

Welcome!

Virtual tutorial starts at 15:00 GMT



PBS Job Submission

ARCHER Virtual Tutorial, Wed 13th January 2016

David Henty <d.henty@epcc.ed.ac.uk>



EPSRC



NERC SCIENCE OF THE ENVIRONMENT



archer



CRAY
THE SUPERCOMPUTER COMPANY



epcc



Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

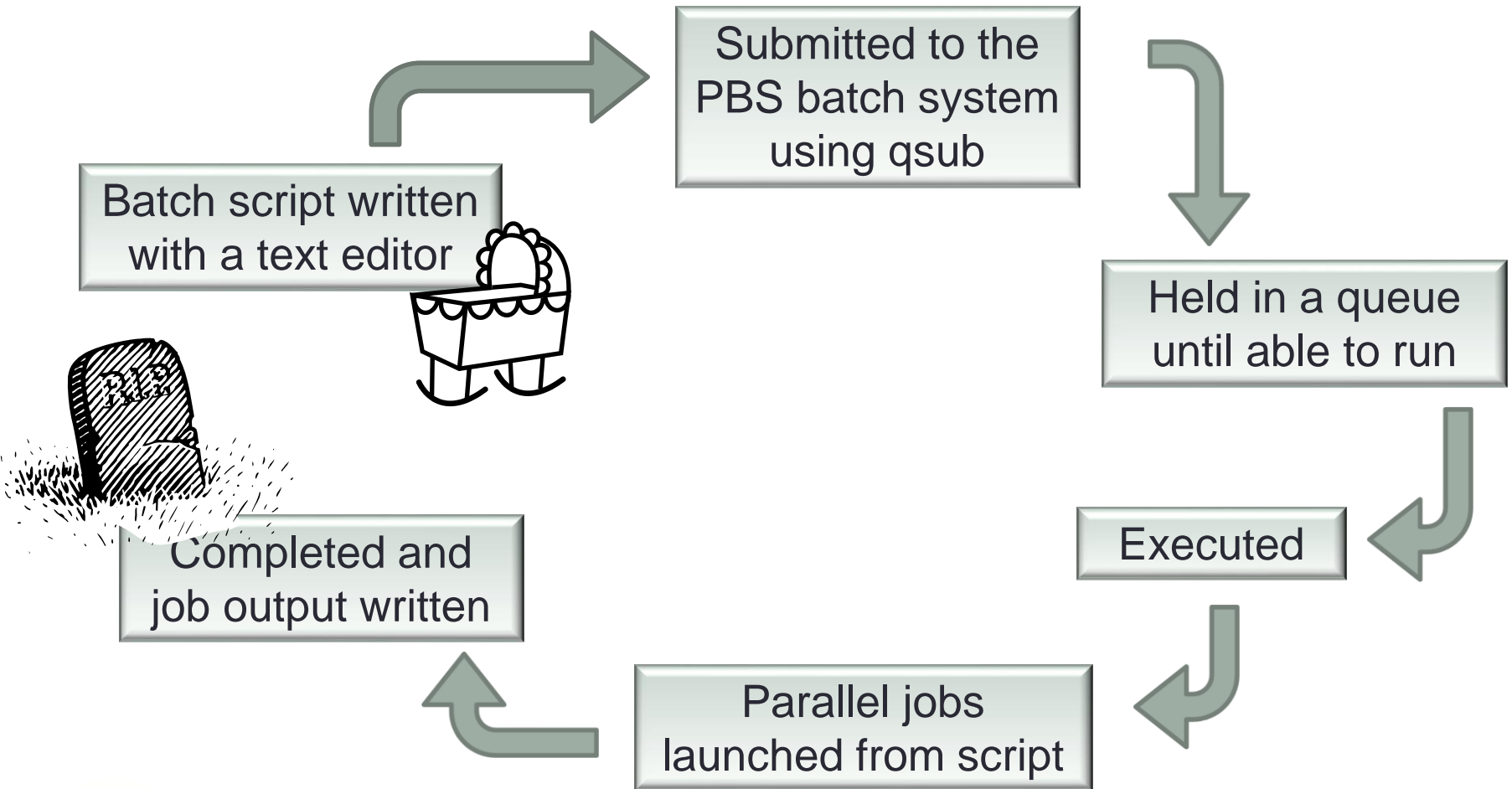
http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.



