GOBIERNO DE ESPAÑA · MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES · Instituto de Salud Carlos III

Introduction to High Performance Computing and its elements

Identify which problems can be solved using a HPC.

BU-ISCIII
29-04 de octubre de 2025
1ª edición

Outline

1. Round of introductions
2. Course overview and objectives
3. Why High-Performance Computing?
4. HPC Architecture and Elements
5. HPC Security, Policies, Do’s and Don’ts
6. Parallelism Concepts intro
7. Scientific Applications of HPC
8. Q&A

Round of introductions

1. Who are you and what's your background?
2. What is your experience (if any) with HPC or Linux?
3. What do you expect to learn or achieve in this course?

[redacted]

Course overview and objectives

Introduction

Identify market needs and initial concepts for a new product.

Access

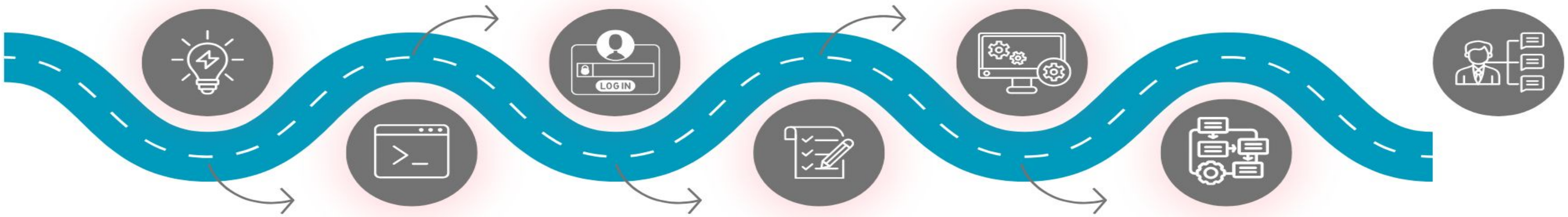
Analyze trends and customer needs to validate the product idea.

Software and customization

Iterate based on feedback from product testing results.

Use cases and troubleshooting

The product is launched to the market with an integrated marketing strategy.



Linux

Develop the main design and features of the product before prototyping.

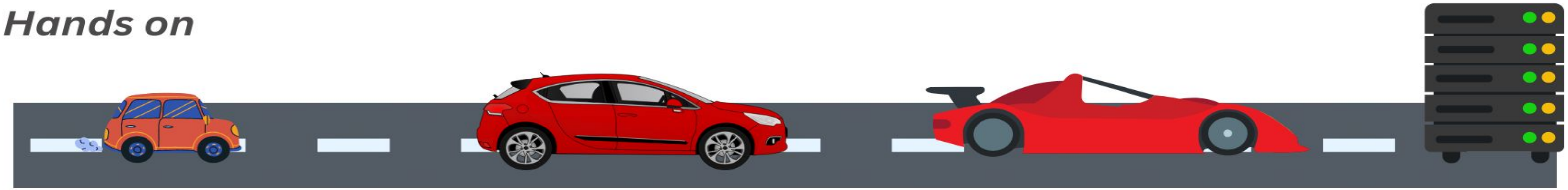
Job scheduling

Test the initial product to identify issues and improvements.

Parallelization and Workflows

The product is launched to the market with an integrated marketing strategy.

