GOBIERNO DE ESPAÑA · MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Instituto de Salud Carlos III

Running processes using a job scheduler (Slurm). Resource reservation and working directories

First steps with slurm

BU-ISCIII

29-04 de octubre de 2025

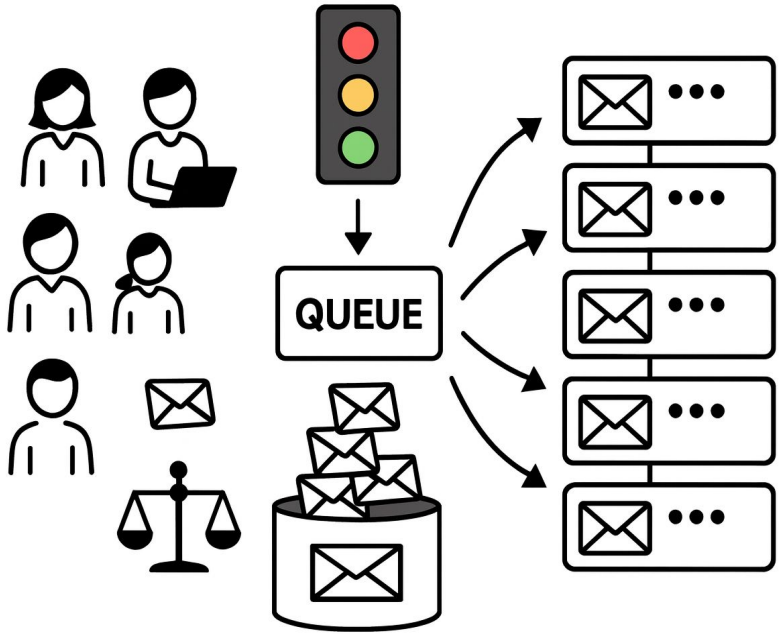
1ª edición

Outline

1. Introduction to job scheduling
2. Working with partitions and resources
3. Basic Slurm workflow
4. Working directories and storage policy
5. Monitoring and job states
6. Best practices and policies
7. Hands on

Introduction to job scheduling: Why a scheduler?

- Many users share the cluster
- Resources are limited and need coordination
- Scheduler ensures fair access and efficiency
- Prevents overload of login nodes
- Automates queuing and execution

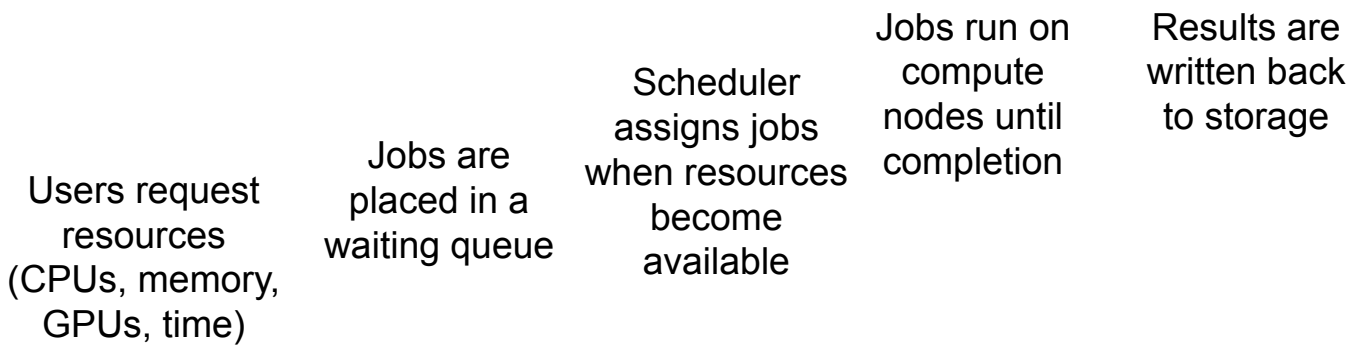


Quiz question

What would happen if all users ran jobs directly on the login node without a scheduler?

- A. Nothing, it would be the same as now
- B. Jobs would compete for resources, slowing down or crashing the node
- C. Only one user at a time could run jobs
- D. It would be more efficient

Introduction to job scheduling: How does a scheduler work?

