Introduction to HPC usage

Working on WEXAC for AI hub users

Amir Kleiner

18/03/2025

Todays Agenda

1. Python scripts
2. Dealing with linux
3. Basic bash scripting
4. HPC basics
5. Schedulers and how to use them

Writing Python scripts

If it Ain't broke, why fix it?

Jupyter	Scripts
Intuitive(ish) usage	(Sometimes) less intuitive
Easier to debug	Harder to debug
Good for data/code exploration	More cumbersome
Possibly unorganized code	Enforced top-bottom organization
Slower performance	Faster performance
Bad parallelism	Not so bad parallelism
Short session times	Longer session times

So how do we write scripts?

Just like writing Jupyter, with all the code in a “single cell”

Runtime: 62.4 s

```
[1]: import numpy as np
     import multiprocessing as mp
     import time
     import sys

[2]: def slow_function(_):
         """A simple function that runs slower in Jupyter."""
         np.random.seed()  # Avoid repeatability in multiprocessing
         A = np.random.rand(10000, 1000)
         B = np.random.rand(1000, 10000)
         return np.linalg.slogdet(A @ B)[1]  # Compute the log determinant

[3]: num_tasks = 10  # Reduce in Jupyter, increase in a script
     ncpus = 10
     start = time.time()

     # Detect if running in Jupyter
     in_jupyter = "ipykernel" in sys.modules

     if in_jupyter:
         # Simulate overhead in Jupyter (worse multiprocessing performance)
         results = list(map(slow_function, range(num_tasks)))
     else:
         # Efficient multiprocessing in script mode
         with mp.Pool(processes=ncpus) as pool:
             results = pool.map(slow_function, range(num_tasks))

     end = time.time()

     print(f"Time taken: {end - start:.2f} seconds")
     print(f"Sum of results: {sum(results)}")
```

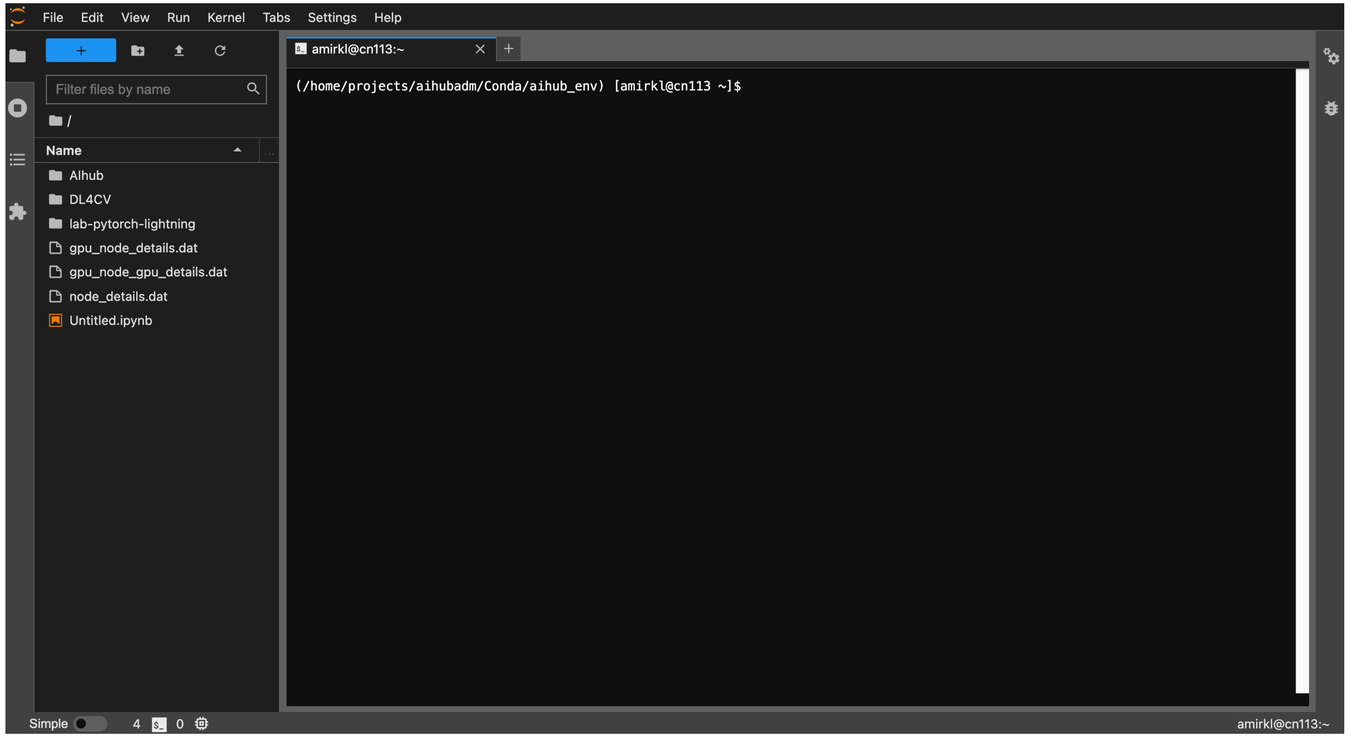
Runtime: 43.2 s

```
1  import numpy as np
2  import multiprocessing as mp
3  import time
4  import sys
5
6  def slow_function(_):
7      """A simple function that runs slower in Jupyter."""
8      np.random.seed()  # Avoid repeatability in multiprocessing
9      A = np.random.rand(10000, 1000)
10     B = np.random.rand(1000, 10000)
11     return np.linalg.slogdet(A @ B)[1]  # Compute the log determinant
12
13 if __name__ == "__main__":
14     num_tasks = 10  # Reduce in Jupyter, increase in a script
15     ncpus = 10
16
17     start = time.time()
18
19     # Detect if running in Jupyter
20     in_jupyter = "ipykernel" in sys.modules
21
22     if in_jupyter:
23         # Simulate overhead in Jupyter (worse multiprocessing performance)
24         results = list(map(slow_function, range(num_tasks)))
25     else:
26         # Efficient multiprocessing in script mode
27         with mp.Pool(processes=ncpus) as pool:
28             results = pool.map(slow_function, range(num_tasks))
29
30     end = time.time()
31
32     print(f"Time taken: {end - start:.2f} seconds")
33     print(f"Sum of results: {sum(results)}")
```

