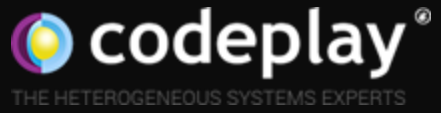


The landscape of Parallel and heterogeneous programming

Gordon Brown & Michael Wong

CppCon 2020 – Sep 2020

- Learning objectives:
  - Learn about the current landscape of Parallel Programming models
  - Learn about the parallel and heterogeneous programming models supporting the broadest platforms

Which one to choose?

OpenACC®
DIRECTIVES FOR ACCELERATORS

OpenMP™

CILK ARTS

SYCL™

C++ AMP
Accelerated Massive Parallelism with Microsoft Visual C++

PPL
stands for
Parallel Patterns Library
Abbreviations.com

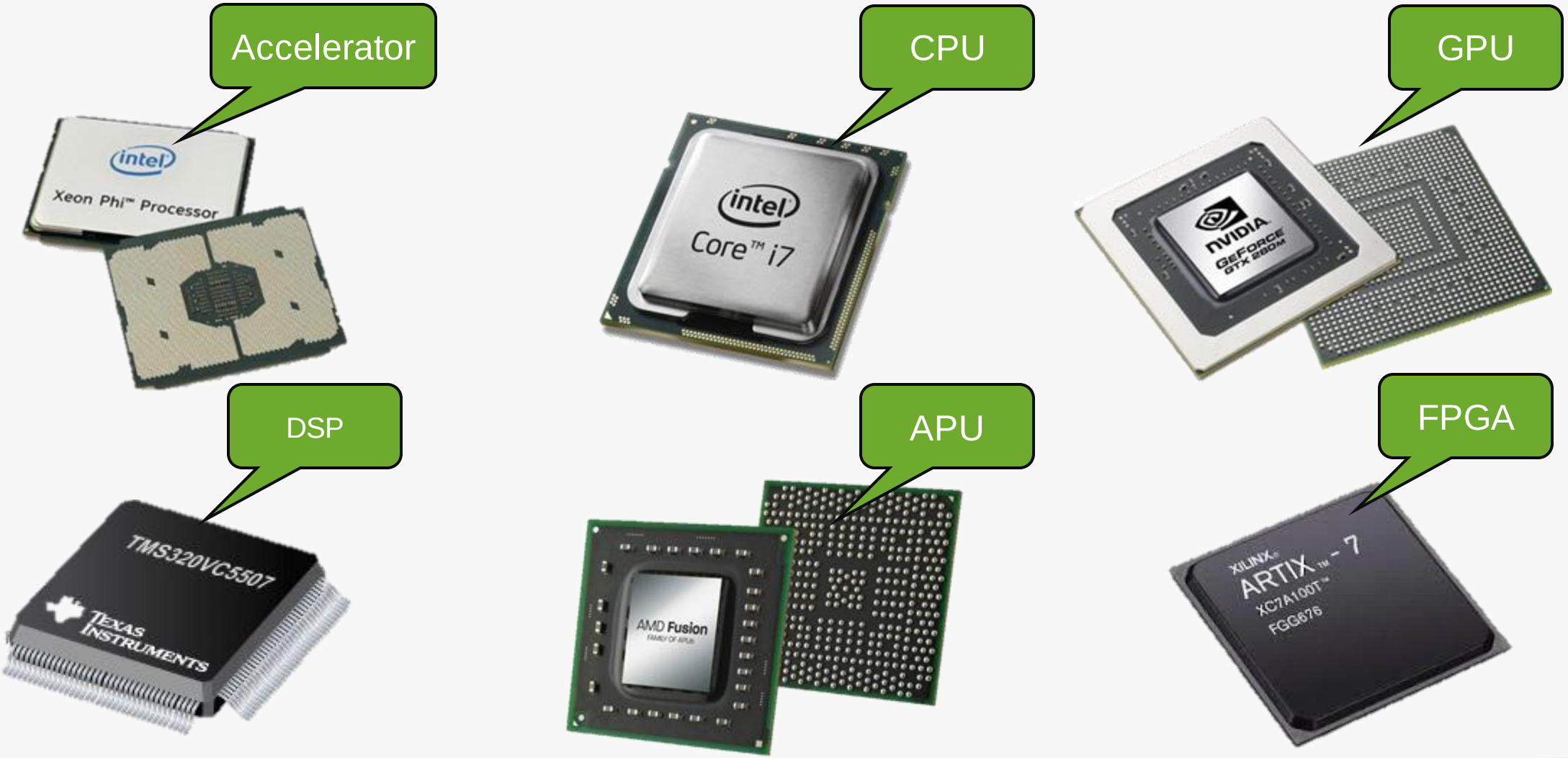
OpenCL

NVIDIA.
CUDA.

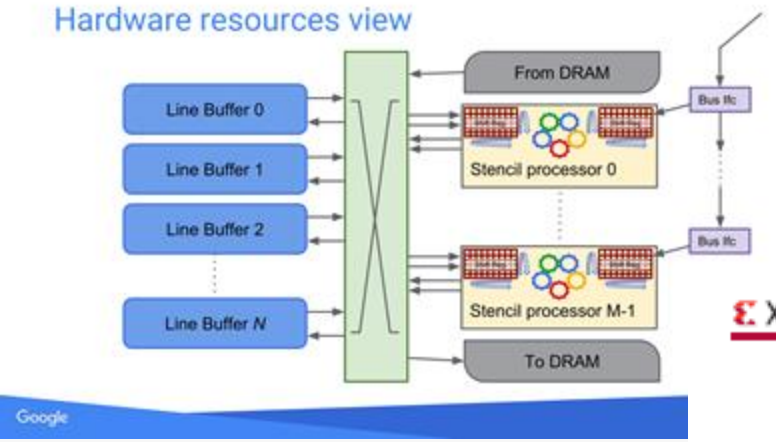
OH, East is East, and West is West,
and never the twain shall meet...

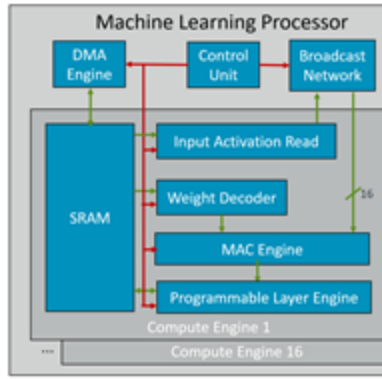
-Rudyard Kipling

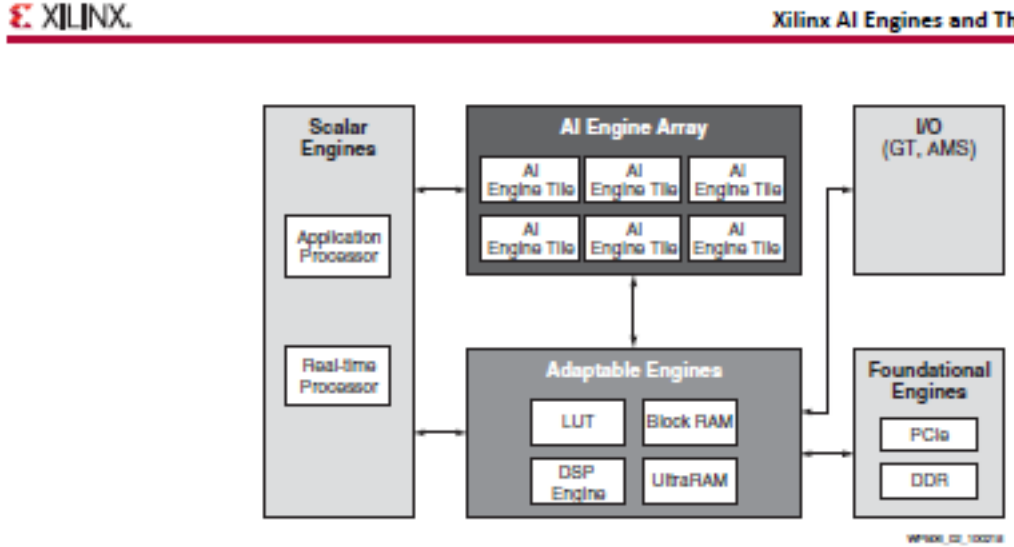
Heterogeneous Devices

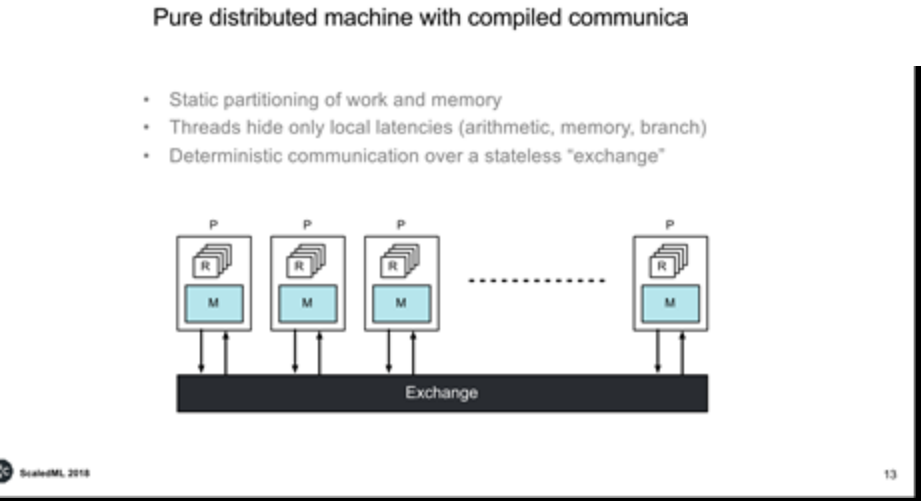


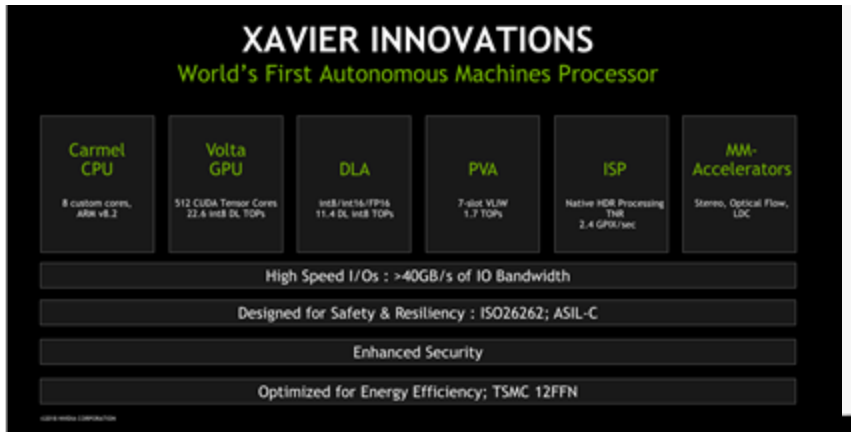
Hot Chips

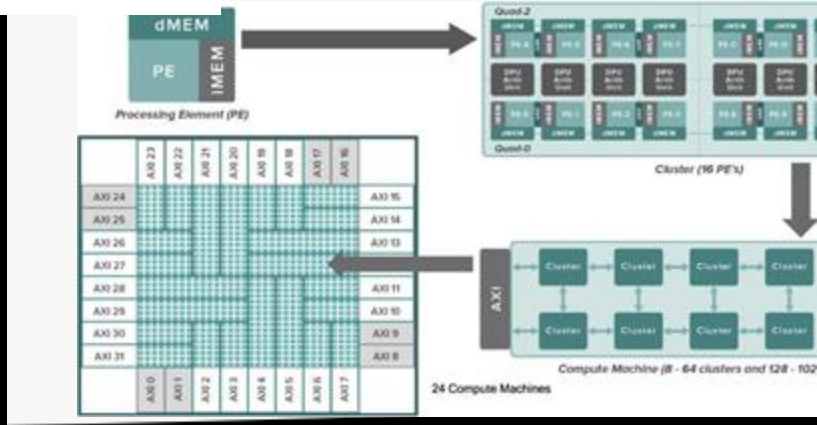






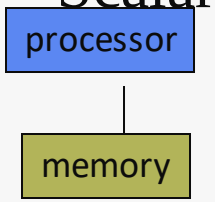




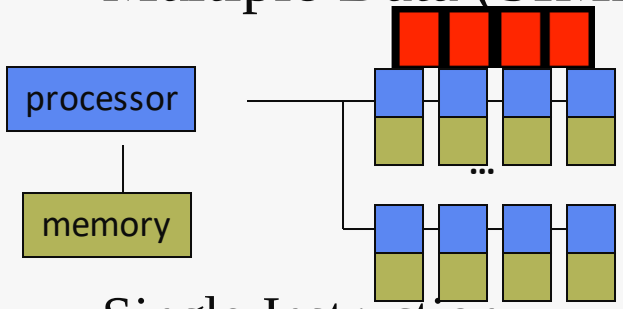


Fundamental Parallel Architecture Types

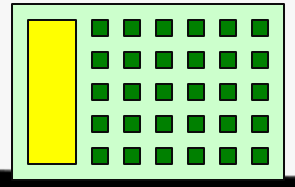
- Uniprocessor
  - Scalar processor



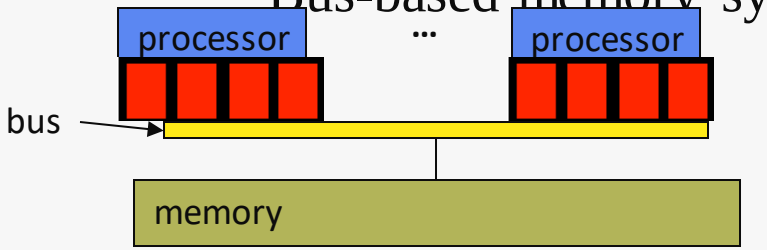
  - Single Instruction Multiple Data (SIMD)



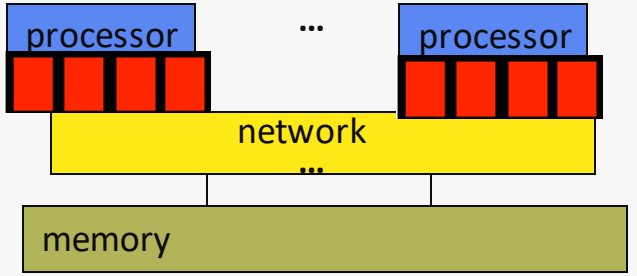
  - Single Instruction Multiple Thread (SIMT)



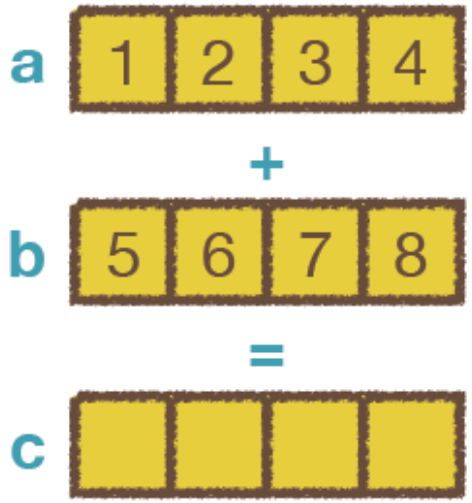
- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

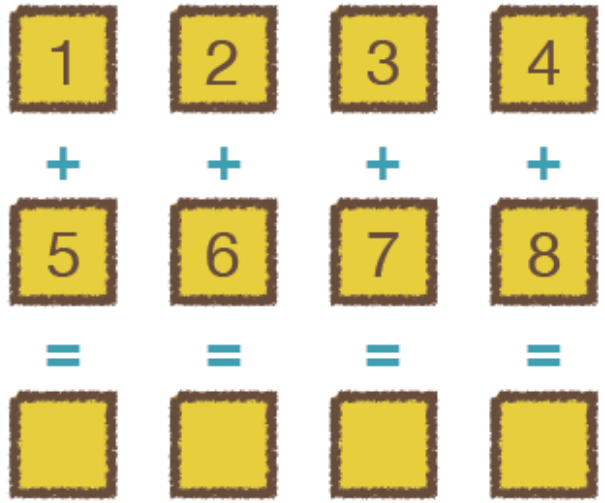


  - Interconnection network



SIMD vs SIMT





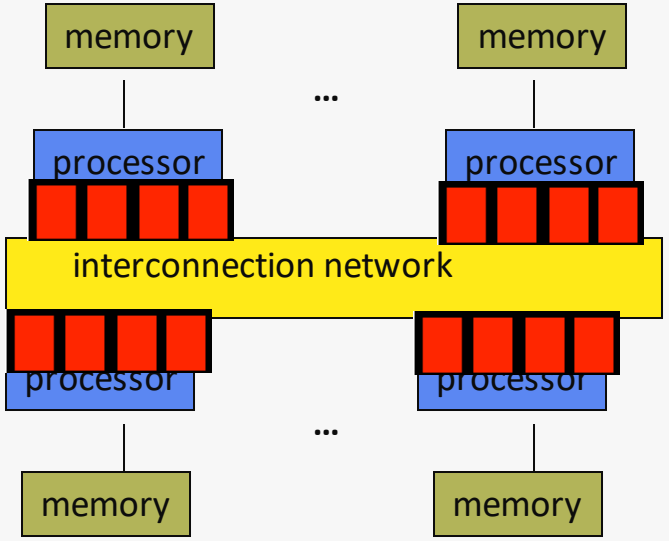

```
__m128 a = _mm_set_ps (4, 3, 2, 1);
__m128 b = _mm_set_ps (8, 7, 6, 5);
__m128 c = _mm_add_ps (a, b);
```

```
float a[4] = {1, 2, 3, 4},
      b[4] = {5, 6, 7, 8}, c[4];

// ...
// Define a compute kernel, which
// a fine-grained thread executes.
{
  int id = ... ;  // my thread ID
  c[id] = a[id] + b[id];
}
```

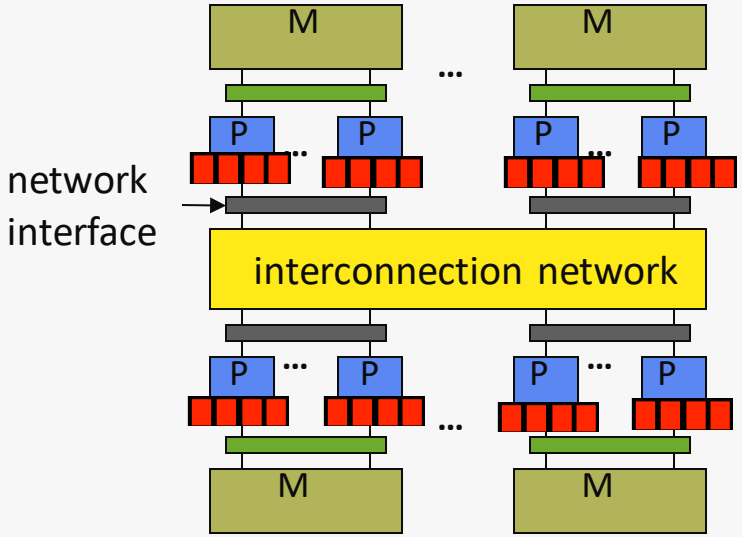
Distributed and network Parallel Architecture Types

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

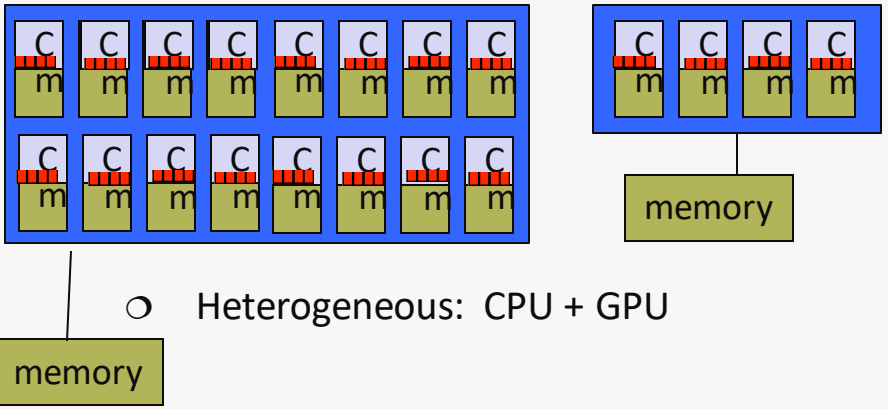
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number

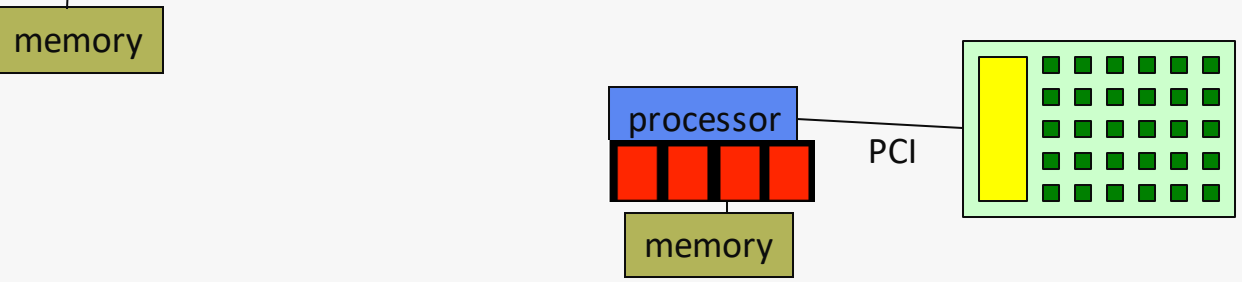
Modern Parallel Architecture

- Multicore Manycore
  - Manycore vs Multicore CPU

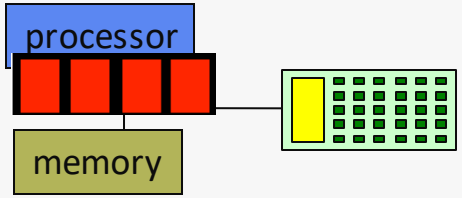


cores can be hardware multithreaded (hyperthread)

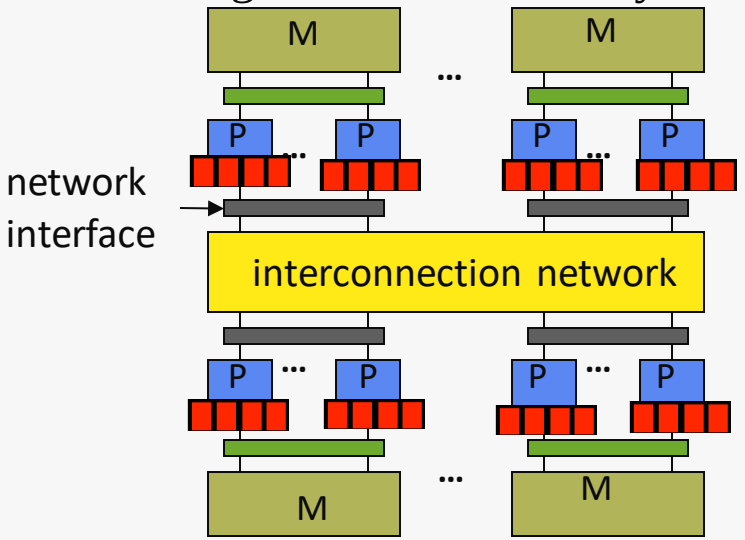
  - Heterogeneous: CPU + GPU



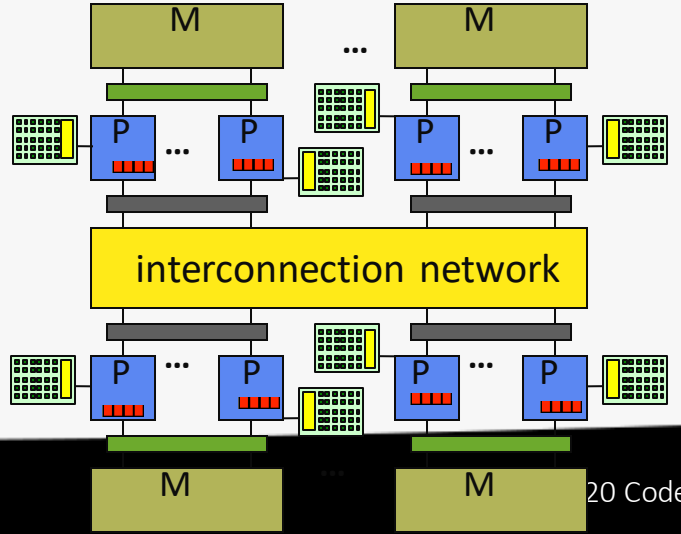
  - Heterogeneous: "Fused" CPU + GPU



- Heterogeneous: CPU+Manycore CPU

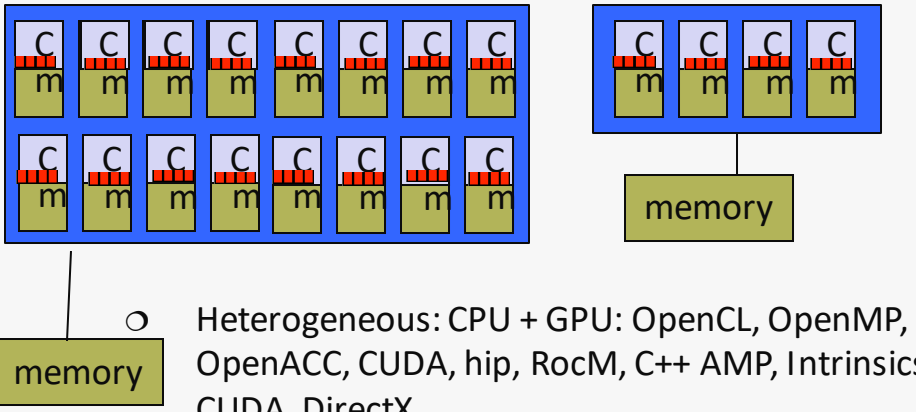


- Heterogeneous: Multicore SMP+GPU Cluster



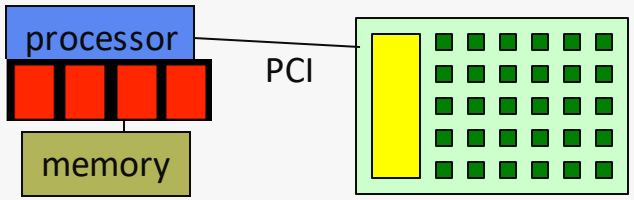
Modern Parallel Programming model

- Multicore Manycore
  - Manycore vs Multicore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread