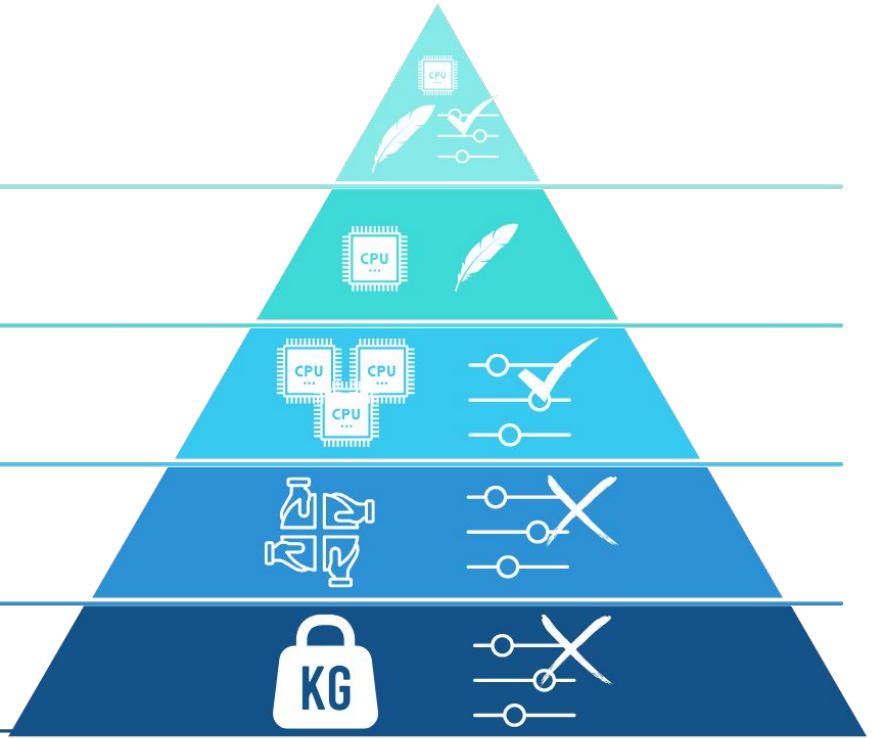


Introduction to job scheduling: Job prioritization

- Jobs rarely start immediately
- Scheduler assigns priorities to decide order
- Factors that affect priority:
  - **Fairshare:** recent resource usage by the user
  - **Job size:** more resources requested → harder to schedule
  - **Time requested:** shorter jobs often start earlier
  - **Partition rules:** limits on time, nodes, or users

Introduction to job scheduling: Job prioritization

1. Small jobs, good parametrization and parallelization
2. Small jobs, few cores
3. Good parametrization, cores
4. Poor parametrization, fairshare
5. Big jobs, poor parametrization



Quiz question

*What is the risk of asking for **much more memory or CPUs** than your job really needs?*

- A. Nothing, safer to over-request
- B. Your job gets lower priority, wastes resources
- C. The scheduler automatically adjusts to what you need
- D. It makes your job faster

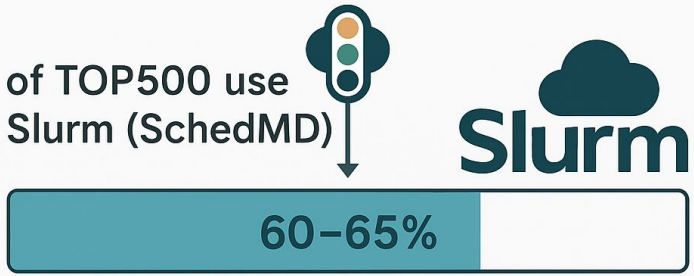
Introduction to job scheduling: Different job schedulers

Scheduler	Year	Notes
PBS (Portable Batch System)	1991	Developed by NASA, one of the earliest open-source schedulers. Basis for Torque.
Torque	2003	Fork of PBS, extended features, popular in early 2000s. Now legacy.
SGE (Sun Grid Engine)	2001	Widely adopted in academia, later forked into Open Grid Scheduler & Son of Grid Engine. Less use.
LSF (Load Sharing Facility)	1992	Developed by Platform Computing, later acquired by IBM. Commercial product, powerful but licensed.
Slurm	2002	Open source, highly scalable, now the most widely used in academic HPC and TOP500 systems.

Introduction to job scheduling: Why Slurm?

