Slurm Job Submission

ARCHER2 Virtual Tutorial, Wed 26th July 2023

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

ARCHER2 CSE Support Team

Reusing this material

Partners

epcc

UKRI

Engineering and Physical Sciences Research Council

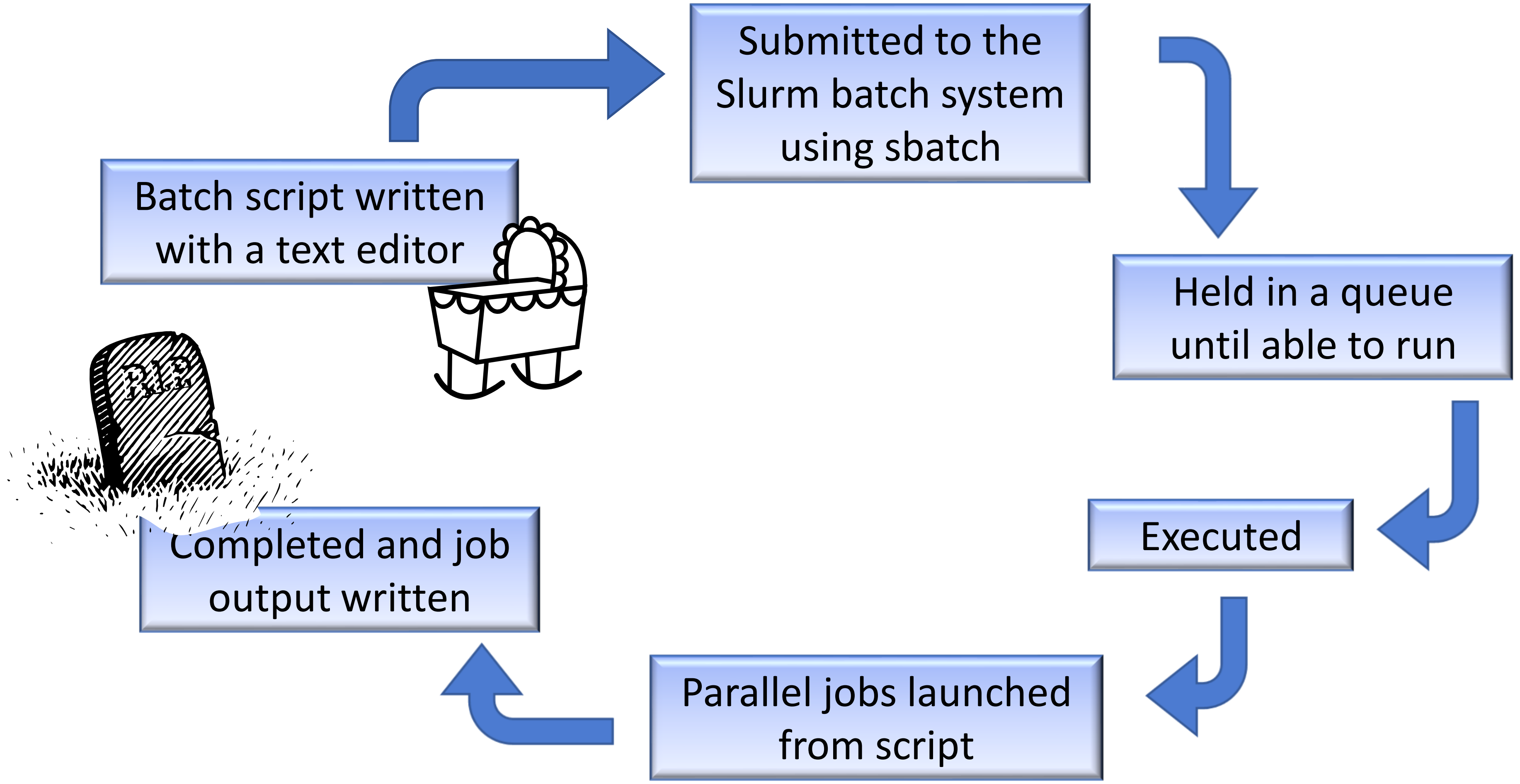
Natural Environment Research Council

epcc

THE UNIVERSITY of EDINBURGH

Hewlett Packard Enterprise

A day in the life of a Slurm job



Batch script written with a text editor
Submitted to the Slurm batch system using sbatch
Held in a queue until able to run
Executed
Parallel jobs launched from script
Completed and job output written
RIP

