



UNIVERSITÉ
DE NAMUR

CÉCI HPC Training 2026

Introduction to high-performance computing



Who I am

- Master of Computer Science (UNamur 2002)
- Working in HPC since 2005
- Based in the Faculty of Sciences at UNamur



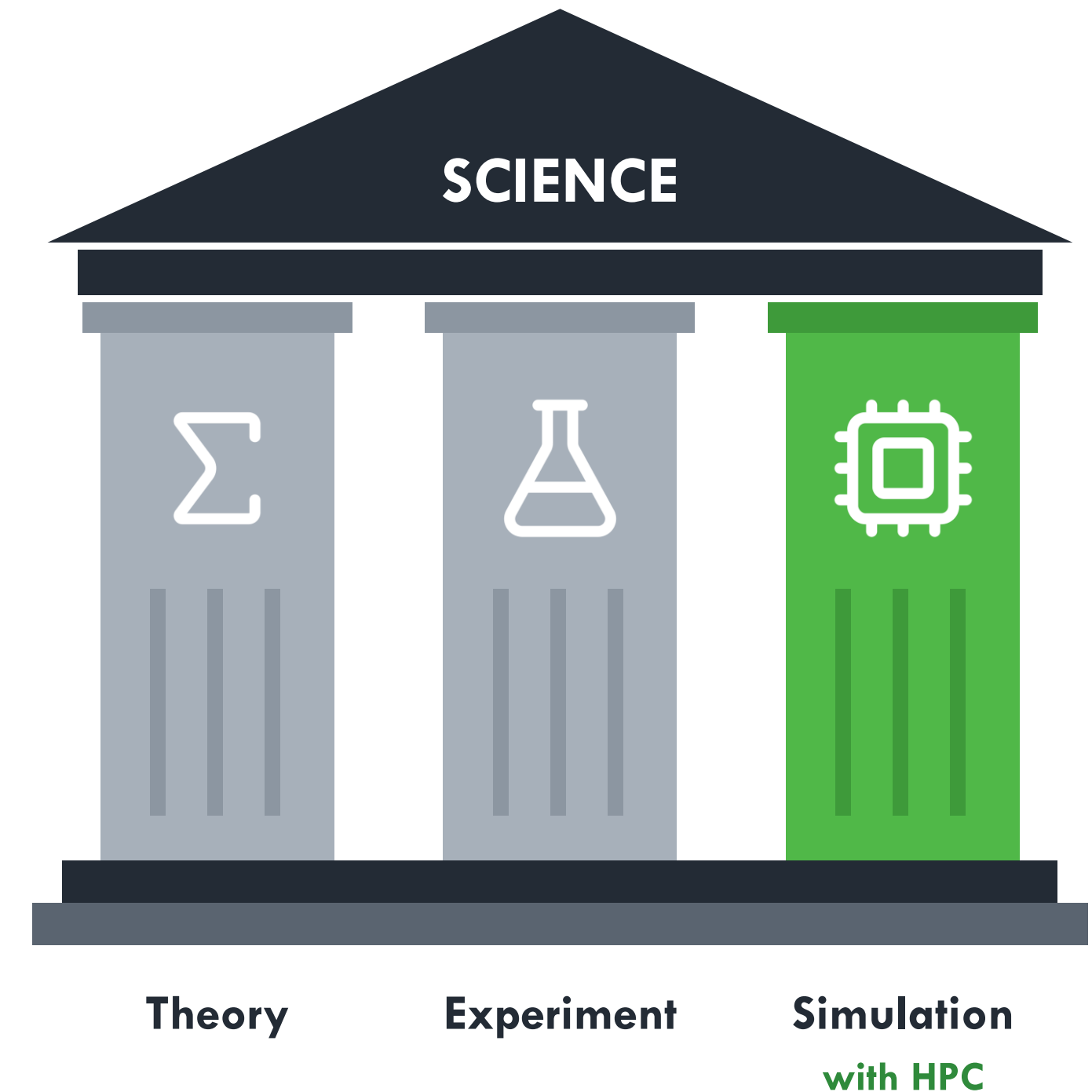
Frédéric Wautelet

High-Performance Computing Team Lead • UNamur

The Pillars of Science

- The three Pillars of Science
 - Pillar 1: Theory (Models)
 - Pillar 2: Experiment
 - Pillar 3: Numerical simulations
- Numerical simulations is an alternative to scale models and lab experiments
 - Faster
 - Cheaper
 - More flexible

→ Thanks to efficient numerical methods, **HPC can solve extremely detailed models** within reasonable timeframes



What is High-Performance Computing ?

High-Performance Computing (HPC) is the method by which scientists and engineers solve complex problems using software that require high bandwidth, low latency networking and high computing capabilities

High-Performance Computing is



An effective algorithm
and an effective
implementation



Computational power



Memory



Storage

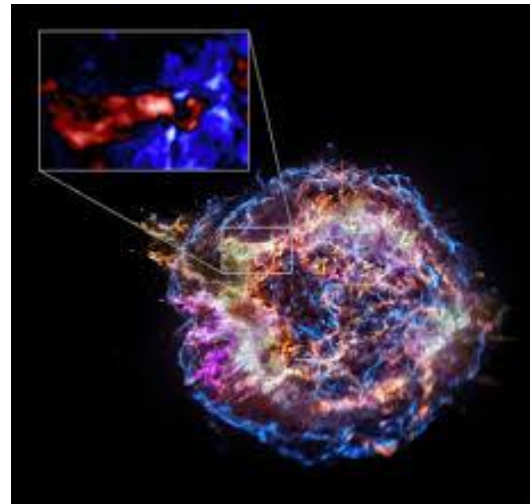
→ For HPC we use **supercomputers**



Why do you need HPC?

- A simulation that requires a lot of memory
- A program that could use many cores
- A program that takes very long to run
- To get access to specialized hardware
 - Accelerator (GPU, FPGA, ...)
 - Large memory
 - High-speed interconnect for parallel runs

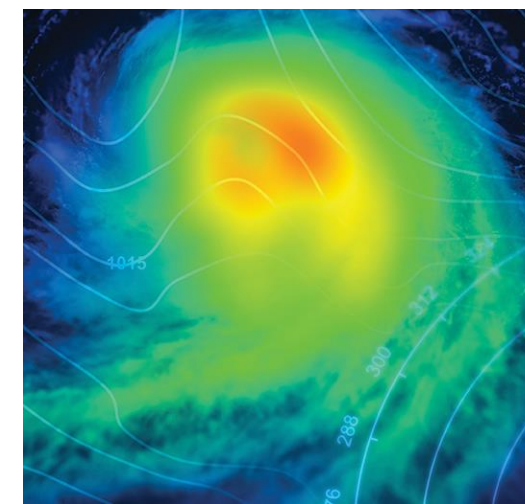
HPC use cases



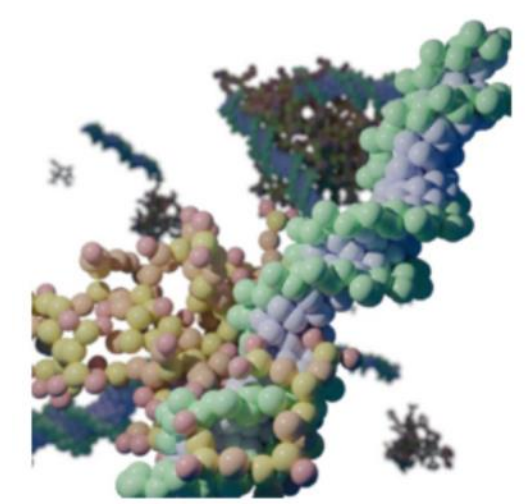
Space science



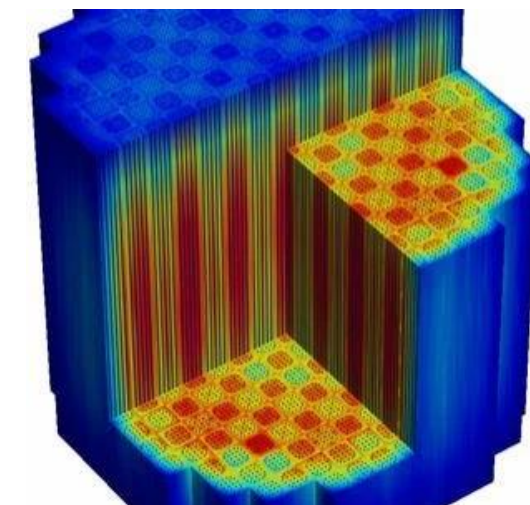
Earth science



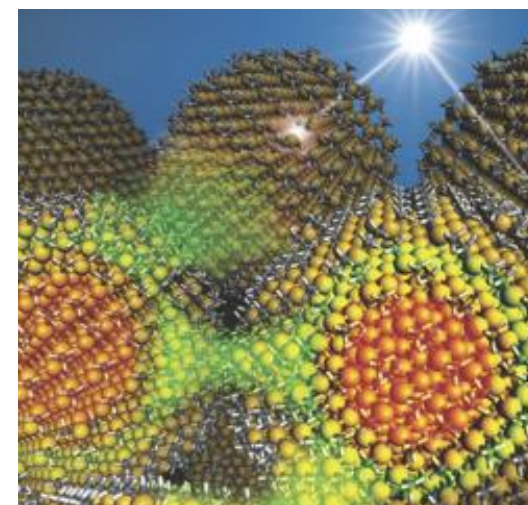
Atmospheric modeling



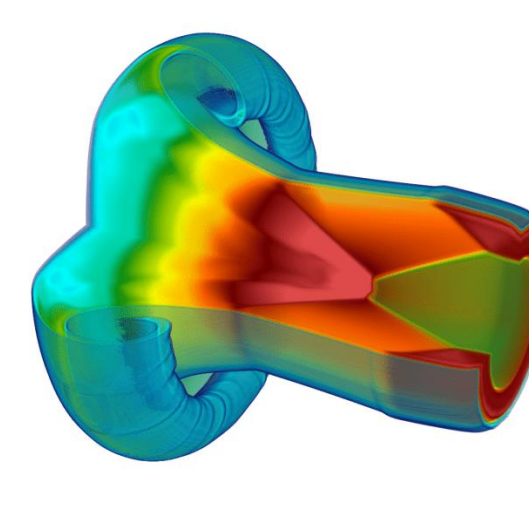
Life science



Nuclear science



Materials science



Engineering



Entertainment industry

Terminology



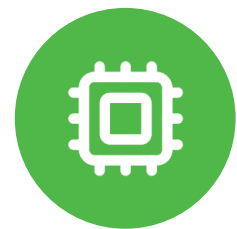
Cluster

a set of connected nodes that work together.



Node

a computer (server). It usually has one or more multi-core processors.



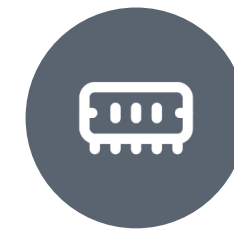
CPU

the primary computational engine of an HPC node. CPUs typically have multiple cores.



Core

an independent processing unit within a CPU.



Memory (RAM)

volatile memory shared between all cores on a node.



Drive storage

devices (spindles or SSD) that hold data persistently, even when power is off. Can be directly attached to a node or shared in storage systems.

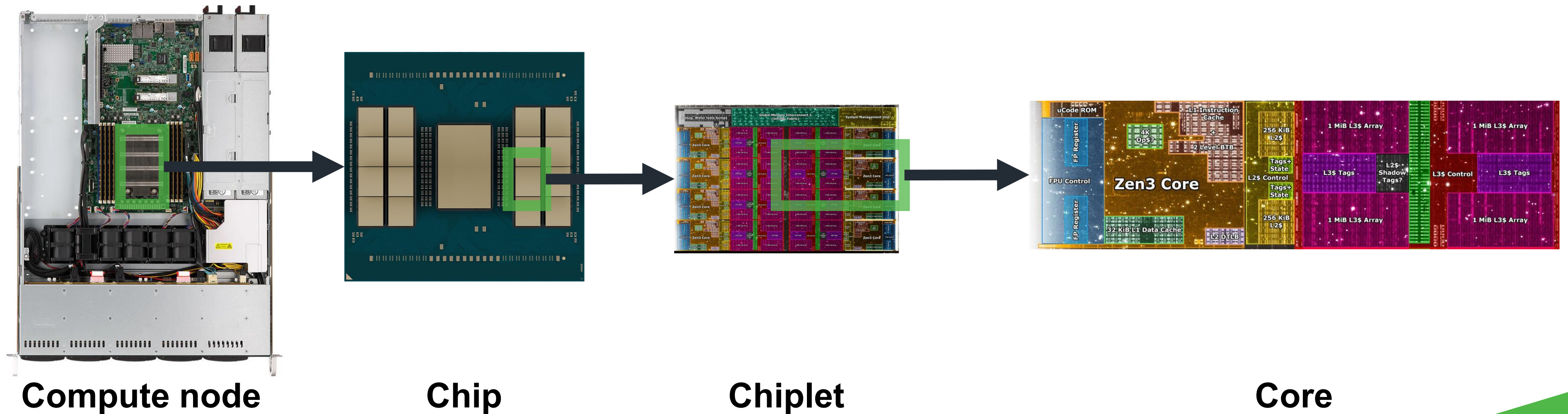


Network

connections linking together nodes and storage systems.

