HPC Architecture Basics

Introduction to High Performance Computing 2021

Dr. Tim Cramer

Levels of parallelism in hardware

- **A modern supercomputer may contain multiple levels of parallelism**
  - → Processor level parallelism: Superscalar, SIMD
  - → Node/Chip level: Several cores/processors run in parallel with access to the same memory
  - → System level: Several nodes run in parallel and are communicating over a network interconnect

fine

level

coarse

- **Parallelism introduces overhead**
  - → Additional computational costs (cycles)
  - → Implementation (hours of work)
- **Overhead increases from processor to system level**

Agenda

Cluster → Node → Core → Accelerator

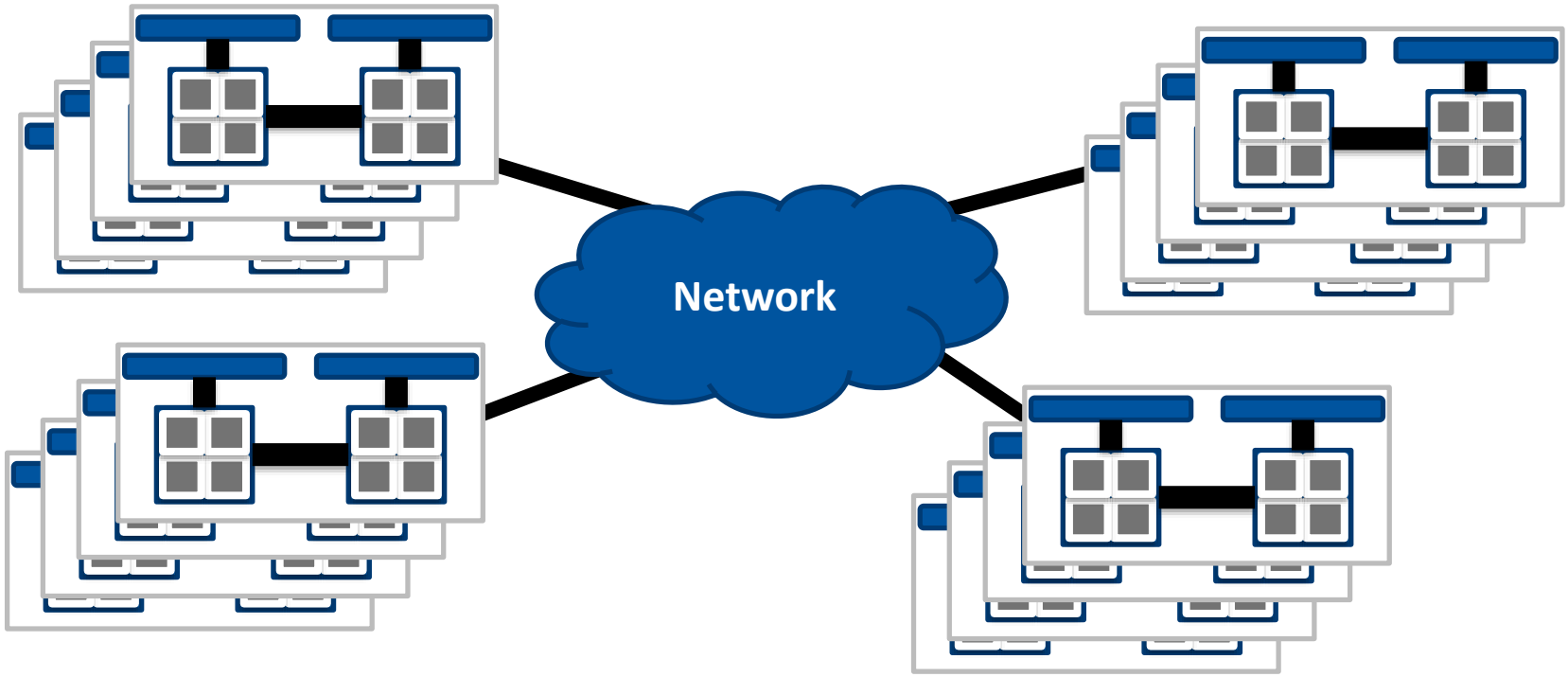
What is a Cluster?

