Antikythera mechanism

Image: Marsyas assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims)., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=469865

Date: 89 BCE

Energy use: zero

FLOPS: zero

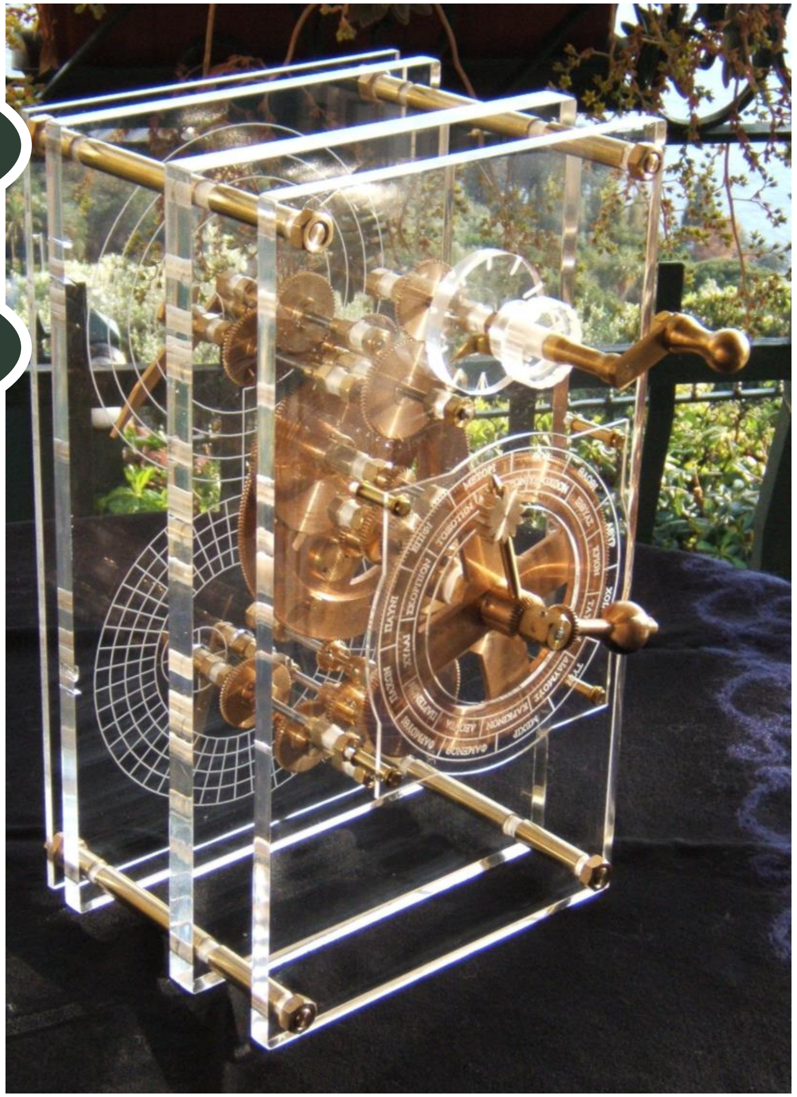
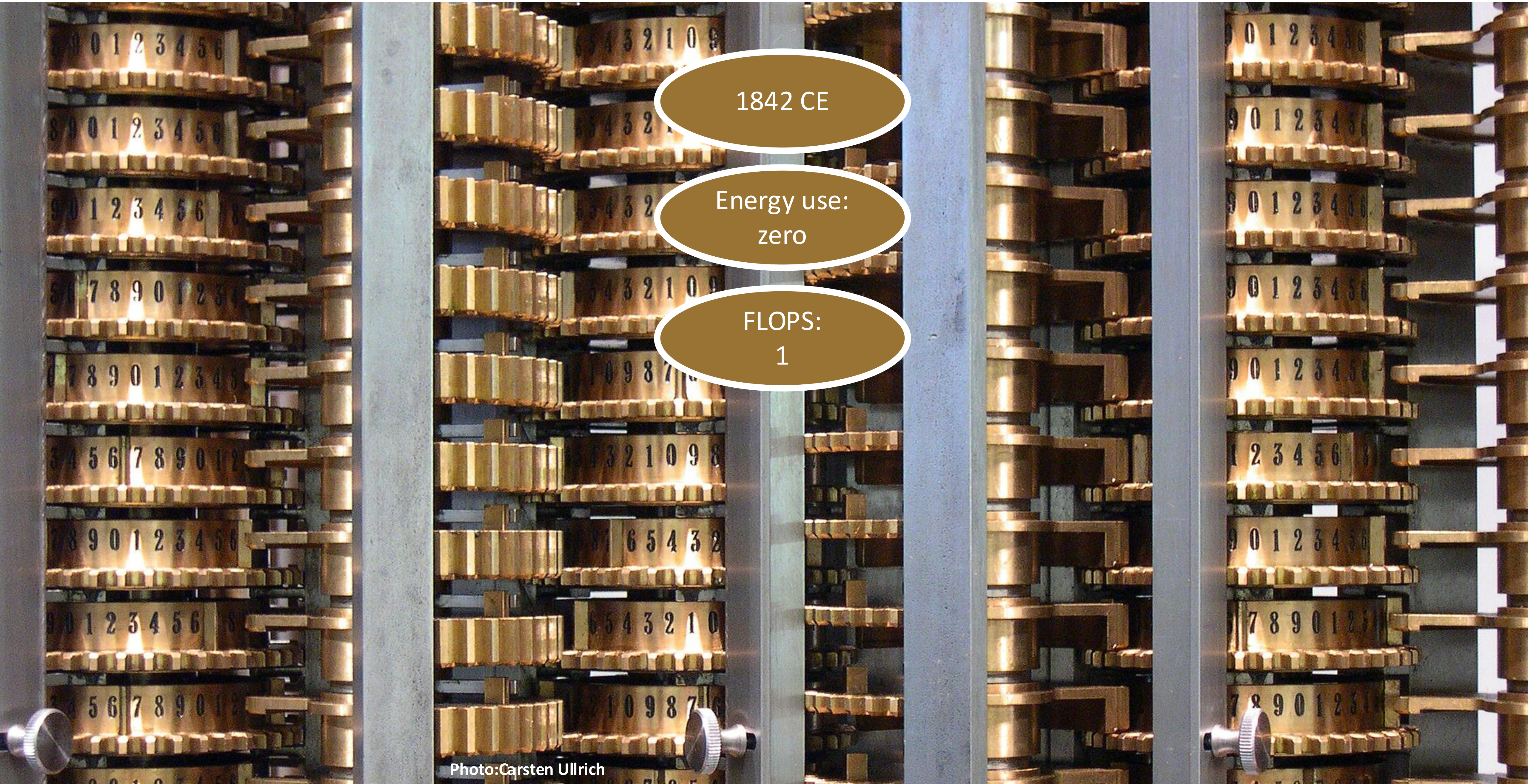


