



Recommendations from Support

NUF26

Torben Rasmussen

NAISS

Short about me



Application Expert since 2008



Background in computational chemistry



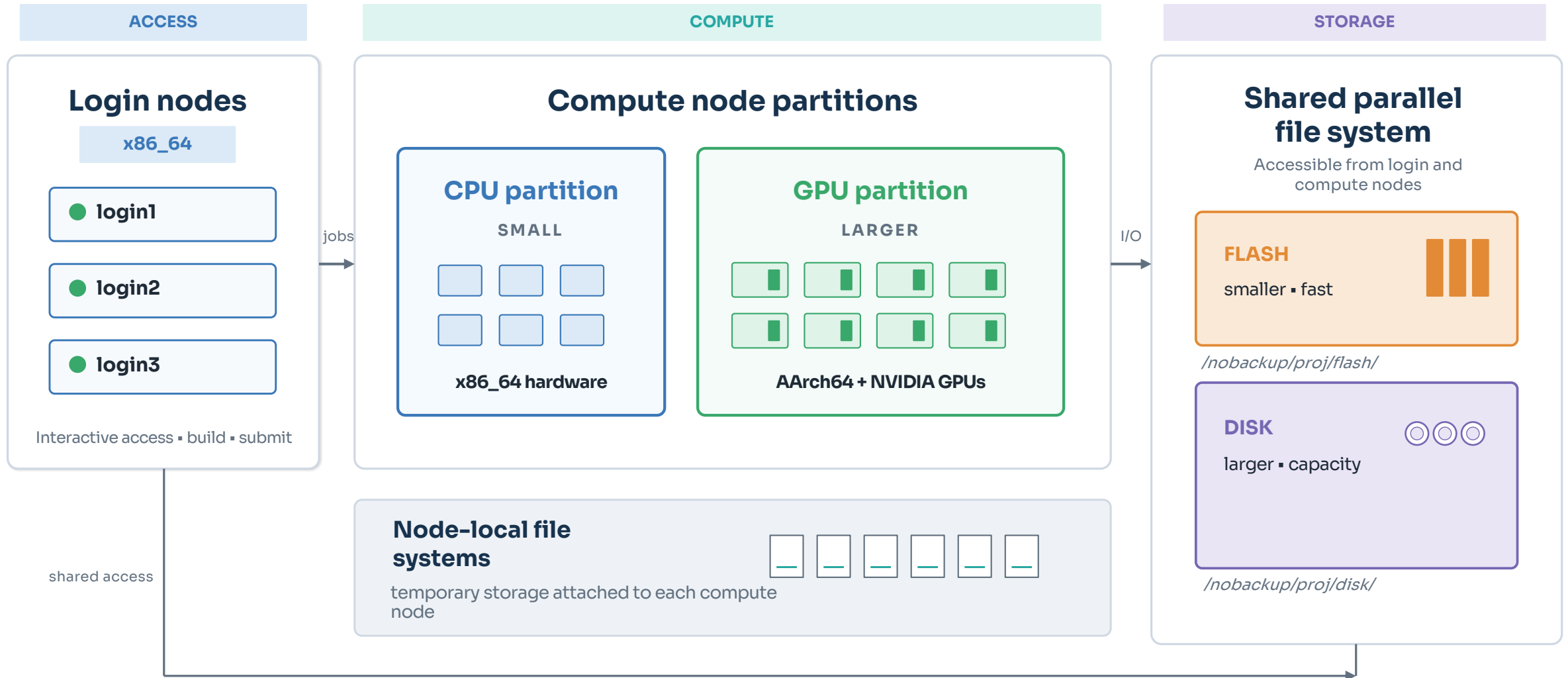
Focus on HPC software environment

The HPC system design



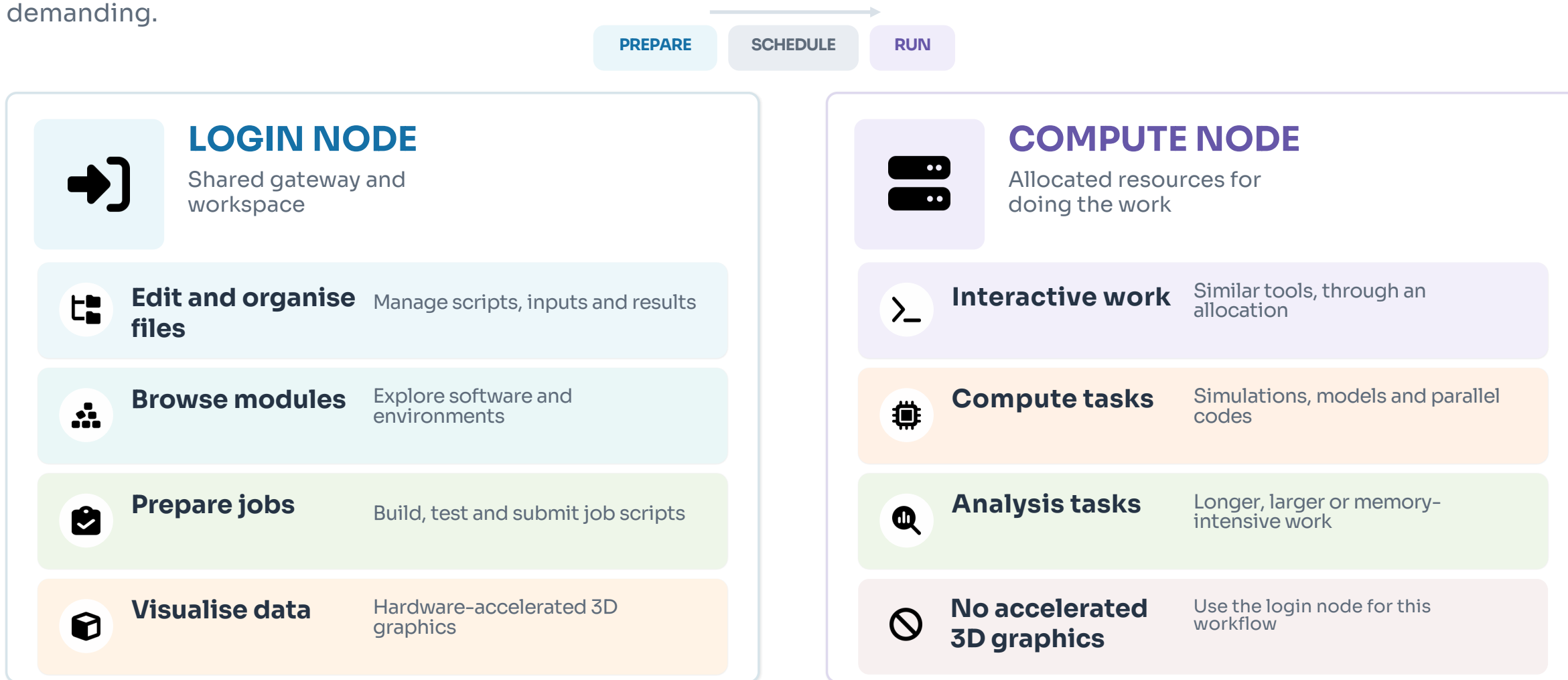
Arrhenius HPC system overview

Enter through x86_64 login nodes, run jobs in two hardware partitions, and access both shared and node-local storage.



Login node or compute node?

Use the shared login environment to prepare work. Move to scheduled compute resources when the work becomes demanding.



Choosing the right file system

Match storage to the I/O pattern, sharing need and data lifetime.



PARALLEL FLASH

Shared ▪ high IOPS

Use for I/O-intensive data

BEST FOR

- Datasets and databases
- Random reads and writes
- Workloads where storage limits runtime

WATCH OUT

Quota for flash is requested separately and requires good motivation.



PARALLEL DISK

Shared ▪ high capacity

Use for normal project storage

BEST FOR

- Sequential reads and writes
- Inputs and outputs shared across jobs
- Large data that is not I/O-critical
- Data kept between allocations or runs

WATCH OUT

Move active datasets to flash when disk I/O becomes a measured bottleneck.



NODE-LOCAL ~1 TB

Private to one node ▪ temporary

Use for per-job scratch and staging

BEST FOR

- Many small or temporary files
- Repeated reads by one node
- Random I/O and intermediate data
- Reducing load on shared storage

