


WEEK 3 – Job Submission on the Digital Research Alliance of Canada Servers

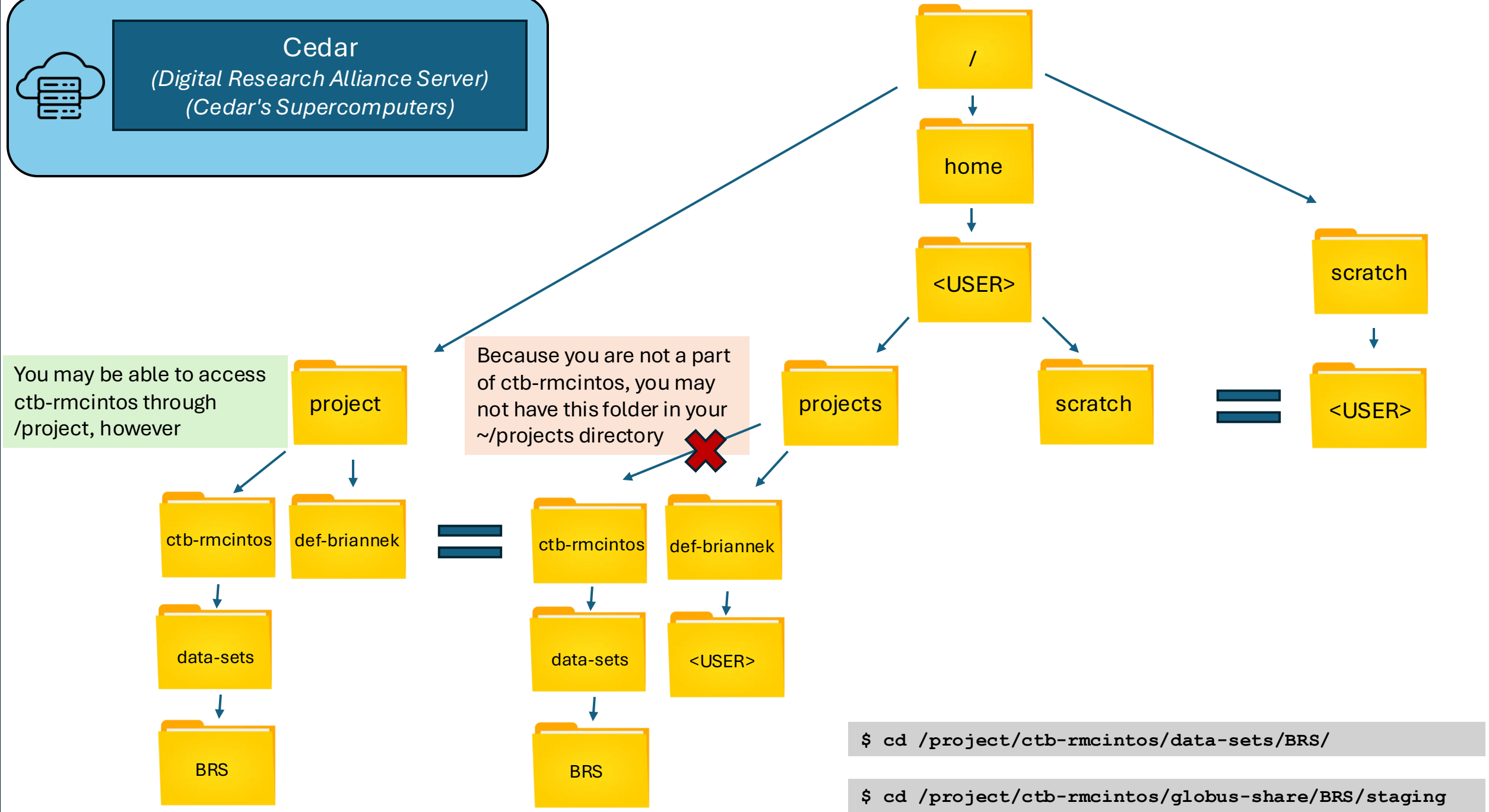
Session Details

- Interactive
 - you'll need to be on a computer to follow along
- Make sure you have your login info for Digital Research Alliance (Compute Canada)
 - Should be the same credentials you use to log into <https://ccdb.alliancecan.ca/security/login>
 - Also have your MFA device ready
- Feel free to interrupt or leave a message in the chat if you have questions, need something repeated, or are having trouble with any steps
- This session will be recorded – but only for internal use, within the BRS