Image: By I, Mogi, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=2523740

Difference engine



1842 CE

Energy use: zero

FLOPS: 1

Photo:Carsten Ullrich

Tianhe-2 天河-2

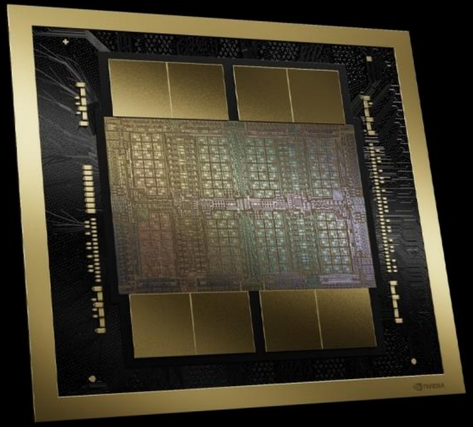
2013

Energy use:
24 Mw

FLOPS:
4×10^{15}

Space:
720 m^2

Nvidia Blackwell B300 GPU



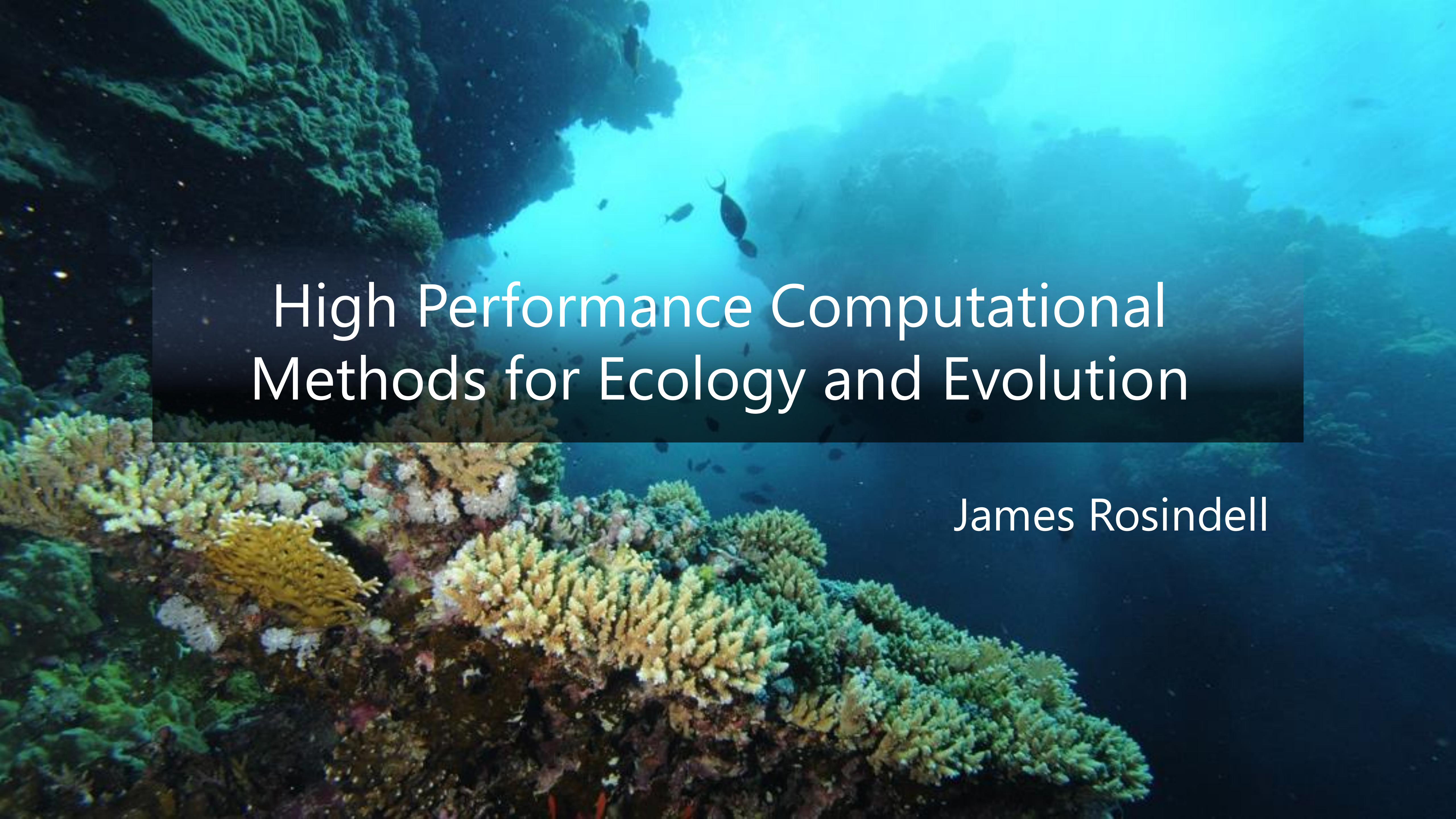
2025

Energy use:
1.400 KW

FLOPS:
3×10^{16}

Space:
$16\ cm^2$

Image: Nvidea, fair use



High Performance Computational Methods for Ecology and Evolution

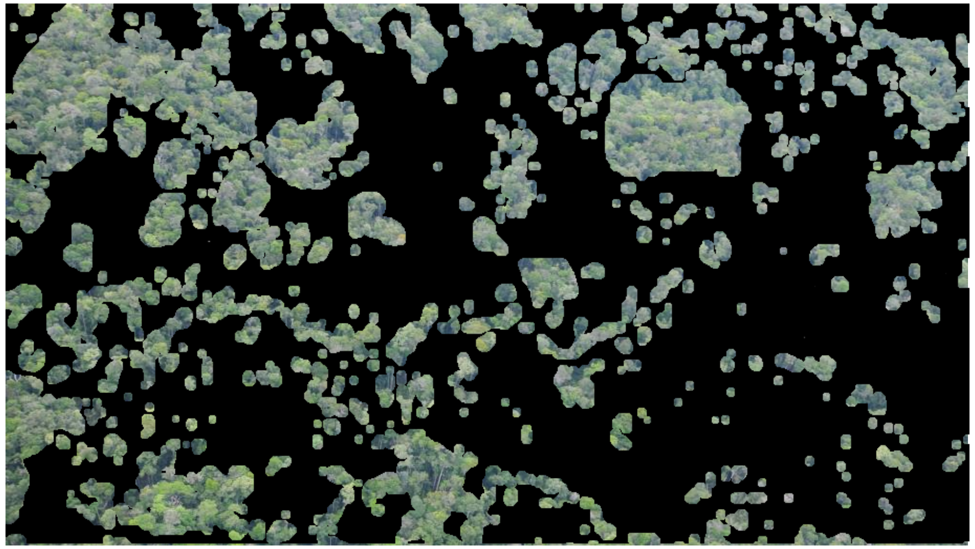
James Rosindell

1 Parallel computing

2 Community simulation

3 Population simulation

4 Applications



Course objectives

Understand high-performance computing and its benefits

Run code in parallel and process the data produced

Build simulations of populations and communities

Demonstrate good programming practice in R

Parallel computing

1. What is HPC and why is it useful?

2. How to manage your files

3. How to run jobs on the cluster

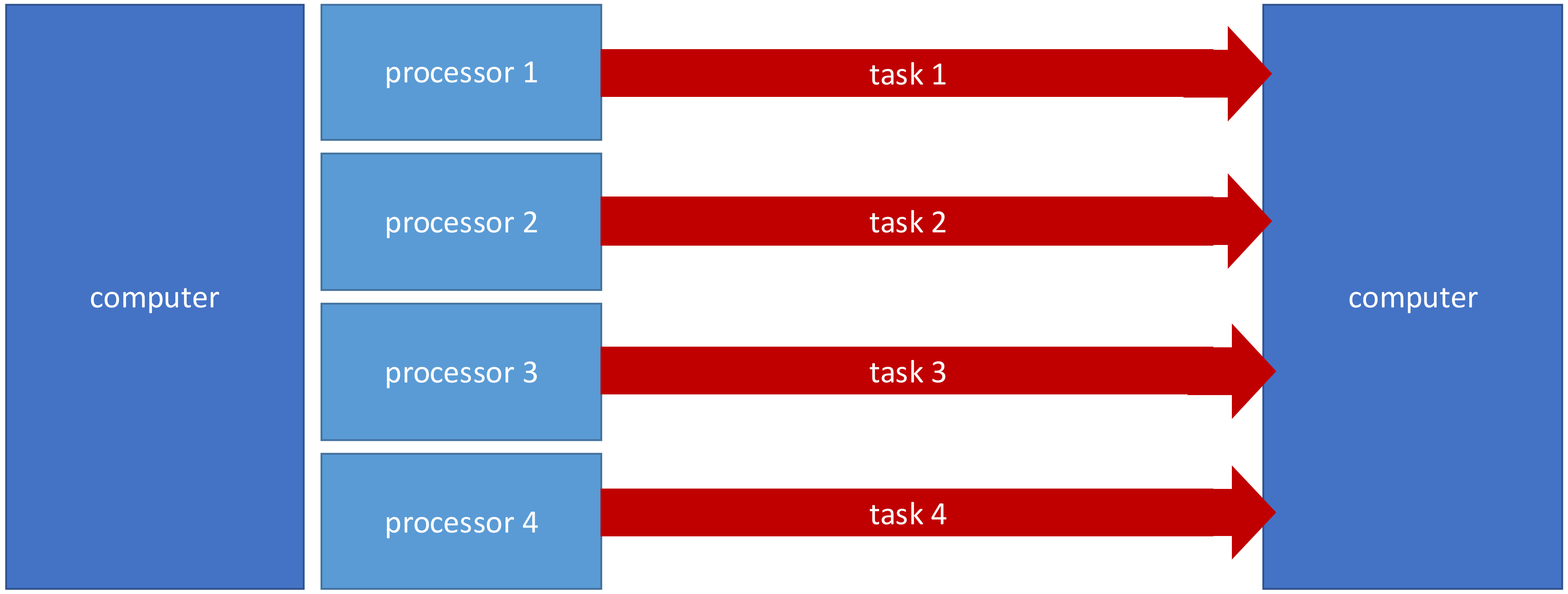
Types of High Performance Computing (HPC)

- Traditionally, HPC could include supercomputers or computer clusters
- Nowadays, a computer termed a “supercomputer” is typically a parallel system – parallelisation is the most efficient way to increase power
- As a result, HPC almost always refers to parallel computing!

- This could take different forms, for example:
    - A single computer with many processors
    - A cluster of many ordinary computers
    - Many computers which access a shared memory

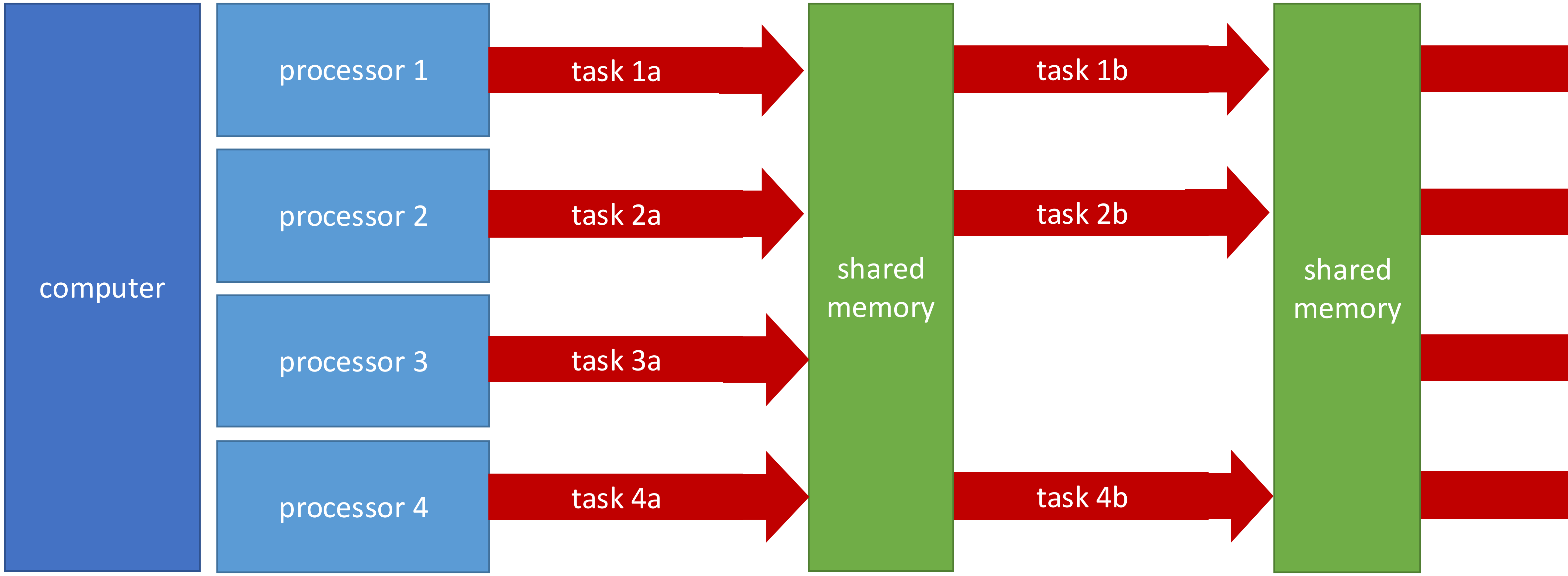
What is parallel computing?

- When many computations are carried out simultaneously



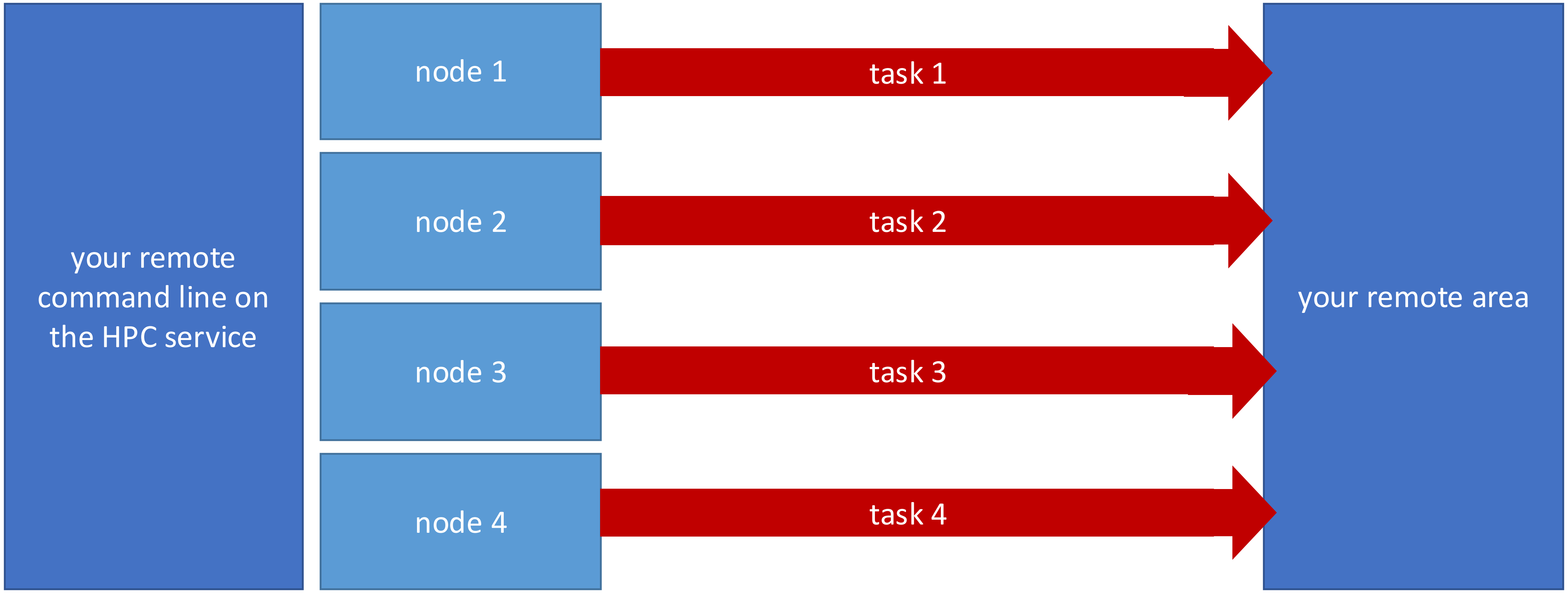
What is parallel computing?

