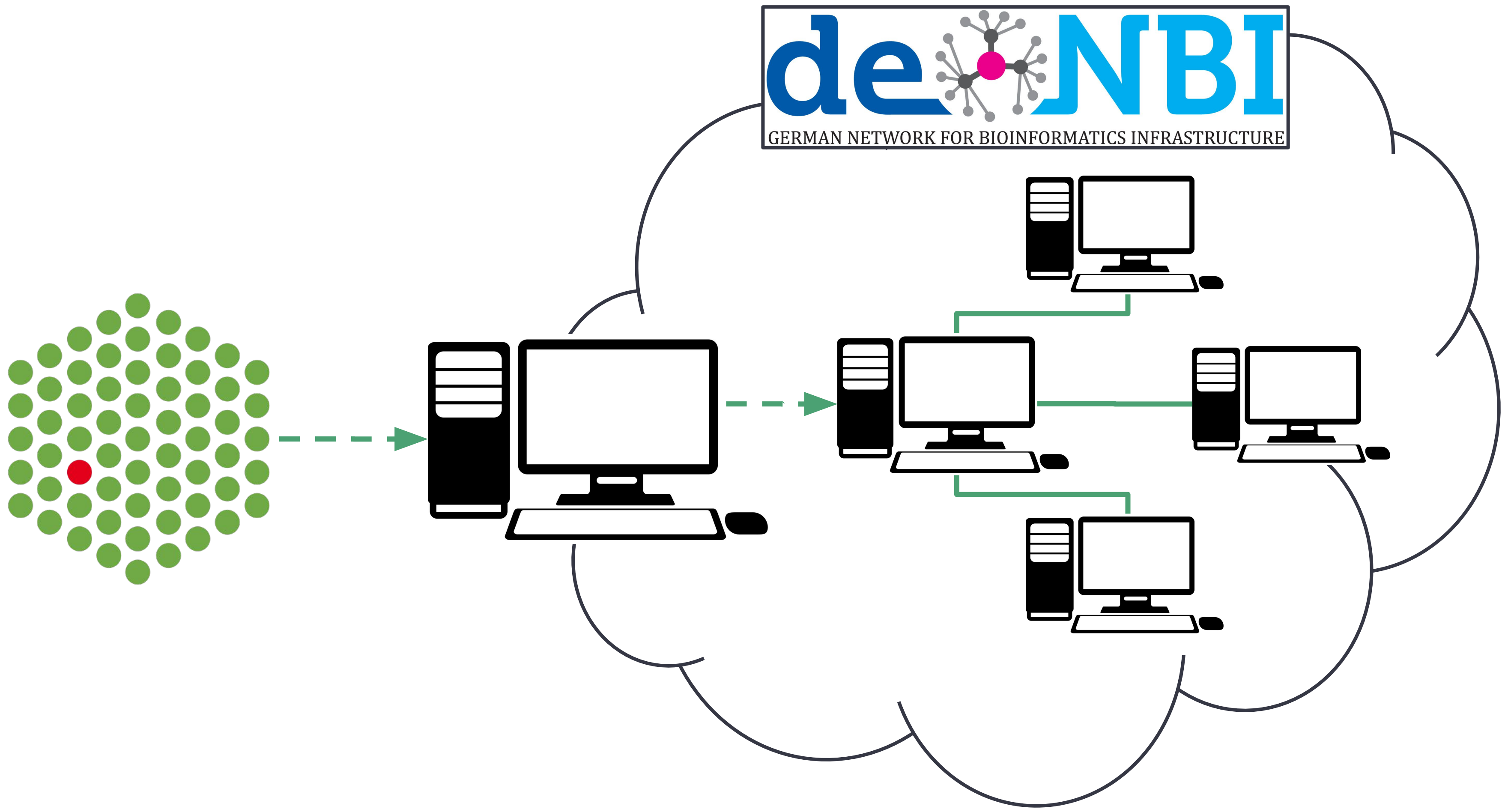
Introduction to using a High-Performance Computing cluster

Mike Smith

Connecting to our cluster



Connecting to our cluster

- Connect using

```
ssh bq_11denbi@129.206.69.162
```

- and then

```
ssh user##@172.16.72.70
```

- Replace “##” with the number of your workstation e.g. `user10`
- Password: `SoftwareC`

