Computação Paralela

Mest. Int. Engenharia Computacional

Ano letivo 2019/2020

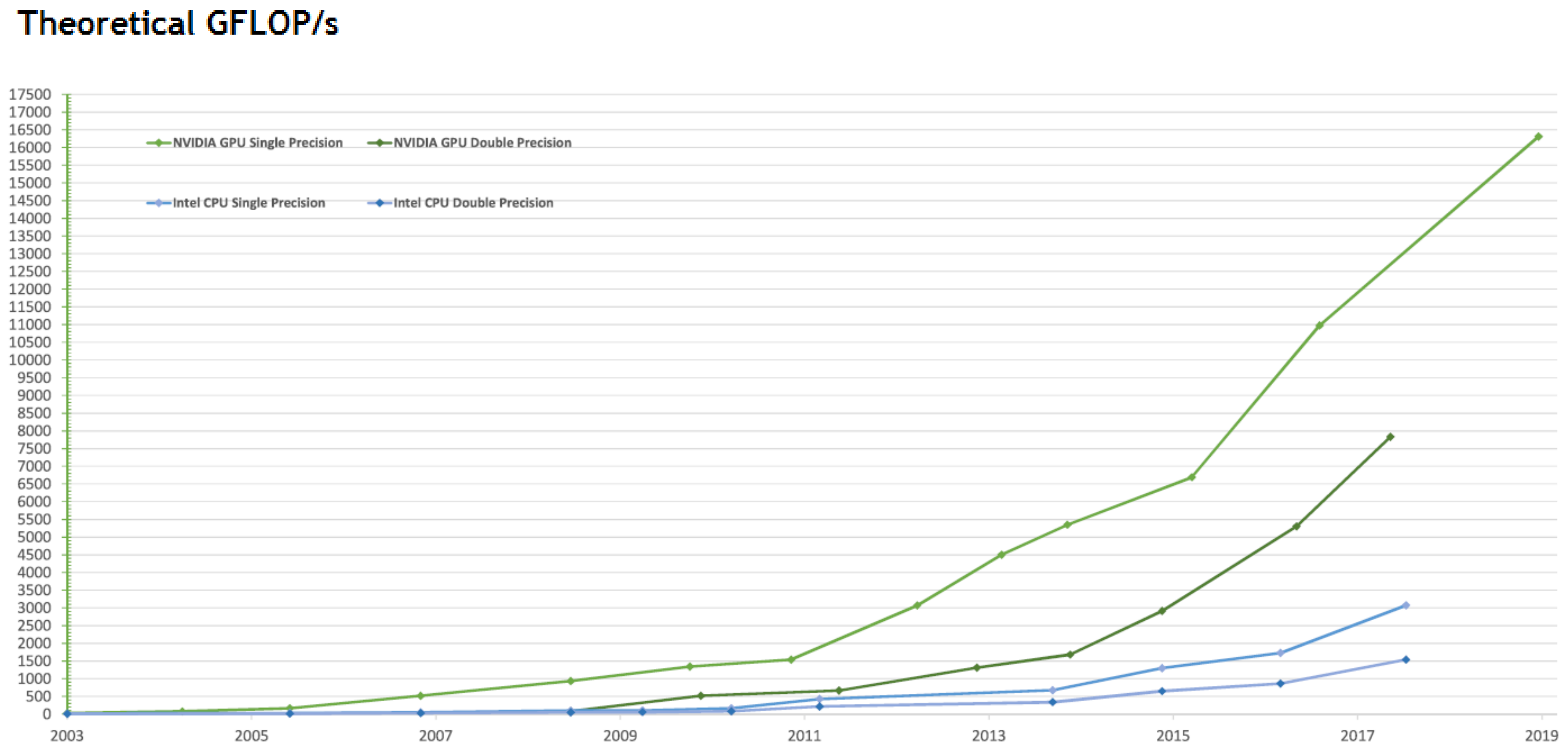
Manuel Barroso, Nuno Lau

CUDA

- Parallel general purpose computation model
- Introduced by NVIDIA in 2006
- Allows using the GPU for the execution of general purpose applications

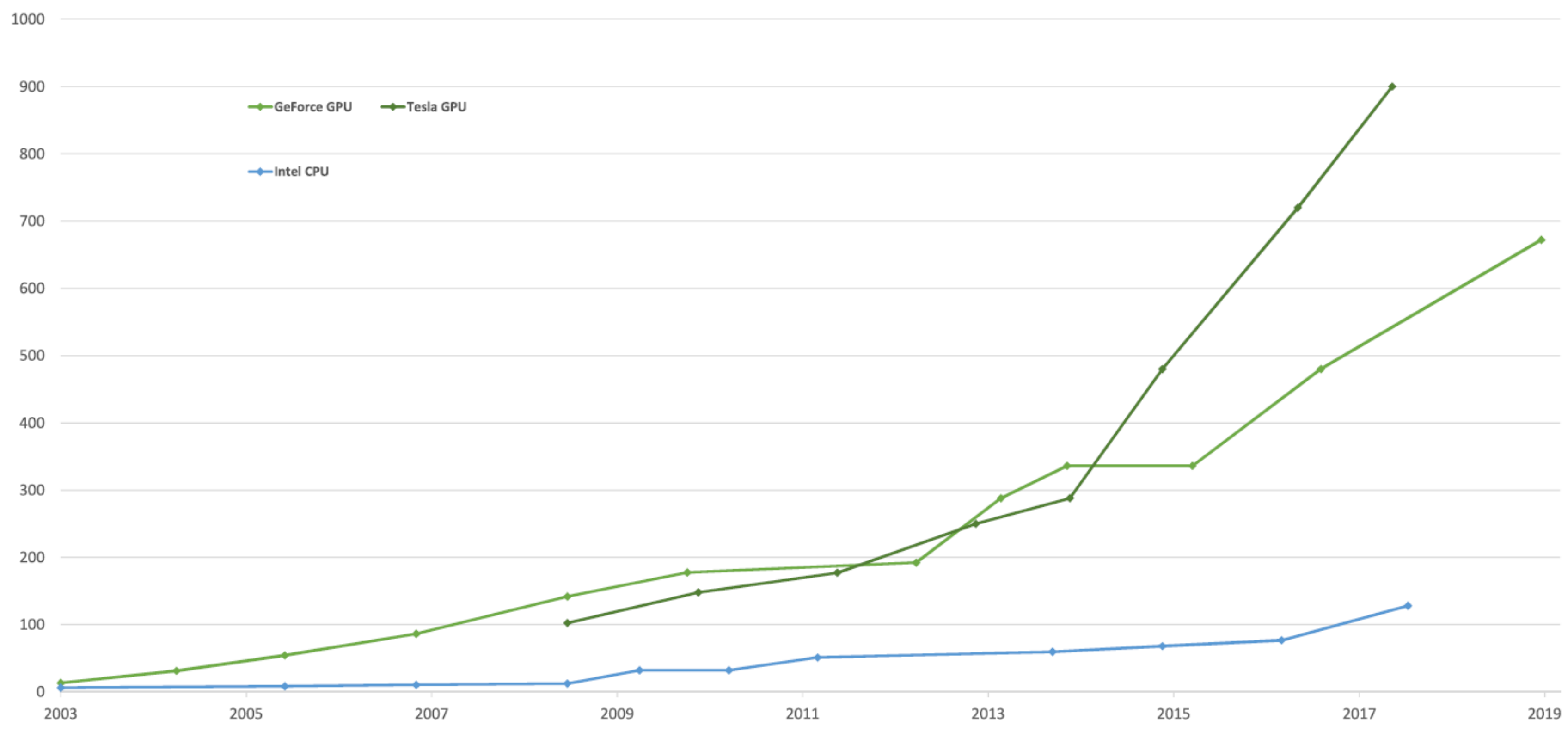
Acknowledgement: This lecture is based on “CUDA C++ Programming Guide”, v10.2, NVIDIA