Hands on



Course overview and objectives

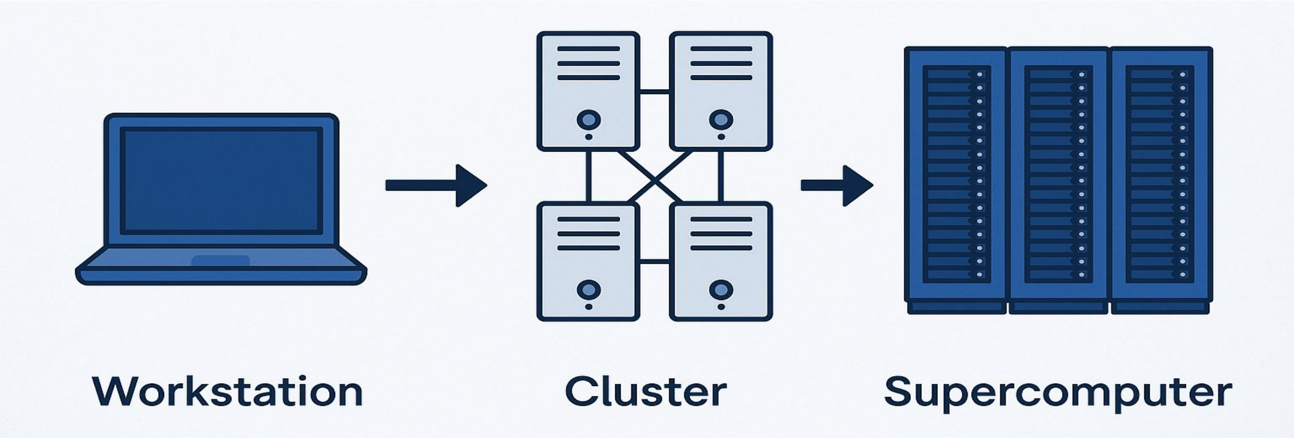
- Introduce the fundamentals of High-Performance Computing (HPC)
- Gain Linux shell skills for interacting with HPC systems
- Learn how to access, transfer, and manage data on an HPC
- Understand how to submit and manage jobs with a scheduler
- Explore software installation and containers
- Write and optimize HPC scripts and workflows
- Address security, policies, and best practices for cluster usage
- Apply knowledge to case studies and troubleshooting common problems

Objective: build the skills to run reliable, reproducible scientific workflows on HPC infrastructure.

Why High-Performance Computing? from workstations to supercomputers

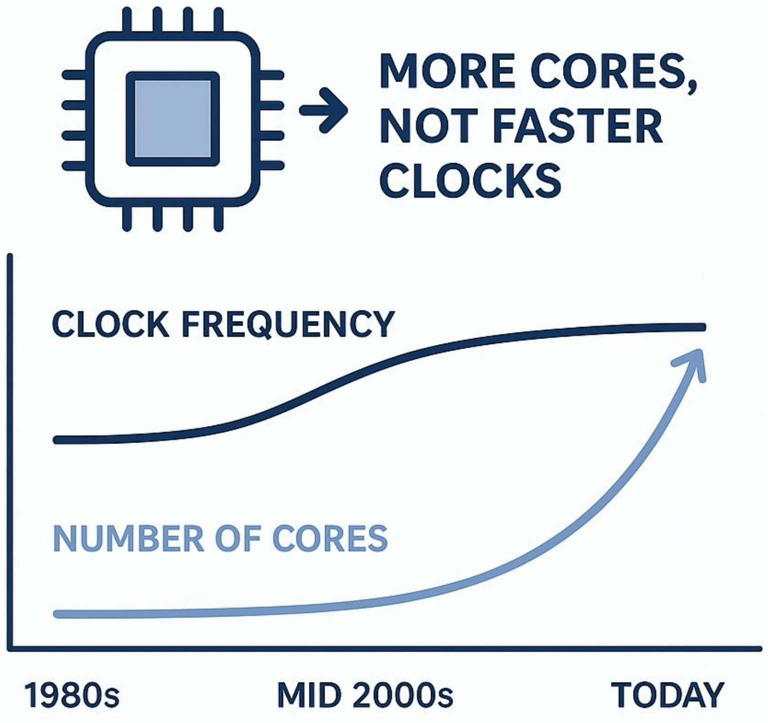
- **Workstations**: limited CPU, memory, storage → good for small tasks
- **Clusters**: interconnected nodes, shared storage, parallel computing
- **Supercomputers**: thousands of nodes, extreme scale, petaflop/exaflop performance

HPC enables solving problems beyond the capacity of a single machine



Why High-Performance Computing? Parallelism becomes essential

1. **Moore's Law:** transistor density doubled every ~2 years
2. **Clock frequency plateaued** (≈ mid 2000s → power & heat limits)
3. **Performance growth shifted to parallelism** (multi-core, many-core)
4. HPC leverages massive parallelism: CPUs + GPUs + accelerators



