Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

# Introduction to HPC at Imperial

Katerina Michalickova – kmichali@imperial.ac.uk

Early Career Researcher Institute, Research Computing and Data Science Programme

IMPERIAL
Early Career Researcher Institute

## Outline for today

- introduction, key links and contacts
- preparing to use the HPC cluster
  - working remotely and login
  - software management on the cluster
  - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

Imperial College London

## Why are we here today?

- Pillars (legs) of science today – theory, experimentation *and computing*.

- Much of scientific experimentation such as modeling, simulations or data management and mining is made possible by computing.

- If you need to use a central resource, it is very likely it will come in a form of **high performance computing cluster**.

Imperial College London

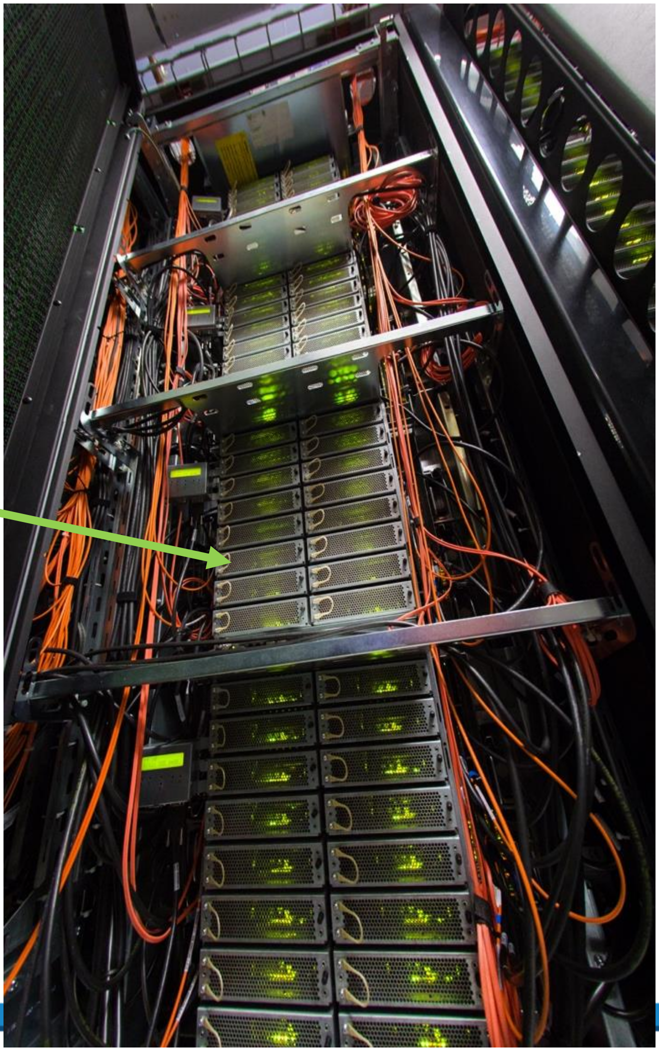
RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Computer cluster

- A **computer cluster** consists of a set of connected computers that work together so that, in many respects, they can be viewed as a single system.

## Inside a compute rack

a single computer
aka a compute node



## Inside a disk rack

All compute nodes see one central file system – the Research Data Store

Imperial College
London

## What can clusters do?

- the cluster can serve to offload code execution from your laptop/workstation
  - code that runs too long or needs too much memory or disk space
- clusters are particularly useful for executing parallel code
  - on one compute node
  - on multiple compute nodes at once

- Note on speed of execution:
- the compute nodes have similar architecture to your desktop - they are not much faster
- the main advantage of cluster computing lies in parallel code execution

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## HPC terminology

I will try to avoid or explain again during the class. In case I forget myself, here are few terms that need explanation:

- **Node** = compute node = group of cores that can access the same memory (I may inadvertently also say a computer or a machine)
- **Memory** = main memory or RAM - fast memory directly connected to the processor, when your program is running it is stored in RAM together with needed data
- **Core** = the basic computation unit inside a processor that can run a single process
- **Serial code** = runs on one core
- **Parallel code** = program that runs on two or more cores
- **Job** = your program on the cluster
- **Submit job** = instruct the cluster to run your program

## Workload types

The resource contains hardware components suited to different type of computations:

- High-throughput tasks – large number of relatively small jobs
- Parallel jobs on a single node
- Parallel jobs on multiple nodes
- Large single node memory jobs (up to TBs of memory)
- Jobs using GPUs (such as machine learning tasks)

## Key contacts

- RCS website - https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/
- User support - https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/get-support/contact-us/
- Technical documentation  https://icl-rcs-user-guide.readthedocs.io/en/latest/
- Weekly drop-in clinics - https://icl-rcs-user-guide.readthedocs.io/en/latest/support/attend-a-clinic/
- Service status (under Zscaler)- https://selfservice.rcs.imperial.ac.uk/service-status
- Self-service (under Zscaler, interactive access, group management, job monitoring and extensions etc.) - selfservice.rcs.imperial.ac.uk

## Outline for today

- introduction, key links and contacts
- preparing to use the HPC cluster
    - working remotely and login
    - software management on the cluster
    - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

Imperial College London

## Before you login

- use **Zscaler** when off-campus
https://www.imperial.ac.uk/admin-services/ict/self-service/connect-communicate/remote-access/unified-access/

- on campus - use **Imperial-WPA wifi**

- for **regular HPC access**, register as follows
https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/get-access/

- some of you have only temporary access for this course - it expires on the first of each month

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Log in using your terminal

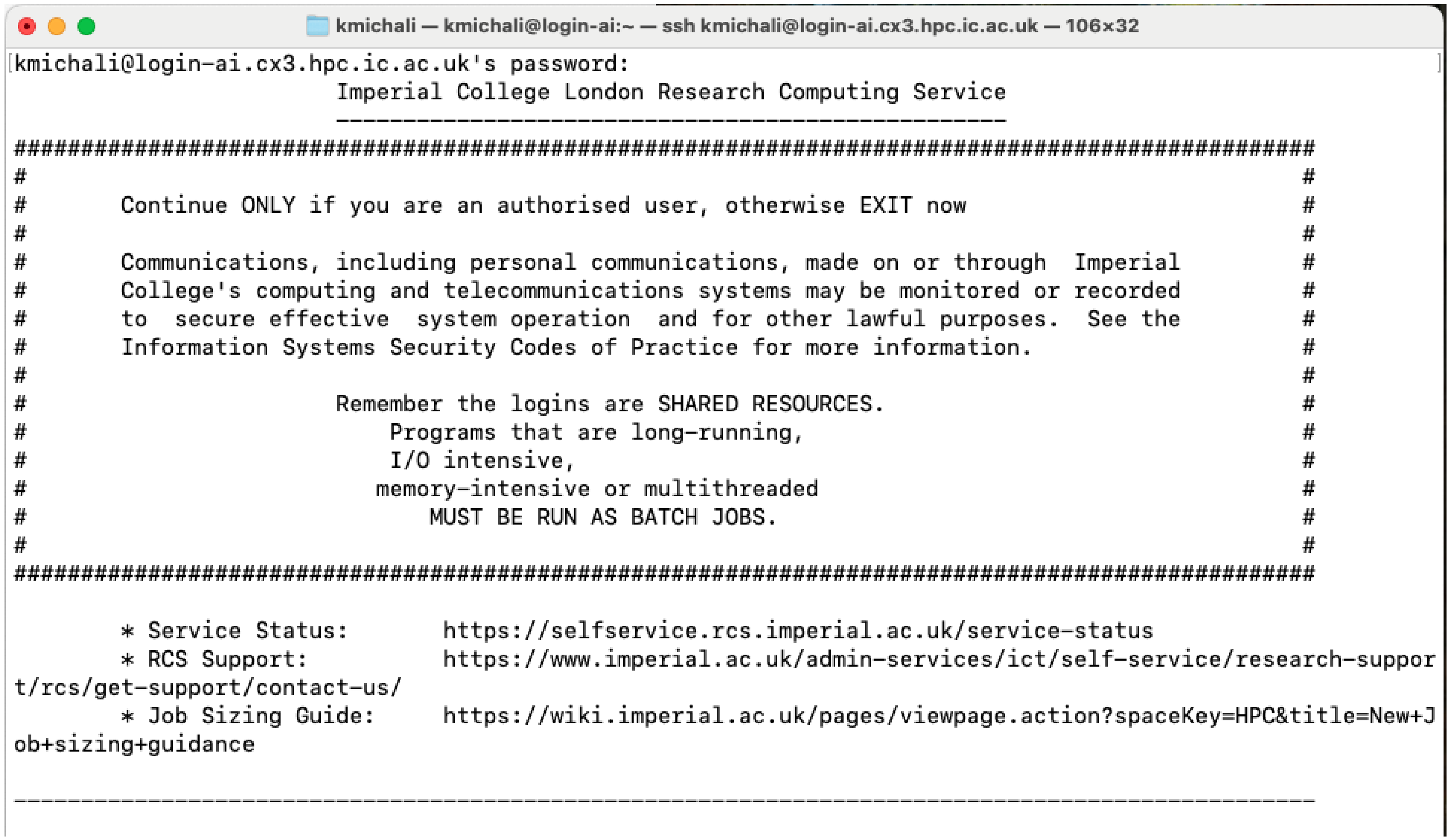
In a terminal, type one of the following lines:

```
ssh your_username@login-b.cx3.hpc.ic.ac.uk
ssh your_username@login-ai.cx3.hpc.ic.ac.uk
ssh your_username@login-bi.cx3.hpc.ic.ac.uk
```

- This course uses the cx3_phase2 system that is now the recommended system to use
- https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/pilot/cx3-phase2/

## HPC environment

- Linux operating system
- Bash shell



kmichali — kmichali@login-ai:~ — ssh kmichali@login-ai.cx3.hpc.ic.ac.uk — 106×32

```
kmichali@login-ai.cx3.hpc.ic.ac.uk's password:
                        Imperial College London Research Computing Service
                        --------------------------------------------------
#################################################################################################
#                                                                                               #
#       Continue ONLY if you are an authorised user, otherwise EXIT now                         #
#                                                                                               #
#       Communications, including personal communications, made on or through  Imperial        #
#       College's computing and telecommunications systems may be monitored or recorded        #
#       to  secure effective  system operation  and for other lawful purposes.  See the        #
#       Information Systems Security Codes of Practice for more information.                    #
#                                                                                               #
#                       Remember the logins are SHARED RESOURCES.                               #
#                           Programs that are long-running,                                     #
#                           I/O intensive,                                                      #
#                          memory-intensive or multithreaded                                    #
#                              MUST BE RUN AS BATCH JOBS.                                       #
#                                                                                               #
#################################################################################################

        * Service Status:       https://selfservice.rcs.imperial.ac.uk/service-status
        * RCS Support:          https://www.imperial.ac.uk/admin-services/ict/self-service/research-suppor
t/rcs/get-support/contact-us/
        * Job Sizing Guide:     https://wiki.imperial.ac.uk/pages/viewpage.action?spaceKey=HPC&title=New+J
ob+sizing+guidance

-------------------------------------------------------------------------------------------------
```

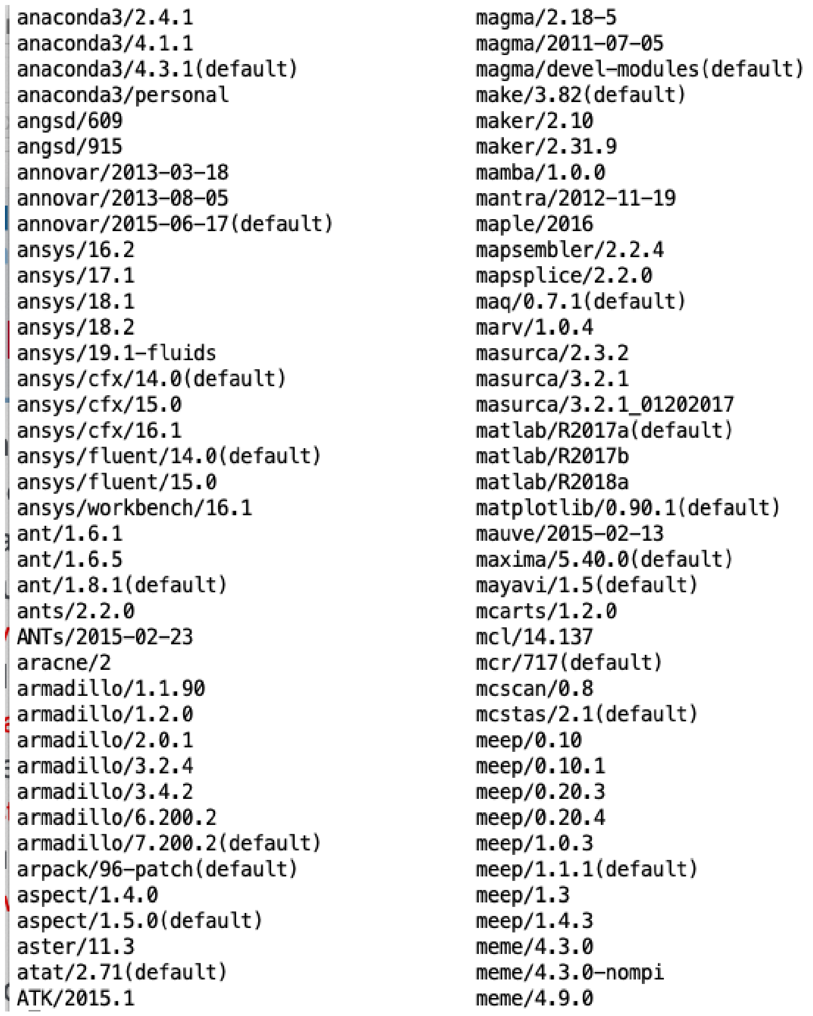
## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
  - working remotely and login
  - software management on the cluster
  - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

## Module system for centrally installed packages

- Lmod system sets required environment to use a software package

- List all modules available
  - ml av or ml ov (available or overview)
- Find a module for your software
  - ml av matlab or ml spider matlab
- Load a module
  - ml MATLAB
- Load a specific version
  - ml MATLAB/2024b
- List all loaded modules
  - ml
- Swap modules
  - ml switch MATLAB/2024b MATLAB/2023b
- Unload a module
  - ml unload matlab/R2018a
- Get rid of all loaded modules
  - ml purge



```
anaconda3/2.4.1                     magma/2.18-5
anaconda3/4.1.1                     magma/2011-07-05
anaconda3/4.3.1(default)            magma/devel-modules(default)
anaconda3/personal                  make/3.82(default)
angsd/609                           maker/2.10
angsd/915                           maker/2.31.9
annovar/2013-03-18                  mamba/1.0.0
annovar/2013-08-05                  mantra/2012-11-19
annovar/2015-06-17(default)         maple/2016
ansys/16.2                          mapsembler/2.2.4
ansys/17.1                          mapsplice/2.2.0
ansys/18.1                          maq/0.7.1(default)
ansys/18.2                          marv/1.0.4
ansys/19.1-fluids                   masurca/2.3.2
ansys/cfx/14.0(default)             masurca/3.2.1
ansys/cfx/15.0                      masurca/3.2.1_01202017
ansys/cfx/16.1                      matlab/R2017a(default)
ansys/fluent/14.0(default)          matlab/R2017b
ansys/fluent/15.0                   matlab/R2018a
ansys/workbench/16.1                matplotlib/0.90.1(default)
ant/1.6.1                           mauve/2015-02-13
ant/1.6.5                           maxima/5.40.0(default)
ant/1.8.1(default)                  mayavi/1.5(default)
ants/2.2.0                          mcarts/1.2.0
ANTs/2015-02-23                     mcl/14.137
aracne/2                            mcr/717(default)
armadillo/1.1.90                    mcscan/0.8
armadillo/1.2.0                     mcstas/2.1(default)
armadillo/2.0.1                     meep/0.10
armadillo/3.2.4                     meep/0.10.1
armadillo/3.4.2                     meep/0.20.3
armadillo/6.200.2                   meep/0.20.4
armadillo/7.200.2(default)          meep/1.0.3
arpack/96-patch(default)            meep/1.1.1(default)
aspect/1.4.0                        meep/1.3
aspect/1.5.0(default)               meep/1.4.3
aster/11.3                          meme/4.3.0
atat/2.71(default)                  meme/4.3.0-nompi
ATK/2015.1                          meme/4.9.0
```

