“OK, I Got Access, Now What?”

Introduction to Supercomputing Basics

Chris Stylianou

Research Engineer

CaSToRC
eurocc.cyi.ac.cy

Co-funded by the European Union

Republic of Cyprus

RESEARCH & INNOVATION FOUNDATION

EuroHPC Joint Undertaking

A Bit About Myself

- Research Engineer at CaSToRC, PhD
- *Training and Skills Development* Task Leader for EuroCC2
- Area of expertise in High Performance Computing (HPC)
- HPC x AI
- Contact & Info:
  - Email: c.stylianou@cyi.ac.cy
  - Website: cstyl.github.io

GitHub Link: https://github.com/CaSToRC-CyI/eurocc2_Supercomputing_for_all

Agenda

- Supercomputing Architecture Overview
- Navigating the HPC Environment
- Job Scheduling and SLURM Basics
- Jupyter Notebook on HPC

What Makes Supercomputing Essential?

- Enables breakthroughs in **science, engineering, and AI.**
- Solves computationally intensive problems faster and more efficiently than traditional systems.
- Supports large-scale **simulations, data processing, and AI training.**

Impact Areas:

- **Research**: Climate modelling, genomics, physics simulations.
- **Industry**: Product design, financial modelling, and optimization.
- **AI and Data Science**: Training large models, real-time analytics.

Learning Objectives

Understand Supercomputing Basics	Key components: nodes, processors, memory, and storage. Roles of login and compute nodes.
Navigate the HPC Environment	Access Cyclone securely via SSH. Manage files and directories with Linux commands.
Comprehend Job Scheduling	Basics of SLURM for resource management. Key commands: job submission, monitoring, and cancellation.
Recognize Resource Management Best Practices	Efficiently allocate CPUs, memory, and GPUs. Avoid common pitfalls in HPC workflows.
Explore Interactive HPC Tools	Benefits of Jupyter Notebooks. Overview of launching Jupyter on HPC systems.

What is a Supercomputer?

Definition:

- A **supercomputer** is a high-performance machine capable of executing complex computations at exceptional speeds.
- Built for **parallel processing** to handle massive datasets, simulations, and AI workloads.

Role in HPC and AI:

- Powers research in fields like **climate modeling, genomics, and physics.**
- Drives advancements in **AI and machine learning** by accelerating training and inference.
- Enables **real-time analytics** and decision-making in industry.

What is a Supercomputer?

Supercomputers vs Cloud Computing

Supercomputers

Specialized Hardware:

- Optimized for **parallel** and **capability computing**
  - solving the **largest** and most **complex problems**
- High-speed interconnects.

Purpose:

- Focused on scientific research, simulations, and high-performance tasks.

Resource Allocation:

- Fixed and highly controlled environment.
- Users **share resources** via job **schedulers** like SLURM.

Cloud Computing

General-Purpose Hardware:

- Flexible and scalable **virtual machines**.
- Designed for **capacity computing**
  - handling many smaller tasks concurrently.
- Standard networking and storage infrastructure.

Purpose:

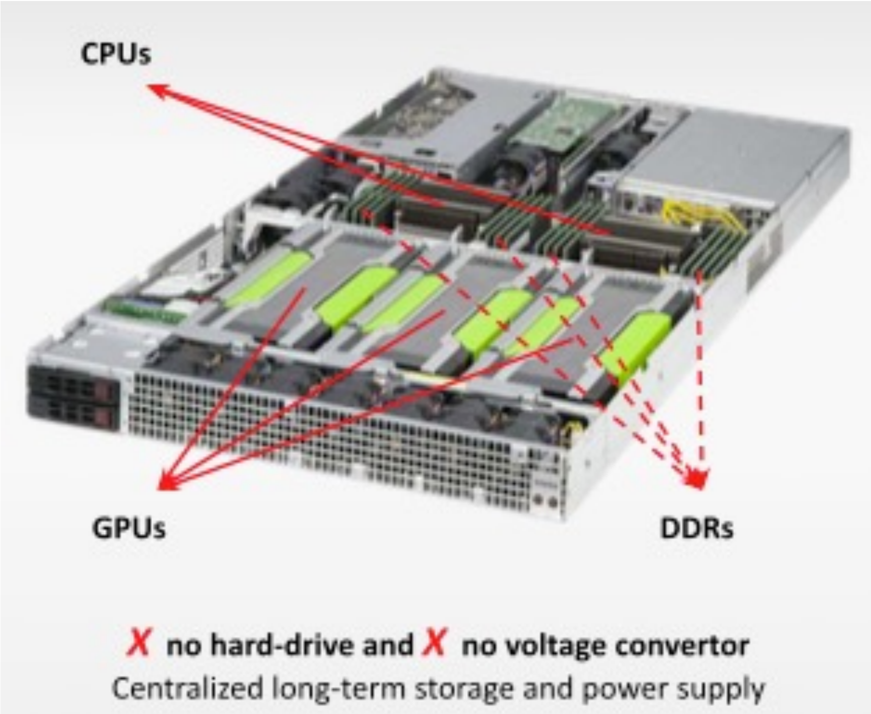
- Ideal for diverse workloads, including web applications, and general-purpose computing.

