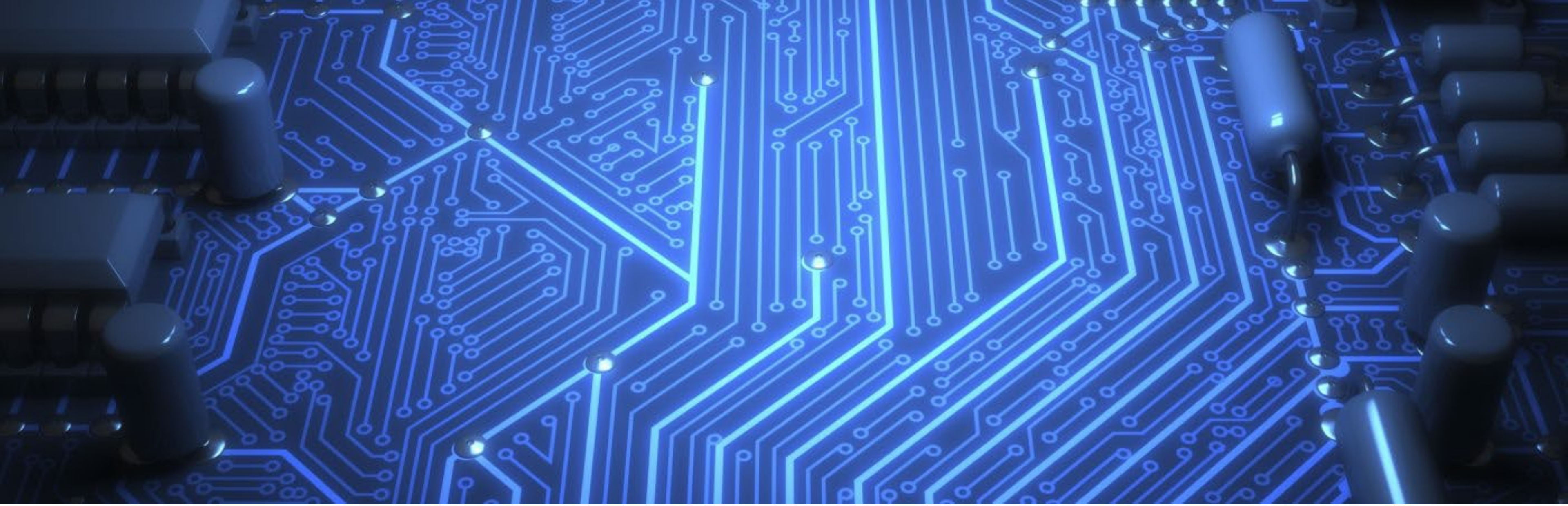


HIGH-PERFORMANCE COMPUTING + EFFICIENCY

IAP 15.S60 Session 8

George Margaritis

(Adapted from Alex Schmid)

Today’s Learning Objectives

- Submit interactive jobs, batch jobs, and job arrays on a computing cluster
- Recognize parallelizable code and implement a simple parallel job with shared memory
- Design a reproducible and efficient pipeline for scientific computing

Clusters

Why do we use computing clusters?

In optimization and stats, we often need a lot of computational power:

Usually need a lot of:

GPU

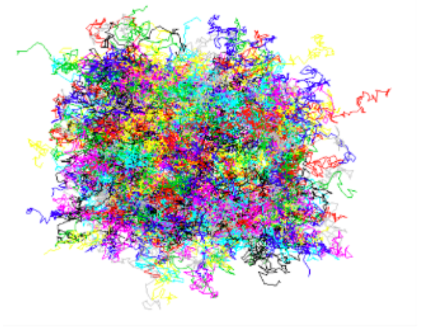
CPU

MEMORY

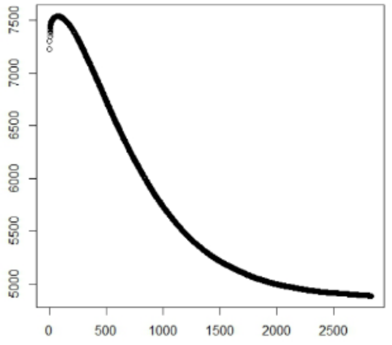
Why do we use computing clusters?

In optimization and stats, we often need a lot of computational power:

Deep learning on a large dataset



Simulations (e.g. Monte-Carlo)



Hard optimization problems

GUROBI OPTIMIZATION

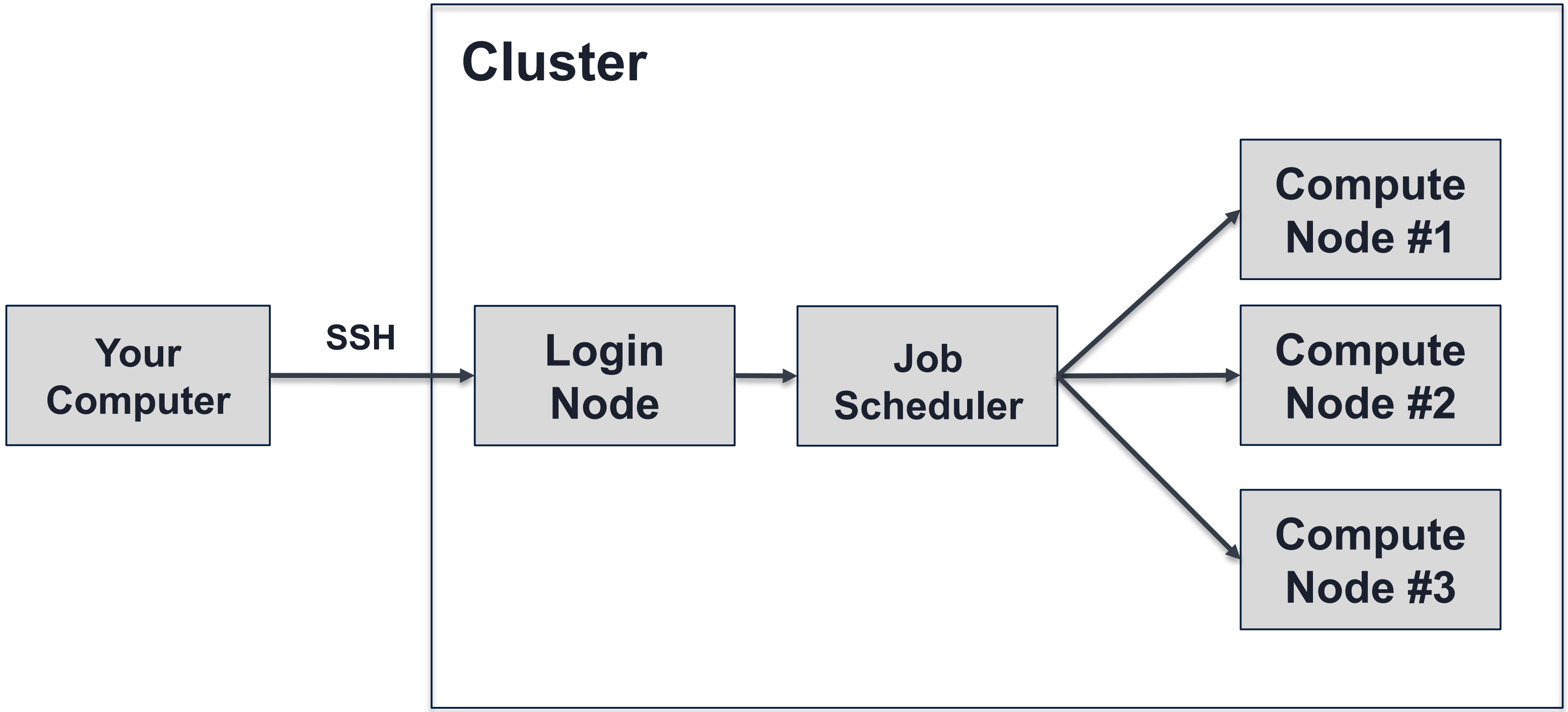
Usually need a lot of:

GPU

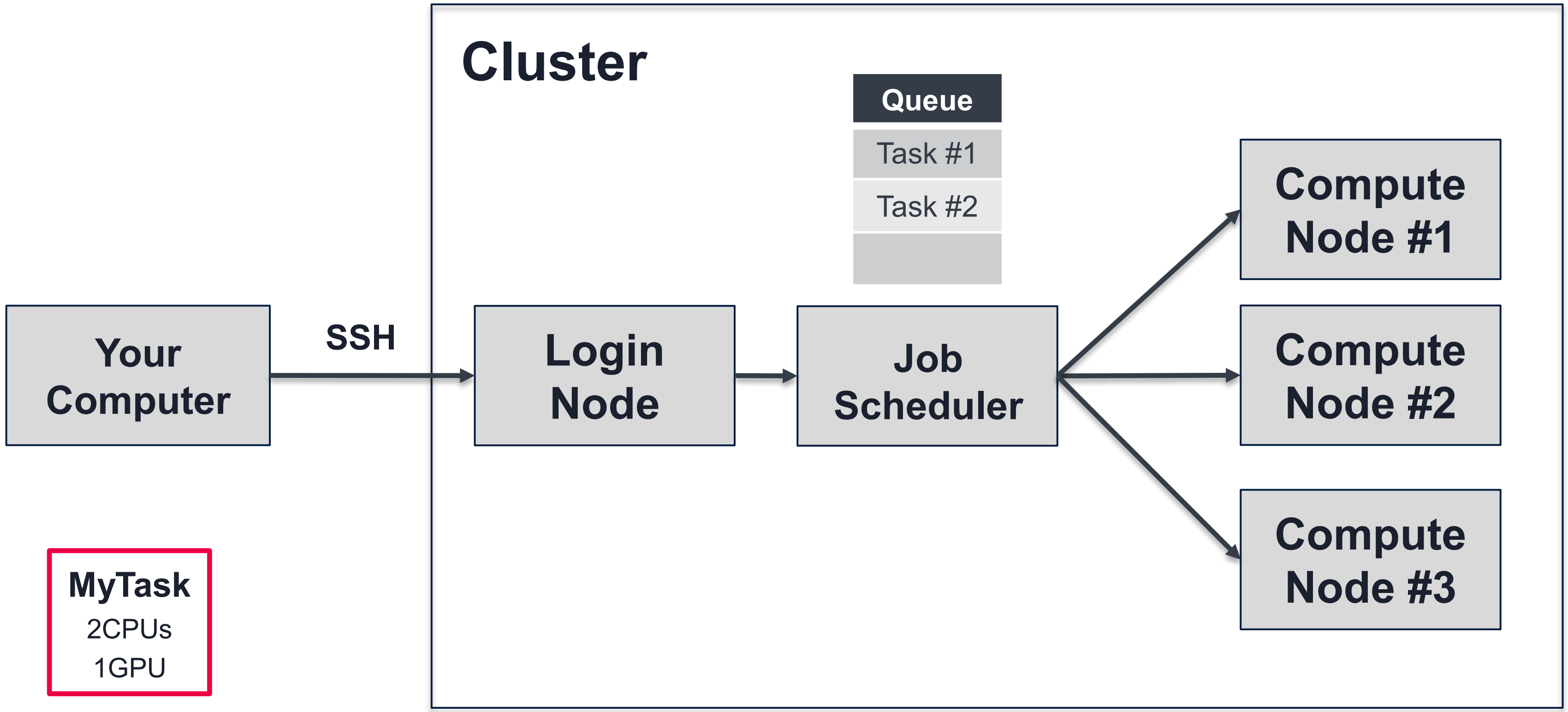
CPU

MEMORY

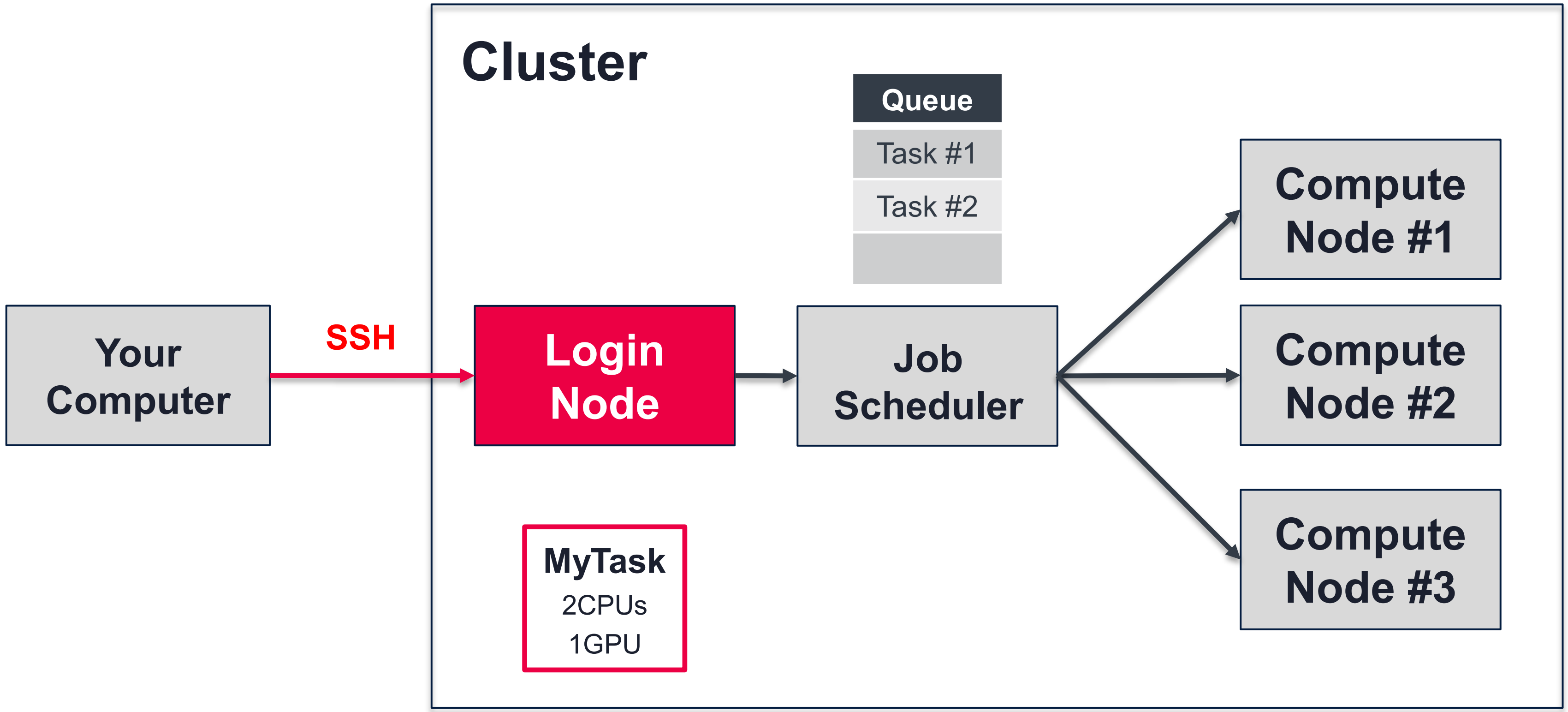
Using a cluster



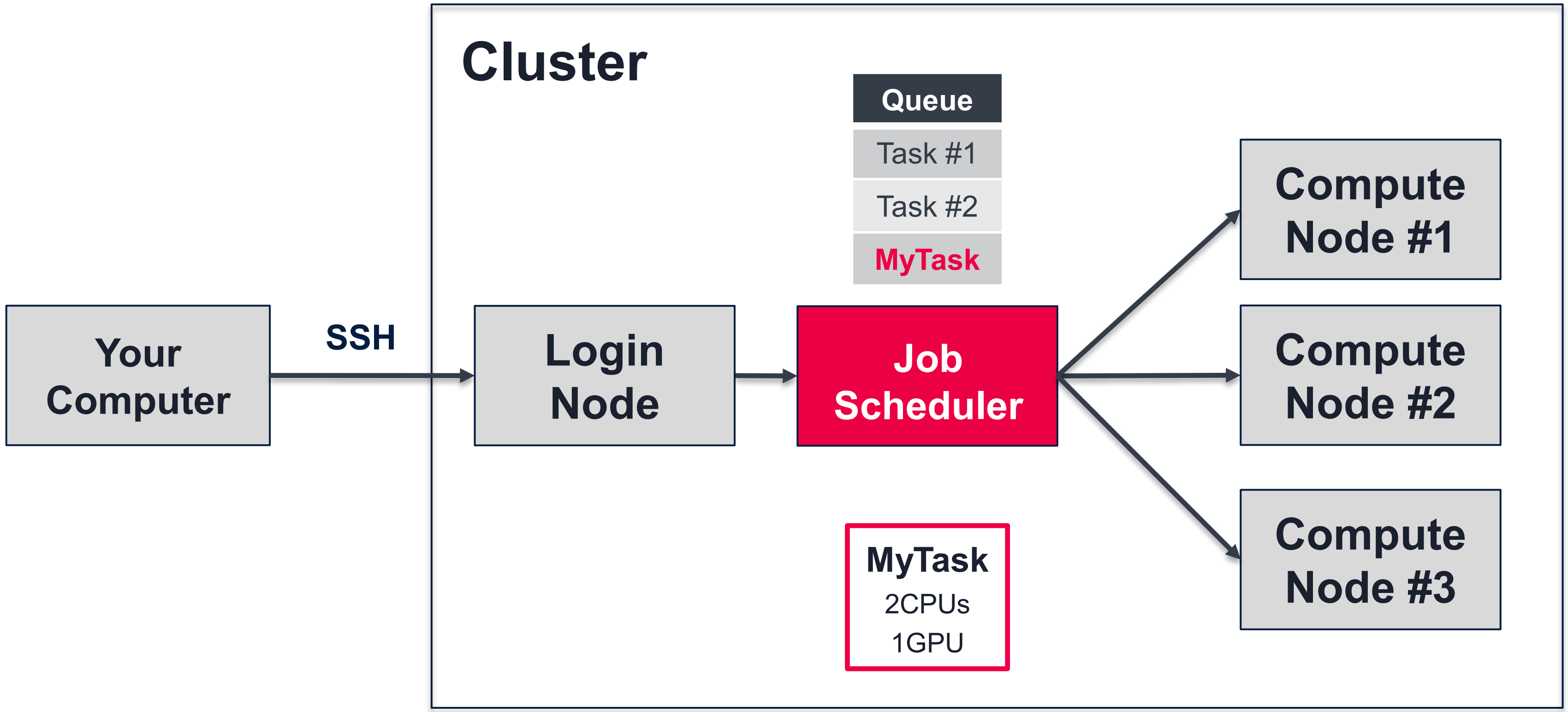
Using a cluster



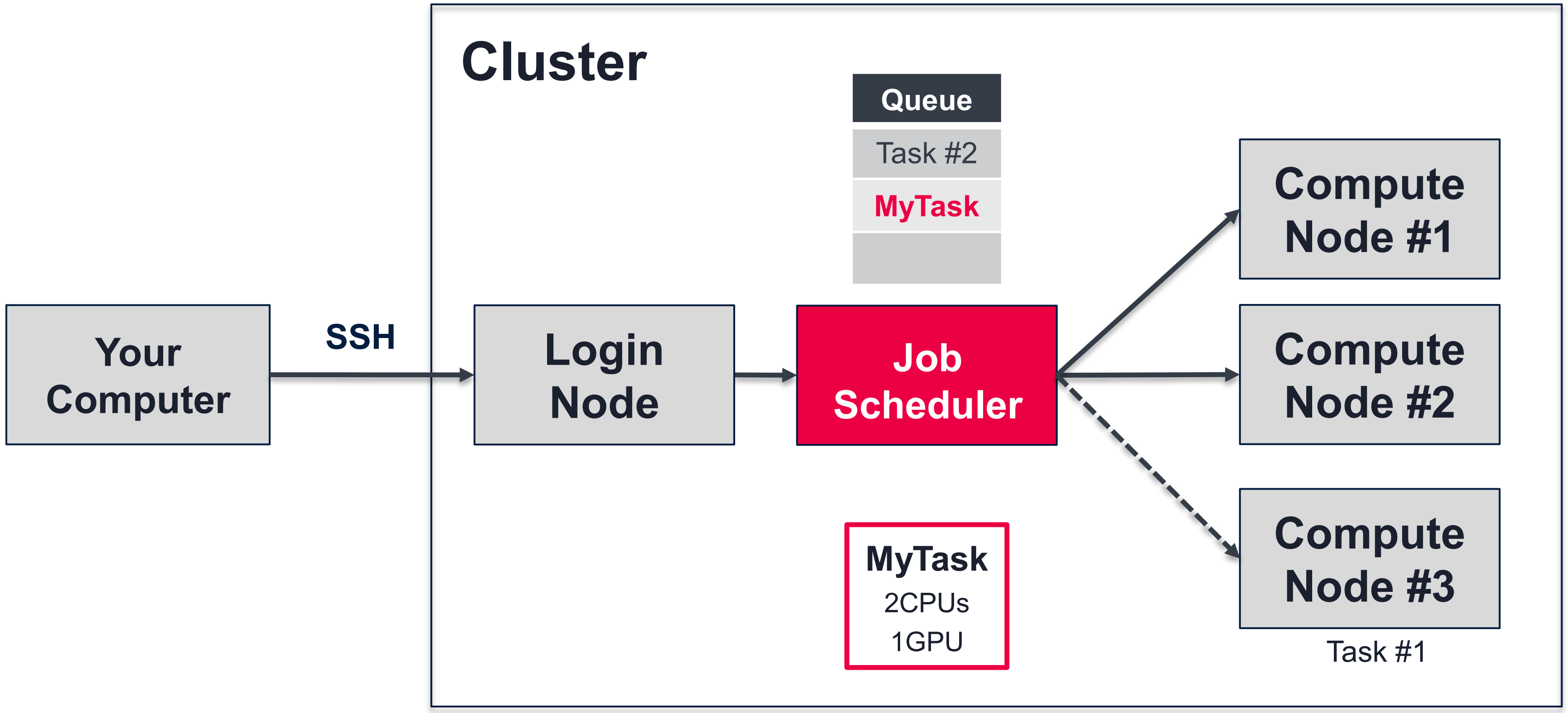
Using a cluster



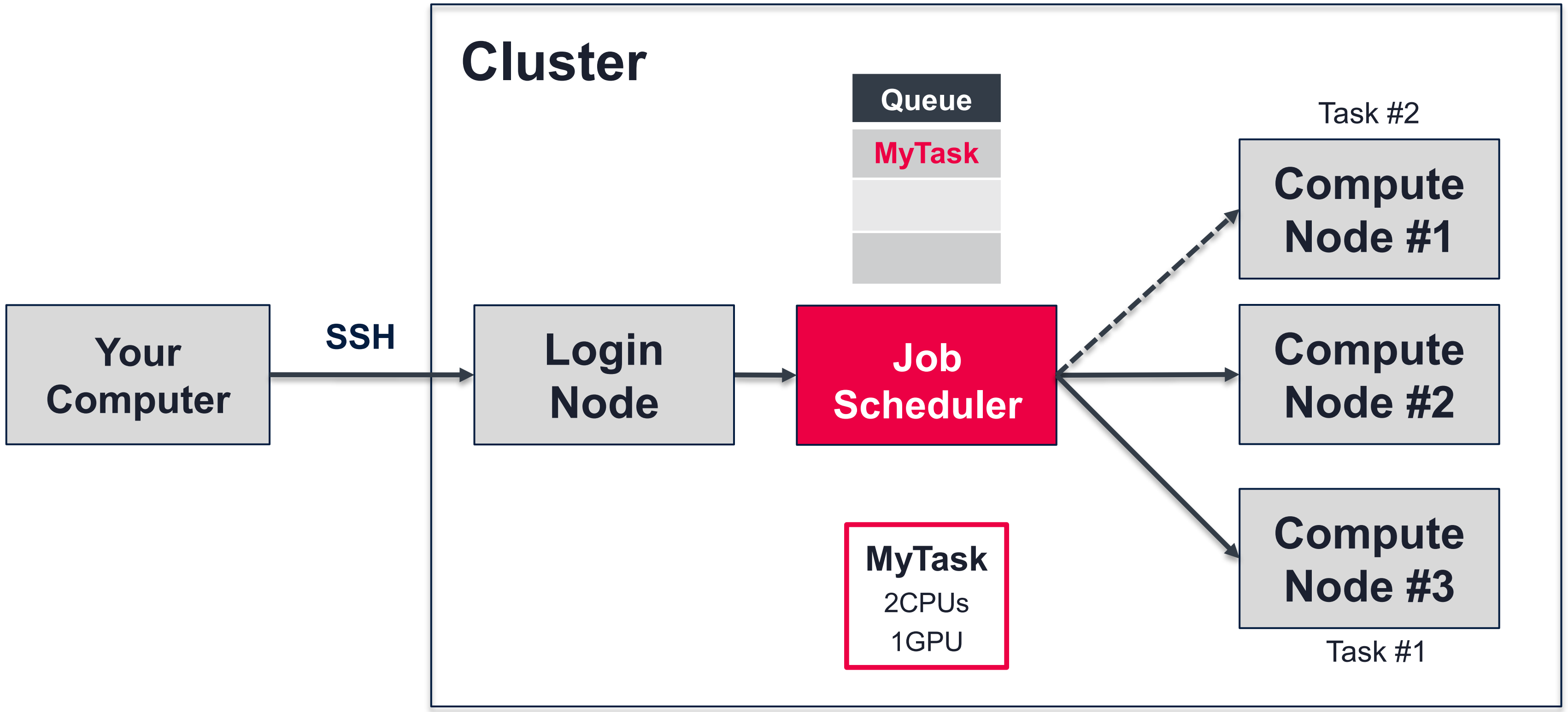
Using a cluster



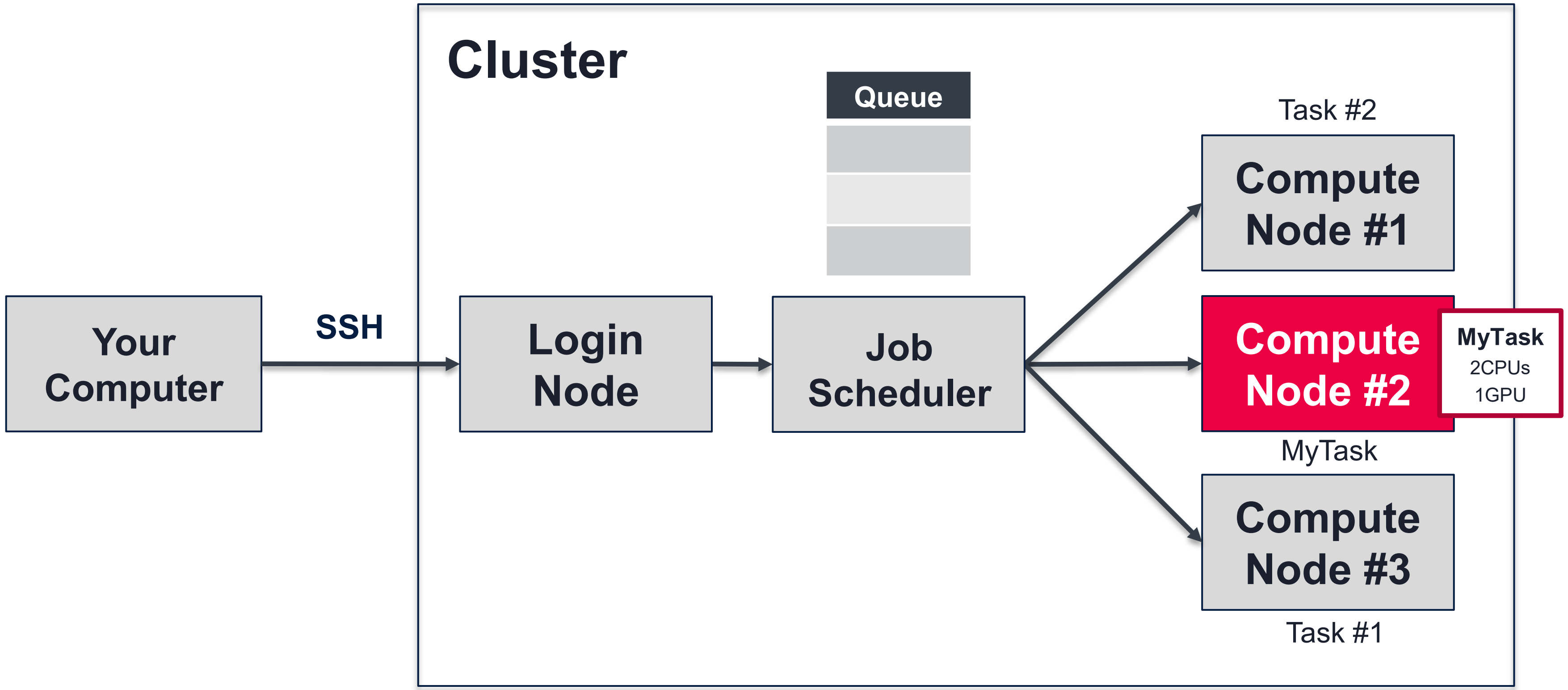
Using a cluster



Using a cluster



Using a cluster



Poll Question

pollev.com/georgemargaritis537

What cluster are you using today?

