

# Introduction to using a High-Performance Computing cluster

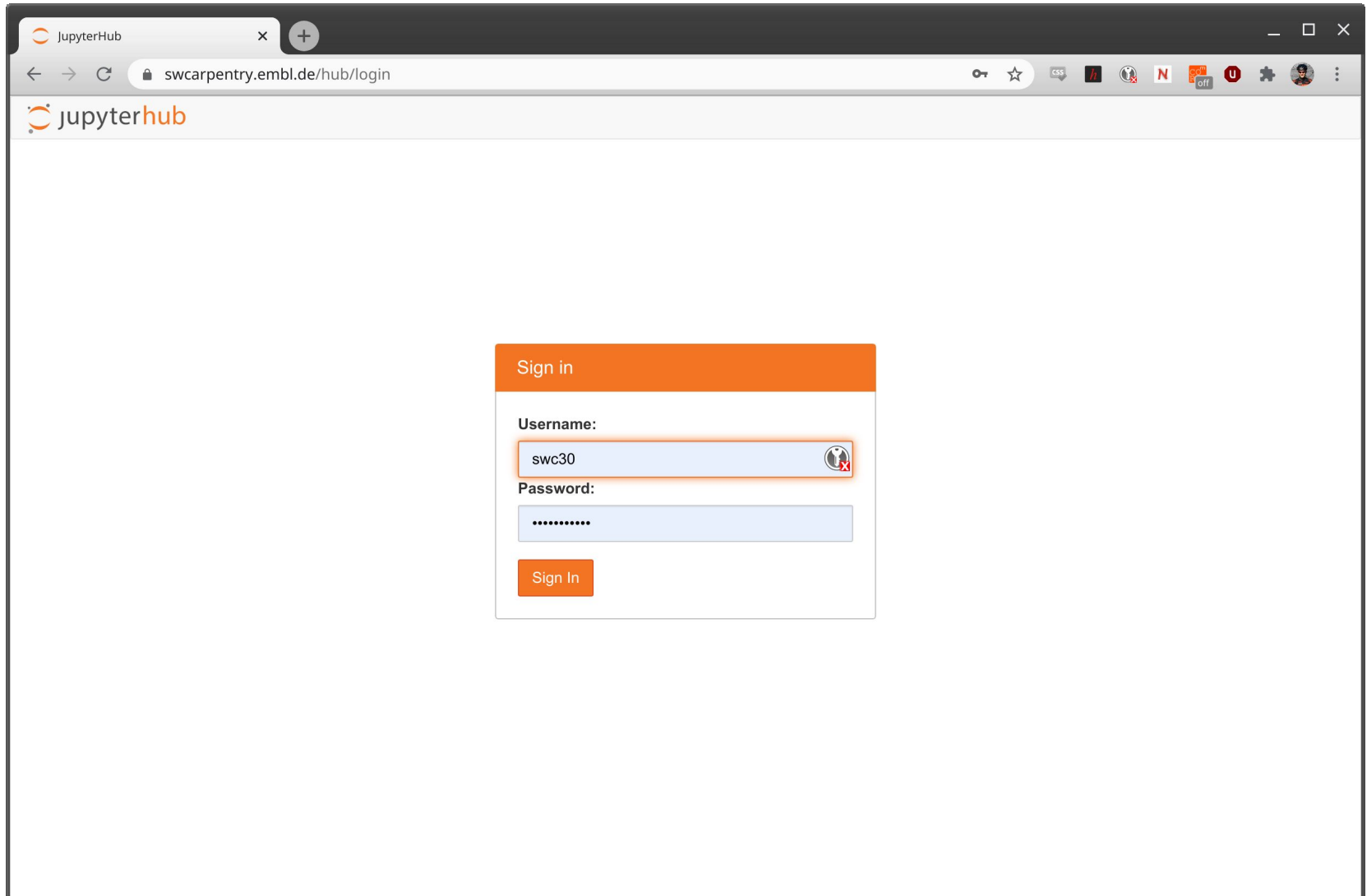
---

Mike Smith

  @grimbough

# Connecting to our cluster:

`https://swcarpentry.embl.de/`



The screenshot shows a web browser window with the JupyterHub interface. The address bar displays `swcarpentry.embl.de/hub/login`. The page features the JupyterHub logo at the top left. In the center, there is a 'Sign in' form with an orange header. The form includes a 'Username:' field containing 'swc30' and a 'Password:' field with masked characters. A 'Sign In' button is located at the bottom of the form. The browser's address bar and various extension icons are visible at the top of the window.

