Scheduling Jobs on a Supercomputer

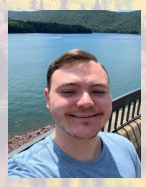
Meet the User Support Team

Layla
Freeborn

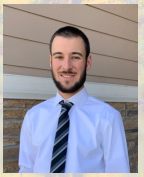
Brandon
Reyes

Andy
Monaghan

Michael
Schneider



John
Reiland



Dylan
Gottlieb

Mohal
Khandelwal

Ragan
Lee

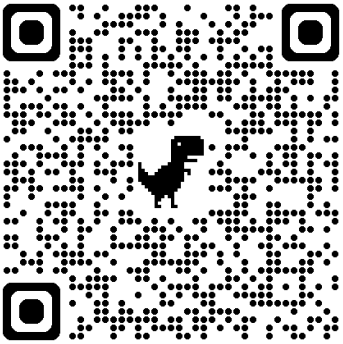
Research Computing
UNIVERSITY OF COLORADO BOULDER

2

Website: www.rc.colorado.edu

Documentation: https://curc.readthedocs.io

Helpdesk: rc-help@colorado.edu



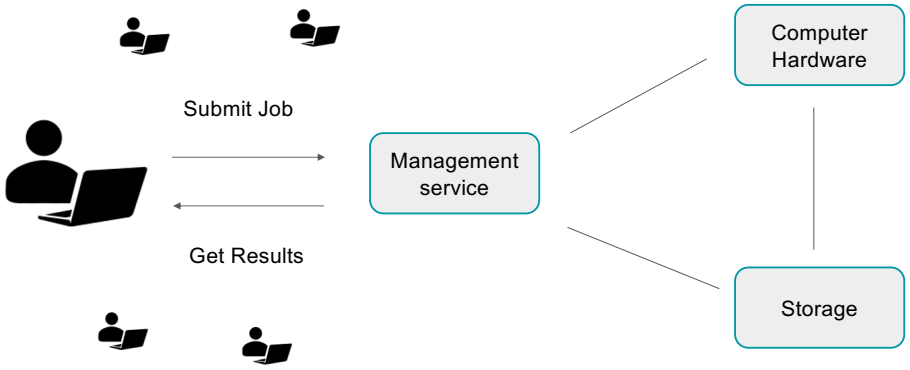
https://github.com/ResearchComputing/scheduling_a_job_on_a_supercomputer_rc_shortcourse

Session Overview

- Overview of Jobs Submissions
- Interactive Jobs
- Job Directives
- Batch Jobs
- Checking/monitoring jobs

Remember, when you login to the HPC system you are put on a login node. You need to then gain access to a compute node to run software.

General Overview of Job Submission

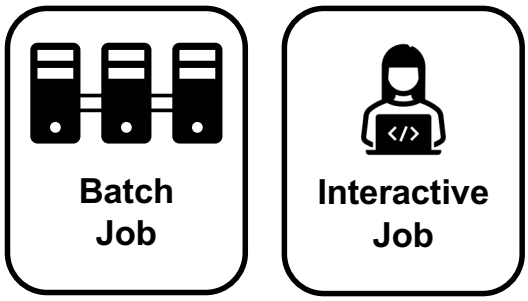


SLURM – Job Scheduler

- **S**imple **L**inux **U**tility for
  Resource **M**anagement
- Allocates compute resources to users through Jobs
- 2 Types of Jobs
  - **Batch Jobs**
  - **Interactive Jobs**

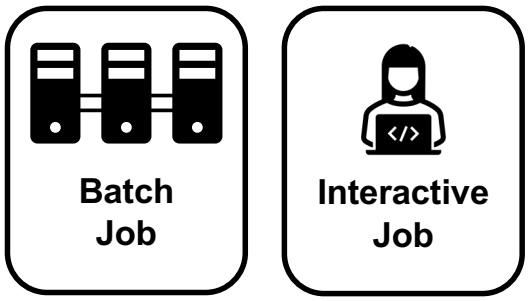
Because our clusters are shared resources with many users trying to utilize available compute with their applications, we need a system to divide compute in a simple and fair system