Dealing with Linux

Terminals, shells and the linux command line

- A ***shell*** is the interface to the operating system.
- A ***terminal*** is a program that runs a shell.
- There are many shell options. In windows PCs **MobaXterm** is recommended. I n Macs **iterm2** is recommended. We will use today **JupyterHub**.
- The ***command line interface (CLI)*** is a text-based way for interacting with a computer program. This is more powerful than graphical interfaces, but has a learning curve.



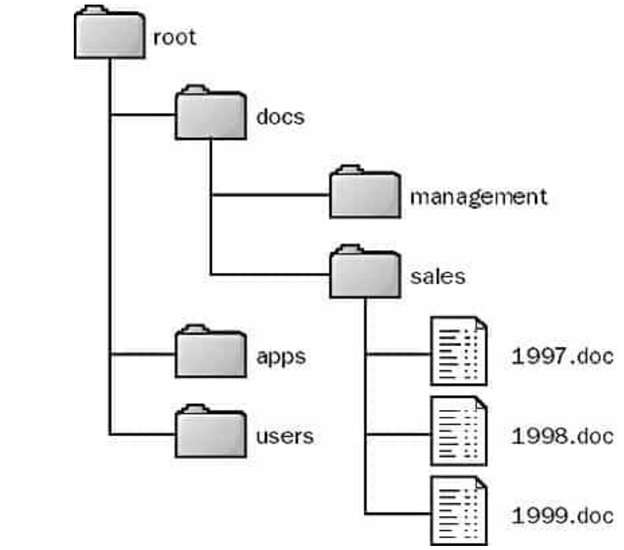
Exercise

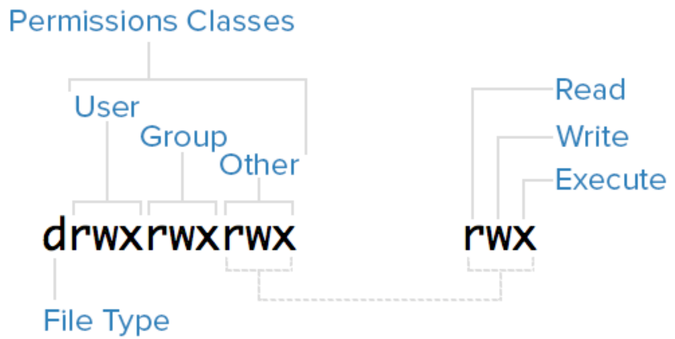
Setup - getting the code for today

1. Log in to WEXACs Jupyter hub
2. Open a terminal
3. Type `git clone https://github.com/ai-hub-weizmann/ex-using_hpc.git`

Paths and File systems

- A **directory (folder)** is an artificial “place” where files are stored.
- A **directory tree (file system)** is a hierarchy of directories.
  The directory in which we are now is called *current*
  A directory inside another is called a *child*
  A directory containing another is called a *parent*
- The **root (home)** directory is the first, basic, parentless directory.
- The **path** is a string of characters used to identify/locate something in a file system.
  Directories are divided by a `/` character.
  - Absolute paths begin at the root. They always begin with a `/`.
  - Relative paths begin at the current directory. They never begin with a `/`
- Hidden files/folders are marked by `.<name>`. They don’t appear with ls unless specifically requested.





