GPU Programming with CUDA

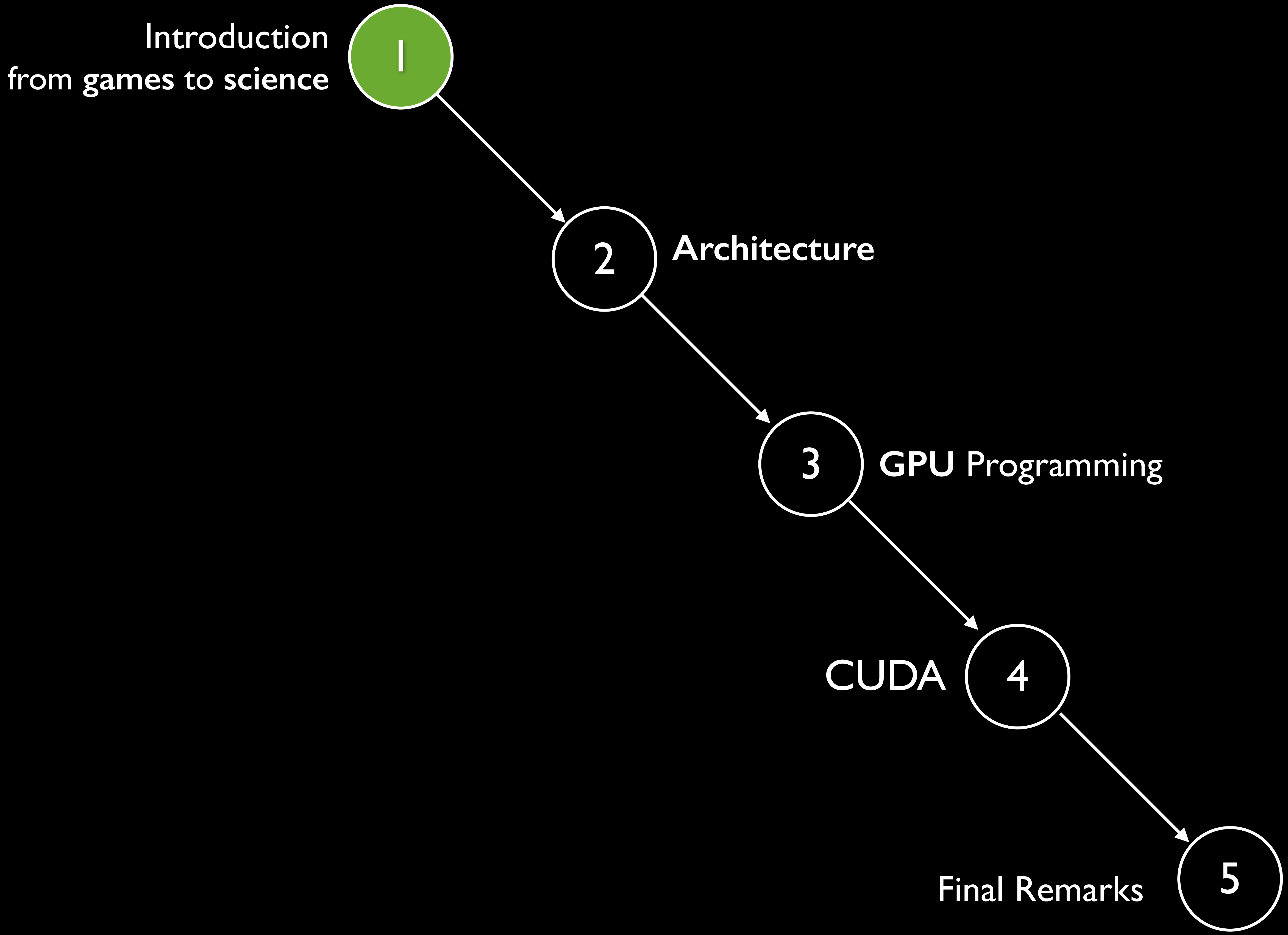
Pedro Velho

Meeting the audience!

How many of you used concurrent programming before?

How many threads?

How many already used CUDA?



Introduction
from **games** to **science**

1

2 **Architecture**

3 **GPU** Programming

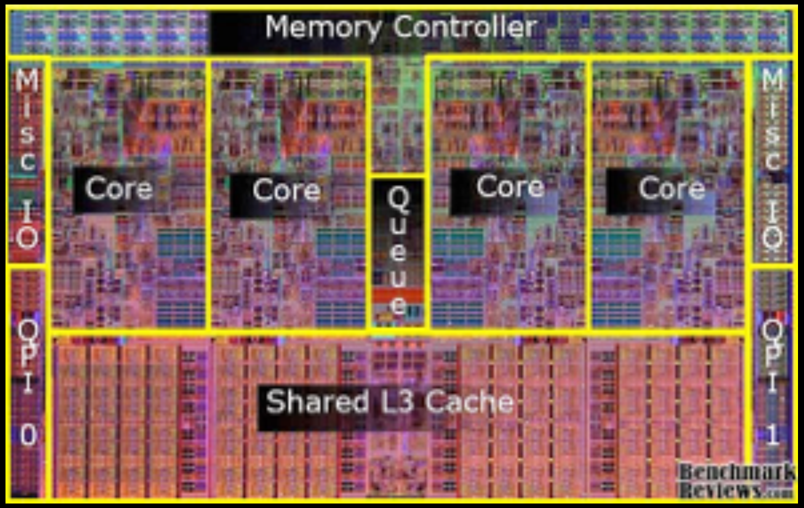
CUDA 4

Final Remarks 5

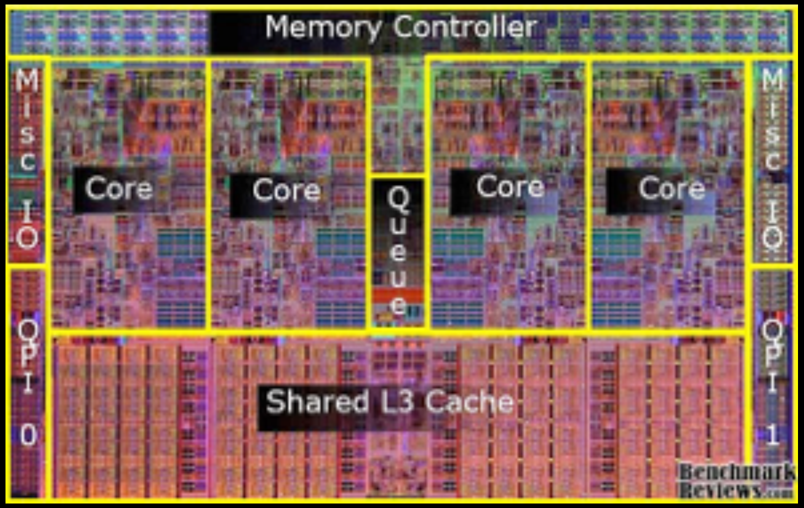
CPU vs GPU

CPU vs GPU

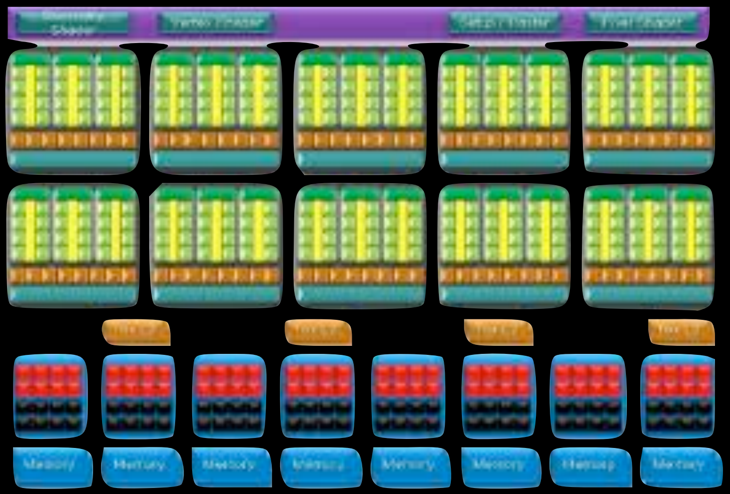
- A few general purpose cores
- Big cache memory
- Eg.: Nehalem i7 quad-core
  - 4 cores (8 threads)
  - Cache is about 50% of die area



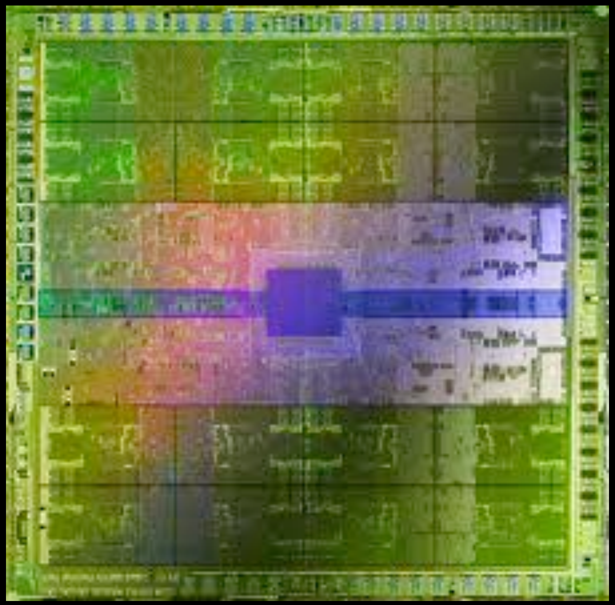
- A few general purpose cores
- Big cache memory
- Eg.: Nehalem i7 quad-core
  - 4 cores (8 threads)
  - Cache is about 50% of die area



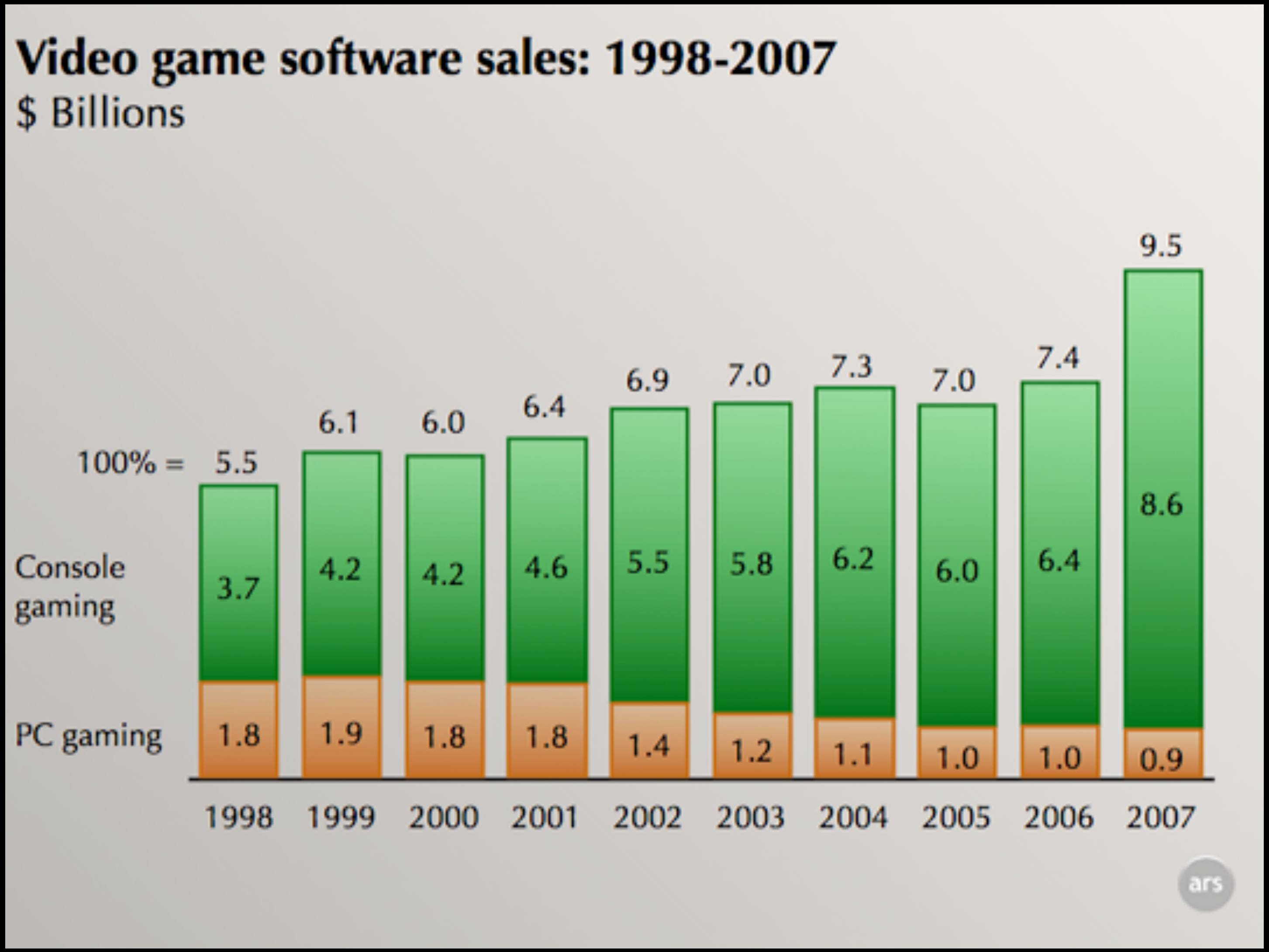
CPU vs GPU



- Design goal massively parallel graphics
- A lot of replicated functional units
- Small cache size
- Eg.: NVIDIA GTX280
  - 240 SP (streaming processors)
  - support for 30720 simultaneous threads