Resource Allocation:

- On-demand provisioning with **pay-as-you-go pricing**.
- Easily scales up or down based on need.

Supercomputer vs Personal Computer

Compute Node **vs** **Desktop**



A node is
~5x faster
than a desktop

Core Components of a Supercomputer

Nodes

- **Login (Head) Nodes**
  - Where users connect, prepare jobs, and submit them to the scheduler.
  - **Do not run compute jobs** here—meant for lightweight tasks.
- **Compute (Worker) Nodes**
  - Dedicated for running user jobs.
  - Can include specialized hardware like GPUs for intensive workloads.

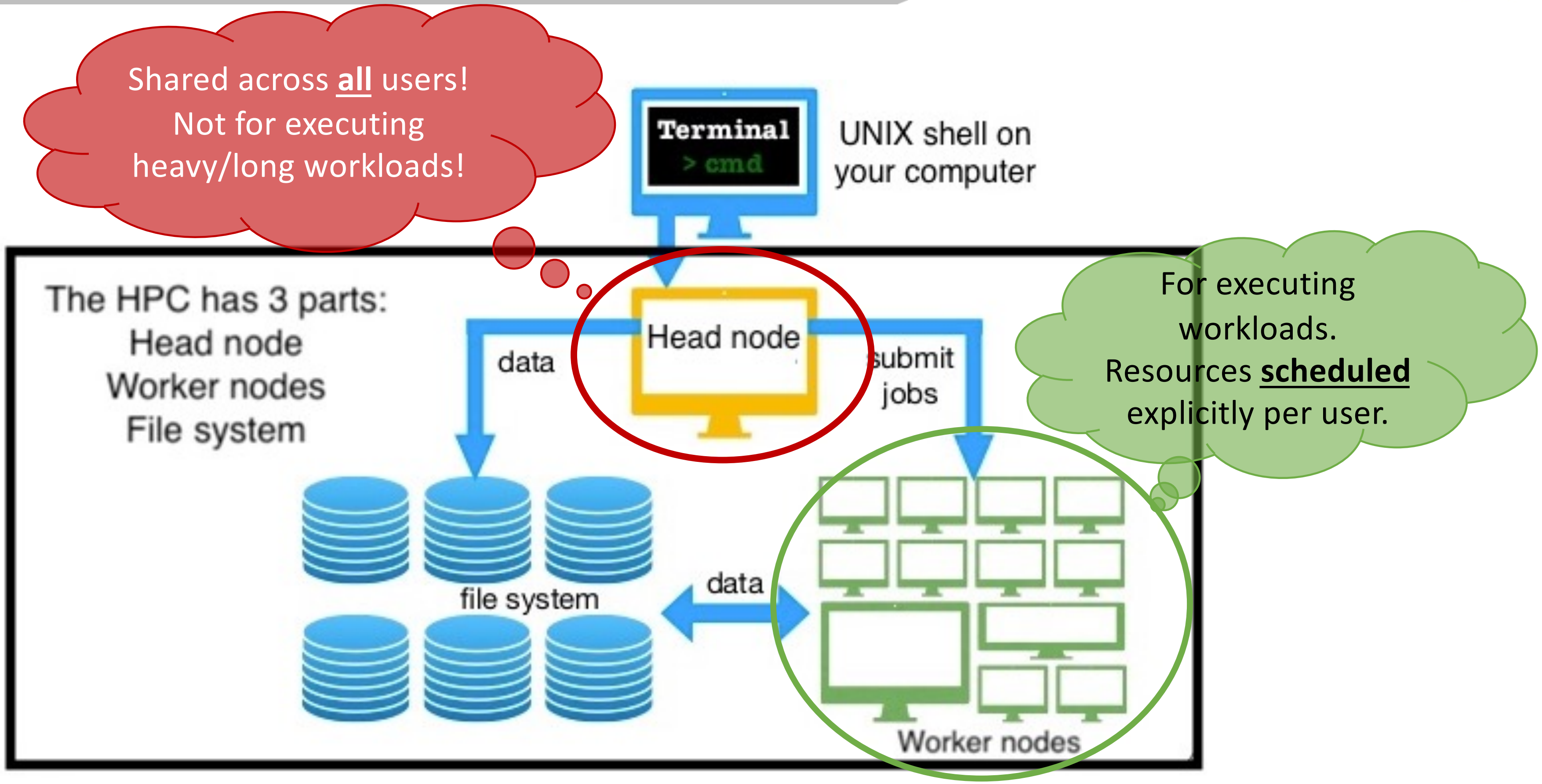
Interconnects

- **High-speed** networks connecting compute nodes in a supercomputer.
- Ensures low-latency, high-bandwidth data transfer critical for parallel computing.
- **Examples**: InfiniBand, Omni-Path.

Storage

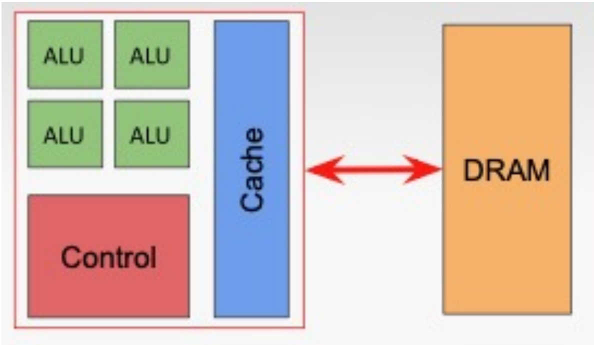
- **Local Storage**
  - **Temporary**, node-specific, fast.
