GPU Programming

Introduction and Historical Notes

Franco Milicchio

https://fmilicchio.bitbucket.io

A Perspective

A Perspective

- Parallel computing is ubiquitous, but it is *not new at all*

A Perspective

- Parallel computing is ubiquitous, but it is *not new at all*
- Astonishingly, the *Analytical Engine* by Charles Babbage in ~1837 was a parallel computer

A Perspective

- Parallel computing is ubiquitous, but it is *not new at all*
- Astonishingly, the *Analytical Engine* by Charles Babbage in ~1837 was a parallel computer
- In 1958 parallel computing was introduced by S. Gill introducing *branching and waiting*, while at IBM J. Cocke and D. Slotnick discussed *parallelism in numerical calculations*

A Perspective

- Parallel computing is ubiquitous, but it is *not new at all*

- Astonishingly, the *Analytical Engine* by Charles Babbage in ~1837 was a parallel computer

- In 1958 parallel computing was introduced by S. Gill introducing *branching and waiting*, while at IBM J. Cocke and D. Slotnick discussed *parallelism in numerical calculations*

- The D825 by Burroughs Corporation, introduced n 1962, was the first *modern* parallel computer with four-processors

History of GPUs

History of GPUs

- In the 1970s computers were bulky and graphics was completely neglected

History of GPUs

- In the 1970s computers were bulky and graphics was completely neglected

- One sector, however, *was* interested in graphics: games

History of GPUs

- In the 1970s computers were bulky and graphics was completely neglected
- One sector, however, *was* interested in graphics: games
- The industry was driven by *arcade games*, *e.g.*, Namco, Taito, and many others

History of GPUs

- In the 1970s computers were bulky and graphics was completely neglected

- One sector, however, *was* interested in graphics: games

- The industry was driven by *arcade games*, *e.g.*, Namco, Taito, and many others

- The 1980s brought a great innovation with the Amiga PC

History of GPUs

- In the 1970s computers were bulky and graphics was completely neglected

- One sector, however, *was* interested in graphics: games

- The industry was driven by *arcade games*, *e.g.*, Namco, Taito, and many others

- The 1980s brought a great innovation with the Amiga PC

- Things changed greatly in the 1990s

S3 Virge

My first graphics card (1991)

The World is 3D

The World is 3D

- The 1990s saw a plethora of graphics cards with different performances (S3, ATI, Matrox, Nvidia, …)

The World is 3D

- The 1990s saw a plethora of graphics cards with different performances (S3, ATI, Matrox, Nvidia, …)

- They were *fixed functions*, hence you had to write games specifically for each of them

The World is 3D

- The 1990s saw a plethora of graphics cards with different performances (S3, ATI, Matrox, Nvidia, …)

- They were *fixed functions*, hence you had to write games specifically for each of them

- Then, progressively, fixed functions were replaced by *programmable functions*

The World is 3D

- The 1990s saw a plethora of graphics cards with different performances (S3, ATI, Matrox, Nvidia, …)

- They were *fixed functions*, hence you had to write games specifically for each of them

- Then, progressively, fixed functions were replaced by *programmable functions*

- Now we know them by the term “GPUs” (thanks to Nvidia’s marketing team)

How Speed Changed

How Speed Changed

- Over the last decades, CPUs relied on clock speed to increase performances

How Speed Changed

- Over the last decades, CPUs relied on clock speed to increase performances
- There is a recent *flat rate of growth* among all CPU vendors

How Speed Changed

- Over the last decades, CPUs relied on clock speed to increase performances
- There is a recent *flat rate of growth* among all CPU vendors
- As it’s now known in industry: the free lunch is over

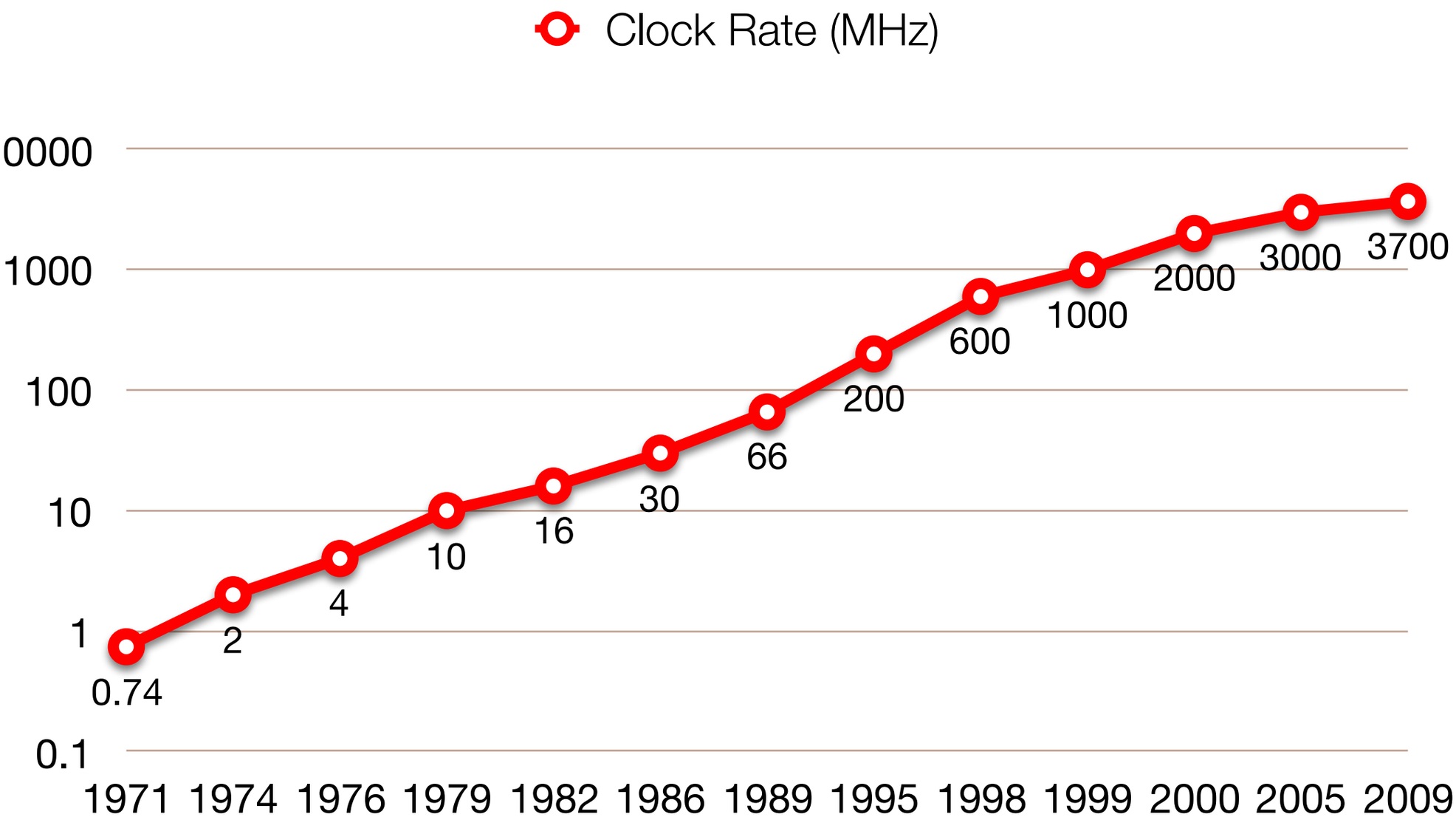
How Speed Changed

- Over the last decades, CPUs relied on clock speed to increase performances
- There is a recent *flat rate of growth* among all CPU vendors
- As it's now known in industry: the free lunch is over
- An increasing GPU market propelled investments in research in many areas

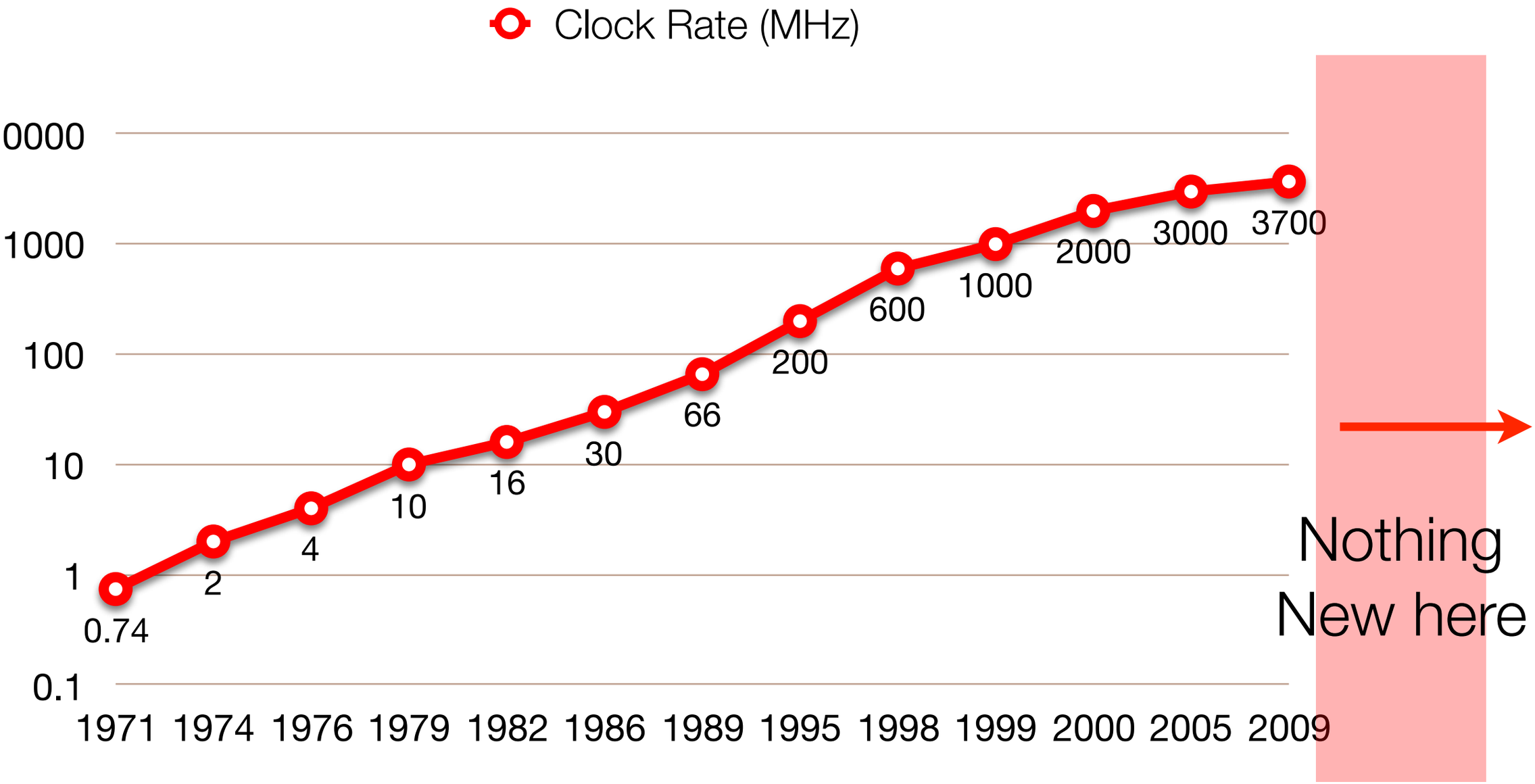
How Speed Changed

- Over the last decades, CPUs relied on clock speed to increase performances
- There is a recent *flat rate of growth* among all CPU vendors
- As it’s now known in industry: the free lunch is over
- An increasing GPU market propelled investments in research in many areas
- The trend is obviously rising for GPUs, flat for CPUs

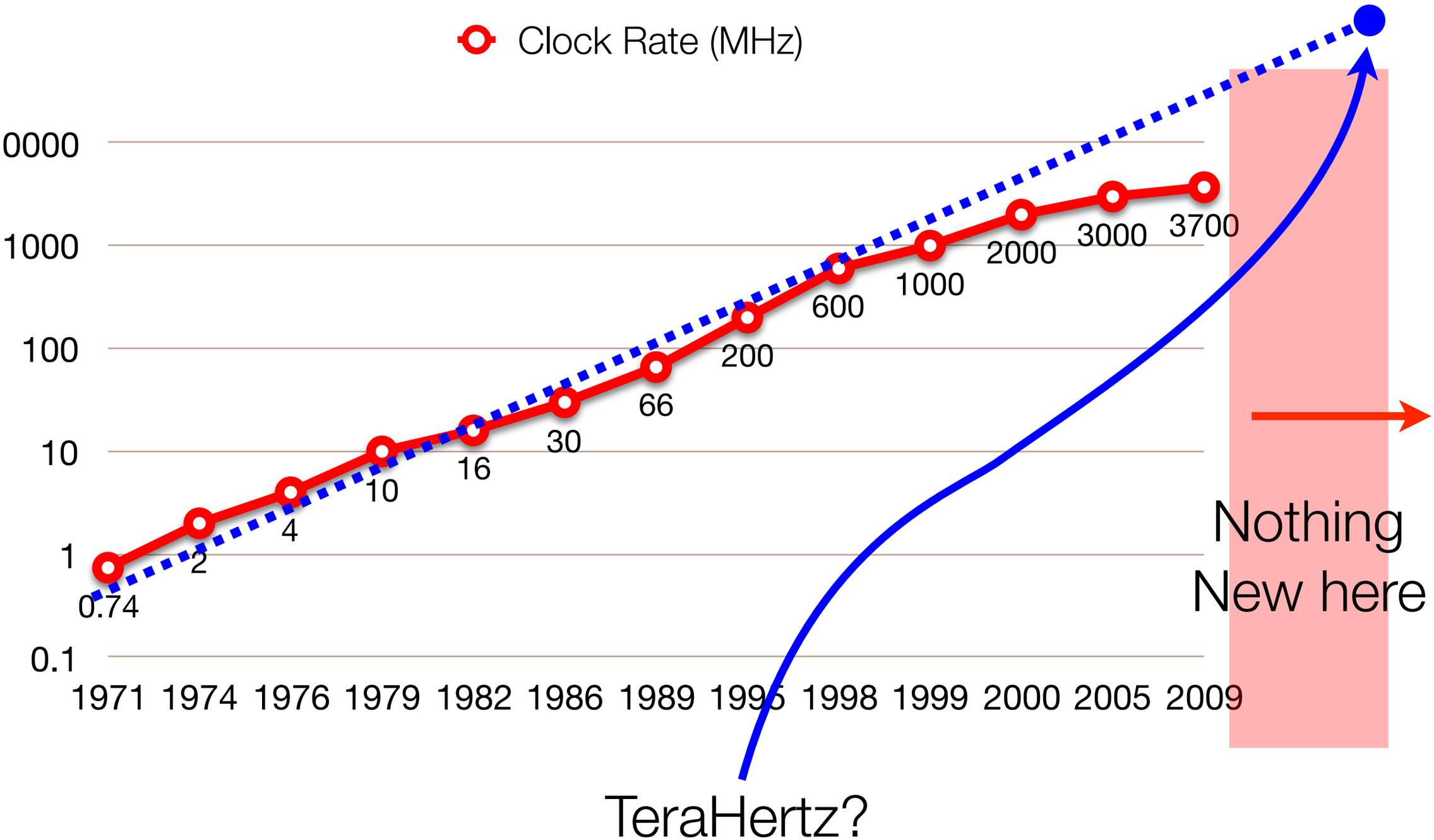
CPU Clock Rate



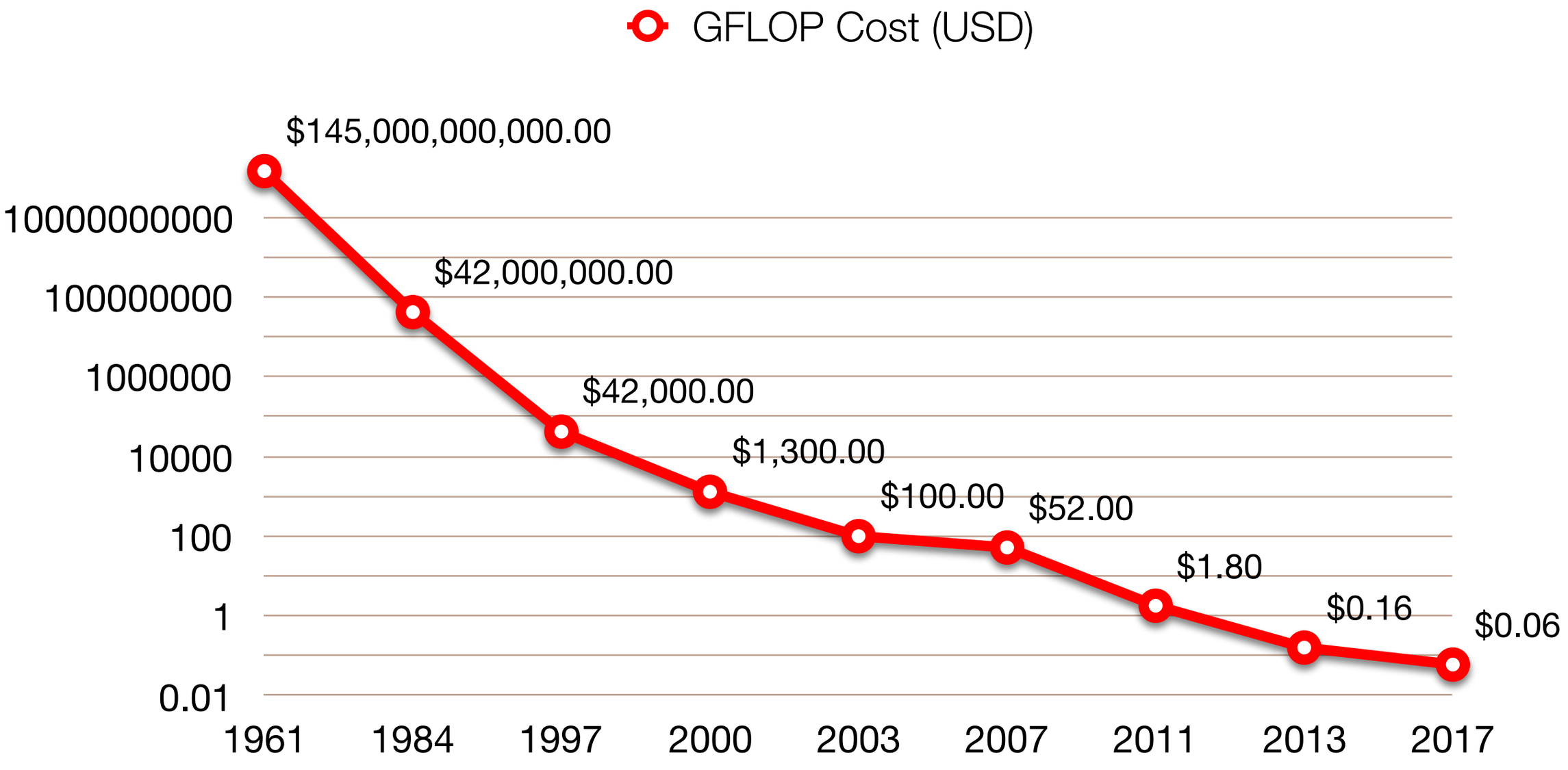
CPU Clock Rate



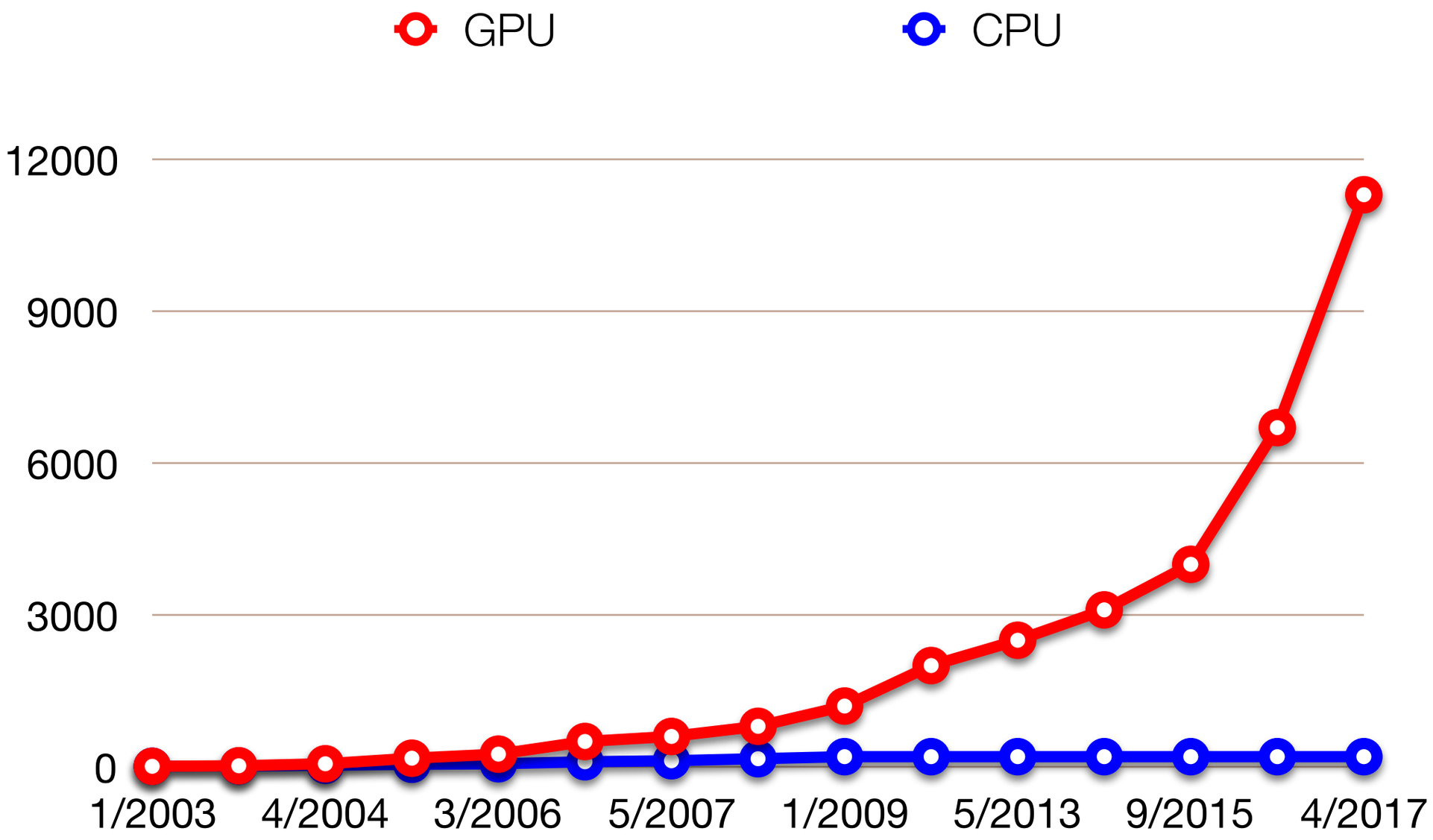
CPU Clock Rate



Cost of Performances



GFLOPs


GPU
CPU
12000
9000
6000
3000
0
1/2003
4/2004
3/2006
5/2007
1/2009
5/2013
9/2015
4/2017

Who uses GPUs?

TL;DR:
everyone, even if you don’t know

Speedup

Speedup

- How fast we can go is highly dependent on how we implement an algorithm

Speedup

- How fast we can go is highly dependent on how we implement an algorithm

- Physical architectures for high performances *matter a lot* and without that knowledge *you won't be fast*

Speedup

- How fast we can go is highly dependent on how we implement an algorithm

- Physical architectures for high performances *matter a lot* and without that knowledge *you won't be fast*

- Let's see now how a GPU works

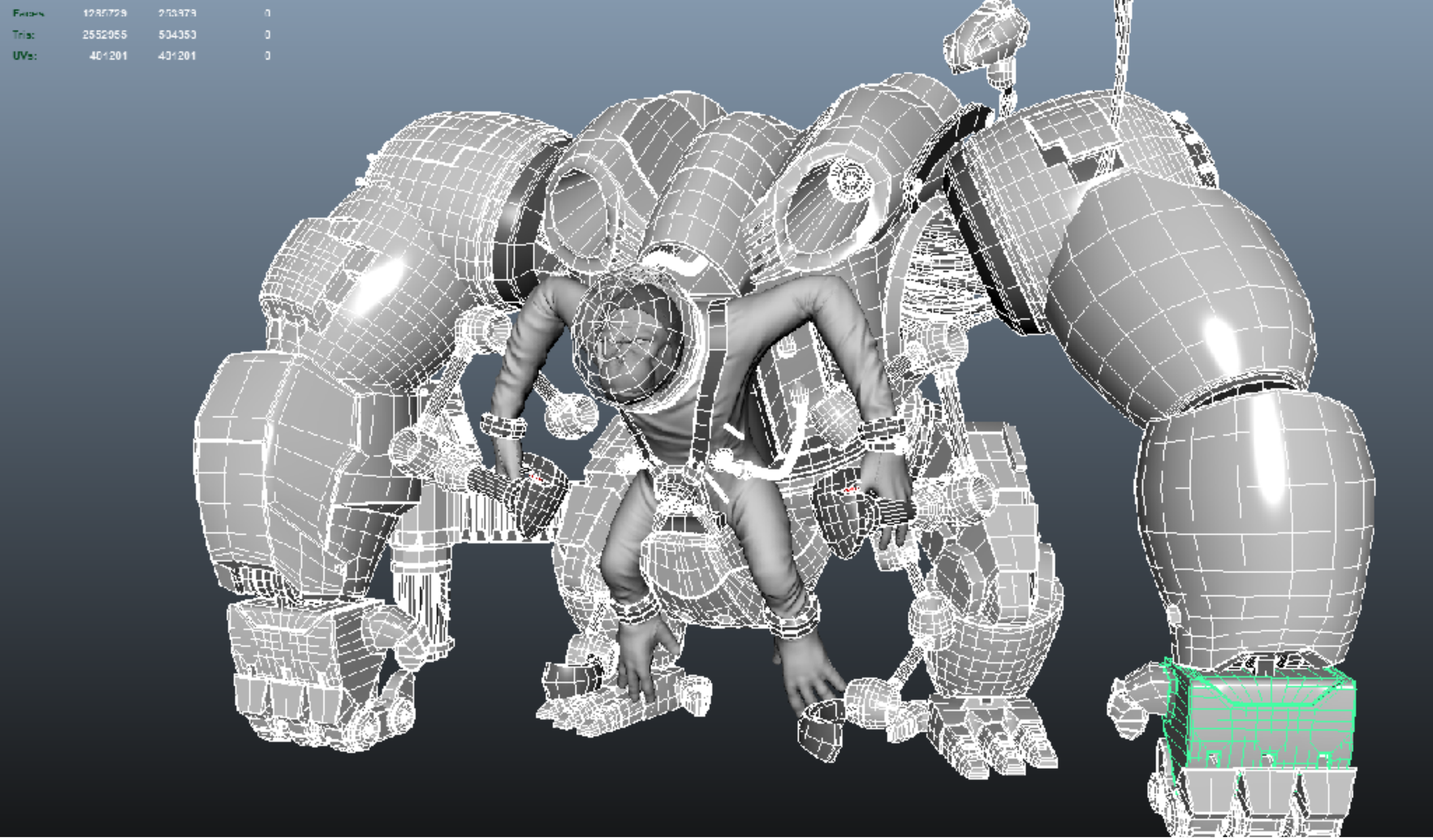
Speedup

- How fast we can go is highly dependent on how we implement an algorithm

- Physical architectures for high performances *matter a lot* and without that knowledge *you won't be fast*

- Let's see now how a GPU works

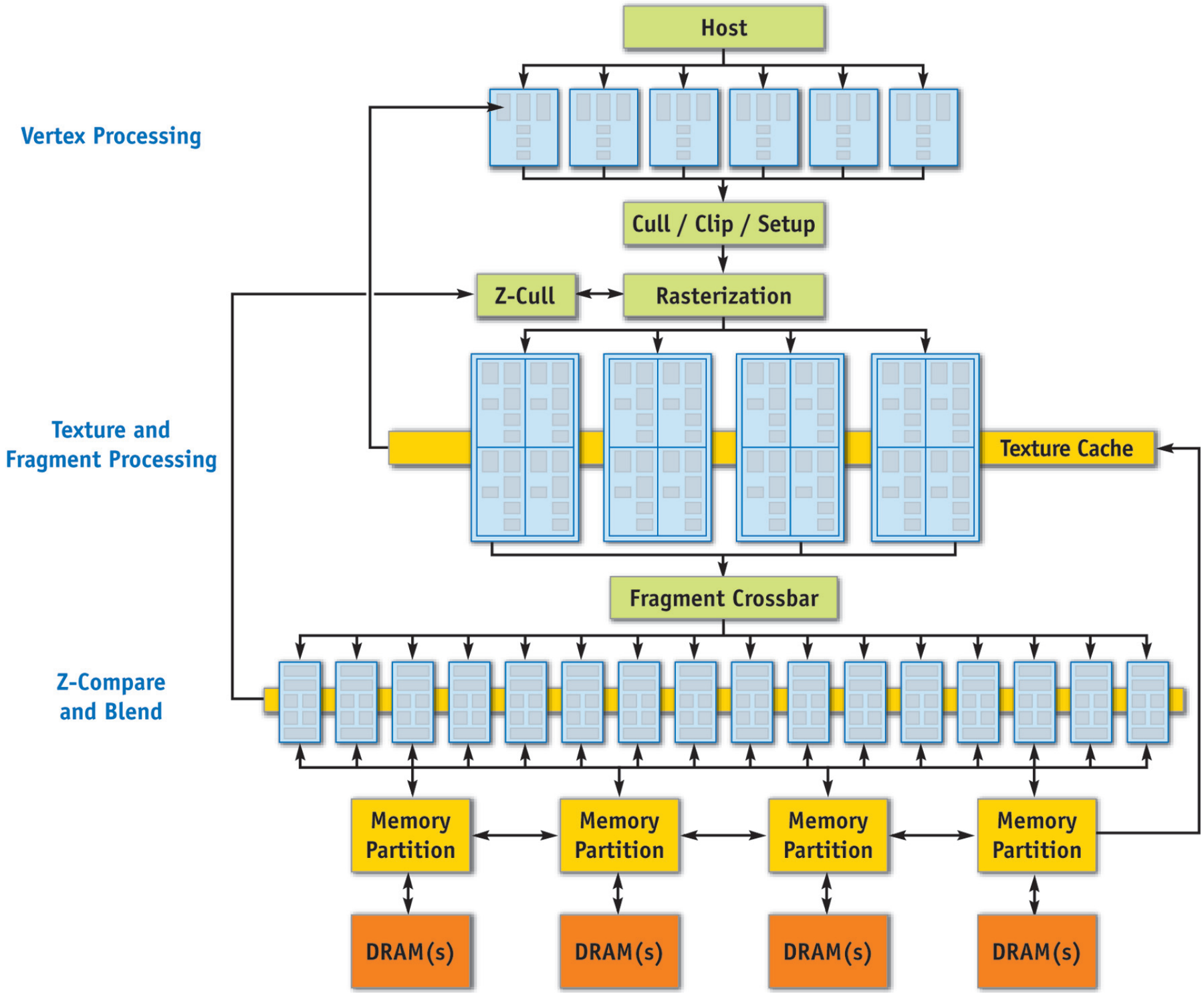
- After, we will see how we can use GPU to achieve way more of just graphics