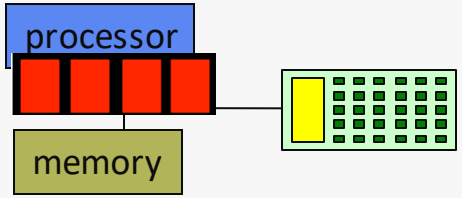


cores can be hardware multithreaded (hyperthread)

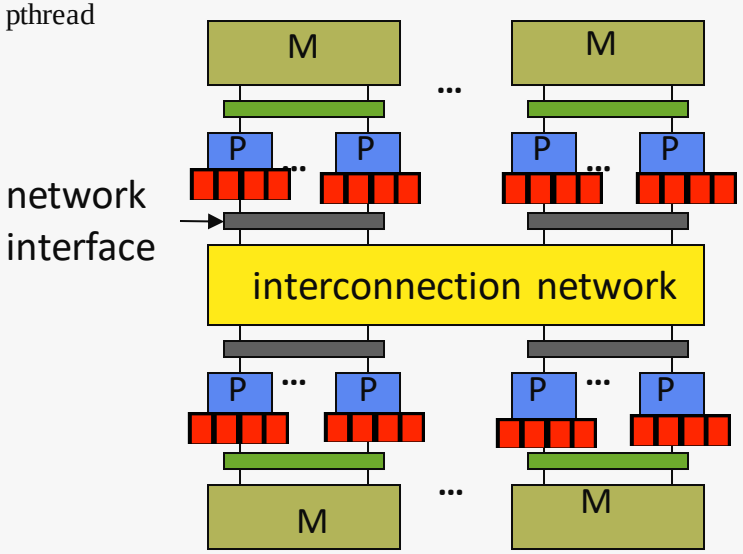
  - Heterogeneous: CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, OpenACC, CUDA, hip, RocM, C++ AMP, Intrinsics, OpenGL, Vulkan, CUDA, DirectX



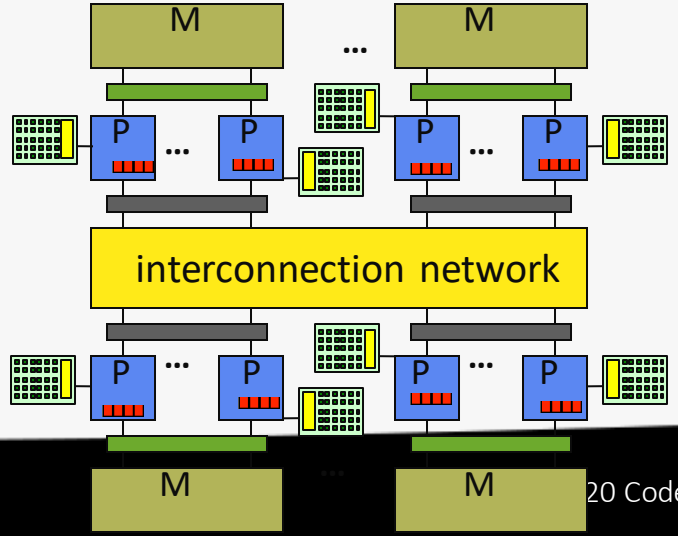
  - Heterogeneous: "Fused" CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, hip, RocM, Intrinsics, OpenGL, Vulkan, DirectX



- Heterogeneous: CPU+Manycore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread

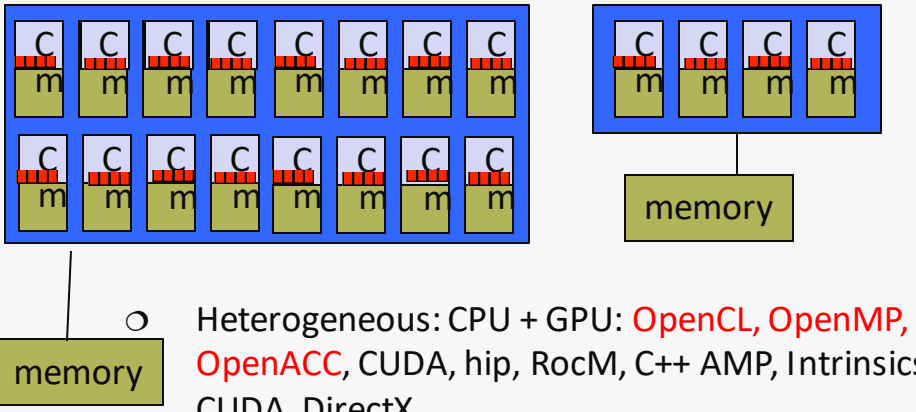


- Heterogeneous: Multicore SMP+GPU Cluster: OpenCL, OpenMP, SYCL, C++17/20



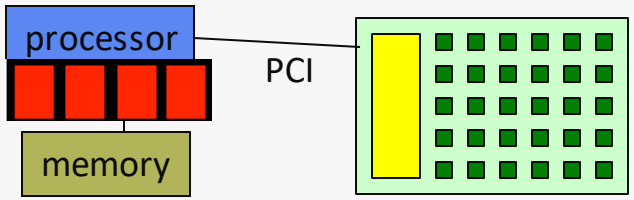
Which Programming model works on all the Architectures?Is there a pattern?

- Multicore Manycore
  - Manycore vs Multicore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread

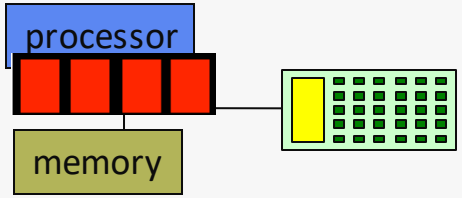


cores can be hardware multithreaded (hyperthread)

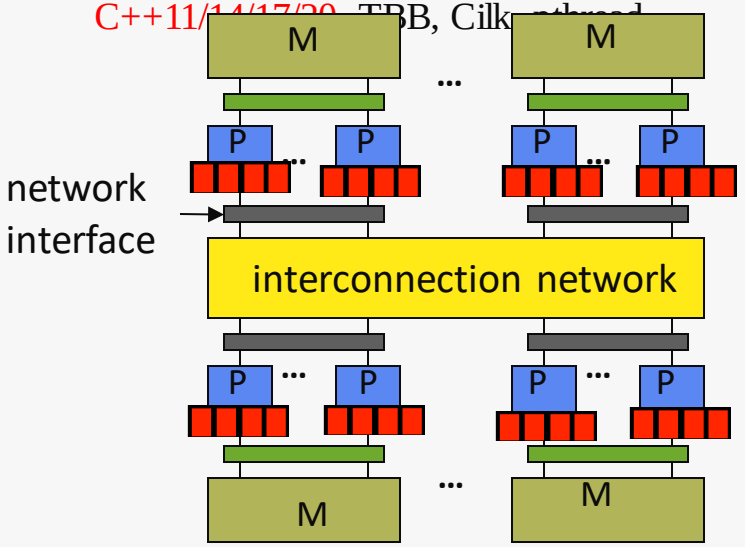
  - Heterogeneous: CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, OpenACC, CUDA, hip, RocM, C++ AMP, Intrinsics, OpenGL, Vulkan, CUDA, DirectX



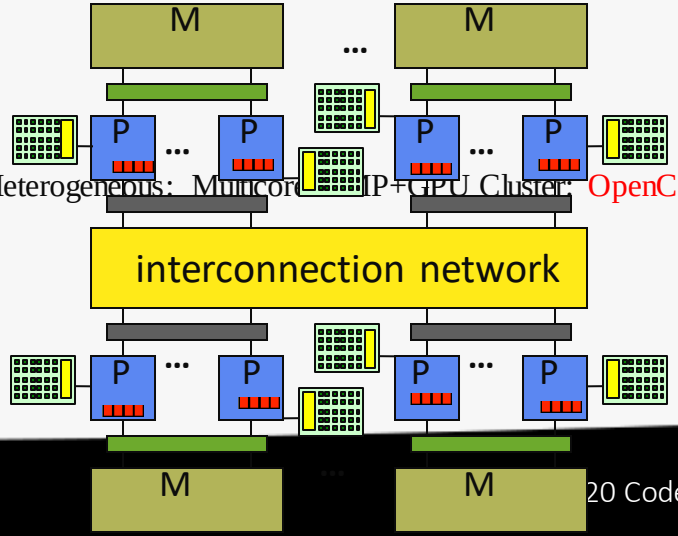
  - Heterogeneous: "Fused" CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, hip, RocM, Intrinsics, OpenGL, Vulkan, DirectX



- Heterogeneous: CPU+Manycore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



- Heterogeneous: Multicore CPU+GPU Cluster: OpenCL, OpenMP, SYCL, C++17/20



To support all the different parallel architectures

- With a single code base
- And if you also want it to be an International Open Specification
- And if you want it to be growing with the architectures
- Most are single source and one is separate source
- Non-pragma based that fully supports heterogeneous C++: SYCL

- You really only have a few choices

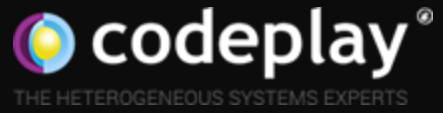
Key takeaways

Programming models that support the broadest architectures and is Open and Non-proprietary

Mostly Single Source : SYCL, C++, OpenMP

Or Separate source: OpenCL

But it you want non-pragma-based that fully supports heterogeneous C++: SYCL



Importance of Parallelism and heterogeneity

Gordon Brown & Michael Wong

CppCon 2020 – Sep 2020

- Learning objectives:
  - Learn about the current landscape of computer architectures
  - Learn about the performance benefits of parallelism
  - Learn about typical CPU and GPU architectures
  - Learn about the key differences between the CPU and GPU

Imagine you need to dig a hole...

But you want to get it done faster...

Let’s say you have three options:

But you want to get it done faster...

Let’s say you have three options:

- Simply dig faster with the one shovel you have

But you want to get it done faster...

Let’s say you have three options:

- Simply dig faster with the one shovel you have
- Buy a better shovel that moves more dirt

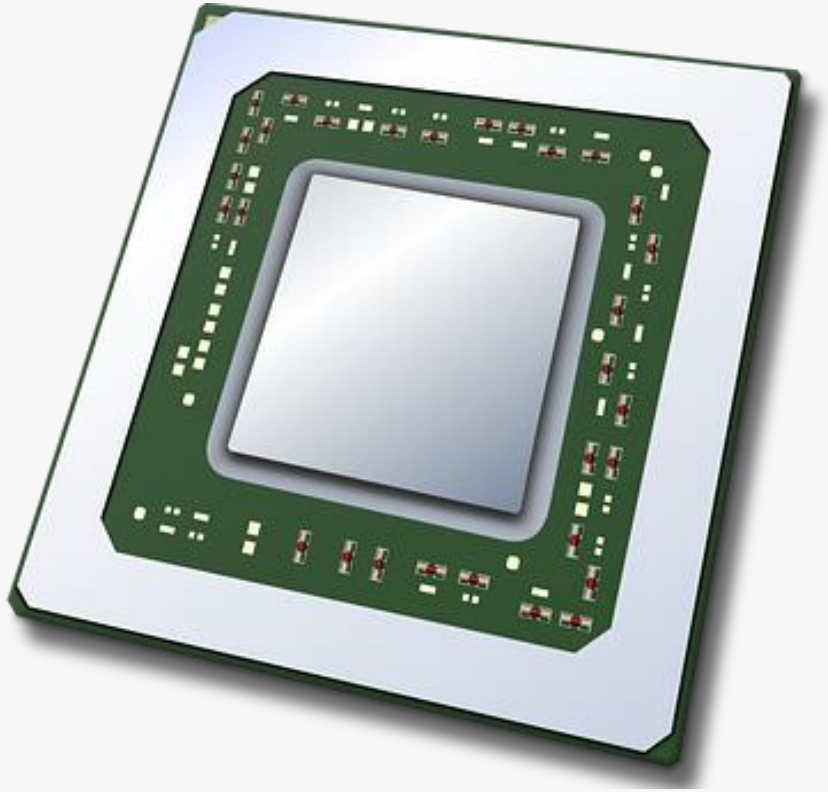
But you want to get it done faster...

Let’s say you have three options:

- Simply dig faster with the one shovel you have
- Buy a better shovel that moves more dirt
- Hire additional people to help you dig

This is analogous to processor design

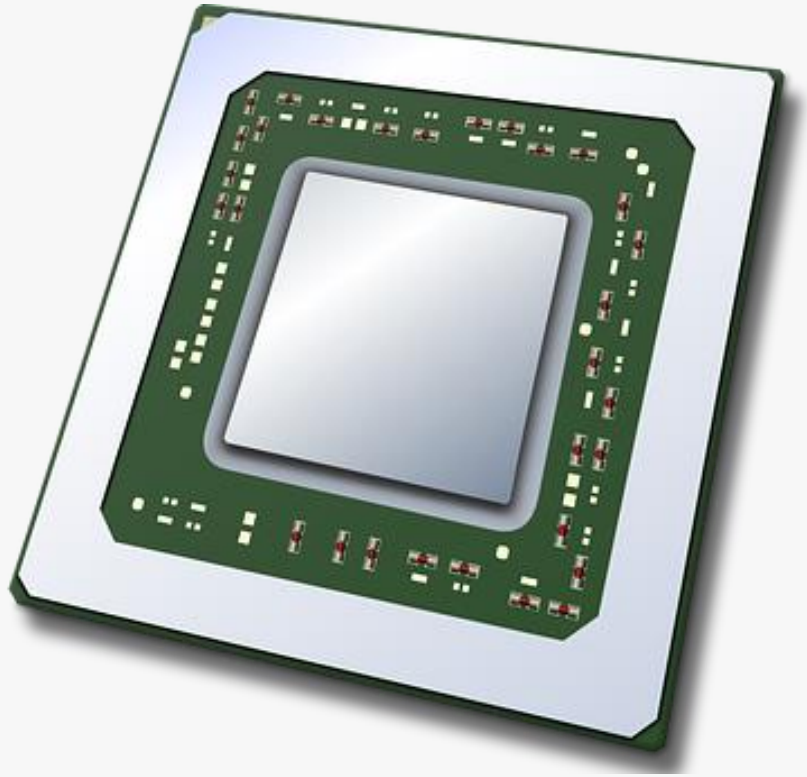
=



When you need a processor to run faster...

You have three options:

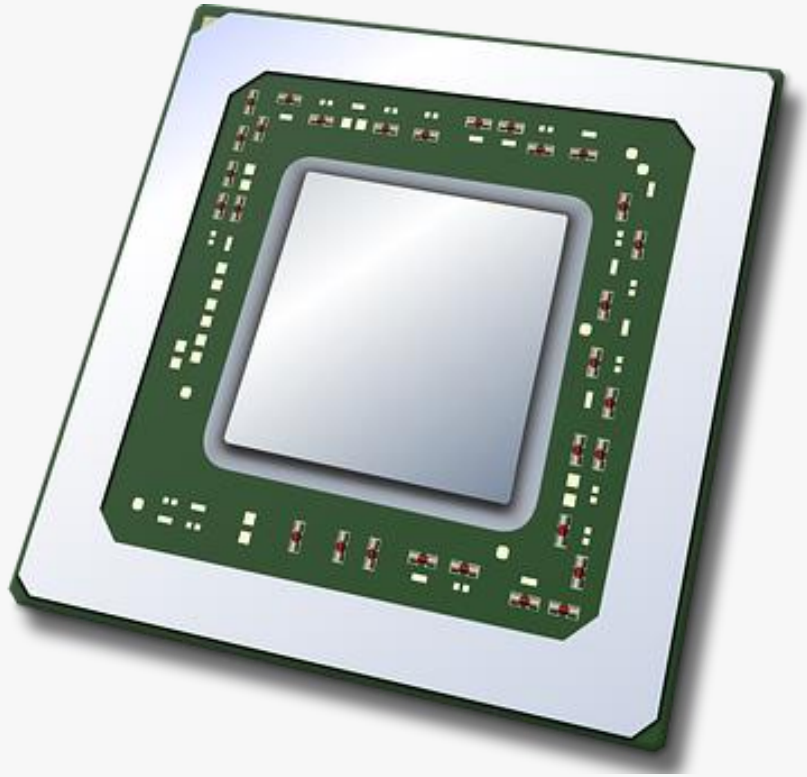
- Run at a higher clock speed



When you need a processor to run faster...

You have three options:

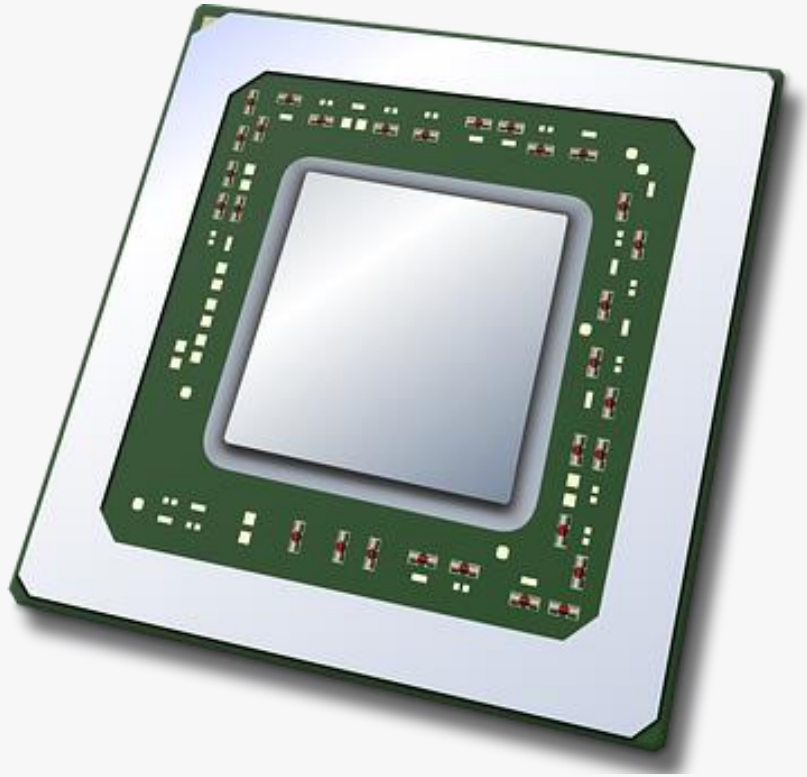
- Run at a higher clock speed
- Do more work on each clock cycle



When you need a processor to run faster...

You have three options:

- Run at a higher clock speed
- Do more work on each clock cycle
- Have multiple processors work in parallel

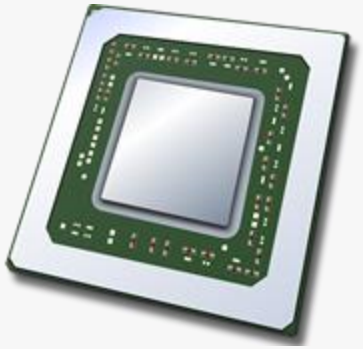


When you need a processor to run faster...

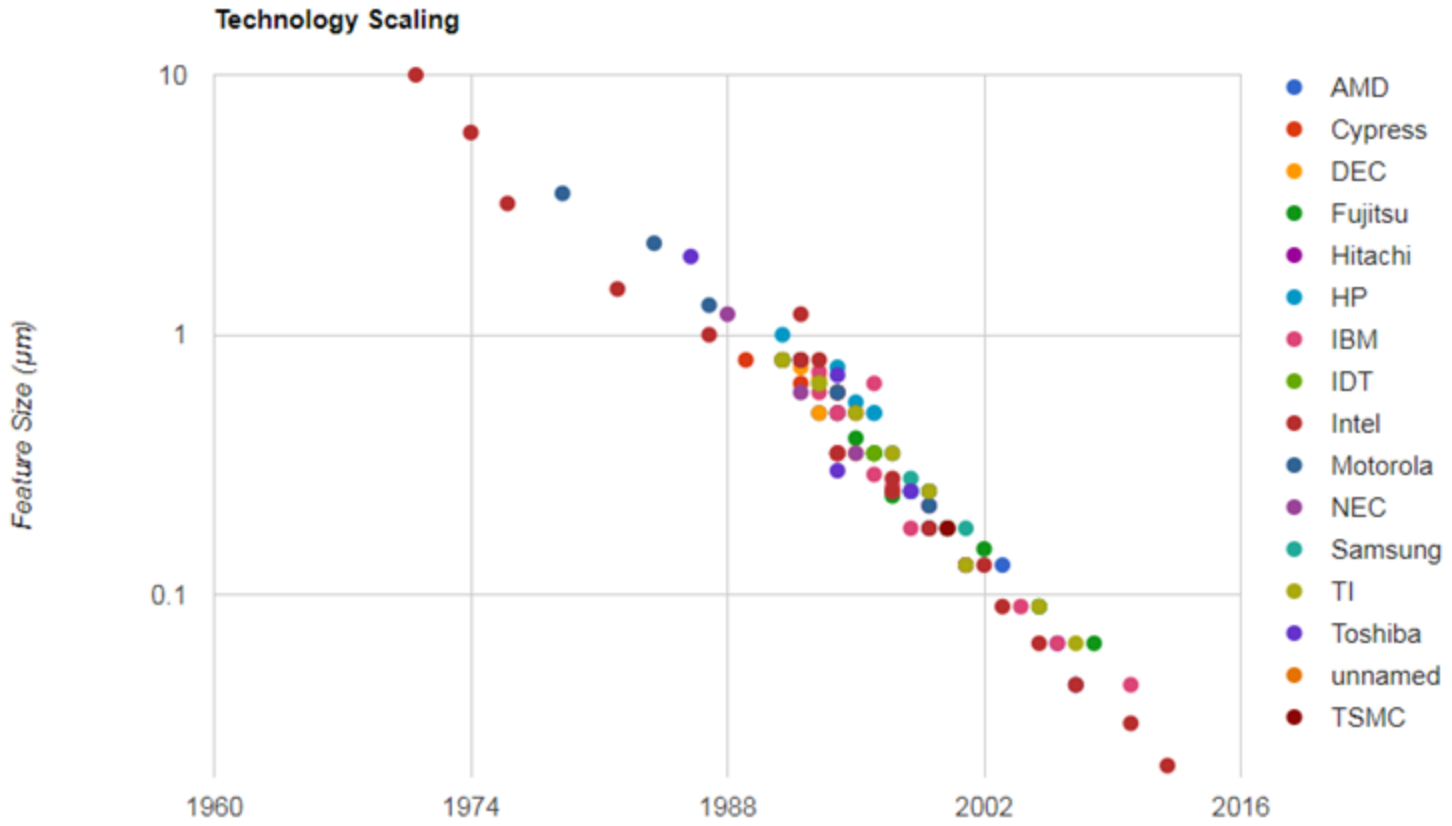
You have three options:

- Run at a higher clock speed
- Do more work on each clock cycle
- Have multiple processors work in parallel

Increasing the clock speed of modern processors increases power consumption

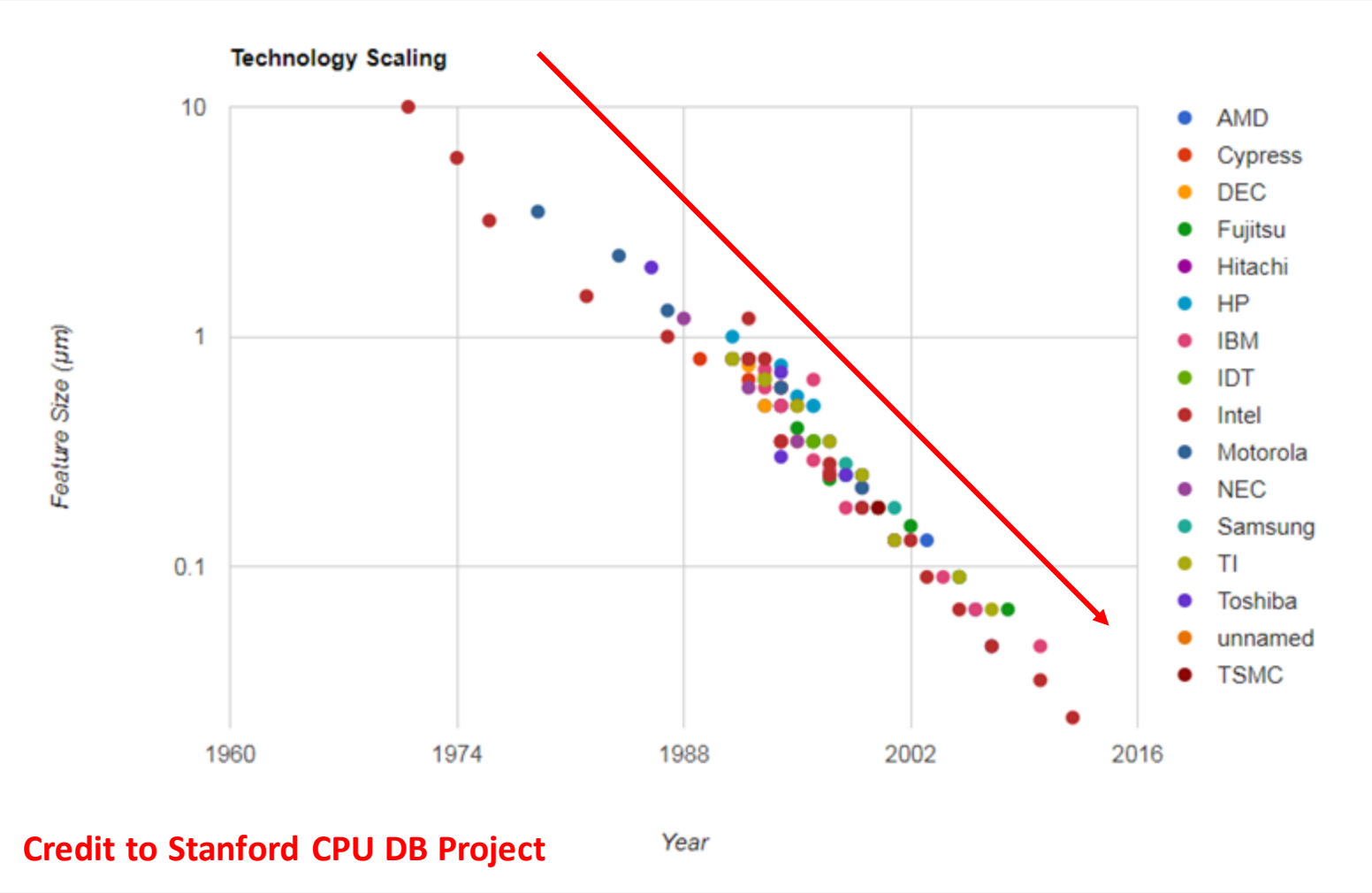


The problem...