Multi-core processor (CPU)

- **Multi-core processor (CPU)** = single computing component with cores
- **Core** = independent processing units, which read and execute program instructions



Floating Point number

Computers store real numbers in a fixed number of bits. **More bits = more correct digits.**

format	size in memory	π stored as	typical use
FP16 half precision	16 bits	3.140625	AI training
FP32 single precision	32 bits	3.14159274	graphics, games
FP64 double precision	64 bits	3.141592653589793	science, TOP500

→ More bits: more correct digits, but **more memory and less speed**

🕒 Example: a gaming card (RTX 5090) does 104.8 TFLOPS in FP32, but only 1.64 TFLOPS in FP64

Measuring supercomputer power

FLOPS

Floating-point **O**perations **P**er **S**econd



Primary measure of supercomputer power

(memory bandwidth and I/O matter too)



Supercomputer performance is measured using specialized benchmarks that test how many floating-point operations can be performed per second

LINPACK

HPCG



Actual performance can vary based on hardware, software optimization and the type of calculation

From TeraFLOPS to ExaFLOPS

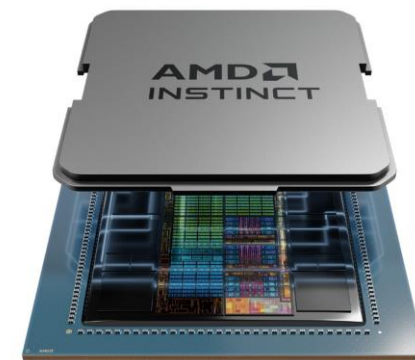
- **TeraFLOPS** = one trillion (10^{12}) floating-point operations per second
- **PetaFLOPS** = one quadrillion (10^{15}) floating-point operations per second
- **ExaFLOPS** = one quintillion (10^{18}) floating-point operations per second



NVIDIA GeForce RTX 5090

1.6 TeraFLOPS

most powerful gaming card (2025)



AMD Instinct MI300A

61 TeraFLOPS

one APU, as in Lemaitre 4 (2026)



LineShine

2.2 ExaFLOPS

#1 TOP500 (June 2026), CPU only

Double precision (FP64)

HPC is ahead of the curve

1997

ASCI Red

#1 supercomputer in the world (TOP500)

1.45 TFLOPS

 fills a room of 150 m²

 850 kW

 \$55 million ($\approx$ \$110 M today)

$\approx$

2025

 **NVIDIA GeForce RTX 5090**

a gaming card

1.64 TFLOPS

 fits in your PC

 575 W

 \$1,999

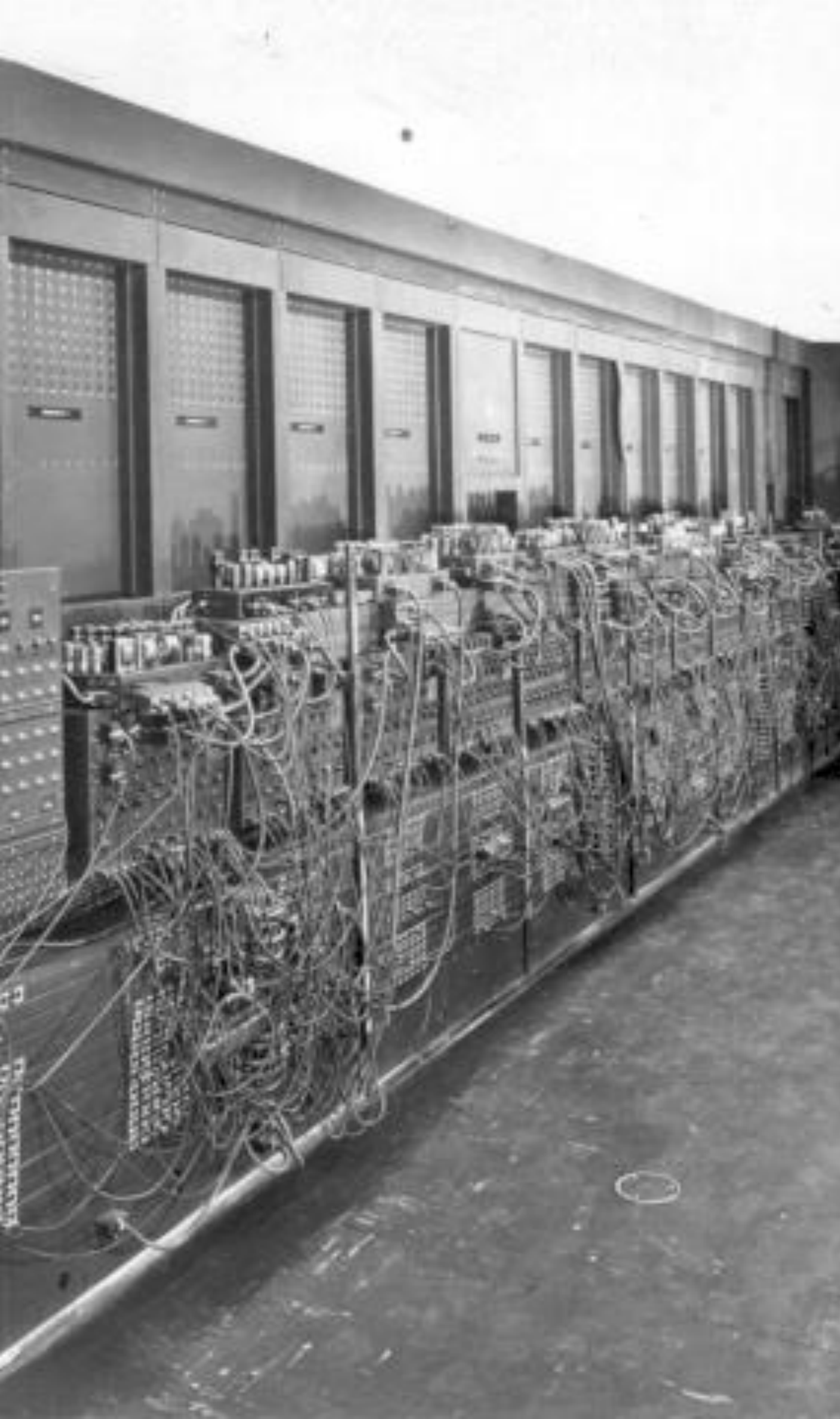
Peak TeraFLOPS in double precision (FP64)

→ Today's supercomputer is **tomorrow's desktop**

Until the 60s: the human computers

- Before the supercomputer, “human computers” were often utilized to solve mathematical problems and process data
- Hidden Figures (2016) movie
- Performance: about 0.05 FLOPS per human





1940s - 1960s: the first supercomputers

- Driven by military needs
 - Artillery tables
 - Code breaking
 - Aircraft, submarine, etc. design
 - Nuclear weapon design
- ENIAC, 1946
 - “Electronic Numerical Integrator and Computer”
 - First all-electronic, programmable computer
 - University of Pennsylvania
 - In operation until 1955
 - About 2,900 FLOPS (estimate)



1975 - 1990: the Cray era

- Vector Processors
 - Designed for operations on data arrays rather than single elements
- Shared memory multiprocessing
 - Small number (up to 8) processors with access to the same memory space
 - Issue with memory contention
- Supercomputers was custom-built systems and very expensive
- Cray 1A (1976)
 - up to 8 MB RAM
 - up to \$8 million price tag
 - 5-ton Freon cooling system
 - 160 MegaFLOPS



1990 - 2010: the cluster era

- The age of effective parallel computers begins
 - Overcome the memory contention
 - Distributed memory computers
- Intel iPSC/1 “Hypercube” (1985)
 - up to 128 nodes
 - 80286 processor + 80287 math co-processor
 - 512K RAM
 - 2 MegaFLOPS
 - \$500,000 price tag (128 nodes version)

1990 - 2010: the cluster era

- “Beowulf” clusters
 - Named from an old English poem
 - Build using commodity “off-the-shelf” components
- First Beowulf computer cluster at NASA Goddard Space Flight Center (1994)
 - 16x Intel 486DX PCs
 - 16 MB RAM
 - 1 GigaFLOPS
 - \$50,000 price tag



2000 – 2020: the GPU and hybrid era

- Accelerator-based architectures
 - CPU and accelerators coprocessing
 - Offload accelerator-intensive workloads
- Very large number of nodes each with:
 - Many-core processors
 - RAM shared by all cores
 - Accelerators (mainly GPUs)
- Computation speed often limited by data-movement rate
- Hybrid programming on multicore architectures is more complex

2000 – 2020: the GPU and hybrid era

- Roadrunner (Los Alamos – 2008)
 - 122,400 cores
 - \$100 million cost
 - First supercomputer to reach one PetaFLOPS
 - TOP500 #1 from June 2008 to June 2009
 - 2 MW power

2020 – now: the age of AI

- Rise of AI/ML lead to a surging demand for computer power
- New classes of workloads appeared
 - Large neural networks, huge training/inference data, massive model databases
 - Require not just faster hardware, but new software frameworks, optimized data access, and integration of HPC + AI workflows
 - Example: AlphaFold2



TOP500.org

- Maintains a list of fastest supercomputers in the world
- Based on the LINPACK benchmark program
- A new list comes out every June and November
- LineShine (Shenzhen, China)
 - #1 (June 2026)
 - 13.8 million cores
 - 2,2 ExaFLOPS on LINPACK
 - CPU only: 304-core LX2 processors
 - 42 MW power





Supercomputers use a lot of power!

- LineShine (#1 TOP500) uses as much electricity as 100,000 Belgian homes!
- And this pales in comparison to AI datacenters
 - **XI** Colossus 2, ~1 TW (over 20× LineShine)
- Lawrence Berkeley National Laboratory: US datacenters used 4.4% of US electricity in 2023, and could use up to 12% by 2028

Green500

- Rank supercomputers in terms of energy efficiency
- Performance per Watt (GFLOPS/W)
- <https://www.top500.org/lists/green500>
- Green500 #1 (June 2026): KAIROS
 - CALMIP, University of Toulouse – CNRS, France
 - 13,056 cores, 3.05 PetaFLOPS
 - 73.3 GFlops/watt
 - For comparison: #1 TOP500 (LineShine): 52 GFlops/watt

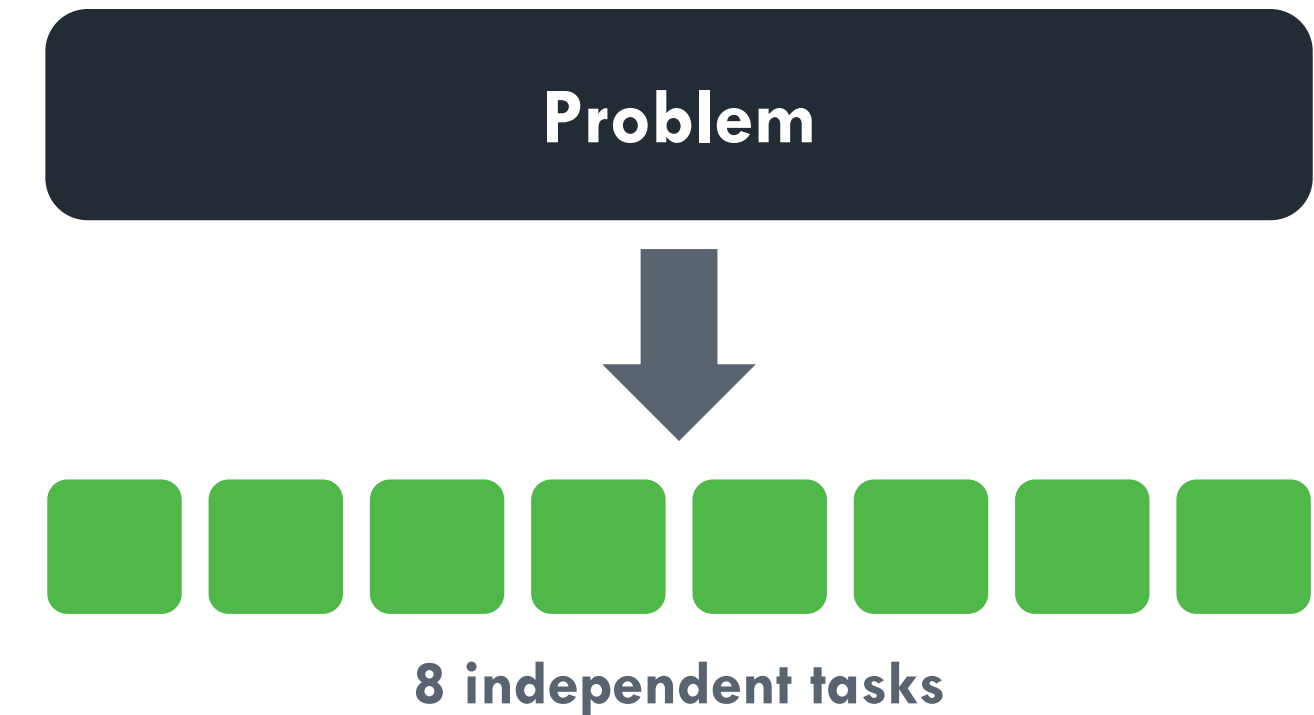
Parallel computing

- Sequential systems are painfully slow
 - Calculations may take days, weeks, years ...
 - More cores can get job done faster
- The key of HPC is parallelism
- The concept is simple:

→ **Parallelism** = use multiple cores for a single problem

From problem to solution

- Analyze the problem:
 - split problem in independent tasks
 - find solutions for each task
- Problems have certain amount of potential parallelism.



