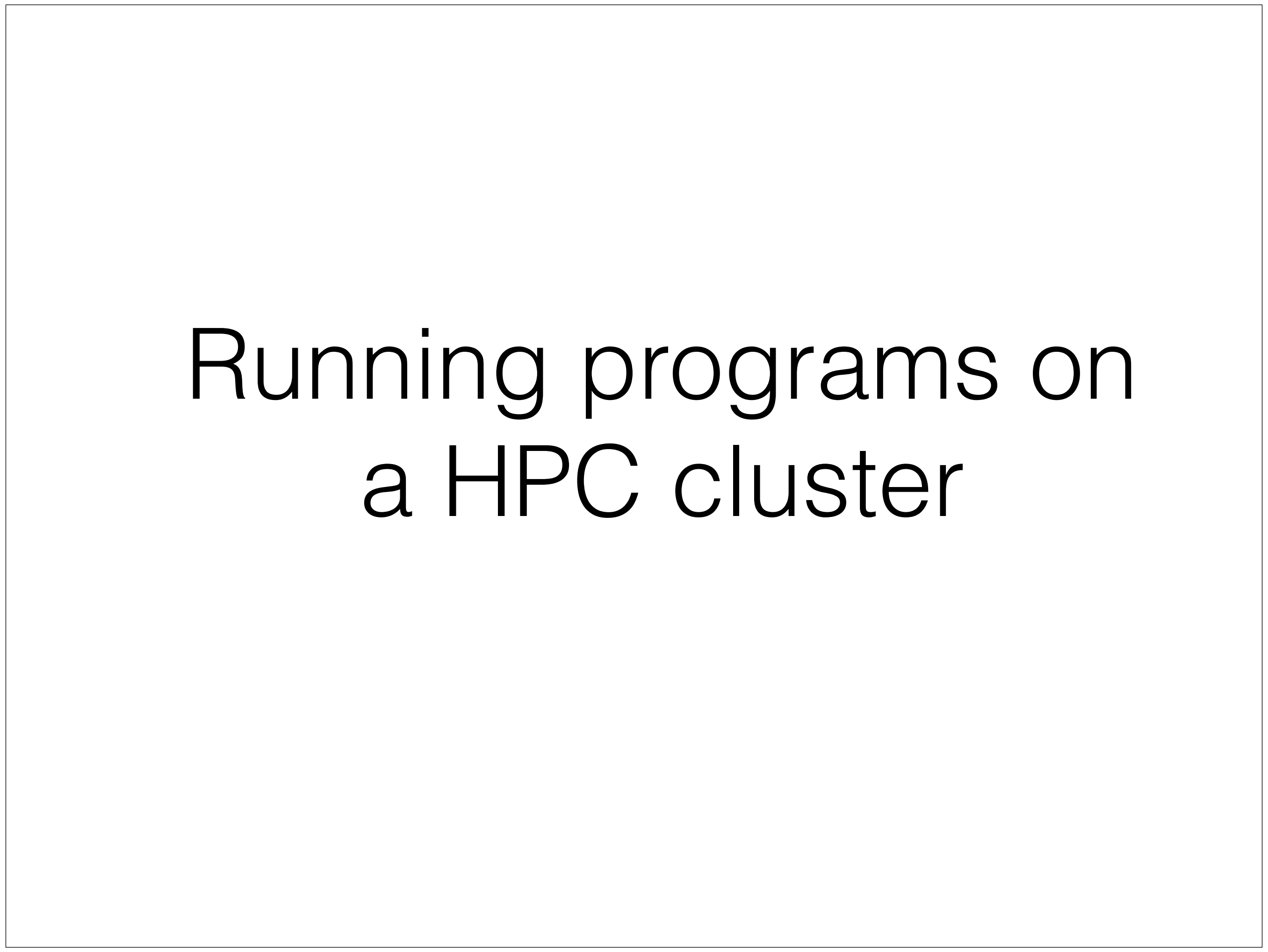


Running programs on a HPC cluster

High-performance computing

Goals of this Class

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

Laptop Not Powerful Enough

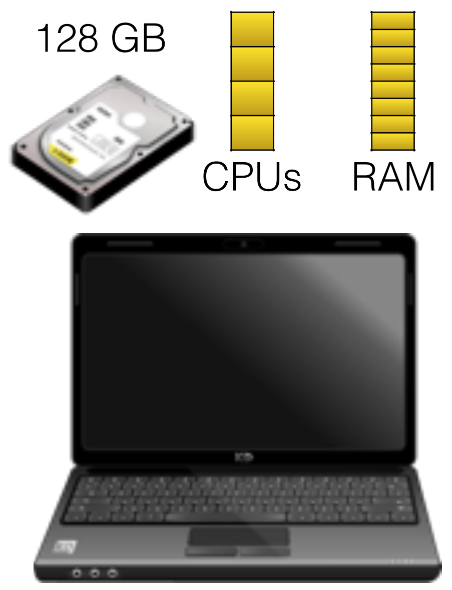
Symptoms

projects **TOO BIG** for your hard drive

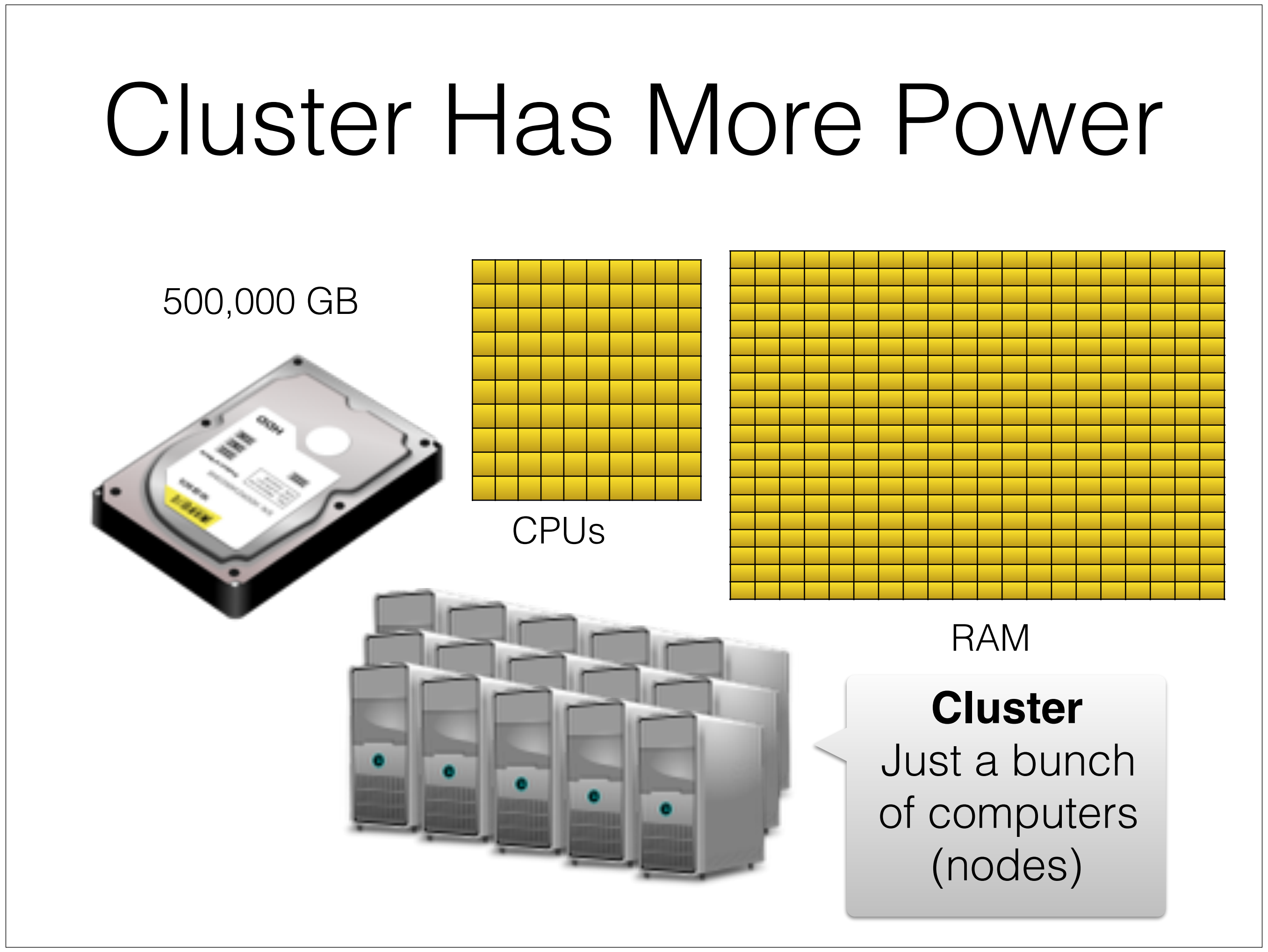
processing many files takes **FOREVER**

high RAM commands **CRASH**

The Problem



When running bioinformatics software on a laptop there are some challenges. Projects can be be too large to fit on your laptop. Running the same command on many files takes forever. Programs with high ram requirements will crash. For example Star an RNA-seq aligner requires 30G of ram.
The problem is you don't have enough resources.
Your hard drive is too small, number of CPUs too few, and there is not enough RAM.

