Introduction to using the CSF

Course materials / exercises / slides available from:
https://ri.itservices.manchester.ac.uk/course/csf-mace

Research Infrastructure Team
IT Services

https://ri.itservices.manchester.ac.uk/csf3/

Today

- How to use the Computational Shared Facility (CSF)
- Investing a little time now learning the basics makes doing your project work easier
- ~2 hours:
  - Slides (some will be "further reading")
  - Practical sessions
  - Time for 1-on-1 Q&As about specific apps at the end
  - But please do ask questions as we progress
  - Course materials are available here:

  https://ri.itservices.manchester.ac.uk/course/csf-mace

Show of Hands

- Has anyone used the CSF already?
- Which apps will you be using for your project?
  - Abaqus
  - Fluent
  - OpenFOAM
  - StarCCM
  - Something else?

- Everyone will benefit from today's course.
- We'll show how to *run* apps on the CSF, but you will need to know (or learn) how to use those apps to achieve your project aims.

https://ri.itservices.manchester.ac.uk/course/csf-mace/

CSF intro: why use the CSF and what it is

WHY & WHAT …

https://ri.itservices.manchester.ac.uk/course/csf-mace/

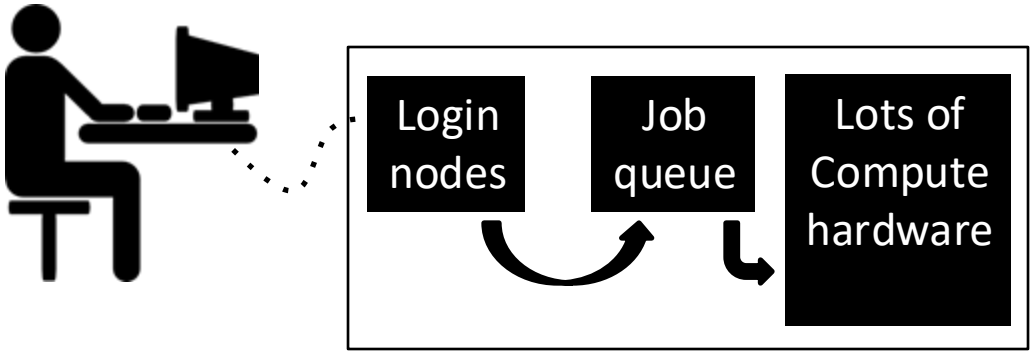
Motivation: Why use the CSF?

- Some (most?) research computation not suitable for your desktop/laptop
  - Takes *too long* to run
  - Needs *more* memory
  - Uses *too much* disk/storage space
- Variety of software applications (200+ already installed)
  - Abaqus, Fluent, StarCCM, OpenFOAM, …
  - python, C/C++/FORTRAN compilers
  - many more, lots of documentation
- Do more, get it done sooner!
  - 1-core (serial) jobs or many-core (parallel) jobs
    - A core is a processing unit within a CPU processor
    - Modern CPUs have several cores: your PC/laptop may have a single CPU with 4,6,8 cores
    - The CSF has a lot more. Many apps can exploit these resources

Do not let the size/capacity/power of your computer dictate the size and complexity of the models/simulations/systems/problems you are solving!

https://ri.itservices.manchester.ac.uk/course/csf-mace/

A new way of working!



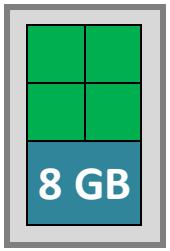
Running on a desktop vs Logging in to the HPC system

- You can fire up a GUI, run an app immediately. BUT:
- Got enough memory, cores, storage?
- Need to keep the PC to yourself (public cluster PC?)
- For several days?!
- Only one "job" (simulation, analysis) at a time?

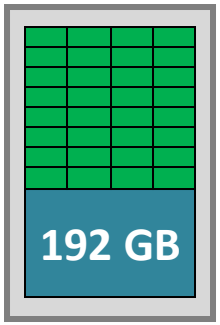
- Submit "**jobs**" to the queue
- Jobs wait until selected to be run
- Jobs run on *high-end* hardware (lots of cores, memory, disk)
- Jobs run safely for days
- Many jobs can run at once
- Can log out any time (jobs still run)
- Log in to check on progress, get results

HPC Example: Finite Element Analysis

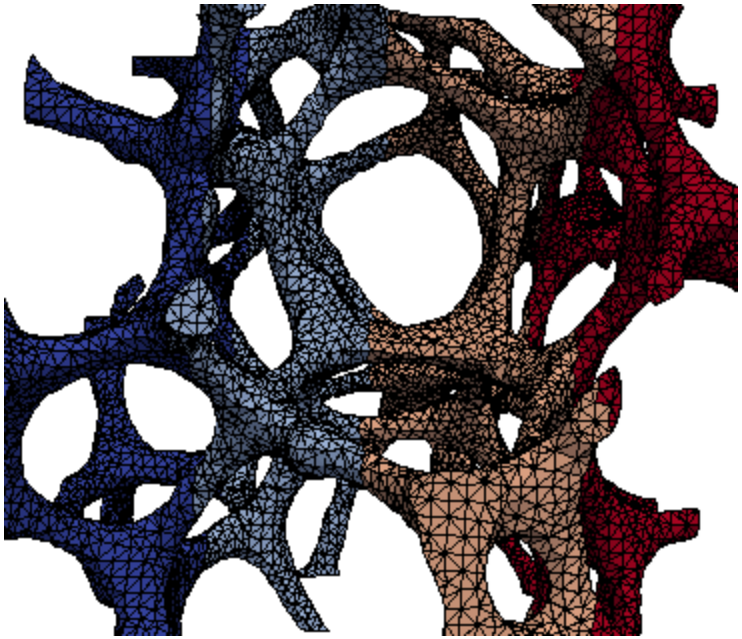
- Perform stress analysis on 3D mesh
  - The app splits the *input* into chunks
  - It performs calculations on chunks, in *parallel*
    - Faster and/or larger problem size



Desktop: 4 cores, 8GB RAM.
8 days to complete

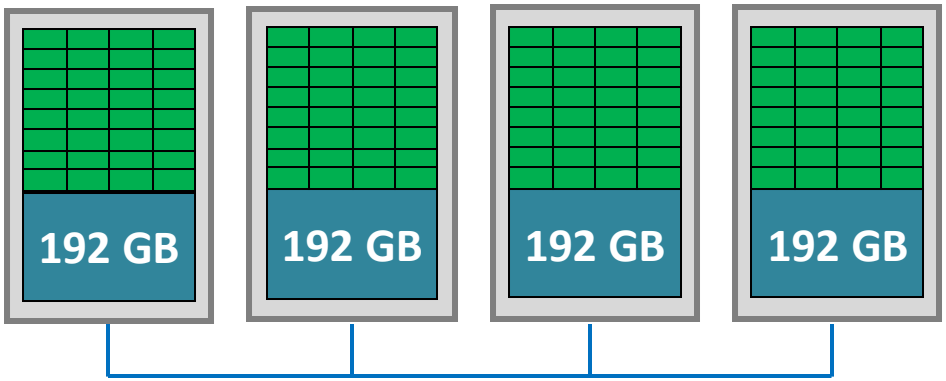


Single HPC *compute node*: 32 cores, 192GB RAM
~2 days to complete



Source: Professor Paul Mummery (MACE) using ParaFEM software http://parafem.org.uk

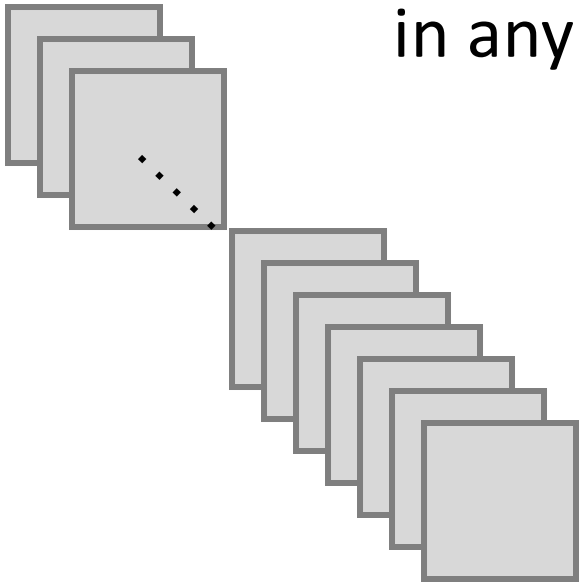
Some apps can use more than one *compute node*



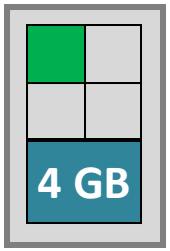
Multiple HPC *compute nodes*: 128 cores, 768GB RAM
~0.5 days to complete

Example: Image Analysis

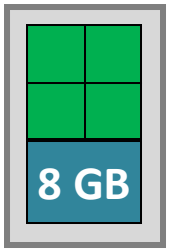
- High *Throughput* Computing
  - Not all apps do "HPC" / parallel
  - Example: you have *lots* of dataset
  - Each image takes 1hr to process (and are independent - process in any order)



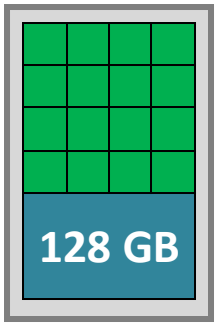
Example: 10,000 image scans to be analysed by an image processing application. Each image takes 1 hour to process.



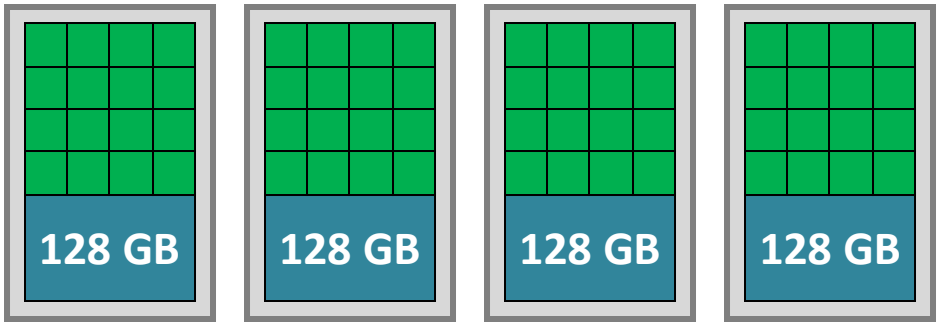
Laptop: 1 copy of software running.
Over 1 year to complete!!



Desktop: 4 cores, 4 copies of software running.
~100 days to complete!



Single HPC compute node: 16 copies of software running.
~26 days to complete



Multiple HPC compute nodes: 64 copies of software running.
~6 days to complete

Some pictures of the CSF

https://ri.itservices.manchester.ac.uk/course/csf-mace/

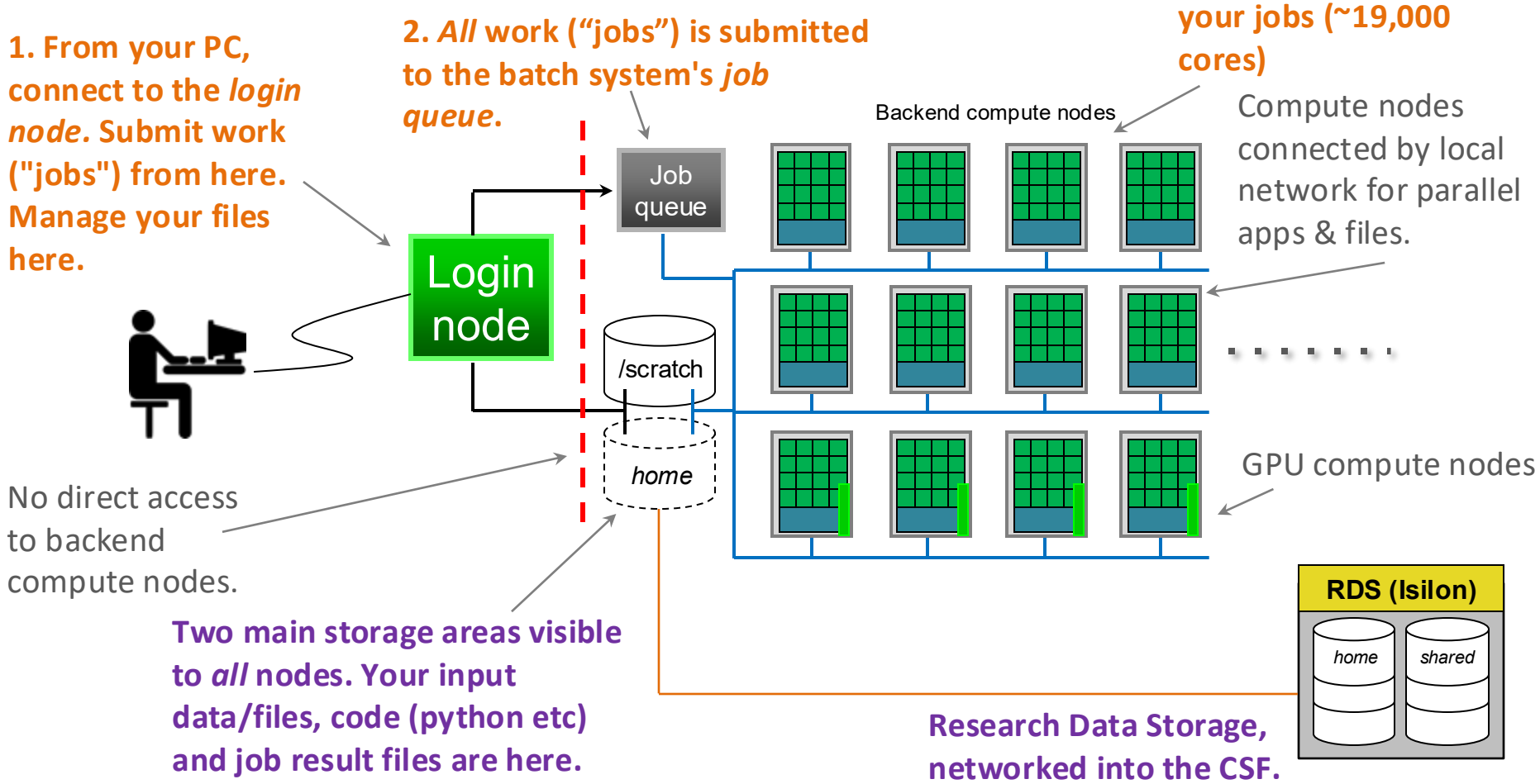
More technical info

OS, logging in, security, linux

DETAILS…

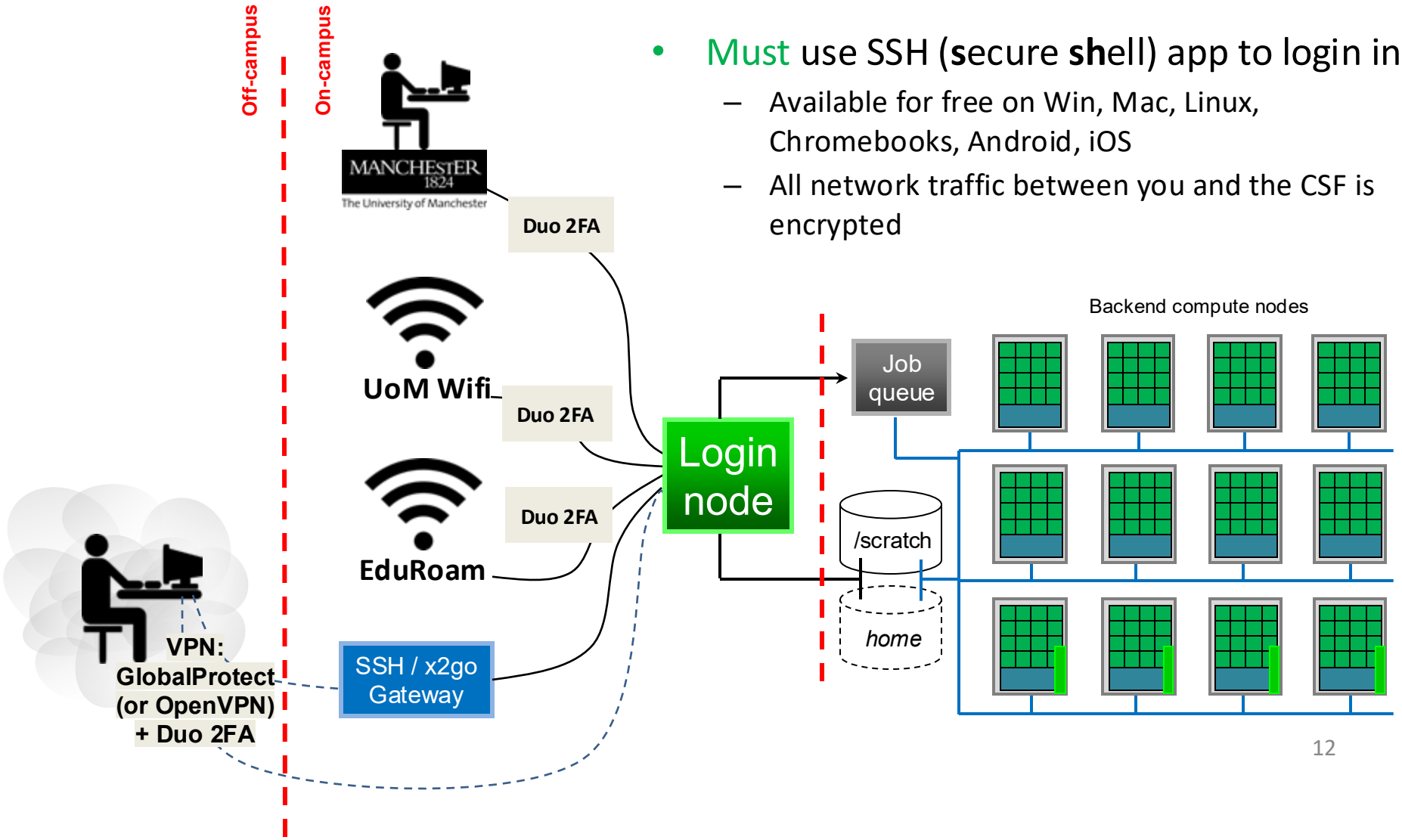
What is the CSF? (more details)

- **C**omputational **S**hared **F**acility
- A *batch* compute *cluster* to run your "jobs" (simulations, analysis,...)
- Here are the main components you'll learn about:



Where can I log in from?

