KOÇ UNIVERSITY

Parallel Programming on Accelerators

Didem Unat

COMP 429/529 Parallel Programming

So Far

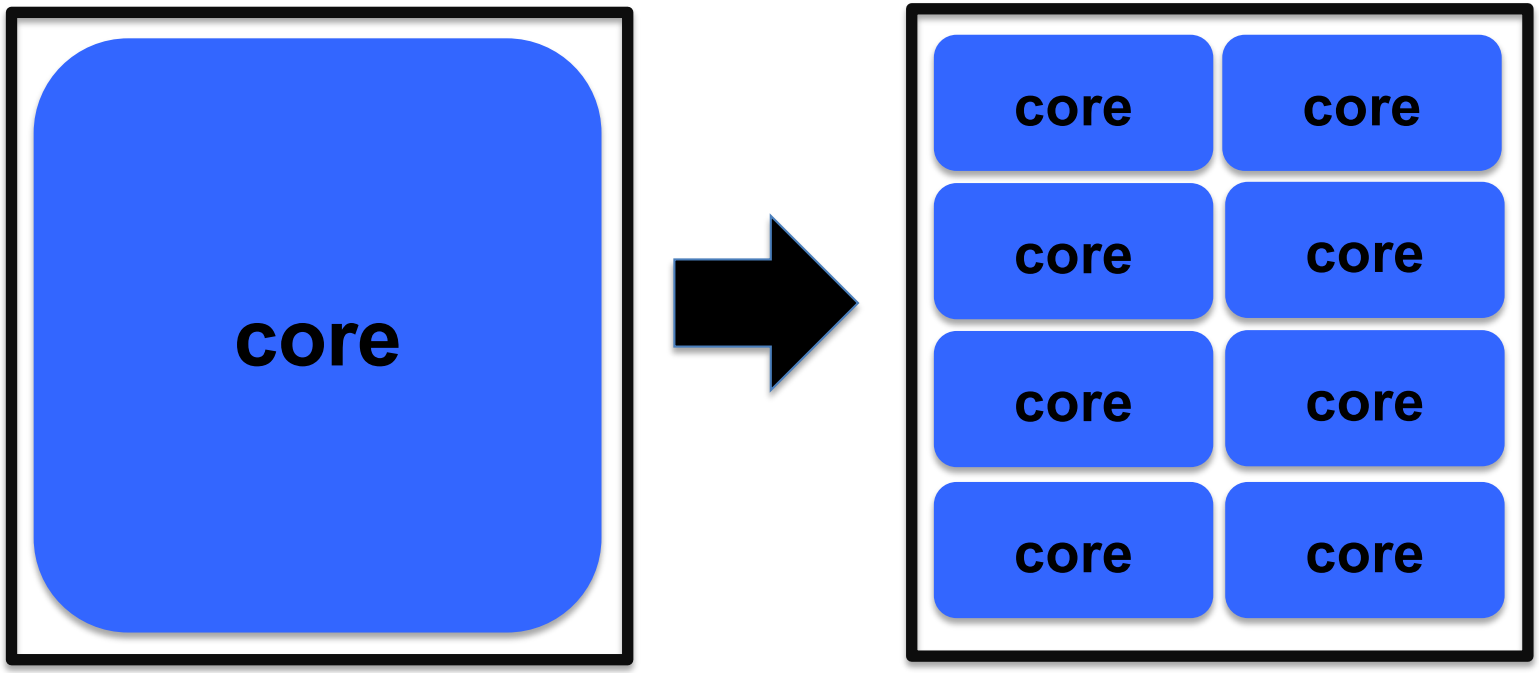
We have covered

- Parallel Architectures
- Performance Characterization
  - Speedup. Scalability and Efficiency
- Shared Memory Programming Model
  - Examples from OpenMP
- Different types of parallelism
  - Task vs Data Parallelism, Nested Parallelism
- Task Graphs

Multicore Era

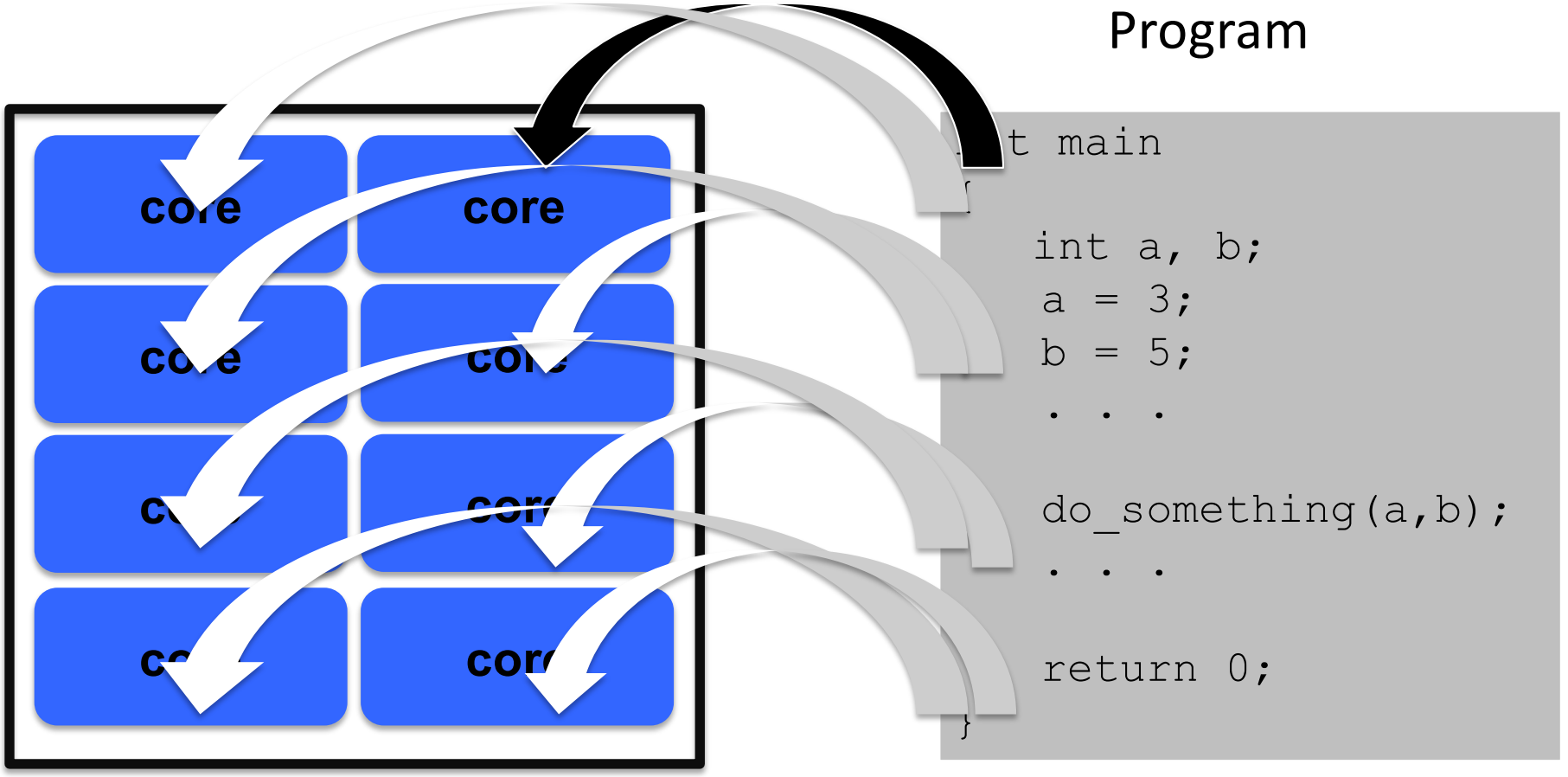
Massive on-chip parallelism

- Reinterpretation of Moore's law
- No clock increases → hundreds of simple "cores" per chip
- A thousand cores on a chip by 2022
  - Already have 72-core chips (intel's Knights Landing)
  - 260 cores per processor in Sunway Supercomputer
- 1 Million to 1 Billion-way system level parallelism



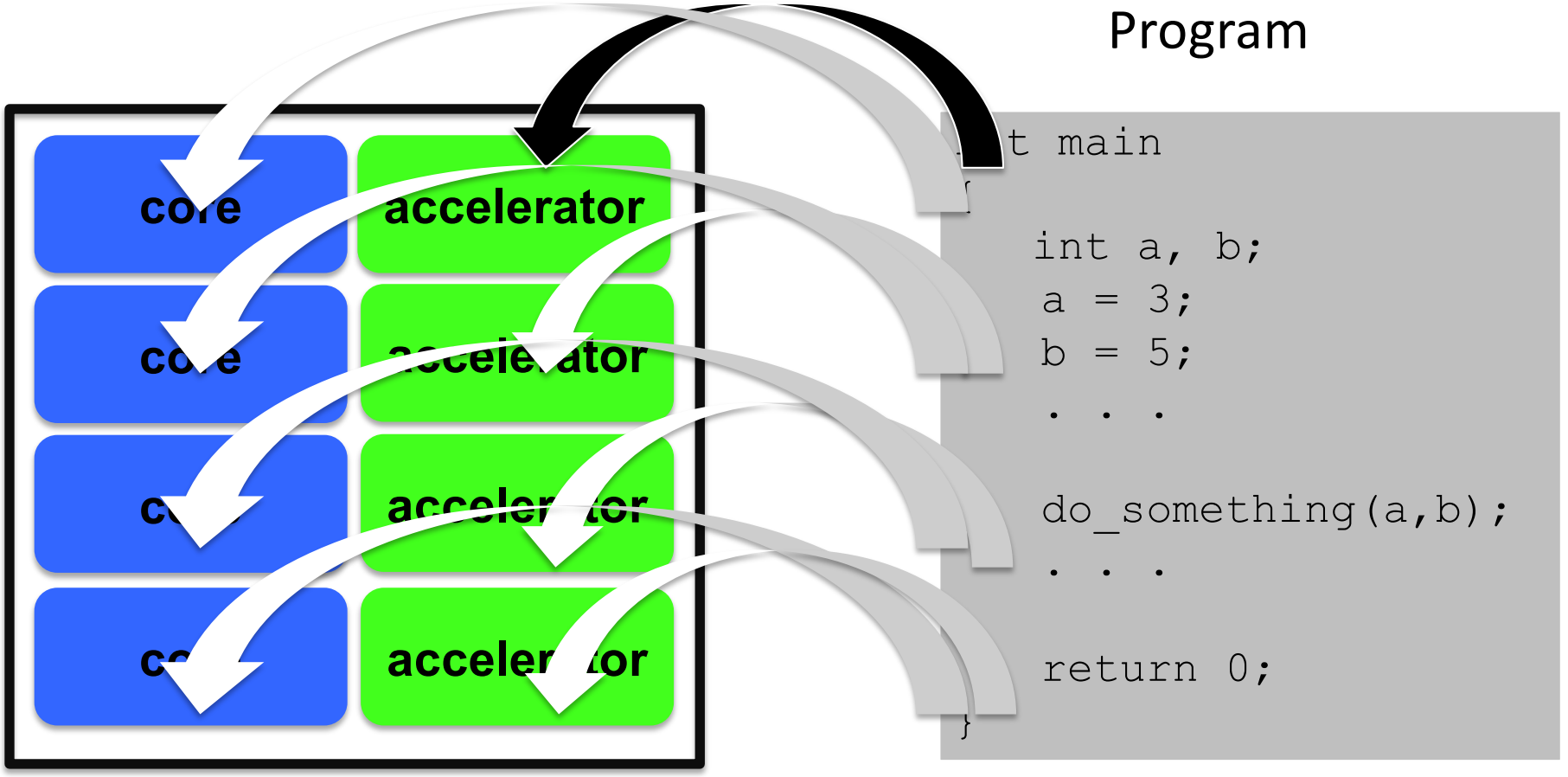
Multicore-Based Systems

- Computers no longer get faster, just wider
- Data-parallel computing is most scalable solution
- Some problems are better suited to task parallelism