  - Best for job-specific scratch data.
- **Shared Storage**
  - Accessible across nodes, larger capacity, slightly slower.
  - Used for shared datasets and outputs.

The Cluster Architecture



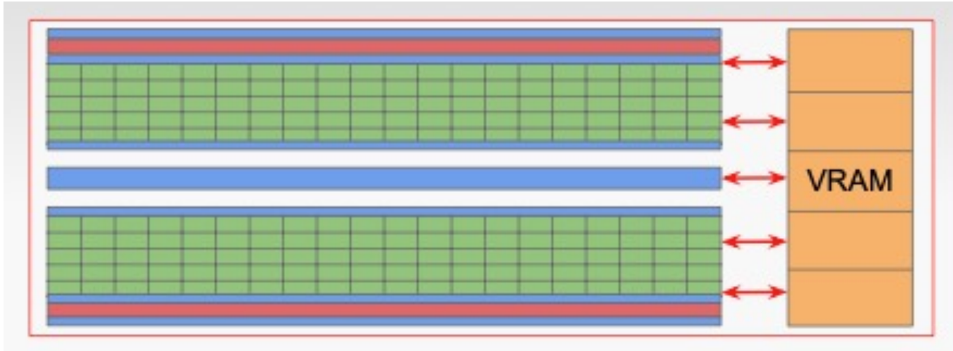
Nodes – Processors: Low Latency or High-throughput?

CPUs (Central Processing Units)

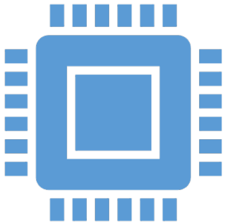


- **Optimised for low-latency** access to cahce data
- Complex control logic
- Large caches (L1, L2, etc.)
- Optimised for serial operations
- Newer CPUs have more parallelism
  - becoming more GPU-like

GPUs (Graphics Processing Units)



- **Optimised for data-parallel** throughput computation
- High latency tolerance
- High compute density per memory access
- High throughput
- Newer GPUs have better control logic
  - becoming more CPU-like



Memory

RAM (Random Access Memory):

- Fast, temporary storage for running programs.
- More RAM = Larger problem sizes.