- Must use SSH (**s**ecure **sh**ell) app to login in
  - Available for free on Win, Mac, Linux, Chromebooks, Android, iOS
  - All network traffic between you and the CSF is encrypted



Login Overview

- We'll soon do exercise 1 to login to the CSF
  - The exercise sheet provides all of the steps, including app installs
  - There are also reference slides here giving all of the details
- Access the CSF from a PC / laptop using a Secure Shell (SSH) app
  - Sometimes called a "terminal"
  - There's no web-based or other fancy GUI on the CSF – use the Linux "command line"
- In summary, on **your** PC/laptop:
  - Windows users will install a free terminal+ssh app called MobaXterm
  - Mac users will install a free app named XQuartz, then use the built-in Terminal app and ssh command
  - Linux users will use the built-in terminal and ssh command

- In all cases you will need:
  - Your UoM IT username (like *mabcxyz1*, NOT your email address!)
  - Your UoM IT password (same as used for email, blackboard etc.)
  - Your DUO 2FA device (most likely your mobile phone.)

- Everyone will use the address of the CSF:
  `csf3.itservices.manchester.ac.uk`

DUO 2FA (when on-campus)

- Note: When on-campus, after you enter your password during CSF login, a*ll* login methods will then ask you to do DUO 2FA:

```
Duo two-factor login for mabcxyz1

Enter a passcode or select one of the
following options:

 1. Duo Push to +XX XXXX XX7890

Passcode or option (1-1): 1
Success. Logging you in...

(the Message of the Day is now displayed)

[mabcxyz1@login1[csf3] ~]$
```

Type **1** (and press Enter ↵) in your ssh app to generate a DUO *push* to your device.

Then accept the push on your device.

You are now logged in :-)

What you see once you've log in

- The CSF uses Rocky Linux (c.f. Red Hat EL)
  - Command line – requires the input of commands, can be a little scary at first to new users
  - A welcome *Message of the Day* - announcements
  - The system awaits input/commands from you at a *prompt* (after you've *logged in*):

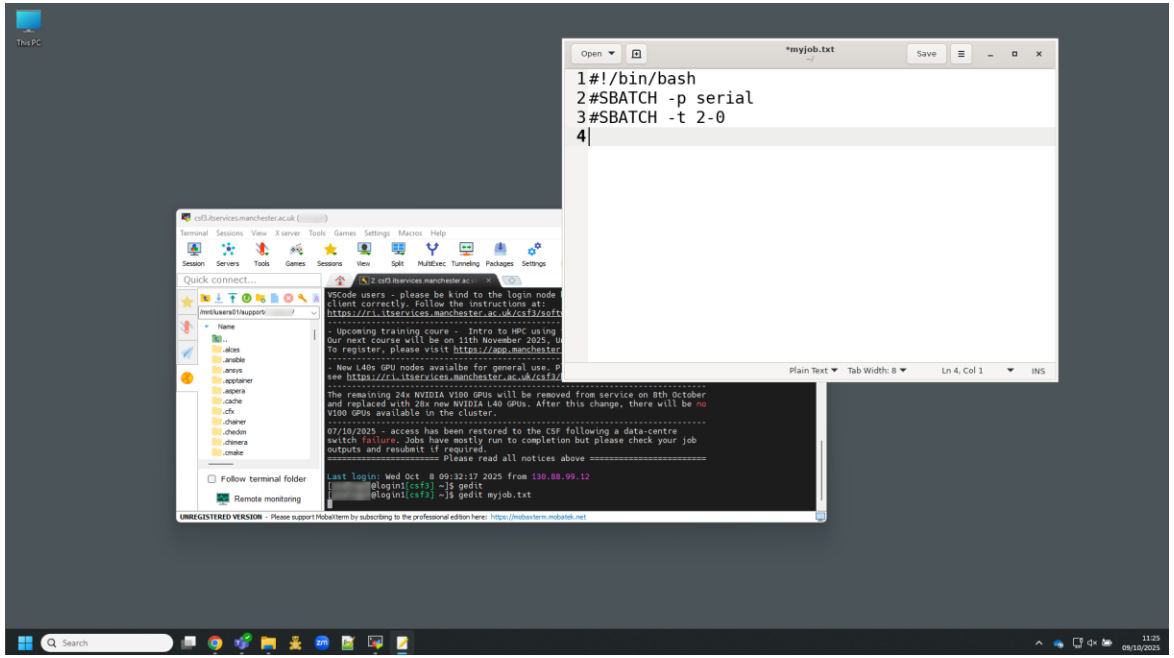
```
[username@login1[csf3] ~]$
[username@login2[csf3] ~]$
[username@login3[csf3] ~]$
```

Type Linux commands at "the prompt"

- Learning Linux commands (more later):
  - https://www.chm.bris.ac.uk/unix/

Displaying a GUI (1)

- Can use the GUI of simple apps such as text editors
  - You'll do this during the exercises
  - Advantage: files written in the CSF's text editor are saved directly on the CSF – no need to upload files
- But don't run the Abaqus / StarCCM / Fluent GUIs
  - Slow and the login nodes are not powerful enough
  - Setup your *simulations* on MACE workstations, *solve* on CSF



Displaying a GUI (2)

- Needs "*Xwindows*" running on *your* PC to run any GUI app, including `gedit`, on the CSF
  - **Windows**: MobaXterm has it and takes care of everything :-)
  - **Linux**: has it, ensure you use `ssh -X` to connect to CSF (UPPERcase X)

  - **Mac**: First, Install X-Quartz (and reboot). Then use `ssh -Y` to connect to the CSF - it automatically starts XQuartz
  - If it doesn't seem to be working, then start XQuartz first:
    - On the dock right click on both the Terminal *and* the big X (x-quartz icon) and quit them.
    - Then start XQuartz.
    - Then start Terminal and reconnect to the CSF.

Security (2)

- It is NOT permitted to share your CSF account
- CSF uses your **IT password** – i.e. same as needed to access UoM email, Canvas and so on ...
  - NEVER share it with ANYONE, including IT staff and your supervisor
  - Forgotten it? You can reset it via the IT Account Manager. Will affect *all* systems that require it.
    - https://iam.manchester.ac.uk/
- Reminder: Other general safety measures
  - Install a virus scanner https://www.itservices.manchester.ac.uk/cybersecurity/advice/virusprotection/
  - By aware of phishing emails https://www.itservices.manchester.ac.uk/cybersecurity/advice/phishing/

Exercise 1 – Connect to the CSF

- In this practical everyone will:
  - Login to the CSF
  - Run basic Linux commands on the CSF login node
  - Start a GUI text-editor on the CSF login node
    - Used in later exercises to write "jobscripts"
  - Does everyone know their IT Username (this is NOT your email) and IT password? Ask us...
  - You also need to be able to do DUO 2FA (got your mobile?)

If doing these exercises again after the course from OFF CAMPUS you MUST have the University GlobalProtect VPN installed and running BEFORE you start this exercise.

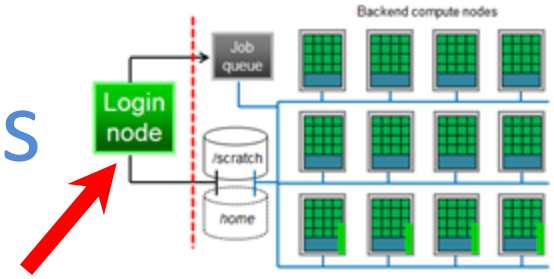
https://ri.itservices.manchester.ac.uk/course/csf-mace/csf-mace-ex1.pdf

Exercise 1 – Logging-in Reference Slides (step by step screenshots)

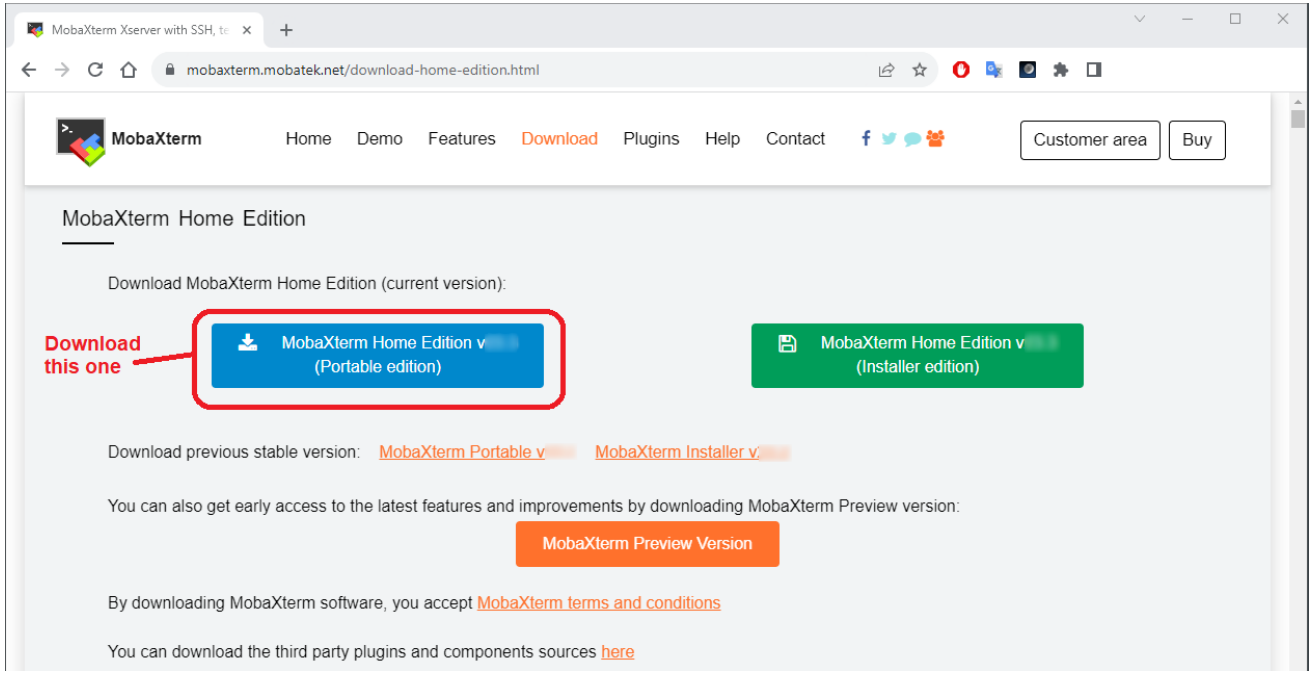
We'll not present these slides – you can refer to them if you need them.

See also the exercise sheet for necessary steps

Connect to CSF from Windows



- **Windows users** need to install a free *terminal* app called **MobaXterm**
- https://mobaxterm.mobatek.net/download-home-edition.html
  the Home edition (portable edition) does *not require* Administrator rights - just *extract* the small .zip file in your P-Drive or Downloads or USB stick for example.



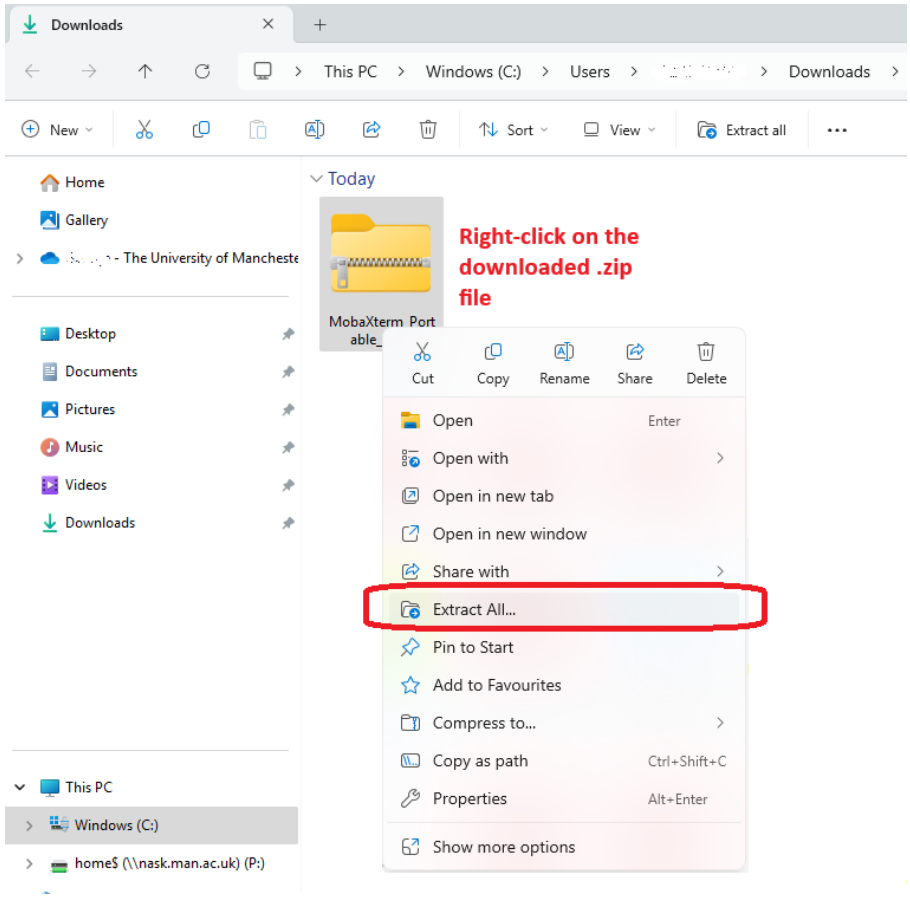
1. Download using the blue box.
2. Once downloaded, *right-click* on the .zip file and select:

   "**Extract all** …"

   This will *unpack* the .zip file to a folder.

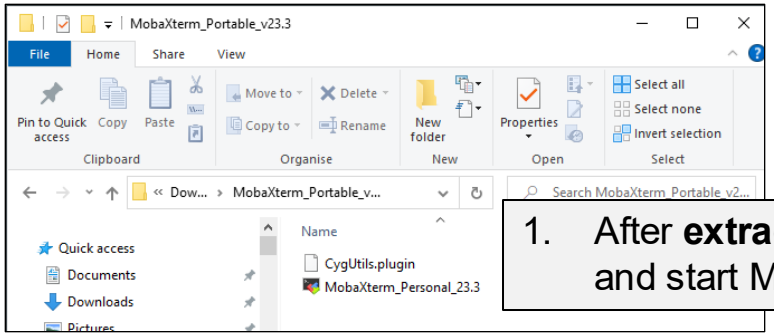
Install the MobaXterm app on your laptop/PC

- You **must** right-click on the downloaded .zip file, then "Extract all..."
- Simple double-clicking on the .zip file to browse the contents does NOT install it correctly.

