Getting Started

- Download the slides from:
  `https://pitt.box.com/v/ling1340-2020s`

Advanced Computing at Pitt CRC

Kim (kimwong), Fangping (fangping), **Barry** (bmooreii), Shervin Sammak (shs159), Leonardo Bernasconi (leb140)

Pitt Center for Research Computing

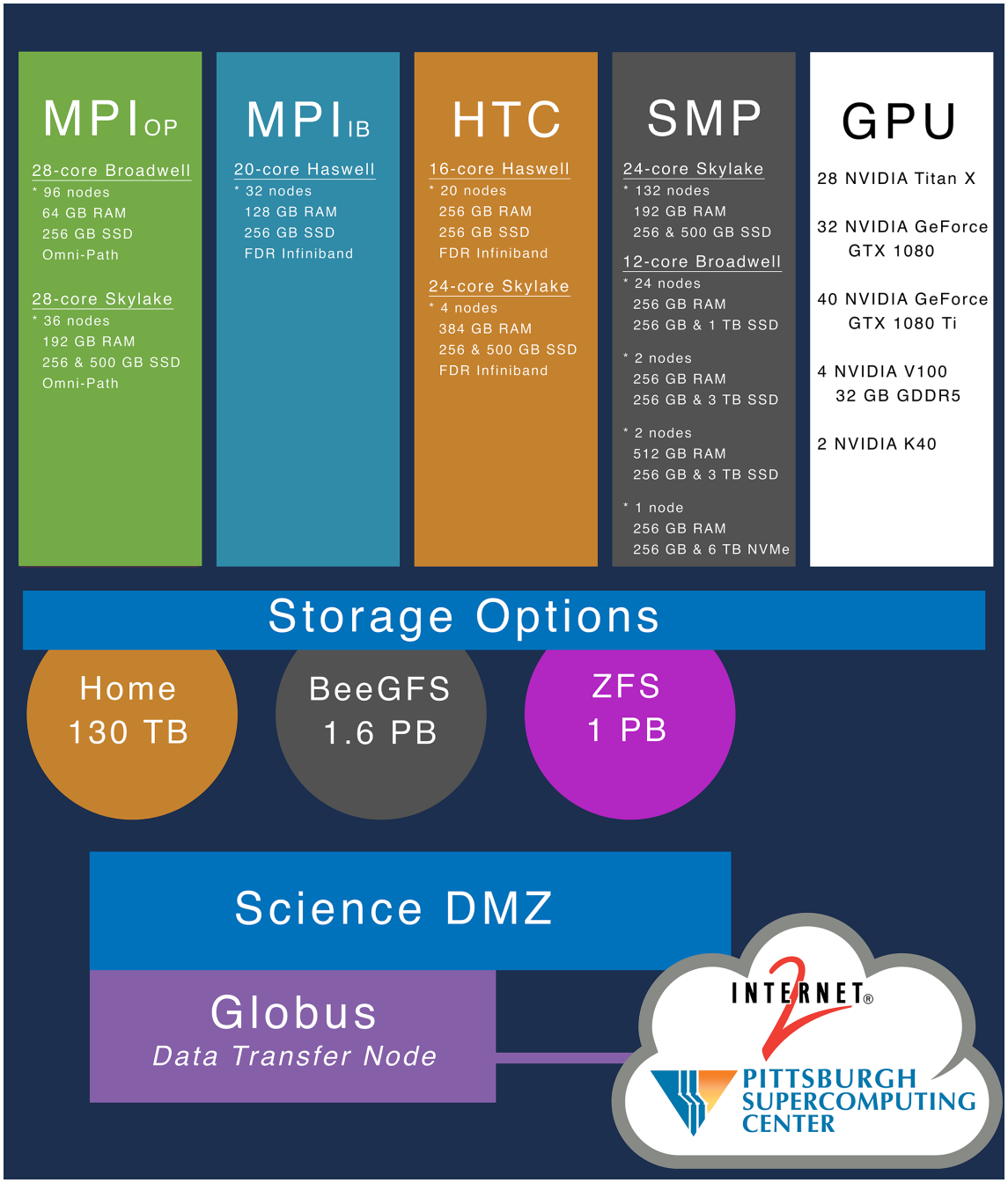
Me

- PhD in Theoretical Chemistry
- Assistant Research Professor
- Consultant at Pitt Center for Research Computing
- Research: Deep Learning applied to Ecology
- Interests: Programming with advanced type systems e.g. Haskell

What will you learn today?

