An underwater photograph of a coral reef. The foreground is filled with various types of coral, including branching and table corals, in shades of yellow, orange, and green. The background shows a deep blue water column with some fish swimming and more coral structures visible in the distance. A semi-transparent purple gradient box is overlaid in the center, containing the title text.

High Performance Computational Methods for Ecology and Evolution

Lauren Attfield

Course objectives

By the end of this course you should be able to:

- Describe different types of high performance computing
- Demonstrate good programming practice
- Identify problems which could benefit from parallel computing
- Run code in parallel on a cluster
- Optimise your code
- Process large amounts of data produced by the cluster

Course objectives

By the end of this course you should be able to:

- ➔ • Describe different types of high performance computing
- Demonstrate good programming practice
- ➔ • Identify problems which could benefit from parallel computing
- ➔ • Run code in parallel on a cluster
- Optimise your code
- Process large amounts of data produced by the cluster

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
4. Demonstration

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

4. Demonstration



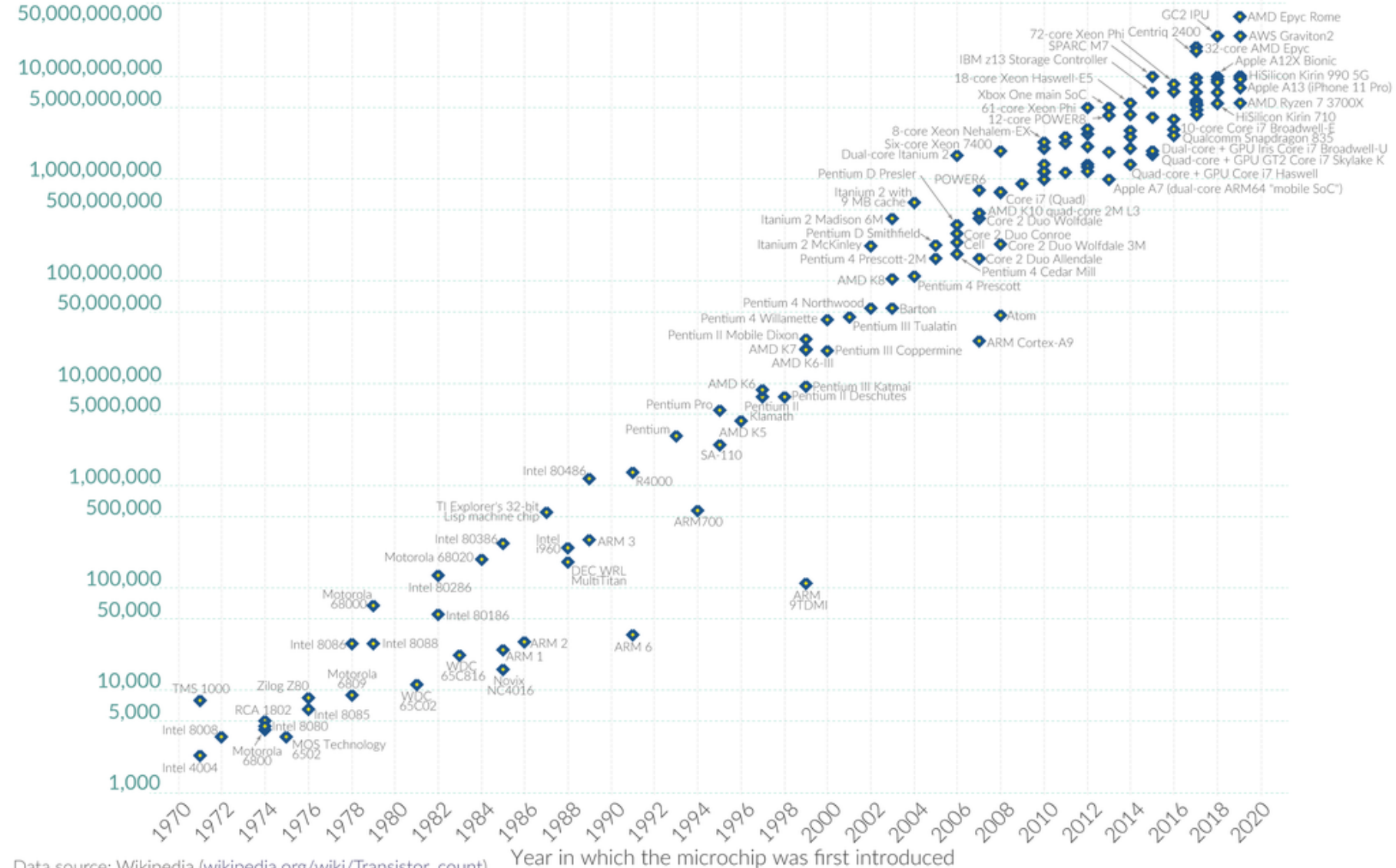
HPC Computer Room

Moore's Law: The number of transistors on microchips doubles every two years

Our World
in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world's largest problems.

