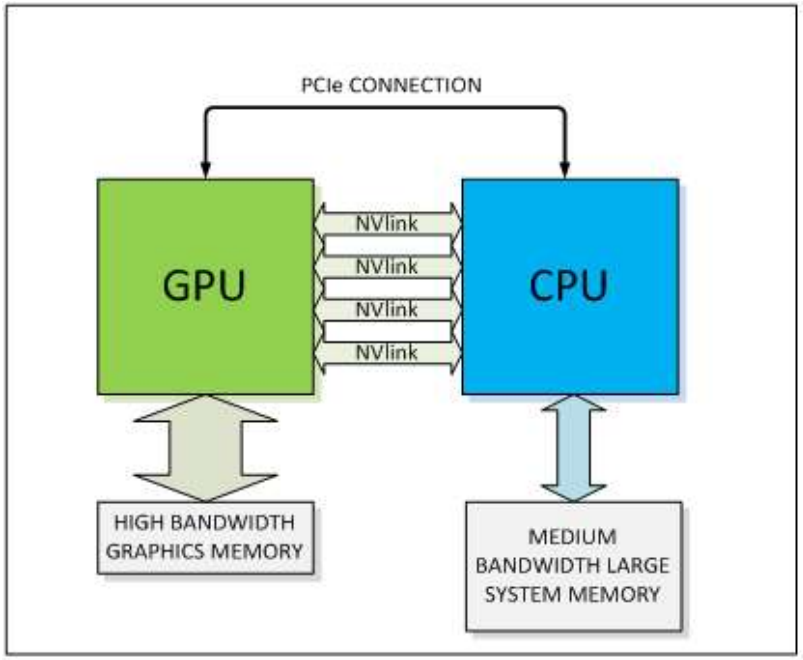


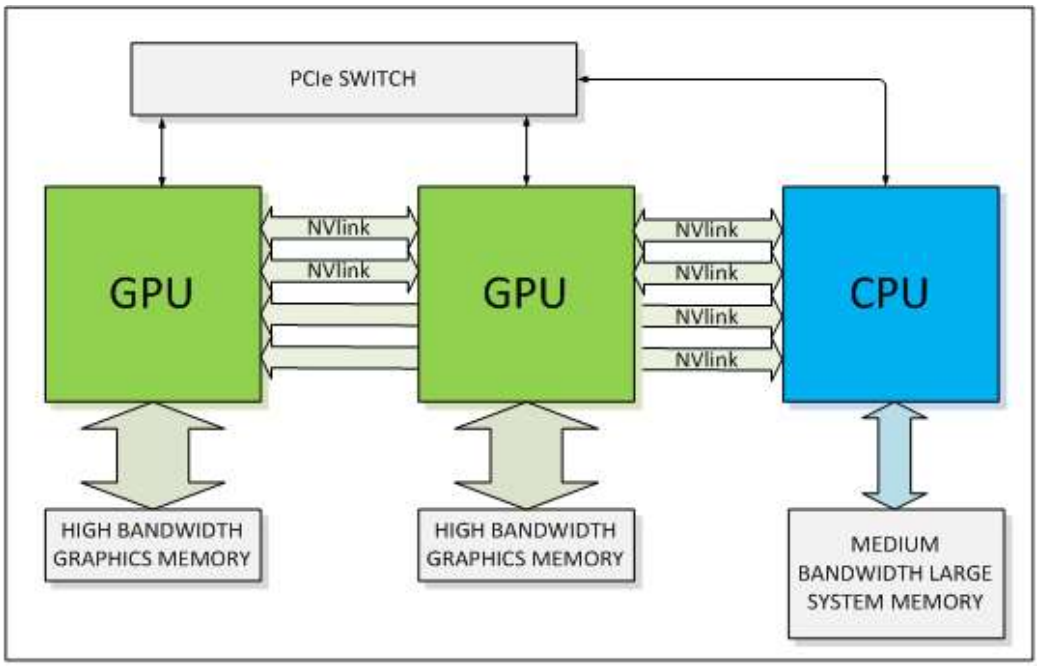


- PCIe
  - v 3.0 speeds ~32 GB-transfers per second per lane
  - v 4.0 is about the double of version 3.0
  - widest supported links = 16 lanes
- Recently: NVLINK (several generations)

With NVLink

Bandwidth of ~80GB/s per link

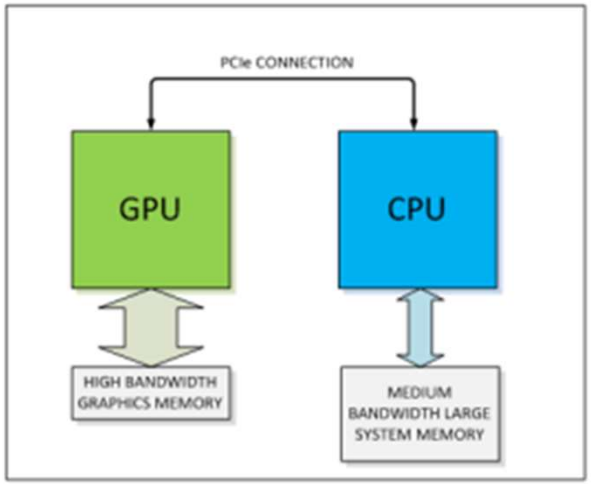


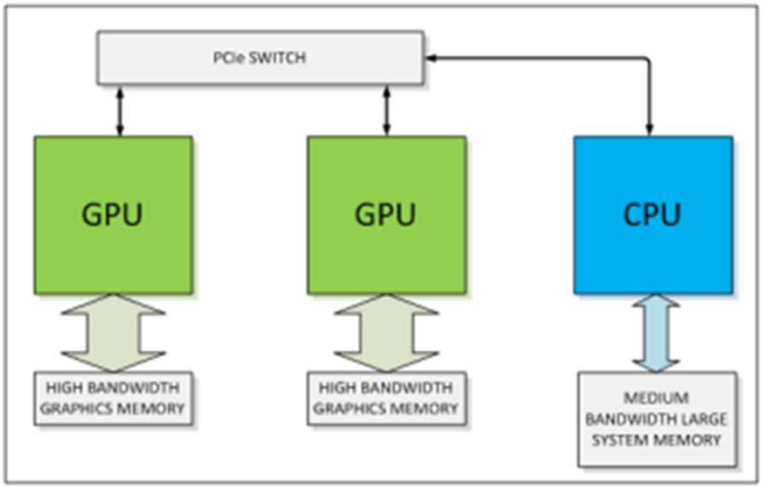


Source: NVIDIA

Speed of PCIe

Version		Speed (x1)
1.0	2.5 GT/s	250 MB/s
2.0	5 GT/s	500 MB/s
3.0	8 GT/s	984.6 MB/s
4.0	16 GT/s	1969 MB/s
5.0	32 GT/s	3938 MB/s
6.0 (expected)	64GT/s	7877 MB/s



