Introduction into Parallel Computing

Ruud van der Pas
Senior Principal Software Engineer
Oracle Linux and Virtualization Engineering
Oracle, USA

PPCES 2024
March 11-15, 2024
RWTH Aachen University

$ whoishe

My background is in mathematics and physics

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

Currently I work in the Oracle Linux Engineering organization

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

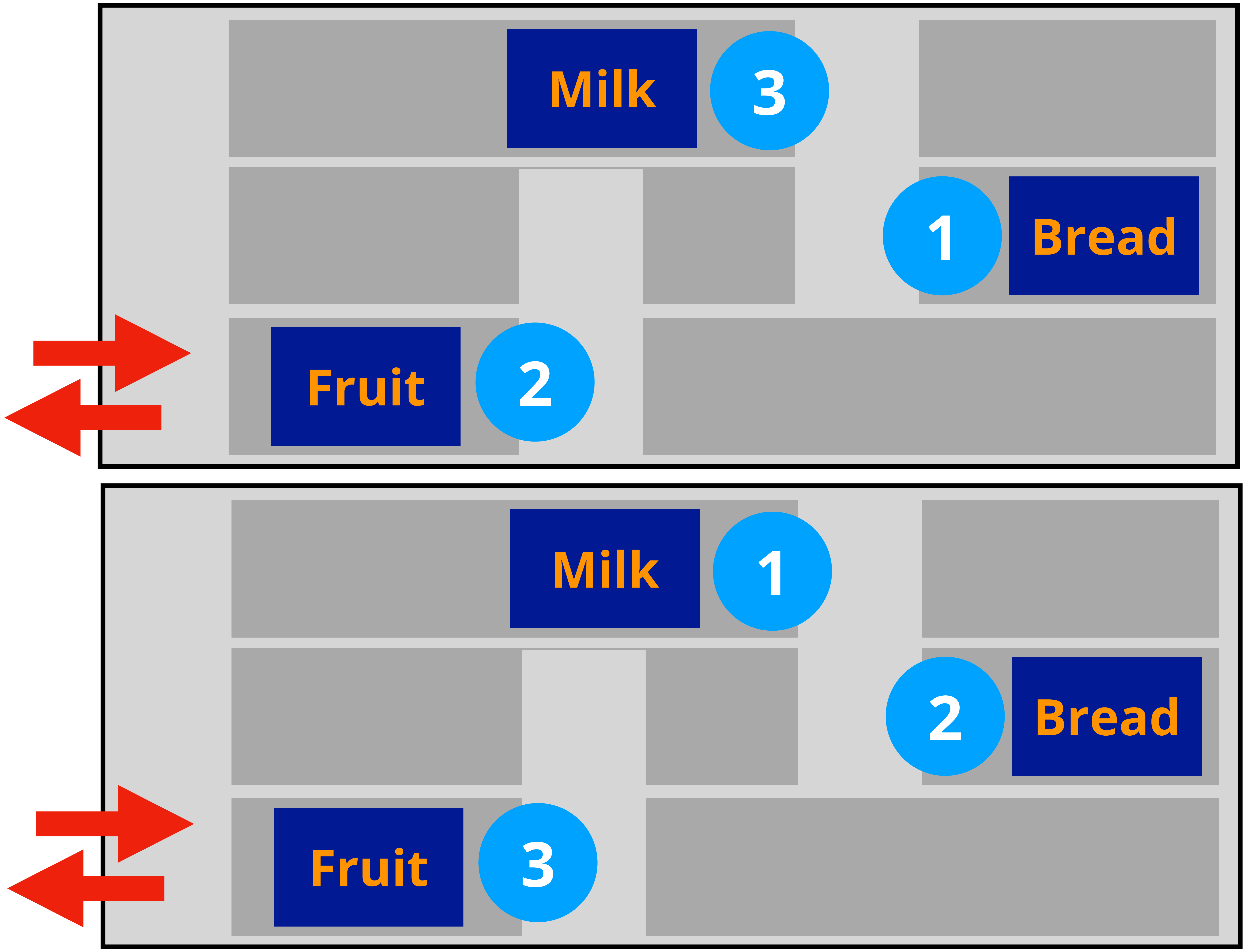
The Topics

- *What is Parallelism?*
- *What is Parallel Computing?*
- *Concepts in Parallel Computing*
  - *What is a thread?*
  - *Serial versus Parallel*
  - *Parallel overhead*
  - *Amdahl's Law*
  - *Load balancing*
  - *Numerical results*
- *Parallel Architectures*
- *Parallel Programming Models*
- *Common Mistakes in Parallel Applications*

What is Parallelism?

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?

No fixed order

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

↓

Independence

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

↓

Parallelism

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

*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

What is Parallel Computing?

Parallel Computing - An Informal Definition

The goal of Parallel Computing is to reduce the time to solve a problem

To achieve this, multiple computational resources are used to solve a single problem

Examples of such computational resources are hardware threads, cores, or even entire systems

Parallel Computing - Methodology

Select a parallel programming model (more on that later)

Identify independent operations/computations in your application

Apply the parallel programming model to distribute the work over the computational resources available to you

Run your program, close your eyes, wait, open them, and hope for the best ;-)

Parallel Computing - The Benefit

Theoretically, can get unlimited performance

For example, 10x using 10 computers, 100x using 100 computers, etc.

In practice, this may be a challenge to achieve though

Concepts in Parallel Computing

What is a Thread?

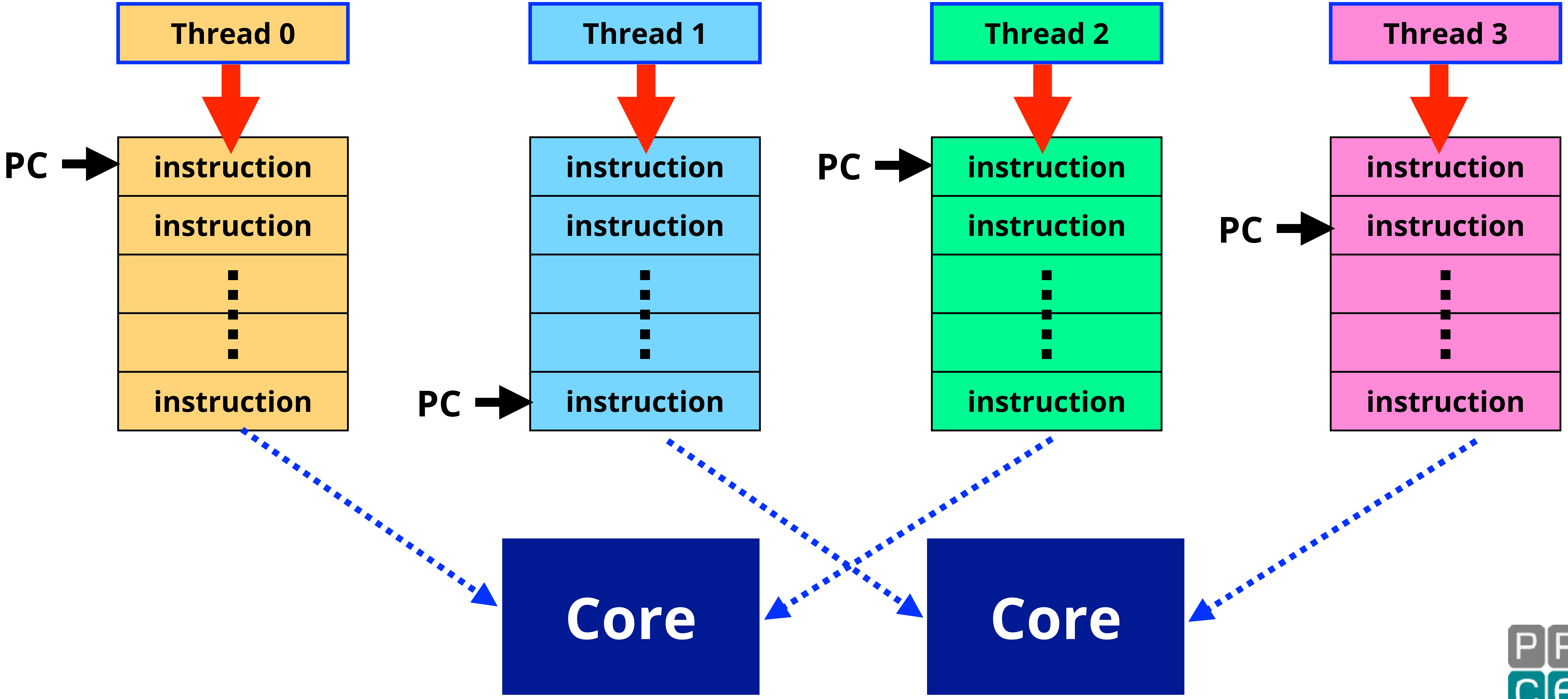
*"A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler"**

In other words, independent parts of an application are executed by threads

Parallel programming is about creating and managing the threads, including assigning work to threads

*) Source: https://en.wikipedia.org/wiki/Thread_(computing)

Example - Four Threads at Work



What is Multithreading?

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

A multithreaded application creates and manages multiple software threads of execution

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 soon, one of the goals of efficient parallelization is to keep the serial part as short as possible

Two Notions of Time

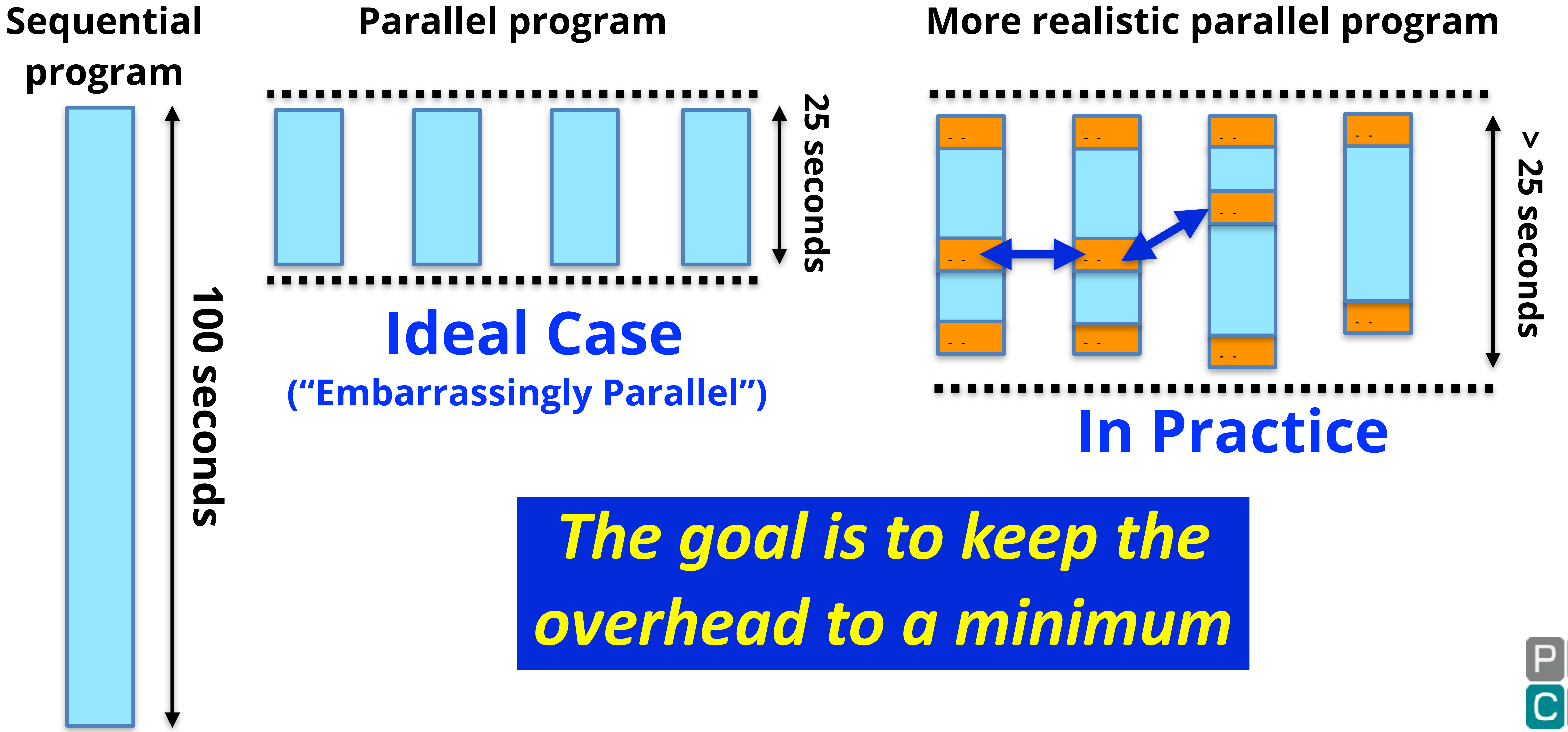
The goal of parallel computing is to reduce the time to solution, usually called the wall clock time, or elapsed time