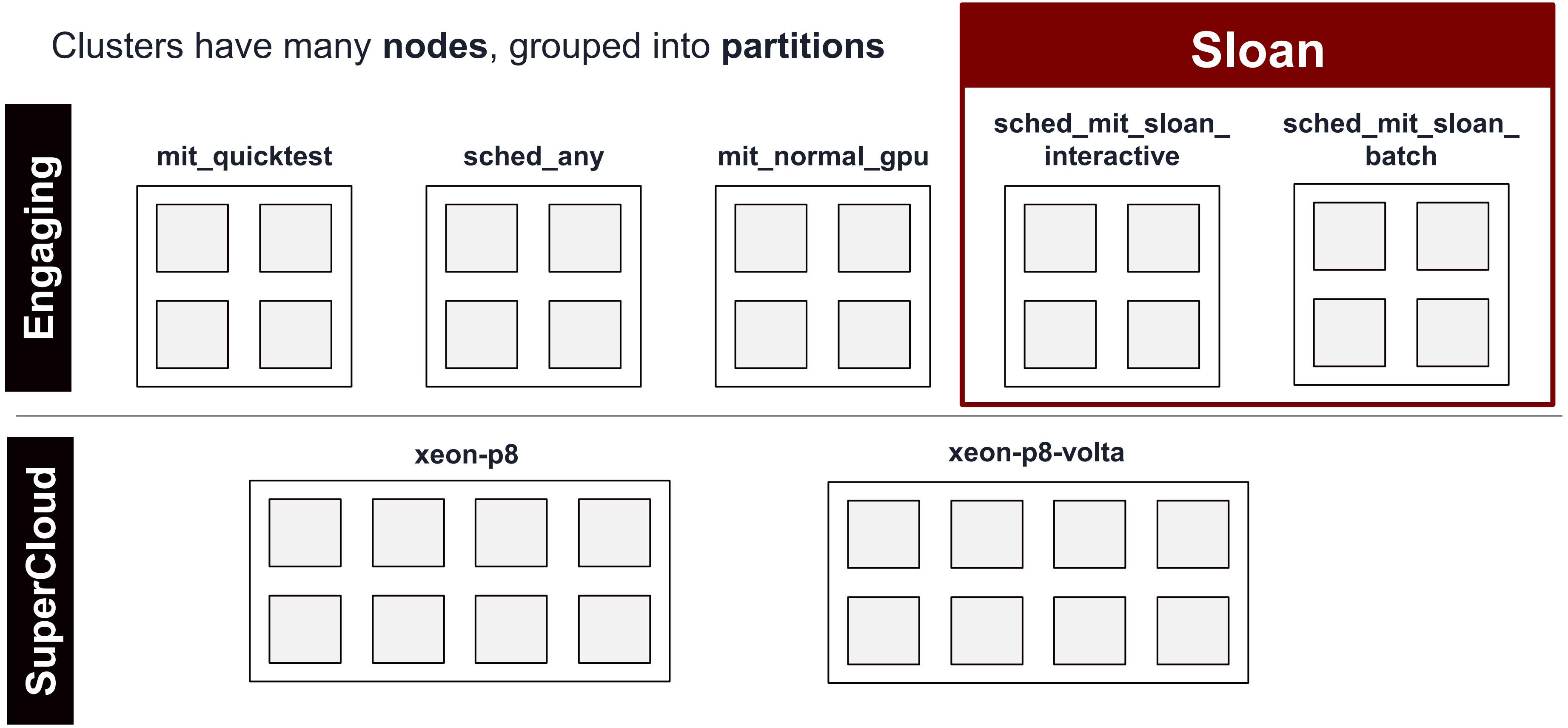
A.) Engaging OnDemand

B.) Engaging with Sloan resources

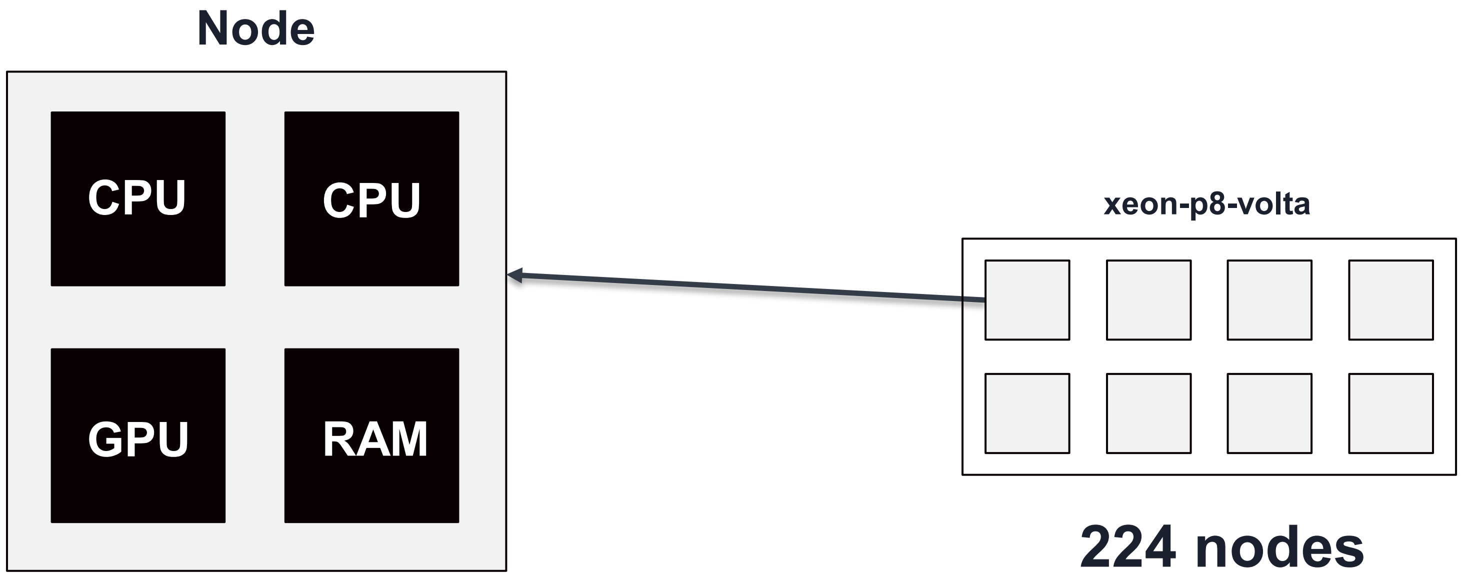
C.) SuperCloud

Partitions

Clusters have many **nodes**, grouped into **partitions**



Node example: Supercloud

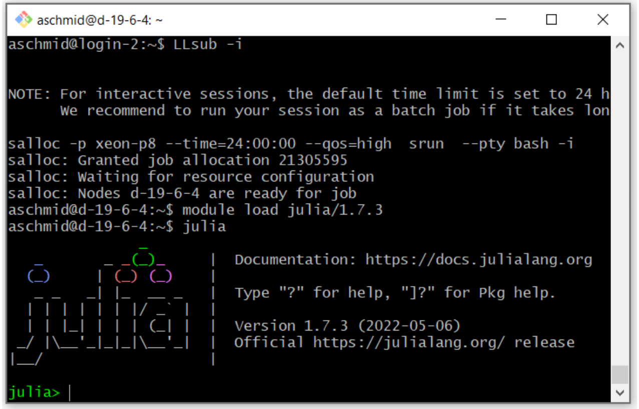


- 40 CPU cores
- 384 GB Ram
- 32 GB GPU

Types of Jobs

Interactive jobs

- Use cluster resources to interact with your code as you would locally
- Prototyping, testing, long jobs

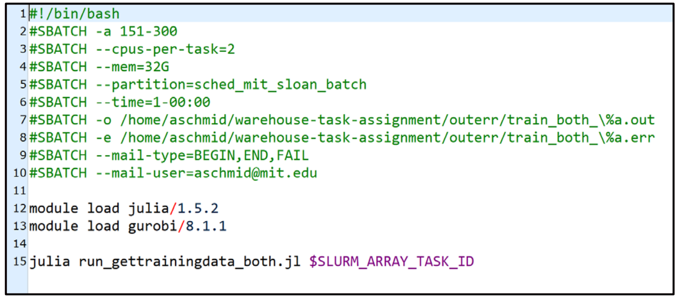


aschmid@d-19-6-4: ~

```
aschmid@login-2:~$ LLsub -i


NOTE: For interactive sessions, the default time limit is set to 24 h
      We recommend to run your session as a batch job if it takes lon

salloc -p xeon-p8 --time=24:00:00 --qos=high  srun  --pty bash -i
salloc: Granted job allocation 21305595
salloc: Waiting for resource configuration
salloc: Nodes d-19-6-4 are ready for job
aschmid@d-19-6-4:~$ module load julia/1.7.3
aschmid@d-19-6-4:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.7.3 (2022-05-06)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

Batch jobs

- Set it and forget it! Request cluster resources to run scripts, then check back later for your results
- “Official” runs, running multiple scripts



```bash
#!/bin/bash
#SBATCH -a 151-300
#SBATCH --cpus-per-task=2
#SBATCH --mem=32G
#SBATCH --partition=sched_mit_sloan_batch
#SBATCH --time=1-00:00
#SBATCH -o /home/aschmid/warehouse-task-assignment/outerr/train_both_\%a.out
#SBATCH -e /home/aschmid/warehouse-task-assignment/outerr/train_both_\%a.err
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=aschmid@mit.edu

module load julia/1.5.2
module load gurobi/8.1.1

julia run_gettrainingdata_both.jl $SLURM_ARRAY_TASK_ID
```

Log in to the cluster with SSH

Open a terminal or Git Bash window

Engaging OnDemand:

```
ssh username@eofe7.mit.edu
ssh username@eofe8.mit.edu
```

Engaging via Sloan:

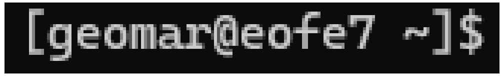
```
ssh username@eosloan.mit.edu
```

SuperCloud:

```
ssh username@txe1-login.mit.edu
```

Login Node

This will land you on the cluster login node



```
[geomar@eofe7 ~]$
```

Don’t run code on the login node! It doesn’t have many compute resources and you may cause issues for others trying to log in.

- **Exception:** on SuperCloud, only the login node has internet access, so you must use it to install new software, add packages, clone repos, etc.

Productivity Hack #1

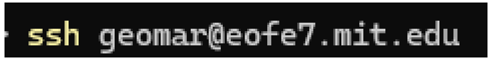
ssh config: (*~/.ssh/config* file)

Alias (we choose it)

```
Host eofe
  HostName eofe7.mit.edu
  User geomar
  IdentityFile ~/.ssh/id_rsa
```

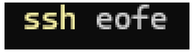
Location of my ssh private key

Instead of



```
ssh geomar@eofe7.mit.edu
```

I can do



```
ssh eofe
```

Productivity Hack #1

ssh config: (*~/.ssh/config* file)

```
Host eofe
  HostName eofe7.mit.edu
  User geomar
  IdentityFile ~/.ssh/id_rsa
```

Alias (we choose it)

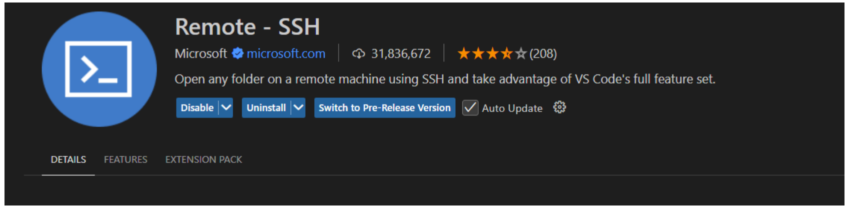
Location of my ssh private key

Instead of `ssh geomar@eofe7.mit.edu` **I can do** `ssh eofe`

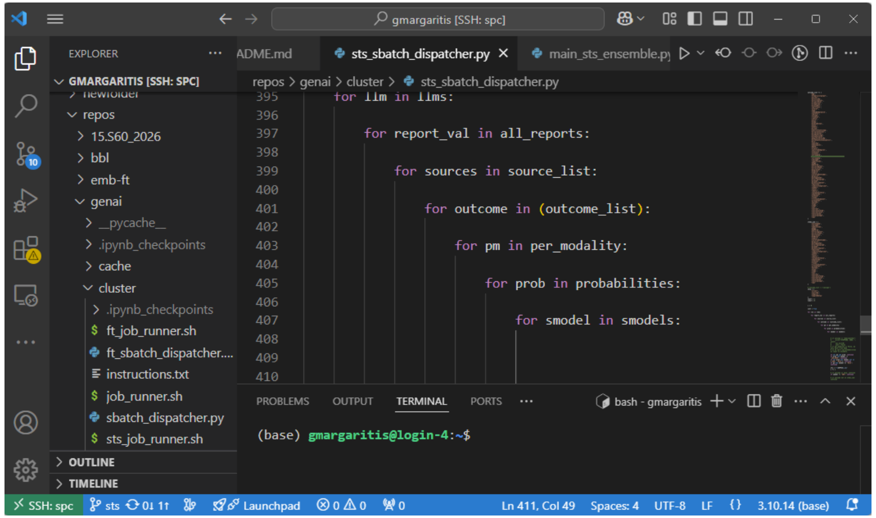
Productivity Hack #2

VSCode + Remote SSH

(**Remote SSH** Extension)



Open any folder on a remote machine using SSH and take advantage of VS Code's full feature set.



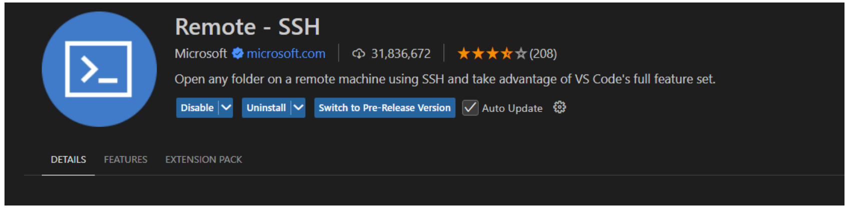
```
for llm in llms:

    for report_val in all_reports:

        for sources in source_list:

            for outcome in (outcome_list):

                for pm in per_modality:

                    for prob in probabilities:

                        for smodel in smodels:

```

```
(base) gmargaritis@login-4:~$
```

Productivity Hack #2

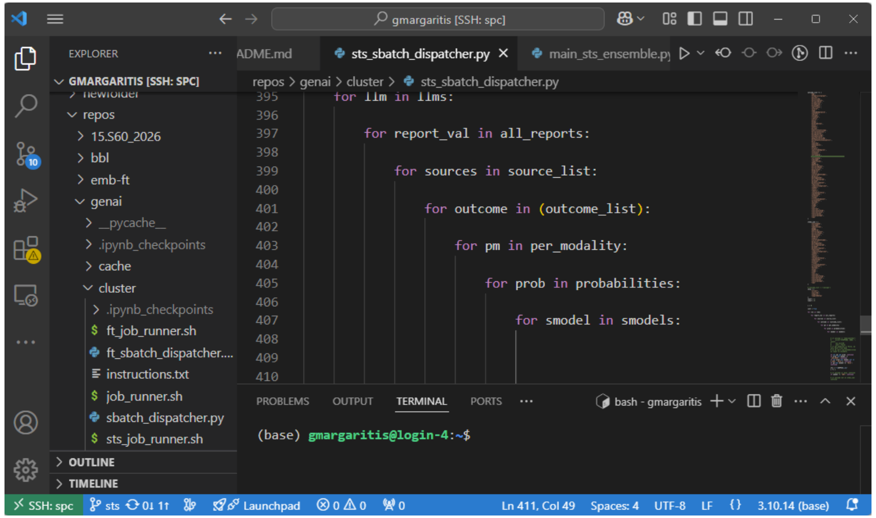
VSCode + Remote SSH

(**Remote SSH** Extension)


Remote - SSH

Microsoft microsoft.com | 31,836,672 | ★★★★☆ (208)

Open any folder on a remote machine using SSH and take advantage of VS Code's full feature set.



```
for llm in llms:

    for report_val in all_reports:

        for sources in source_list:

            for outcome in (outcome_list):

                for pm in per_modality:

                    for prob in probabilities:

                        for smodel in smodels:
```

```
(base) gmargaritis@login-4:~$
```

File system on the cluster

- Analogous to the file system on your local machine

  → We must move files and data to the cluster to run them

- **Engaging**: `/home/username`

- **SuperCloud**: `/home/gridsan/username`

Interacting with file system

Usual terminal commands

```
pwd, ls, cd
```

Move files from local to cluster

```
scp filename username@hostname:destination_file_path
```

e.g.

```
scp filename username@eofe7.mit.edu:/home/username/<destfolder>

scp filename username@eosloan.mit.edu:/home/username/<destfolder>

scp filename username@txe1-login.mit.edu:/home/gridsan/username/<destfolder>
```

Move **folder** from local to cluster

```
scp -r foldername/ username@hostname:destination_file_path
```

File Transfer – Try it out

Open a **second** terminal window to manipulate local files and run the following:

1. **Local window:** Create a new file on your local machine

```
touch newfile.txt
```

2. **Local window:** Move the file to your home directory on the cluster

```
scp newfile.txt user@eofe7.mit.edu:/home/user
scp newfile.txt user@txe1-login.mit.edu:/home/gridsan/user
```

3. **Cluster window:** Create a new folder on the cluster

```
mkdir newfolder
```

4. **Local:** Move the new folder to your machine

```
scp -r user@eofe7.mit.edu:/home/user/newfolder/  ./
scp -r user@txe1-login.mit.edu:/home/gridsan/user/newfolder/ ./
```

Productivity Hack #3

More powerful alternative to scp: **Rsync:**

- Can resume transfer if it breaks, while scp restarts transfer
- Only transfers the delta (what is left), while scp copies everything
- Rsync is faster for large files (flag –z compresses before transferring)
- Rsync also has progress bar!

Examples:

```
rsync -avz --progress filename username@eofe7.mit.edu:/home/username/<destfolder>

rsync -avz --progress mydir/ username@eofe7.mit.edu:/home/username/<destfolder>
```

Productivity Hack #3

More powerful alternative to scp: **Rsync**:

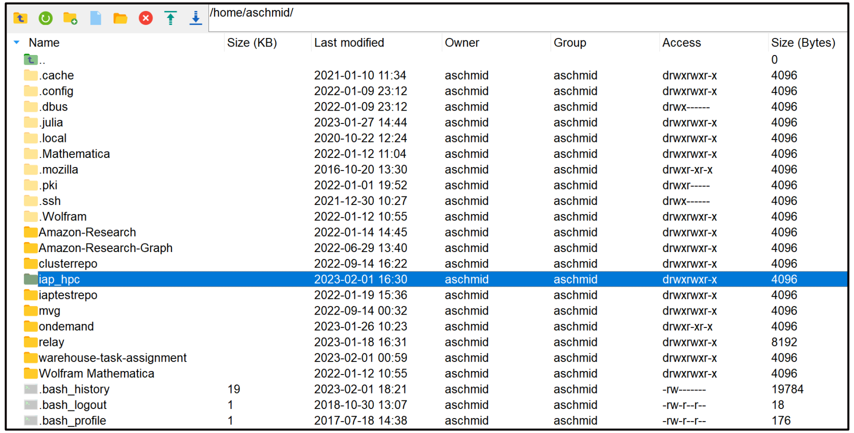
- Can resume transfer if it breaks, while scp restarts transfer
- Only transfers the delta (what is left), while scp copies everything
- Rsync is faster for large files (flag –z compresses before transferring)
- Rsync also has progress bar!

Examples:

```
rsync -avz --progress filename username@eofe7.mit.edu:/home/username/<destfolder>

rsync -avz --progress mydir/ username@eofe7.mit.edu:/home/username/<destfolder>
```

Aside: Graphical Interface for Files

- Engaging OnDemand has a GUI interface for viewing and manipulating your files
- You can view the SuperCloud file system through the web portal
- Windows users can also download and use MobaXterm, which has a nice GUI file system in addition to a shell for running slurm commands
- Mac users can use ForkLift (premium)



/home/aschmid/

Name	Size (KB)	Last modified	Owner	Group	Access	Size (Bytes)
..						0
.cache		2021-01-10 11:34	aschmid	aschmid	drwxrwxr-x	4096
.config		2022-01-09 23:12	aschmid	aschmid	drwxrwxr-x	4096
.dbus		2022-01-09 23:12	aschmid	aschmid	drwx------	4096
.julia		2023-01-27 14:44	aschmid	aschmid	drwxrwxr-x	4096
.local		2020-10-22 12:24	aschmid	aschmid	drwxrwxr-x	4096
.Mathematica		2022-01-12 11:04	aschmid	aschmid	drwxrwxr-x	4096
.mozilla		2016-10-20 13:30	aschmid	aschmid	drwxr-xr-x	4096
.pki		2022-01-01 19:52	aschmid	aschmid	drwxr-----	4096
.ssh		2021-12-30 10:27	aschmid	aschmid	drwx------	4096
.Wolfram		2022-01-12 10:55	aschmid	aschmid	drwxrwxr-x	4096
Amazon-Research		2022-01-14 14:45	aschmid	aschmid	drwxrwxr-x	4096
Amazon-Research-Graph		2022-06-29 13:40	aschmid	aschmid	drwxrwxr-x	4096
clusterrepo		2022-09-14 16:22	aschmid	aschmid	drwxrwxr-x	4096
iap_hpc		2023-02-01 16:30	aschmid	aschmid	drwxrwxr-x	4096
iaptestrepo		2022-01-19 15:36	aschmid	aschmid	drwxrwxr-x	4096
mvg		2022-09-14 00:32	aschmid	aschmid	drwxrwxr-x	4096
ondemand		2023-01-26 10:23	aschmid	aschmid	drwxr-xr-x	4096
relay		2023-01-18 16:31	aschmid	aschmid	drwxrwxr-x	8192
warehouse-task-assignment		2023-02-01 00:59	aschmid	aschmid	drwxrwxr-x	4096
Wolfram Mathematica		2022-01-12 10:55	aschmid	aschmid	drwxrwxr-x	4096
.bash_history	19	2023-02-01 18:21	aschmid	aschmid	-rw-------	19784
.bash_logout	1	2018-10-30 13:07	aschmid	aschmid	-rw-r--r--	18
.bash_profile	1	2017-07-18 14:38	aschmid	aschmid	-rw-r--r--	176

We can also use Git and Github!

In the terminal logged in to your cluster, clone today's repo into your home directory:

```
git clone https://github.com/karlkzhu/15.S60_2026.git
```

If you get an error about certificate verification, you may need to run:

```
git config --global http.sslVerify false
```

Go to the folder `8_hpc_and_efficiency` . Four folders for today's four examples:

- `1_interactive` (`_sc` for SuperCloud or `_eng` for Engaging)
- `2_batch`
- …

Recommended setup

95% of your development time

My recommendations:

- **Cluster login/File editing/Code running:** VSCode + Remote SSH
- **Small file transfer:** SCP or GUI (vscode/mobaxterm/forklift)
- **Large file/multiple files/directory transfer:** rsync (faster and more reliable)

Interactive Jobs

Starting an interactive job

Launch an interactive job with default resources

Engaging

```
srun --pty --partition=mit_quicktest bash
```

SuperCloud

```
LLsub -i
```

You can also specify your resources

[Engaging]()

```
srun --pty --partition=mit_quicktest
--cpus-per-task=1 --mem=2G bash
```

[SuperCloud]()

```
LLsub -i -s 20 -g volta:1
```

# Cpus → 20

1 GPU → volta:1

Loading Software

A variety of software is installed on the cluster, including many versions of Python, Julia, R, etc. To use them, we must load the appropriate module.

Load a known module	`module load julia/1.7.3`
See all modules	`module avail`
See specific modules, e.g. Julia	`module avail julia`

Start an Interactive Job

1. Start your interactive job

```
srun --pty --partition=mit_quicktest bash

LLsub -i
```

2. Load Julia

```
module load julia/1.7.3
```

3. Navigate to the first directory for today

`cd 1_interactive_sc` or `cd 1_interactive_eng`

4. Run our test script

```
julia testscript.jl
```

Passing Arguments

Let's run some of the scripts in `1_interactive` which find the shortest path between two nodes in the network given in the CSV file

1. First, let's check out the network details

```
julia networkdetails.jl
```

2. Let's find the shortest path between node 1 and node 20

```
julia shortestpath_noargs.jl
```

3. Now, let's pass two arguments to find the shortest path between nodes 6 and 7

```
julia shortestpath_args.jl 6 7
```

The .bashrc file

The problem:

- In order to load Julia, I need to run `module load julia/1.7.3`
- I need to do that **every time** before running my code
- Can I avoid that? Can I tell the cluster to automatically load Julia?

What is .bashrc ?

- File that is executed **every time** you log into the cluster or a node
- Can be edited using `nano ~/.bashrc`
- Append any code you want to automatically run every time **at the end of the file**

Batch Jobs

Batch Jobs

Instead of interacting directly, we’ll tell the cluster to run a set of commands on its own time!

- Run scripts with long computational time or that require lots of resources
- Running many scripts at once (e.g. testing an algorithm on 100 different datasets or instances)

Shell scripts

We tell the cluster what commands to execute with a shell script:

```
#!/bin/bash

#Set up computing environment
#SBATCH --cpus-per-task=2
#SBATCH --mem=16G
#SBATCH --partition=sched_mit_sloan_batch
#SBATCH --time=1-00:00
#SBATCH -o outputlog.out

#Load software
module load julia/1.7.3
module load gurobi/9.0.3

#Run the script as usual
julia myscript.jl
```

"shebang"

request resources *(optional)*

load software

run script, pass arguments

Kicking off a batch job

Write your batch shell script!

```
touch mybatchfile.sh
nano mybatchfile.sh
```

Kickoff job

```
sbatch mybatchfile.sh
```

Monitor your jobs

Engaging:

```
squeue -u <username>
```

SuperCloud:

```
LLstat
```

```
squeue
```

Batch Job – Try it out

1. Navigate to the folder `2_batch_sc` or `2_batch_eng`

2. Take a look at the batch file

```
cat batchjob.sh
```

3. Kick it off, check it out, then look at the results!

```
sbatch batchjob.sh
```

`squeue -u <username>` or `LLstat`

```
cat outputlog.out
```

Resources

Task	Syntax
CPUs	`--cpus-per-task=1`
Memory	`--mem=4G`
Time	`--time=1-00:00`

Engaging

Check partition resources with:

```
sinfo
```

SuperCloud

Check available resources with:

```
LLfree
```

When the cluster is busy, your job will be queued until the resources are available.
The job will fail if you use more than the requested memory.

Takeaways

- **Interactive** jobs let you code as if the cluster were your computer, while **batch jobs** run without your intervention
- Use **interactive** jobs when **prototyping** and **testing**, use **batch** jobs to kick off your **final runs**
- **Never run your scripts on the login node!** Use `sbatch` to start a batch job, or `srun / LLsub -i` to start an interactive job
- **DOCUMENTATION (IMPORTANT):**

  https://engaging-web.mit.edu/eofe-wiki/ (OLDER, SLIGHTLY DEPRECATED)

  https://orcd-docs.mit.edu/ (MORE RECENT, MORE OFFICIAL)

Break

Job Arrays

Submitting many batch jobs at once

We have a script and we want to run it for several different parameters.

e.g. testing a new optimization model or algorithm on many instances

Let’s check out an example in this folder:

`3_array_eng/`

or

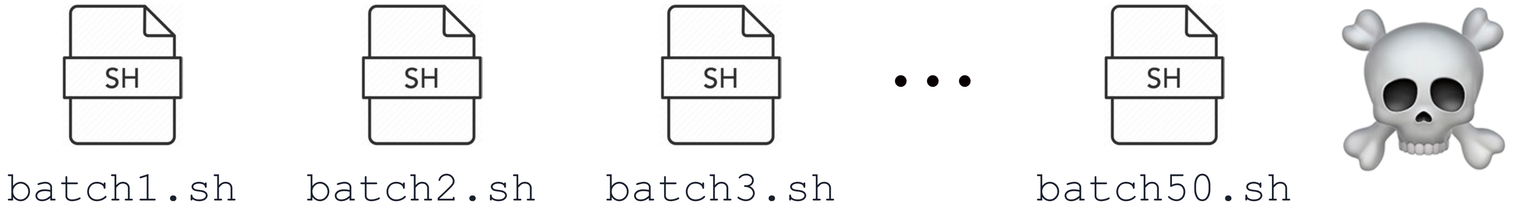
`3_array_sc/`

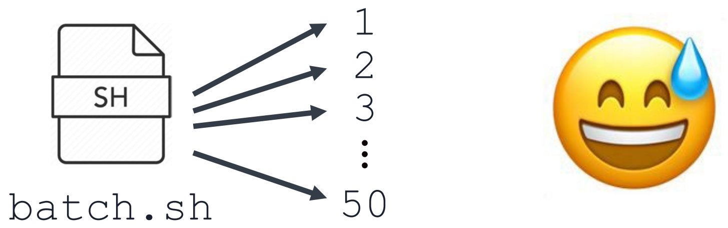
How would you approach this?

Script: `shortestpath_one.jl`

Instance list: `data/`

How can we use the cluster to accomplish this?





Job Array Batch Script - Engaging

```
#!/bin/bash

#SBATCH -a 1-50
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --partition=sched_mit_sloan_batch
#SBATCH --time=0-00:10
#SBATCH -o /home/aschmid/iap_hpc/run_\%a.out
#SBATCH -e /home/aschmid/iap_hpc/run_\%a.err

#Load software
module load julia/1.7.3

#Run the script as usual
julia shortestpath_one.jl $SLURM_ARRAY_TASK_ID
```

Job Array Batch Script - SuperCloud

```
#!/bin/bash

#SBATCH -o array.sh.log-%a
#SBATCH -a 1-3

#Load software
module load julia/1.7.3

julia shortestpath_many.jl  $SLURM_ARRAY_TASK_ID
$SLURM_ARRAY_TASK_COUNT
```

Modifying our Julia script

We need to tell Julia that we'll be passing arguments and specify how she should handle them. For example,

```
runid = parse(Int, ARGS[1])

networkfile = string("data/network", runid, ".csv")
outputfile = string("outputs/network", runid, ".csv")
```

Kickoff Job Array – Try it out

1. Kick off the batch job

```
sbatch array.sh
```

2. Once it completes, check out the `outputs` folder

```
ls outputs/
```

3. Run the script `combineoutputfiles.jl` to gather the outputs from the experiments into one file (use an interactive job to avoid login node!)

```
srun... or LLsub -i
module load julia/1.7.3
julia combineoutputfiles.jl
```

4. Check out the combined file

```
cat outputs/combined.csv
```

Poll Question

pollev.com/georgemargaritis537

What if instead of having each of the 50 runs write to its own output file, we had them all write to one file?

A.) All 50 would write to the output file, but the rows may be in a random order

B.) Some of the 50 may overwrite each other, leaving an incomplete output file

C.) An error, as all jobs are trying to access the same file at the same time

Best Practices

- Before kicking off *n* jobs (where *n* is large), test code locally, then perhaps interactively, then kick-off 1-3 batch runs to avoid having all *n* fail

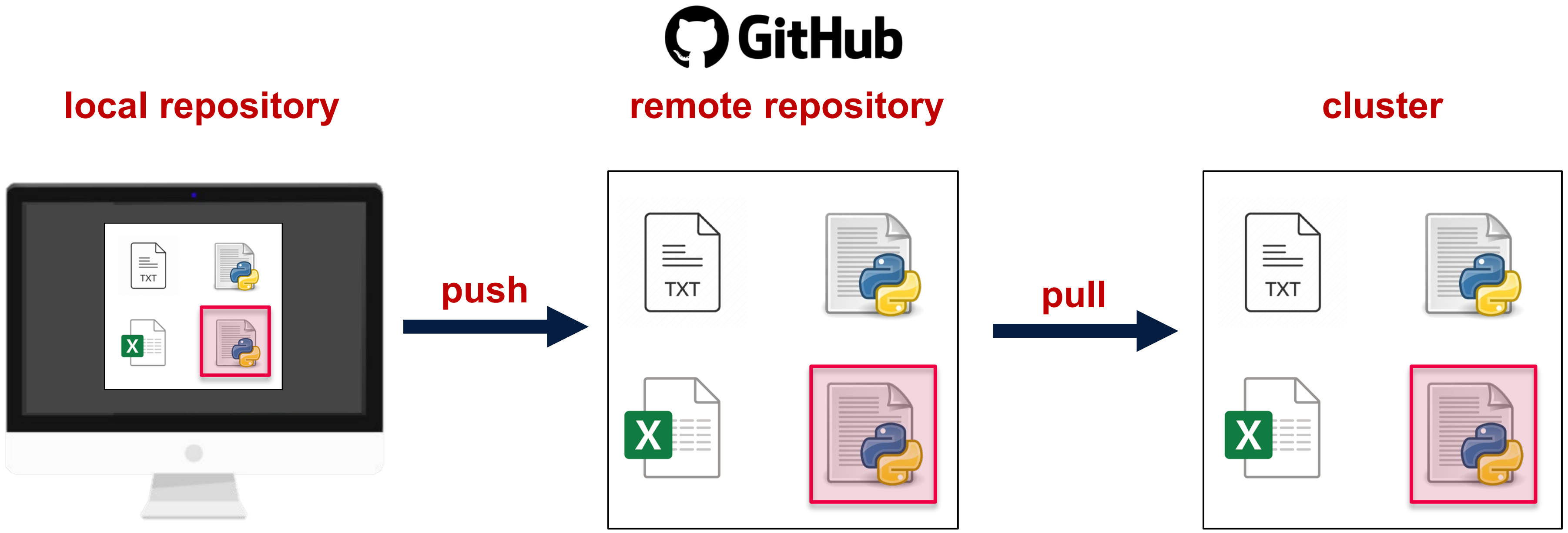
- Be mindful about the resources you request! The scheduler may give you lower priority if you run jobs requesting a lot of resources, so make sure you request what you need

- To automate the last step (gathering files), check out LLMapReduce if you're using SuperCloud

Collaboration Tips

Collaborating with the Cluster

We know we can use Git/Github on the cluster. We can then:



Edit code here

Run code here

Connecting Engaging to Github

Follow the link below for instructions on how to generate an SSH key for your Github account, add your key to the SSH agent, then add the key to your Github account https://docs.github.com/en/get-started/quickstart/set-up-git

Aside: Data Storage on the Cluster

- If your data is small, you can just store it on Github
- Otherwise, SuperCloud has a nice collaboration features that lets you create a shared group directory, which will be stored in `/home/gridsan/groups`
    - To request a group directory, send email to supercloud@mit.edu (details)

Storage space:

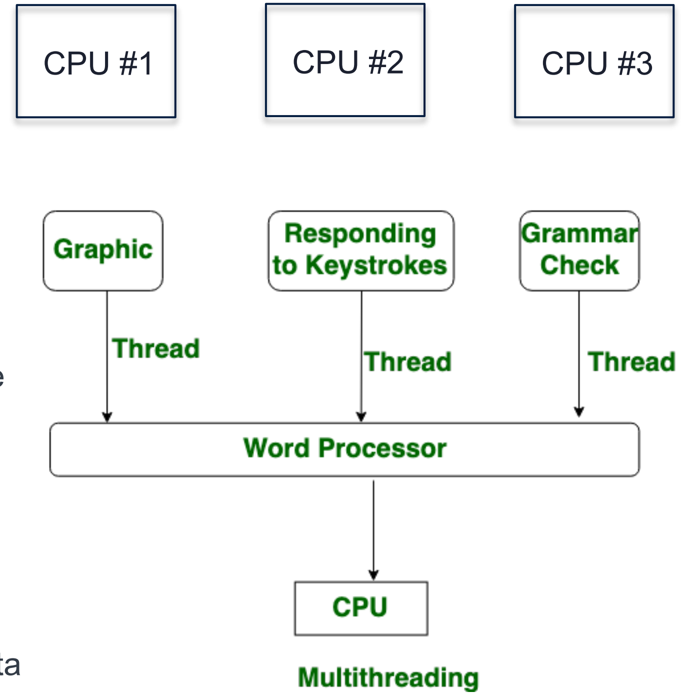
- **Engaging**: `/home/username` (100GB per user)

  `/pool001/username` (1TB per user)
- **SuperCloud**: `/home/gridsan/username` (no limit)

Parallel Computing: Multithreading

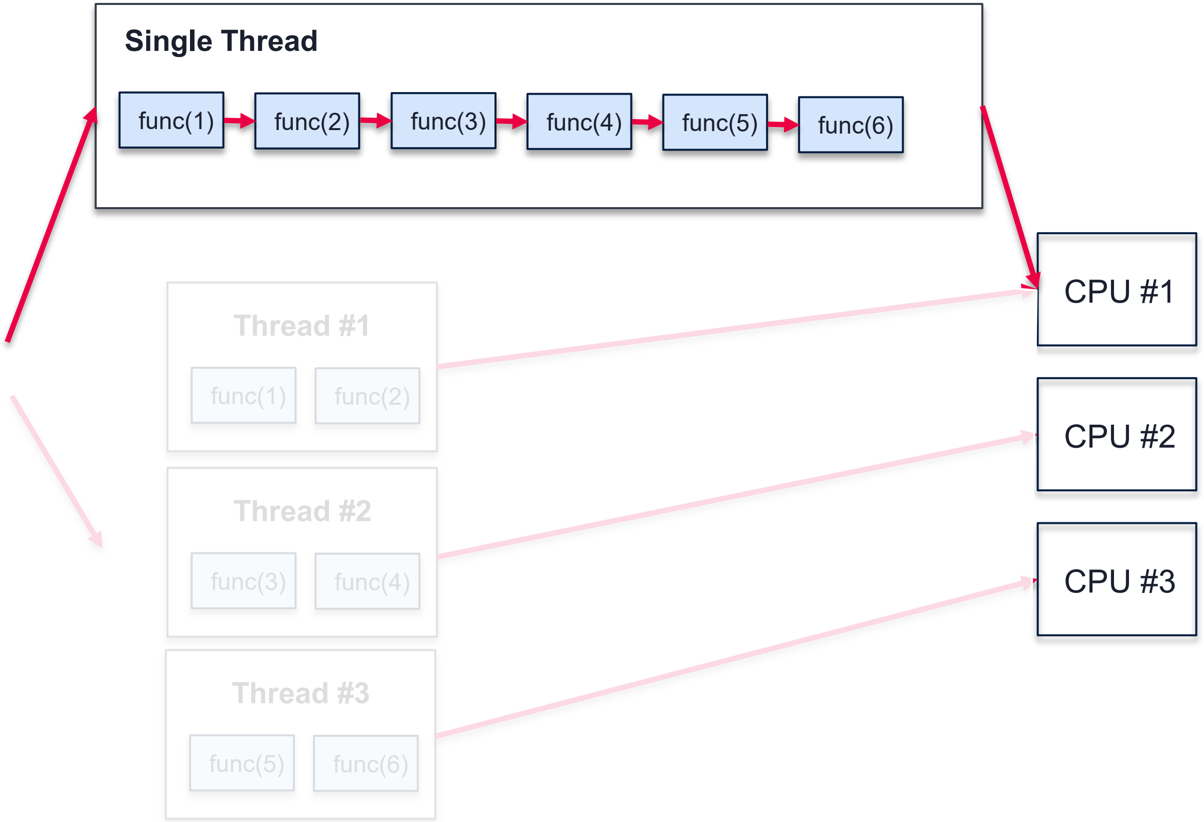
Multithreading

- Multithreading allows a computer program to perform multiple tasks concurrently.
- **Thread:** A lightweight unit of a process that can execute independently. Think of it as a small unit of work within a program.
- If we have multiple CPUs, multiple threads can be processed at the same time by our CPUs:
  - **Program can run much faster!**
- Biggest challenge:
  - **Race conditions:** Occur when multiple threads access shared data at the same time. May lead to unpredictable outcomes



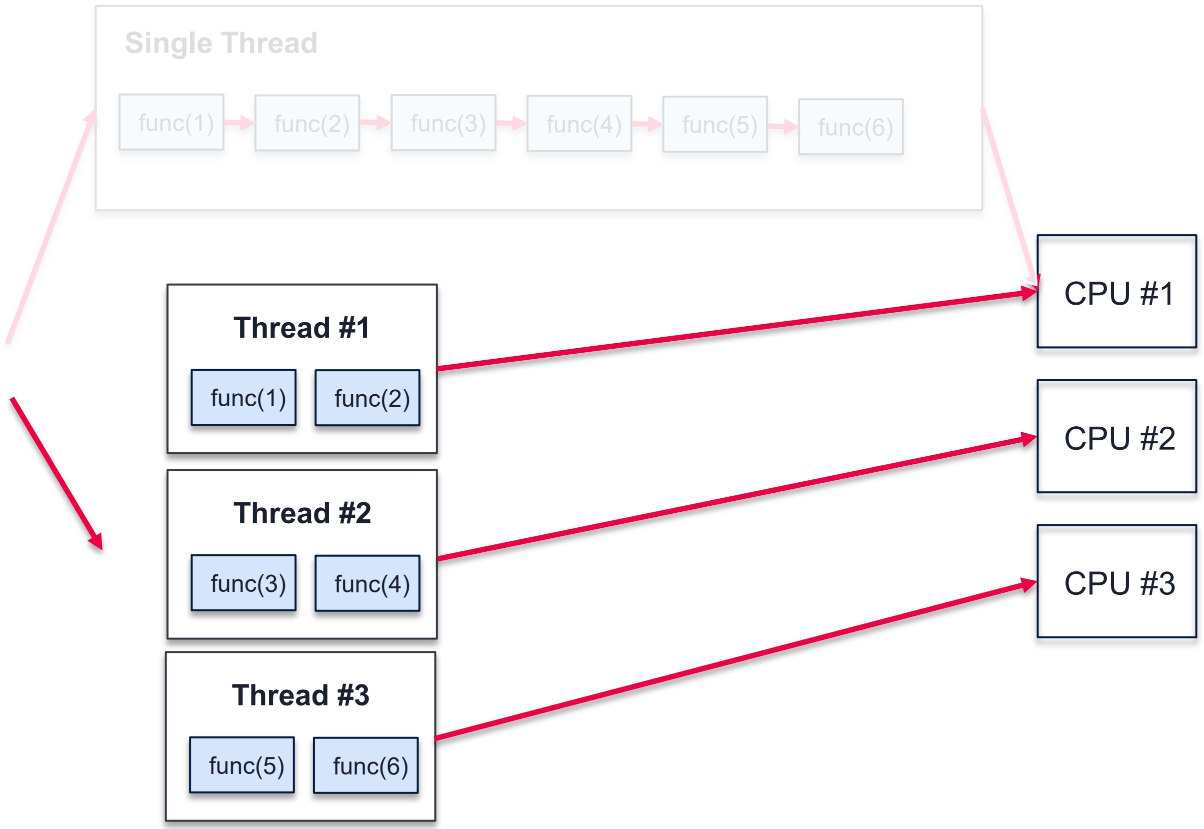
Single-threading vs Multithreading