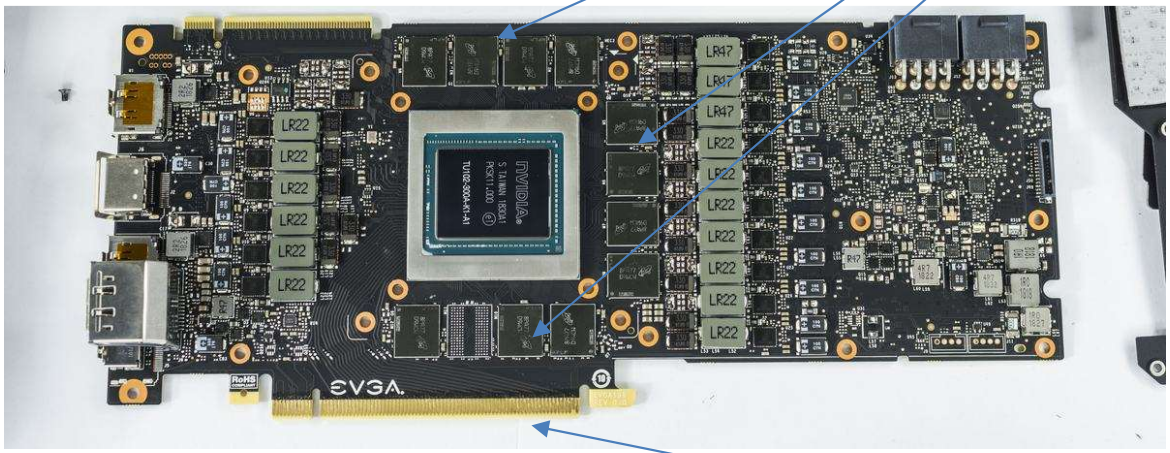


NVLINK

- From NVIDIA
- Starting from PASCAL chips
- Point-to-point connection
- GPU-to-GPU and CPU-GPU connections
- Speeds:

NVIDIA RTX 2080 Ti XC Ultra

Memory Modules

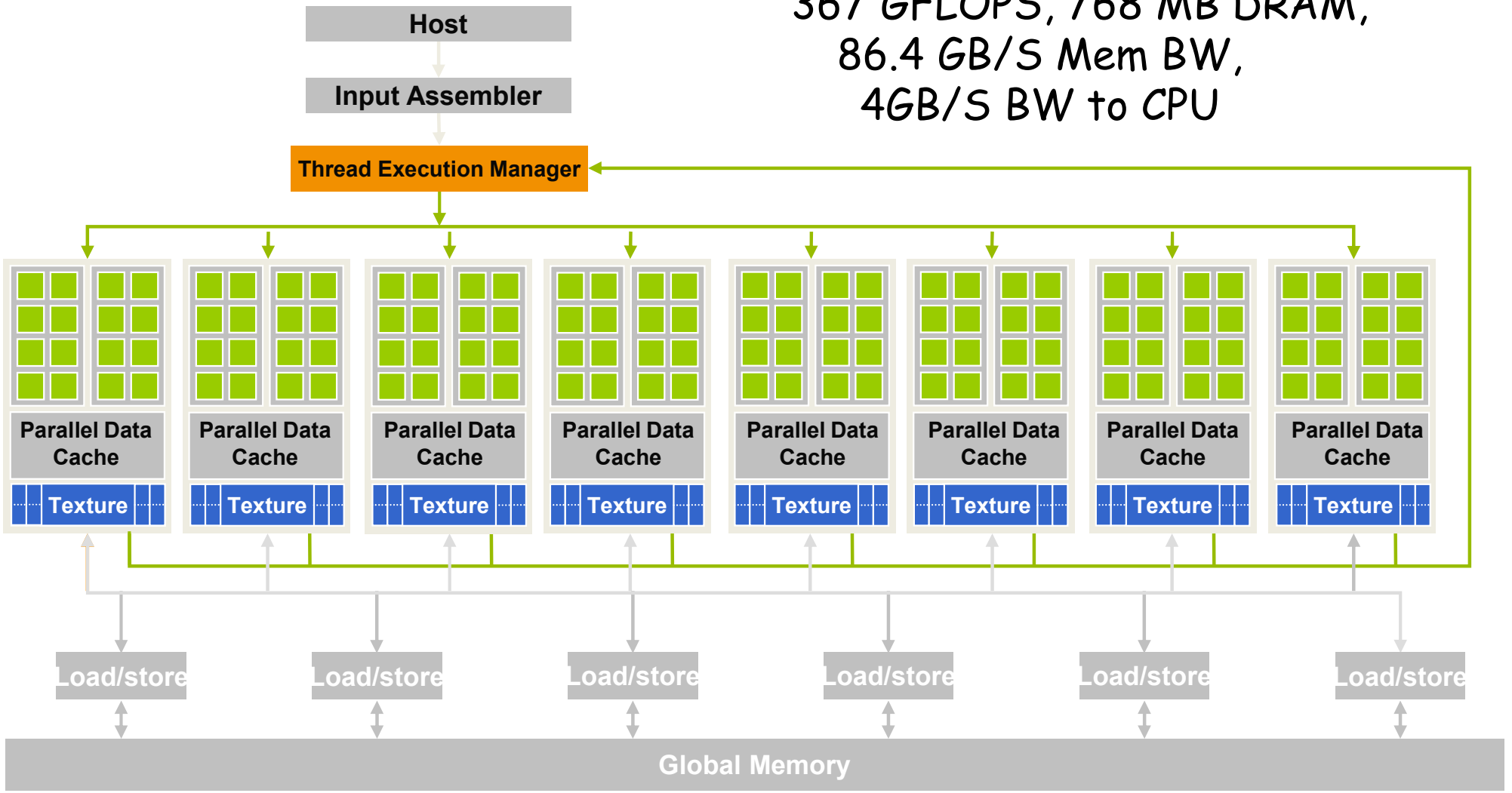


Connection to motherboard

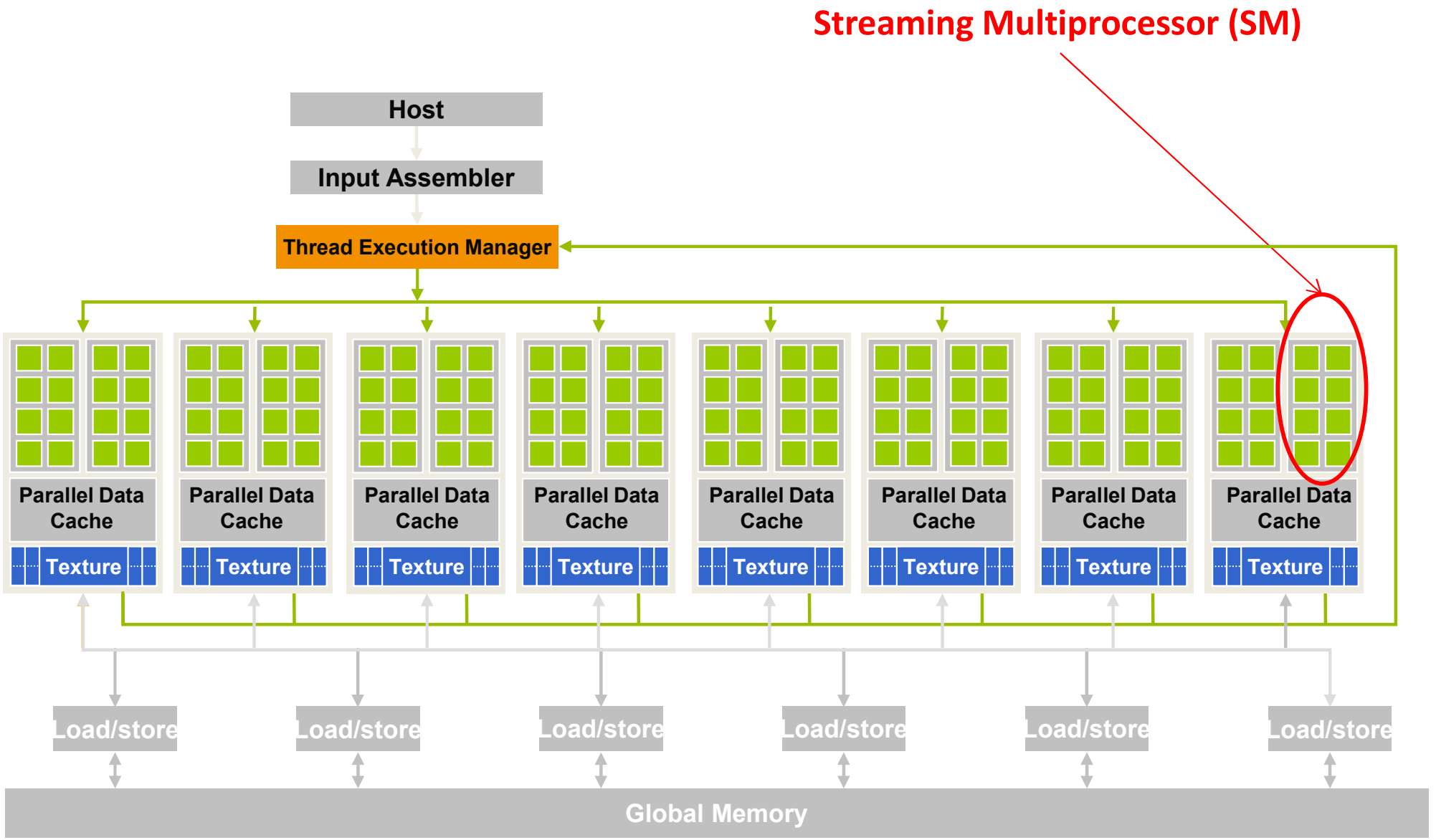
Source: https://xdevs.com/guide/evga_2080tixc/

A Glimpse at At A GPGPU: GeForce 8800 (2007)

16 highly threaded SM's, >128 FPU's,
367 GFLOPS, 768 MB DRAM,
86.4 GB/S Mem BW,
4GB/S BW to CPU