CLI - getting started

Moving around

- **pwd** (**p**rint **w**orking **d**irectory) - displays the absolute path to the current directory.
- **cd** (**c**hange **d**irectory) <path> - moves to the directory specified by <path>
- **mkdir** (**m**a**k**e **dir**ectory) <path> - creates the non-existing directory specified by <path>
- **rmdir** (**r**e**m**ove **dir**ectory) <path> - removes the empty directory specified by <path>

Two important aliases

- `./` - marks the current directory
- `../` - marks the parent directory

CLI - getting started

Files

- **ls** (**l**i**s**t) <path> - prints to the screen all files and folder in the specified directory
- **cp** (**c**o**p**y) <path1> <path2> - copies file from path1 to path2
- **mv** (**m**o**v**e) <path1> <path2> - moves file/folder from path1 to path2
- **touch** <path> - makes an empty file at specified path. If file exists it updates the time stamp
- **rm** (**r**e**m**ove) <path> - deletes the specified file
- **echo** <string> - prints specified string to screen
- **cat** <path> - prints contents of specified file
- **less** <path> - opens specified file for reading

CLI - getting started

Modifying command behavior with flags/options

- The behavior of commands can be modified by adding “flags” (options) after the command
- `--` is used before keyword options. `-` is used for abridged options. There is some overlap...
- `--help` or `h` usually lists a short summary of command functionality and available flags
- Flags for `cp`
  - `-r` recursively copies all files in path (also folders)
- Flags for `rm`
  - `-r` recursively removes files. Allow deleting folders.
  - `f` forces removal of files without asking for comfirmation
- Flags for `ls`
  - `-a` or `--all` shows all files, including hidden
  - `-l` displays files in a longer format, including permissions, owner, date of modification, size, etc.
  - `-h` displays file sizes in human-readable format

CLI - getting started

Chaining and redirection

- The output of most commands can be made to either be the input to another command (chaining) or to be written in a file (redirection).
- **`>`** - writes the output of commands on the left to the file on the right (if file exists this deletes previous content!)
- **`>>`** - appends the output commands on the left to the file on the right (if file exists this does not delete previous content!)
- **`<`** - uses content of file on the right as input for commands on the left
- **`|`** - uses utput of command on the left as input for command on the right

Exercise

Using the CLI

1. Navigate to the workshop directory
2. Type in `tar -xzvf find_the_file.tar.gz`
3. Read the contents of the `lorem_vimpsum.txt` file
4. Copy the `lorem_vimpsum.txt` file
    1. Add some text to this file
    2. Add the list of files in the current directory to the end of this file
5. Navigate to the `dirtree` directory
    1. Try to follow the instructions

Basich bash scripting

Scripts

Basics

- Scripts allow us to perform multiple operations, sometimes very complex, with a single command.
- Extensions are arbitrary. By convention we tend to use `.sh`
- We run the script by calling `bash <path to script>`.
  - To make sure the script runs with bash, every script begins with `#!/bin/bash`

Scripts

Variables and comments

- Define a variables by `variable_name=value`
- Access the value of a variable with `$variable_name`
- Define arrays by `array_name=(value1, value2, ..., valueN)`
- Define a list of consecutive numbers with `conseq_nums=({start..end..jump})`
- Access element *idx* of an array by `${array_name[*idx*]}` This is 1-indexed.
  - Access all elements at once by either `@` or `*`
- Comment a line with `#`

Scripts

Arithmetics

- We will use simple arithmetics in some of our scripts
- To evaluate an expression it has to be wrapped by `$((<expression>))`
- We can use variables as part of the expression.
- The supported arithmetic operations are:

`+` (addition)	`-` (subtraction)	`*` (multiplication)
`**` (exponentiation)	`/` (division)	`%` (modulus)

Scripts

Logical operations

- Logical operations compare two numbers and return a value of `true` or `false`
- The general syntax of a logical operation is **`num1 -<logical operation> num2`**
- The available operations are:

`-eq` (equal)	**`-gt`** (greater than)	**`-ge`** (greater or equal)
`-ne` (not equal)	**`-lt`** (lower than)	**`-le`** (lower or equal)

Scripts

If statements

- `if` statements determine the flow of code, by executing different code depending on if a criterion is met or not.
- The syntax of an if-block is:

```
if [ conditions ]
   then
      commands
   elif [ conditions ]; then
      commands
   else
      default commands
fi
```

- Any number of elif blocks
  Can be added
- ***Spaces are important!***

Scripts

Loops

- Bash has bot ***for*** and ***while*** loops, but we will focus only on ***for*** loops here.
- The general syntax of a for loop is:

```
for element in ${list[*]}
  do
     Commands
  done
```

- The list can be either a variable or provided on the spot.

