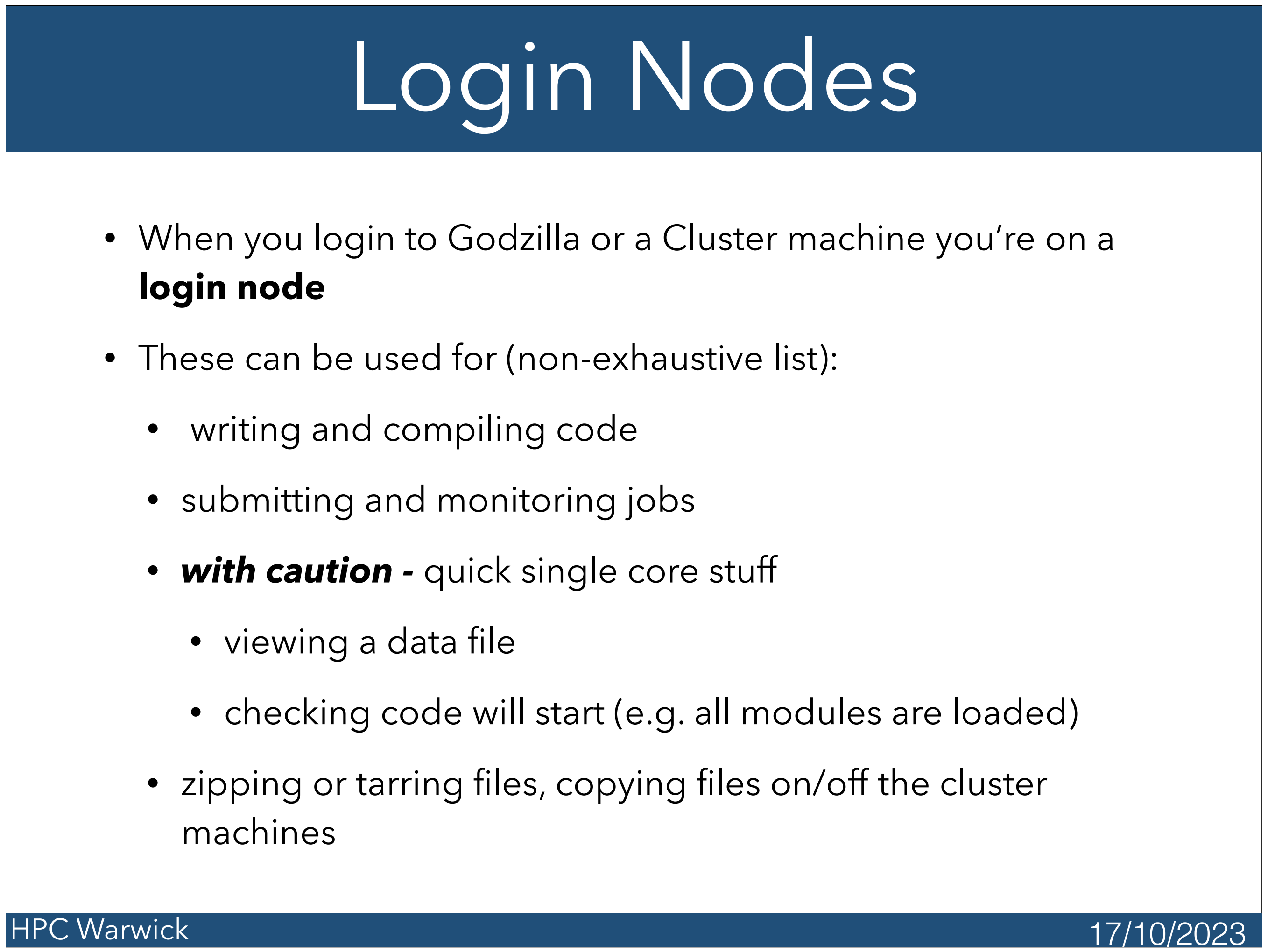
HPC at Warwick and Beyond
Section 2 - Running Jobs

Warwick RSE

Updated for 2023

Queues and Jobs



Login Nodes

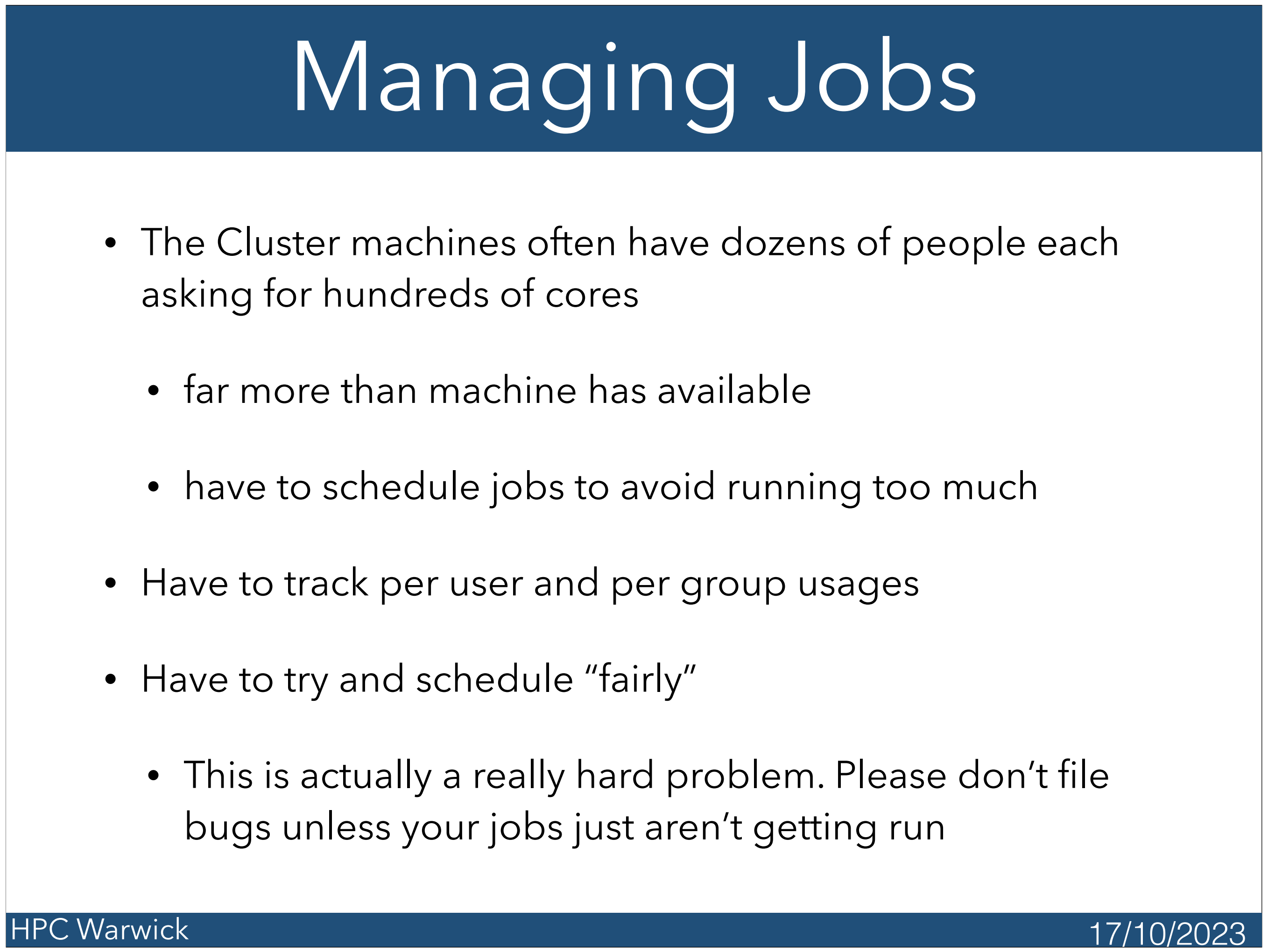
- When you login to Godzilla or a Cluster machine you're on a **login node**
- These can be used for (non-exhaustive list):
  - writing and compiling code
  - submitting and monitoring jobs
  - ***with caution -*** quick single core stuff
    - viewing a data file
    - checking code will start (e.g. all modules are loaded)
  - zipping or tarring files, copying files on/off the cluster machines