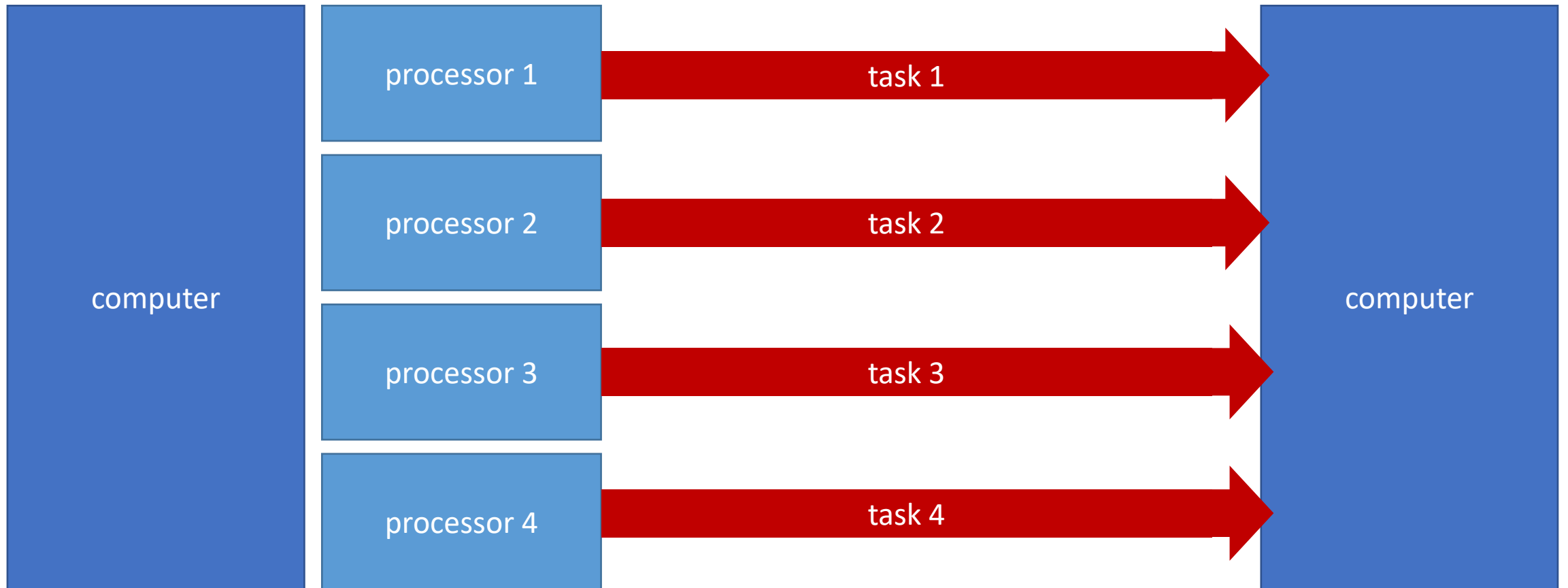
Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

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

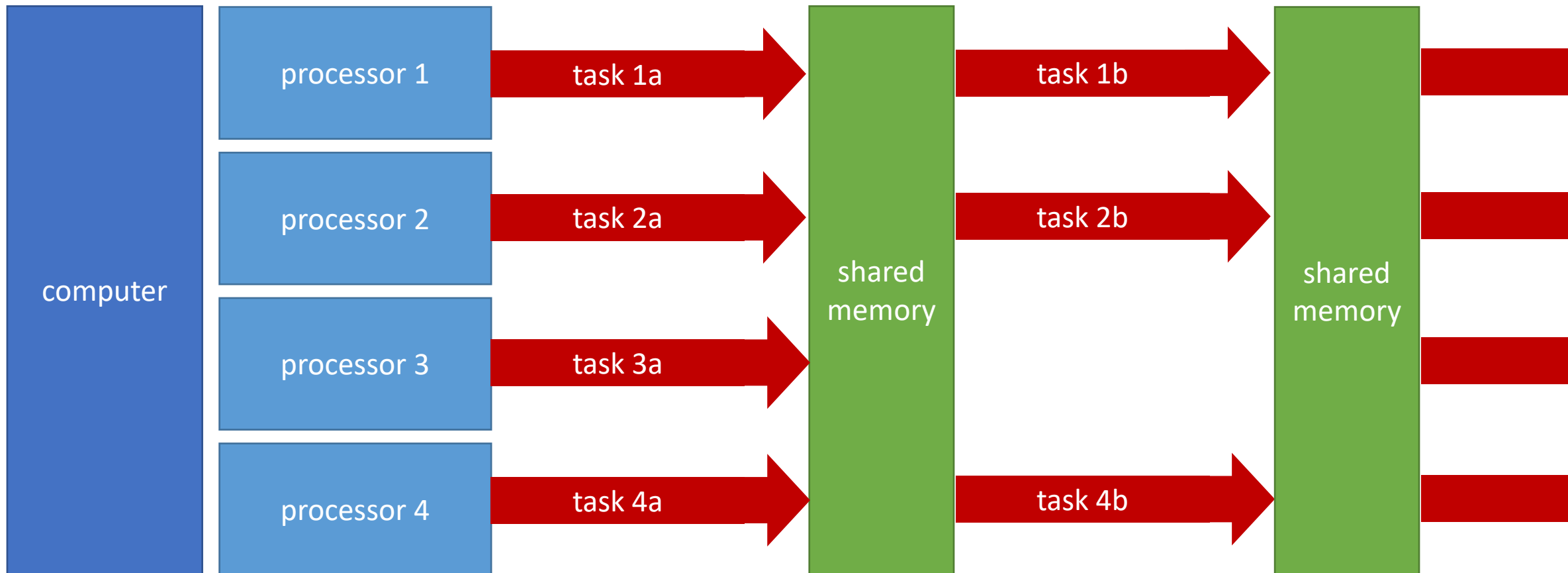
What is parallel computing?