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 # Execute script via the bash shell
- and
  - #SBATCH is special to batch system
  - #SBATCH --job-name=Example_MPI_Job   # Pass on as arguments to sbatch

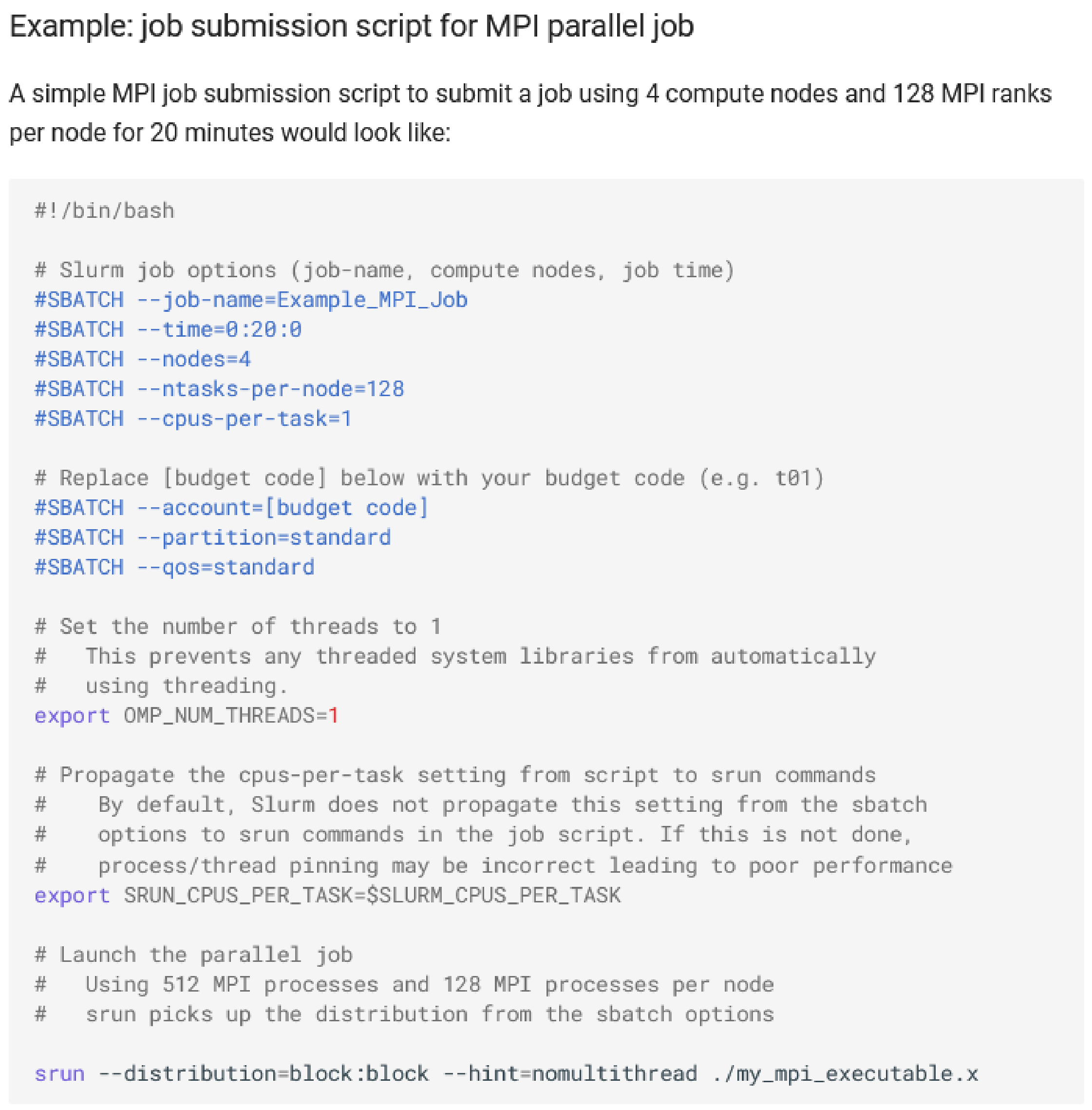
Why?

