NetSI PhD Bootcamp 2025

Code + Data + Cluster Computing

Introduction to High-Performance Computing

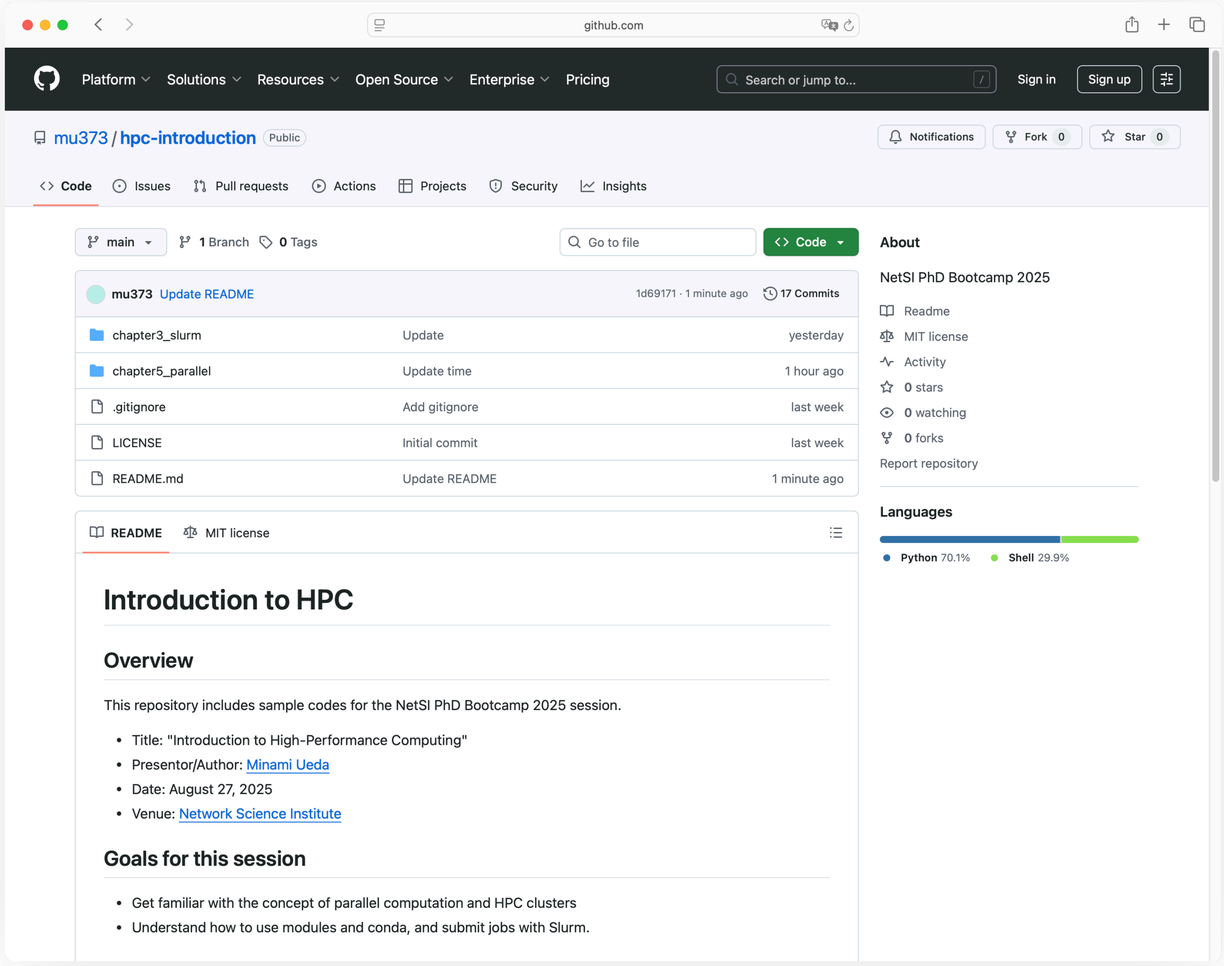
Minami Ueda

August 27, 2025

Network Science Institute

Goals for this session

- Get familiar with the concept of:
  - Parallel computation
  - HPC clusters
- Understand how to
  - Submit jobs with Slurm
  - Use modules and conda



https://github.com/mu373/hpc-introduction/

Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

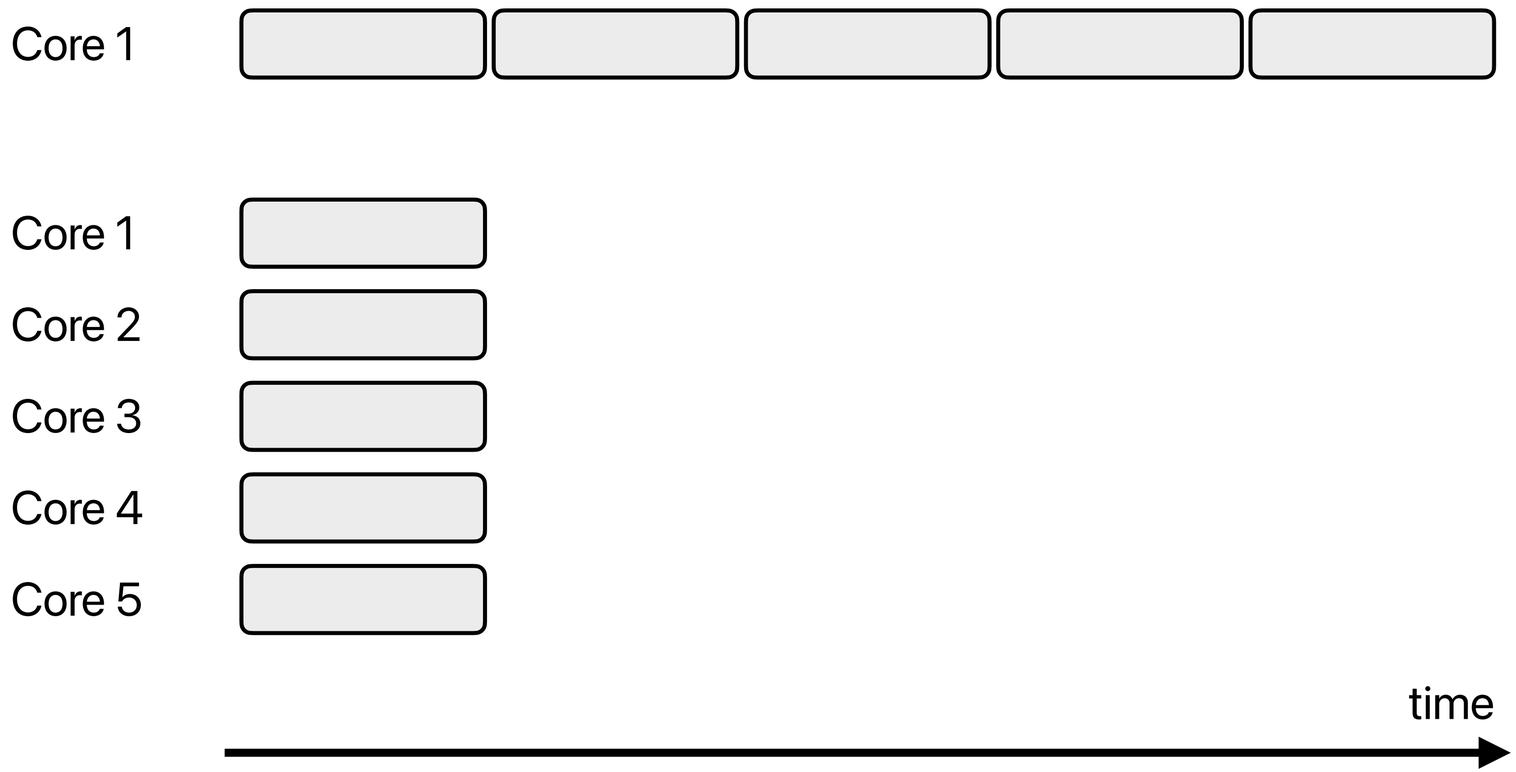
Goals

- Understand the overview and key concepts in
  - Parallel computation
  - HPC clusters
  - Slurm

Why parallel?

Parallel computing can save time by doing multiple things simultaneously

Serial
(single core)



5 minutes

Parallel
(5 cores)

*assuming that there is no dependency between each calculation

1 minute

5x speed

Why not a single powerful workstation?

- CPU: 64 Core
- RAM (memory): 128 GB
- Storage: 10 TB

Why not a single powerful workstation?

Limitations

- **Scalability**: What if you need 1,000 cores or 2 TB of RAM?
- **Availability**: What if your simulation takes two weeks to run? That single machine is exclusively occupied for the entire period.
- **Efficiency**: After your big simulation is done, you might only use 5% of the workstation's power for daily tasks. The expensive resource sits mostly idle.
- **Concurrency**: What if 10 people in your group all need to run demanding calculations at the same time? A single workstation creates a bottleneck where everyone has to wait in line.