Sign in

Username:

swc30

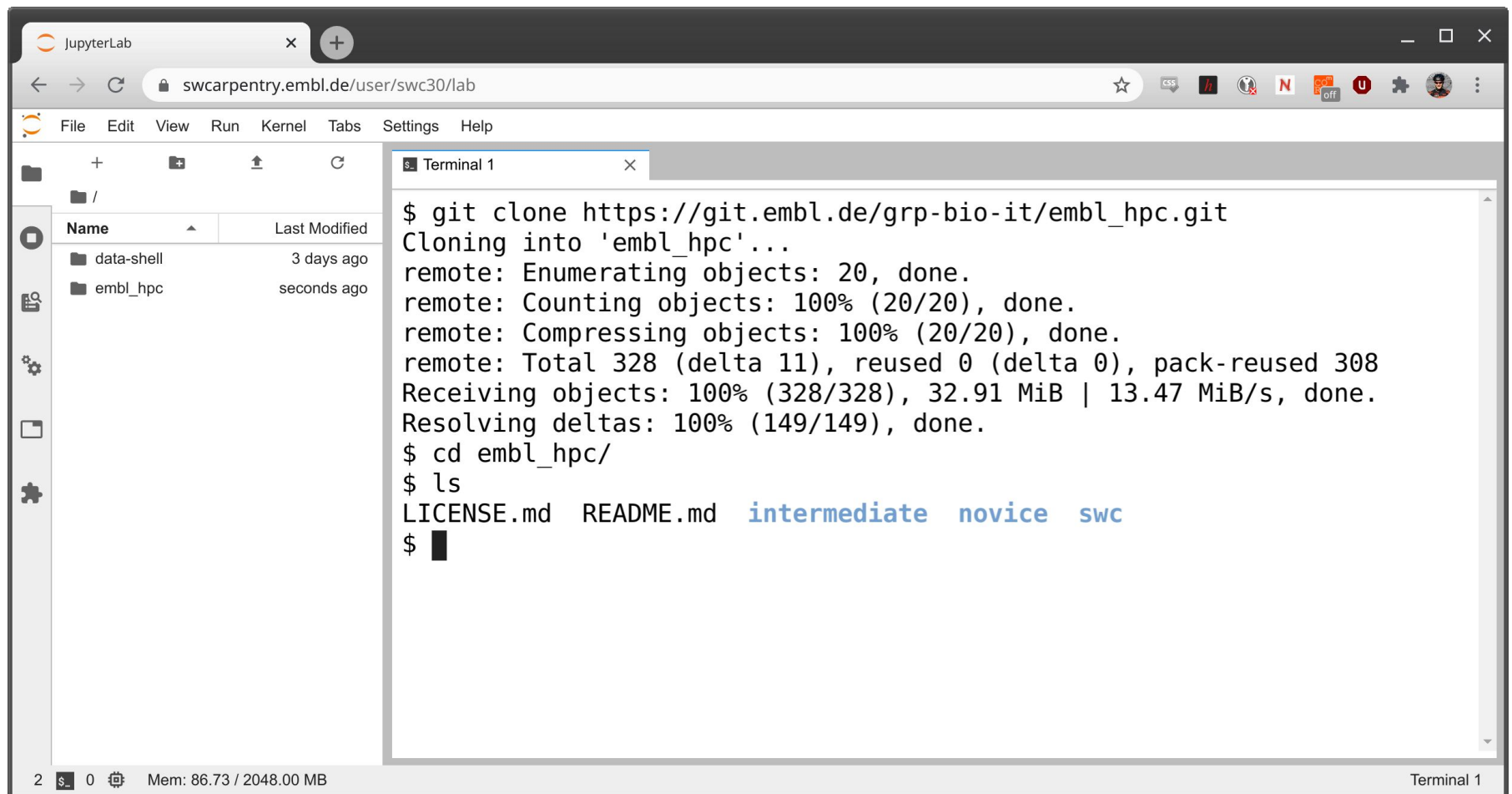
Password:

.....