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

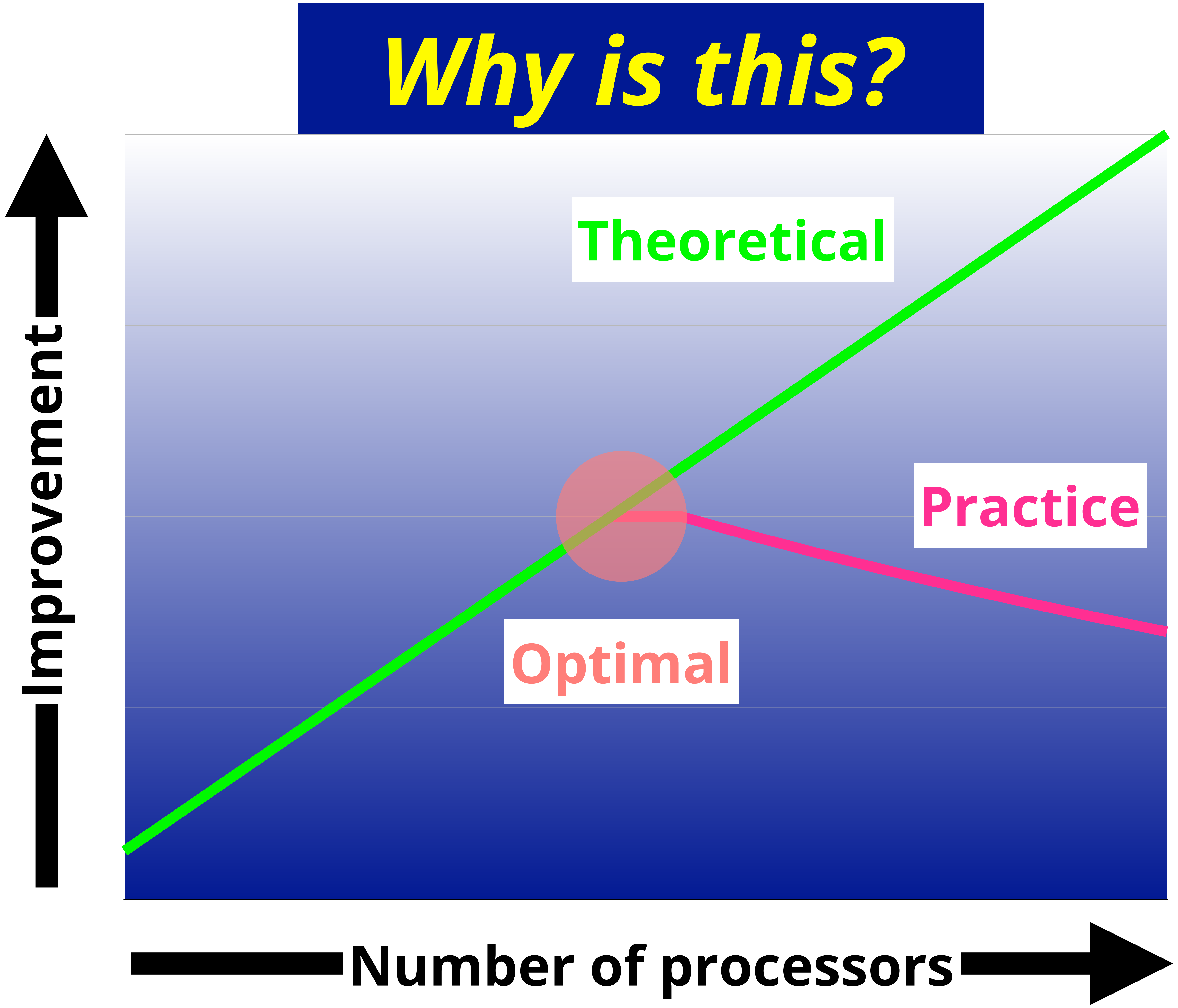
Parallel Computing - Overhead



The goal is to keep the overhead to a minimum

Amdahl's Law

Parallel Speed Up - How Much Faster?



Amdahl's Law - An Example

Suppose your application needs 100 seconds to run

If 80% of this run time can execute in parallel, the time using 4 threads is 80/4+20 = 40 seconds

This means that your program is 2.5x faster, 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: $T(1) = f*T(1) + (1-f)*T(1)$

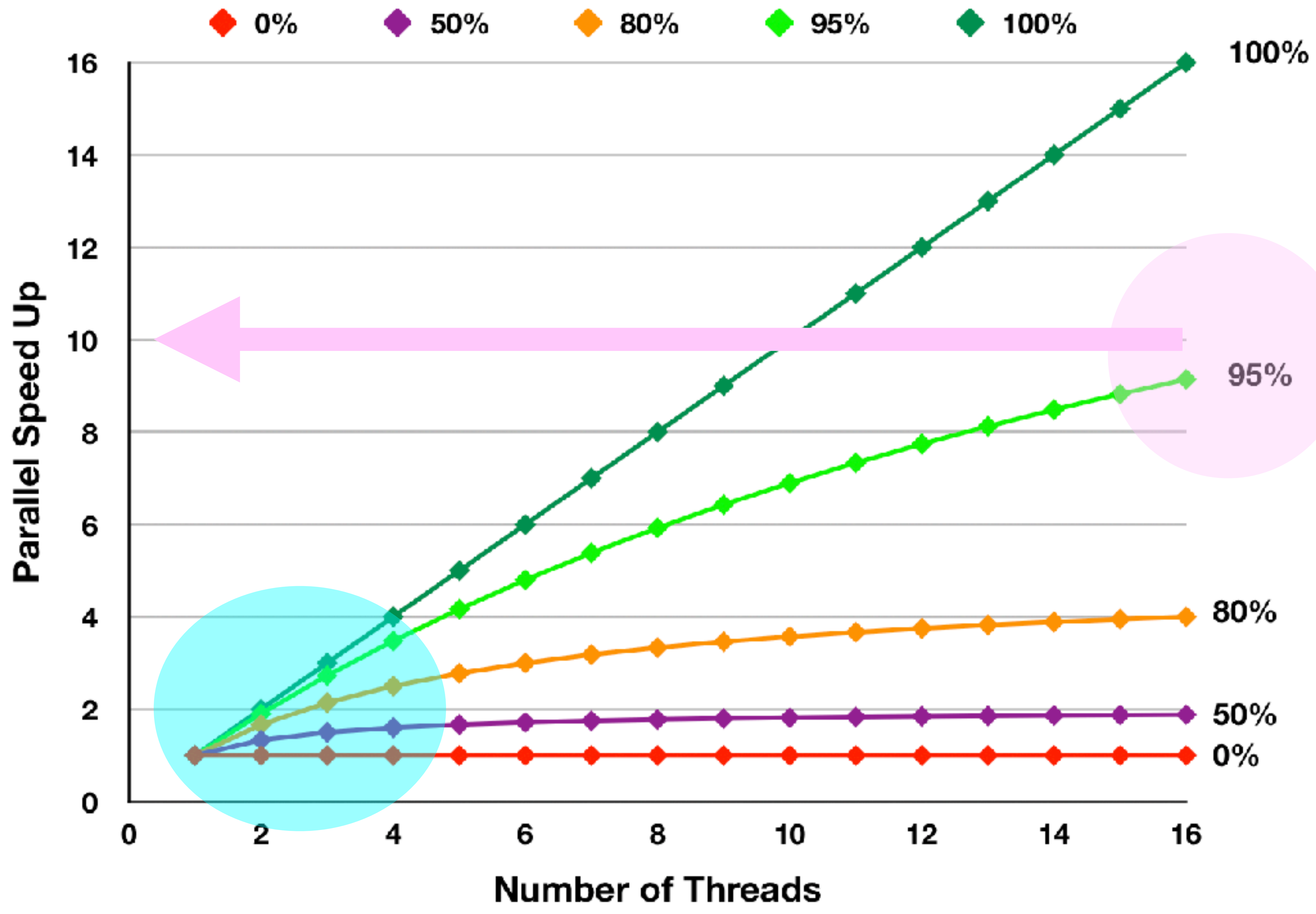
On P threads: $T(P) = f*T(1)/P + (1-f)*T(1) = (f/P + 1-f)*T(1)$

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



Amdahl's Law - Practical Use

Rewrite the formula: $f = (1 - T(P)/T(1))/(1 - 1/P)$

The righthand side can be computed!

Example: $T(1) = 100$ and $T(4) = 37$ => $T(P)/T(1) = 0.37$

It follows that $f = (1-0.37)/(1-1/4) = 0.63/0.75 = 0.84 = 84\%$

Estimated speed-up $S(8, 0.84) = 1/(0.84/8 + 0.16) = 3.78$

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

The most 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

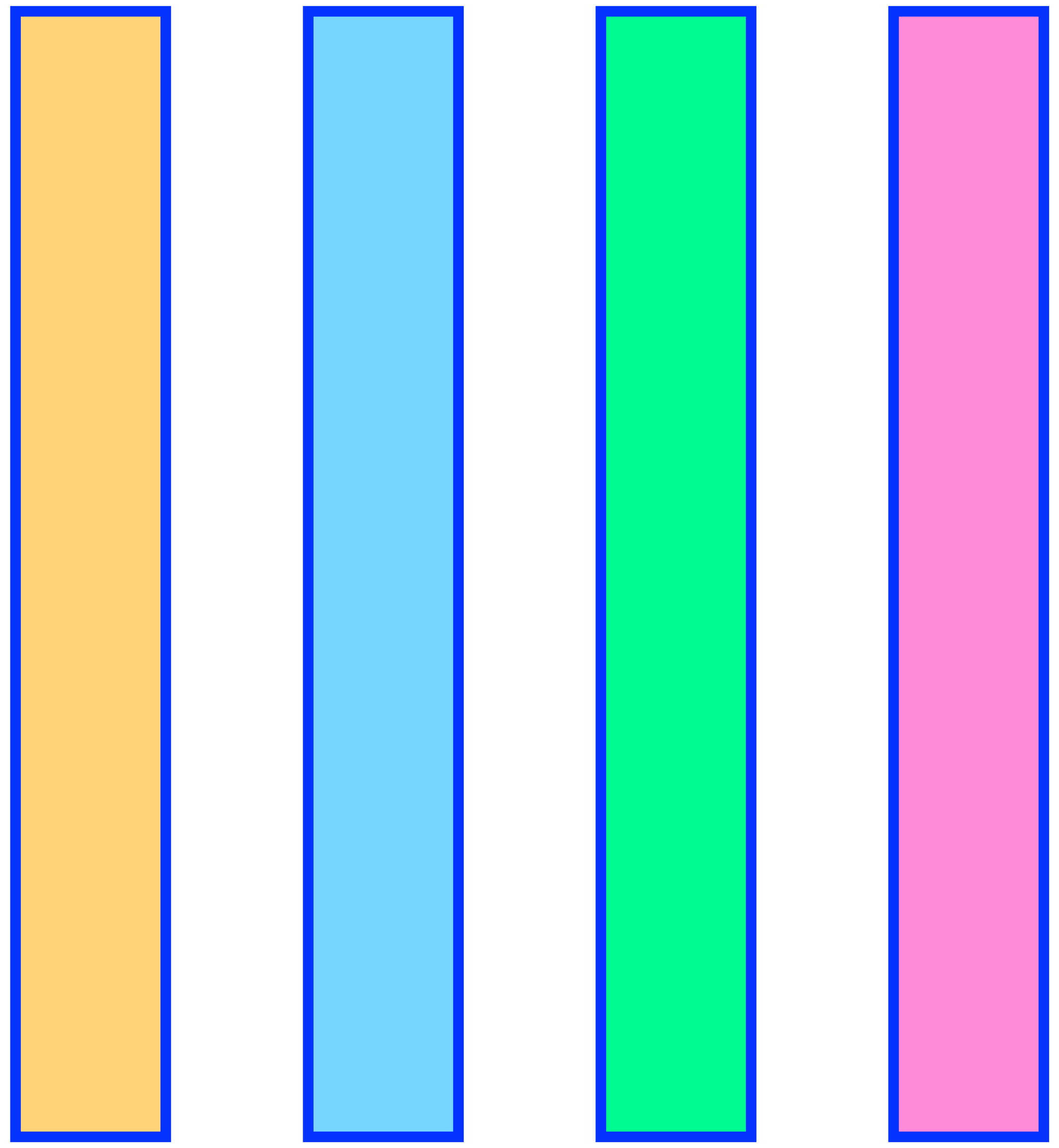
You have to pay attention to single thread performance