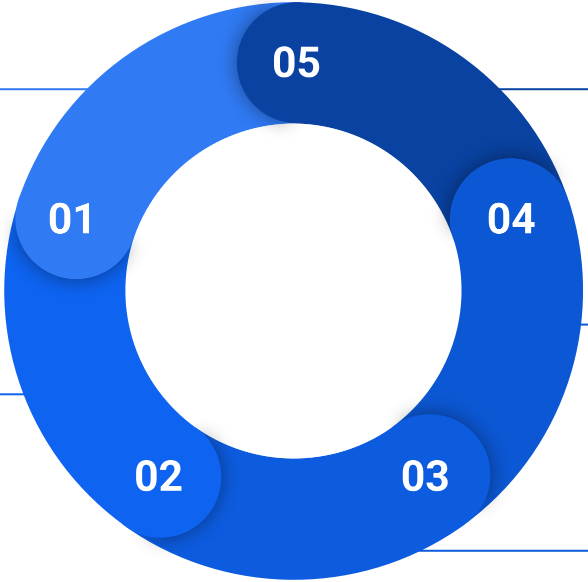
A typical workflow on HPC Systems

Login to the cluster

SSH

Prepare data and environment

Winscp, filezilla, rsync.
Software and customization



Submit job to scheduler

SLURM, PBS; etc..

Monitor job execution

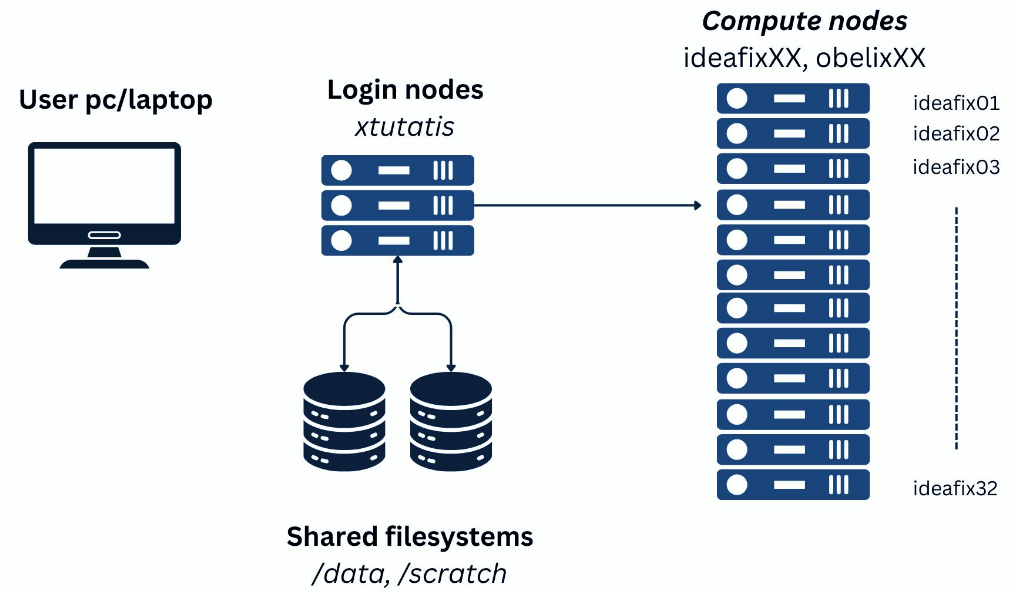
Squeue, sacct, scontrol

Retrieve and analyze results

Copy data back, clean temp files and reset environment.
Tertiary analysis with scripting (R, python, etc)

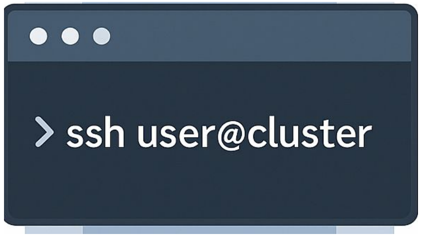
HPC Architecture and Elements

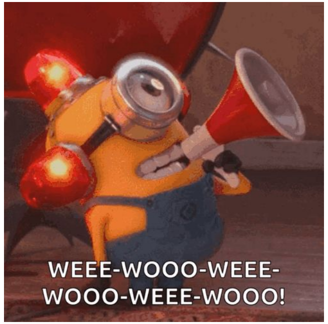
- HPC is composed of **specialized components** working together
- **Separation of roles:** access, computation, storage, communication, scheduling
- **Optimized** for scale, speed, and fair resource usage

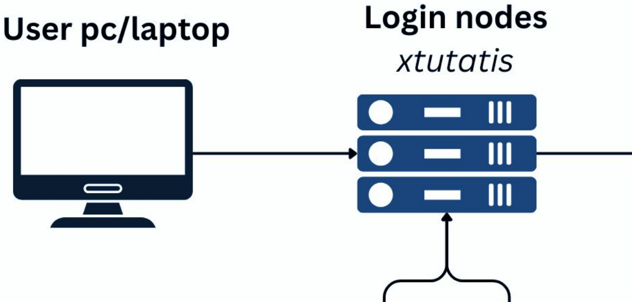


Access and login nodes

- Entry point to the cluster
- Linux-based, no graphical interface
- ⚠️ Lightweight tasks only: editing, compiling, submitting jobs
- ⚠️ Not for heavy computation