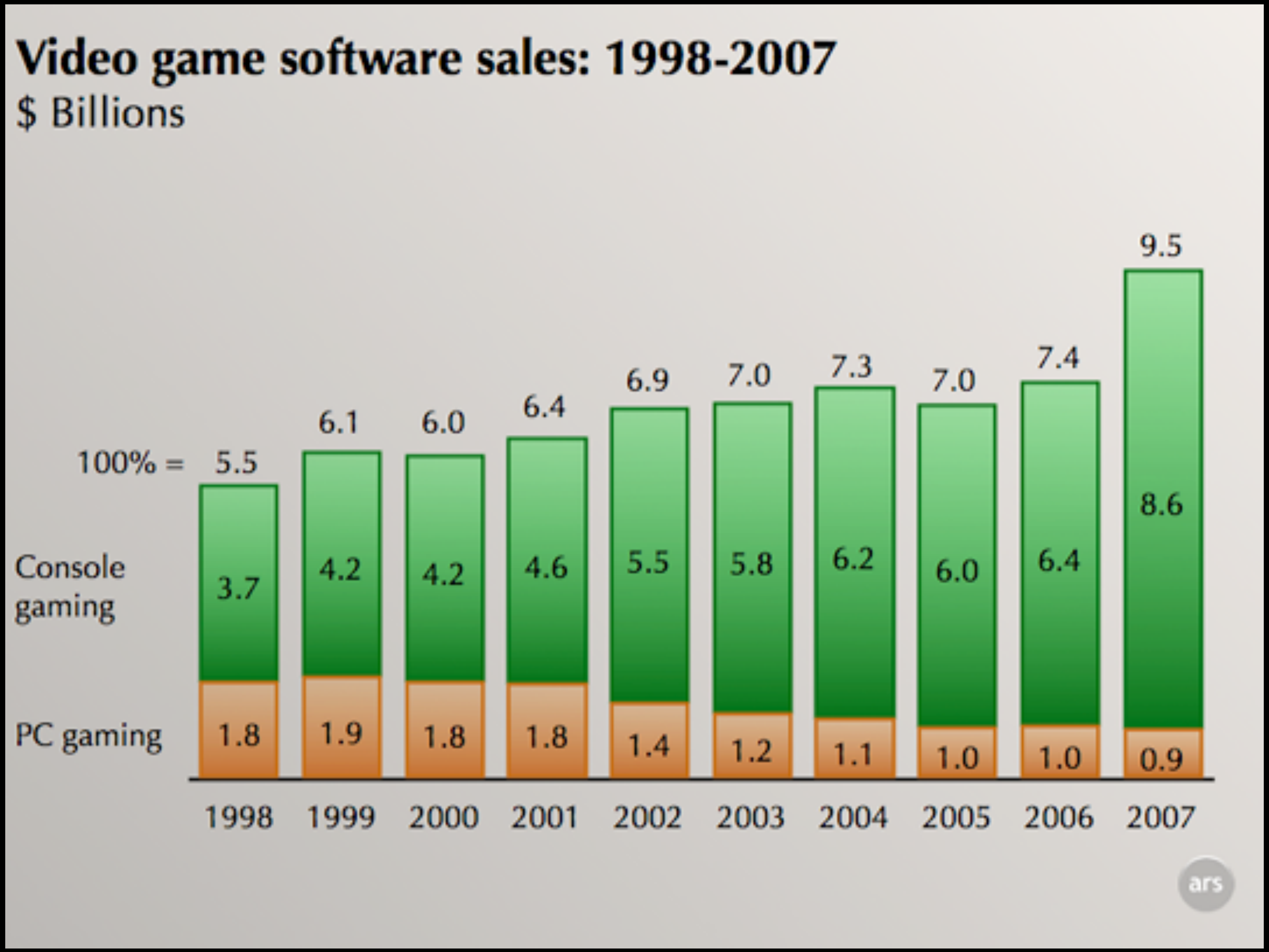


Computer Graphics is a Computational intensive application



Computer Graphics is a Computational intensive application

A lot of $$$ from game industry

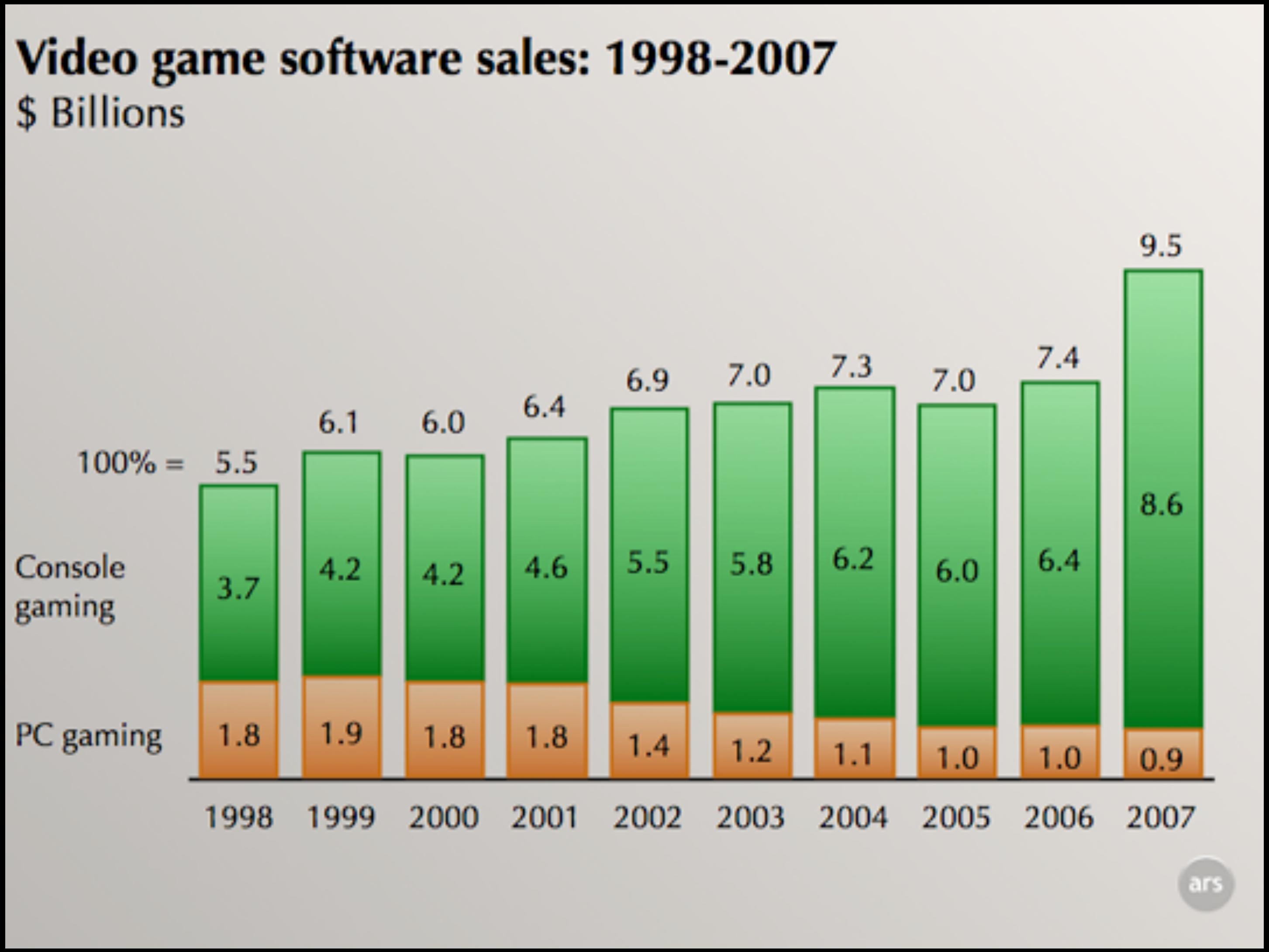


Computer Graphics is a Computational intensive application

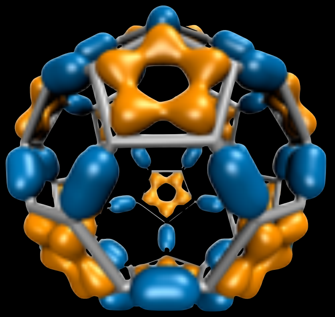
A lot of $$$ from game industry

Expressive gain in performance for parallel graphics rendering

Caught attention from the scientific community

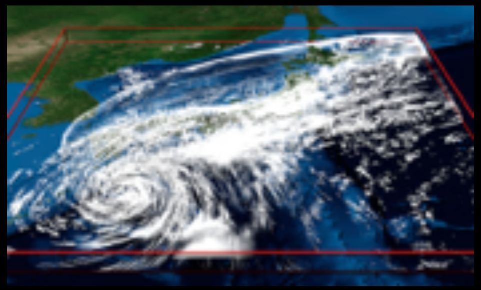


GPU is also adapted to several scientific applications



Molecular Biology

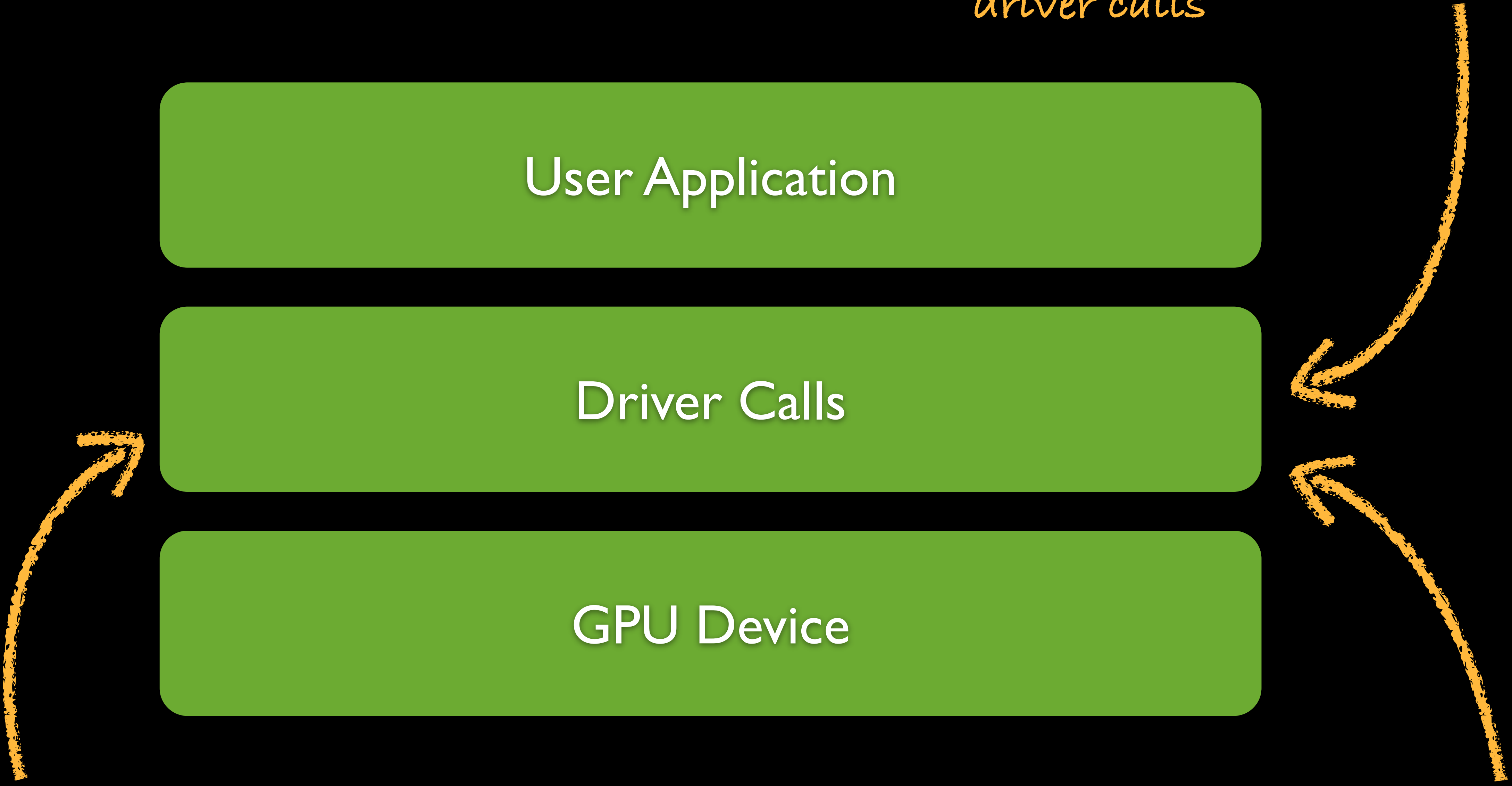
Fluid Simulation



Weather Forecast

GPGPU

Model the application directly using Computing Graphics driver calls



User Application

Driver Calls

GPU Device

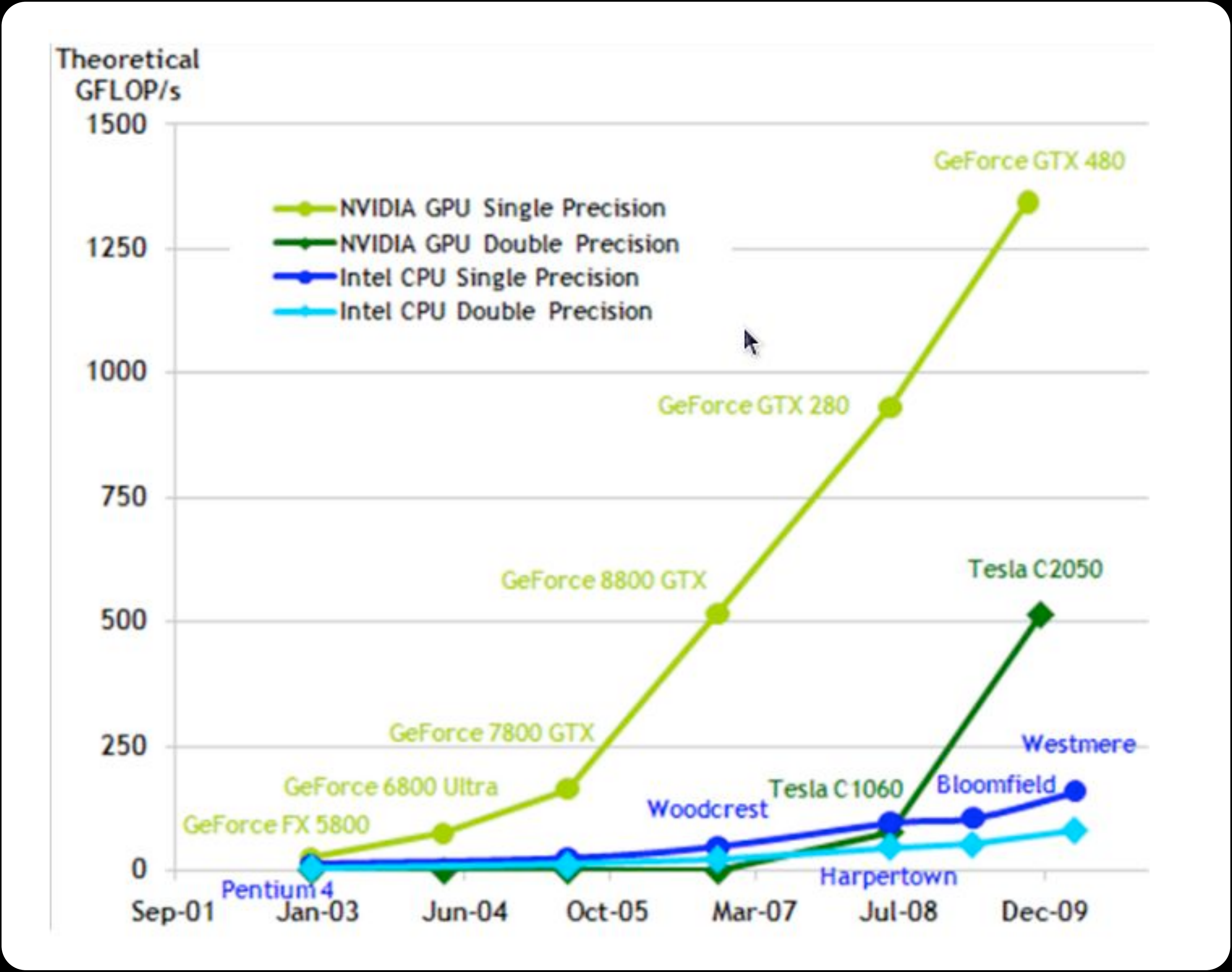
Need to port the problem to a complete different domain

DirectX or OpenGL are not easy to figure out

Potential Gain in Performance

100 times faster!

CPU vs GPU



Theoretical GFLOP/s
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2050
Tesla C1060
Westmere
Bloomfield
Woodcrest
Harpertown
Pentium 4
Sep-01
Jan-03
Jun-04
Oct-05
Mar-07
Jul-08
Dec-09

Potential Gain in Performance

100 times faster!

Several guys from Intel

This is a myth!

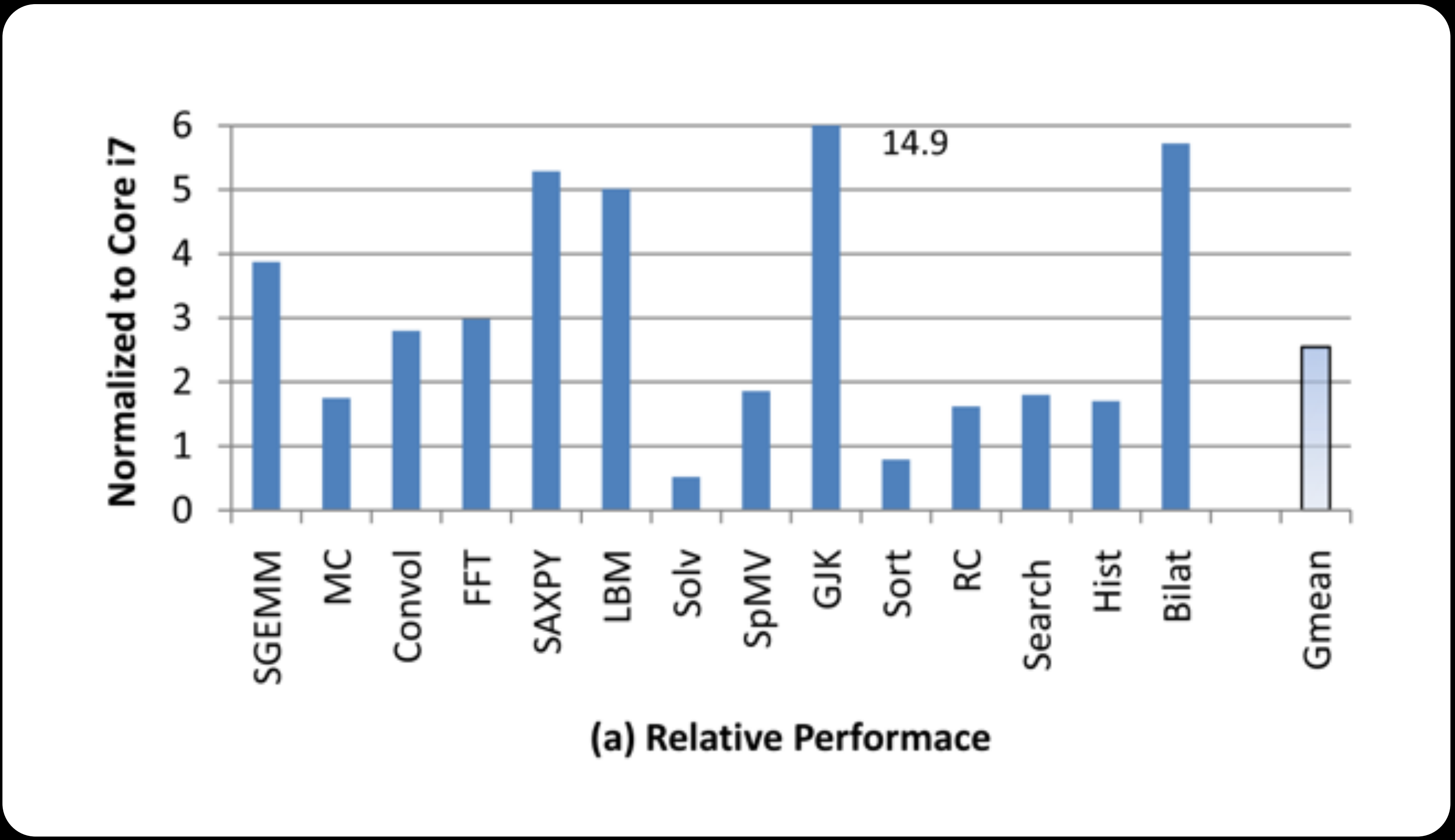
Victor W Lee *et. al.*, Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU

Core i7 - quadcore
vs.
GTX280

14 kernels

relative performance!

Reason:
Rethink your problem is challenging



(a) Relative Performace

Potential Gain in Performance

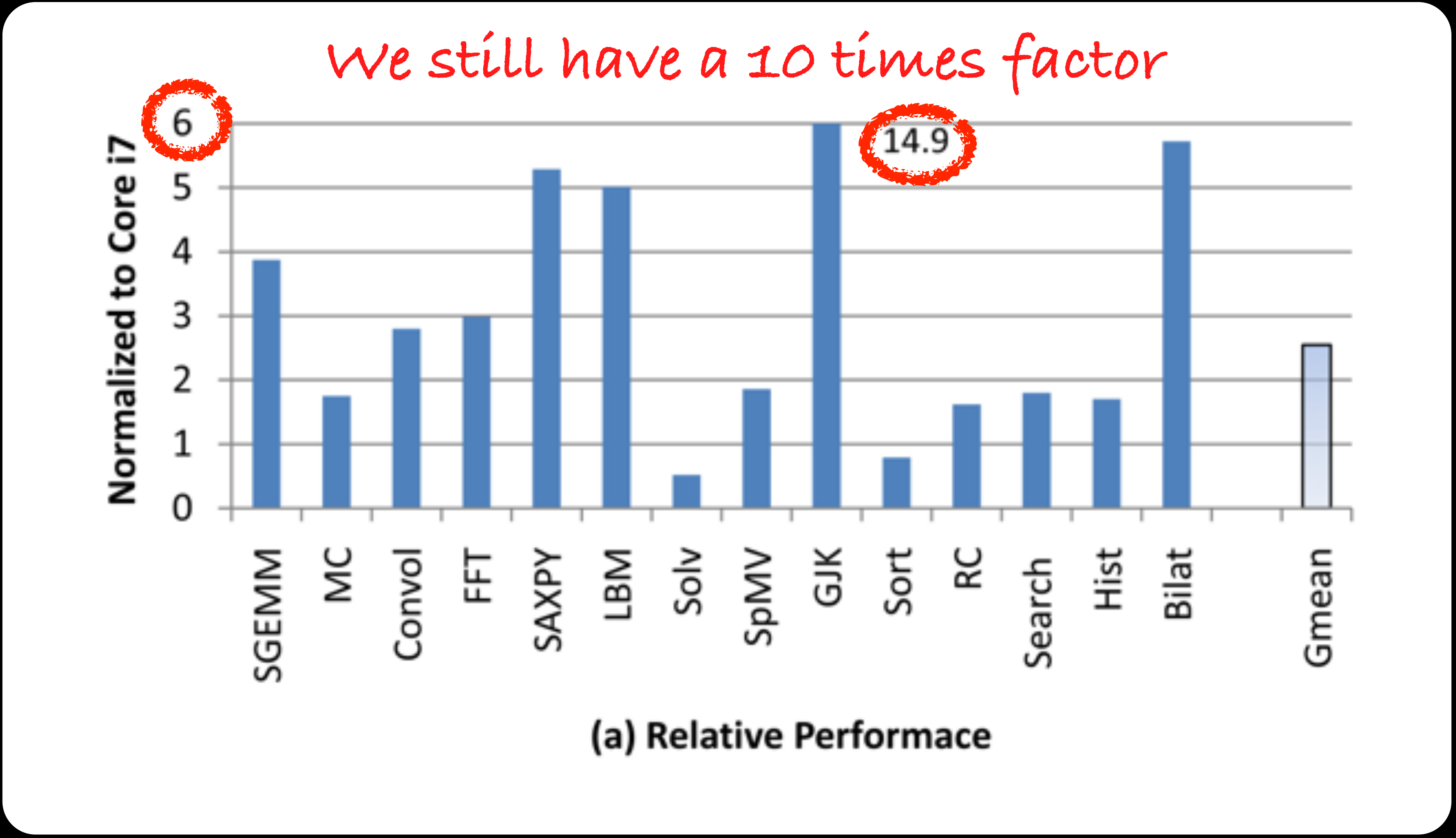
100 times faster!

Several guys from Intel

This is a myth!

Victor W Lee *et. al.*, Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU

$$\frac{T_{\text{GTX280}}}{T_{\text{Core i7}}}$$



We still have a 10 times factor

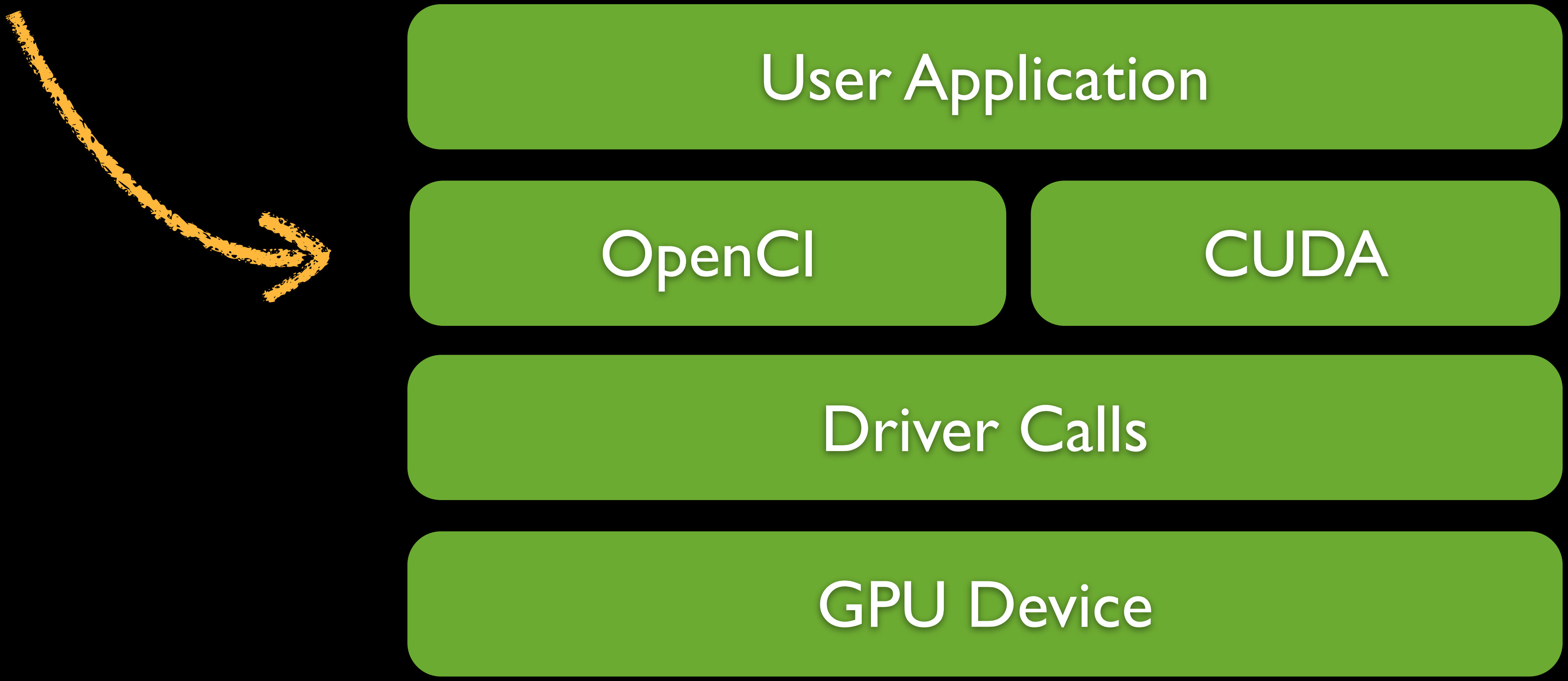
(a) Relative Performace

Substantial gain in execution time (10x)!

before GPU	**with GPU**
one year	one month plus a week
one day	two hours and twinty four minutes
one hour	six minutes

GPU Programming today

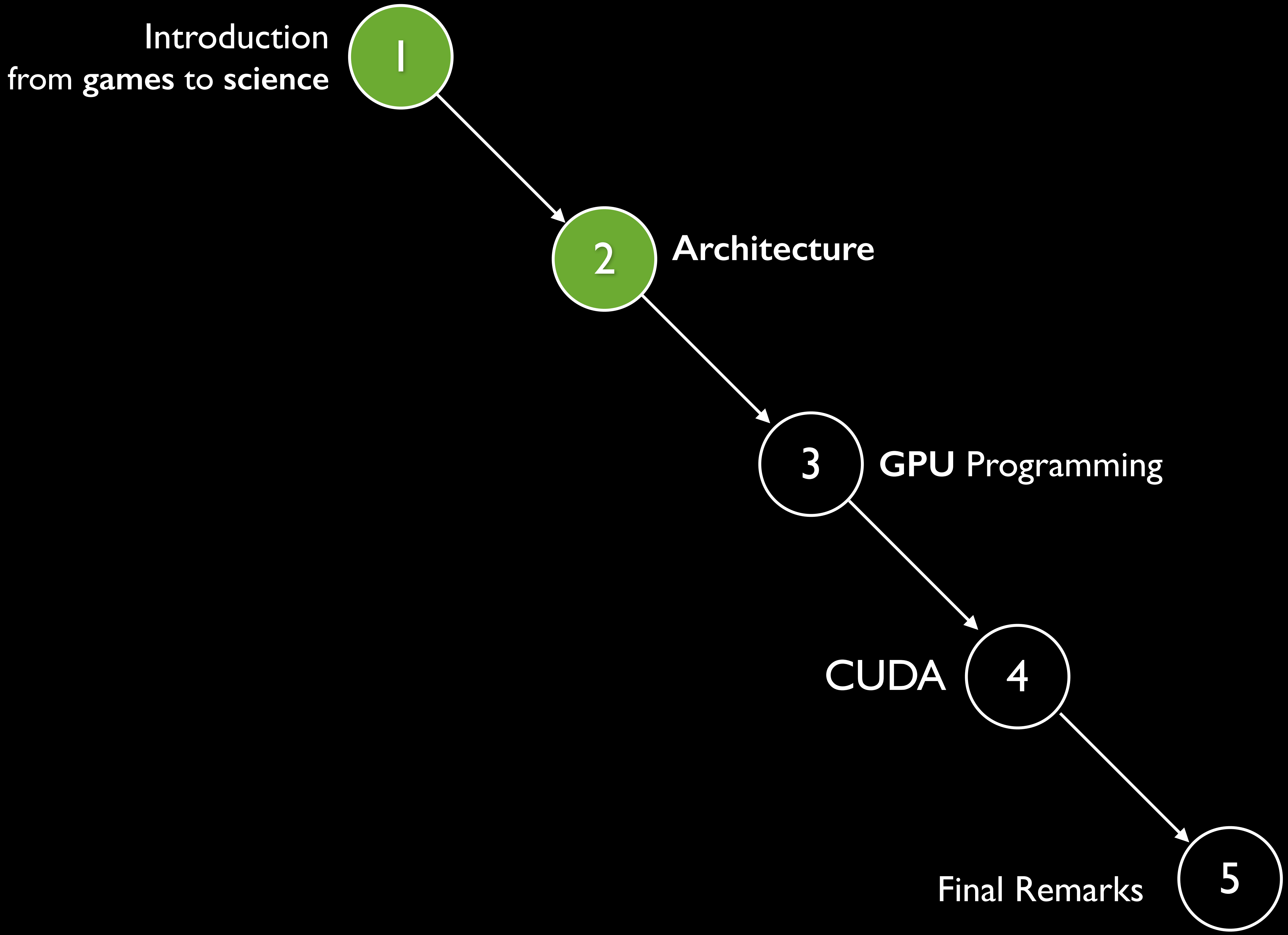
Don’t need to port the application to DirectX or OpenGL