Exercise

Simple scripting

- Try as many of the following as you can:

    1. Make a script that prints all numbers from 1 to 10, each in a new line

    2. Make a script that lists all the files in the directory, each in a new line

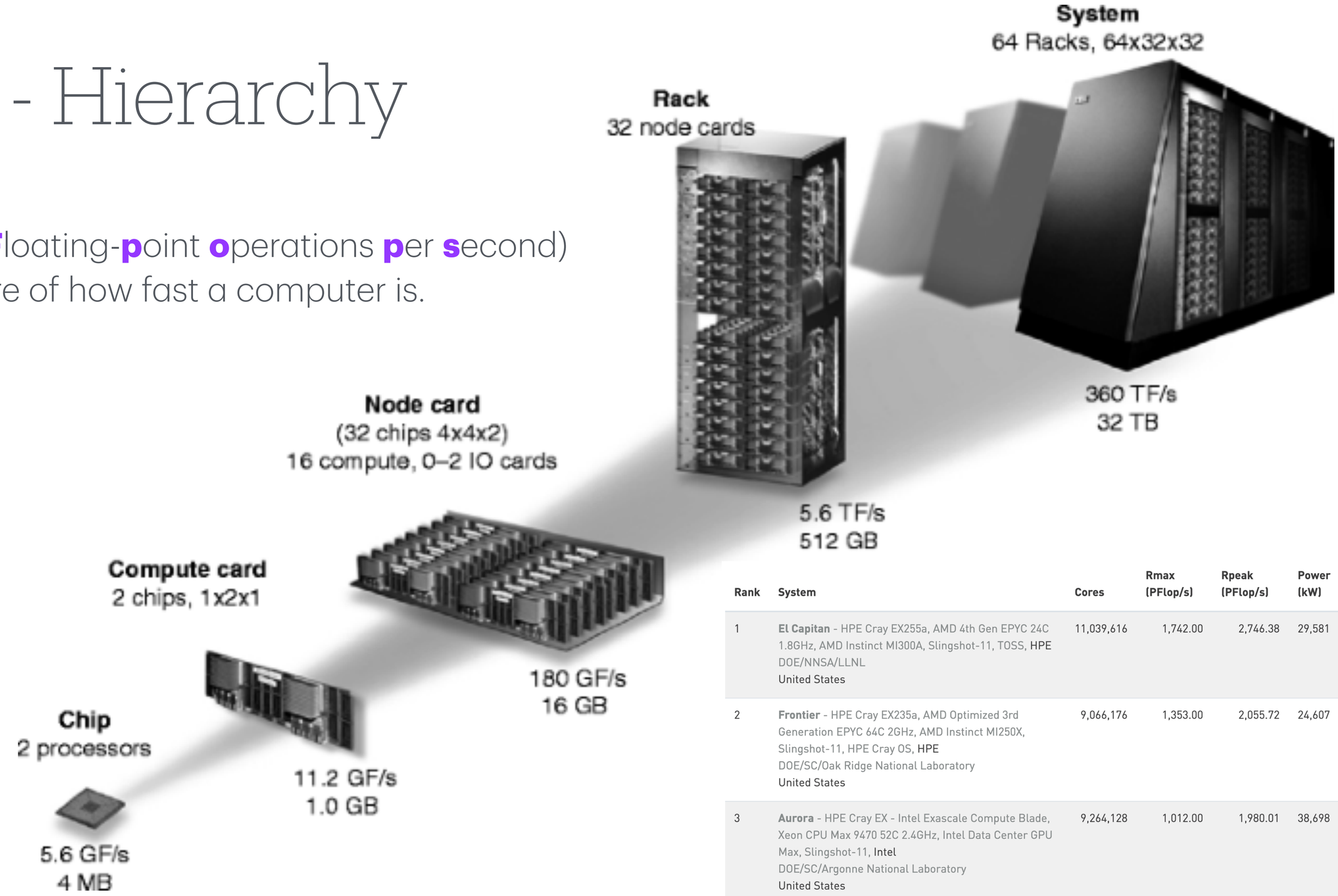
        1. Add `_copy` to ech file before printing its name

        2. Make directory called `copies` and copy to there all files (with `_copy` at the end)

Basics of
High-Performance Clusters

HPC - Hierarchy

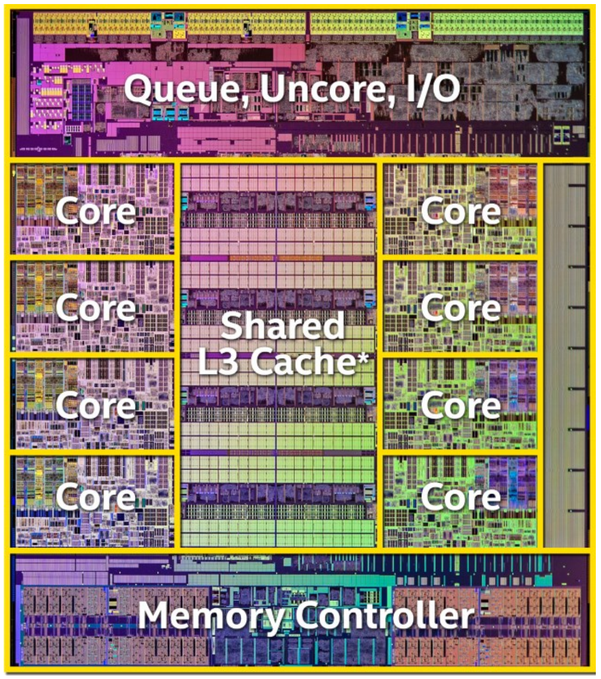
- **FLOPS** (**F**loating-**p**oint **o**perations **p**er **s**econd)
  A measure of how fast a computer is.

