

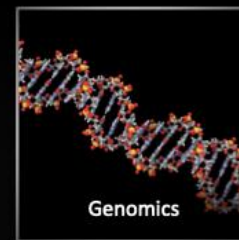
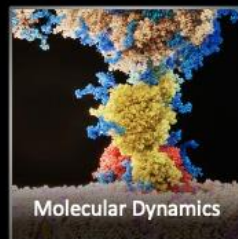
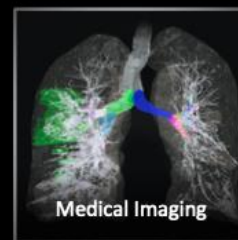
Introduction to GPU Computing

N-ways bootcamp

Introduction to gpu computing

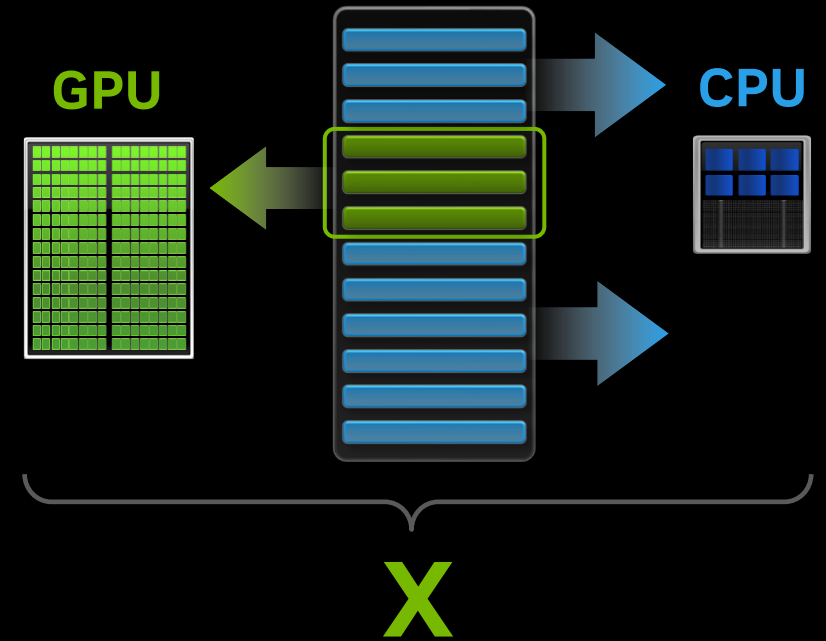
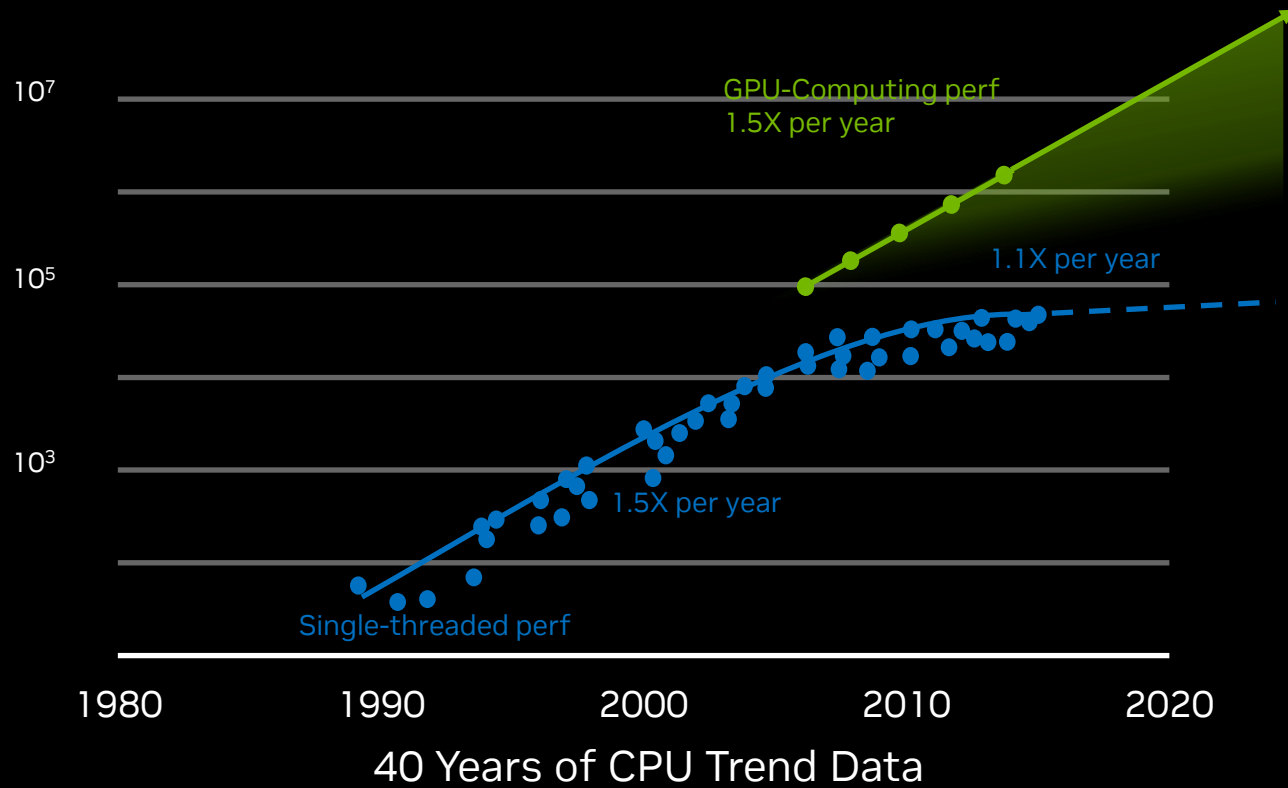
What to expect?

- Overview of GPU Architecture
- Approaches to getting started with GPUs in your work
- Intro to programming GPUs



Universe of GPU Computing

Rise of GPU Computing

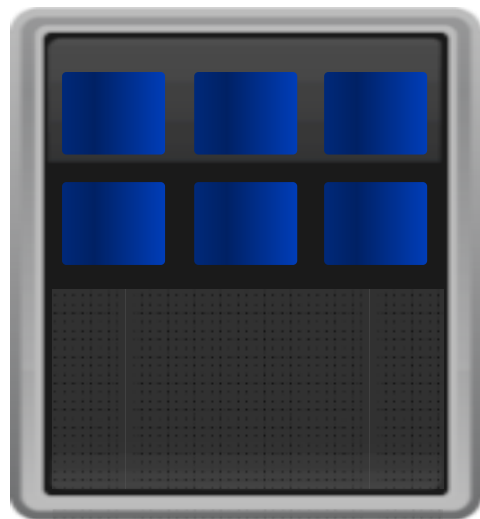


Original data up to the year 2010 collected and plotted by M. Horowitz,
F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