Solution: High-Performance Computing cluster

a group of computers interconnected by high-speed network, that works as a single system

https://www.mghpcc.org/big-data-monster-storage-the-northeast-storage-exchange/

Discovery/Explorer

Northeastern's HPC cluster, physically hosted by MGHPCC in Holyoke, MA

MGHPCC

Massachusetts Green High Performance Computing Center

https://www.bu.edu/tech/support/research/rcs/mghpcc/; https://www.mghpcc.org/about-the-mghpcc/;

Discovery/Explorer

Northeastern's HPC cluster, physically hosted by MGHPCC in Holyoke, MA

1,024
CPU nodes

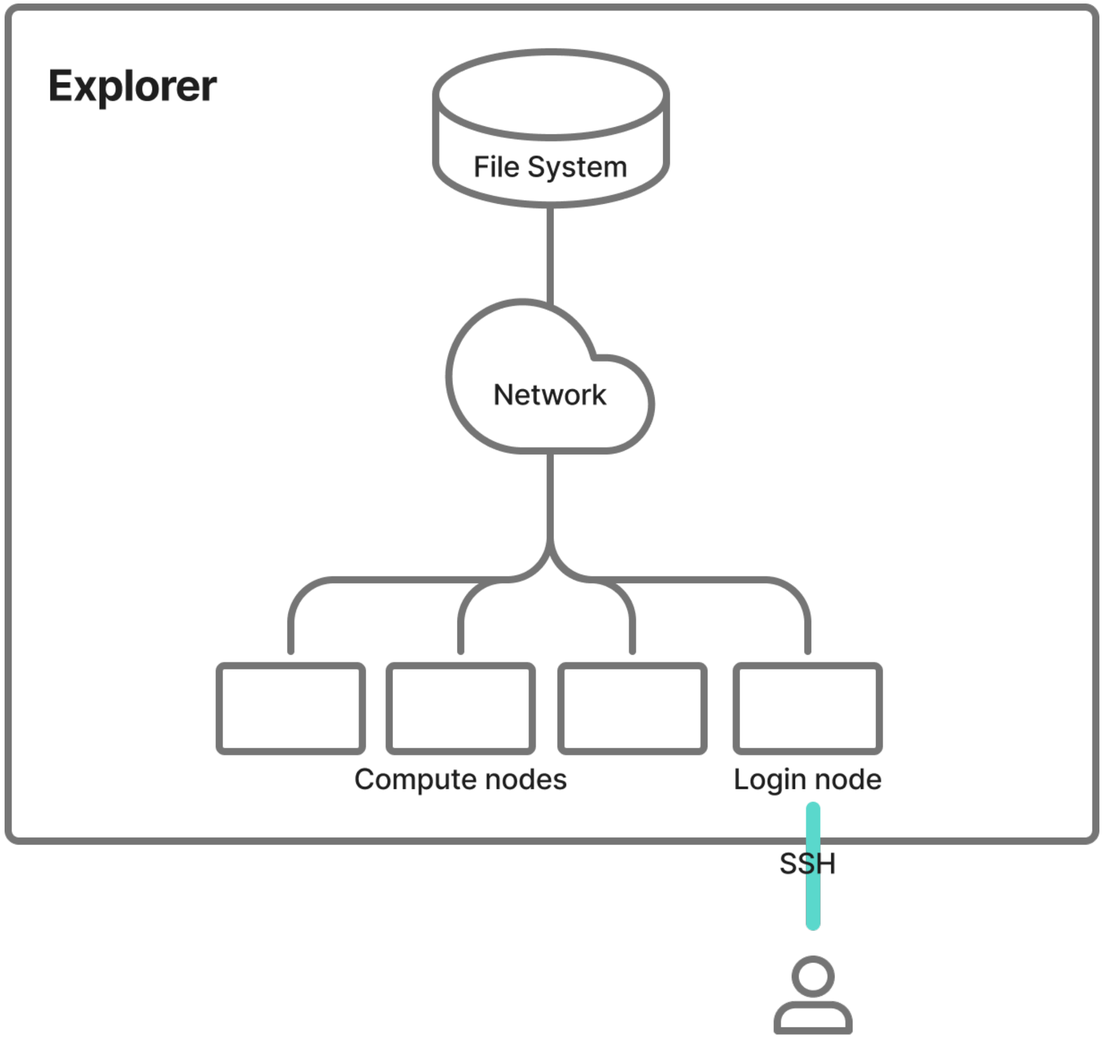
50,000
CPU cores

200
GPUs

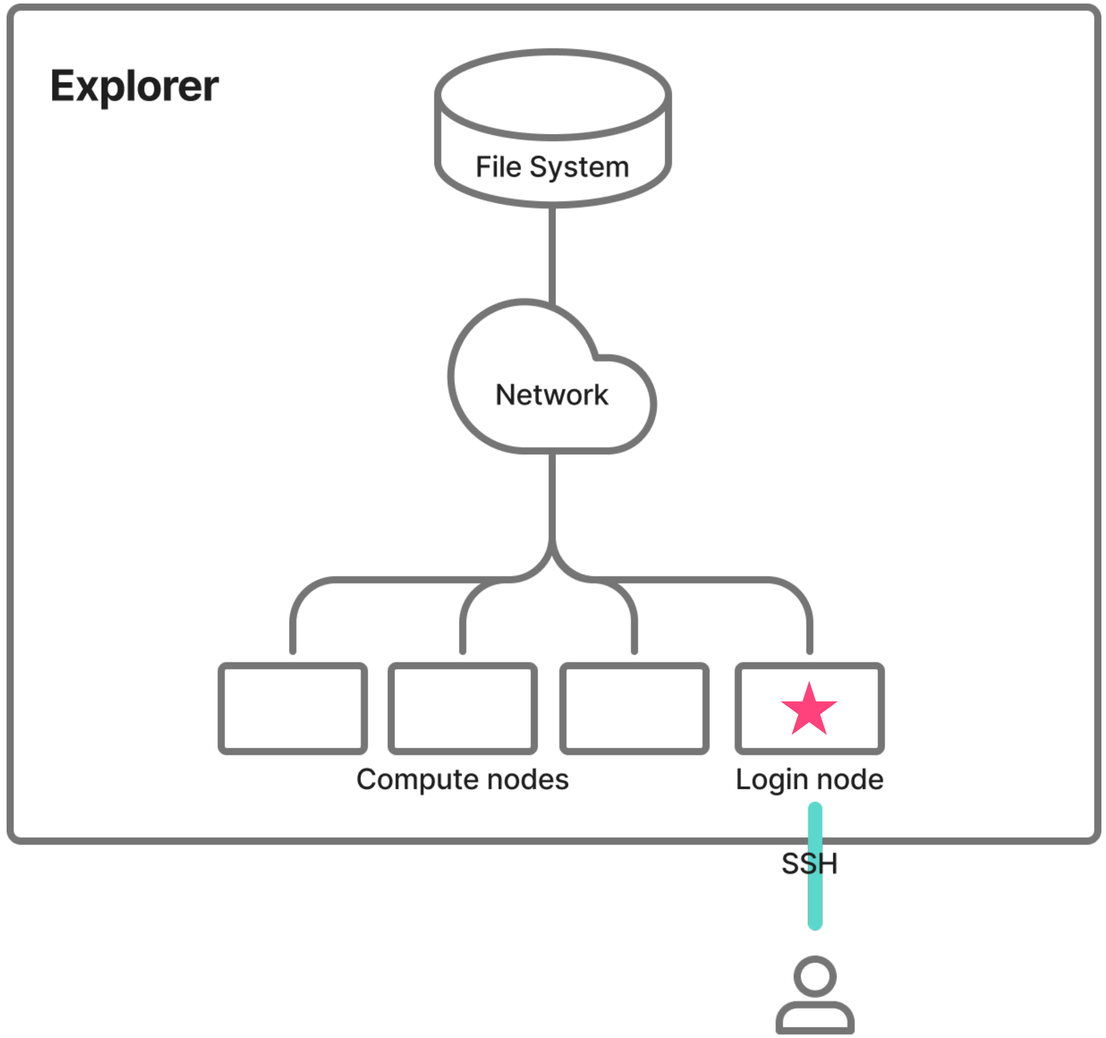
200 Gbps
internode network connection