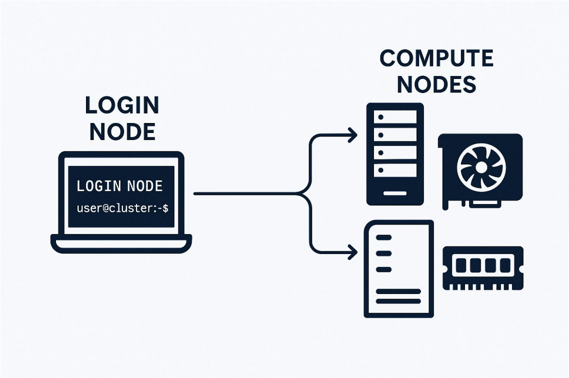


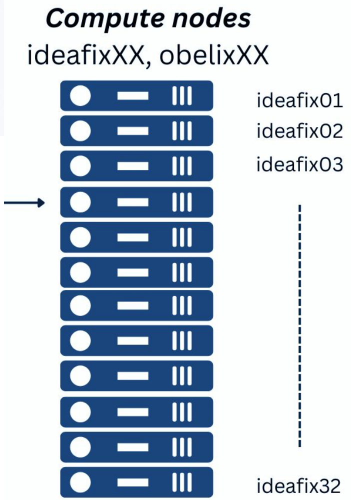
Compute Nodes

Where heavy computation happens

Types:

- Standard (general-purpose CPUs)
- GPU nodes (accelerated parallel tasks, AI, simulations)
- High-memory nodes - fat nodes (large datasets, genomics, ML)

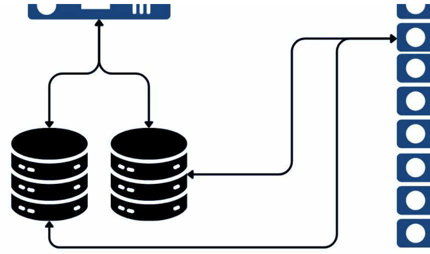




Storage and archival systems

- **Storage and archival systems**
  - Storage arrays provide large-scale, fast storage
  - Archival systems for long-term backup and compliance
  - Data management policies (quotas, retention)
  - This storage are divided and configured in several filesystems.

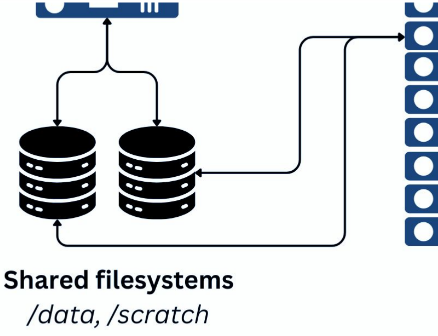
👉 *This is the **physical infrastructure** that provides raw storage capacity'*



Shared filesystems
/data, /scratch

Filesystems

– **Home**: small, permanent, for personal configs/scripts
– **Data**: shared among groups, medium/long-term storage
– **Scratch**: large, temporary, high-performance workspace
– **Archive**: large, long-term storage
– Each optimized for different use cases depending on the HPC

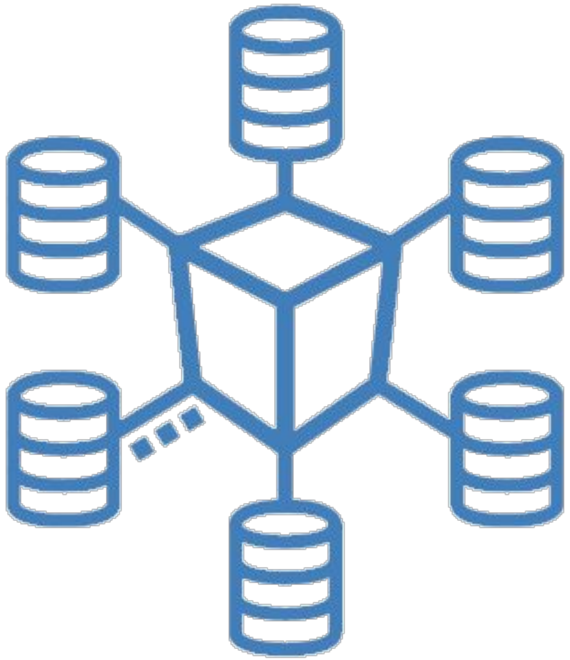


👉 *Filesystems is the **logical view** that divide and configure the storage hardware into areas optimized for different use cases.*