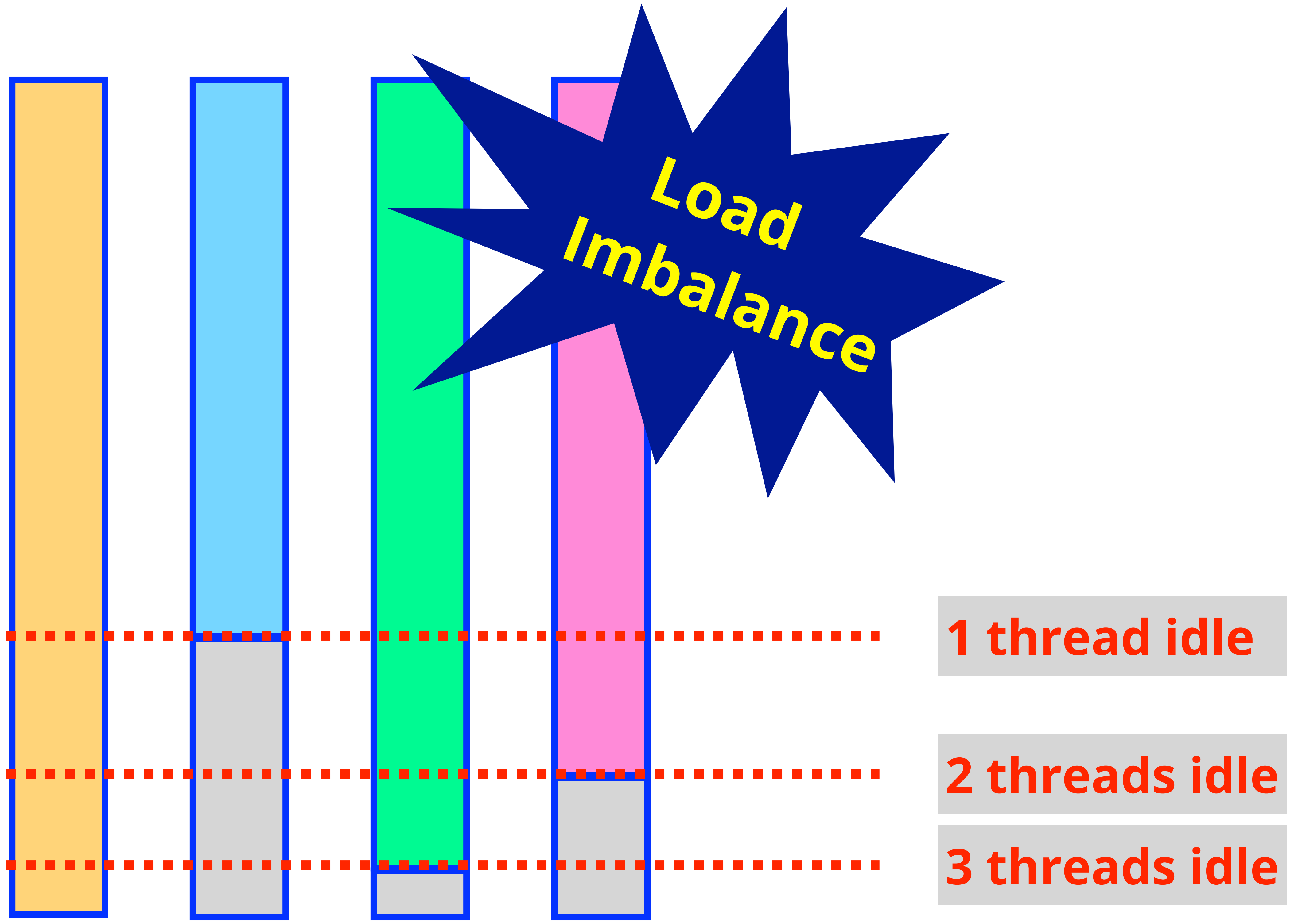
Why? If your code performs badly on 1 core, what do you think will happen on 10 cores, 20 cores, … ?

Scalability can mask poor performance!

(a slow code tends to scale better …)