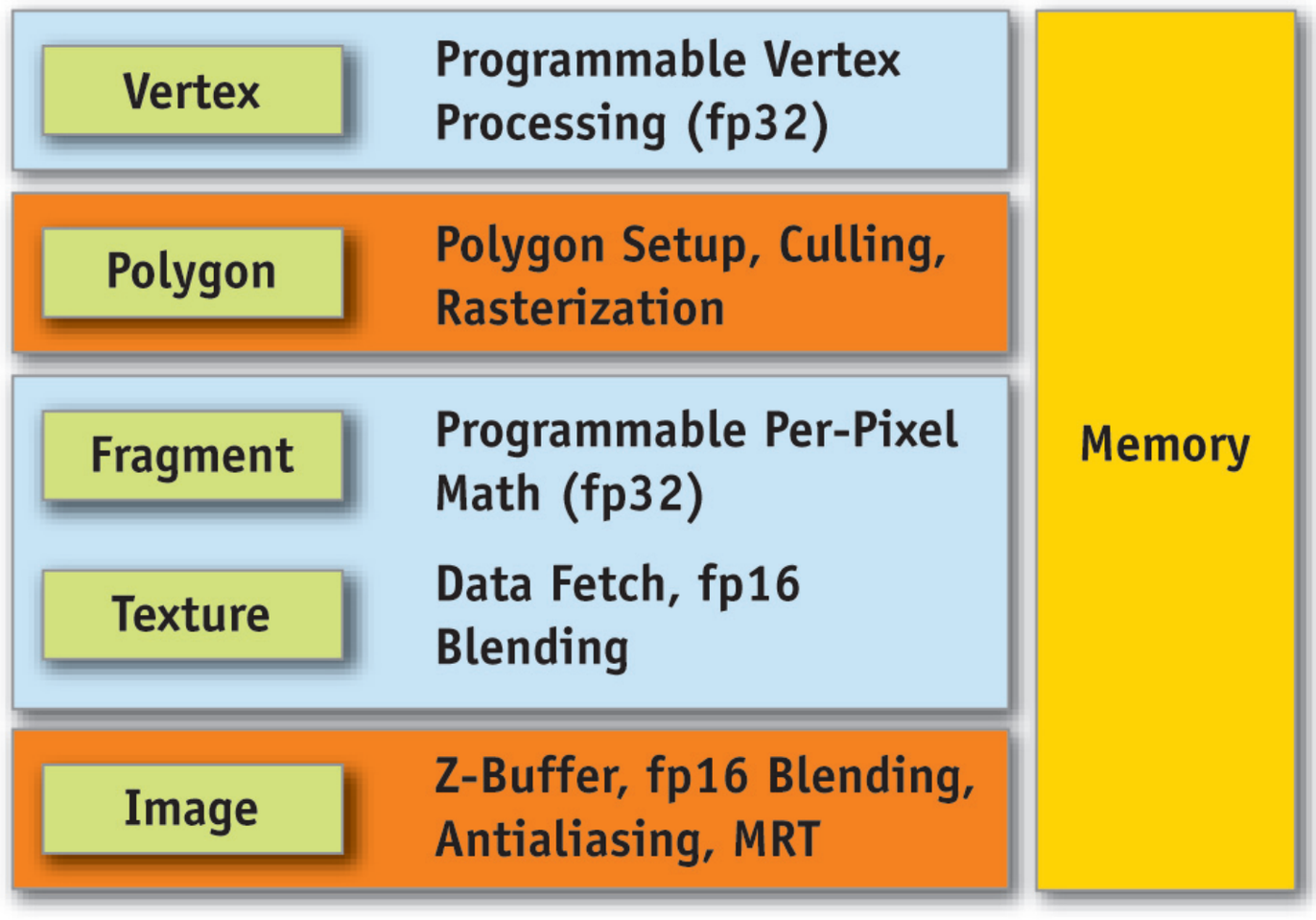
Vertices, Cells, Textures

This mesh will be rendered into pixels

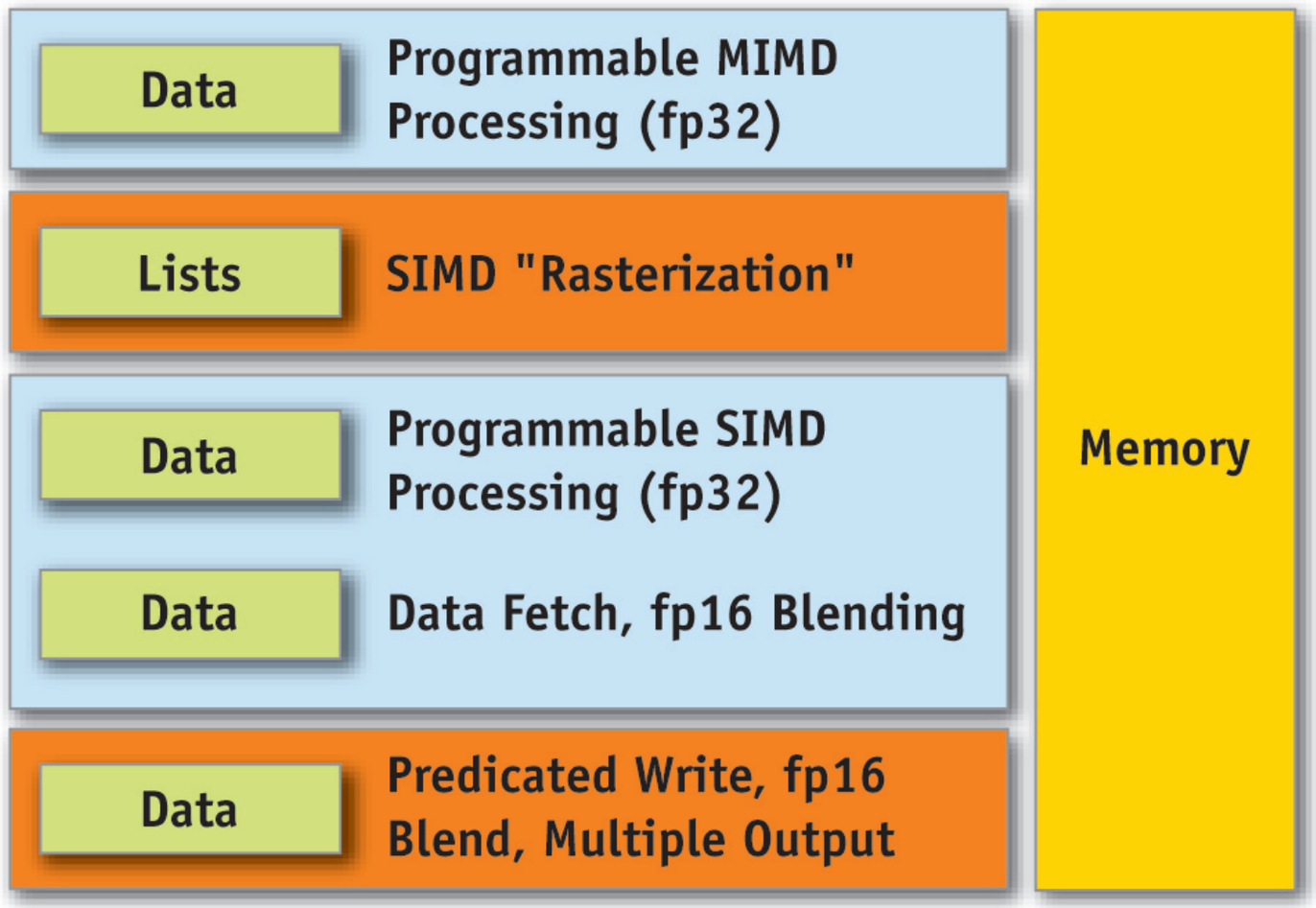
GPU Architecture



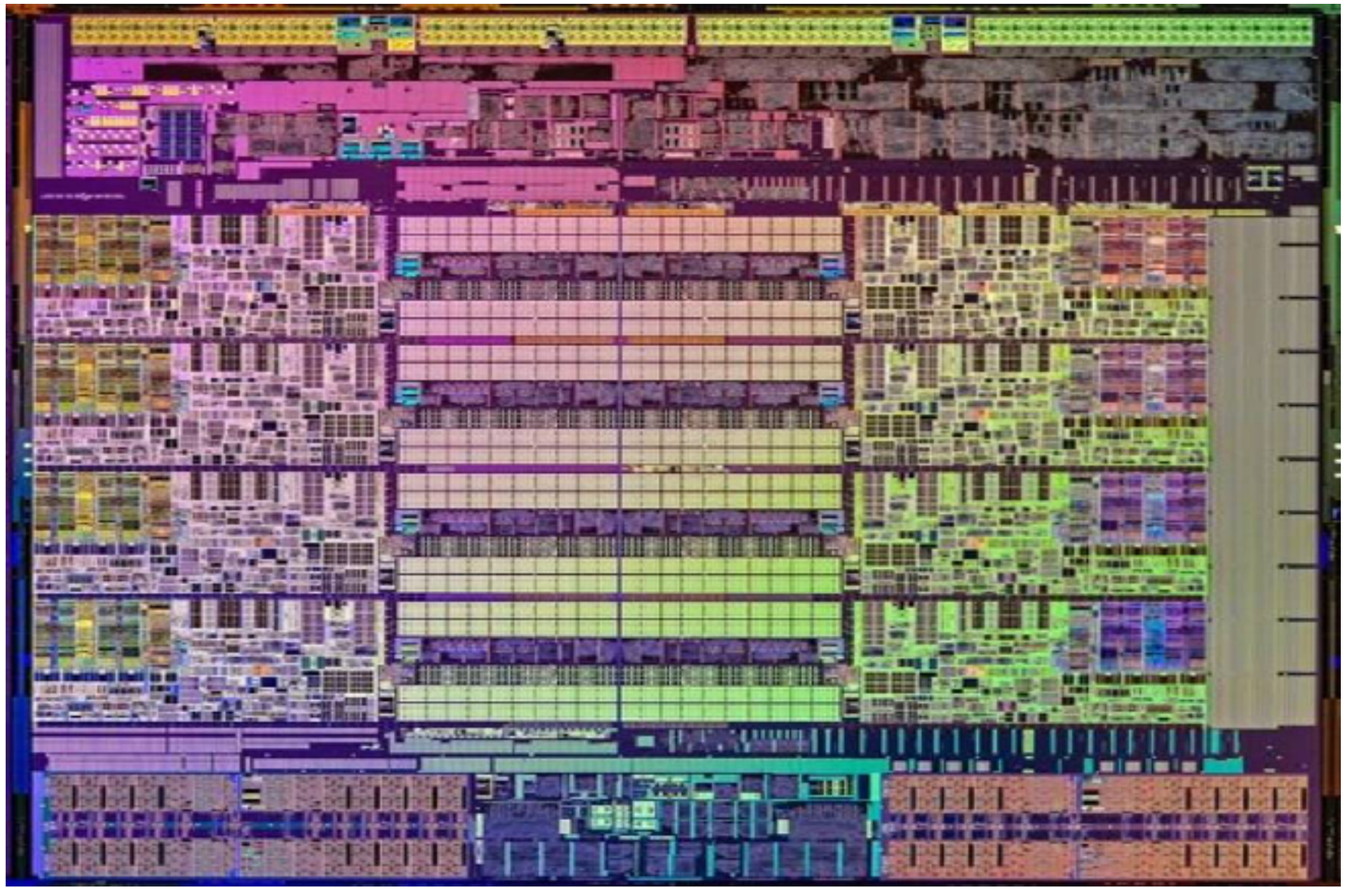
Graphics Pipeline



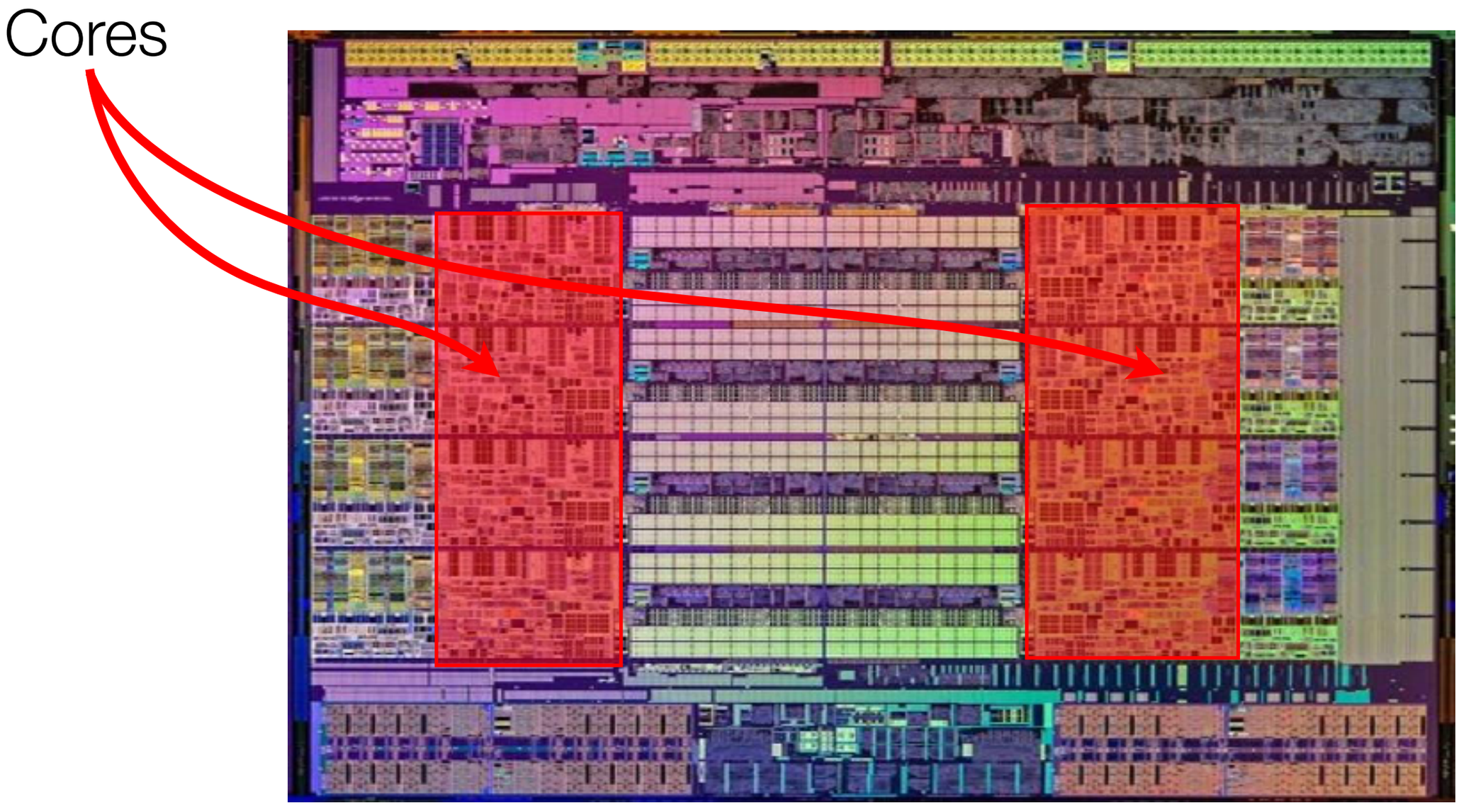
Graphics Pipeline



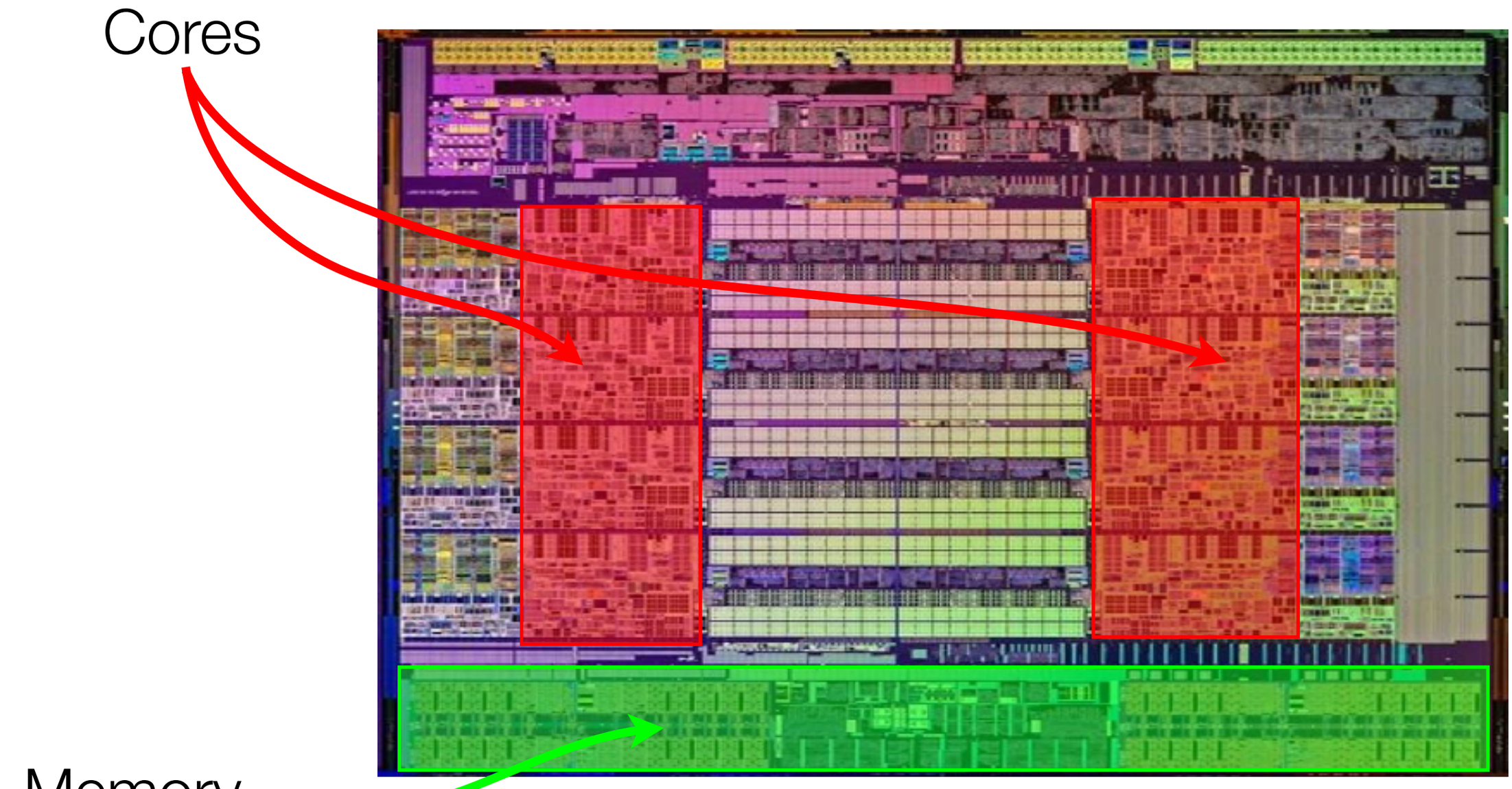
CPU Architecture (Intel i7)



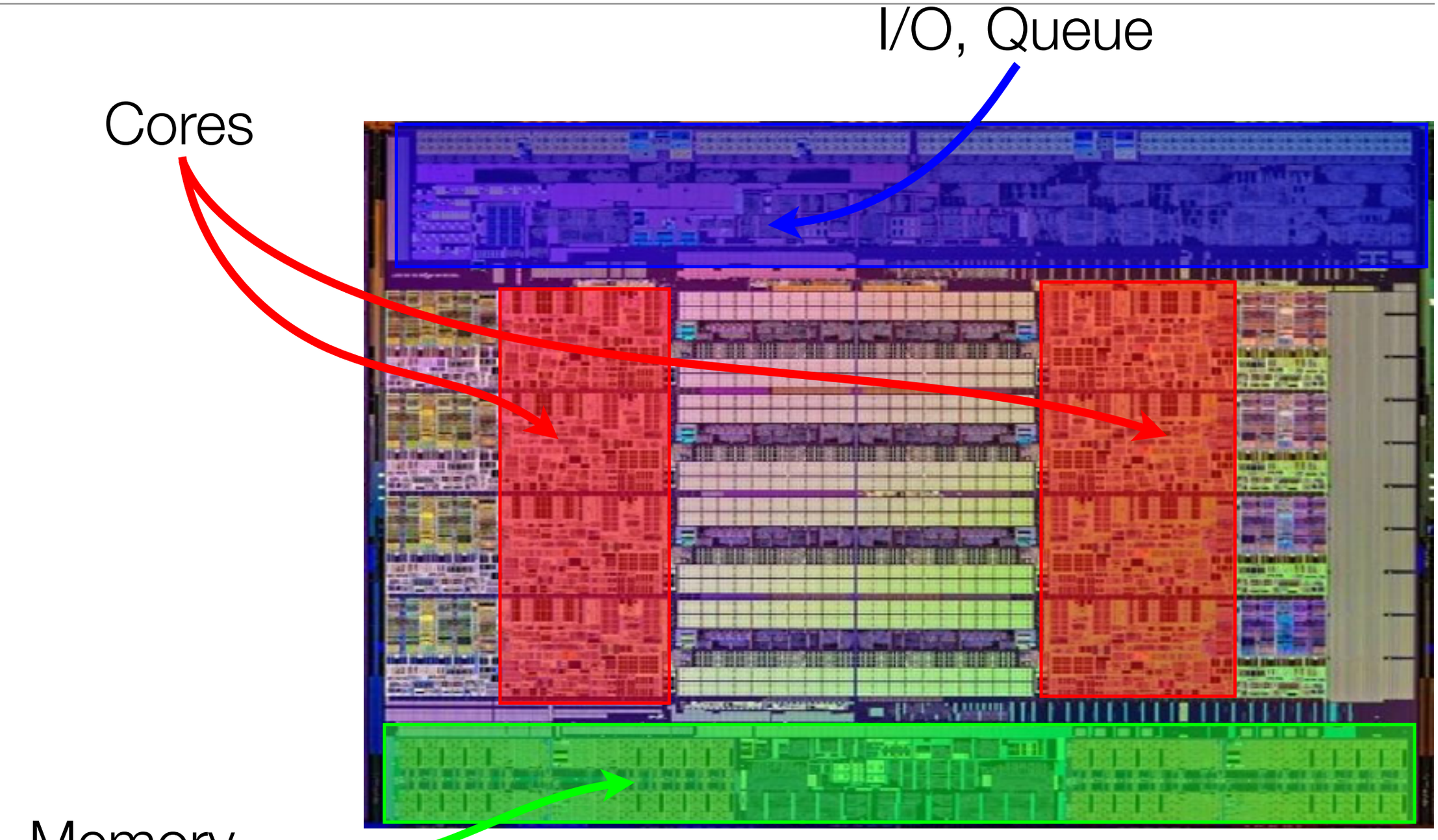
CPU Architecture (Intel i7)



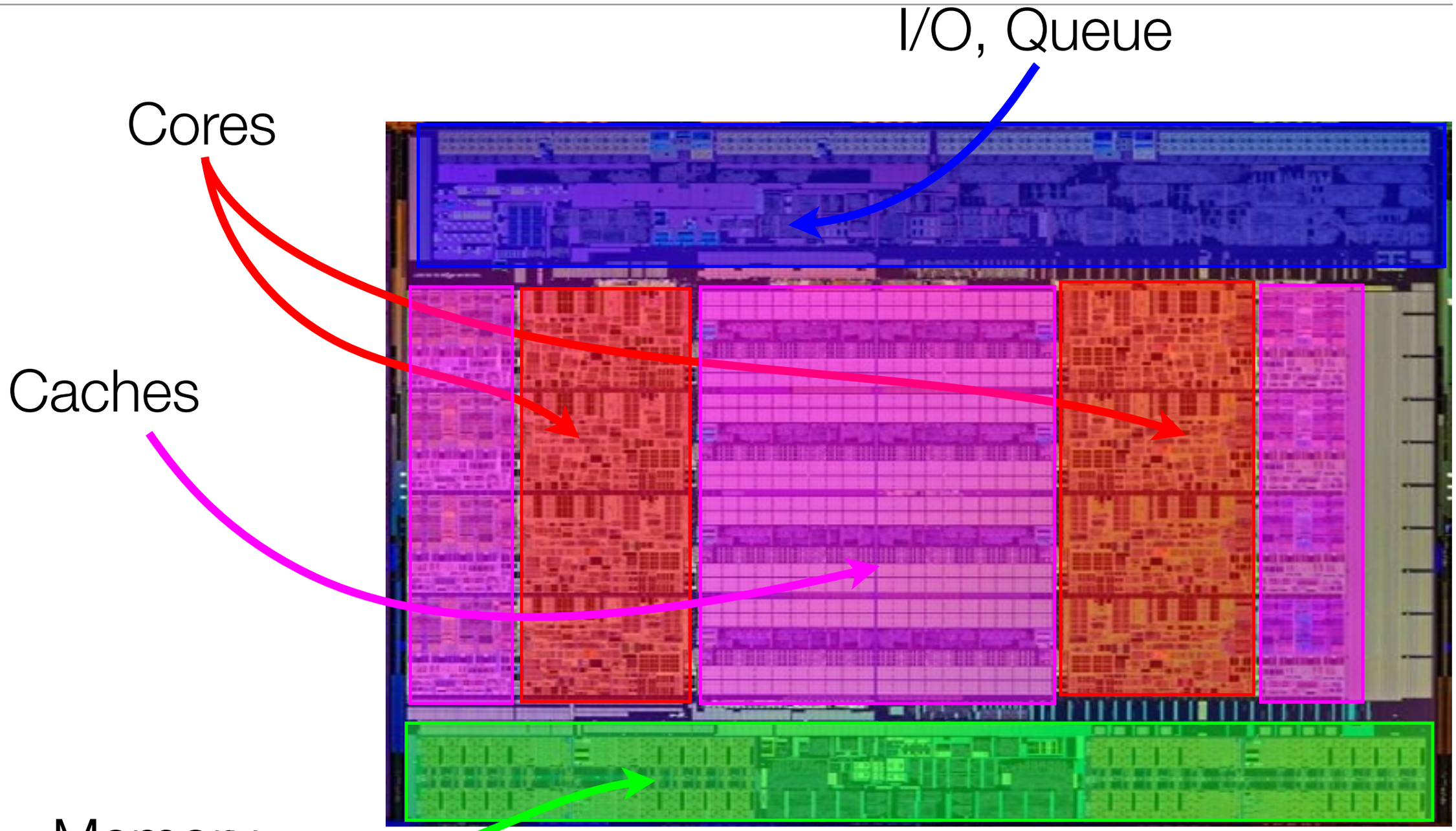
CPU Architecture (Intel i7)



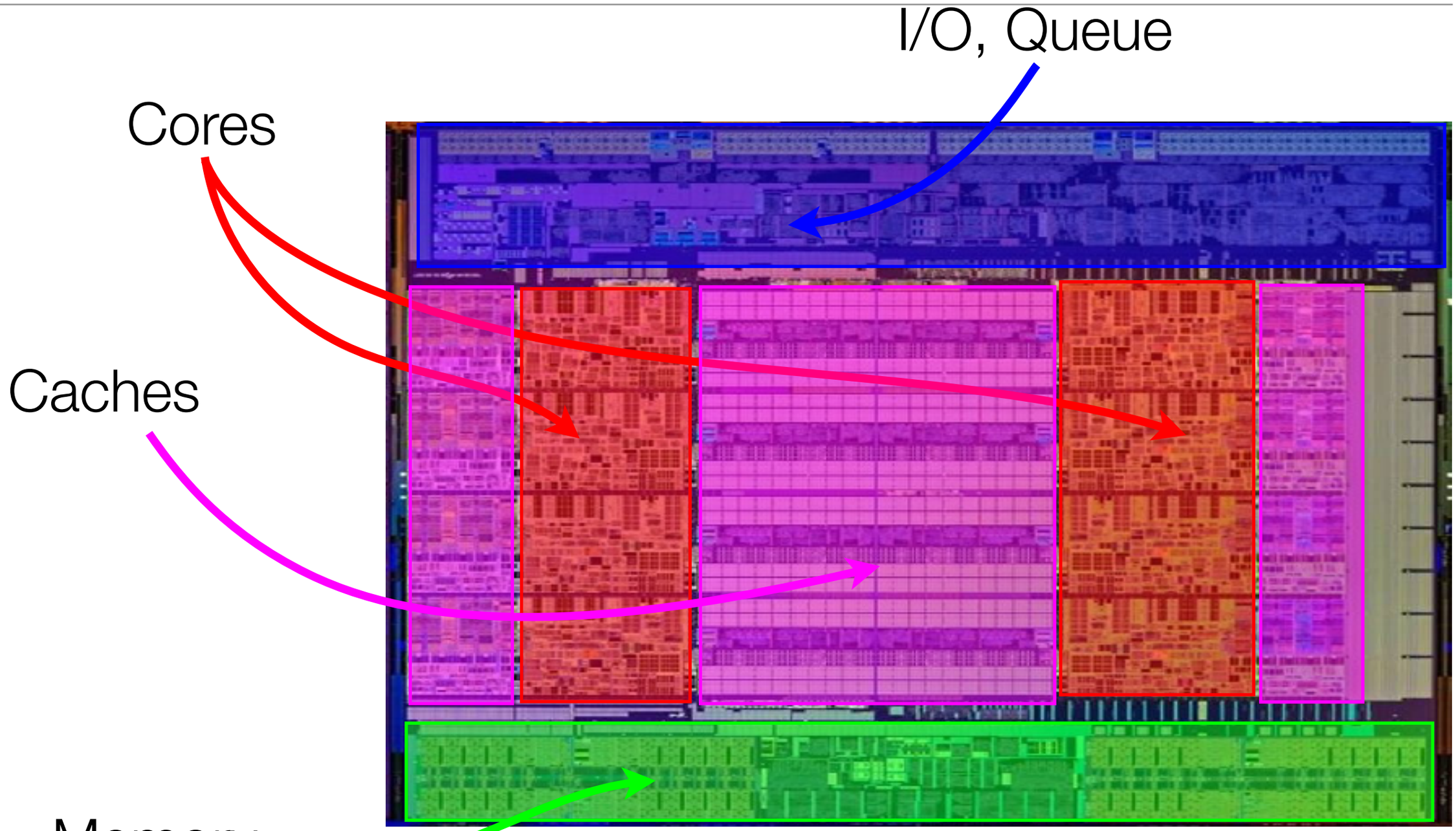
CPU Architecture (Intel i7)



CPU Architecture (Intel i7)



CPU Architecture (Intel i7)



“Multicore” architecture

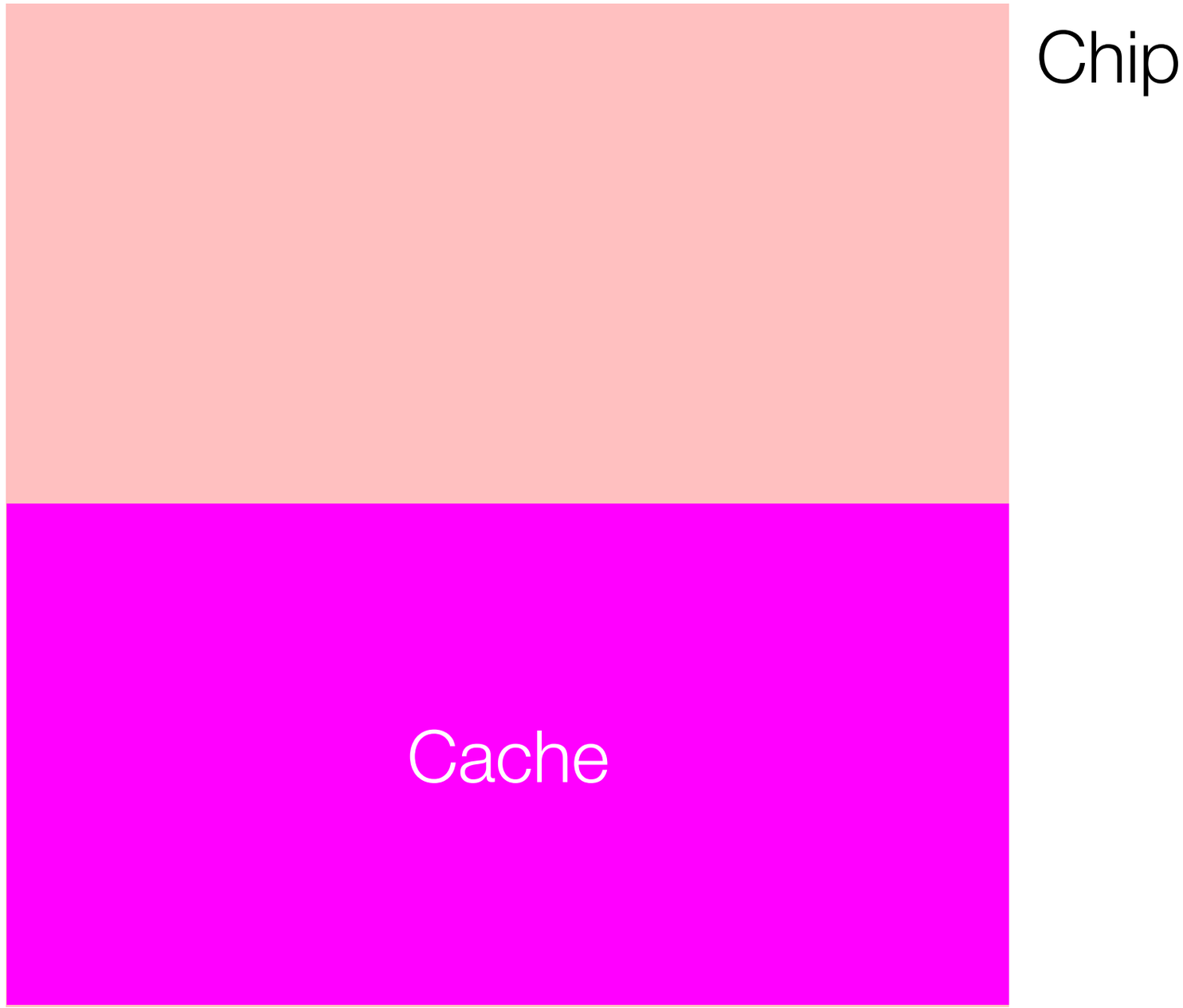
What if we got rid of everything?

