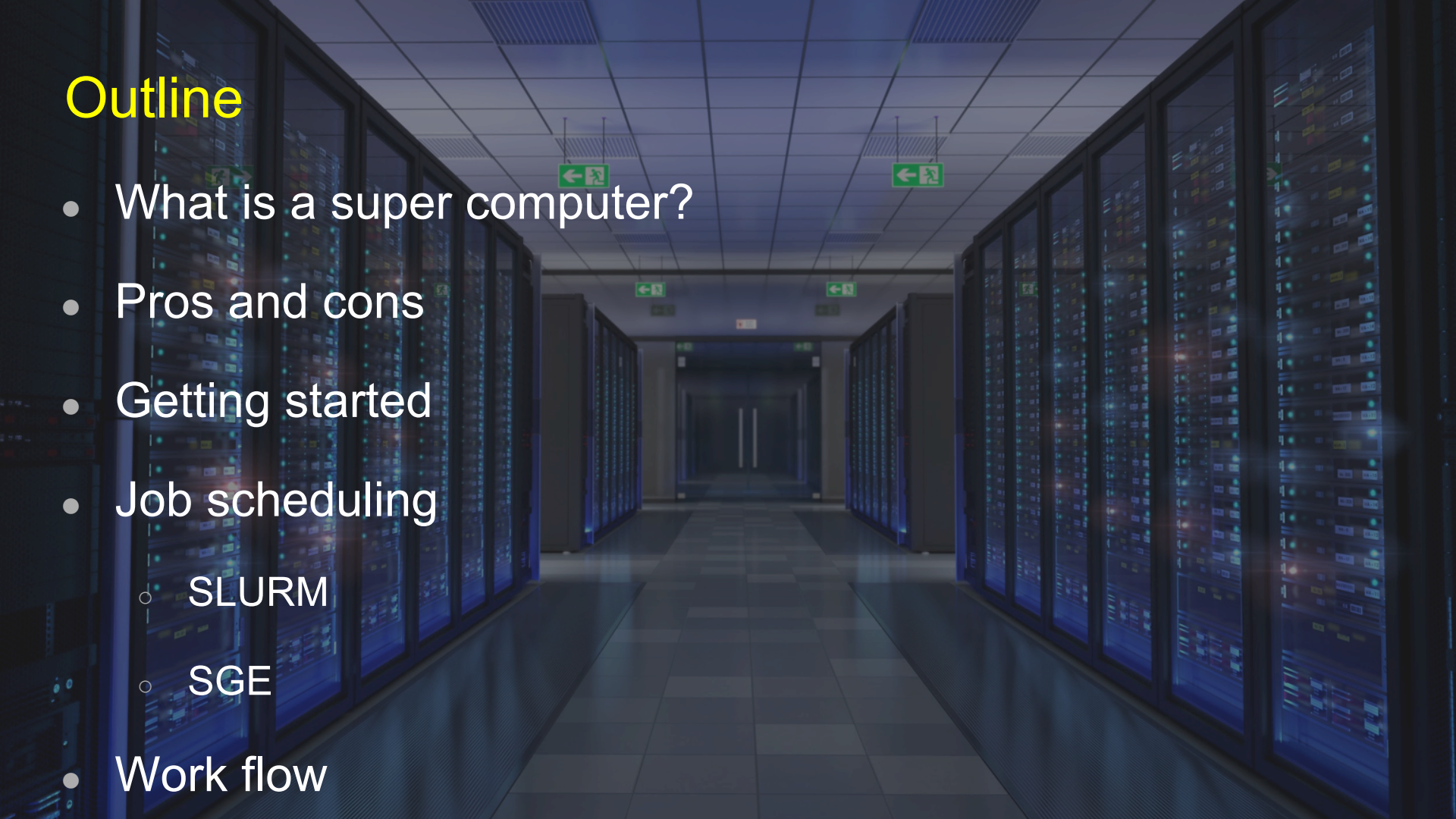
Introduction to Supercomputing

Brainhack Global DC

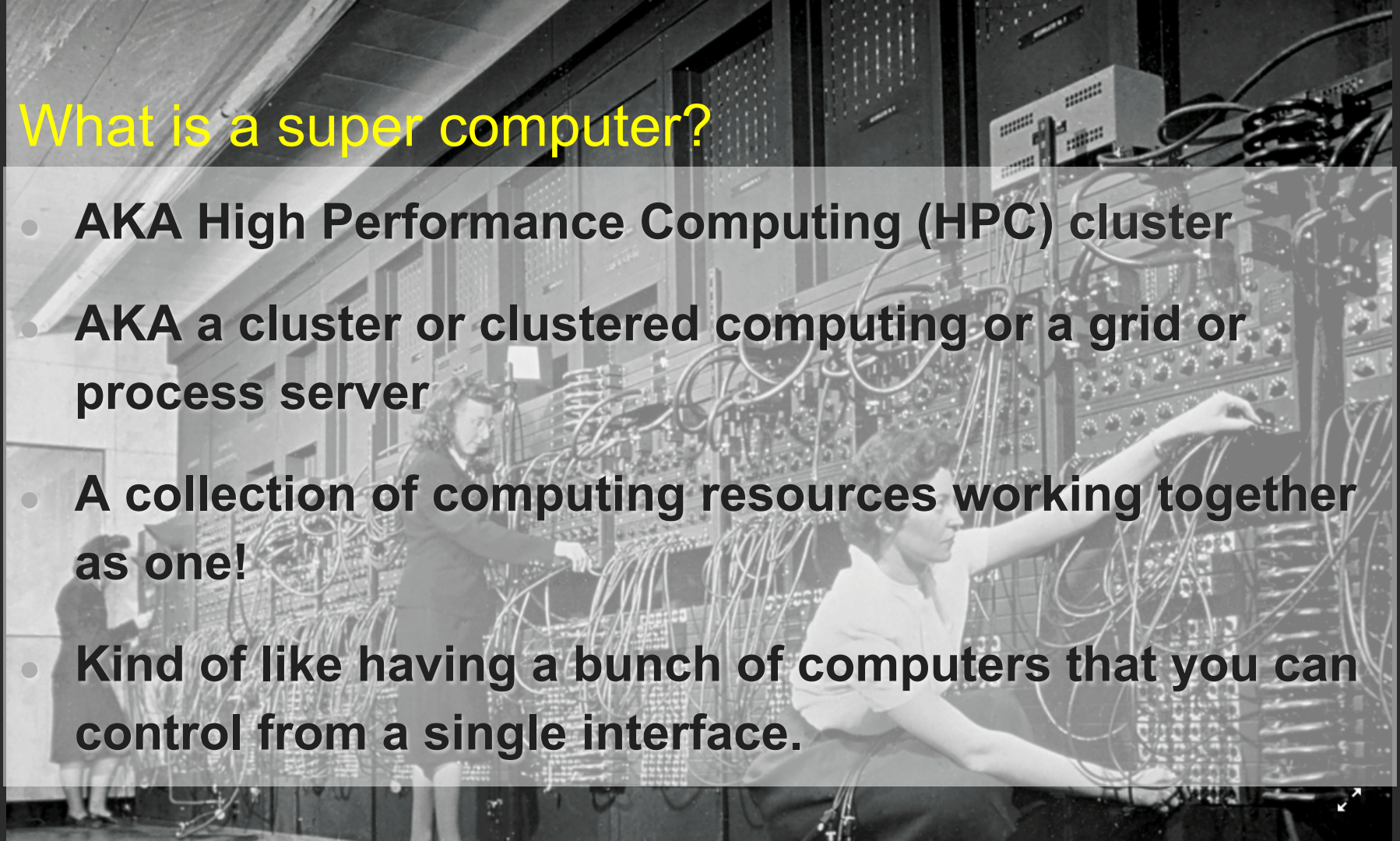
November 16, 2019

Junaid Merchant



Outline

- What is a super computer?
- Pros and cons
- Getting started
- Job scheduling
  - SLURM
  - SGE
- Work flow



What is a super computer?

