Ragnar HPC Cluster

High-Performance Computing for Astrophysics

Institute of Astrophysics, Universidad Andrés Bello

July 27, 2025

Agenda

- Cluster overview and hardware resources
- Architecture and partitions
- Storage and operating environment
- Access and account creation
- Software environment and modules
- Module management with Lmod & EasyBuild
- SLURM job management
- Batch scripts examples (20/28 cores)
- Interactive jobs & monitoring
- Parallel computing concepts
- Best practices & data management
- Graphical applications via ThinLinc
- Astrophysics use cases
- Conclusion & support
- Community & training

Introduction & Motivation

Purpose: provide massive computing power to model and analyse complex astrophysical phenomena and to accelerate scientific discovery.

Creation: the Institute of Astrophysics built Ragnar to support undergraduate projects, doctoral theses and collaborative research requiring high-performance computing.

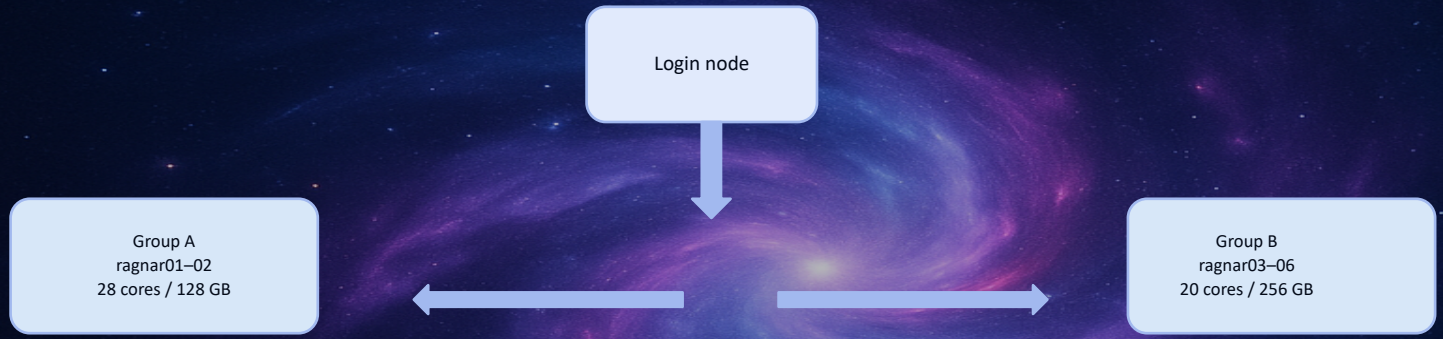
Users: open to bachelor and Ph.D. students, postdocs and faculty who need access to a high-performance cluster.

Hardware resources

Node group	CPU cores	Memory (GB)
ragnar01–02	28	128
ragnar03–06	20	256
Total	136	1280

Compute nodes: 6 nodes dedicated to simulations and data analysis.
Cores: 136 in total (28 cores on ragnar01–02, 20 cores on ragnar03–06).
Memory: 1.28 TB across the cluster.

Cluster architecture



Resource manager: SLURM
Operating system: CentOS (upgrade to Rocky Linux forthcoming)

Partitions and node states

```
[root@ragnar-01 ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
LocalQ*      up   infinite      1  down* ragnar-06
LocalQ*      up   infinite      2  alloc ragnar-[03-04]
LocalQ*      up   infinite      3   idle ragnar-[01-02,05]
[root@ragnar-01 ~]#
```

Partition: LocalQ* is the default queue; jobs submitted without specifying a partition will be queued here.

Node states: nodes can be down (offline for maintenance), alloc (allocated to running jobs) or idle (available).
Example: in the screenshot ragnar-06 is down, ragnar-03–04 are allocated, and ragnar-01–02,05 are idle.

Storage & filesystems

/mnt/nfs1 (home)
Small quota for scripts and code
Backed up daily
Shared across all nodes

/scratch
Local to each node
High I/O throughput
Not backed up; files purged regularly

Data management tips:
- Keep only scripts and small files in /mnt/nfs1.
- Use /scratch for large simulations and temporary output.
- Move important results back to your home directory for backup.

[6][7]

Operating environment

Operating system:
CentOS Linux on all nodes; upcoming upgrade to Rocky Linux.