A day in the life of a PBS job



Conception: batch script written ...

- What is a batch script?
 - a list of commands that are executed in order exactly as if you typed them into the shell on the command line
 - recommended to use bash
- Lines starting “#” are comments
- Except ...
 - `#!` is special to operating system
 - `#!/bin/bash --login` # Run script as if a bash login session
- and
 - `#PBS` is special to batch system
 - `#PBS -N myjob` # Pass on as arguments to qsub



Great! So I can run the batch script in advance to check it works since it's just a bunch of commands

Why?

So some commands that work on the login nodes won't work under PBS?



I'm afraid it's not that simple

Your batch script runs on a different computer in a different environment

Actually, some commands that *don't* work on the login nodes *will* work under PBS

Birth: submitted to batch system

```
user@archer> qsub -l select=6 myjob.pbs  
123456.sdb
```

- PBS takes a *copy* of your batch script and stores it
 - ascertains resource requirements (e.g. no. of nodes)
 - from command line arguments or from #PBS lines
 - other resources: `-l walltime=03:00:00`
- Job is queued until resources are available
 - query status with `qstat -u <myusername>`
 - job status set to “Q”



Resource selection

- Unlike some systems, most jobs go to a single queue
 - some special queues, e.g. for short and long jobs
 - see later
- Jobs **scheduled** entirely on **requested** resources
 - i.e. run time and number of (24-core) nodes
- Can specify high memory, e.g for 6 nodes
 - `#PBS -l select=6:bigmem=true`



Great! So I can edit the same job script and resubmit straight away?

Why?

How does it decide when to run my job?

So I should take care when specifying these?

I'm afraid it's not that simple

You're probably running an executable from the script and it will see the version that's there at run time - OK for a package but not if you're recompiling

It's a balance of your requested number of nodes and the runtime compared to all the other jobs in the system

Yes – don't ask for huge amounts of runtime if you don't need it!

Childhood: job script runs

- A set of compute nodes is reserved for your job
 - but your batch script is actually executed on the **MOM nodes**
 - backronym for “Machine Oriented Mini-server”
 - **qstat** job status set to “R”
- The **only way** to access the compute nodes is with aprun

```
#PBS -N myjob
```

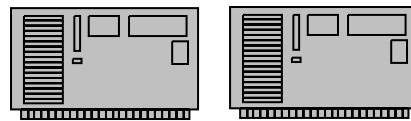
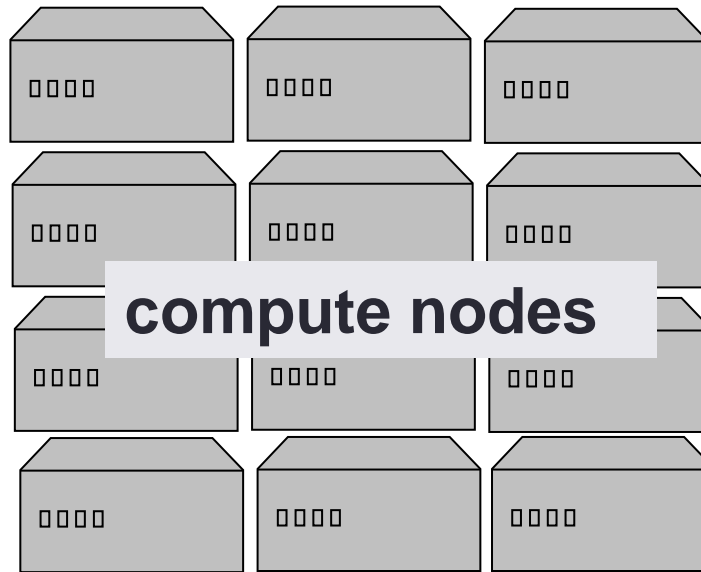
```
...
```

```
# Now run the job (assume I used select=6)
```