## Cannot find what you’re looking for?

- submit a request for installation (variable waiting times): https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/get-support/contact-us/

- you can install packages in your home directory (take advantage of various pre-installed libraries via module system)

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Setting up Python with Easybuild*

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/applications/guides/python/

```
ml tools/prod #always have to be loaded
ml av Python  #check out Python available
ml av SciPy-bundle #scientific Python bundle
ml SciPy-bundle/2024.05-gfbf-2024a #load scientific bundle
mkdir -p  ~/venv #make dir for virtual environments
virtualenv ~/venv/example_env #make a virtual env ”example_env”
source ~/venv/example_env/bin/activate #activate virtual env
python -m pip install some_python_module #use pip to install what you need
deactivate
```

***Easybuild is a community-based repository of optimised software, libraries and packages for HPC - https://easybuild.io/**

## Using python with Anaconda

We distribute personal version of Anaconda environment on the original cluster (ssh login.hpc.ic.ac.uk). Install on the command line using commands below. This is done only once.

```
module load anaconda3/personal
anaconda-setup
```

On current cluster, load as follows:

```
eval "$(~/anaconda3/bin/conda shell.bash hook)"
```

Anaconda enables you to create separate package environments for your projects. This helps to avoid version and dependency conflicts.

```
conda create -n projectx          #create new environment
conda activate projectx            #activate env. (conda/source)
conda install scipy                #install scipy
conda deactivate                   #deactivate current env.
```

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/applications/guides/python/

## Setting up R with Easybuild

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/applications/guides/R/

```
ml tools/prod #always have to be loaded
ml spider R  #check available R
ml av R-bundle #check the scientific bundles
ml R-bundle-ICL/2024.10-foss-2023b #load ICL R-bundle
```

## Installing own R modules

- You can install your own modules within R. Before you start, you have set a correct environment (path to R libraries). Since the cx3_phase2 is a mixture of two CPU types – Intel and AMD, it matters which login node you install on. Login-b is an AMD node while login-ai and bi are Intel nodes.

- Ideally, you should install on both - AMD and Intel - separately. Your new module (adapted for the right architecture) will be stored in two different directories under your home directory.

- Your job will utilize the correct one depending on which compute node your job executes. This means that you have to add a few lines into your jobs scripts to detect the CPU type.

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Own R modules – setting up

- On login node, set up path to your installation and R dependencies by typing the following five commands:

```
ml tools/rlibpath
cpu=$(rlibpath.sh | awk -F " " 'END{print}')
export R_LIBS_USER=$HOME/R/${cpu}
mkdir -p ${R_LIBS_USER}
```

- You are done setting up, you should see a new directory: either ~/R/rome or ~/R/icelake
- To make sure that you are set up for both types of CPUs in the cluster, do this on login-b and also on login-ai (or login-bi)

## Own R modules – installing a module

- Once your path to your modules is set up on both Intel and AMD nodes, install your module using R. Do this again on both CPU types – on login-b as well as on login-ai (or login-bi).

- On login node, load and start R

```
ml tools/prod
ml R-bundle-ICL
R
```

- Within R, use install.packages(‘package_name’) to install a module

## Own R modules – example job script

- If you use modules that you installed yourself, you must set the right environment in your script; the same way as you did before installing a module. Example script:

```
#PBS -l walltime=08:00:00
#PBS -l select=1:ncpus=1:mem=2gb

#set up R environment for current CPU type
ml tools/rlibpath
cpu=$(rlibpath.sh | awk -F " " 'END{print}')
export R_LIBS_USER=$HOME/R/${cpu}
mkdir -p ${R_LIBS_USER}

#load R module
ml tools/prod
ml R-bundle-ICL

#execute your script
R CMD BATCH a_script.R
```

## Using R with Anaconda

R and libraries can be installed using personal Anaconda.

Setup an a new environment and install R inside it.  Packages can be installed using conda install or from within R (plus other libs, e.g. bioconductor-teqc)

```
eval "$(~/anaconda3/bin/conda shell.bash hook)"
conda create -n Renv
conda activate Renv                  # (conda or source)
conda install r -c conda-forge      #install R in new env.
                                     #using conda-forge channel
conda install bioconductor-teqc -c conda-forge
conda deactivate
```

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/applications/guides/R/

## Application-specific guidance

Information for selected popular applications:

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/applications/

Note: These may have to be adapted for the cx3_phase2 or used via the old cx3 (ssh username@login.hpc.ic.ac.uk). The documentation should eventually reflect only the cx3_phase2.

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
    - working remotely and login
    - software management on the cluster
    - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

Imperial College London

## Before you start

- If not on campus, use Imperial **VPN or Zscaler**
- On campus - use **Imperial-WPA wi-fi**

## File management

All computers in the HPC resource are connected to one parallel filesystem – Research Data Store (RDS). You get access to:

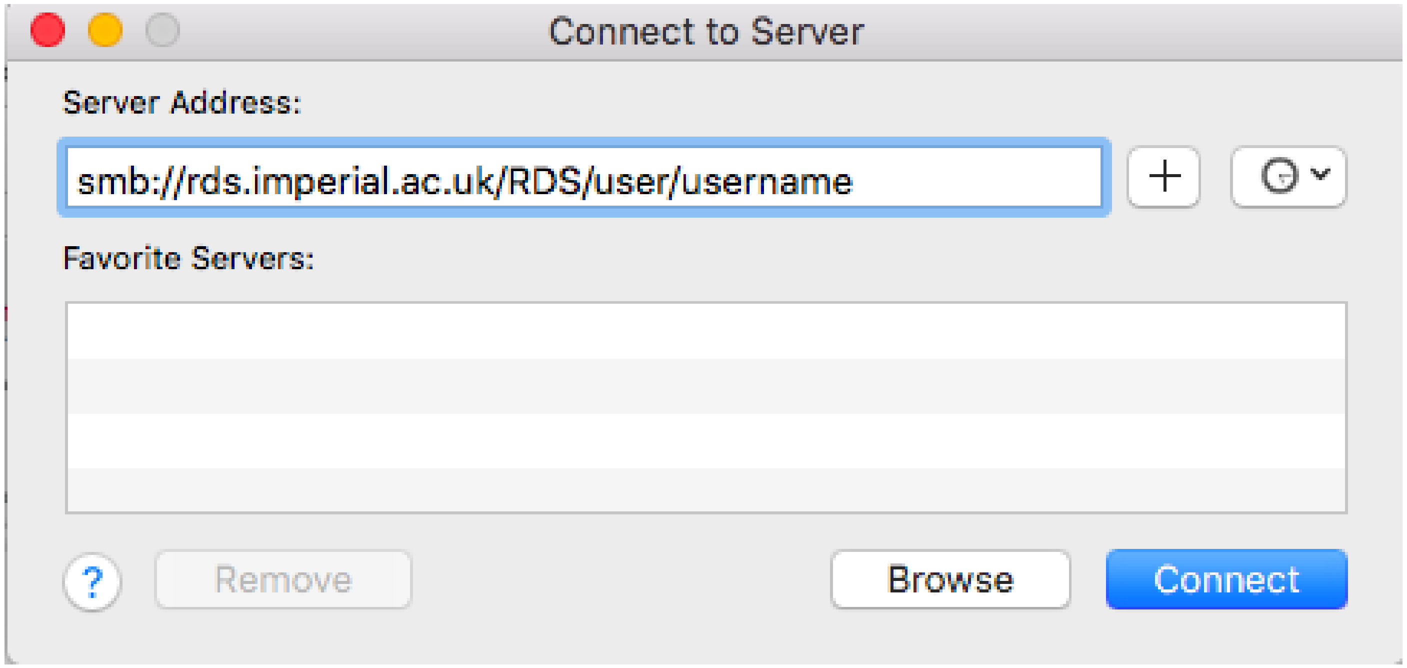
| | |
|---|---|
| your home directory ($HOME) | • personal working space<br>• 1 TB allocation (up to 10 million files) |
| temporary storage ($EPHEMERAL) | • additional individual working space<br>• unlimited allocation<br>• files deleted after 30 days |
| allocated project space ($RDS_PROJECT) | • your project allocations (if your supervisor has any) |

check your usage with “quota -s”

documentation: [https://icl-rcs-user-guide.readthedocs.io/en/latest/rds/](https://icl-rcs-user-guide.readthedocs.io/en/latest/rds/)

## 1a. Connect to RDS with Finder on your Mac

- **on a Mac (recent macOS version)**
- Finder -> Go -> Connect to Server -> Server Address
- **`smb://rds.imperial.ac.uk/RDS/user/your_username`**



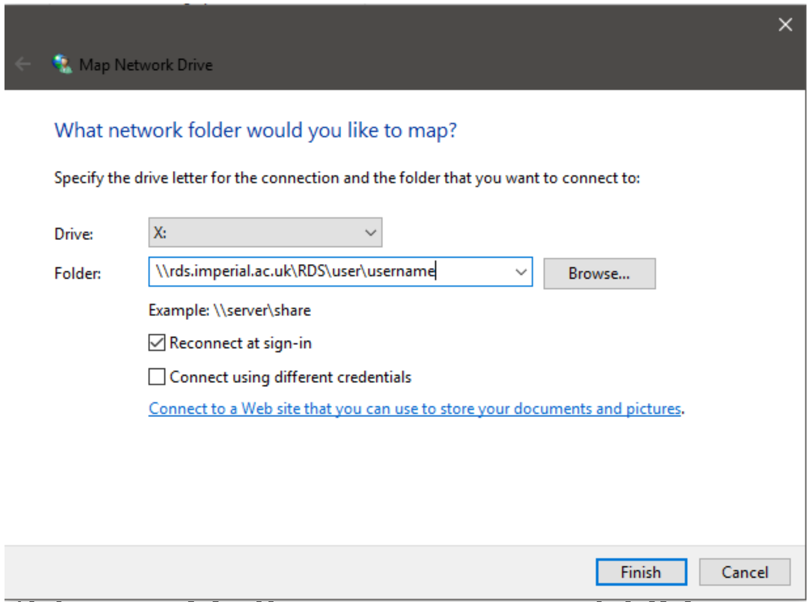
Connect to Server

Server Address: smb://rds.imperial.ac.uk/RDS/user/username

Favorite Servers:

## 1b. Connect to RDS with File Explorer on Windows

- **recent update of Windows 10**
- File Explorer -> My PC -> Map network drive
- `\\rds.imperial.ac.uk\RDS\user\your_username`



Map Network Drive

What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: X:

Folder: \\rds.imperial.ac.uk\RDS\user\username

Example: \\server\share

☑ Reconnect at sign-in

☐ Connect using different credentials

Connect to a Web site that you can use to store your documents and pictures.

- if you don't have College-managed PC, you may need to use "**IC\username**" when loging in.

## 1b. Connect to RDS with File Explorer on Windows

Detailed description of connecting to RDS from Windows File Explorer:

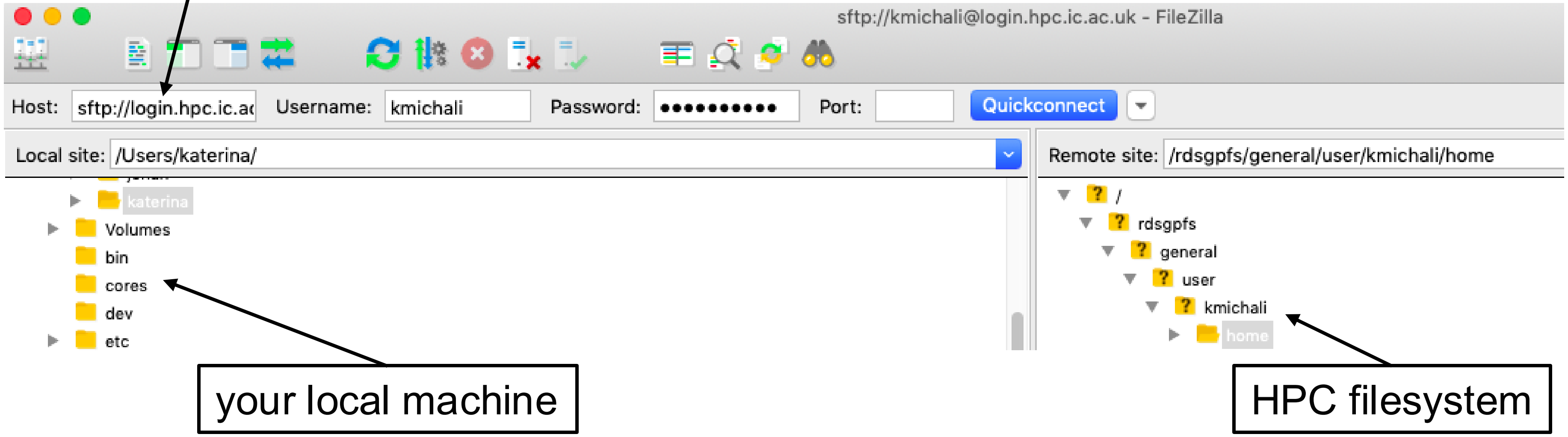
https://github.com/ImperialCollegeLondon/RCS_UserSupport_public/tree/main/RDS_FSmounting/MS_Windows

## 2. Copying files between your laptop to RDS with FileZilla

- if on laptop, use **Imperial-WPA wifi** or **VPN** outside the College
- **FileZilla** file transfer client: https://filezilla-project.org

  type **sftp://login.hpc.ic.ac.uk** into the host window,

  (add username, password, leave port empty, click Quickconnect)


sftp://kmichali@login.hpc.ic.ac.uk - FileZilla
Host: sftp://login.hpc.ic.ac
Username: kmichali
Password: ●●●●●●●●●●●
Port:
Quickconnect
Local site: /Users/katerina/
katerina
Volumes
bin
cores
dev
etc
Remote site: /rdsgpfs/general/user/kmichali/home
/
rdsgpfs
general
user
kmichali
home
your local machine
HPC filesystem

- **WinSCP – good alternative for Windows** https://winscp.net/

## 3. Copying files on the command line

**Secure copy (scp) command will copy files between remote systems**

- **scp   file.txt   username@login.hpc.ic.ac.uk:~**

 this copies file.txt from the current directory on your machine to your home directory on RDS

- **scp  username@login.hpc.ic.ac.uk:~/file2.txt   .**

 this copies a file2.txt from your home directory on RDS to the current directory on your machine

- The scp command can be used with any directory path, for example:

```
scp /Users/katerina/Desktop/file3.txt  username@login.hpc.ic.ac.uk:/rds/general/user/kmichali/home/projectx
```

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
  - working remotely and login
  - software management on the cluster
  - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

## Your command

Assembling the command line is your responsibility, it requires some knowledge of the particular package.  We can help with making sure the application runs correctly and advise you on managing your files.  We can comment on individual applications to some degree (depending on having the expertise in the group).

Examples:

- **python** python_code.py –i input.txt –o output.txt
- **matlab** –batch matlab_script
- **R** CMD BATCH r_script.R
- **bash** shell_script.sh
- **g09** water.com > log
- **blastall** -i query -d fasta_db -o output
- mpiexec **nwchem** input.nw > log

## Running software

Do not execute your program from the command line.

Why?

When you log into the cluster, you find yourself on one of four login nodes. If all users compute here, the nodes would be overwhelmed and the rest of the cluster would be idle. Instead, instruct the **cluster resource manager** to execute your job.