- What is available at Pitt CRC?
- What does advanced computing look like?
- Accessing H2P
- Accessing software on H2P
- Submitting jobs to H2P

Hardware Overview

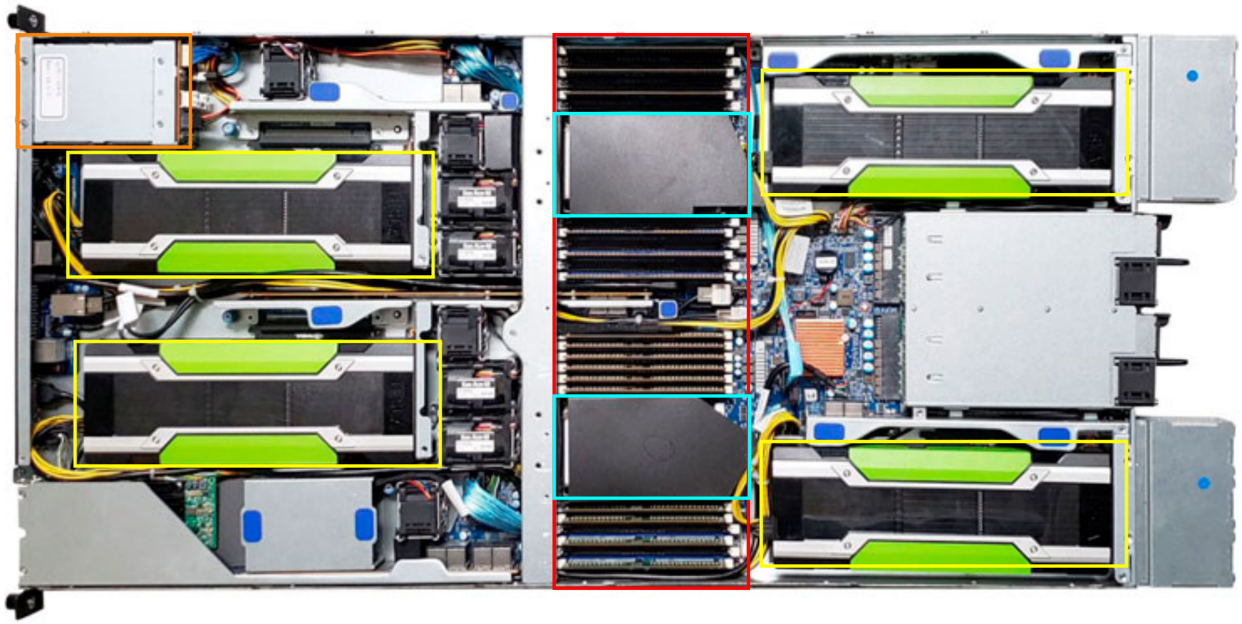


World's Most Powerful Computer

- Summit at Oak Ridge National Laboratory
  - 4,600 nodes
  - 6 V100 GPUs per node
  - 2 IBM POWER9 chips per node
  - 512GB RAM per node

What's Inside?

