VANDERBILT UNIVERSITY

ACCRE
Advanced Computing Center for Research & Education

Follow us on Twitter for important news and updates:

@ACCREVandy

GPU Cluster Computing

Advanced Computing Center for Research and Education

What is GPU Computing?

- Graphics processing units (GPUs) are used for rendering graphics on a display (e.g. laptop monitor).
- The gaming and high-definition graphics industries drive the development of fast graphics processing.
- Recently, programming GPUs has been made generic enough to work with scientific computing problems.
- Due to their massively parallel architecture, GPUs enable the completion of computationally intensive tasks much faster than conventional CPUs.
- Hybrid CPU-GPU system:
  - GPU: Computationally-intensive (massively parallel) part.
  - CPU: sequential part.

ACCRE Advanced Computing Center for Research & Education

GPU Performance: Computing

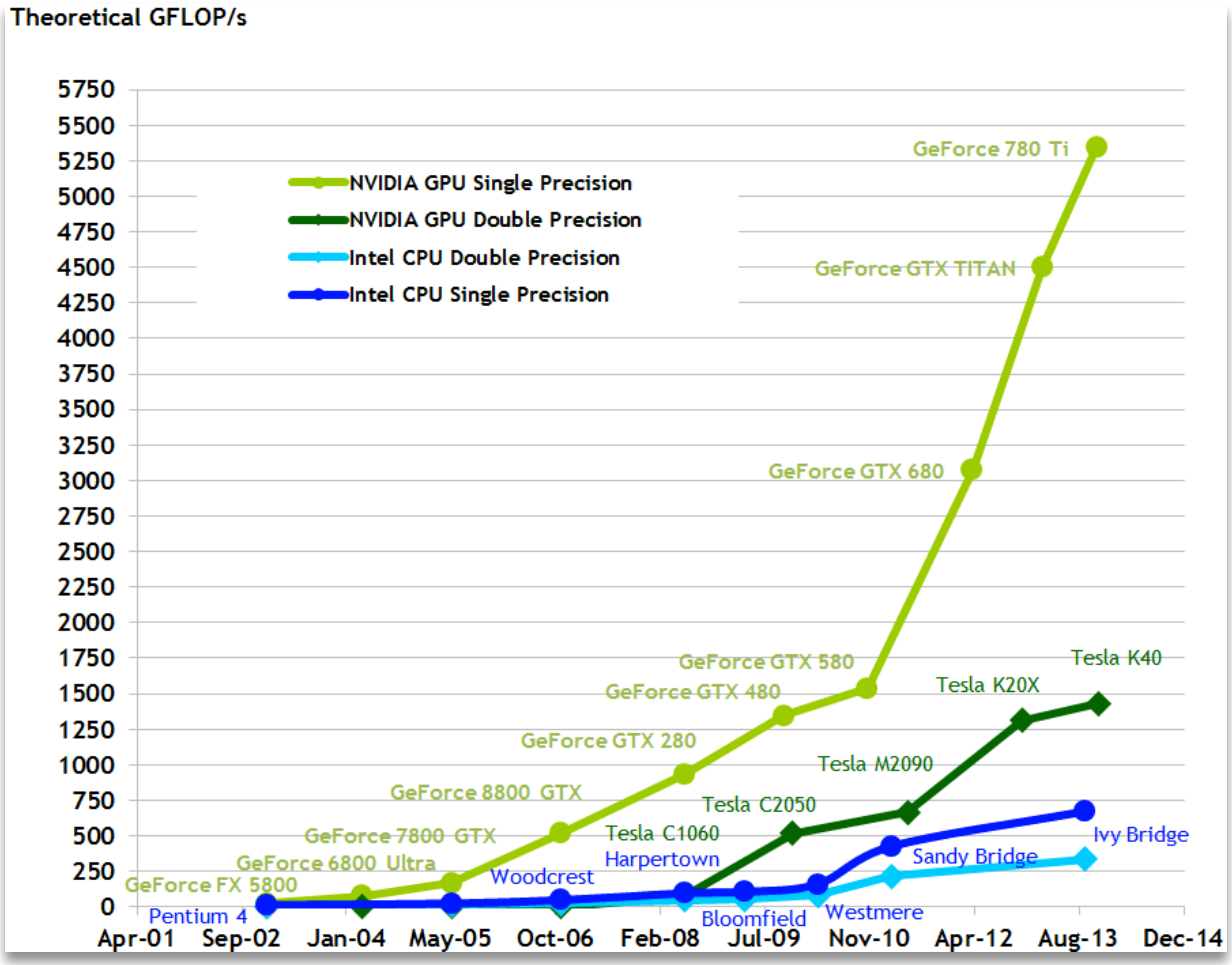


Image taken from CUDA Programming Guide

GPU Performance: Memory Bandwidth

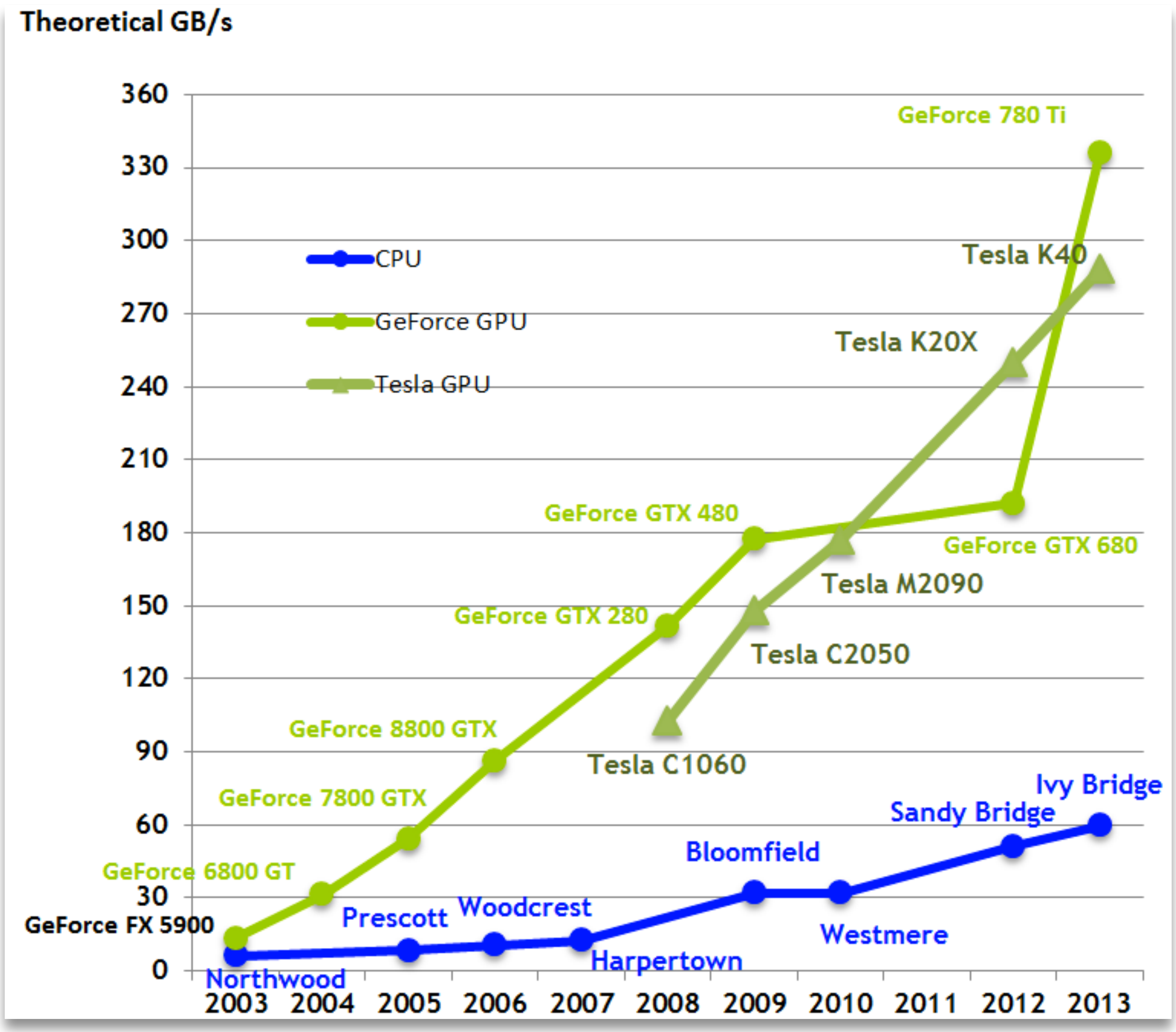
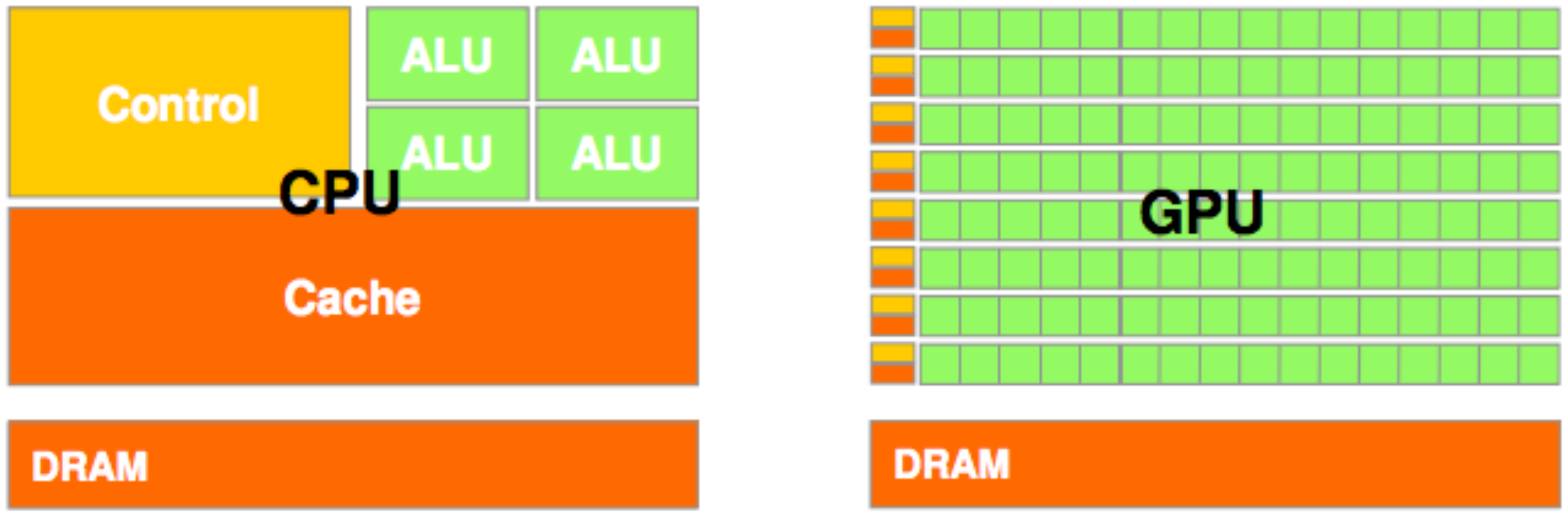


Image taken from CUDA Programming Guide

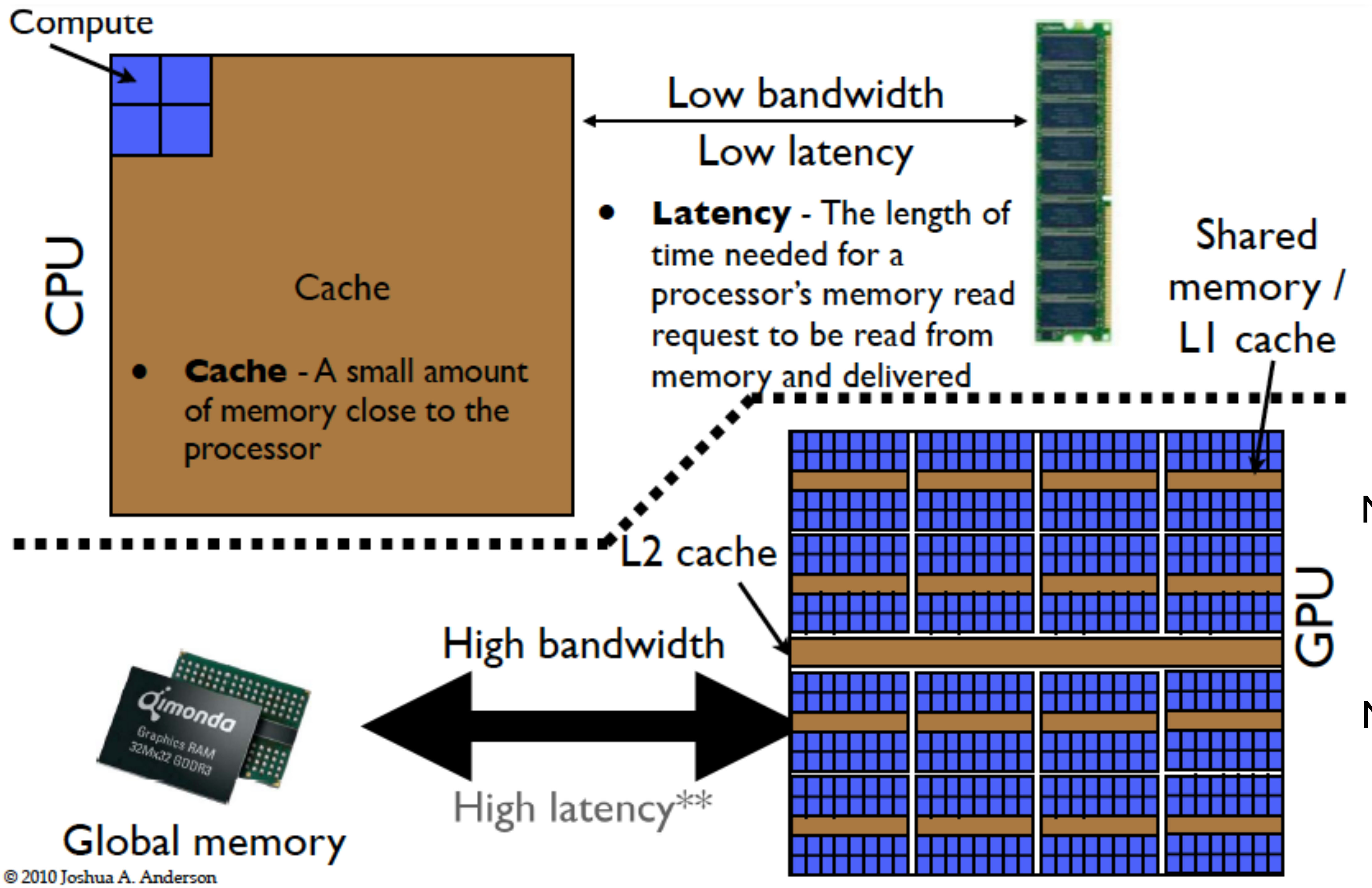
Why is the GPU So Fast?

