INTRODUCTION TO GPU COMPUTING

Evan Schneider

WHERE TO LEARN MORE

Online resources:

Numba tutorial:

https://numba.pydata.org/numba-doc/dev/cuda/index.html

CUDA Programming Guide:

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Hack weeks:

Oak Ridge GPU Hackathons:
https://www.olcf.ornl.gov/for-users/training/gpu-hackathons/

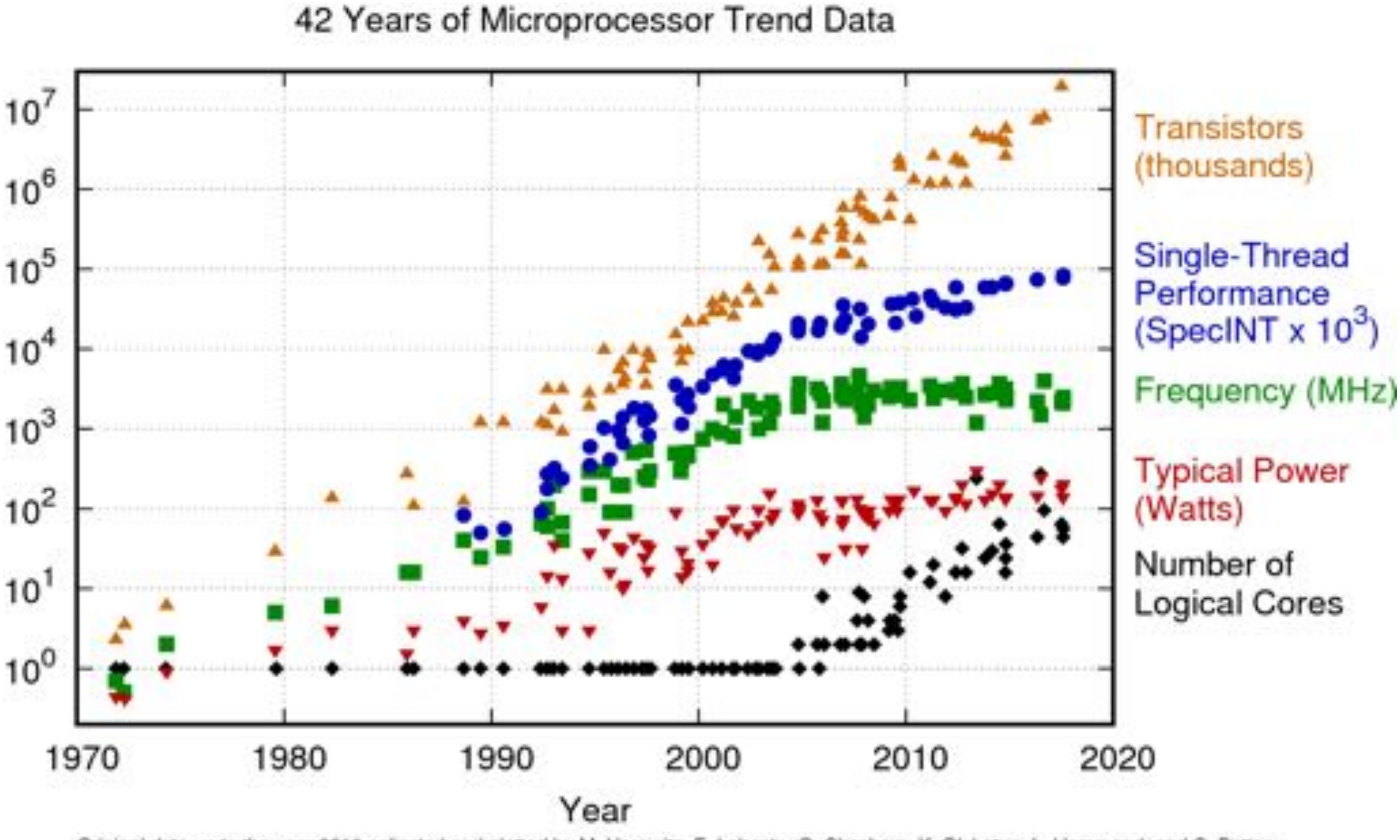
Books: Cuda by Example, Programming Massively Parallel Processors… etc.

Google!

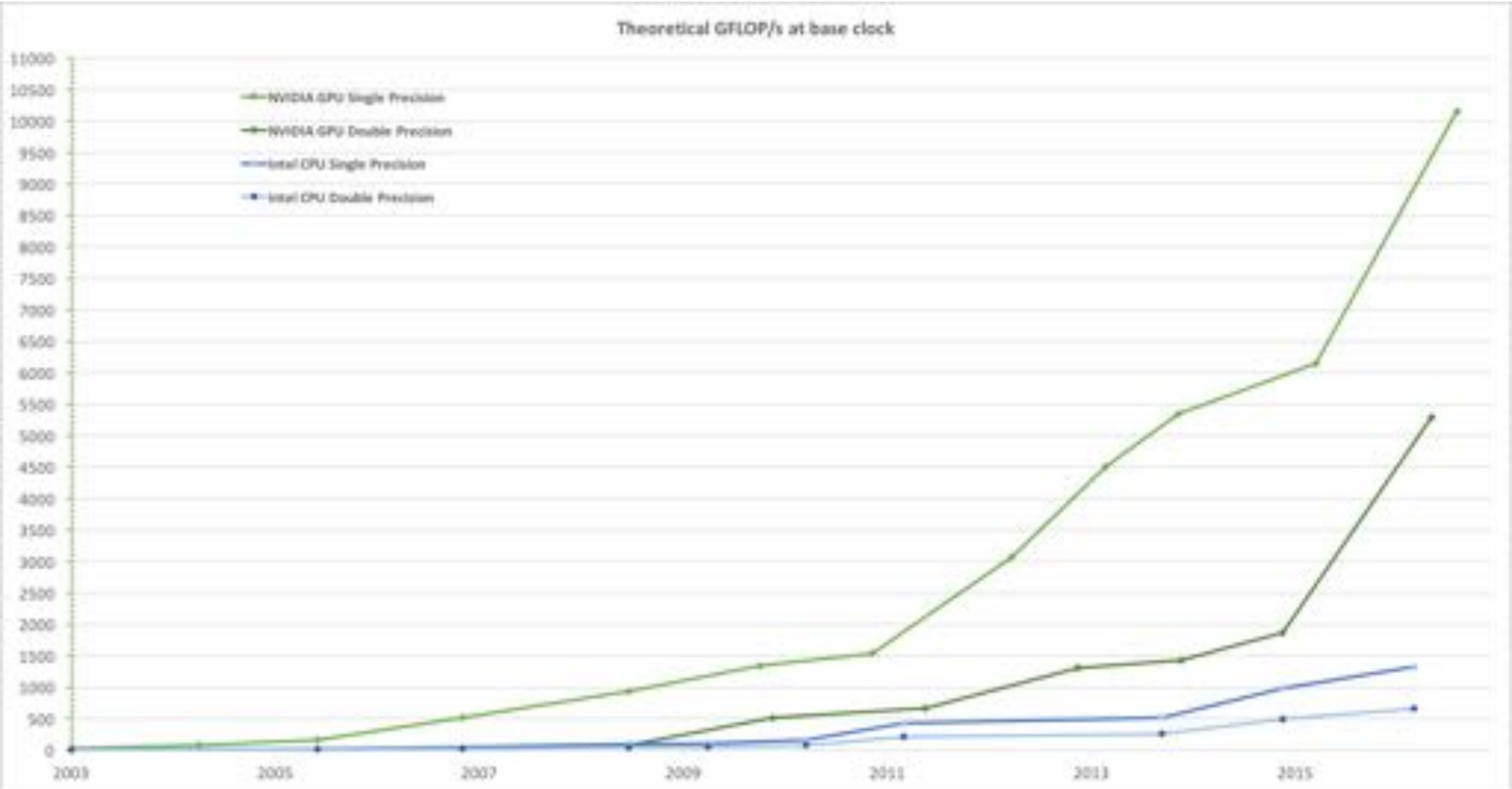
WHY WERE GPUS DEVELOPED?

