VSC HPC Introduction

ICTS KU Leuven
VSC staff:
Ehsan Moravveji
Mag Selwa
Geert Jan Bex (Uhasselt)
Jan Ooghe

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

What is High Performance Computing?

- using supercomputers to solve advanced computation problems
- Reduce the computation time from days, years, decades, or centuries to minutes, hours, days, or weeks
- The key is parallelism

In practice, it is more like …

The concept is simple: **Parallelism** = employing multiple processors for a single problem

Outline

- What is the VSC?
- What is a cluster?
- Genius Cluster
- Storage
- Login nodes
- Connection Setup
- Software environment
- How to submit jobs?
- Dedicated hardware
- How to choose resources?
- Optional material
  - Linux in brief
  - Conda for Python and R
  - Worker Framework

VSC

(Vlaams Supercomputer Centrum)

VSC PARTNERSHIP

Supported by

fwo

ASSOCIATIE UNIVERSITEIT GENT

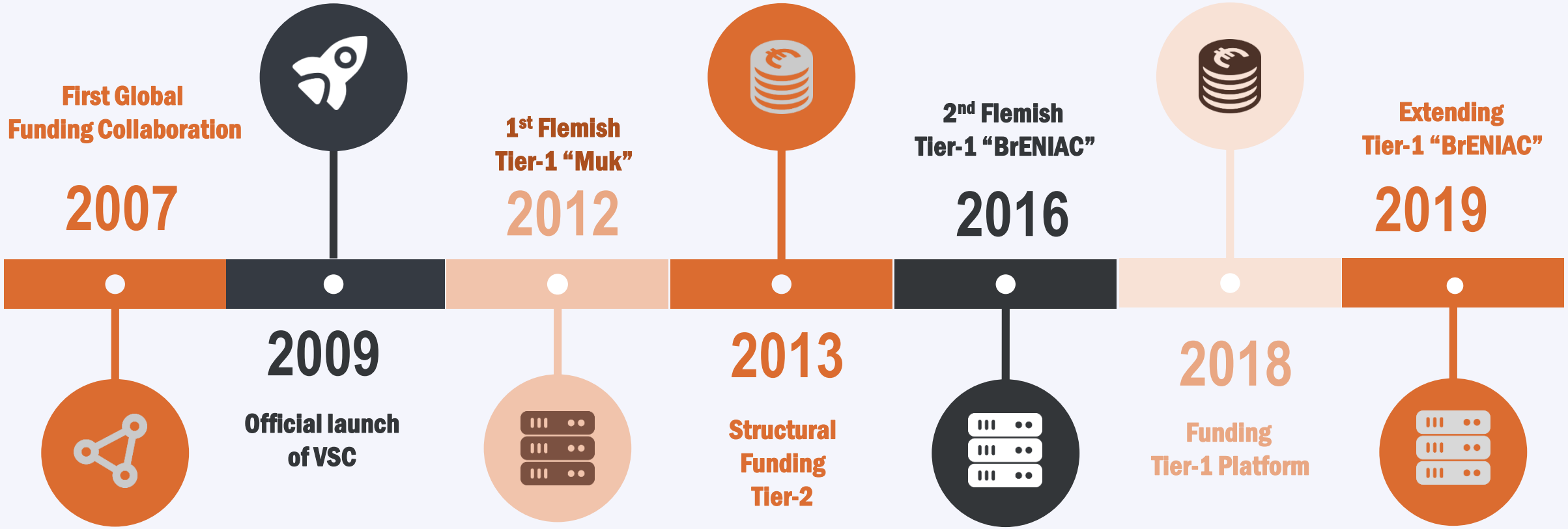
universitaire associatie BRUSSEL

ASSOCIATIE KU LEUVEN

ASSOCIATIE UNIVERSITEIT-HOGESCHOLEN LIMBURG

auha antwerp university association

VSC history


First Global Funding Collaboration
2007
2009
Official launch of VSC
1st Flemish Tier-1 "Muk"
2012
2013
Structural Funding Tier-2
2nd Flemish Tier-1 "BrENIAC"
2016
2018
Funding Tier-1 Platform
Extending Tier-1 "BrENIAC"
2019

VSC HPC Environments

Tier-0

PRACE

Vlaanderen
is supercomputing

Tier-1

BrENIAC

Tier-2

GHENT UNIVERSITY
STEVIN

VUB VRIJE UNIVERSITEIT BRUSSEL
HYDRA

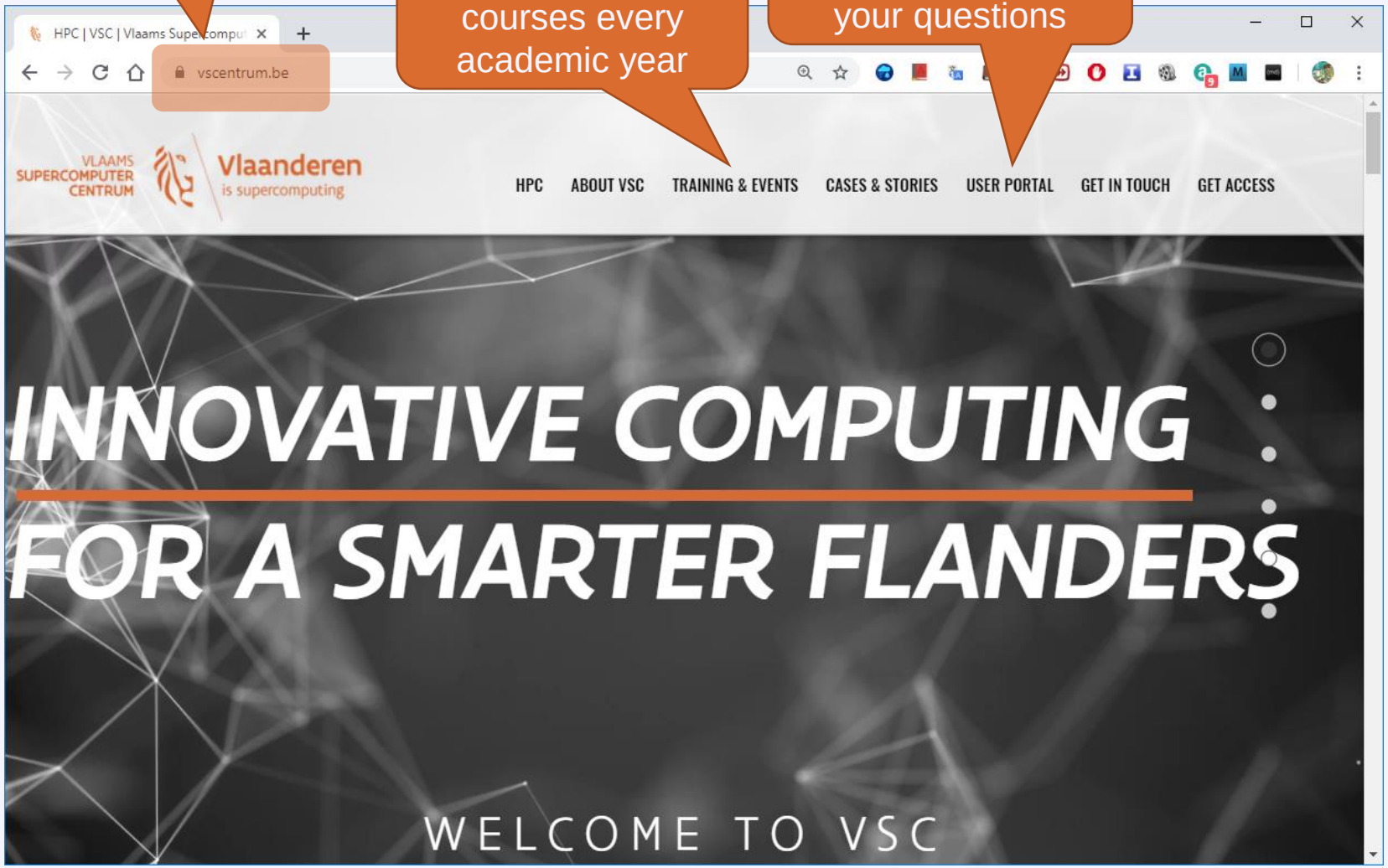
Universiteit Antwerpen
HOPPER/LEIBNIZ

KU LEUVEN UHASSELT
THINKING//GENIUS

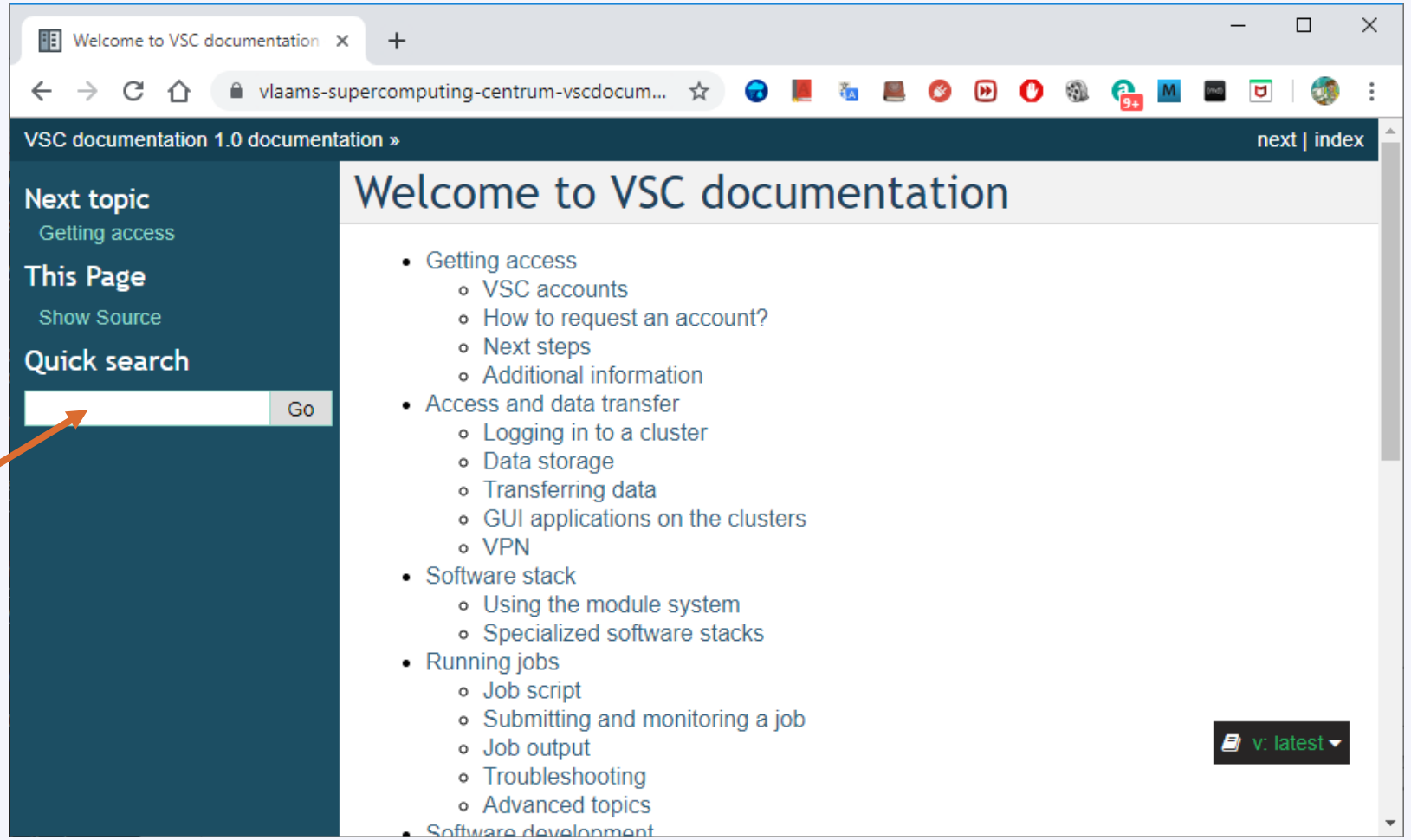
www.vscentrum.be

Very rich suite of courses every academic year

Documentation! Answers >80% of your questions



VLAAMS SUPERCOMPUTER CENTRUM



Search you keywords here; e.g. qsub

Support and Services

Basic support

- Helpdesk (hpcinfo@kuleuven.be)
- Monitoring and reporting

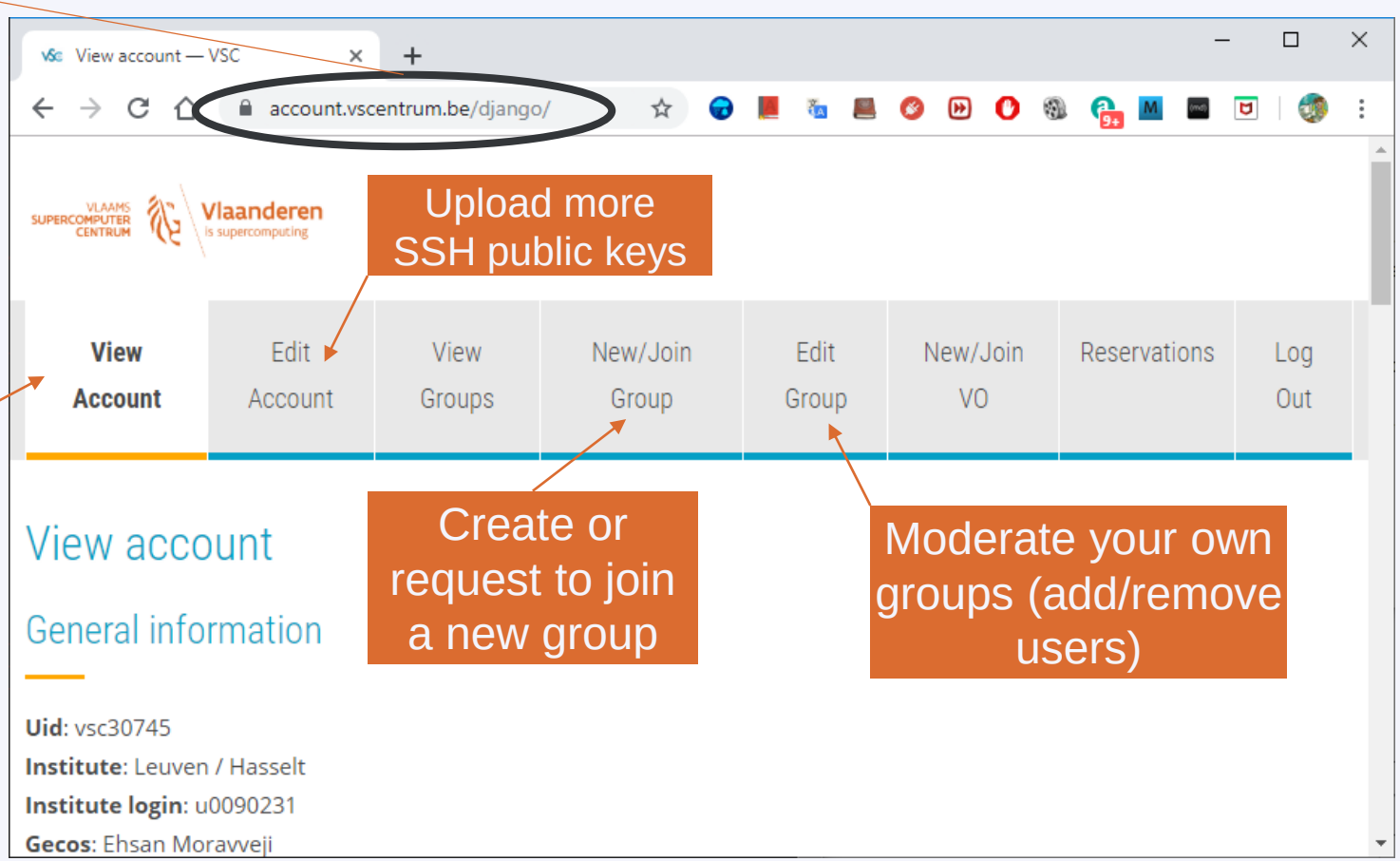
Application support

- Installation and porting
- Optimisation and debugging
- Benchmarking
- Workflows and best practices

Training

- Documentation and tutorials
- Scheduled trainings / workshops
- On request workshops
- One-to-one sessions

To manage your VSC account: account.vscentrum.be



Upload SSH Key
Request account

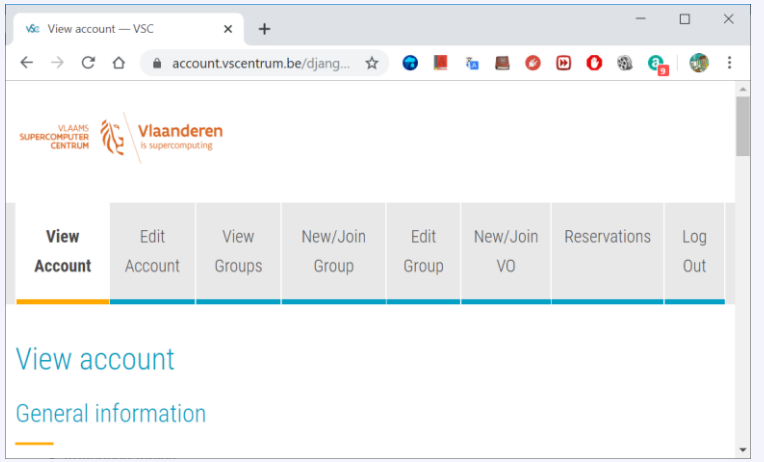
Create or request to join a new group

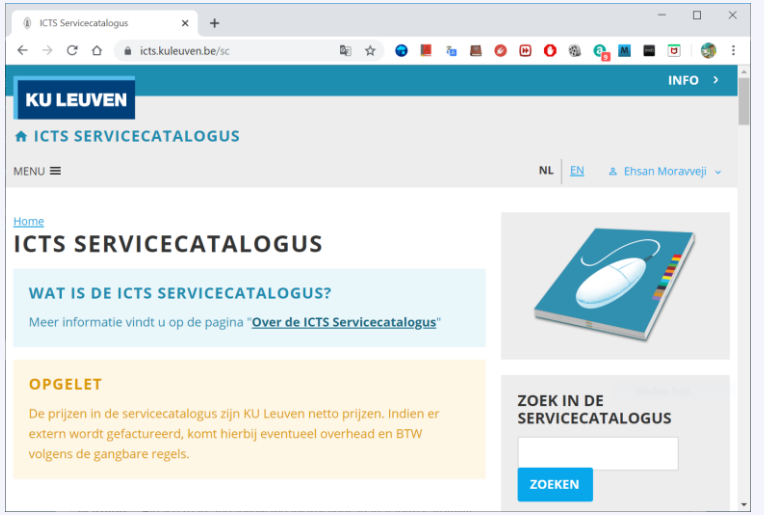
Moderate your own groups (add/remove users)

VLAAMS SUPERCOMPUTER CENTRUM

Become a VSC user

- ❑ Create a secure (4096 bit) SSH key pairs
  Upload it on the account page: www.account.vscentrum.be
- ❑ You need to request a VSC account
  Normally processed swiftly
- ❑ Request introductory credits (2000 free credits for 6 months)
- ❑ Request project credits (for supervisors and project leaders)
  You need to create a VSC group
  Add users to the group to give them access to use credits
  Fill out the request form
- ❑ Extra storage requests
  Scratch extension: free of charge
  Archive fileset: 70 € per TB per year
  Staging fileset: 130 € per TB per year
- ❑ All service costs (compute and storage) are all explained
  Go to ICTS service catalogus: https://icts.kuleuven.be/sc
  Click on High Performance Computing (NL/EN)

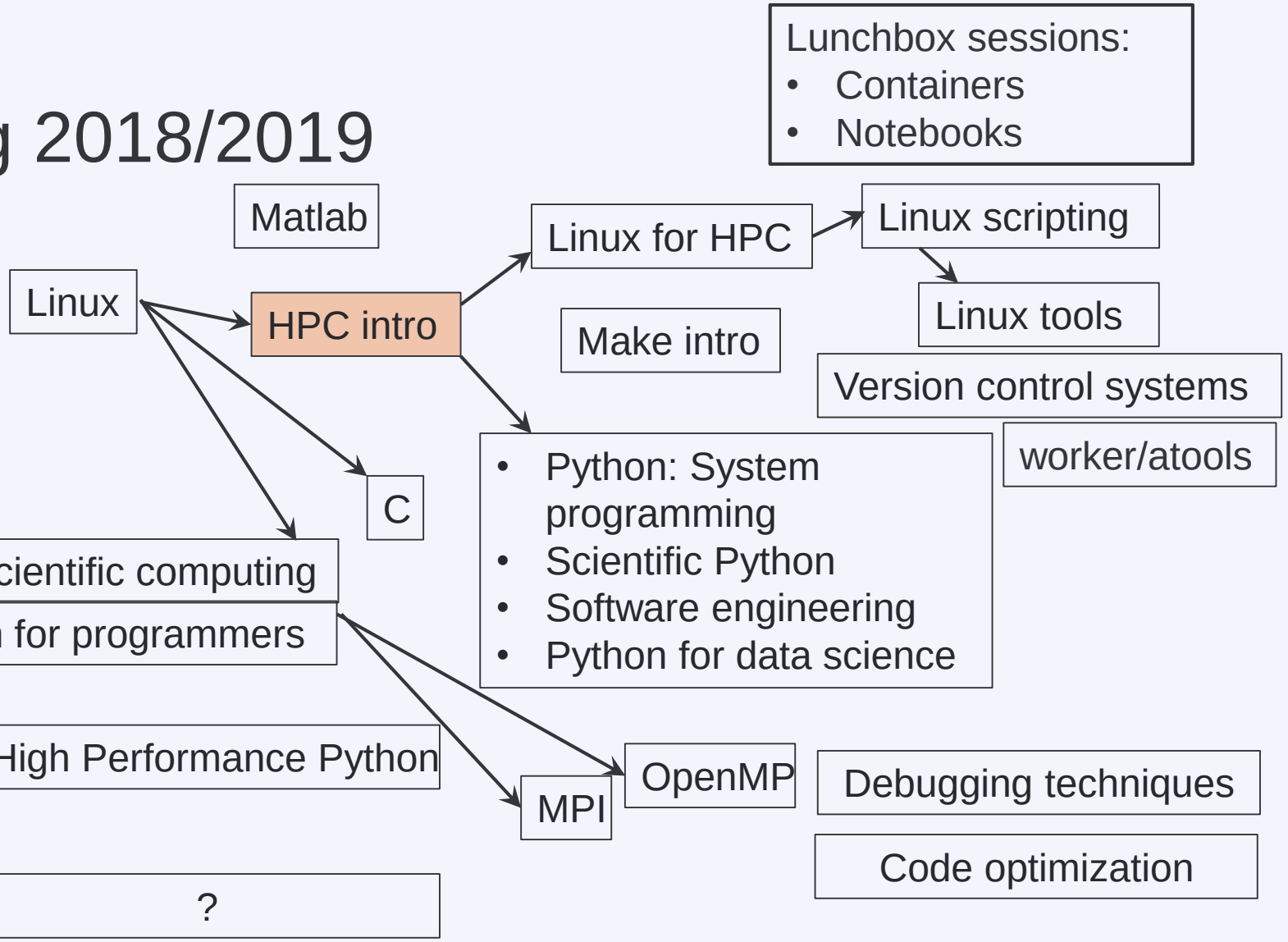




VLAAMS SUPERCOMPUTER CENTRUM

VSC training 2018/2019

- Introductory
- Intermediate
- Advanced
- Specialized track



Stay up-to-date https://www.vscentrum.be/en/education-and-trainings

To Acknowledge VSC in publications

Why?

- ❑ a contractual obligation for the VSC
- ❑ helps VSC secure funding
- ❑ you will benefit from it in the long run

At KU Leuven

- ❑ add the relevant papers to the virtual collection "High Performance Computing" in Lirias

In het nederlands

De rekeninfrastructuur en dienstverlening gebruikt in dit werk, werd voorzien door het VSC (Vlaams Supercomputer Centrum), gefinancierd door het FWO en de Vlaamse regering – departement EWI.

In English

The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government – department EWI.

