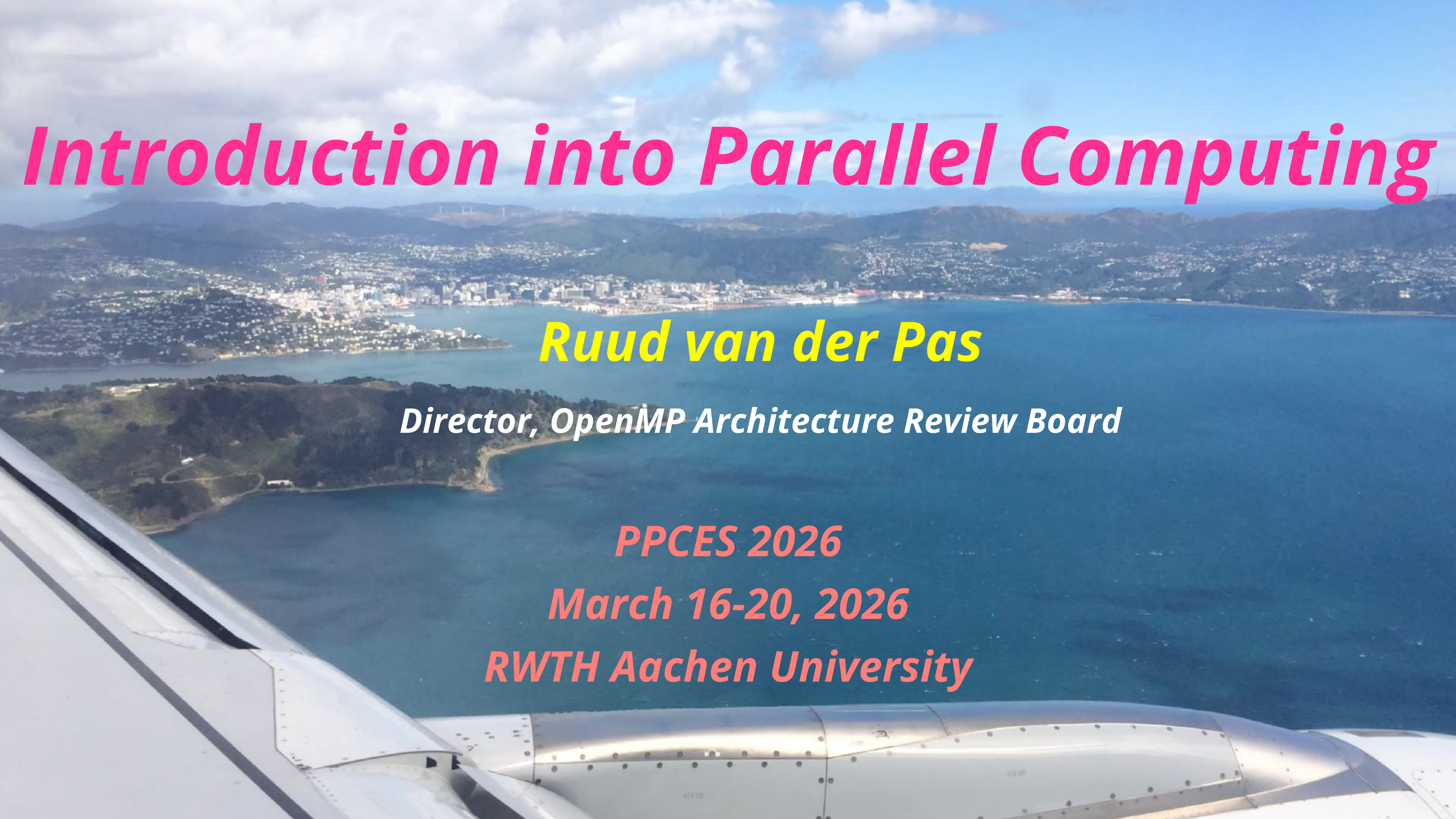


Introduction into Parallel Computing

Ruud van der Pas

Director, OpenMP Architecture Review Board

PPCES 2026

March 16-20, 2026

RWTH Aachen University

$ whoishe

My background is in mathematics and physics

Previously, I worked at Philips Electronics, the University of Utrecht, Convex Computer, SGI, Sun Microsystems, and Oracle

I have been involved with OpenMP since the introduction

I am passionate about performance and OpenMP in particular

About this Talk

A seemingly random collection of topics

The common element is Parallel Computing

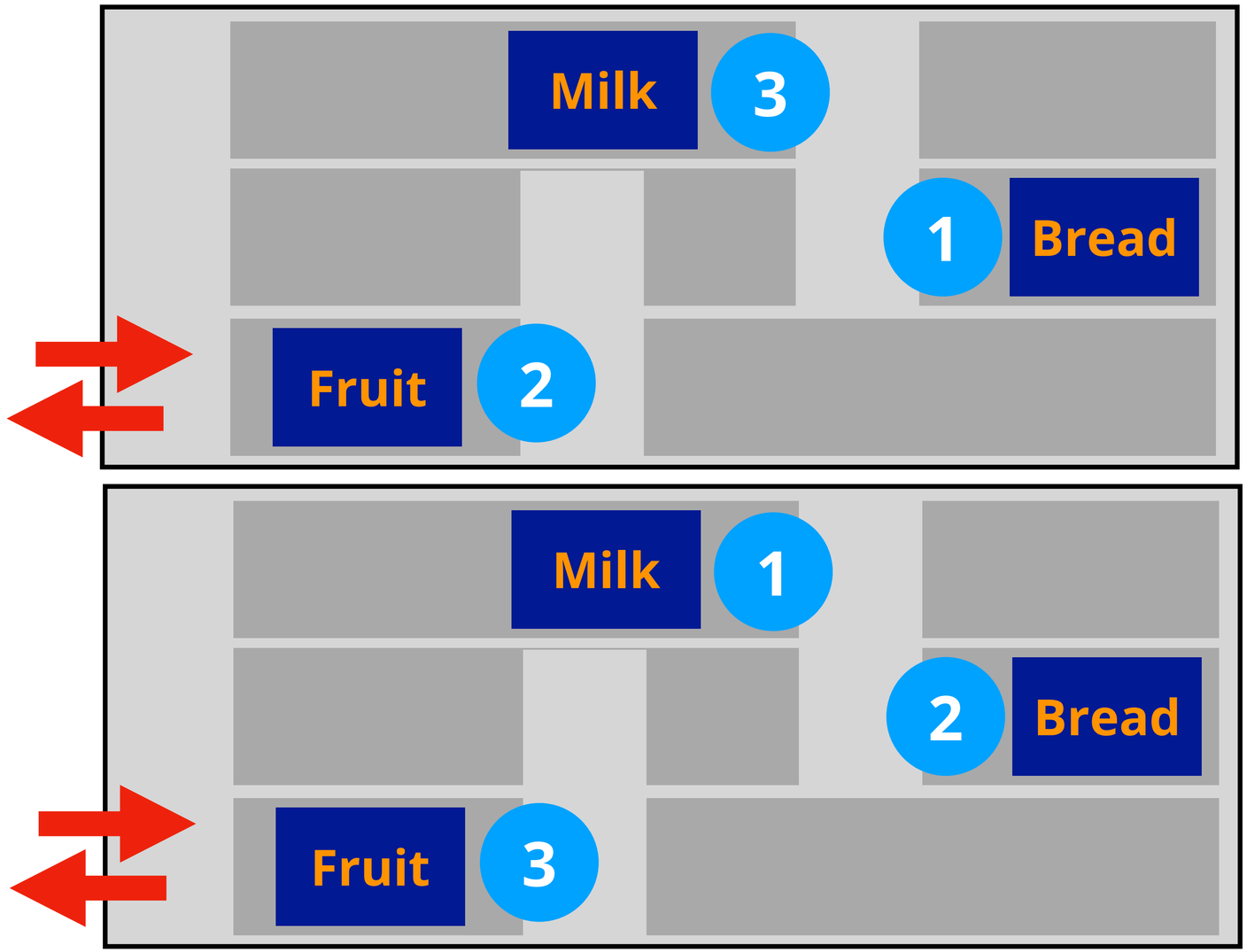
The Topics

- *What is Parallelism?*
- *What is Parallel Computing?*
- *Various Concepts in Parallel Computing*
- *Parallel Architectures*
- *Parallel Programming Models*
- *Common Mistakes in Parallel Applications*

What is Parallel Computing?

A Trip to Ruud's Supermarket

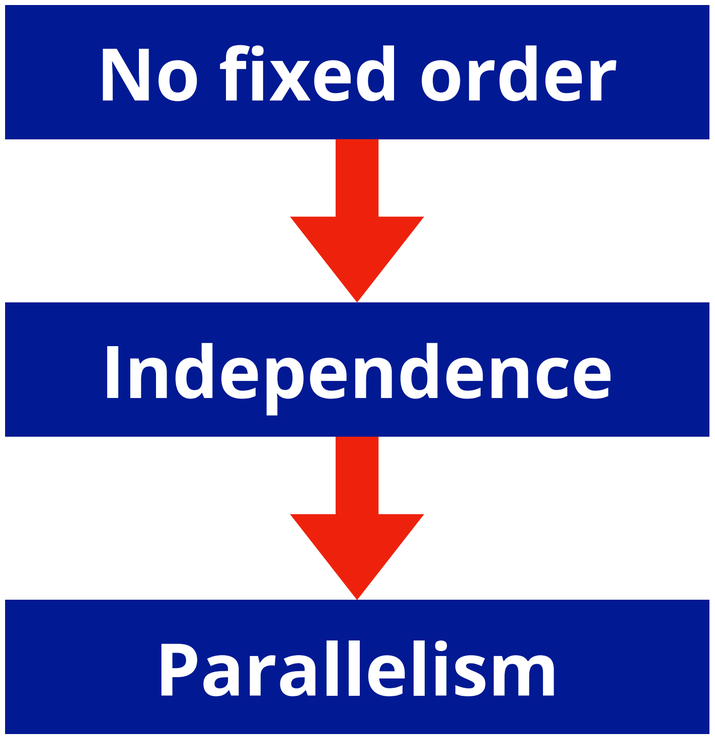
My shopping list
- **Bread**
- **Fruit**
- **Milk**



The order does not matter

In both cases, my final shopping basket is the same

What is Parallelism?




```
a = 2 * b
c = 3 + d
```

```
c = 3 + d
a = 2 * b
```

```
c = 3 + d    a = 2 * b
```

The Goal of Parallel Computing

Exploit the parallelism in an application

Reduce the time to solve a problem

To achieve this

Use multiple (computational) resources to solve a single problem

Examples of computational resources

cores, an entire system, multiple, or even all systems

Parallel Granularity*

Multiple instructions

A collection of program statements

Calls to functions or subroutines

A larger part of your program

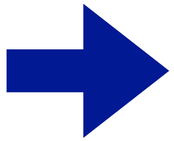
Granularity increases

*) A granule is a small particle of a substance, like a granule of sugar

Instruction Level Parallelism*

A total of 8 floating-point instructions

```
for (i=0; i<3; i++)
  {
    a[i] = 2 * b[i];
    c[i] = 3 + d[i];
  }
```

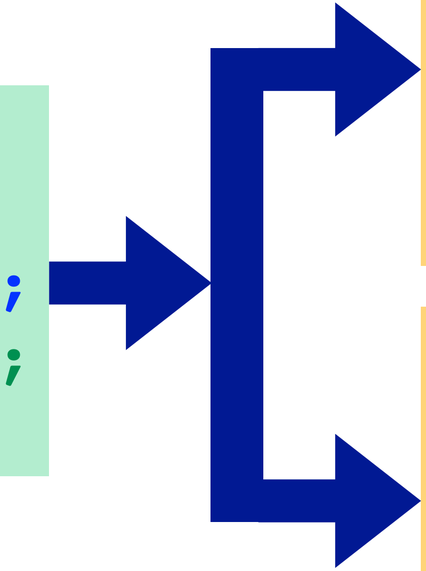


```
for (i=0; i<3; i++)
  {
    fmul 2,b[i],a[i];fadd 3,d[i],c[i]
  }
```

) This depends on what the core supports, if anything.

Single Instruction Multiple Data - SIMD*

```
for (i=0; i<3; i++)
  {
    a[i] = 2 * b[i];
    c[i] = 3 + d[i];
  }
```



Vectorization

```
a[0] = 2 * b[0]
a[1] = 2 * b[1]
a[2] = 2 * b[2]
a[3] = 2 * b[3]
```

```
c[0] = 3 + d[0]
c[1] = 3 + d[1]
c[2] = 3 + d[2]
c[3] = 3 + d[3]
```

A total of 2 floating-point instructions

```
vfmul 2,b,a
```

```
vfadd 3,d,c
```

**) This depends on what the core supports, if anything.*

How to Parallelize an Application?

Select a parallel programming model

Posix Threads, Java Threads, OpenMP, MPI, ...