WHY WERE GPUS DEVELOPED?

WHY WERE GPUS DEVELOPED?

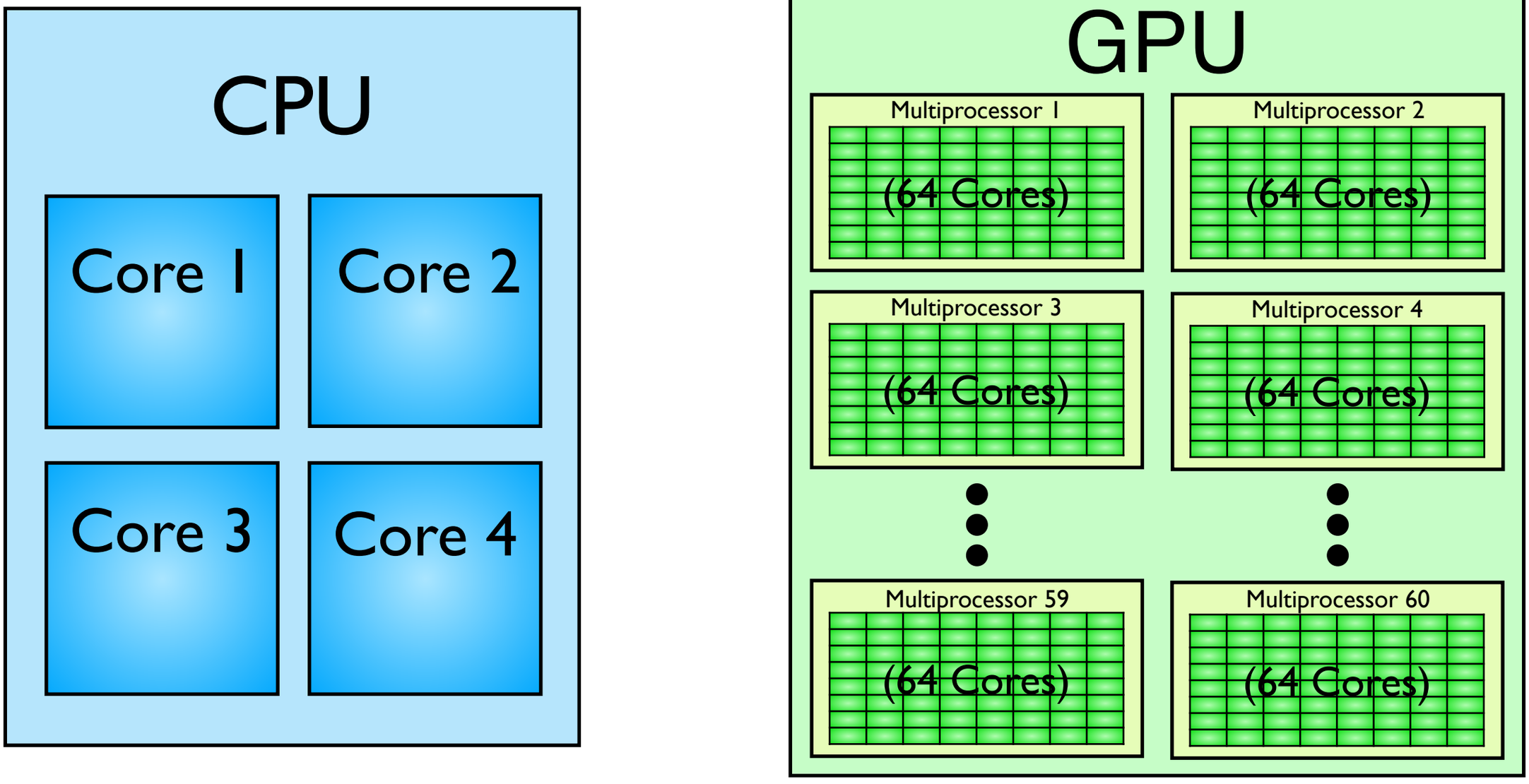


WHY WERE GPUS DEVELOPED?



WHAT ARE GPUS?

The GPU's compute cores are divided amongst dozens of Streaming Multiprocessors (SMs).

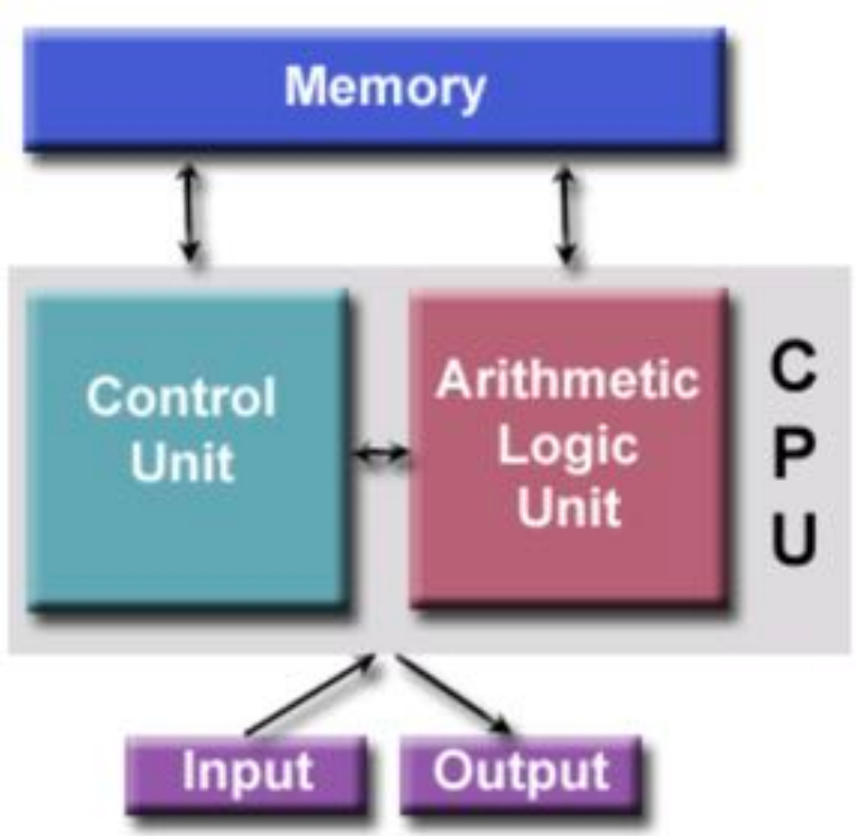


Modern GPUs have thousands of individual cores.

WHAT ARE GPUS?