WATCH OUT

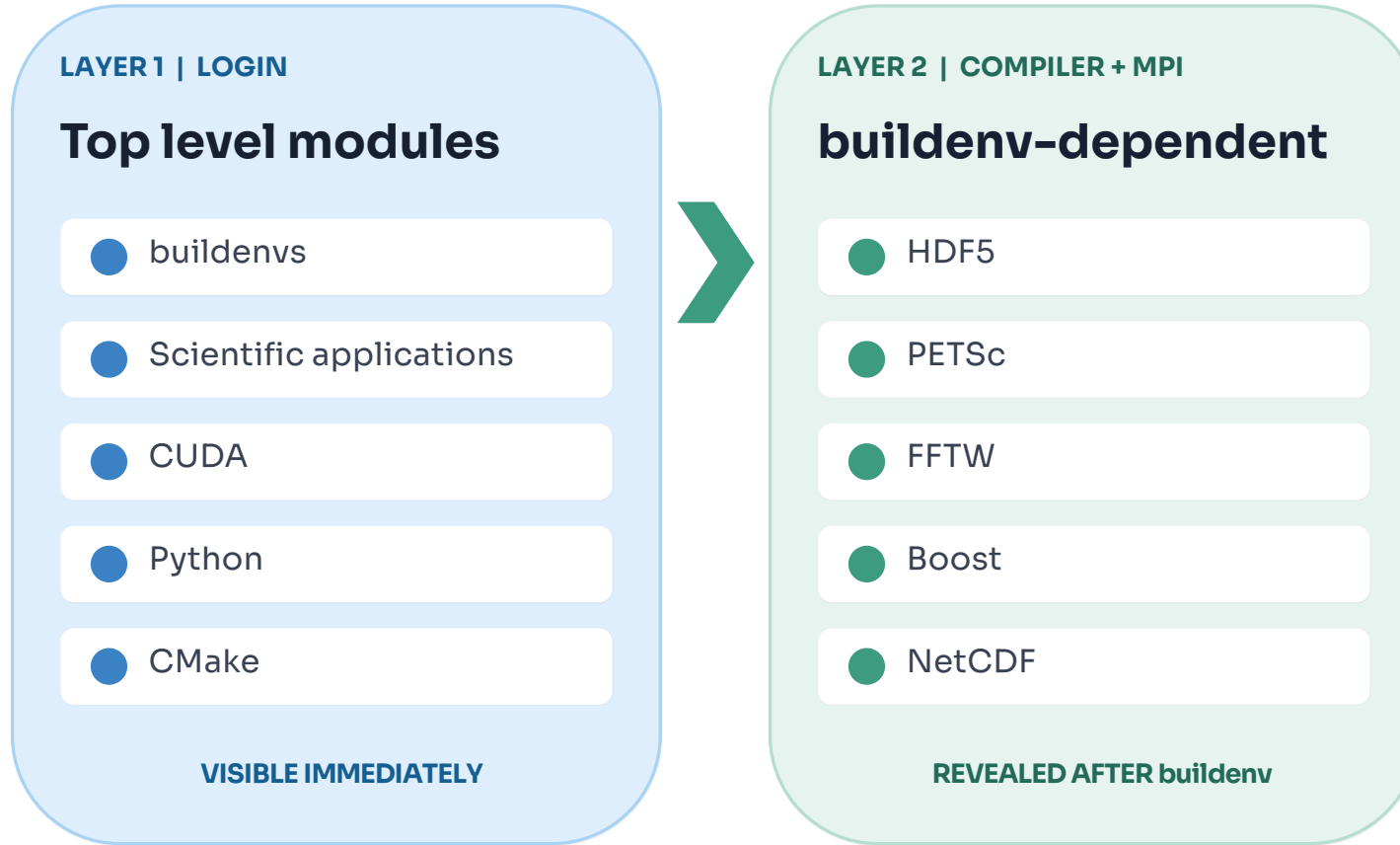
Copy required results back before the job ends. Other nodes cannot see the data!

Rule of thumb: disk for capacity and persistence, flash for shared random I/O, node-local for temporary single-node I/O.

The software setup

Software modules – several layers

What you see at login is only the first layer. Loading a buildenv module reveals the next compatible layer.



KEY IDEA

module avail shows the current layer. Loading a module can extend MODULEPATH and make compatible modules visible.

Software not installed?

Start with the simplest reproducible option. Install privately when you can, use a container when the instructions assume a Linux package manager, and ask support when needed.

The software you need is not available as a module

1

Can you build or install it yourself?

INSTALL IN YOUR OWN USER SPACE

- Use a personal environment
- Keep versions and dependencies reproducible
- Test with a small job before scaling

You own updates and maintenance.

2

Are the instructions for Docker, Ubuntu or apt?

CREATE AN APPTAINER IMAGE

- Start from the published container or recipe
- Package the required OS-level dependencies
- Run the image with the scheduler

Support can help translate the recipe.

3

Unsure how to build, install or containerise?

ASK SUPPORT FOR HELP

- Send the software name and version
- Include a link to source or instructions
- Describe hardware and workflow needs

Support provides installation or guidance.

CLEAR REQUESTS GET FASTER HELP

If possible Include version, source or container link, installation instructions, required libraries, target CPU/GPU and a small test case.

