

Fundamentals of high-performance computing

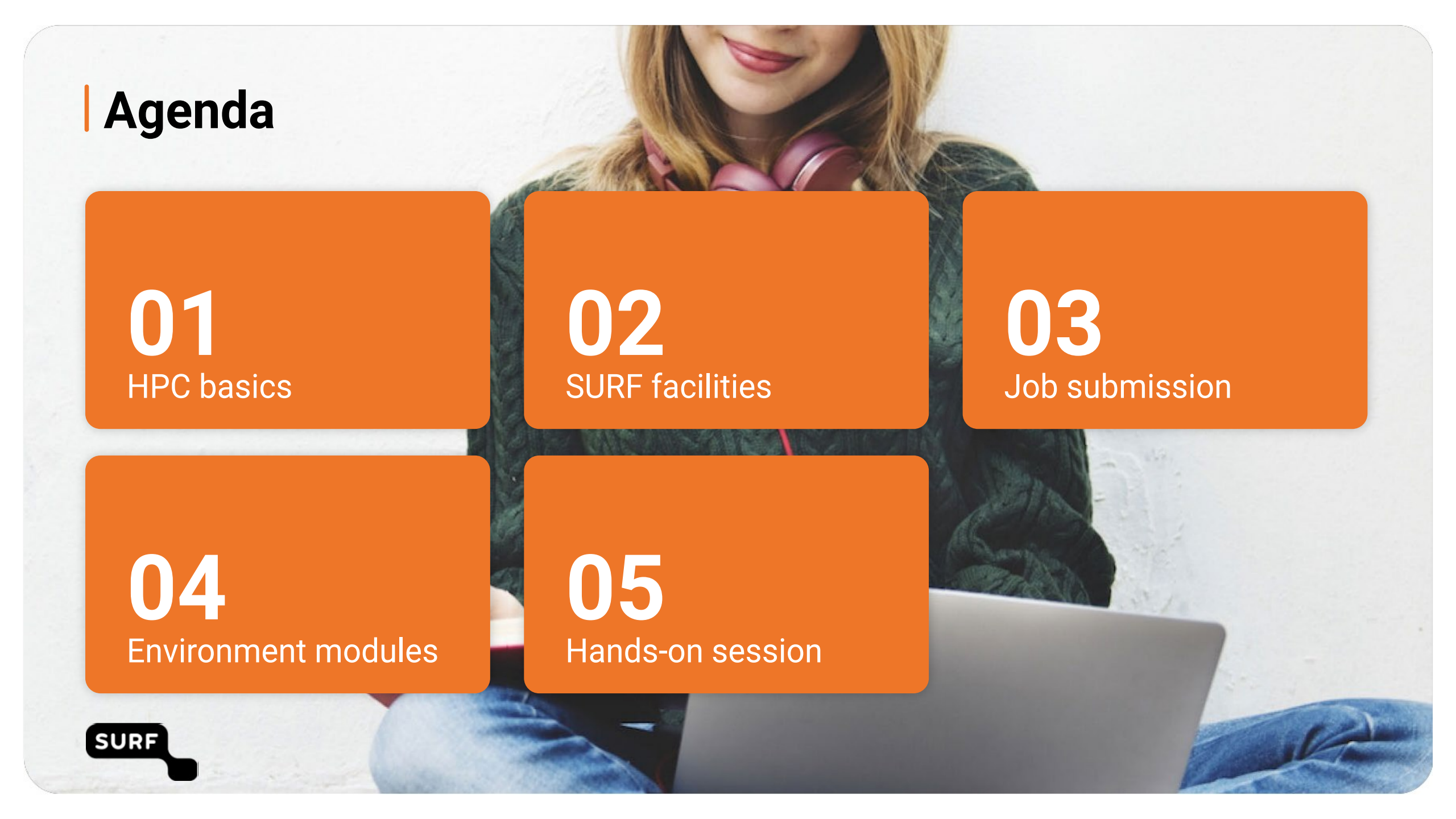
General online course

6 March, 2026

Adapted and updated by Rohit Goswami

SURF





| Agenda

01

HPC basics

02

SURF facilities

03

Job submission

04

Environment modules

05

Hands-on session

SURF

| HPC basics

| Science and HPC



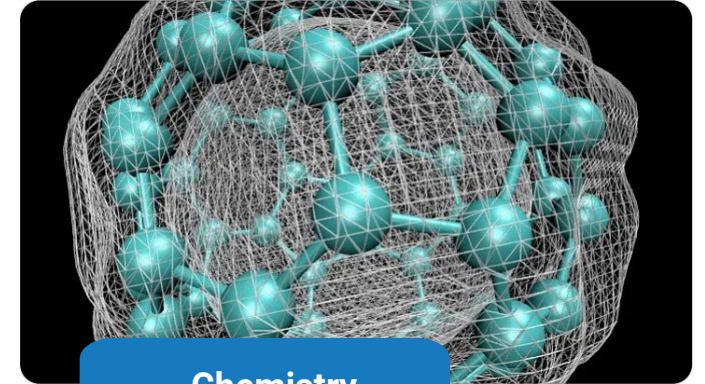
Astrophysics

[source](#)



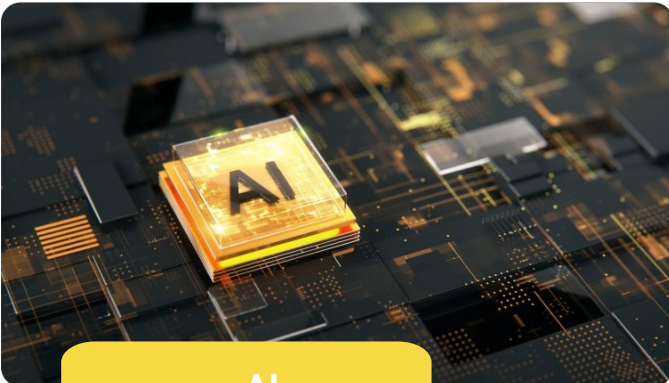
CFD

[source](#)



Chemistry

[source](#)



AI

[source](#)



Climate

[source](#)



Human behavior

[source](#)

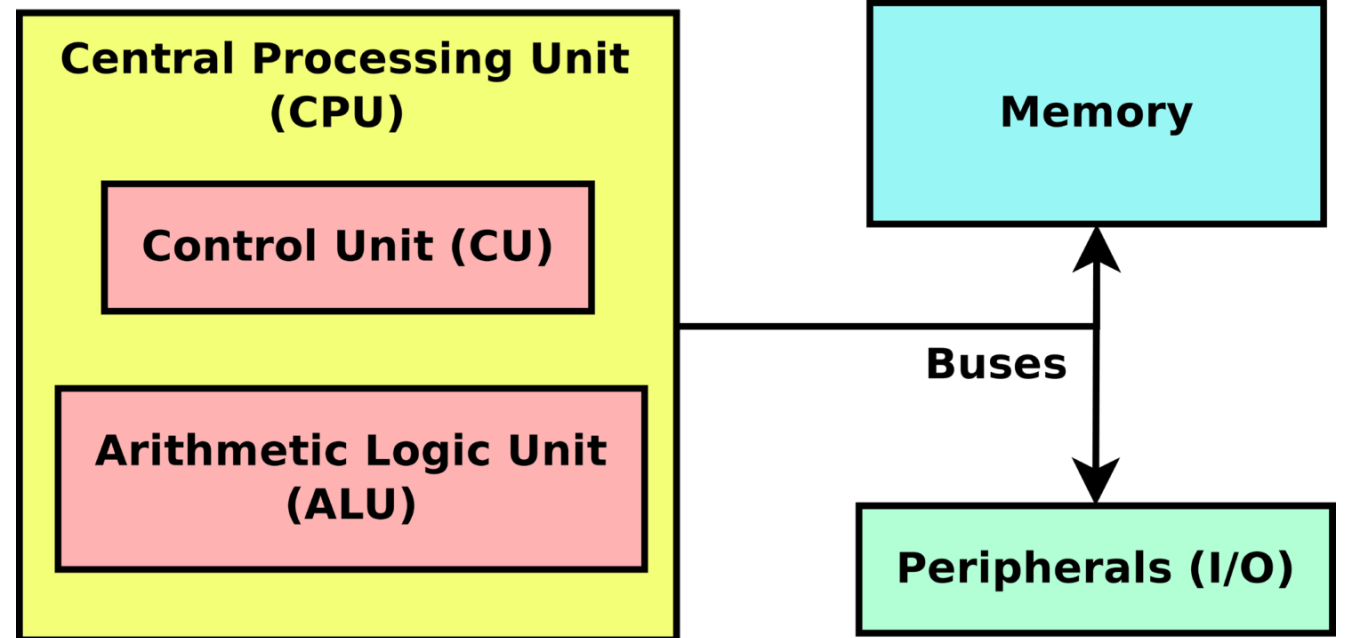
| High-performance computing is

- 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 to solve **large** problems in science, engineering, or business, at much higher performance than a typical desktop computer or workstation.
- the use of super computers and parallel processing techniques for solving **complex** computational problems.