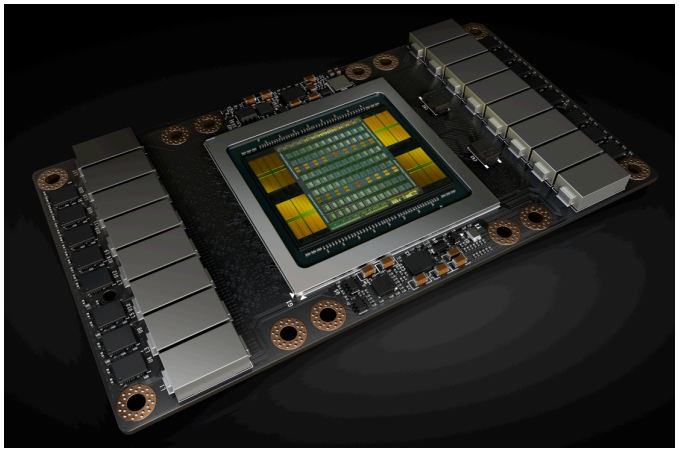
Accelerator-Based Systems

- Use the best match accelerator for the job
- Heterogeneity in the system: different types of cores



#2 in Top 500

- Summit is #2 Supercomputer in the world
- 2.4 Million cores
- 4,608 Compute nodes
- 27648 GPUs in total
- 6 x Nvidia Volta GV100 per node