Tier-2 Clusters

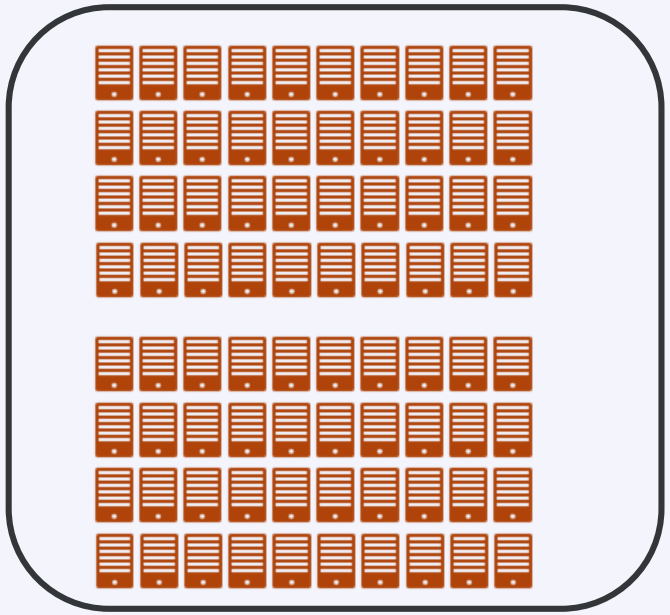
Tier-2 Clusters @ KU Leuven

ThinKing (since 2014)
352 nodes: 7,616 cores

Genius (since 2018)
250 nodes: 8,936 cores

Tier-2 Overview

Compute nodes

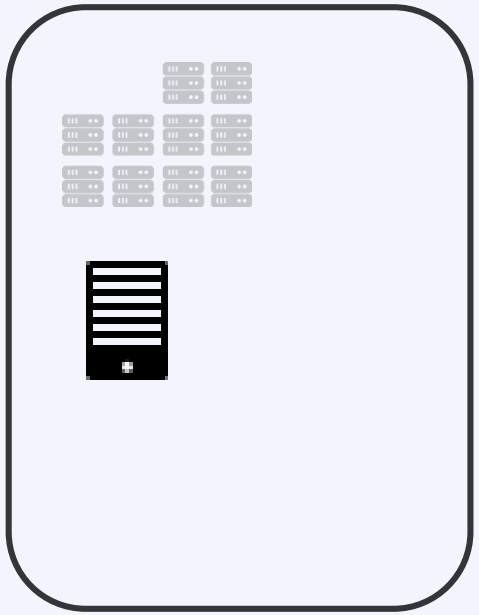


Thinking:
+ 176x + 32x Ivybridge 20c 64/128 GB

(EOL)
+ 48x + 86x Haswells 24c 64/128 GB

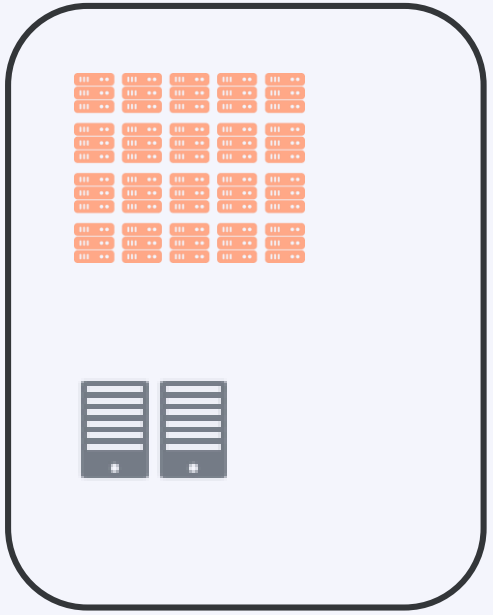
Genius:
+ 96x Skylake 36c 192 GB
+ 120x CascadeLake 36c 192 GB

Large memory nodes



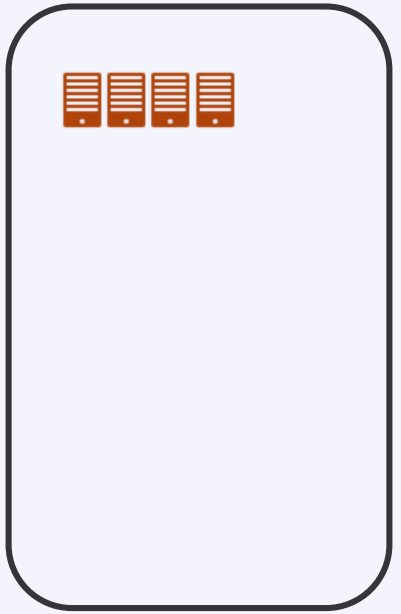
Genius:
+ 10x Skylake 36c 768GB
+ 1x Superdome 112c 6 TB

GPU nodes



Genius:
+ 20x Skylake 36c 192 GB
  4x P100 16GB
+ 2x CascadeLake 36c 192GB
  8x V100 32GB

Exp nodes



Genius:
4 AMD Naples 64c 256 GB

Technical Hardware Specifications

	Tier 2							Tier 1			
Cluster name	ThinKing				Genius			BrENIAC			
Processor type	Ivybridge		Haswell		SkyLake		Cascade Lake	Broadwell		SkyLake	
Cores per node	20		24		36		36	28		28	
Base Clock Speed	2.8 GHz		2.5 GHz		2.3 GHz		2.6 GHz	2.4 GHz		2.6Ghz	
Total nodes	176	32	48	96	86	10	120	580		408	
Node memory (GB)	64	128	64	96	192	768	192	128	256	192	
Memory per core (GB)	3.0	6.2	2.5	3.8	5.2	21.2	5.3	4.4	9.0	6.7	
Total cores	4,160		3,456		3,456		4,320	27,664			
Peak performance (Flops/cycle)	4 DP FLOPs/cycle: 4-wide AVX addition OR 4-wide AVX multiplication		8 DP FLOPs/cycle: 4-wide FMA (fused multiply-add) instructions AVX2		16 DP FLOPs/cycle: 8-wide FMA (fused multiply-add) instructions AVX-512		16 DP FLOPs/cycle: 8-wide FMA (fused multiply-add) instructions AVX-512	8 DP FLOPs/cycle: 4-wide FMA (fused multiply-add) instructions AVX2		16 DP FLOPs/cycle: 8-wide FMA (fused multiply-add) instructions AVX-512	
Network	Infiniband QDR 2:1		Infiniband FDR		Infiniband EDR		Infiniband EDR	Infiniband EDR			
Cache (L1 KB/L2 KB/L3 MB)	10x(32i+32d) / 10x256 / 25 MB		12x(32i+32d) / 12x256 / 30MB		18x(32i+32d) / 18x1024 / 25 MB		18x(32i+32d) / 18x1024 / 25 MB	14x(32i+32d) / 14x 256 / 35 MB		18x(32i+32d) / 18x1024 / 25 MB	

	Tier 2	Tier 1
Access	All researchers and students	Approved project experienced users
Start with	2,000 introduction credits	500 node days starting grant
Project Credits	Pay as you GO	Panel Review Project proposal Free
Computation Limit	None	500 - 5,000 node days
Application deadline	None	3 February 5 June 2 October
Max. Project Duration		8 months
Available walltime	1h, 24h, 72h, 7d	Max. 3 days

VLAAMS SUPERCOMPUTER CENTRUM

Genius

SkyLake (since 8/2018)
CascadeLake (since 1/2020)

Tier-2 Cluster: Genius

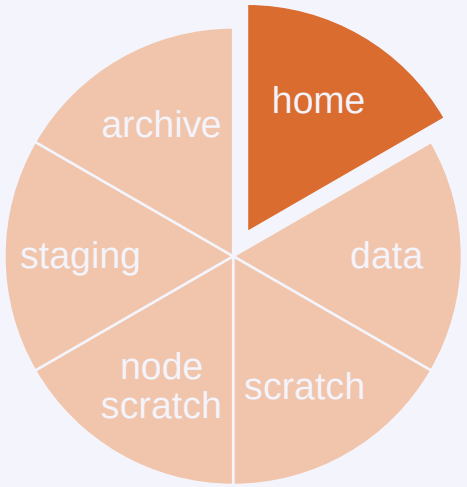
Type of node	CPU type	Inter-connect	# cores	installed mem	local discs	# nodes
SkyLake	Xeon 6140	IB-EDR	**36**	**192 GB**	800 GB	**86**
SkyLake **large mem**	Xeon 6140	IB-EDR	**36**	**768 GB**	800 GB	**10**
SkyLake **GPU**	Xeon 6140 **4xP100** SXM2	IB-EDR	**36**	**192 GB**	800 GB	**20**
CascadeLake	Gold 6240	IB-EDR	**36**	**192 GB**	800 GB	**120**
CascadeLake **GPU**	Gold 6240 **8xV100** SMX2	IB-EDR	**36**	**192 GB**	800 GB	**2**
SkyLake **Superdome**	Gold 6132	Flex Grid	**14**	**6 TB**	6 TB	**8**

Storage

Overview of the storage infrastructure

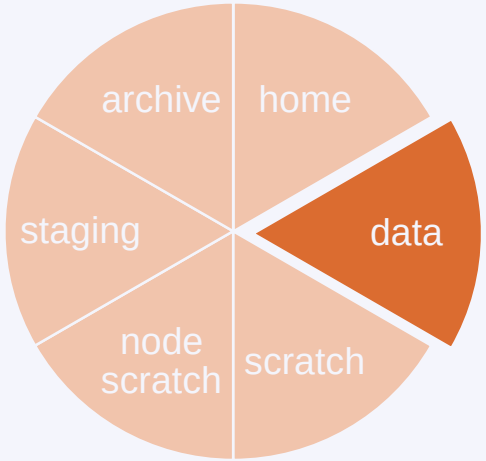
- Your files are owned only by you.
  Other VSC users have no permission to read/write/execute your files (POSIX)
- A VSC account has 3 default storages (free of charge)
  - `$VSC_HOME`
  - `$VSC_DATA`
  - `$VSC_SCRATCH`
- You can additionally request `staging` and `archive` storages
- Different storage volumes have different:
  - mount point
  - size and performance
  - use case
  - backup and maintenance policy
- More info on ICTS Service Catalog (EN/NL)

Storage



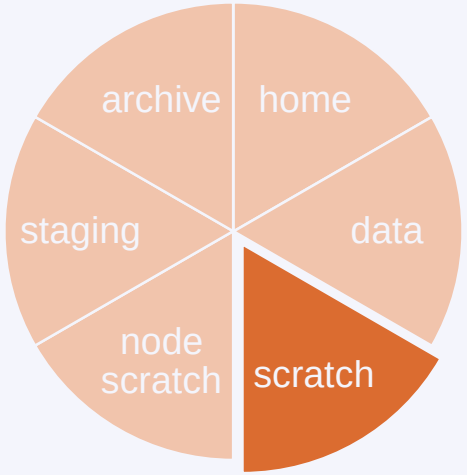
Storage	home folder
Env. Variable	`$VSC_HOME`
Filesystem Type	NFS
Access	Global
Backup	Hourly, daily, weekly (until last month)
Default Quota	3 GB
Extension	Not possible
Usage	Only storing SSH keys, config files
Remarks	- Stay away from using it - Can easily overflow: + Your jobs may crash + Login issues

Storage



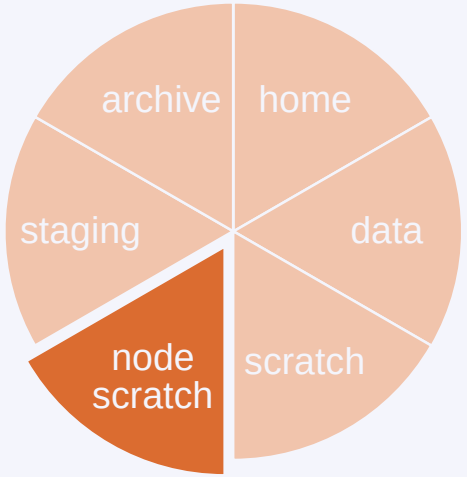
Storage	data folder
Env. Variable	`$VSC_DATA`
Filesystem Type	NFS
Access	Global
Backup	Hourly, daily, weekly (until last month)
Default Quota	75 GB
Extension	On purchase
Usage	Your data, code, software, results
Remarks	- Permanent storage for initial/final results - Not optimal for intensive or parallel I/O

Storage



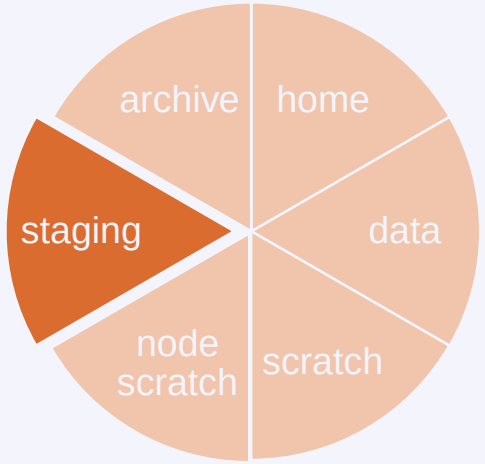
Storage	scratch folder
Env. Variable	`$VSC_SCRATCH`
Filesystem Type	GPFS
Access	Global
Backup	delete after 28 days from last access
Default Quota	100 GB
Extension	Free
Usage	Intensive, parallel I/O, temporary files
Remarks	- Recommended storage for all jobs - Copy scratch files to VSC_DATA or local storage after jobs are done - Deleted files cannot be recovered

Storage



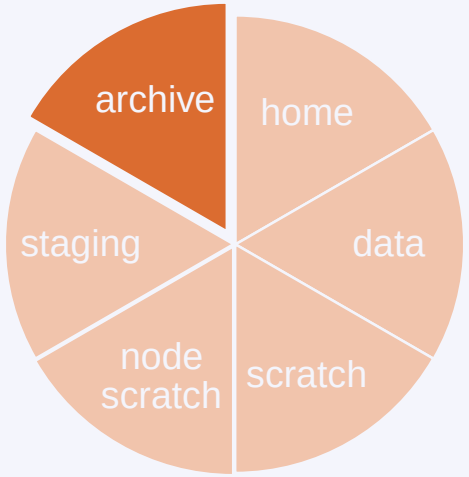
Storage	Node scratch folder
Env. Variable	`$VSC_SCRATCH_NODE`
Filesystem Type	GPFS
Access	On compute node, only at runtime
Backup	None
Default Quota	200 GB
Extension	Read about beeOND
Usage	Temporary storage at runtime
Remarks	- Fastest I/O, attached to the node - Is cleaned after job terminates - Copy the data to your home, scratch, or staging before job ends

Storage



Storage	Staging folder
Path	`/staging/leuven/stg_000XX`
Filesystem Type	GPFS
Access	On demand, only Tier-2@KUL
Backup	None
Default Quota	None
Extension	On purchase, from 1 TB
Usage	Permanent; share with a group
Remarks	- Accessible from login/compute nodes - Fast, parallel I/O

Storage



Storage	Archive folder
Path	`/archive/leuven/arc_000XX`
Filesystem Type	NFS
Access	On demand, only Tier-2@KUL
Backup	snapshots
Default Quota	None
Extension	On purchase, from 1 TB
Usage	Permanent; share with a group
Remarks	- Accessible only from login nodes - Cannot use it from compute nodes - Slow, but durable filesystem

Storage

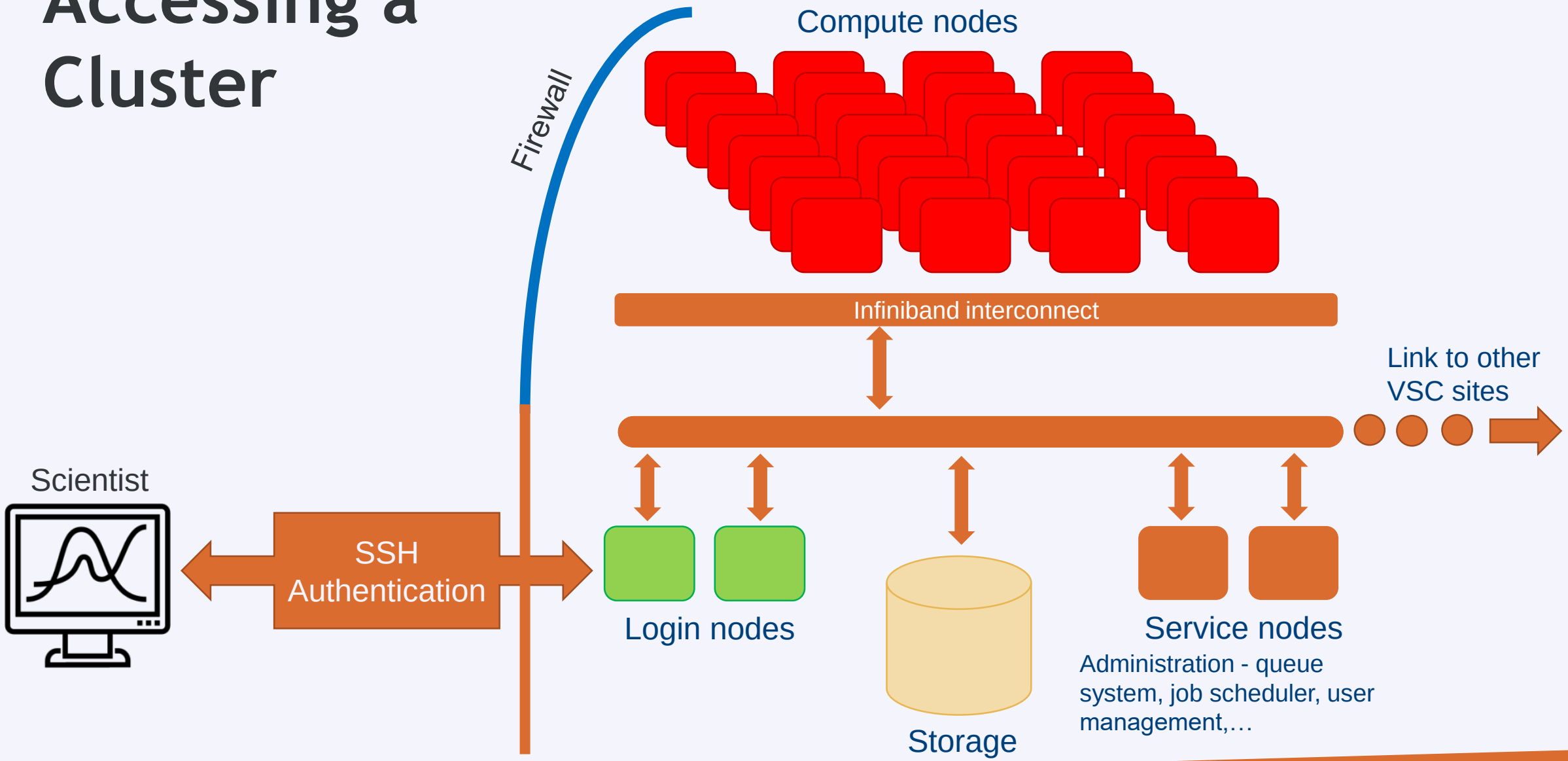
- Request form for extra storage
- More information
- Do not use `/tmp`
  It is only 10 GB and is reserved for the OS and root processes
  Your application can crash if using `/tmp`
- You are automatically logged into your home folder upon login.
  Make sure you immediately go to your other storages, e.g.
  `$> cd $VSC_DATA`
- Always check your storage balance using `myquota` command

Example

```
$> myquota
file system $VSC_HOME
    Blocks: 1479M of 3072M
    Files: 12934 of 100k
file system $VSC_DATA
    Blocks: 102G of 225G
    Files: 1043k of 10000k
file system $VSC_SCRATCH
    Blocks: 15M of 1.5T
```

Login Nodes

Accessing a Cluster



Using Login Nodes

Tips

- ❑ To develop and/or compile code and/or software
- ❑ To check your storage and credit balance
- ❑ To manage jobs (submit, check status, debug, resubmit, …)
- ❑ To move data around
  within VSC: use data, scratch, staging, archive
  outside VSC: copy/sync from/to your local storage
- ❑ To pre-process or post-process your data/jobs
- ❑ To visualize your data
- ❑ To share files/folders

Warning

- ❑ Login nodes are shared resources
- ❑ **Do not** execute heavy-lifting tasks (core, memory)
- ❑ Instead, submit jobs

Login Hosts on Different Machines/partitions

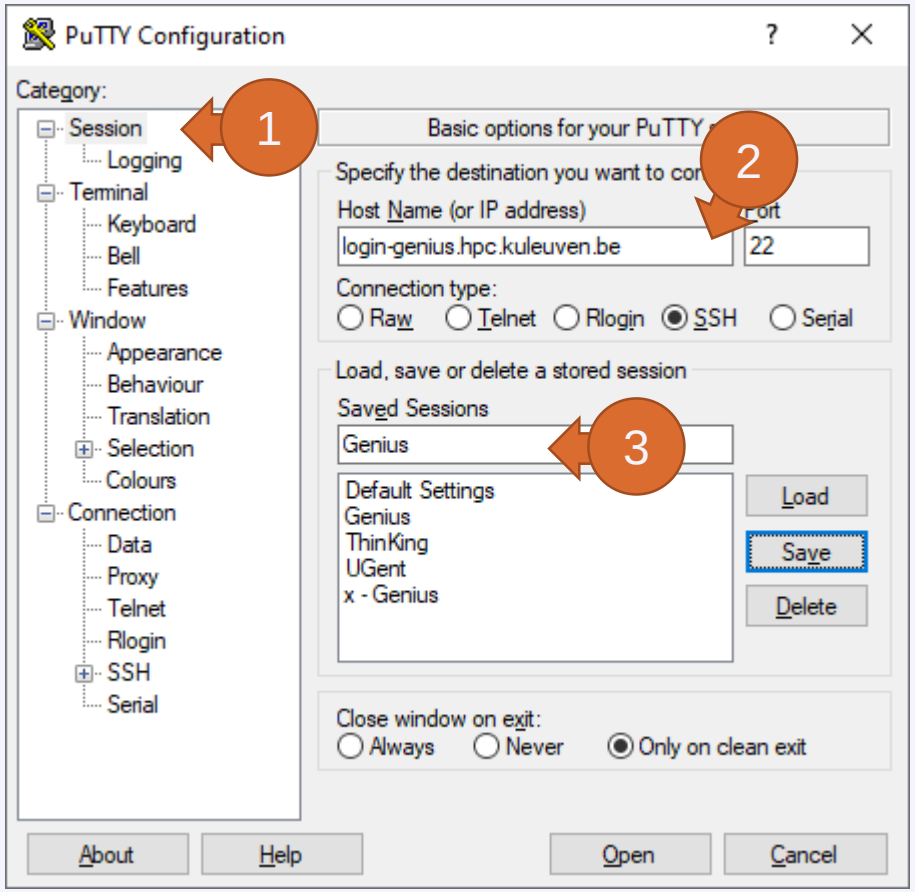
- Windows: PuTTY or MobaXterm or NX
  Linux/Mac: terminal or NX
- To login, you need an active VSC number and a hostname
  `$> ssh -X vscXXXXX@<hostname>`

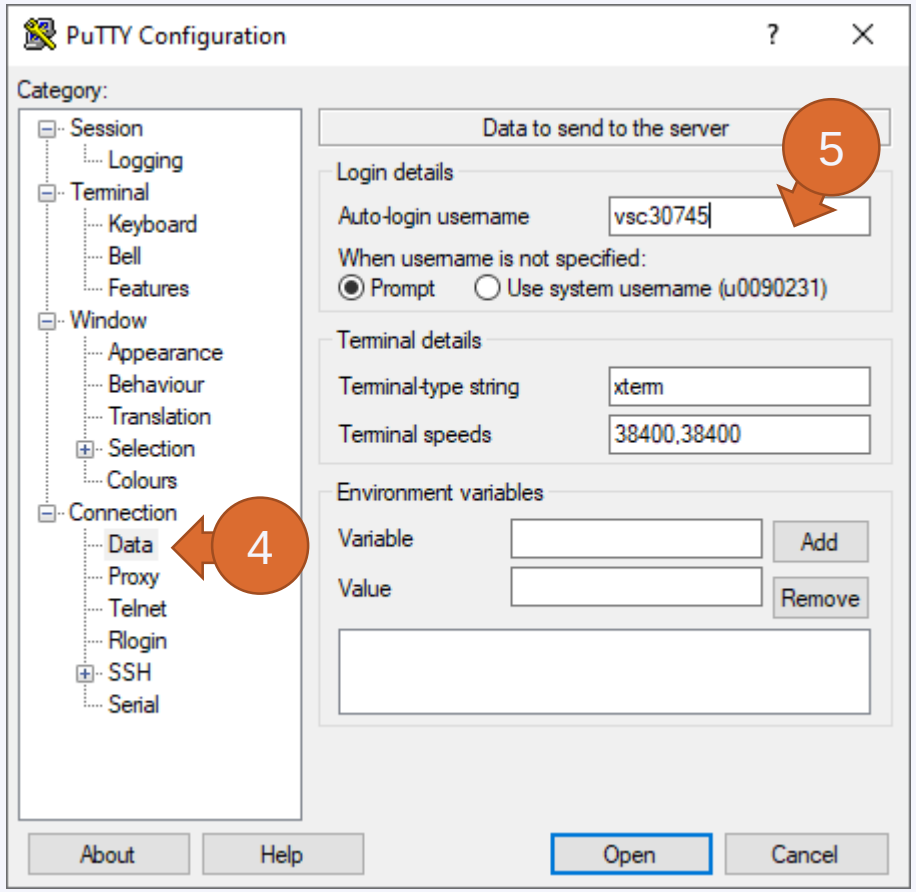
Cluster / Partition	<hostname>	Remark(s)
ThinKing	`login-thinking.hpc.kuleuven.be`	Recommended
	`login7-tier2.hpc.kuleuven.be` `login8-tier2.hpc.kuleuven.be`	Haswell partition
Genius	**`login.hpc.kuleuven.be`** `login-genius.hpc.kuleuven.be`	Recommended
	`login{1,2}-tier2.hpc.kuleuven.be`	No GPU
	`login{3,4}-tier2.hpc.kuleuven.be`	Nvidia Quadro P6000
Superdome	Any Genius login node (above)	`module load superdome;` `partition=superdome`
AMD	Any Genius login node (above)	`Partition=amd`

Login Setup

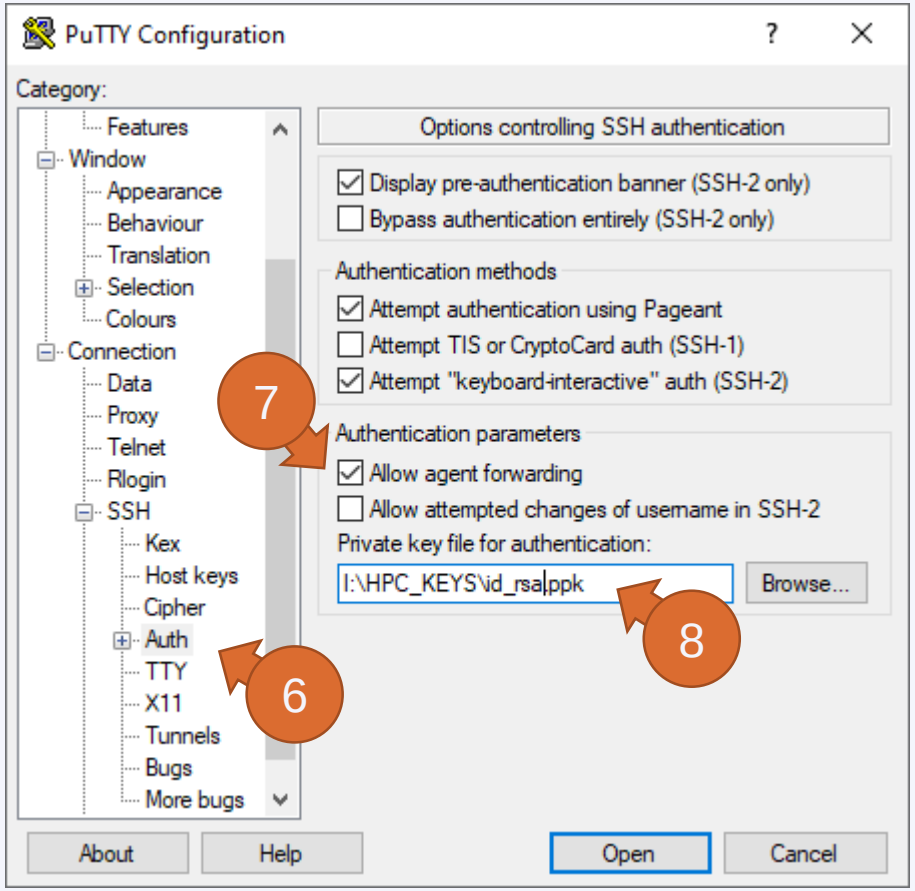
- ❑ PuTTY
- ❑ Terminal

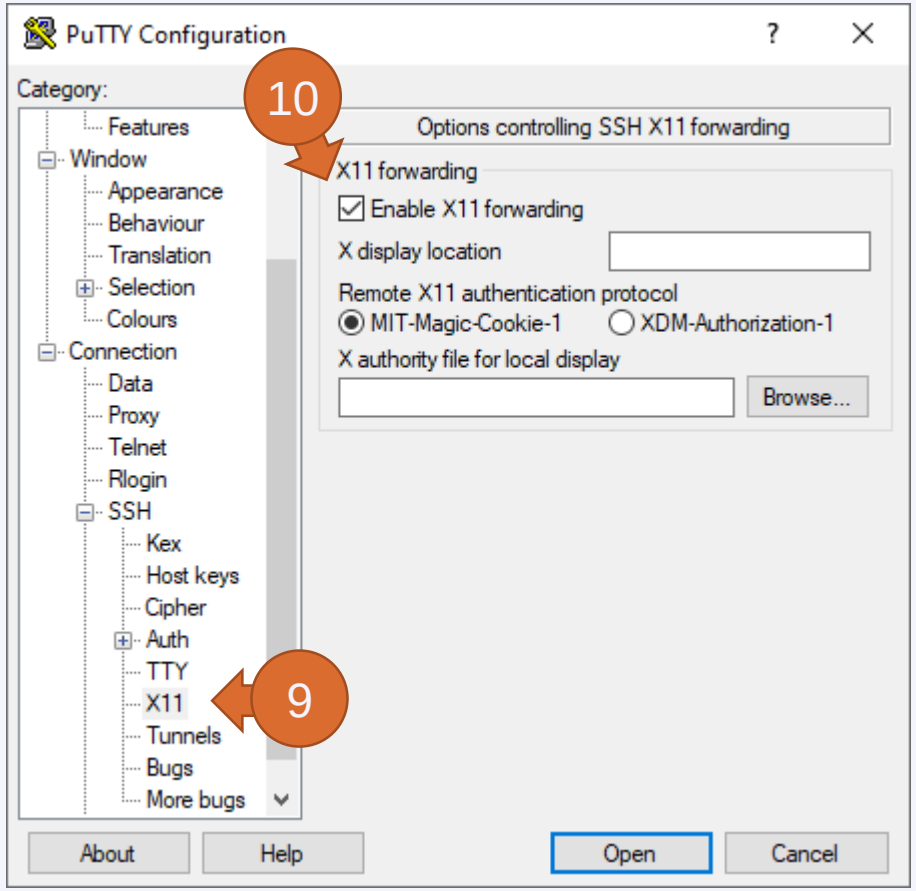
Setup PuTTY in 12 Clicks
(Windows only)