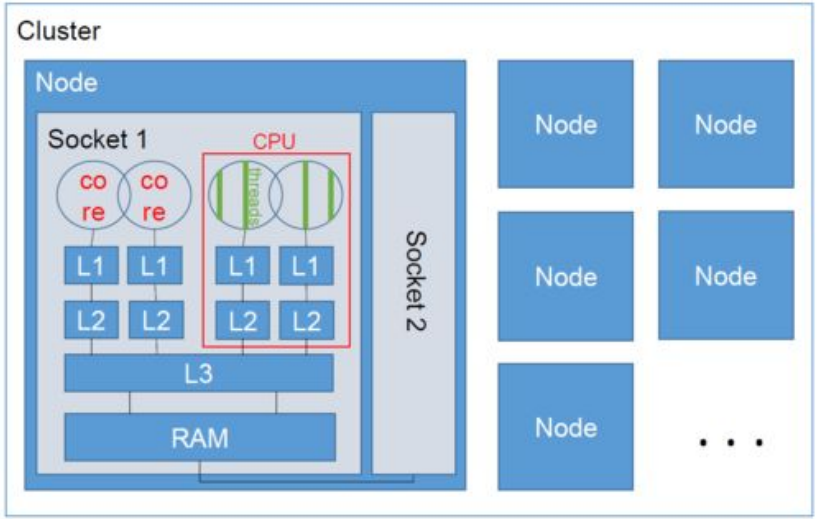
- Open source and free to use
- Highly scalable: from small clusters to supercomputers
- Runs on most TOP500 supercomputers
- Active community and strong documentation
- Flexible: supports CPUs, GPUs, memory, heterogeneous nodes
- Widely adopted in academic HPC centers

https://slurm.schedmd.com/quickstart.html



Introduction to job scheduling: inside a node

- Node = independent compute server in the cluster
- Contains multiple CPUs (sockets)
- Each CPU has several cores
- Each core may run multiple threads
- Node also provides memory and local disk (/local_scratch)
- Jobs request these resources through the scheduler



Slurm architecture

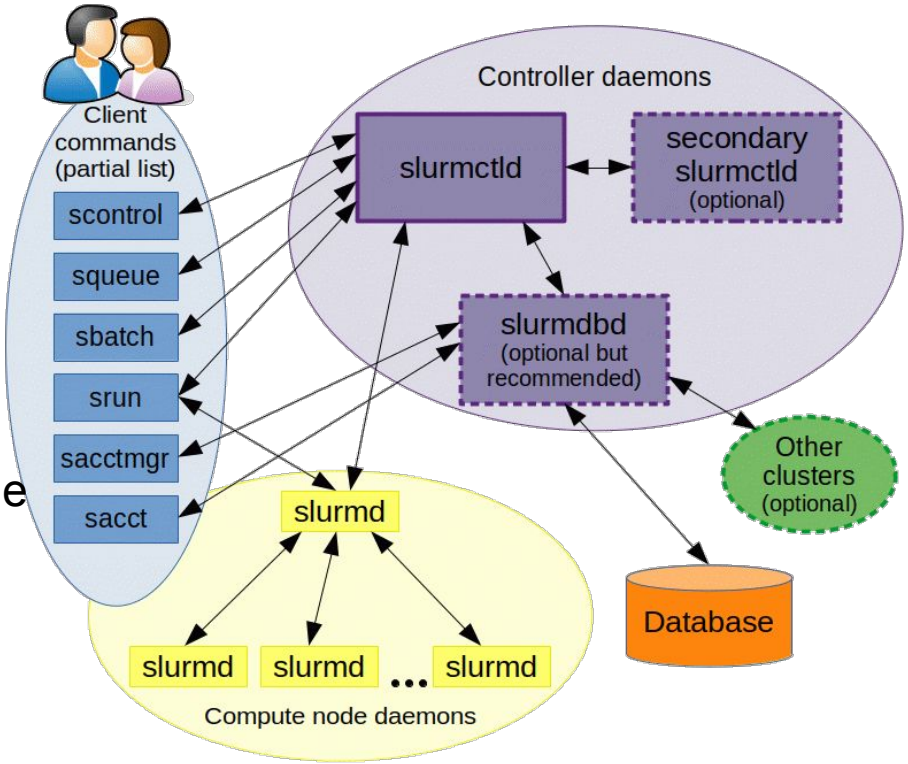
Slurm is organized as a set of daemons (small programs than run continuously):