Cedar

(Digital Research Alliance Server)
(Cedar's Supercomputers)



Getting started: JupyterHub

JupyterHub for Cedar:







<https://jupyterhub.cedar.alliancecan.ca/>

Documentation:

<https://docs.alliancecan.ca/wiki/JupyterHub>

JupyterHub on clusters

On the following clusters[†], use your Alliance username and password to connect to JupyterHub:

JupyterHub	Comments
Béluga 	Provides access to JupyterLab servers spawned through jobs on the Béluga cluster.
Cedar 	Provides access to JupyterLab servers spawned through jobs on the Cedar cluster.
Narval 	Provides access to JupyterLab servers spawned through jobs on the Narval cluster.
Niagara 	Provides access to JupyterLab as one of the applications of the SciNet Open OnDemand portal. To learn more, see the wiki page  .
Graham 	Provides access to JupyterLab servers spawned through jobs on the Graham cluster.

Server Options

Reservation

None

Account

def-_cpu

Time (hours)

2.0

Number of cores

1

Memory (MB)

2375

☐ **Enable core oversubscription?** Recommended for interactive usage

GPU configuration

None

User interface

JupyterLab

Start

Log in!

- **Open a terminal**
- **Login to cedar**

```
$ ssh YOURUSERNAME@cedar.computecanada.ca
```

Update ["pull"] the newest copy of the session codebase ["repository"]

- Navigate to your scratch directory
 - `cd scratch/`
- "Clone" the repository
 - This creates a copy of the code from the website onto your scratch
 - `git clone [path]`
- "Pull" the repository
 - This copies the newest version of the repository onto your scratch
 - `cd ~/scratch/BRS_training`
 - `git pull`

Errors? Don't worry

```
[jwangbay@login2 BRS_Training]$ git pull
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 31 (delta 16), reused 13 (delta 6), pack-reused 0 (from 0)
Unpacking objects: 100% (31/31), 109.96 KiB | 1.57 MiB/s, done.
From https://github.com/INN-SFU/BRS_Training
   44ee6ca..f70c0fc  main       -> origin/main
Updating 44ee6ca..f70c0fc
error: Your local changes to the following files would be overwritten by merge:
       Training_Session_Notebook.ipynb
Please commit your changes or stash them before you merge.
error: The following untracked working tree files would be overwritten by merge:
       .ipynb_checkpoints/Training_Session_Notebook-checkpoint.ipynb
Please move or remove them before you merge.
Aborting
```

- If you have important changes you've made to your copy of **BRS_Training**

- `cd ~/scratch; mv BRS_Training my_BRS_Training; git clone https://github.com/INN-SFU/BRS_Training.git; cd BRS_Training`

- Otherwise

- `git stash; git pull`

Code catchup!

- If you haven't been able to follow the Week 2 Tutorial
 - Please try on your own time!
 - To follow along this week, run
 - `cp -R /project/ctb-rmcintos/jwangbay/DO_NOT_USE_THIS_OR_jwangbay_OUTSIDE_OF_TRAINING/data-sets/BRS_subset/ ~/scratch`
 - This simulate you having run all the Week 2 code

Outline

- Using JupyterHub on cedar
- Working with data on cedar
 - Extracting subsets of data (e.g. participants with GAD/BDI)
 - Loading in tabular data
 - Copying subsets of data from projects to scratch
 - Generating a new summary score for a given participant
 - Visualizing tabular data (e.g. printing tables, quick plots)
 - Simple analyses (e.g., correlations)
- Running scripts from command line
- Running jobs on cedar/Fir
- Running "array" jobs on cedar/Fir
- Running interactive jobs

Running scripts

- **Open a terminal**
- **Login to cedar**

```
$ ssh YOURUSERNAME@cedar.computecanada.ca
```

- **Navigate to scratch**

```
$ cd ~/scratch/BRS_Training
```

- **Load in any required packages**

```
$ module load scipy-stack/
```

Some Helpful Tips for Creating Scripts

- **Create a "header" that describes:**

- What the script does
- Inputs needed
- Outputs produced

- **Add comments**

- Comment the "why" - explain the reasoning, not just actions
- Add comments at different steps for tracking progress
- Provide error messages for when something goes wrong

