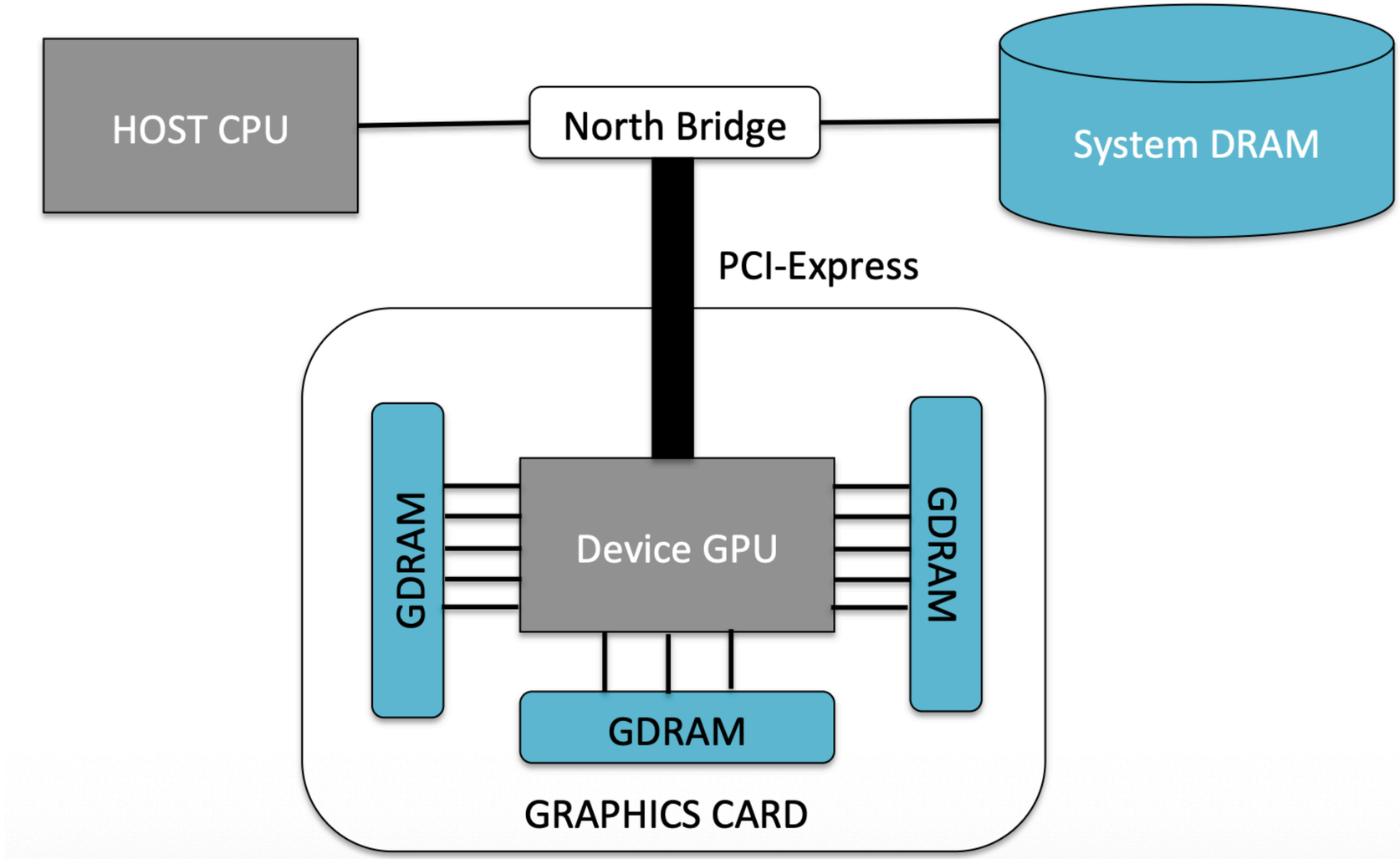
Introduction to Parallel & Distributed Programming