- Some parameters of your job are significant at submit time
  - to enable Slurm to schedule your job appropriately
  - e.g. number of nodes, wallclock time, ...

- Other aspects of your job are significant at execution time
  - setting environment variables
  - job preparation: copying files, pre-processing scripts, ..

- Handy to have all of these in the same file
  - rather than pass huge number of arguments to sbatch (--nodes=, -- time=,...)
  - hide these in the script as comments
  - you have a complete copy of all the parameters for your job

https://docs.archer2.ac.uk/user-guide/scheduler



Example: job submission script for MPI parallel job

A simple MPI job submission script to submit a job using 4 compute nodes and 128 MPI ranks per node for 20 minutes would look like:

```
#!/bin/bash

# Slurm job options (job-name, compute nodes, job time)
#SBATCH --job-name=Example_MPI_Job
#SBATCH --time=0:20:0
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
#SBATCH --partition=standard
#SBATCH --qos=standard

# Set the number of threads to 1
#   This prevents any threaded system libraries from automatically
#   using threading.
export OMP_NUM_THREADS=1

# Propagate the cpus-per-task setting from script to srun commands
#    By default, Slurm does not propagate this setting from the sbatch
#    options to srun commands in the job script. If this is not done,
#    process/thread pinning may be incorrect leading to poor performance
export SRUN_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK

# Launch the parallel job
#   Using 512 MPI processes and 128 MPI processes per node
#   srun picks up the distribution from the sbatch options

srun --distribution=block:block --hint=nomultithread ./my_mpi_executable.x
```

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

I'm afraid it's not that simple

Why?

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

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

Yes, and some commands that *don't* work on the login nodes *only* work under Slurm

☹

Birth: submitted to batch system

```
user@archer2$ sbatch myscript.job
Submitted batch job 4015920.sdb
```

- Slurm takes a *copy* of your batch script and stores it somewhere
  - ascertains resource requirements (e.g. no. of nodes)
    - from command line arguments or from **`#SBATCH`** lines
  - returns a unique job number

- Job is queued until resources are available
  - query status with **`squeue –u <myusername>`**
    - easier to use **`squeue --me`**
  - job status set to pending: “PD”

Reproducibility

- Useful to have a copy of your Slurm script in the job output
  - since we often edit the same script over and over ...
  - useful trick

```
echo "+--------------------------------------------------------+"
echo "| Start of contents of SLURM job script for job $SLURM_JOB_ID |"
echo "+--------------------------------------------------------+"

cat $0

echo "+--------------------------------------------------------+"
echo "|  End of contents of SLURM job script for job $SLURM_JOB_ID  |"
echo "+--------------------------------------------------------+"
```

- Submission parameters can be accessed as environment variables
  - $0 is the path to Slurm’s unique copy of your batch script

Resource selection

- Unlike some systems, most jobs on ARCHER2 go to a single queue
    - **`#SBATCH --partition=standard`**
    - **`#SBATCH --qos=standard`**
  - some special queues, e.g. for short and long jobs, higher memory, ...
  - a check is done at submission time to ensure you have a reasonable budget

- Jobs **scheduled** entirely based on **requested** resources
  - i.e. run time and number of (128-core) nodes
  - parallel compute nodes are always allocated in exclusive mode

- Can specify high memory
  - **`#SBATCH --qos=highmem`**
  - also have short, long, serial, reservation, ...

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

I'm afraid it's not always that simple

Why?

If your're running a self-compiled executable from the script then it will see the version that's there at run time - OK for a package but not if you're frequently recompiling

How does it decide when to run my job?

A balance of your requested number of nodes and the runtime compared to all the other jobs in the system

So I should take care when specifying these?