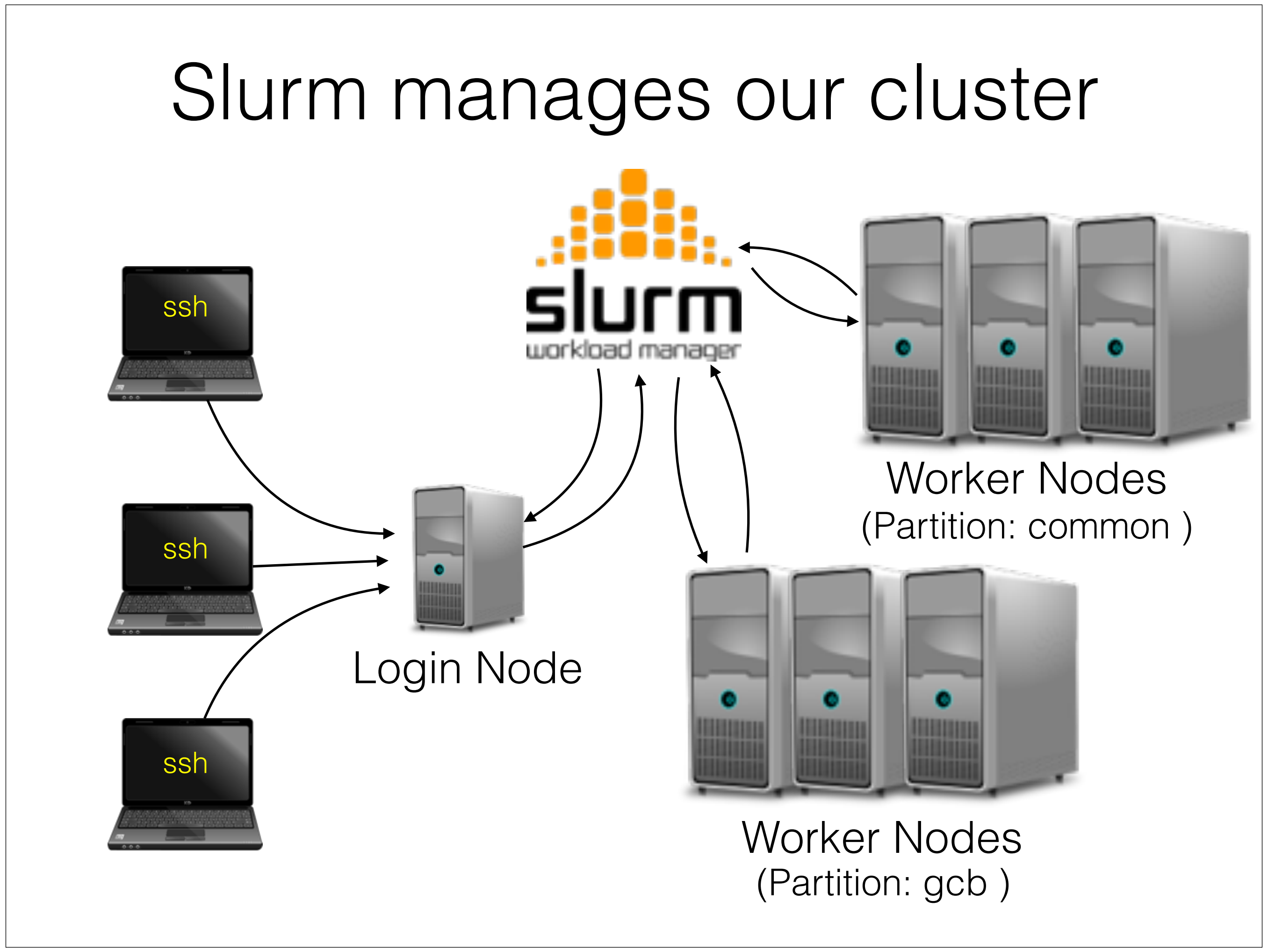


Cluster Has More Power

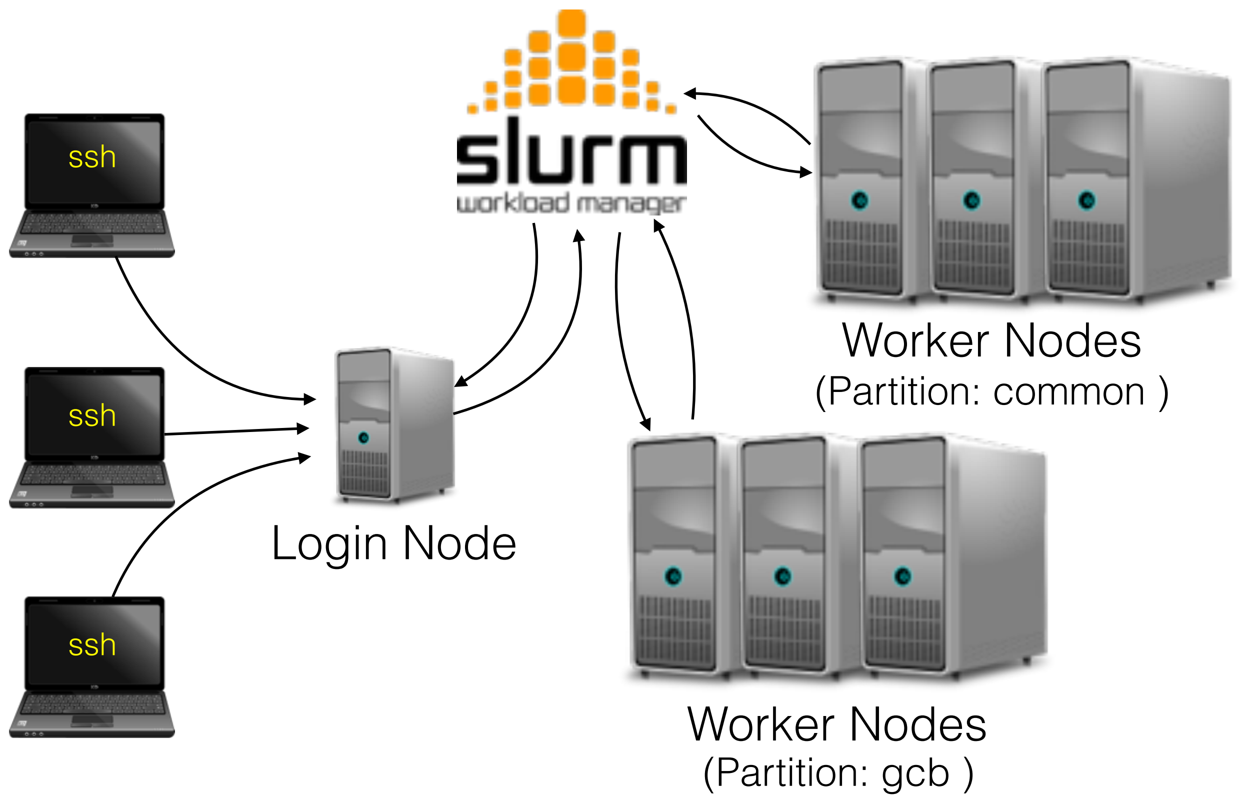
500,000 GB
CPUs
RAM
Cluster
Just a bunch of computers (nodes)

The cluster is just a bunch of more powerful computers.
So logically it has more of everything your laptop does.
We now have more space to store more files,
many more CPUs and more RAM.
When talking about clusters we refer each computer in the cluster as a node.

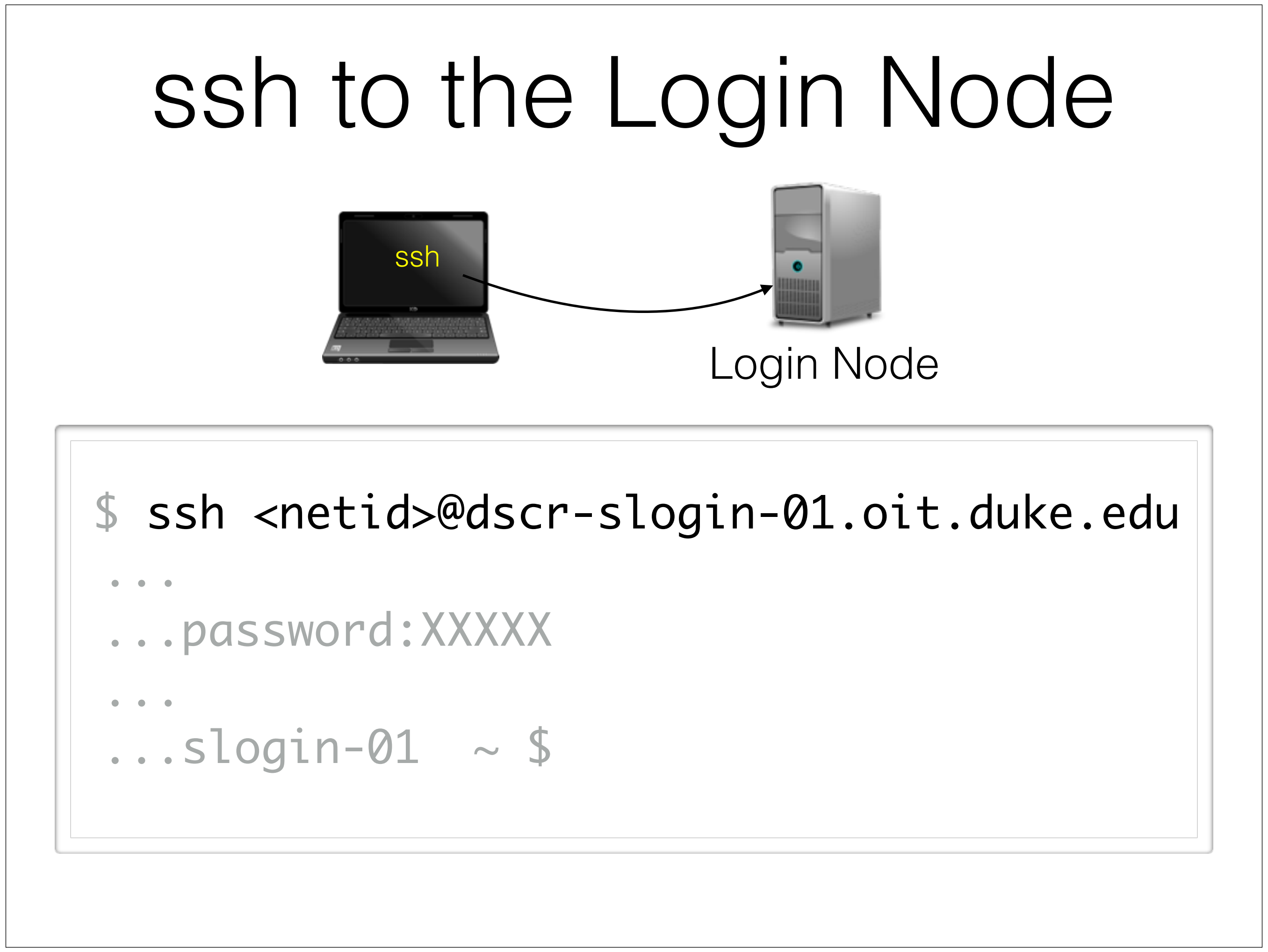
However a cluster brings in some additional challenges.
How do we control all these computers?
How do we share these computers with others?
We need a cluster manager.



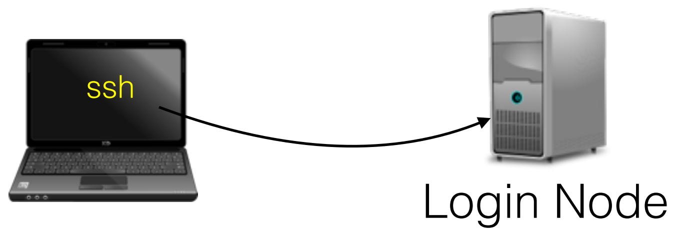
Slurm manages our cluster



Slurm is our cluster manager.
As you can see many users connect from their laptop to a shared Login Node over ssh.
Once on the login node they can ask Slurm to run jobs on the Worker Nodes.
The Login Node is a small computer just for the purpose of talking to Slurm.
All of your work should be done on the Worker nodes.
Worker nodes are grouped in partitions.
A partition is just a name you can use to refer to a group of nodes when running a job.
For this class we will be always using the default common partition.



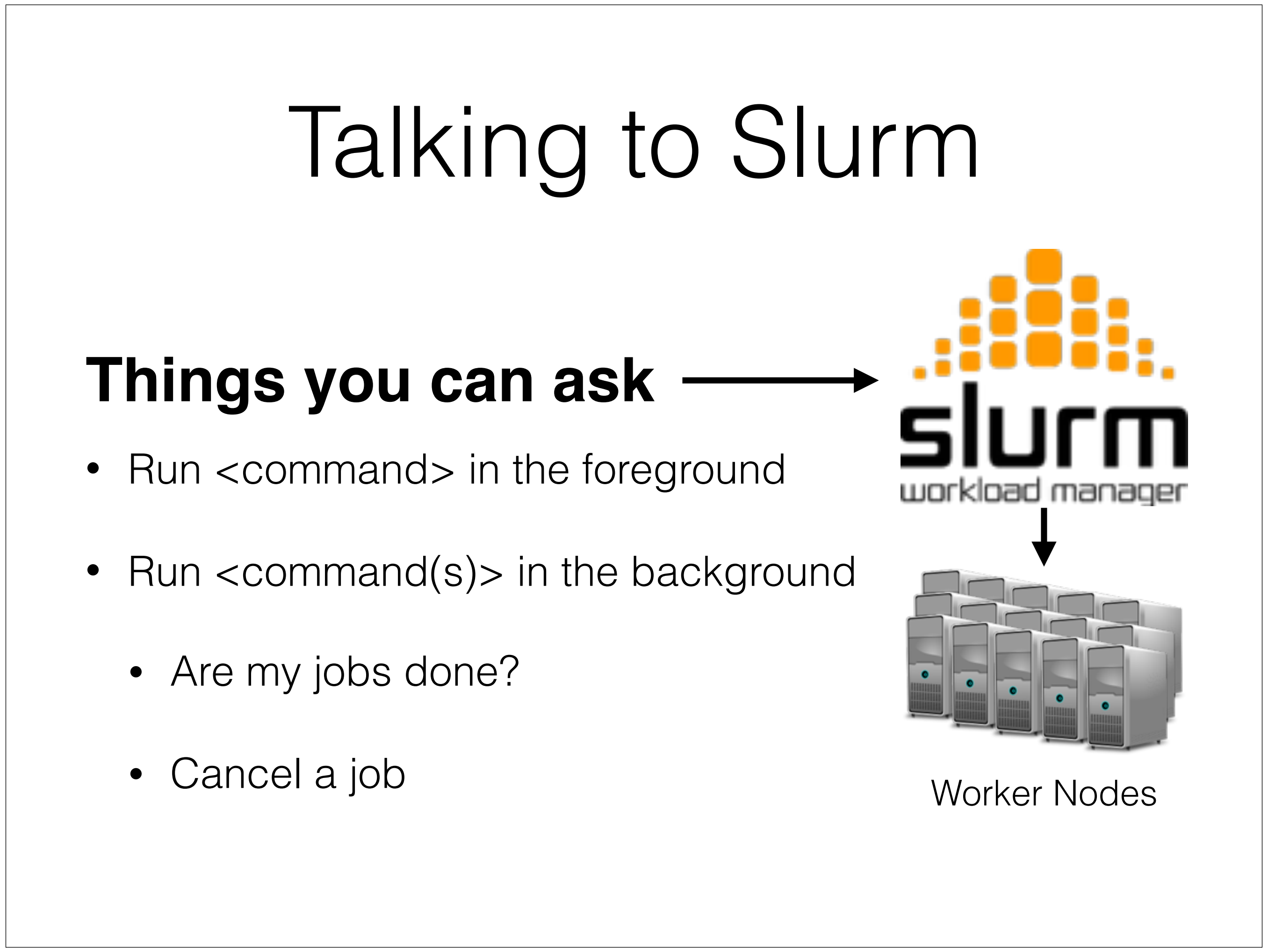
ssh to the Login Node



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

Now we are going to get onto the login node of the cluster.
Open your terminal or git bash application on your laptop. Enter the black ssh command into your terminal. Raise your green sticky note when this is done.
—wait for green notes

If so congrats you are now on the cluster!
When you first login you will be in your home directory so you can create and edit files in here.
Leave this terminal open for future steps.



Talking to Slurm

Things you can ask ⟶

- Run <command> in the foreground
- Run <command(s)> in the background
  - Are my jobs done?
  - Cancel a job

slurm
workload manager
Worker Nodes

Now that we are on the cluster we can talk to Slurm.
Everything you say to Slurm revolves around having Slurm run programs for you.
When you ask Slurm to run a program Slurm calls that a job.
You can ask Slurm to run a job for you either in the foreground or the background.

When Slurm runs a job in the foreground you will be unable to issue more commands until the job completes. For example run fastqc on some bam file and I'll sit and stare at my terminal until it finishes. If it errors or has output you will see printed to your terminal. If you close your terminal or your internet goes down these jobs will be stopped.