## Cluster resource manager

The resource manager takes care of:

- keeps track of available hardware resources and how busy they are
- sorting your requests into job classes
- scheduling your requests
- starting jobs when compute resources become available
- monitoring jobs
- producing logs

x

## Talking to the resource manager

- Write **a shell script (or job script)** instructing the resource manager what to do for you. Passing your finished script to the resource manager is called “submitting a job”.

- The resource manager needs to know your **command** and also what **resources** your job needs.

  - Every job script must specify **run time, memory (RAM) and number of cores** needed to run the job.

  - If the requested resources are exceeded during the job run, the job is terminated.

  - The bigger the resource, the longer the wait for a job to start.

Imperial College London

## Download practice repo

```
git clone https://github.com/kmichali/hellohpc.git
```

## A job script

```
#!/bin/bash
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml Python

Python $PBS_O_WORKDIR/hellohpc.py
```

job params

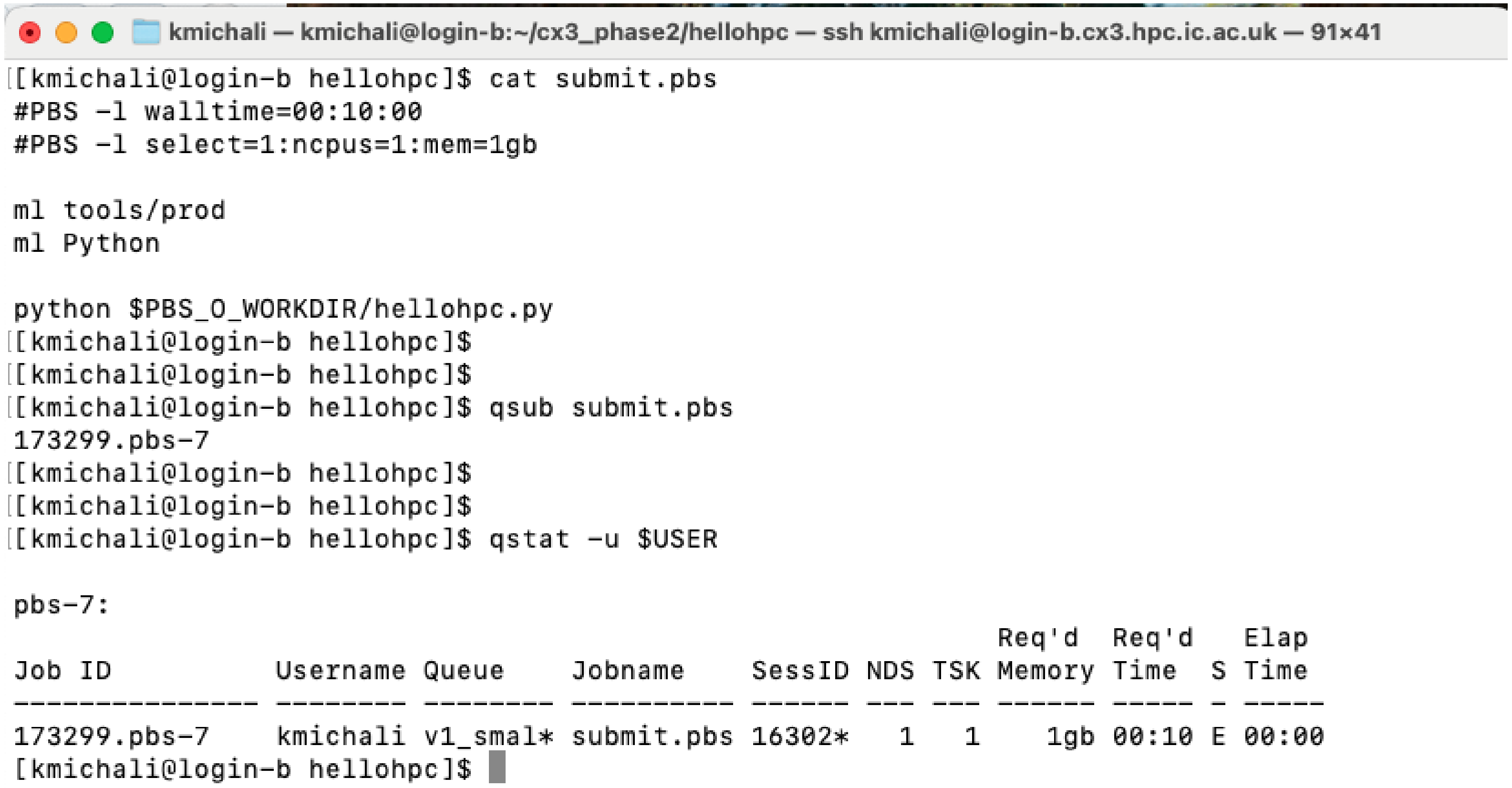
load software

your command

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Submit and monitor a job



kmichali — kmichali@login-b:~/cx3_phase2/hellohpc — ssh kmichali@login-b.cx3.hpc.ic.ac.uk — 91×41

```
[kmichali@login-b hellohpc]$ cat submit.pbs
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml Python

python $PBS_O_WORKDIR/hellohpc.py
[kmichali@login-b hellohpc]$
[kmichali@login-b hellohpc]$
[kmichali@login-b hellohpc]$ qsub submit.pbs
173299.pbs-7
[kmichali@login-b hellohpc]$
[kmichali@login-b hellohpc]$
[kmichali@login-b hellohpc]$ qstat -u $USER

pbs-7:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
173299.pbs-7    kmichali v1_smal* submit.pbs 16302*   1   1    1gb 00:10 E 00:00
[kmichali@login-b hellohpc]$
```

## Resource manager commands

| | | |
|---|---|---|
| submit | **qsub** your_script | system returns a jobid |
| monitor | **qstat**<br>qstat job_id | short overview of your jobs<br>one job overview |
| delete | **qdel** jobid | caution: it takes a bit of time to see the job disappear from qstat output |

## Job status using selfservice

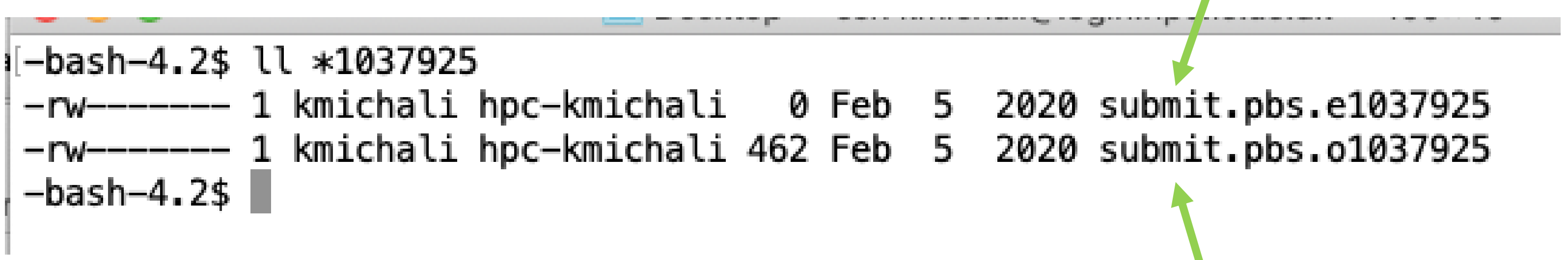
This page will list all your active jobs (queuing and running).

https://selfservice.rcs.imperial.ac.uk/jobs/qstat

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## Every job produces two log files


```
-bash-4.2$ ll *1037925
-rw------- 1 kmichali hpc-kmichali   0 Feb  5  2020 submit.pbs.e1037925
-rw------- 1 kmichali hpc-kmichali 462 Feb  5  2020 submit.pbs.o1037925
-bash-4.2$ 
```

error messages

screen output + res. manager messages

Imperial College
London

## Troubleshooting your jobs

All standard output (screen and errors - STDOUT and STDERR) from your job plus other useful info is written into log files. **Read the log files** carefully even if all seems fine.

Default names:

STDOUT - e.g. `submit.pbs.o9795758`

STDERR - e.g. `submit.pbs.e9795758`

**If asking for help**, use the ICT ASK form at https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/get-support/contact-us/

provide as much information as your can to help us troubleshoot:

- job id
- steps to reproduce the error
- your script
- the logs (expected and observed)

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Example of a python script

```
#!/bin/bash
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml SciPy-bundle

source ~/venv/your_env/bin/activate
cd $PBS_O_WORKDIR
python myscript.py
```

## Example of a R script

```
#!/bin/bash
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml R-bundle-ICL

cd $PBS_O_WORKDIR
R CMD BATCH myscript.R
```

## Matlab script

```
#!/bin/bash
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

ml MATLAB

cd $PBS_O_WORKDIR
matlab -batch myscript
```

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Introduction to HPC at Imperial

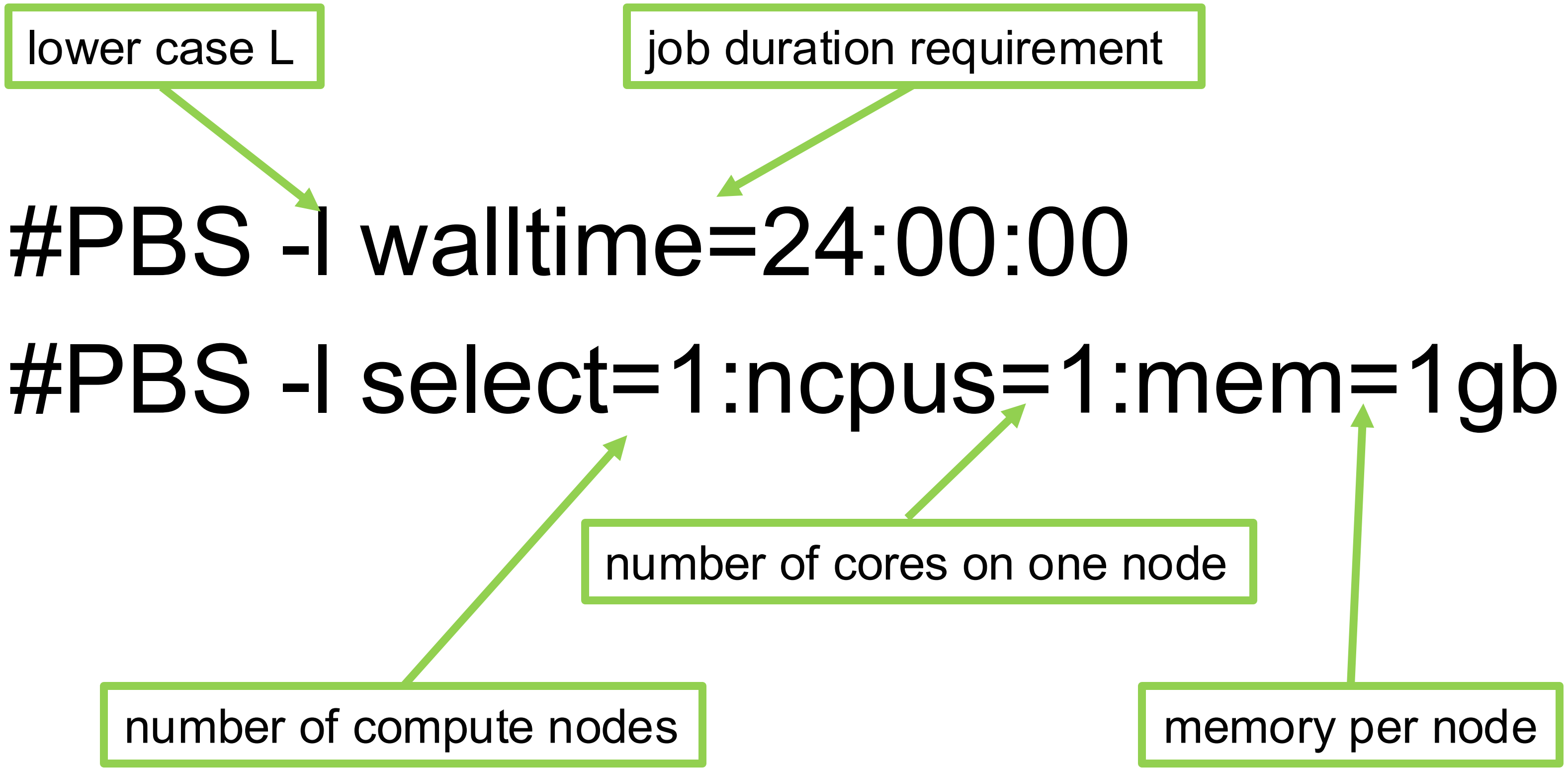
- introduction, key links and contacts
- preparing to use the HPC cluster
    - working remotely and login
    - software management on the cluster
    - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## Requirements syntax

The system expects the following syntax at the top of your job script:



```
#PBS -l walltime=24:00:00
#PBS -l select=1:ncpus=1:mem=1gb
```

## Job duration requirement

- This specifies how long you expect the program to run.
- If you are not sure at all how long your job will run, sensible values are 8, 24 hours, 48 hours or 72 hours. Otherwise choose time that you think your job will take plus some just to be sure.
- If possible, use checkpointing, i.e. save intermediate results that can be used to restart your program.

## Memory requirement

- When you run a program, the program and data are read into the main memory (RAM). The data is processed in memory and eventually written out (usually into a file on the disk).

- Majority of programs that do not deal with lots of data need no more than 2gb of RAM. The data usually determines the bulk of memory requirement. If your program reads large data files, ask for more memory.

- If in doubt, select 2GB to start with. If this is not enough, the resource manager will terminate your program and place a message (including how much RAM you tried to use) in a **log file**. In this case, increase the memory requirement and run again.

## Number of parallel processes

- You program may be capable of parallel execution. If you are not sure, read the manual, ask your colleagues or check the HPC logs that specify how many cores your program tried to use. Failing that, visit the RCS clinic.