Motivation

Clusters

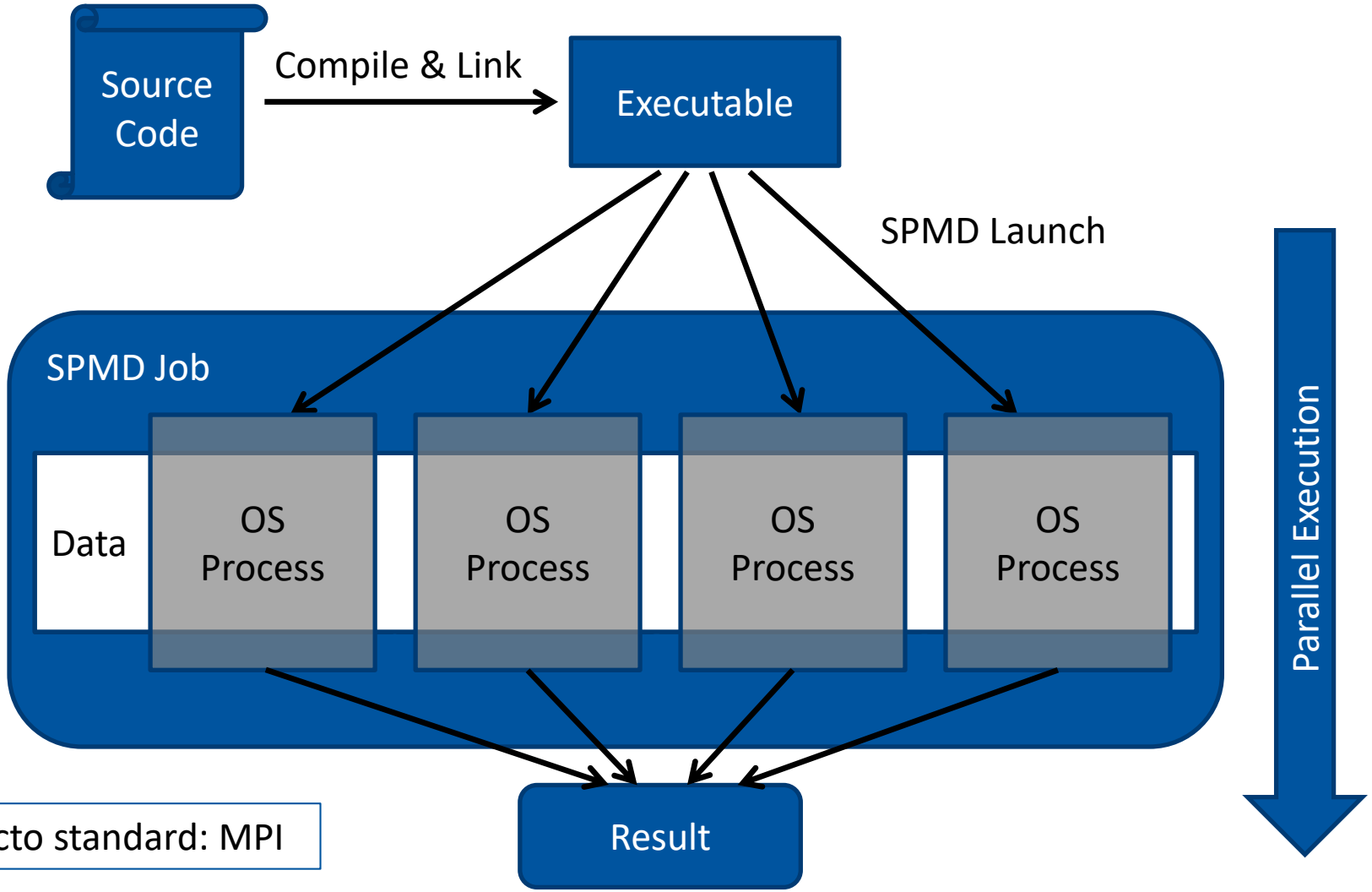
→ HPC market is dominated by distributed memory multicomputers (clusters)

