Running programs on a HPC cluster

Goals of this Class

- **Understand**
  - A cluster provides CPU, RAM, and Disk space.
  - Terms: Cluster, Partition, Node, Job, and Job Step
  - Job life-cycle
- **Be able to**
  - Run a job on the cluster
  - Run batch and batch array jobs
  - Monitor/cancel jobs

Laptop Not Powerful Enough

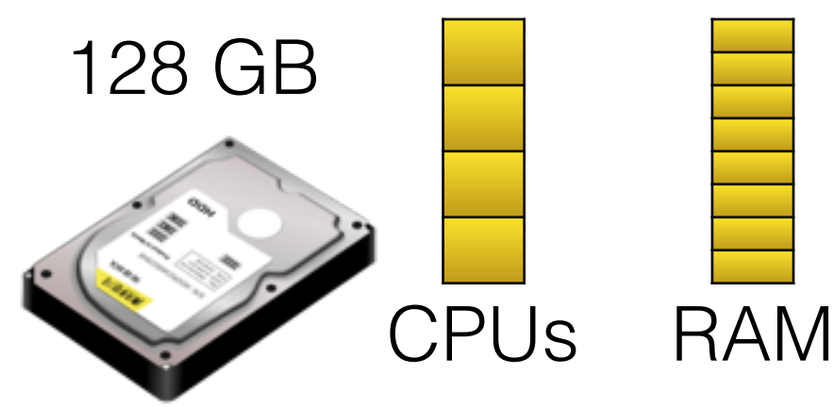
Symptoms

projects **TOO BIG** for your hard drive

processing many files takes **FOREVER**

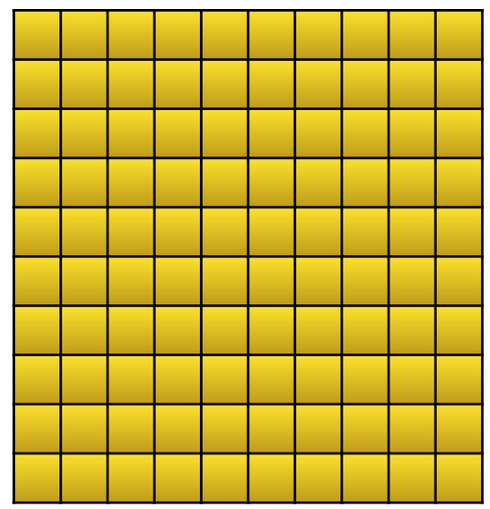
high RAM commands **CRASH**

The Problem

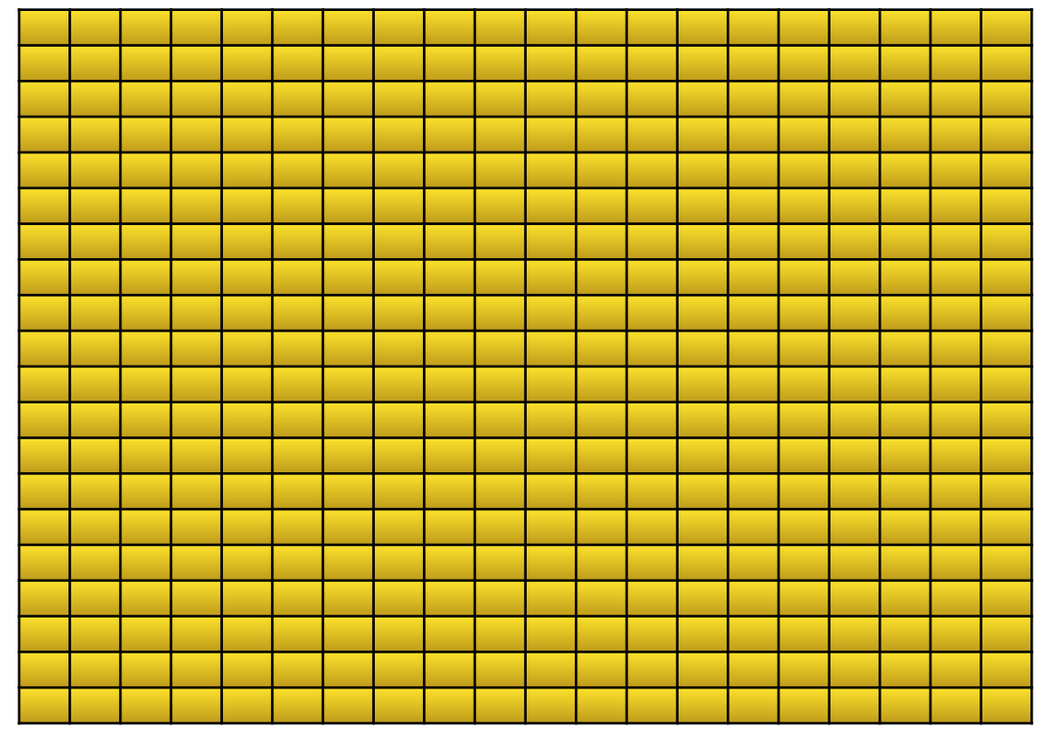