- On the HPC cluster, we generally see two types of parallelism:
  - shared memory (jobs that run on one node using threads)
  - distributed memory (jobs that run on multiple nodes using MPI)

## Submit script for threaded parallel jobs

Threaded (OpenMP) jobs run on multiple cores on a single node - they require shared memory.  In your script, increase ncpus count, memory (to accommodate more cores) and add ompthreads parameter (ompthreads is always equal or less than ncpus).

```
#PBS -l walltime=00:01:00
#PBS -l select=1:ncpus=4:mem=16gb:ompthreads=4
```

number of cores per node

memory per node

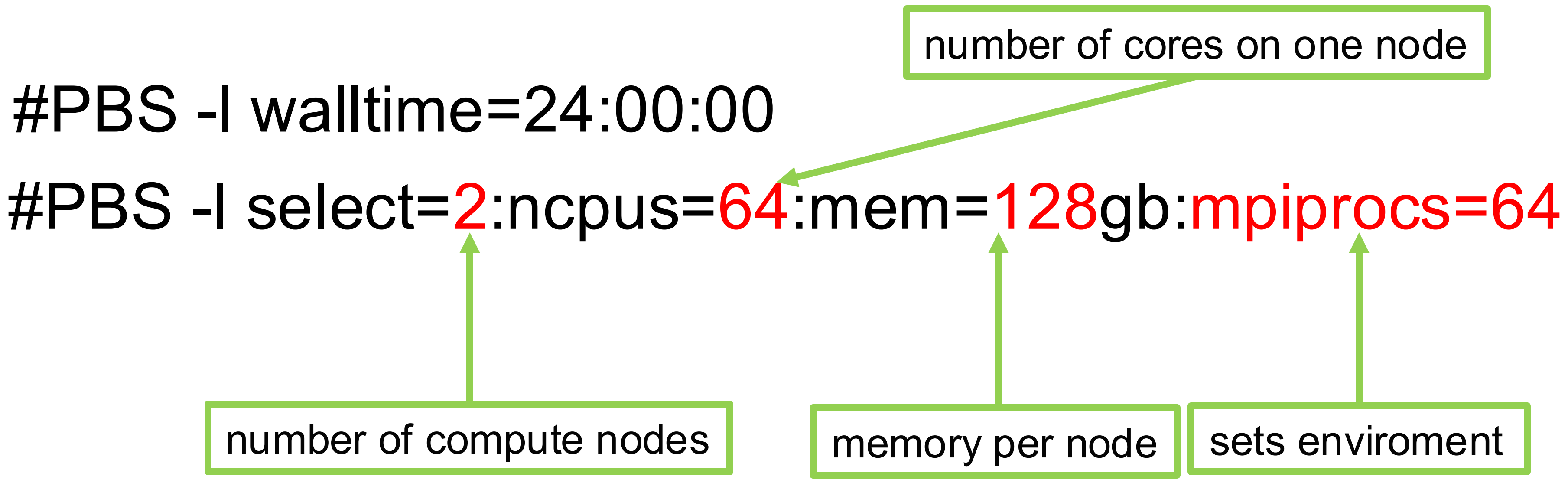
sets env. variable

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Submit script for MPI jobs

MPI jobs don’t require shared memory and run on multiple cores on multiple nodes. State the number of nodes using “select”. The rest of the params (ncpus, mem and mpiprocs) state the requirement per one node; mpiprocs are always equal (or less than) ncpus.



```
#PBS -l walltime=24:00:00
#PBS -l select=2:ncpus=64:mem=128gb:mpiprocs=64
```

## Possible job paramenters

Decide what your job needs and select your job params so your job fits one of the following (wide) categories:

https://icl-rcs-user-guide.readthedocs.io/en/latest/hpc/pilot/cx3-phase2/#job-sizing-guidance

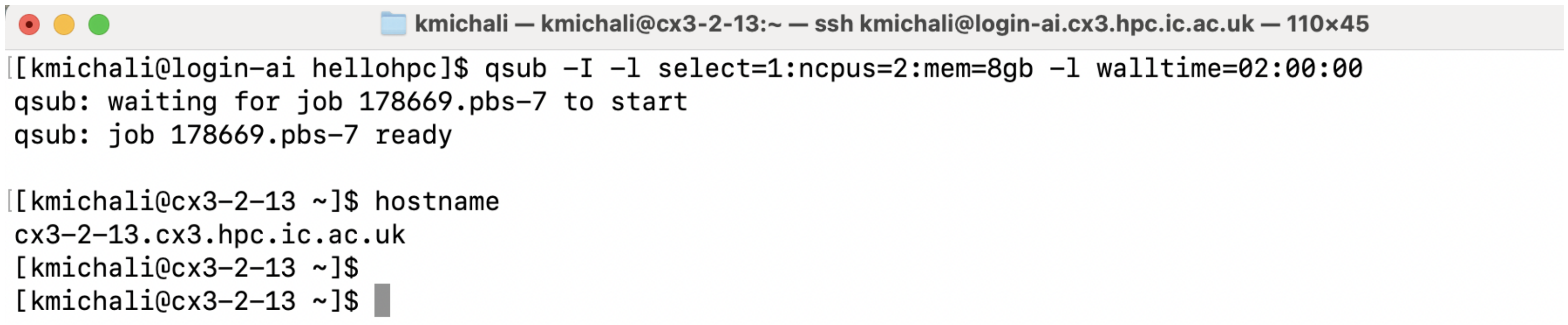
## Interactive jobs

- It is possible to have a direct access to bash shell on a compute cluster via an ”interactive” job.

- In terminal type (adapt your job requirement as you need):

qsub -I -l select=1:ncpus=2:mem=8gb -l walltime=02:00:00

- The job will queue and when it starts running you will have access to bash shell directly on a compute node.


kmichali — kmichali@cx3-2-13:~ — ssh kmichali@login-ai.cx3.hpc.ic.ac.uk — 110×45

```
[kmichali@login-ai hellohpc]$ qsub -I -l select=1:ncpus=2:mem=8gb -l walltime=02:00:00
qsub: waiting for job 178669.pbs-7 to start
qsub: job 178669.pbs-7 ready

[kmichali@cx3-2-13 ~]$ hostname
cx3-2-13.cx3.hpc.ic.ac.uk
[kmichali@cx3-2-13 ~]$
[kmichali@cx3-2-13 ~]$
```

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
    - working remotely and login
    - software management on the cluster
    - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs

## Current working directory for a job

When a job starts running, **the current working directory changes.** On cx3_phase2, it is your home directory.

This is somewhat counter-intuitive and can lead to file management errors.

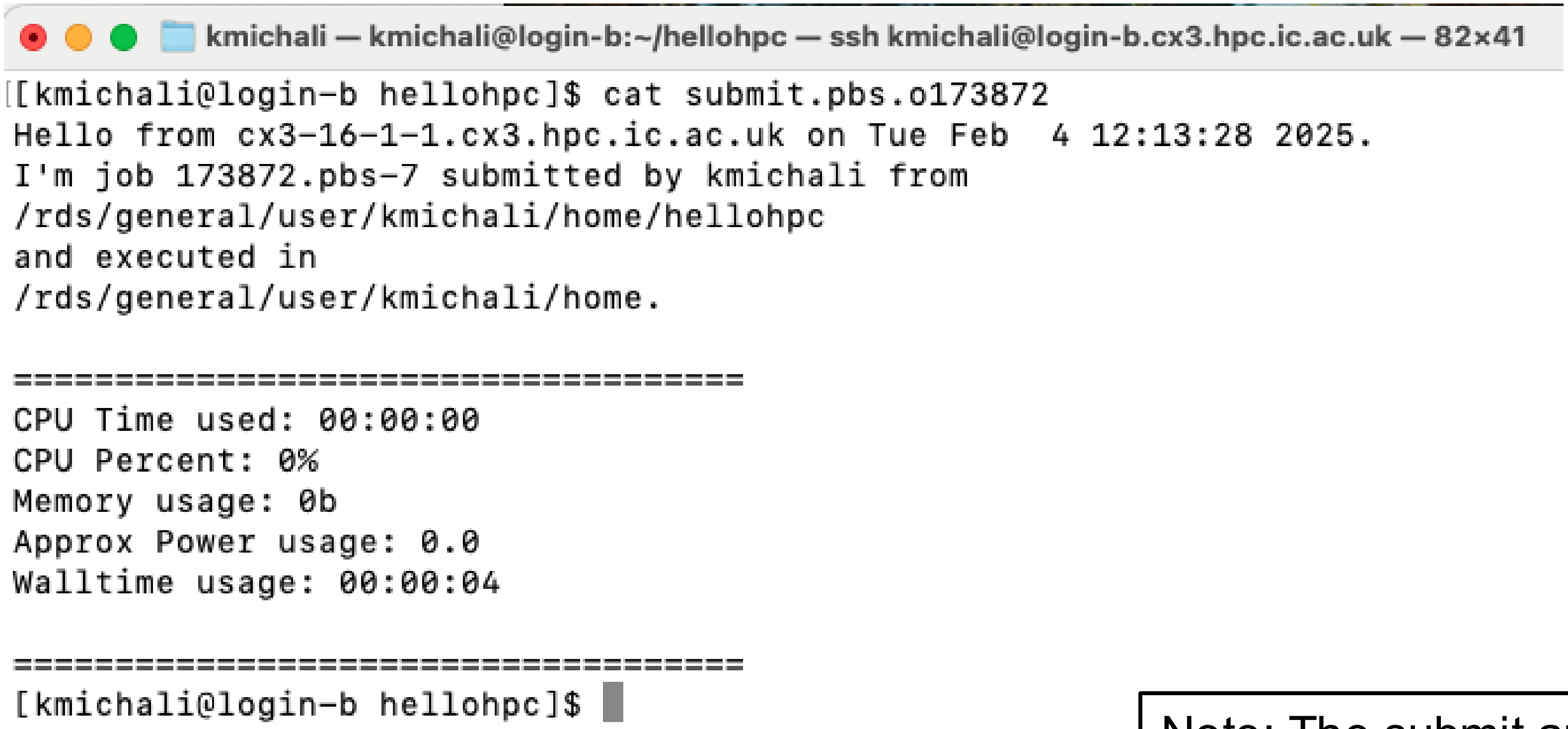
For example, if you place your input file in the same directory as your job script and expect your program to read it in a current directory, the file will not be found.

The next few slides show different ways of managing this situation.

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## Hellohpc output


kmichali — kmichali@login-b:~/hellohpc — ssh kmichali@login-b.cx3.hpc.ic.ac.uk — 82×41

```
[kmichali@login-b hellohpc]$ cat submit.pbs.o173872
Hello from cx3-16-1-1.cx3.hpc.ic.ac.uk on Tue Feb  4 12:13:28 2025.
I'm job 173872.pbs-7 submitted by kmichali from
/rds/general/user/kmichali/home/hellohpc
and executed in
/rds/general/user/kmichali/home.

====================================
CPU Time used: 00:00:00
CPU Percent: 0%
Memory usage: 0b
Approx Power usage: 0.0
Walltime usage: 00:00:04

====================================
[kmichali@login-b hellohpc]$
```

Note: The submit and execute directories are different

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Job script with input and output files

This will not work because the default current directory for a job is not the same as the directory where the job was submitted (and where you expect to find your input file).

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

ml myprog

# input and output will not be found
# the current dir changes when the job is running

myprog input output
```

## Environment variable pointing to your submit directory

**$PBS_O_WORKDIR**

- will exist when your job is running
- this is where you submitted your job
- usually a directory where your job script resides
- you can use this variable in your script

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Job script for with $PBS_O_WORKDIR

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

ml myprog


cd $PBS_O_WORKDIR
myprog input output
```