```
r = 0
for i=1…6
     r += func(i)
end
```



Single-threading vs Multithreading

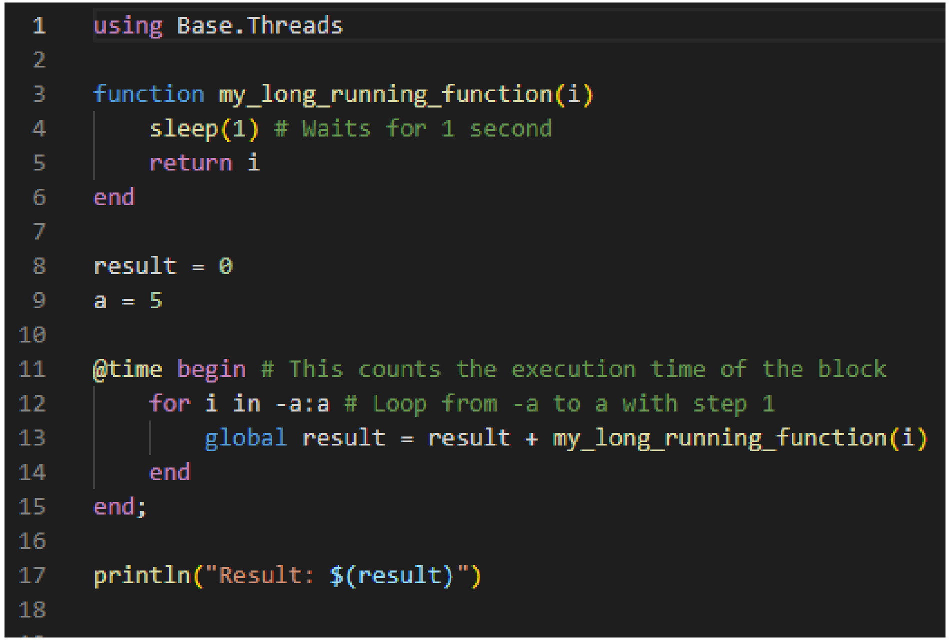
```
r = 0
for i=1…6
    r += func(i)
end
```



pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_1.jl`



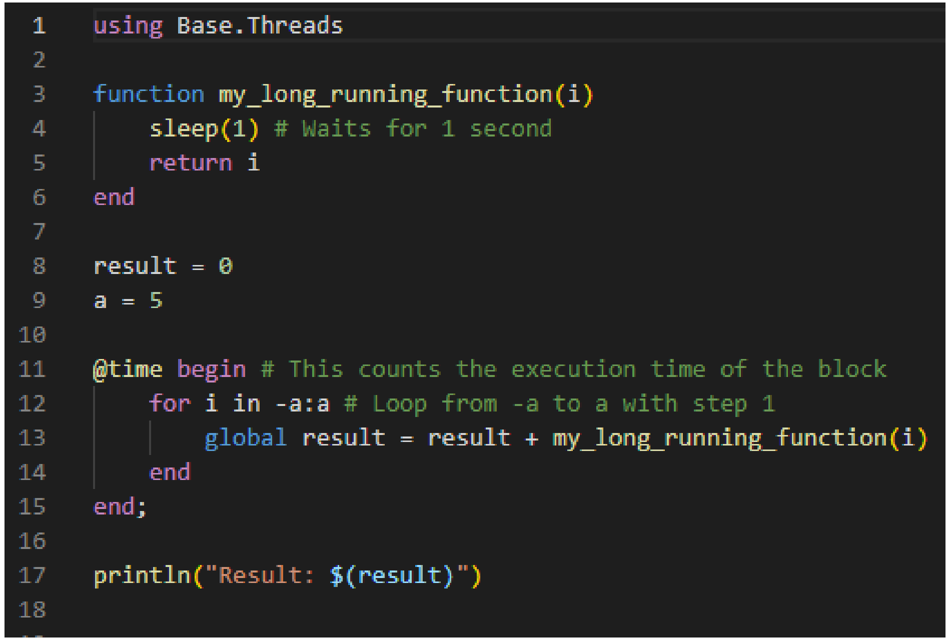
```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")

```

- Assume we only have 1 CPU:
  - **Supercloud:** `LLsub -i -s 1`
  - **Engage:** `srun --pty --cpus-per-task=1 bash`
- Answer the following:
  - Loop execution time?
  - Result?

pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_1.jl`



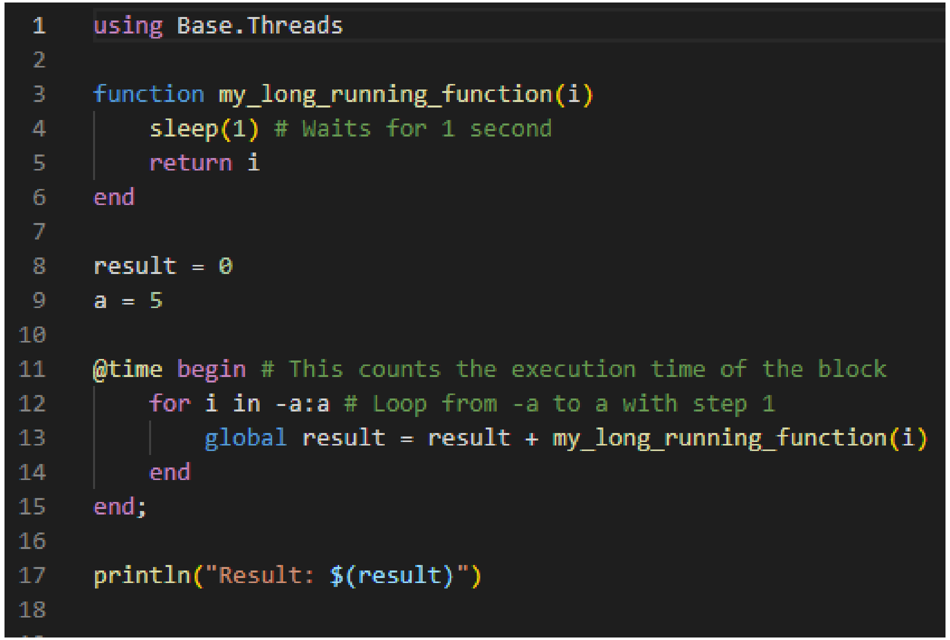
```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")

```

- Assume we only have 1 CPU:
  - **Supercloud:** `LLsub -i -s 1`
  - **Engage:** `srun --pty --cpus-per-task=1 bash`
- Answer the following:
  - Loop execution time? **11 seconds**
  - Result? **0**

pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_1.jl`



```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")
```

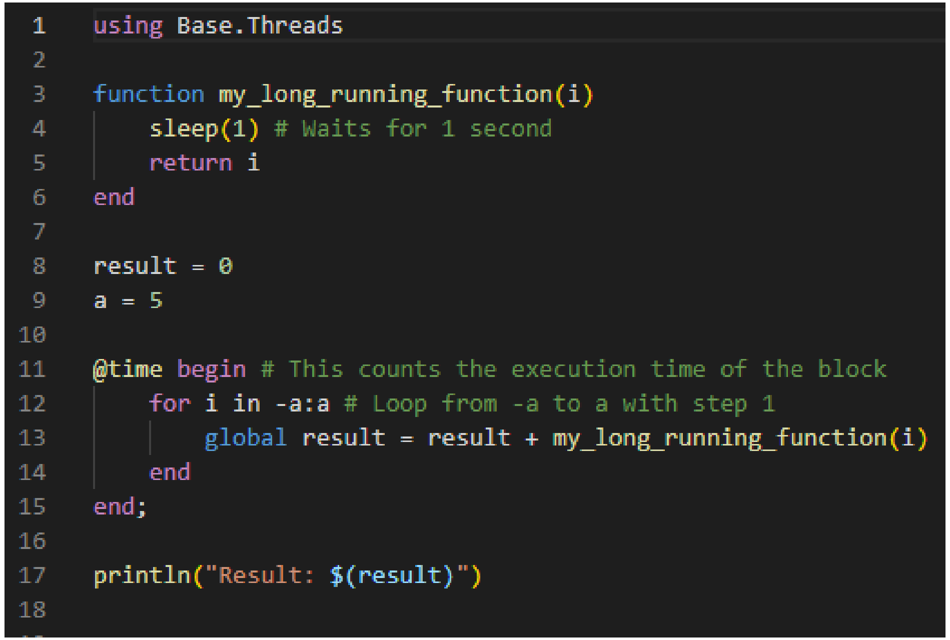
Make sure to run `julia -t <num_threads> loop_1.jl` to run Julia with multiple threads, where `<num_threads>=<num_cpus>`

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time?
  - Result?

pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_1.jl`



```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")

```

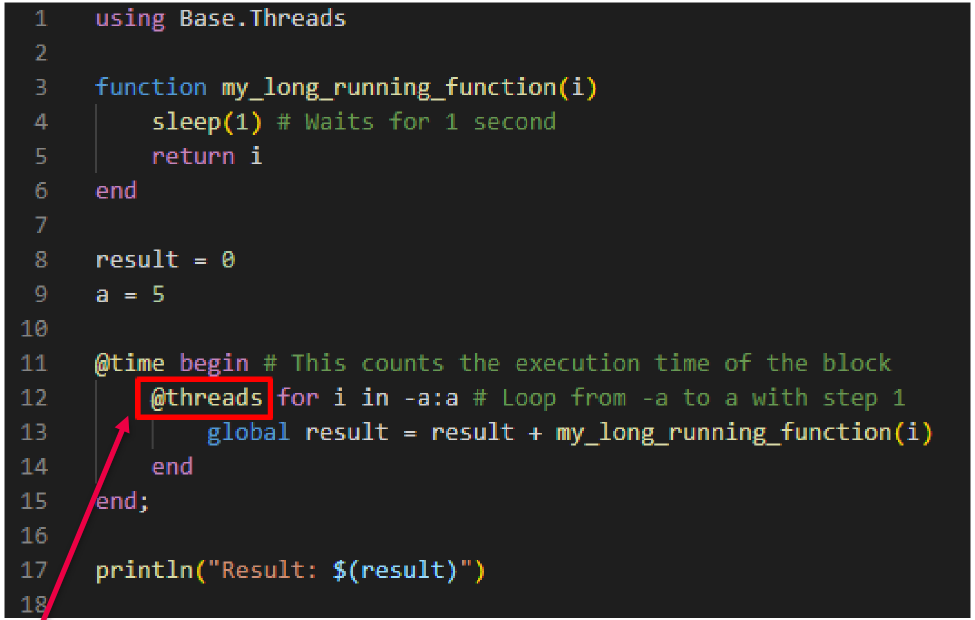
Make sure to run `julia -t <num_threads> loop_1.jl` to run Julia with multiple threads, where `<num_threads>=<num_cpus>`

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time? **11 seconds**
  - Result? **0**

pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_2.jl`



```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    @threads for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")

```

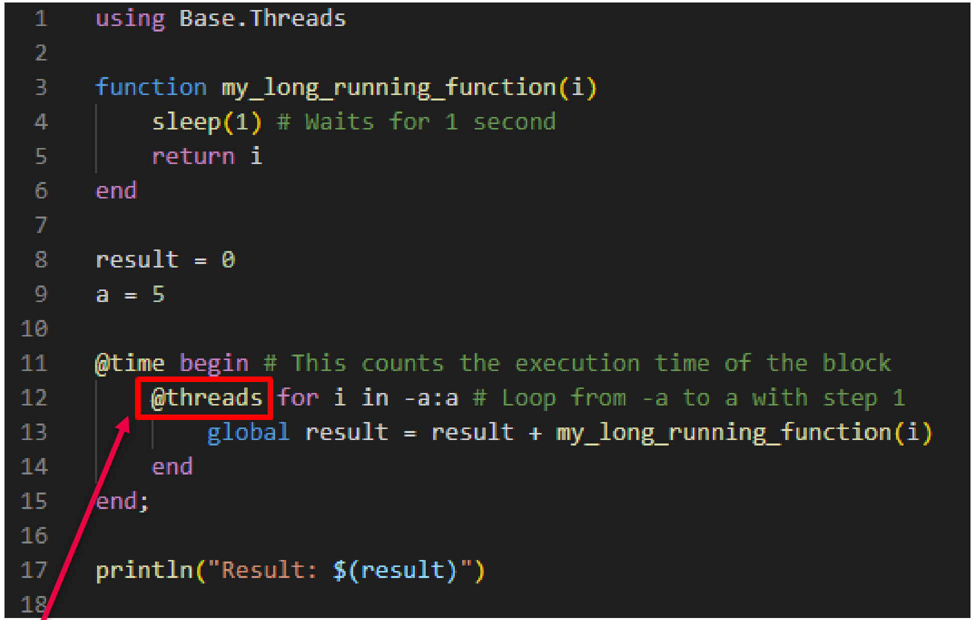
This macro **parallelizes** the loop and runs it across multiple threads. Make sure to run `julia -t <num_threads> loop_2.jl` to run Julia with multiple threads, where `<num_threads>=<num_cpus>`

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time?
  - Result?

pollev.com/georgemargaritis537

Parallelizing Julia for loops

File: `loop_2.jl`