Credit to Stanford CPU DB Project

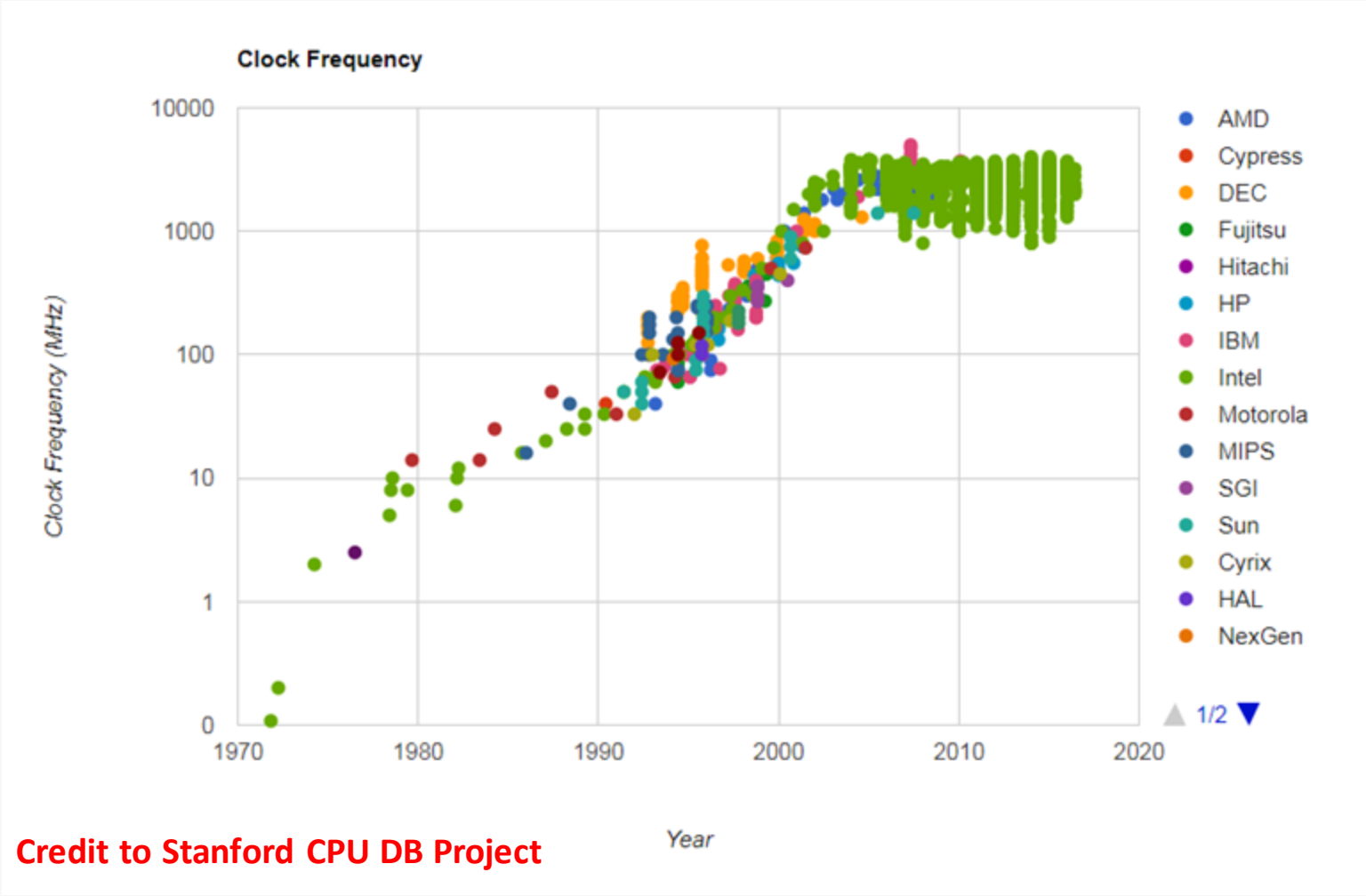
The problem...



Credit to Stanford CPU DB Project

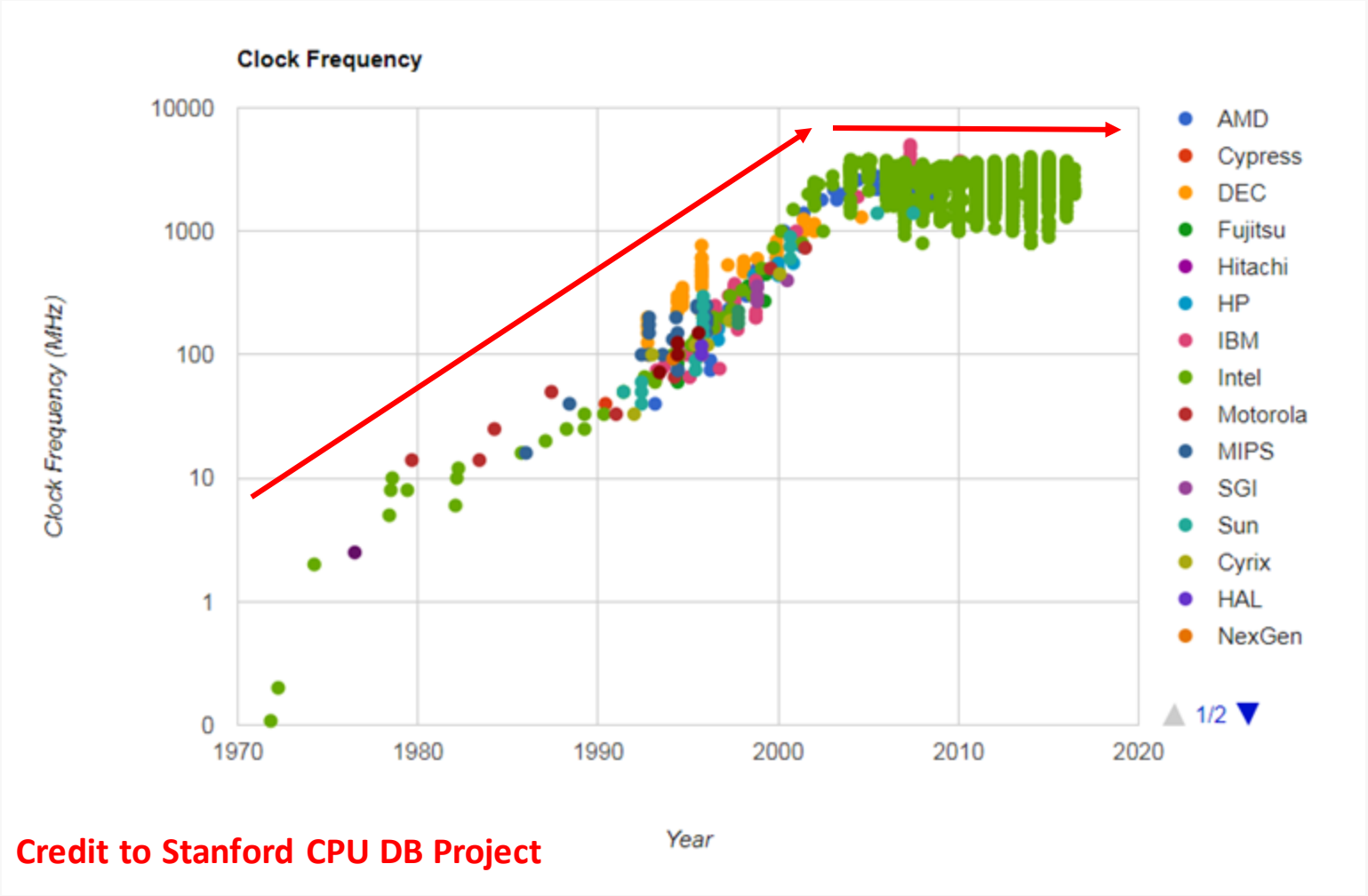
The size of transistors are continuing to decrease

The problem...



Credit to Stanford CPU DB Project

The problem...



Credit to Stanford CPU DB Project

However in recent years CPU clock speeds have stopped increasing

CPU clock speeds have stopped increasing despite transistor sizes continuing to decrease, why is this?

The problem is power

- Fitting more transistors on a single processor
  - Requires more power
  - Which generates more heat

- Cooling these devices becomes the limiting factor in processor design
  - Applies to everything from HPC to mobile devices

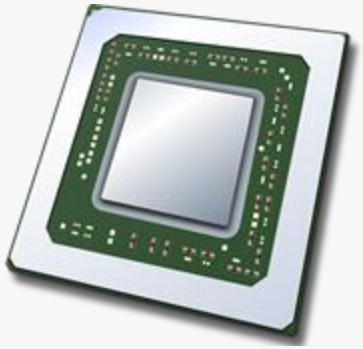
When you need a processor to run faster...

You have three options:

- Run at a higher clock speed
- Do more work on each clock cycle
- Have multiple processors work in parallel

Making use of more powerful instructions can provide a performance gain

However there's a limit to how much instruction level parallelism can be achieved



The problem…

There are ways to more efficiently utilize the clock cycles of a processor:

- Efficient scheduling of instructions
- Out-of-order execution of instructions
- Instruction-level parallelism
- SIMD instructions

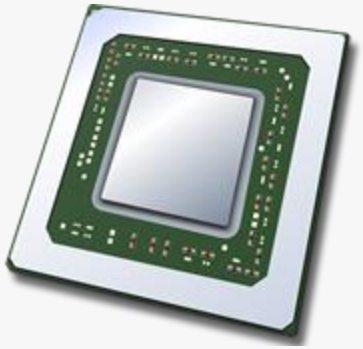
These are all important, however there is a limit to how much performance gain can be achieved

When you need a processor to run faster...

You have three options:

- Run at a higher clock speed
- Do more work on each clock cycle
- Have multiple processors work in parallel

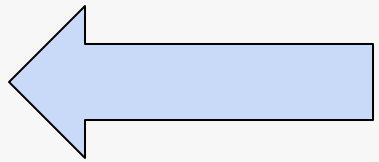
Having a large number of smaller processors execute in parallel can provide additional performance gain

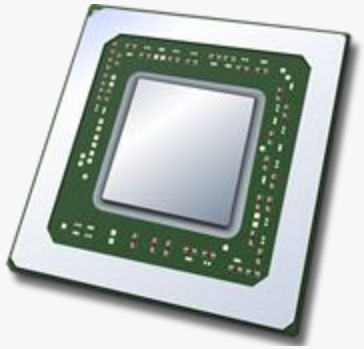


When you need a processor to run faster...

You have three options:

- Run at a higher clock speed
- Do more work on each clock cycle
- Have multiple processors work in parallel

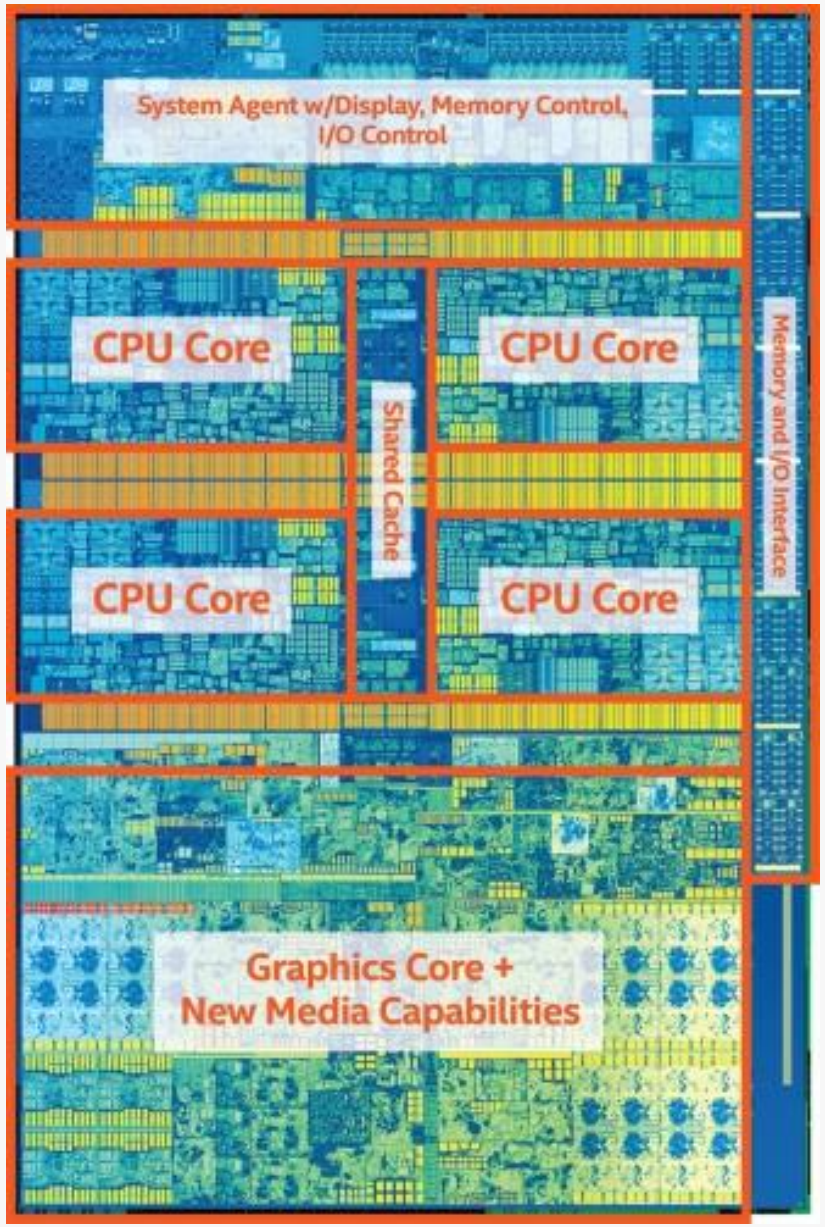




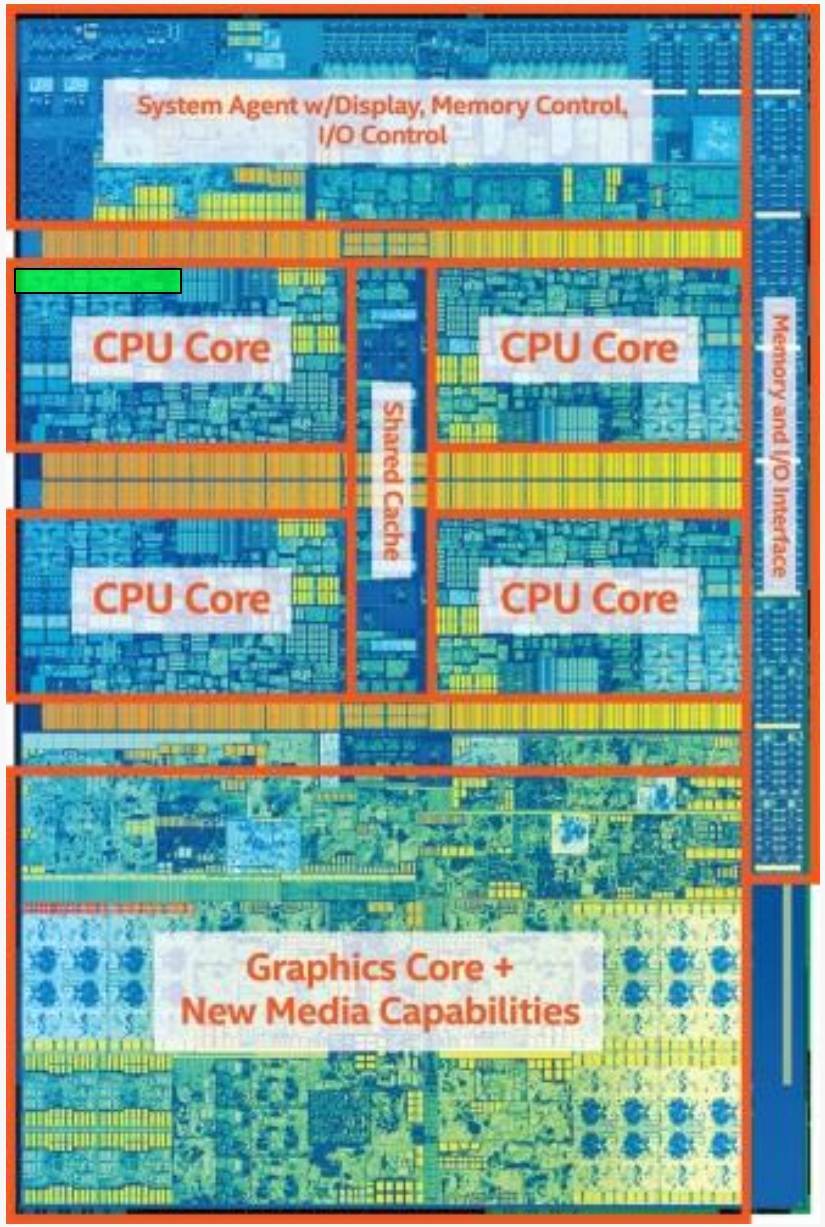
Take a typical Intel chip…

Intel Core i7 7th Gen

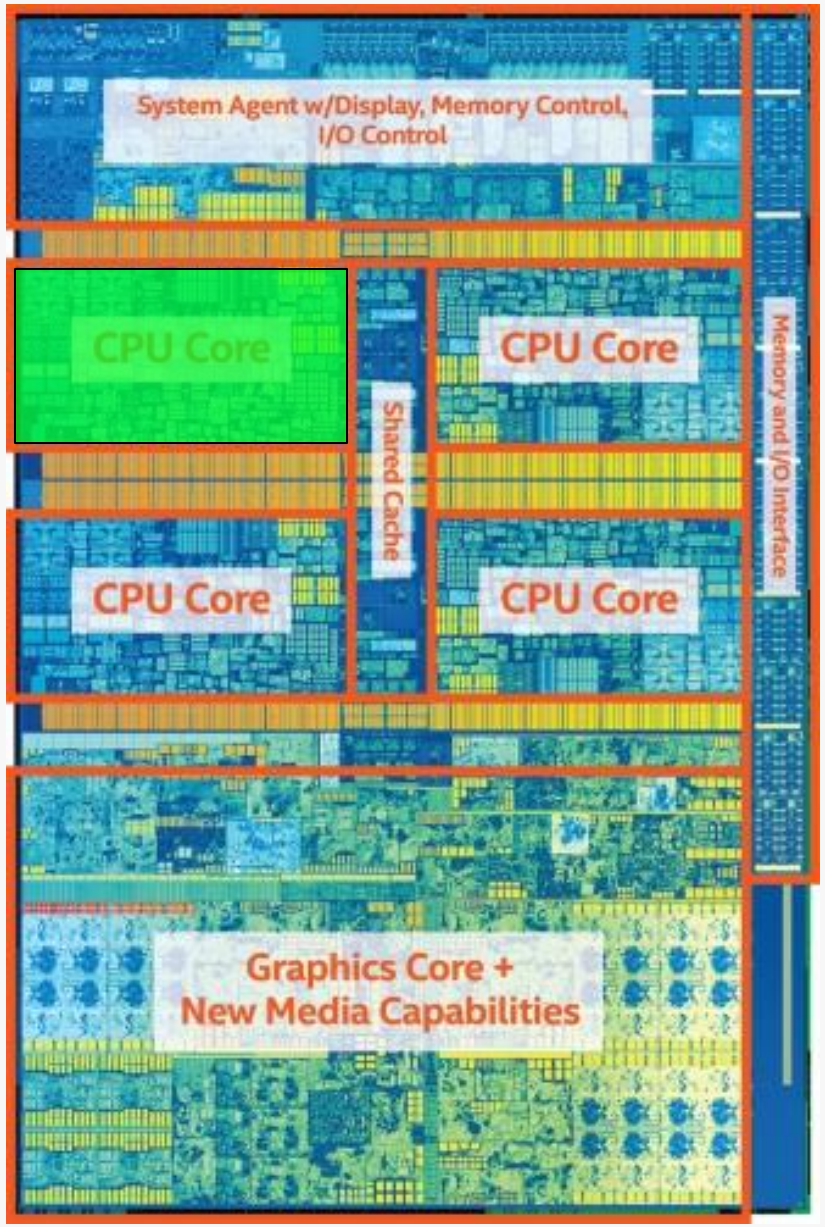
- 4x CPU cores
  - Each with hyperthreading
  - Each with support for 256bit AVX2 instructions
- Intel Gen 9.5 GPU
  - With 1280 processing elements



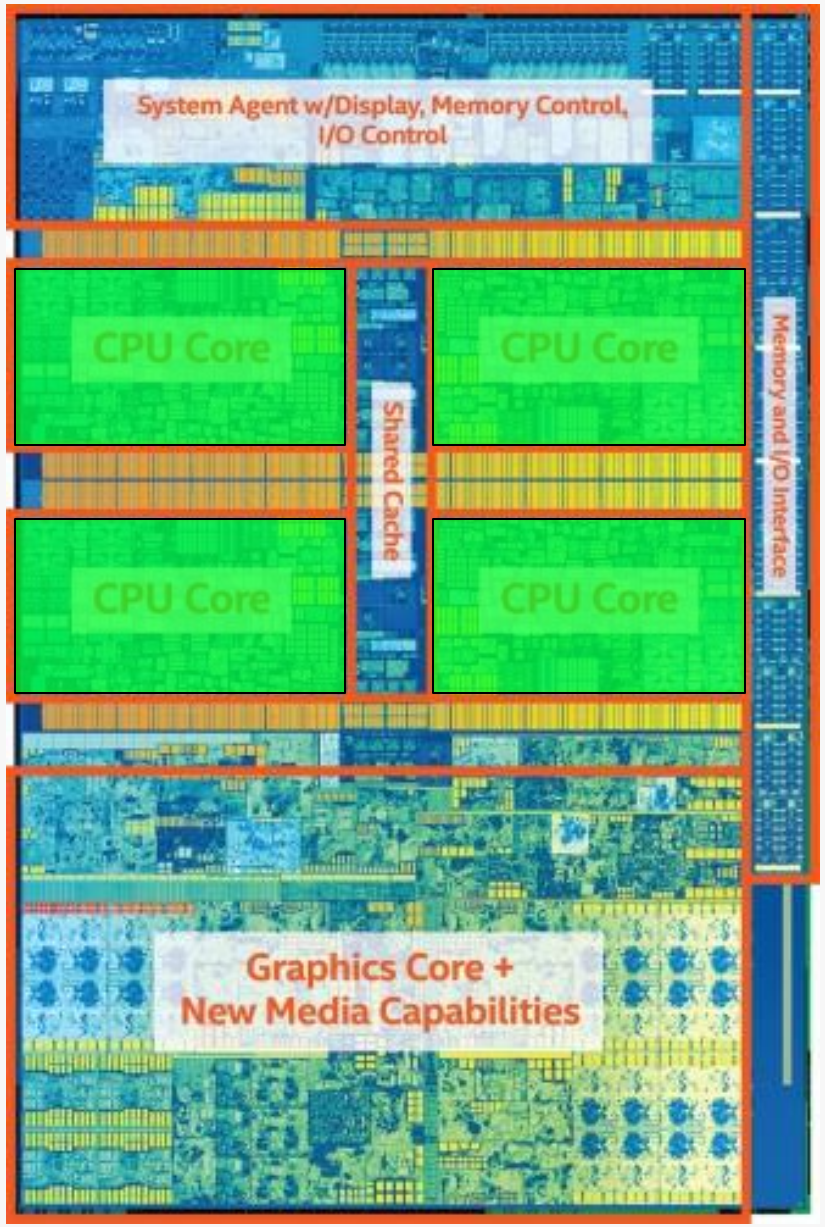
Regular sequential C++ code (non-vectorised) running on a single thread only takes advantage of a very small amount of the available resources of the chip



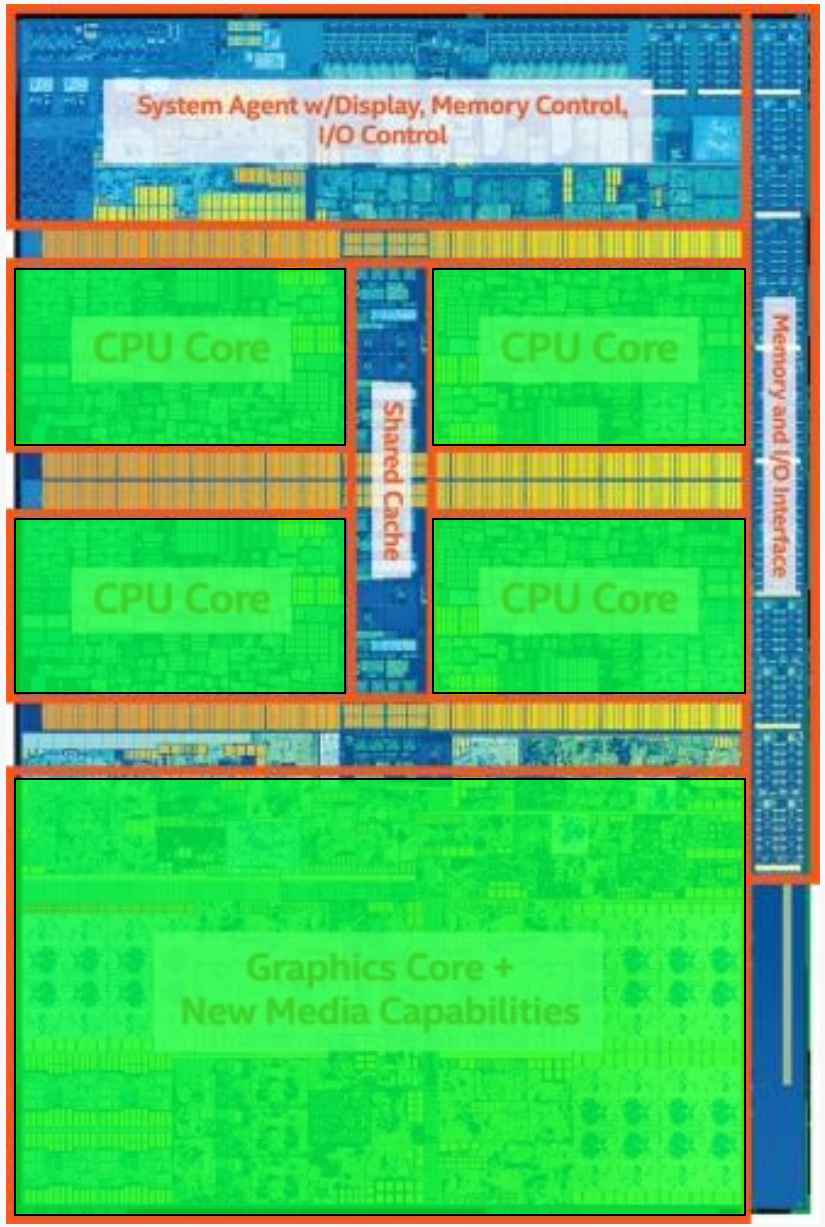
Vectorisation allows you to fully utilise a single CPU core



Multi-threading allows you to fully utilise all CPU cores



Heterogeneous dispatch allows you to fully utilise the entire chip



So now you have multiple diggers...