- When many computations are carried out simultaneously

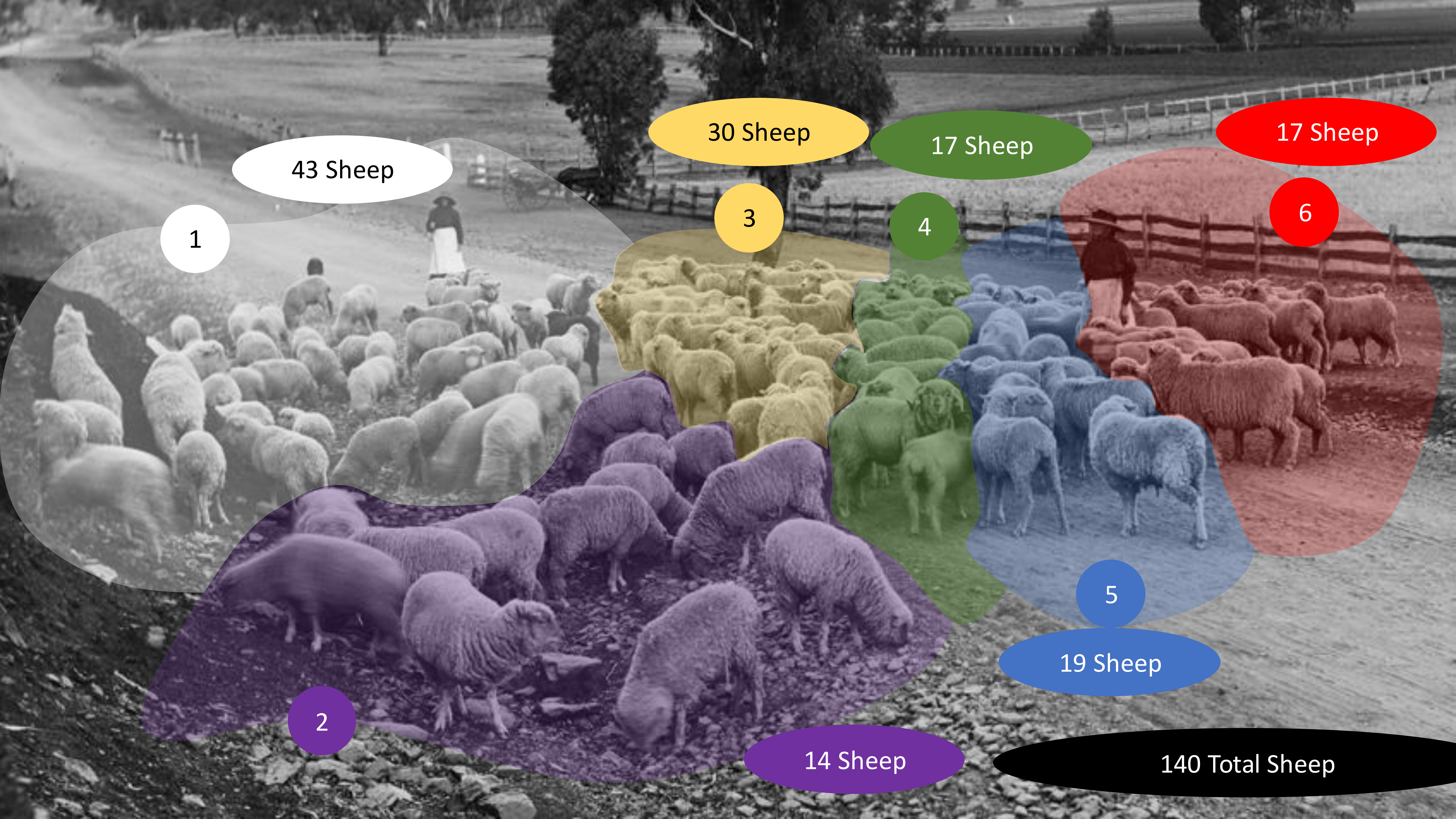


What is parallel computing?

- When many computations are carried out simultaneously



1
2
3
4
5
6


43 Sheep
1
30 Sheep
3
17 Sheep
4
17 Sheep
6
5
19 Sheep
2
14 Sheep
140 Total Sheep

What kinds of tasks benefit from HPC?

- Some tasks lend themselves easily to parallelisation
    - “embarrassingly parallel” / “perfectly parallel”
    - E.g., 3D computer graphics rendering – every pixel is independent but there are **a lot** of pixels
    - E.g., simulations which need to be run with many different parameters
    - E.g., stochastic simulations

- Some tasks are less well-suited to parallelisation
    - “inherently serial” – each step needs to run in series
    - Fluid dynamics (and solving other systems of differential equations)
    - Algorithms

What kinds of tasks benefit from HPC?

- Some tasks lend themselves easily to parallelisation
  - “embarrassingly parallel” / “perfectly parallel”
  - E.g., 3D computer graphics rendering – every pixel is independent but there are **a lot** of pixels
  - E.g., simulations which need to be run with many different parameters
  - E.g., stochastic simulations

- Some tasks are less well-suited to parallelisation
  - “inherently serial” – each step needs to run in series
  - Fluid dynamics (and solving other systems of differential equations)
  - Algorithms

GPU vs. many CPUs an analogy

What kinds of tasks benefit from HPC?

- Tasks where we need more memory than our local machine can handle
  - E.g. analysing or visualising very large data sets
- Tasks where we need to do a relatively simple operation for a long time or with many different parameters
  - Called “high-throughout computing” or HTC
- Tasks which can be automated
  - This allows us to free up our own machine to do other things
- There is always a trade-off as the process of getting jobs set up on the cluster is time-consuming in and of itself

What kinds of tasks benefit from HPC?

- Tasks where we need more memory than our local machine can handle
  - E.g. analysing or visualising very large data sets
- Tasks where we need to do a relatively simple operation for a long time or with many different parameters
  - Called “high-throughout computing” or HTC
- Tasks which can be automated
  - This allows us to free up our own machine to do other things
- There is always a trade-off as the process of getting jobs set up on the cluster is time-consuming in and of itself

HPC and using Imperial’s cluster

1. What is HPC and why is it useful?

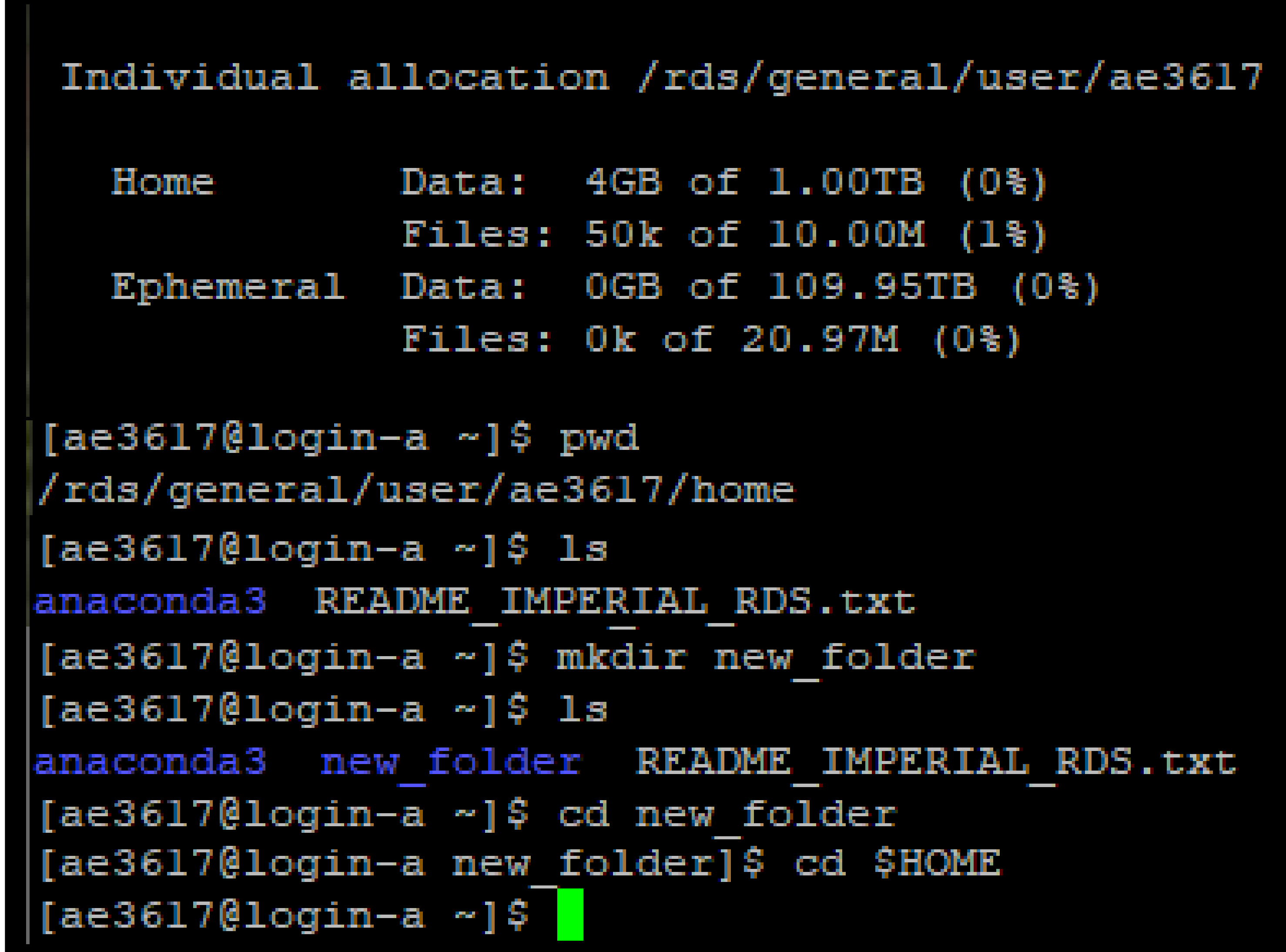
2. How to manage your files

3. How to run jobs on the cluster

File management

When logged in with SSH, you can navigate around your remote area and make new folders, move files and folders, and write new files in the command line, just as you can on your own computer from the command line.