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

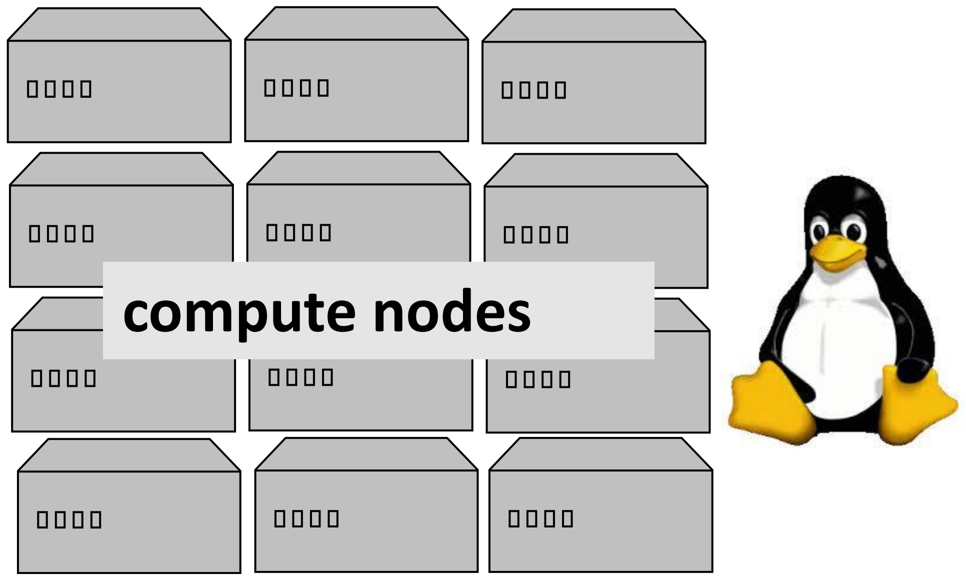
Childhood: job script runs

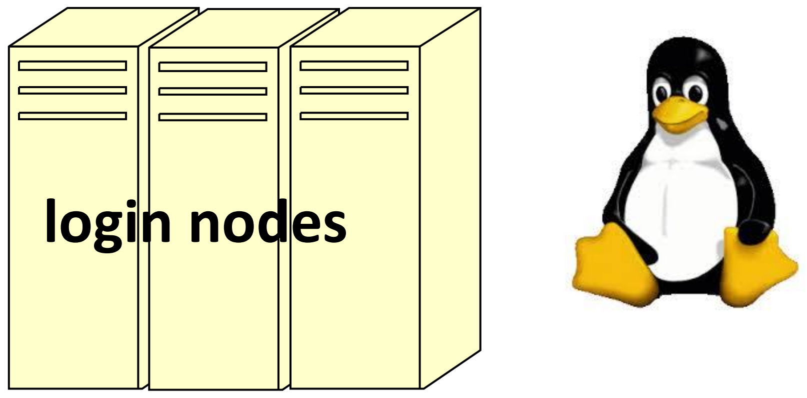
- A set of compute nodes is reserved for your job
  - **squeue** job status set to "R"
  - your bash script is executed on one of the allocated compute nodes
    - the *Slurm management node*
    - ... the lowest-numbered one?
  - The **only way** to access your other compute nodes is via **`srun`**

```
#SBATCH --job-name=Example_MPI_Job
...
# Now run the parallel job
srun mympiprogram
```