How do you manage them?

So now you have multiple diggers...

How do you manage them?

- Make sure they all have work to do

So now you have multiple diggers...

How do you manage them?

- Make sure they all have work to do
- Make sure they are all working efficiently

So now you have multiple diggers...

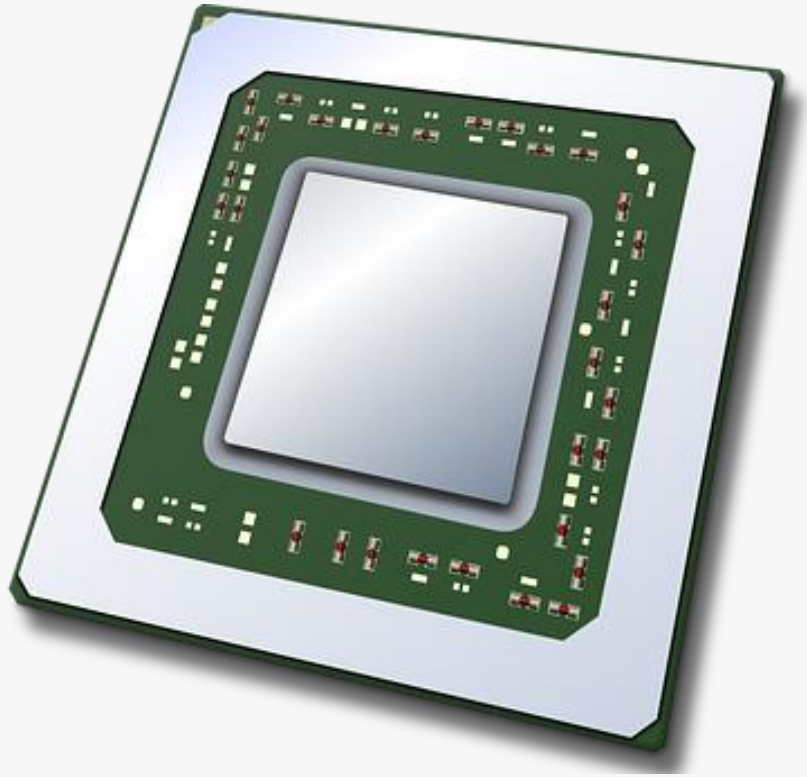
How do you manage them?

- Make sure they all have work to do
- Make sure they are all working efficiently
- Make sure they don't get in each other's way

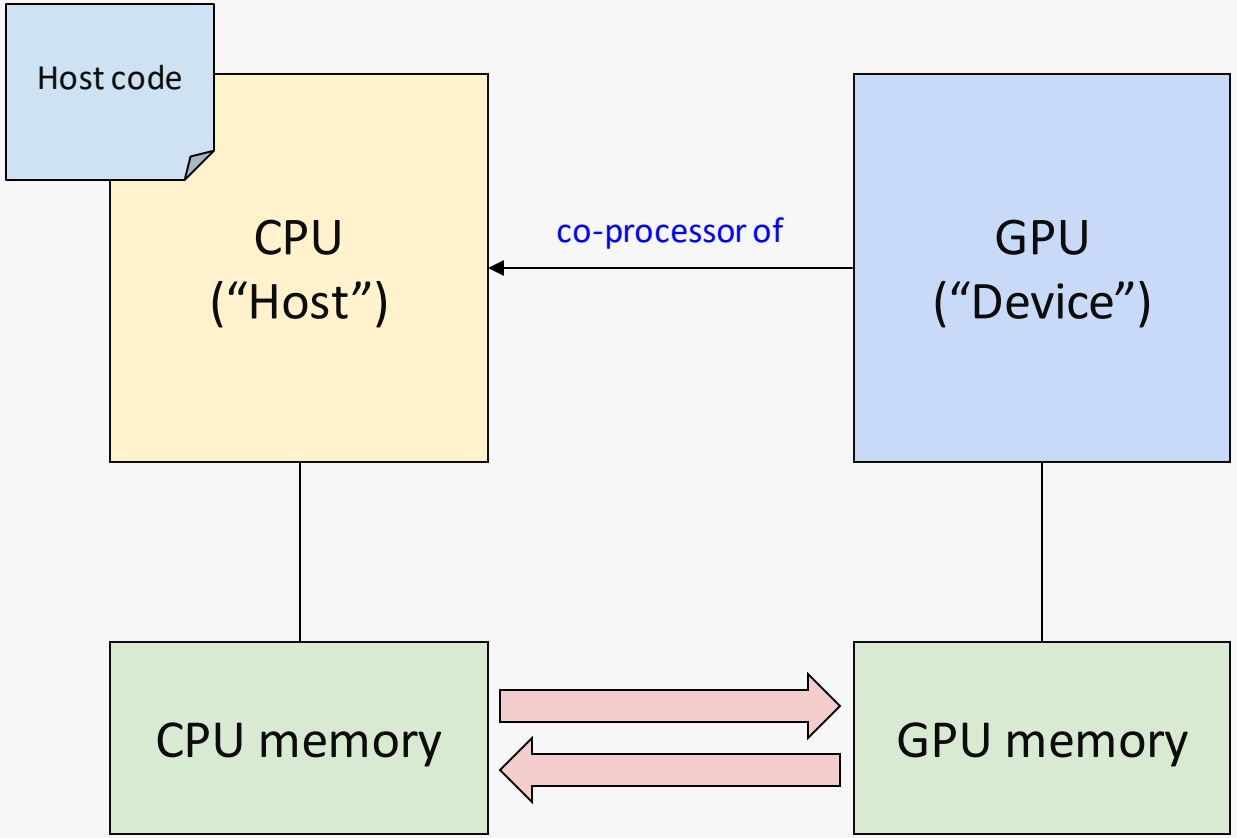
This also applies to parallel processors

How do you manage them?

- Make sure they all have work to do
- Make sure they are all working efficiently
- Make sure they don’t get in each other’s way



Programming on heterogeneous systems

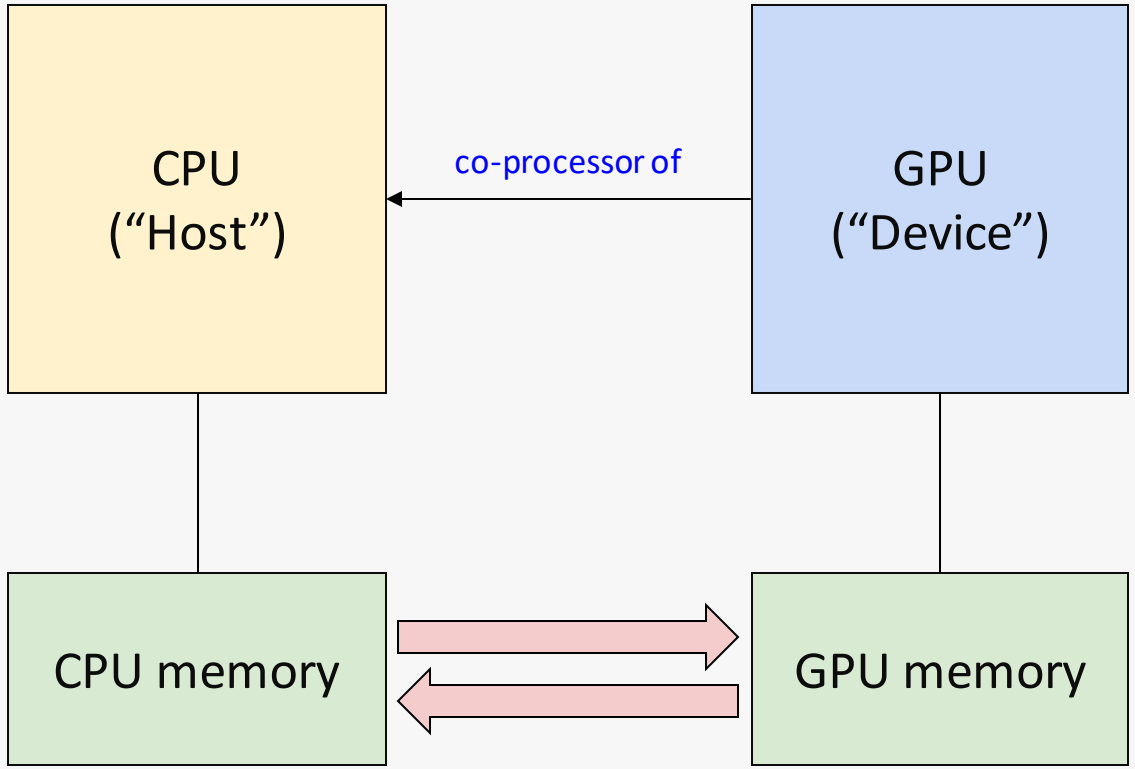


CPU vs GPU

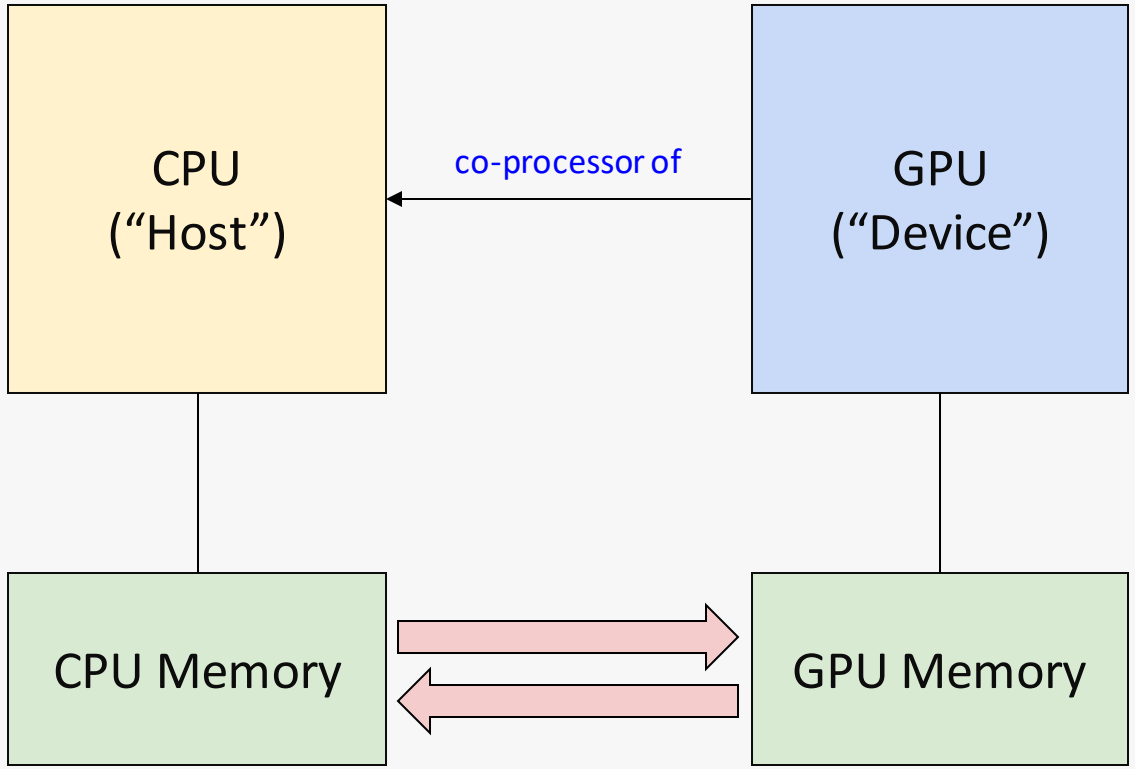
- Small number of large processors
- Flat memory model
- More control structures and less processing units
  - Can do more complex logic
  - Requires more power
- Optimize for latency
  - Minimising the time taken for one particular task
- Flexible programming model

- Large number of small processors
- Hierarchical memory model
- Less control structures and more processing units
  - Can do less complex logic
  - Lower power consumption
- Optimized for throughput
  - Maximising the amount of work done per unit of time
- Restricted programming model

Programming on heterogeneous systems

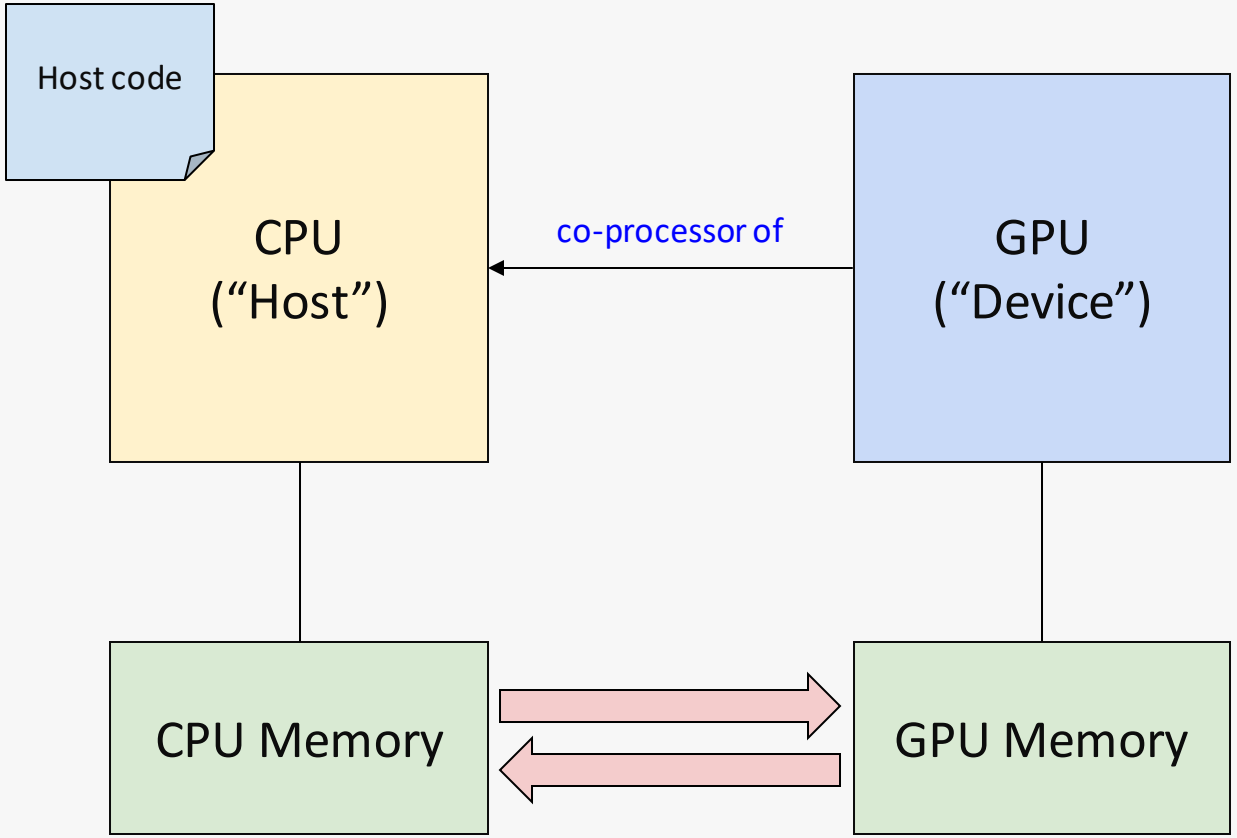


Programming on heterogeneous systems

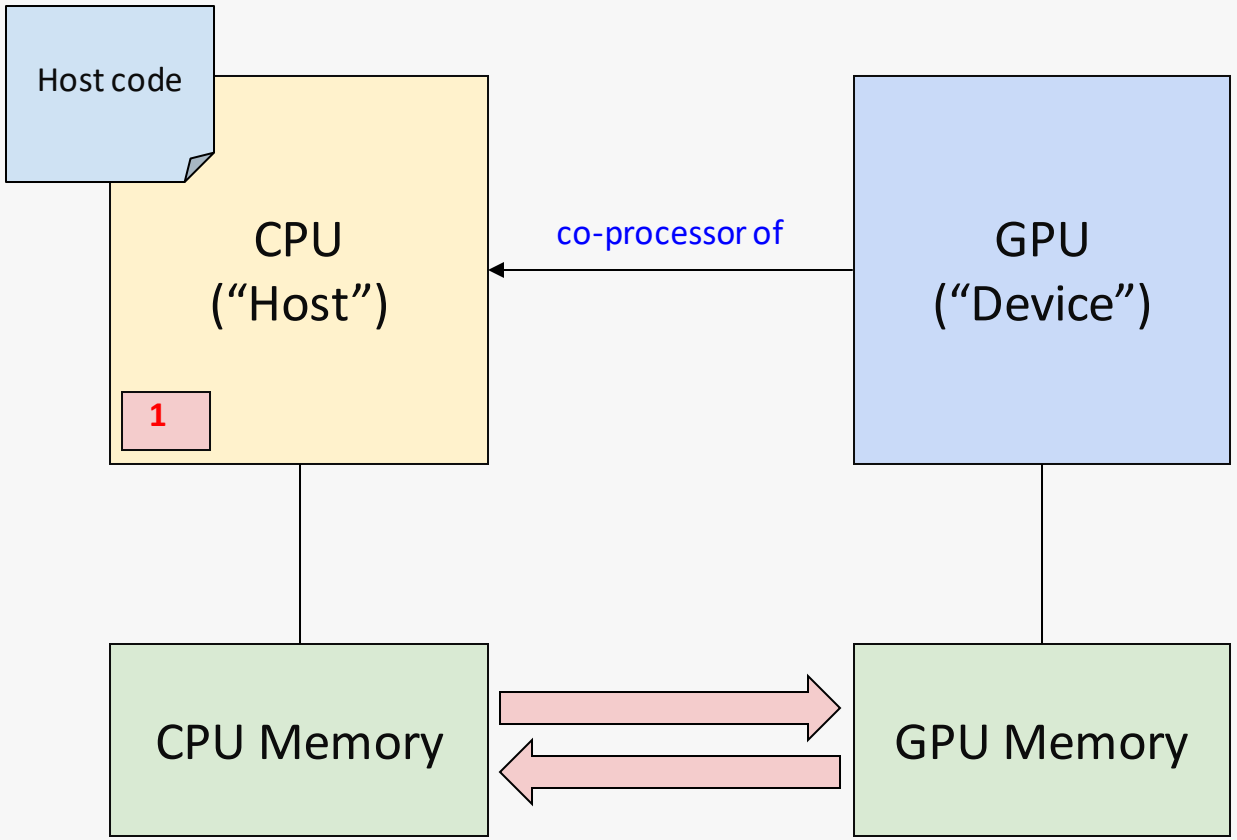


Explicit heterogeneous programming models can allow you to write SPMD code that will run on SIMD CPUs and GPUs