Lec 14— GPUs

Subodh Sharma | March 07, 2026

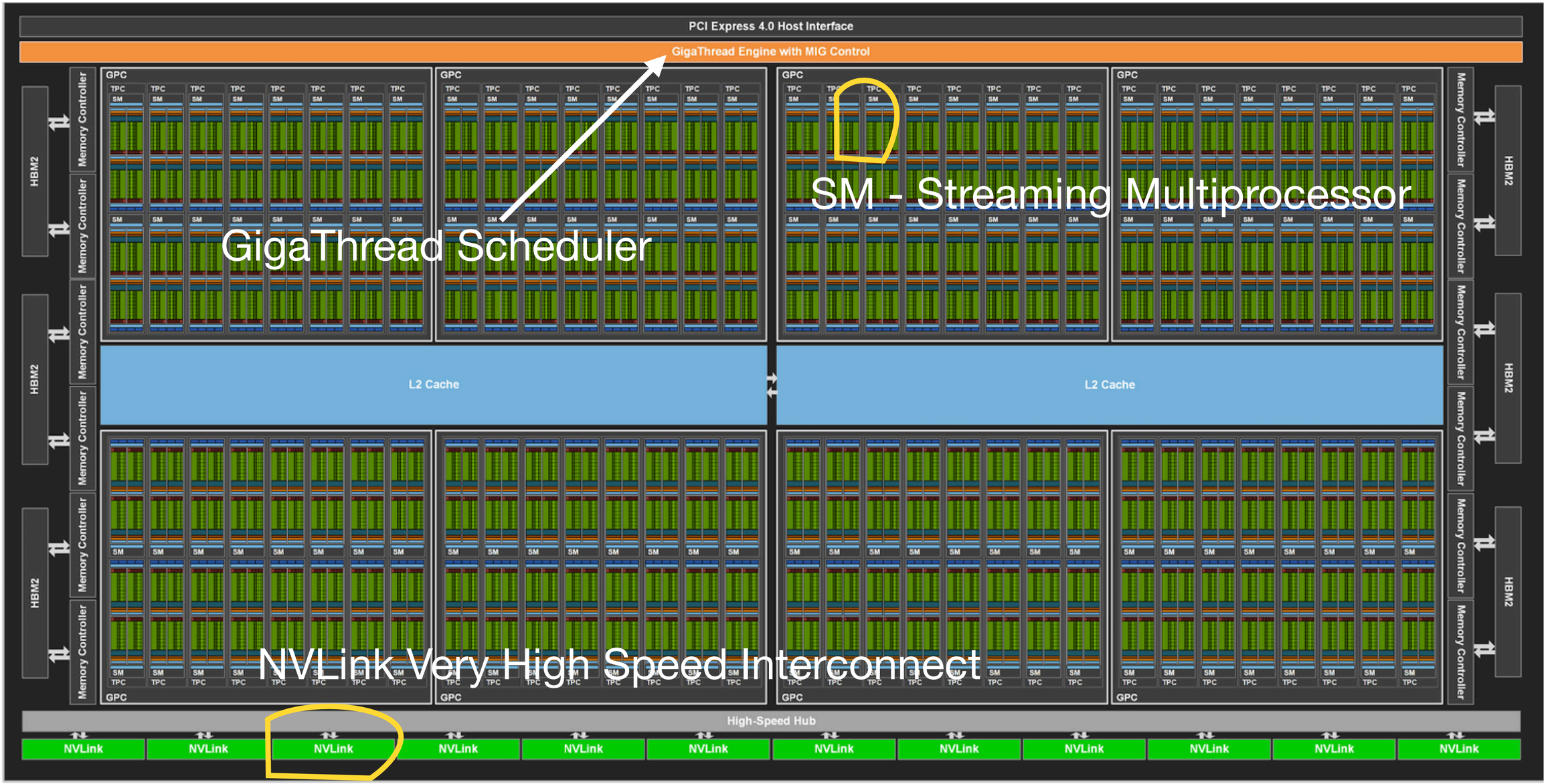
Graphical Processing Units (GPUs) & Computer Organisation



GPUs: Design Philosophy

- CPU's design philosophy: execute a **single task really fast**
  - Achieved through:
    - Large and layered caches
    - Complex and power hungry **Branch prediction** and **Out-of-order execution** unit
    - High clock frequency
- GPUs on the other hand: execute thousands of operations **really fast**
  - Achieved through:
    - **Larger number** (~16000) of **slower cores** (~1 Ghz)
    - Single Instruction Multiple Threads (**SIMT**) - **Lock step execution semantics**
    - Smaller caches and **primitive** control logic
    - But very high memory bandwidth (~ 1-3 TB/sec vs CPU's ~100GB/sec)

