



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

Ine Arts, Franky Backeljauw, Michele Pugno, Robin Verschoren

Version Fall 2026 — Part 1

# Table of contents – Part 1

## 1. Introduction to the VSC

- UAntwerp Tier-2 infrastructure
- VSC Tier-1 infrastructure
- Characteristics of a HPC cluster

## 2. Connect to the cluster

- Types of cluster nodes
- Open OnDemand web portal

## 3. Transfer your files to the cluster

- File systems and user directories
- Block and file quota limits
- Globus data transfer platform
- Best practices for file storage

## X. Using the command line

- Using SSH and SCP/SFTP
- Globus and OneDrive CLI interfaces

## 4. Select the software and build your environment

- System, development and application software
- Software installation and support
- Selecting software using modules
- Toolchains & the CalcUA modules
- Searching, loading and unloading modules
- Best practices for using modules

## 5. Define and submit your jobs

- Running batch jobs
- Job submission workflow
- Job script example
- Important Slurm concepts
- Slurm resource requests
- Non-resource-related options
- The job environment



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 1 — Introduction to the VSC



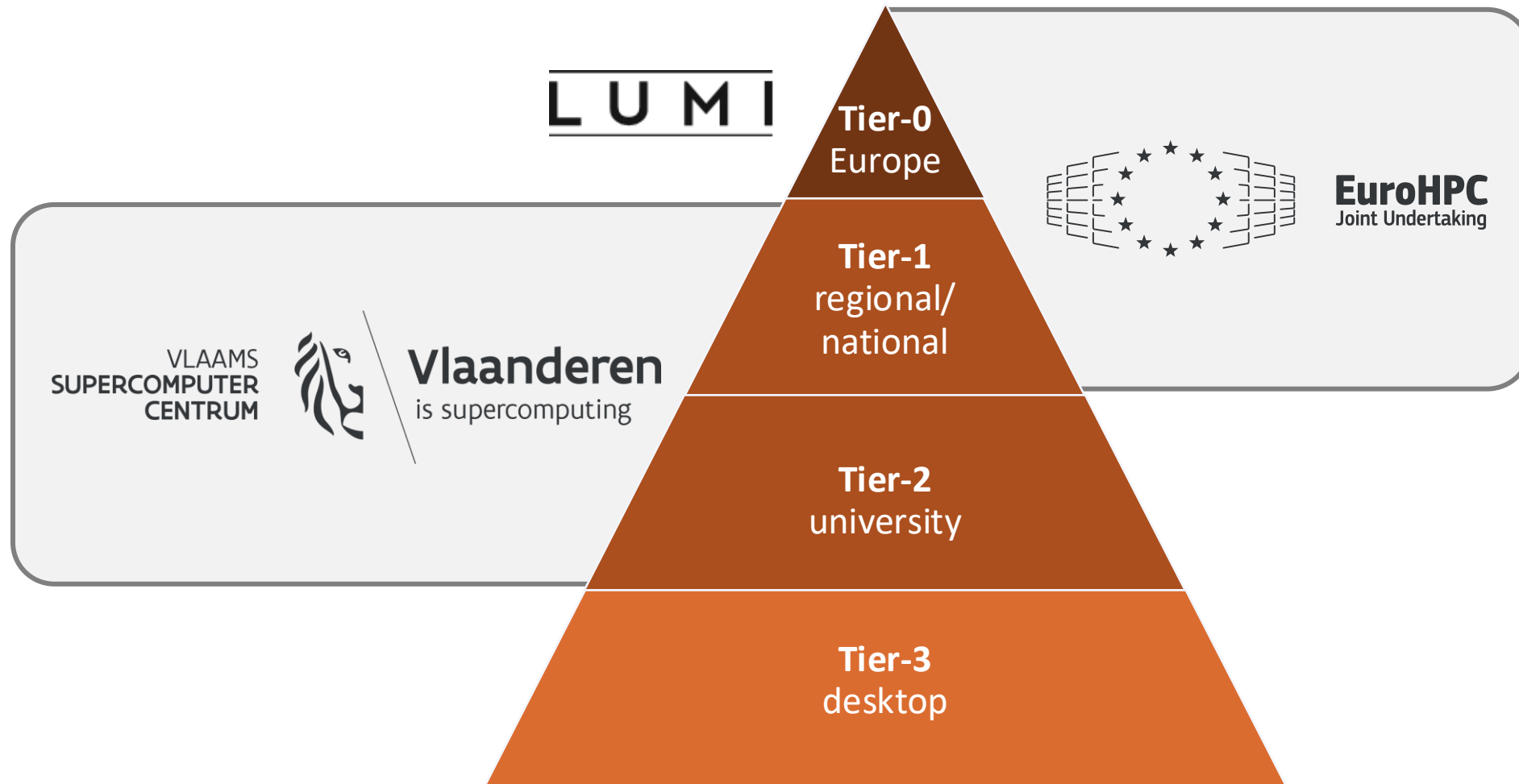
## ➤ HPC core facility CalcUA

- provides HPC infrastructure & software for researchers
- offer training & support
- UAntwerp Tier-2 infrastructure (local)

## ➤ Vlaams Supercomputer Centrum (VSC)

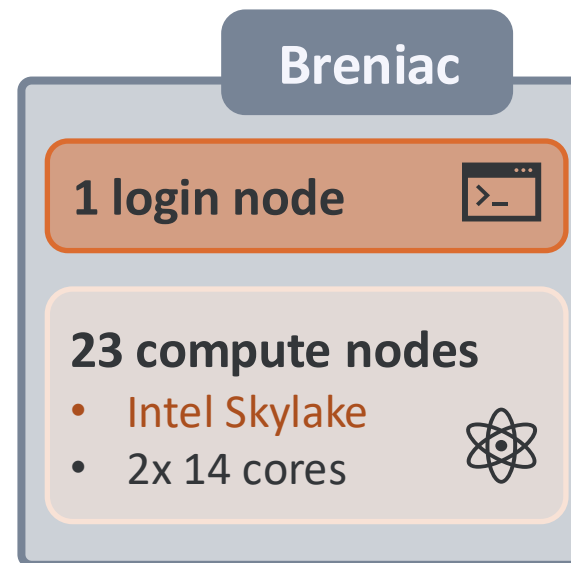
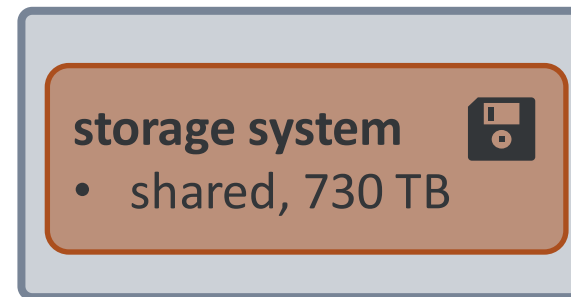
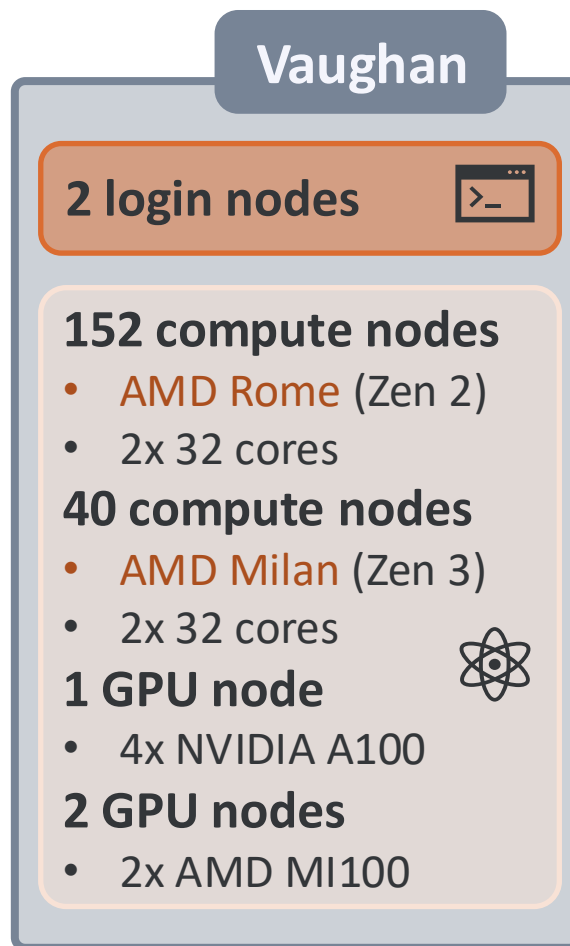
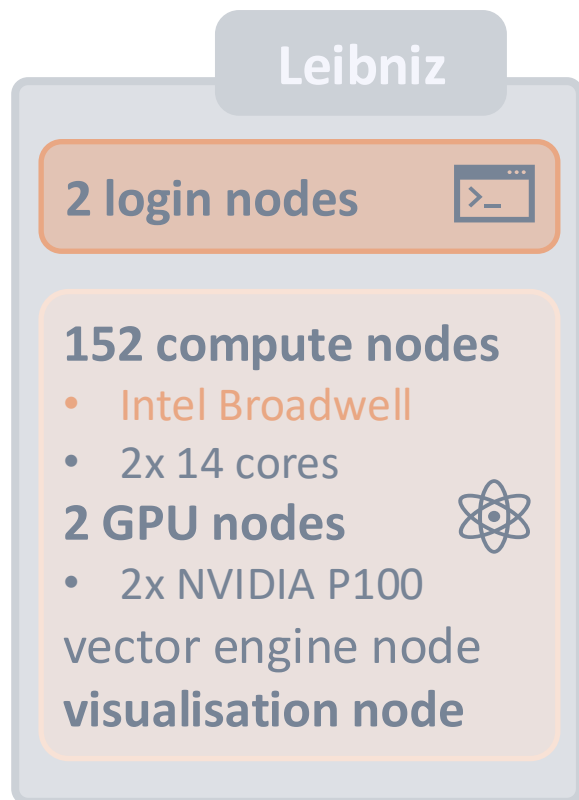
- partnership between 5 University associations: Antwerp, Brussels, Ghent, Hasselt, Leuven
- FWO funded (Research Fund – Flanders)
- goal: make HPC available to all researchers in Flanders – academic and industrial
- provides **central Tier-1** infrastructure
- other **local Tier-2** infrastructures: VUB, UGent and KU Leuven / UHasselt

# The European HPC landscape



# UAntwerp Tier-2 infrastructure

🔗 [UAntwerp Tier-2 Infrastructure](#)



# UAntwerp Tier-2 infrastructure



\ Orchestrating a brighter world

NEC

# New UAntwerp Tier-2 infrastructure — Operational in 2027

## New Tier-2 (UAntwerp)

### 24 CPU compute nodes

- Intel Xeon 6 6960P (Granite Rapids)
- 2 x 72 cores
- 2.5 GB/core



### 1 high-end GPU node

- NVIDIA DGX B200 (Blackwell)
- 8x NVIDIA B200 SXM6
- 1440 GB HBM3e
- 2 x 54 cores
- 18 GB/core



### 3 login/portal nodes



### 1 visualisation node

- 2x NVIDIA RTX PRO 4500

### storage system

- Lustre parallel filesystem
- 300 TB data
- 1.5 PB scratch



➤ Will replace Leibniz — *so long, and thanks for all the fish computations!*

# VSC Tier-1 infrastructure

🔗 [VSC Tier-1 Infrastructure](#)

## Hortense (UGent)

2 Login nodes



384 compute nodes

- AMD Rome (Zen 2)
- 2x 64 cores

20 GPU nodes

- 4x NVIDIA A100



### PHASE 2

384 compute nodes

- AMD Milan (Zen 3)
- 2x 64 cores

20 GPU nodes

- 4x NVIDIA A100



Storage system  
shared, 5.4 PB



# VSC Tier-1 infrastructure

🔗 [VSC Tier-1 Infrastructure](#)

## Sofia (VUB)

### 56 high-core nodes

- AMD Turin (Zen 5c)
- 2 x 192 cores
- 2 GB/core



### 16 high-memory nodes

- 2 x 96 cores
- 8 GB/core



### 22 GPU nodes

- 8x NVIDIA H200 SXM5
- 2 x 96 cores
- 4 GB/core



### 4 visualisation nodes

storage system  
shared, 4.46 PB



# Characteristics of a HPC cluster

- **Shared infrastructure**, used by multiple users simultaneously
  - you need to request the appropriate resources
  - you may have to wait a while before your computation starts
- Expensive infrastructure
  - **software efficiency matters!**
- Built for parallel jobs
  - **no parallelism = no supercomputing**
  - not meant for running a single one-core job
- Remote computation model
  - for *batch computations* rather than interactive applications
- Linux-based systems
  - no Windows or macOS software

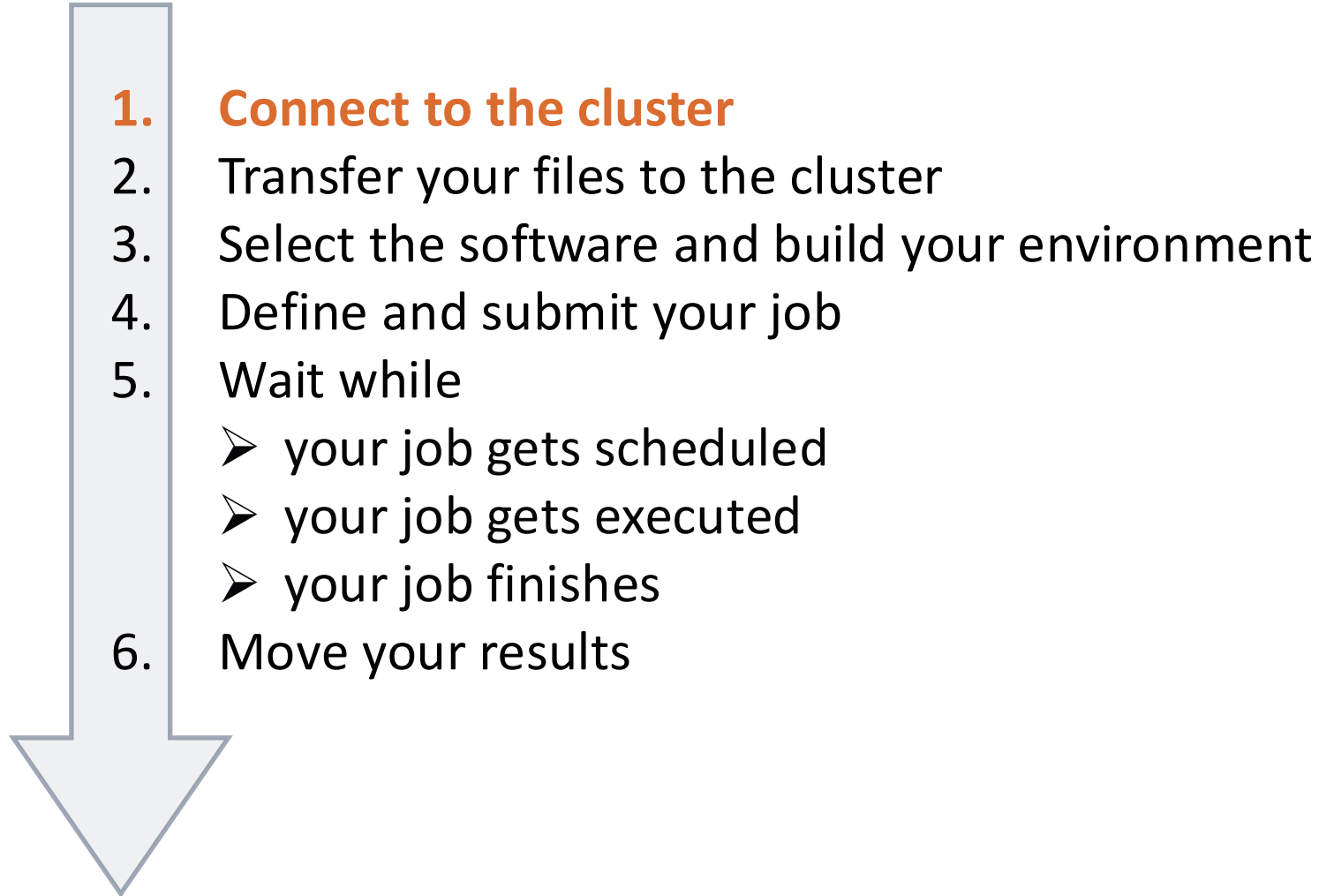


Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

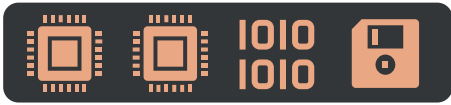
## 2 — Connect to the cluster

# A typical workflow

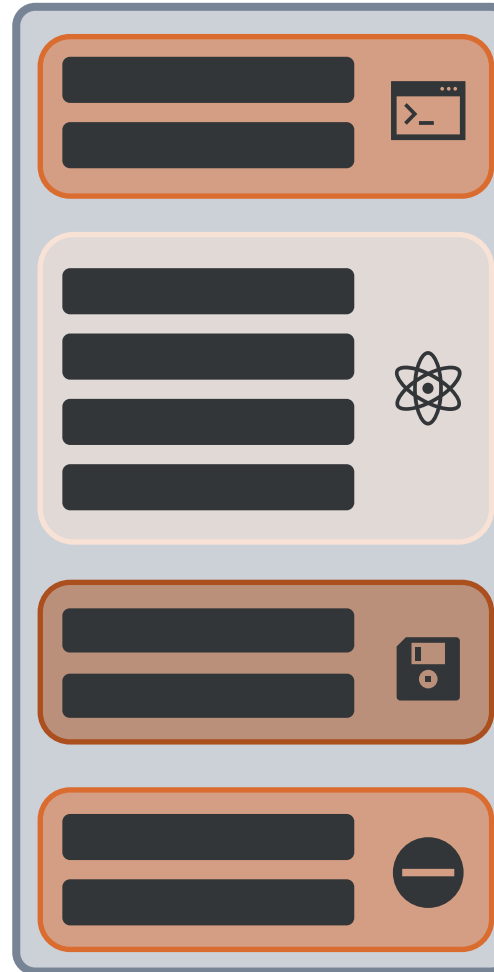


# Types of cluster nodes

- Computer cluster consists of **nodes**
  - each node has specific task(s)



- **Login nodes**
  - SSH access to the clusters
  - edit & submit jobs
  - small compilations
- **Open OnDemand web portal**
  - web access to the clusters
- **Compute nodes**
  - actual computations



Login section

Compute section

Storage section

Admin section

# Web portal access

## 🔗 Open OnDemand web portal

- Provides a user interface to HPC clusters from **within a web browser**
  - browse, create, transfer, view and/or edit files
  - open a shell on one of the login nodes
  - submit and monitor your jobs
  - create job templates
- Use interactive applications
  - interactive shell on the compute nodes, or desktop sessions
  - programming environments — e.g., VS Code, JupyterLab, RStudio
- Log in with UAntwerp or VSC account — *no SSH keys required!*