- Proprietary (only work on NVIDIA)
- Enhanced software support
- Several software libraries and examples

CUDA vs OpenCl

- Open specification
- Work on NVIDIA and ATI video cards
- Aim at any computing device



1 Introduction
from games to science

2 Architecture

3 GPU Programming

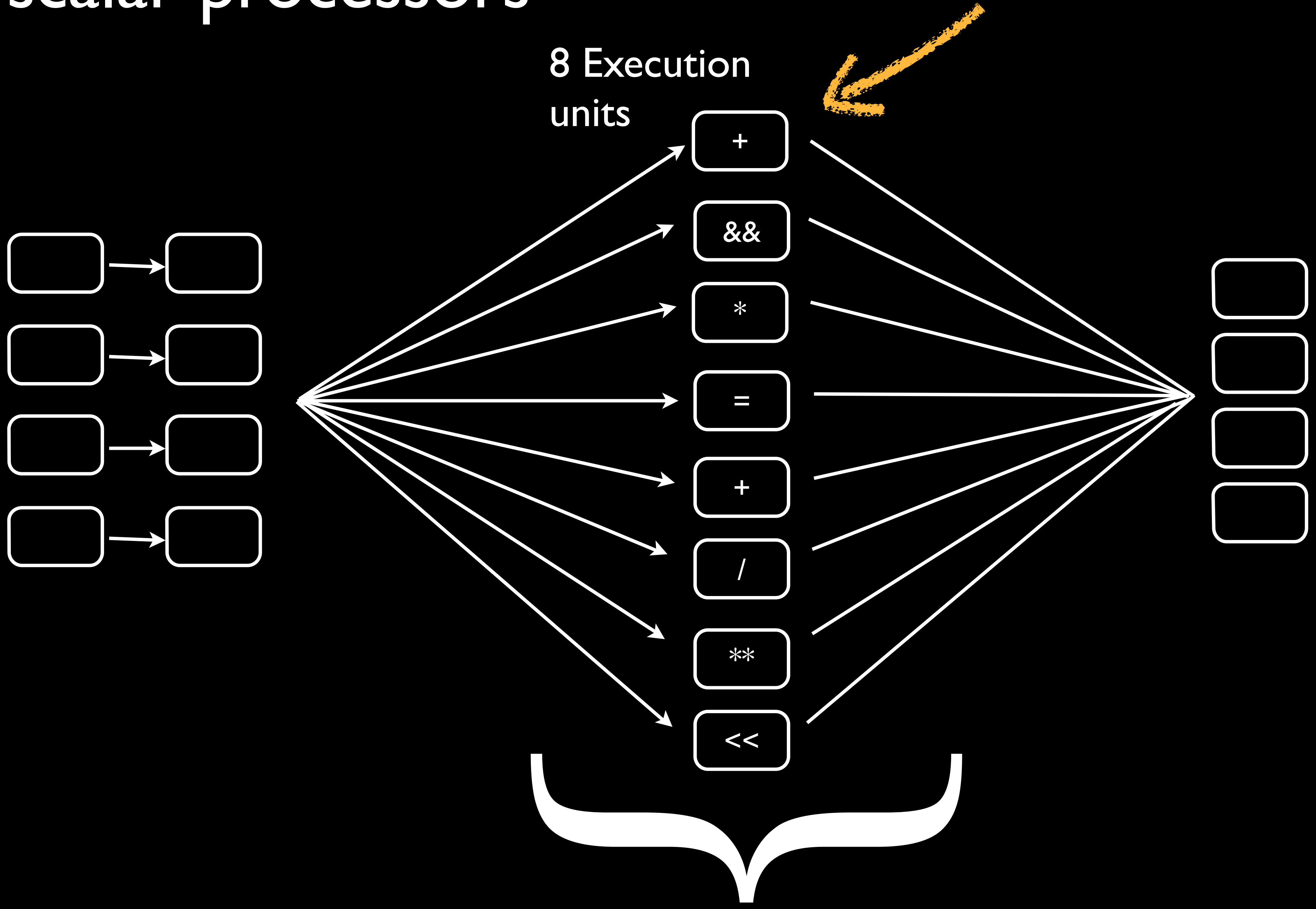
4 CUDA

5 Final Remarks

Computer Architectures
from single thread to multithread

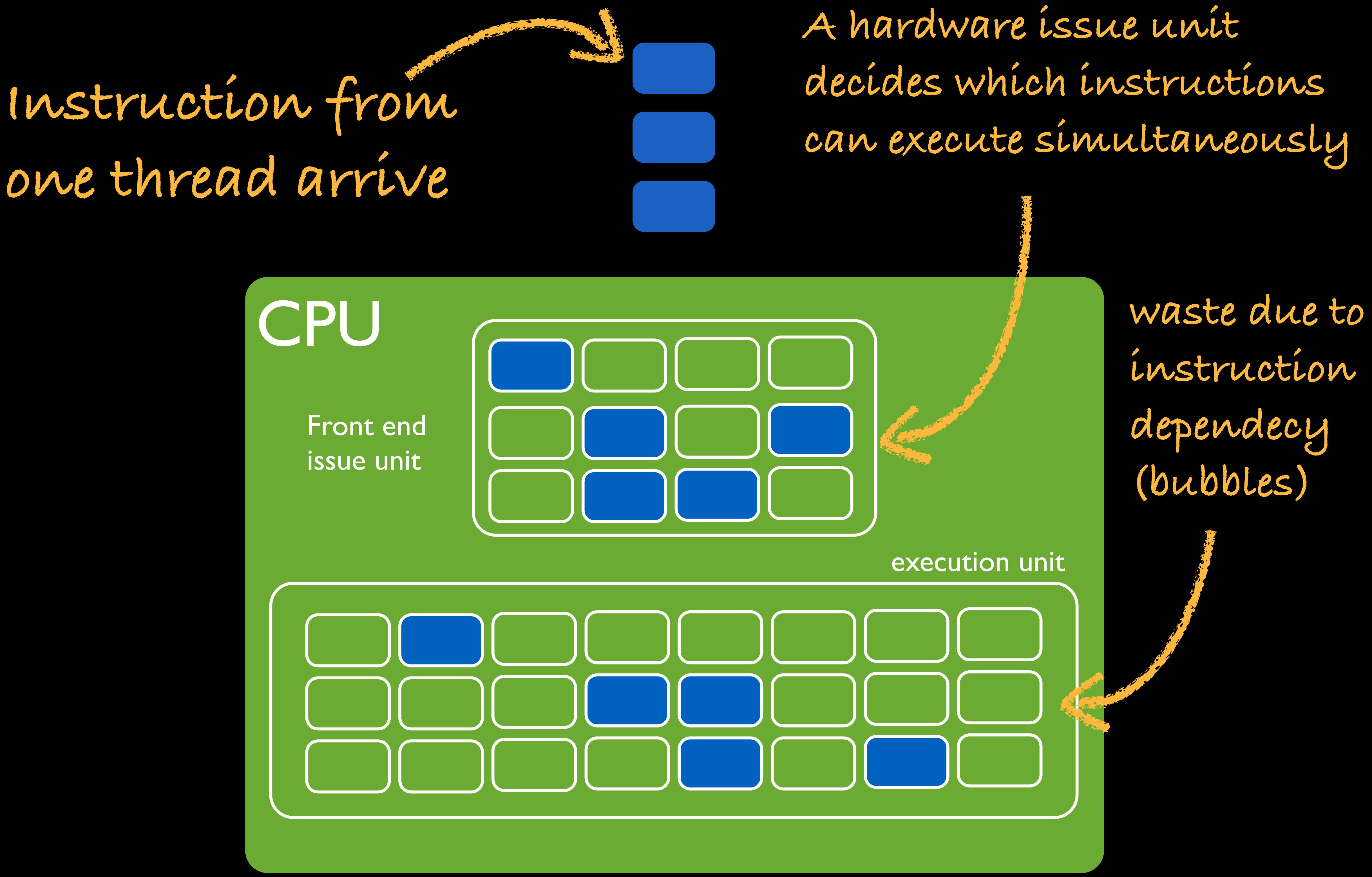
Superscalar processors

Execute up to 8 instructions simultaneously

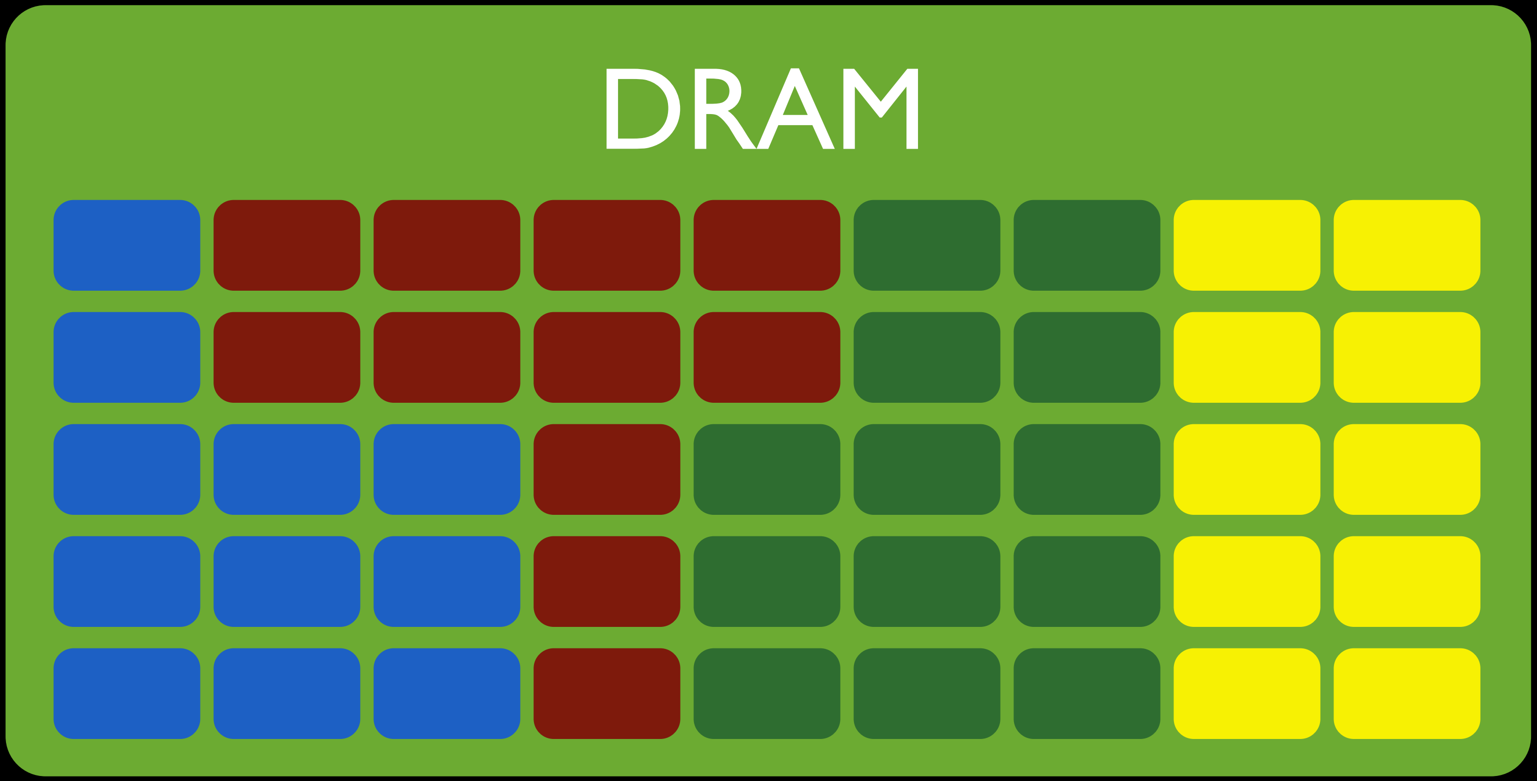


4 stage pipeline

Superscalar processors make the illusion of concurrent execution

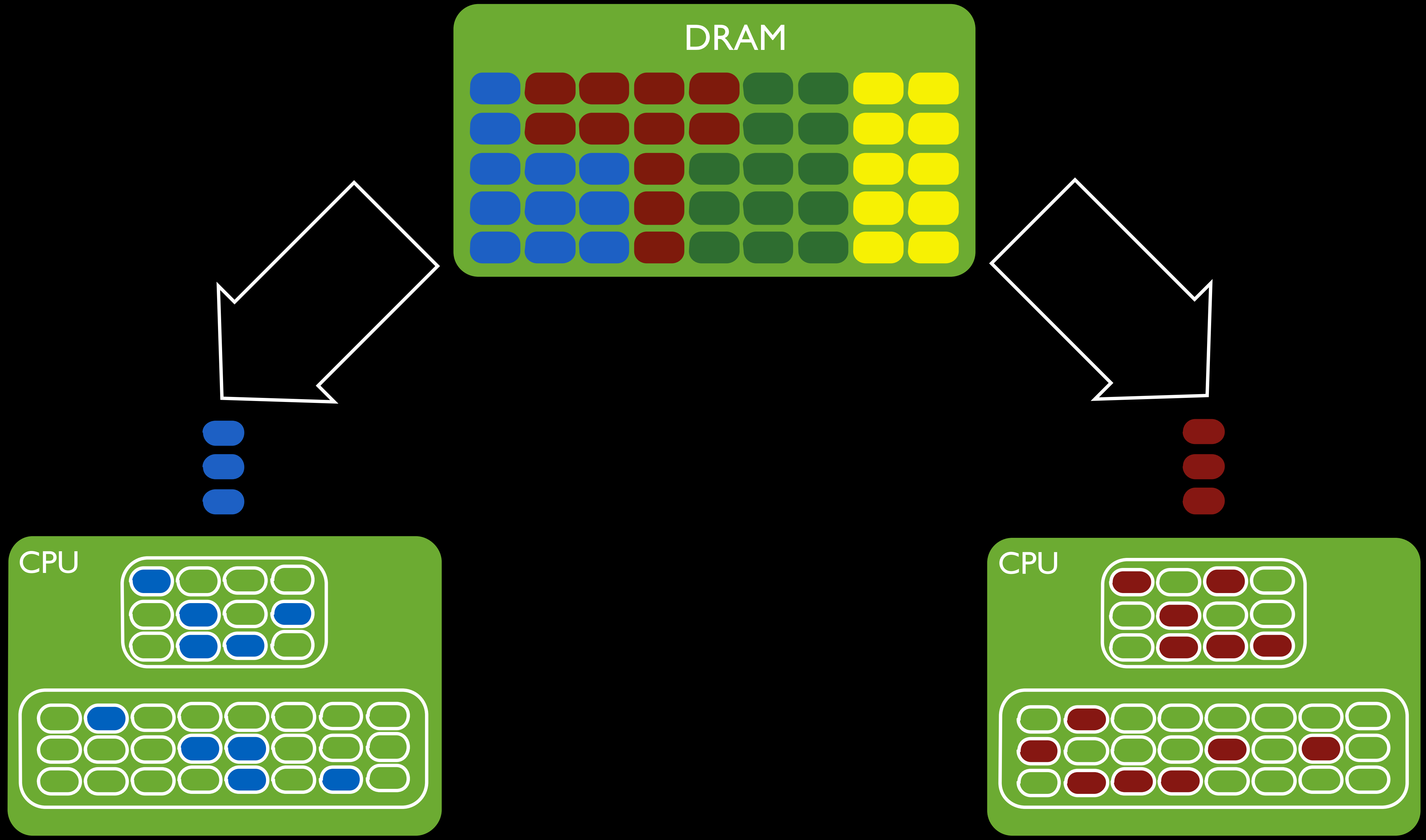

Instruction from one thread arrive
A hardware issue unit decides which instructions can execute simultaneously
CPU
Front end issue unit
execution unit
waste due to instruction dependecy (bubbles)

