Vlaanderen
is supercomputing

HPC@UAntwerp introduction

Ine Arts, Franky Backeljauw, Stefan Becuwe, Bert Jorissen, Kurt Lust, Carl Mensch, Michele Pugno, Bert Tijskens, Robin Verschoren

Version Spring 2025 – Part 1

VLAAMS SUPERCOMPUTER CENTRUM | *Innovative Computing for A Smarter Flanders*

vscentrum.be

Table of contents – Part 1

VLAAMS
SUPERCOMPUTER
CENTRUM

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

1 – Introduction to the VSC

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

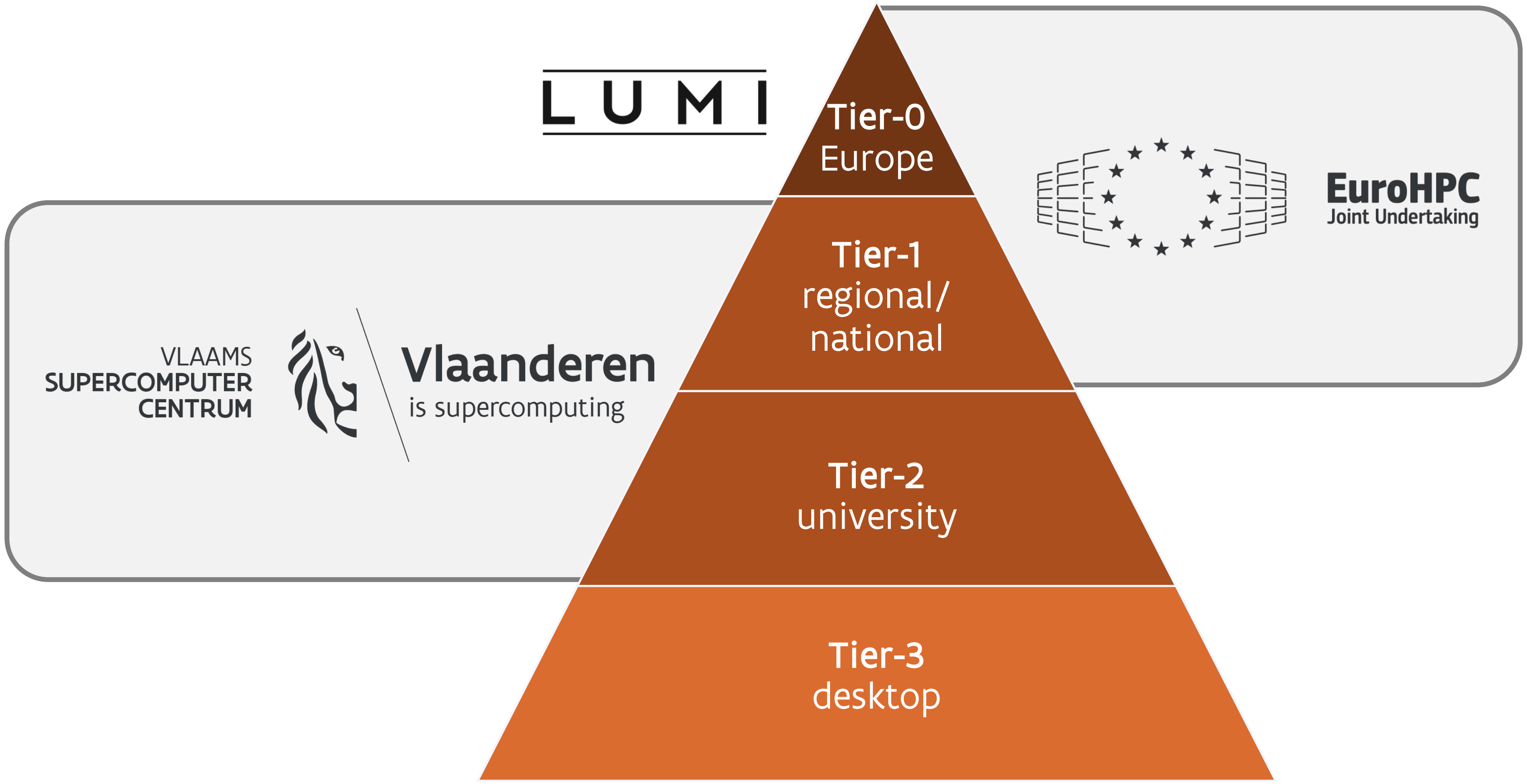
vscentrum.be

CalcUA and 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

Leibniz

2 Login nodes

152 compute nodes
- Intel Broadwell
- 2x 14 cores

2 GPU nodes
- 2x Nvidia P100

Vector engine node

Visualization node

Vaughan

2 Login nodes

152 compute nodes
- AMD Zen2
- 2x 32 cores

40 compute nodes
- AMD Zen3
- 2x 32 cores

1 GPU node
- 4x Nvidia A100

2 GPU nodes
- 2x AMD MI100

Storage system shared, 730 TB

Breniac

Login node

23 compute nodes
- Intel Skylake
- 2x 14 cores

UAntwerp Tier-2 infrastructure

CalcUA

VSC

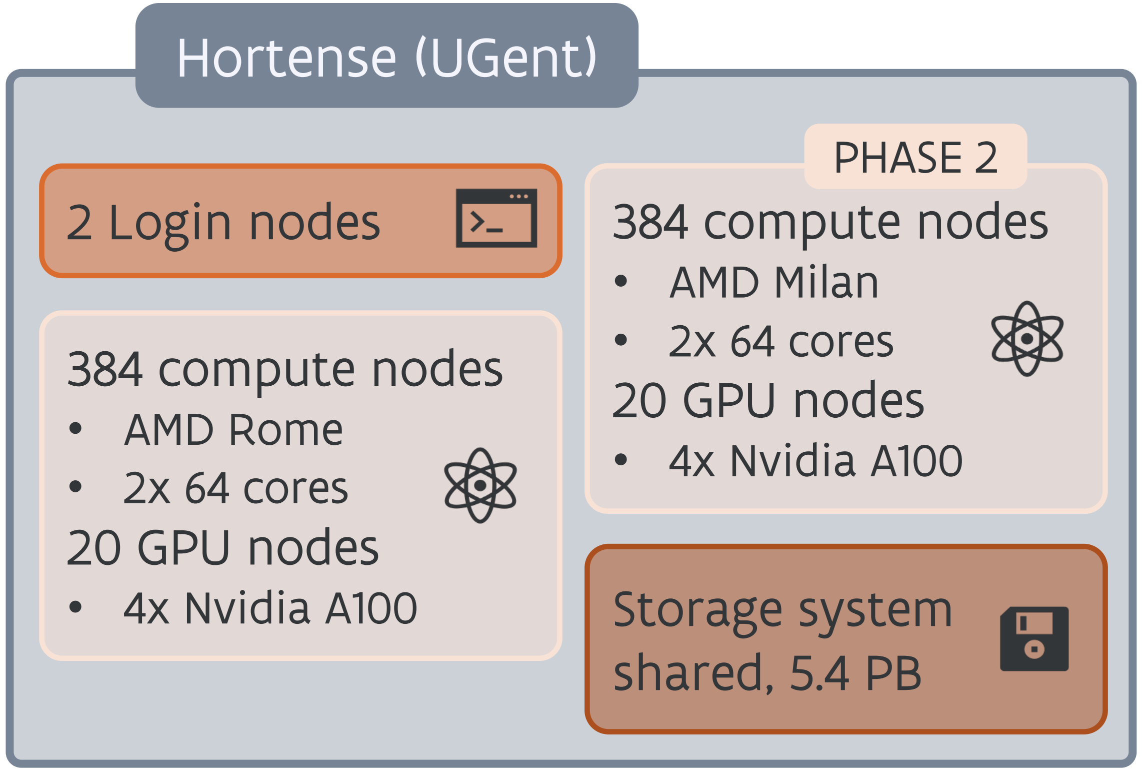
\Orchestrating a brighter world

NEC

VLAAMS
SUPERCOMPUTER
CENTRUM

VSC Tier-1 infrastructure

[VSC Tier-1 Infrastructure]