Use the features to express parallelism in the application

Yes, this is what you need to do

This results in a parallel application

If you did this correctly, you will enjoy improved performance

The Challenges of Parallel Computing

Find the parallelism in the application

Focus on the most time consuming part

Make sure to get the correct answer

This turns out to be quite the challenge

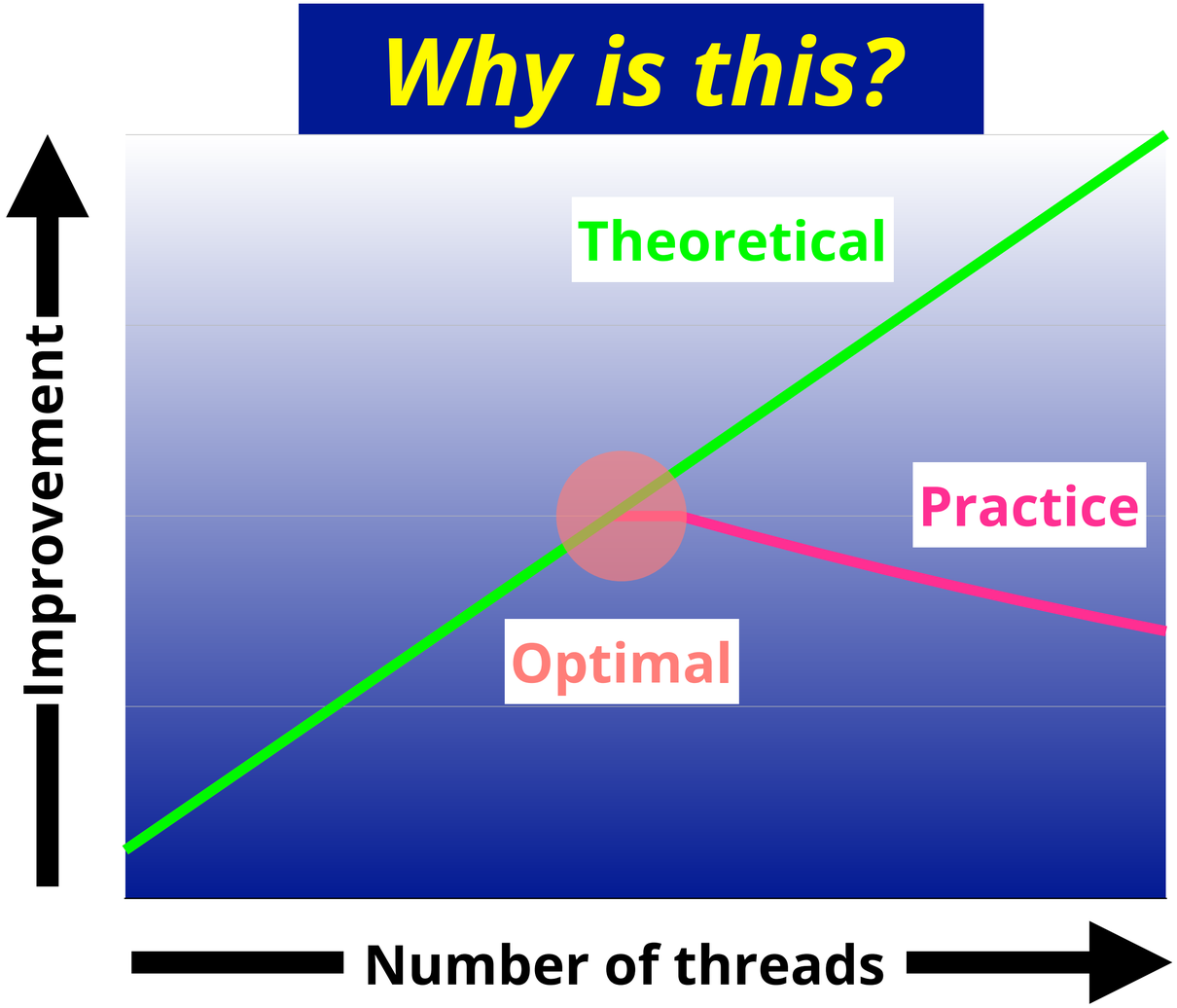
Make the application perform ("scale") well

This is where Amdahl's Law* comes in

***) No, Amdahl's Law is not another EU regulation on the volume of vacuum cleaners. It is a Computer Science thing.**

Amdahl's Law

Parallel Speed Up - How Much Faster?



Serial versus Parallel

T(total) = T(serial) + T(parallel)

The part of the application that has not been parallelized is called the serial, single threaded, or sequential, part

As we shall see next, one of the goals of efficient parallelization is to keep the serial part as short as possible

Amdahl's Law - An Example

Suppose an application needs 100 seconds to run

Assume that 80% of that time can be executed in parallel

The time using 4 threads is then:

80/4 + 20 = 20 + 20 = 40 seconds

This means that the performance improvement is 2.5x, not 4x

Amdahl's Law - The Formula*

Suppose that you have parallelized a fraction "f" of the run time

*Split the single thread time in two parts: T(1) = f*T(1) + (1-f)*T(1)*

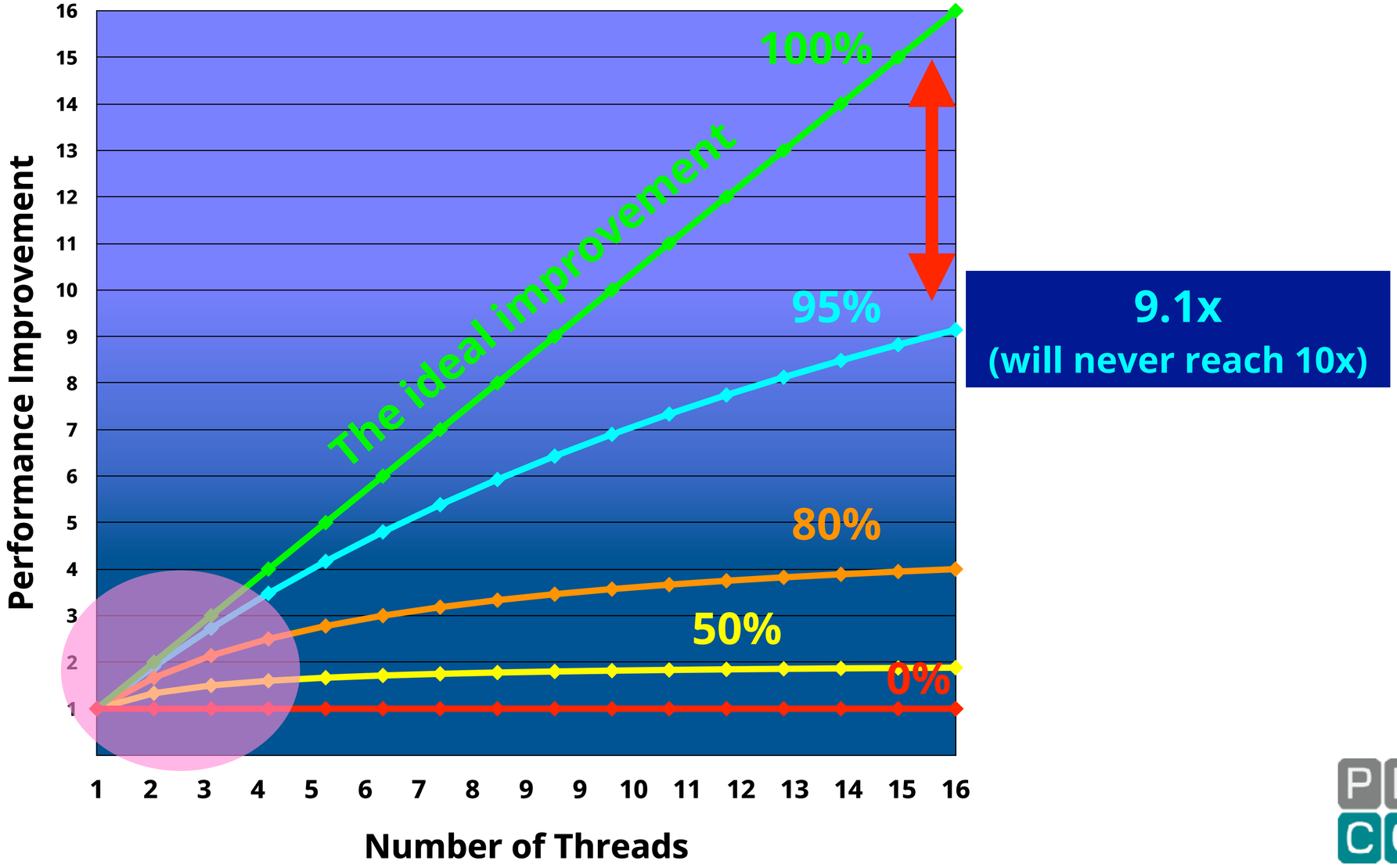
*On P threads: T(P) = f*T(1)/P + (1-f)*T(1) = (f/P + 1-f)*T(1)*

Amdahl's Law: S(P,f) = T(1)/T(P) = 1/(f/P + 1-f)

Example for f = 0.8: S(4,0.8) = 1/(0.8/4 + 0.2) = 2.5

*) This is a simplification - The parallel overhead is ignored, often causing the estimate to be optimistic

Amdahl's Law Using 16 Threads



9.1x
(will never reach 10x)

Morale

Amdahl's Law shows that you need to parallelize a significant fraction of the run time, in order to see a decent speed up for higher thread counts

This implies that the parallel overhead should be minimal

An important issue is that the serial, single thread, part needs to be minimal: it will dominate sooner than later

About Single Thread Performance and Scalability

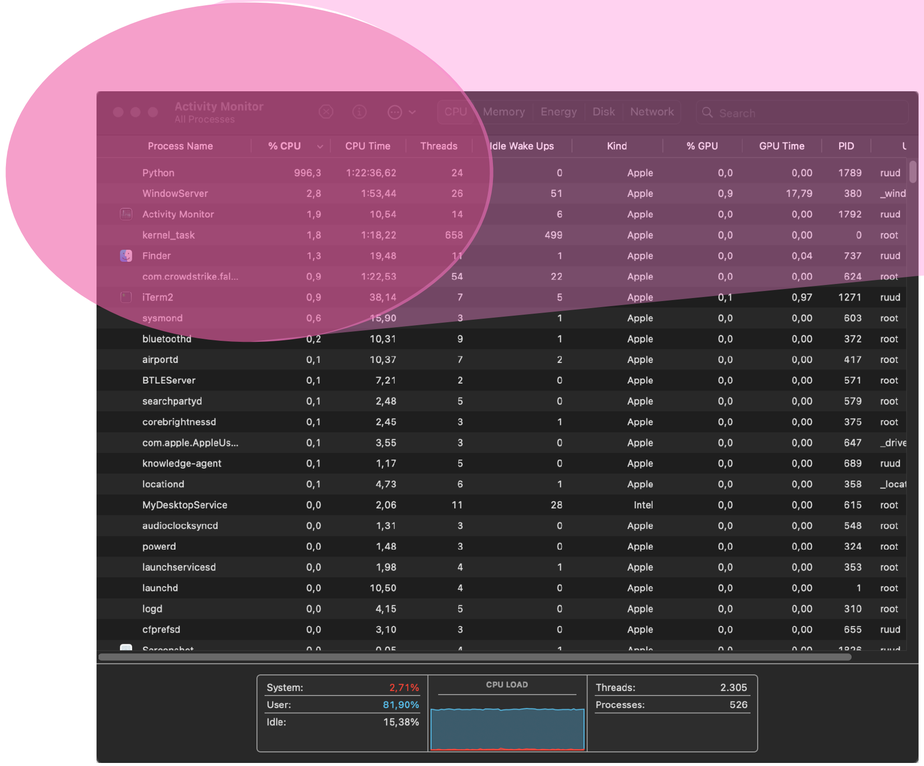
You have to pay attention to single thread performance

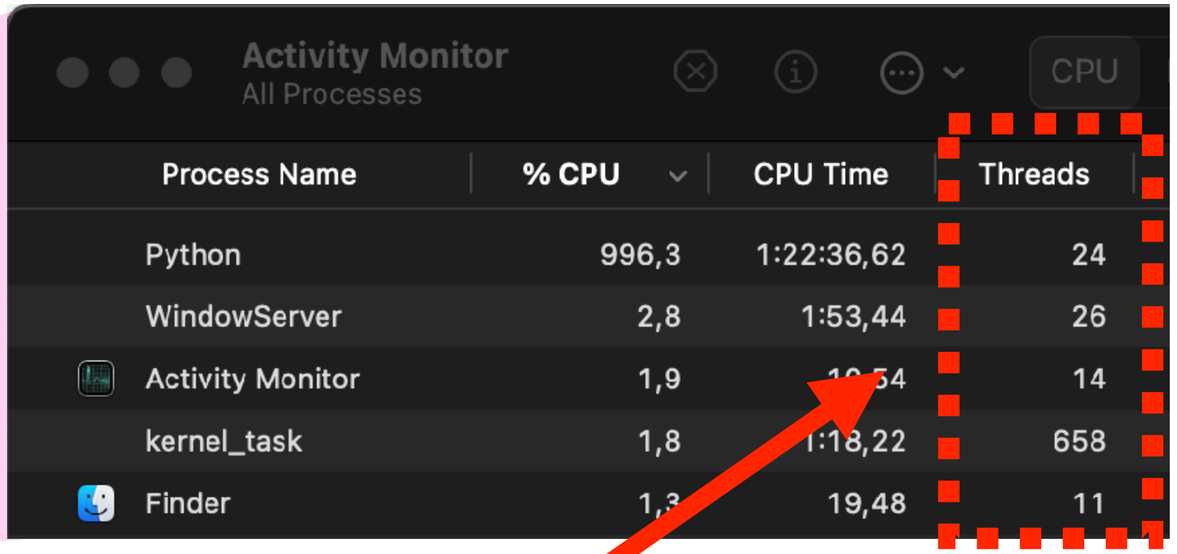
Why? *If your code performs badly on 1 core, what do you think will happen on 10 cores, 20 cores, … ?*

Scalability can mask poor performance!
(a slow code tends to scale better …)

Threads and Multithreading

The Applications Running on My Laptop





What are these "threads" and why are they there?