A program has instructions for several threads in memory

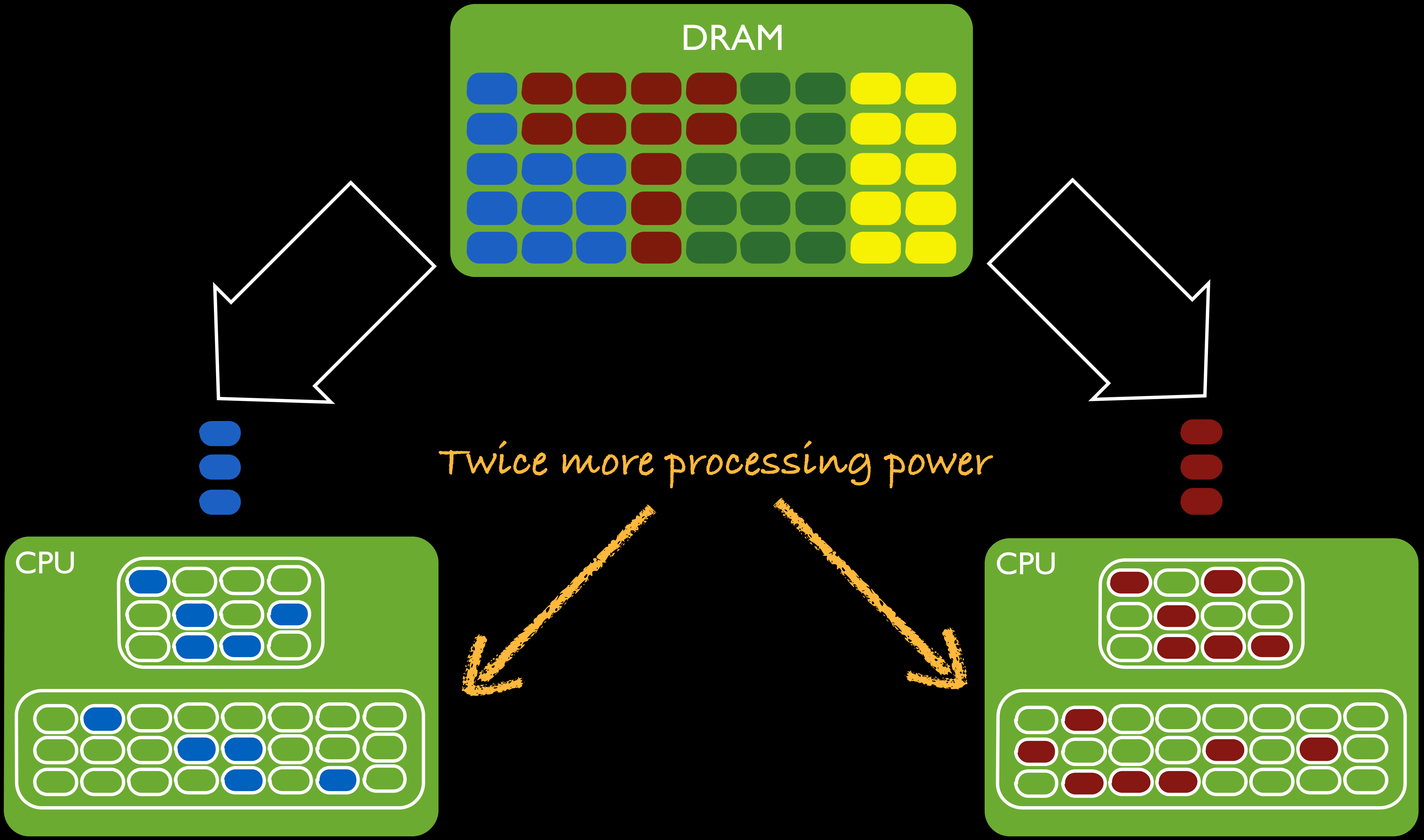


- 🟦 blue thread
- 🟥 red thread
- 🟩 green thread
- 🟨 yellow thread

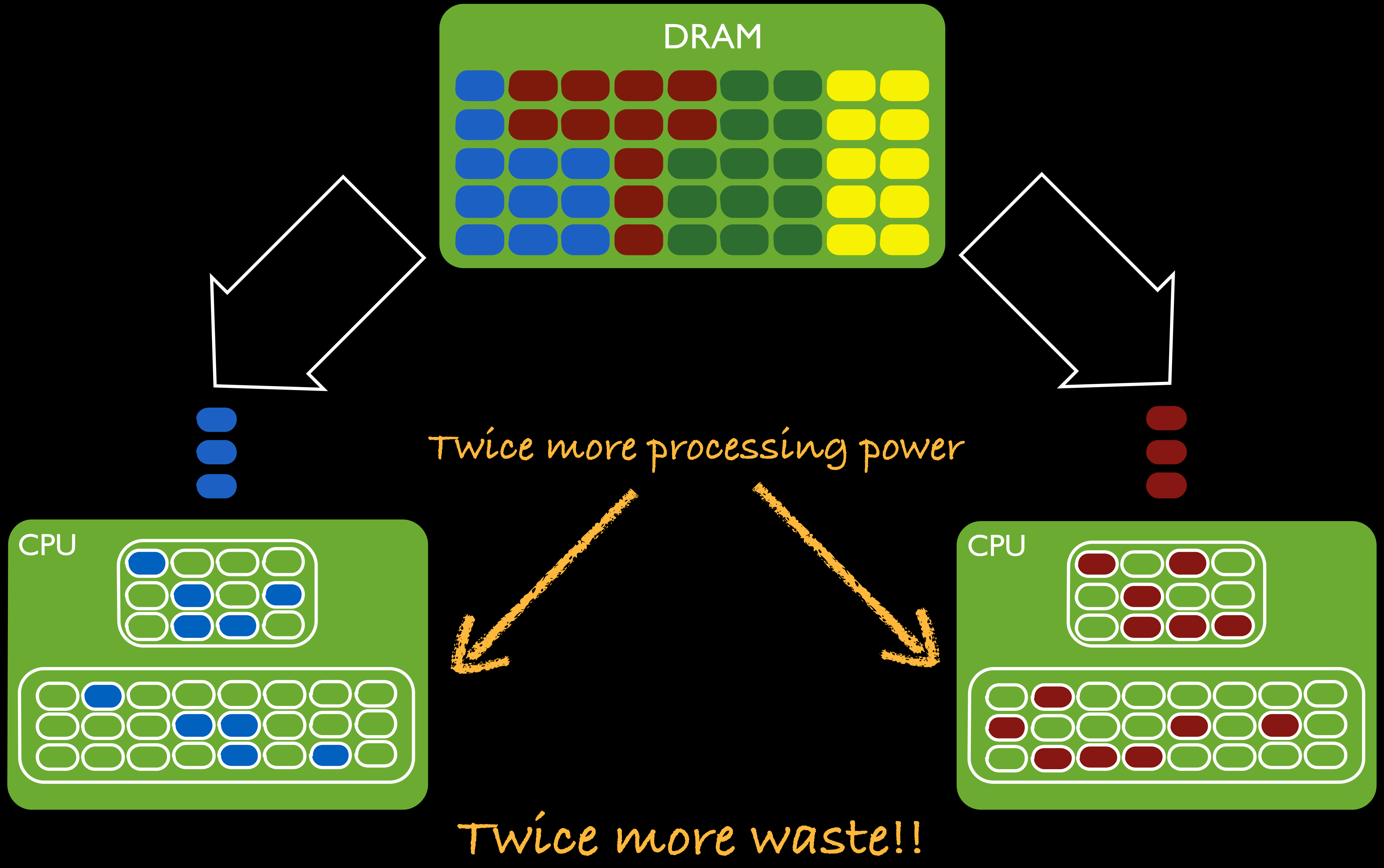
Single threaded multicore



Single threaded multicore



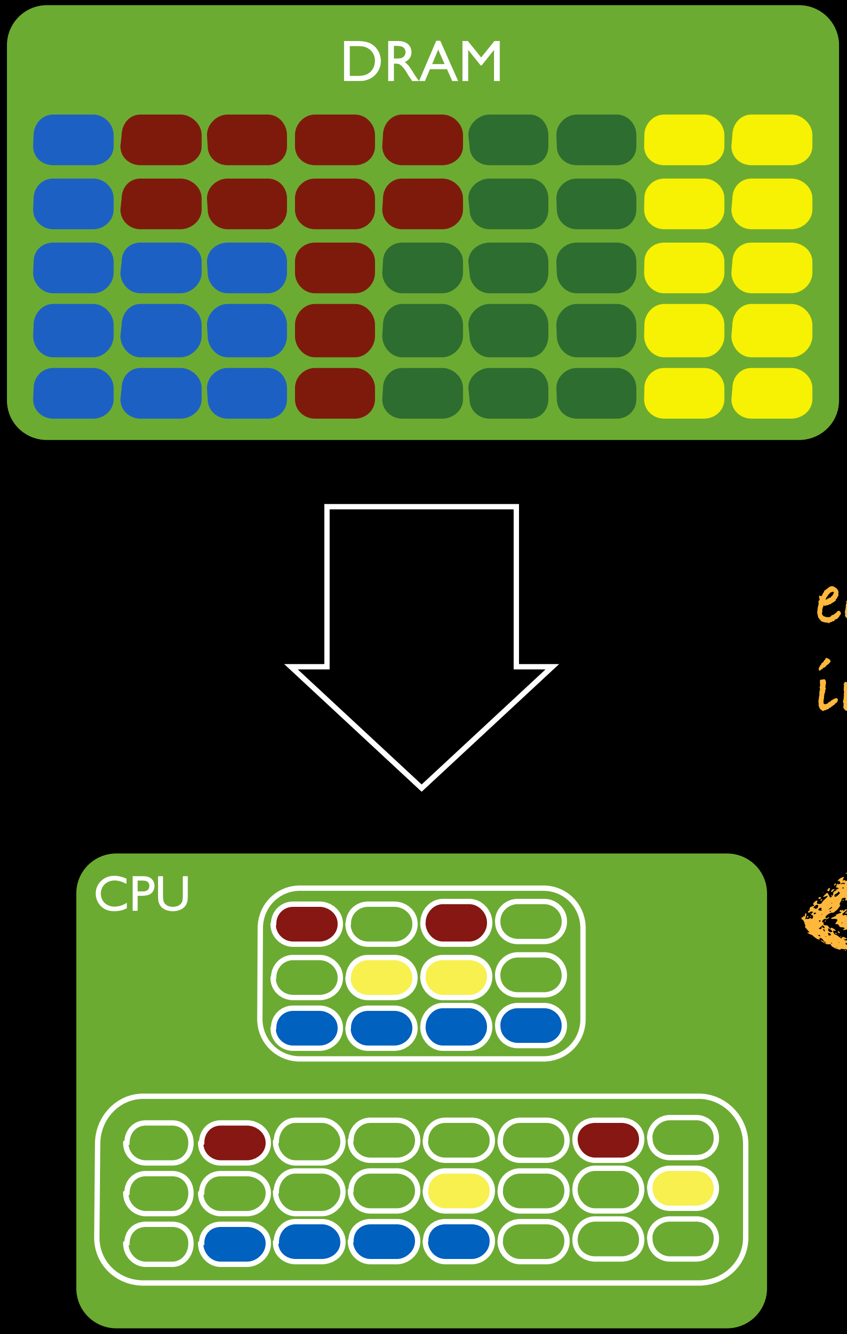
Single threaded multicore



Twice more processing power

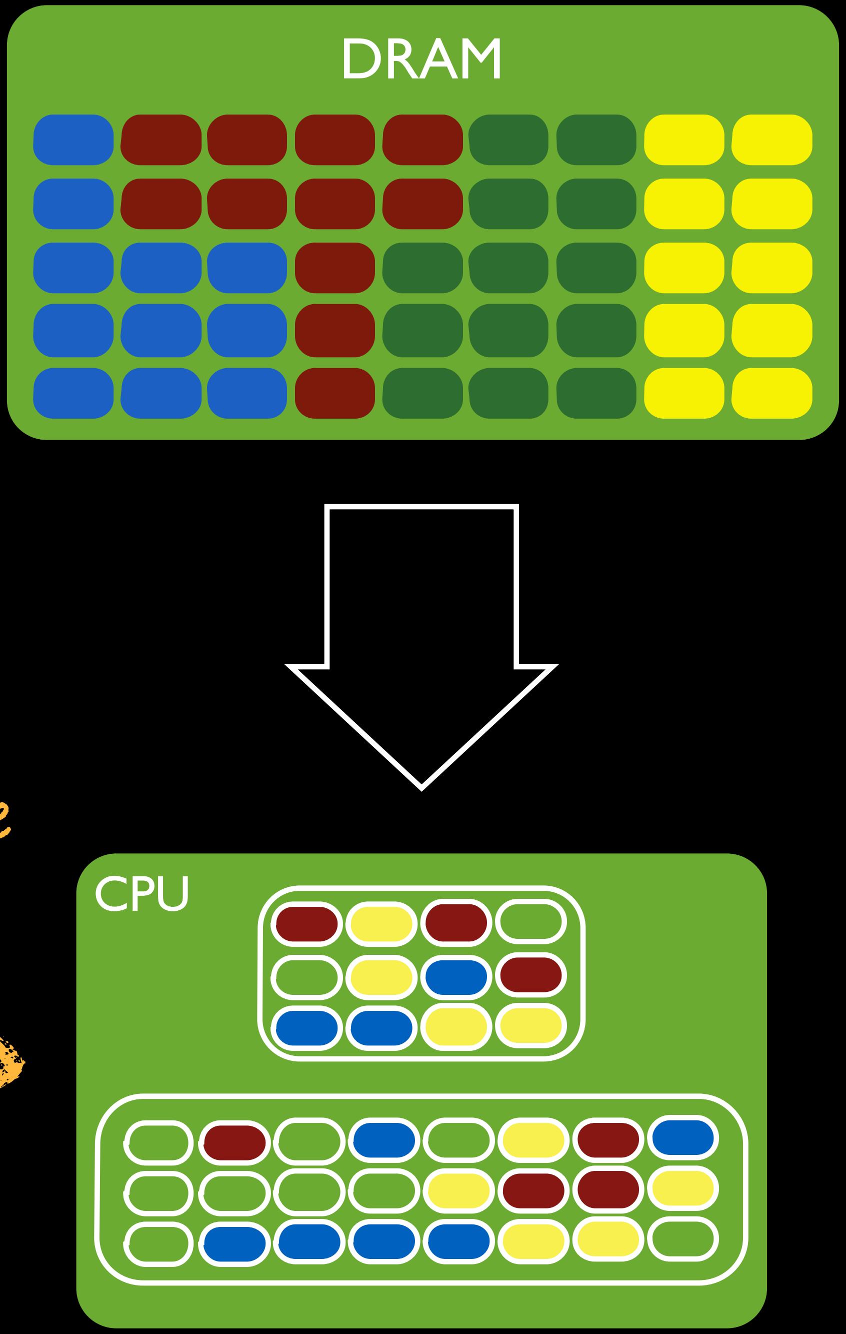
Twice more waste!!

Super-threadeding



each stage only run
instructions from one thread

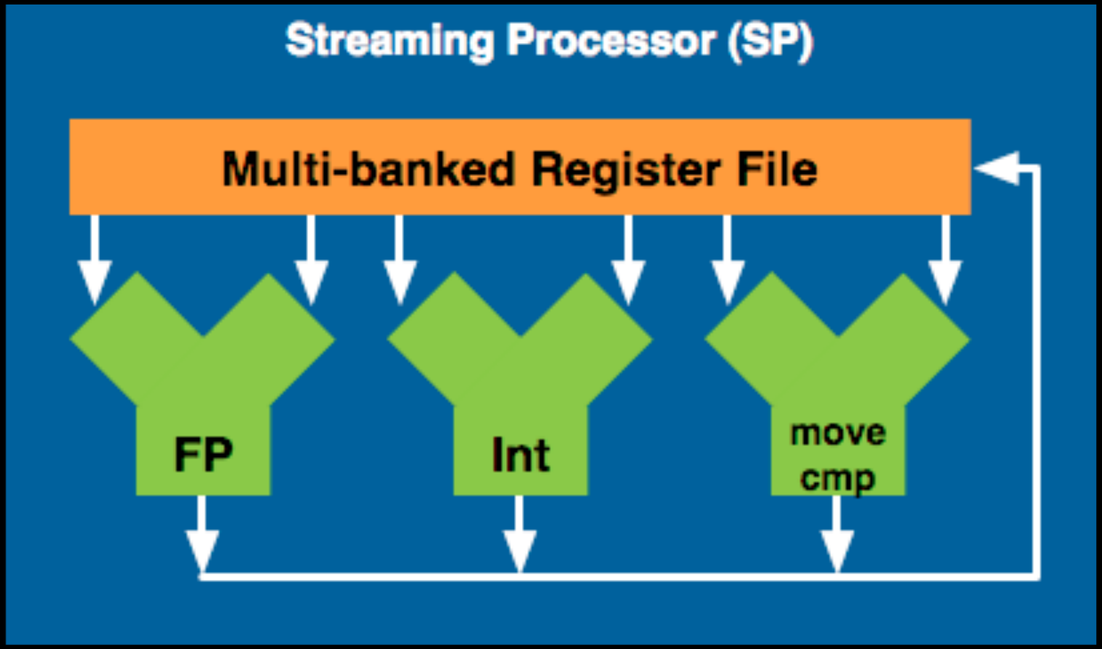
Multi-threadeding



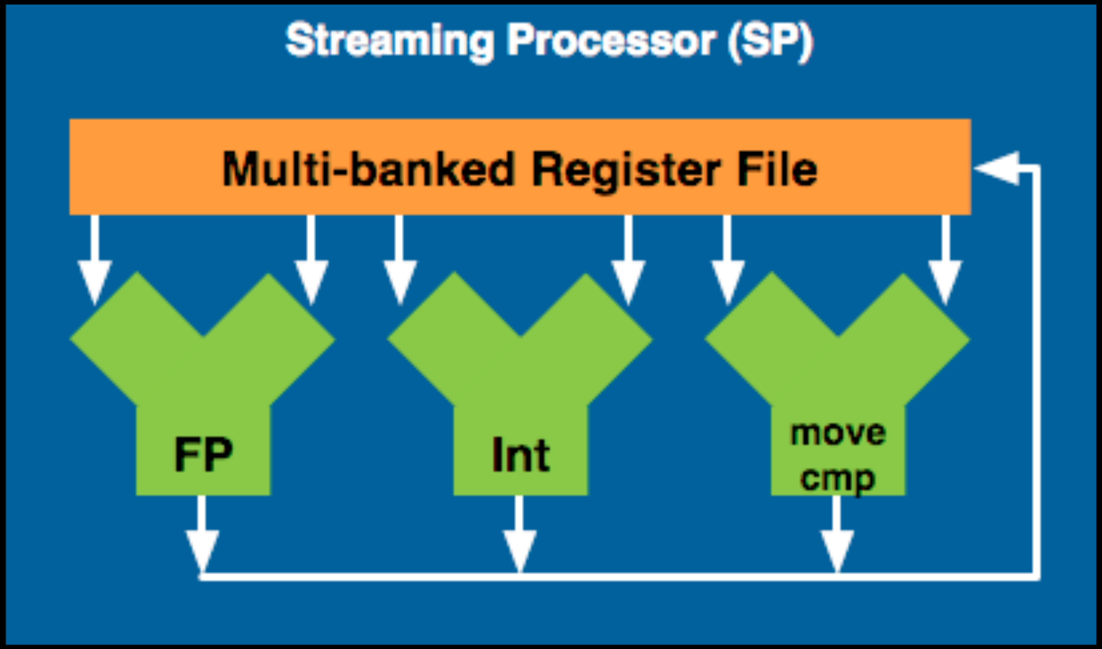
Execute instruction from more than 1 thread at a time

GPU architecture

Streaming Processor (SP)



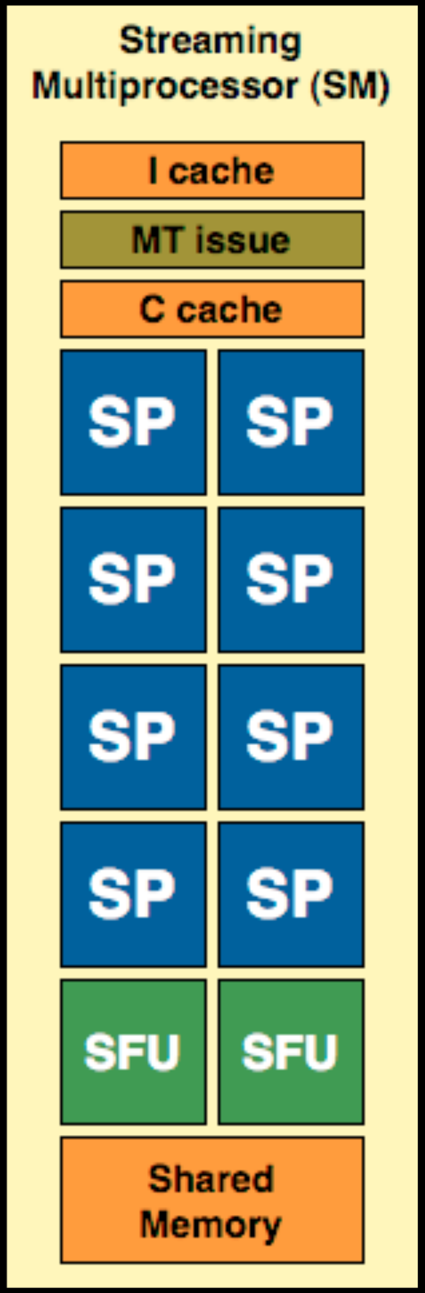
Streaming Processor (SP)



Cacheless
Pipelined
Single issue

Streaming Multiprocessor (SM)

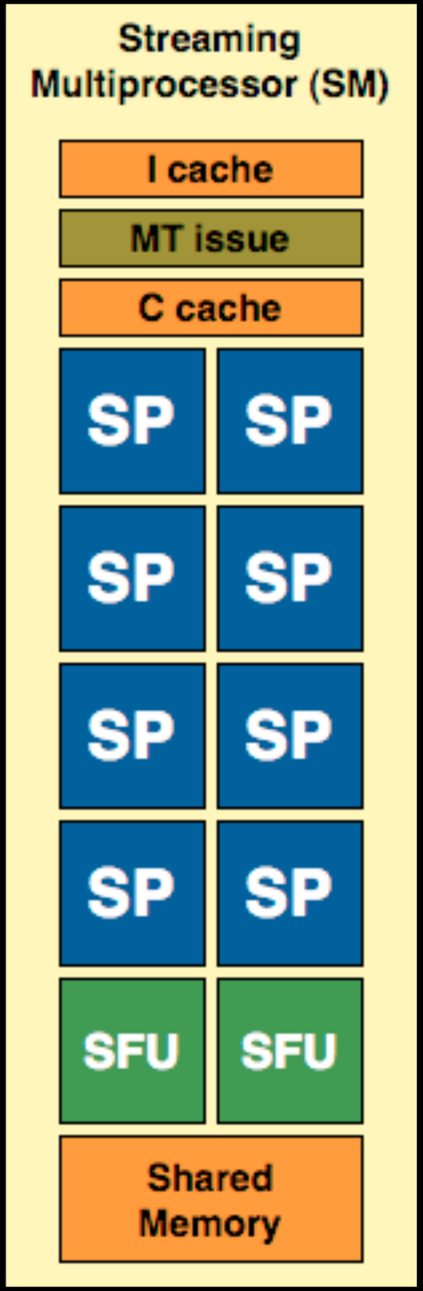
Each SFU
4 FP multiply
for sin, cosin



Array of 8 (eight) SPs

Streaming Multiprocessor (SM)

Each SFU
4 FP multiply
for sin, cosin



Streaming Multiprocessor (SM)

I cache

MT issue

C cache

SP SP

SP SP

SP SP

SP SP

SFU SFU

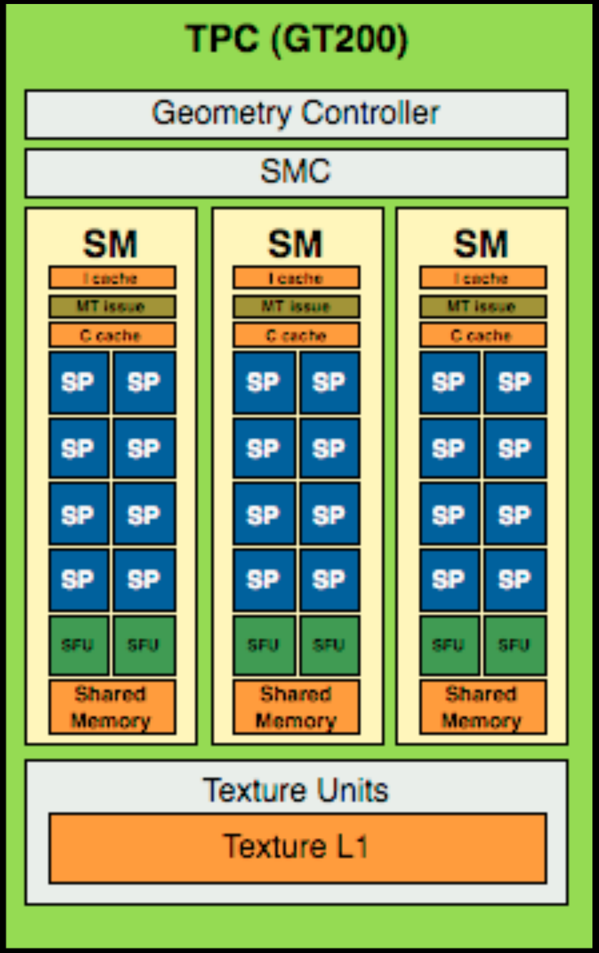
Shared Memory

Multi-threaded
can issue several instructions

Array of 8 (eight) SPs

GPU Architecture (GT200)

Texture Processor Cluster
3 SM's



GPU Architecture (GT200)

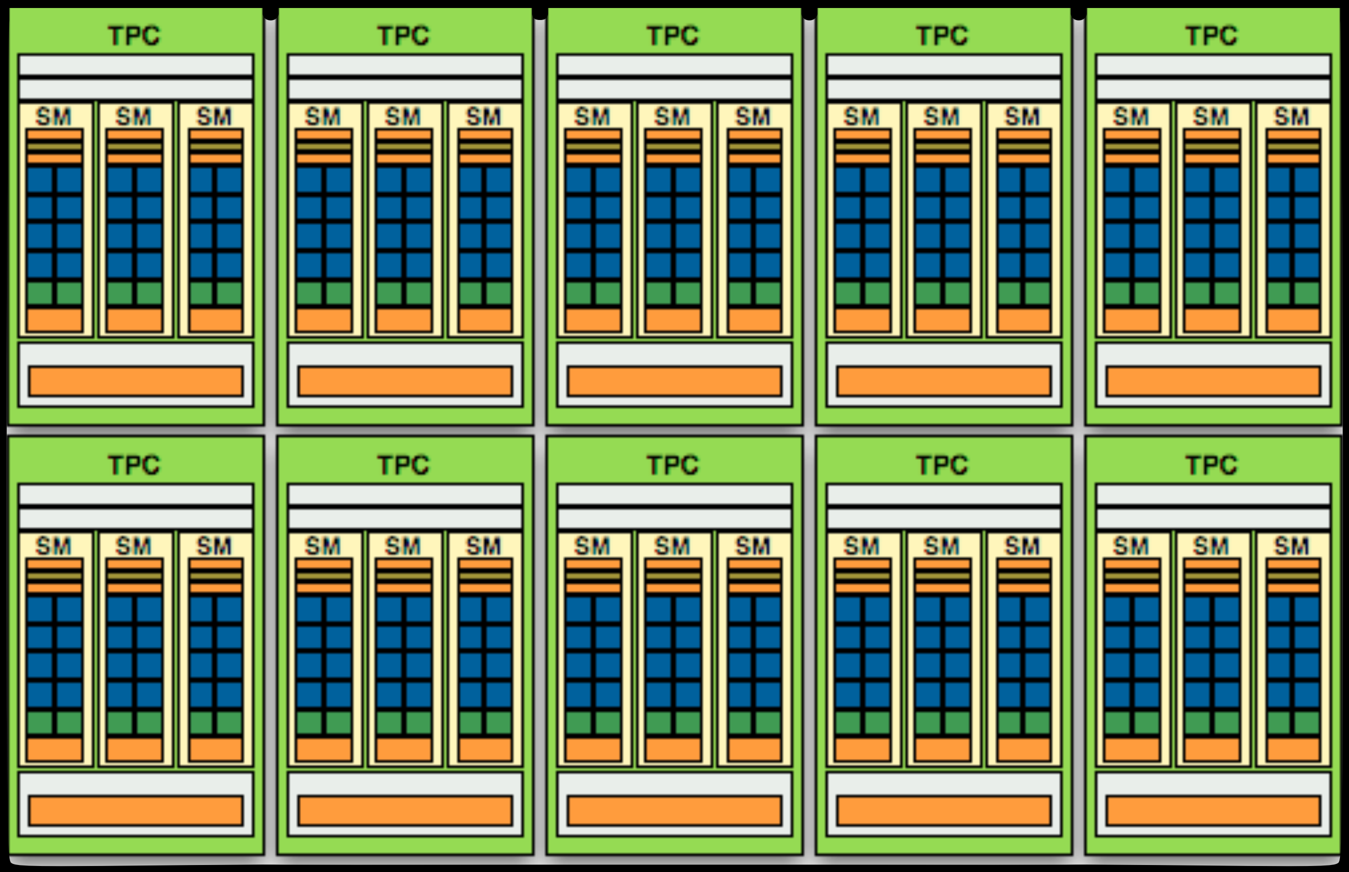
The beast

10 TPC's

3 SM's per TPC

8 SP's per SM

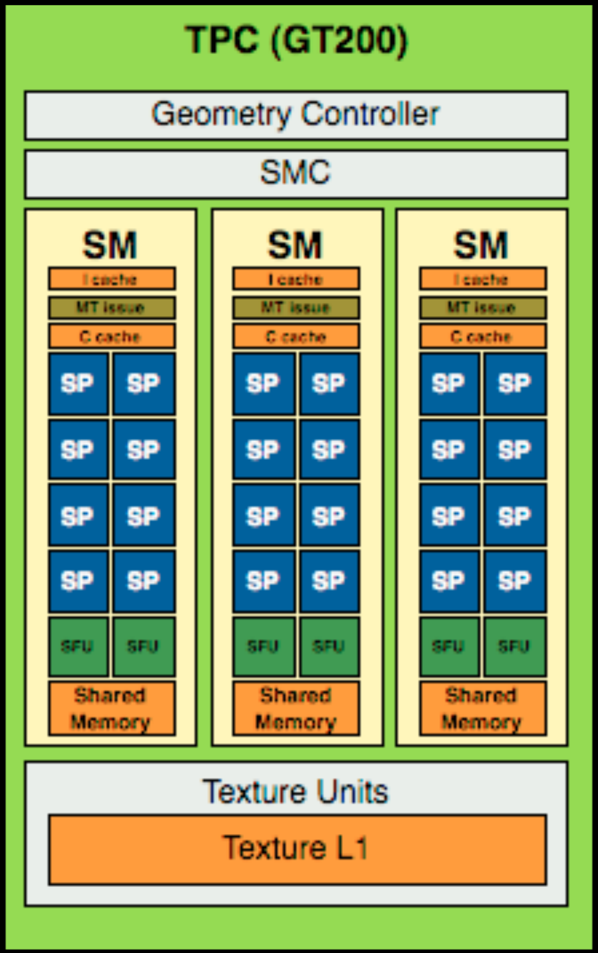
Total of 240 SP's



GPU Architecture (GT200)