Job scheduler:
SLURM handles job queues, resource allocation and monitoring.

Module system:
Lmod provides environment modules; EasyBuild manages reproducible software builds.

[8][9]

Access & login

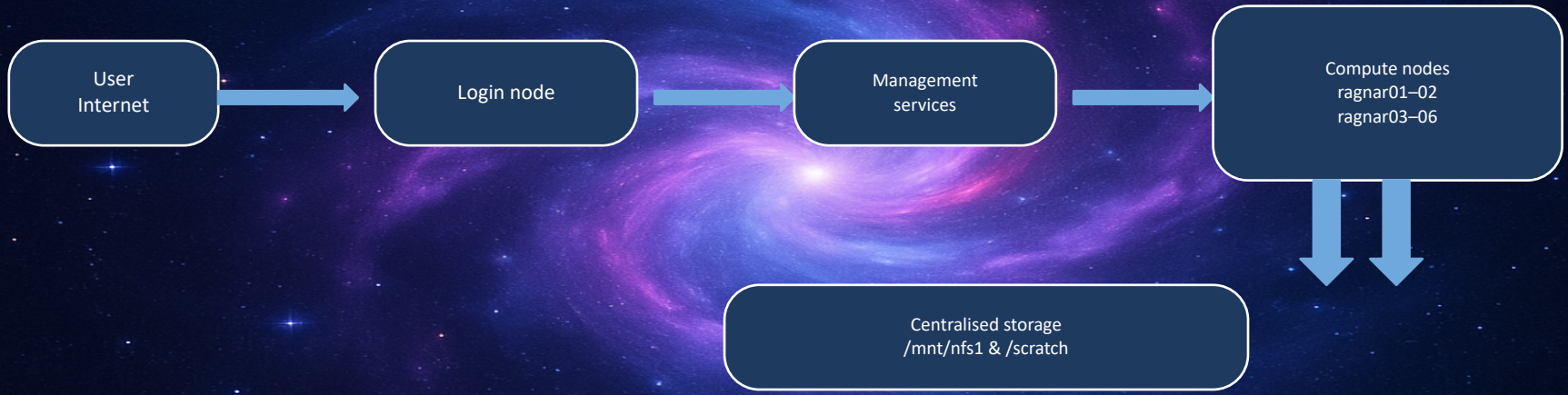
Network access:
connect from the university network or via VPN.

SSH connection:
use SSH to reach the login node before submitting jobs.

Etiquette:
never run heavy computations on the login node; only prepare scripts and monitor jobs from there.

```
$ ssh user@172.20.1.138
```

Connection workflow



Users connect via S$H from the Internet to the login node. Jobs are scheduled through management services to compute nodes. Compute nodes read and write data from shared storage (/mnt/nfs1) and local scratch for high-throughput I/O.

Account creation

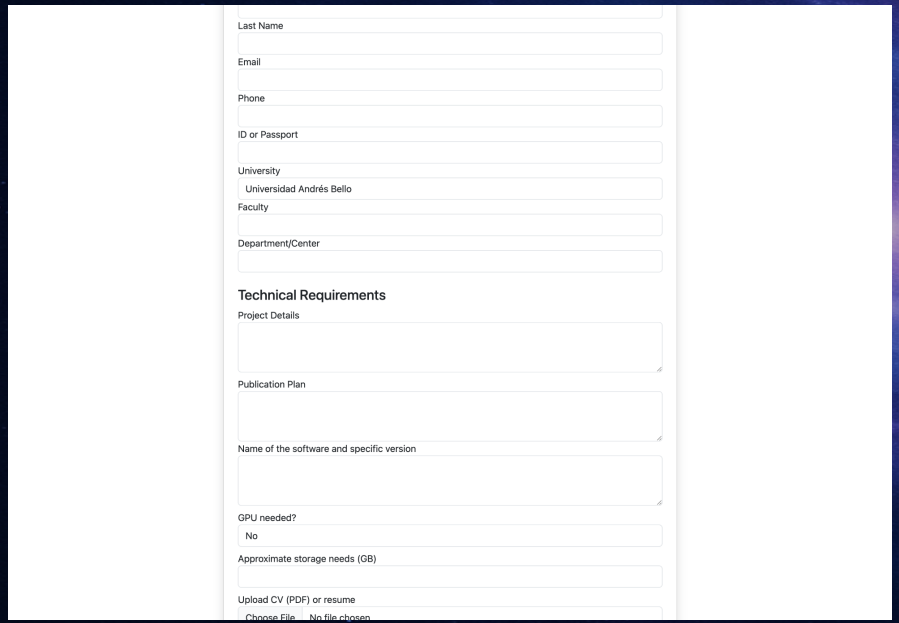
Submit your account request at astrounab.cl/cluster, where you can also find information about the cluster.