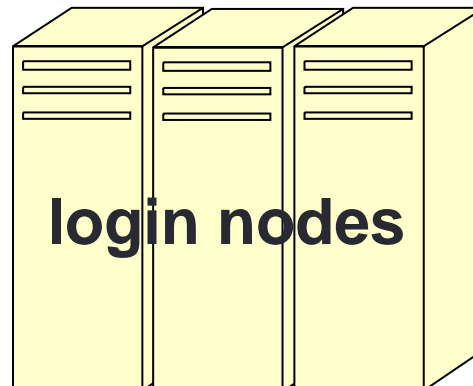
```
aprun -n 144 mympiprogram
```



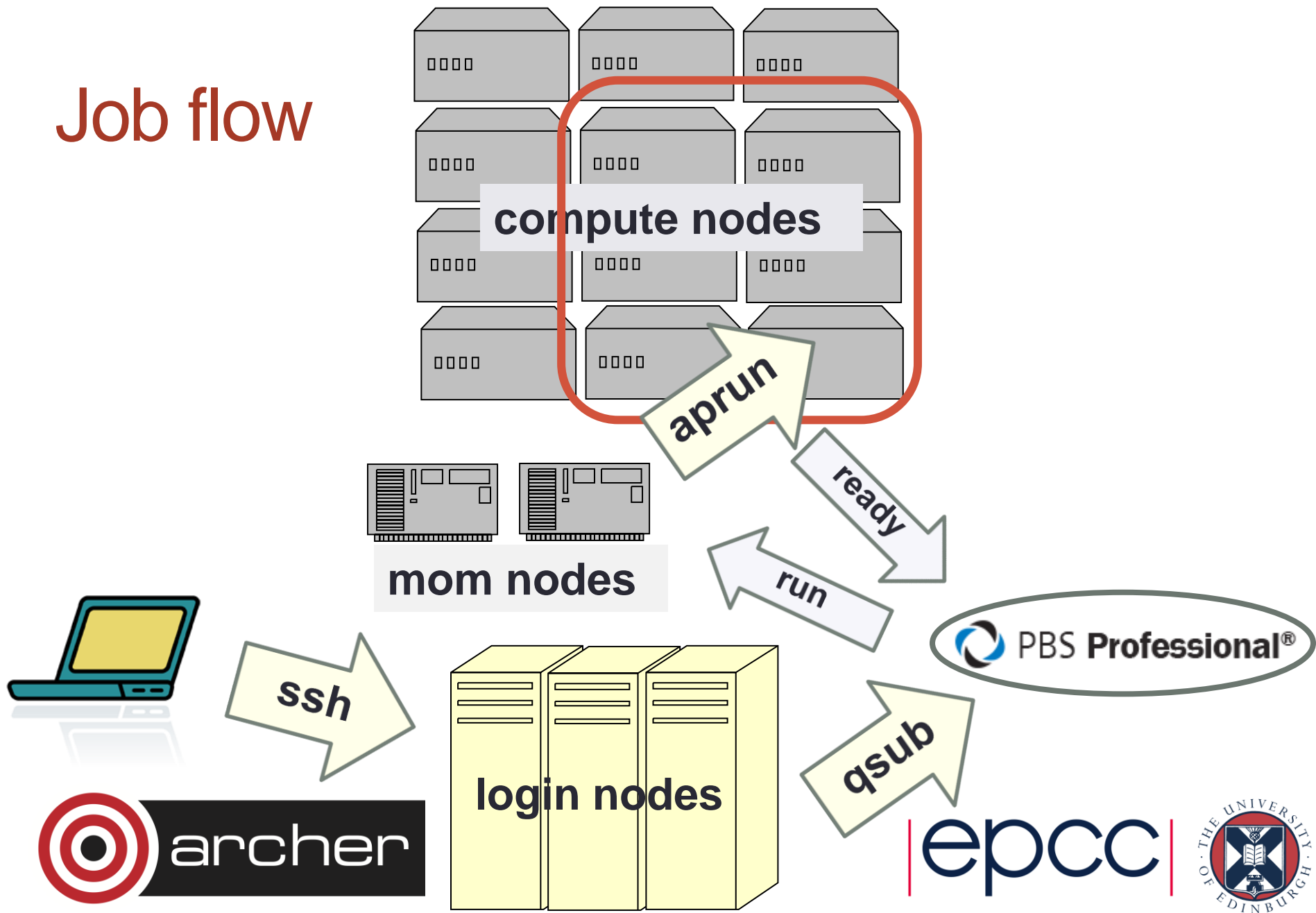
Operating Systems



mom nodes



Job flow

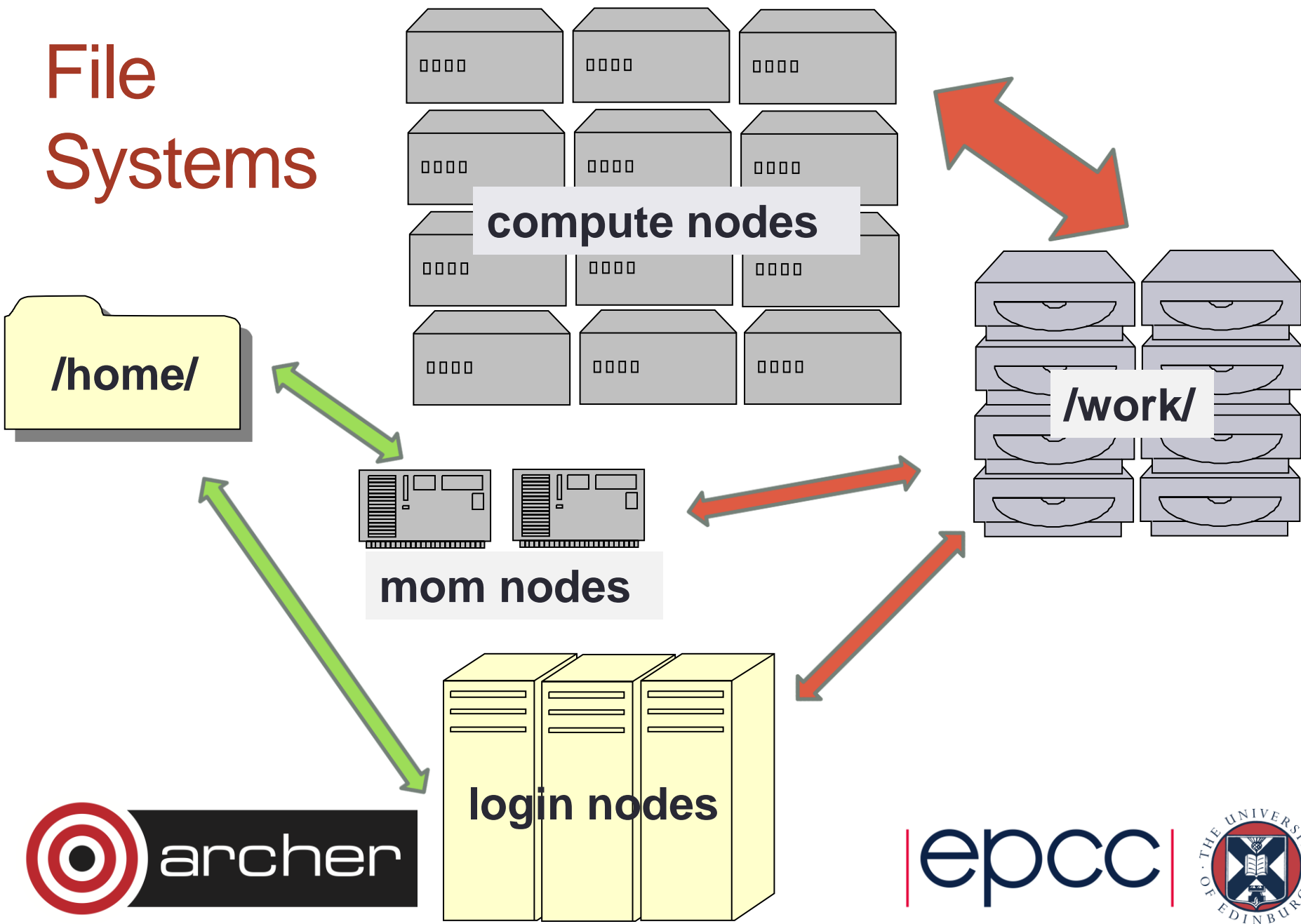


Adulthood: parallel jobs

- Compute nodes reserved for duration of job
 - PBS doesn't care if/how you use them!
 - all commands from batch script executed on MOM node
 - aprun on MOM node causes parallel jobs to run on compute nodes
- Do production runs in `/work/` filesystem, not `/home/`.
 - `$PBS_O_WORKDIR` is where the job was submitted from
- `aprun` does the following
 - **broadcasts the executable** to all the compute nodes
 - **gathers** the standard outputs / errors from all the PEs



File Systems



Retirement: the end of your job

- Job finishes
 - after the all the commands in script have been executed ...
 - ... or the wallclock limit is exceeded
- All running parallel jobs are killed
 - e.g. wallclock exceeded or aprun running in background (see later)
