Fundamentals of high-performance computing

General online course

13 May, 2025

Agenda

01
Introduction to HPC

02
SURF facilities

03
Job submission

04
Environment modules

05
Hands-on session

Introduction to HPC

SURF

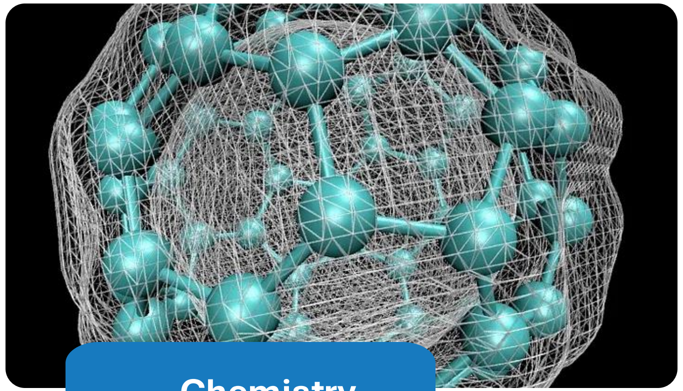
Science and HPC

Astrophysics

source

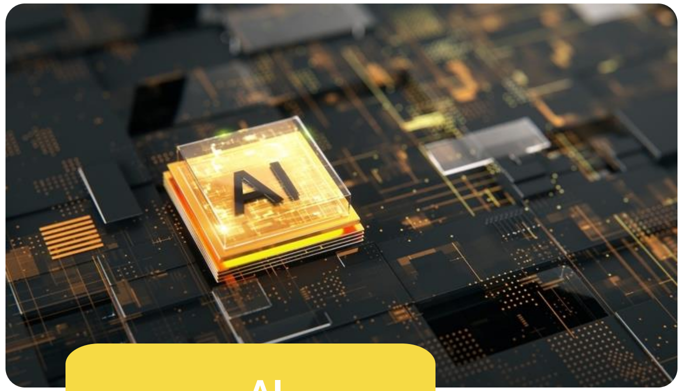
CFD

source



Chemistry

source



AI

source

Climate

source

Human behavior

source

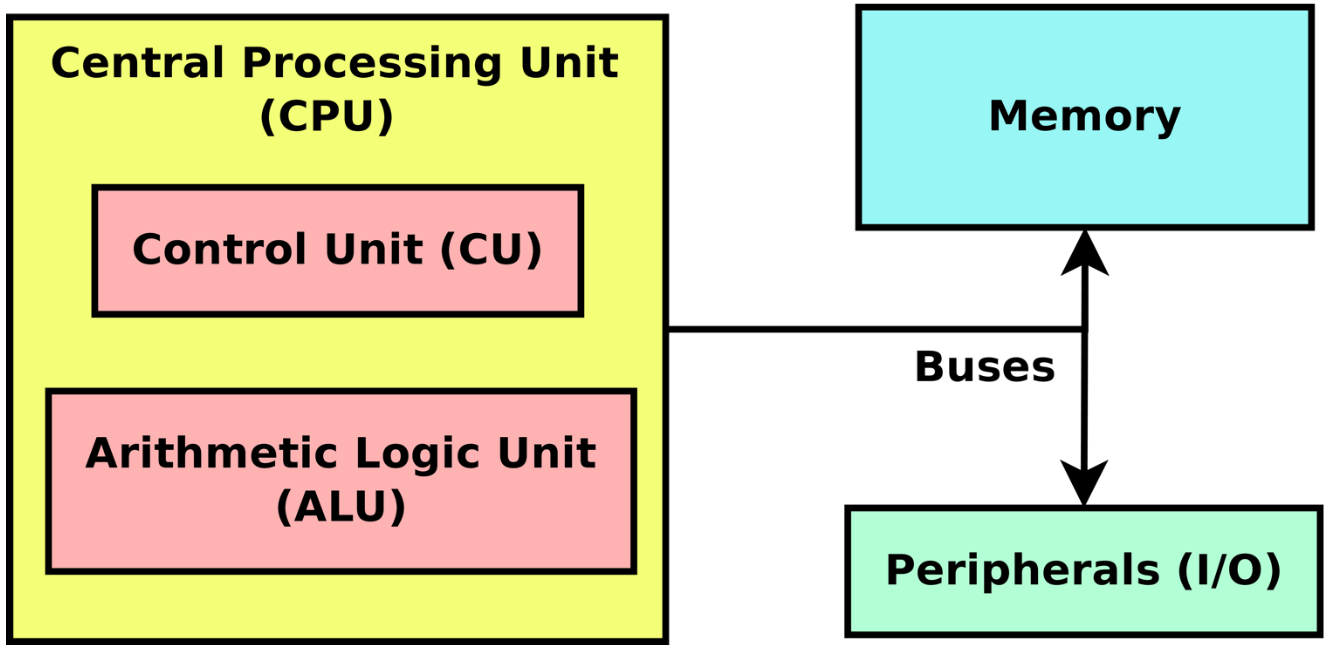
High-performance computing is

- an area of computer-based **computation**. It includes all computing work that requires a high computing capacity or storage capacity.
- the use of **parallel** processing for running advanced application programs efficiently, reliably and fast.
- the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve **large** problems in science, engineering, or business.
- the use of super computers and parallel processing techniques for solving **complex** computational problems.

Snellius Supercomputer

A computer is

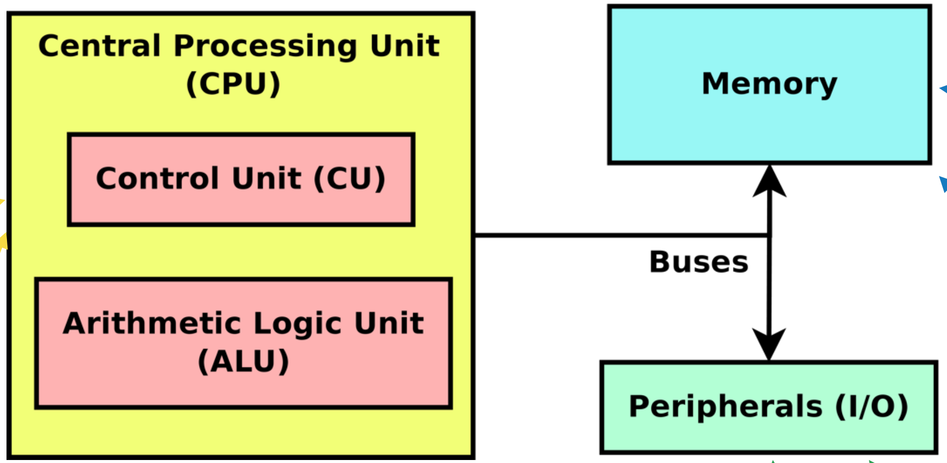
- a machine that processes data by executing instructions to perform calculations, automate tasks, and **solve complex problems** efficiently.
- an **electronic system** with a CPU (control unit and ALU) that processes data, transfers it via buses to memory (RAM, cache, storage), and interacts with peripherals.



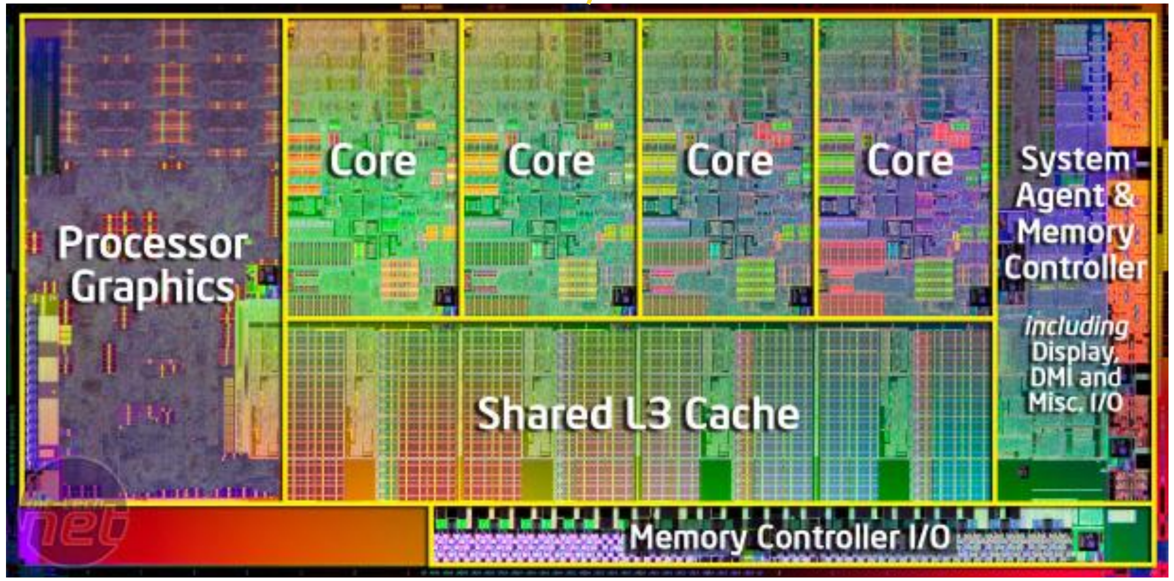
And Boolean logic, 0s and 1s!