Sign In

# Download example programs

```
git clone https://git.embl.de/grp-bio-it/embl_hpc.git
```



The screenshot shows a JupyterLab web interface in a browser window. The address bar displays `swcarpentry.embl.de/user/swc30/lab`. The left sidebar contains a file explorer showing a directory structure with `data-shell` (modified 3 days ago) and `embl_hpc` (modified seconds ago). The main area features a terminal window titled "Terminal 1" with the following output:

```
$ git clone https://git.embl.de/grp-bio-it/embl_hpc.git
Cloning into 'embl_hpc'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 328 (delta 11), reused 0 (delta 0), pack-reused 308
Receiving objects: 100% (328/328), 32.91 MiB | 13.47 MiB/s, done.
Resolving deltas: 100% (149/149), done.
$ cd embl_hpc/
$ ls
LICENSE.md  README.md  intermediate  novice  swc
$
```

The bottom status bar indicates 2 files, 0 errors, and memory usage of 86.73 / 2048.00 MB. The terminal window is labeled "Terminal 1" in the bottom right corner.

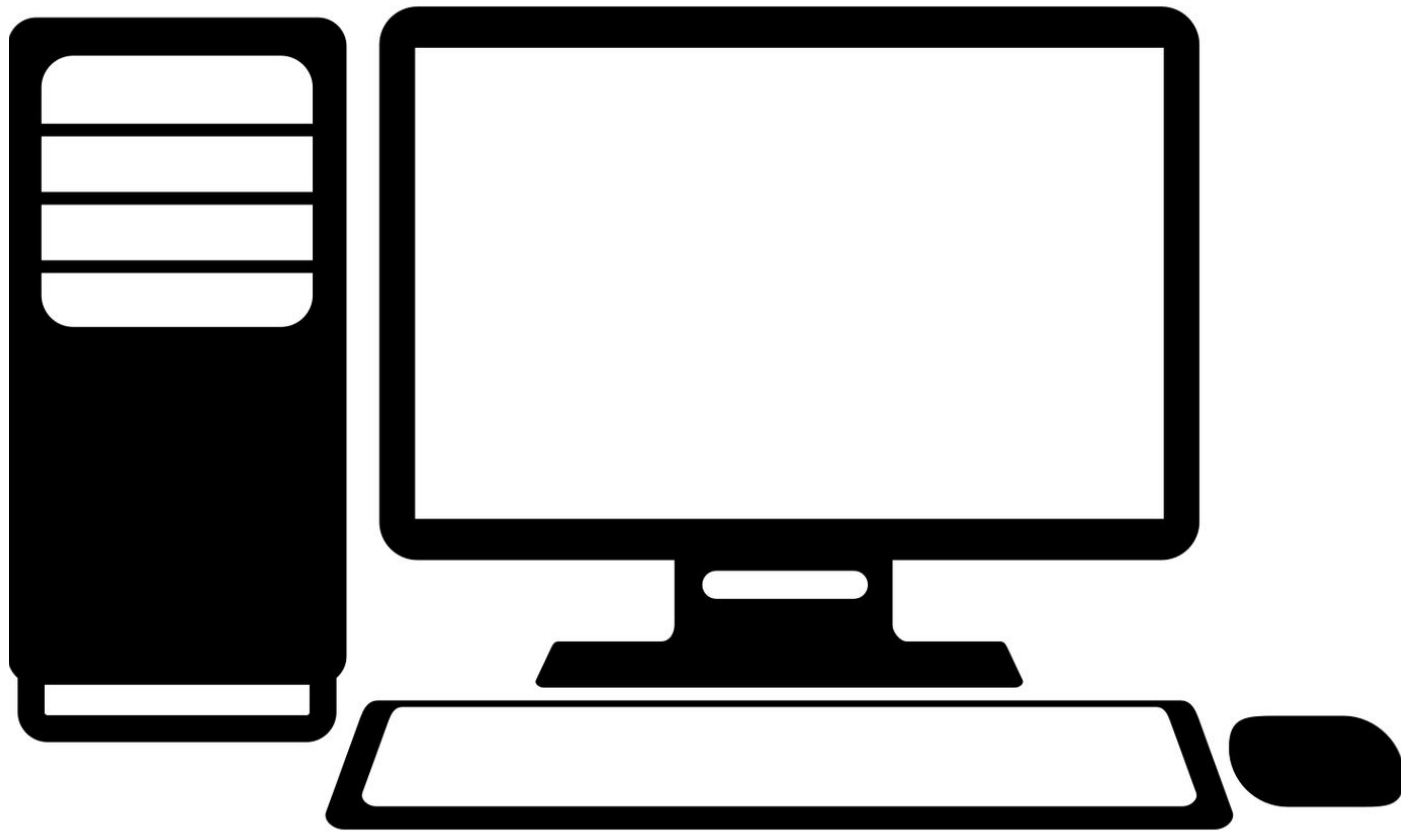
# When is HPC useful?