very small cache

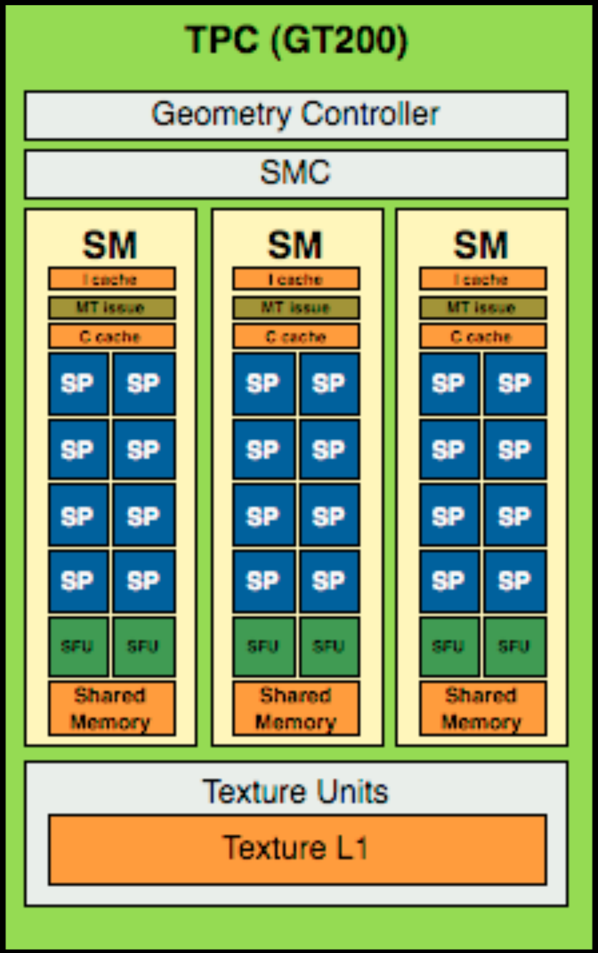
To hide memory latency need several threads available per SM



Schedule per group of 32 threads, called a **warp**

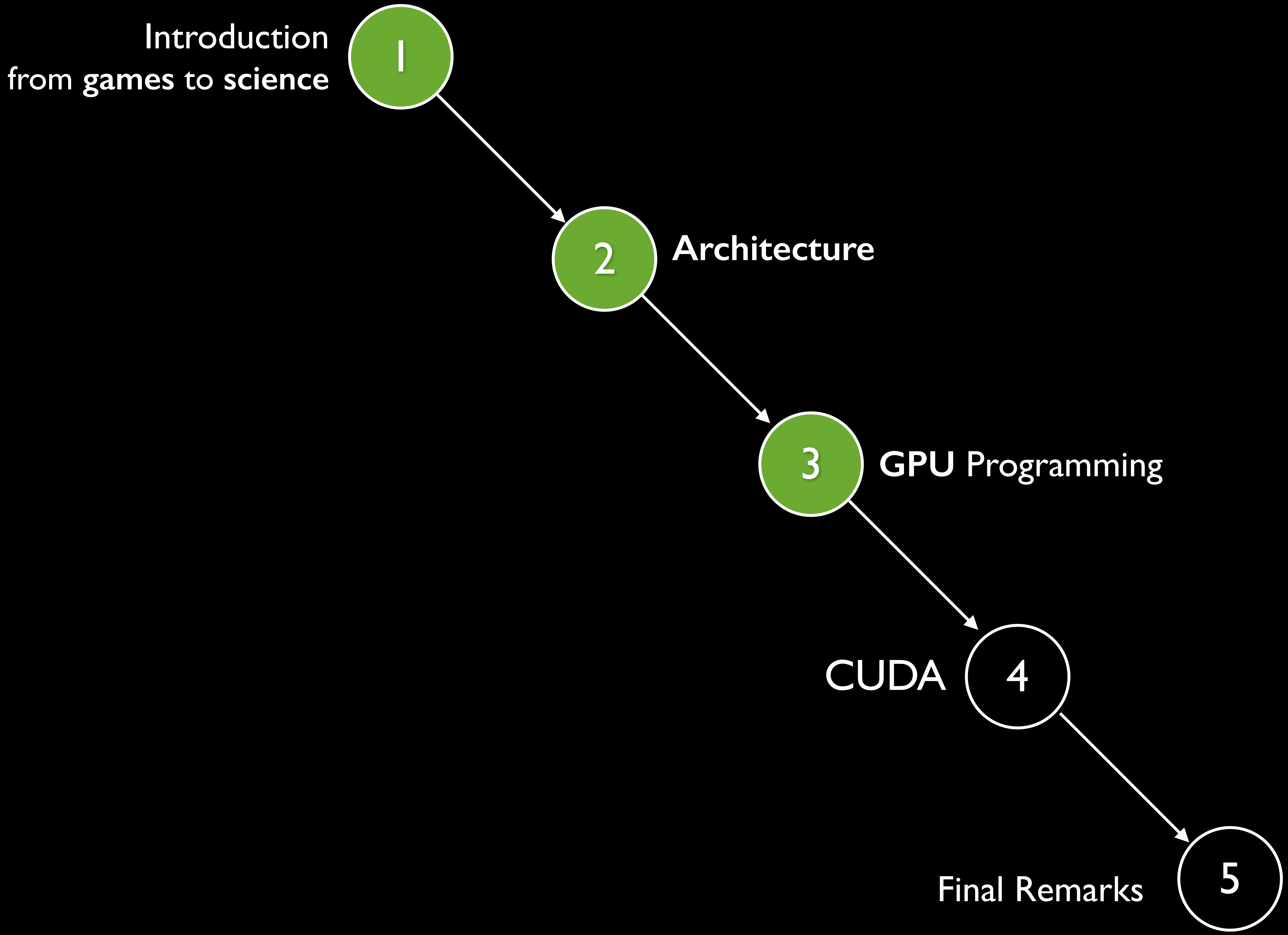
GPU Architecture (GT200)

Each SM handles 32 **warps** simultaneously



32 x 32 = 1024 threads per SM

1024 x 30 = 30720 simultaneous threads



1 Introduction
from games to science

2 Architecture

3 GPU Programming

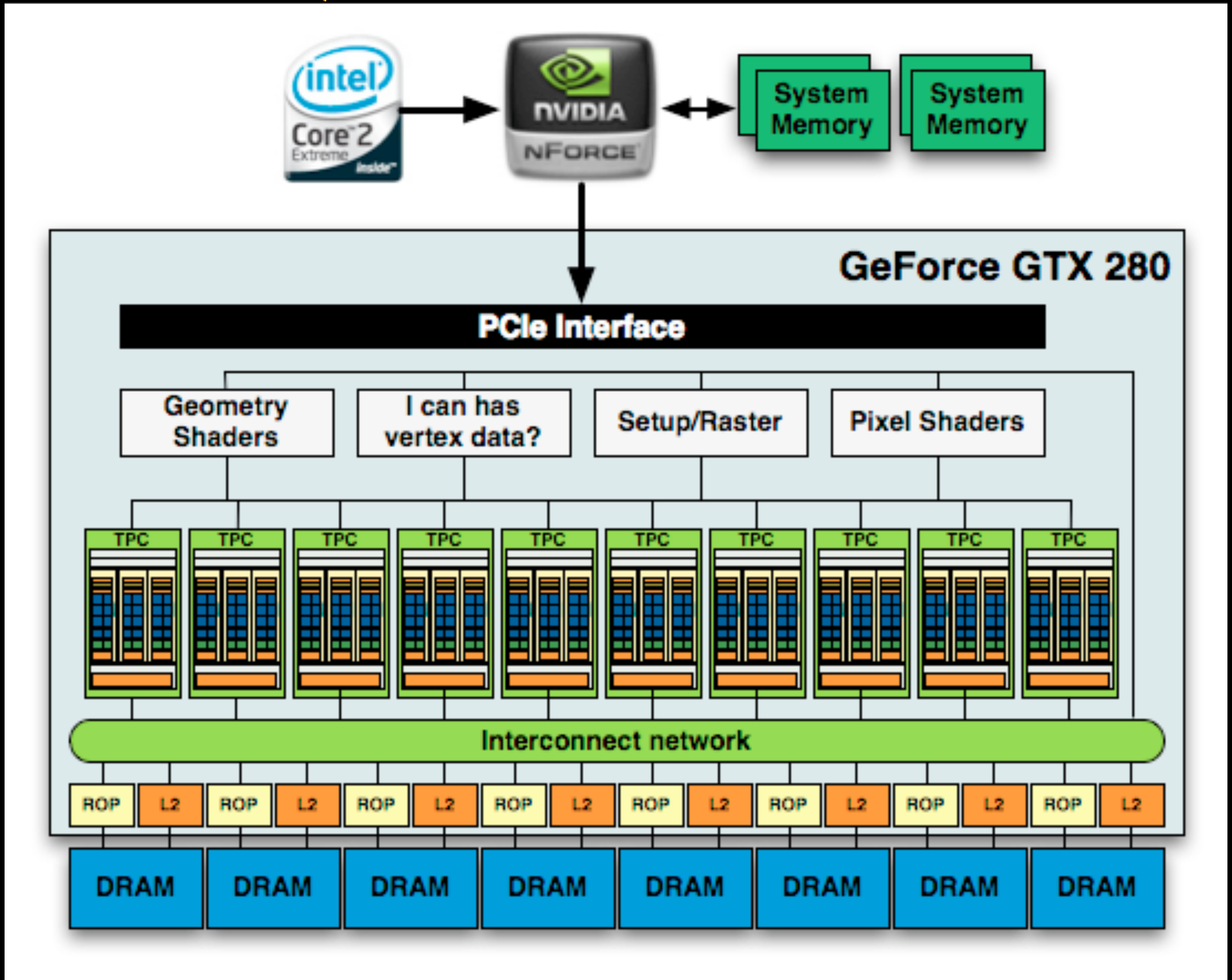
4 CUDA

5 Final Remarks

GPU Programming

CPU is the HOST Processor

GPU is a co-processor



GPU has
its own DRAM

GPU Programming

Massively parallel processor (GT200 - 30720 Threads)

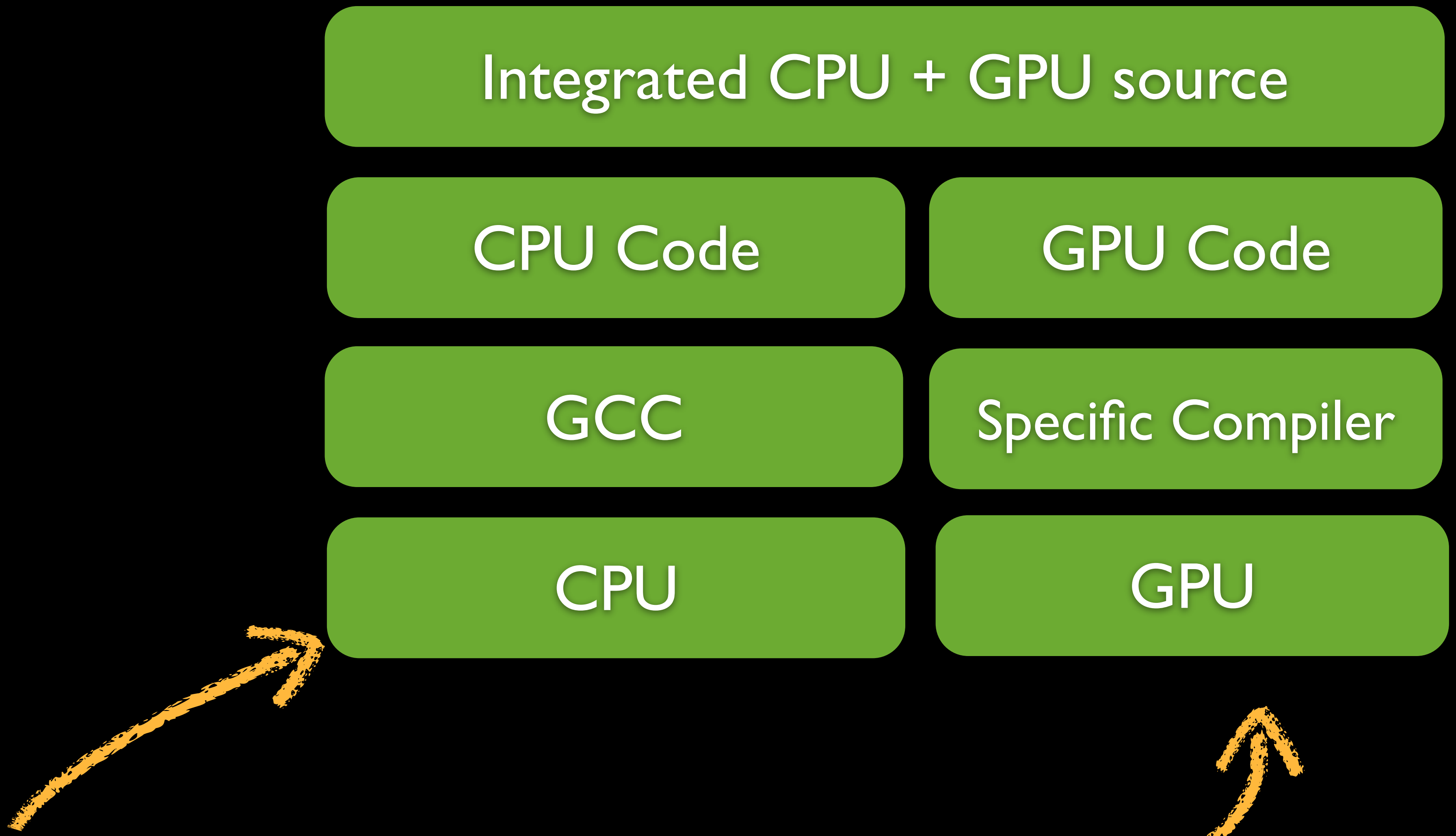
- CPU send burst of threads to execute on the GPU

Use DMA to transfer from CPU DRAM to GPU DRAM

CPU becomes can do something useful aside with GPU

Applications must be rewritten to cope with GPU

GPU Programming



Integrated CPU + GPU source

CPU Code

GPU Code

GCC

Specific Compiler

CPU

GPU

Same code can run on both devices CPU and GPU

Example 1
Discover your CUDA environment.

Using CUDA on Guane Step-by-step

Connect to guane

```
$ ssh guane
```

Reserve a node on shared mode (so other users can have it too)

```
$ oarsub -l nodes=1 -t timesharing -I
```

Download the CUDA exemples from sc-camp.org

```
$ wget http://www.sc-camp.org/cuda/gpu_programming.tgz
```

Configure the path to CUDA_SDK

```
$ export CUDA_SDK_PATH=/usr/local/cuda-6.5/
```

Using CUDA on Guane Step-by-step

Uncompress the folder

```
$ tar zxvf gpu_programming.tgz
```

Enter the directory

```
$ cd gpu_programming/01-devicequery
```

Compile

```
$ make
```

Run

```
$ ./device_query
```

Using CUDA on Guane Step-by-step

All nodes have 8 GPU Tesla S2050

Yet no support to reserve a single CPU+GPU

Share these GPUs wisely with your mates

GPU Programming

Based on the idea of kernel

Essentially SPMD

Define single thread application code

Use thread id to assign different data per thread

GPU Programming

Definition of a single thread computing function (or **kernel**)

GPU Programming

Definition of a single thread computing function (or **kernel**)

```
int kernel()
{
    int i = thread.id;
    a[i] = a[i] + b[i];
}
```

GPU Programming

Definition of a single thread computing function (or **kernel**)

```
int kernel()
{
    int i = thread.id;
    a[i] = a[i] + b[i];
}
```

1- How to Compute the thread ID?

2- How do we copy data from CPU to GPU?

3- How to dispatch kernel on the device?

4- How to get results back when done?

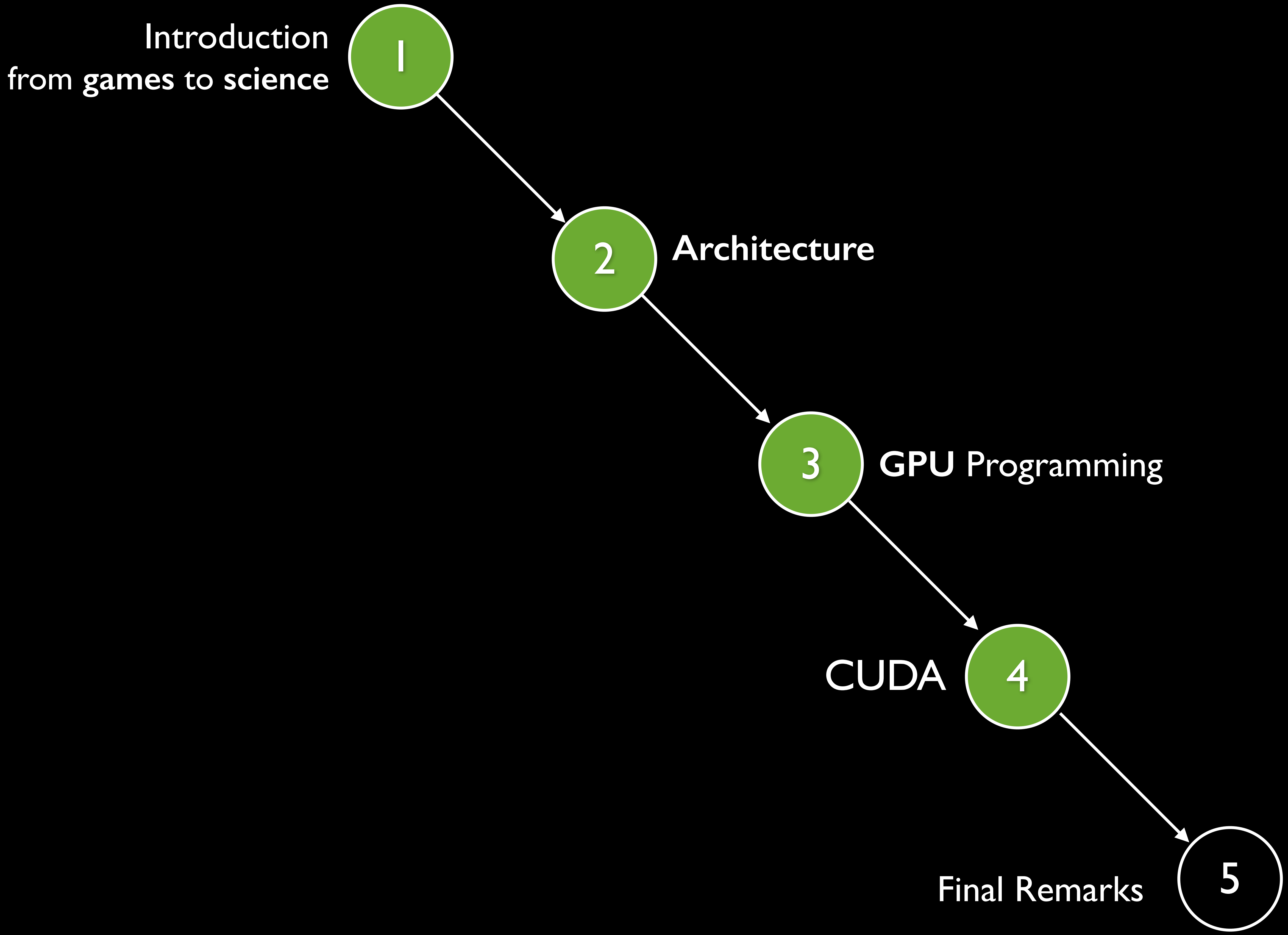
GPU Programming

Have support for operations on the Host (CPU) and Device (GPU)

1- Copy data from Host to Device
2- Execute kernel on the device
3- Wait for kernel to finish
4- Copy data from Device to Host

mallocDeviceMemory
copyFromHostToDevice
computeKernel
copyFromDeviceToHost

Depends on the programming interface



1 Introduction
from games to science

2 Architecture

3 GPU Programming

4 CUDA

5 Final Remarks

CUDA Programming

- C extension

- Support for several platforms:
  - Linux
  - Windows
  - MacOS

- Need to install NVIDIA Driver, Toolbox and SDK

CUDA Programming

Provide several libraries

STL C++ Port to CUDA

Linear Algebra
cuBLAS

CUDA Programming

Requirements for Linux

- 1 NVIDIA CUDA aware card
- GCC installed
- Downloaded Toolkit, Driver, and SDK

MOSTRAR SITE

Step-by-step installation:

- Install the CUDA Toolkit

```
$ ./cudatoolkit_4.2.9_linux_64_ubuntu11.04.run
```

- Install the driver

```
$ sudo ./devdriver_4.2_linux_64_295.41.run
```

- Restart GUI

```
$ sudo /etc/init.d/gdm start
```

- Install SDK

```
$ ./gpucomputingsdk_4.2.9_linux.run
```

Only the driver requires superuser priviledges

CUDA Programming API

Function directives

Kernel function must respect several properties

must return void

no static variables

no recurrence

no variable number of arguments

	Execute on	Called from
__device__ float DeviceFunc(...)	device	device
__global__ void kernelFunc(...)	device	host
__host__ float HostFunc(...)	host	host

Can be used combined with __device__

Example II
Simple kernel hello world.