A computer is **a programmable electronic system that processes data through hardware and software.** Hardware, including the **CPU (control unit and ALU), memory, buses, and peripherals**, handles computation, data transfer, and interaction. Software, such as the **operating system and applications**, directs hardware operations for automation, problem-solving, and data processing.

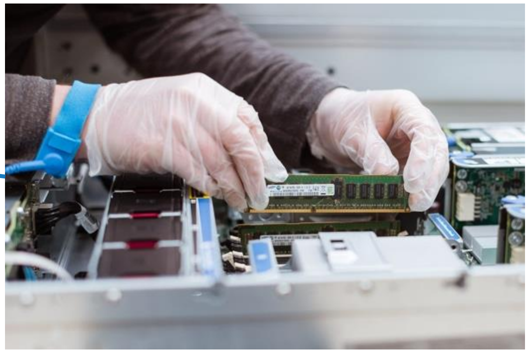
A computer is



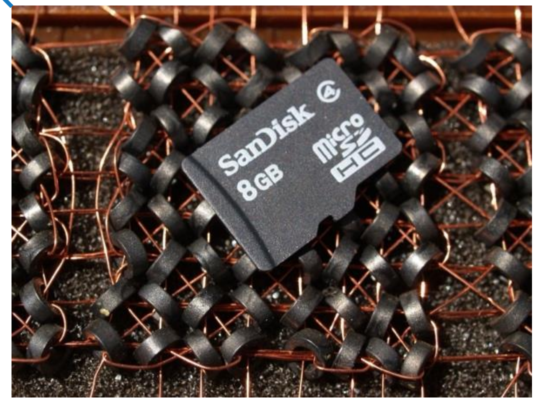
source



source



source

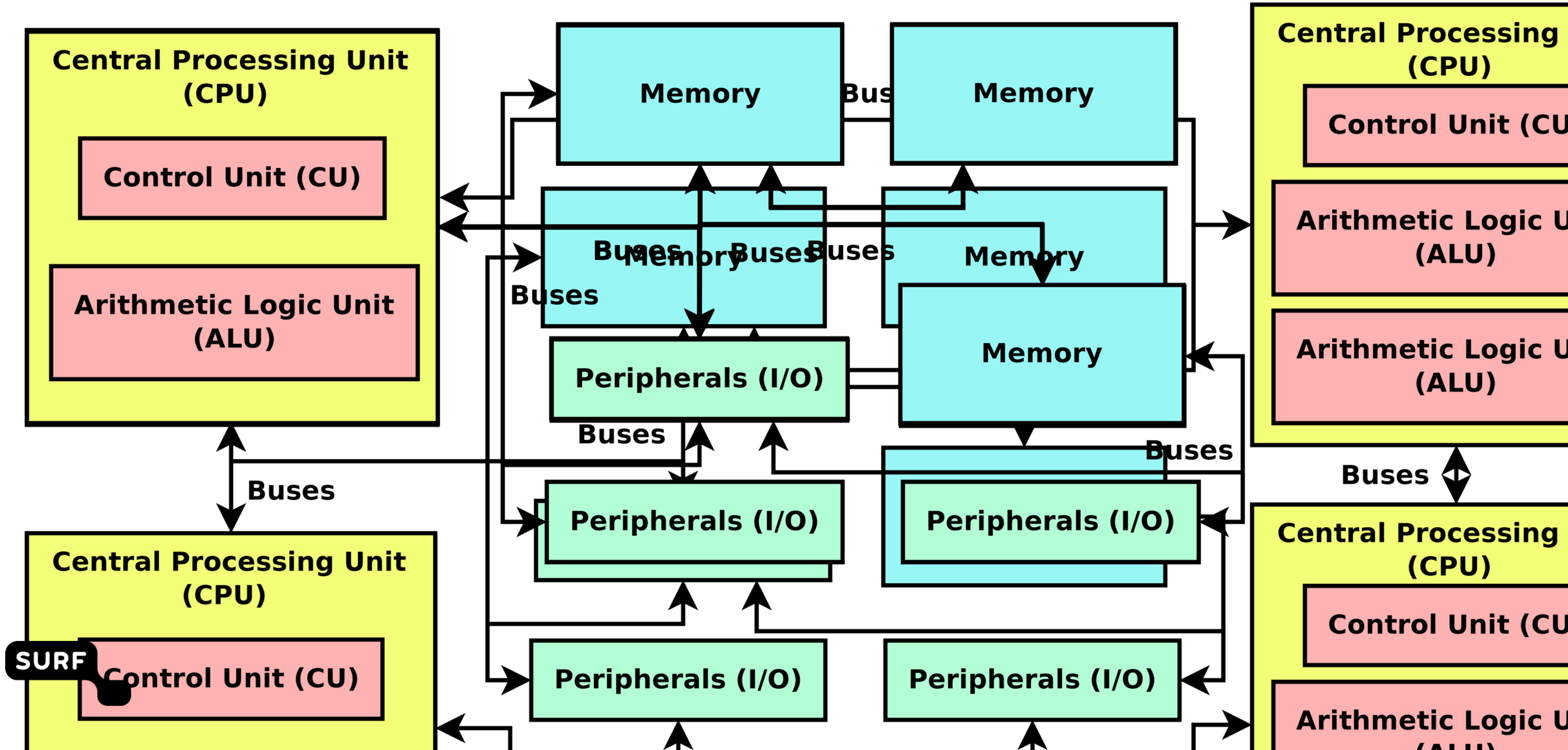


source

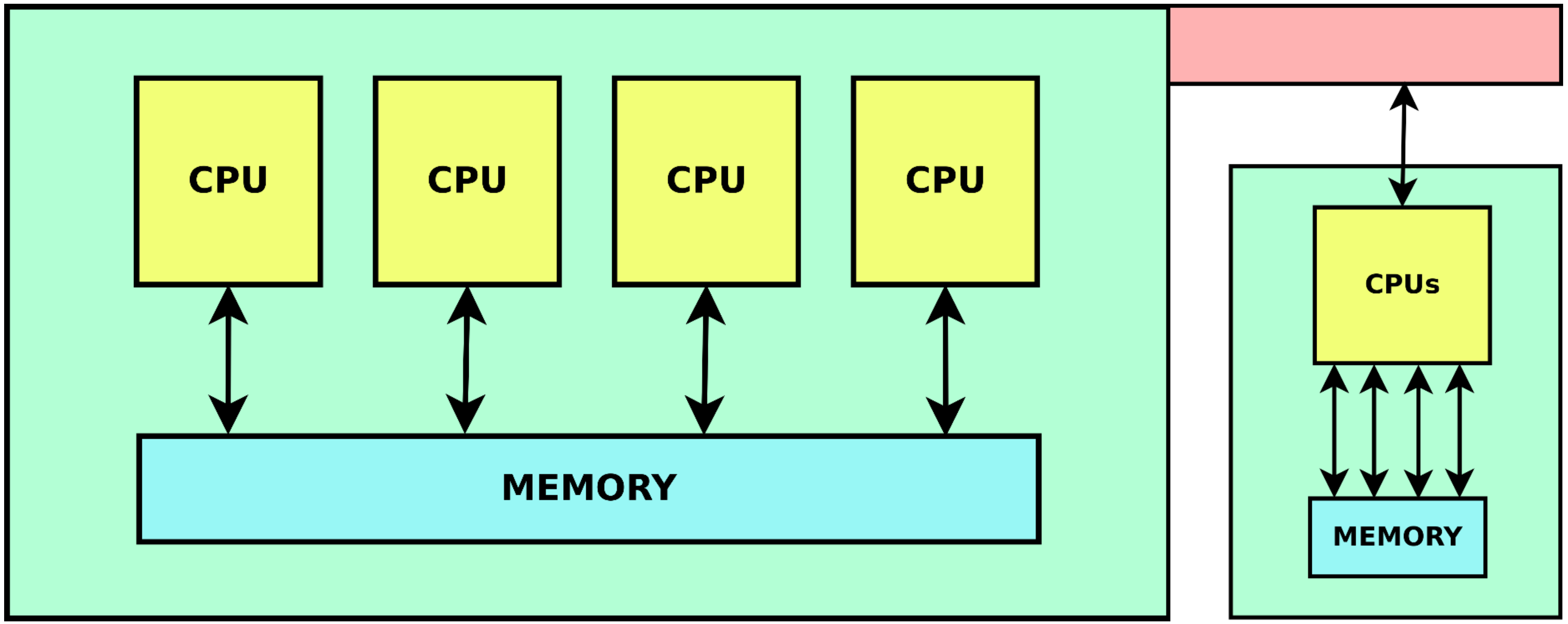
source

source

A larger computer could be



A larger computer actually is



High-performance computing is

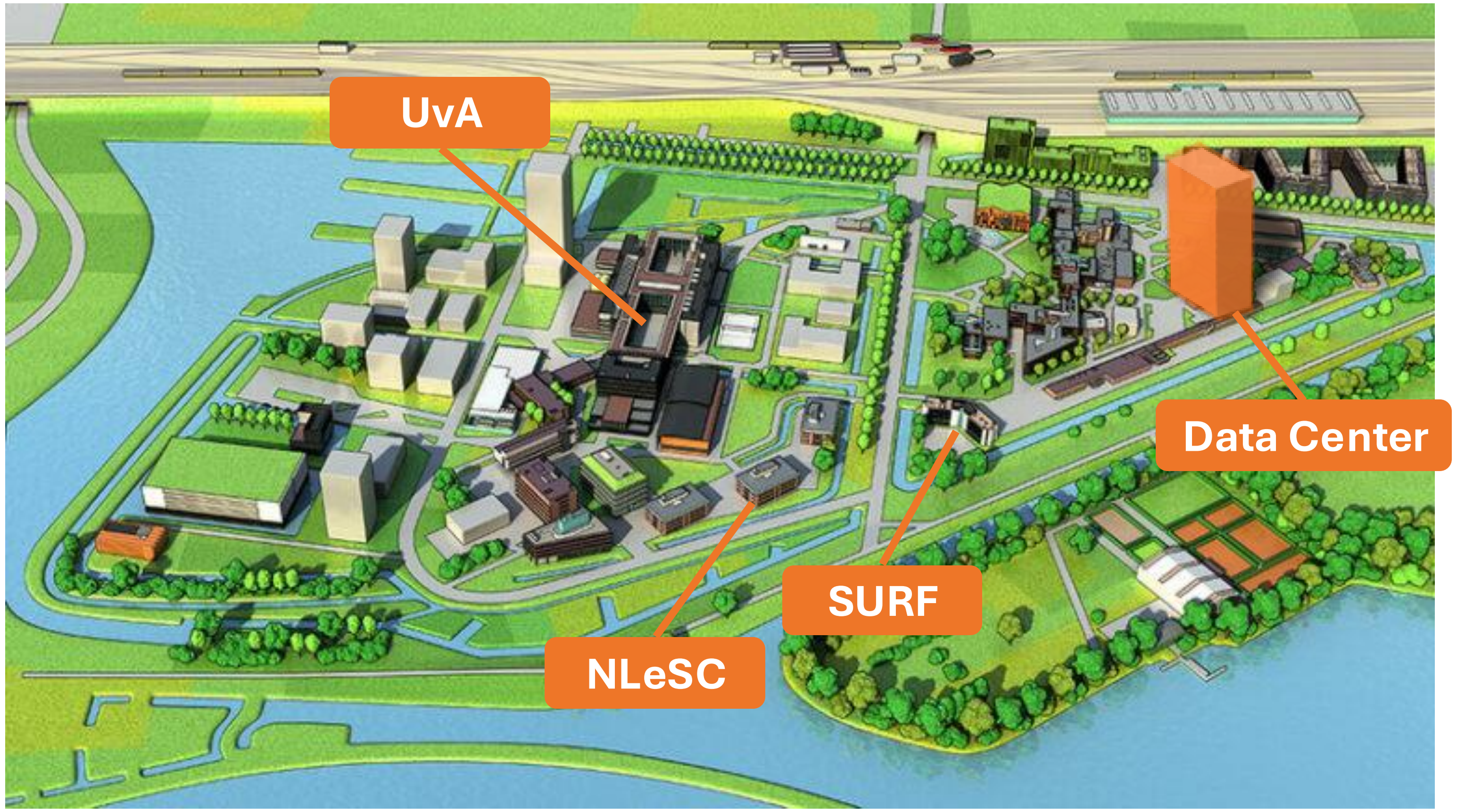
- an area of computer-based **computation**. It includes all computing work that requires a high computing capacity or storage capacity.
- the use of **parallel** processing for running advanced application programs efficiently, reliably and fast.
- the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve **large** problems in science, engineering, or business.
- the use of super computers and parallel processing techniques for solving **complex** computational problems.

High-Performance Computing (HPC) aggregates computing power and employs **parallel processing** to solve complex problems in **science, engineering, and business**. It enables **advanced simulations, large-scale data analysis, and high-speed computations** beyond the capabilities of standard desktops or workstations.

Snellius Supercomputer

SURF facilities