Source: https://rc.northeastern.edu/compute/

HPC cluster overview

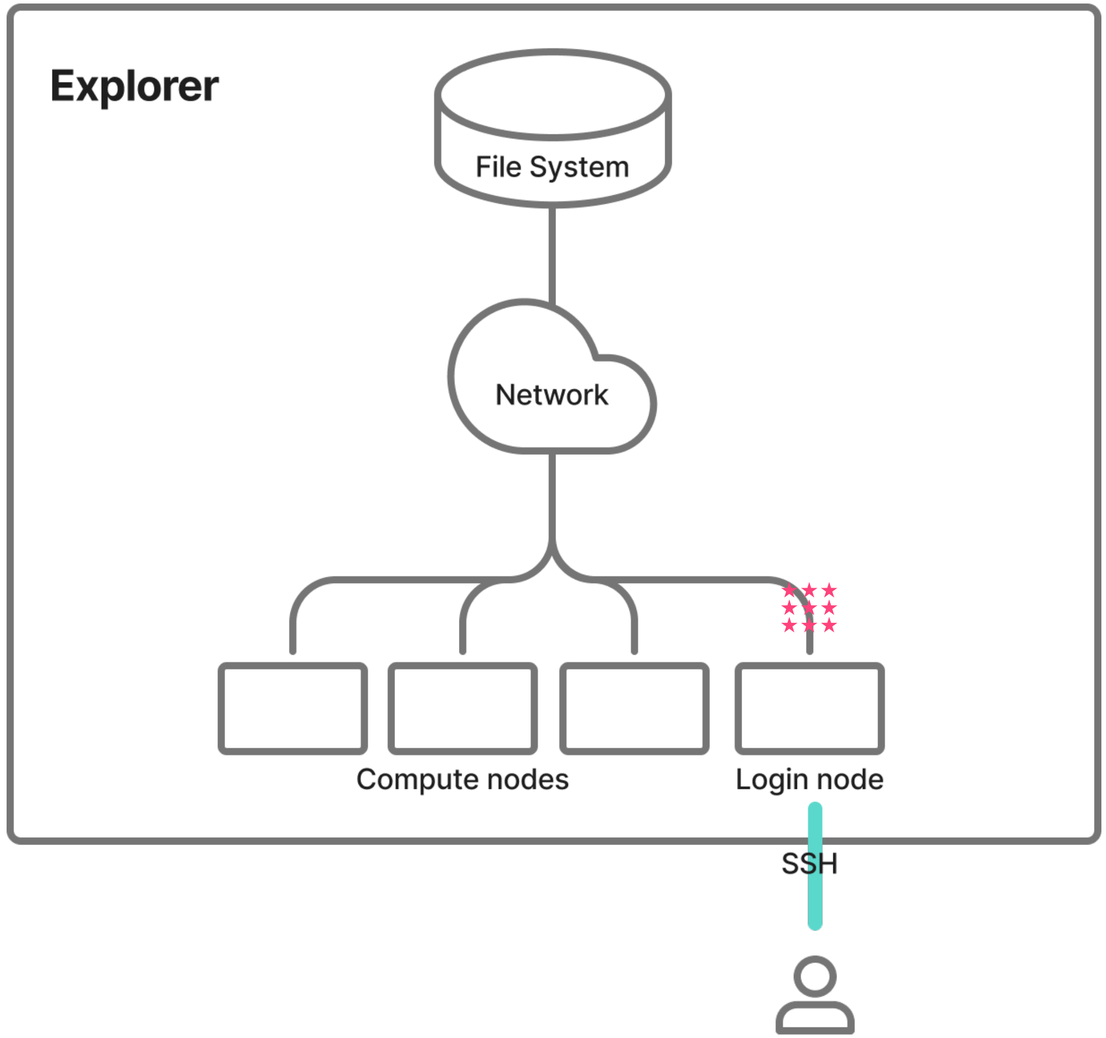


HPC cluster overview



We have a batch file

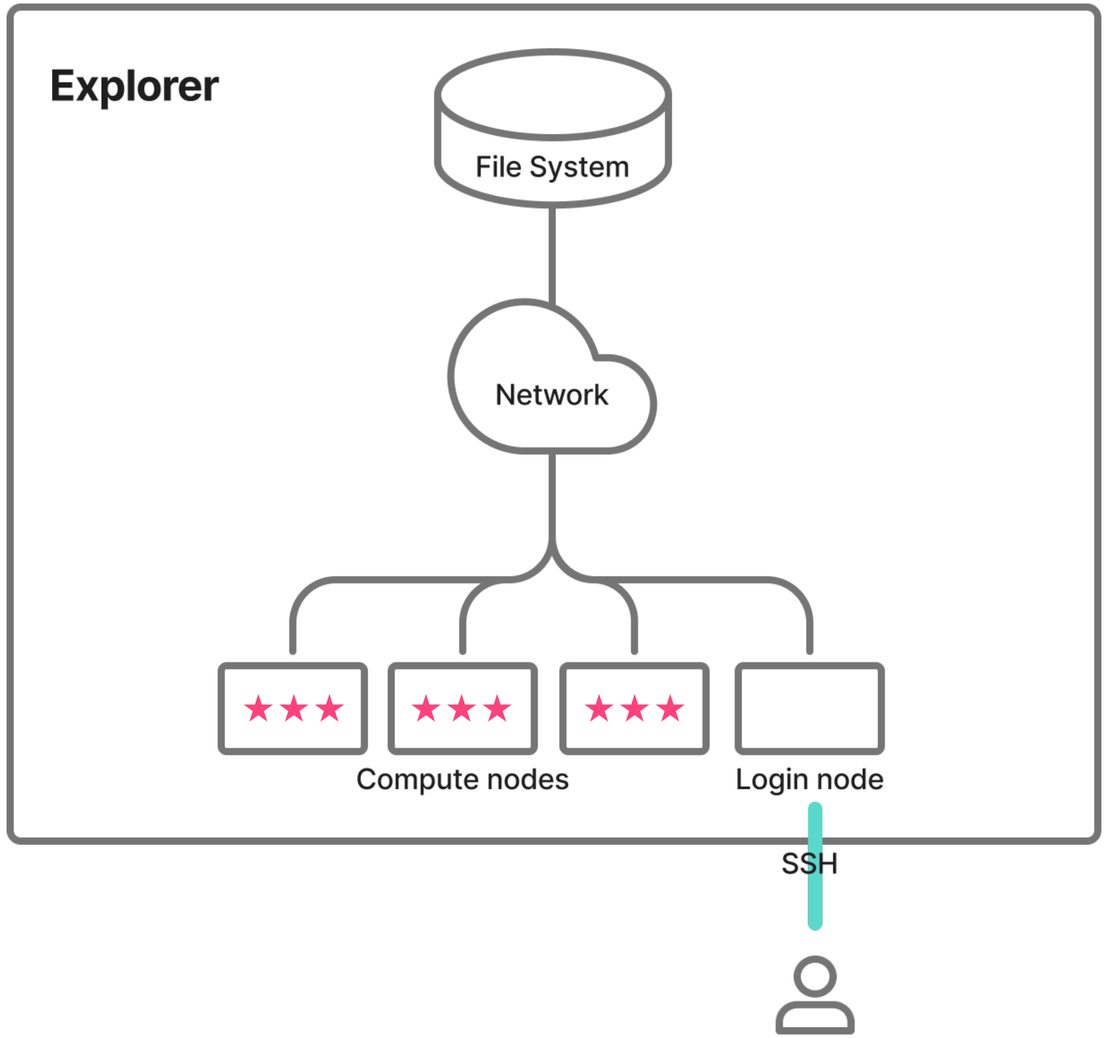
HPC cluster overview



We want to run 9 tasks

HPC cluster overview

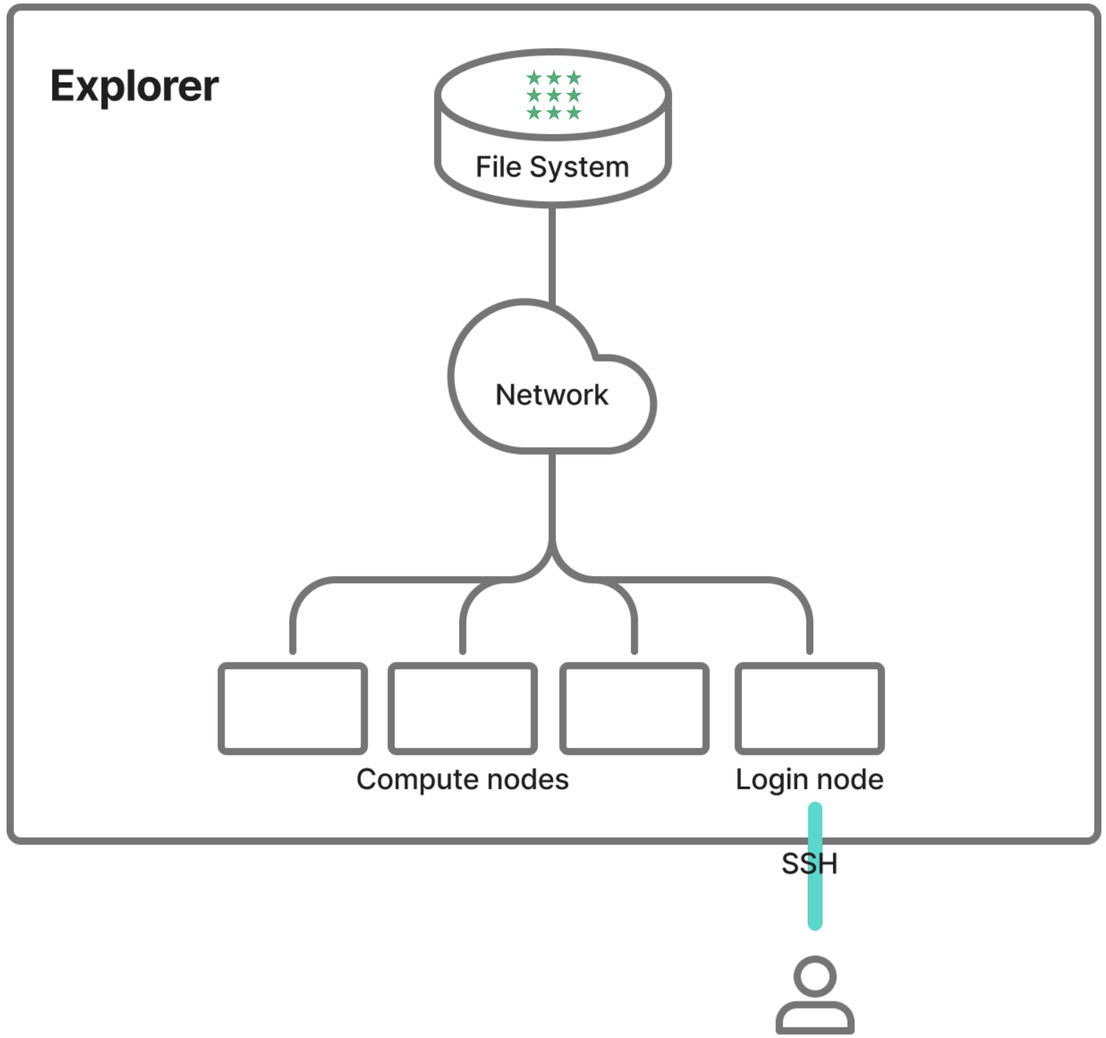
Tasks are allocated to available resources



HPC cluster overview

How do we orchestrate such complex system?

Results would be saved to the file system



Slurm

Job managing software for high-performance computing

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
- Manages queue of all user requests (jobs)
- Allocates jobs among the available computing nodes
- Ensures fair and efficient use of resources