- The GPU is specialized for compute-intensive, highly data parallel computation (owing to its graphics rendering origin).
  - More transistors can be devoted to data processing rather than data caching and control flow.
  - Where GPUs excel: high arithmetic intensity (the ratio between arithmetic operations and memory operations), matrix and vector operations (remember, your graphics monitor is just a matrix of pixels!).



How are GPUs Different than CPUs?

"CPUs are designed to handle *complexity* well, while GPUs are designed to handle *concurrency* well." - Axel Kohlmeyer



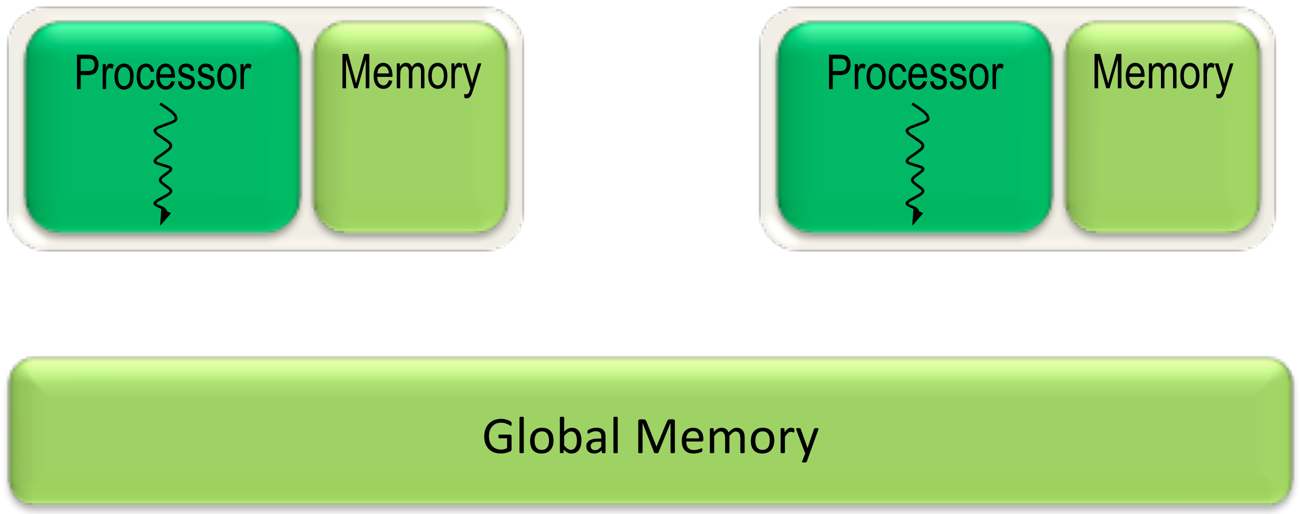
- **Cache** - A small amount of memory close to the processor
- **Latency** - The length of time needed for a processor's memory read request to be read from memory and delivered

Single Instruction, Multiple Thread (SIMT)

$$\begin{tikzcd} \text{GPU} \arrow[d] \\ \text{Multiprocessor} \arrow[d] \\ \text{Core} \arrow[d] \\ \text{Thread} \end{tikzcd}$$

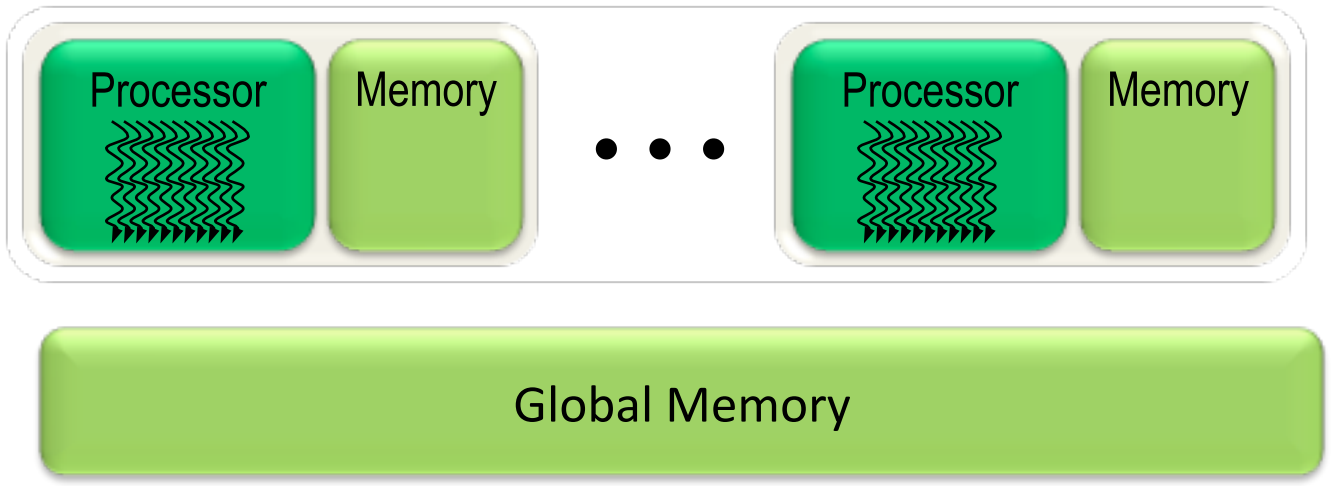
© 2010 Joshua A. Anderson
Recorded for the Virtual School of Computational Science and Engineering