slurmctld: central controller, manages jobs and resources

slurmd: runs on every compute node, launches and monitors tasks

slurmdbd (optional): tracks accounting and usage data

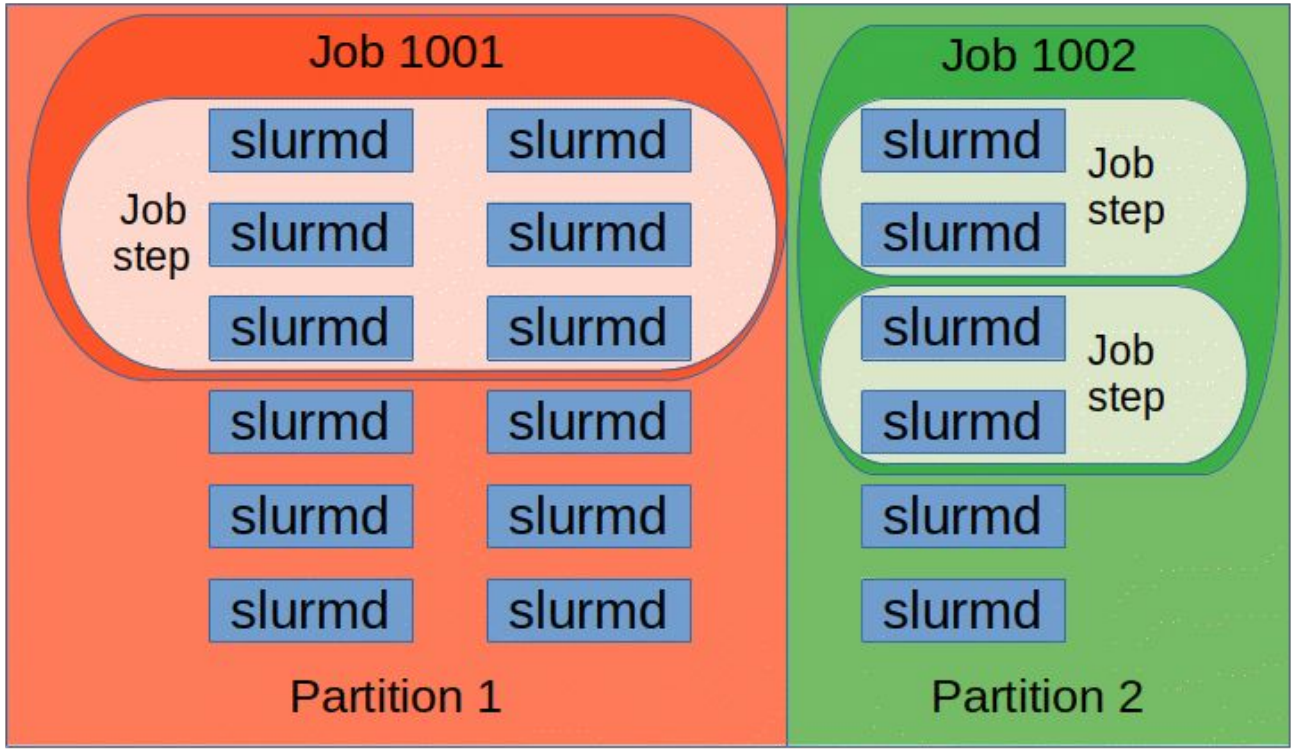
Entities managed: nodes, partitions, jobs, job steps, tasks



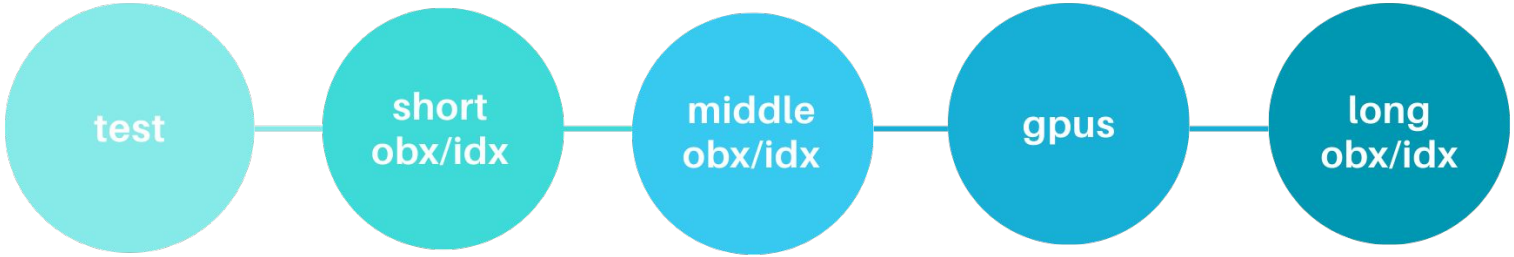
Introduction to job scheduling: Key concepts