Running a job is run in the background is the standard approach for HPC. You can issue additional commands while it is working. You start jobs and can disconnect your laptop. You will need to check with Slurm about the status.

For example run fastqc on these 100 bam files, email me when it's done.
This requires us to come back later to ask Slurm if they are done.
This is where the real work gets done using Slurm.

Slurm Job Lifecycle

Active Job List - squeue

User	Cmd	State	Job ID
Bob	Star Aligner	Running	123

Historical Job List - sacct

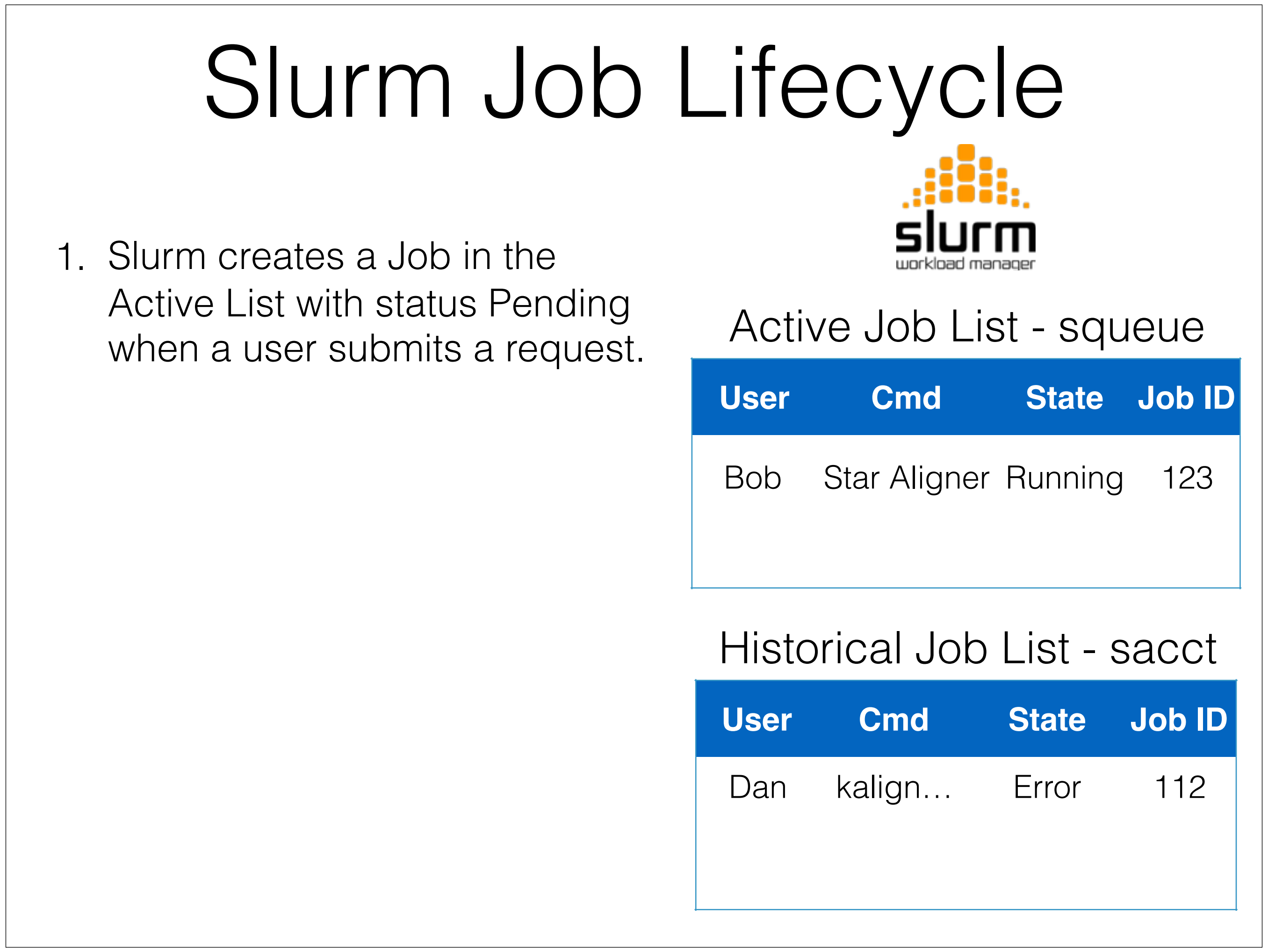
User	Cmd	State	Job ID
Dan	kalign…	Error	112

As slurm runs jobs for you they are moved between the Active and Historical Job Lists.
The short name for the Active Job List is squeue and the short name for the Historical Job List is sacct.

*When a user submits a job request slurm creates a job in the pending state under the Active Job List. Assigning a unique job id.

*Once there is enough free resources Slurm will start the job on Worker nodes and change it's state to Running.

*When the job completes either in error or success, Slurm moves the job to its Historical Job list. Where Slurm records the job's exit status, how much memory was used and other values.



Slurm Job Lifecycle

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

Active Job List - squeue

User	Cmd	State	Job ID
Bob	Star Aligner	Running	123

Historical Job List - sacct

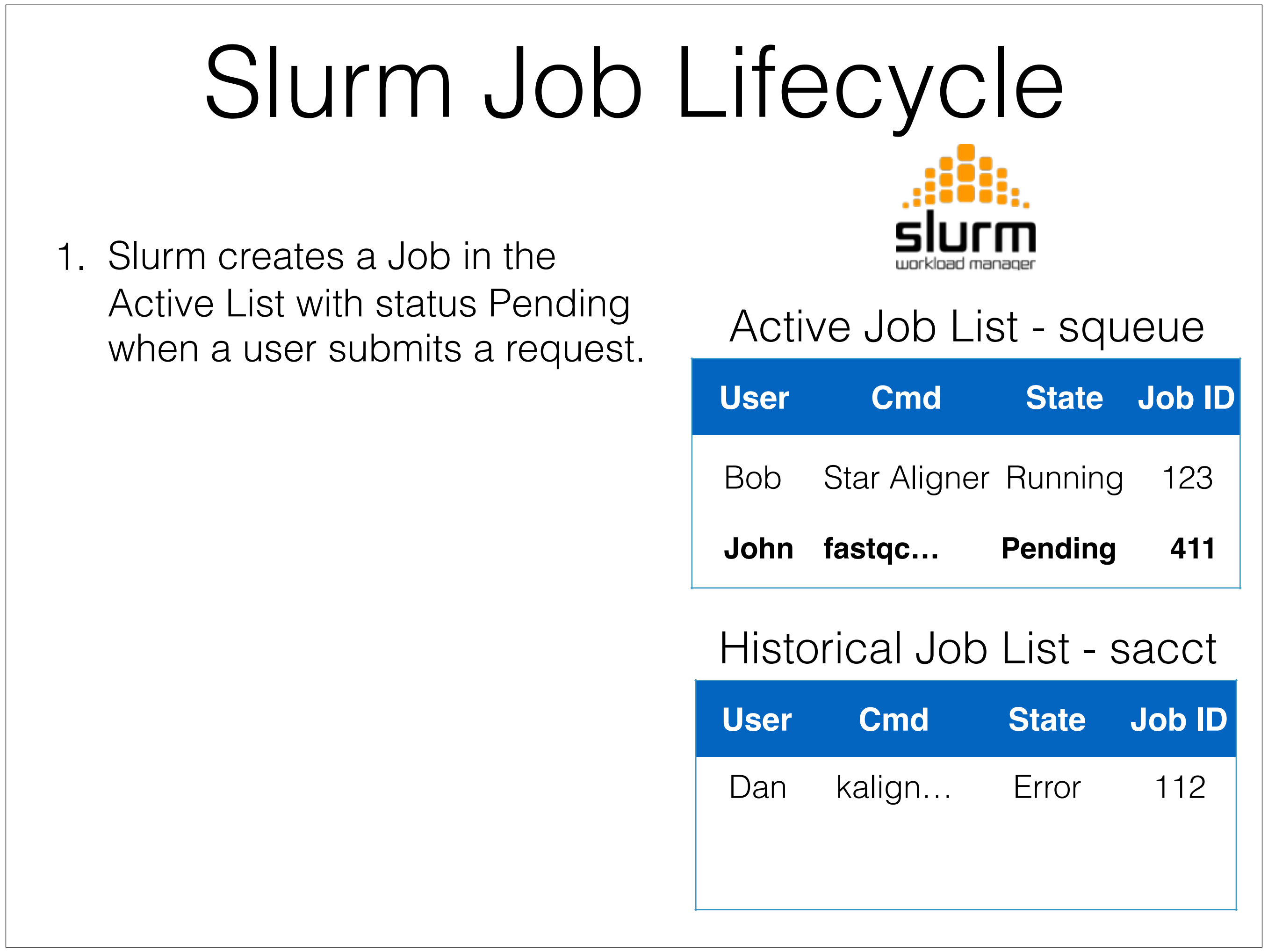
User	Cmd	State	Job ID
Dan	kalign…	Error	112

As slurm runs jobs for you they are moved between the Active and Historical Job Lists.
The short name for the Active Job List is squeue and the short name for the Historical Job List is sacct.

*When a user submits a job request slurm creates a job in the pending state under the Active Job List. Assigning a unique job id.

*Once there is enough free resources Slurm will start the job on Worker nodes and change it's state to Running.

*When the job completes either in error or success, Slurm moves the job to its Historical Job list. Where Slurm records the job's exit status, how much memory was used and other values.



Slurm Job Lifecycle

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

Active Job List - squeue

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

Historical Job List - sacct

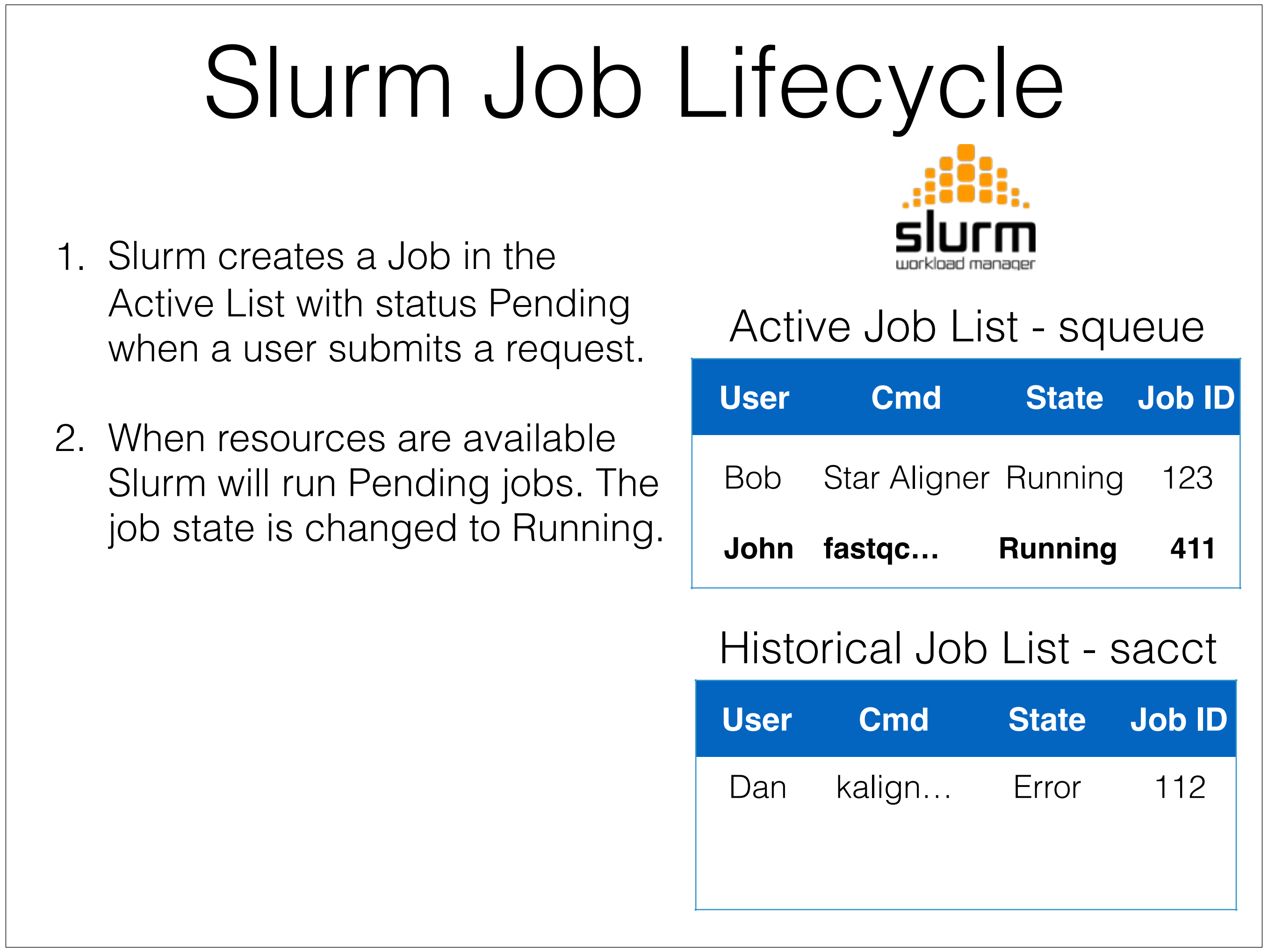
User	Cmd	State	Job ID
Dan	kalign…	Error	112

As slurm runs jobs for you they are moved between the Active and Historical Job Lists.
The short name for the Active Job List is squeue and the short name for the Historical Job List is sacct.