Batch vs Interactive



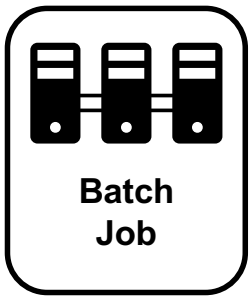
Batch vs Interactive

- Runs a job script
- No need for user input
- Best for tested workflows and long/big jobs



Batch vs Interactive

- Runs a job script
- No need for user input
- Best for tested workflows and long/big jobs



Batch Job

Interactive Job

- Requires active user input
- Terminal or GUI
- Best for development and exploration

Interactive jobs

acompile

- Quick access to resources
- Limit of 4 CPUs for up to 12 hours
- Focused on compiling/building software and small workflows

`sinteractive`

- Request any partition, more resources, and longer walltime
- Subject to standard wait times (not immediate access)
- Focused on GUI-based software

Running an interactive job

- Here we will run a Python script using an interactive job
- First request resources:

```
$ sinteractive --partition=atesting --qos=testing --time=00:10:00 --ntasks=1
```

- When the job starts you will be put on a compute node and you can execute your commands:

```
$ module load anaconda
$ python my_very_cool_script.py
```

- To quit:

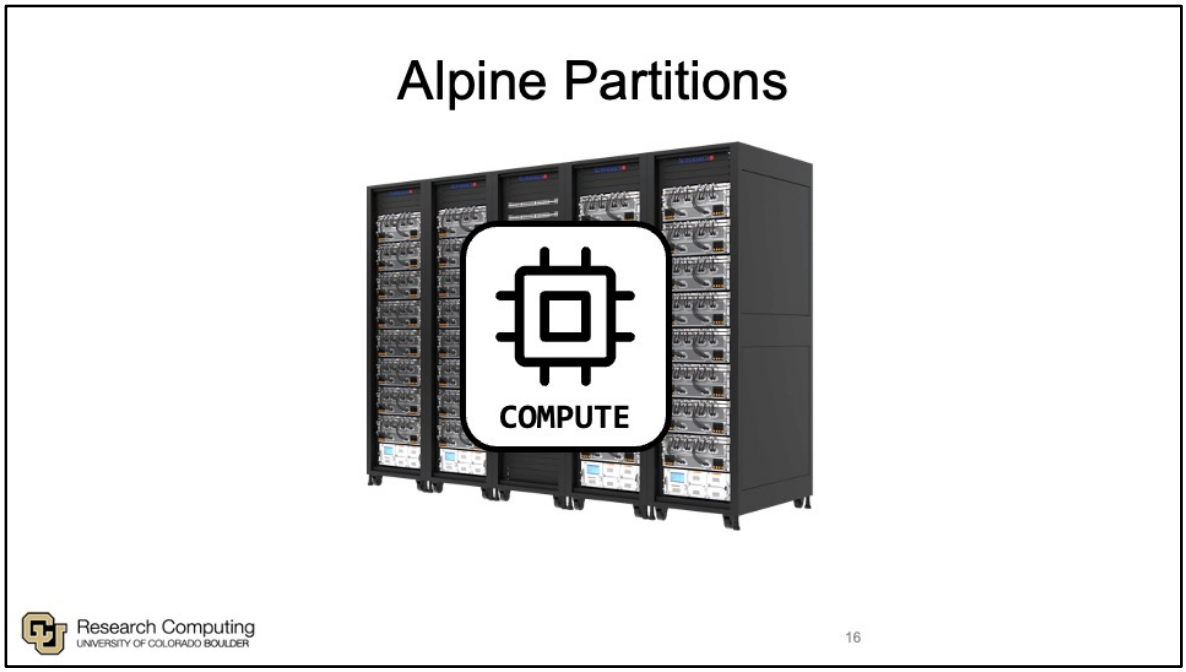
```
$ exit
```

Common directives

- **Partition – A collection of compute nodes**
  - `--partition=<partition_name>`
- **Quality of service (QoS) – System defined constraints for a job (more on this later!)**
  - `--qos=<qos>`
- **Allocation – Account to "charge to"**
  - `--account=<account_name>`
- **Number of nodes to run on**
  - `--nodes=<nodes>`
- **Number of cores to run on**
  - `--ntasks=<number-of-tasks>`

