

SDSS and SRCC Computing Resources

Mark R. Yoder, Ph.D.

19 April 2023

Overview

Scope: Introductory, review of resources, basic HPC-foo

- Summary of Research Computing resources
- Documentation
- Basic HPC concepts
- SLURM Tips (especially in the Sherlock environment)
- Filesystems, inodes, and best practices

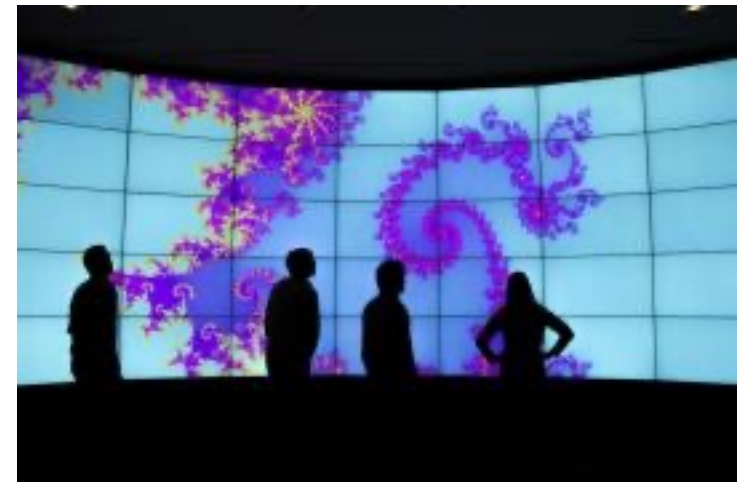
Organizations and acronyms:

- Stanford Doerr School of Sustainability (SDSS)
- SDSS-CC: SDSS Center for Computing (formerly known as CEES)
SDSS-CC provides a variety of high performance computing (HPC) resources to support the SDSS research community
- SRCC: Stanford Research Computing Center
SRCC provides HPC and other research computing support via a number of platforms, including the Sherlock HPC and Oak LUSTRE storage platforms. SRCC also provides Carina OnPrem and Cloud based resources for high risk data, Cloud computing consultation, and a variety of other services related to research computing. Recently, SDSS-CFC has partnered with SERC to provide expanded and enhanced computing resources to SE3 research teams.

Stanford Research Computing (SRCC)

“Your laptop is not for computing.”

- *“The Stanford Research Computing Center (SRCC) is a joint effort of the Dean of Research and University IT to build and support a comprehensive program to advance research at Stanford.”*
- Provides and manages a wide range of services and facilities
 - Sherlock (general purpose), Carina (high risk), Farmshare (education) HPC
 - Cloud resources and consulting
 - The HIVE visualization facility
 - Compute consulting services
 - Grant and CDMP writing assistance
- If it’s about “research” and “computing”, ask...



SDSS-CC Resources

- Sherlock *serc* partition (9040 CPUs, 88 v/a100 GPUs, up to 1 TB RAM, 128 CPUs)
 - **200 SH3-CBASE (AMD, 32 cores, 256GB RAM)**
 - **8 SH3-CPERF (AMD, 128 cores, 1TB RAM)**
 - **10 SH3-G86F64 (AMD, 128 cores, 8 x A100 GPU (6x40 GB, 4x80 GB), 1TB RAM)**
 - 12 SH2 base (Intel Skylake, 24 cores, 384 GB RAM)
 - 2 SH2 GPU (Intel Skylake, 24 cores, 4 x V100 GPU)
 - 1.35+PB Oak storage
- Sherlock Public partitions: *normal, dev, bigmem, gpu, owners*
- GCP
 - Excess capacity
 - Non-standard configuration projects, data gateways, etc.
 - Multi-institution collaborations
- Public HPC (ACCESS, National Labs, etc.)

SDSS-CC Resources: Sherlock

***** Please limit SERC jobs to 300 - 500 total concurrent CPUs! *****

- Request accounts: srcc-support@stanford.edu
- Primary compute platform for most users
- Single large, *serc* partition shared by most SDSS users; some PIs may also have private/group partitions
- Please be considerate to your friends and colleagues when queueing jobs.
 - Please restrict big, long-running jobs to ~300 CPUs
 - For lots of smaller, short running jobs, ~500 CPUs
 - For larger jobs or other questions, stop into Sh. Office Hours
- New workflows should probably start here

SDSS-CC Resources: Oak

- 1.35+ PB storage
- LUSTRE “Cheep and deep” storage
- Optimized for large volume and speed
- Performs poorly for lots of small files, so please tar, zip, or otherwise consolidate small files into HDF5 (or something)
- Some additional setup
- `/oak/stanford/schools/ees/{PI_SUNETID}`
- GCP:
 - Special cases; talk to Bob or Mark

SDSS-CC Resources: GCP

- Special cases, unique configurations, etc.
- Excess compute capacity
- Data gateways, crowd-sourcing projects
- Multi-institution collaborations
- Cloud based data sharing

SDSS-CC Resources: ACCESS, etc.

- We hope to increase our usage of National Labs, NSF, NASA, NOAA, etc. compute resources
 - Free!
 - Sometimes, agencies want you to use their platform
 - Can be a great way to run a ton of well defined jobs
- ACCESS (NSF) is especially easy to get small to medium size allocations
- SDSS-CC and SRC can help with proposals (for large allocations), renewal requests, etc.
 - Resource justification, scaling analysis, etc.

SDSS-CC and SRCC: Other stuff too

- RC Technical support and consultations
 - Sherlock Office Hours (Tu: 10:00, Th:3:00)
 - By appointment
- OnBoardings (first Wednesday of the Month, 1:00)
- Grant writing support
 - Code and Data Management Plan (CDMP)
 - Compute resources planning and budget