ACCELERATED COMPUTING

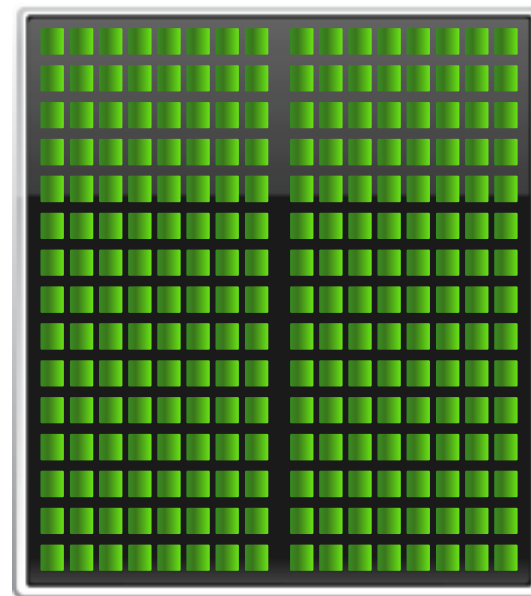
CPU

Optimized for
Serial Tasks



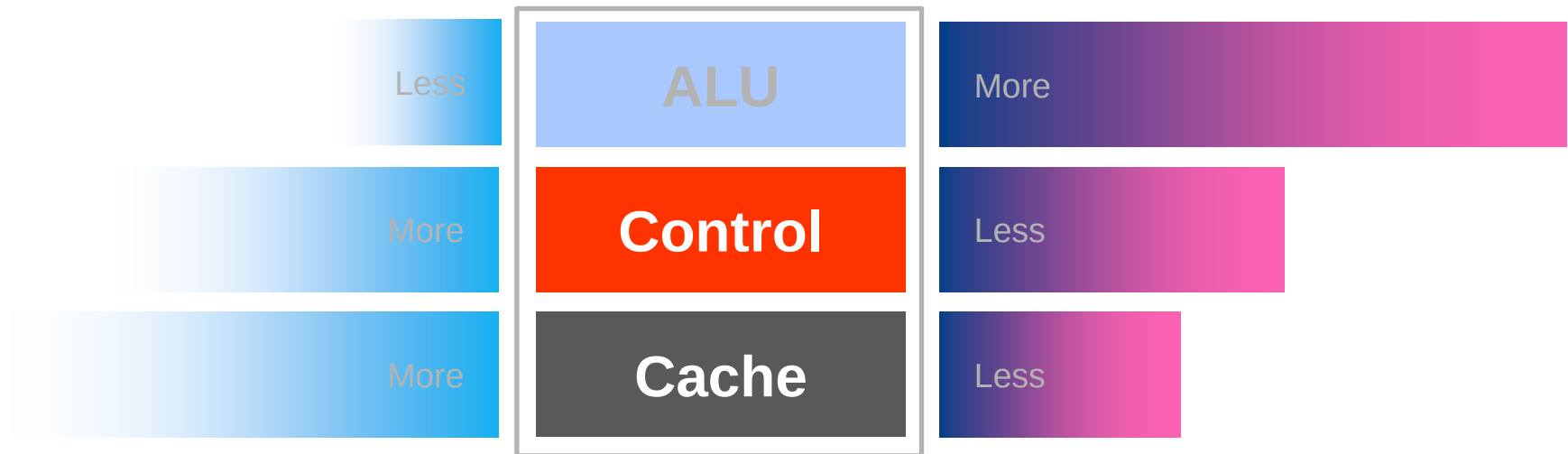
GPU Accelerator

Optimized for
Parallel Tasks



SILICON BUDGET

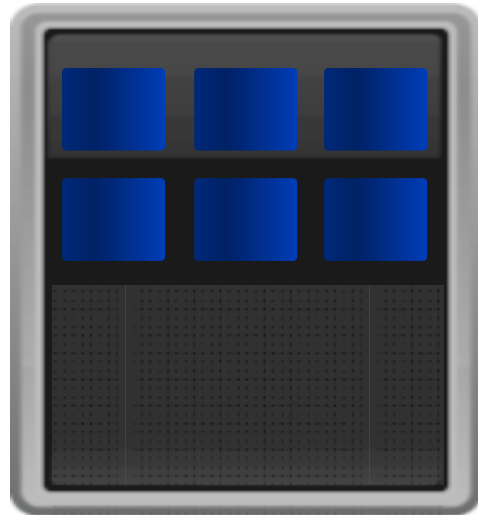
- The three components of any processor



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU

Optimized for
Serial Tasks



GPU Accelerator

CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

GPU IS ALL ABOUT HIDING LATENCY

GPU Strengths

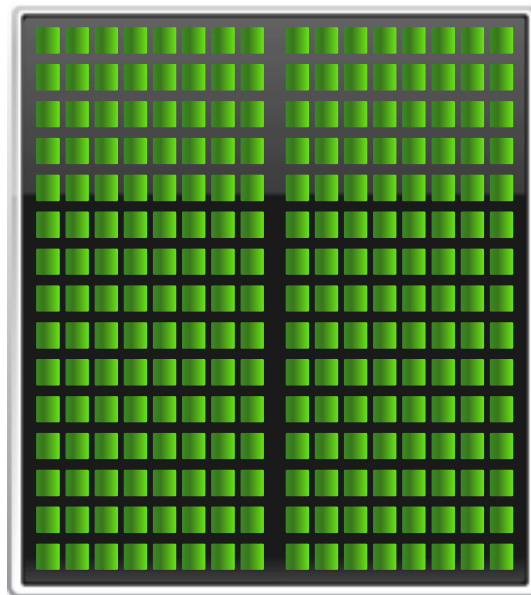
- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

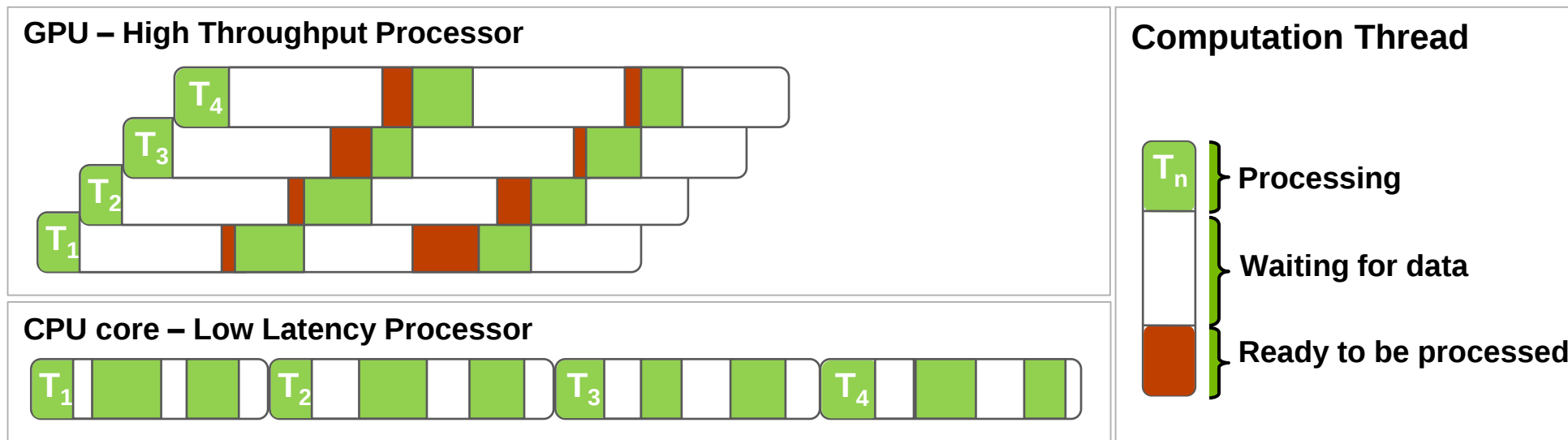
GPU Accelerator

Optimized for
Parallel Tasks



LOW LATENCY VS HIGH THROUGHPUT

- CPU architecture must **minimize latency** within each thread
- GPU architecture **hides latency** with computation (data-parallelism, to 30k threads!)