CPU Architecture

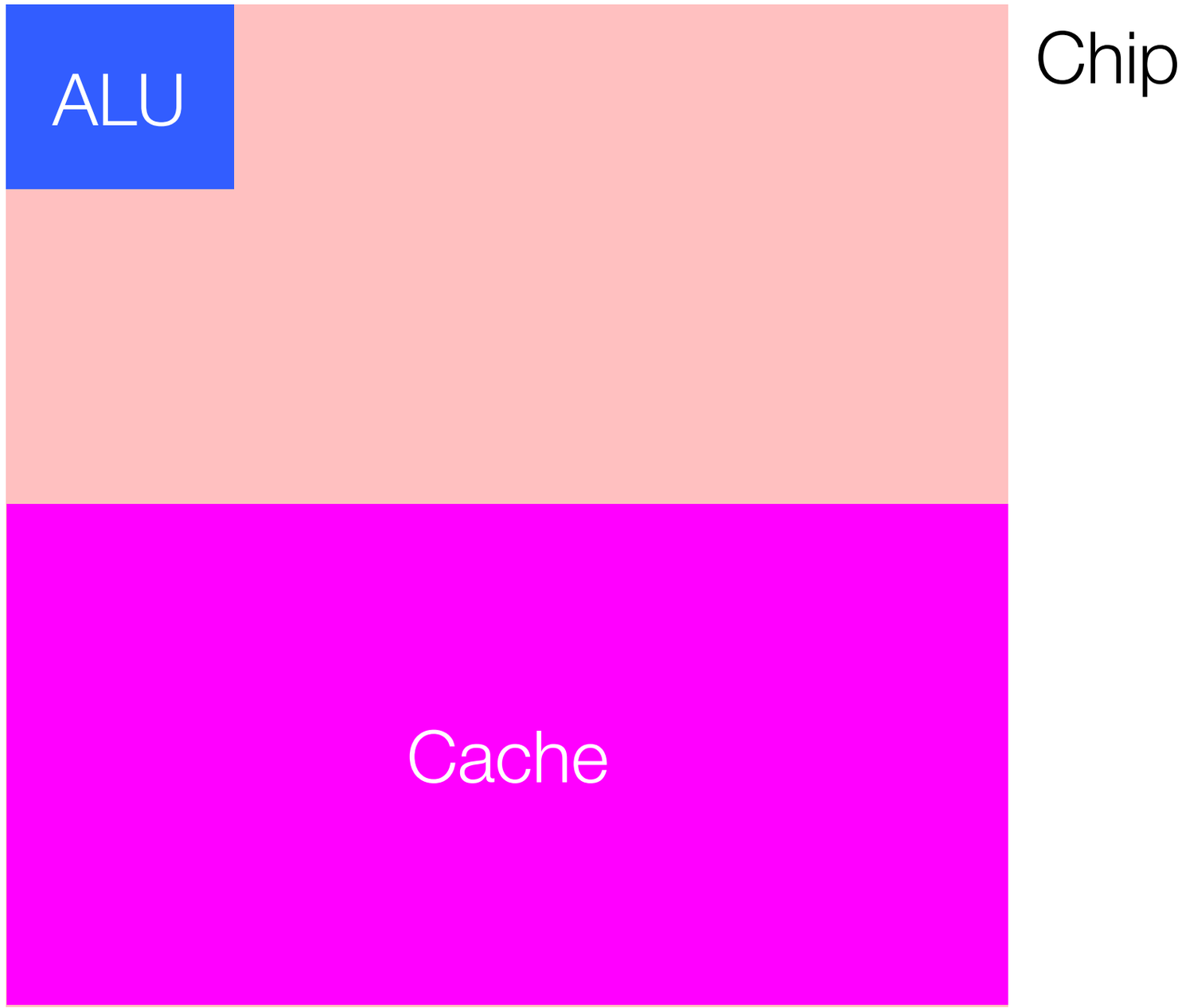
CPU Architecture

Chip

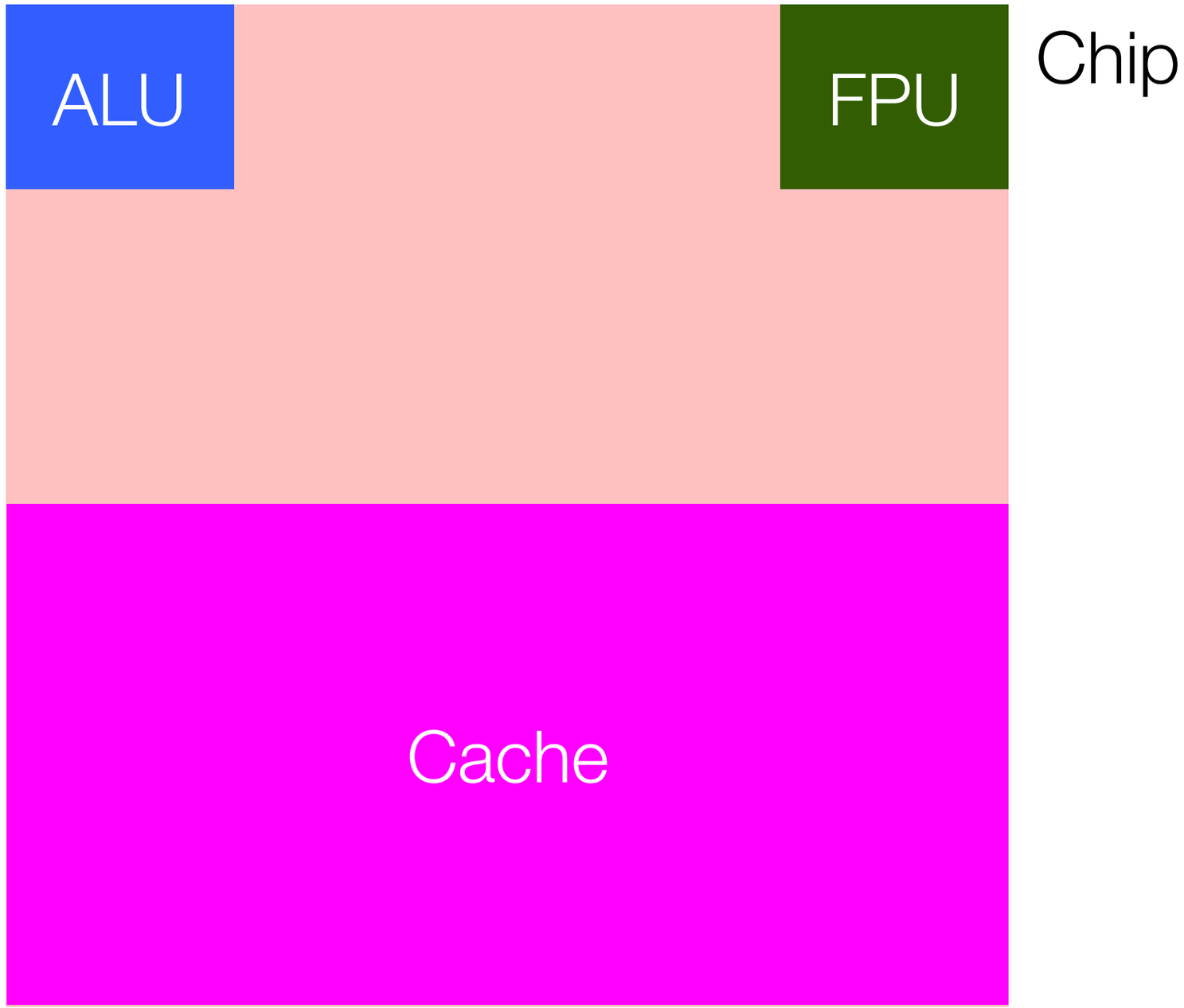
CPU Architecture



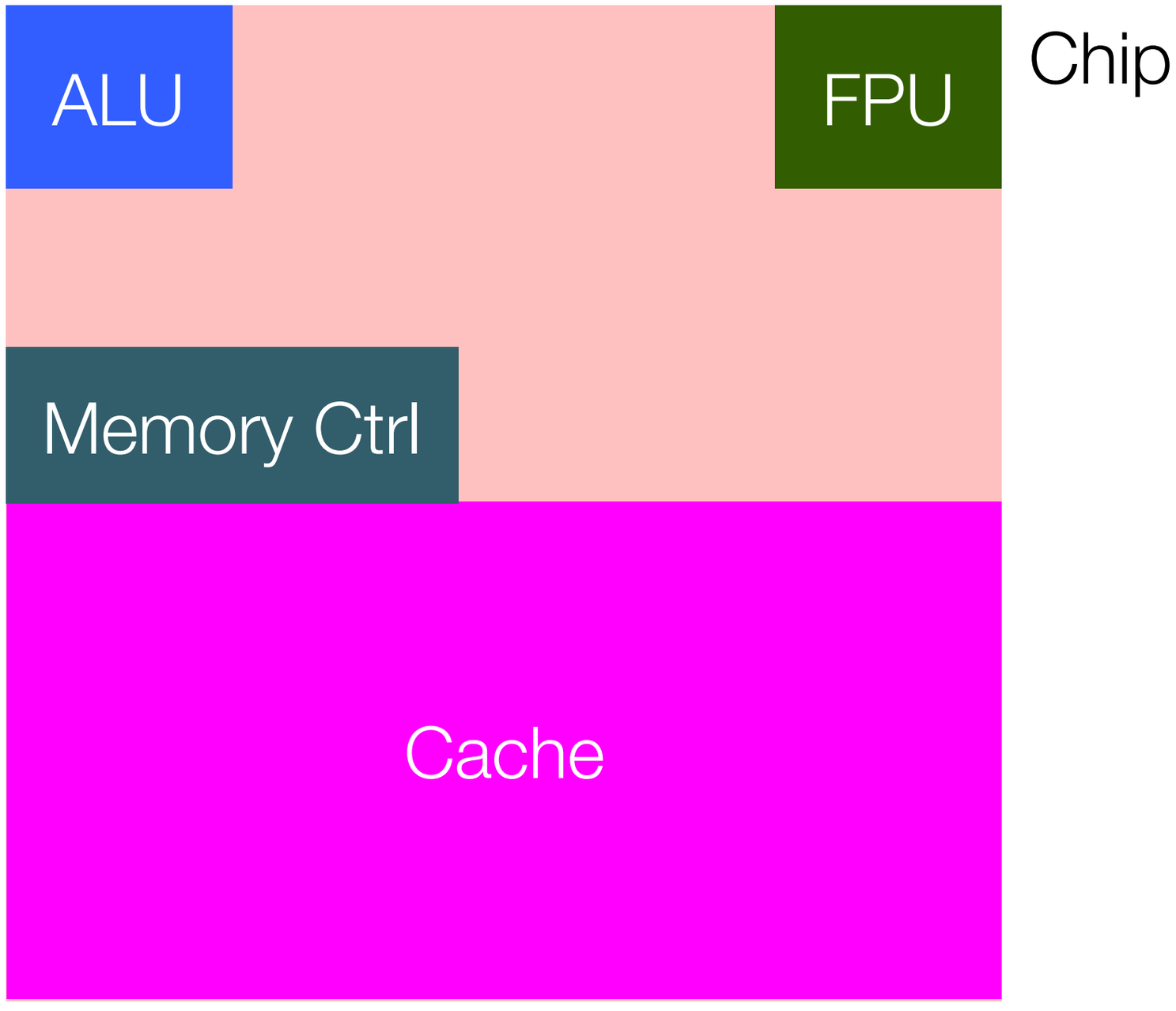
CPU Architecture



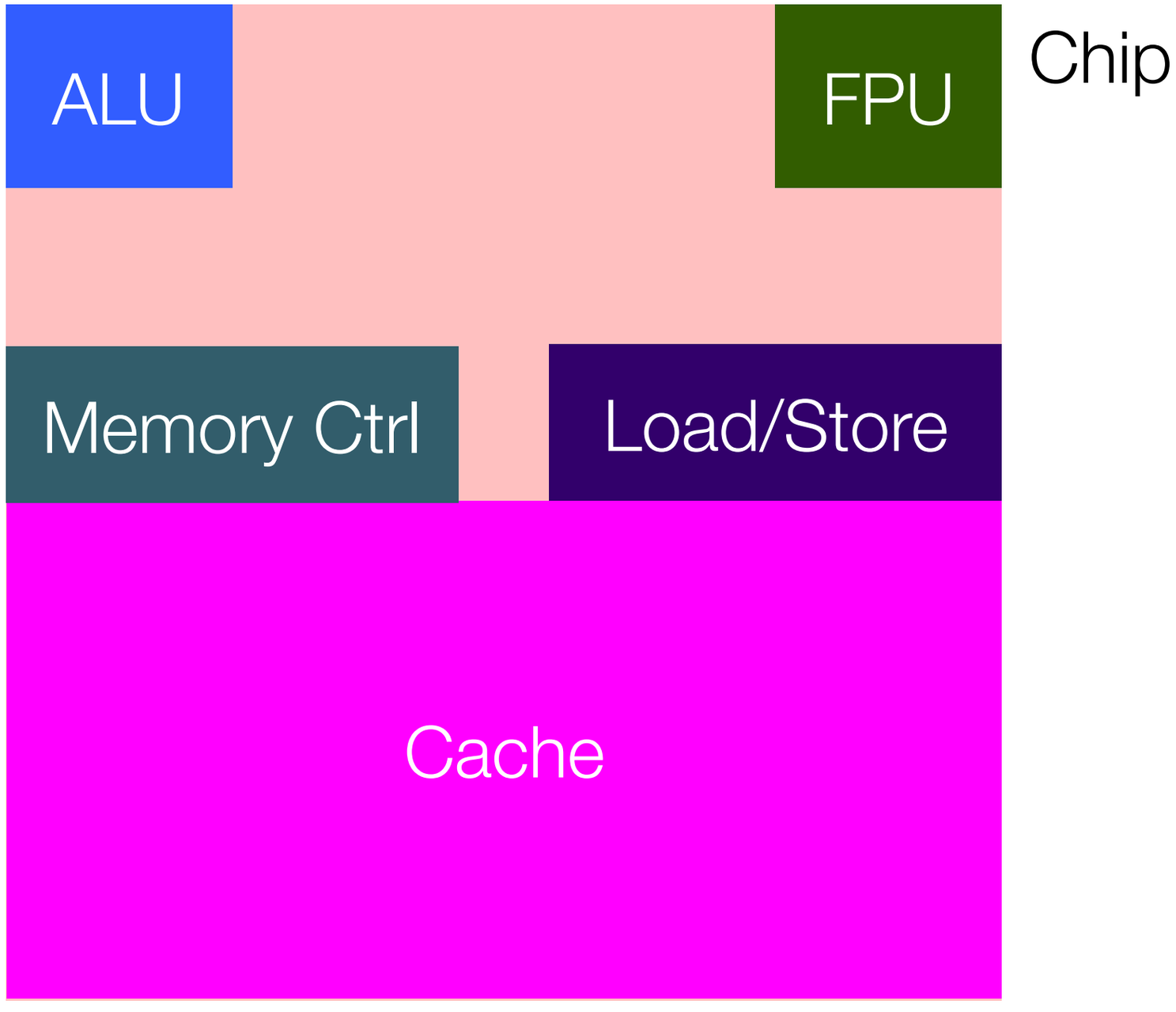
CPU Architecture



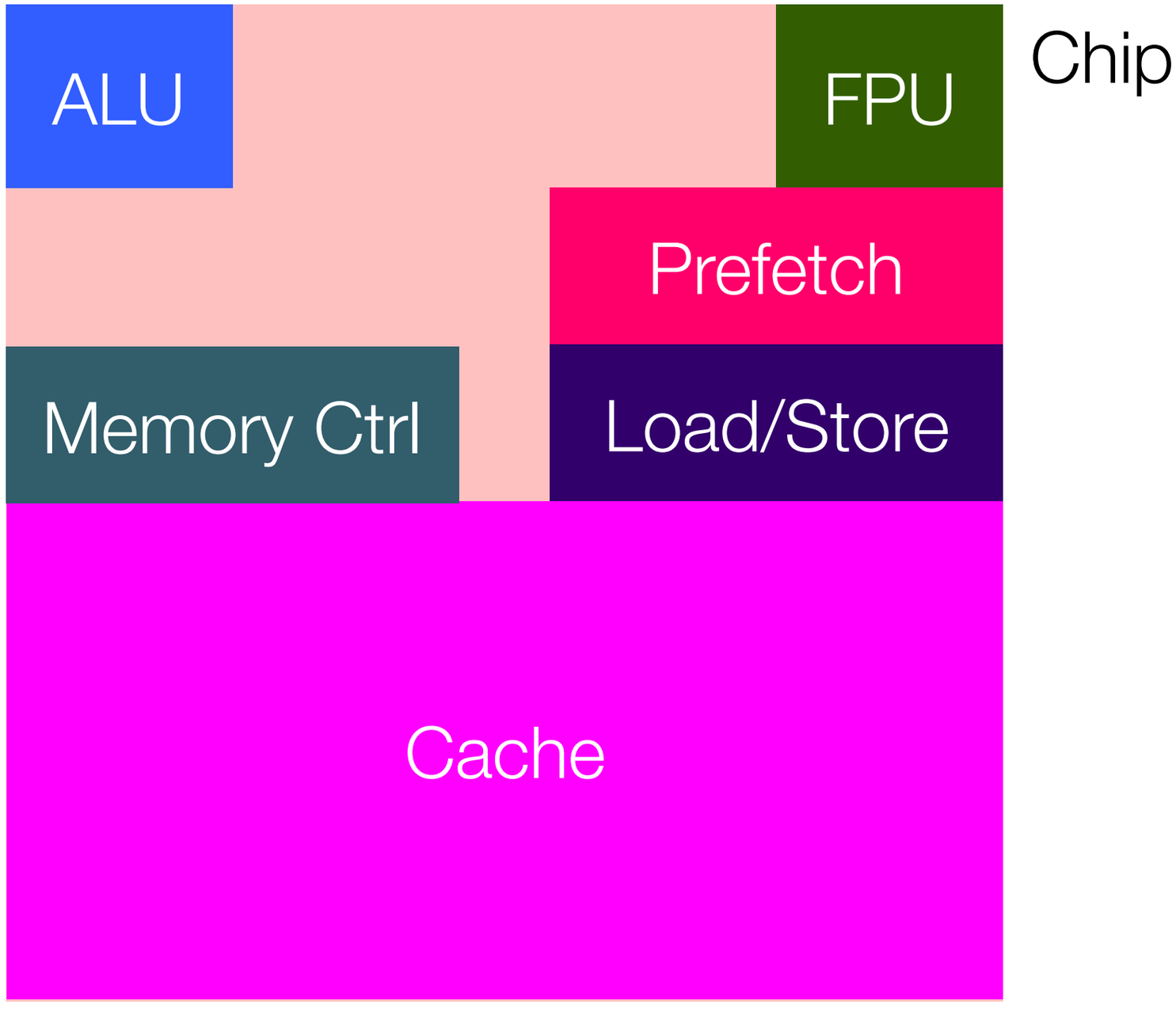
CPU Architecture



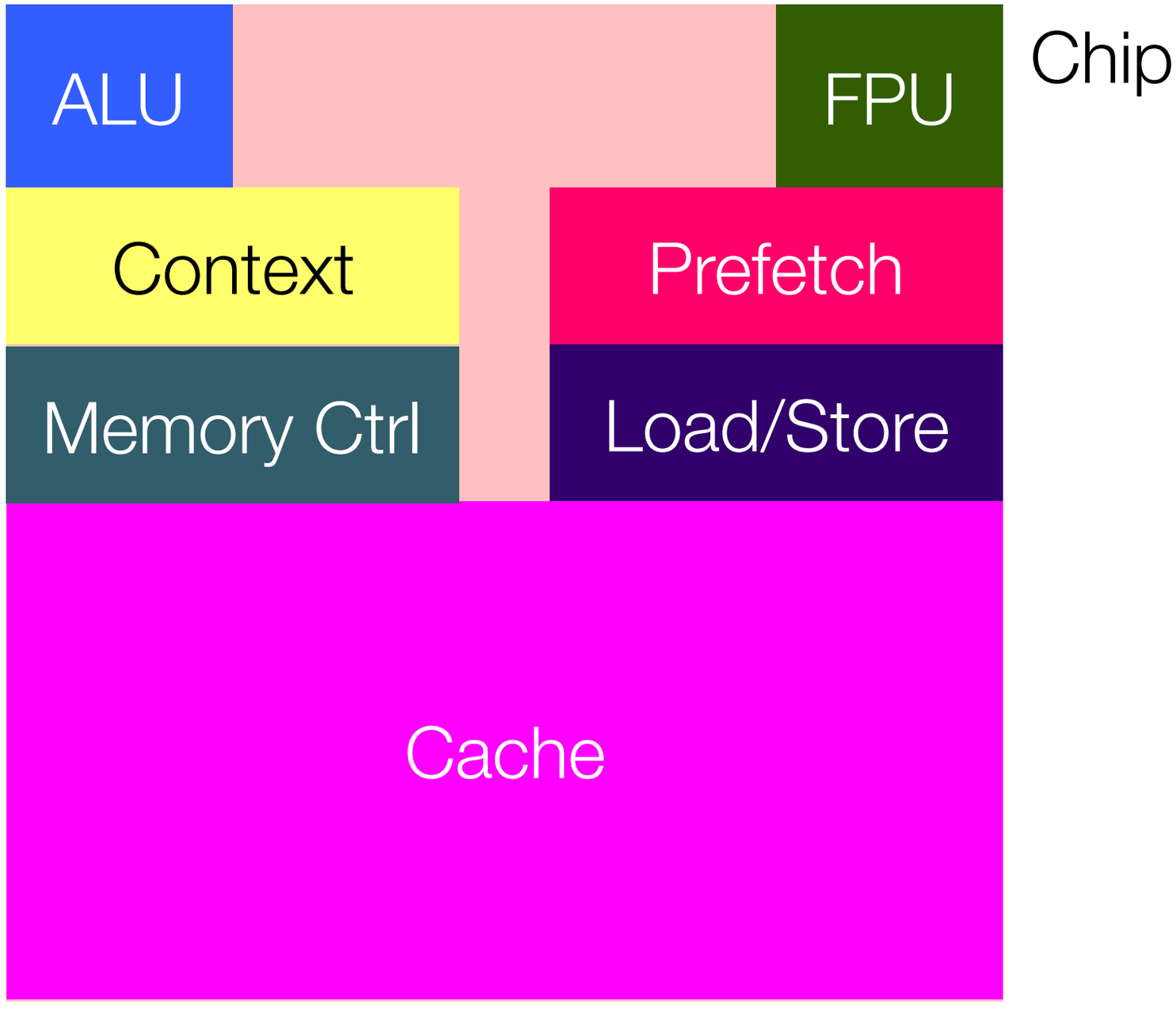
CPU Architecture



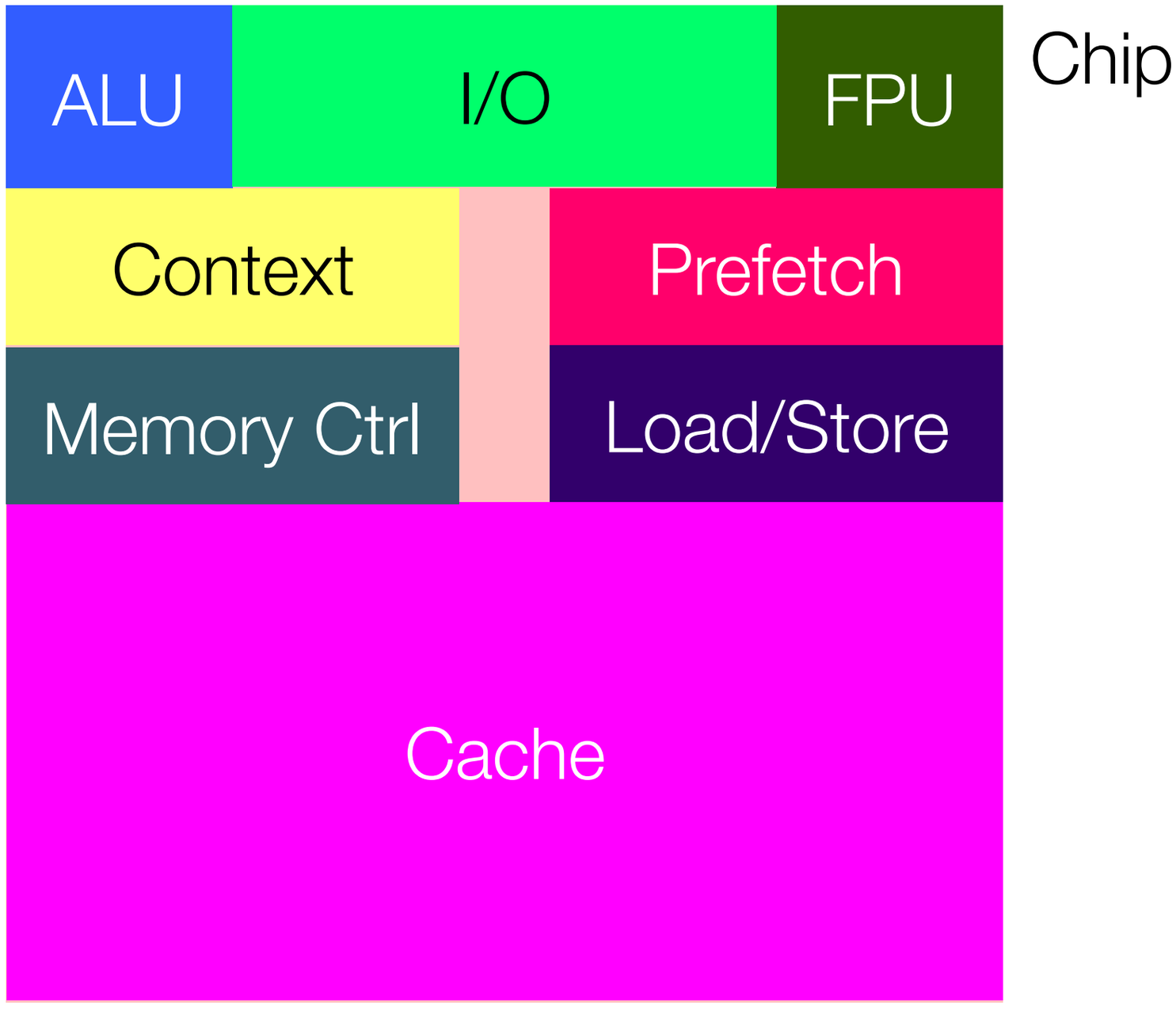
CPU Architecture



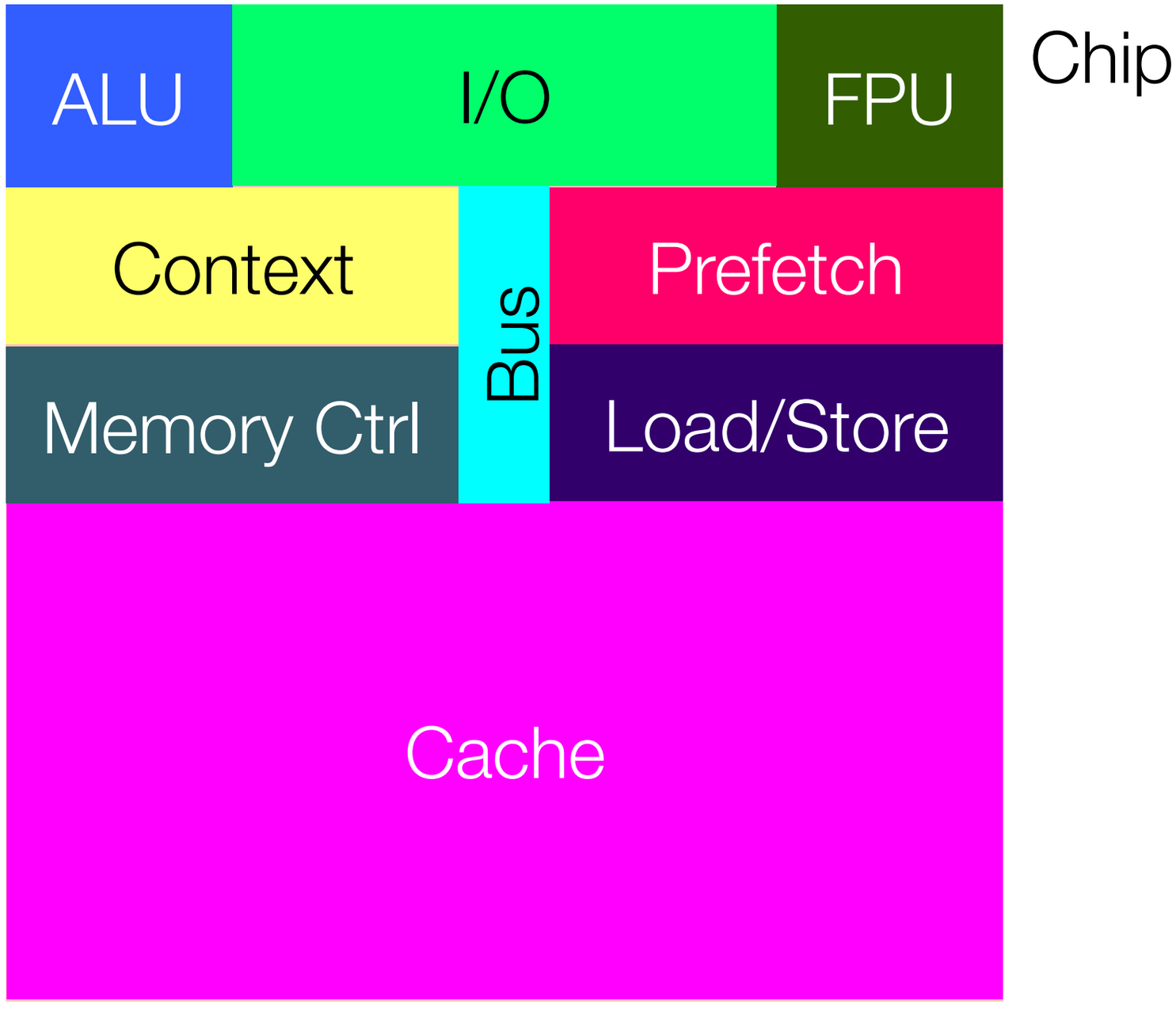
CPU Architecture



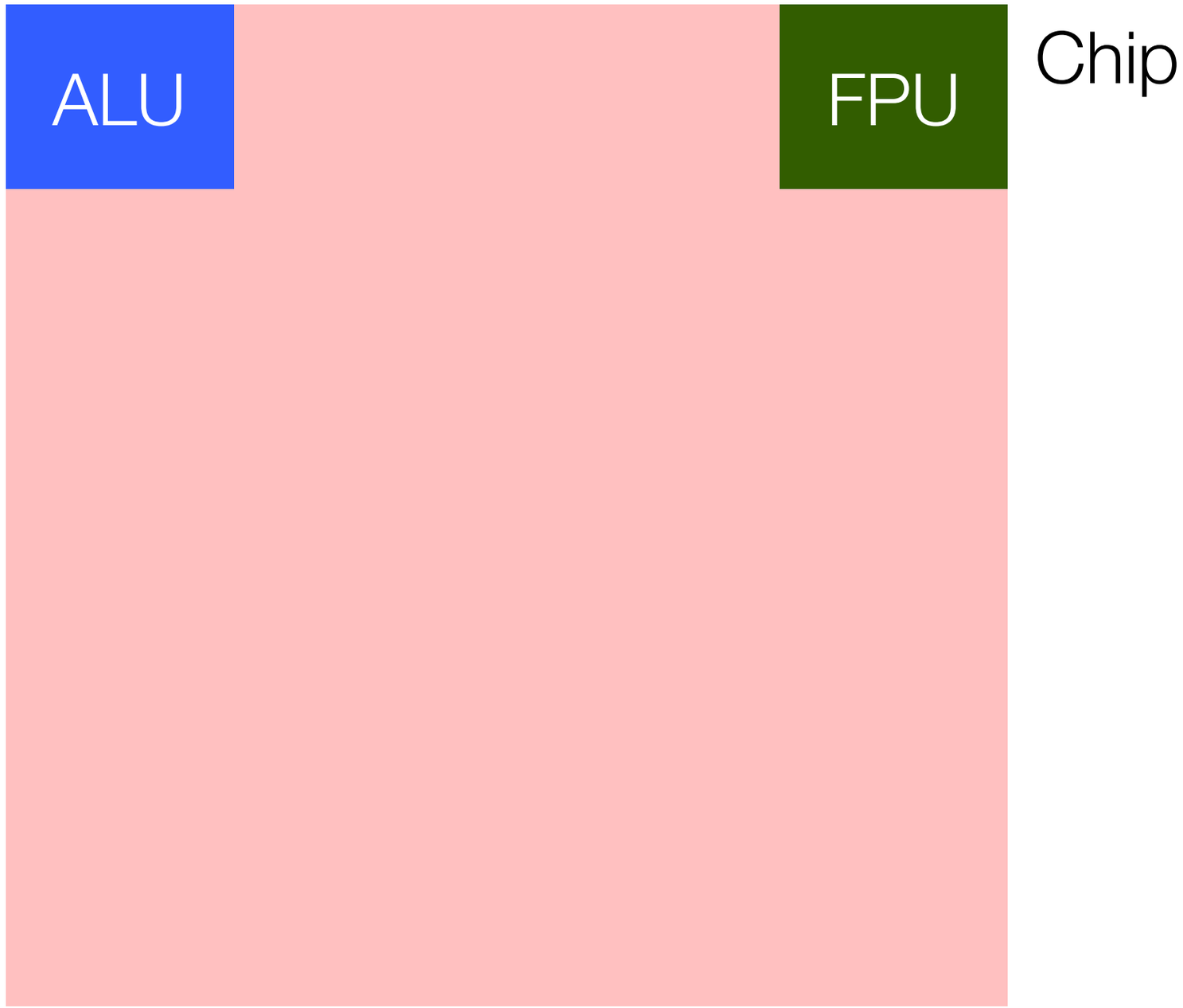
CPU Architecture



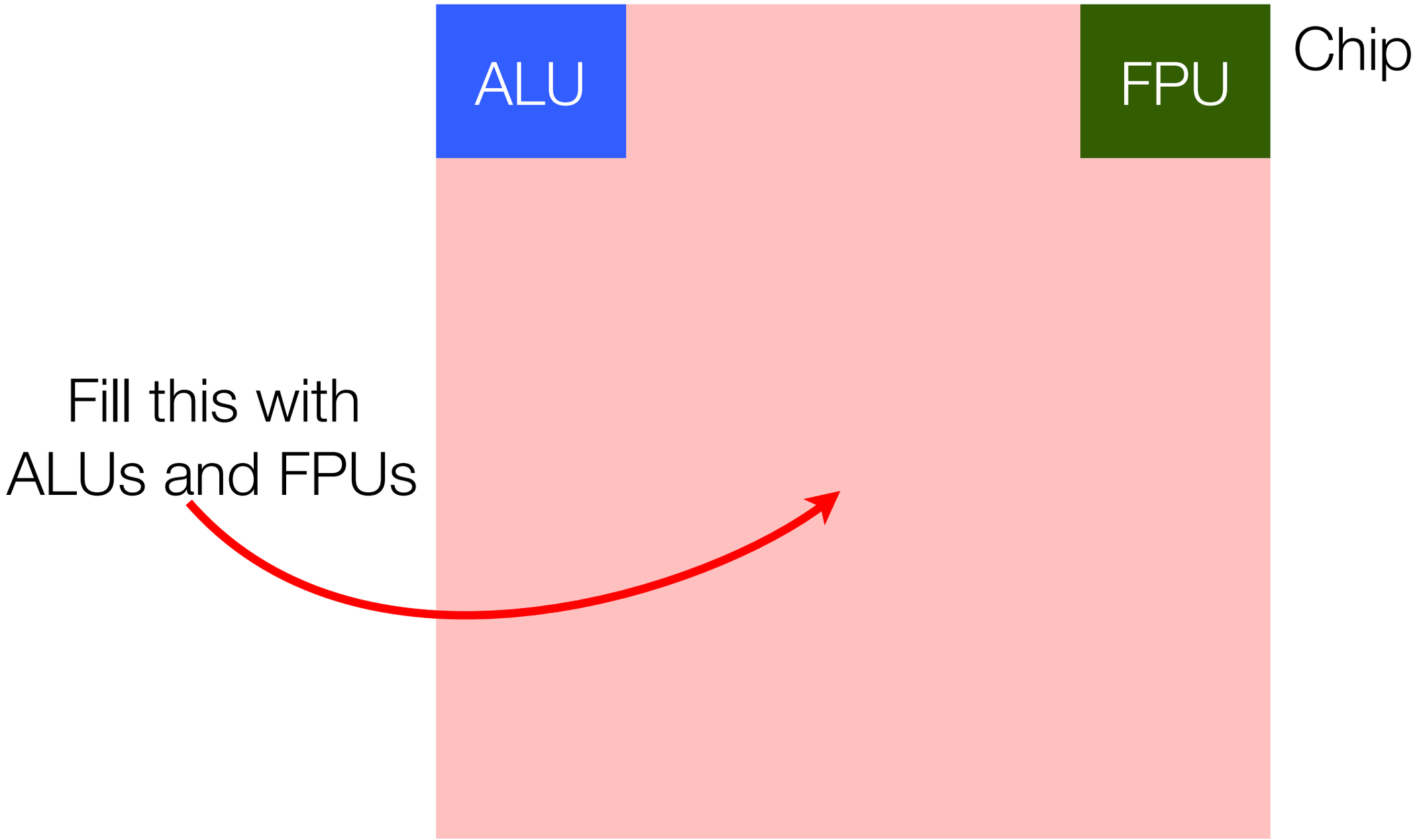
CPU Architecture



CPU Architecture



CPU Architecture



GPU Architecture

GPU Architecture

- Compared to CPUs, GPUs removed all components that sped up the execution of *single instructions*