Sequential execution



Tasks depend on each other

OR



No parallel resources: one core, little memory, slow network

8 tasks on 1 core



→ Sequential: 8-time steps

Embarrassingly parallel



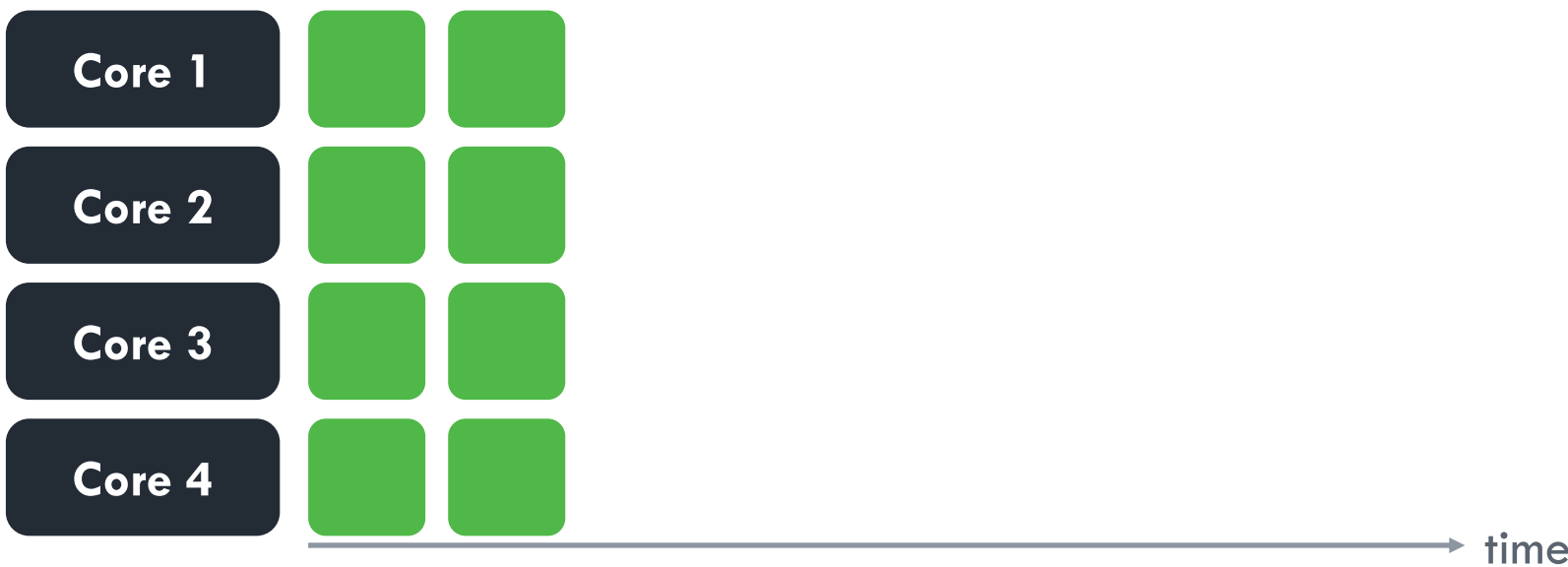
Tasks are independent, in any order

AND



Many cores, enough memory and bandwidth

the same 8 tasks on 4 cores



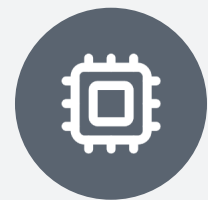
→ Parallel: 2 steps, 4× faster

In the real world



Tasks are not all independent

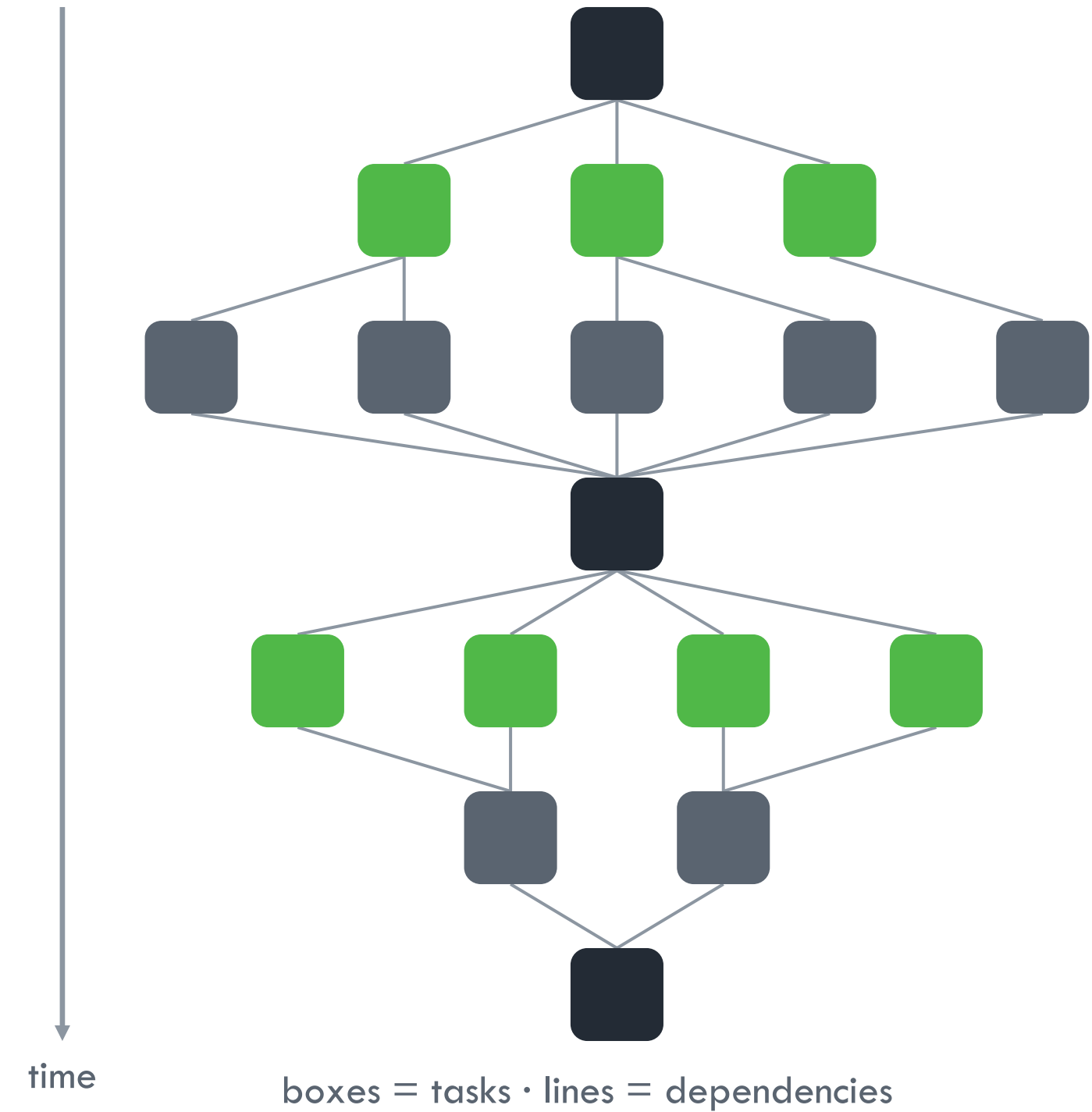
AND



Resources are limited

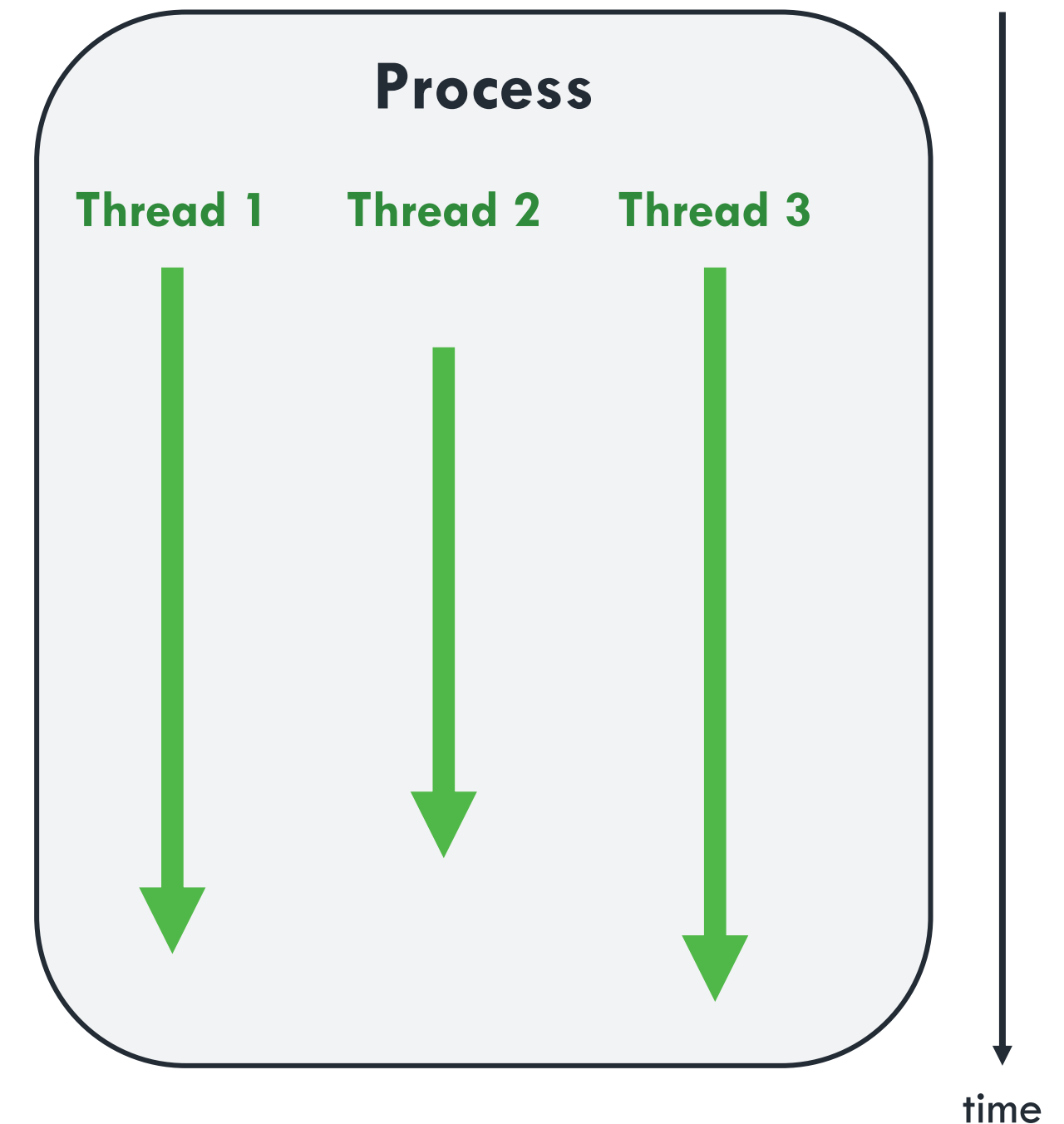
- Multi/many cores machine
- Limited memory
- Limited communication bandwidth

→ Look for a **good compromise**



Process and thread

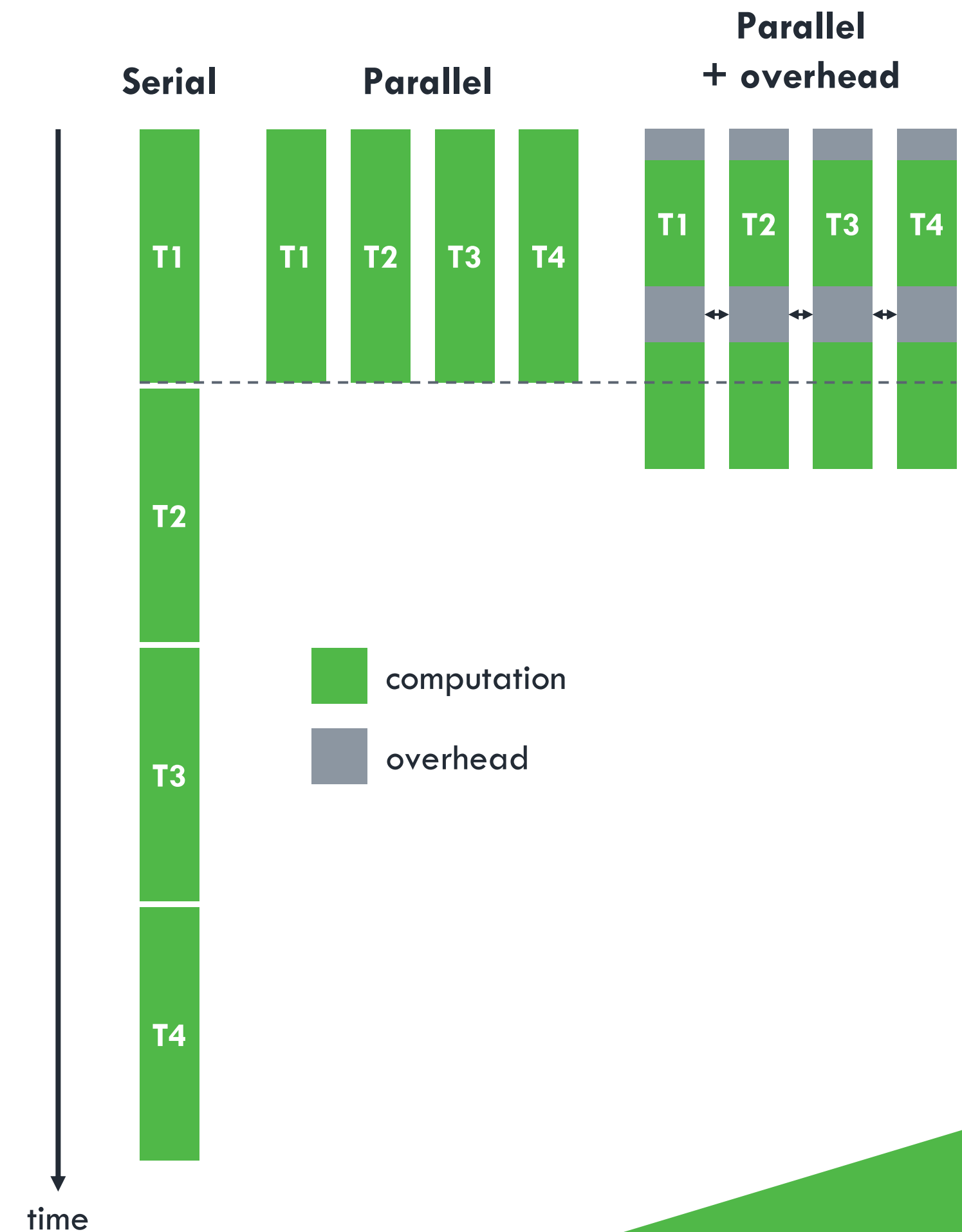
- Software concepts
- A thread is essentially just **an execution of a sequence of instructions**
- A process is a **collection of one or more threads**
- **A parallel program will execute threads in parallel**
- In HPC, one thread is executed on one core



Parallel Overhead

- Parallelization comes with some **costs**:
 - Communication between the threads
 - Synchronization
 - Resources management

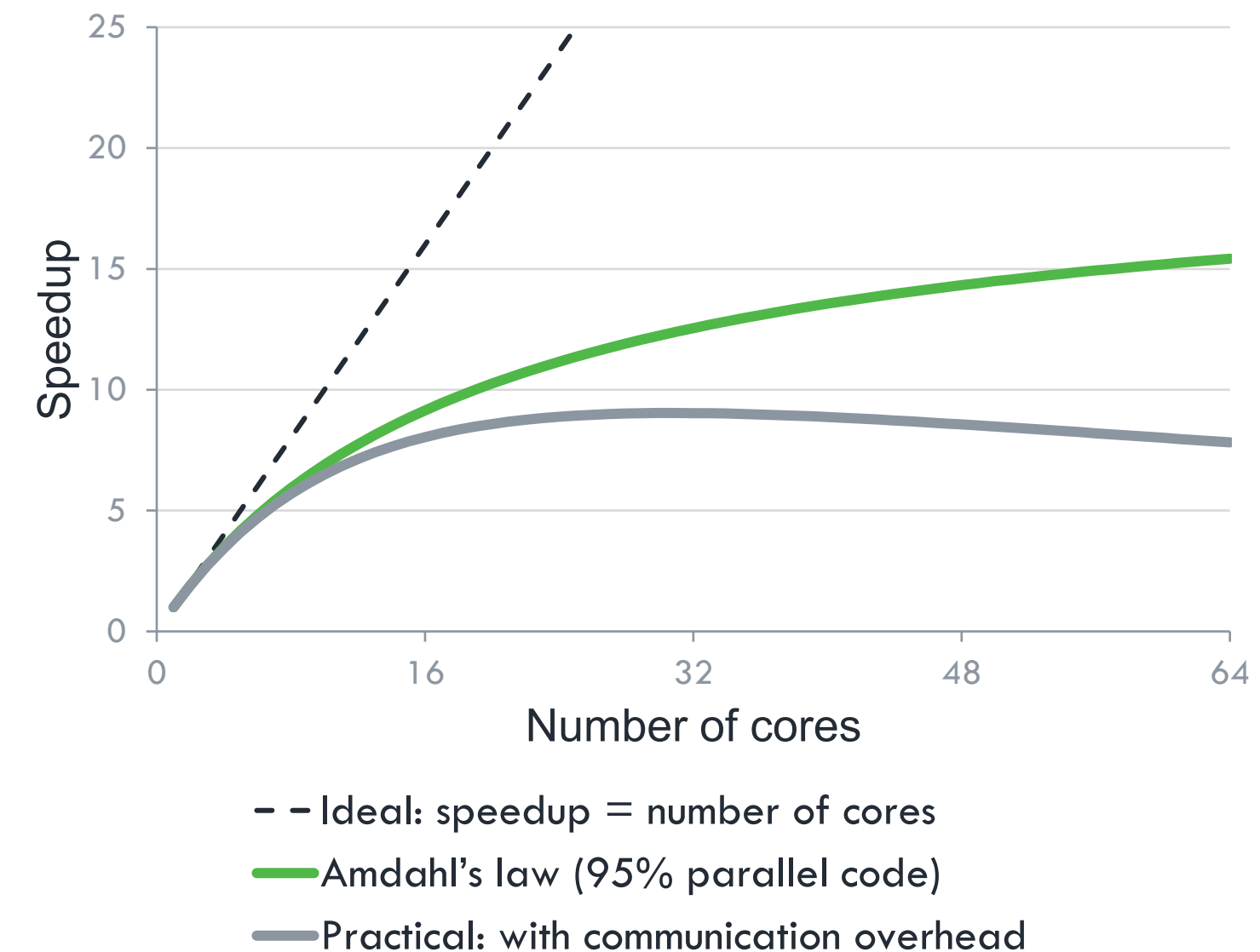
→ To get efficient parallelization, you need to **minimize the overhead**



Amdahl's Law

- Use to estimate the parallel program performance speedup when using several cores
- In the real world, speedup is limited by:
 - the serial part of the program, whose time remains constant
 - the hardware factors (communication bandwidth, memory speed, ...)
 - the parallel overhead

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$



Explicit vs. implicit parallelism



Explicit Parallelism

Done by the developer

- Annotate the tasks for parallel execution or explicitly assign tasks to cores
- Control the execution and synchronization points