GPUs: An Architectural View

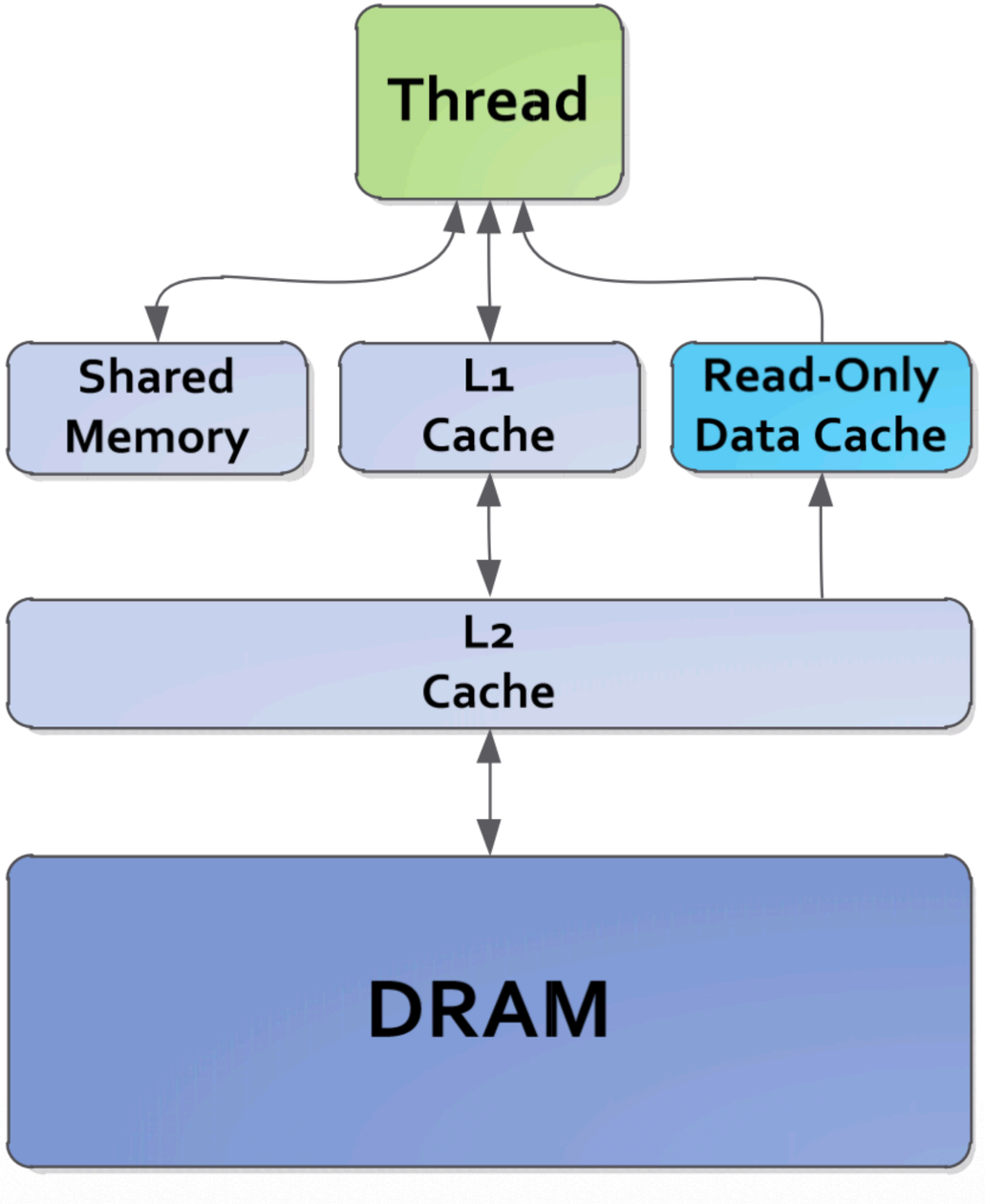


Streaming MultiProcessor (SM)

- In addition to INT32 and FP32/64 core, each SM has 4 **Tensor Cores**
  - **TC:** To speed up 4x4 matric operations (64 multiplications ..)
    - D = A x B + C
    - Multiplication and accumulation as one fused operation
- Dual-use 192 KB L1 Cache/Shared Memory
- A warp — collection of 32 threads that execute in **lock-step**

Memory Hierarchy

- L2 cache is shared among the GPCs
  - Typically quite large (~ 40MB)



Warp Scheduler

- A thread block is assigned to an SM
- Within a block there are many warps
- Warp scheduler picks a **ready warp** and issues an instruction for it
  - The **dispatcher unit** sends the warp's instruction to appropriate execution units
    - FP/INT pipelines, ld/st units, Tensor cores, etc.
  - How does the scheduler manage different paths for the same warp? **(Warp Divergence)**
    - Scheduler maintains an active mask:
      - half the threads are inactive

```
11111111111111110000000000000000
```