## Helloworld with an output file

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml Python

cd $PBS_O_WORKDIR
python $PBS_O_WORKDIR/hellohpc.py > mylog
```

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
    - working remotely and login
    - software management on the cluster
    - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism – array jobs
- parallel programs

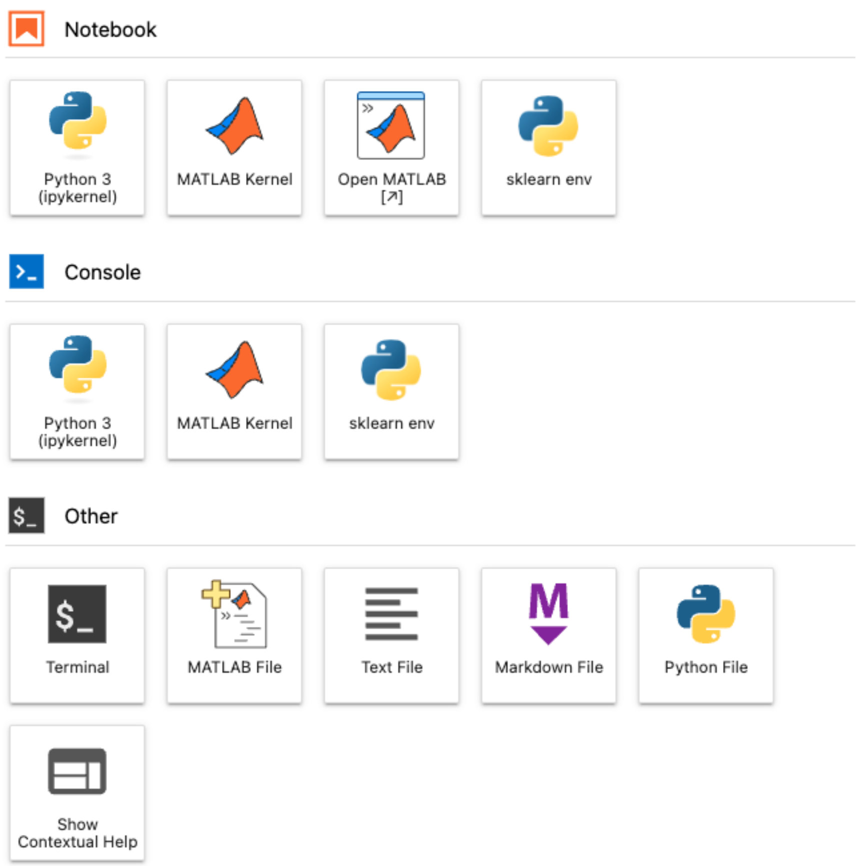
## Interactive resources

- It is possible to use the HPC cluster via:
  - Jupyter Notebooks
  - RStudio

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

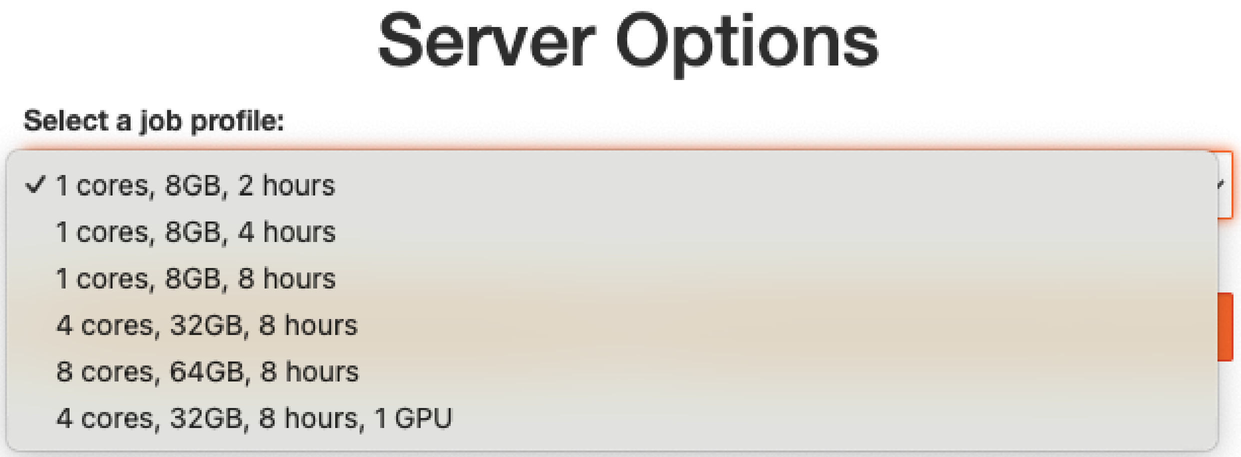
## Jupyter Lab https://jupyterhub-11.rcs.ic.ac.uk/



Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Jupyter lab options


Server Options

Select a job profile:

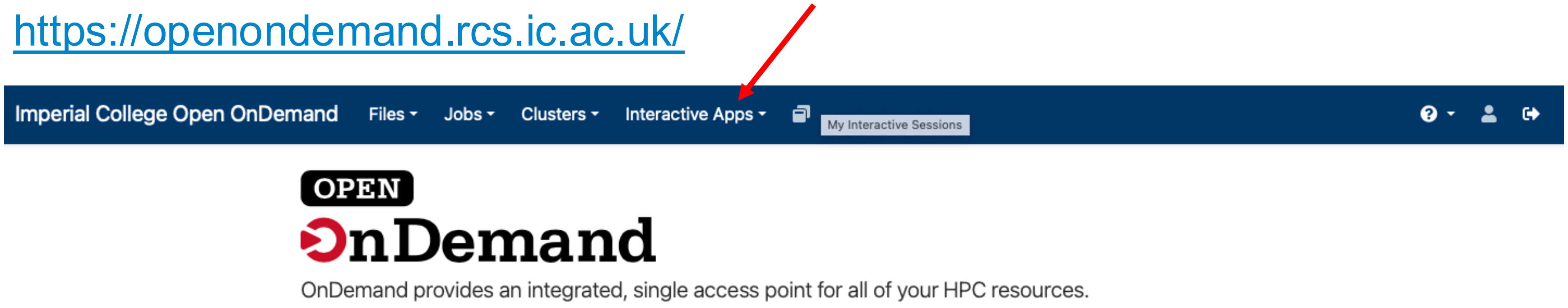
✓ 1 cores, 8GB, 2 hours
1 cores, 8GB, 4 hours
1 cores, 8GB, 8 hours
4 cores, 32GB, 8 hours
8 cores, 64GB, 8 hours
4 cores, 32GB, 8 hours, 1 GPU

Imperial College London

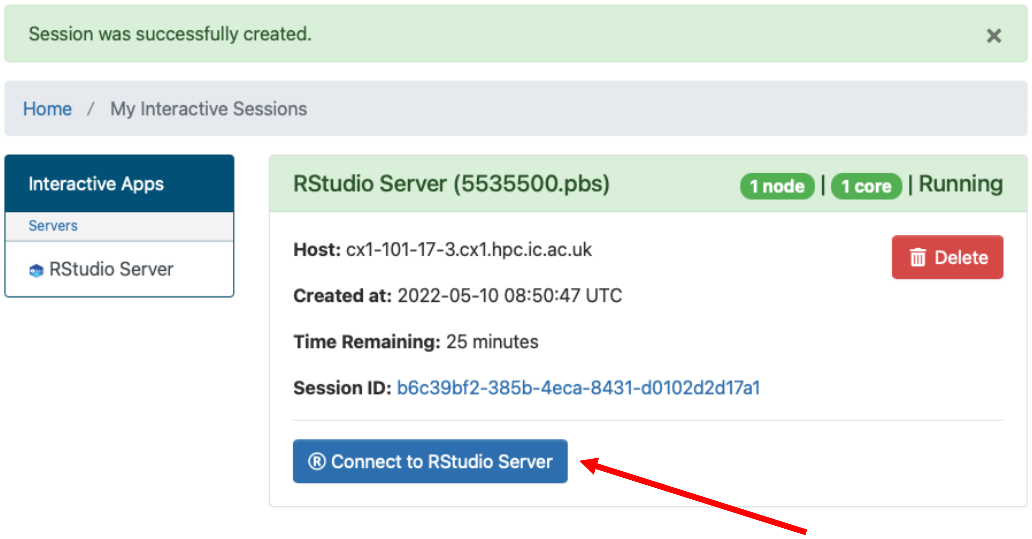
RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## RStudio through Open OnDemand

https://openondemand.rcs.ic.ac.uk/



OnDemand provides an integrated, single access point for all of your HPC resources.



Session was successfully created.

RStudio Server (5535500.pbs) | Running

**Host:** cx1-101-17-3.cx1.hpc.ic.ac.uk

**Created at:** 2022-05-10 08:50:47 UTC

**Time Remaining:** 25 minutes

**Session ID:** b6c39bf2-385b-4eca-8431-d0102d2d17a1

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
  - working remotely and login
  - software management on the cluster
  - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism – array jobs
- parallel programs

## Parallel computing

Data parallelism (array jobs)

- concurrent processing of data on an HPC cluster

Parallel programs

- parallelism is implemented on code level

Imperial College London

RCS | RESEARCH COMPUTING SERVICE
Gateway to Discovery

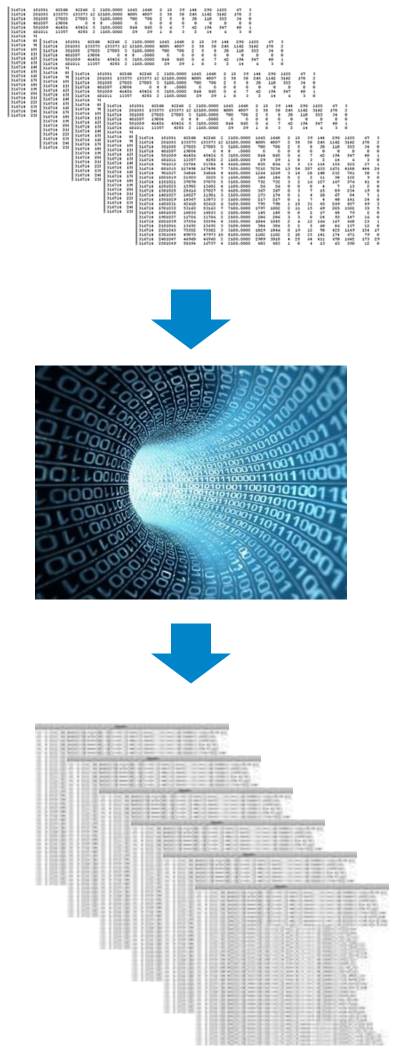
## Data parallelism

Data parallelism examples:

- processing multiple files
- processing large files that can be split
- simulation replicates
- parameter sweeps

Individual cases have to be independent:

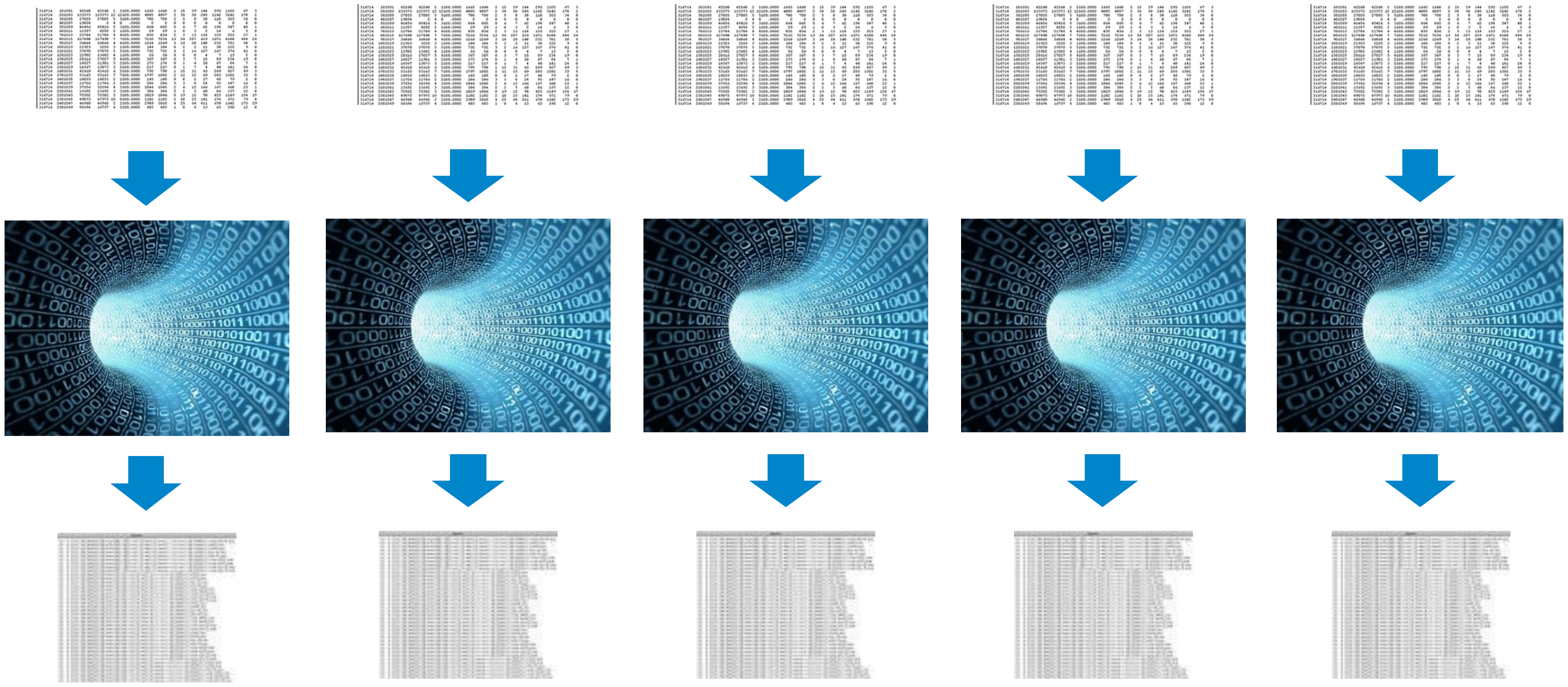
- they do not exchange data
- input for any case doesn't depend on output of the other



Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Data parallelism – array jobs



## Advantages of data parallelism

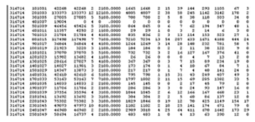
- jobs run concurrently (in semi-parallel fashion)
- relatively easy to implement
- jobs run independently when a resource becomes available; no need to wait until a whole compute node is available (program-level parallel jobs would have to)

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Single job (before parallelisation)

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb


ml myprog

myprog input output
```

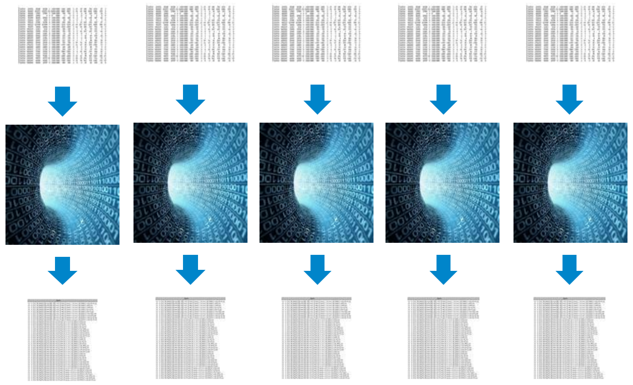


Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

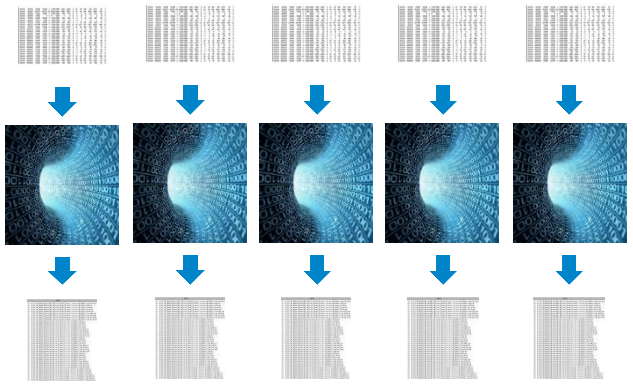
## Parallelisation - array job script

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-5


ml myprog


myprog  input? output?
```



## Parallelisation - array job script

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS –J 1-5


ml myprog


myprog  input? output?
```



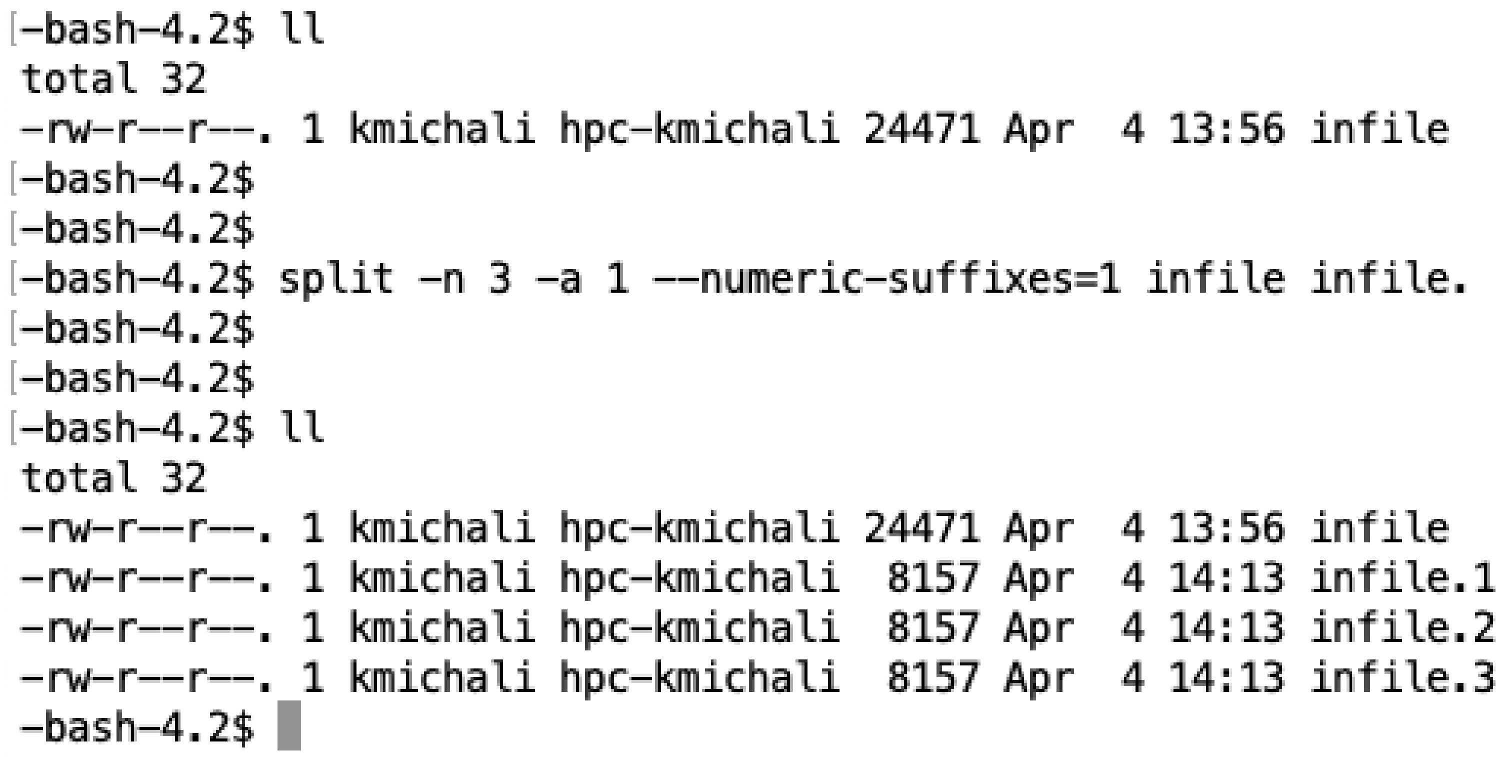
**$PBS_ARRAY_INDEX** takes values from 1 to 5 (or any other value specified by –J)

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

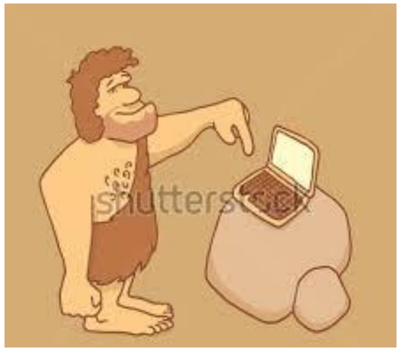
## Array job scenario 1 – numbered files

Task: **run filter.py on infile;** BUT the file is too big and takes a long time to process.
Solution: Split infile to smaller chunks and process them concurrently.



```
-bash-4.2$ ll
total 32
-rw-r--r--. 1 kmichali hpc-kmichali 24471 Apr  4 13:56 infile
-bash-4.2$
-bash-4.2$
-bash-4.2$ split -n 3 -a 1 --numeric-suffixes=1 infile infile.
-bash-4.2$
-bash-4.2$
-bash-4.2$ ll
total 32
-rw-r--r--. 1 kmichali hpc-kmichali 24471 Apr  4 13:56 infile
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.1
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.2
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.3
-bash-4.2$
```

Imperial College
London



## Hands-on – array jobs

Download and run array run case scenario 1:

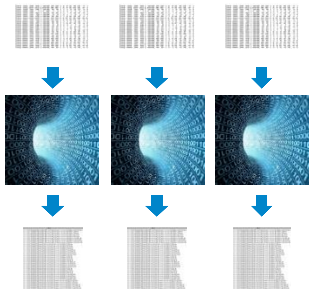
```
git clone https://github.com/kmichali/hellohpc.git
cd array/filter
qsub submit_filter_array.pbs
```

## Array job script – numbered files (before parallelisation)

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -N class

ml tools/prod
ml Python

cd $PBS_O_WORKDIR
python filter.py infile outfile
```

## Array job script – numbered files

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-3

ml tools/prod
ml Python

cd $PBS_O_WORKDIR
python filter.py infile.$PBS_ARRAY_INDEX outfile.$PBS_ARRAY_INDEX
```