Location of SURF Amsterdam



source

Dutch national supercomputers

Year	Name	Machine	R_p (Gflop/s)	kW	Gflop/s/kW
1984		CDC Cyber 205-611 1-pipe	0.10	250	0.0004
1988		CDC Cyber 205-642 2-pipe	0.20	250	0.0008
1990		IBM ES 9000-720	0.83	100	0.0083
1990		Cray Y-MP4/464	1.33	200	0.0067
1993		Parsytec GCel 3/512	0.80	100	0.0080
1994	**Elsa**	Cray Y-MP C98/4256	4	300	0.0133
1995		IBM SP2/76	20	200	0.1010
1997	**Elsa**	Cray Y-MP C916/121024	12	500	0.0240
1998		IBM SP P2SC	48	350	0.1371
2000	**Teras**	SGI Origin 3800	1,024	300	3.4000
2004	**Aster**	SGI Altix 3700	2,176	200	11
2007	**Huygens**	IBM p575 Power5+	14,592	375	40
2008	**Huygens**	IBM p575 Power6	62,566	540	116
2009	**Huygens**	IBM p575 Power6	64,973	560	116
2013	**Cartesius**	Bull bullx DLC	250,000	260	962
			1,000,000	520	1923
			1,840,000	850	2168
			9,810,000	710	13,817
			14,794,000	1,500	9,862
			36,455,000	1,800	20,252

Year	Machine	R_p (Gflop/s)	kW	Gflop/s/kW
2000	PlayStation 2	6.2	0.05	124
2006	PlayStation 3	230.4	0.2	1,150
2012	Raspberry Pi 1	0.041	0.002	16.4
2019	MacBook Pro M1 Max	10,400	0.1	104,000
2021	iPhone 13	1,712	0.015	114,133
2023	High-end gaming laptop	110,000	0.3	366,666

SARA: https://top500.org/site/48715/
SARA/IC3A: https://top500.org/site/48716/
SURFsara: https://top500.org/site/50498/
SURF: https://top500.org/site/50900/

SURF

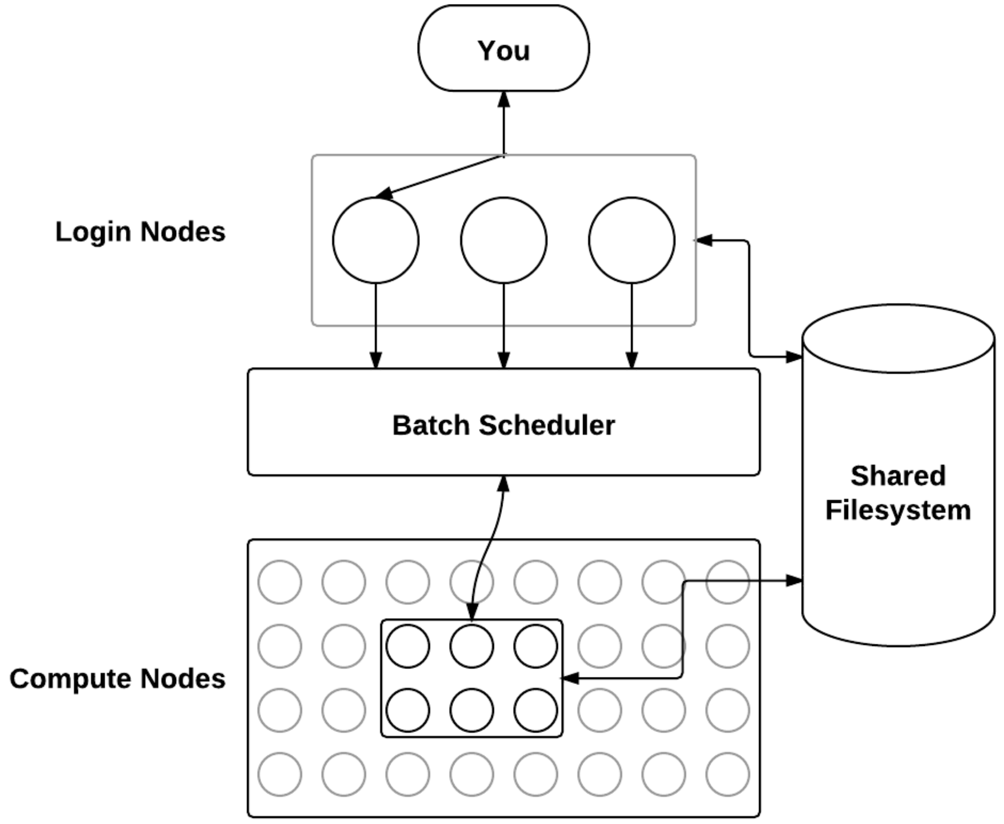
Schematic overview of a supercomputer

Users interact with the system through **login nodes**, which serve as the entry point.

- **Login nodes** provide an environment for **code development, compilation, and job submission**.