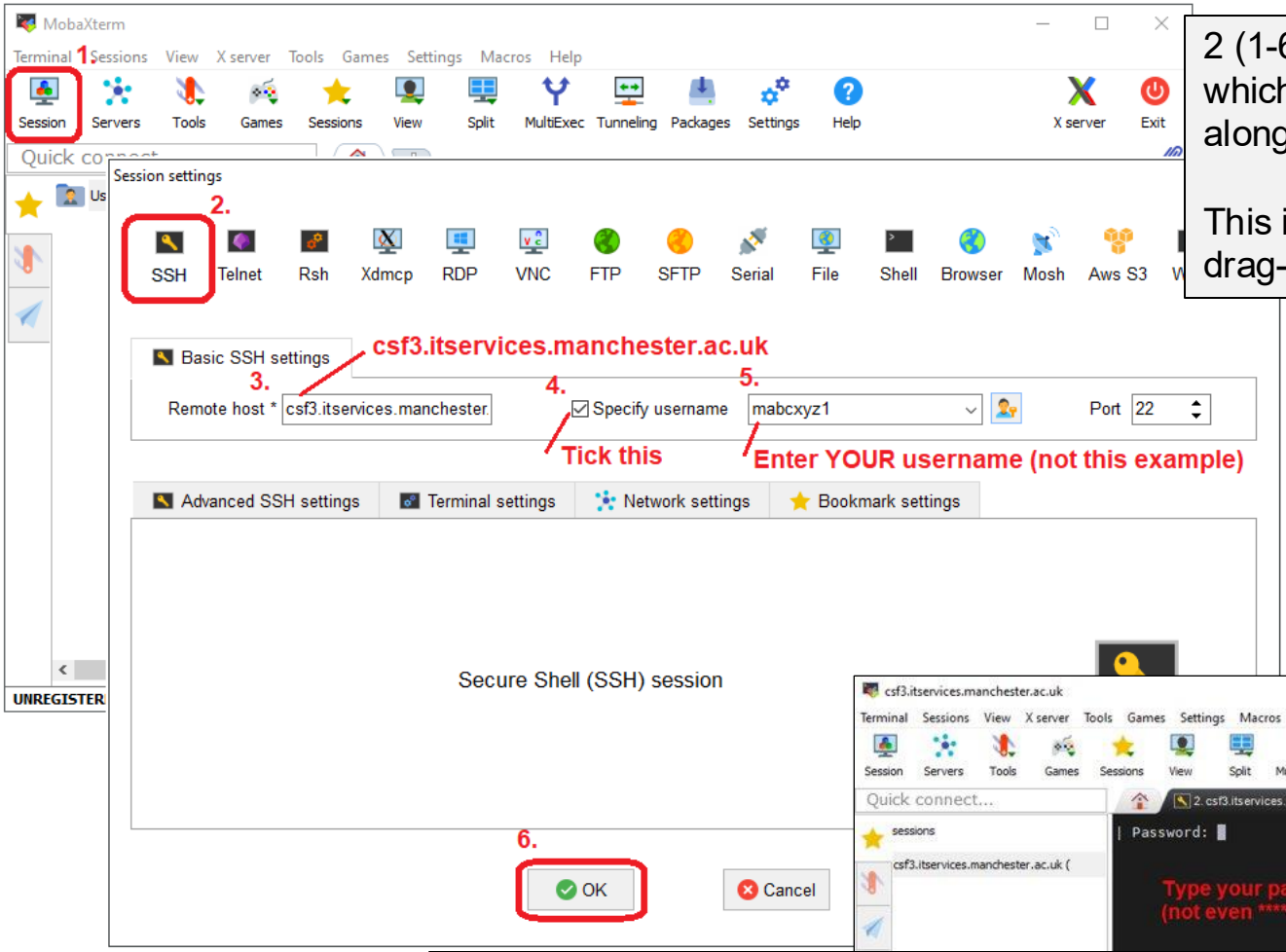


MobaXterm “Session”

(username saved in the session setup)



1. After **extracting** the .zip file, go into the MobaXterm_Personal_v*xy.z* folder and start MobaXterm_Personal_*xy.z* (double-click on the icon)



2 (1-6). Create a "Session" which saves the CSF's details along with *your* **username**.

This is needed to make file drag-n-drop work (see later.)

3. This will then start to log you into the CSF – it will ask for your password. Type carefully!

4. See slide about 2FA – you may be asked for DUO after your password

If asked to save your password, we recommend you say "No", for security.

Drag-n-drop file browser for upload / download

(new users won't have as many items in the list!)

We're on (one of) the CSF login nodes. Any commands you use will be typed "at the prompt", which shows your username and current directory (folder.)

csf3.itservices.manchester.ac.uk ([redacted])

Terminal Sessions View X server Tools Games Settings Macros Help

Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit

Quick connect...

3. csf3.itservices.manchester.ac.uk

/mnt/iusers01/support/[redacted]

Name

.. .alces .ansible .ansys .apptainer .aspera .cache .cfx .chainer .checkm .chimera .cmake .compucell3d_py3 .comsol .comsol_old_173471 .conda .conda.csf3 .conda.old .config .continuum .cpan .cpan-ignore2 .cpanm .cst-workdir .cst2012 .cupy .cytoscape .dart .dbus .dcm2nii

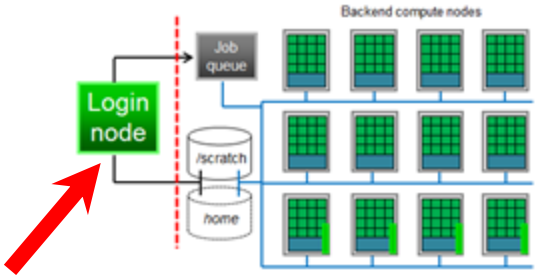
Follow terminal

Remote monitoring

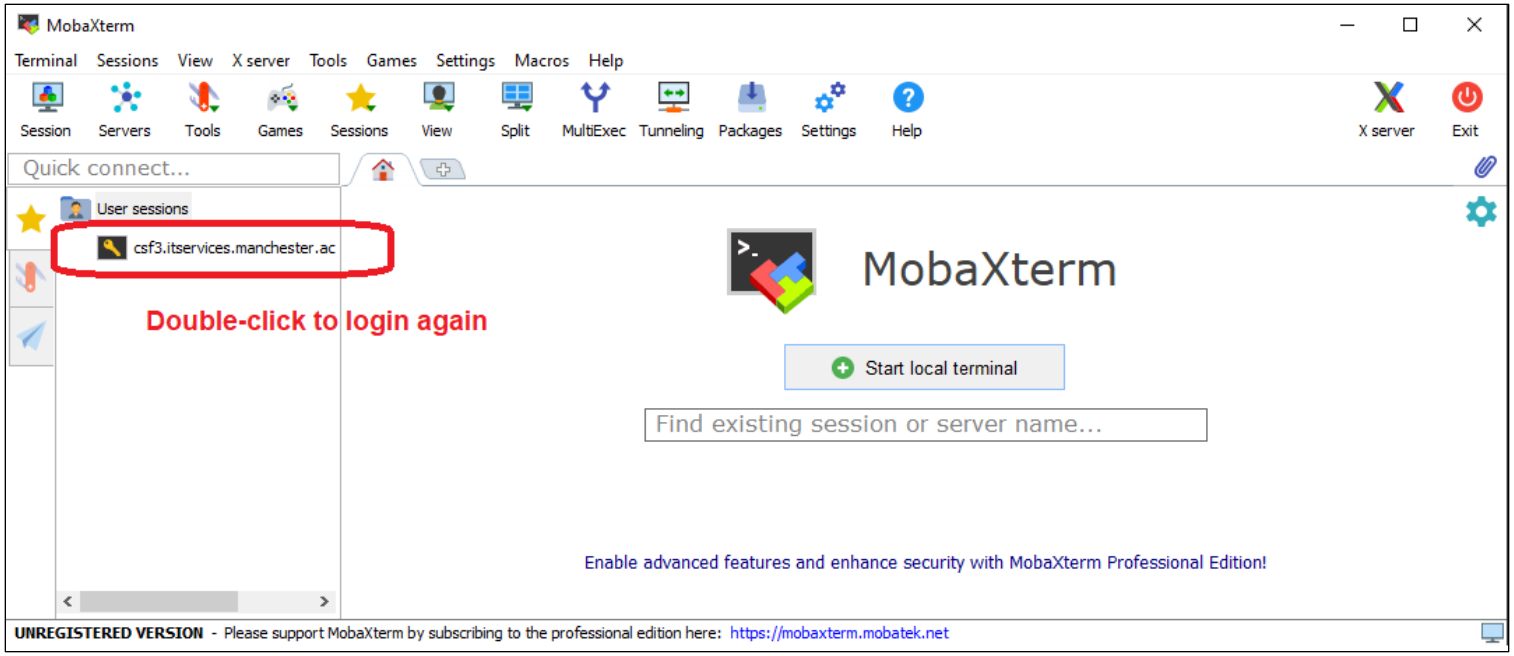
```
                                Welcome to CSF3
                                ===============

Docs: https://ri.itservices.manchester.ac.uk/csf3/getting-started
Help: its-ri-team@manchester.ac.uk

==============================================================================

                    *** REMINDER: Scratch Tidy In Operation ***

Reminder that scratch cannot be used for long term storage. Files not used
(not read or written by you or your jobs) for 3 months or longer will be
removed. However, please note that if you have recent jobs reading very old
datasets, those datasets will NOT be deleted.

                        !!! You may have files at risk !!!
    Use your CSF 'home' dir or RDS for keeping important files long term.
              These areas are backed up. Scratch is NOT backed up.

------------------------------------------------------------------------------
Jan 2023: New - check your scratch usage (space consumed and number of files)
          by running the following command on the login node: scrusage

------------------------------------------------------------------------------
19th June 2023: Due to the cyber-incident the University has turned off the
web-proxy. Therefore, users will NOT be able to access external websites,
repositories etc. via the web-proxy. If external access is required, please
use a batch job or interactive session on a compute node (via qrsh.) For more
info on doing this please see:
https://ri.itservices.manchester.ac.uk/csf3/batch/qrsh/

------------------------------------------------------------------------------
22nd Aug 2023: All access to scratch and RDS (isilon) storage has been
restored and batch jobs are running normally. Please check the outputs of
any recent jobs to ensure they completed as expected.

------------------------------------------------------------------------------
6th Sept 2023: scratch performance issues have been resolved.

                    ================= Please read all notices above =========================

[[redacted]@login1 [csf3] ~]$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

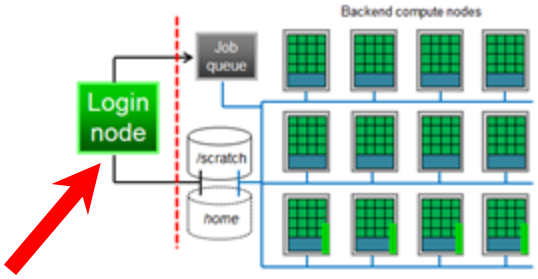
Next time you want to login to CSF from Windows



- Just double-click the csf3 "session" in the list of "User sessions"
- The CSF details are saved in the "session"
- (this also makes the file browser work, for drag-n-drop file transfers.)



Connect to CSF from a Mac



- **Mac users** - have a *terminal* application by default
  - **You will first need to install X-Quartz first**

    https://www.xquartz.org/ (install, then you should **reboot your Mac**)
  - Start a *Terminal* app (possibly from Go > Utilities > Terminal) and then type the following command:

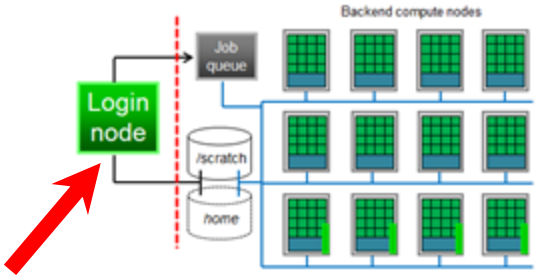
```
ssh -Y username@csf3.itservices.manchester.ac.uk
```

UPPERcase Y

Central IT Services username.
Answer 'Yes' to continue *if* asked.
Enter central IT password when asked (same as for email)

- Finished using CSF? Log out with: `logout` or `exit`

Connect to CSF from Linux



- **Linux** users - have a *terminal* application available by default
  - Start a Terminal (e.g., MATE terminal) and type the following command:

```
ssh -X username@csf3.itservices.manchester.ac.uk
```

UPPERcase X

Central IT Services username.
Answer 'Yes' to continue *if* asked.
Enter central IT password when asked (same as for email)

- Finished using CSF? Log out with: **`logout`** or **`exit`**

Linux / Mac Terminals

- You might be asked a question upon first login, before it asks for your password. It is safe to answer "yes".

```
meow:~ mabcxyz1$ ssh -Y mabcxyz1@csf3.itservices.manchester.ac.uk
```

The usual `ssh` command

```
[mabcxyz1@babbage ~]$ ssh -X mabcxyz1@csf3.itservices.manchester.ac.uk
The authenticity of host 'csf3.itservices.manchester.ac.uk (10.99.[redacted])' can't be established.
RSA key fingerprint is 66:8a:02:81:71:d2:71:d2:71:d2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'csf3.itservices.manchester.ac.uk' (RSA) to the list of known hosts.
mxyzabc1@csf3.itservices.manchester.ac.uk's password:
```

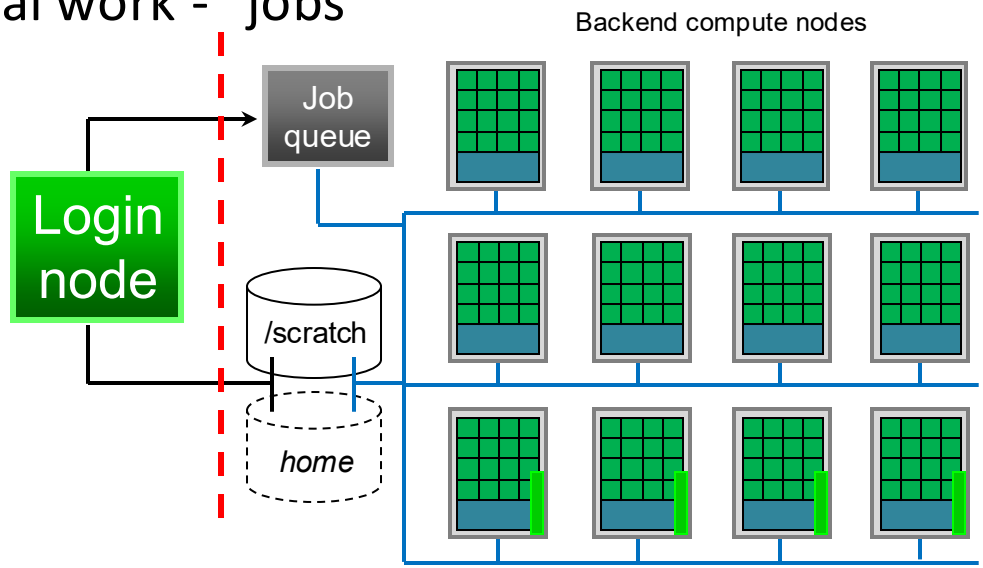
If asked “yes/no?”, answer: **`yes`**
(First login from Linux and Mac)

Doing real work on the CSF

RUNNING JOBS

Jobs, Jobscripts and the Batch System

- We want to do computational work - “jobs”



- You decide:
  - Which program(s) to run
  - Which resources it needs (#cores, CPU type, memory, GPU?)
  - How much time the job will need to complete its work
  - Which of your folders ("directory") to run the job in
- You'll put these requirements into a *jobscript* file
- Then submit your *jobscript* to the batch system ("Slurm")
- Slurm decides *when* the job runs and on which compute node(s). It ensures you get all of your requested resources.

Warning: The login nodes

- The server you connect to when you ssh to CSF
- Do *not* run computational work here:
  - Not enough cores
  - Not enough memory
  - 100+ users connected, so running work causes serious problems
- You *can* do the following:
  - Transfer files on and off the CSF
  - Set up and submit your jobs (covered in next few slides)
  - Basic data processing/viewing
- Computational work running on the login node will be killed without warning!

Creating Jobscript text files

- You need to be able to create a small text file to describe your job – the "jobscript"
- Run `gedit` on the CSF login node - a simple visual (GUI) text editor
  - Creates and saves the file *on the CSF*
  - `gedit` is similar to notepad
  - Other editors on CSF: nano, emacs, vi
  - Once logged in to the CSF type:

    ```
    gedit &
    ```

  - Navigate to a file or start typing and then save
  - Ignore cryptic warning messages from gedit

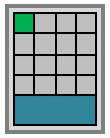
Can I Write Jobscripts in Notepad?

- A warning about Windows text files
  - There's an inconsistency over the end-of-line (hidden) characters in text files:
    - Windows: CR (carriage return) + LF (line feed)
    - Linux/Unix: LF (line feed)
  - It causes `sbatch` to reject your job immediately.

```
sbatch: error: Batch script contains DOS line breaks (\r\n)
sbatch: error: instead of expected UNIX line breaks (\n).
```

- Solutions
  - Use `gedit` on the CSF login node (writes Linux text files)
  - Or use notepad, upload then run `dos2unix myfile.txt`
    - Use only on jobscripts
    - Do not come to rely on this – it is too easy to forget to do it – use `gedit`!
    - Also means you have to transfer the file to the CSF, `gedit` creates it on the system for you.

A simple Jobscript – *Serial* (1 core)



`#!` on first line only (a special line)

First line indicates we use the *bash* scripting language to write our jobscript.

`#SBATCH` indicates a **batch system parameter** to specify our job requirements. We'll use various combinations of these.

myjob.txt

```
#!/bin/bash --login

#SBATCH -p serial
#SBATCH -n 1
#SBATCH -t 5

# Let's do some work
date
hostname
sleep 120
date
```

`-p` (`--partition=`) think of this as the queue, for serial (1-core) jobs in this case.

`-n` (`--ntasks=`) number of cores, which must be 1 for serial jobs (**optional line!**).

`-t` (`--time=`) maximum "wallclock" time the job is allowed to run for. Various formats. 5 is 5 minutes. 4-0 would be 4 days (0 hours).

lines are just comments - anything on the line after it will be ignored.

Actual Linux commands we run in our job. They will execute on a compute node.

Submit Jobscript to Job Queue

- Submit the jobscript from the login node with:

```
sbatch jobscript        # EG: sbatch myjob.txt
```

- You will be given a unique *JobID* (currently a 7-digit number). Can use this in other commands.

```
Submitted batch job 5980521
```

- You can then:
  - carry on with other work
  - check on your job queue and the jobs
  - submit more jobs, without disturbing previous jobs
  - log out of the CSF and your jobs will still run

Check status of your jobs

- To see your job(s) in the batch system, run:

  `squeue`

  `ST` (state) is either pending (`PD`), running (`R`) or failed (`F`). These are the most common states.