| A computer is

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



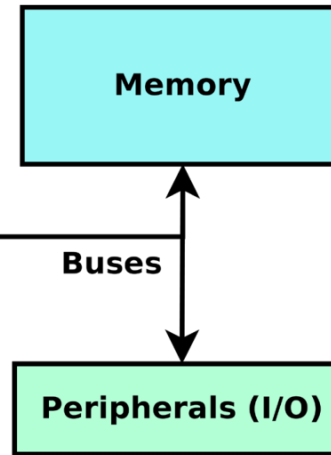
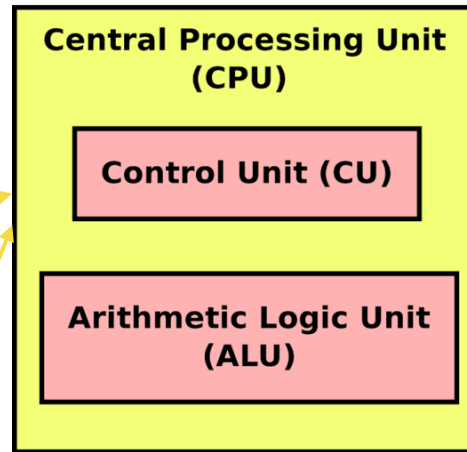
And Boolean logic, zeros and ones!

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.

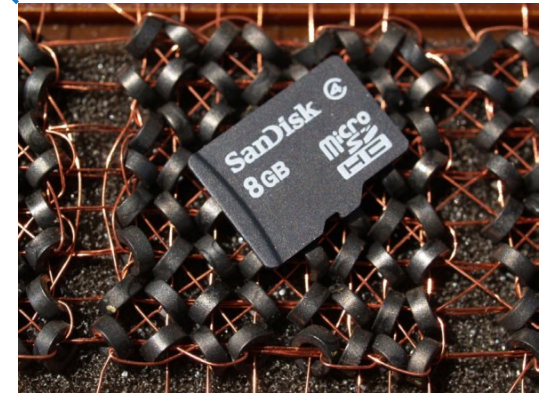
A computer is



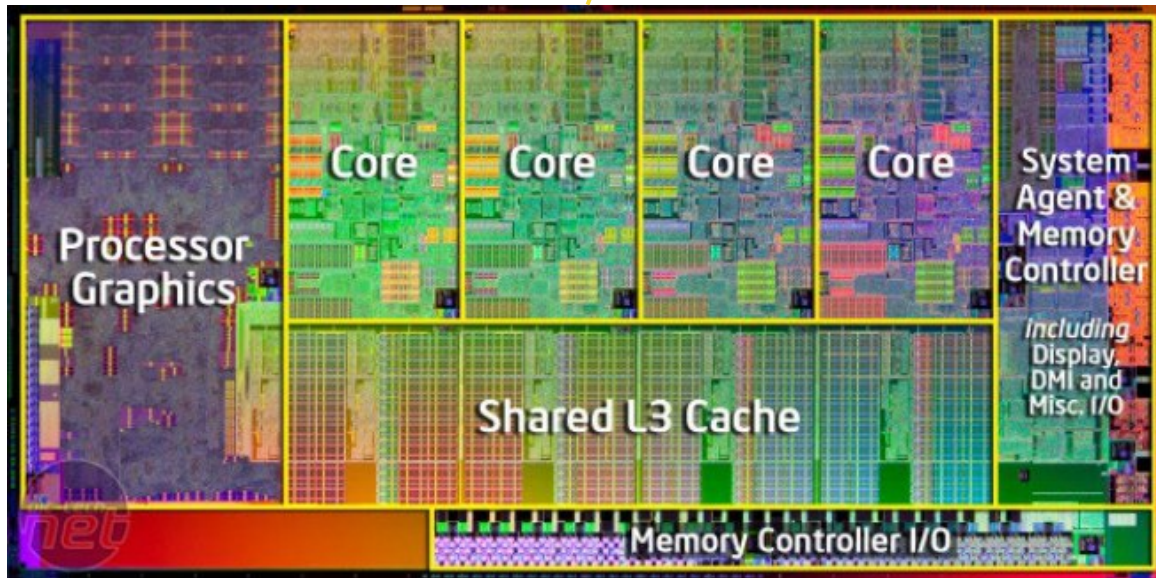
[source](#)



[source](#)



[source](#)



[source](#)



[source](#)



[source](#)

| A larger computer could be

One big node

Many CPUs, cores, or accelerators in one machine.
One operating system, one memory.

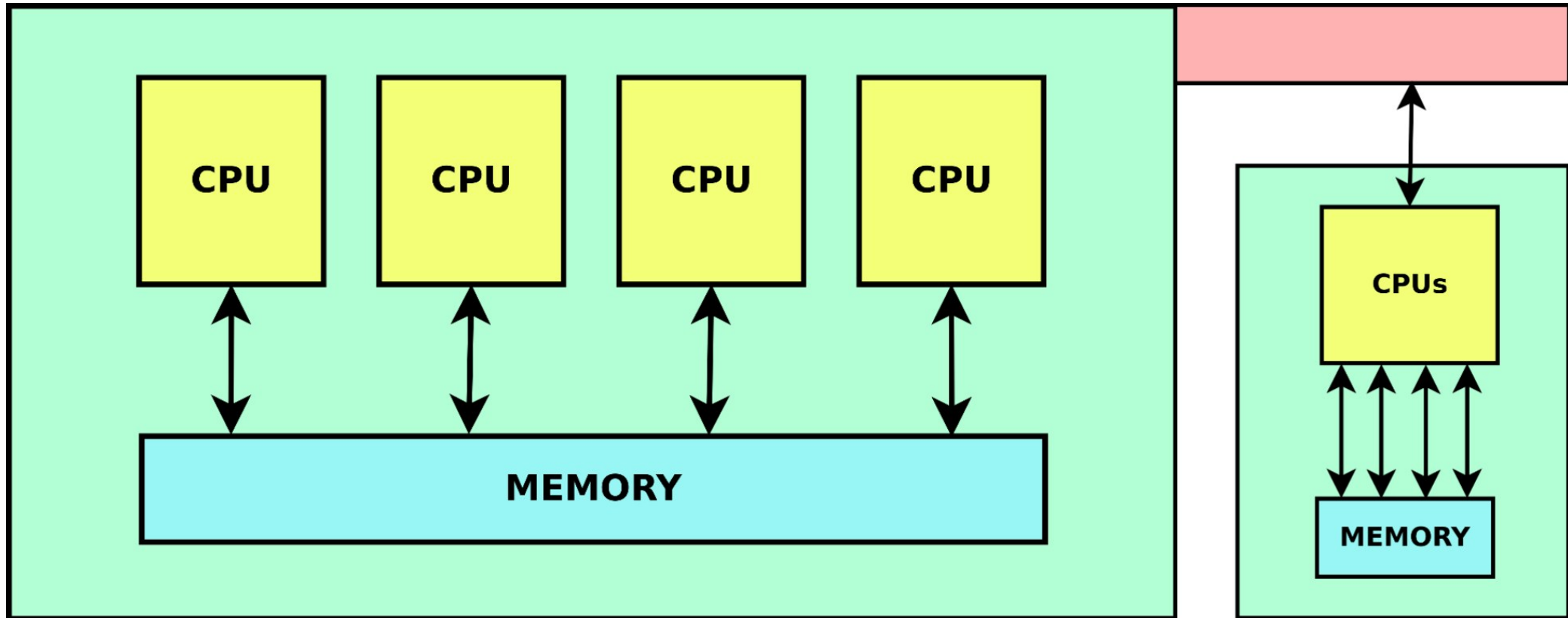
Several nodes, shared memory

Separate operating systems.
Software makes the memory look shared.

Many nodes

Each node has its own system and memory.
The nodes exchange messages over a network.

| 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 to solve **large** problems in science, engineering, or business, at much higher performance than a typical desktop computer or workstation.
- the use of super computers and parallel processing techniques for solving **complex** computational problems.