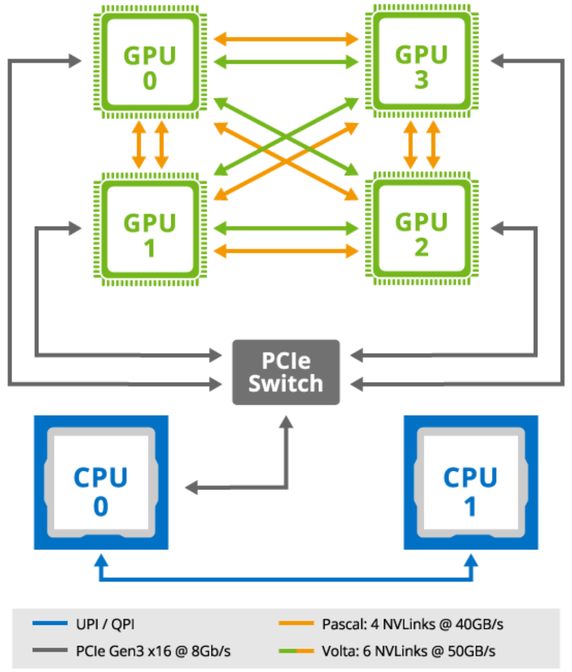


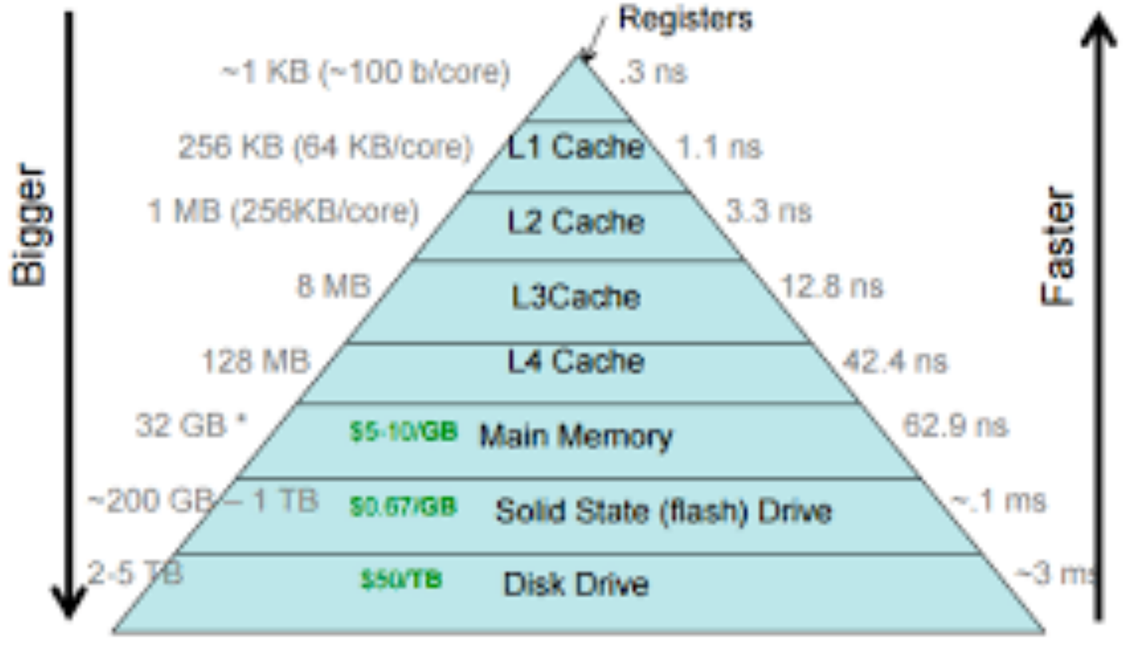
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698

Inside the node

- Resources:
  - CPU cores (num)
  - RAM (GB)
  - GPU units (num)
  - GPU memory (GB)
  - Time (h:m:s)