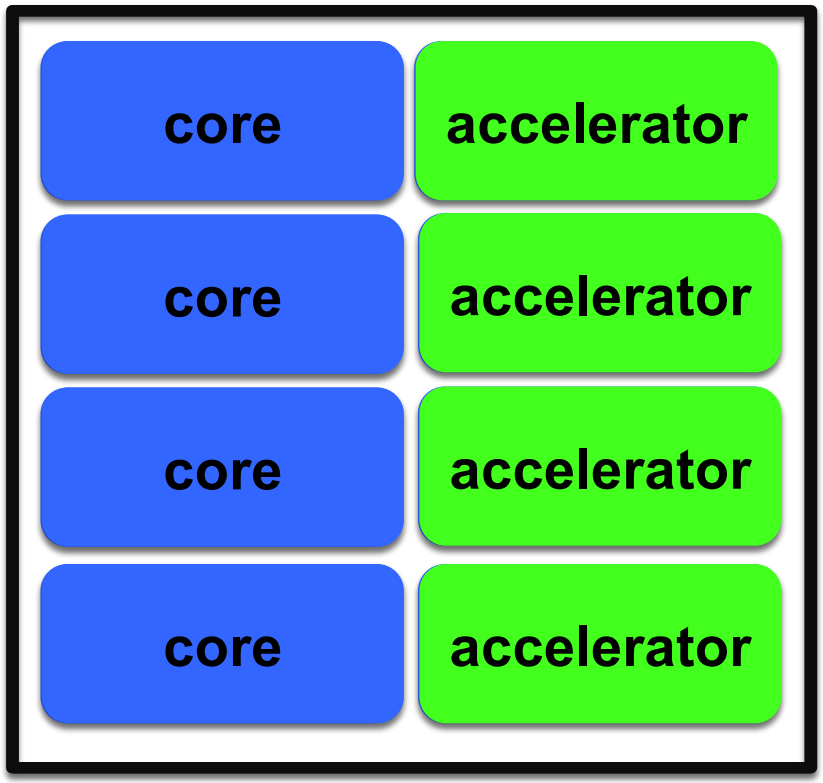


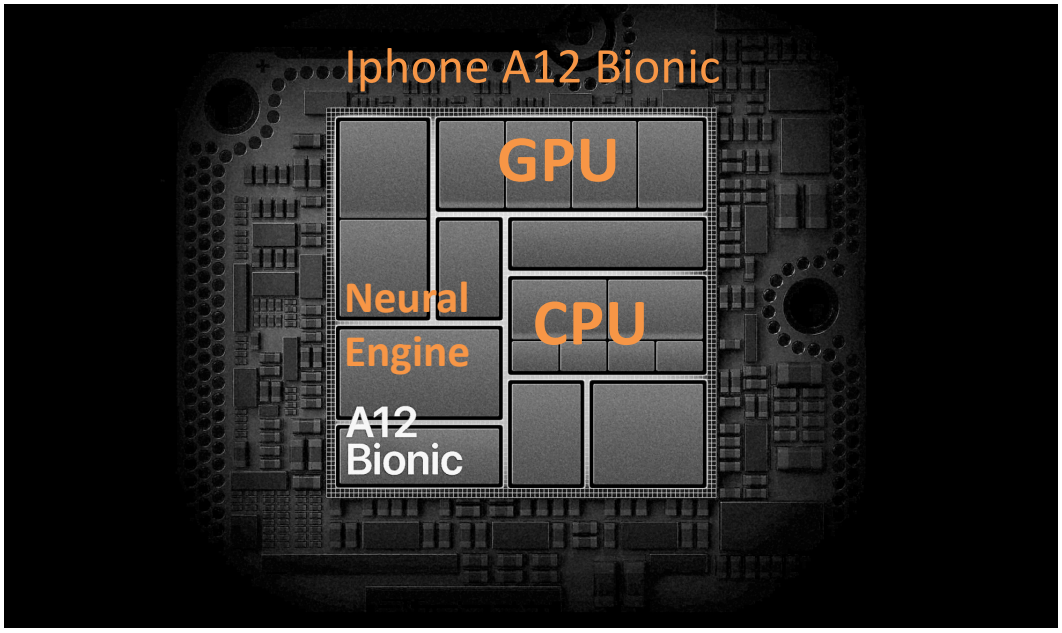
Nvidia Volta GV100

Summit in USA Peak 200 Petaflops
sustained ~148 Petaflops

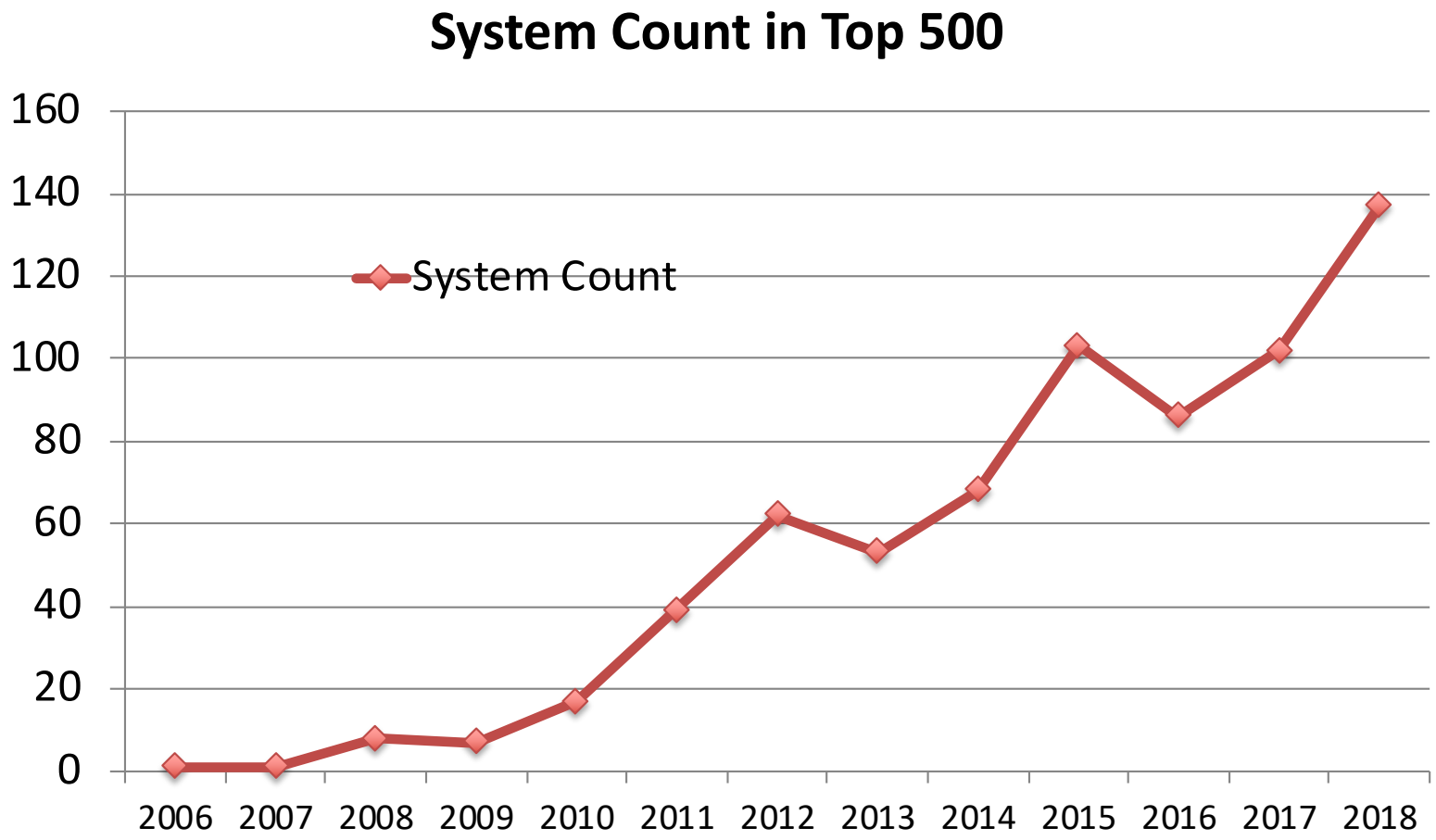
Cell Phones

heterogeneity



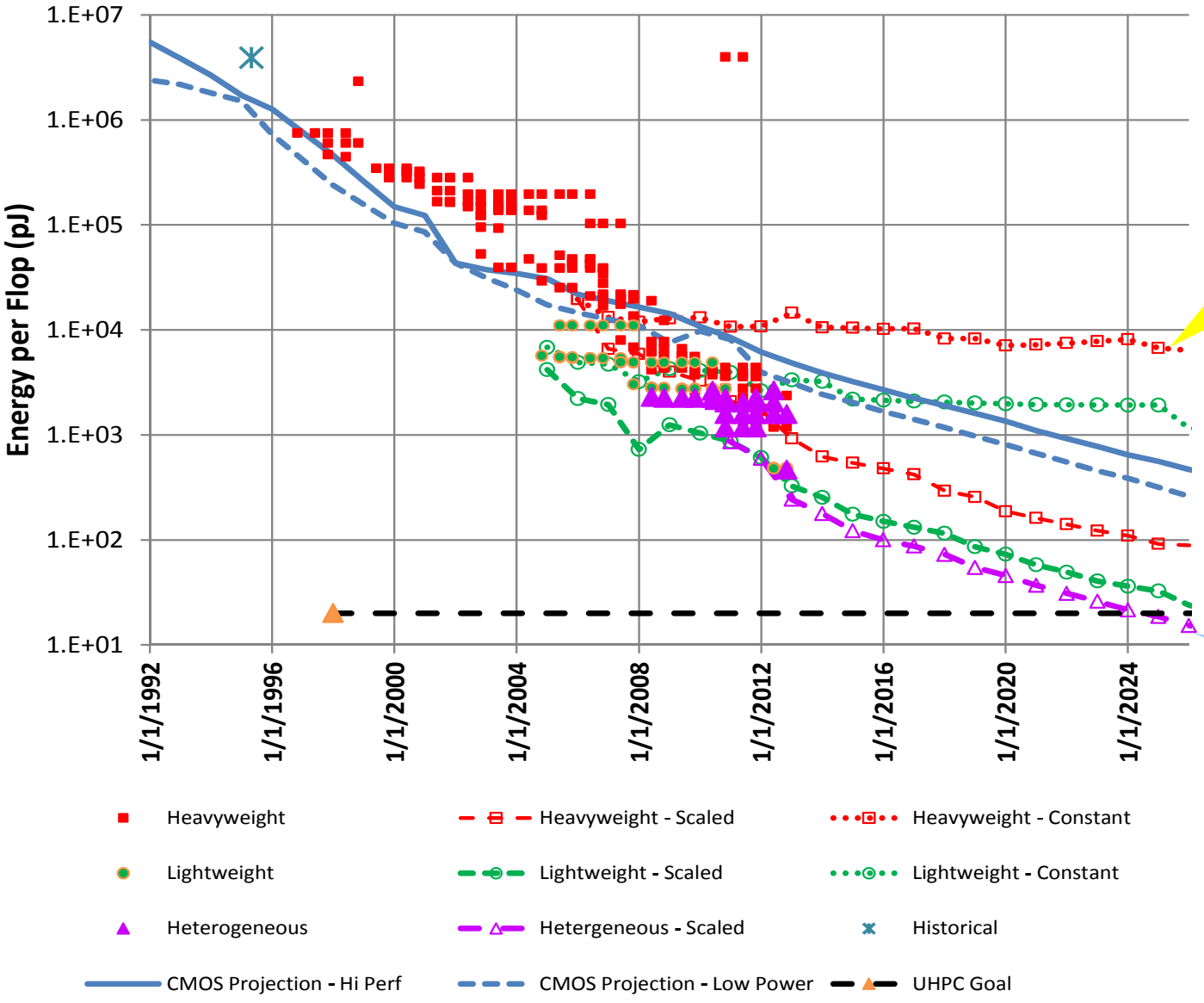


Number of Systems with Accelerators in Top 500



Seven systems in Top 10 in the world are equipped with accelerators, mainly Nvidia GPUs.

Mere Multi-Core is NOT good enough!
(need to go to simpler cores)

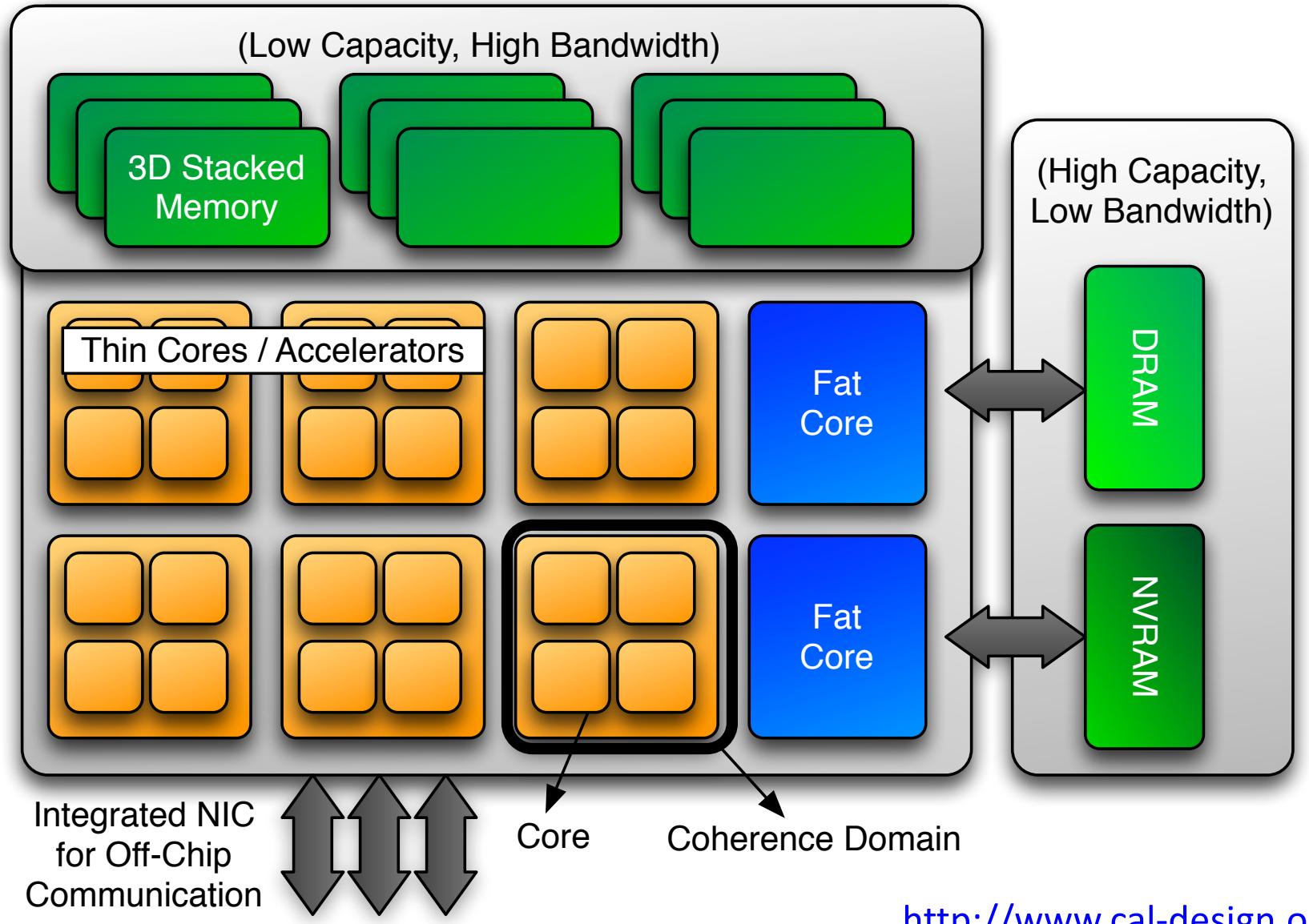


Can continue with conventional x86 architectures

Lightweight cores **OR** Hybrid is the only approach that provides energy per flop improvement.

Image Src: John Shalf

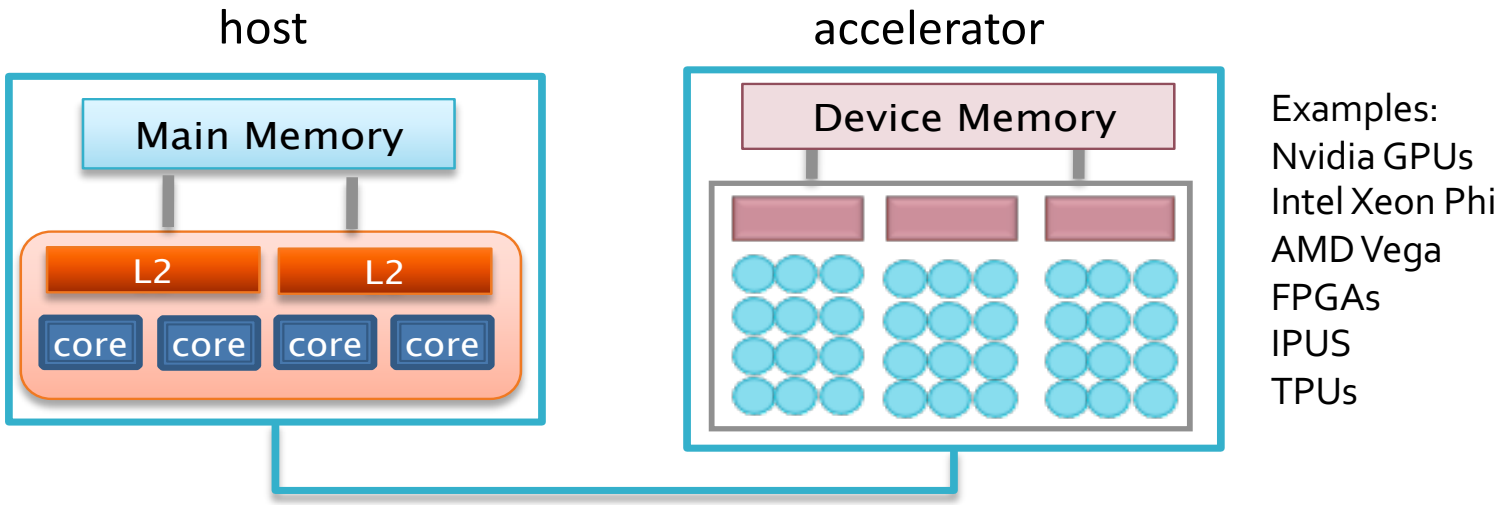
Abstract Machine Model for Exascale



http://www.cal-design.org/

Heterogeneous Programming

- Accelerators
  - Typically attached to a host (a multicore CPU)
  - Massively parallel single chip processor
  - Low power cores: trade off single thread performance
  - Offload compute intensive parts of the program to the accelerator
  - There is a bus connecting accelerator to the host

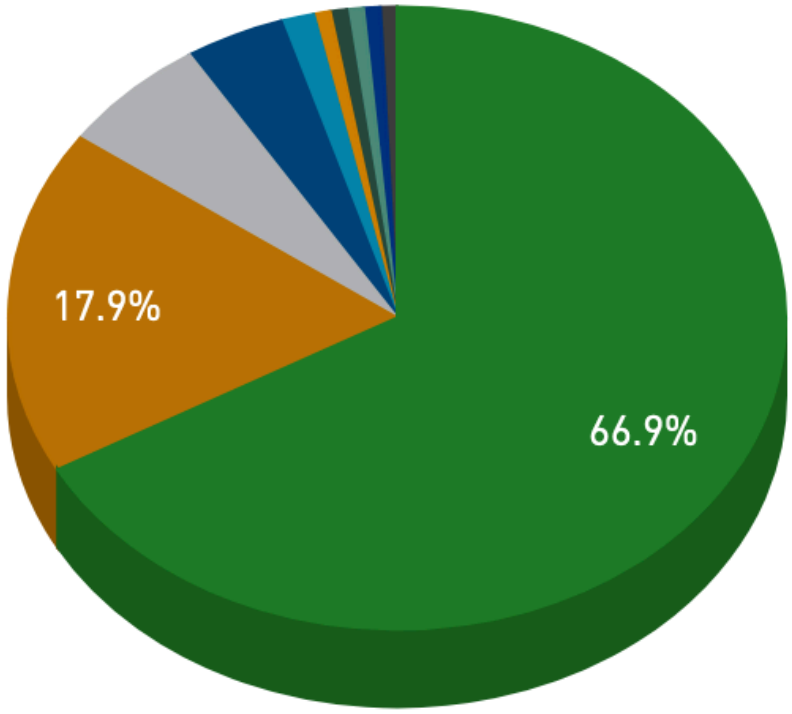


GPU Architectures

- In this class, we will focus on Nvidia GPU accelerators
  - There are other types of accelerators
  - TPUs, IPUs, FPGAs, AMD GPUs