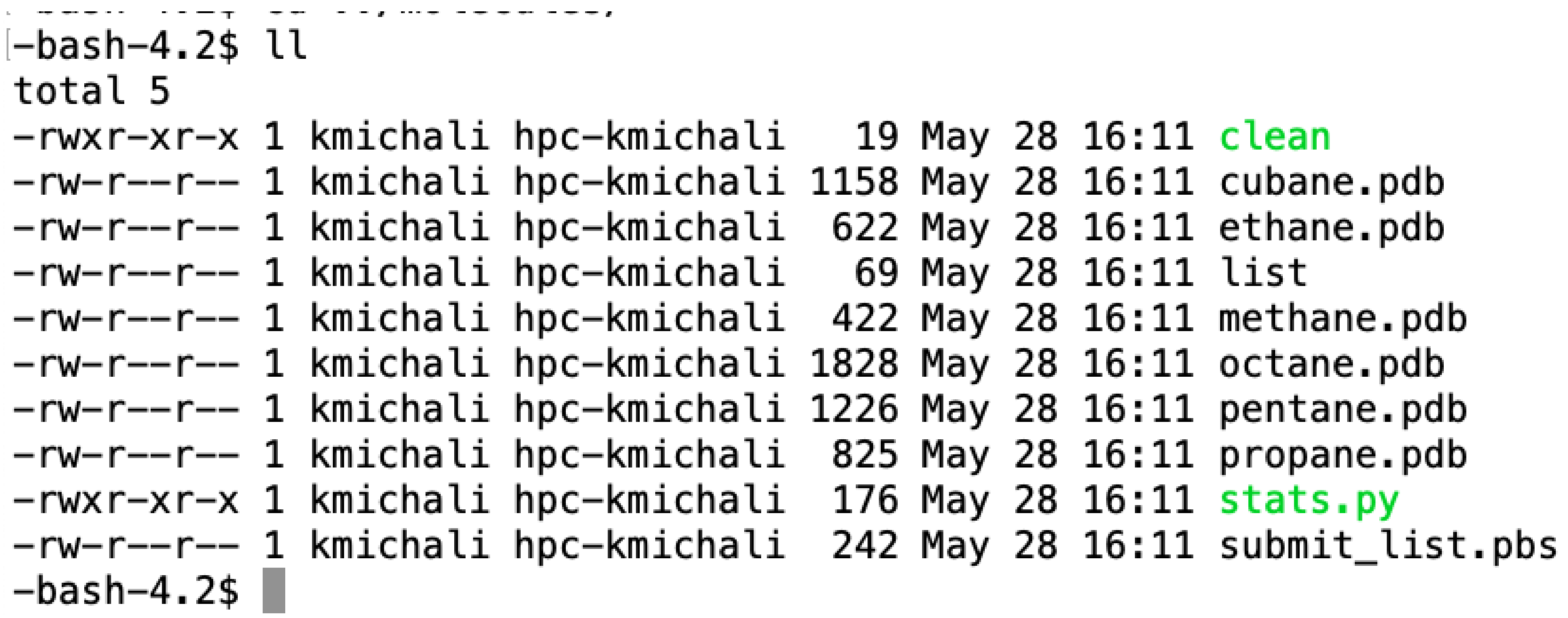
$PBS_ARRAY_INDEX takes values from 1 to 3 (or any other value specified by –J)

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
  - working remotely and login
  - software management on the cluster
  - copying files to and from the cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism – array jobs 2
- parallel programs

## Array job scenario 2 – file list

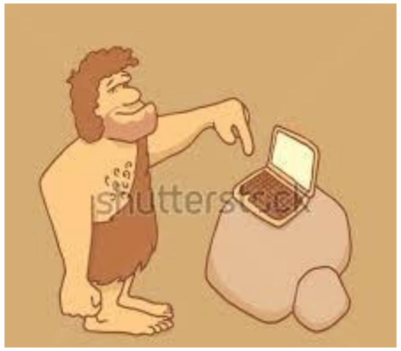
Task: **run stats.py on many molecules;** there are too many and it takes too long.
Solution: Process the molecules concurrently on the HPC cluster.



```
-bash-4.2$ ll
total 5
-rwxr-xr-x 1 kmichali hpc-kmichali   19 May 28 16:11 clean
-rw-r--r-- 1 kmichali hpc-kmichali 1158 May 28 16:11 cubane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali  622 May 28 16:11 ethane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali   69 May 28 16:11 list
-rw-r--r-- 1 kmichali hpc-kmichali  422 May 28 16:11 methane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali 1828 May 28 16:11 octane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali 1226 May 28 16:11 pentane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali  825 May 28 16:11 propane.pdb
-rwxr-xr-x 1 kmichali hpc-kmichali  176 May 28 16:11 stats.py
-rw-r--r-- 1 kmichali hpc-kmichali  242 May 28 16:11 submit_list.pbs
-bash-4.2$
```

Imperial College London

## Hands-on – array jobs



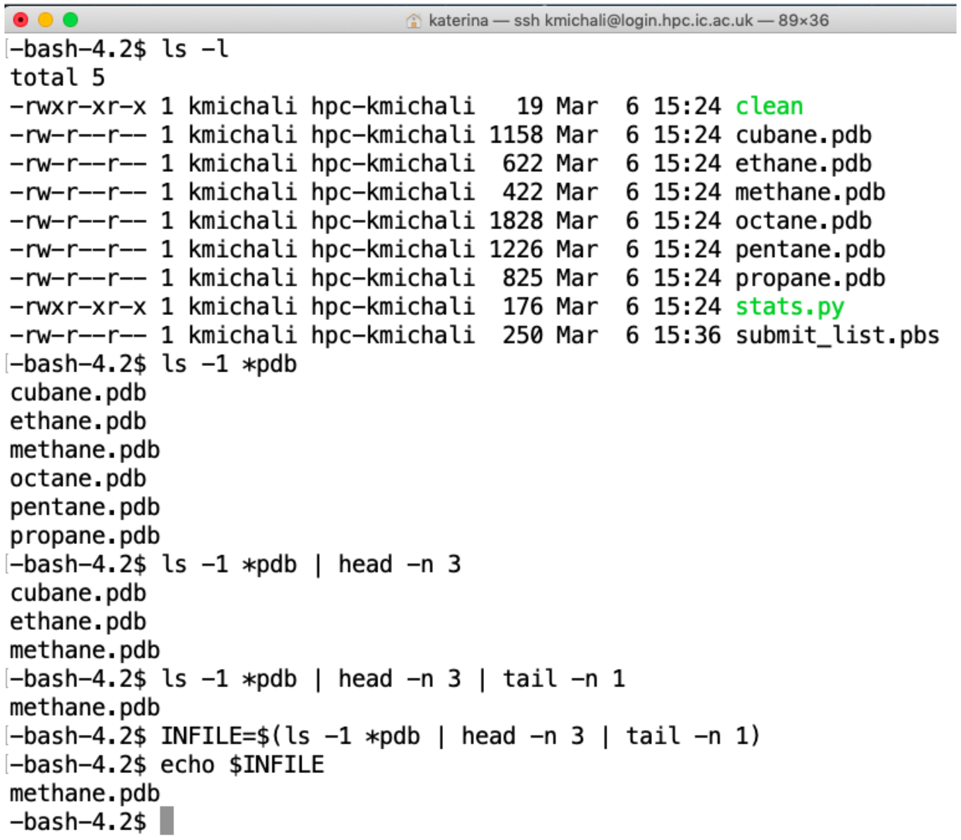
Download and run array run case scenario 2:

```
git clone https://github.com/kmichali/hellohpc.git
cd array/molecules
qsub submit_list.pbs
```

## Array job script for one file (before parallelisation)

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb

ml tools/prod
ml Python

cd $PBS_O_WORKDIR

INFILE=cubane.pdb
python stats.py $INFILE ${INFILE}.out
```

## Leveraging $PBS_ARRAY_INDEX to choose input



katerina — ssh kmichali@login.hpc.ic.ac.uk — 89×36

```
-bash-4.2$ ls -l
total 5
-rwxr-xr-x 1 kmichali hpc-kmichali   19 Mar  6 15:24 clean
-rw-r--r-- 1 kmichali hpc-kmichali 1158 Mar  6 15:24 cubane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali  622 Mar  6 15:24 ethane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali  422 Mar  6 15:24 methane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali 1828 Mar  6 15:24 octane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali 1226 Mar  6 15:24 pentane.pdb
-rw-r--r-- 1 kmichali hpc-kmichali  825 Mar  6 15:24 propane.pdb
-rwxr-xr-x 1 kmichali hpc-kmichali  176 Mar  6 15:24 stats.py
-rw-r--r-- 1 kmichali hpc-kmichali  250 Mar  6 15:36 submit_list.pbs
-bash-4.2$ ls -1 *pdb
cubane.pdb
ethane.pdb
methane.pdb
octane.pdb
pentane.pdb
propane.pdb
-bash-4.2$ ls -1 *pdb | head -n 3
cubane.pdb
ethane.pdb
methane.pdb
-bash-4.2$ ls -1 *pdb | head -n 3 | tail -n 1
methane.pdb
-bash-4.2$ INFILE=$(ls -1 *pdb | head -n 3 | tail -n 1)
-bash-4.2$ echo $INFILE
methane.pdb
-bash-4.2$ 
```

The screen shows Linux commands that list the input files, pick the n-th file from the list, and store the file name in a variable.

These commands can be used in an array run job script to process a list of files.

The example picks the third file from the list and stores it in a variable called $INFILE.

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

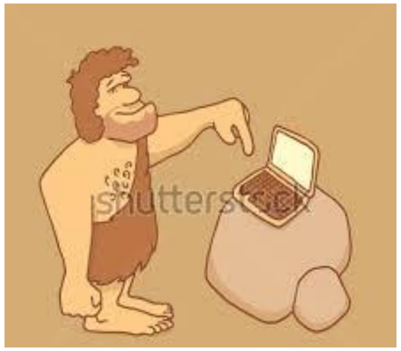
## Array job script – file list

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-6

ml tools/prod
ml Python

cd $PBS_O_WORKDIR

INFILE=$(ls -1 *pdb | head -n $PBS_ARRAY_INDEX | tail -n 1)
python stats.py $INFILE ${INFILE}.out
```

## Array job scenario 3 – for loops and matrices

- if you are running simulation replicates or parameter sweep, your program might contain a very time-consuming for loop.
- If the loop iterations are independent, there is a good chance that you can deploy it as an array run.
- The following Python example uses $PBS_ARRAY_INDEX value directly in the Python code.
- The code iterates through a matrix and uses individual elements for a calculation. In the original code, the calculations happen in serial manner.

Imperial College
London



## Hands-on – array jobs

Download and run array run case scenario 3:

```
git clone https://github.com/kmichali/hellohpc.git
cd array/matrix
qsub submit_matrix.pbs
```

## Array job scenario 3 – for loops

Task: Iterate though a large matrix and use each element for a calculation using an array run.

Solution:

The following Python example uses $PBS_ARRAY_INDEX value directly in the Python code.

The code uses $PBS_ARRAY_INDEX to execute the correct iteration of the loop that processes just one element of the matrix.

All array runs combined process all elements of the matrix in parallel.

## Python template for matrix iteration

```
input = np.loadtxt(”matrix.txt", dtype='f′,
delimiter=',′)
counter = 0

for i in range(0,input.shape[0]):
    for j in range(0, input.shape[1]):
       print('processing element', input[i,j])
       # your code for processing goes in here
```

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Python template for matrix iteration with array run

```
input = np.loadtxt(”matrix.txt", dtype='f’,
delimiter=',’)
counter = 0
array_index = int(os.environ['PBS_ARRAY_INDEX'])

for i in range(0,input.shape[0]):
    for j in range(0, input.shape[1]):
        counter = counter+1
        if counter == array_index:
            print('processing element', input[i,j])
            # your code goes in here
```

## Submit script for 3x3 matrix

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-9


ml tools/prod
ml Python

cd $PBS_O_WORKDIR

python matrix_array.py
```

## Caution - beware of too many files in one dir!

- HPC parallel file systems are sensitive to a large number of small files in one directory. The filesystem may slow down significantly in these circumstances and this will affect all users.

- If your array run is likely to produce > 10K files in one directory, you must adapt your script so it does not happen.

- In $PBS_O_WORKDIR, make directories for your logs. For example: ./out_logs and ./err_logs. In your script, redirect logs with:

```
#PBS -o ./out_logs -e ./err_logs
```

- Writing of your output files can be redirected to subdirectories. One possibility is to use $PBS_ARRAY_INDEX variable to name each subdirectory.

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Managing log and output files for arrayruns

- This example uses PBS directives to redirect logs and makes a separate directory for results for each element of the array (do not forget to make directories for logs before you run a script):

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -o ./out_logs -e ./err_logs
#PBS -J 1-6

ml tools/prod Python
cd $PBS_O_WORKDIR

#make a directory for your results
mkdir $PBS_ARRAY_INDEX

#run python script
python myscript.py input.$PBS_ARRAY_INDEX PBS_ARRAY_INDEX/output
```

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs on the cluster
  - OpenMP
  - MPI
  - python

## Parallel programs

So far, we have implemented parallelism using array runs on the cluster; the software that was used remained unchanged.

The next section is about parallelism that is achieved by writing parallel code.

Two types of parallel code:

- OpenMP or multiprocessing – requires shared memory
- Message Passing Interface – on multiple compute nodes at once

## Parallel codes

- As it has become difficult and expensive to produce faster and faster processors, the focus has shifted in combining the power of multiple CPUs to work on a single task. We supports two main parallel programming paradigms.

- **OpenMP (C,C++,Fortran) and multiprocessing (Python)**

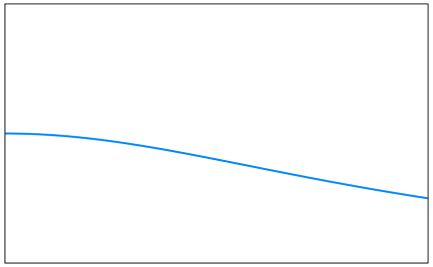
  OpenMP jobs require shared memory. The parallelization is achieved by using compiler directives around individual blocks of code, often loops or functions. Converting your serial program into OpenMP is "relatively" easier than attempting MPI programming.

- **Message Passing Interface (C, C++,Fortran, Python)**

  MPI jobs do not require shared memory and can be executed on separate cluster nodes. MPI standard consists of library routines that create parallel processes and pass necessary data (messages) between them. The messages are passed across the network if needed.