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

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
```

Time ↓

Sequential Computation

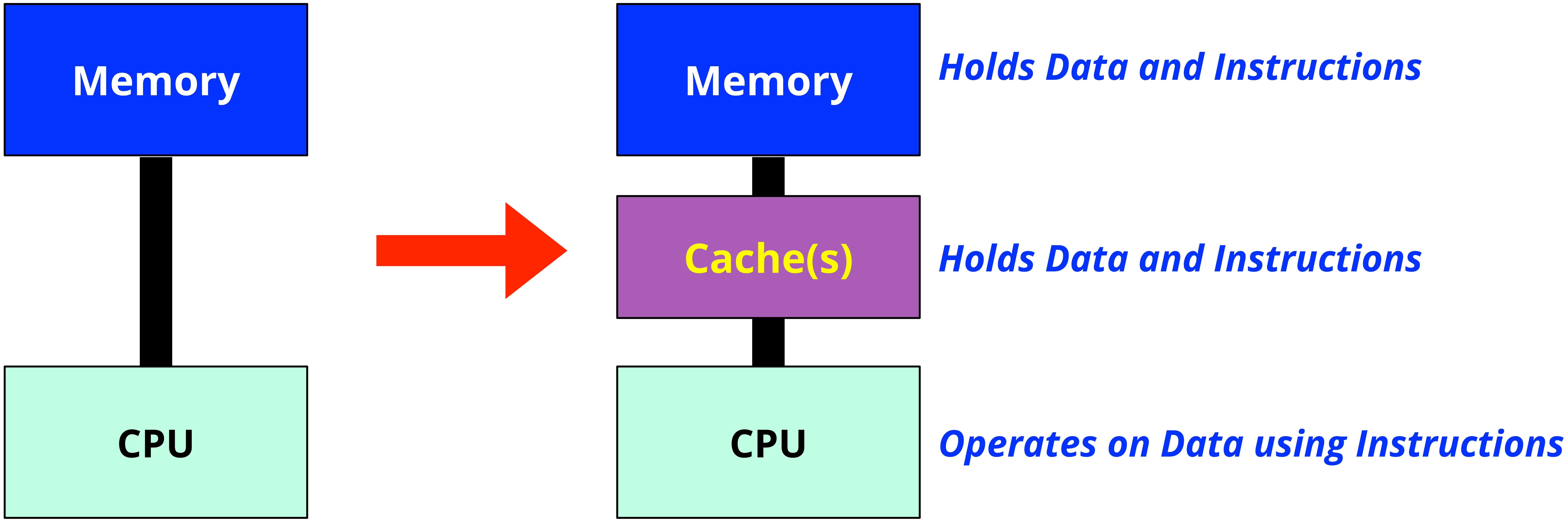
```
A = B + C
A = A + D
A = A + E
```

Parallel Computation

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

Parallel Architectures

A Computer



Note: CPU = Central Processing Unit

Intermezzo - Cache Coherence

Required in a system with shared memory and caches

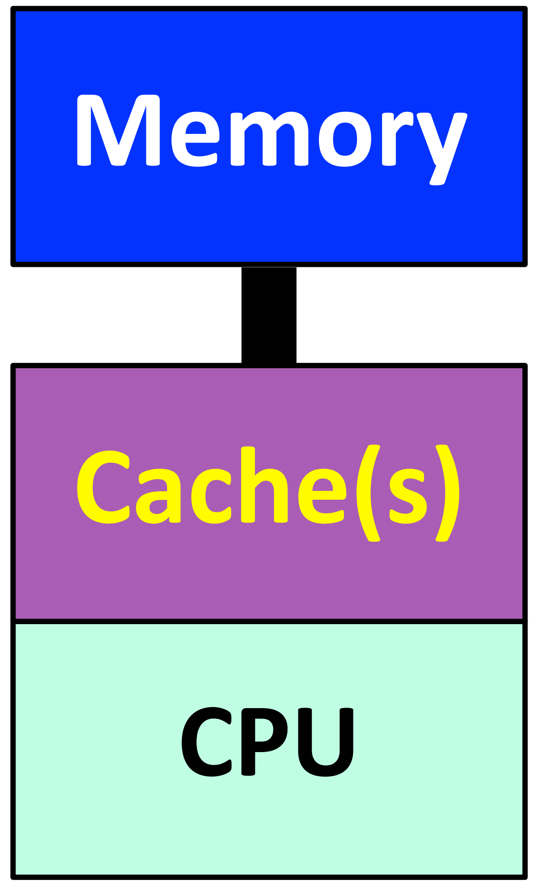
In <u>very</u> simple terms, cache coherence ensures that the system knows where data is, and what the coherency state is

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

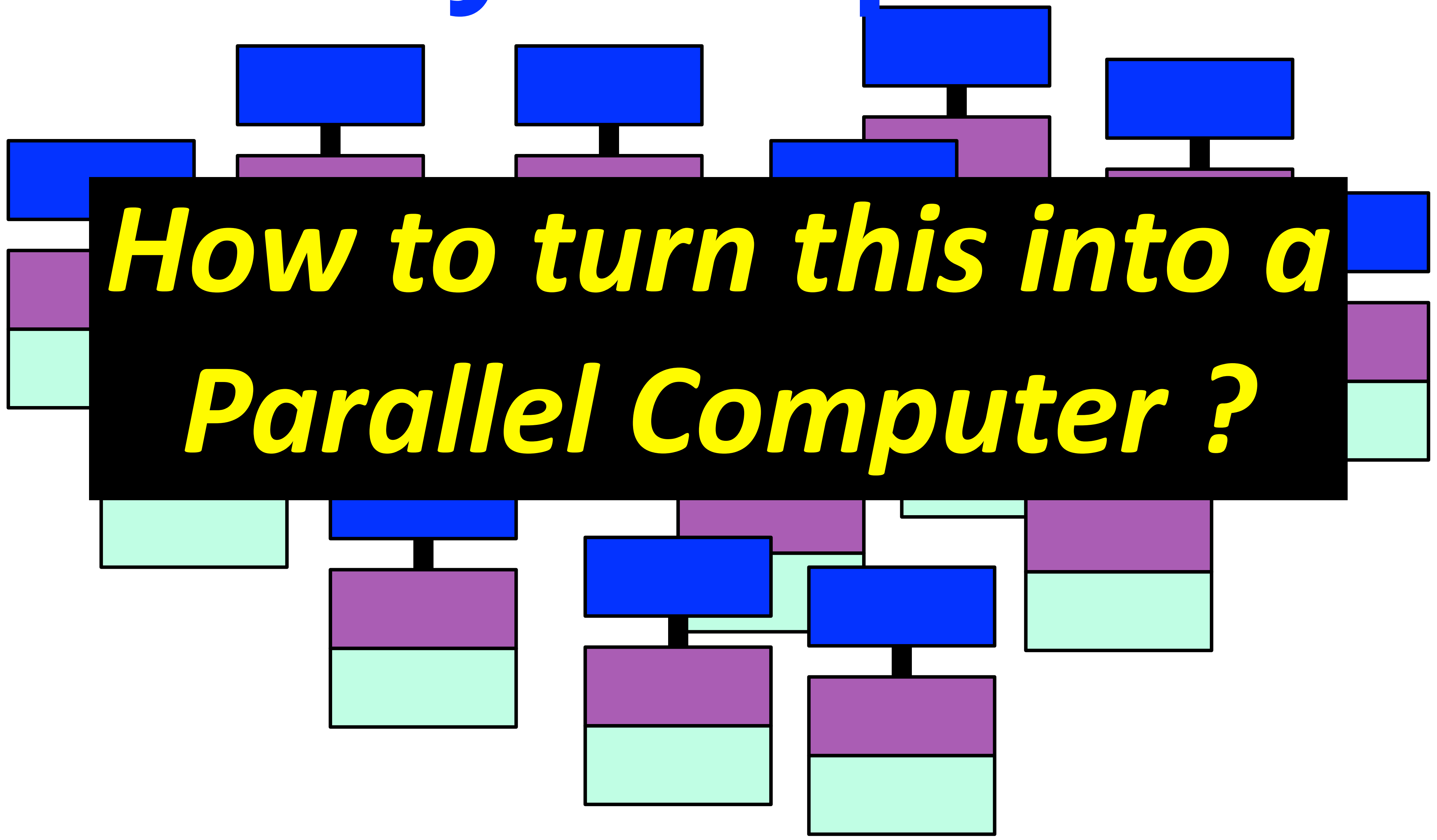
It allows for transparent parallel programming, since the user does not need to know where data physically resides

About Computers

A Computer



Many Computers



Note: CPU = Central Processing Unit

Parallel Architectures - World's Briefest History

Note that Accelerators/ GPUs may be added to one or more nodes