Programming on a SIMD CPU

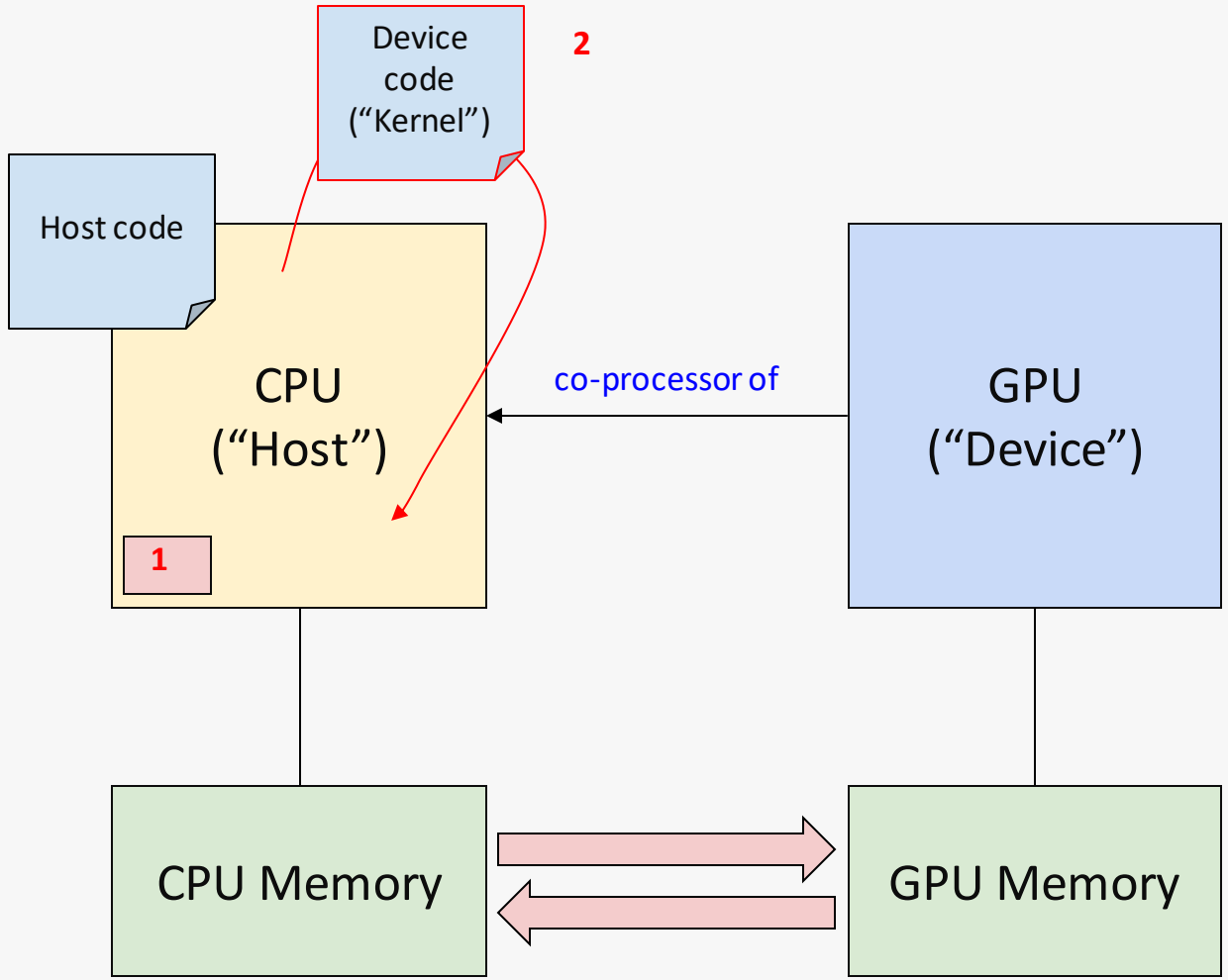


Programming on a SIMD CPU



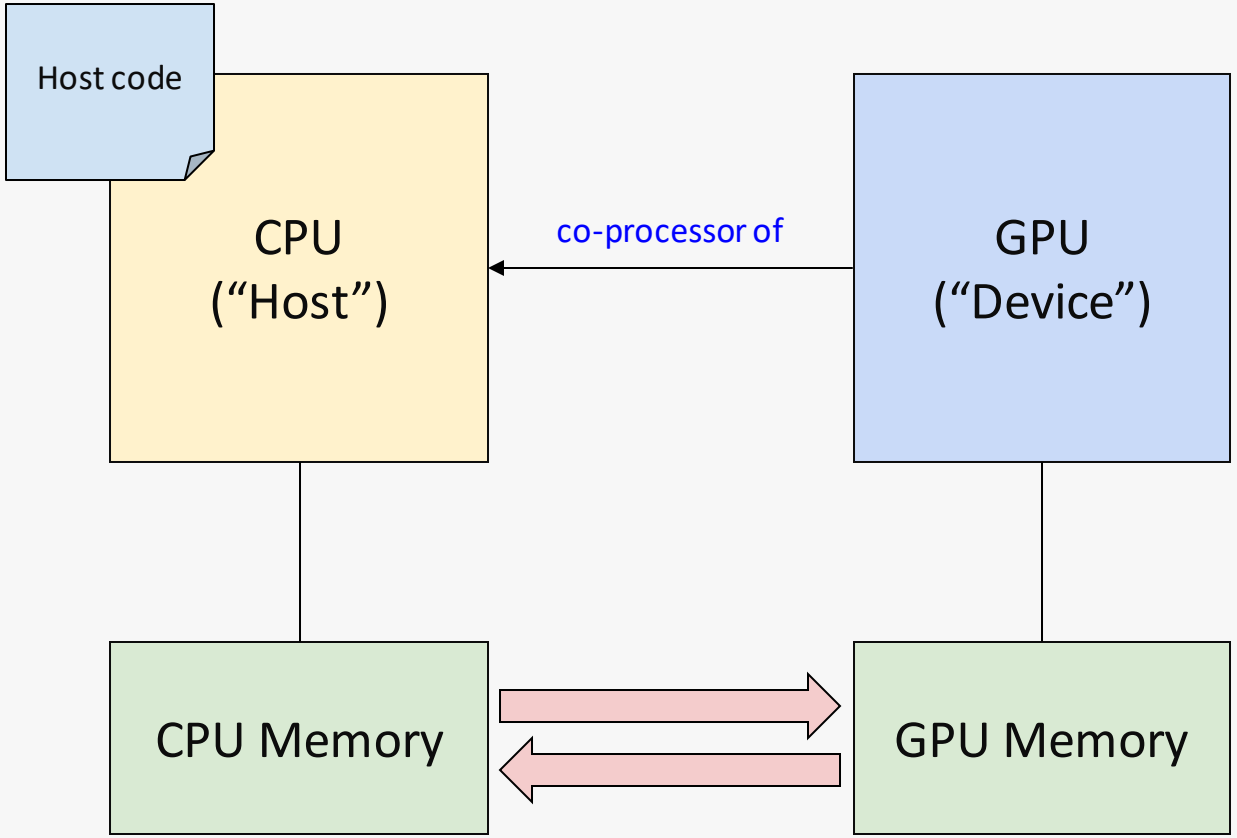
1. The CPU allocates memory on the CPU

Programming on a SIMD CPU

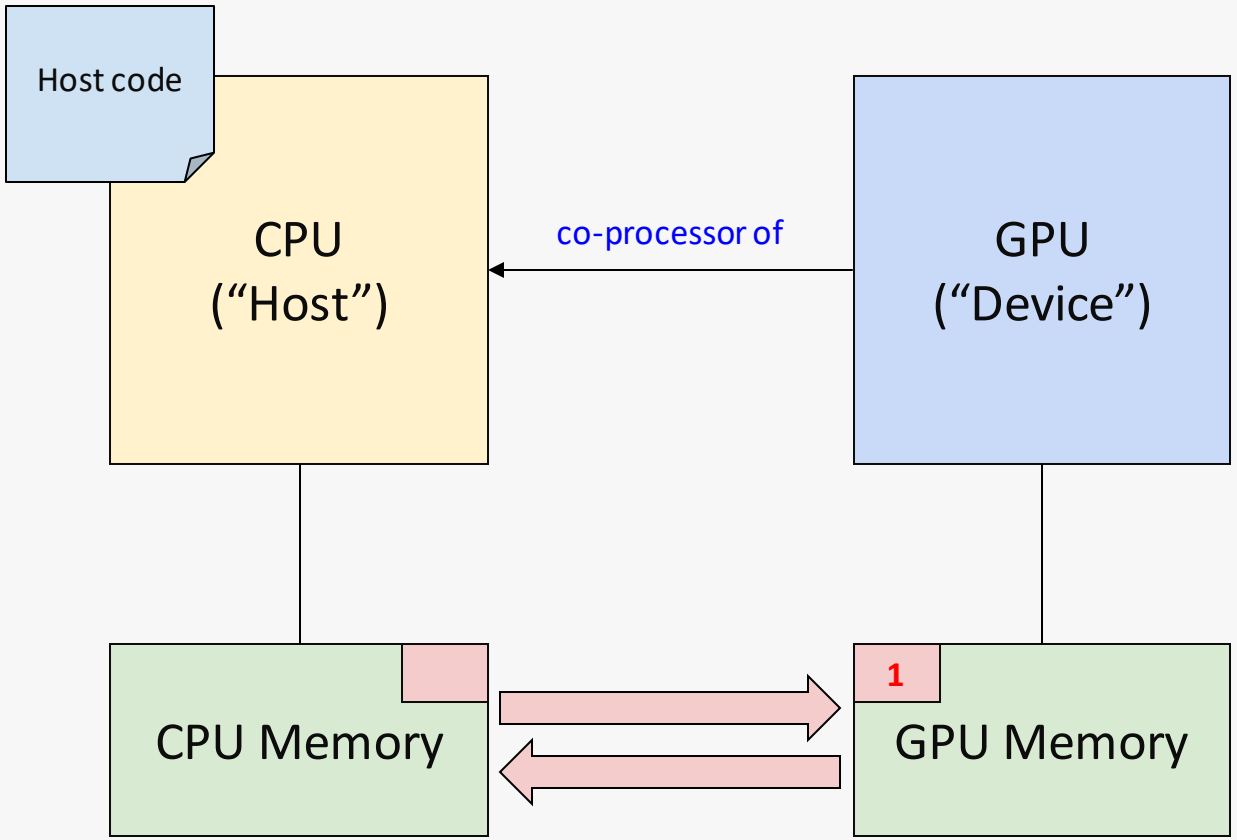


1. The CPU allocates memory on the CPU
2. The CPU executes a kernel using threads and SIMD instructions

Programming on a GPU

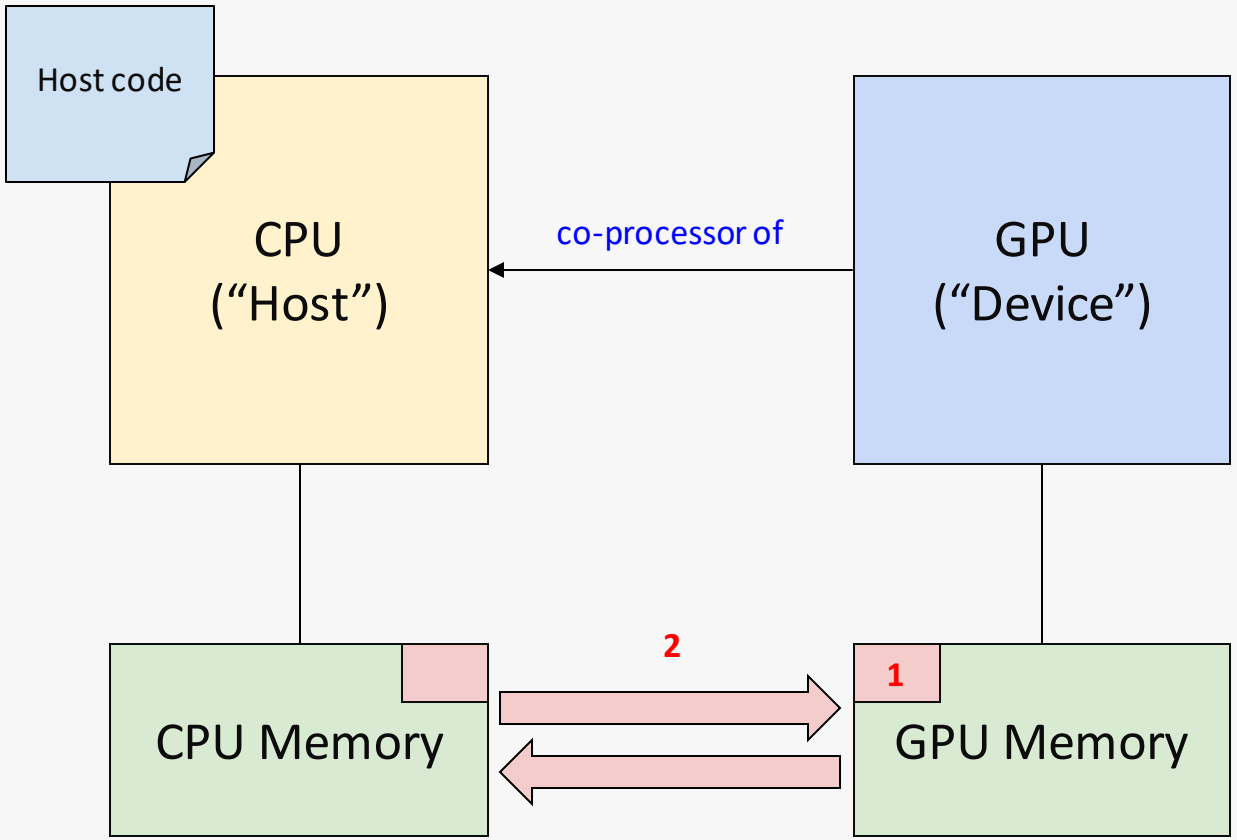


Programming on a GPU



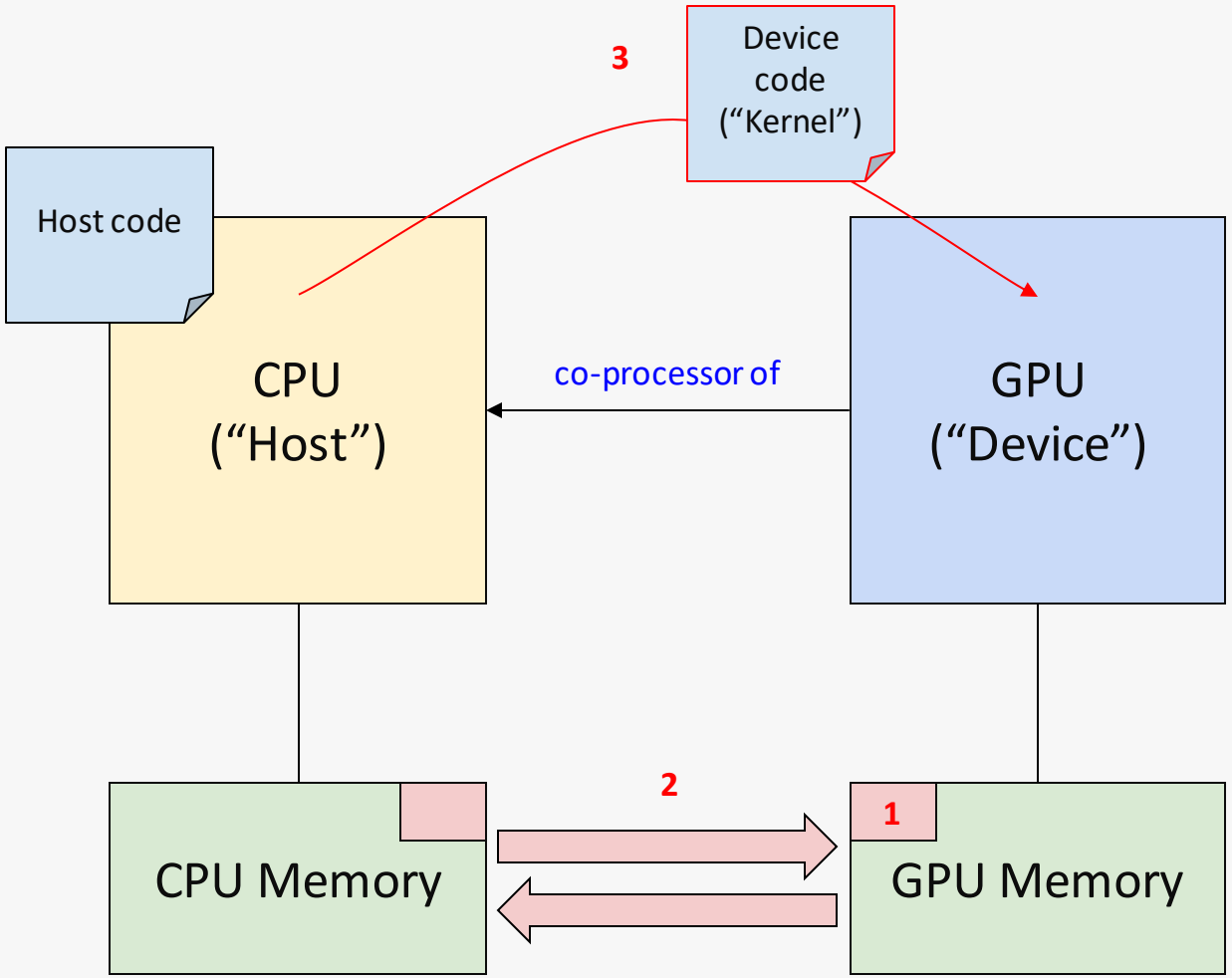
1. The CPU allocates memory on the GPU

Programming on a GPU



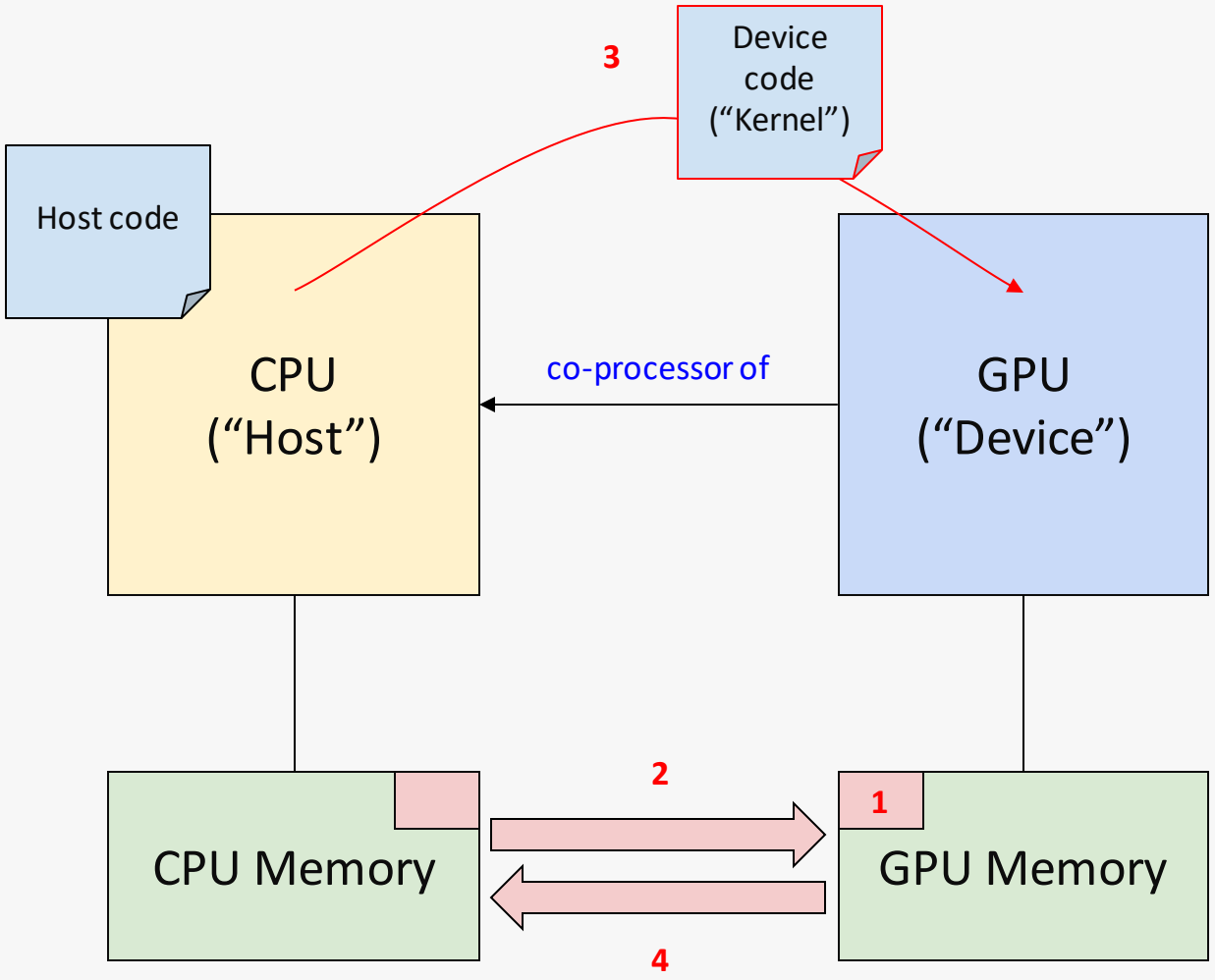
1. The CPU allocates memory on the GPU
2. The CPU copies data from CPU to GPU

Programming on a GPU



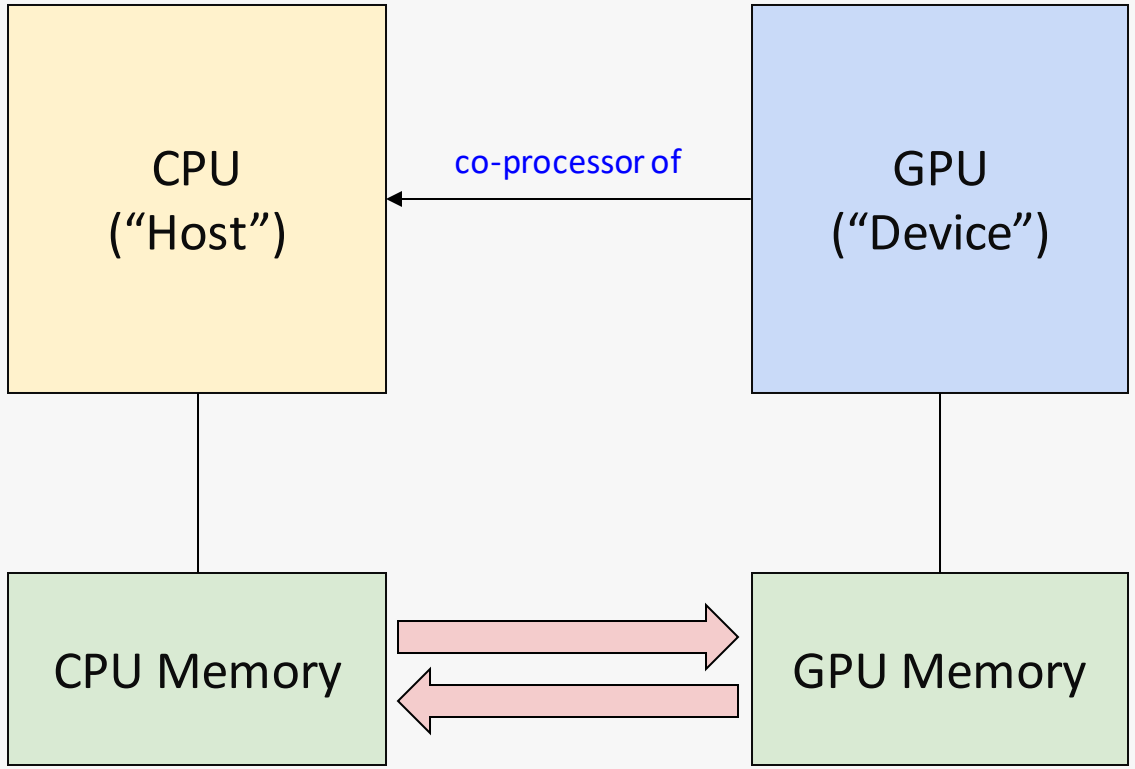
1. The CPU allocates memory on the GPU
2. The CPU copies data from CPU to GPU
3. The CPU launches kernel(s) on the GPU

Programming on a GPU

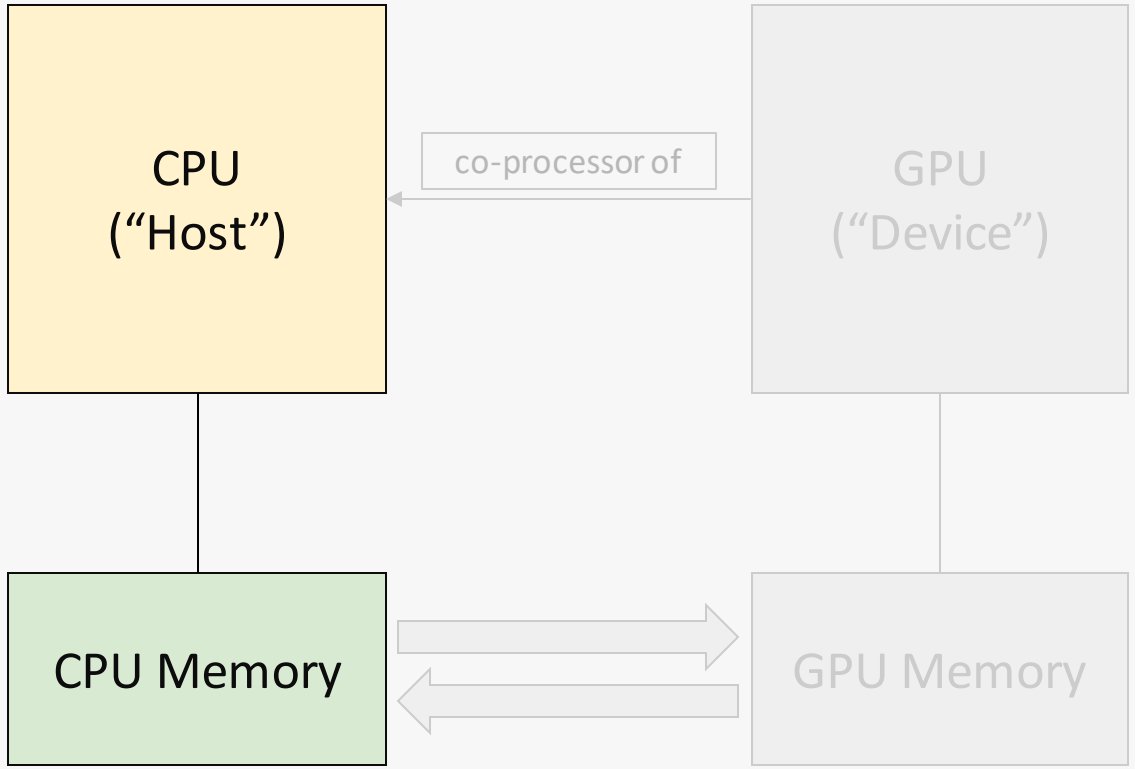


1. The CPU allocates memory on the GPU
2. The CPU copies data from CPU to GPU
3. The CPU launches kernel(s) on the GPU
4. The CPU copies data to CPU from GPU