Concept	Definition	Example
Job	Allocation of resources for a program during a fixed time	Reserve 4 CPUs and 16 GB RAM for 2 hours
Task	Program instance running inside a job (usually 1 per core)	A BLAST search using 1 CPU
Step	Subdivision of a job, serial or parallel sections	Preprocessing step → analysis step → postprocessing step
Partition	Queue grouping nodes with similar features and limits	short (12 h), gpu (72 h)
Resources	Elements assigned to the job	CPUs, memory, GPUs, execution time

Introduction to job scheduling: Job, step, task



Partitions and resources



max time	12h	12h	48h	72h	120h
max cpus	10	20/32	20/32	20+gpus	20/32
available gpus	✗	✗	✗	✓	✗
max nodes/job	2	6	16	2	10

Partitions and resources

- **So…in your jobs you can request:**
  - 🖥️CPUs (cores)
  - 💾Memory (per CPU or per node)
  - 🎮GPUs (when available)
  - ⏱️Wall time (execution time limit)

Slurm commands

Command	Description
⚡ **srun**	Run a job interactively or launch tasks inside a job allocation
🖥️ **salloc**	Allocate resources interactively for running commands
📄 **sbatch**	Submit a job script to the queue (batch mode)
📊 **squeue**	Show the status of jobs in the queue
📑 **sacct**	Display accounting info for completed jobs
❌ **scancel**	Cancel jobs in the queue or running

Slurm commands: srun

- Runs a job interactively (direct execution)
- Can also launch tasks inside a job allocation
- Useful for testing and short jobs
- By default runs on 1 CPU, short partition, limited resources

Example (minimal usage)

```
$ srun command
```

Output: shows the compute node assigned, e.g. `ideafix01`

srun basic options

- **Partition**: `--partition=test` → select queue
- **Time**: `--time=00:05:00` → max wall time
- **CPUs**: `--cpus-per-task=4` → reserve cores
- **Memory**: `--mem=8G` → RAM per node
- Combine them in one command

```
$ srun --partition=test \
    --time=00:05:00 \
    --cpus-per-task=4 \
    --mem=8G \
    /bin/hostname
```

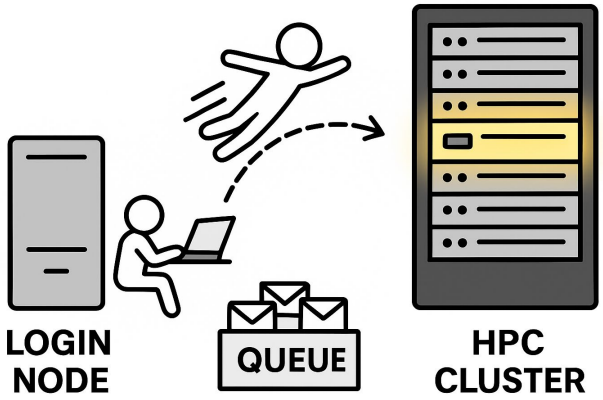
srun advance options

- Launch parallel tasks across CPUs or nodes
- **Tasks per node**: `--ntasks=8`
- **Nodes**: `--nodes=2`
- **GPUs**: `--gres=gpu:1`
- Useful for MPI, OpenMP, GPU jobs
- Can combine with interactive session (`salloc`)

```
$ srun --partition=gpus \
    --nodes=2 \
    --ntasks=8 \
    --gres=gpu:1 \
    ./my_parallel_program
```

srun interactive mode

- `srun` can open an **interactive shell** on a compute node
- Useful for quick testing and debugging
- You get a prompt directly on the assigned node
- All commands run within the allocated resource




```
$ # Request an interactive shell for 30 minutes with 2 CPUs
srun --partition=test --time=00:30:00 --cpus-per-task=2 --pty bash

# Prompt changes → now you are on the compute node
[username@ideafix03 ~]$
```

srun useful options

Output files

- `--output=out_%j.log` → save stdout (`%j` = job ID)
- `--error=err_%j.log` → save stderr separately