CUDA

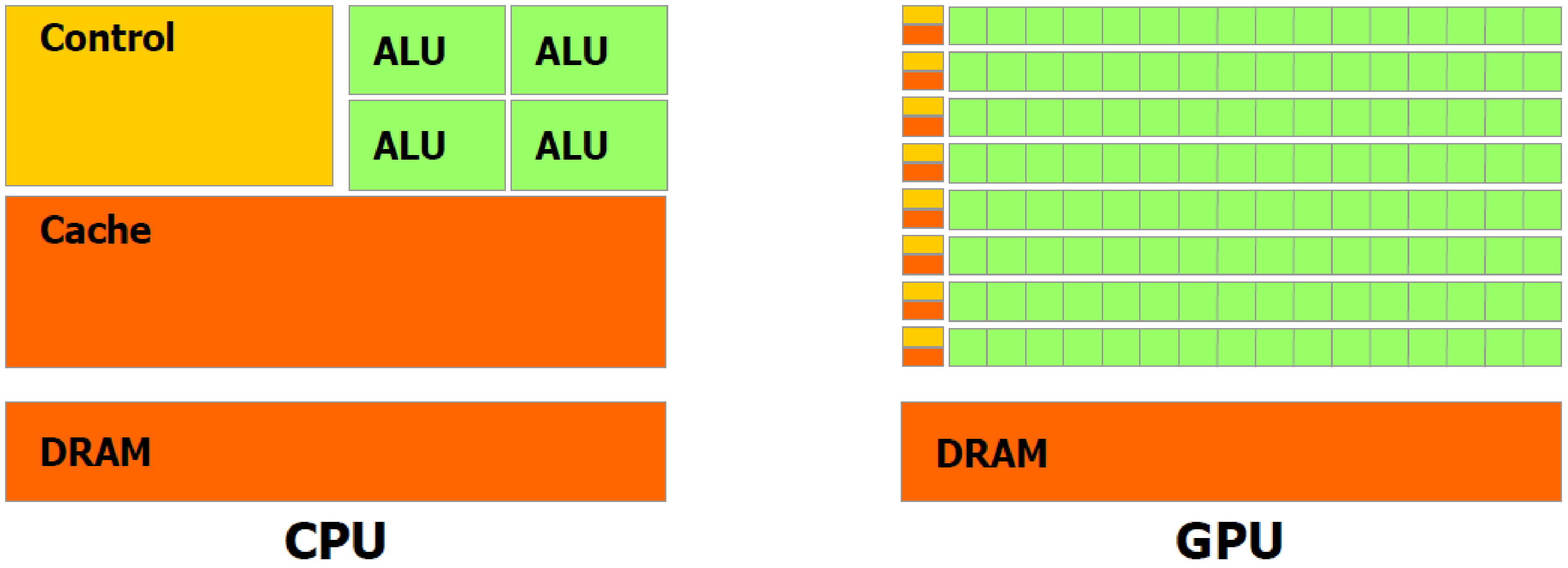

Theoretical GFLOP/s
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
17500
17000
16500
16000
15500
15000
14500
14000
13500
13000
12500
12000
11500
11000
10500
10000
9500
9000
8500
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
2003
2005
2007
2009
2011
2013
2015
2017
2019

