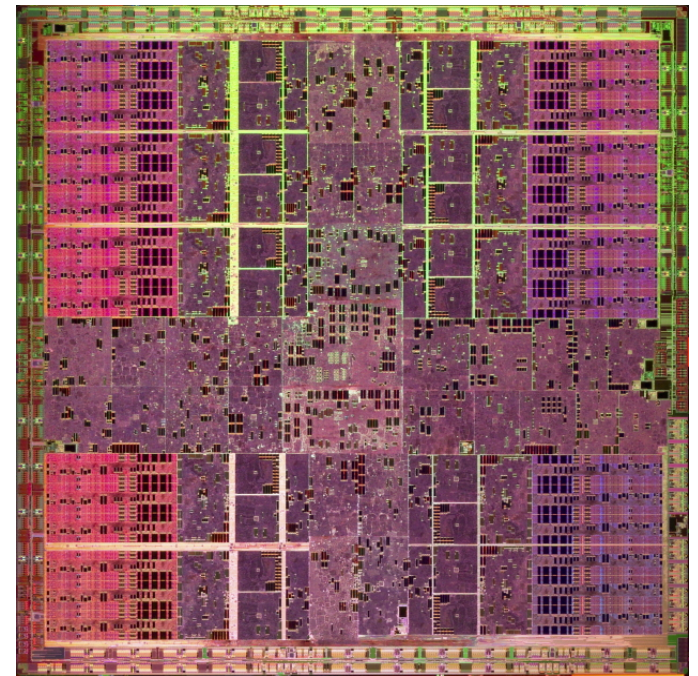
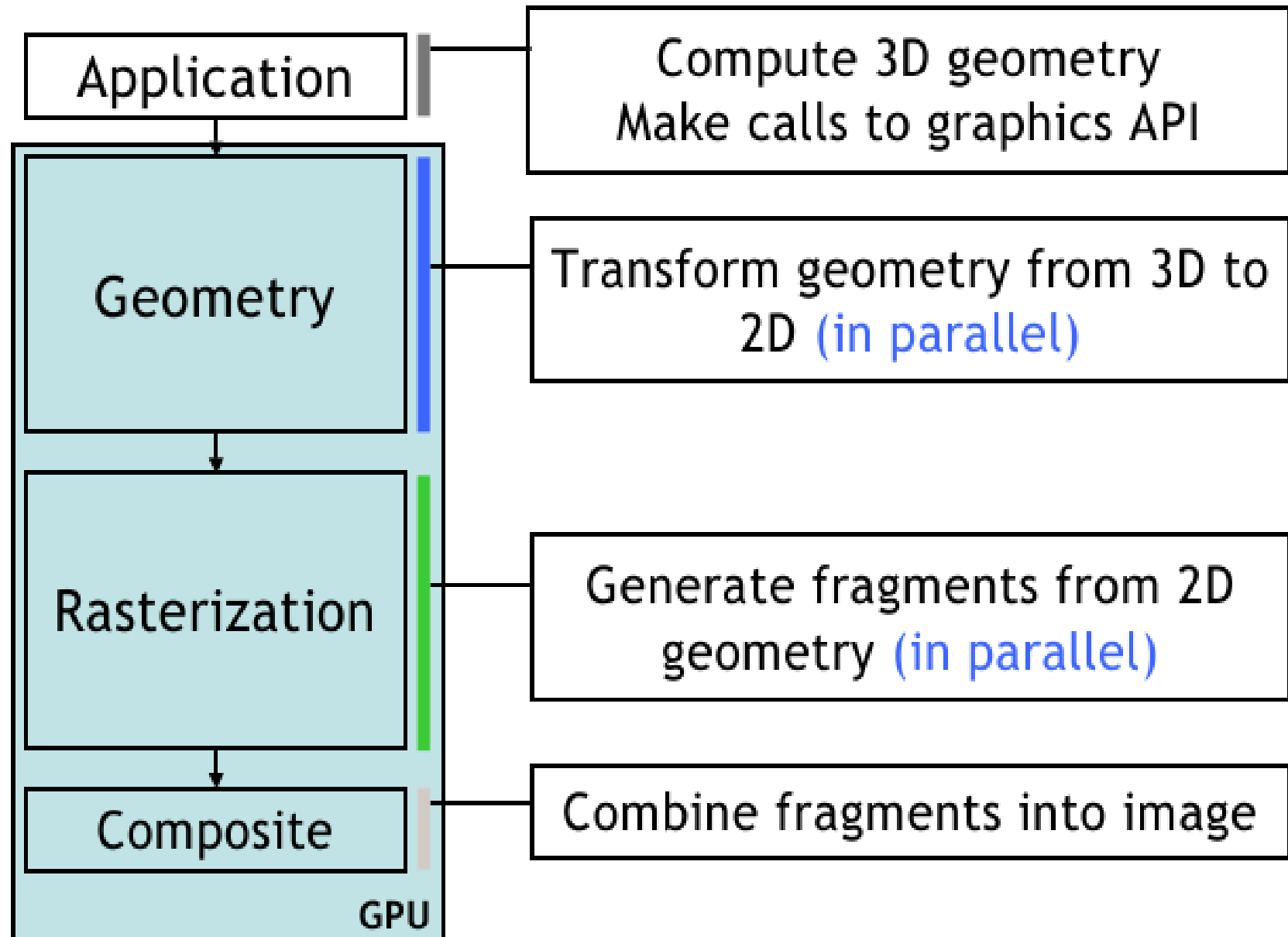