However, you cannot transfer files between your local machine and the remote machine.



```
 Individual allocation /rds/general/user/ae3617

   Home        Data:  4GB of 1.00TB (0%)
               Files: 50k of 10.00M (1%)
   Ephemeral   Data:  0GB of 109.95TB (0%)
               Files: 0k of 20.97M (0%)

[ae3617@login-a ~]$ pwd
/rds/general/user/ae3617/home
[ae3617@login-a ~]$ ls
anaconda3  README_IMPERIAL_RDS.txt
[ae3617@login-a ~]$ mkdir new_folder
[ae3617@login-a ~]$ ls
anaconda3  new_folder  README_IMPERIAL_RDS.txt
[ae3617@login-a ~]$ cd new_folder
[ae3617@login-a new_folder]$ cd $HOME
[ae3617@login-a ~]$ 
```

File management

- SSH (secure shell) enables you to communicate with the HPC – you need to login using SSH in order to do things like set jobs running
- While logged in with SSH you can also manage your files on the remote machine (e.g., you can make new folders and files, move files around, and delete files)
- However, with SSH alone you cannot get access to your **local** file directory. To transfer files between your local machine and your remote area you will need to use SCP or SFTP.

File transfer

Using SFTP or SCP you can copy files between your local machine and the remote machine.

Use SFTP: from the directory of your code in a shell window type ...

- `sftp username@login.hpc.imperial.ac.uk`
- You will be asked for your standard cluster password
- `put filename.R` (will copy filename.R from your current **local** working directory to your current **remote** working directory)
- `get filename.R` (will copy filename.R from your current **remote** working directory to your current **local** working directory)
- `exit`

Or use SCP: from a shell window type ...

- `scp path/to/file.txt username@login.hpc.imperial.ac.uk:/home/username/`

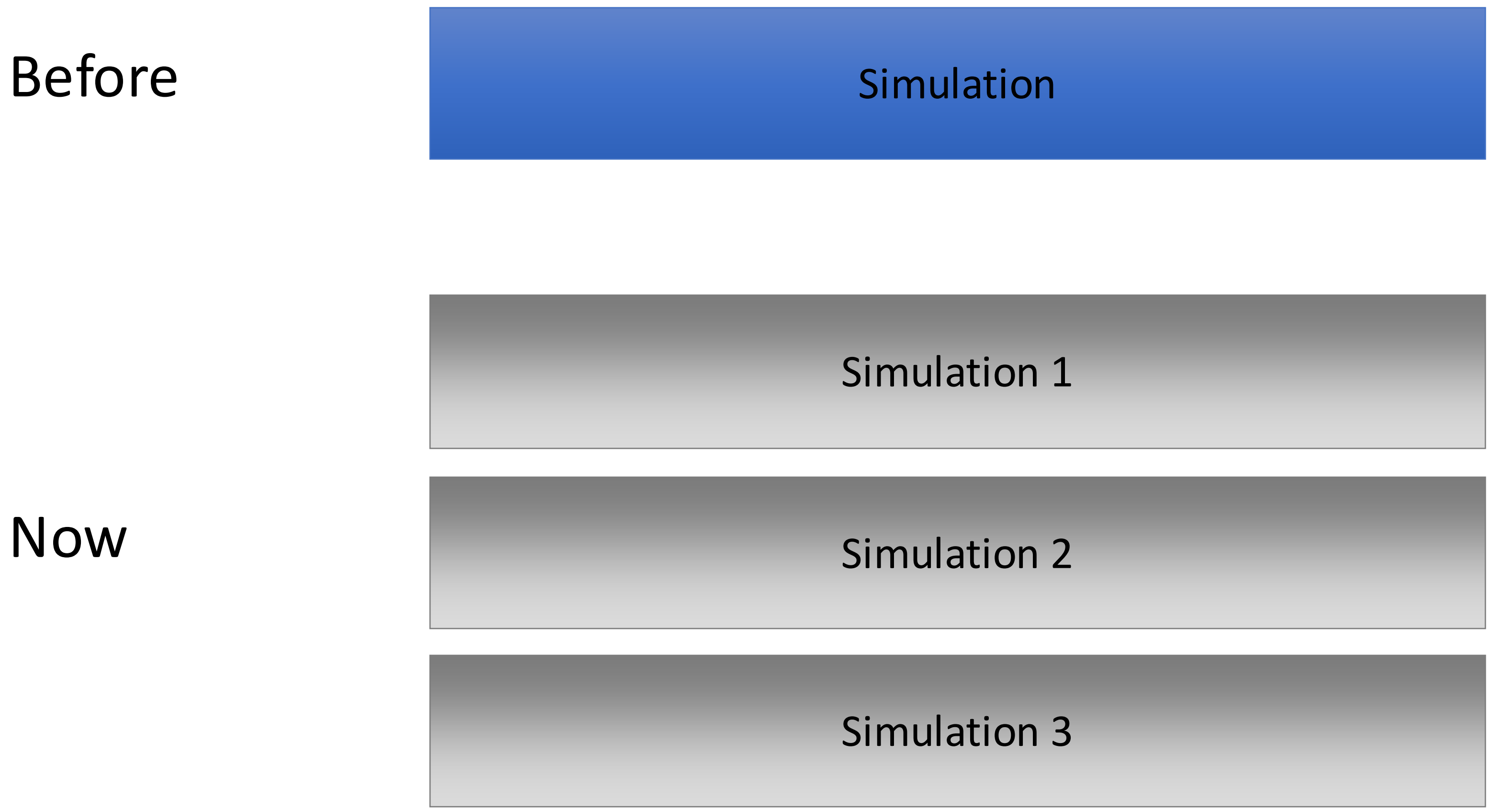
This works in a similar way to the cp command in standard directory management is run directly from the terminal (rather than logging in and then using "put" and "get").

HPC and using Imperial’s cluster

1. What is HPC and why is it useful?
2. How to manage your files
3. **How to run jobs on the cluster**

How do you parallelise your code?

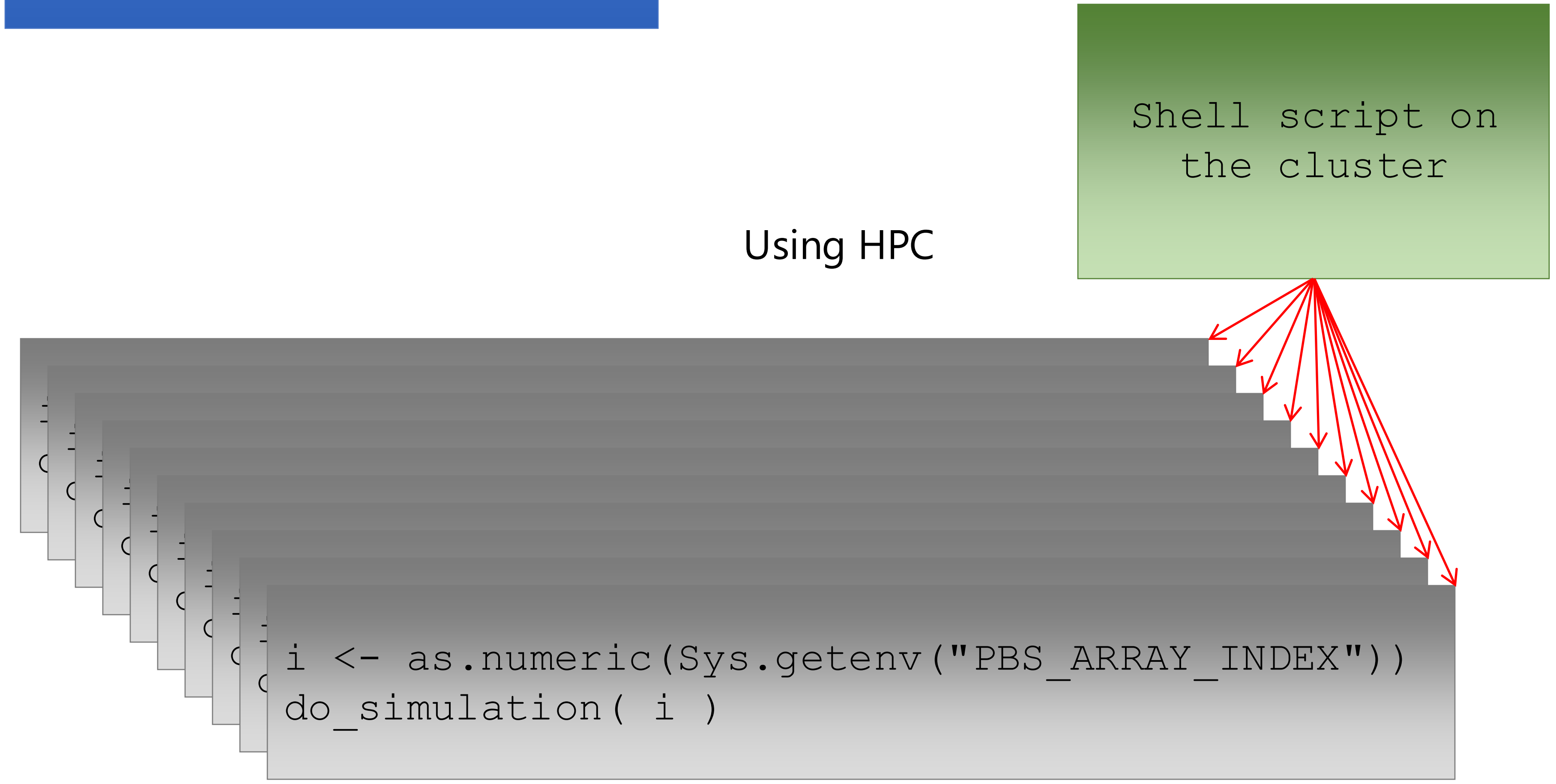
- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number