When you login to a cluster, there's a reminder that you must not run jobs on the login node, as well as reminders of links to the mailing list and wiki. Generally, you should always read the sign-on messages. They can be important!

IMPORTANT!

- Login nodes, including Godzilla, ***MUST NOT be used to run code!!!***
- ***Misusing login nodes will result in account suspension***

If this happens, you should raise a Bugzilla to get reactivated and please try to understand what went wrong - if you ran things simply by mistake don't worry, but do be more careful. If you're not sure whether your use counts as "running" please ask!



Managing Jobs

- The Cluster machines often have dozens of people each asking for hundreds of cores
    - far more than machine has available
    - have to schedule jobs to avoid running too much
- Have to track per user and per group usages
- Have to try and schedule “fairly”
    - This is actually a really hard problem. Please don’t file bugs unless your jobs just aren’t getting run

Our clusters are usually kept 85-95% full of running jobs and nearly always have plenty in the queue waiting.
Point 3, “running too much” means overloading a processor so that it has to work on one job, then swap to another, to the point where running the jobs at the same time takes longer than running them one after another. If you’re interested, look up Context Switching.
Bit more about fairshare later - for instance it is proportional to departmental subscription.

Queues

- We use the Slurm scheduling system
- You submit a job to the system, asking for a given amount of:
  - time
  - processors and/or nodes (if using Hybrid)
  - memory per core
- Your job enters the queue
- Assuming you have hours to spend, your job runs when resources are available
- Sometimes you have to specify a partitition, aka a queue (e.g. devel)

Devel queues (where present) are special - 2 nodes max, 1hr runtime, max 2 simultaneous jobs per user. Gets high priority, intended for quick running while developing. MUST NOT be used for running normal jobs. See https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-reslimits.html#development-queues

Note on Slurm

- Slurm mainly manages resources - it gives you exclusive access to a certain number of CPUs and (where asked for) GPUs
- Slurm **also** runs programs for you on those resources that you requested
- Programs use resources differently

Note on Slurm

- You first ask slurm for **nodes** - the number of computers that you want. Each node in Avon has 48 CPUs and on Sulis has 128 CPUs
- Then you specify **ntasks-per-node**. Each task is a "slot" for running a program in. You can specify up to the number of CPUs on the node here
- Some programs split themselves into **threads** where each program runs on multiple CPUs, in which case you specify **cpus-per-task** to say how many CPUs each program should have
- The product of **ntasks-per-node** and **cpus-per-task** should be the actual number of CPUs on the system

Real docs

- We can only show a handful of simple, common examples
- Full documentation online covers lots more including R, Python and uses that aren't conventional HPC
- https://docs.scrtp.warwick.ac.uk/hpc-pages/avon-slurm-examples/avon.html

Modules

- Unlike starting a job directly, a queued job may not know about the environment when it was submitted
- This means any modules your job needs have to be set in the job script
- For more on modules see https://docs.scrtp.warwick.ac.uk/general-pages/software-pages/modules.html

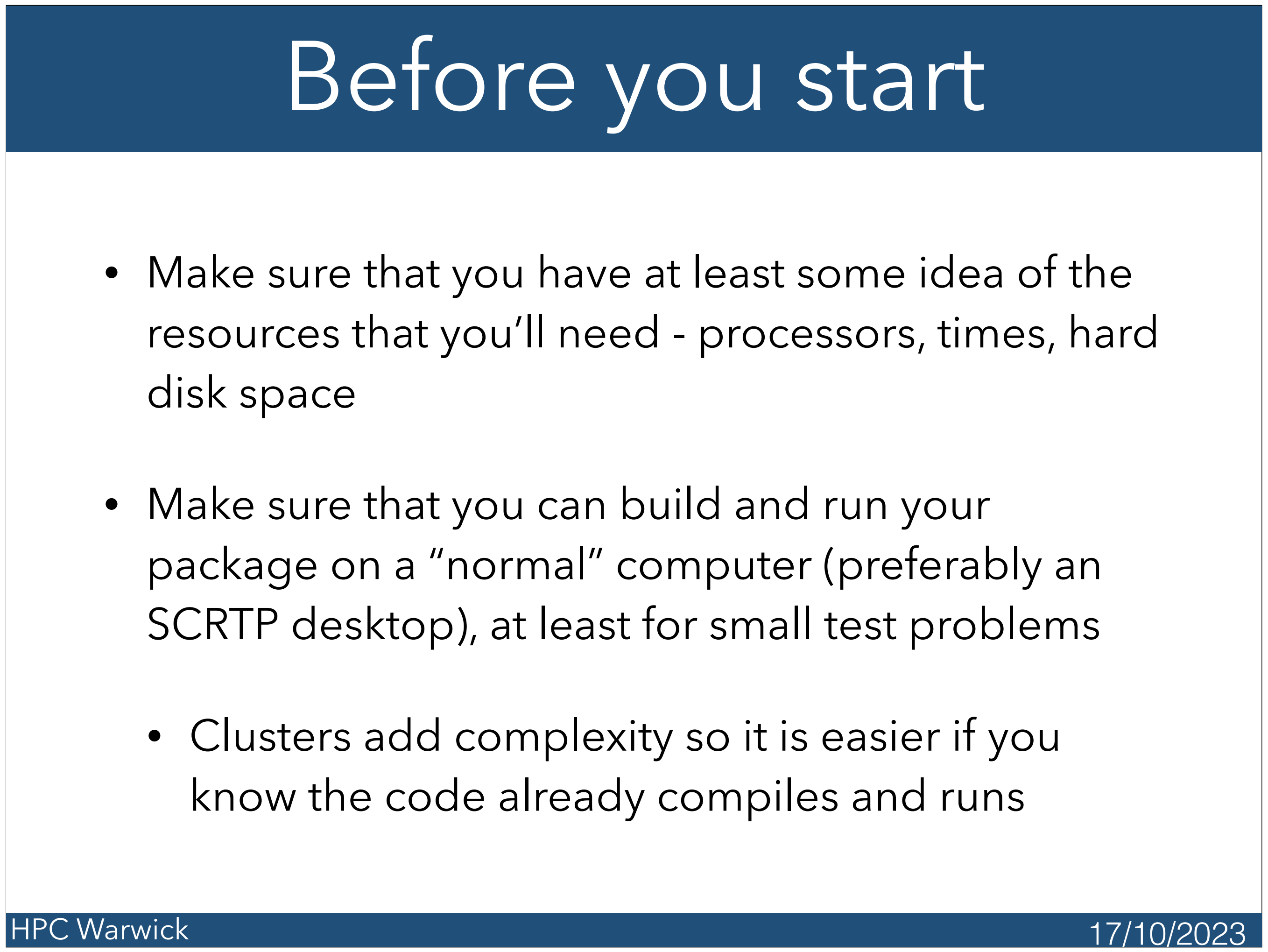
Note that jobs do know the directory they were submitted from, and might know more. If you have a lot of default modules, you might want to add a `module purge` before the `module load` commands in your job scripts, to avoid unexpected module collisions etc