A brief introduction to *Nvidia CUDA*

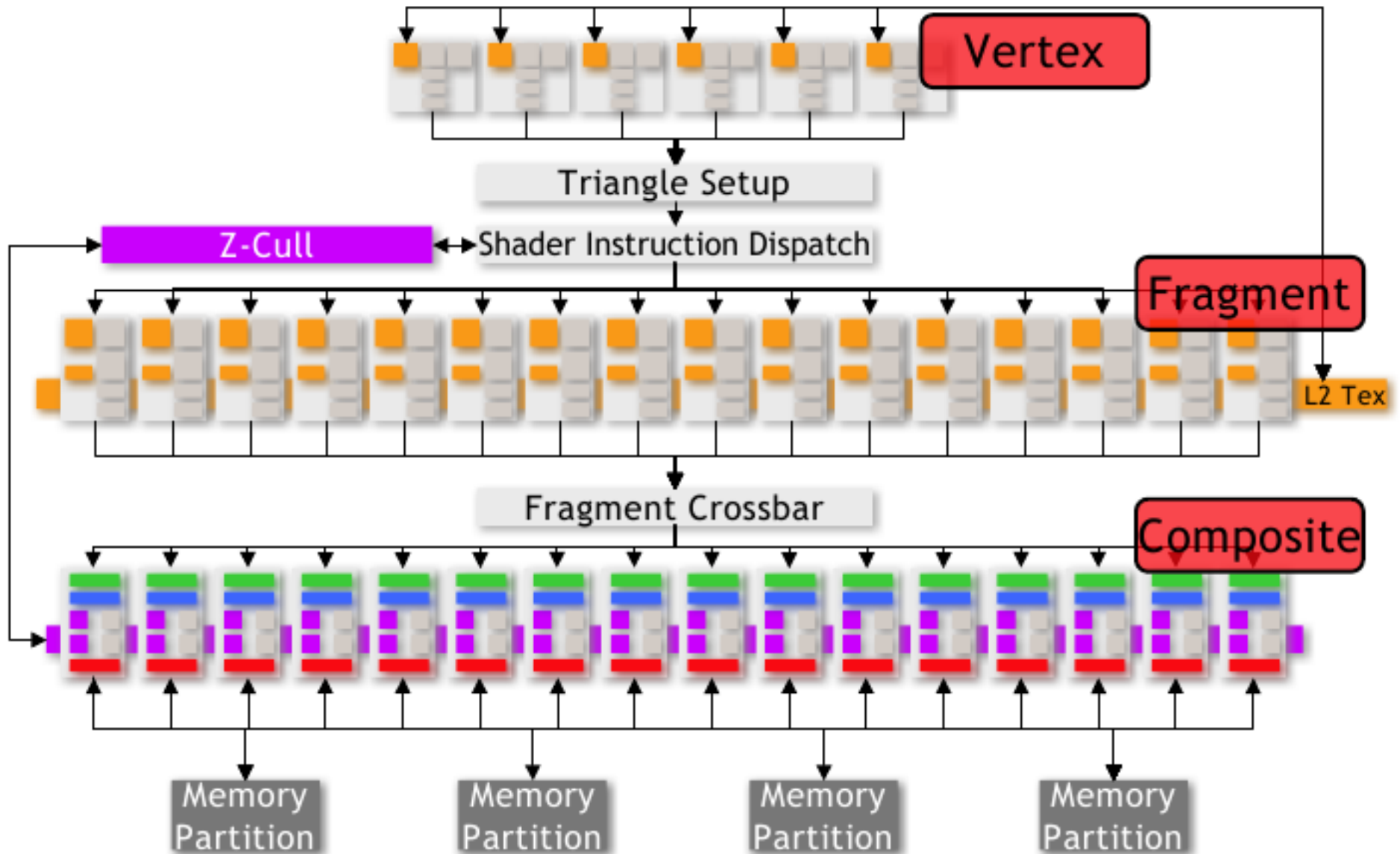


Andrea Di Biagio
dibiagio@elet.polimi.it

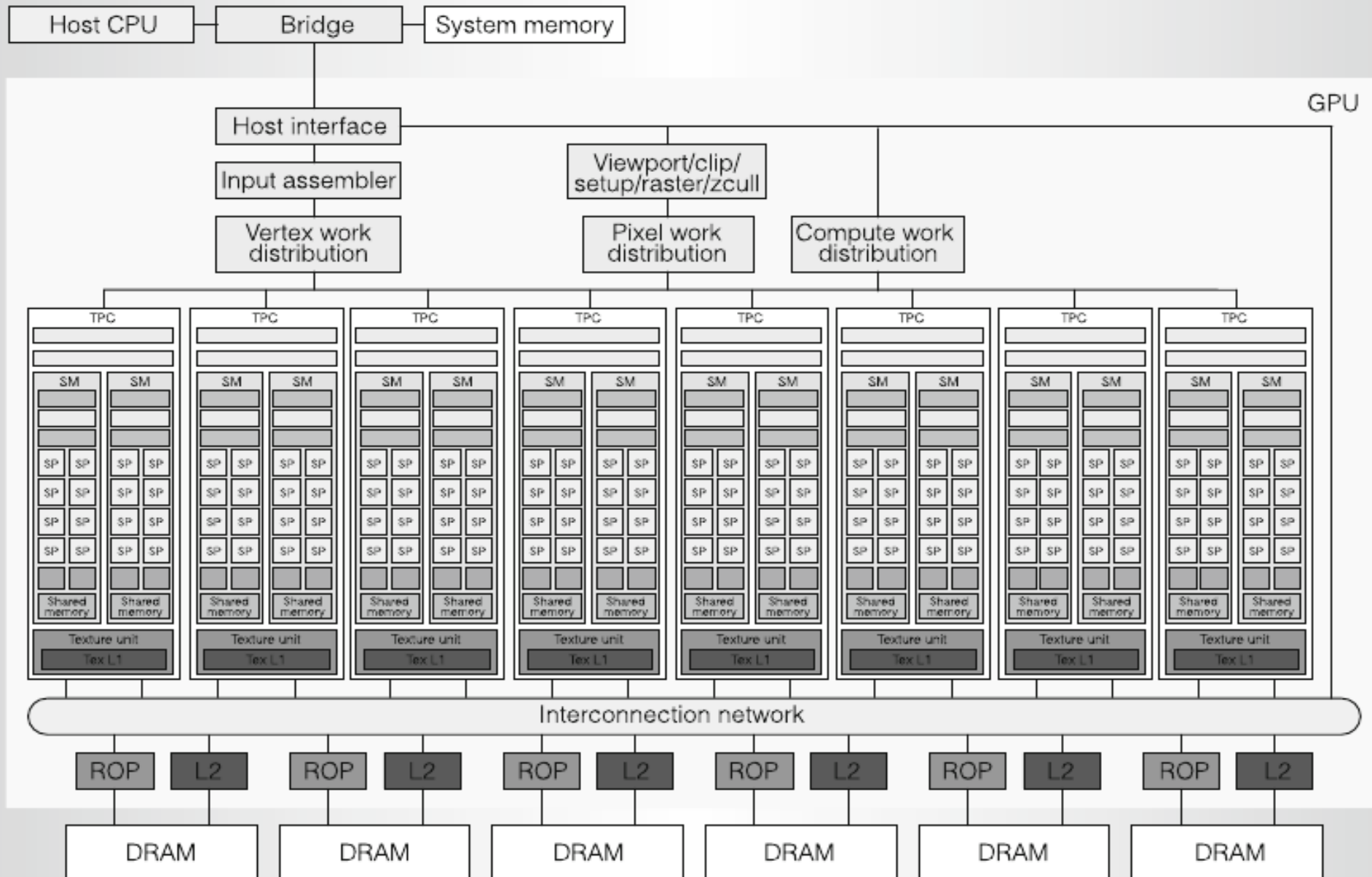
Rendering Pipeline

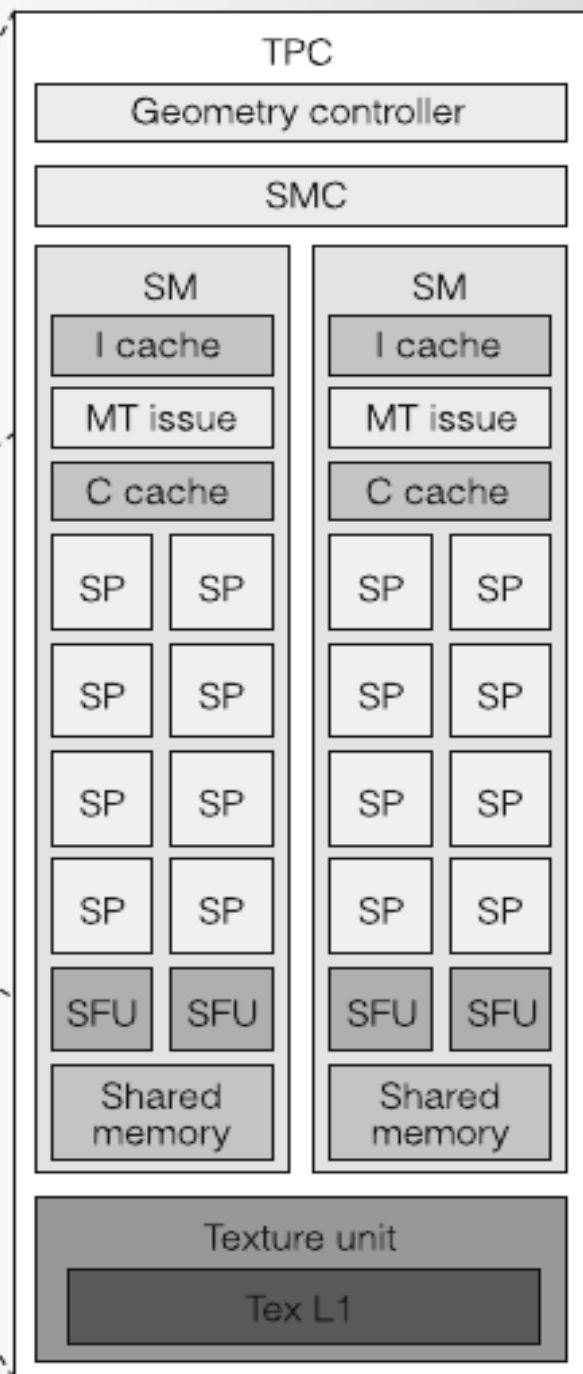
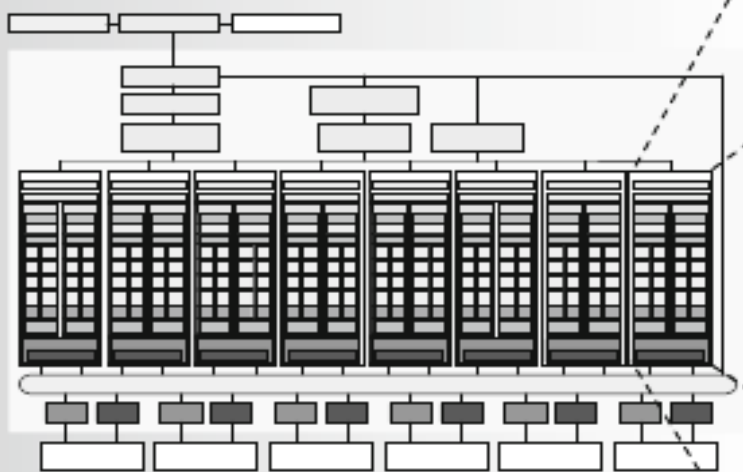


NVidia *GeForce 6800*



NVidia GeForce 8800 GT





CUDA

- CUDA is a scalable parallel programming model and a software environment for parallel computing
 - Expose the computational power of NVIDIA GPUs
 - Enable GPU computing
- Minimal extensions to familiar C/C++ environment
- Heterogeneous serial-parallel programming model
- NVIDIA's TESLA architecture accelerates CUDA

CUDA - Some design goals

- Scale to 100's of cores, 1000's of parallel threads
- Let programmers focus on parallel algorithms
- Enable heterogeneous systems
 - CPU + GPU
- CPU and GPU are separate devices with separate DRAMs