## 🔗 Open OnDemand

# Web portal access — <https://portal.hpc.uantwerpen.be>



University  
of Antwerp

**UAntwerp OnDemand**

Integrated, single access point for all your HPC resources

## Latest news

---

### Upcoming trainings

- **Introduction to Linux**  
October 15 and 16, 9.30-13.30 h (CMI G.025 (15th), CMI G.026 (16th))
- **HPC@UAntwerp introduction**  
October 22 and 23, 9.30-13.30 h (CMI G.025 (22nd), CMI G.026 (23rd))
- **Supercomputers for starters**  
October 29 and 30, 9.30-13.30 h (CMI G.025 (29th), CMI G.026 (30th))

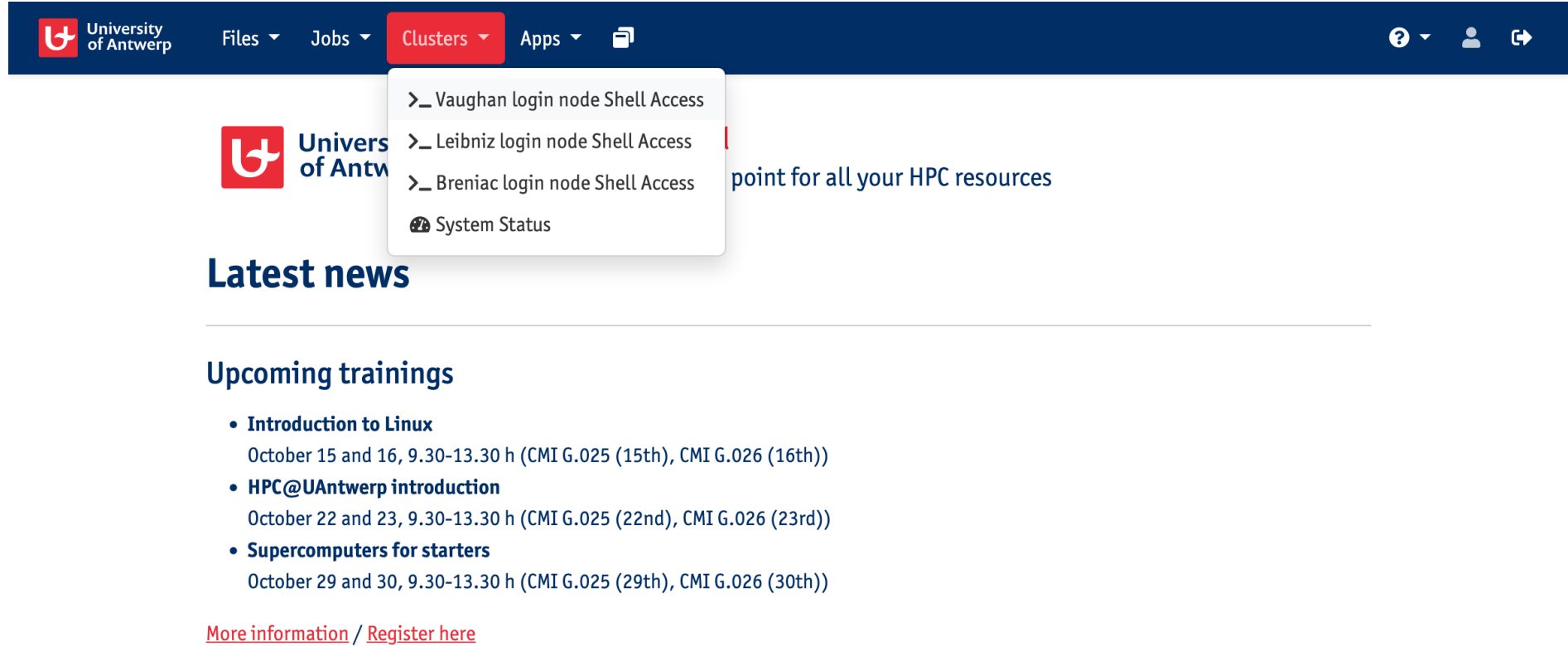
[More information](#) / [Register here](#)

powered by



OnDemand version: 4.0.7

# Web portal access — Connect to the cluster



University of Antwerp

Files ▾ Jobs ▾ Clusters ▾ Apps ▾

University of Antwerp

point for all your HPC resources

## Latest news

### Upcoming trainings

- **Introduction to Linux**  
October 15 and 16, 9.30-13.30 h (CMI G.025 (15th), CMI G.026 (16th))
- **HPC@UAntwerp introduction**  
October 22 and 23, 9.30-13.30 h (CMI G.025 (22nd), CMI G.026 (23rd))
- **Supercomputers for starters**  
October 29 and 30, 9.30-13.30 h (CMI G.025 (29th), CMI G.026 (30th))

[More information](#) / [Register here](#)

powered by



OnDemand version: 4.0.7

# Web portal access — Connect to the cluster

```
Host: login.vaughan Themes: Default
Last login: Wed Oct 15 13:53:20 2025 from 10.28.228.232

  vsc20003@login2.vaughan
  CLUSTER: vaughan
  ARCH: zen2
  OS: rocky9

UAntwerp Tier-2 infrastructure:
https://docs.vscentrum.be/antwerp/tier2\_hardware.html
https://www.uantwerpen.be/hpc

VSC documentation and status:
https://docs.vscentrum.be
https://status.vscentrum.be

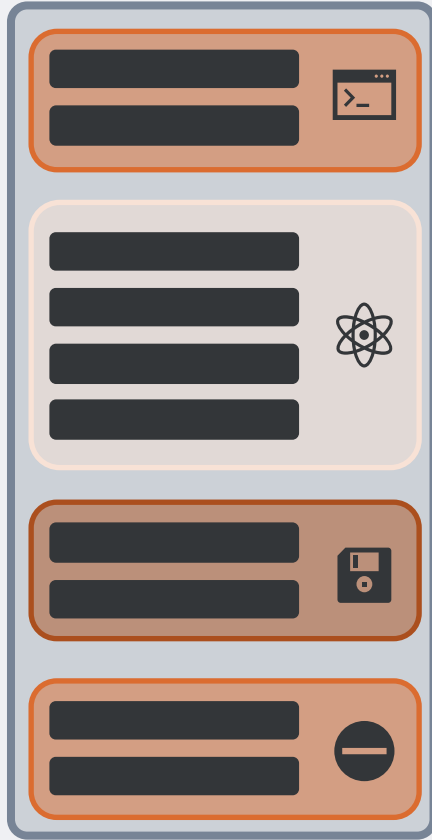
Questions or problems? Do not hesitate and contact us:
hpc@uantwerpen.be

Your current quota usage is:

file system /user/antwerpen
  using 2.575G of 3G space, 6,987 of 20k files
file system /data/antwerpen
  using 1.272G of 256G space, 23,931 of 1M files
file system /scratch/antwerpen
  using 883.9G of 10T space, 116,172 of 2M files

✓ [23:37] vsc20003@login2.vaughan $VSC_HOME $
```

# Summary — Access to a HPC cluster



- A HPC cluster is **shared infrastructure** with multiple users
- It consists of many nodes with different functions: login nodes, compute nodes, storage nodes, ...
- You access the machine through the **login nodes**
  - web portal: *login node shell access*
- You **request** a **compute** allocation for your computations
  - **batch scripts** (see 5 - define and submit your job)
  - web portal: *interactive shell*



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 3 — Transfer your files to the cluster

# A typical workflow



1. Connect to the cluster
- 2. Transfer your files to the cluster**
3. Select the software and build your environment
4. Define and submit your job
5. Wait while
  - your job gets scheduled
  - your job gets executed
  - your job finishes
6. Move your results

# File systems and user directories

- **/scratch/antwerpen/2xx/vsc2xxyy**
  - working directory: calculation input/output
  - highest performance – for large files
  - local only, no backup
- **/data/antwerpen/2xx/vsc2xxyy**
  - for datasets, software, longer-term
  - slower – for small files
  - exported to other VSC sites (\*)
- **/user/antwerpen/2xx/vsc2xxyy**
  - only for account configuration files
  - exported to other VSC sites (\*)

\$VSC\_SCRATCH



\$VSC\_DATA



\$VSC\_HOME



# Block and file quota limits

- **Block quota** limits the *size of data*
- **File quota** limits the *number of files*
- Default values (but you can request more)

|                                                                                   | <u>File system</u> | <u>Block quota</u> | <u>File quota</u> |
|-----------------------------------------------------------------------------------|--------------------|--------------------|-------------------|
|  | /scratch           | 50 GB              | 100 k             |
|  | /data              | 25 GB              | 100 k             |
|  | /home              | 3 GB               | 20 k              |

- Show quota: at login or with the **myquota** command
- Note: on /scratch, the number of files corresponds to number of **data chunk files**
  - 1 end-user created file can be spread over at most 8 data chunk files
  - does not include the number of directories

# Web portal access — Transfer your files

 University of Antwerp

Files ▾ Jobs ▾ Clusters ▾ Apps ▾ 

[Open in Terminal](#) [Refresh](#) [+ New File](#) [New Directory](#) [Upload](#) [Download](#) [Globus](#) [Copy/Move](#) [Delete](#)

[Home Directory](#)  
[VSC\\_HOME](#)  
[VSC\\_DATA](#)  
[VSC\\_SCRATCH](#)

[↑](#) / data / antwerpen / 200 / vsc20003 / [Change directory](#) [Copy path](#)

☐ Show Owner/Mode ☐ Show Dotfiles Filter:

Showing 6 of 9 rows - 0 rows selected

| <input type="checkbox"/> | Type ▲ | Name      |                                                                                       | Size | Modified at         |
|--------------------------|--------|-----------|---------------------------------------------------------------------------------------|------|---------------------|
| <input type="checkbox"/> | Folder | easybuild |    | -    | 25/9/2025 15:18:09  |
| <input type="checkbox"/> | Folder | github    |    | -    | 28/11/2024 09:25:20 |
| <input type="checkbox"/> | Folder | scripts   |   | -    | 18/3/2025 15:56:26  |
| <input type="checkbox"/> | Folder | selenium  |  | -    | 1/7/2025 07:11:58   |
| <input type="checkbox"/> | Folder | tests     |  | -    | 20/2/2025 14:38:03  |
| <input type="checkbox"/> | Folder | vsc       |  | -    | 23/4/2024 11:59:23  |

# Globus — Data sharing platform

## 🔗 Globus web interface

- Web app to transfer large amounts of data between local computer and/or remote servers
  - offers premium data sharing features, guest collections and connectors (e.g., for OneDrive)
  - transfers will resume automatically, synchronization options available
  - has a command-line interface as well as a Python SDK
- HPC@UAntwerp collection: VSC UAntwerpen Tier2 filesystems
  - access to both user and shared group directories
  - log in with UAntwerp or VSC account — *note: active VSC account needed*
- Transfer from/to local computer (laptop/desktop): Globus Connect Personal
- Transfer between remote servers: *no local software needed!*

## 🔗 Globus data sharing platform — Overviews & Concepts

# Globus — Data sharing platform



## Log in

### Use your organizational login

e.g., university, national lab, facility, project



Universiteit Antwerpen

By selecting Continue, you agree to Globus [terms of service](#) and [privacy policy](#).

Continue

LOG IN USING LESS COMMON OPTIONS



Didn't find your organization? Then use [Globus ID to sign in](#). ([What's this?](#))

# Globus — Data sharing platform



Account ▾

## Globus Web App would like to:

- ✓ Manage your Globus groups (v2) ⓘ
- ✓ View the identities in your Globus account ⓘ
- ✓ Manage data using Globus Transfer ⓘ
- ✓ Search for data using your identities and groups ⓘ

To work, the above will need to: ▾

By clicking "Allow", you allow **Globus Web App**, in accordance with its [terms of service](#) ⓘ and [privacy policy](#) ⓘ, to use the above listed information and services. You can rescind this and other [consents](#) ⓘ at any time.

Allow

Deny

# Globus — Data sharing platform

The screenshot displays the Globus File Manager interface. On the left is a dark blue sidebar with navigation icons for File Manager, Activity, Collections, Groups (with a '2' badge), Flows, Compute, Timers, Console, Settings, and Logout. The main header area includes the Globus logo, a 'File Manager' title, and a 'Panels' section with three icons. Below the header, the 'Collection' dropdown is set to 'VSC UAntwerpen Tier2 filesystems', and the 'Path' field is empty. A toolbar contains icons for 'select all', 'up one folder', 'refresh list', 'filter', and a 'view' button. The central content area features a yellow warning icon and the text: 'Authentication/Consent is required for Globus transfer service to manage data on this collection.' Below this, it states 'Missing required data\_access consent' and provides a blue 'Continue' button, followed by a 'Need Help?' link. On the right side, a context menu is open, listing actions: Share, Transfer or Sync to..., New Folder, Rename, Delete Selected, Download, Open, Upload, Get Link, Show Hidden Items, and Manage Consent.

File Manager

Panels

Collection: VSC UAntwerpen Tier2 filesystems

Path:

select all up one folder refresh list filter view

**Authentication/Consent is required for Globus transfer service to manage data on this collection.**

Missing required data\_access consent

[Continue](#)

[Need Help?](#)

- Share
- Transfer or Sync to...
- New Folder
- Rename
- Delete Selected
- Download
- Open
- Upload
- Get Link
- Show Hidden Items
- Manage Consent

# Globus — Transfer between remote servers

The screenshot displays the Globus File Manager interface. On the left is a dark blue sidebar with navigation icons for File Manager, Activity, Collections, Groups (with a '2' badge), Flows, Compute, Timers, Console, Settings, and Logout. The main area is titled 'File Manager' and shows a transfer configuration between two collections.

**Transfer Configuration:**

- Collection 1:** VSC UAntwerpen Tier2 filesystems
- Path 1:** /data/antwerpen/200/vsc20003/
- Collection 2:** VSC UGent Tier1 projects
- Path 2:** /dodrio/scratch/projects/astaff/vsc20003/

Buttons for 'Start' (play icon) and 'Transfer & Timer Options' (dropdown) are visible. Below the configuration, a toolbar contains icons for file operations (new, up, refresh, filter, settings, menu).

**File List (Left Panel):**

| Icon              | Name           | Timestamp            | More |
|-------------------|----------------|----------------------|------|
| Folder            | easybuild      | 9/25/2025, 03:18 PM  | >    |
| Folder            | github         | 11/28/2024, 09:25 AM | >    |
| Folder (Selected) | scripts        | 3/18/2025, 03:56 PM  | >    |
| Folder            | selenium       | 7/1/2025, 07:11 AM   | >    |
| Folder            | tests          | 2/20/2025, 02:38 PM  | >    |
| File              | Untitled.ipynb |                      |      |

1 item selected

**Context Menu (Right Panel):**

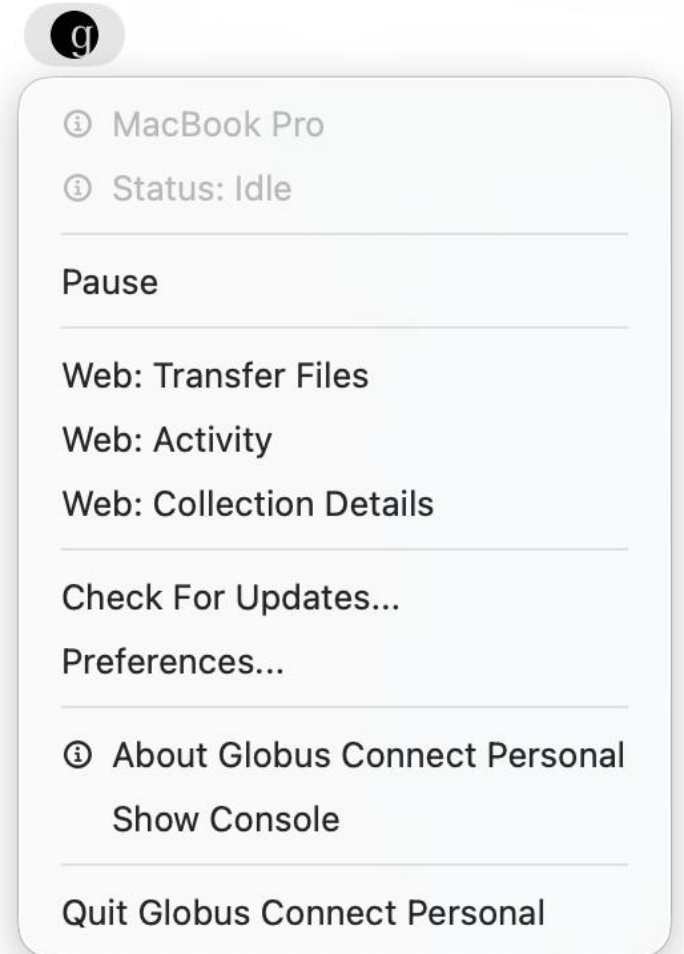
- Share
- Transfer or Sync to...
- New Folder
- Rename
- Delete Selected
- Download
- Open
- Upload
- Get Link

The rightmost panel shows a message: 'This folder is empty.'