GPU Architecture

- Compared to CPUs, GPUs removed all components that sped up the execution of *single instructions*

- No prefetching, out of order execution, branch prediction and so on, nothing of that: just a *minimal context*

GPU Architecture

- Compared to CPUs, GPUs removed all components that sped up the execution of *single instructions*

- No prefetching, out of order execution, branch prediction and so on, nothing of that: just a *minimal context*

- All of the remaining chip area can be now filled with ALUs and can therefore run *multiple threads*

GPU Architecture

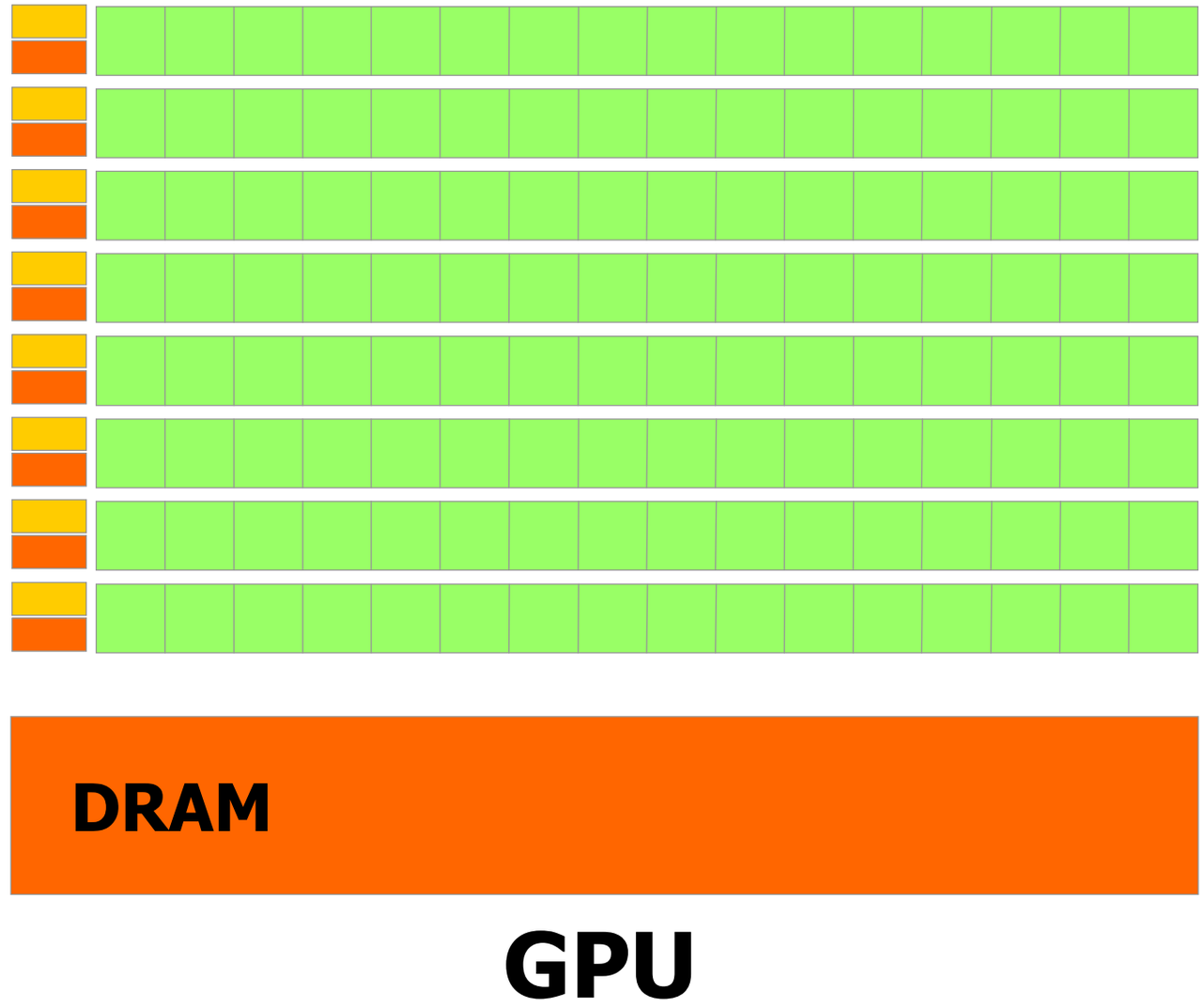
- Compared to CPUs, GPUs removed all components that sped up the execution of *single instructions*

- No prefetching, out of order execution, branch prediction and so on, nothing of that: just a *minimal context*

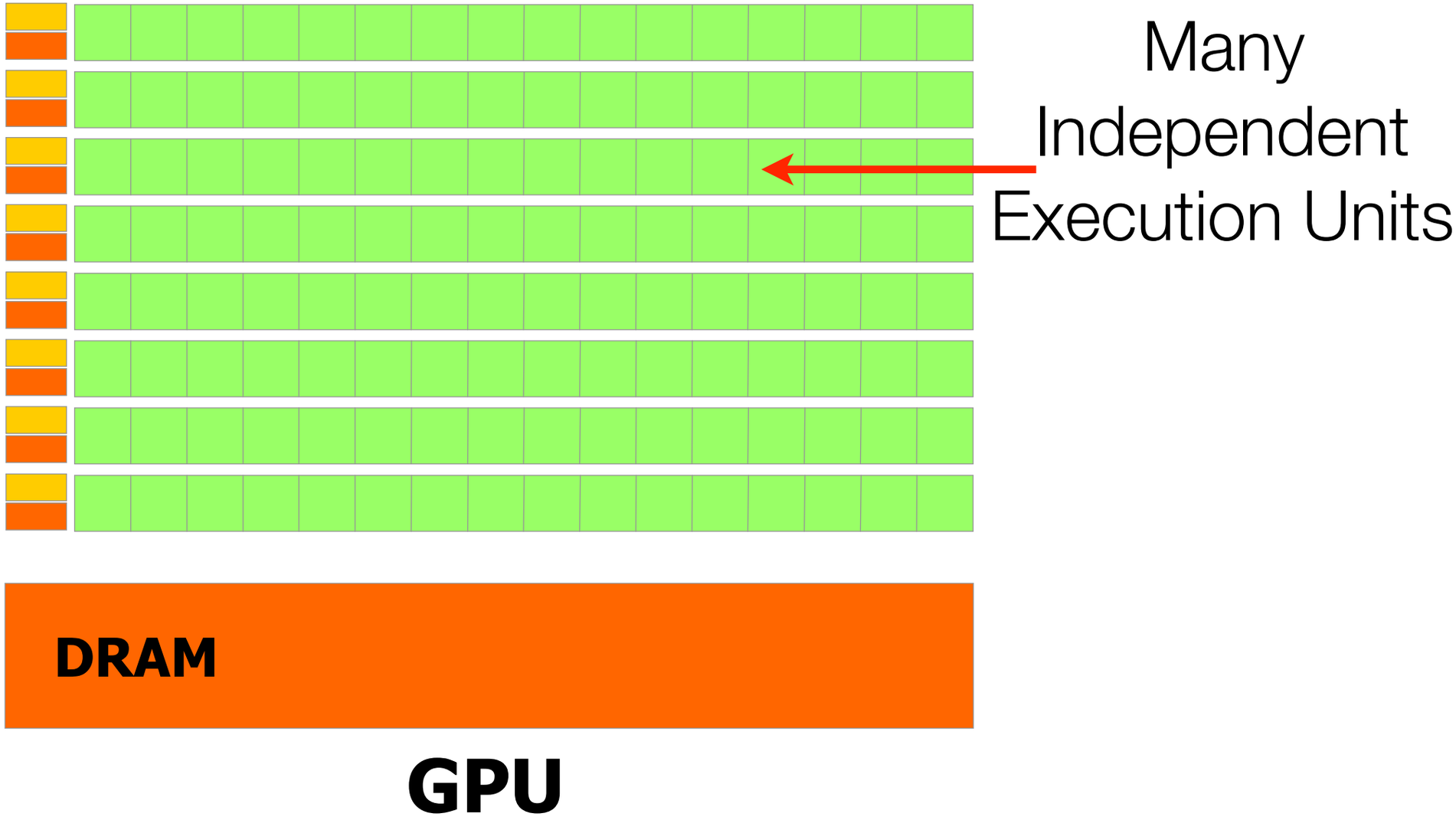
- All of the remaining chip area can be now filled with ALUs and can therefore run *multiple threads*

- Since GPUs actually *share control* among cores, one single instruction is executed by several threads at once by adding execution units instead of control ones

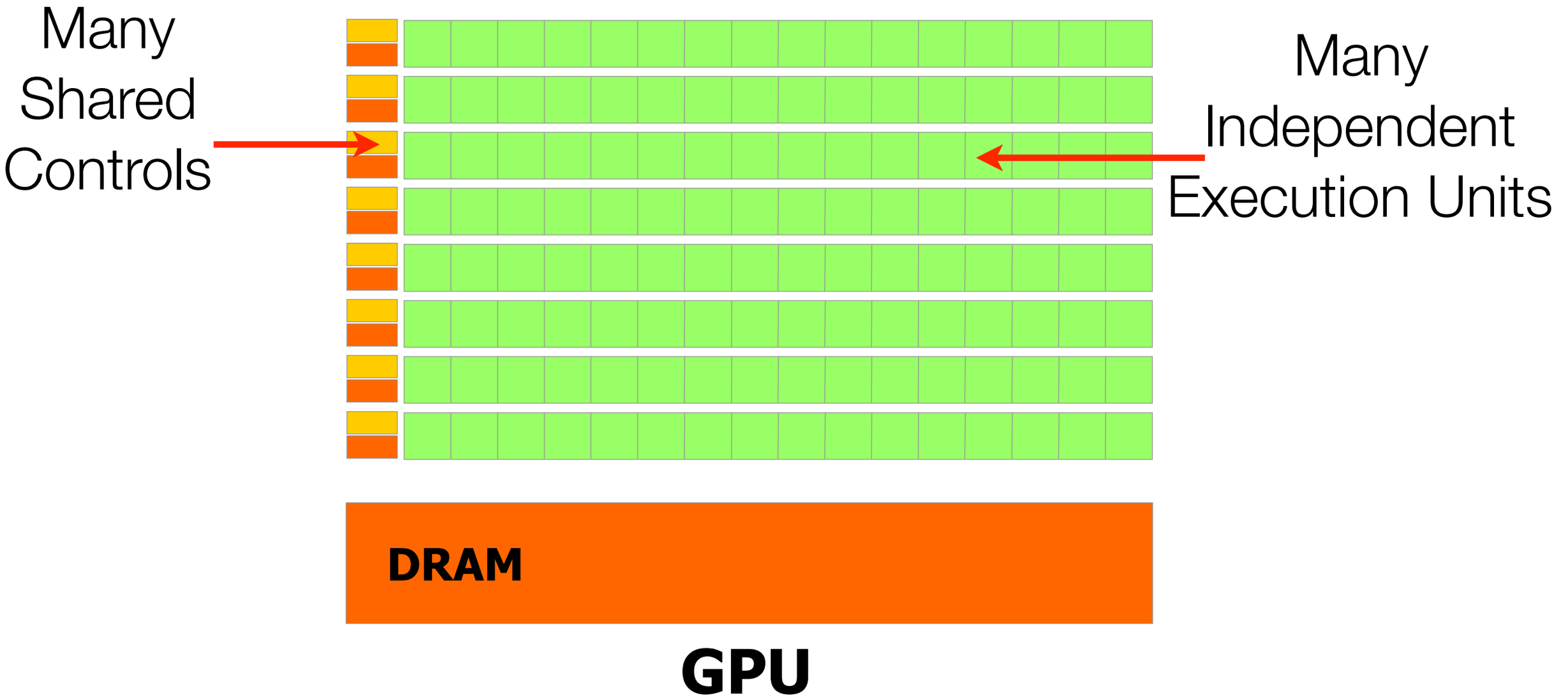
SIMD Architecture



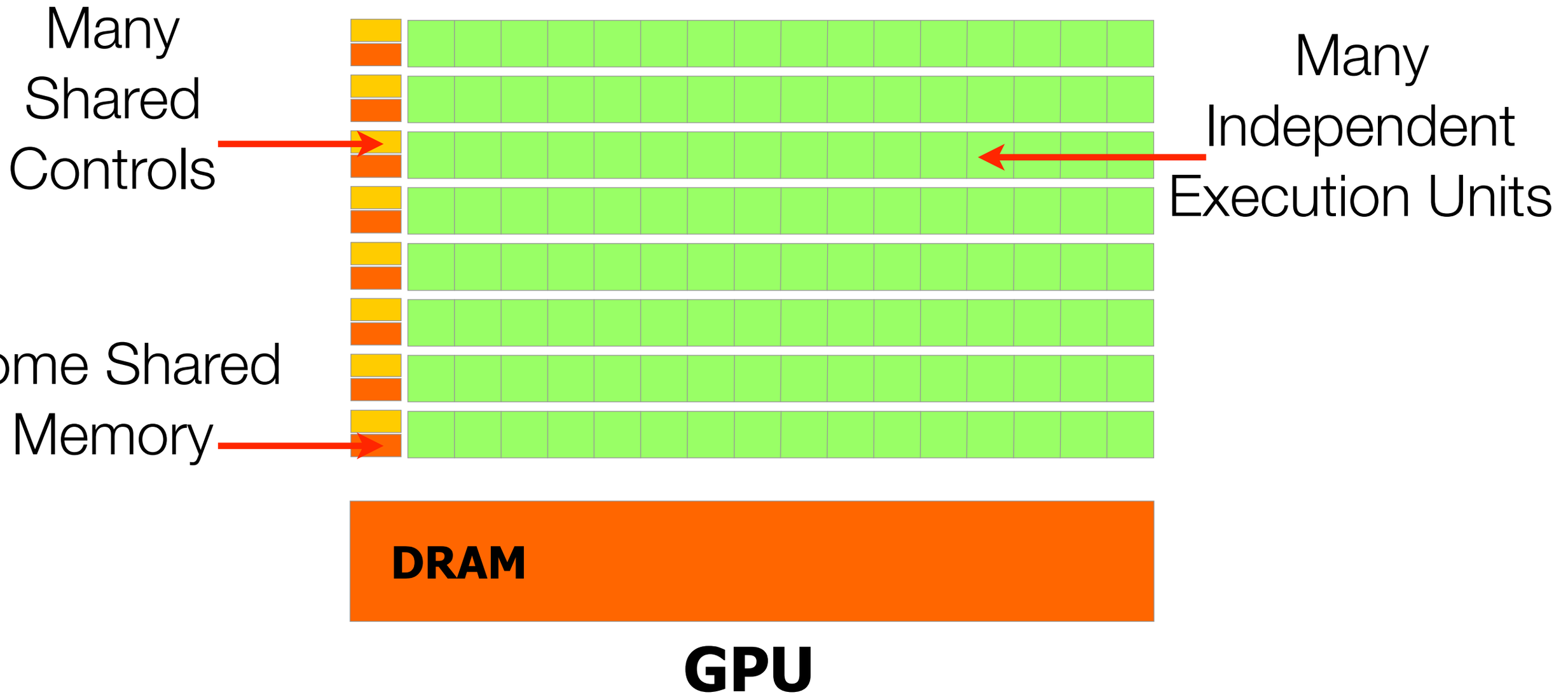
SIMD Architecture



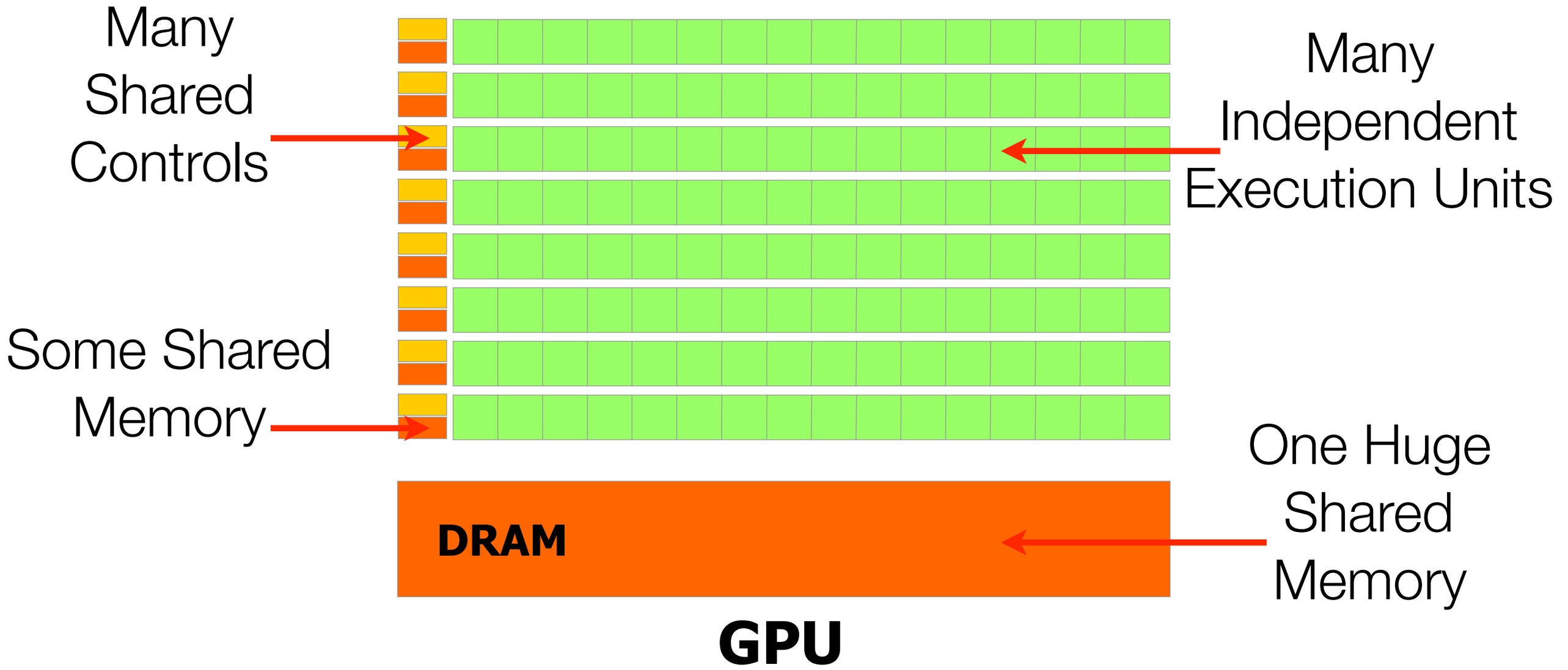
SIMD Architecture



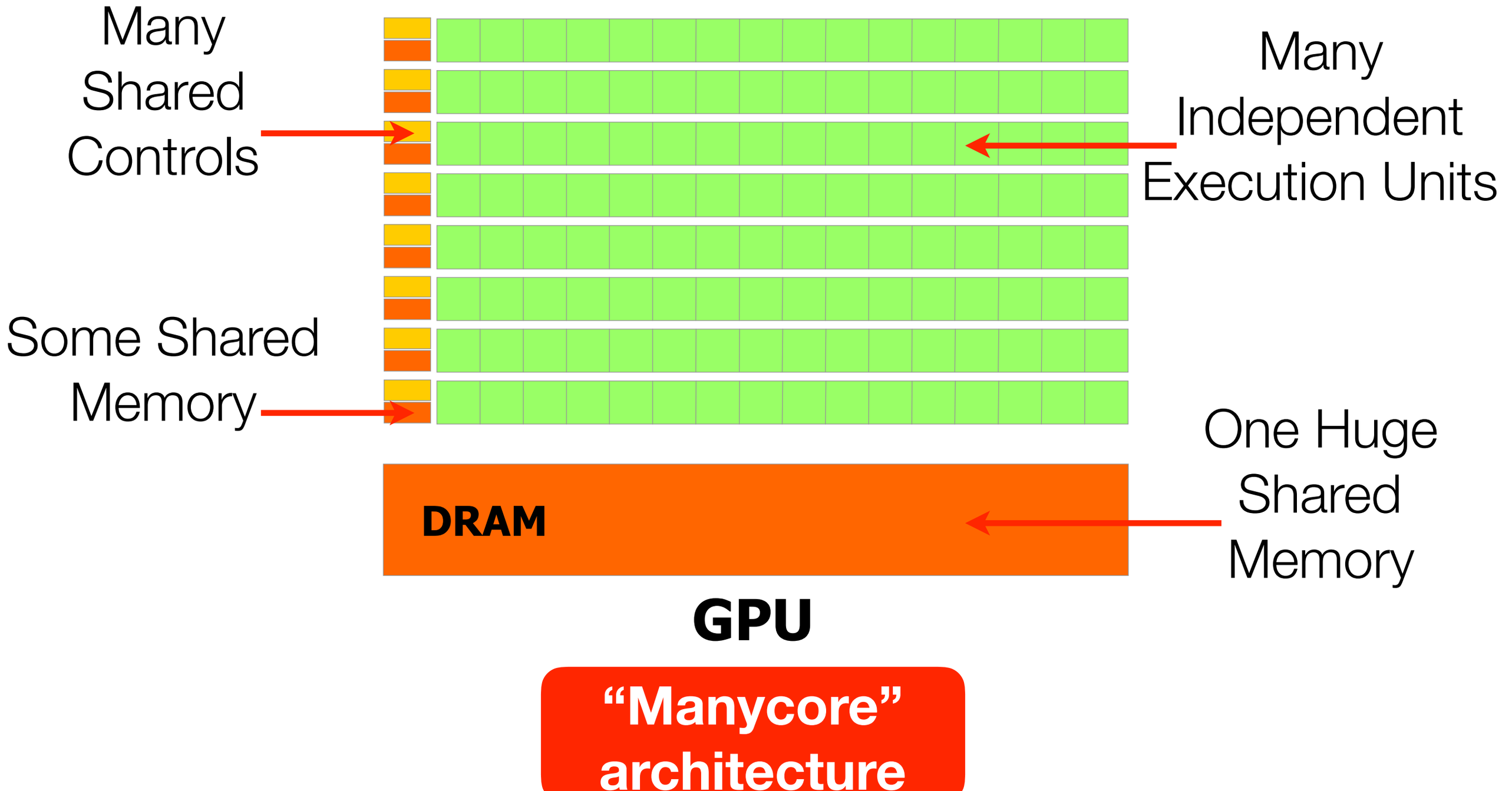
SIMD Architecture



SIMD Architecture



SIMD Architecture



“Manycore” architecture

Try It Yourself

Use Amazon EC2 or similar for free

Historical Notes

Historical Notes

- When shaders were available, people from academia started using GPUs for something different from graphics

Historical Notes

- When shaders were available, people from academia started using GPUs for something different from graphics

- Matrix multiplication was easy and developed in 2001, while LU decomposition was written in 2005

Historical Notes

- When shaders were available, people from academia started using GPUs for something different from graphics

- Matrix multiplication was easy and developed in 2001, while LU decomposition was written in 2005

- Back then one had to use OpenGL and DirectX

Historical Notes

- When shaders were available, people from academia started using GPUs for something different from graphics

- Matrix multiplication was easy and developed in 2001, while LU decomposition was written in 2005

- Back then one had to use OpenGL and DirectX

- Some programming languages emerged, like *Brook* or *Sh*, but NVidia changed the development perspective

Historical Notes

- When shaders were available, people from academia started using GPUs for something different from graphics

- Matrix multiplication was easy and developed in 2001, while LU decomposition was written in 2005

- Back then one had to use OpenGL and DirectX

- Some programming languages emerged, like *Brook* or *Sh*, but NVidia changed the development perspective

- In 2007 they released CUDA

CUDA

CUDA

- CUDA is acronym for “Compute Unified Device Architecture”

CUDA

- CUDA is acronym for “Compute Unified Device Architecture”
- It is a general purpose parallel computing architecture

CUDA

- CUDA is acronym for “Compute Unified Device Architecture”
- It is a general purpose parallel computing architecture
- It comprises hardware, compilers (C, C++, and FORTRAN), and several libraries

CUDA

- CUDA is acronym for “Compute Unified Device Architecture”
- It is a general purpose parallel computing architecture
- It comprises hardware, compilers (C, C++, and FORTRAN), and several libraries
- It may use several GPUs, if present on the device

CUDA

- CUDA is acronym for “Compute Unified Device Architecture”
- It is a general purpose parallel computing architecture
- It comprises hardware, compilers (C, C++, and FORTRAN), and several libraries
- It may use several GPUs, if present on the device
- Obviously, it runs only on NVidia’s hardware (but compiles on every computer)

Main Characteristics

Main Characteristics

- Nvidia suggests to use its extension to common ISO C

Main Characteristics

- Nvidia suggests to use its extension to common ISO C
- *Threads* are a native concept

Main Characteristics

- Nvidia suggests to use its extension to common ISO C
- *Threads* are a native concept
- CUDA is a shared memory architecture, and it supports *barriers* and *synchronization* calls

Main Characteristics

- Nvidia suggests to use its extension to common ISO C
- *Threads* are a native concept
- CUDA is a shared memory architecture, and it supports *barriers* and *synchronization* calls
- A CUDA program is GPU “independent” (sort of)

Main Characteristics

- Nvidia suggests to use its extension to common ISO C
- *Threads* are a native concept
- CUDA is a shared memory architecture, and it supports *barriers* and *synchronization* calls
- A CUDA program is GPU “independent” (sort of)
- It runs the same program on different GPUs

Vector Sum

Vector Sum

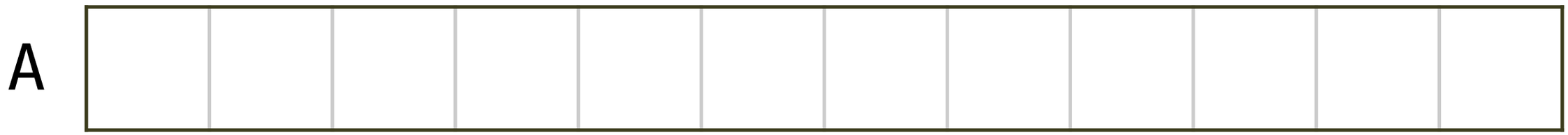
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

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

Vector Sum

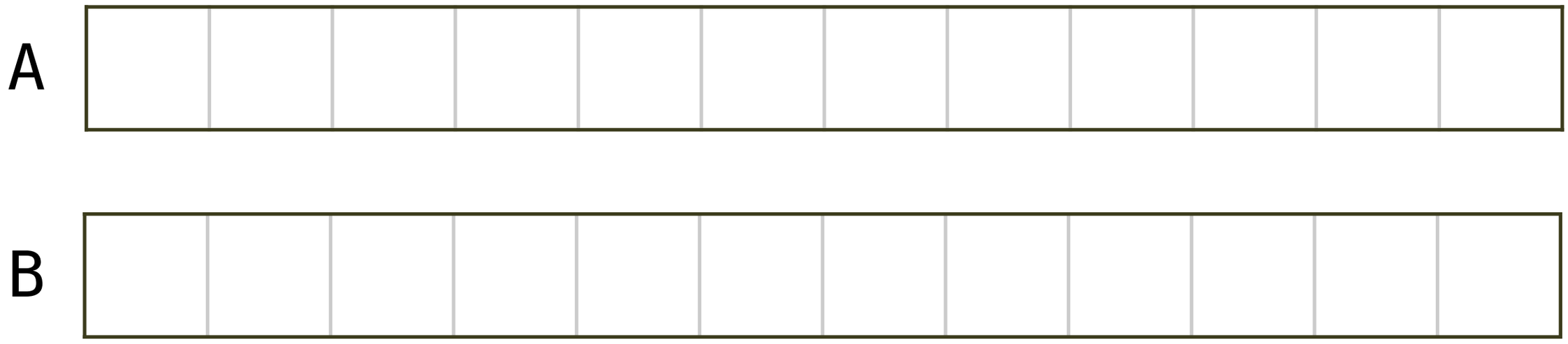
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