ASIDE - code snippets

- Slides will show some snippets of code
- Usually put them in back-ticks, e.g. `code`
- Going to use Linux man syntax
- `command [-n ***option***]`
  - [] denote an optional piece of command
  - text in ***italics*** should be replaced with required value(s) (using bold italics for ease of viewing)

E.g.
ls [-l] [--sort=***word***]
List files, in long format (-l), sorting by ***word,*** e.g. --sort=time

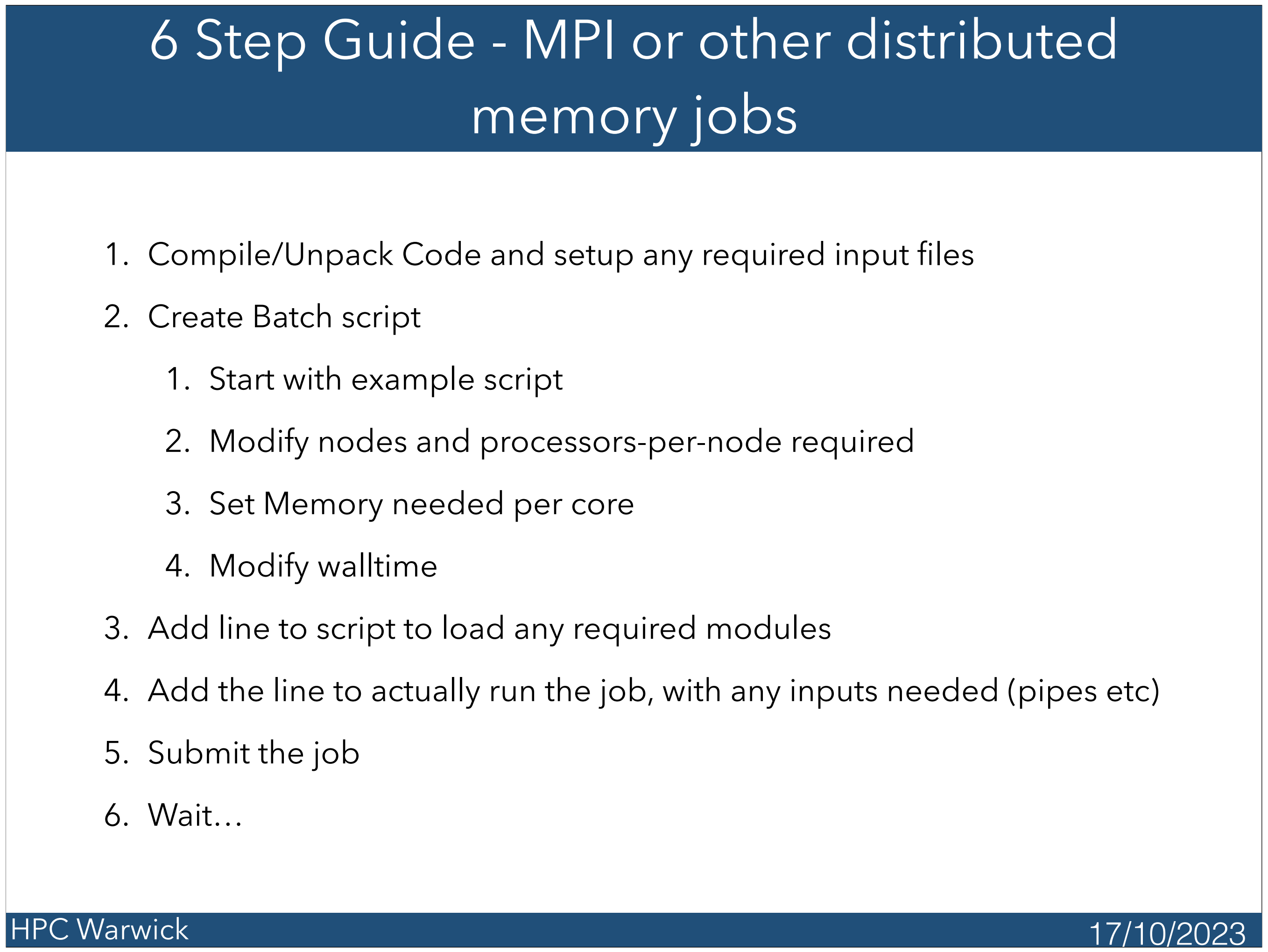
Running Jobs with Slurm



Before you start

- Make sure that you have at least some idea of the resources that you'll need - processors, times, hard disk space
- Make sure that you can build and run your package on a "normal" computer (preferably an SCRTP desktop), at least for small test problems
  - Clusters add complexity so it is easier if you know the code already compiles and runs

If you are using a package that is provided on the system then just jump straight to the cluster - we've already tested that it works



6 Step Guide - MPI or other distributed memory jobs

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Start with example script
    2. Modify nodes and processors-per-node required
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to script to load any required modules
4. Add the line to actually run the job, with any inputs needed (pipes etc)
5. Submit the job
6. Wait…