Generic Multicore Chip



- Modern multicore processors
- Handful of CPUs each supporting ~1-2 hardware threads
- On-chip memory near processors (registers , cache)
- Shared global memory space (external DRAM)

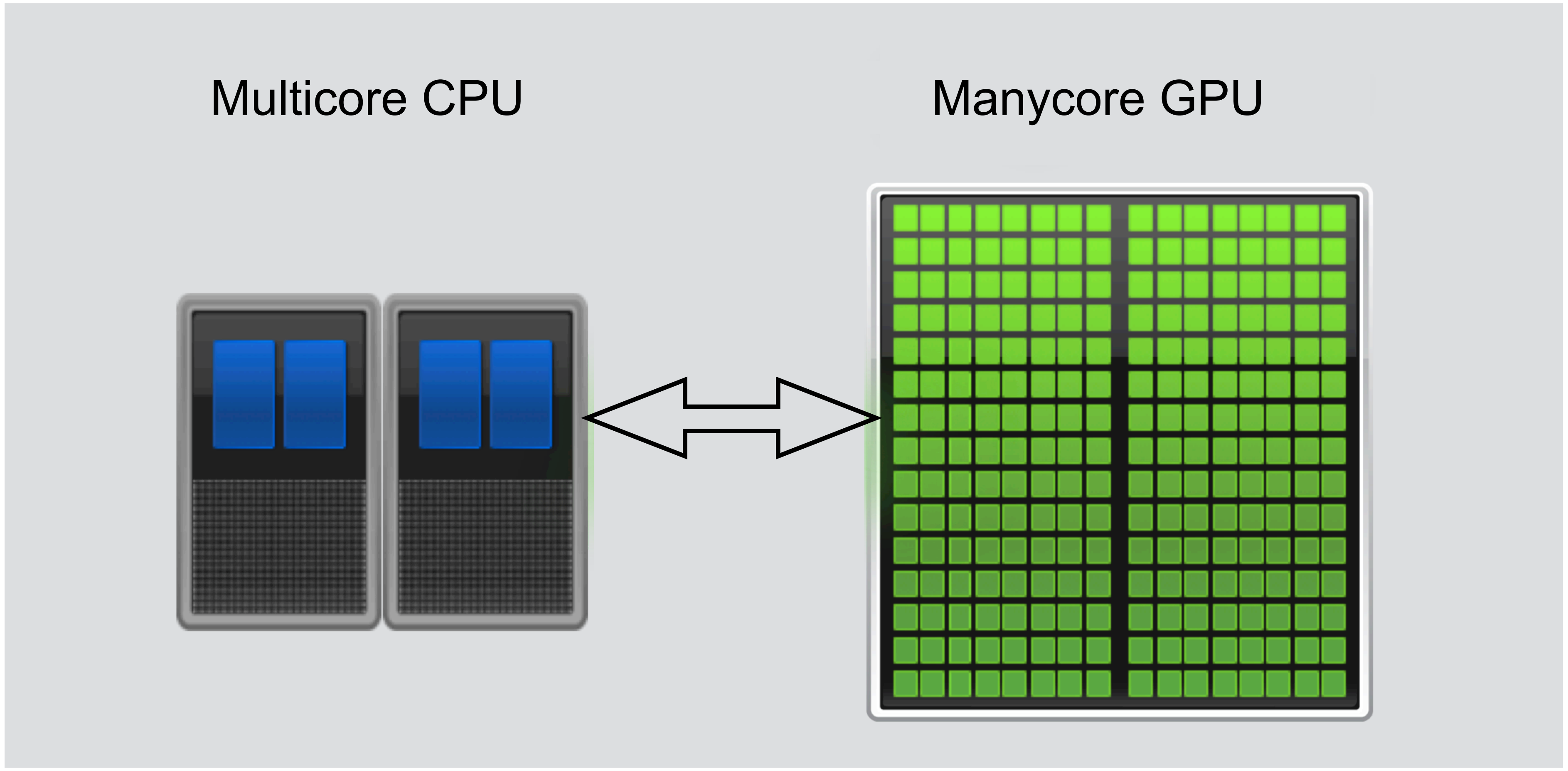
Generic Manycore Chip



- Programmable GPUs
- Several processors each supporting numerous hardware threads
- On-chip memory near processors (registers , cache)
- Shared global memory space (external DRAM)

Heterogeneous

- Typically the GPU and CPU coexist in a heterogeneous setting.
- Less computationally intensive part runs on CPU (coarse-grained parallelism), and more intensive parts run on GPU (fine-grained parallelism).



ACCRE Advanced Computing Center for Research & Education

GPUs

- GPU is typically a computer card, installed into a PCI Express slot.
- Market leaders: NVIDIA, AMD (ATI)
  - Example NVIDIA GPUs