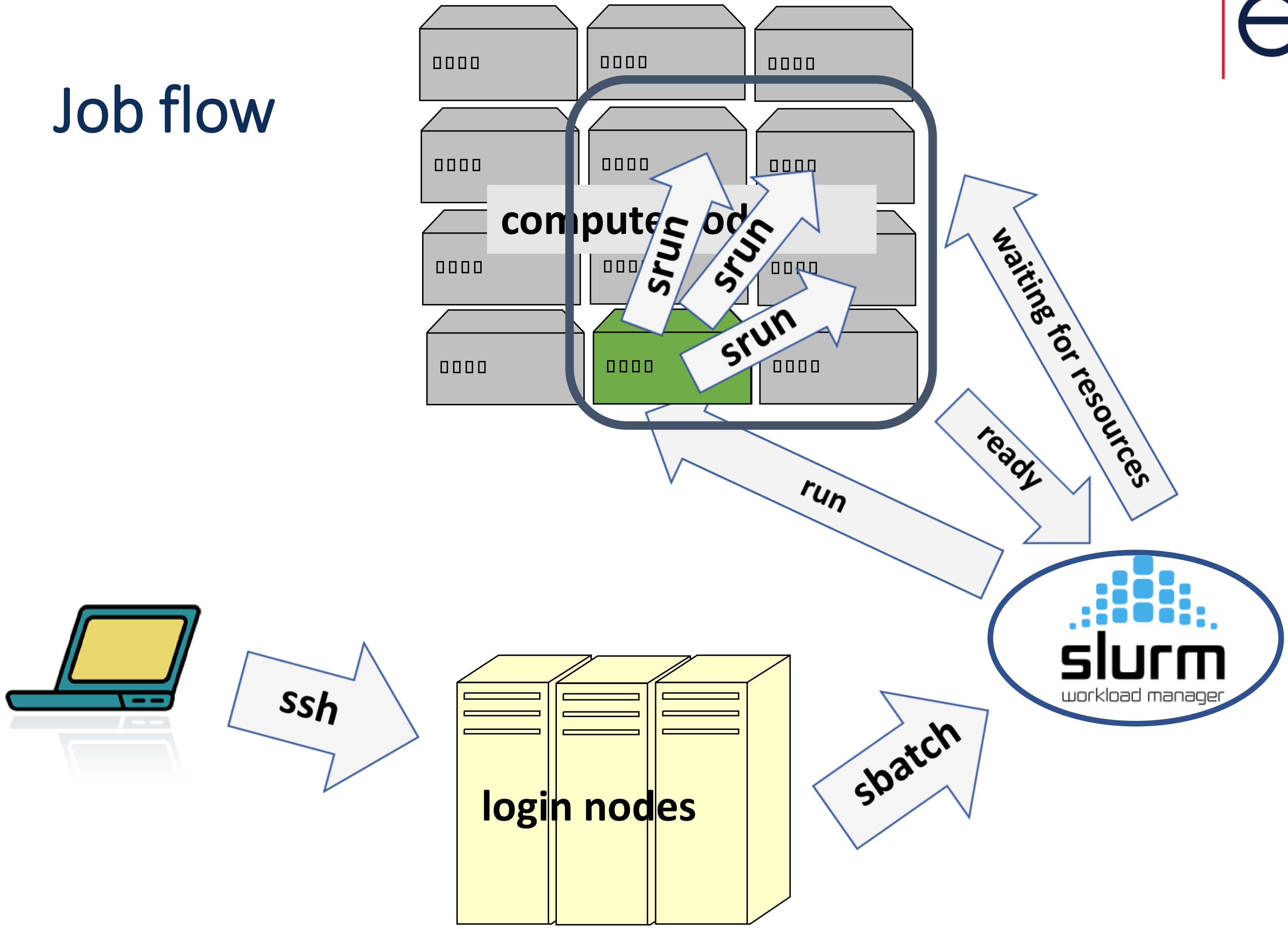
- number of MPI processes etc. computed from submission parameters
  - can be over-ridden (with care!)

Operating Systems





Job flow


compute nodes
srun
srun
srun
waiting for resources
ready
run
ssh
login nodes
sbatch
slurm
workload manager