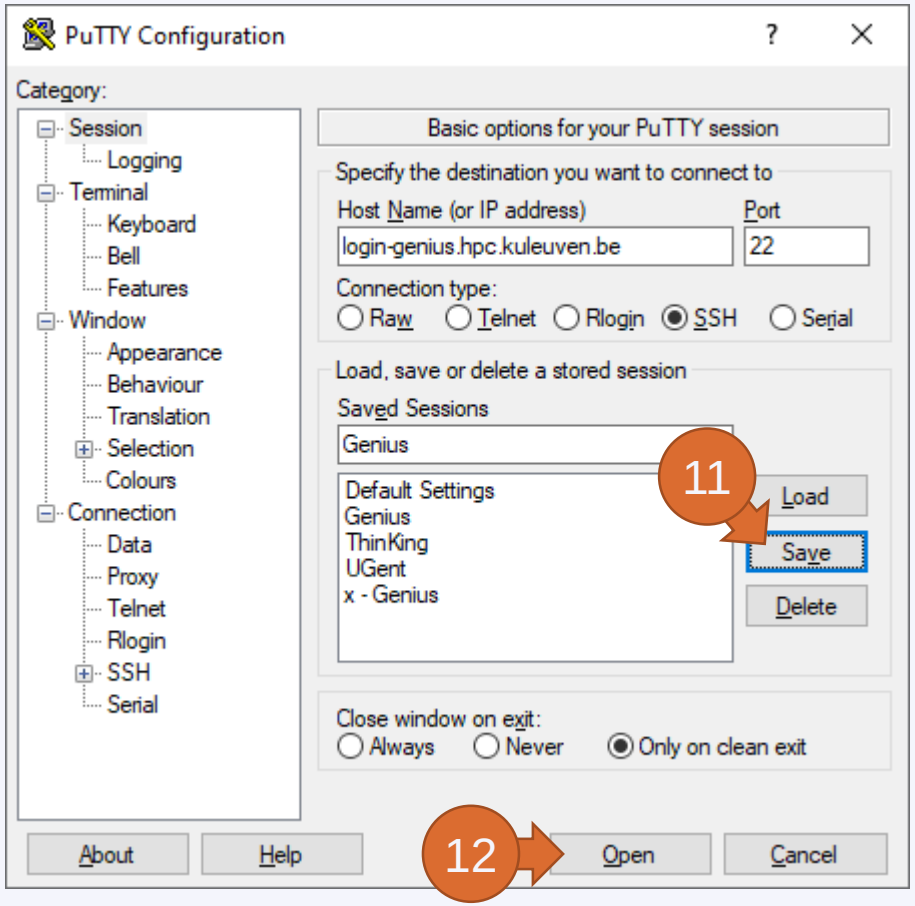
Setup PuTTY in 12 Clicks
(Windows only)

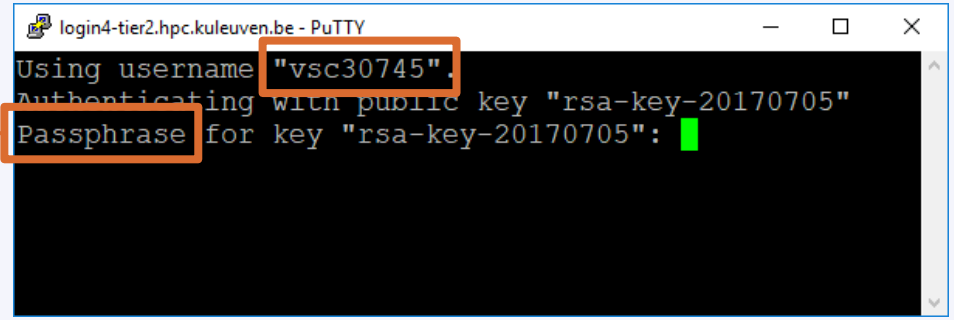


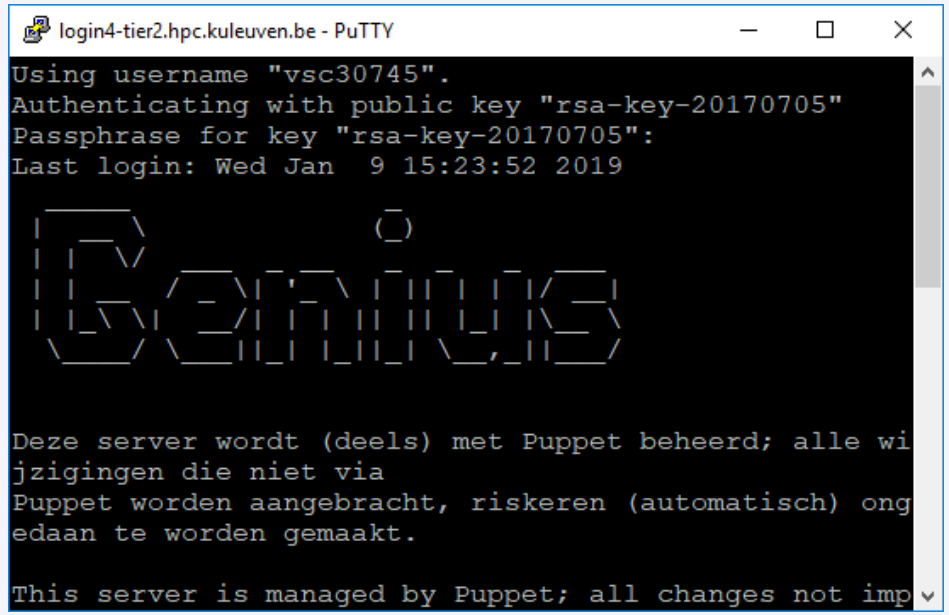


Setup PuTTY in 12 Clicks
(Windows only)

If PuTTY asks for **password**, exit immediately, and check the path to your private key. Else you will be blocked for 24h.



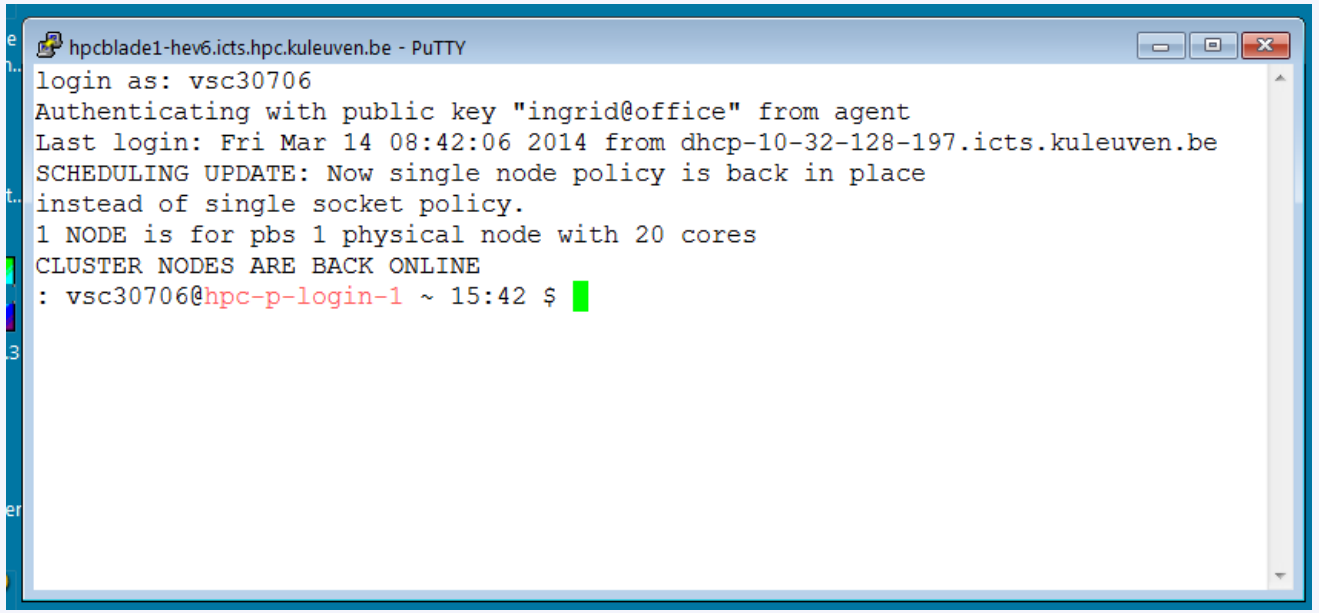




Connecting via Terminal
(Linux and Mac)

- Check your SSH Agent. Is your SSH key found?
  `$> ssh-add -l`
- If your key is not found, add it to the agent:
  `$> ssh-add </path/to/my/private/keyfile>`
- Try to ssh now
  `$> ssh vscXXXXX@<hostname>`

If asked for **password**, please stop connecting and contact suport, otherwise after a few attempts you will be blocked for 24h.



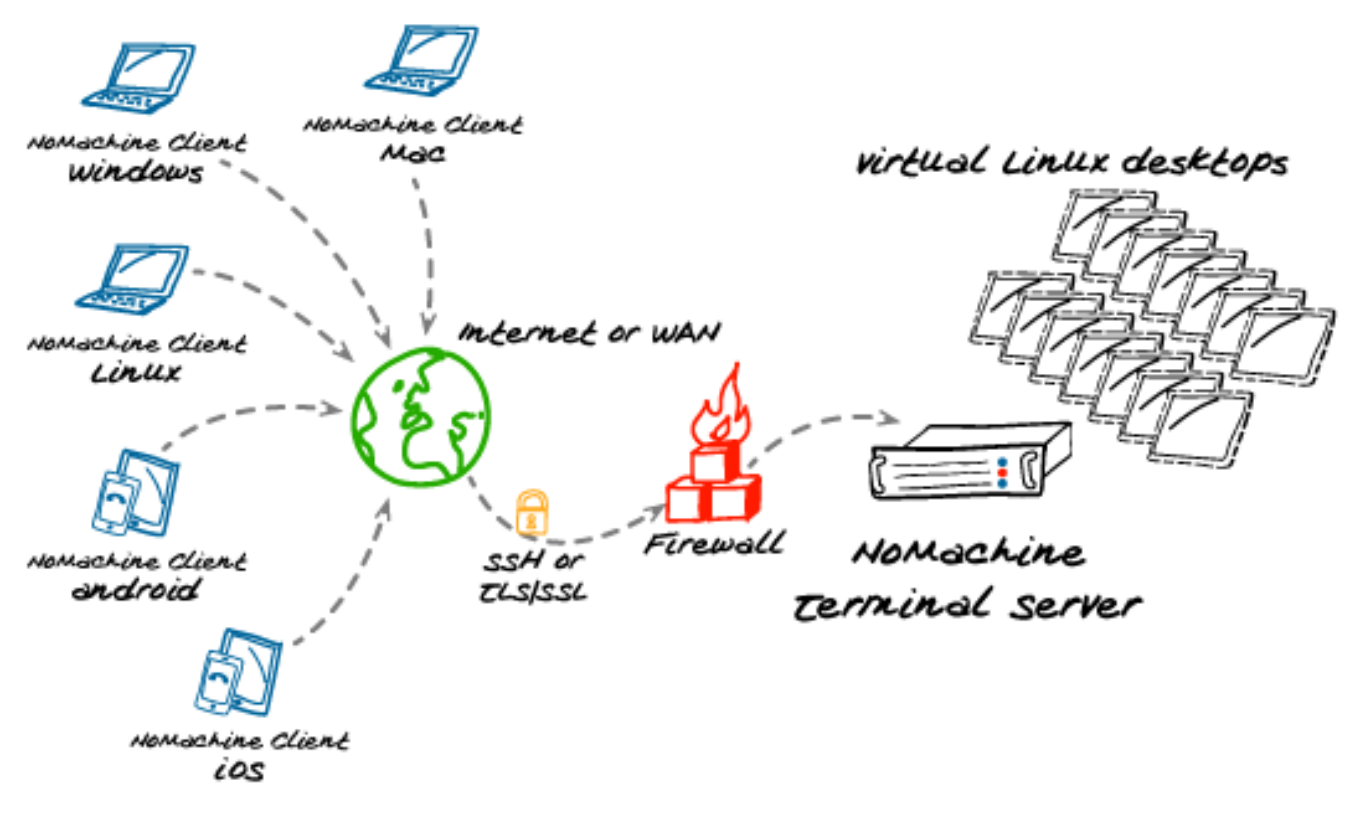
NX

- ❑ About
- ❑ Setup

NX – The Graphical Login

NX is:

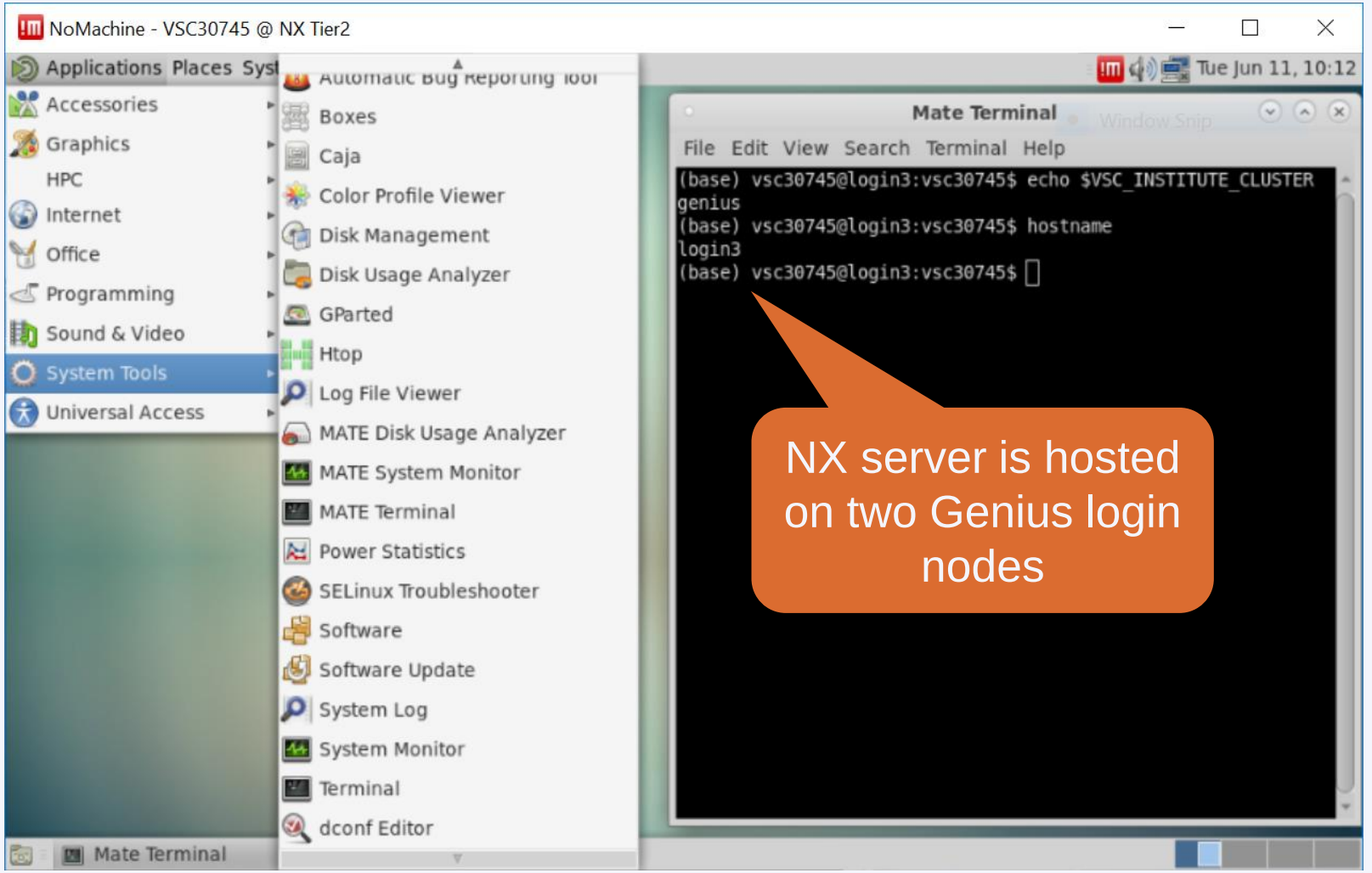
- ❑ a graphical remote desktop server
- ❑ login to VSC
- ❑ start a virtual Linux instance(s)
- ❑ Client: NoMachine
- ❑ Configuration Guide
- ❑ Using OpenSSH key
  Convert your key



Advantages of NX

- ❑ Graphical login
- ❑ A session stays alive/active even if you close your client (e.g. laptop)
- ❑ Resume a session, and immediately keep working where you left off
- ❑ More interactive jobs
- ❑ Available apps:
  - Internet browser, terminal, …
  - Text editor, PDF reader, Image viewer, …
  - Matlab, RStudio, SAS
- ❑ Remark:
  - NX is shared among tens of users
  - **Do not compute/test your code there**
  - Instead, submit jobs from NX to compute nodes

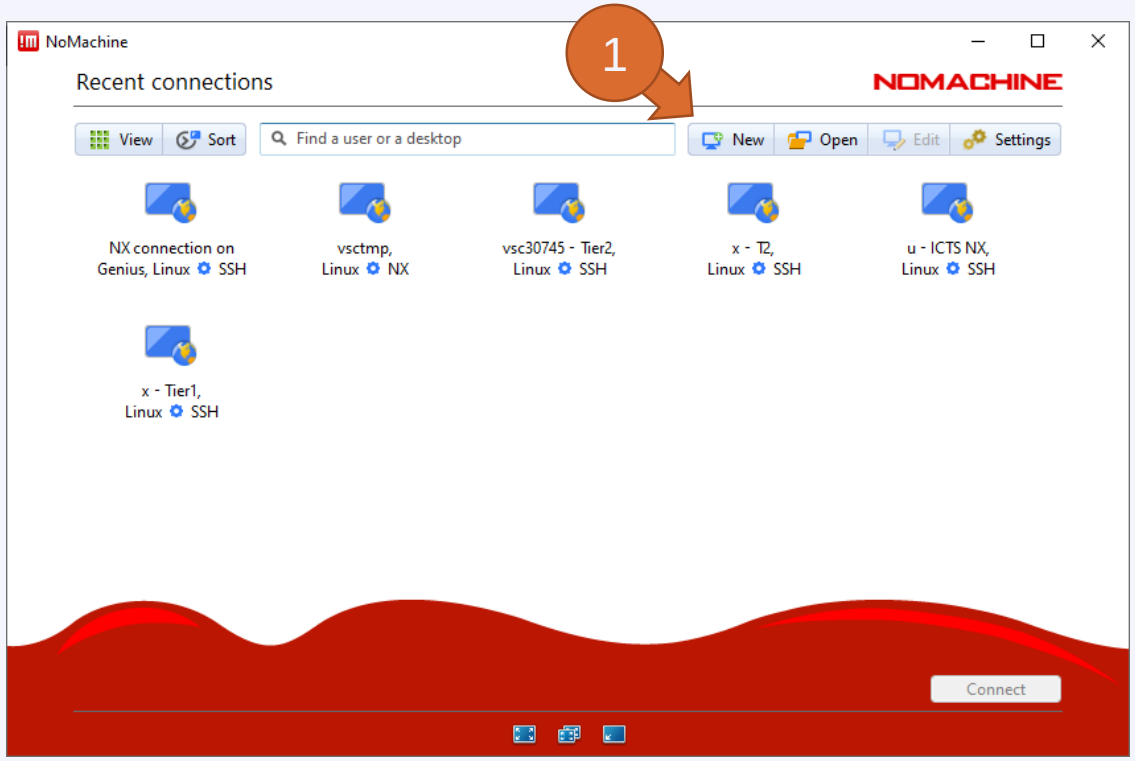
NX virtual desktop

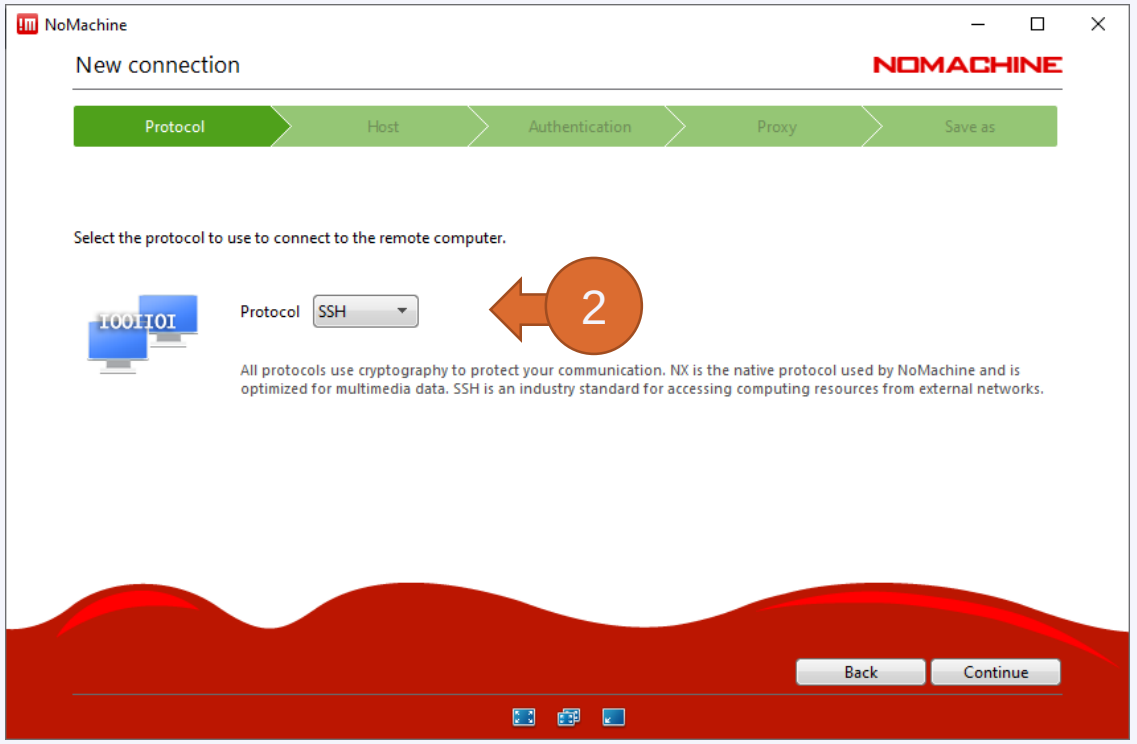


NX: available software

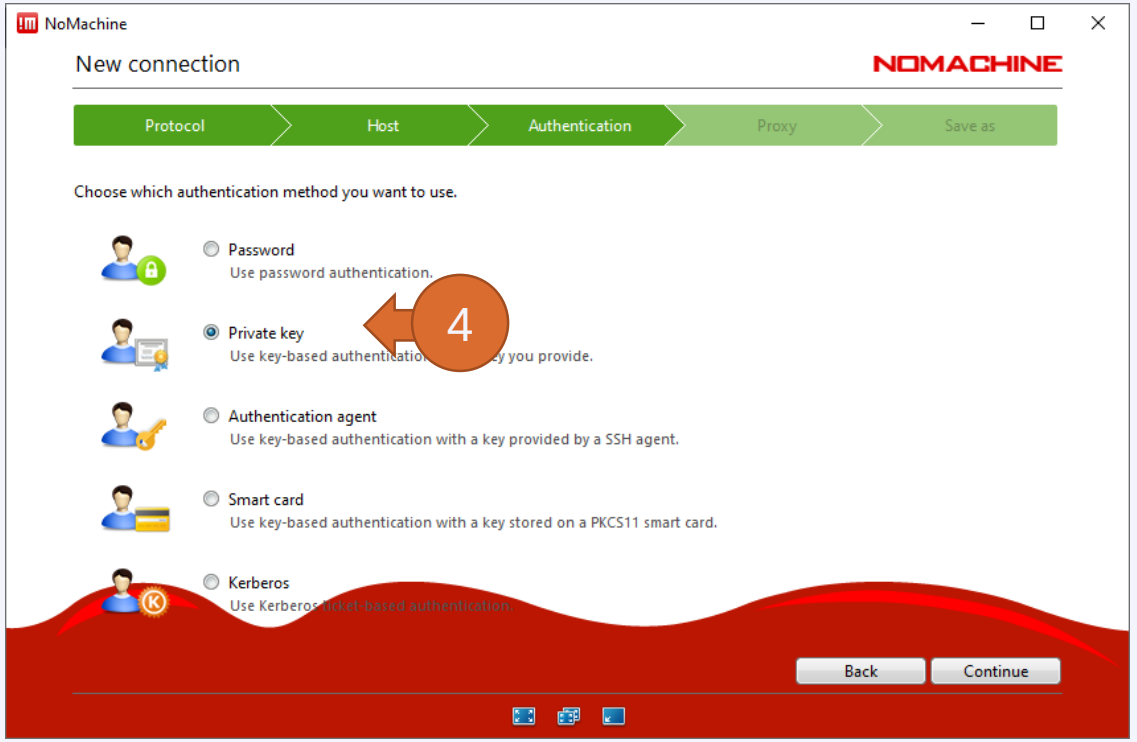
- **Accesories**: Gedit, Vi IMproved, Emacs (dummy version), Calculator
- **Graphics**: gThumb (picture viewer), Xpdf Viewer
- **Internet**: Firefox
- **HPC**: **Computation**: Matlab (2018a), RStudio, SAS
- **Visualisation**: Paraview, VisIt, VMD
- **Programming**: Meld Diff Viewer (visual diff and merge tool)
- **System tools**: File Browser, Terminal
- **Additionally**: Gnuplot (graphing utility), Filezilla (file transfer tool), Evince (PDF, PostScript, TIFF, XPS, DVI Viewer)
- Software launched though modules from Terminal.