CUDA - kernels and threads

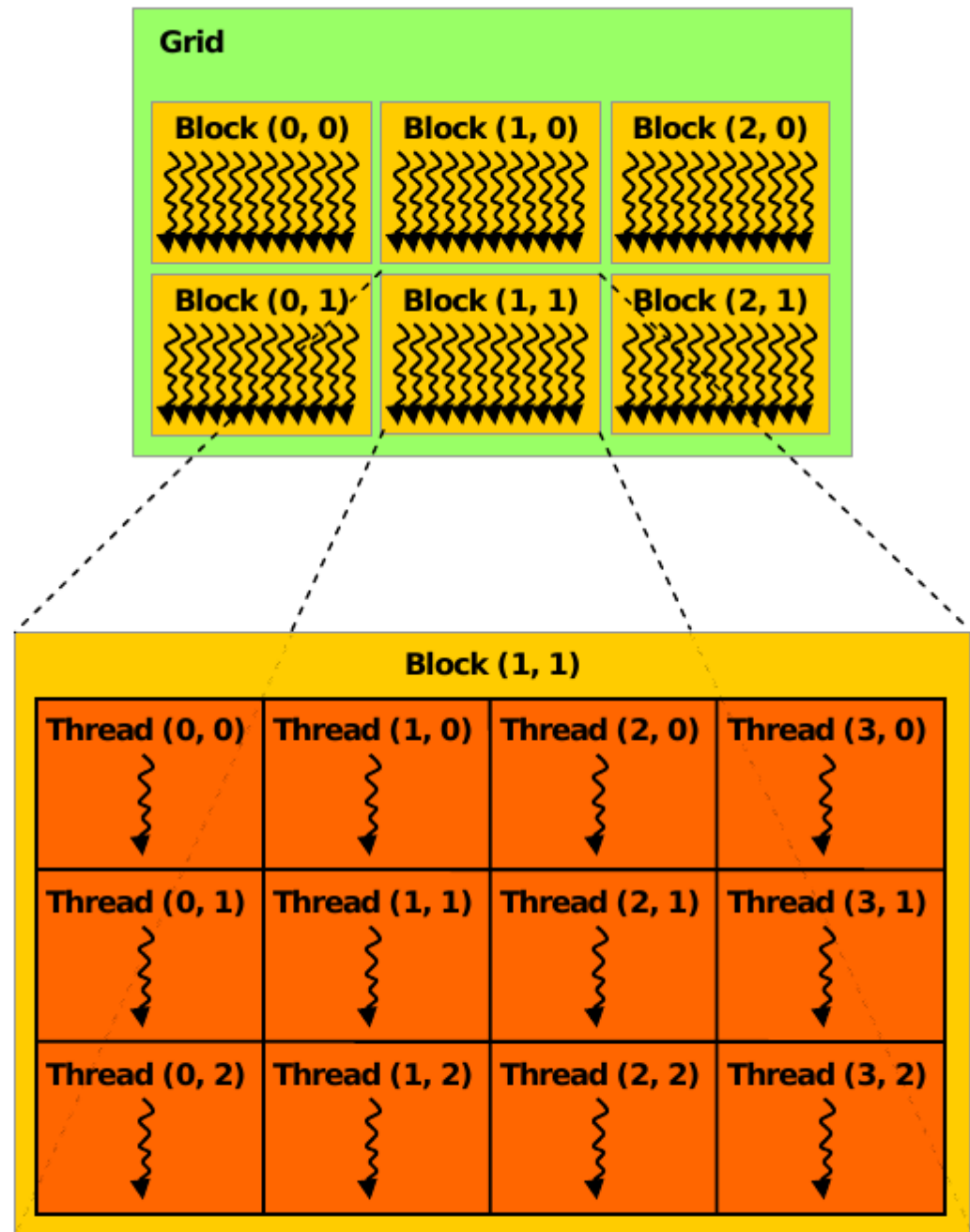
- Parallel portions of an application are executed on the device as kernels.
 - One kernel is executed at a time
 - Many threads execute each kernel
- Differences between CUDA and CPU threads
 - CUDA threads are extremely lightweight
 - Very little creation overhead
 - Instant switching
- CUDA uses 1000s of threads to achieve efficiency while Multi-core CPUs can use only a few

Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
 - All threads run the same code
 - Each thread has an ID that it uses to compute memory addresses and make control decisions
- Cooperation between threads is achieved via shared memory.
 - A group of threads can share data through a very little shared memory.
 - Synchronization mechanisms are needed in order to grant coherency in memory accesses

KERNEL EXECUTION

- A Kernel launches a grid of thread blocks
- Threads within a block cooperate via shared memory
- Threads in different blocks cannot cooperate