➢ new Flemish Tier-1 supercomputer (Green Energy Park @ VUB) to be operational in 2025

VLAAMS SUPERCOMPUTER CENTRUM

VSC Tier-1 infrastructure

Hortense (UGent)

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

HPC@UAntwerp introduction

2 – Get a VSC account

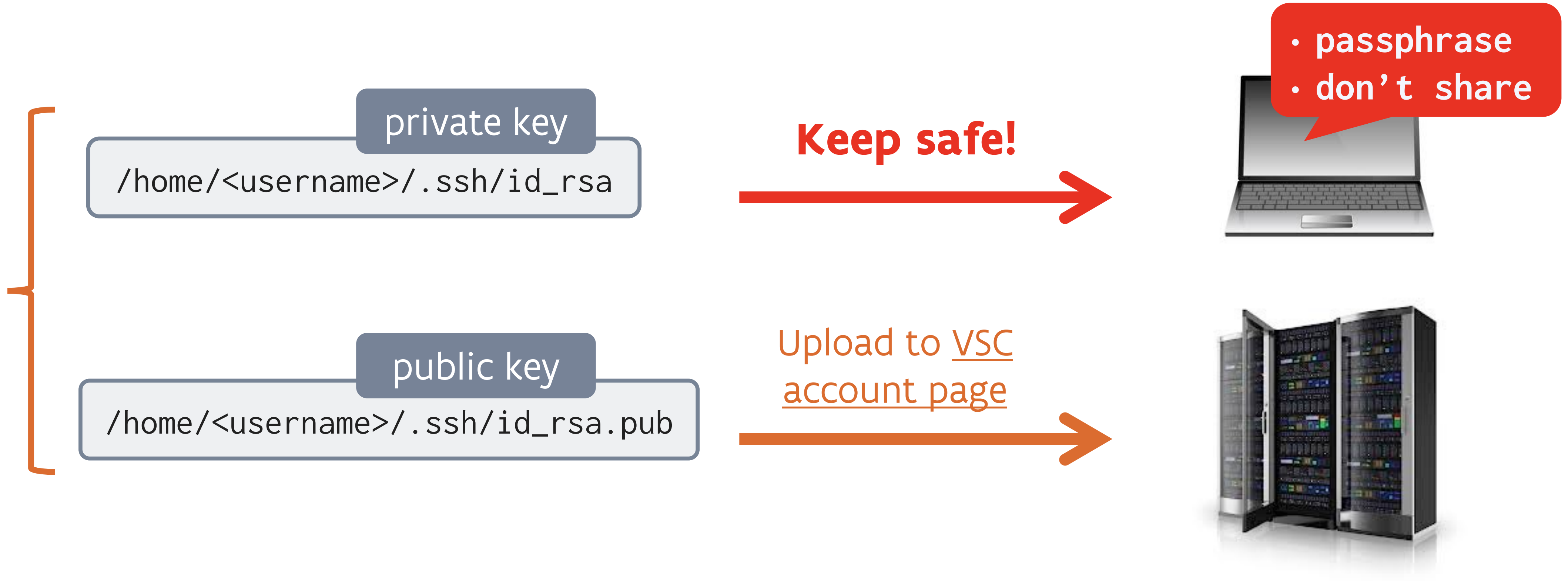
Vlaanderen is supercomputing

VLAAMS SUPERCOMPUTER CENTRUM | *Innovative Computing for A Smarter Flanders*

vscentrum.be

SSH and 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


private key
/home/<username>/.ssh/id_rsa
Keep safe!
• passphrase
• don't share
public key
/home/<username>/.ssh/id_rsa.pub
Upload to VSC account page

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)
  - MobaXterm combines a SSH/SFTP client, X server and VNC server in one
  - PuTTY used to be a popular GUI SSH client

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

Create your VSC account

- Create a public/private key pair
  - create RSA key pair (at least 4096 bits)

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

  - 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
  - web-based registration procedure

➢ your VSC username is **vsc2xxxx**

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

3 – Connect to the cluster

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

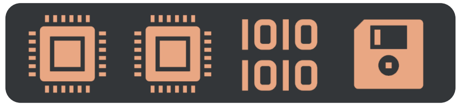
vscentrum.be

A typical workflow

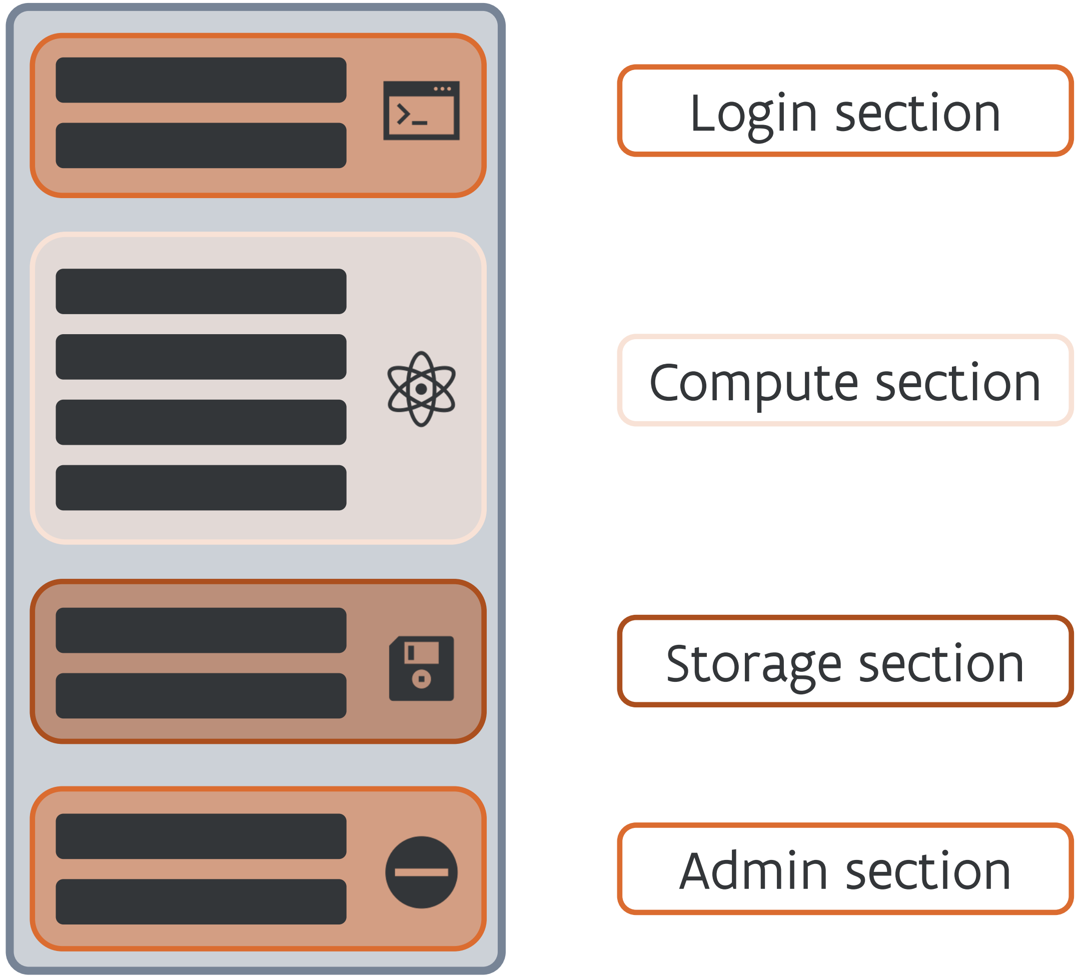
1. **Connect to the cluster**
2. Transfer your files to the cluster
3. Select the software and build your environment
4. Define and submit your job
5. Wait while
   - ➢ your job gets scheduled
   - ➢ your job gets executed
   - ➢ your job finishes
6. Move your results