> *These filesystems are transparently mounted on login and compute nodes through network protocols (NFS, sometimes SAMBA). Performance depends on I/O bandwidth and concurrency.*

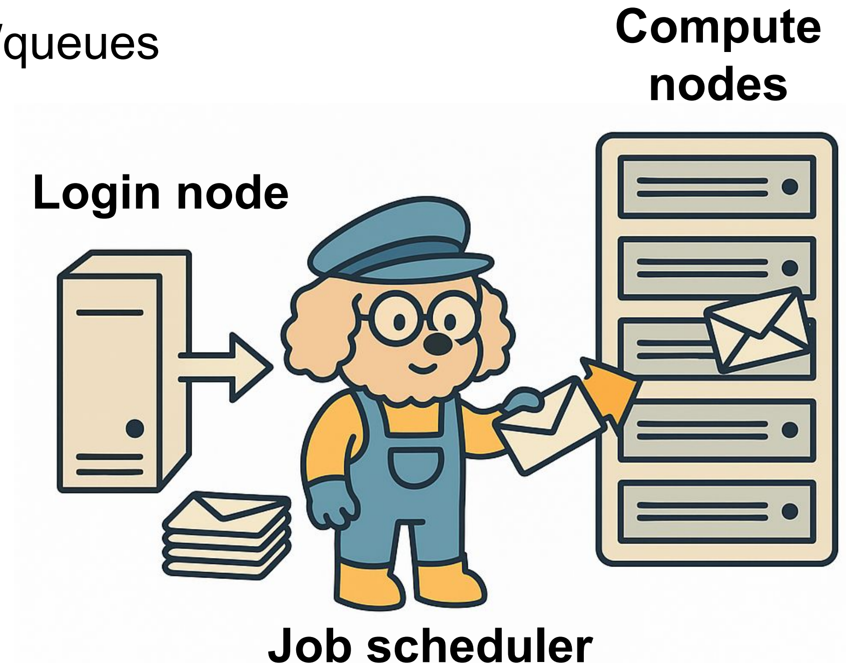
High-speed networks

- Low latency and high bandwidth are essential
- Typical technologies: InfiniBand, Omni-Path, high-speed Ethernet
- Enables parallel applications (MPI) to scale across nodes



Job Scheduling (SLURM)

- Allocates resources fairly among users
- Jobs submitted → placed in partitions/queues
- Priorities based on policies and usage
- Examples: SLURM, PBS, LSF

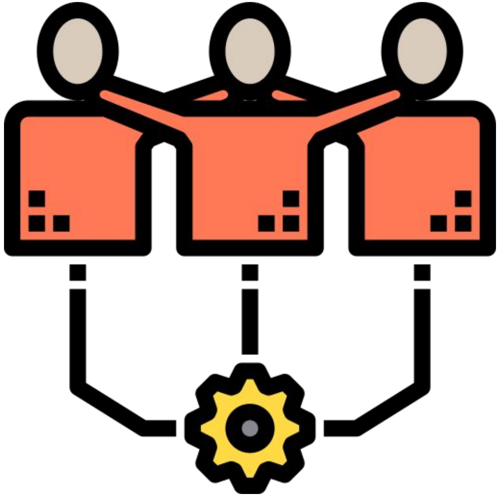


Quiz question

*Where does your job actually runs: **login node** or **compute** node? And so, Which environment is the job using (software, env variables, etc..)?*

HPC security and data policies, Do’s and Don’ts

- HPC is a shared resource → rules keep it efficient and safe
- Security and fair use are everyone’s responsibility
- Follow policies for data, storage, and job submission



Data policies

✅ Handle sensitive data with care (GDPR compliance)