# Globus — Transfer from/to local computer

## 🔗 Globus Connect Personal

- Create a **local collection** for your laptop/desktop
  - easily and reliably move and share large amounts of data
  - easily synchronize folders repeatedly — e.g., for **back up**
  - interact with other Globus collections — e.g., VSC ...
  - transfers will resume automatically
- *Warning: be careful when setting accessible directories*
  - e.g., only allow access to dedicated directory for Globus
  - by default, your home directory is selected, so *remove it*



# Globus — Transfer from/to local computer

The screenshot displays the Globus File Manager interface. On the left is a vertical sidebar with icons for File Manager, Activity, Collections, Groups (with a '2' badge), Flows, Compute, Timers, Console, Settings, and Logout. The main area is titled 'File Manager' and shows two panels. The left panel is for 'MacBook Pro' with path '/Users/backelj/Shares/Globus/'. The right panel is for 'VSC UAntwerpen Tier2 filesystems' with path '/data/antwerpen/200/vsc20003/'. Below the panels are 'Start' buttons and a 'Transfer & Timer Options' dropdown. The main content area is split into two columns. The left column lists files: 'handson.tar.gz' (6.81 KB), 'input.zip' (2.16 MB), 'intro-hpc-20251022.pdf' (7.55 MB), 'intro-linux-20251015.pdf' (8.16 MB), and 'scripts.zip' (6.26 KB). The first three files are selected, indicated by checkmarks. The right column lists folders: 'easybuild', 'github', 'scripts', 'selenium', 'tests', and 'vsc'. A status bar at the bottom left indicates '3 items selected'.

| Collection                       | Path                          |
|----------------------------------|-------------------------------|
| MacBook Pro                      | /Users/backelj/Shares/Globus/ |
| VSC UAntwerpen Tier2 filesystems | /data/antwerpen/200/vsc20003/ |

| File/Folder              | Size    | Selected |
|--------------------------|---------|----------|
| handson.tar.gz           | 6.81 KB | Yes      |
| input.zip                | 2.16 MB | Yes      |
| intro-hpc-20251022.pdf   | 7.55 MB | No       |
| intro-linux-20251015.pdf | 8.16 MB | No       |
| scripts.zip              | 6.26 KB | Yes      |

| Folder    | Size                 | Selected |
|-----------|----------------------|----------|
| easybuild | 9/25/2025, 03:18 PM  | No       |
| github    | 11/28/2024, 09:25 AM | No       |
| scripts   | 3/18/2025, 03:56 PM  | No       |
| selenium  | 7/1/2025, 07:11 AM   | No       |
| tests     | 2/20/2025, 02:38 PM  | No       |
| vsc       | 4/23/2024, 11:59 AM  | No       |

# Best practices for file storage

- **The cluster is not for long-term file storage**
  - move back your results to your laptop or server in your department
  - backup exists for /user and /data – not for very volatile data
  - old data on /scratch can be deleted if scratch fills up
- Cluster is **optimised for parallel access to large files**
  - not for tons of small files (e.g., one per MPI process)
- Request more quota on /scratch
  - block quota – without too much motivation
  - file quota – you will have to motivate why you need more files
- *Note: text files are good for summary output, or data for a spreadsheet, but not for storing 1000x1000-matrices — use binary files for that!*

# Hands-on

- Copy some files between your laptop and CalcUA
  - o using the webportal as well as using the **Globus** web app
  - o copy them back to your laptop with both tools
  - o transfer a directory as well
- Download and install Globus Connect Personal
  - o login with vsc2xxxx@vscentrum.be
  - o choose a name for your collection — e.g. <name>-laptop
  - o good practice: configure it to use a dedicated subdirectory only — e.g. ~/Globus
  - o initiate a transfer from CalcUA to your laptop
    - look at the options

# Summary — Transfer your files to the cluster



- There are 3 user directories:
  - **\$VSC\_SCRATCH** - data you need for calculations (large files)
  - **\$VSC\_DATA** - software or data you regularly need on multiple VSC sites
  - **\$VSC\_HOME** - config files only
- Regularly **back up** your files
  - transfer them using **Globus** or `scp/sftp`



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

4 — Select the software and  
build your environment

# A typical workflow

- 
1. Connect to the cluster
  2. Transfer your files to the cluster
  3. **Select the software and build your environment**
  4. Define and submit your job
  5. Wait while
    - your job gets scheduled
    - your job gets executed
    - your job finishes
  6. Move your results

# System software

- Operating system: **Rocky Linux** – currently, version 9.8 – upgrade to Rocky Linux 10 in H1 2027
  - Red Hat Enterprise Linux (RHEL) clone
  - Installed on all CalcUA clusters: Vaughan, Leibniz and Breniac
    - *all clusters are kept in sync as much as possible*
- Resource management and job scheduler: **Slurm**
- Software build and installation framework: **EasyBuild**
- Environment modules system: **Lmod**



# Development software

- C/C++/Fortran compilers
  - Intel oneAPI, AOCC, GCC
  - with OpenMP support
- Message passing libraries
  - Intel MPI, Open MPI
- Mathematical libraries
  - Intel MKL, OpenBLAS, FFTW, MUMPS, GSL, ...
- File formats and data partitioning
  - HDF5, NetCDF, Metis, ...
- Scripting and programming languages
  - Python, Perl, R, ...

# Application software



- Quantum Chemistry / Computational Chemistry / Electronic Structure Calculations
  - ABINIT, CP2K, QuantumESPRESSO, VASP, Gaussian, ORCA, NWChem, OpenMX, Siesta
- Molecular Dynamics (MD) and Biomolecular Simulation
  - GROMACS, NAMD, AMBER, LAMMPS, CHARMM, Desmond, Tinker, DL\_POLY
- Computational Fluid Dynamics (CFD) – TELEMAC, OpenFOAM
- Optimization and Operations Research – Gurobi, CPLEX
- Bioinformatics / Computational Biology – BLAST, Bowtie, Guppy, MAFFT, HMMER, Nextflow
- Data Analysis / Statistical Computing / Scientific Computing – MATLAB, R, Python (SciPy/NumPy), Julia
- Machine Learning / AI / Deep Learning Frameworks – TensorFlow, PyTorch, Scikit-learn, ...
- ... — *not limited to the above list*

# Licensed software

## ➤ VSC or campus-wide license

- e.g.: MATLAB, Mathematica, Maple, ...
- restrictions may apply if you don't work at UAntwerp
  - for external institutions (ITG, VITO) and companies

## ➤ Other restricted licenses

- e.g.: VASP, Gaussian, ...
  - typically paid for by research groups (or individual users)
  - sometimes just other license restrictions that must be respected
- access controlled via *group membership*
  - talk to the owner of the license first
  - request group membership via the [VSC account page](#) (“New/Join group”)
  - the group moderator will grant or refuse access

# Software installation and support

- Installed in **/apps/antwerpen**
  - preferably built and installed using **EasyBuild**
  - often multiple versions of the same package
- *Additional software – installed on demand*
  - system requirements should be met – *note: no Windows software*
  - provide building instructions (no rpm/deb packages)
    - is the software supported by EasyBuild?
  - commercial software must have a *cluster-use license*
  - assist in testing – we can't have expertise in all domains
- Limited (compilation) support
  - best effort, no code fixing
  - many packages are tested with only one compiler

# Selecting software

## ➤ Using **modules**

- dynamic software management
- no version conflicts
- automatically loads required dependencies
- sets environment variables
  - generic – \$PATH, \$LD\_LIBRARY\_PATH, ...
  - application-specific – \$PYTHONPATH, ...
  - EasyBuild related – \$EBROOT...

## ➤ Module naming scheme

```
<name of software>/<version>[-<toolchain info>][-<additional info>]
```

- toolchain = bundle of compiler + compatible MPI and math libraries
- additional information: used to distinguish between versions

# Toolchains

- **Toolchain** = bundle of compiler + compatible MPI and math libraries
  - **intel** – Intel & GNU compilers, Intel MPI and MKL libraries
  - **foss** – GNU compilers, Open MPI, OpenBLAS, FFTW, ...
- **Subtoolchains** — not including MPI or mathematical libraries
  - **gfbf** = GCC + FlexiBLAS + FFTW
  - **GCC** = GCCcore + binutils
  - **GCCcore** — GNU compilers only
- **System toolchain** – system compilers (installed as part of the OS)
- Refreshed yearly (actually, twice per year) → 2026.1, 2025a, 2024b, 2024a, 2023b, 2023a
  - offers more recent versions of the components (and of the software built with it)
- 🔗 [Overview of common toolchains](#) (and their component versions)

# CalcUA modules

- Used to group software installed in the same time frame

| <u>CalcUA module</u>       | <u>Software collection</u>                                                        |
|----------------------------|-----------------------------------------------------------------------------------|
| <code>calcua/2025a</code>  | version 2025a of the <i>toolchain compiler</i> modules + software built with them |
| <code>calcua/system</code> | software built with <i>system compilers</i>                                       |
| <code>calcua/x86_64</code> | software installed from <i>binaries</i> (x86_64)                                  |
| <code>calcua/all</code>    | all currently available software (all of the above)                               |

- Currently available versions of the toolchain compiler modules
  - 2026.1, 2025a, 2024a, 2023a: mostly foss, but also intel
- Good practice: *always load a calcua module first!*

# Using modules

- One command for searching, loading and unloading modules: **module**

```
$ module av openfoam
```

Show/search *available* modules

- depends on currently loaded `calcua` module
- *case-insensitive*

```
$ module spider openfoam
```

Show/search all *installed* modules

- also includes *extensions* (e.g., Python packages, ...)

```
$ module spider  
openfoam/11-foss-2023a
```

Display additional information about a specific module

- shows which `calcua` modules provide it

```
$ module load  
OpenFOAM/11-foss-2023a
```

**Load** a *specific version* of a module

- advise: explicitly specify name & version
- *case-sensitive*

```
$ module list
```

List all *loaded* modules (in the current session)

# Best practices for using modules

```
$ module purge
```

Unload *all* modules – start from a clean environment

- removal of a sticky module using `--force`

```
$ module load calcua/2025a
```

Load appropriate *calcua* module first

- makes the modules available (here, from 2025a)

```
$ module load  
OpenFOAM/13-foss-2025a
```

Load the modules you want to use

- advise: explicitly specify name & version

➤ *Advice: do not load modules in your `.bashrc`*

o consider using module collections instead – subcommands: `save`, `savelist`, `describe`, `restore`

🔗 Module system basics

🔗 User's Tour of the Module Command

# Hands-on

- Which software are you going to use?
  - can you find which versions we have?
  - if we do not have it, is it supported by EasyBuild?
    - yes → let us know
    - no → look for instructions & let us know
- Use our advice to load the modules
  - start from a clean environment
  - load an appropriate calcua module
  - load the module you want to use
- Try out saving and restoring a module collection

# Summary — Select software & build your environment



- Installed software is isolated from the rest of the system using **modules**

```
$ module load R/4.5.1-gfbf-2025a
```

- The modules are grouped in different **CalcUA modules**
  - `calcua/all` is loaded by default
- You can gain access to licensed software by requesting group membership in the VSC account page
- You can request additional software

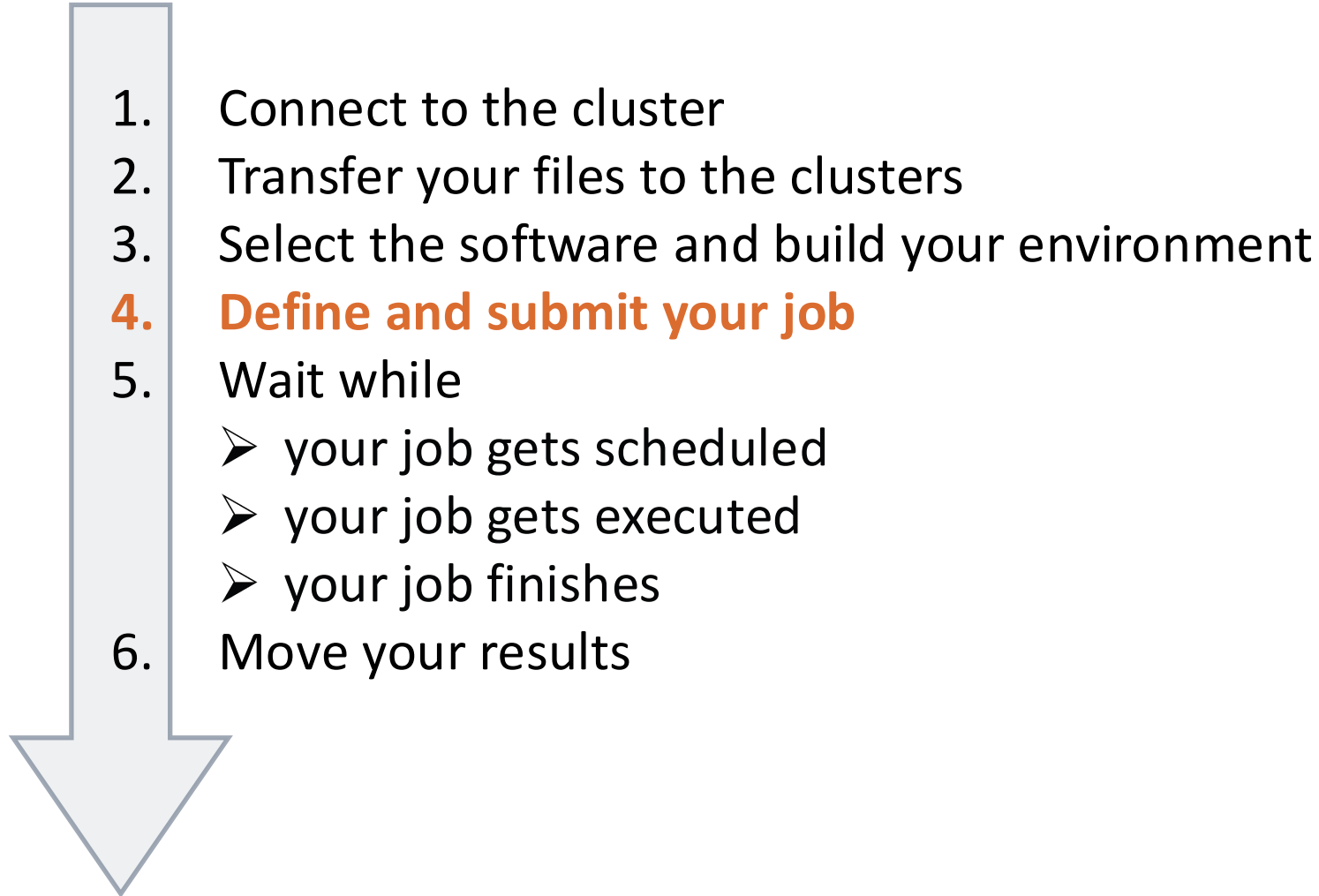


Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

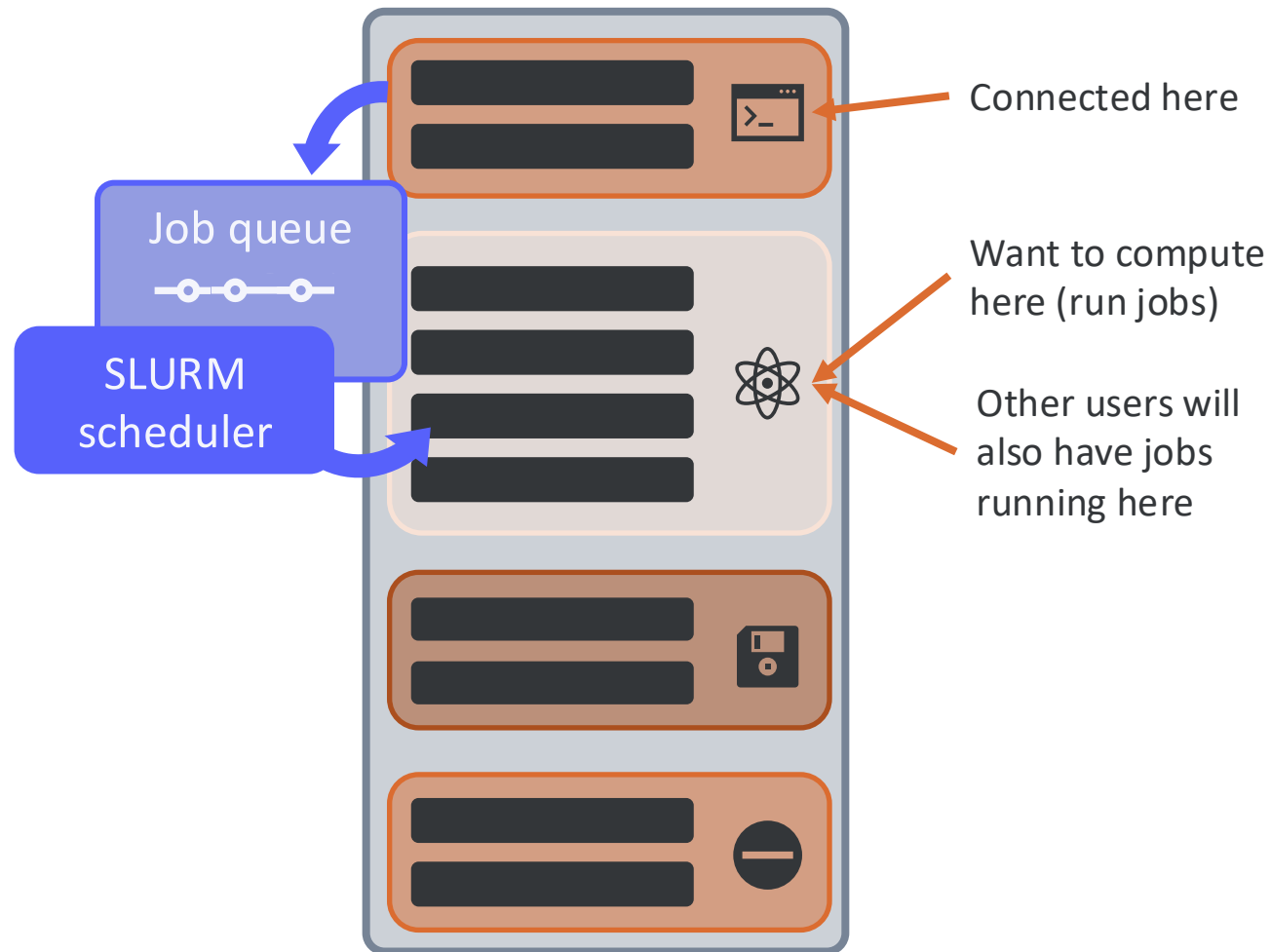
## 5 — Define and submit your job

# A typical workflow

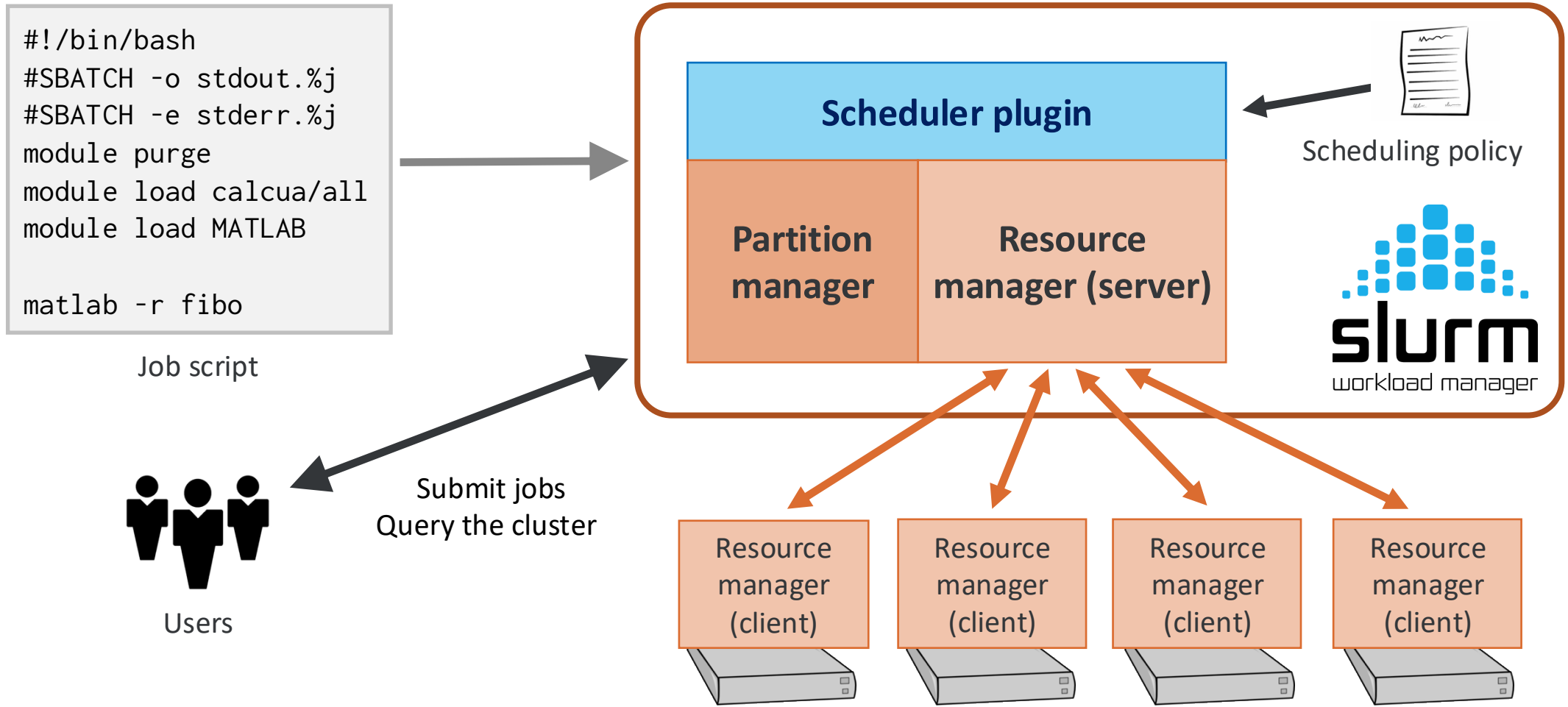


# Running batch jobs

- Running computations → **batch jobs**
  - script with **resource specifications**
- Submitted to a **queueing system**
  - managed by a **resource manager**
- Next job selected by a **scheduler**
  - in a fair way – *fair share*
  - based on available resources
  - & scheduling policies
- Remember:
  - a cluster is a shared infrastructure
  - jobs might not start immediately