Documentation, support

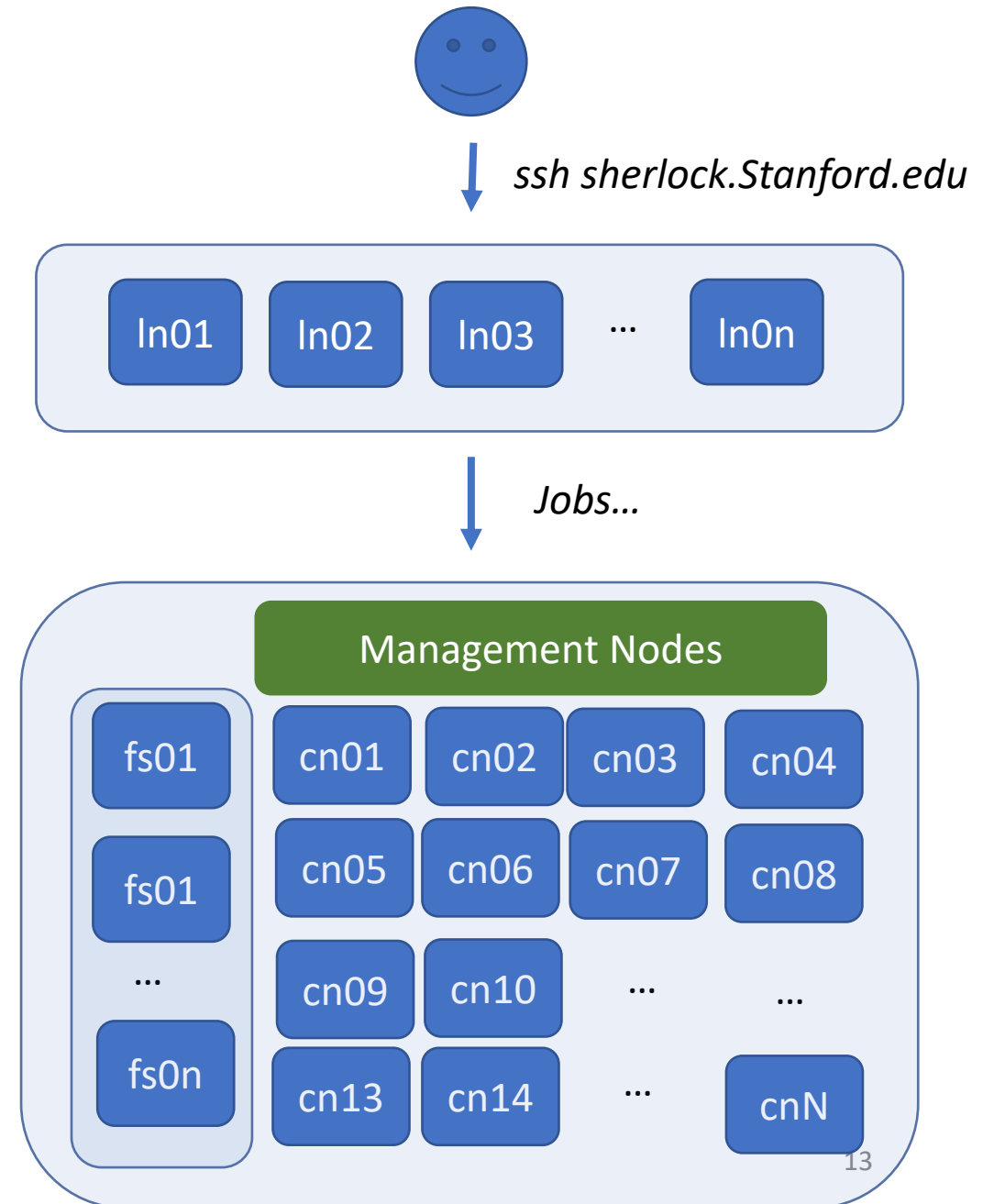
- The *Google*
- *SDSS-CC* Documentation: GitHub Pages
 - <https://stanford-rc.github.io/docs-earth/>
 - ** will be moving to something like *sdsscc-docs.stanford.edu*, so standby...
- Sherlock Documentation:
 - <https://www.sherlock.stanford.edu/docs/overview/introduction/>
- NOTE: These docs are searchable!
- NOTE Also: You can *Google* for these docs, eg “Stanford Sherlock docs”
- Occasional OnBoarding, Basic HPC, and other classes provided by *SRCC*
- Support requests:
 - *CEES* Slack channel
 - srcc-support@Stanford.edu

HPC Basics

- What is HPC?
- Basic architecture of HPC
- Nomenclature
- Connecting: ssh, Open on Demand (OoD)
- Running Jobs: Batch, Interactive OoD
- SLURM-foo
- Software on HPC (general), Sherlock
- Filesystems

HPC Basics: What is HPC?

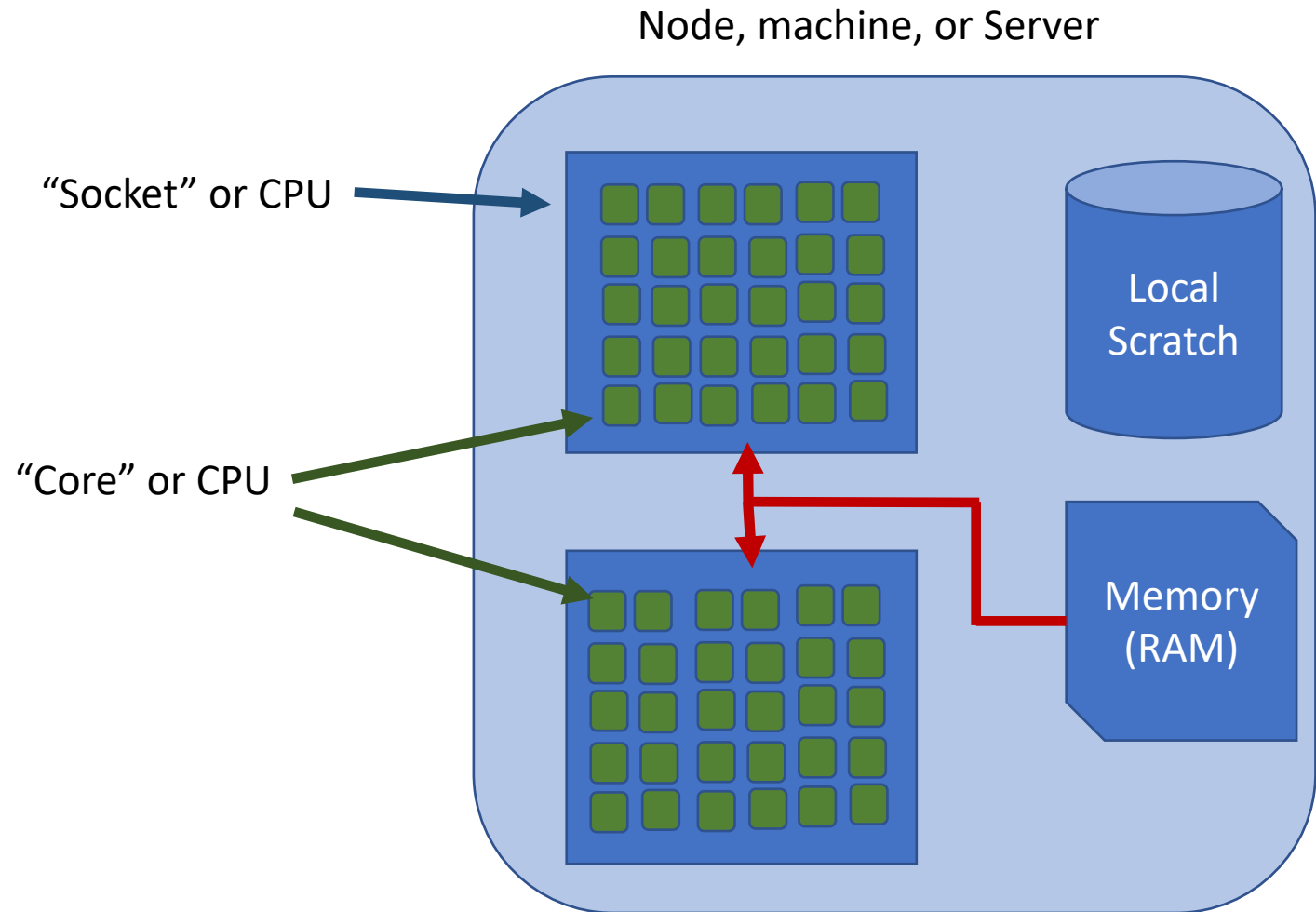
- *High Performance Computing*
- HPC is a bunch of computers:
 - Connected by high speed network
 - Sharing high speed filesystem(s)
 - And some management nodes
- Users access the login API/machines
- Request resources, to run jobs, via a Job Scheduler
- Scheduler and management nodes push compute jobs to the compute machines/nodes



The unfortunate state of HPC nomenclature:

What, exactly, is a “node”?