- AKA High Performance Computing (HPC) cluster
- AKA a cluster or clustered computing or a grid or process server
- A collection of computing resources working together as one!
- Kind of like having a bunch of computers that you can control from a single interface.

Important Terminology

Node: a computing unit that is comprised of a CPU, RAM, and maybe GPU

- Like a single computer
- Different nodes might have different specs

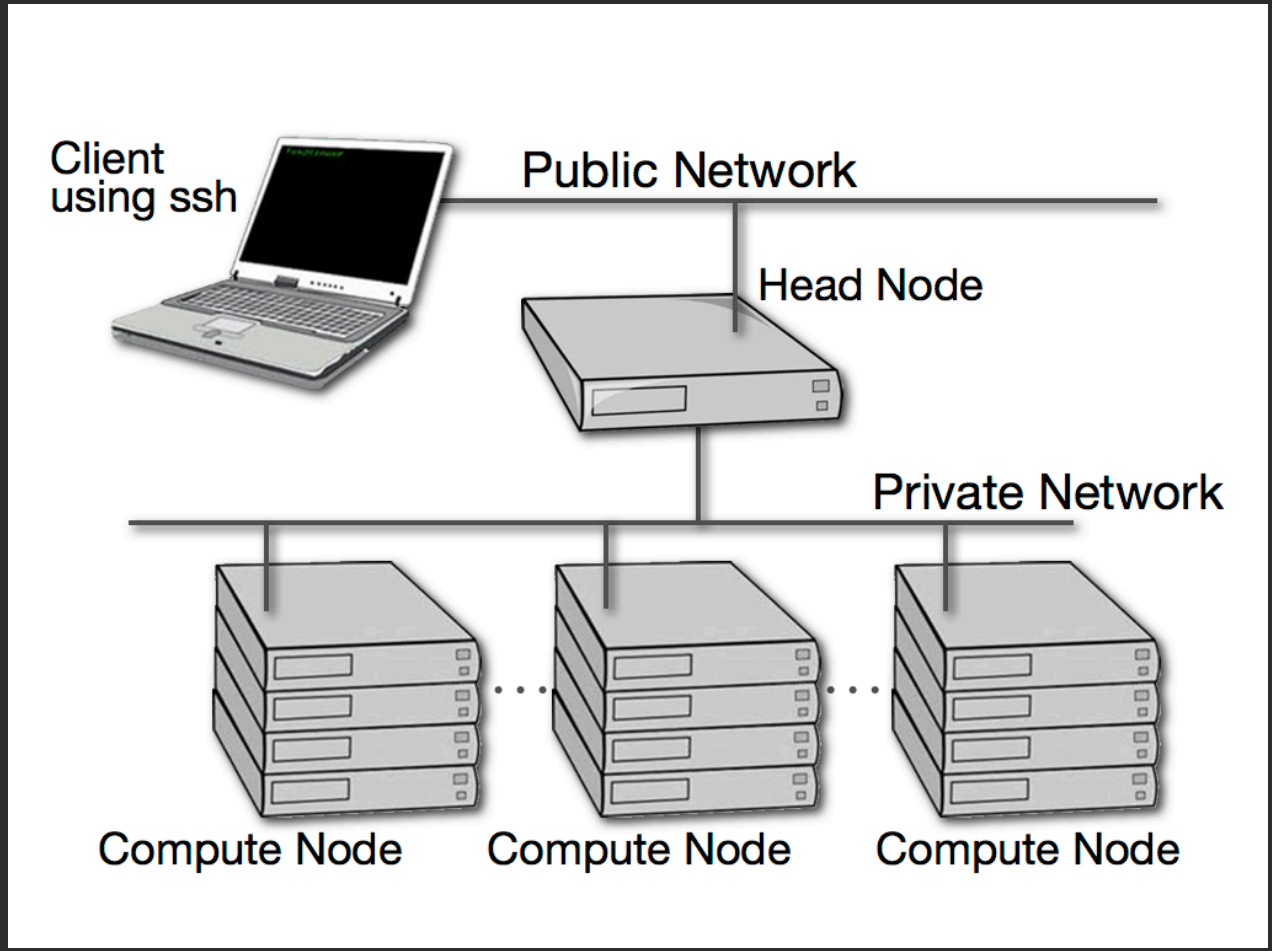
Head node: Where you (and everyone else) login & submit jobs from

- Master node that decides which node jobs are submitted to
- Shared resource, so DON'T run any jobs here

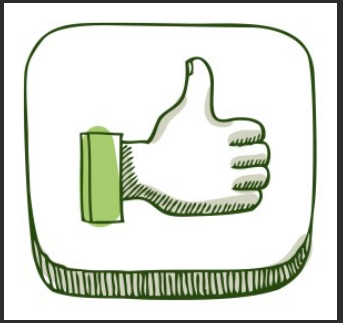
Job: A single process or script or program that you want to run on a node

- Can be comprised of subprocesses
- must run with no interaction (more on this later)

Important Terminology



Pros



More processing power >= Sum of it's parts!

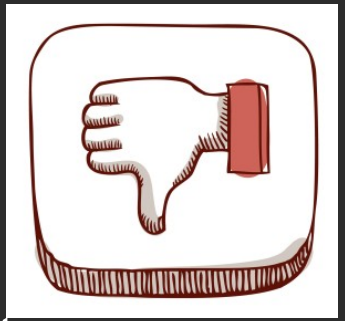
- Parallel and distributed processing (a lot of data all at once).
- High specification nodes (e.g. nodes with crazy amounts of RAM)

Centralized data that is maintained and backed up

- Sign in from anywhere

Potentially free!

Cons



Requires grid commands to harness parallelization

- Different workflow approach than what you may be used to
- A little bit of a learning curve

No virtual desktop environment (no VNC) or SMB

- Requires comfort with the terminal/command line
- But there are some work around for graphic user interface

Requires some level of conscientious use

Getting started

Find the right HPC for you, and get an allocation:

National/International options:

https://www.xsede.org/

https://www.nsgportal.org/

University of Maryland:

http://hpcc.umd.edu/

https://oacs.umd.edu/oacs-cloud/bsos-high-performance-computing-cluster

Georgetown University:

https://wiki.uis.georgetown.edu/display/HPCMedusa/

Getting started

Figure out the node specifications you need for your processing:

- Different nodes have different specifications. If you need really high specs, you might have to wait.
- Learn what job scheduling software is used
- Find the software applications that are available

Getting started: Find your specs (example)

The following table lists the hardware on the Deepthought2 cluster:

Description	Processor	Number of nodes	Cores/node	Memory/node GB	Memory/core GB	Scratch space per node, GB	GPUs/node	Interconnect
C8220	Ivy Bridge, 2.8 GHz	444	20	128	6.4	750	0	FDR Infiniband
C8220X	Ivy Bridge, 2.8 GHz	40	20	128	6.4	750	2	FDR Infiniband
Poweredge R820	Ivy Bridge, 2.2 GHz	4	40	1024	25.6	750	0	FDR Infiniband

All of the nodes except for the 1 TB RAM nodes have dual Intel Ivy Bridge E5-2680v2 processors running at 2.80 GHz. Memory is DDR3 at 1866 MHz.