# Job submission workflow – *behind the scenes*



# Job script example

- Start with *shebang* line
- Request resources + give instructions
  - first block
  - start with **#SBATCH**
    - these look like comments to bash
- Load relevant modules
  - *build a suitable job environment*
- Actual computation commands

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

```
#SBATCH --ntasks=1 --cpus-per-task=1  
#SBATCH --time=0:10:00  
#SBATCH --account ap_course_hpc_intro  
#SBATCH --partition=zen2  
#SBATCH --output stdout.%j  
#SBATCH --error stderr.%j
```

```
module purge  
module load calcua/2026.1  
module load Python/3.14.2-GCCcore-15.2.0
```

```
python pi.py
```

# Important Slurm concepts

|                  |                                                                                                                                                                                                                                                                                                             |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Node</b>      | Compute node                                                                                                                                                                                                                                                                                                |
| <b>Core</b>      | Physical core (in physical cpu)                                                                                                                                                                                                                                                                             |
| <b>CPU</b>       | Virtual core – <i>hardware thread</i> <ul style="list-style-type: none"><li>• on the CalcUA clusters, hyperthreading is disabled → <b>CPU = Core</b></li></ul>                                                                                                                                              |
| <b>Partition</b> | Group of nodes with job limits and access controls – <i>aka job queue</i>                                                                                                                                                                                                                                   |
| <b>Job</b>       | Submitted work = resource allocation request + commands to execute                                                                                                                                                                                                                                          |
| <b>Job step</b>  | Set of (possibly parallel) tasks within a job <ul style="list-style-type: none"><li>• the job script itself is a special step – <i>the batch job step</i></li><li>• e.g., a MPI application typically runs in its own job step</li></ul>                                                                    |
| <b>Task</b>      | Corresponds to a (single) Linux process, executed in a job step <ul style="list-style-type: none"><li>• a single task can not use more CPUs than available in a single node</li><li>• e.g., for a MPI application, each rank (MPI process) is a task but a shared memory program is a single task</li></ul> |

# Slurm resource requests – Overview

| <u>Long option</u>                   | <u>Short option</u> | <u>Description</u>                   |
|--------------------------------------|---------------------|--------------------------------------|
| <b>--ntasks</b> =<number>            | <b>-n</b> <number>  | Number of tasks                      |
| <b>--cpus-per-task</b> =<ncpus>      | <b>-c</b> <ncpus>   | Number of CPUs per task              |
| <b>--mem-per-cpu</b> =<amount><unit> |                     | Amount of memory per CPU             |
| <b>--time</b> =<time>                | <b>-t</b> <time>    | Time limit (wall time)               |
| <b>--account</b> =<ap_proj>          | <b>-A</b> <ap_proj> | Project account to use               |
| <b>--partition</b> =<pname>          | <b>-p</b> <pname>   | Partition to submit to               |
| <b>--switches</b> =<count>           |                     | Max count of leaf switches           |
| <b>--job-name</b> =<jobname>         | <b>-J</b> <jobname> | Name of the job                      |
| <b>--output</b> =<outfile>           | <b>-o</b> <outfile> | Redirect stdout                      |
| <b>--error</b> =<errfile>            | <b>-e</b> <errfile> | Redirect stderr                      |
| <b>--mail-type</b> =<type>           |                     | Event notification (start, end, ...) |
| <b>--mail-user</b> =<email>          |                     | Email address                        |

# Slurm resource requests – Tasks & CPUs per task

| <u>Long option</u>                         | <u>Short option</u>            | <u>Job environment variable</u>           | <u>Description</u>      |
|--------------------------------------------|--------------------------------|-------------------------------------------|-------------------------|
| <code>--ntasks=&lt;number&gt;</code>       | <code>-n &lt;number&gt;</code> | <code>SLURM_NTASKS</code> (if set)        | Number of tasks         |
| <code>--cpus-per-task=&lt;ncpus&gt;</code> | <code>-c &lt;ncpus&gt;</code>  | <code>SLURM_CPUS_PER_TASK</code> (if set) | Number of CPUs per task |

- Specify number of (parallel) tasks and CPUs (cores) per task
  - o **Task** = single process (runs within a single node)
  - o **CPUs per task** → number of computational threads for a task
- *Note: CPUs per task can never exceed the number of cores per node*
- If not set, **default = 1 task & 1 CPU**

# Slurm resource requests – Memory per CPU

| <u>Long option</u>                                    | <u>Job environment variable</u>               | <u>Description</u>       |
|-------------------------------------------------------|-----------------------------------------------|--------------------------|
| <code>--mem-per-cpu=&lt;amount&gt;&lt;unit&gt;</code> | <code>SLURM_MEM_PER_CPU</code> (in megabytes) | Amount of memory per CPU |

- **Memory per CPU** – not per task
  - **unit** = kilobytes (k), megabytes (m) or gigabytes (g)
  - **amount** = integer — 3.75g is invalid, use 3840m instead
- If not set, **default = maximum available memory per requested CPU**
  - depends on node or partition setting
- *Note: if requesting more than maximum available per CPU → number of CPUs will be increased*
- *Note: on CalcUA clusters, per node 16 GB is reserved for the OS and file system buffers*
  - e.g., on a Vaughan compute node with 256 GB of (installed) memory, the default value is 3840m
    - calculated from  $(256 \text{ GB} - 16 \text{ GB}) / 64 \text{ CPUs} = 240 / 64 = 3.75\text{GB} = 3840 \text{ MB (per core)}$

# Slurm resource requests – Wall time

| <u>Long option</u>               | <u>Short option</u>          | <u>Job environment variable</u>            | <u>Description</u>            |
|----------------------------------|------------------------------|--------------------------------------------|-------------------------------|
| <code>--time=&lt;time&gt;</code> | <code>-t &lt;time&gt;</code> | SLURM_JOB_START_TIME<br>SLURM_JOB_END_TIME | Time limit = <b>wall time</b> |

- Formats :      `mm` | `mm:ss` | `hh:mm:ss` | `d-hh` | `d-hh:mm` | `d-hh:mm:ss`
  - d = days, hh = hours, mm = minutes, ss = seconds
- **Maximum time limit** on the CalcUA clusters
  - compute nodes: 3 days (Vaughan, Leibniz), 7 days (Breniac)
  - GPU nodes: 1 day
- *Wall time exceeded → job will be killed*
- *Wall time > maximum → job will not start*
- If not set, **default = 1 hour**

# Slurm resource requests – Project account

| <u>Long option</u>                     | <u>Short option</u>             | <u>Job environment variable</u> | <u>Description</u>     |
|----------------------------------------|---------------------------------|---------------------------------|------------------------|
| <code>--account=&lt;ap_proj&gt;</code> | <code>-A &lt;ap_proj&gt;</code> | <code>SLURM_JOB_ACCOUNT</code>  | Project account to use |

- Required to specify a **project account** at CalcUA clusters
  - o accounting for both compute (jobs) and storage (files)
  - o *ask your supervisor or project account manager to get access*
  - o use an appropriate account according to the project
- Show accounts you have access to: `myprojectaccounts`
  - o all project accounts start with `ap_`
  - o *during this course* → `ap_course_hpc_intro`

🔗 [UAntwerp Accounting](#)

# Slurm resource requests – Partitions

| <u>Long option</u>                     | <u>Short option</u>           | <u>Job environment variable</u>  | <u>Description</u>     |
|----------------------------------------|-------------------------------|----------------------------------|------------------------|
| <code>--partition=&lt;pname&gt;</code> | <code>-p &lt;pname&gt;</code> | <code>SLURM_JOB_PARTITION</code> | Partition to submit to |

- **Partition** = group of nodes
  - access controls and scheduling policies — e.g.: restrict access to a limited group of users
  - job defaults & resource limits – e.g.: def/max mem per CPU, max time limit, def CPUS per GPU
- If not set, use the **default partition defined per cluster**
  - *note: job does not get automatically assigned to the optimal partition*
- ☑ [UAntwerp Tier-2 Infrastructure](#) – available partitions per cluster + resource limits

# CalcUA clusters – Partitions and node information

| <u>Cluster</u> | <u>Partition</u> | <u>#</u> | <u>Specifications</u>         | <u>CPU – GPU</u> | <u>Mem per CPU</u> | <u>Max WT</u> |
|----------------|------------------|----------|-------------------------------|------------------|--------------------|---------------|
| Vaughan        | <b>zen2</b>      | 152      | AMD Zen 2, 256 GB RAM         | 64 CPU           | 3.75 GiB – 3840m   | 3 days        |
|                | zen3             | 28       | AMD Zen 3, 256 GB RAM         | 64 CPU           | 3.75 GiB – 3840m   |               |
|                | zen3_512         | 12       | AMD Zen 3, 512 GB RAM         | 64 CPU           | 7.75 GiB – 7936m   |               |
|                | ampere_gpu       | 1        | Zen 2, NVIDIA Ampere GPUs     | 4 GPU – 64 CPU   | 3.75 GiB – 3840m   | 1 day         |
|                | arcturus_gpu     | 2        | Zen 2, AMD Arcturus GPUs      | 2 GPU – 64 CPU   | 3.75 GiB – 3840m   |               |
| Leibniz        | <b>broadwell</b> | 144      | Intel Broadwell, 128 GB RAM   | 28 CPU           | 4 GiB – 4096m      | 3 days        |
|                | broadwell_256    | 8        | Intel Broadwell, 256 GB RAM   | 28 CPU           | 8,5 GiB – 8704m    |               |
|                | pascal_gpu       | 2        | Broadwell, NVIDIA Pascal GPUs | 2 GPU – 28 CPU   | 4 GiB – 4096m      | 1 day         |
| Breniac        | <b>skylake</b>   | 23       | Intel Skylake, 192 GB RAM     | 28 CPU           | 6.29 GiB – 6436m   | 7 days        |

➤ **bold** = default partition for the corresponding cluster

# Hands-on

➤ And now it's **time to run your first job** – *finally!*

- *start by cloning our repository for this course*

**git clone** <https://github.com/hpcuantwerpen/intro-hpc>

➤ Create a small job script which

- uses the correct project account – *for this course*
- needs 1 core, has a wall time of 10 minutes, and will run on the zen2 partition
- loads the module vsc-tutorial/202510-intel-2025a – *according to our advice*
- executes a “hello world” script by using the command: serial\_hello

➤ Submit your first job

- submit the job – use **sbatch** → you get a *job id*
- be patient, the job will start soon – check the job status using **squeue**
- look at what happens – e.g.: which files are generated?

# Slurm resource requests – Faster communication

| <u>Long option</u>        | <u>Description</u>                                   |
|---------------------------|------------------------------------------------------|
| <code>--switches=1</code> | Request all nodes to be connected to a single switch |

- Node communication through network switches
  - Nodes are grouped on *edge switches* which are connected by *top switches*
    - hence communication/traffic between two nodes passes through either 1 or 3 switches
- Some programs are *latency-sensitive* – e.g.: GROMACS
  - will run much better on nodes which are all connected to a single (edge) switch
- *Note: using this option might increase your waiting time*

# Slurm resource requests – Exclusive node access

| <u>Long option</u> | <u>Description</u>                               |
|--------------------|--------------------------------------------------|
| <b>--exclusive</b> | Request exclusive access to the node for the job |

- Nodes are **shared resources**
  - *if you don't request all cores, remaining cores might be used by another user*
  - *if you submit multiple jobs, those might end up on the same or on different nodes*
- Sometimes it is better to request **exclusive access to the compute nodes**
  - *because jobs influence each other (L3 cache, memory bandwidth, communication channels, ....)*
  - *prevents sharing of allocated nodes with other jobs – even from the same user*
- *Be aware, you will be charged for a full node*

# Slurm resource requests – Number of nodes

| <u>Long option</u>                  | <u>Short option</u>            | <u>Job environment variable</u>  | <u>Description</u> |
|-------------------------------------|--------------------------------|----------------------------------|--------------------|
| <code>--nodes=&lt;number&gt;</code> | <code>-N &lt;number&gt;</code> | <code>SLURM_JOB_NUM_NODES</code> | Number of nodes    |

- For each task, all of the CPUs for that task are allocated on a single compute node
  - but different (parallel) tasks might end up on either the *same* or *different* compute nodes
  - depends on what is already running on these nodes — from you or another user
- Advice: *bundle tasks from the same job on as few nodes as possible*
  - to make the communication latency between tasks as small as possible
- Specify the number of nodes the job may use
  - tell the scheduler how many nodes it needs to allocate for the job
  - note: also possible to specify a min/max number of nodes using `--nodes=<min>-<max>`

# Non-resource-related options – Job name

| <u>Long option</u>                      | <u>Short option</u>             | <u>Job environment variable</u> | <u>Description</u> |
|-----------------------------------------|---------------------------------|---------------------------------|--------------------|
| <code>--job-name=&lt;jobname&gt;</code> | <code>-J &lt;jobname&gt;</code> | <code>SLURM_JOB_NAME</code>     | Name of the job    |

- Assign a name to your job – the *job name*
  - job name can be used when defining the output and error files
- If not given, the **default name = name of the batch job script**
  - or “sbatch” if read from standard input

# Non-resource-related options – Redirect stdout / stderr

| <u>Long option</u>         | <u>Short option</u> | <u>Description</u> |
|----------------------------|---------------------|--------------------|
| <b>--output</b> =<outfile> | <b>-o</b> <outfile> | Redirect stdout    |
| <b>--error</b> =<errfile>  | <b>-e</b> <errfile> | Redirect stderr    |

- By **default** = **redirect both stdout and stderr** → **slurm-<jobid>.out**
  - that file is present as soon as the job starts and produces output
  - but delays may occur due to buffering or filesystem caches
- If only **--output** is given → redirect *both* stdout and *stderr to the same file*
- Possible to use *filename patterns* to define the filename
  - examples: **%x** for the job name, **%j** for job id, ...

🔗 [Filename patterns](#)

# Non-resource-related options – Mail notifications

| <u>Long option</u>                     | <u>Description</u>                   |
|----------------------------------------|--------------------------------------|
| <code>--mail-type=&lt;type&gt;</code>  | Event notification (start, end, ...) |
| <code>--mail-user=&lt;email&gt;</code> | Email address                        |

- The scheduler can send you a mail when a job begins (starts), ends or fails (gets aborted)
  - type = BEGIN | END | FAIL | ALL | TIME\_LIMIT\_xx
- **default email address** = linked to your VSC-account

# The job runtime environment

- On UAntwerp clusters, we only set a **minimal environment** for jobs by default

- equivalent to exporting only these environment variables

```
--export=HOME,USER,TERM,PATH=/bin:/sbin
```

- *hence you need to (re)build a suitable environment for your job – using modules*
- Other available environment variables include
  - **VSC\_\*** — for user directories, but also for cluster/os/architecture
  - **EB\*** + module specific variables – defined by loading modules
  - **SLURM\_\*** variables – set by Slurm (next slide)

↗ [The job environment](#)

# The job runtime environment

- Slurm defines several variables when a job is started
  - these can be used when calling programs – e.g.: to pass the number of available CPUs
  - *some are only present if explicitly set*

| <u>Environment variable</u> | <u>Explanation</u>                          |
|-----------------------------|---------------------------------------------|
| SLURM_SUBMIT_DIR            | The directory from which sbatch was invoked |
| SLURM_JOB_ACCOUNT           | Account name selected for the job           |
| SLURM_JOB_NUM_NODES         | Total number of nodes for the job           |
| SLURM_JOB_NODELIST          | List of nodes allocated to the job          |
| SLURM_JOB_CPUS_PER_NODE     | CPUs available to the job on this node      |
| SLURM_TASKS_PER_NODE        | Number of tasks to run on this node         |

↗ Output environment variables

# Summary — Define & submit your job

jobscript.sh

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --time=0:10:00
#SBATCH --account ap_course_hpc_intro
#SBATCH --output stdout.%j
#SBATCH --error stderr.%j
module purge
module load calcua/2023a
module load R/4.3.2-gfbbf-2023a
...
```

- Computations (batch jobs) are scheduled by **Slurm** to run on the compute nodes
- In your job script, you **request** the appropriate **resources**:
  - number of cores/ tasks,
  - memory,
  - partition,
  - **project account**,
  - wall time, ...



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

X — Using the command line

# SSH access — Required software

## ➤ Windows

- SSH client included in latest versions of Windows 10 or above
  - check if present in Windows Settings > System > Optional features
- optional: use [Windows Subsystem for Linux \(WSL\)](#)
  - install and use a Linux distribution of your choice
  - now also supports running Linux GUI apps (X11 and Wayland)
- optional: use [Windows Terminal](#) (available via the Microsoft Store)
  - choose between Command Prompt, PowerShell, and bash (via WSL)
- [PuTTY](#) used to be a popular GUI SSH client

# SSH access — Required software

## ➤ macOS

- SSH client included
- Terminal app (built-in) or [iTerm2](#)
- for graphical applications (X11), use [XQuartz](#)
- optional: [Homebrew](#)
  - allows to install Linux commands
  - can also install applications
- remark: macOS is based on BSD (Unix)
  - (BSD variants of) commands may behave differently

## ➤ Linux

- SSH client included
- choice of terminal and shell
- supports graphical applications

# SSH access — Public/private key pairs

- Communication with the cluster happens through **SSH** (Secure SHell)
  - Protocol to log in to a remote computer, transfer files (**SFTP**), ...
  - uses **public/private key pairs**



# SSH access — Public/private key pairs

## 🔗 Create a public/private key pair

- o create RSA key pair (at least 4096 bits)