- When you realise your standard computer is too small or too slow for your data
  - Compute Intensive: Task requiring a large amount of computation
    - e.g. more rigorous sequence alignment
  - Memory Intensive: Task requiring a large amount of memory
    - e.g. scaling up from bacteria to human genome
  - Data Intensive: Task involves operating on a large amount of data
    - e.g. 50 human genomes

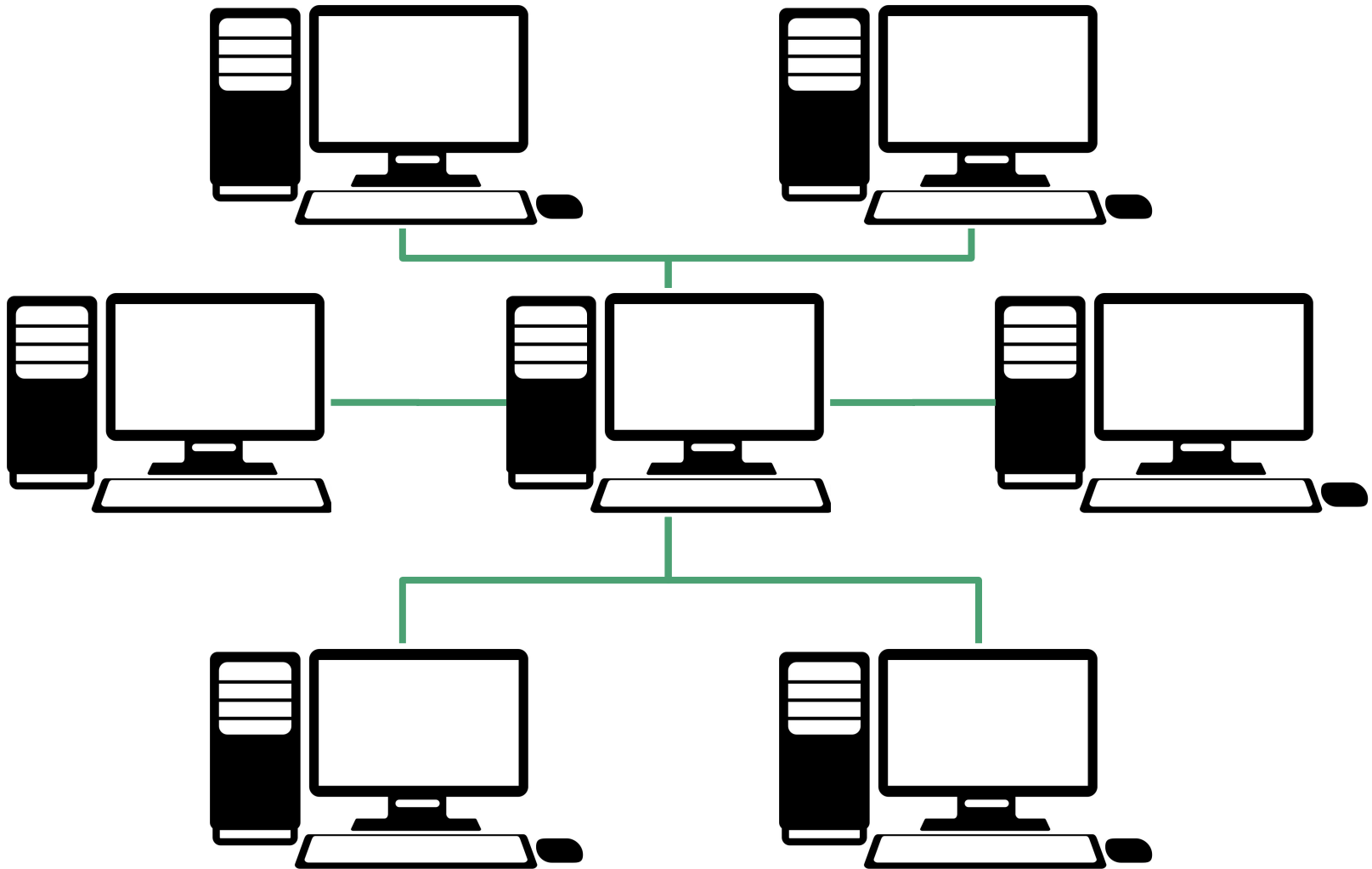
# Types of Cluster - Shared Memory



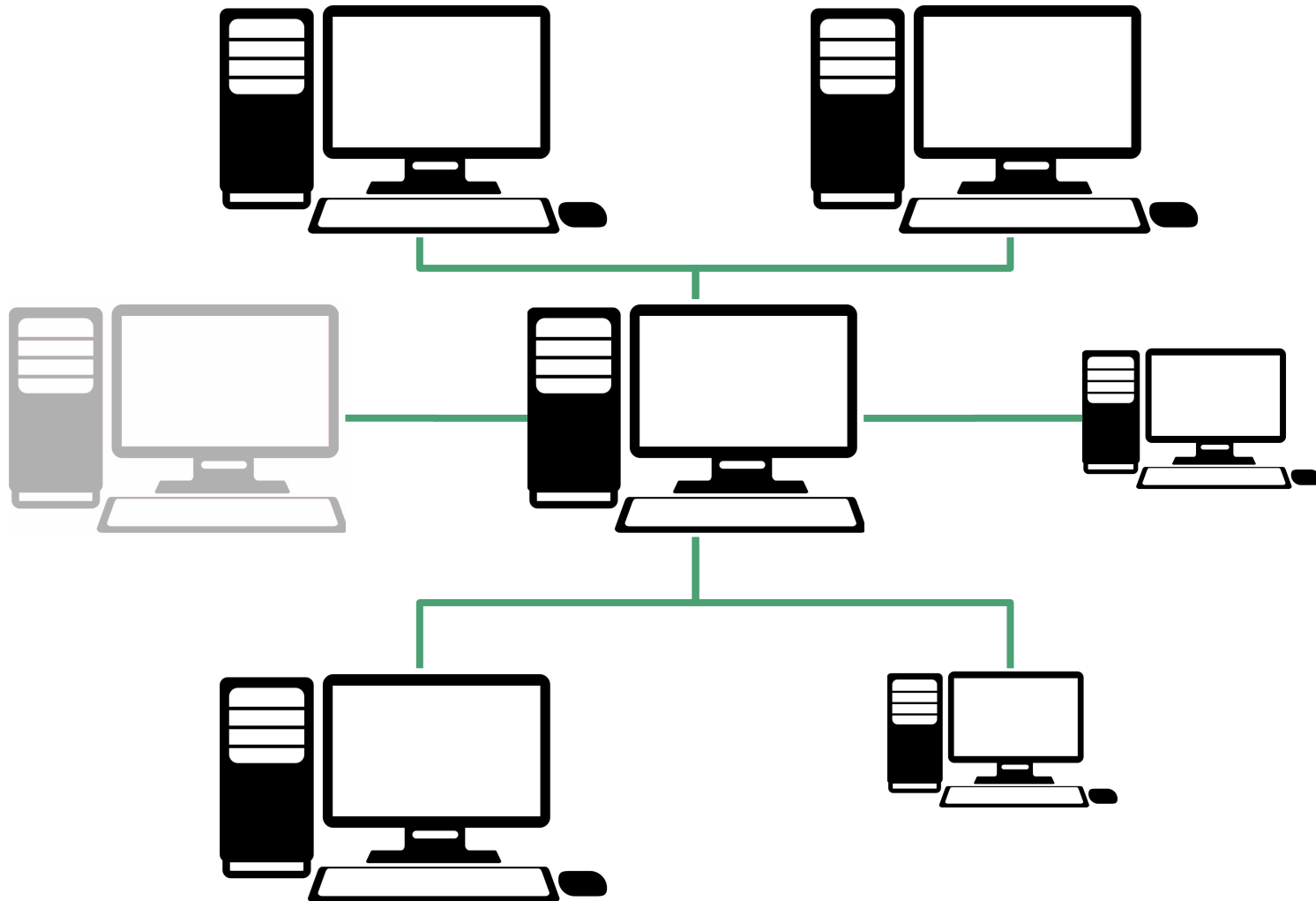
# Types of Cluster - Shared Memory



# Types of Cluster - Distributed Memory



# Types of Cluster - Distributed Memory





# How do we work with a distributed cluster?

- Typically interact with a single 'head' node
- A job scheduler manages where and when tasks are run
  - There are many options available e.g. LSF, Torque, SLURM, Condor, Univa Grid Engine
- Matches job requirements with available resources
- If no slots are available a job will wait until resources are available



# Our example cluster

- Consists of five nodes:
  - **master** - 14 cores, 32GB RAM (doesn't do any work)
  - **worker-1-1** - 4 cores, 8GB RAM
  - **worker-2-1** / **worker-2-2** - 1 core, 2GB RAM
  - **worker-3-1** - 8 cores, 16GB RAM
- Not enough resources for us all to run programs simultaneously
- Clusters are about sharing!
- `scontrol show nodes` - shows makeup of the cluster

# Our first cluster job

```
hostname
```

- `srun` - submits a job to the cluster

```