✅ Use approved storage for research data

✅ Regularly backup important files

❌ Don’t store personal/confidential data in scratch areas

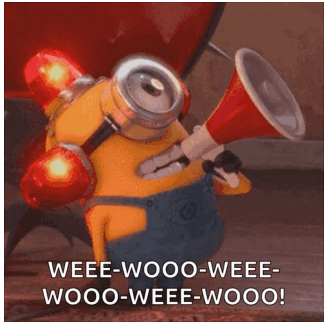
Proper use of storage areas

✅ Home → configs, small scripts, limited space

✅ Data→ shared, group-level, medium-long term

✅ Scratch → large, temporary, high-performance; cleaned

❌ **Don’t use** scratch as permanent storage



Instituto de Salud Carlos III

>_BU-ISCIII

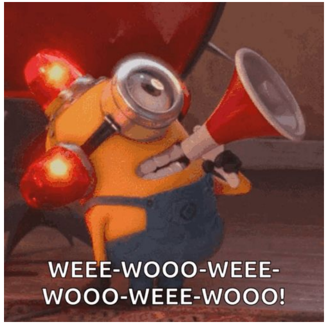
Responsible Job Submission

✅ Submit jobs through the scheduler (SLURM)

✅ Cancel misconfigured jobs promptly

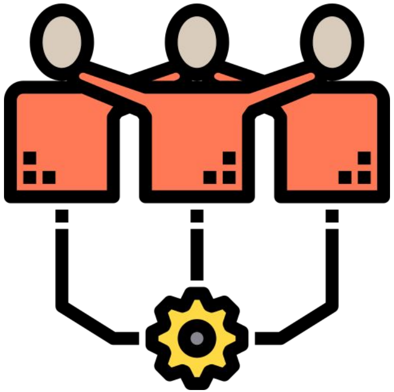
❌ Never run heavy jobs on the login node

✅ Request only the resources you need



Fair use and reporting

- Respect assigned quotas (storage, CPU, GPU time)
- Communicate problems responsibly (support tickets, admins)
- Shared resource → your behavior impacts others



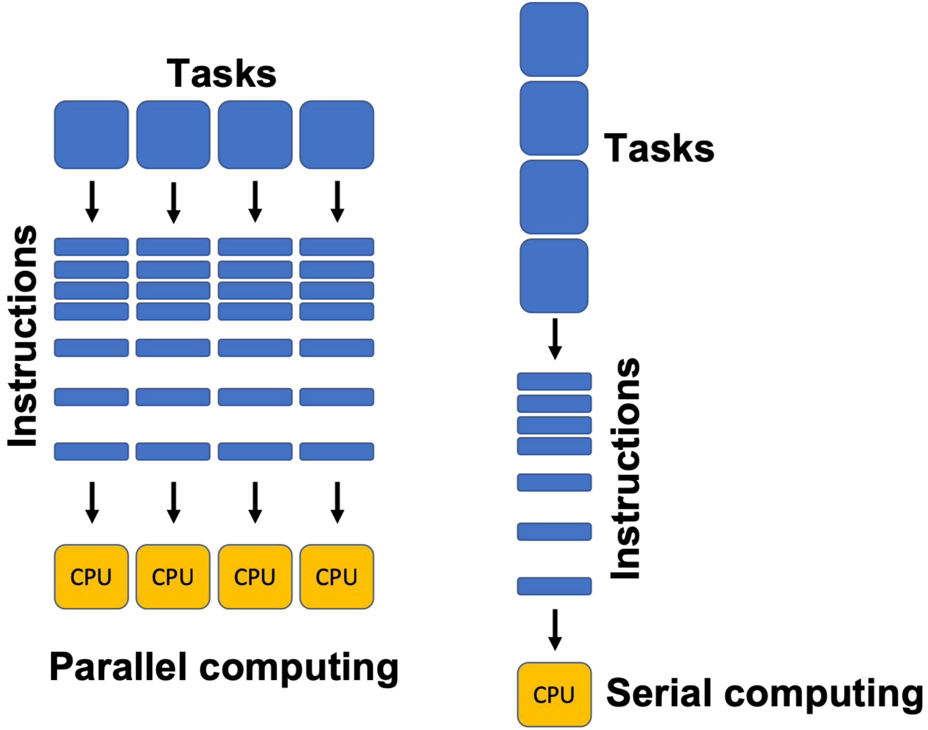
“Think before you compute: Secure, Fair, Responsible”

Quiz/Trick question

Would you store sensitive patient data in scratch, home, or project?

Why Parallelism?

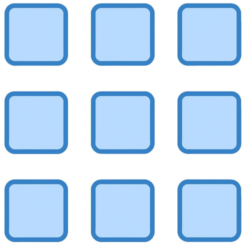
- Problems too large for a single CPU → need many processors
- Data sizes grow faster than hardware speed (Moore's Law plateau).
- Faster results = faster science and decision making.