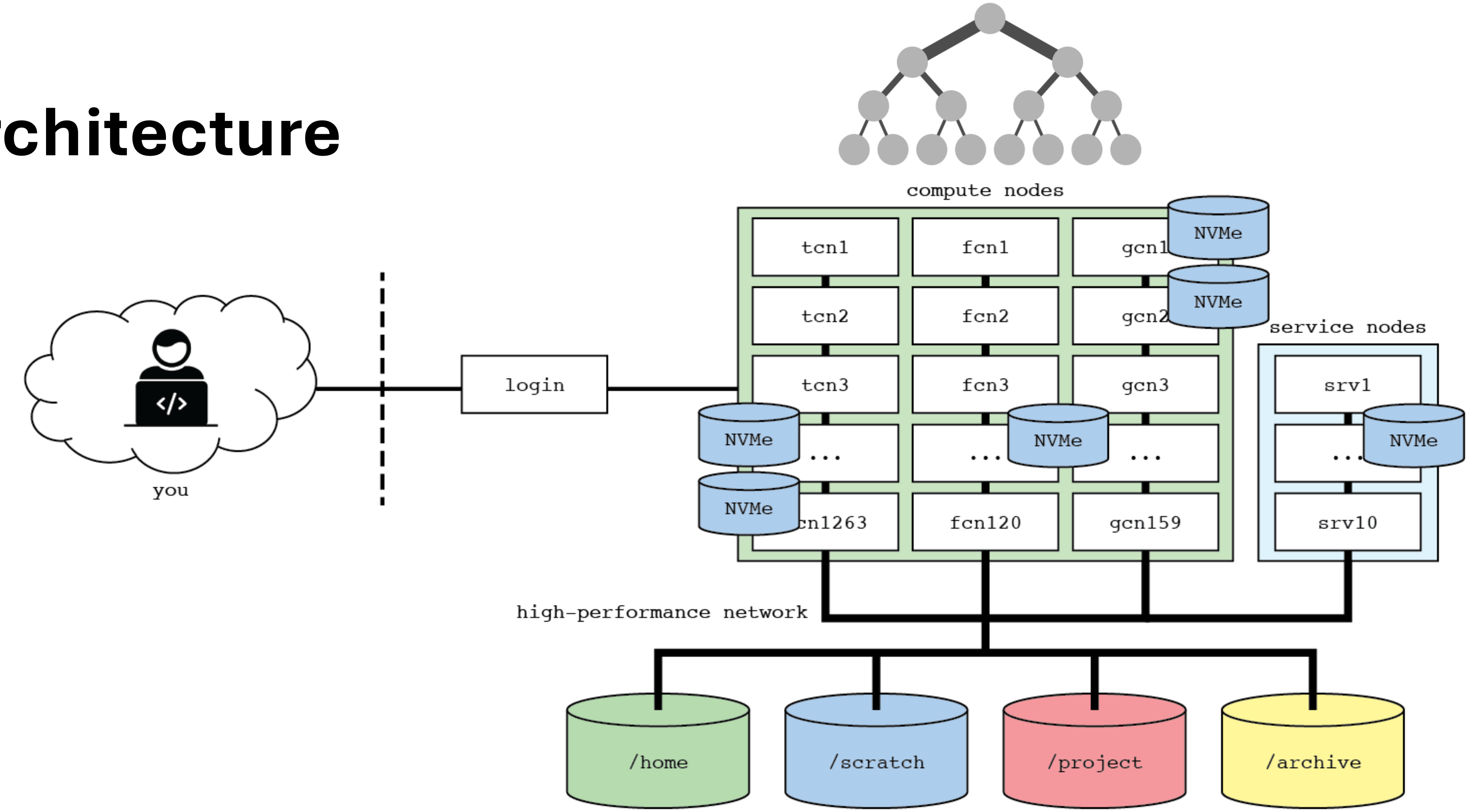
Login nodes are shared by all online users in the system and hence **heavy tasks must be avoided**.

- **Batch scheduler** manages job distribution, queuing, and execution on **compute nodes**.
- **Compute nodes** are dedicated for running large-scale computations, allocated dynamically by the scheduler.
- **Shared filesystem** ensures **data accessibility** between login nodes and compute nodes, enabling seamless workflow execution.



SURF

Snellius architecture



- 238,816 cores + 640 GPUs: **36.5 Pflop/s of peak performance** and **1,421** TB memory.
- Low-latency interconnection network: fat-tree **InfiniBand HDR100, HDR200, and NDR200**.
- **720 TB for home** directories and **12.4 PB for scratch and project** spaces (GPFS).
- Specific policy for software installation and maintenance, with three simultaneous available software stacks.

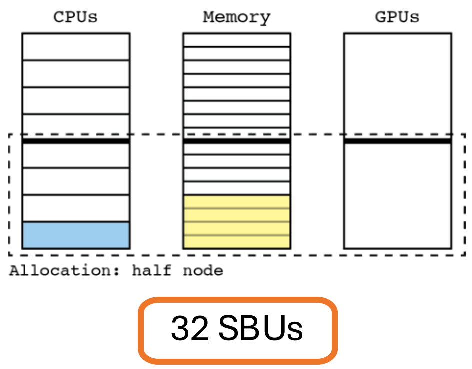
Snellius system details

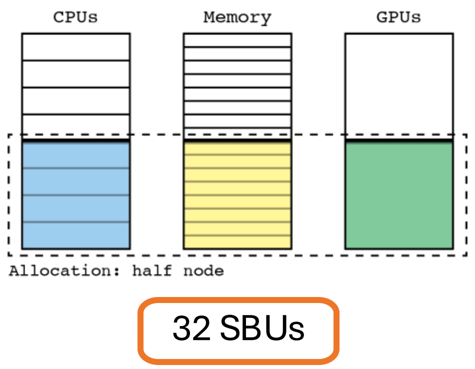
Partition	#Nodes	Memory per node	#Cores	CPU model	#GPUs	GPU model	SBUs/hour	Smallest allocation
rome	525	256 GB	128	AMD Rome 7H12 2.6 GHz			128	1/8
rome/fat	72	1,024 GB	128	AMD Rome 7H12 2.6 GHz			192	1/8
himem/4tb	2	4,096 GB	128	AMD Rome 7H12 2.6 GHz			256	1/8
himem/8tb	2	8,192 GB	128	AMD Rome 7H12 2.6 GHz			384	1/8
genoa	738	384 GB	192	AMD Genoa 9654 2.4 GHz			192	1/12
genoa/fat	48	1,536 GB	192	AMD Genoa 9654 2.4 GHz			288	1/12
gpu/a100	72	512 GB	72	Intel Xeon 8360Y 2.5 GHz	4	NVIDIA A100 40 GB	512	1/8
gpu/h100	88	768 GB	64	AMD EPYC 9334 2.7 GHz	4	NVIDIA H100 (SXM5) 94 GB	768	1/4
service	10	256 GB	16	AMD EPYC 7F32 3.2 GHz			32	1/32
login	3	**Login nodes MUST NOT be used for heavy tasks.** Their hardware configuration may differ.						

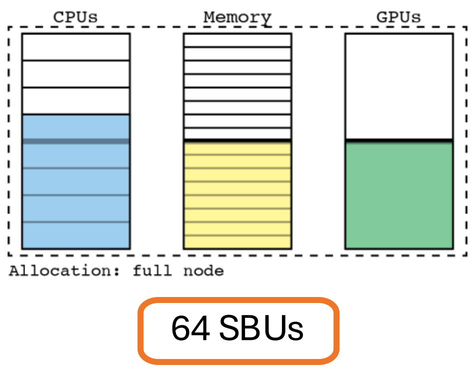
Snellius system details: https://servicedesk.surf.nl/wiki/display/WIKI/System+details
Obtaining an account: https://servicedesk.surf.nl/wiki/display/WIKI/Obtaining+an+account+on+Snellius
Software stack and building: https://servicedesk.surf.nl/wiki/display/WIKI/Software

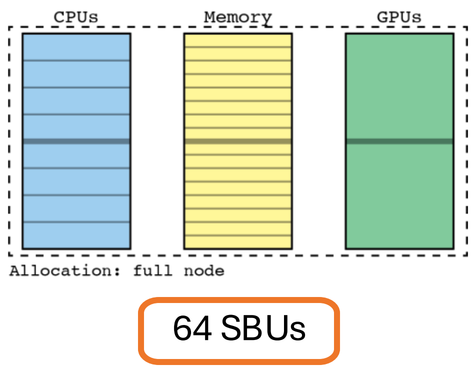
How does accounting work?

Partition	#Nodes	Memory per node	#Cores	CPU model	#GPUs	GPU model	SBUs/hour	Smallest allocation
Example	1	16 GB	8	Processor	2	Accelerator	64	1/2





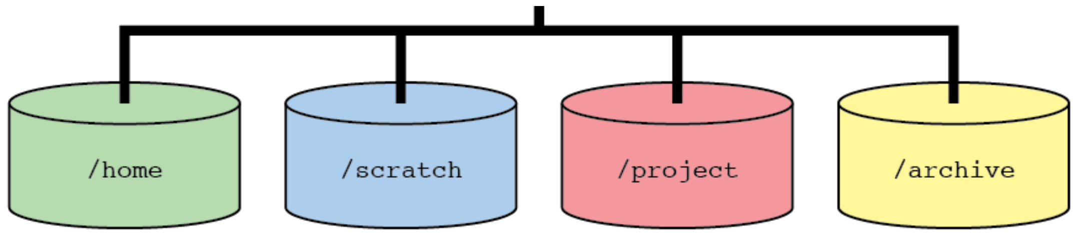




Each node provides a range of resources, including CPU cores, memory, accelerators (e.g., GPUs), and local disk storage. For simplicity, documentation typically refers to allocations in terms of the number of CPU cores. However, in practice, **resource allocation is determined by the most restrictive resource requested**.