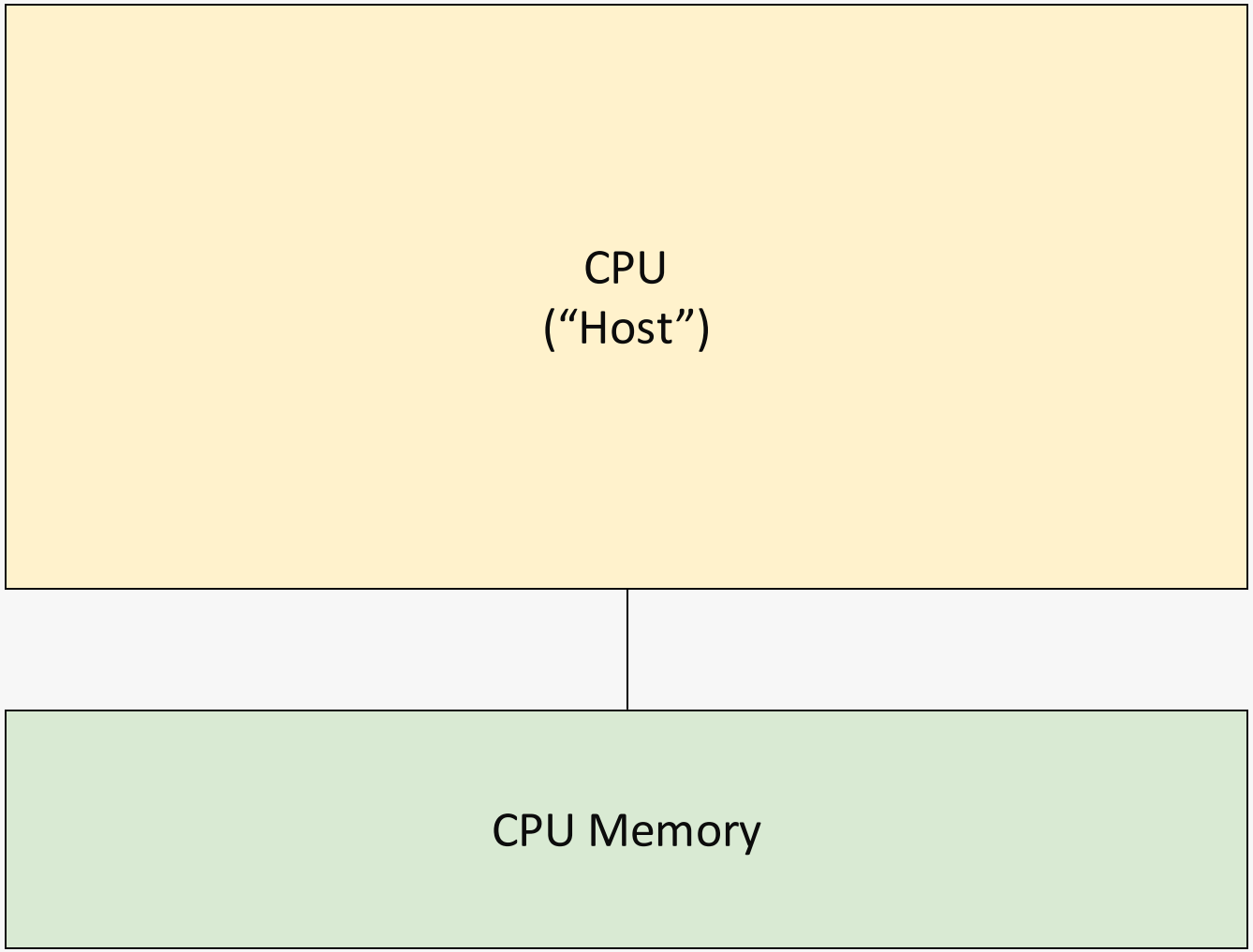
So let’s take a look now at each of these



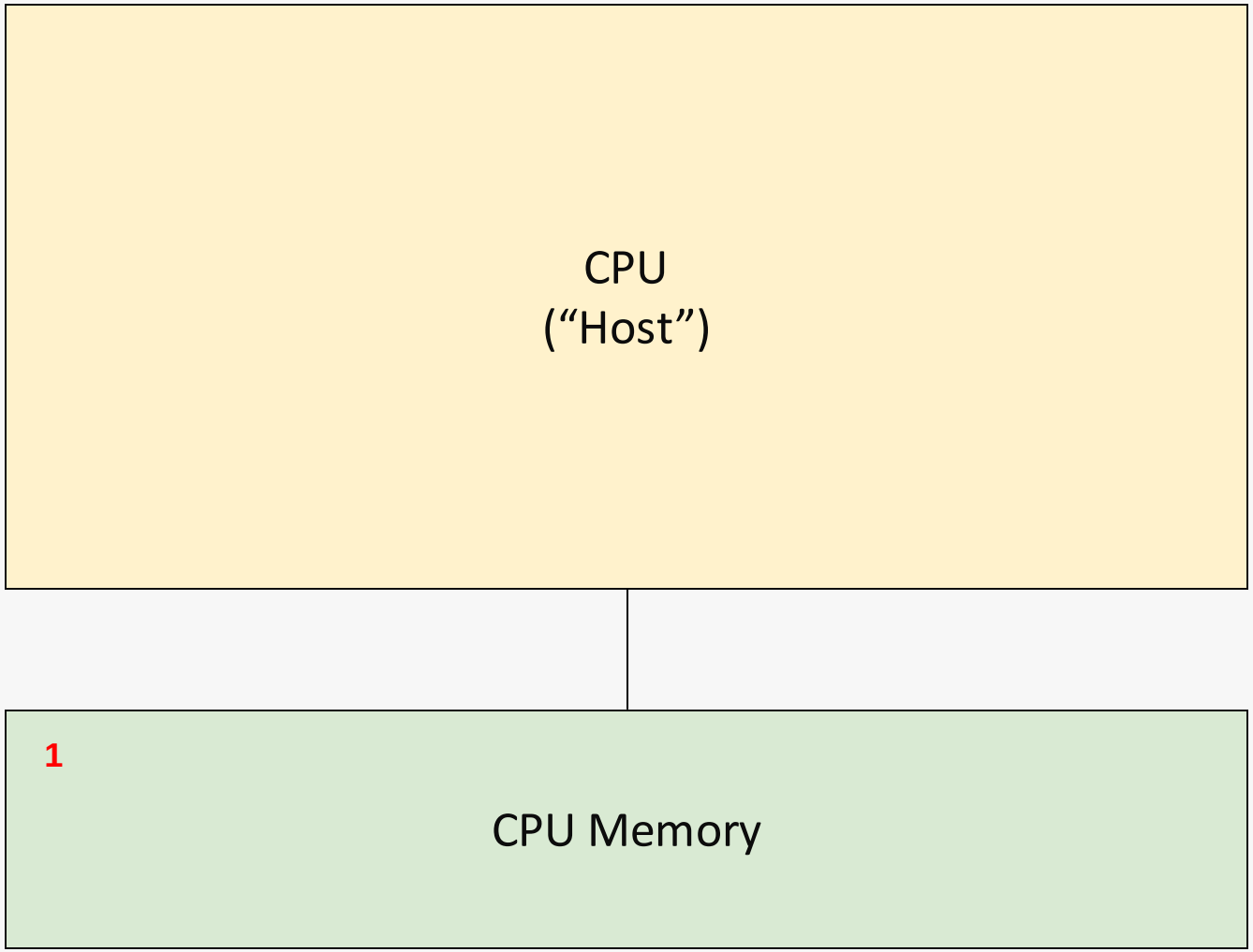
Let’s take a look at the CPU...



Let’s take a look at the CPU...

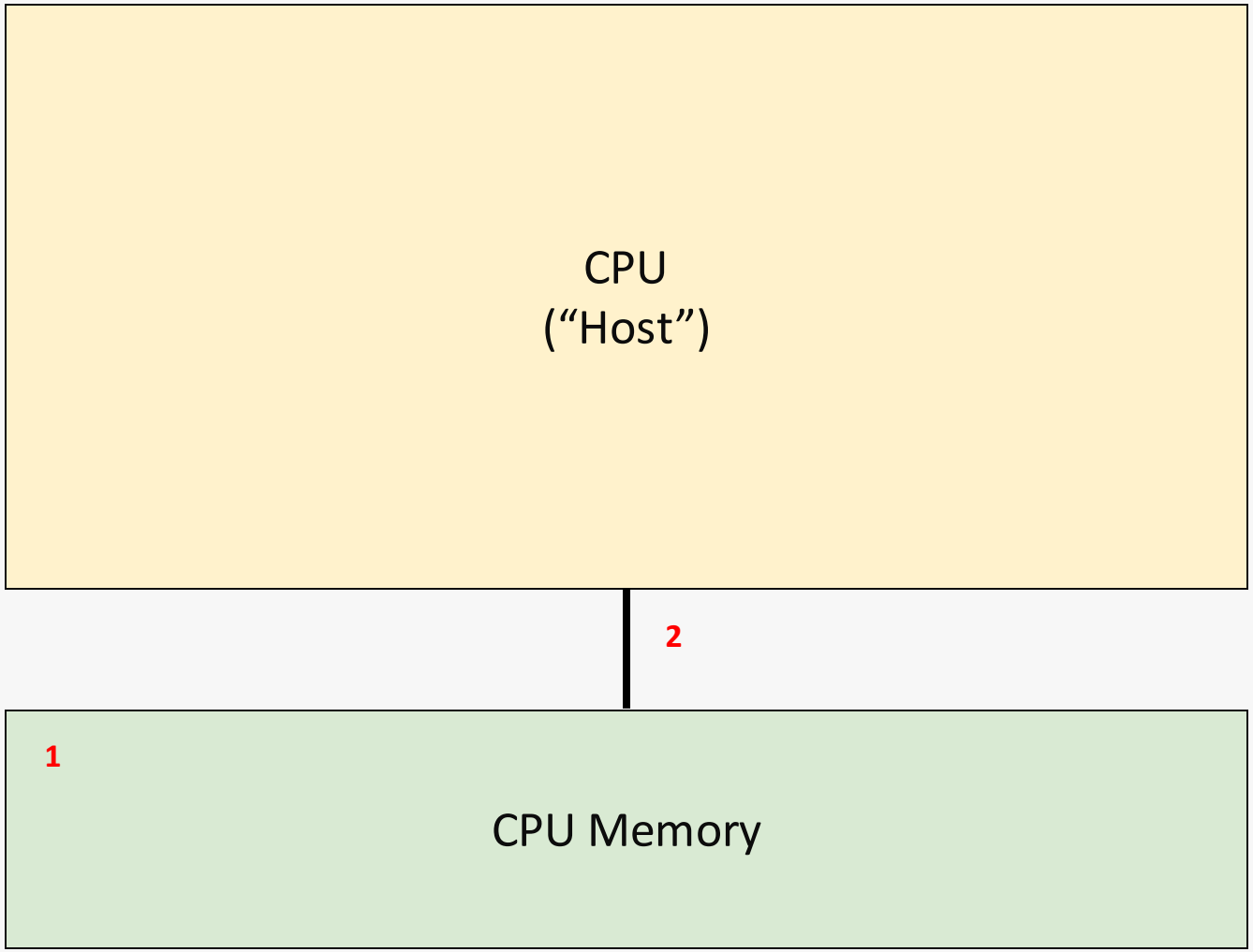


Let’s take a look at the CPU...



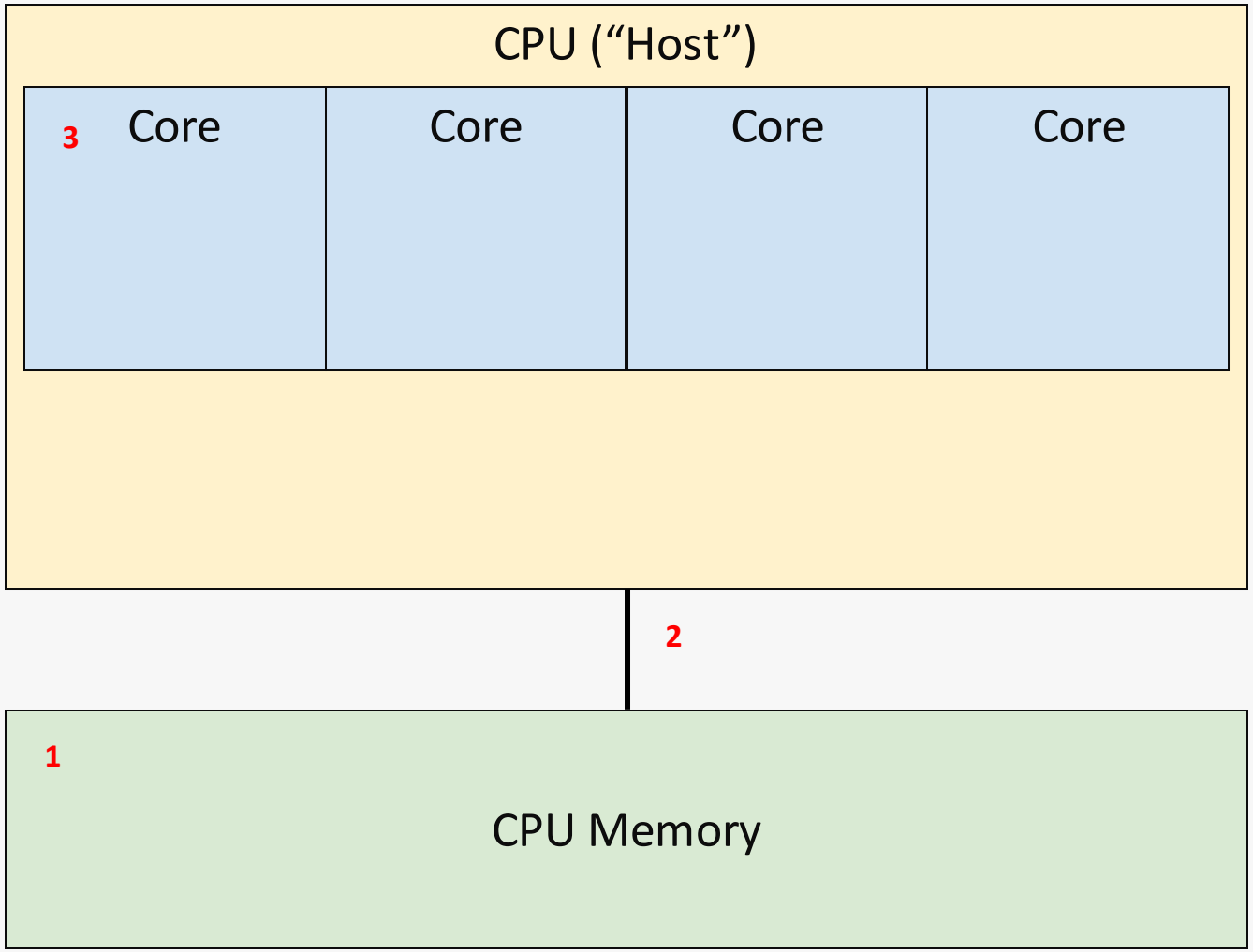
1. A CPU has a region of dedicated memory

Let’s take a look at the CPU...



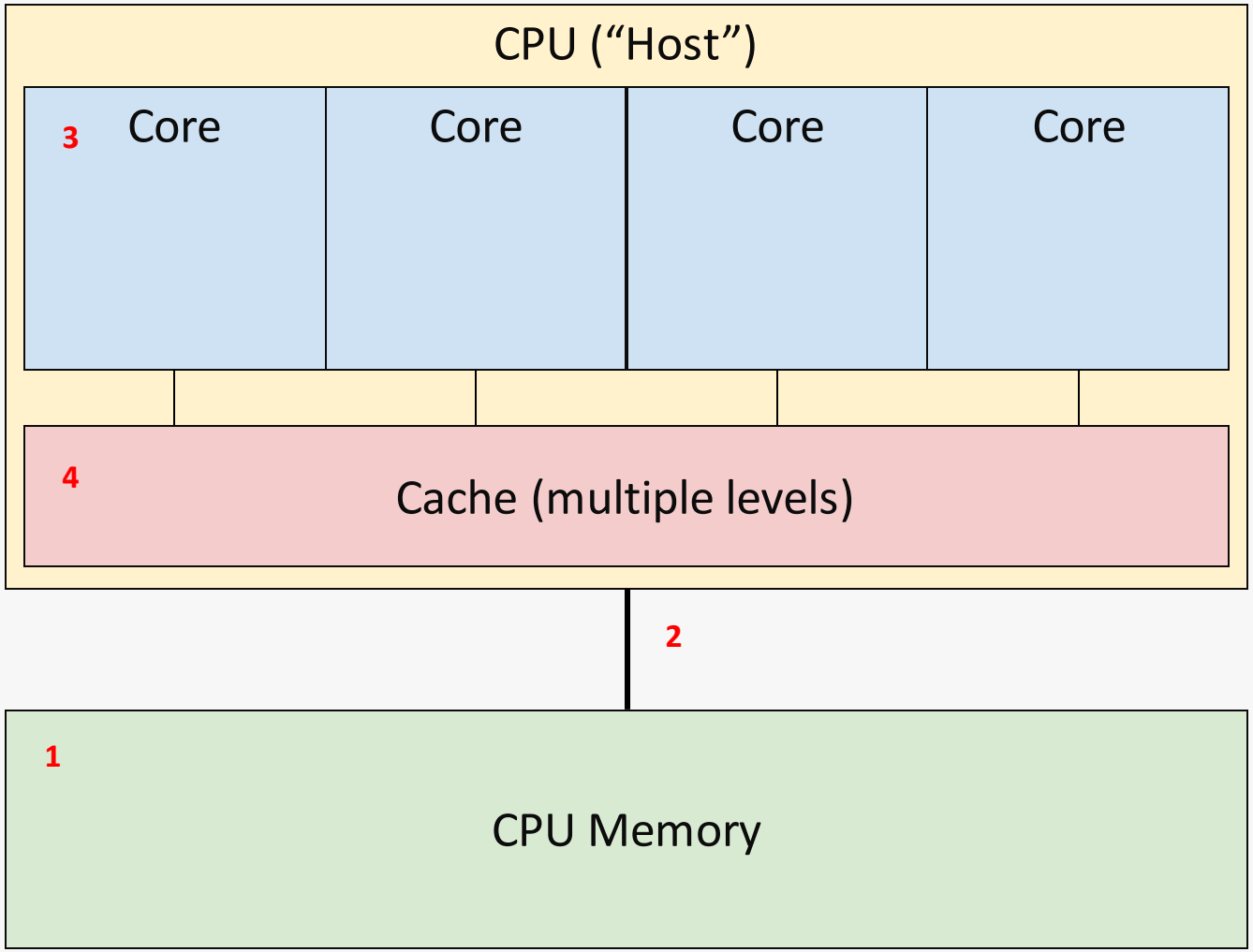
1. A CPU has a region of dedicated memory
2. CPU memory is connected to the CPU via a bus

Let’s take a look at the CPU...



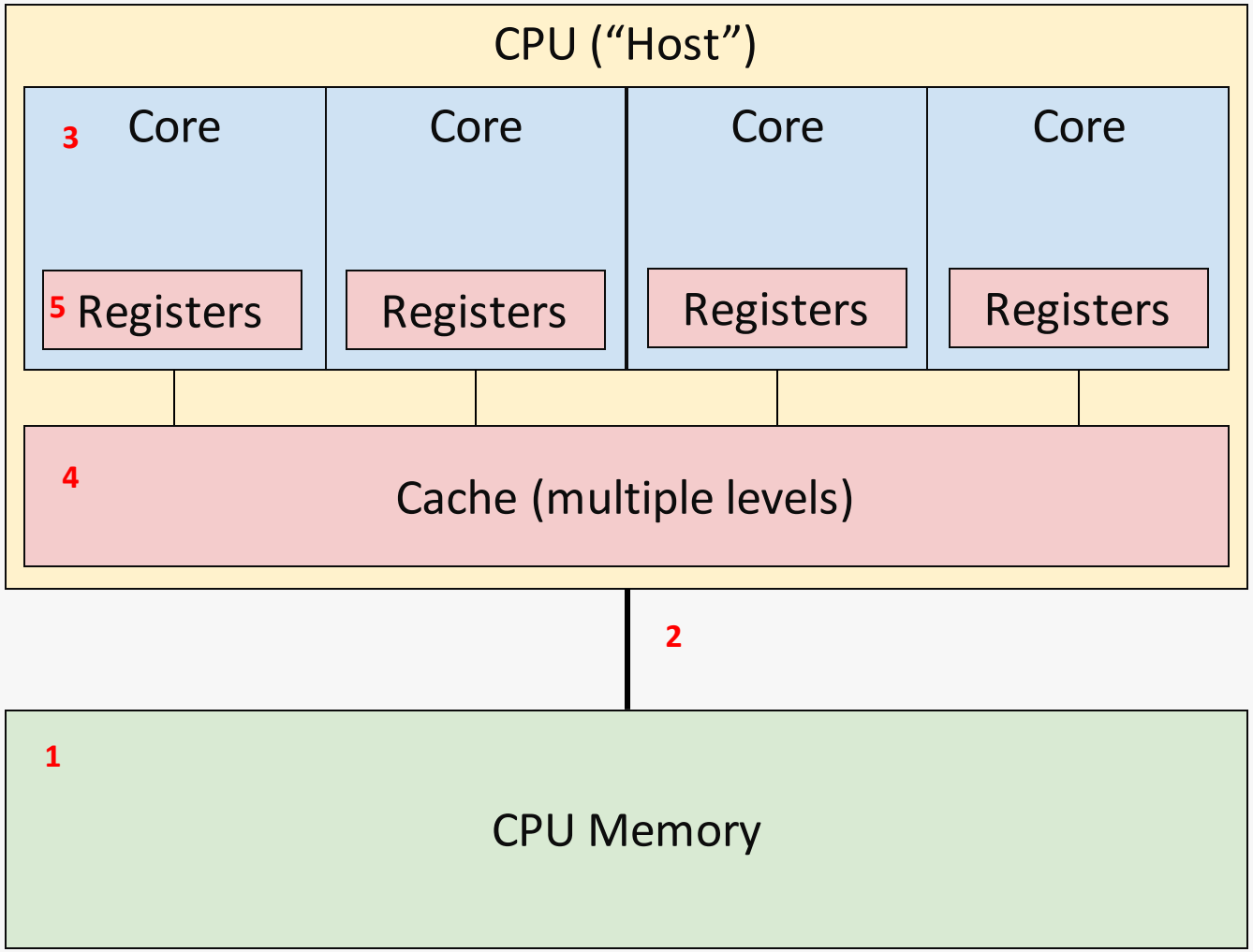
1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores

Let’s take a look at the CPU...



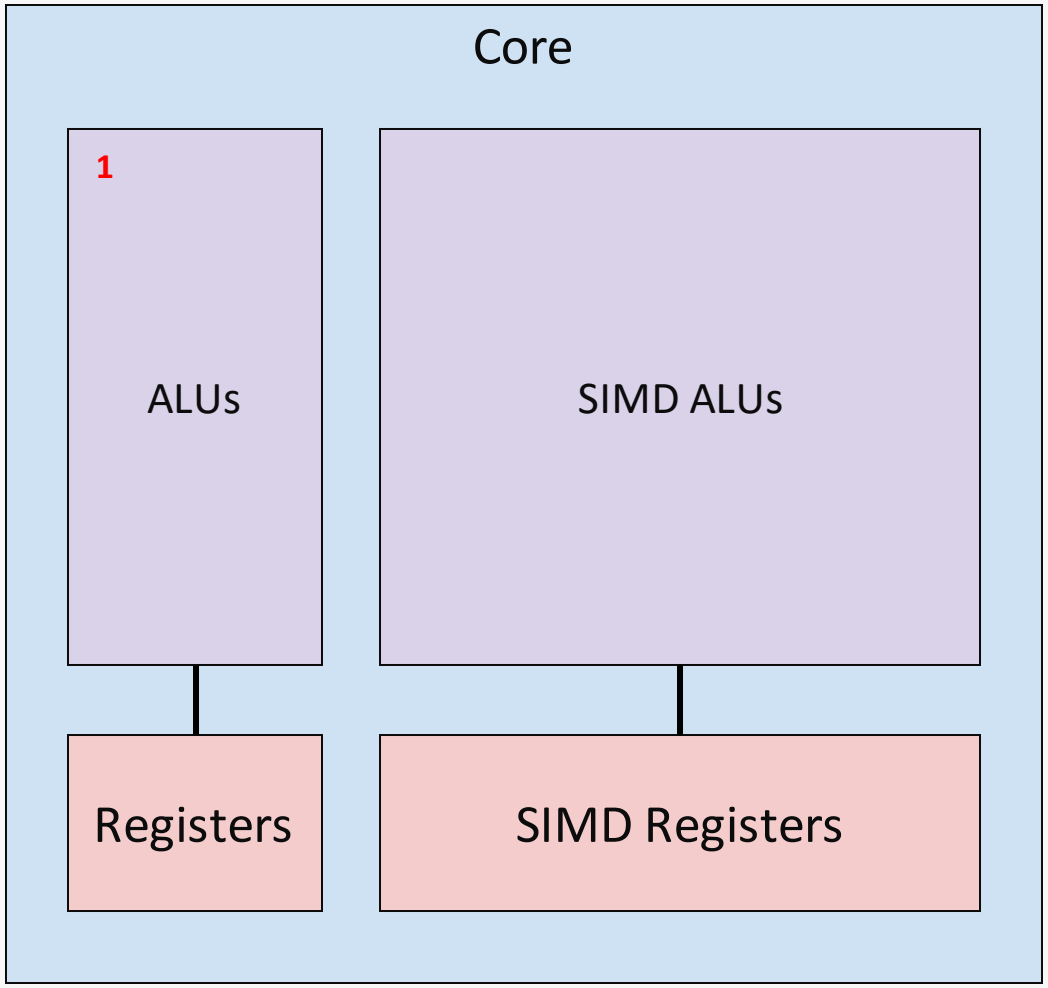
1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores
4. A CPU has a number of caches of different levels

Let’s take a look at the CPU...