About Threads and Multithreading

A thread consists of a series of instructions with its own state

The state includes the Program Counter (PC)

This means threads can execute independently of each other

Threads are created and managed in software

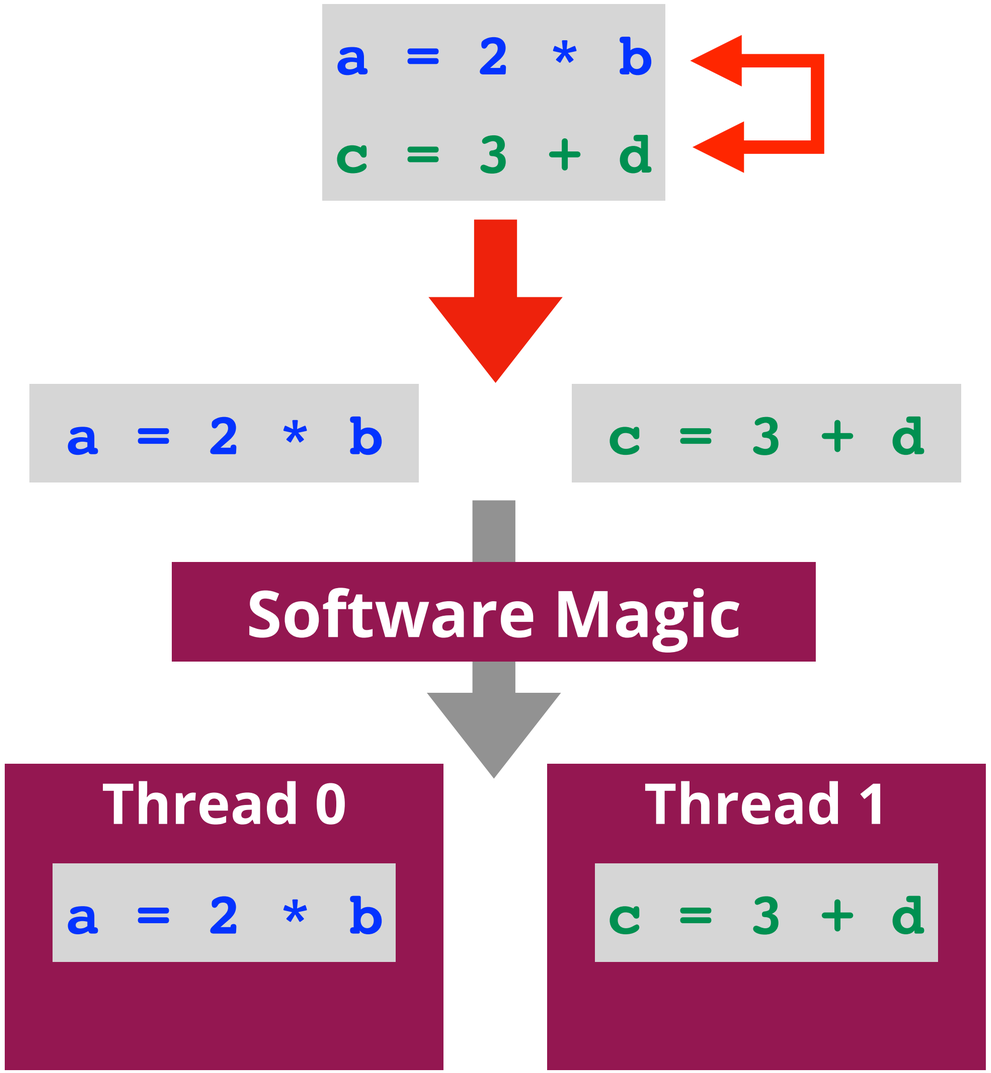
An application that uses multiple threads is said to be multithreaded

Multithreading - Hardware and Software

A multithreaded architecture has multiple independent execution vehicles (e.g. cores, hardware threads, ...)

A multithreaded application creates and manages multiple software threads of execution

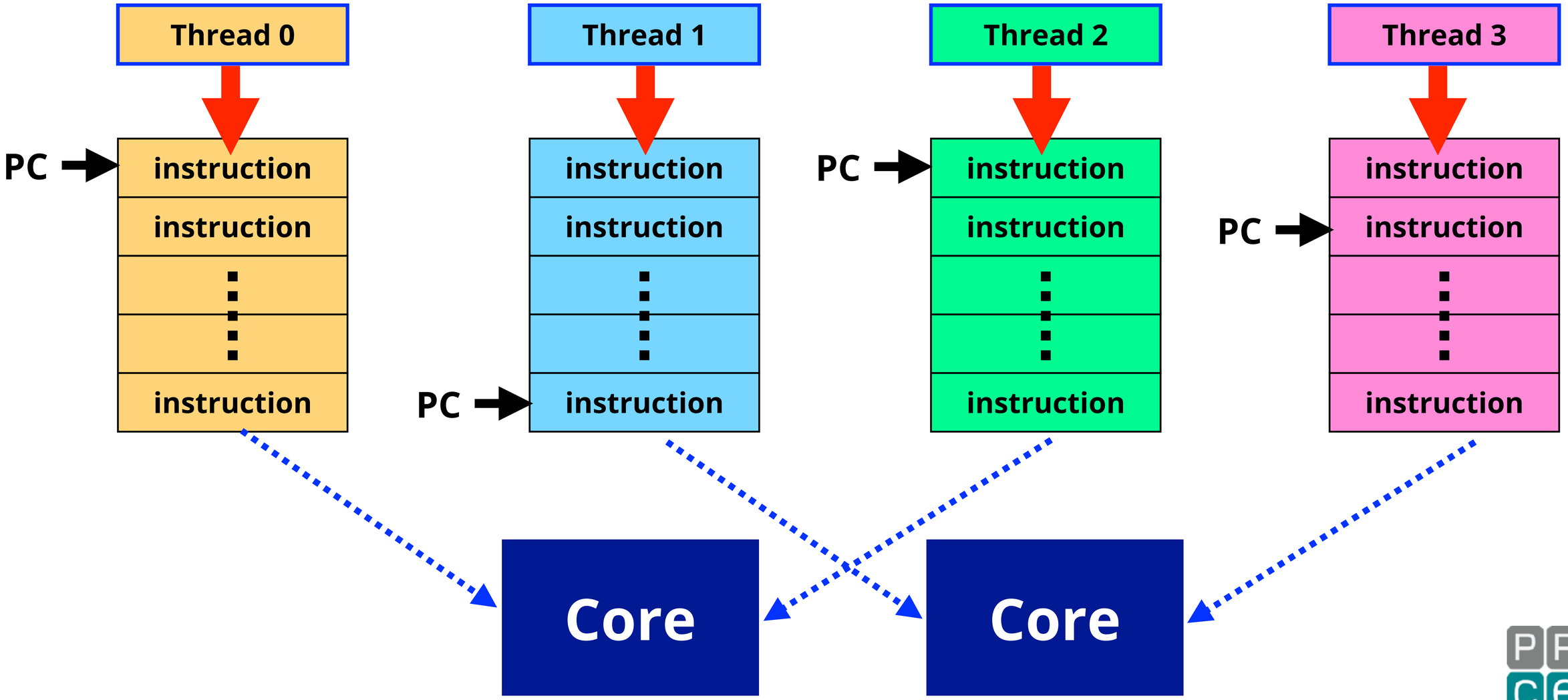
The Example Revisited



Independence

Parallelism

Scheduling - Four Threads at Work on Two Cores



Parallel Overhead

Two Notions of Time

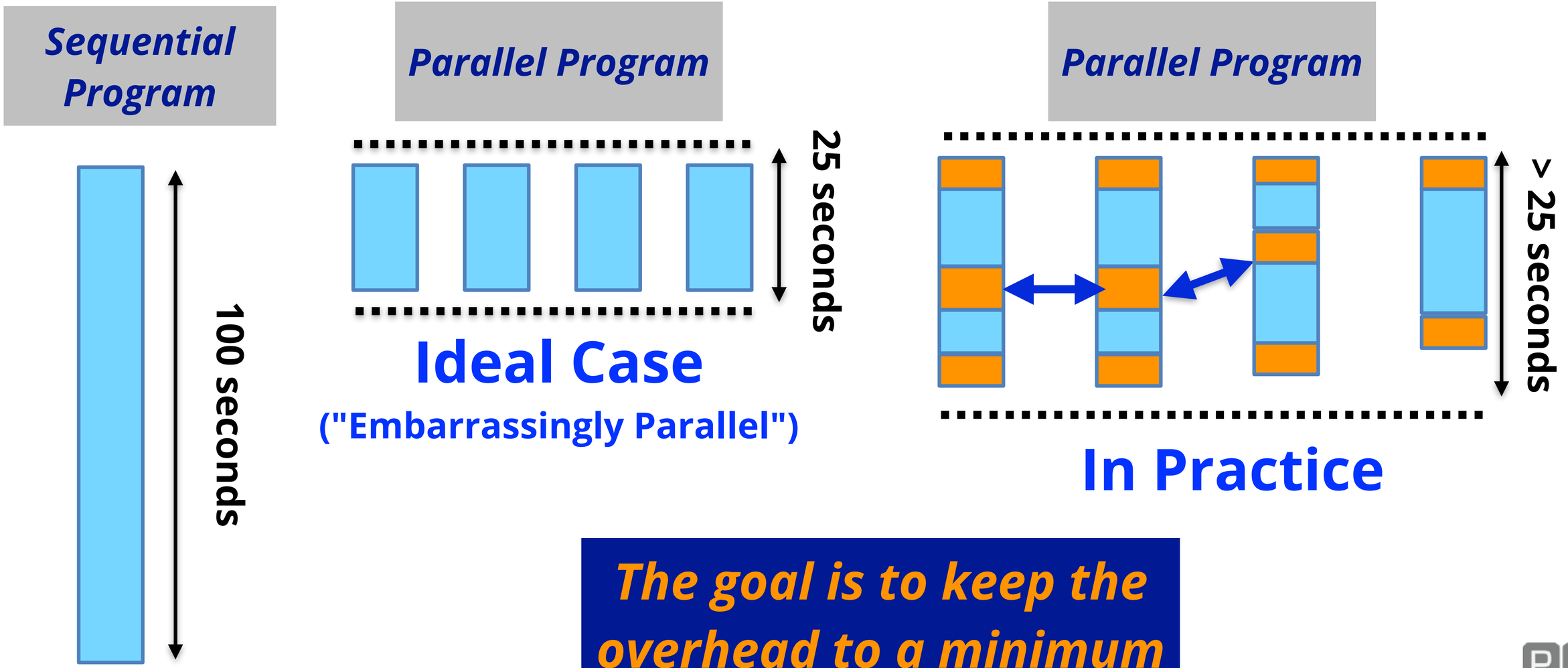
The goal of parallel computing is to reduce the time to solution, usually called the wall clock time, or elapsed time

In doing so, the total CPU time tends to be higher, compared to the sequential version of the application

This is because there is additional code that needs to be executed, often called the (parallel) overhead

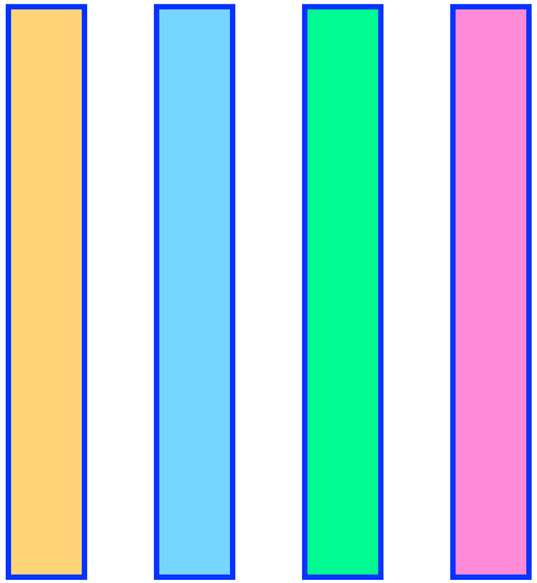
The goal is to write efficient parallel code and keep the overhead to a minimum

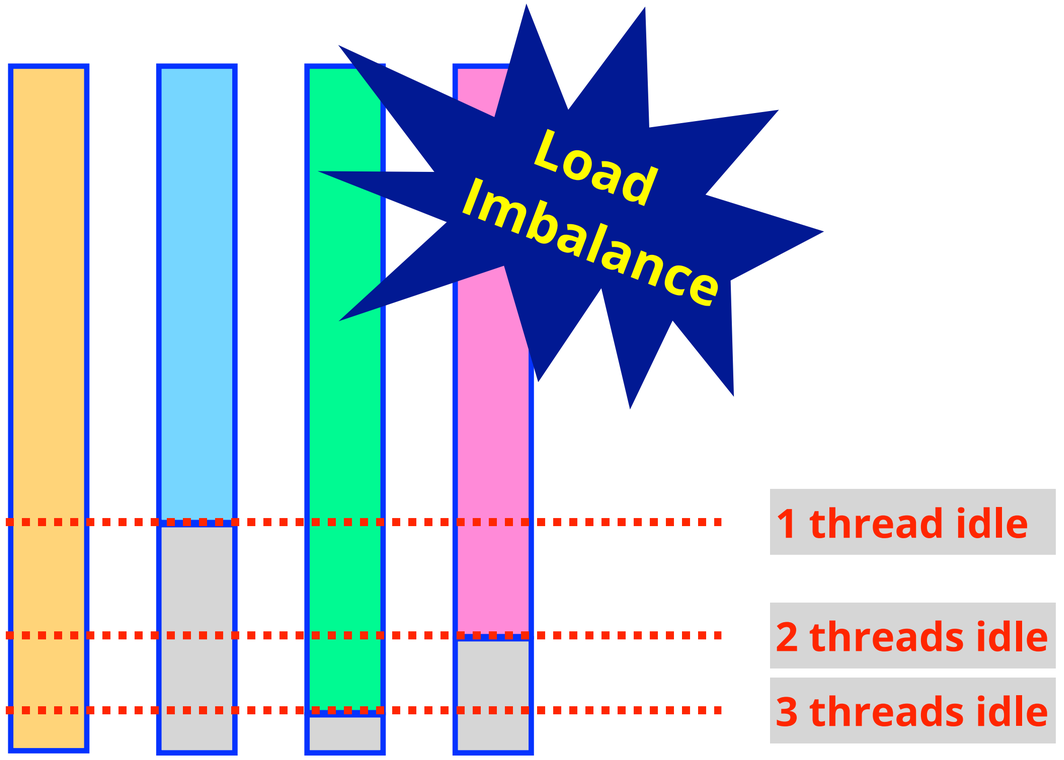
Parallel Overheads



The goal is to keep the overhead to a minimum

Load Balancing





Ideal situation

- All threads start and finish at the same time
- Shortest execution time

Suboptimal situation

- Threads waste time and energy doing nothing
- Longer execution time

= thread is waiting ("idle")