Setup NX in 10 Steps

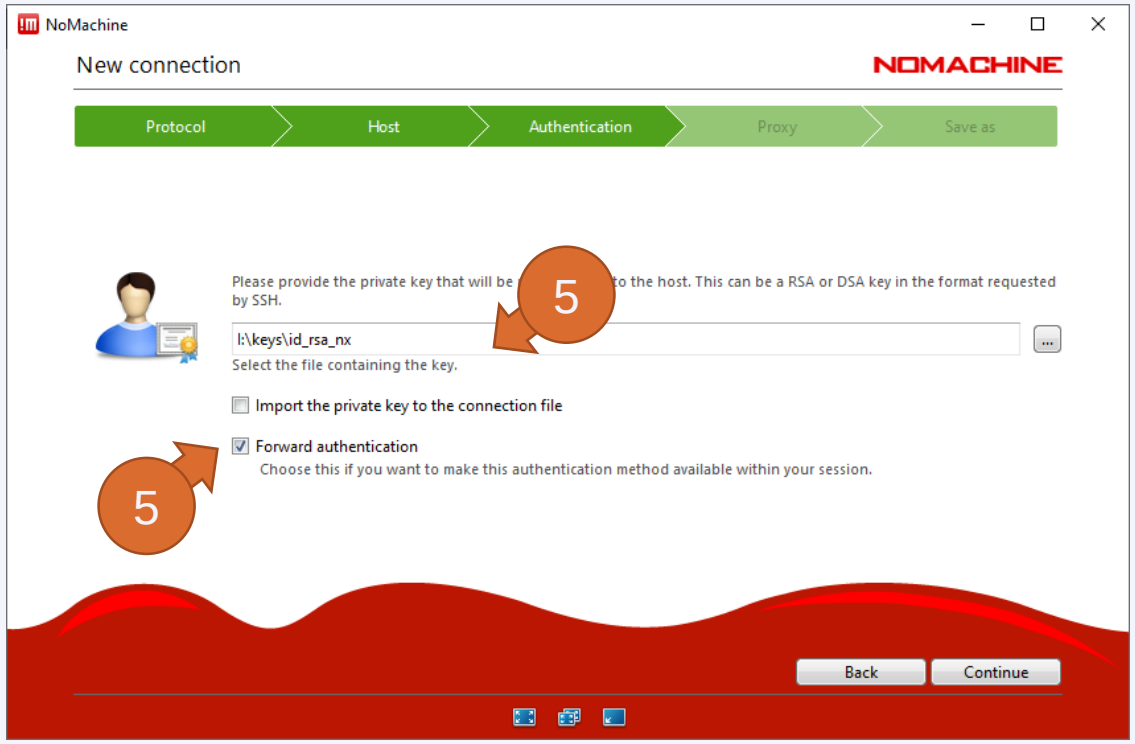


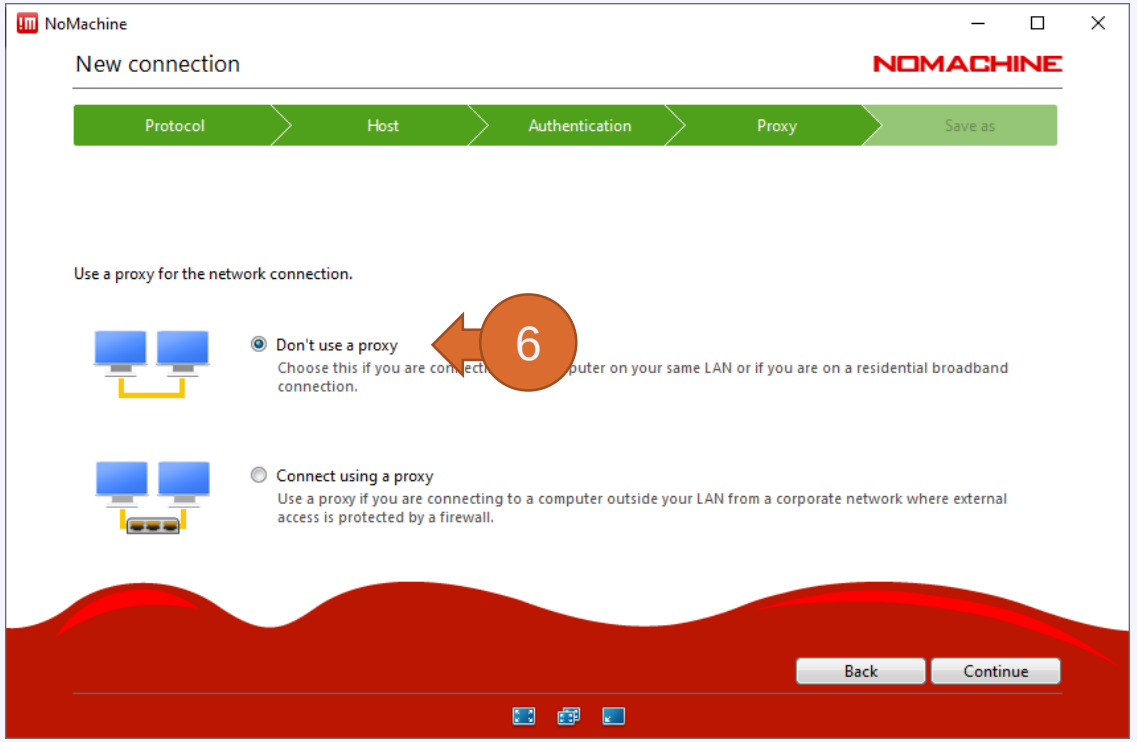


Setup NX in 10 Steps

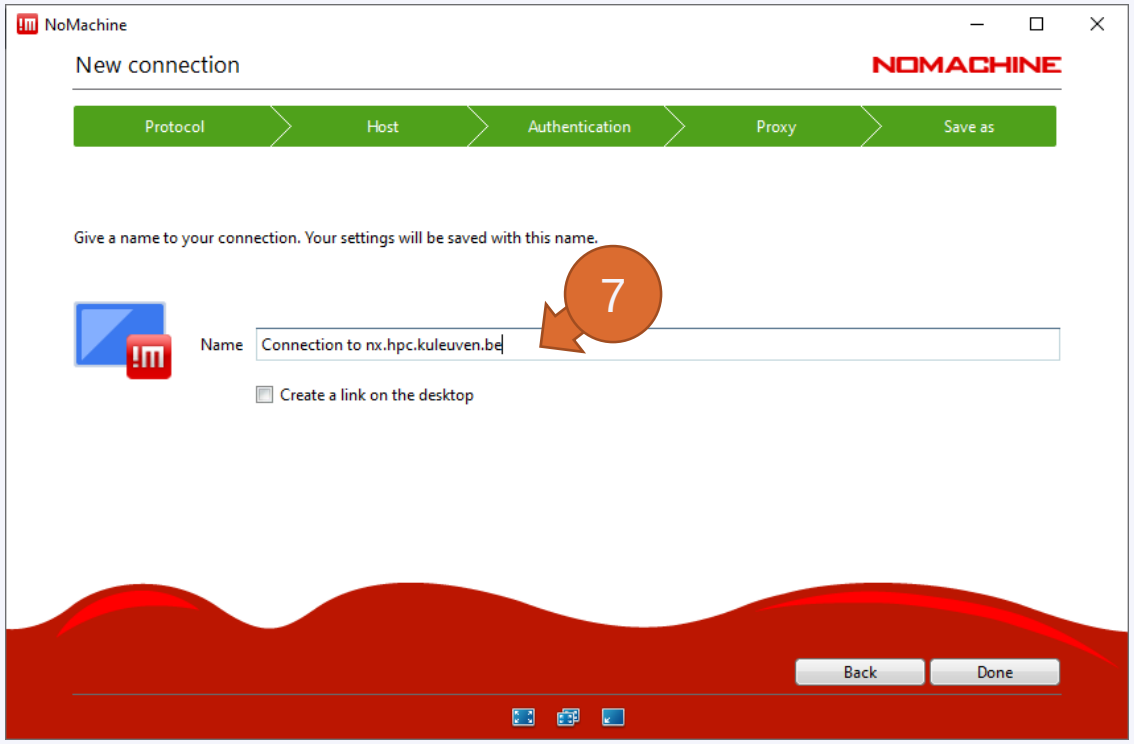


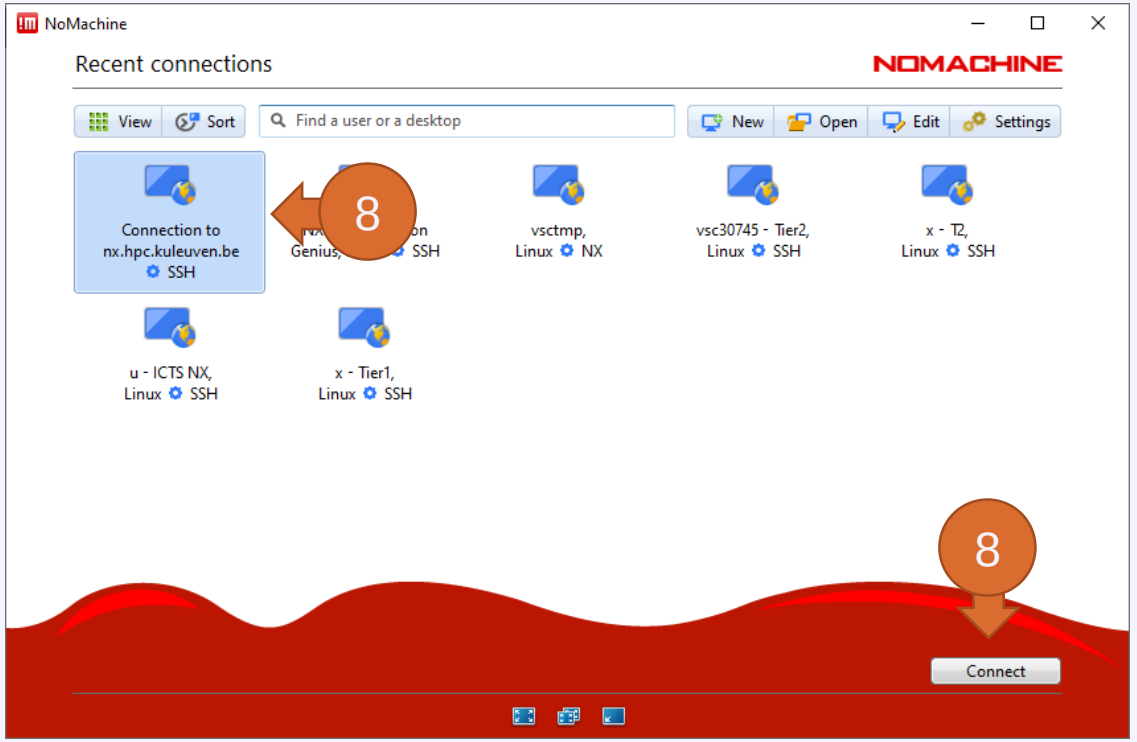
Setup NX in 10 Steps



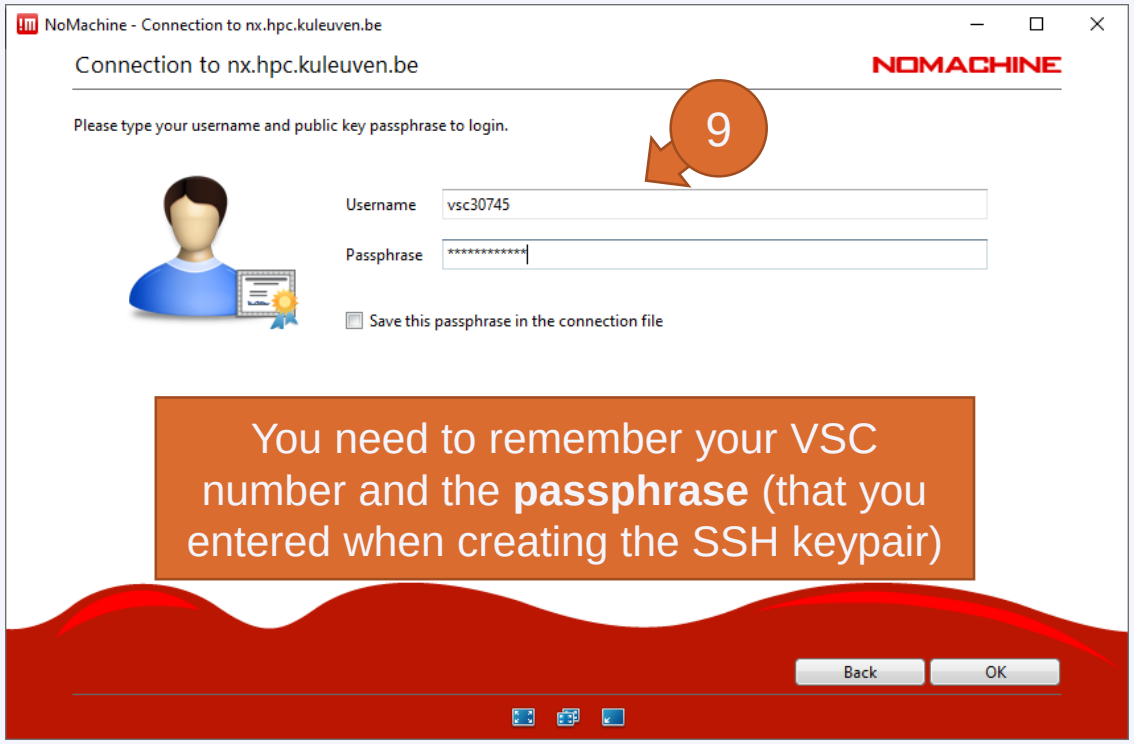


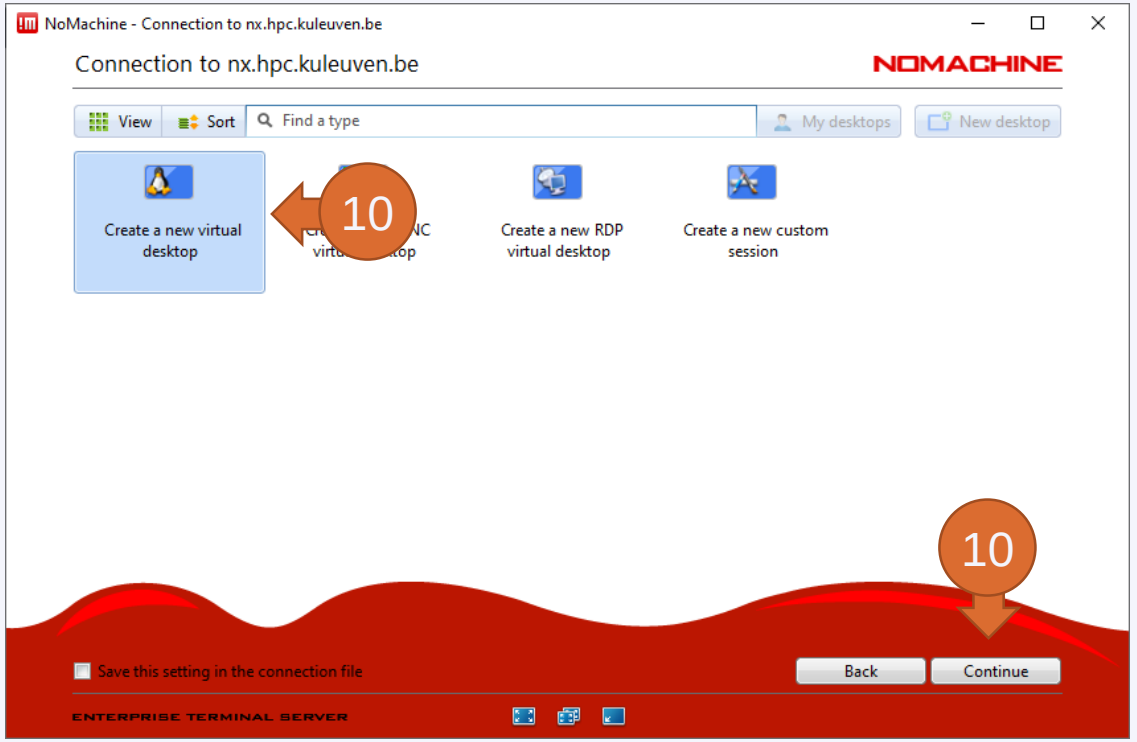
Setup NX in 10 Steps

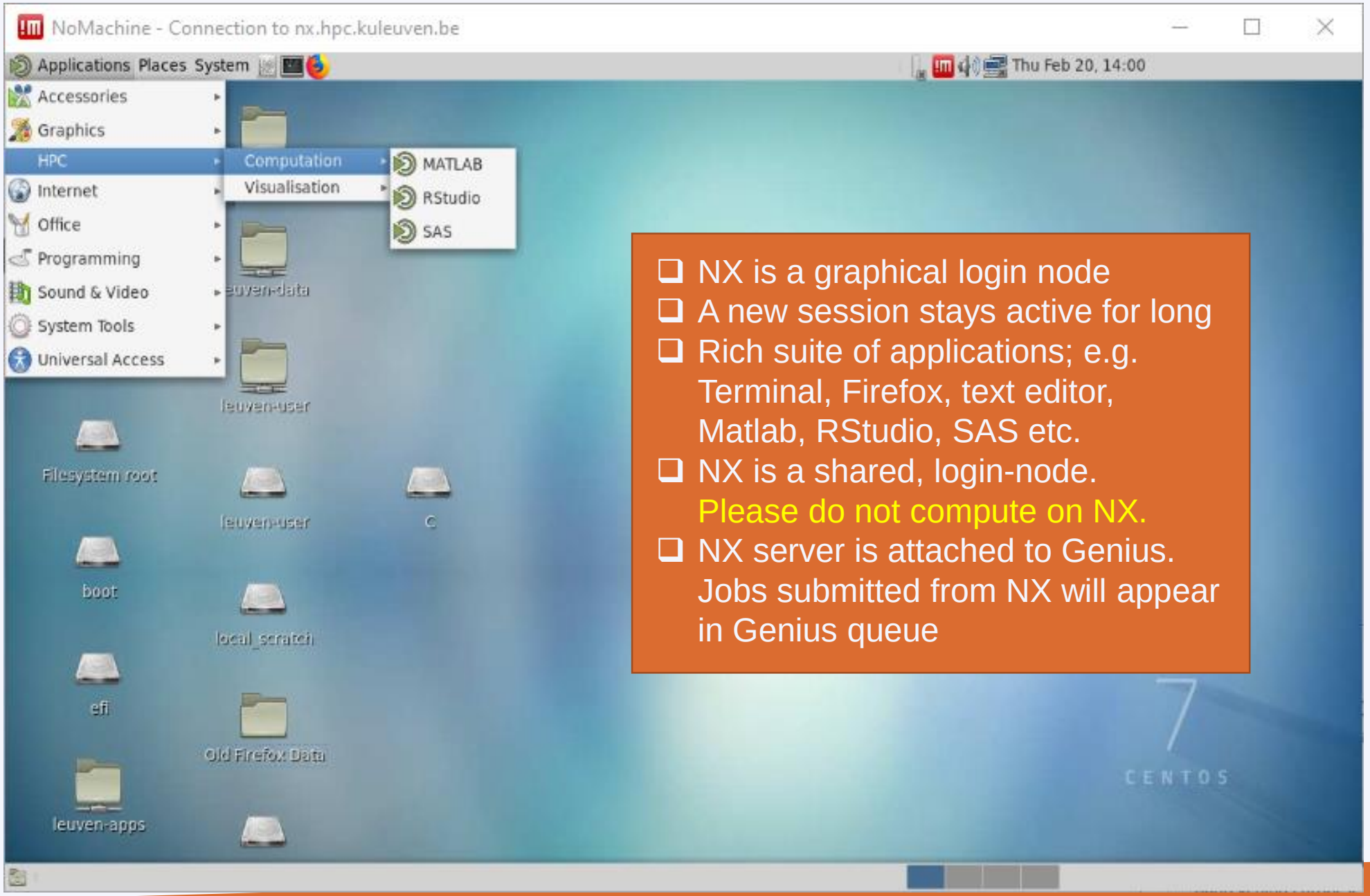




Setup NX in 10 Steps



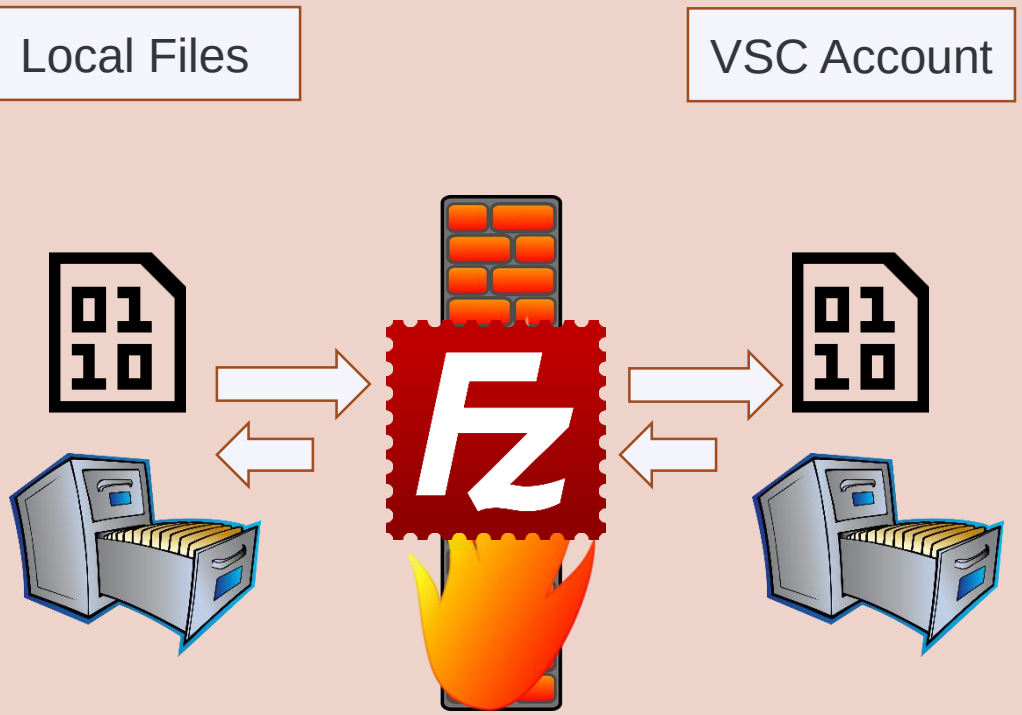




- NX is a graphical login node
- A new session stays active for long
- Rich suite of applications; e.g. Terminal, Firefox, text editor, Matlab, RStudio, SAS etc.
- NX is a shared, login-node. Please do not compute on NX.
- NX server is attached to Genius. Jobs submitted from NX will appear in Genius queue