CUDA

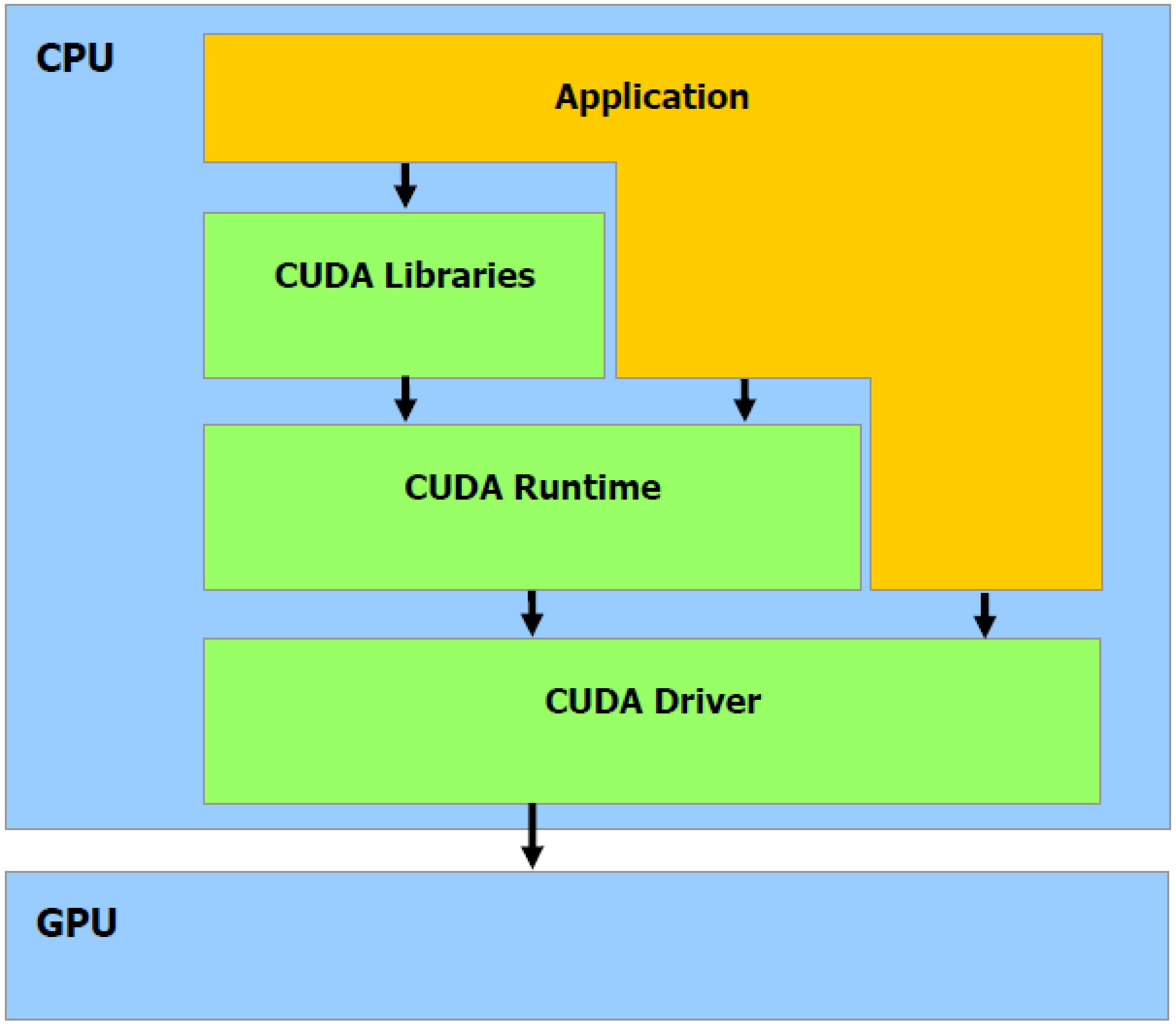
Theoretical GB/s



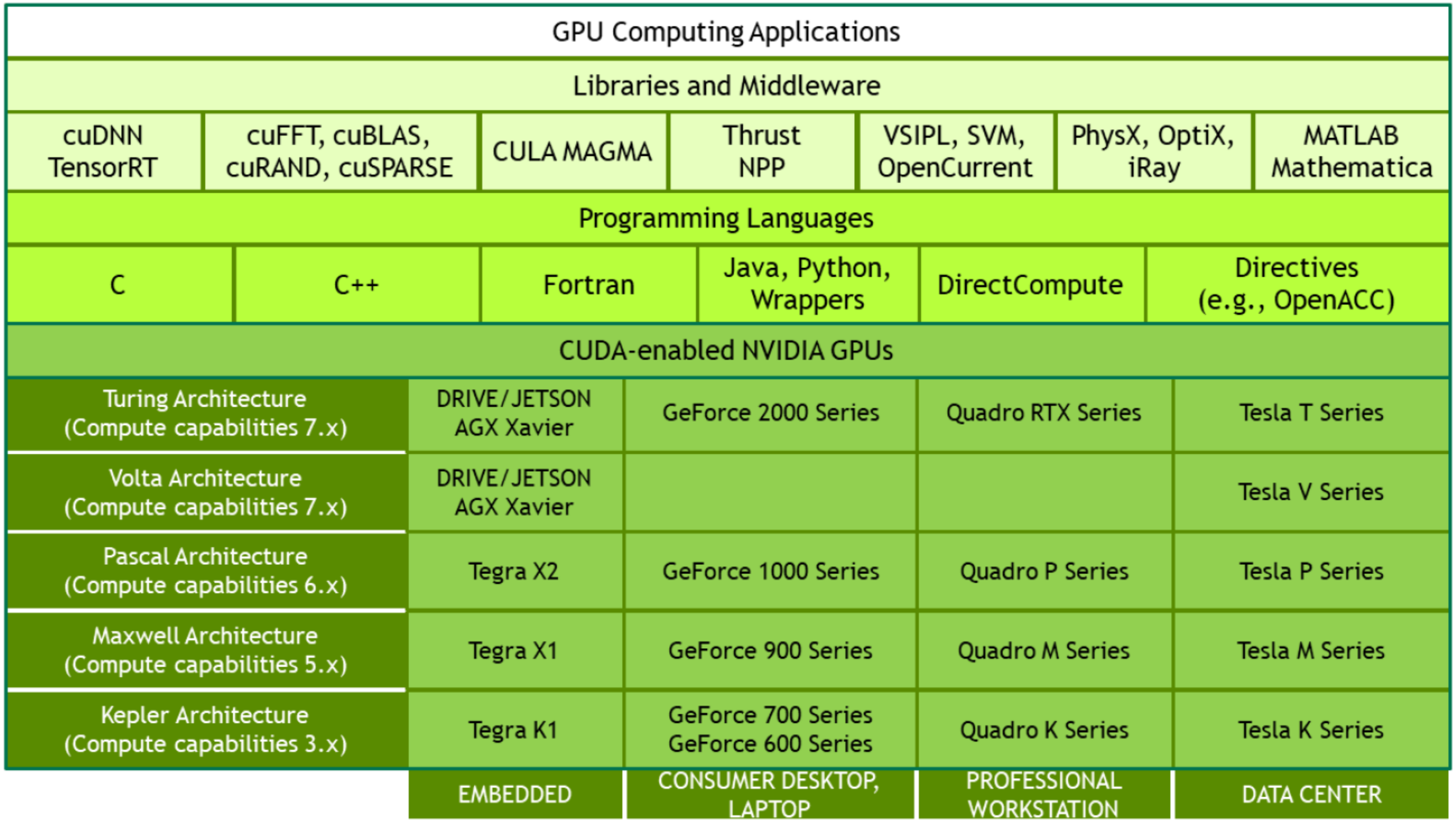
CUDA – CPU and GPU Architecure



CUDA Architecture



CUDA Computing Applications



GPU Computing Applications

Libraries and Middleware

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

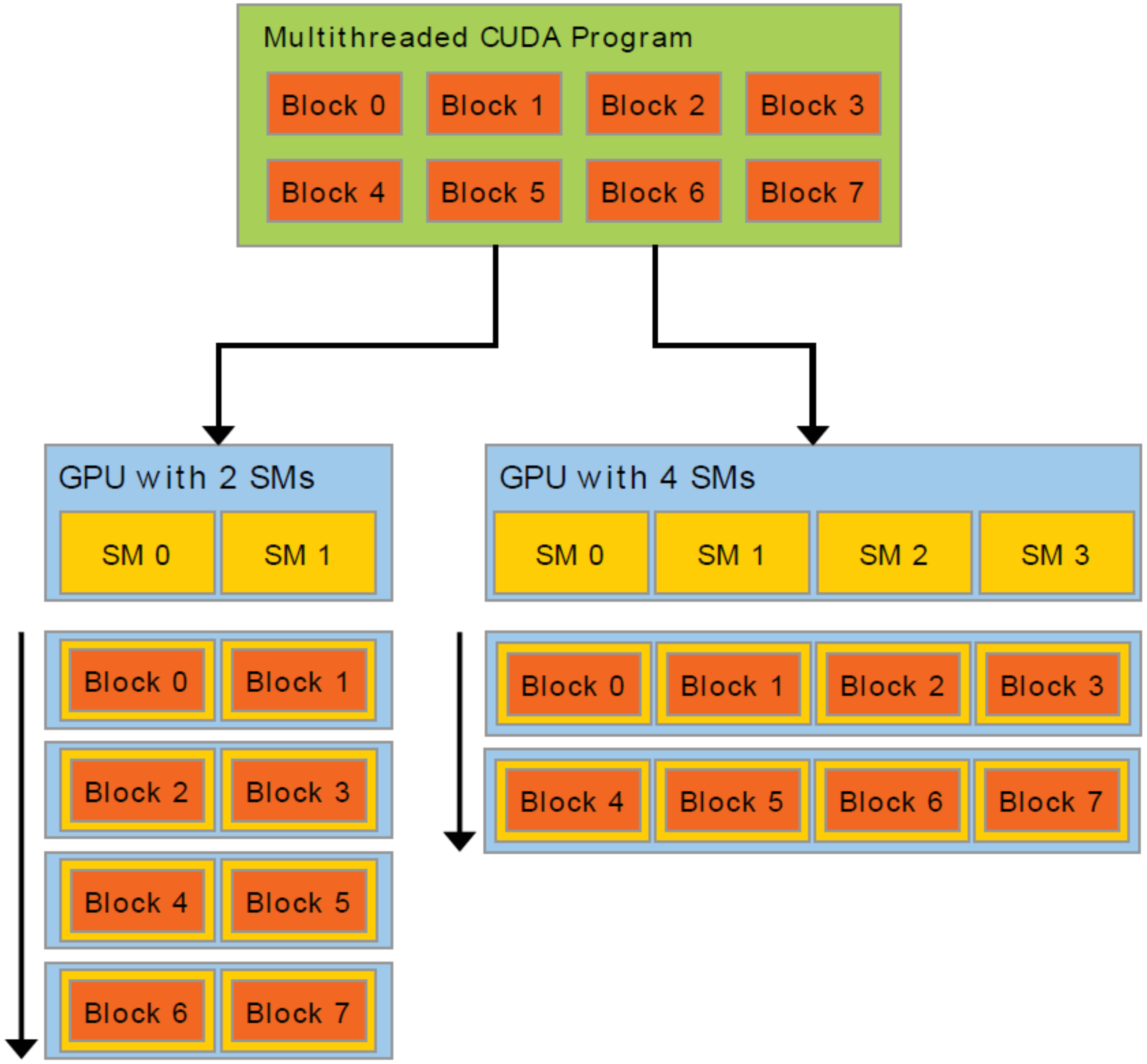
Programming Languages