Cluster Has More Power

500,000 GB



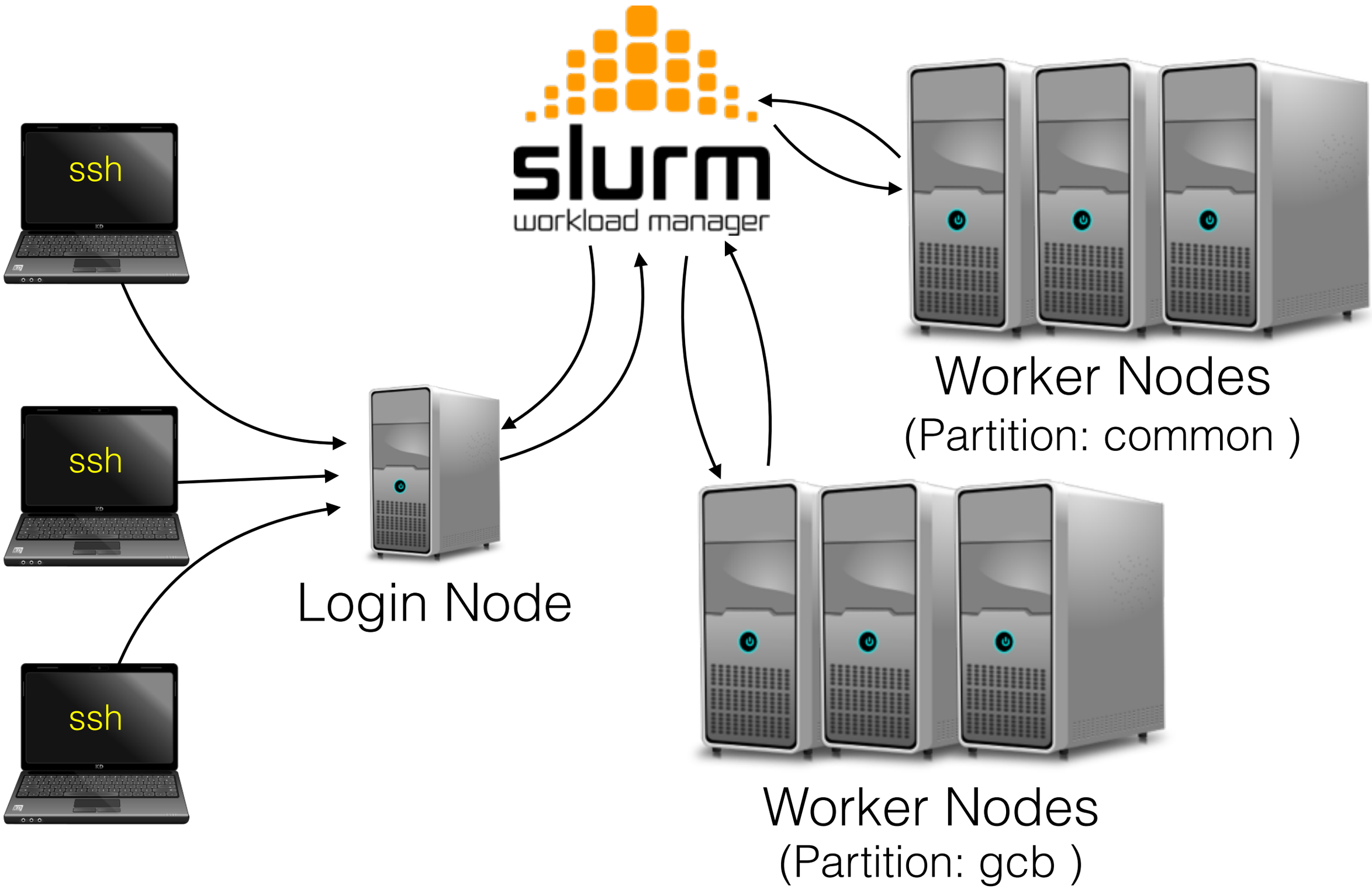
CPUs



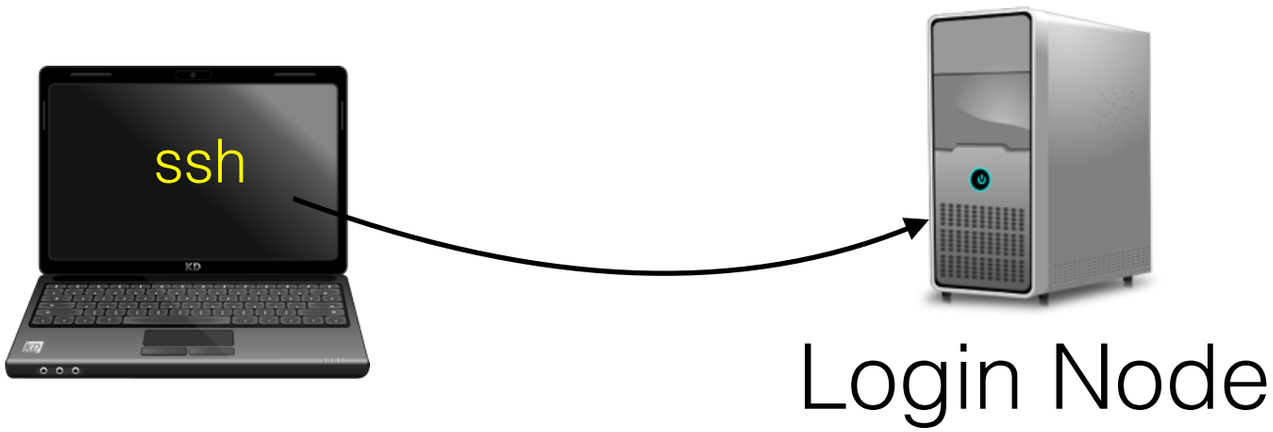
RAM

Cluster
Just a bunch of computers (nodes)

Slurm manages our cluster



ssh to the Login Node




```
$ ssh <netid>@dscr-slogin-01.oit.duke.edu
...
...password:XXXXX
...
...slogin-01  ~ $
```

hostname

what machine am I on?

Run hostname command

```
..slogin-01 $ hostname
dcc-slogin-01
```

This command prints out the name of the machine we are running it on. In this case the login node.

NOTE: Do not run intensive commands on the login node

srun

Slurm run a command in the foreground

Ask slurm to run the hostname command on a worker node

```
..slogin-01 $ srun hostname
srun: job 51 queued and waiting for resources
srun: job 51 has been allocated resources
dcc-adrc-01
..slogin-01 $
```

srun

Specify memory requirements

By default DCC allocates 2G memory per job.
Run hostname command specifying 4 G of RAM (memory)

```
..slogin-01 $ srun --mem=4G hostname
srun: job 51 queued and waiting for resources
srun: job 51 has been allocated resources
dcc-adrc-01
..slogin-01 $
```

Why bother?