CUDA - Memory Model

A **SIMT** execution model

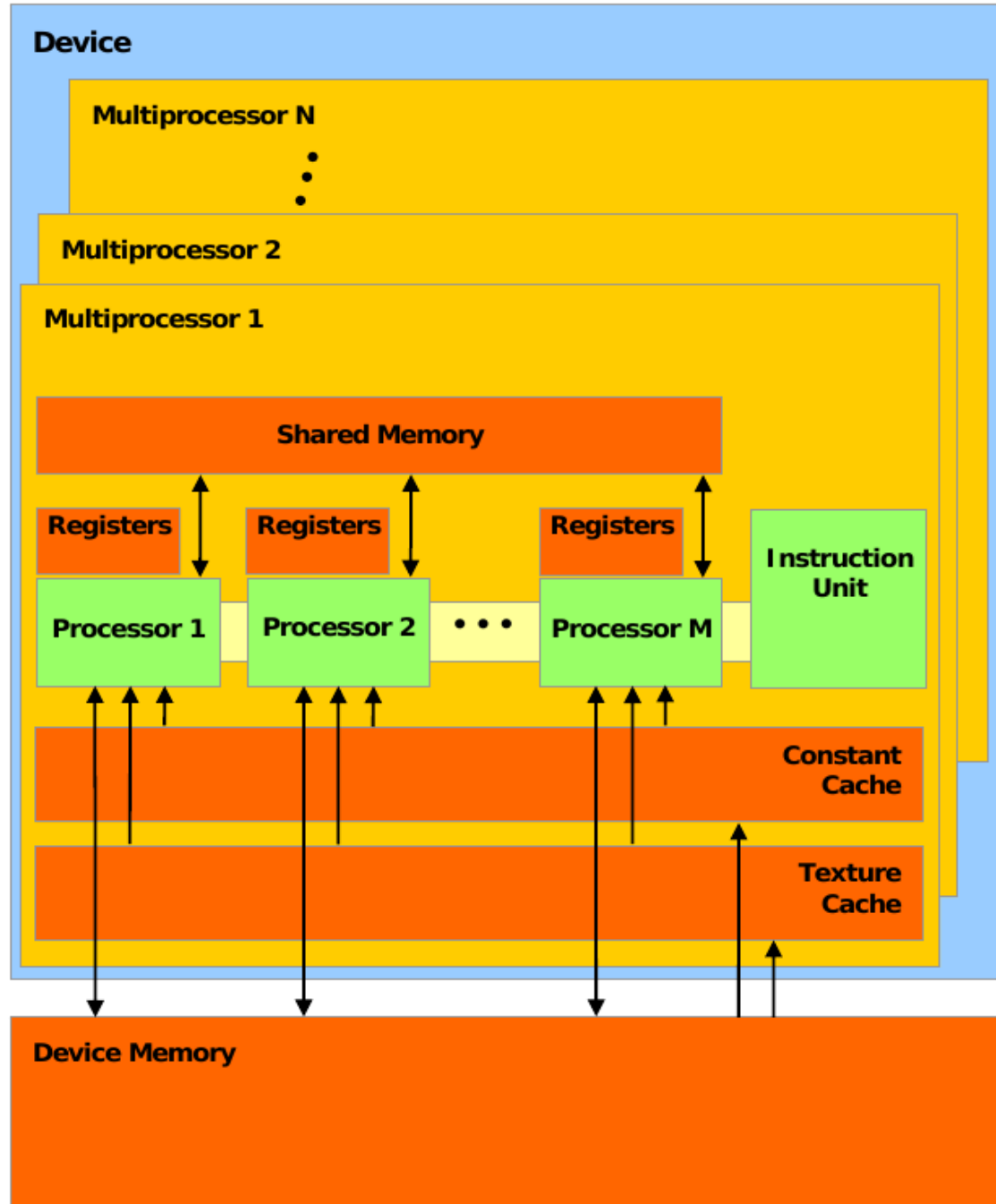
Every two GPU's clock cycles a batch of thread (called warp) is scheduled for execution.

Similar to a SIMD execution, however the control flow may diverge for some threads in a warp.

Threads in a warp can be active or not

A thread can access data located in several memory spaces

Depending on the memory, access times can be faster or slower



Increment array example

CPU program

```
void inc_cpu(int *a, int N)
{