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

Vector Sum

Vector Sum

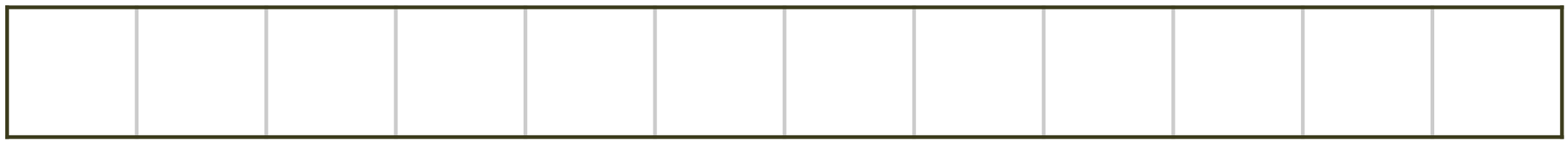


Vector Sum

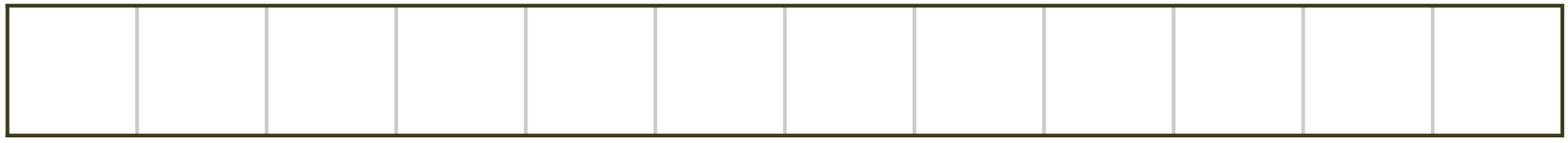


Vector Sum

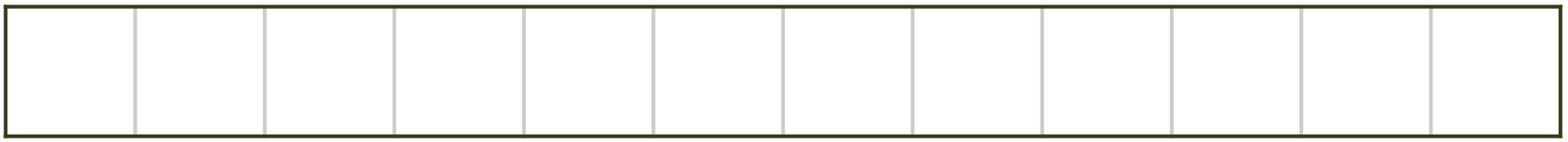
A



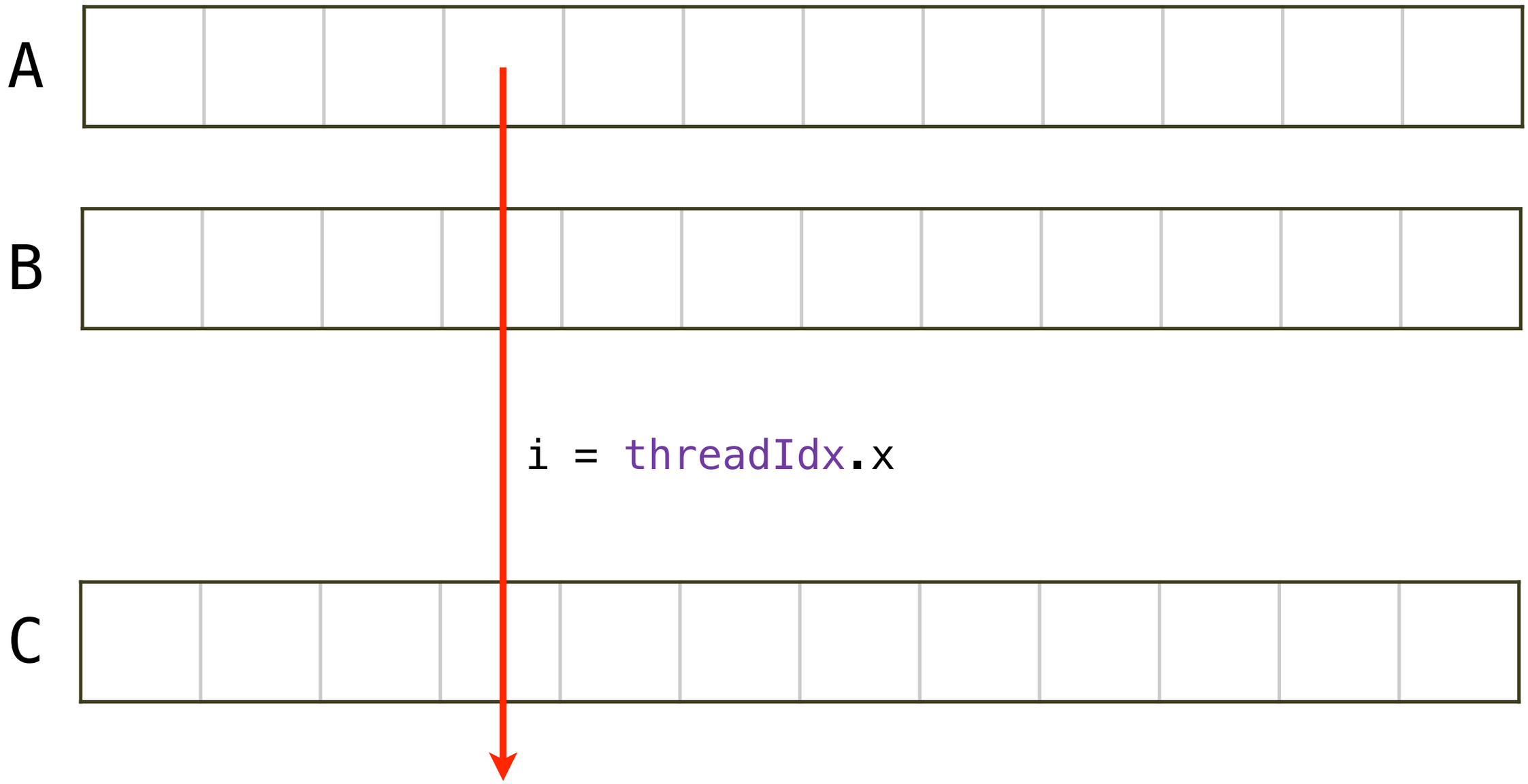
B



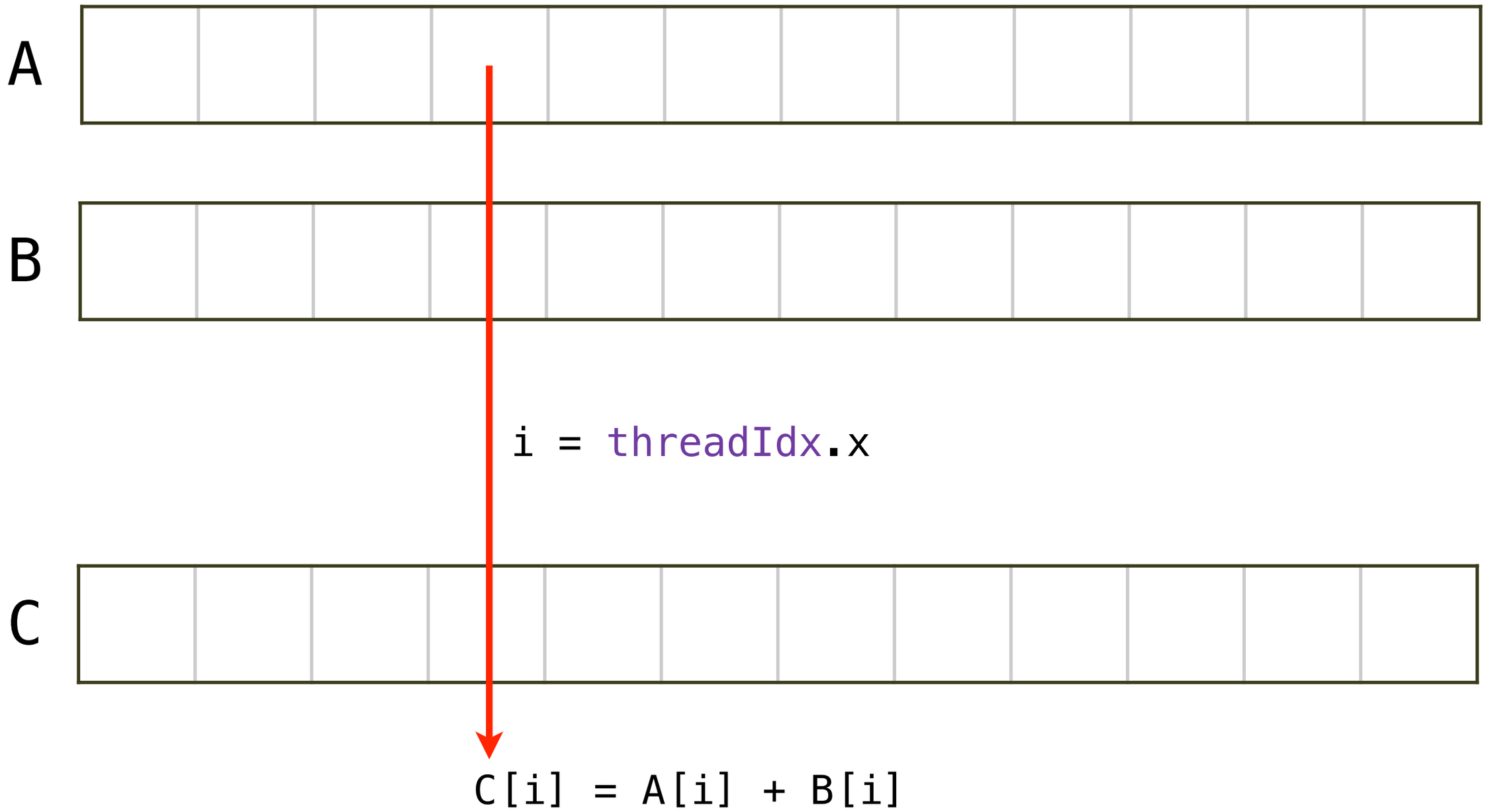
C



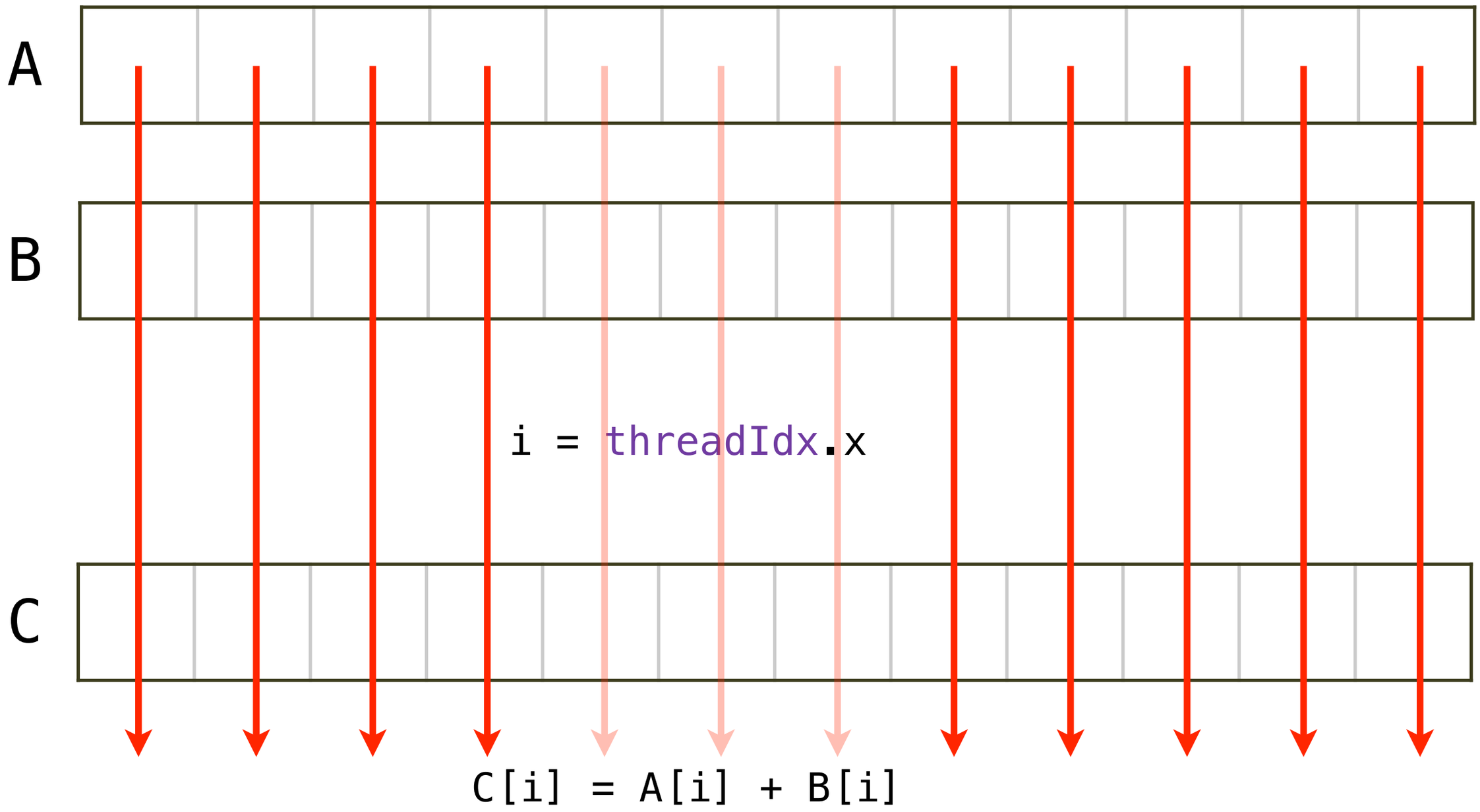
Vector Sum



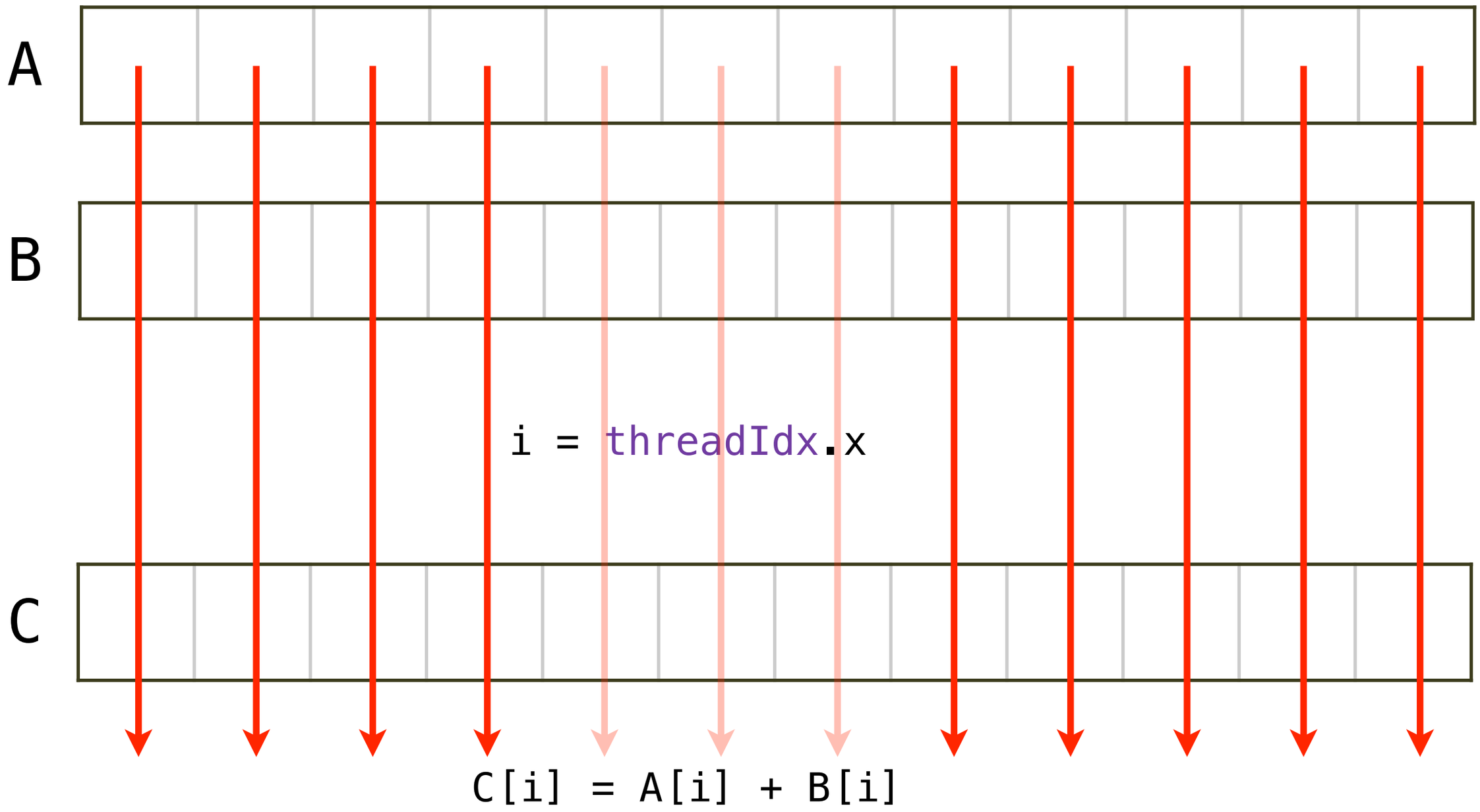
Vector Sum



Vector Sum

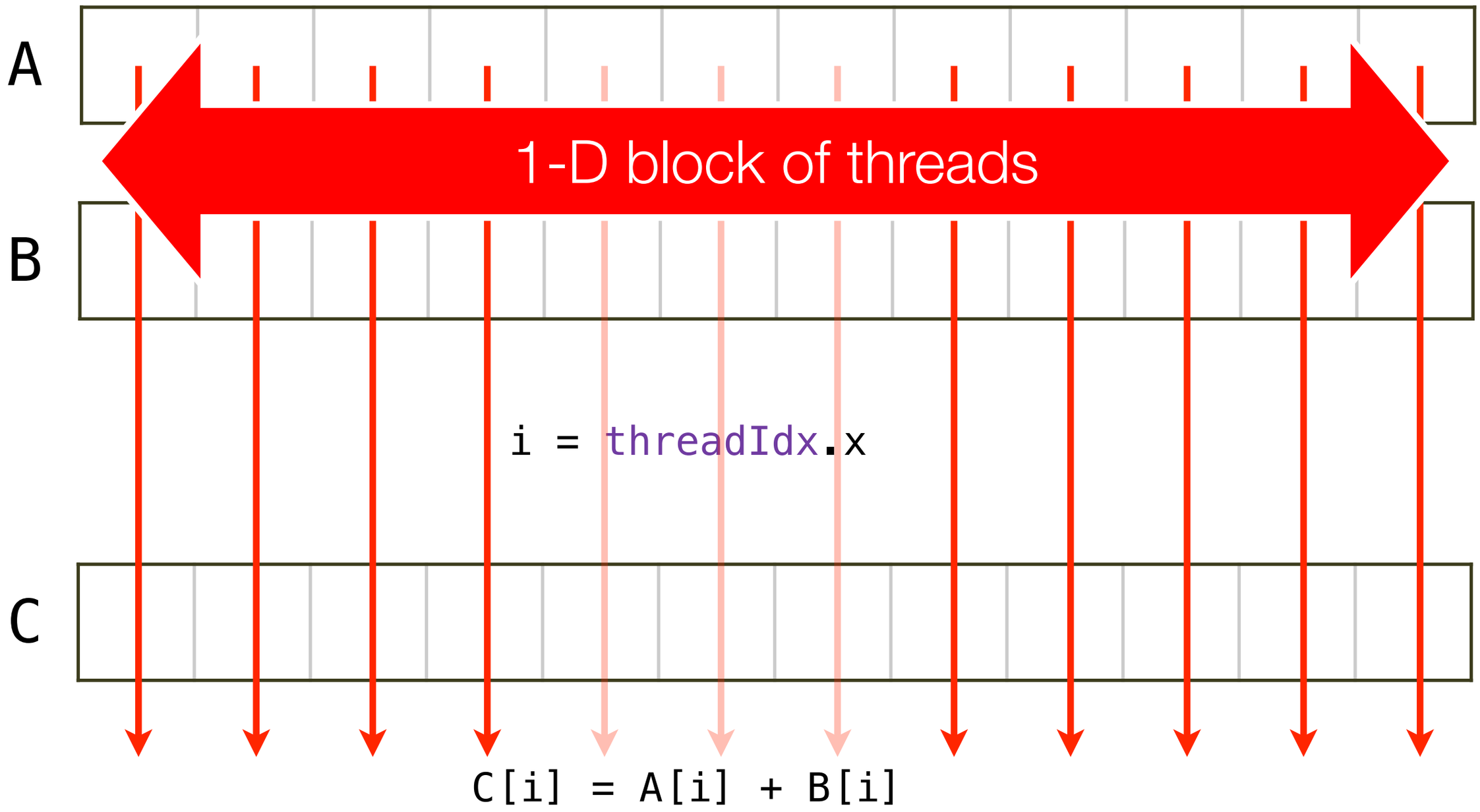


Vector Sum




```
VecAdd<<<1, N>>>(A, B, C);
```

Vector Sum




```
VecAdd<<<1, N>>>(A, B, C);
```

What about branches?

Single control, multiple threads

Branching without Control

Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```

Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```

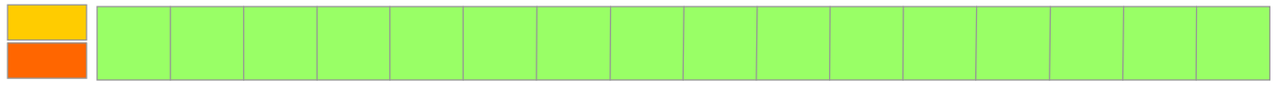
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```

Time

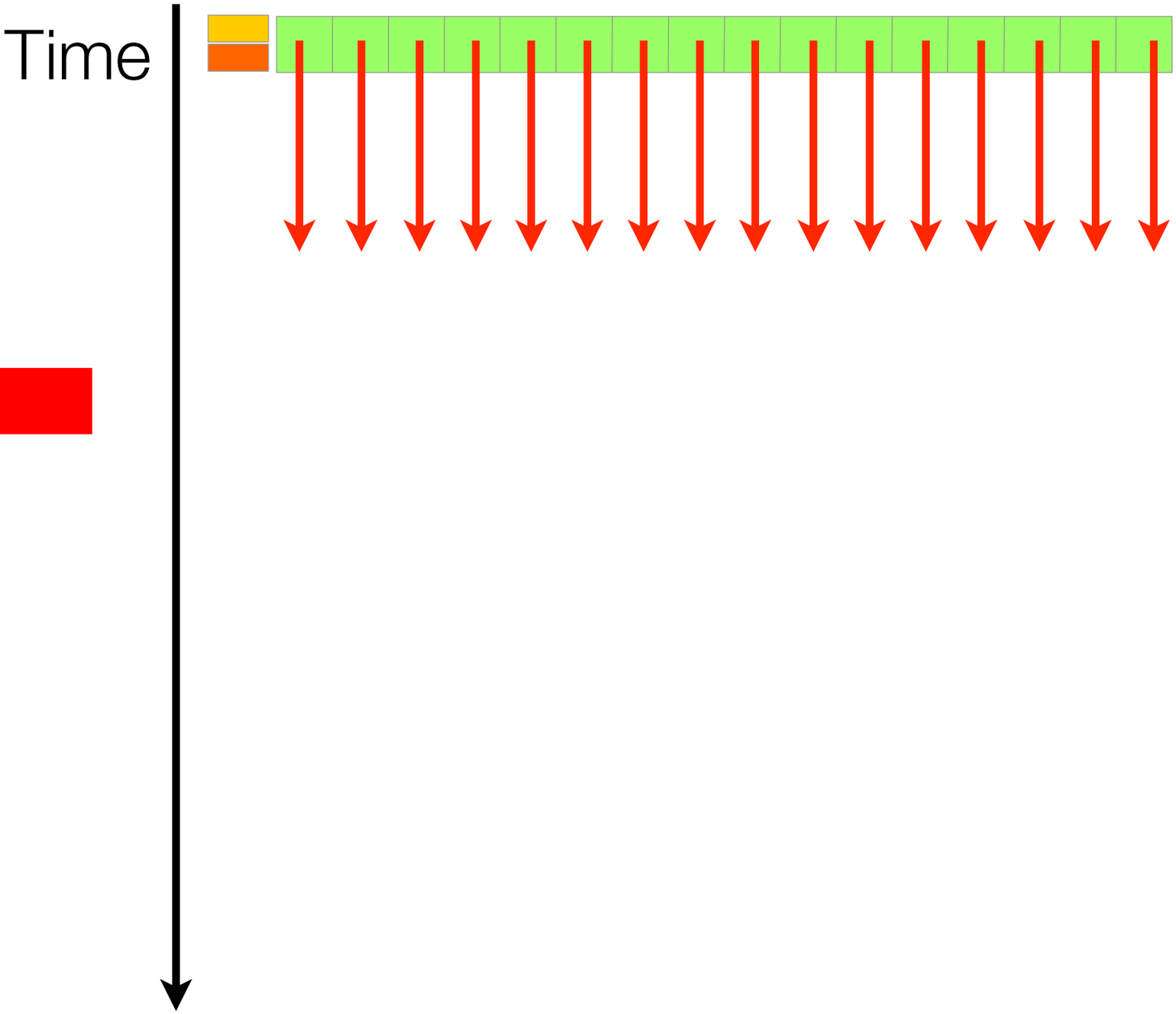
Branching without Control

Time



```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```

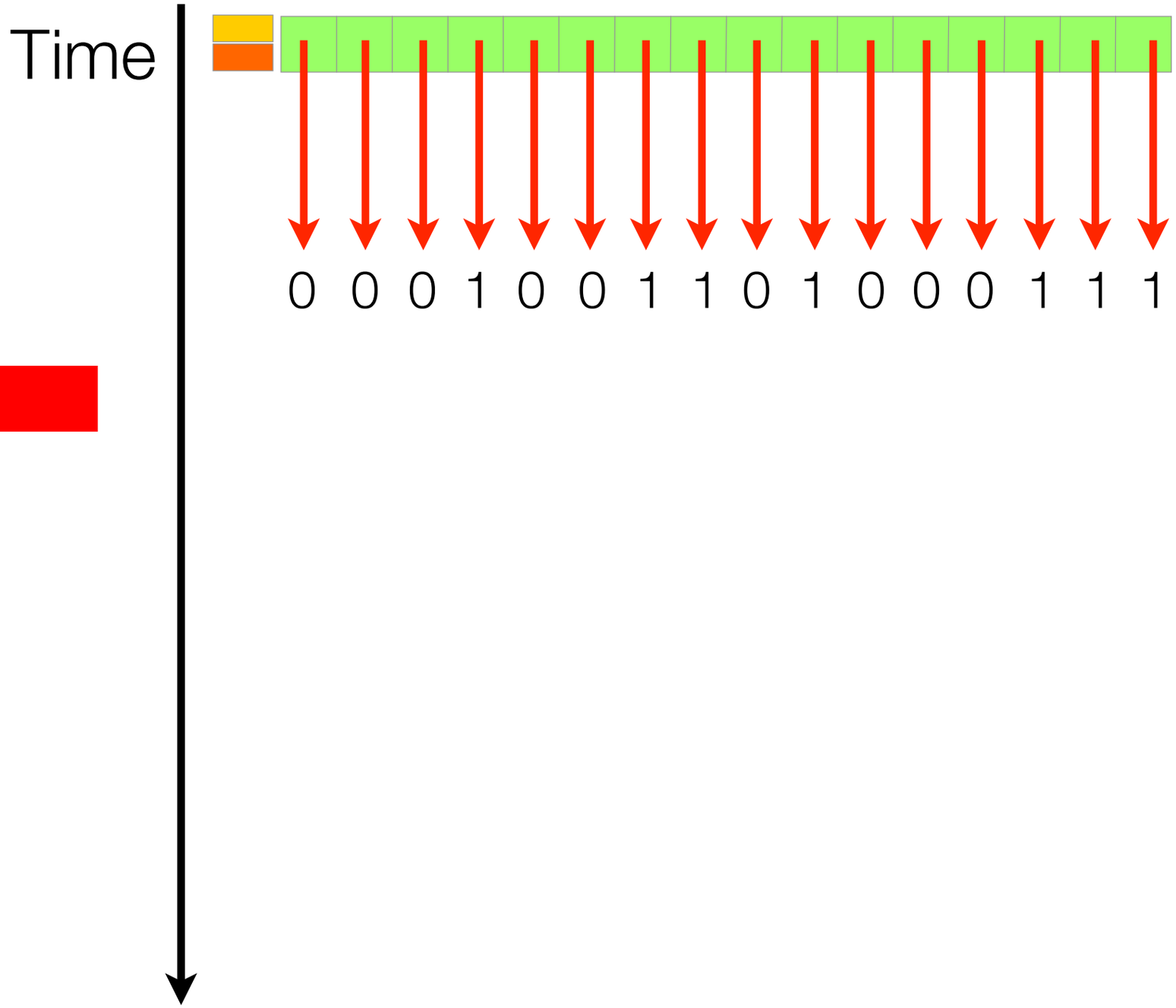
Branching without Control




```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```

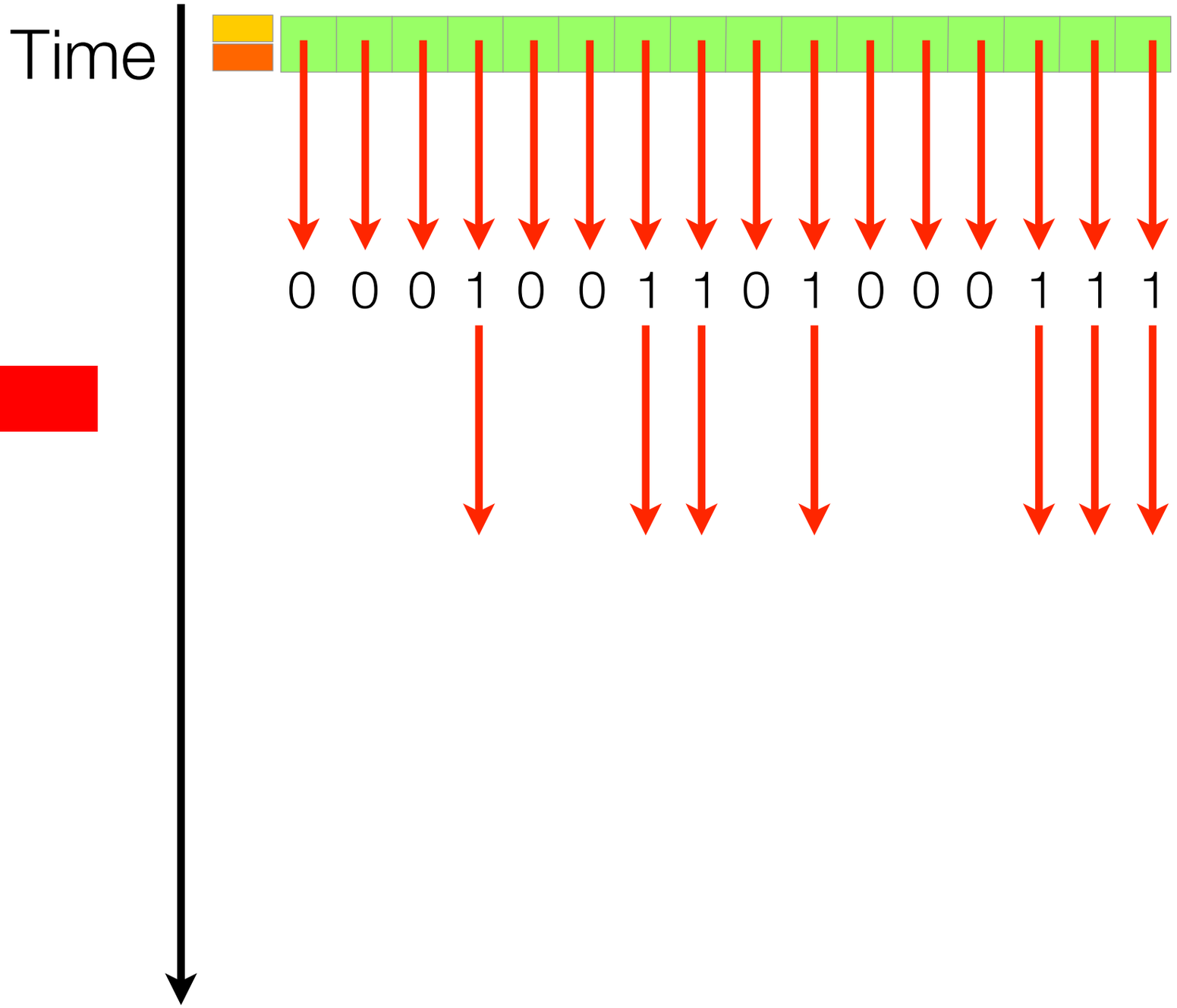
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



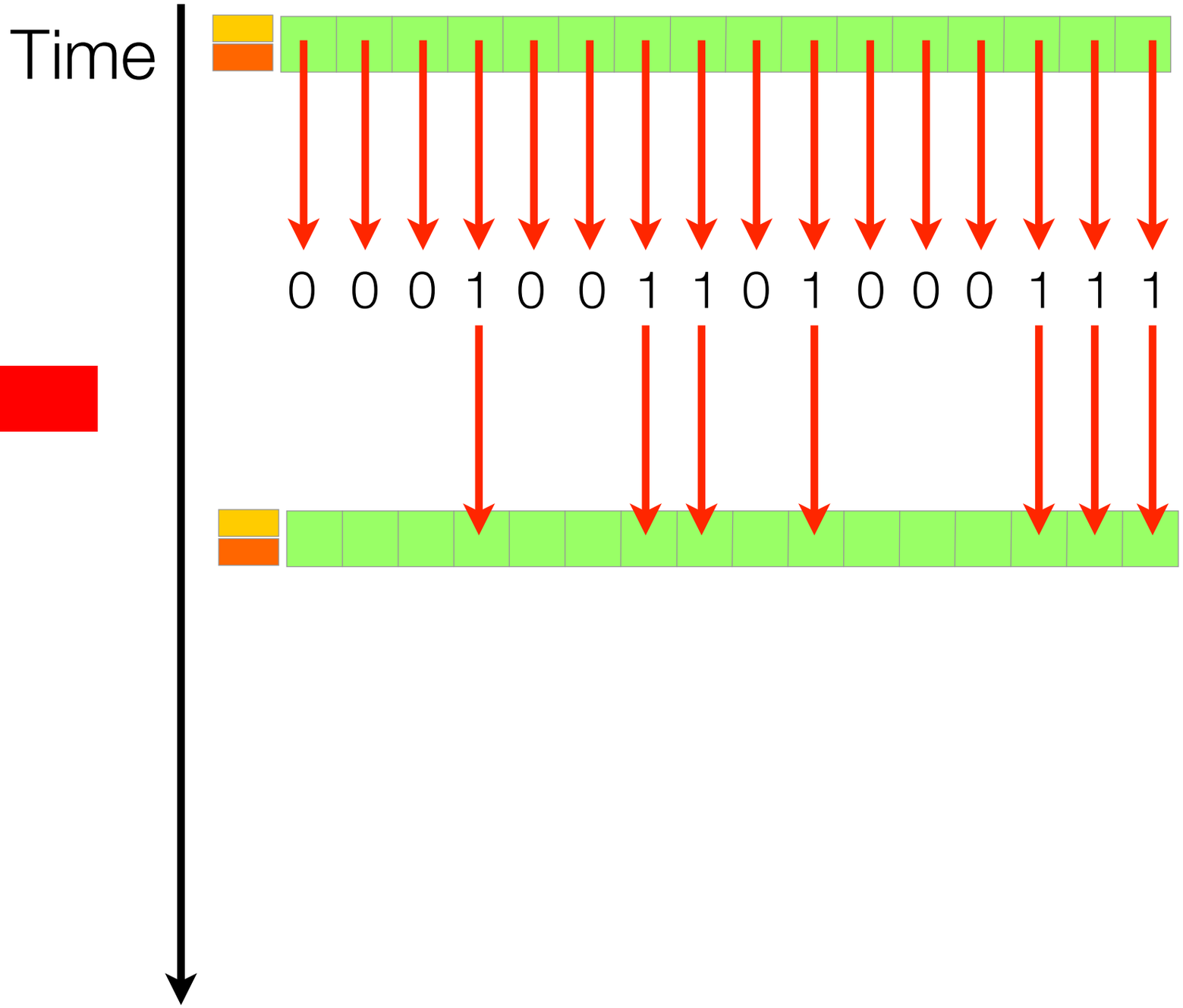
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



