Northwestern
INFORMATION TECHNOLOGY

Introduction to Quest

Michael Stroh,
with assistance from the RCS team

Outline

- Section 1 – What is Quest?
- Section 2 – How to interact with Quest
- Section 3 – Navigating the file system
- Section 4 – Software on Quest
- Section 5 – Job submission

Section 1
What is Quest?

Quest is Northwestern's High-Performance Compute Cluster (HPC)

Why "high-performance"?

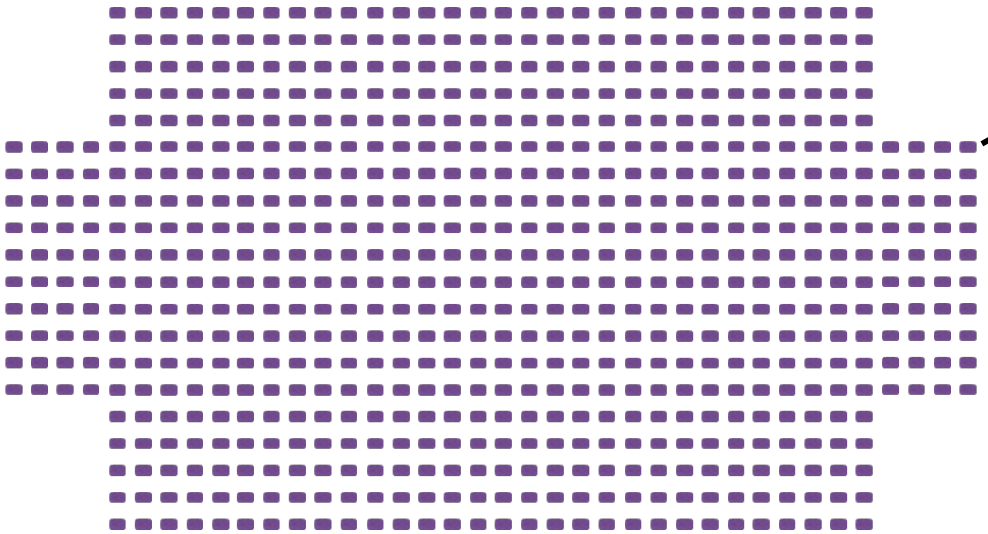
- The computers available at Quest are much more powerful and stable than a typical personal computer or workstation.

Why "cluster"?

- Quest consists of hundreds of computers that can work in harmony with each other.

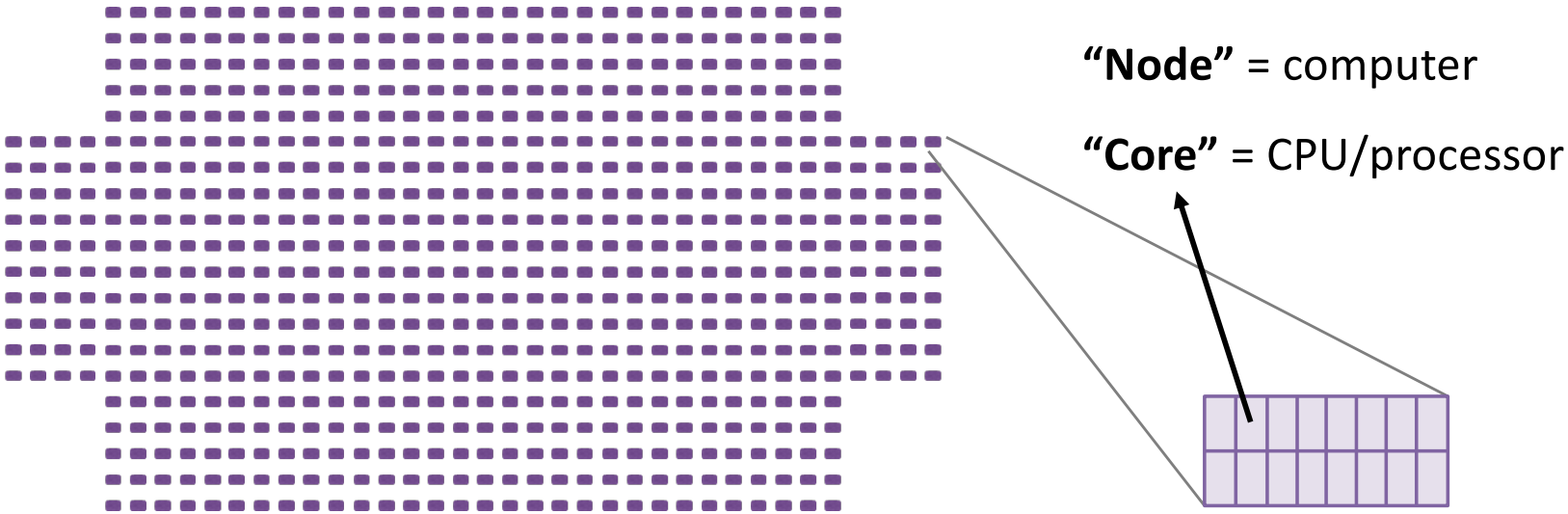
Quest is Northwestern's High-Performance Compute Cluster (HPC)