```
$ ssh-keygen -t rsa -b 4096
```

- o note: on Windows, when using PuTTYgen key generator
  - use PuTTY key format 2 in latest version
  - Convert the public key to OpenSSH format

## 🔗 Upload public key → VSC account page

- o web-based registration procedure

# SSH access — Connect to the cluster

## ➤ You need:

- VSC account name: **vsc2xxxx**
- Hostname of a login node
- Private/public key pair  
(upload public key once)

## ➤ Restricted public access

- outside of Belgium: use **VPN**
  - [vpn.uantwerpen.be](https://vpn.uantwerpen.be)
  - Instructions on Pintra (staff)

My Subsites > Department ICT

> ICT Guide > Remote working - VPN

or Studentportal (students)

Dashboard > ICT > Network > VPN

## Cluster

## Hostname of login node

Vaughan

login-vaughan.hpc.uantwerpen.be

Vaughan  
(indiv. login nodes)

login1-vaughan.hpc.uantwerpen.be  
login2-vaughan.hpc.uantwerpen.be

Leibniz

login-leibniz.hpc.uantwerpen.be  
**login.hpc.uantwerpen.be**

Leibniz  
(indiv. login nodes)

login1-leibniz.hpc.uantwerpen.be  
login2-leibniz.hpc.uantwerpen.be

Leibniz (vis. node)

viz1-leibniz.hpc.uantwerpen.be

Breniac

login-breniac.hpc.uantwerpen.be

# SSH access — Connect to the cluster

## ➤ Login via secure shell

- if your private key has the standard filename (`~/.ssh/id_rsa`)

```
$ ssh vsc2xxxx@login.hpc.uantwerpen.be
```

- otherwise, explicitly specify the filename

```
$ ssh -i ~/.ssh/id_rsa_vsc vsc2xxxx@login.hpc.uantwerpen.be
```

↗ [Text-mode access using OpenSSH](#)

# SSH access — Using an SSH configuration file

```
Host *
```

```
    ServerAliveInterval 60
```

```
Match final User vsc2xxxx
```

```
    IdentityFile ~/.ssh/id_rsa_vsc
```

```
Host calcua
```

```
    User vsc2xxxx
```

```
    HostName login.hpc.uantwerpen.be
```

```
    ForwardAgent yes
```

```
    ForwardX11 yes
```



for all hosts

- (try to) keep the connection alive



when connecting as user vsc2xxxx

- use this private key



create a shorthand “calcua”

- connect as user vsc2xxxx
- use login node login.hpc.uantwerpen.be
- use agent forwarding (for subsequent ssh calls (-A))
- use X11 forwarding (for visualisation (-X))

➤ Save this file as ~/.ssh/config and then you can connect using: `ssh calcua`

🔗 [SSH config](#)

# SCP/SFTP access — Transfer your files

➤ For simple file transfers: secure copy (**SCP**)

- copy from your local computer to the cluster

```
$ scp file.ext vsc2xxxx@login.hpc.uantwerpen.be:
```

- copy from the cluster to your local computer

```
$ scp vsc2xxxx@login.hpc.uantwerpen.be:file.ext .
```

➤ Need more features (e.g.: file browsing, resuming transfers, ...): use **SFTP**

- command-line: **sftp**
- any graphical sftp file manager of your choice

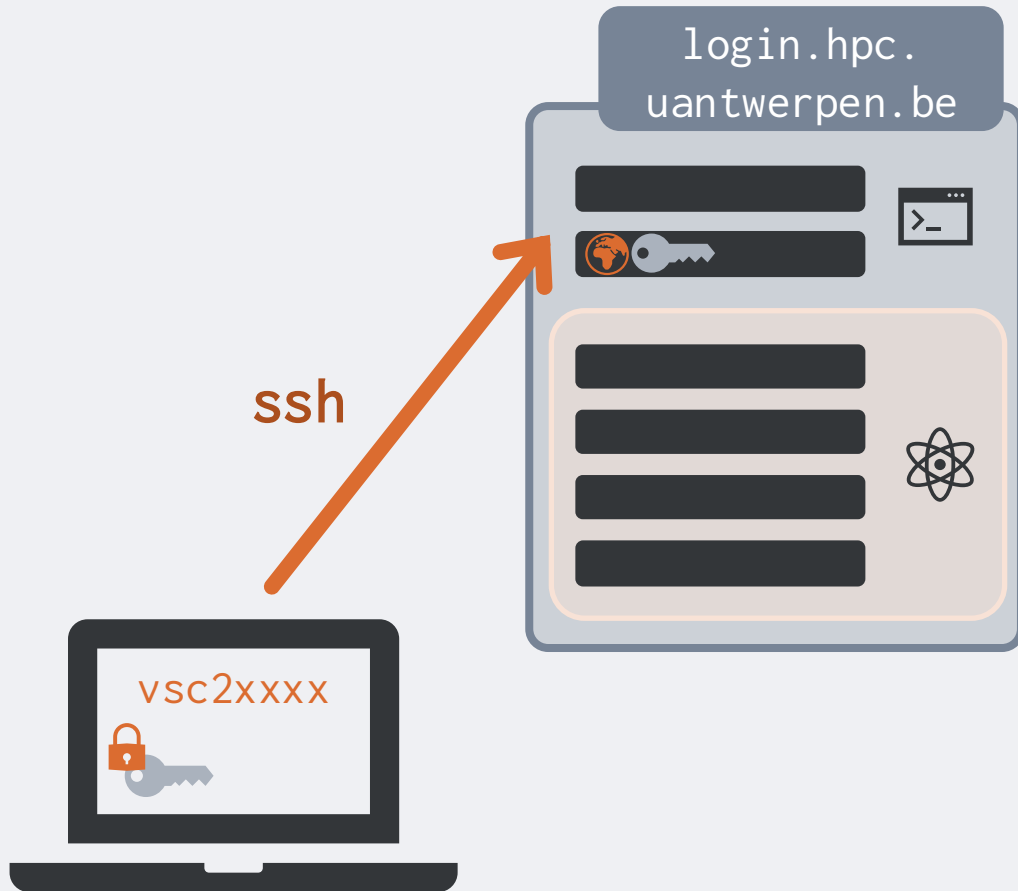
➤ For large file transfers, we recommend **Globus**

🔗 [Data transfer on external computers](#)

# Hands-on

- Install the required software
- Create your VSC account
  - create a public/private key pair
  - upload your public key
- Login to a CalcUA cluster via **ssh**
- Copy a file from CalcUA to your computer using **scp**
- Create a SSH configuration file
  - *feel free to choose your own shorthand name*
  - login using the shorthand name

# Summary — SSH access



- You can connect directly to the cluster's **login nodes**, not the compute nodes
- To connect using **ssh**, you need:
  - your **VSC account** name
  - the **hostname** of the login node
  - your **private key**
- Your private key is private, keep it safe!

# Globus – Command line interface

## Setup

- `$ module load Globus-CLI/3.34.0-GCCcore-13.3.0`
- `$ globus login`
  - paste url in browser, follow instructions & copy code back to terminal
- `$ globus endpoint search "VSC UAntwerpen Tier2 filesystems"`
- `$ Tier2Antwerp=6a13242d-6506-4b3d-a49c-ac981b35ab7d`
  - you can add these as environment variables to `~/.bashrc`
- `$ globus ls $Tier2Antwerp`
  - first time access error: "Missing required data\_access consent"
  - `$ globus session consent "urn:globus:auth:scope:transfer.api.globus.org:all [*https://auth.globus.org/scopes/<COLLECTION_ID>/data_access]"`

# Globus – Command line interface

## Transfer

- `$ module load Globus-CLI/3.34.0-GCCcore-13.3.0`
- `$ globus endpoint search --filter-scope my-endpoints <name>`
  - `$ Tier2Antwerp=6a13242d-6506-4b3d-a49c-ac981b35ab7d`
  - `$ MyPC=...`
- `$ globus ls $Tier2Antwerp:/scratch/antwerpen/xxx/vsc2xxyy/`
- `$ globus transfer $Tier2Antwerp:path/to/data/on/uantwerp/tier2 \`  
`$MyPC:path/to/data/on/your/pc/ --label "test_transfer"`

🔗 [Globus Command Line Interface \(CLI\) — Reference](#)

# OneDrive – Command line interface

## 🔗 OneDrive Client for Linux

- o alternative: UAntwerp OneDrive connector for Globus — *not yet available*

## Setup

### ➤ \$ onedrive

- o copy url in browser, follow instructions, copy response url back to terminal
- o use Chrome if you encounter issues

### ➤ Configure a directory to be synchronized:

- o \$ mkdir -p \$VSC\_DATA/onedrive/calcula-sync
- o \$ config=~/.config/onedrive/config
- o \$ echo sync\_dir = \"\$VSC\_DATA/onedrive\" > \$config # set onedrive directory
- o \$ echo 'skip\_symlinks = "true"' >> \$config # do not sync symlinks
- o \$ echo 'skip\_dotfiles = "true"' >> \$config # do not sync hidden files

# OneDrive – Command line interface

- You don't want to sync your entire OneDrive
  - ~/.config/onedrive/sync\_list defines which files and directories to sync
    - o \$ echo calcua-sync >> ~/.config/onedrive/sync\_list
    - o supports excluding and/or including files and/or directories (see example)
- After changing sync\_list:
  - o \$ onedrive --resync --sync --verbose --dry-run

## Transfer

- Perform a synchronization:
  - o \$ onedrive --sync --verbose

🔗 [Using the client](#)

~/.config/onedrive/sync\_list

```
# exclusions should be listed first
!/Documents/secret/ # exclude this directory, otherwise
included in next line
/Documents/ # include Documents folder in OneDrive root
Results/ # include Results folder anywhere in OneDrive
Reports/*.pdf # include all .pdf files in a Reports
folder anywhere in OneDrive
```

## Part 2 — Sneak preview

- In **Part 1**, you learned how to
  - connect to the cluster and transfer your files
  - use modules and setup your job environment
  - properly specify your resource requests
  - write and submit your job scripts
- In **Part 2**, you will learn
  - more about the Slurm commands and how to use them
  - the different types of multi-core parallel jobs
  - how to organize your job workflows
  - running large number of jobs
  - and some best practices



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

Ine Arts, Franky Backeljauw, Michele Pugno, Robin Verschoren

Version Fall 2026 — Part 2

# Table of contents – Part 2

## 6. Slurm commands

- sbatch : submit a batch script
- squeue : check the status of your jobs
- scancel : cancel a job
- srun : parallel tasks launcher
- salloc : create a resource allocation
- sstat and sacct : information about jobs
- sinfo : get an overview of the cluster and partitions
- scontrol : view Slurm configuration and state

## 7. Multi-core parallel jobs

- Why parallel computing?
- Types of parallel computing
- Running a shared memory job – Multithreading (OpenMP)
- Running a distributed memory job – MPI
- Running a hybrid OpenMP/MPI job
- Job monitoring

## 8. Multi-job submission

- Running a large batch of small jobs
- Job arrays and atools

## 9. Best practices

- Writing job scripts
- Resource efficiency
- Heavy I/O
- Alternatives to Conda

## X. Extra topics

- Installing your own software
- Checkpointing
- Interactive applications
- Organizing job workflows

## Final notes



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 6 — Slurm commands

# Slurm commands – Overview

|                                                                                     | <u>Command</u>        | <u>Description</u>                          |
|-------------------------------------------------------------------------------------|-----------------------|---------------------------------------------|
|    | <code>sbatch</code>   | Submit a batch script                       |
|    | <code>squeue</code>   | Check the status of your jobs               |
|    | <code>scancel</code>  | Cancel a job                                |
|    | <code>srun</code>     | Parallel tasks launcher                     |
|    | <code>salloc</code>   | Create a resource allocation                |
|    | <code>sstat</code>    | Show information about <i>running</i> jobs  |
|   | <code>sacct</code>    | Show information about (terminated) jobs    |
|  | <code>sinfo</code>    | Get an overview of the partitions and nodes |
|  | <code>scontrol</code> | View current Slurm configuration and state  |

# sbatch — Submit a batch script



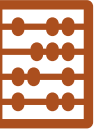
- `sbatch` *<sbatch arguments>* `jobscript` *<arguments of the job script>*
  - submits a job script *for later execution* — *batch job = queued, non-interactive*
  - does not wait for the job to start or end
  - can also read the job script from stdin instead
- What `sbatch` does:
  - submits the job script to the selected partition — *aka job queue*
  - returns Submitted batch job *<jobid>*
- What Slurm does — *behind the scenes*
  - creates an **allocation** when resources become available
  - creates **batch job step** in which it runs the job script
- *Note: “batch script” and “job script” are used interchangeably*

# sbatch — Submit a batch script

- To pass resource (and non-resource) requests
  - add **#SBATCH comment lines** at the beginning of your job scripts
  - use **environment variables** beginning with **SBATCH\_**
    - followed by the name of the matching command line option
    - can be useful if you have access to only one project account
    - overrules **#SBATCH** lines
  - on the command line as **options** to **sbatch**
    - overrules both **#SBATCH** and **SBATCH\_\***

🔗 [sbatch manual page](#)

# queue — Check the status of your jobs



- **queue** checks the status of your jobs in the job queue

```
$ queue
```

| JOBID | PARTITION | NAME | USER     | ST | TIME | NODES | NODELIST(REASON) |
|-------|-----------|------|----------|----|------|-------|------------------|
| 26170 | zen2      | bash | vsc20259 | R  | 6:04 | 1     | r1c02cn3.vaughan |

- o **ST** = state of the job

| <u>ST</u> | <u>Explanation</u> |
|-----------|--------------------|
|-----------|--------------------|

|           |                                 |
|-----------|---------------------------------|
| <b>PD</b> | Pending – waiting for resources |
|-----------|---------------------------------|

|           |                                    |
|-----------|------------------------------------|
| <b>CF</b> | Configuring – nodes becoming ready |
|-----------|------------------------------------|

|          |         |
|----------|---------|
| <b>R</b> | Running |
|----------|---------|

|           |                                        |
|-----------|----------------------------------------|
| <b>CD</b> | Successful completion – exit code zero |
|-----------|----------------------------------------|

| <u>ST</u> | <u>Explanation</u> |
|-----------|--------------------|
|-----------|--------------------|

|          |                                 |
|----------|---------------------------------|
| <b>F</b> | Failed job – non-zero exit code |
|----------|---------------------------------|

|           |                              |
|-----------|------------------------------|
| <b>TO</b> | Timeout – wall time exceeded |
|-----------|------------------------------|

|            |                                     |
|------------|-------------------------------------|
| <b>OOM</b> | Job experienced out-of-memory error |
|------------|-------------------------------------|

|           |                                    |
|-----------|------------------------------------|
| <b>NF</b> | Job terminated due to node failure |
|-----------|------------------------------------|

🔗 [queue manual page](#) – [job state codes](#)

# queue — Check the status of your jobs

- **queue** checks the status of your jobs in the job queue

```
$ queue
```

```
  JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
  26170      zen2     bash vsc20259 PD    0:00      1 (Priority)
```

- **NODELIST(REASON)** = reason why a job is waiting for execution

| <u>NODELIST(REASON)</u> | <u>Explanation</u>                                                     |
|-------------------------|------------------------------------------------------------------------|
| Priority                | There are one or more higher priority jobs in the partition            |
| QOSMaxNodePerUserLimit  | The limit on the maximum number of nodes per user is exceeded          |
| AssocMaxJobsLimit       | The limit on the number of running jobs for each user has been reached |
| JobHeldAdmin            | The job is held by an administrator                                    |

🔗 [job reason codes](#)

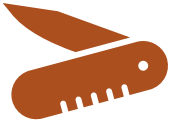
# scancel — Cancel a job



- **scancel** `<jobid>` cancels a job + all its job steps (if already running)
  - cancel a specific job step: **scancel** `<jobid>.<stepid>`
    - *if you suspect a job step hangs, but you still want to execute the remainder of the job script to clean up and move results*
  - cancel a (sub)job of a job array: **scancel** `<jobid>_<arrayid>`
- Some other possibilities
  - **--state** `<state>` or **-t** `<state>` : cancel only jobs with given state
    - `<state>` = pending, running, or suspended
  - **--partition** `<part>` or **-p** `<part>` : cancel only jobs in given partition

🔗 [scancel manual page](#)

# srun — Parallel tasks launcher



- **srun** “Swiss Army Knife” to launch & manage (parallel) tasks within a job
- In *non-interactive* batch job scripts, it is used to
  - start an OpenMP/MPI program — preferred over **mpirun** and **mpiexec**
  - run multiple programs *simultaneously*, each using a *subset* of the allocated resources
- In a (login) shell, it can be used to
  - start a program or *interactive* shell session in a new or existing job (for debugging and testing)
- What Slurm does — *behind the scenes*
  - creates a **job step** that reserves all or a subset of the allocated resources (exclusive access)
  - launches the executable(s) and **manages task distribution and communication** (strict control)
- Usage will become clear in the following sections — 7, 8 and 10

🔗 [srun manual page](#)

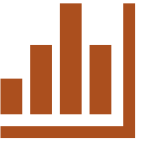
# salloc — Create a resource allocation



- `salloc` creates a resource allocation
- What `salloc` does – *behind the scenes*
  - requests the resources and waits until they are allocated
  - then start a shell on the node where you executed `salloc` – usually the login node
  - afterwards, releases the resources
- *Important: it does **not** create a job step!*
  - the shell is **not** running on (one of) the allocated nodes!
  - but, from the shell, you can start job steps on the allocated resources using `srun`

🔗 [salloc manual page](#)

# sstat — Information about running jobs



- `sstat -j <jobid>[.<stepid>]` shows **real-time information** about a job or job step
  - you can specify a subset of status fields to display using the `-o`, `--format` or `--fields` option
- Get an idea of the **load balancing** (for an MPI job)

```
$ sstat -a -j 12345 -o JobID,MinCPU,AveCPU
```

| JobCPU       | MinCPU    | AveCPU    |
|--------------|-----------|-----------|
| -----        | -----     | -----     |
| 12345.extern | 00:00.000 | 00:00.000 |
| 12345.batch  | 00:00.000 | 00:00.000 |
| 12345.0      | 22:54:20  | 23:03:50  |

- shows the minimum and average amount of *consumed* CPU time for all job steps
  - interpretation: here, step 0 is an MPI job, and we see that the minimum CPU time consumed by the task is close to the average, which indicates that the job is running properly and that the load balance is ok

# sstat — Information about running jobs

## ➤ Checking memory usage

```
$ sstat -a -j 12345 -o JobID,MaxRSS,MaxRSSTask,MaxRSSNode
```

| JobID        | MaxRSS  | MaxRSSTask | MaxRSSNode |
|--------------|---------|------------|------------|
| -----        |         |            |            |
| 12345.extern |         |            |            |
| 12345.batch  | 4768K   | 0          | r1c06cn3.+ |
| 12345.0      | 708492K | 16         | r1c06cn3.+ |

- provides a snapshot of the job's real memory usage – RSS = Resident Set Size
  - gives an insight into how much of the requested memory the job is actively using
  - interpretation: the largest process in the MPI job step is consuming roughly 700MB at this moment, and it is task 16 and running on compute node r1c06cn3.vaughan

## ➤ For a list of possible job status fields, use: `sstat -e`

🔗 [sstat manual page](#)

# sacct — Information about (terminated) jobs



- **sacct** shows information kept in the *job accounting database*
  - o e.g.: job start/end times, resource usage, job status, user/account details, ...
  - o useful for monitoring, billing, performance analysis, ...
  - o note: for running jobs the information may enter only with a delay