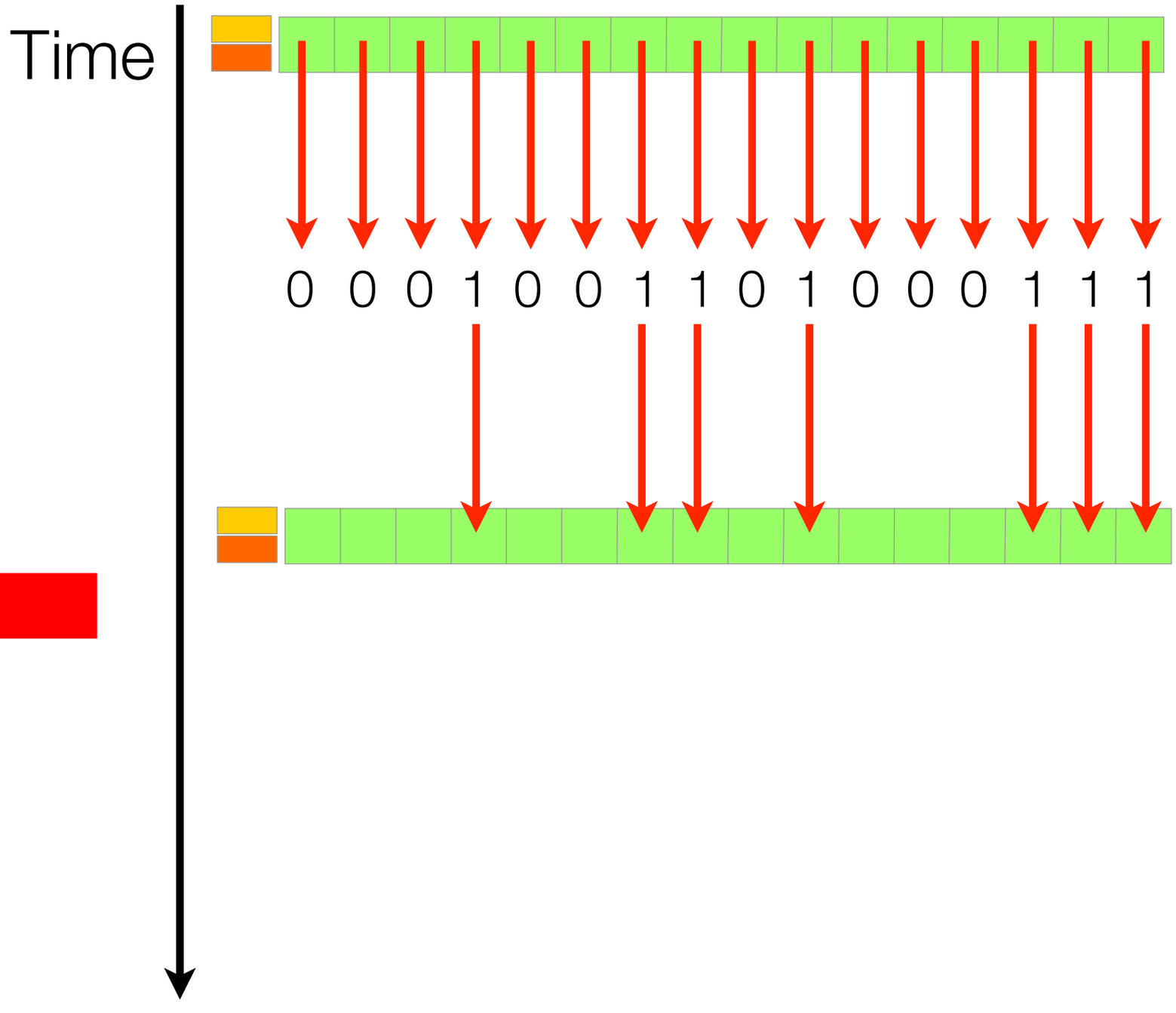
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



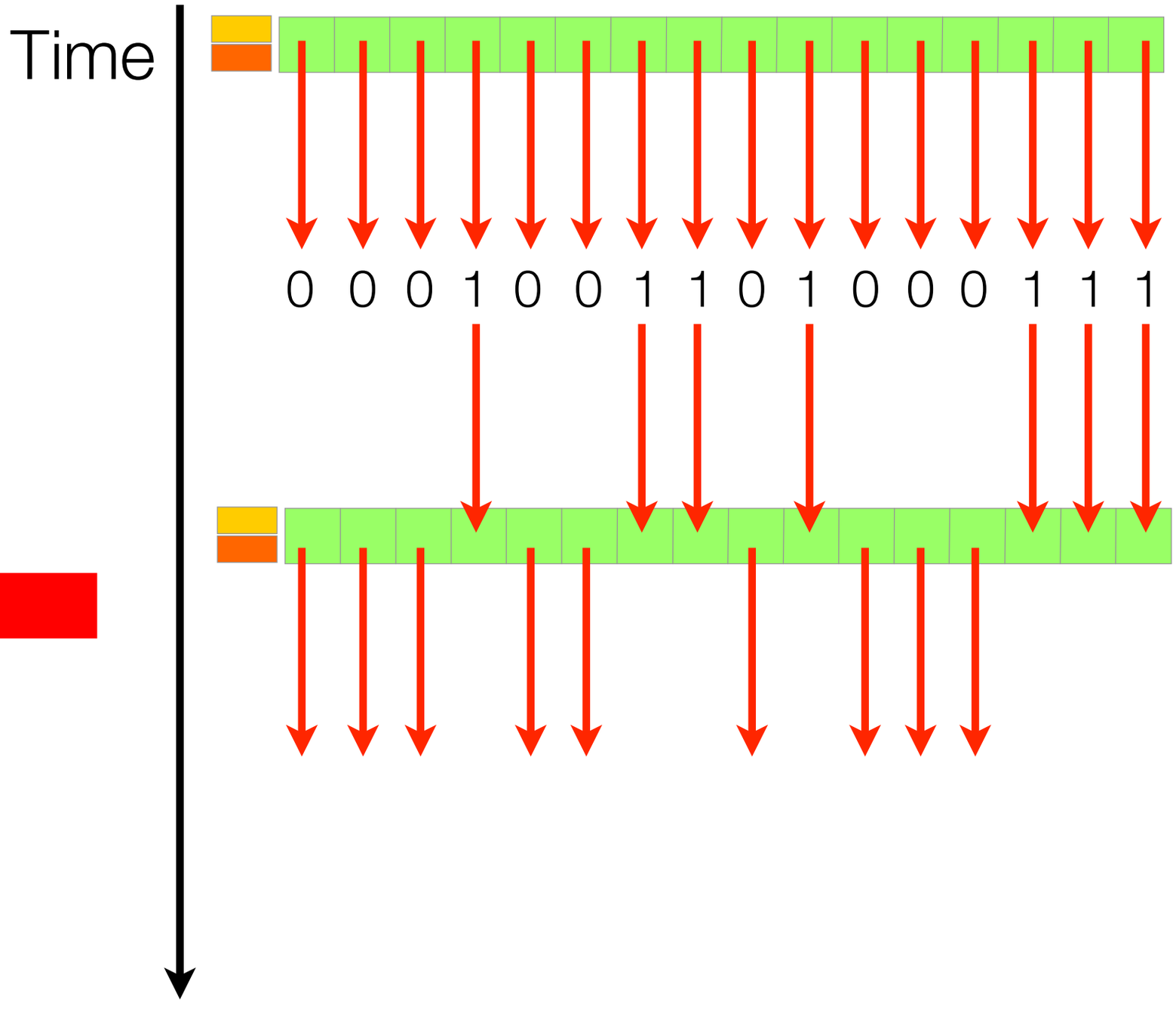
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



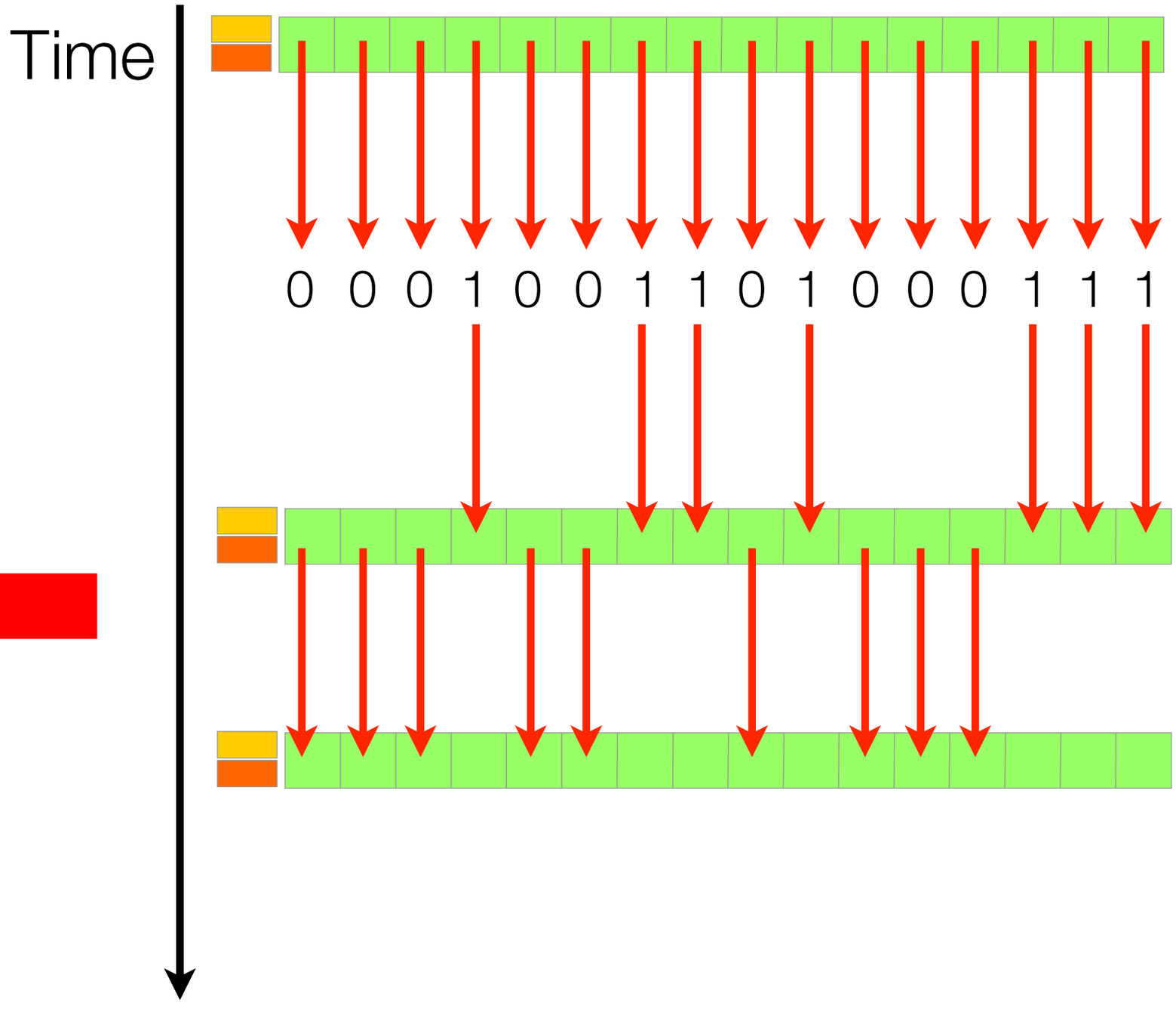
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



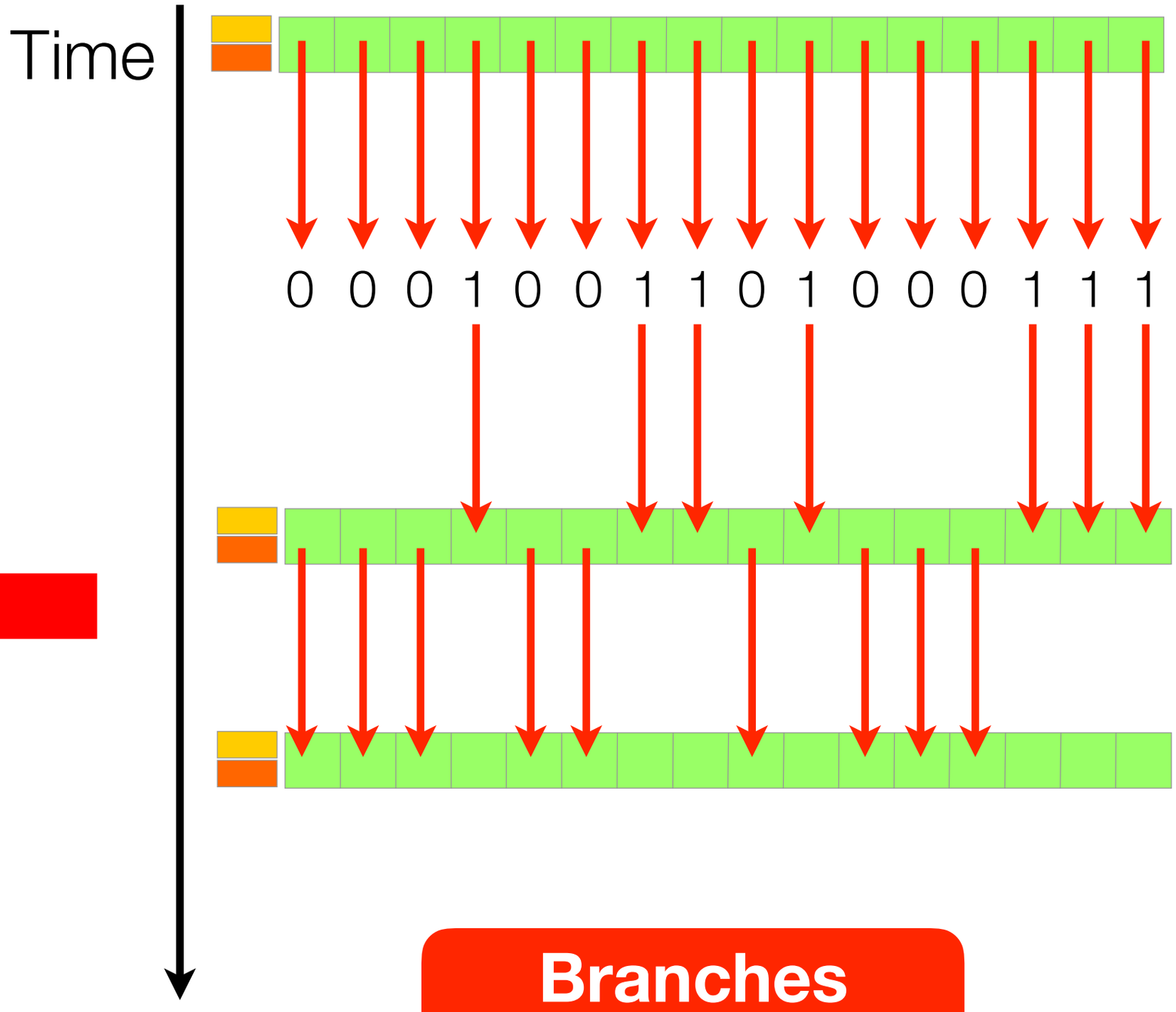
Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



Branching without Control

```
// ...

if (i > 3.1415)
{
   // ...
}
else
{
   // ...
}
```



Branches produce stalls

Cache Effects

Cache Effects

- As we removed the control unit and shared among several execution ones, we’re not able to branch

Cache Effects

- As we removed the control unit and shared among several execution ones, we're not able to branch

- Moreover, we also remove the *memory cache*

Cache Effects

- As we removed the control unit and shared among several execution ones, we're not able to branch
- Moreover, we also remove the *memory cache*
- This means that accessing memory is *hugely expensive*

Cache Effects

- As we removed the control unit and shared among several execution ones, we're not able to branch

- Moreover, we also remove the *memory cache*

- This means that accessing memory is *hugely expensive*

- However, we built *lots and lots* of threads

Cache Effects

- As we removed the control unit and shared among several execution ones, we're not able to branch
- Moreover, we also remove the *memory cache*
- This means that accessing memory is *hugely expensive*
- However, we built *lots and lots* of threads
- So, all these problems are ameliorated by the numbers