Numerical Results

Numerical Results

Due to roundoff effects, the order of the floating-point computations may affect the results

In parallel computing, the order of operations is non-deterministic ...

```
A = B + C + D + E
```

Sequential Computation

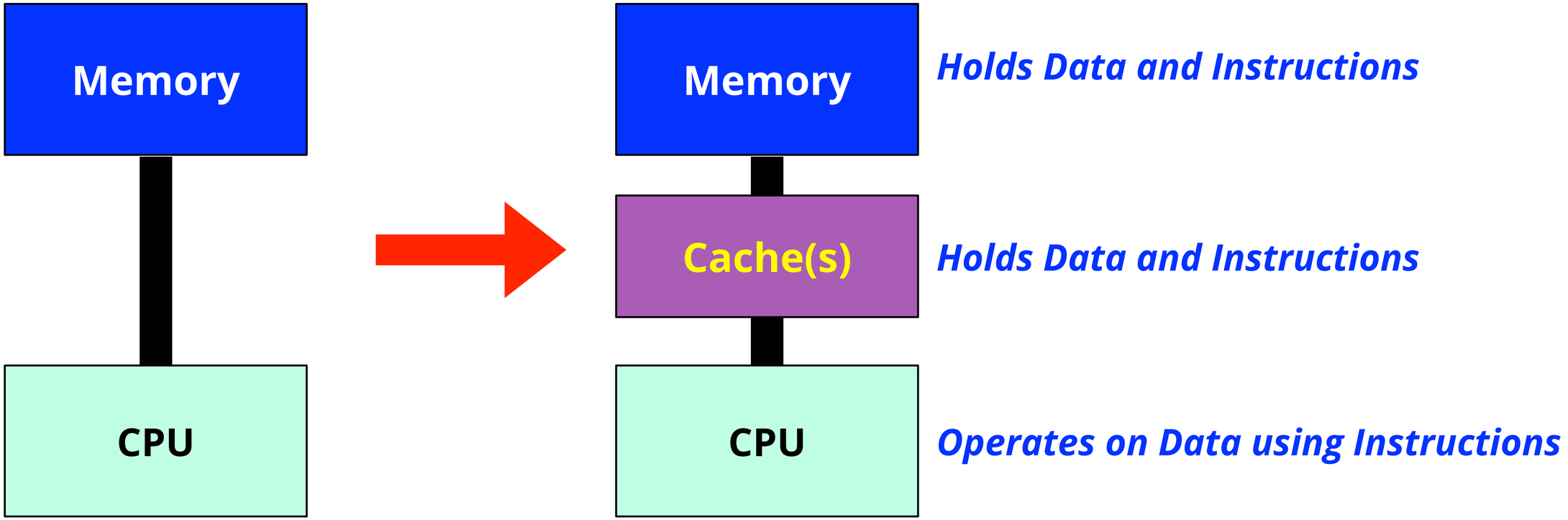
```
A = B + C
A = A + D
A = A + E
```

Parallel Computation

```
T1 = B + C     T2 = D + E
A = T1 + T2
```

Parallel Architectures

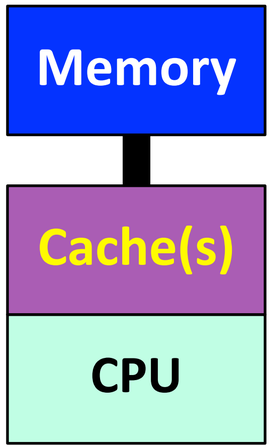
A Computer



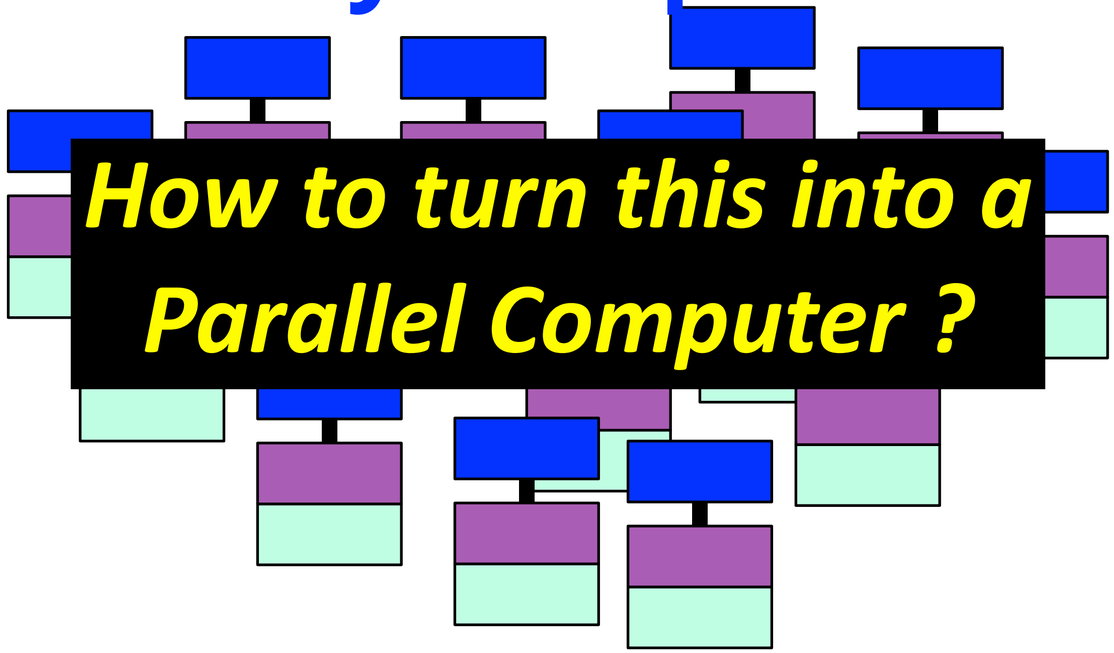
Note: CPU = Central Processing Unit

About Computers

A Computer



Many Computers



Note: CPU = Central Processing Unit

Intermezzo - Cache Coherence

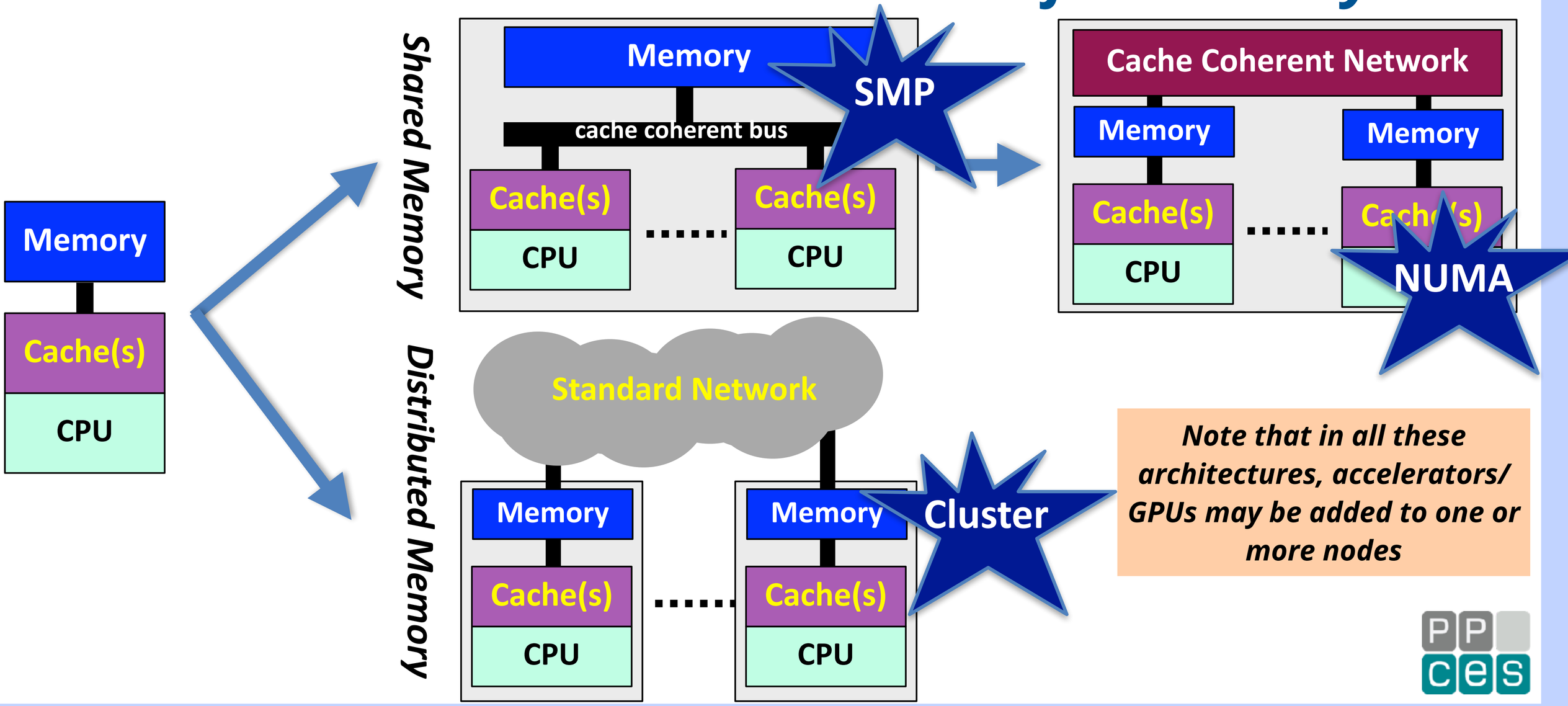
Required in a system with shared memory and caches

In very simple terms, cache coherence ensures that the system knows where data is, and what the coherency state is

The coherency state indicates whether data in a particular location can be used, or not

It allows for transparent parallel programming, since the user does not need to know where data physically resides

Parallel Architectures - World's Briefest History



Note that in all these architectures, accelerators/ GPUs may be added to one or more nodes

Cores and Multicore

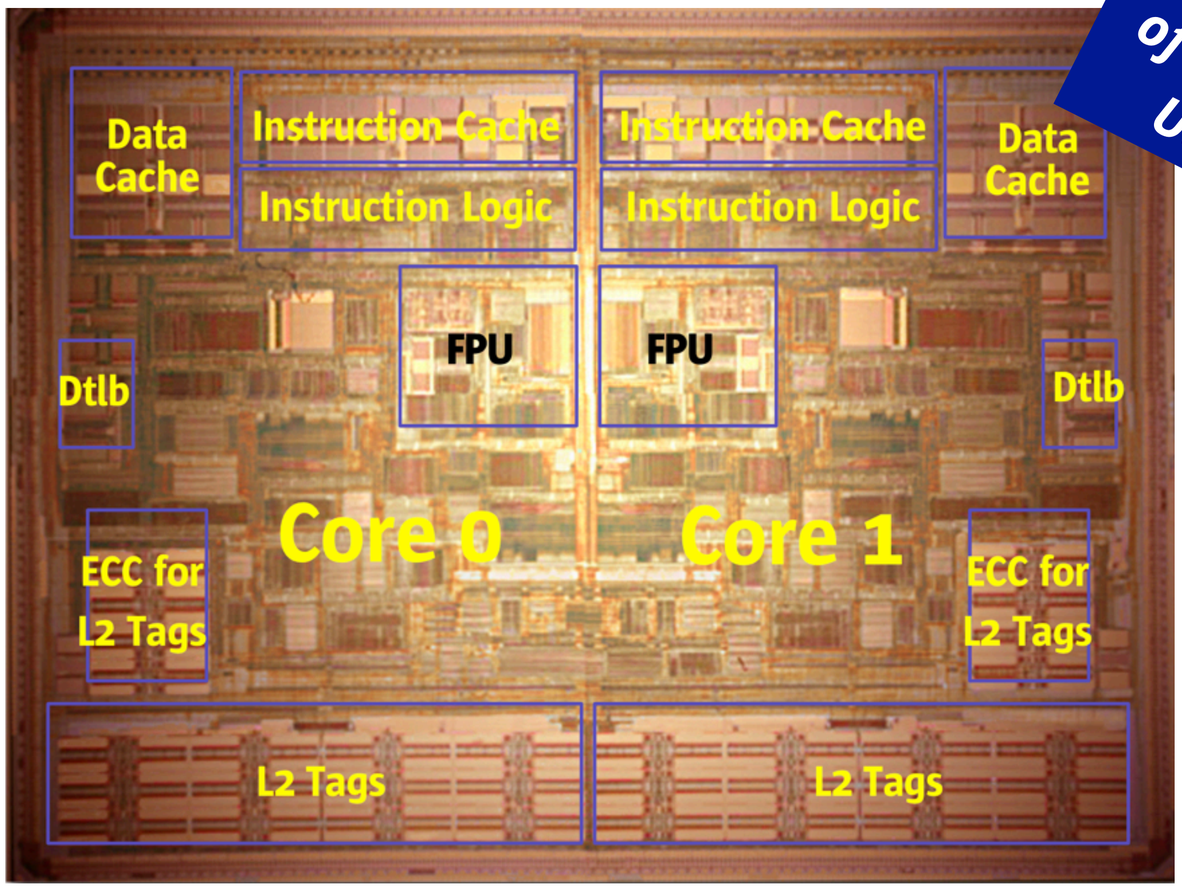
For a long time, the word CPU was used

This stands for Central Processing Unit and is the part of the hardware with the logic controls, computational units, etc.