Storage

Local Storage

- Temporary, node-specific, fast.
- Best for job-specific scratch data.

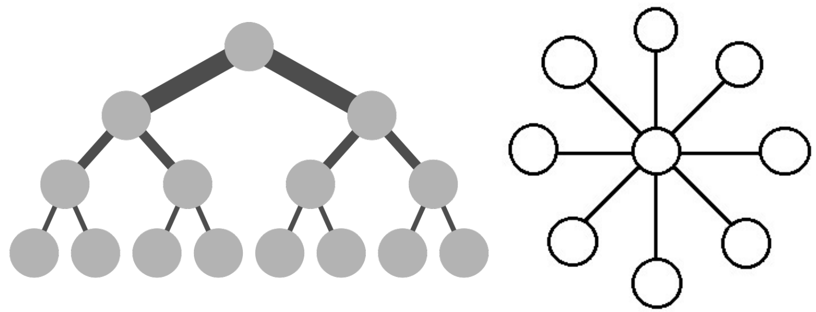
What are Interconnects?

- High-speed networks connecting compute nodes in a supercomputer.
- Designed for **low latency** and **high bandwidth** communication.

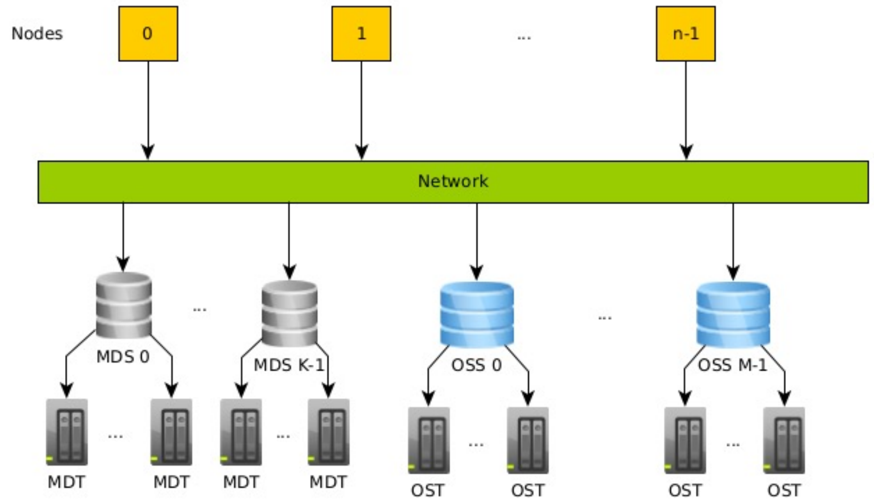
Why are Interconnects Critical?

- Enable efficient **data exchange** for parallel computing tasks.
- Support scalability: Essential for workloads distributed across thousands of nodes.

Feature	HPC Interconnects	Standard Networks
Latency	Very low (<1 ms)	Higher (ms)
Bandwidth	Very high (up to 400Gbps)	Moderate
Topology	Custom (e.g., fat-tree)	Standard (e.g., star)



Storage: File System Overview



Directory Structure:

1. **Home Directory**:
    - Persistent storage for scripts and small files.
    - Limited capacity.
2. **Scratch Directory**:
    - High-speed, temporary storage for jobs.
    - Files are **deleted after a set retention period.**
3. **Shared Directories**:
    - Large, slower storage for shared datasets or collaboration.

Best Practices:

- Use **scratch** for active jobs.
- Back up important data from **scratch** to **home** or external storage.

How to Access Cyclone

- Use **Secure Shell (SSH)** to log in to the supercomputing system.
- Requires a **username** and **password** or **key-based authentication**.