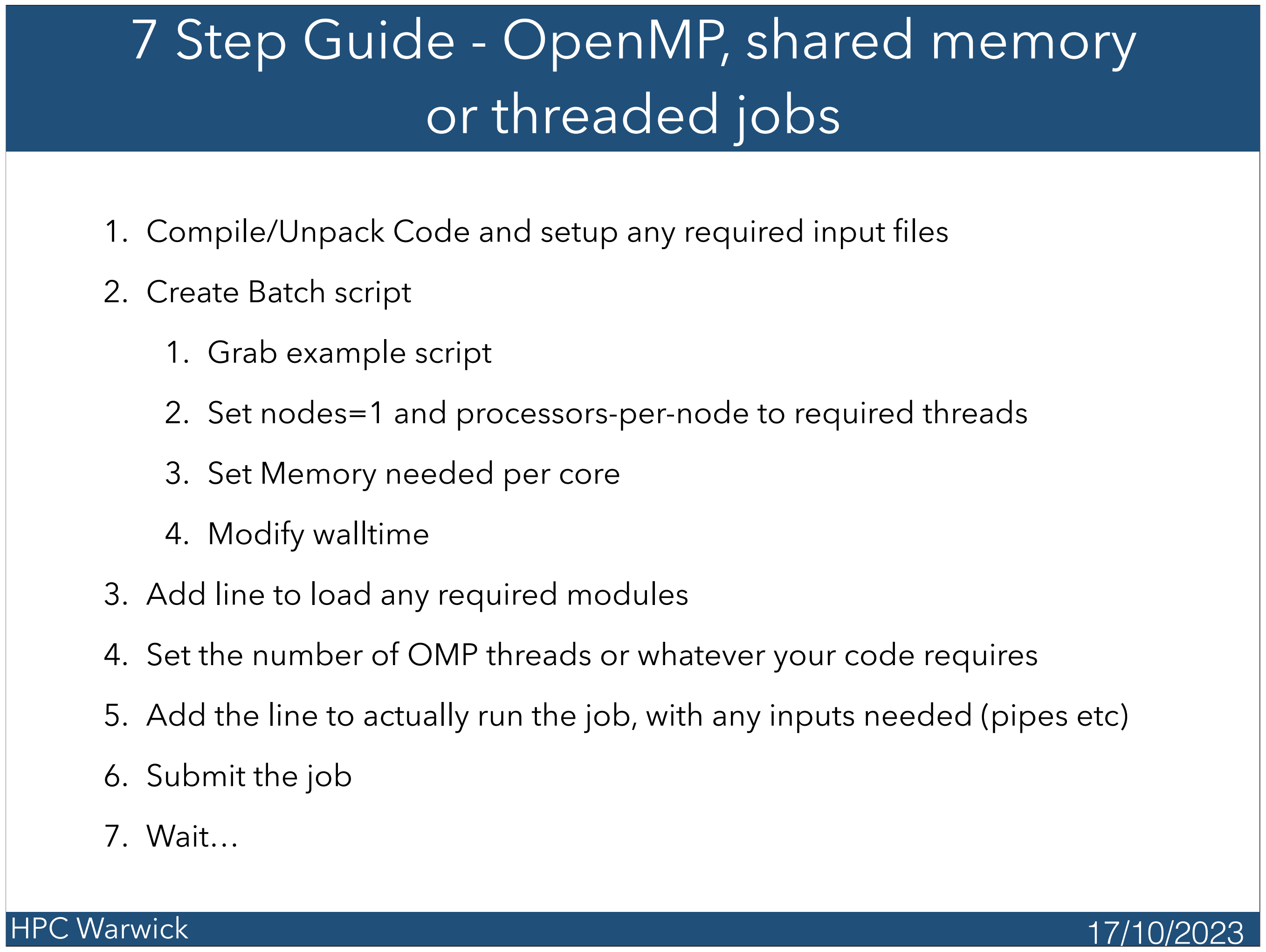
Example scripts at https://warwick.ac.uk/research/rtp/sc/rse/training/hpcbeyond (probably where you got these notes)

Demo

1. You can look at https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-reslimits.html to see what resources are available on a given cluster

2. Walltime format is dd:hh:mm:ss. You can omit larger units if they're zero, so if you put something like 1:30 this is interpreted as 1 minute, 30 seconds

3. Output and Error (stdout and stderr) are written to a file slurm-<job-id>.out in the directory you submit from



7 Step Guide - OpenMP, shared memory or threaded jobs

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Grab example script
    2. Set nodes=1 and processors-per-node to required threads
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to load any required modules
4. Set the number of OMP threads or whatever your code requires
5. Add the line to actually run the job, with any inputs needed (pipes etc)
6. Submit the job
7. Wait…

Note that you rarely want to set more processors per node than there are available

Demo

1. Note that this can use only one node!

2. If you're using a node, nobody else can, so ideally you'll use all the processors

3. The main exception to that is when you need lots of memory - you might run as few as one task on a node, but use all of the memory. Alternately, this is what FAT or HiMem nodes are for

7 Step Guide - Hybrid

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Grab example script
    2. Modify nodes and processors-per-node required
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to load any required modules
4. Set the number of OMP threads or other threading parameters
5. Add the line to actually run the job, with any inputs needed (pipes etc)
6. Submit the job
7. Wait…

Demo

1. It’s quite common to use one MPI task per node, and one OMP thread per core. This is usually most efficient, but isn’t necessary

2. There's more than one way to specify the number of cores, nodes etc - you may want to refer to Slurm docs and cluster docs for details