- **Keep it modular**

- Small, single-purpose functions

- **Use descriptive variable and function names**

```
"""
Script Name: calculate_average.py
Author: Leanne
Date: 2025-08-13

Description:
    This script calculates the average (mean) of a list of numbers
    from a text file.

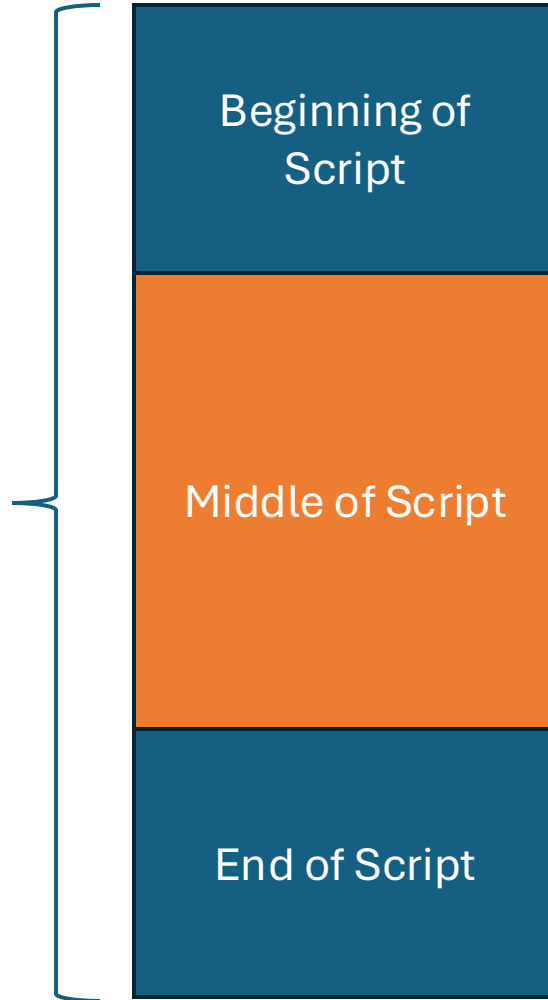
Inputs:
    - A text file containing one number per line (e.g., numbers.txt)

Outputs:
    - Prints the average value to the screen

Usage:
    python calculate_average.py numbers.txt
"""
```

What do we do when our scripts take a long time? Submit a job!

If this script takes, for instance, 5 hours to run, it should be a job



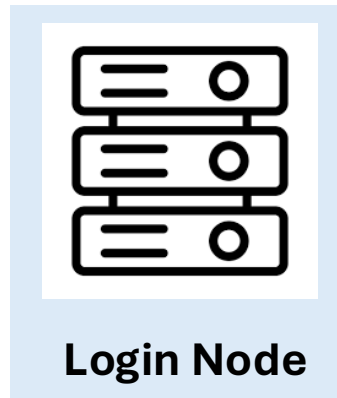
- Long jobs
- Jobs using a lot of memory
- Jobs using GPUs

Nodes

You usually interact with this node



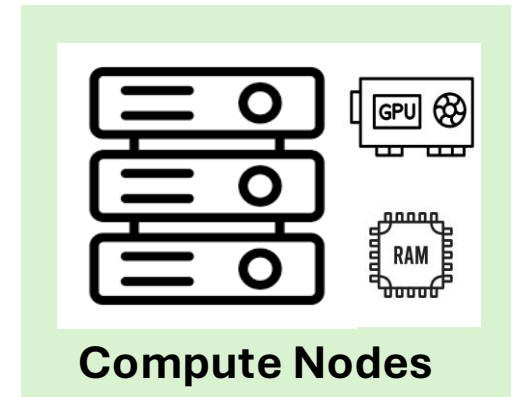
Login with `ssh`



*e.g. cedar1, cedar2,
cedar5, login1, login2, ...*

Job gets submitted with `sbatch`

The compute node runs the submitted script.
With `sbatch`, you don't interact with it directly.



e.g. cdr4235, cdr0292, ...