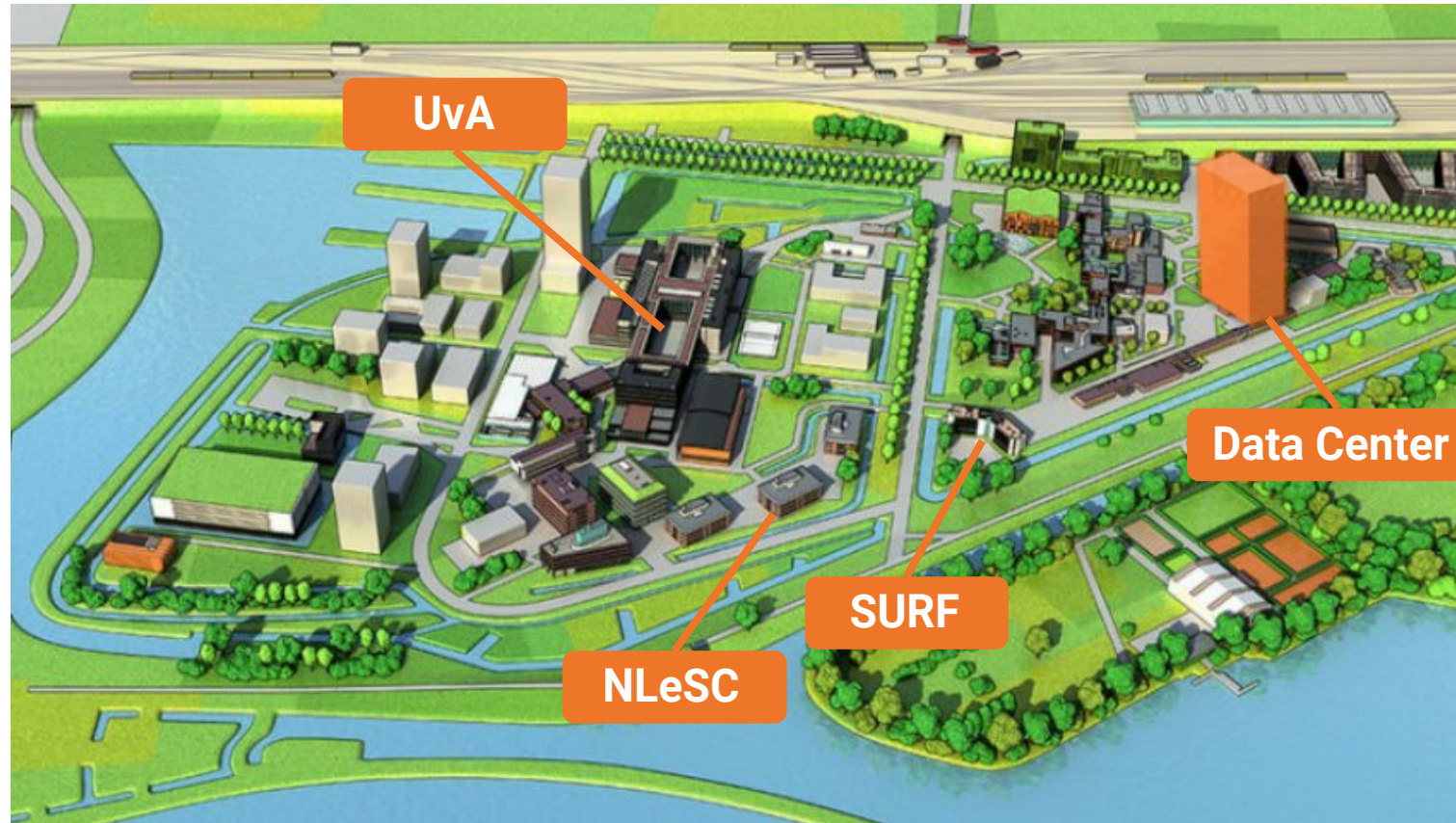
High-Performance Computing (HPC) aggregates computing power and employs **parallel processing** to solve complex problems in **science, engineering, and business**. It enables **advanced simulations, large-scale data analysis, and high-speed computations** beyond the capabilities of standard desktops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



| SURF facilities



| SURF location in 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

Year	Machine	R _p (Gflop/s)	kW	Gflop/s/kW			
2000	PlayStation 2	6.2	0.05	124	1,000,000	520	1923
2006	PlayStation 3	230.4	0.2	1,150	1,840,000	850	2168
2012	Raspberry Pi 1	0.041	0.002	16.4	9,810,000	710	13,817
2019	MacBook Pro M1 Max	10,400	0.1	104,000	14,794,000	1,500	9,862
2021	iPhone 13	1,712	0.015	114,133	36,455,000	1,800	20,252
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/>