Types of cluster nodes

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



- Login nodes
  - access to cluster
  - edit & submit jobs
  - small compilations
- Compute nodes
  - actual computations



VLAAMS SUPERCOMPUTER CENTRUM

Connecting to the cluster – Using SSH

- You need:
  - VSC account name: `vsc2xxxx`
  - Hostname of a login node
  - Private key (public key already uploaded)
- Restricted public access
  - outside of Belgium: use **VPN**
    - 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

Connecting to the cluster – Using SSH

- 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

VLAAMS
SUPERCOMPUTER
CENTRUM

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`))

➢ Put this file in `~/.ssh/config` and then you can connect using: `ssh calcua`

SSH config

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**

- Create a SSH configuration file
  - *feel free to choose your own shorthand name*
  - login using the shorthand name

VLAAMS
SUPERCOMPUTER
CENTRUM

HPC@UAntwerp introduction

4 – Transfer your files to the cluster

Vlaanderen is supercomputing

VLAAMS SUPERCOMPUTER CENTRUM | *Innovative Computing for A Smarter Flanders*

vscentrum.be

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`
  - fast but temporary storage
  - highest performance – for large files
  - local only, no backup
- `/data/antwerpen/2xx/vsc2xxyy`
  - long-term storage
  - 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

- Block quota limits the *size of data*
- File quota limits the *number of files*

- Default values (but you can request more)

	File system	**Block quota**	**File quota**
	/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

Transferring 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
- We recommend **Globus** (next slide)
  - also has a command-line interface as well as a Python SDK

[Data transfer on external computers]

Globus data sharing platform

Globus web app

- web service to transfer large amounts of data between local computers and/or remote servers
- offers data sharing features (guest collections), connectors (for OneDrive), CLI interface

➢ HPC@UAntwerp collection: **VSC UAntwerpen Tier2**

- login with UAntwerp or VSC account *– note: active VSC account needed*
- access to /data and /scratch

➢ Transfer between: local computer (laptop/desktop) ⟷ remote server

- required software: Globus Connect Personal
- transfers will be resumed automatically

➢ Direct transfer: remote server ⟷ remote server

- initiated from your local computer (no local software needed)

Globus data sharing platform

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
  - feel free to use either command-line tools (`scp` and/or `sftp`) or a graphical client
  - check on which clusters these files are available
- Copy the files back using the Globus web app
  - download and install Globus Connect Personal
  - good practice: configure it to use a dedicate subdirectory of your choice
  - initiate the transfer back to your laptop
    - look at the options

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

5 – Select the software and build your environment

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing for A Smarter Flanders

vscentrum.be

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 8.10 *– note: upgrade to 9.x is pending*
  - 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 and 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, …

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, HMMER, MAFFT
- Pharmacokinetics / Pharmacodynamics Modeling – *MonolixSuite*
- 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, MonolixSuite, …
  - restrictions may apply if you don't work at UAntwerp
    - institutions that have access (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) → 2024a, 2023b, 2023a, 2022b, 2022a, …
  - 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

CalcUA module	**Software collection**
`calcua/2024a`	version 2024a of the *toolchain compiler* modules + software built with them
`calcua/system`	software built with *system compilers*
`calcua/x86_64`	software installed from *binaries* (x86_64)
`calcua/all`	all currently available software (all of the above)

- Currently available versions of the toolchain compiler modules
  - 2024a, 2023a, 2022a, 2021a : mostly foss, but also intel
  - 2020a : intel only