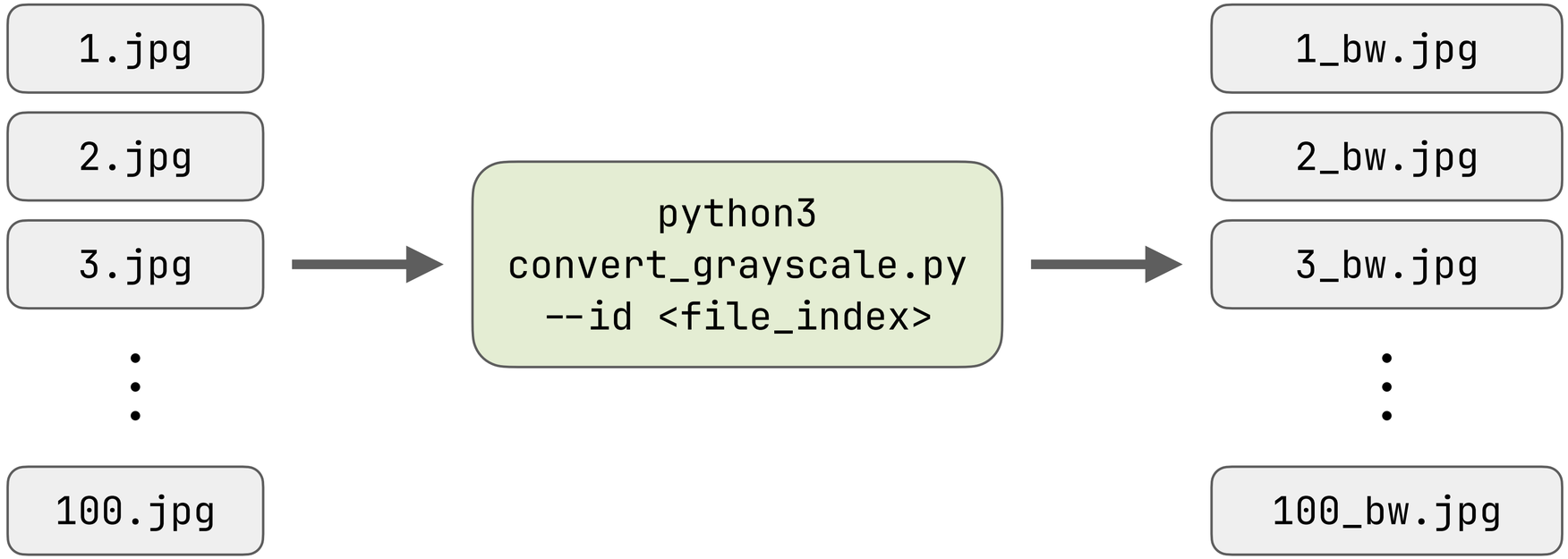
Key terms in Slurm

- **Node**
  - A single computing resource in the cluster. (Could be physical/virtual)
  - Login node: entry point for managing files and submitting jobs
  - Compute node: actual workhorses where your jobs actually run
- **Partition**
  - A logical group of compute nodes, often by specific hardware feature or usage policy
  - e.g., `express, short, gpu, netsi_standard`

Key terms in Slurm

- **Job**
  - A set of resource request and the programs you want to run
    - e.g., 4 CPU core, 16 GB RAM for 10 minutes + `bash myscript.sh`
  - Options specified via command line, or as a batch file (more common)
- **Job arrays**
  - A collection of jobs that can be submitted to run same/similar jobs

Job array example



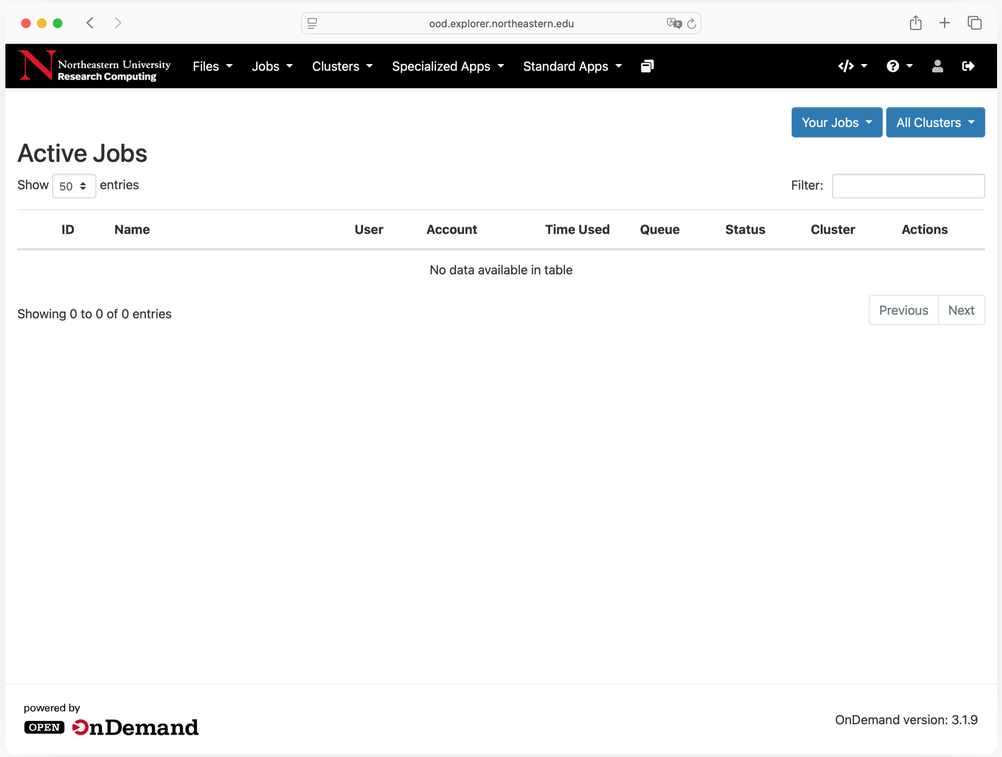

```
#SBATCH --array=1-100%10
```

“Make 100 jobs, and run at most 10 jobs in parallel“

Open OnDemand (OOD)

Web portal to Explorer cluster

- https://ood.explorer.northeastern.edu
- View/edit files
- Manage jobs (start/stop/monitor jobs)
- Start interactive sessions (e.g., JupyterLab)



Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

Hands-on Session

Goals

- Be able to remote access (SSH) to Northeastern HPC clusters
- Edit files in VS Code
- Run commands on login node in terminal