```
$ sacct -j 12345 -o JobId,JobName,Partition,State,Elapsed,NodeList
```

| JobID        | JobName    | Partition | State   | Elapsed  | NodeList        |
|--------------|------------|-----------|---------|----------|-----------------|
| 12345        | NAMD-S-00+ | zen2      | RUNNING | 00:57:41 | r4c02cn1.vaugh+ |
| 12345.batch  | batch      |           | RUNNING | 00:57:41 | r4c02cn1.vaugh+ |
| 12345.extern | extern     |           | RUNNING | 00:57:41 | r4c02cn1.vaugh+ |
| 12345.0      | namd2      |           | RUNNING | 00:07:28 | r4c02cn1.vaugh+ |

- For a list of possible job accounting fields, use: **sacct -e**

# sacct — Information about (terminated) jobs

## ➤ Retrieving job details

- o get an overview for jobs in a given time range

```
sacct -S <start-datetime> -E <end-datetime> -X
```

- datetime format: YYYY-MM-DD[THH:MM[:SS]] (other formats possible)

- o get *all* details of a given job — module load Miller

```
sacct -j <jobid> -o ALL -XP | mlr --c2x --ifs='|' cat
```

- o get the batch script of a given job

```
sacct -j <jobid> -B
```

🔗 [sacct manual page](#)

# sinfo — Get an overview of the cluster



- **sinfo** shows information about the **partitions and their nodes** in the cluster

```
$ sinfo
```

| PARTITION  | AVAIL | TIMELIMIT  | NODES | STATE | NODELIST              |
|------------|-------|------------|-------|-------|-----------------------|
| zen2       | up    | 3-00:00:00 | 38    | mix   | r1c01cn1.vaughan, ... |
| zen2       | up    | 3-00:00:00 | 112   | alloc | r1c01cn2.vaughan, ... |
| zen2       | up    | 3-00:00:00 | 1     | idle  | r4c05cn2.vaughan      |
| zen3       | up    | 3-00:00:00 | 24    | idle~ | r6c01cn1.vaughan, ... |
| broadwell  | up    | 3-00:00:00 | 2     | down~ | r2c08cn1.leibniz, ... |
| ampere_gpu | up    | 1-00:00:00 | 1     | idle  | nvam1.vaughan         |

- show number of node is state allocated / mixed / idle / down
- note: ~ = the node is in powersave mode

# sinfo — Get an overview of the cluster

## ➤ Show info *per node*

```
$ sinfo -N -l -n r6c01cn4.vaughan,r1c02cn3.leibniz,amdarc2.vaughan
```

| NODELIST         | NODES | PARTITION    | STATE     | CPUS | S:C:T  | MEMORY |
|------------------|-------|--------------|-----------|------|--------|--------|
| amdarc2.vaughan  | 1     | arcturus_gpu | idle      | 64   | 2:32:1 | 245760 |
| r1c02cn3.leibniz | 1     | broadwell    | allocated | 28   | 2:14:1 | 114688 |
| r6c01cn4.vaughan | 1     | zen3         | idle~     | 64   | 2:32:1 | 245760 |

- **MEMORY** = *total* amount of memory that can be allocated on the node (in kilobytes)
- **S:C:T** = structure of the node → sockets / cores / (hardware) threads

🔗 [sinfo manual page](#)

# scontrol — View Slurm configuration and state



- **scontrol** view Slurm configuration and state
- Show information about:
  - jobs: `scontrol -d show job <jobid>`
    - shows CPU\_IDs of CPUs assigned to the job
  - partitions: `scontrol show part [<part>]`
  - Slurm configuration: `scontrol show config`
- Inside a job script to:
  - get a list of node names one per line: `scontrol show hostnames`
    - `$SLURM_JOB_NODELIST` contains the same list but separated by commas

🔗 [scontrol manual page](#)

# Hands-on

- Given the incomplete job script `matrix.slurm`, which compiles and runs `matrix_multiply.c`
  - make these changes to the job script
    - add the project account to the jobscript – use `ap_course_hpc_intro`
    - request 1 task with 10 cores
    - change the output and error formats to be `<jobname>.<jobid>.out`
      - *remember:* replace `<jobname>` and `<jobid>` by the proper *filename pattern*
    - send yourself an email when the job is finished
    - add a 300 second sleep at the end of the script – so it stays in the queue for a while longer
  - submit the jobscript
    - while the job is running, try several of the Slurm commands – `squeue` / `sstat` / `sacct`
    - what information is stored in the accounting database? – `sacct`
- Clone our repository: `git clone https://github.com/hpcuantwerpen/intro-hpc`

# Summary — Slurm commands



- You can interact with the scheduler/resource manager Slurm through a series of commands
- You cannot go without:
  - **sbatch**: submit a batch script
  - **srun**: *Swiss army knife*
    - start a job step: run parallel tasks
    - start an interactive job
  - **sacct**: show information about (past) jobs



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 7 — Multi-core parallel jobs

# Why parallel computing?

## ➤ Faster time to solution

- distributing code over N cores
- hope for a speedup by a factor of N

## ➤ Larger problem size

- distributing your code over N nodes
- increase the available memory by a factor N
- hope to tackle problems which are N times bigger

## ➤ In practice

- gain limited due to communication, memory overhead, sequential fractions in the code, ...
- optimal number of cores/nodes is problem-dependent
- but no escape possible – computers don't really become faster for *serial* code

## ➤ *Parallel computing is here to stay!*

# Types of parallel computing

## 1. Multithreading

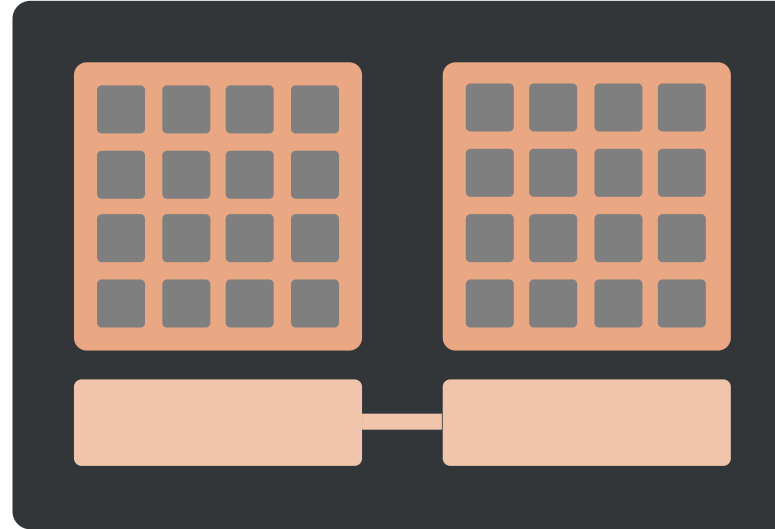
- *shared memory*
- OpenMP

## 2. Multiprocessing

- *distributed memory*
- MPI

## 3. Hybrid

- *combination*

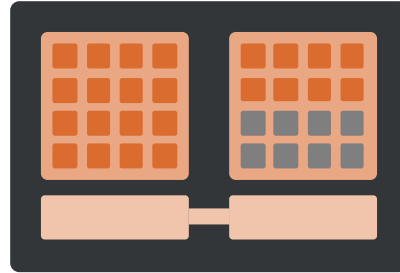


- Compute node
- Processor (socket)
- Core (=CPU)
- Memory

# Types of parallel computing

## 1. Multithreading

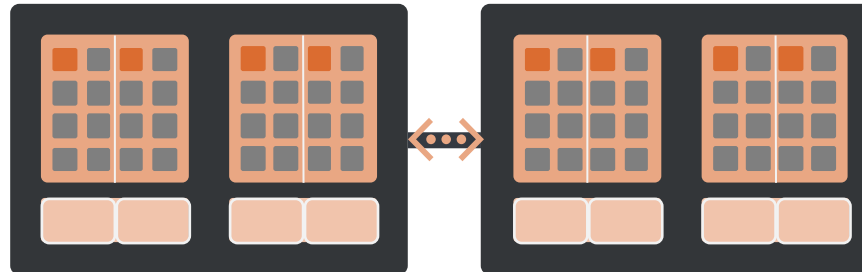
- *shared memory*
- OpenMP



OpenMP software uses multiple or all cores in a **single** node  
*e.g. 24 threads within 1 node*

## 2. Multiprocessing

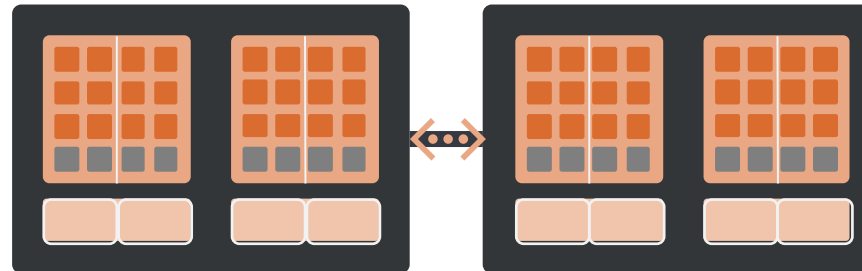
- *distributed memory*
- MPI



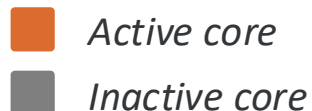
MPI software can use (all) cores in **multiple** nodes  
*e.g. 8 tasks spread over 2 nodes*

## 3. Hybrid

- *combination*

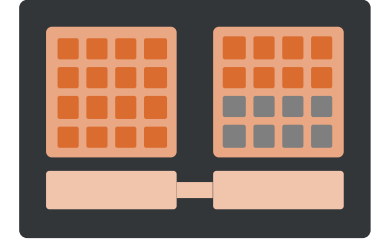


Hybrid OpenMP/MPI software  
*e.g. 6 threads per task  
& 8 tasks over 2 nodes  
(each task stays within 1 node)*



# Running a shared memory job – Multithreading

- **Shared memory job** = *single* task with multiple CPUs per task
  - all threads for the task run on within a *single* node
- Tell the program *how many threads* it can use
  - depends on the program - e.g.: for MATLAB, use `maxNumCompThreads(N)`
    - *note: autodetect usually only works if the program gets the whole node*
  - many OpenMP programs use the environment variable `OMP_NUM_THREADS`
    - Intel OpenMP recognizes Slurm CPU allocations
  - for MKL-based code/operations, use `MKL_NUM_THREADS` – instead of `OMP_NUM_THREADS`
  - for OpenBLAS (FOSS toolchain), use `OPENBLAS_NUM_THREADS`
- *Check the manual of the program you use!*
  - e.g., NumPy has several options (depending on how it was compiled)



# Running a shared memory job – Multithreading

## ➤ OpenMP example script

generic-omp.slurm

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

```
#SBATCH --job-name=OpenMP-demo
```

```
#SBATCH -A ap_course_hpc_intro
```

```
#SBATCH --ntasks=1 --cpus-per-task=64
```

```
#SBATCH --mem-per-cpu=2g
```

```
module --force purge
```

```
module load calcua/2025a
```

```
module load vsc-tutorial/202510-intel-2025a
```

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
```

```
export OMP_PROC_BIND=true
```

```
omp_hello
```

← 1 task with 64 CPUs (so 64 threads)

← 2 GB per CPU, so 128 GB total memory

← load the calcua module

← load vsc-tutorial – also loads the Intel toolchain (for the OpenMP run time)

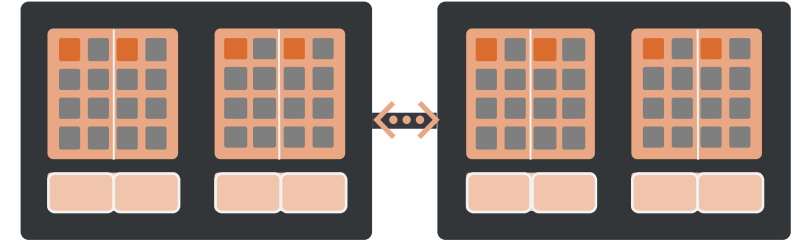
← set the number of (OpenMP) threads to use

← threads stay on the core where they're created

← run the program

# Running a distributed memory job – MPI

- **Distributed memory job** = *several* tasks running in parallel
  - the tasks can be spread over *multiple (different)* nodes
  - communication → **message passing interface (MPI)**
- Every distributed memory program needs a **program starter**
  - some packages use system starter internally
  - **mpirun** works, but depends on variables set in the intel modules
    - ensure to *properly load the module!*
  - the **preferred program starter for Slurm** = **srun**
    - knows how Slurm distributes processes
    - needs no further arguments if resources are correctly requested – *tasks & CPUs per task*
  - *Check the manual of the program you use!*
    - is there an option to explicitly set the program starter?



# Running a distributed memory job – MPI

## ➤ (Intel) MPI example script

generic-mpi.slurm

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

```
#SBATCH --job-name mpihello
```

```
#SBATCH -A ap_course_hpc_intro
```

```
#SBATCH --ntasks=128 --cpus-per-task=1
```

```
#SBATCH --mem-per-cpu=1g
```

← 128 MPI processes (uses 2 nodes on Vaughan, or 5 nodes on Leibniz/Breniac)

```
module --force purge
```

```
module load calcua/2025a
```

← load the calcua module

```
module load vsc-tutorial/202510-intel-2025a
```

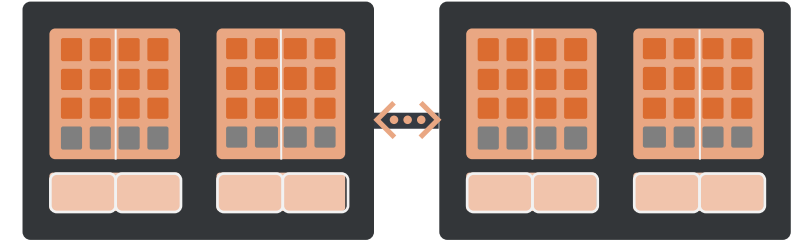
← load vsc-tutorial – also loads the Intel toolchain (for the MPI libraries)

```
srun mpi_hello
```

← run the MPI program – srun communicates with the resource manager

# Running a hybrid OpenMP/MPI job

- **Hybrid job** = combination of **OpenMP** and **MPI**
- No additional tools needed to start hybrid programs
  - **srun** does all the miracle work
    - or **mpirun** in Intel MPI – provided the environment is set up correctly
    - no need for **vsc-mypirun** (still used by some VSC sites)



# Running a hybrid OpenMP/MPI job

generic-hybrid.slurm

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

```
#SBATCH --job-name hybrid_hello
```

```
#SBATCH -A ap_course_hpc_intro
```

```
#SBATCH --ntasks=8 --cpus-per-task=16
```

```
#SBATCH --partition=zen2 --nodes=2
```

```
module --force purge
```

```
module load calcua/2025a
```

```
module load vsc-tutorial/202510-intel-2025a
```

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
```

```
export OMP_PROC_BIND=true
```

```
srun -c $SLURM_CPUS_PER_TASK mpi_omp_hello
```

← 8 MPI processes with 16 threads

← make sure the job uses 2 Vaughan compute nodes (to avoid cluttering)

← load the software stack module

← load vsc-tutorial – also load the Intel toolchain

← set the number of (OpenMP) threads to use

← threads stay on the core where they're created

← run the MPI program (mpi\_omp\_hello)  
srun does all the magic

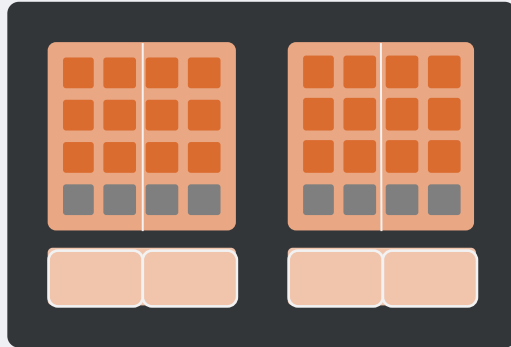
# Job monitoring — Commands for interactive monitoring

- When your job is running
  - how do I know how much memory my job is using?
  - how can I check if my job is running properly, i.e. using the allocated CPUs?
- While your job is running, you can log on to the compute nodes assigned to that job
  - check which compute nodes a job uses: `squeue -j <jobid> -O NodeList: -h`
  - run a shell on one of these nodes — inside the allocation, controlled by Slurm  
`srun --jobid <jobid> -w <compute_node> --overlap --pty bash`
- When logged in on the compute node, check the behaviour
  - **htop** → core & memory usage
  - **sar** → system performance metrics like CPU / memory / disk usage *over time*
  - **vmstat** → monitors system memory / processes / CPU activity / I/O statistics *in real-time*
  - **ps tree** → display a tree view of the running processes

# Hands-on

- Submit the parallel jobs from this section using the provided job scripts
  - a shared memory (OpenMP) job: `prime-omp.slurm`
  - a distributed memory (MPI) job: `prime-mpi.slurm`
  - a hybrid OpenMP/MPI job: `prime-hybrid.slurm`
- While the jobs are running
  - check where the job is running
  - log on to the first node allocated to that job
  - run the job monitoring commands
    - is your job behaving properly?
- When your job finishes
  - check the output files

# Summary — Multi-core parallel jobs



- The key to supercomputing is **parallelization**
- A program can be parallelized in 2 ways:
  - **shared memory**: multithreading
  - **distributed memory**: multiprocessing (needs a program starter like **srun**)
- You should **monitor your computation** to ensure resources are used properly



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 8 — Multi-job submission

# Running a large batch of small jobs

- Scenario: you want to run many, many, many small (short, serial) jobs
  - but: submitting and tracking many short jobs → *burden on scheduler*
- Solutions:
  - **Job arrays**: submit a large number of related yet independent jobs at once
    - to manage array jobs, use atools
  - **srun** can be used to launch more tasks than requested in the job request
    - running no more than the indicated number of tasks simultaneously
  - GNU parallel: tool to easily run shell commands in parallel with different inputs
    - general-purpose tool, can be used in multiple scenarios
- Note: these independent **sub jobs** can also run simultaneously across multiple nodes

# Job arrays

- Starts from a job script for a single (sub) job in the array

job\_array.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=512M
#SBATCH --time 15:00
```

```
INPUT_FILE="input_${SLURM_ARRAY_TASK_ID}.dat"
OUTPUT_FILE="output_${SLURM_ARRAY_TASK_ID}.dat"
```

← for every run, there is a separate input file and an associated output file

```
./calculation ${SLURM_ARRAY_TASK_ID} -input ${INPUT_FILE} -output ${OUTPUT_FILE}
```

- Specify the number of sub jobs in the array

```
sbatch --array 1-100 job_array.slurm
```

- Result: the program will be run for all input files (100)

# Job arrays – atools

## ➤ Features of **atools**

- provides a logging facility and commands to investigate the logs
  - which items failed or did not complete → restart only those
- has limited support for Map-Reduce scenarios
  - preparation phase → split up data in manageable chunks
  - process all chunks in parallel
  - postprocessing phase → combine the results into one file

## ➤ **atools** versus **srun** versus **GNU parallel**

- **atools** uses job arrays, relies on the Slurm scheduler to start all sub jobs
- **srun** uses job steps, sub jobs are started in the allocation of the job — controlled by Slurm
- **parallel** starts the sub jobs itself — not controlled by Slurm

🔗 **atools** – by Geert Jan Bex

# atools example – Parameter exploration

- The **field names** of the header in the CSV file are used as **variables** inside the job script

weather.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --time=10:00
module --force purge
module load calcua/2024a atools/1.5.1

source <(aenv --data data.csv)
./weather -t $temperature -p $pressure -v $volume
```

data.csv