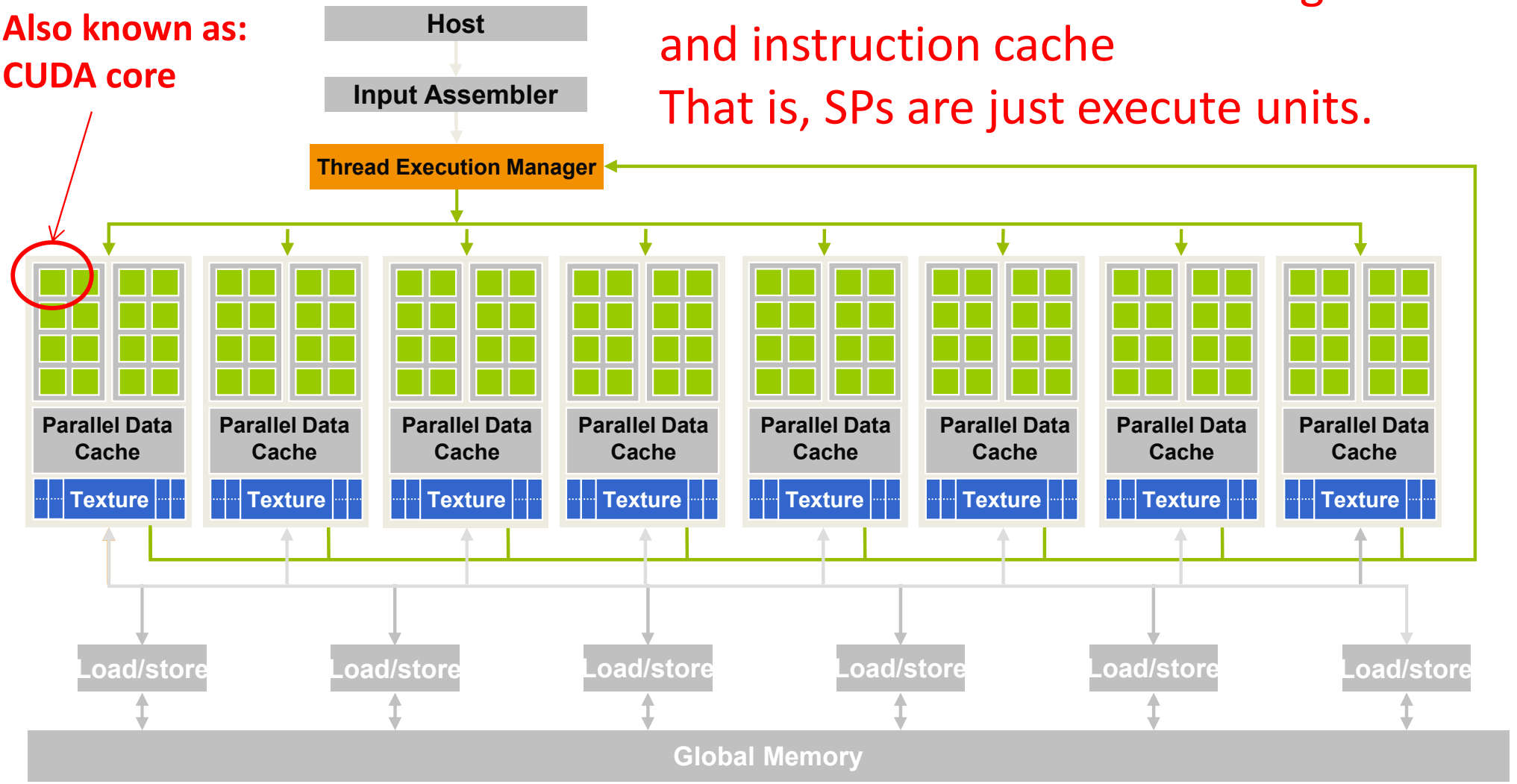
A Glimpse at A Modern GPU

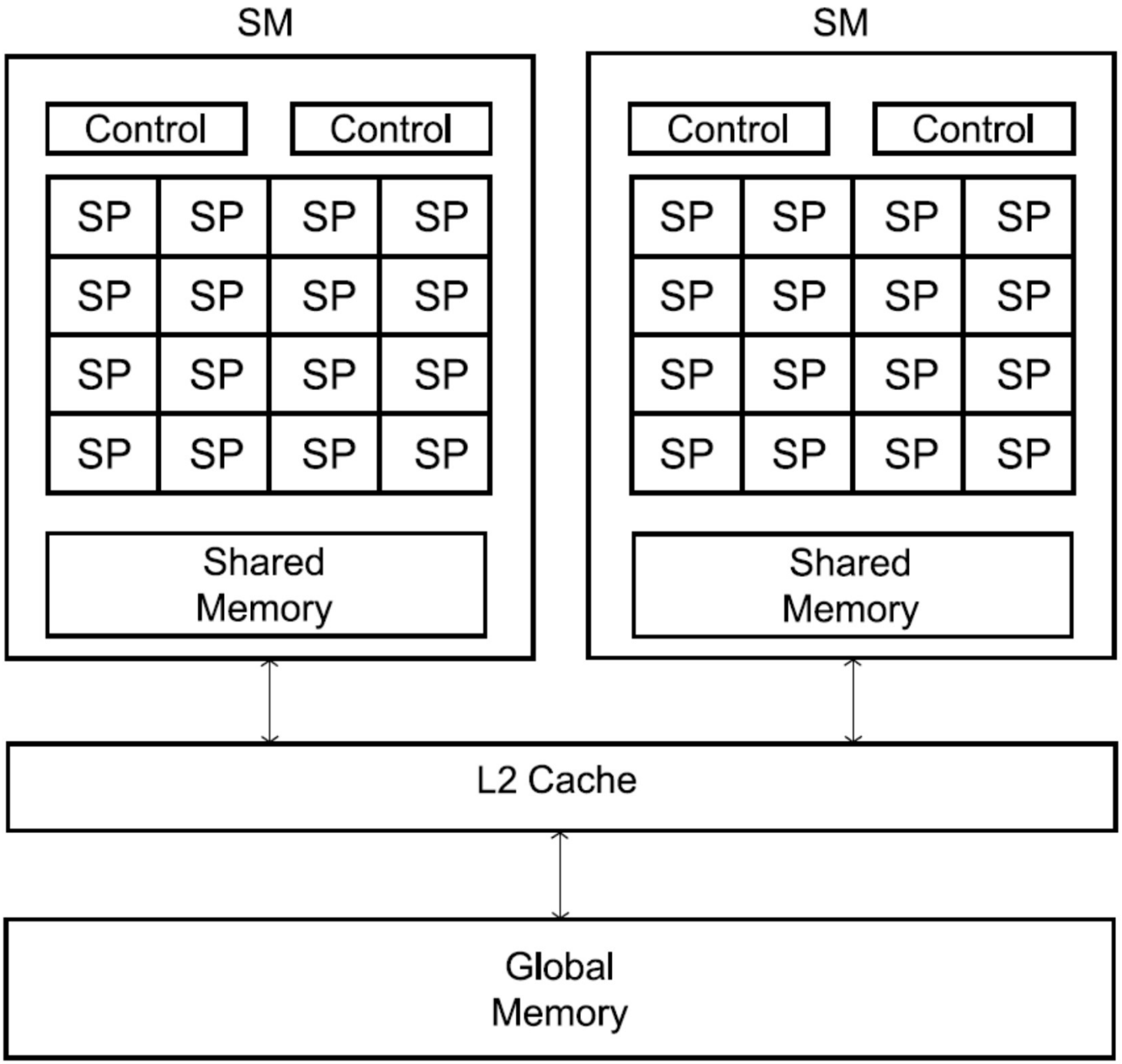


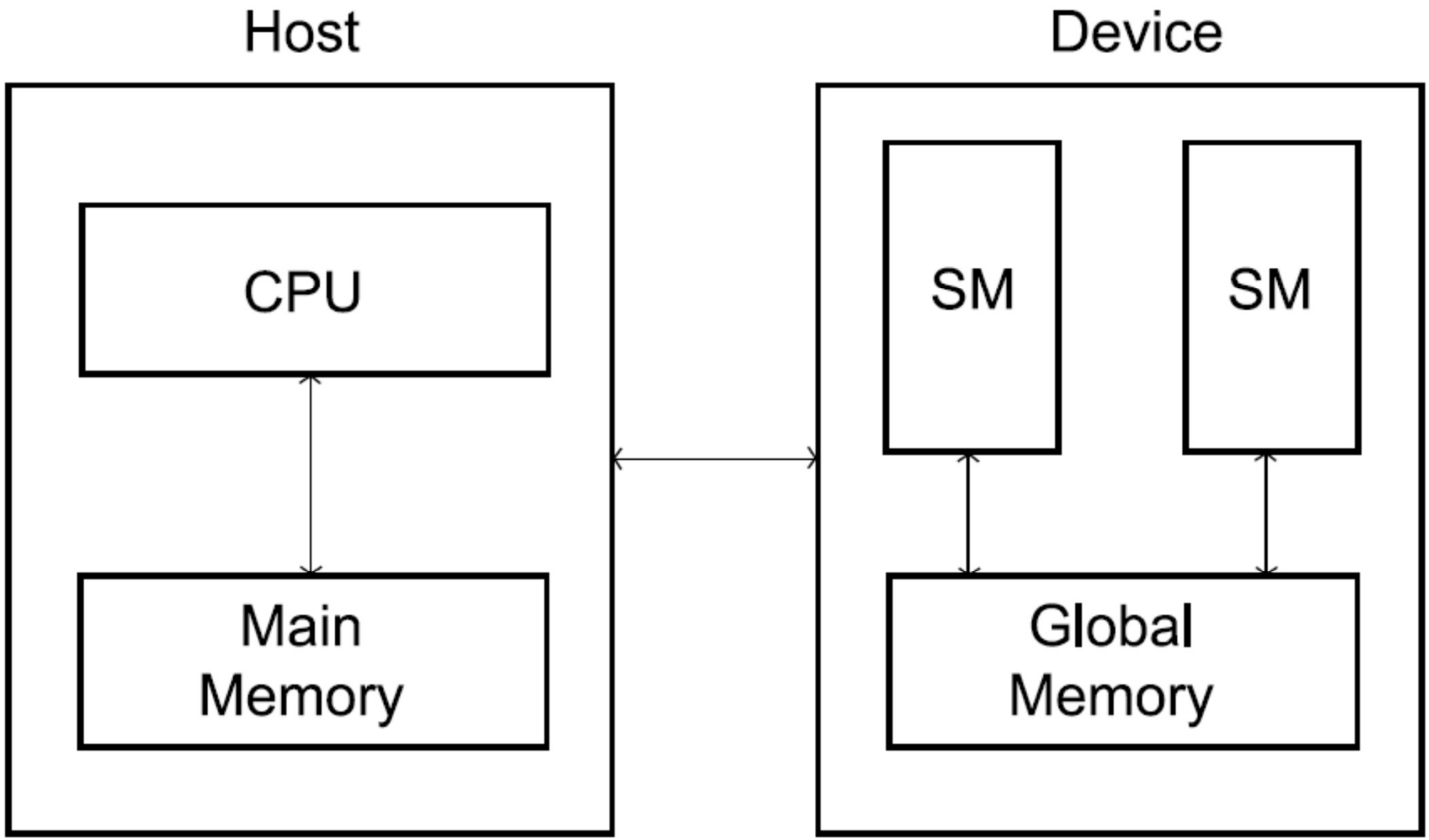
A Glimpse at A Modern GPU

Streaming Processor (SP)
Also known as: CUDA core

SPs within SM share control logic and instruction cache
That is, SPs are just execute units.




SM
SM
Control
Control
Control
Control
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
SP
Shared Memory
Shared Memory
L2 Cache
Global Memory