Tools for SSH by Platform:

- **Linux/Mac**: SSH is pre-installed; use the terminal.
- **Windows**:
  - Use tools like **PuTTY** or **Windows Subsystem for Linux (WSL)** for SSH access.
  - **Tip**: Install OpenSSH through PowerShell or the Settings menu.
    - Requires **Administrative priviledges**!
- **Example SSH Command:**

```
$ ssh -i /path/to/ssh/private/key cstyl@cyclone.hpcf.cyi.ac.cy
```

Accessing the System

```
[cstyl@cstyl-2:~$ ssh -i ~/.ssh/cyclamen cstyl@cyclone.hpcf.cyi.ac.cy
   _____           _
  / ____|         | |
 | |    _   _  ___| | ___  _ __   ___
 | |   | | | |/ __| |/ _ \| '_ \ / _ \
 | |___| |_| | (__| | (_) | | | |  __/
  \_____\__, |\___|_|\___/|_| |_|\___|
         __/ |
        |___/


NONE of the data on this system is backed up. Please keep backups of valuable
 files at alternative locations. Maintaining backups is the sole responsibili
ty of the user.


Welcome to the Cyclone HPC system!
- System Documentation: https://hpcf.cyi.ac.cy/documentation/index.html
- Support: Email to hpc.support@cyi.ac.cy or open a ticket here:
  https://hpcf-support.atlassian.net/servicedesk/customer/portal/3
- Use "module avail" and "module load" to list/use available software
- You can now use the "qhist" command to check your project's usage
- Utilize --reservation=short for 1-hour jobs on specific nodes during
  work hours for no queue time

Last login: Thu Nov 21 16:06:56 2024 from 46.199.227.111
(base) [cstyl@front02 ~]$
```

Key Linux Commands:

Command	Purpose	Example
ls	List directory contents	`ls -l`
cd	Change directory	`cd /path/to/directory`
pwd	Print the current directory path	`pwd`
cp	Copy files or directories	`cp file1 file2`
mv	Move or rename files or directories	`mv oldname newname`
rm	Remove files (use cautiously!)	`rm file`

Best Practices:

- Use `ls` and `pwd` to verify your current location.
- Avoid running `rm` without confirming the file path.

Navigating Directories

```
(base) [cstyl@front02 ~]$ pwd
/nvme/h/cstyl

(base) [cstyl@front02 ~]$ ls -l
total 3783350
lrwxrwxrwx  1 cstyl qcd          15 Jun 14  2022 data_p069 -> /onyx/data/p069
lrwxrwxrwx  1 cstyl qcd          15 Jul 10  2023 data_p163 -> /onyx/data/p163
lrwxrwxrwx  1 cstyl qcd          15 Jul 11  2023 data_p165 -> /onyx/data/p165
lrwxrwxrwx  1 cstyl qcd          20 Apr 17  2024 edu20 -> /nvme/scratch/edu20/
drwxr-xr-x  4 cstyl qcd           9 Apr 27  2024 gpt-fast
lrwxrwxrwx  1 root  root         20 Mar 11  2024 scratch -> /nvme/scratch/cstyl/
drwxr-xr-x 13 cstyl qcd          15 Apr 16  2024 wee_archie
```

File System Structure

Directory Overview

Home Directory: Persistent storage for scripts and small files.

$HOME=/nvme/h/<username>

Scratch Directory: Temporary, high-speed storage for job data.

$SCRATCH=/nvme/scratch/<username>

Shared Directory: Collaboration space for team projects.

$DATA_<pid>=/onyx/data/<pid>

Best Practices for File Management

Store **active jobs** in the **scratch directory**.

Store **source code** and **build executables** in **home directory**.

Store **large** shared project **data** in **shared directory.**

Move important results to **home** or external storage to prevent loss.

Note: NO BACKUPS

What Are Modules?

- **Modules** manage software environments, ensuring compatibility with HPC resources.
- Load, switch, or unload software dynamically.
- **Note: No <u>sudo</u> access on HPC Systems!**

Key Commands:

Command	Purpose	Example
module avail	List all available software modules	module avail
module load	Load a specific software module	module load gcc
module unload	Unload a specific module	module unload gcc
module list	List currently loaded modules	module list

Tip: Check module dependencies to avoid conflicts.