    int idx;


    for (idx = 0; idx < N; idx++)
        a[idx] = a[idx] + 1;
}
```

```
int main()
{
    ...
    inc_cpu(a, N);
}
```

CUDA program

```
__global__ void inc_gpu(int *a, int N)
{
    int idx = blockIdx.x * blockDim.x
            + threadIdx.x;

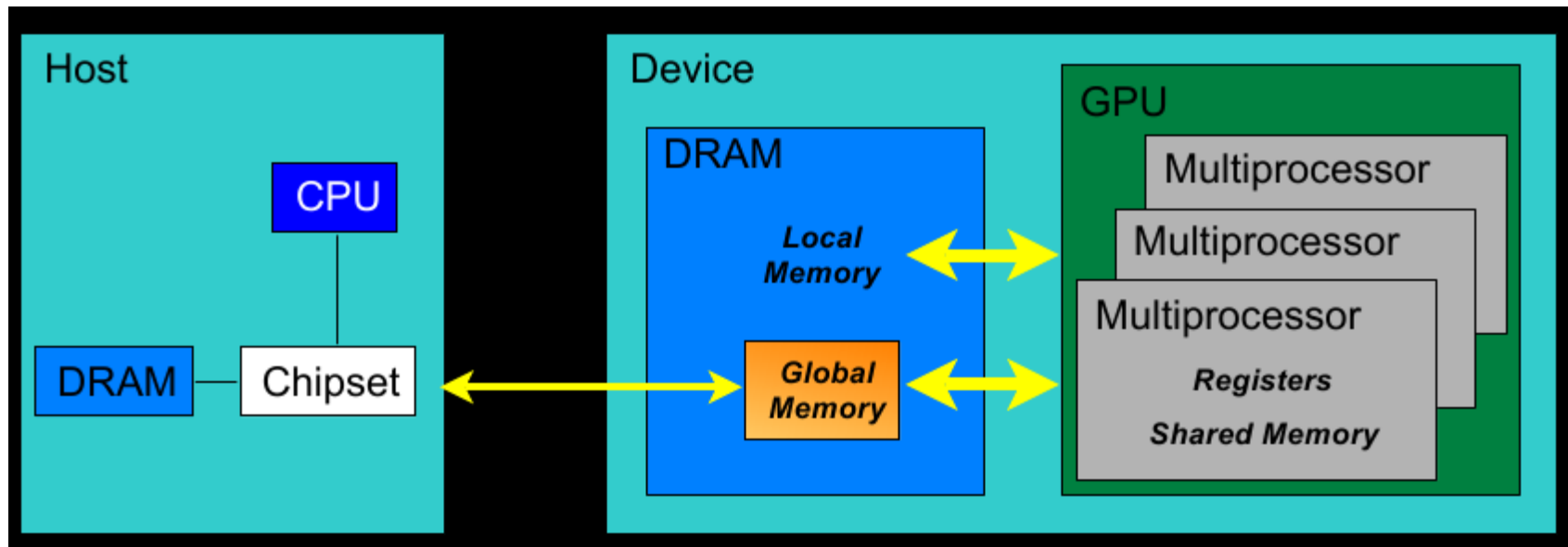
    if (idx < N)
        a[idx] = a[idx] + 1;
}
```



```
int main()
{
    ...
    dim3 dimBlock (blocksize);
    dim3 dimGrid( ceil( N / (float)blocksize) );
    inc_gpu<<<dimGrid, dimBlock>>>(a, N);
}
```