SPEED V. THROUGHPUT

Speed

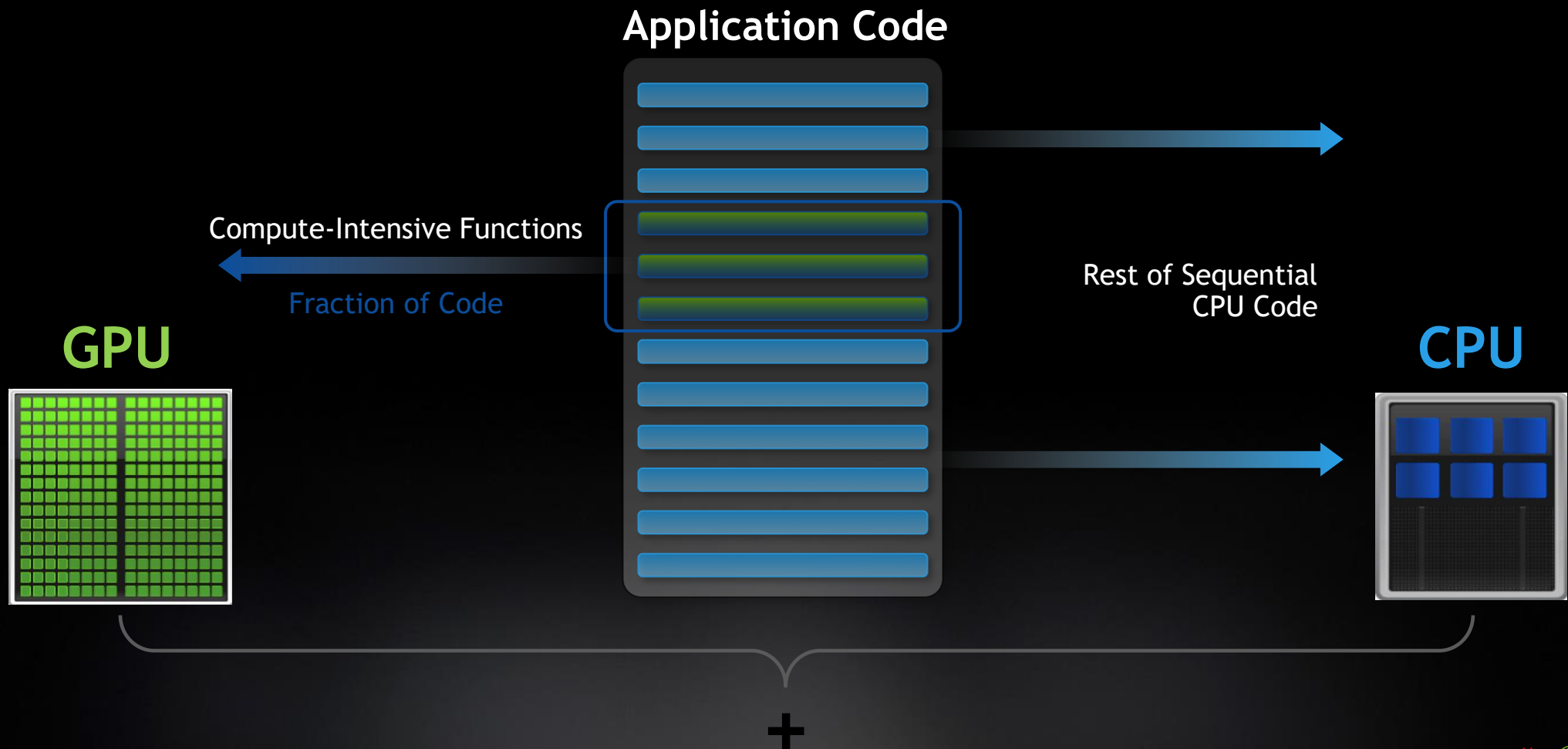


Throughput

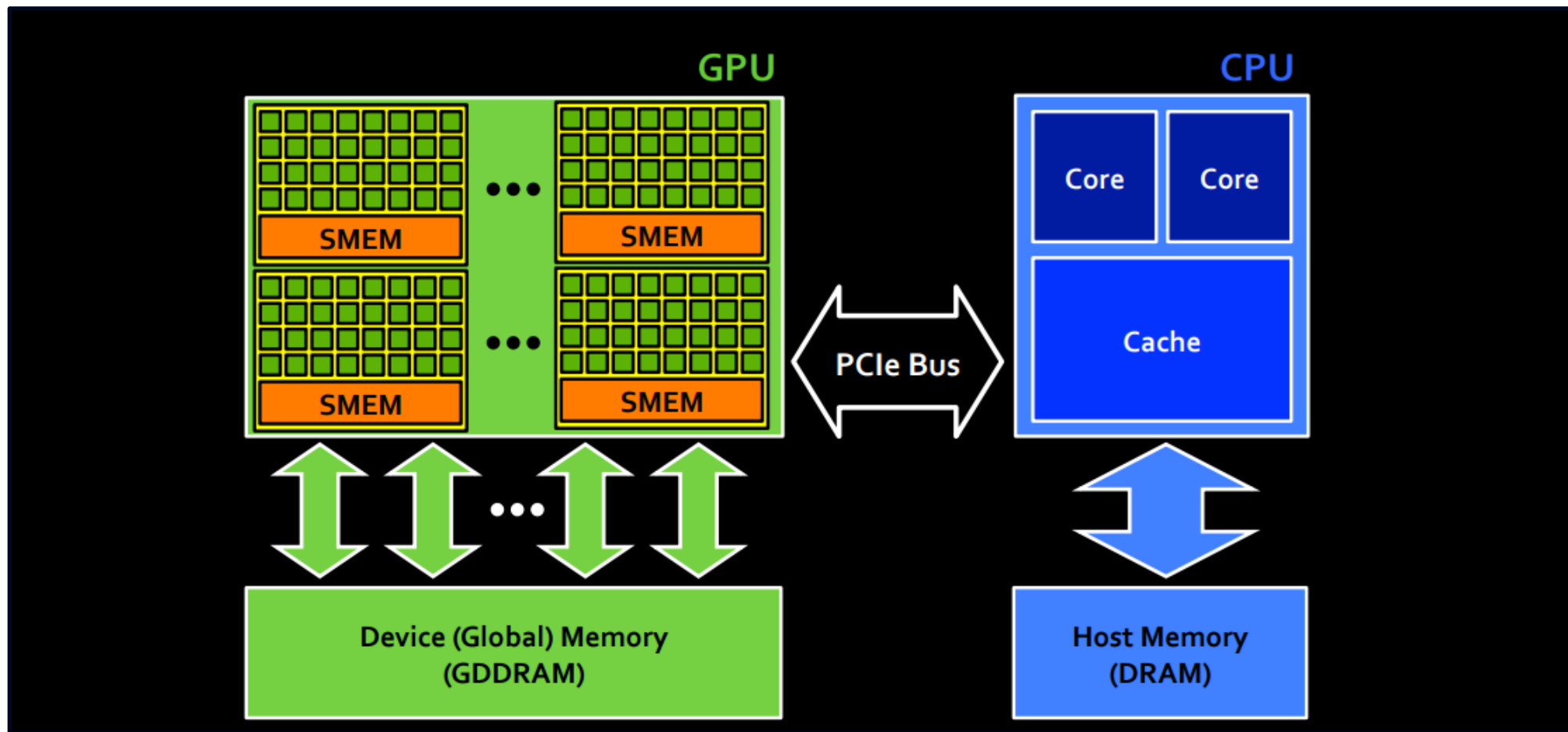


Which is better depends on your needs...