- Standard outputs collated
 - written to `myjob.o123456`
- `qstat` job status may be set to “E” for a few minutes
 - job is “Exiting” (not “Exited”) but script has finished



Charging

- You are charged for the number of nodes you requested
 - regardless of whether you actually used them
 - minimum allocation is a node; reserved exclusively for single user
- You are charged for the amount of time your job ran
 - regardless of how much time you requested

- **Classic error!**

```
#PBS -l select=32          # ask for 32 nodes
aprun -n 32 ./mympijob     # run on 32 cores - wrong!
```

- MPI job runs on first node and 8 cores of second node
 - but you are charged for all 32 nodes (768 cores)!
 - should be `aprun -n 768` (or `select=2` if you really want `aprun -n 32`)



aprun

- Can issue multiple aprun's in a single job
 - single job + many aprun's may be better than many jobs

```
#PBS -l select=6                # 6*24 = 144 cores
```

```
...
```

```
aprun -n 144 mympiprogram dataset1
```

```
aprun -n 144 mympiprogram dataset2
```

```
aprun -n 144 mympiprogram dataset3
```

```
...
```

aprun is quite clever

- Can manage multiple parallel jobs simultaneously
 - e.g. imagine we had 4 runs each using 72 cores

```
#PBS -l select=6                # 6*24 = 144 cores
```

```
...
```

```
aprun -n 72 mympiprogram dataset1
```

```
aprun -n 72 mympiprogram dataset2
```

```
aprun -n 72 mympiprogram dataset3
```

```
aprun -n 72 mympiprogram dataset4
```

```
# Incorrect! - all these run sequentially
```



Multiple aprun's in the background (i)

```
aprun -n 72 mympi program dataset1 &  
aprun -n 72 mympi program dataset2 &  
aprun -n 72 mympi program dataset3 &  
aprun -n 72 mympi program dataset4 &
```

```
# Incorrect: "Job finishes after the all the  
# commands in script have been executed".  
# Final aprun returns immediately, script  
# reaches end and finishes, aprun's killed.
```