Evaluation criteria:

- Proposal quality – projects funded by national or international grants are favoured.
- Applicant CV – early career researchers are judged fairly based on stage.
- HPC experience – assess whether your programme will scale efficiently.
- Contributions – previous publications, theses and collaborations are considered.

[13][14]

Account request form



Access the form at https://astrounab.cl/cluster/ and be prepared to provide the following information:

General data: first and last name, institutional email, phone, ID/passport, university, faculty and department.

Technical requirements: project details, publication plan, software and version, need for GPUs, storage estimates and CV upload.

Policies & confirmation: you must accept the usage policies, cite the cluster in publications, authorise data use for statistics and confirm non-commercial use.

Policies & agreement

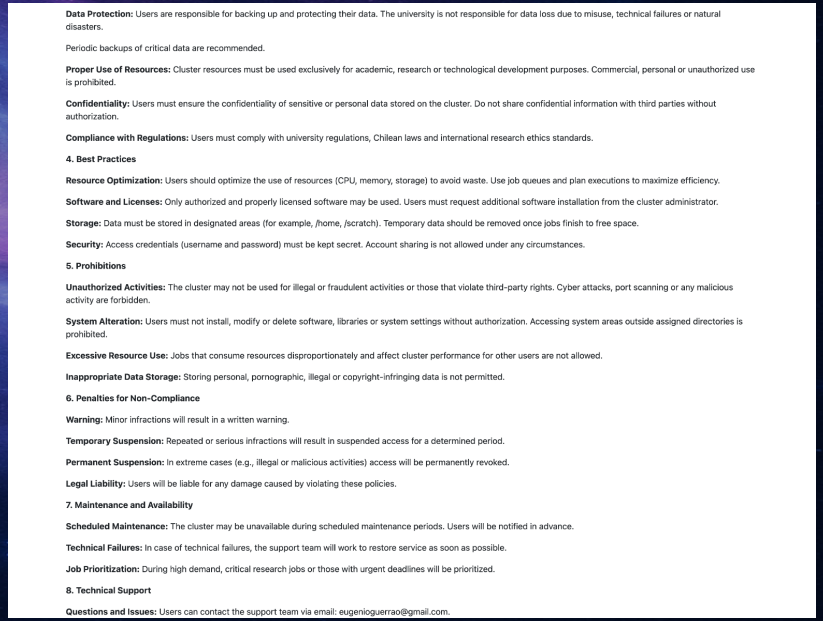
Purpose: HPC resources support research, teaching and technological development.

Authorised users: researchers, academics and students; external collaborators with approval.

Responsibilities: protect data, use resources only for scientific purposes and respect confidentiality.

Best practices: optimise resource usage, request licensed software properly and store data in designated areas (/home, /scratch).

Confirmation: users must accept these policies to obtain an account.



Data Protection: Users are responsible for backing up and protecting their data. The university is not responsible for data loss due to misuse, technical failures or natural disasters.

Periodic backups of critical data are recommended.

Proper Use of Resources: Cluster resources must be used exclusively for academic, research or technological development purposes. Commercial, personal or unauthorized use is prohibited.

Confidentiality: Users must ensure the confidentiality of sensitive or personal data stored on the cluster. Do not share confidential information with third parties without authorization.

Compliance with Regulations: Users must comply with university regulations, Chilean laws and international research ethics standards.

4. Best Practices

Resource Optimization: Users should optimize the use of resources (CPU, memory, storage) to avoid waste. Use job queues and plan executions to maximize efficiency.

Software and Licenses: Only authorized and properly licensed software may be used. Users must request additional software installation from the cluster administrator.