- Blue: CPUs – Compute Units
- Red: RAM – Fast memory fed to compute units when processing
- Yellow: GPUs – Specialized compute units for linear algebra
- Orange: SSDs or HDDs – Long term storage device
- `/ihome`, `/bgfs` – Network mounted storage with many HDDs to increase speed



Getting Connecting to the Cluster

- I usually show new folks to connect to the cluster via `ssh`, however
  - Installing software on folks computers takes too much time
  - Check out `https://pitt.box.com/v/cluster-training-spring2020` for `ssh` access
- For today, we are going to use Pitt CRC's JupyterHub
- Log in to `https://hub.crc.pitt.edu`

An Aside: Linux Text Editors

- On the cluster, there is no Microsoft Word. You will likely need to use a console editor:
  - `nano` – **use this for today**
  - `emacs`
  - `vim`
- Warning, Personal Opinions:
  - Don't wait to learn `vim`
  - Mainly muscle memory, therefore efficiency comes with time
  - Try using a graphical cheatsheet
  - Tweak your `.vimrc`, mine is here

Basic Linux Commands

- Going to go over **minimal** commands here
- Anatomy: `command <options> <input>`

Command	Description
`cd <directory>`	Change to `directory`
`echo <thing>`	Print `thing` to command line
`ls [<directory>]`	List contents of `directory`
`cp <file> <location>`	Copy `file` to a `location`
`man <command>`	Manual on `command`
`mkdir <directory>`	Make `directory`
`pwd`	Print working directory
`export [<VAR>=<thing>]`	Make or view environment variables

What Software is Available?

- You might ask, “How do I use Tensorflow!?”
- We use a tool called Lmod:

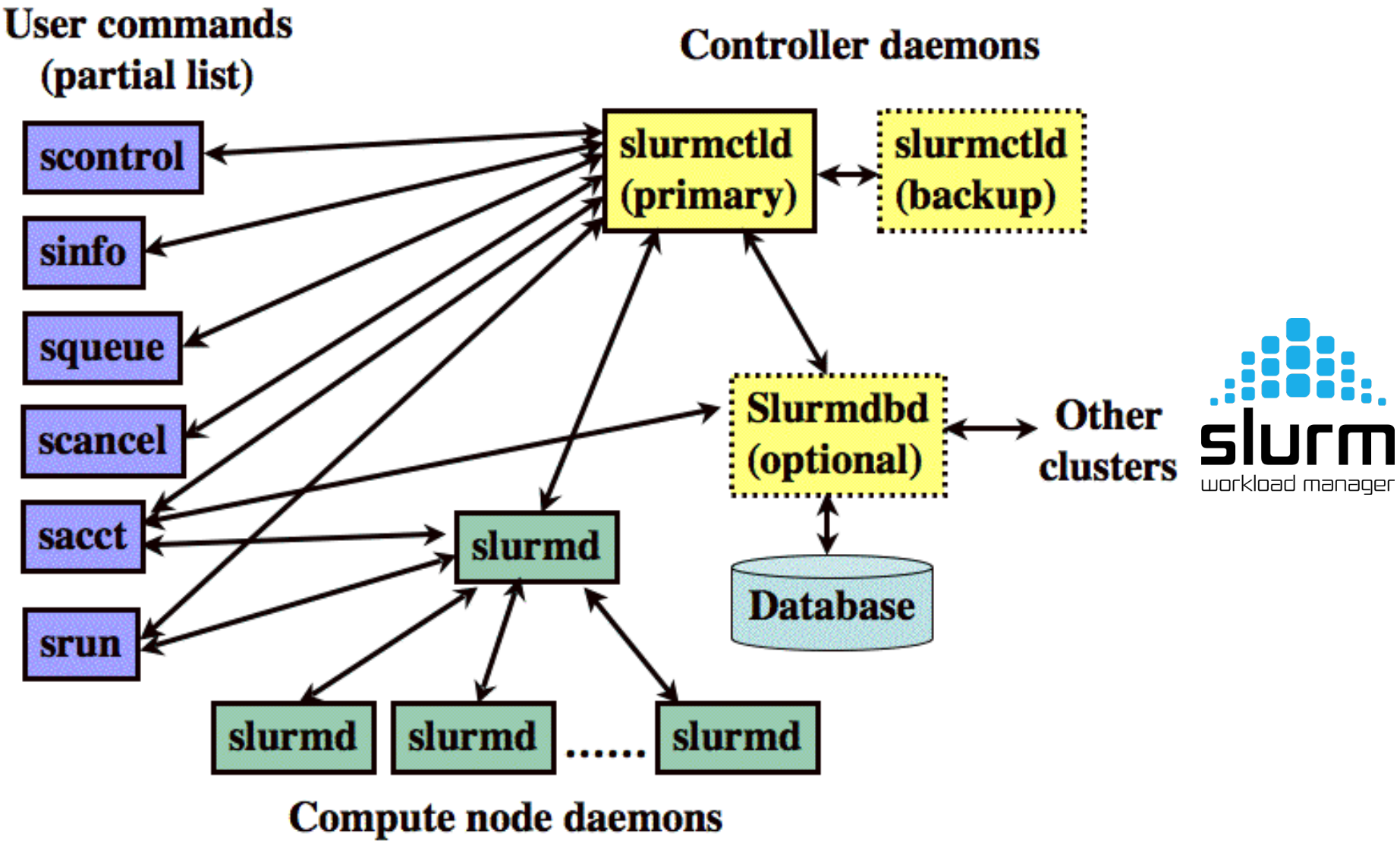
Command	Description
`module avail`	What’s available in my Lmod Path?
`module spider`	What’s available?
`module load <module>...`	Load a module
`module purge`	Unload all modules
`module list`	List loaded modules
`module save <name>`	Save module collection
`module savelist`	Show saved collections
`module restore <name>`	Load saved collection
`module describe <name>`	Show modules in collection