The 1 TB RAM nodes require a different processor in order to support that much memory. These have quad Intel Xeon Ivy Bridge E5-4640v2 processors running at 2.20 GHz. The memory is DDR3 at 1333 MHz in these nodes.

The nodes containing GPUs have dual Nvidia Tesla K20m (GK110GL) GPUs (supporting Cuda compute capability 3.5).

The cluster has over 1 PB of file storage, and has FDR infiniband interconnects between the nodes with a theoretical maximum throughput of about 56 Gb/s.

Getting started: connecting to your server

Once you have an account, you can shell in through a terminal; for example:

```
ssh jmerch@login.bswift.umd.edu
```

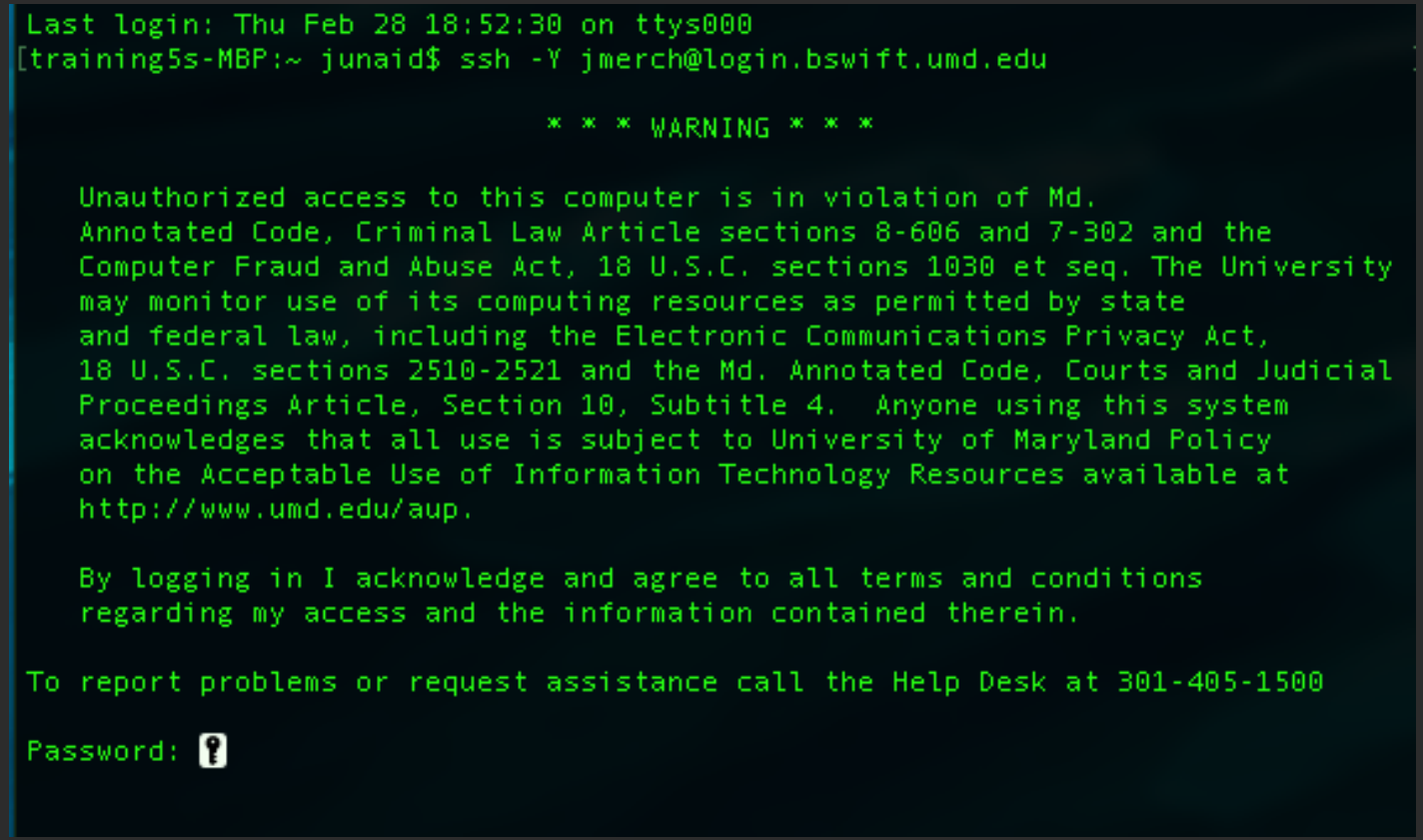
```
ssh <user name>@<server address>
```

Enter your password when prompted

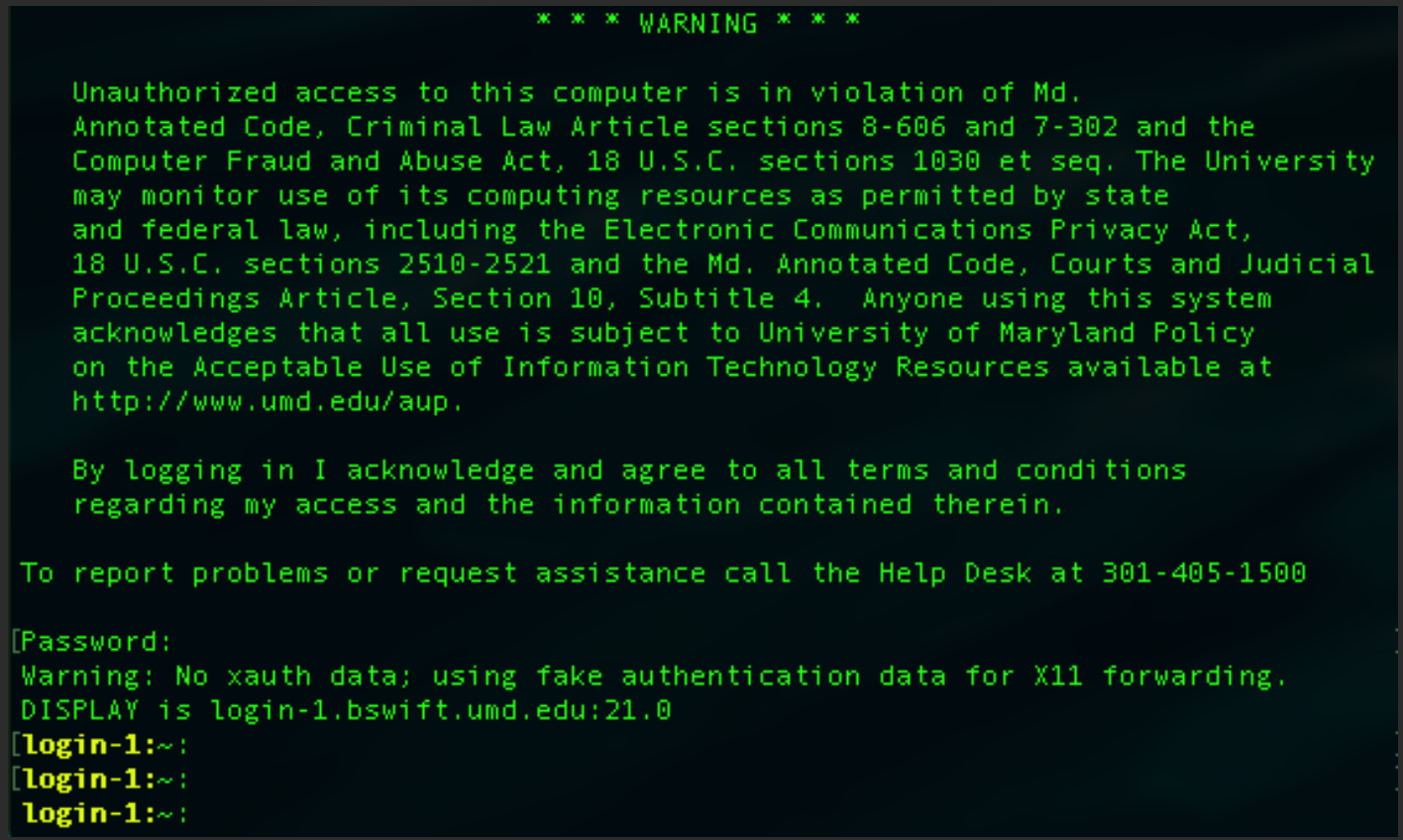
Now you’re on the head node of the HPC, where you can run basic commands, explore the directory structure, and submit jobs