- When many computations are carried out simultaneously



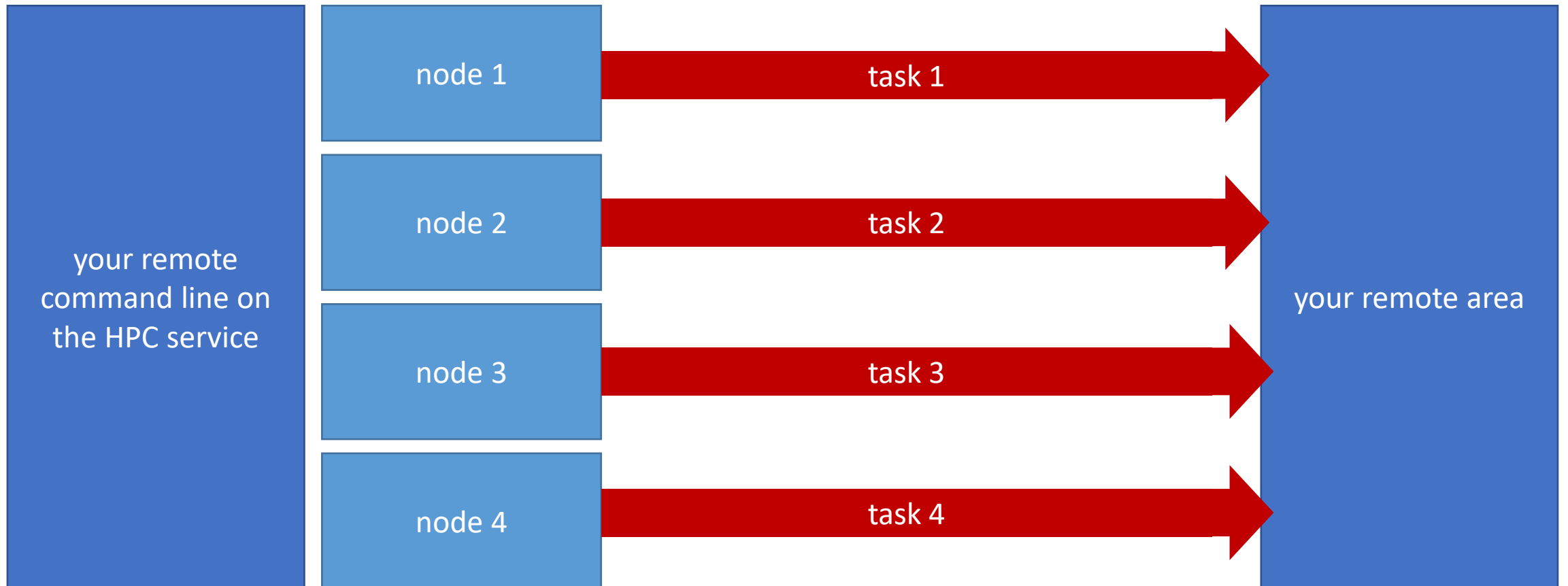
What is parallel computing?

- When many computations are carried out simultaneously



What is parallel computing?

- When many computations are carried out simultaneously



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

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

4. Demonstration

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

4. Demonstration

How do you parallelise your code?