- If a user requests all CPU cores on a node but no GPUs, the node is still considered **fully allocated**. Even though the GPUs remain idle, other users cannot access them because all CPUs are occupied.
- Conversely, if a user requests only one CPU core, the system **rounds up the allocation to the smallest permitted unit**. This ensures that the necessary CPU count required for node engagement is available.

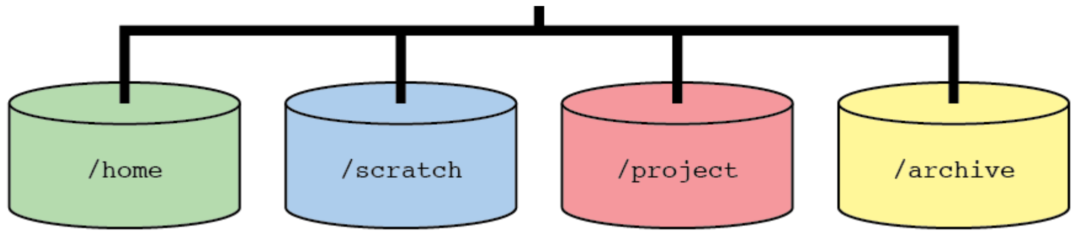
File systems on Snellius



`/home/user`

- User home directory, currently **200 GB per user.**
- **Based on fast, parallel file system (GPFS).**
- **Backed up regularly**, ensuring data safety.
- Intended for storing **important files**, such as:
  - **Source codes**
  - **Scripts**
  - **Small to medium-sized input and output data**
- **Not intended for large datasets or high-throughput I/O workloads.**
  - Keep essential and frequently used files here.
  - Regularly clean up unused data to stay within quota.
  - Move large files to project storage or scratch space for performance efficiency.

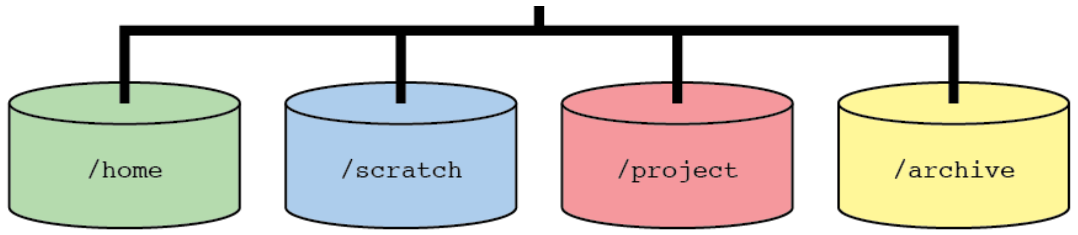
File systems on Snellius



`/scratch-local` and `/scratch-shared`

- User working space, **currently 8 TB guaranteed.**
- **Based on fast, parallel file system (GPFS).**
- **Not backed up.**
- **Temporary storage**—data is removed after 6 days (local), 14 days (shared).
- Two different implementations:
  - **Scratch local.**
    - Dedicated to **individual jobs and compute nodes** on Snellius.
    - Uniquely configured for each compute node upon the initiation of a job.
    - Accessed via the environment variable `$TMPDIR`.
  - **Scratch shared**: shared, constant directory visible from both login and compute nodes.

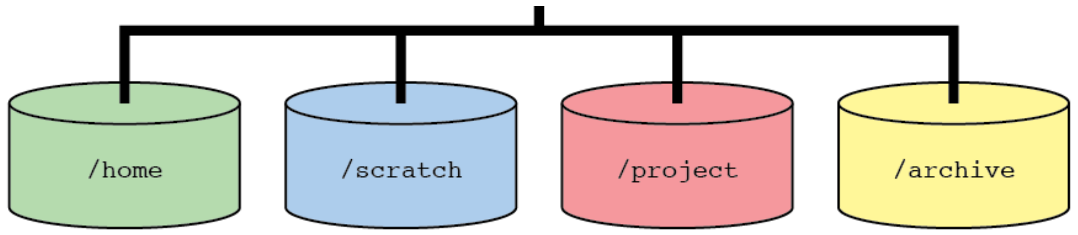
File systems on Snellius



`/scratch-node`

- Truly node-local working space of variable size (on demand) with **max 6.4 TB per node**.
- **Based on local NVMe SSD.**
- **Not backed up.**
- **Temporary storage**—data is deleted immediately after the job finishes.
- Requested with flag `--constraint=scratch-node`, then accessed via the environment variable `$TMPDIR` (replaces Scratch local).

File systems on Snellius



`/project`

- Project working space, given on request.
- **Based on fast, parallel file system (GPFS).**
- **Not backed up.**
- **Permanent storage**—data is removed at the end of the associated project.
  - Yet, not intended for permanent storage of unused files. Think about **self-managed scratch**.

`/archive`

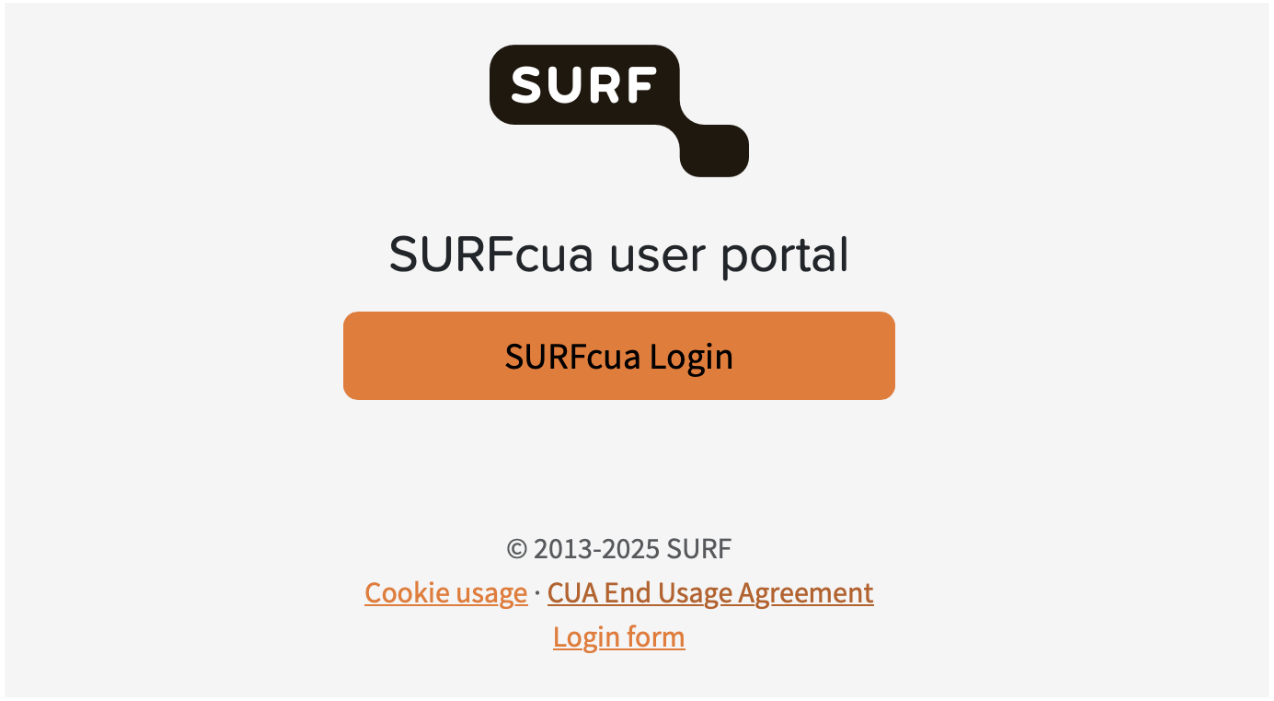
- Long-term archiving space, given on request with **virtually unlimited quota**.
- Based on **tape storage** and connected to the tape robot.
- **Backed up**.
- Intended for long-term storage of files in compressed format.
  - Slow—especially to retrieve "old" data—and not available on compute nodes.

Connecting to Snellius

Before first access to the system, login into the SURF user portal, CUA!

- You should have received an email from SURFcua (no-reply@surf.nl) with the subject "**SURFcua new login <username> created**."

https://portal.cua.surf.nl


SURFcua user portal

© 2013-2025 SURF

Cookie usage · CUA End Usage Agreement

Login form

Connecting to Snellius via Terminal

Primary way of accessing Snellius requires a **terminal-based connection using SSH**, allowing users to interact with the system, manage files, and submit jobs remotely from their local machine.

A **terminal** is a text-based interface that allows users to interact with a system by entering commands.