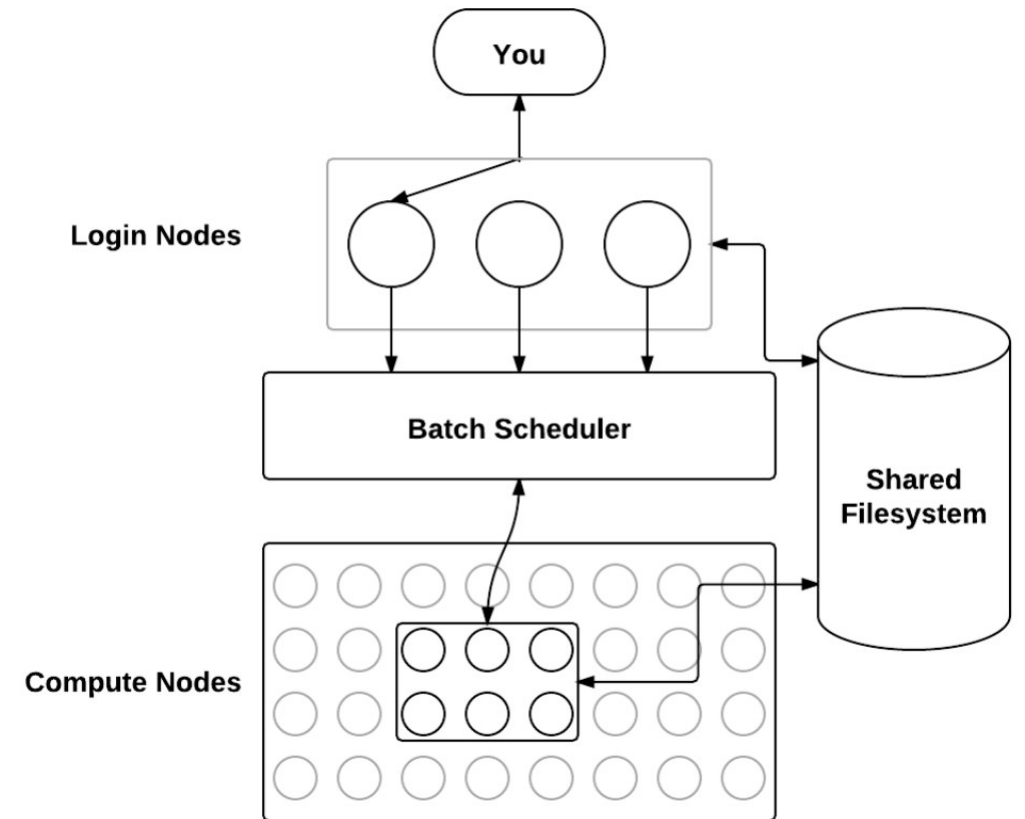
| Schematic overview of a supercomputer

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

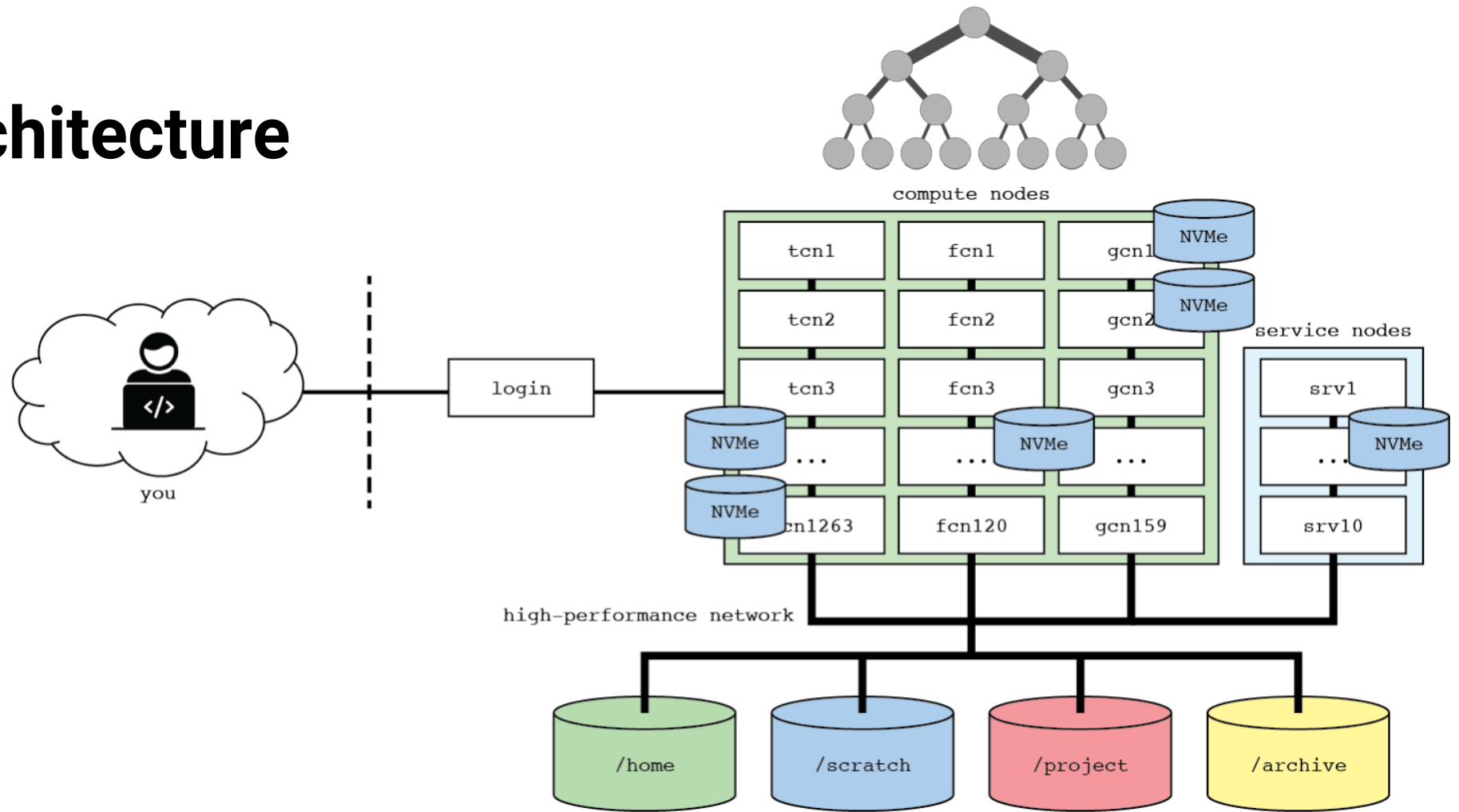
- **Login nodes** provide an environment for **code development, compilation, and job submission**.

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** makes the **same data available** on login nodes and compute nodes, so one workflow can use both.



| Snellius architecture



- 1,258 thin rome/genoa nodes + fat/high-memory nodes + 604 GPUs.
- 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	SBU/hour	Smallest allocation
rome	521	256 GB	128	AMD Rome 7H12 2.6 GHz			128	1/8
fat_rome	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	737	384 GB	192	AMD Genoa 9654 2.4 GHz			192	1/8
fat_genoa	48	1,536 GB	192	AMD Genoa 9654 2.4 GHz			288	1/8
gpu_a100	63	512 GB	72	Intel Xeon 8360Y 2.5 GHz	4	NVIDIA A100 40 GB	512	1/4
gpu_h100	88	768 GB	64	AMD EPYC 9334 2.7 GHz	4	NVIDIA H100 (SXM5) 94 GB	768	1/4
staging	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/spaces/WIKI/pages/92668136/System+details>

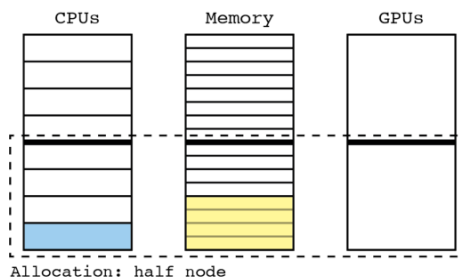
Obtaining an account: <https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/30660190/Obtaining+an+account+on+Snellius>

Software stack and building: <https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/30660266/Software>

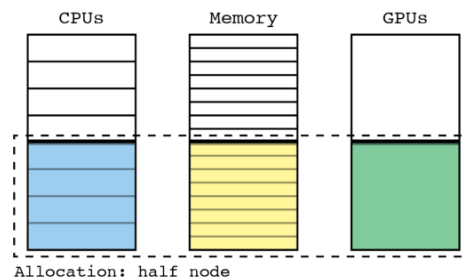


| How does accounting work?

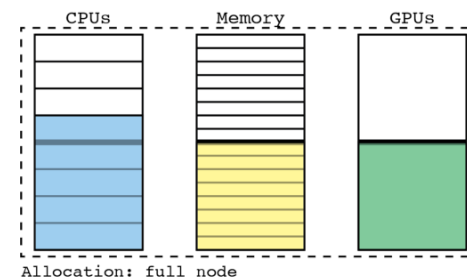
Partition	#Nodes	Memory per node	#Cores	CPU model	#GPUs	GPU model	SBU/hour	Smallest allocation
Example	1	16 GB	8	Processor	2	Accelerator	64	1/2



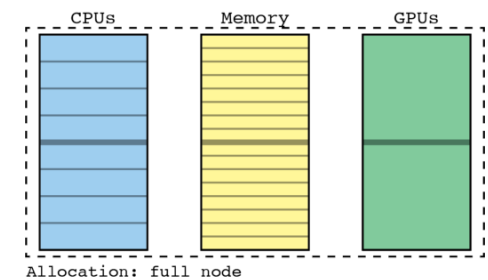
32 SBU



32 SBU



64 SBU



64 SBU

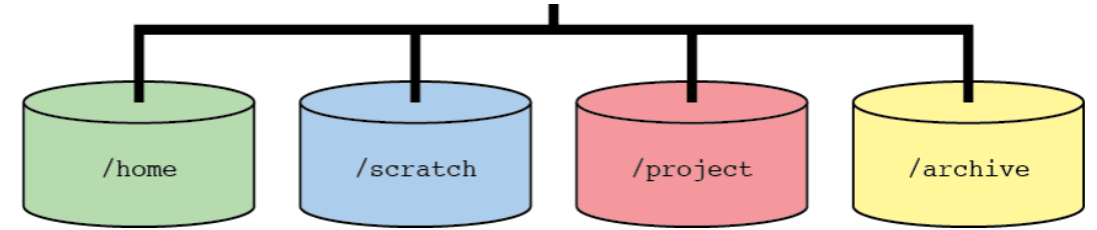
Each node provides a range of resources, including CPU cores, memory, accelerators (e.g., GPUs), and local disk storage. For simplicity, documentation typically refers to allocations in terms of the number of CPU cores. However, in practice, **resource allocation is determined by the most restrictive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

| File systems on Snellius

/home/user

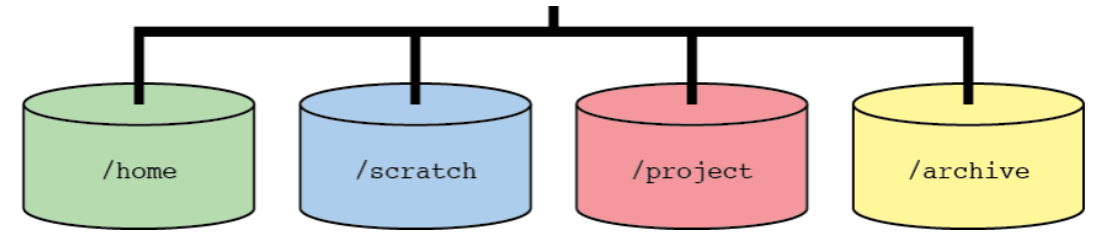
- User home directory, **200 GB per user**.
- **Based on fast, parallel file system (GPFS).**
- **Backed up regularly**, so data stays safe.
- Intended for storing **important files**, such as:
 - **Source codes**
 - **Scripts**
 - **Small to medium-sized input and output data**
- **Not intended for 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

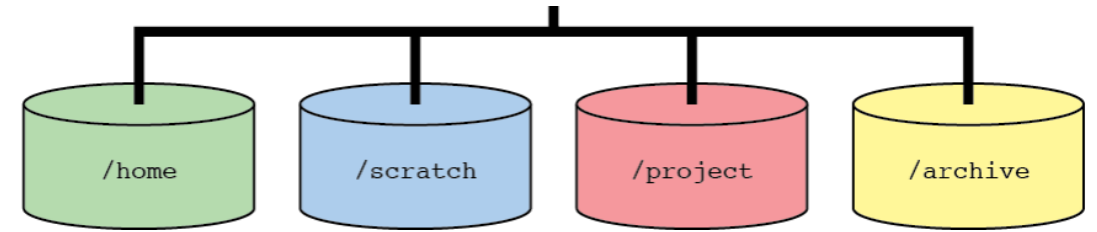
- User working space, **8 TB guaranteed**.
- **Based on fast, parallel file system (GPFS).**
- **Not backed up.**
- **Temporary storage:** data is removed after 6 days (local), 14 days (shared).
- Two different implementations:
 - **Scratch local.**
 - Dedicated to **individual jobs and compute nodes** 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

- 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).