GeForce GTX 480

Tesla 2070

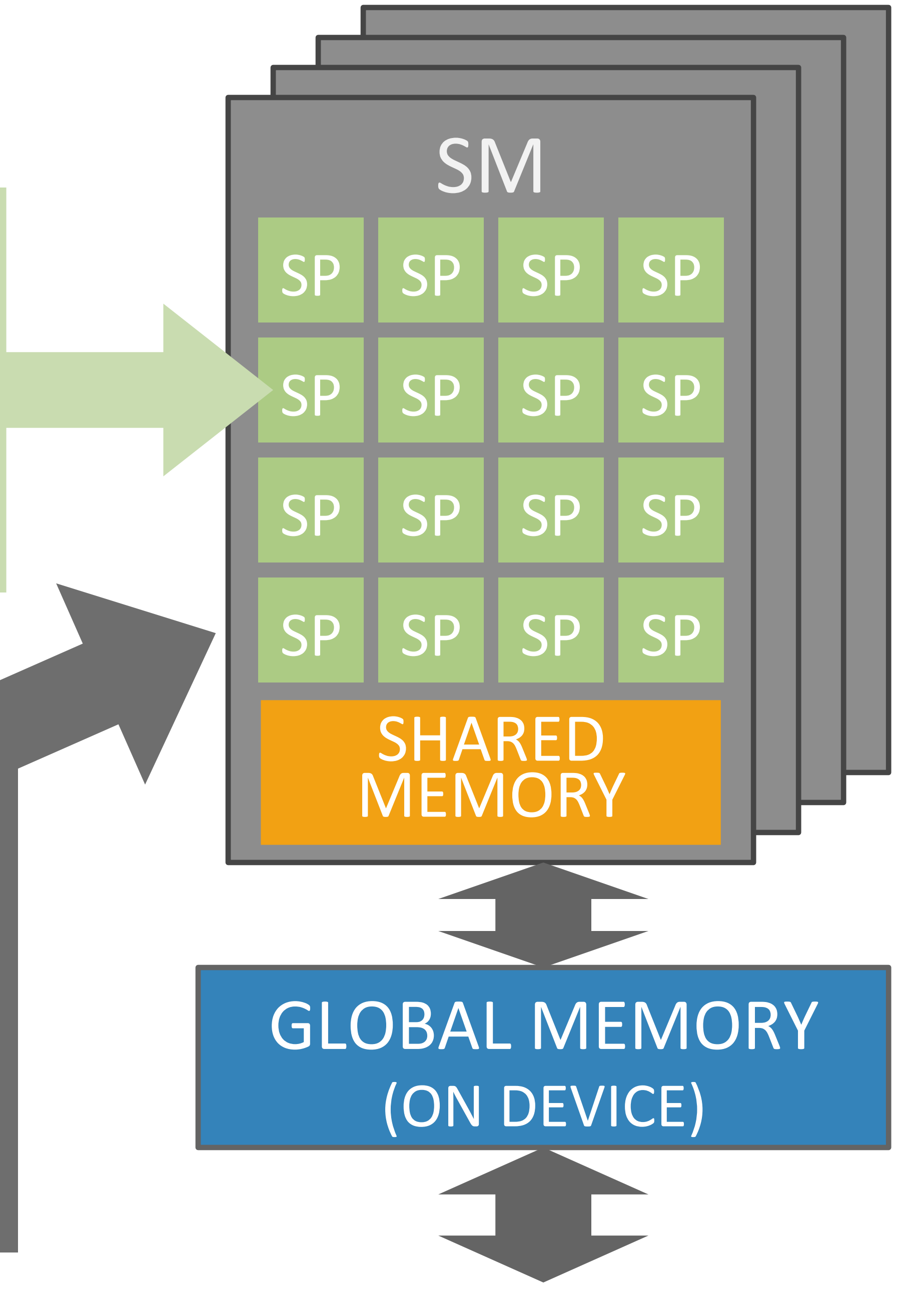
NVIDIA GPU Architecture

SP: scalar processor
'CUDA core'

Executes one thread

SM
streaming multiprocessor
32xSP (or 16, 48 or more)

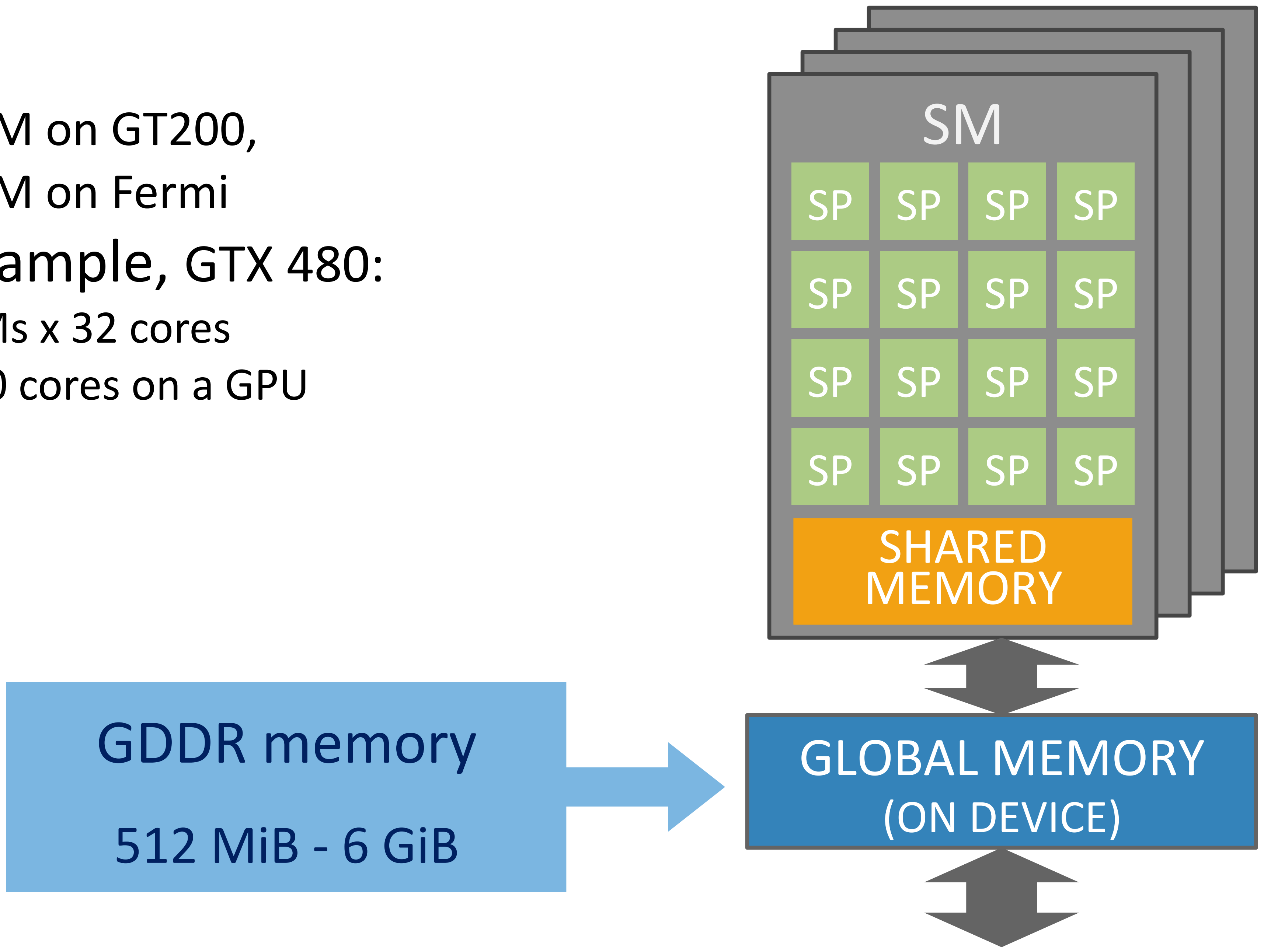
Fast local '**shared memory**'
(shared between SPs)
16 KiB (or 64 KiB)



NVIDIA GPU Architecture

GPU:

- SMs
  - 30 SM on GT200,
  - 15 SM on Fermi
- For example, GTX 480:
  - 15 SMs x 32 cores
    = 480 cores on a GPU