When multiple processing parts were put on a single chip, the terms core and multicore were introduced

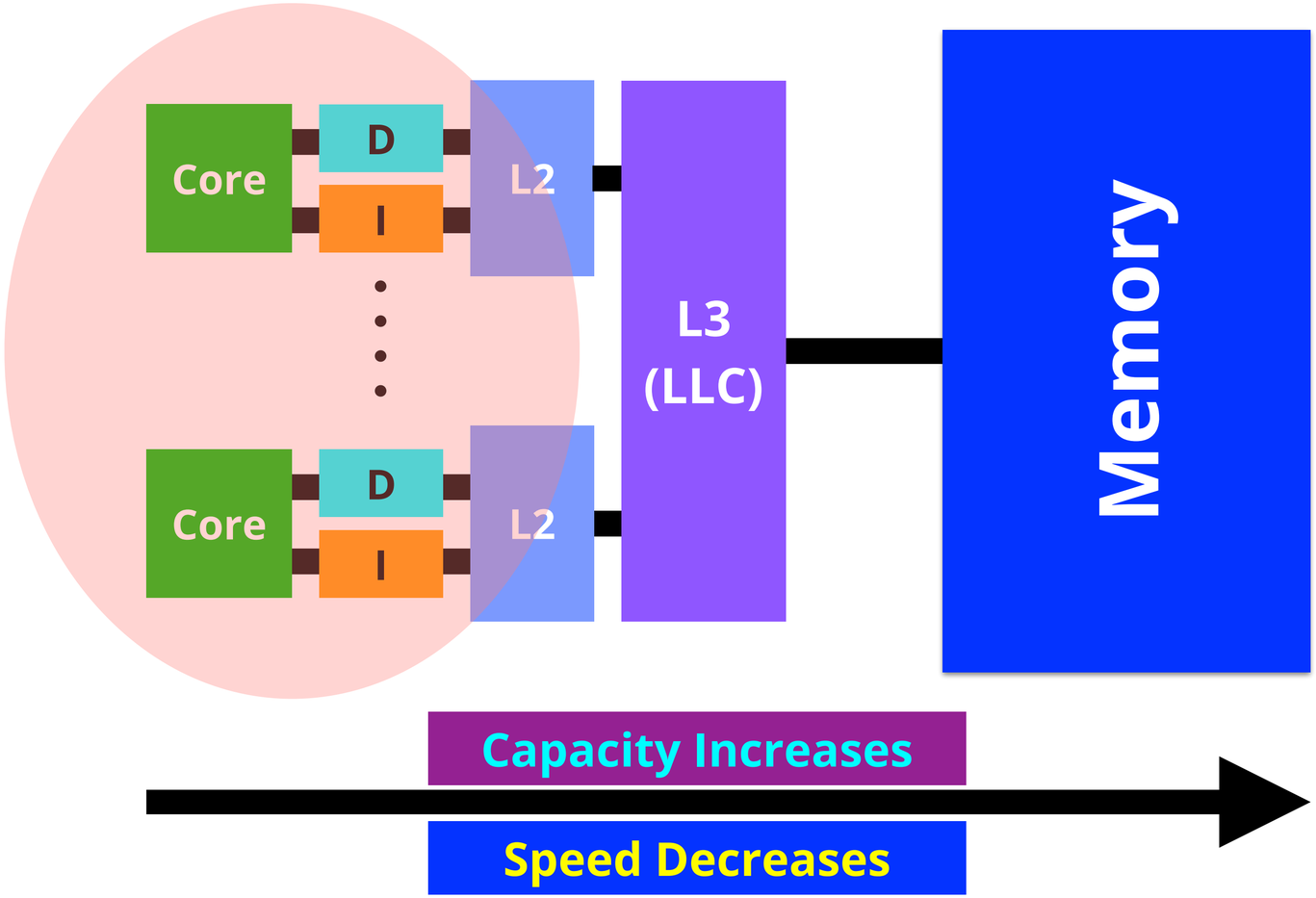
Multicore processors became available around 2004

UltraSPARC IV Dual Core - Introduced 2004



The RWTH Aachen was one of the first to use SPARC US IV based servers!

A Typical Memory Hierarchy



The unit of transfer is a "cache line"

A cache line contains multiple elements

About Cores and Hardware Threads

A core may, or may not, support hardware threads

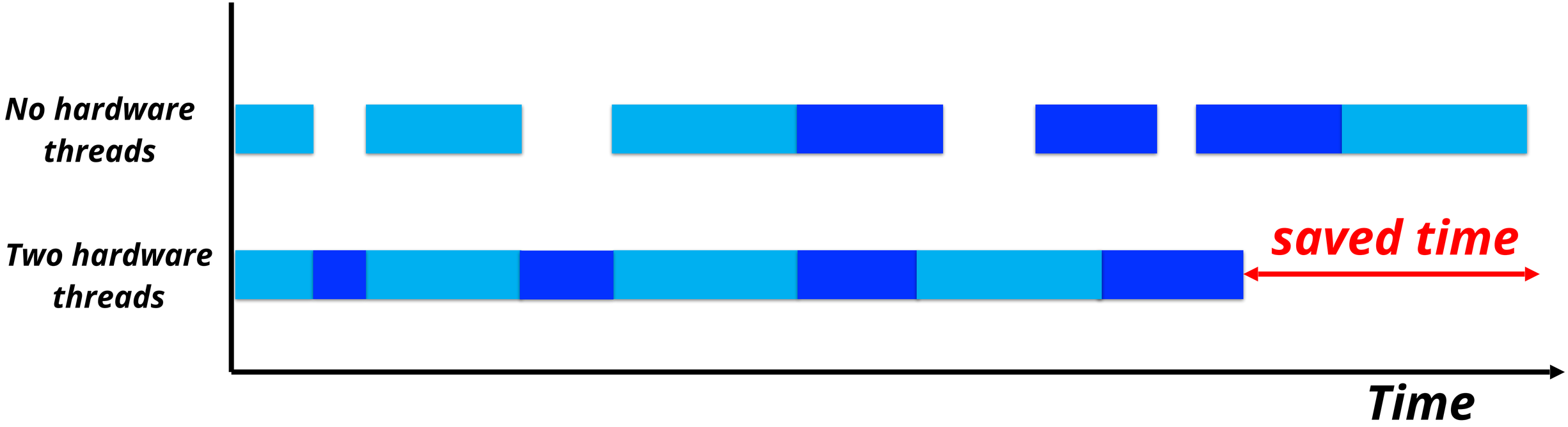
This is part of the design

These hardware threads may accelerate the execution of a single application, or improve the throughput of a workload

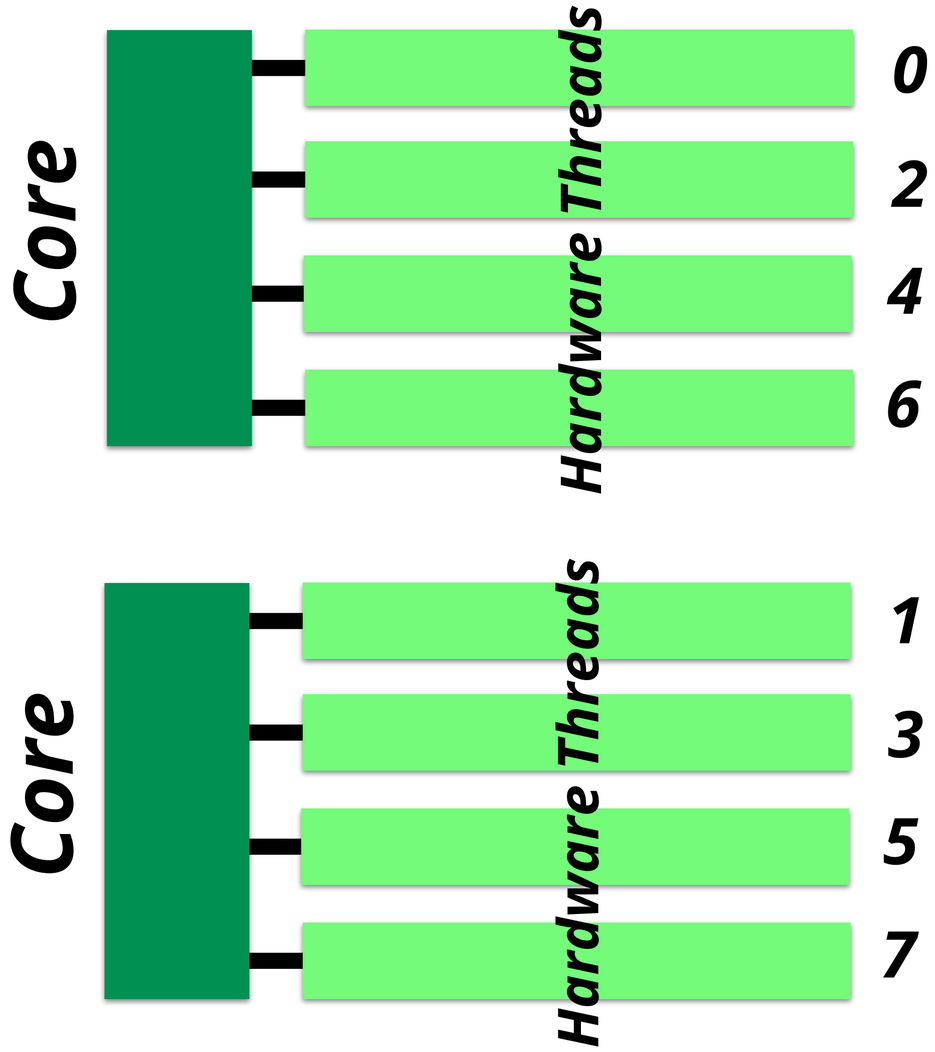
The idea is that the pipeline is used by another thread in case the current thread is idle