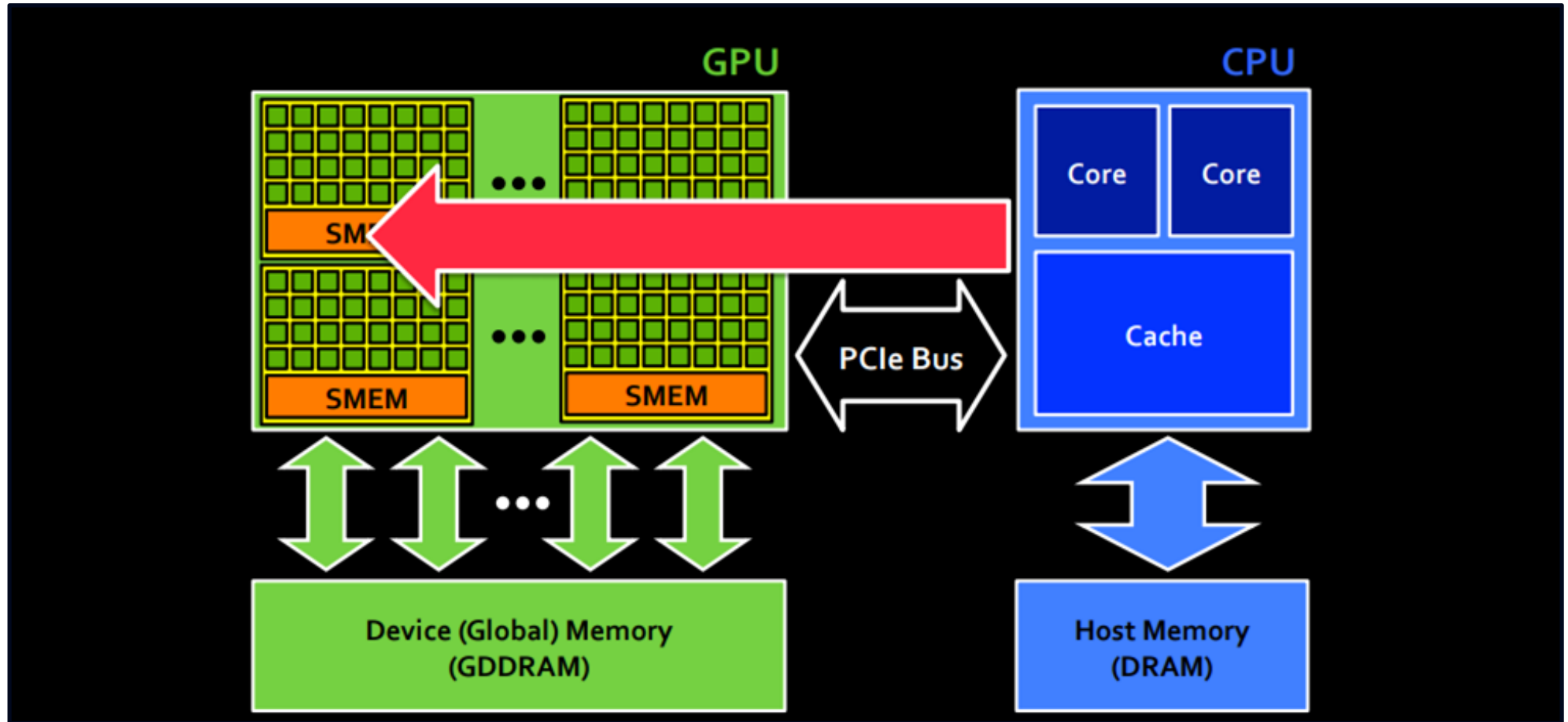
HOW GPU ACCELERATION WORKS



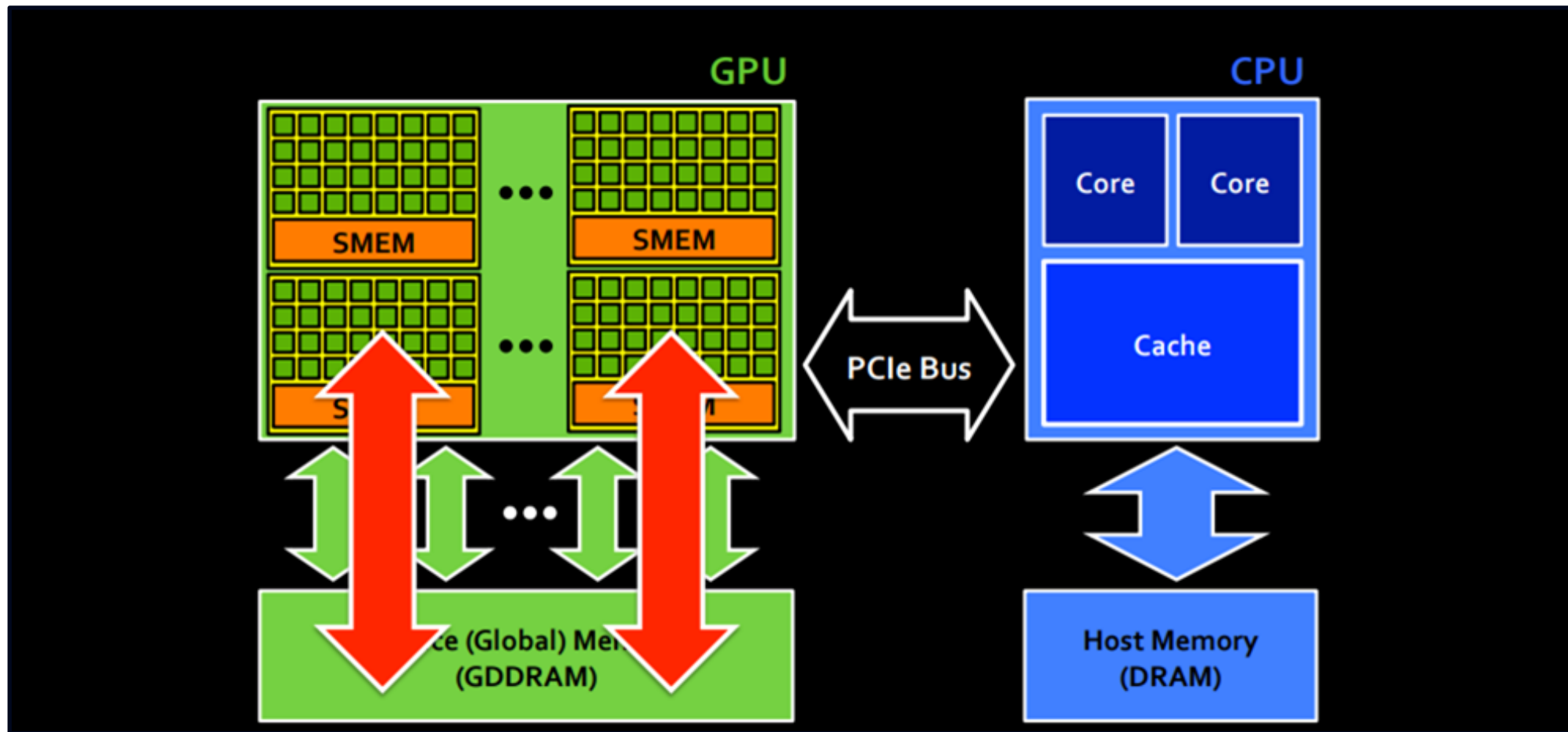
Heterogeneous Programming



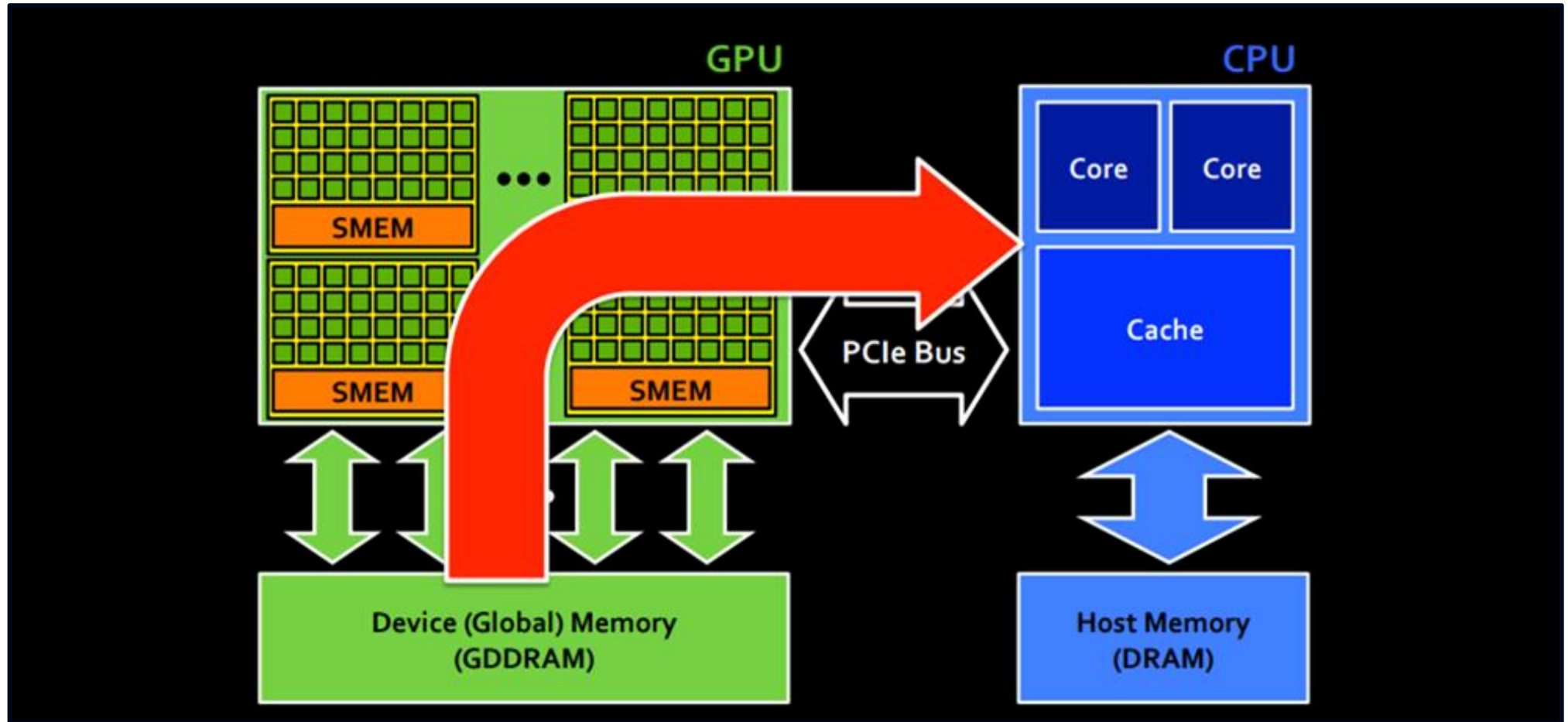
Execution FLOW – H2D



Execution FLOW – kernel launch



Execution FLOW – D2H



GH100 GPU architecture