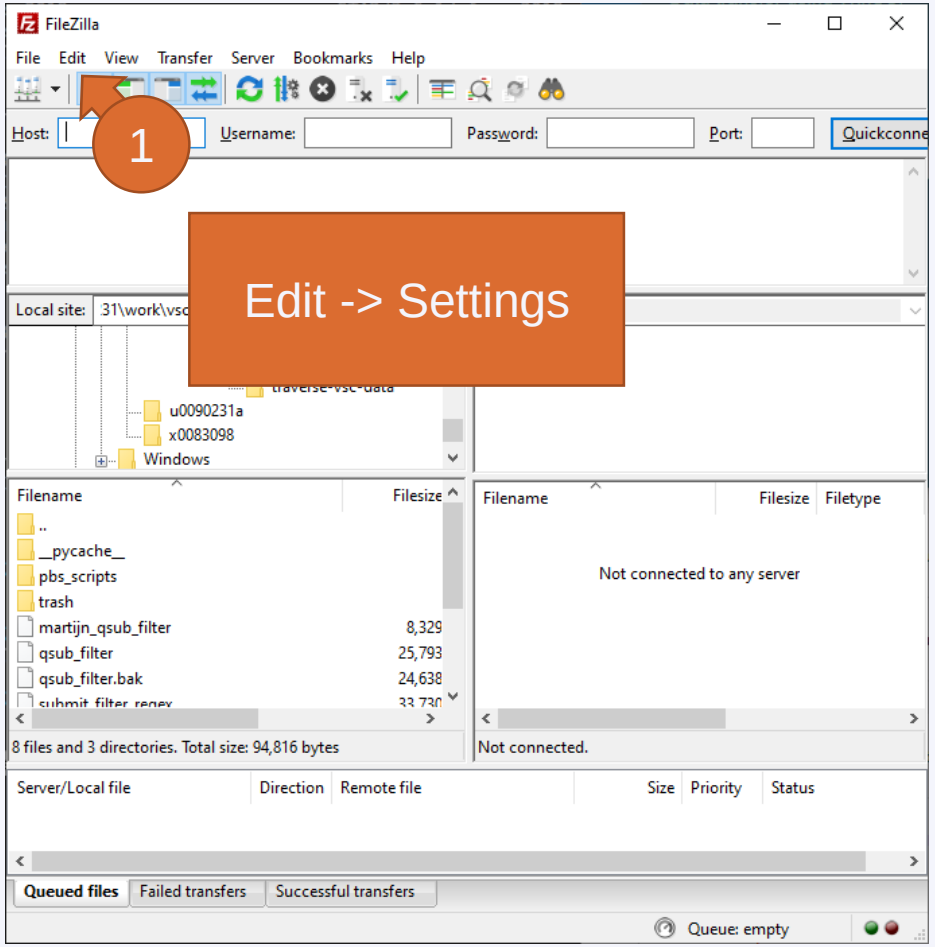
File Transfer with FileZilla

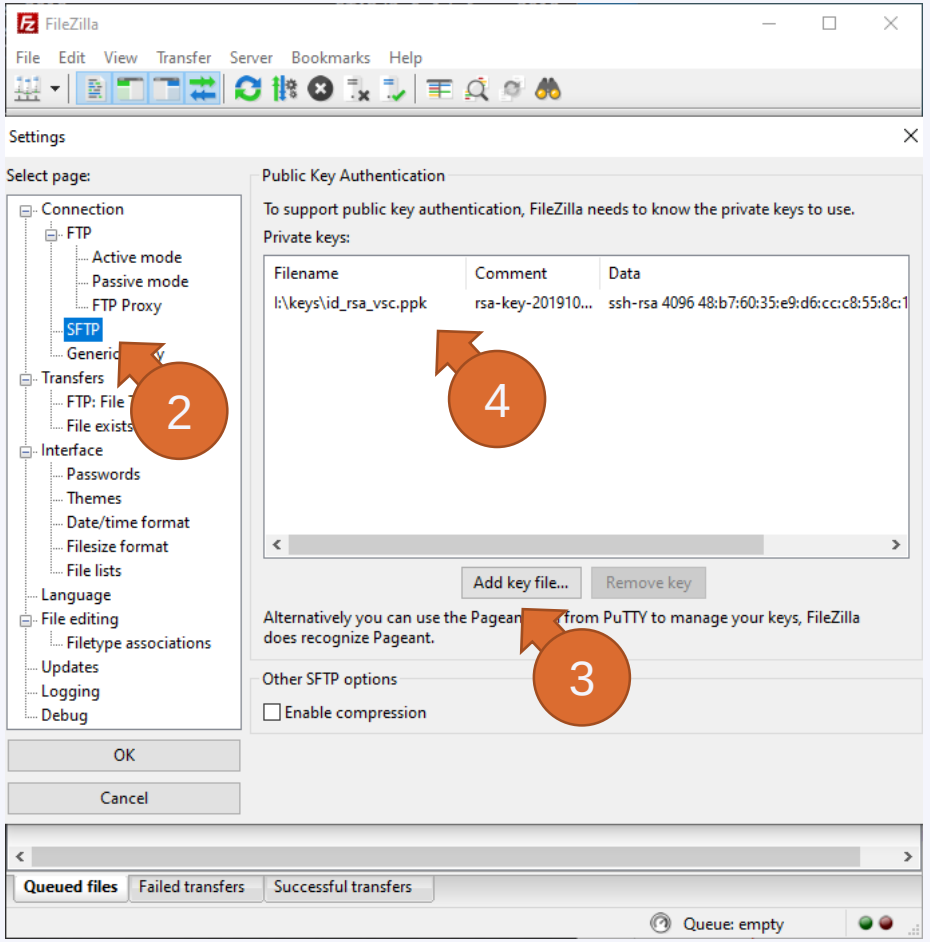
File transfer



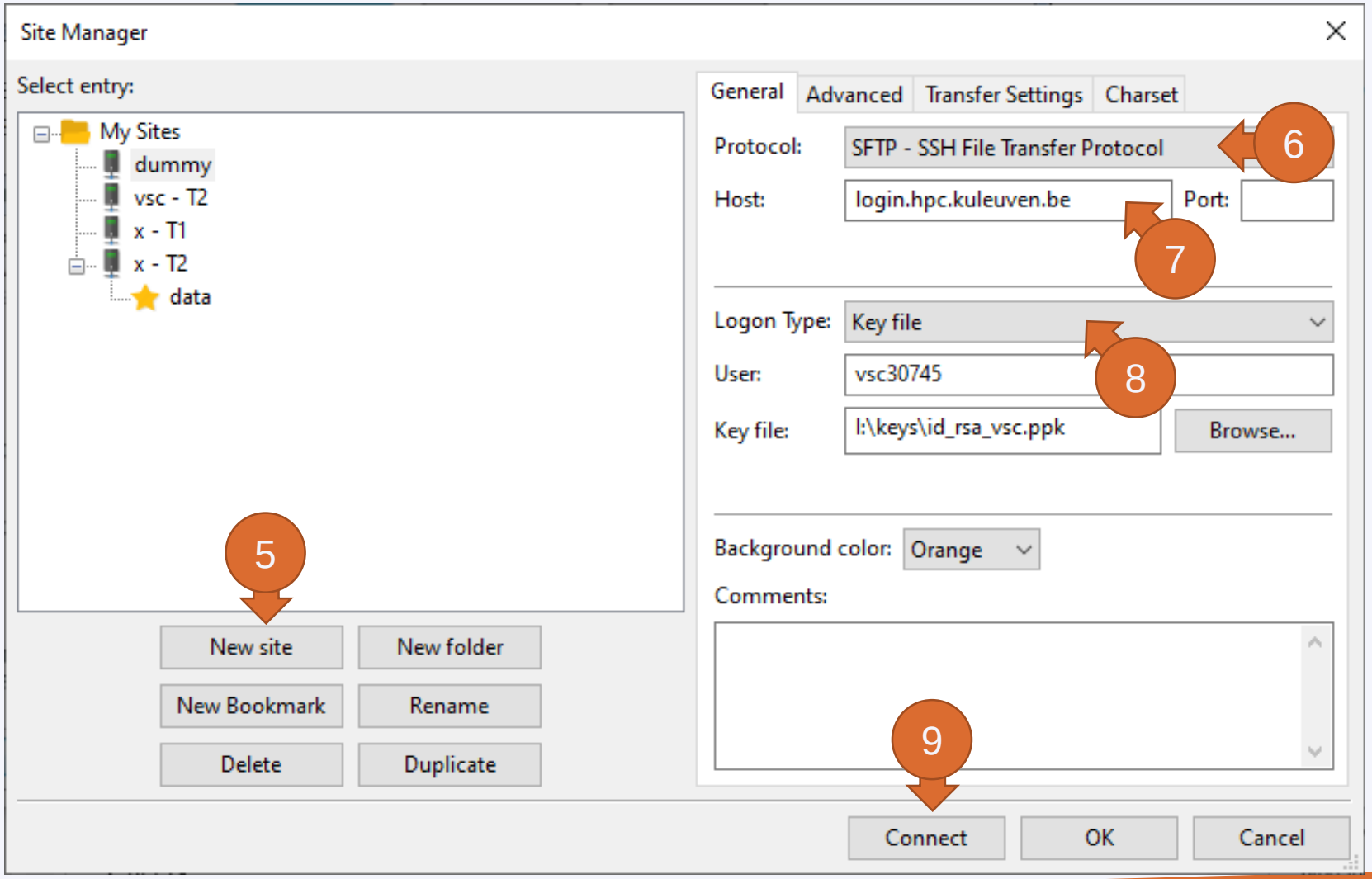
Application	OS
FileZilla	Windows, Linux, Mac
WinSCP	Windows
rsync, scp	Linux, Mac
Globus	Window, Linux, Mac

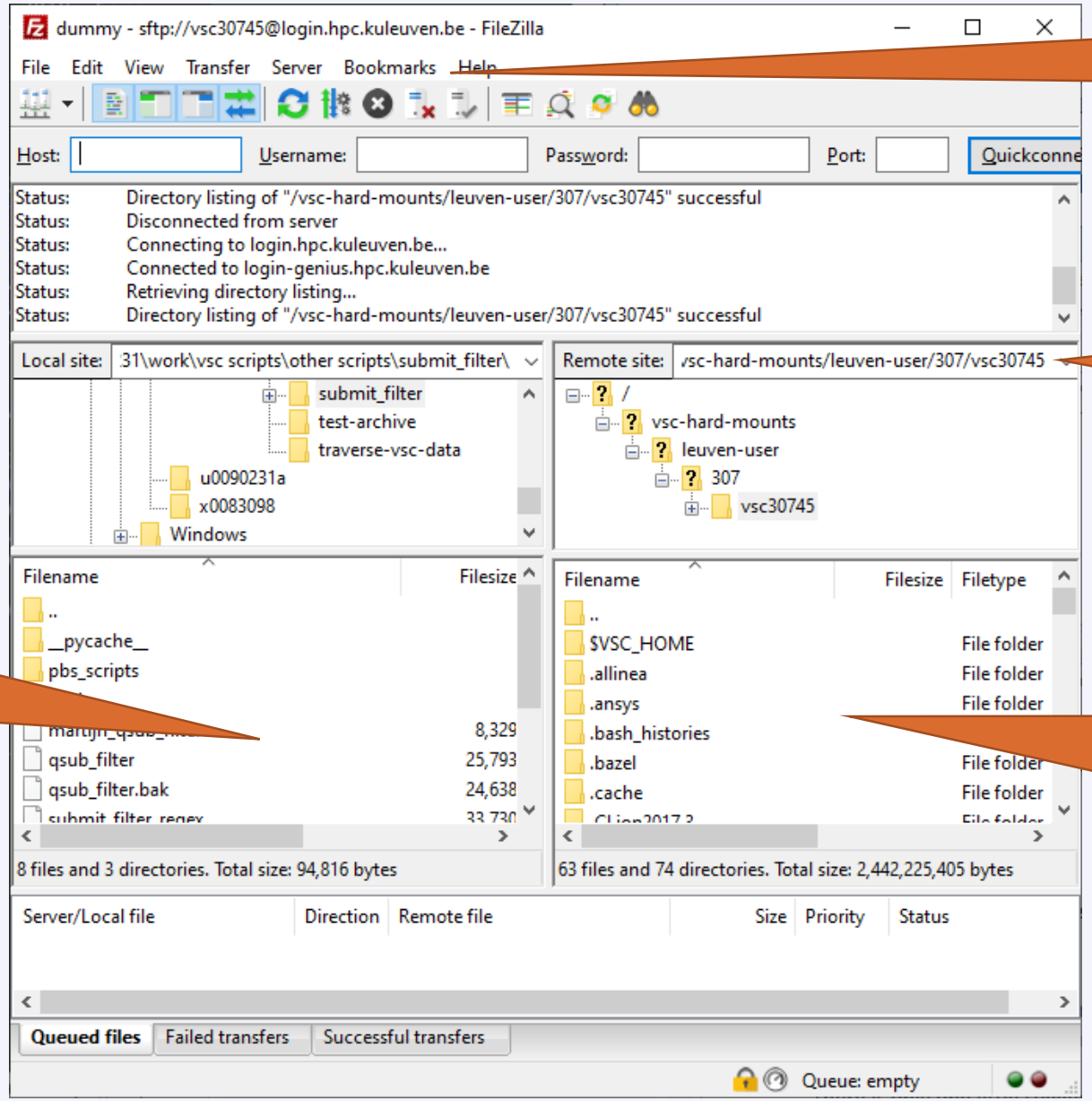
File transfer





File transfer





For convenience, you'd better bookmark your data and scratch folders!

Instead, go to your `$VSC_DATA` folder, e.g. /data/leuven/399/vsc39934

Your local machine (e.g. laptop)

Your VSC storage (e.g. `$VSC_DATA`, `$VSC_SCRATCH`). **Note:** by default you arrive at your own `$VSC_HOME`. Copy nothing there!

Software Stack

Software: Available Modules

- OS: Linux CentOS 7.6, Kernel 3.10.0-1062.1.2.el7.x86_64
- Compilers:
  Intel 2018a (icc, icpc, ifort; Intel MPI; Intel MKL)
  FOSS 2018a (gcc, g++, gfortran; OpenMPI; ScaLAPACK, OpenBLAS, FFTW)
- Note: Never mix FOSS and Intel compilers (gives dependency conflict)

Command	Remark
`module av`	List all installed modules
`module av Python`	List all Python-related modules
`module load Python/3.6.4-intel-2018a`	Load a specific module
`module list`	List all loaded modules and their dependencies
`module unload Python/3.6.4-intel-2018a`	Unload a module (but dependencies still stay)
`module purge`	Remove all modules from your work session

Software: Your Specific Needs

- You can always install your desired software in your $VSC_DATA
  Use Intel or FOSS toolchains
- If you cannot, ask us for help
- Python/R packages for AI and ML must be installed by the users themselves
- Install miniconda in your $VSC_DATA

```
$> wget https://repo.continuum.io/miniconda/Miniconda3-latest-
Linux-x86_64.sh
$> bash Miniconda3-latest-Linux-x86_64.sh -p $VSC_DATA/miniconda3
```

- Create conda environment per every project

```
conda create –n EnvTF tensorflow-gpu=2.0.0 pytorch
```

- Use your own conda/Python in your jobs/scripts

```
conda activate EnvTF
```

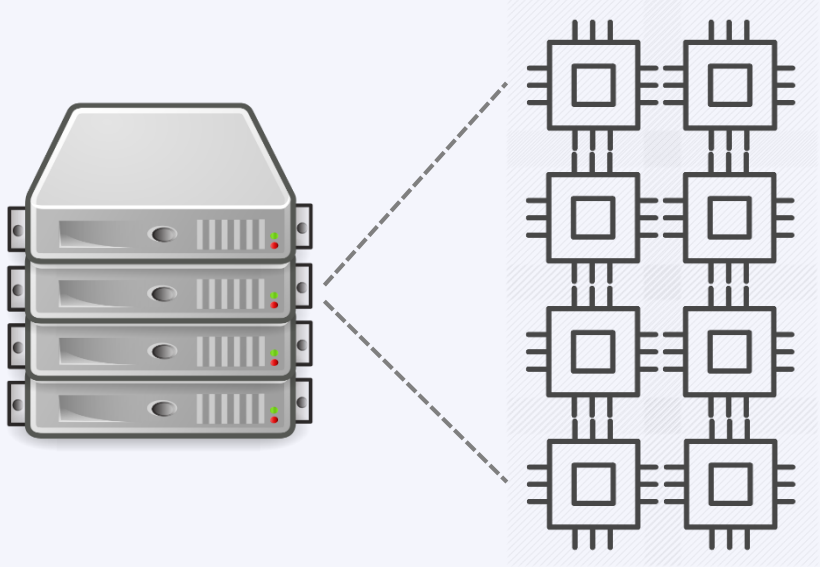
Starting to Compute

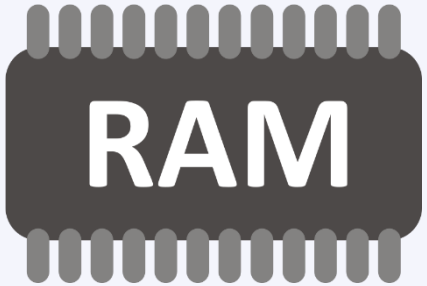
Extra services

- Request introduction credits
  https://admin.kuleuven.be/icts/onderzoek/hpc/request-introduction-credits
- Request project credits
  https://admin.kuleuven.be/icts/onderzoek/hpc/request-project-credits
  https://icts.kuleuven.be/sc/forms/Aanvraagformulier_HPC_Credits
- Extra project credits (to add to existing project)
  https://admin.kuleuven.be/icts/onderzoek/hpc/extra-project-credits
- Request extra storage
- https://admin.kuleuven.be/icts/onderzoek/hpc/hpc-storage

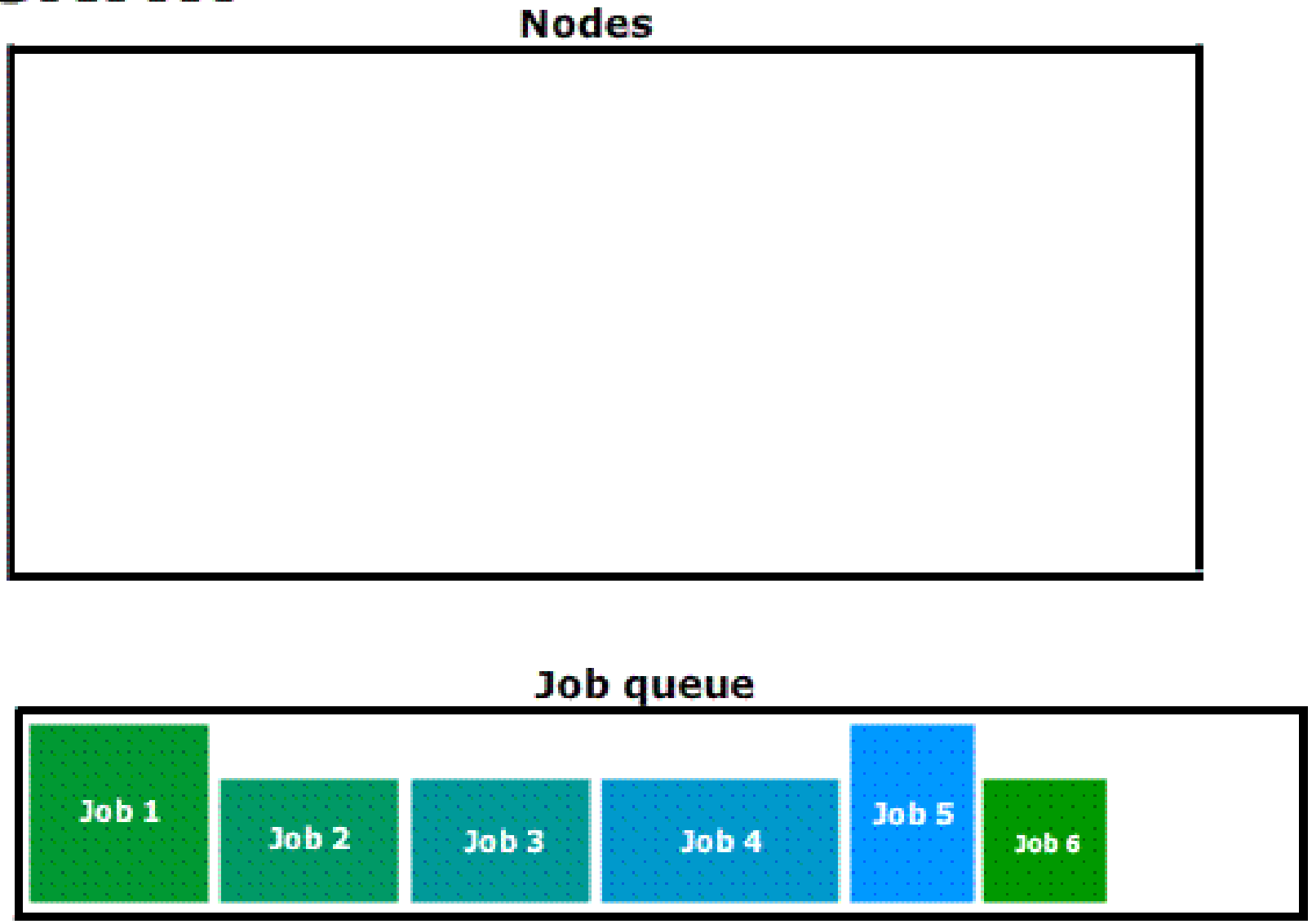
Resource Glossary

- **Nodes**: how many compute servers to request?
- **Cores**: how many cores per node to use?
- **Memory requirement**: how much memory each core needs?
- **Partition**: GPU, BigMem, Superdome, AMD
- **Walltime**: how long to use resources?





Backfill

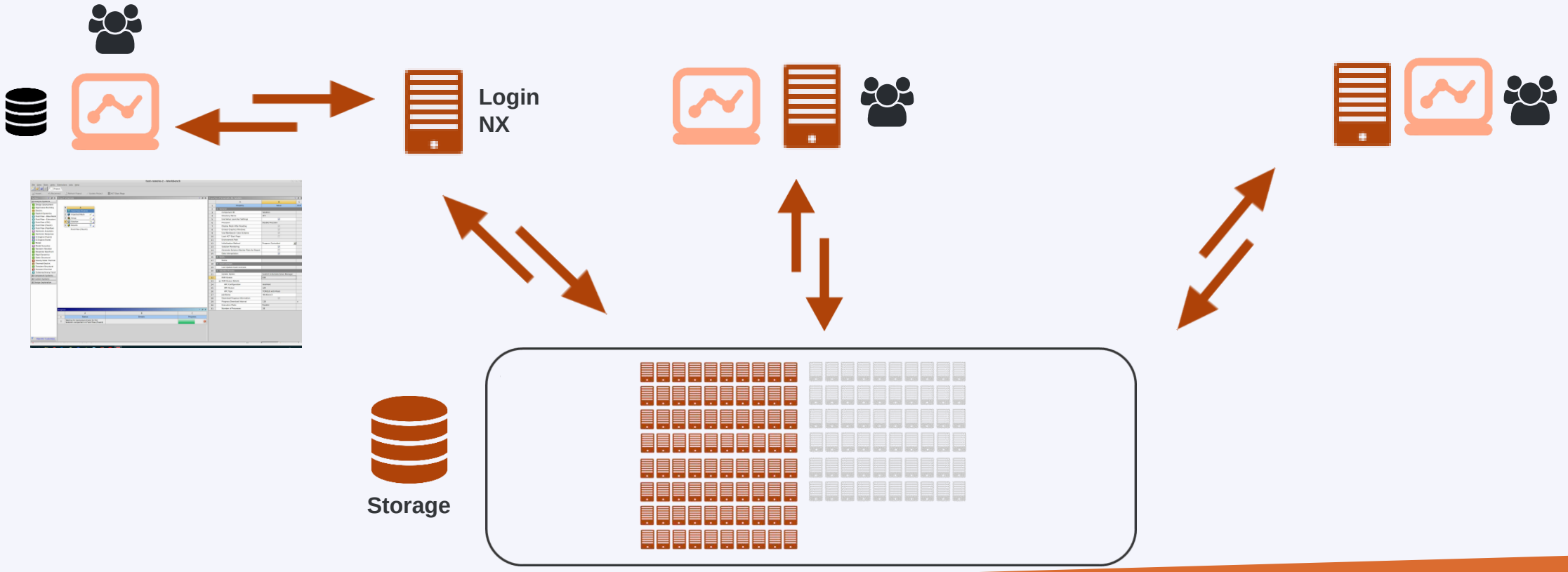


Ways to use the cluster

INTERACTIVE GUI WORKLOAD

BATCH WORKLOAD

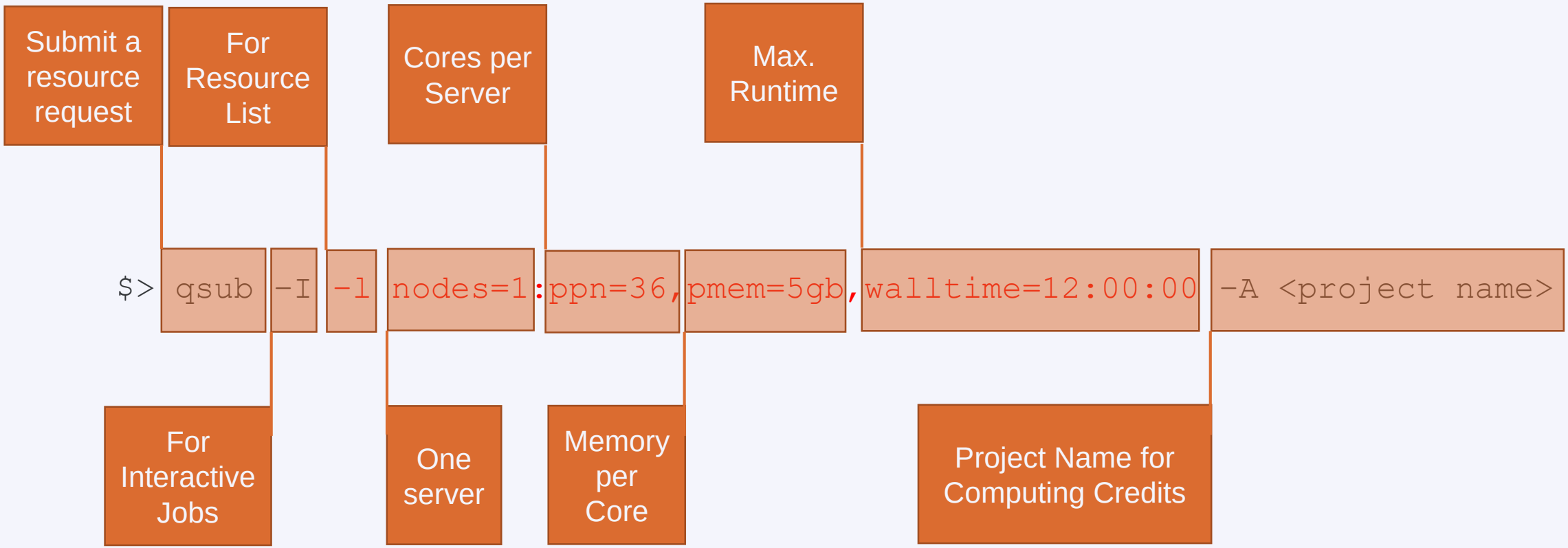
INTERACTIVE WORKLOAD



Types of Jobs

1) **Batch jobs** are by far the most common, and allow for the most efficient use of the infrastructure. Essentially, a batch job is a bash shell script that is executed on a compute node, and that can spawn a parallel computation on many nodes. These jobs are placed in the queue, and the user can forget about it until it is finished.
2) **Interactive jobs** are intended to work on one or more compute nodes interactively. This can be useful in the context of software development for debugging and profiling applications, or for interactive calculations or visualizations. Basically, one gets a shell on one of the compute nodes.
3) To get GPU(s), you can use **JupyterHub** with your internet browser

Requesting Compute Resources Interactively



Remarks

- Specifying <project name> for credits is mandatory, e.g.: **`-A lp_hpcinfo_training`**
- If you request 2000 introductory credits, you use them like: **`-A default_project`**
- Implicit defaults are `nodes=1:ppn=1,pmem=5gb,walltime=1:00:00`

Interactive Jobs

- Interactive job: 1 core for 1 hour (default)

```
$> qsub -I -A lp_hpc
```

- Interactive job with X-forwarding

```
$> qsub -I -X -A lp_hpc
```

- Ask 2 SkyLake nodes, 36 cores each, 4 GB RAM per core for 12 hours

```
$> qsub -I -l nodes=5:ppn=36:skylake,pmem=4gb,walltime=12:00:00 -A lp_hpc
```

- Ask 2 CascadeLake nodes, 36 cores each, 4GB RAM per core for 30 min

```
$> qsub -I -l nodes=2:ppn=36:cascadelake,pmem=4gb,walltime=30:00 -A lp_hpc
```

- Request fraction of a node

```
$> qsub -I -l nodes=1:ppn=8:cascadelake -A lp_hpc
```

Batch Workload

- Create a new text (ASCII) file
- Give it a good name!
  E.g. `simulation.pbs`
- The first line is the shebang!
  `#!/bin/bash`
- Put all commands line by line, e.g.
  `echo "Hello World!"`
- Submit the job to the batch server
- Receive a unique JobID
- Error and output files

Job Script

```
#!/bin/bash
echo "Hello World!"
```

Command Line

```
$ qsub simulation.pbs -A default_project
```

JobID

```
50041238.tier2-p-
moab2.tier2.hpc.kuleuven.be
```

stderr, stdout

```
$ ls simulation*
simulation.pbs
simulation.pbs.e50041238
simulation.pbs.o50041238
```

Standard Error File

- Always created
- <JobScript>.**e**<JobID>
- Contains all errors and warnings
- If empty: everything went well
- Always study it
- Addess all warnings and errors (if you can)
- Typical error examples …

stderr

```
$ ls *.e*
simulation.pbs.e50041238
```

Job Crash

```
forrtl: error (78): process killed (SIGTERM)
Stack trace terminated abnormally.
```

Short Walltime

```
PBS: job killed: walltime 432031 exceeded
limit 432000
```

Low Disk Space

```
IOError: [Errno 122] Disk quota exceeded
```

Standard Output File

- Always created
- <JobScript>.o<JobID>
- Contains all standard output (instead of screen)
- Always study it

stdout

```
$ ls *.o*
simulation.pbs.o50041238
```

Output File

```
time: 3600
nodes: 1
procs: 1
account string: lpt2_sysadmin
queue: q1h
============================
Hello World!
============================
Date: Fri Mar 20 14:32:01 CET 2020
Allocated nodes:
r26i27n11
Job ID: 50240161.tier2-p-moab-2.tier2.hpc. …
Resource List:
pmem=5gb,walltime=01:00:00,nednodes=1:ppn=36
Resources Used:
cput=00:00:01,vmem=0kb,walltime=00:00:04,mem=0kb,e
nergy_used=0
Queue Name: q1h
----------------------------
time: 4
nodes: 1
procs: 1
account: lpt2_sysadmin
```

Summary

stdout

Resources

summary

Example: PBS Job Script

```
#!/bin/bash
#PBS -l walltime=00:10:00
#PBS -l nodes=1:ppn=36
#PBS -l pmem=5gb
#PBS -N testjob
#PBS -A lp_hpcinfo_training
#PBS -m abe
#PBS -M my.name@kuleuven.be

module load intel/2018a
which icc

cd $PBS_O_WORKDIR
cp /apps/leuven/training/HPC-intro/cpujob.pbs $VSC_SCRATCH
touch output.log
echo I am done
```