- Slurm will stop your job if you use more than requested
- If you are not using 5G of memory it can take longer to get your job scheduled

Interactive Job

typing srun and waiting is tedious

Steps

1. Connect to Login Node
2. Start interactive job using **srun**
3. Run whatever commands you want
4. type **exit** to quit interactive job

```
...slogin-01 $ srun --pty bash
<workernode> $ hostname
```

Getting code onto the Cluster

Works just like on your laptop!

```
$ git clone https://github.com/johnbradley/
scicomp-hpc.git
```

Change into this directory

```
$ cd scicomp-hpc
```

See the files we downloaded

```
$ ls
```

sbatch

Run command(s) in the background

Make a file called countgc.sh using nano:

```
#!/bin/bash
echo "Starting GC counter"
python fasta_gc.py data/E2f1_dna.fasta
```

Run it by using the **sbatch** command:

```
$ sbatch countgc.sh
Submitted batch job 26651766
```

When done Slurm will create an output file(s) based on jobid.

```
$ cat slurm-*.out
```

Slurm Job Lifecycle

1. Slurm creates a Job in the Job Queue with status Pending when a user submits a request.

2. When resources are available Slurm will run Pending jobs. The job state is changed to Running.

3. When a job is finished Slurm removes it from the Job Queue. Slurm records the job in the Accounting List with the final state.

Job Queue - squeue

User	Cmd	State	Job ID
Bob	Star Aligner	Running	123
John	**fastqc…**	**Pending**	**411**

Accounting List - sacct

User	Cmd	State	Job ID
Dan	kalign…	Error	112
John	**fastqc…**	**Complete**	**411**

squeue

shows active job status

Look at your active jobs.

```
$ squeue -u <netid>

JOBID PARTITION     NAME      USER  ST       TIME ...
6335778       all long_ru...  jpb67  R        0:05 …
...
```

Job Status Column

R - Running

P - Pending

Start a long running job the repeat the above command.

```
$ sbatch long_running.sh
```

scancel

Terminate a running Job

Find the job id of that long_running job.

```
$ squeue -u <netid>
```

Stop a single job

```
$ scancel <JOBID>
```

Or stop all jobs for your user

```
$ scancel -u <netid>
```

will cancel interactive jobs

sacct

historical job status

```
$ sacct
       JobID    JobName  ...       State ExitCode
------------ ----------- ... ----------- --------
26705496     countgc.sh ...   COMPLETED      0:0
26705496.ba+      batch ...   COMPLETED      0:0
26705566     countgc.sh ...   FAILED         1:0
26705566.ba+      batch ...   FAILED         1:0
26706541     countgc.sh ...   CANCELED       0:0
26706541.ba+      batch ...   CANCELED      0:15
```

Only shows results from current day by default.
Checkout `starttime` flag to see a better date range.

sacct

How much memory did that use?

```
$ sacct -o JobName,State,MaxRSS,ReqMem
   JobName      State     MaxRSS     ReqMem
---------- ---------- ---------- ----------
countgc.sh  COMPLETED                    2Gc
     batch  COMPLETED      4960K         2Gc
countgc.sh  COMPLETED                    2Gc
     batch  COMPLETED          0         2Gc
countgc.sh  COMPLETED                  400Mn
     batch  COMPLETED      4936K      400Mn
```

- MaxRss / 1024 = MB for use with `sbatch --mem`
- See all options sacct can show: `sacct -e`

sbatch

memory requirements

Change countgc.sh using nano:

```
#!/bin/bash
#SBATCH --mem=400M
python fasta_gc.py data/E2f1_dna.fasta
```

400MB RAM

The `#SBATCH` comment tells sbatch to pretend that the following flag was passed along the command line. This is preferable to typing the flags again and again.

srun and **sbatch** commands share many of the same arguments**.**

sbatch

email when job completes

Add two lines countgc.sh using nano:

```
#!/bin/bash
#SBATCH --mail-type=END
#SBATCH --mail-user=<your_email_address>
#SBATCH --mem=400M
echo "Starting GC counter"
python fasta_gc.py data/E2f1_dna.fasta
```

Run it with sbatch

```
$ sbatch countgc.sh
```

job steps

break job into steps

Create jobsteps.sh using nano:

```
#!/bin/bash
FILENAME=data/E2f1_dna.fasta
srun wc --count-lines $FILENAME
srun python fasta_gc.py $FILENAME
```

Run our sbatch script

```
$ sbatch jobsteps.sh
```

Once it finishes look at

```
$ sacct
```

sbatch --array

make a bunch of jobs

Create array_test.sh using nano:

```
#!/bin/bash
#SBATCH --mem=400
#SBATCH --array=0-4%2
echo $SLURM_ARRAY_TASK_ID
```

