



GPGPUs and CUDA

Alan Sussman, Department of Computer Science



UNIVERSITY OF
MARYLAND

Announcements

- Assignment 1 (MPI) was due today at 11:59 pm
 - We are waiving the penalty for late submissions, so you have until Wed. at 11:59PM to get full credit
 - Questions?
- Assignment 2 will be released on Wed.
 - Will discuss in class on Thursday

GPGPUs

- Originally developed to handle computation related to graphics processing
- Also found to be useful for scientific computing
- Hence the name: **General Purpose Graphics Processing Unit**

Accelerators

- IBM's Cell processors
 - Used in Sony's Playstation 3 (2006)
- GPUs: NVIDIA, AMD, Intel
 - First programmable GPU: GeForce 256 (1999)
 - Around 1999-2001, early GPGPU results
- FPGAs

<https://www.cs.unc.edu/xcms/wpfiles/50th-symp/Harris.pdf>

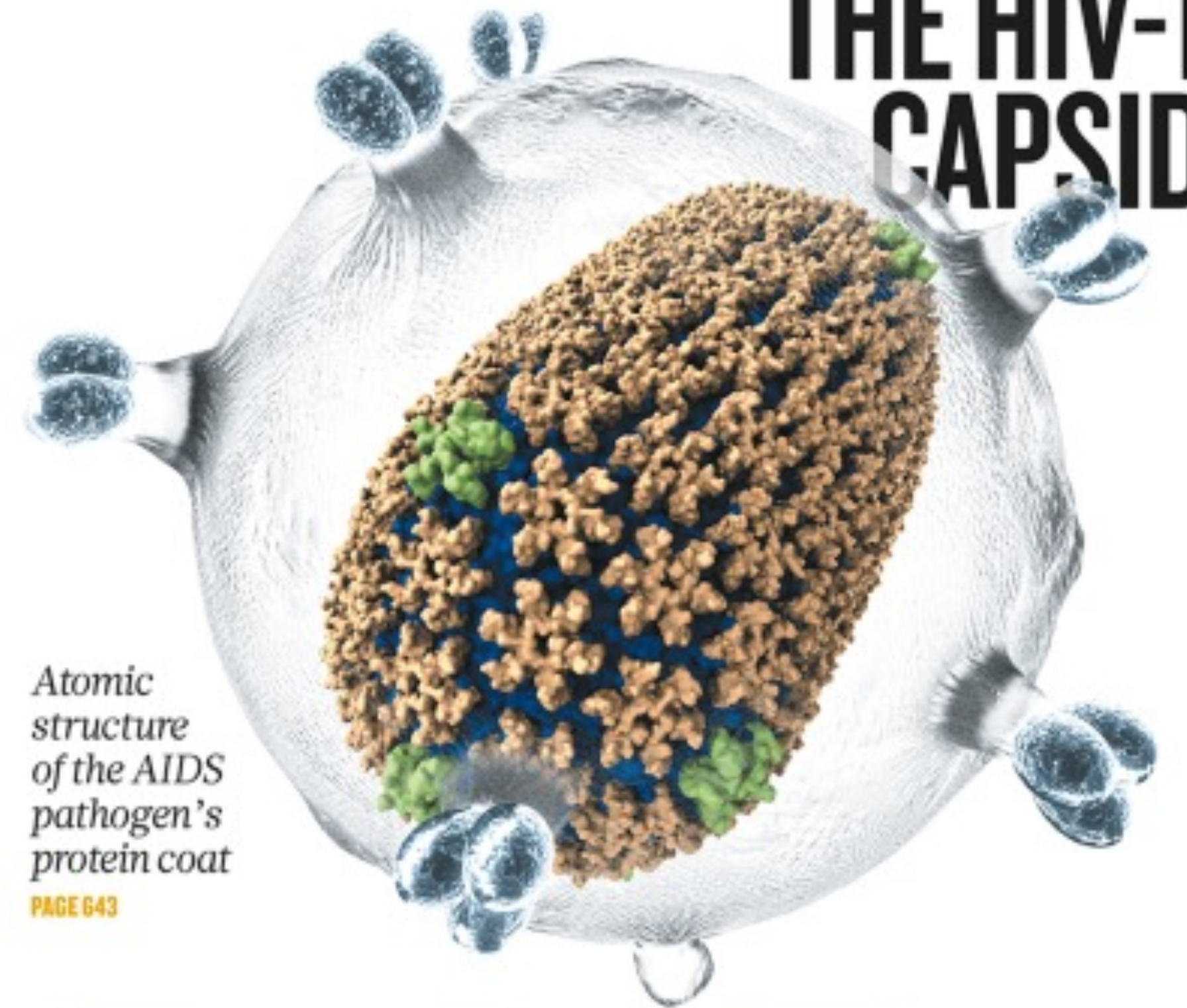
Used for mainstream HPC

- 2013: NAMD used for molecular dynamics simulations on a supercomputer with 3000 NVIDIA Tesla GPUs