Hello World

```
__global__ void mykernel (void){
  //simple kernel does nothing
}
```

```
int main(void) {
  mykernel<<<1,1>>>();
  printf("Hello World!\n");
  return 0;
}
```

GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Where should we use parallel computing?

GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Multi threaded application

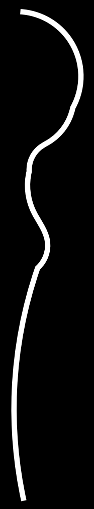
GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Multi threaded application

```
thread 23
c[23] = a[23] + b[23];
```



GPU Programming

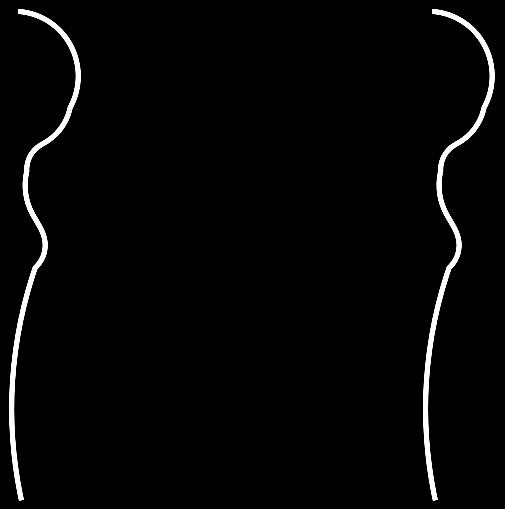
Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Multi threaded application

thread 23
c[23] = a[23] + b[23];

thread 2
c[2] = a[2] + b[2];

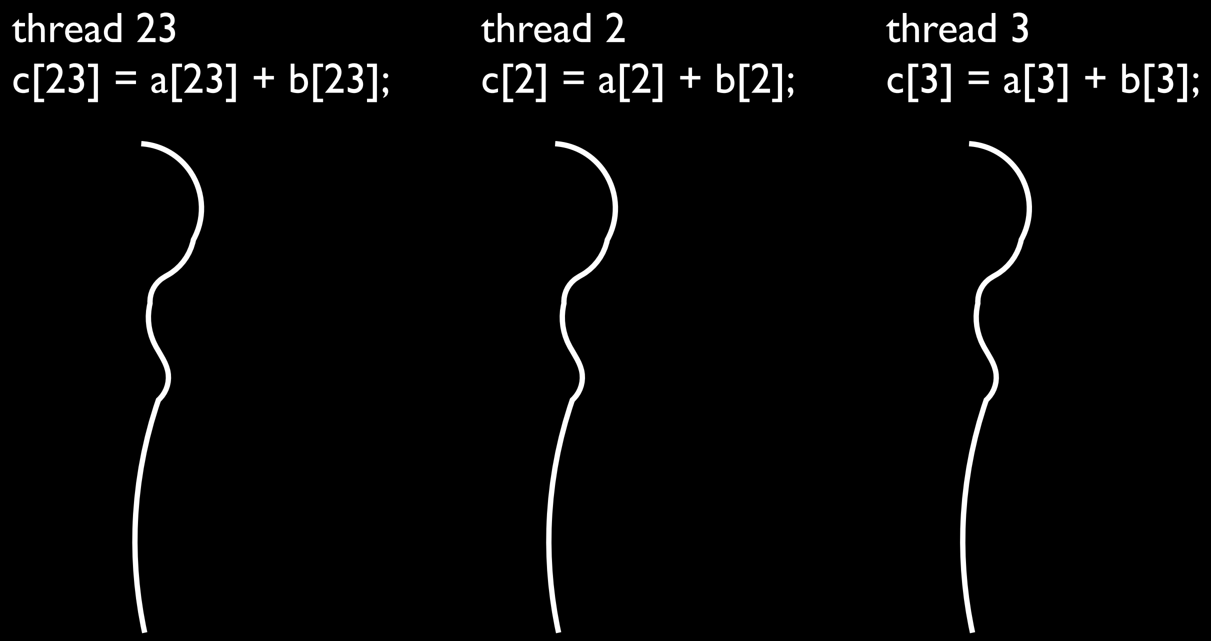


GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Multi threaded application



thread 23
c[23] = a[23] + b[23];

thread 2
c[2] = a[2] + b[2];

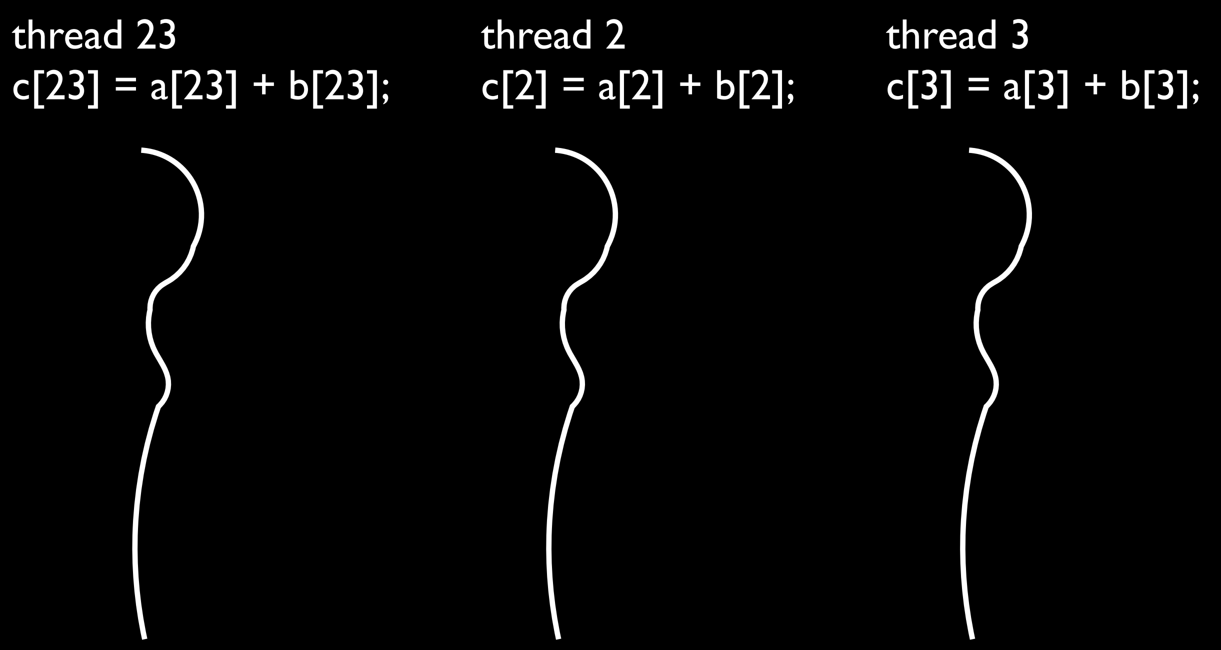
thread 3
c[3] = a[3] + b[3];

GPU Programming

Single threaded application

```
int a[1024];
int b[1024];
int c[1024];

int main()
{
    for(int i=0; i<1024; i++){
        c[i] = a[i] + b[i];
    }
}
```

Multi threaded application



thread 23
c[23] = a[23] + b[23];

thread 2
c[2] = a[2] + b[2];

thread 3
c[3] = a[3] + b[3];

Need to instantiate 1024 threads

GT200 supports up to 30720 threads simultaneously!!!

Example III

Add two integers.

Adding two integers

```
__global__ void add(int *a, int *b, int *c) {
   *c = *a + *b;
}
```

Adding two integers

```
int main(void) {
     int a, b, c;               // host copies of a, b, c
     int *d_a, *d_b, *d_c;      // device copies of a, b, c
     int size = sizeof(int);
     // Allocate space for device copies of a, b, c
     cudaMalloc((void **)&d_a, size);
     cudaMalloc((void **)&d_b, size);
     cudaMalloc((void **)&d_c, size);
     // Setup input values
     a = 2;
     b = 7;
     // Copy inputs to device
     cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
     cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);
     // Launch add() kernel on GPU
     add<<<1,1>>>(d_a, d_b, d_c);
     // Copy result back to host
     cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
     // Cleanup
     cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
     return 0;
  }
```

CUDA Programming API

Memory allocation

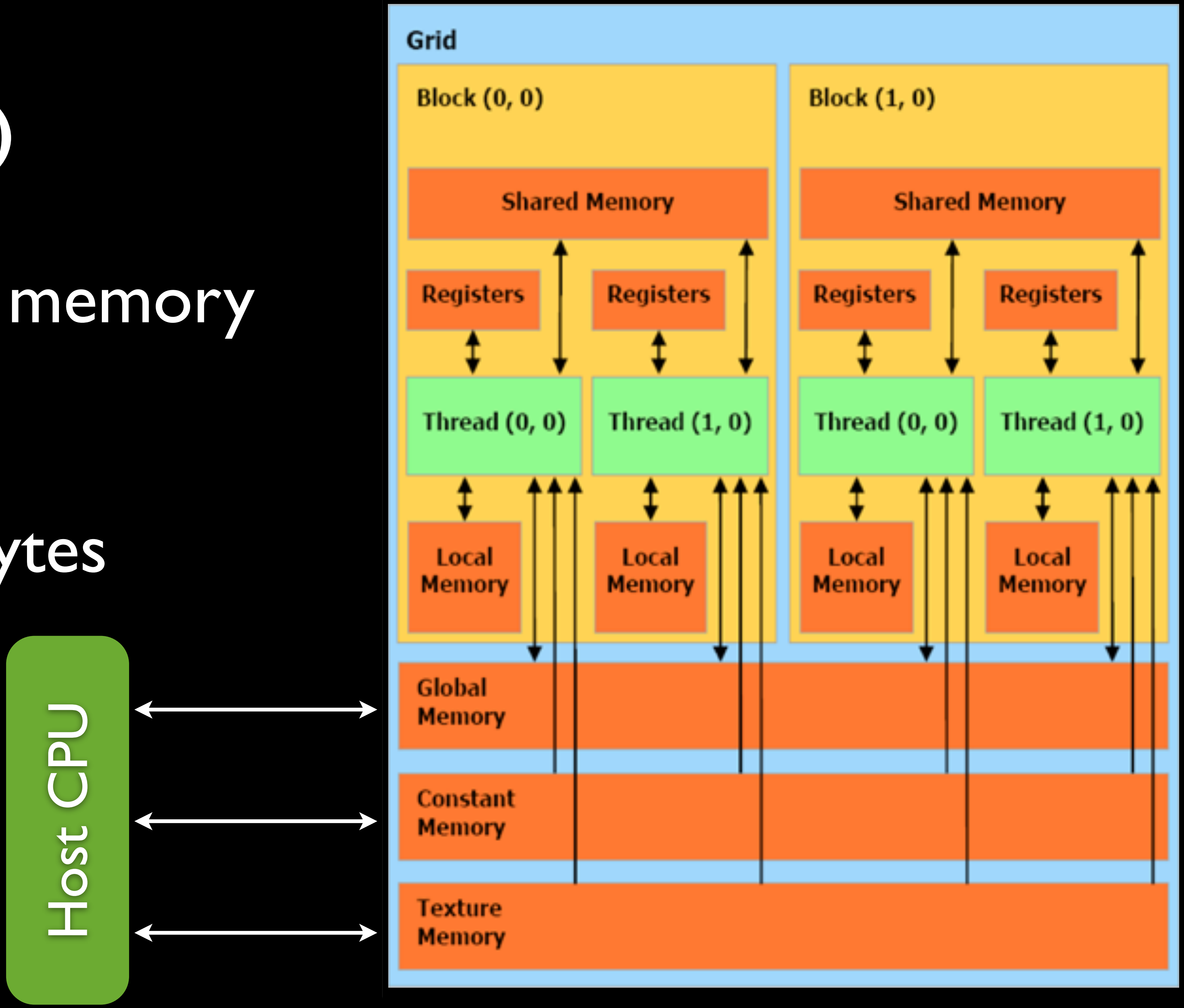
cudaMalloc(...)

Allocate global memory

2 parameters:

Pointer

Number of bytes



CUDA Programming API

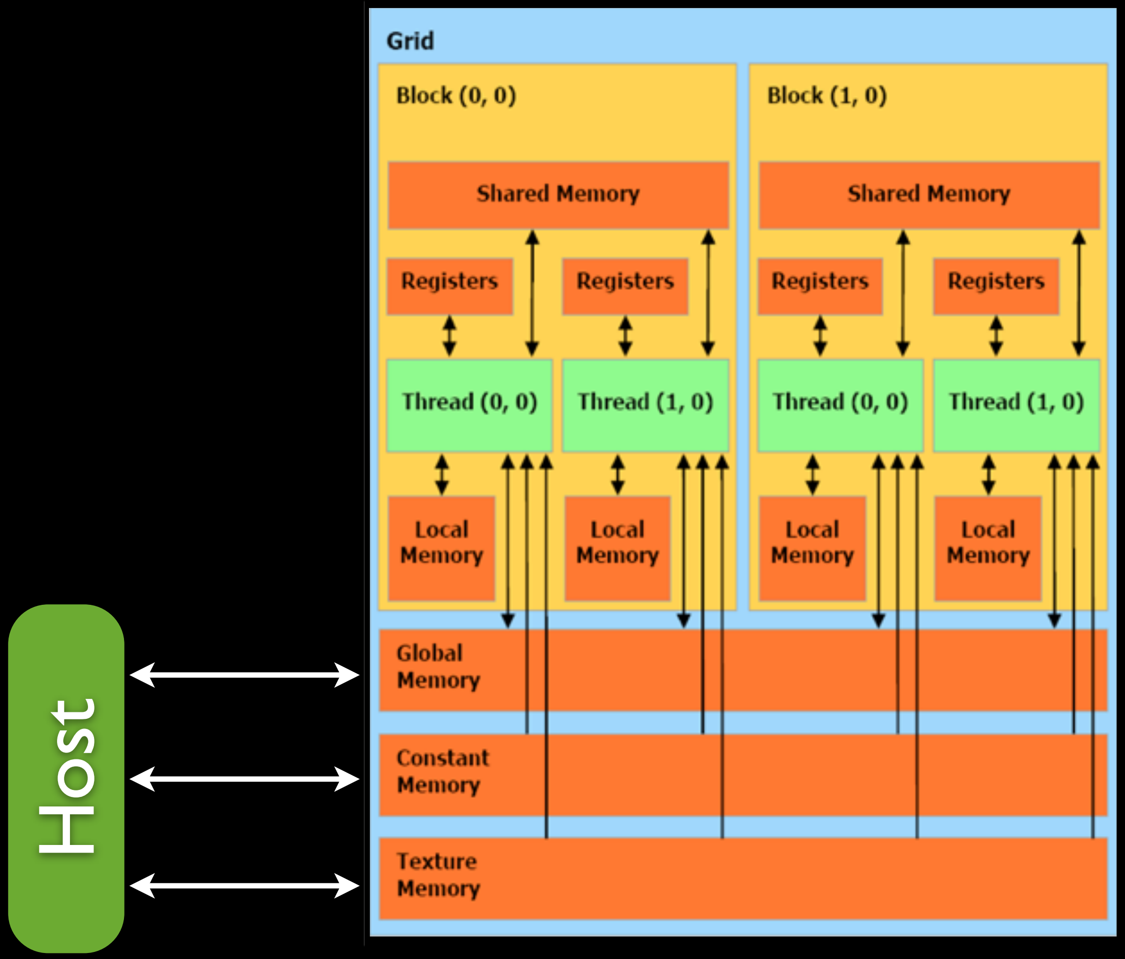
Transfer data

cudaMemcpy(...)

4 parameters:

- Destination pointer
- Source pointer
- Bytes to copy
- Transfer type

  - HostToHost
  - HostToDevice
  - DeviceToHost
  - DeviceToDevice

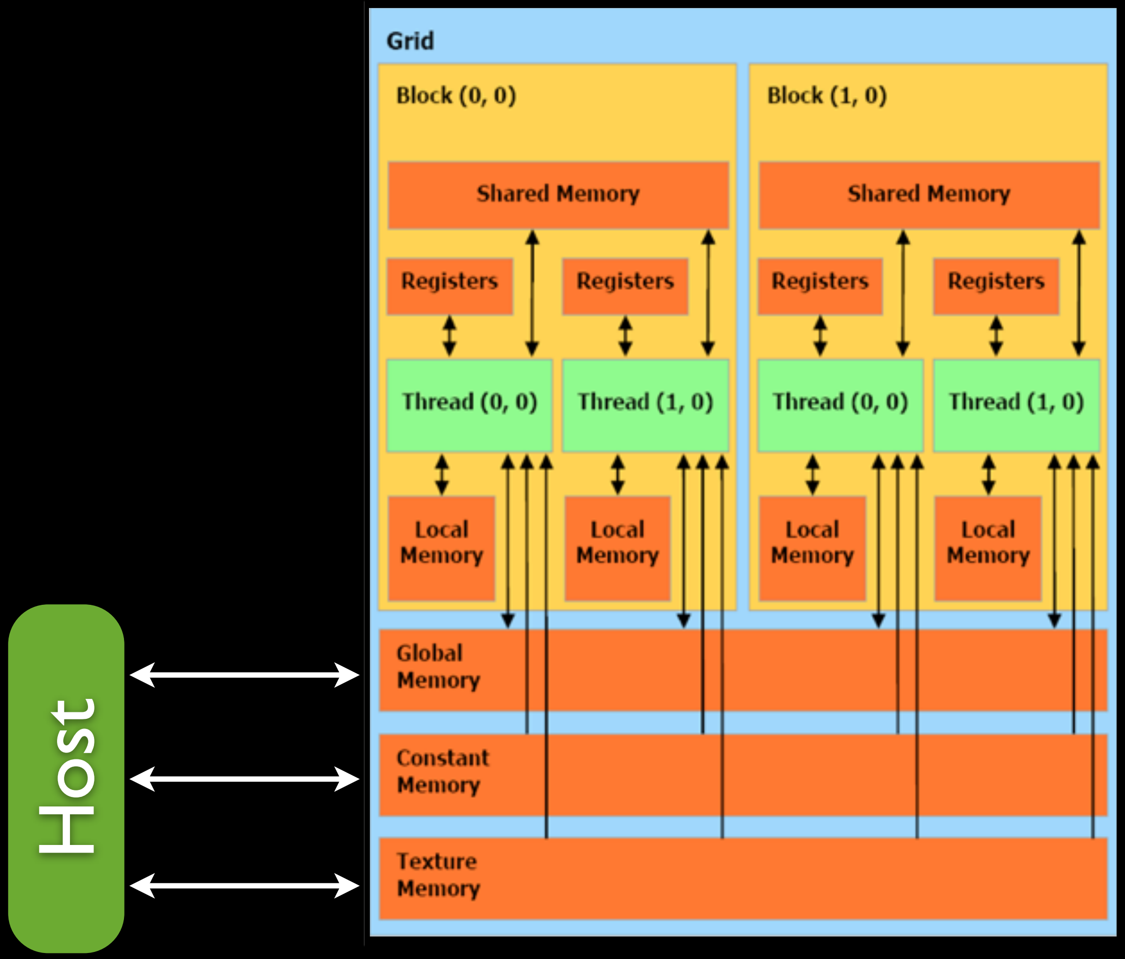


CUDA Programming API

Memory deallocation

cudaFree(...)

Frees global memory
1 parameter:
Pointer

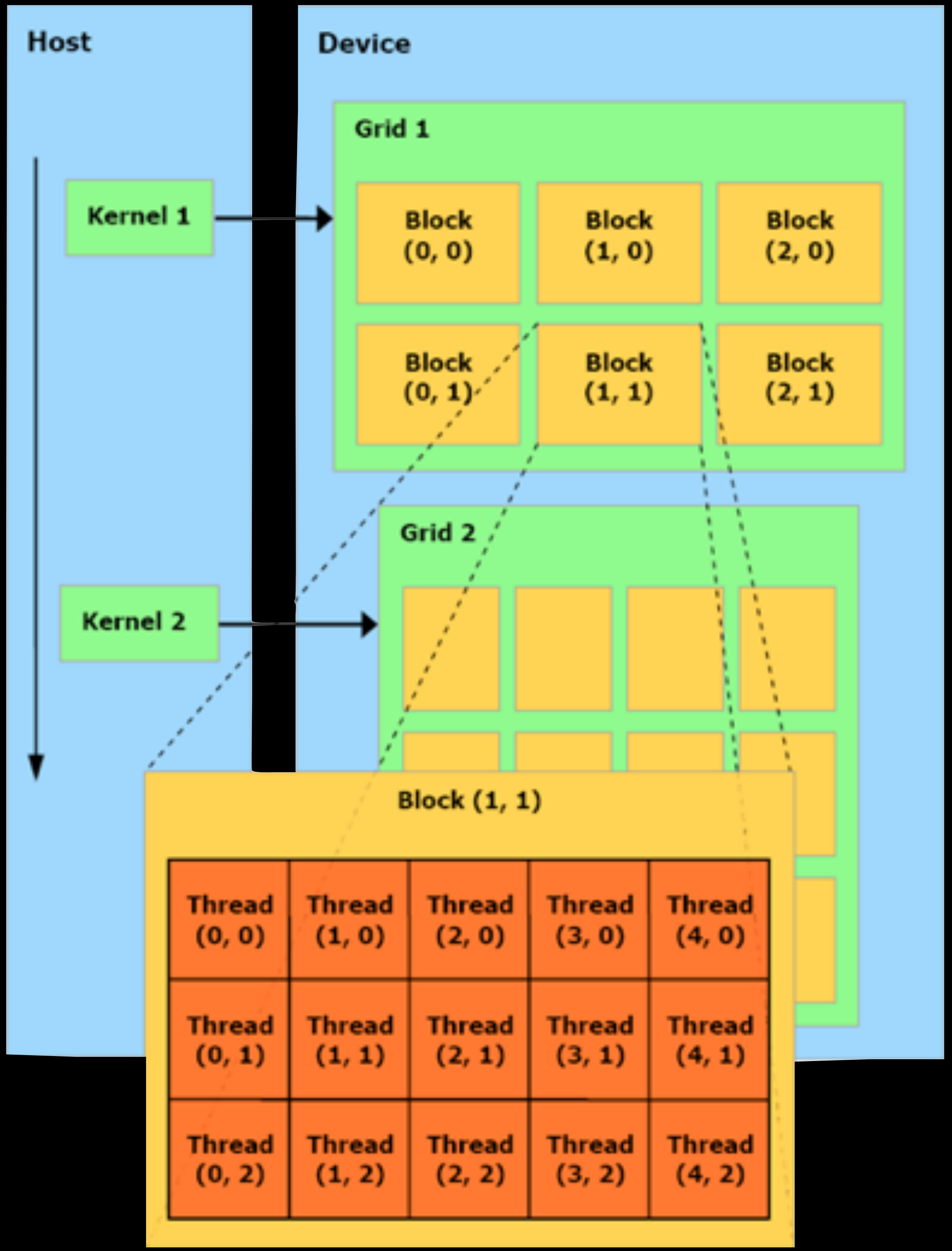


CUDA Programming

Simplified code

```
...
float *aHost, *bHost, *cHost;
...
__global__ void kernel(float *a, float *b, float *c){
     int i = threadidx.x;
     c[i] = a[i] + b[i];
}

int main(){
    float *aDev, *bDev, *cDev;

    cudaMalloc(void *aDev, 512 * sizeof(float));
    cudaMemcpy(aDev, aHost, 512 * sizeof(float));

    cudaMalloc(void *bDev, 512 * sizeof(float));
    cudaMemcpy(bDev, bHost, 512 * sizeof(float));

    kernel<<<1,512>>> (aDev, bDev, cDev);

    cudaFree(aDev); cudaFree(bDev); cudaFree(cDev);
}
```