Using your PC

```
for ( i in 1 : 10 )
{
        do_simulation( i )
}
```

Using HPC

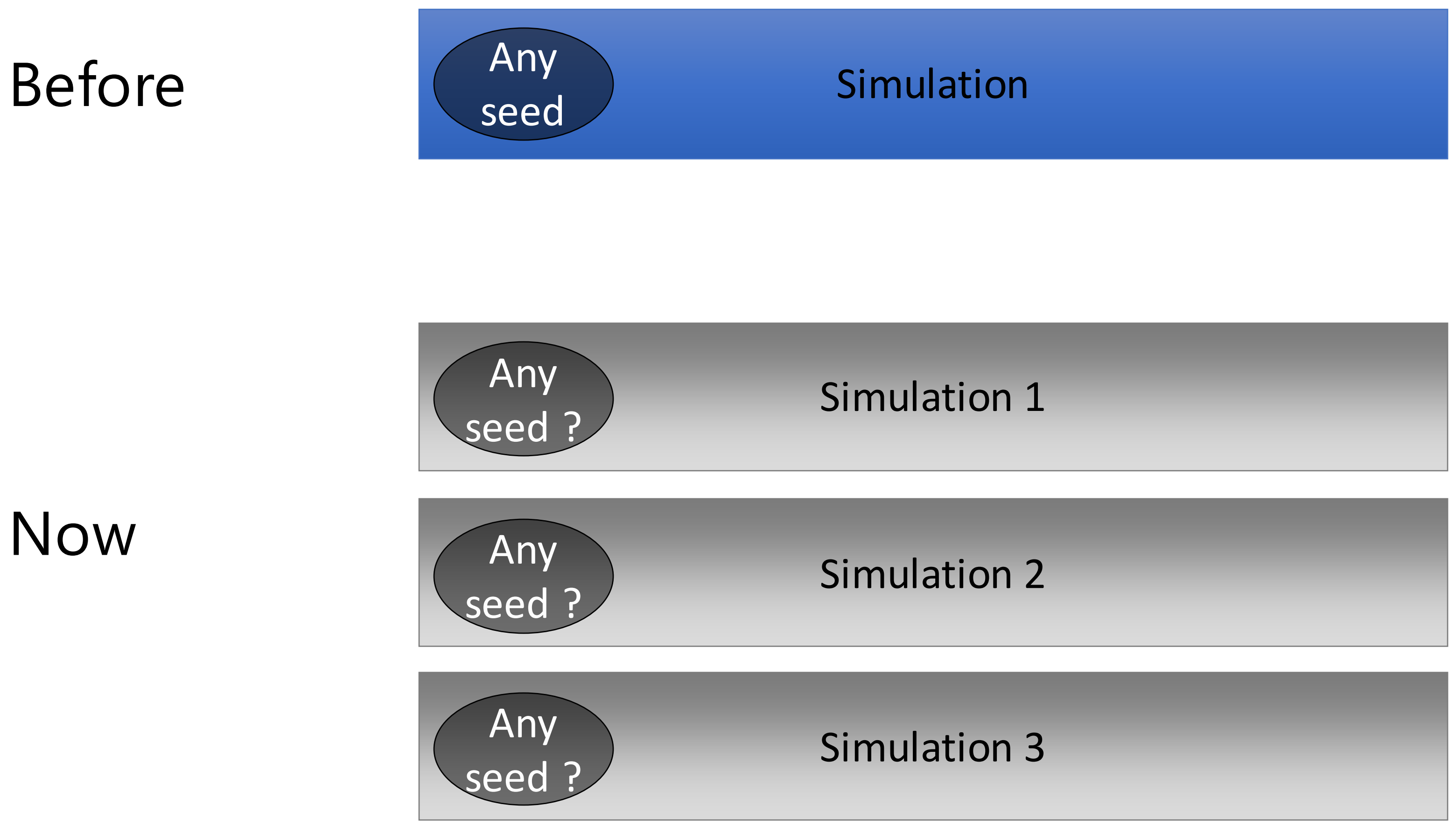


```
i <- as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))
do_simulation( i )
```

```
sample(c("heads","tails"),1)
```

How do you parallelise your code?

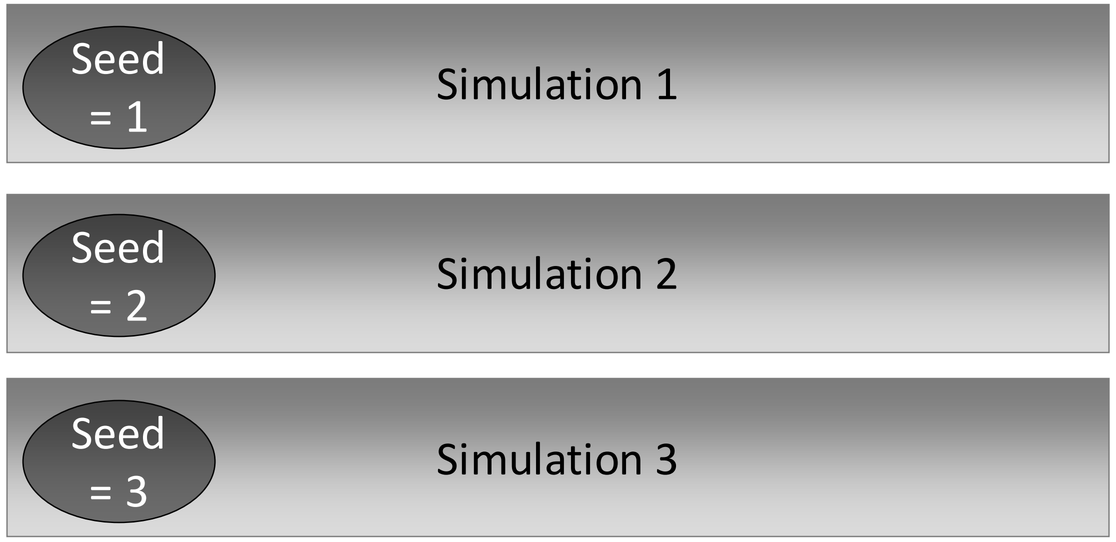
- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number

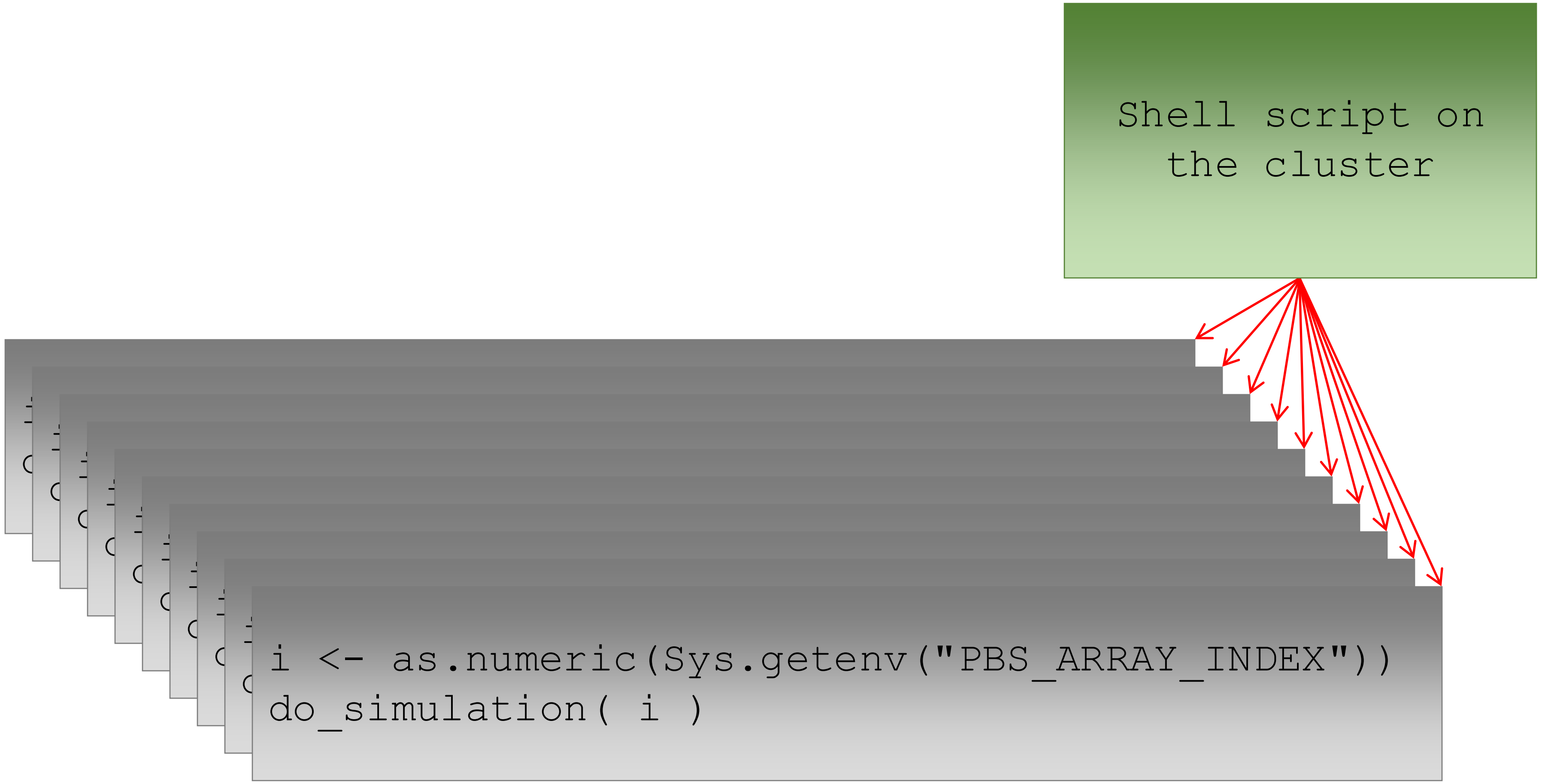


How do you parallelise your code?

- `as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))`
  - Should be used in your code to give a simulation number

- Pseudorandom numbers
  - Given a certain random number seed, you get the same sequence of random numbers every time
  - Each simulation should have a different seed