<https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper>



GH100 GPU architecture

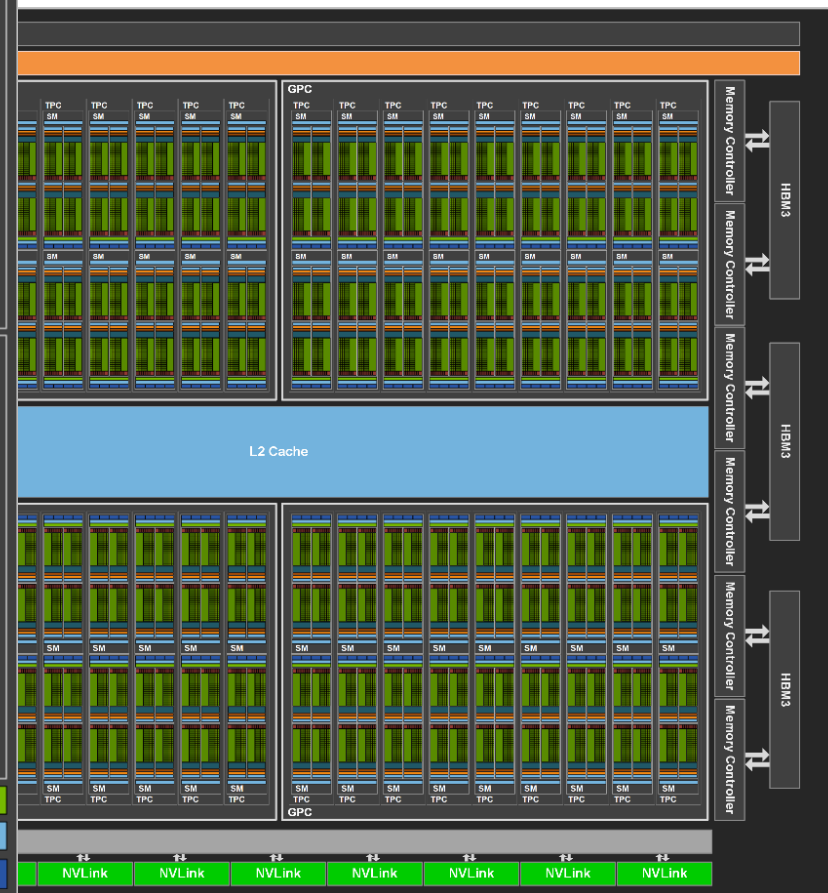
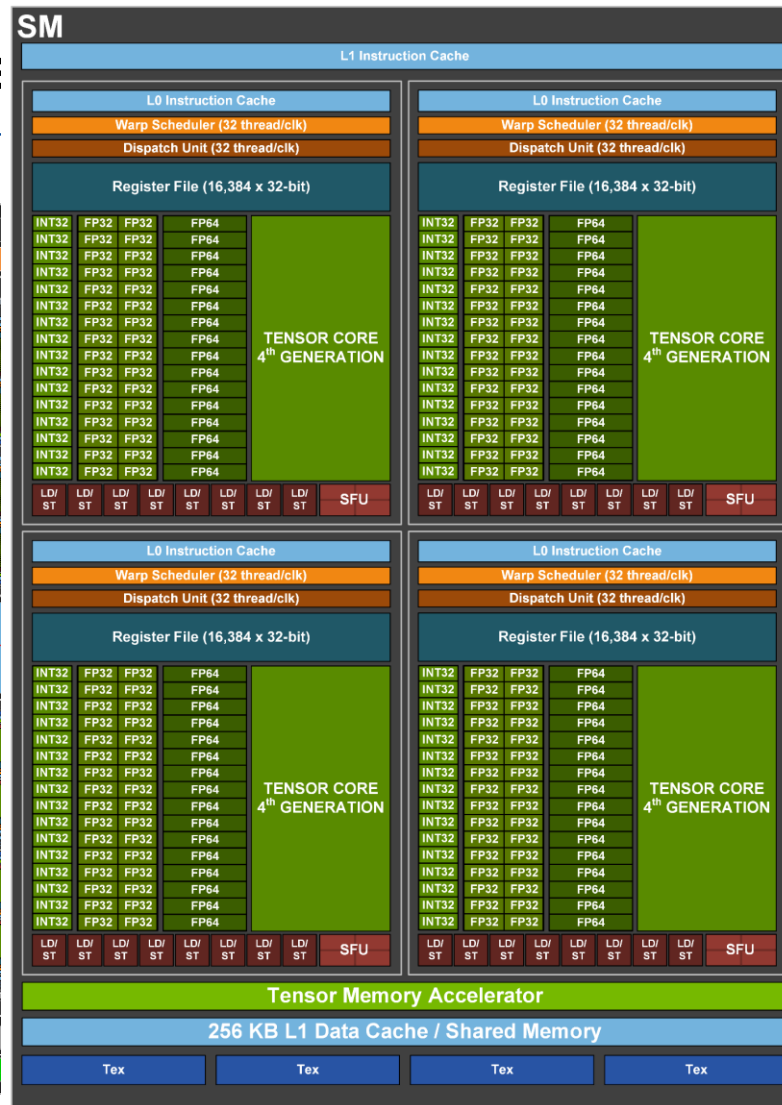
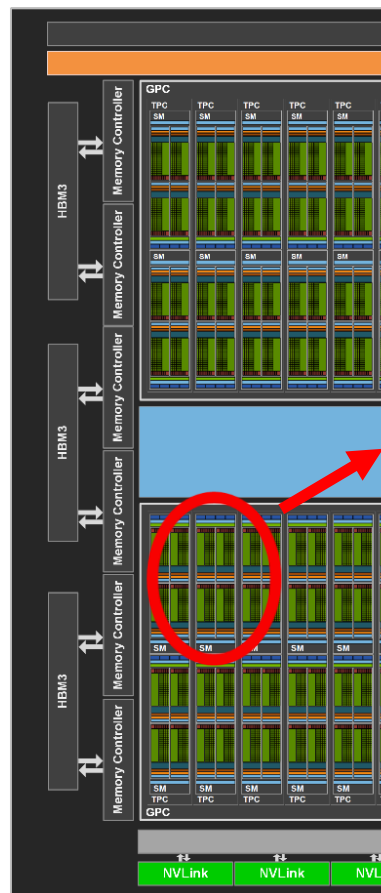
<https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper>