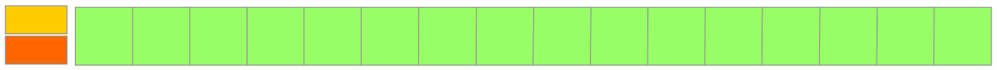
No Caching World

No Caching World

Time

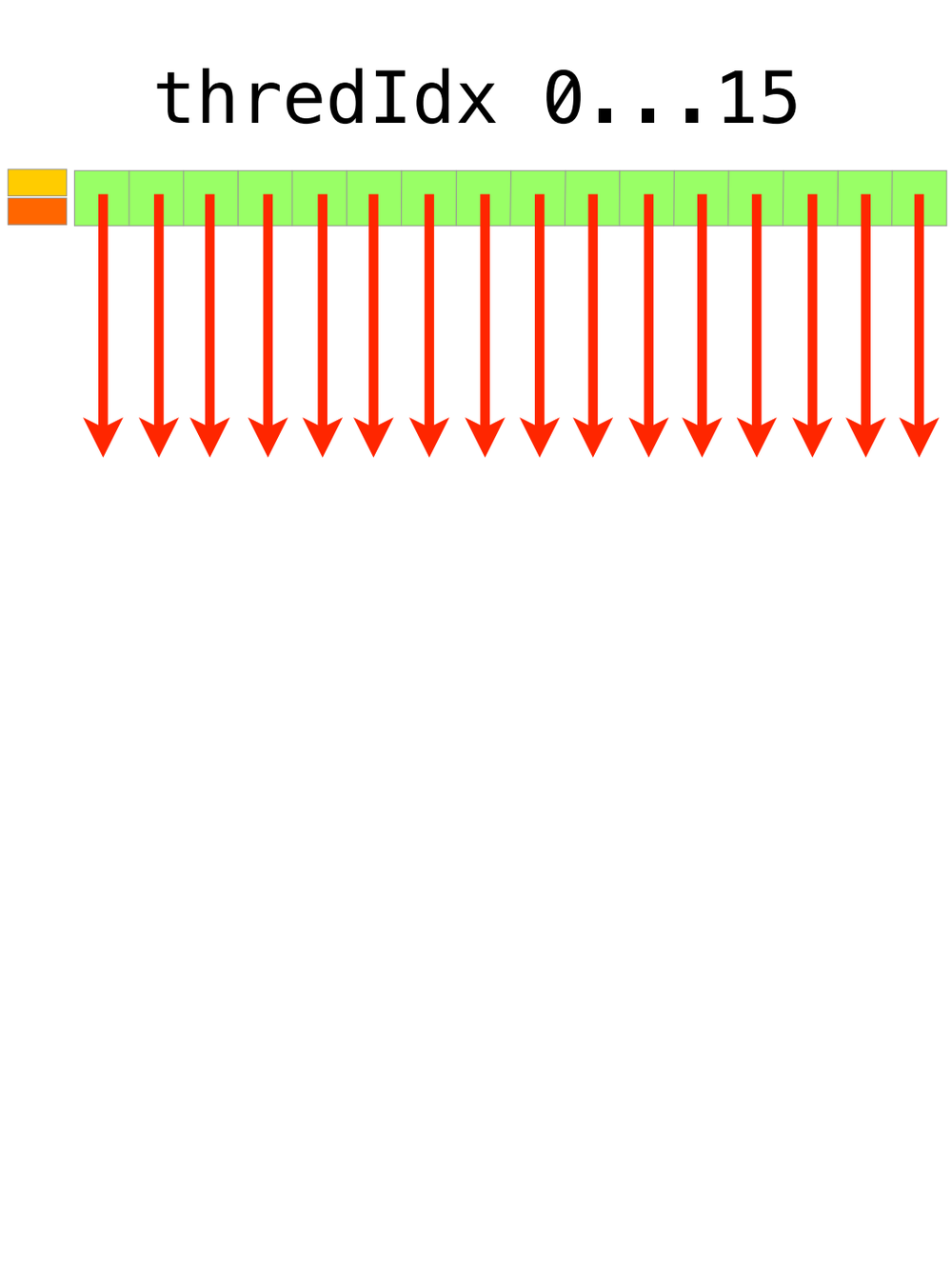
No Caching World

Time

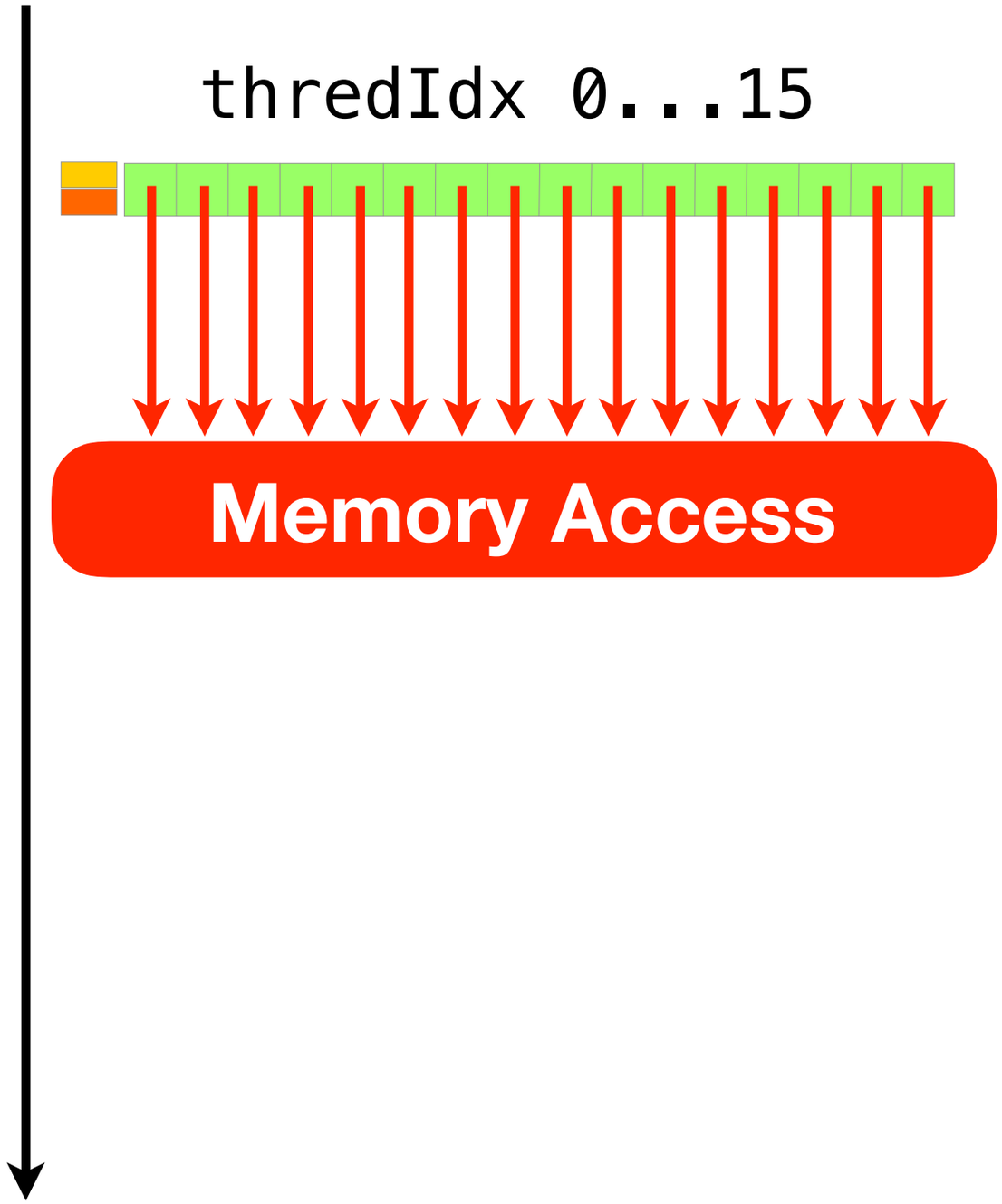


No Caching World

Time

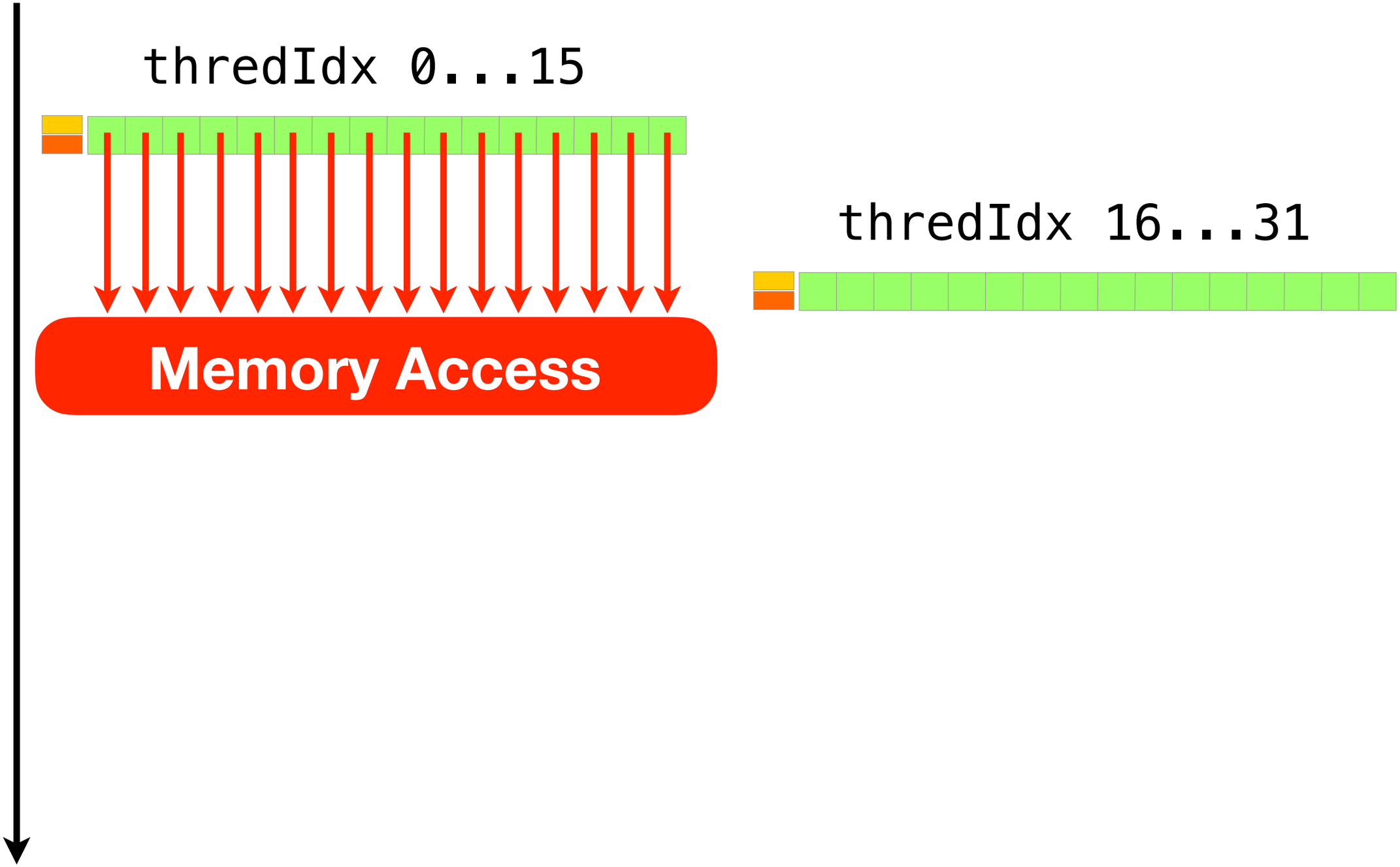


No Caching World



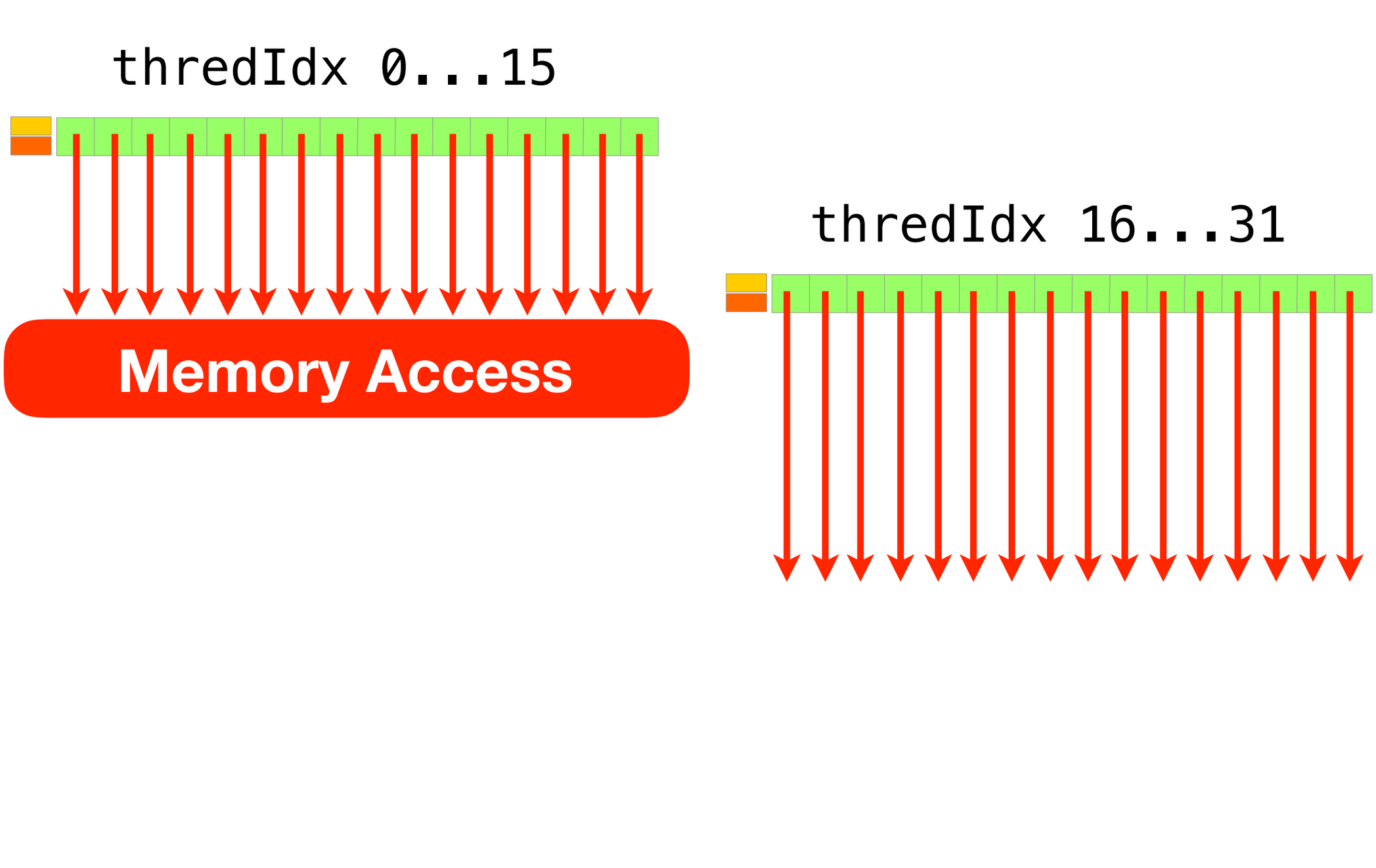
No Caching World

Time



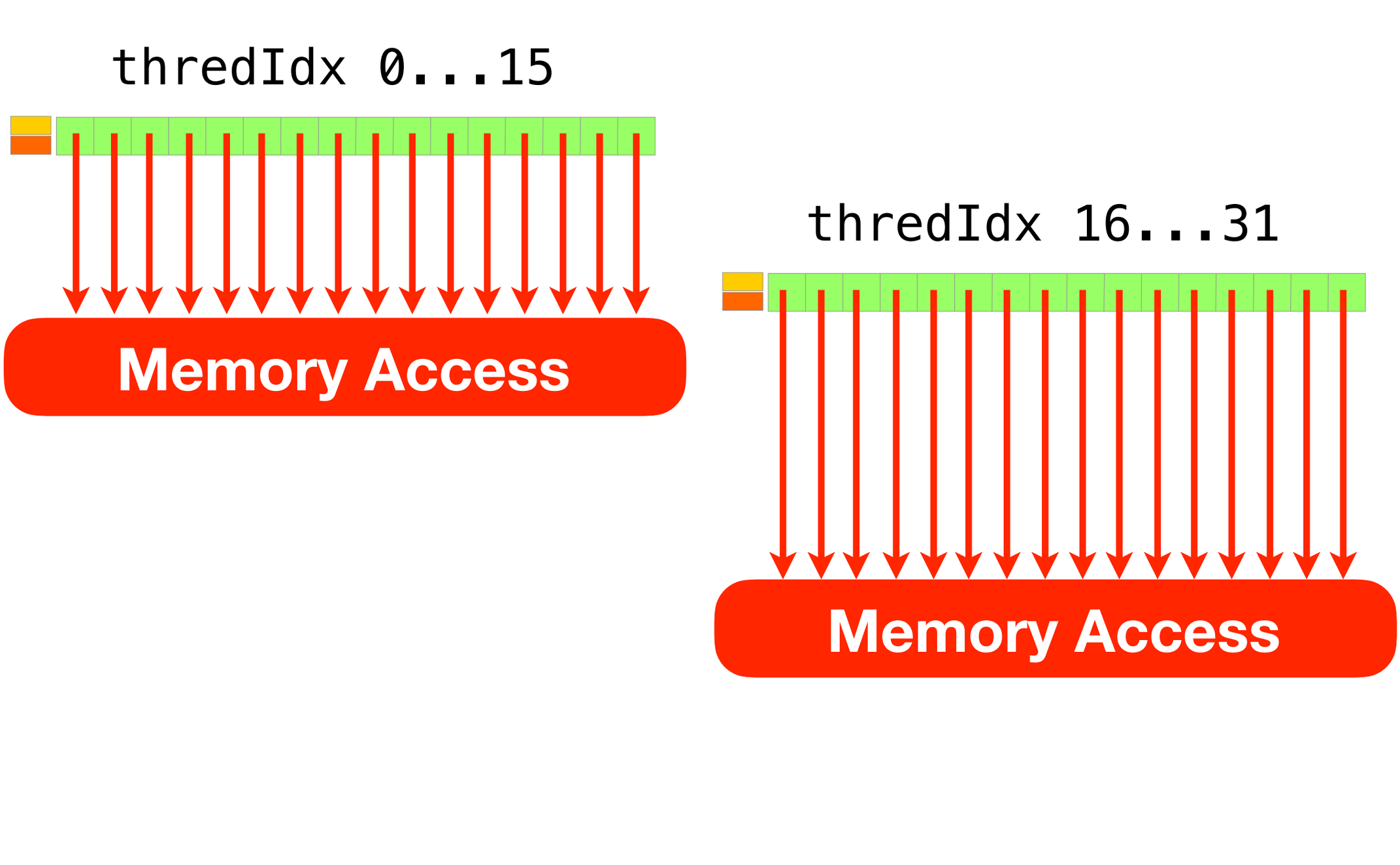
No Caching World

Time



No Caching World

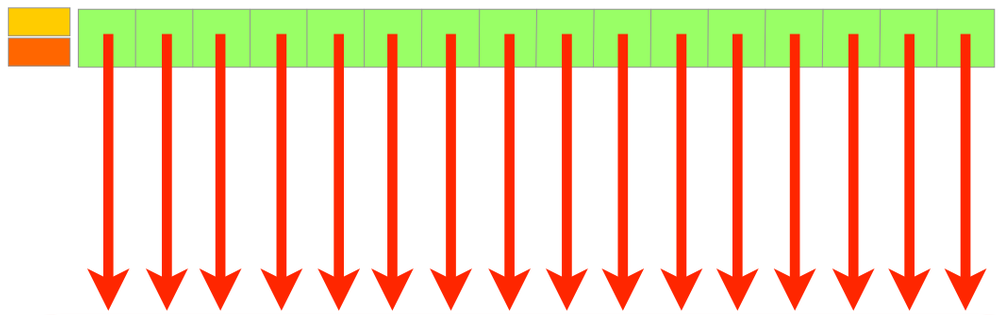
Time



No Caching World

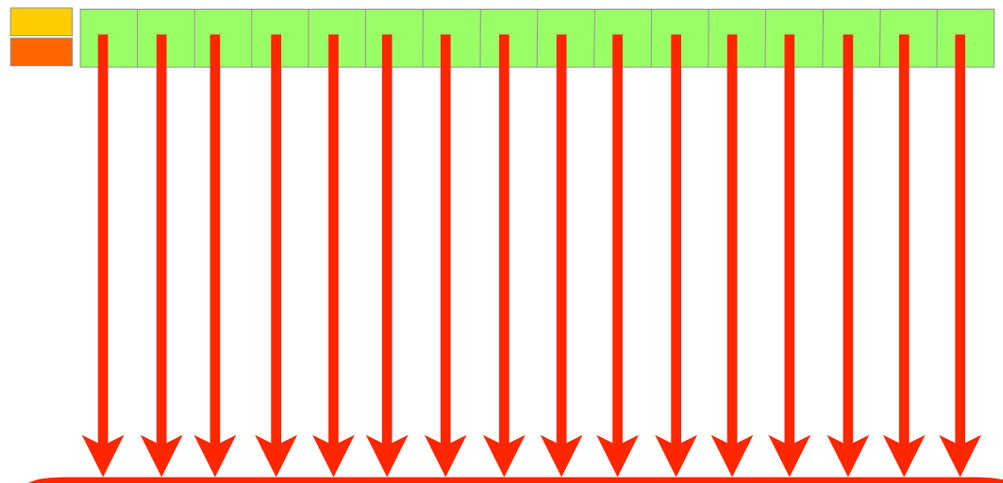
Time

thredIdx 0...15



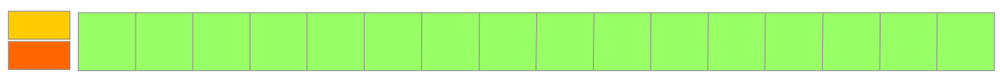
Memory Access

thredIdx 16...31

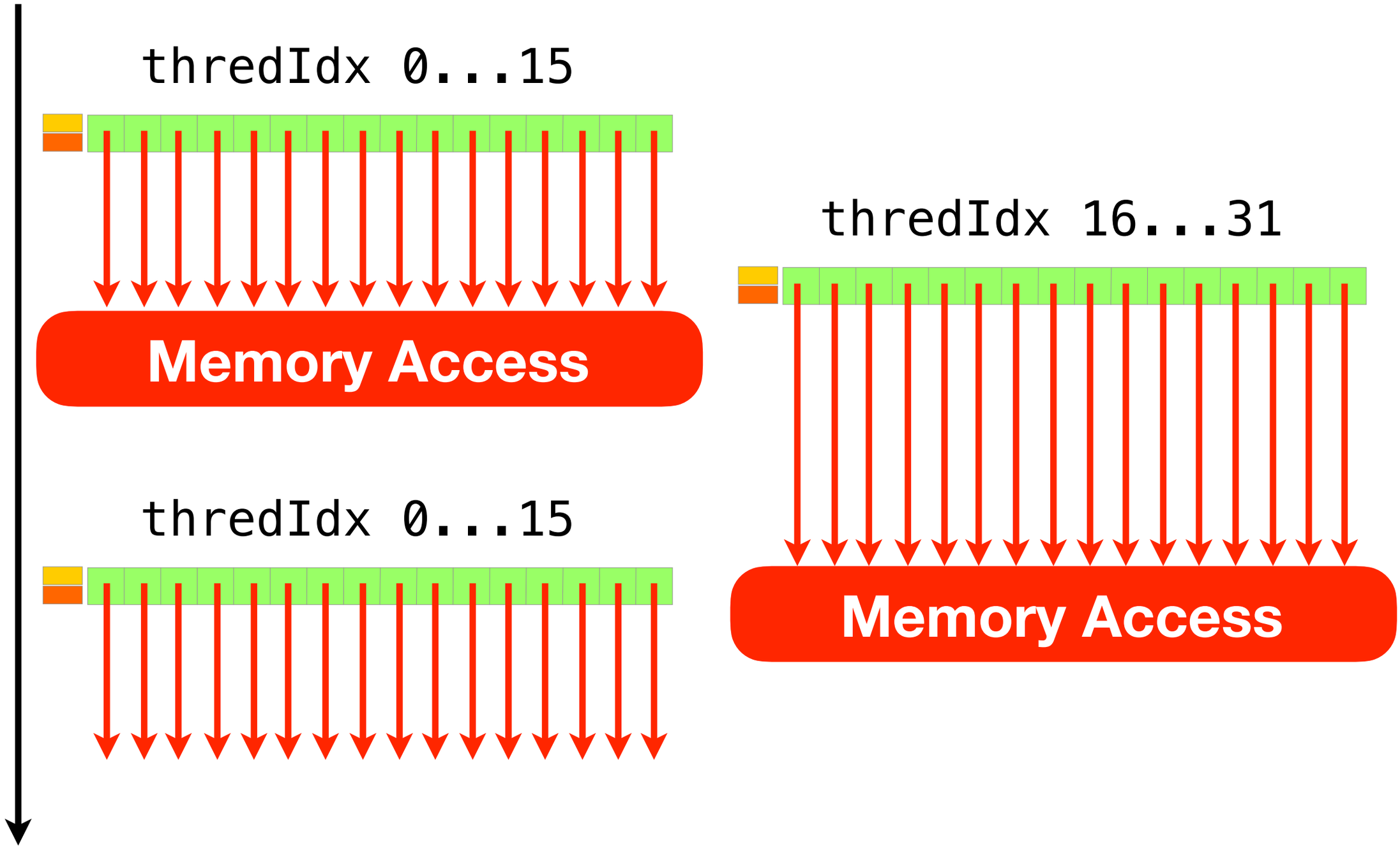


Memory Access

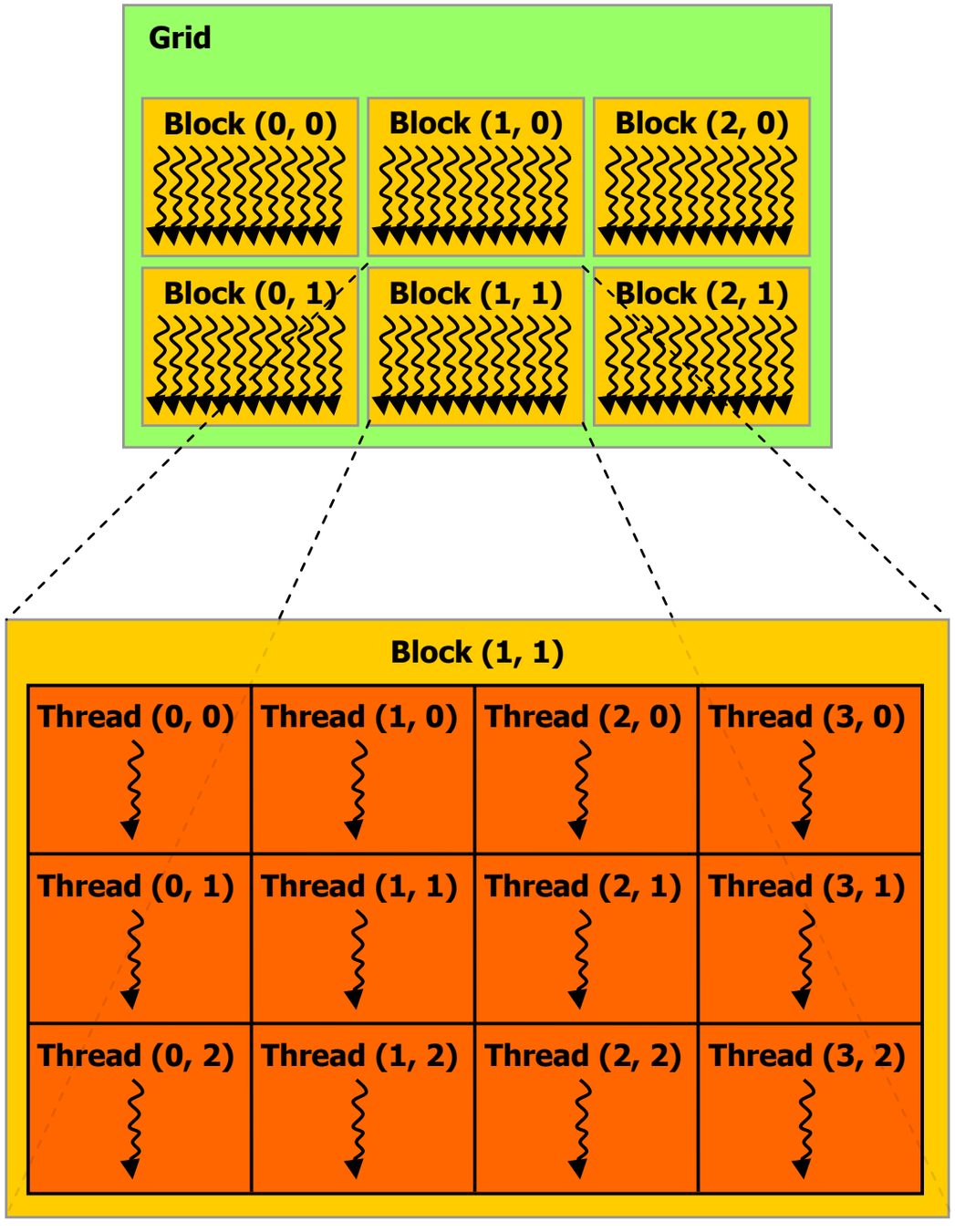
thredIdx 0...15



No Caching World

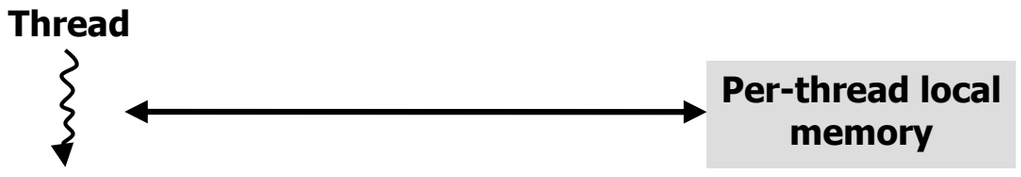


Thread Blocks

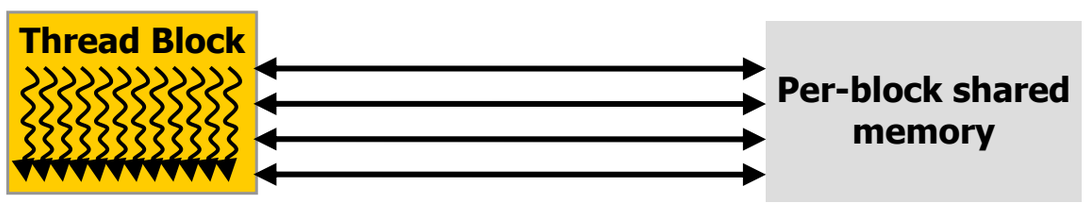


Memory Model

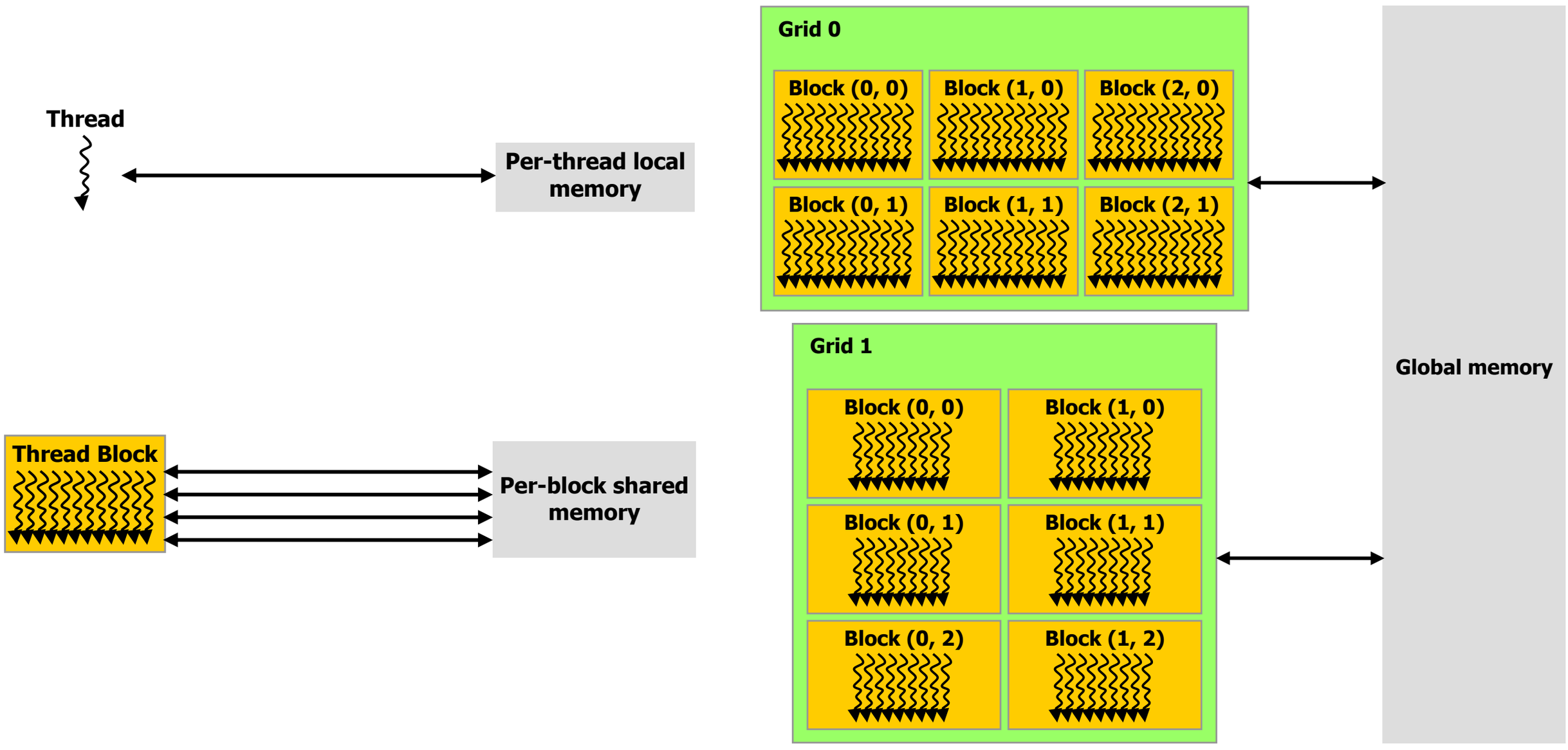
Memory Model



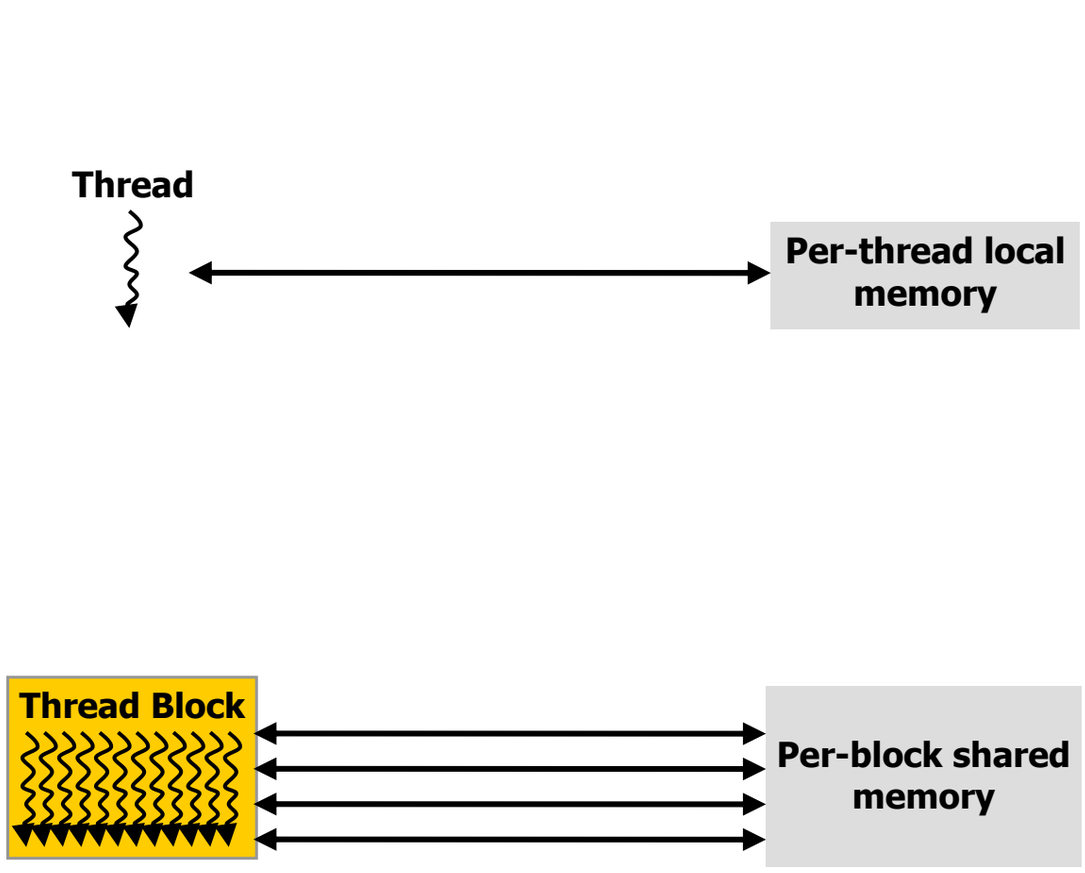
Memory Model

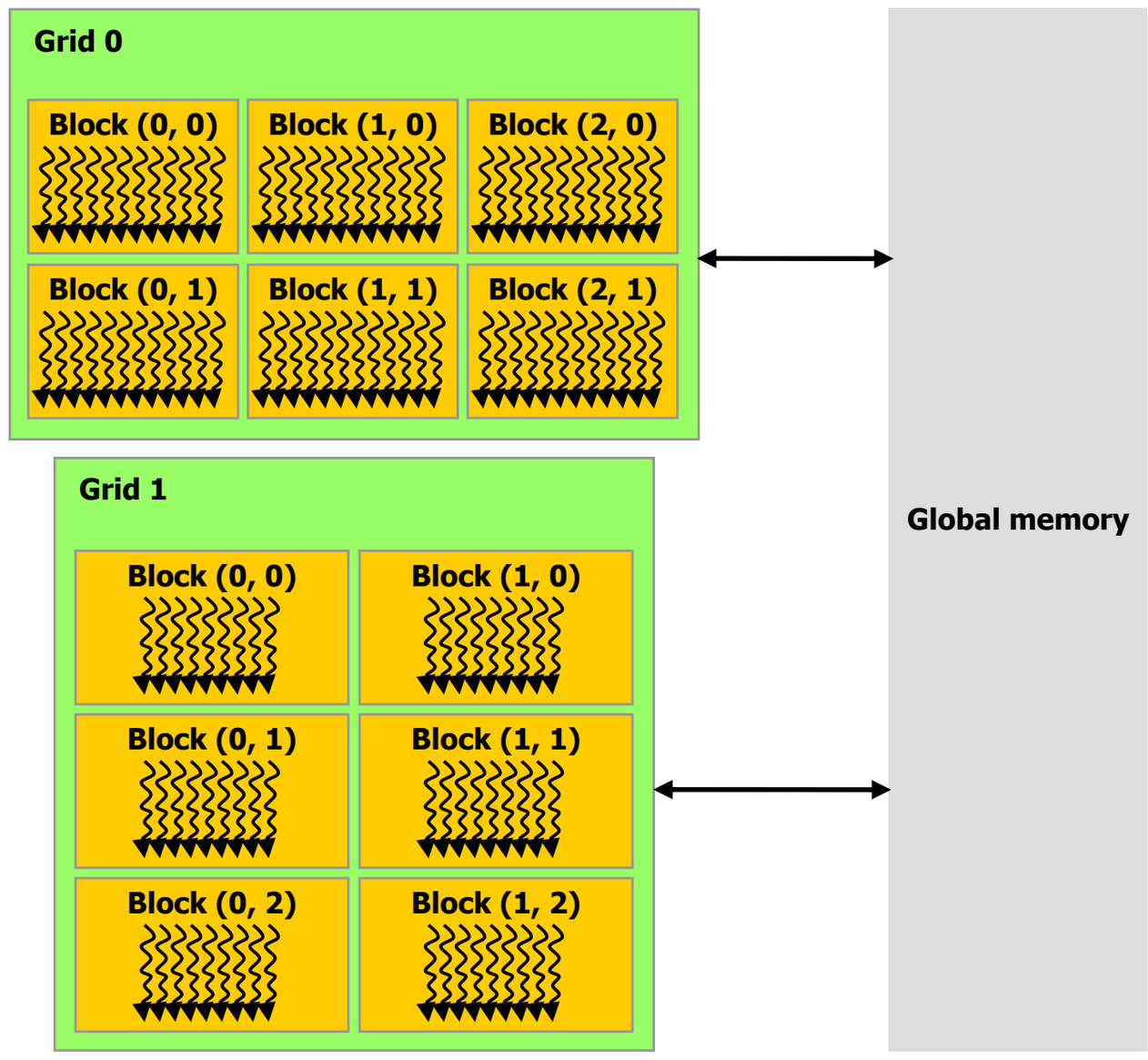


Memory Model



Memory Model





In Nvidia's jargon, 32 threads are called a *warp*

Limits and Threads

Limits and Threads

- Nvidia uses *threads* and *blocks* in a *grid* to help programmers map their problem to the GPU

Limits and Threads

- Nvidia uses *threads* and *blocks* in a *grid* to help programmers map their problem to the GPU

- The number of threads *per block is limited* by the hardware

Limits and Threads

- Nvidia uses *threads* and *blocks* in a *grid* to help programmers map their problem to the GPU

- The number of threads *per block is limited* by the hardware

- Nowadays, GPUs may support 512 threads per block

Limits and Threads

- Nvidia uses *threads* and *blocks* in a *grid* to help programmers map their problem to the GPU

- The number of threads *per block is limited* by the hardware

- Nowadays, GPUs may support 512 threads per block

- Kernels can be executed simultaneously in several blocks

Limits and Threads

- Nvidia uses *threads* and *blocks* in a *grid* to help programmers map their problem to the GPU

- The number of threads *per block is limited* by the hardware

- Nowadays, GPUs may support 512 threads per block

- Kernels can be executed simultaneously in several blocks

- Blocks can be arranged in arrays or grids, as developers see suitable for their case

Blocks and Warps: Scheduling

Blocks and Warps: Scheduling

- Blocks are identified by a variable called `blockIdx`

Blocks and Warps: Scheduling

- Blocks are identified by a variable called `blockIdx`
- Each block can run a *limited number of threads*

Blocks and Warps: Scheduling

- Blocks are identified by a variable called `blockIdx`
- Each block can run a *limited number of threads*
- A warp is a group of 32 threads: this helps the hardware scheduler to execute a *huge number of threads*

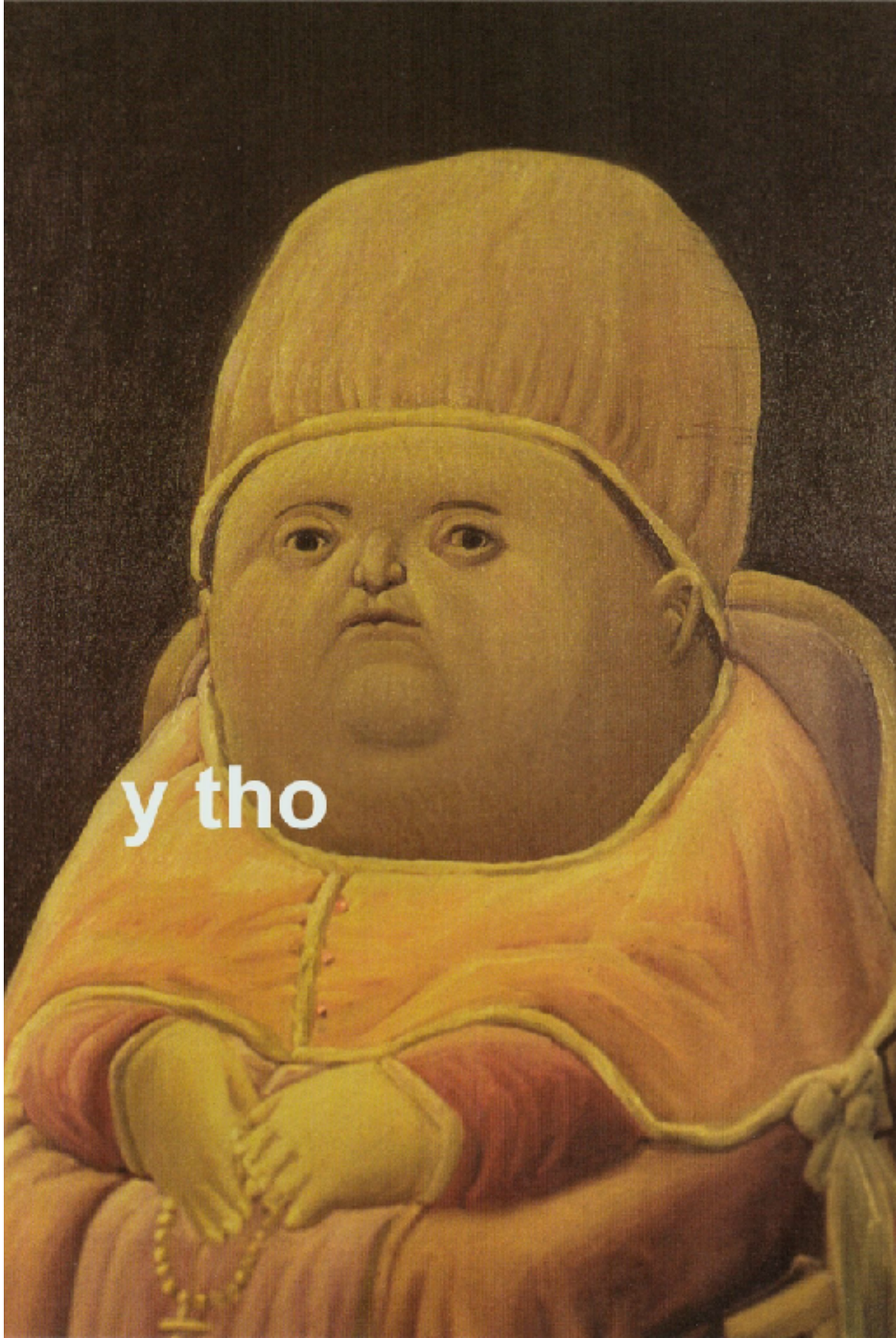
Blocks and Warps: Scheduling

- Blocks are identified by a variable called `blockIdx`
- Each block can run a *limited number of threads*
- A warp is a group of 32 threads: this helps the hardware scheduler to execute a *huge number of threads*
- Developers have *no influence* on the scheduler except for synchronization between threads

Blocks and Warps: Scheduling

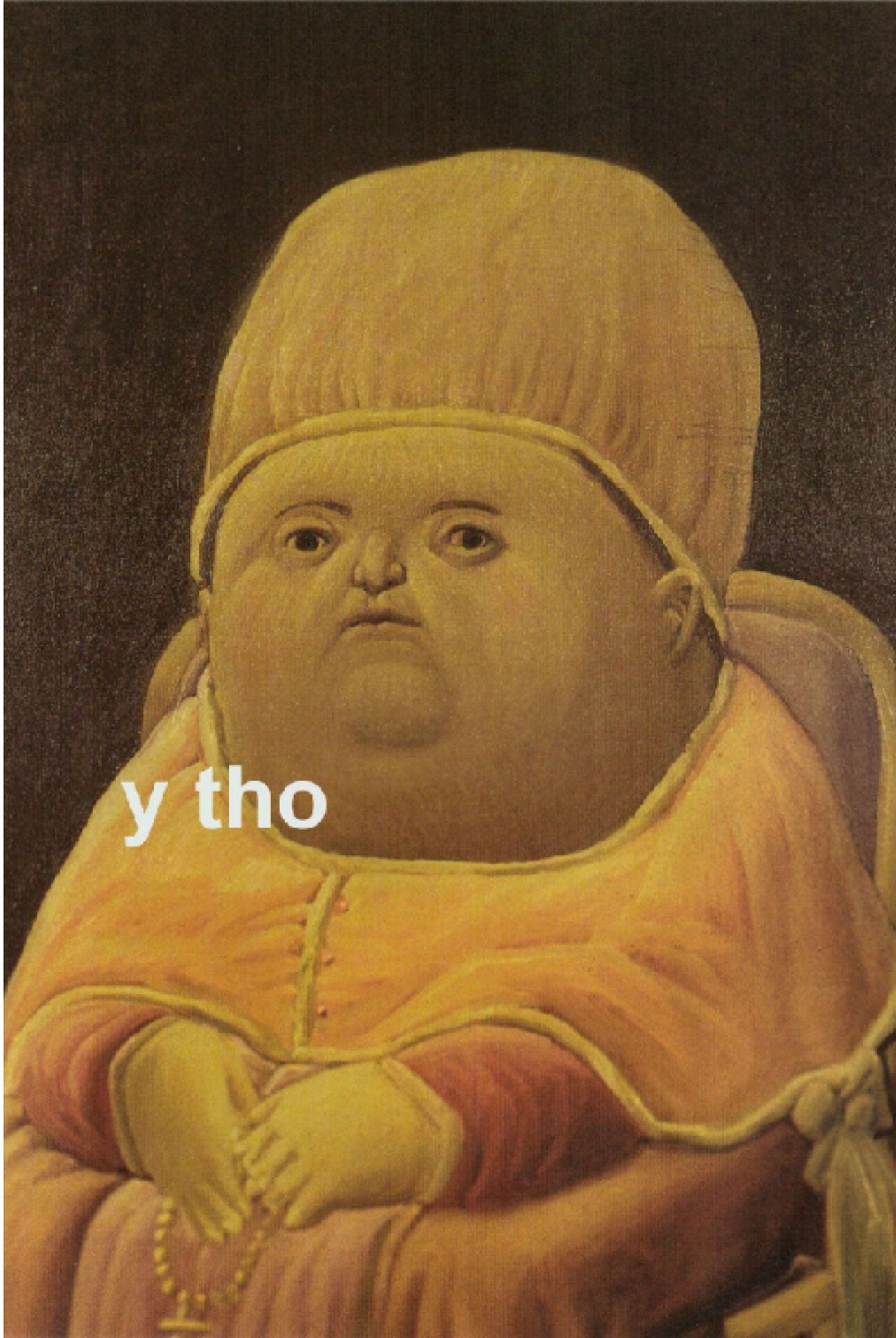
- Blocks are identified by a variable called `blockIdx`
- Each block can run a *limited number of threads*
- A warp is a group of 32 threads: this helps the hardware scheduler to execute a *huge number of threads*
- Developers have *no influence* on the scheduler except for synchronization between threads
- Each grid can be 1D, 2D, or 3D although this is just convenient from the programmer's point of view