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

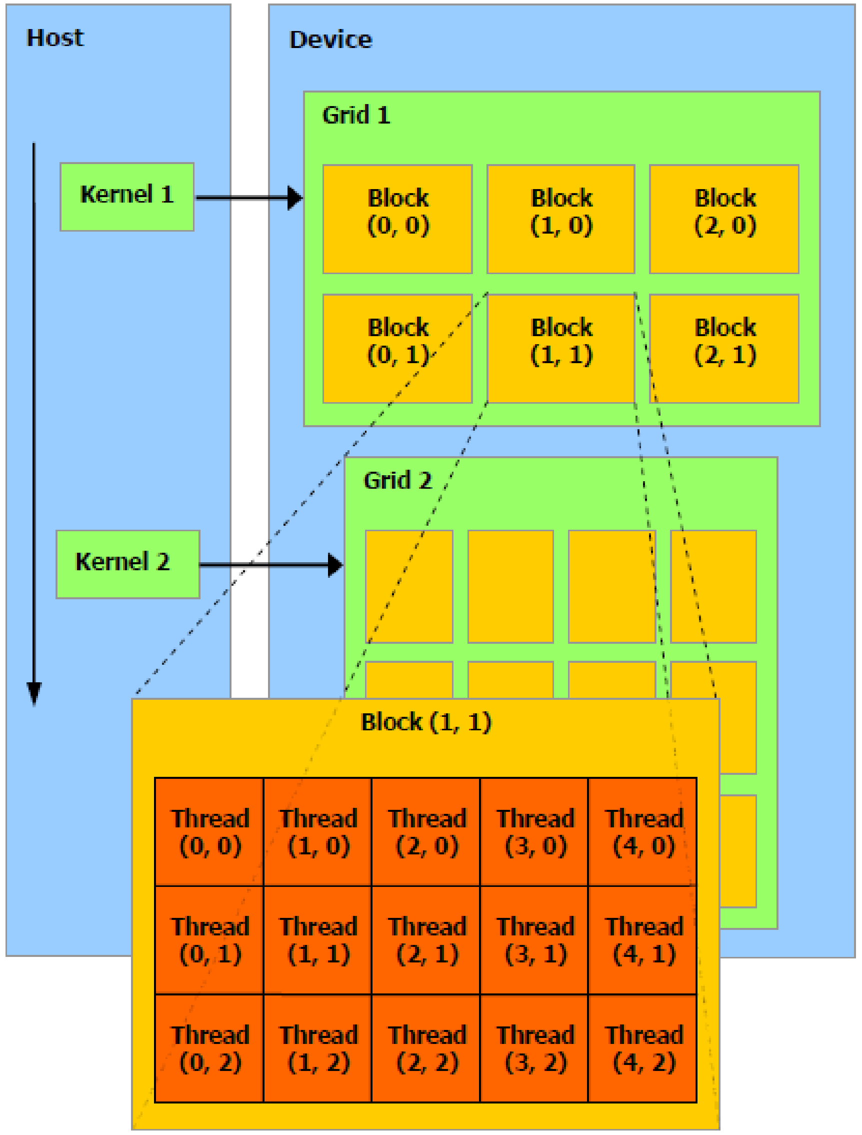
CUDA-enabled NVIDIA GPUs

Turing Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier	GeForce 2000 Series	Quadro RTX Series	Tesla T Series
Volta Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier			Tesla V Series
Pascal Architecture (Compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
Maxwell Architecture (Compute capabilities 5.x)	Tegra X1	GeForce 900 Series	Quadro M Series	Tesla M Series
Kepler Architecture (Compute capabilities 3.x)	Tegra K1	GeForce 700 Series GeForce 600 Series	Quadro K Series	Tesla K Series
	EMBEDDED	CONSUMER DESKTOP, LAPTOP	PROFESSIONAL WORKSTATION	DATA CENTER