Quest consists of 800+ nodes

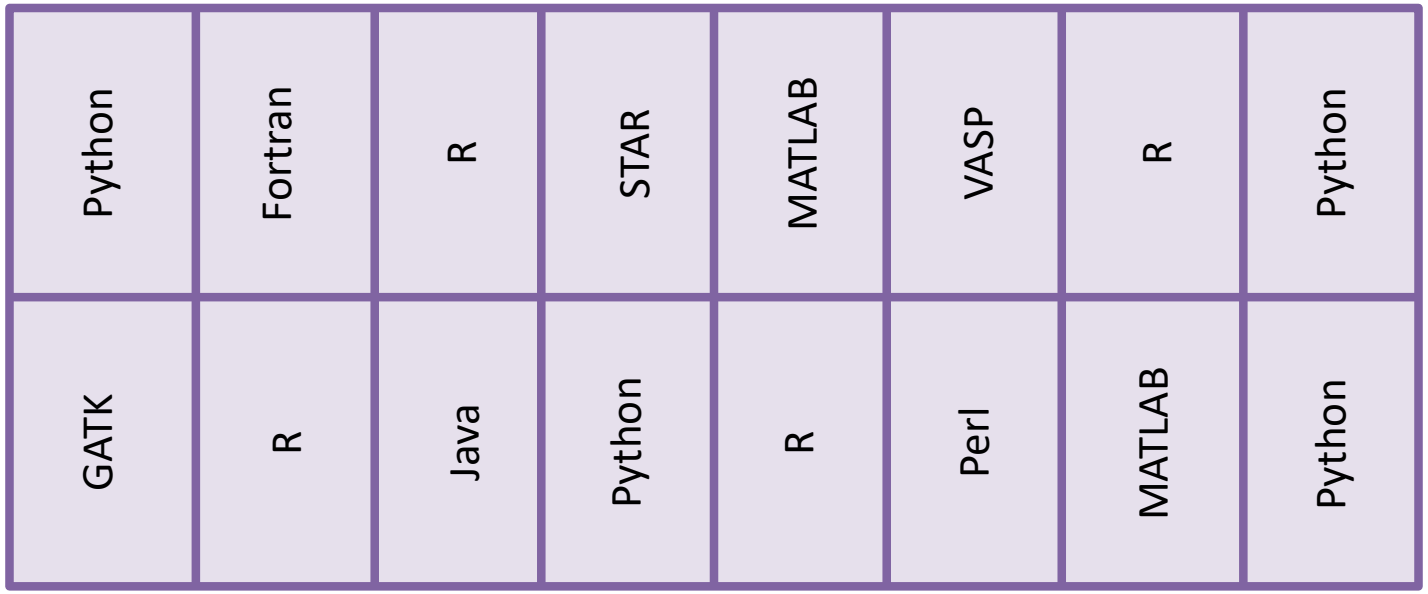


“Node” = computer

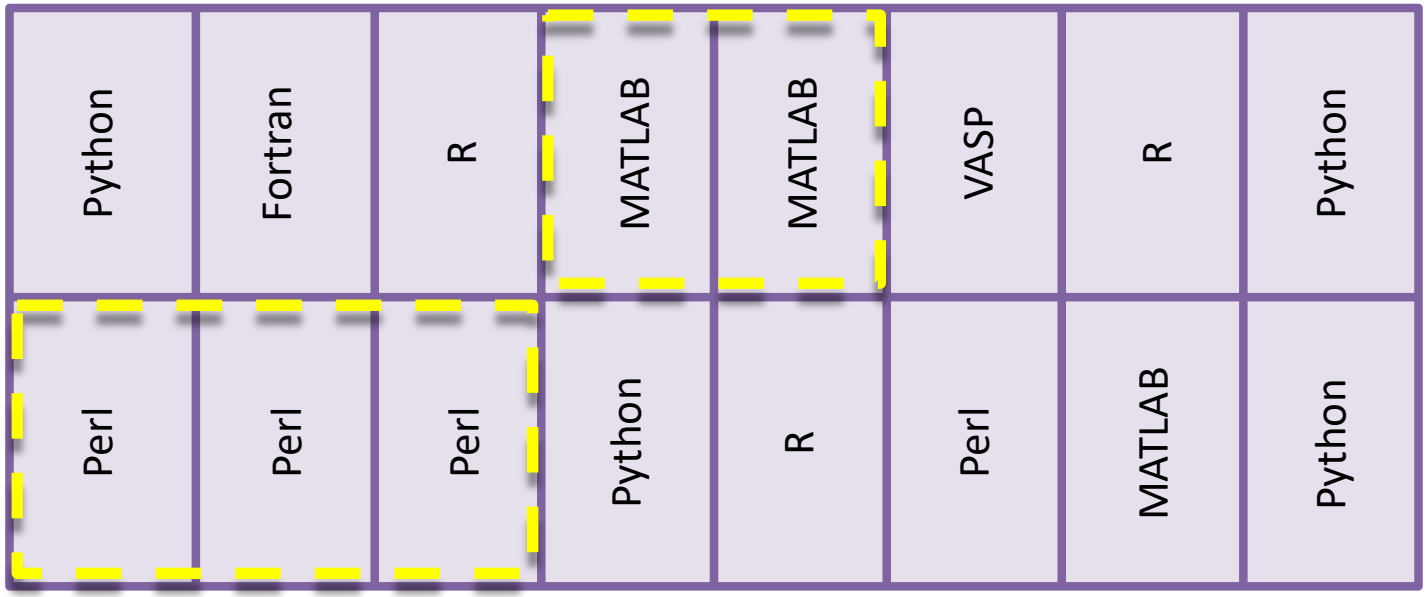
Each node has 40-64 cores



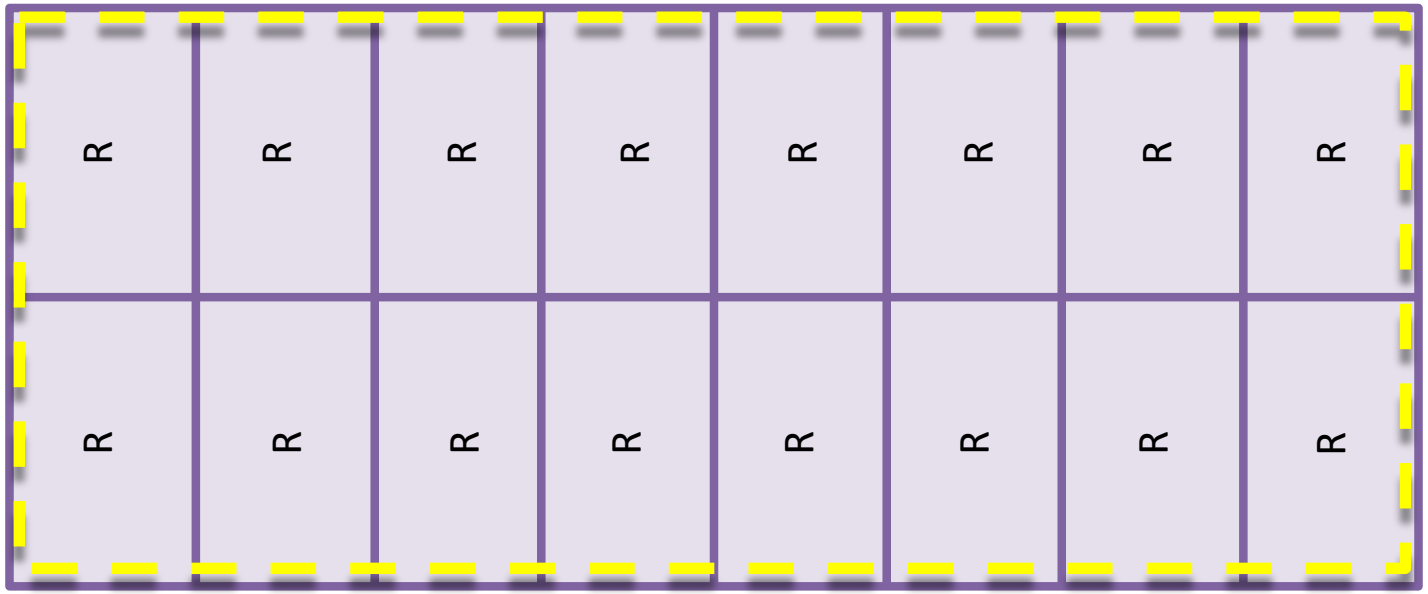
Each node can run separate jobs on each core