*When a user submits a job request slurm creates a job in the pending state under the Active Job List. Assigning a unique job id.

*Once there is enough free resources Slurm will start the job on Worker nodes and change it’s state to Running.

*When the job completes either in error or success, Slurm moves the job to its Historical Job list. Where Slurm records the job’s exit status, how much memory was used and other values.



Slurm Job Lifecycle

1. Slurm creates a Job in the Active List 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.

Active Job List - squeue

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

Historical Job List - sacct

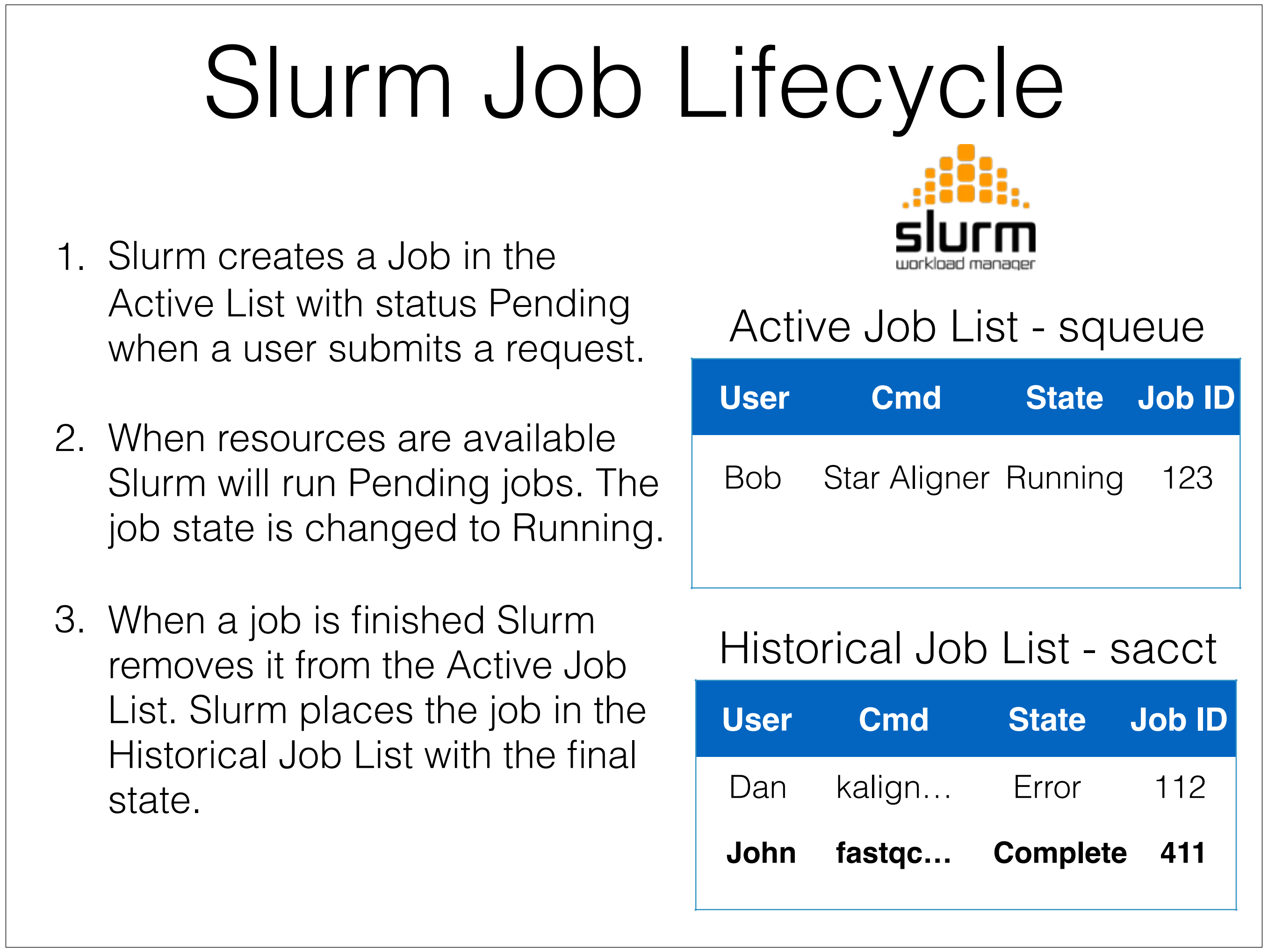
User	Cmd	State	Job ID
Dan	kalign…	Error	112

As slurm runs jobs for you they are moved between the Active and Historical Job Lists.
The short name for the Active Job List is squeue and the short name for the Historical Job List is sacct.

*When a user submits a job request slurm creates a job in the pending state under the Active Job List. Assigning a unique job id.

*Once there is enough free resources Slurm will start the job on Worker nodes and change it's state to Running.

*When the job completes either in error or success, Slurm moves the job to its Historical Job list. Where Slurm records the job's exit status, how much memory was used and other values.



Slurm Job Lifecycle

1. Slurm creates a Job in the Active List 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 Active Job List. Slurm places the job in the Historical Job List with the final state.

Active Job List - squeue

User	Cmd	State	Job ID
Bob	Star Aligner	Running	123

Historical Job List - sacct

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

As slurm runs jobs for you they are moved between the Active and Historical Job Lists.
The short name for the Active Job List is squeue and the short name for the Historical Job List is sacct.

*When a user submits a job request slurm creates a job in the pending state under the Active Job List. Assigning a unique job id.

*Once there is enough free resources Slurm will start the job on Worker nodes and change it’s state to Running.

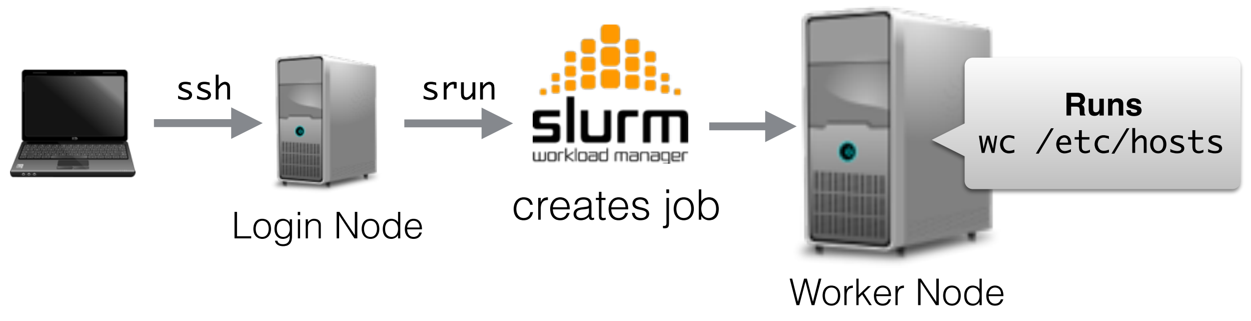
*When the job completes either in error or success, Slurm moves the job to its Historical Job list. Where Slurm records the job’s exit status, how much memory was used and other values.

srun

Slurm run a command in the foreground

Ask slurm to count the number of lines in a text file.

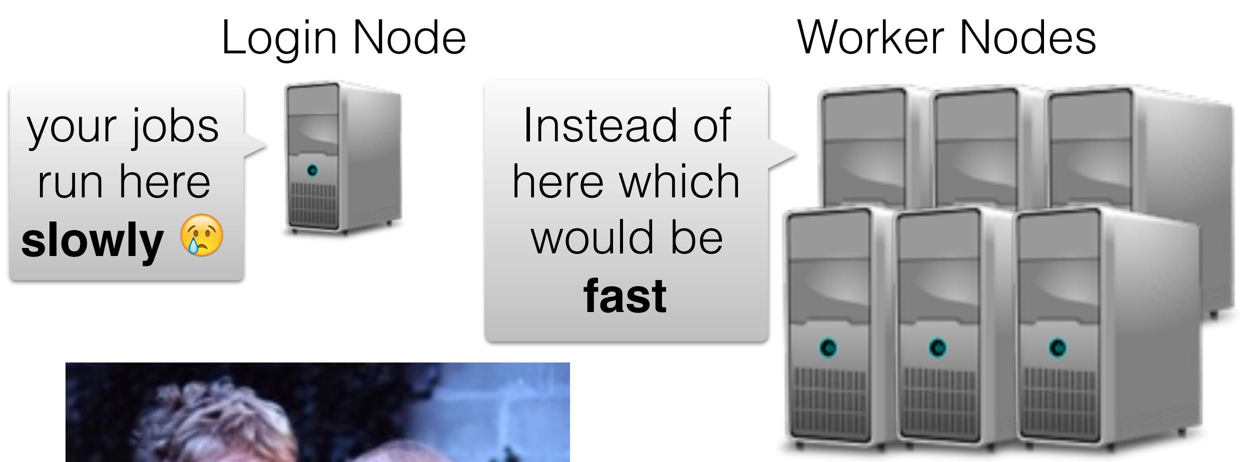
```
..slogin-01 $ srun wc /etc/hosts
srun: job 51 queued and waiting for resources
srun: job 51 has been allocated resources
2  10 158 /etc/hosts
..slogin-01 $
```



Now you are all going to finally run a command on the cluster in the foreground.
Enter the black srun command into your terminal. If you are not on the login node please login again. Raise your green sticky when done.
—wait for green sticky

From the login node and we issued an srun command.
The arguments to srun is the command and arguments we want Slurm to run.
In this case we want slurm to run wc and pass it /etc/hosts.
This command will count the number of lines, words, bytes in the /etc/hosts text file.
Slurm creates a job in pending, waits for one of the nodes to become available, tells a free node to run the command we passed to srun and prints the output back to us.
srun is only for short running jobs.
If you press ctrl-c or close your terminal it will cancel your srun job.
If you temporarily lose network connection your job gets canceled.

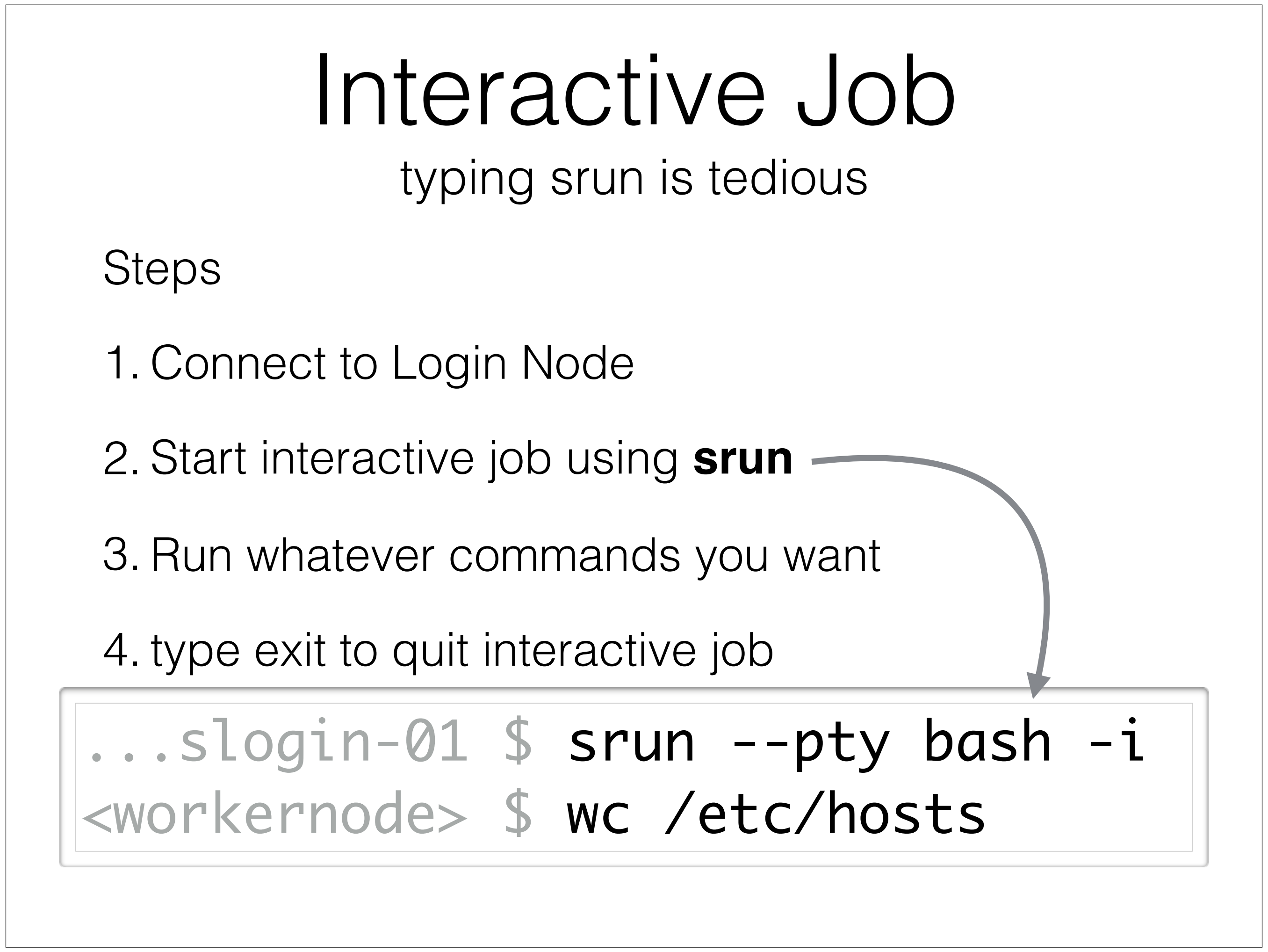
If You Forget **srun**


Login Node
Worker Nodes
your jobs run here **slowly** 😢
Instead of here which would be **fast**

Other users can't submit jobs

It turns out your command would have worked without the srun part.
But there are severe drawbacks to doing it this way.