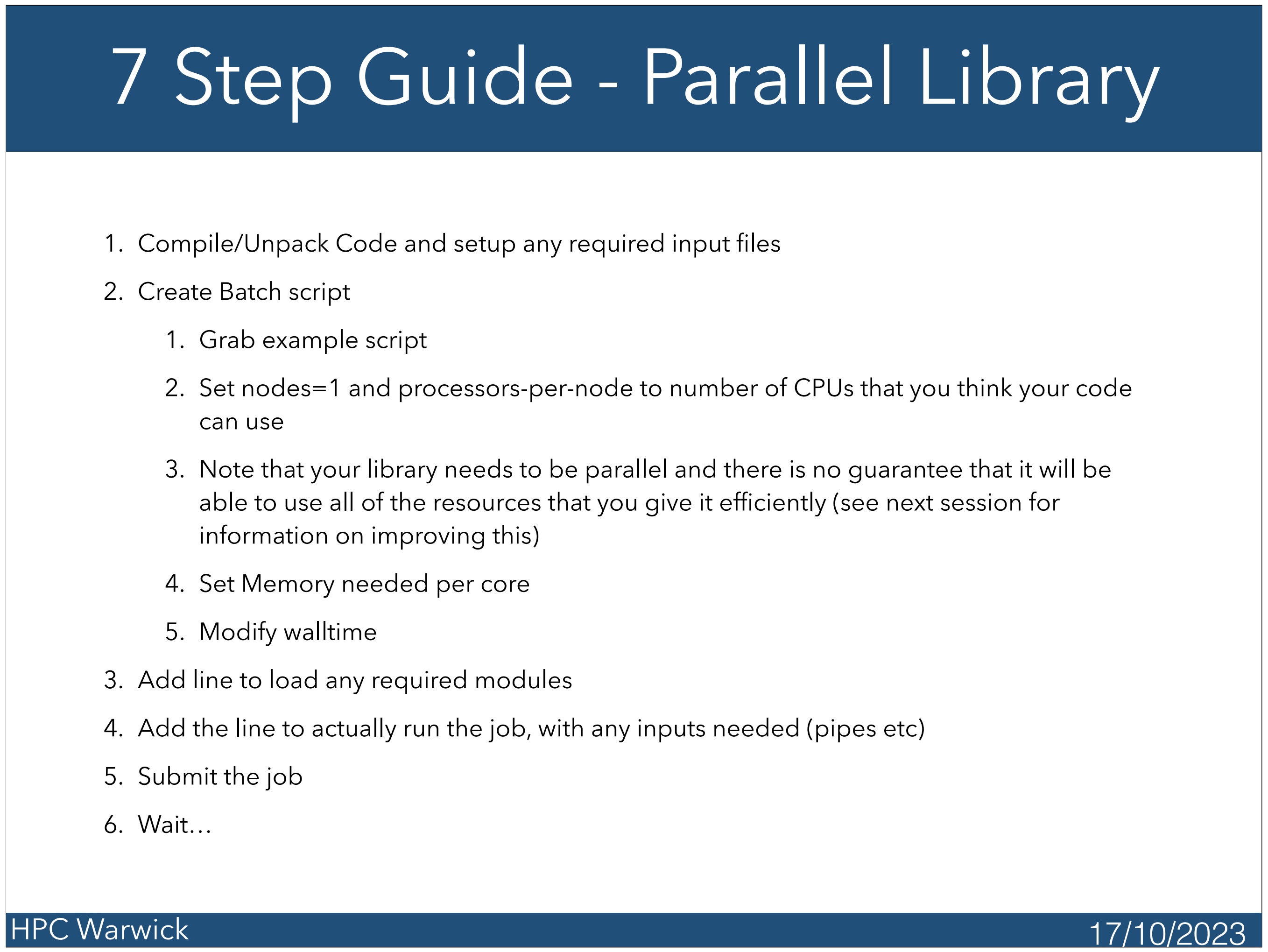
6 Step Guide - GPUs

1. Compile/Unpack Code and setup any required input files
    1. Load CUDA module
    2. Compile with nvcc etc if necessary
2. Create Batch script
    1. Grab example script
    2. Set tasks per node and cpus per task
    3. Select one gpu
    4. Modify walltime
3. Load any required modules
4. Add the line to actually run the job, with any inputs needed (pipes etc)
5. Submit the job
6. Wait…

Demo

1. NEVER try and use the GPU nodes to jump the CPU queue!

2. Avon has 16 GPU nodes, each with 3 x NVIDIA RTX 6000.



7 Step Guide - Parallel Library

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Grab example script
    2. Set nodes=1 and processors-per-node to number of CPUs that you think your code can use
    3. Note that your library needs to be parallel and there is no guarantee that it will be able to use all of the resources that you give it efficiently (see next session for information on improving this)
    4. Set Memory needed per core
    5. Modify walltime
3. Add line to load any required modules
4. Add the line to actually run the job, with any inputs needed (pipes etc)
5. Submit the job
6. Wait…

HPC Warwick 17/10/2023

Quite commonly nowadays library code is written to use multiple processor cores. In this case you don't have to write parallel code yourself, you just use the library and parallelism "happens" for you. If your code makes heavy use of a particular library then read the documentation for the library to see if it has parallel (also called threaded or multi threaded) capabilities and see if you need to do anything to turn them on.

Note that parallel libraries are often not able to use all the computational resources that you can throw at them - check to see how many processors you can use efficiently and request processors to match

Demo

1. Note that almost all libraries can only run on a single node
2. There are libraries that run across nodes but they will generally need more work from you - read the documentation
3. If you're using a node, nobody else can, so ideally you'll use all the processors
4. The main exception to that is when you need lots of memory - you might run as few as one task on a node, but use all of the memory. Alternately, this is what FAT or HiMem nodes are for

6 Step Guide - Job Array

1. Compile your code as for running a single job
2. Create Batch script
    1. Grab example script
    2. Set tasks per node and cpus per task
    3. Modify walltime
    4. Modify the "array" line to select how many jobs to run
3. Load any required modules
4. Add the line to actually run the job. You can access the job array ID number using the environment variable SLURM_ARRAY_TASK_ID
5. Submit the job
6. Wait for them all to finish …

Demo

1. Job arrays are the preferred way of running lots of individual jobs

2. They are easier for the scheduler and easier for you

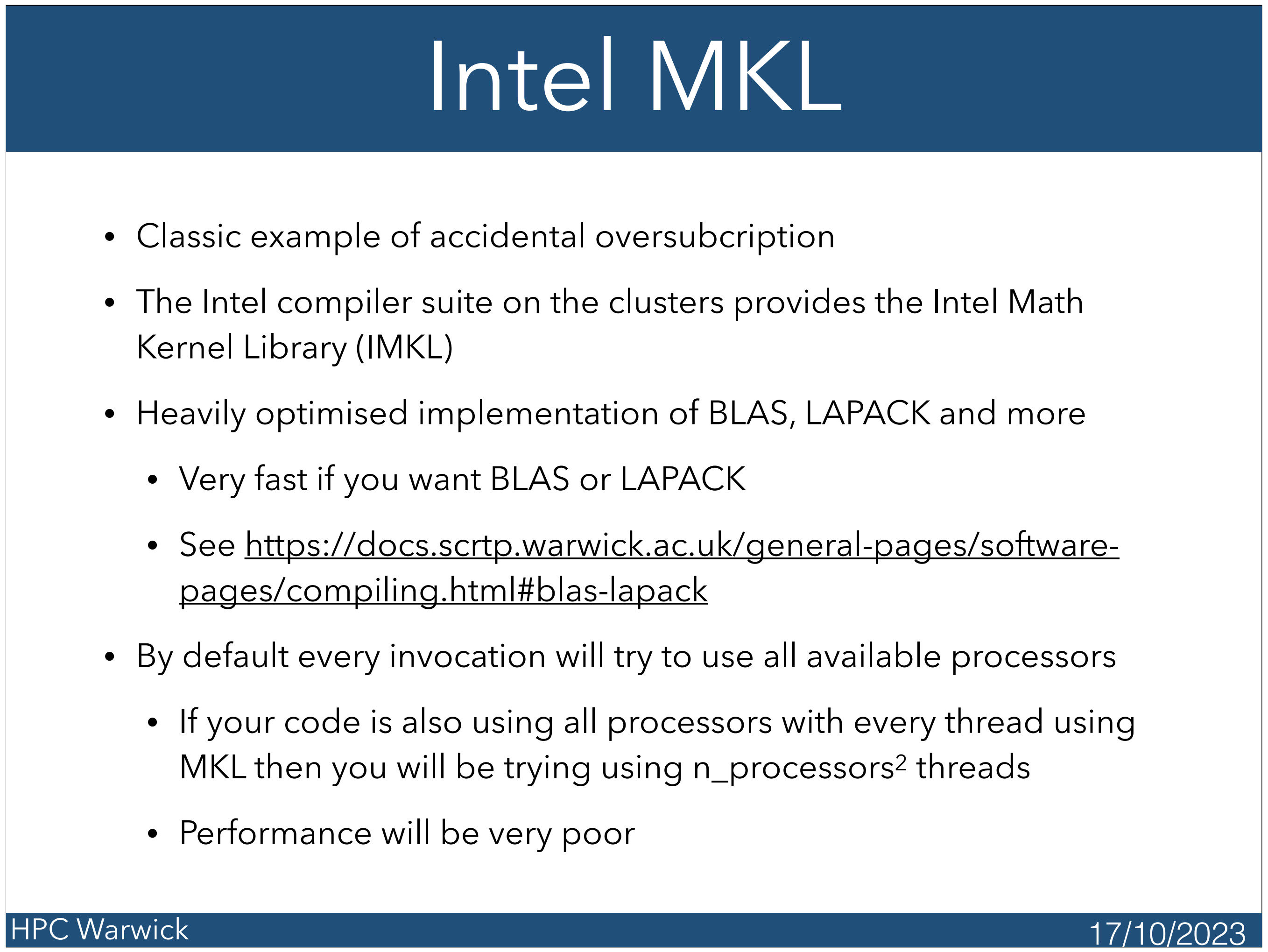
3. You can add dependencies (don’t run job B until job A has finished) but not conditionality (run job B if job A gives a specific value)