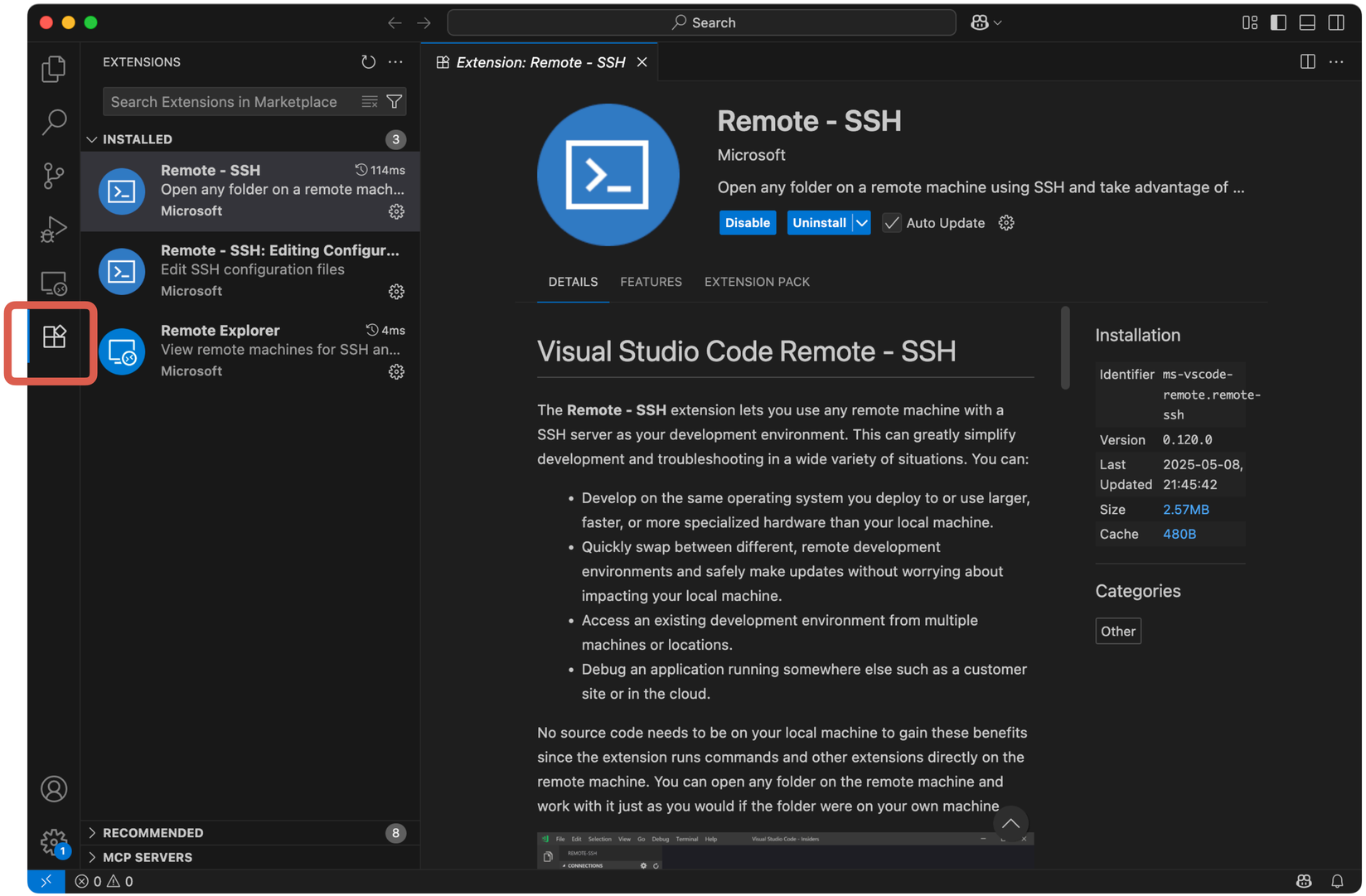
Setting up VS Code

Hands-on Session

https://code.visualstudio.com/

Install and enable Remote SSH extension

Hands-on Session



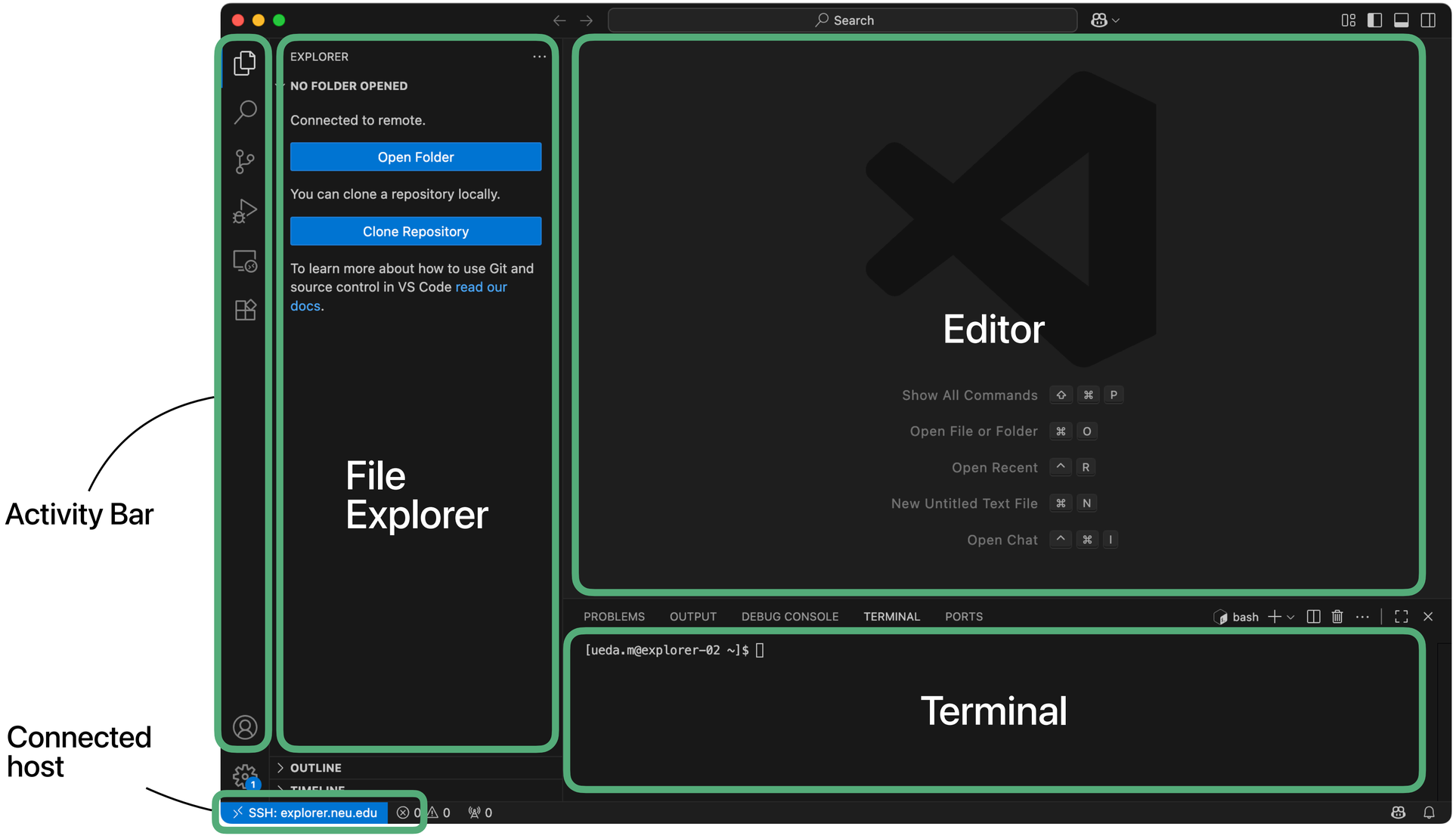
Connecting to Explorer

Hands-on Session

- Open Command Palette ⌘ + shift + P
- Start typing and select “Remote-SSH: Connect to Host...”
  - Enter `YOUR_USERNAME@explorer.northeastern.edu`
  - Enter password for your Northeastern account

Connection established to Explorer!

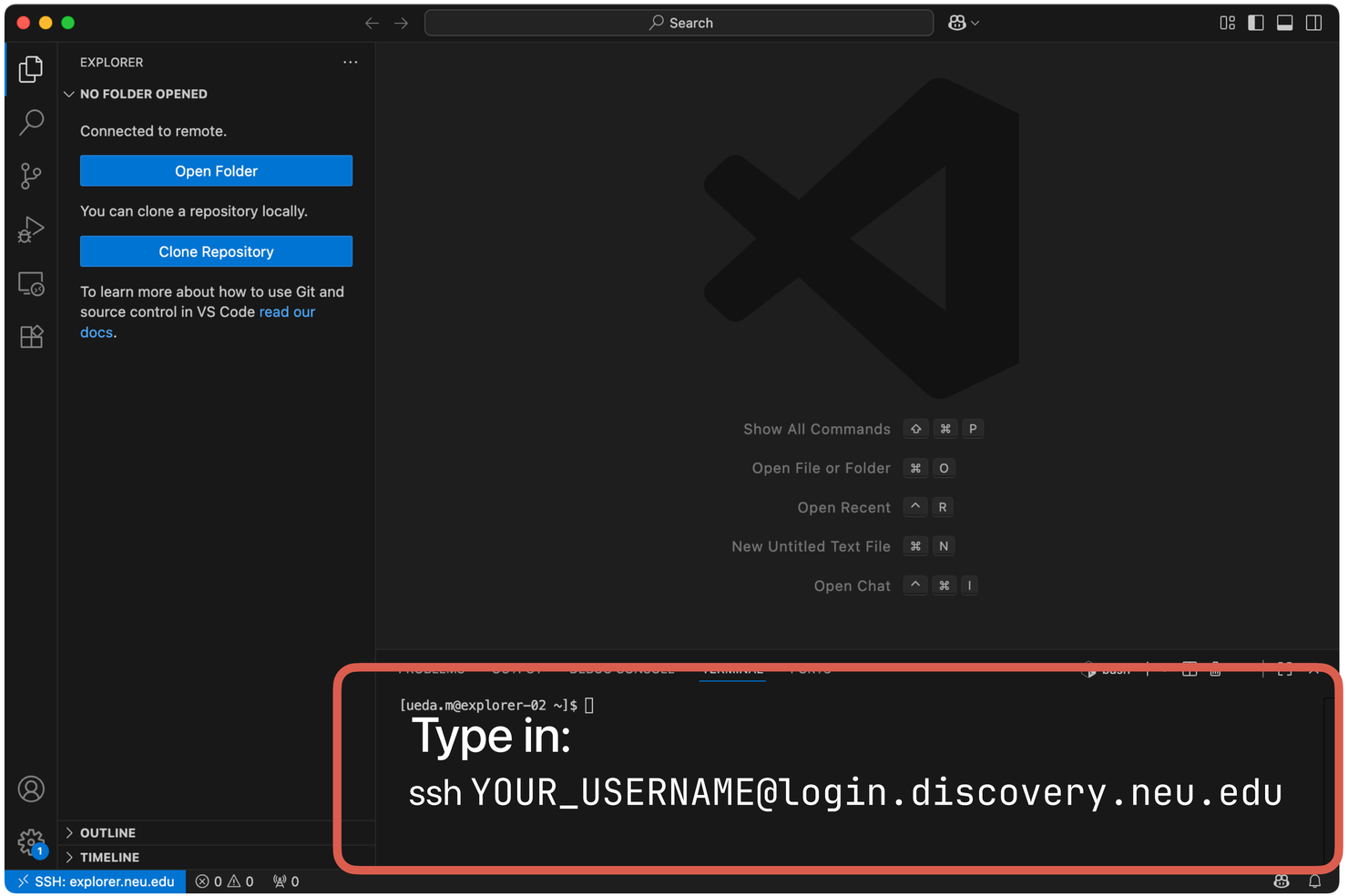
Hands-on Session



We also need to SSH to Discovery for now...

Hands-on Session

This is a temporary workaround since not everyone has a valid permission on Explorer for now.



Basic commands

```
# Check current directory
$ pwd
/home/ueda.m

# List files and directories
$ ls
note.txt      my_project

# Make new directory
$ mkdir bootcamp

# Change directory
$ cd my_project
$ ls
simulation.py   job.sh
```

Getting started

Hands-on Session

```
# Clone git repository
$ git clone https://github.com/mu373/hpc-introduction

# Move to downloaded repo
$ cd hpc-introduction

# List files
$ ls
chapter3_slurm  chapter5_parallel  LICENSE  README.md
```

Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm

Hands-on Session

4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

Goal

- Understand how to submit jobs with Slurm

Batch file for Slurm jobs

This defines:

- Resources to request for
- Job options
- Commands to execute/setup
  - anaconda environments
  - modules
  - analysis scripts

```
#!/bin/bash
#============ Slurm Options ===========
#SBATCH --job-name=my-job-1
#SBATCH --partition=express
#SBATCH --time=0-00:01:00

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

#SBATCH --mail-user=YOUR_USER_NAME@northeastern.edu
#SBATCH --mail-type=END
#=====================================

# Load required module
module load anaconda3

# Activate any necessary environments
source activate base

# Run program of interest here!
python3 example1.py
```

Slurm commands

```
sbatch my_job.sh
```

Submit a job to the queue.

```
squeue --me
```

Show the status of your jobs.

```
scancel <job_id>
```

Cancel a pending or running job.

```
srun --job-name test --time 00:01:00 -p short --pty bash -i
```

Start an interactive job.

Let’s try using Slurm!

Hands-on Session

Step 1: Preparation

- Open `chapter3_slurm` directory. This includes:
  - `hello_cluster.py`: Python script
  - `job.sh` : Batch file
- Open each files in the editor, and see what it does.
- For job.sh, change email address to yours.