## Real problem - approximate Pi with numerical integration

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

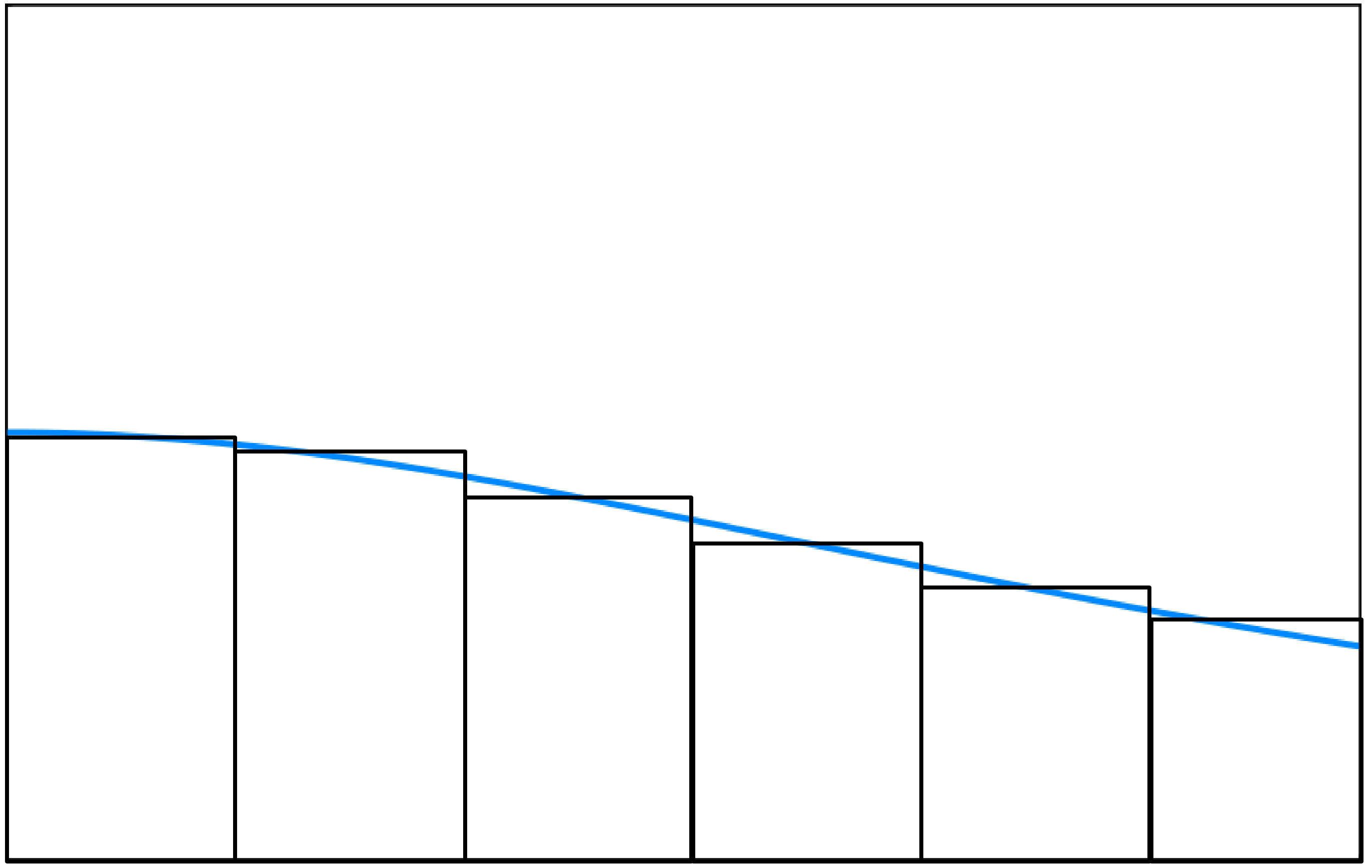


The answer becomes more accurate with increasing N

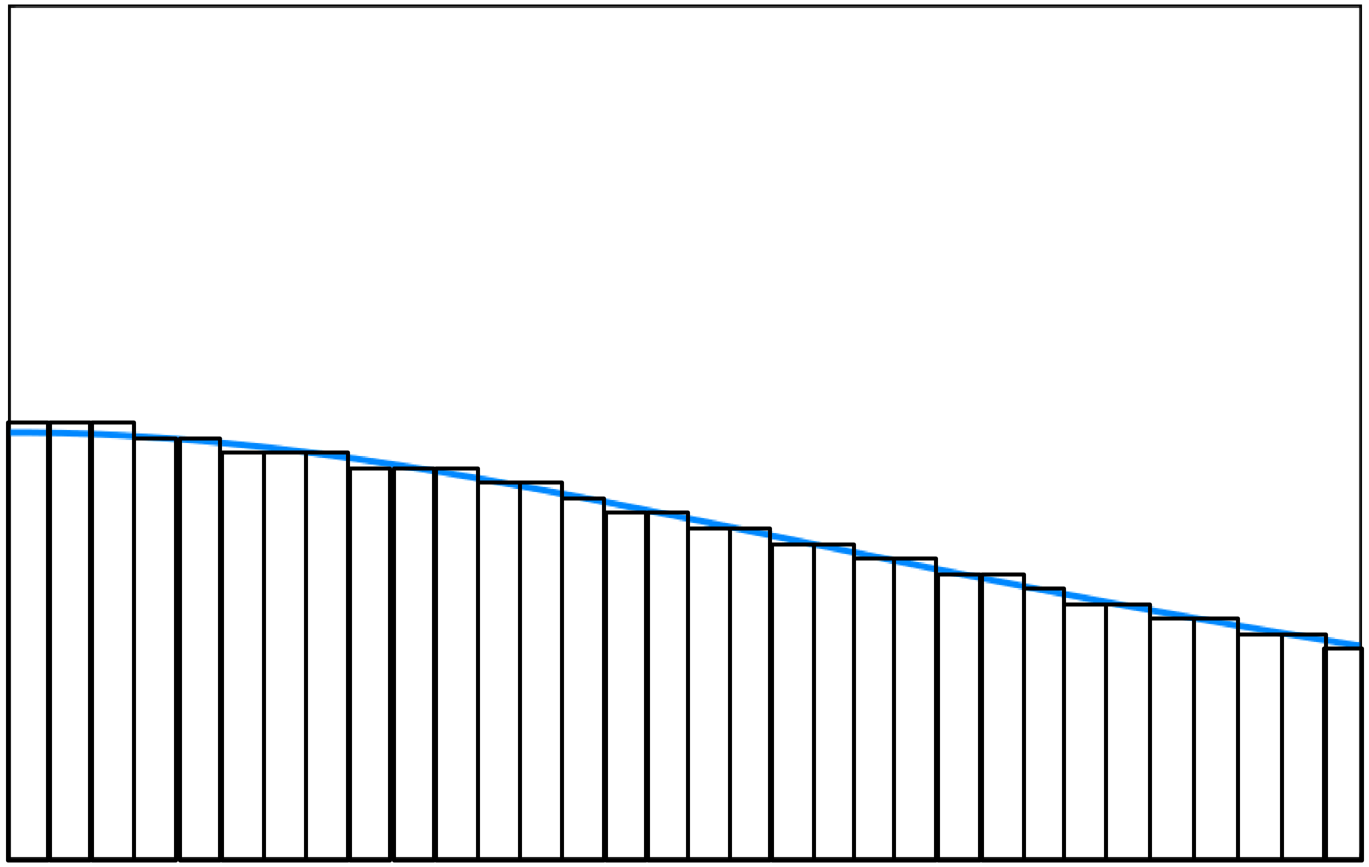
Iterations over N are independent, the calculation can be parallelised

We'll discuss pi code examples using serial, OpenMP, MPI and various python approaches.

## Numerical integration



## Numerical integration



Imperial College London

## Serial Pi

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

```
program pi_serial

implicit none
integer(KIND=8)  :: n, i
double precision :: w, x, sum, pi, a, start, finish
double precision :: duration, timef
n=1E10
start=timef()

w=1.0/n
sum=0.0

do i=1,n
   x=w*(i-0.5)
   sum=sum+4.0/(1.0+x*x)
end do

pi=w*sum
finish=timef()
duration=finish-start
print*,n,"  ",pi,"  ",duration
end
```

Imperial College London

## Hands-on – get codes, compile and run

```
#clone repository
git clone https://github.com/kmichali/hellohpc.git

# go to repository
cd parallel

# load compiler, parallel MPI library and compile
bash compile.sh

#run serial pi code
cd serial
qsub pi_serial.pbs
```

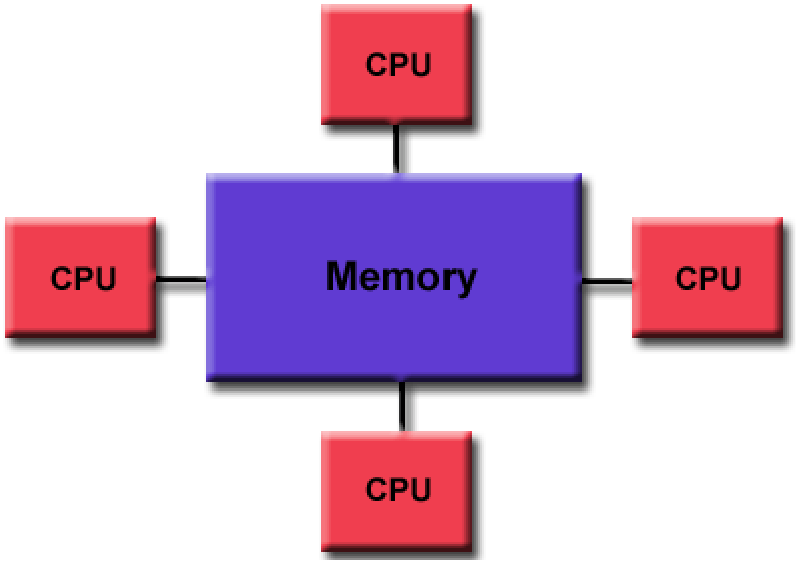
## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs on the cluster
    - OpenMP
    - MPI
    - python

## Parallel programming paradigms – OpenMP

**OpenMP** (open multiprocessing)

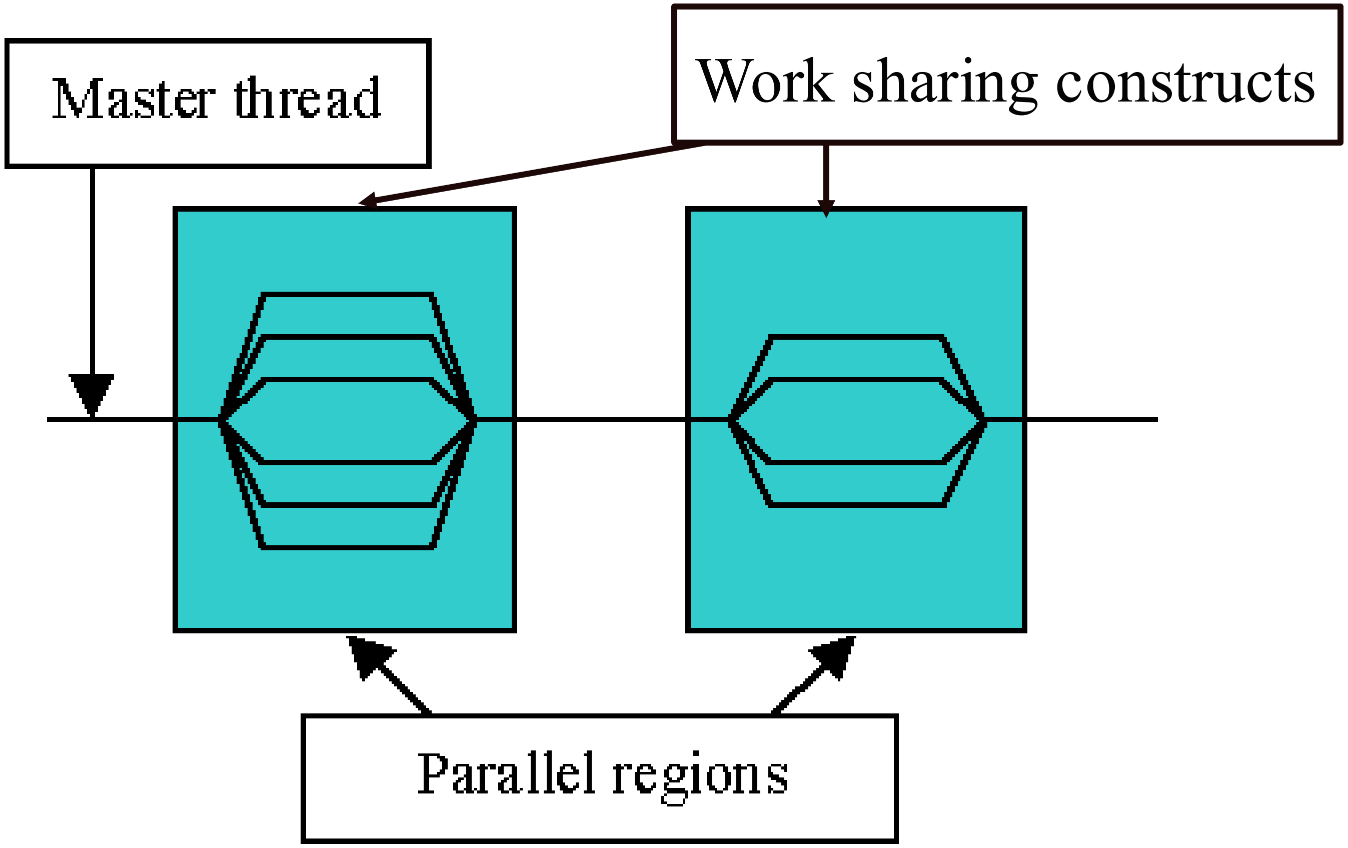
- set of extensions (compiler pragmas and API calls) to provide Fortran/C/C++ with the ability to run certain parts of the code in parallel, without explicitly managing (creating, destroying, assigning) threads
- requires shared memory



Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## OpenMP



## Hello world in C and OpenMP

```
#include <omp.h>
#include <stdio.h>
main (int argc, char *argv[])
{

  #pragma omp parallel
   {
       printf("Hello world!");
   }
}
```

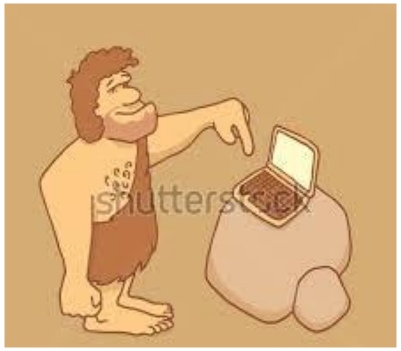
Imperial College London

RESEARCH COMPUTING SERVICE
Gateway to Discovery

## OpenMP submit script

```
#PBS -l walltime=01:00:00
## Use 1 compute node with 8 cores and 4gb of memory
#PBS -l select=1:ncpus=8:mem=4gb:ompthreads=8

ml GCC

$PBS_O_WORKDIR/hello_openmp
```

Imperial College London



## Hands-on – calculate Pi with OpenMP code

```
#run openMP pi code
cd ../openmp
qsub pi_openmp.pbs
```

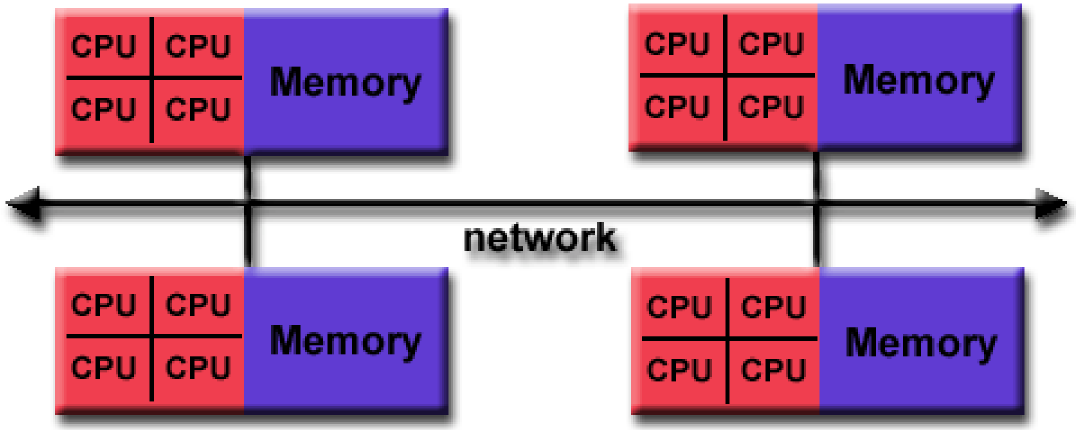
3.141592653589793238462643

## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs on the cluster
  - OpenMP
  - MPI
  - python