Pro

- Can be very efficient



Cons

- Developers are responsible for all details
- Developers must have deep knowledge of the computer architecture to achieve maximum performance



Implicit Parallelism

Done by the compiler

- Automatically detects potential parallelism in the software



Pro

- Frees the developer from the details of parallel execution
- Flexible solution



Cons

- Not always efficient



Parallel Programming Models

Two dominant explicit parallel programming models

Shared memory

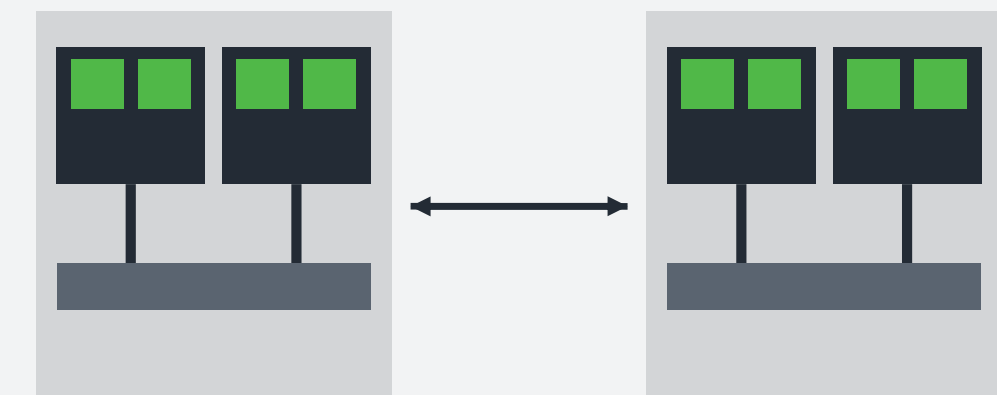


- Threads share one memory
- Runs on one node

OpenMP

Distributed memory

(Message Passing)



- Each process has its own memory
- Runs on many nodes

MPI

Shared memory

- Multiple threads operate independently but share the same memory
 - Single address space
 - No need to transfer data from one thread to another
 - Changes in a memory location made by one thread are visible to all other threads
 - OpenMP

→ execution on a **single node!**



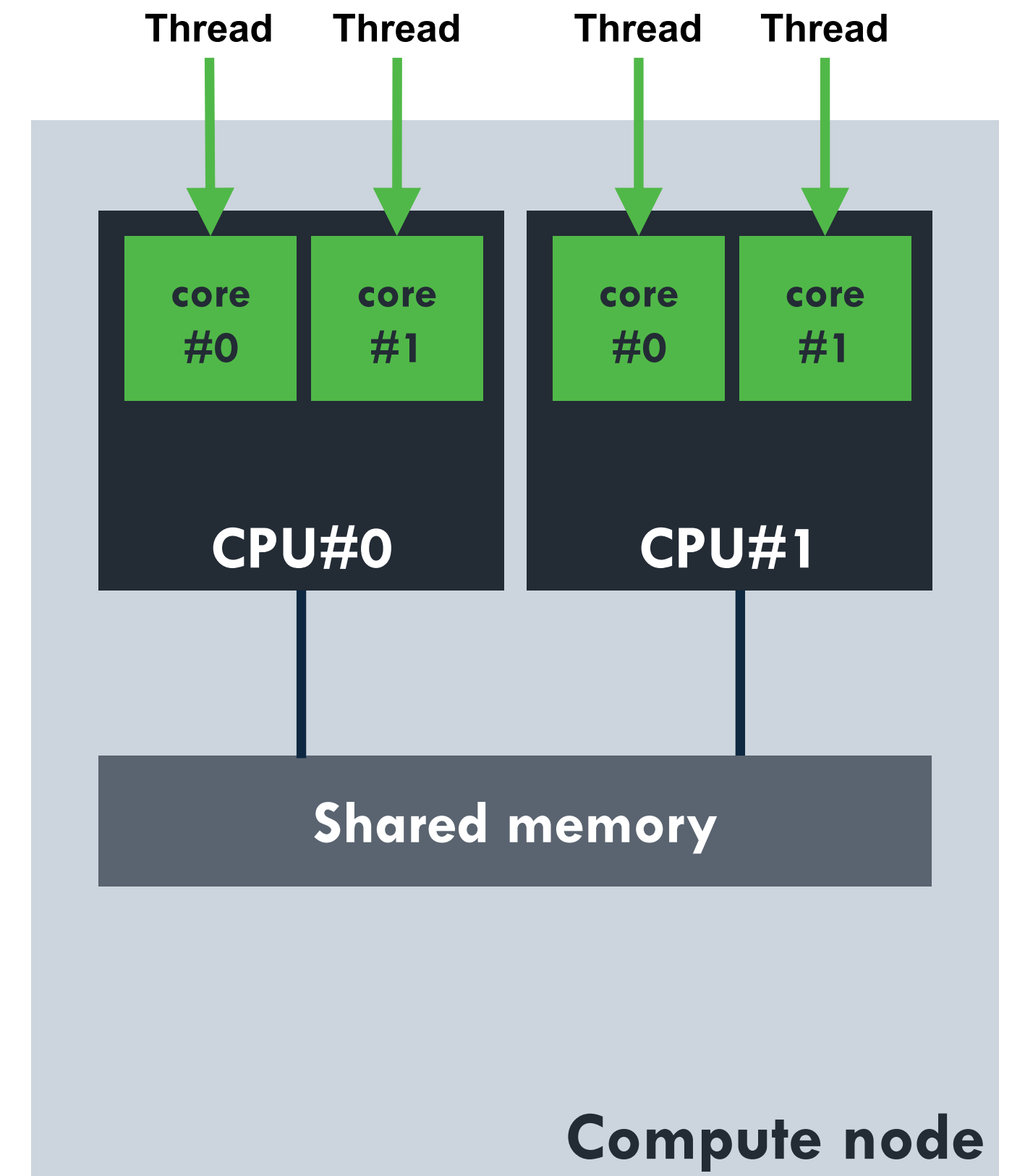
Pro

- Data sharing between threads is fast
- “Easy” to use



Con

- Scalability is not good
- Programmer is responsible for specifying synchronization



Distributed memory : Message Passing

- All data is private to the processes
 - Separate address spaces
 - Data is shared by exchanging messages
 - Data transfer requires cooperative operations by each process
 - MPI is the "de facto" industry standard for message passing

→ can run on **multiple nodes**



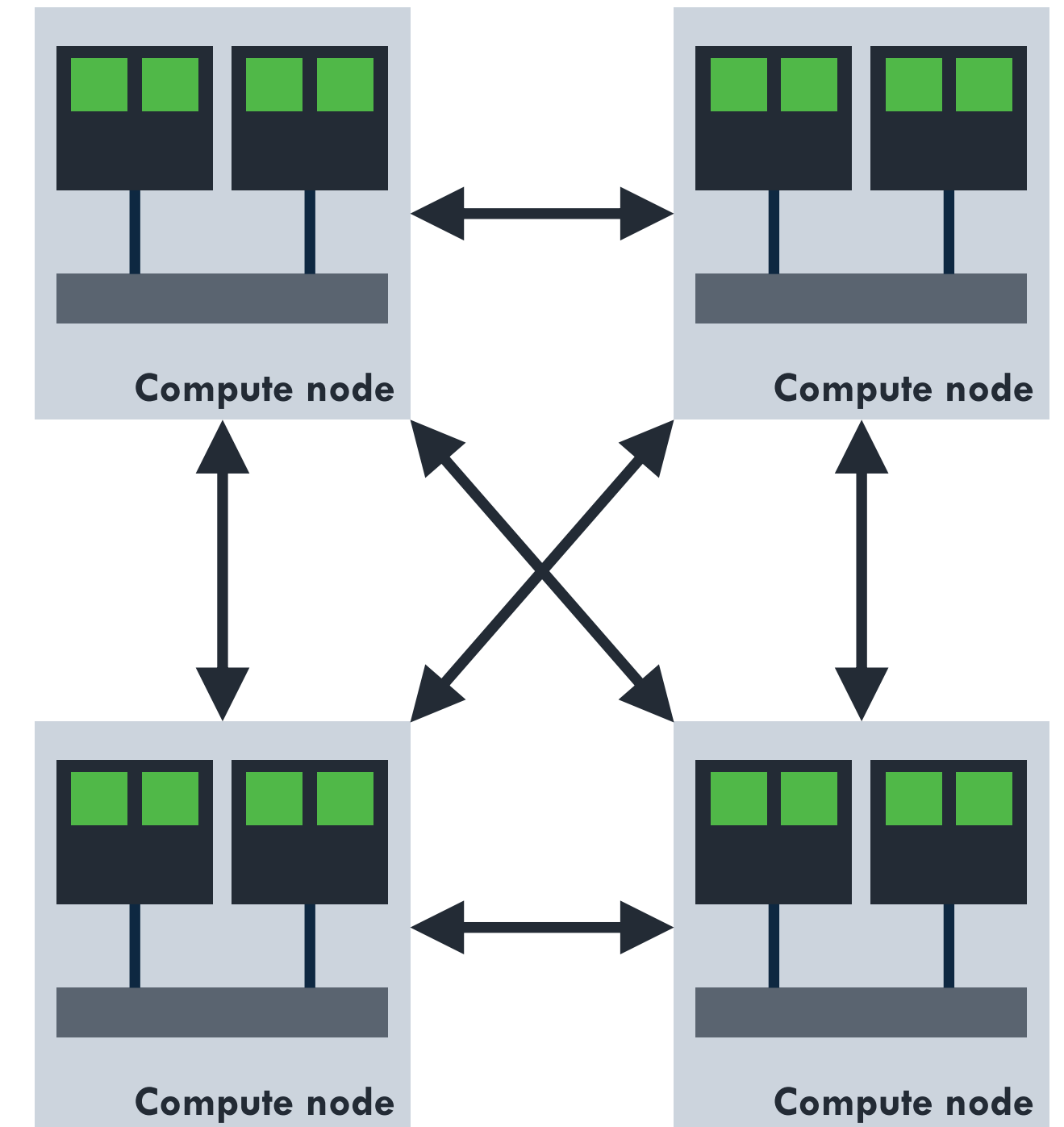
Pro

- Good scalability



Con

- Complex
- Converting a program to this model requires a complete rewrite



Accelerated computing

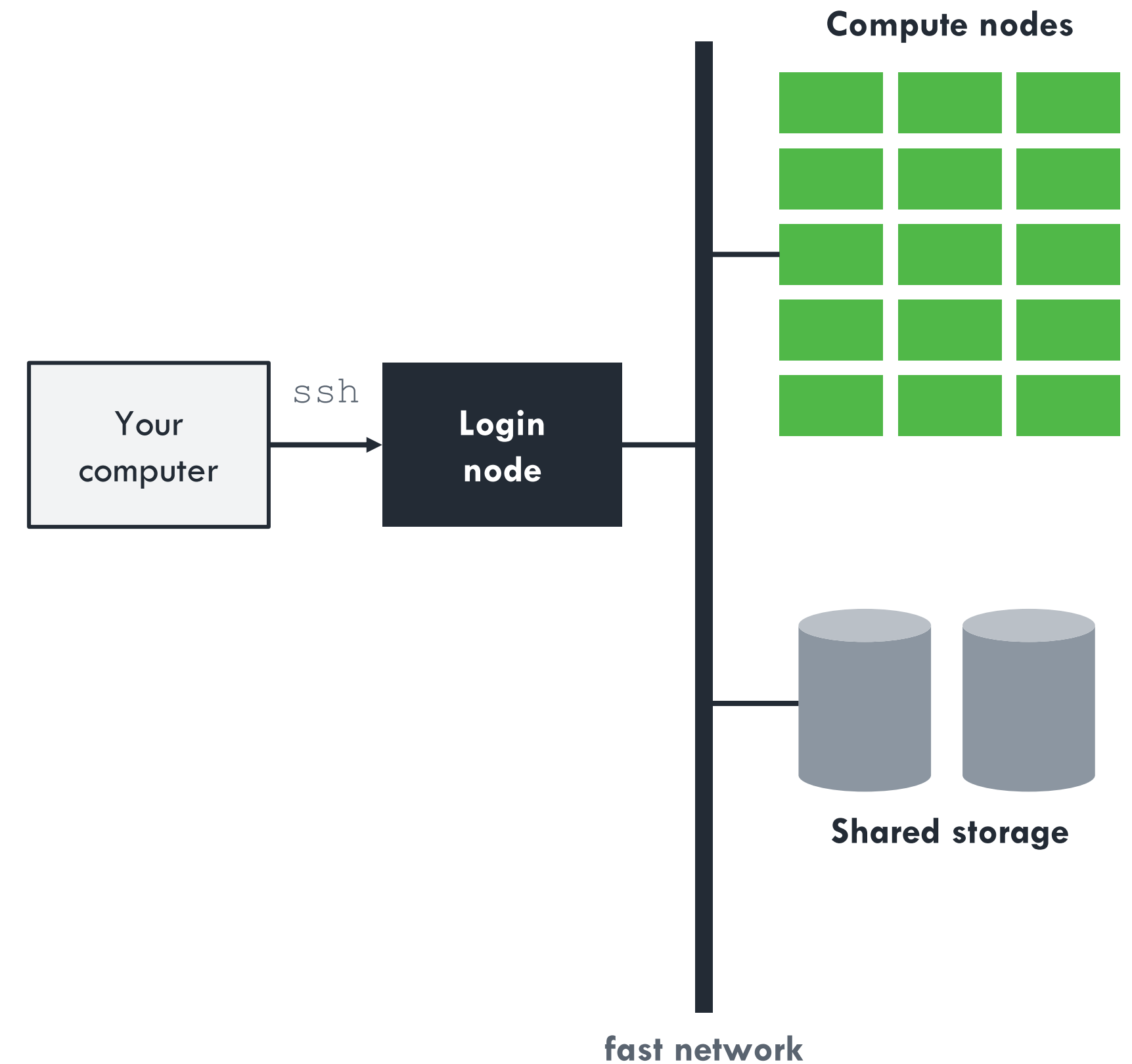
- Use of specially designed hardware and software to speed computing tasks
 - Up to several order of magnitude speedup in some cases
- Mostly GPUs (graphics processing units) but also
 - Application-Specific Integrated Circuits (ASICs)
 - Field Programmable Gate Arrays (FPGAs)
- Require the use of dedicated software stacks
- They can make your code run faster but can be hard to program

Parallel programming in practice

- Parallel programming needs **compilers and toolboxes**
- Many scientific software comes now in a parallel version
- Not so easy to set-up
- On HPC cluster, the job scheduler take care of asking resources correctly

HPC cluster

- One or several login nodes
- Many compute nodes
- One or several shared storage systems
- A high-performance network
- A software stack
 - Scientific software
 - Compilers and libraries





HPC cluster

- HPC cluster is like your personal computer
 - Hardware is very similar to personal computer
 - Can run the same software as your personal computer
- But it is a multi-user environment
 - You will need to connect and log in
 - You share storage space
 - You share computational resources
- Policies in place to ensure the fair sharing of resources
 - All program execution must be submitted to the job scheduler

Login node

Can be used to:

- ✓ Develop your code
- ✓ Manage your files
- ✓ Check your storage usage
- ✓ Manage your jobs
- ✓ To pre-/post-process your data/jobs
- ✓ To visualize your data



NEVER

run programs on the login node

Login nodes are shared resources
Computation-heavy jobs **MUST** be submitted to
the scheduler and not run directly.