```
temperature, pressure, volume
293.0,      1.0e05,    87
...,       ...,     ...
313,       1.0e05,    75
```

input data in **CSV** format

- Run weather for all data values (from data.csv)

```
module load atools/1.5.1
sbatch --array $(arange --data data.csv) weather.slurm
```

# Summary — Multi-job submission



- Another type of parallelization is *embarrassingly parallel* (or **high throughput**), where all the tasks are independent
- Slurm supports **job arrays** to submit a job script multiple times
- **atools** helps you manage job arrays
  - get parameters from a .csv file
  - rerun failed jobs
  - combine/aggregate data

# Hands-on

## ➤ *Round the table questions:*

- *What type of software will you be running?*
  - *Which scenario applies most to your use case?*
    - will you be running large parallel jobs — make sure your jobs use all the resources
    - or some medium-sized jobs
    - or lots of small jobs — try bundling the jobs whenever possible
  - *How will you be organizing your jobs?*
    - will (most of) your jobs use a similar job script — try using variables and arguments
    - will your jobs depend on each other or are they independent
- *Discuss with your neighbours* and check out the relevant extra examples in `more_examples`



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

## 9 — Best Practices

# Best practices – Table of contents

## Writing job scripts

- first test small, then scale up
- debugging tips

## Resource efficiency

- *is my job wasting money?*
- resource monitoring at runtime & after job has ended

## Heavy I/O

- *why is /scratch so slow?*
- reduce metadata operations on /scratch

## Alternatives to Conda

- *when your software can only be installed with Conda*
- containerize with hpc-container-wrapper

# Writing job scripts

## ➤ First **test small**, then **scale up**

- are required modules correctly loaded?
- does the computation start in the right directory?

## ➤ Some **debugging tips**:

- interactive session: catch errors without having to (re)queue
- shell settings:
  - `set -e` # exit immediately after failed command
  - `set -u` # treat unset variables as an error
- unique output log files
- self-logging

catastrophe.sh

```
...  
cd /nonexistent/path  
rm -rf *  
...
```

jobscript.sh

```
#SBATCH --output=logs/%x_%j.out  
#SBATCH --error=logs/%x_%j.err # %x=JobName, %j=JobID  
...  
echo "Job ID $SLURM_JOB_ID started at: $(date)"  
echo "Running on node: $(hostname)"  
...  
echo "Job finished at: $(date)"
```

# Resource efficiency

*is my job wasting money?*

- Resource monitoring at runtime: `ssh <compute_node>`
  - **CPU:** `htop`
  - **memory:** `htop`
  - **GPU:** `nvidia-smi`
- After the job has ended: using `sacct` (or `sstat`)
  - **CPU:** `sacct -j <jobid> -o JobID,MinCPU,AveCPU`
    - shows min and average CPU time *per task* – CPU time = real time \* (active) CPUs
    - if MinCPU << AveCPU, then compute load is unbalanced
  - **memory:** `sacct -j <jobid> -o JobID,MaxRSS,MaxRSSTask,MaxRSSNode`
    - shows *resident set size* per task + the task and node with which peak memory usage
  - **walltime estimation:** `sacct -j <jobid> -o JobID,Elapsed`

# Heavy I/O

*why is /scratch so slow?*

➤ Parallel filesystems like **/scratch**

- High throughput for (large) files distributed over multiple data servers
- Bottleneck = metadata server

➤ Reduce **metadata** operations on **/scratch**

- ***Know what your software does!***
- Use local filesystem for intensive I/O data
  - **/tmp** – local SSD of the node
  - **/dev/shm** – RAM of the node
- Reduce number of files on **/scratch**
  - Archives & containers

🔗 Best practices: files and I/O

# Alternatives to Conda

*when your software can only be installed with Conda*

- Conda package manager is well liked by scientists but not so much by our shared filesystems
  - 1000's of small files hitting quota or overwhelming metadata server
- Solution: **hpc-container-wrapper** tool
  - uses `environment.yaml` (Conda) or `requirements.txt` (pip) to build a container image
  - provides **wrapper scripts** to transparently call executables within the container
- Create the container with **conda-containerize**
  - \$ `module load hpc-container-wrapper`
  - \$ `conda-containerize new --prefix "$VSC_SCRATCH/container-name" environment.yaml`
- Use your containerized installation
  - \$ `export PATH="$VSC_SCRATCH/container-name/bin:$PATH"`

🔗 Section X – Installing your own software, [getting scientific software installed](#)



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

X — Extra topics

# Extra topics – Table of contents

## Installing your own software

- Packages and environments
- Container wrappers
- Apptainer container

## Checkpointing

- Creation and restart
- Signal trapping

## Interactive applications

- Web portal apps
  - Interactive shell
  - JupyterLab
- Interactive jobs
- Using the visualisation node

## Job workflows

- Examples
- Passing variables to job scripts
- Passing arguments to job scripts
- Job dependencies

# Installing your own software

- Custom software should be installed in your own directory — *preferably in \$VSC\_DATA*
- If the package is supported by **EasyBuild**
  - modify an existing build script — *called "easyconfig"*
  - use our helper script to setup an EasyBuild environment (under \$VSC\_DATA/easybuild)  
\$ `source init-easybuild-user.sh`
- Otherwise: **manually install** the package
  - find the building instructions for the package
  - load a (sub)toolchain module and other modules that provide the libraries you need
  - make sure to set the proper options for the architecture
- Alternative: use **Apptainer containers** instead → *see next slides*

🔗 [EasyBuild documentation](#) and [tutorial](#)

# Installing software – Packages and environments

## ➤ Python (pip), R, Julia, ... packages

- load an appropriate Python, R, Julia, ... module
- point the install prefix to an appropriate directory
  - e.g.: R\_LIBS\_USER, JULIA\_DEPOT\_PATH → in a subdirectory of \$VSC\_DATA
- note: when using **pip**, also change the location of the cache directory `~/.cache` → *file quota!*

## ➤ Conda environments are *discouraged*

- installations involve many small files → *file quota exceeded and metadata server bottleneck*
- typically, they do not use system and software stack libraries – *possible performance issues*
- consume a lot of disk space and put stress on the filesystem
- **alternative** → *wrap the Conda environment in a container*

🔗 Installing packages using pip

🔗 R package management

# Installing software – Container wrappers

- Solution: use **hpc-container-wrapper** – *formerly known as Tykky*
  - tool to **wrap your Python installation into a container**, designed for use on HPC systems
  - uses `environment.yaml` (Conda) or `requirements.txt` (pip) to build a container image
  - provides **wrapper scripts** to transparently call executables within the container environment
    - also provides `wrap-container` to generate wrapper scripts for an existing container
- Create the container with **conda-containerize**
  - \$ `module load hpc-container-wrapper`
  - \$ `conda-containerize new \`  
`--prefix "$VSC_SCRATCH/bsoup" environment.yaml`
- Similar for **pip-containerize**

environment.yaml

```
name: bsoup4
channels:
  - conda-forge
dependencies:
  - beautifulsoup4
  - ...
```

# Installing software – Container wrappers

## ➤ Syntax of `environment.yaml`

- **name:** `my_env`           # name of the conda environment
- **channels:**               # repositories to search
  - `conda-forge`
  - `defaults`
- **dependencies:**
  - `python=3.11`
  - `numpy>=1.24`       *# version constraint*
  - `pandas`           *# latest compatible version*

## ➤ **Generate `environment.yaml`** from existing conda environment

```
$ conda activate my_env
```

```
(my_env) $ conda export --from-history > environment.yaml
```

## ➤ Equivalent for `pip`

```
$ pip freeze > requirements.txt
```

# Installing software – Container wrappers

- Use your containerized Python installation (e.g., from within a job)

```
$ export PATH="$VSC_SCRATCH/bsoup/bin:$PATH"
```

```
$ which python
```

```
$VSC_SCRATCH/bsoup/bin/python
```

```
$ python -c "from bs4 import BeautifulSoup; soup = BeautifulSoup('<p>Hello  
World</p>', 'html.parser'); print(soup.p.text)"
```

```
Hello World
```

- Still missing packages? → **update the container**

```
$ conda-containerize update --post-install  
post.sh "$VSC_SCRATCH/bsoup"
```

post.sh

```
pip install requests  
conda install -c bioconda pyfaidx
```

🔗 [GitHub repository](#) or [documentation](#)

# Installing software – Apptainer container

- **Apptainer** is available to build and run your container images
  - *note: Docker is typically not supported due to security concerns*
- Option: **convert** a (pre-build) Docker *image* to Apptainer

```
$ apptainer pull docker://hello-world:latest
$ apptainer run hello-world_latest.sif
```
- Alternative: **build an image from scratch** using build scripts called *definition files* – similar to a Dockerfile

```
$ export APPTAINER_CACHEDIR=$VSC_SCRATCH/apptainer/cache
$ export APPTAINER_TMPDIR=$(mktemp -d -p /dev/shm)
$ apptainer build ubuntu_fpc.sif ubuntu_fpc.def
$ apptainer exec ubuntu_fpc.sif fpc -h
```

ubuntu\_fpc.def

BootStrap: docker

From: ubuntu:oracular

%post

apt-get update

apt-get install -y fpc

🔗 Can I run containers on the HPC systems?

🔗 Containers for HPC — VSC course by Geert Jan Bex

# Summary — Installing your own software



- You can install your own software:
  - with Easybuild
  - by doing a manual install
  - using packages or environments
- Sometimes **Conda environments** are necessary
  - **hpc-container-wrapper** is strongly recommended
- Use **Apptainer** to build and run container images

# Checkpointing

- When your calculation hits max walltime
  - increase parallelization?
  - increase efficiency?
  - divide job into parts?
  - use different partition?
- Create a **checkpoint** to resume with a subsequent job
  - read your software's manual
    - often there is a built-in stop and restart mechanism
  - **DMTCP** tool
    - saves checkpoint (application state) to disk
    - supports both multithreaded and serial applications

🔗 [DMTCP \(Distributed MultiThreaded CheckPointing\)](#)

🔗 [Job Checkpoints and Restarts](#) — VUB docs

# Checkpointing – Creation and restart

- **Create** checkpoints at time intervals

```
dmctp_launch -i <interval_seconds> --ckptdir /path/to/ckptdir <your_calculation>
```

- **Restart** execution

```
dmctp_restart -i <interval_seconds> /path/to/ckptdir/checkpoint.dmtcp
```

- **.dmtcp checkpoint files**

- size = memory used by job
  - save on \$VSC\_SCRATCH → check if you have enough space
  - 1 per job → overwritten every interval
  - use unique name per job
- What happens to your **calculation files**?
    - include **open files** in checkpoint: **--ckpt-open-files**
    - working on local filesystem? → copy your files to /scratch yourself

# Checkpointing – Signal trapping

- Send signal **X** seconds before job terminates to trigger checkpoint creation
  - o #SBATCH --signal=B:<signal>@<X>

signal\_trapping\_dmtcp.sh

```
...
#SBATCH --signal=B:USR1@300
...
save_ckpt() {
    echo "[$(date)] Walltime approaching. Issuing DMTCP checkpoint..."
    dmtcp_command --ckptdir "${CKPT_DIR}" --bcheckpoint
    echo "[$(date)] Checkpoint saved to ${CKPT_DIR}."
    exit 0
}
trap 'save_ckpt' USR1
...
dmctp_launch -i 0 --ckptdir /path/to/ckptdir <your_calculation>
```

# Summary — Checkpointing



- Pause and restart calculations that take longer than max walltime
- **Create checkpoint**
  - `dmctp_launch -i <interval> --ckptdir /path/to/ckptdir <your_calculation>`
- **Restart execution**
  - `dmctp_restart -i <interval> /path/to/ckptdir/checkpoint.dmtcp`

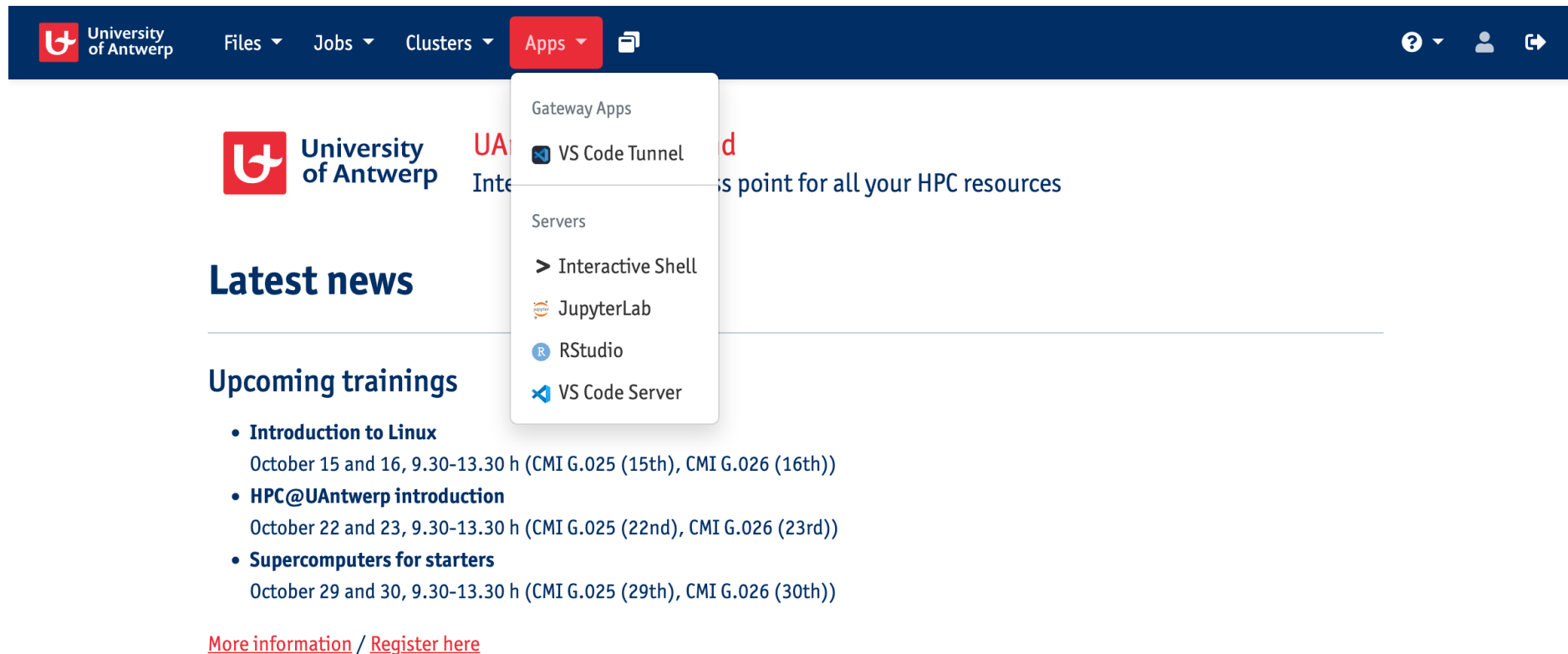
# Interactive applications

- With `sbatch` you run jobs on compute nodes *without user interaction*
  - you need to provide all the input in advance
  - you cannot interact with the script or program while it is running
- Sometimes, you need to interact with a running program
  - when building software on a compute node in an interactive shell
  - when running an interactive Python session in a Jupyter notebook
  - when developing software using a dedicated IDE (integrated development environment)
- You can run interactive applications on the compute nodes
  - using web portal apps
  - with `srun --pty bash -l`
  - or using `ssh`

# Interactive applications – Web portal apps

- Currently available web portal apps — *more apps will be added*
  - Interactive Shell
  - [JupyterLab](#)
  - [RStudio Server](#)
  - [VS Code Server](#) or [VS Code Tunnel](#)
    - *I have VS Code installed on my pc → VS Code Tunnel suggested*
- *Important: interactive apps keep on running if you close the browser window*
  - you need to explicitly cancel the app listed in “My Interactive Sessions”
- 🔗 [VSC Open OnDemand Portals](#) — apps listed in navigation panel on the left

# Interactive applications – Web portal apps



The screenshot shows the University of Antwerp web portal. The top navigation bar is dark blue with the University of Antwerp logo and links for Files, Jobs, Clusters, and Apps. The 'Apps' menu is open, showing options: Gateway Apps, VS Code Tunnel, Servers, Interactive Shell, JupyterLab, RStudio, and VS Code Server. The main content area features the University of Antwerp logo, a heading 'Latest news', and a section 'Upcoming trainings' with three items: 'Introduction to Linux', 'HPC@UAntwerp introduction', and 'Supercomputers for starters'. Each item includes dates and times. At the bottom of the trainings section, there are links for 'More information' and 'Register here'.

University of Antwerp

Files ▾ Jobs ▾ Clusters ▾ Apps ▾

Gateway Apps

VS Code Tunnel

Servers

Interactive Shell

JupyterLab

RStudio

VS Code Server

University of Antwerp

Latest news

Upcoming trainings

- **Introduction to Linux**  
October 15 and 16, 9.30-13.30 h (CMI G.025 (15th), CMI G.026 (16th))
- **HPC@UAntwerp introduction**  
October 22 and 23, 9.30-13.30 h (CMI G.025 (22nd), CMI G.026 (23rd))
- **Supercomputers for starters**  
October 29 and 30, 9.30-13.30 h (CMI G.025 (29th), CMI G.026 (30th))

[More information](#) / [Register here](#)

powered by



OnDemand version: 4.0.7

# Interactive applications – Interactive Shell

 University of Antwerp

Files ▾ Jobs ▾ Clusters ▾ Apps ▾ 

 ▾  

Home / My Interactive Sessions / Interactive Shell

Apps

Gateway Apps

 VS Code Tunnel

Servers

> Interactive Shell

 JupyterLab

 RStudio

 VS Code Server

## Interactive Shell

Launch an interactive shell on the compute nodes.

**Cluster**

leibniz ▾

**Partitions**

broadwell ▾

**Project account**

ap\_course\_hpc\_intro ▾

**Number of cores**

14 ▴ ▾

**Number of nodes**

1 ▴ ▾

# Interactive applications – Interactive Shell



[Home](#) / [My Interactive Sessions](#)

## Apps

### Gateway Apps

VS Code Tunnel

### Servers

Interactive Shell

JupyterLab

RStudio

VS Code Server

## Interactive Shell (2713595)

1 node | 14 cores | Running

**Host:** r3c06cn3.leibniz

Cancel

**Created at:** 2025-10-21 14:17:17 CEST

**Time Remaining:** 54 minutes

**Session ID:** 3dbf10ed-6736-42e6-96df-5882cbe6c1c9

Connect to Shell

- Select & Copy to Clipboard
  - Line: **Shift+Left Click & Drag**
  - Block: **Shift+Alt+Left Click & Drag**
- Paste From Clipboard
  - **Shift+Insert**
  - **Ctrl+Shift+V**

# Interactive applications – Interactive Shell