- *Good practice: always load a `calcua` module first!*

VLAAMS SUPERCOMPUTER CENTRUM

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 *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/2023a
```

Load appropriate calcua module first

- makes the modules available (here, from 2023a)

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

Load the modules you want to use

- advise: explicitly specify name & version

➢ *Advice: do not load modules in your .bashrc*

- 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

VLAAMS SUPERCOMPUTER CENTRUM

HPC@UAntwerp introduction

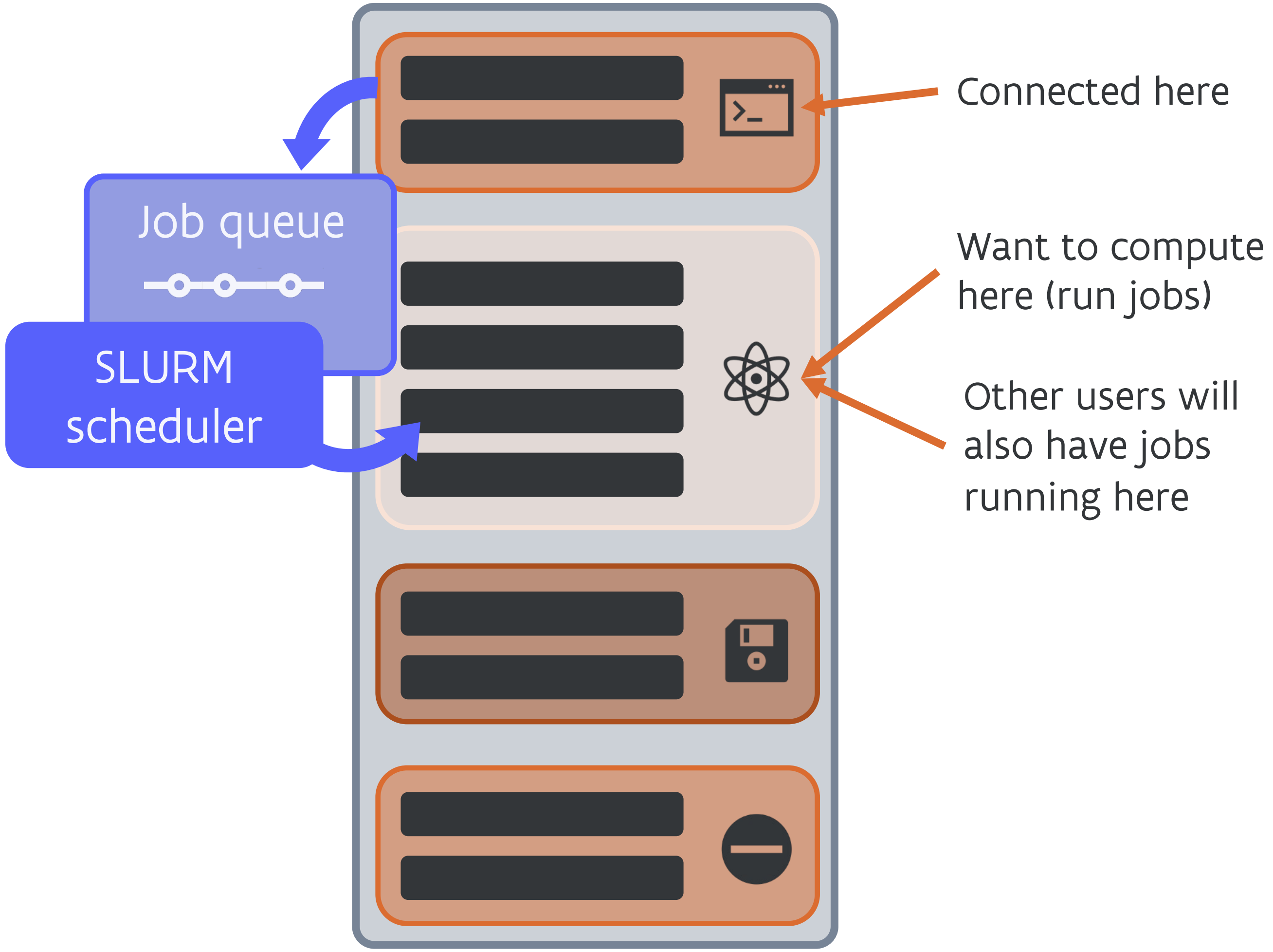
6 – Define and submit your job

A typical workflow

1. Connect to the cluster
2. Transfer your files to the clusters
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

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

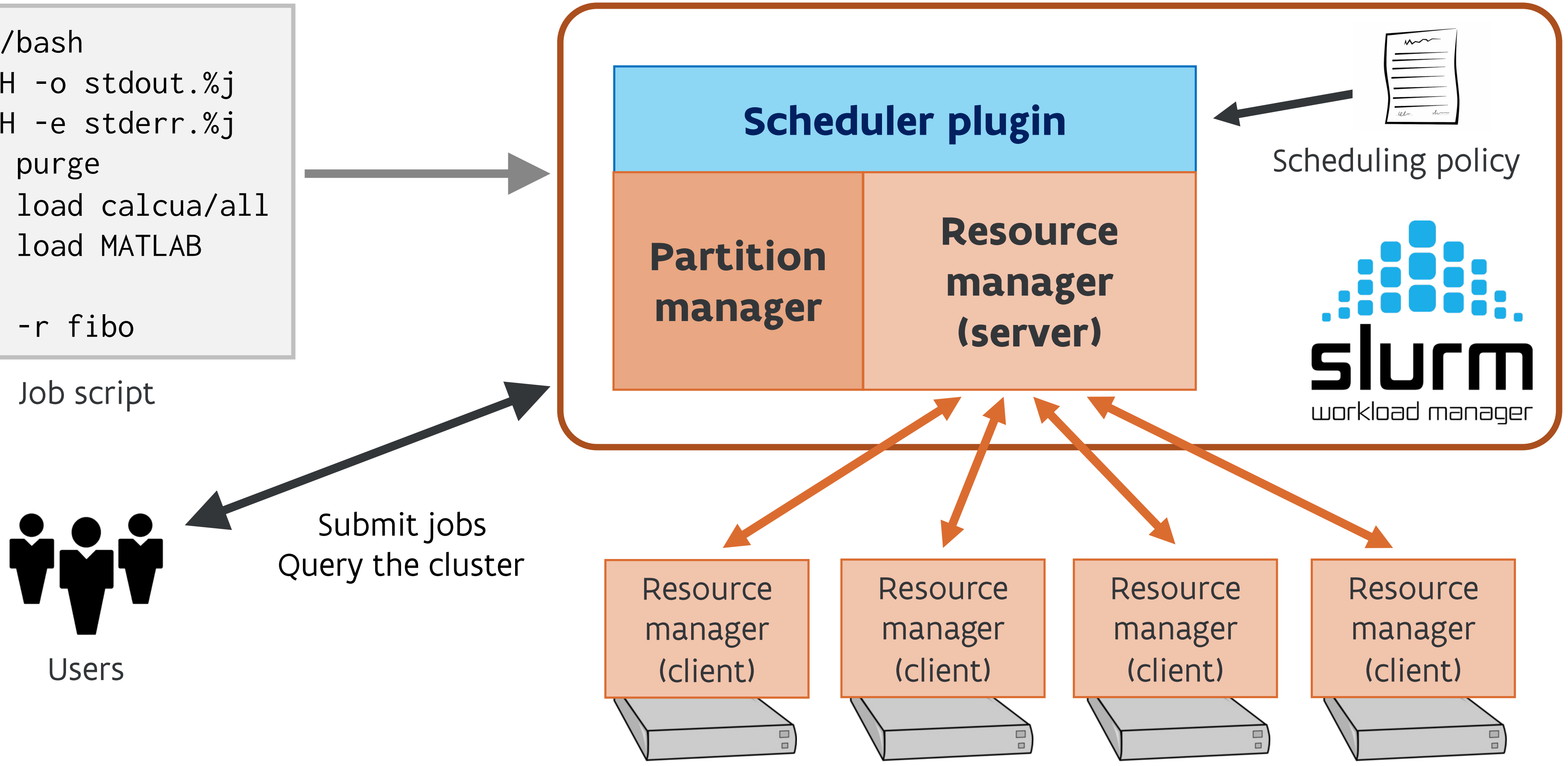

Job queue
SLURM scheduler
Connected here
Want to compute here (run jobs)
Other users will also have jobs running here

Job submission workflow – Behind the scenes

```
#!/bin/bash
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
module purge
module load calcua/all
module load MATLAB

matlab -r fibo
```

Job script


Scheduler plugin
Partition manager
Resource manager (server)
Scheduling policy
slurm workload manager
Submit jobs
Query the cluster
Users
Resource manager (client)
Resource manager (client)
Resource manager (client)
Resource manager (client)

VLAAMS SUPERCOMPUTER CENTRUM

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/2024a
module load Python/3.12.3-GCCcore-13.3.0

python pi.py
```