Cores and Multicore

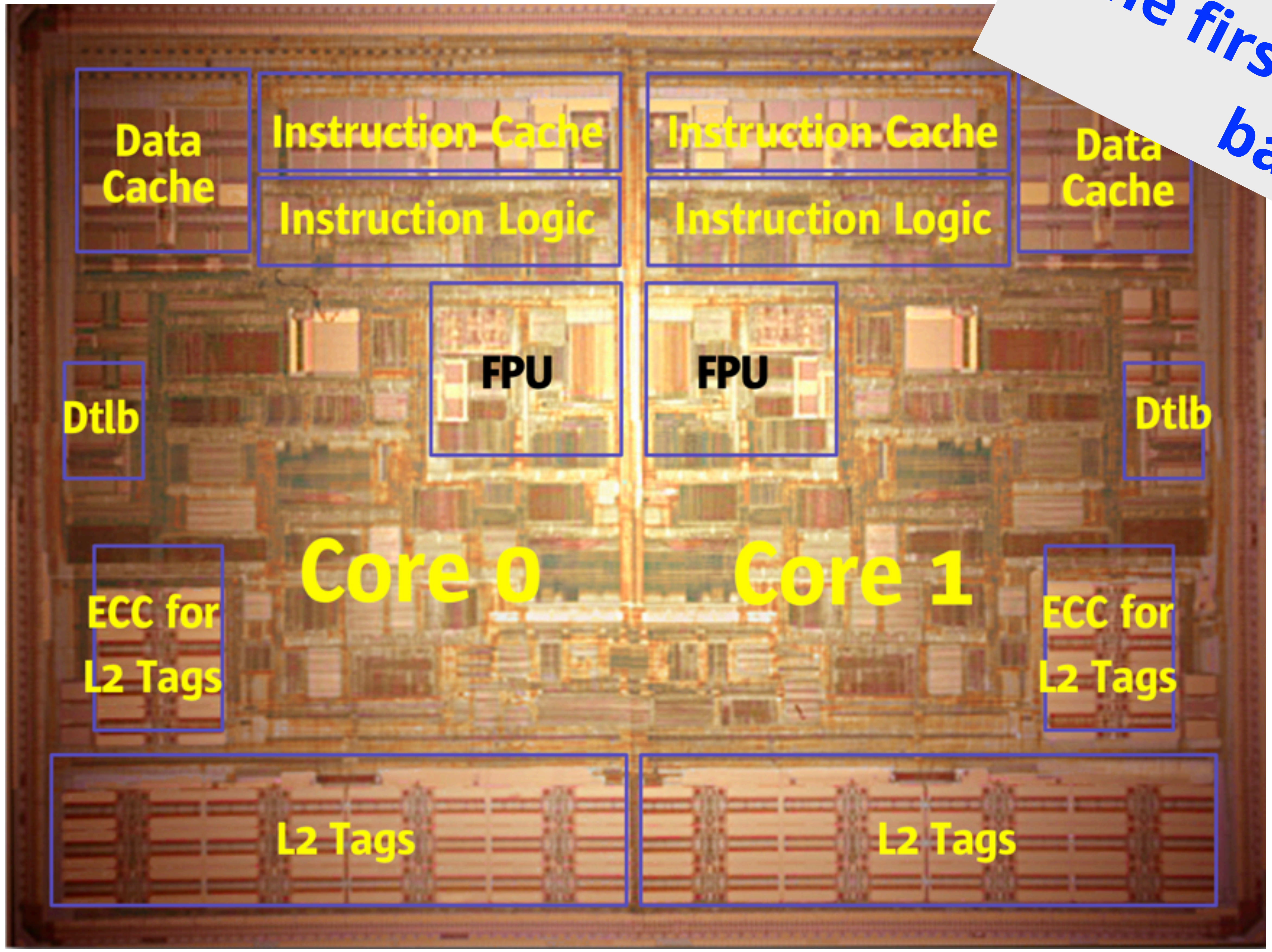
For a long time, the word CPU was used

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

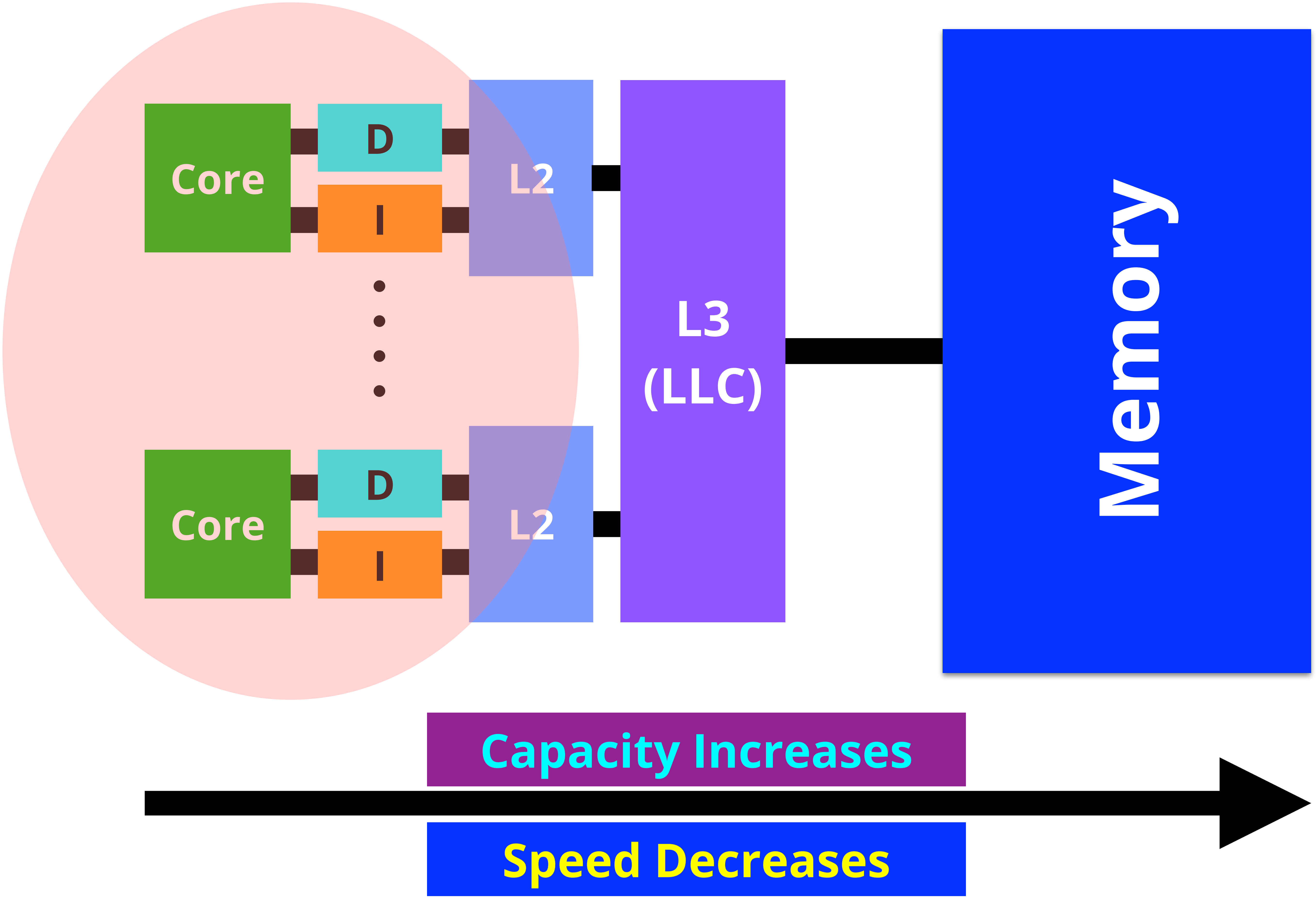
Multicore processors became available a little over 20 years ago

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***

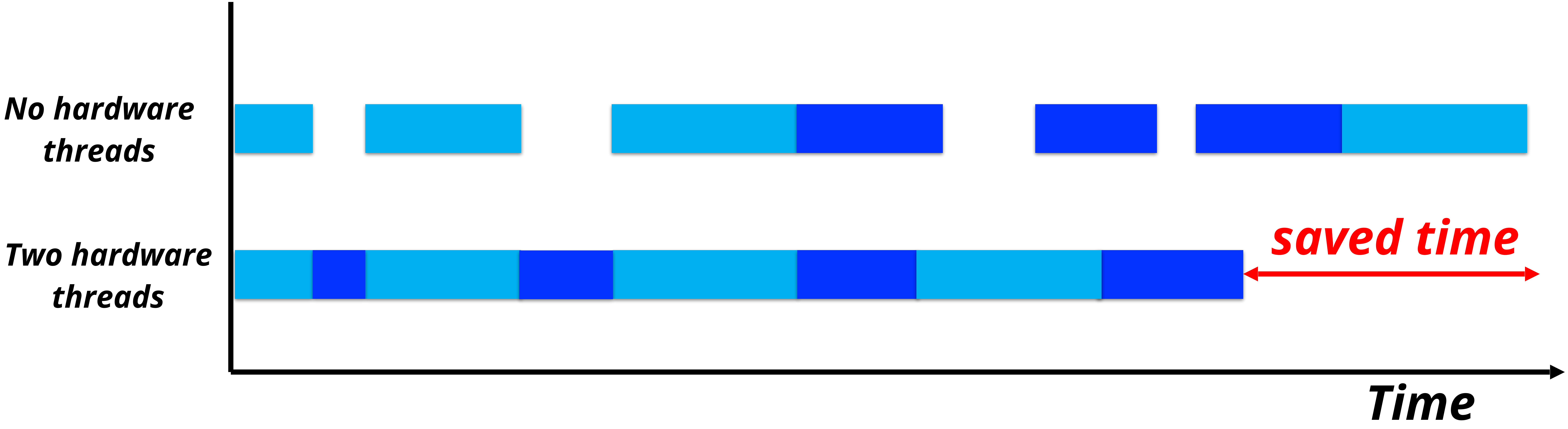
This is part of the design

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

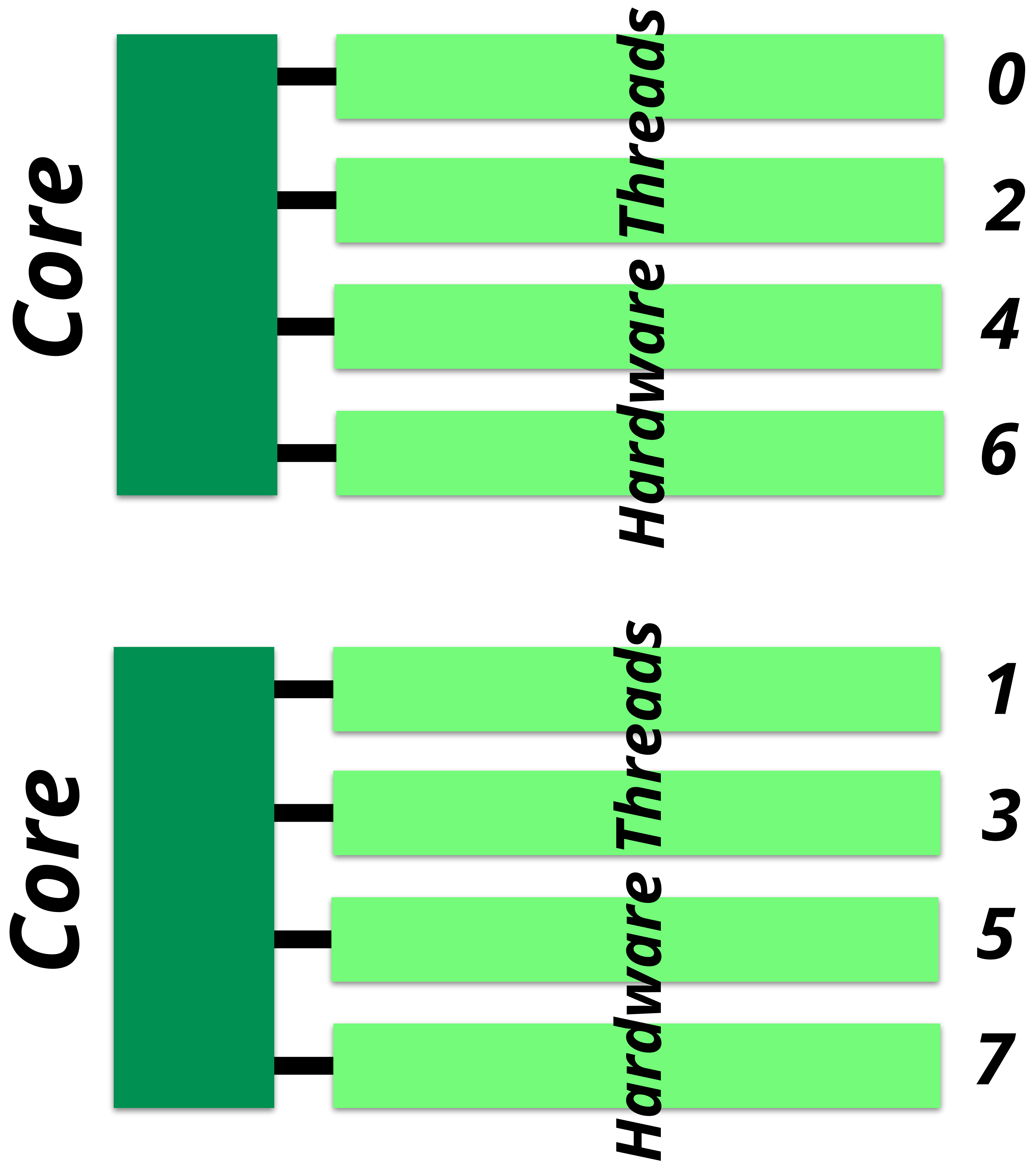
The idea is that the pipeline is used by another thread in case the current thread is idle

Each hardware thread has a unique ID in the system

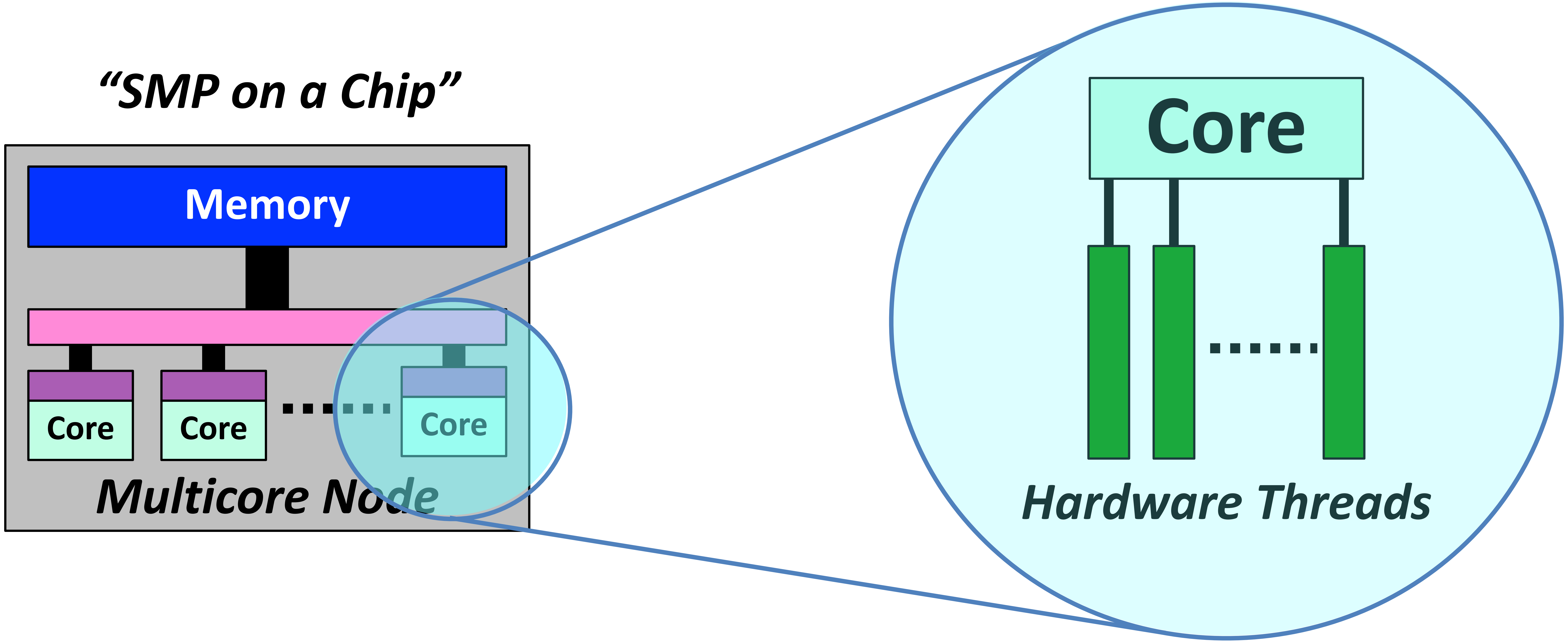
How Hardware Threads Work



Hardware Thread IDs

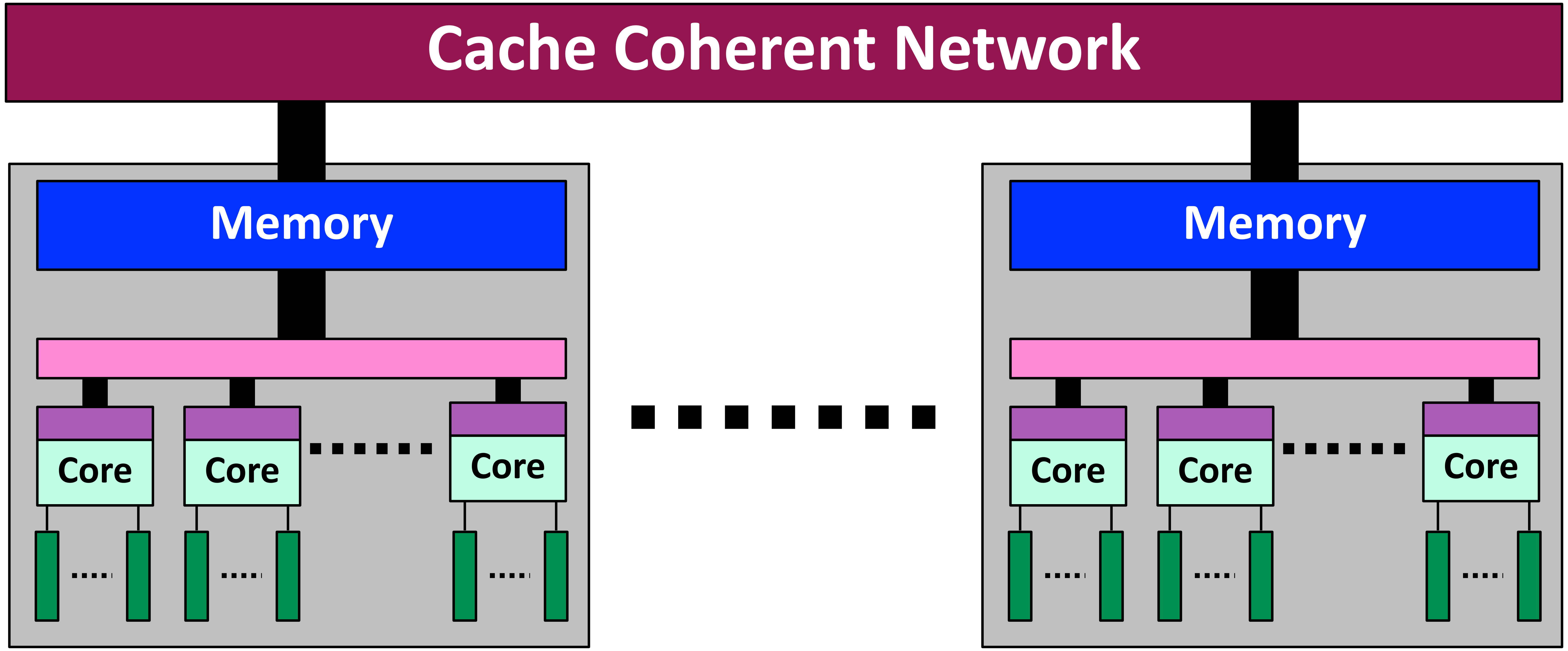


Multicore and Hardware Threads



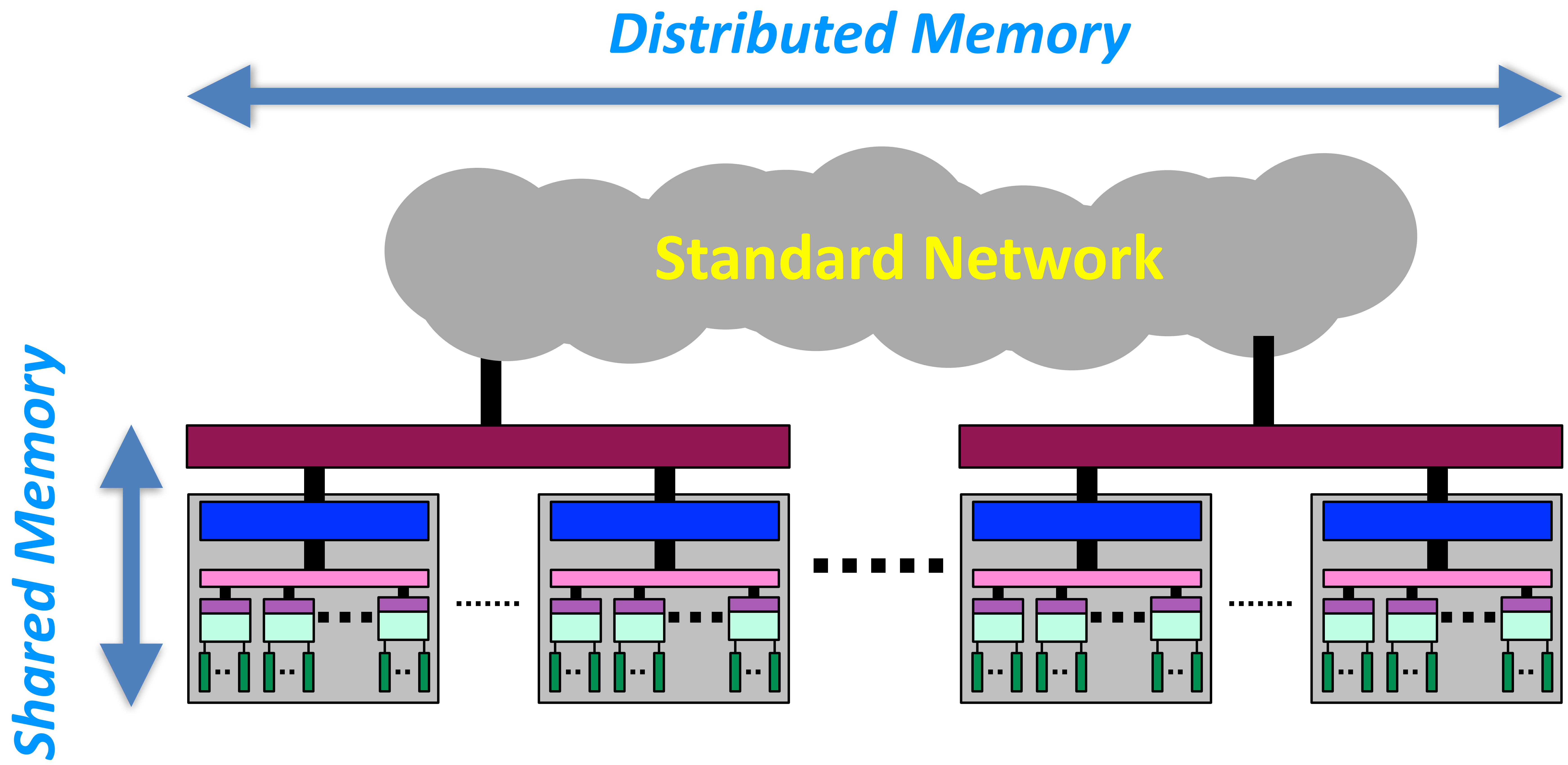
A Contemporary Multi-Node System

A "Single System Image" NUMA System



Multicore Node or "Socket" *Multicore Node or "Socket"*

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