```julia
using Base.Threads

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

result = 0
a = 5

@time begin # This counts the execution time of the block
    @threads for i in -a:a # Loop from -a to a with step 1
        global result = result + my_long_running_function(i)
    end
end;

println("Result: $(result)")

```

This macro **parallelizes** the loop and runs it across multiple threads. Make sure to run `julia -t <num_threads> loop_2.jl` to run Julia with multiple threads, where `<num_threads>=<num_cpus>`

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time? **1 seconds**
  - Result? **Unknown!!!!**

Threads read/write to the same variable *`result`* at the same time: "**Race condition**"

Race Condition

Shopping List on Google Sheets

Read 1

Item	Quantity
Apples	1
...	...

Read 1

I need **2** more Apples for a smoothie:

Total apples = 1+2=**3**

I need **3** more Apples for Apple pie:

Total apples = 1+3=**4**

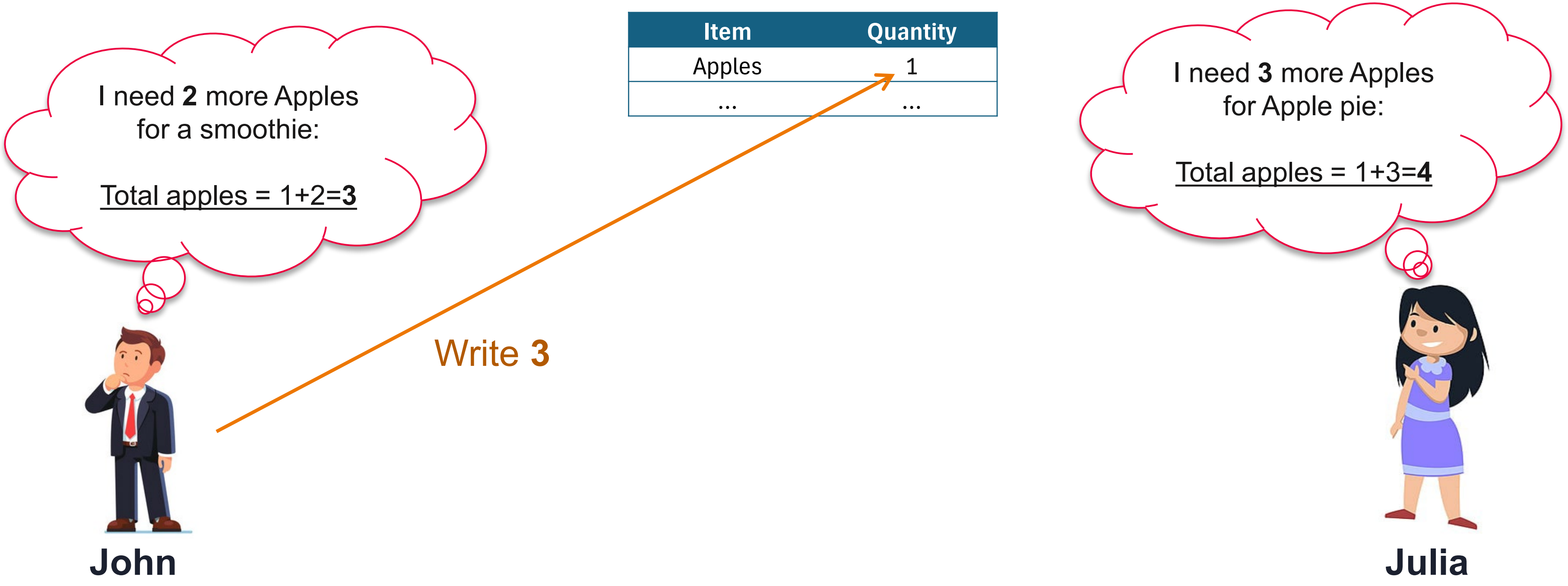
John

Julia

Race Condition

Shopping List on Google Sheets

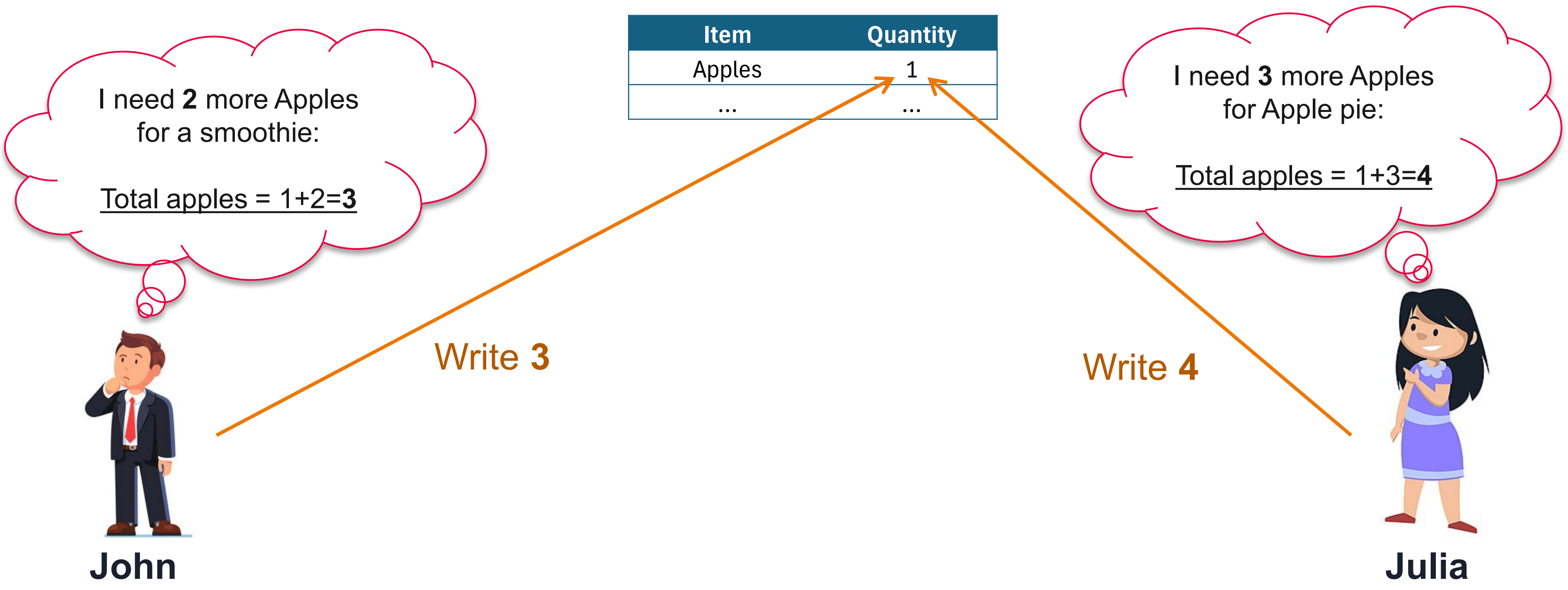
Item	Quantity
Apples	1
...	...


I need **2** more Apples for a smoothie:
Total apples = 1+2=**3**
I need **3** more Apples for Apple pie:
Total apples = 1+3=**4**
Write **3**
John
Julia

Race Condition

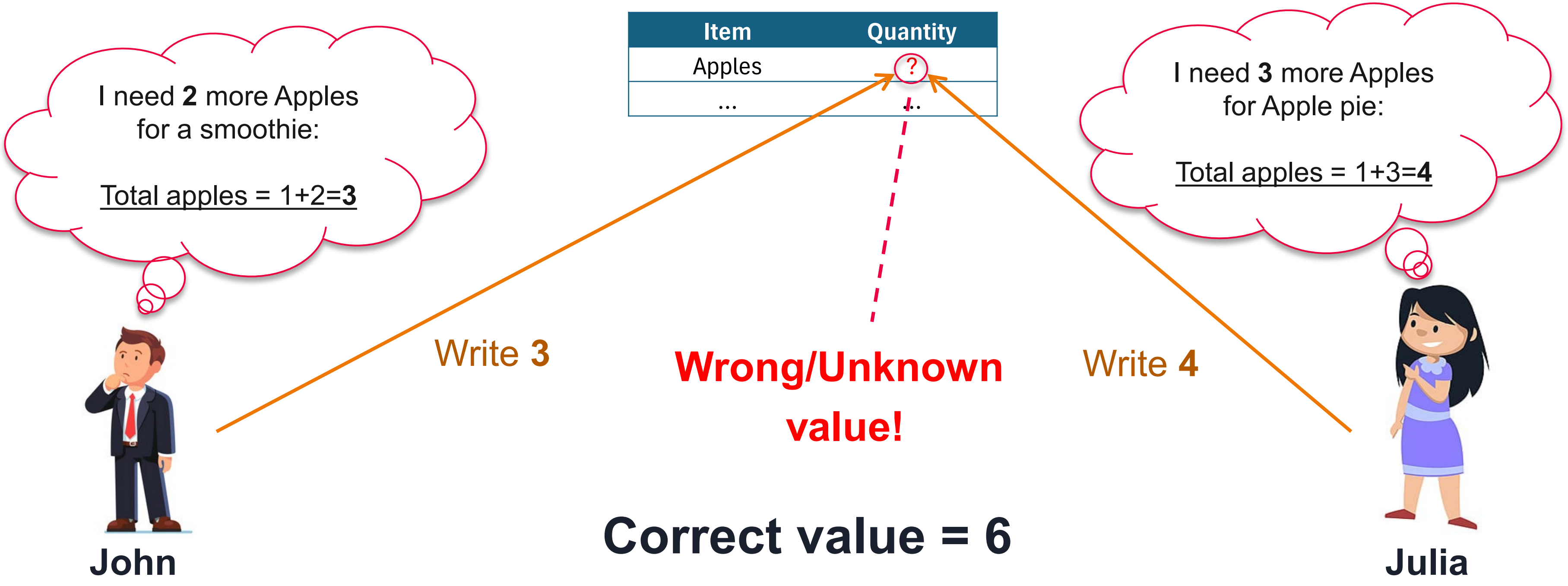
Shopping List on Google Sheets

Item	Quantity
Apples	1
...	...


I need **2** more Apples for a smoothie:
Total apples = 1+2=**3**
Write **3**
Write **4**
I need **3** more Apples for Apple pie:
Total apples = 1+3=**4**
John
Julia

Race Condition

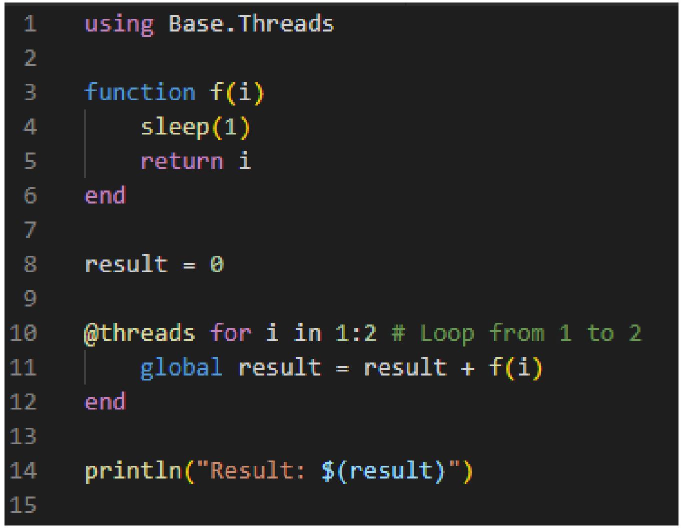
Shopping List on Google Sheets



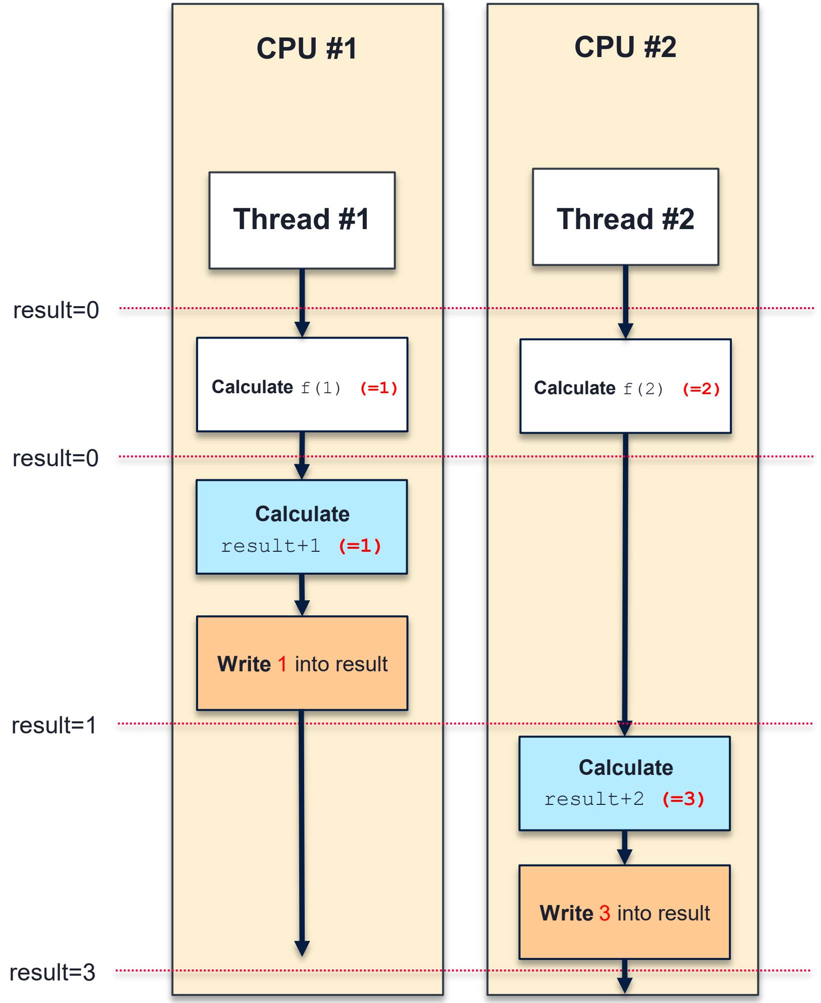
Item	Quantity
Apples	?
...	...

Correct value = 6

Race Condition

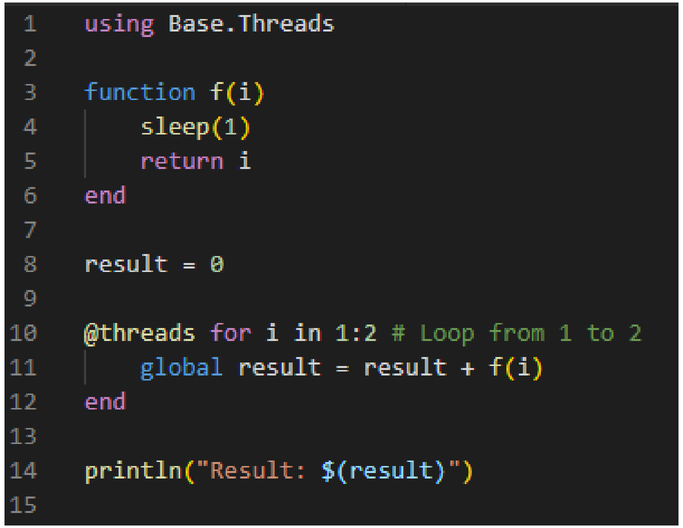