CUDA Programming API



Thread indexing

Threads are organized in blocks

Blocks are organized in grids

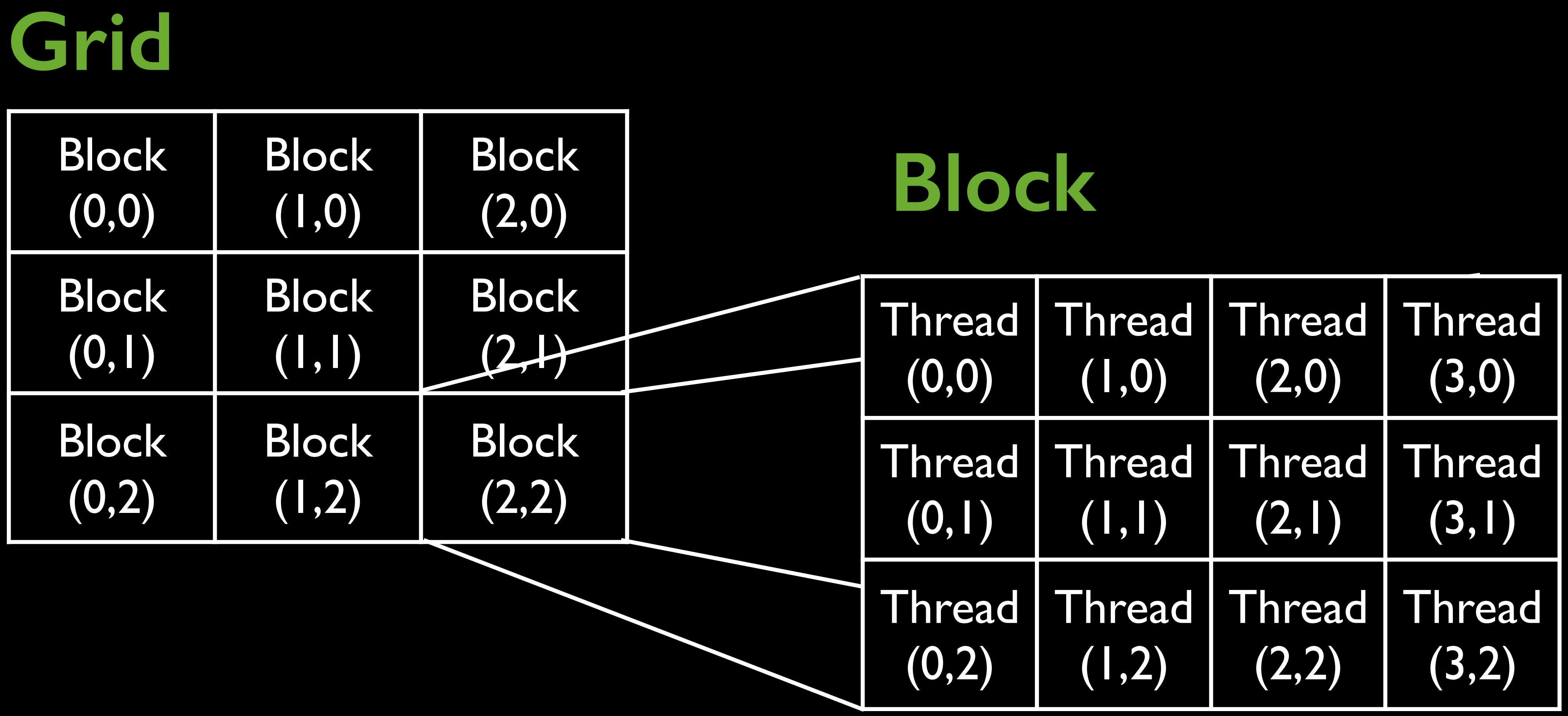
Legacy from CG applications

CUDA Threads

Grid

Block (0,0)	Block (1,0)	Block (2,0)
Block (0,1)	Block (1,1)	Block (2,1)
Block (0,2)	Block (1,2)	Block (2,2)

CUDA Threads



Grid

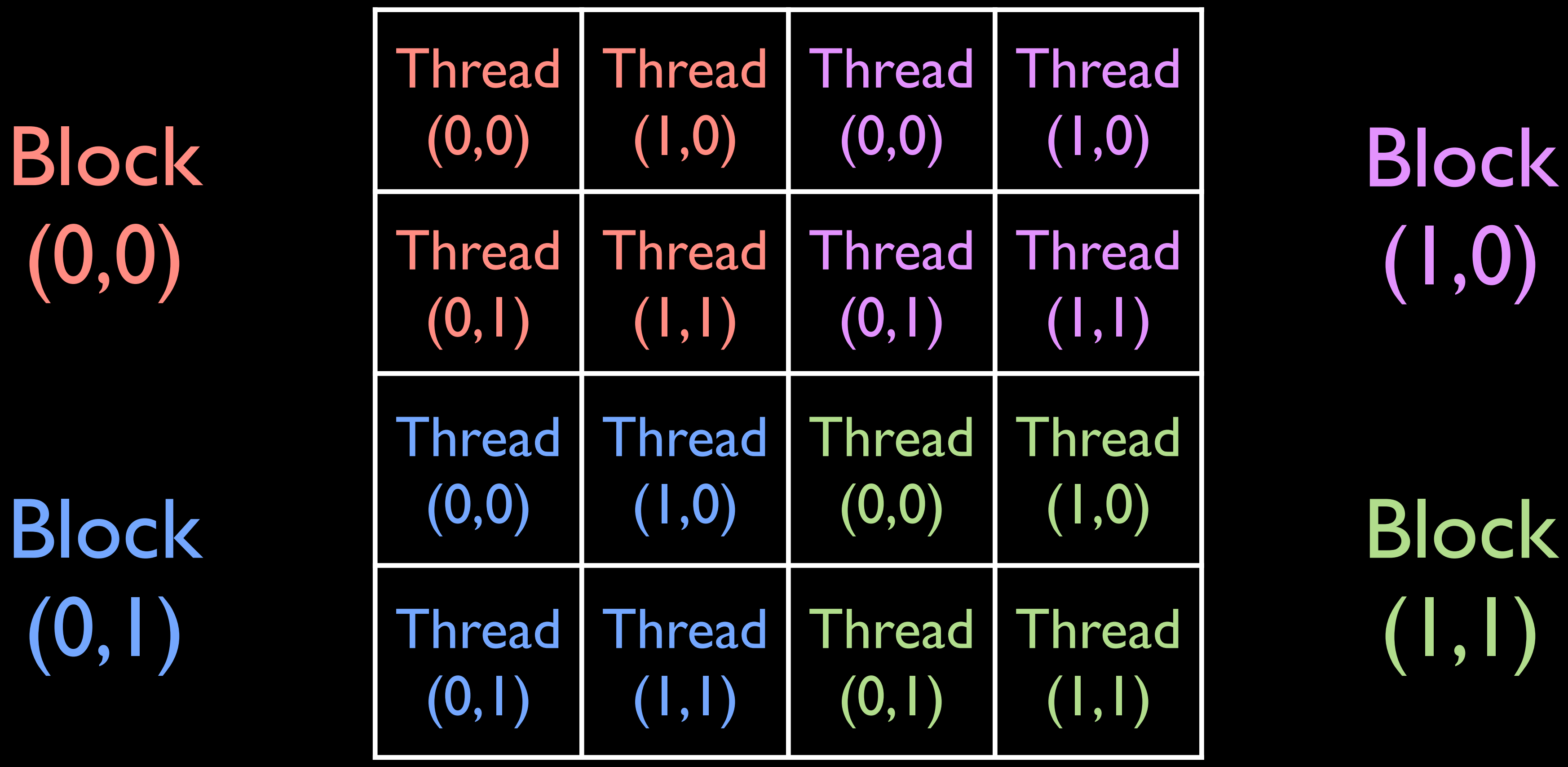
Block (0,0)	Block (1,0)	Block (2,0)
Block (0,1)	Block (1,1)	Block (2,1)
Block (0,2)	Block (1,2)	Block (2,2)

Block

Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)
Thread (0,1)	Thread (1,1)	Thread (2,1)	Thread (3,1)
Thread (0,2)	Thread (1,2)	Thread (2,2)	Thread (3,2)

CUDA Threads

mapping threads



Block (0,0)

Block (1,0)

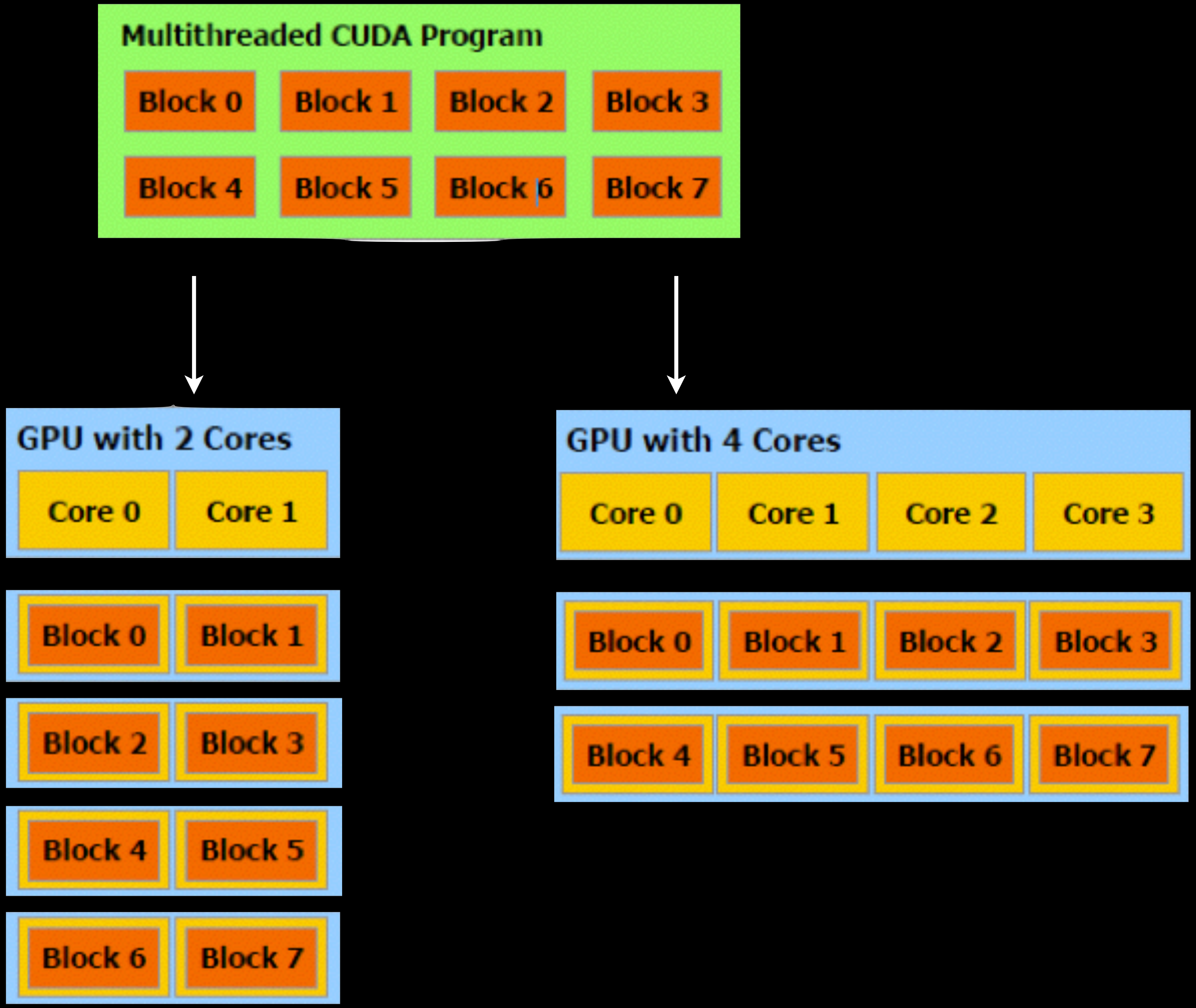
Block (0,1)

Block (1,1)

Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)
Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)
Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)
Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)

```
dim3 Grid(2,2);
dim3 Block(2,2);
kernel<<<Grid,Block>>>(parameters);
```

CUDA Threads



CUDA Threads

How can we arrange 6 threads?

Block (0,0)

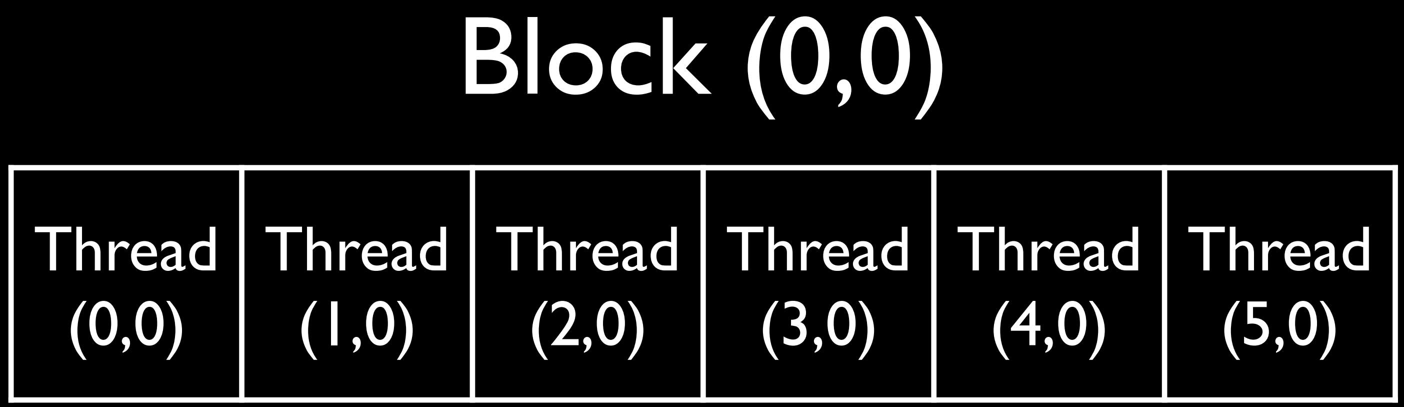
Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)	Thread (4,0)	Thread (5,0)

MAX
THREADS PER BLOCK
DEPEND ON THE
ARCHITECTURE

DEVICE QUERY

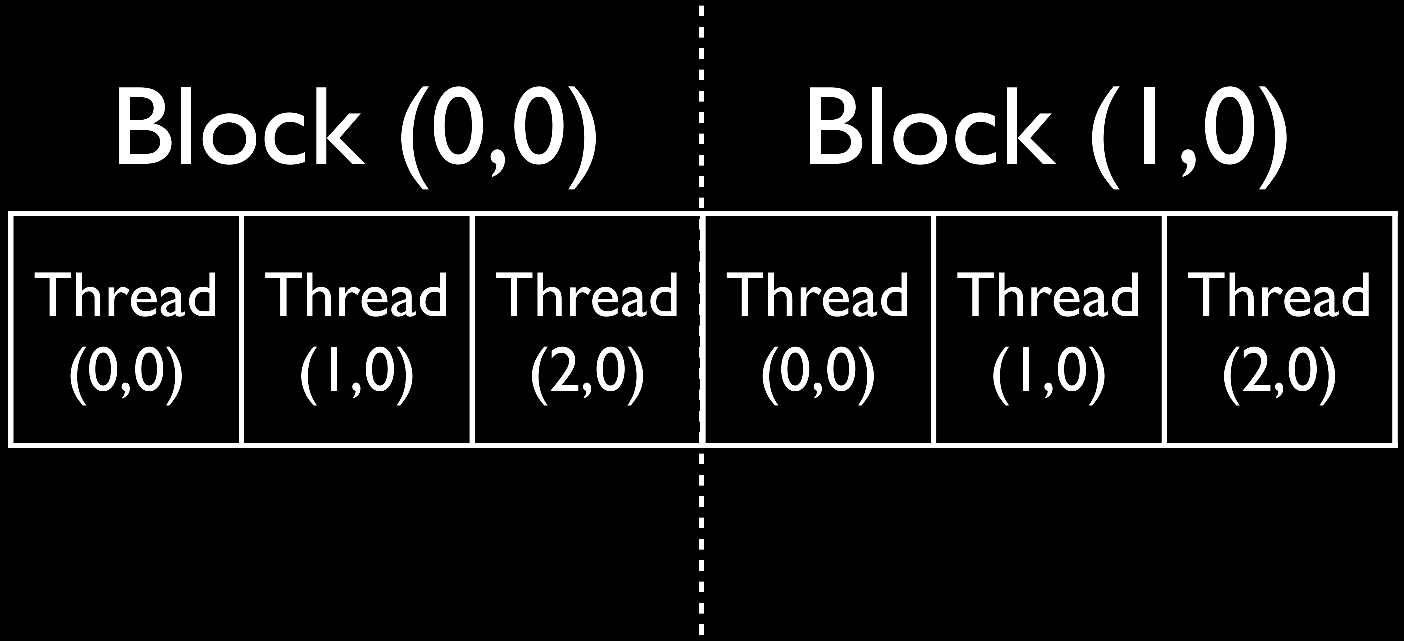
CUDA Threads

How can we arrange 6 threads?



Block (0,0)

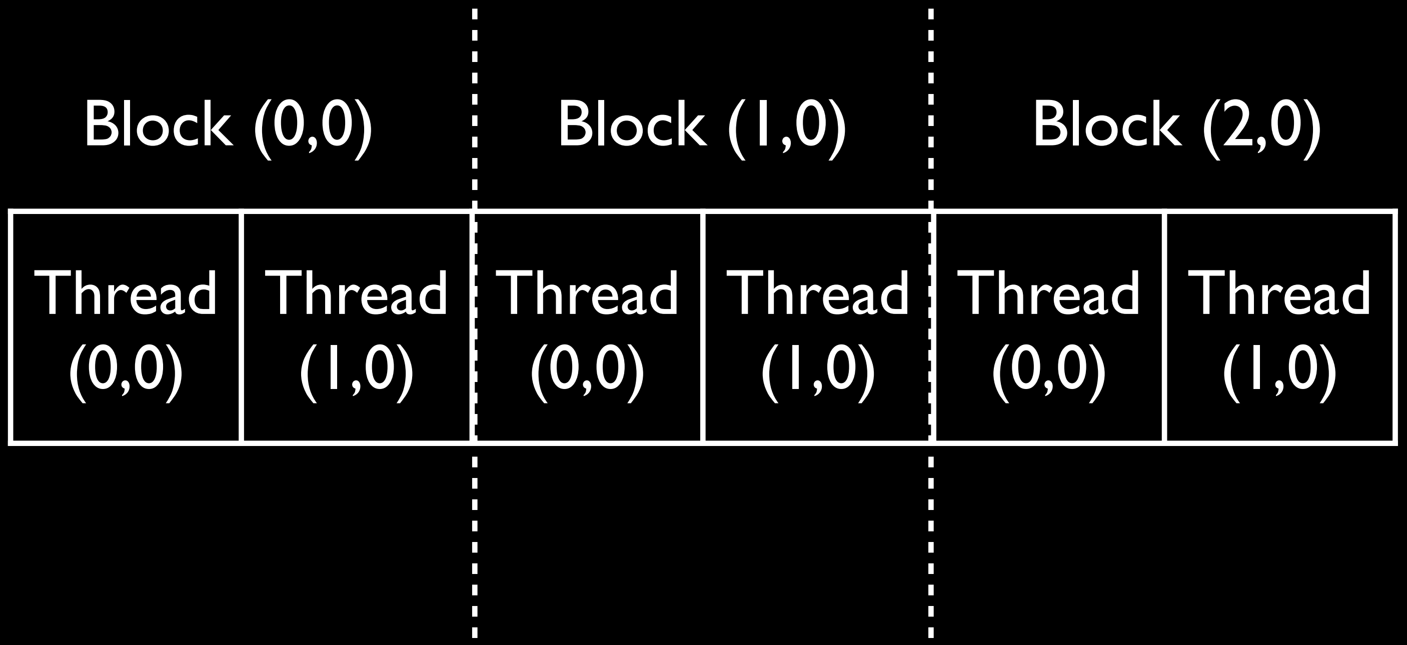
Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)	Thread (4,0)	Thread (5,0)



Block (0,0)			Block (1,0)		
Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (0,0)	Thread (1,0)	Thread (2,0)

CUDA Threads

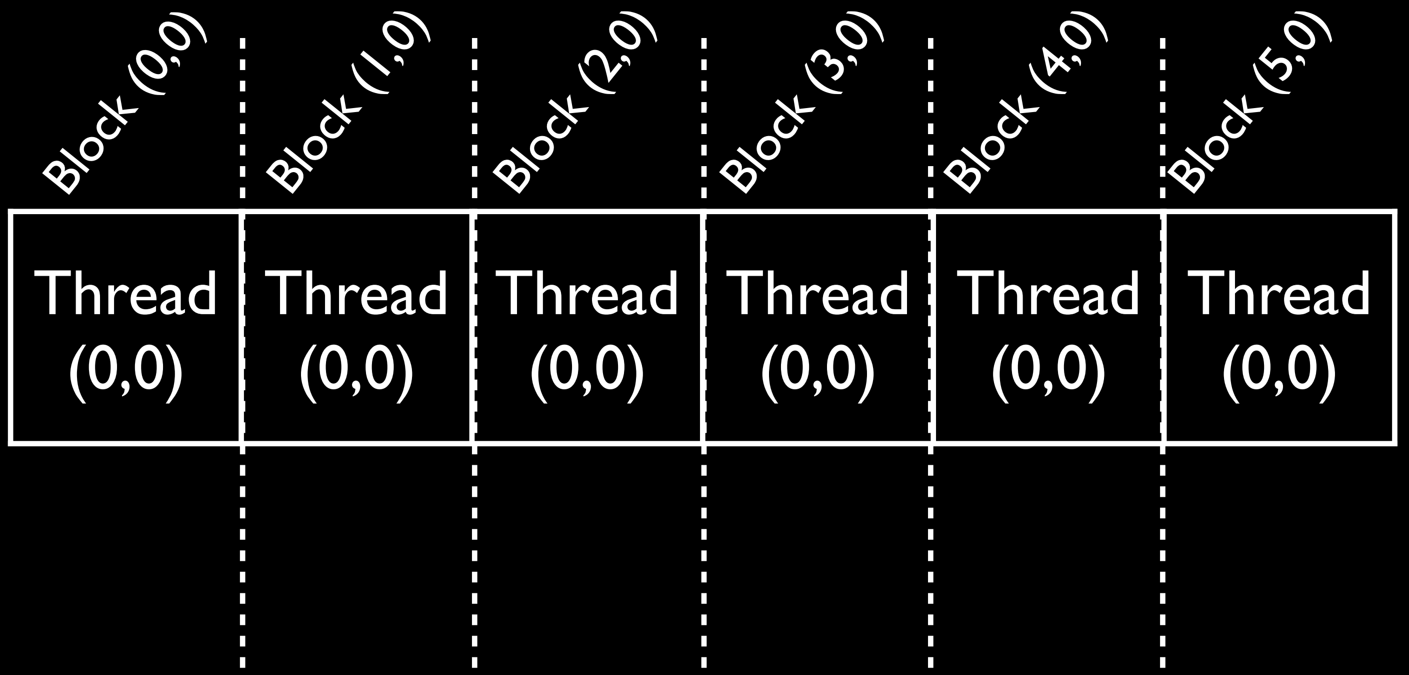
How can we arrange 6 threads?