nature

THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE

THE HIV-1 CAPSID



Atomic structure of the AIDS pathogen's protein coat

PAGE 643

COSMOLOGY

THE FIRST LIGHT

In pursuit of the most distant galaxies

PAGE 554

CITATION

CROSSING THE BORDERS

International collaborations make the most impact

PAGE 557

ANTICANCER DRUGS

A SITTING TARGET

An indirect hit on "undruggable" KRAS protein

PAGES 577 & 638

NATURE.COM/NATURE

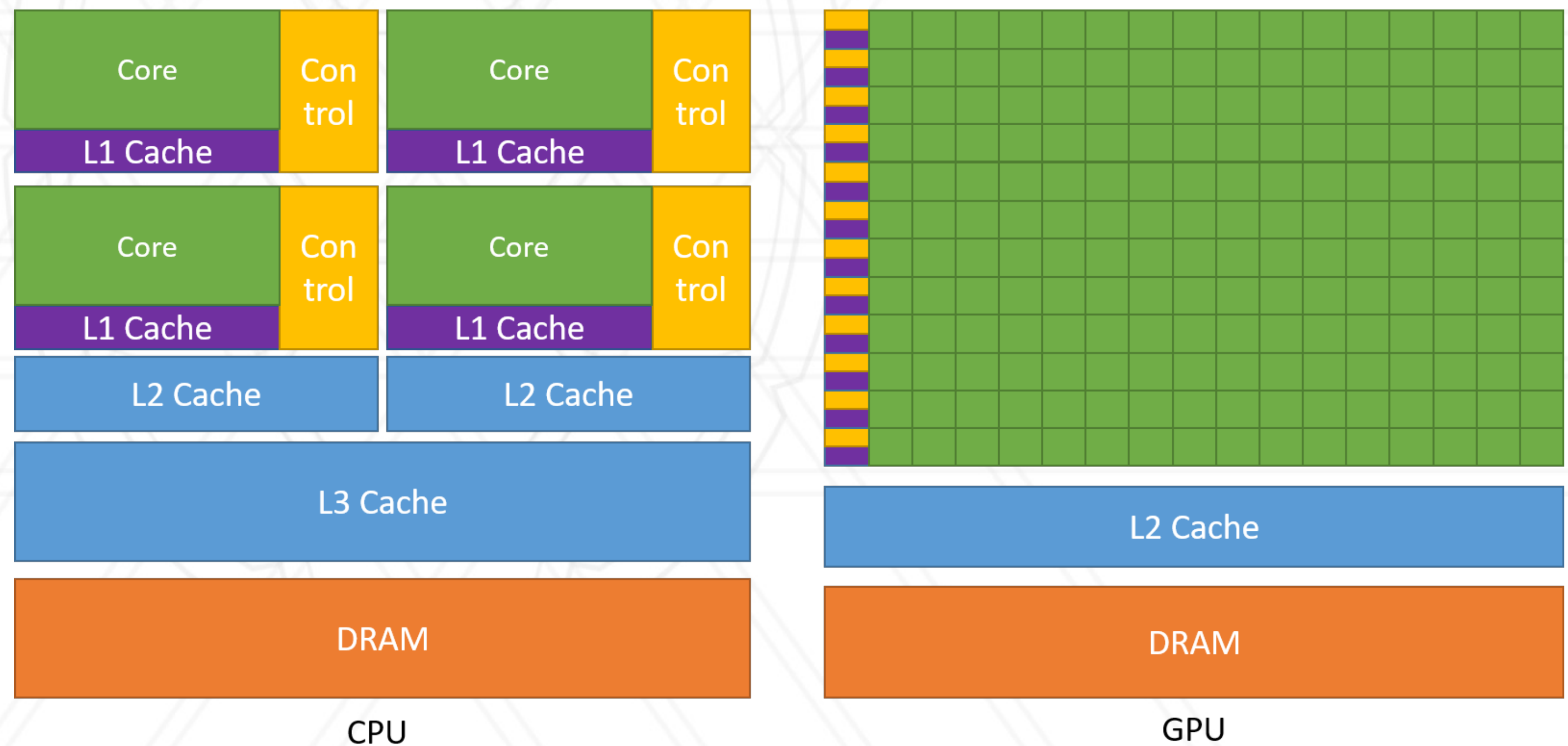
30 May 2013

\$10.00 US \$12.99 CAN 22p



GPGPU Hardware

- Higher instruction throughput
- Hide memory access latencies with computation



Comparing GPUs to CPUs

- Intel i9 11900K

- 8 cores
- 3.3 GHz

- AMD Epyc 7763

- 64 cores
- 2.45 GHz

- NVIDIA GeForce RTX 3090

- 10,496 cores
- 1.4 GHz

- NVIDIA A100

- 17,712 cores
- 0.76 GHz

Volta GV100 SM

- CUDA Core
 - Single serial execution unit
- Each Volta Streaming Multiprocessor (SM) has:
 - 64 FP32 cores
 - 64 INT32 cores
 - 32 FP64 cores
 - 8 Tensor cores
- CUDA capable device or GPU
 - Collection of SMs