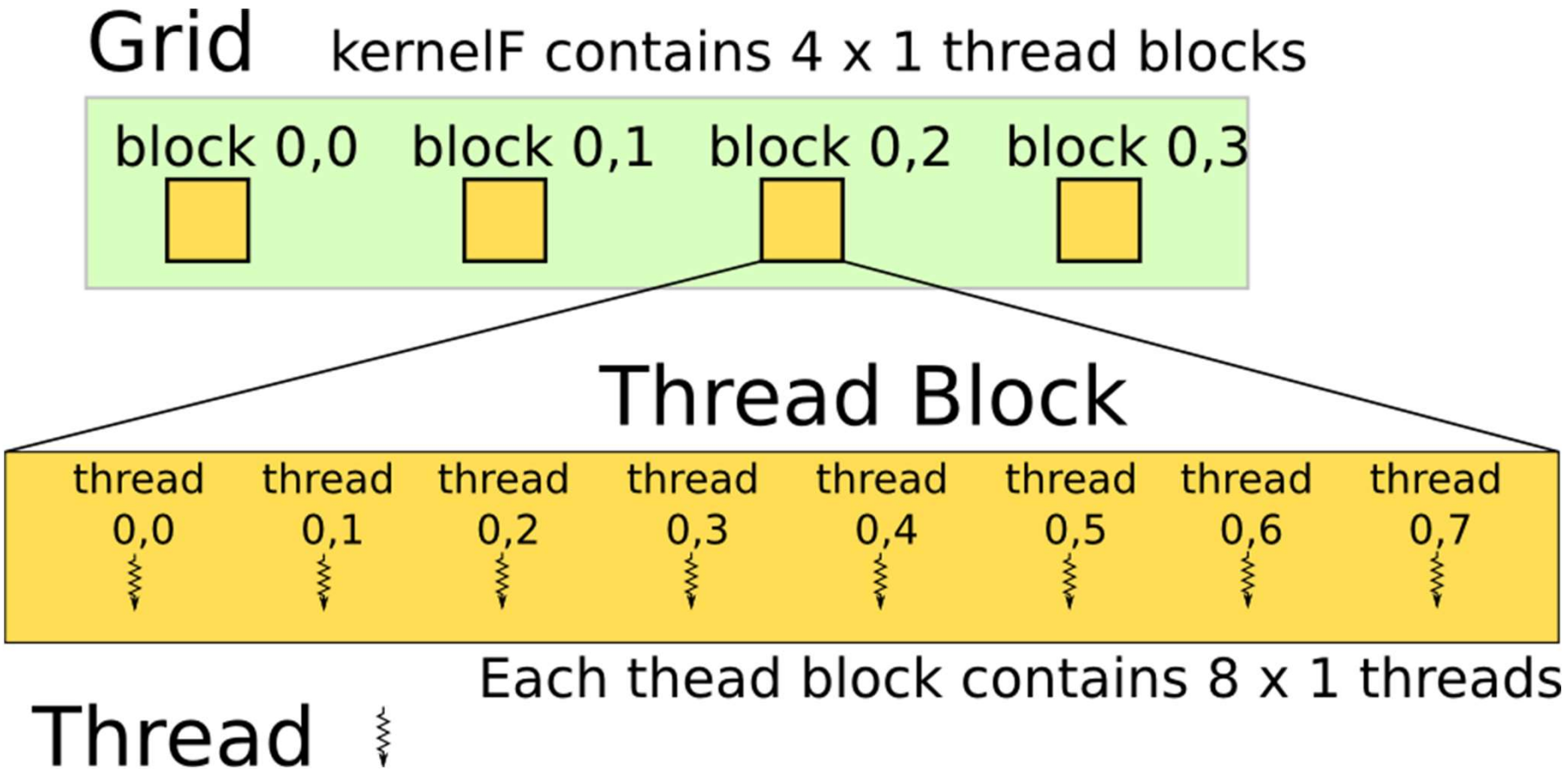

Host
Device
CPU
SM
SM
Main
Memory
Global
Memory

Scalar vs GPU Threaded

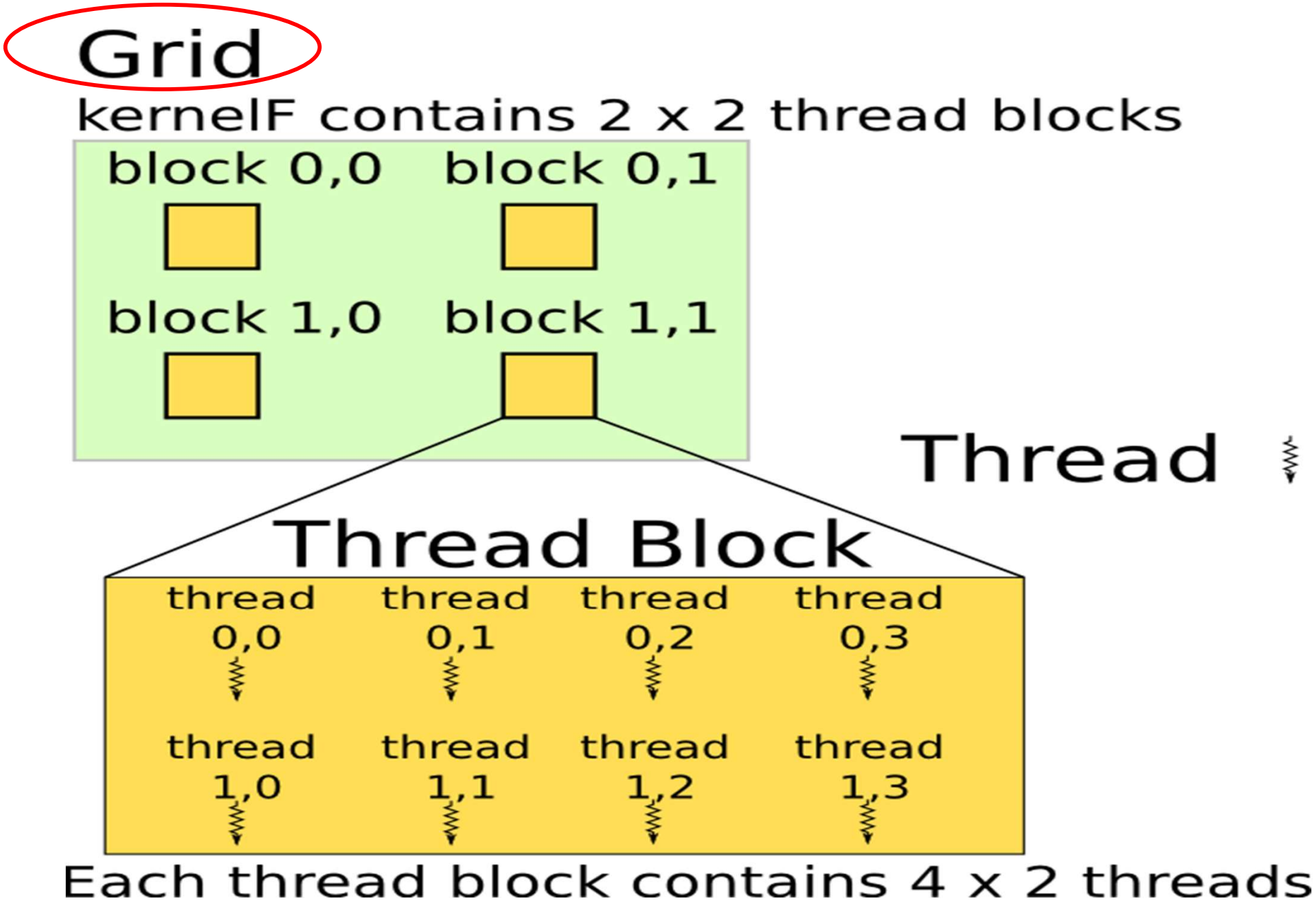
Scalar program (i.e. sequential)

```
float A[4][8];

for(int i=0;i<4;i++){

    for(int j=0;j<8;j++){
        A[i][j]++;
    }
}
```

Multithreaded: (4x1)blocks – (8x1) threads



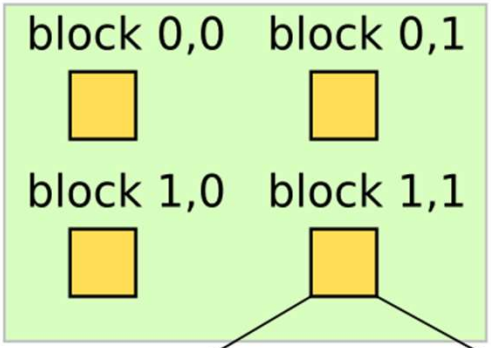
Multithreaded: (2x2)blocks – (4x2) threads