Important Slurm concepts

Node	Compute node
Core	Physical core (in physical cpu)
CPU	Virtual core – *hardware thread* • on the CalcUA clusters, hyperthreading is disabled → **CPU = Core**
Partition	Group of nodes with job limits and access controls – *aka job queue*
Job	Submitted job script – *resource allocation request*
Job step	Set of (possibly parallel) tasks within a job • the job script itself is a special step – the *batch job step* • e.g., a MPI application typically runs in its own job step
Task	Corresponds to a (single) Linux process, executed in a job step • a single task can not use more CPUs than available in a single node • e.g., for a MPI application, each rank (MPI process) is a task but a shared memory program is a single task

Slurm resource requests – Overview

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

Slurm resource requests – Project account

Long option	Short option	Job environment variable	Description
`--account=`*<ap_proj>*	`-A` *<ap_proj>*	`SLURM_JOB_ACCOUNT`	Project account to use

- Required to specify a **project account** at CalcUA clusters
  - accounting for both compute (jobs) and storage (files)
  - *ask your supervisor or project account manager to get access*
  - use an appropriate account according to the project

- Show accounts you have access to `myprojectaccounts`
  - all project accounts start with **ap_**
  - during this course → `ap_course_hpc_intro`

Accounting @ CalcUA (slides & video)

VLAAMS
SUPERCOMPUTER
CENTRUM

Slurm resource requests – Tasks & CPUs per task

Long option	Short option	Job environment variable	Description
`--ntasks`=*<number>*	`-n` *<number>*	`SLURM_NTASKS` (if set)	Number of tasks
`--cpus-per-task`=*<ncpus>*	`-c` *<ncpus>*	`SLURM_CPUS_PER_TASK` (if set)	Number of CPUs per task

- Specify number of (parallel) tasks and CPUs (cores) per task
  - Task = single process (runs within a single node)
  - 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

Long option	**Job environment variable**	**Description**
`--mem-per-cpu=<amount><unit>`	`SLURM_MEM_PER_CPU` (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 GB - 16 GB) / 64 CPUs = 240 / 64 = **3.75GB = 3840 MB (per core)**

VLAAMS SUPERCOMPUTER CENTRUM

Slurm resource requests – Wall time

Long option	Short option	Job environment variable	Description
`--time=<time>`	`-t <time>`	`SLURM_JOB_START_TIME` `SLURM_JOB_END_TIME`	Time limit = ***wall time***

- 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 – Partitions

Long option	Short option	Job environment variable	Description
`--partition=`*<pname>*	`-p` *<pname>*	`SLURM_JOB_PARTITION`	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

Cluster	Partition	#	Specifications	CPU – GPU	Mem per CPU	Max WT
Vaughan	**zen2**	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	**broadwell**	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	**skylake**	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/202203-intel-2024a` – *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 file are generated?

Slurm resource requests – Faster communication

Long option	Description
`--switches=1`	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

Long option	Description
`--exclusive`	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

Long option	**Short option**	**Job environment variable**	**Description**
`--nodes=`*`<number>`*	`-N` *`<number>`*	`SLURM_JOB_NUM_NODES`	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

Long option	Short option	Job environment variable	Description
`--job-name=`*<jobname>*	`-J` *<jobname>*	`SLURM_JOB_NAME`	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

Long option	Short option	Description
`--output=<outfile>`	`-o <outfile>`	Redirect stdout
`--error=<errfile>`	`-e <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

VLAAMS
SUPERCOMPUTER
CENTRUM

Non-resource-related options – Mail notifications

Long option	**Description**
`--mail-type=`*<type>*	Event notification (start, end, …)
`--mail-user=`*<email>*	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

VLAAMS
SUPERCOMPUTER
CENTRUM

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*

Environment variable	Explanation
`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

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, Stefan Becuwe, Bert Jorissen, Kurt Lust, Carl Mensch, Michele Pugno, Bert Tijskens, Robin Verschoren

Version Spring 2025 – Part 2

VLAAMS SUPERCOMPUTER CENTRUM | *Innovative Computing for A Smarter Flanders*

vscentrum.be

Table of contents – Part 2

VLAAMS
SUPERCOMPUTER
CENTRUM

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

7 – Slurm commands

VLAAMS
SUPERCOMPUTER
CENTRUM

*Innovative Computing
for A Smarter Flanders*

vscentrum.be

Slurm commands – Overview

Command	Description
`sbatch`	Submit a batch script
`srun`	Run parallel tasks – start an interactive job
`salloc`	Create a resource allocation
`squeue`	Check the status of your jobs
`scancel`	Cancel a job
`sstat`	Information about *running* jobs
`sacct`	Information about (terminated) jobs
`sinfo`	Get an overview of the cluster, partitions and nodes
`scontrol`	View current Slurm configuration and state

sbatch – Submit a batch script

- `sbatch <sbatch arguments> jobscript <arguments of the job script>`
  - 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 batch script

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

squeue – Check the status of your jobs

- `squeue` checks the status of your *own* jobs in the job queue

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

  - ST = **state** of the job

ST	Explanation	ST	Explanation
PD	Pending – waiting for resources	**F**	Failed job – non-zero exit code
CF	Configuring – nodes becoming ready	**TO**	Timeout – wall time exceeded
R	Running	**OOM**	Job experienced out-of-memory error
CD	Successful completion – exit code zero	**NF**	Job terminated due to node failure

squeue manual page – job state codes

squeue – Check the status of your jobs

- `squeue` checks the status of your *own* jobs in the job queue

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

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

NODELIST(REASON)	Explanation
Priority	There are one or more higher priority jobs in the partition
QOSMaxNodePerUserLimit	The limit on the maximum number of nodes per user will be 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 single job + all its job steps (if already running)
  - cancel a specific job step: `scancel <jobid>.<stepid>`
    - *e.g., 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

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

VLAAMS
SUPERCOMPUTER
CENTRUM

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

`srun` – Run parallel tasks

- `srun` *"Swiss Army Knife"* to create & manage (parallel) tasks within a job
  - in Slurm terminology: it creates a job step that can run one or more parallel tasks
  - run multiple jobs steps *simultaneously*, each using a part of the allocated resources
  - *the better way of starting MPI programs* – preferred over `mpirun` and `mpirun`
    - usage will be shown through examples
  - run a command on all allocated nodes of a running job:

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

  - run a shell on the first allocated nodes of a running job:

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

  - alternatively, use **ssh** to log into any allocated node of a running job
    - but only possible as long as the job is running

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: the shell is not running on 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
  - it is possible to specify a subset of 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

sstat manual page

sacct – Information about (terminated) jobs

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

```
$ sacct -j 12345
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
12345         NAMD-S-00+      zen2 antwerpen+         64  COMPLETED      0:0
12345.batch        batch            antwerpen+         64  COMPLETED      0:0
12345.extern      extern            antwerpen+         64  COMPLETED      0:0
12345.0            namd2            antwerpen+         64  COMPLETED      0:0
```

sacct – Information about (terminated) jobs

- Retrieving job details
  - 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)
  - get (all) the details of a given job – module load Miller

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

  - get the batch script of a given job

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

sacct 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 `<job_name>.<job-id>.out`
    - 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`
- Our repository for this course: https://github.com/hpcuantwerpen/intro-hpc