Jobs can be parallelized to multi-core

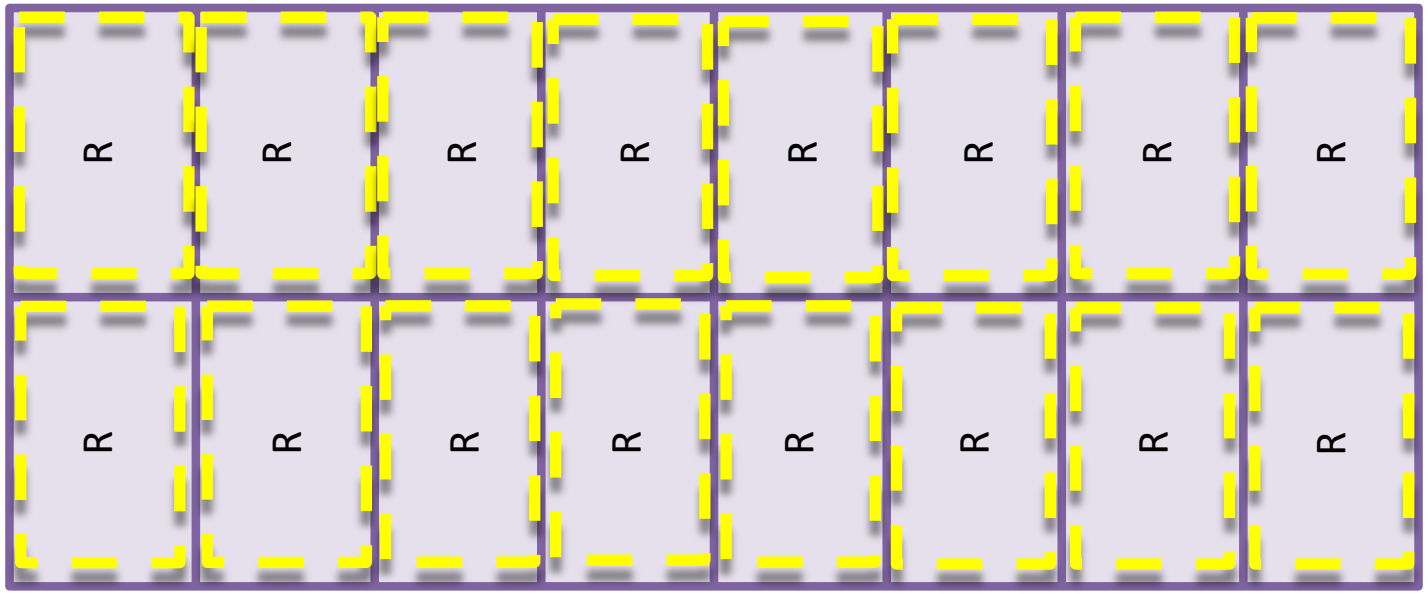


A node can run a job with 40-64 cores at a time



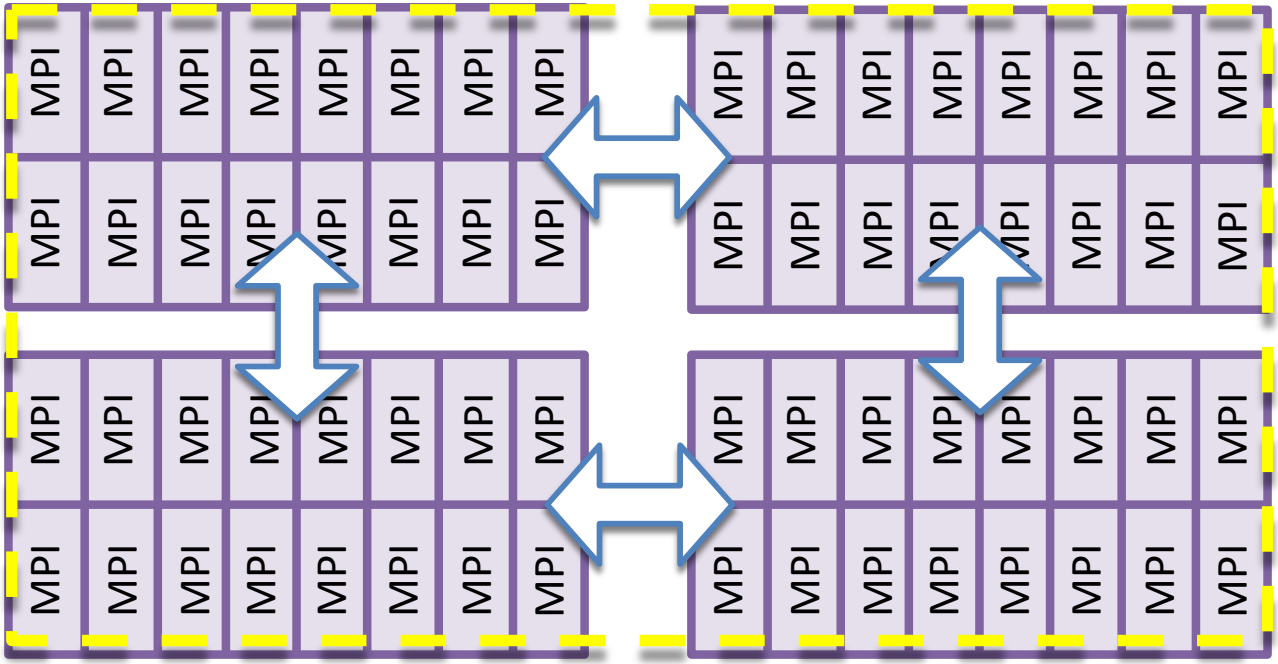
= one multi-core job

You can run thousands of jobs at a time



= a single job

Jobs can even run across multiple nodes



No job is too small to run on Quest

Python	Fortran	R	STAR	MATLAB	VASP	R	Python
GATK	R	Java	Python	R	Perl	MATLAB	Python

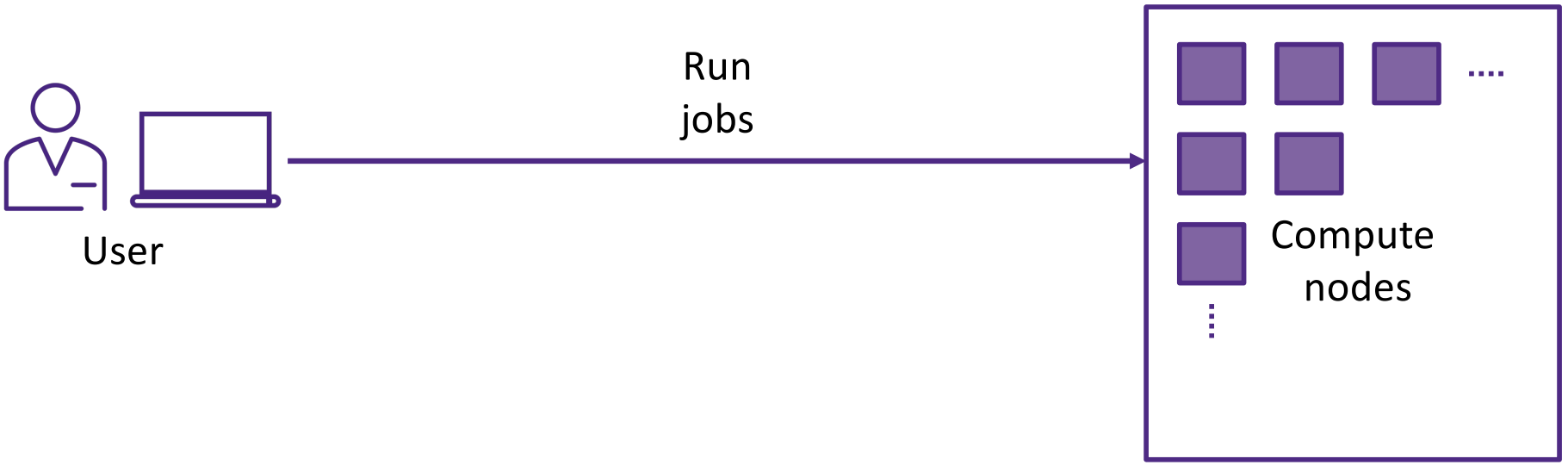
Make use of this opportunity to learn how to use a cluster. This could be useful to put on your CV or highlight on a graduate school application!