```
[redacted]@login2:/mnt/iusers01/support/[redacted]
File  Edit  View  Terminal  Tabs  Help
[mabcxyz1@login2[csf3] ~]$ squeue
          JOBID PRIORITY  PARTITION NAME          USER     ST SUBMIT_TIME    START_TIME     TIME        NODES  CPUS NODELIST(REASON)
        5980521 0.0020309 serial    myjob.txt     mabcxyz1 R  09/09/25 14:01 09/09/25 14:01 0:08           1     1 node001
```

  `CPUS` is the number of cores (1 by default - a *serial* job)

Serial (1-core) Job Properties

- Our simple example job:
  - Serial (only **1** core is used)
  - Standard memory (*no* `#SBATCH` line asking for *more*)
    - Our job gets ~5GB RAM to work with
    - The serial *partition* contains node with Intel CPUs, 5GB/core
  - We said the job would need no more than 5 minutes runtime (max permitted is 7-0 i.e., 7 days)

- NOTE: You will see that the example jobscripts *in the exercises* contain the line:
  `#SBATCH -reservation=course`
  - **Only for use today** (reserved nodes on a teaching day).
  - **Remove** if you are running jobs *after* today's session.
  - **Never** use on your regular jobs (they'll wait forever!)

So where did my results go?

- If `squeue` returns no output - means job has finished!
- Where are my results? Three possibilities:
  1. If your app usually prints to screen: output captured to a text file called ***slurm-JOBID*.out** where
     - *JOBID* is the number of your job
     - Previous example: `slurm-5980521.out`
  2. Output file(s) specific to your application
     - EG Abaqus: *casename*`.dat`, *casename*`.prt`,…
  3. Your job had a problem or failed: it may be reported in one of the above files – check the ***slurm-JOBID*.out** file.
- Various ways to view the files (they are plain text):

`cat` *filename*	# Example: `cat slurm-5980521.out`
`less` *filename*	(allows you to page through with spacebar)
`head` *filename*	(show the first few lines of a file)
`tail` *filename*	(show the last few lines of a file)
`gedit` *filename*	(not recommended if it is large)

Why are my jobs still waiting?

- Your jobs will wait until there are cores / mem / GPUs available (meeting your requirements)
- Initially frustrating (perhaps) but advantages:
  - You can log off, switch off your PC but your jobs will stay on the CSF. Log in later to check on jobs / collect the results.
  - You can submit *many* jobs
  - Several / many jobs can run at the same time
    - Might need to run jobs from different folders, which also keeps your files tidy

How busy is it?

- To see all queue entries for everyone

```
\squeue          # Put a \ at the start
```

- Note: all partitions shown as *one list* by `squeue`
  - Your job is *not* necessarily stuck behind all others above yours in the `squeue` output.
  - But, some jobs submitted *after* yours may start *before* yours!
  - **Don't wait** to submit jobs, hoping for the CSF to appear less busy. **It's always busy!**
  - A realistic wallclock timelimit will help.
- Contact us if your job appears stuck (a wait longer than 24 hours)

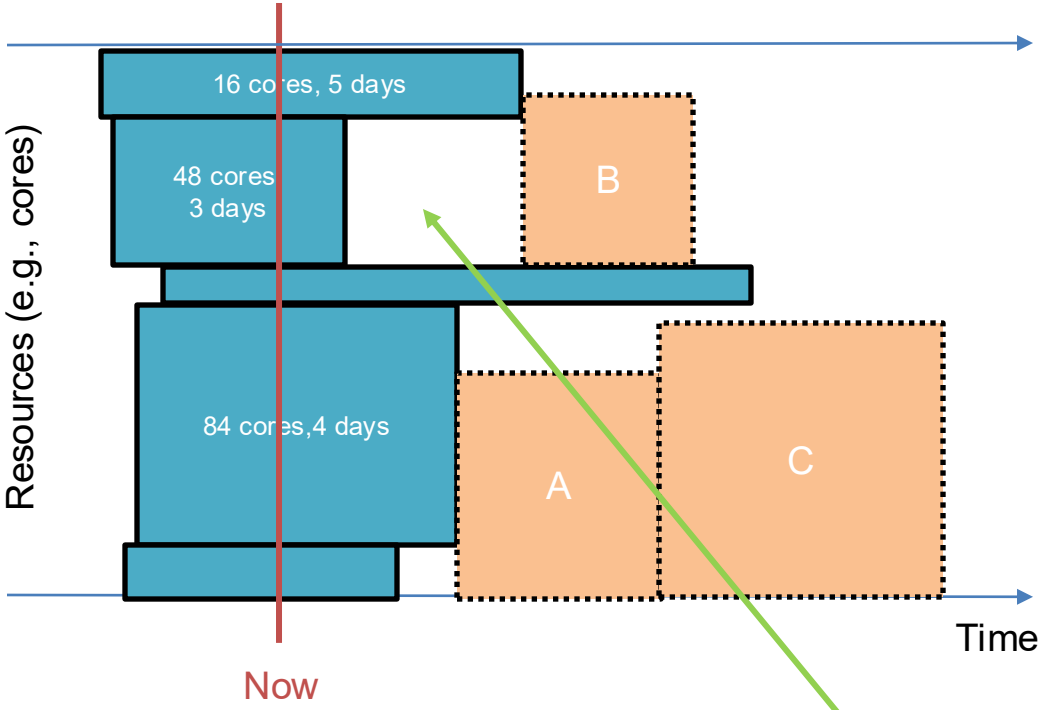
Why do I need to specify a Wallclock time limit?

- We require that you add a wallclock time to your jobscripts:
  - `#SBATCH -t 2-0` 2 days (0 hours). **7 days is the max**.
  - `#SBATCH -t 30` 30 minutes
  - `#SBATCH -t d-hh:mm:ss` Various formats accepted

- By specifying an accurate(ish) wallclock, Slurm can better plan when resources will become free.
  - Always err on side of caution – too much time is better than not enough time!
  - **Slurm will kill a job if still running, once the job's wallclock limit has been reached!**
  - You might have to run a few jobs to get a feel for how long they take.
  - Or run the first job with 7-days then check the actual runtime.

- Can't I just request 7-days for all of my jobs (or 4-days for GPU jobs)?
  - Yes, you can. But …
  - It might be possible to fit a shorter, small job in before larger jobs are able to use the resources.
  - Ultimately, everyone will wait longer

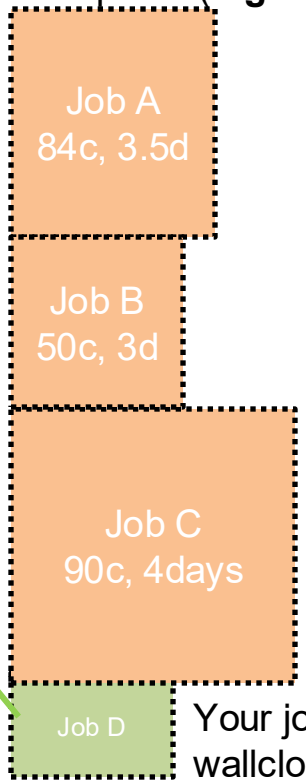
Backfill Scheduling



Running jobs

Pending jobs

Job queue (**highest priority at top**)



Your job with a realistic wallclock limit.

A resource gap can be filled by a **lower priority job**, *provided* doing so will not delay higher priority jobs.

Without backfill, those resources would be idle.

Checking your jobs and deleting jobs

- `squeue` reports your job as ‘`F`’ (failed)
- Or your job has finished but you suspect it failed to complete correctly
  - The `slurm-`*`JOBID`*`.out` contains errors / incomplete results
  - Ask Slurm for info about your job once it has finished
    `seff` *`JOBID`* - Resource usage efficiency and exit code
    `sacct -j` *`JOBID`* - LOTS of stats about job (`MaxRSS` is peak mem used.)
- Most common causes of errors:
  - Job ran out of memory ("OOM" error in `slurm-`*`JOBID`*`.out`)
  - Missing directory (did you rename the directory before job ran?)
  - Unusual characters or **spaces** in file and directory names
  - No disk space on the filesystem – run in the scratch area to avoid this
- Detailed advice:
  https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/monitoring/
- To delete a job from the queue (e.g., a failed job, or you just no longer want it – pending or running):

  `scancel` *`JOBID`*

Practical 2 – Submit a serial job

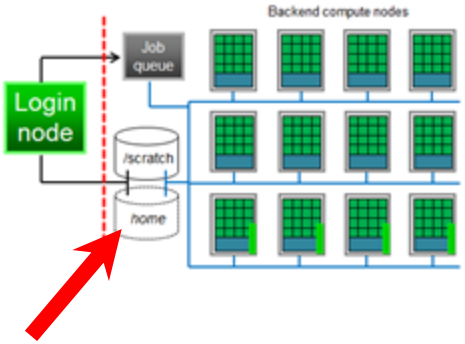
- Run a simple serial job and check the results

https://ri.itservices.manchester.ac.uk/course/csf-mace/csf-mace-ex2.pdf

Where to store your files…

CSF STORAGE (FILESYSTEMS)

Storage – Home (1)



- Upon login, you are automatically placed in your *home* directory (folder), for example:

  `/mnt/iusers01/fse-ugpgt01/mace01/`*`username`*

- Very limited space, shared by everyone
- To see the quota for your group

  ```
  df -h .        # The . means "current dir"
  ```

- Where am I?

  ```
  pwd
  ```

- How much space am I using?

  ```
  du -sh
  ```

- How big is that file?

  `ls -lh` *`filename`*

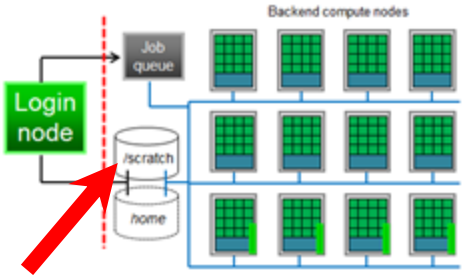
- How much space is used/free overall?

  ```
  df -h .                    # The . is important!
  ```

Storage - Home (2)

- *Home* is backed up and mirrored to another datacentre
  - Keep important files here (results, jobscripts, source code, …)
  - Deleted a file by mistake? Contact us via https://ri.itservices.manchester.ac.uk/hpc-help - we'll tell *you* how to recover it
- Only you can access your home directory
  - File permissions can be used to give others access
  - Contact us via https://ri.itservices.manchester.ac.uk/hpc-help if you want advice on this as they can be complex
- Do not run jobs from your *home* area (see later)
  - Can generate a lot of files, some of them large
  - Using up all of the shared space will make your colleagues unhappy!
  - Consider compressing large (text) files with `gzip`

Storage - Scratch (1)



- *Scratch* Filesystem local to CSF for:
  - Temporary files
  - Running jobs from (it's faster!)
- Shared by all CSF users, but we have 1.2PB
- Tidy up after jobs finish
- Clean-up policy applied: files not read or written in last 3 months may be deleted automatically without warning
- Not backed up
  - Copy important results to *home* area
  - *scratch* not considered safe for long term storage - hardware failure could cause data loss

Storage - Scratch (2)

- Using scratch is easy: after log in move to it:

  `cd ~/scratch`

  - This uses a ‘symlink’ (short cut) in your home dir to the real directory:

    `/scratch/`*`username`*

- Create a directory (now we’re in scratch):

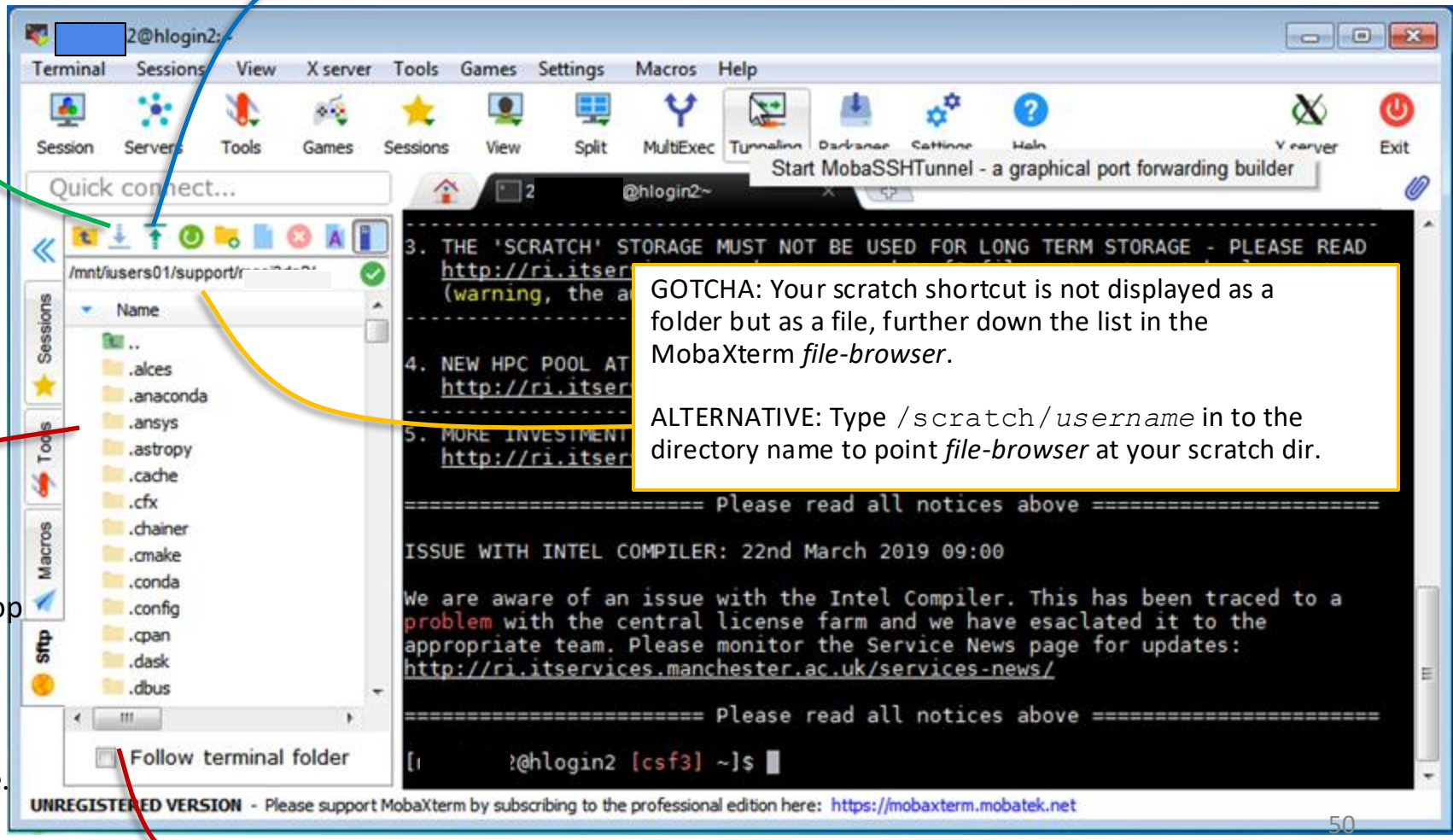
  `mkdir myjobdir`

- Put all files relevant to your job in that directory and run your jobs - we’ll try this out soon...
- All compute nodes see the same scratch area

Uploading and downloading files using MobaXterm for Windows

2. First select files in the MobaXterm browser. Then the ***Download*** button opens a file-browser to select a destination folder on your PC.

3. The ***Upload*** button opens a file-browser to select files on your computer to **upload to CSF** (current directory).



GOTCHA: Your scratch shortcut is not displayed as a folder but as a file, further down the list in the MobaXterm *file-browser*.

ALTERNATIVE: Type `/scratch/username` in to the directory name to point *file-browser* at your scratch dir.

1. Drag-n-drop files from Windows Explorer or Desktop to/from here.

This can be flaky - do not use

Uploading and downloading files using Linux, Mac or Mobaxterm command-line

Note: the commands below are **not** run on the CSF

The **:** is important!

- Transfer a file **from** your computer **to** your CSF home dir

```
scp myfile.txt username@csf3.itservices.manchester.ac.uk:
```

No destination directory after the : means "use your home directory"

- Transfer a file **from** your CSF home dir **to** your computer

```
scp username@csf3.itservices.manchester.ac.uk:results.out .
```

The **.** is shorthand meaning "the current directory" on your computer

- Transfer a file **from** your computer **to** your CSF scratch dir

```
scp file2.txt username@csf3.itservices.manchester.ac.uk:scratch/
```

- Transfer a file **from** your CSF scratch **to** your computer

```
scp username@csf3.itservices.manchester.ac.uk:scratch/results2.out results2.copy
```

Now you get a copy with a different name on your computer.
Or could just use a . to mean current directory on your computer.

Basic Linux File Commands

A good Linux tutorial is available at: https://www.chm.bris.ac.uk/unix/

Command	Description

`cd dir1`
`cd ~/dir1/dir2`
`cd ..`
`cd`
	Change directory (go in to `dir1` which is located in the current dir)
	Go in to `dir2` in `dir1` in home (~ is shorthand for *home*)
	Go up to parent directory (e.g., from `~/dir1/dir2` to `~/dir1`)
	Go back to *home* (useful if you become lost)

`pwd`	Lost? Print Working Directory (display current location)

`ls`
`ls -lh`
`ls -lh file1 dirA`
`ls -lh dirA/*.dat`
	List content (names of files and directories) of current directory
	List in long form (dates, file sizes, names) current directory
	List in long form (dates, file sizes, names) specified files, directories …
	List in long form all files ending in `.dat` in directory `dirA`

`mkdir dirA`	Make directory named `dirA` (in the current directory)
`cp fileA fileB`	Copy (duplicate) a file (copy `fileA` to a new file `fileB`)

`mv fileC fileD`
`mv fileE dirA`
`mv fileF dirA/fileG`
	Rename a file (from `fileC` to `fileD`). Works for directories too.
	Move `fileE` in to sub-directory `dirA` (`dirA` must exist)
	Move `fileF` AND rename it all in one go (`dirA` must exist)

`rm fileH`	Delete (remove) a file (caution!!)
`rm -rf dir1`	Delete directory and all files (and other sub-dirs) in there (**caution**!!!!!)

`gzip bigfile`
 	
`gunzip bigfile.gz`
	Compress a file (becomes `bigfile.gz`) to make better use of disk-space. Text files usually compress well.
	Uncompress previously compressed file (becomes `bigfile`).

Basic Linux File Commands

A good Linux tutorial is available at: https://www.chm.bris.ac.uk/unix/

Command	Description
`less file1` `zless file2.gz`	Display the content of `file1` (text file) a page at a time on screen. If you've compressed `file2` with `gzip`, no need to uncompress first. Press `space` to page down through a long file Press `return` to scroll down a line at a time Press `b` to scroll back up a page Press `G` to go to end of file Press `q` to quit/exit
`cat file1` `zcat file2.gz`	Dump entire file to screen (a quick way to look at text files). If you've compress `file2` with `gzip`, no need to uncompress first.
`gedit file1`	Edit `file1` using a simple graphical text editor (similar to notepad on Windows). See later for more on opening graphical programs on the CSF so that they display a window on your computer.
`file filenameA`	Try to tell us what type of data is in `filenameA`. Useful to determine the output of some program where you are not sure what type of output it has generated. For example: `file output.dat` Might be `ASCII text` (so we can look at it with `less` or `gedit`) or might be `data` (you'll need some other program to read it)
`du -sh .`	How much disk space is current directory (all files and subdirs) using?
`df -h .`	How much free space is there in the current area?

Practical 3 – File Transfer

- Download a file to your laptop/PC from the course website, then:
- Upload the file to the CSF (**needed to do exercise 4 later**)

https://ri.itservices.manchester.ac.uk/course/csf-mace/csf-mace-ex3.pdf

REAL SOFTWARE AND PARALLEL WORK

Accessing application software

- We saw a 'module' command in exercise 1's jobscript
- Apps on the CSF are accessed through ‘modulefiles’
- Ensures your job can find the software on the system
- Load the module in your jobscript so you always know which app and version you used:
  - It is possible to load modulefiles on the login node, but this is not recommended for batch jobs.
- Examples (the modulefile names are quite long!)