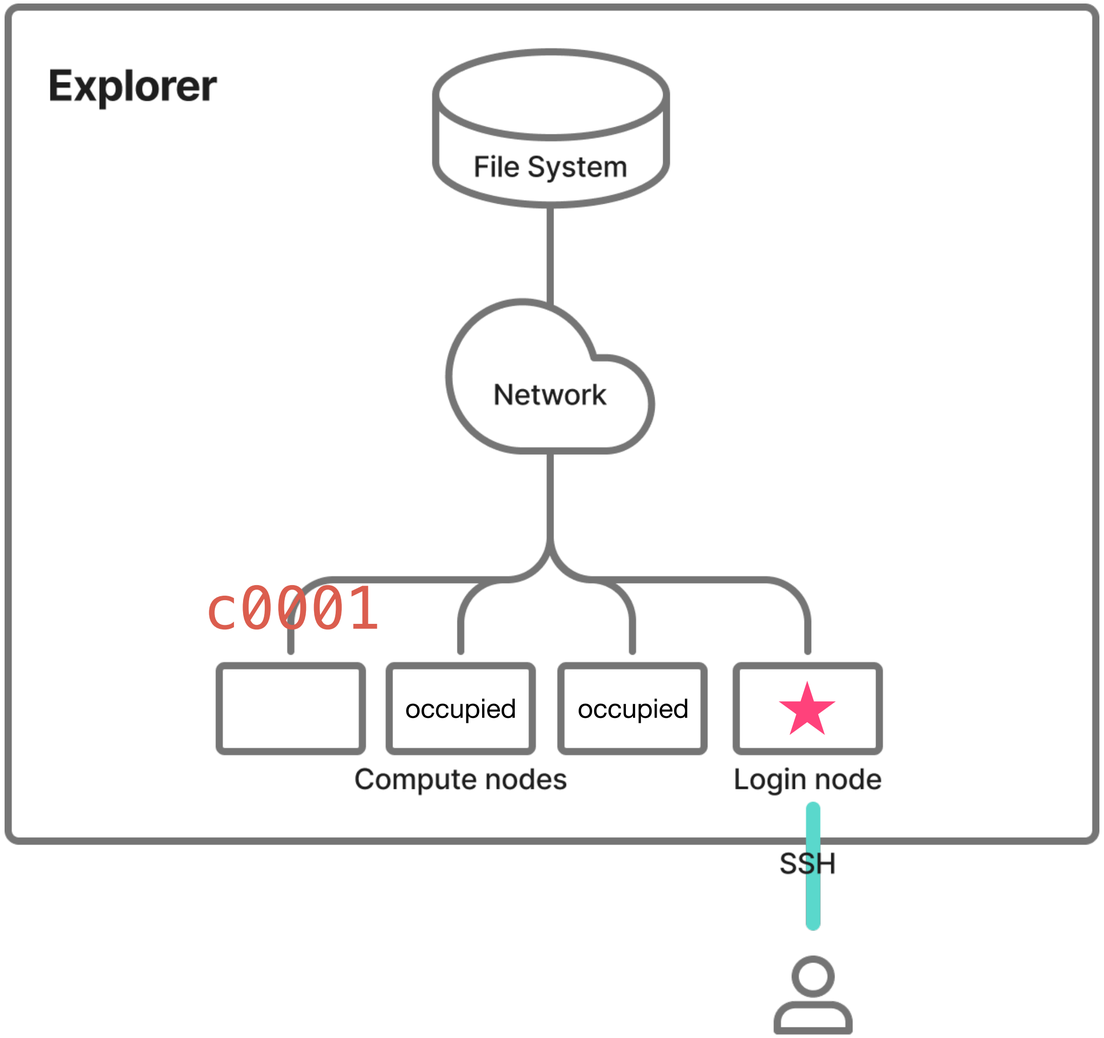
Let’s try using Slurm!

Hands-on Session

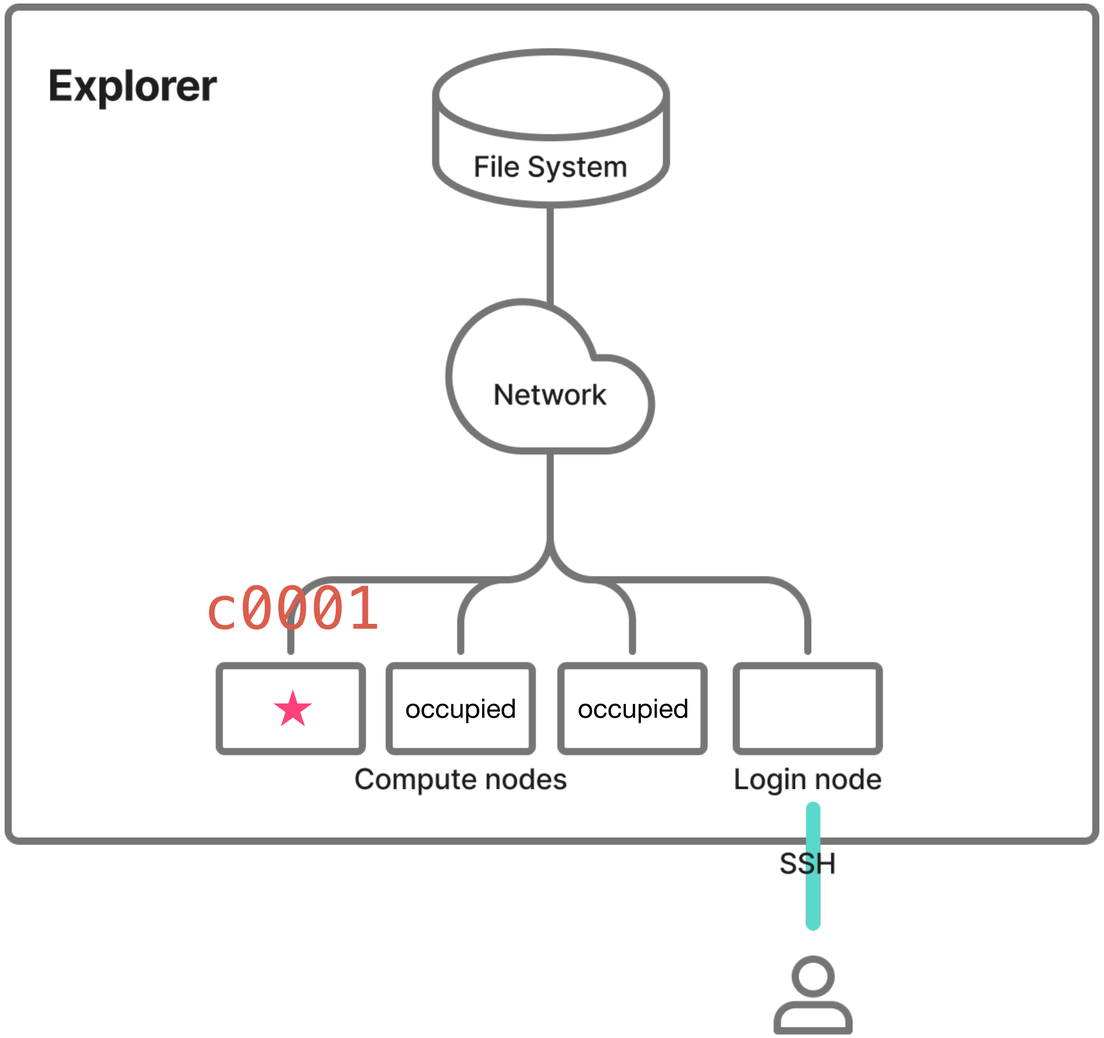
Step 2: Submitting job

```
# Change directory
$ cd chapter3_slurm

# Submitting job
$ sbatch job.sh
Submitted batch job 49725148

# Watch for the job to run
# Look for ST (status) column. PD=pending, R=Running.
# Use <Ctrl-C> to quit
$ watch -n 1 squeue --me

# Look for the output file
# Open slurm-***.out and see what the result looks like
$ ls
hello_cluster.py  job.sh  slurm-49725148.out
```

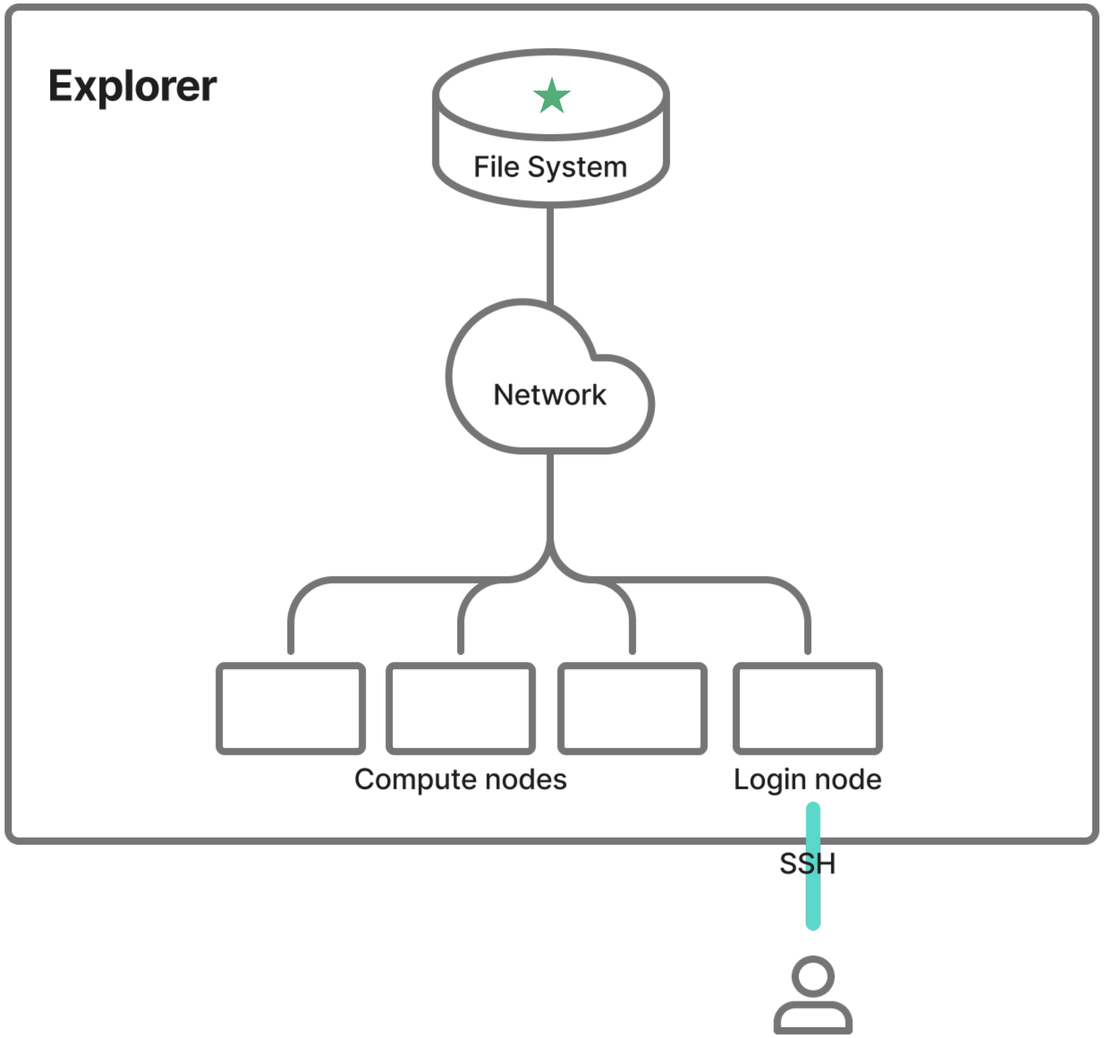
HPC cluster overview



HPC cluster overview



HPC cluster overview



Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments — Hands-on Session
5. Running simulations in parallel
6. Best practices