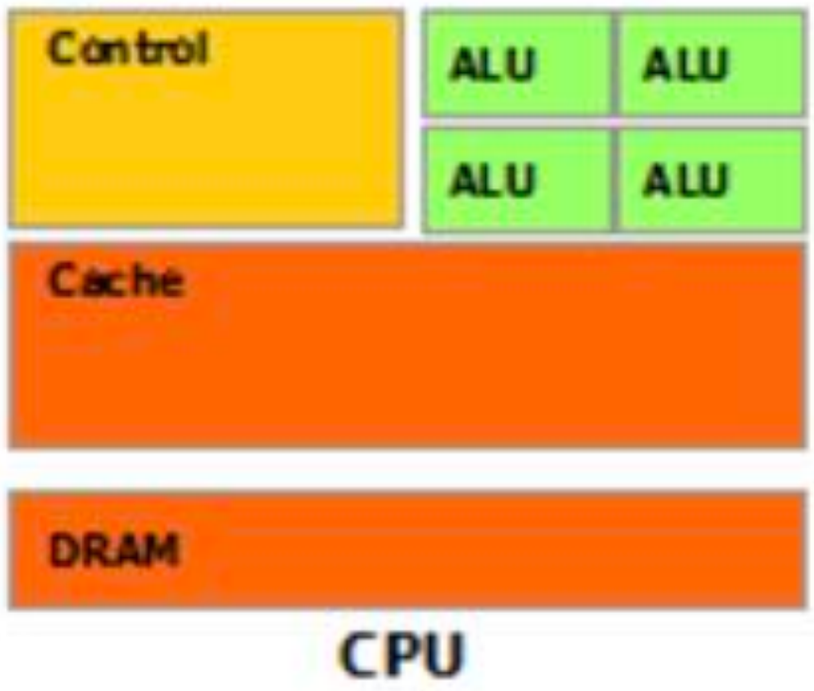
Computers comprised of four main components:

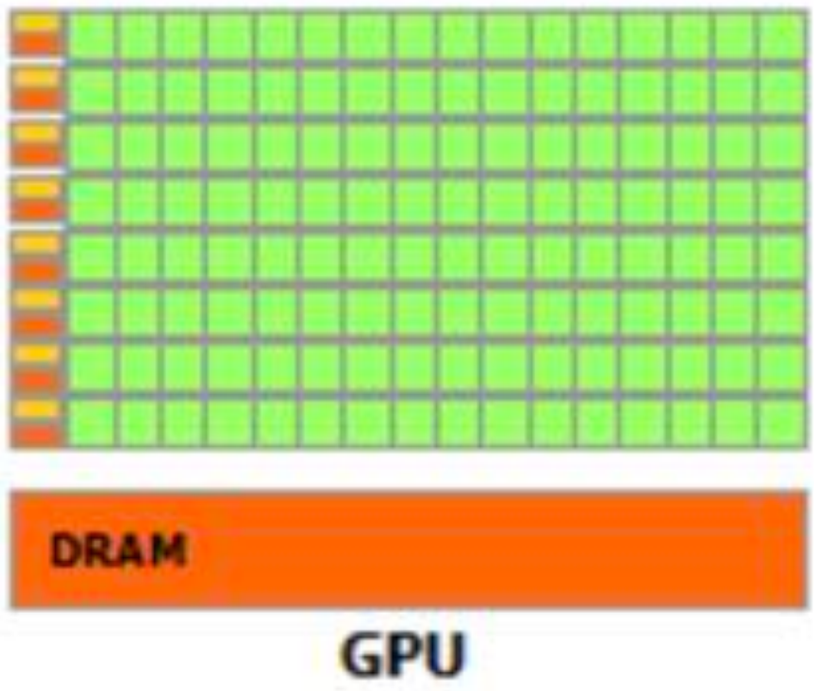
- Memory
- Control Unit
- Arithmetic Logic Unit
- Input / Output



WHAT ARE GPUS?

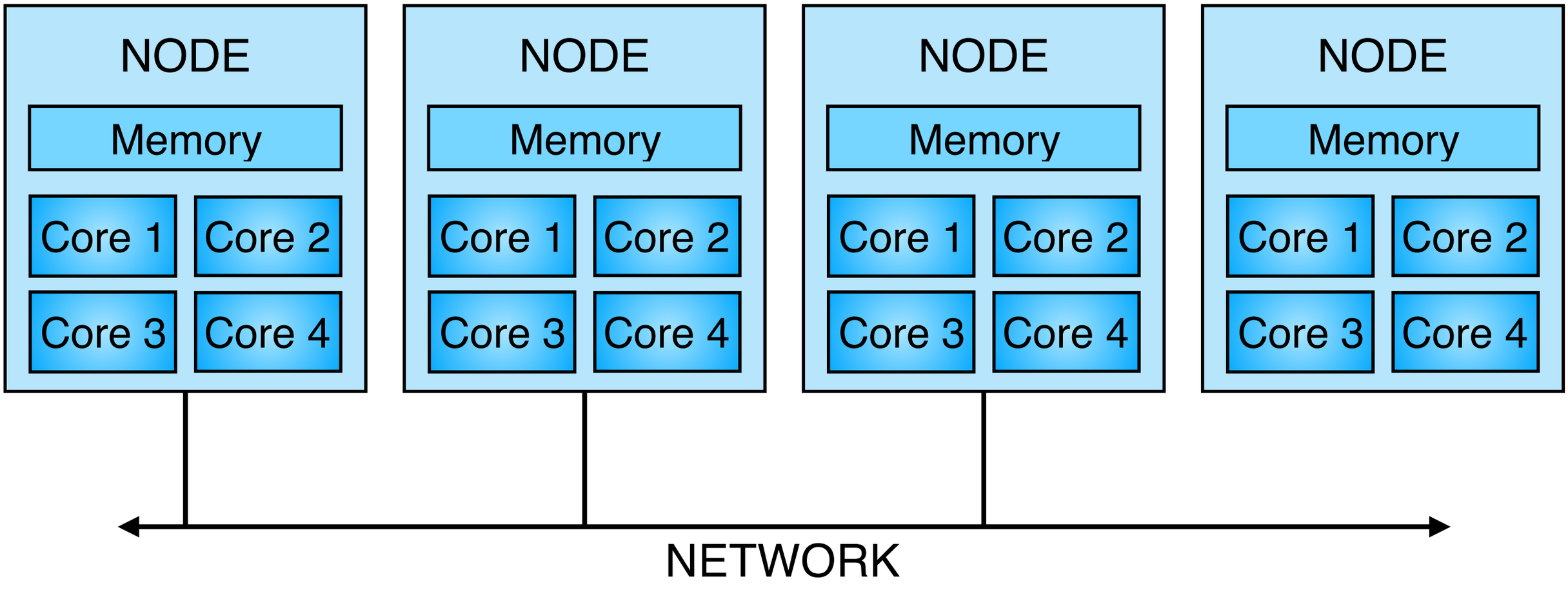
The GPU devotes far more transistors to data processing.