1) It's gonna take much longer or might not be able to run at all
2) It's gonna hog the login node and make other users unable to run jobs.



Interactive Job

typing srun 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

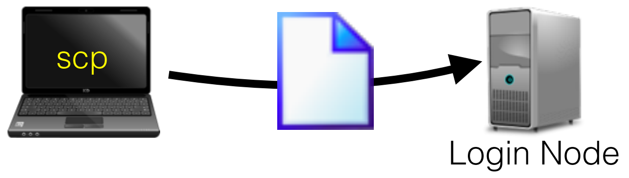
```
...slogin-01 $ srun --pty bash -i
<workernode> $ wc /etc/hosts
```

So we always use srun to run our jobs. Srun this srun that srun srun srun…
That get's old. So you can start an interactive job and just run commands without including srun every time.
Enter these commands in black and put the green sticky note up when it has worked for you.
—wait for green sticky

Keep in mind this will only using one node and one CPU with a small amount of memory.
This is almost always my first step when using the cluster and I recommend the same thing to you.

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
```

From a new terminal - not the one running an interactive job. Create a text file and enter the scp command. You will then be prompted for a password and it will copy the file onto the cluster.
Raise your green sticky when done.
—wait for green sticky

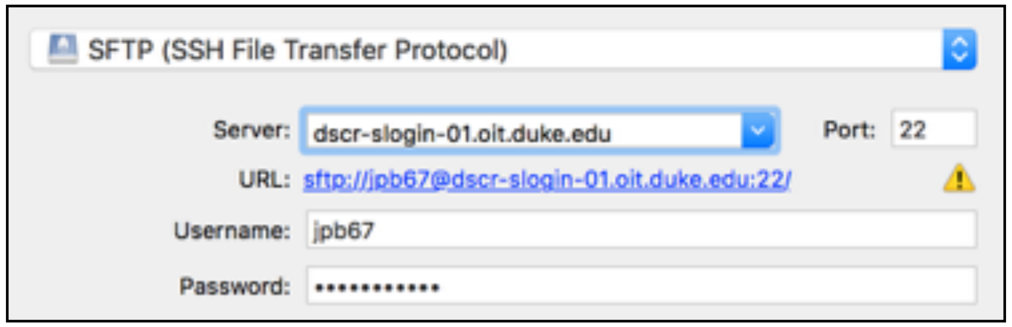
This is the scp command it performs a secure copy between your laptop to your home directory on the cluster. Explain parameters.
One important point is you can't use this command from on the cluster. You can only use it from a terminal on your laptop. If you wanted to copy a file from the cluster to your laptop you can reverse the order of the parameters to scp.

Here we are using the Login node to transfer files. Some clusters have a special node called a transfer node that is really fast at transferring files in and out. To my knowledge DCC doesn't have one of these but the HARDAC cluster does.

You can close this extra terminal or bash shell.

SFTP Client

Getting your files onto the Cluster



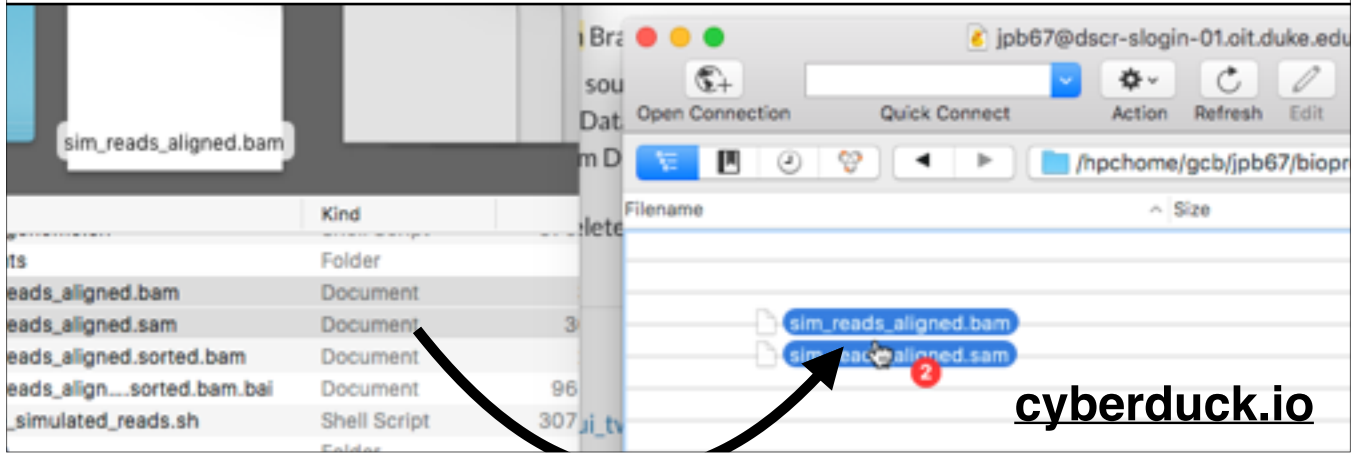
SFTP (SSH File Transfer Protocol)

Server: dscr-slogin-01.oit.duke.edu Port: 22

URL: sftp://jpb67@dscr-slogin-01.oit.duke.edu:22/

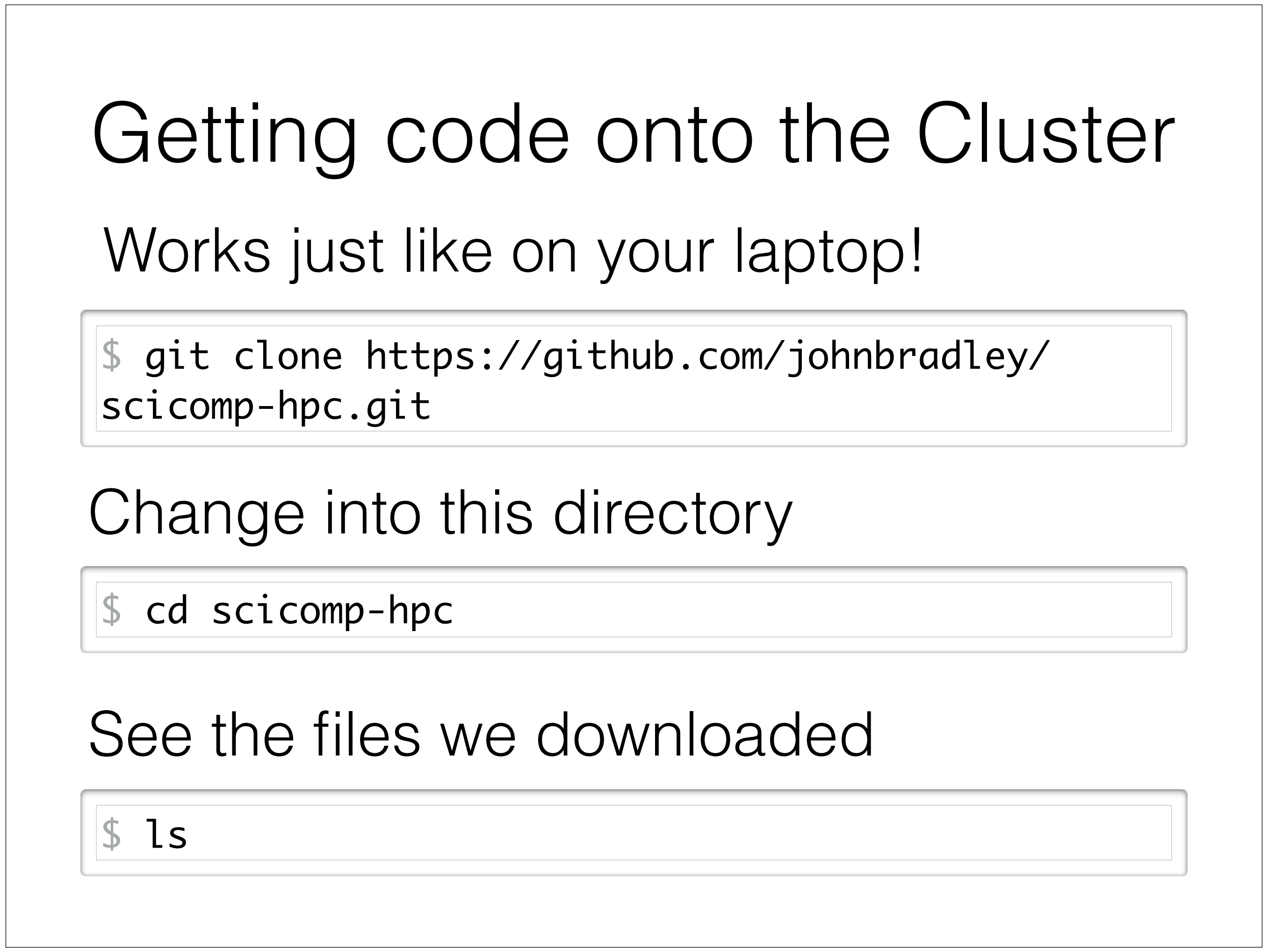
Username: jpb67

Password: ••••••••••••



cyberduck.io

Another option is using a SFTP client like CyberDuck.
This can allow you to drag and drop files onto the cluster once you have connected.
When setting up your connection make sure you choose the SFTP option and not plain FTP.
Use the same hostname that you used for ssh and scp.



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
```

Now clone a git repo into the cluster.
Type the black commands into your terminal and raise your green sticky when done.
—wait for green sticky

The first will download a git repository into your current directory. We are using the recursive option so we can checkout they python code from yesterday in addition to my scripts.
This is the same command you would use on your laptop. Then we change into the directory it created so we can use these scripts. Finally run the ls command to see the contents of this folder.

sbatch

Run command(s) in the background

Make a file called countgc.sh using nano:

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

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

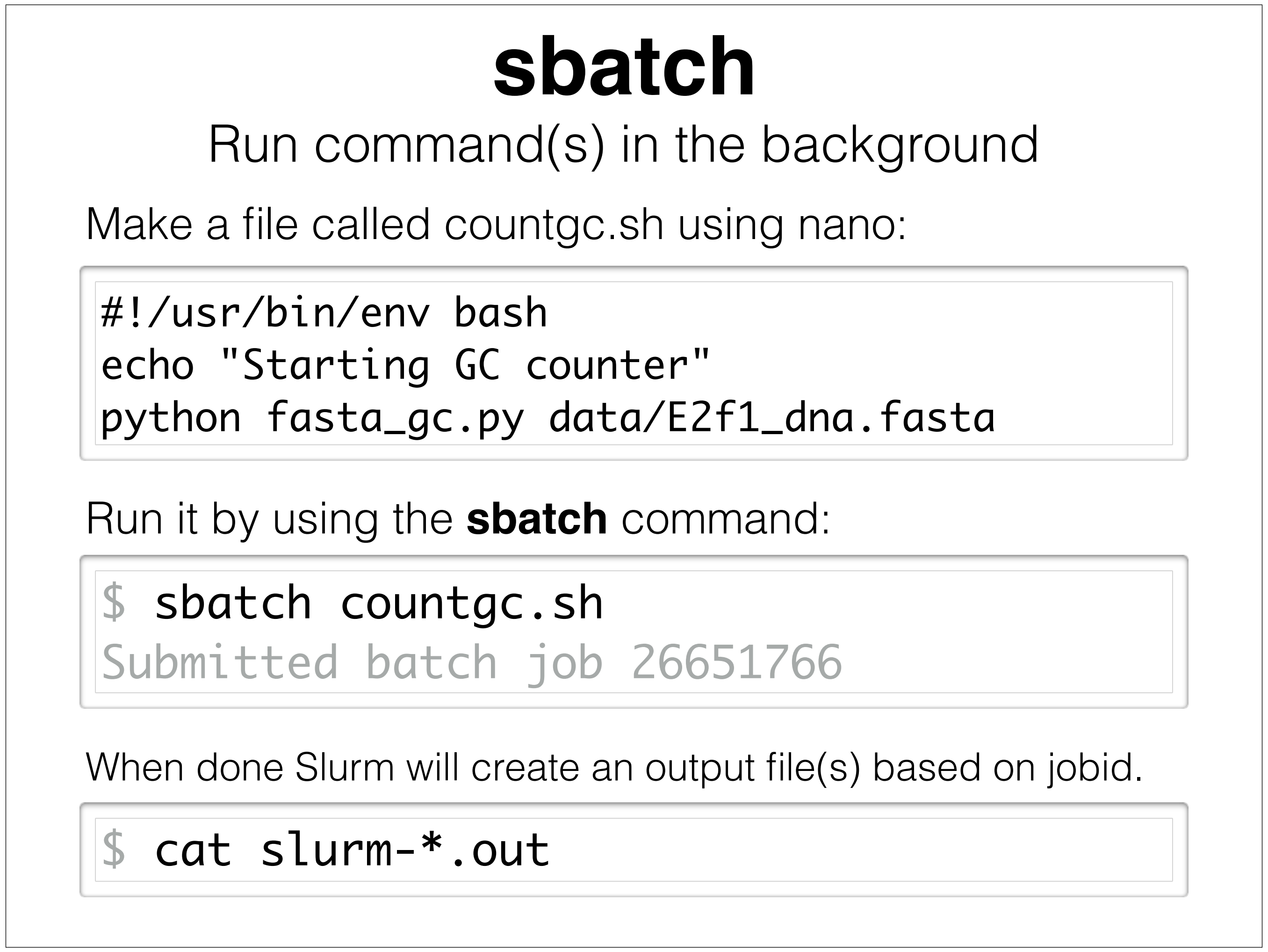
```
$ sbatch countgc.sh
Submitted batch job 26651766
```

Now we will run a python script from yesterday in the background.
fasta_gc.py and the data file are in the code we just checked out.
Type the text in the top box into a shell script named countgc.sh.
Then run it using sbatch as shown.
Put the green sticky note up when you see a job id printed to your terminal.
—wait for green sticky

sbatch is the most important command for talking to Slurm.
It is the command you should use for long running jobs. It will not terminate if you close your terminal or shutdown your laptop. This command lets you use multiple CPUs more Memory and really make heavy use of the cluster.