Outline

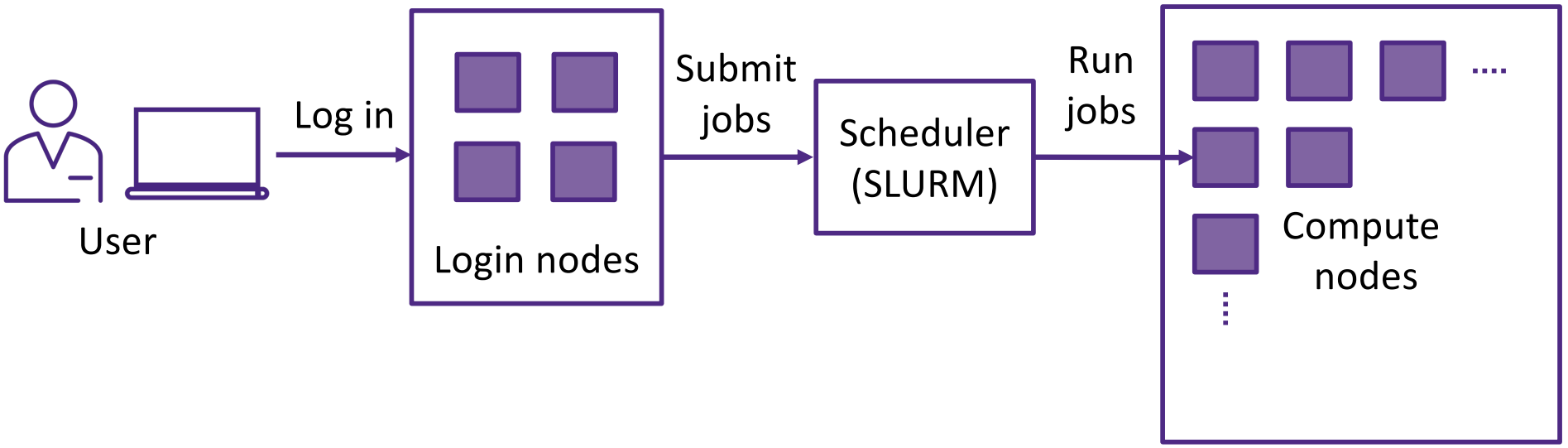
- Section 1 – What is Quest?
- Section 2 – How to interact with Quest
- Section 3 – Navigating the file system
- Section 4 – Software on Quest
- Section 5 – Job submission

Section 2
How to interact with Quest

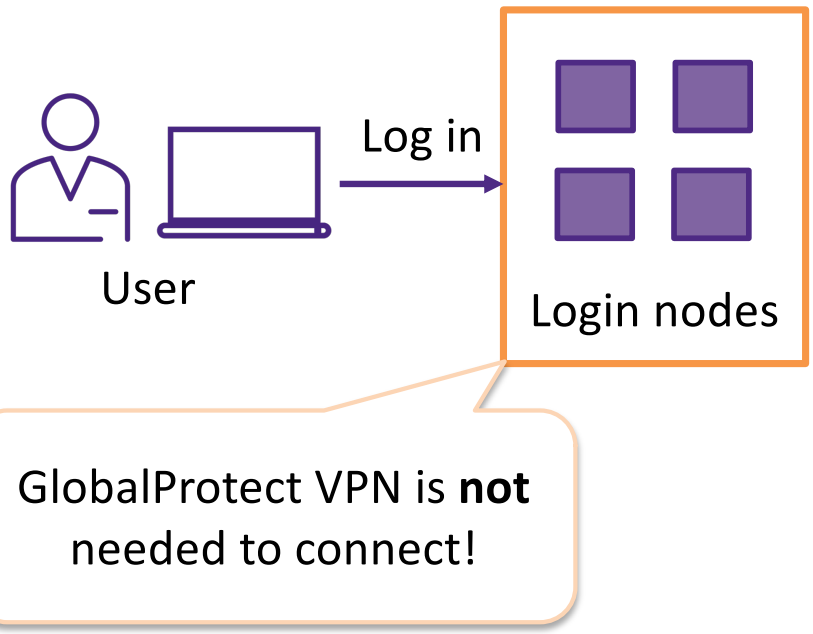
How to access the Quest compute nodes



Log into the login nodes & submit jobs to the scheduler



Resources are limited on Log-in Nodes



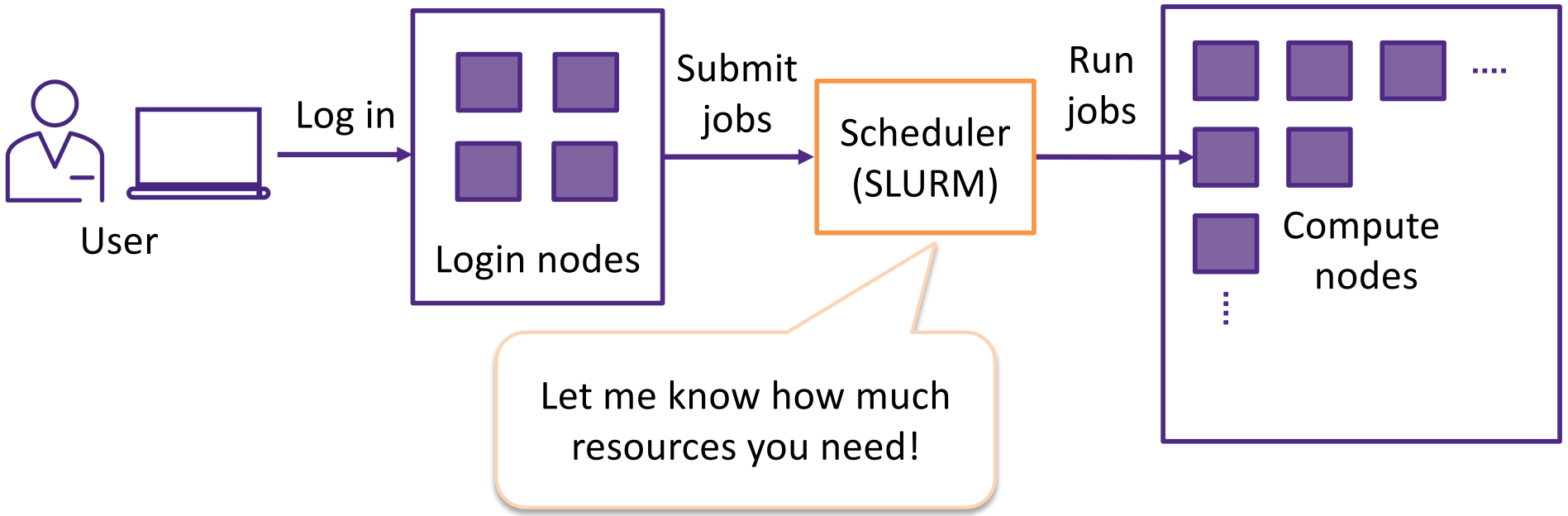
What are they for?

- Landing point on Quest for all users
- Edit files
- Troubleshoot small problems

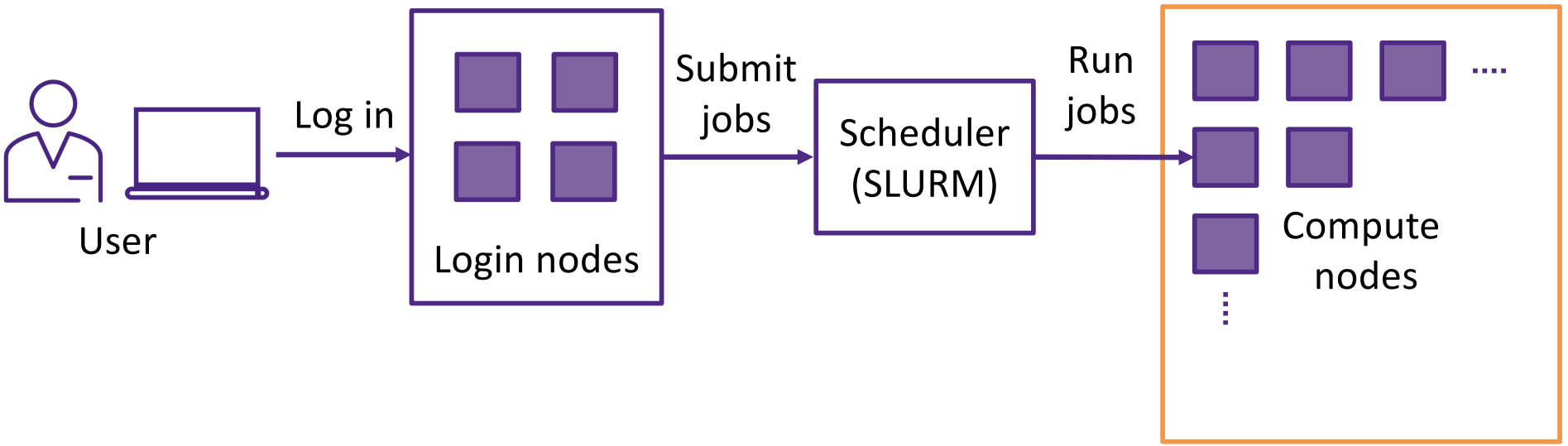
Characteristics

- Limited computational resources (4 CPUs, 6 GB memory)
- Constantly shared by many users