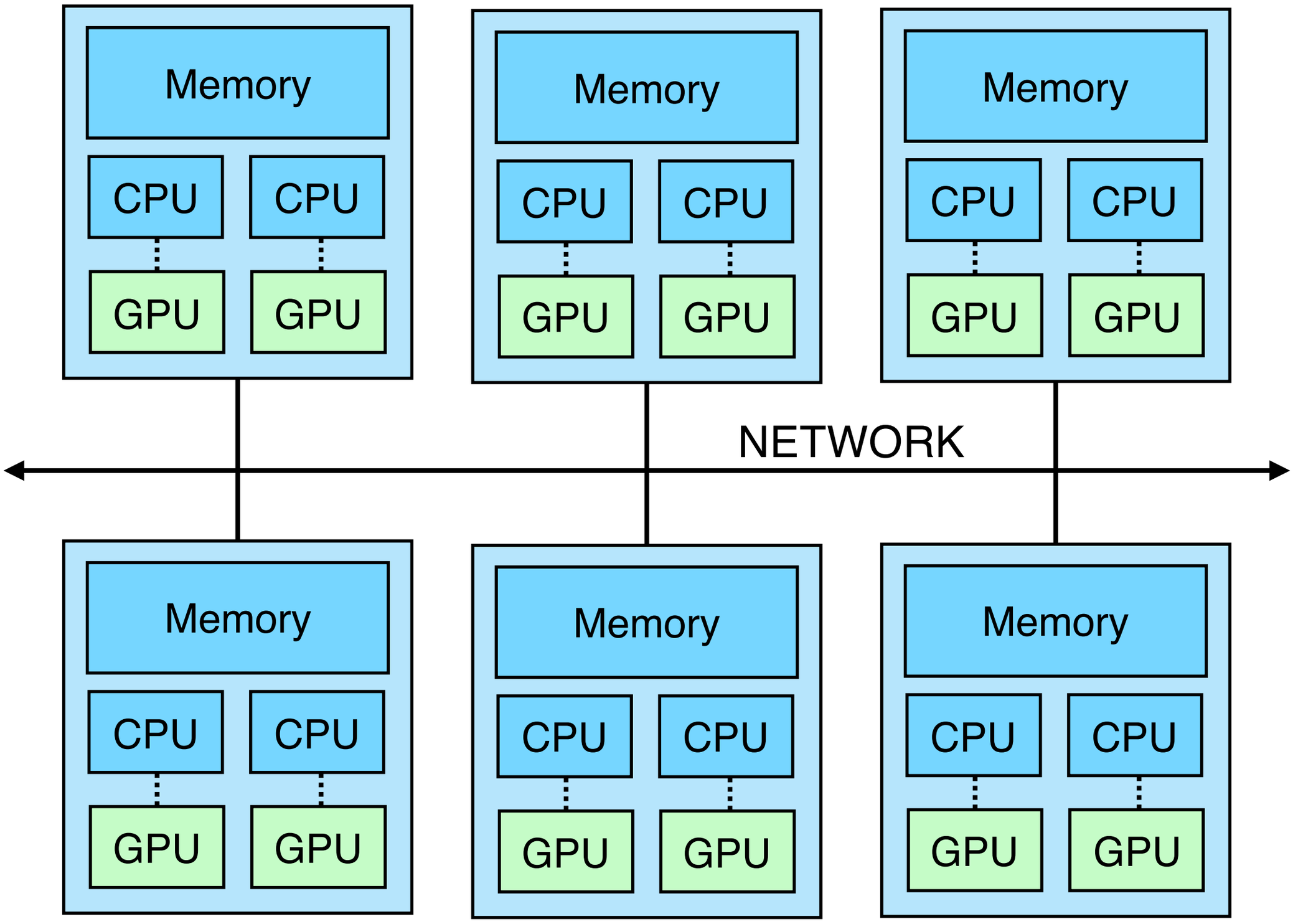


CURRENT ARCHITECTURES – TRADITIONAL ARCHITECTURE*

**Modern computing clusters typically have CPUs with far more cores than shown in this illustration.*

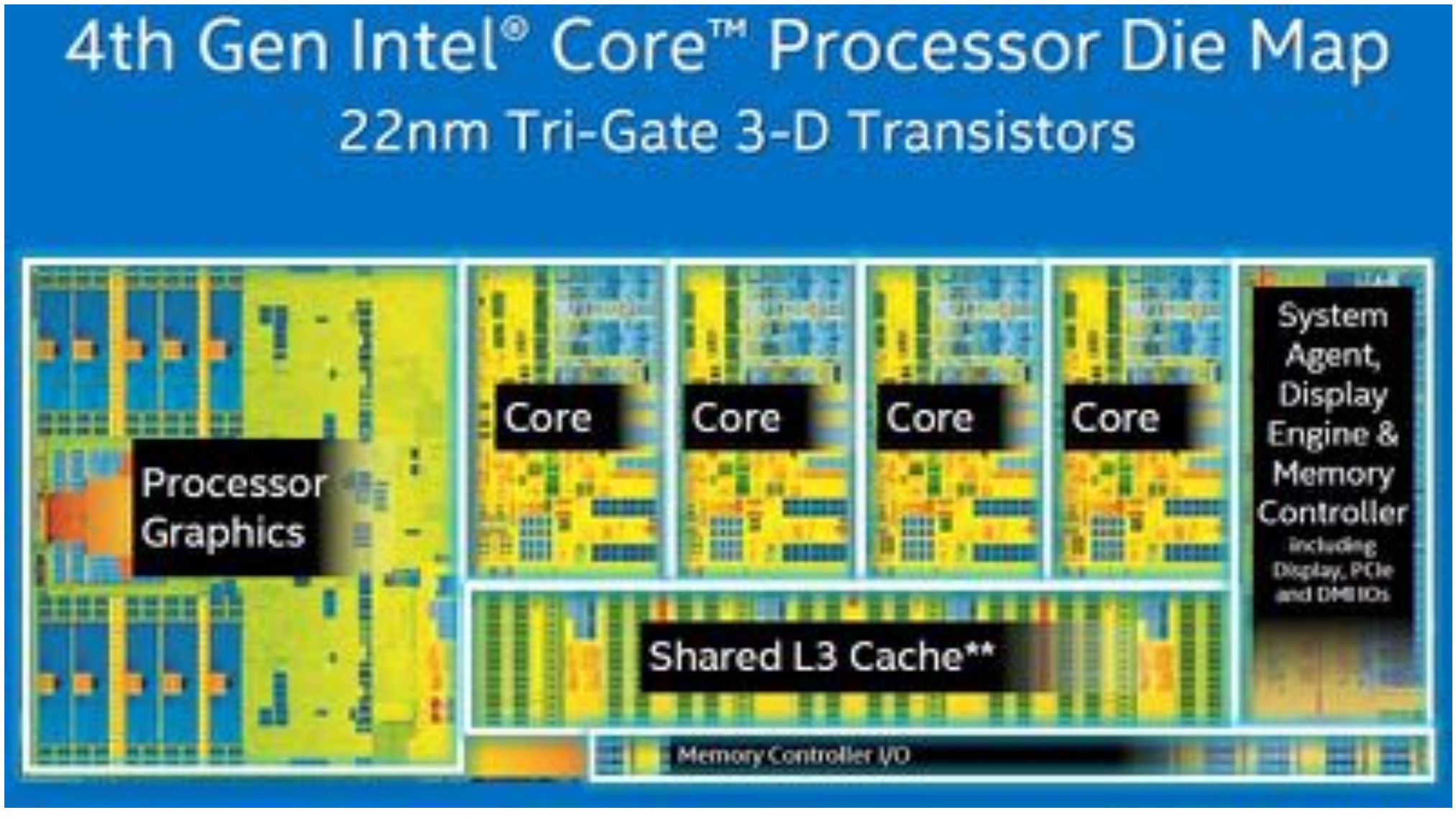


CURRENT ARCHITECTURES – HETEROGENEOUS ARCHITECTURE



CURRENT ARCHITECTURES

Even laptops now are heterogeneous!