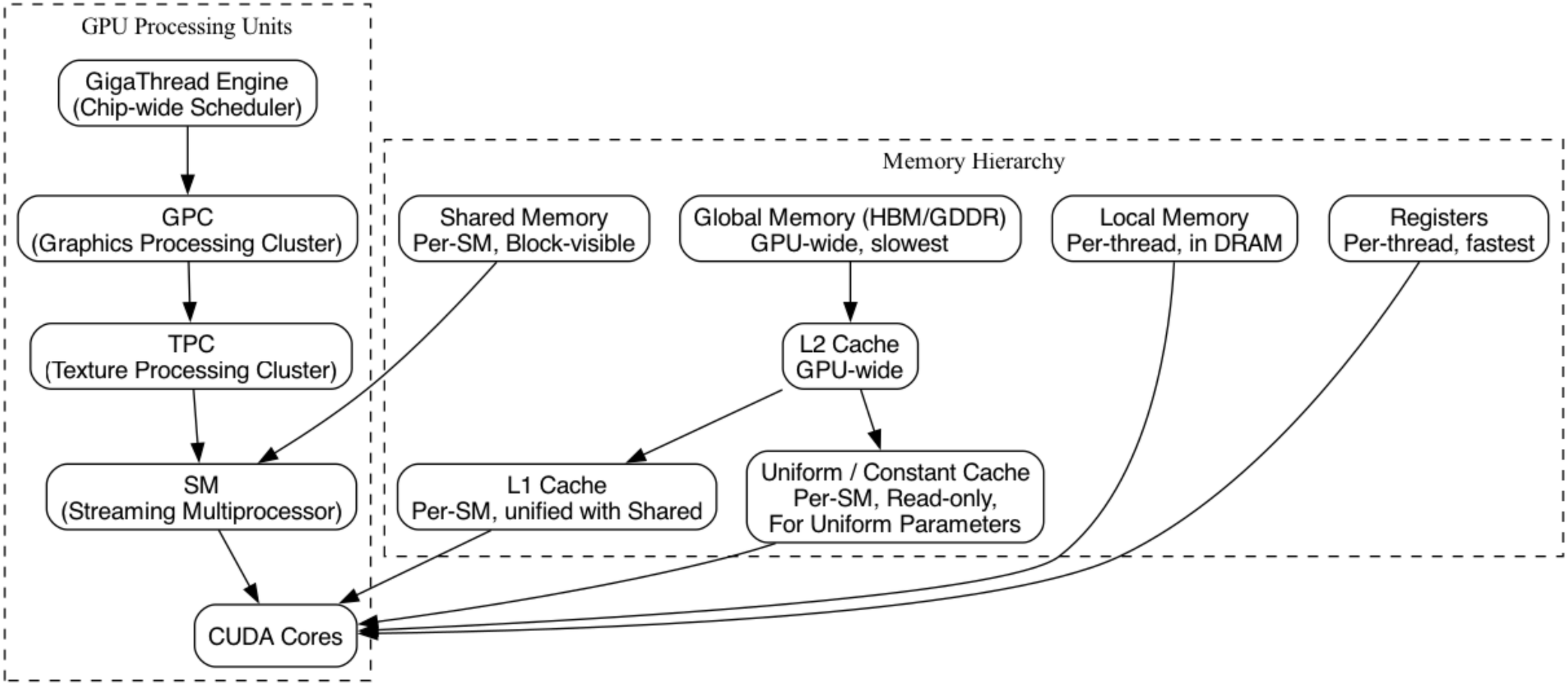
```
if (x > 0)
    A[i] = B[i] + C[i];
else
    A[i] = B[i] - C[i];
```

Warp Scheduler

- A thread block is assigned to an SM
- Within a block there are many warps
- Warp scheduler picks a **ready warp** and issues an instruction for it
  - The **dispatcher unit** sends the warp's instruction to appropriate execution units
    - FP/INT pipelines, ld/st units, Tensor cores, etc.
  - **Warp Divergence:** Compiler strategies to deal with it
    - Replace condition `if (cond) A = B + C;`
      - By `A = cond ? B + C : A;`

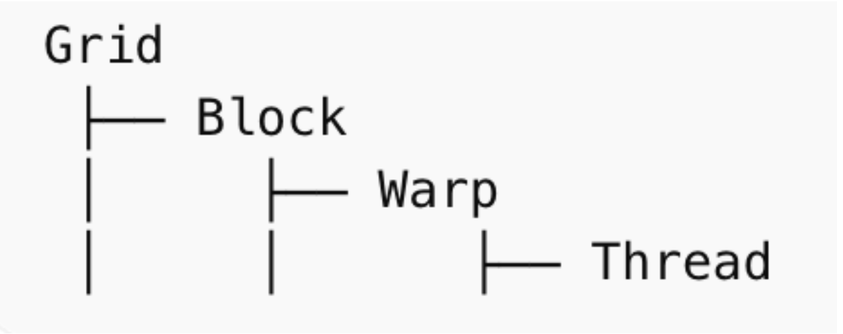
```
if (x > 0)
    A[i] = B[i] + C[i];
else
    A[i] = B[i] - C[i];
```