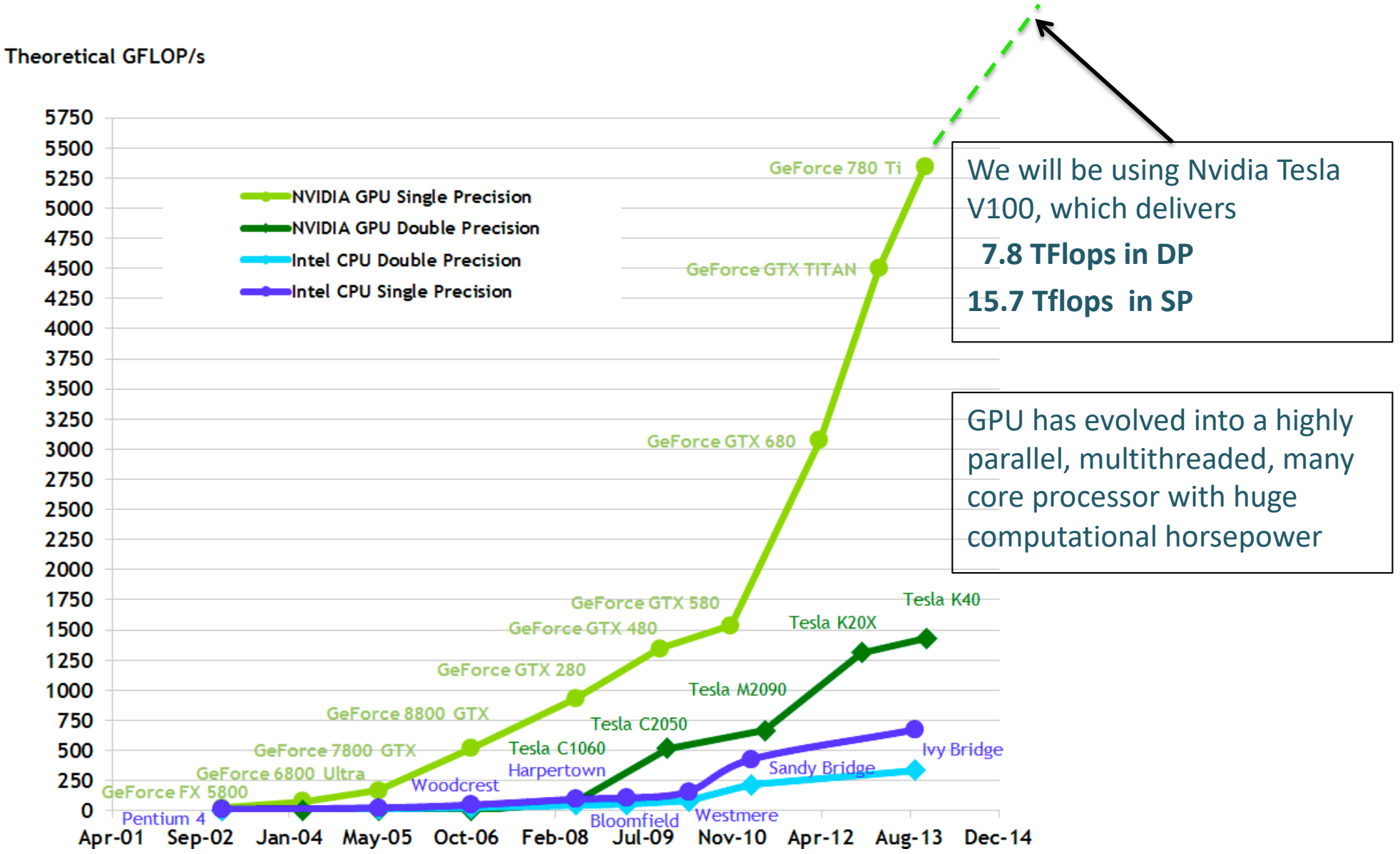
Distribution in Top500

Accelerator/CP Family System Share



	Count
NVIDIA Volta	97
NVIDIA Ampere	26
NVIDIA Pascal	9
NVIDIA Kepler	6
Intel Xeon Phi	2
Matrix-2000	1
MN-Core	1
AMD Vega	1
NVIDIA Fermi	1
Hybrid	1

Computational Performance (marketing slides)



We will be using Nvidia Tesla V100, which delivers

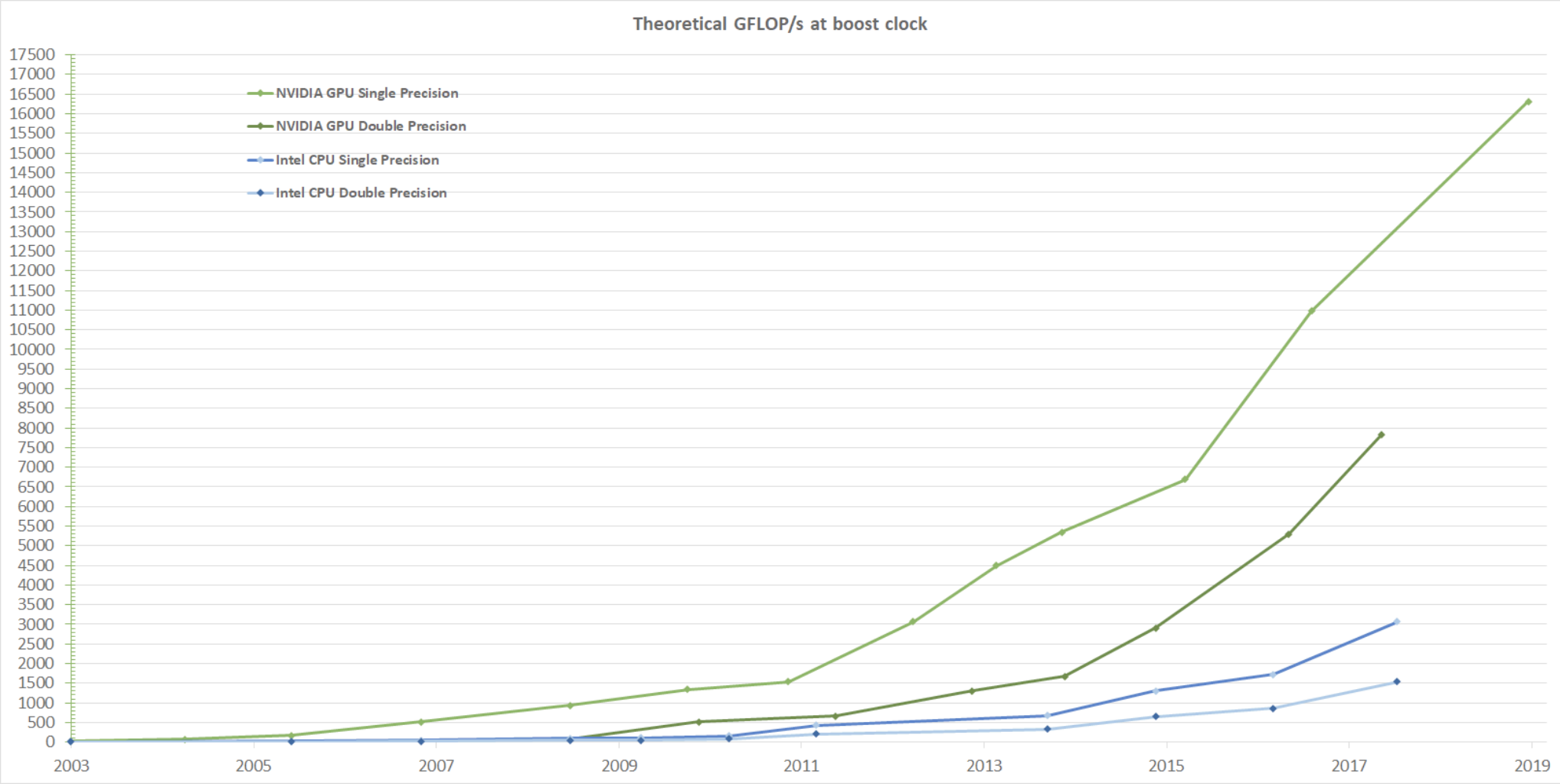
7.8 TFlops in DP

15.7 Tflops in SP

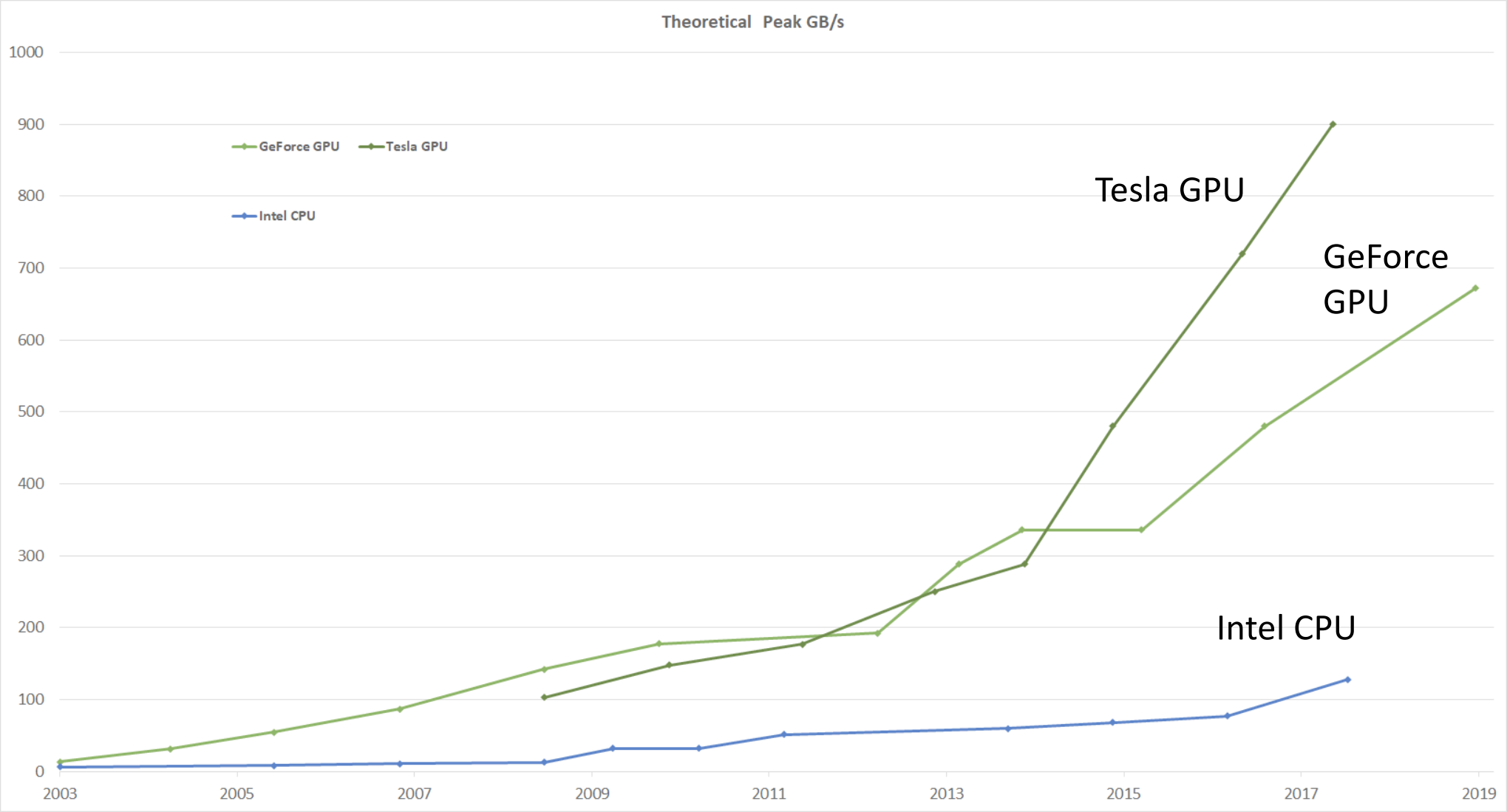
GPU has evolved into a highly parallel, multithreaded, many core processor with huge computational horsepower