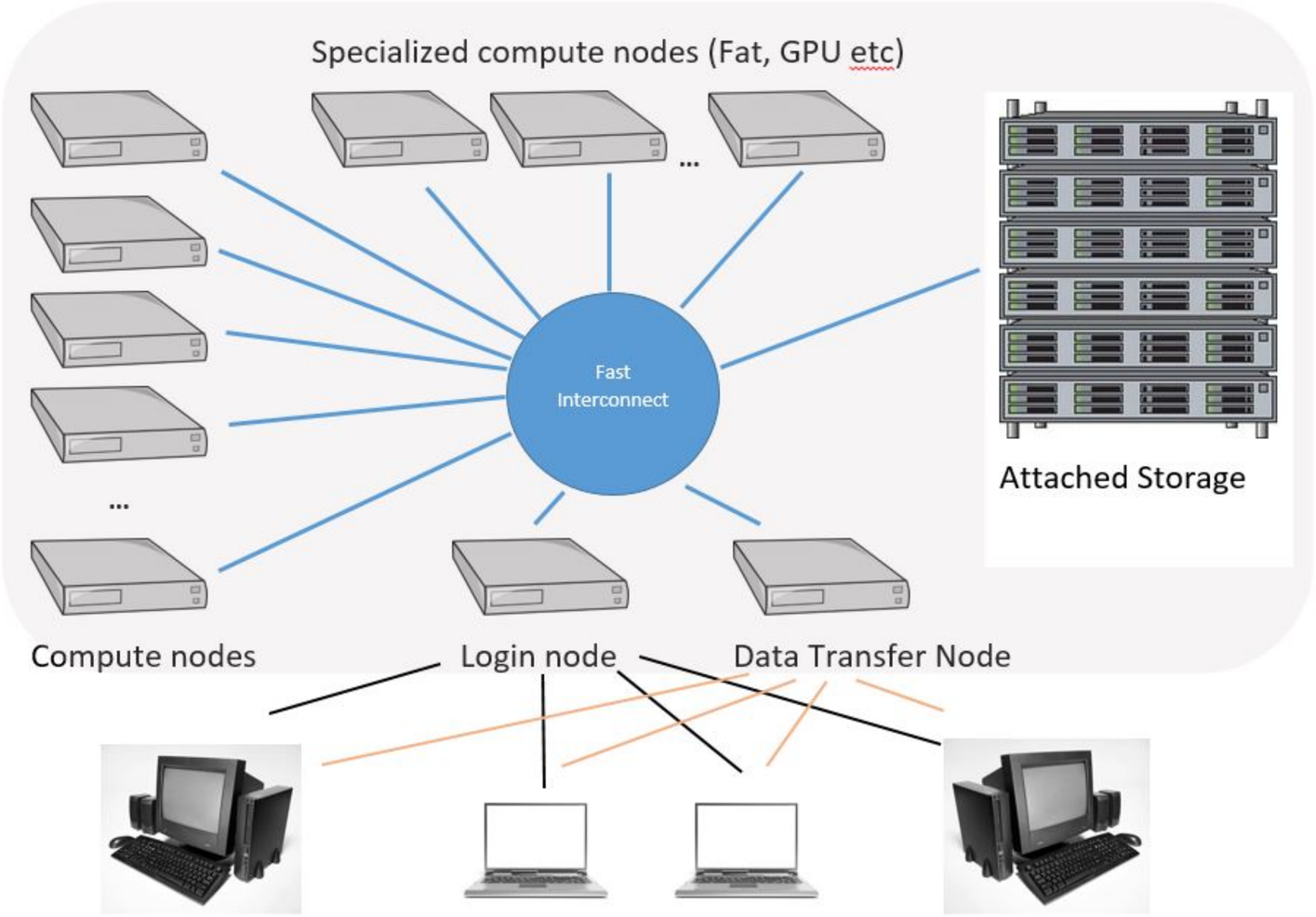






Typical HPC organization

- **Login nodes** - Few, shared by many. Good for human interaction, bad for computations.
- **Work/Compute nodes** - Most of the cluster. Optimized for long heavy-duty work. Not human friendly - little to no support for graphics and interactivity. Depending on cluster, may include GPU or other accelerations.
- **Interactive nodes** - Like compute nodes, but human friendly - with graphical support and interactivity. Limited to small and short calculations. Good for interactivity, code developement, and data analysis.
- **Data transfer nodes** - Like compute nodes but specifically optimized for data transfer.