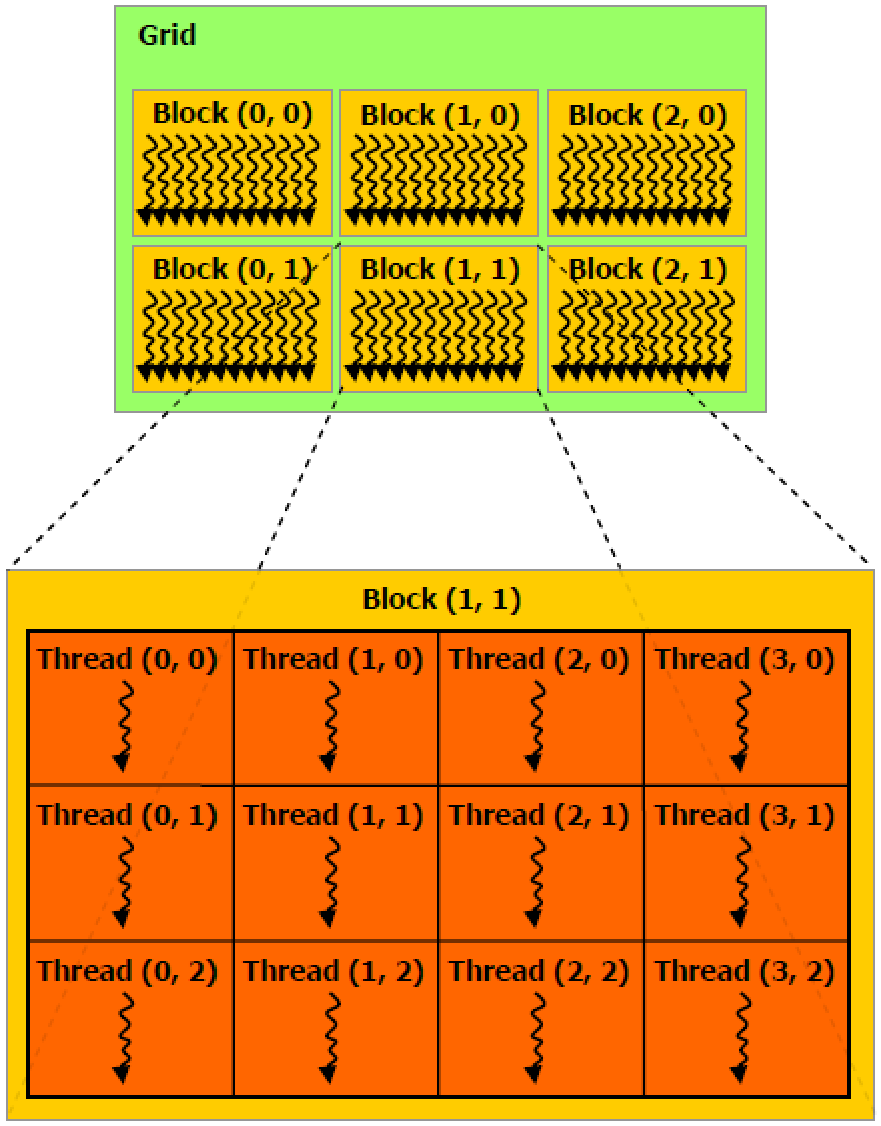
CUDA Automatic Scalability



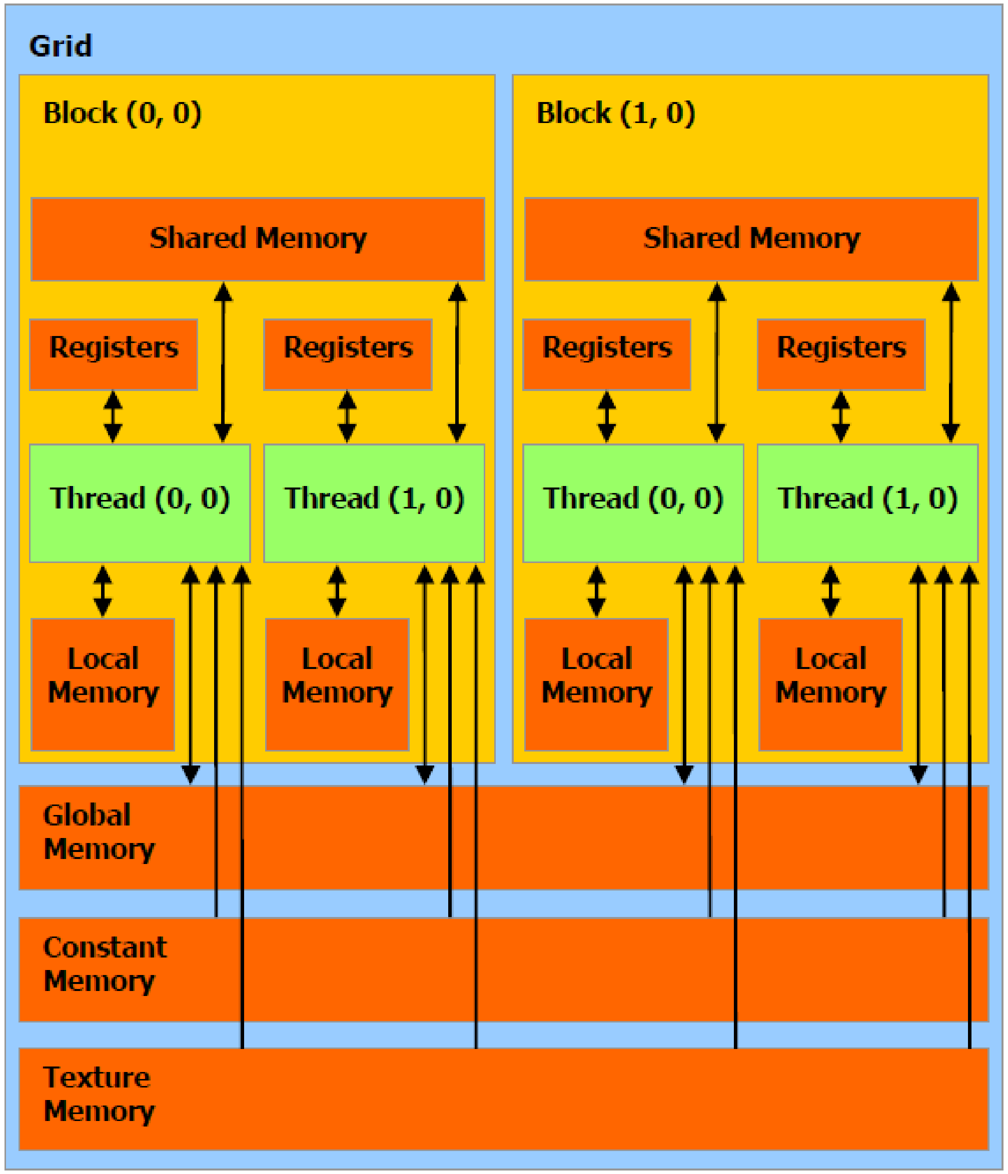
CUDA Kernel, Grid, Block, Thread



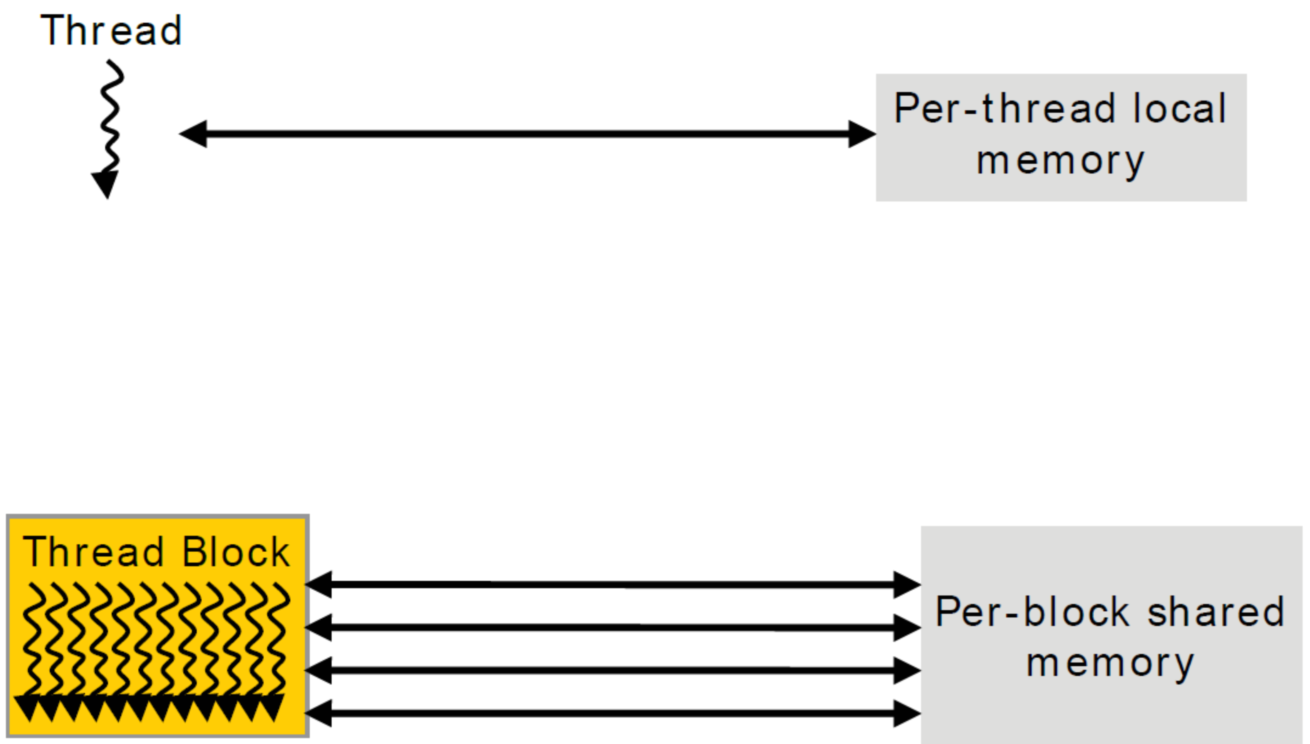
CUDA Grid of Threads



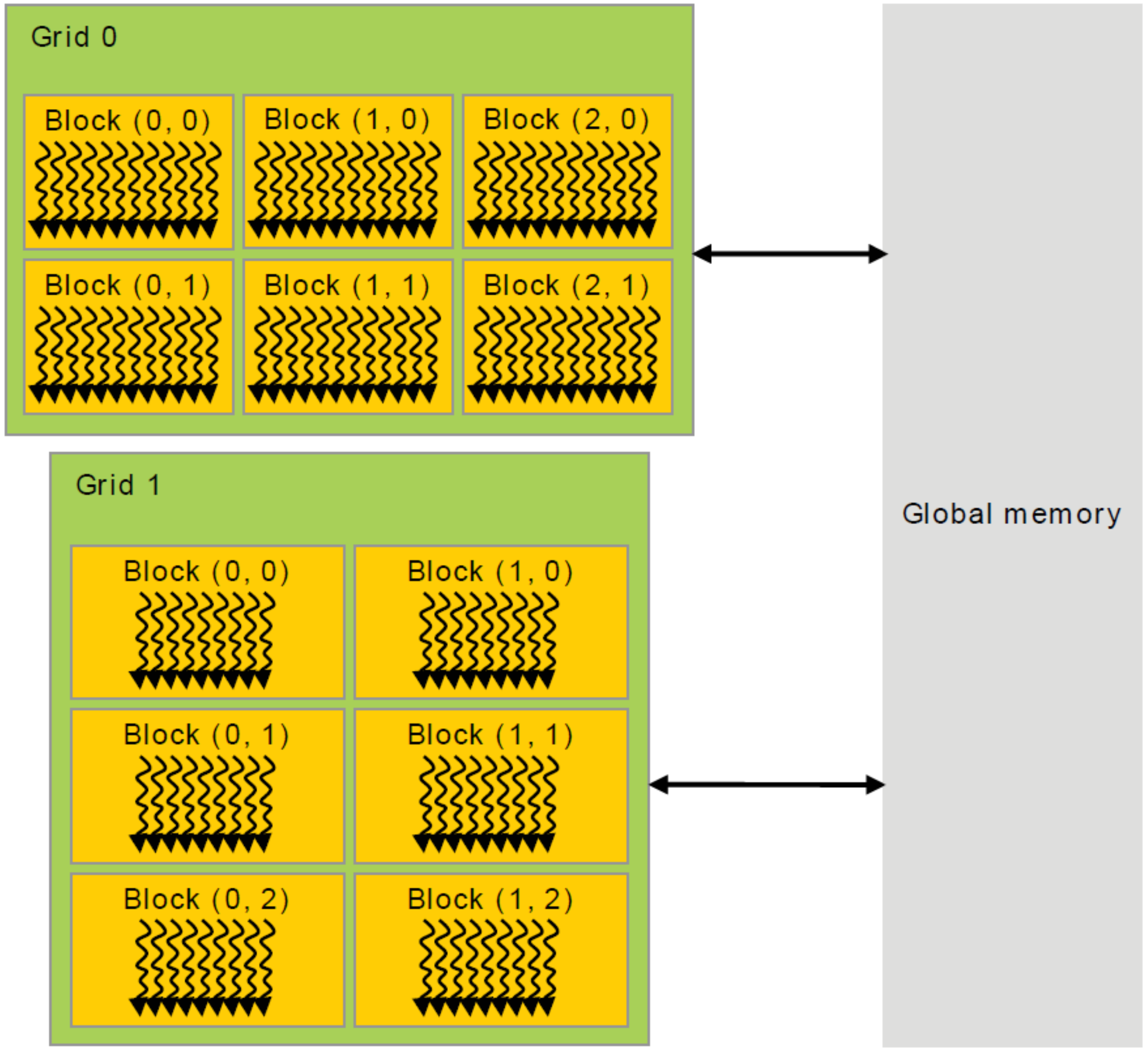
CUDA memory


Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Local Memory
Local Memory
Local Memory
Local Memory
Global Memory
Constant Memory
Texture Memory

CUDA memory


Thread
Per-thread local memory
Thread Block
Per-block shared memory

CUDA memory


Grid 0
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 1
Block (0, 0)
Block (1, 0)
Block (0, 1)
Block (1, 1)
Block (0, 2)
Block (1, 2)
Global memory

CUDA Compute Capability

Compute Capability 1.x

- Global memory (read and write)
  - Slow and uncached
  - Requires sequential and aligned 16 byte reads and writes to be fast
- Texture memory (read only)
  - Cache optimized for 2D spatial access pattern
- Constant memory
  - This is where constants and kernel arguments are stored
  - Slow, but with small cache
- Shared memory (16 kb per MP)
  - Fast, but take care of bank conflicts
  - Can be used to exchange data between threads in a block
- Local memory (used for data that does not fit into registers)
  - Slow and uncached
- Registers
  - Fastest, scope is thread local

CUDA Compute Capability

Compute Capability 2.x

- Global memory (read and write)
  - Slow, but now with cache
- Texture memory (read only)
  - Cache optimized for 2D spatial access pattern
- Constant memory
  - Slow, but with cache (8 kb)
- Shared memory
  - Fast, but slightly different rules for bank conflicts now
- Local memory
  - Slow, but now with cache
- Registers (32768 32-bit registers per MP)