Shebang

Resource List

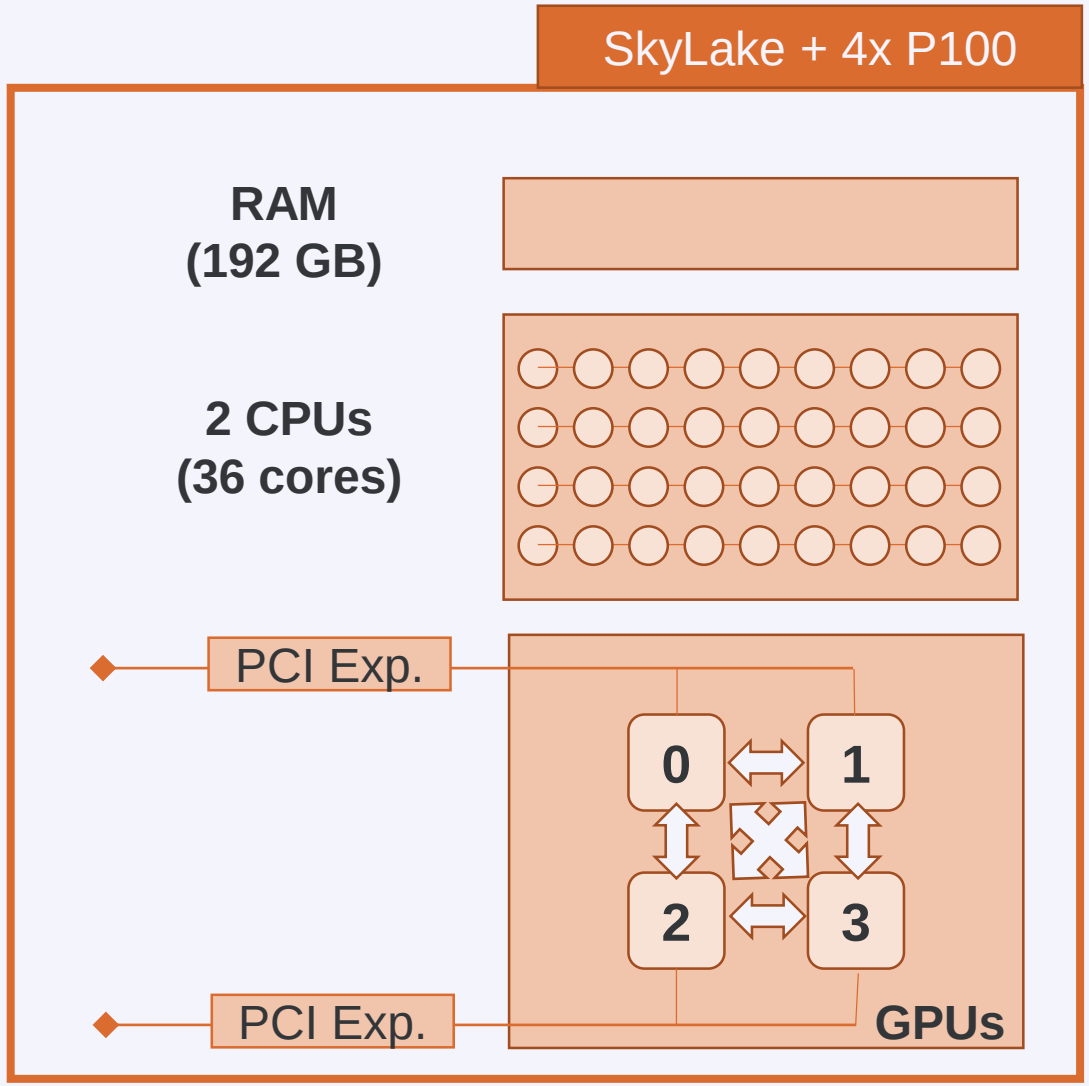
Module load(s)

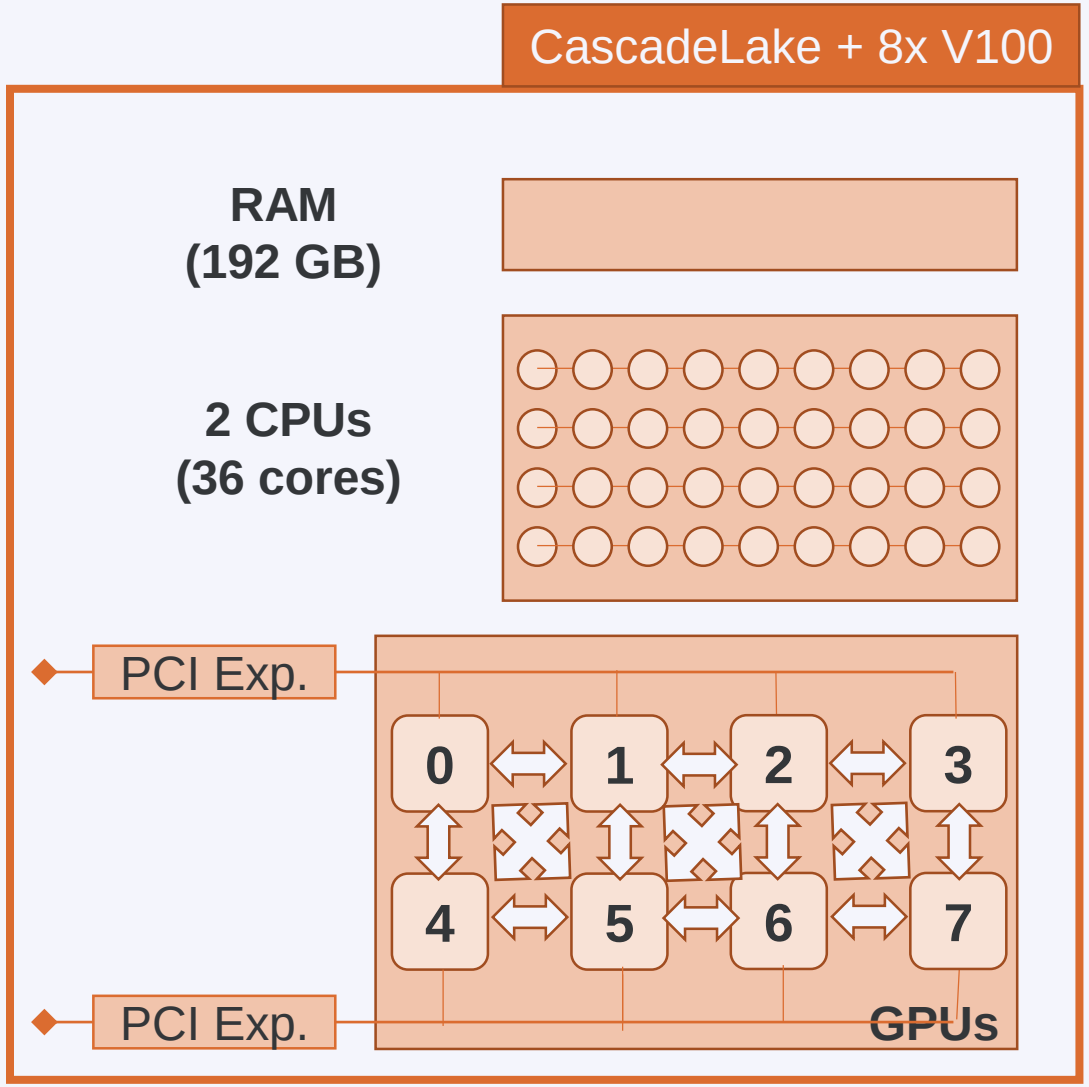
Move data

Execure commands

GPU Partition

GPU Nodes


SkyLake + 4x P100
RAM
(192 GB)
2 CPUs
(36 cores)
PCI Exp.
PCI Exp.
0
1
2
3
GPUs

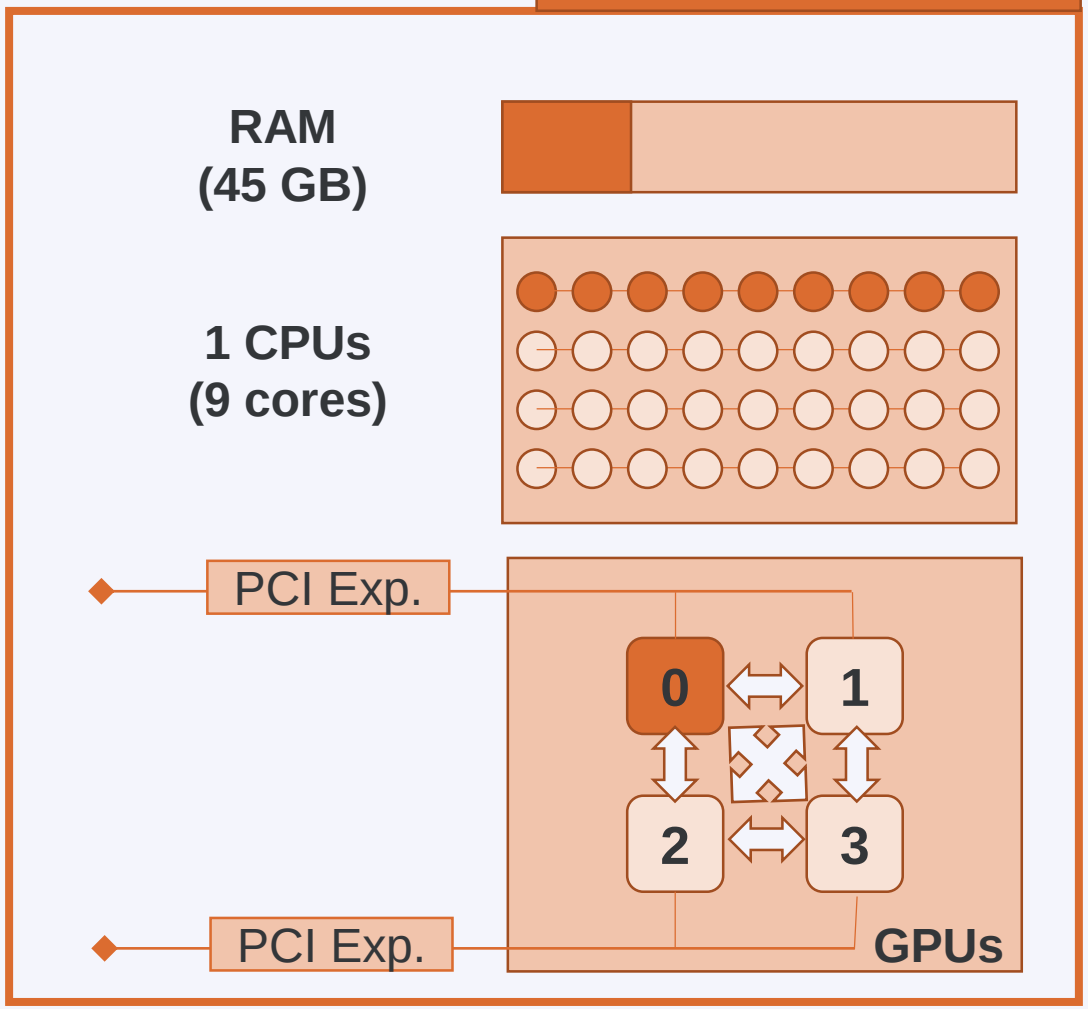

CascadeLake + 8x V100
RAM
(192 GB)
2 CPUs
(36 cores)
PCI Exp.
PCI Exp.
0
1
2
3
4
5
6
7
GPUs

Using P100 GPU(s)

Using 1 GPU

```
#!/bin/bash -l
#PBS -l walltime=00:30:00
#PBS -l partition=gpu
#PBS -l nodes=1:ppn=9:gpus=1:skylake
#PBS -l pmem=5gb
#PBS -N testjob
#PBS -A default_project
```

You get …

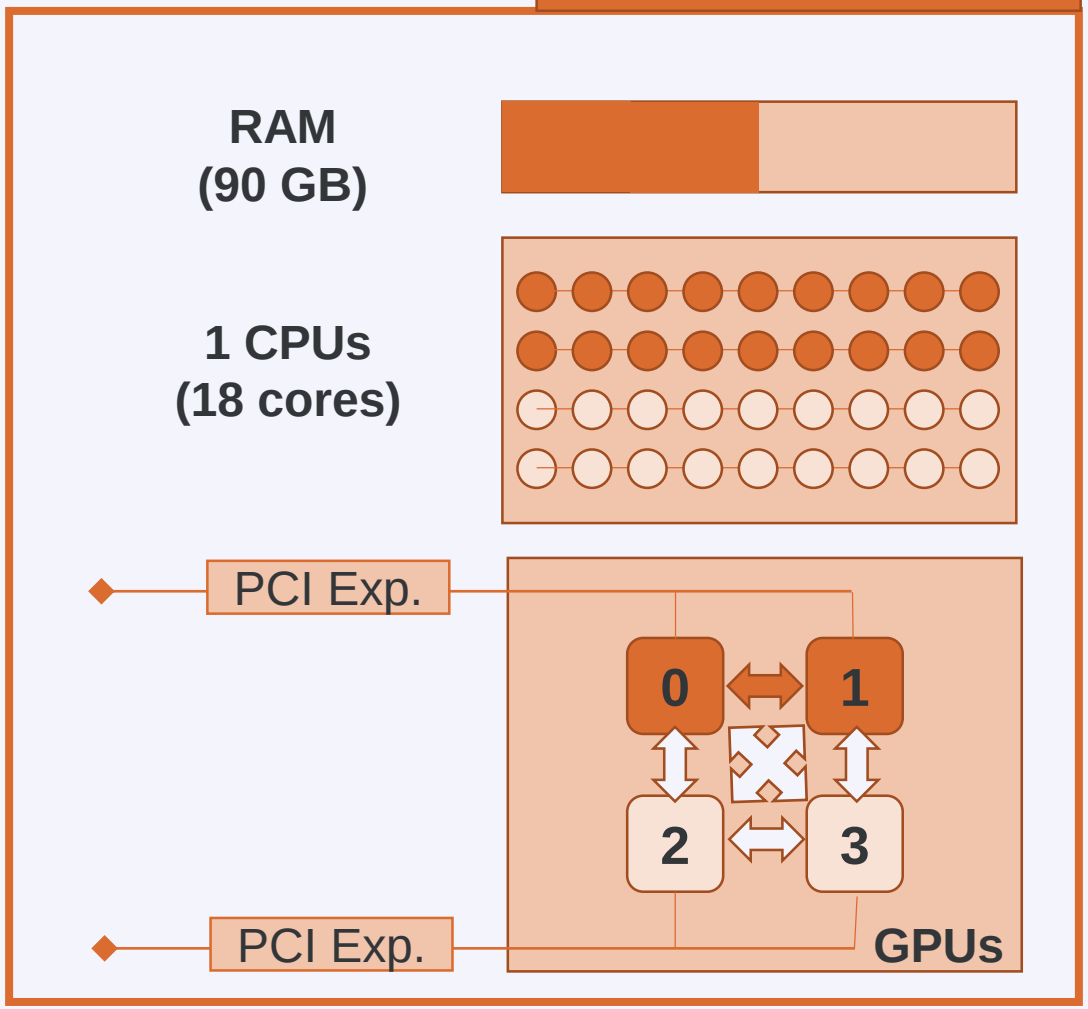


Using P100 GPU(s)

Using 2 GPUs

```
#!/bin/bash -l
#PBS -l walltime=00:30:00
#PBS -l partition=gpu
#PBS -l nodes=1:ppn=18:gpus=2:skylake
#PBS -l pmem=5gb
#PBS -N testjob
#PBS -A default_project
```

You get …

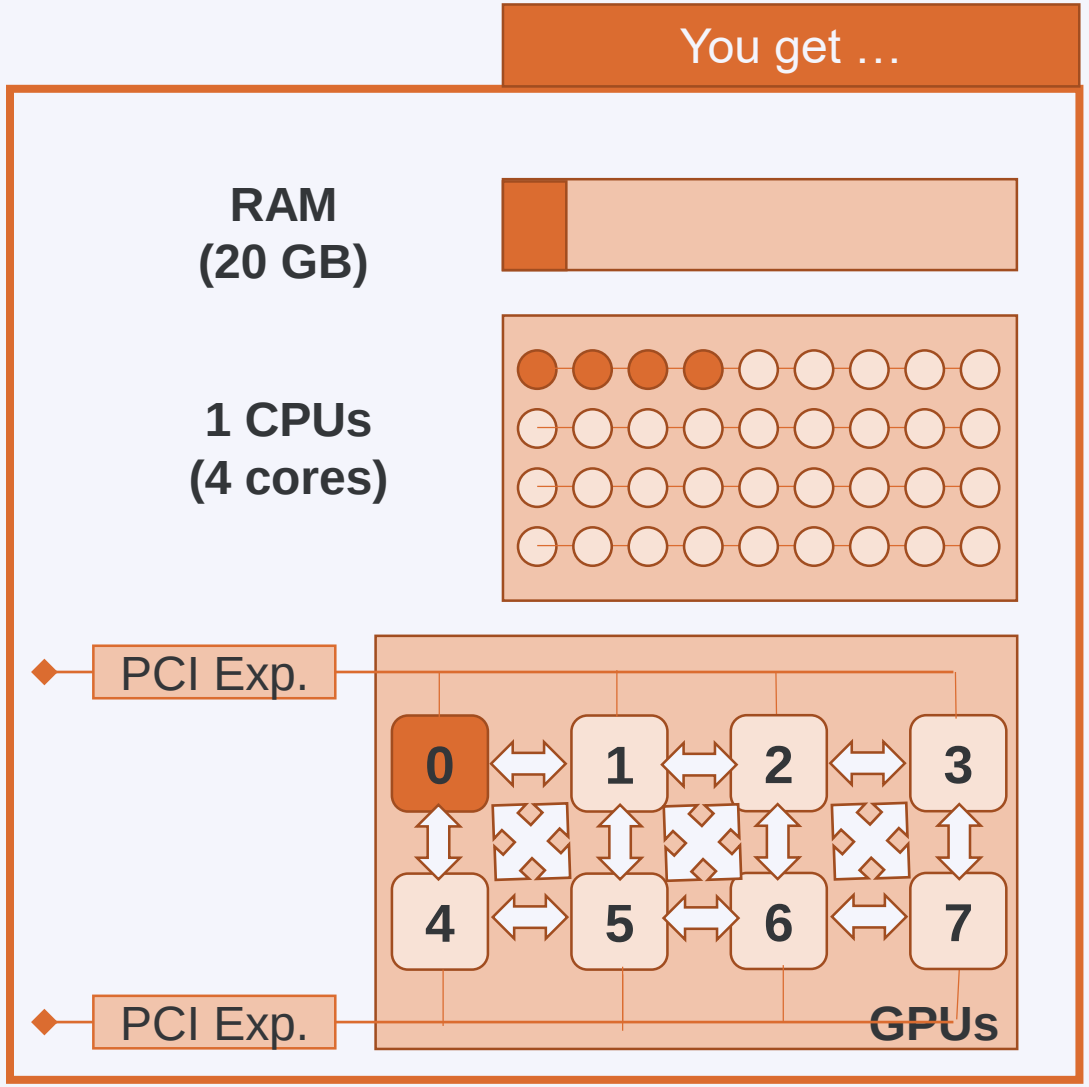


Using V100 GPU(s)

Using 1 GPU

```
#!/bin/bash -l
#PBS -l walltime=00:30:00
#PBS -l partition=gpu
#PBS -l
nodes=1:ppn=4:gpus=1:cascadelake
#PBS -l pmem=5gb
#PBS -N testjob
#PBS -A default_project
```

You get …

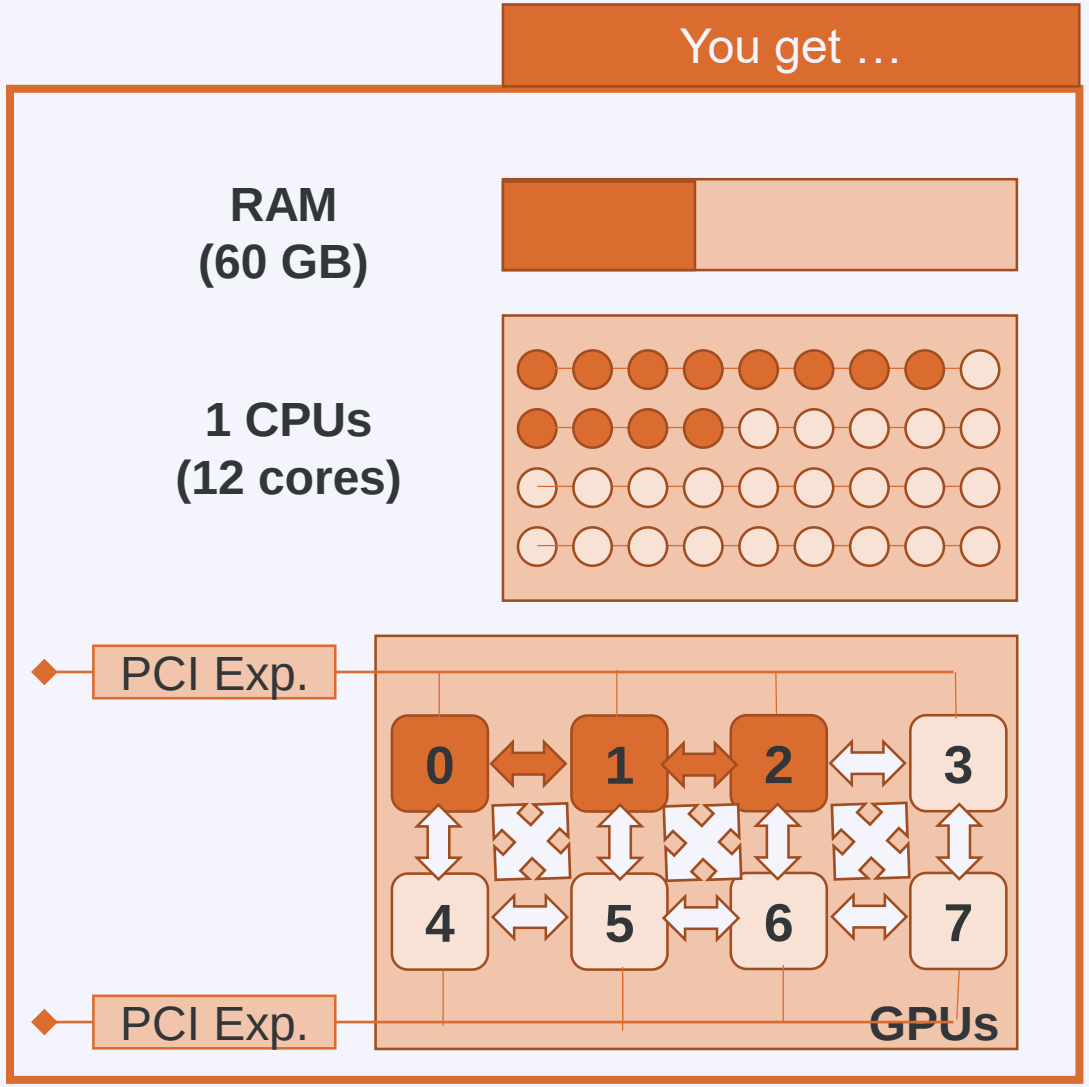


Using V100 GPU(s)

Using 3 GPU

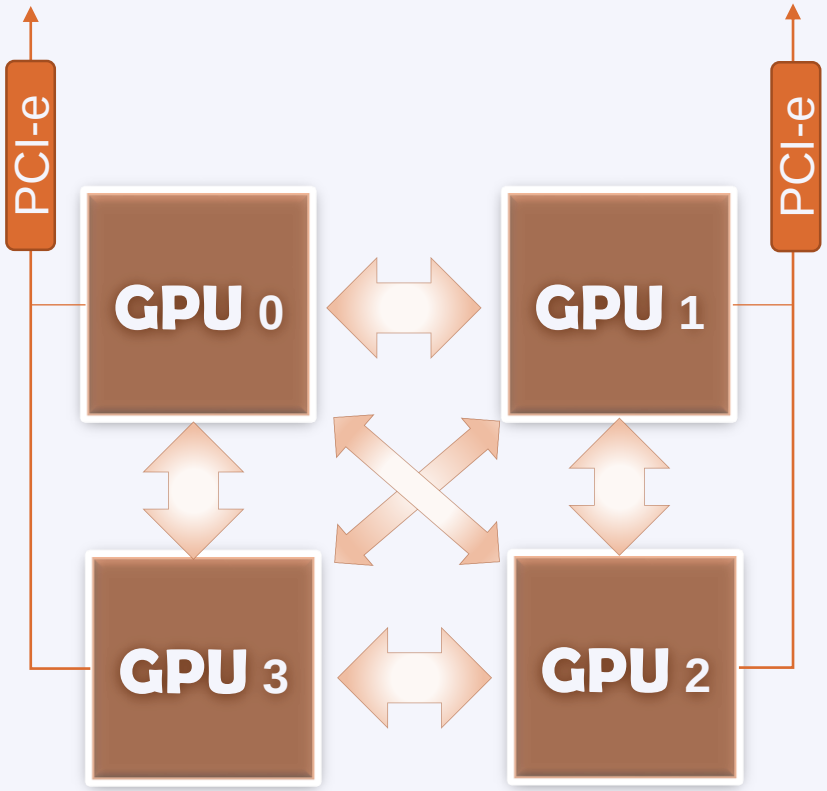
```
#!/bin/bash -l
#PBS -l walltime=00:30:00
#PBS -l partition=gpu
#PBS -l
nodes=1:ppn=12:gpus=3:cascadelake
#PBS -l pmem=5gb
#PBS -N testjob
#PBS -A default_project
```

You get …



Peer-to-Peer Bandwidth (GB per seconds)