→ Many nodes with no direct access to other nodes' memory



SPMD – Program Lifecycle

SPMD: Single Program Multiple Data

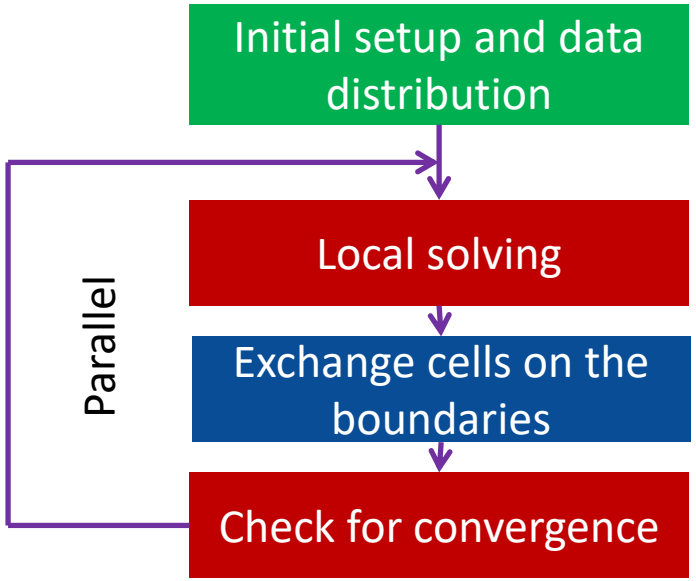


De-facto standard: MPI

System level parallelism

- **Example: Domain decomposition in CFD: Mapping of 3D mesh to the processors**

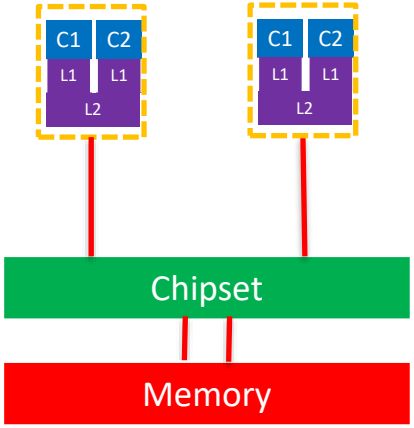
- **Programming techniques**
  - → Data parallel approach
    - → Distribute data structures
  - → Parallel algorithms
  - → Explicit data exchange (MPI)



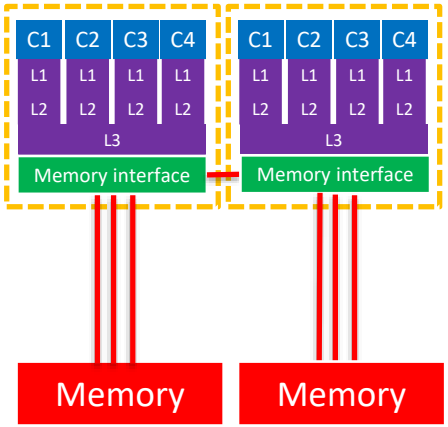
What is a Node in a Cluster?

Node in a Cluster

- **A node may contain**
  - → One or more (multi-core) processors
  - → Memory hierarchy (caches, disks, etc.)
  - → Interconnects, power supply, fans, …
  - → Accelerators



- **Multicore Designs**
  - → Early multicore design
    - → Uniform Memory Architecture (UMA)
    - → Flat Memory design
  - → Recent multicore design
    - → ccNUMA (Cache Coherent Non-Uniform Memory Architecture)
    - → Memory Interface + HT/QPI provides inter-socket connectivity



Worksharing

- **Distribution of work among threads**
  - → No need for explicit message passing (same memory)
- **Most common use case: *for* loop**
  - → Example: OpenMP

C/C++

```
int i;
#pragma omp parallel for
for (i = 0; i < 100; i++)
{
   a[i] = b[i] + c[i];
}
```

  - → Distribution of loop iterations over all threads in a team.
  - → Scheduling of the distribution can be influenced.
- **Loops often account for most of a program‘s runtime!**