Img. Src Nvidia

Newer GPUs – FLOPs rate



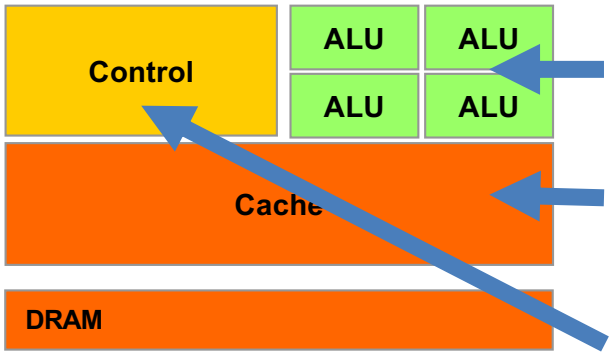
Memory Bandwidth GB/s



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

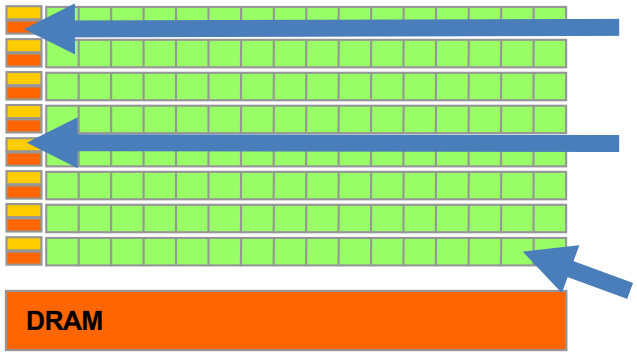
CPUs vs GPUs

CPU



- Powerful ALU
  - Reduced operation latency
- Large caches
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

GPU

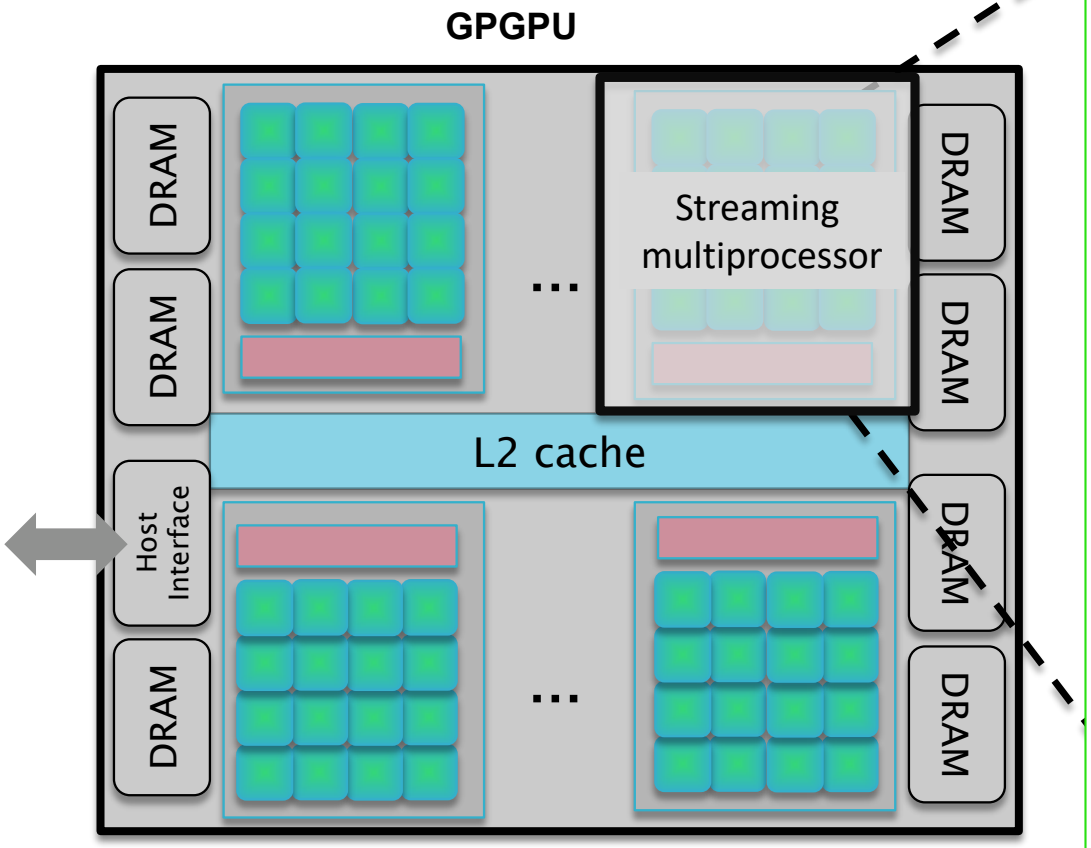


- Small caches
  - multithreading can hide latency => skip the big caches
- Simple control and energy efficient ALUs
  - No branch prediction and No data forwarding
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

Why is GPU different from a CPU?

- Different goals produce different designs
  - GPU assumes workload is highly parallel
  - CPU must be good at everything, parallel or not
- CPU: minimize latency experienced by 1 thread
  - Few number of cores – 10s
- GPU: maximize throughput of all threads
  - 1000s of cores
  - Tesla V100 has 5376 CUDA FP32 cores

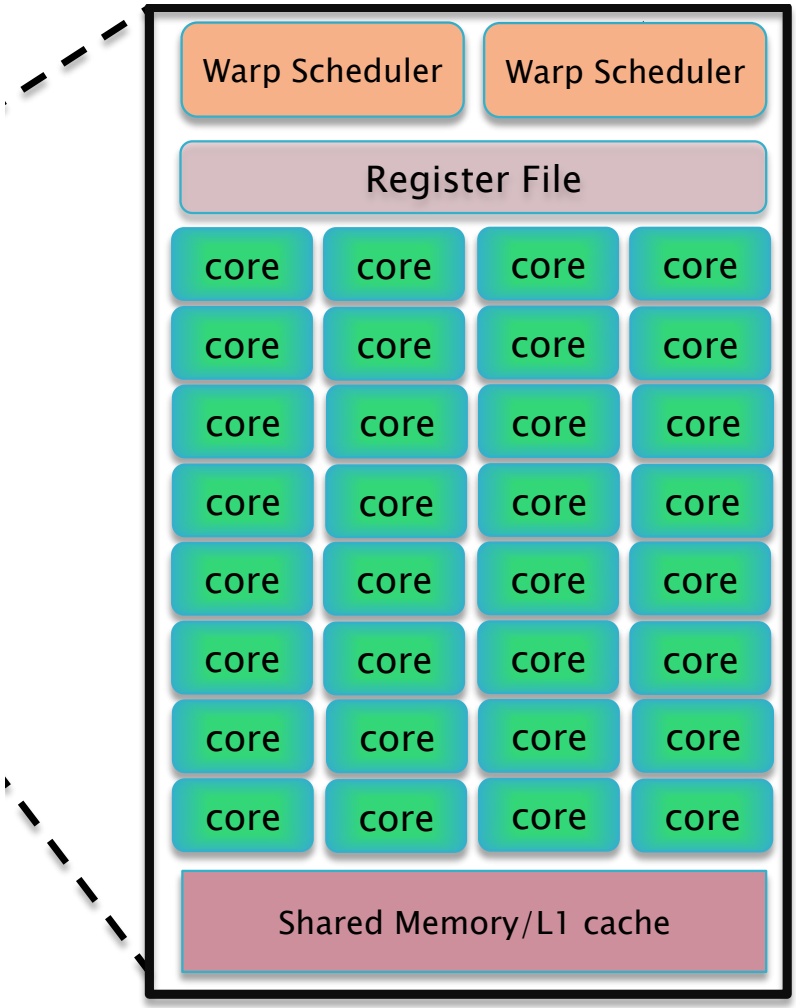
GPU Architecture



- A GPU device has a number of streaming multiprocessor (~10s)
- A streaming multiprocessor contains a large number of lightweight CUDA cores

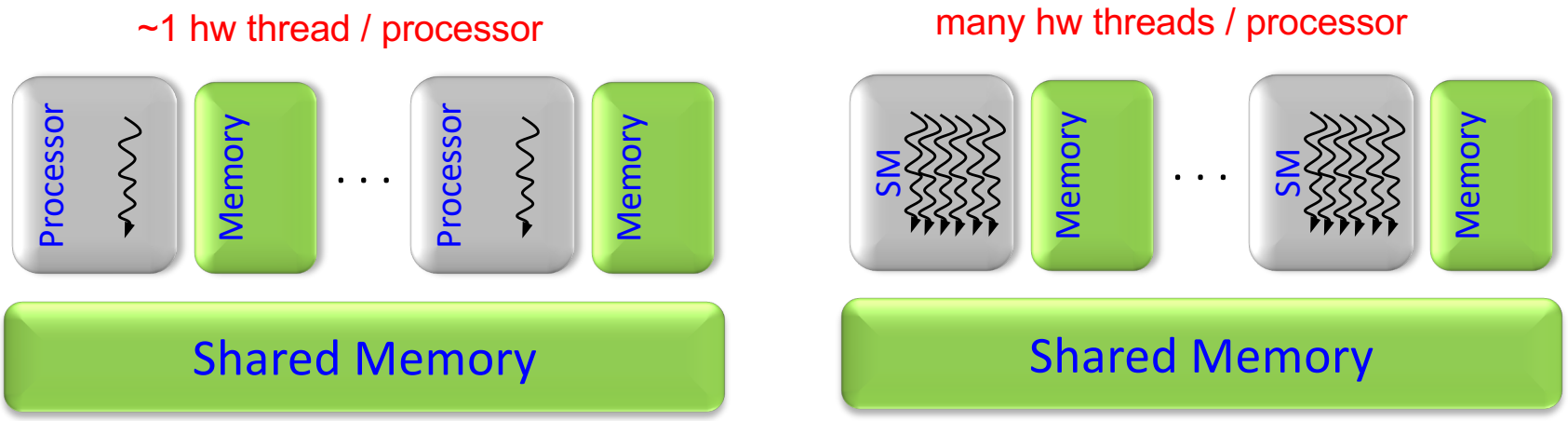
GPU Architecture

- A streaming multiprocessor (SM)
  - contains a large number of lightweight cores (CUDA cores)
- All the cores in an SM
  - Accesses to a shared memory and L1 cache
  - Shares shared memory (a.k.a) local storage which is a software managed cache!
  - There is a large register file