High | Med | Low

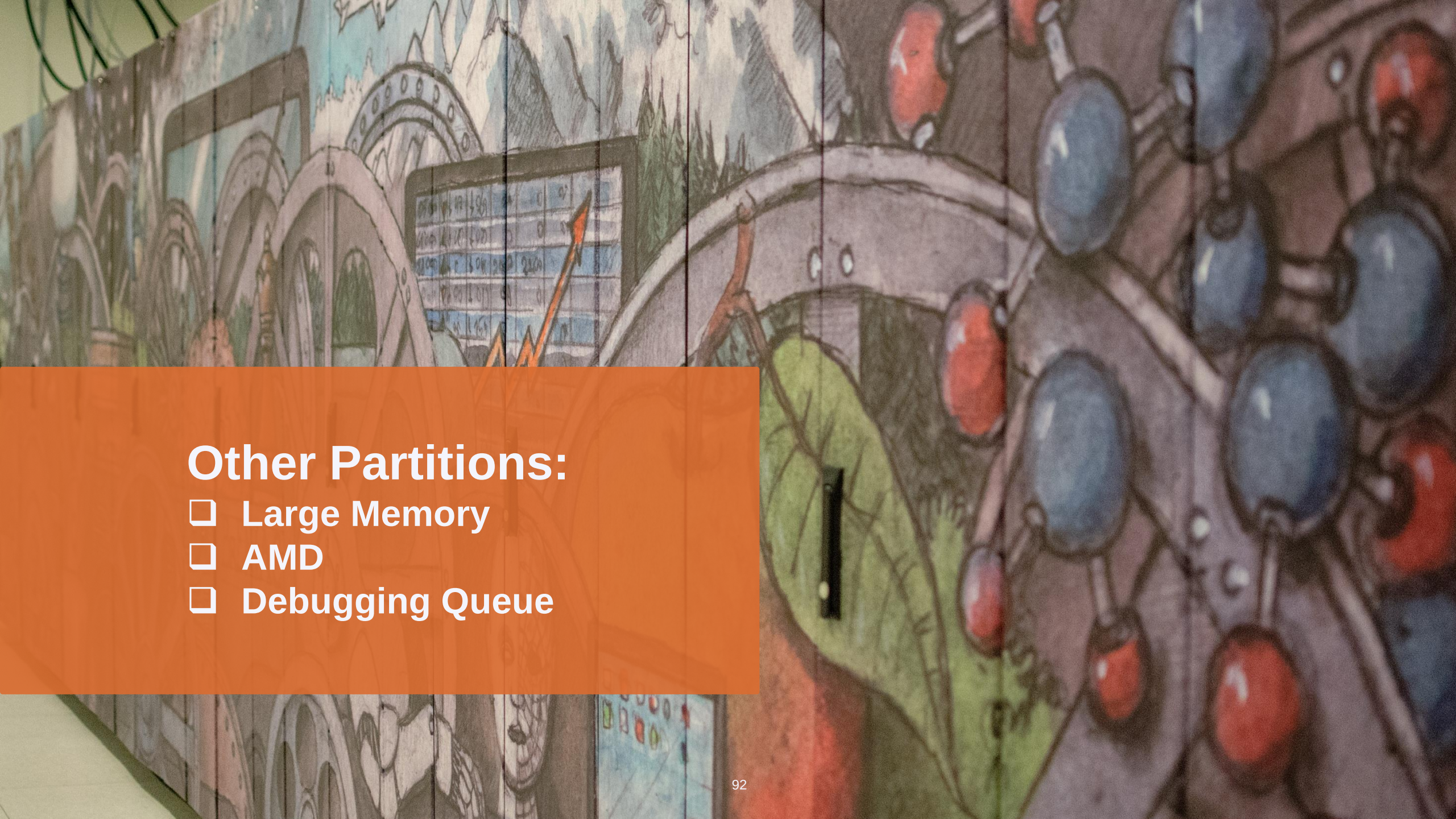


Dev. ID	0	1	2	3
0	509	10	19	18
1	10	508	18	18
2	19	18	508	10
3	18	18	10	507

Bi-directional P2P: **PCIe**

Dev. ID	0	1	2	3
0	508	37	37	61
1	37	507	61	37
2	36	61	508	37
3	62	37	37	506

Bi-directional P2P: **NVLink**
(Nvidia P100)



Other Partitions:

- ❑ Large Memory
- ❑ AMD
- ❑ Debugging Queue

Requesting Large Memory Nodes

- All Thin Nodes have 192 GB memory each
  You need to leave 6 GB aside for the Operating System, and use the rest
- By default, every core can have
  `pmem = (192GB – 6GB) / (36 cores) ≈ 5GB per-core`
- There are 3 methods you can request larger memory than normal:
  - Use less cores (`ppn`), but larger memory-per-core (`pmem`)
  - Use `bigmem` partition
  - Use `Superdome` machine

First Method

- You sacrifice `ppn`, in favor of `pmem`
- E.g. if you need 20GB memory per core:
  `$> qsub -I -l nodes=1:ppn=8,pmem=20gb …`

- Note: the total memory requested must be less than 192GB – 6GB = 186GB, i.e.
  `ppn × pmem ≤ 186GB`

Requesting Large Memory Nodes

Second Method

- We have 10 Large Memory nodes, with 768 GB memory each
- Specify `-l partition=bigmem` with `qsub` command
- Each core can have
  `pmem=(768 GB - 6 GB) / (36 cores)  ≈ 21 GB per-core`

- E.g.
  ```
  $> qsub -I -l partition=bigmem,nodes=1:ppn=36,pmem=21gb …
  ```

- You can also compromise `ppn` for `pmem`
  E.g.
  ```
  $> qsub -I -l partition=bigmem,nodes=1:ppn=10,pmem=76gb …
  ```

- Make sure `ppn × pmem  ≤ 760GB`

Requesting Large Memory Nodes

Third Method

- Superdome: 8 nodes, 14 cores-per-node, **6 TB** total RAM
- first do `module load superdome`
- Default `pmem=50gb`
- `numanode` is total CPUs to use (max. 8)
- `lprocs` is total number of cores to use
- E.g.

```
$> module load superdome
$> qsub -l partition=superdome -q qsuperdome \
          -L tasks=1:lprocs=14:place=numanode=1
```

Request **1/8th** of the machine, with 50 GB per core

- E.g.

```
$> qsub -l partition=superdome -q qsuperdome \
          -L tasks=1:lprocs=42:place=numanode=3
```

Request **3/8th** of the machine, with 50 GB per core

```
$> qsub -l partition=superdome -q qsuperdome \
          -L tasks=1:lprocs=7:place=numanode=1:memory=700gb
```

If you want to use > 50GB/core
Eg 7 cores with 100GB/core

(Experimental) AMD Nodes

- 4 nodes
- 2x EPYC 7501 CPUs (2.0 GHz, 32 cores)
- 8 memory channels / CPU
- Max Mem BW 158.95 GiB/s
- RAM: 256 GB
- Remark: `pmem=3800mb`
  If not specified, the job will not start

Job Script

```
#!/bin/bash -l
#PBS -l walltime=01:30:00
#PBS -l partition=amd
#PBS -l nodes=1:ppn=64
#PBS -l pmem=3800mb
#PBS -N testjob
#PBS -A lp_my_project
```

Debugging / Testing Jobs

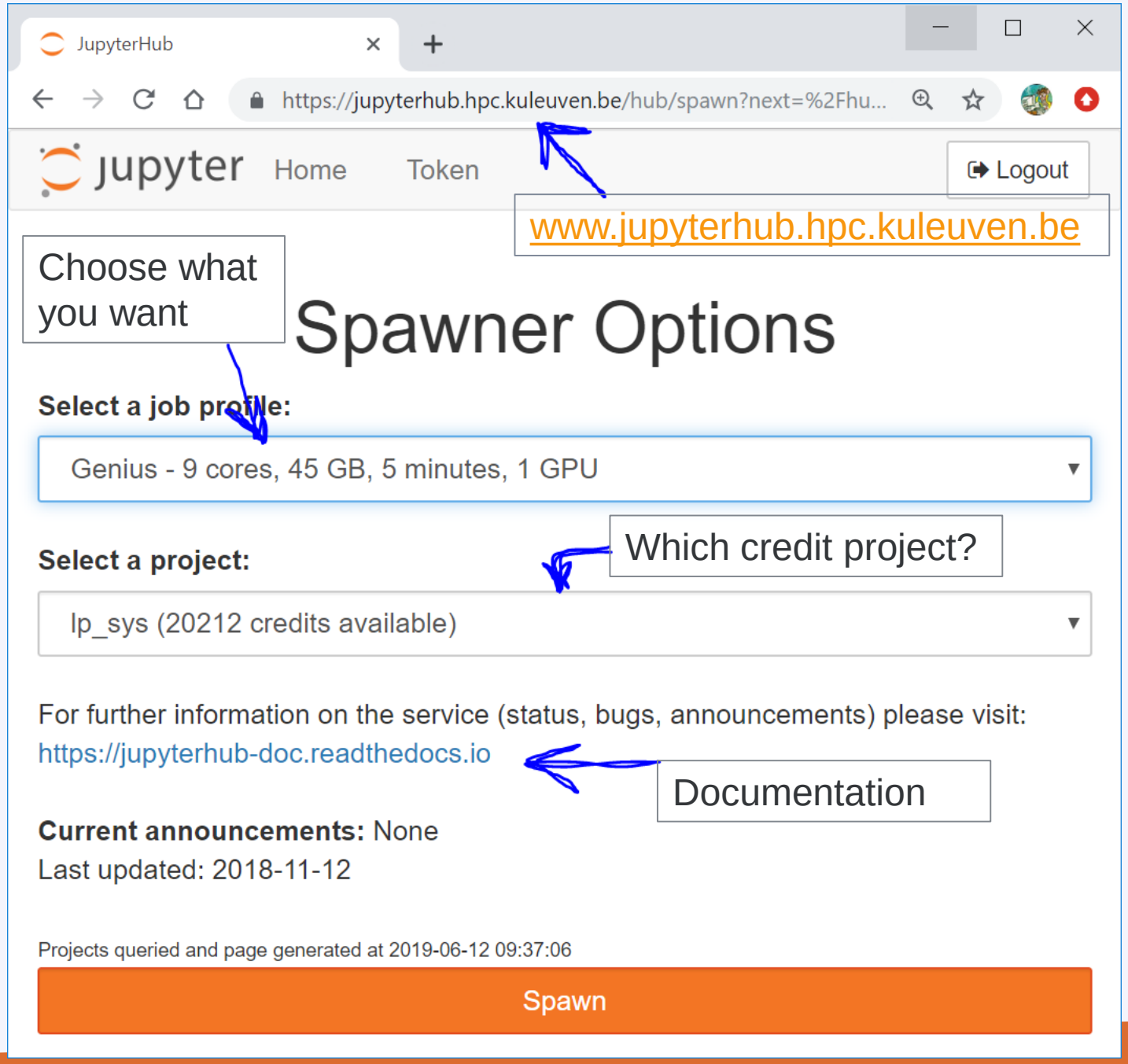
- To quickly test/debug your (parallel) application
- 2 dedicated nodes on ThinKing and Genius
  One GPU node and one CPU node
- Such jobs do **not** go to the normal queue, so they start faster
- Max. walltime is **30 minutes**
- You must specify Quality of Service (**qos**)

Request Debugging Nodes

```
$> qsub -l nodes=1:ppn=10 -l qos=debugging -l \
      walltime=30:00 -A default_project
```

Managing & Monitoring Jobs

Command	Purpose
`$> qsub ….`	Submit a job (batch/interactive)
`$> qdel <JobID>`	Delete a specific job
`$> checkjob -vvv <JobID>`	Very detailed job info (very useful to diagnose issues)
`$> qstat -n`	Status of all recent jobs
`$> qstat -Q -f`	Info about available queues
`$> showstart <JobID>`	Give a *rough* estimate of start time
`$> showq` `$> showq -p gpu`	Show minimal info about a queue or partition (`-p`)
`$> pbstop`	Overview of the cluster
`$> module load accounting` `$> mam-balance`	Overview of different credit projects that you can use (`qsub -A <Project>`)
`$> mam-list-allocations`	Detailed overview of your credit projects


JupyterHub
https://jupyterhub.hpc.kuleuven.be/hub/spawn?next=%2Fhu...
jupyter
Home
Token
Logout
www.jupyterhub.hpc.kuleuven.be
Choose what you want
Spawner Options
Select a job profile:
Genius - 9 cores, 45 GB, 5 minutes, 1 GPU
Select a project:
Which credit project?
lp_sys (20212 credits available)
For further information on the service (status, bugs, announcements) please visit:
https://jupyterhub-doc.readthedocs.io
Documentation
Current announcements: None
Last updated: 2018-11-12
Projects queried and page generated at 2019-06-12 09:37:06
Spawn

Jupyter Notebook on NX

- Install Miniconda in your $VSC_DATA

```
$> wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
$> bash Miniconda3-latest-Linux-x86_64.sh -p $VSC_DATA/miniconda3
```

- Create a conda env including Jupyter

```
$> conda create –n JupNtbk jupyter scikit-learn
```

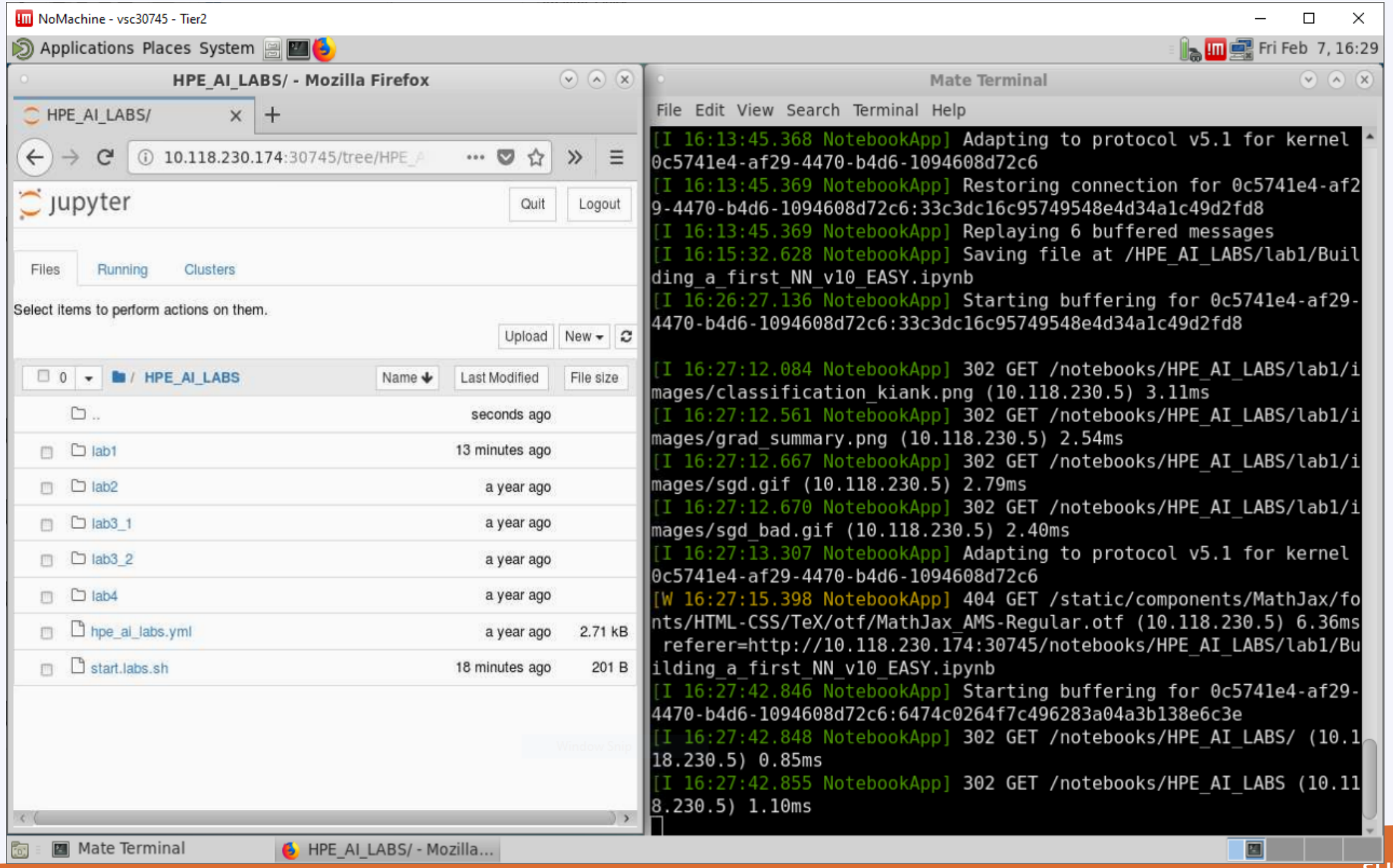
- Activate this env

```
$> conda activate JupNtbk
```

- Start an interactive GPU job (`qsub –I …`)
- In another terminal, launch the notebook to get URL

```
$> jupyter notebook --ip=`hostname -i` --port=${USER:3} --no-browser
--allow-root
```

- Copy-paste the URL into NX Firefox; done



Singularity Containers

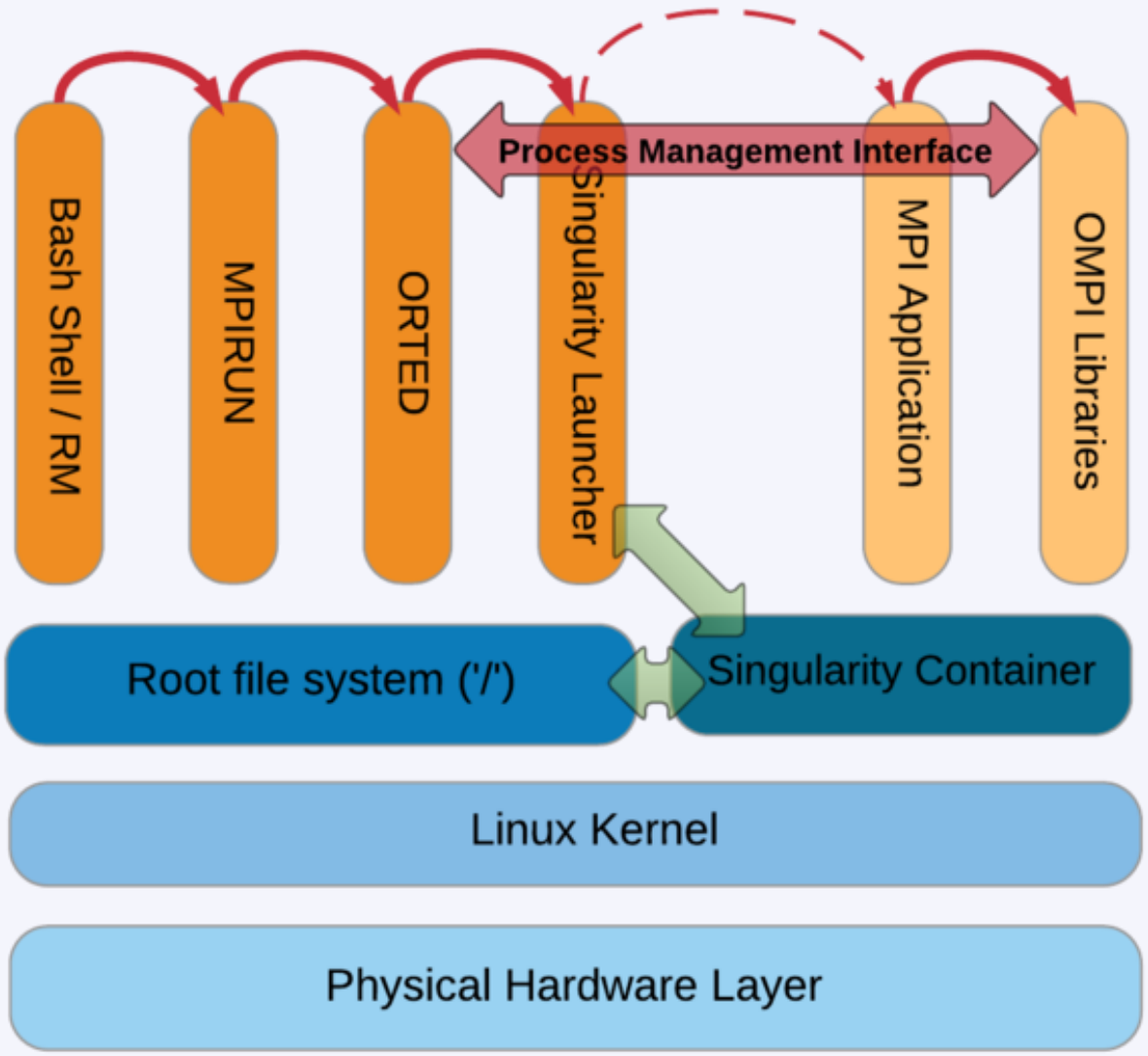
- What?
  + Self-contained OS & software & data
- Why?
  + fully resolved dependency chains
  + portable workflow
- How?
  + You create the image
  + Run it on Genius
  + MPI/OpenMP is supported

PBS Script

```
#!/bin/bash -l
#PBS -l walltime=00:30:00
#PBS -l nodes=4:ppn=36:pmem=5gb
#PBS -A lp_hpcinfo_training

cd $PBS_O_WORKDIR

singularity run Project.simg ./model.exe
```



Where to run your jobs

Which system should I use?

Knowing yourself is
the beginning
of all wisdom

What kind of work I do?

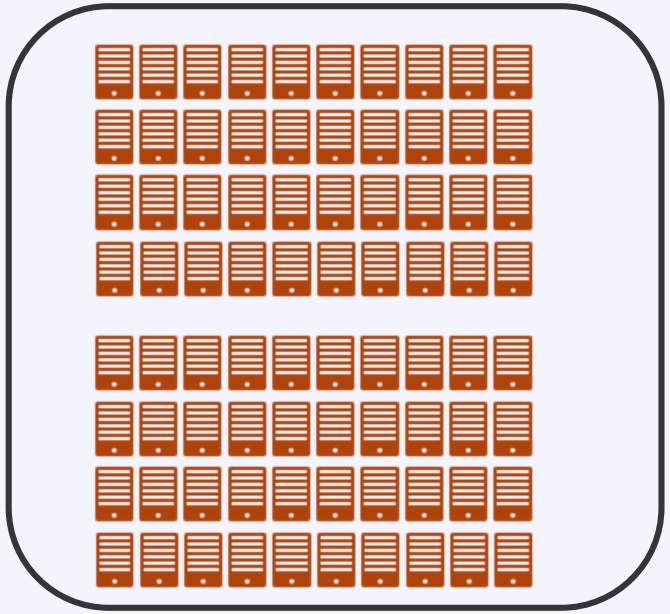
- CPU usage
  - Do you run single core jobs or parallel?
    - Single core
      - Is your code optimized to AVX ?
    - Parallel:
      - single node or multinode?
        - Single node: does my application scale? Up to how much cores ?
      - Multinode:
        - Does my application scale?
- Memory usage
  - How much memory (per core) my jobs use?
  - Is your processing memory bound?

What kind of work I do?

- How to find out
  - Know your algorithm
  - Benchmark your typical workload on the different architectures
  - Use tools to track behavior of your code
    - Use Monitor tool
    - Use profiling tools eg. ARM MAP or Scalasca
  - Consult training agenda
    - Upcoming Allinea ARM Forge training

Which system for what?

Compute nodes (thin nodes)



Thinking:

- Haswells 24 cores
  64/128 GB

Single core jobs
Worker jobs
Single node parallel jobs

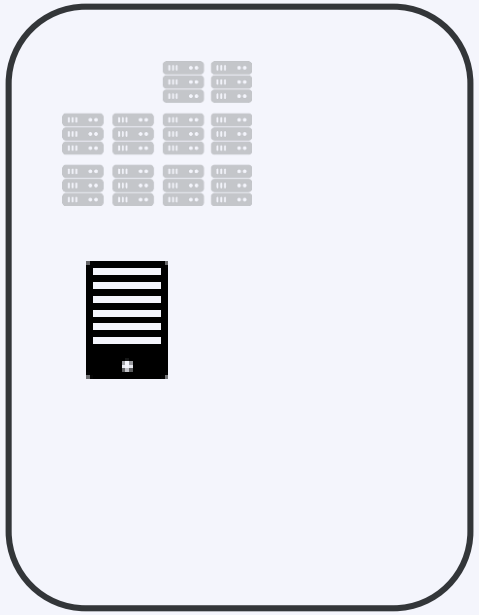
Genius:

- Skylake: 36 cores
- Cascade Lake: 36 cores
- 192 GB
- IB EDR

Multinode parallel jobs
AVX 512
Higher core/memory ratio
Higher memory bandwidth

Which system for what?

Compute nodes (Large memory)



Genius:

- Big memory 36 cores
- 768 GB

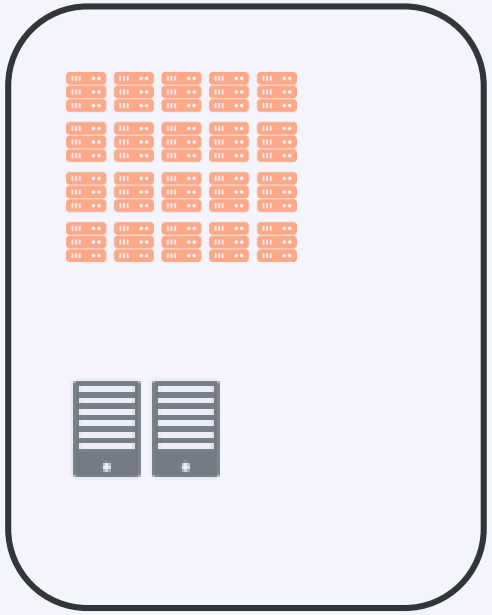
5GB<mem/core<20GB

- Superdome 112 cores
- 6TB

> 20GB mem/core
> 700 GB of total RAM
Single node scaling > 36 cores

Which system for what?

Compute nodes (GPU nodes)



Genius Skylake GPU: P100

Single GPU workload
Multi GPU workload
with limited scaling

Genius Cascadelake GPU: V100

> 4 GPU workload
Specific ML workloads
(convolutional networks)

Credits

Credits card concept:

- Pre-authorization: holding the balance as unavailable until the merchant clears the transaction
- Balance to be held as unavailable: based on requested resourced (walltime, nodes)
- Actual charge based on what was really used: used walltime (you pay only what you use, e.g. when job crashes)
- See output file

How to check available credits?

```
$ mam-balance
```

How to check the rates?
Check service catalogue

```
$ mam-list-chargerates
```

Credit Pricing

- You pay as you go!
- For academic projects:
  1000 credits = 3.5 EUR
- Credits needed for a job:

  #credits = Walltime (hr) × #nodes × Factor

- For shared nodes, you pay a fraction of the costs, based on the ppn specified

Cluster / Partition	Credits/h
Genius Cascadelake	11.3
Genius Cascadelake 8 GPUs	40
Genius Skylake 4 GPUs	20
Genius Skylake BigMem	12
Genius Skylake	10
Genius / Superdome	10
Genius / AMD nodes	10
ThinKing / Haswell	6.68
ThinKing / IvyBridge	4.76

How to Manage Credits?

Command	Purpose
`mam-balance`	List active projects and available credits
`mam-list-allocations`	List the validity dates of different projects/allocations
`mam-list-chargerates`	List current charge rates for different nodes
`mam-list-accounts -a <account-name>`	List the members in an account

Hands-on 2

- Copy /apps/leuven/training/HPC_bioinf to your $VSC_DATA directory and go to this directory
  ```
  cp -r /apps/leuven/training/HPC_bioinf $VSC_DATA;
  cd $VSC_DATA/HPC_bioinf
  ```

- Submit jobscript cpujob.pbs to the cluster
  ```
  qsub cpujob.pbs
  ```
- Check the status of your job(s) (`qstat`)
- Analyze outputs (i.e. display the output file – commands `cat, more, less filename`)

Optional Slides:

- ❑ (Brief) Linux Intro
- ❑ Python & R with Conda
- ❑ Worker Framework

(Very) Basic Linux Commands

ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the “.” character.