Managing Modules

```
(base) [cstyl@front02 ~]$ python --version
Python 3.10.13

(base) [cstyl@front02 ~]$ module avail Python/
------------------------------ /eb/modules/all ------------------------------
   Python/2.7.16-GCCcore-8.3.0
   ...
   Python/3.10.8-GCCcore-12.2.0
   Python/3.11.5-GCCcore-13.2.0

(base) [cstyl@front02 ~]$ module load Python/3.11.5-GCCcore-13.2.0

(base) [cstyl@front02 ~]$ python --version
Python 3.11.5
```

A Note on Alternative Editors and IDEs

Why Use Alternative Tools?

- Enhance **productivity** with **user-friendly interfaces**.
- **Simplify file management** and script editing for supercomputing workflows.
- Provide advanced features like **SSH integration**, **file transfer**, and **remote execution**.

Tool	Purpose	Best For	Platform
VS Code	Remote editing and debugging	Writing scripts, interactive debugging, SSH access	**Windows, Mac, Linux**
MobaXTerm	SSH client with GUI	SSH access, file management	Windows
FileZilla	File transfer	Transferring data to/from supercomputers	Windows, Mac, Linux

What is SLURM?

Definition

SLURM (Simple Linux Utility for Resource Management) is a job scheduler that allocates resources and manages workloads on supercomputers.

It ensures efficient sharing of resources among multiple users.

Key Roles

Resource Allocation: Assigns CPUs, memory, GPUs, and other resources to jobs.

Job Scheduling: Prioritizes jobs based on factors like resource availability and queue position.

Monitoring: Tracks the status of jobs and manages failures or cancellations.

Why SLURM?

Maximizes system utilization.

Ensures **fair resource distribution**.

Writing a Basic SLURM Script

A SLURM script is a text file with commands that specify:

1. **Job Information**: Job name, output files, email notifications.
2. **Resource Requests**: CPUs, memory, GPUs, and wall time.
3. **Execution Commands**: The actual program or script to run.

```
#!/bin/bash
#SBATCH --job-name=example_job    # Job name
#SBATCH --output=output.txt       # Output file
#SBATCH --error=error.txt         # Error file
#SBATCH --ntasks=1                # Number of tasks (CPUs)
#SBATCH --mem=4G                  # Memory allocation
#SBATCH --time=01:00:00           # Time limit (hh:mm:ss)

module load Python                # Load required module

python script.py                  # Run Python script
```

Submitting and Monitoring Jobs

Example Workflow:

1. Submit a job using `sbatch`.
2. Monitor progress with squeue.
3. Cancel if necessary with `scancel`.

Job Status Symbols:

- **R**: Running.
- **PD**: Pending (waiting in queue).
- **F**: Failed.

Category	Command	Action
Job Submission	sbatch <job_script>	Submit a batch job
	srun <command>	Run a command in parallel
Job Monitoring	squeue	View queued jobs
	squeue --user=$USER	View MY queued jobs
	scontrol show job <job_id>	Detailed job info
Job Management	scancel <job_id>	Cancel a job
	scontrol hold <job_id>	Hold a job
	scontrol release <job_id>	Release a held job
Resource Allocation	sinfo	Information about nodes and partitions
	scontrol show node <node_id>	Show Node details

Best Practices:

- Request only what your job needs to avoid wasting resources.
- Use `--time` to limit job runtime and prioritize efficiency.
- Use `--account` to specify which project to "charge".

Resource	Description	Job Specification
Nodes	Number of Nodes	`--nodes`
CPU tasks	Number of CPU tasks per node	`--ntasks-per-node`
CPU threads	Number of CPU threads (cores) per task	`--cpus-per-task`
Memory	Amount of RAM per Job	`--mem`
GPUs	Request for GPUs if needed	`--gres`
System Partition	Either use the CPU or GPU part of the system	`--partition`

Example Use Case: Running a Python Script

Command Workflow:

1. Write the submission script (python_job.sh)
2. Submit: **sbatch** python_job.sh
3. Monitor: **squeue** -u <username>
4. Cancel (if needed): **scancel** job_id

Output:

- Job results and logs will appear in python_output.txt, errors in python_error.txt.
- Check logs for troubleshooting if the job fails.

```bash
#!/bin/bash
#SBATCH --job-name=python_job
#SBATCH --output=python_output.txt
#SBATCH --error=python_error.txt

#SBATCH --partition=cpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --time=00:30:00

module load Python
python example_script.py
```