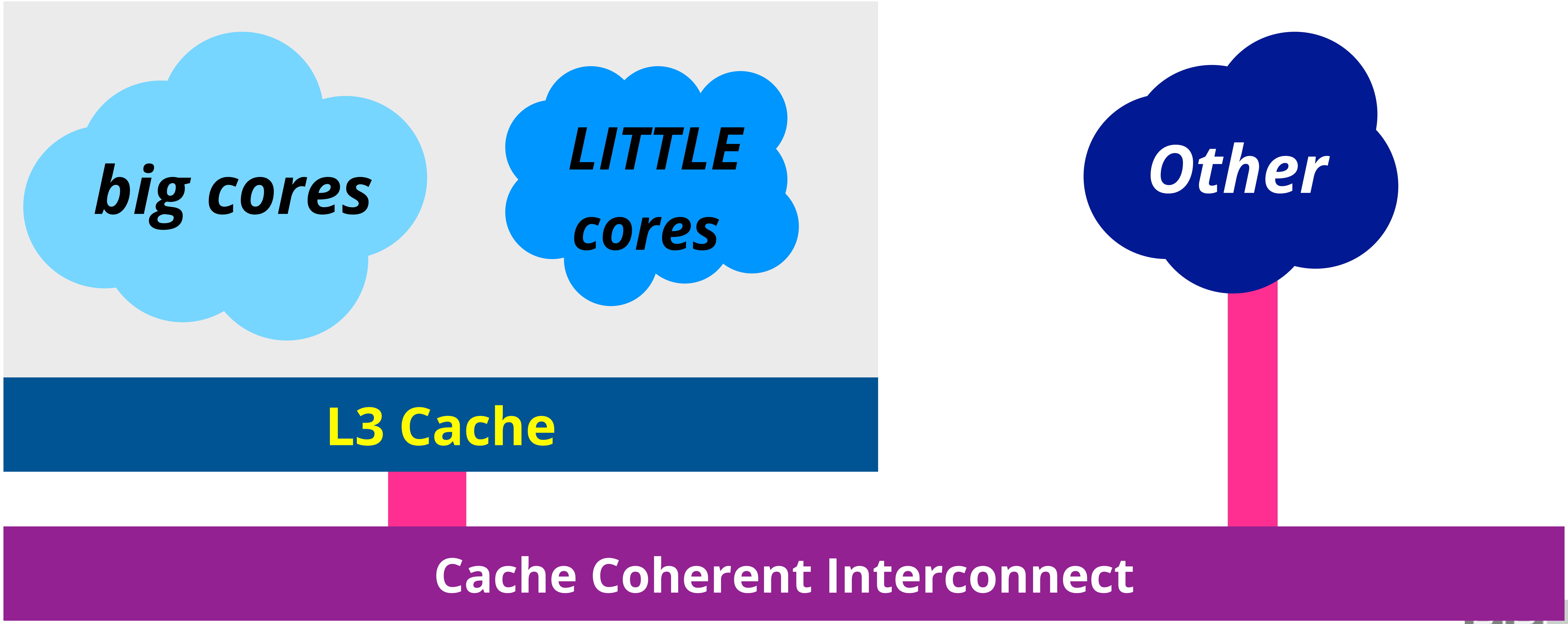
Large companies leverage this model

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

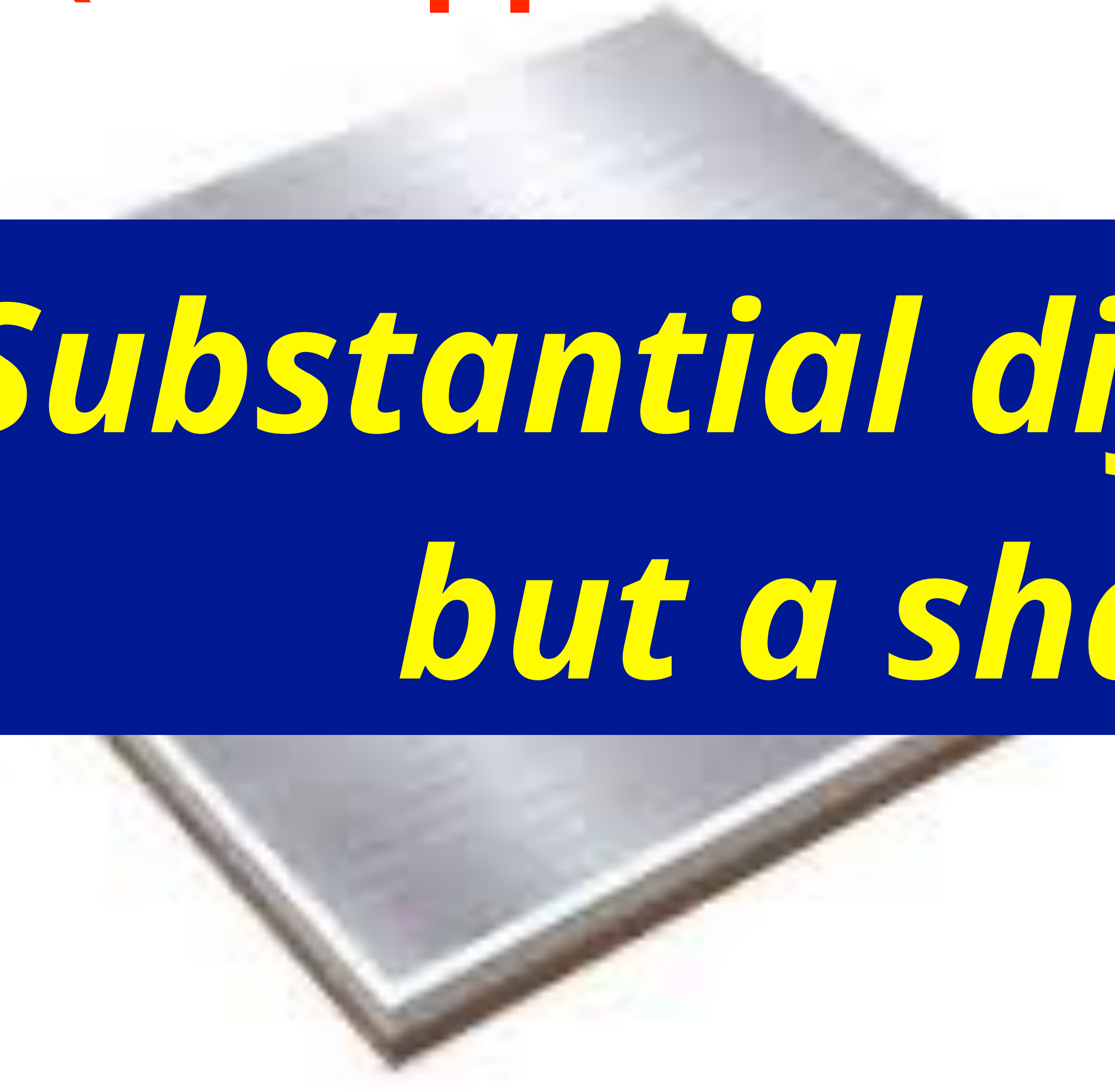
big.LITTLE DynamIQ Concept From Arm



Recent Server Level Processors

48 cores

(+4 support cores)



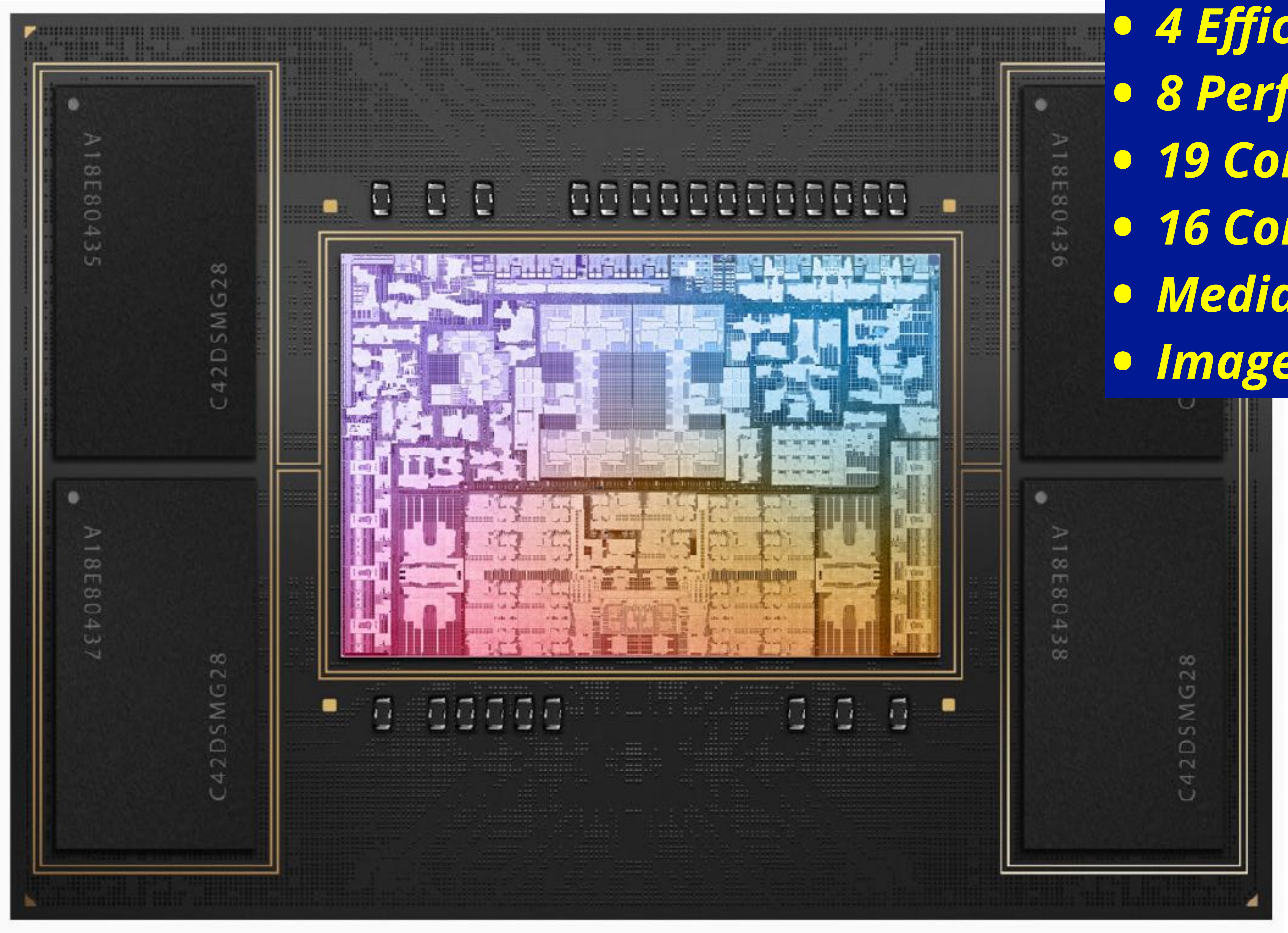
80 cores

Substantial differences under the hood, but a shared memory system

Arm based

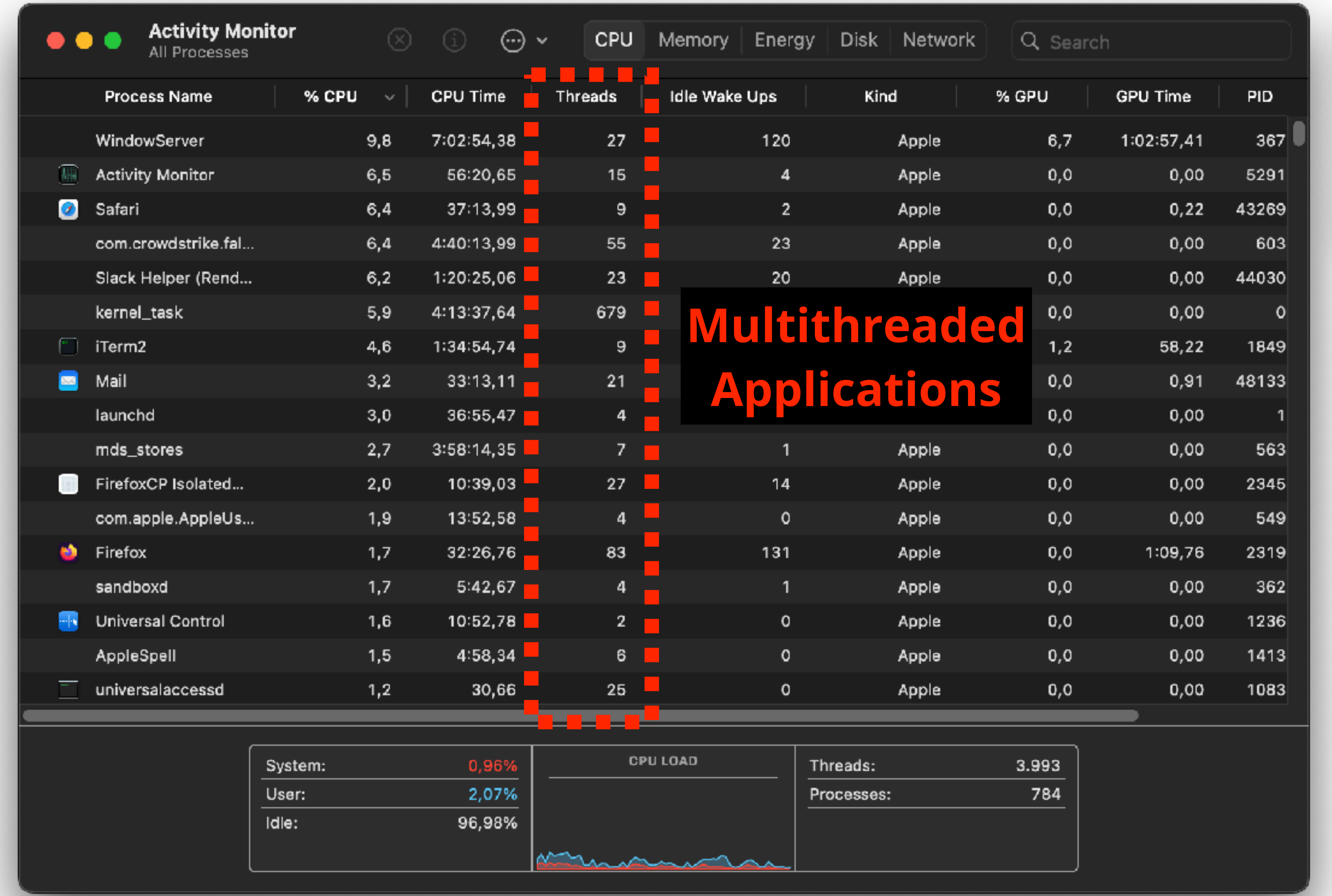
Arm based

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



Activity Monitor
All Processes

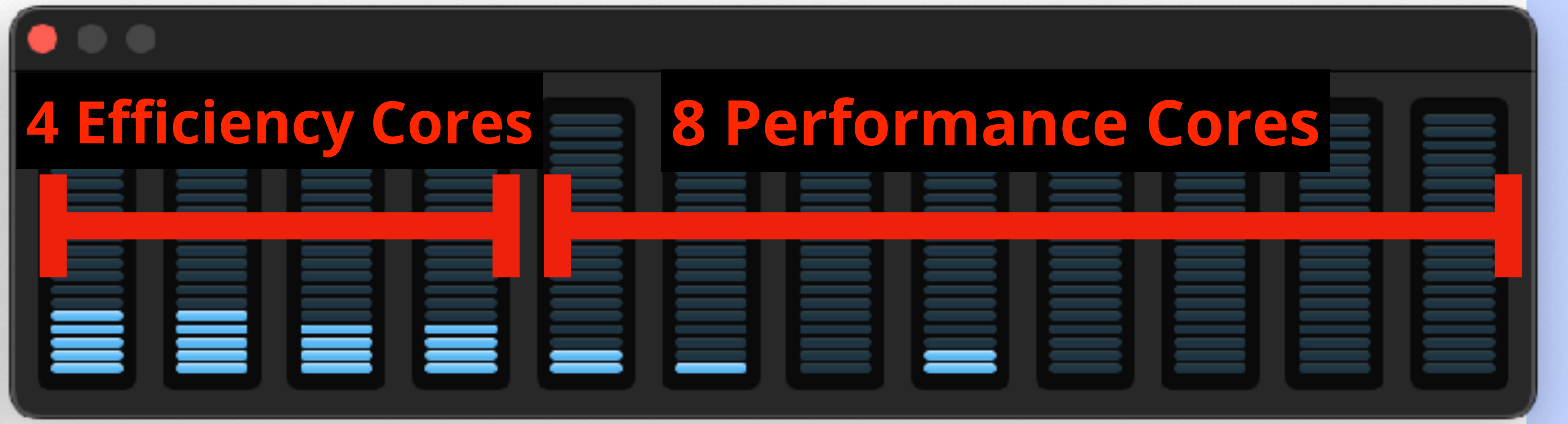
CPU Memory Energy Disk Network

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	Kind	% GPU	GPU Time	PID
WindowServer	9,8	7:02:54,38	27	120	Apple	6,7	1:02:57,41	367
Activity Monitor	6,5	56:20,65	15	4	Apple	0,0	0,00	5291
Safari	6,4	37:13,99	9	2	Apple	0,0	0,22	43269
com.crowdstrike.fal...	6,4	4:40:13,99	55	23	Apple	0,0	0,00	603
Slack Helper (Rend...	6,2	1:20:25,06	23	20	Apple	0,0	0,00	44030
kernel_task	5,9	4:13:37,64	679			0,0	0,00	0