Command line

```
$ ls
data  results  run.sh

$ module load Python

$ sbatch run.sh

Submitted batch job 123456

$ squeue --me
```



HPC clusters usually don't have graphical desktop environments installed



Users interact with the HPC clusters by typing commands in a terminal



A program (the “shell”) interprets these commands and communicates with the operating system



Powerful and flexible



A bit of practice is needed

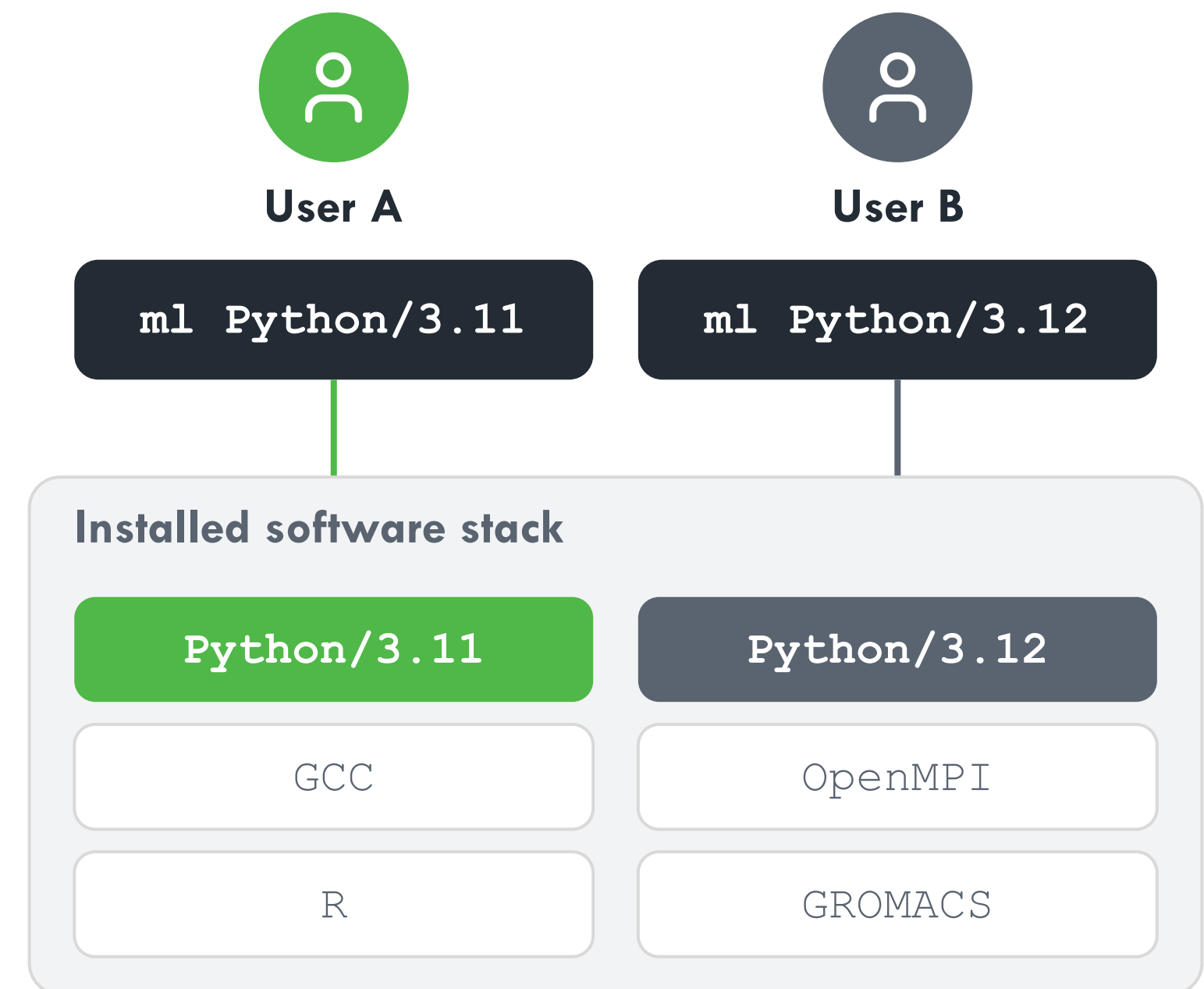
Concept of “modules”

- Large software stacks are often installed on HPC clusters
- Different users may need different tools, or different versions of the same software
- To avoid conflicts between the different software or different versions of the same software

Modular approach:

- ✓ Software packages/tools are available as “modules”
- ✓ Environment variables are set accordingly
- ✓ Conflicts are avoided

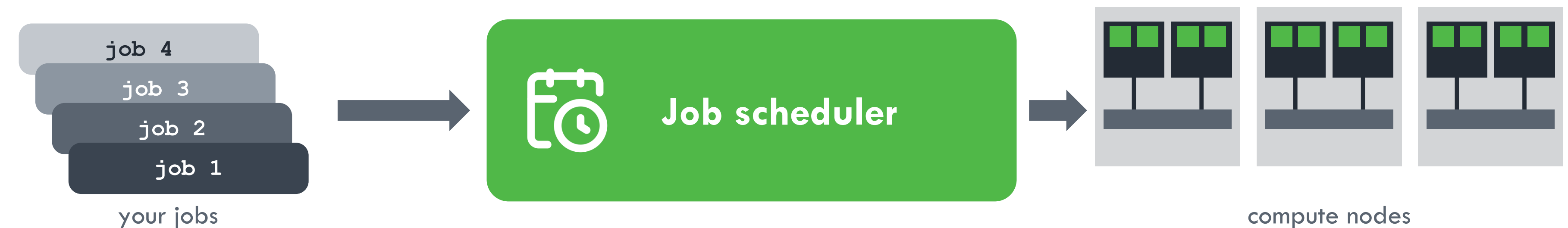
Example: two users, two versions



Batch processing

Unattended execution is the typical HPC usage

The job scheduler is a program that manages unattended program execution (batch processing)



1 checks the resources available

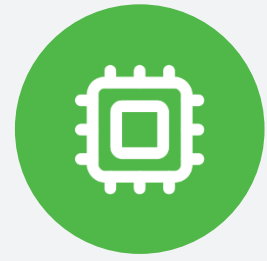
2 prioritizes the jobs and controls the execution order

3 schedules the execution of jobs on compute nodes when requested resources are available

→ prepare your program so that it can run **without any intervention**

Job scheduler

You will need to specify the resources needed



of nodes/cores



amount of memory



expected run time
(wall-clock time)



GPUs



...



This is done in a **job script**



The job scheduler relies on this information



BE ACCURATE! a misuse of the resources may affect all the users

Job script

```
#!/bin/bash
#SBATCH --job-name=demo
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --partition=gpu
#SBATCH --gres="gpu:1"

ml purge
ml LIST_THE_MODULES_YOU_NEED_HERE

srun aprogram < ~/mydata001
```

Job parameters
+
Needed resources

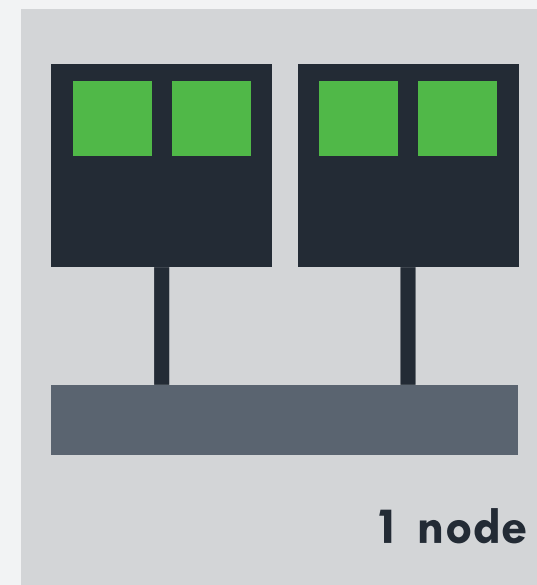
Modules

Job steps

Parallel jobs

You need to specify the number of nodes and cores you need

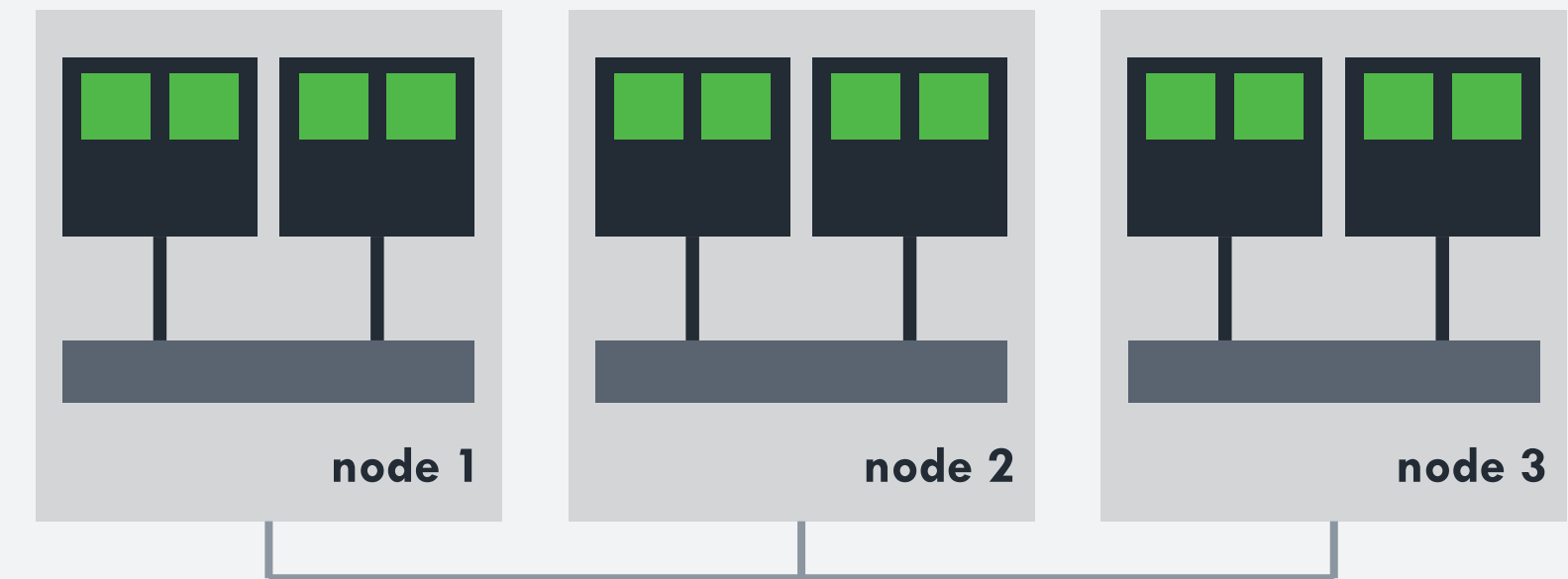
Shared memory · OpenMP



all threads share its memory

A shared memory program (e.g. OpenMP) requires cores on one single node

Distributed memory · MPI



network: messages

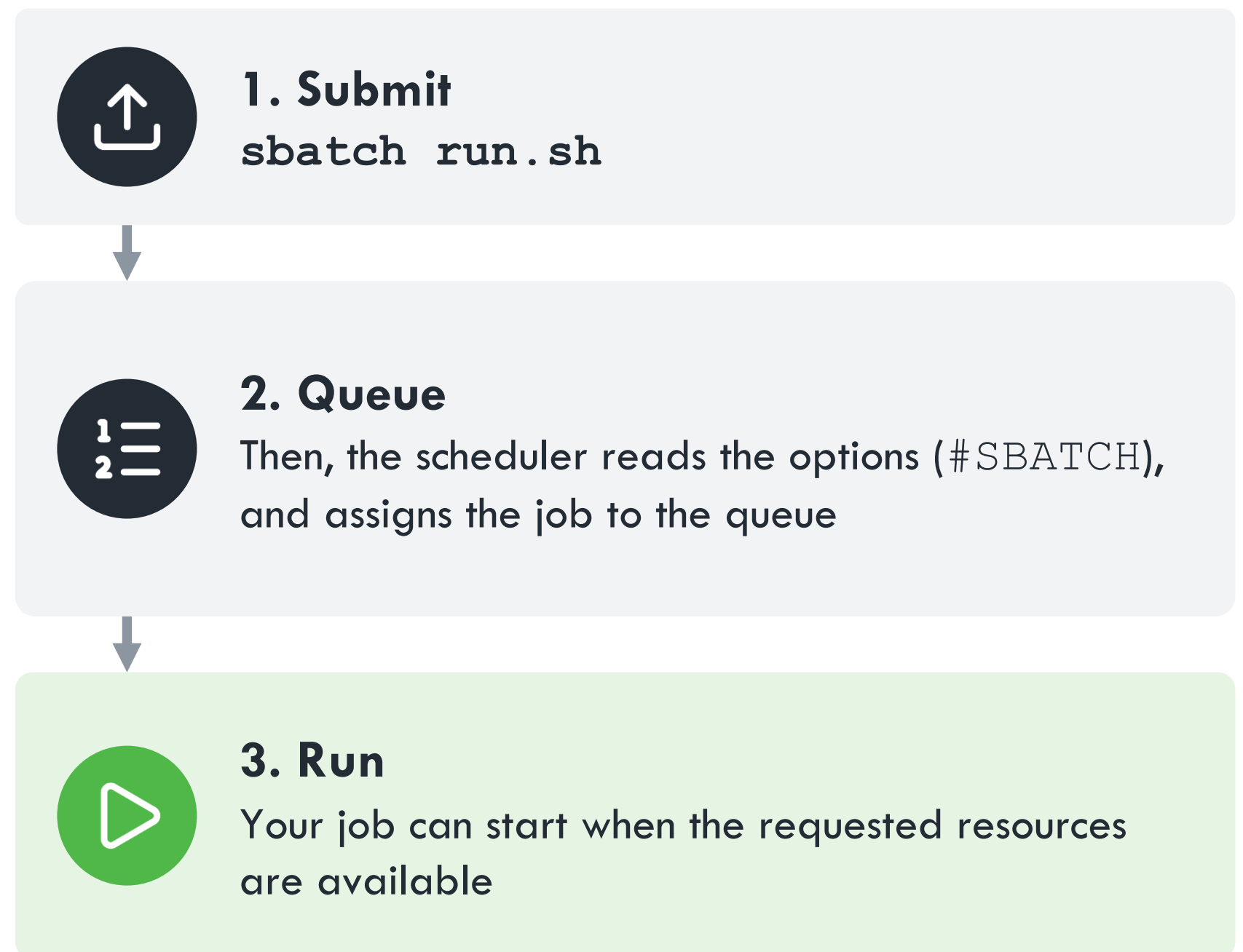
A distributed memory program (e.g. MPI) can run on multiple nodes

Submit your job

From the directory where you have your files, type:

```
$ sbatch run.sh  
Submitted batch job 123456
```

Where `run.sh` is your job file



Summary



1. Prepare

- Prepare all the program needs for the execution
- Estimate the resources you need
- Write the job script