Goal

- Understand what module environments and Anaconda are
- Be able to use “module” and “conda” commands

Managing softwares and environments on HPC

- Large number of scientific applications, softwares, and packages
- Different versions, different dependencies
  - Python 3.9, 3.10, 3.11, etc...
  - numpy v2.3.2 + pandas 2.1.1 + ...
- This is where these two becomes convenient:
  - **Environment modules**
  - **Anaconda**

Environment modules: `module` command

- Major softwares for scientific computing are pre-installed on HPC
- You can load specific version of a software with module command

```
$ module avail
------------------------------------------ /shared/EL9/explorer/modulefiles -------------------------------------------
Abaqus/2024           cuda/12.1.1              HDF5/nvhpc-1.14.6            miniconda3/24.11.1   python/3.13.5
admin/2018-10-10      cuda/12.3.0              htop/3.3.0                   Molcas/25.06         R/4.4.1
AMBER/24              cuda/12.3.0.old          igv/2.18.4                   MPICH/4.3.0b1        ROCm/6.3.0
anaconda3/2024.06     cuda/12.8.0              intel/compilers-2025.0.4     namd/3.0.1           samtools/1.21
bamtools/2.5.2        cuDNN/9.10.2             intel/compilers-rt-2025.0.4  namd/3.0.1-CUDA      SAS/9.4
bbmap/39.11           discovery/1.0            intel/mkl-2025.0             netlogo/6.4.0        schrodinger/2024-4
bcftools/1.21         EMAN2/2.99.47            intel/mpi-2021.14            nextflow/24.10.3     schrodinger/2025-3
bedtools/2.31.1       explorer/1.0             intel/tbb-2022.0             nodejs/v22.11.0      sratoolkit/12Dec2024
blast/2.13.0          fastqc/0.12.1            intel/umf-0.9.1              Nsight/2024.7.1      star/2.7.11b
Boost/1.88.0          FFmpeg/7.1.1             job-assist/1.0               nvidia-hpc-sdk/24.7  starccm+/18.02.008
bowtie2/2.5.4         GA/5.9.2                 jq/1.7.1                     OpenBabel/2.4.1      starccm+/18.04.009
bwa/0.7.18            gaussian/g16             julia/1.10.4                 OpenBLAS/0.3.29      Stata/17
clustal/2.1           glew/2.2.0               knime/5.3.1                  OpenCV/4.10.0        Tinker/8.11.3
cmake/3.30.2          GNU-parallel/2025-04-22  LAMMPS/29Aug2024             OpenFOAM/12          trimmomatic/0.39
code-server/4.91.1    go/1.23.2                mathematica/14.1.0           OpenJDK/22.0.2       vcftools/0.1.16
code-server/4.101.1   gromacs/2024.3           matlab/3.0.1                 OpenMPI/4.1.6        VMD/1.9.3
comsol-jornet/60      gurobi/11.0.3            matlab/R2023b                ParaView/5.13.0-RC2  VMD/1.9.4a55
comsol-nanofem/62     gurobi/12.0.2            matlab/R2024b                ParaView/5.13.1      VTune/2025.0
comsol-west/60        HDF5/1.14.6              matlab/R2025a                perl/5.40.0          weka/3.8.6
```

Environment modules: `module` command

Hands-on Session

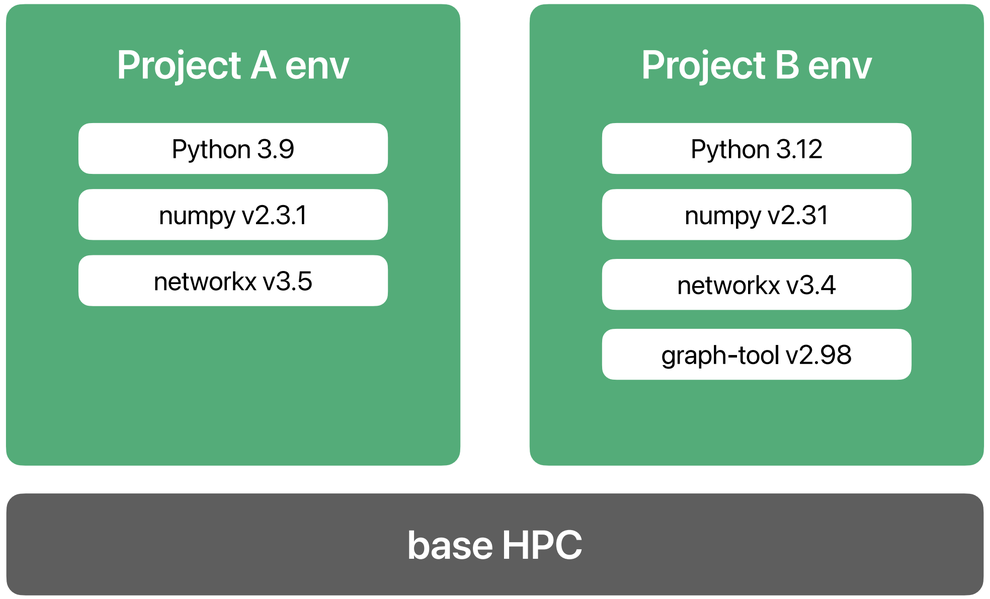
```
# List loaded modules
$ module list
Currently Loaded Modulefiles:
 1) explorer/1.0

# Search for specific module
$ module avail anaconda3
--------------- /shared/EL9/explorer/modulefiles ---------------
anaconda3/2024.06

# Load module
$ module load anaconda3

# List loaded modules again
# Now you can use conda command!
$ module list
Currently Loaded Modulefiles:
 1) explorer/1.0   2) anaconda3/2024.06
```

Anaconda

- An open-source package manager, environment manager, and Python distribution
- Used for scientific computing, data science, and machine learning

Why use Anaconda?

- **Environment isolation**: Self-contained virtual environments for different projects to prevent package version conflicts
- **Reproducibility**: Easily share and recreate environment



Basic conda commands

*Hands-on session follows in the next slide

```
# Load anaconda. conda command won't work without this!
$ module load anaconda3

# Create new virtual environment
$ conda create -n my-env python=3.10

# List environments
$ conda env list

# Activate virtual environment
$ source activate my-env

# Install package inside
(my-env)$ conda install numpy pandas networkx

# Deactivate virtual environment
$ source deactivate
```

Let’s try using conda

Hands-on Session

```
# Check which Python is called by default
$ which python3
/usr/bin/python3

# Fails because networkx is not installed
$ python3 -c "import networkx as nx; G = nx.karate_club_graph(); print(G.number_of_nodes());"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
    import networkx as nx
ModuleNotFoundError: No module named 'networkx'

# Activate anaconda
$ module load anaconda3

# Activate bootcamp environment
$ source activate /home/ueda.m/conda/bootcamp

# Confirm that Python from the bootcamp environment is used
$ which python3
/home/ueda.m/conda/bootcamp/bin/python3

# Should return 34
$ python3 -c "import networkx as nx; G = nx.karate_club_graph(); print(G.number_of_nodes());"
34
```

Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

Hands-on Session

Goal

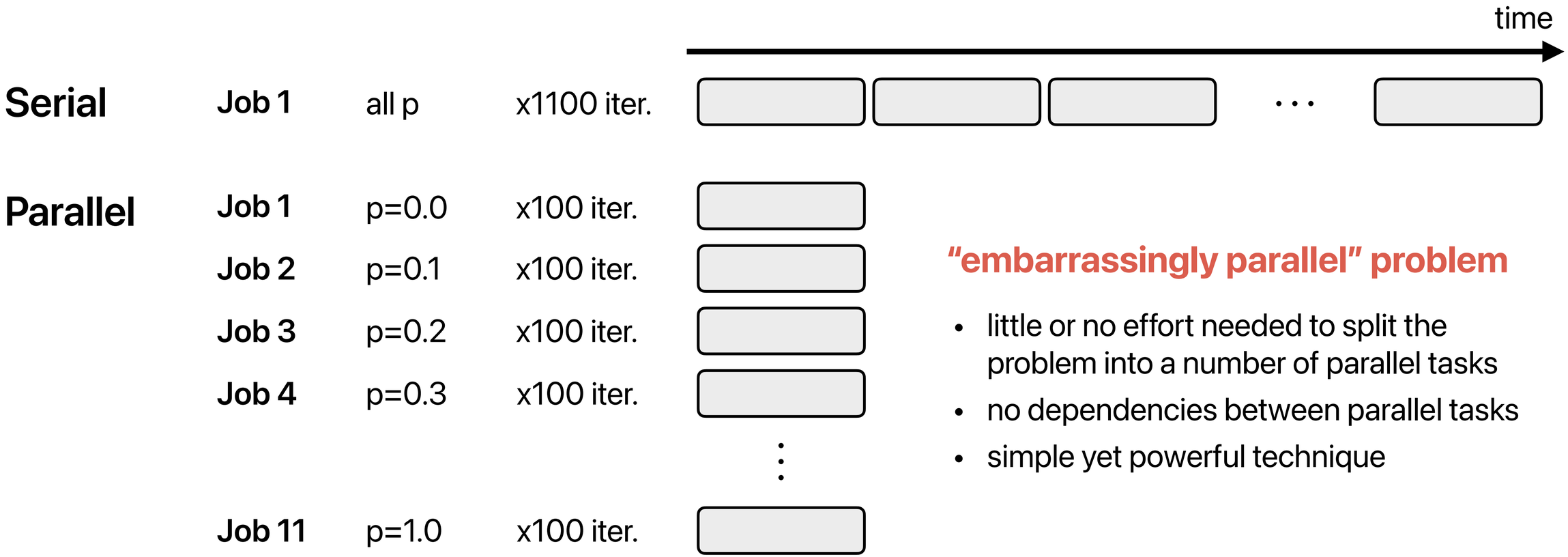
- Be able to use job array to run “embarrassingly parallel” type computations

Problem setting

- Interested in the property of ER graph
  - Relationship between **probability of connection** and **average degree**
- p-value ranges from [0, 0.1, 0.2, ..., 1.0]
- Simulate 100 ER graphs for specific range of p-values
- Total number of simulations: 11 p-values * 100 interactions = 1100 simulations

How do we compute this?