Storage: Data must be stored in designated areas (for example, /home, /scratch). Temporary data should be removed once jobs finish to free space.

Security: Access credentials (username and password) must be kept secret. Account sharing is not allowed under any circumstances.

5. Prohibitions

Unauthorized Activities: The cluster may not be used for illegal or fraudulent activities or those that violate third-party rights. Cyber attacks, port scanning or any malicious activity are forbidden.

System Alteration: Users must not install, modify or delete software, libraries or system settings without authorization. Accessing system areas outside assigned directories is prohibited.

Excessive Resource Use: Jobs that consume resources disproportionately and affect cluster performance for other users are not allowed.

Inappropriate Data Storage: Storing personal, pornographic, illegal or copyright-infringing data is not permitted.

6. Penalties for Non-Compliance

Warning: Minor infractions will result in a written warning.

Temporary Suspension: Repeated or serious infractions will result in suspended access for a determined period.

Permanent Suspension: In extreme cases (e.g., illegal or malicious activities) access will be permanently revoked.

Legal Liability: Users will be liable for any damage caused by violating these policies.

7. Maintenance and Availability

Scheduled Maintenance: The cluster may be unavailable during scheduled maintenance periods. Users will be notified in advance.

Technical Failures: In case of technical failures, the support team will work to restore service as soon as possible.

Job Prioritization: During high demand, critical research jobs or those with urgent deadlines will be prioritized.

8. Technical Support

Questions and Issues: Users can contact the support team via email: eugenioguerrao@gmail.com.

Software overview

Category	Representative modules
Astronomy	Astropy, IRAF, DS9, SExtractor
Languages	Python, Julia, MATLAB, R
Visualisation	Gnuplot, Matplotlib, ParaView
Utilities	Git, CMake, MPI, CUDA

Environment modules are grouped by toolchain and architecture.
Use Lmod commands to load, search and manage packages.

Managing modules

List loaded modules
Display currently loaded modules.

```
$ ml
```

List available modules
List all software grouped by toolchain.

```
$ ml av
```

Search for a package
Find all versions and prerequisites of a module.

```
$ ml spider python
```

Load a module
Activate a software version and its dependencies.

```
$ ml Python/3.11.3
```

Unload / purge modules
Remove a specific module or reset to a clean environment.

```
$ ml unload Python/3.11.3 ; ml purge
```

[18][19][20][21]