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

8 – Multi-core parallel jobs

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

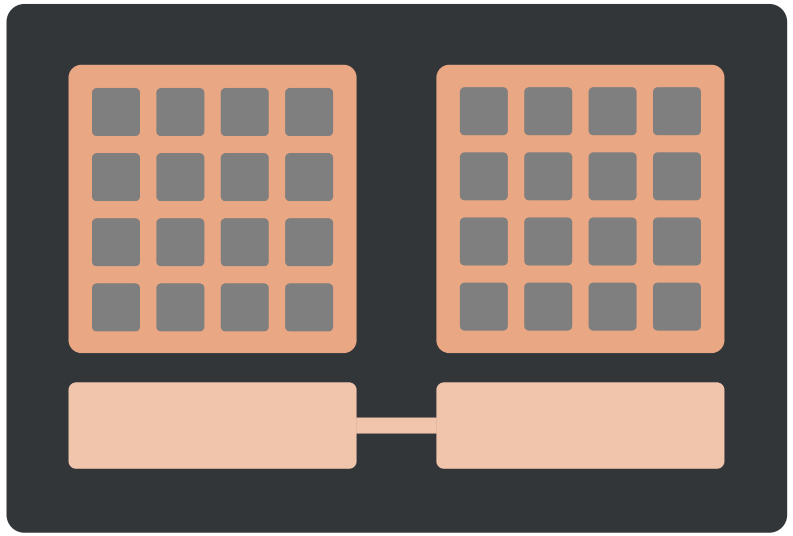
vscentrum.be

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

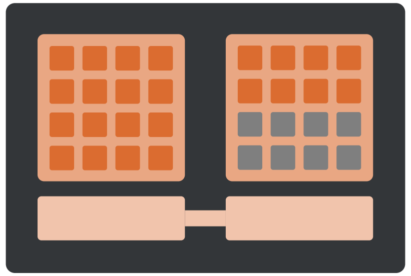
1. Multi**threading**
   - *shared memory*
   - OpenMP
2. Multi**processing**
   - *distributed memory*
   - MPI
3. Hybrid
   - *combination*



⬛ Compute node
🟧 Processor (socket)
◼ Core (=CPU)
▭ Memory

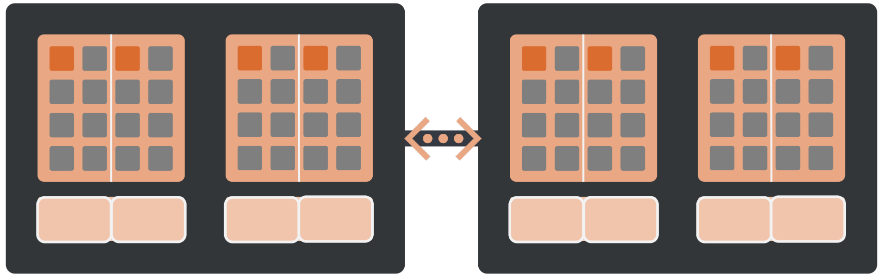
Types of parallel computing

1. Multi**threading**
   - *shared memory*
   - OpenMP



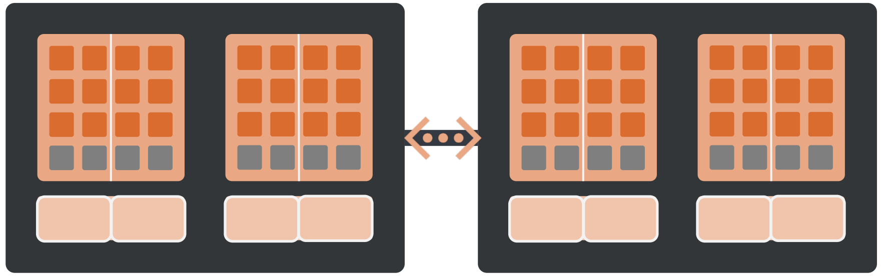
OpenMP software uses multiple or all cores in a **single** node
e.g. 24 threads within 1 node

2. Multi**processing**
   - *distributed memory*
   - MPI



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

3. Hybrid
   - *combination*

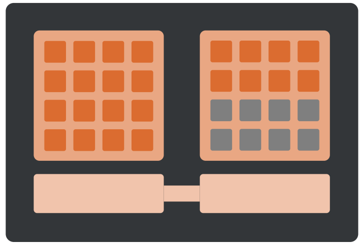


Hybrid OpenMP/MPI software
e.g. 6 threads per task
& 8 tasks over 2 nodes
(each task stays within 1 node)

🟧 *Active core*
⬛ *Inactive core*

VLAAMS SUPERCOMPUTER CENTRUM

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

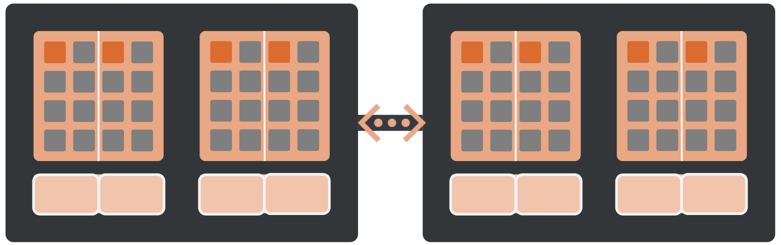
```
#!/bin/bash

#SBATCH --job-name=OpenMP-demo
#SBATCH -A ap_course_hpc_intro
#SBATCH --ntasks=1 --cpus-per-task=64           ← 1 task with 64 CPUs (so 64 threads)
#SBATCH --mem-per-cpu=2g                        ← 2 GB per CPU, so 128 GB total memory

module --force purge
module load calcua/2024a                        ← load the calcua module
module load vsc-tutorial/202203-intel-2024a     ← load vsc-tutorial – also loads the Intel
                                                   toolchain (for the OpenMP run time)

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK     ← set the number of (OpenMP) threads to use
export OMP_PROC_BIND=true                       ← threads stay on the core where they're created
omp_hello                                       ← 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
    - so 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?

VLAAMS SUPERCOMPUTER CENTRUM

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

module --force purge
module load calcua/2024a
module load vsc-tutorial/202203-intel-2024a

srun mpi_hello
```

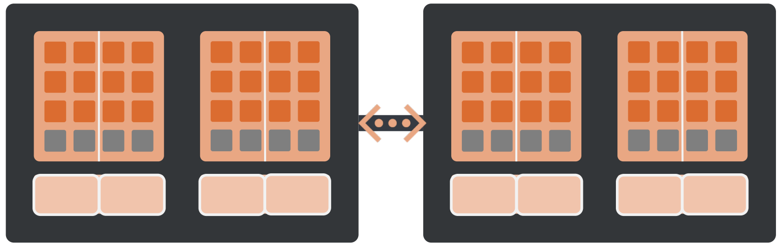
← 128 MPI processes (uses 2 nodes on Vaughan, or 5 nodes on Leibniz/Breniac)

← load the calcua module

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

← 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-mympirun` (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/2024a
module load vsc-tutorial/202203-intel-2024a