<https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>

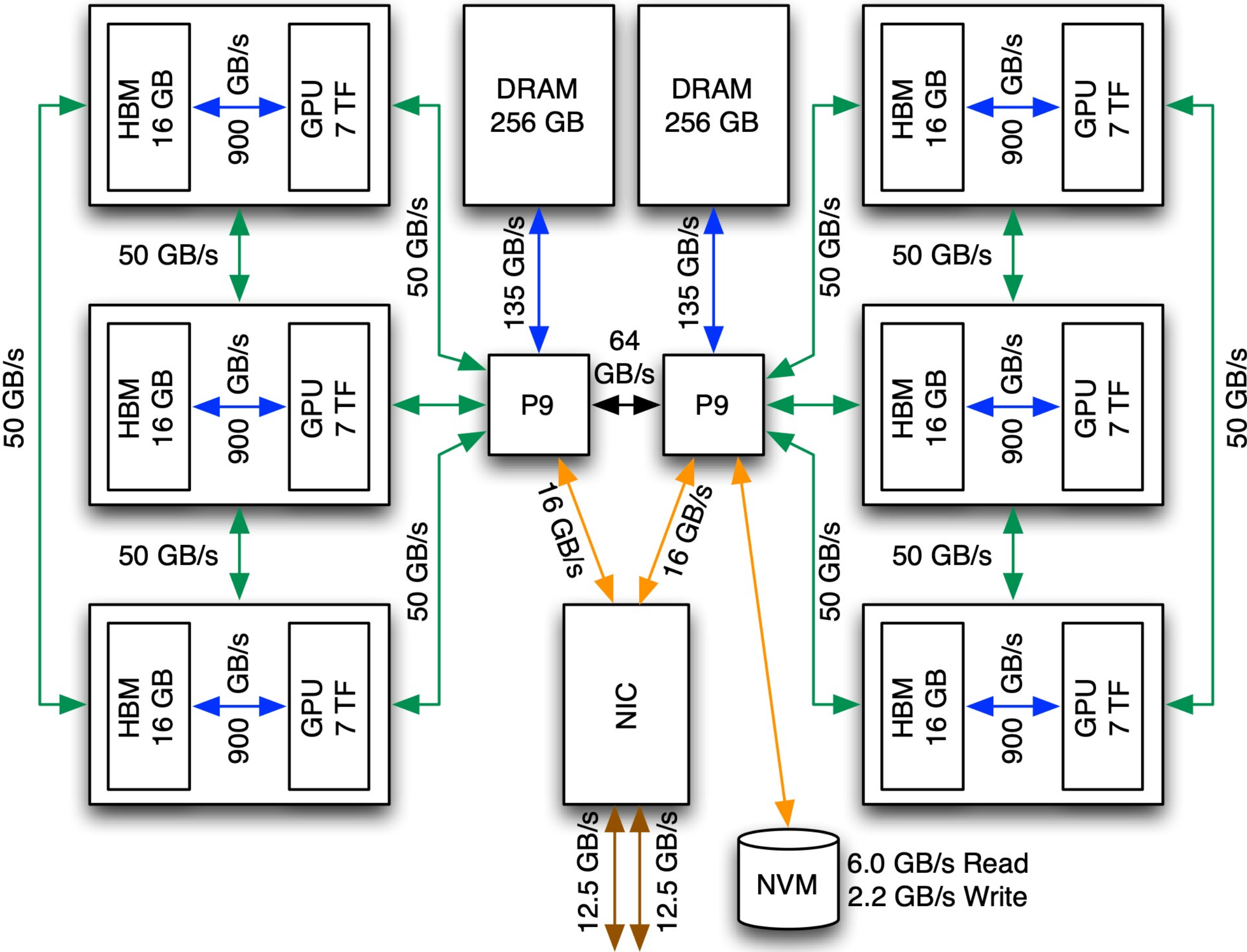


Vo



GPU-based nodes

- Figure on the right shows a single node of Summit @ ORNL

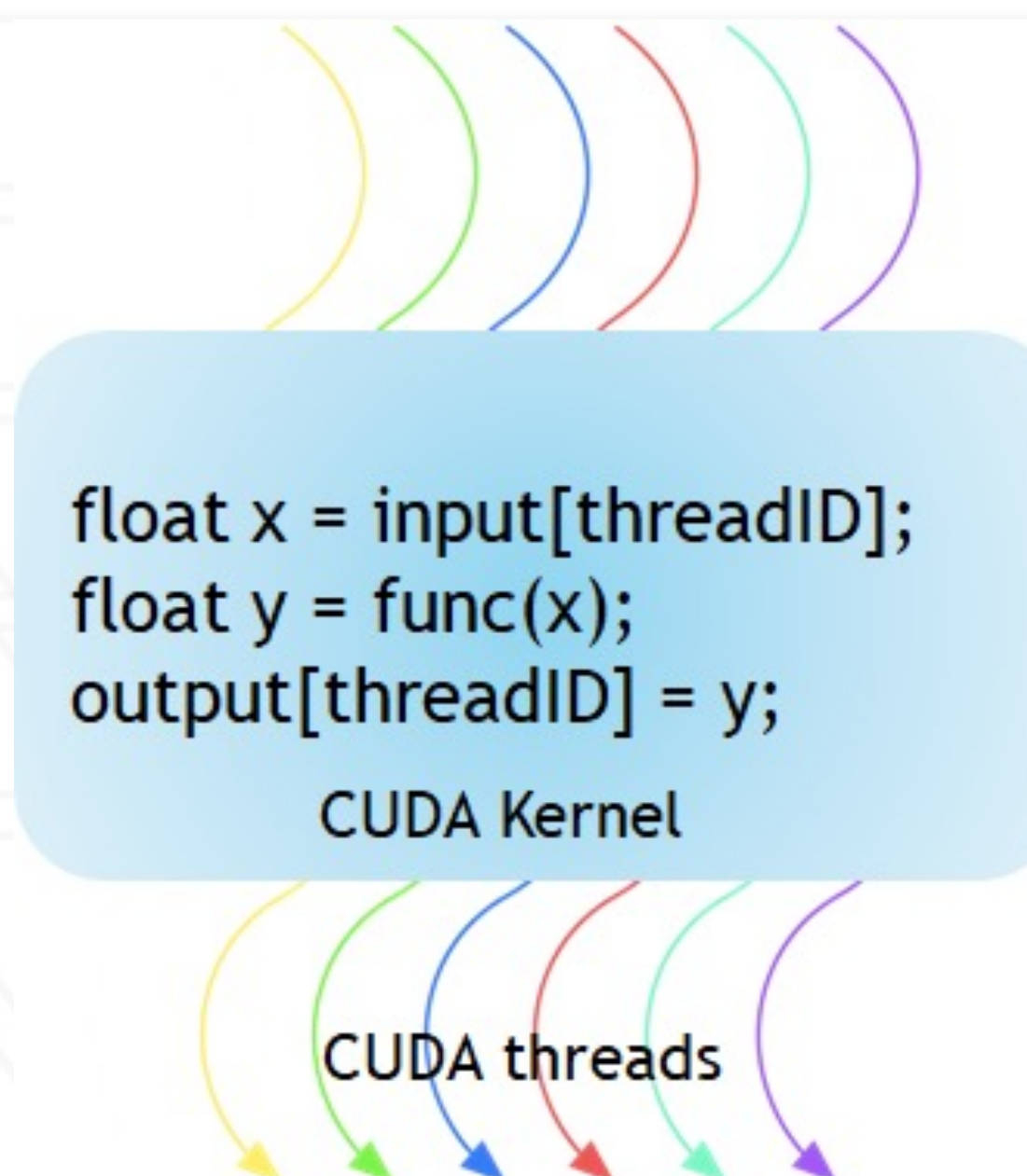


TF	42 TF (6x7 TF)		HBM/DRAM Bus (aggregate B/W)
HBM	96 GB (6x16 GB)		NVLINK
DRAM	512 GB (2x16x16 GB)		X-Bus (SMP)
NET	25 GB/s (2x12.5 GB/s)		PCIe Gen4
MMsg/s	83		EDR IB

HBM & DRAM speeds are aggregate (Read+Write).
 All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

CUDA: A programming model for NVIDIA GPUs

- Allows developers to use C++ as a high-level programming language
- Built around threads, blocks and grids
- Terminology:
 - Host: CPU
 - Device: GPU
 - CUDA kernel: a function that gets executed on the GPU