Log into the login nodes & submit jobs to the scheduler

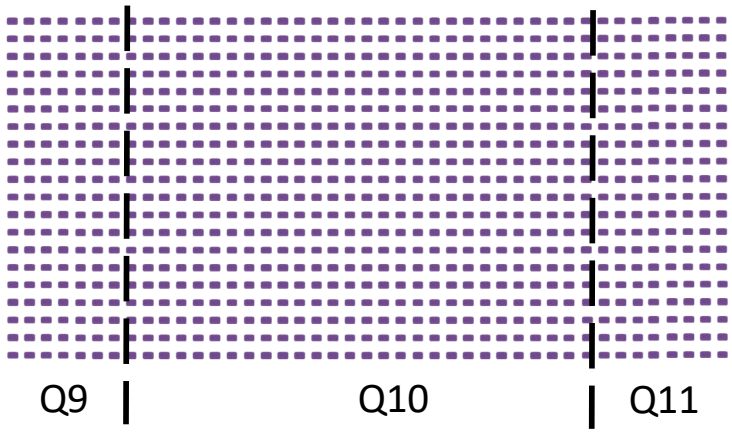


Log into the login nodes & submit jobs to the scheduler



Technical specifications of compute nodes vary

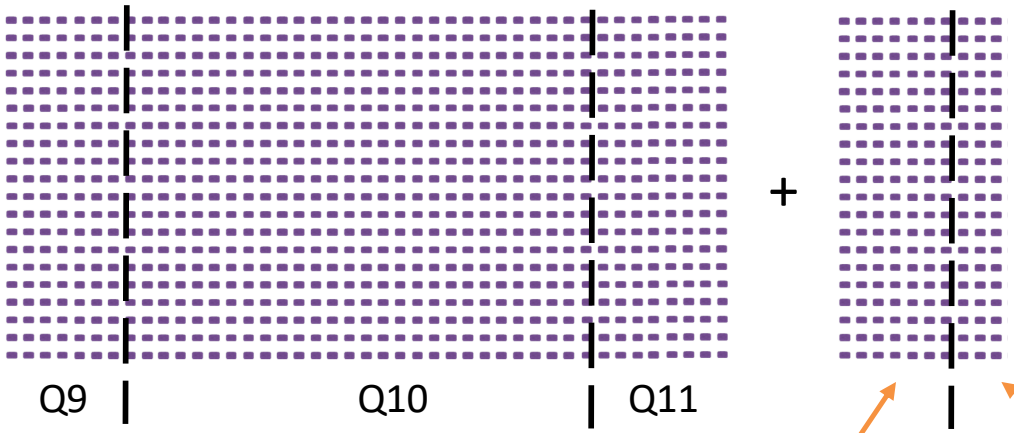
If we take Regular Compute Nodes as an example…



Generation	# cores per node	Total memory	Memory per core
Quest 9	40	192 GB	≈4.8 GB
Quest 10	52	192 GB	≈3.7 GB
Quest 11	64	256 GB	≈4.0 GB

There are several generations of nodes!

High-resource nodes for special needs



GPU Nodes

- ✓ 2 GPUs + 52 CPUs each
- ✓ 192 GB memory

High-memory Nodes

- ✓ 0.5 – 2 TB memory

Section 2 Demo

Outline

- Section 1 – What is Quest?
- Section 2 – How to interact with Quest
- Section 3 – Navigating the file system
- Section 4 – Software on Quest
- Section 5 – Job submission

Section 3
Navigating the file system of Quest using the command line

Quest file system is shared across all nodes

What does this mean?

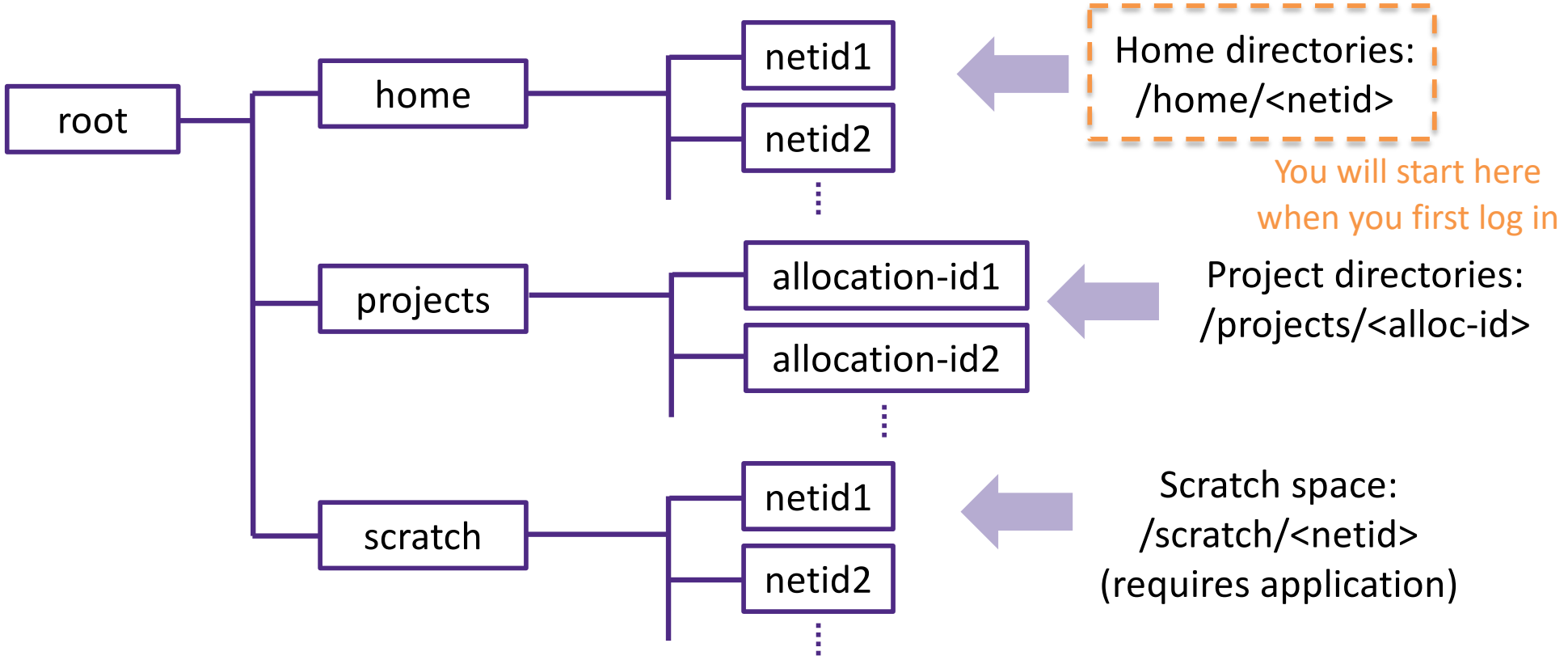
- Whether you are on a log-in node, compute node, or analytics node, you can access all the same files and folders.

Ok, so… what else will I learn in this section?