Available modules

```
[root@ragnar-01 ~]# ml av

------------------------------------------------------------ /mnt/nfs3/modules/.local/easybuild/modules/all ------------------------------------------------------------
   Aladin/12.060                                OpenSSL/1.1                                  hwloc/1.11.2-GCC-4.9.3-2.25
   Autoconf/2.69-GCC-4.9.3-2.25                 PCRE2/10.42-GCCcore-13.2.0                   hwloc/1.11.12-GCCcore-8.3.0
   Autoconf/2.69-GCCcore-8.3.0                  PMIx/4.2.4-GCCcore-12.3.0                    hwloc/2.9.1-GCCcore-12.3.0
   Autoconf/2.69                                PMIx/4.2.6-GCCcore-13.2.0             (D)    hwloc/2.9.2-GCCcore-13.2.0             (D)
   Autoconf/2.71-GCCcore-12.3.0                 Perl-bundle-CPAN/5.38.0-GCCcore-13.2.0       intltool/0.51.0-GCCcore-13.2.0
   Autoconf/2.71-GCCcore-13.2.0          (D)    Perl/5.30.0-GCCcore-8.3.0-minimal            jbigkit/2.1-GCCcore-13.2.0
   Automake/1.15-GCC-4.9.3-2.25                 Perl/5.30.0-GCCcore-8.3.0                    libGLU/9.0.3-GCCcore-13.2.0
   Automake/1.15                                Perl/5.36.1-GCCcore-12.3.0                   libarchive/3.6.2-GCCcore-12.3.0
   Automake/1.16.1-GCCcore-8.3.0                Perl/5.38.0-GCCcore-13.2.0            (D)    libarchive/3.7.2-GCCcore-13.2.0        (D)
   Automake/1.16.5-GCCcore-12.3.0               Python-bundle-PyPI/2023.10-GCCcore-13.2.0    libdeflate/1.19-GCCcore-13.2.0
   Automake/1.16.5-GCCcore-13.2.0        (D)    Python/3.11.3-GCCcore-12.3.0                 libdrm/2.4.117-GCCcore-13.2.0
   Autotools/20150215-GCC-4.9.3-2.25            Python/3.11.5-GCCcore-13.2.0          (D)    libevent/2.1.12-GCCcore-12.3.0
   Autotools/20150215                           Rust/1.73.0-GCCcore-13.2.0                   libevent/2.1.12-GCCcore-13.2.0         (D)
   Autotools/20180311-GCCcore-8.3.0             SDL2/2.28.5-GCCcore-13.2.0                   libfabric/1.18.0-GCCcore-12.3.0
   Autotools/20220317-GCCcore-12.3.0            SEVN/2.10.0                                  libfabric/1.19.0-GCCcore-13.2.0        (D)
   Autotools/20220317-GCCcore-13.2.0     (D)    SExtractor/2.28.2-GCCcore-13.2.0             libffi/3.4.4-GCCcore-12.3.0
   BLIS/0.9.0-GCC-12.3.0                        SQLite/3.42.0-GCCcore-12.3.0                 libffi/3.4.4-GCCcore-13.2.0            (D)
   Bison/3.0.4-GCCcore-4.9.3                    SQLite/3.43.1-GCCcore-13.2.0          (D)    libglvnd/1.7.0-GCCcore-13.2.0
   Bison/3.0.4                                  ScaLAPACK/2.0.2-gompi-2016a-OpenBLAS-0.2.15-LAPACK-3.6.0    libiconv/1.17-GCCcore-12.3.0
   Bison/3.3.2-GCCcore-8.3.0                    ScaLAPACK/2.2.0-gompi-2023a-fb        (D)    libiconv/1.17-GCCcore-13.2.0           (D)
   Bison/3.3.2                                  Szip/2.1.1-GCCcore-12.3.0                    libjpeg-turbo/3.0.1-GCCcore-13.2.0
   Bison/3.8.2-GCCcore-12.3.0                   Tcl/8.6.13-GCCcore-12.3.0                    libpciaccess/0.14-GCCcore-8.3.0
   Bison/3.8.2-GCCcore-13.2.0            (D)    Tcl/8.6.13-GCCcore-13.2.0             (D)    libpciaccess/0.17-GCCcore-12.3.0
   Bison/3.8.2                           (D)    Tk/8.6.13-GCCcore-13.2.0                     libpciaccess/0.17-GCCcore-13.2.0       (D)
   Brotli/1.1.0-GCCcore-13.2.0                  UCC/1.2.0-GCCcore-12.3.0                     libpng/1.6.40-GCCcore-13.2.0
   CASA/6.2.1-7                                 UCC/1.2.0-GCCcore-13.2.0              (D)    libreadline/8.0-GCCcore-8.3.0
   CASA/6.7.0-31-py3.12                  (D)    UCX/1.14.1-GCCcore-12.3.0                    libreadline/8.2-GCCcore-12.3.0
   CFITSIO/4.3.1-GCCcore-13.2.0                 UCX/1.15.0-GCCcore-13.2.0             (D)    libreadline/8.2-GCCcore-13.2.0         (D)
   CMake/3.26.3-GCCcore-12.3.0                  UnZip/6.0-GCCcore-12.3.0                     libtool/2.4.6-GCC-4.9.3-2.25
   CMake/3.27.6-GCCcore-13.2.0           (D)    UnZip/6.0-GCCcore-13.2.0              (D)    libtool/2.4.6-GCCcore-8.3.0
   DB/18.1.32-GCCcore-8.3.0                     WRF/4.5.1-foss-2023a-dmpar                   libtool/2.4.6
```

The `ml av` command lists all available modules grouped by the toolchain used for their compilation.

Modules marked with (D) are default versions; others may require loading a toolchain first.

Use `ml spider <name>` to display descriptions, versions and prerequisites for a specific package.

[22][23]

EasyBuild & additional software

What is EasyBuild?
A framework that automates building and installing scientific software, ensuring reproducibility and managing dependencies.

Co-existence of versions: EasyBuild installs each version in a separate prefix and generates environment modules so you can switch between them.