Block (0,0)		Block (1,0)		Block (2,0)	
Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)

CUDA Threads

How can we arrange 6 threads?



Block (0,0)
Block (1,0)
Block (2,0)
Block (3,0)
Block (4,0)
Block (5,0)

Thread (0,0)
Thread (0,0)
Thread (0,0)
Thread (0,0)
Thread (0,0)
Thread (0,0)

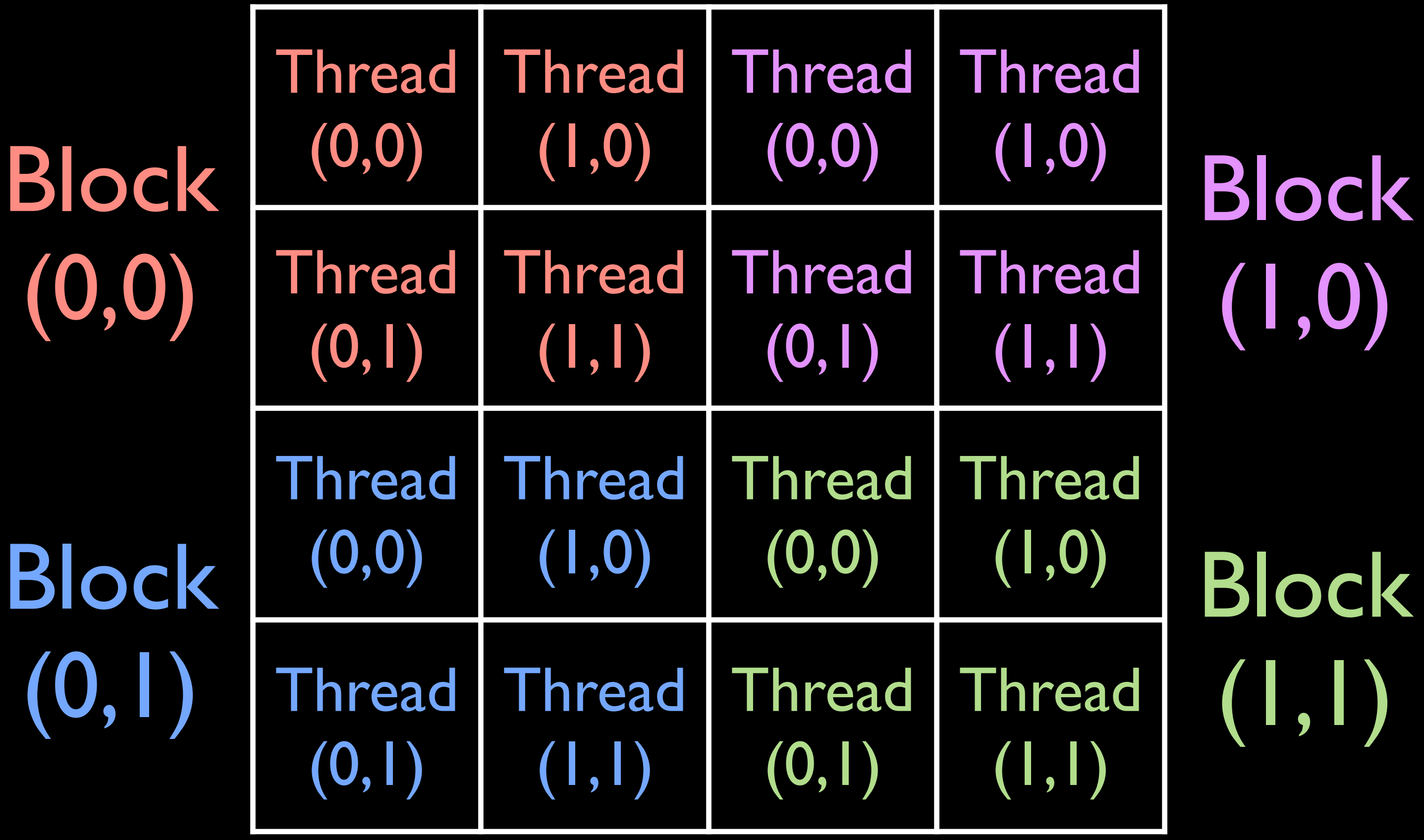
CUDA Threads

Mapping on an unique grid

Block (0,0)	Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)	Block (1,0)
	Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)	
Block (0,1)	Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)	Block (1,1)
	Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)	

CUDA Threads

Mapping on an unique grid



Block (0,0)

Block (1,0)

Block (0,1)

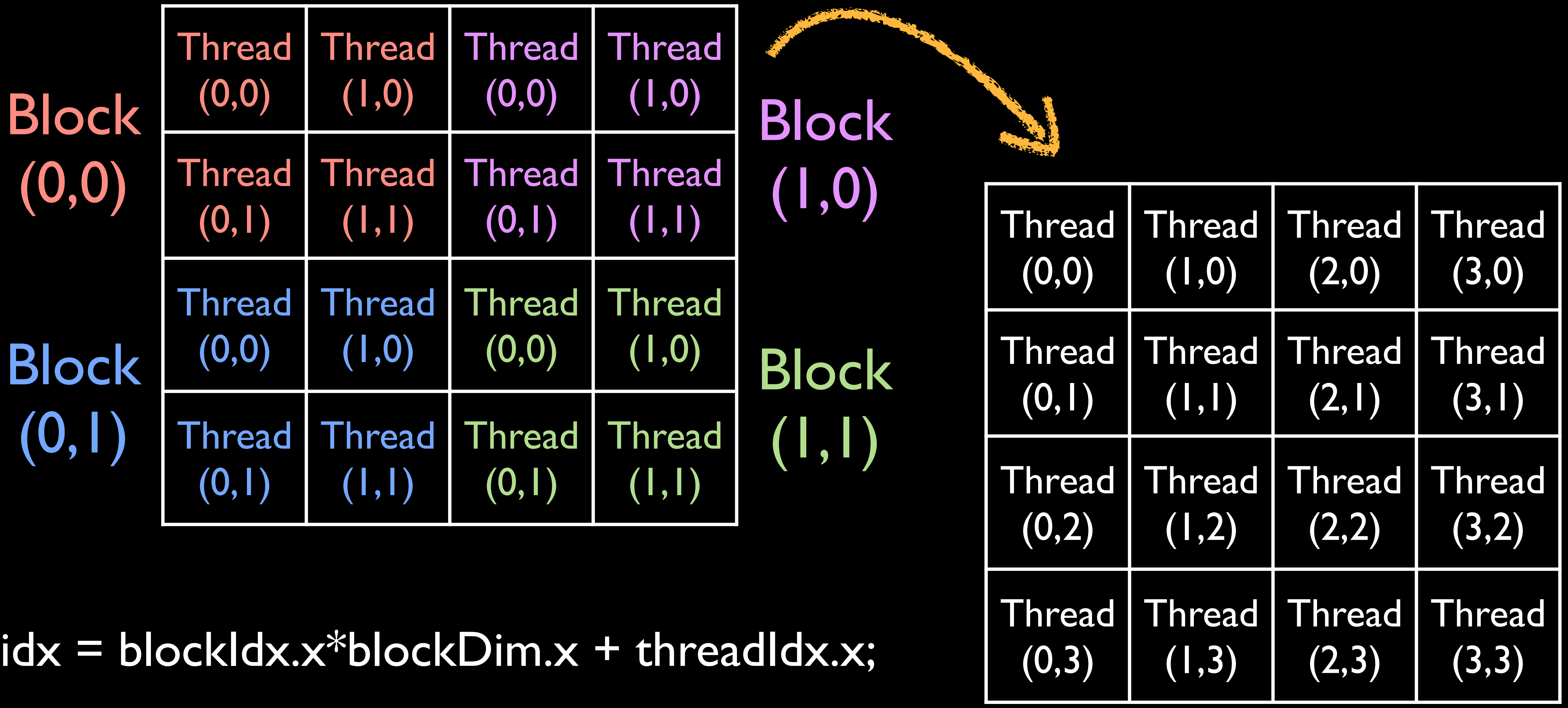
Block (1,1)

Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)
Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)
Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)
Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)

```
idx = blockIdx.x*blockDim.x + threadIdx.x;

idy = blockIdx.y*blockDim.y + threadIdx.y;
```

CUDA Threads

Mapping on an unique grid



Block (0,0)	Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)	Block (1,0)
	Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)	
Block (0,1)	Thread (0,0)	Thread (1,0)	Thread (0,0)	Thread (1,0)	Block (1,1)
	Thread (0,1)	Thread (1,1)	Thread (0,1)	Thread (1,1)	

Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)
Thread (0,1)	Thread (1,1)	Thread (2,1)	Thread (3,1)
Thread (0,2)	Thread (1,2)	Thread (2,2)	Thread (3,2)
Thread (0,3)	Thread (1,3)	Thread (2,3)	Thread (3,3)

```
idx = blockIdx.x*blockDim.x + threadIdx.x;

idy = blockIdx.y*blockDim.y + threadIdx.y;
```

CUDA Threads

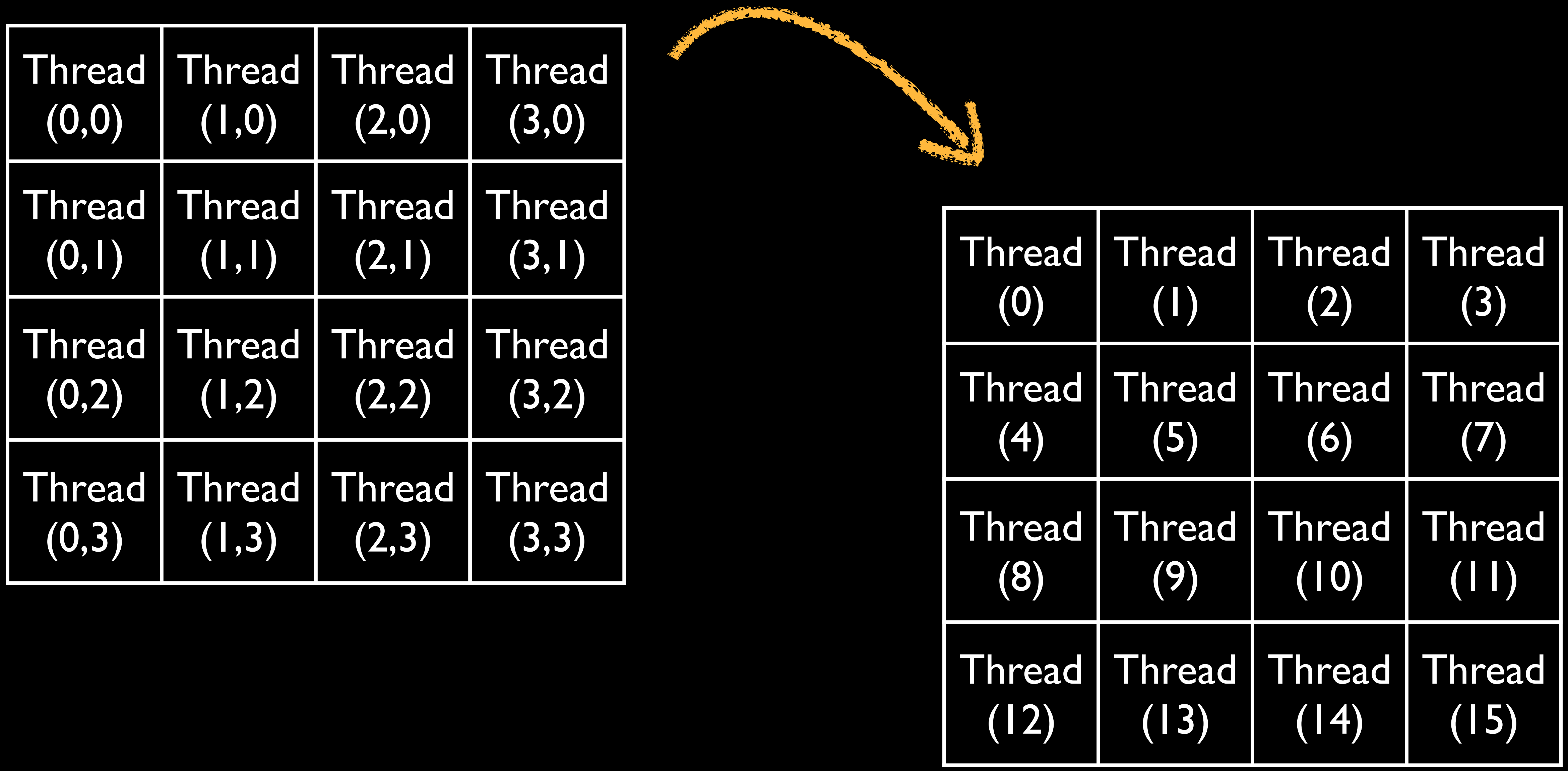
Get an unique thread index

Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)
Thread (0,1)	Thread (1,1)	Thread (2,1)	Thread (3,1)
Thread (0,2)	Thread (1,2)	Thread (2,2)	Thread (3,2)
Thread (0,3)	Thread (1,3)	Thread (2,3)	Thread (3,3)

```
k = idx + idy*blockDim.x*gridDim.x;
```

CUDA Threads

Get an unique thread index



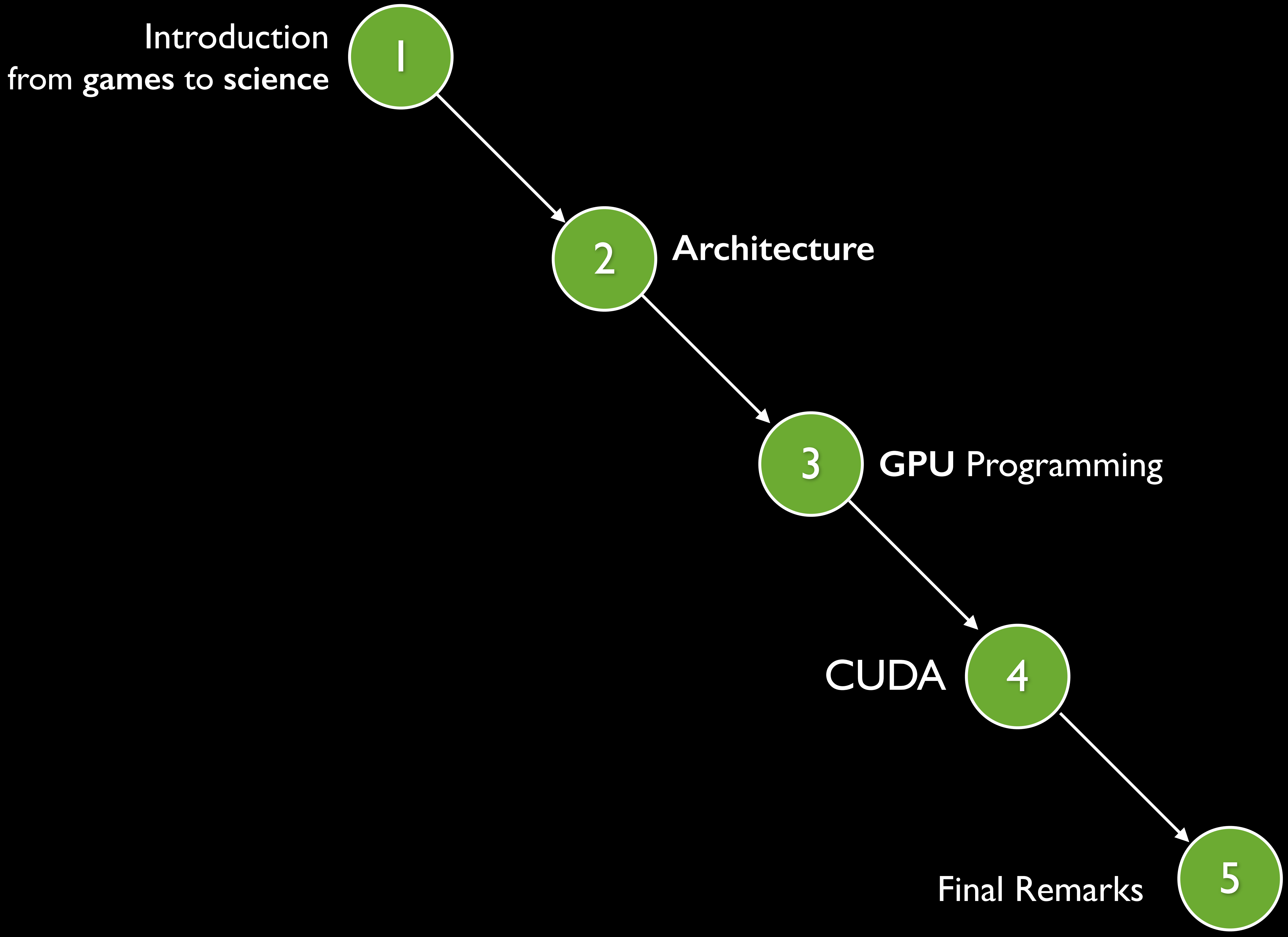
Thread (0,0)	Thread (1,0)	Thread (2,0)	Thread (3,0)
Thread (0,1)	Thread (1,1)	Thread (2,1)	Thread (3,1)
Thread (0,2)	Thread (1,2)	Thread (2,2)	Thread (3,2)
Thread (0,3)	Thread (1,3)	Thread (2,3)	Thread (3,3)

Thread (0)	Thread (1)	Thread (2)	Thread (3)
Thread (4)	Thread (5)	Thread (6)	Thread (7)
Thread (8)	Thread (9)	Thread (10)	Thread (11)
Thread (12)	Thread (13)	Thread (14)	Thread (15)

```
k = idx + idy*blockDim.x*gridDim.x;
```

Exercise

Implementing the sum of two vectors using CUDA of a unlimited number of elements.



1 Introduction
from games to science

2 Architecture

3 GPU Programming

4 CUDA

5 Final Remarks

CUDA Programming

SDK has many applications:

```
$ cd $NVIDIA_CUDA_SDK

$ make

$ make check

$ C/bin/linux/release/
```

GPU is good for...

loosely coupled threads (avoid synchronisation)

computing bound applications

these architectures can not replace general purpose CPU

great insight for future architectures

CUDA Pros

Support for several OS

A lot of documentation

Many libraries available

Great performance

CUDA Cons

NVIDIA proprietary

Architectures of Today

Highly heterogeneous

AMD Fusion (APU)
CPU + GPU

Architectures of Today

Highly heterogeneous

NVIA Tegra
ARM + GPU

Architectures of Today

Highly heterogeneous

Intel Xeon Phi

“ Moving a code to Intel Xeon Phi might involve sitting down and adding a couple lines of directives that takes a few minutes. Moving a code to a GPU is a project. ”

Dan Stanzione, Deputy Director at Texas Advanced Computing Center

The Intel® Xeon® Phi™ Coprocessor: Parallel Processing, Unparalleled Discovery

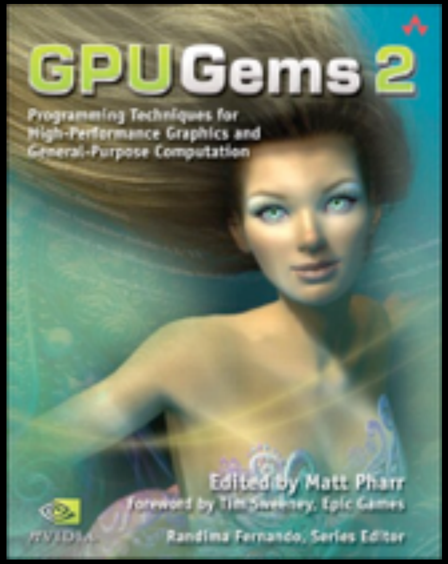
From intel's website

Further help

CUDA Developer Network

http://developer.download.nvidia.com/compute/cuda/4_1/rel/toolkit/docs/online/group__CUDART__MEMORY_g48efa06b81cc031b2aa6fdc2e9930741.html

Bibliography



GPU Gems 2, available online

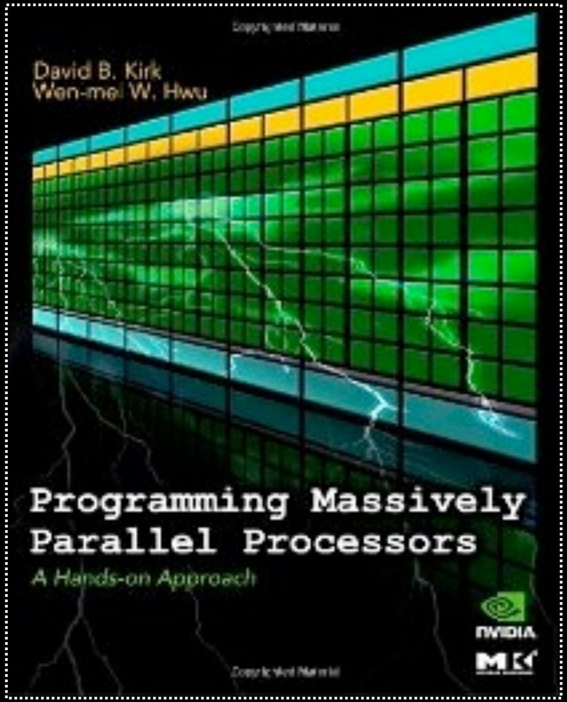
http://http.developer.nvidia.com/GPUGems2/gpugems2_part01.html

GPU Gems 3, available online

https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_pref01.html

Bibliography

Programming Massively Parallel Processors: A Hands-on Approach, David B. Kirk and Wen-Mei Hwu, Second Edition, Morgan Kaufmann, 2009



NVIDIA developer zone, http://developer.nvidia.com/

Exercise IV
Naïve matrix multiplication on GPU.

http://www.es.ele.tue.nl/~mwijtvliet/5KK73/?page=mmcuda