Batch Jobs

- The world's advanced research computers run *batch jobs*
- *batch* means we will let a scheduler control where and when our job with run
- A *job* is
  - a set of instructions describing resources needed (Cores, RAM, GPUs, etc)
  - a description of the environment to run in (Python 3.6)
  - a workload to complete (`python myCoolScript.py`)

How do you submit jobs?

High-Level Overview



Service Unit

- A service unit is approximately equivalent to using a compute core for 1 hour
- We take the maximum of your compute and RAM requests, imagine a computer with 12 cores and 8 GB RAM / core.

Num. CPUs	GB RAM	SUs
1	8	1
1	16	2
2	8	2
12	96	12

- GPU pricing is similar, but we charge for card hours

Slurm: Basic Commands

Command	Description
`sinfo`	Quick view of partitions
`smap`	Colored ncurses quick view of partitions
`sview`	Graphical view of partitions (detailed)
`sbatch <job>`	Submit your job[a]
`squeue`	View all jobs
`squeue -u <user>`	Look at **your** jobs
`sshare`	View group's fairshare
`sprio`	View queued jobs priorities
`crc-sinfo.py`	View all clusters/partitions
`crc-squeue.py`	Practical defaults for squeue
`crc-usage.pl`	View your group's usage
`crc-interactive.py`	Submit interactive jobs

[a] please use `.slm`, `.slurm` ending

Definitions

- Multifactor Priority – Determines which jobs get queued first
  - Age – How long have you been waiting?
  - Fair-share – Has everyone had an equal chance to compute?
  - Job size – Larger jobs are prioritized
  - Partition – Unused
  - QOS – Quality of service
    - short (max 1 day in queue) – 2
    - normal (max 3 days in queue) – 1
    - long (max 6 days in queue) – 0
  - Generic RESources – More GPUs are prioritized

Clusters and Partitions

```
10:37:19  login0:~ crc-sinfo.py
CLUSTER: gpu
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
gtx1080*        up   infinite     17    mix gpu-n[16-25],gpu-stage[08-14]
titanx          up   infinite      7    mix gpu-stage[01-07]
k40             up   infinite      1   idle smpgpu-n0
v100            up   infinite      1    mix gpu-n27

CLUSTER: mpi
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
opa*            up   infinite     75  alloc opa-n[0-33,35-44,49-63,68-73,78-81,86-89,94-95]
opa*            up   infinite     21   idle opa-n[34,45-48,64-67,74-77,82-85,90-93]
opa-high-mem    up   infinite      4    mix opa-n[125-128]
opa-high-mem    up   infinite     15  alloc opa-n[96-101,103-108,129-131]
opa-high-mem    up   infinite     17   idle opa-n[102,109-124]
ib              up   infinite      2    mix ib-n[9-10]
ib              up   infinite      9  alloc ib-n[0-8]
ib              up   infinite     21   idle ib-n[11-31]

CLUSTER: smp
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
smp*         up   infinite     10    mix smp-n[26,75-76,100,104,106,111,120,131,141]
smp*         up   infinite    141  alloc smp-n[0-16,19-25,27-74,77-99,101-103,105,107-110,112
high-mem     up   infinite      2    mix smp-512-n[1-2]
high-mem     up   infinite      3  alloc smp-256-n[1-2],smp-nvme-n1
```