GPUS IN HPC – THE TOP 500

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory **United States**	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband **IBM**	2,397,824	143,500.0	200,794.9	9,783
2	DOE/NNSA/LLNL **United States**	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband **IBM / NVIDIA / Mellanox**	1,572,480	94,640.0	125,712.0	7,438
3	National Supercomputing Center in Wuxi **China**	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway **NRCPC**	10,649,600	93,014.6	125,435.9	15,371
4	National Super Computer Center in Guangzhou **China**	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 **NUDT**	4,981,760	61,444.5	100,678.7	18,482
5	Swiss National Supercomputing Centre (CSCS) **Switzerland**	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 **Cray Inc.**	387,872	21,230.0	27,154.3	2,384

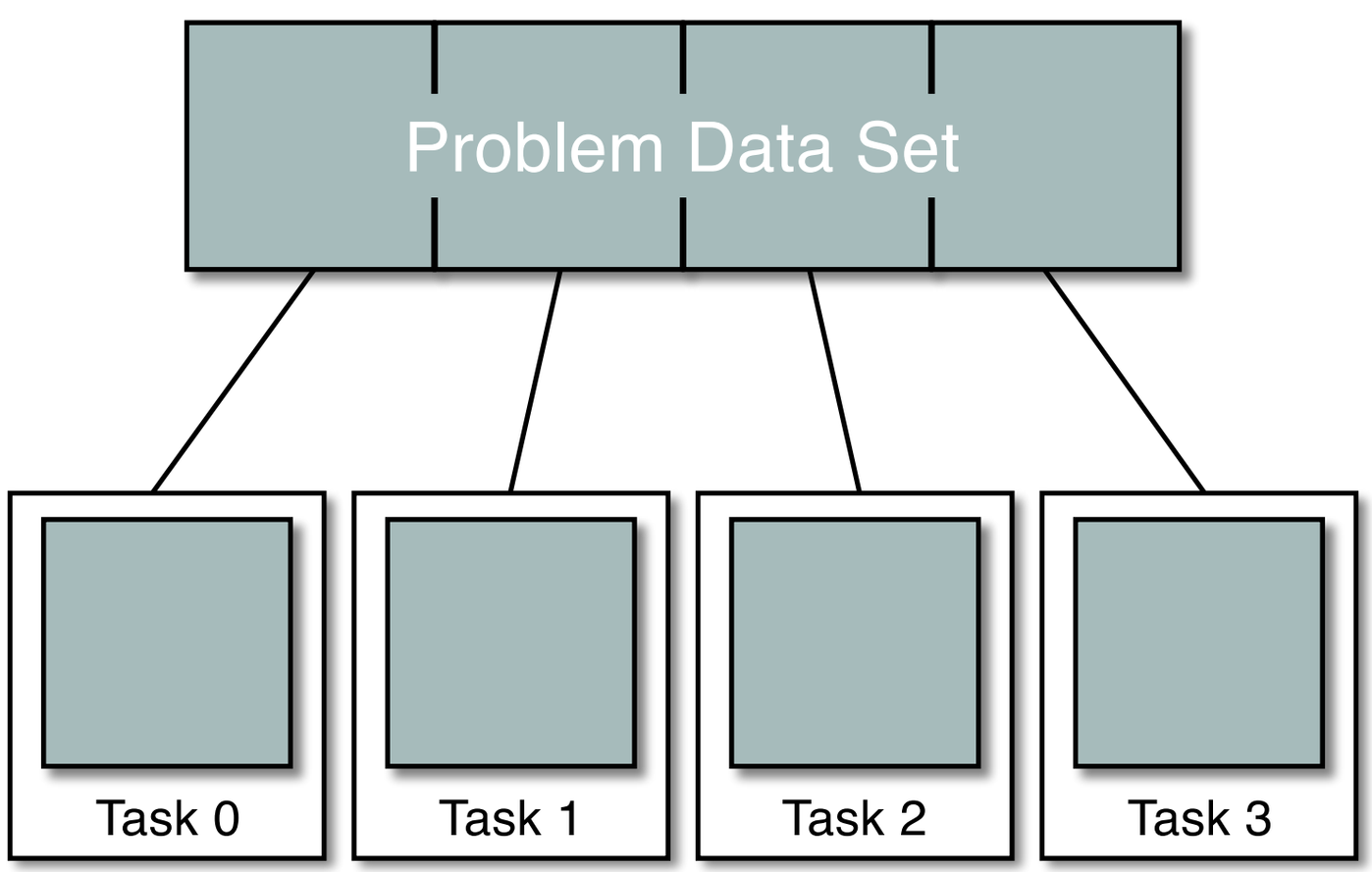
GPUS IN HPC – THE GREEN 500

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	375	**Shoubu system B** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , **PEZY Computing / Exascaler Inc.** Advanced Center for Computing and Communication, RIKEN **Japan**	953,280	1,063.3	60	17.604
2	374	**DGX SaturnV Volta** - NVIDIA DGX-1 Volta36, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla V100 , **Nvidia** NVIDIA Corporation **United States**	22,440	1,070.0	97	15.113
3	1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , **IBM** DOE/SC/Oak Ridge National Laboratory **United States**	2,397,824	143,500.0	9,783	14.668
4	7	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , **Fujitsu** National Institute of Advanced Industrial Science and Technology (AIST) **Japan**	391,680	19,880.0	1,649	14.423
5	22	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 , **HPE** GSIC Center, Tokyo Institute of Technology **Japan**	135,828	8,125.0	792	13.704