iTerm2	4,6	1:34:54,74	9			1,2	58,22	1849
Mail	3,2	33:13,11	21			0,0	0,91	48133
launchd	3,0	36:55,47	4			0,0	0,00	1
mds_stores	2,7	3:58:14,35	7	1	Apple	0,0	0,00	563
FirefoxCP Isolated...	2,0	10:39,03	27	14	Apple	0,0	0,00	2345
com.apple.AppleUs...	1,9	13:52,58	4	0	Apple	0,0	0,00	549
Firefox	1,7	32:26,76	83	131	Apple	0,0	1:09,76	2319
sandboxd	1,7	5:42,67	4	1	Apple	0,0	0,00	362
Universal Control	1,6	10:52,78	2	0	Apple	0,0	0,00	1236
AppleSpell	1,5	4:58,34	6	0	Apple	0,0	0,00	1413
universalaccessd	1,2	30,66	25	0	Apple	0,0	0,00	1083

System:	0,96%
User:	2,07%
Idle:	96,98%

CPU LOAD

Threads:	3.993
Processes:	784



Parallel Programming Models

Programming Parallel Systems

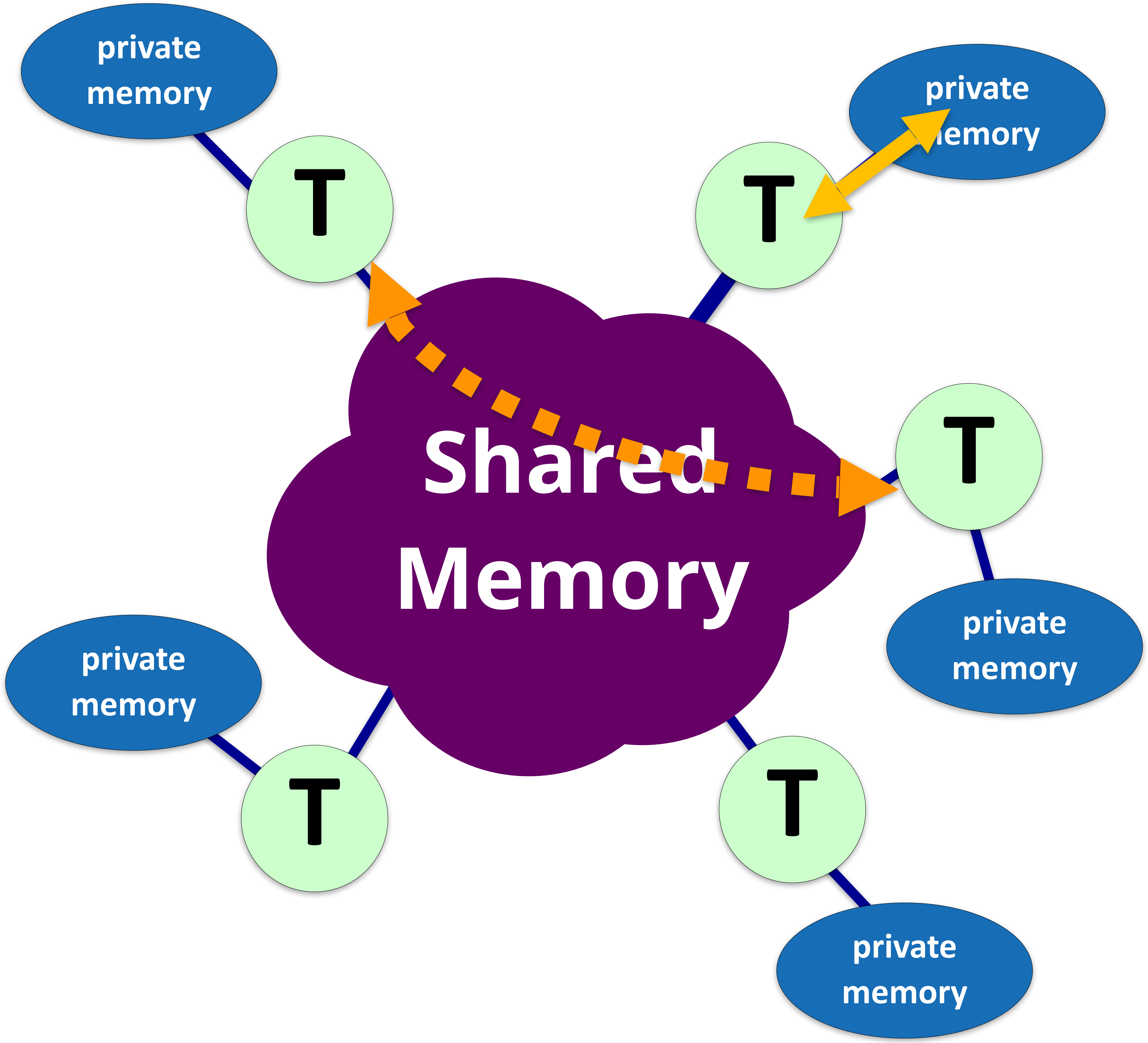
How do we program such systems?

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

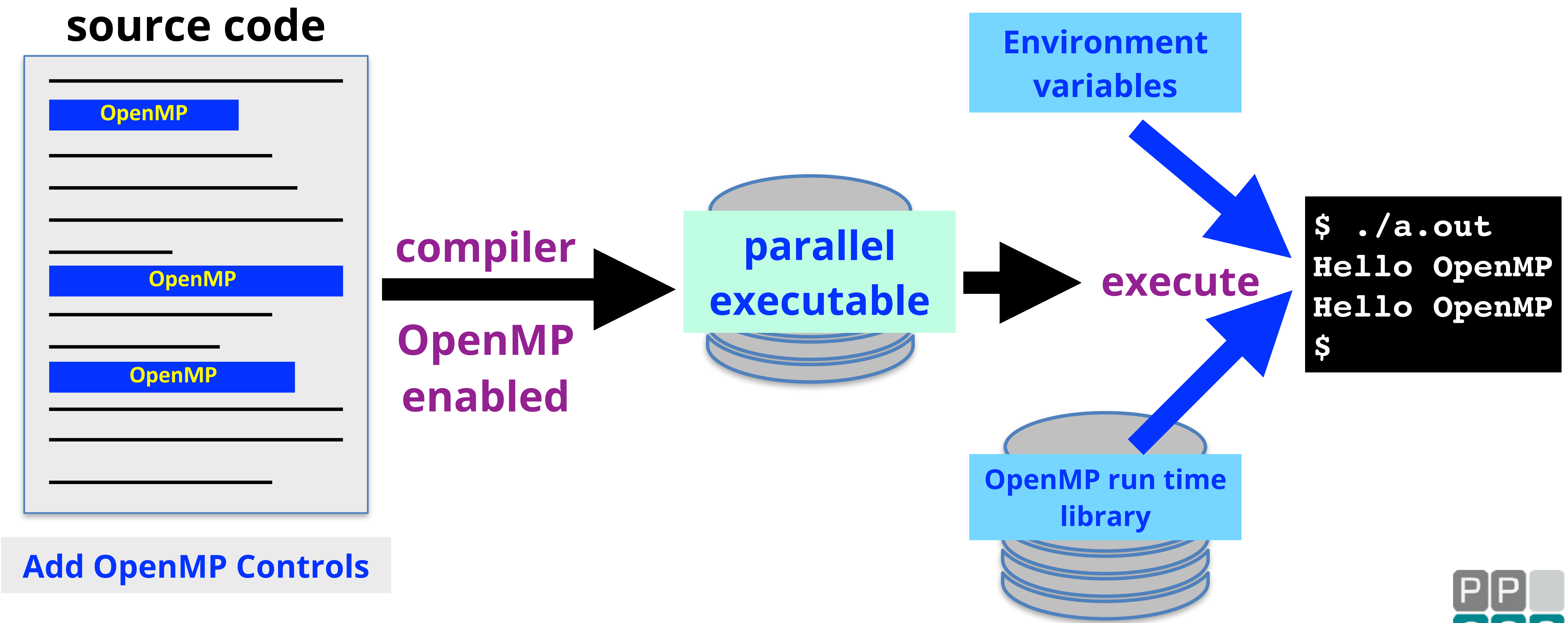
The Shared Memory Model (e.g. OpenMP)



Transparent sharing via Shared Memory

Each thread has a private memory as well

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;
```