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

Before



Simulation

Now



Simulation 1

Simulation 2

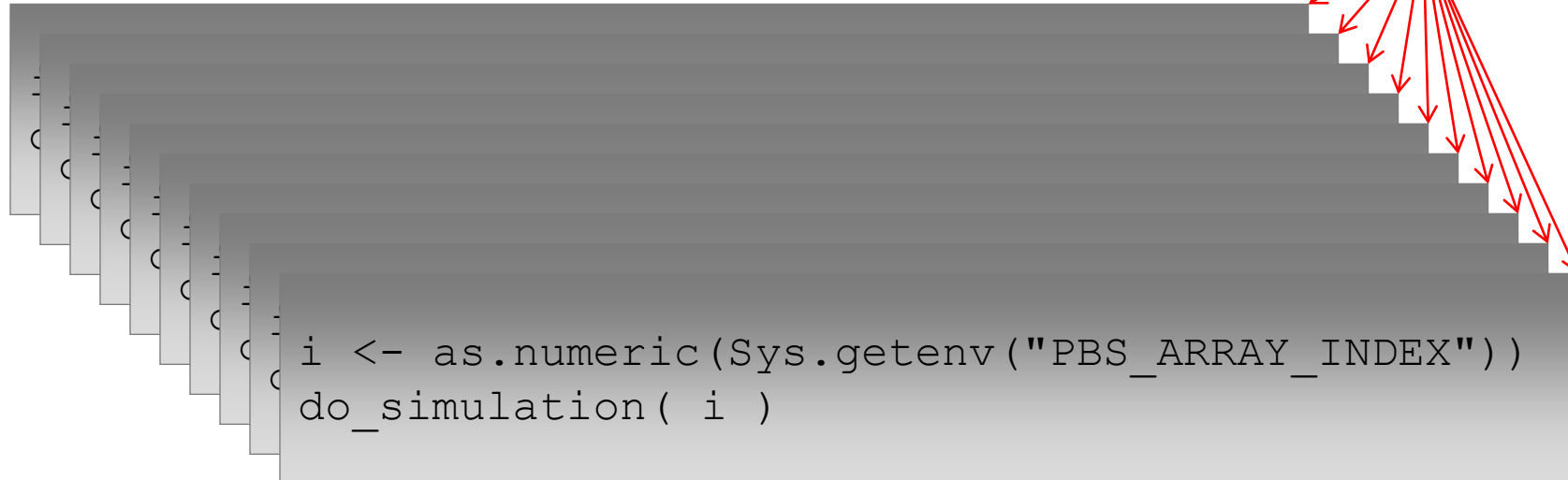
Simulation 3

Using your PC

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

Using HPC

Shell script on
the cluster



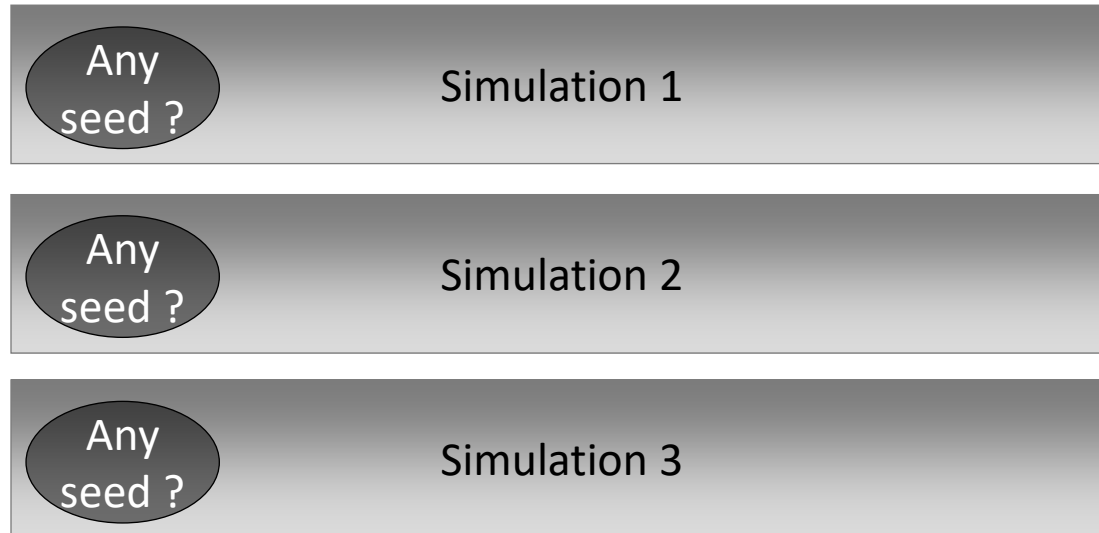
How do you parallelise your code?

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

Before

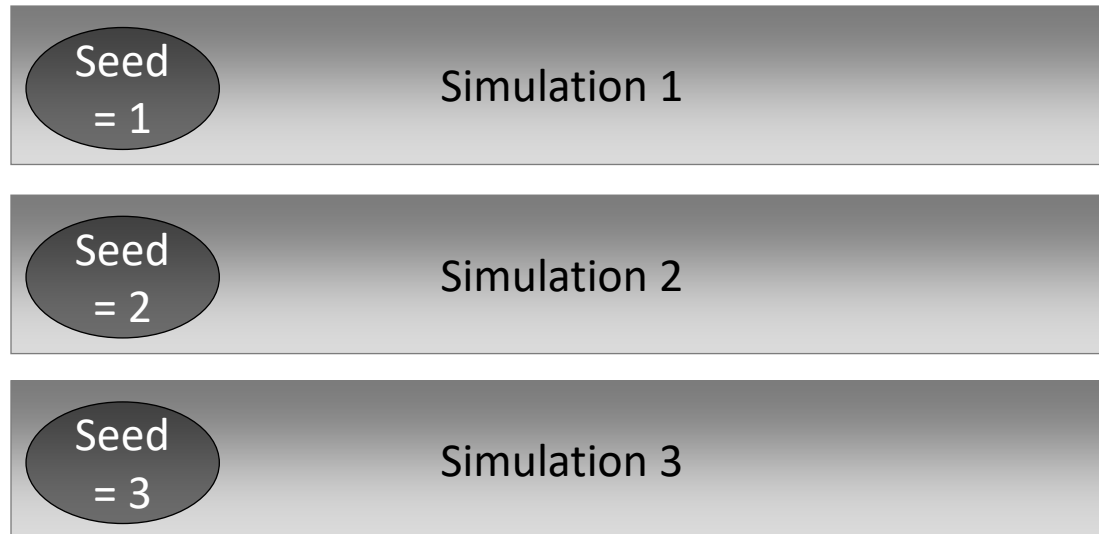


Now



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