Hardware

- Most Slurm commands accept a `--clusters, -M` argument

```
1  10:37:29  login0:~ scontrol -M smp show partition smp
2  PartitionName=smp
3     AllowGroups=ALL AllowAccounts=ALL AllowQos=short,normal,long,smp-smp-s,smp-smp-n,smp-smp-l
4     AllocNodes=ALL Default=YES QoS=N/A
5     DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
6     MaxNodes=1 MaxTime=UNLIMITED MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
7     Nodes=smp-n[24-122,126-155],smp-n[0-16,19-23]
8     PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
9     OverTimeLimit=NONE PreemptMode=OFF
10    State=UP TotalCPUs=3360 TotalNodes=151 SelectTypeParameters=NONE
11    DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
12    TRESBillingWeights=CPU=0.8,Mem=0.102G
```

- `Line 7`: What nodes are in this partition?
- `Line 10`: Total number of nodes and cores
- `Line 12`: Cost of service unit

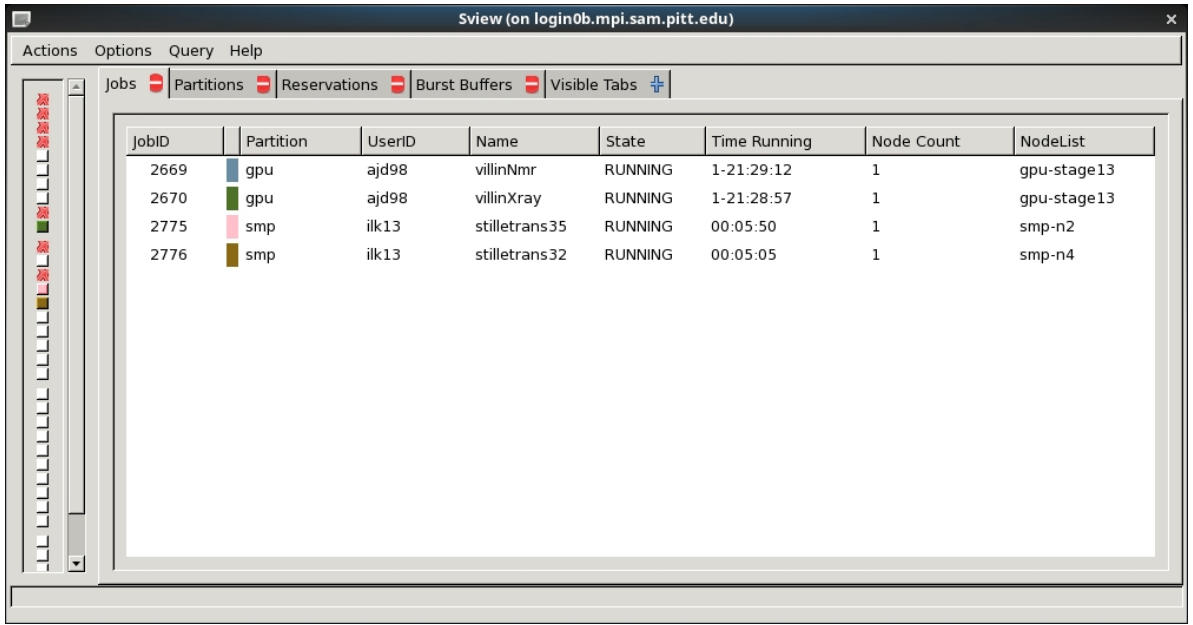
Partitions and Hardware – Graphical

`smap` – No X11 necessary

```
####.....C...DE...................

Mon Jan 09 11:05:08 2017
ID JOBID              PARTITION USER     NAME      ST      TIME NODES NODELIST
A  2669               gpu       ajd98    villinNmr R 1-21:28:12     1 gpu-stage13
B  2670               gpu       ajd98    villinXra R 1-21:27:57     1 gpu-stage13
C  2774               gpu       abb58    particles R   00:08:50     1 gpu-stage13
D  2775               smp       ilk13    stilletra R   00:04:50     1 smp-n2
E  2776               smp       ilk13    stilletra R   00:04:05     1 smp-n4
```

`sview` – Requires X11 forwarding



Looking at jobs – `crc-squeue.py`

```
10:40:10  login0:~ crc-squeue.py --help
 crc-sqeueue.py -- An squeue Slurm helper
Usage:
    crc-squeue.py [-hvasw]

Options:
    -h --help                       Print this screen and exit
    -v --version                    Print the version of crc-squeue.py
    -a --all                        Show all jobs
    -s --start                      Add the approximate start time