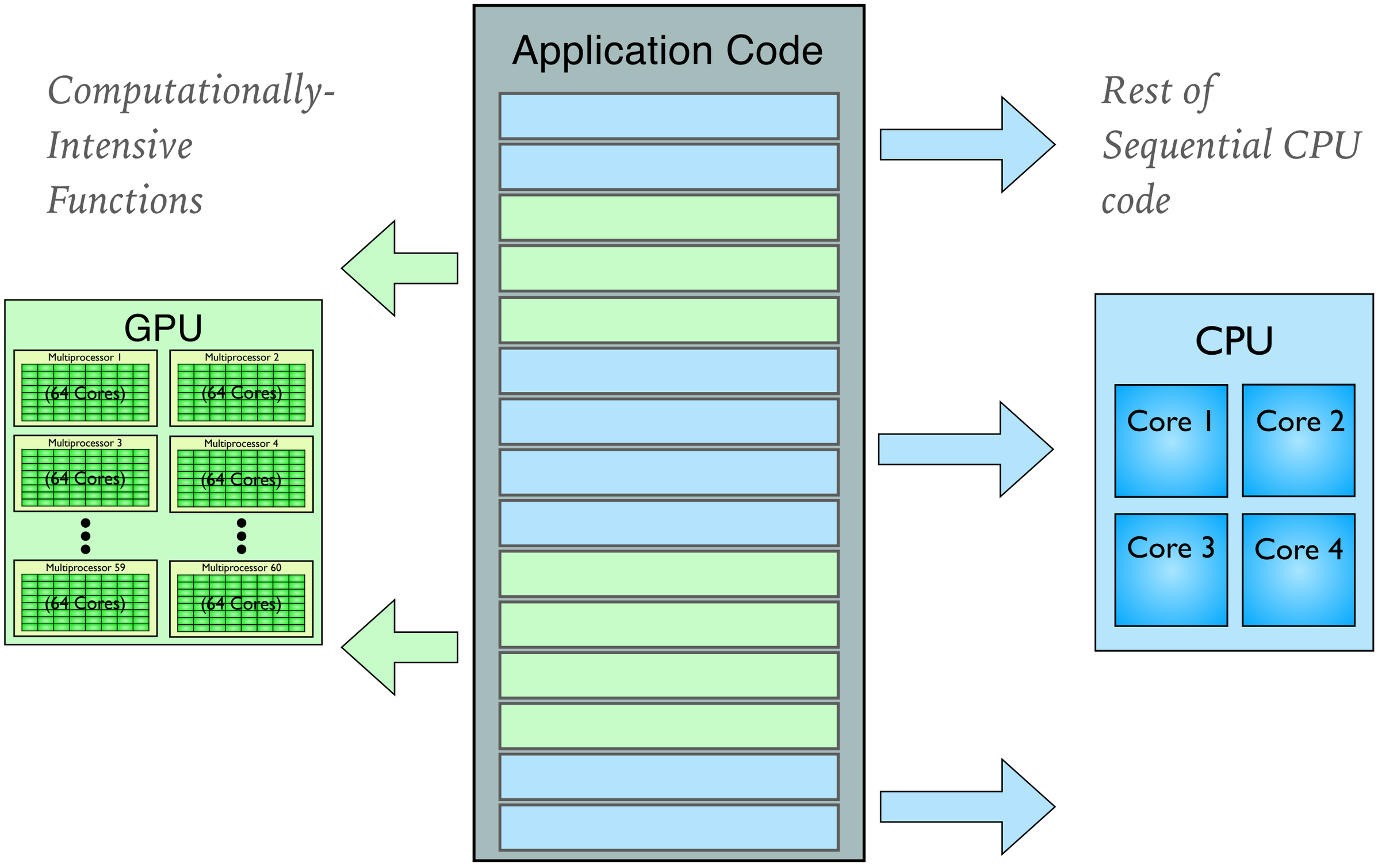
RECAP OF IMPORTANT PARALLEL PROGRAMMING CONCEPTS

SPMD - Single Program, Multiple Data

- Single Program: All tasks execute their copy of the same program, simultaneously.
- Multiple Data: Different tasks act on different portions of the data set.



WHERE GPUS HELP



GROUP PROBLEM – WHAT KINDS OF PROBLEMS WORK WELL ON GPUS?

- Dense linear algebra
- Large FFTs
- Massively parallel problems
  - Protein folding
  - N-body calculations
  - Astrophysical hydrodynamics…

GPU PROGRAMMING MODELS

- CUDA
  - Developed by NVIDIA for their GPUs
  - Supports C/C++
  - Often provides fastest execution
    - NVIDIA and 3rd party support for libraries (e.g. cuFFT)
- Numba - python CUDA interface for GPUs
- OpenCL - cross platform/cross vendor; supports C/C++
- OpenACC - vendor neutral; supports C/C++ and Fortran
- OpenMP - multi platform; supports C/C++ and Fortran

CUDA PROGRAMMING MODEL – KEY CONCEPTS

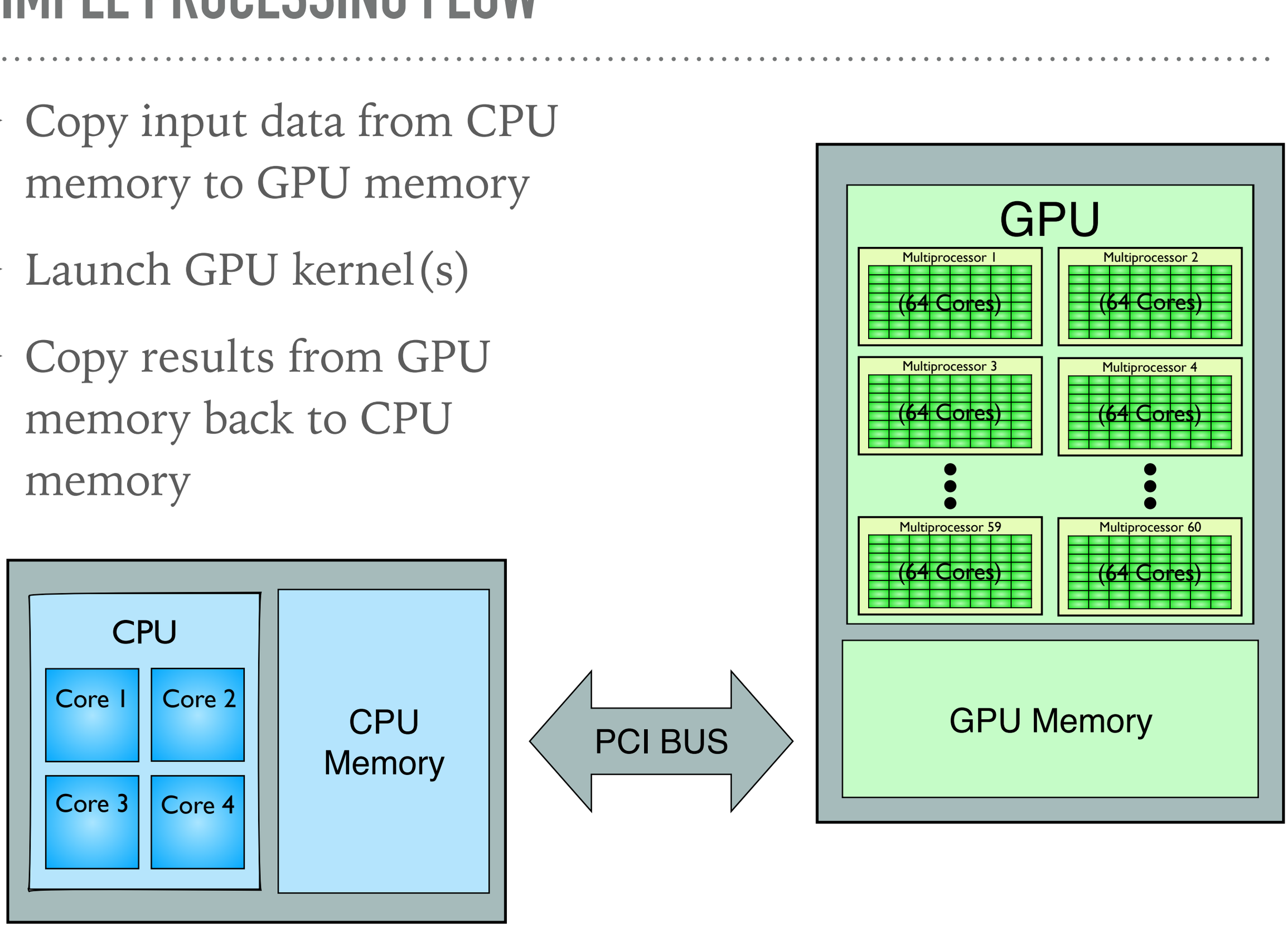
- Heterogeneous Computing
- Threads, Blocks, & Indexing
- Shared Memory
- Synchronization