Using batch is a three step process:
First create a shell script.
Second ask sbatch to run that script.
Finally wait for your job to finish.

You must include the shebang line(that weird #! thing at the top) from yesterday’s talk in your shell script or sbatch will not run your script.
The -p option tells slurm which partition we will be using.
If you leave this off the jobs will be put in the common pool which is less responsive.



sbatch

Run command(s) in the background

Make a file called countgc.sh using nano:

```
#!/usr/bin/env 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
```

Now we will run a python script from yesterday in the background.
fasta_gc.py and the data file are in the code we just checked out.
Type the text in the top box into a shell script named countgc.sh.
Then run it using sbatch as shown.
Put the green sticky note up when you see a job id printed to your terminal.
—wait for green sticky

sbatch is the most important command for talking to Slurm.
It is the command you should use for long running jobs. It will not terminate if you close your terminal or shutdown your laptop. This command lets you use multiple CPUs more Memory and really make heavy use of the cluster.

Using batch is a three step process:
First create a shell script.
Second ask sbatch to run that script.
Finally wait for your job to finish.

You must include the shebang line(that weird #! thing at the top) from yesterday's talk in your shell script or sbatch will not run your script.
The -p option tells slurm which partition we will be using.
If you leave this off the jobs will be put in the common pool which is less responsive.

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

Type in the squeue command to see the status of your active jobs.
Raise your sticky note when you have run this command.
—wait for green sticky

Remember once they finish they will no longer display here.
You may not have any jobs currently running.
I have a shell script that will take a long time to run.
Start it via sbatch long_running.sh.
Then run the above command again and you should see your job in the list.
For the rest of the class you can use squeue to check if your job has finished.

Raise your green sticky when you are able to see the long running job in the active job list for your netid.
—wait for green sticky

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

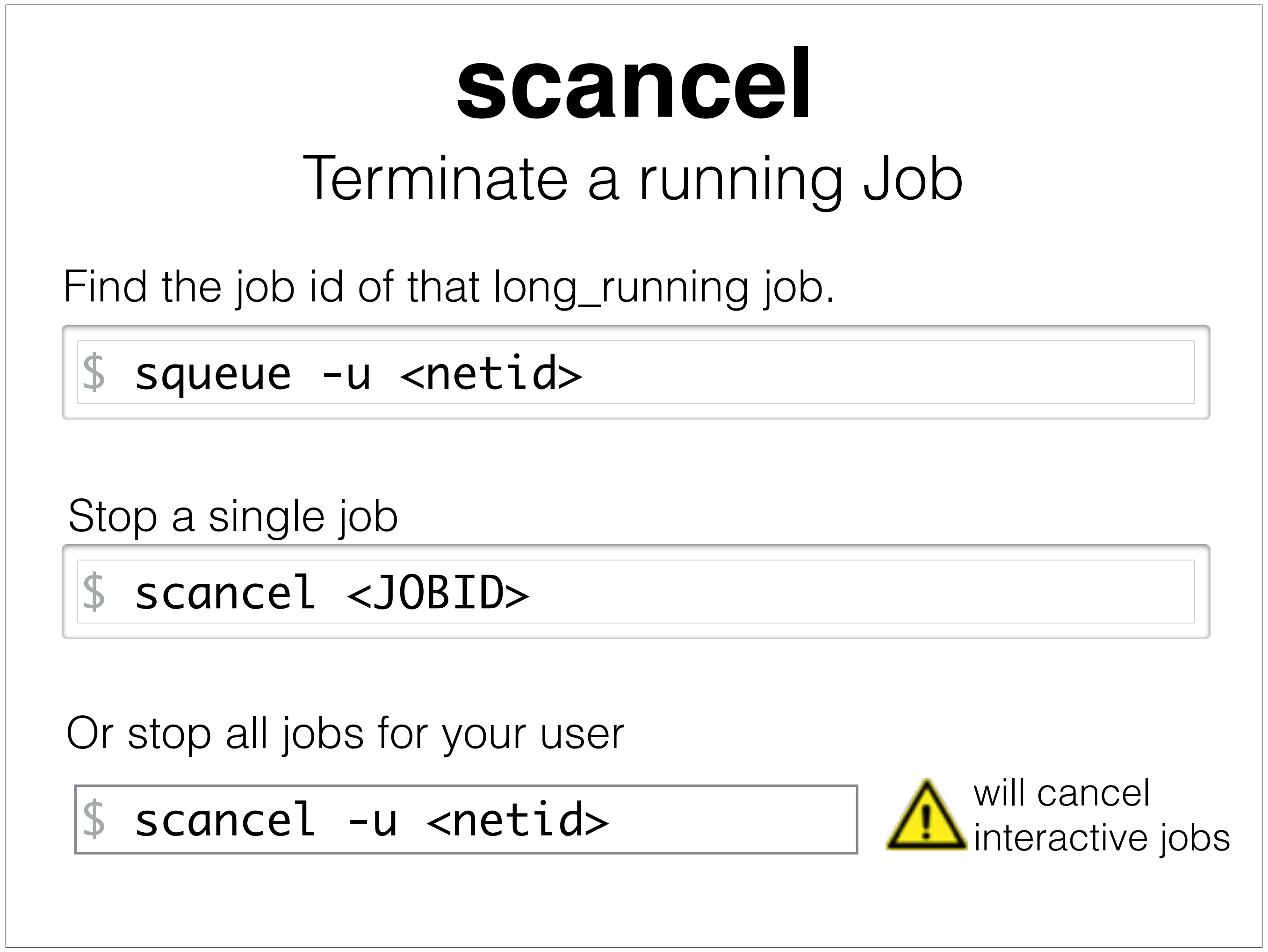
Start a long running job the repeat the above command.

```
$ sbatch long_running.sh
```

Type in the squeue command to see the status of your active jobs.
Raise your sticky note when you have run this command.
—wait for green sticky

Remember once they finish they will no longer display here.
You may not have any jobs currently running.
I have a shell script that will take a long time to run.
Start it via sbatch long_running.sh.
Then run the above command again and you should see your job in the list.
For the rest of the class you can use squeue to check if your job has finished.

Raise your green sticky when you are able to see the long running job in the active job list for your netid.
—wait for green sticky



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

Enter the squeue command to find the job id for your long running job.
Then cancel it via scancel.
Raise a sticky note when done.
—wait for green sticky

With sbatch you must the use the scancel command to terminate a job.
Perhaps you realize you entered the wrong parameters or the output so far looks wrong.
The most precise way is to cancel a single job by passing the jobid.
So we looked up the job id of of your long_running job with squeue. Then cancel it via scancel.

Another approach is just to cancel all jobs for your user - keep in mind this will end your interactive job as well. It is a very nice option when you have accidentally scheduled a bunch of jobs.
You do not need to run the bottom item.

sbatch

memory requirements

Change countgc.sh using nano:

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

$\longrightarrow$ **400MB RAM**

Not enough RAM

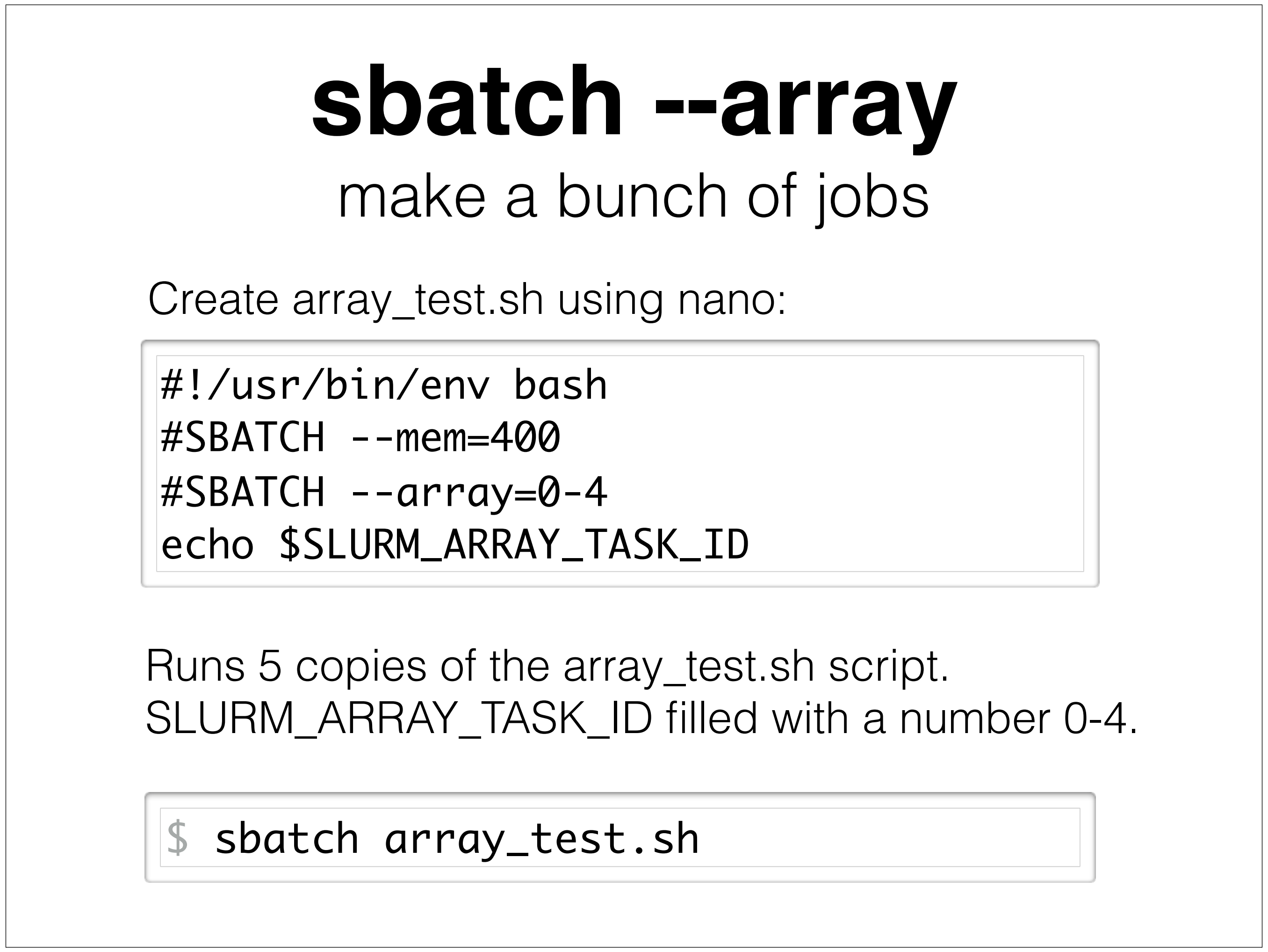
```
...
srun: error: … Killed
srun: Exceeded job memory limit
```

Too much RAM

(other users)

Change countgc.sh adding this new line. Run countgc.sh with sbatch to make sure your syntax is correct.
Raise a sticky note when done.
—wait for green sticky

By adding comments that begin with SBATCH you change how your script is run by sbatch.
In this case we are adding the mem flag to specify the amount of memory we will need.
In this case we are requesting 400MB of RAM.
If the program uses more than this Slurm will terminate the process.
On the other hand if you request a giant amount of memory it may take longer to schedule.
You want to be considerate of other users not requesting more than necessary.
In practice it is really hard to guess how much RAM you will need.
In a bit we can check how much RAM was used via the historical job list - sacct.