Each hardware thread has a unique ID in the system

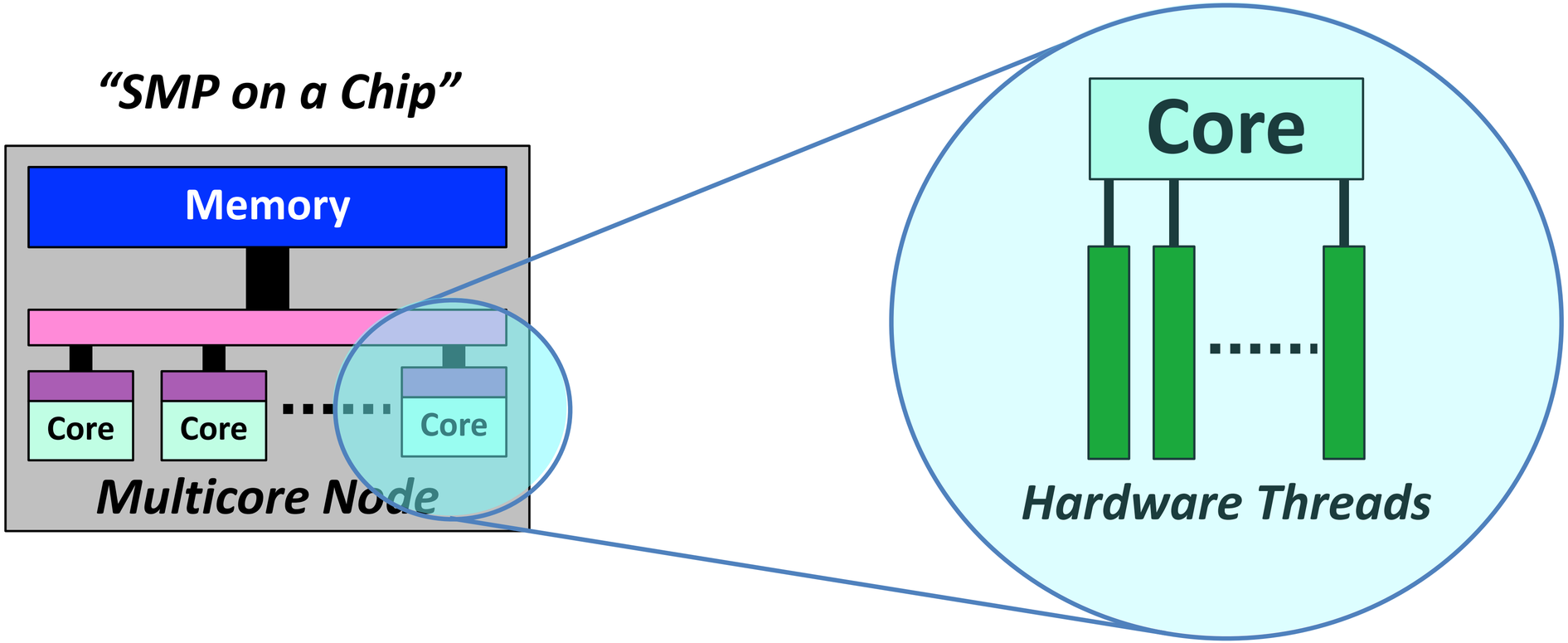
How Hardware Threads Work



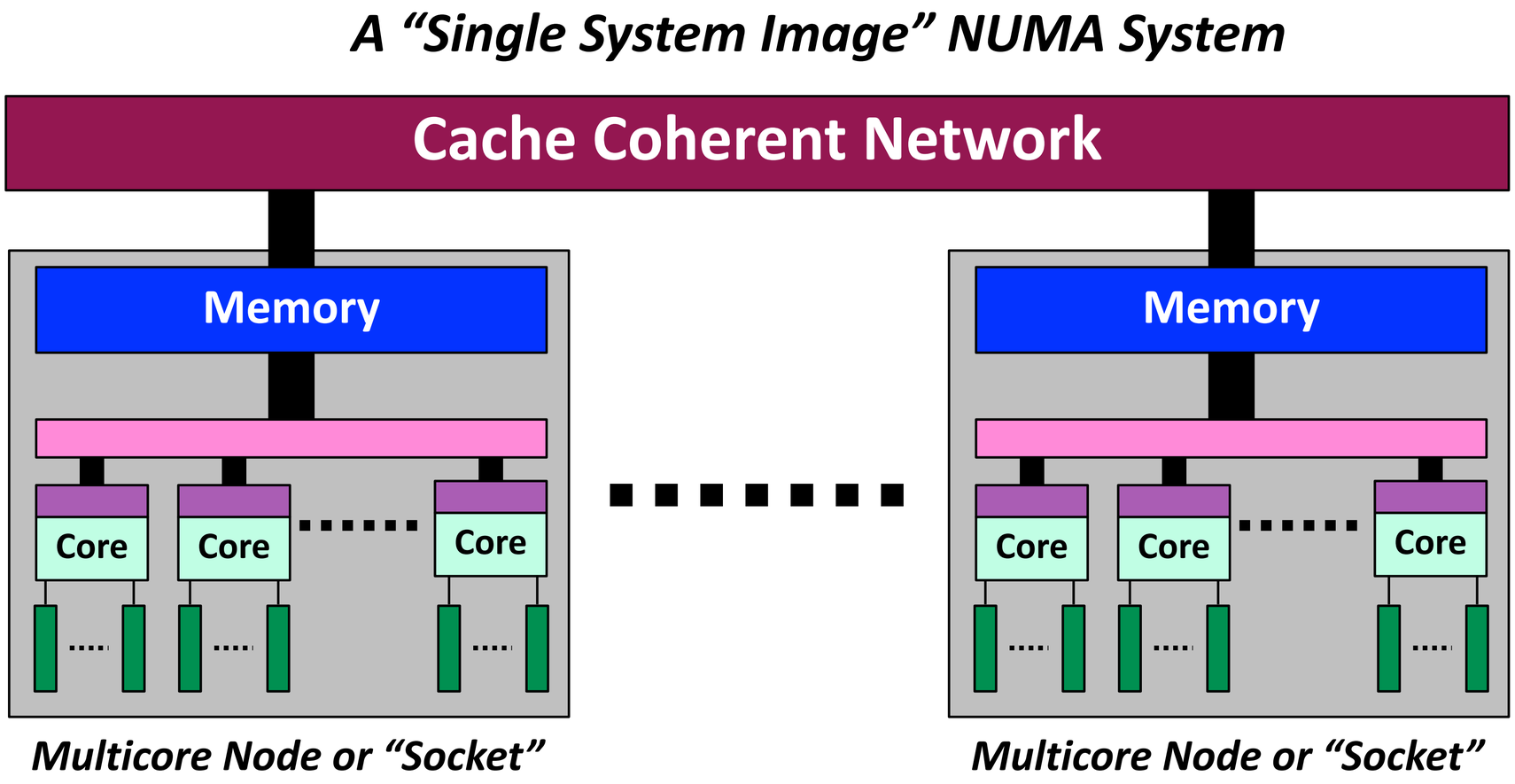
Hardware Thread IDs



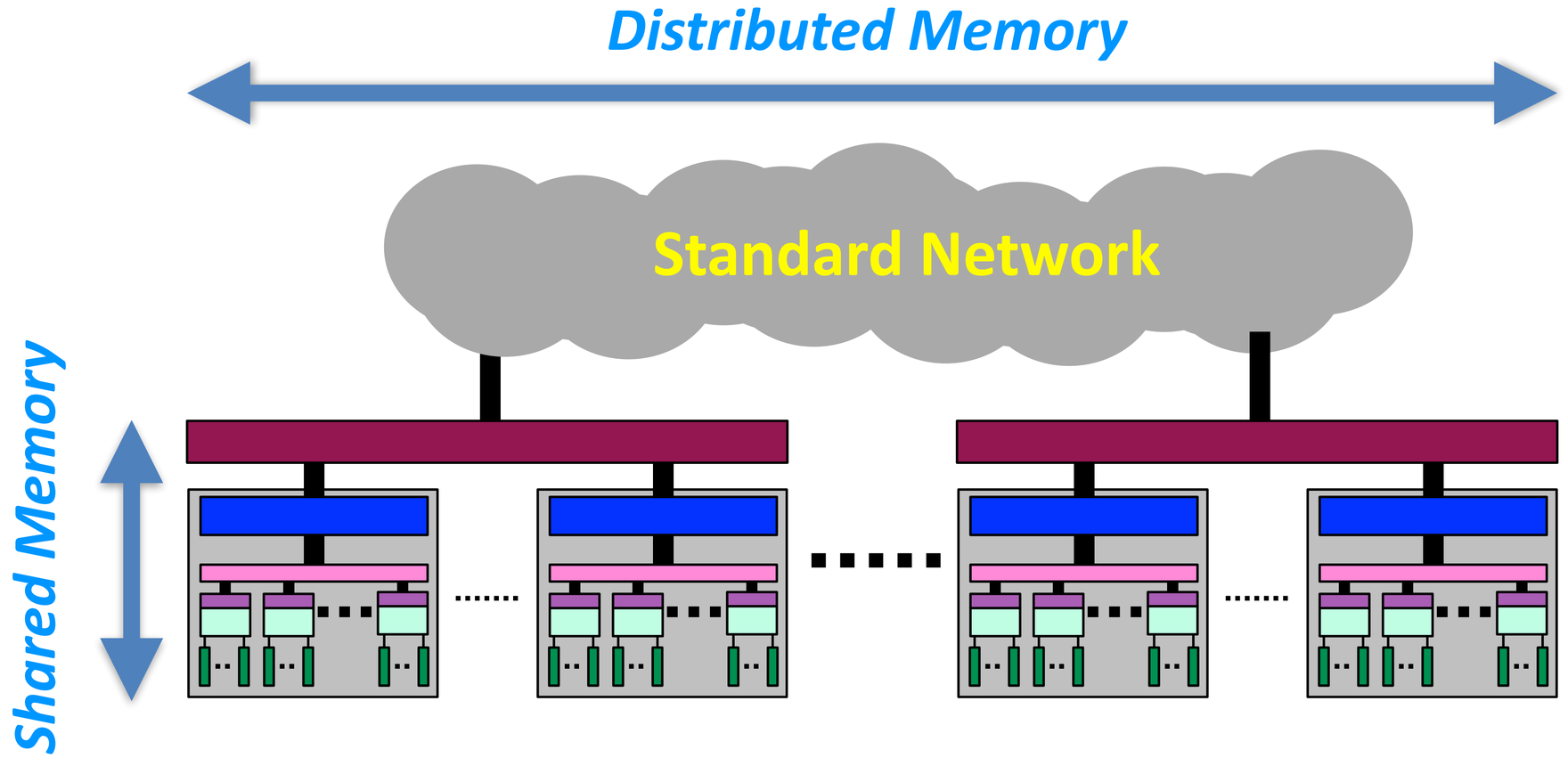
Multicore and Hardware Threads



A Contemporary Multi-Node System



A Hybrid Parallel System



The Graphical Processing Unit (GPU)

Started as an add-on card for graphics processing

By now, the GPUs are very powerful parallel compute engines

While they are still "added" to a conventional processor, they often handle a large part of the workload

Not all workloads can benefit from a GPU, but for example, they are very heavily used in AI computations

During PPCES you will learn more how to use the GPU(s)

A Silent Hardware Evolution

Arm sells a processor or device design, not a product

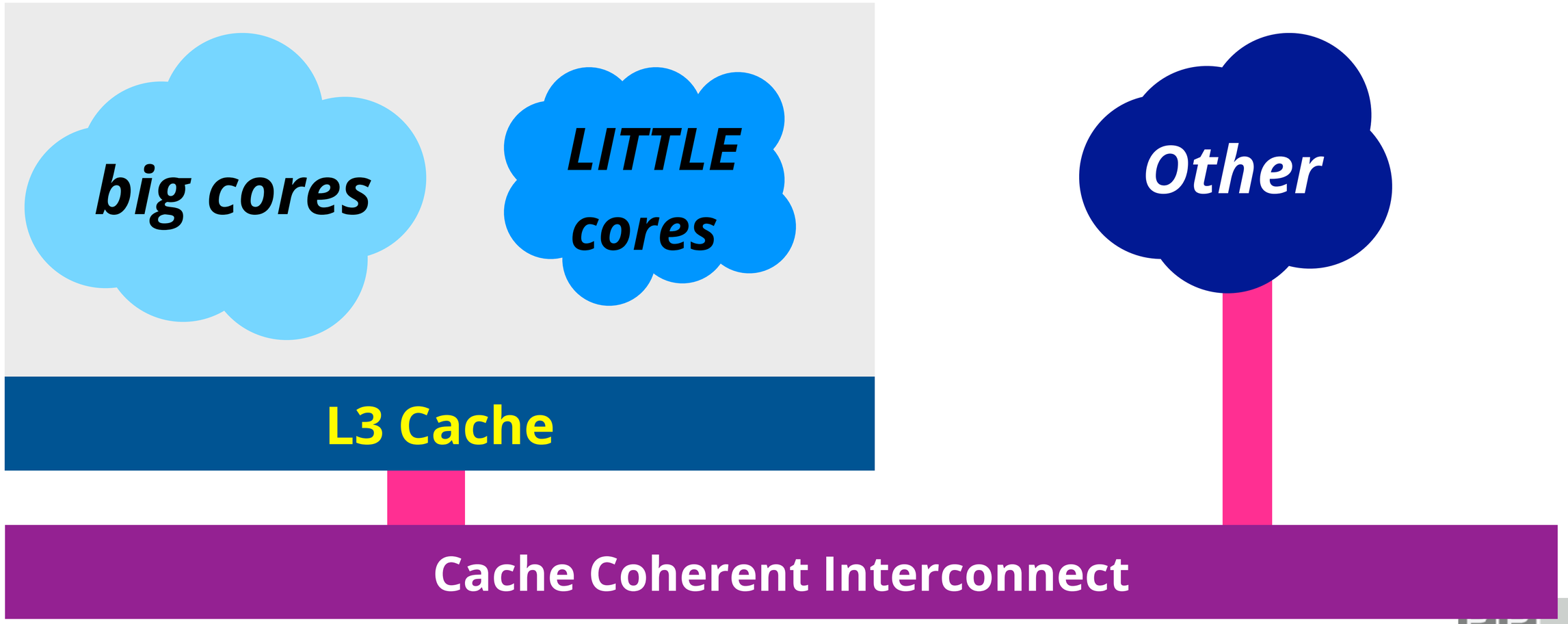
Large companies leverage this model

They collaborate with Arm on very sophisticated designs

Those processors go into servers, but also into commodity products, like cellphones

Let's see what that brings us

big.LITTLE DynamIQ Concept From Arm



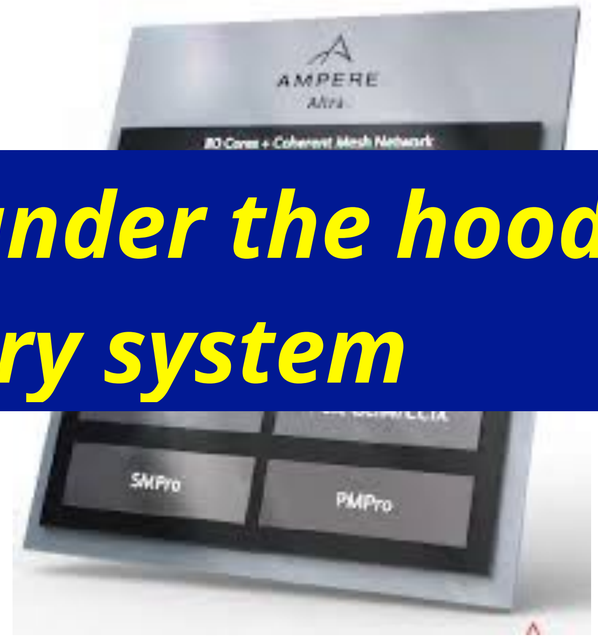
Recent Server Level Processors

48 cores

(+4 support cores)