4. There are tools for dealing with conditional execution but they aren’t part of SLURM

Oversubscription

Oversubscription

- Computer cores generally work best when they have one and only one task to do
- Only use as many threads as there are CPU cores
- If you use more threads than cores then this is called oversubscription and this generally slows things down, often badly
  - Two threads per core often works OK if the CPU has simultaneous multithreading (hyperthreading) as most of our CPUs do but more is always bad
- Accidental oversubscription can happen if you use a parallel library from a parallel code!
  - You have to be explicitly writing your code to be parallel otherwise no problem



Intel MKL

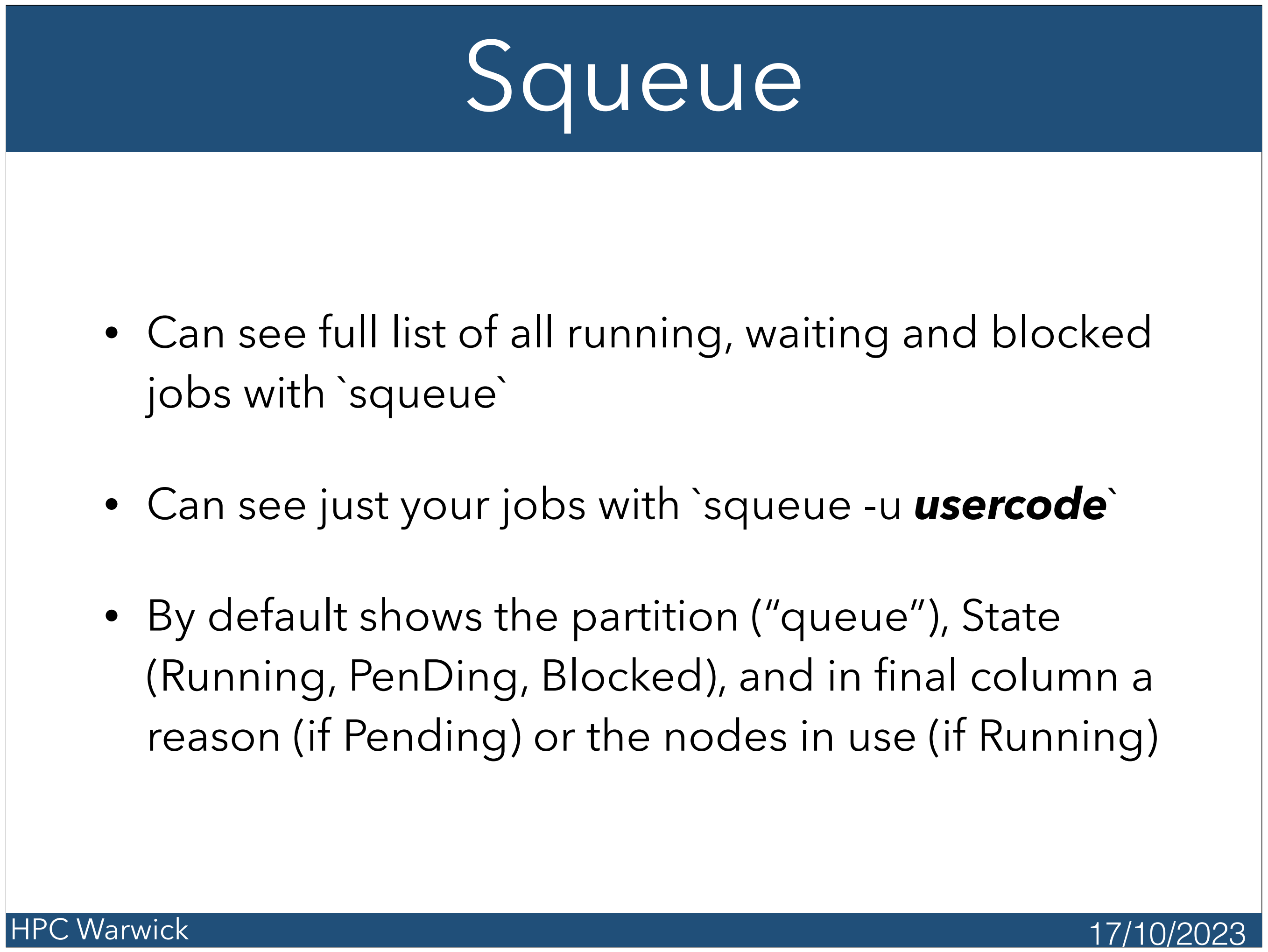
- Classic example of accidental oversubcription
- The Intel compiler suite on the clusters provides the Intel Math Kernel Library (IMKL)
- Heavily optimised implementation of BLAS, LAPACK and more
  - Very fast if you want BLAS or LAPACK
  - See https://docs.scrtp.warwick.ac.uk/general-pages/software-pages/compiling.html#blas-lapack
- By default every invocation will try to use all available processors
  - If your code is also using all processors with every thread using MKL then you will be trying using n_processors^2 threads
  - Performance will be very poor

For more about the linking step of building code in general, see https://en.wikipedia.org/wiki/Linker_%28computing%29#Overview or https://www.airs.com/blog/archives/38

A brief explanation of the thing to watch for: a code which uses IMKL and is run in parallel starts some number of threads on each node, usually one per processor. These can be separate MPI processors, OpenMP threads or some combination. Each core has one thread to work on. IMKL by default will start one thread per available processor, each time it's loaded by one of your threads. This can quickly run out of control and grind to a halt.