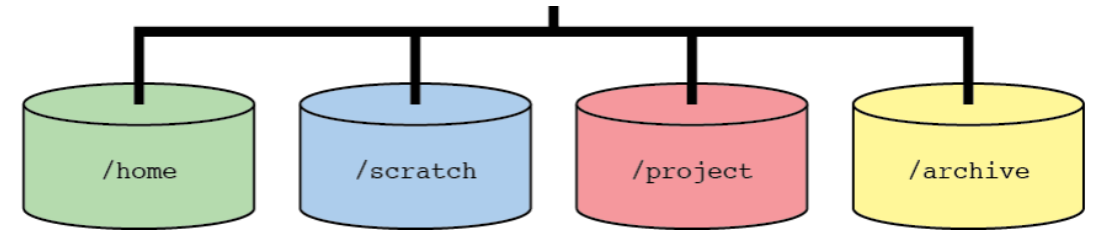
| File systems on Snellius

/projects

- Project working space, given on request.
- **Based on fast, parallel file system (GPFS).**
- **Not backed up.**
- **Persistent 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.

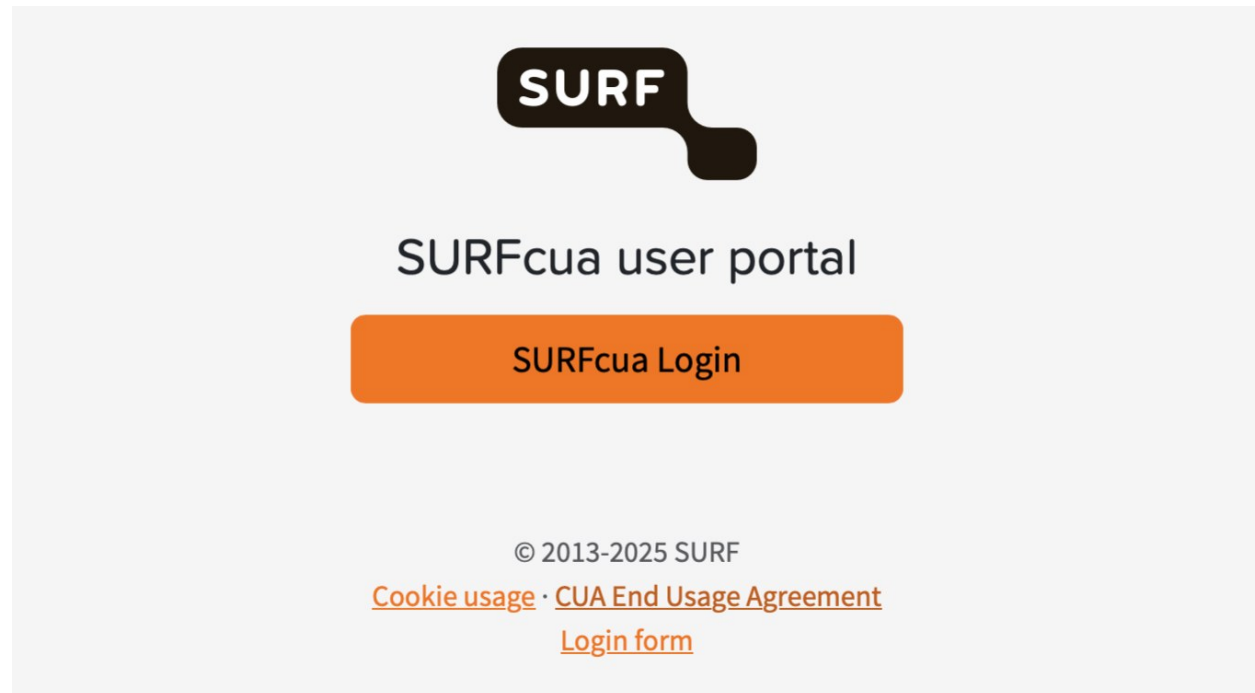


| Connecting to Snellius

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.”

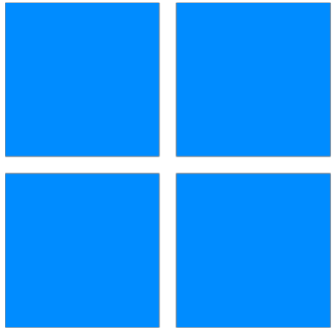
<https://portal.cua.surf.nl>



Connecting to Snellius via terminal

The primary way of accessing Snellius is a **terminal-based connection using SSH**. Through it, you work on the system, manage files, and submit jobs remotely from your local machine.

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



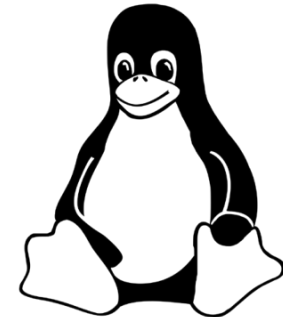
Windows

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



macOS

- Native terminal. XQuartz needed for graphical output.
 - <https://www.xquartz.org>



Linux

- Native terminal.

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

| Connecting to Snellius via terminal

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

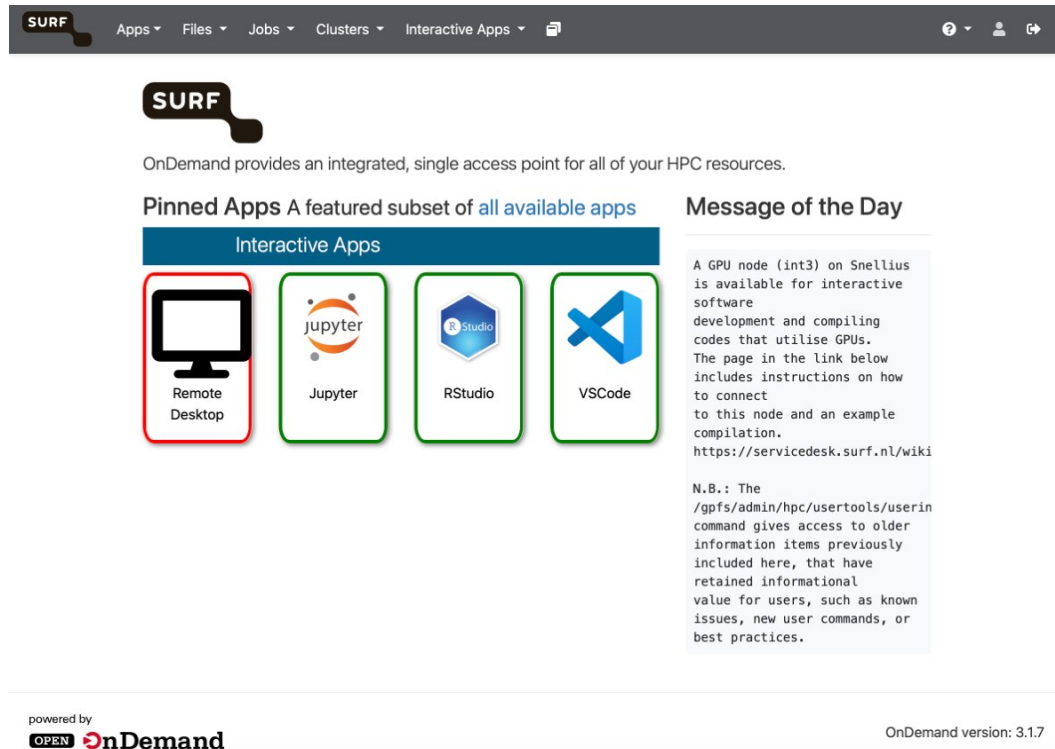
- Type: `ssh <username>@snellius.surf.nl`.
- For untrusted connections, use the documented SURF door node.