srun hostname
```

# Example python program

- Program should be present in the 'swc/exercises' directory
- Takes two arguments
  - -t Time to wait in seconds
  - -l Length of list to create

```
./hpc_example.py -t 10 -l 100
```

- Prints arguments to screen ⇒ creates list ⇒ waits ⇒ prints memory usage ⇒ exits

# Example python program

```
srun ./hpc_example.py -t 10 -l 100
```

- Not super convenient, use **sbatch** to run in background

```
sbatch ./hpc_example.py -t 60 -l 100
```

- `squeue` - lists current jobs

```
squeue --Format="JobID,UserName,State,Reason,TimeUsed,NodeList"
```

# Redirecting output

- Not always helpful to use default file
- Use `--output=output.txt`

```
sbatch --output=output.txt \  
    ./hpc_example.py -t 20 -l 100
```

## Try creating a larger list

```
sbatch --output=output.txt \  
    ./hpc_example.py -t 30 -l 50000000
```

# Requesting Additional Resources

- Sharing resources between users is a key function of the job scheduler
- Jobs are killed if they try to use more than their allocated share
- View configuration with:

```
scontrol show partition
```



# Requesting Additional Resources

- Sharing resources between users is a key function of the job scheduler
- Jobs are killed if they try to use more than their allocated share
- We can raise this limit using `--mem=500`

```
sbatch --mem=500 \  
      --output=output.txt \  
      ./hpc_example.py -t 30 -l 500000000
```

# Try reserving a LARGE amount of memory

```
sbatch --mem=8000 \  