- The current state of HPC nomenclature is... unfortunate. In SLURM:
- “Node”: A machine; a server; a “computer”
- “Socket”: A traditional “CPU” device.
- Each “socket” or CPU will have many “cores” or CPUs.



HPC Scale: How big is a node?

- Sherlock 3.0 CBASE (standard) Node:
 - 1 socket x 32 cores/socket: 32 compute cores (or cpus)
 - 256 GB RAM (8 GB/core or cpu)
- Sherlock 3.0 CPERF (performance) Node:
 - 2 sockets x 64 cores/socket: 128 cpus
 - 1TB RAM (8GB/cpu)
- ~1 or 2 TB local storage (\$LSCRATCH)
- Most filesystems are mounted from designated FS servers.
- As a point of reference:
 - Your laptop (probably): 1 socket, 2-6 CPUs/cores, 16-32 GB RAM.
 - Many NSF HPCs are similar to Sherlock CPERF, except ~2GB/core

Connecting to Sherlock (also see docs)

- Connect:
 - Terminal or *nix CLI:
 - `$ ssh sherlock.stanford.edu`
 - To port graphics `$ssh -XY sherlock.stanford.edu`
 - Homepage:
<https://www.sherlock.stanford.edu>
 - OnDemand:
<https://login.sherlock.stanford.edu/>
 - 2-factor authentication required
 - Data transfer nodes:
 - `dtn.sherlock.stanford.edu`
 - `oak-dtn.stanford.edu`

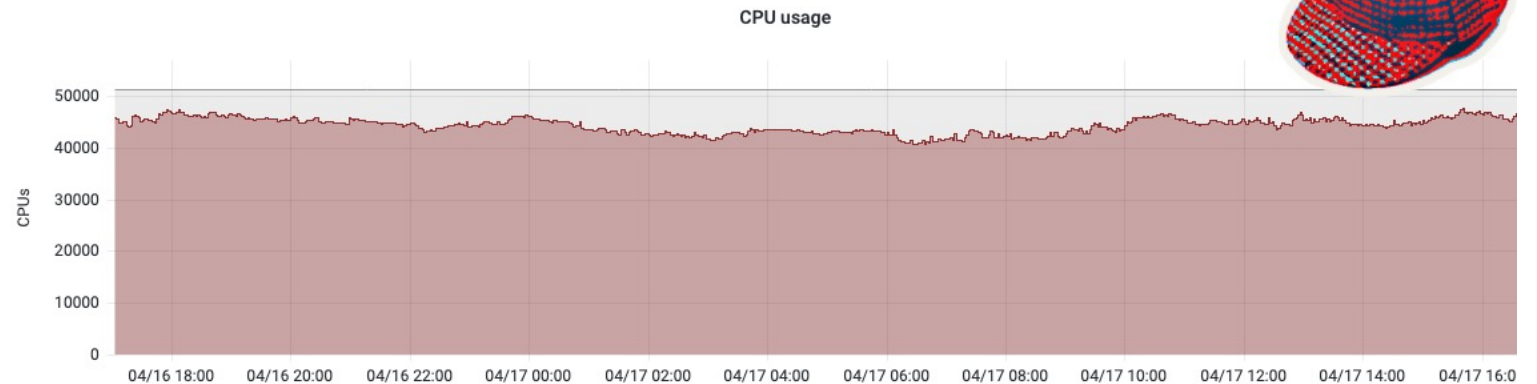
OnDemand and Web UI

login.sherlock.Stanford.edu

- Shell (terminal CLI)
- Upload files to Sherlock
- OnDemand applications
- Good for Windows users

Welcome to Sherlock OnDemand

An integrated, easy to use, and unified access point for Sherlock resources.



Getting Started

Sherlock OnDemand is your unified interface for access to Sherlock resources. Here, you can upload and download files, create, edit, submit, and monitor jobs, run applications, and connect via SSH, all via a web browser, with no client software to install and configure.

File manager

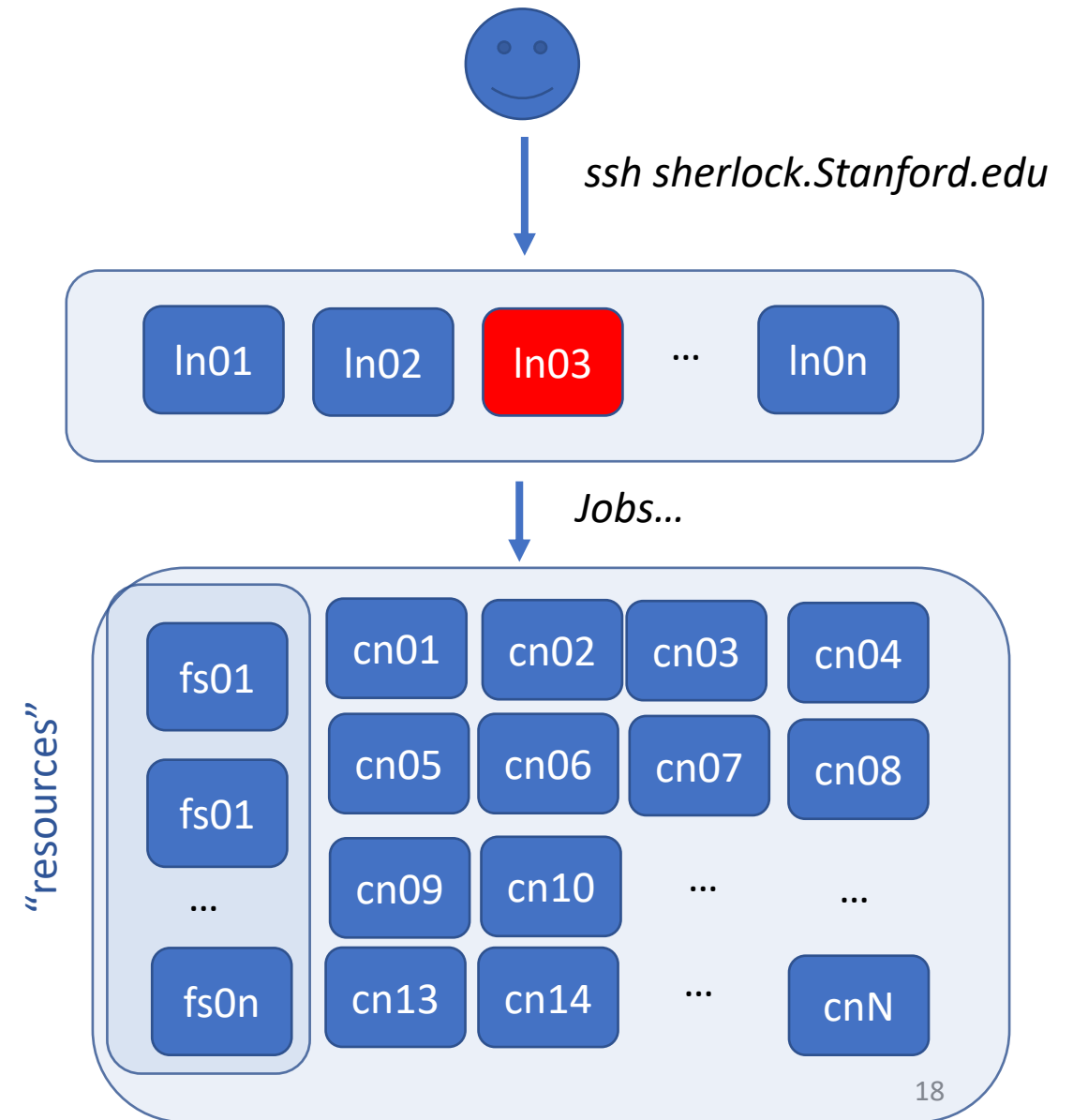
The web-based File Explorer can be used to upload and download files to your home or scratch directory, and copy, delete, rename, and edit files.