GH100 GPU architecture

ht

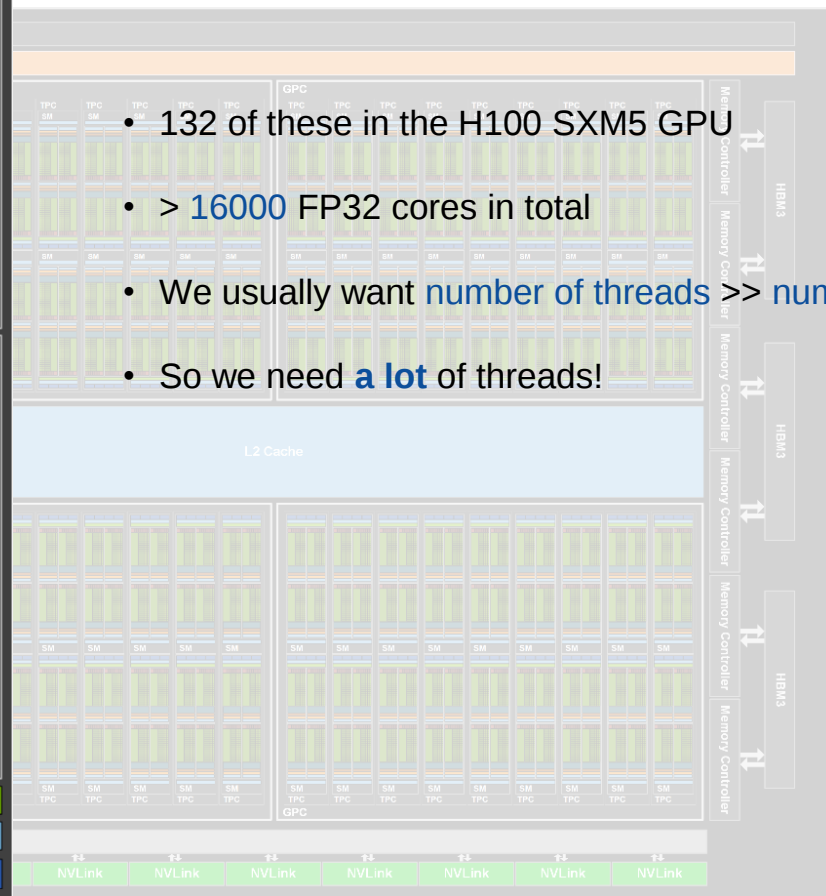
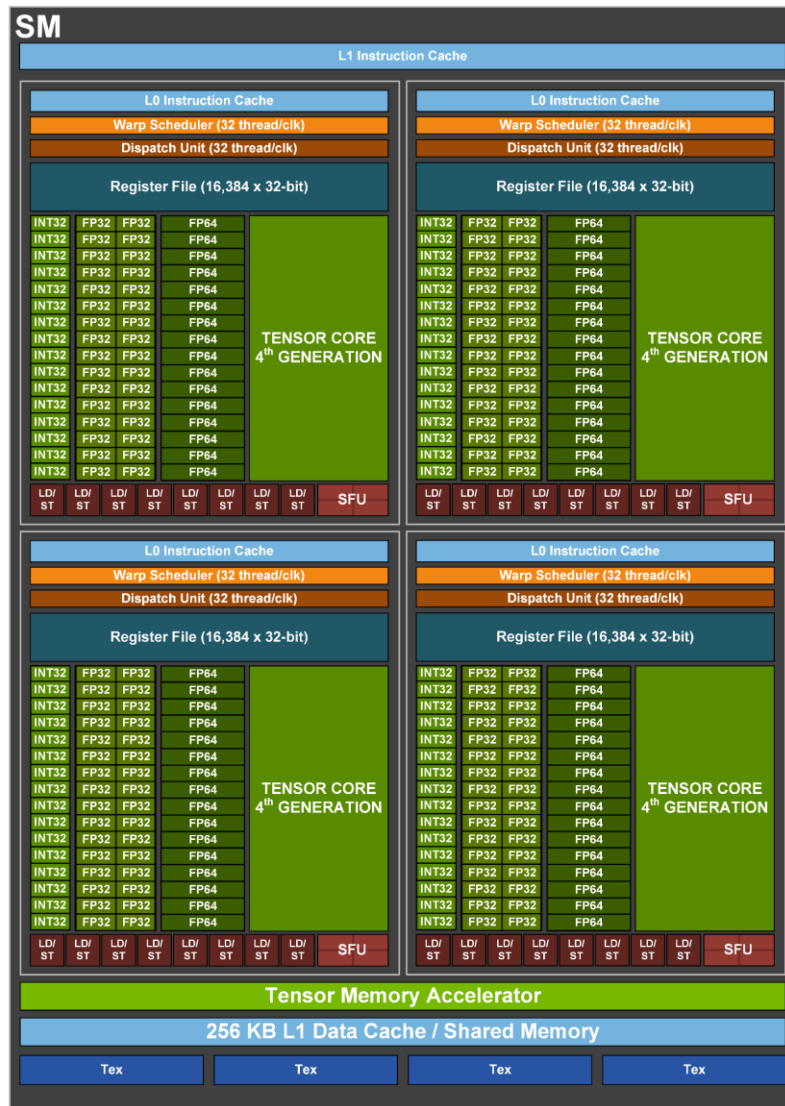
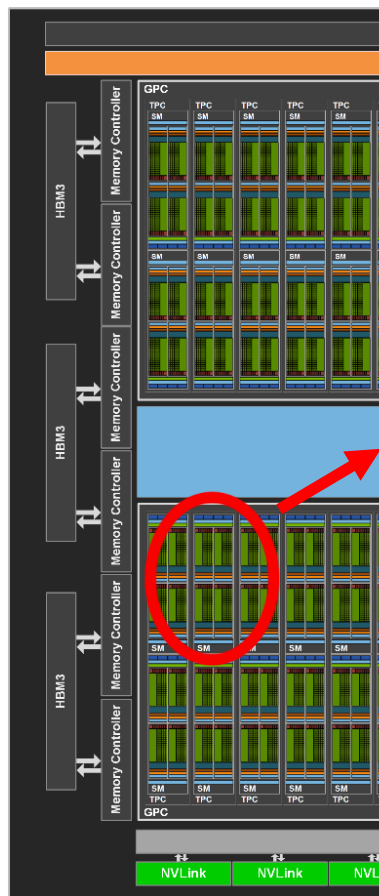
e/gtc22-whitepaper-hopper