Shell script on
the cluster



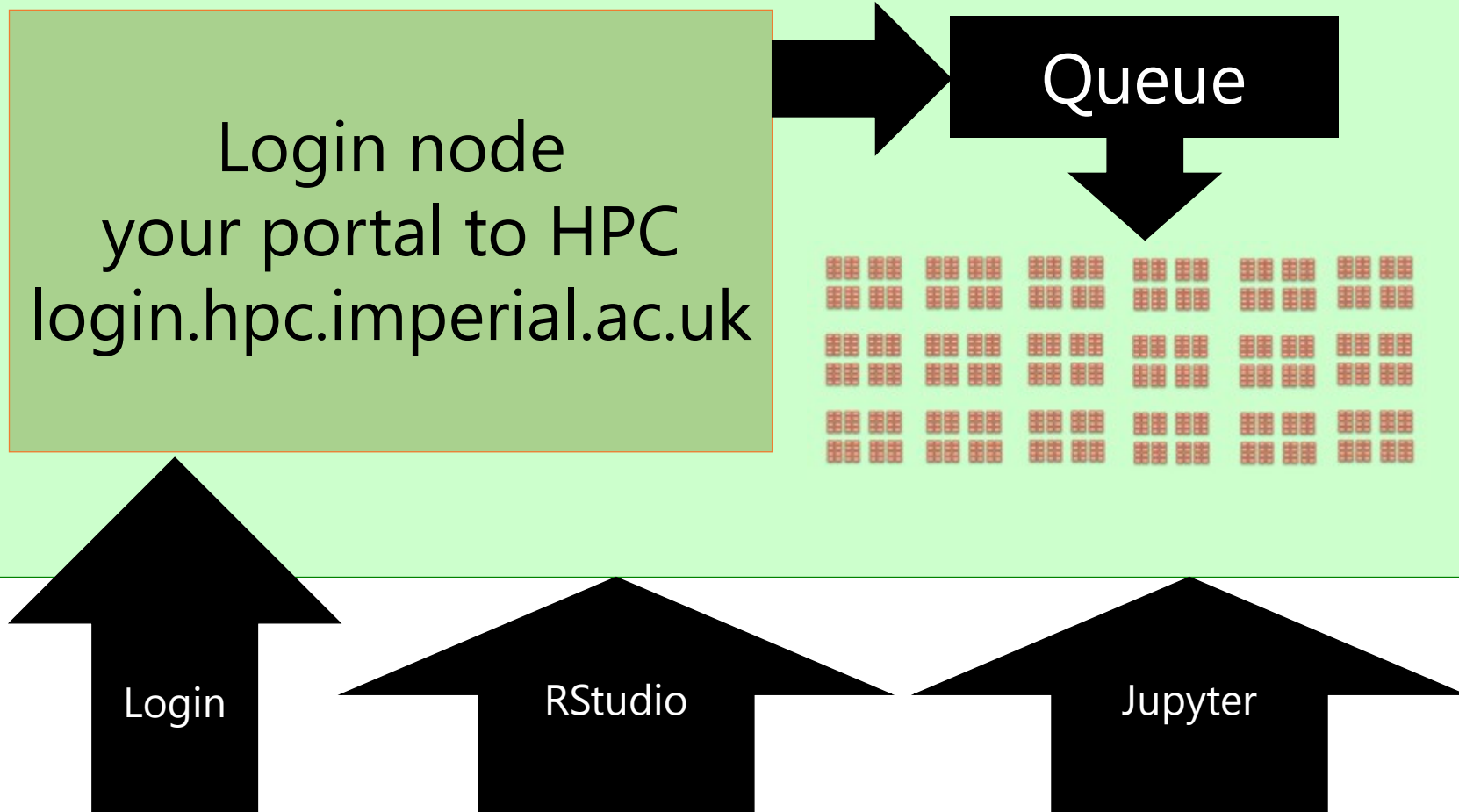
The diagram illustrates a distributed computing setup. At the top right, a green box labeled "Shell script on the cluster" has ten red arrows pointing downwards to a series of ten overlapping gray boxes. Each gray box represents a node in a cluster and contains a shell script snippet. The first part of the script in each box is:

```
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  
}
```