2. Run

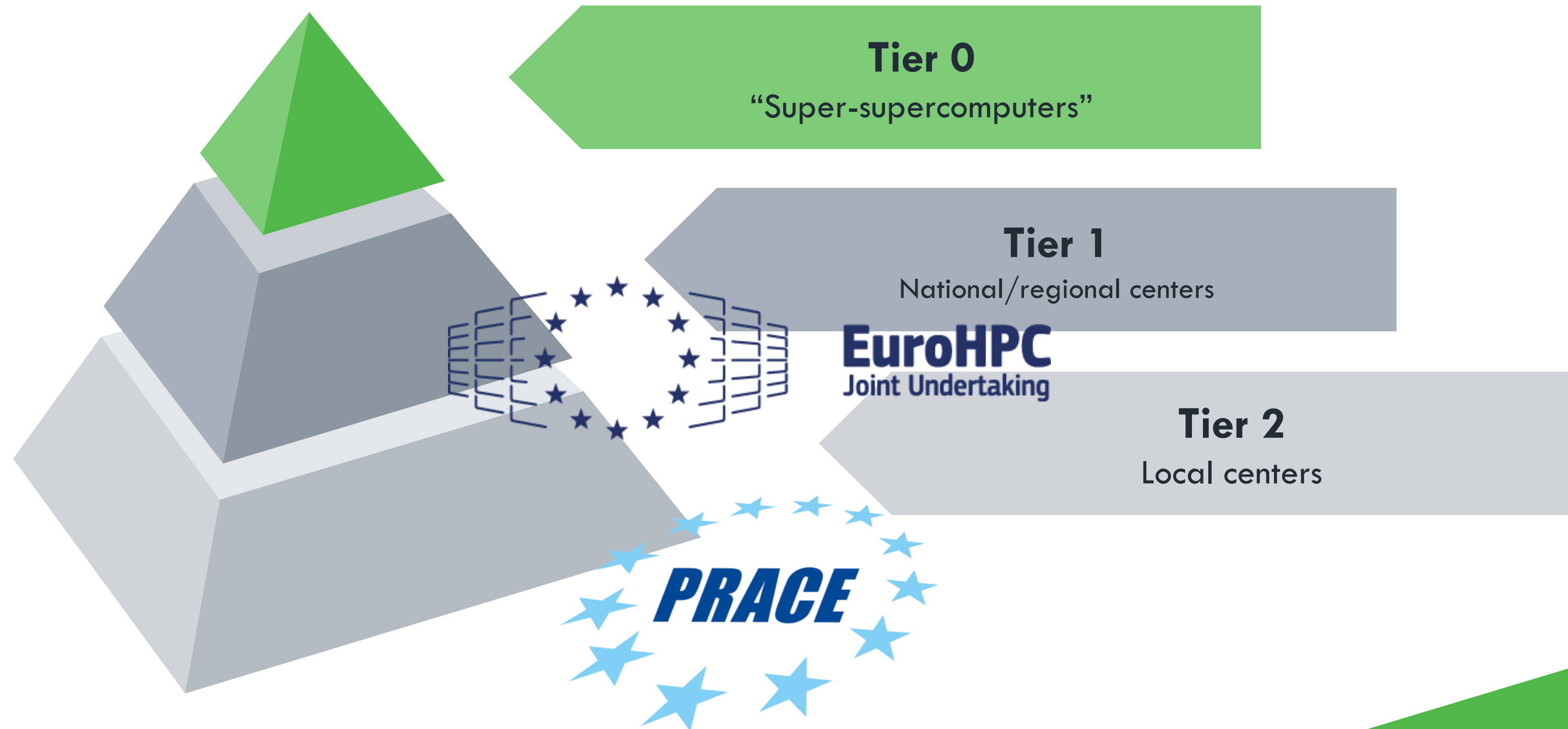
- Submit your job script
- Check your job status



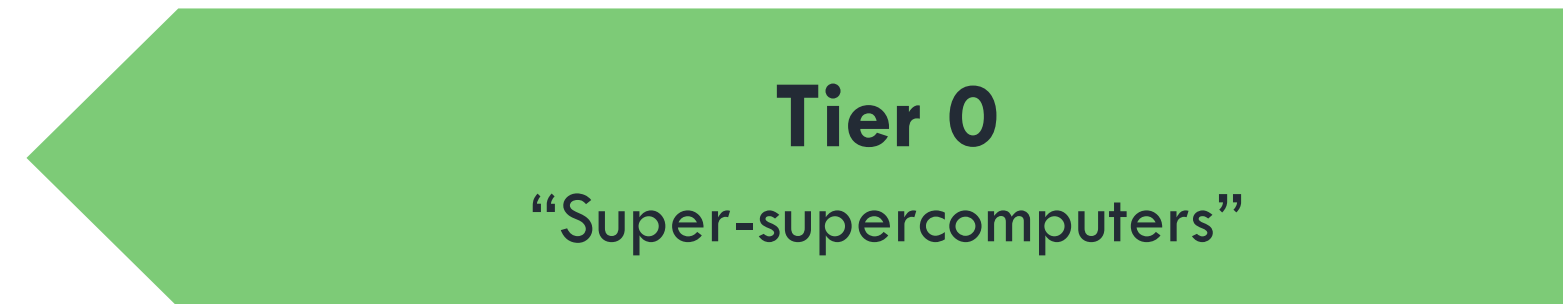
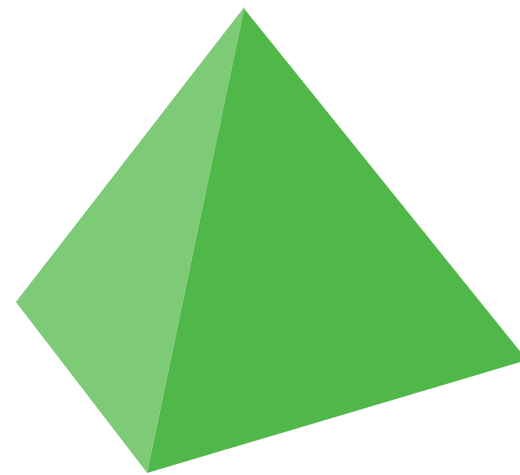
3. Wait

- Wait patiently for the results

HPC ecosystem



HPC ecosystem



EuroHPC
Joint Undertaking

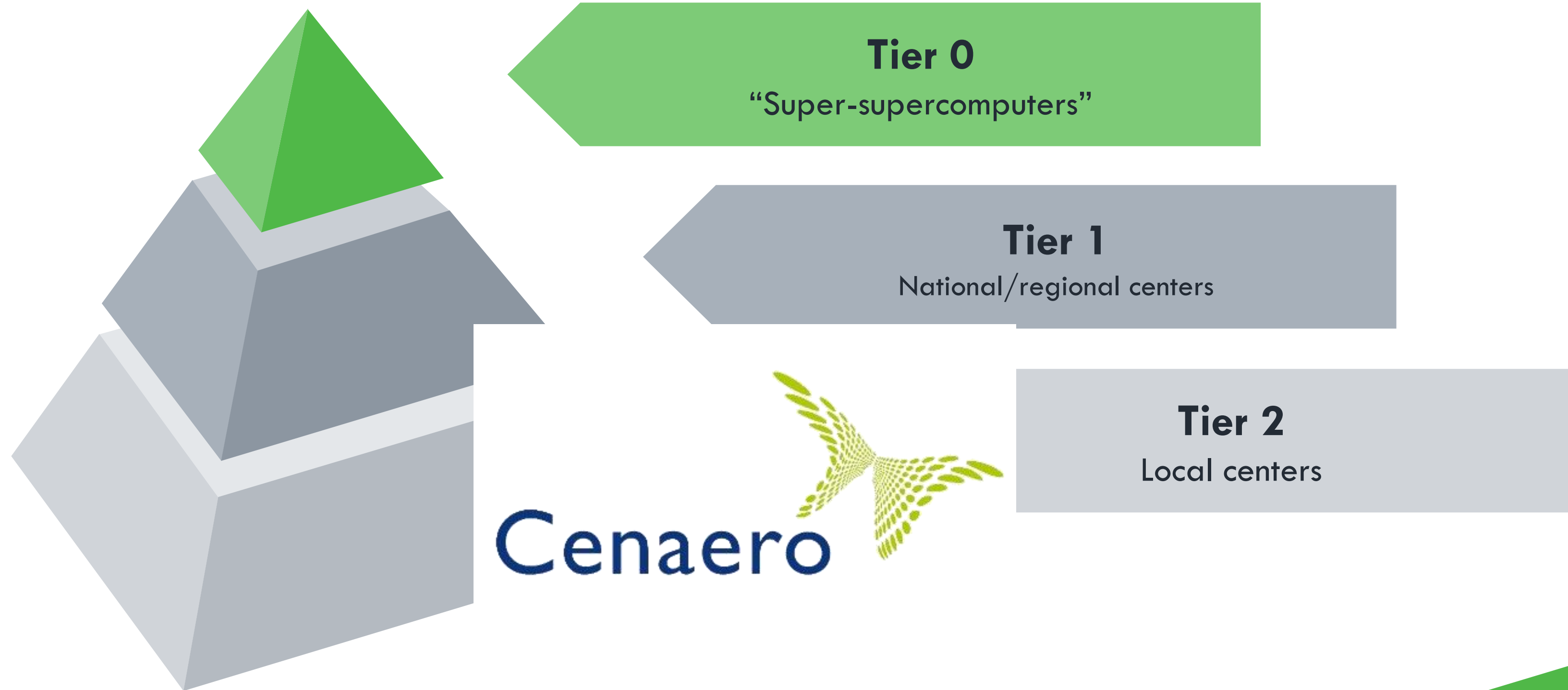




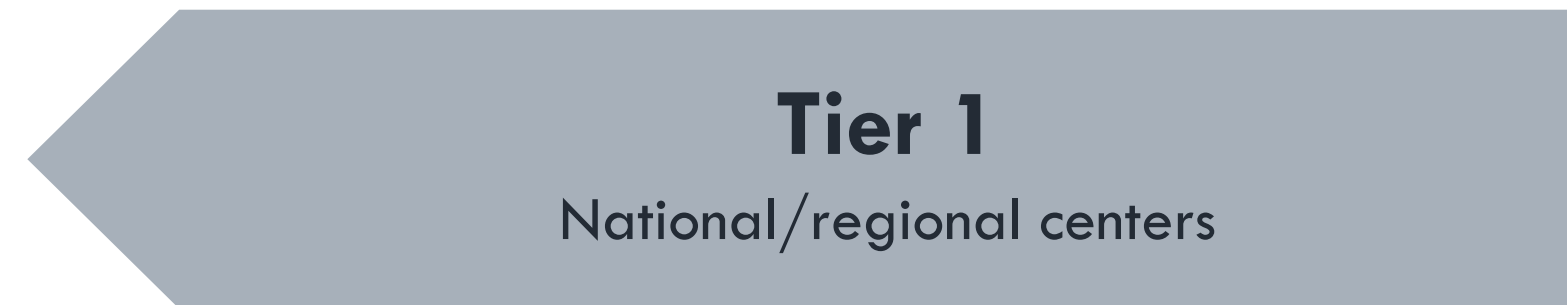
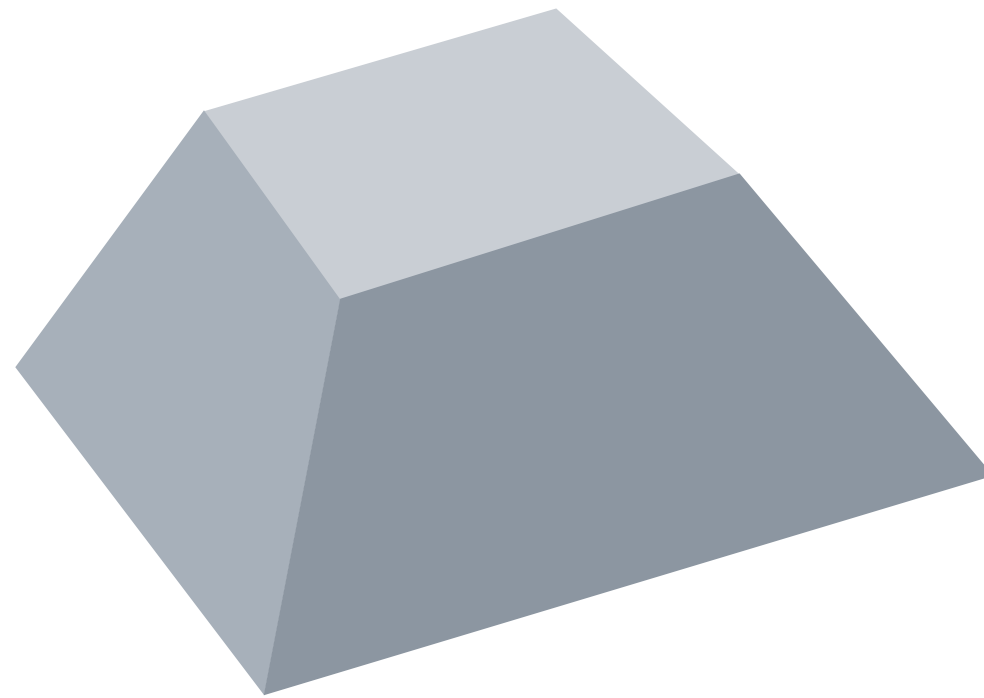
LUMI

- 360,000+ cores
- up to 4TB memory
- 10,000+ GPU
- 100+ PB storage
- #11 TOP500 (June 2026)
- 379 PetaFLOPS