Working directory

- `--chdir=/scratch/unidad/$USER/job123` → run job in specific path

Node selection

- `--nodelist=ideafix05` → force job to run on a node (use with caution)
- `--exclude=ideafix[03,04]` → avoid certain nodes
- `--partition=short_idx`

srun useful options

```
$ srun --partition=test \
    --time=00:05:00 \
    –partition=short_idx  \
    --cpus-per-task=2 \
    --output=out_%j.log \
    --error=err_%j.log \
    --chdir=/scratch/unidad/$USER/test \
    /bin/hostname
```

Slurm commands: salloc

- Allocates resources interactively
- Opens a shell on the login node with reserved resources
- From there, run commands with `srun`
- Useful for testing workflows and debugging

```
$ salloc --partition=test
--time=00:10:00 --cpus-per-task=2
--mem=4G
srun hostname
```

Slurm commands: salloc

- Allocates resources interactively
- Opens a shell on the login node with reserved resources
- From there, run commands with `srun`
- Useful for testing workflows and debugging

```
$ salloc --partition=test
--time=00:10:00 --cpus-per-task=2
--mem=4G

srun hostname
```

Slurm commands: sbatch

- Submits a job script to the queue
- Script contains #SBATCH directives (resources, partition, time, etc.)
- Job runs non-interactively when resources are available
- Output and error logs captured automatically
- Main tool for production jobs

```
$ sbatch script.sbatch
```

Slurm commands: squeue

- Shows jobs in the queue (pending and running)
- Default: lists all jobs from all users
- Useful to monitor job status and queue load
- Typical fields: JOBID, USER, PARTITION, STATE, TIME, NODES

```
$ squeue

JOBID  USER   PARTITION  STATE   TIME   NODES  NODELIST
12345  user1  short_obx  R       0:10   1      ideafix01
12346  user2  long_obx   PD      0:00   4      (None)
```

squeue filters and formats

- Show only your jobs:

```
$ squeue -u $USER
```

- Estimate start time:

```
$ squeue --start
```

- Customize output:

```
squeue -o "%.18i %.9P %.8j %.8u %.2t %.10M %.6D %R"
```

Sort by priority or time: `--sort=P` or `--sort=S`

Slurm commands: sacct

- Shows information about **completed jobs**
- **Default**: lists jobs from current day
- **Fields**: JobID, JobName, Partition, State, ExitCode, Elapsed, CPUTime, Memory
- Useful to check **resource usage** and debug **job failures**

```
$ sacct

JobID    JobName    Partition    State        ExitCode    Elapsed    MaxRSS
12345    myjob      short_obx    COMPLETED    0:0         00:10:05   2048K
12346    myjob2     long_obx     FAILED       1:0         00:00:03   1024K
```

sacct filters and formats

- Show jobs from a date range:

```
$ sacct -S 2025-08-01 -E 2025-08-25
```

- Show only your jobs:

```
$ sacct -u $USER
```

- Customize fields (recommended):

```
$ sacct -o
JobID,JobName%15,State,Elapsed,MaxRSS,AllocCPUS
```