Common directives

- **Wall time – How long you want to run on these resources**
  - `--time=<wall time>`
- **Job name**
  - `--job-name=<jobname>`
- **Output – Where all output that would be written to the terminal should go**
  - `--output=<name>`
- **Send an email when events happen in the job**
  - `--mail-type=<type>`
- **Email address to send updates to**
  - `--mail-user=<user>`

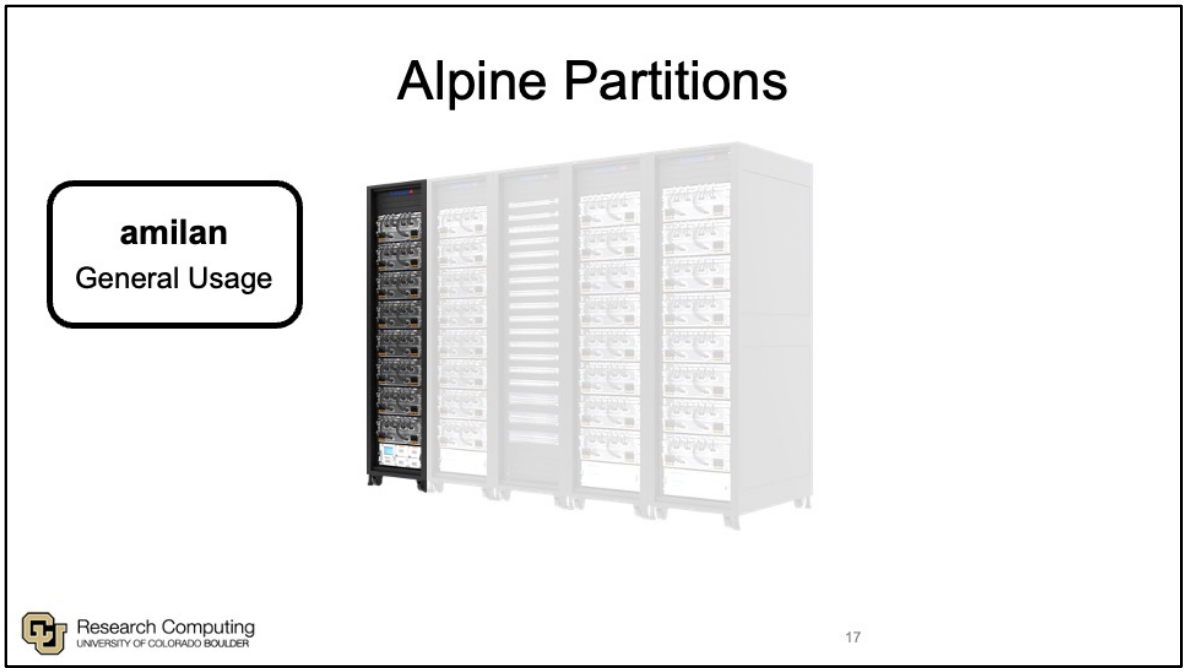


Compute nodes are organized into partitions – logical groups of similarly spec'd hardware.

Partitions make it easier to request specific resources and ensure a more consistent user experience.