```
module load apps/binapps/abaqus/2021
module load apps/gcc/openfoam/v2012
module list
module show apps/binapps/tensorflow/2.8.0-39-gpu
```

- Each app on the CSF has a webpage, which includes the correct ‘modulefile’ to use:

  https://ri.itservices.manchester.ac.uk/csf3/software/
- Not all software available to all users due to licensing
  - Check our webpages to find out how to request access, it may involve paperwork e.g. Abaqus

Loading modulefiles: On login nodes OR in the jobscript

Inherit from the login node (**not recommended**)

Jobs in Slurm will inherit any modulefile settings (i.e. loaded modules) from the login node at the point when you *submit* (sbatch) the job.

```
# On the login node:
module load apps/R/4.4.1
sbatch myjob.txt
```

myjob.txt

```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 2-0



# We'll use whichever version
# of R was loaded on the login
# node. Which version of R did
# I use 6 months ago???
R CMD BATCH myscr.R
```

Only in the jobscript (**recommended!**)

```
# On the login node:
sbatch myjob.txt
```

myjob.txt

```
#!/bin/bash --login
#SBATCH -p serial
#SBATCH -t 2-0

# Start with a clean env and
# load module inside jobscript
module purge
module load apps/R/4.4.1

# We know the version of R!
R CMD BATCH myscr.R
```

A note about our documentation

- Over the summer we change the batch system from SGE to Slurm
  - SGE uses #$ as the jobscript *sentinel*
  - Slurm uses #SBATCH (as we've seen earlier)
  - Our applications documentation is being updated to convert example jobscripts from SGE to Slurm
  - If you see #$ in our documentation, you'll need to write the equivalent Slurm jobscript (using #SBATCH).
  - See our SGE-to-Slurm guide, which shows how #$ flags map to #SBATCH flags https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/sge-to-slurm/

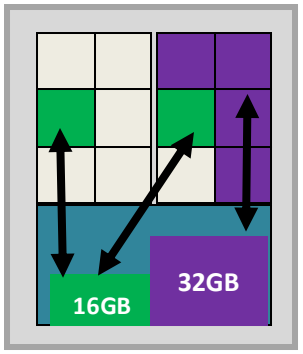
PARALLEL COMPUTING

Background

Motivations for Parallel Computing

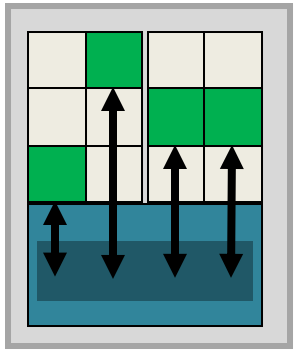
- CSF compute nodes have multiple CPU cores
  - up to 32 cores on Intel compute nodes
  - Up to 168 cores on AMD compute nodes
- Some apps can use several cores to: Speed up computation
  - Can the computation be split over multiple CPU cores?
    - Each core does a small(er) part of the computation
  - May need to *combine* results together at end
  - Should get overall result quicker
    - Ideally *N* cores giving results *N* times quicker
- Also provides access to more memory
  - Each core has access to ~8GB RAM (std nodes)
    - Ideally *M* cores for *M* times larger problem
- Both of the above!

AMD 168-core node,
8GB/core (only 12 cores shown)

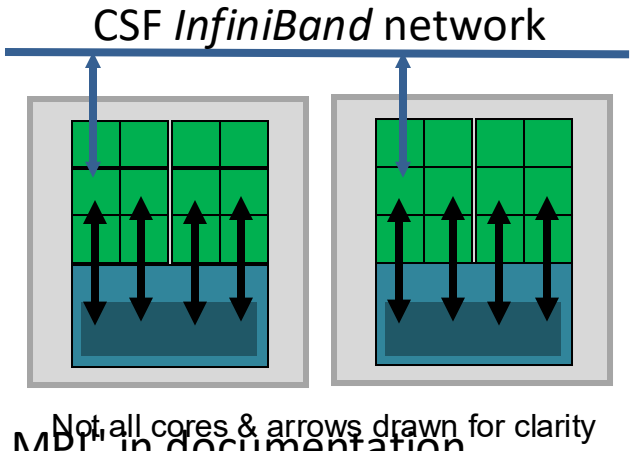


Parallel Job Types

- Single node (shared memory apps)
  - EG: Use 4 cores (can be 2 – 168 on CSF)
  - App reads input data (e.g., 3D mesh geometry)
  - Each core in the job works on a section of the data
  - The technique used to implement these apps means only a single compute node can be used
  - Look for "Shared memory / multi-threaded / OpenMP" in documentation



- Single and multi-node (distributed memory apps)
  - The technique used to implement these apps means jobs can use the cores on a single node (a small job) or the cores on multiple nodes (a large job)
    - One "master" process coordinates the others
    - They may need to exchange data
      - Can only see their own data chunk
      - Send messages to each other
  - Look for "Distributed memory / Message Passing / MPI" in documentation
  - **Multi-node hardware is not available to you – most users only need to run single-node jobs now (up to 32 cores on Intel nodes and 168 cores on the AMD nodes.) Multinode jobs are not further covered in these notes.**



Parallel Jobscript on CSF

- Use a jobscript to ask the batch system to find *N* free cores
  - While matching other requirements (memory, architecture, fast networking, GPU etc).

1. Add extra lines in jobscript to request:
   - *a parallel partition* (for multi-core or multi-node jobs)
   - *and number of cores* to reserve
2. Inform your app how many cores to use
   - Remember, the jobscript says how many cores your *job* requires (the batch system will allocate those cores to your job.)
   - **But** *you* must still ensure your app uses no more!!
     - This is not always automatic and how you do it varies from app to app

Parallel Jobscript – Multi-core (*single AMD compute-node*)

#! - see serial jobscript earlier.

myparajob.txt

multicore is the partition name. This one means: app will multiple cores on a single compute node (2 to 168 AMD "Genoa" cores.)

```
#!/bin/bash --login
#SBATCH -p multicore
#SBATCH -n 8
#SBATCH -t 3-0

# Set up to use a simulation app
module purge
module load apps/intel-17.0/serpent/2.1.31

# Inform app how many cores to use
export OMP_NUM_THREADS=$SLURM_NTASKS

# Run your app (serpent in this case)
sss2-omp mysim.inp
```

-n (**--ntasks=**) **8** is the number of **cores** we want to *reserve* in the system. Each **partition** has a maximum number.

-t wallclock time limit for the job. This jobs is given 3 days (0 hours).

The commands we run in our job. They execute on a compute node that has the required number of cores free. **sss2-omp** is an app named "Serpent".

PLEASE NOTE: serpent is a restricted app (ask us for access)

$SLURM_NTASKS is automatically set to the number, **8** in this case, given on **-n** line. Will be **1** in "serial" partition.

Parallel Jobscript - Single-node (Intel)

- That was a multicore (*single* compute node) example
- Using an app named Serpent as an example

  https://ri.itservices.manchester.ac.uk/csf3/software/applications/serpent/
- Requested a partition (-p) and number of cores (-n)
  - Job will run the app on a single AMD "Genoa" node, allocating multiple cores on that node to the job.
- Informed the app to use 8 cores via `OMP_NUM_THREADS` environment variable (very common).
  - Special `$SLURM_NTASKS` variable always set to number of cores on the `-n (--ntasks=)` line.

Parallel jobscript - Multi-core (cont…)

- As with the serial job, submit it to the system with
  `sbatch` *`jobscript`*

- Monitor with `squeue`
- It may take longer for *more* cores to become free in the system)
- You'll get the usual output file
  `slurm-`*`JOBID.out`*

Parallel Partitions

https://ri.itservices.manchester.ac.uk/csf3/batch/parallel-jobs/

Partition Name	Description
`multicore`	2-168 cores, single compute node. 8GB per core. Jobs will be placed on AMD EPYC "Genoa" (max 168 cores/job)
No optional flags	

Partition Name	Description
`multicore_small`	2-32 cores, single compute node. ~4-5GB per core. Jobs will be placed on Intel "haswell" (max 24 cores/job) or Skylake (max 32 cores/job)
`-C` *`architecture`*	**Not recommended!** (`haswell` or `skylake`)

- **Max permitted runtime limit** is 7 days (but *must* be specified by you in the jobscript.)
- Our simple jobscript did *not* use any of the above extra flags. Not needed in most cases.
- If you limit a job by *architecture* it may wait longer in the queue.

Parallel jobscripts – Abaqus

- Restricted app – email us, there's "paperwork"
  https://ri.itservices.manchester.ac.uk/csf3/software/applications/abaqus/
- Can run serial, single-node, multi-node. You must tell it how much memory it has available too.

```
#!/bin/bash --login
#SBATCH -p multicore    # AMD nodes (2-168 cores) [8GB per core]
#SBATCH -n 168          # Number of cores (maybe try a smaller job first!)
#SBATCH -t 4-0          # Wallclock time limit (4-0 is 4 days)

# Setup to use Abaqus then check for available licenses
module purge
module load apps/binapps/abaqus/2023
. $ABAQUS_HOME/liccheck.sh

# 168 cores, with 8GB per core = 1344GB total memory available
abq2023 job=myabqjob input=myabqjob cpus=$NSLOTS scratch=$HOME/scratch \
        memory="1344 gb" interactive
```

Parallel jobscripts – Fluent

- Restricted app – email us for access
  https://ri.itservices.manchester.ac.uk/csf3/software/applications/fluent/
- Can run single-node parallel and interactive (GUI)
- See our docs for how to compile User Defined Functions (UDF) and run interactively
- Note: Earlier versions limited to max 16 cores, not 32

```
#!/bin/bash --login
#SBATCH -p multicore    # AMD nodes (2-168 cores) [8GB per core]
#SBATCH -n 8            # Number of cores
#SBATCH -t 4-0          # Wallclock time limit (4-0 is 4 days)

# Set up to use Fluent
module purge
module load apps/binapps/fluent/2021R1

# The '3d' is the simulation type (e.g., '2d', '2ddp', '3d', '3ddp')
fluent 3d -g -t$SLURM_NTASKS -mpi=openmpi -i input.jou
```

Parallel jobscripts – StarCCM+

- Restricted app – email us for access
  https://ri.itservices.manchester.ac.uk/csf3/software/applications/starccm/
- Can run single-node and multi-node parallel
  - Different precision: "mixed" for accuracy and speed. "Double" is more accurate but slower.

```
#!/bin/bash --login
#SBATCH -p multicore    # AMD nodes (2-168 cores) [8GB per core]
#SBATCH -n 32           # Number of cores (See -cpubind flag below)
#SBATCH -t 4-0          # Wallclock time limit (4-0 is 4 days)
# Set up to use StarCCM+ and check for available licenses
module purge
module load apps/binapps/starccm/15.02-mixed
. $STARCCM_HOME/liccheck.sh

# Add "-cpubind off" when NOT using all 168 cores on the node
starccm+ -batch -pio -mpi openmpi -batchsystem slurm myinput.sim \
         -cpubind off
```

```
#SBATCH -n 168          # Large AMD-node job (all cores on the node)
                        # No -cpubind flag when using all cores
starccm+ -batch -pio -mpi openmpi -batchsystem slurm myinput.sim
```

Parallel jobscripts – OpenFOAM

- Open source, no restrictions on access
  https://ri.itservices.manchester.ac.uk/csf3/software/applications/openfoam
- Can run serial, single-node and multi-node jobs
- Several apps to pre-process, simulate, post-process.
- Num cores in job is also needed in mesh control file!

```
#!/bin/bash --login
#SBATCH -p multicore    # AMD nodes (2-168 cores) [8GB per core]
#SBATCH -n 8            # Number of cores
#SBATCH -t 4-0          # Wallclock time limit (4-0 is 4 days)
# Set up to use OpenFOAM (notice extra setup step)
module purge
module load apps/gcc/openfoam/v2012
source $foamDotFile