Worksharing illustrated

Pseudo-Code
Here: 4 Threads

Serial

```
do i = 0, 99
    a(i) = b(i) + c(i)
end do
```

Thread 1

```
do i = 0, 24
    a(i) = b(i) + c(i)
end do
```

Thread 2

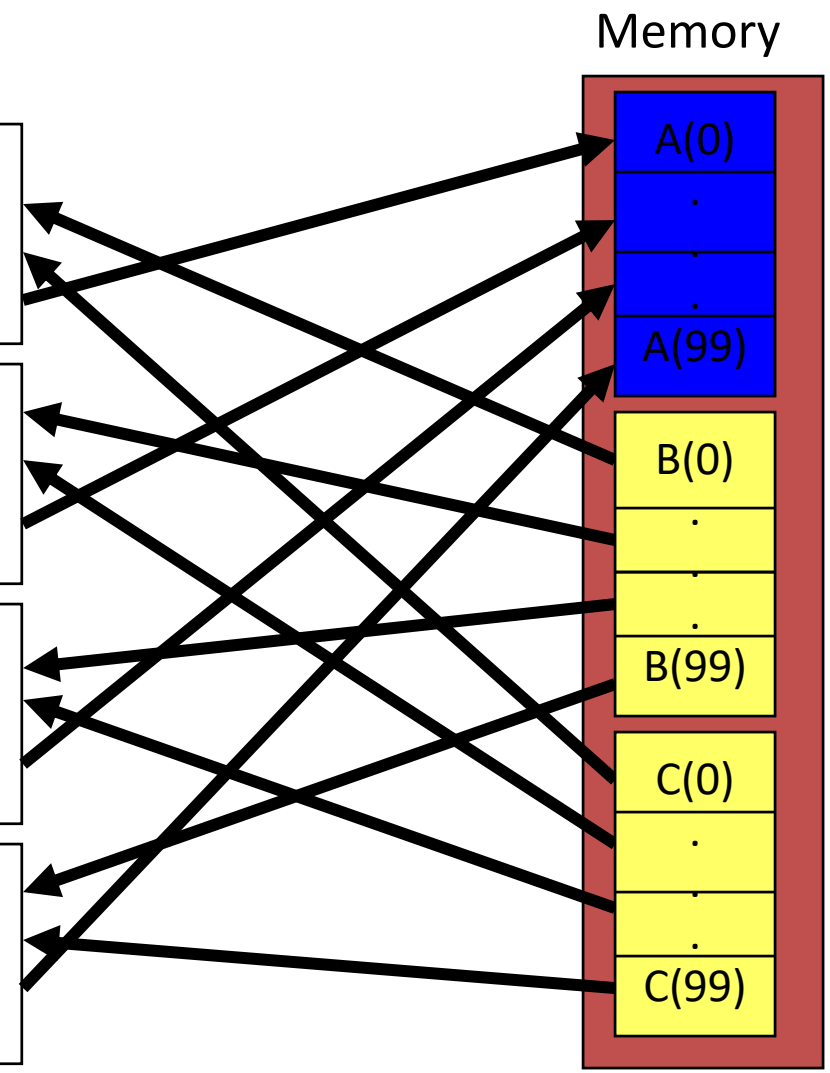
```
do i = 25, 49
    a(i) = b(i) + c(i)
end do
```

Thread 3

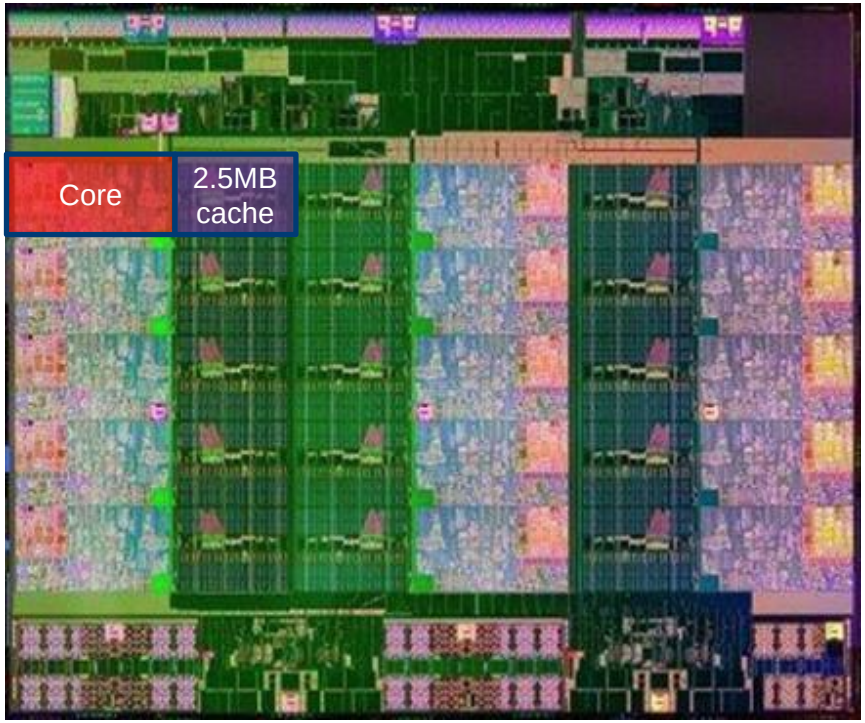
```
do i = 50, 74
    a(i) = b(i) + c(i)
end do
```