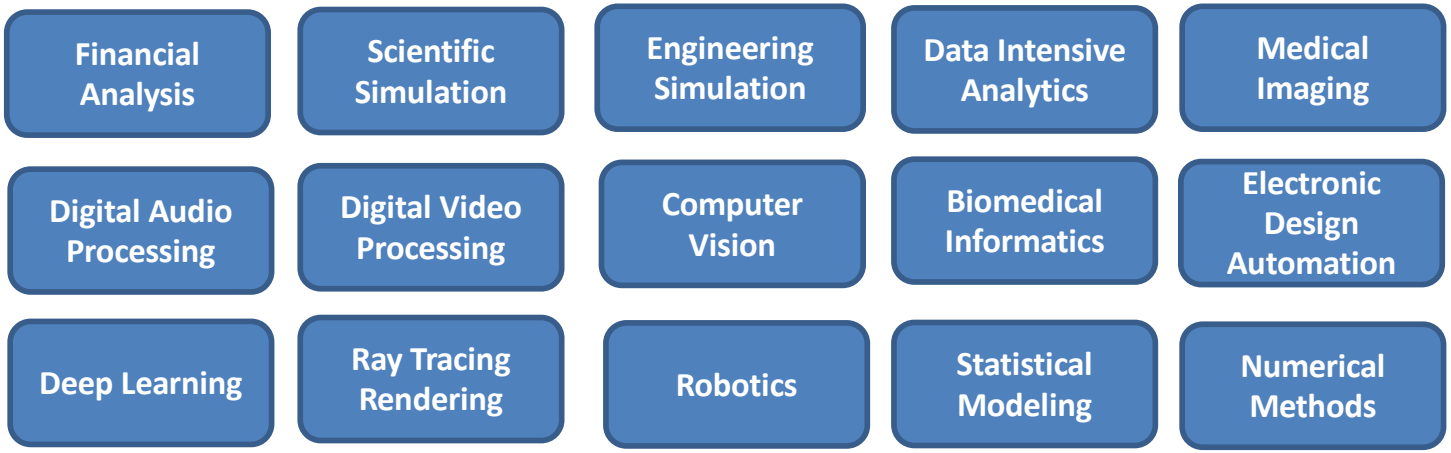
Execution Model

- SIMT (Single Instruction Multiple Thread) execution
  - Threads run in groups of 32 called warps
  - Threads in a warp share control logic – e.g. the current instruction
  - However, threads are allowed to diverge – automatically handled by HW
- Hardware (HW) multithreading
  - HW manages resource allocation & thread scheduling (not OS)
  - HW relies on threads to hide latency
  - Context switching is (basically) free

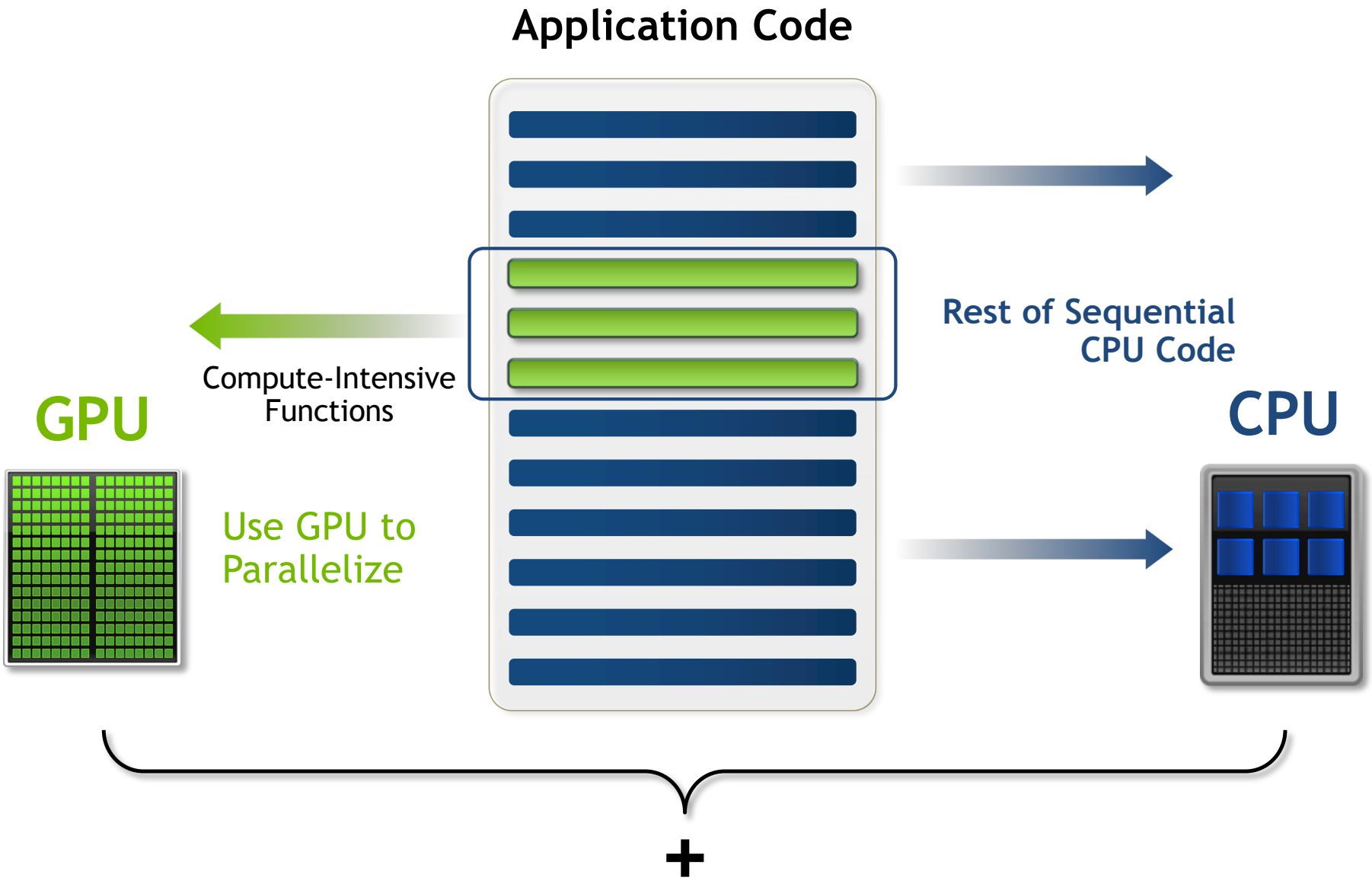


Applications

- CPUs for sequential parts 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 code