Windows

- MobaXterm (portable recommended):
  - https://mobaxterm.mobatek.net
- PuTTY.
  - https://www.putty.org

macOS

- Native terminal. XQuartz needed for graphical output.
  - http://www.xquartz.org

Linux

- Native terminal.

SURF wiki guide: https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/30660216/Connecting+to+the+system

SURF

Connecting to Snellius via Terminal

When you log in with `ssh`, you access the login nodes.

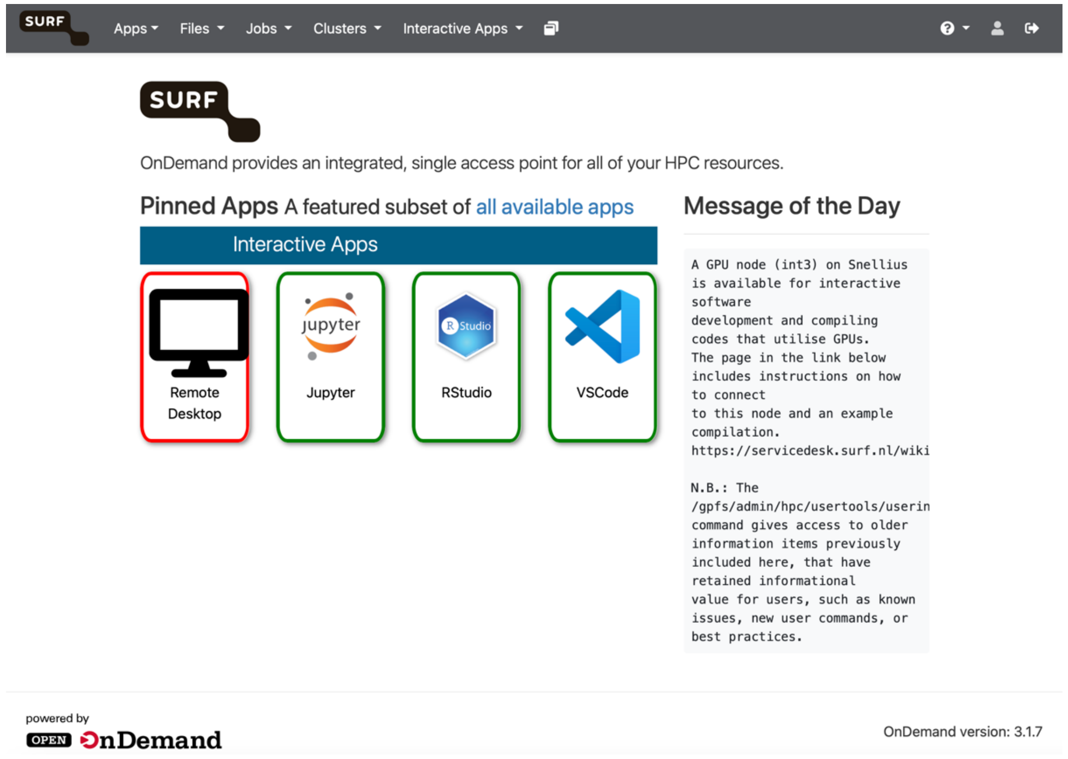
- Type: `ssh <username>@snellius.surf.nl`.
- Alternative for untrusted connections: `ssh <username>@doornode.hpcv.surf.nl`.

```
user@local:~$ ssh scur0000@snellius.surf.nl
Password:
scur0000@int4:~$
```

With `scp` you can transfer files to and from your local machine.

```
user@local:~$ ls
local-file.txt
user@local:~$ scp ./local-file.txt scur0000@snellius.surf.nl:~/
user@local:~$ scp scur0000@snellius.surf.nl:~/snellius-file.txt ./
user@local:~$ ls
snellius-file.txt local-file.txt
user@local:~$ ssh scur0000@snellius.surf.nl
Password:
scur0000@int2:~$ ls
snellius-file.txt local-file.txt
```

Connecting to Snellius via OpenOnDemand



OnDemand provides an integrated, single access point for all of your HPC resources.

Pinned Apps A featured subset of all available apps

Message of the Day

```
A GPU node (int3) on Snellius
is available for interactive
software
development and compiling
codes that utilise GPUs.
The page in the link below
includes instructions on how
to connect
to this node and an example
compilation.
https://servicedesk.surf.nl/wiki

N.B.: The
/gpfs/admin/hpc/usertools/userin
command gives access to older
information items previously
included here, that have
retained informational
value for users, such as known
issues, new user commands, or
best practices.
```

OnDemand version: 3.1.7

Besides accessing Snellius via a terminal, the system can also be used via a web browser using the **OpenOnDemand** demand service allowing users to:

- manage files,
- work in a shell without using a terminal client on their remote system,
- launch jobs based on predefined templates,
- run interactive web services such as RStudio and Jupyter Notebooks,
- run applications with graphical user interfaces.

https://ondemand.snellius.surf.nl

LUMI

LUMI is the **3rd fastest supercomputer in Europe** and the eight fastest in the world. It is a pre-exascale system based on **AMD technology**. It contains 2048 CPU-based compute nodes and 2978 GPU nodes, interconnected with a fast Slingshot-11 network.

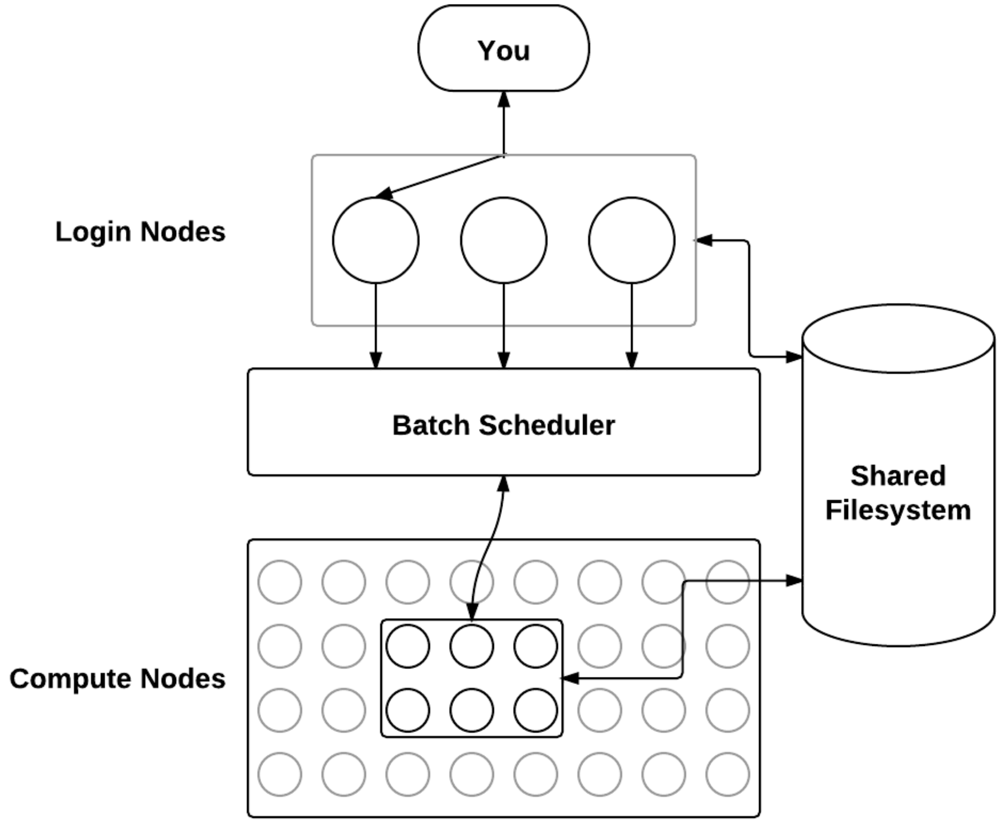
- LUMI offers a **large GPU partition**.
- LUMI uses **100 percent renewable electricity**. Waste heat produced by LUMI is used to heat hundreds of households in the city of Kajaani, Finland, where LUMI is located.
- Half of LUMI's computational resources belong to the LUMI consortium countries, **including the Netherlands**.
- Access to LUMI for pilot and regular projects can be obtained **directly via SURF and NWO**.