sbatch --array

make a bunch of jobs

Create array_test.sh using nano:

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

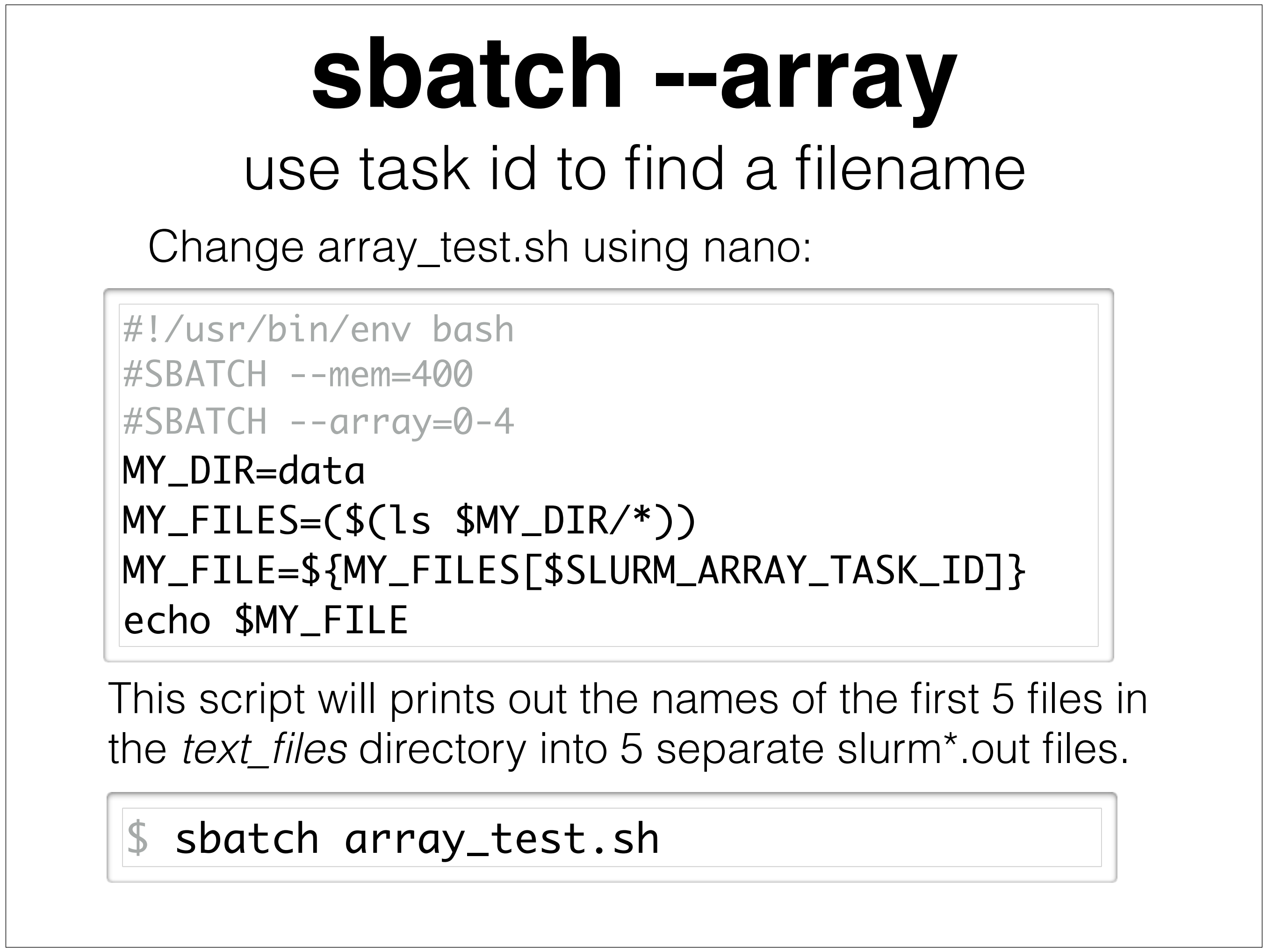
Runs 5 copies of the array_test.sh script.
SLURM_ARRAY_TASK_ID filled with a number 0-4.

```
$ sbatch array_test.sh
```

When you have a bunch of files you need to run the same command on this is the tool to use.
This is the only responsible way to run a bunch of similar jobs.
So create array_test.sh with the content seen above and run it with sbatch.
Raise your green sticky when done.
—wait for green sticky

With this option enabled Slurm will create multiple jobs to run your shell script.
Each job will have a unique JOBID, but they will all share a a common JOBID prefix.
So that each job can do something different sbatch fills in the environment variable SLURM_ARRAY_TASK_ID for each job instance. This allows them to process different data. Your shell script can use this value to determine which file, url or other data to perform it's work on.

Note: This isn't a real world example we are just trying to see how the SLURM_ARRAY_TASK_ID environment variable gets filled in for each job in our array.



sbatch --array

use task id to find a filename

Change array_test.sh using nano:

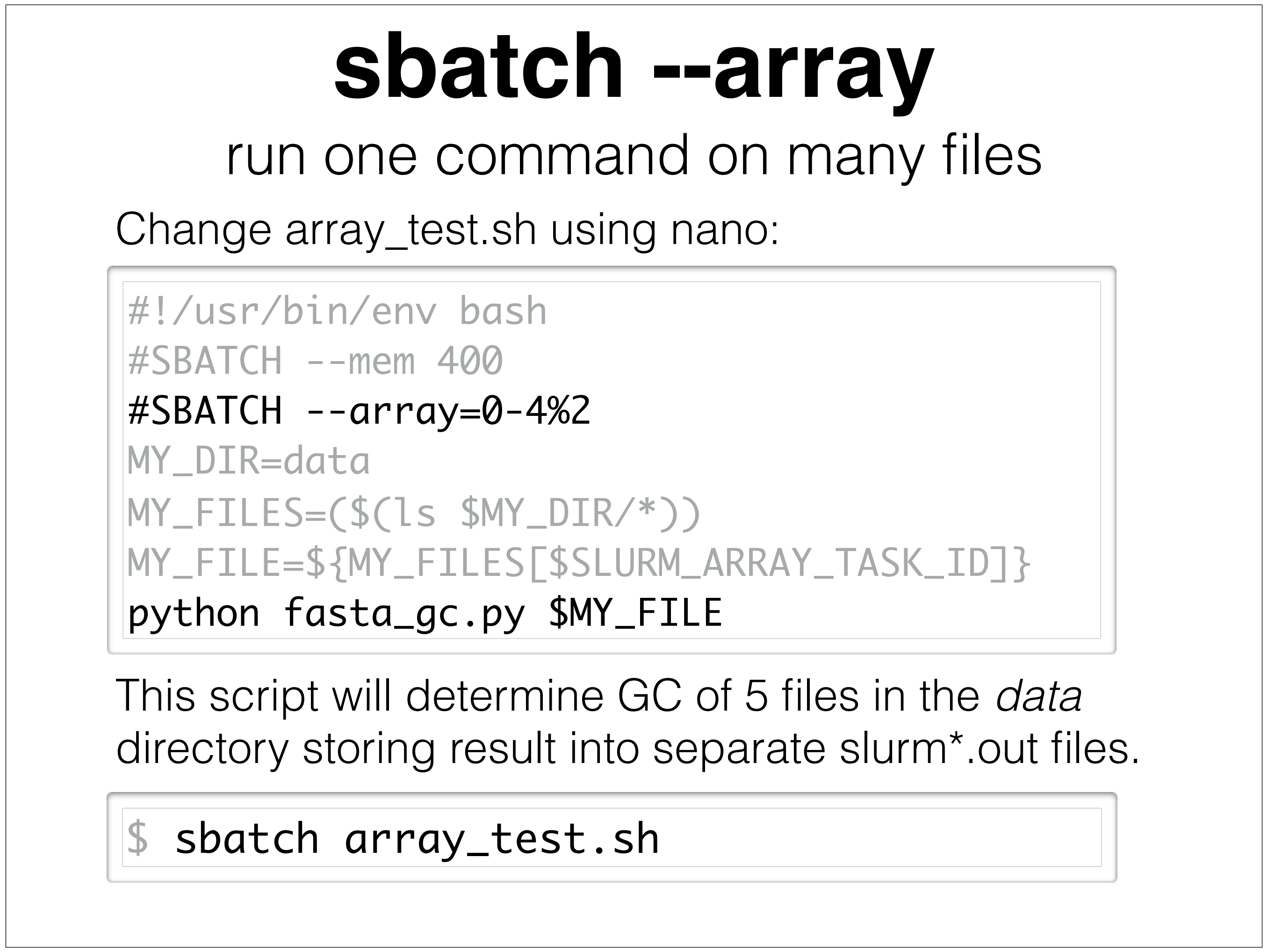
```
#!/usr/bin/env bash
#SBATCH --mem=400
#SBATCH --array=0-4
MY_DIR=data
MY_FILES=($(ls $MY_DIR/*))
MY_FILE=${MY_FILES[$SLURM_ARRAY_TASK_ID]}
echo $MY_FILE
```

This script will prints out the names of the first 5 files in the *text_files* directory into 5 separate slurm*.out files.

```
$ sbatch array_test.sh
```

Now change array_test.sh to list the contents of the text_files directory into an array.
Then run sbatch again.
—wait for green sticky

Here we create a variable called MY_DIR which is the directory we want to process files from.
Then we create MY_FILES, a bash array holding the names of the files in MY_DIR.
Then we pick out a single file based on the array job task id.
Finally we print that filename out which is put into the slurm output file.



sbatch --array

run one command on many files

Change array_test.sh using nano:

```
#!/usr/bin/env bash
#SBATCH --mem 400
#SBATCH --array=0-4%2
MY_DIR=data
MY_FILES=($(ls $MY_DIR/*))
MY_FILE=${MY_FILES[$SLURM_ARRAY_TASK_ID]}
python fasta_gc.py $MY_FILE
```

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

```
$ sbatch array_test.sh
```

Now change array_test.sh to run a command on each file and run sbatch again.
When this is finished you should have several slurm*.out files with filenames in them.
Raise your green sticky when this is done.
—wait for green sticky

In this step we added the %2 option. This tells Slurm to only run two at a time.
When running large array jobs this will keep your job from overwhelming the cluster by trying to run them all at once.
We also called the python program from the previous step making this now a functional array job script.

This script contains some difficult to read code but can be re-used for running different commands on multiple files. Remember that the values in the array setting determine how many files we will process so you will need to count the number of files in the directory to perform your task on all of them.

sbatch

email when job completes

Create email_wc.sh using nano:

```
#!/usr/bin/env bash
#SBATCH --mail-type=END
#SBATCH --mail-user=<your_email_address>
wc looking-glass.txt
```

Run it with sbatch

```
$ sbatch email_wc.sh
```

You can configure sbatch to email you when the job is done.
create email_wc.sh filling in your email address and run sbatch on it.
Once sbatch prints out your job id raise your green sticky.
No need to wait for the email just remember it should be arriving in a few minutes.
—wait for green sticky