Jobs with CUDA



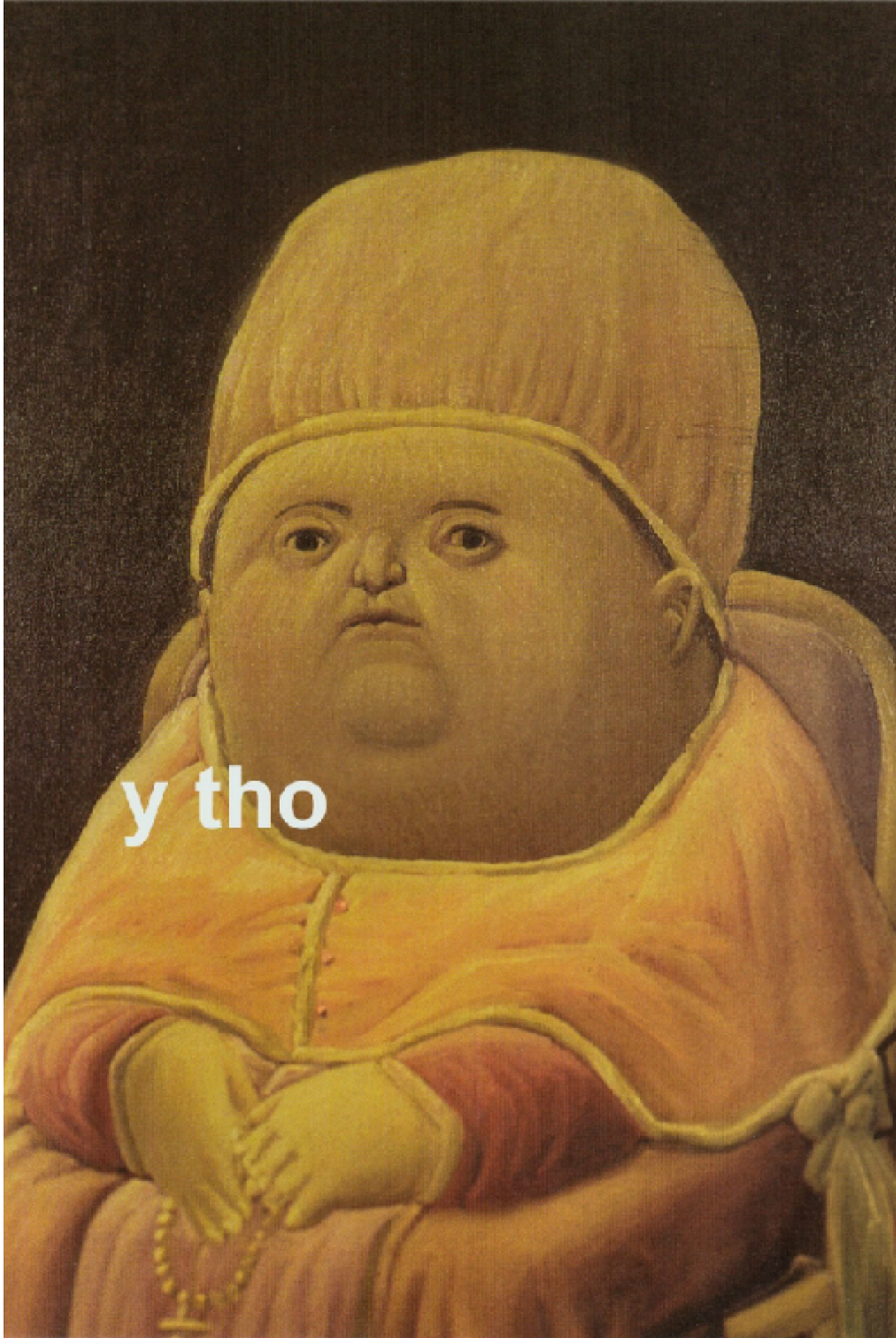
Jobs with CUDA

- Physics and Graphics



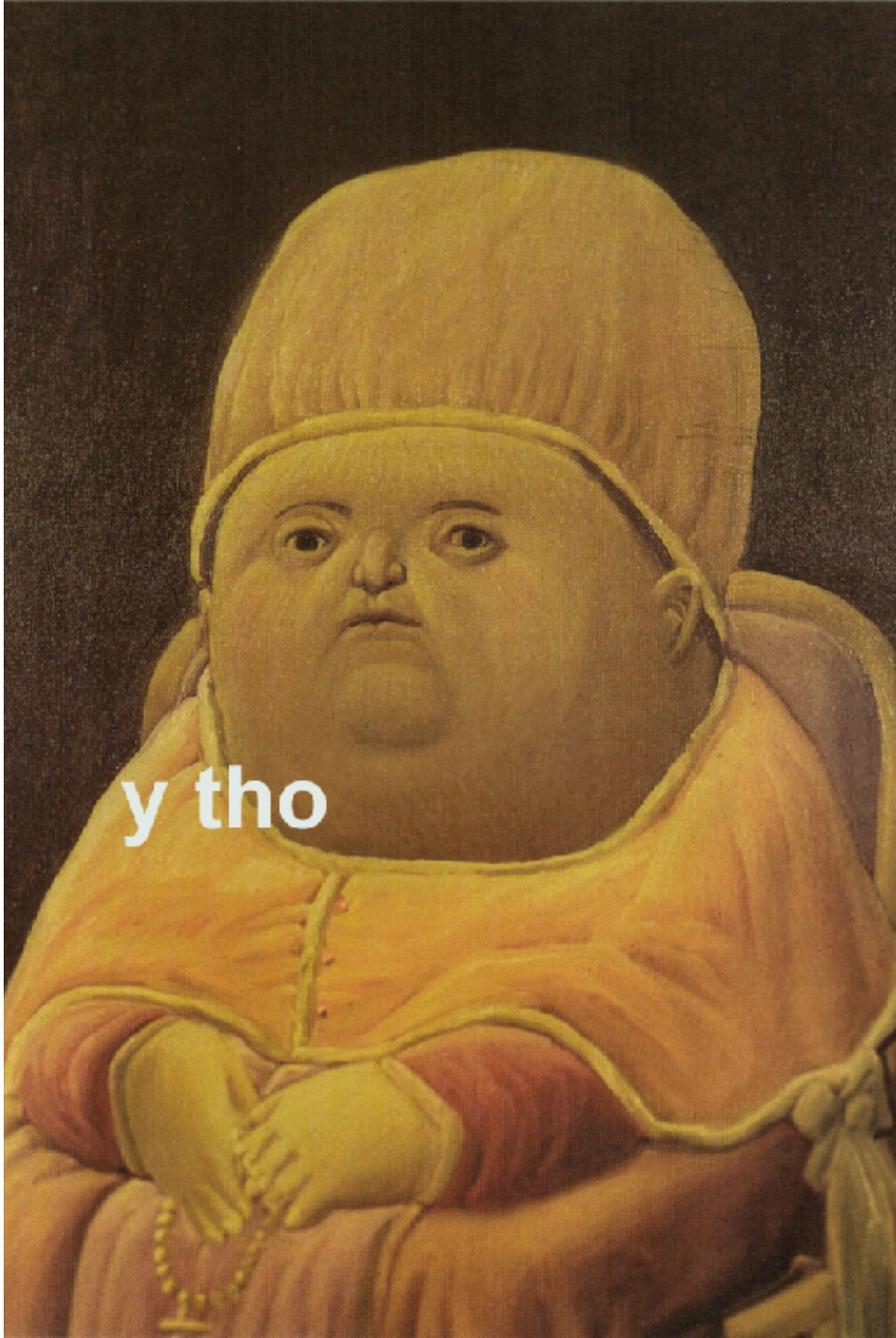
Jobs with CUDA

- Physics and Graphics
- Scientific Communities



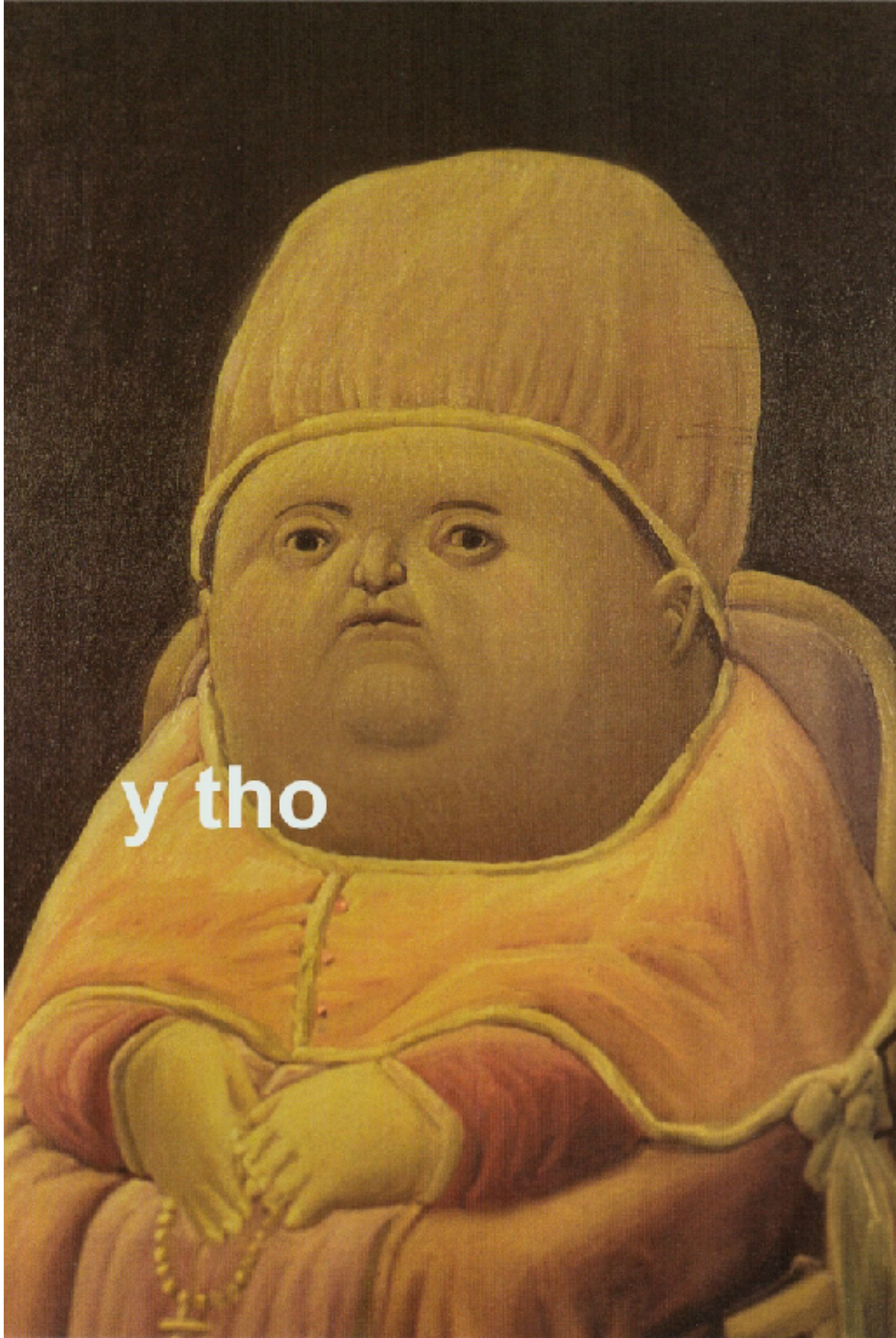
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare



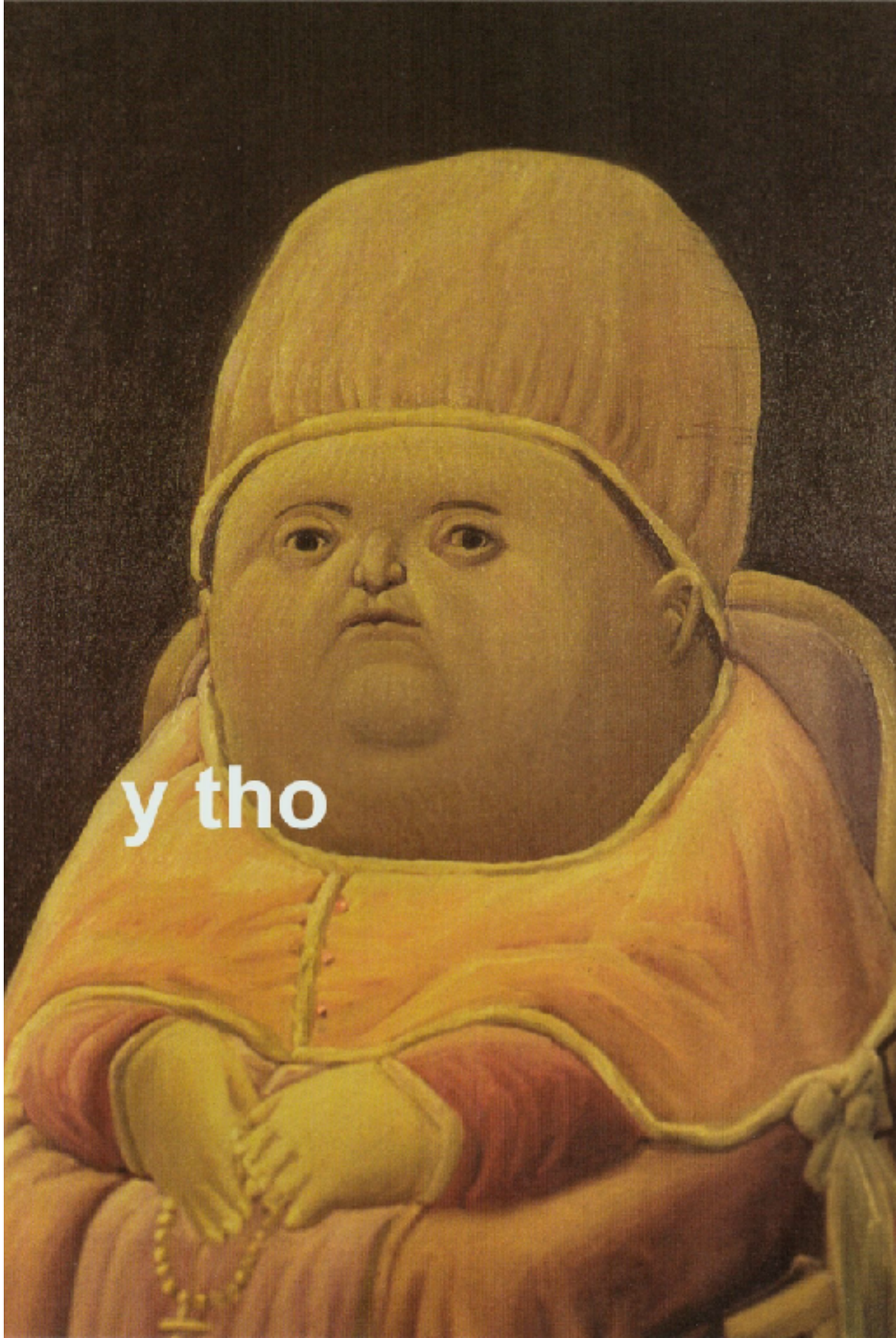
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy



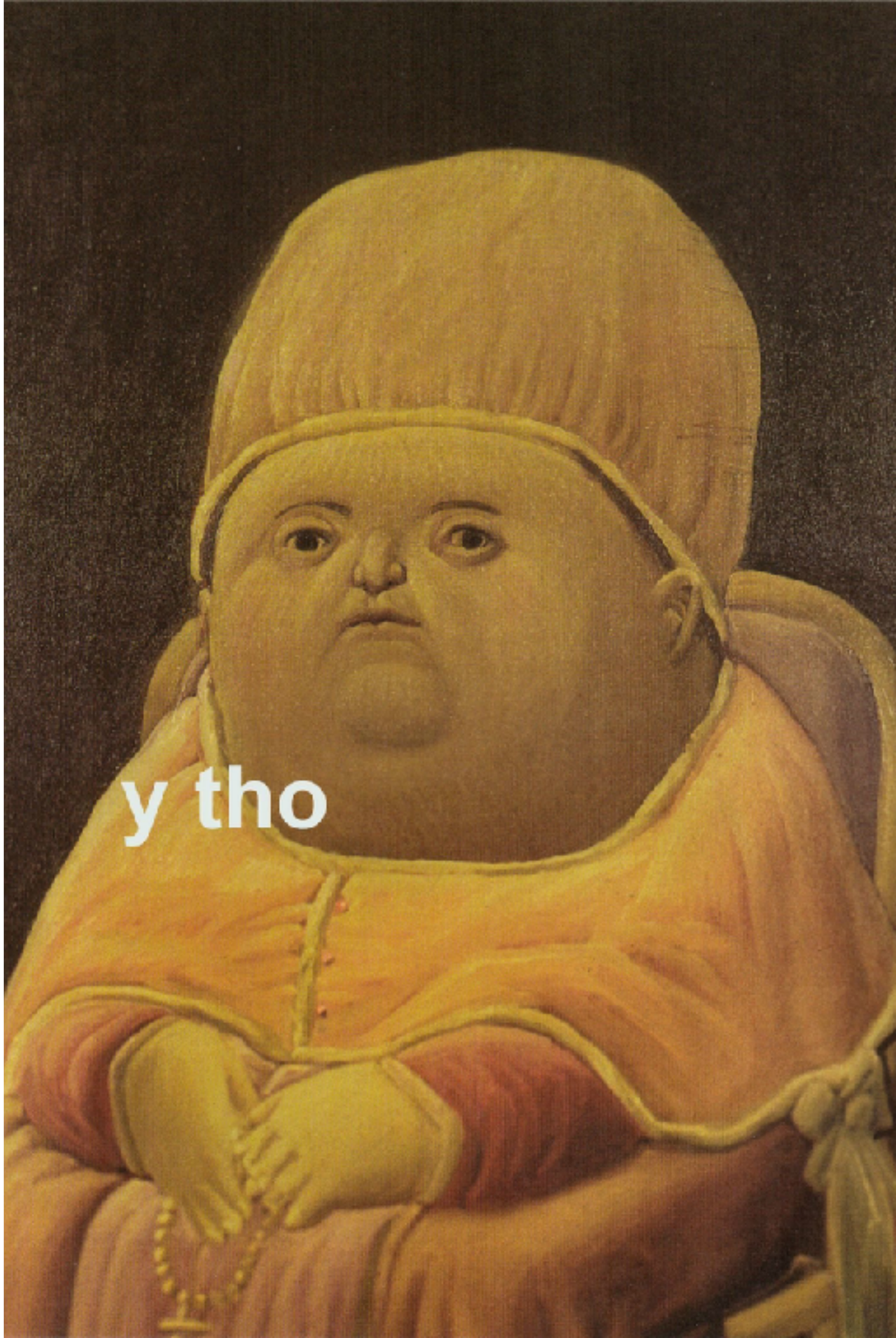
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy
- Audio-video Processors



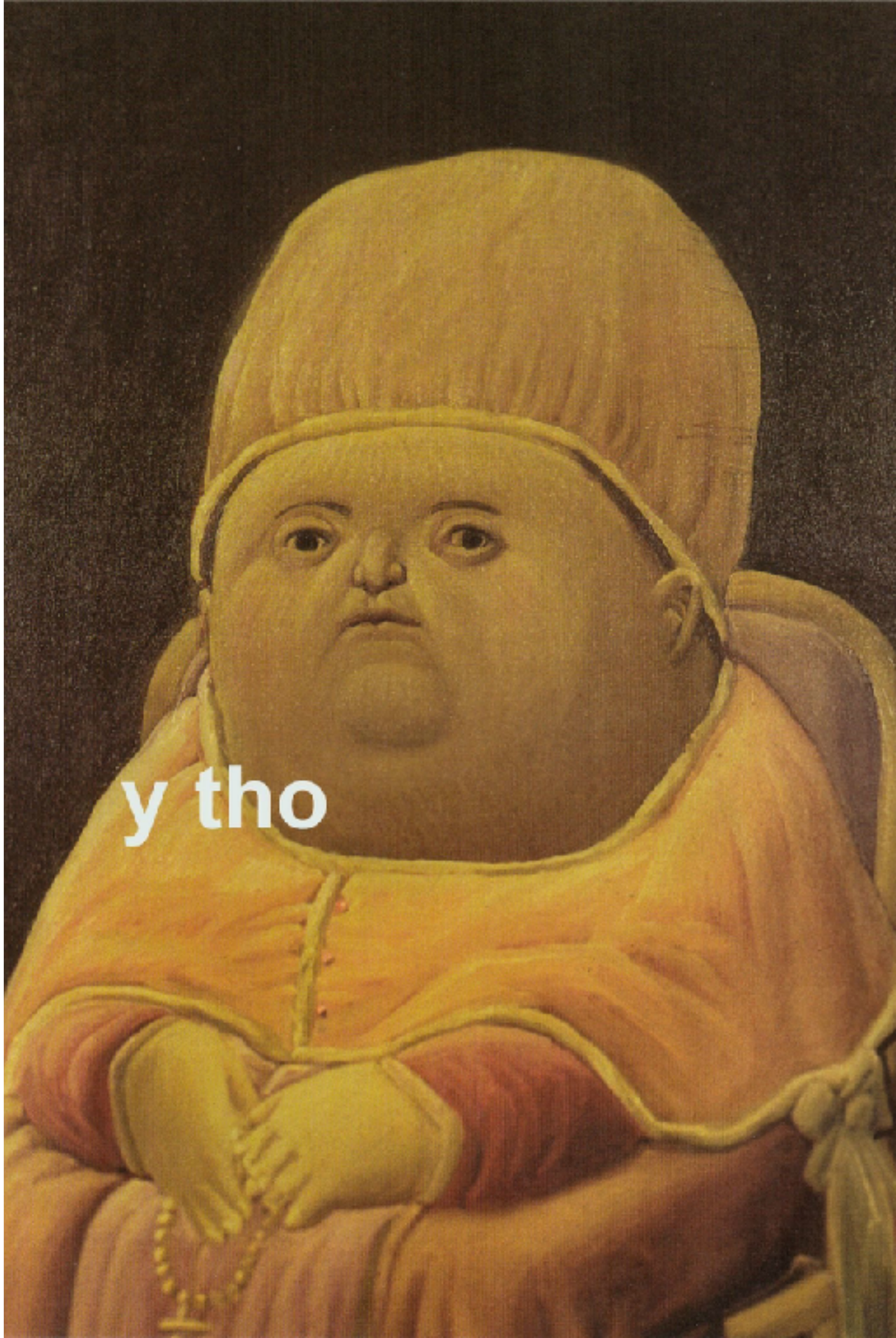
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy
- Audio-video Processors
- Computer Vision and Imaging



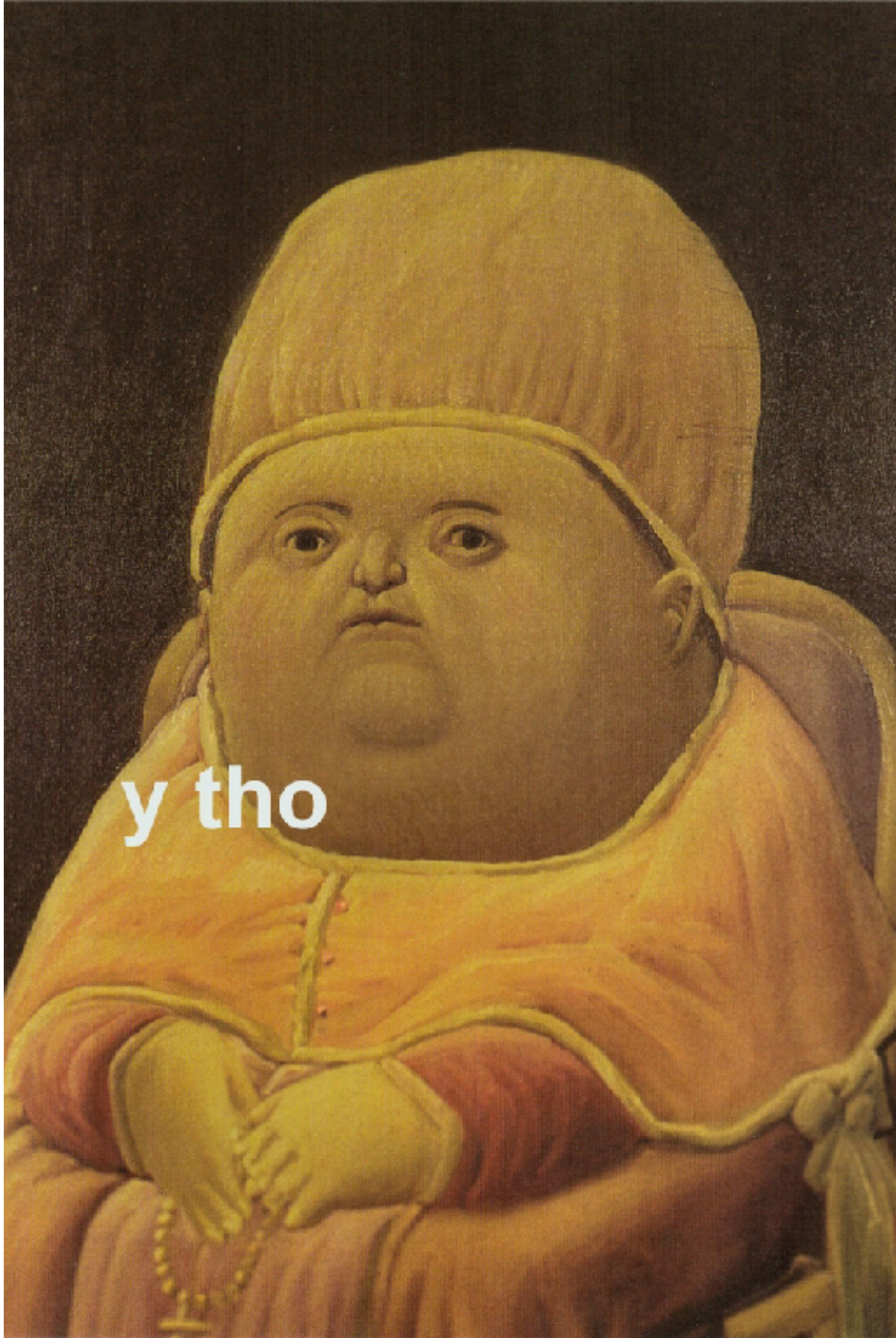
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy
- Audio-video Processors
- Computer Vision and Imaging
- AI (*e.g.*, DeepMind



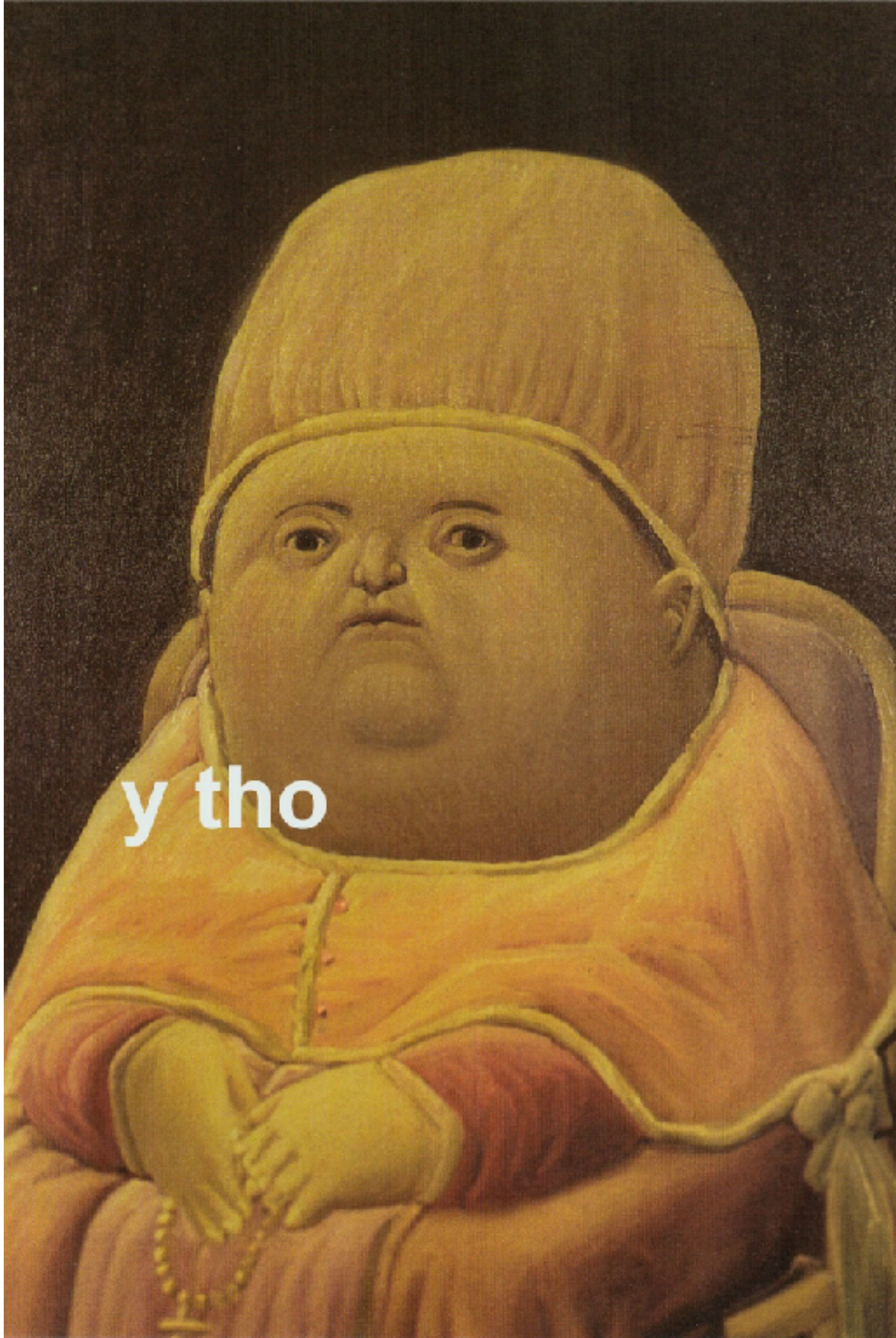
Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy
- Audio-video Processors
- Computer Vision and Imaging
- AI (*e.g.*, DeepMind
- Cryptography and Security



Jobs with CUDA

- Physics and Graphics
- Scientific Communities
- Bioinformatics and Healthcare
- Finance and Economy
- Audio-video Processors
- Computer Vision and Imaging
- AI (*e.g.*, DeepMind
- Cryptography and Security
- (*and the list goes on…*)



Speedup

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*

- Rendering (raytracing): 60×

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*
- Rendering (raytracing): 60×
- AI (classification of neurons in Electron Microscopy): 60×

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*
- Rendering (raytracing): 60×
- AI (classification of neurons in Electron Microscopy): 60×
- Lattice-Boltzmann (cardiac flow): 500×

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*
- Rendering (raytracing): 60×
- AI (classification of neurons in Electron Microscopy): 60×
- Lattice-Boltzmann (cardiac flow): 500×
- Basically, when linear algebra is involved, it's fast

Speedup

- Actually *not everything* is suitable for the GPUs, but where it can be applied, *it's fast*
- Rendering (raytracing): 60×
- AI (classification of neurons in Electron Microscopy): 60×
- Lattice-Boltzmann (cardiac flow): 500×
- Basically, when linear algebra is involved, it's fast
- Just to know, almost everything *is* modeled with linear algebra