Arm based

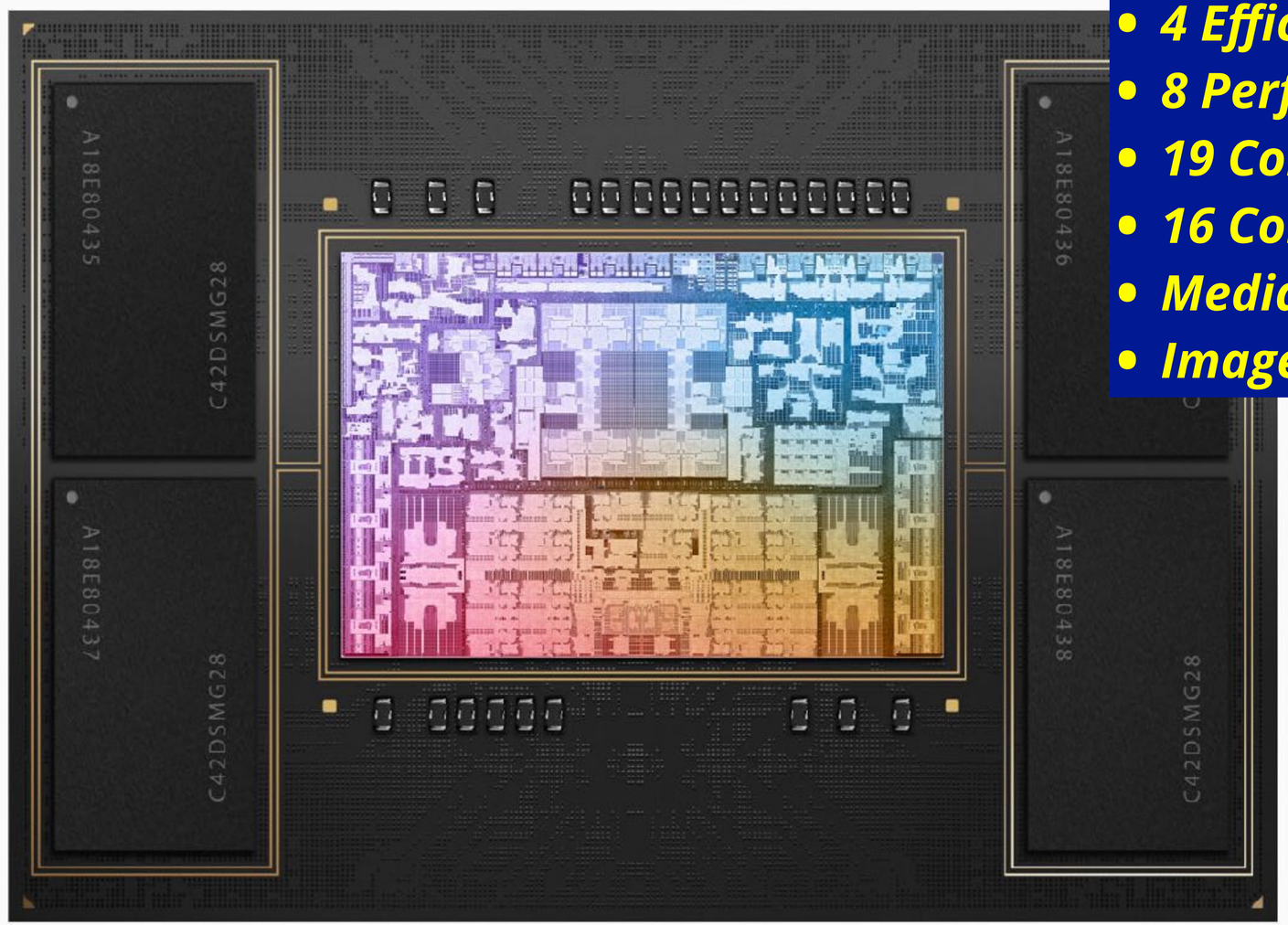
80 cores



Arm based

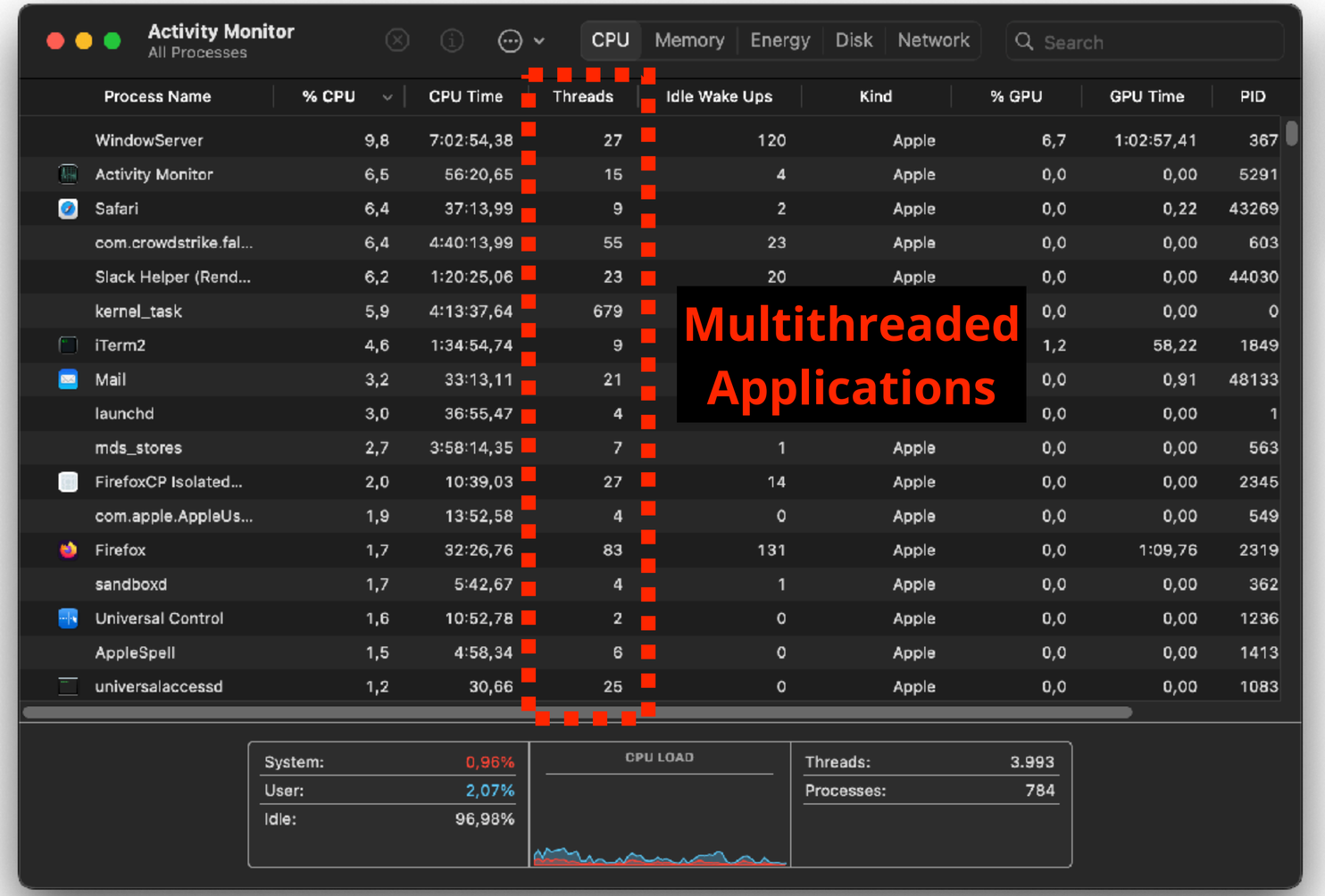
Substantial differences under the hood, but a shared memory system

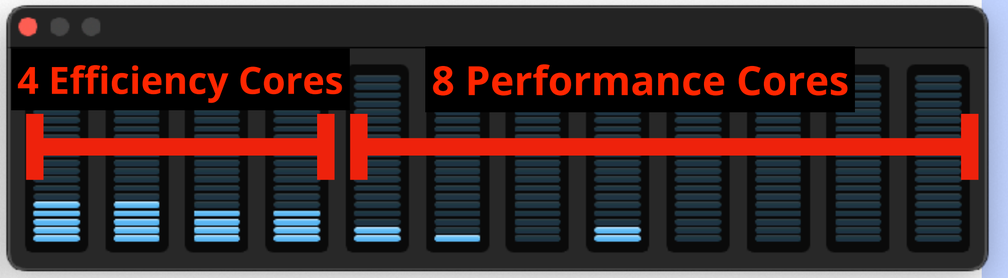
A Laptop or HPC System? - The Apple M2 Pro Processor



- 4 Efficiency Cores
- 8 Performance Cores
- 19 Core GPU
- 16 Core Neural Engine
- Media Engine
- Image Signal Processor

It is Definitely a Multithreaded Architecture





Parallel Programming Models

Programming Parallel Systems

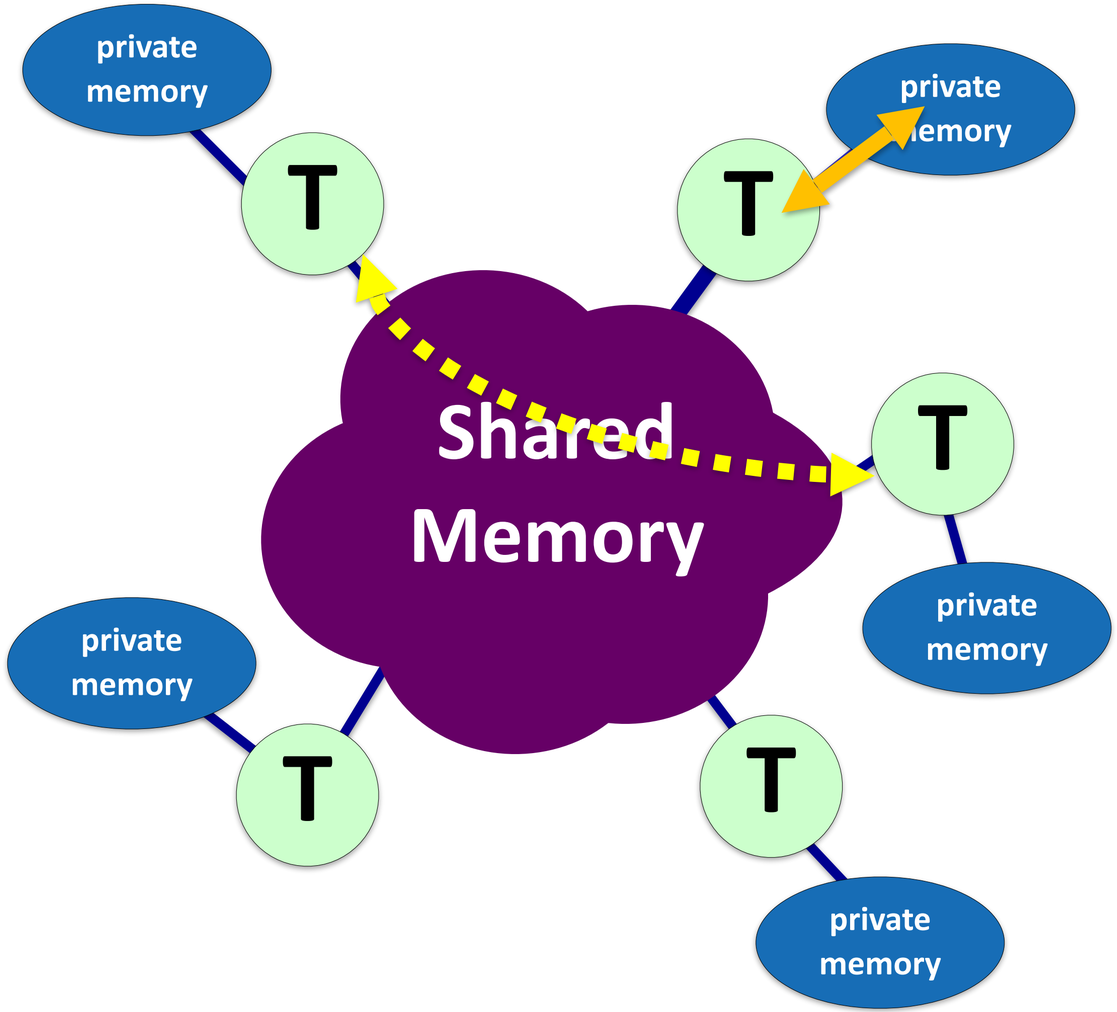
How do we program such systems?