Requesting new software: if a package is missing, contact the cluster administrator with the version and your project details. The team will evaluate feasibility and provide an EasyBuild recipe.

What is SLURM?

Definition:
SLURM (Simple Linux Utility for Resource Management) is an open-source workload manager used to allocate compute nodes, launch and schedule jobs and track resource usage.

Purpose:
co-ordinates queues (partitions), nodes and tasks so multiple users can share a cluster fairly and efficiently. It handles job submission, prioritisation, scheduling and accounting.

Global adoption:
SLURM is the de facto standard on most academic and national supercomputers worldwide due to its scalability, flexibility and open development model.

[25][26]

SLURM overview

Partitions: queues grouping nodes by characteristics; jobs must specify a partition (e.g., LocalQ*).

Tasks (-n): number of MPI processes to launch.
CPUs per task (-c): threads within each task (OpenMP or intra-process parallelism).
Memory (--mem): total RAM requested for the job.
Time (--time): wall-clock limit; jobs exceeding this limit are terminated.

Batch script: 1 task × 20 cores

```
#!/bin/bash
#SBATCH --job-name=python20
#SBATCH --partition=Local0
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=20
#SBATCH --mem=100G
#SBATCH --time=08:00:00
#SBATCH --output=python20_%j.out

module load python/3.11.3
srun python my_script.py
```

Submit with *sbatch script.sh*; monitor with *squeue*.

Batch script: 1 task × 28 cores

```
#!/bin/bash
#SBATCH --job-name=python28
#SBATCH --partition=Local0
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=128G
#SBATCH --time=12:00:00
#SBATCH --output=python28_%j.out

module load python/3.11.3
srun python heavy_task.py
```

Request more cores and memory for heavily parallel applications.

Command outputs

Node status (sinfo)

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES STATE NODELIST
LocalQ*    up    infinite    1     down*
ragnar-06
LocalQ*    up    infinite    2     alloc
ragnar-[03-04]
LocalQ*    up    infinite    3     idle
ragnar-[01-02,05]
```

Job queue (squeue)

```
$ squeue -u user
JOBID  USER  PARTITION  TIME      STATE
12345  user  LocalQ*    00:12:34  RUNNING
12346  user  LocalQ*    00:00:00  PENDING
```

Interactive shell (srun)

```
$ srun --pty bash
srun: job 12347 has been allocated resources
[user@login-node ~]$
```

[30][31][32]

Interactive jobs & monitoring

sinfo
Show node and partition status

```
$ sinfo
```

squeue
List queued and running jobs

```
$ squeue
```

sbatch
Submit a batch script

```
$ sbatch job.sh
```

srun
Run interactively in an allocated resource

```
$ srun --pty bash
```

scancel
Cancel a job

```
$ scancel <jobid>
```

sacct
Query completed jobs

```
$ sacct
```

[33][54]

Resource flags & parallelism

Selecting resources

-n (--ntasks): number of MPI processes to launch; use 1 for a single program or more for distributed parallelism.
-c (--cpus-per-task): threads per process (OpenMP). Only request extra threads when your code uses shared memory parallelism.
--mem: total RAM needed; request enough to prevent swapping.
--time: wall-clock limit; jobs exceeding it are terminated.

MPI (task parallelism)
4 independent processes with one thread each.

```
$ sbatch --ntasks=4 --cpus-per-task=1 --
mem=8G --time=1:00:00 script.sh
```

OpenMP (data parallelism)
1 process using 20 threads on a single node.

```
$ sbatch --ntasks=1 --cpus-per-task=20 --
mem=32G --time=2:00:00 script.sh
```

Hybrid
4 processes with 7 threads each.

```
$ sbatch --ntasks=4 --cpus-per-task=7 --
mem=48G --time=2:00:00 script.sh
```

[35][56][37]

Parallel computing models

Task parallelism (MPI)

Independent processes with private memory communicate via message passing across nodes.

Data parallelism (OpenMP)

Threads share a memory space and divide data among themselves on a single node.

Hybrid

Combines MPI between nodes with OpenMP threads inside each node for many-core architectures.

[38][39][40]

Parallelism in practice

MPI program
Matrix multiplication using 4 MPI processes across nodes.