The **0-4** part says to run array_test.sh script 5 times with `SLURM_ARRAY_TASK_ID` filled with a number 0-4. The **%2** part says to only run 2 at a time.

```
$ sbatch array_test.sh
```

sbatch --array

use task id to find a filename

Create a text file with the names of all files to process

```
$ ls data/*.fasta > files.txt
```

Change array_test.sh using nano:

```
#!/bin/bash
#SBATCH --mem=400
#SBATCH --array=0-4%2
FILENAME=$(./get_filename.sh files.txt $SLURM_ARRAY_TASK_ID)
echo $FILENAME
```

Run your array job

```
$ sbatch array_test.sh
```

sbatch --array

run one command on many files

Change array_test.sh using nano:

```
#!/bin/bash
#SBATCH --mem=400
#SBATCH --array=1-6%2
FILENAME=$(./get_filename.sh files.txt $SLURM_ARRAY_TASK_ID)
python fasta_gc.py $FILENAME
```

This script will determine GC of 5 files in the *data* directory storing result into separate slurm*.out files.

```
$ sbatch array_test.sh
```

sinfo

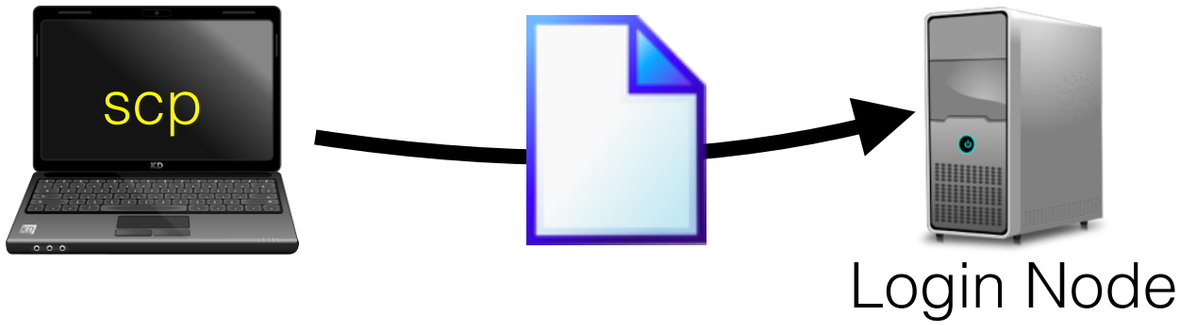
How busy is the cluster?

Show status of nodes in the "common" partition

```
$ sinfo -p common
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
common    up     90-00:00:0 2      idle  dcc-core-[1-2]
common    up     90-00:00:0 1      down  dcc-core-3
common    up     90-00:00:0 1      mix   dcc-core-7
common    up     90-00:00:0 3      alloc dcc-core-[4-6]
```