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

VLAAMS SUPERCOMPUTER CENTRUM

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>*
  - log on to a compute node: `ssh` *<compute-node>*
  - run a command on all nodes: `srun --jobid` *<jobid>* `--overlap` *<command>*
- When logged in on the compute node, check the behavior
  - **`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*
  - **`pstree`** → display a tree view of the running processes

Job monitoring – The `monitor` module

- Add monitoring in your job script
  - sample a programs’ metrics – CPU usage and memory consumption
  - can also check the sizes of (temporary) files
  - only single node jobs are supported – not MPI support
- Usage examples:
  - `monitor -d 30 -n 20 -l monitor.log` *<command>*
    - use a sample rate (delta) of 30 seconds, keep only the last 20 results, and log to a file
  - `monitor -f file1.tmp,file2.tmp <command>`
    - check the size of the (temporary) files
  - `monitor -d 60 -- matlab -nojvm -nodisplay computation.m`
    - delimit the monitor’s options (to avoid confusion)

[Monitoring memory and CPU usage of programs]

[Github repository for monitor] – by Geert Jan Bex

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

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

9 – Organizing job workflows

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Examples of job workflows

- 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

Passing (environment) 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*

VLAAMS
SUPERCOMPUTER
CENTRUM

Passing command line 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.
```

VLAAMS SUPERCOMPUTER CENTRUM

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*

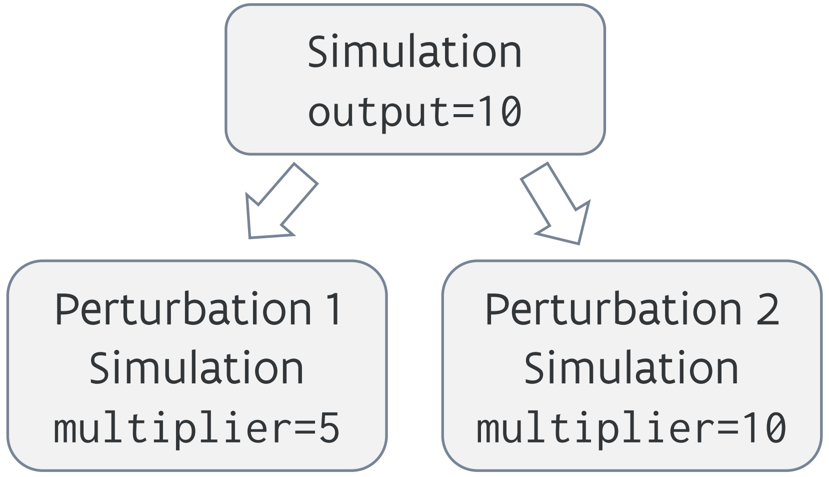
    ```
    sbatch --dependency=afternotok:<jobid>
    ```

- 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`

Job dependencies – Example

- Use case: let's transform this example of *sequential* simulation runs into
  - a job that runs the *first* simulation
  - followed by a bunch of *subsequent* perturbations that use the result of the first simulation
    - the perturbations are mutually independent



job.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=30:00

echo "10" >outputfile ; sleep 300

multiplier=5
mkdir mult-$multiplier ; cd mult-$multiplier
resultFirst=$(cat ../outputfile)
echo $(($resultFirst*$multiplier)) >outputfile
cd ..

multiplier=10
mkdir mult-$multiplier ; cd mult-$multiplier
resultFirst=$(cat ../outputfile)
echo $(($resultFirst*$multiplier)) >outputfile
```

VLAAMS SUPERCOMPUTER CENTRUM

Job dependencies – Example

job_first.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --time=10:00

echo "10" >outputfile
sleep 300
```

job_depend.slurm

```
!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --time=10:00

mkdir mult-$multiplier
cd mult-$multiplier
resultFirst=$(cat ../outputfile)
echo $(($resultFirst*$multiplier)) >outputfile
sleep 300
```

- To automate the submission, store the job id of the first job in a variable and pass it to the dependency options for the subsequent jobs

job_launch.sh

```
#!/bin/bash
first=$(sbatch --parsable --job-name job_leader job_first.slurm)
sbatch -J job_mult_5 --export=multiplier=5 --dependency=afterok:$first job_depend.slurm
sbatch -J job_mult_10 --export=multiplier=10 --dependency=afterok:$first job_depend.slurm
```

Job dependencies – Example

- After start of the first job, the other check will be in state PD (pending) – see with `squeue`

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
24869      zen2 job_mult vsc20259 PD       0:00      1 (Dependency)
24870      zen2 job_mult vsc20259 PD       0:00      1 (Dependency)
24868      zen2 job_lead vsc20259  R       0:25      1 r1c01cn1
```

- When the first job finishes successfully, the subsequent jobs will start running

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
24869      zen2 job_mult vsc20259  R       0:01      1 r1c01cn1
24870      zen2 job_mult vsc20259  R       0:01      1 r1c01cn1
```

- Finally, the output files will contain the proper results:

```
cat outputfile              10
cat mult-5/outputfile       50
cat mult-10/outputfile     100
```

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

10 – Multi-job submission

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

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
  - **`worker`** framework: manages "embarrassingly parallel" computations in a single job
    - can be used for any scenario that can be reduced to a Map-Reduce approach
  - 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

VLAAMS SUPERCOMPUTER CENTRUM

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"

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

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

VLAAMS SUPERCOMPUTER CENTRUM

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 worker and GNU `parallel`
  - `atools` is less efficient than `worker` for very small jobs
  - because `atools` uses job arrays, so relies on the scheduler to start all work items
  - while `worker` does all the job management for the work items itself (including starting them)

worker-and-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
```

Hands-on

- *Round the table questions*
  - *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
- Run the appropriate scenarios from the previous chapters

HPC@UAntwerp introduction

11 – Extra topics

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Running an interactive job

- Example: interactive session to run a *shared memory* application

```
login $ srun -n 1 -c 16 --interactive --pty bash
rXcYYcnZ $ module --force purge
rXcYYcnZ $ ml calcua/2024a vsc-tutorial/202203-intel-2024a
rXcYYcnZ $ omp_hello
rXcYYcnZ $ exit
```

- Example: starting an *MPI program* in an interactive session

```
login $ srun -n 64 -c 1 --interactive --pty bash
rXcYYcnZ $ module --force purge
rXcYYcnZ $ ml calcua/2024a vsc-tutorial/202203-intel-2024a
rXcYYcnZ $ srun mpi_hello
rXcYYcnZ $ exit
```

  - note: the `--interactive` option is intentionally not documented

Running an interactive job – X11

- First make sure that your login session supports X11 programs
  - log in to the cluster using **`ssh -X`** to forward X11 traffic
  - or work from a terminal window in a VNC session
- Similar to non-X11 interactive jobs, but explicitly add the **`--x11`** option before `--pty bash`

```
login $ srun -n 1 -c 64 --x11 --pty bash
rXcYYcnZ $ module --force purge
rXcYYcnZ $ ml calcua/2024a …
rXcYYcnZ $ xclock
rXcYYcnZ $ exit
```

- Or immediately start X11 programs directly through srun

```
login $ srun -n 1 -c 1 --x11 xclock
```