>_ Shell

Connect to Sherlock from your web browser! To open a terminal on Sherlock, select the "Clusters" menu and choose Sherlock.

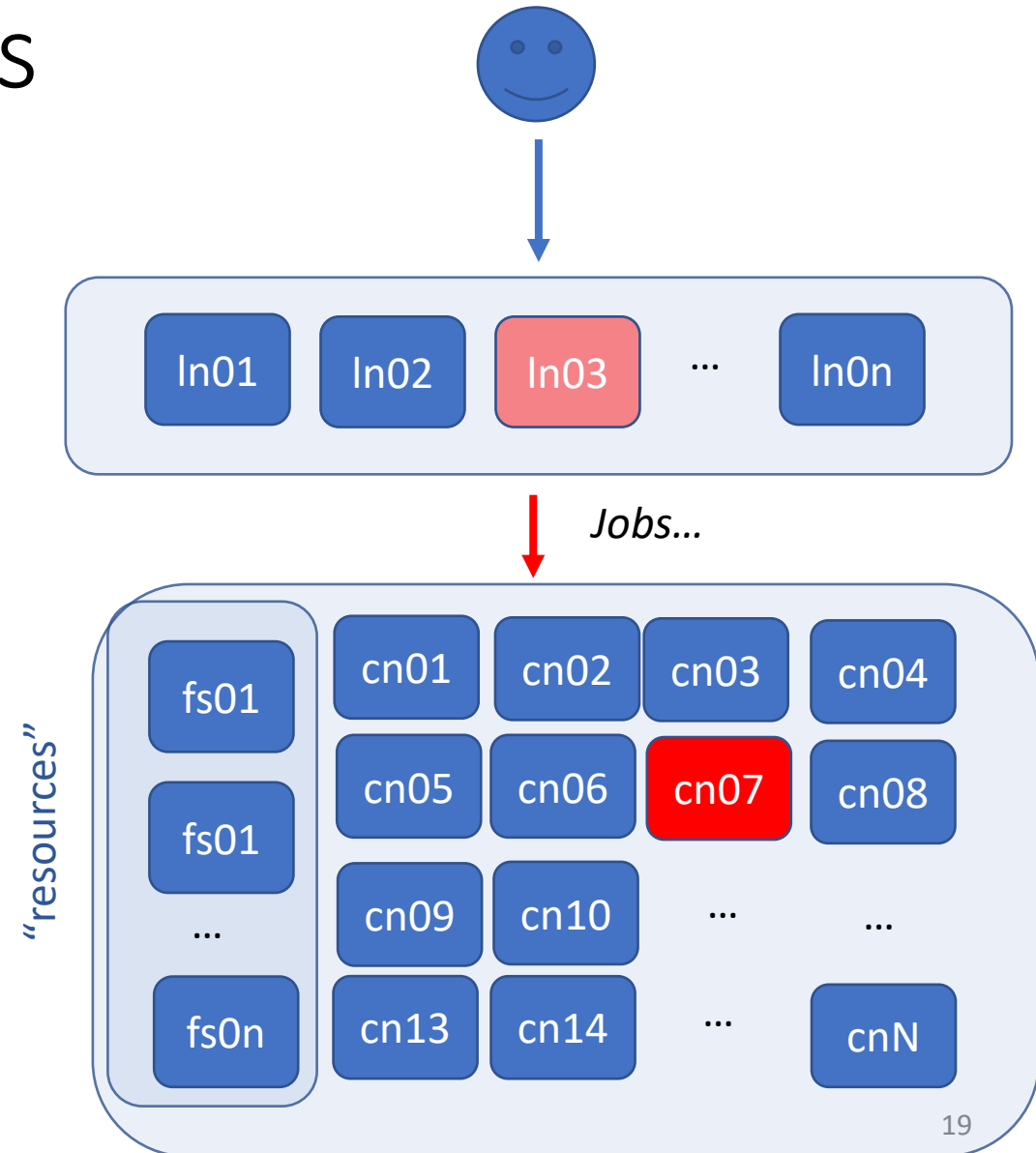
HPC Basics: Login nodes

- Login nodes are **NOT** for computing!
- Connect to HPC, request resources, pass jobs to compute nodes.
- Simple, light weight tasks:
 - Moderate file copy
 - Simple code compiles
 - Very small, lightweight test runs
- Since we own nodes, there are few good reasons to run jobs on LN.
- Over-subscribed (shared and busy)
- Quota managers will kill jobs
- Restarted frequently and without warning
- Generally, not a reliable test platform



HPC Basics: Interactive jobs

- Work in real-time
 - Like on the login node or on your laptop
 - But on a compute node
- Command line/shell, jupyter notebook, R-studio, MatLab, etc.
- Connect to login node, then request resources:
 - `srun -pty -partition=serc,normal bash`
 - `salloc -partition=serc,normal`
 - (You may then need to ssh to allocation)
- Must keep session connected
 - (You can use *screen* to maintain background tasks, but they can be reset)