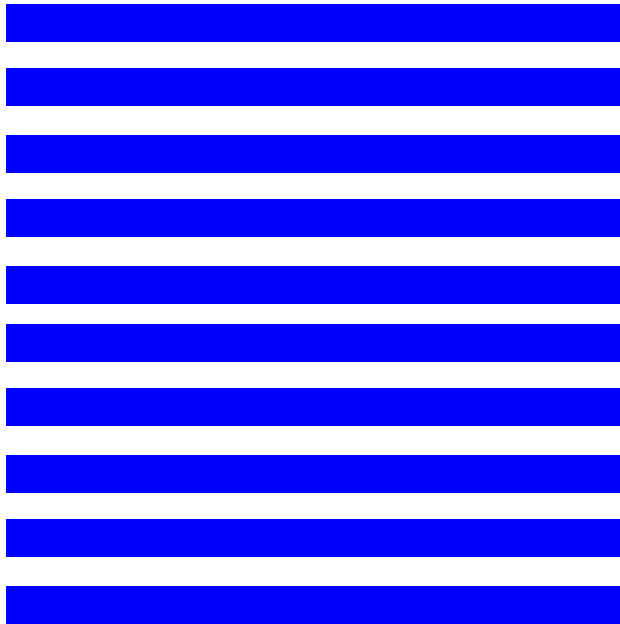
The Imperial HPC cluster

login.hpc.imperial.ac.uk



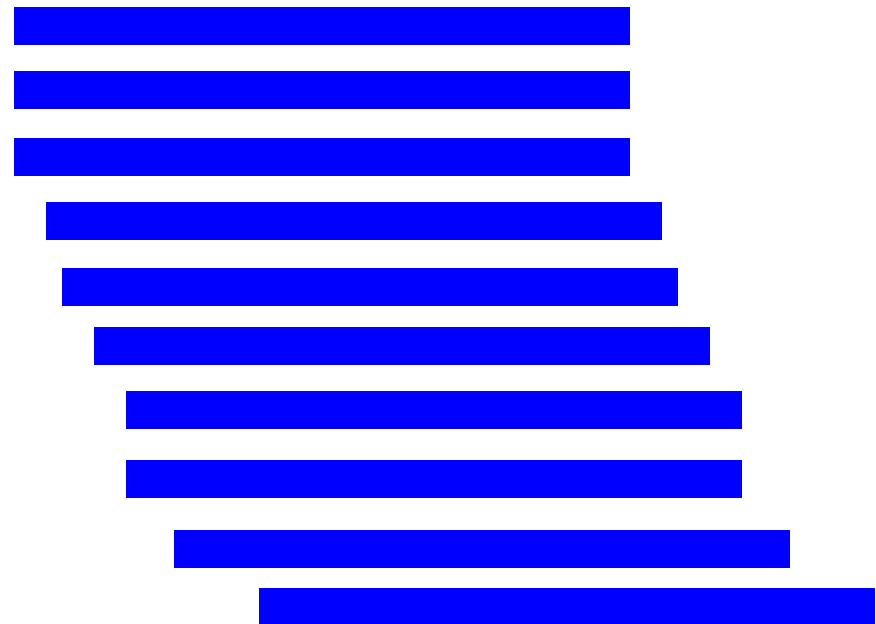
How jobs are run

What we
might think



time

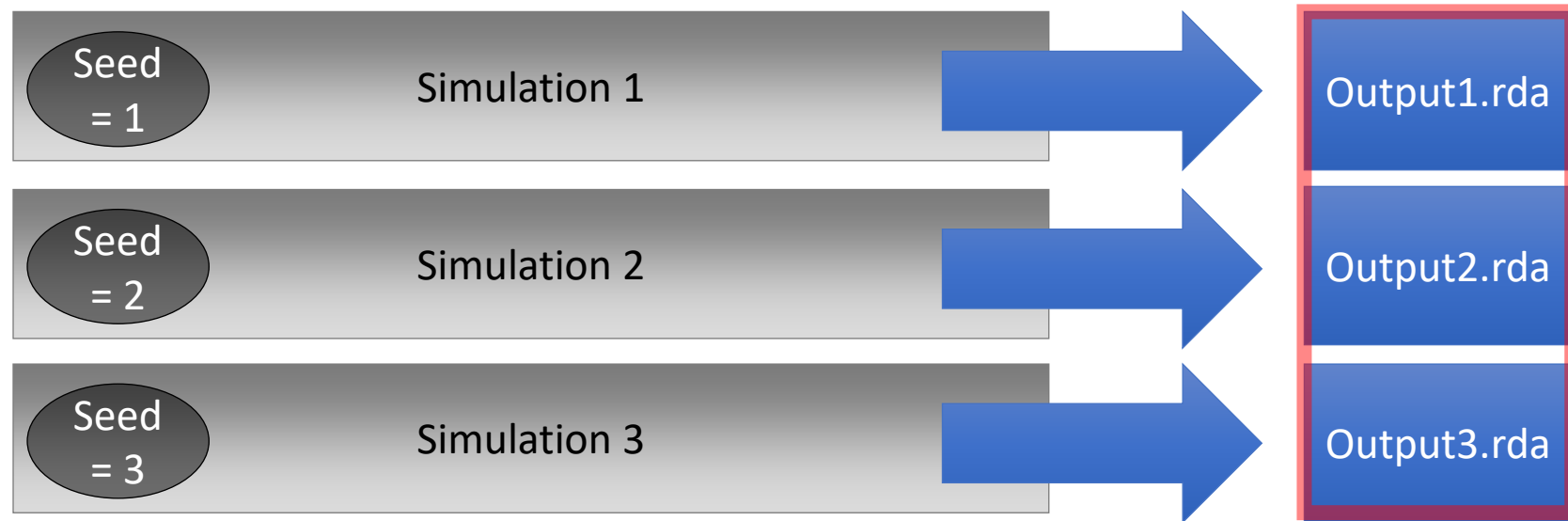
What really
happens



time

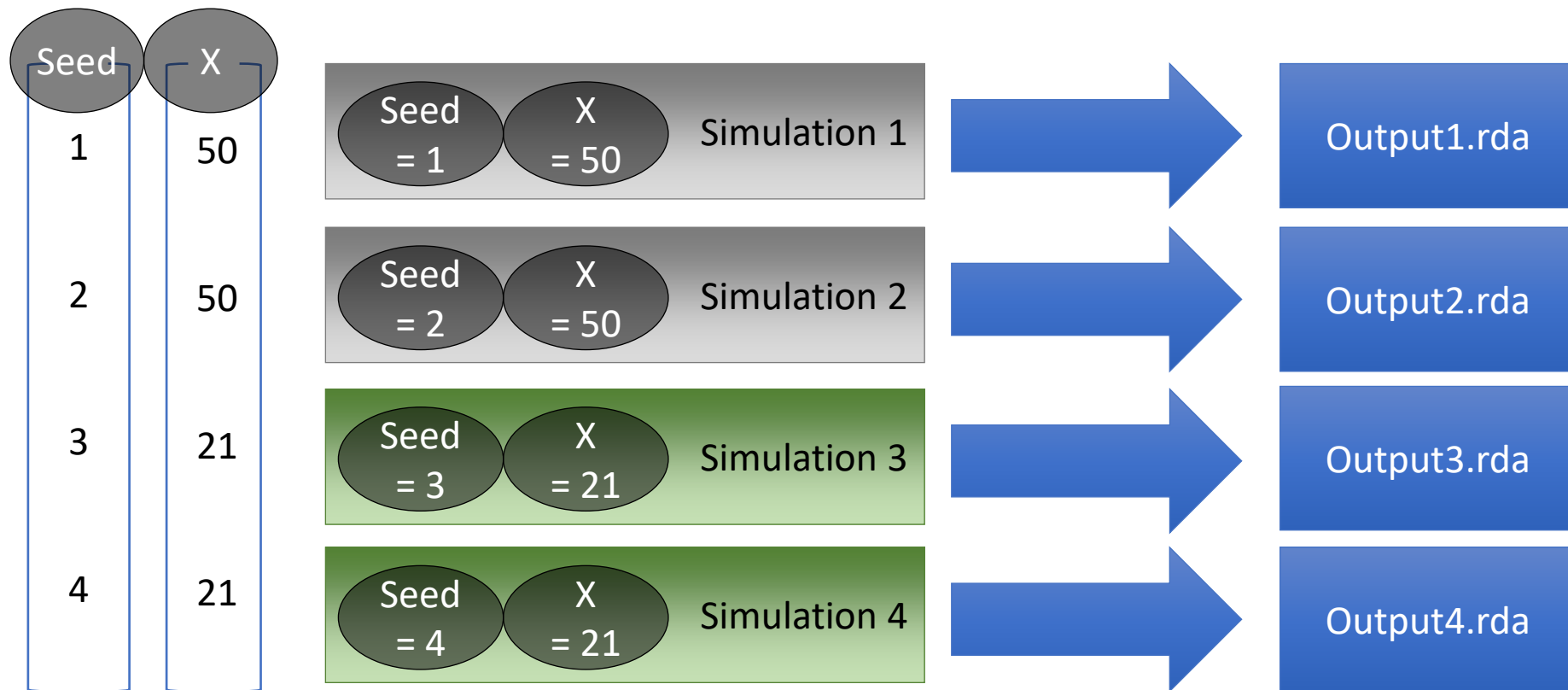
Handling memory

- 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

Job class	Number of nodes N	ncpus/node	Max mem/node	Max walltime/hr	Max number of running jobs per user
throughput	1	1-8	96GB	up to 72hr	unlimited for jobs <=24hr in length
general	1 - 16	32	62GB or 124GB	up to 72hr	unlimited for jobs <=24hr in length
singlenode	1	48	124GB	up to 24hr	10
multinode	3 - 16	12	46GB	up to 48hr	unlimited
debug	1	1-8	96GB	up to 30 mins	1
large memory	1	12 or 24	190 or 380GB	48 hr	unlimited
GPU	1-2	1-8	1 - 32GB	48hr 72hr (P1000 only)	8
long	1	1-8	96GB	72 - 1000 hr	1