      --output=output.txt \  
      ./hpc_example.py -t 30 -l 50000000
```

- Look at the running jobs with `squeue`
- Only a small number of jobs will be allowed to run simultaneously

# Understanding the compute requirements of your task is key to effectively using a HPC cluster

- Ask for too much
  - Job will wait for a long time necessarily
  - Reserve resources you don't need
- Ask for too little
  - Job may be killed without finishing
  - You start using resources you haven't asked for, potentially slowing things down for everyone
- Run tests on a subset
- Some programs let you specify resource usage, explore the manual / try Google

# Interactive jobs

- Sometimes we want to interact with a job
  - e.g. if we're testing code works

```
srun --pty bash
```

- All other parameters can also be used as before

```
srun --mem=250 --pty bash
```

# Using sbatch

- Jobs can be submitted as scripts as well

```
sbatch batch_job.sh
```

- Try modifying *batch\_job.sh* to run the python program twice with different parameters

# Job dependencies

- We can make part 2 run only when part 1 is finished

```
jid=$(sbatch --parsable batch_job.sh)  
sbatch --dependency=afterok:$jid batch_job.sh
```

# Things we haven't covered

- We have discussed only memory, jobs can have many more resource requirements
  - In particular the number of cores / threads you want to use
- Job checkpoints, suspension and resumption
- Executing more complex parallel programs
- ...

Questions?

---