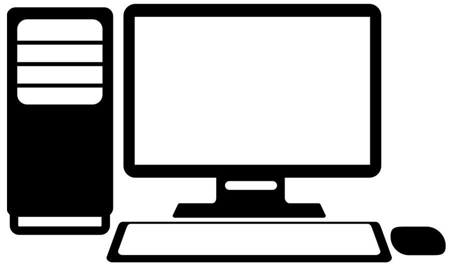
Download example programs

```
git clone https://github.com/grimbough/embl_swc_hpc.git
```

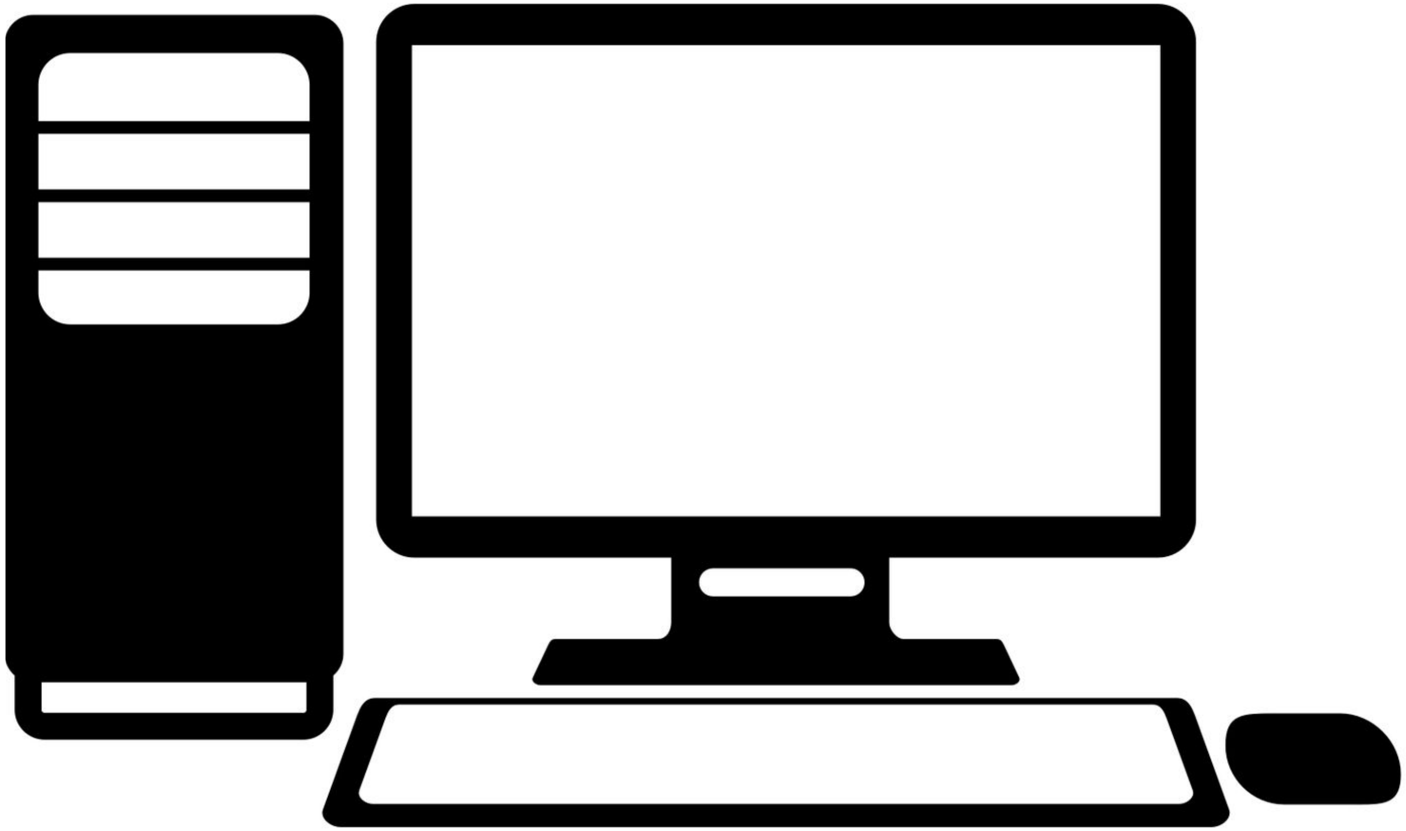
When is HPC useful?