# Many different solvers (e.g., interFoam). Some need different steps.
blockMesh             # Only uses 1 core
setFields             # Only uses 1 core
decomposePar          # Only uses 1 core (see webpage for parallel ver)
mpirun interFoam -parallel    # Knows how many cores (8) to use
recomposePar          # Only uses 1 core (see webpage for parallel ver)
```

Practical 4 – Parallel job

- Submit a small parallel OpenFOAM job using files from exercise 3
- Generate a movie of the simulation
- Download the movie file to your laptop / PC

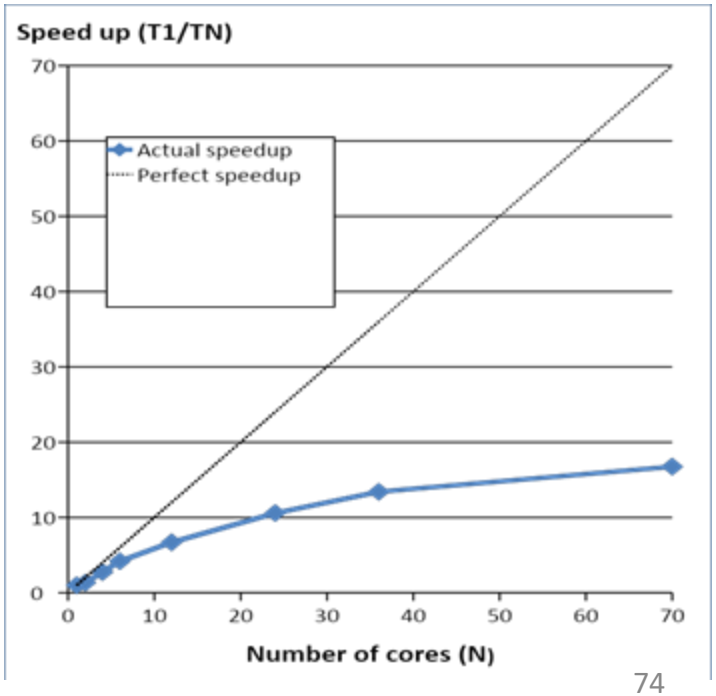
https://ri.itservices.manchester.ac.uk/course/csf-mace/csf-mace-ex4.pdf

The following slides are "further reading" when doing the 2-hour classroom course.

PARALLEL SCALING

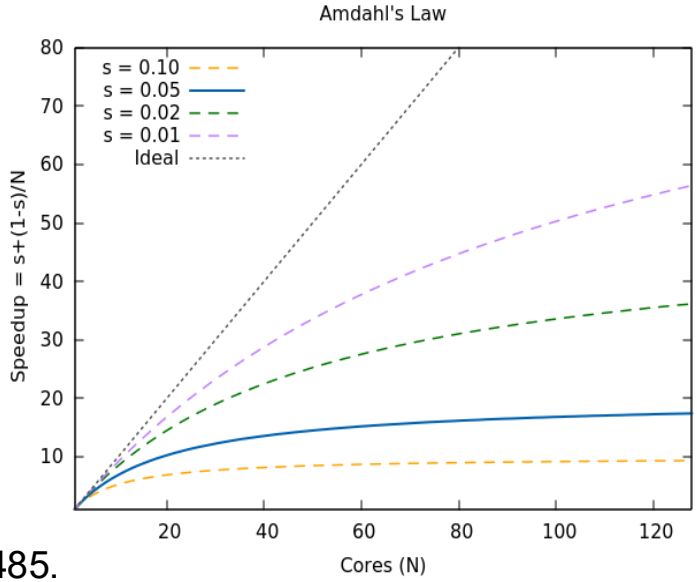
Parallel Software "Scaling"

- Does adding more cores reduce solution time?
  - Does the app "scale"?
- We can calculate: Speedup = T_1 / T_N
  - T_1 is runtime on one core, T_N is time on N cores
  - Plot for different N. Also "perfect" speedup (where the speedup is N on N cores)
  - Perfect speedup: when all cores contribute 100% of their computational power.
- Alternatively, calculate: Par Efficiency = $T_1 / (N x T_N)$
  - Ratio of actual speedup and ideal (perfect) speedup



Strong vs Weak Scaling

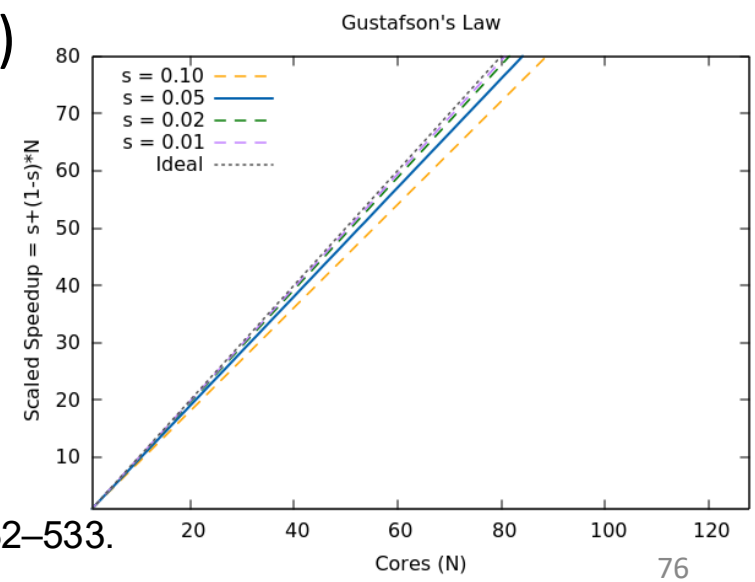
- Strong Scaling (fixed problem size, incr #cores)
  - Amdahl's law: speedup *limited* by fraction of the application that *can't* be parallelized
  - Gives: Speedup = 1 / (s + p/N)
    - s = proportion of runtime spent doing serial work,
      p = proportion of runtime spent doing parallel work (= 1-s)
      N = number of cores
    - EG: 20 hours runtime:
      1 hour doing serial work,
      19 hours doing parallel
      s=0.05, p=0.95
    - Even if N->∞ (so that p/N->0)
      speedup only 1/s (= 20) ☹
      (so why buy large HPC systems?
      See next slide...)



Amdahl, Gene M. (1967). *AFIPS Conference Proceedings.* (30): 483–485.
doi: 10.1145/1465482.1465560

Strong vs Weak Scaling

- Weak Scaling (increase problem size & #cores)
  - Gustafson's law: *scaled speedup* increases with number of cores as we *increase the problem size*
    - Assumes the time spent on serial work in the app doesn't increase as you increase your sim size, say.
  - Gives: Scaled Speedup = s + p x N
    - Previous example: s=0.05, p=0.95
    - If N is large (e.g. 1000 cores) Scaled speed up is 950 ☺
  - Now we can tackle larger problems in ~ same time as smaller problems that ran on smaller systems



Gustafson, John L. (1988). *Communications of the ACM*. 31 (5): 532–533. doi: 10.1145/42411.42415

Parallel Software "Scaling"

- You may be running an app (job) *many* times
  - Simply running a huge job might not be beneficial and you may queue for longer than a smaller job.
- Worth a small investigation to find optimal performance parameters (small vs large job, #cores & #nodes)
  - How many cores should I use?
- Do a few runs with different number of cores
  - Plot the speedup
  - Easy to do on CSF: override lines from jobscript by adding to sbatch command-line:
  - *Must* use $SLURM_NTASKS in jobscript to automatically use the correct number of cores (see earlier)

```
sbatch -p serial    -n 1 myjobscript.txt
sbatch -p multicore -n 2 myjobscript.txt
sbatch -p multicore -n 4 myjobscript.txt
sbatch -p multicore -n 8 myjobscript.txt
```

Exercise 5 (Advanced)

- Optional exercise if we have time (or do later)
  - Compile a parallel (MPI) code written in C
  - Run a job several times with different numbers of cores (but same data size)
  - Determine how the code is scaling (see scaling formula earlier)

https://ri.itservices.manchester.ac.uk/course/csf-mace/csf-mace-ex5.pdf

INTERACTIVE JOBS

Interactive work (1)

- Some apps can use a GUI to setup and even run the simulation
  - e.g., StarCCM+, Abaqus, Fluent, ...
  - This is probably how you do it on the MACE workstations
  - Q: Do the workstations log you out (or reboot!) after a while?

- Please **do not** do this on the CSF login node
  - Datasets (e.g., FEA mesh) might be too large
  - You might start the simulation running (on all login node cores!!)
  - You'll probably annoy everyone else on the login node

- But we've already seen that batch jobs do not pop-up a GUI, allow no interaction and simply capture all text output to file.
  - So how to use the GUI of a "big" app on the CSF (not just `gedit`)?

- Instead, "login" to a compute-node and run the app there:
  - We'll use the `srun` command (not `sbatch`) to get an "interactive session" on a *compute node*
  - Allows app's GUI to pop-up – appears on your screen (just like `gedit` does)
  - But it is using the resources of the *compute node*, not the login node

Interactive work (2)

- Method 1 (just the app):

```
# On the login node, run:
cd ~/scratch/my-simulation-data
module load name/of/module/1.2.3
srun -p interactive -t 0-1 appname [flags ...]
# Or for a parallel interactive session:
srun -p interactive -t 0-1 -n 4 appname [flags 4 ...]
# (exit the app to end session and return to login node.)
```

- Method 2 (the app and possibly other commands):

```
# On the login node, run:
srun -p interactive -t 0-1 --pty bash
# Or for a parallel interactive session:
srun -p interactive -t 0-1 -n 4 --pty bash
   # (now wait, then type commands on the compute node)
   module load name/of/module/1.2.3
   cd ~/scratch/my-simulation-data
   appname flags $SLURM_NTASKS …
   # After exiting the app, can do more work on node or:
   exit
# (we are now back on the login node – notice the prompt)
```

- The nodes used have limited graphics capability (GPU nodes won't help us)
- Only 8GB *per core*, max 1 hour runtime (enough time for simulation setup).
- As with `gedit` use MobaXTerm on Windows, X-Quartz on Mac
- Setup your sim interactively using the GUI. Then **run sim as a batch job**!

Exercise 6 – Interactive Work

- This example uses "paraFoam" to view the OpenFOAM simulation results.
  - Cut-n-paste from here to your terminal should work correctly (do one line at a time)

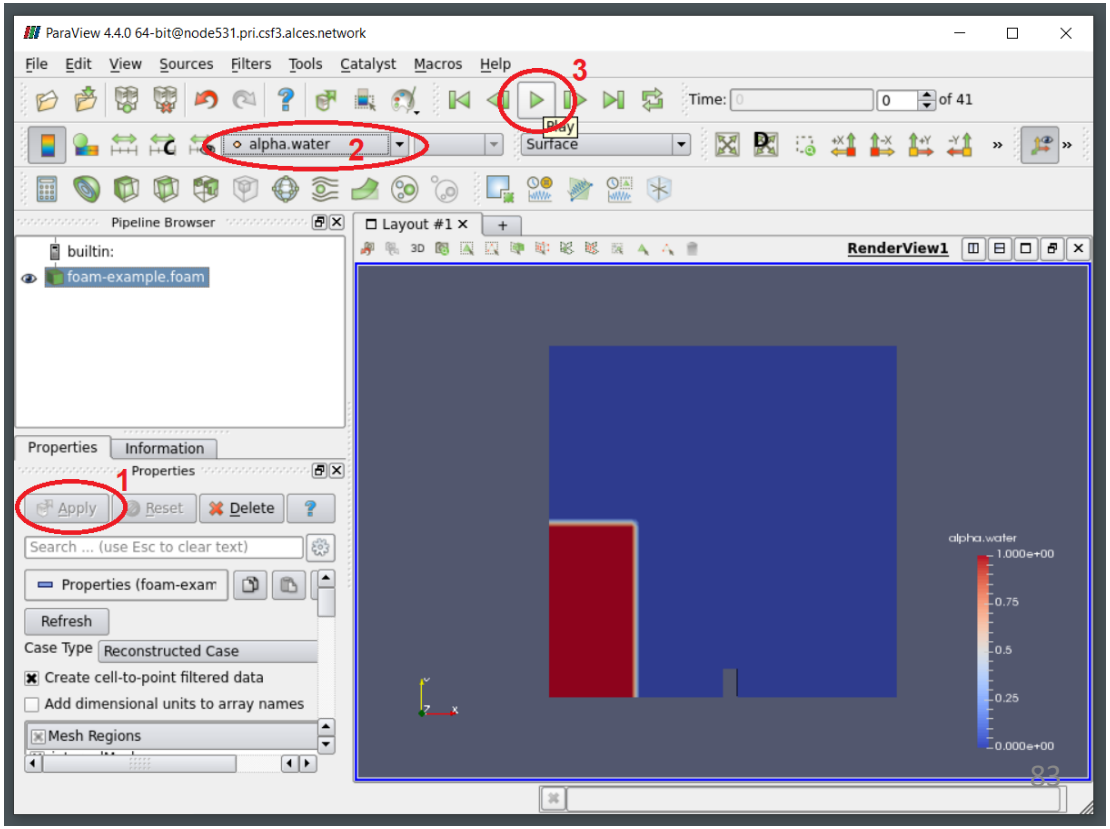
```
# On the login node, run:
cd ~/scratch/mace-course/foam-example
module purge
module load apps/gcc/openfoam/v2012
source $foamDotFile
module load apps/binapps/paraview/5.13.3-gui
# Start paraFoam interactively (uses a compute node)
srun -p interactive -t 0-1 paraFoam --mesa
```

  - **MacOS**: See next slide if libGL error reported
  - 1. Press the green "Apply" button on the left hand-side to load the results from the simulation we ran earlier.
  - 2. Then change the "`vtkBlockColors`" in the drop-down menu at the top to "`alpha.water`" (the first one in the menu).
  - 3. Then press the green "Play" ▶ button at the top.
- **Please note**: ParaFoam (based on paraview) is **slow** when run *this* way
- Only use this method for small simulations / meshes / spreadsheets (.csv)
- Next, we will present a better way of running paraview for large datasets.

Interactive work - paraFOAM

- **MacOS**: may see a `libGL` error about `fbConfigs`
  - Log out of CSF. Then, before logging back in, run:
    `defaults write org.xquartz.X11 enable_iglx -bool true`
  - Now login again as before:
    `ssh -Y` *`username`*`@csf3.itservices.manchester.ac.uk`
    `Password:`
    `Duo (1-1): 1`

- **All**: Do these 3 steps in paraFOAM:
  - (The "Apply" button will be green initially)



Paraview for larger datasets

- We *really* want to avoid downloading large datasets (e.g., simulation results) to vizualize on a PC/laptop
  - Results from CSF simulations can be huge (good!)
  - Example: OpenFOAM 3D mesh geometry & simulation “field” variables per timestep (velocity, temperature, pressure, …)

- But running the paraview GUI on the CSF (even using `srun`) is inefficient for large datasets / dense meshes / lots of timesteps
  - A lot of geometric data (e.g., 3D mesh) and colour data (e.g., representing field values at points on the mesh) is transferred over the network so it can be *rendered* (converted to an image) "at your end" where the paraview GUI is being *displayed* (this is how OpenGL works)
  - *Rendering* can require a lot of local memory & compute power.
  - A GPU in your PC it might help. But network transfer cannot be avoided.
  - This can make paraview feel unresponsive / slow.
  - (But, it *is* an option if you can't install paraview on your own PC/laptop.)

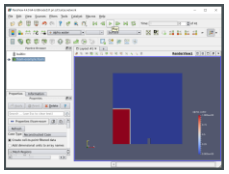
Paraview Client / Server Method

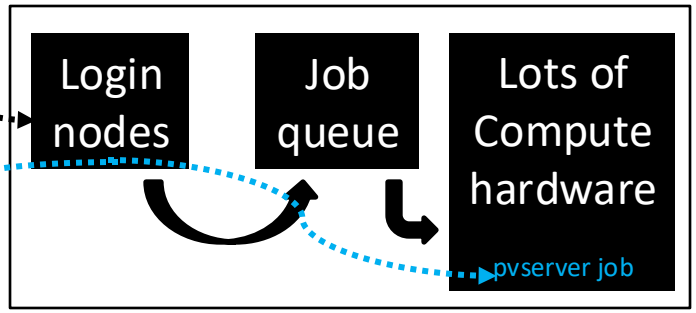
- Client/Server apps are familiar - EG: web browser/server
- Paraview can be used in a client/server manner
- Datasets *remain* on CSF & are rendered there in a (large, parallel) `pvserver` batch job.
- You run the `paraview` GUI (the client) on *your* PC/laptop
- Commands from GUI client are sent to server job. Rendered images from job are sent back to the GUI.
  - https://ri.itservices.manchester.ac.uk/csf3/software/applications/paraview/

1. Login to CSF and submit a (large parallel) `pvserver` job as usual

2. Login to CSF again, "tunnelling" to the job's node

3. Run `paraview` on PC/laptop, connecting to "tunnel" to *send* commands & *receive* rendered images





Paraview Client/Server Method

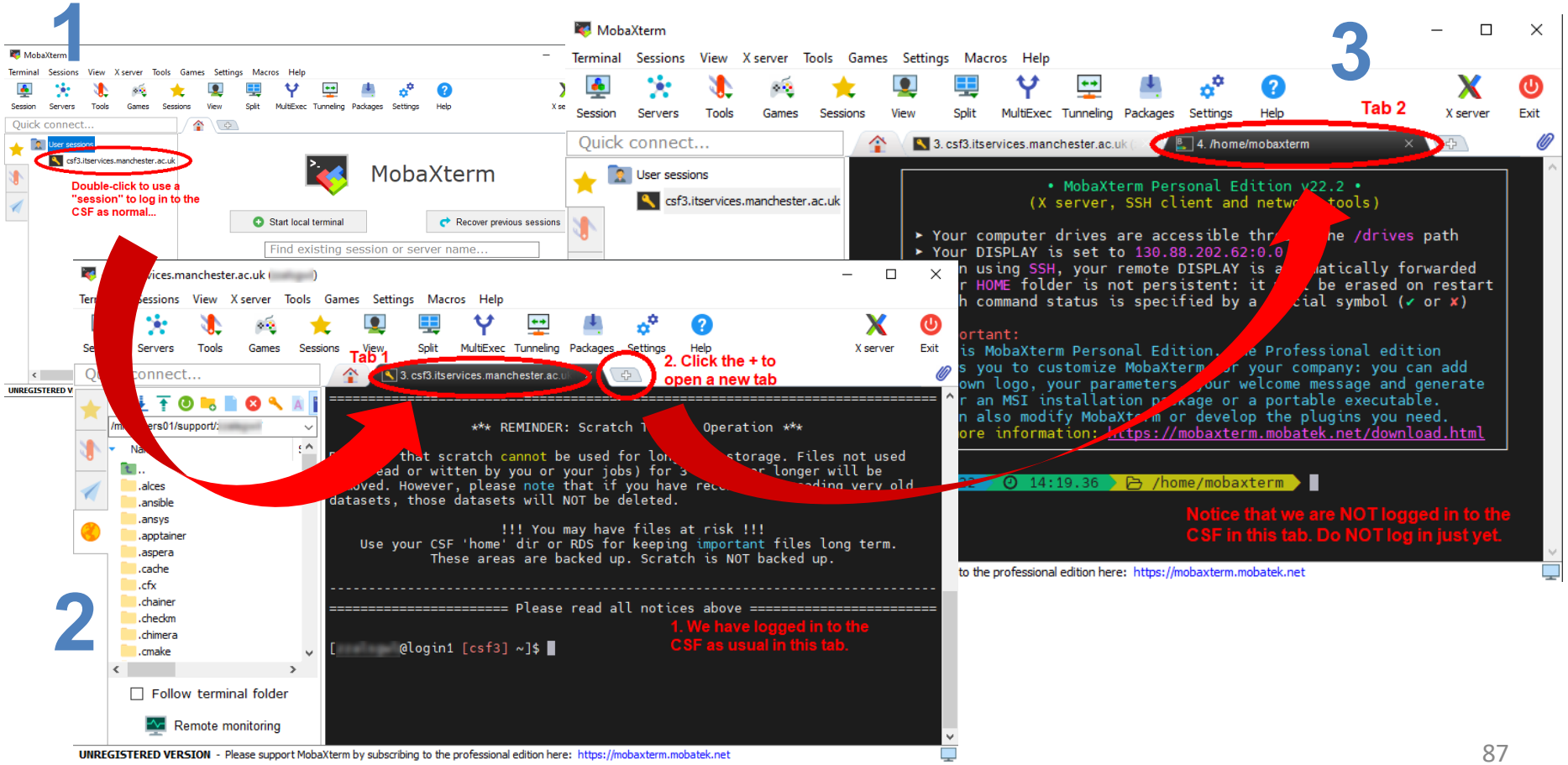
- If you don't already have paraview on your PC/laptop you'll need to download v5.10.1 from:

  https://www.paraview.org/download/

  - Please choose **v5.11** from the drop-down (NOT v5.12 today)
    - If a newer version becomes available on CSF in future, install that version. Check!
    - The version on your PC must match that on the CSF
  - Then your OS/platform from the "Windows Linux Macos" buttons
  - Look for `ParaView-5.11.2-`*`XXX`*`-Python3.9-`*`XXX`* packages
  - You DO NOT need the MPI version if given a choice (Windows)
  - On Windows, the .zip packages DO NOT need administrator rights
- Install it and make sure you can start it however you do that in your OS (Start Menu? Dock/App drawer? Command-line?)
- Exit paraview. We will start it later…

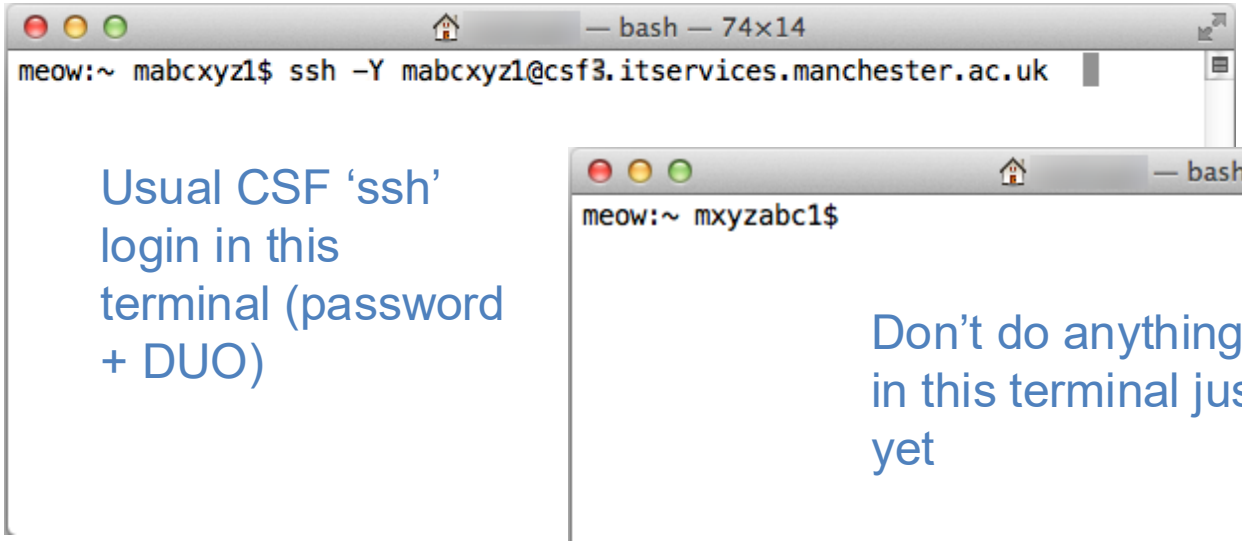
Paraview Client/Server Method

- Now open two terminal windows on your PC / laptop
  - MobaXterm: Open two *tabs* (like a web-browser)
    - Use "saved session" to open the first tab (will ask for password + DUO)
    - Then click the button to open another (but don't do anything there just yet)

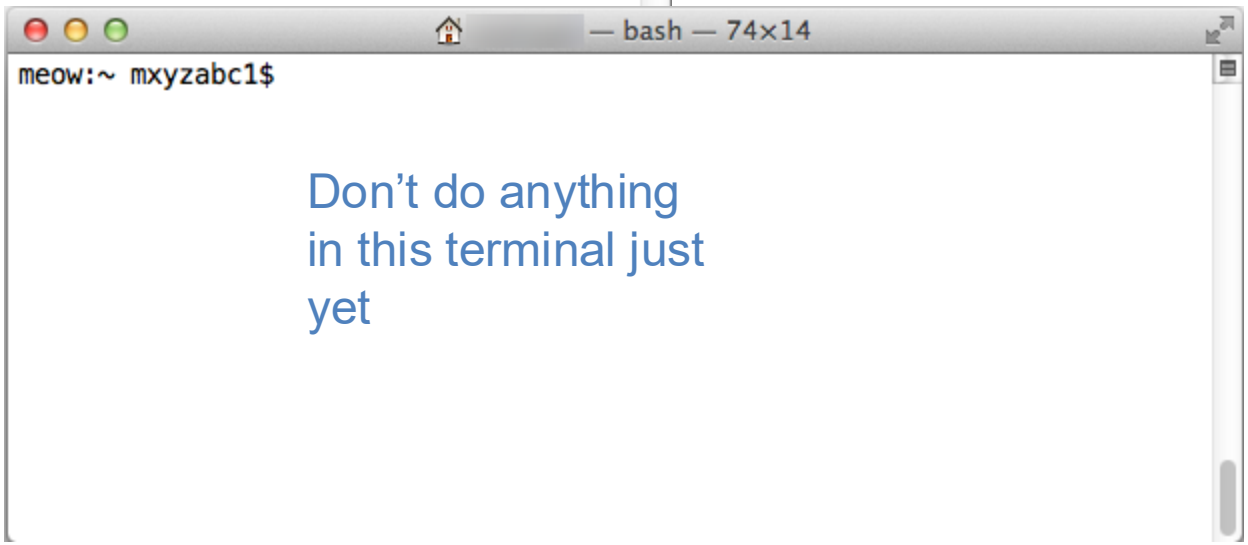


Paraview Client/Server Method

- Open two terminal windows on your PC / laptop
  - MacOS and Linux: Open two *terminal* apps
  - Login in to CSF3 in **one** of the terminals using the usual ssh command (with password + DUO)



Usual CSF 'ssh' login in this terminal (password + DUO)



Don't do anything in this terminal just yet

Paraview Client/Server Method

- In the **first terminal** that *is now logged in* to the CSF
  - Submit a job to run the `pvserver` app (part of ParaView)
  - A flag is added to `pvserver` to allow rendering in batch jobs (see below)
  - Default port is 11111. Add `-p N` to change (job fails if 11111 already in use)
  - Job can be a single-core or multi-core or large multi-node job
  - The bigger your dataset the bigger the job you'll need
  - Note: You must use `mpiexec`, not `mpirun` for pvserver jobs
  - In this example we use a single-node 8 core job
  - Submit the job using `sbatch pvserver.sh` (for example)

pvserver.sh