Managing Memory

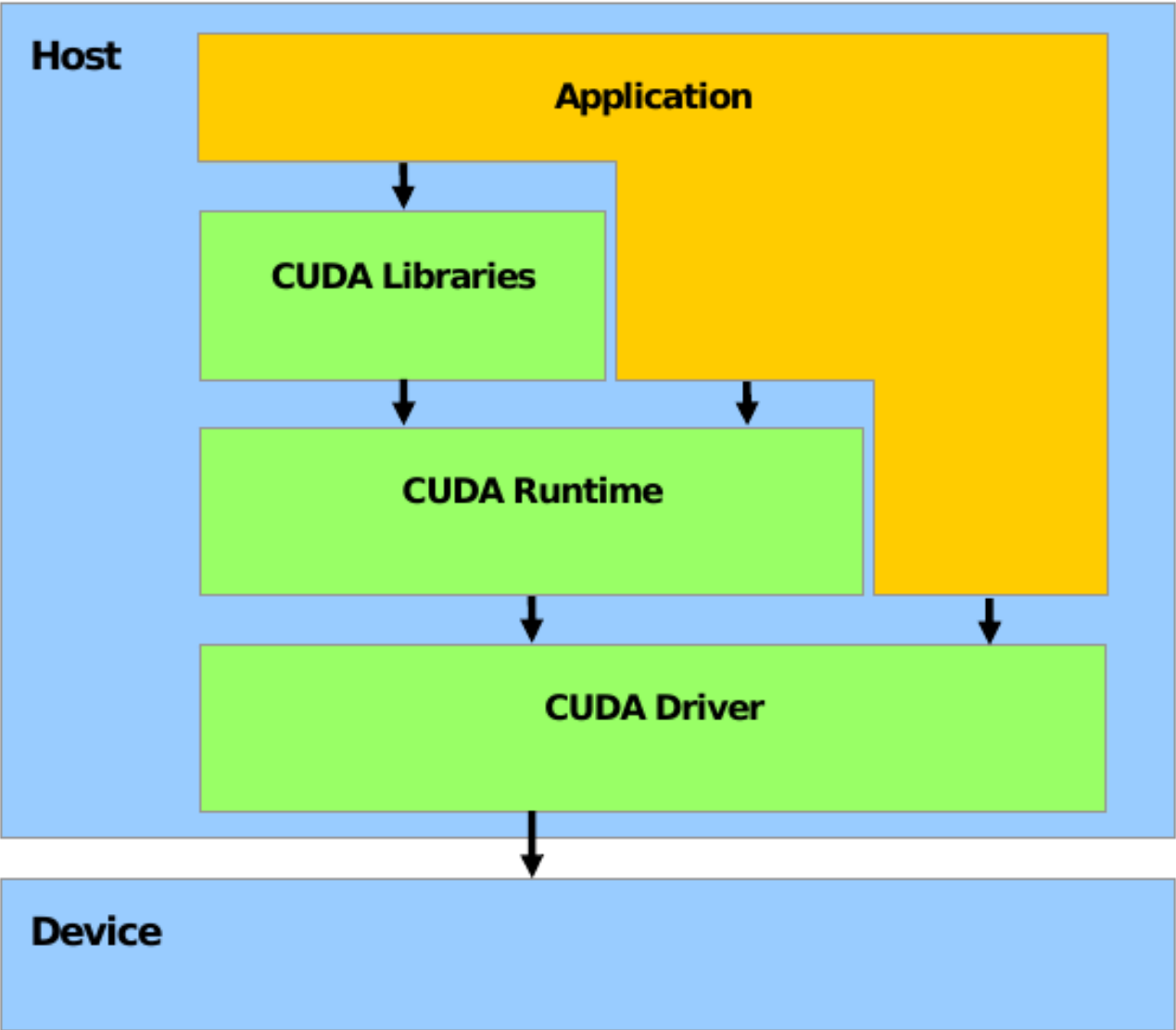
- CPU (Host) and GPU (Device) have separate memory spaces.
- Communication is achieved via bus transfers from the host memory to the device memory and vice versa.