CUDA software abstraction

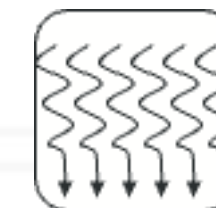
- Thread

- Serial unit of execution



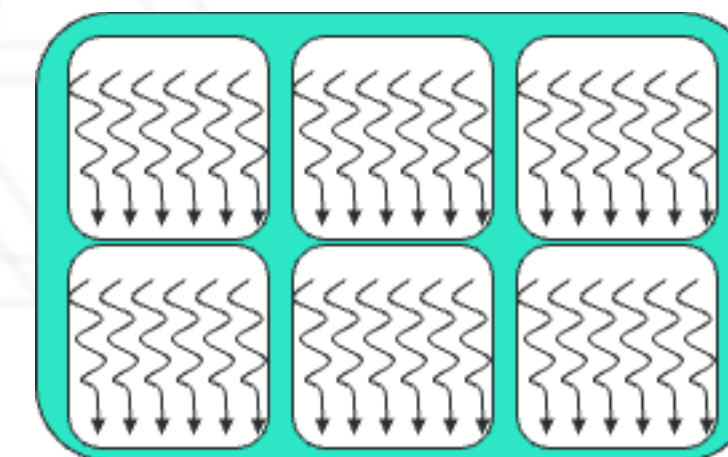
- Block

- Collection of threads
- Number of threads in block ≤ 1024

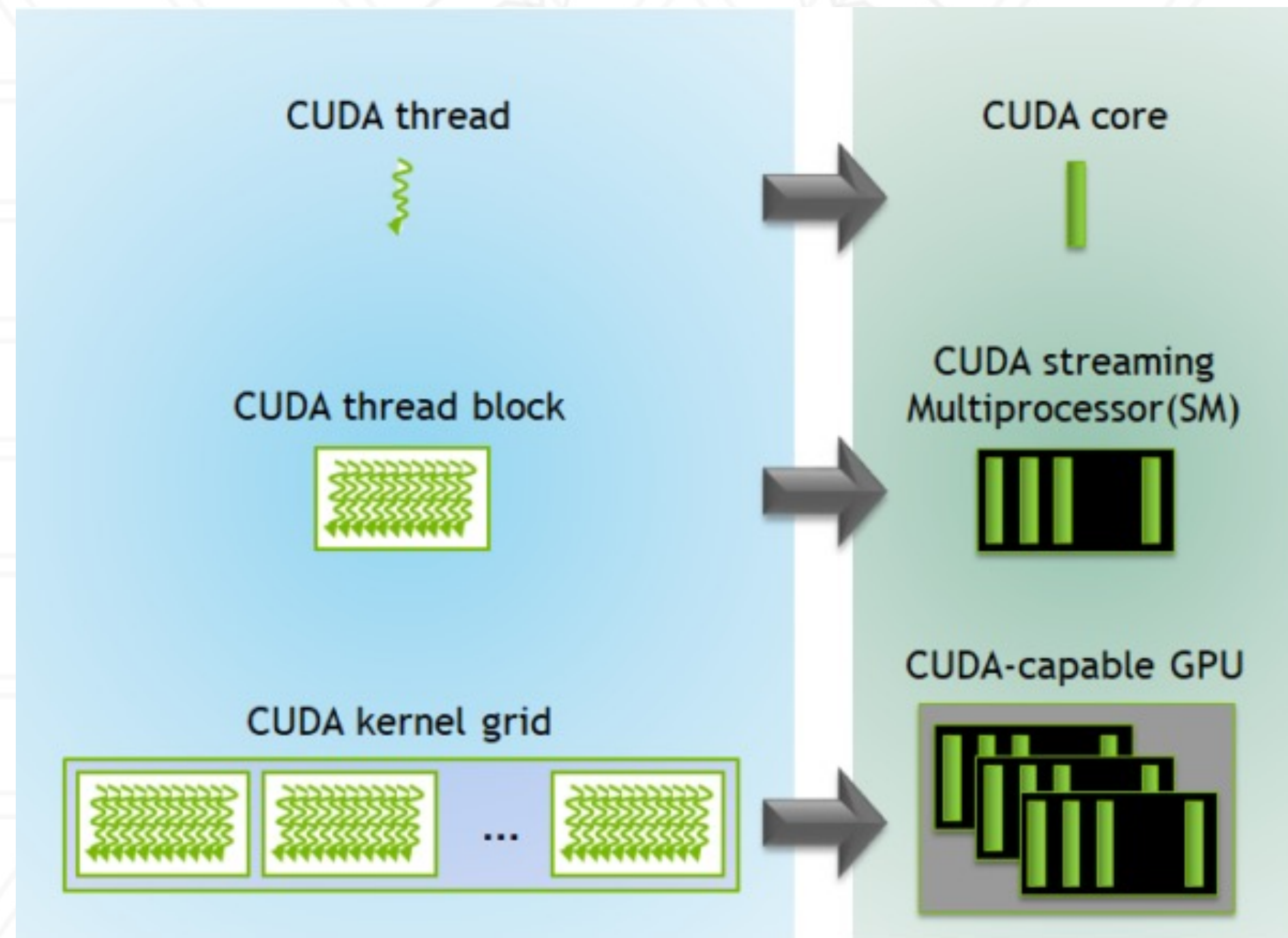


- Grid

- Collection of blocks



Software to hardware mapping



<https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/>

Three steps to writing a CUDA kernel

- Copy input data from host to device memory
- Load the GPU program (kernel) and execute
- Copy the results back to host memory

Copying data to the GPU

```
double *d_Matrix, *h_Matrix;
h_Matrix = new double[N];

cudaMalloc(&d_Matrix, sizeof(double)*N);

// ... initialize h_Matrix ...
cudaMemcpy(d_Matrix, h_Matrix, sizeof(double)*N, cudaMemcpyHostToDevice);

// ... some computation on GPU ...

cudaMemcpy(h_Matrix, d_Matrix, sizeof(double)*N, cudaMemcpyDeviceToHost);

cudaFree(d_Matrix);
```

cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice
cudaMemcpyHostToHost
cudaMemcpyDefault

CUDA syntax

```
__global__ void saxpy(float *x, float *y, float alpha) {  
    int i = threadIdx.x;  
    y[i] = alpha*x[i] + y[i];  
}
```

```
int main() {  
    ...  
    saxpy<<<1, N>>>(x, y, alpha);  
    ...  
}
```