Job submission

Batch scheduler for job submission

Batch schedulers distribute work to **compute nodes.**

- Workflow:
  1. **You** upload your data from your computer to the cluster system.
  2. **You** create a job script with the work steps.
  3. **You** submit the job script to the scheduler.
  4. **The scheduler** looks for available computers to run your work.
  5. When a batch node with the requirements you specified becomes available, your work runs.
  6. When the job is finished, **you** download the results to your computer.
- Batch scheduler on Snellius is Slurm:
  - http://slurm.schedmd.com



Slurm basic commands

Command	Parameters	Description
`sbatch`	`<jobscript>`	Submits a job to the scheduler
`squeue`		Inspects the job queue visible to user
	`-u <username>`	Filters the job queue to inspect jobs of user `<username>`
	`-j <jobid>`	Filters the job queue to inspect job <job-id>
`scancel`	`<jobid>`	Cancels job `<job_id>` before it finishes
`sinfo`		Displays information about the state of compute nodes
`sacct`		Displays accounting data for jobs and job steps visible to user
`scontrol`	`scontrol show job <job_id>`	Displays job information, including estimated start time

Sbatch: https://slurm.schedmd.com/sbatch.html
Squeue: https://slurm.schedmd.com/squeue.html
Scancel: https://slurm.schedmd.com/scancel.html
Sinfo: https://slurm.schedmd.com/sinfo.html
Sacct: https://slurm.schedmd.com/sacct.html
Scontrol: https://slurm.schedmd.com/scontrol.html

Running jobs, first example

```
#!/bin/bash
#SBATCH --job-name="job-test"
#SBATCH --nodes=1
#SBATCH --ntasks=10
#SBATCH --time=00:01:00
#SBATCH --partition=thin

echo "Who am I?"
whoami
echo

echo "Where ?"
srun hostname
echo

sleep 120
date

echo "DONE"
```

- Create a text file with the lines in the example; name the file “job.sh.”
- Submit this job with **`sbatch job.sh`** and look the status with **`squeue –u <username>`**.
- Use **`scontrol show job <jobid>`** to find out when your job will run.
- Look at your home directory to see what has happened; look for new files (use **`ls`** command).
- Which files have been created? Check their content.
  - **Can you solve the error?**
- Try to set email notifications: add the following two lines in your script:
  - `#SBATCH --mail-type=BEGIN,END`
  - `#SBATCH --mail-user=<your_email_address>`

Running jobs, second example

```
#!/bin/bash
#SBATCH --job-name="mpi-test"
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=00:02:00
#SBATCH --partition=thin

echo "Running an MPI job..."
srun hostname
echo "DONE"
```

- Create a text file with exactly the lines in the example; name the file “mpi-job.sh.”
- Submit the job with **`sbatch mpi_job.sh`**.
- Check the output files.
- Compare the output when changing **`--ntasks=4 to --ntasks=8`**.
- What happens if you remove **`srun`**?
- What happens if you use **`mpirun`** instead?

Running jobs, third example

```
#!/bin/bash
#SBATCH --job-name=”first-job"
#SBATCH --time=00:01:00
#SBATCH --partition=thin

echo "First job running..."
sleep 30
echo "First job finished."
```

```
#!/bin/bash
#SBATCH --job-name=”second-job"
#SBATCH --time=00:01:00
#SBATCH --partition=thin

echo "Second job started after first
job!"
```

- Create two text files the lines in the examples; name the file “first-job.sh” and “second-job.sh.”
- Submit the first job with **`sbatch first-job.sh`**.
- Get its job ID (**`squeue -u <username>`**).
- Submit the second job so that it only starts **after the first job completes**:
  - **`sbatch --dependency=afterok:<jobid> second-job.sh`**
- Check the queue. What do you observe?

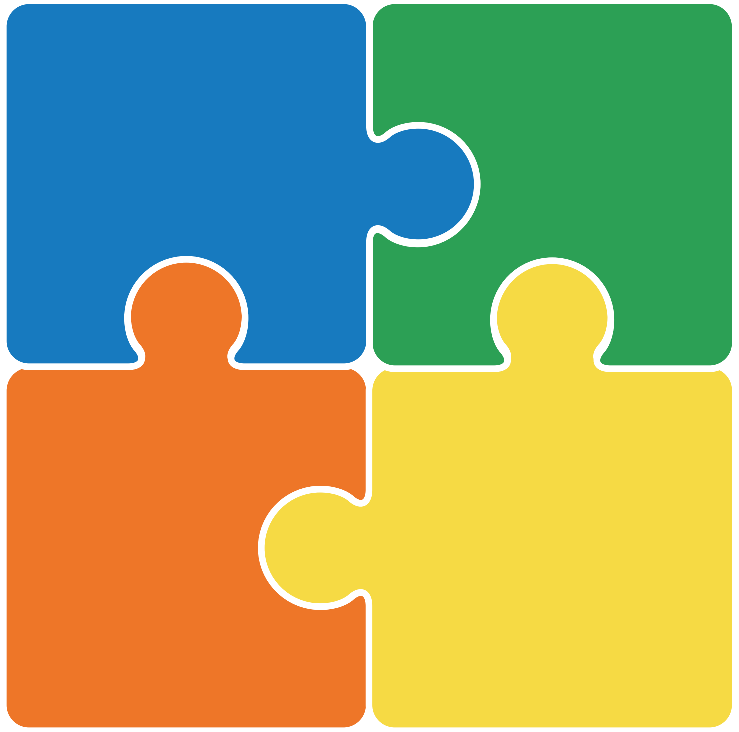
Running jobs, best practices

Give the scheduler an accurate wall-time estimate

A **realistic job time** helps optimize scheduling, reduces queue times, and prevents unnecessary resource reservations that could delay other jobs.

Don’t use `/home` as work dir

Your home directory has limited performance for high-throughput tasks—use `/scratch` storage for faster I/O.



Don’t preload software modules

Include software module loading in your job script to ensure **consistency and reproducibility** across runs.

Leverage parallelism

Run parallel versions of your programs and use `srun` to efficiently launch multi-process applications within Slurm.

Anatomy of a job script

```
#!/bin/bash
```

```
#SBATCH --job-name="firsttest"
#SBATCH --nodes=1
#SBATCH --ntasks=10
#SBATCH --time=00:01:00
#SBATCH --partition=thin
```

```
module load 2024
module load foss/2024a
```

```
cp -r <my_folder> $TMPDIR
cd $TMPDIR
```

```
srun a.out
```

```
cp -r $TMDPDIR/* ~/results
```

Job scripts consist of:

- the "shebang" line: `#!/bin/bash`,
- scheduler directives,
- command(s) that load software modules and set the environment,
- command(s) to prepare the input,
- command(s) that run your main task(s),
- command(s) to save your output.

Slurm useful directives

Option	Description
`--job-name=name`	Informative name for the job, aiding in identification.
`--nodes=#`	Specifies the number of nodes to utilize.
`--ntasks=#`	Specifies the number of tasks to execute (distributed parallelism).
`--ntasks-per-node=#`	Number of tasks assigned to each physical node.
`--cpus-per-task=#`	Specifies the number of cores allocated per task (shared memory parallelism).
`--time=[[DD-]HH:]MM:SS`	Maximum allowable runtime for the job.
`--mem-per-cpu=#`	RAM required per CPU (in MB).
`--mem=#`	RAM required per node (in MB).
`--output=name`	File for standard output of the job.
`--error=name`	File for error output of the job.
`--chdir=[dirname]`	Working directory for the job.

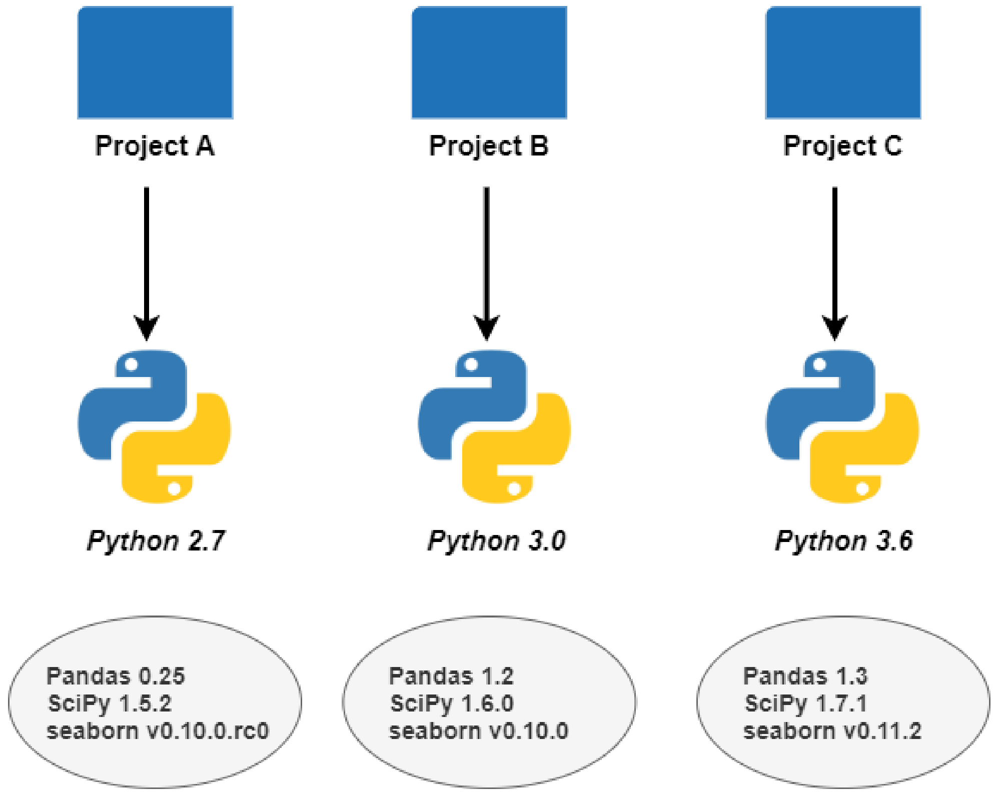
Sbatch: https://slurm.schedmd.com/sbatch.html

Environment modules

Software on HPC systems

Challenges of managing software on HPC

- **Diversity**: HPC systems serve researchers from different fields (e.g., physics, AI, bioinformatics).
- **Dependencies**: users need different versions of software, and different applications require different versions of the same library.
- **Performance**: precompiled binaries vs. optimized builds for HPC architecture (e.g., AVX512, CUDA).
- **Permission**: Root access is restricted; users must manage software within the given environment.