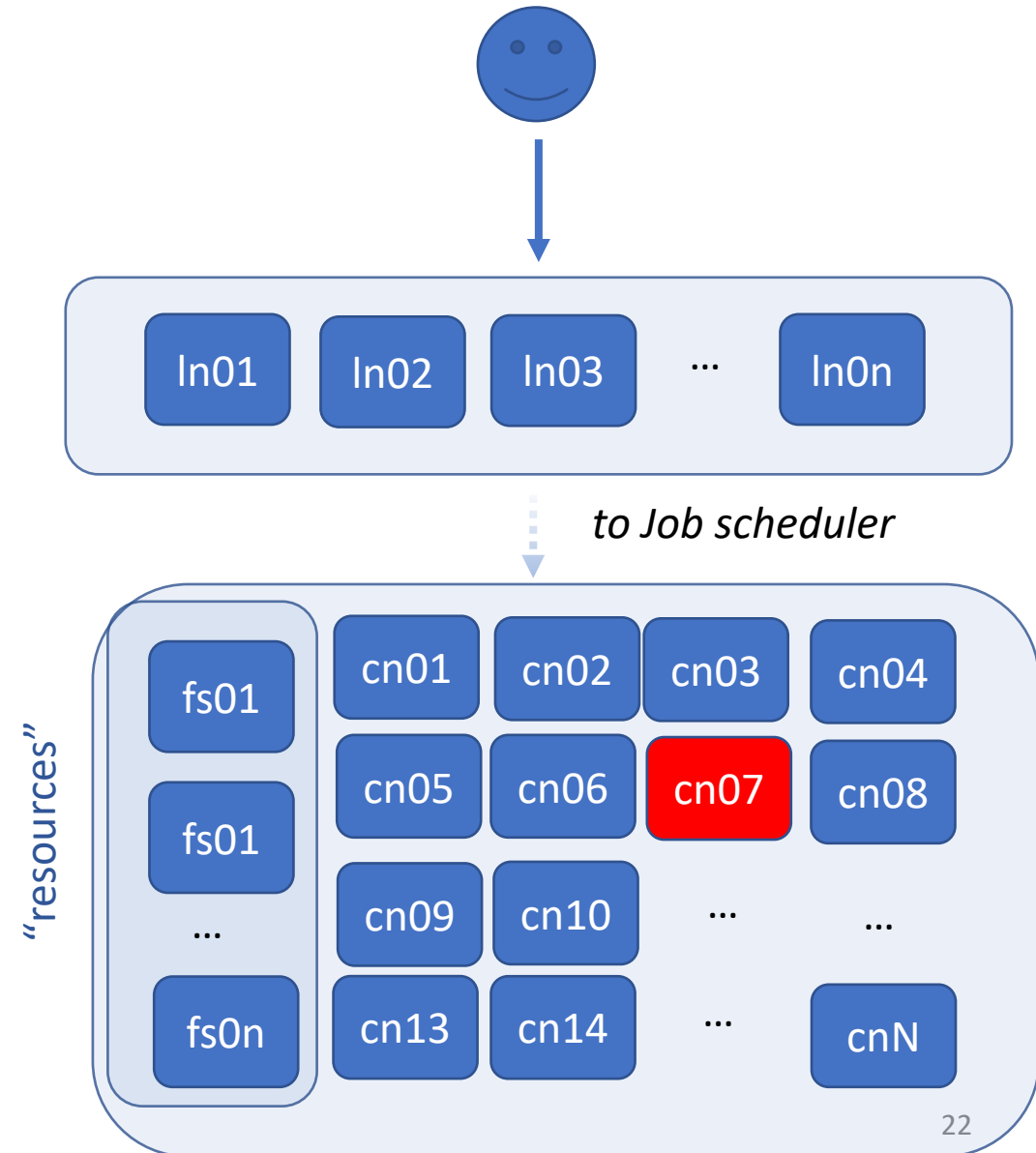


Running Jobs on Sherlock: Interactive

- Request resources (ie, you can't just log in and run stuff)
- Interactive shell sessions:
 - `$ sdev -p serc`
 - `$ srun -pty -partition=serc {other SLURM directives} bash`
 - `$ saloc -partition=serc {other SLURM directives}`
 - Note: *srun bash* and *saloc* are usually interchangeable
- Jupyter notebooks, R-Studio, MatLab from shell sessions or OnDemand (see docs)
- Sherlock OnDemand:
 - Connect: <https://login.sherlock.stanford.edu/>
 - Docs: <https://www.sherlock.stanford.edu/docs/user-guide/ondemand/>

HPC Basics: Batch jobs

- *Preferred modus operandi for HPC!*
- Submit job script to scheduler
- Scheduler will find and allocate resources, then run the job when resources are available
- Runs independently on compute nodes
- Job *std.out* and *std.err* output to file(s)
- Steeper learning curve, but very rewarding.
- Best for big, resource intensive jobs



Sherlock Partitions

- **Partitions** are groups of machines, designated a specific purpose or specific users, with distinct access rules.
- **serc partition is shared by all SDSS users**
- PI partitions: Some PIs have private partitions on Sherlock
- Public partitions:
 - *normal*: Default partition; heavily subscribed
 - *dev*: Restricted to interactive sessions.
 - *bigmem, gpu*: large memory (4TB) and (public) GPU nodes
- **owners**:
 - Virtual partition consists of all unassigned resources, available to all owners.
 - Jobs in *owners* will be *preempted* (killed) with a 30 second warning signal
 - Good for short jobs, Monte Carlo type jobs, well checkpointed tasks
 - At last assessment, preemption rate was about 2-3%, more or less time-independent
- **\$OAK storage**:
 - `/oak/stanford/schools/ees/{pi_sunet}`
 - `/oak/stanford/groups/{pi_sunet}`

Exercise 1: Interactive and batched jobs

1. Log in to Sherlock
2. Start an interactive session
3. Write the simplest Batch script
4. Execute the script as a test
5. Submit the script the scheduler
6. Review the output
7. Review job performance

NOTE: It will be up to the reader, as an additional exercise, to identify minor discrepancies, and assess their significance, in the instructions and the shown examples.

Exercise 1, Step 1: Log in and get session

- Log in (preferably via CLI terminal, for expediency)
- Get an interactive session. Either:
 - `srun -pty -partition=serc -mem-per-cpu=4g -time=01:00:00 bash`
 - `salloc -partition=serc -mem-per-cpu=4g -time=01:00:00 bash`

```
(base) [myoder96@sh02-ln04 login ~/toy_job]$ srun --pty --partition=serc  
--mem-per-cpu=4g --time=01:00:00 bash
```

```
srun: job 16715027 queued and waiting for resources  
srun: job 16715027 has been allocated resources
```

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $
```


Exercise 1, Step 2: Set up a working directory

- `$ mkdir test_job`
- `$ cd test_job`
- `$ vim test_job.sh`

Exercise 1, Step 3: Write your test script

```
#!/bin/bash
#
#SBATCH --job-name=toy-job
#SBATCH --partition=serc
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=4g
#SBATCH --output=toy_job.out
#SBATCH --error=toy_job.err
#SBATCH --time=00:15:00
#
module purge
module load gcc/12.1
#
echo "running a job for ${USER} on ${SLURM_CPUS_PER_TASK} on machine: $(hostname)"
sleep 10
exit 42
```

Exercise 1, Step 4: Execute toy_script.sh