```julia
using Base.Threads

function f(i)
    sleep(1)
    return i
end

result = 0

@threads for i in 1:2 # Loop from 1 to 2
    global result = result + f(i)
end

println("Result: $(result)")
```



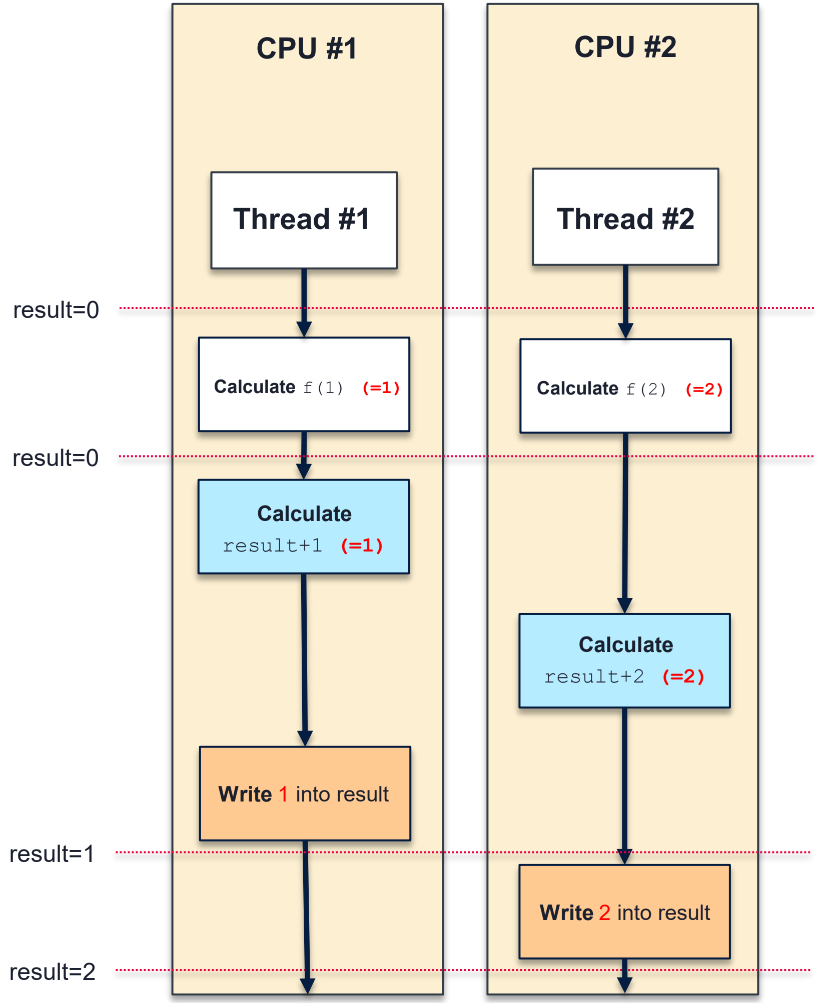
Correct value!

Race Condition



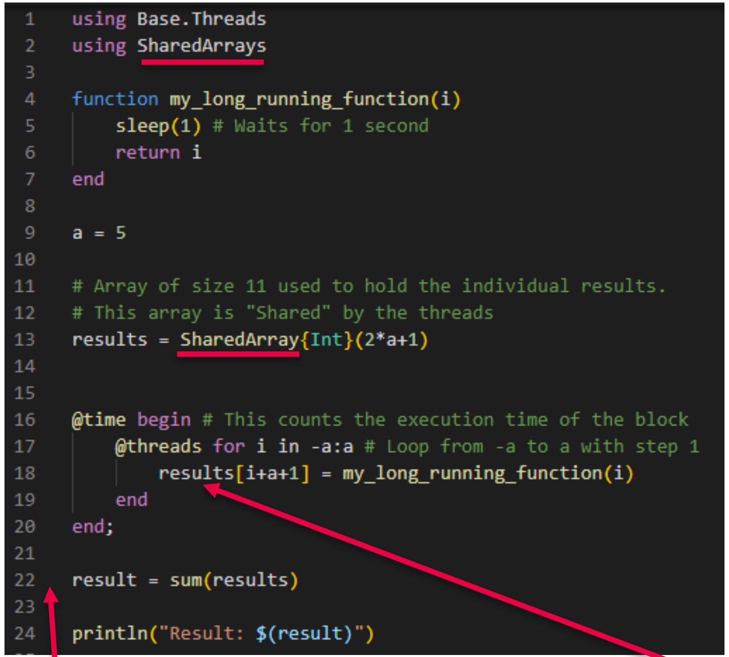
```julia
using Base.Threads

function f(i)
    sleep(1)
    return i
end

result = 0

@threads for i in 1:2 # Loop from 1 to 2
    global result = result + f(i)
end

println("Result: $(result)")

```

Incorrect value!



Parallelizing Julia for loops

File: `loop_3.jl`



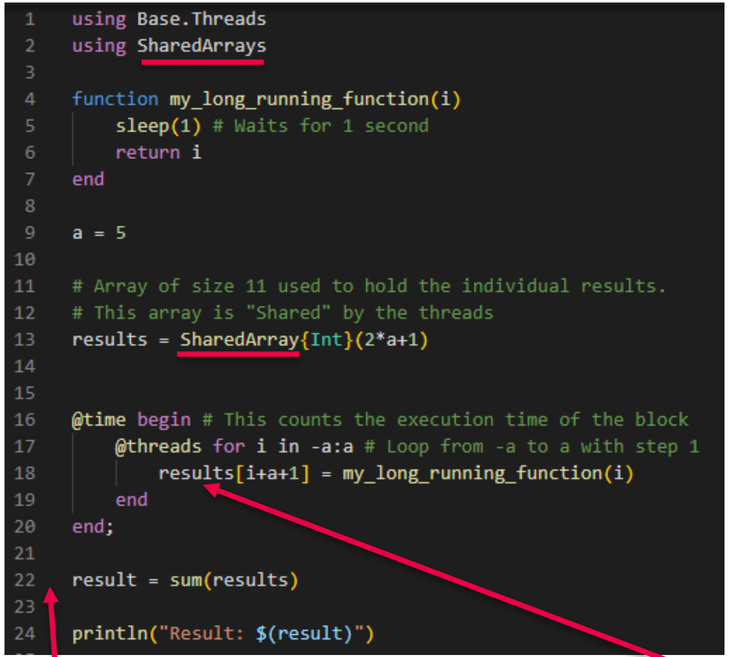
```julia
using Base.Threads
using SharedArrays

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

a = 5

# Array of size 11 used to hold the individual results.
# This array is "Shared" by the threads
results = SharedArray{Int}(2*a+1)


@time begin # This counts the execution time of the block
    @threads for i in -a:a # Loop from -a to a with step 1
        results[i+a+1] = my_long_running_function(i)
    end
end;

result = sum(results)

println("Result: $(result)")
```

1. We first write the individual results in a thread-safe array
2. We accumulate in the end

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time?
  - Result?

Parallelizing Julia for loops

File: `loop_3.jl`

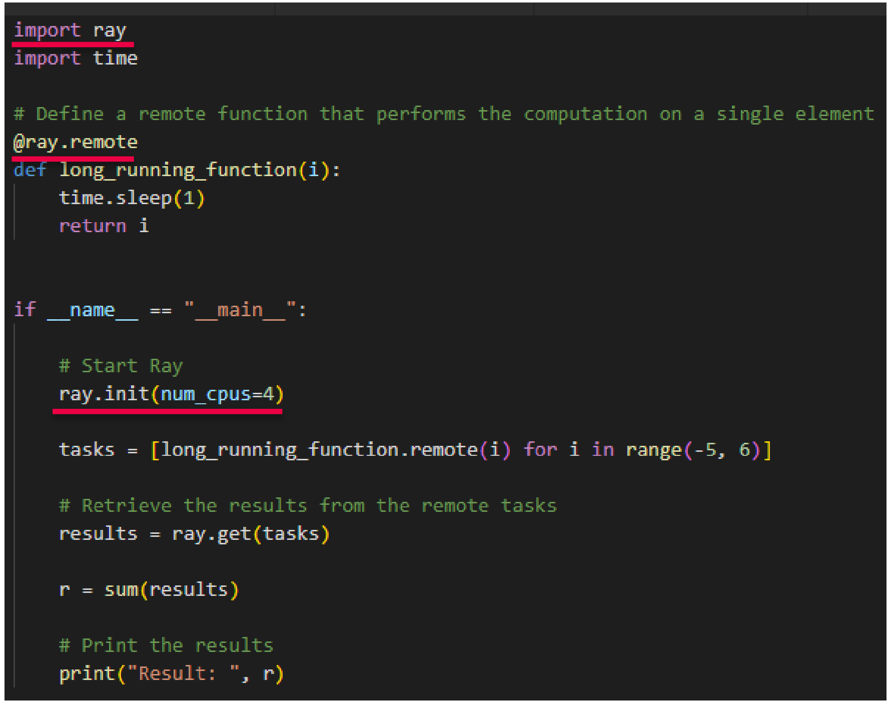


```julia
using Base.Threads
using SharedArrays

function my_long_running_function(i)
    sleep(1) # Waits for 1 second
    return i
end

a = 5

# Array of size 11 used to hold the individual results.
# This array is "Shared" by the threads
results = SharedArray{Int}(2*a+1)


@time begin # This counts the execution time of the block
    @threads for i in -a:a # Loop from -a to a with step 1
        results[i+a+1] = my_long_running_function(i)
    end
end;

result = sum(results)

println("Result: $(result)")
```

1. We first write the individual results in a thread-safe array
2. We accumulate in the end

- Assume we have 11 CPUs:
  - **Supercloud:** `LLsub -i -s 11`
  - **Engage:** `srun --pty --cpus-per-task=11 bash`
- Answer the following:
  - Loop execution time? **1 seconds**
  - Result? **0!**

Multithreading is language-specific: Python



```python
import ray
import time

# Define a remote function that performs the computation on a single element
@ray.remote
def long_running_function(i):
    time.sleep(1)
    return i


if __name__ == "__main__":

    # Start Ray
    ray.init(num_cpus=4)

    tasks = [long_running_function.remote(i) for i in range(-5, 6)]

    # Retrieve the results from the remote tasks
    results = ray.get(tasks)

    r = sum(results)

    # Print the results
    print("Result: ", r)
```

- Python code that performs exactly the same Julia task we saw before
- Personal opinion: Best parallelization library in python is **ray**

Takeaways

- Increasing CPU count **does not** necessarily make your program faster:
  - **First**, examine if your program can use multiple CPUs
  - **Then,** request more than 1 CPUs: Be mindful about the resources you request

- Some libraries (e.g. Gurobi, numpy) already exploit multiple cores
- ≠ If you write your own code, you need to parallelize it **yourself:**
  - Each language has its own syntax for multiprocessing/multithreading
  - Be careful about **Race Conditions:**
    - **Multiple threads accessing the same variable at the same time**
    - Consider using “**thread-safe**” variables or **mutexes/locks.** More Info

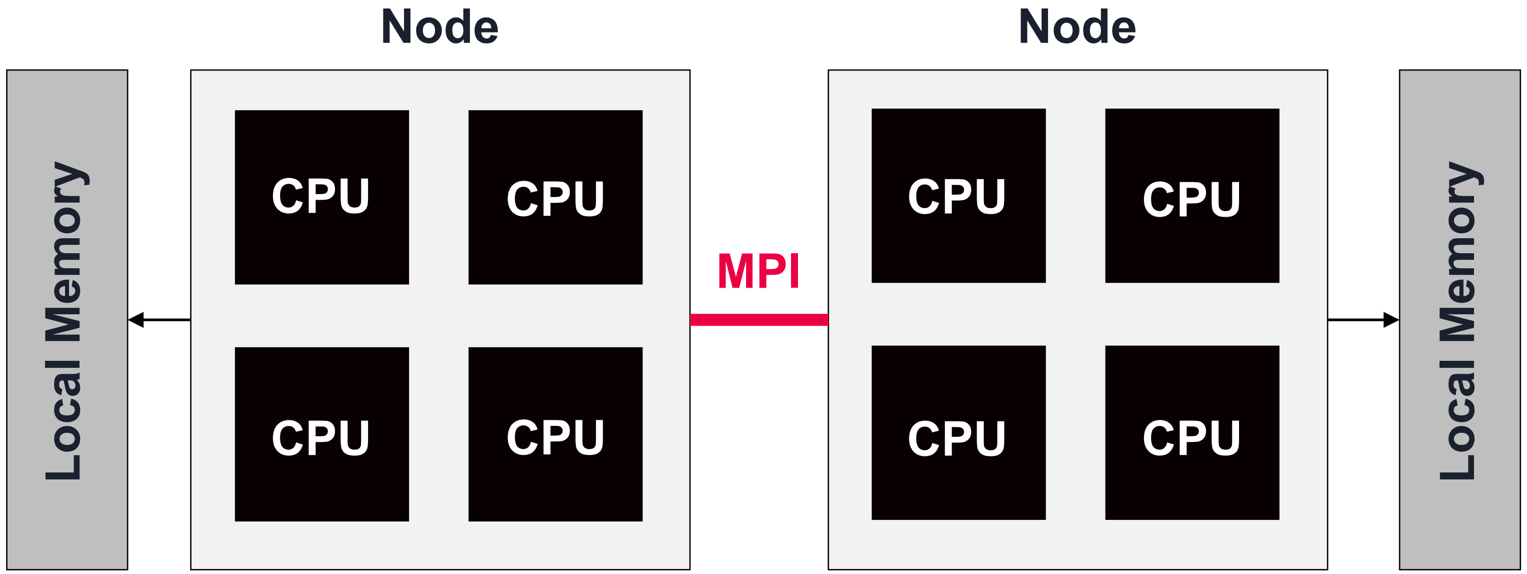
Parallel Computing on SuperCloud: MPI

Adapted from Alex Schmid & Lauren Milechin

Distributed computing

- Multithreading can only happen in the **same node**
- Multithreading is limited by the **number of CPUs** in the node:
    - Using more threads than CPUs doesn’t increase performance
- How can we parallelize across nodes & CPUs?
- **MPI!**

Distributed computing

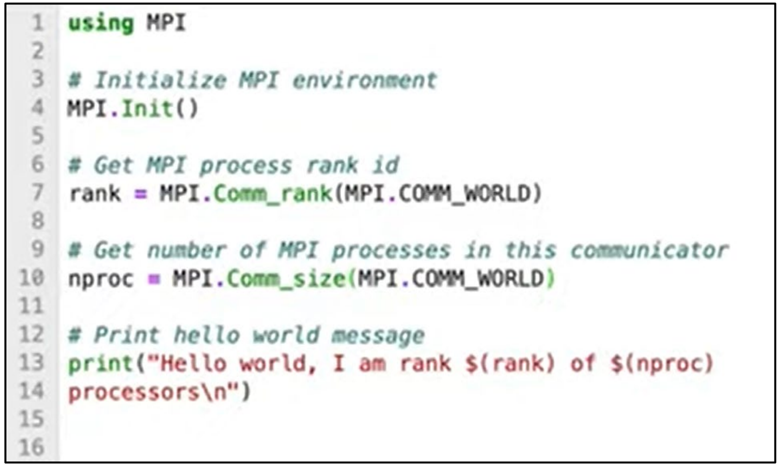


Parallelization

We can use **MPI (Message Passing Interface)** to parallelize our code!

Unlike the commands we've run so far today, integrating MPI requires language-specific code

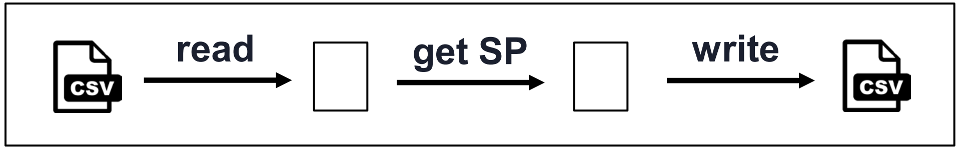
→ We will use **MPI.jl** to integrate parallelization into our Julia code



```julia
using MPI

# Initialize MPI environment
MPI.Init()

# Get MPI process rank id
rank = MPI.Comm_rank(MPI.COMM_WORLD)

# Get number of MPI processes in this communicator
nproc = MPI.Comm_size(MPI.COMM_WORLD)

# Print hello world message
print("Hello world, I am rank $(rank) of $(nproc)
processors\n")


```

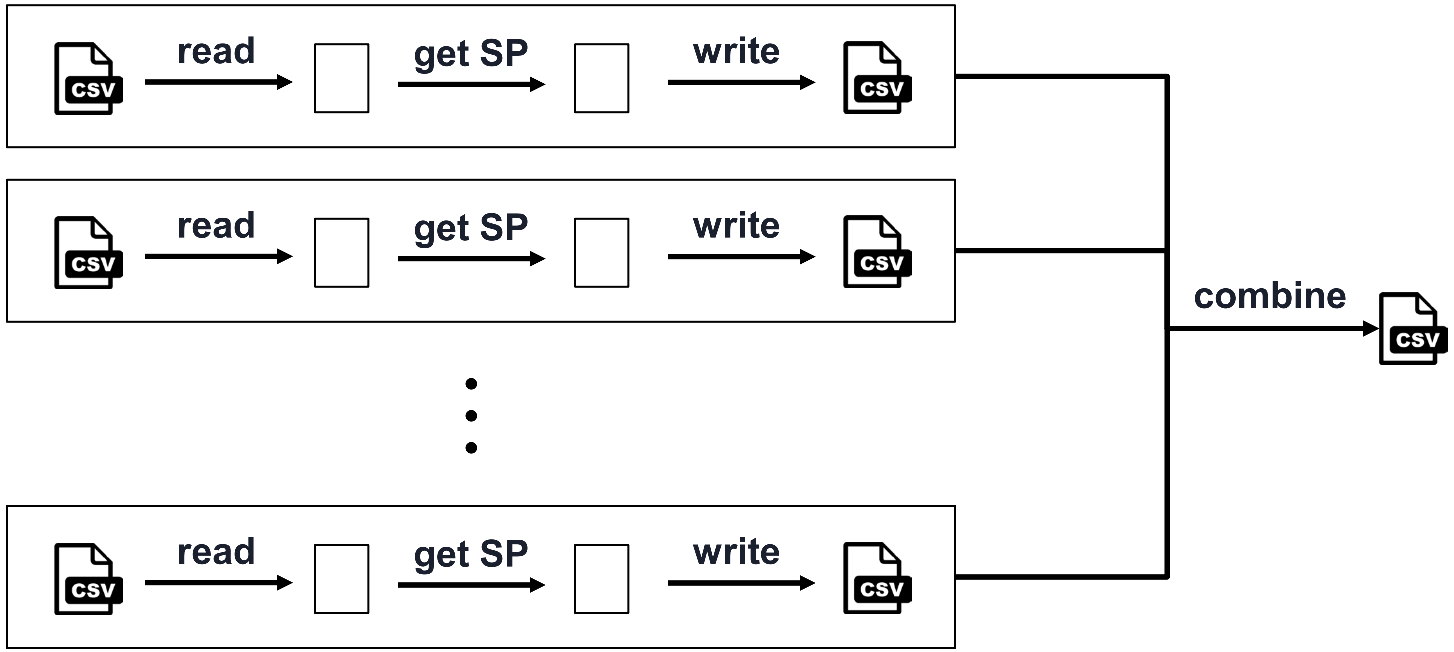
Option 1: One big for loop