Multiple aprun's in the background (ii)

```
aprun -n 72 mympiprogram dataset1 &  
aprun -n 72 mympiprogram dataset2 &  
aprun -n 72 mympiprogram dataset3 &  
aprun -n 72 mympiprogram dataset4
```

```
# Incorrect: script finishes when dataset4  
# finishes, but other datasets may still be  
# running at that time.
```

Multiple aprun's in the background (iii)

```
aprun -n 72 mympiprogram dataset1 &  
aprun -n 72 mympiprogram dataset2 &  
aprun -n 72 mympiprogram dataset3 &  
aprun -n 72 mympiprogram dataset4 &
```

```
wait
```

```
# Correct! "wait" blocks until all spawned  
# processes are complete
```



Task farms with aprun

```
aprun -n 72 mympi program dataset1 &  
aprun -n 36 mympi program dataset2 &  
aprun -n 72 mympi program dataset3 &  
aprun -n 36 mympi program dataset4 &  
aprun -n 36 mympi program dataset5 &
```

```
wait
```

```
# ordering might be: 1, 2, 4, 3, 5
```



why can my job script see my home files but my MPI program can only see my work files?

why can I store my MPI executable in /home but not its input files?

how do "interactive" batch jobs work?

Because the script runs on the MOM nodes which see /home but MPI runs on the compute nodes which only see /work

aprun broadcasts the executable from /home to the compute nodes, but not any dependent files

You are effectively submitting a job which runs a bash shell on the MOM nodes

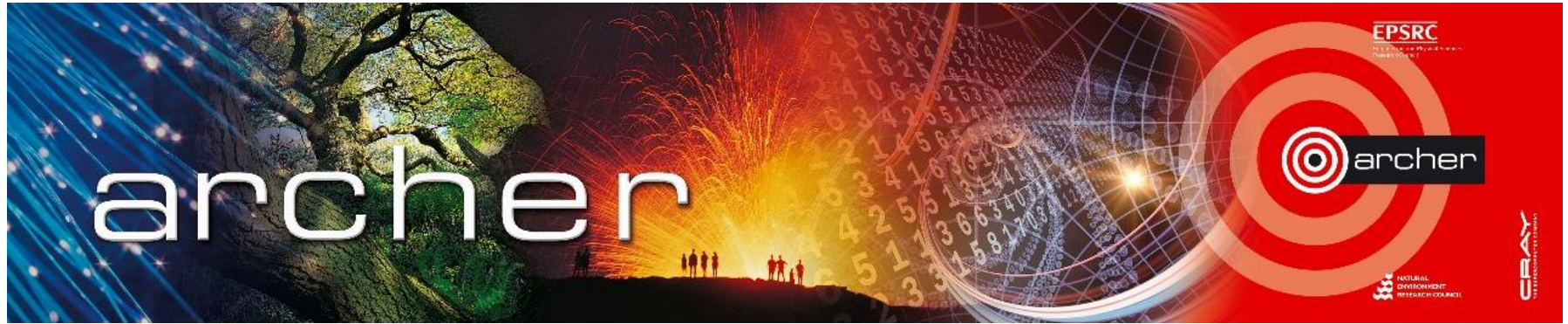
Special queues

- Regular jobs
 - up to 24 hours; 1-4,920 nodes (24-118,080 cores)
- Long jobs (qsub -q long)
 - up to 48 hours; 1-256 nodes (24-6,144 cores)
- Short (debug) jobs (qsub -q short)
 - up to 20 minutes; 1-8 nodes (24-192 cores)
 - only available 0900-1700 UK time Mon-Fri.
- Low priority jobs (qsub -q low)
 - up to 3 hours; 1-512 nodes (24-12288 cores)
 - uncharged, but only enabled when machine is lightly used
- Serial jobs (qsub -q serial)
 - run on separate postprocessing nodes – see documentation for details



THE END





Goodbye!

Virtual tutorial has finished