```
#!/bin/bash --login
#SBATCH -p multicore    # AMD nodes (2-168 cores) [8GB per core]
#SBATCH -n 8            # Number of cores
#SBATCH -t 4-0          # Wallclock time limit (4-0 is 4 days)
#SBATCH --reservation=course   # ONLY FOR TODAY - REMOVE OTHERWISE
# Set up to use paraview tools
module purge
module load apps/binapps/paraview/5.11.2

# $SLURM_NTASKS is automatically set to the number of cores above
mpiexec -n $SLURM_NTASKS  pvserver --force-offscreen-rendering
```

Paraview Client/Server Method

- In the **first terminal** that is logged in to the CSF
  - Wait for the job to **run** (check using `squeue`)

```
# Run this in the terminal that IS logged in to the CSF
squeue

7347222 0.0000009 multicore pvserver.sh mabcxyz1 R ......... 8 node1251
```

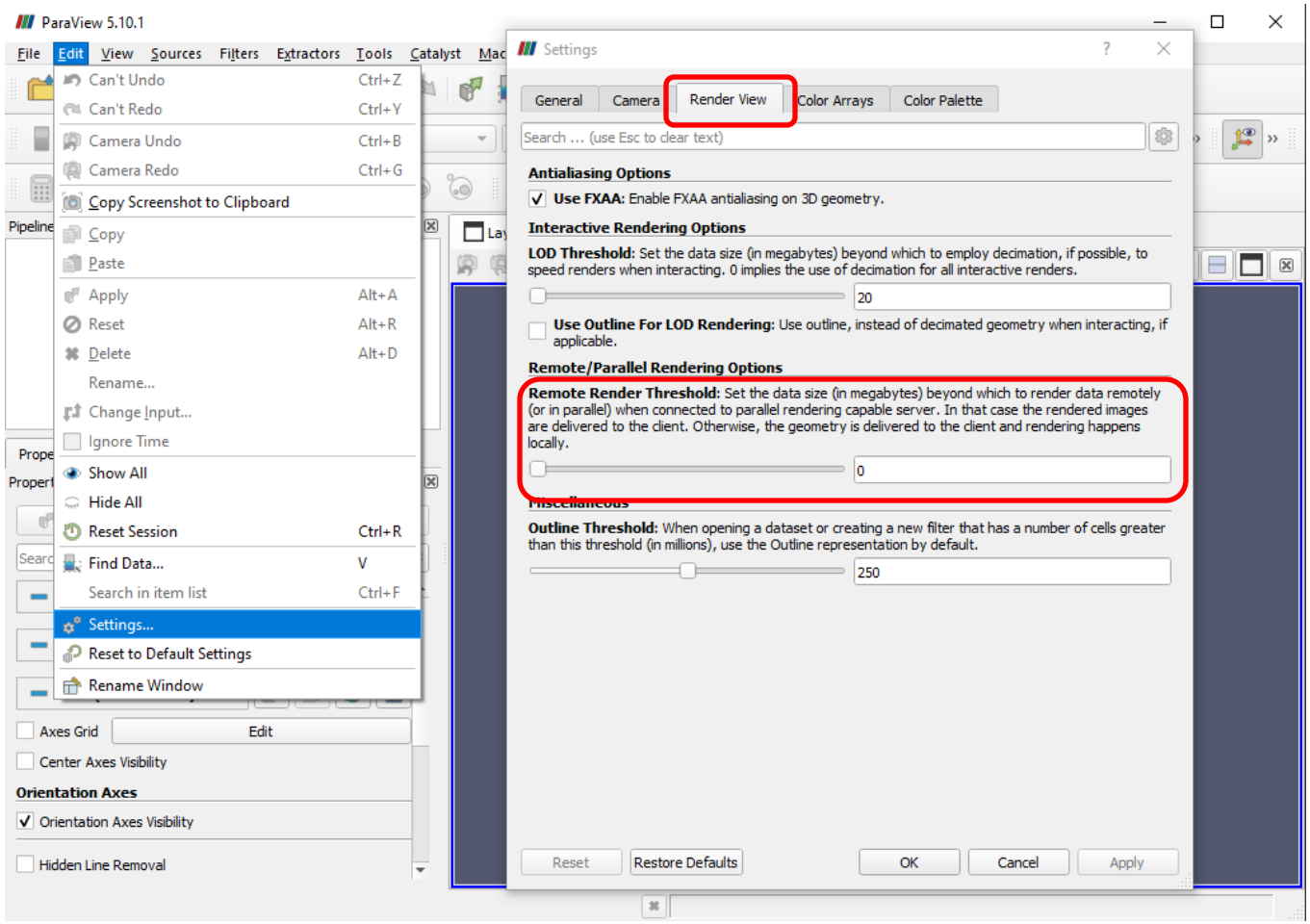
  - Make a note of the `node`*NNNN* part of the queue where your job is **r**unning
  - **`node1251`** in this example but your node may be something else

- In the **second terminal** (*not* currently logged in to the CSF)
  - Log in with some extra options to connect port 11111 on your PC/laptop to port 11111 on the node where the pvserver job is running
  - This will allow your PC/laptop to communicate with the pvserver job!
  - Use *your* **username** and the **node name** you noted above!