Putting it together



Thread Hierarchy

- **Thread** is the smallest execution unit
  - Each thread perform a computation
    - Eg: $C[i] = A[i] + B[i]$
- A **Warp** is a group of 32 threads that **move simultaneously** in each cycle executing the **same instruction**
  - This is model of computation is called **SIMT**
- A **Block** contains many warps
  - Threads in a block can synchronise through **shared memory**




```
Warp 0:
Thread 0 → A[0] + B[0]
Thread 1 → A[1] + B[1]
Thread 2 → A[2] + B[2]
...
Thread 31 → A[31] + B[31]
```

GPU Model of Computation

- Programs runs partly on **Host** (CPU), and partly on **Device** (GPU)
- Host orchestrates the computation
  - Meaning — send computation task and data to the GPU
- Device executes tasks specified as **kernels** in parallel
  - In parallel Kernel execution — > Through non-blocking launches (using **Streams)**
  - **Example Kernel & its launch code**
- Threads smallest unit of execution
- Threads are organised in TBs
  - TB executes on a single SM

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

```
addVectors<<<gridSize, blockSize>>>(A,B,C);
```

Memory Model

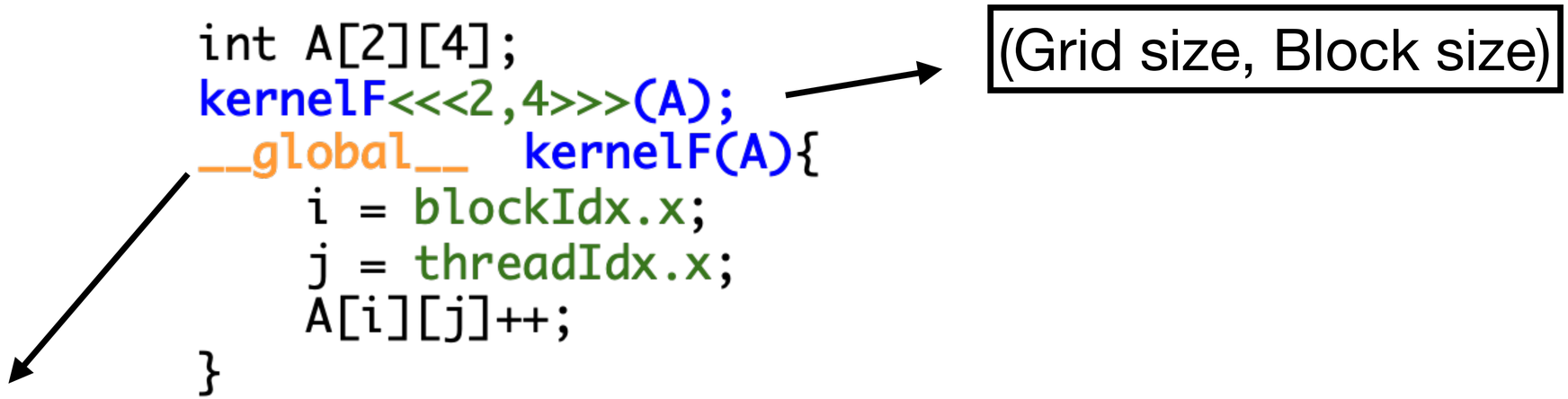
- Scope:
  - Registers — > Per thread
  - Shared Memory —> per block
  - Constant Memory —> ?
  - Global Memory —> whole device
  - …
  -

Memory Consistency Model

- Visibility: (Weakly consistent — Relaxed ordering)
  - Thread always sees its own writes in order
  - In Block scope, after synchronisation, all writes to shared memory become visible to all other threads in the **same block**
  - Device scope: Writes to global memory are **eventually visible (**after a fence or a synchronisation)
- Memory Fences: TBD (later)
- Coherence:
  - L1 caches are **not kept coherent** with each other
  - L2 cache is coherent across the GPU

CUDA Basics

- **Kernel** - The unit of computation for a GPC



Device Qualifier: On device but called from the host

CUDA Basics

Device Qualifiers

- **__device__ :** Called from the device and executes in the device
  - Used for helper functions
- **__host__:** Called from and executes on the host
  - Standard C++ functions
- **__global__:** On device called from the host
  - Defines a kernel