Total number of simulations: 11 p-values * 100 interactions = 1100 simulations



"embarrassingly parallel" problem

- little or no effort needed to split the problem into a number of parallel tasks
- no dependencies between parallel tasks
- simple yet powerful technique

Two step workflow

Hands-on Session

1. Run simulations for all p-values. Export average degree as CSV file.
2. Load CSV files, calculate average average degree for each p-value, and then plot.

```
$ cd ../chapter5_parallel

$ ls
job_plot.sh  job_sim.sh   plot.py   sim.py
```

1. Running simulation

Hands-on Session

- Open sim.py and see what it does.
- Open job_sim.sh and edit your email address
- Submit job on Discovery

```
# Submitting job
$ sbatch job_sim.sh

# Watch for the job to run
# Look for ST (status) column. PD=pending, R=Running.
# Use <Ctrl-C> to quit
$ watch -n 1 squeue --me

# Look for the result directory
$ ls
job_plot.sh  job_sim.sh   plot.py   sim.py  figs  logs  results
```

1. Running simulation

Hands-on Session

```
p,trial_id,avg_degree
0.3,0,297.942
0.3,1,300.628
0.3,2,299.56
0.3,3,300.336
0.3,4,300.506
0.3,5,299.928
0.3,6,298.894
0.3,7,299.578
0.3,8,298.606
0.3,9,299.738
0.3,10,300.442
0.3,11,299.356
0.3,12,298.828
0.3,13,299.282
0.3,14,299.322
0.3,15,299.594
0.3,16,299.664
0.3,17,299.078
0.3,18,300.25
0.3,19,300.384
```

2. Plotting results

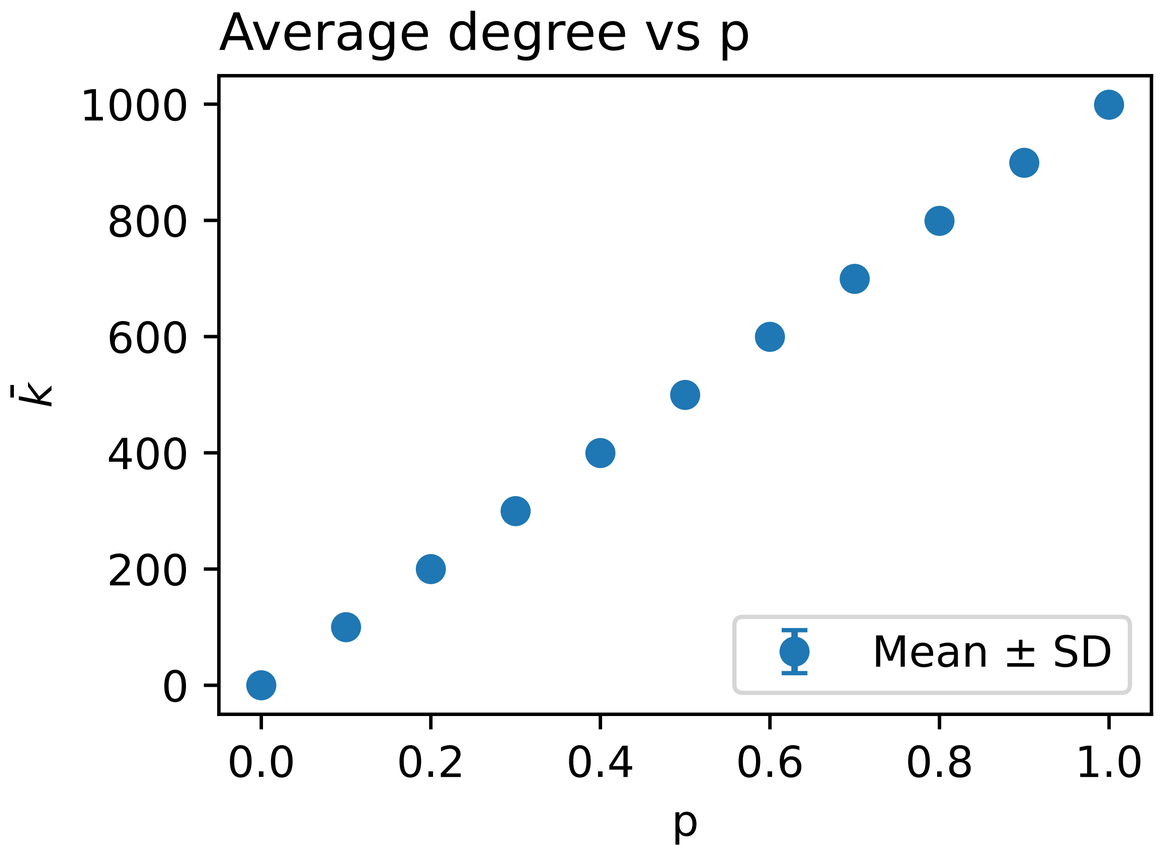
Hands-on Session

- Open plot.py and see what it does.
- Open job_plot.sh and edit your email address
- Submit job on Discovery

```
# Submitting job
$ sbatch job_plot.sh

# Watch for the job to run
# Look for ST (status) column. PD=pending, R=Running.
# Use <Ctrl-C> to quit
$ watch -n 1 squeue --me

# Look for the figs directory
$ ls
job_plot.sh  job_sim.sh   plot.py   sim.py  figs  logs  results
```

2. Plotting results

Hands-on Session



$$\bar{k} = p(n-1)$$

Agenda

1. Introduction
2. Environment setup
3. Batch jobs on Slurm
4. Managing softwares and environments
5. Running simulations in parallel
6. Best practices

Interacting with HPC clusters

- **Don’t run heavy processes on the login node**
  - Login node is a shared resource for everyone for editing files, remote access, and submitting jobs.
  - Heavy process examples:
    running analysis/simulation script, uncompressing large zip file,
    `conda install <some packages>`
  - Instead, use `sbatch` to submit jobs or use `srun` to start an interactive job

Interacting with HPC clusters

- **Beware of fair resource usage**
  - HPC clusters (including compute nodes) are shared resources among the users
  - Set limits to task duration, and maximum concurrent tasks (for job array)
  - Consult to Research Computing Team when you are in trouble
    - e.g., The job is “stuck” and does not release the allocated resources for a long time.

Interacting with HPC clusters

- **Use appropriate file storages**
  - Home directory `~/`: Permanent storage space for yourself.
  - Scratch directory `/scratch`: Use for large, temporary data.

    Purged periodically, so don't keep important files here!
  - Work or Project directory `/work` or `/project`:

    A storage space for your lab or project, administered by PIs.

    NetSI has `/work/netsi`

Source: https://rc-docs.northeastern.edu/en/latest/best-practices/homequota.html

Writing codes

- **Write docstrings and comments**
  - Surprisingly, you can easily forget what you wrote!
  - Good documentation helps you and others to understand what the code does
  - Explain “why” in comments, not “what”
  - Follow standard docstring format (e.g., NumPy style)

Writing codes

- **Write docstrings and comments**
  - Surprisingly, you can easily forget what you wrote!
  - Good documentation helps you and others to understand what the code does
  - Explain “why” in comments, not “what”
  - Follow standard docstring format (e.g., NumPy style)

Writing codes

- **Keep logs for debugging and reproducibility**
  - Logs are the key to debug failed jobs
  - Avoid using print(). Use dedicated library.

```
from logging import getLogger
logger = getLogger(__name__)
logger.info(‘message')
logger.debug(‘some error message’)
```

Writing codes

- **Start with the overall workflow, module and experiment design, then proceed to the actual coding**
  - Primary role is shifting from a writer of code to an architect
  - LLMs can generate codes pretty well, but its output may lack consistency or design philosophy as the code base grows.
  - Design modules that can be reused in the projects
  - Running simulations can take time. Design and plan out the experiment well.
    - Which parameters are you changing, and what are you measuring?

Writing codes

- **Command-line arguments are useful for large scale tasks**
  - Receive Slurm task ID as command-line argument
  - Parse it using argparse
  - Use the task ID as an index of parameter sets, file lists, etc.
    - `parameter_sets = [ {alpha=0.1, beta=0.1}, {alpha=0.1, beta=0.2}, …]`
    - `parameters = parameter_sets[task_id]`
    - `run_simulation(**parameters)`

Ensuring reproducibility

- **Use Git (version control software)**
  - Git tracks changes to the code, allowing you to revert the code to some previous versions.
  - It’s essential for collaboration!
  - By keeping contents on GitHub, you can avoid data losses.
    It works as a remote backup!

Ensuring reproducibility

- **Use virtual environments in conda**
  - Don’t use same environment for everything
  - It can cause conflicts in package versions, due to dependencies
  - Make a separate environment for graph-tool!

Useful resources

- Northeastern Research Computing Team: rchelp@northeastern.edu
- Explorer Documentation: https://rc-docs.northeastern.edu/en/explorer-main/index.html
- Guide from Princeton Research Computing: https://researchcomputing.princeton.edu/get-started/guide-princeton-clusters
  - Provides thorough guide and tutorial for HPC.
- My documentation website: https://minamiueda.com/docs/category/computation
- PHYS7332 materials by Matteo Chinazzi, Qian Zhang, Brennan Klein, and Alyssa Smith: https://asmithh.github.io/network-science-data-book
- Snippet manager “pet”: https://github.com/knqyf263/pet
  - You don’t have to type or remember the exact commands and options every time!