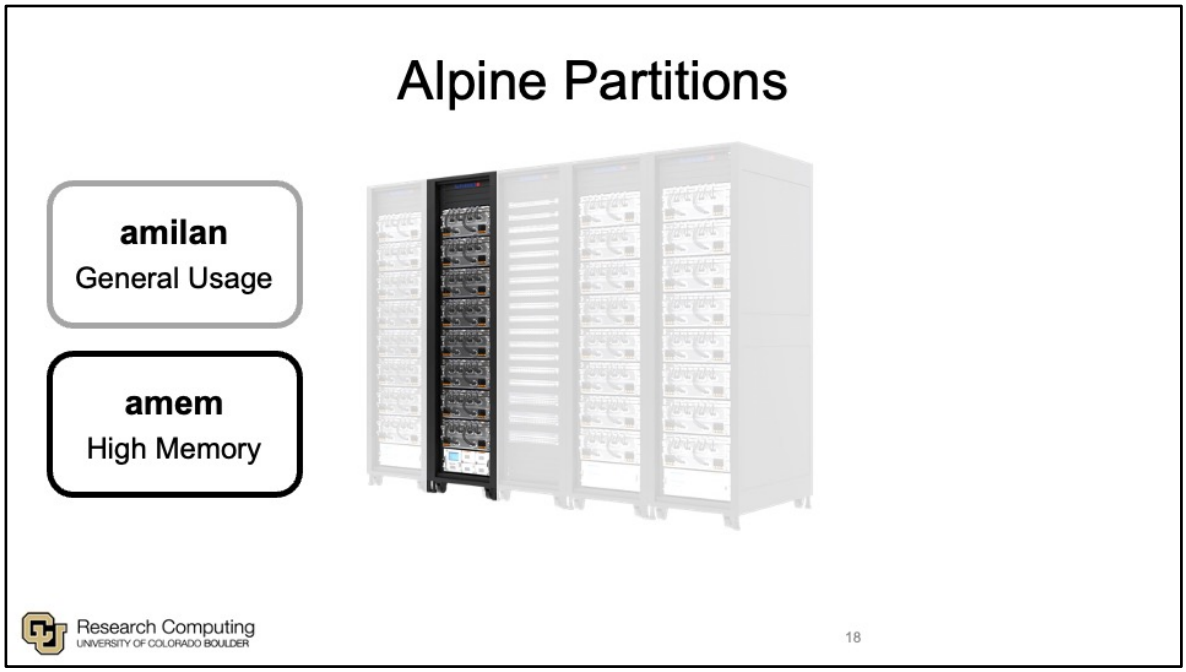
The Alpine Cluster has four primary partitions.

Detailed Information on Alpine Hardware:
https://curc.readthedocs.io/en/latest/clusters/alpine/alpine-hardware.html



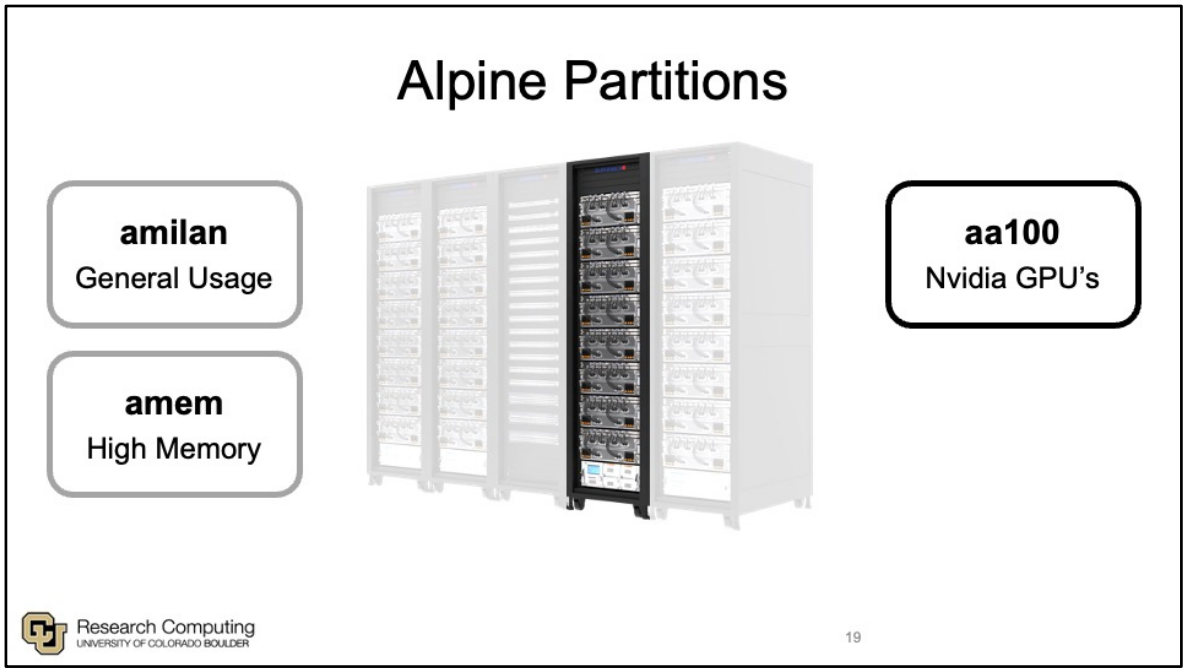
amilan:
285 Nodes
64 CPU Cores per Node
3.75 GB of RAM per CPU (240 GB total)