scp

Getting your files onto the Cluster



From a **NEW** Terminal or Bash Shell on your **LAPTOP**

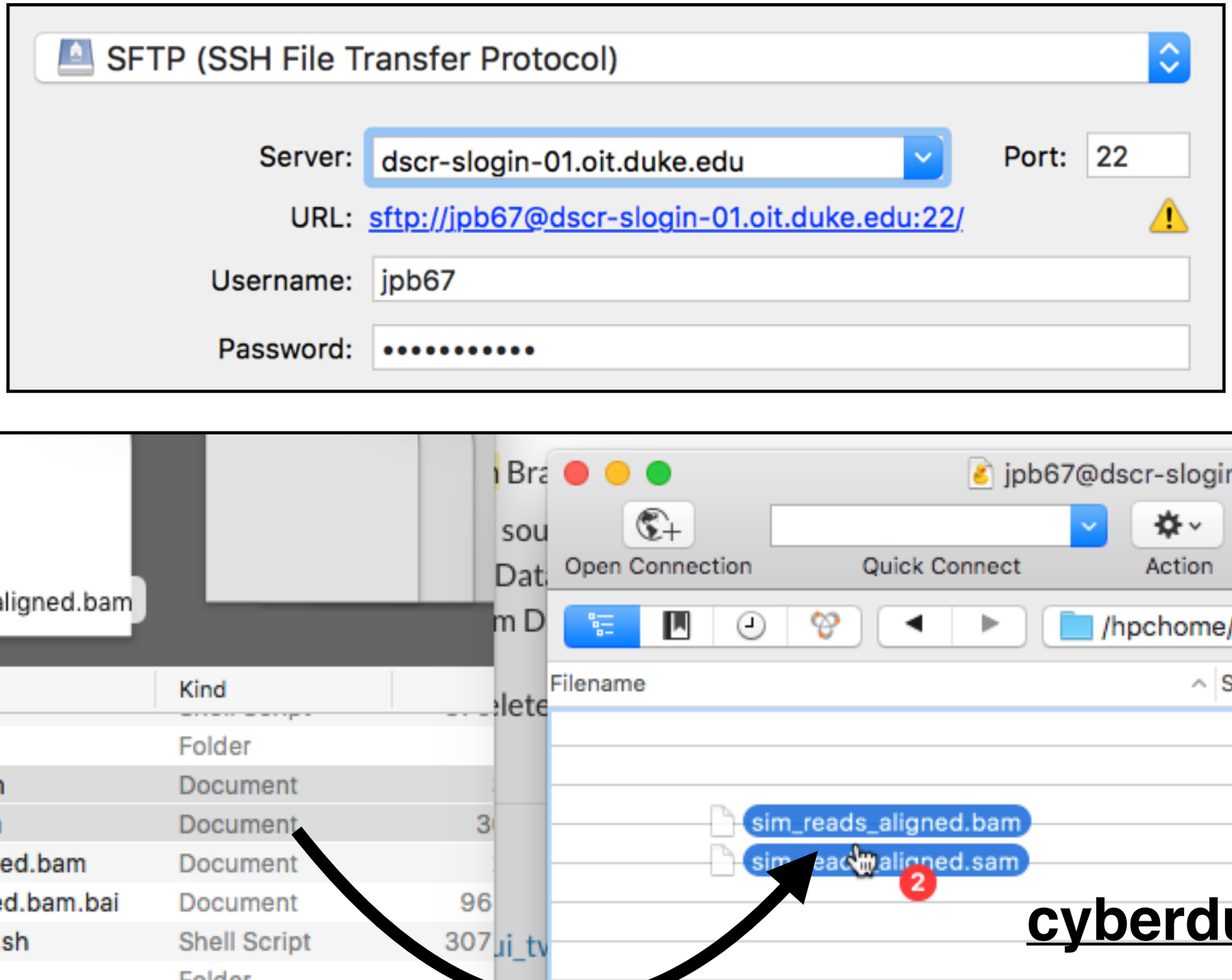
```
echo "Data Staging" > datafile.txt
scp datafile.txt
  <netid>@dscr-slogin-01.oit.duke.edu:datafile.txt
```

Back in your other shell (the interactive job):

```
<workernode> $ cat datafile.txt
```