```
for runid in 1:50
```

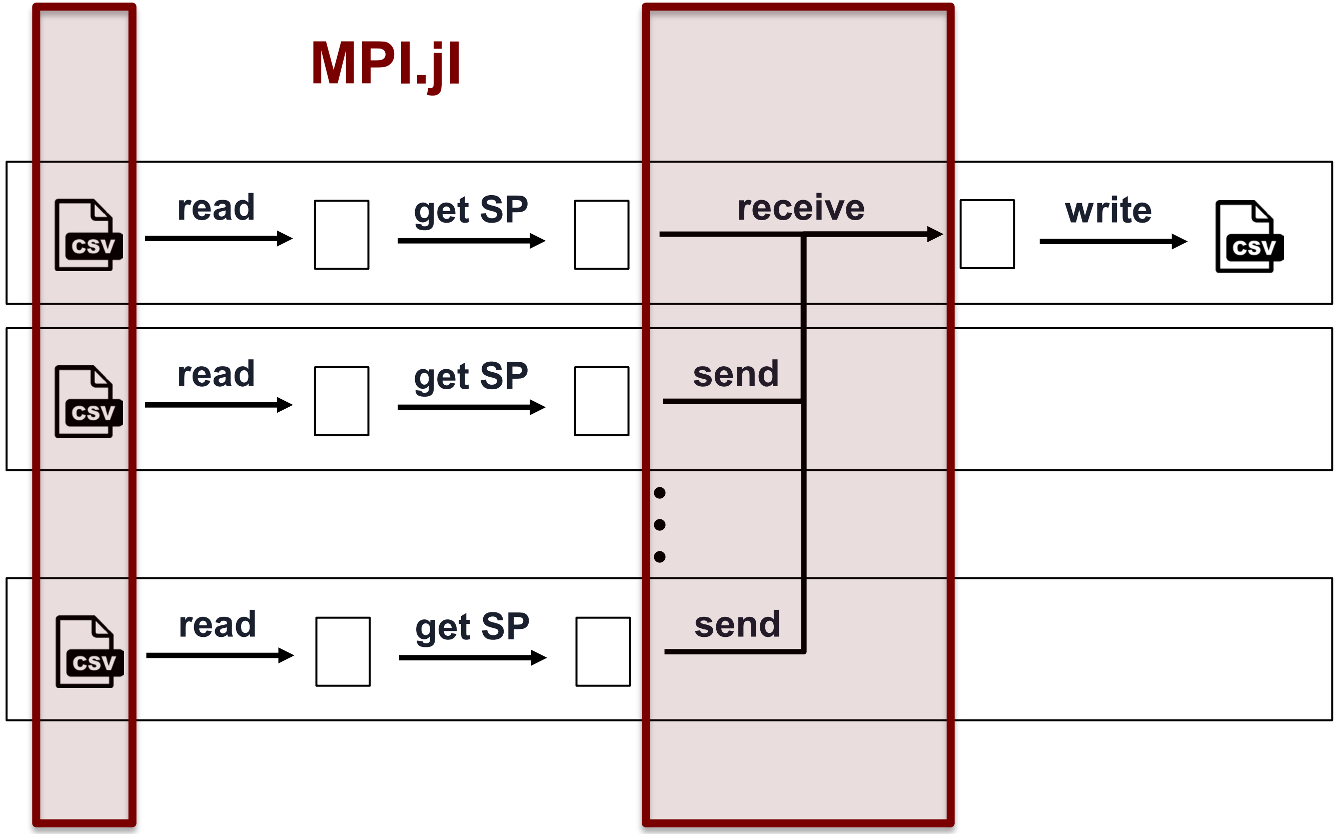


```
end
```

Option 2: Job array / MapReduce



Option 3: Parallelize across nodes & cores



MPI Commands

Initialize MPI environment	`MPI.Init()`
MPI communicator	`MPI.COMM_WORLD`
The number of MPI processes	`MPI.Comm_size`
The rank / ID of a given process	`MPI.Comm_rank`
Send message	`MPI.send(message, recvr_rank, my_id, comm)`
Receive message	`MPI.recv(sender_rank, my_id, comm)`

Running parallelized code with MPI

1. Navigate to `5_parallel_mpi`

2. Load mpi, add MPI.jl, and build MPI.jl

```
module load Julia/1.8.5
module load mpi/openmpi-4.1.5
julia
using Pkg
Pkg.add(“MPI”)
Pkg.build(“MPI”)
```

If you get a wrong MPI version warning during mpirun:

```
Pkg.add(“MPIPreferences”)
julia --project -e 'using MPIPreferences; MPIPreferences.use_system_binary()’
```

3. Kickoff `hello_mpi.jl` with mpi and four cores

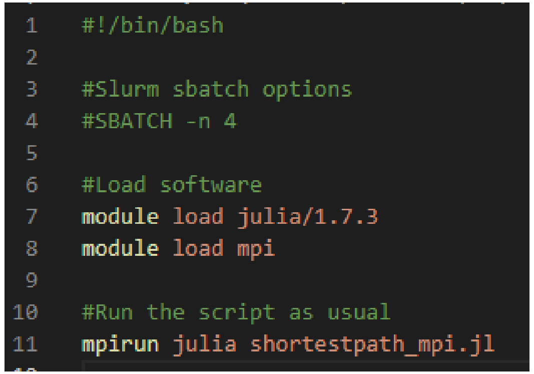
```
mpirun –n 4 julia hello_mpi.jl
```

Running parallelized code with MPI

1. Navigate to `5_parallel_mpi`

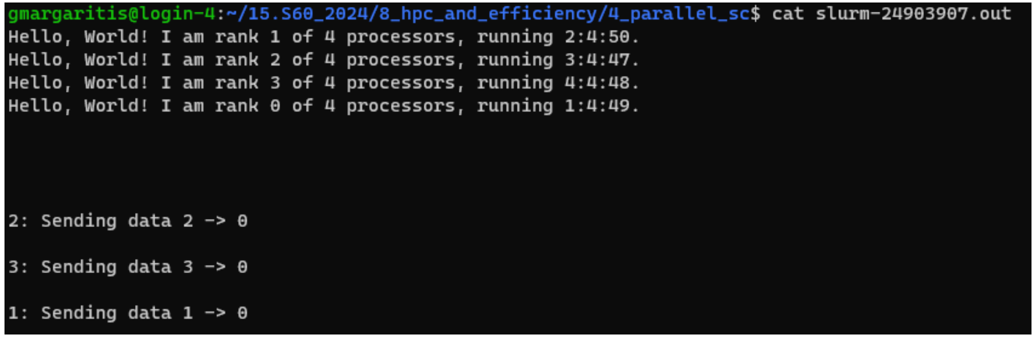
2. Run `sbatch sp_mpi.sh` which runs the following script:

   This runs the shortest path algorithm we saw in previous parts, but it uses MPI to split the scenarios across the different CPUs



```
#!/bin/bash

#Slurm sbatch options
#SBATCH -n 4

#Load software
module load julia/1.7.3
module load mpi

#Run the script as usual
mpirun julia shortestpath_mpi.jl
```

3. Observe the output file:



```
gmargaritis@login-4:~/15.S60_2024/8_hpc_and_efficiency/4_parallel_sc$ cat slurm-24903907.out
Hello, World! I am rank 1 of 4 processors, running 2:4:50.
Hello, World! I am rank 2 of 4 processors, running 3:4:47.
Hello, World! I am rank 3 of 4 processors, running 4:4:48.
Hello, World! I am rank 0 of 4 processors, running 1:4:49.




2: Sending data 2 -> 0

3: Sending data 3 -> 0

1: Sending data 1 -> 0
```

Managing Dependencies

Challenges in Managing Dependencies

- Have you ever had problems installing packages in Engaging?
    - **Example:** I want to install pytorch/transformers to run LLMs in Engaging

- Why is it more difficult to install packages in the cluster than in your computer?
    - **No root access!**
    - Outdated compilers (gcc) and standard library (glibc)
    - We are forced to have a setup that works with the specific module versions that are in the cluster

Solving Package Installation Issues

Solving Package Installation Issues

Try different versions of the package you are installing (e.g. pytorch 2.2, 2.4)

Solving Package Installation Issues

Try different versions of the package you are installing (e.g. pytorch 2.2, 2.4)	
Try different versions of the language (e.g. different python) or compiler (gcc) using **`module load`**	

Solving Package Installation Issues

Try different versions of the package you are installing (e.g. pytorch 2.2, 2.4)
Try different versions of the language (e.g. different python) or compiler (gcc) using `` `module load` ``
Install the missing package/dependencies **from source** (since you cannot do `` `sudo apt install` ``)

Solving Package Installation Issues

Try different versions of the package you are installing (e.g. pytorch 2.2, 2.4)

Try different versions of the language (e.g. different python) or compiler (gcc) using `**module load**`

Install the missing package/dependencies **from source** (since you cannot do `sudo apt install`)

Ask the cluster admins to update a package/module

Solving Package Installation Issues

Try different versions of the package you are installing (e.g. pytorch 2.2, 2.4)

Try different versions of the language (e.g. different python) or compiler (gcc) using **`module load`**

Install the missing package/dependencies **from source** (since you cannot do `sudo apt install`)

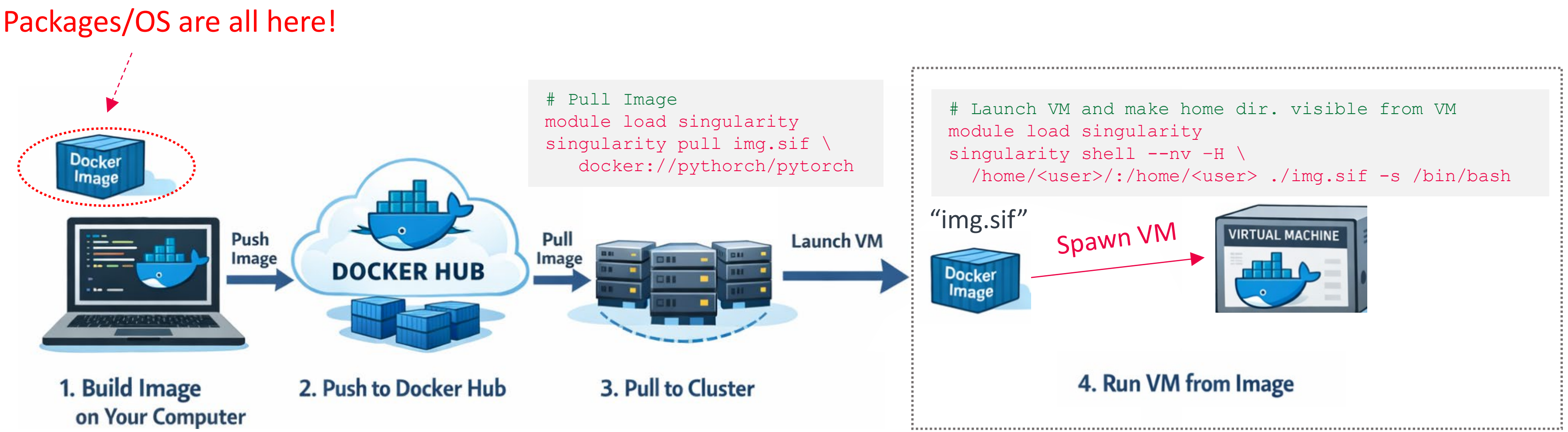
Ask the cluster admins to update a package/module

Create a Virtual Machine (VM) using **Docker** and run it on eofe/eosloan using **Singularity/Apptainer**

Using VMs in Engaging

- **Docker/Singularity/Apptainer:** Software that allows us to:
  - Create Virtual Machines (VM) with whatever OS and packages we want in them (we have root access)
  - Run these VMs in different platforms (e.g. Engaging) without having to interfere the environment at all!
  - Share the VMs configuration with other people (e.g. collaborators) so that all of us work in exactly the same environment

Using VMs in Engaging

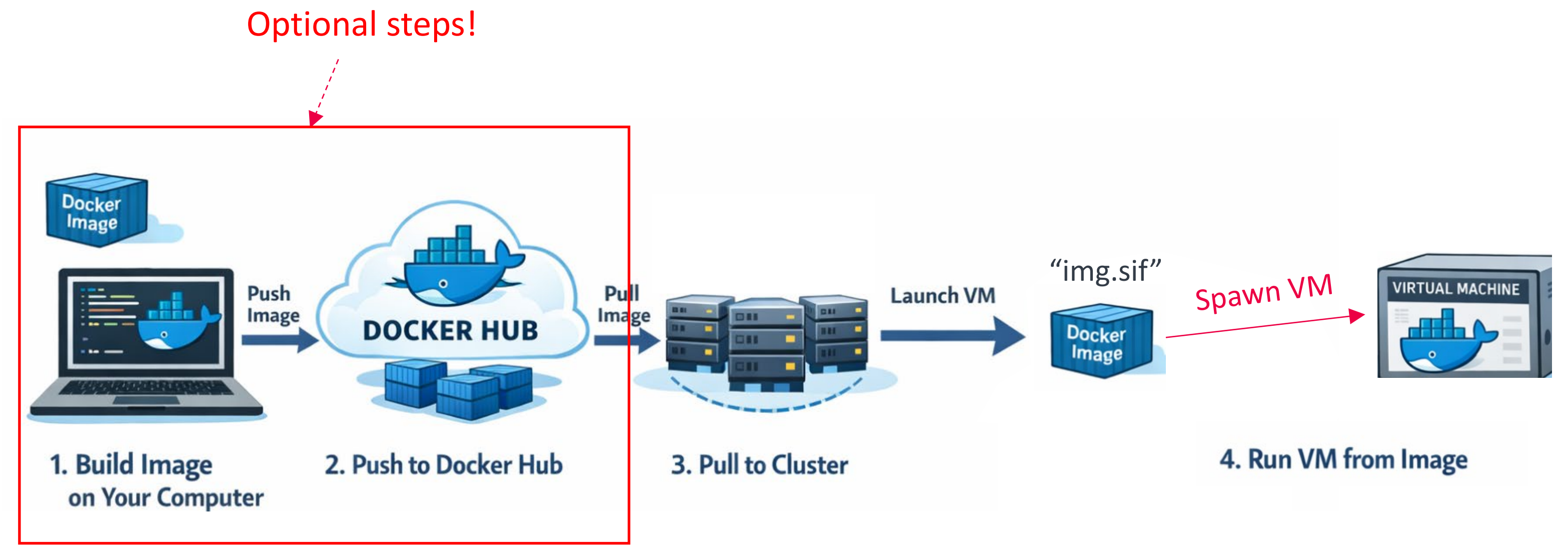


```
# Pull Image
module load singularity
singularity pull img.sif \
    docker://pythorch/pytorch
```

```
# Launch VM and make home dir. visible from VM
module load singularity
singularity shell --nv -H \
  /home/<user>/:/home/<user> ./img.sif -s /bin/bash
```

- **Docker Image** (a file that contains all of the OS and packages needed to spawn a VM)
- Push the docker image in Docker Hub (like GitHub but for docker images)
- From Engaging, pull the image from Docker Hub
- Spawn the VM using apptainer or singularity (similar to docker)

Using VMs in Engaging



- **Good news:** First 2 steps can be **optional:**
  - If someone else has created an image for our use-case, we can use it directly
  - You can look for already built images in Docker Hub (e.g. pytorch with cuda here)
  - Also: All of my collaborators can use my image **without installing anything**!

Conclusion

- This should not be the first thing you try, but:
  - When all of the other methods fail, this is the only one that can work
  - **Extremely powerful**

- No need to “pollute” the cluster environment with dependencies:
  - Different projects can have different requirements
  - You can build an image per project

- Once you build the image, you can use it forever or share it with others

Questions?

Final assignment in `assignment`

Due: Sunday, Feb 1 at 11:59pm **(hard deadline)**

Best of luck in the upcoming semester ☺

Thank you!