disk at the end

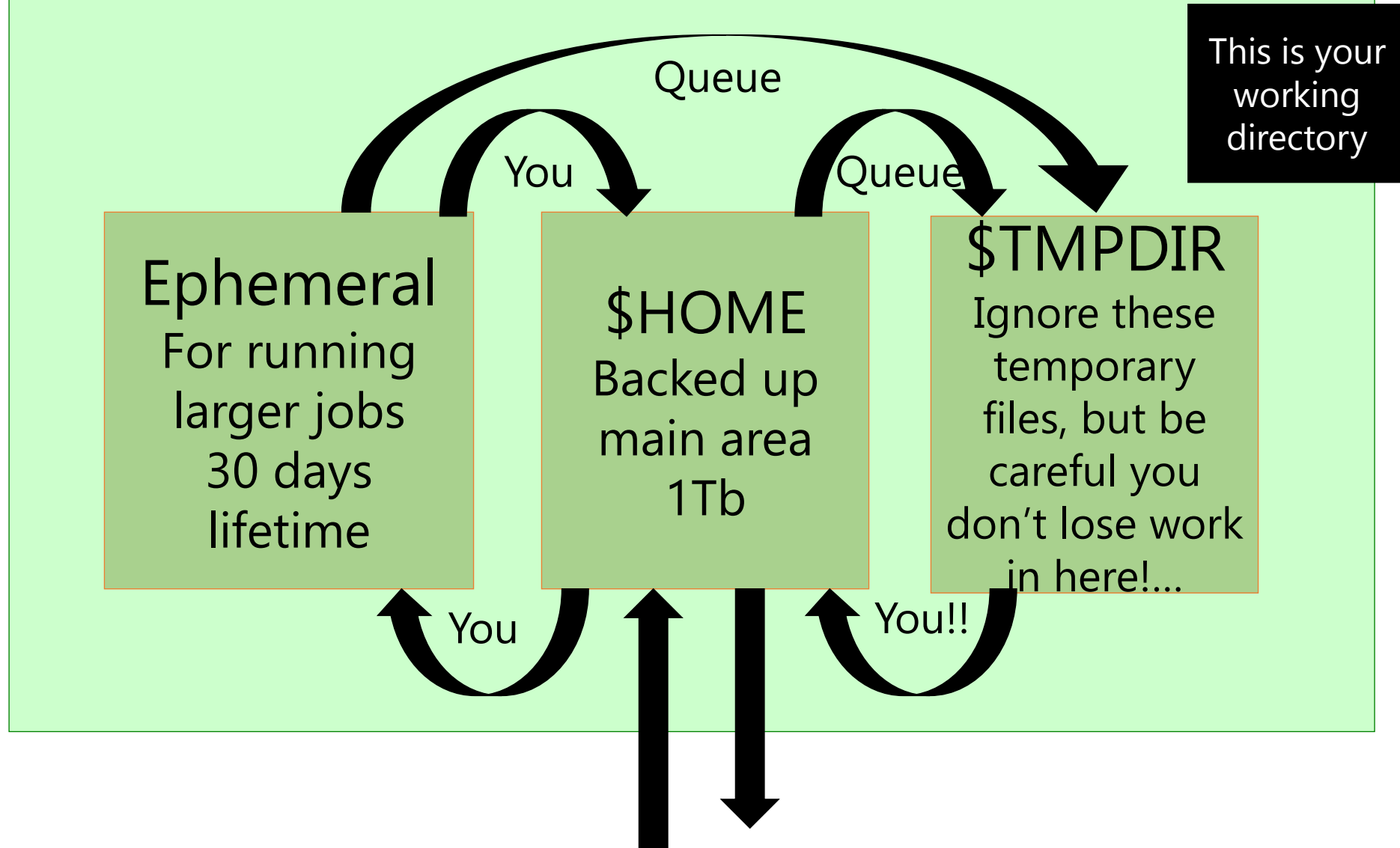
automatically

time

Job sizing

Job class	Number of nodes N	ncpus/node	Max mem/node	Max walltime/hr	Max number of running jobs per user
throughput	1	1-8	96GB	up to 72hr	unlimited for jobs <=24hr in length
general	1 - 16	32	62GB or 124GB	up to 72hr	unlimited for jobs <=24hr in length
singlenode	1	48	124GB	up to 24hr	10
multinode	3 - 16	12	46GB	up to 48hr	unlimited
debug	1	1-8	96GB	up to 30 mins	1
large memory	1	12 or 24	190 or 380GB	48 hr	unlimited
GPU	1-2	1-8	1 - 32GB	48hr 72hr (P1000 only)	8
long	1	1-8	96GB	72 - 1000 hr	1

Where your data is stored...



Step 1: A parallelised script

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

e.g.:

```
1 # Find out the job number:
2 seed_number <- as.numeric(Sys.getenv("PBS_ARRAY_INDEX"))
3
4 # Set this as the random seed so that all runs have a unique seed:
5 set.seed(seed_number)
6
7 # Run whatever simulation we want:
8 output <- runif(n=10000,min=0,max=1) # this is just an example "simulation"
9
10 # Save this to a file, ensuring each output has a unique filename:
11 save(output,file=paste("output_",seed_number,".rda"))
12 # You could also do this part inside another R function which you call
13
14 # Remove our objects from the environment:
15 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

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 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)

```
[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 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`