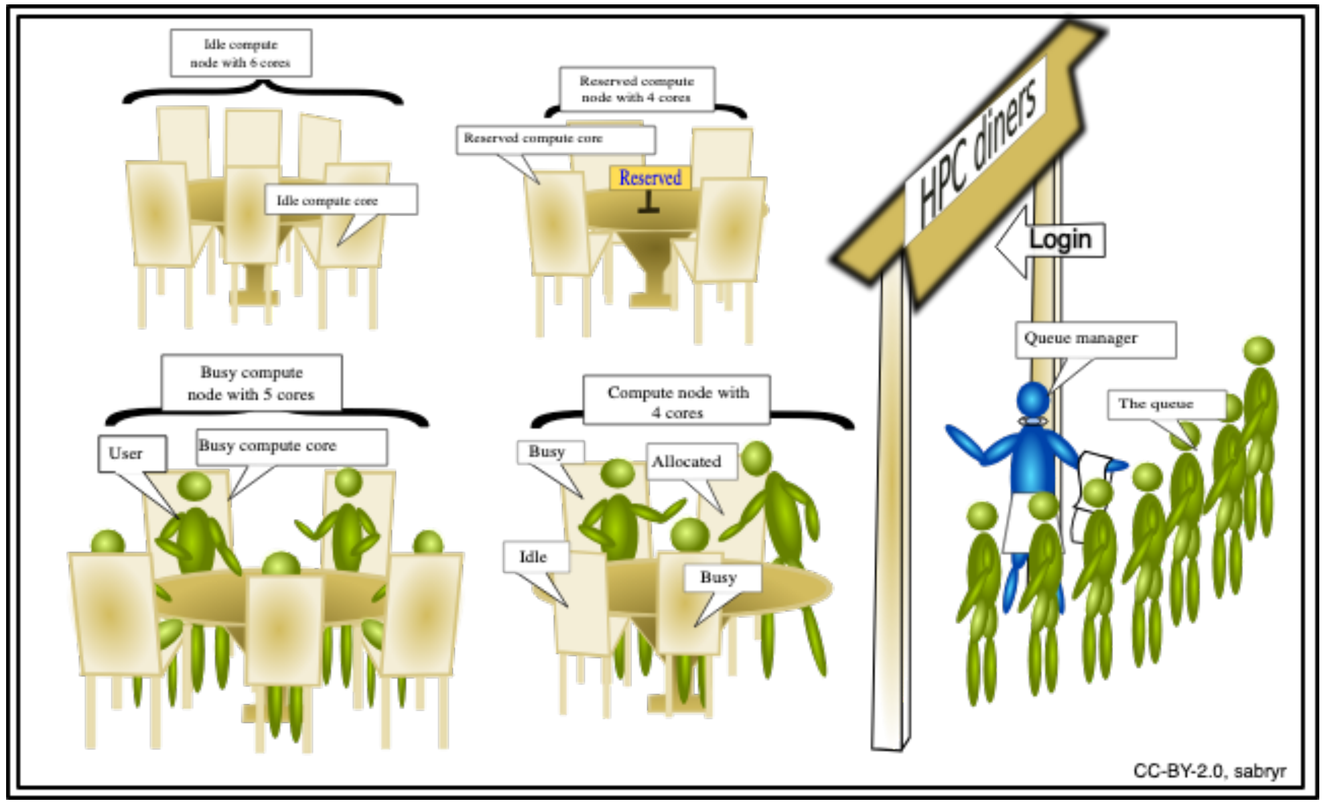
Schedulers and how to use them

Schedulers

Why do we need them?

WEXAC = LSF scheduler

- We need to balance:
  - Thousands-millions of cores
  - Hundred-thousands of users
  - Millions of jobs
  - Minimize waiting time
  - Maximize overall usage
  - Try to be fair with everybody
- These tough constrains require specialized help - The Scheduler



Schedulers

Queues

- Each queue has:
  - A name
  - Minimal and maximal amount of allowed resources (memory, cpus, gpus, time)
  - priority
  - Users that are allowed to use it
- **bqueues** - prints existing queues and their usage
  - -l <queue_name> - prints detailed information on specified queue
  - -u $USER - prints only queues you are allowed to use

```
(/home/projects/aihubadm/Conda/aihub_env) [amirkl@login1 workshop]$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
service-ood-amd 250  Open:Active       -   30    -    -     0     0     0     0
gsla-mem        240  Open:Active       -    -    -    -    32     0    32     0
gsla-cpu        240  Open:Active       -    -    -    -  3430   300  3130     0
gsla_high_gpu   240  Open:Active       -    -    -    -     3     0     3     0
service-ood     200  Open:Active       -   20    -    -    13     1    12     0
service-ood-int 200  Open:Active       -   30    -    -     0     0     0     0
talide-gpu      190  Open:Active       -    -    -    -     0     0     0     0
waic-short      189  Open:Active       -    -    -    -     2     1     1     0
waic-medium     189  Open:Active       -    -    -    -     0     0     0     0
waic-long       189  Open:Active       -    -    -    -     7     0     7     0
waic-risk       185  Open:Active       -    -    -    -   464   448    16     0
elinav          180  Open:Active       -    -    -    -     0     0     0     0
sorek-gpu       145  Open:Active       -    -    -    -     0     0     0     0
leeat-gpu       145  Open:Active       -    -    -    -     0     0     0     0
molgen-gpu      140  Open:Active       -    1    -    -     0     0     0     0
gpu-interactive  89  Open:Active       -   30    -    -     0     0     0     0
berg             80  Open:Active       -    -    -    -     0     0     0     0
bio              80  Open:Active       -    -    -    -     0     0     0     0
bio-pipe         80  Open:Active       -    -    -    -     0     0     0     0
schwartz         80  Open:Active       -    -    -    -     0     0     0     0
physics-long     79  Open:Active   - 1000    -    -    85     0    85     0
physics-medium   79  Open:Active   - 3000    -    -     0     0     0     0
physics-short    79  Open:Active   - 4000    -    -     0     0     0     0
interactive-gpu  75  Open:Active       -   30    -    -    30    13    17     0
short-gpu        74  Open:Active       -  800    -    -  6254  6043   211     0
long-gpu         74  Open:Active       -  500    -    -   398   287   111     0
risk-gpu         73  Open:Active   - 5000    -    -  1455  1063   392     0
interactive      72  Open:Active       -   30    -    -    46     3    43     0
short            71  Open:Active   - 22000    -    -   596   190   406     0
medium           71  Open:Active   - 7000    -    -  8827  2308  6437    82
long             71  Open:Active   - 3000    -    -  1395   120  1255    20
test-shlomit     70  Open:Active       -    -    -    -     0     0     0     0
test-alexey      70  Open:Active       -    -    -    -     0     0     0     0
pycourse         70  Open:Active       -    -    -    -     0     0     0     0
risk             70  Open:Active   - 50000    -    -     0     0     0     0
fleishman-prior  60  Open:Active       -   40    -    -     0     0     0     0
fleishman        50  Open:Active   - 1024    -    -     0     0     0     0
ulitsky          50  Open:Active       -    -    -    -     0     0     0     0
fleishman-servi  45  Open:Active       -    -    -    -     0     0     0     0
lost_and_found    1  Closed:Inact      -    0    0    -     1     1     0     0
```