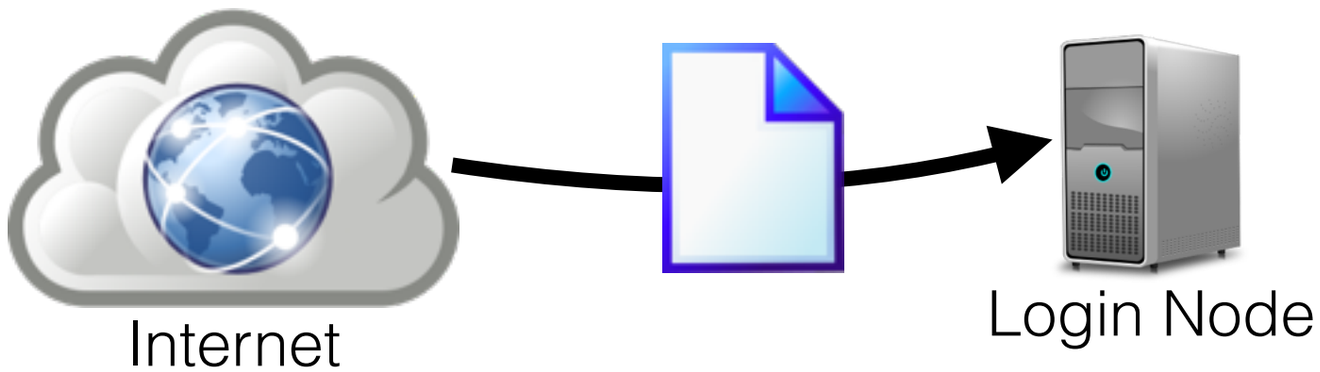
SFTP Client

Getting your files onto the Cluster



wget

Download files onto the Cluster



Download yeast chromosome `I` FASTA file

```
$ wget http://hgdownload.cse.ucsc.edu/
goldenPath/sacCer3/chromosomes/chrI.fa.gz
```

Getting Software Requirements

1. **module** command - powerful
2. Finding software already installed
3. Build it yourself
4. **singularity** command
5. Ask cluster owner for help

module

List and use software

Find all software that is available

```
$ module avail
...
Anaconda2/2.7.13    ... RepeatMasker/4.0.5
```

Load the RepeatMasker/4.0.5 module

```
$ module load RepeatMasker/4.0.5
```

Run software (on interactive node)

```
$ RepeatMasker -species yeast chrI.fa.gz
```

module

run job using a module

Create a file named repmask.sh using nano:

```
#!/bin/bash
module load RepeatMasker/4.0.5
RepeatMasker -species yeast chrI.fa.gz
```

Notice how this explicitly defines the version of the software used.

Run your batch job

```
$ sbatch repmask.sh
```

Example HPC Workflow

1. Start interactive job
2. Get your code and data on the cluster using **git** and **scp**
3. Run software using **srun** to get it working.
4. Run the software using **sbatch** on a small data set
5. Run the software using **sbatch** on all of the data
6. Monitor jobs via **squeue** and cancel with **scancel**
7. Troubleshoot using **sacct** to look at job exit codes and reading the **slurm-*out** files

Help for Slurm commands

All Slurm commands (srun, sbatch, squeue, …) have a -h option.

```
$ sbatch -h
```

Or just google `slurm <command>` to find the online documentation.

http://slurm.schedmd.com/sbatch

slurm
workload manager
Version 16.05

About
- Overview
- What's New
- Slurm Team
- Meetings
- Testimonials
- Legal Notices

Using
- Tutorials
- Documentation
- FAQ
- Publications

sbatch

Section: Slurm Commands (1)
Updated: Slurm Commands
Index

NAME

sbatch - Submit a batch script to Slurm.

SYNOPSIS

sbatch [*options*] *script* [*args*...]

DESCRIPTION

sbatch submits a batch script to Slurm. The batch script may be given to sbatch through a file name on the command line, or if no file name is specified, sbatch will read in a script from standard input. The batch script may contain options preceded with "#SBATCH" before any executable commands in the script.

Two Clusters

DCC

General Purpose

HARDAC

Bioinformatics Focused

Finding Software - DCC

HPC clusters vary in their approach to providing software. For DCC the current method is to just install it in specific locations available to all the nodes.

https://wiki.duke.edu/display/SCSCusers/DSCR+Installed+Applications

You can also reach out to your Point of Contact:

https://wiki.duke.edu/display/SCSC/DSCR+Point+of+Contact+list

Finding Software - HARDAC

Load specific versions of bioinformatics software.

```
$ module avail
bcl2fastq2/2.17.1.14-gcb01  htslib/1.3.1-gcb01        samtools/1.1-f...
bedtools2/2.19.1-gcb01      icu4c/54.1-fasrc01        samtools/1.2-f...
bedtools2/2.25.0-fasrc01    infernal/1.0.2-fasrc01    samtools/1.3.1...
$ module load bedtools2/2.19.1-gcb01
$ bedToBam -i afile.bed -g human.hg18.genome > afile.bam
```