SOME MORE TERMINOLOGY

- **Host**: The CPU and its memory (host memory)
- **Device**: The GPU and its memory (device memory)
- **Kernel Function**: A function that runs on the GPU, but is launched from the host
- **Device Function**: A function that runs on the device and is launched by a kernel

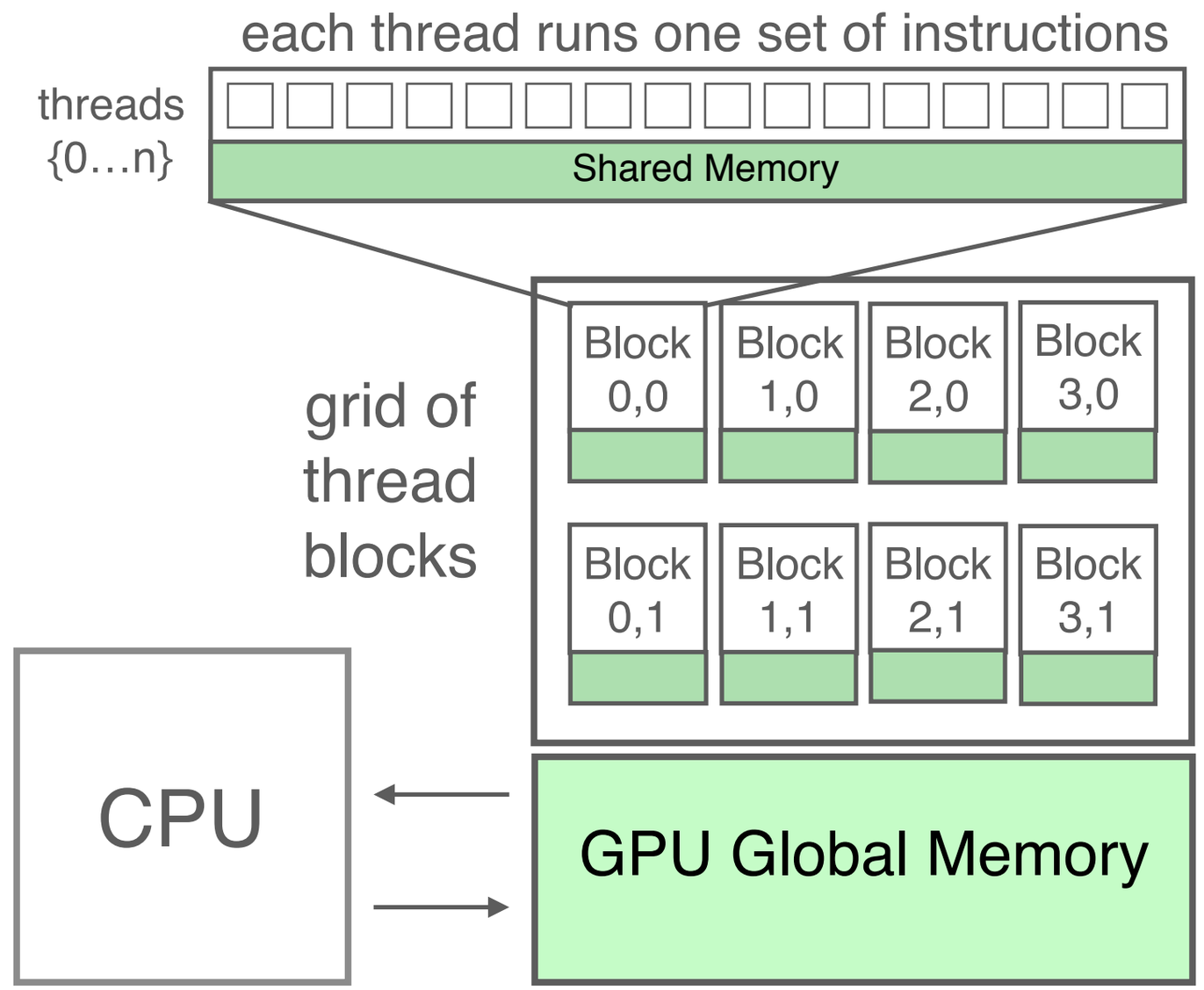
SIMPLE PROCESSING FLOW

- Copy input data from CPU memory to GPU memory
- Launch GPU kernel(s)
- Copy results from GPU memory back to CPU memory



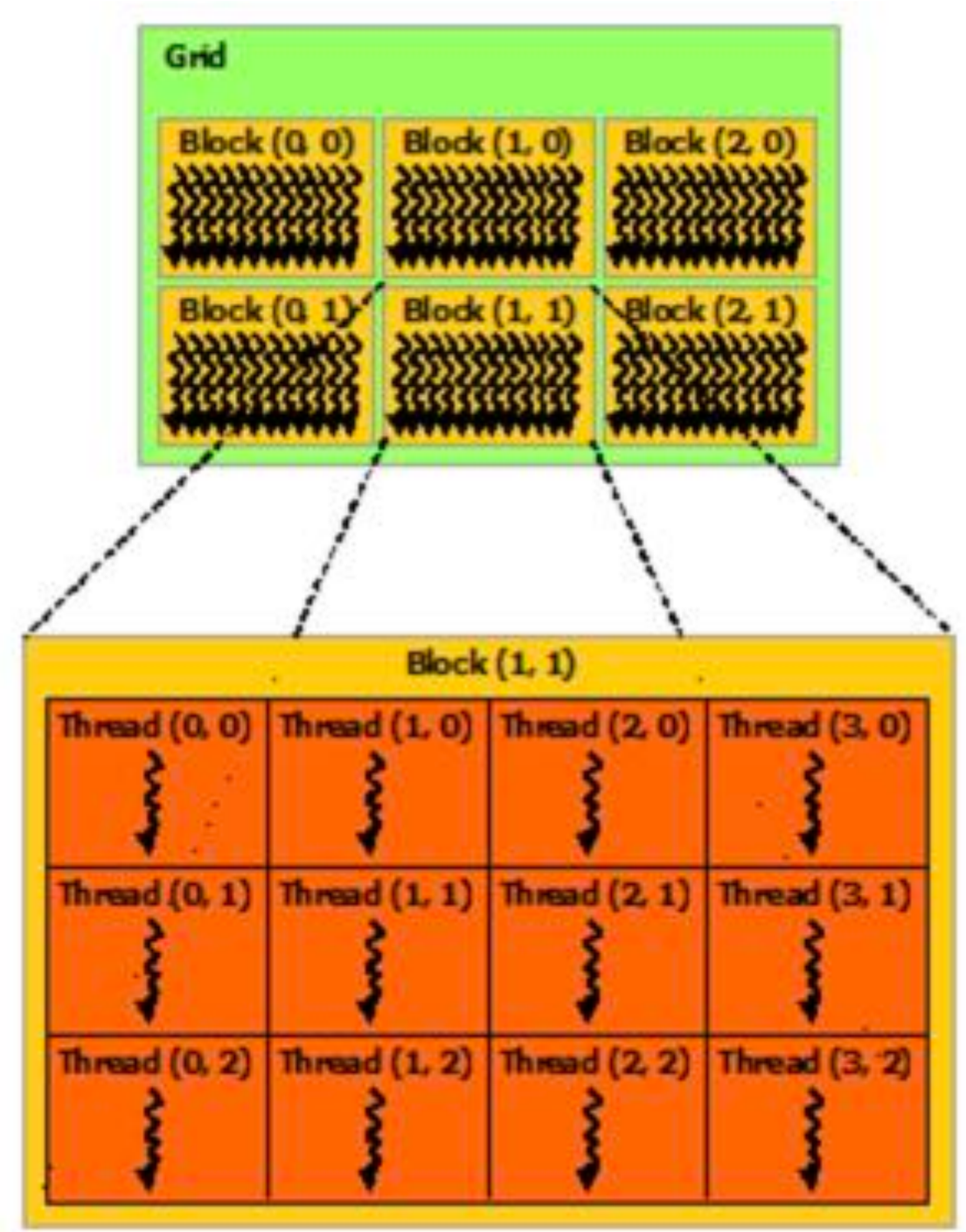
CUDA PROGRAMMING MODEL

- Cuda functions, called kernels, are launched from the host to run on the device
- Each kernel specifies a grid with a certain number of **blocks** and **threads**.
- Grids and blocks can be 1, 2, or 3D.