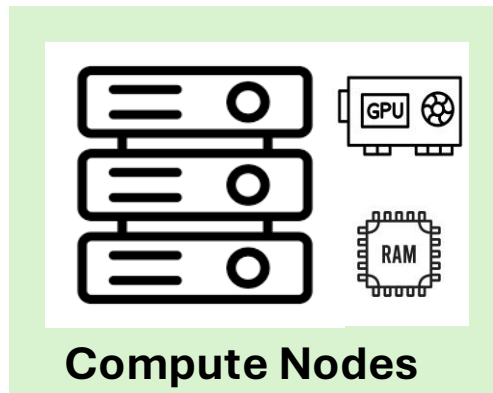
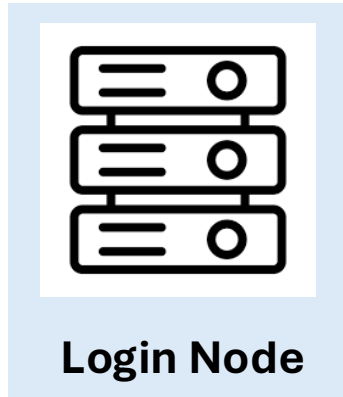
For quick or non-computationally intensive tasks.

- copying or moving files
- inspecting file contents
- scripts < 15 min, no GPU, and low memory

For long and/or many tasks.

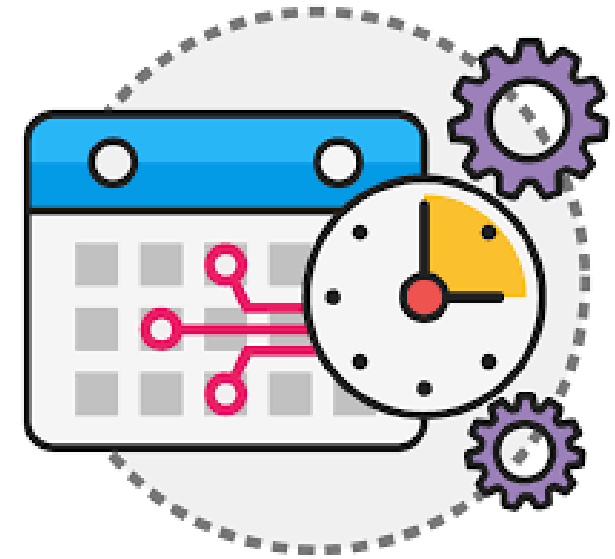
- Scripts > 15min
- Scripts using GPUs
- Scripts using a lot of memory

What happens when you run **sbatch**?



- **SLURM**

- a job scheduler
- will put your job in a queue for access to a compute node with the resources you request
- you should only request what you need – you'll end up waiting less!
- Your jobs will be non-interactive!



SLURM job submission script

submission_script_template.sh

```
#!/bin/bash

#SBATCH --job-name=<job_name>           # Name of the job
#SBATCH --account=def-<account_name>     # Account
#SBATCH --time=00:10:00                  # Max runtime (HH:MM:SS)
#SBATCH --cpus-per-task=1                # Number of CPU cores per task
#SBATCH --mem=1G                         # Memory per task
#SBATCH --output=logs/sub_task_%j.out    # File for standard output (%j = job ID)

# Run code
python <path to script you want to run>
```

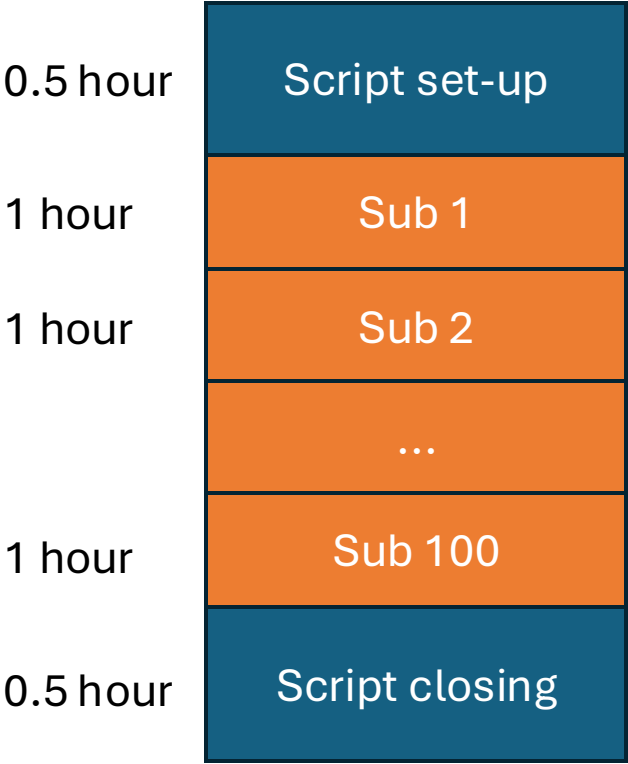
Submit this job with: `sbatch submission_script_template.sh`