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

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

```
user@local:~$ printf 'hello\n' > local-file.txt
user@local:~$ scp ./local-file.txt scur0000@snellius.surf.nl:~/
user@local:~$ scp scur0000@snellius.surf.nl:~/local-file.txt downloaded.txt
user@local:~$ cat downloaded.txt
hello
user@local:~$ ssh scur0000@snellius.surf.nl 'ls local-file.txt'
local-file.txt
```

Connecting to Snellius in a web browser



Besides [accessing Snellius via a terminal](#), the system can also be used via a web browser using the **OpenOnDemand** 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>

| This course runs on a course cluster built like Snellius

- The hands-on of this course run on a **course cluster** built from SURF Research Cloud workspaces with the same parts as Snellius: Slurm, the EESSI software stack, a shared file system, a web portal. What you type here works on Snellius; the numbers are smaller.
- **Login:** `ssh <name>@<manager>` or the web portal; your trainer's setup link makes your password. There is one login node and no SURFcua account.
- **Software:** `module load 2025` loads EESSI 2025.06, the names Snellius' 2025 stack uses. Old names in published scripts (2024, foss/2023a) load the same stack. QCG-PilotJob is a site build here, and the course scripts' own lines (`module load 2024, QCG-PilotJob/0.14.1-gfbf-2024a`) load it, as on Snellius.
- **Nodes:** two cores each, ten on the GPU node, 16 in all, Ethernet in place of InfiniBand. `rome` and `thin_course` are names for the same nodes, `gpu` holds the two cards. Shrink `--ntasks` and `--cpus-per-task` on the `sbatch` line; the scripts stay as published.
- **File systems:** `/home` and `/project` on one shared file system, `/scratch` over the nodes' disks, `$TMPDIR` on the node. No `/scratch-shared`, `/scratch-node`, `/archive`, no `myquota`: use `df -h` and `du -sh`.
- **Budget:** none. The collaboration's wallet pays; `accinfo` and system billing units do not exist; `sacct` reports core-seconds in `CPUTimeRAW`.

LUMI, a European flagship supercomputer



1 SYSTEM
380 Petaflop/s
PEAK PERFORMANCE

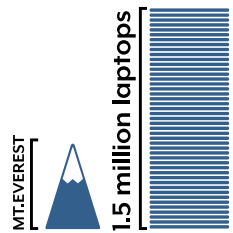
One of Europe's fastest
supercomputers

One of the greenest supercomputers in
the world

#11 on the Top500 list

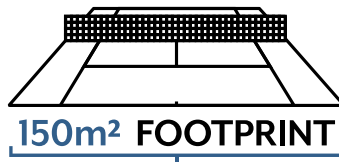
 **100%**
HYDROELECTRIC
ENERGY

COMPUTING POWER EQUALS
1.5 MILLION
MODERN LAPTOP'S
CAPACITY



LUMI

2 x



ONE OF THE MOST
ADVANCED
PLATFORMS FOR AI

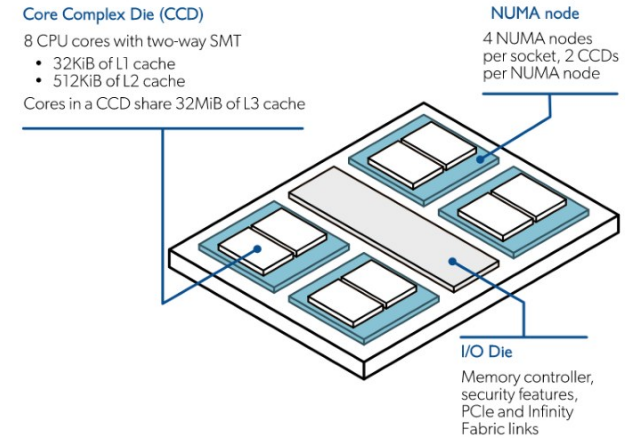
Owned by EuroHPC JU and
hosted by the LUMI
Consortium

SURF

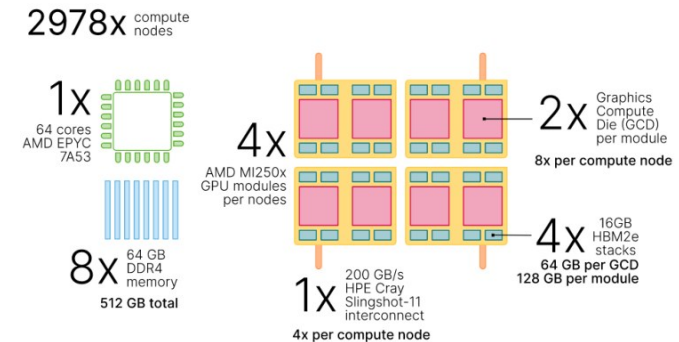
LUMI Hardware

- LUMI-C
 - 2048 CPU nodes
 - Two 64-core AMD EPYC 7763 (Milan)
 - 256 GiB RAM (512 GiB / 1024 GiB for 128/32 high memory nodes)
- LUMI-G
 - 2978 GPU nodes
 - Four AMD MI250x and one 64-core AMD EPYC (Trento)
- Network
 - HPE Cray Slingshot-11 interconnect, Dragonfly topology

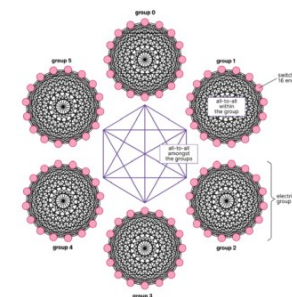
AMD EPYC 7763



LUMI-G node

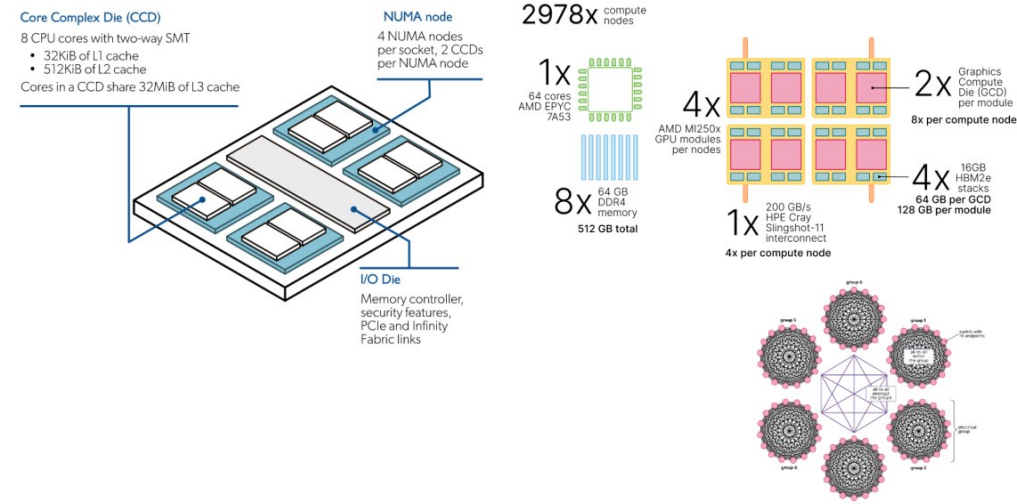


Dragonfly topology



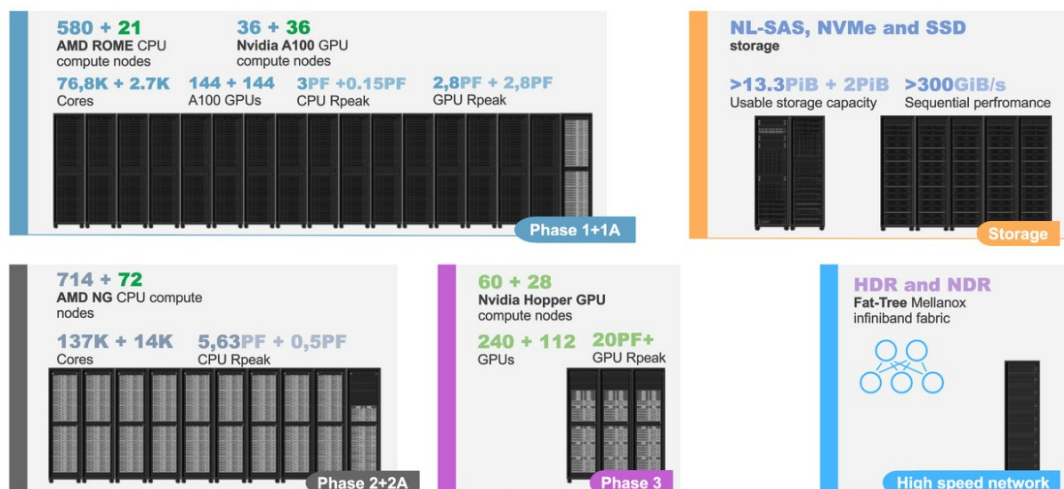
LUMI Hardware

- LUMI
 - 2048 CPU nodes (AMD Milan)
 - 2978 GPU nodes with AMD GPUs
 - HPE Cray Slingshot-11 interconnect, Dragonfly topology
- Snellius
 - 1382 CPU nodes, including fat and high-memory nodes
 - 151 GPU nodes with NVIDIA GPUs
 - Infiniband network, Fat-tree topology