```
$ mpirun -n 4 python matrix_mpi.py
```

OpenMP program
Large array computation with 20 threads on one node.

```
$ OMP_NUM_THREADS=20 python matrix_openmp.py
```

Hybrid program
4 MPI processes each launching 7 threads.

```
$ mpirun -n 4 OMP_NUM_THREADS=7 python
hybrid_code.py
```

[41][42][43]

Best practices

- Do not run heavy computations on the login node—submit jobs through SLURM.
- Organise your scripts and output files clearly.
- Request only the CPUs, memory and time you actually need.
- Use /scratch for large temporary data; /mnt/nfs1 is for code and small files.
- Avoid saturating I/O with continuous large reads or writes.
- Cancel jobs you no longer need to free resources.

[44][45][46]

Data management

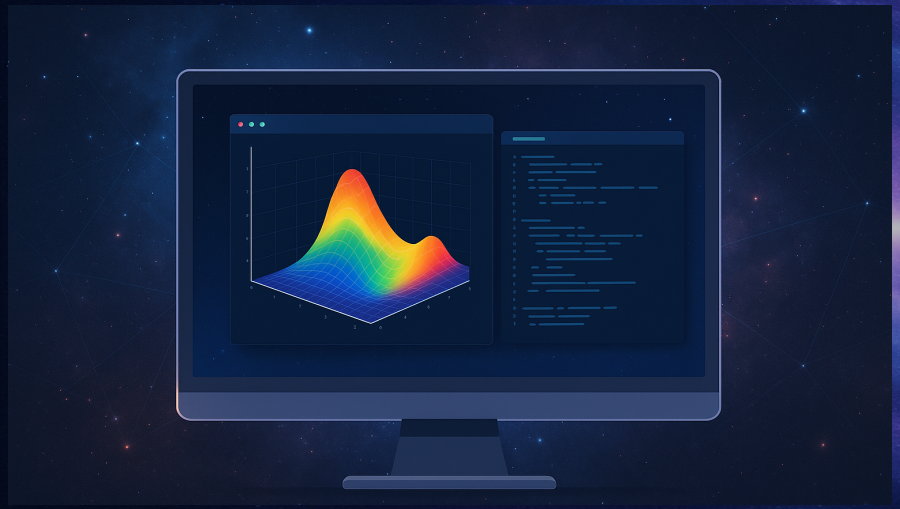
Home vs scratch:
store your code and small files in /mnt/nfs1; it is backed up but has limited quota.
use /scratch on the compute node for large data and intermediate results. Scratch is faster but not backed up and files may be deleted after inactivity.

Moving data:
regularly archive your results back to your home directory or external storage after jobs finish.

Cleanup: remove unnecessary files from scratch to make room for others and avoid policy-driven purges.

[47][48]

Graphical applications with ThinLinc



ThinLinc is a remote desktop service allowing you to run graphical applications on the cluster.

Benefits:
- Visualise large datasets without downloading them.
- Maintain long-running sessions without tmux or screen.
- Launch GUI tools like DS9 and TOPCAT with access to compute nodes.

Caveats:
– ThinLinc can be less stable than SSH; reconnect if disconnected.
– Do not run heavy simulations inside the ThinLinc session—submit jobs via SLURM.

[49][50][51][52]

Astrophysics use cases

- Simulating galaxy formation and cluster evolution.
- Modelling gravitational waves and calibrating cosmological parameters.
- Processing large astronomical catalogues and transient detection.
- Comparing performance: cluster vs laptop.

[53]

Conclusion & support

Empower your research: the Ragnar cluster offers 136 cores and over 1 TB of memory for astrophysical simulations that exceed laptop capabilities.

Optimise resource usage: plan jobs with SLURM, choose the right number of tasks and threads, and leverage /scratch for temporary data.

Stay connected: visit astrounab.cl/cluster for documentation and forms, attend HPC workshops, and email eugenioguerrao@gmail.com with any questions.

[54][55][56]

Community & training

Internal documentation: visit astrounab.cl/cluster for updated policies, examples, best practices and account forms.

Support team: email eugenioguerrao@gmail.com with questions or software requests.

Training workshops: participate in HPC tutorials and astrophysics computing courses offered by the institute.

[57][58]