CUDA Programming Model

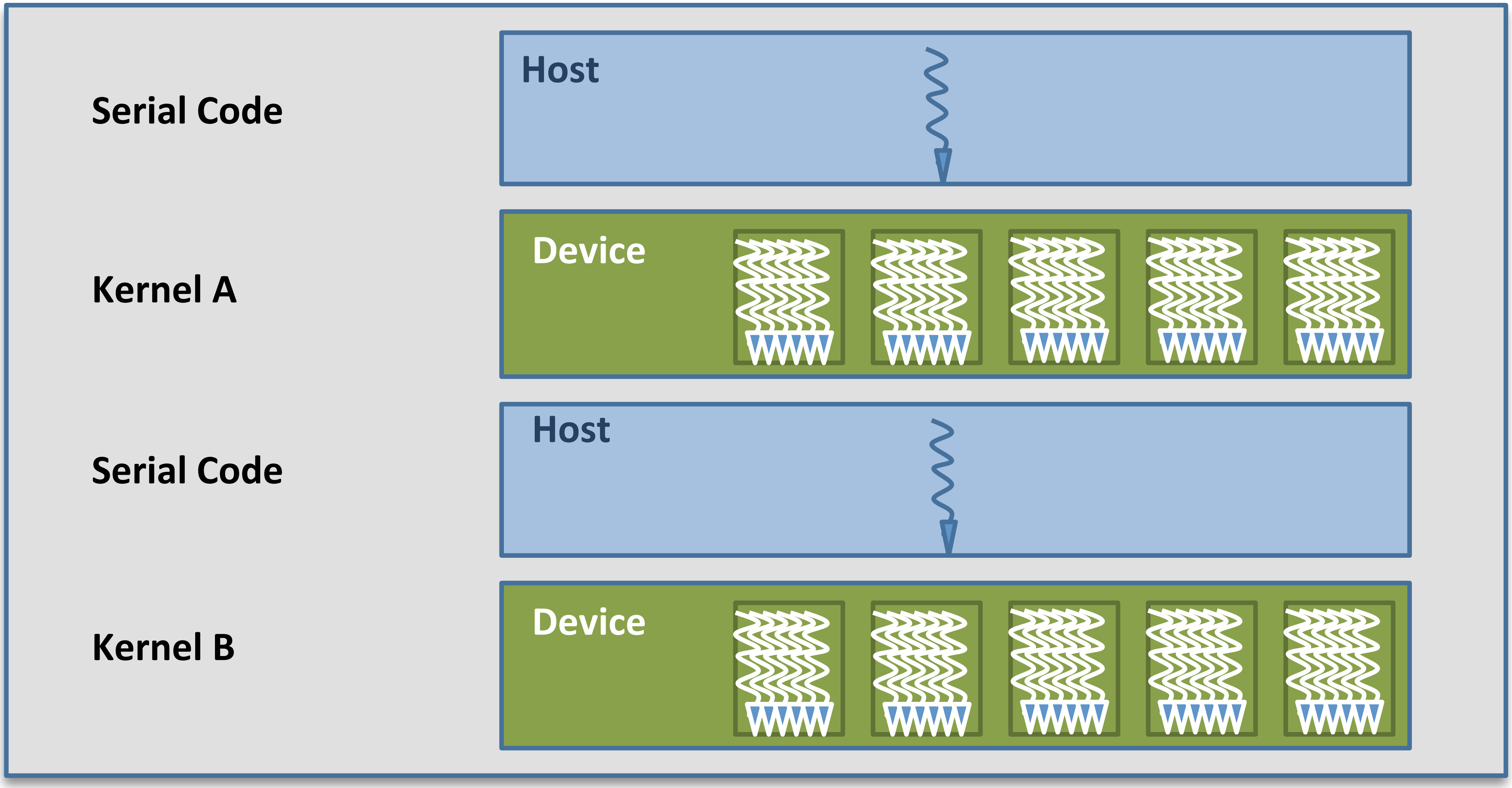
- The **GPU** is viewed as a compute **device** that:
  - Is a co-processor to the **CPU** or **host.**
  - Has its own DRAM (device memory, or global memory in CUDA parlance).
  - Runs many threads in parallel.
- Data-parallel portions of an application run on the device as kernels which are executed in parallel by many threads.
- Differences between GPU and CPU threads:
  - GPU threads are extremely lightweight: very little creation overhead.
  - GPU needs 1000s of threads for full efficiency.
    - Multi-core CPU needs fewer heavy threads.

Software View

A master process running on the CPU performs the following steps:

- Initialize card
- Allocates memory in host and on device
- Copies data from host to device memory
- Launches multiple blocks of execution “kernel” on device
- Copies data from device memory to host
- Repeats 3-5 as needed
- Deallocates all memory and terminates