- Display steps inside a job with `-j JOBID`

Slurm commands: scancel

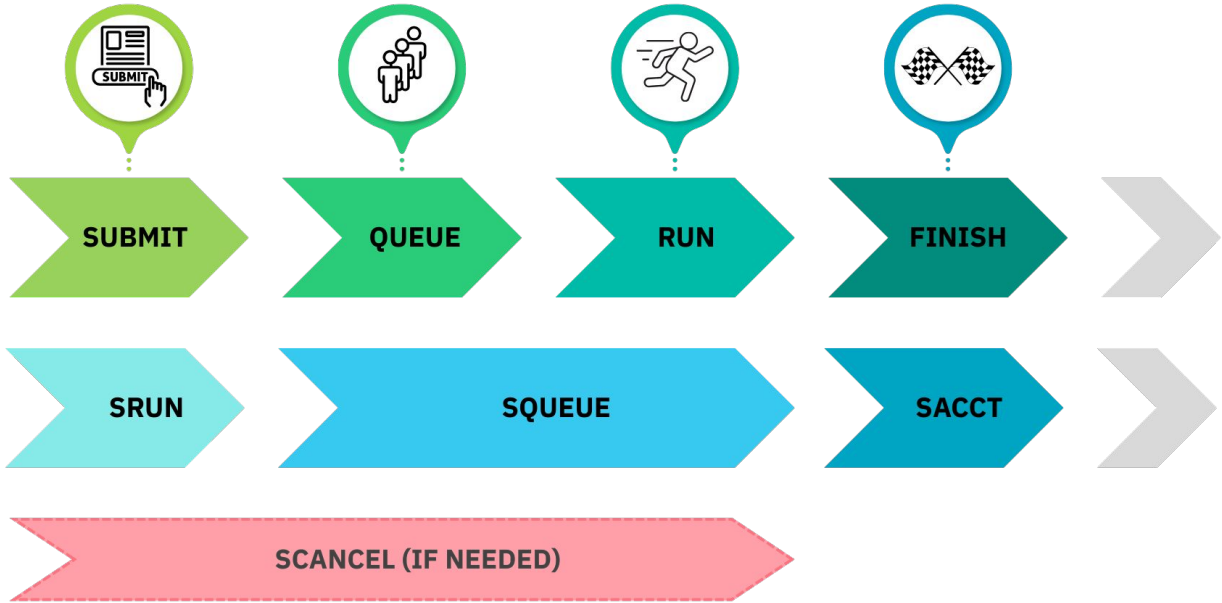
- Cancels jobs in the queue or currently running
- Requires the JobID
- Useful when job is misconfigured or no longer needed
- Can cancel multiple jobs with filters

```
# Cancel a single job
$ scancel 12345

# Cancel all your pending jobs
$ scancel -t PD -u $USER

# Cancel all your jobs
$ scancel -u $USER
```

Monitoring and job states

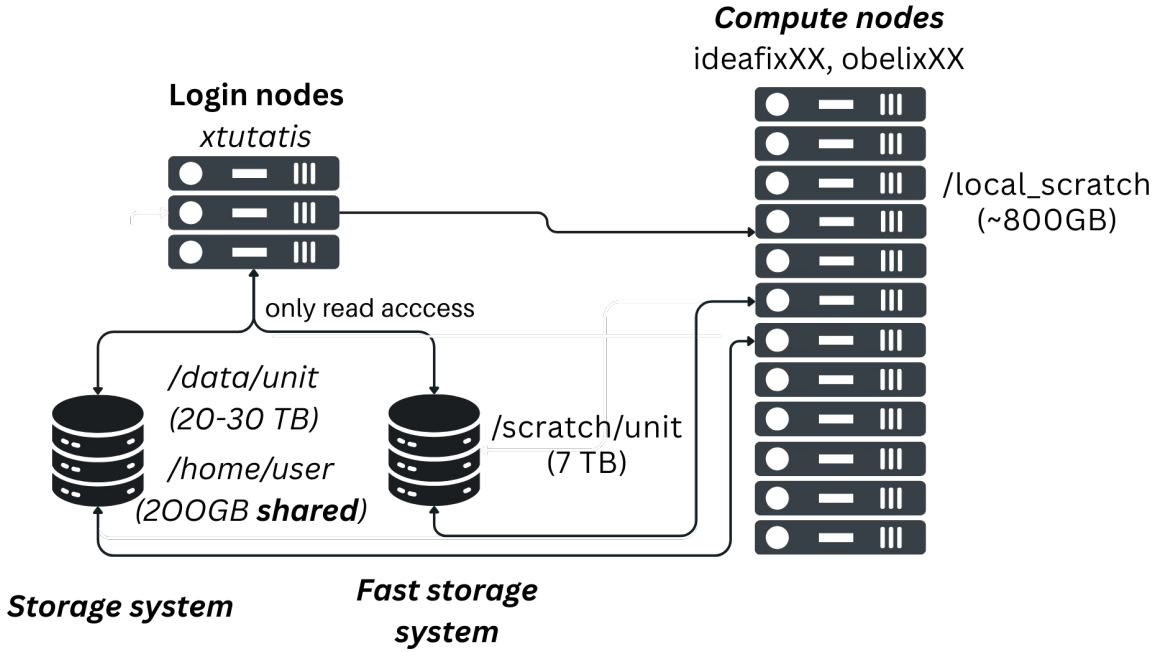
Code	State	Meaning
🟡 **PD**	Pending	Waiting for resources or priority
🟢 **R**	Running	Currently executing on allocated nodes
🟡 **CG**	Completing	Finishing processes, cleaning up
🟢 **CD**	Completed	Finished successfully
🔴 **F**	Failed	Ended with error or non-zero exit code
🔴 **TO**	Timeout	Cancelled for exceeding time limit
🔴 **CA**	Cancelled	Cancelled by user or administrator