Do NOT run big jobs on the head node

Getting started: connecting to your server



Getting started: connecting to your server



Getting started: bash commands

I cannot do a comprehensive tutorial of bash commands, so you'll have to do your homework on that end, or HPC will not work out for you.

However, there are plenty of good guides if you're unfamiliar with bash:

https://lifehacker.com/a-command-line-primer-for-beginners-5633909

http://swcarpentry.github.io/shell-novice/

Alternatively, if you are more familiar with another programming language, you can shell in, load your preferred flavor of python, for example, and work in that language. I'm more proficient at bash, so I'll be showing you everything in bash.

Getting started: bash commands

Here are a list of commands to get familiar with to get started:

- ls – list; list items in current directory if no other options are given.
- cd – change directory; change your current working directory.
- mkdir – make directory; create a new directory
- cp – copy; you can copy files or directories with this command.
- mv – move; you can move files or directories with this command.
- pwd – present working directory; find out what directory you are in.
- rm – remove; delete files or folders.

Getting started: bash commands

Here are a list of commands to get familiar with to get started:

• for – for loops; really useful! (We'll look at a for loop later as a way of submitting a bunch of job files)

• if/then – if-then statements really useful for bash scripting!

• tar – compress/archive files or folders; really useful for copying files to/from HPC (really speeds up transfers). To compress and archive an entire directory:

tar -zcvf /path/to/create/Folder.tar.gz /path/to/folder/

Getting started: bash commands

Here are some ways to move data between your local machine and server:

scp – secure copy; copy files to/from the HPC server.

You can use FileZilla for a graphical interface transfer: https://filezilla-project.org/ , but I've occasionally gotten corrupted files using FileZilla.

Getting started: bash commands

Here is a way to edit code from within your terminal:

nano – Command line text editor. There are others, but this is the easiest to use. This will allow you to easily create new scripts, or edit existing ones without having to copy the scripts/code/text files back and forth to/from HPC.

To create a new script in your current directory:

nano NewScript.sh

And enter or copy/paste in whatever code you want. Hit control+x to exit, and type ‘y’ to save the edits.

Alternatively, you can use vim: https://en.wikipedia.org/wiki/Vim_(text_editor)

Getting started: Software

MRI analysis software: MATLAB and SPM (and whatever related toolboxes you upload), FSL, AFNI, FreeSurfer, ANTS etc.

All basic Linux/Unix bash commands

Other languages/software: Python, R, Perl, code compilers, containers and literally 100s more applications

Talk to your system administrator to get new software installed

Or, you can type the following to get the full list: `module avail`

Getting started: Software

```
[login-1:~: module avail

------------------------------------------------------------------- /usr/local/Modules/versions --------------
3.2.10         3.2.9          3.2.9+flavours

------------------------------------------------------------------ /usr/local/Modules/3.2.10/modulefiles ---------
dot         module-git  module-info modules     null        use.own

------------------------------------------------------------------ /cell_root/system/common/modulefiles/sys -------
BESST/2.1                                                       leveldb/1.18
BESST/2.2.6                                                     libdatrie/0.2.12/gnu/4.9.3
GapCloser/1.12-r6                                               libdatrie/gnu/4.9.3/0.2.12
R/3.0.3                                                         libdatrie/hold
R/3.1.2                                                         libint/2.4.2/gnu/6.1.0/nosimd/openmp
R/3.2.2                                                         libint/gnu/6.1.0/nosimd/openmp/2.4.2
R/3.3.2                                                         libqglviewer/2.6.3
R/3.5.1                                                         libreoffice/3.3
RAxML/8.1.22/hybrid/avx                                         libreoffice/5.1.3
RAxML/8.1.22/hybrid/sse3                                        librs/3.1.0
RAxML/8.1.22/mpi/avx                                            libsx/2.04
RAxML/8.1.22/mpi/sse3                                           libxc/4.2.3/gnu/6.1.0
RAxML/8.1.22/pthreads/avx                                       libxc/gnu/6.1.0/4.2.3
RAxML/8.1.22/pthreads/sse3                                      ligo/20120612
SOAPdenovo/2.04-r240                                            lis/7.1rp7/esmf/5.2.0rp3
acigs/staff                                                     lis/7.1rp7/intelmpi-mt
afni/16.0.00                                                    lis/7.1rp7/nompi
afni/17.2.10                                                    llvm/4.0.1
agalma/0.5.0/python/2.7.8                                       lncRNApipe/1.0.7
agalma/1.0.0                                                    lz4/1.7.5
```

Getting started: Software

Now, unlike your personal workstation, the software is not ready to run.

You have to load it first using the following command:

```
module load <application name>
```

Make sure you type the application name exactly how it's specified in module list.

If there are multiple versions available, make sure you specify. For example:

```
module load matlab/2018b
```

Now you can launch the application by typing the name of the software or use this in your job submission code!

Job Scheduling: The heart of supercomputing

The real utility of working on a Supercomputer is through job submission

- Run jobs in parallel: multiple jobs at once!
- Distributes to node w/most available resources
- If a job crashes, it doesn’t stop everything else
- Don’t need to monopolize a lab workstation
- Can run heavy-duty jobs that you can’t on a PC