SLURM commands

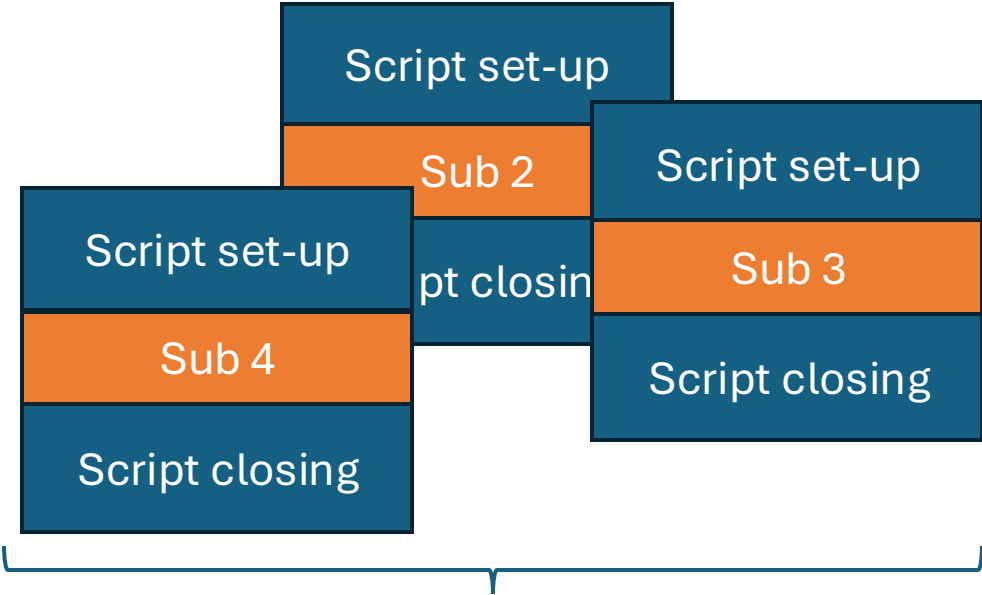
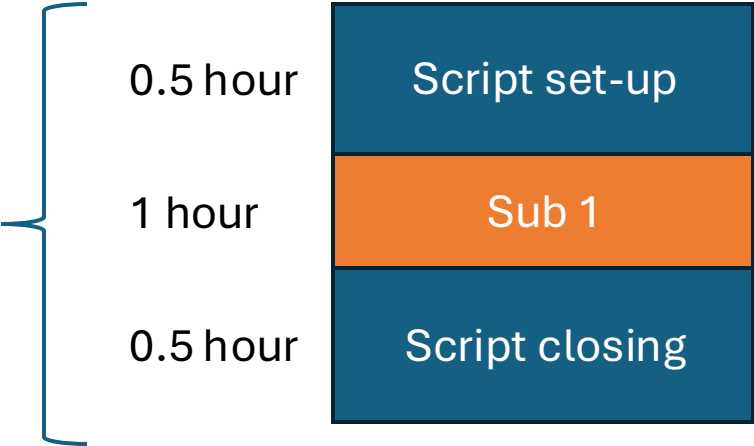
- **sbatch**
 - Submit a job (interactive or non-interactive)
 - Will assign the job a jobID
 - Non-interactive jobs will also output a slurm output log
- **sq**
 - Check progress of jobs
- **scancel <job_id>**
 - Cancels jobs

Parallelizing Scripts with Array Jobs

This would take
100 hours (4 days)
to run!



This would take
2 hours to run.



If we send each of these to a cluster, this
would take only 2 hours total!
(assuming no job queue waiting time)

Array job submission script

This example splits jobs into arrays based off subject ID, but they can be used to split any repetitive tasks

arrayjob_submission_script_template.sh

```
#!/bin/bash

#SBATCH --job-name=<job_name>           # Name of the job
#SBATCH --account=<account_name>        # Account
#SBATCH --time=00:5:00                  # Max runtime
#SBATCH --cpus-per-task=1                # Number of CPUs per task
#SBATCH --mem=1G                         # Memory per task
#SBATCH --output=logs/sub_task_%A_%a.out # Standard output log
#SBATCH --error=logs/sub_task_%A_%a.err  # Standard error log
#SBATCH --array=1-46                     # Number of tasks (one per subject). Change to match the number of subs

# Each array task will use $SLURM_ARRAY_TASK_ID to pick a subject from the list
subject_list=($(cat <path to subject list file>))
SUB="${subject_list[$((SLURM_ARRAY_TASK_ID-1))]}"

# Run Python script for this subject
python <path to script you want to run to process a single subject> $SUB
```