- `$ ls -a` (all)
  Lists all the files (including .* files)
- `$ ls -l` (long)
  Long listing (type, date, size, owner, permissions)
- `$ ls -t` (time)
  Lists the most recent files first
- `$ ls -S` (size)
  Lists the biggest files first
- `$ ls -r` (reverse)
  Reverses the sort order
- `$ ls -ltr` (options can be combined)
  Long listing, most recent files at the end

Moving around

- **P**rint **W**orking **D**irectory
  - displays your current location within the file system.
  - `$ pwd`

- **C**hange **D**irectories
  - changes the position to the specific directory
  - `$ cd dir_name`
  - You can specify directory names in two ways:
    - Absolute pathname (starts from the root of the tree)

      ```
      $ cd /u/home/hpc/test/bin
      ```
    - Relative pathname (relative to your current directory)

      ```
      $ cd
      $ cd .
      $ cd ..
      $ cd test/bin
      ```
    - Use environment variables

      ```
      $ cd $VSC_DATA
      ```

Special directories

- .
  - The current directory.
  - Useful for commands taking a directory argument.
  - Useful to run commands in the current directory
  - ./readme.txt and readme.txt are equivalent.
- ..
  - The parent (enclosing) directory. Always belongs to the . Directory
  - Typical usage:
    `cd ..`
  - ~
    - Shells just substitute it by the home directory of the current user.
  - -
    - `cd -` jump back to the previous directory

Directories

- Create directories
  - The `mkdir` command is used to create directories
  - `$ mkdir dir1 dir2 dir3`
- Remove directories
  - The `rmdir` command removes directories
  - `$ rmdir dir1`
  - `rmdir` will only remove empty directories.
    To remove a non-empty directory, use
    `$ rm -r [DIRECTORY]` instead.

Copy a file

- The `cp` command copies files and directories
- The default behavior will overwrite any existing file(s). The **-i** option overrides this behavior and prompts the user before overwriting the destination file.
- **syntax: cp [OPTIONS] [SOURCE] [DESTINATION]**
  - `$ cp <source_file> <target_file>`
    Copies the source file to the target.
  - `$ cp file1 file2 file3 ... dir`
    Copies the files to the target directory (last argument).
  - `$ cp -i` (interactive)
    Asks for user confirmation if the target file already exists
  - `$ cp -r <source_dir> <target_dir>` (recursive)
    Copies the whole directory.
  - `$ cp -v` (verbose)
    Displays what has been copied

Move or rename files

- Move or rename files and directories: `mv`
- The default behavior will overwrite any existing file(s).
- **syntax: `mv [OPTIONS] [SOURCE] [DESTINATION]`**
  - `$ mv old_name new_name`
    Renames the given file or directory.
  - `$ mv -i` (interactive)
    If the new file already exits, asks for user confirm
- The `mv` command can also be used to move or rename directories
  - `$ mv NewFiles/ OldFiles/`
  - -r option is not necessary

Remove files

- The rm command removes files.
- `$ rm file1 file2 file3 ...`
  Removes the given files.
- `$ rm -i` (interactive)
  Always ask for user confirmation.
- `$ rm -r dir1 dir2 dir3` (recursive)
  Removes the given directories with all their contents.

- Tip:
  Whenever you use wildcards with `rm` (besides carefully checking your typing!), test the wildcard first with `ls`. This will let you see the files that will be deleted. Then press the up arrow key to recall the command and replace the `ls` with `rm`.

Auto-Completion

- Have the shell automatically complete commands or file paths.
- Activated using the **<TAB>** key on most systems
- examples
  - `$ whe<TAB>`
  - `$ whereis`
  - `$ ls -l /etc/en<TAB>`
  - `$ ls -l /etc/environment`
- When more than one match is found, the shell will display all matching results (use **<TAB>** twice)
  - `$ ls -l /etc/host<TAB>`

Command history: Arrow Up

- Previously executed commands can be recalled by using the **Up Arrow** key on the keyboard.
- Most Linux distributions remember the last five hundred commands by default.
- Display commands that have recently been executed
  - The `history` command displays a user's command line history.
  - You can execute a previous command using `![NUM]` where NUM is the line number in history you want to recall.

Displaying file contents

Several ways of displaying the contents of files.

- `$ cat file1`
  displays the contents of the given file.
- `$ cat file1 file2 file3 ...` (concatenate)
  Concatenates and outputs the contents of the given files.
- `$ more file1`
  Display the output of a command or text file one page at a time.
  - Can also jump to the first occurrence of a keyword (/ command).
- `$ less file1`
  Does more than more.
  Doesn't read the whole file before starting.
  Press q to exit

The head and tail commands

- `$ head [-<n>] <file>`
  Displays the first <n> lines (or 10 by default) of the given file.
  Doesn't have to open the whole file to do this!
- `$ tail [-<n>] <file>`
  Displays the last <n> lines (or 10 by default) of the given file.
  No need to load the whole file in RAM! Very useful for huge files.
- `$ tail -f <file>` (follow)
  Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
  Very useful to follow the changes in a log file, for example.

The grep command

- `$ grep <pattern> <files>`
  Scans the given files and displays the lines which match the given pattern.
- `$ grep error *.log`
  Displays all the lines containing error in the *.log files
- `$ grep -i error *.log`
  Same, but case insensitive
- `$ grep -ri error .`
  Same, but recursively in all the files in the current directory and its subdirectories
- `$ grep -v info *.log`
  Outputs all the lines in the files except those containing info.
- http://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples/

wget

- Instead of downloading files from your browser,
  just copy and paste their URL and download them with wget
- main features
  - http and ftp support
  - Can resume interrupted downloads
  - Can download entire sites or at least check for bad links
  - Very useful in scripts or when no graphics are available
    (system administration, embedded systems)

- `$ wget -c http://microsoft.com/customers/dogs/winxp4dogs.zip`

Continues an interrupted download.

Conda Package Management: Python and R

Installing your own packages using conda

Installing Miniconda

- Download the Bash script that will install it from conda.io using, e.g., wget:
  ```
  $ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
  ```
- Once downloaded, run the installation script:
  ```
  $ bash Miniconda3-latest-Linux-x86_64.sh -p $VSC_DATA/miniconda3
  ```
- Optionally, you can add the path to the Miniconda installation to the PATH environment variable in your .bashrc file. This is convenient, but may lead to conflicts when working with the module system or OS, so make sure that you know what you are doing in either case.
  The line to add to your .bashrc file would be:
  ```
  export PATH=${VSC_DATA}/miniconda3/bin:${PATH}
  ```

Installing Python packages using conda

Creating an environment

- First, ensure that the Miniconda installation is in your PATH environment variable. The following command should return the full path to the conda command:

```
$ which conda
```

- If the result is blank, or reports that conda can not be found, modify the `PATH` environment variable appropriately by adding miniconda's bin directory to PATH.
- Creating a new conda environment is straightforward:

```
$ conda create -n bioinf numpy biopython
```

- This command creates a new conda environment called science, and installs a number of Python packages.
- This will default to the latest Python 3 version, if you need a specific version, e.g., Python 2.7.x, this can be specified as follows:

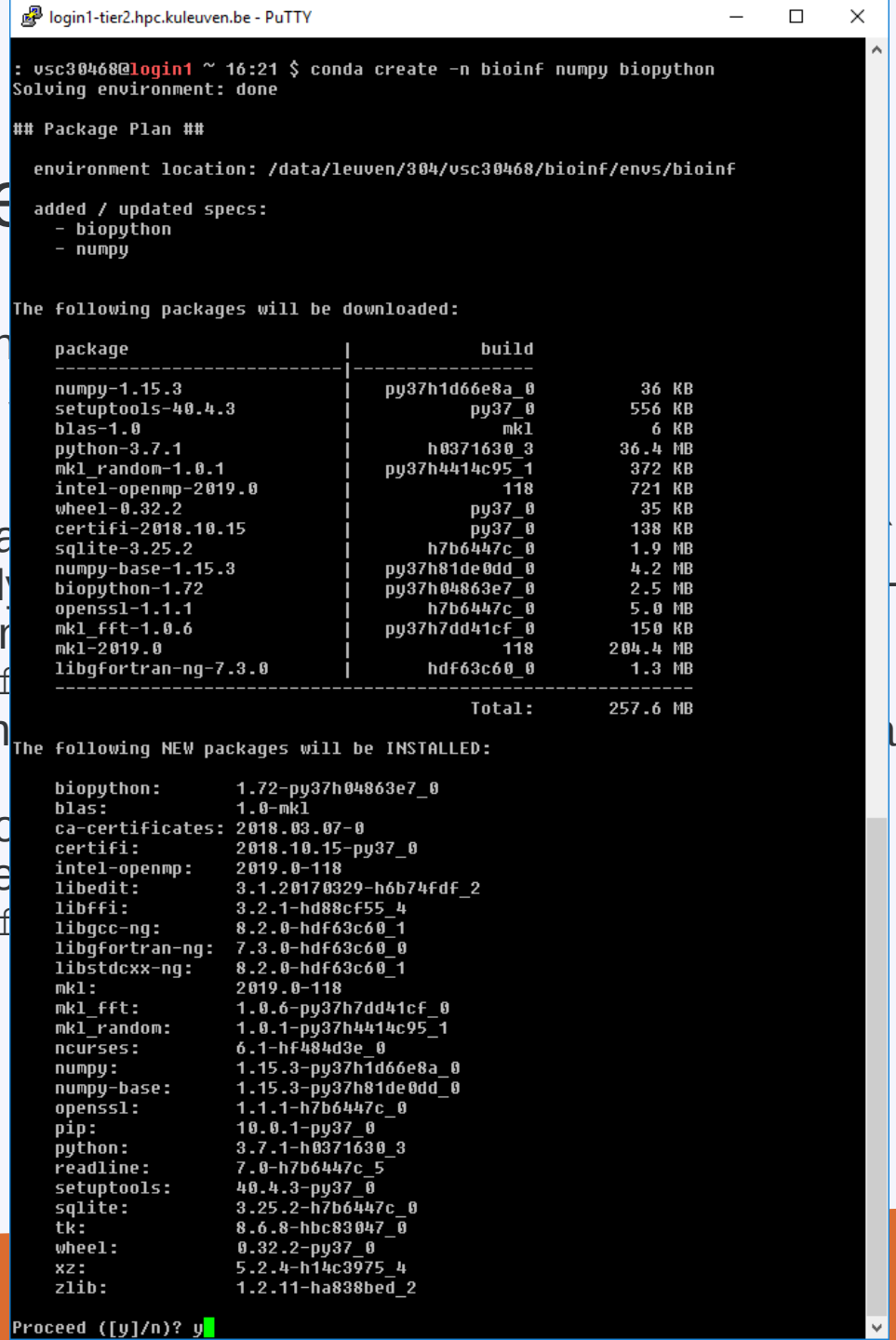
```
$ conda create -n bioinf python=2.7 numpy biopython
```

Installing Python package

Creating an environment

- First, ensure that the Miniconda in
  variable. The following command
  command:
  ```
  $ which conda
  ```
- If the result is blank, or reports tha
  environment variable appropriatel
- Creating a new conda environmer
  ```
  $ conda create -n bioinf
  ```
- This command creates a new con
  number of Python packages.
- This will default to the latest Pytho
  e.g., Python 2.7.x, this can be spe
  ```
  $ conda create -n bioinf
  ```

H.



Installing Python packages using conda

Working with the environment

- To work with an environment, you have to activate it. This is done with, e.g.,
  ```
  $ source activate bioinf
  ```
  Here, science is the name of the environment you want to work in.

Install an additional package

- To install an additional package, e.g., `pandas`, first ensure that the environment you want to work in is activated.
  ```
  $ source activate bioinf
  ```

- Next, install the package:
  ```
  $ conda install pandas
  ```
  Note that conda will take care of all dependencies, including non-Python libraries. This ensures that you work in a consistent environment.

Installing Python packages using conda

Updating/removing

- Using conda, it is easy to keep your packages up-to-date. Updating a single package (and its dependencies) can be done using:
  ```
  $ conda update biopython
  ```

- Updating all packages in the environement is trivial:
  ```
  $ conda update --all
  ```

- Removing an installed package:
  ```
  $ conda remove pandas
  ```

Deactivating an environment

- To deactivate a conda environment, i.e., return the shell to its original state, use the following command
  ```
  $ source deactivate
  ```

Installing Python packages - alternatives

Checking for installed packages

- Pip utility will list all packages that are installed for the Python distribution you are using, including those installed by you, i.e., those in your `PYTHONPATH` environment variable.

- Load the module for the Python version you wish to use, e.g.,:
  ```
  $ module load Python/2.7.14-foss-2018a
  ```

- Run pip:
  ```
  $ pip freeze
  ```
- Note that some packages, e.g., mpi4py, h5py, pytables, ..., are available through the module system, and have to be loaded separately. These packages will not be listed by pip unless you loaded the corresponding module.

- If you have any packages installed in **`.local`** directory, it will always take priority on whatever the Python version used (conda, module, system). That can lead to strange problems, so please avoid using that location.

Installing Python packages - pip

1. Load the appropriate Python module, i.e., the one you want the python package to be available for:
   ```
   $ module load Python/2.7.14-foss-2018a
   ```
2. Create a directory to hold the packages you install, the last three directory names are mandatory:
   ```
   $ mkdir -p "${VSC_DATA}/python_lib/lib/python2.7/site-packages/"
   ```
3. Add that directory to the PYTHONPATH environment variable for the current shell to do the installation:
   ```
   $ export PYTHONPATH="${VSC_DATA}/python_lib/lib/python2.7/site-packages/:${PYTHONPATH}"
   ```
4. Add the following to your .bashrc so that Python knows where to look next time you use it:
   ```
   export PYTHONPATH="${VSC_DATA}/python_lib/lib/python2.7/site-packages/:${PYTHONPATH}"
   ```
5. Install the package, using the prefix install option to specify the install path (this would install the biopython package):
   ```
   $ pip install --user biopython
   ```
   or
   ```
   $ pip install --install-option="--prefix=${VSC_DATA}/python_lib" biopython
   ```

Hands-on 3

1. Install conda environment with the Biopython package in your $VSC_DATA directory
2. Submit jobscript pythonjob1.pbs to the cluster. Python script checks how many times string ATA occurs in a given dna1.txt file. Check the answer in output file.
3. Copy the file pythonjob1.pbs to pythonjob2000.pbs and modify it so that:
   - You will be notified by e-mail when job finishes or aborts
   - Name of the job is count-string-2000
   - Instead of Python from 2018a toolchain conda Python is used
   - First action that the job does is running multiply.sh script
   - Instead of count_substr1.py count_substr2000.py script is used
   - The script looks for ATG string instead of ATA

   Submit the job and check how many times string ATG occurs in dna2000.txt file.

Installing R packages using conda

Creating an environment

- First, ensure that the Miniconda installation is in your PATH environment variable. The following command should return the full path to the conda command:

  ```
  $ which conda
  ```

- If the result is blank, or reports that conda can not be found, modify the `PATH` environment variable appropriately by adding iniconda's bin directory to PATH.

- Creating a new conda environment is straightforward:

  ```
  $ conda create -n science -c r r-essentials r-rodbc
  ```

  This command creates a new conda environment called science, and installs essentials and required packages.

Installing R packages using conda

Working with the environment

- To work with an environment, you have to activate it. This is done with, e.g.,
  ```
  $ source activate science
  ```
  Here, science is the name of the environment you want to work in.

Install an additional package

- To install an additional package, e.g., `r-ggplot2`, first ensure that the environment you want to work in is activated.
  ```
  $ source activate science
  ```

- Next, install the package:
  ```
  $ conda install -c r r-ggplot2
  ```
  Note that conda will take care of all independencies. This ensures that you work in a consistent environment.

Installing R packages using conda

Updating/removing

- Using conda, it is easy to keep your packages up-to-date. Updating a single package (and its dependencies) can be done using:
  ```
  $ conda update r-rodbc
  ```
- Updating all packages in the environement is trivial:
  ```
  $ conda update --all
  ```
- Removing an installed package:
  ```
  $ conda remove r-mass
  ```

Deactivating an environment

- To deactivate a conda environment, i.e., return the shell to its original state, use the following command
  ```
  $ source deactivate
  ```

Installing other R packages using conda

- Installing CRAN package:

```
$ conda skeleton cran readr
$ conda build r-readr
$ conda install --use-local r-readr
```

  Doing that for the first time you need to install conda-build before:

```
$ conda install conda-build
```

- Some packages not available in r-essentials are still available on conda channels, in that case, it's simple:

```
$ conda config --add channels r; conda install r-readxl
```

Installing R packages – alternatives

1. Load the appropriate R module, i.e., the one you want the package to be available for:
   ```
   $ module load R/3.5.0-foss-2018a-bare
   ```

2. start R and install the package from there:
   ```
   > install.packages("DEoptim")
   ```

3. Alternatively you can download the desired package:
   ```
   $ wget cran.r-
   project.org/src/contrib/Archive/DEoptim/DEoptim_2.0-0.tar.gz
   ```

   And Install the package from the command line:
   ```
   $ R CMD INSTALL DEoptim_2.2-3.tar.gz  -l /$VSC_HOME/R/
   ```

4. These packages might depend on the specific R version, so you may need to reinstall them for the other version.

Worker Framework

Glossary

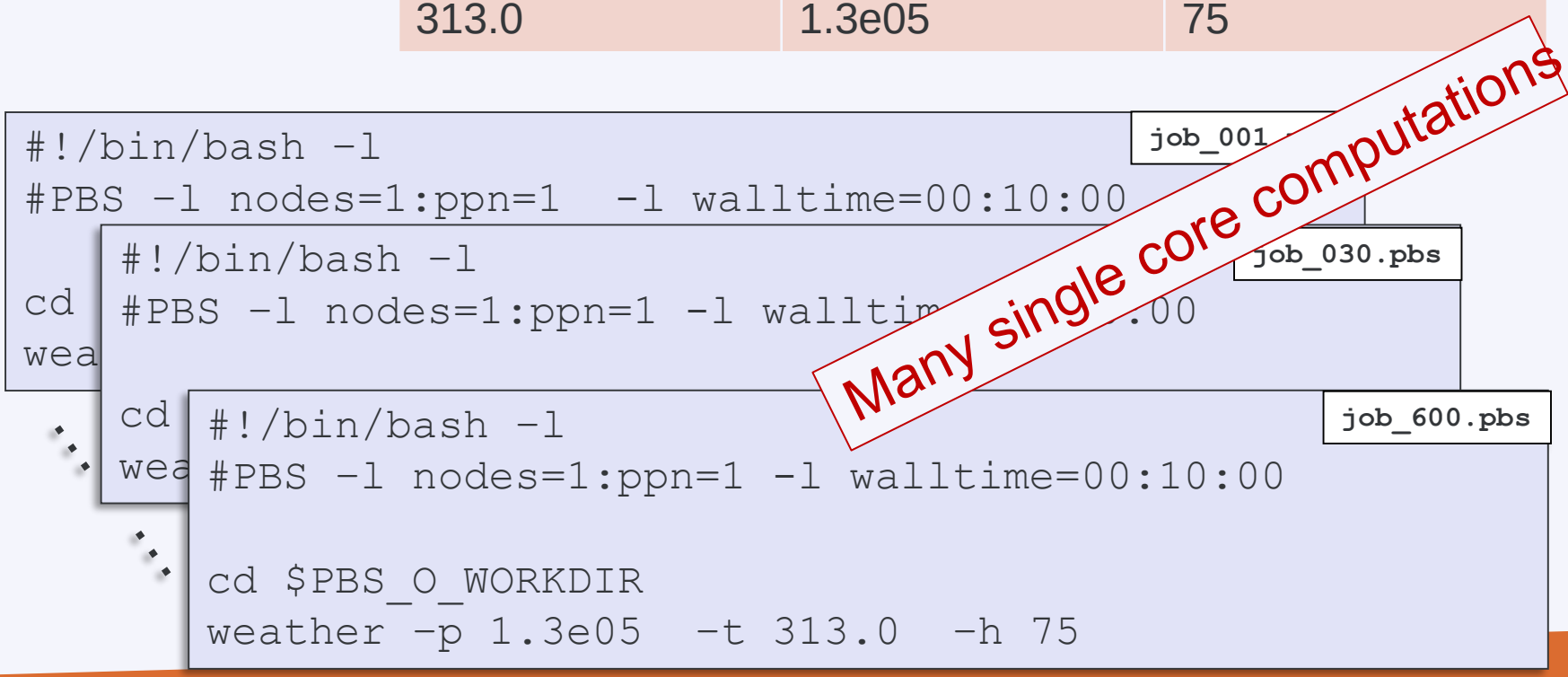
- **Serial application**: a program that runs a single process, with a single thread. All computations are done sequentially, i.e., one after the other, no explicit parallelism is used.
- **Multi-core application**: a multi-core application uses more than one core of processor during its execution by running multiple threads, also called a shared memory application.
- **Distributed application**: an application that uses multiple compute nodes for its computations, concurrent computations are executed as processes. These processes communicate by exchanging messages, typically implemented by calls to an MPI library. Messages can be used to exchange data and coordinate the execution.

Parallel Computing

- Serial:
  - one program, on one core
- ’Embarrassingly parallel’ problems:
  - lots of runs of one program, with different parameters
- Problems that require ’real’ parallel algorithms
  - OpenMP
  - MPI : Message Passing Interface

Use case: parameter exploration

temperature	pressure	humidity
293.0	1.0e05	87
…	…	…
313.0	1.3e05	75



Solution: worker with `-data`

temperature	pressure	humidity
293.0	1.0e05	87
…	…	…
313.0	1.3e05	75

data.csv

job.pbs

```
#!/bin/bash -l
#PBS -l nodes=5:ppn=20  -l walltime=01:20:00

cd $PBS_O_WORKDIR
weather -p $pressure  -t $temperature  -h $humidity
```

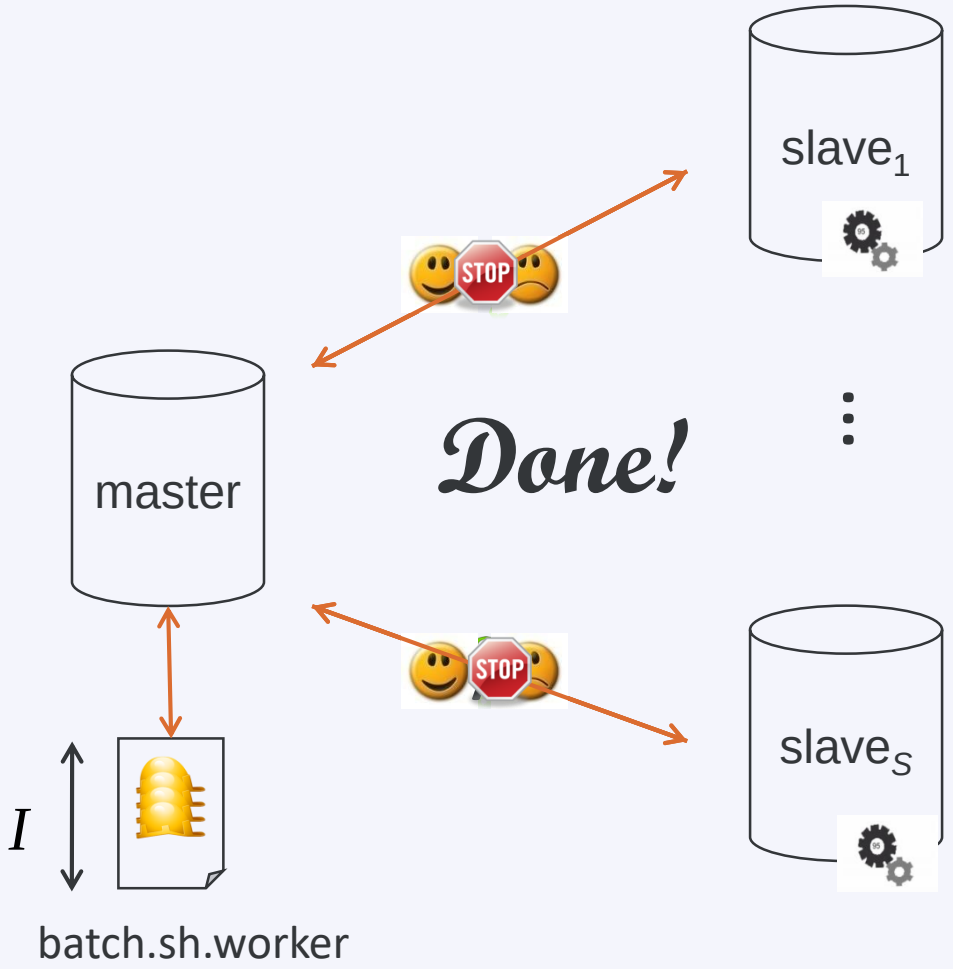
```
$ module load worker
$ wsub  -data data.csv  -batch job.pbs
```

Data exploration: steps

- Write PBS script with parameters
- Create Excel sheet with data
  - Convert to CSV format
- Submit with `wsub`
  - walltime is time to complete all work items

$$walltime_{\text{job}} \geq \frac{N \cdot walltime_{\text{work item}}}{nodes \cdot ppn}$$

worker processing: informally



slave queries for work

master sends work

slave notifies on success/failure, queries for more work

master sends stop

$$nodes = S + 1$$

Resuming jobs: `wresume`

- Resumi

```
$ wresume  -l walltime=1:30:00  -jobid 445948
```

- Redoing failed work items

```
$ wresume  -jobid 445948  -retry
```

How to use worker well?

- Many work items, i.e., #work items/#proc >> 1
- time(work item) > 1 minute
- Work item is not multithreaded
- Work item is multithreaded
  - will work, but user must be careful to request the right resources
  - Use `-threaded <n>` flag with `wsub`
- There be dragons: licensing!

worker & conflicts

- worker module only required for
  - job submission, i.e., `wsub`, `wresume`
  - data aggregation, …, e.g., `wcat`, `wreduce`, …
- No need to load in PBS script
  - use `module purge`
  - minimizes conflicts
  - work items run in own Bash shell
- However, MPI may be problematic
  - e.g., `mpi4py`

worker & multithreading

- Some software uses multithreading automatically, e.g.,
  - R
  - Matlab
- Will use as many threads as there are cores, regardless of system load
  - 36 cores/node
  - 36 work items/node

} 36 × 36 threads/node

Oversubscription: *very* inefficient!!!

Controlling number of threads

- R, most of the time: `OMP_NUM_THREADS=1`

program_pe.pbs

```
#!/bin/bash -l
#PBS -N my-pe
#PBS -l
walltime=1:00:00,nodes=5:ppn=20

module load R
cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=1
Rscript  program  $a  $b
```

- Matlab
  - Use `maxNumCompThreads(1)` function call
  - Use compiler flag: `mcc -singleCompThread …`

Help on worker

- See documentation http://worker.readthedocs.io/
- Each command has help, use `-h`

Epilogue

Questions

- Helpdesk:
  hpcinfo@kuleuven.be or
  https://admin.kuleuven.be/icts/HPCinfo_form/HPC-info-formulier
- If you need support:
  Explain your problem in detail; provide jobID; path to your job scripts
- VSC web site: http://www.vscentrum.be/
- VSC documentation: https://vlaams-supercomputing-centrum-vscdocumentation.readthedocs-hosted.com/en/latest/
  VSC agenda: training sessions, events
- Systems status page:
  http://status.kuleuven.be/hpc
  https://wwsw.vscentrum.be/en/user-portal/system-status