Job Scheduling: Job Scripts

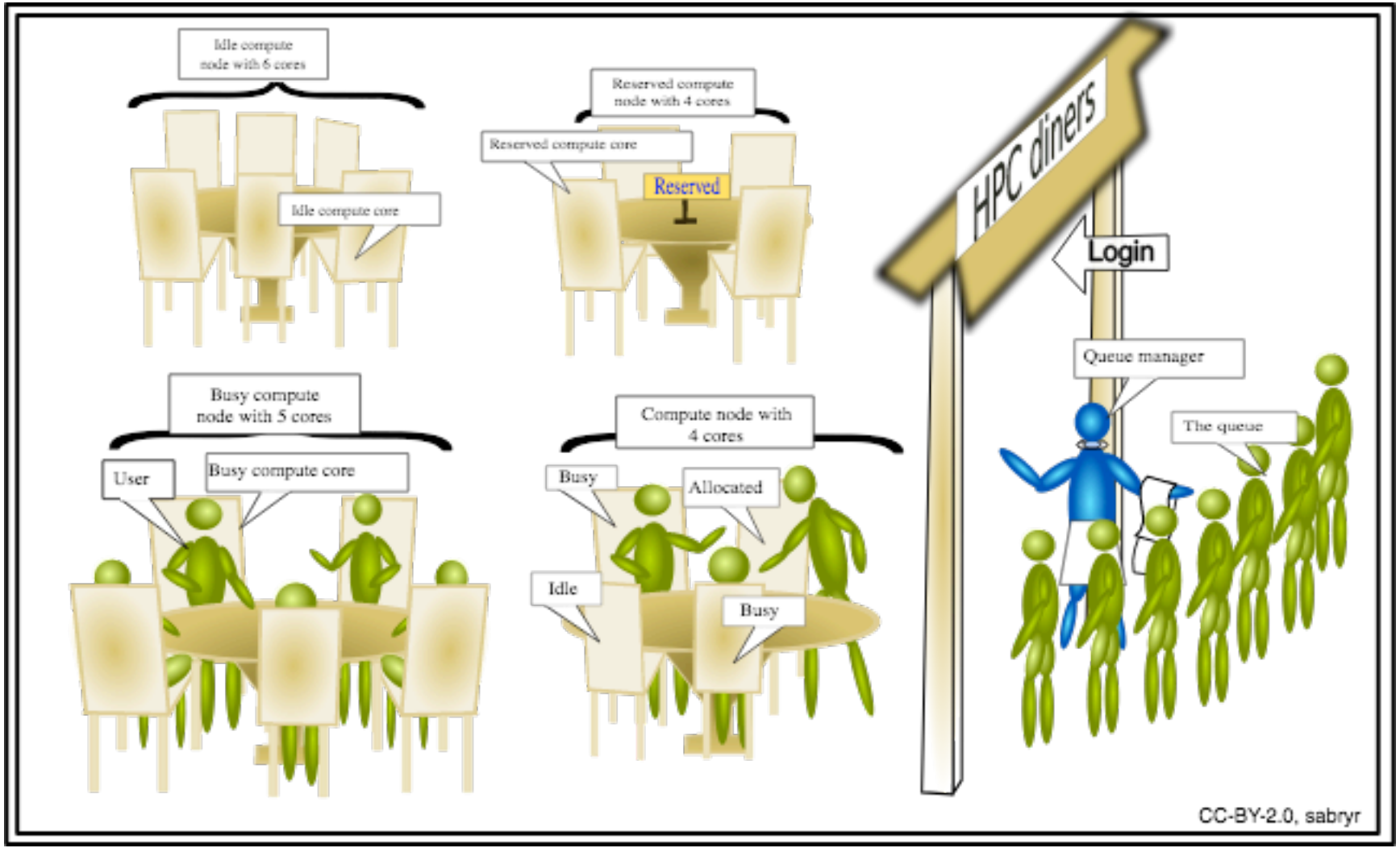
- Remember, a job is a set of processes that you want to run on a node (e.g. preprocessing pipeline for a single subject)
- All commands comprising the job must be contained in a script that is in the programming language of your choice
- Job script must contain specifications you require for the job: amount of time, # of nodes, # of CPUs, RAM etc
- Job script must contain software loadings and specific paths required, i.e. module load software, & absolute paths to data
- Must be able to run job file without interaction!
- Can test job scripts on an interactive node (more on this later)

Job Scheduling: Submitting Jobs

Once your job script is ready and tested, you can submit to the supercomputer!

- When you submit a job, the job scheduler finds an open node that matches your specification needs, runs the job there
- If there aren't available nodes, or if the specifications you require are greater than what is available, your job is put in queue till the next available node is open
- Always a good idea to not ask for more resources than you need for quicker job submission
- Once submitted, a log file will be generated that contains what would normally output to the terminal

https://hpc-carpentry.github.io/hpc-intro/13-scheduler/index.html



Job Scheduling

There are a number of job scheduling software available, but I will focus on 2 that are commonly used on research dedicated supercomputers:

SLURM – uses the command sbatch to submit jobs
SGE – uses the command qsub to submit jobs

Both systems use different commands to submit a job, check status of a job, delete a job, and launch an interactive session

Will go into the specifics of these systems in just a bit!

Job Scheduling

IMPORTANT THINGS TO REMEMBER WHEN WRITING JOB SCRIPTS FOR SUBMISSION

It’s totally non-interactive, so make sure you write and test properly before submitting

Paths--when a job is submitted, it starts in your homes directory, so give full paths where ever you can

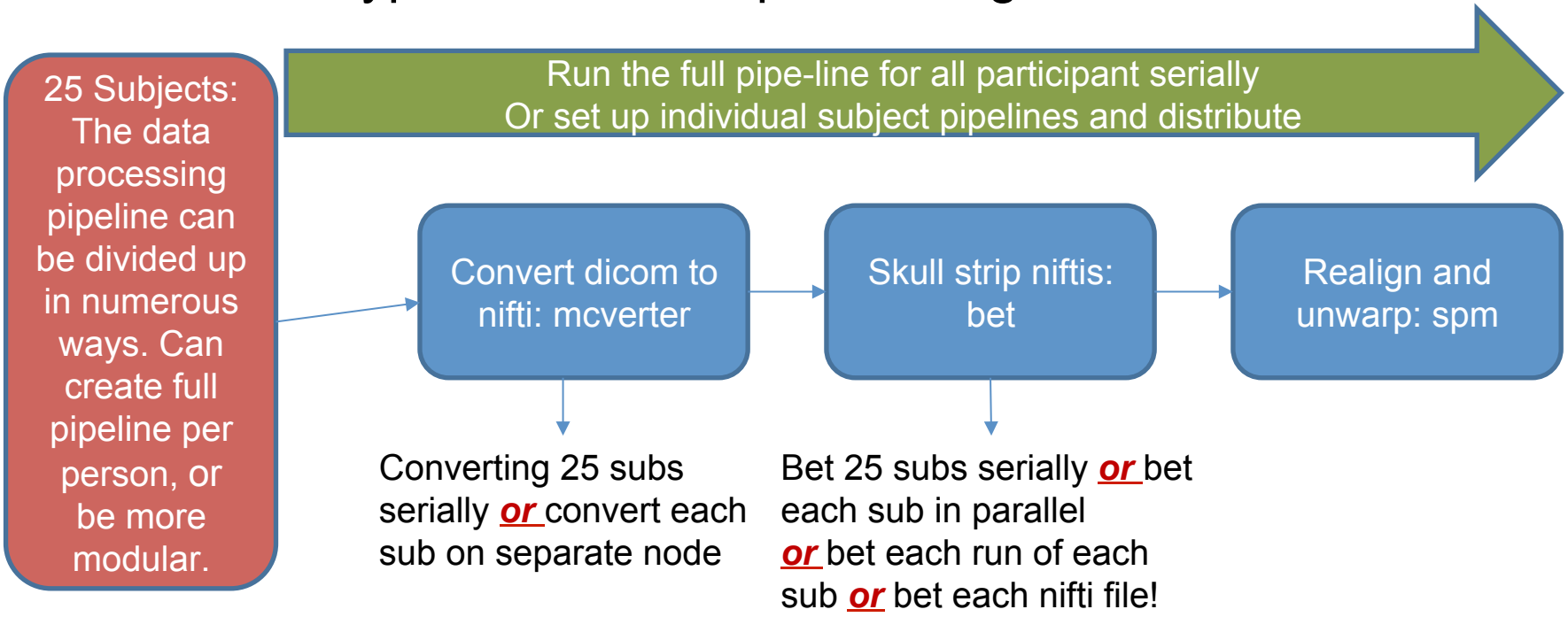
Make sure you specify the options correctly

You can specify a lot of other things: https://slurm.schedmd.com/sbatch.html

or http://www.dartmouth.edu/~rc/HPC/man/qsub.html

What and When to parallelize

Hypothetical data processing section:



How to Parallelize Jobs

Loops – First learn how to do a simple loop in bash, tcsh, or whatever langauge

- Now, if you have a number of job scripts you want to run, you can create a loop to submit all of them
- Even more fancy, you can create a job script that takes an input (e.g. subject ID) which performs the process on the input, and loop through subject IDs

Array Jobs – a single script that you submit which specifies what jobs you want to run in parallel & the scheduler distributes it for you

- More eloquent, but I don't have as much on this. More info here:

https://slurm.schedmd.com/job_array.html

https://docs.hpc.shef.ac.uk/en/latest/parallel/JobArray.html

Before going into the specifics…

For more, interactive help on the basics of supercomputing, I highly recommend working your way through this software carpentry-style tutorial: https://hpc-carpentry.github.io/hpc-intro/

SLURM

SLURM

Simple Linux Utility for Resource Management (SLURM)

Workload manger, job scheduler, & queuing system for running processes on an HPC

Figures out what specifications you need, finds the available node that meets the requirements, and allocates your job to that node!