Job allocations

Use Slurm with confidence

Start simple ▪ verify the allocation ▪ scale in controlled steps



1

Start with the simplest case

Use one task, one node and a short time limit.
Keep the first script minimal.

1 task • 1 node • short run



2

Test interactively

Use an interactive allocation for quick tests, command checks and troubleshooting.

`interactive (salloc)`

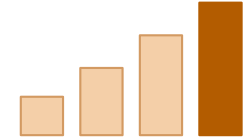


3

Check what Slurm granted

Compare requested resources with the job allocation and what the application actually uses.

`scontrol show job <jobid>`



4

Scale one dimension at a time

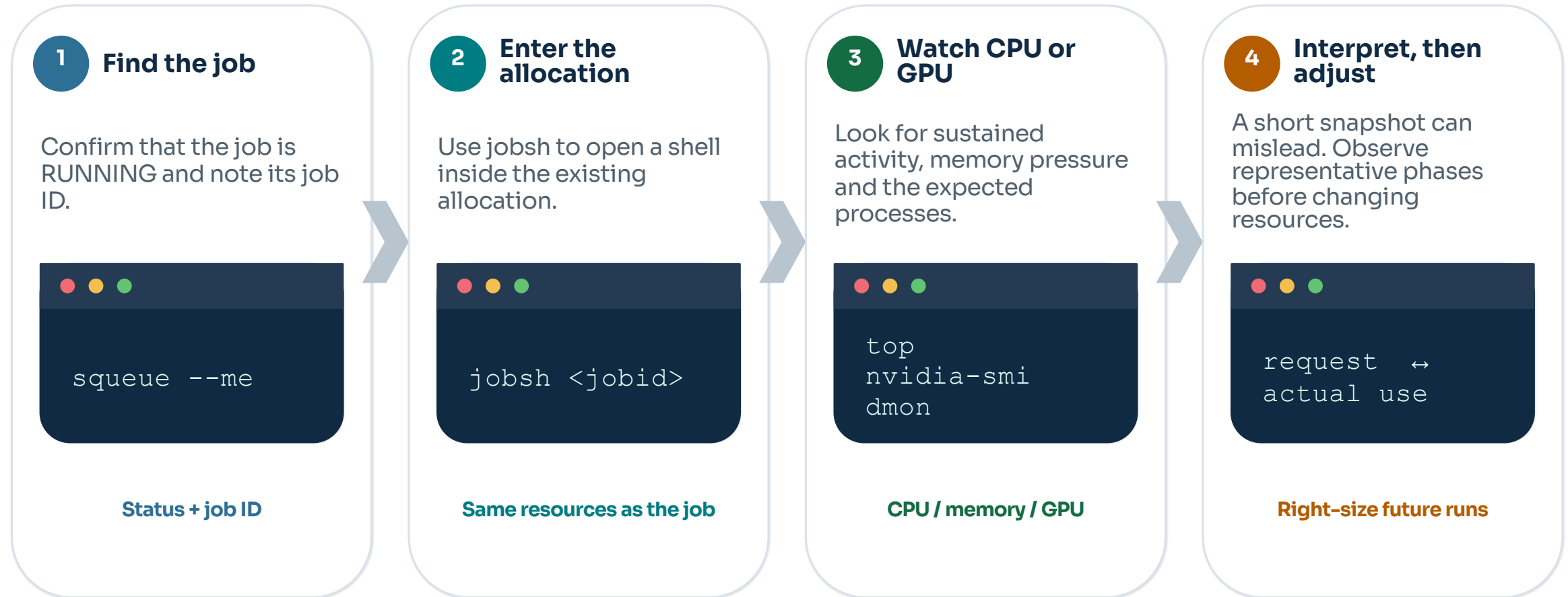
Increase cores, tasks, nodes or GPUs in small steps. Measure before increasing again.

1 → 2 → 4 → 8 ...

Rule of thumb: First ensure that the job is correct and well-matched to its allocation. Then scale it.

Inspecting a running batch job

Attach to the allocation ▪ observe the workload ▪ compare use with the request



Look for: expected processes ▪ active CPU cores ▪ memory headroom ▪ GPU utilisation and memory ▪ changes over time

Some closing remarks

Help us help you



01

Contact us early

Don't suffer in silence. Reach out when you are stuck so we can help move things forward.



02

One issue per ticket

A focused ticket is easier to understand, route and follow through to resolution.



03

Several issues? Several tickets

Submit separate tickets when you have multiple, unrelated issues.



04

Prefer the web form

It shows what information we need and adds useful metadata to your request.

The goal: clear requests, better routing and faster progress



Thank you!