Heterogeneous Programming



GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

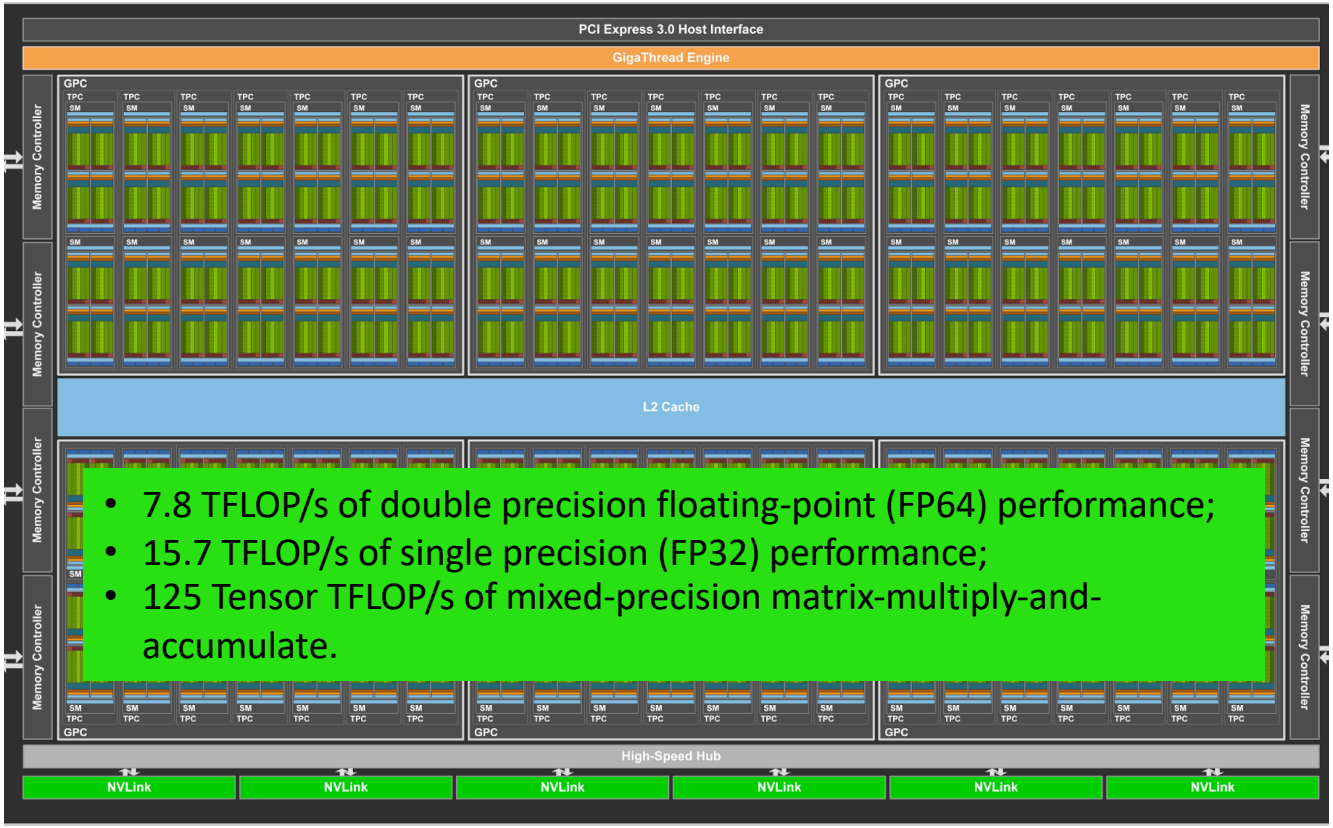
CUDA-Enabled NVIDIA GPUs

NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
				

Img. Src Nvidia

Nvidia Volta - Tesla V100 Specs

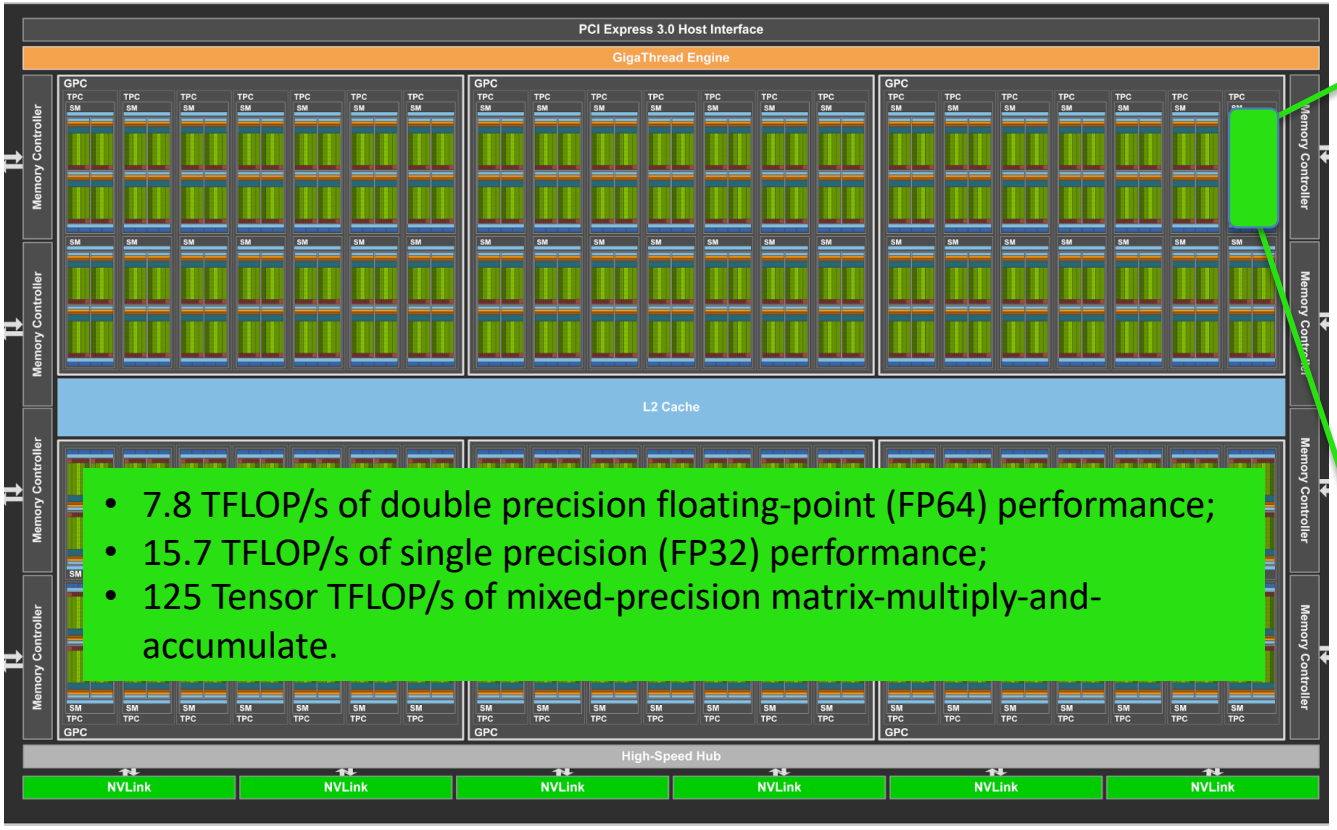
- Tesla V100 is engineered for the convergence of HPC and AI.
- Supports up to 6 NVLink links at 25 GB/s for a total of 300 GB/s.
- 32GB HBM2 memory with 900 GB/sec peak memory bandwidth
- Includes 21.1 billion transistors, 300 Watts
- Clock Frequency of 1.5 GHz



Img. Src Nvidia

Nvidia Volta - Tesla V100 Specs

- Volta may have up to 163,840 concurrent threads
- 80-84 Volta SMs, each has
  - L2 Cache Size 6 MB
  - Shared Memory Size configurable up to 96 KB
  - L1 Cache Size 128 KB
  - Register File Size / SM 256 KB



SM

Img. Src Nvidia

Device Query

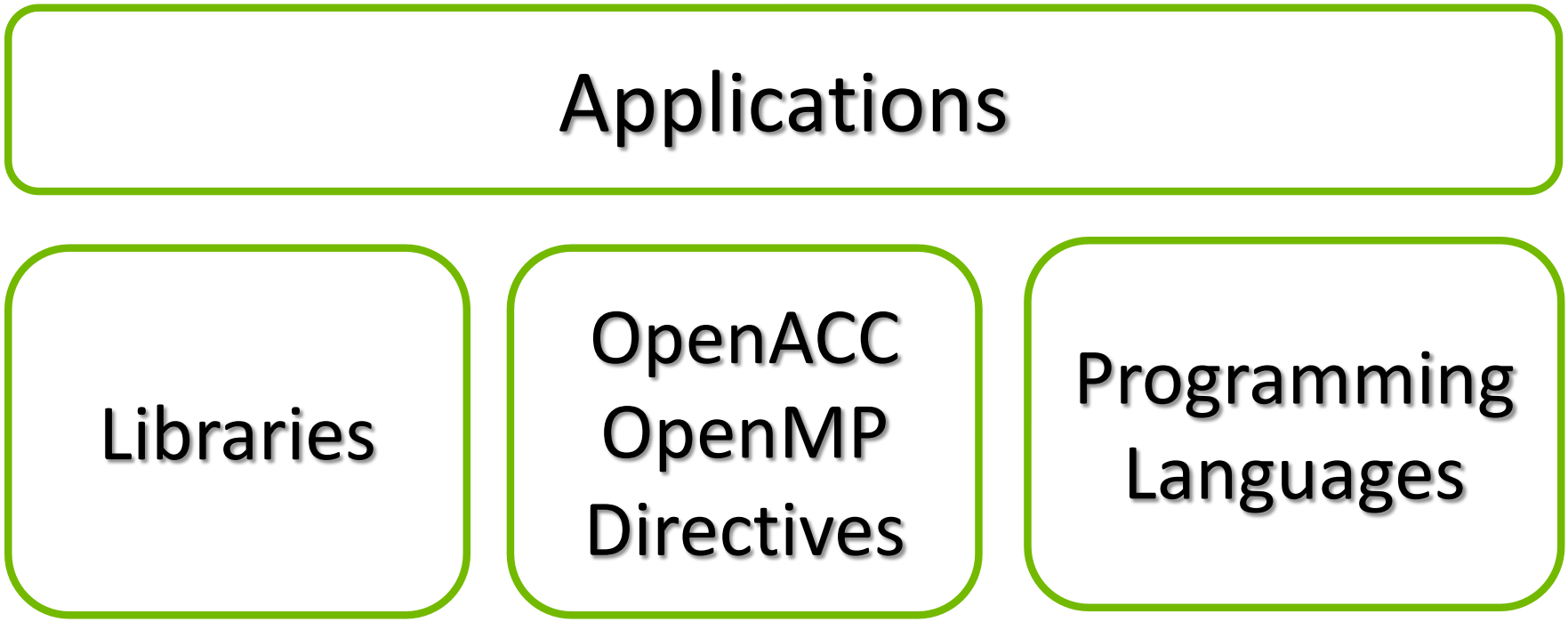
```
[dunat@it01 1_deviceQuery]$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla V100-PCIE-32GB"
  CUDA Driver Version / Runtime Version          9.1 / 9.1
  CUDA Capability Major/Minor version number:    7.0
  Total amount of global memory:                 32510 MBytes (34089730048 bytes)
  (80) Multiprocessors, ( 64) CUDA Cores/MP:     5120 CUDA Cores
  GPU Max Clock rate:                            1380 MHz (1.38 GHz)
  Memory Clock rate:                             877 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 6291456 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 7 copy engine(s)
```

3 Ways to Accelerate Applications

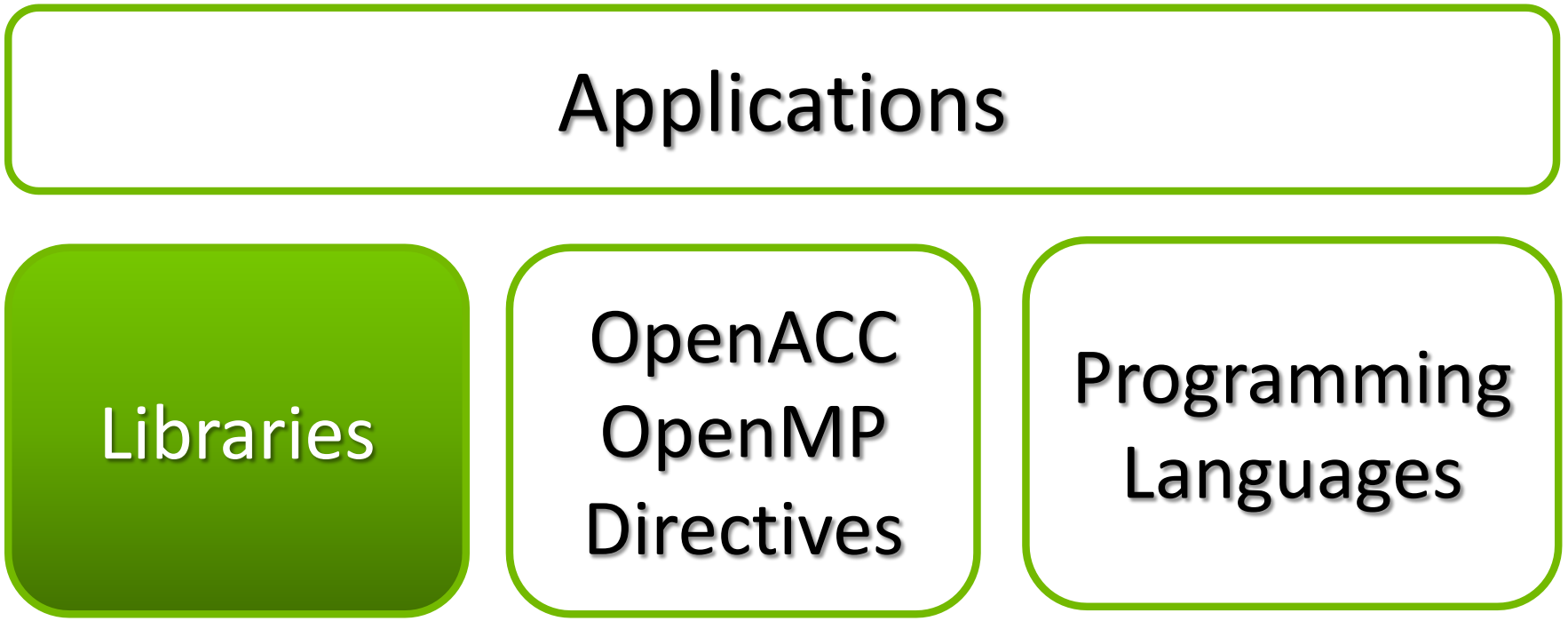


"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Flexibility

3 Ways to Accelerate Applications



"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Flexibility

Libraries

Advantages:

- Ease of use: Using libraries enables GPU acceleration without in-depth knowledge of GPU programming

- "Drop-in": Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes

- Performance: NVIDIA libraries are tuned by experts

Disadvantages:

- Limitation: You are restricted to what library can offer.