A Distributed Memory System is typically programmed using network sockets, or (in HPC mostly) using MPI

A Shared Memory System is often programmed using Pthreads, or OpenMP, or Java Threads in the case of Java

A Hybrid System uses the combination of these two: MPI across the cluster nodes and OpenMP within a node

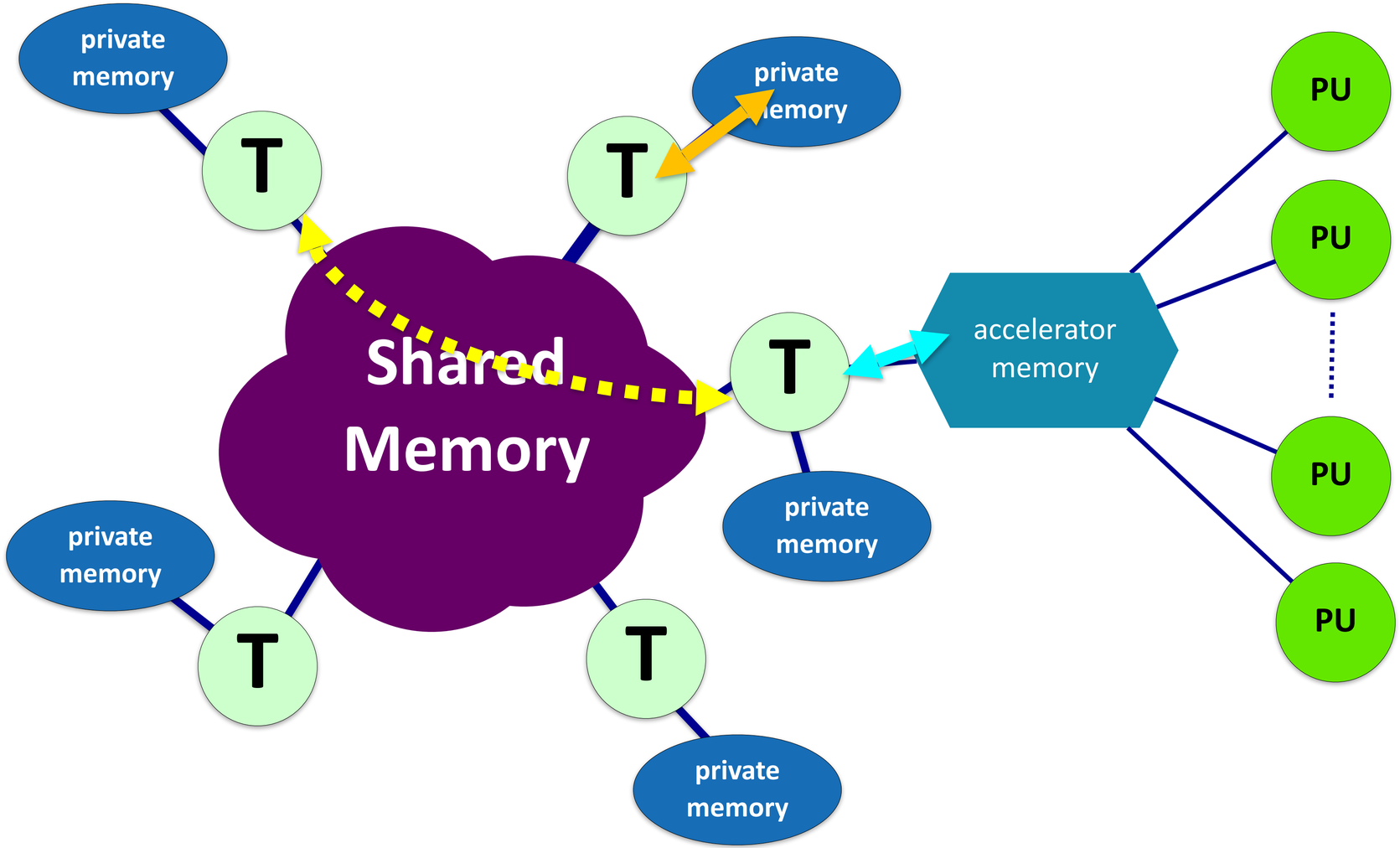
The Shared Memory Model



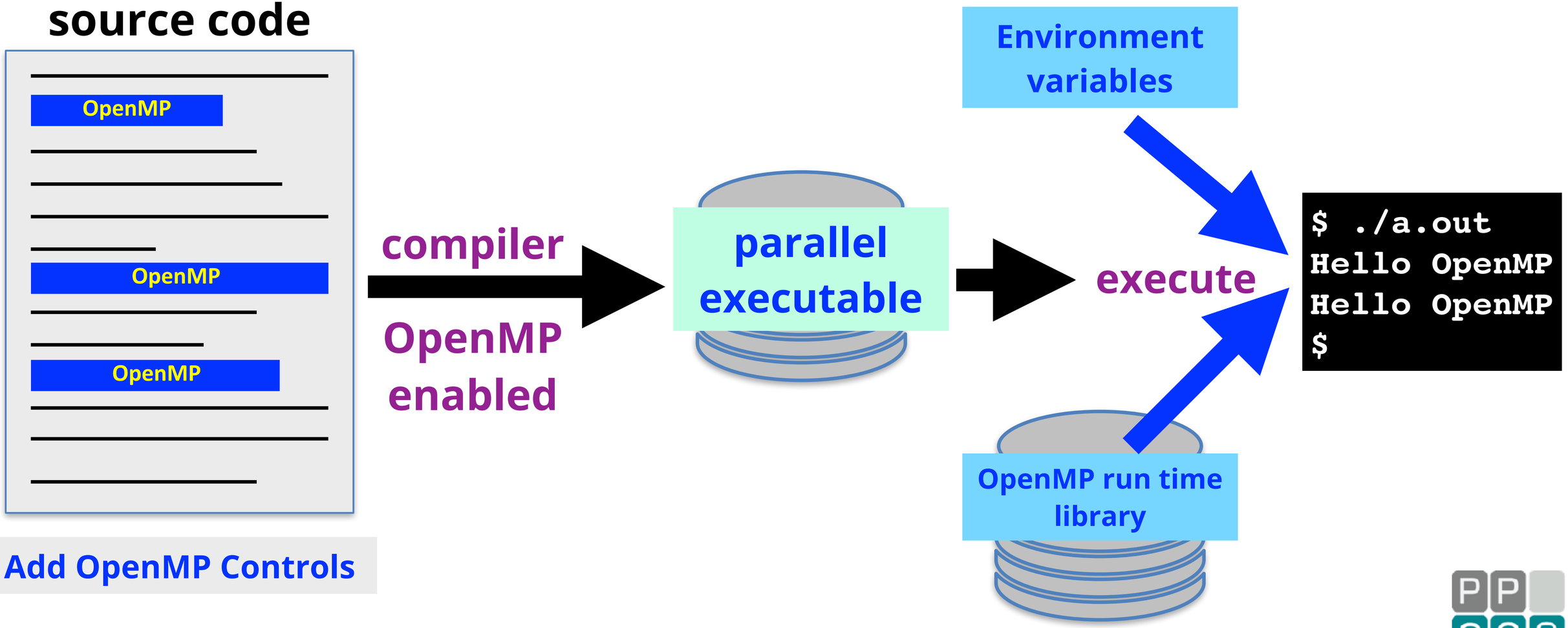
Transparent sharing via Shared Memory

Each thread has a private memory as well

The OpenMP Memory Model



How Does OpenMP Work?



An Example of an OpenMP Program

```
#pragma omp parallel for private(i) shared(a)
for (i=0; i<10; i++)
  a[i] = 0;
```

Static
In the source code

Command Line
Set the number of threads to 2

Dynamic
At run time

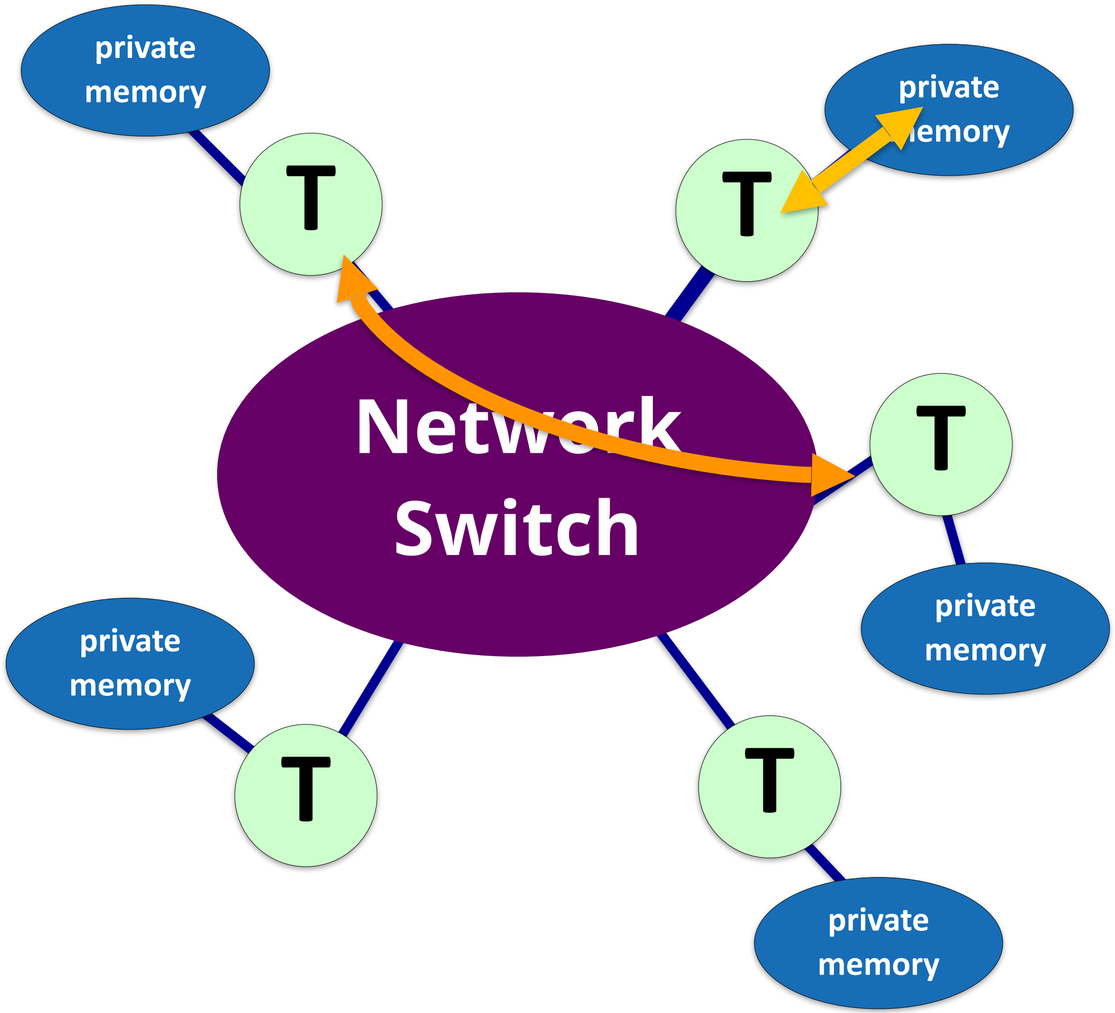
Thread 0

```
for (i=0; i<=4; i++)
  a[i] = 0;
```

Thread 1

```
for (i=5; i<=9; i++)
  a[i] = 0;
```

The Distributed Memory Model (e.g. MPI)



Nothing is shared

Sharing is through sending and receiving messages

Example - Fragment of an MPI Program

```
integer data(10), status(MPI_STATUS_SIZE)
you = 1
him = 0
call MPI_Init(ierr)
call MPI_Comm_Rank(MPI_COMM_WORLD, me, ierr)
if (me == 0) then
  call MPI_Send(data, 10, MPI_INTEGER, you, 1957, MPI_COMM_WORLD, ierr)
else if (me == 1) then
  call MPI_Recv(data, 10, MPI_INTEGER, him, 1957, MPI_COMM_WORLD, status,ierr)
end if
call MPI_Finalize(ierr)
```

Status of receive operation

Initialize MPI environment

Get the ID of the MPI rank executing the code

If I am rank 0, send 10 integers to you

If I am rank 1, receive 10 integers from him

Stop the MPI environment

Example - Two Processes are Started

MPI Rank 0

```
you = 1
him = 0
call MPI_Init(...)
call MPI_Comm_Rank(me)
call MPI_Send(you)
call MPI_Finalize(...)
```

Sets me = 0

MPI Rank 1

```
you = 1
him = 0
call MPI_Init(...)
call MPI_Comm_Rank(me)
call MPI_Recv(him)
call MPI_Finalize(...)
```

Sets me = 1

Common Mistakes in Parallel Computing

What Could Go Wrong in Parallel Computing?

Every programming model comes with specific pitfalls
We list some of the more common ones

OpenMP

- **Illegal parallelization**
- **Incorrect scoping**
- **Synchronization errors**
- **Data races**
- **…**

MPI

- **Illegal parallelization**
- **Send/receive mismatch**
- **Message label incorrect**
- **Individual process may crash**
- **…**

Data Races

In a shared memory model, updates of shared data may require care

Since each thread can read and write shared data, one has to be careful this happens correctly

Failure to do so, introduces a “data race”

Definition of a Data Race

Two different threads in a multithreaded shared memory program, access the same (=shared) memory location

Concurrently and

Without holding any common exclusive locks and

At least one of the accesses is a write/store operation

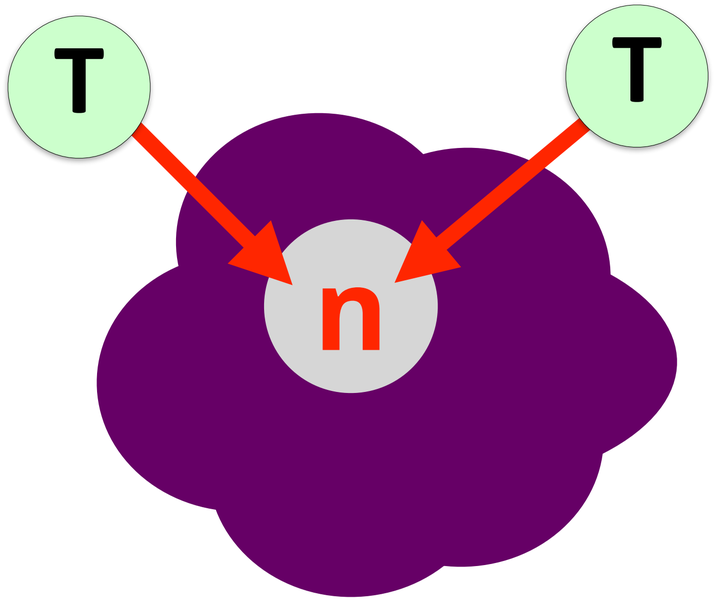
If all these 3 conditions are met, the program has a data race

A data race leads to silent data corruption …

An Example of a Data Race

```
#pragma omp parallel  shared(n)
{
   n = omp_get_thread_num();

} // End of parallel region
```



What is the final value of variable "n"?

It depends. Even from run to run …

Note: As you will learn during PPCES, OpenMP has constructs to avoid data races

Why Writing Parallel Programs?

Parallel Programming is Great Fun!

It does come with its own set of pitfalls

Don't despair though and don't give up

The reward is blazing performance :-)


Thank You And ... Stay Tuned!
Bad OpenMP
Does Not Scale
Ruud van der Pas