Detailed Information on Alpine Hardware:
https://curc.readthedocs.io/en/latest/clusters/alpine/alpine-hardware.html

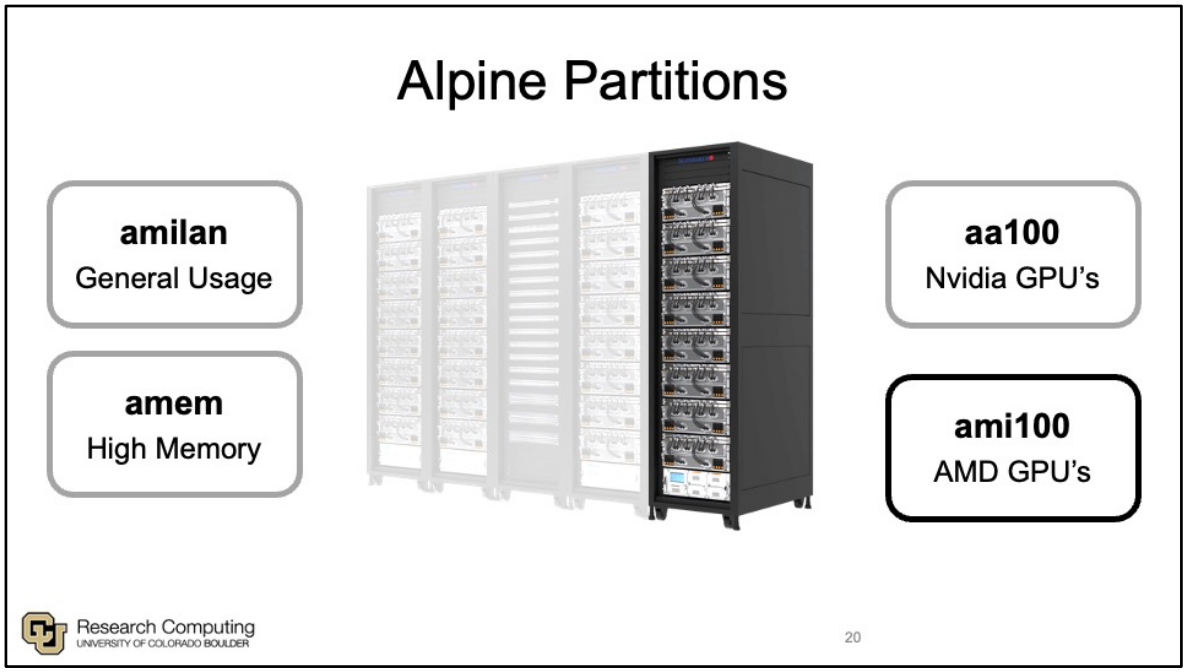


amem:
22 Nodes
48 CPU Cores per node (12)
64 CPU Cores per node (10)
21.5 GB of RAM per CPU (1 TB total)

Detailed Information on Alpine Hardware:
https://curc.readthedocs.io/en/latest/clusters/alpine/alpine-hardware.html



aa100
12 Nodes
64 CPU Cores per Node
3.75 GB of RAM per CPU (240 GB total)
3 GPU's per Node (Nvidia A100)



ami100
8 Nodes
64 CPU Cores per Node
3.75 GB of RAM per CPU (240 GB total)
3 GPU's per Node (AMD MI100)

Detailed Information on Alpine Hardware:
https://curc.readthedocs.io/en/latest/clusters/alpine/alpine-hardware.html

Quality of Service (QoS)

QoS	Description	Max wall time	Max jobs/user	Max nodes/user
normal	Default QoS	24 H	1000	128
long	For jobs needing longer wall times	7 D	200	20
mem	High-memory jobs	7 D	n/a	12
testing	For jobs submitted to testing partitions	1 H	1	n/a

Batch Jobs

- **Batch Jobs** are jobs you submit to the scheduler that are run later without supervision
- A job script is simply a script that includes **SLURM directives** (resource specifics) ahead of any commands.