HPC ecosystem in Europe



HPC ecosystem in Europe

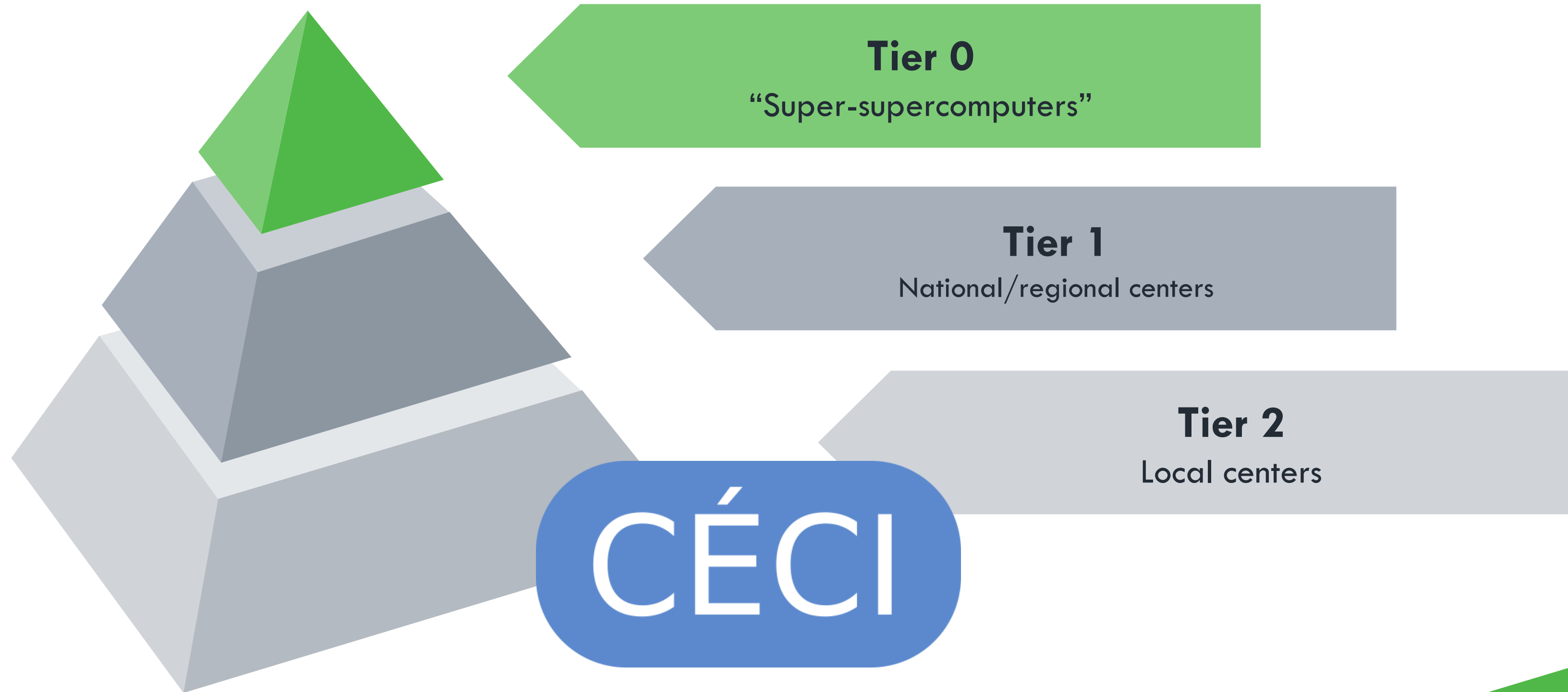




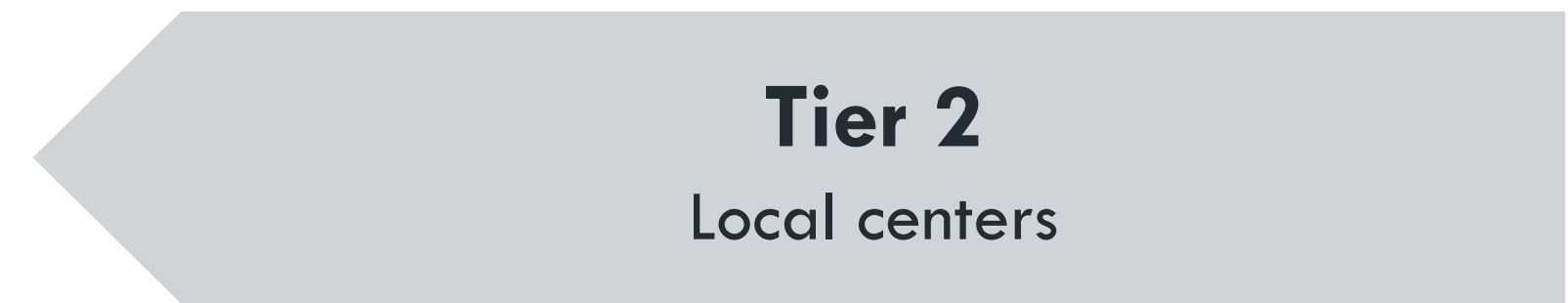
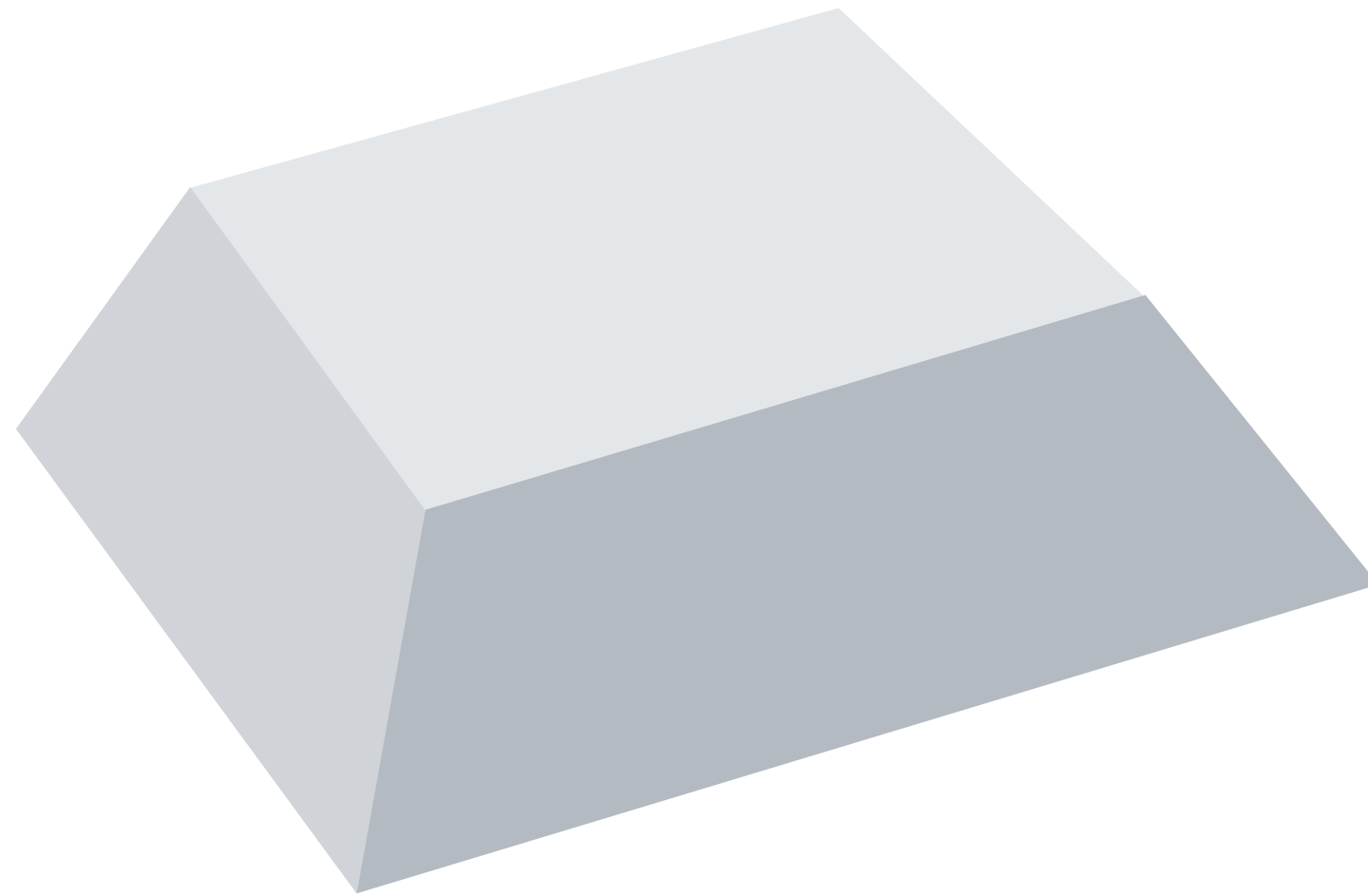
Lucia

- 38,000 cores
- up to 4 TB memory
- 200+ GPU
- 3 PB storage
- 4 PetaFLOPS

HPC ecosystem in Europe



HPC ecosystem in Europe



Consortium des Équipements de Calcul Intensif

5

universities

UCLouvain · ULB · ULiège · UMONS · UNamur

1

account

for all CÉCI clusters



shared support

helpdesk, training, documentation

UMONS
Université de Mons

ULB
UNIVERSITÉ
LIBRE
DE BRUXELLES

UCLouvain
Université

UNIVERSITÉ
DE NAMUR

LIÈGE
université

CÉCI

Five clusters, one account



NIC6

ULiège

Parallel jobs
I/O intensive



Lemaitre 4

UCLouvain

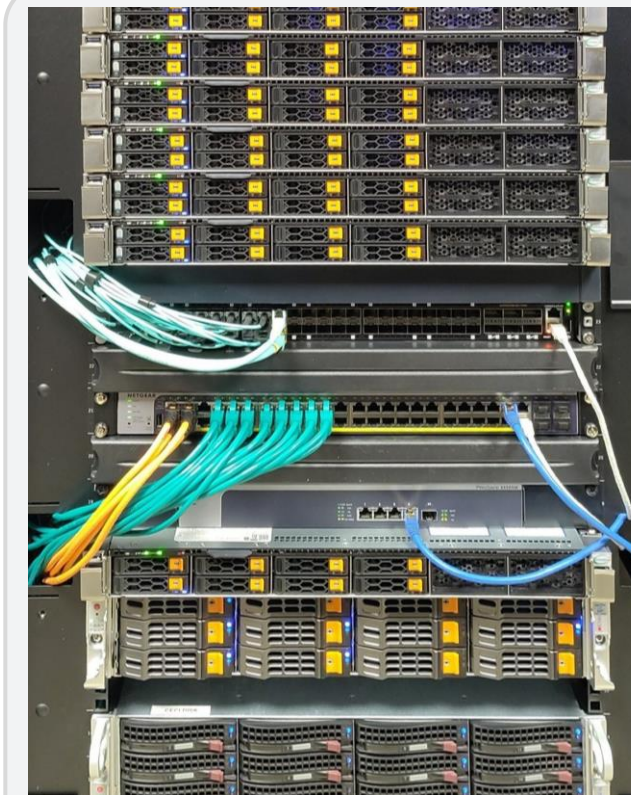
Parallel jobs
New: APU nodes



Hercules 2

UNamur

High memory
Long jobs (15 days)



Dragon 3

UMONS

Shared memory
Long jobs (21 days)



Lyra

ULB

Data-intensive
AI, GPUs

→ one login, **pick the right cluster**



Introduction to HPC

NIC6 @ LIÈGE université

- 2304 cores
- 384 GB memory max
- 800 TB storage
- Suitable for:
 - Parallel jobs
 - I/O intensive jobs
 - Short duration jobs



Lemaitre 4 @ UCLouvain Université

- 5120 cores
- 768 GB memory max
- 320 TB storage
- **NEW: 5 AMD MI300A APU nodes**
- Suitable for
 - Parallel jobs
 - I/O intensive jobs
 - Short duration jobs

Hercules 2 @ UNIVERSITÉ DE NAMUR

- 2992 cores
- 3 TB memory max
- 200 TB storage
- 16 GPU
- Suitable for:
 - Single node job only
 - High memory jobs
 - Long duration jobs (15 days)
 - Single precision GPU job



Dragon 3 @UMONS

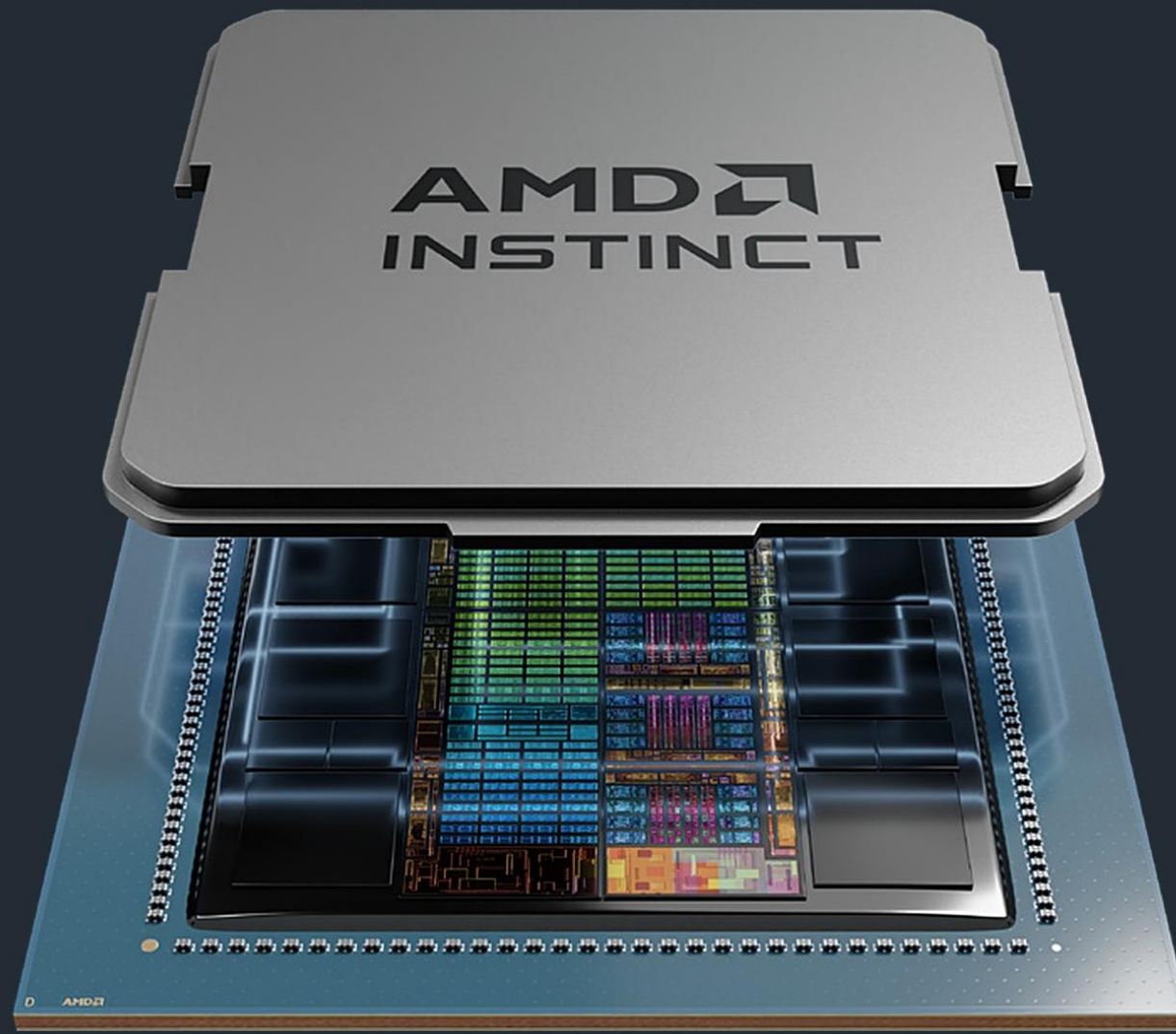
Université de Mons

- 1344 cores
- 768 GB memory max
- 98 TB storage
- Suitable for:
 - Shared-memory parallel jobs
 - Long duration jobs (21 days)
 - High memory jobs



Lyra @ UNIVERSITÉ LIBRE DE BRUXELLES

- For data-intensive jobs
 - BigData
 - Machine Learning
 - Deep Learning
 - AI
 - High Performance Data Analytics
- 40 NVIDIA GPUs