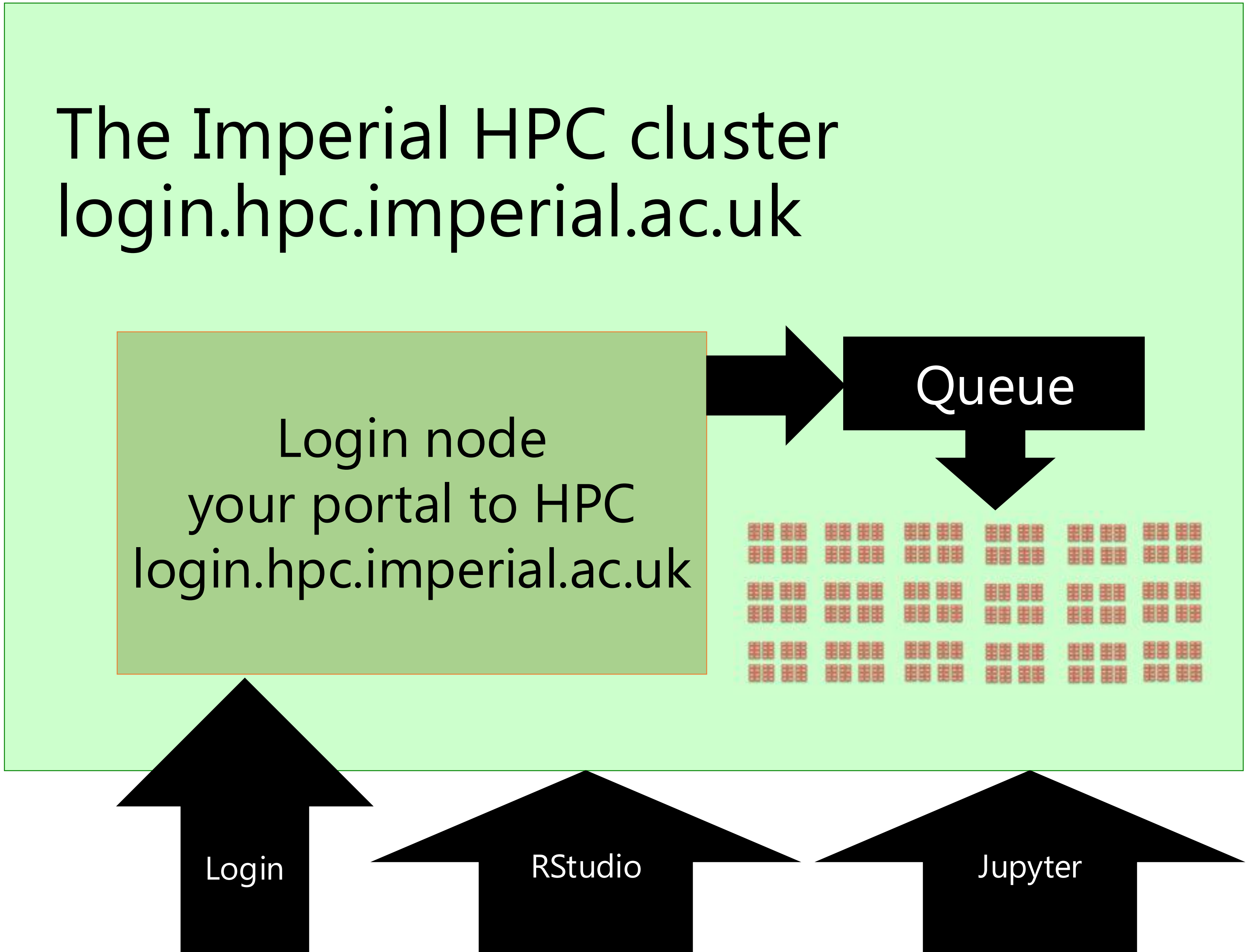




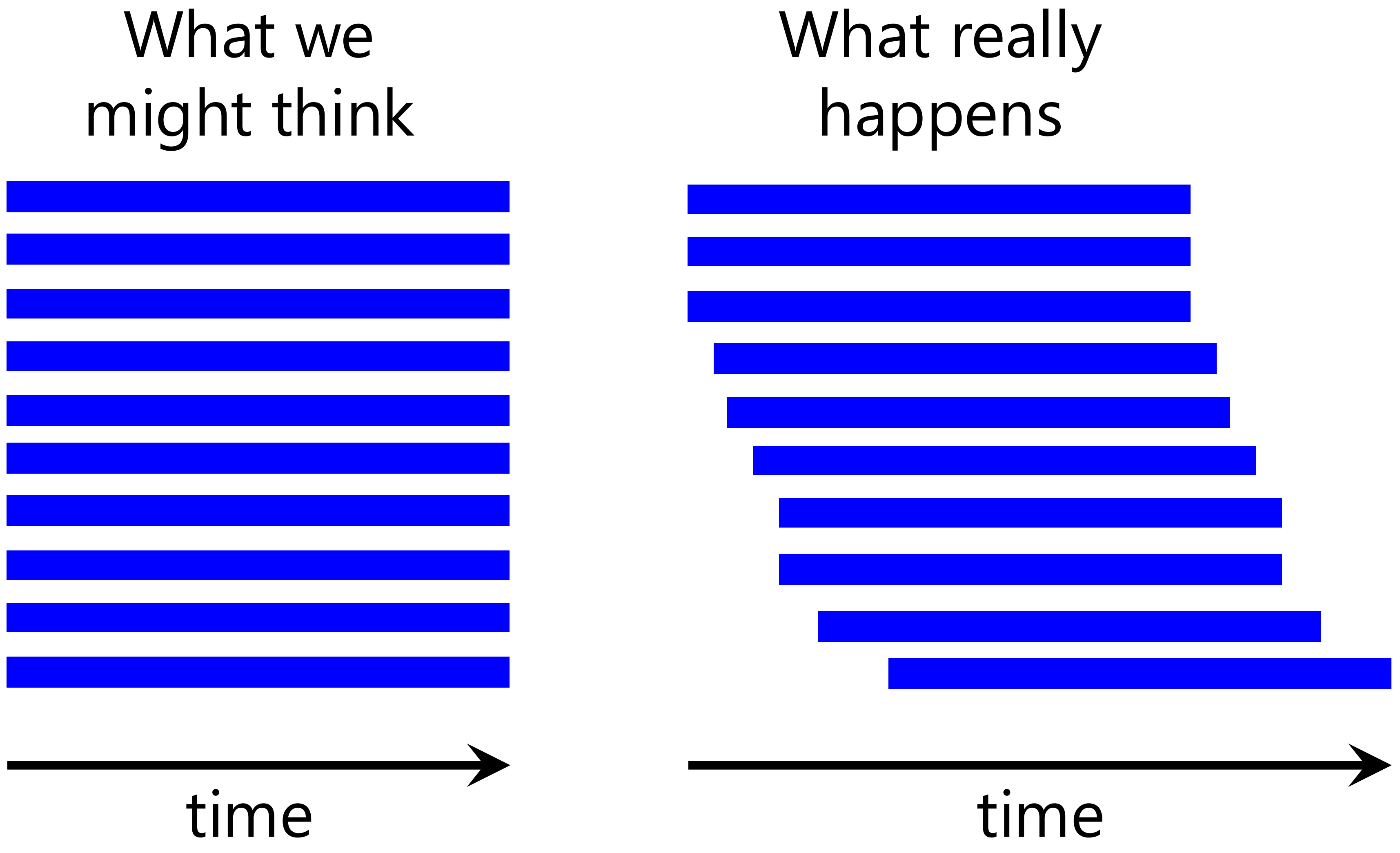
```
i <- as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))
do_simulation( i )
```

```
do_simulation <- function( i )
{
        # set random seed as i
        # select your simulation parameters
        # do your simulation
        # save your output in a file named …i…
        # include a timer
}
```