- Note: few X11 programs support distributed memory computing
  - so usually you'll only be using one task …

VLAAMS
SUPERCOMPUTER
CENTRUM

Using the visualisation node

- 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

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

```
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

VLAAMS SUPERCOMPUTER CENTRUM

Installing your own packages

- 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!*
  - 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

Using containers – hpc-container-wrapper

- 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
  - ...
```

VLAAMS
SUPERCOMPUTER
CENTRUM

Using containers – hpc-container-wrapper

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

```
$ export PATH="$VSC_SCRATCH/containers/bsoup/bin:$PATH”
$ which python
$VSC_DATA/containers/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/containers/bsoup"
```

post.sh

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

Github repository or documentation

VLAAMS SUPERCOMPUTER CENTRUM

Using containers – apptainer

- Apptainer is available to build and run your container images – *fka Singularity*
  - *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, with GitHub repository – by Geert Jan Bex

HPC@UAntwerp introduction

12 – Final notes

Vlaanderen is supercomputing

VLAAMS SUPERCOMPUTER CENTRUM

Innovative Computing for A Smarter Flanders

vscentrum.be

Some best practices

- Before starting to submit jobs, you should always check
  - are there any errors in the script?
  - are the required modules loaded?
  - is the correct executable used?
  - did you use the right process starter (srun)?
  - does the job start in the right directory?
- Check your jobs at runtime
  - login to a compute node and inspect your jobs
    - If you see that the CPU is idle most of the time that might be the problem
  - check the job accounting information (e..g.: MinCPU and AvgCPU)
  - alternatively: run an interactive job for the first run of a set of similar runs
  - try to benchmark the software for (I/O) scaling issues when using MPI

Warnings

- Avoid submitting many small jobs (in number of cores) by grouping them
  - using a job array
  - or using the atools or the Worker framework
- Runtime is limited by the maximum wall time of the queues
  - for longer wall time, use checkpointing
  - properly written applications have built-in checkpoint-and-restart options
- Requesting many processors could imply long waiting times
  - though we're still trying to favour parallel jobs

what a cluster is not

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

Some site policies

- Our policies on the cluster
  - nodes are shared resources
  - priority based scheduling – so not “first come, first get”
  - fairshare 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*
    - no cryptocurrency mining or SETI@home and similar initiatives!
    - not for private use
  - you have to acknowledge the VSC in your publications

- Do not share your account nor your keys

Project accounts and credits

- At **UAntwerp Tier-2**, we introduction accounting – as of March 2024
  - 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
  - has fixed start-up cost
  - used resources (number and type of nodes)
  - duration (used wall time)

VLAAMS SUPERCOMPUTER CENTRUM

User support

- **Questions?** → contact us via **hpc@uantwerpen.be**
  - offices @ **CMI – G.309-G.311**

- mailing-list for announcements: calcua-announce@sympa.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

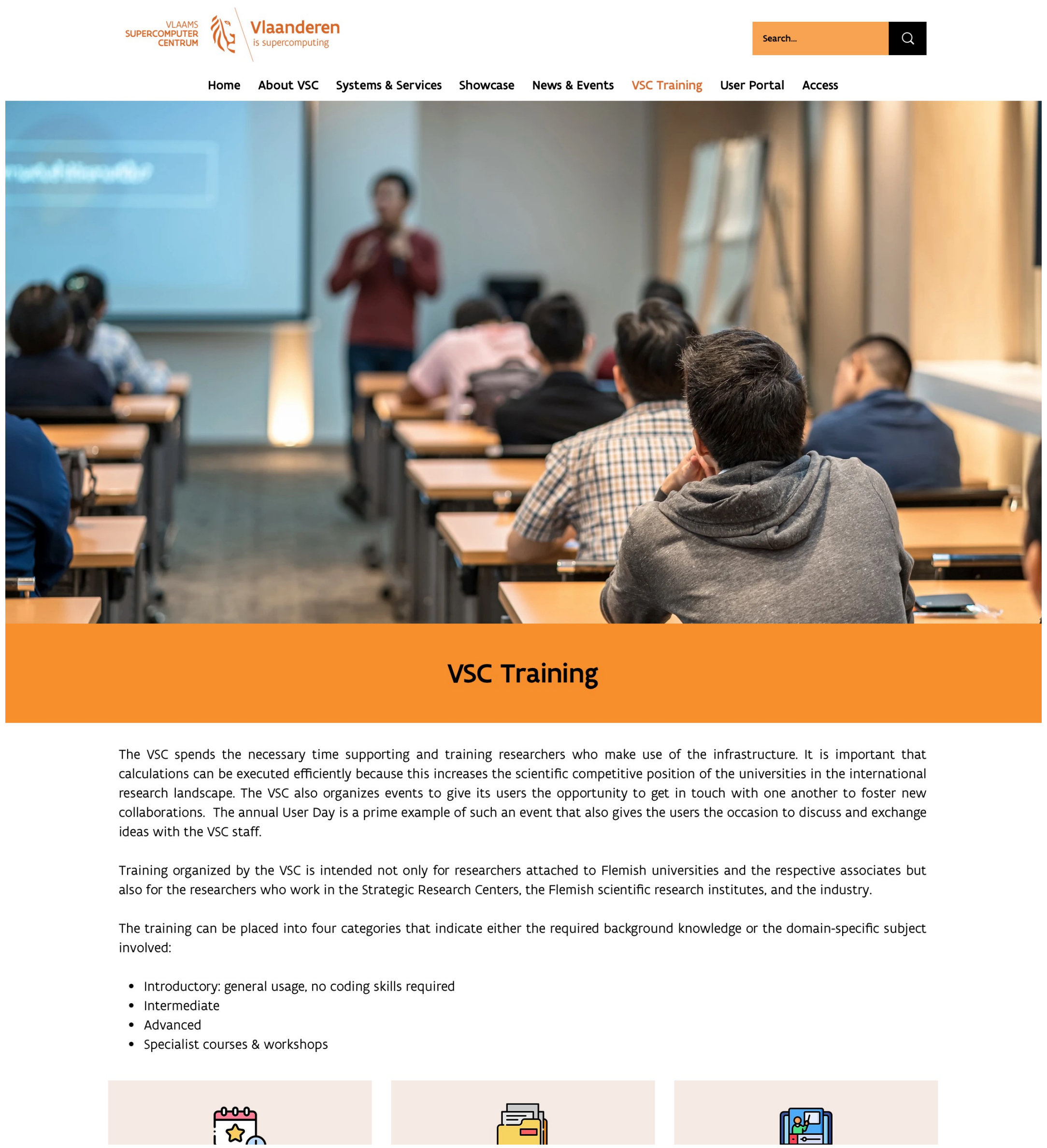
CalcUA website – VSC docs – Slurm docs

Evaluation

- Please fill in our short questionnaire before 13 Mar
- Let us know what you liked and how we can improve our courses
- Thank you for your participation!

More training

- HPC core facility CalcUA
  - HPC@UAntwerp introduction
  - More Linux commands
- VSC Trainings
  - trainings organized by other VSC sites and abroad (including LUMI, PRACE, EUROCC)



VSC Training

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