Schedulers

Submitting jobs

- We use bash scripts to begin interactions with the scheduler

- First - a block of requests and requirements for the scheduler

  - **#BSUB -<option> <value>**
    note that the # is *not* a comment!

- Then - a bash script for

```
#!/bin/bash
#BSUB -J my_1st_job # Job name
#BSUB -q short        # Submit to the 'short' queue
#BSUB -n 8            # Request CPU cores
#BSUB -R "rusage[mem=16GB]" # Request memory
#BSUB -R “span[hosts=1]”  # Ensure all cpus on same node
#BSUB -W 1:30         # Set wall time limit to 1 hour 30 minutes
#BSUB -o output.%J    # Redirect standard output to a file (output.<job_id>)
#BSUB -e error.%J     # Redirect error output to a file (error.<job_id>)

source ~/.bashrc
conda activate aihub_env
python script.py
```

- Finally, we send the request to the scheduler

  bsub < <job script name>

Schedulers

Common bsub options

- **-J** - sets the name of the job
- **-q** - specifies the requested queue
- **-g** - specifies which group to bill (relevant if you have more than one group)
- **-n** specifies number of tasks to run (usually translates to number of CPUs)
- **-R** - requests specific resources
- **-o**/**-e** - redirects output/error to specified file

Schedulers

Job status

- We can check on the status of the job after we submitted it with

bjobs -u <user>

- Job status can be:

```
Every 2.0s: bjobs

JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
133090     amirkl  RUN   interactiv login1      cn242       *upyterhub Mar 16 12:32
167058     amirkl  PEND  short      login1                  my_1st_job Mar 16 14:43
```

PEND (pending)	**RUN** (running)	**DONE** (completed)
EXIT (terminated)	***SUSP** (suspended)	**WAIT** (waiting)

Schedulers

Manipulating jobs and information

- **bhist** - displays information about past jobs
- **bpeek** - displays output and errors of unfinished job
- **bkill** - kills (permanenly stops) unfinished job
- **bstop** - suspends (pauses) unfinished job
- **bresume** - resumes suspended unfinished job

Exercise

Running jobs on WEXAC

- Read and then run the `example1.job` file
  - Look at the output. How much time did the job take? How much memory did you use? How much time did the python part took?
- Copy the `example1.job` file, and modify it to run the `parallel_script_2.py` file
- Change the number of CPUs and Memory you request, and rerun the job.
  1. How do these changes affect the running time?
  2. How do these changes (should) affect the time in queue?

Bonus

Aliases

- An **alias** is a way to shorthand compelx operations.
- We define an alias by:

alias <shorthand>=‘<complex operation>’

- We can automatically add aliases to our default environment by adding them to the `~/.bashrc` file.
- We can also create an `~/.alias` file and add it to the `~/.bashrc`

Requesting GPUs

Requesting multiple nodes with multiple GPUs

- New keyword `**-gpu <string>**`:
  - `*num*` - requested number of GPUs
  - `*j_exclusive*` - requests exclusive use of GPU
  - `*gmem*` - requests amount of GPU memory
- Queue has to have GPUs
- If more than 1 GPU node is required, replace the `span` line with:
  #BSUB -R “span[ptile=<nnodes>]same[model]”
  - This replicates the resources <nnodes> time!
  - The `*same*` part ensures all resources are of the same model and type.

```
#!/bin/bash
#BSUB -J gpu_python_job   # Job name
#BSUB -q gpu-short            # Submit to a GPU queue (adjust if needed)
#BSUB -n 10             # Request 10 CPU cores
#BSUB -gpu “num=1:j_exclusive=yes:gmem=48GB”       # Request 1 GPU
#BSUB -R "rusage[mem=48GB]"  # Request memory
#BSUB -R "span[hosts=1]"    # Ensure all resources are on the same node
#BSUB -W 2:00             # Set wall time limit (adjust as needed)
#BSUB -o output.%J         # Redirect standard output to output.<job_id>
#BSUB -e error.%J          # Redirect error output to error.<job_id>

# Load necessary modules (if required)
# module load python/your_version
# module load cuda/your_version  # If CUDA is needed

# Activate your virtual environment (if needed)
# source /path/to/your/venv/bin/activate

# Run the Python script
python gpu_python_script.py
```