Monitoring and job states



Working directories and permissions: Filesystems in Xtutatis

- **/home:** personal, small scripts & configs
- **/data/unidad:** project results, shared per unit
- **/scratch/unidad:** main space for jobs, purged if inactive >5 days
- **/local_scratch:** per-node temporary SSD, auto-cleaned
- **/srv/fastq_repo**



Working directories and permissions: Permissions and usage

- `/scratch` → **read-only** on login node, **read/write** on compute nodes
- Always write to `/scratch` through a job (`srun`, `salloc`, `sbatch`)
- Use `/data` to store and share results across projects
- Copy files between areas depending on workflow

❌ Fails from login node:

```
$ rsync input.fq /scratch/unidad/
# Permission denied
```

✅ Works via compute node:

```
$ srun --partition=test --time=00:05:00 \
    -- rsync -av ~/input.fq /scratch/unidad/
```

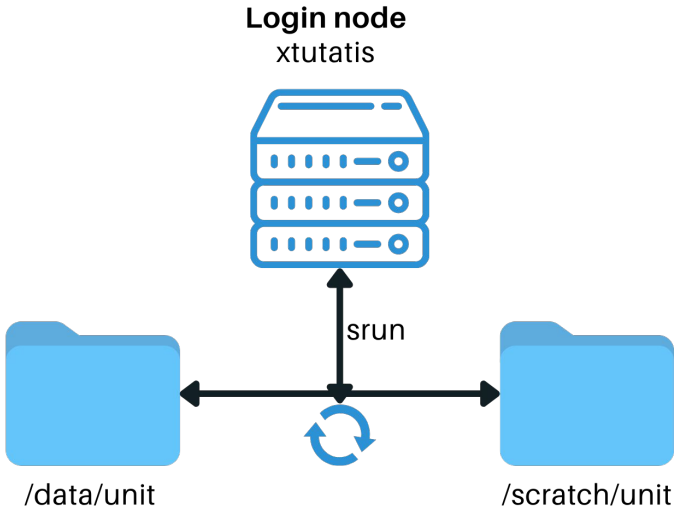
Working directories and permissions: Permissions and usage

To copy into `/scratch`, run the copy **through a compute node** with `srun`

Use `rsync` instead of `cp` for safer and faster transfers

- Preserves permissions, timestamps, directory structures
- Efficient for large datasets

```
# Copy to scratch
$ srun --partition=test --time=00:05:00 \
    -- rsync -av ~/input.fq /scratch/unidad/

# Copy back to /data
srun --partition=test --time=00:05:00 \
    -- rsync -av /scratch/unidad/  /data/unidad/
```



Quiz question

*Why can you **not write to** `/scratch` from the login node?*

Options:

- A. Scratch is only mounted on GPUs
- B. It is mounted read-only on the login node
- C. Scratch is only for admins
- D. Scratch is archived automatically

Best practices and policies

- 🚫 Do not run heavy jobs on the login node
- ⚖️ Always request realistic resources (CPUs, memory, time)
- 📂 Use `/scratch` for execution, copy results back to `/data`
- 🧹 Clean up scratch space after jobs finish
- 🤝 Share resources responsibly, cancel misconfigured jobs
- 📖 Acknowledge HPC resources in publications

Key takeaways

- Slurm scheduler ensures fairness and efficient use of cluster resources
- Jobs must request resources (CPUs, memory, time, GPUs) to run properly
- Monitor jobs with `squeue` (active) and `sacct` (finished)
- Cancel misconfigured or unnecessary jobs with `scancel`
- Use `/scratch` only for execution, `/data` for long-term storage
- Keep scratch space clean, respect shared resources

Hands-on

- Use `srun` with basic options from `/home` (CPU, time, memory)
- Try `srun` with advanced options (nodes, ntasks, GPUs)
- Launch `srun` jobs in background (&) and check with `squeue`
- Open an interactive session with `srun --pty bash`

  Monitor jobs: list active with `squeue`, review finished with `sacct`
- Cancel jobs with `scancel` (by JobID, user, or job name)
- Copy data between `/data` and `/scratch` using `srun rsync`
- Create a data structure in `/scratch`, run `fastqc` on real data
- Copy results back to `/data` and clean `/scratch`

Thank you for your attention

Questions?