Parallelism: two flavors

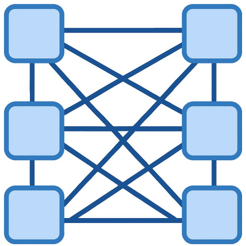
Embarrassingly parallel

- Tasks independent of each other.
- Example: running genome assembly.



Tightly coupled

- Tasks communicate constantly.
- Example: climate models, molecular dynamics.



What kind of parallel problem is this?

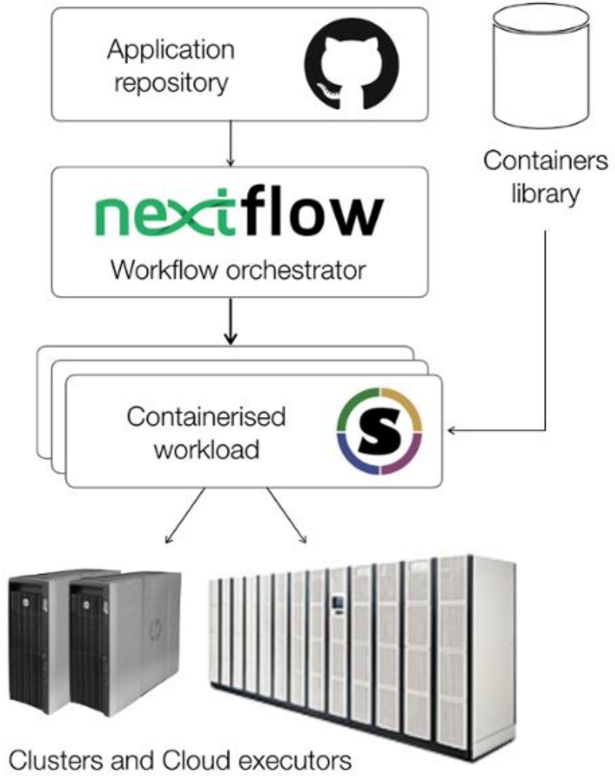
Imagine you need to analyze 1,000 sequencing samples. Would you treat this as an ***embarrassingly parallel problem*** *or a* ***tightly coupled problem****? Why?*

Scientific applications of HPC

- **Climate and weather forecasting** → long-term simulations, urgent forecasts
- **Physics and engineering** → fluid dynamics, astrophysics, materials science
- **Genomics and bioinformatics** → genome sequencing, protein modeling, phylogenetics
- **Artificial Intelligence** → deep learning, large language models