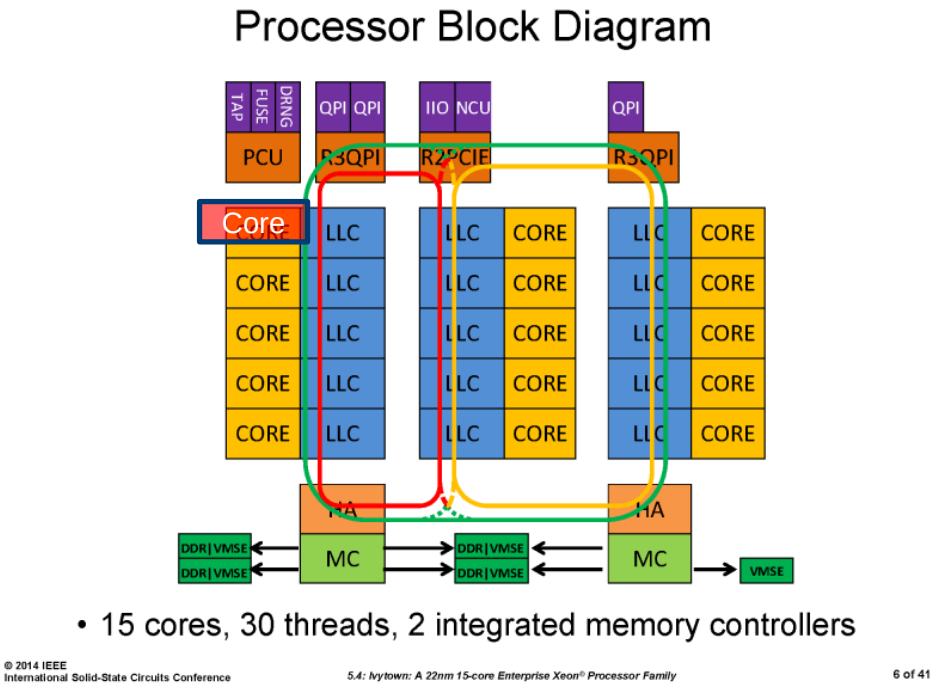
Thread 4

```
do i = 75, 99
    a(i) = b(i) + c(i)
end do
```



What is a Core?