- About the 3 most important directories you will be working with on Quest, as well as how to use them
- What that looks like on Quest & how to navigate the space using command line

For more information: https://services.northwestern.edu/TDClient/30/Portal/KB/ArticleDet?ID=1546

The 3 most important directories on Quest

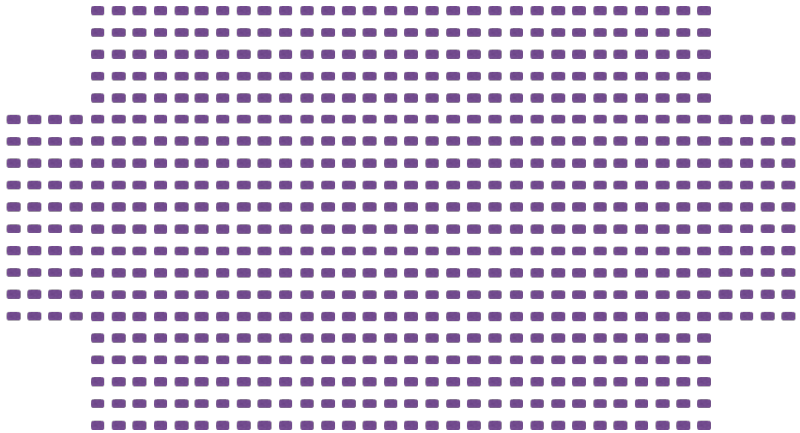


What's the difference between the three?

	/home/<netid>	/projects/<alloc-id>	/scratch/<netid>
Who has access?			
Storage space			
Intended use			
To check usage:			
Backups			

Types of allocations

- Research allocation I & II (“pXXXXX”)
- Buy-in storage (“bXXXX”)
- Classroom allocation (“eXXXXX”)



Note:

You cannot store contractually or legally restricted data!

Section 3 Demo

Connect to Quest

SSH to Quest

- `ssh` – Remote login to Quest

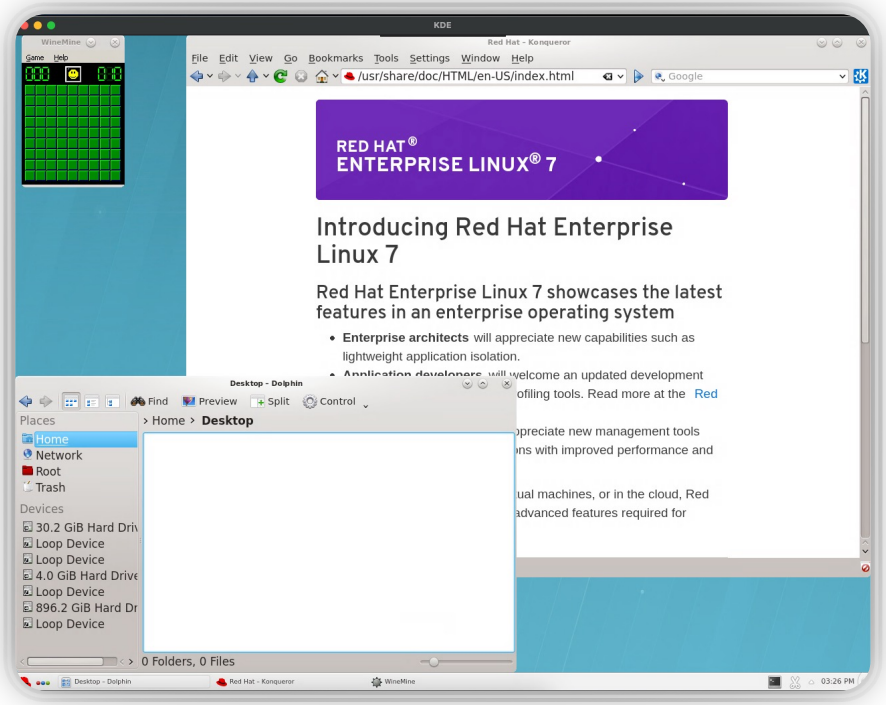
```
$ ssh -X <NetID>@quest.northwestern.edu
[netid@quser31 ~]$ pwd
/home/<NetID>
```

Which shell to use?

- Bash – supported by Quest
- Only use another shell if the rest of your group is using it.

Connect to Quest

FastX3 for a Graphical User Interface (GUI)



Instructions at: https://services.northwestern.edu/TDClient/30/Portal/KB/ArticleDet?ID=1511

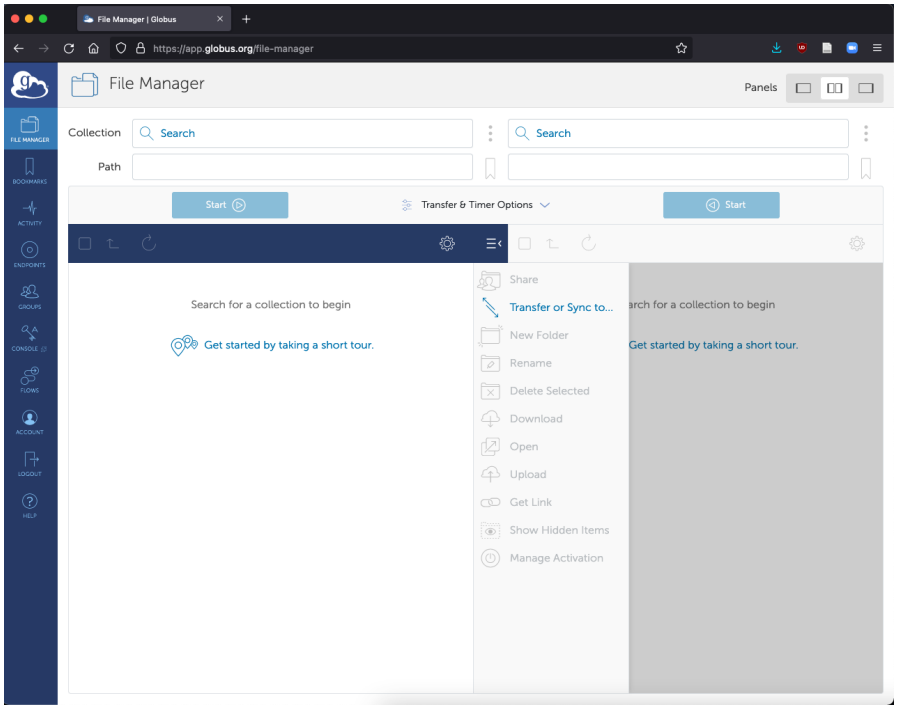
Use one of the following for your host:

- ~~quest.northwestern.edu~~
- quser31.ci.northwestern.edu
- quser32.ci.northwestern.edu
- quser33.ci.northwestern.edu
- quser34.ci.northwestern.edu

Connect to Quest

File transfers

- Globus (globus.org)
  - Large file transfers
  - Great for cluster-to-cluster file transfers
- SFTP
- SCP
- RSYNC
- Cyberduck (or other FTP clients)



Useful Bash Commands

Navigating directories (cont'd)

- `pwd` – print working directory
- `cd <dirname>` – change directory

```
[netid@quser31 ~]$ pwd
/home/netid
[netid@quser31 ~]$ cd /projects/e32016/simple_job
[netid@quser31 ~]$ pwd
/projects/e32016/simple_job
[netid@quser31 ~]$ cd ~
[netid@quser31 ~]$ pwd
/home/netid
```

Useful Bash Commands

Other useful commands (cont'd)

- `logout / exit` – end SSH session

```
[netid@quser31 ~]$ exit
logout
Connection to quest.it.northwestern.edu closed.
My-Local-Computer:~ username$
```

Commands specifically for Quest

```
[netid@quser31 ~]$ homedu
[netid@quser31 ~]$ groups
[netid@quser31 ~]$ checkproject
[netid@quser31 ~]$ checkscratch 10
```

For more info on bash commands…

Experienced users:

- `man <command>` – look at manual of the given command

Beginning users:

- Websites like linuxize.com offer command manual with examples.