sacct

historical job status

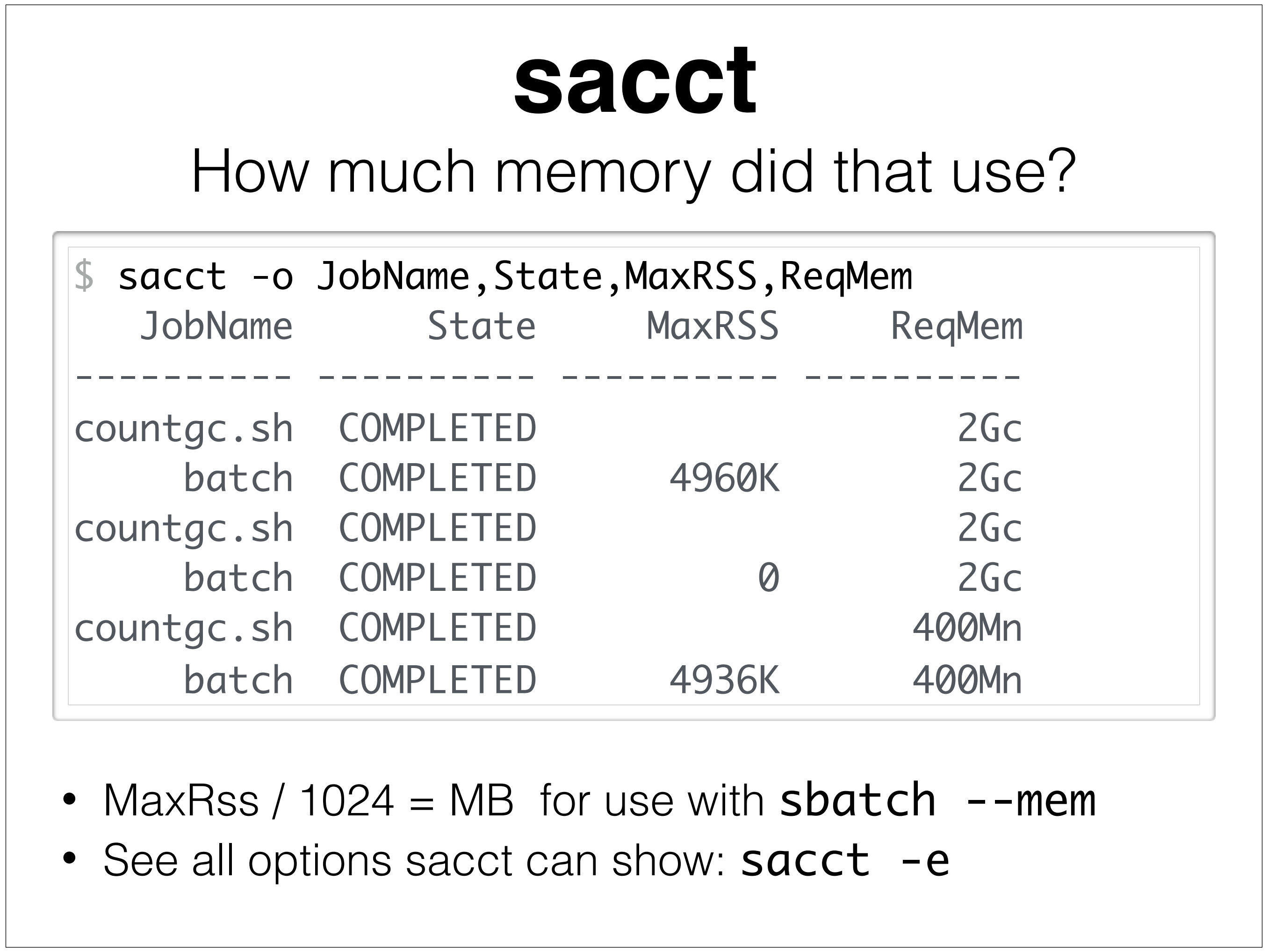
```
$ sacct
       JobID    JobName  ...      State ExitCode
------------ ---------- ... ---------- --------
26705496     countgc.sh ...   COMPLETED      0:0
26705496.ba+      batch ...   COMPLETED      0:0
26705566     countgc.sh ...   FAILED         1:0
26705566.ba+      batch ...   FAILED        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

Now let's look at Slurm's historical job list.
Enter the sacct command in your terminal to see jobs that finished.
Raise your green sticky once you are finished looking at your historical jobs.
—wait for green sticky

There is a big gotcha that will bite you with sacct.
It only display jobs that finished today.
The starttime flag will let you control this.

There are many more columns sacct can display.



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
```

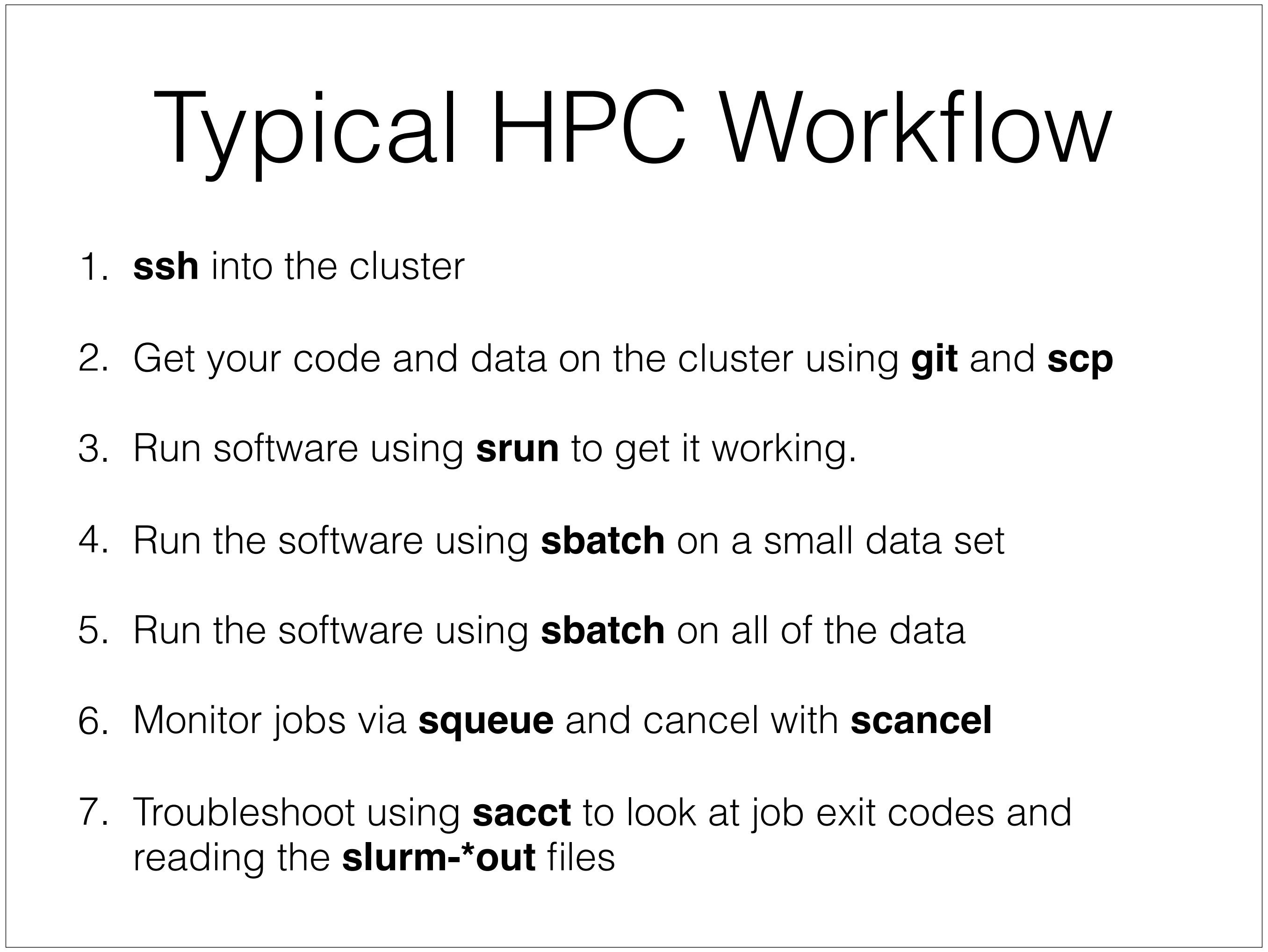
- MaxRss / 1024 = MB for use with `sbatch --mem`
- See all options sacct can show: `sacct -e`

Run the black sacct command specifying which columns to show. Make sure you don't put spaces between the comma separated list.
Raise your green sticky when you this works.
—wait for green sticky

The -o argument gives sacct a list of fields we want printed out for each job.
The two new ones are MaxRSS and ReqMem.

MaxRSS tells us how much ram we used. This can help us better calculate a value for the mem flag used with sbatch. Only it doesn't always work. This value is based on process memory when slurm checked on your job. Short running jobs may not have a value because the finished before slurm could check on them.

ReqMem is the amount of requested memory.
So in the example here we requested 400MB for some and 2G for others.
The lowercase n and c are where this allocation is per cpu or per node.



Typical HPC Workflow

1. **ssh** into the cluster
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

Now we are going to go over how all we have covered fits together.
First you connect to the cluster using ssh.
Then you can checkout your code using git and copy files onto the server using scp or a SFTP desktop program like Cyberduck.
Next using srun to get your software working with a tiny data set. At this point you just want to make sure it doesn't just error out immediately.
Then move to sbatch creating a shells script and running your software against a small data set.
Once that is successful scale up to your large data set.
Large jobs can take hours or days depending on what you are running.
While it is running you can monitor it with squeue and possibly terminate with scancel.
Once your jobs finish you can use sacct to look at their exit codes.
Also remember to check your slurm output files.

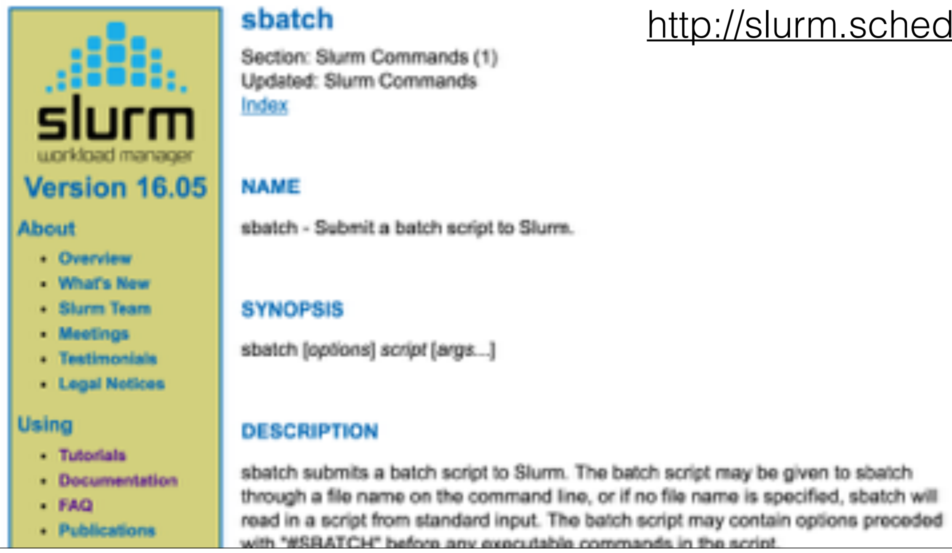
Help for Slurm commands

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.

Here are two ways to get help for your slurm commands.
Raise your green sticky once you have run sbatch -h command.
—wait for green sticky

The official online documentation shown here is just a prettier version of what the command line programs. Searching for slurm tutorials is also another option. Several universities have some good tutorials online, but keep in mind there will be differences between their clusters and ours.

I've heard that there is a way to defer running a job until another job is finished.
Look through the output of -h and put up a sticky when you found the flag that will cause job to wait for another job to finish.

Two Clusters

DCC

General Purpose

HARDAC

Bioinformatics
Focused

So there are two clusters that I know of that can be used for bioinformatics.
First is the DCC which we have been using for this class.
DCC is a general purpose cluster.
The other is HARDAC which is run by GCB.
This is a bioinformatics cluster.
Ask your PI which cluster your group uses.

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

In the future DCC will support *modules* which provides command line tools for loading many different software packages.

You can also reach out to your Point of Contact:

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

Finding Software - HARDAC

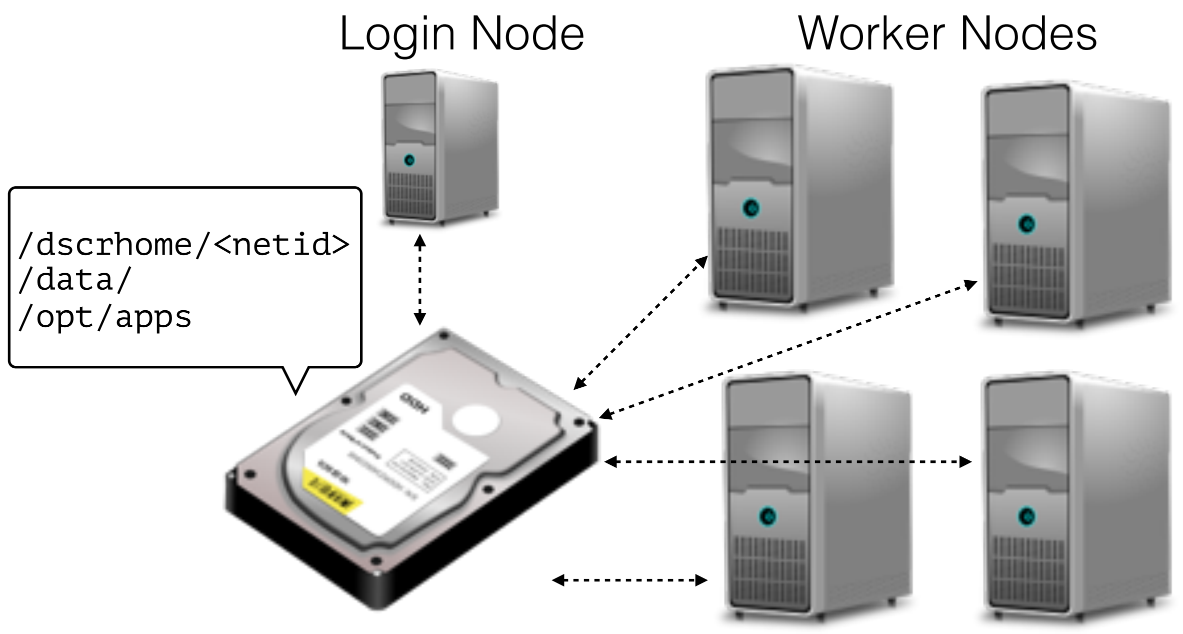
Load many bioinformatics modules.

```
module avail
module load <modulename>
... use module
```

Request additional software be added to modules:

gcb-help@duke.edu

Our Cluster has Shared Folders



Delete your data when through - NOT long term storage

We have some shared folders on our cluster.

- save files into your home directory
- not all directories are shared - if you save to /tmp you have a while goose chase
- this is not permanent 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.