Example of squeue Output

```
[(base) [cstyl@front02 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
  998469_[19-25%1]       cpu CNO_300K sbhowmic PD       0:00      1 (JobArrayTaskLimit)
   1007611_[4-5%1]       cpu Ag1H+_D1 sbhowmic PD       0:00      1 (JobArrayTaskLimit)
           1008164       cpu enthalpy  cy22kp1 PD       0:00      2 (Dependency)
           1008163       cpu analysis  cy22kp1 PD       0:00      1 (Dependency)
 1008403_[141-146]       cpu nanli44c  cy24nl1 PD       0:00      1 (Priority)
 1008402_[131-136]       cpu nanli44c  cy24nl1 PD       0:00      1 (Priority)
   1008392_[66-68]       cpu nanli44c  cy24nl1 PD       0:00      1 (Resources)
         998469_18       cpu CNO_300K sbhowmic  R   15:45:59      1 cn09
         1007611_3       cpu Ag1H+_D1 sbhowmic  R   10:08:58      1 cn02
           1008334       cpu data_min  jmoreno  R   23:43:31      1 cn06
           1008525       cpu       zn  eg20ra1  R   17:30:39      2 cn[15-16]
        1008392_65       cpu nanli44c  cy24nl1  R    7:51:38      1 cn13
        1008392_63       cpu nanli44c  cy24nl1  R   14:24:14      1 cn05
        1008392_64       cpu nanli44c  cy24nl1  R   14:24:14      1 cn12
        1008392_61       cpu nanli44c  cy24nl1  R   17:12:05      1 cn14
        1008392_62       cpu nanli44c  cy24nl1  R   17:12:05      1 cn17
        1008392_59       cpu nanli44c  cy24nl1  R   17:32:16      1 cn03
        1008392_60       cpu nanli44c  cy24nl1  R   17:32:16      1 cn04
```

Why Jupyter Notebooks on HPC?

Advantages of Jupyter Notebooks:

- **Interactive Workflows**: Combines code, visualizations, and narratives in a single interface.
- **AI and Data Analysis**: Perfect for tasks like data exploration, visualization, and model development.
- **Ease of Use**: Intuitive, web-based platform that lowers the learning curve for complex workflows.

Why Use HPC for Jupyter?

- Access to powerful compute resources (e.g., CPUs, GPUs).
- Handle large-scale datasets beyond the capabilities of local machines.
- Perform **AI training** and **numerical simulations** interactively and efficiently.

Launching Jupyter on HPC

How It Works:

- Use a **SLURM script** to allocate HPC resources for the Jupyter session.
- Establish a **tunnel** between the HPC system and your local machine to access the notebook in your web browser.

Workflow Steps:

1. Write a SLURM script to launch Jupyter.
2. Submit the script with sbatch.
3. Create an SSH tunnel to forward the notebook port.
4. Open the notebook in your browser using the forwarded port.

Writing a Basic SLURM Script to Launch Jupyter

```bash
#!/bin/bash
#SBATCH --job-name=jupyter_notebook
#SBATCH --output=jupyter_%j.log
#SBATCH --ntasks-per-node=1                    # Single task for the notebook
#SBATCH --cpus-per-task=4                      # Allocates 4 CPU cores
#SBATCH --mem=8G                               # Reserves 8 GB of memory
#SBATCH --gres=gpu:1                           # Requests a single GPU
#SBATCH --time=02:00:00                        # Runs session for two hours

# Assume python and jupyter modules exist on system
module load python
module load jupyter

# Launches Jupyter on the specified port (8888) without a browser.
jupyter-notebook --no-browser --port=8888
```

Establishing the SSH Tunnel

What is SSH Tunneling?

- A method to securely forward ports from the HPC system to your local machine.