- 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 rigourous 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

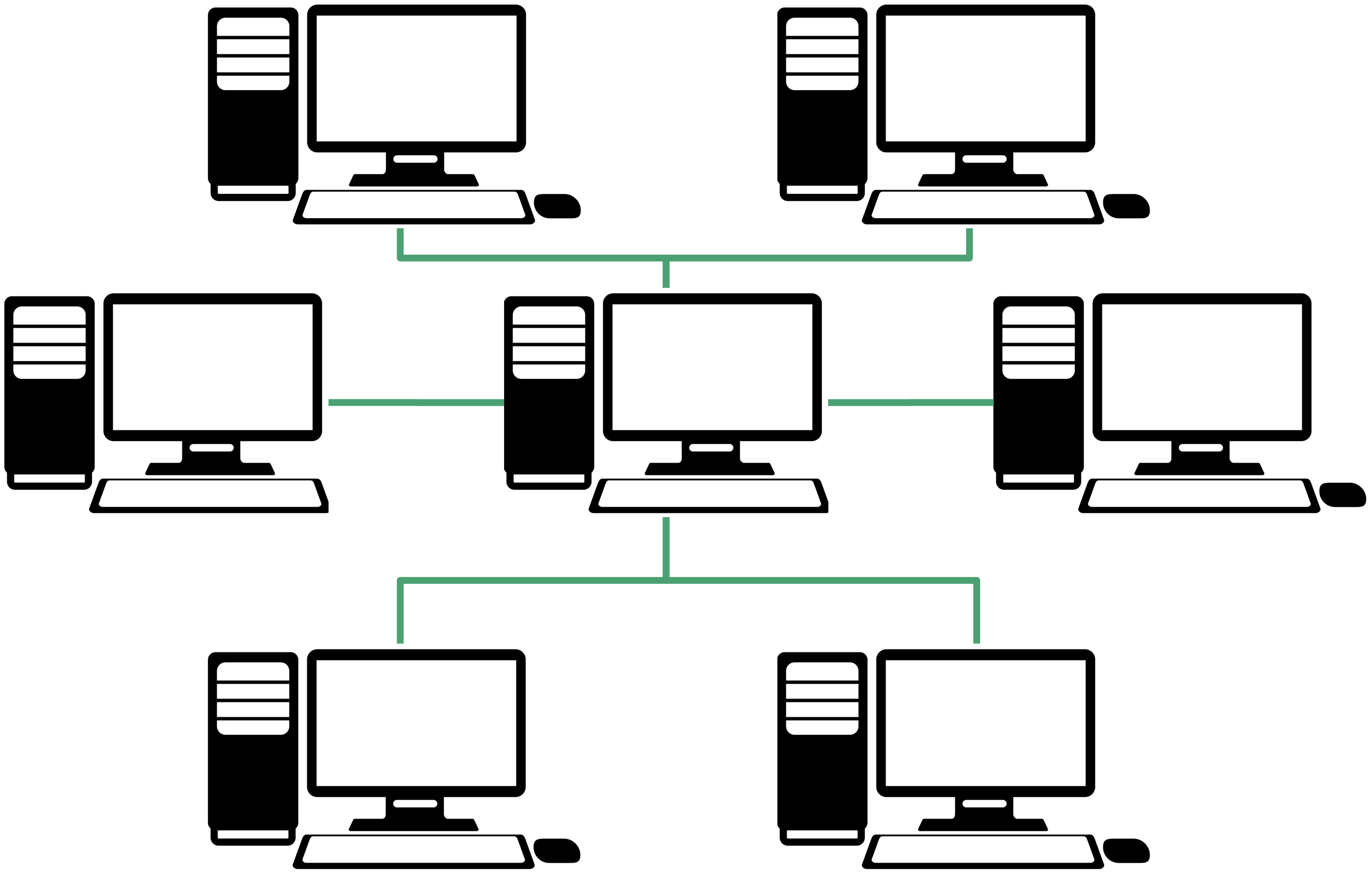
Types of Cluster - Shared Memory



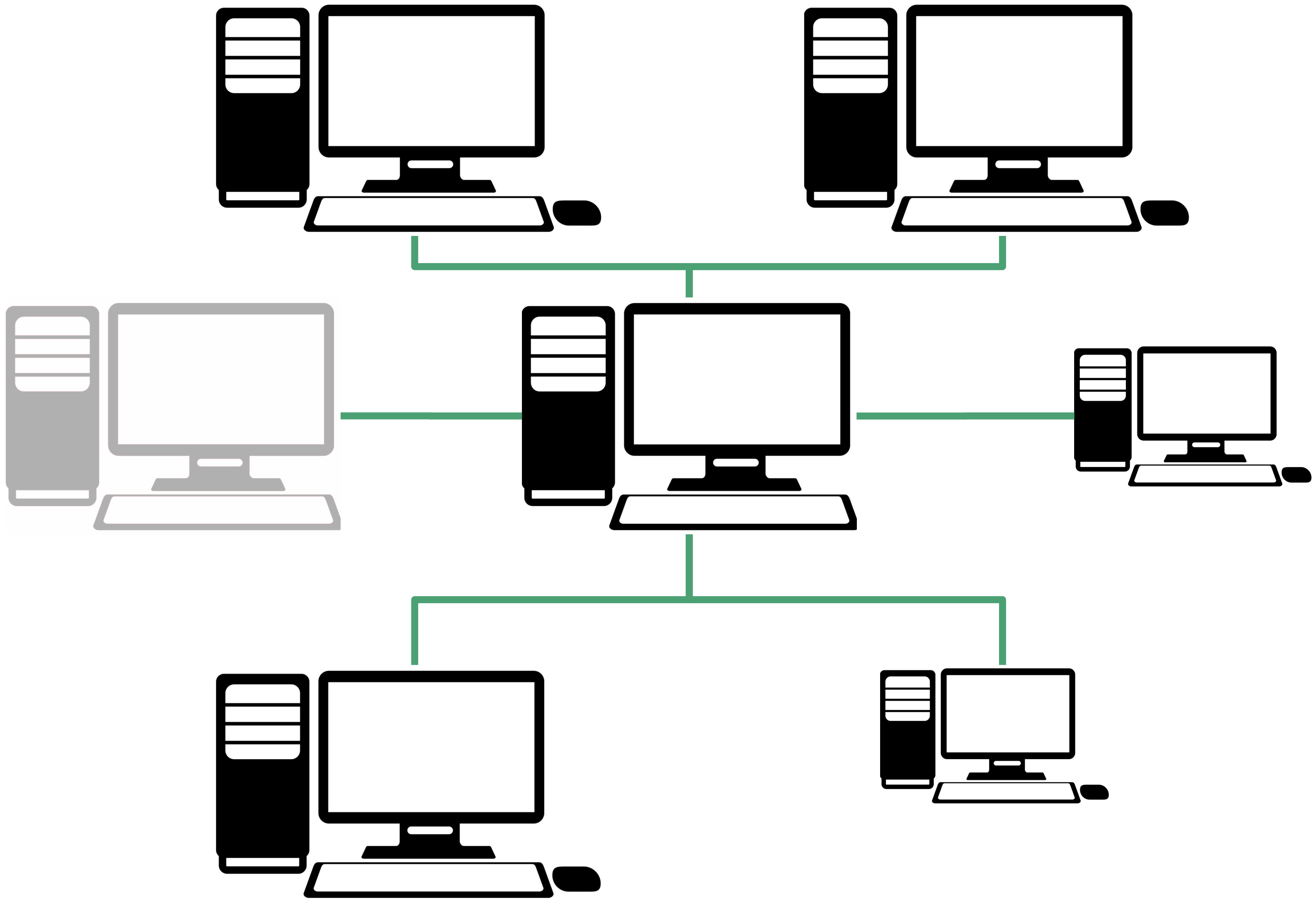
Types of Cluster - Shared Memory



Types of Cluster - Distributed Memory



Types of Cluster - Distributed Memory



How do we work with a distributed cluster?

- Typically interact with a single ‘Master’ 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 four nodes:
  - `master` - 2 cores, 4GB RAM (doesn't do any work)
  - `node1` - 2 cores, 4GB RAM
  - `node2` - 4 cores, 8GB RAM
  - `node3` - 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 '*exercises*' directory
- Takes two arguments
    - `-t` Time to wait in seconds
    - `-l` Length of list to create (don't go over 4,000,000 !)

```
./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 120 -l 100
```

- `squeue` - lists current jobs

Redirecting output

- Not always helpful to print things to screen
- Use `srun --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 5000000
```

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=250`

```
sbatch --mem=250 \
     --output=output.txt \
     ./hpc_example.py -t 30 -l 5000000
```

Try reserving a LARGE amount of memory

```
sbatch --mem=8000 \
       --output=output.txt \
       ./hpc_example.py -t 30 -l 5000000
```

- 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, so read the manual

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?