## Message Passing Interface - MPI

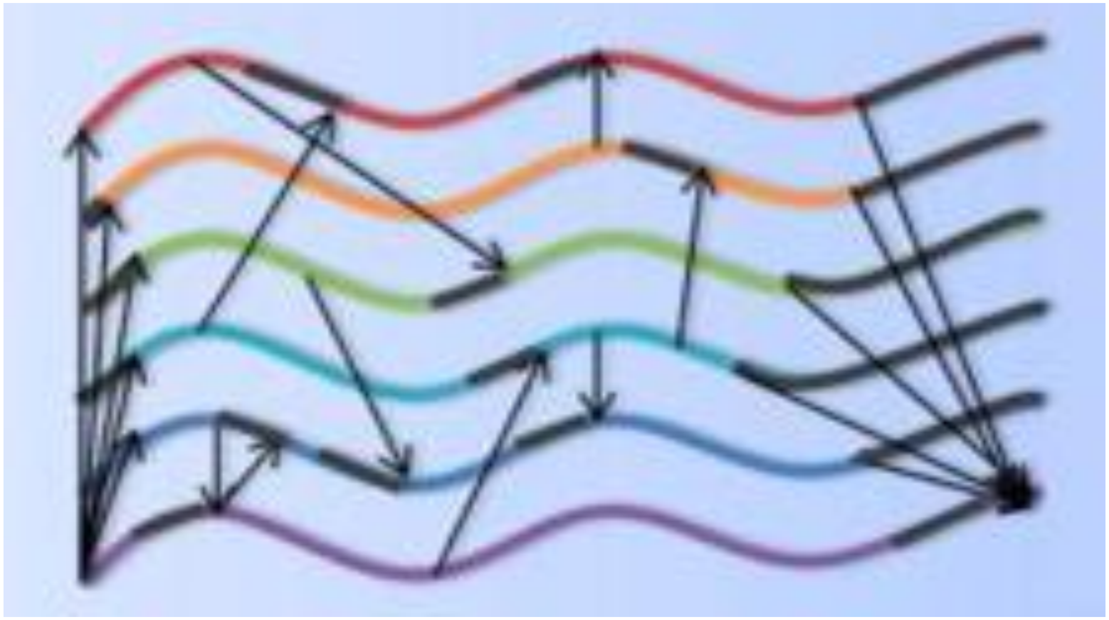
- message passing parallel programming model
- one program is deployed on multiple nodes with distributed memory
- messages (data) are sent over the network
- MPI is a specification for a library with different implementations (Open MPI, Intel, MVAPICH)

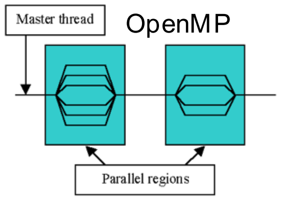


Imperial College London

RCS | RESEARCH COMPUTING SERVICE
Gateway to Discovery

## MPI program





Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## Hello world in C and MPI

```
int main(int argc,char **argv) {
  MPI_Comm comm;
  int nprocs,procid;

  MPI_Init(&argc,&argv);
  comm = MPI_COMM_WORLD;
  MPI_Comm_size(comm,&nprocs);
  MPI_Comm_rank(comm,&procid);

  int name_length = 100;
  char proc_name[name_length];
  MPI_Get_processor_name(proc_name,&name_length);


  printf("Hello from process %d out of %d running on %s\n",
       procid, nprocs, proc_name);


  MPI_Finalize();
}
```

## MPI job submit script

```
#PBS -l walltime=01:00:00
## Use 2 nodes with 24 cores each and 4 gb of memory per node.
#PBS -l select=2:ncpus=24:mem=4gb:mpiprocs=24

ml GCC OpenMPI

mpiexec $PBS_O_WORKDIR/hello_mpi
```

Imperial College
London

## Hands-on 4 – calculate Pi with MPI code

```
#run MPI pi code
cd ../mpi
qsub pi_mpi.pbs
```

# Introduction to HPC at Imperial

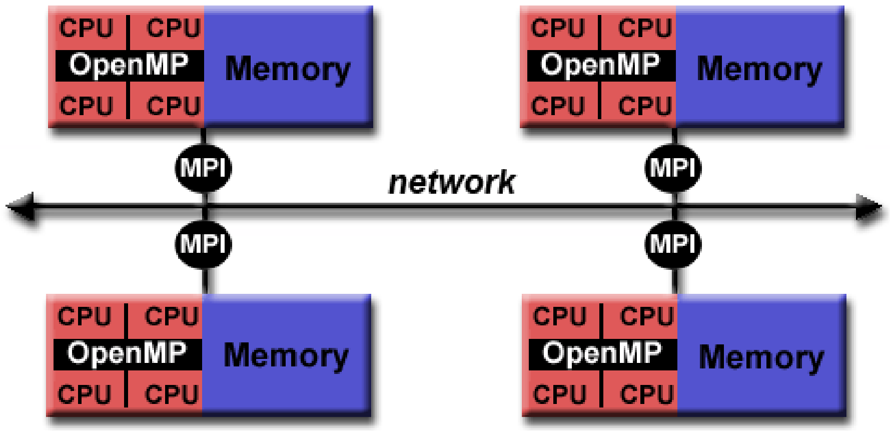
- introduction, key links and contacts
- preparing to use the HPC cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs on the cluster
  - OpenMP
  - MPI
  - python

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Hybrid jobs

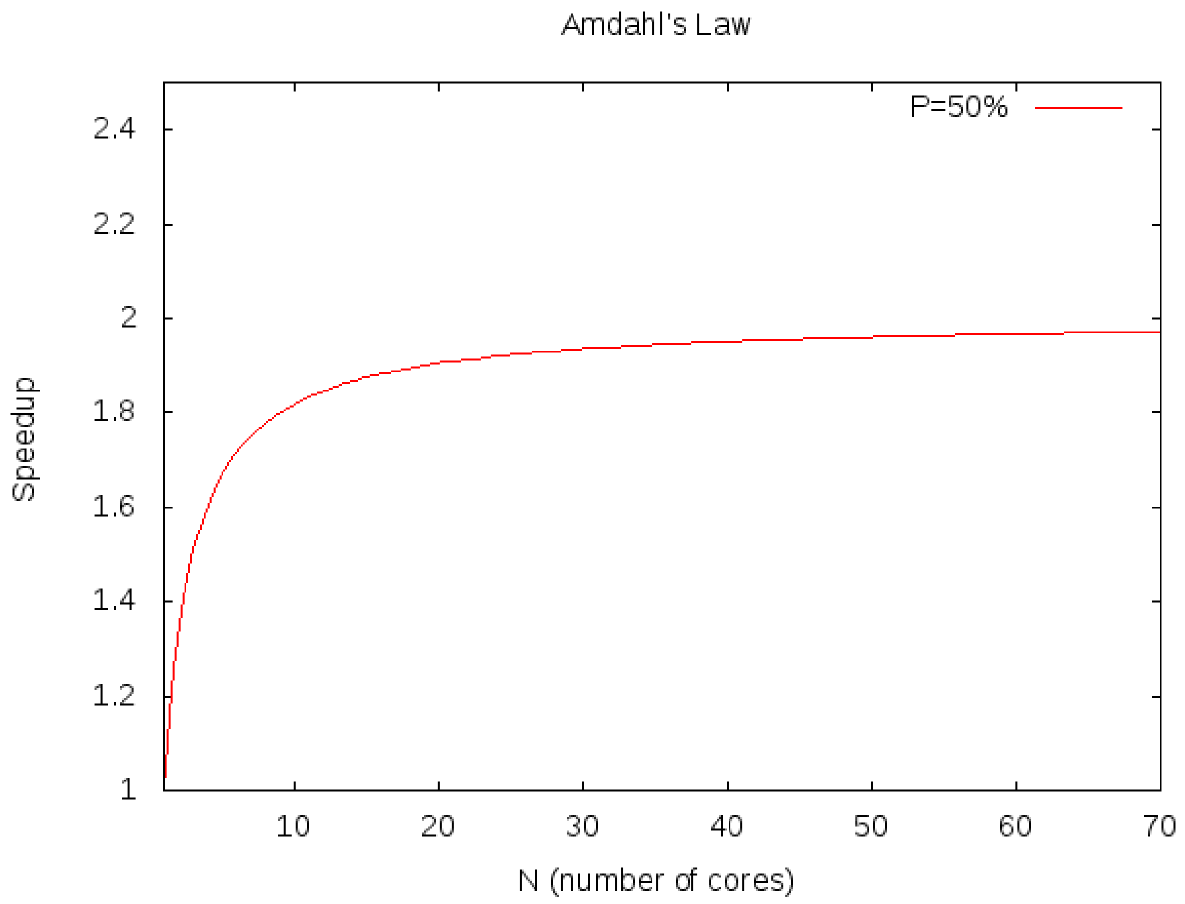
- hybrid code combines MPI with OpenMP
- MPI provides distributed memory parallelism
- OpenMP provides on-node shared memory parallelism
- the model reduces data movement within a node
- way to combine vectorisation and large scale parallel code



## Hybrid submit script

```
#!/bin/sh
#PBS -l walltime=01:00:00
#PBS -l select=2:ncpus=24:mem=4gb:mpiprocs=1:ompthreads=24
## Use 2 nodes with 24 cores each and 4 gb of memory per node.
## Each node will host 1 mpirank and 24 threads.

ml GCC OpenMPI

mpiexec $PBS_O_WORKDIR/myprog
```

## Amdahl’s law


Amdahl's Law
P=50%
Speedup
2.4
2.2
2
1.8
1.6
1.4
1.2
1
10
20
30
40
50
60
70
N (number of cores)

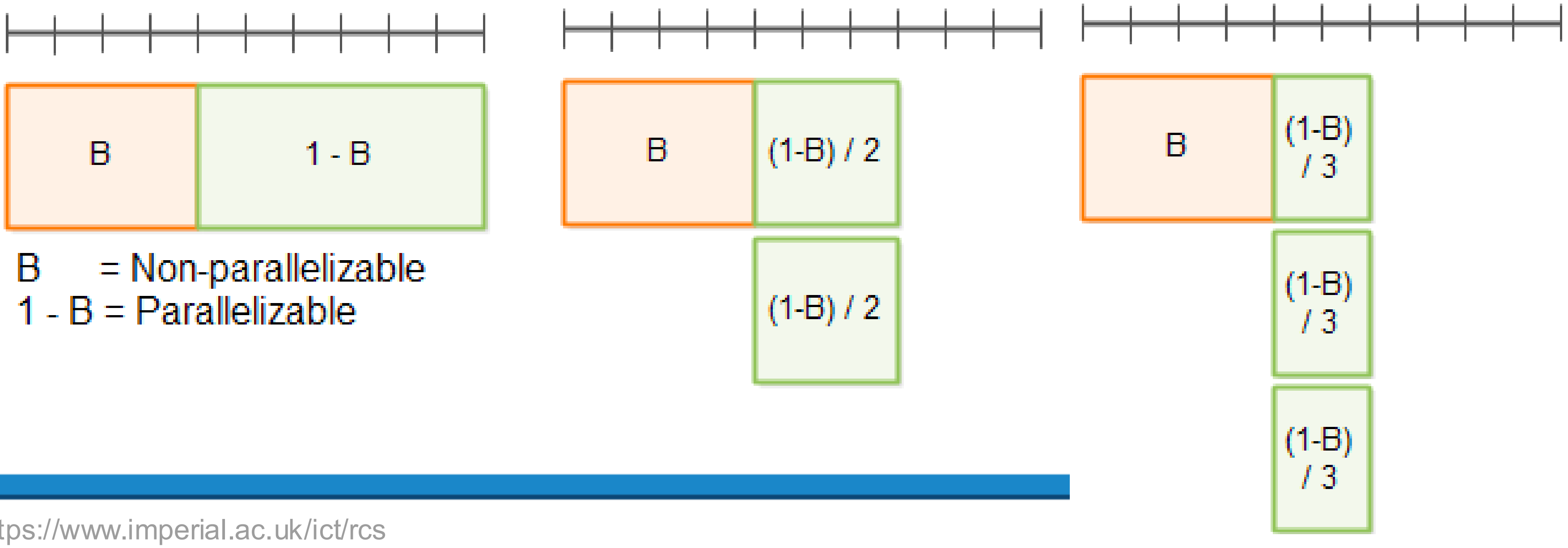
Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Amdahl's law

Amdahl's law states that in parallelization, if 1-B is the proportion of a system or program that can be made parallel, and B is the proportion that remains serial, then the maximum speedup that can be achieved using N number of processors is 1/(B+((1-B)/N)).

If N tends to infinity then the maximum speedup tends to 1/B.



## Introduction to HPC at Imperial

- introduction, key links and contacts
- preparing to use the HPC cluster
- software execution on the cluster
- job parameters and job scripts
- managing input and output files in your script
- interactive resources
- data parallelism
- parallel programs on the cluster
    - OpenMP
    - MPI
    - python

## Parallel python?

- It is possible but not straightforward.
- The usual suspects – multiprocessing and MPI – were not the best way to go.

## Pi code with python

- serial with python interpreter
- multiprocessing with Pool
- python MPI with mpi4py
- serial with pypy3 interpreter
- serial with just in time compiling (numba and jit)
- parallel with numba and jit

Imperial College London

## Hands-on – get codes

```
#clone repository
git clone https://github.com/kmichali/hellohpc.git
cd parallel/python

#you will not be able to run the codes unless you build
python dependencies, installation help files are
included in the repository
```

## Serial pi in Python

```
import time

n = int(10e9)
start_time = time.time()
w = 1.0/n
psum = 0.0

for i in range(1,n+1):
    x = w*(i - 0.5)
    psum = psum+4.0/(1.0 + x*x)

pi=w*psum
duration = time.time() - start_time
print(f"{n:,d}", "   ", pi, "   ", duration)
```

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

## Multiprocessing with Pool

```
import time
import multiprocessing as mp
import functools
import operator

def calc_sum(i):
    psum = 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
w = 1.0/n
start_time = time.time()

pool = mp.Pool(24)
result = pool.map(calc_sum, range(1,n+1))
total = functools.reduce(operator.add, result)
pi=w*total
print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time)
```

## MPI with mpi4py

```
import numpy as np
from mpi4py import MPI

n = int(10e9)

comm = MPI.COMM_WORLD
myrank = comm.Get_rank()
nproc = comm.Get_size()

if myrank == 0:
    start_time = time.time()

start = 1+myrank * int(n/nproc)
finish = (myrank+1) * int(n/nproc)

psum = 0.0
pi = 0.0
w = 1.0/n
```

```
for i in range(start, finish + 1):
    x = w*(i - 0.5)
    psum = psum+4.0/(1.0 + x*x)

psum = np.asarray(psum)
pi = np.asarray(pi)
comm.Reduce(psum, pi, op=MPI.SUM)
pi=w*pi

if myrank == 0:
    duration = time.time() - start_time
    print(n, "   ", pi, "   ", duration)
```

## Numba and jit implemenation

```
import time
from numba import jit

@jit(nopython=True)
def calc_sum(n,w):
    psum = 0.0

    for i in range(1,n+1):
        psum += 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
start_time = time.time()
w = 1.0/n
sum = calc_sum(n,w)
pi=w*sum
print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time
```

## Parallel Numba

```
import time
from numba import jit, prange

@jit(nopython=True, parallel=True)
def calc_sum(n,w):
    psum = 0.0

    for i in prange(1,n+1):
        psum += 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
start_time = time.time()
w = 1.0/n
sum = calc_sum(n,w)
pi=w*sum
print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time)
```

Imperial College London

RCS RESEARCH COMPUTING SERVICE Gateway to Discovery

## Serial and parallel python – pi with 10 billion iterations

- serial Fortran at 16s and 24 core OpenMP at 1.3s

- serial with python interpreter – 40-50 min
- python MPI with mpi4py – 61s (32 cores)
- python multiprocessing with Pool and Queue – does not scale
- modified multiprocessing with Pool – 117s (8 cores)

- serial with pypy3 interpreter – ~2  min
- serial python with just-in-time compiling – numba and jit – ~18s
- python with parallel numba and jit –   ~4s (24 cores)

Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

Thank you!