CUDA Compute Capability

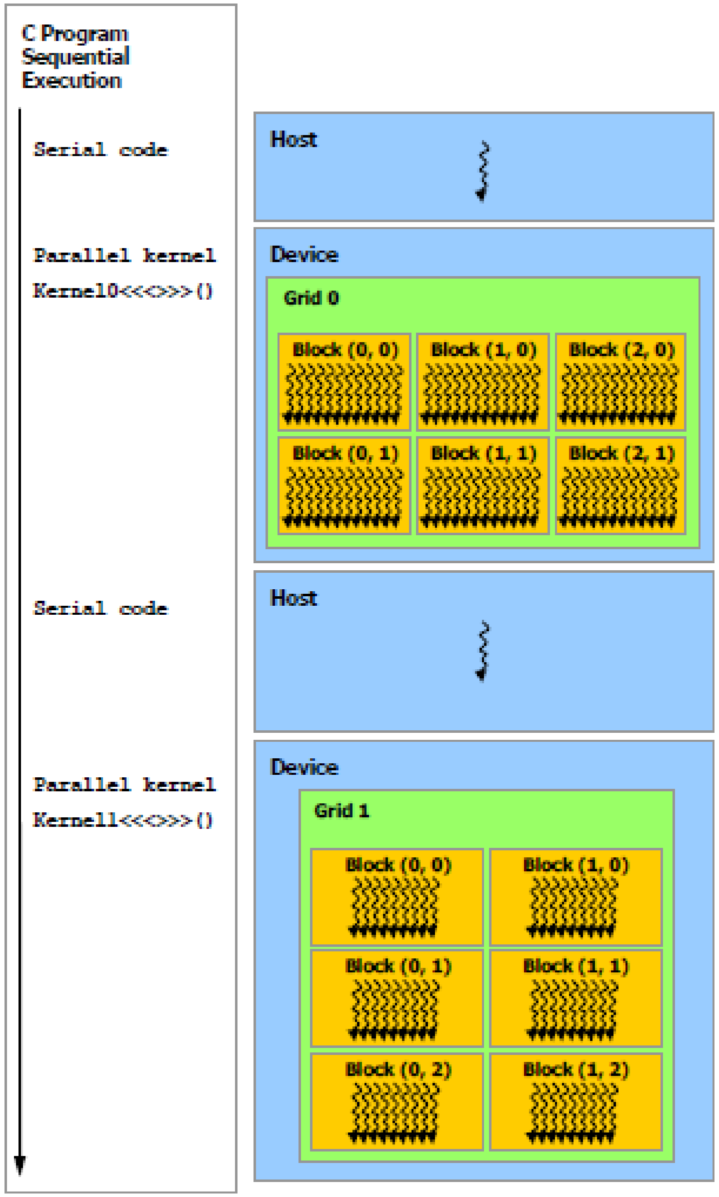
Compute Capability 3.x

- Global memory (read and write)
  - Generally cached in L2, but not in L1
- Constant memory
  - Cache shared by all functional units
- Shared memory
  - 32 banks with two addressing modes
- Local memory
  - Cached in L1

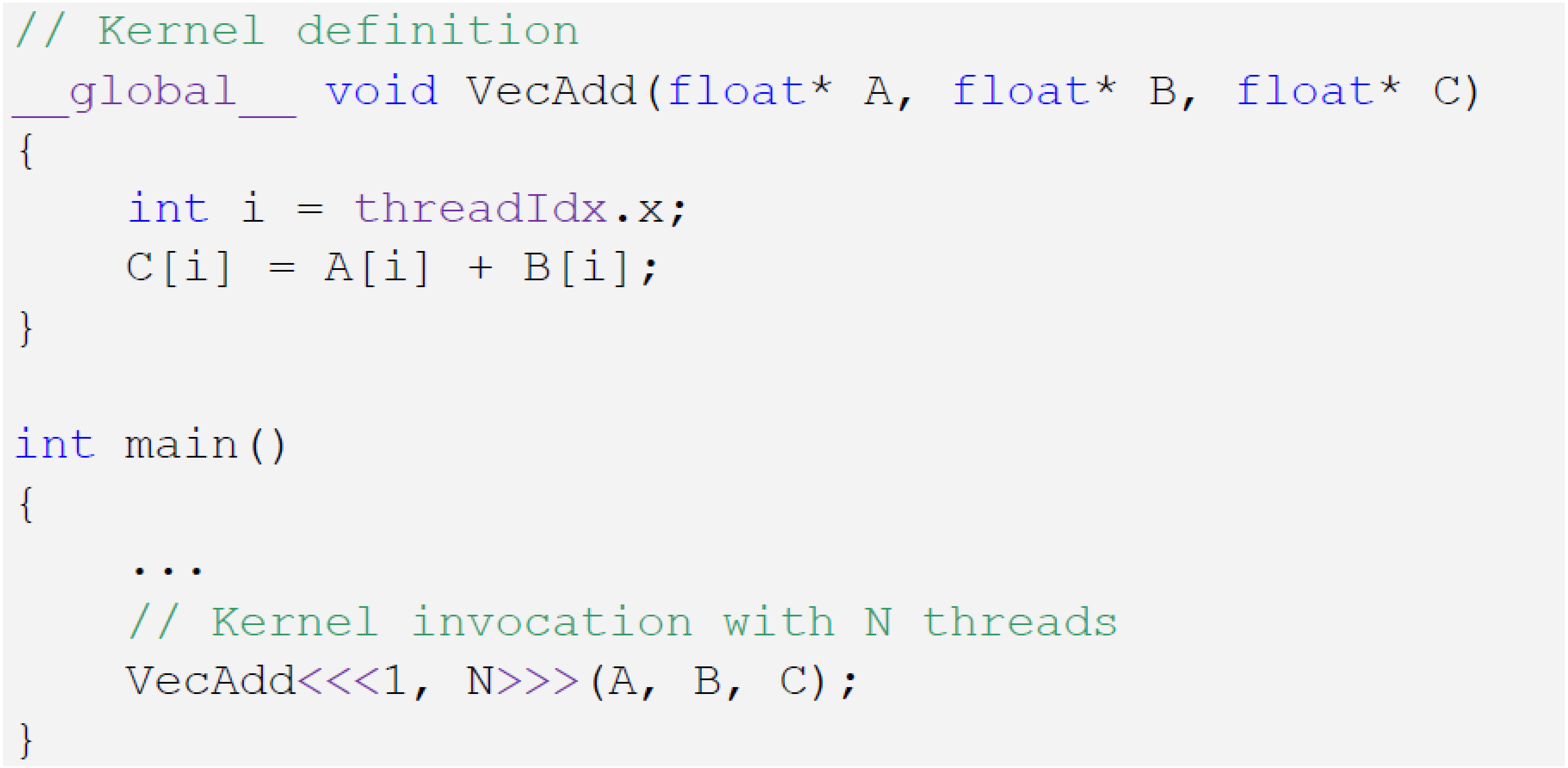
Compute Capability 5.x

- Global memory
  - Identical to cp 3.x
- Shared memory
  - 32 banks organized such that successive 32-bit words map to successive banks

CUDA Heterogeneous Programming



CUDA



```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
}
```

CUDA

- Kernel invocation

kernel<<< Dg, Db, Ns, S >>>

- **Dg** is the grid dimension (dim3 type)
- **Db** is the block dimension (dim3 type)
- **Ns** is the number of shared memory bytes
- **S** is the stream

CUDA

- Block
  - threadIdx.x, threadIdx.y, threadIdx.z
    - Identify the thread in the block
  - blockDim.x, blockDim.y, blockDim.z
  - threadID = x+y*Dx+z*Dx*Dy
  - Threads are executed in *warps*
    - 32 threads of the same block with consecutive ids
    - Blocks should have more than 32 threads

CUDA

- Grid
  - blockIdx.x, blockIdx.y, blockIdx.z
    - Identify the block in the grid
  - gridDim.x, gridDim.y, gridDim.z