Runs multiple jobs in parallel at once.

Use in conjunction with the job script you want to run

https://slurm.schedmd.com/quickstart.html

SLURM Commands: sbatch

sbatch – probably the most important command

Allows you to submit a job to the appropriate node

Use with job script for whatever sort of process you want to run

Things you absolutely MUST specify when using sbatch to submit a job:

- # of Nodes
- Amount of RAM
- Time to process the job

You can specify a lot of other things: https://slurm.schedmd.com/sbatch.html

SLURM Commands: sbatch

sbatch https://slurm.schedmd.com/sbatch.html

Usage: `sbatch [options] <job script>`

For example: `sbatch RunSimulation.sh`

This will give you a job number that you can check in on later

```
[login-1:~: sbatch example.sh
Submitted batch job 82581
login-1:~:
```

SLURM Job Scripts

Before going too much further, we should look inside a job script

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.edu
#SBATCH --mail-type=ALL
#
module load singularity
#
echo Starting fMRIprep at:
echo working on sub-JAM014
date
echo ----------------------------------------------------------------
```

SLURM Job Scripts

Before going too much further, we should look inside a job script

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.edu
#SBATCH --mail-type=ALL
#
module load singularity
#
echo Starting fMRIprep at:
echo working on sub-JAM014
date
echo ----------------------------------------------------------------
```

First line is called the 'shebang' and indicates what type of script it is. In this case it is bash. You can do python this way:
#!/bin/python
Or R:
#!/usr/bin/env Rscript
Matlab is a little different (more on this later)

SLURM Job Scripts

Before going too much further, we should

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terp
ail.umd.ed
#SBATCH --mail-type=ALL
#
module load singularity
#
echo Starting fMRIprep at:
echo working on sub-JAM014
date
echo ----------
```

The next few lines are sbatch options. These options can be specified within the job script, or outside of the script as I described before:

`sbatch [options] <job script>`

-Time is the amount of time you want allocated in hours:minutes:seconds

-Nodes is # of nodes

-Mem is amount of RAM in megabytes

-Output is the name of the output log file that contains what would be printed to terminal and any errors that occurred

-Mail-you can give your email so it sends you a message when the job starts and ends!

SLURM Job Scripts

Before going too much further, we should look inside a job script

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.ed
#SBATCH --mail-type=ALL
#
module load singularity
#
echo Starting fMRIprep at:
echo working on sub-JAM014
date
echo ----------------------------------------------------------------------
```

The remainder of the script is what you want it to do! If you are wanting to load any programs, do that in the first few lines, and have at it!

SLURM Job Scripts

Now, let's look at the special case of creating a job script with a MATLAB process.

For this, you want to create a bash script that looks the same as the previous example, but then calls matlab

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.edu
#SBATCH --mail-type=ALL
#
module load matlab/2018b
#
matlab -nosplash -nodisplay -nodesktop -r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

SLURM Job Scripts

Now, let's look at the special case of creating a job script with a MATLAB process.

For this, you want to create a bash script that looks the same as the previous example, but then calls matlab

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.edu
#SBATCH --mail-type=ALL
#
module load matlab/2018b
#
matlab -nosplash -nodisplay -nodesktop -r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

First, load the version of matlab that you want

SLURM Job Scripts

Now, let's look at the special case of creating a job script with a MATLAB process.

For this, you want to create a bash script that looks the same as the previous example, but then calls matlab

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.ed
#SBATCH --mail-type=ALL
#
module load matlab/2018b
#
matlab -nosplash -nodisplay -nodesktop -r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

Next, you want to start matlab using the nodisplay options because otherwise it will try to launch the matlab window

SLURM Job Scripts

Now, let's look at the special case of cr process.

For this, you want to create a bash scri us example, but then calls matlab

```
#!/bin/bash
#
#
#SBATCH --time=144:00:00
#SBATCH --nodes=1
#SBATCH --mem=24000
#SBATCH --output=sub-JAM014
#SBATCH --mail-user=jmerch@terpmail.umd.edu
#SBATCH --mail-type=ALL
#
module load matlab/2018b
#
matlab -nosplash -nodisplay -nodesktop -r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

Finally, you want to feed into the matlab command everything you want to do within matlab using the -r option, followed by all the matlab operations within quotes. This can be tricky if you're used to using matlab interactively, so I recommend making sure you have a set of working matlab that you test outside of a job submission before starting this.

Matlab/SPM tricks

Note: If you are using Matlab, you must launch it with no graphical interface. The below example loads the default matlab (2017a), and launches it in the terminal with no display, desktop, or Java:

```
module load matlab
matlab -nodesktop -nodisplay -nojvm
```

You can also launch Matlab, and tell it to run the commands/scripts following the '-r' option, which will become useful when we go over job submission:

```
matlab -nodesktop -nodisplay -nojvm -r
"run('YourCommand.m'); exit"
```

Matlab/SPM tricks

Matlab from the command line:

```
matlab -nodisplay -nodesktop -nosplash -nojvm
```

Spm from the command line. If you already have SPM batch jobs created and saved as .mat:

```
% initiates spm configuration
spm_jobman('initcfg');
% load spm job .mat
load('path/to/spm/batchjob.mat');
% run job without GUI
spm_jobman('run',matlabbatch);
```

Matlab/SPM tricks

Now, combining what we learned in the previous 2 slides, here's how to launch matlab, and have it start running a SPM batch job:

```
matlab -nodesktop –nodisplay -nojvm –r
“spm_jobman('initcfg'); load('path/to/spm/batchjob.mat');
spm_jobman('run',matlabbatch); exit”
```

This will launch matlab with no interface, run the SPM batch job, and then exit out of matlab when it finishes. This is essentially what goes into job submission script so that it can be run on a non-interactive node. But, I'm getting ahead of myself..

SLURM Loops & Parallelization

For example, pretend you have EEG data for 50 subjects that you want to preprocess: `/path/to/subject/dir/sub_001 sub_002 ...`

First, you want to write a job script that takes subject ID as an input, which performs the process on the subject. PreprocessEEG.sh:

```
#!/bin/bash

CurrentSubject=$1

module load EEGprogram

Process $CurrentSubject

...
```

SLURM Loops & Parallelization

Now, you can loop this job script for all the subjects, and have them process simultaneously:

```
for sub in $(ls /path/to/subject/dir); do

sbatch PreprocessEEG.sh $sub

done
```

SLURM Loops & Parallelization

```
for sub in $(ls /path/to/subject/dir); do

sbatch PreprocessEEG.sh $sub

done
```

This should give you something like this:

Now go have a beer because you are

processing all the data simultaneously!