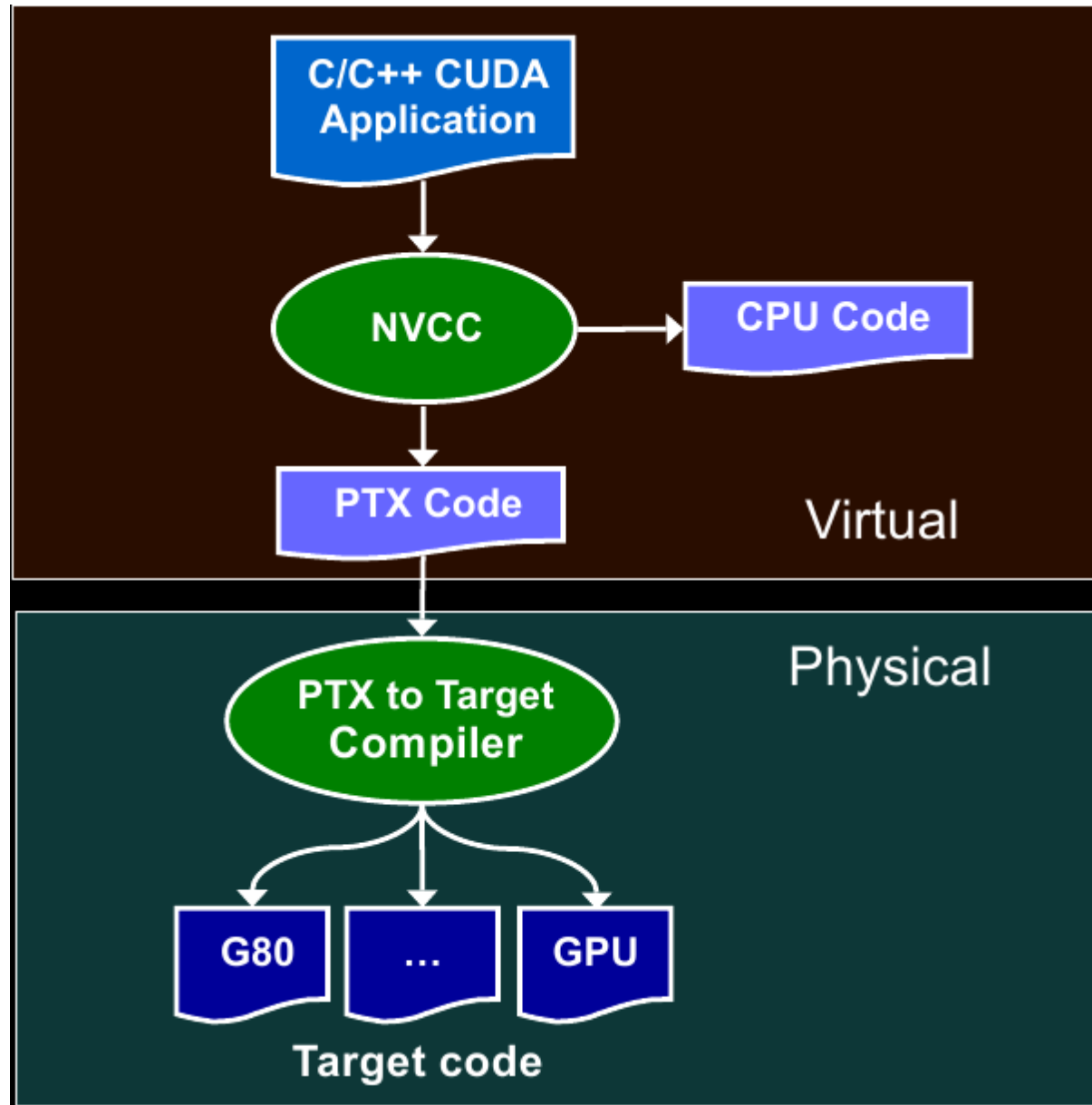
Synchronization

- From CPU code
 - A process can wait for the termination of a device kernel computation
 - `memcpy` operations from host to device are synchronized by default.
- From device code
 - The builtin function `__syncthreads ()` synchronizes all threads in a block
 - Useful when threads are concurrently accessing to a shared location in memory

CUDA - Framework Overview



CUDA - compilation process



Website



CUDA WEBSITE -- <http://www.nvidia.com/cuda>