GH100 GPU architecture

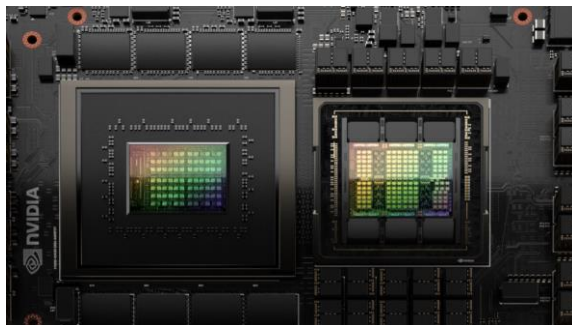
ht

e/gtc22-whitepaper-hopper

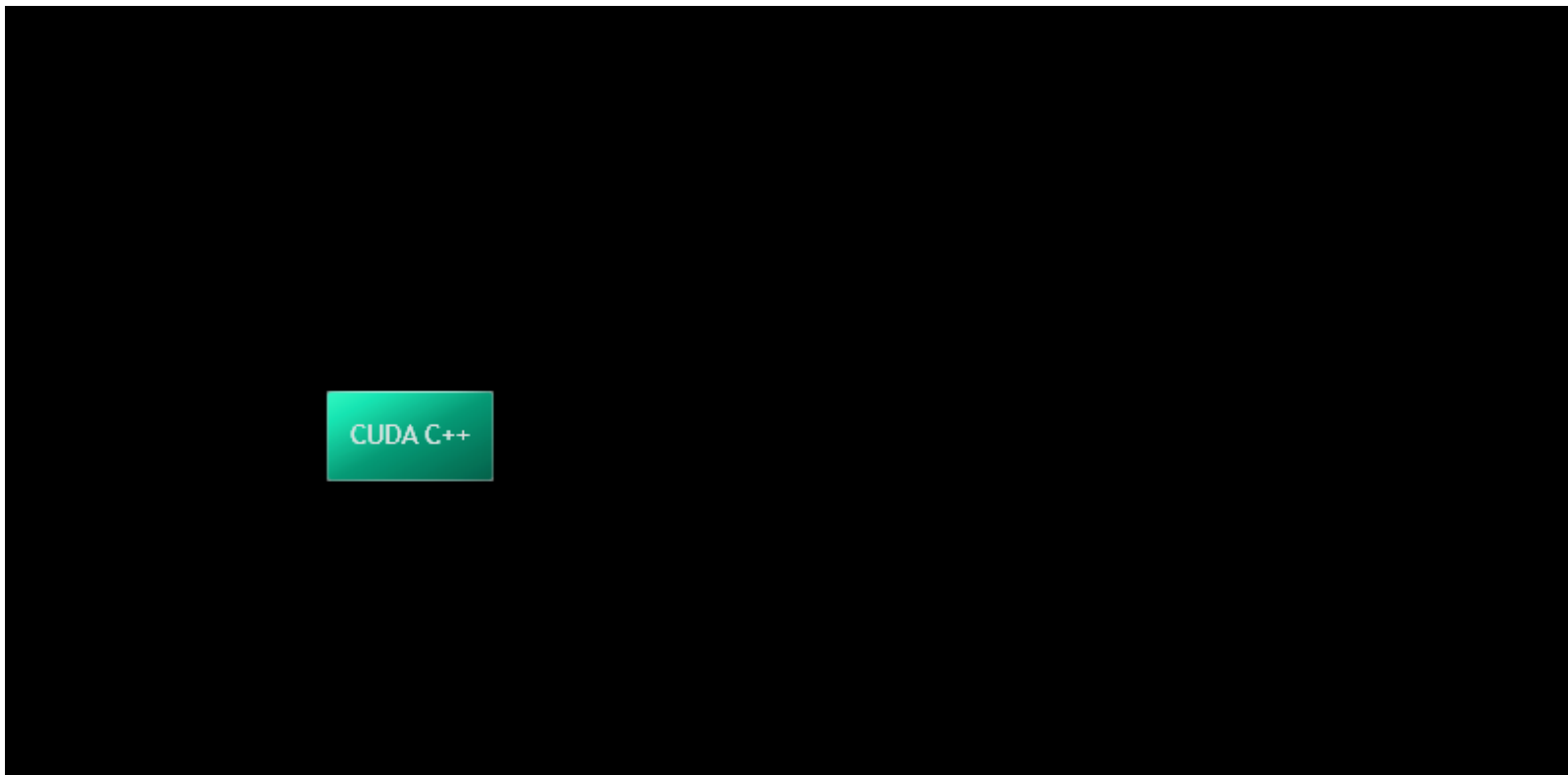


- 132 of these in the H100 SXM5 GPU
- > 16000 FP32 cores in total
- We usually want number of threads >> num cores
- So we need a lot of threads!

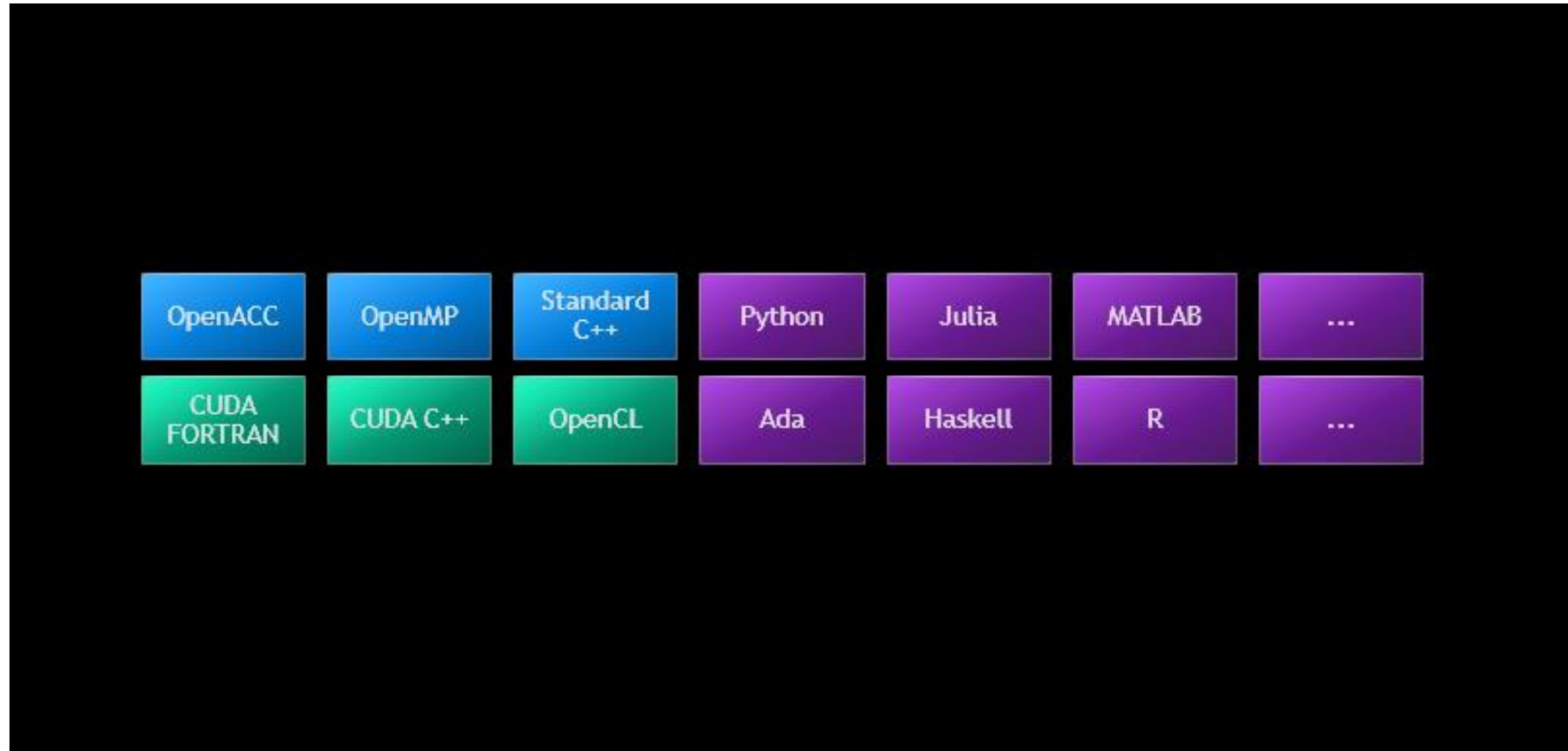
Accelerating at all scales



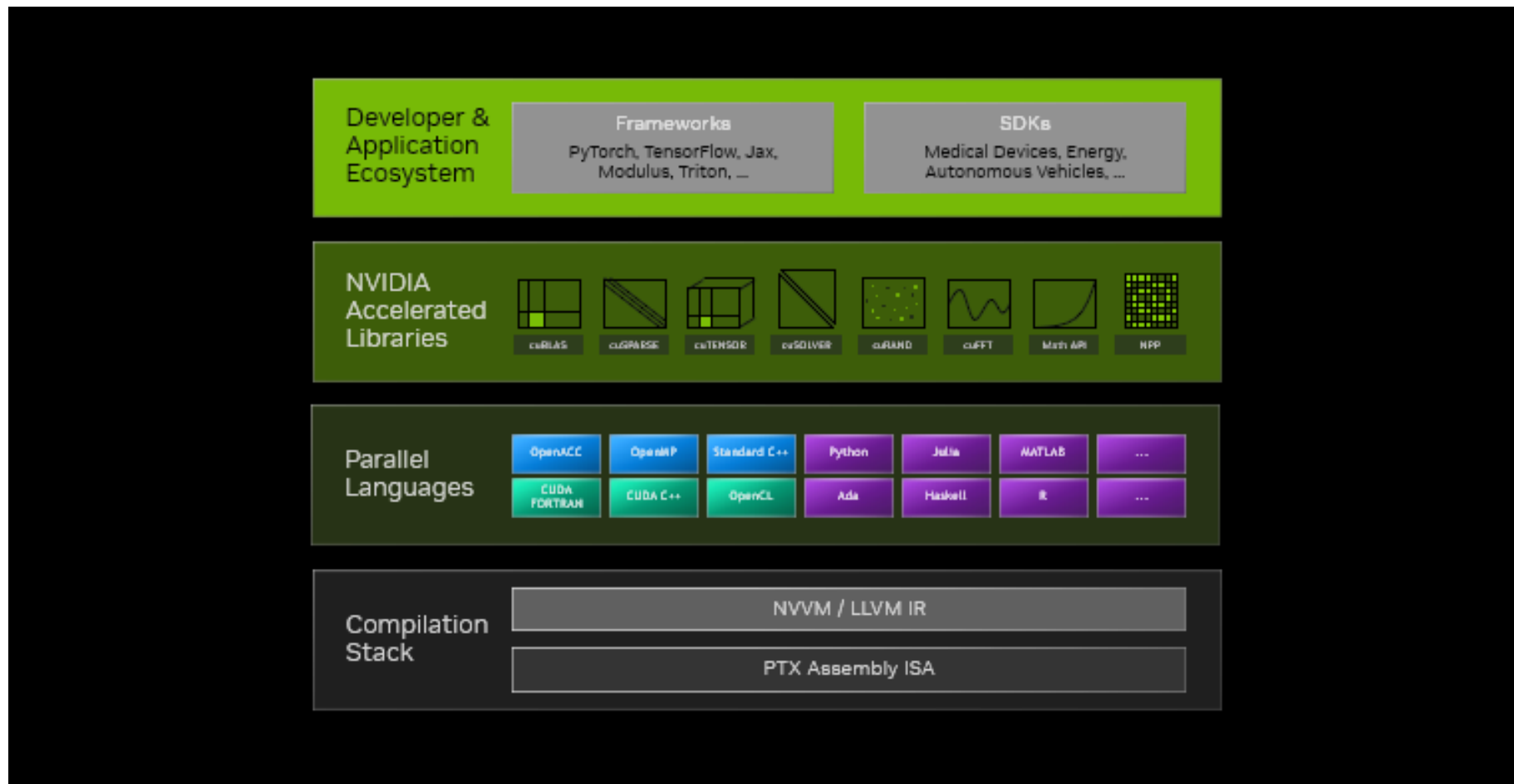
What is CUDA?