```
# Run this in the terminal NOT logged in to the CSF (type carefully!)
ssh -L 11111:node1251:11111 mabcxyz1@csf3.itservices.manchester.ac.uk
Password:
DUO (option 1-1) …
```

Paraview Client/Server Method

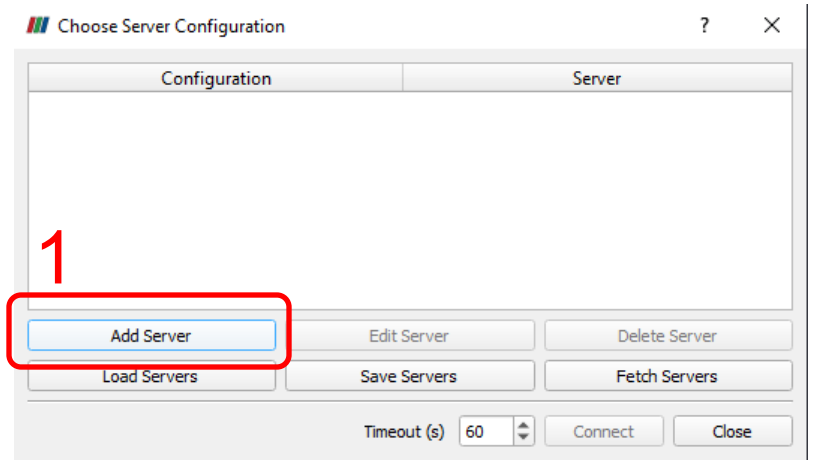
- Now run paraview on your PC/laptop in the usual way
- Then one-time setup of paraview: Edit > Settings > Render View
  - On a Mac it is: ParaView > Preferences > RenderView
  - Set the **Remote Render Threshold** to zero (so all rendering done on CSF)

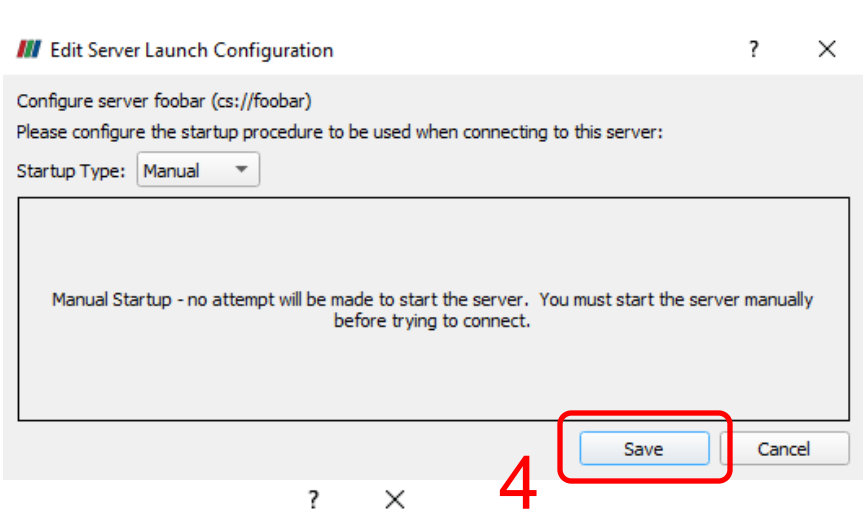


Paraview Client/Server Method

- One-time setup of paraview: File > Connect or use the icon:

- Create a new entry for a "server" using localhost:11111
  - Recall, port 11111 is the default used by `pvserver` (see batch job!)
  - You may have used another port in the job if someone else running p/v

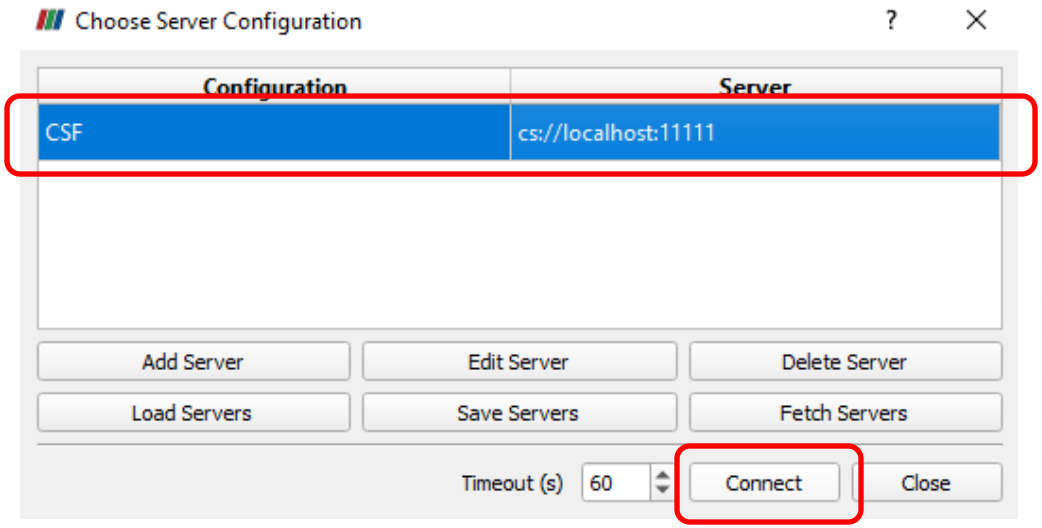




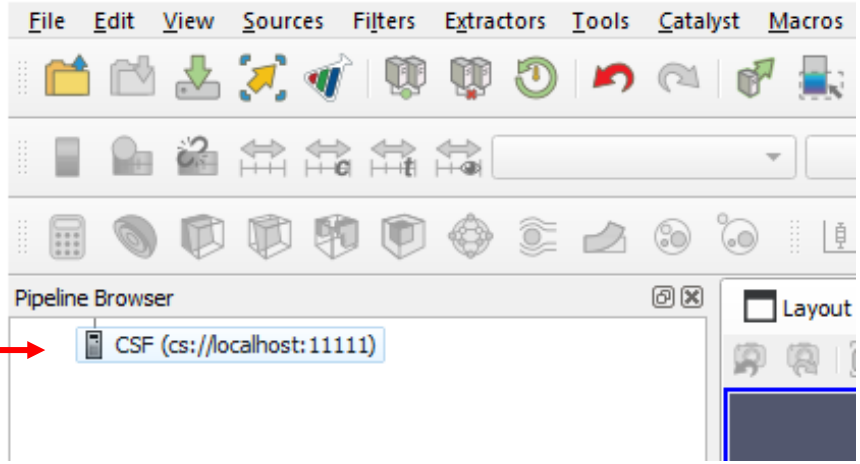
Paraview Client/Server Method

- Now “connect” to the server (always start here from now on)

  - This will ask paraview to connect to a server (the pvserver app) via localhost:11111
  - localhost:11111 means port 11111 on **your** PC/laptop
  - But remember that earlier we used ssh to “tunnel” your PC/laptop port 11111 to port 11111 on the node in the CSF where the pvserver job is running.
  - So even though paraview thinks it is talking to a local pvserver, it is actually talking to your CSF job!



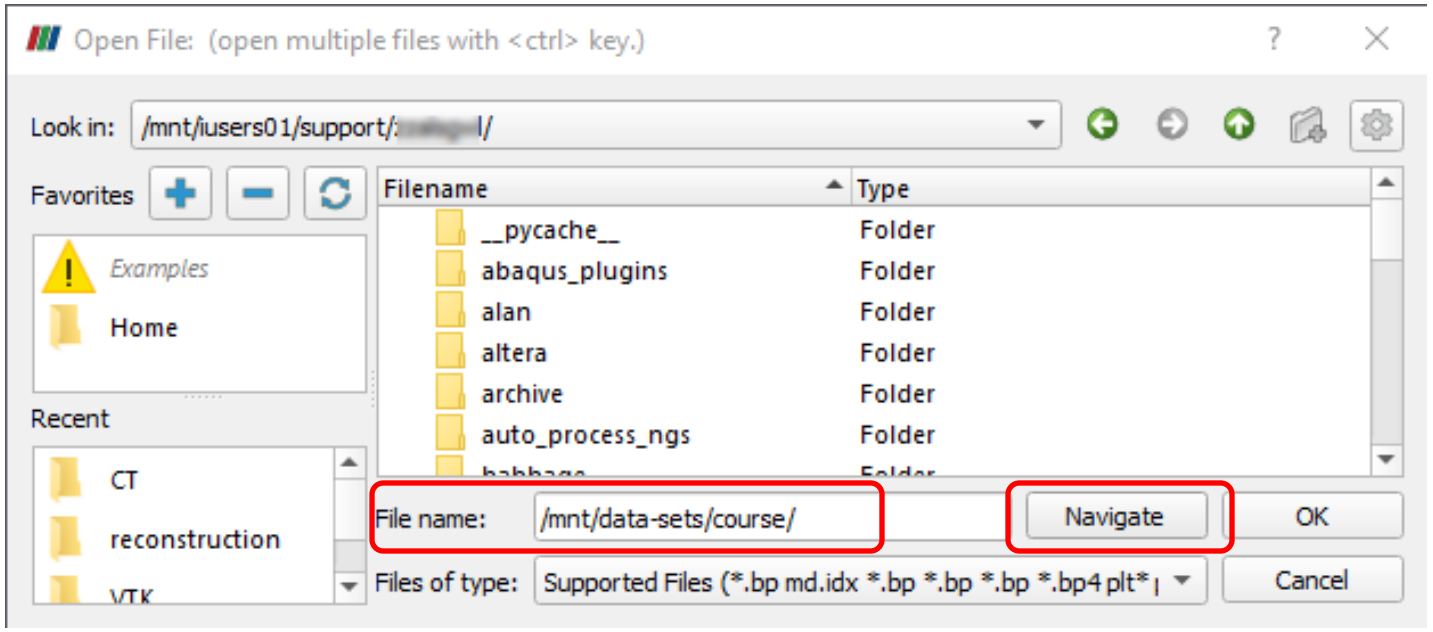
  - The Pipeline Browser in the GUI now shows we are connected to a server (can take ~ 30s to establish connection)



Paraview Client/Server Method

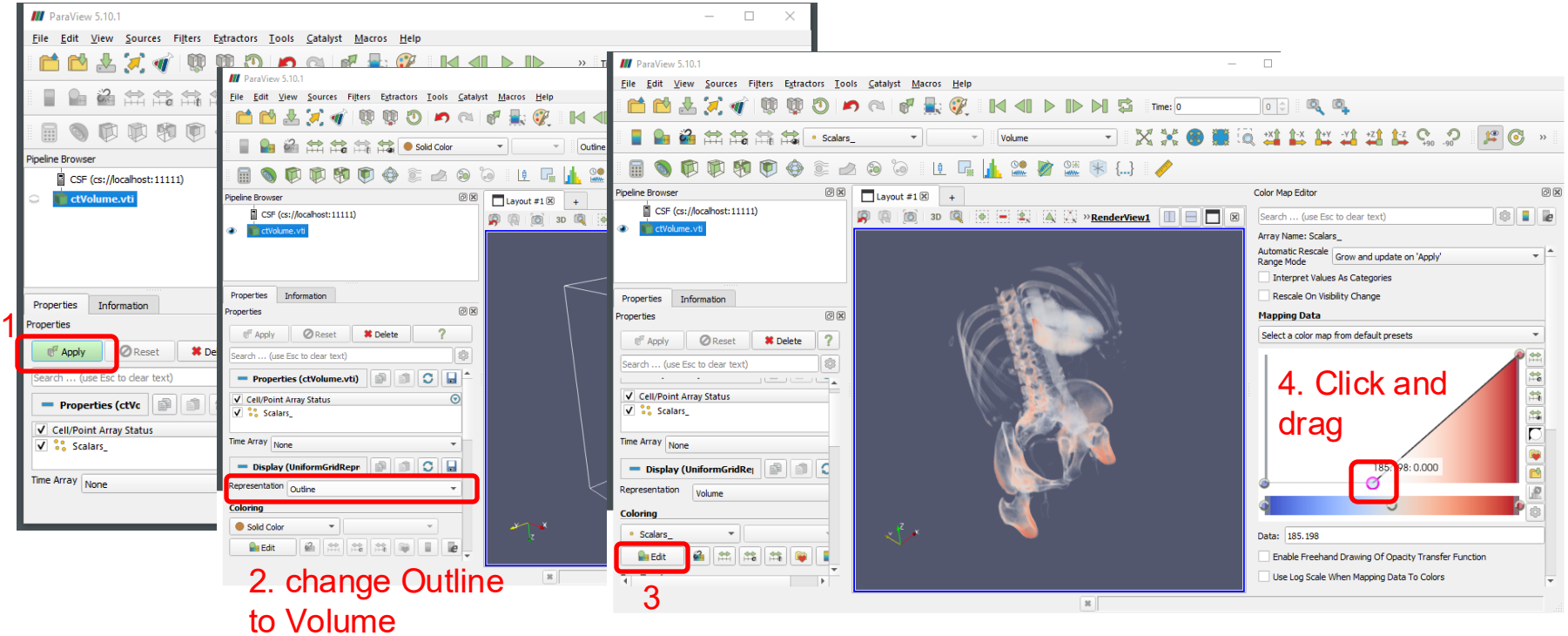
- We can now load data in paraview to visualize it

  - Notice the file-browser shows your **CSF** files (*not* your local PC/laptop files.)
  - This is exactly what we went - datasets remain on the CSF and are processed there in our `pvserver` job.
- Type in the `File name` box: `/mnt/data-sets/course/` and click `Navigate`. This will take you into that folder.
- Then double click on the `mace` folder and open the `.vti` file.



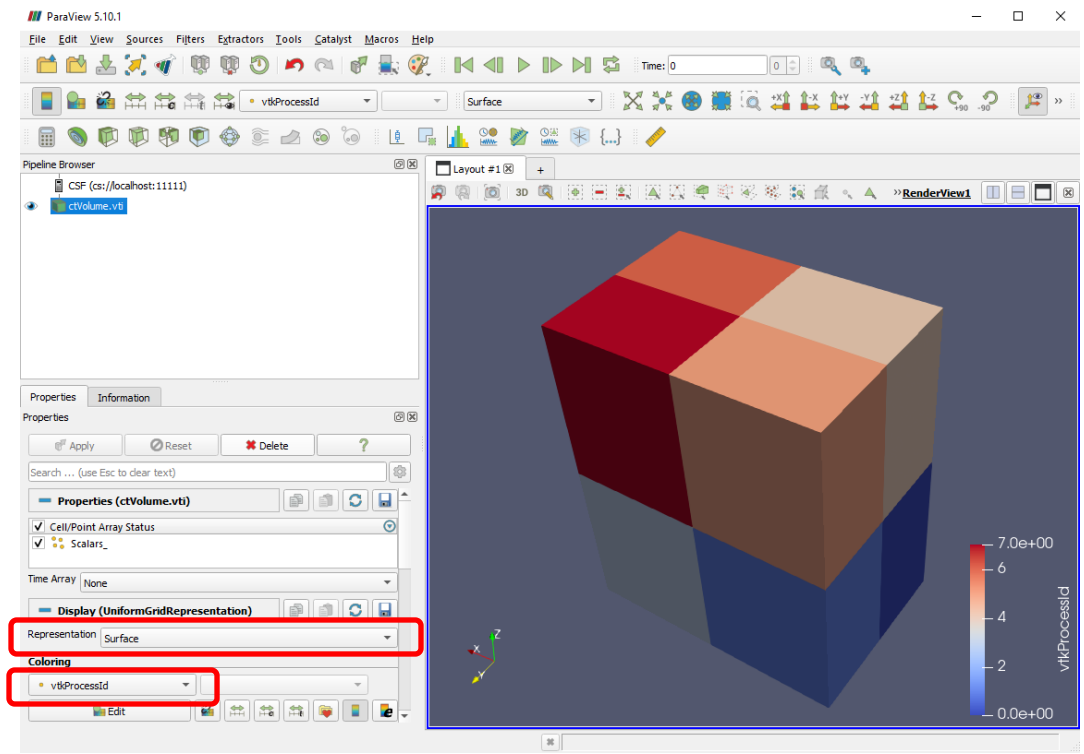
Paraview Client/Server Method

- Press the green Apply button below the Pipeline browser to render the data (displays as a wire-frame bounding box initially)
- In the `Properties` area, scroll down and change the `Representation` from `Outline` to `Volume`
  - Proceed if warned about memory / processing time.
- In the `Properties` area, scroll down to `Coloring` and press `Edit` to open the `Color Map Editor`. Try changing the graph by clicking to create new points and dragging the points.



Paraview Client/Server Method

- To check that the dataset is distributed to multiple processes
  - Change the `Representation` to `Surface`
  - Change the `Coloring` to `vtkProcessID`
  - Will see a block per core being used by our CSF `pvserver` job (8 in this case)
- Not all data files can be distributed amongst parallel processes
  - But can still use client/server method to avoid data downloads. See:
  - https://www.paraview.org/Wiki/ParaView/ParaView_Readers_and_Parallel_Data_Distribution



Paraview Client/Server Method

- Exiting the paraview GUI will terminate your CSF job
- If your CSF job exits (e.g., reaches its time limit) then the paraview GUI will inform you it needs to exit.
  - Save your work / viz often. Saved files will appear on the CSF.
  - If paraview GUI not behaving, check your CSF pvserver job is still running!

- Don't forget to log out of the two terminal windows when done
  - The one where you submitted the batch job
  - The one where you tunnelled to the compute node
  - If you submit a new pvserver job, you'll need to create a new tunnel even if it lands on the same node as before.
- The pvserver job does the "heavy lifting."
  - pvserver jobs can use a lot of cores and/or target the high-memory nodes. They are just batch jobs like all other batch jobs after all.
- Once you've done the logins / job submissions / connect-to-sever a few times you'll find it quick and easy to viz your data without having to download it from the CSF!!

GPU JOBS

Nvidia GPUs

- CSF3 has 164 x Nvidia GPUs
  - A100(80GB), L40S, (and some A100(40GB) for a specific research group) are available to all users.

84 x Ampere A100 GPUs in total – 4 GPUs/node
80GB(**76x**) or 40GB(**8x**) GPU mem, Mem b/w 2TB/s
6912 CUDA cores (108 Multiprocessors, 64 cores/MP)
432 Tensor cores
Peak FP64 9.7 TFLOPS
48-core AMD Epyc "Milan"
512GB RAM host node
40GB A100 nodes restricted to one research group

84 x Ada Lovelace L40S GPUs in total – 4 GPUs/node
48GB GPU memory, Mem bandwidth 864GB/s
18176 CUDA cores (142 Multiprocessors, 128 cores/MP)
142 RT cores
568 Tensor cores
Peak SP 91.6 TFLOPS
48-core Intel Xeon Gold "Sapphire Rapids"
512GB RAM host node

Parallel Jobscript – multi-*node*

gpuA, gpuL, or **gpuA40GB** is the **partition** name. It means: app will use a GPU compute node containing the indicated type of GPU - A100, L40S or gpuA40GB (this one has restricted access.)

-t wallclock time limit for the job. Max in **gpuX** is 4 days.

```
#!/bin/bash --login
#SBATCH -p gpuX      # Partition (A,L)
#SBATCH -G 2         # GPUs: 1-4
#SBATCH -n 8         # Cores: 1-8,12
#SBATCH -t 4-0       # Max is 4-0 here
# Set up to use the CUDA toolkit
module purge
module load libs/cuda/12.4.1

# Run a GPU app. Slurm will ensure no
# other jobs can use your GPUs.
deviceQuery
```

-G (--gpus=) **2** is the total number of **GPUs** we want to *reserve* in the system.

-n (--ntasks=) **8** is the total number of **host CPU cores** we want to *reserve* in the system.

Slurm will set some environment variables for use in your jobscript:
$CUDA_VISIBLE_DEVICES gives the device IDs (**0** or **0,1** or **0,1,2** or **0,1,2,3**) depending number of GPUs.
$SLURM_GPUS gives the number of GPUs you request on the **-G** line.
$SLURM_NTASKS as previous jobs, the number on the **-n** (host CPU cores).

GPU Limits

- Most users get access to **up to two GPUs** from the gpuA or gpuL partitions, in use at any one time with a max of 4 overall (e.g., 2xA100 and 2xL40S.)
  - This is "free at point of use access", funded by the Research Lifecycle Programme, managed by Research IT.
  - Users from some contributing groups who have funded GPUs may get access to more GPUs.
  - All GPU nodes contain 4 GPUs. Multi-node (>4 GPU jobs) are NOT possible.
- CPU host cores are limited by number of GPUs in job and by node-type
  - Note that unless the jobscript contains the `-n` flag, jobs will only have one host CPU core.
  - Many GPU apps can still make use of multiple host CPU cores for some of their processing.

GPU Partition (GPU type)	Host CPU type	Max host CPU cores per GPU	Host RAM per CPU core (GB)	Max host RAM per GPU (GB)
`gpuA` (A100 80GB)	48-core AMD EPYC "Milan"	12	10.4	125
`gpuL` (L40S)	48-core Intel Xeon "Sapphire Rapids"	12	10.4	125
`gpuA40GB` (A100 40GB) (restricted access)	48-core AMD EPYC "Milan"	12	10.4	125

Other GPU Notes

- GPUs are run in **`DEFAULT`** compute mode (not **`EXCLUSIVE_PROCESS`**.)
  - You can run multiple processes / apps on the same GPU – e.g., several small chemistry simulations.

- You can monitor your GPU jobs by accessing the compute node and GPU once your job as started.
  - Use the **`srun`** command on the login node to "login" to the compute node and resource container where your job is running:
    `srun --jobid=`*`JOBID`*` --pty bash`
    (wait until you are logged into the compute node where you job is running. You'll see the same GPUs.)
    `nvidia-smi`
    or, for example:
    `module load libs/cuda/12.4.1`
    `ncu-ui` or **`nvvp`** (or other Nvidia tool)

Further Info

- CSF Slurm documentation

  https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/
- Job Arrays - multiple similar jobs from a single submission script

  https://ri.itservices.manchester.ac.uk/csf3/batch-slurm/job-arrays-slurm/
- SSHFS - another means of file transfer

  https://ri.itservices.manchester.ac.uk/userdocs/file-transfer/
- Virtual Desktop Service – another means of connecting and running GUIs and logging in from off campus

  https://ri.itservices.manchester.ac.uk/virtual-desktop-service/

News

- MOTD when you log into the CSF - please keep an eye on this
- Problems e.g. system down, can’t log in, minor changes to the service:

  https://ri.itservices.manchester.ac.uk/services-news/
- Prolonged problems or major changes to the emailed to all users

Need more help with the CSF?

- Extensive documentation about all aspects of the service:

  https://ri.itservices.manchester.ac.uk/csf3/

- Contact the Research Infrastructure Team

  https://ri.itservices.manchester.ac.uk/hpc-help

  Please log a ticket through which we'll provide assistance.

Thank you! Questions?