System at a glance

1.525	230K	552	>30PF
Compute nodes	CPU Cores	GPUs	Rpeak



| LUMI Hardware

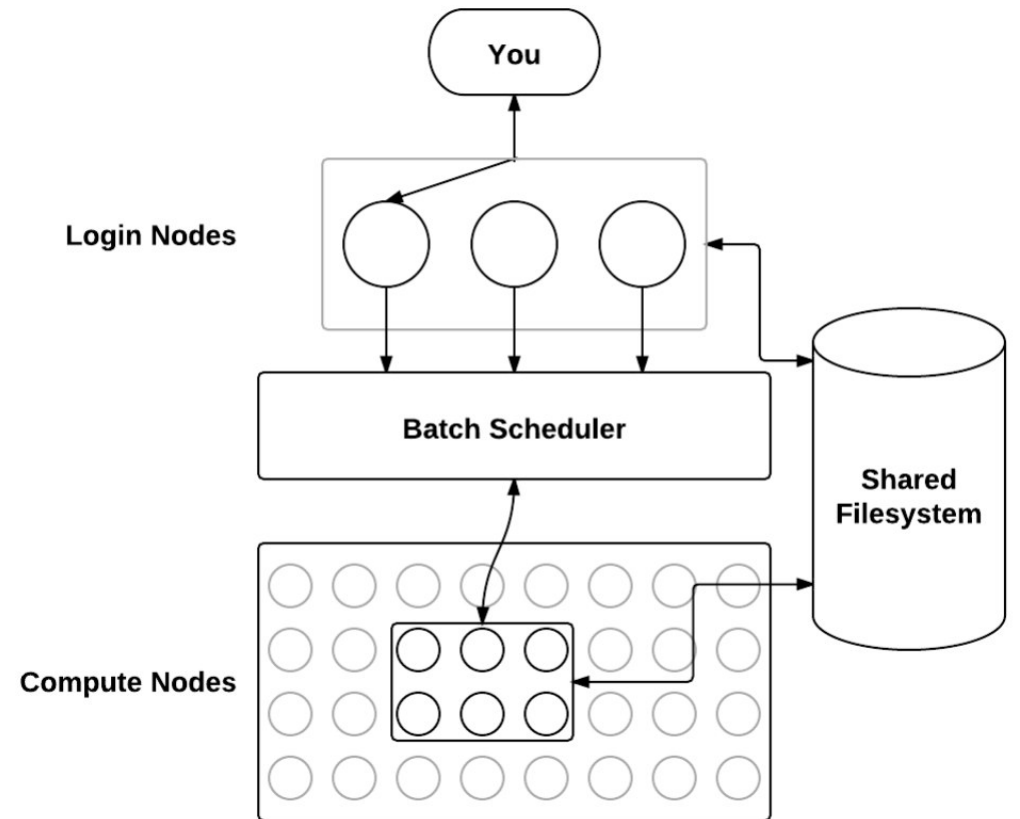
- Projects that benefit from [LUMI's scale](#)
 - Large-scale GPU projects
 - E.g. distributed training of deep neural networks
 - More than ~32-64 GPUs
- Projects interested in [LUMI specifics](#)
 - AMD hardware and tools
 - HPE Cray software stack and tools
 - Slingshot interconnect and Dragonfly network architecture
- For which projects is [Snellius](#) the better option?
 - Large-scale CPU-only projects
 - Projects that require NVIDIA GPUs

| 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:
 - <https://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 <user name>
	-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=rome

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 check 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=rome

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

- Copy exactly the lines of the example into a new 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**?
- Load 2025 foss/2025a before trying **mpirun hostname**.

| Running jobs, third example

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

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

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

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

- Put the lines of the two examples in two text files, "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?

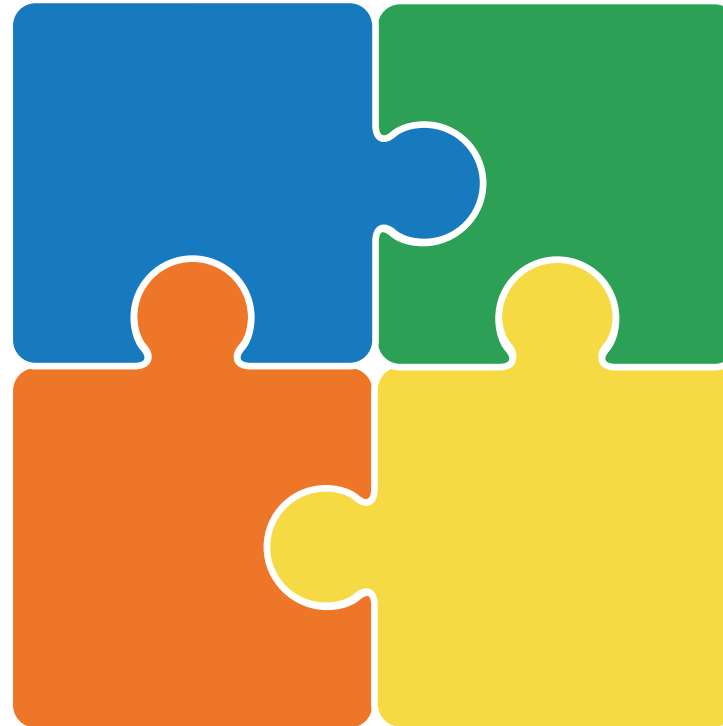
| Running jobs, practical tips

Give the scheduler an accurate wall-time estimate

An **accurate job time** helps the scheduler plan, cuts queue times, and avoids holding resources that other jobs need.

Don't use /home as work dir

Your home directory has limited performance for high-throughput tasks; use /scratch - shared 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.

Use 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=rome
```

```
module load 2025  
module load foss/2025a
```

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