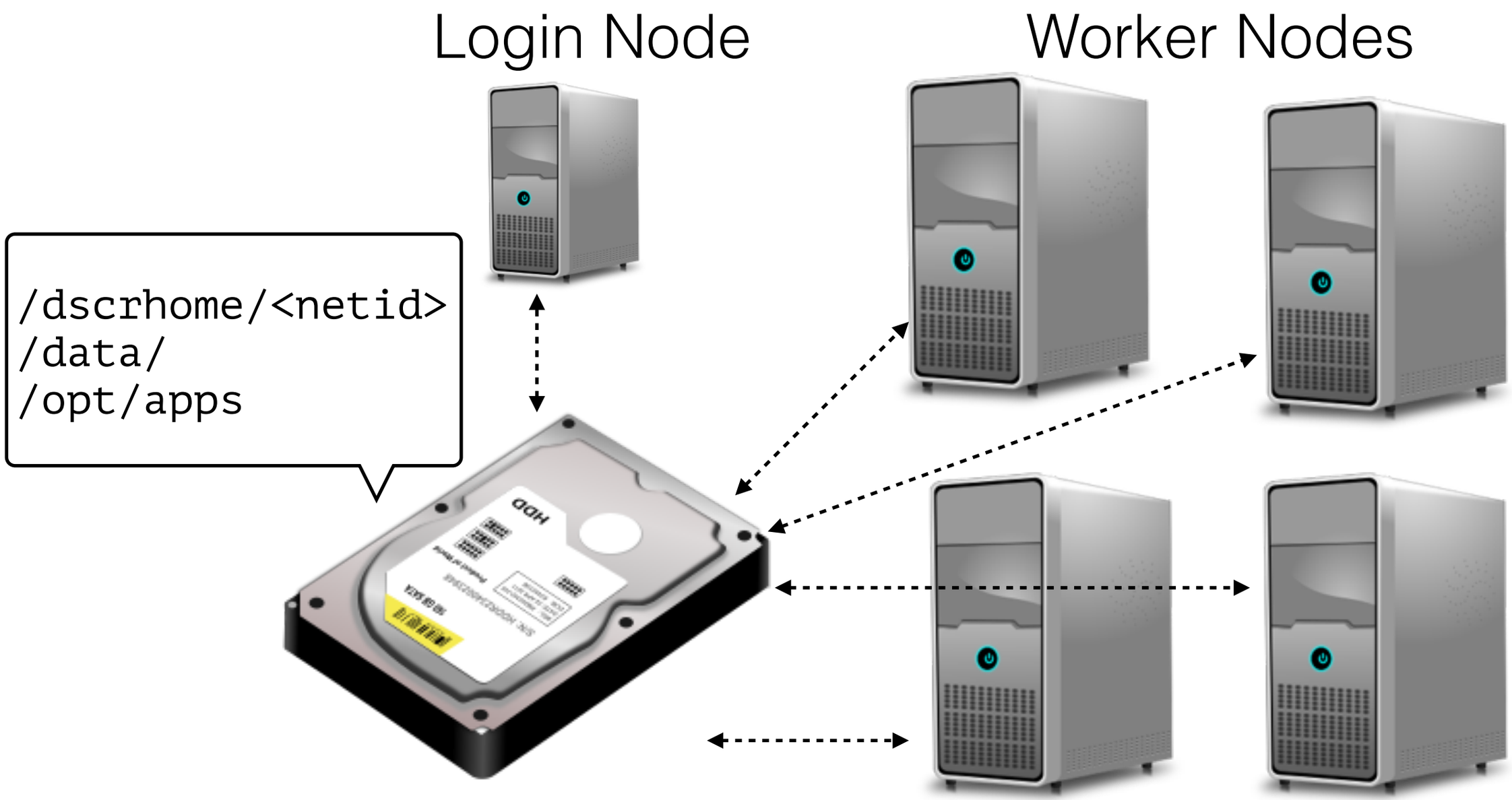
By specifying your software version this allows for reproducible science.

See currently loaded modules: `module list`

Request additional software or other HARDAC help

gcb-help@duke.edu

Our Cluster has Shared Folders



Delete your data when through - NOT long term storage

Advanced Job Types

These jobs require special sbatch flags and are rather complicated to setup but can provide faster results.

- Multithreaded programs
- MPI - programs written to work together on a cluster

Each application has it’s own particulars so searching for your particular application and Slurm is a good start.

Do not run jobs on the login node!

Login Node

your jobs run here **slowly** 😢

other users **can't start their jobs** 😢

Worker Nodes

Instead of here which would be **fast**