Command for SSH Tunneling: Assume job is running on **`gpu01`** compute node

```
$ ssh -N -J cstyl@cyclone.hpcf.cyi.ac.cy cstyl@gpu01 -L 8888:localhost:8888
```

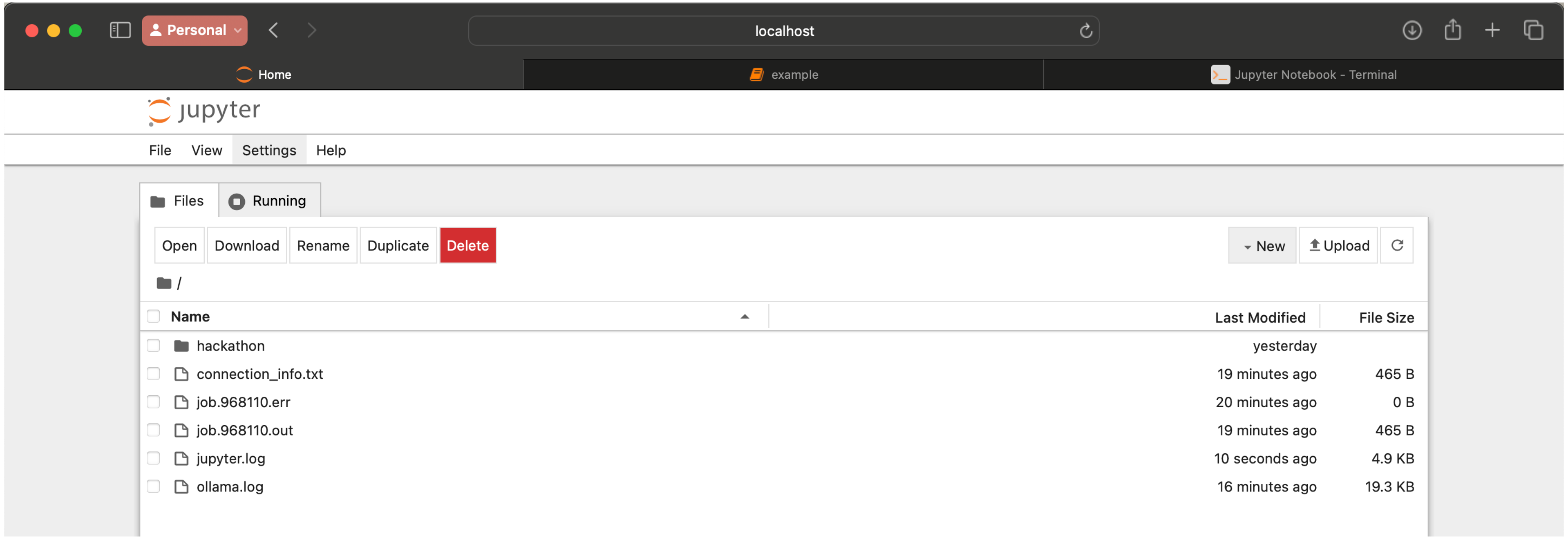
- **8888**: Local and remote port for the Jupyter Notebook.
- Replace username with your HPC username.

Steps to Access Jupyter:

1. Copy the Jupyter Notebook link provided in the SLURM log (e.g., http://localhost:8888).
2. Open the link in your local browser.
3. Authenticate using the token in the log output.

Launching a Jupyter Notebook on Cyclone – Open On Browser

- After the SSH Tunnel has been established, open your browser and enter the link to Jupyter Notebook. E.g.,:
  - `http://localhost:<port>/?token=<token>`



Summary and Key Takeaways

- Supercomputers and cloud systems serve different purposes (**capability computing** vs. **capacity computing**).
- Supercomputers allow us to solve larger and more complex problems beyond the capabilities of personal computers or workstations
  - Address challenges of insufficient memory and compute resources.
- Efficient navigation, resource management, and job scheduling are essential for success.
- Tools like Jupyter and alternative IDEs enhance **usability** and **productivity**.
- **Next on Hands-on Session:** Apply these concepts to efficiently navigate and utilize Cyclone for your projects.

Thank you for your attention!

More information:

https://eurocc.cyi.ac.cy/
https://www.linkedin.com/company/eurocc2

Contact us at:
eurocc-contact@cyi.ac.cy

Co-funded by the European Union

Republic of Cyprus

RESEARCH & INNOVATION FOUNDATION

EuroHPC Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus (co-funded by the EU within the framework of the Cohesion Policy Programme “THALIA 2021-2027”), Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.