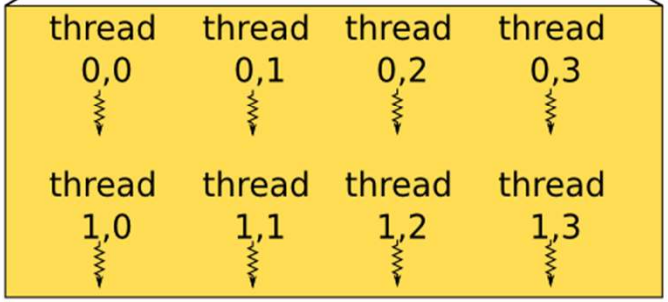
Scheduling Thread Blocks on SM

Grid

kernelF contains 2 x 2 thread blocks



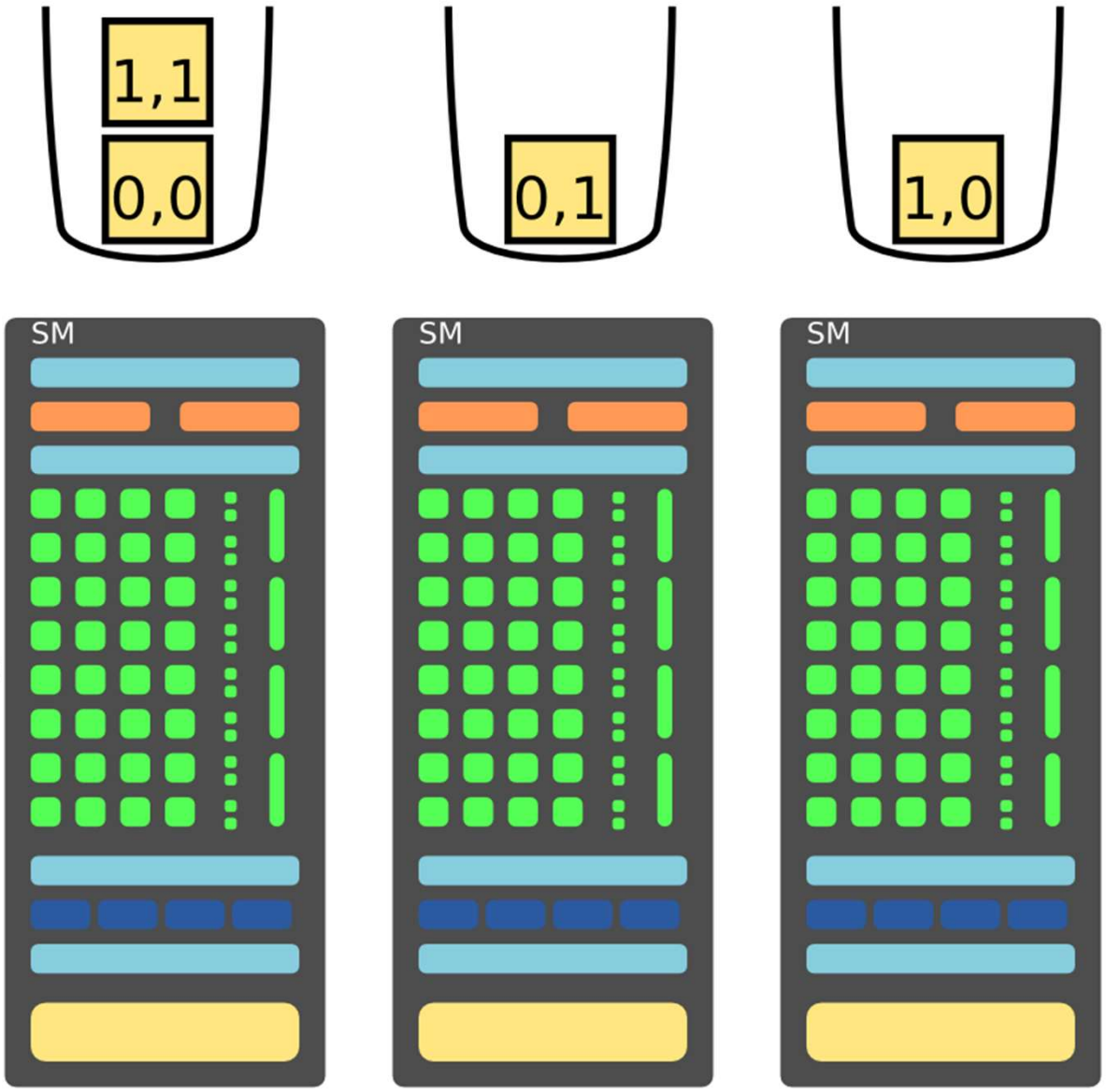
Thread Block



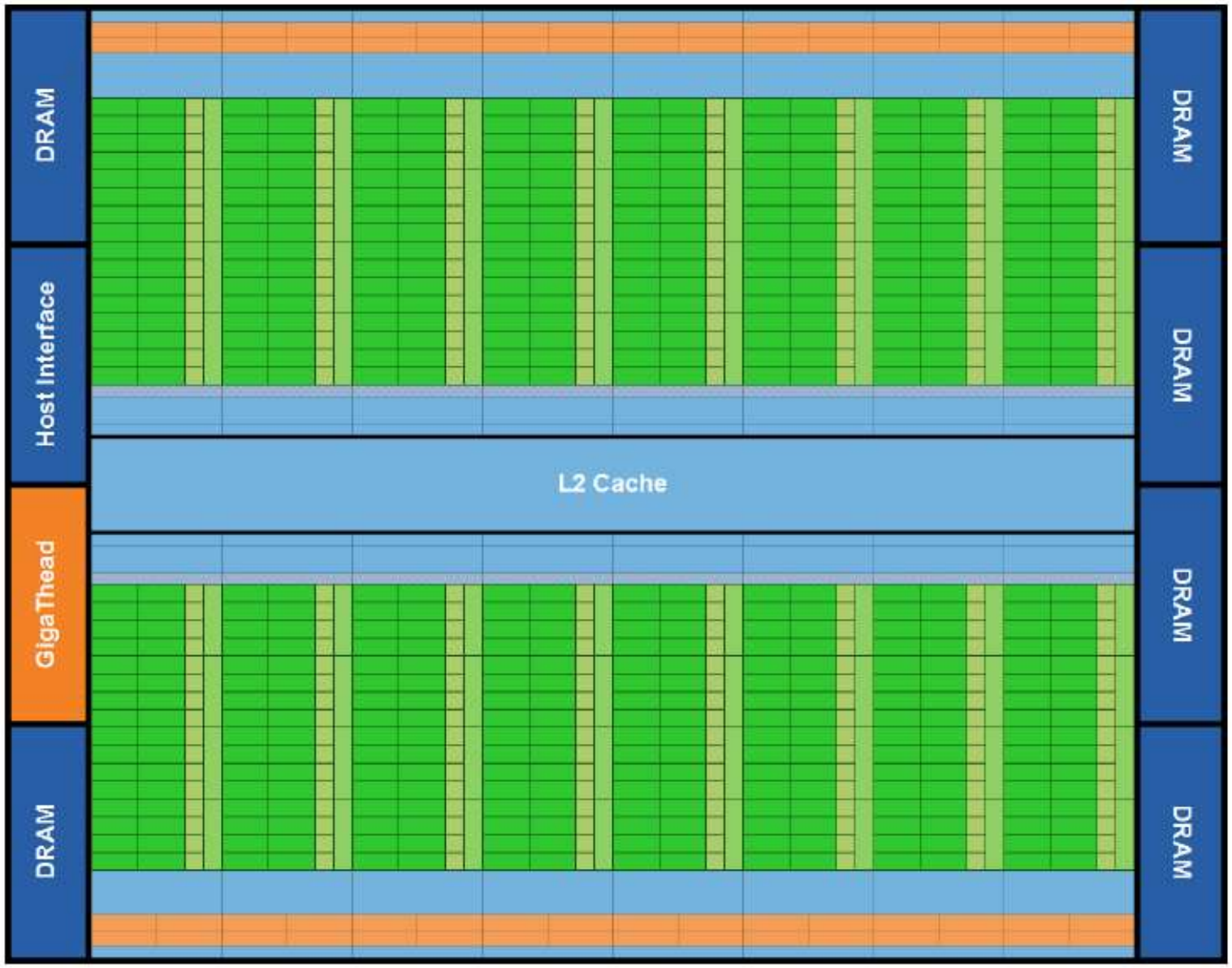
Each thread block contains 4 x 2 threads

Example:

Scheduling 4 thread blocks on 3 SMs.



Another NVIDIA GPU: FERMI



32 cores/SM

~3B Transistors

Another NVIDIA GPU: Kepler

~7.1B transistors
192 cores per SMX

Nvidia Chip GK110 Based on Kepler Architecture

- 7.1 billion transistors
- More then 1 TFlop of double precision throughput
  - 3x performance per watt of Fermi
- New capabilities:
  - Dynamic parallelism
  - Hyper-Q (several cores using the same GPU)
  - Nvidia GPUDirect