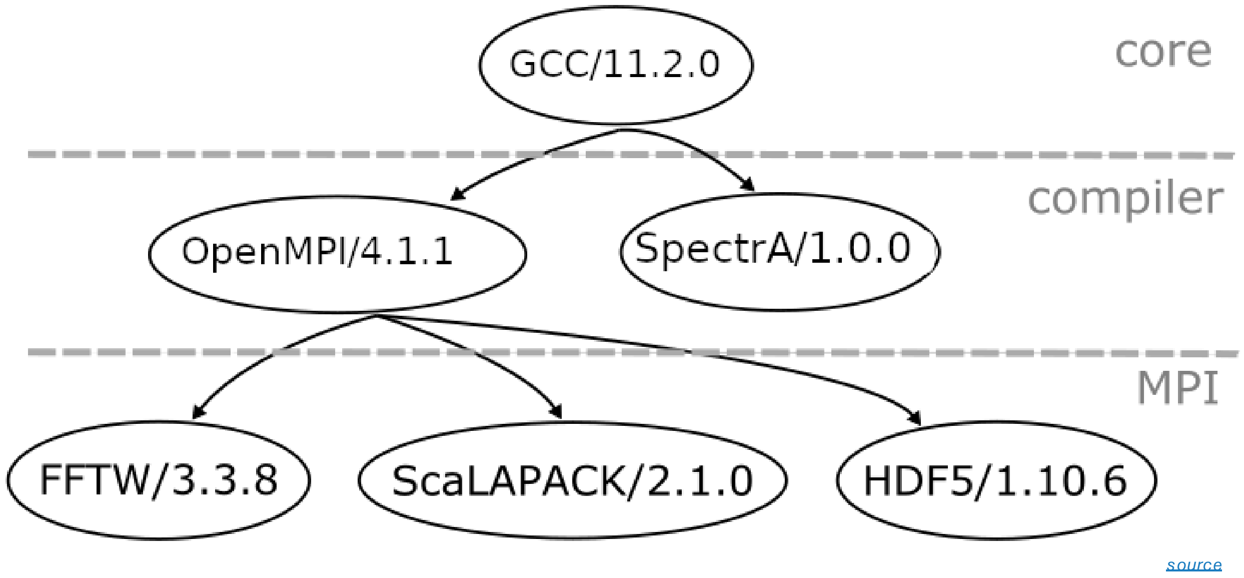


source

Common approaches include **compiling software from source**, **virtual environments** (e.g., Conda), or **containers** (e.g., Apptainer).

- Individual solutions may work for users but are **ineffective for shared HPC environments**.
- **Supporting users with local installations is not scalable** and complicates system-wide consistency, performance tuning, and long-term reproducibility.

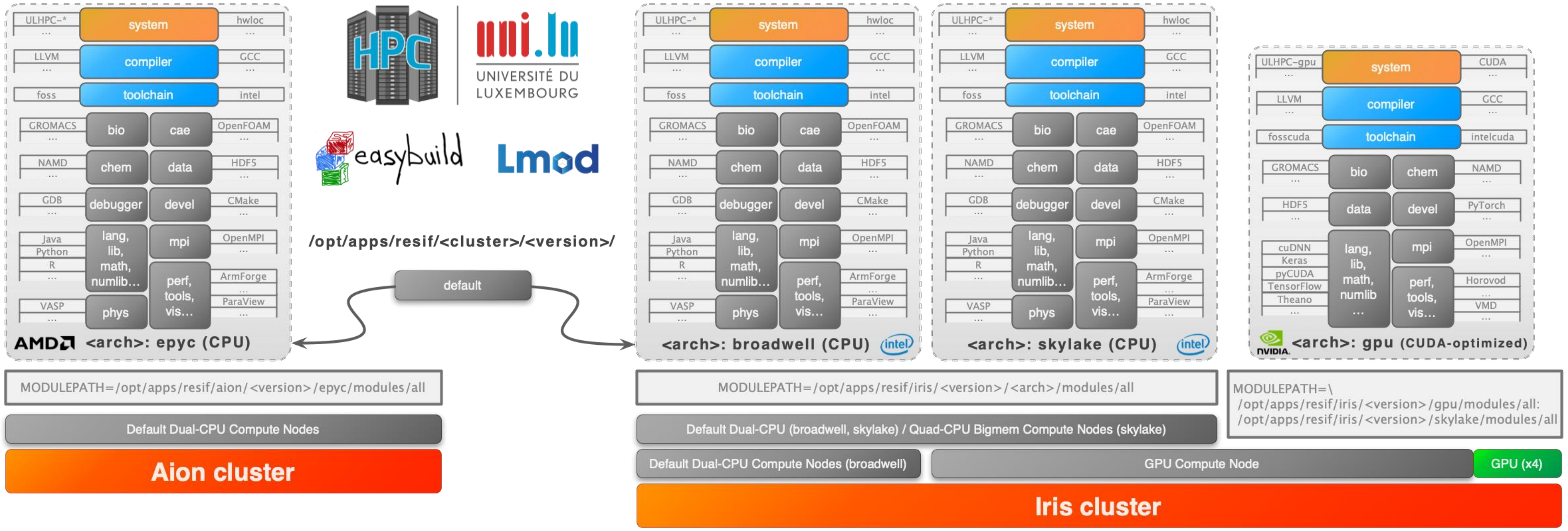
Environment modules



Environment modules allow system administrators to...

- Provide **a clean and modular environment** for users.
- Manage **multiple software versions** without conflicts.
- Optimize software performance for the **HPC architecture**.
- Ensure **system-wide consistency** across applications.
- Simplify **maintenance and updates** without affecting users.
- Control **software access** based on user permissions or project needs.
- **Centralize issue resolution** ensuring all users benefit without needing individual workarounds.

Environment modules



system
compiler
toolchain
ULHPC-*
hwloc
LLVM
GCC
foss
intel
bio
cae
chem
data
debugger
devel
lang, lib, math, numlib…
mpi
perf, tools, vis…
phys
GROMACS
OpenFOAM
NAMD
HDF5
GDB
CMake
Java
Python
R
OpenMPI
ArmForge
ParaView
VASP
AMD <arch>: epyc (CPU)
HPC
uni.lu UNIVERSITÉ DU LUXEMBOURG
easybuild
Lmod
/opt/apps/resif/<cluster>/<version>/
default
<arch>: broadwell (CPU)
<arch>: skylake (CPU)
intel
ULHPC-gpu
CUDA
fosscuda
intelcuda
PyTorch
cuDNN
Keras
pyCUDA
TensorFlow
Theano
Horovod
VMD
NVIDIA <arch>: gpu (CUDA-optimized)
MODULEPATH=/opt/apps/resif/aion/<version>/epyc/modules/all
MODULEPATH=/opt/apps/resif/iris/<version>/<arch>/modules/all
MODULEPATH=\
/opt/apps/resif/iris/<version>/gpu/modules/all:
/opt/apps/resif/iris/<version>/skylake/modules/all
Default Dual-CPU Compute Nodes
Aion cluster
Default Dual-CPU (broadwell, skylake) / Quad-CPU Bigmem Compute Nodes (skylake)
Default Dual-CPU Compute Nodes (broadwell)
GPU Compute Node
GPU (x4)
Iris cluster

Software on Snellius

SURF provides a managed and optimized software stack. We recommend using the provided software as much as possible.

- SURF does not guarantee that locally installed Conda software function properly and efficiently.
- SURF does **not offer support for locally installed Conda software**.
- We strongly advise **not to mix software from the module system and from Conda environments**, as both systems are incompatible to significant degrees.

Module	Description
2024	**Production** stack and is fully supported
2023	**Previous production** stack, it only has limited support
2022	**Deprecated** and provided as-is without support
EESSI	Collaborative effort by multiple European partners in HPC community to build a **common stack of scientific software installations for HPC** systems and beyond.

Lmod basic commands

Command	Description
`module help`	Provides information about the command options.
`module avail [<modulename>]`	Displays modules directly available in the system.
`module spider [<modulename>]`	Reports all the modules that can be loaded. In a hierarchical system, module avail only reports modules that can be directly loaded directly whereas module spider returns all the modules that are possible.
`module list`	Shows the modules currently loaded in your environment.
`module load <modulename>`	Loads `<modulename>` into your environment.
`module unload <modulename>`	Removes `<modulename>` from your environment.
`module purge`	Removes all modules from your environment.
`module whatis <modulename>`	Shows information about `<modulename>`.

Lmod: https://lmod.readthedocs.io/en/latest/010_user.html

SURF

Hands-on

SURF