    -w --watch                      Updates information every 10 seconds

10:39:29  login0:~ crc-squeue.py
CLUSTER: gpu
  JOBID PAR                   NAME ST  TIME  NODES CPUS NODELIST(REASON)
 180416 tit  run-inference-pipeline PD  0:00      1   12      (Resources)

CLUSTER: mpi
  JOBID PAR                   NAME ST  TIME  NODES CPUS NODELIST(REASON)

CLUSTER: smp
  JOBID PAR                   NAME ST  TIME  NODES CPUS NODELIST(REASON)
```

- All of the `crc-*` scripts have `--help`

How many SUs left?

```
[bmooreii@login0b ~]$ crc-usage.pl
-----------------------------------------
        Total SUs (CPU Hours):  1666666667
-----------------------------------------
  Account       User  SUs (CPU Hours)  Percent of Total SUs
--------- ---------- ---------------- --------------------
      sam                      47132                0.0028
      sam   bmooreii             47130                0.0028
      sam   fangping                 1                0.0000
      sam      ketan                 0                0.0000
      sam    kimwong                 0                0.0000
```

Writing Job Scripts (`sbatch`)

- `sbatch` is the only non-wrapped command
- The arguments set useful environment variables for scripts

Option	Environment Variables
`--output`	–
`--time`[1]	–
`--job-name`	`SLURM_JOB_NAME`
`--nodes`	`SLURM_NNODES`
`--ntasks`	`SLURM_NTASKS`
`--cpus-per-task`	`SLURM_CPUS_PER_TASK`
`--ntasks-per-node`	`SLURM_NTASKS_PER_NODE`
`--partition`	`SLURM_JOB_PARTITION`
`--mem`	`SLURM_MEM_PER_NODE`
`--account`	`SLURM_JOB_ACCOUNT`

[1] Format: `DAYS-HOURS:MINUTES:SECONDS`

Slurm: Hello World

An example job:

```
#!/usr/bin/env bash
#SBATCH --job-name=hello
#SBATCH --output=hello.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
##SBATCH --partition=smp

echo "hello world"
```

Output:

```
hello world
```

What happens here?

Job:

```
1  #!/bin/bash
2  #SBATCH --job-name=hello
3  #SBATCH --output=hello.out
4  #SBATCH --nodes=1
5  #SBATCH --ntasks=6
6  ##SBATCH --partition=smp
7
8  srun echo "hello world"
```

What happens here?