Checking on your Jobs

This section shows the Slurm commands first. For completeness we show the Moab options in these notes, in particular checkjob, which has not direct Slurm equivalent. Watch the mailing lists for the continuing changeover.



Squeue

- Can see full list of all running, waiting and blocked jobs with `squeue`
- Can see just your jobs with `squeue -u ***usercode***`
- By default shows the partition ("queue"), State (Running, PenDing, Blocked), and in final column a reason (if Pending) or the nodes in use (if Running)

Reasons include: Resources (cluster cannot run your job right now), Priority (you can't run more jobs right now - e.g. you're using all the CPUs you're allowed to at once), Dependency (you said this job must follow another).

Stopping Jobs

Scancel

1. Cancel a job (remove from queue, or stop if running) with `scancel ***jobid***`
2. You can't cancel somebody else's job as you don't have permission, so don't worry
    1. If you do have permission, congratulations on your new sysadmin job
        1. Get back to work!

Interactive Jobs

Interactive Running

- For heavy data analysis, or small test runs, can use interactive mode
- You request a number of processors for a fixed amount of time and wait until they're available
- Once they're ready, you can run things
- Time starts when resource is ready, not when you start using it

As always, see the docs for details https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-interactive.html#interactive-jobs

Demo

1. Two ways to do this, either for one processor for heavy serial work, or multiple for parallel work

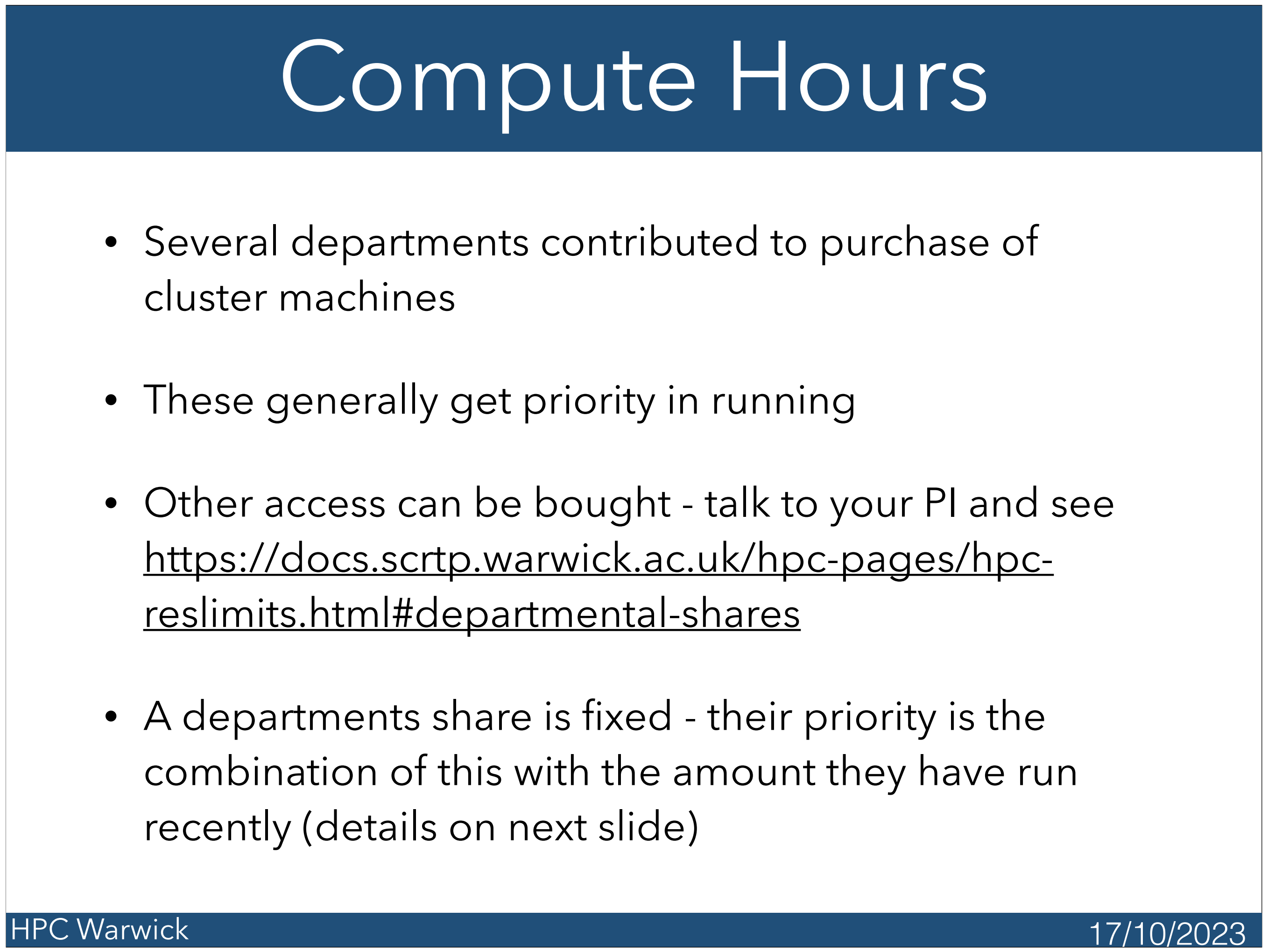
2. It's a good idea to check the queue before submitting a request, so you know whether resources are available

3. Use Slurm directly with salloc. The resources are put aside for us and we invoke srun on login node - this then uses what we've been given

4. Finish up from salloc with `exit`

Remember that serial running ties up an entire node even if you only use one processor. (This is not the case everywhere, so check docs for the system you're using if it's not ours)