- 15 cores, 30 threads, 2 integrated memory controllers

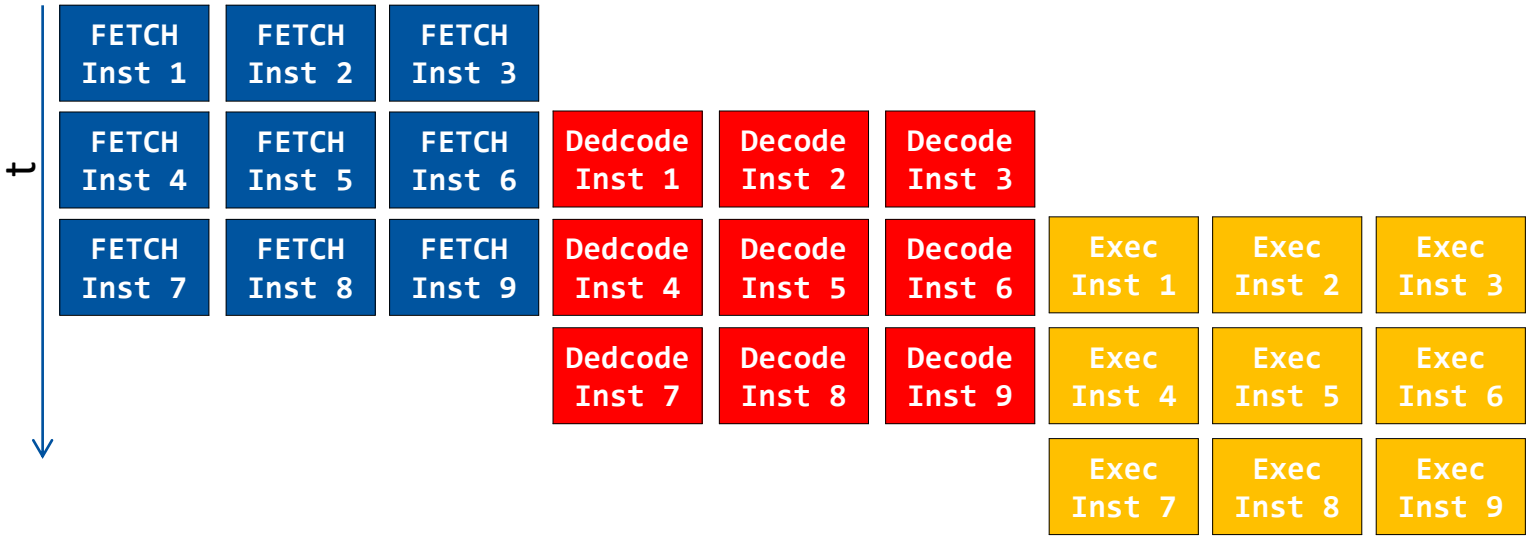
Processor level parallelism

- **Parallelism at processor/ instruction level**
  - → Pipelining (overlap in execution: fetch, decode, execute)

Clock cycle	1	2	3	4	5	…
Fetch Instruction	I(1)	I(2)	I(3)	I(4)	I(5)	…
Decode Instruction		I(1)	I(2)	I(3)	I(4)	…
Execute Instruction			I(1)	I(2)	I(3)	…

Processor level parallelism

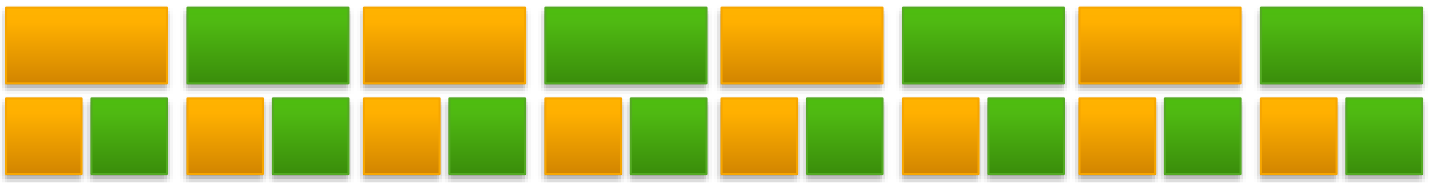
- **Parallelism at processor/ instruction level**
  - → Pipelining (overlap in execution: fetch, decode, execute)
  - → Superscalar (redundant arithmetical units: Multiplication, Addition, …)
    - → Example: 3-way superscalar



Processor level parallelism

Length of a double: 8 bit
Length of a float: 4 bit

- **Parallelism at processor/ instruction level**
  - → Pipelining (overlap in execution: load, decode, execute)
  - → Superscalar (redundant arithmetical units: Multiplication, Addition, …)
  - → SIMD execution (e.g. 512 bit registers, AVX-512)



8 x double

16 x float

- **Programming techniques**
  - → Code modifications: Unrolling, Cache reuse
  - → Compiler optimizations

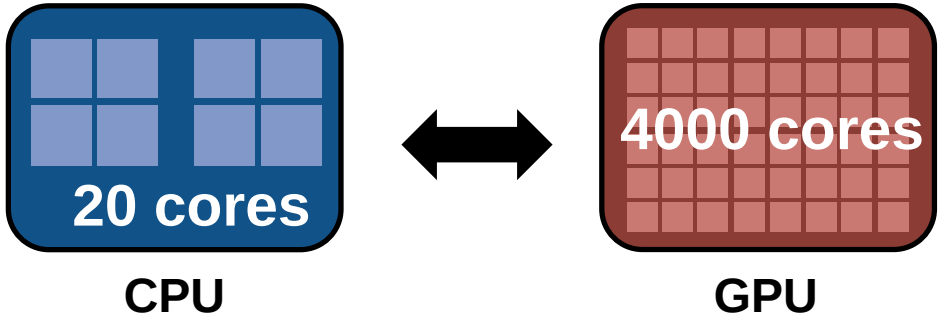
What is a Accelerator in a Node?