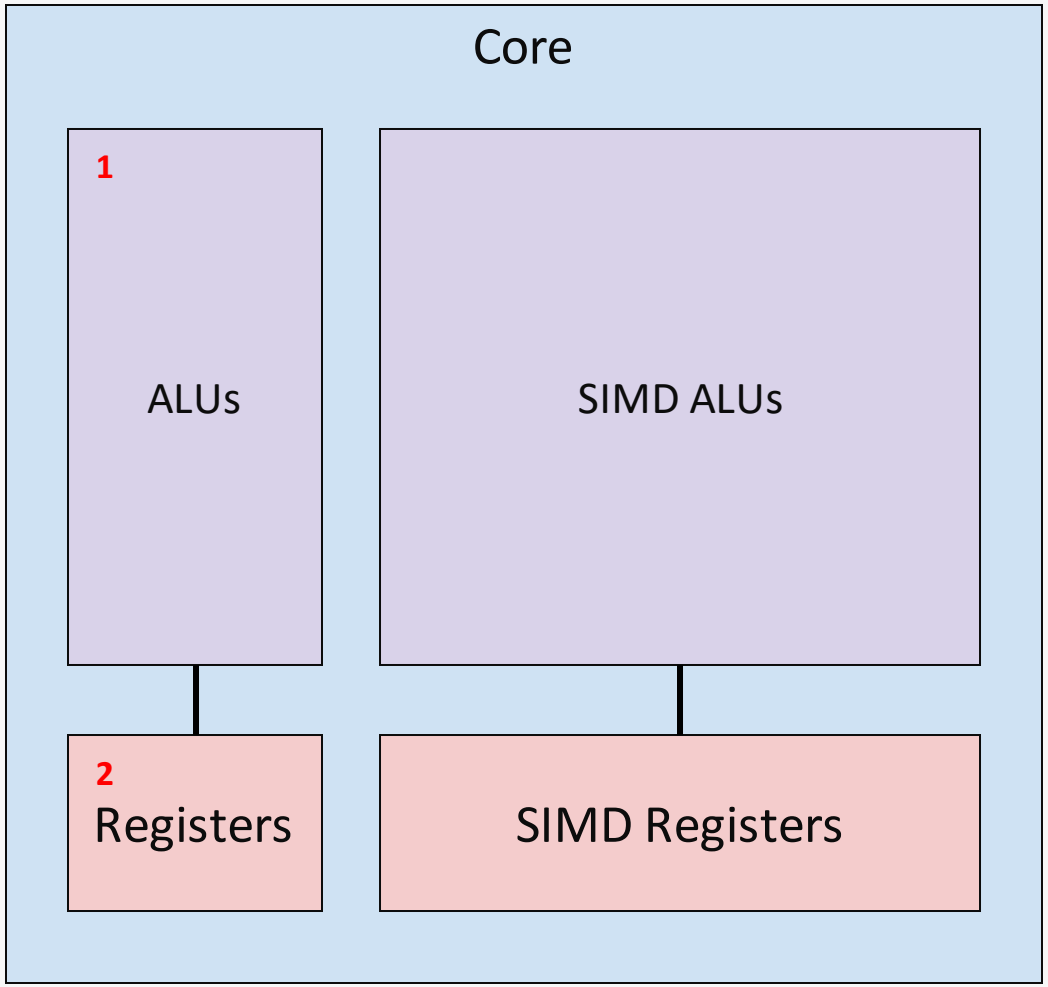
1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores
4. A CPU has a number of caches of different levels
5. Each CPU core has dedicated registers

Inside a CPU core



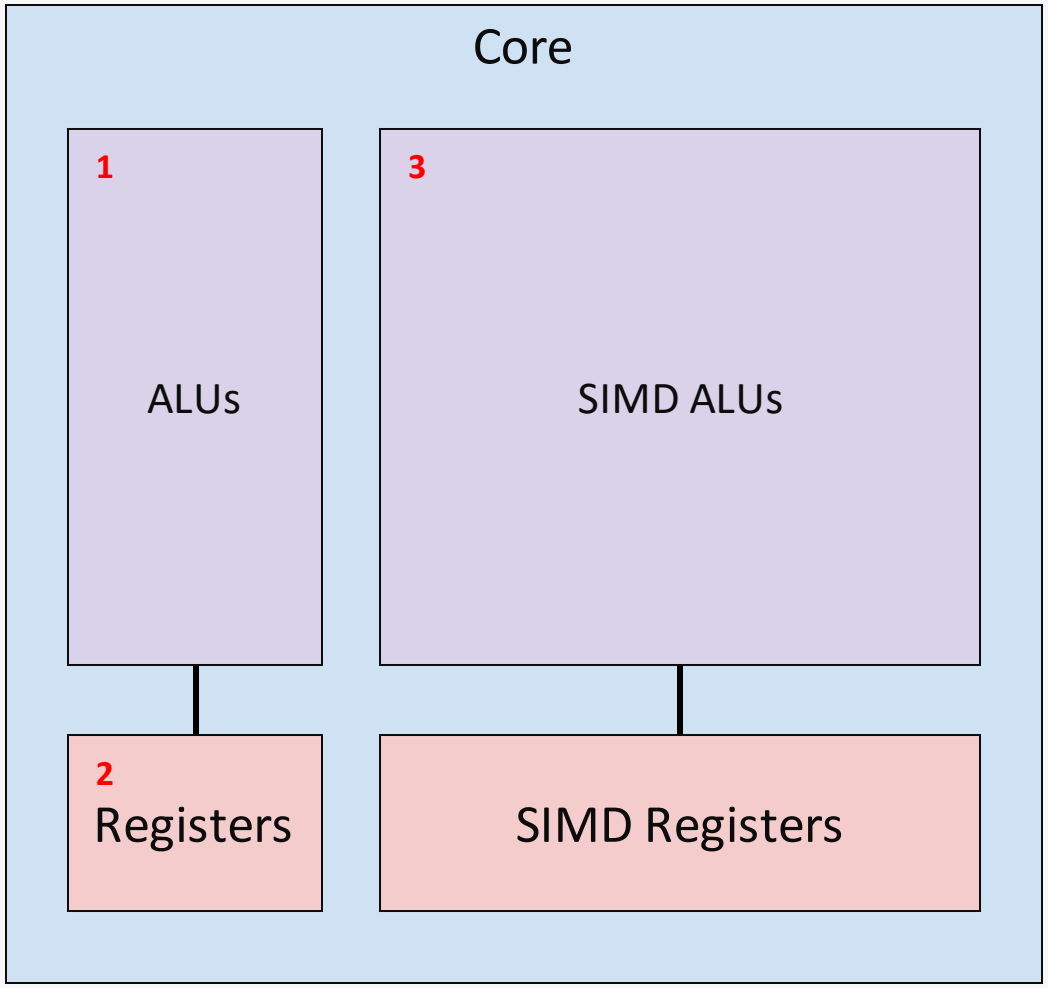
1. A CPU core you have a number of standard ALUs

Inside a CPU core



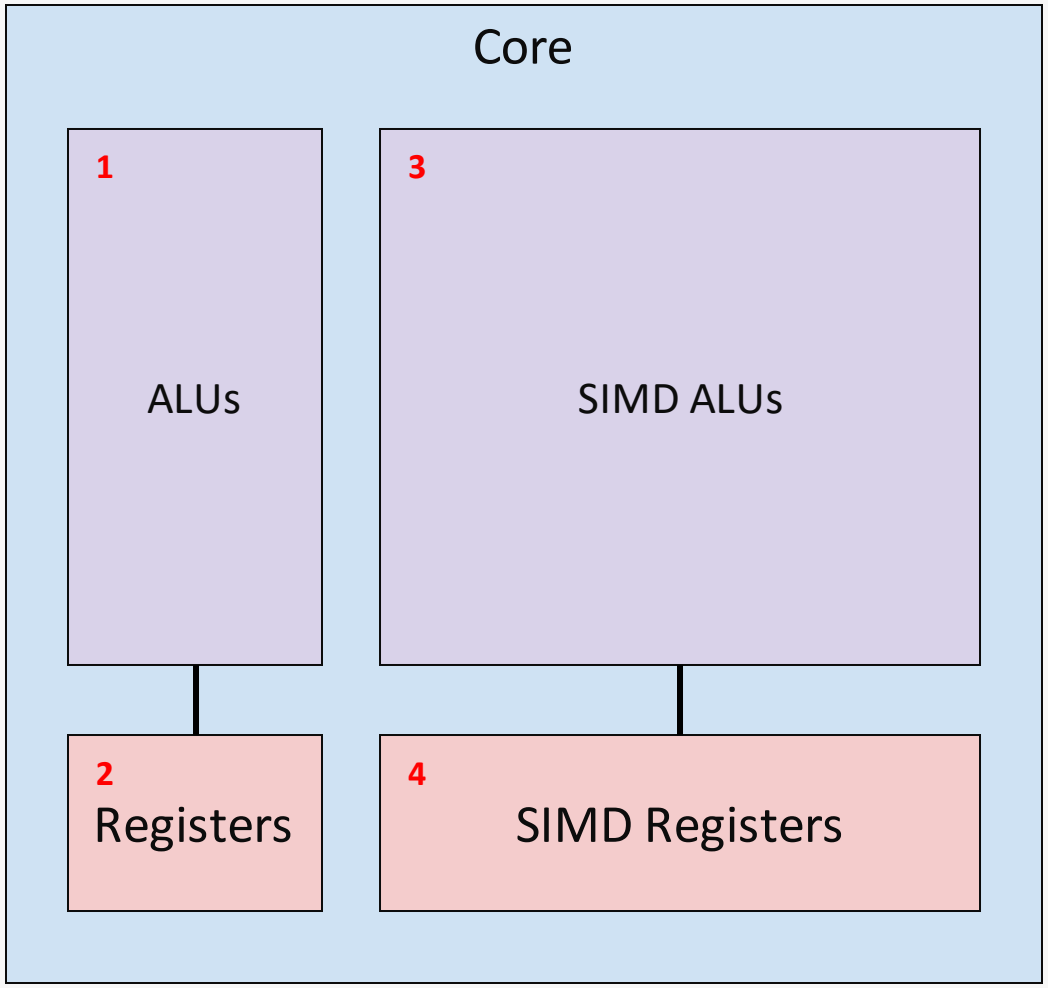
1. A CPU core you have a number of standard ALUs
2. These are connected to standard registers

Inside a CPU core



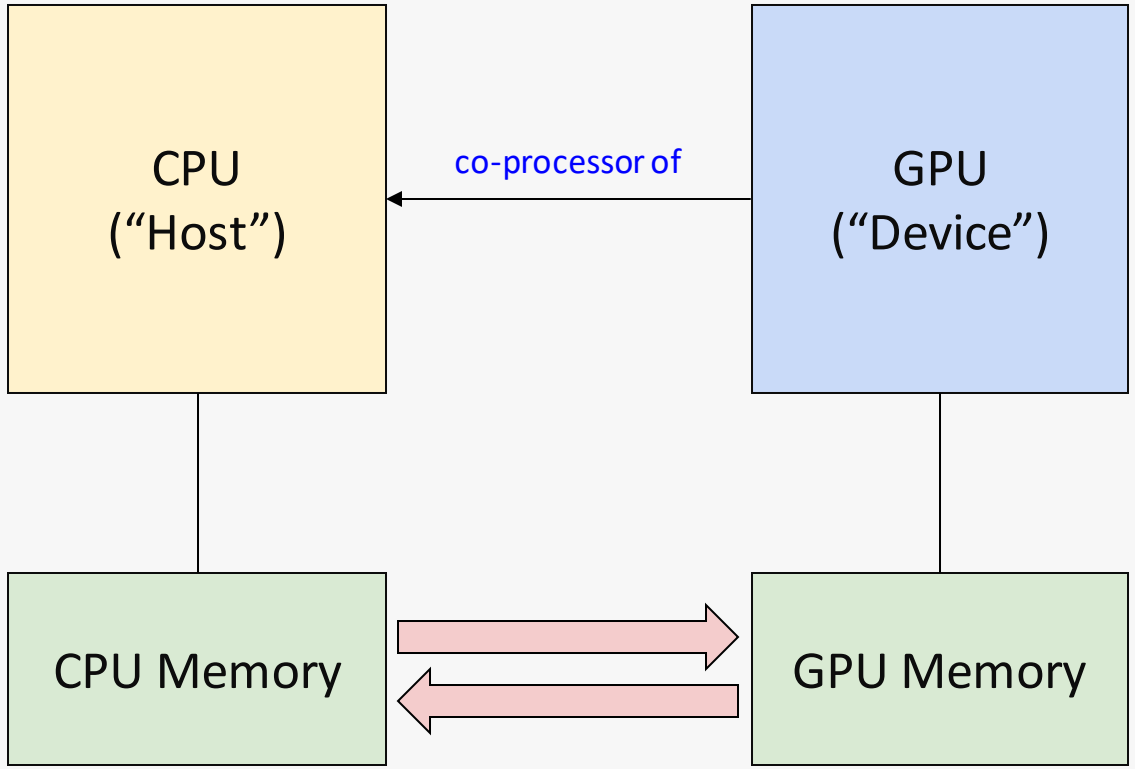
1. A CPU core you have a number of standard ALUs
2. These are connected to standard registers
3. A CPU core also have SIMD ALUs

Inside a CPU core

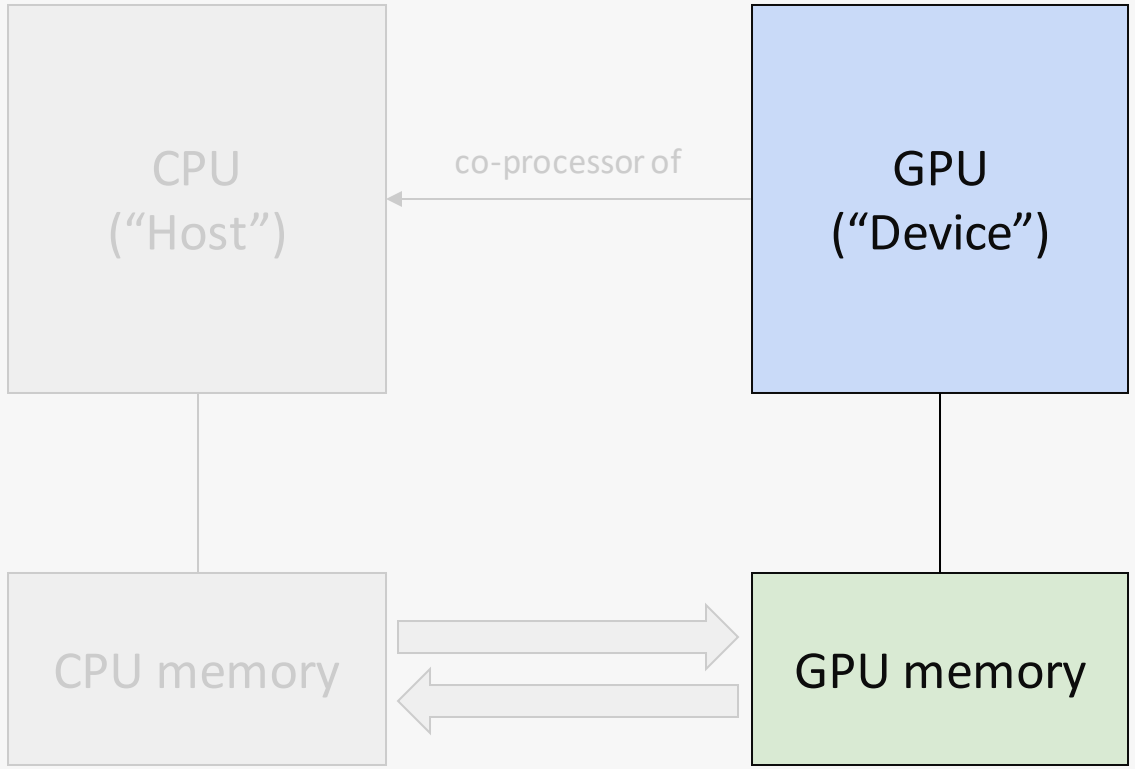


1. A CPU core you have a number of standard ALUs
2. These are connected to standard registers
3. A CPU core also have SIMD ALUs
4. These are connected to SIMD registers

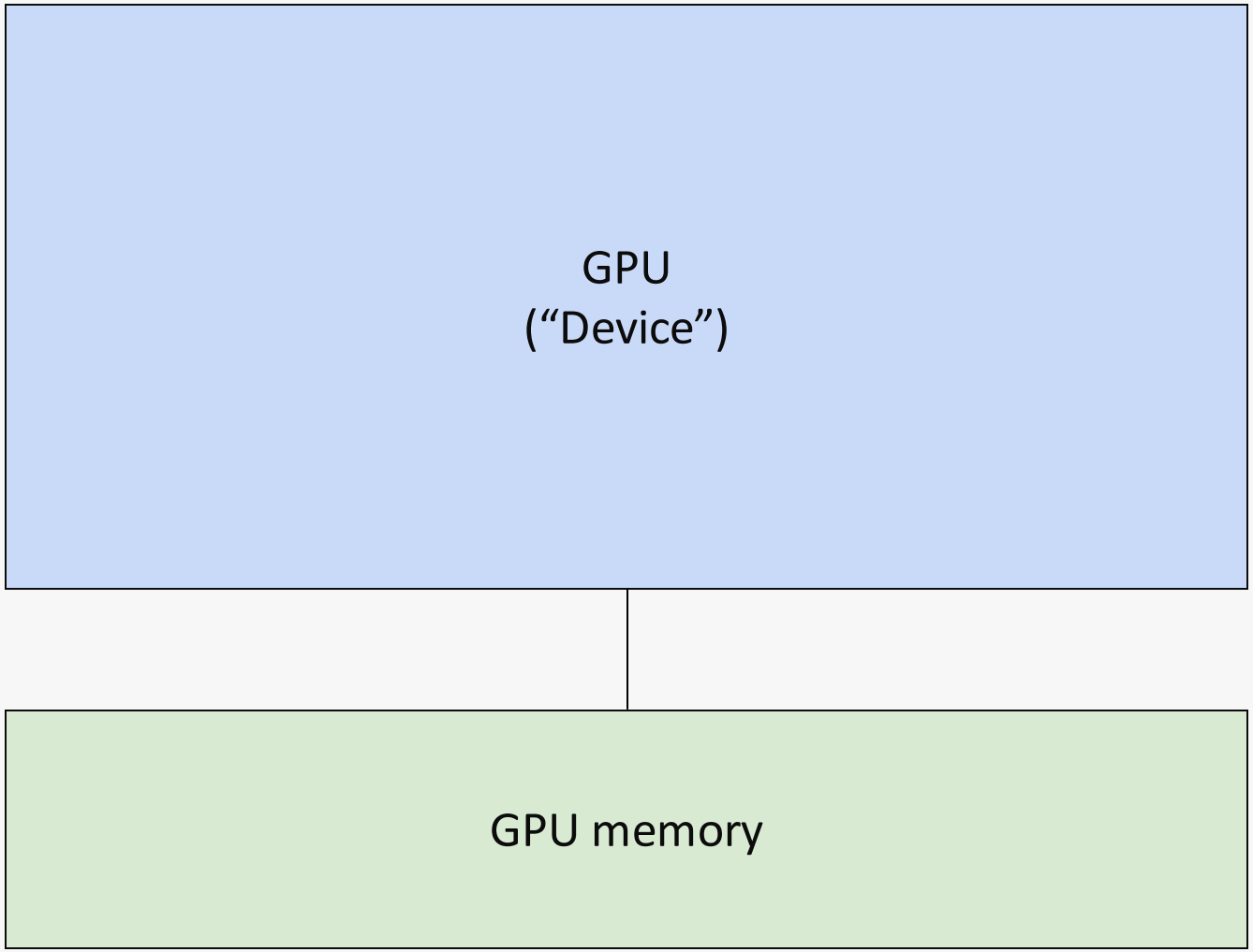
Now let’s take a look at the GPU...



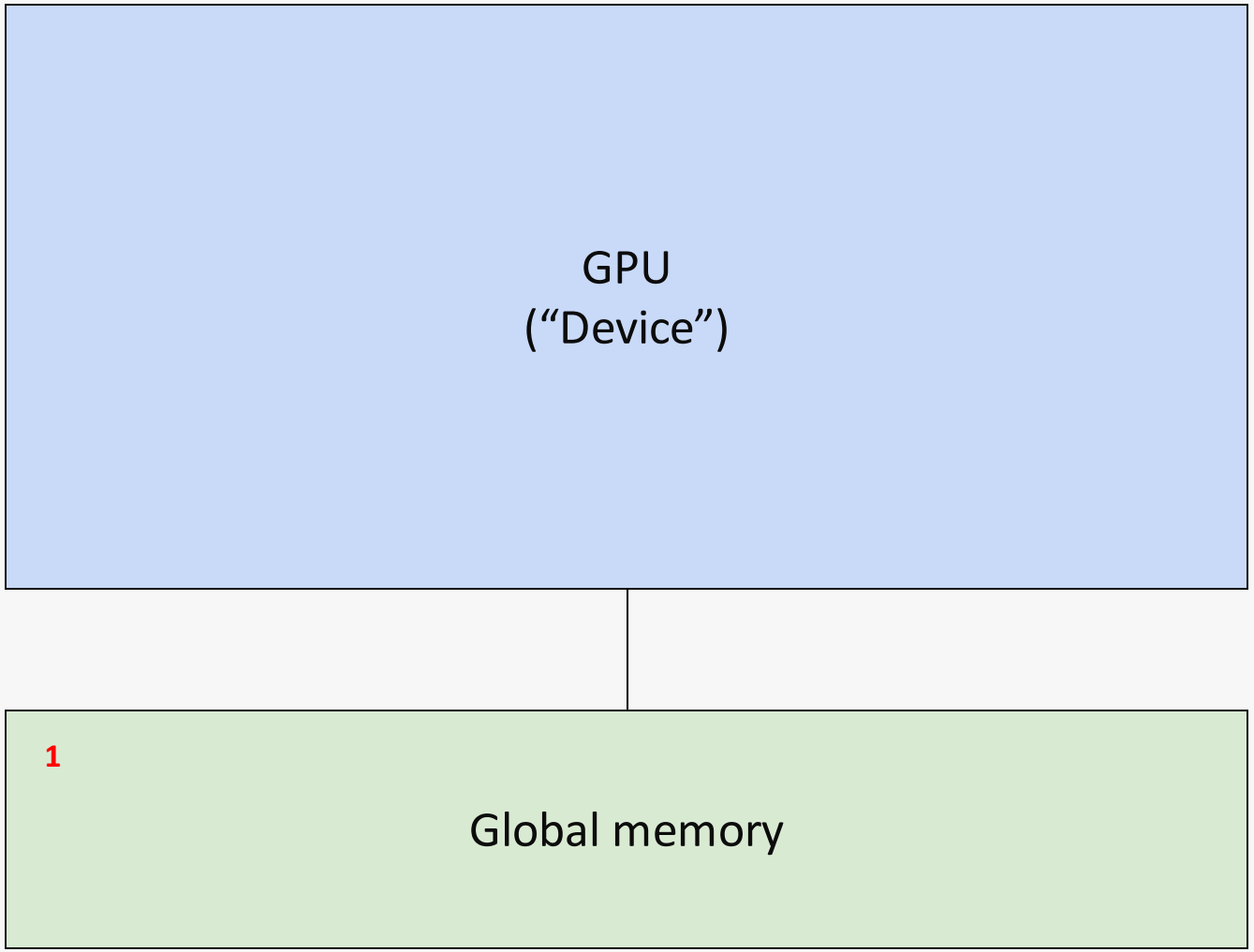
Now let’s take a look at the GPU...



Now let’s take a look at the GPU...

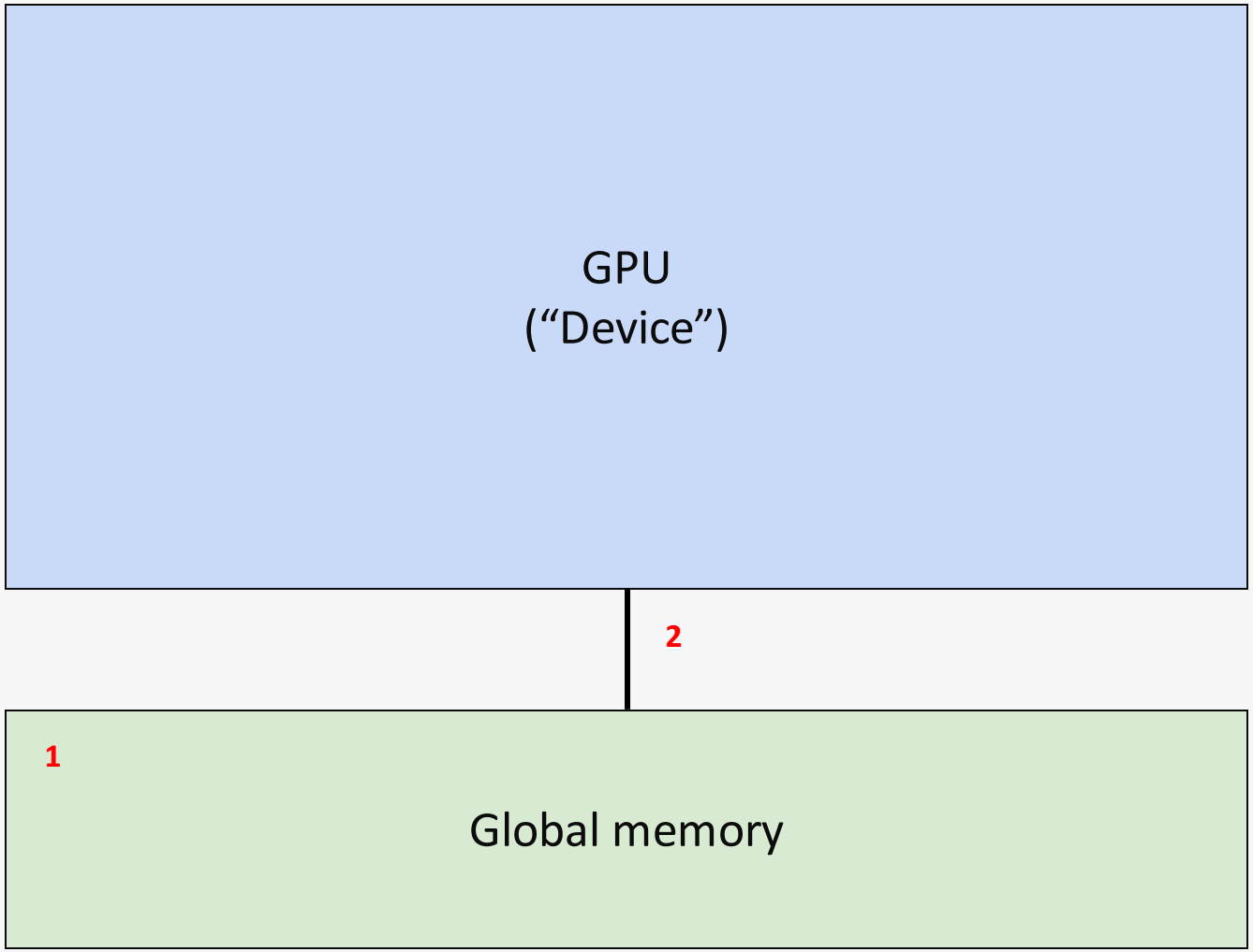


Now let’s take a look at the GPU...