```
(base) [myoder96@sh02-ln04 login ~/toy_job]$ chmod +x toy_job.sh
```

```
(base) [myoder96@sh02-ln04 login ~/toy_job]$ ./toy_job.sh
```

The following modules were not unloaded:

(Use "module --force purge" to unload all):

1) devel 2) math

“running a job for myoder96 on on machine: sh02-ln04.stanford.edu”

```
(base) [myoder96@sh02-ln04 login ~/toy_job]$
```

Exercise 1, Step 5: Success! Now batch script

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ sbatch --  
time=00:05:00 --partition=serc,normal --output=toy_job_%j.out  
toy_job.sh
```

Submitted batch job 16715319

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ queue -u  
$USER
```

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $  
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ sbatch --time=00:05:00 --output=toy_job_%j.out toy_job.sh  
Submitted batch job 16715319  
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ queue -u $USER
```

JOBID	ARRAY	NAME	USER	ACCOUNT	PARTITIO	NODES	CPUS	TIME	T
16715319	N/A	toy-job	myoder96	ruthm	serc	1	1	0:01	
16715027	N/A	bash	myoder96	ruthm	serc	1	1	9:47	

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ █
```

Exercise 1, Step 7: Monitor your job

- Use `squeue -u \$USER` to view your jobs

```
[(base) [myoder96@sh03-08n51 ~] (job 16911703) $ squeue -u $USER --Format=jobid:
JOBID      NAME          USER          ACCOUNT  PARTITION  NODES  CPUS  NODELIST
16911703   interactive  myoder96      ruthm    serc        1      4     sh03-08n51
(base) [myoder96@sh03-08n51 ~] (job 16911703) $
```

FROM A SHERLOCK SESSION, GO TO THAT NODE (SSH SH03-08N51)

- Use `ps` or `htop` to monitor activity

```
myoder96 - ssh sherlock.stanford.edu - 203x54
.../myoder96 - ssh sherlock.stanford.edu  /Users/myoder96 - -zsh  ...6 - myoder96@maz-login01:~ - -zsh  ... /Users/myoder96 - -zsh  ... - myoder96@maz-login01:~ - -zsh  ... +

0[|||||] 100.0% 8[|||||] 100.0% 16[|||||] 100.0% 24[|||||] 100.0%
1[|||||] 100.0% 9[|||||] 100.0% 17[|||||] 100.0% 25[|||||] 0.0%
2[|||||] 100.0% 10[|||||] 100.0% 18[|||||] 100.0% 26[|||||] 0.0%
3[|||||] 100.0% 11[|||||] 100.0% 19[|||||] 100.0% 27[|||||] 100.0%
4[|||||] 100.0% 12[|||||] 100.0% 20[|||||] 100.0% 28[|||||] 0.0%
5[|||||] 100.0% 13[|||||] 100.0% 21[|||||] 0.0% 29[|||||] 100.0%
6[|||||] 100.0% 14[|||||] 100.0% 22[|||||] 0.0% 30[|||||] 100.0%
7[|||||] 100.0% 15[|||||] 100.0% 23[|||||] 0.0% 31[|||||] 100.0%
Mem[|||||]
Swp[|||||]
7.31G/251G Tasks: 2, 0 thr; 1 running
0K/4.00G Load average: 26.08 26.03 26.05
Uptime: 62 days, 15:08:32

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%  TIME+  Command
13457 myoder96   20   0  112M  3684  1776 S  0.0  0.0  0:00.22 bash
14716 myoder96   20   0  116M  2800  1468 R  0.0  0.0  0:00.01 htop -u myoder96
```

Exercise 1, Step 7: Review output

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ cat  
toy_job_16715319.out
```

```
"running a job for myoder96 on 1 on machine: sh03-09n72.int"
```

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $ cat toy_job.err
```

```
The following modules were not unloaded:
```

```
(Use "module --force purge" to unload all):
```

```
1) devel 2) math
```

```
(base) [myoder96@sh03-09n67 ~/toy_job] (job 16715027) $
```

Exercise 1, Step 8: Review job performance

- Slurm Accounting, `sacct` records various data about your job
- <https://slurm.schedmd.com/sacct.html>
- Use `--format=` option to specify fields of interest
- Eg: `cpu_efficiency=totalCPU/(allocCPUs*elapsed)`

```
[(base) [myoder96@sh03-08n51 ~] (job 16911703) $ sacct --user=$USER --start=2023-04-17 --format=jobid,jobname,par
JobID          JobName      Partition  AllocCPUS  Elapsed    TotalCPU    MaxRSS      ReqMem      State      ExitCode
-----
16714857      -pty        normal     1          00:00:00   00:00:00    6400M      CANCELLED+  0:0
16714928      -pty        normal     1          00:00:03   00:00.017   6400M      FAILED      2:0
16714928.ex+  extern      normal     1          00:00:07   00:00.001   6400M      COMPLETED  0:0
16714928.0    -pty        normal     1          00:00:00   00:00.016   6400M      FAILED      2:0
16715027      bash        serc       1          00:14:53   00:01.775   4G         COMPLETED  0:0
16715027.ex+  extern      serc       1          00:14:53   00:00:00    96K        COMPLETED  0:0
16715027.0    bash        serc       1          00:14:52   00:01.774   6534K      COMPLETED  0:0
16715267      toy-job     serc       1          00:00:12   00:00.245   4G         FAILED      42:0
16715267.ba+  batch      serc       1          00:00:12   00:00.245   4G         FAILED      42:0
16715267.ex+  extern      serc       1          00:00:12   00:00:00    4G         COMPLETED  0:0
16715319      toy-job     serc       1          00:00:13   00:00.214   4G         FAILED      42:0
```

Some SLURM Basics

- `--ntasks`: Parallelization between nodes; independent instances of an app, nominally that communicate via something like MPI
- `--cpus-per-{something}`: Like it sounds. Note that some of these directives can conflict with one another.
- `--mem`: memory *per node*
- `--mem-per-{node, cpu, gpu}`: Determine the memory bottleneck and request memory to scale with that element
- NOTE: Default memory units are MB. Ask for GB (gp, g, G, etc. should all work, depending on SLURM configuration)
 - `--mem-per-cpu=8g`

--ntasks vs --cpus-per-task

```
(base) [myoder96@sh02-ln01 login ~]$ srun -p serc --ntasks=4 --nodes=4  
hostname
```

```
srun: job 16812329 queued and waiting for resources
```

```
srun: job 16812329 has been allocated resources
```

```
sh03-09n67.int
```

```
sh03-09n53.int
```

```
sh03-09n57.int
```

```
sh03-09n64.int
```

```
(base) [myoder96@sh02-ln01 login ~]$ srun -p serc --ntasks=1 --cpus-per-  
task=4 hostname
```