GPU Programming: Overview

- **GPGPUs = General Purpose Graphics Processing Units**

- **History – a very brief overview**
  - → ‘80s - ‘90s: Development is mainly driven by games
    Fixed-function 3D graphics pipeline
    Graphics APIs like OpenGL, DirectX popular
  - → Since 2001: Programmable pixel and vertex shader in graphics pipeline (adjustments in OpenGL, DirectX)
    Researchers take notice of performance growth of GPUs: Tasks must be cast into native graphics operations
  - → Since 2006: Vertex/pixel shader are replaced by a single processor unit
    Support of programming language C, synchronization,…
    → “General purpose”

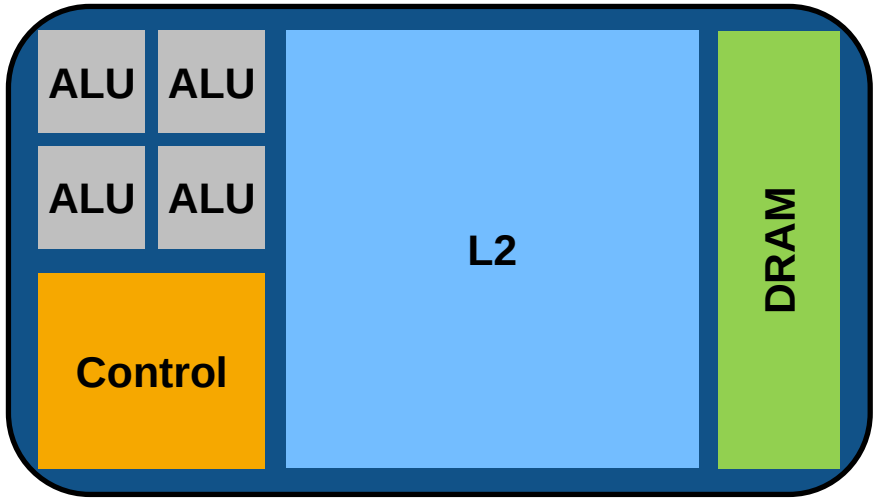
Comparison CPU ⇔ GPU



- **GPU-Threads**
  - → Thousands ("few" on CPU)
  - → Light-weight, little creation overhead
  - → Fast switching
- **Lots of parallelism needed on GPU to get good performance!**

Comparison CPU ⇔ GPU

- **Different design**



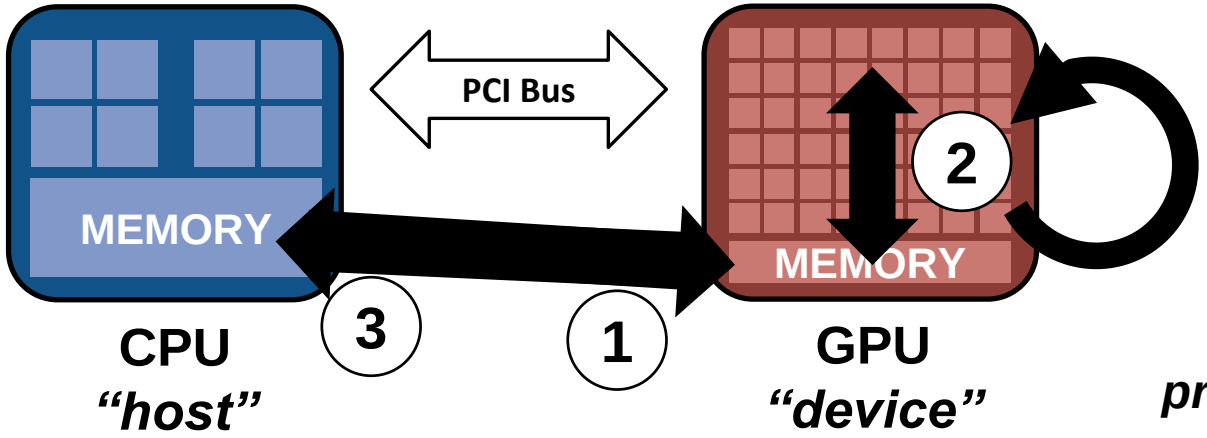
CPU

→ Optimized for low latencies
→ Huge caches
→ Control logic for out-of-order and speculative execution
→ **Targets on general-purpose applications**

GPU

→ Optimized for data-parallel throughput
→ Architecture tolerant of memory latency
→ More transistors dedicated to computation
→ **Suited for special kind of apps**

Offloading



We refer to "discrete GPUs" here.

- **Weak memory model**
  - → Host + device memory = separate entities
  - → No coherence between host + device
    - → Data transfers needed
- **Host-directed execution model**
  - → Copy input data from CPU mem. to device mem.
  - → Execute the device program
  - → Copy results from device mem. to CPU mem.

processing flow *(simplified)*