The Imperial HPC cluster login.hpc.imperial.ac.uk

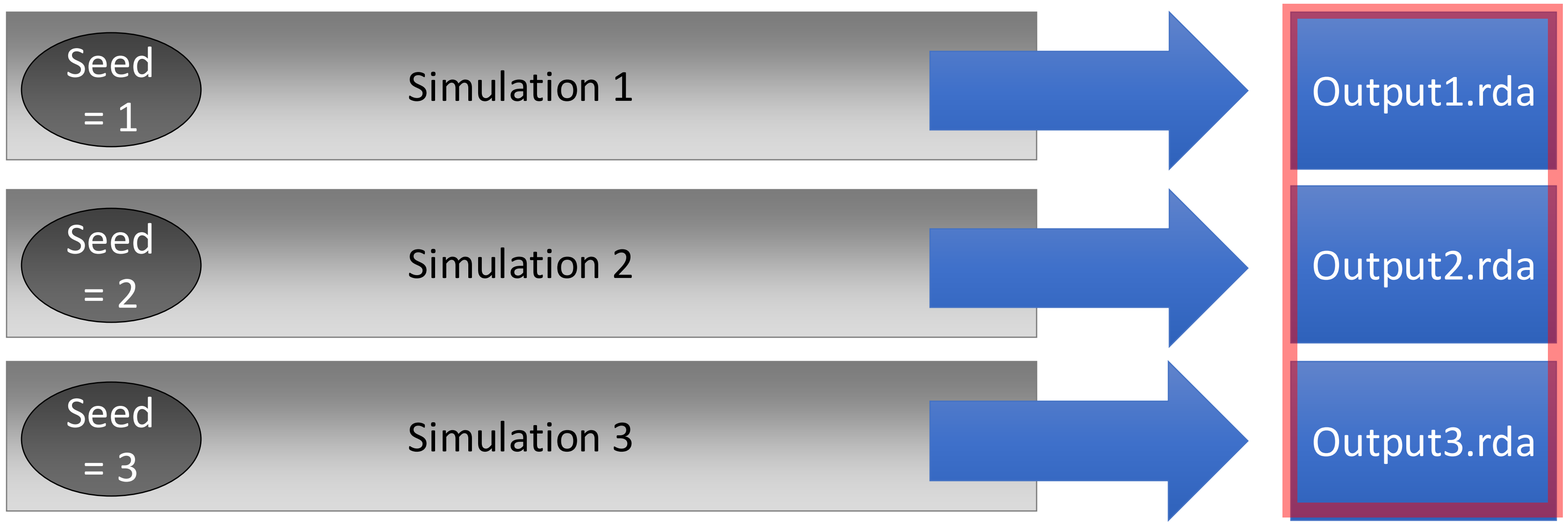


How jobs are run



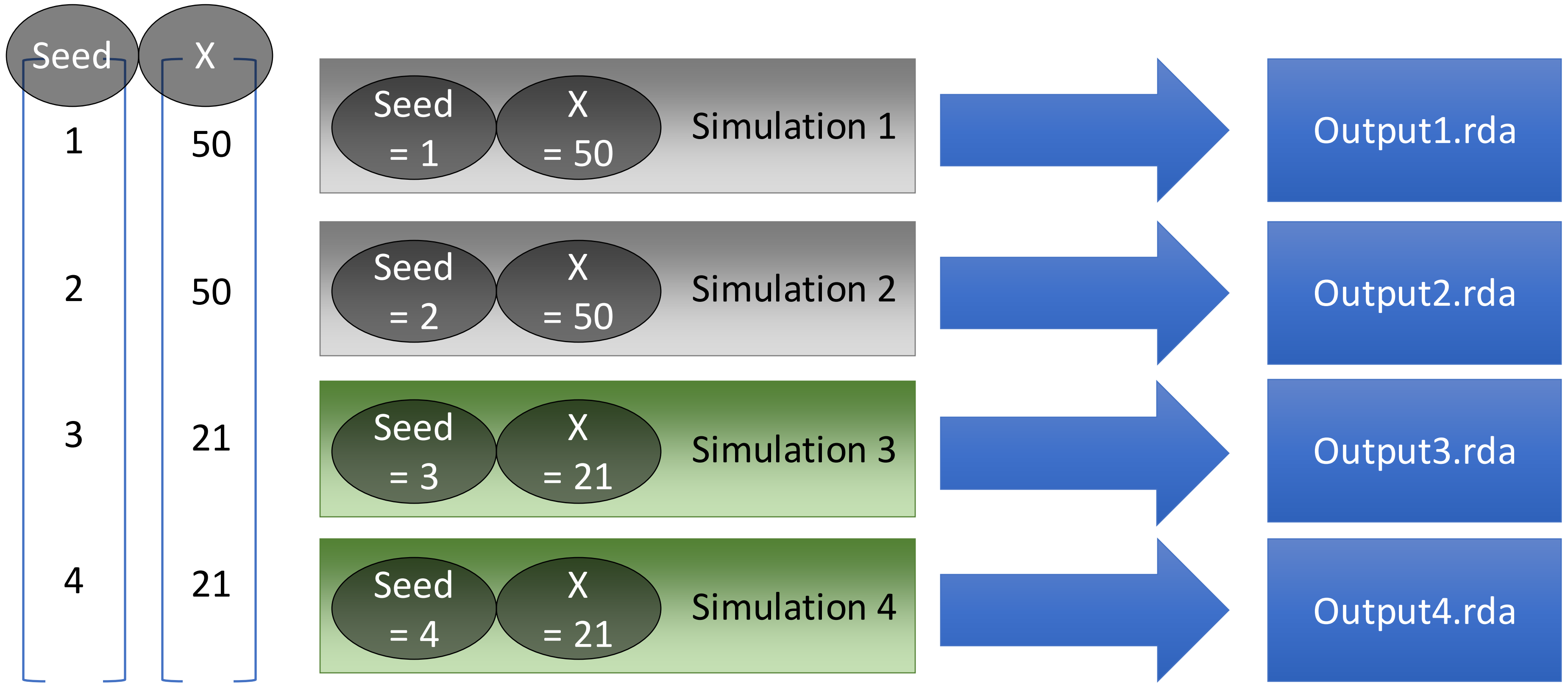
Handling memory

- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically



Handling memory

- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically



Handling memory

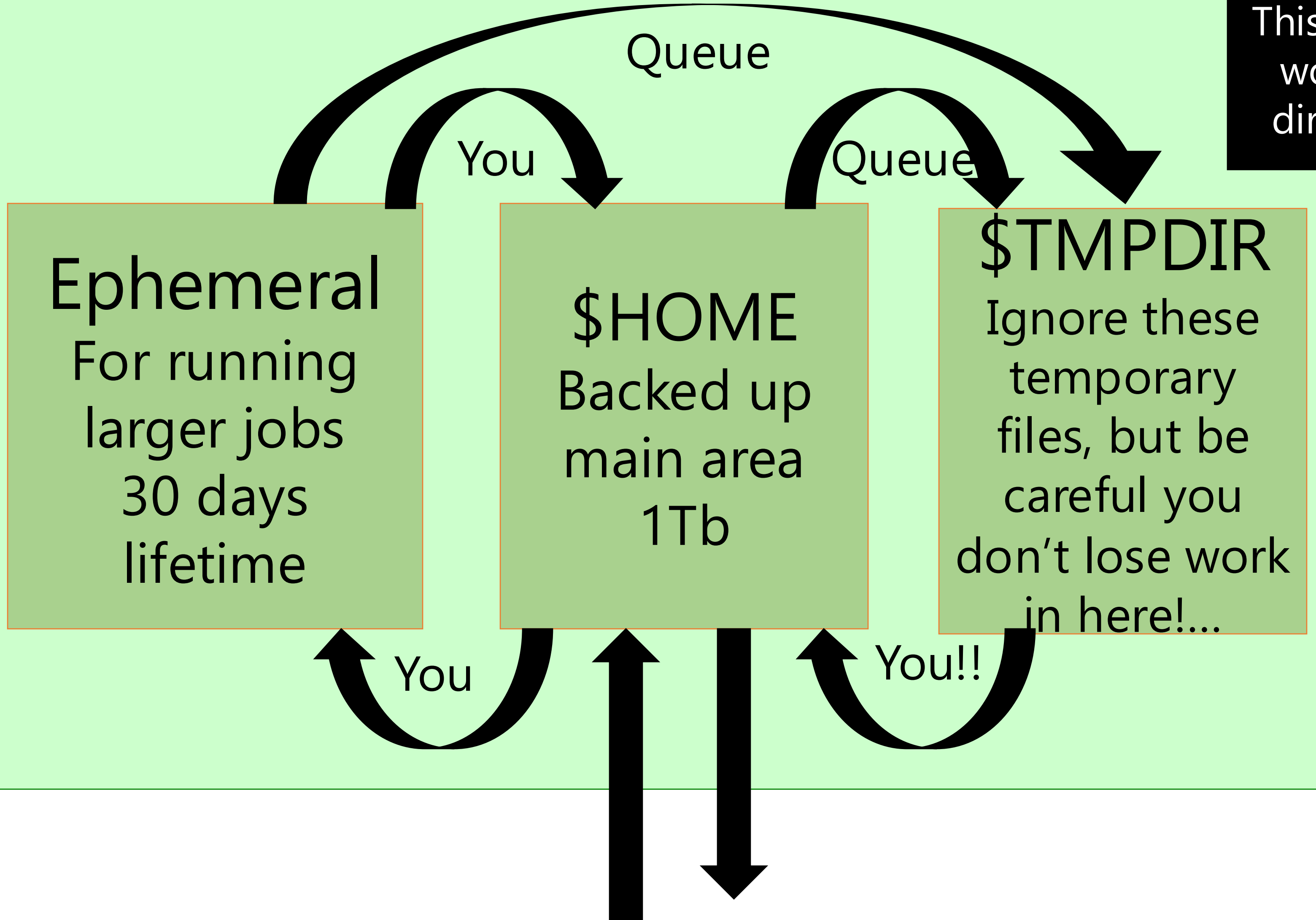
- Save your results in memory and then write to disk at the end
- Output your code to a series of files
- Write local code to read in your series of files automatically
- Build a timer into your code
- Test your code locally to know your memory and time requirements

Job sizing

Queue	Use Cases	Nodes per job	No. of cores per node (ncpus)	Mem per node (GB)	Walltime (hrs)
small24	Low core jobs 24h	1	1 - 16	1 - 128	0 - 24
small72	Low core jobs 72h	1	1 - 16	1 - 128	24 - 72
jupyter	Queue for JupyterHub jobs*	1	1, 4, 8	8, 32, 64	2, 4, 8
jupytergpu	Queue for JupyterHub GPU jobs*	1	4	32	8
medium24	Single-node jobs 24h	1	1 - 64	1 - 450	0 - 24
medium72	Single-node jobs 72h	1	1 - 64	1 - 450	24 - 72
large24	Whole node jobs 24h	1	1 - 128	1 - 920	0 - 24
large72	Whole node jobs 72h	1	1 - 128	1 - 920	24 - 72
~~largemem72~~	~~Large memory jobs~~	~~1~~	~~1 - 128~~	~~921 - 4000~~	~~0 - 72~~
gpu72	Main queue for gpu jobs*	1	1 - 64	1 - 920	0 - 72
capability24	Multi-node jobs 24h	2 - 4	1 - 64	1 - 450	0 - 24
capability48	Multi-node jobs 48h	2 - 4	1 - 64	1 - 450	24 - 48

Where your data is stored...

This is your working directory



Ephemeral
For running larger jobs
30 days lifetime

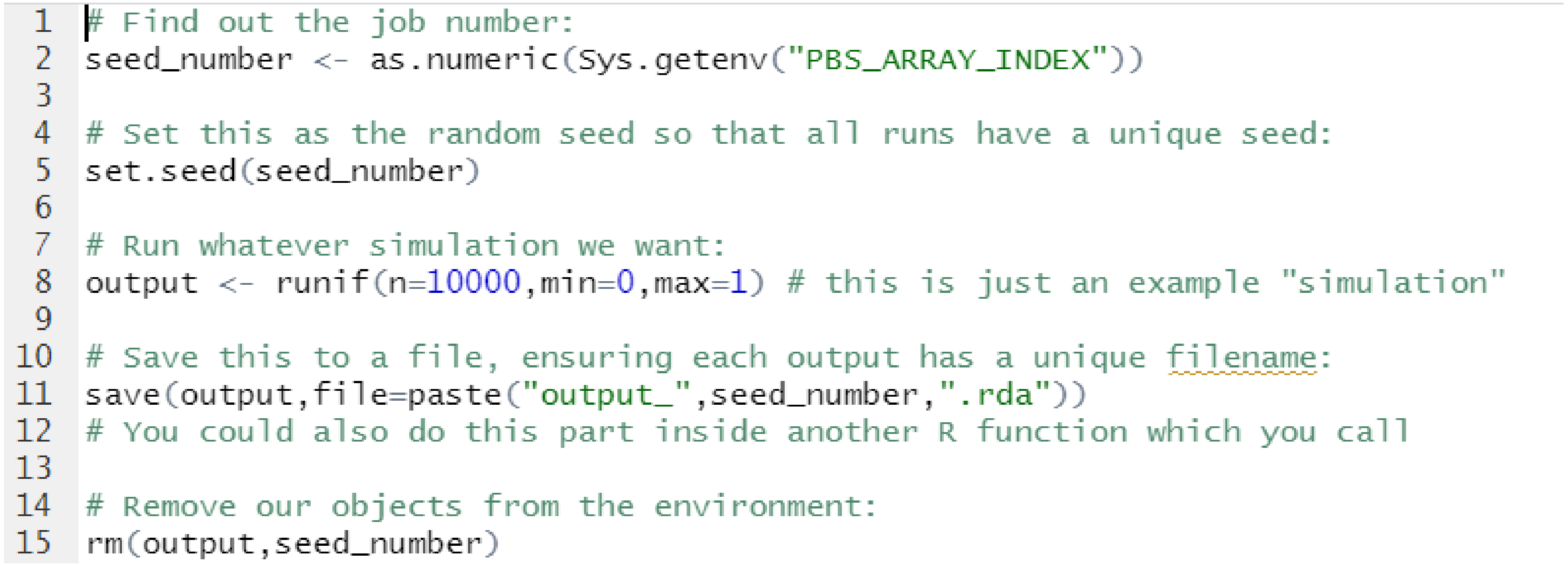
$HOME
Backed up main area
1Tb

$TMPDIR
Ignore these temporary files, but be careful you don't lose work in here!...

Step 1: A parallelised script

Write a script which runs your analysis using the parallelisation instructions.

e.g.:



```r
# Find out the job number:
seed_number <- as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))

# Set this as the random seed so that all runs have a unique seed:
set.seed(seed_number)

# Run whatever simulation we want:
output <- runif(n=10000,min=0,max=1) # this is just an example "simulation"

# Save this to a file, ensuring each output has a unique filename:
save(output,file=paste("output_",seed_number,".rda"))
# You could also do this part inside another R function which you call

# Remove our objects from the environment:
rm(output,seed_number)
```

Save this .R file.

Step 2: Copy parallelised script to the cluster

Copy your script into your desired part of the cluster.

e.g.:

```
scp HPC_script.R username@login.hpc.imperial.ac.uk:/home/username/
```

This would copy this script from the current local working directory into the home directory on the cluster.

Step 3: Write a shell file which will set up the job

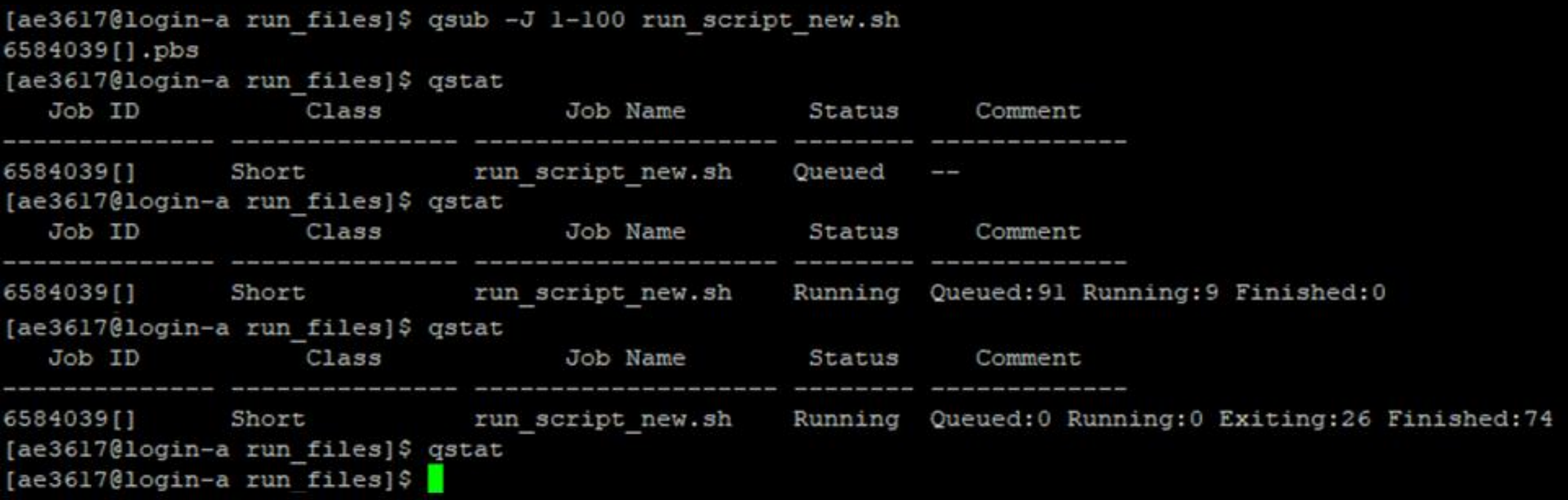
After logging on to the HPC, from the command line within the file you want to save your shell file in:

```
cat > run_script.sh
```