```
srun: job 16812419 queued and waiting for resources
```

```
srun: job 16812419 has been allocated resources
```

```
sh03-09n28.int
```

```
(base) [myoder96@sh02-ln01 login ~]$
```

SLURM Requests: Hardware constraints

(<https://slurm.schedmd.com/sbatch.html>)

- SERC partition includes multiple node configurations and HW architectures
- HW optimized codes, MPI programs, etc. should make HW specific requests using `-constraint= directive(s)`
- To show available constraints:
 - `$ sh_node_feat`
- Examples:
 - `$ sbatch -partition=serc ntasks={n} -constraint=CLASS:SH3_CBASE my_mpi_job.sh`
 - `$ sbatch -partition=serc ntasks={n} -constraint="[CLASS:SH3_CBASE | CLASS:SH3_CBASE.1 | CLASS:SH3_CPERF]"`
 - `my_mpi_job.sh`
 - `$ sbatch -partition=serc ntasks={n} -constraint=CPU_MNF:AMD my_amd_job.sh`

SERC Node features: --constraint=

```
(base) [myoder96@sh02-ln01 login ~]$ sh_node_feat -p serc  
CLASS:SH3_CBASE  
CLASS:SH3_CBASE.1  
CLASS:SH3_CPERF  
CLASS:SH3_G8TF64  
CLASS:SH3_G8TF64.1  
CPU_FRQ:2.00GHz  
CPU_FRQ:2.25GHz  
CPU_FRQ:2.30GHz  
CPU_FRQ:2.45GHz  
CPU_FRQ:2.50GHz  
CPU_FRQ:2.75GHz  
CPU_GEN:MLN  
CPU_GEN:RME
```

```
...  
CPU_GEN:SKX  
CPU_MNF:AMD  
CPU_MNF:INTEL  
CPU_SKU:5118  
CPU_SKU:7502  
CPU_SKU:7543  
CPU_SKU:7662  
CPU_SKU:7742  
CPU_SKU:7763  
GPU_BRD:TESLA  
GPU_CC:7.0  
GPU_CC:8.0
```