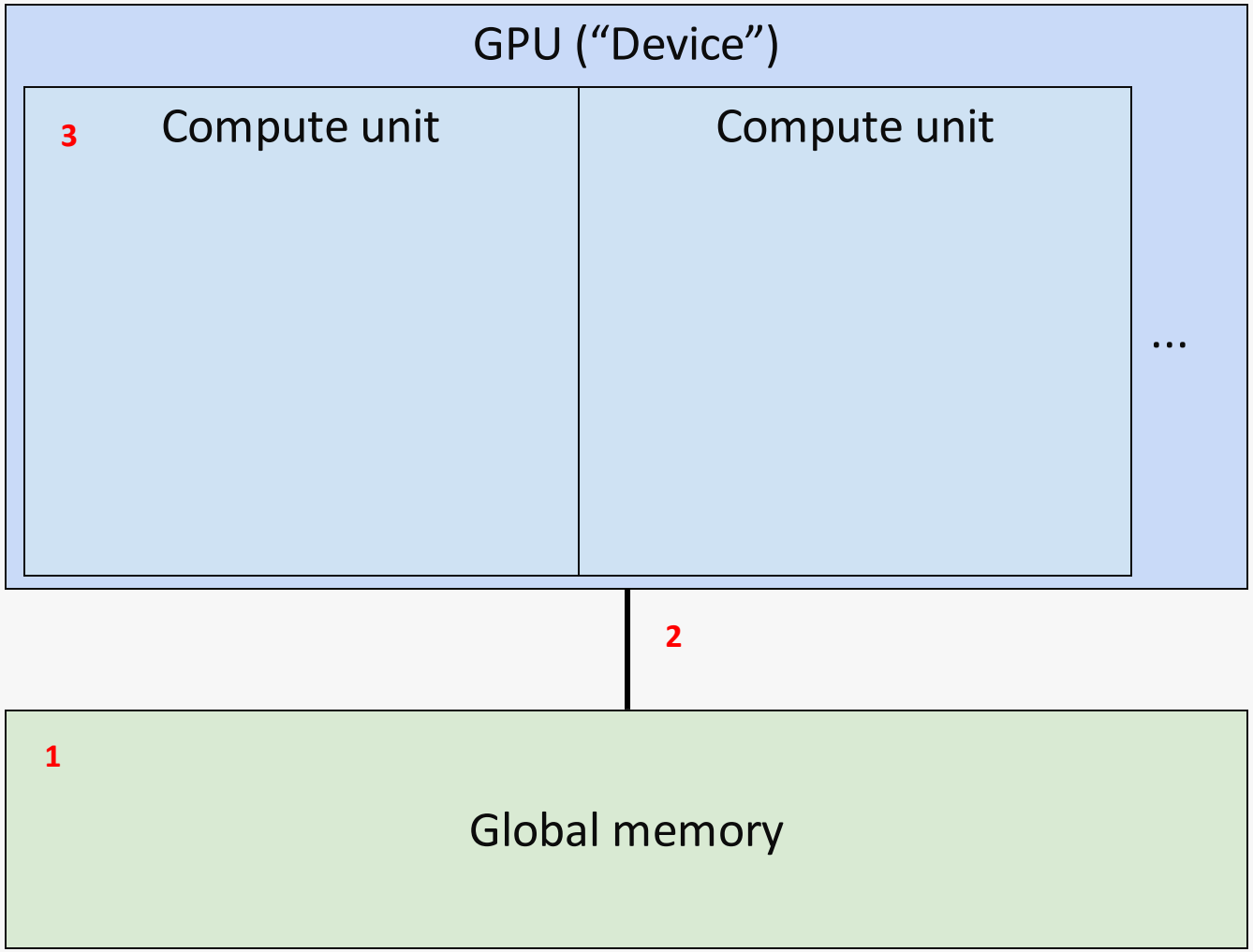
1. A GPU has a region of dedicated global memory

Now let’s take a look at the GPU...



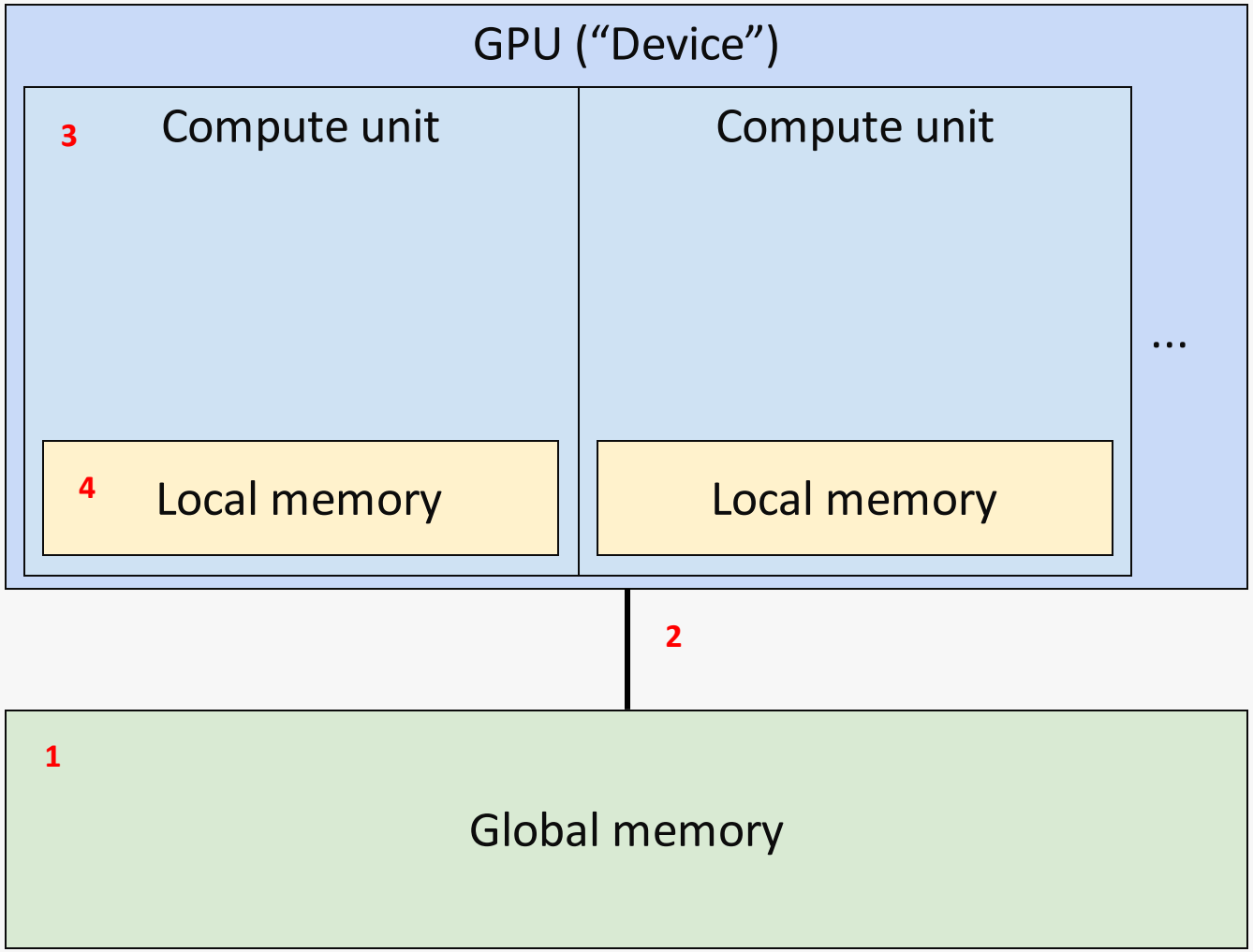
1. A GPU has a region of dedicated global memory
2. Global memory is connected via a bus

Now let's take a look at the GPU...



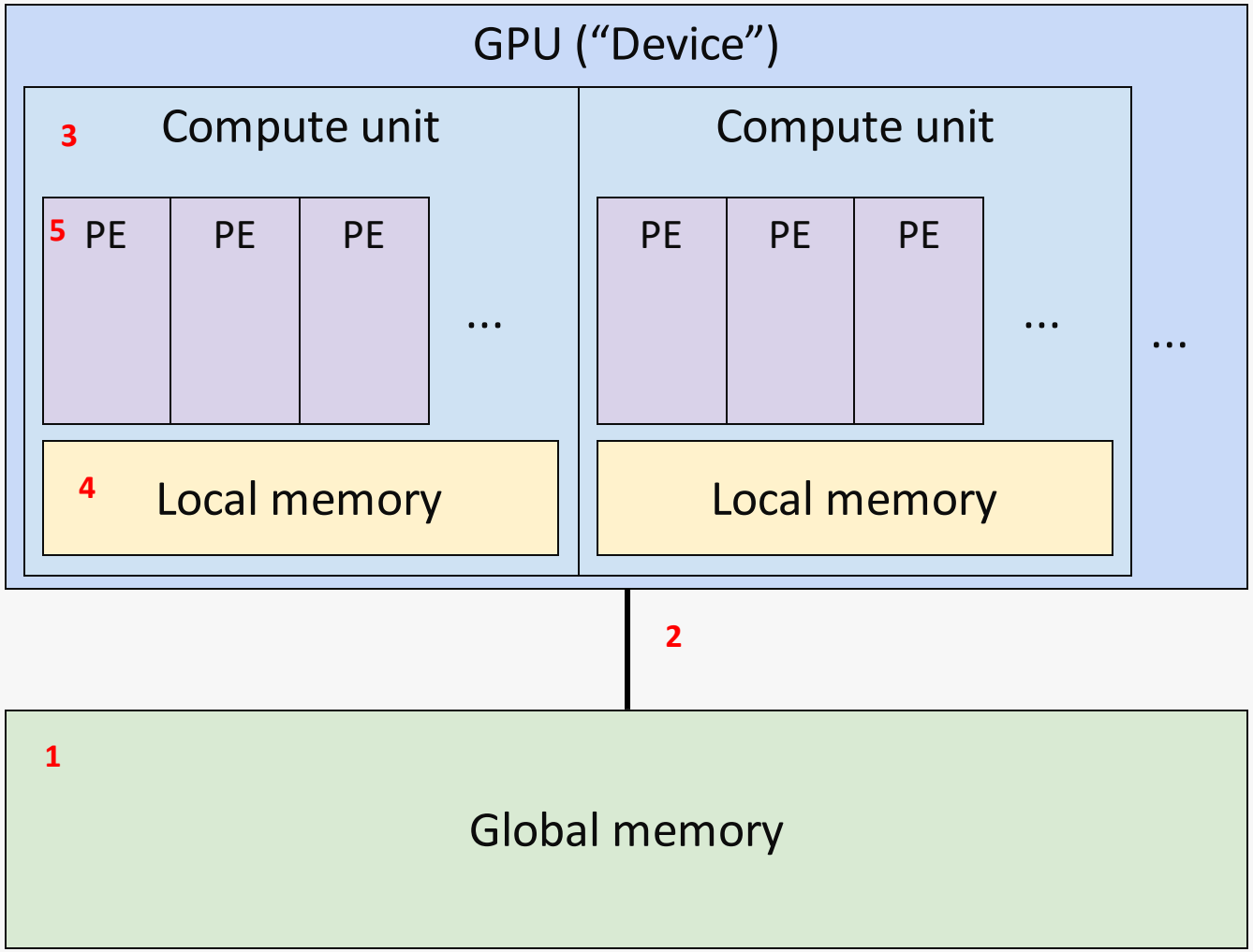
1. A GPU has a region of dedicated global memory
2. Global memory is connected via a bus
3. A GPU is divided into a number of compute units

Now let's take a look at the GPU...



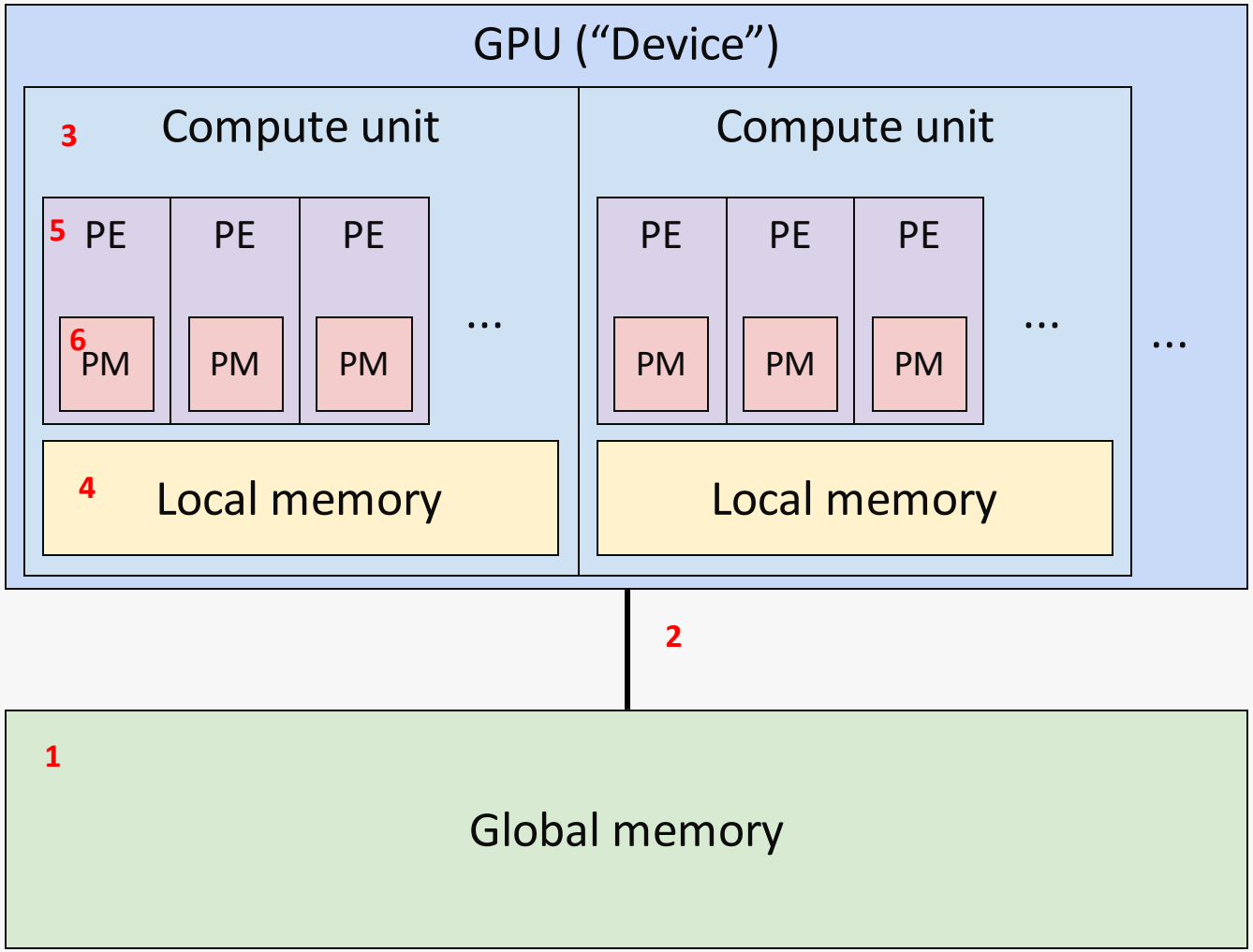
1. A GPU has a region of dedicated global memory
2. Global memory is connected via a bus
3. A GPU is divided into a number of compute units
4. Each compute unit has dedicated local memory

Now let's take a look at the GPU...



1. A GPU has a region of dedicated global memory
2. Global memory is connected via a bus
3. A GPU is divided into a number of compute units
4. Each compute unit has dedicated local memory
5. Each compute unit has a number of processing elements

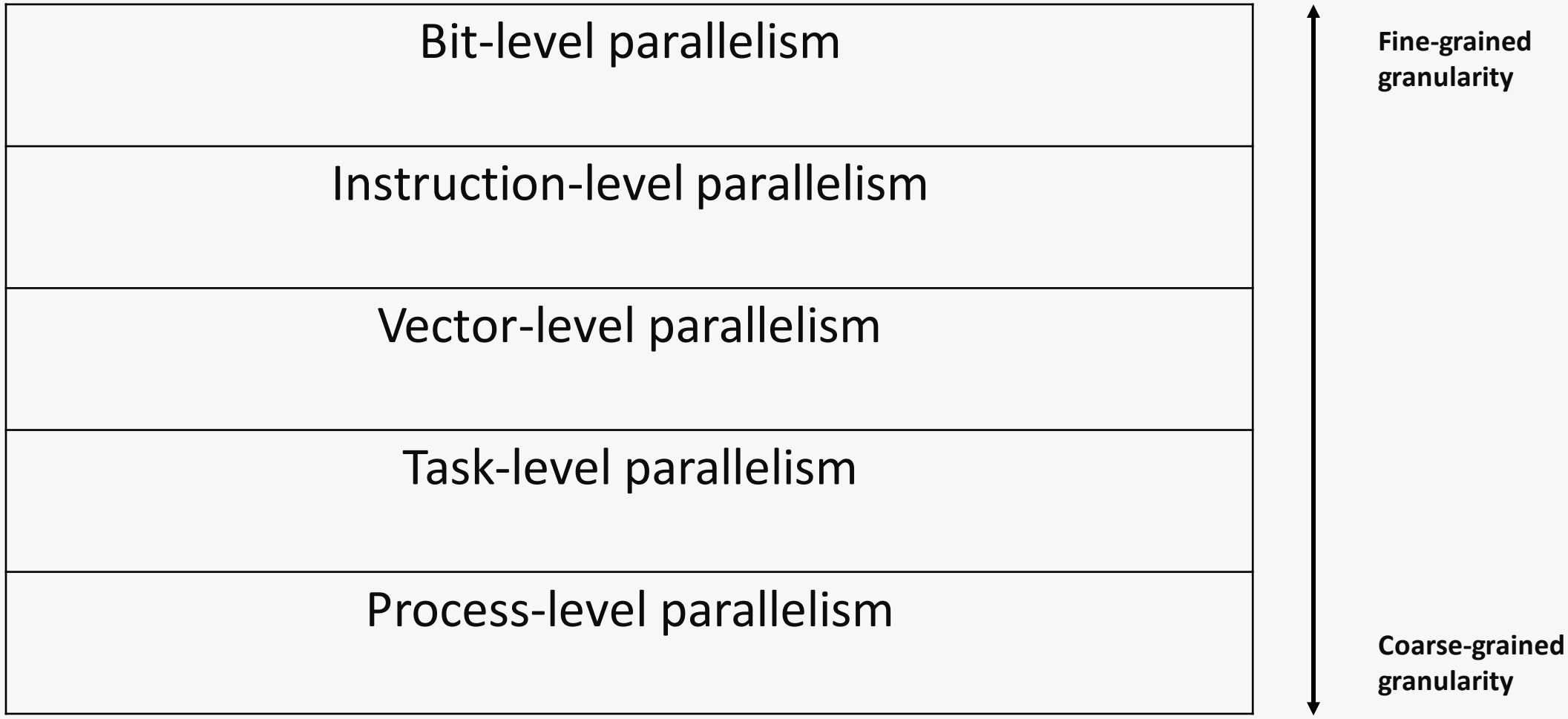
Now let's take a look at the GPU...



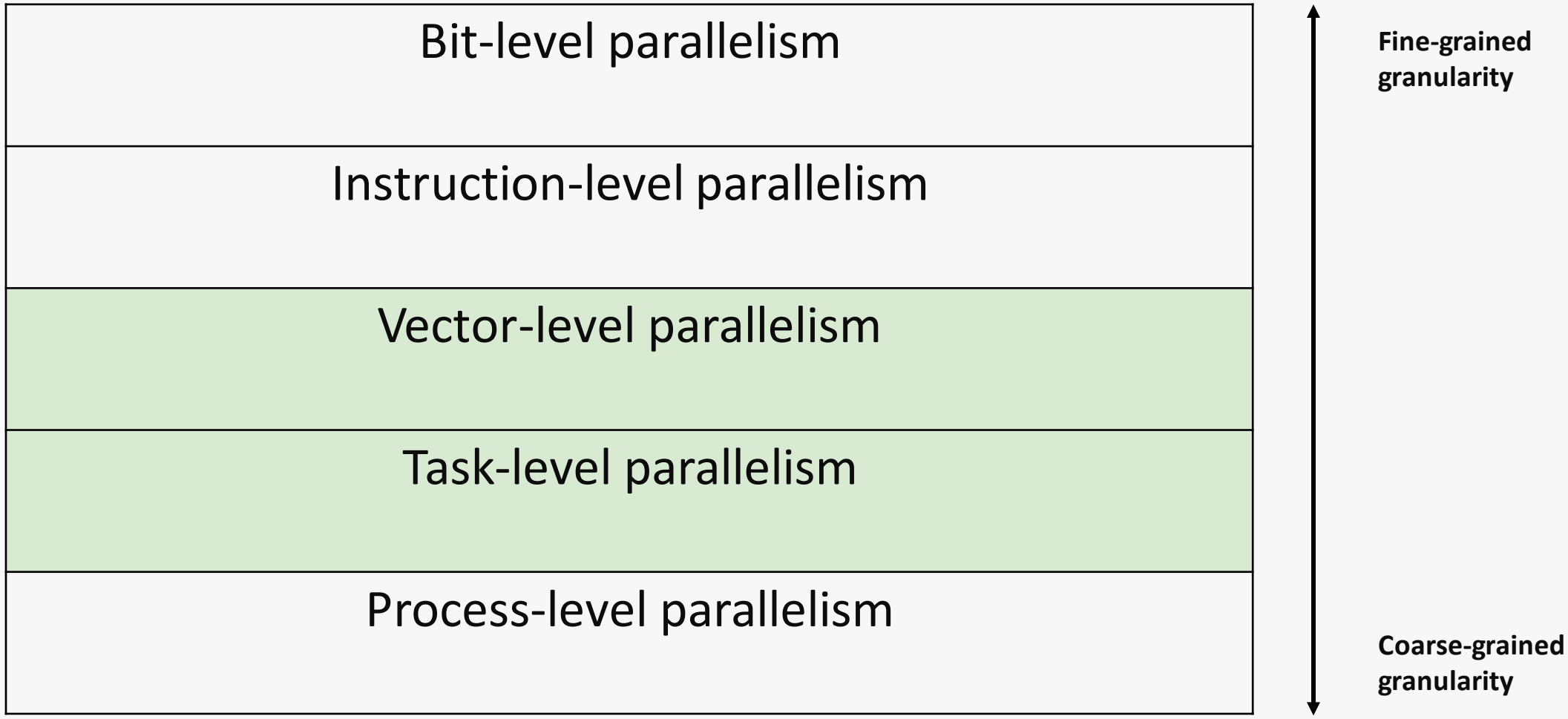
1. A GPU has a region of dedicated global memory
2. Global memory is connected via a bus
3. A GPU is divided into a number of compute units
4. Each compute unit has dedicated local memory
5. Each compute unit has a number of processing elements
6. Each processing element has dedicated private memory

Parallelism vs concurrency

The different kinds of parallelism



The different kinds of parallelism



The challenge of parallel computing

Parallel computing is about breaking up a problem into smaller tasks and having multiple processors working together to solve a problem

The challenge comes in constructing those tasks in such a way that they:

- Make efficient use of the available hardware
- Adhere to the benefits and limitations of the hardware
- Coordinate effectively to complete the work
- Maintaining the correctness of your application

Key takeaways

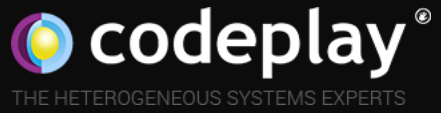
The clock speeds of CPUs are not getting any faster, so gaining further performance must come from parallelism

Sequential code alone uses a very small fraction of the available resources of a typical chip

The challenge of parallel computing is efficiently coordinating tasks across multiple processes in order to solve a problem

CPUs are optimized for latency and GPUs are optimized for throughout

When programming for a co-processor such as a GPU you must compile and offload functions and manage data movement



Questions?