CUDA PROGRAMMING MODEL

- Cuda functions, called kernels, are launched from the host to run on the device
- Each kernel specifies a grid with a certain number of **blocks** and **threads**.
- Blocks are distributed on the GPU amongst the streaming multiprocessors.



MEMORY HIERARCHY

***Registers**: Local memory (per thread)*

***Shared Memory**: Semi-local memory (per block)*

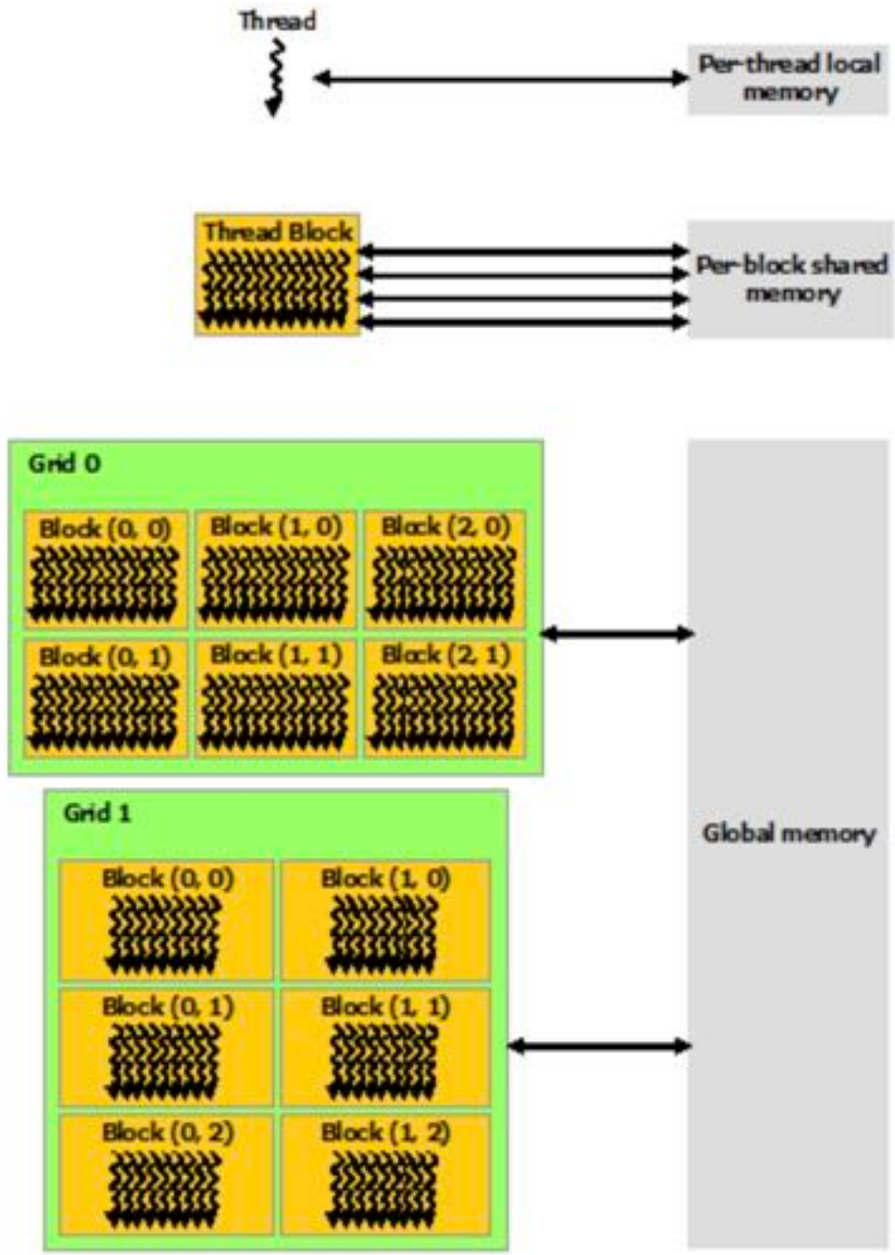
Typical numbers (48 - 128 KB)

***Global Memory**: Non-local memory (per GPU)*

Typical numbers (6 - 24 GB)

***CPU Memory**: Bus-access (per node)*

Typical numbers (12 - 256 GB)



If you do a lot of linear interpolation, ask me about texture memory.

HOW TO GPU – NUMBA VERSION

Fundamentals of GPU kernels.

- Kernels cannot explicitly return a value - all result data must be written to an array that is passed to the function
- Kernels explicitly declare their thread block hierarchy when called.

```
threadsperblock = 32
blockspergrid = (an_array.size + (threadsperblock - 1)) // threadsperblock
increment_by_one[blockspergrid, threadsperblock](an_array)
```

```
@cuda.jit
def increment_by_one(an_array):
    """
    Increment all array elements by one.
    """
```

HOW TO GPU – NUMBA VERSION

Fundamentals of GPU kernels.

- Block size determines how many threads share memory
- Block size must be big enough to keep all the execution units occupied
- Threads execute in groups of 32 (a **warp**), a good rule-of-thumb is to always use block sizes that have a multiple of 32 threads

USING BLOCKS AND THREADS TO INDEX FUNCTIONS

Every thread in the kernel executes the same code. So, there must be some way for a thread to know "who it is", so that it knows which data it is responsible for working on.

```
@cuda.jit
def increment_by_one(an_array):
    # Thread id in a 1D block
    tx = cuda.threadIdx.x
    # Block id in a 1D grid
    ty = cuda.blockIdx.x
    # Block width, i.e. number of threads per block
    bw = cuda.blockDim.x
    # Compute flattened index inside the array
    pos = tx + ty * bw
    if pos < an_array.size:  # Check array boundaries
        an_array[pos] += 1
```

USING BLOCKS AND THREADS TO INDEX FUNCTIONS

Every thread in the kernel executes the same code. So, there must be some way for a thread to know "who it is", so that it knows which data it is responsible for working on.

A Numba shortcut:

```
@cuda.jit
def increment_by_one(an_array):
    pos = cuda.grid(1)
    if pos < an_array.size:
        an_array[pos] += 1
```

```
@cuda.jit
def increment_a_2D_array(an_array):
    x, y = cuda.grid(2)
    if x < an_array.shape[0] and y < an_array.shape[1]:
        an_array[x, y] += 1
```