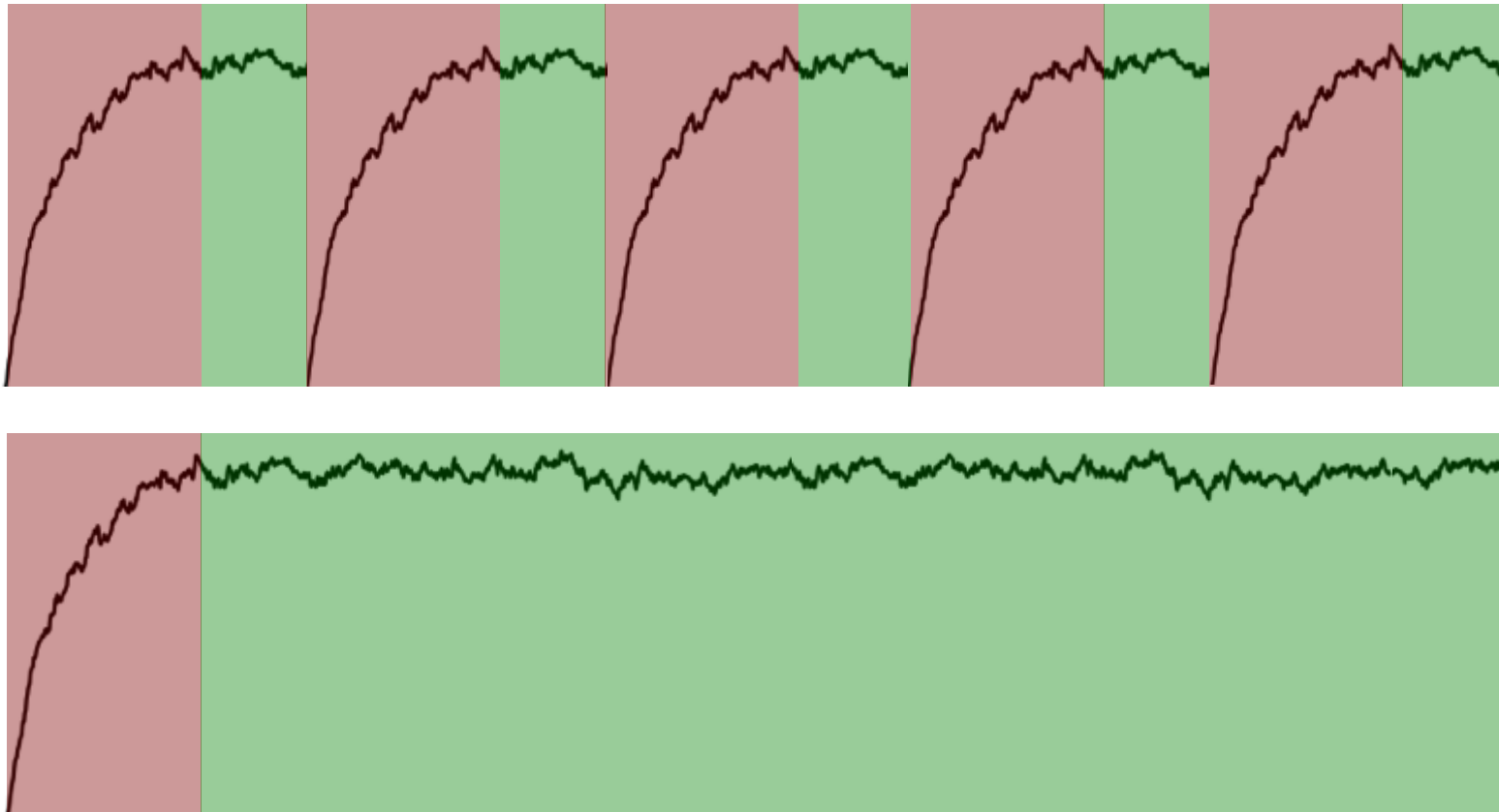
Step 6: Get your results back (continued)

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.cx1.hpc.ic.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.

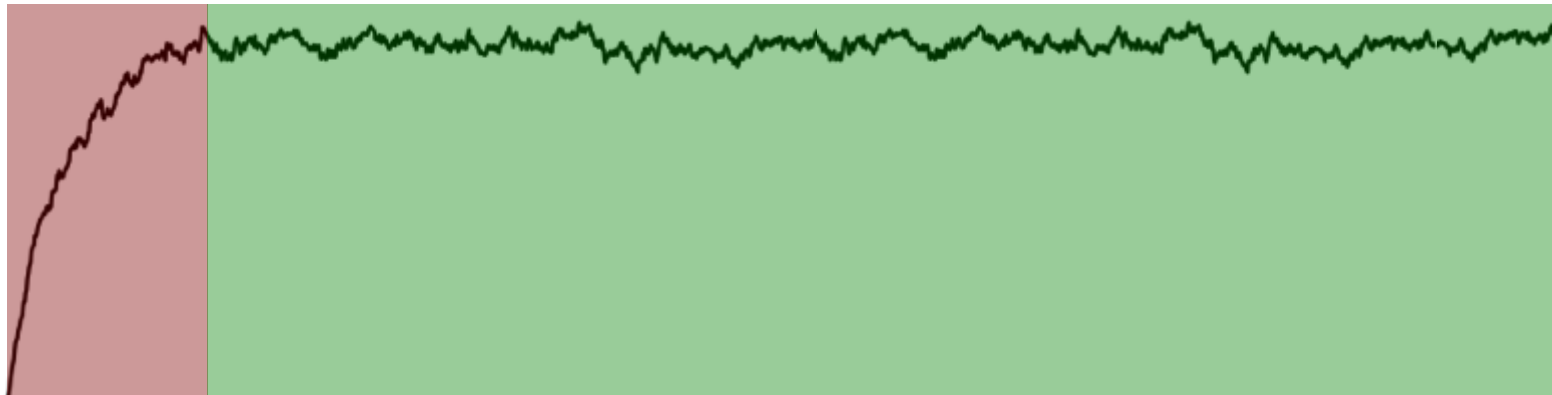
For your neutral theory exercises

- You'll be asked to adapt your code from yesterday to run on the cluster for a much bigger ecological community size
- You'll need to collect species abundance data as before and average over a large number of parallel simulations.



For your neutral theory exercises

- You'll be asked to adapt your code from yesterday to run on the cluster for a much bigger ecological community size
- You'll need to collect species abundance data as before and average over a large number of parallel simulations.
- Use a "burn in" period and check the species abundance distribution periodically. You should plot species richness against time and make a conservative judgment, but for neutral theory $8 \times$ metacommunity size complete turnovers of the community is a good rule of thumb.





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

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

4. Demonstration