```
Submitted batch job 82500
Submitted batch job 82501
Submitted batch job 82502
Submitted batch job 82503
Submitted batch job 82504
Submitted batch job 82505
Submitted batch job 82506
Submitted batch job 82507
Submitted batch job 82508
Submitted batch job 82509
Submitted batch job 82510
Submitted batch job 82511
Submitted batch job 82512
Submitted batch job 82513
Submitted batch job 82514
Submitted batch job 82515
Submitted batch job 82516
Submitted batch job 82517
Submitted batch job 82518
```

Other SLURM Commands

squeue – to check the status of your jobs that are running. If you type this by itself, it will list all the jobs that are running/pending:

```
[login-1:~$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          52875_64  standard runeebo.   fklein  R 15-15:26:22      1 compute-4-26
          52875_65  standard runeebo.   fklein  R 15-15:26:22      1 compute-4-26
          52873_56  standard runeebo.   fklein  R 15-15:26:52      1 compute-4-25
          52873_60  standard runeebo.   fklein  R 15-15:26:52      1 compute-4-25
          52743_62  standard runeebo.   fklein  R 16-13:56:00      1 compute-4-25
          52743_63  standard runeebo.   fklein  R 16-13:56:00      1 compute-4-25
          52741_59  standard runeebo.   fklein  R 16-13:56:27      1 compute-4-26
          52741_61  standard runeebo.   fklein  R 16-13:56:27      1 compute-4-26
          52739_57  standard runeebo.   fklein  R 16-13:57:19      1 compute-4-27
          52629_53  standard runeebo.   fklein  R 17-11:23:58      1 compute-4-4
          52629_54  standard runeebo.   fklein  R 17-11:23:58      1 compute-4-4
          52629_55  standard runeebo.   fklein  R 17-11:23:58      1 compute-4-4
          52629_52  standard runeebo.   fklein  R 17-11:23:59      1 compute-4-4
             82557  standard SingPrep   jmerch  R    7:30:59      1 compute-5-2
             82580  standard js_ALL6_ mbotdorf  R    2:10:56      1 compute-4-2
             82578  standard js_ALL6_ mbotdorf  R    2:11:38      1 compute-4-1
        81612_160  standard runeela.   fklein  R 1-11:32:22      1 compute-4-28
```

Other SLURM Commands

squeue – you can be more specific with this queue, and just list the jobs you have submitted using the –u option followed by your user ID. For example:

```
squeue -u jmerch
```

```
[login-1:~$ squeue -u jmerch
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             82557  standard SingPrep   jmerch  R    7:31:08      1 compute-5-2
```

Other SLURM Commands

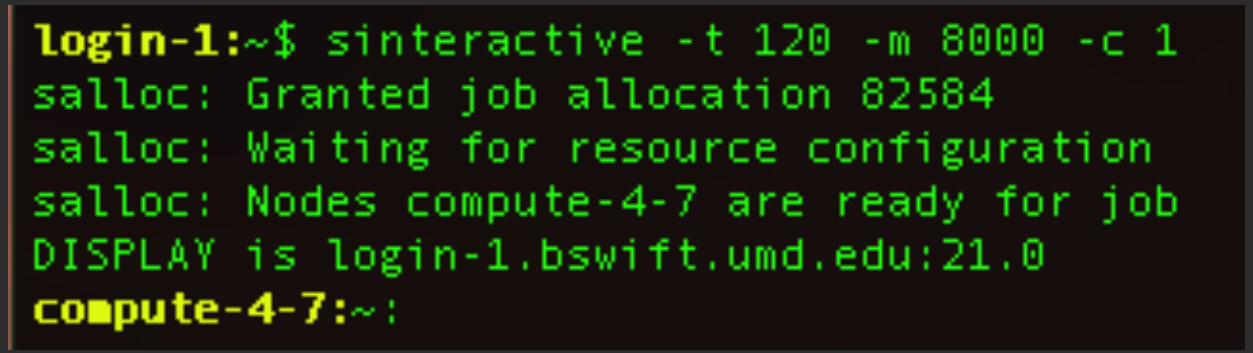
scancel – if you have a job running that you want to cancel, you can use this command followed by the job ID given by slurm; for example:

```
scancel 82557
```

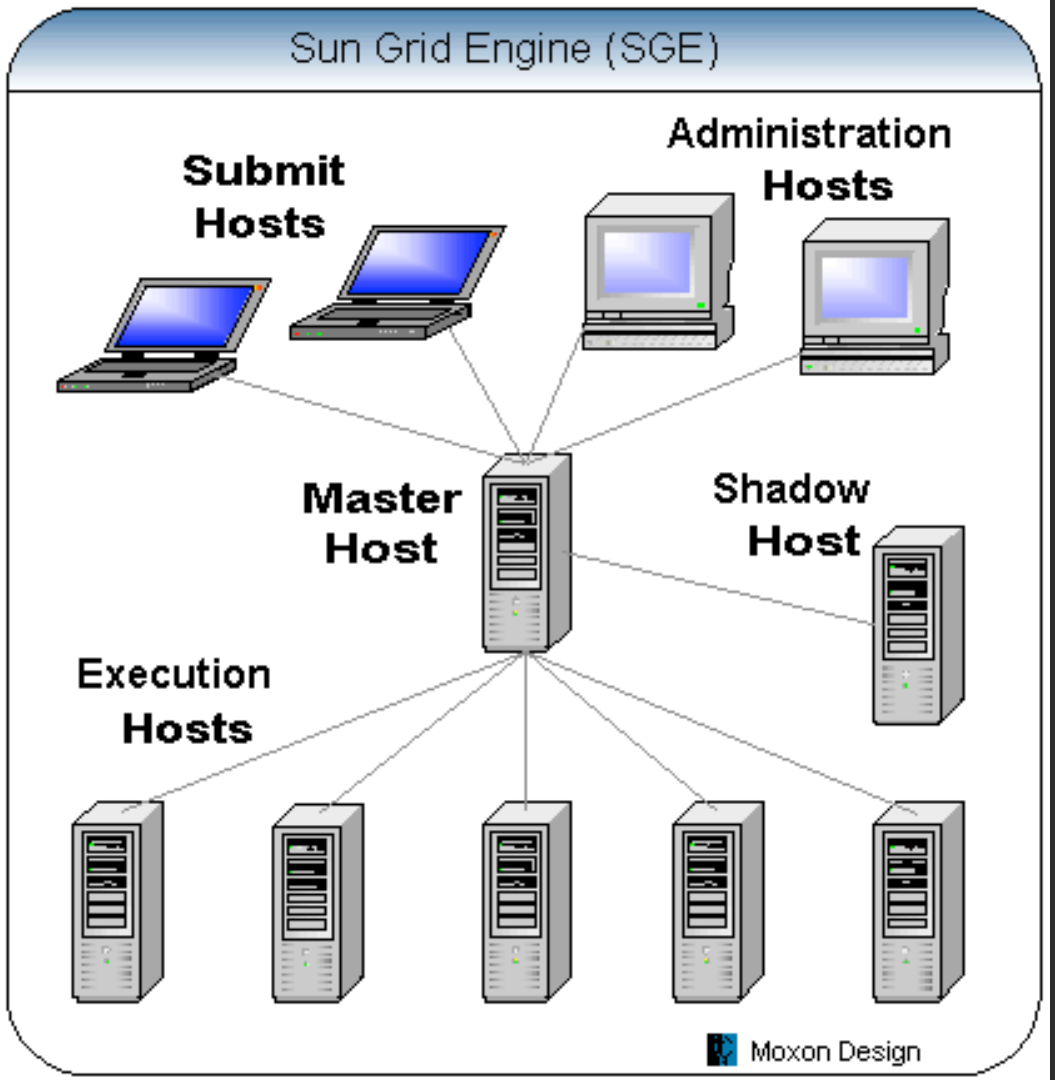
Other SLURM Commands

sinteractive – if you want to shell into one of the compute nodes to do some interactive processing, to test code for example, you can use this. Remember how I said never do any big processing on the head node, well you can shell in to one of the compute nodes and do some big processing. Like sbatch, you have to specify the time (in minutes), memory (in mb), and cpus. For example, this will give you an interactive node for 120 minutes, with 8 gb of RAM, and 1 CPU:

```
sinteractive -t 120 -m 8000 -c 1
```



SGE


Sun Grid Engine (SGE)
Submit Hosts
Administration Hosts
Master Host
Shadow Host
Execution Hosts
Moxon Design

SGE Commands: qsub

qsub – probably the most important command. Basic usage:

```
qsub [options] /path/to/jobfile.sh
```

Very similar to sbatch in that this contains all the processes you want to run in a script, and things you absolutely MUST specify when using qsub to submit a job:

- # of Nodes
- Amount of RAM
- Time to process the job

You can specify a lot of other things: http://www.upv.es/upl/U0115421.htm

SGE Commands

The more general form of the job submission commands are the ‘q’ commands:

- • qsub – submit a job to the que; this will submit whatever command/script that comes after to the node with the most available resources. This is worth spending the time to learn to learn all the options for.
- • qstat – monitors state of jobs that are running. Once you submit a job, this command lets you know if it is still running, where it is running, etc.
- • qdel – delete a job. If you want to stop a job, use this command.
- • qrsh – launch an interactive window on a new node. Whereas qsub’ing will start the job on a new node that you can’t see, use qrsh to have an interactive window. The downside to this approach is that once you close this window, it kills the processes.

Example script: qsub_spm_job.sh:

```
#!/bin/bash
# Bash wrapper to qsubmit spm .mat jobs to the Medusa. -JSM 20170523
#
# QSUB OPTIONS:
# This options starts in the current working directory
#$ -cwd
# Combine the output and error logs into one
#$ -j y
# Use the bash environment
#$ -S /bin/bash
# Use same environment variables
#$ -V
# Send email when job is finished.
#$ -m e -M jm3080@georgetown.edu
# Name the job
#$ -N qSubSpmJob
# Allocate 16 gb of ram for this job.
#$ -l h_vmem=16G
#
# COMMANDS TO RUN:
# load latest matlab module
module load matlab/R2017a
#
# Start matlab, submit the matlab commands entered after the '-r' option, then exits matlab.
# $1 variable is the SPM batch .mat file submitted with this script.
matlab -nosplash -nodisplay -nodesktop –r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

Example script: qsub_spm_job.sh:

```
#!/bin/bash
# Bash wrapper to qsubmit spm .mat jobs to the Medusa. -JSM 20170523
#
# QSUB OPTIONS:
# This options starts in the current working directory
#$ -cwd
# Combine the output and error logs into one
#$ -j y
# Use the bash environment
#$ -S /bin/bash
# Use same environment variables
#$ -V
# Send email when job is finished.
#$ -m e -M jm3080@georgetown.edu
# Name the job
#$ -N qSubSpmJob
# Allocate 16 gb of ram for this job.
#$ -l h_vmem=16G
#
# COMMANDS TO RUN:
# load latest matlab module
module load matlab/R2017a
#
# Start matlab, submit the matlab commands entered after the '-r' option, then exits matlab.
# $1 variable is the SPM batch .mat file submitted with this script.
matlab -nosplash -nodisplay -nodesktop –r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

```
#!/bin/bash
# Bash wrapper to qsubmit spm .mat jobs to the Medusa. -JSM 20170523
#
# QSUB OPTIONS:
# This options starts in the current working directory
#$ -cwd
# Combine the output and error logs into one
#$ -j y
# Use the bash environment
#$ -S /bin/bash
# Use same environment variables
#$ -V
# Send email when job is finished.
#$ -m e -M jm3080@georgetown.edu
# Name the job
#$ -N qSubSpmJob
# Allocate 16 gb of ram for this job.
#$ -l h_vmem=16G
#
# COMMANDS TO RUN:
# load latest matlab module
module load matlab/R2017a
#
# Start matlab, submit the matlab commands entered after the '-r' option, then exits matlab.
# $1 variable is the SPM batch .mat file submitted with this script.
matlab -nosplash -nodisplay -nodesktop –r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

Example job script for qsub

```
#!/bin/bash
# Bash wrapper to qsubmit spm .mat jobs to the Medusa. -JSM 20170523
#
# QSUB OPTIONS:
# This options starts in the current working directory
#$ -cwd
# Combine the output and error logs into one
#$ -j y
# Use the bash environment
#$ -S /bin/bash
# Use same environment variables
#$ -V
# Send email when job is finished.
#$ -m e -M jm3080@georgetown.edu
# Name the job
#$ -N qSubSpmJob
# Allocate 16 gb of ram for this job.
#$ -l h_vmem=16G

# COMMANDS TO RUN:
# load latest matlab module
module load matlab/R2017a

# Start matlab, submit the matlab commands entered after the '-r' option, then exits matlab.
# $1 variable is the SPM batch .mat file submitted with this script.

matlab -nosplash -nodisplay -nodesktop –r "clear; addpath('/share/apps/spm/spm12');
spm_jobman('initcfg'); load $1; spm_jobman('run',matlabbatch); exit"
```

Grid commands: qstat

Qstat shows status of your queued jobs

- Simply type *qstat*
- Gives information about jobs in your queue:
  - Job-ID and name, state of job, and what node it's on

```
junaid@ong9 ~ $ qstat
job-ID  prior   name       user         state submit/start at     queue                          slots
-----------------------------------------------------------------------------------------------------
 244551 0.55500 spmbatchte junaid       r     05/25/2016 08:53:56 all.q@ong4.uoregon.edu             1
 244552 0.55500 spmbatchte junaid       r     05/25/2016 08:53:56 all.q@ong0.uoregon.edu             1
 244553 0.55500 spmbatchte junaid       r     05/25/2016 08:53:56 all.q@ong2.uoregon.edu             1
junaid@ong9 ~ $
```

FSL commands: fsl_sub

fsl_sub is a bash wrapper to submit jobs for grid computing.
From FSL, but can use for non-FSL scripts.
Usage: fsl_sub <terminal command>

fsl_sub bet mprage.nii b_mprage.nii -R
 # works well with fsl bet function

FSL commands: fsl_sub

Example bet loop using fsl_sub:

```
module load fsl/fsl-5.0.9 # load fsl
cd ~/sanlab/Studies/SGE_test/REV001/ppc/functionals/BART1
# starting from inside a directory of functionals
for Sub in $(ls *.nii)
do
fsl_sub bet $Sub b_$Sub -R
done
```

FSL commands: fsl_sub

```
junaid@ong5 /home/research/sanlab/Studies/SGE_test/REV001/ppc/functionals/BART1 $ for Sub in $(ls *.nii)
> do
> fsl_sub bet $Sub b_$Sub -R
> done
258940
258941
258942
258943
258944
258945
258946
258947
258948
258949
258950
258951
258952
258953
258954
258955
258956
258957
258958
258959
258960
258961
258962
258963
258964
258965
258966
```

Questions?

I’m available for consulting on how to get you set up on an HPC for the price of some beer and/or food (or cash)!

#AlwaysLookingForASideHustle

merchantjs@gmail.com

828-301-3155