Job:

```
#!/bin/bash
#SBATCH --job-name=hello
#SBATCH --output=hello.out
#SBATCH --nodes=1
#SBATCH --ntasks=6
##SBATCH --partition=smp

srun echo "hello world"
```

Output:

```
hello world
hello world
hello world
hello world
hello world
hello world
```

- Wait, what's `srun`?
  - Replaces `mpirun` or `prun`
  - Hardware and Slurm aware

Running a Tensorflow job on H2P

```
1 module purge
2 module load python/3.7.0 venv/wrap
```

- `module purge` clears your environment
- The `venv/wrap` module loads `virtualenvwrapper`
  - workon – lists virtual environments
  - `mkvirtualenv <name>` – makes a new virtual environment with `<name>`
  - `rmvirtualenv <name>` – remove a virtual environment with `<name>`
  - `workon <name>` – activate the virtual environment with `<name>`
  - `deactivate` – deactivate the current virtual environment

Installing Tensorflow into a Virtual Environment

- With Tensorflow 2.1.0, it is really easy to install CPU and GPU capable version

```
1 mkvirtualenv tf2
2 workon tf2
3 which pip
4 pip install tensorflow==2.1.0
```

- Before using Tensorflow on the GPU you will also need to `module load cuda/10.1`

A Basic Sequencial TF Example

```
1  import tensorflow as tf
2
3  mnist = tf.keras.datasets.mnist
4
5  (x_train, y_train), (x_valid, y_valid) = mnist.load_data()
6  x_train, x_valid = x_train / 255.0, x_valid / 255.0
7
8  model = tf.keras.models.Sequential(
9      [
10         tf.keras.layers.Flatten(input_shape=(28, 28)),
11         tf.keras.layers.Dense(128, activation="relu"),
12         tf.keras.layers.Dropout(0.2),
13         tf.keras.layers.Dense(10, activation="softmax"),
14     ]
15 )
16
17 model.compile(
18     optimizer="adam",
19     loss="sparse_categorical_crossentropy",
20     metrics=["accuracy"]
21 )
22
23 model.fit(x_train, y_train, epochs=5)
24 model.evaluate(x_valid, y_valid)
```

Submission Script

```
#!/usr/bin/env bash
#SBATCH --job-name=hello-tf
#SBATCH --output=hello-tf.out
#SBATCH --time=12:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --cluster=gpu
#SBATCH --partition=gtx1080
#SBATCH --gres=gpu:1

module purge
module load python/3.7.0 venv/wrap cuda/10.1
workon tf2

python hello-tf.py

crc-job-stats.py
```

Interactive Jobs

```
11:44:01  login0:~ crc-interactive.py -h
 crc-interactive.py -- An interactive Slurm helper
Usage:
    crc-interactive.py (-s | -g | -m | -i | -d) [-hvzo] [-t <time>] [-n <num-nodes>]
        [-p <partition>] [-c <num-cores>] [-u <num-gpus>] [-r <res-name>]
        [-b <memory>] [-a <account>] [-l <license>] [-f <feature>]

Positional Arguments:
    -s --smp                          Interactive job on smp cluster
    -g --gpu                          Interactive job on gpu cluster
    -m --mpi                          Interactive job on mpi cluster
    -i --invest                       Interactive job on invest cluster
    -d --htc                          Interactive job on htc cluster
Options:
    -h --help                         Print this screen and exit
    -v --version                      Print the version of crc-interactive.py
    -t --time <time>                  Run time in hours, 1 <= time <= 12 [default: 1]
    -n --num-nodes <num-nodes>        Number of nodes [default: 1]
    -p --partition <partition>        Specify non-default partition
    -c --num-cores <num-cores>        Number of cores per node [default: 1]
    -u --num-gpus <num-gpus>          Used with -g only, number of GPUs [default: 1]
    -r --reservation <res-name>       Specify a reservation name
    -b --mem <memory>                 Memory in GB
    -a --account <account>            Specify a non-default account
    -l --license <license>            Specify a license
    -f --feature <feature>            Specify a feature, e.g. `ti` for GPUs
    -z --print-command                Simply print the command to be run
    -o --openmp                       Run using OpenMP style submission
```

- Useful when debugging scripts, compiling code, etc

How to get help?

- Submit a ticket
- A good ticket:
  - Has the correct cluster selected
  - Provides the path to your submission script and output
  - Explains how to reproduce your problem
  - Provides details for what you have tried

Questions?