```
srun a.out
```

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

SURF

Job scripts consist of:

- the “shebang” line: `#!/bin/bash`,
- scheduler directives,
- command(s) that load software modules and set the environment,
- steps to prepare the input,
- the 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 use.
- -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.

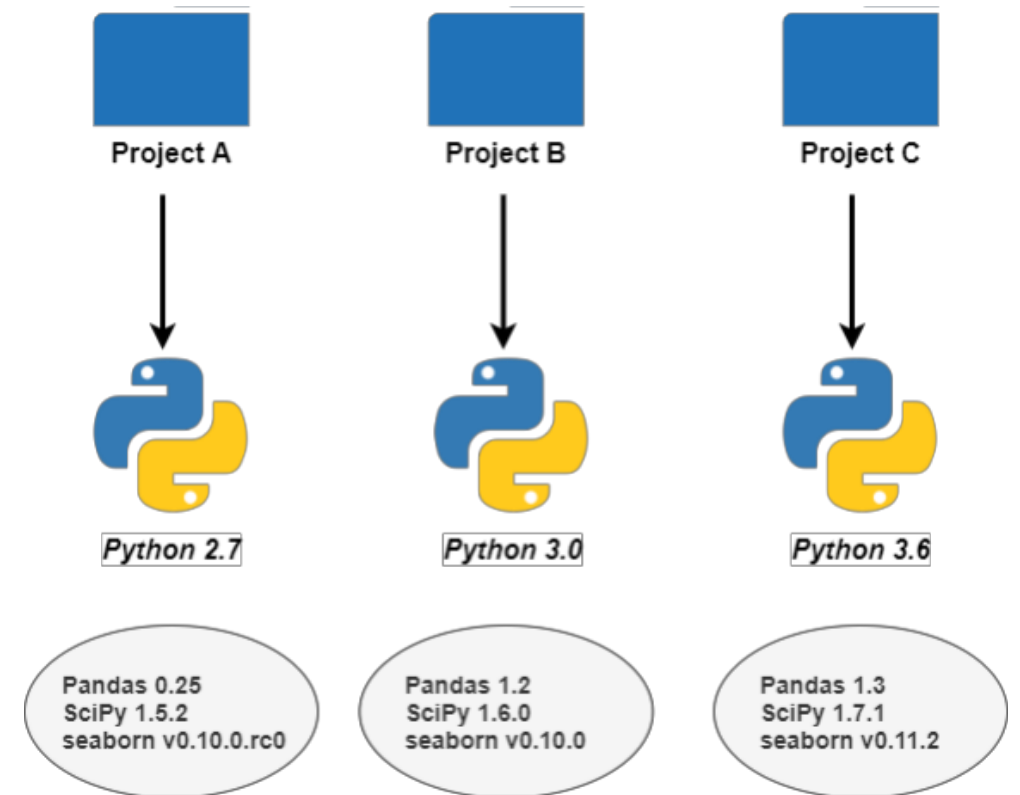
Sbatch: <https://slurm.schedmd.com/sbatch.html>

| 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.

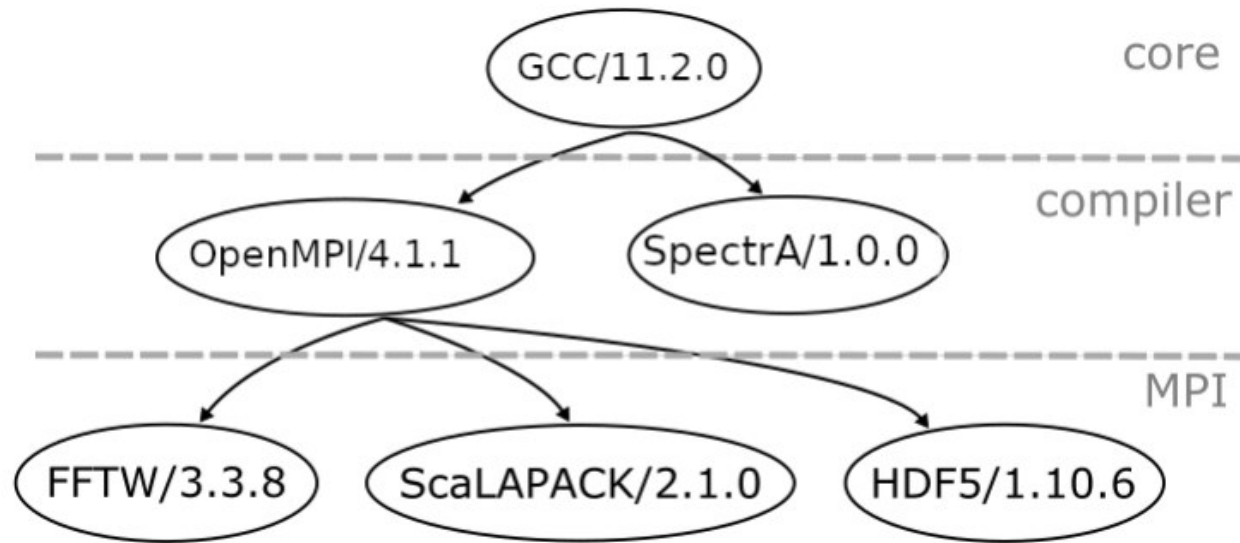


[source](#)

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 does not scale** and complicates system-wide consistency, performance tuning, and long-term reproducibility.

| Environment modules



[source](#)

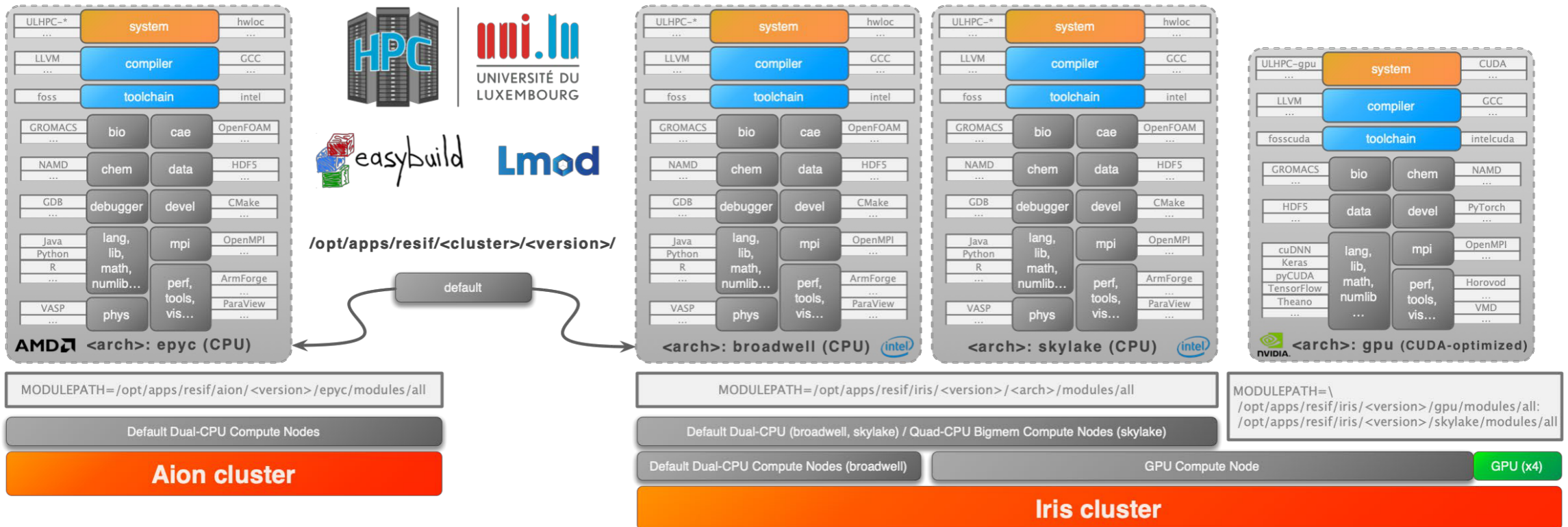
Lmod



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



| 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 Conda software installed by users works properly and efficiently.
- SURF does **not offer support for locally installed Conda software**.
- We strongly advise **against mixing software from the module system with Conda environments**; the two systems often conflict.

Module	Description
2025	Production stack and is fully supported
2024	Previous production stack, it only has limited support
2023	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
<code>module help</code>	Provides information about the command options.
<code>module avail [<modulename>]</code>	Displays modules directly available in the system.
<code>module spider [<modulename>]</code>	Reports every module on the system. In a hierarchical system, module avail lists only the modules you can load right away, whereas module spider also finds the ones behind other modules.
<code>module list</code>	Shows the modules loaded in your environment.
<code>module load <modulename></code>	Loads <modulename> into your environment.
<code>module unload <modulename></code>	Removes <modulename> from your environment.
<code>module purge</code>	Removes all modules from your environment.
<code>module whatis <modulename></code>	Shows information about <modulename>.

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

| Hands-on

| Modules example

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

<< LOAD MODULES >>
echo "START"
<< CALL THE R SCRIPT >>
echo "DONE"
```

- Could you modify the initial script to prepare it as a template to run an R expression?
 - You will need to add the needed modules and also include the call to the command "Rscript"
 - Sample hint: Rscript -e "rnorm(1)"
- Explore the module environment to find the modules (module avail, module load)
- Remember that you can submit this job with "**sbatch** job.sh" and check the status with "**squeue -u login_id**"
- Can you see the output of the call?

| Batch jobs: Python, then OpenMP

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=32
#SBATCH --time=00:10:00
#SBATCH --partition=rome
set -euo pipefail
module load 2025
module load GCC/14.2.0
for ncores in 1 2 4 8 16 32; do
    if (( ncores <= SLURM_CPUS_PER_TASK ))
    then
        export OMP_NUM_THREADS=$ncores
        srun --cpu-bind=cores ./pi
    fi
done
```

- Clone the course and open `python.sh` in its hands-on directory:
 - `git clone https://github.com/sara-nl/introduction-to-supercomputing.git`
`cd introduction-to-supercomputing/course-module-fundamentals-of-hpc/hands-on`
- Run it through “**sbatch** `python.sh`” and follow it in the queue with “**squeue -u login_id**”
- If you needed to use some input file or you would generate an output file, where would you put the copy commands for scratch?
- Now try the same with “`pi.sh`”, but first compile the code! (you may run the script `./compilepi` in the login node directly)
- Can you play around with the variable ‘`ncores`’ and see some parallel efficiency?

| Some additional information

- <https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/30660184/Snellius>
- More courses by SURF (for research and more):
 - EuroCC Netherlands Agenda
 - <https://eurocc-netherlands.nl/calendar/category/training-en/>
 - SURF Agenda
 - <https://www.surf.nl/en/agenda>
 - SURF training mailing list
 - <https://list.surf.nl/postorius/lists/training-announce.list.surf.nl/>

| Abbreviations

- Arithmetic logic unit (ALU)
- Advanced Micro Devices (AMD)
- Control Data Corporation (CDC)
- Computational fluid dynamics (CFD)
- Central user administration (CUA)
- Direct liquid cooling (DLC)
- European Environment for Scientific Software Installations (EESSI)
- AMD server processor family (EPYC)
- Hewlett Packard Enterprise (HPE)
- International Business Machines (IBM)
- Intel Xeon Phi Knights Landing (KNL)
- Large Unified Modern Infrastructure (LUMI)
- Stichting Academisch Rekencentrum Amsterdam (SARA)
- System billing unit (SBU)
- Silicon Graphics (SGI)
- Solid-state drive (SSD)
- Samenwerkende Universitaire Reken Faciliteiten (SURF), the original name

Thank you!