And then type out the script line-by-line to write it to file (something like this):

```
#!/bin/bash
#PBS -l walltime=12:00:00
#PBS -l select=1:ncpus=1:mem=1gb
module load anaconda3/personal
echo "R is about to run"
R --vanilla < $HOME/run_files/HPC_script.R
mv datafilename* $HOME
echo "R has finished running"
# this is a comment at the end of the file
```

Step 4: Set the job running



```
[ae3617@login-a run_files]$ qsub -J 1-100 run_script_new.sh
6584039[].pbs
[ae3617@login-a run_files]$ qstat
   Job ID          Class             Job Name          Status     Comment
-------------- -------------- ------------------- -------- -------------
6584039[]      Short          run_script_new.sh    Queued   --
[ae3617@login-a run_files]$ qstat
   Job ID          Class             Job Name          Status     Comment
-------------- -------------- ------------------- -------- -------------
6584039[]      Short          run_script_new.sh    Running  Queued:91 Running:9 Finished:0
[ae3617@login-a run_files]$ qstat
   Job ID          Class             Job Name          Status     Comment
-------------- -------------- ------------------- -------- -------------
6584039[]      Short          run_script_new.sh    Running  Queued:0 Running:0 Exiting:26 Finished:74
[ae3617@login-a run_files]$ qstat
[ae3617@login-a run_files]$
```

Step 5: Check that all is well

- Wait 5-10 minutes then check that nothing has gone wrong
- `qstat` (is your job running still?)
- `ls` (are output files as expected?)
- `cat filename.sh.ejob-id.index`

  (are error files empty?)
- `cat filename.sh.ojob-id.index`

  (are standard output files as expected?)
- `qstat` (is your job running still?)
- `exit` (you’re done for now; come back later)

Step 6: Get your results back

- `qstat` (is your job running still?)
- `ls` (output files as expected?)
- `cat output filename` (contents as expected?)
- `cat filename.sh.ejob-id.index`

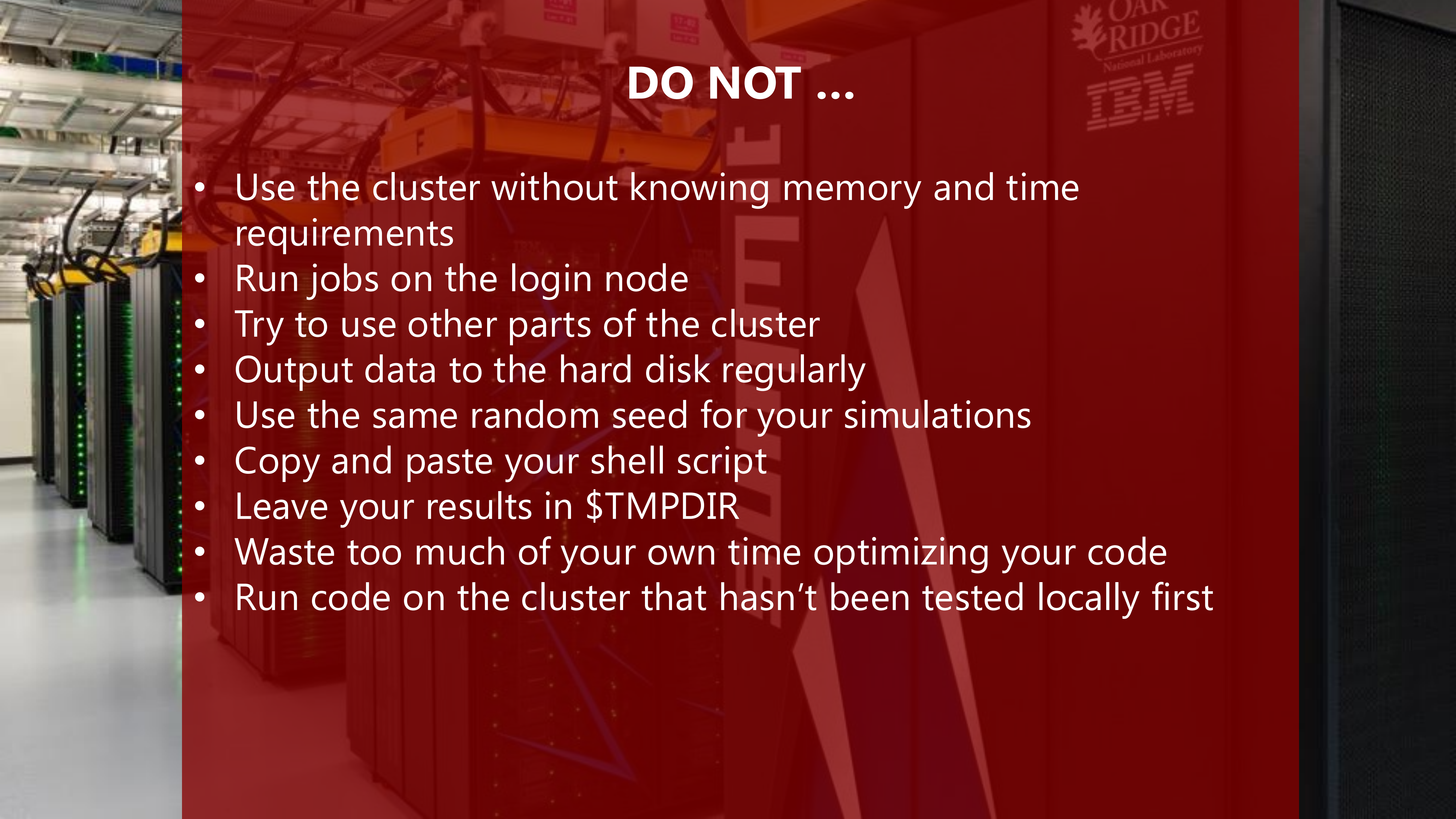
  (error files empty?)
- `cat filename.sh.ojob-id.index`

  (standard output files as expected?)
- `tar czvf filename.tgz *`
- `mv filename.tgz $HOME`
- `exit`

Step 6: Get your results back (continued)

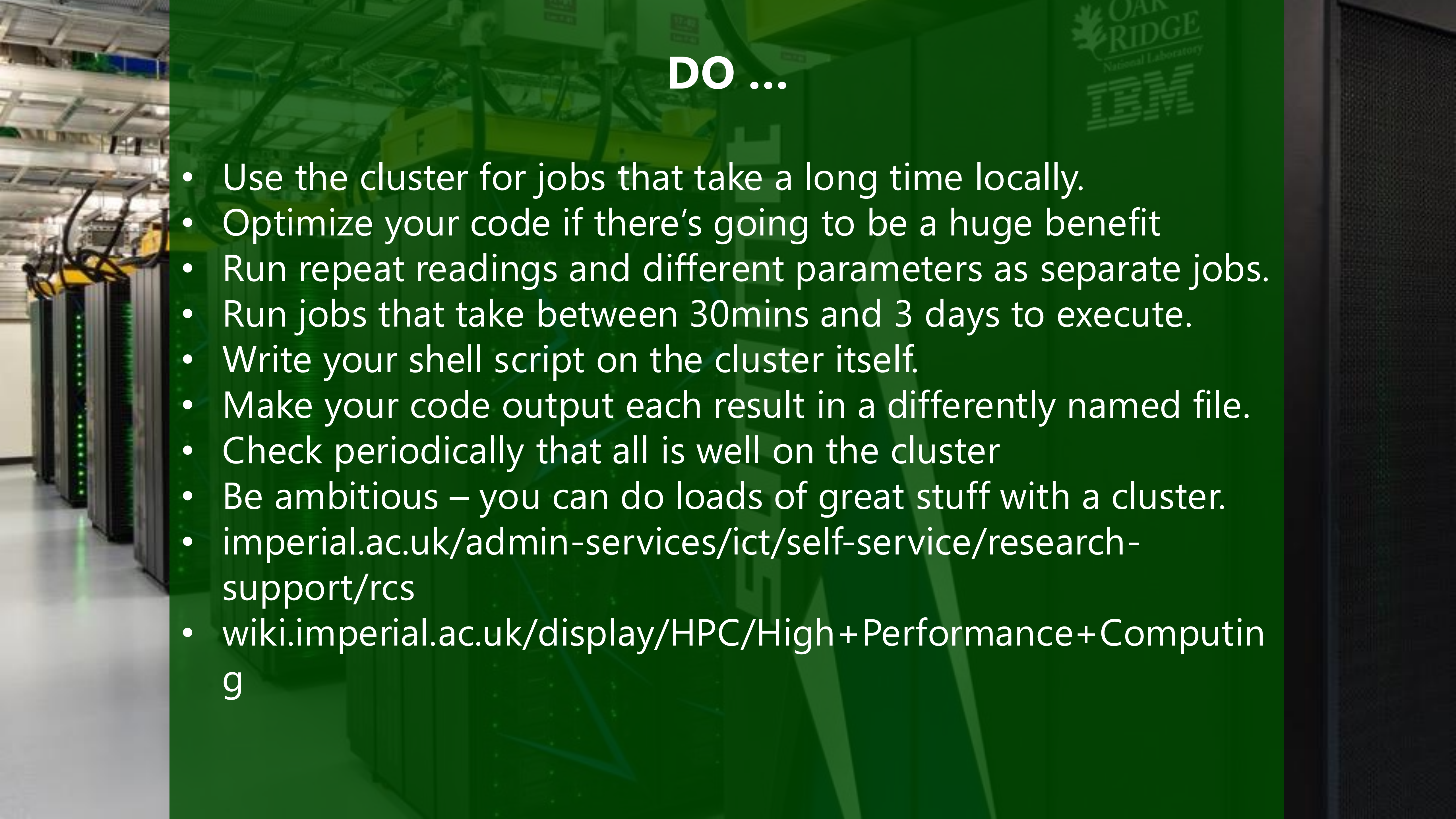
Use SFTP: from a new directory on your own computer of where you want the results to be.   Open a shell and type …

  - `sftp username@login.hpc.imperial.ac.uk`
  - You will be asked for your standard cluster password
  - `get filename.tgz`
  - `exit`
- Your results are now all on your own computer
  - `tar xzvf filename.tgz`
- Your results are now complete uncompressed and ready for use. Now you need to write some R code to read in and analyse all those files.



DO NOT …

- Use the cluster without knowing memory and time requirements
- Run jobs on the login node
- Try to use other parts of the cluster
- Output data to the hard disk regularly
- Use the same random seed for your simulations
- Copy and paste your shell script
- Leave your results in $TMPDIR
- Waste too much of your own time optimizing your code
- Run code on the cluster that hasn’t been tested locally first



DO …

- Use the cluster for jobs that take a long time locally.
- Optimize your code if there’s going to be a huge benefit
- Run repeat readings and different parameters as separate jobs.
- Run jobs that take between 30mins and 3 days to execute.
- Write your shell script on the cluster itself.
- Make your code output each result in a differently named file.
- Check periodically that all is well on the cluster
- Be ambitious – you can do loads of great stuff with a cluster.
- imperial.ac.uk/admin-services/ict/self-service/research-support/rcs
- wiki.imperial.ac.uk/display/HPC/High+Performance+Computing