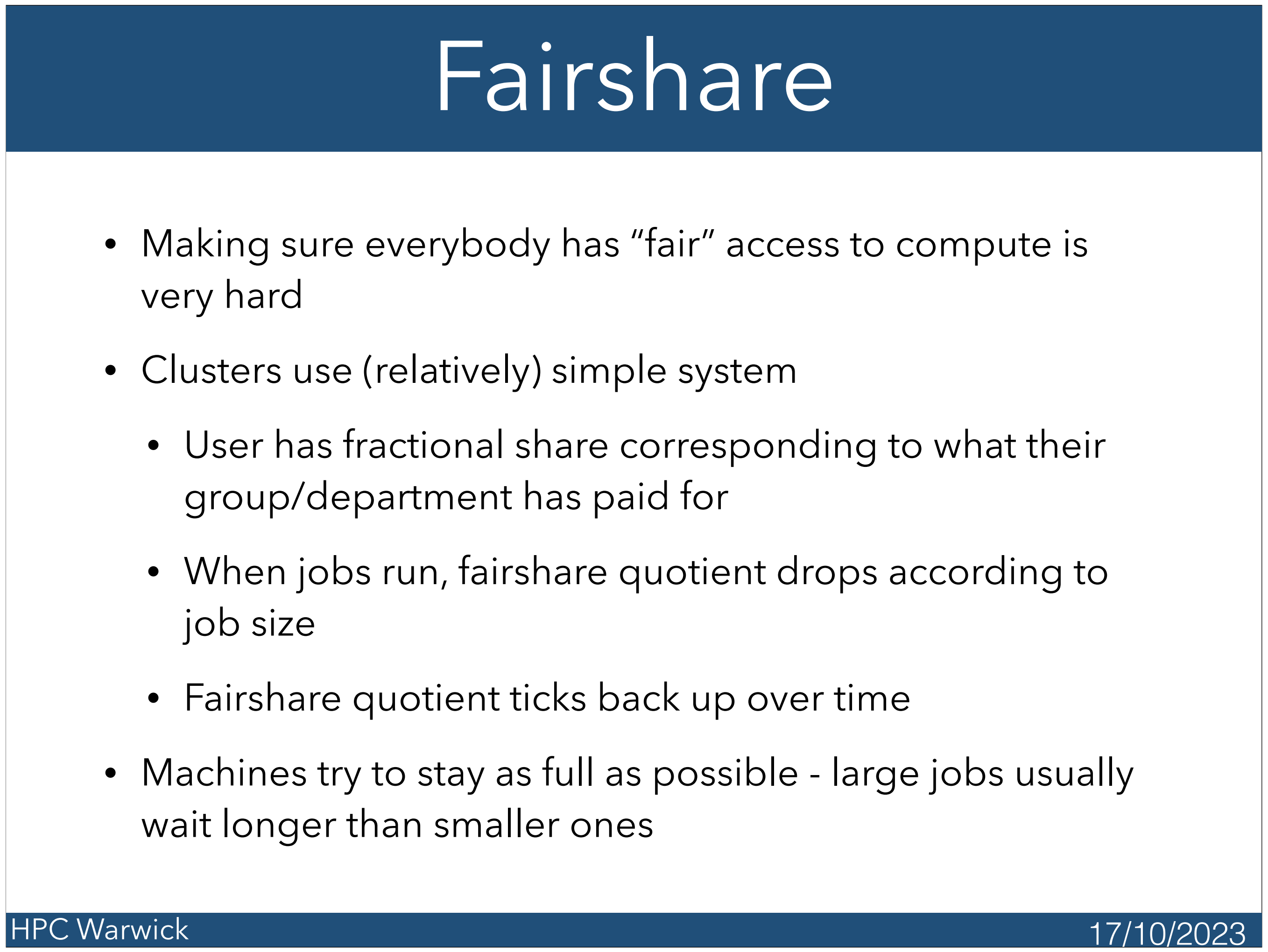
Fairshare and Quotas



Compute Hours

- Several departments contributed to purchase of cluster machines
- These generally get priority in running
- Other access can be bought - talk to your PI and see https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-reslimits.html#departmental-shares
- A departments share is fixed - their priority is the combination of this with the amount they have run recently (details on next slide)

Note that low priority jobs might run fine when there's not much on the machines. When they're heavily loaded however, they might not run at all. As far as I (HR) know, the scheduling system doesn't care how long something has been queued, although two jobs of equal priority often schedule first-come-first-serve



Fairshare

- Making sure everybody has “fair” access to compute is very hard
- Clusters use (relatively) simple system
  - User has fractional share corresponding to what their group/department has paid for
  - When jobs run, fairshare quotient drops according to job size
  - Fairshare quotient ticks back up over time
- Machines try to stay as full as possible - large jobs usually wait longer than smaller ones

The details of the share calculation are not simple - lots of effort has gone and still goes into working out the ‘best’ systems. It’s also not easy to work out what the results of a set of scheduling rules should be.
E.g. when the machines aren’t full anything not blocked will schedule quickly; as they fill up groups with larger shares will have higher priority and occupy more of the machines

Disk Quotas - Desktop

- Storage available on Home (fully backed up) and Storage
- See https://docs.scrtp.warwick.ac.uk/linux-pages/storage.html
- Check usage using instructions at https://docs.scrtp.warwick.ac.uk/linux-pages/storage.html#storage-quotas
- If you run out of Home space you'll be unable to log in and will have to raise a bug!

Disk Quotas - Clusters

- There is a home for the clusters, but this is different to the Desktop home
- See https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-storage.html for details
- Check your quota with `mmlsquota --block-size auto` or see https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-storage.html#quota
- Note there is no backup! This is working space for code runs!
- See https://docs.scrtp.warwick.ac.uk/hpc-pages/hpc-storage.html#scrtp-linux-storage for how to access linux storage from the cluster login nodes

Walltime

- There is a 48 hour limit for runtime on Warwick machines
- Bugzilla requests for a longer walltime may be granted if there is a very good reason
- If using a proprietary code, read its docs for 'checkpointing' or 'restart' capability
- If writing your own code, try and build in a mechanism for restarting code from a file
  - RSE forum might be able to help

Almost all clusters have such a limit, commonly anywhere from 24 to 72 hours. Scheduling is almost impossible without it. Some have special long queues, or a way to apply for a longer runtime - see their documentation for details.

Other Essentials

Taskfarm Jobs

- Taskfarm -
  - Groups buy dedicated machines
  - Some shared nodes
  - These are kept in SCRTP machine rooms, and available via a special queue
  - If your supervisor has one you should use, they should tell you the queue name
- Uses Slurm just like the clusters
- MUST run from working (storage) directory, NOT home

“Serial” jobs

- Clusters aren’t designed to run single-core jobs
  - Always get a whole node
- If you have multiple single-core jobs can run them all at once
- You can use GNU parallel to manage this or submit a “Job Array”
  - More sophisticated libraries like Dask
- Details at https://docs.scrtp.warwick.ac.uk/hpc-pages/avon-slurm-examples/avon-serial.html#serial-jobs
- Or see https://warwick.ac.uk/rse/training/parallelismprimer

Note again some clusters do allow you to use fractions of a node - we don’t for good reasons that we’re happy to explain if you ask but won’t go into here

Wrap Up

Summary

- NEVER run code on login nodes
  - Use interactive jobs even for heavy compilation loads
- Cluster jobs use queue system. Scripts are slightly different for MPI, OMP, hybrid, single node or array jobs but always specify:
  - number of processors
  - memory
  - walltime
- Fairshare really is the fairest way to run things, honestly!

Useful Links

- For recap of basic Linux commands: https://docs.scrtp.warwick.ac.uk/general-pages/terminal.html

- For everything you need to know about the clusters: https://docs.scrtp.warwick.ac.uk/hpc.html

- If you have problems you can't work out: https://bugzilla.csc.warwick.ac.uk/bugzilla/

Command Line Download

- If you want to get the examples that we used here on Avon then running the following commands on Avon is an easy way of getting them
- **wget https://warwick.ac.uk/research/rtp/sc/rse/training/hpcbeyond/intro_hpc.zip**
- **unzip intro_hpc.zip**