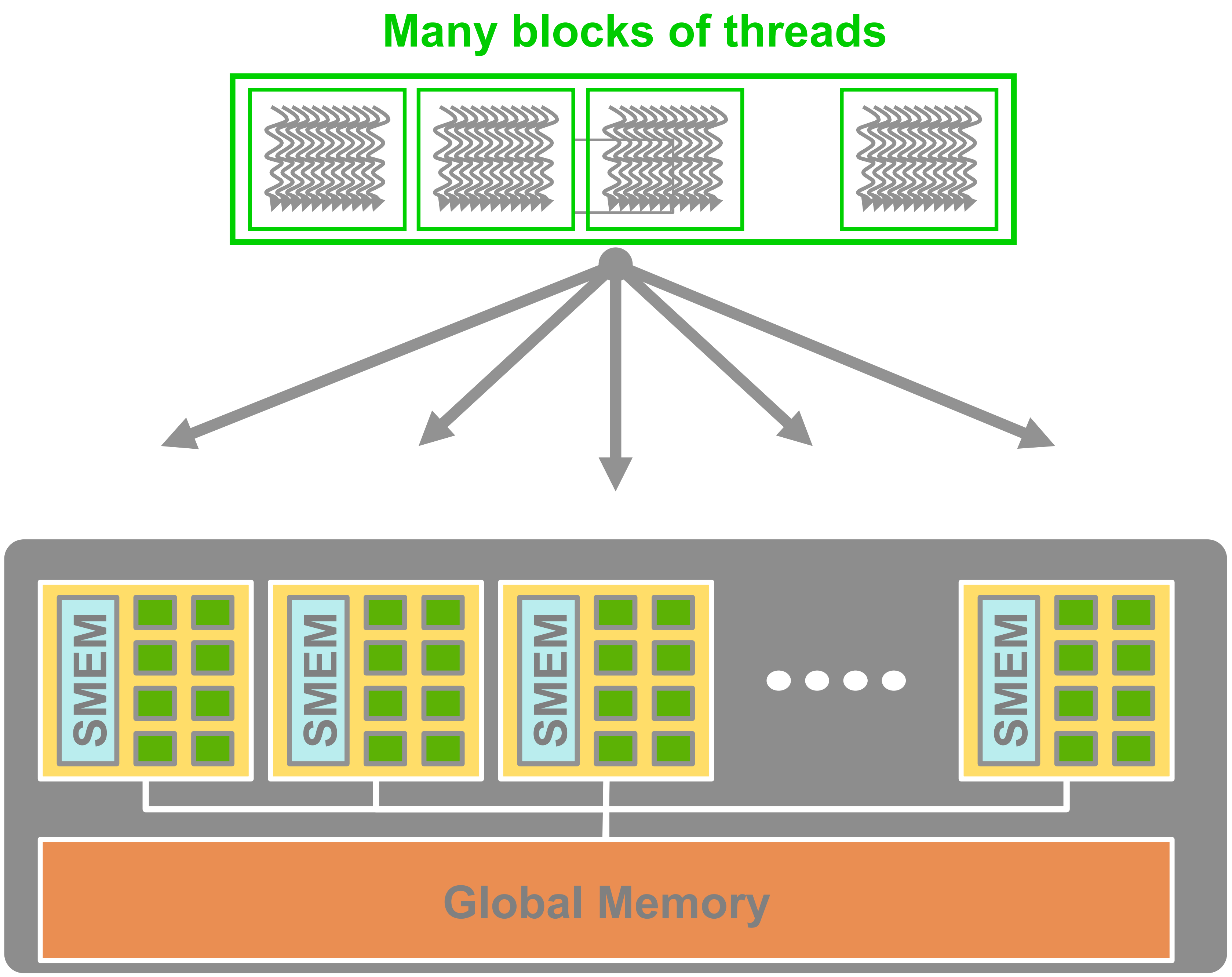
Execution Model



Software view

- Threads launched for a parallel section are partitioned into thread blocks.
- Grid = all blocks for a given launch
- Thread block is a group of threads that can:
  - Synchronize their execution
  - Communicate via shared memory

Whole Grid Runs on GPU



ACCRE Advanced Computing Center for Research & Education

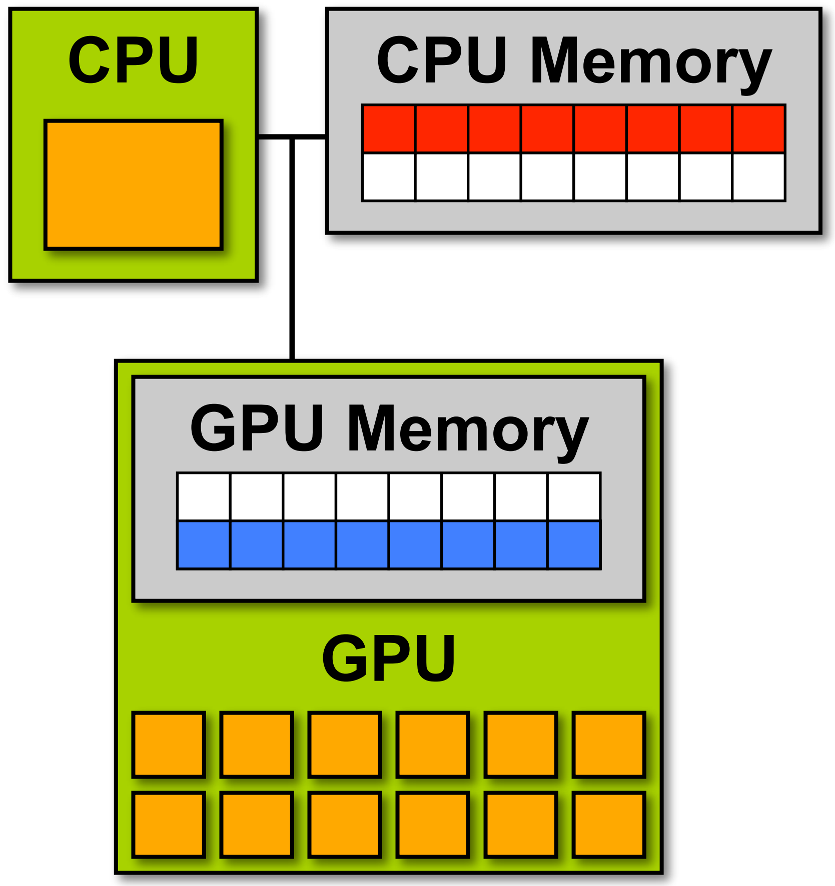
Software View - within GPU

- Each block of the execution kernel executes on a SM (streaming multiprocessor).
- If the number of blocks exceeds the number of SMs, then more than one will run at a time on each SM if there are enough registers and shared memory, and the others will wait in a queue and execute later.
- All threads within one block can access local shared memory but can't see what the other blocks are doing (even if they are on the same SM).
- There are no guarantees on the order in which the blocks execute.

Software View

```
void function(…) {
  Allocate memory on the GPU
  Transfer input data to the GPU
  Launch kernel on the GPU
  Transfer output data to CPU
}

__global__ void kernel(…) {
  Code executed on
   the GPU goes here…
}
```



ACCRE GPU Nodes

- 48 compute nodes with:
  - Two quad-core 2.4 GHz Intel Westmere processors, 48 GB of memory
  - 4 Nvidia GTX 480
  - 10 Gigabit Ethernet
- Nvidia GTX 480
  - 15 Streaming Multiprocessors per GPU
  - 480 Compute cores per GPU
  - Peak performance: 1.35 TFLOPS SP / 168 GFLOPS DP
  - Memory: 1.5 GB
  - Memory Bandwidth: 177.4 GB/sec

Software Stack

- Operating system:
  - CentOS 6.5
- Batch scheduler system:
  - SLURM
- All compilers and tools are available on the GPU gateway (vmps81)
  - type: "*ssh* vunetid@vmps81.accre.vanderbilt.edu" OR "rsh vmps81" if you're already logged into the cluster
  - GCC, Intel compiler
  - Compute nodes share the same base OS and libraries