Anatomy of a job script

```
#!/bin/bash

## Directives
#SBATCH --<option>=<value>

## Software
module load <software>

## User scripting
<command>
```

Anatomy of a job script

```
#!/bin/bash

## Directives
#SBATCH --<option>=<value>

## Software
module load <software>

## User scripting
<command>
```

Directives in a job script

```
#SBATCH --<option>=<value>
```

Example job script

```
#!/bin/bash

## Directives
#SBATCH --ntasks=1                    # Number of requested tasks/cores
#SBATCH --time=0:01:00                # Max run time
#SBATCH --partition=amilan            # Specify Alpine CPU node
#SBATCH --output=test_%j.out          # Rename standard output file

## Software
module purge                          # Purge all existing modules

## User commands
echo "This is a test of user $USER"
```


Output files are important for debugging and verifying a job has completed successfully. Always include a --output directive!

The output file is created in directory job was run unless specified in your --output directive.

If the directive --output is not provided, then a generic file name will be used (slurm_xxxxxx.out).

Software job script example

```
#!/bin/bash

## Directives
#SBATCH --ntasks=1                    # Number of requested tasks/cores
#SBATCH --time=0:01:00                # Max run time
#SBATCH --partition=amilan            # Specify Alpine CPU node
#SBATCH --output=test_%j.out          # Rename standard output file

## Software
module purge                           # Purge all existing modules
module load anaconda                   # Load Anaconda
conda activate <my-conda-environment>  # Activate CONDA environment

## Run Python script
python my_cool_script.py
```


Software can be loaded using the software module system or by accessing user installed software in their /projects directory.

https://curc.readthedocs.io/en/latest/compute/modules.html

Be careful not to run GUI or other software that requires user interaction!

Submitting a Job script

`sbatch <job_file> <other-directives>`

- Command to submit a job to the SLURM job scheduler

Submitting a Job script

`sbatch <job_file> <other-directives>`

- Command to submit a job to the SLURM job scheduler
- If we created the job script “my_first_job.sh” then we would submit it as follows:

```
sbatch /path/to/my_first_job.sh
```

Submitting a Job script

`sbatch <job_file> <other-directives>`

- Command to submit a job to the SLURM job scheduler
- If we created the job script “my_first_job.sh” then we would submit it as follows:

```
sbatch /path/to/my_first_job.sh
```

- Modify slurm directives in the batch script:

```
sbatch /path/to/my_first_job.sh --ntasks=12
```

Checking your jobs

- squeue: Monitor your jobs status **in queue and while running**:
  - By default, shows all jobs in queue can specify using:

```
$ squeue -u <username>
$ squeue -p <partition>
```

- sacct: Check back on usage statistics of **previous Jobs**
  - By default, only checks all jobs from the start of the current day can specify using:

```
$ sacct -u <username>
$ sacct --start=MM/DD/YY -u <username>
$ sacct -j <job-id>
```

Checking your jobs

- Another method of checking details of your job while running is with `scontrol`
  - Advanced command usually used by system administrators, but you can use it too!

```
$ scontrol show job <job number>
```

- To check the percentage of CPU and memory usage of a job **after it completes,** use seff

```
$ module load slurmtools
$ seff <job number>
```

Thank you!

- **Documentation**: curc.readthedocs.io/
- **Trainings with Center for Research Data and Digital Scholarship (CRDDS)**: https://www.colorado.edu/crdds/
- **Helpdesk**: rc-help@colorado.edu
- **Consult Hours** **(Tuesday 12:00-1:00 in-person, Thursday 1:00-2:00 virtually)**

Survey and feedback

https://tinyurl.com/curc-survey18