It's Not Just C++



Target the Abstraction Layer That Works Best For Your Application



Ways to ACCELERATION

Applications and Frameworks

Libraries

“Drop-in”
Acceleration

Directives

Easily Accelerate
Applications

Programming
Languages

Maximum
Flexibility

GPU accelerated APPS and frameworks

- All major DL frameworks – PyTorch, TensorFlow etc
- Top 15 most used HPC apps globally
- Apps in a huge range of fields
 - Over 3000 apps in total - catalogue: [link](#)
- Domain-specific frameworks – robotics, vis, healthcare, smart cities etc
- [GROMACS](#), [VASP](#), LAMMPS, RELION, QE, NAMD, SPECFEM3D ...

<https://developer.nvidia.com/hpc-application-performance>

GPU ACCELERATED MATH LIBRARIES



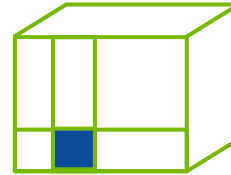
cuBLAS

BF16, TF32 and FP64
Tensor Cores



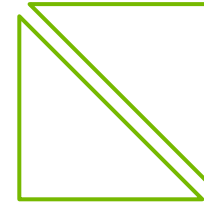
cuSPARSE

Increased memory BW,
Shared Memory & L2



cuTENSOR

BF16, TF32 and
FP64 Tensor Cores



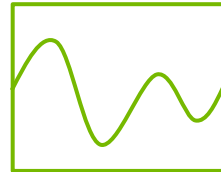
cuSOLVER

BF16, TF32 and
FP64 Tensor Cores



nvJPEG

Hardware Decoder



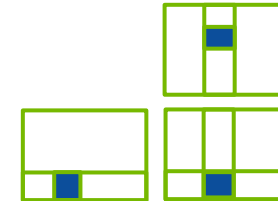
cuFFT

BF16, TF32 and FP64
Tensor Cores



CUDA Math API

Increased memory BW,
Shared Memory & L2



CUTLASS

BF16 & TF32
Support

Programming the NVIDIA Platform

CPU, GPU, and Network

ACCELERATED STANDARD

ISO C++, ISO Fortran

```
std::transform(par, x, x+n, y, y,  
    [=] (float x, float y){ return y +  
    a*x; }  
);
```

```
do concurrent (i = 1:n)  
    y(i) = y(i) + a*x(i)  
enddo
```

```
import cunumeric as np  
...  
def saxpy(a, x, y):  
    y[:] += a*x
```

INCREMENTAL PORTABLE

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {  
    ...  
    std::transform(par, x, x+n, y, y,  
        [=] (float x, float y){  
            return y + a*x;  
        });  
    ...  
}  
  
#pragma omp target data map(x,y) {  
    ...  
    std::transform(par, x, x+n, y, y,  
        [=] (float x, float y){  
            return y + a*x;  
        });  
    ...  
}
```

PLATFORM SPECIALIZATION

CUDA

```
__global__  
void saxpy(int n, float a,  
    float *x, float *y) {  
    int i = blockIdx.x*blockDim.x +  
        threadIdx.x;  
    if (i < n) y[i] += a*x[i];  
}  
  
int main(void) {  
    ...  
    cudaMemcpy(d_x, x, ...);  
    cudaMemcpy(d_y, y, ...);  
  
    saxpy<<<(N+255)/256,256>>>(...);  
  
    cudaMemcpy(y, d_y, ...);  
}
```

ACCELERATION LIBRARIES

Core

Math

Communication

Data Analytics

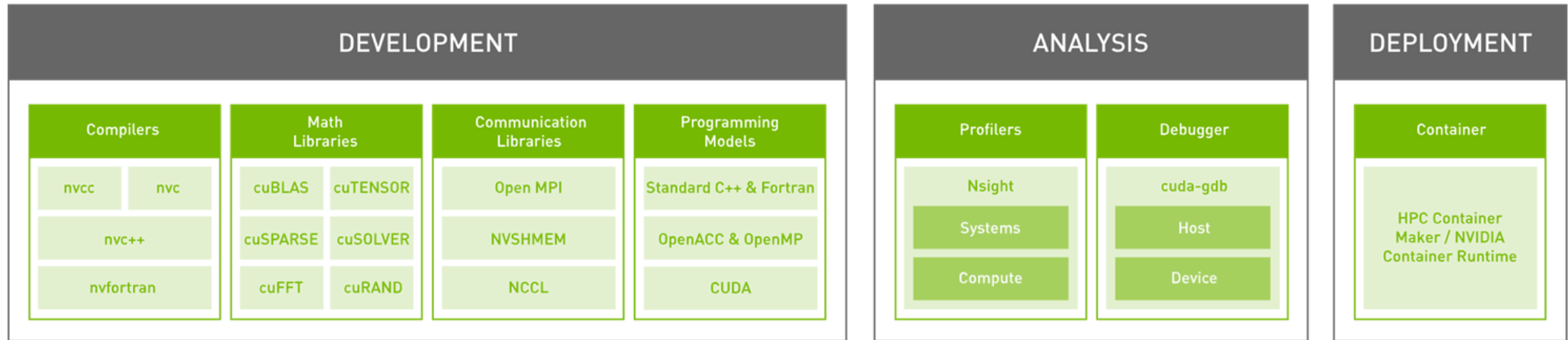
AI

Quantum

NVIDIA HPC SDK

Download at developer.nvidia.com/hpc-sdk

NVIDIA HPC SDK



Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect
HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

Programming Model Interoperability

All programming models are interoperable:

- Can be used in the same source file.
- ABI compatible with each other and NVCC.

```
nvc++ -stdpar -cuda -acc -mp
```



Standard
Parallelism

CUDA

OpenACC

OpenMP

```
__global__ void  
times_two(float* first, uint64_t n) {  
    auto t = blockIdx.x*blockDim.x+threadIdx.x;  
    if (t < n) first[t] *= 2;  
}
```

```
void compute(std::vector<float>& x) {  
    auto const b = ((x.size() - 1) / 64) + 1;  
    times_two<<<b, 64>>>(x.data(), x.size());  
    cudaDeviceSynchronize();  
}
```

```
#pragma omp target teams loop  
for (auto& e : x) e *= e;
```

```
#pragma acc parallel loop  
for (auto& e : x) e += e;
```

```
std::sort(std::execution::par,  
          x.begin(), x.end());
```

```
}
```

Acknowledgment

Copyright © 2022 OpenACC-Standard.org. This material is released by OpenACC-Standard.org, in collaboration with NVIDIA Corporation, under the Creative Commons Attribution 4.0 International (CC BY 4.0). These materials may include references to hardware and software developed by other entities; all applicable licensing and copyrights apply.