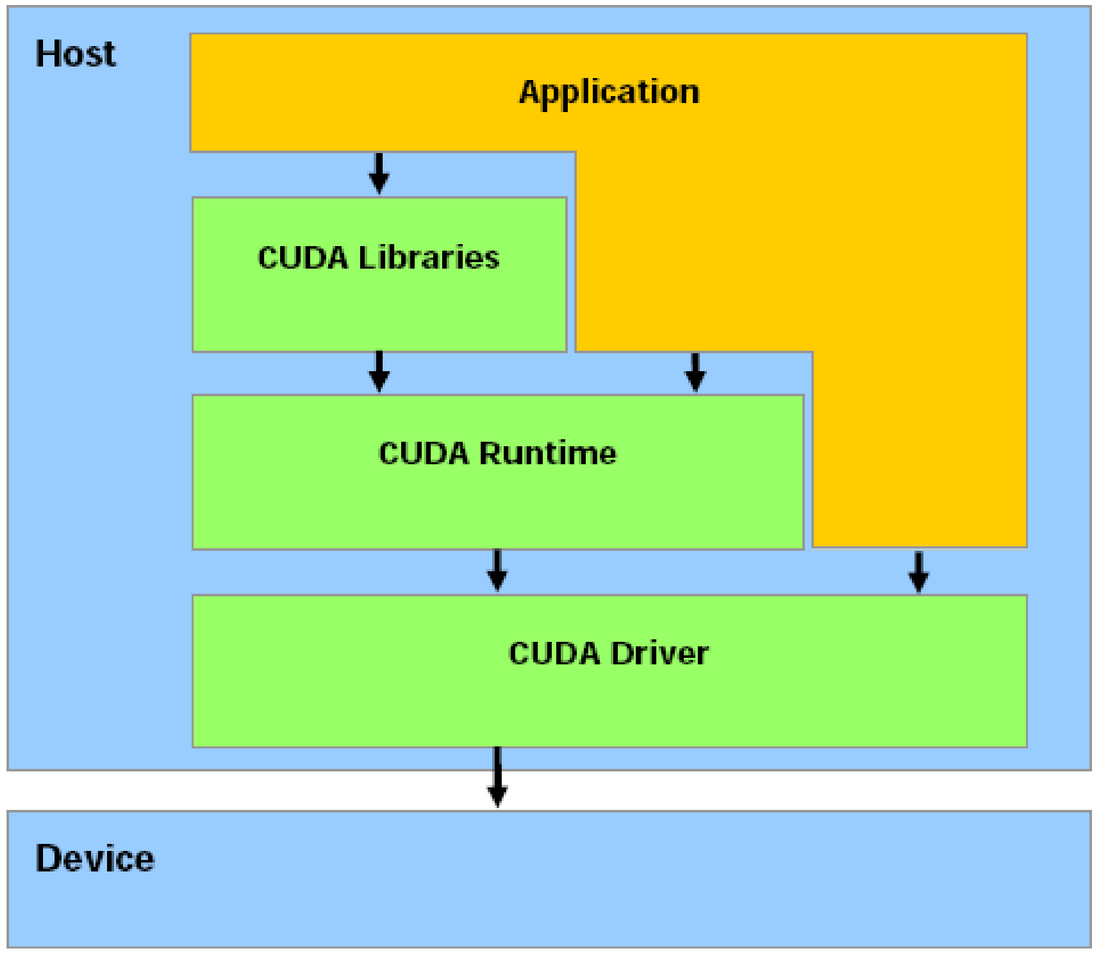
CUDA

- Based on C with some extensions.
- C++ support increasing steadily.
- FORTRAN support provided by PGI compiler.
- Lots of example code and good documentation.
- Large user community on NVIDIA forums.

CUDA Components

- Driver
  - Low-level software that controls the graphics card
- Compute Capability
  - Refers to the GPU architecture, and features supported by hardware
- Toolkit (CUDA version)
  - nvcc CUDA compiler, libraries, profiling/debugging tools
- SDK (Software Development Kit)
  - Lots of demonstration examples
  - Some error-checking utilities
  - Not officially supported by NVIDIA
  - Almost no documentation

CUDA Architecture



CUDA Toolkit

- Various versions of the CUDA toolkit are installed on the cluster. To view a list type pkginfo l grep cuda. To load the appropriate version into your environment type setpkgs -a cuda7.0.
- Source files with CUDA language extensions (.cu) must be compiled with ***nvcc.***
- Actually, **nvcc** is a compile driver.
  - Works by invoking all the necessary tools and compilers like gcc, cl, ...

ACCRE
Advanced Computing Center
for Research & Education

CUDA Toolkit

```c
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

```
$ nvcc hello_world.c
$ a.out
Hello World!
$
```

NVIDIA compiler (nvcc) can be used to compile programs with no **device** code.

ACCRE Advanced Computing Center for Research & Education

CUDA Toolkit

```
__global__ void mykernel(void) {
}

int main(void) {
    mykernel<<<1,1>>>();
    printf("Hello World!\n");
    return 0;
}
```

CUDA C/C++ keyword `__global__` indicates a function (i.e. **kernel**) that:

- Runs on the device
- Called from host (CPU) code

`nvcc` separates source code into host and device components

- Device functions, e.g. mykernel(), processed by NVIDIA compiler
- Host functions, e.g. main(), processed by standard host compile, e.g. gcc, cl.exe

CUDA SDK

- Provides hundreds of code samples to help you get started on the path of writing software with CUDA C/C++.
- Code samples cover a wide range of applications and techniques.
- Installed at `/usr/local/cuda-7.0/samples`.
    - Examples from image processing, finance, fluid dynamics, astronomy, molecular dynamics, and so on.
    - Also useful utilities and examples for querying a device's hardware specifications and linking to CUDA libraries (CuFFT, CuBLAS).

Job Submission

- Do not ssh to a compute node.
- Must use SLURM to submit jobs.
  - Either as batch job or interactively.
  - Provides exclusive access to a node and all 4 GPUs.
  - Must include partition and account information.
- If you need interactive access to a GPU for development or testing:

```
salloc --time=6:00:00 --partition=gpu --account=<mygroup>_gpu
```

- For batch jobs, include these lines:

```
#SBATCH --partition=gpu
#SBATCH --account=<account>_gpu
```

ACCRE
Advanced Computing Center for Research & Education

GPU Jobs

```
#!/bin/bash
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=2:00:00      # 2 hours
#SBATCH --mem=100M
#SBATCH --output=gpu-job.log
#SBATCH --partition=gpu
#SBATCH --account=accre_gpu    # substitute appropriate group here
setpkgs -a hoomd
pwd
date
hoomd simple-script.py
date
```

- Must be in a GPU group and include the --partition and --account options
- Single job has exclusive rights to allocated node (4 GPUs per node)
- See https://github.com/accre/SLURM/tree/master/gpu-job

Getting help from ACCRE

- ACCRE website FAQ and Getting Started pages: www.accre.vanderbilt.edu/?page_id=35
- ACCRE Help Desk:

  www.accre.vanderbilt.edu/?page_id=369
- Accre-forum mailing list
- ACCRE office hours: open a help desk ticket and mention that you would like to work personally with one of us in the office. The person most able to help you with your specific issue will contact you to schedule a date and time for your visit.