Another NVIDIA GPU: Maxwell

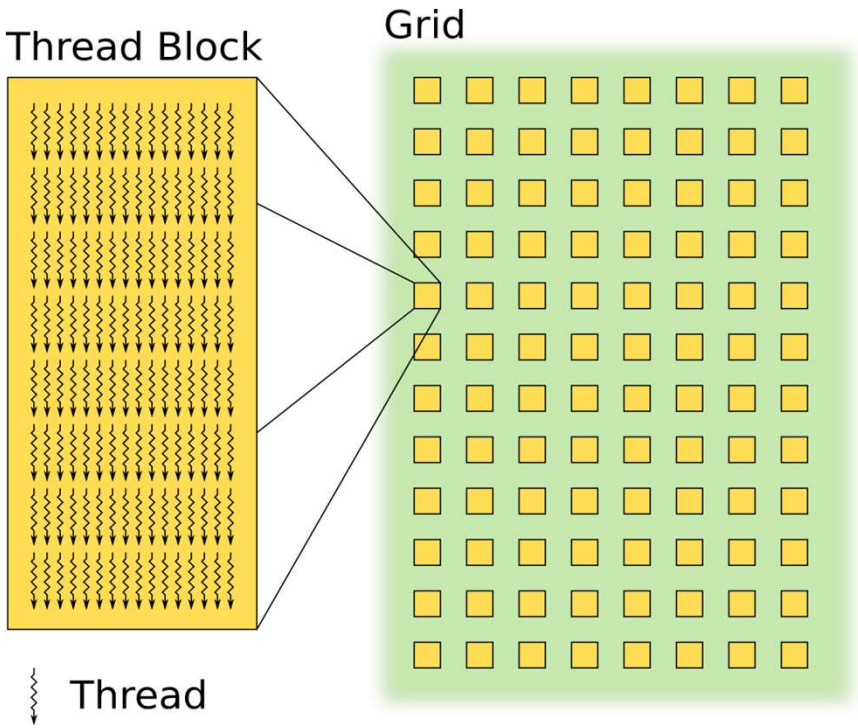
~8B transistors
128 cores per SMM
(Nvidia claims a **128** CUDA core SMM has **90**% of the performance of a **192** CUDA core SMX.)

NVIDIA Architecture Code Names

Tesla
Fermi
Kepler
Maxwell
Pascal
Volta
Turing
Ampere
Hopper
Lovelace

Quick Glimpse At Programming Models

Application → **Kernels** → **Threads** → **Blocks**



↓

Grid

Quick Glimpse At Programming Models

- Application can include multiple kernels
- Threads of the same block run on the same SM
  - So threads in SM can operate and share memory
  - Block in an SM is divided into warps of 32 threads each
  - A warp is the fundamental unit of dispatch in an SM
- Blocks in a grid can coordinate using global shared memory
- Each grid executes a kernel

Scheduling In NVIDIA GPUs

- Modern GPUs can simultaneously execute multiple kernels of the same application
- Two warps from different blocks (or even different kernels) can be issued and executed simultaneously
- More advanced GPUs can do more than that but we will concentrate on the above only here.

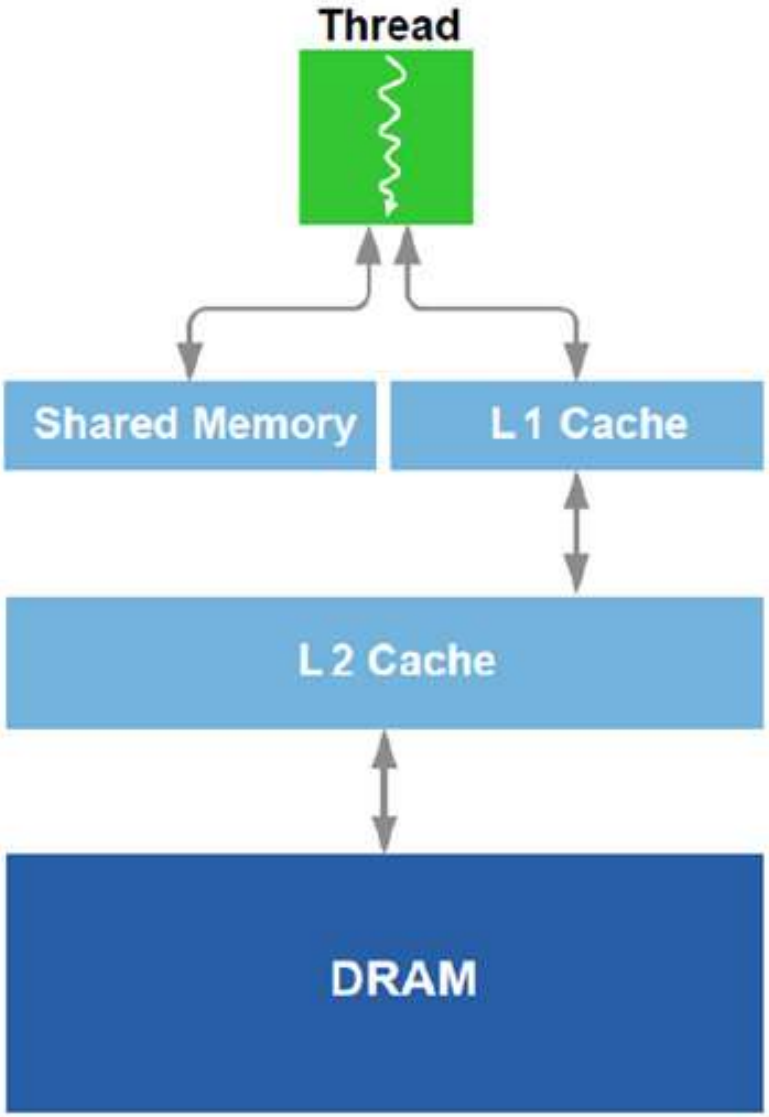
Scheduling In NVIDIA GPUs

- Two-level, distributed thread scheduler
  - At the chip level: a global work distribution engine schedules thread blocks to various SMs
  - At the SM level, each warp scheduler distributes warps of 32 threads to its execution units.

The Memory Hierarchy

- Local memory in each SM
  - first-level (L1) cache
  - shared memory.
- GPUs are also equipped with an L2 cache, shared among all SMs.
- The last level is the global memory.

Fermi Memory Hierarchy



GPUs Today

- Are more and more general purpose and not only for graphics
- Discrete
  - separate chip on-board like all Nvidia GPUs and AMD GPUs
- Integrated
  - With the CPU on the same chip like the GPU in Intel Sandy Bridge and Ivy Bridge

Memory Bus

- Memory bus
  - Path between GPU itself and the video card memory
  - Bus width and speed of memory → bandwidth (GB/s) → more is better
  - Example:
    - GTX 680: 6GHz memory and 256-bit interface → 192.2 GB/s
    - GTX Titan: 6GHz memory and 384-bit interface → 288.4 GB/s
  - Since most modern GPUs use 6GHz memory, the bus width is the one that makes the difference.

Conclusions

- The main keywords:
  - data parallelism
  - kernel, grid, block, and thread
  - warp
- Some applications are better run on CPU while others on GPU
- Main limitations
  - The parallelizable portion of the code
  - The communication overhead between CPU and GPU
  - Memory bandwidth saturation