AMD Instinct MI300A

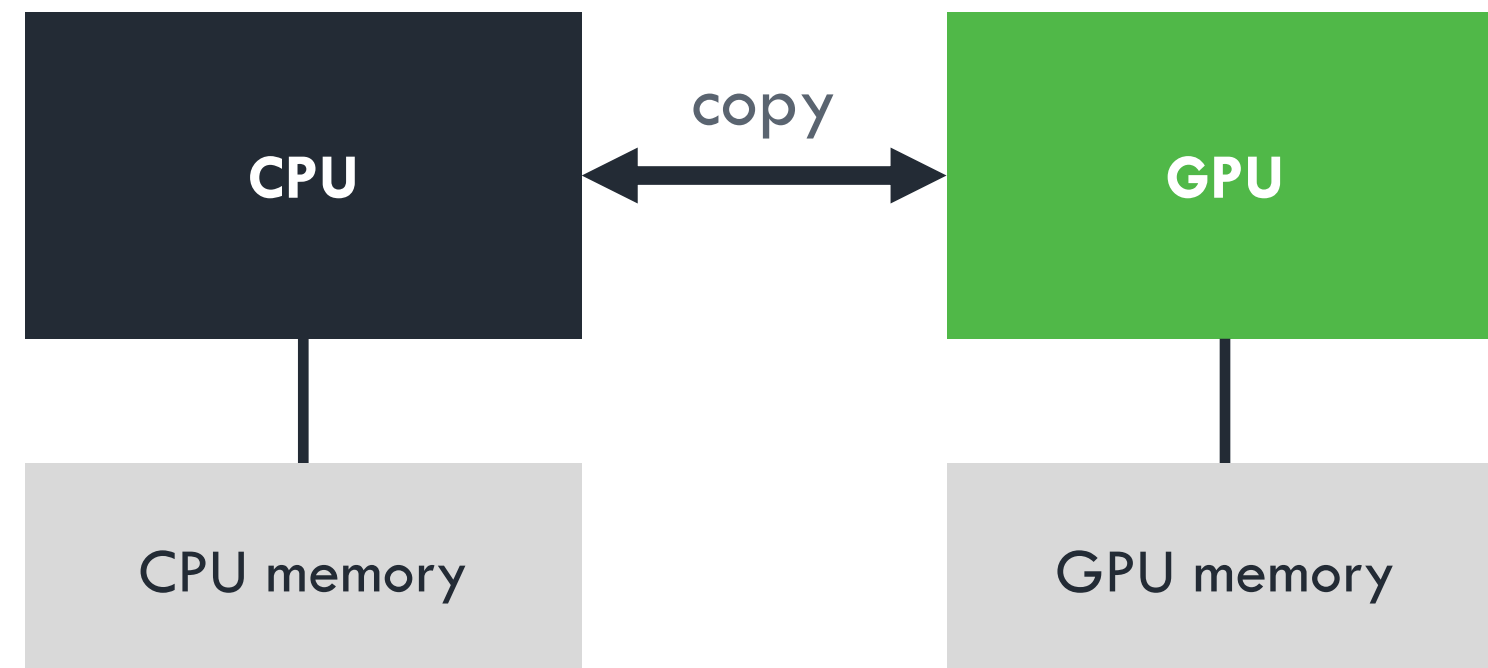
24 CPU cores + 228 GPU compute units
128 GB HBM3 shared by both

A new architecture: APU

- Accelerated Processing Unit
- Breaking the CPU–GPU barrier
 - CPU + GPU integrated in a single chip
 - Unified CPU/GPU memory
 - Optimized for double precision (FP64)

CPU + GPU vs. APU

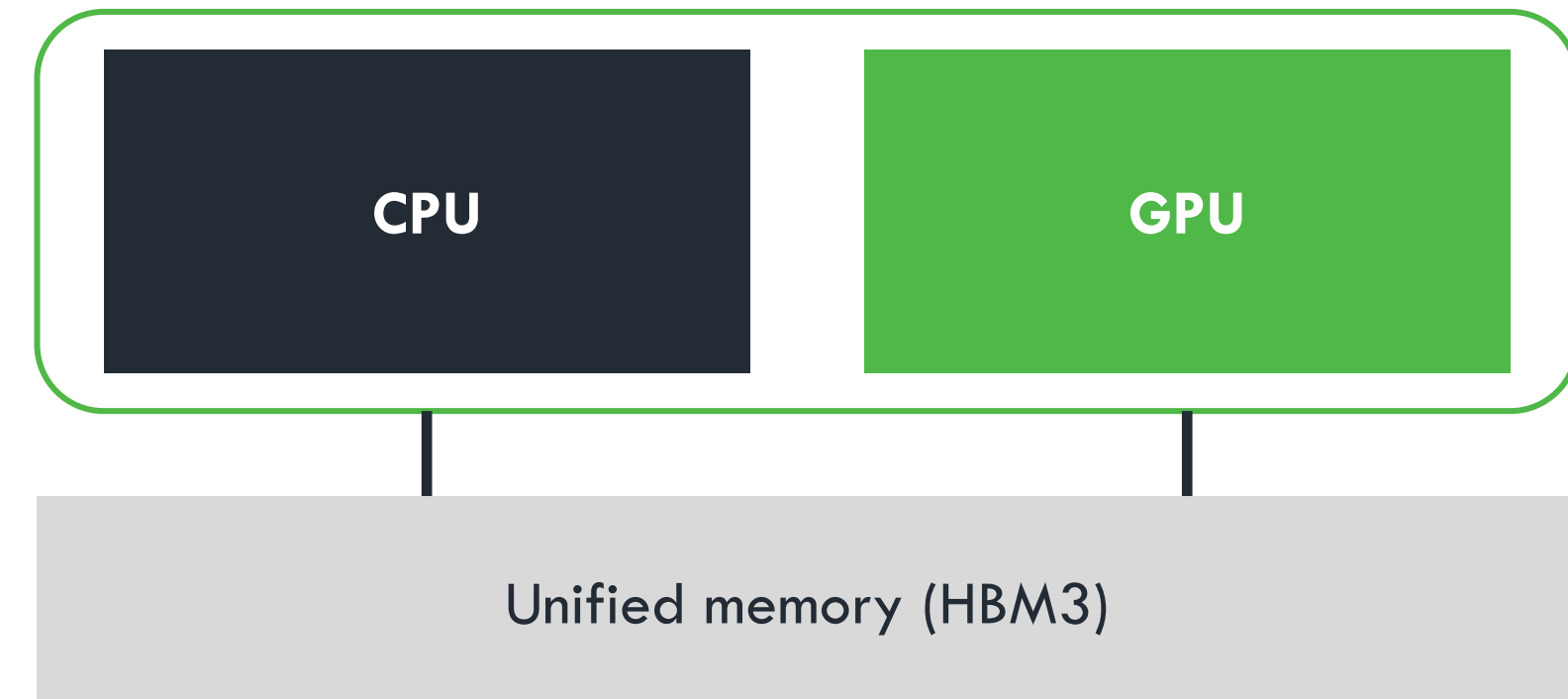
Separate CPU and GPU



Data is copied between the two memories:
extra time

vs.

APU: one chip, one memory

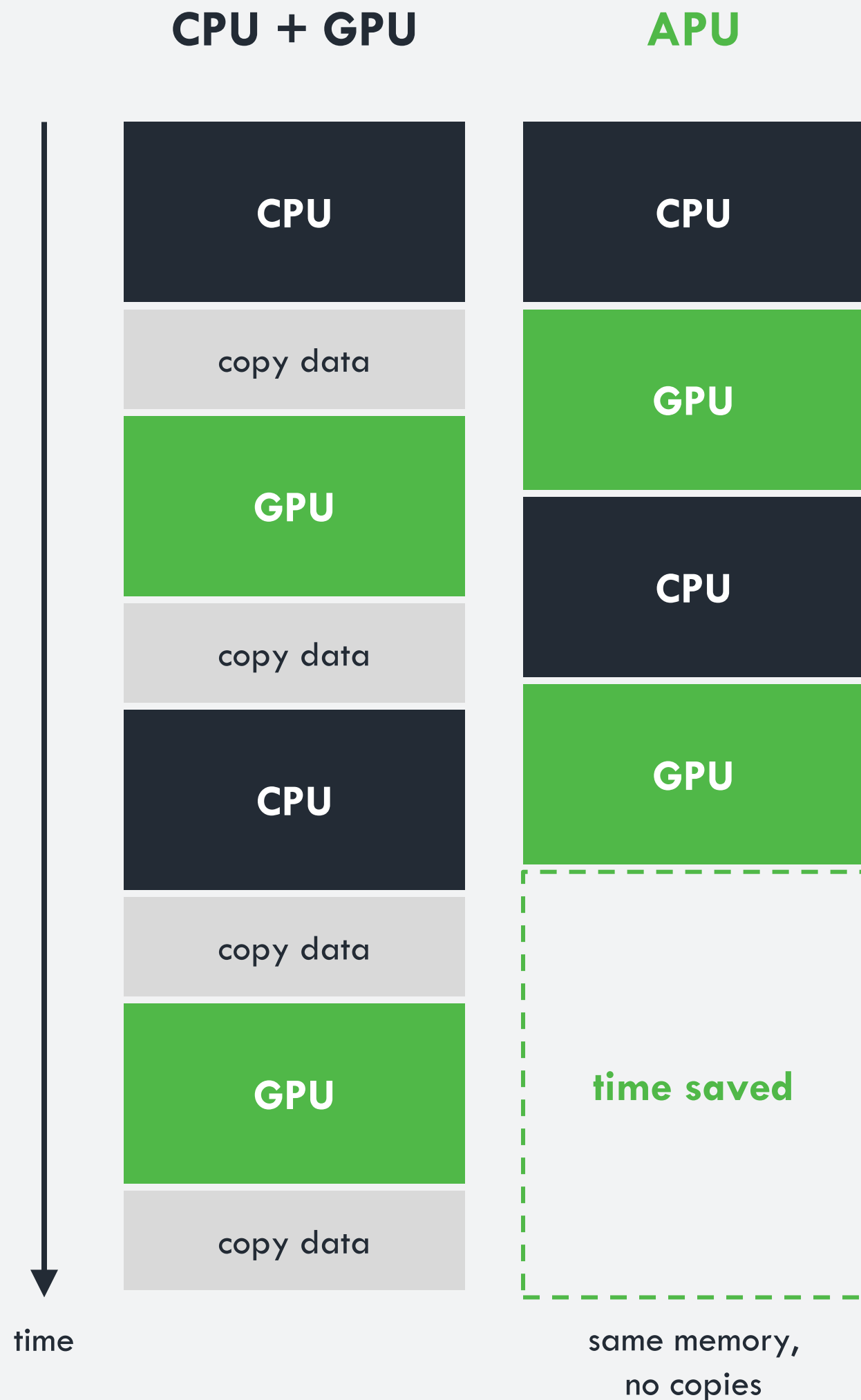


CPU and GPU work on the same data:
no copies



The New APUs

- 5 nodes with **AMD MI300A APUs**
 - 4 APU units per node
 - **128 GB HBM3 unified memory per APU**
 - CPU: 24 cores per APU
 - **30 TB \$LOCALSCRATCH**
- Up to **61 TFLOPS (FP64)** per APU



A new type of workflows

- Designed for tightly coupled CPU–GPU workloads
- Ideal for memory-intensive and large-scale simulations
- Same AMD Instinct family as LUMI (MI250X)
- **Integrated on Lemaitre 4**
- Available now

Using the APU nodes

- Partition **apu**: 5 nodes $\times$ 4 APUs
- 1 APU = 1 GPU in Slurm, with its 24 cores
- Up to 2 days per job
- Software built for the APUs: the **_apu** release

```
#SBATCH --partition=apu
```

```
#SBATCH --gres=gpu:mi300a:1
```

```
module load releases/2024a_apu
```

Storage

- HPC cluster offer several types of storage space with varying speeds and sizes.
- **Scratch Space**
 - High speed storage
 - Ephemeral
 - Can be node-attached or shared
- **Homedir**
 - Your data, shared among a cluster
- **CÉCI Common Storage**
 - Shared among all CÉCI clusters
 - Can be used to share files between users using “CÉCI projects”



CÉCI Support and Services



Basic support

- Helpdesk
- Cluster status page



Application support

- Installation
- Workflows
- Best practices



Training

- Documentation
- YouTube channel
- Scheduled trainings



Scan for our videos

→ Stuck? Ask the helpdesk: support.cec-hpc.be/cecihelp

CÉCI training

HPC Formation 2026: 13 sessions, October 2026 – February 2027. Today: Introduction to HPC and SSH



Using HPC infrastructure

15 Oct

Linux terminal, software, vim

5 Nov

Bash scripting, code versioning

12 Nov

Submit jobs with Slurm

19 Nov

Organise your work and storage



Programming on a cluster

26 Nov

Good practice, scripting languages

3 Dec

Compiling and debugging

10 Dec

Intro to Python and Julia



Going parallel

14 Jan

Parallel computing

21 Jan

Shared memory, complex Slurm jobs

28 Jan

Programming on GPU



Scripting for HPC

11 Feb

Python and Julia on HPC



Data on HPC

18 Feb

Handling data on HPC



Scan for the full program

Full program: indico.cism.ucl.ac.be/event/181

Become a CÉCI user in two steps (almost)

1

Go to <http://www.cec-hpc.be>

2

Click here

CÉCI

Clusters

News

Training

FAQ

Documentation

Support

Contact

Create/Manage Account

C

E

I

C.E.C.I

Consortium des Équipements de Calcul Intensif

6 clusters, 10k cores, 1 login, 1 home directory

About

CÉCI is the "Consortium des Équipements de Calcul Intensif", a consortium of high-performance computing centers of UCLouvain, ULB, ULiège, UMon, and UNamur. The CÉCI is supported by the F.R.S-FNRS and the Walloon Region. Read more.



Quick links

- Connecting from a Windows computer
- Connecting from a UNIX/Linux or MacOS computer
- Sturm tutorial and quick start
- Sturm Frequently Asked Questions
- Tier-1 Zenobe quickstart
- Submission Script Generation Wizard

Quick search

Photo Gallery



Save the date!

The next CÉCI scientific day will take place on Thursday April 25th in Brussels.
More information soon!

Latest News

LEMATRE3 installed at UCL

MONDAY, 04 JUNE 2018

Lemaitre3 is now operational and replaces Lemaitre2, which will be decommissioned this Summer. It has 80 nodes (SkyLake 2.3 GHz, 24CPUs, 96GB RAM) interconnected with the Intel OmniPath Architecture and more than half a petabyte of scratch space.

Dragon1 cluster featured in a Belnet article

TUESDAY, 08 MAY 2018

The Dragon1 CÉCI cluster is highlighted in an interview from Belnet to Chantal Poiriet, professor in Information and Communication Technology at the University of Mons.
Follow [this link](#) to read the complete note.

Survey on Big Data and Machine Learning needs

THURSDAY, 03 MAY 2018

We are conducting a survey about current and future High Performance Data Analysis (HPDA) works & needs, covering BigData, DeepLearning, MachineLearning, AI & co. Research groups already active in those fields are our primary center of interest. However, those moving or intending to move into those fields are welcome to fill the survey too. Our objective is to identify concrete hardware and software requirements for the future CÉCI Vega2 cluster which will be oriented towards HTC (High Throughput Computing) and HPDA.
You are therefore cordially invited to [follow this link](#) and fill the survey.

10th CÉCI Scientific Meeting

MONDAY, 26 MARCH 2018

The next CÉCI scientific day will take place on May 4th in Namur.
Details and registration [here](#).

PRACE Call for Proposal

FRIDAY, 15 MARCH 2018

PRACE has issued the 17th call for Proposals. Deadline: 2nd May 2018, 10:00 CET; Stake: Single-year and Multi-year proposals starting 2nd October 2018; Resources: Joliot-Curie, Hazel Hen, JUWELS, Marconi, MareNostrum IV, Pl2 Daint and SuperMUC.
Let us know if you apply and participate!

RSS

CÉCI

Contact

73

Key takeaways

1



Many cores working together:
bigger problems, or faster results

2



Parallel work has a cost: keep the
overhead small

3



OpenMP on one node, MPI
(message passing) across many
nodes

4



Login node: for preparing your
work, never for computing

5



Job script: run your work through the
scheduler, ask for the right resources

6



CÉCI: five clusters, one account, new
AMD MI300A APU nodes



Thank you!



Create your account: www.cec-hpc.be



Ask the helpdesk: support.cec-hpc.be/cecihelp



Join the next trainings