```
GPU_GEN:AMP  
GPU_CC:7.0  
GPU_CC:8.0  
GPU_GEN:AMP  
GPU_GEN:VLT  
GPU_MEM:32GB  
GPU_MEM:40GB  
GPU_MEM:80GB  
GPU_SKU:A100_SXM4  
GPU_SKU:V100_PCIE  
IB:EDR  
IB:HDR  
NO_GPU
```

SLURM Requests: Be specific

- Ask for *what* you want and how *how* you need it
- Be specific about memory, time, and core/node configuration. Will your request scale?
 - --time=HH:MM:SS
 - --mem-per-cpu, --mem-per-gpu, --mem-per-node (ie, --mem), etc.
 - --cpus-per-gpu
- How does your program use memory and CPUs?
- Default time for interactive sessions is 2 hours
- SBATCH docs: <https://slurm.schedmd.com/sbatch.html>

SLURM Examples (incomplete) for a 128 core job

- MPI program, parallelizes well by messaging; get resources as quickly as possible:
 - `-- ntasks=128 --constraint="[CLASS:SH3_CBASE | CLASS:SH3_CPERF]"`
- Runs via MPI (no OMP), but benefits fewer nodes:
 - `-- ntasks=128 --cpus-per-task=1 --constraint="[CLASS:SH3_CBASE | CLASS:SH3_CBASE.1 | CLASS:SH3_CPERF]" --nodes=2-8` ←
- Benefits well from OMP (threads) parallelization:
 - `-- ntasks=4 --cpus-per-task=32 --constraint="[CLASS:SH3_CBASE | CLASS:SH3_CBASE.1 | CLASS:SH3_CPERF]"`
- No MPI but good OMP (or other threaded) parallelization
 - `--ntasks=1 --cpus-per-task=128 --constraint=CLASS:SH3_CPERF`

More SLURM directive examples...

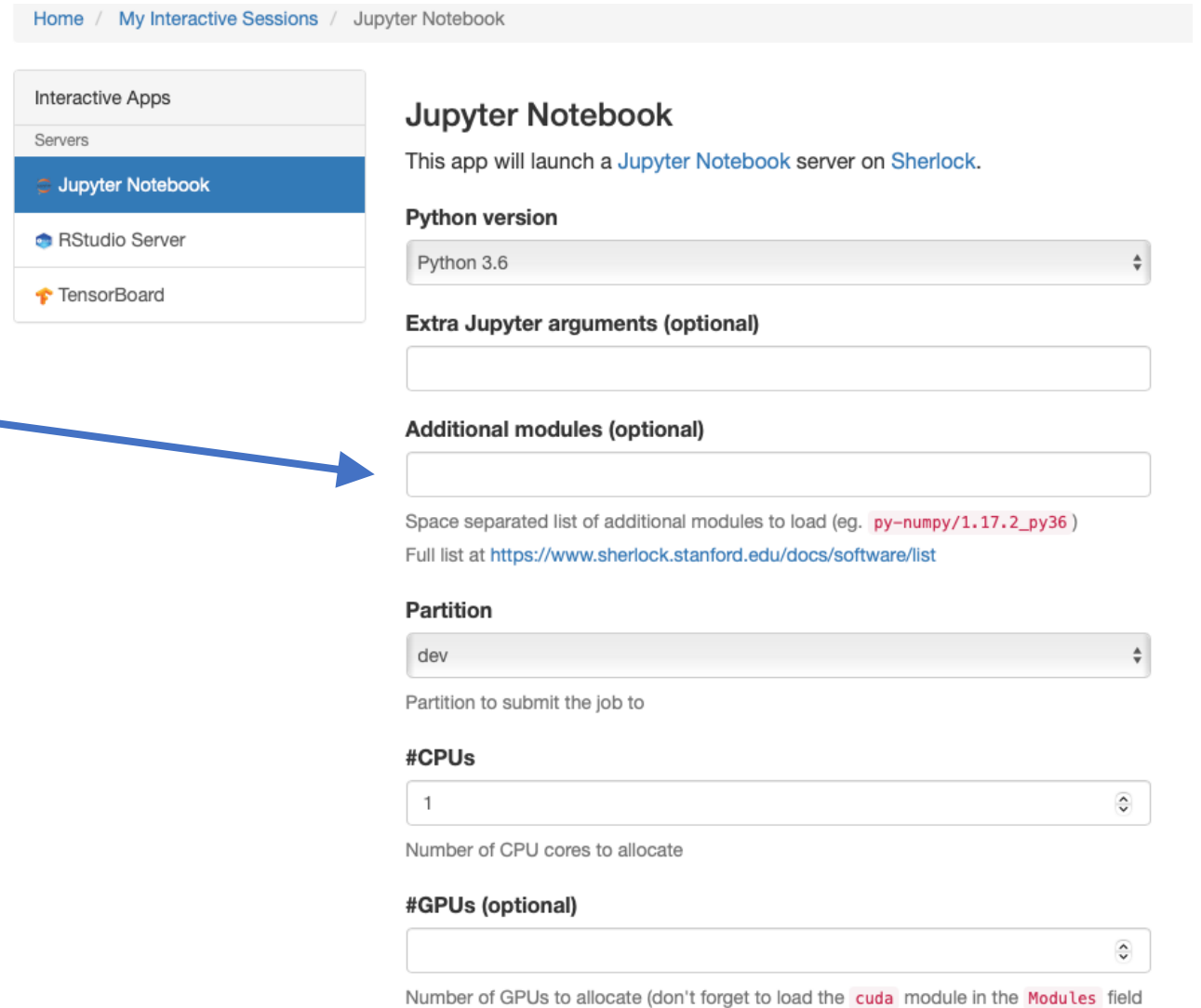
- Python job, using `multiprocessing`:
 - NOTE: This job is probably not HW sensitive, so consider framing it to use any *serc* hardware
 - `--ntasks=1 --cpus-per-task=24`
- Most GPU jobs will run on one node, as a single task.
- Sherlock has a public *gpu* partition, but ***our GPUs are in serc***
 - `--partition=serc --ntasks=1 --gpus=1 --cpus-per-gpu=8 --mem-per-cpu=8g`
 - NOTE: *serc* GPUs are 128 cores, 8 GPU (16 cpus/gpu) and 8GB/core, so this example is an under-ask.

Jupyter Notebooks

- Yes, you can run *Jupyter* Notebooks on Sherlock!
- Option 1: ssh and port forwarding (see documentation)
- Option 2: *Sherlock OnDemand*
 - Web based GUI interface
 - Connect: <https://login.sherlock.stanford.edu/>
 - Docs: <https://www.sherlock.stanford.edu/docs/user-guide/ondemand/>
 - OnDemand has come a *LONG* way – both on Sherlock and in general, in the past few years, further improvement is expected.

Jupyter notebooks...

- OnDemand will help you define SLURM directives...
- Test your module list in a terminal



The screenshot shows the OnDemand web interface for configuring a Jupyter Notebook session. The breadcrumb navigation at the top reads "Home / My Interactive Sessions / Jupyter Notebook". On the left, a sidebar menu lists "Interactive Apps" with sub-items "Servers", "Jupyter Notebook" (highlighted), "RStudio Server", and "TensorBoard". The main content area is titled "Jupyter Notebook" and contains the following configuration options:

- Python version:** A dropdown menu set to "Python 3.6".
- Extra Jupyter arguments (optional):** An empty text input field.
- Additional modules (optional):** An empty text input field. Below it, a note says "Space separated list of additional modules to load (eg. `py-numpy/1.17.2_py36`)" and provides a link to "https://www.sherlock.stanford.edu/docs/software/list".
- Partition:** A dropdown menu set to "dev". Below it, the text "Partition to submit the job to" is displayed.
- #CPUs:** A dropdown menu set to "1". Below it, the text "Number of CPU cores to allocate" is displayed.
- #GPUs (optional):** An empty dropdown menu. Below it, the text "Number of GPUs to allocate (don't forget to load the `cuda` module in the `Modules` field)" is displayed.

Software on Sherlock

- Sherlock SW stack and modules
 - Uses LMOD
 - Available Modules: module spider {something} or module avail
- Custom compiles:
 - \$GROUP_HOME is usually the best place
- SDSS and Custom builds:
 - Some corner-case or custom built SW modules are available via:
`/home/groups/s-ees/share/cees/modules/modulefiles`
 - If standard Sherlock SW is not working for you, put in a ticket and we'll figure out something.

Compiling SW on Sherlock

- Root access:
 - No, you do not have root access.
 - No, you cannot have root access. Please do not ask.
- Yes, you can still install most SW.
- Set installation directory with `-prefix` or `-DCMAKE_INSTALL_PREFIX`
- Choose your compiler!
 - The default compiler for CentOS-7 is `gcc@4.8.5`
 - `module load gcc/12.1.0`
 - `module load gcc/10.1.0`

Spack: Almost Magic!

- Spack is a dependency manager, SW installation platform
- Mostly out of LLNL
- LOTS of scientific computing SW already packaged up
- Also good for building environments with complex dependencies, then compile your obscure, but necessary, code
- Good starting place for containers

Containers

- Basically, a sandboxed disk space that contains all the libraries a given piece of SW needs to run
- Docker is the standard.
- No, we do not have docker
 - Requires root to build and run containers
- We have Singularity – Docker’s OpenSource knock-off.
 - Requires root to build, so containers cannot be built on of Sherlock
- Next gen OS should include AppTainer, which will allow users to build containers – without root!

Sherlock Filesystems: Flavors, limits, quotas

<https://www.sherlock.stanford.edu/docs/storage/overview/#quotas-and-limits>

- Use: `$ sh_quota`
- `$HOME` (15 GB): Small, backed up
 - `/home/users/{SUNetID}`
 - Small data files, small SW builds, your personal space
- `$GROUP_HOME` (1TB): Bigger! Backed up.
 - `/home/groups/{pi SUNetID}`
 - Shared data, specialized SW builds. Secure for your group.
- `$SCRATCH`, `$GROUP_SCRATCH` (100 TB each): FAST! Temporary
 - `/scratch/users/{SUNetID}`
 - `/scratch/groups{PI SUNetID}`
 - Fast. Temporary (90 day rolling purge). When feasible, do your IO here.
- `$L_SCRATCH`: Local (on-node) scratch; not shared. Very temporary
- `$OAK` (1 + PB): Storage
 - `/oak/stanford/schools/ees/{PI SUNetID}`
 - `/oak/stanford/groups/{PI SUNetID}` (if your PI has a group space)
 - Most of your data will go here
 - Shared containers and some SW

```
$ sh_quota
+-----+
| Disk usage for user kilian (group: ruthm) |
+-----+
| Filesystem | volume / limit | inodes / limit |
+-----+
| HOME | 9.4GB / 15.0GB [|||||] 62% | - / - ( -%) |
| GROUP_HOME | 562.6GB / 1.0TB [|||||] 56% | - / - ( -%) |
| SCRATCH | 65.0GB / 100.0TB [ 0%] | 143.8K / 50.0M ( 0%) |
| GROUP_SCRATCH | 172.2GB / 100.0TB [ 0%] | 53.4K / 50.0M ( 0%) |
| OAK | 30.8TB / 240.0TB [| 12%] | 6.6M / 36.0M ( 18%) |
+-----+
```

Summary

- Lots of compute resources in Sherlock!
- Understand your jobs and resource requirements; ask for what you need!
- *Please do not run jobs you do not understand!*
- Sherlock access and support:
 - srcc-support@Stanford.edu
 - SDSS-CC Docs: <https://stanford-rc.github.io/docs-earth/>
 - Sherlock Docs: <https://www.sherlock.stanford.edu/docs/overview/introduction/>
 - `$ ssh sherlock.stanford.edu`
 - Use batch and interactive jobs
 - Jupyter Notebooks: *Sherlock OnDemand*, ssh-forwarding