HPC in Genomics and Bioinformatics

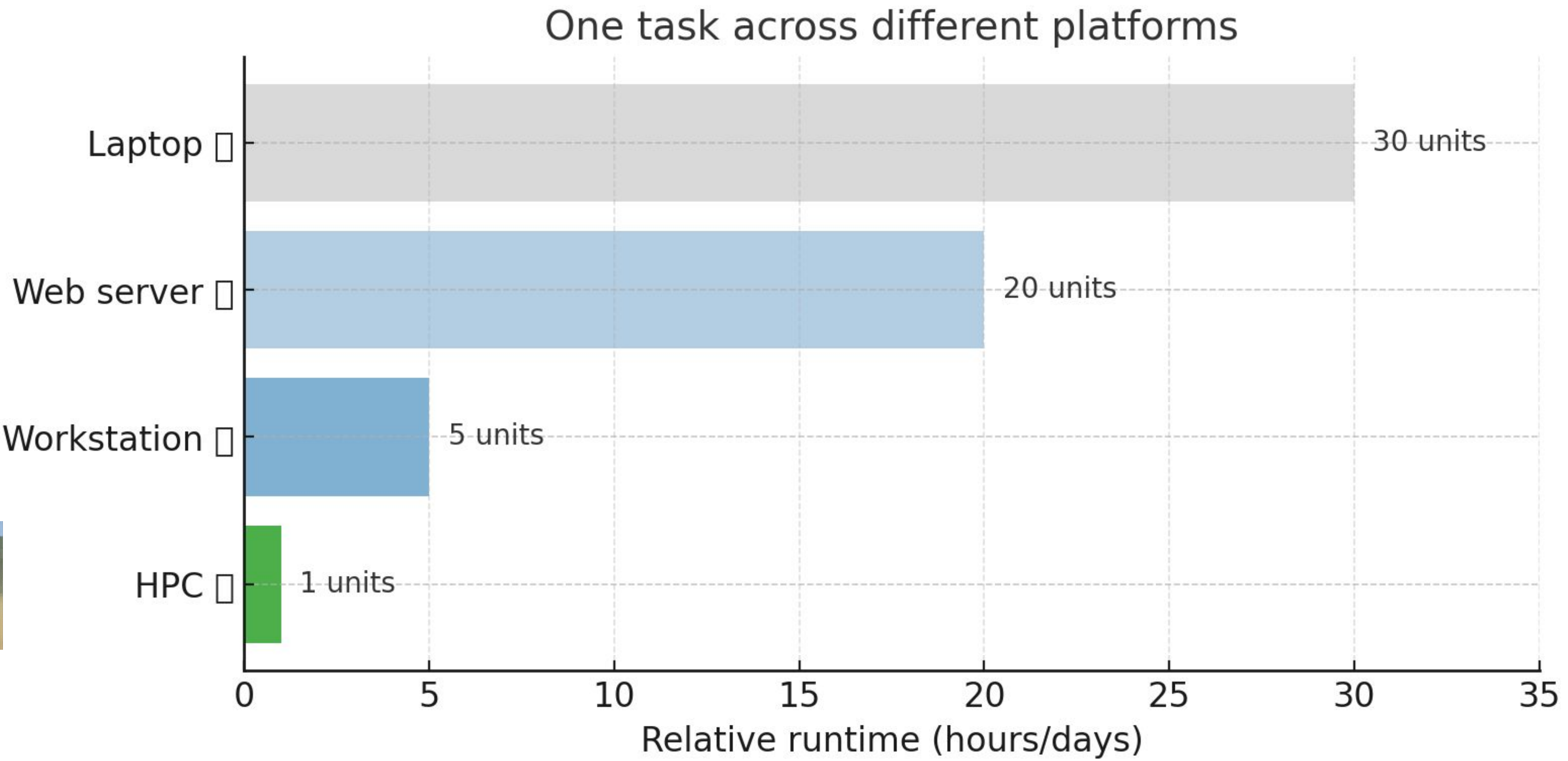
- Genome assembly from massive sequencing data
- Variant calling & population studies
- Phylogenetic analysis & molecular epidemiology
- Protein structure prediction & drug design



Discussion point

*What HPC applications do you know in your field of research? Do they fit better with **independent analyses** (easily parallelizable) or with **strongly coupled problems**?*

Why we need HPC: From weeks to hours



Why we need HPC: From weeks to hours

Platform	Genomics: 10 WGS (30×) (GATK / nf-core sarek)	Microscopy: 1000 images 2048×2048 (Segmentation CellPose/Ilastik)
Laptop (4 CPU, 16 GB RAM, no GPU)	Not feasible (> **1 - 2 months**; RAM limits & instability)	**3 - 5 days** (CPU-only)
Shared web server (8–16 CPU, 32–64 GB, no GPU, queued)	**3 - 4 weeks** (queues + limited parallelism)	**1 - 2 days** (queues; CPU-bound)
Workstation (32 CPU, 128 GB, 1× mid-range GPU)	**4 - 6 days** (batching; decent I/O)	**8 - 12 hours** with GPU 20–30 h CPU-only
HPC cluster (200–500+ CPU total, TB RAM, SLURM; GPU nodes available)	**12 - 24** hours (parallelize per sample; high I/O)	**1 - 3 hours** multi-GPU 4–8 h CPU across nodes

Key Takeaways

- HPC is a **shared resource** → fair and responsible use
- **Typical workflow:** login → prepare data → submit jobs → retrieve results
- Architecture elements: **login node** (xportuatis), **compute nodes** (ideafix), **storage**, **scheduler**
- **Parallelism is essential**: independent (embarrassingly parallel) vs. tightly coupled problems
- **HPC powers science**: genomics, climate, physics, AI

Next steps in the course

- 🐧 Linux Shell → basic skills to work on HPC (next lecture)
- 🎛️ SLURM → resource reservation & job submission
- 📦 Software installation & containers → reproducible environments
- 🔀 Scripting & parallelization (OpenMP/MPI) → scaling your workflows
- ⚙️ Workflow managers (Nextflow) → reproducibility & automation
- 🧩 Troubleshooting & case studies → solving real HPC problems

Thank you for your attention

Questions?