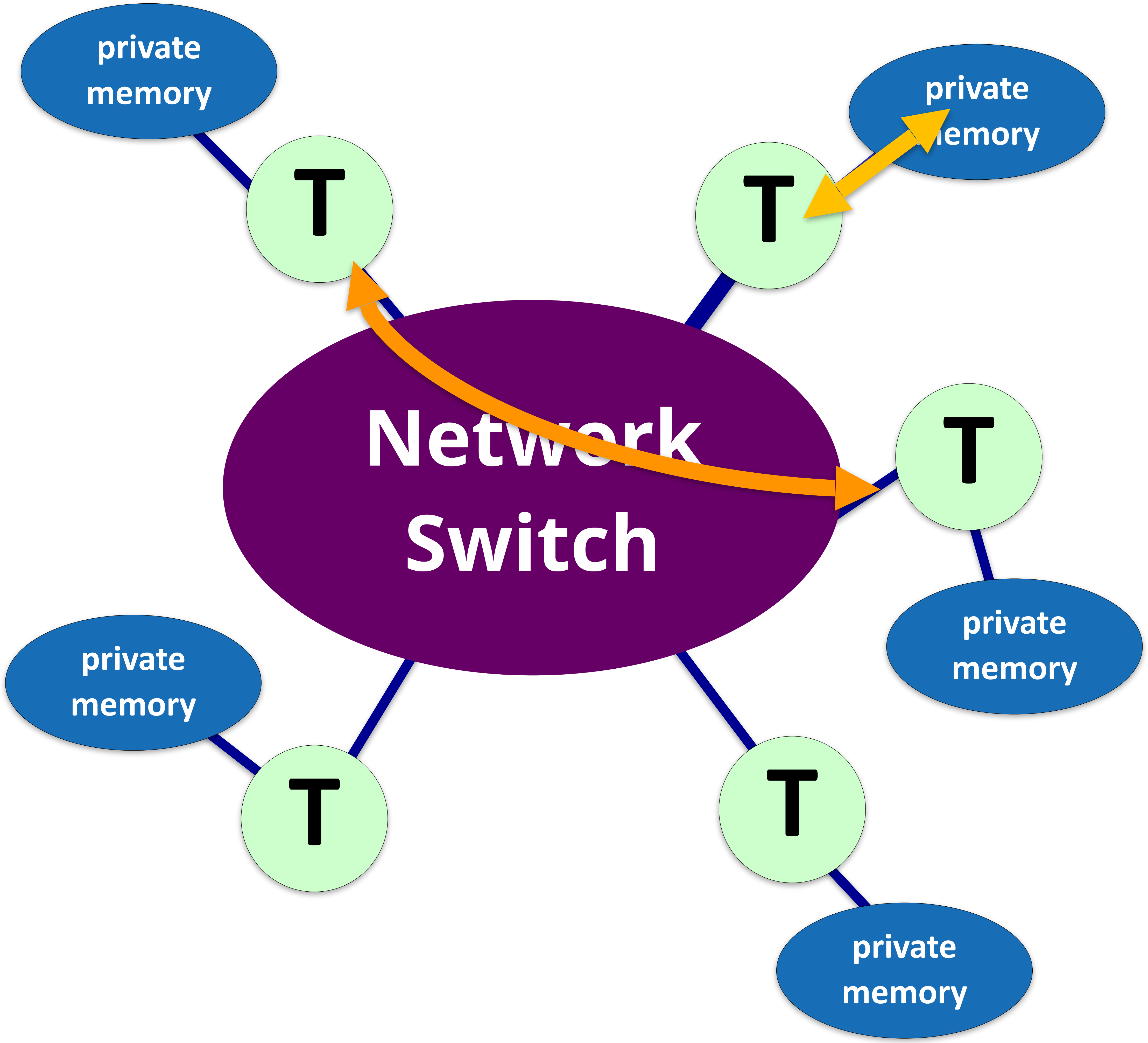
Thread 0

```
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

Connection Established

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***

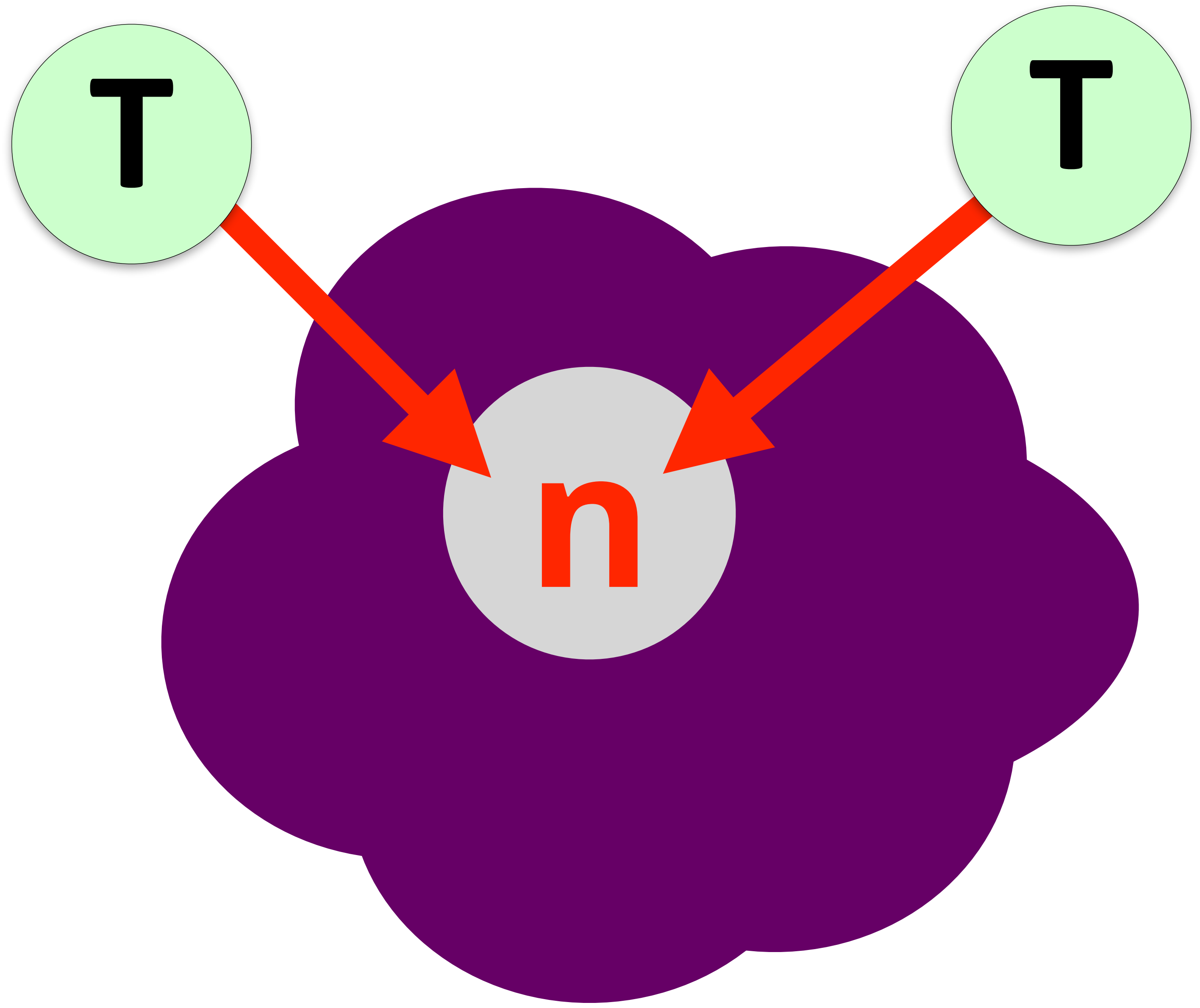
If all these 3 conditions are met, the program has a data race

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