GPU-accelerated Libraries

Signal, Image and Video Libraries

cuFFT

GPU-accelerated library for Fast Fourier Transforms

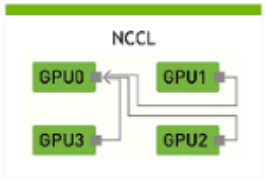
NVIDIA Performance Primitives

GPU-accelerated library for image and signal processing

NVIDIA Codec SDK

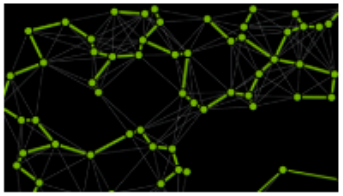
High-performance APIs and tools for hardware accelerated video encode and decode

Parallel Algorithm Libraries



NCCL

Collective Communications Library for scaling apps across multiple GPUs and nodes



nvGRAPH

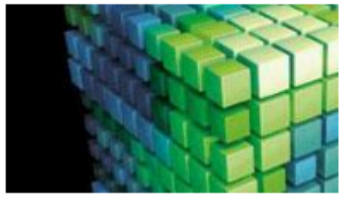
GPU-accelerated library for graph analytics

Thrust

GPU-accelerated library of parallel algorithms and data structures

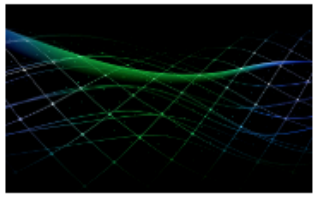
GPU-accelerated Libraries

Linear Algebra and Math Libraries



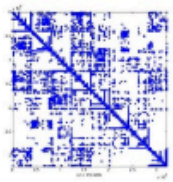
cuBLAS

GPU-accelerated standard BLAS library



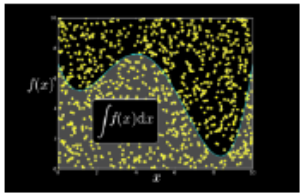
CUDA Math Library

GPU-accelerated standard mathematical function library



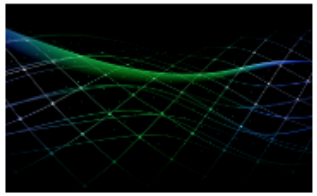
cuSPARSE

GPU-accelerated BLAS for sparse matrices



cuRAND

GPU-accelerated random number generation (RNG)



cuSOLVER

Dense and sparse direct solvers for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications

AmgX

GPU accelerated linear solvers for simulations and implicit unstructured methods

- Full list is available at
  - https://developer.nvidia.com/gpu-accelerated-libraries

Using GPU-accelerated Libraries

- **Step 1:** Substitute library calls with equivalent CUDA library calls

  `saxpy ( … )` `cublasSaxpy ( … )`

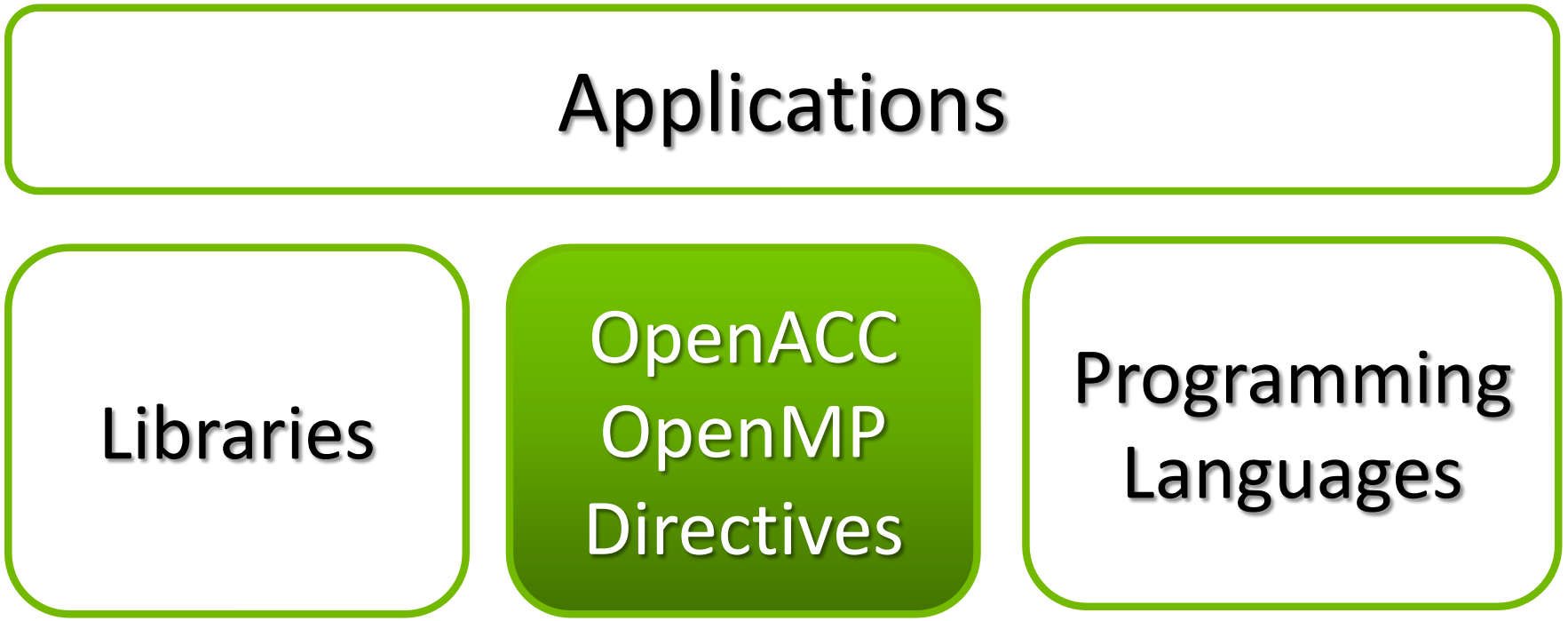
- **Step 2:** Manage data locality

  - with CUDA: `cudaMalloc(), cudaMemcpy(), etc.`
  - with CUBLAS: `cublasAlloc(), cublasSetVector(), etc.`

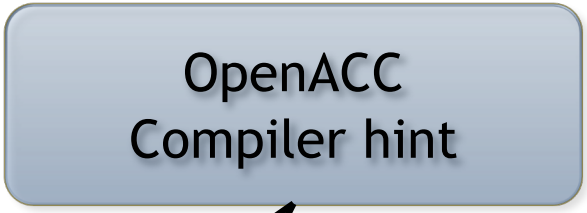
- **Step 3:** Rebuild and link the CUDA-accelerated library

```
nvcc myobj.o –l cublas
```

3 Ways to Accelerate Applications



OpenACC Directives




```
Program myscience
   ... serial code ...
#pragma acc kernels

   ... Parallel code ...

#pragma end kernels
   ...
End Program myscience
```

Your original Fortran or C code

Simple compiler hints

Compiler parallelizes code

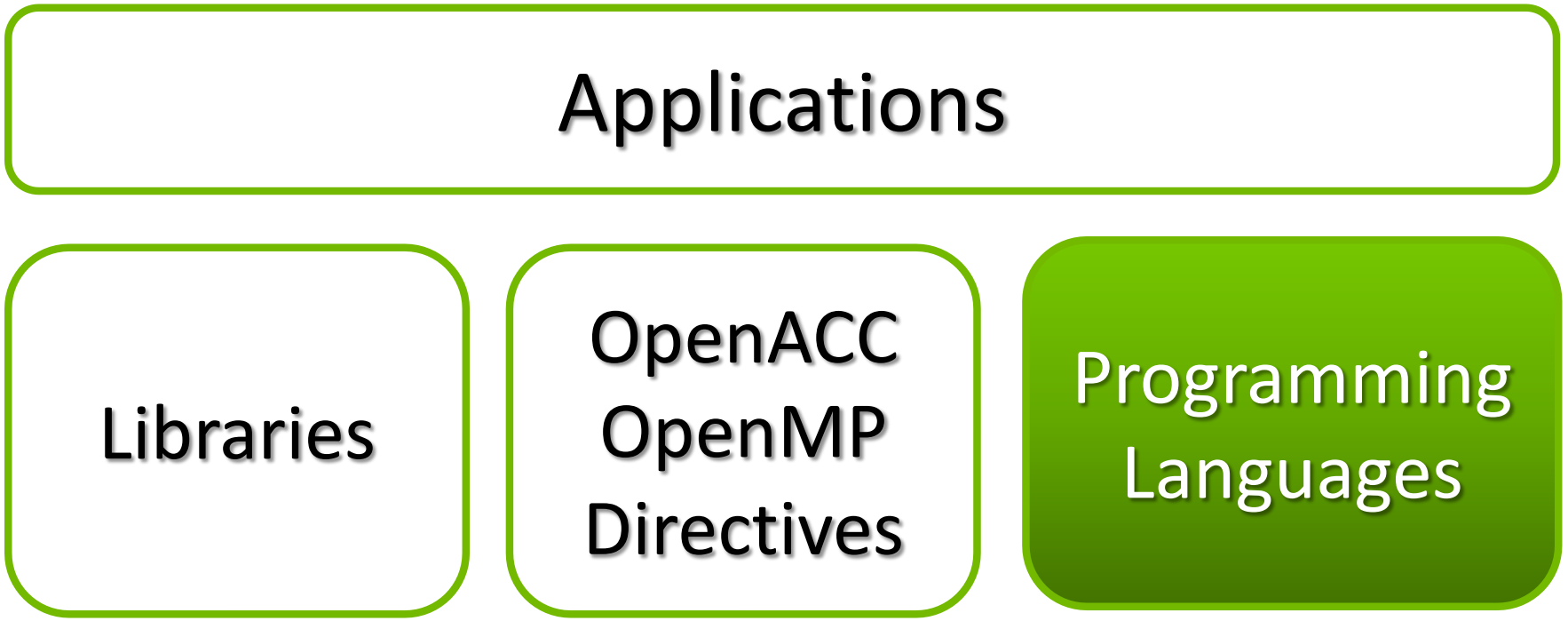
Works on many-core GPUs & multicore CPUs

OpenACC Directives

- **Advantages:**
  - **Easy:** Directives are the easy path to accelerate compute intensive applications
  - **Powerful:** GPU Directives allow complete access to the massive parallel power of a GPU

- **Disadvantages:**
  - **Performance**: can be suboptimal, may not work well for all loop types
  - **Learning:** Still need to learn a lot of pragmas.
  - **Compiler:** Need to pay for the compiler license – free trials

3 Ways to Accelerate Applications



GPU Programming Languages

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	CUDA Fortran
C	**CUDA C, OpenCL, SyCL, HiPS**
C++	Thrust, CUDA C++, SyCL, HiPS
Python	PyCUDA, Copperhead
F#	Alea.cuBase

CUDA C

- **Advantages:**
  - **Flexible:** Most flexible way to program GPUs
  - **Performance:** Gives full control over the hardware to get best performance

- **Disadvantages:**
  - **Difficult:** Can be difficult to learn for non-experts, may require hardware and parallel programming knowledge

Acknowledgments

- These slides are inspired and partly adapted from
  - Mary Hall (Univ. of Utah)
  - Programming Massively Parallel Processors: A Hands On Approach, available from *http: David Kirk and Wen-mei Hwu, February 2010, Morgan Kaufmann Publishers, ISBN 0123814723.*
  - NVidia, *CUDA Programming Guide,* available from *http://docs.nvidia.com/cuda/cuda-c-programming-guide*
  - Why GPU Computing? By Mark Ebersole – Nvidia
    - https://developer.nvidia.com/cuda-education
  - CS193g Spring 2010 GPU Computing Course at Stanford
    - https://github.com/jaredhoberock/stanford-cs193g-sp2010