```
✓ [14:41] vsc20003@r3c06cn3.leibniz $VSC_DATA $ _
```

```
[Job-2713672] 1: bash
```

```
Tue 21/10 14:41
```

# Interactive applications — JupyterLab

 University of Antwerp

Files ▾ Jobs ▾ Clusters ▾ Apps ▾ 

 ▾  

Session was successfully deleted. 

[Home](#) / My Interactive Sessions

Apps

Gateway Apps

 VS Code Tunnel

Servers

 Interactive Shell

 JupyterLab

 RStudio

 VS Code Server

JupyterLab (2713665) 1 node | 8 cores | Running

Host: r2c10cn3.leibniz  Cancel

Created at: 2025-10-21 14:30:20 CEST

Time Remaining: 11 hours and 53 minutes

Session ID: b34efdcc-c21e-4674-b355-d63d87e53582

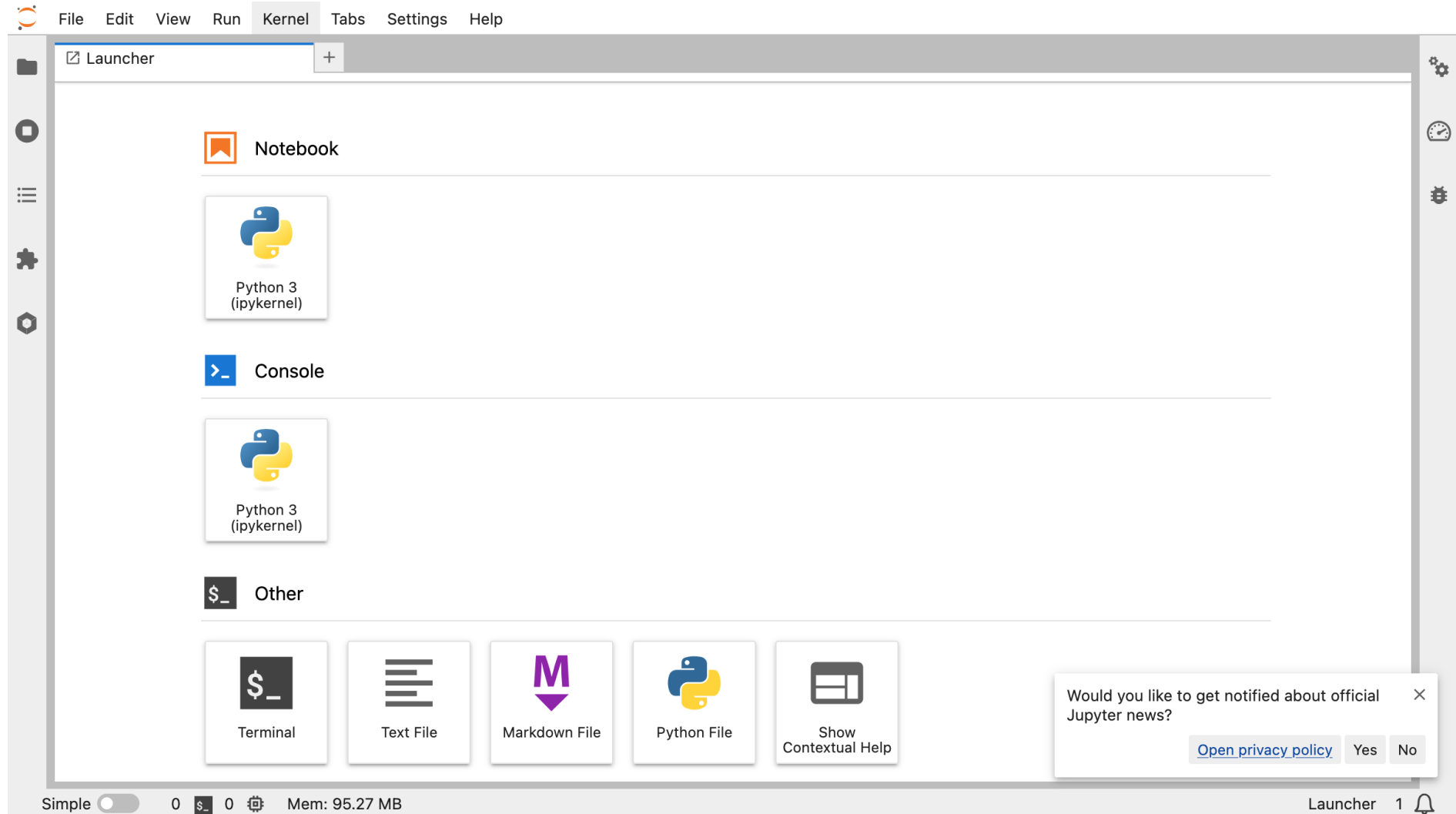
 Connect to JupyterLab

powered by



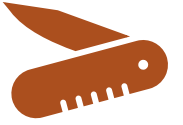
OnDemand version: 4.0.7

# Interactive applications — JupyterLab



# Interactive applications — Interactive jobs

without using portal apps



➤ run programs with **srun** *within an existing job* from your current (login) session

◦ run a command, *oversubscribing* the resources of a running job:

```
srun -n 1 -c 1 --jobid <jobid> --overlap <command>
```

◦ use **--pty** to start an *interactive command*

```
srun --jobid <jobid> --overlap --pty bash
```

- redirects input and output to your current (active) session
- other examples: python, julia, ...

➤ alternative; using **salloc**

◦ remember: it only creates an allocation and starts a shell on the login node

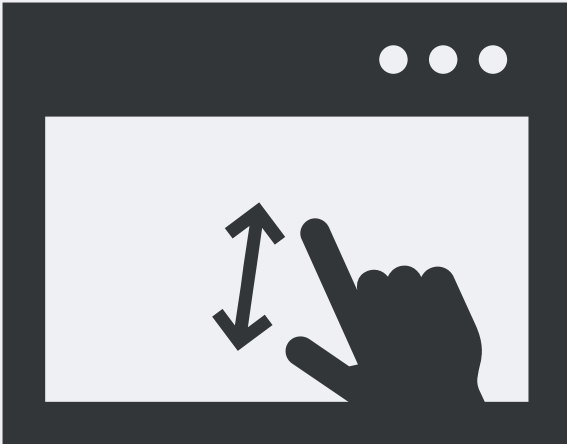
◦ subsequently, use **srun** to start your programs (like you would do in a batch job script)

# Interactive applications — Using the visualisation node

- *Note: will soon be replaced by a **desktop portal web app***
- Use case: sometimes running GUI programs is necessary – e.g.: for visualisation of results
  - and some GUI programs need GPU-accelerated hardware — e.g., GaussView, MonolixSuite
- Leibniz has one *visualisation node*: **viz1.leibniz**
  - NVIDIA Quadro Pascal P5000 GPU
  - has Xfce as desktop/window manager
  - uses VirtualGL for graphics acceleration → e.g.: **vglrun glxgears**
- To access to remote desktop, you need to
  - use a **VNC client**, such as TurboVNC or TigerVNC
  - setup an SSH-tunnel (when accessing from outside Belgium)

🔗 Remote visualisation @ UAntwerp

# Summary — Interactive applications

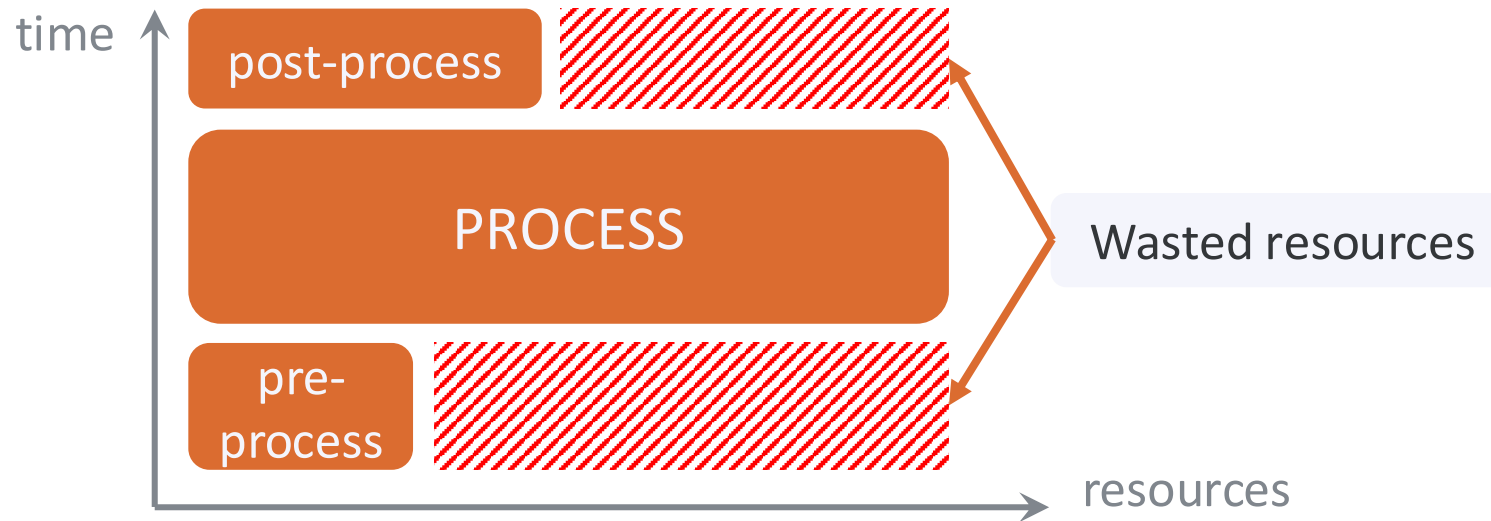


- Useful when debugging or developing
- Easy access via **Web portal**
  - Interactive shell
  - Web portal apps:
    - jupyterlab
    - Rstudio sever
    - VS Code
    - ...
- Interactive job via terminal
  - **srun** <resource-options> **--overlap --pty bash**

# Job workflows — Examples

## ➤ Some scenarios

- run simulations using results of a previous simulation, but with a different number of nodes
  - e.g., in CFD: first a coarse grid computation, then refining the solution on a finer grid
- perform extensive sequential pre- or postprocessing of a parallel job



# Job workflows — Examples

## ➤ Some scenarios

- run simulations using results of a previous simulation, but with a different number of nodes
  - e.g., in CFD: first a coarse grid computation, then refining the solution on a finer grid
- perform extensive sequential pre- or postprocessing of a parallel job
- run a sequence of simulations, each depending on result of previous one
  - what to do when the max. wall time is reached?
- run a simulation, apply perturbations to the solution
  - then run subsequent simulations for each perturbation

➤ **Workflow** = order in which the jobs will be submitted or run

# Job workflows — Passing variables to job scripts

- Remember: on UAntwerp clusters, only a minimal environment is passed to the job
- Variables need to be passed *explicitly*, otherwise `sbatch` will not see them
  - propagate a value of (already existing) environment variables

```
sbatch --export=<myenv1>,<myenv2>
```

- pass a variable with given value to the job environment

```
sbatch --export=<myenv>=<value>
```

- *note: SLURM\_\* variables are always propagated*

# Job workflows — Passing arguments to job scripts

- Command line arguments for the job script are passed *after the name of the job script*

- Create a test script

get\_parameter.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=500m
#SBATCH --time=5:00

echo "Hello $1."
```

- Now run

```
sbatch get_parameter.slurm people
```

- The output file will contain

```
Hello people.
```

# Job workflows — Job dependencies

- You can instruct Slurm to start a job only
  - when some (or all) jobs from list of jobs have *ended*

```
sbatch --dependency=afterok: <jobid>
```

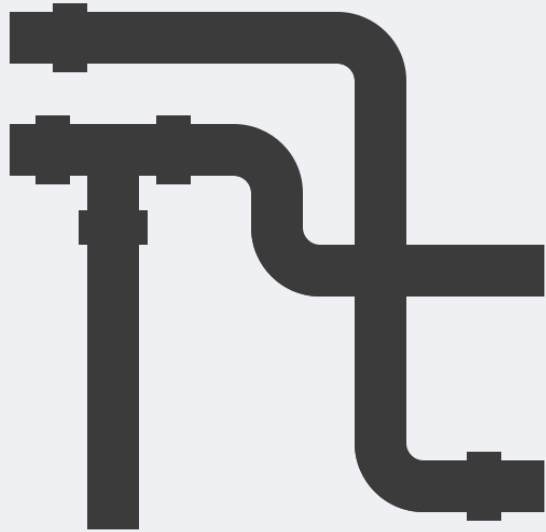
- after a job has *failed*

```
sbatch --dependency=afternotok: <jobid>
```

- Useful to organize series of (subsequent) jobs
  - powerful in combination with environment variables
  - or command line arguments passed to job scripts

🔗 [the sbatch manual page](#) – look for --dependency

# Summary — Job workflows



- A **workflow** is the order and rules that decide how jobs run
- Slurm supports job **dependencies**, where a job can only start after another job has ended  
`$ sbatch --dependency=...`
- You can pass environment variables and arguments to a job  
`$ sbatch --export=ENVVAR job.sh arg`



Vlaanderen  
is supercomputing

# HPC@UAntwerp introduction

Final notes

VLAAMS  
SUPERCOMPUTER  
CENTRUM

*Innovative Computing  
for A Smarter Flanders*

[vscentrum.be](https://vscentrum.be)

# Some site policies

- Our policies on the cluster
  - nodes are shared resources
  - **priority** based scheduling – so not “first come, first get”
  - **fair share** mechanism – to make sure one user cannot monopolise the cluster
  - Accounting @ CalcUA → using a **project account** is mandatory
- Implicit user agreement
  - the cluster is valuable research equipment
  - *do not use it for other purposes than your research for the university*
  - please do acknowledge the VSC in your publications
- Do not share your account  
nor your keys

what a cluster is not

a computer that will automatically run the code of your (PC) application much faster or for much bigger problems

# Project accounts and credits

- At **UAntwerp Tier-2**, we use accounting
  - using a **project account** is mandatory
  - billing is done for both computing (jobs) and storage (files)
- On **VSC Tier-1**, you get compute time allocation
  - number of core hours or GPU hours
  - enforced through **project credits**
  - requested through a *project proposal*
  - free test ride “**Starting Grant**” — motivation required
- On **KU Leuven Tier-2**, you need **compute credits**
  - bought directly via KU Leuven

🔗 [Project access Tier-1](#)

🔗 [Credits to use KU Leuven infrastructure](#)

# Upgrade to Rocky Linux 10 — gradual migration in H1 2027

## ➤ Start preparing for the upgrade to Rocky Linux 10 on all clusters!

### ○ Changes will include:

- new versions of kernel packages, drivers, and system installed user packages
- update of Slurm to version 26 — and of PMIx to version 5 or 6
- only toolchain versions **2025a and above** will be available

### ○ Users will need to:

- *start using modules from 2025a or above*
- *recompile their code and/or recreate their (virtual) environments*
- *check that their code still runs (including performance tests)*
- *update job scripts (if needed)*

## ➤ **Free test period** for jobs running within the **rocky10** reservation — starting now!

```
$ sbatch --reservation=rocky10
```

## ➤ Let us know your experience!

# User support

- **Questions?** → contact us via [hpc@uantwerpen.be](mailto:hpc@uantwerpen.be)
    - offices @ CMI — G.309-G.313
  - mailing-list for announcements: [calcua-announce@lists.uantwerpen.be](mailto:calcua-announce@lists.uantwerpen.be)
    - every now and then a more formal “HPC newsletter”
  - Some guidelines for help
    - be as precise as possible – e.g.: give job id, submit dir, output files, ...
    - help us help you – read (and understand) the relevant documentation
- 🔗 [CalcUA website](#) – [VSC docs](#) – [Slurm docs](#)

# The end

## Course feedback

- Please fill in our short [questionnaire](#) before November 1<sup>st</sup>
- Let us know what you liked and how we can improve our courses
- *Thank you for your participation!*



# More training

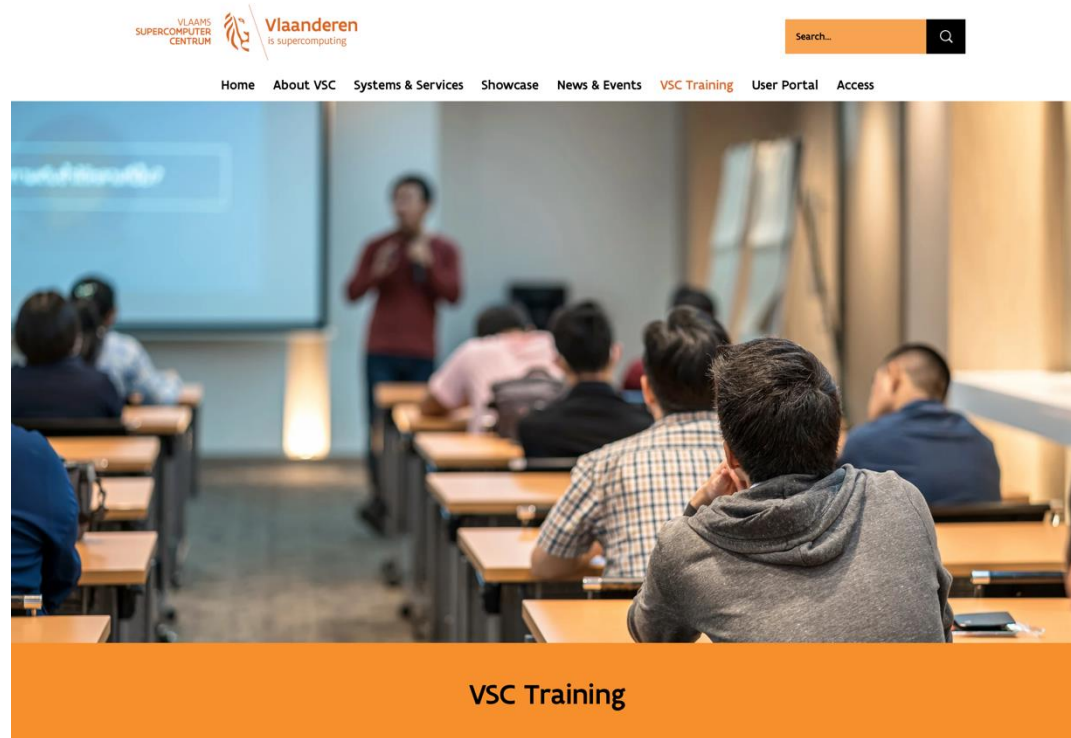
## ➤ HPC core facility CalcUA

- Introduction to Linux
- HPC@UAntwerp introduction
- Supercomputers for starters

## ➤ VSC Trainings

- trainings organized by other VSC sites and abroad (including LUMI, and EuroCC)

## ➤ Training sessions by Geert Jan Bex



The VSC spends the necessary time supporting and training researchers who make use of the infrastructure. It is important that calculations can be executed efficiently because this increases the scientific competitive position of the universities in the international research landscape. The VSC also organizes events to give its users the opportunity to get in touch with one another to foster new collaborations. The annual User Day is a prime example of such an event that also gives the users the occasion to discuss and exchange ideas with the VSC staff.

Training organized by the VSC is intended not only for researchers attached to Flemish universities and the respective associates but also for the researchers who work in the Strategic Research Centers, the Flemish scientific research institutes, and the industry.

The training can be placed into four categories that indicate either the required background knowledge or the domain-specific subject involved:

- Introductory: general usage, no coding skills required
- Intermediate
- Advanced
- Specialist courses & workshops