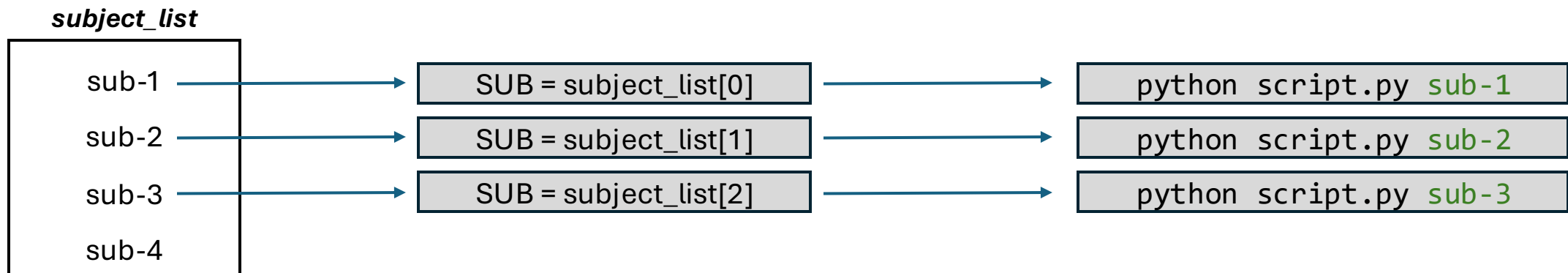
Submit this job with: `sbatch arrayjob_submission_script_template.sh`

```
#!/bin/bash

#SBATCH --job-name=<job_name>           # Name of the job
#SBATCH --account=<account_name>        # Account
#SBATCH --time=00:5:00                  # Max runtime
#SBATCH --cpus-per-task=1                # Number of CPUs per task
#SBATCH --mem=1G                         # Memory per task
#SBATCH --output=logs/sub_task_%A_%a.out # Standard output log
#SBATCH --error=logs/sub_task_%A_%a.err  # Standard error log
#SBATCH --array=1-46                     # Number of tasks (one per subject). Change to match the number of subs

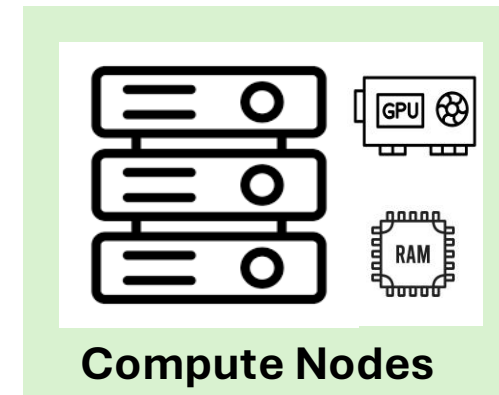
# Each array task will use $SLURM_ARRAY_TASK_ID to pick a subject from the list
subject_list=($(cat <path to subject list file>))
SUB="${subject_list[${SLURM_ARRAY_TASK_ID}-1]}]"

# Run Python script for this subject
python <path to script you want to run to process a single subject> $SUB
```



What is an interactive job?

- Jobs we've looked at so far are "non-interactive"
- Interactive jobs let you work directly in a compute node while accessing its resources (e.g. high memory, GPU)
 - JupyterLab is a type of interactive job
- When you need to develop/debug code WHILE using the GPU or a lot of memory
 - Working with neuroimaging data
 - Jupyter Notebooks
 - Virtual Machines
 - Analyzing large files



Instead of submitting a script as a job, you can directly access and work on the Compute Nodes

Running an interactive job

- https://docs.alliancecan.ca/wiki/Running_jobs#Interactive_jobs

- **Login to cedar**

```
$ ssh YOURUSERNAME@cedar.computecanada.ca
```

- **Navigate to scratch**

```
$ cd ~/scratch/
```

- **Request an interactive job allocation**

```
$ salloc --time=1:0:0 --mem-per-cpu=3G --ntasks=1 --account=<def-someuser>
```

- **To exit type:**

```
$ exit
```

Summary of jobs

- Jobs
 - Long jobs
 - Large resource tasks
- Array jobs
 - Jobs with many subjects
 - Repeating sub-tasks
- Interactive jobs
 - Testing or debugging using compute resources