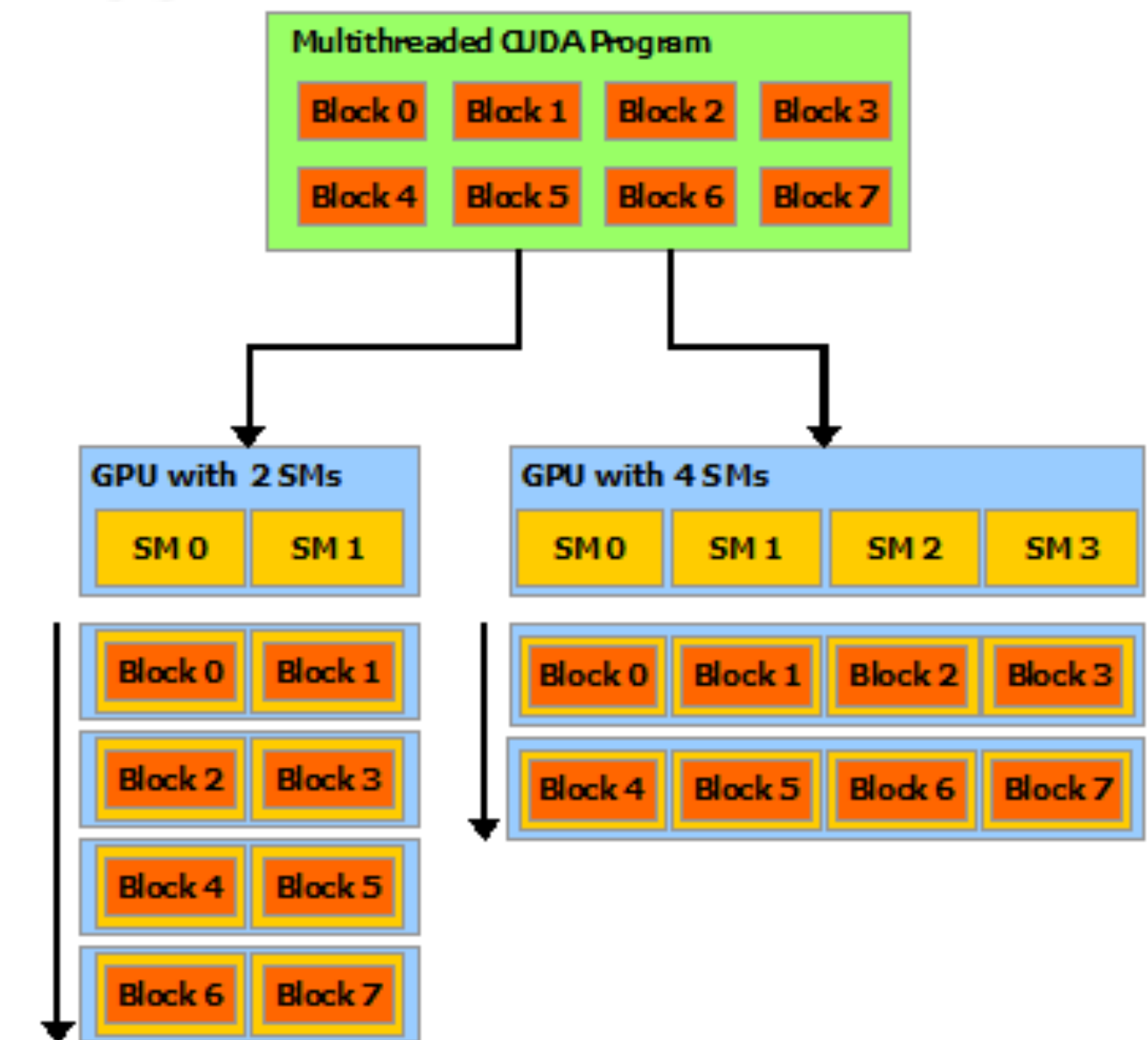
What happens when:
N > 1024
or N > #device threads

`<<<#blocks, threads_per_block>>>`

Multiple blocks

```
__global__ void saxpy(float *x, float *y, float alpha, int N) {  
    int i = blockDim.x * blockIdx.x + threadIdx.x;  
    y[i] = alpha*x[i] + y[i];  
}
```

```
int main() {  
    ...  
    int threadsPerBlock = 512;  
    int numBlocks = N/threadsPerBlock  
        + (N % threadsPerBlock != 0);  
  
    saxpy<<<numBlocks, threadsPerBlock>>>(x, y, alpha, N);  
    ...  
}
```



Questions?



UNIVERSITY OF
MARYLAND