Bash scripting YouTube tutorial

Outline

- Section 1 – What is Quest?
- Section 2 – How to interact with Quest
- Section 3 – Navigating the file system
- Section 4 – Software on Quest
- Section 5 – Job submission

Section 4
Software on Quest

You do not have admin/sudo access on Quest

What does this mean?

- You cannot install software system-wide
- If a software installation requires access to certain system directories, you might not be able to install directly

How do we get around this problem?

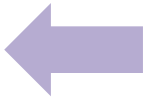
1. Local installation: Install software to your /home or /projects
2. Modules: IT performs system-wide software installations as “modules”
3. Virtual environments & containers: Install via Anaconda, Singularity

Avoid using system-level software at all costs!

What does this mean?

- There are some default software packages installed in places like: /usr/lib:/usr/lib64:/usr/local/lib:/usr/bin etc. etc.
- For example:

```
[netid@quser31 ~]$ which python
/usr/bin/python
[netid@quser31 ~]$ which python3
/usr/bin/python3
```



DO NOT USE THESE

Why should I avoid these?

- These packages exist to ensure stability of Quest, and are often extremely old.
- There are only a bare-minimum number of packages available in this way.

1. Local installations to `/home` or `/projects`

How do I do this?

- Oftentimes, you can install software packages in your home directory, in places like `/home/<netid>/.local/bin` or `/home/<netid>/bin`
- Specific installation instructions are found on each software package's website

When to use /home vs. /projects?

- For example, installing to /projects is helpful to ensure that everyone in the research group uses the same version of the software.
- If you install to /projects, make sure to add the path to the software in your $PATH variable!

2. Modules: What are they?

What are “modules”?

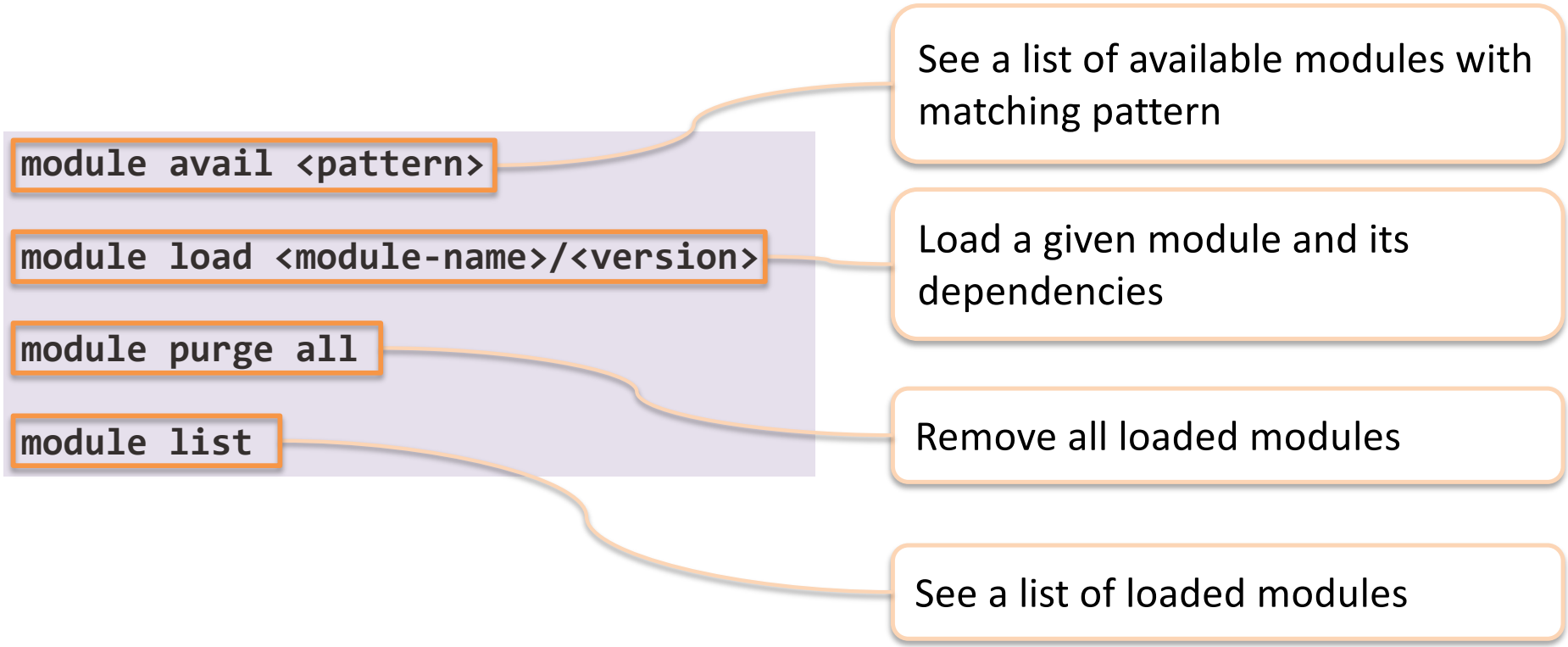
- A library of centrally-installed software packages that you can easily load/unload.
- Popular software like R, python, Java, MPI have several different versions packaged and you can point to any you like!

Why do we need modules?

- Although we generally encourage users to install their own software packages, some packages are used by many users.
- Modules allow system-wide installation of these popular software packages, but in a way that a user can pick and choose versions, what to make available etc.

Note: Always specify the version of a module rather than loading the default!

2. Modules: Useful commands for managing



3.1 Anaconda virtual environments

What are anaconda virtual environments?

- You can create several isolated environments to which you can install different versions of Python/R packages, tailored to the needs of each project

What are the advantages of this approach?

- No more version conflicts: Each environment is self-contained
- No need to keep track: conda solves the dependencies for you
- Reproducibility: Easier for others to reproduce your work

Panopto tutorials

3.2 Singularity Containers

YouTube tutorial

What are containers?

- Just like the virtual environment, but even more isolated and independent!
- It does this by virtualizing the Operating System, but using the same physical resources

What are the advantages of this approach?

- Independent of host OS: Anaconda virtual envs are still dependent on host OS
- Super user privileges: Install any packages you like, even at the OS-level
- Reproducibility

Note

- sudo privileges required to build the container → locally, cloud builder

Section 4 Demo

Video Outline

- Section 1 – What is Quest?
- Section 2 – How to interact with Quest
- Section 3 – Navigating the file system
- Section 4 – Software on Quest
- Section 5 – Job submission

Section 5
Job submission on Quest

Types of jobs: batch vs. interactive

- **Batch job**
  - Submit your job as a pre-written bash script
  - Benefit – submit & forget about it
- **Interactive job**
  - Run interactive session on the compute nodes
  - Benefit – exploratory work, troubleshooting etc.

Copy sample job

```
$ ssh -X <netID>@quest.northwestern.edu
[netid@quser31 ~]$ pwd
/home/netid
[netid@quser31 ~]$ ls
...
[netid@quser31 ~]$ cp -r /projects/e32016/simple_job .
[netid@quser31 ~]$ ls
... simple_job ...
[netid@quser31 ~]$ cd simple_job
[netid@quser31 ~]$ nano example_submit.sh
```

What does a job script look like?

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-py37-test

python --version
python slurm_test.py
```

Your script might look like this.
To run this batch job on Quest, you would enter:

```
$ sbatch <script-filename>.sh
```

Let's break down the components!

What does a job script look like?

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-py37-test

python --version
python slurm_test.py
```

Tell Slurm what you want with these “headers”

Load any modules you need

Run your cool program!

What does a job script look like?

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-py37-test

python --version
python slurm_test.py
```

Your script might look like this.
To run this batch job on Quest, you would enter:

```
$ sbatch <script-filename>.sh
```

Let's break down the components!

Side note: there are short-forms for headers

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu
```

=

```
#!/bin/bash
#SBATCH -A p12345
#SBATCH -p short
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -t 00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH -J sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu
```

We will use the long-form in this tutorial for clarity!

Indicate allocation in “--account” or “-A”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

Which allocation should I use?

→ Try running “groups <netid>”

```
[abc1234@quser21 ~]$ groups abc1234
abc1234 p12345 b1000 e10001
```

Indicate partition in “--partition” or “-p”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How do I know which partition to select?

If your allocation starts with “**p**” or “**e**”:

→ “short” “normal” “long”

→ “gengpu” for GPU

→ “genhimem” for high-memory

Indicate partition in “--partition” or “-p”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How do I know which partition to select?

If your allocation starts with “**b**”:

→ generally, use allocation name

e.g. “**b1234**”

There are some exceptions (e.g. b1042)

→ ask allocation manager

Indicate number of nodes in “--nodes” or “-N”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How many nodes should I select?

Unless your program explicitly states that it can run across nodes, default to nodes=1.

If your job doesn’t rely on software like **OpenMPI, MPICH, Intel-MPI,** then most likely nodes=1.

Indicate number of cores in “--ntasks-per-node”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-

python --version
python slurm_test.py
```

How many cores should I select?

Unless your program is specifically designed to run on multiple cores, select “1”.

Examples of multi-core:

Python’s scikit-learn, R’s DESEQ2 etc.

NOTE: you still need to explicitly tell the software!

Indicate number of cores in “--ntasks-per-node”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.ed

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slur

python --version
python slurm_test.py
```

Side note: Useful environment variable

The variable `$SLURM_NTASKS` can be used within the script to point to the number of cores.

e.g.

```
python slurm_test.py -n $SLURM_NTASKS
```

Indicate time in “--time” or “-t”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How much time should I request?

→ Start with a conservative estimate.

→ Once you have a better idea of how much time your job will take, adjust accordingly.

Indicate memory in “--mem” or “--mem-per-cpu”

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How much memory should I request?

You can take the same approach and start with a conservative estimate, but make sure to adjust later.

This affects your job’s pending time a lot!

If you request a lot of memory, your job might be restricted to certain nodes.

Why not request maximum time and memory?

- **On Quest, job priority is determined using the user's FairShare Score**
  - FairShare Score makes sure that everyone gets their fair share
  - Requesting a ton of resources will lower your job priority and increase pending time

More on fairshare scores

How to check resource utilization

Useful commands:

- ”checkjob” – detailed summary
- “seff” – efficiency summary

Can you spot any problematic trends that should be adjusted?

```
[quest_demo@quser23]$ seff 2261088
------------------------------------------------
JOB EFFICIENCY
------------------------------------------------
Job ID: 2261088
Cluster: quest
User/Group: netid/netid
State: COMPLETED (exit code 0)
Cores: 4
CPU Utilized: 00:32:42
CPU Efficiency: 24.84% of 00:32:42 core-walltime
Job Wall-clock time: 00:32:42
Memory Utilized: 366.84 MB
Memory Efficiency: 3.66% of 10.00 GB
```

How to check resource utilization

Useful commands:

- ”checkjob” – detailed summary
- “seff” – efficiency summary

Can you spot any problematic trends that should be adjusted?

```
[quest_demo@quser23]$ seff 2261088
------------------------------------------------
JOB EFFICIENCY
------------------------------------------------
Job ID: 2261088
Cluster: quest
User/Group: netid/netid
State: COMPLETED (exit code 0)
Cores: 4
CPU Utilized: 00:32:42
CPU Efficiency: 24.84% of 00:32:42 core-walltime
Job Wall-clock time: 00:32:42
Memory Utilized: 366.84 MB
Memory Efficiency: 3.66% of 10.00 GB
```

Set job name for ease of identification

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How to decide job name?

This is for your own convenience – Slurm will not care what you name your job. Something short and descriptive can be useful later.

Set output file path

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-p

python --version
python slurm_test.py
```

How to decide output file path?

- Any print statements will go here
- Slurm expects you to specify a **full path** with a file name, not just a directory.
- Direct your output to **/projects**! Some programs have a lot of print statements and can overflow your /home directory

Set email preferences and address

```
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-miniconda3
source activate /projects/intro/envs/slurm-py37-

python --version
python slurm_test.py
```

What do I set for email preferences?

- --mail-type=ALL if you want all
- Options are START, END, FAIL, or ALL with any combination

Types of jobs: batch vs. interactive

- **Batch job**
  - Submit your job as a pre-written bash script
  - Benefit – submit & forget about it
- **Interactive job**
  - Run interactive session on the compute nodes
  - Benefit – exploratory work, troubleshooting etc.

Interactive job resources are requested similarly

```
[netid@quser23]$ srun --account=p12345 --time=1:00:00 -n 1 -p short --mem=1G --pty bash -l
----------------------------------------
srun job start: Wed Oct  5 14:11:43 CDT 2022
Job ID: XXXXXXX
Username: netid
Queue: short
Account: p12345
----------------------------------------
The following variables are not guaranteed to be the same in prologue and the job run
script
----------------------------------------
PATH (in prologue) :
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/lpp/mmfs/bin:/opt/ibutils/bin:/opt/z
eek/bin:/home/netid/.local/bin:/home/netid/bin
WORKDIR is: /home/netid
----------------------------------------
[netid@qnode8075]$ # code away!
```

“salloc” similar to “srun” but requires SSH

```
[quest_demo@quser23]$ salloc --x11 --account=p12345 --partition=short --nodes=1 --tasks-per-
node=1 --time=00:20:00
salloc: Pending job allocation 8575069
salloc: job 8575069 queued and waiting for resources
salloc: job 8575069 has been allocated resources
salloc: Granted job allocation 8575069
salloc: Waiting for resource configuration
salloc: Nodes qnode6702 are ready for job
[quest_demo@quser23]$ ssh -X qnode6702
Last login: Mon Dec 13 12:46:35 2021 from quser23
[quest_demo@qnode6702 ~]$ # Code away!
```

Useful commands 1/4: see pending and running jobs

```
[netid@quser23]$ squeue -u netid # inspect pending or running jobs
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2632440    p12345 sample_j tempuser  R       0:02      1 qnode5057
           2632441    p12345 sample_j tempuser  R       0:02      1 qnode5057
```

Useful commands 2/4: see all jobs within a time window

```
[netid@quser23]$ sacct -X -u netid --starttime=9/24/22 --endtime=9/28/22 # past jobs
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
1835975         testjob     short     p12345          1    TIMEOUT      0:0
2261088         testjob    normal     p12345          1  COMPLETED      0:0
2261164         testjob    normal     p12345          1 CANCELLED+      0:0
```

Useful commands 3/4: See script of past jobs

```
[netid@quser23]$ sacct -j <jobid> -B
Batch Script for <jobid>
--------------------------------------------------------------------------------
#!/bin/bash
#SBATCH --account=p12345
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
#SBATCH --job-name=sample_job
#SBATCH --output=outlog
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=email@northwestern.edu

module purge all
module load python-anaconda3
source activate /projects/intro/envs/slurm-py37-test

python slurm_test.py
```

Useful commands 4/4: estimate pending time

```
[netid@quser23]$ sbatch --test example_submit.sh
sbatch: Job 2632434 to start at 2021-04-29T09:03:56 using 1 processors on nodes qnode5057
in partition w10001
```

Example jobs can be found on our GitHub

https://github.com/nuitrcs/examplejobs

For CIERA REU students

```
cp /projects/b1094/software/.bashrc $HOME/.bashrc
```

Thank you!

CIERA members:
ciera-computing@northwestern.edu

Questions:
quest-help@northwestern.edu