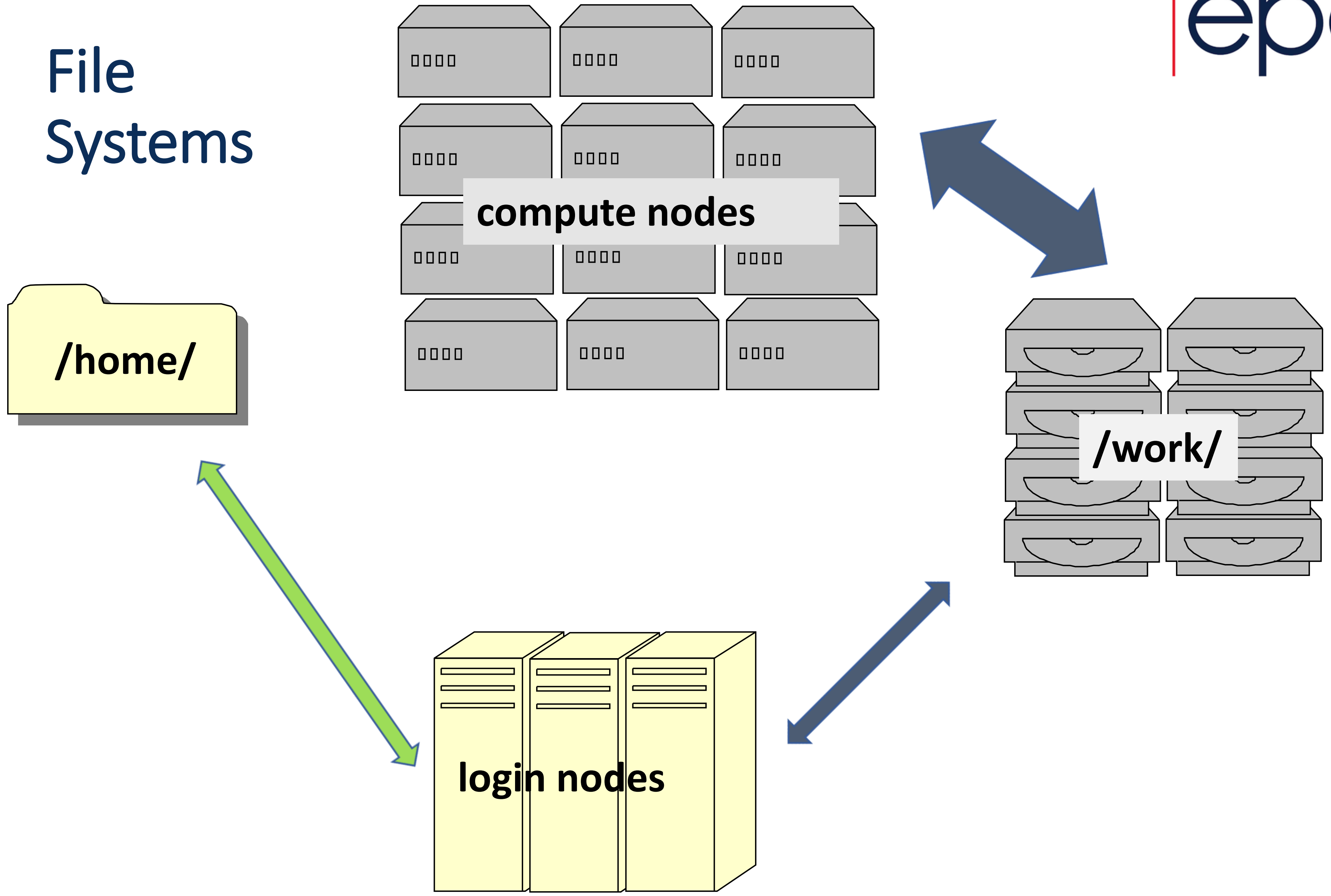
Adulthood: parallel jobs

- Compute nodes reserved for duration of job
  - Slurm doesn’t care if/how you use them!
  - all commands from batch script executed on management node
  - **`srun`** on management node causes parallel jobs to run all compute nodes

- Do production runs in **`/work/`** filesystem, not **`/home/`**.
  - your script automatically starts executing from where it was submitted

- **`srun`** does the following
  - **launches the executable** from all the compute nodes (reads from filesystem)
    - can use **sbcast** to explicitly copy to /tmp on each node
  - **gathers** the standard outputs / errors from all the PEs and sends to log file

File Systems



Reach 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 srun running in background (see later)

  - outputs collated and flushed to file
    - e.g. written to myjob-1234567.out
  - sbatch job status set to "CG" for a little while
    - job is "Completing" (not "Completed") but script has finished
  - then disappears

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
  - ideally request slightly more time that the actual runtime

- A job that is killed due to running for too long is still charged
  - unless it hung due to system error
  - users can request a refund

How does srun place processes and threads? (i)

- key parameters are:
  - --nodes
  - --tasks-per-node
  - --ntasks
  - --cpus-per-task

- some redundancy here
  - e.g. nodes=4 & tasks-per-node=128 is same as ntasks=512 & tasks-per-node=128
  - Slurm complains if they're not the same (and has some rules for precedence)

- at submit time, Slurm just needs to work out how many nodes you need
  - at runtime it's a bit more complicated

How does srun place processes and threads? (ii)

- we recommend:

```
srun --hint=nomultithread --distribution=block:block mympiprogram
```

- #SBATCH job parameters are all passed automatically to srun
  - **except** you must set: export SRUN_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK
    - only really has an effect if you have --cpus-per-task larger than 1
- this places MPI processes sequentially across each node
  - fills up first node entirely before moving on to second
  - each process is given *cpus-per-task* CPU cores
    - e.g. cpus-per-task = 2 gives 64 MPI processes on even-numbered cores of each node
  - --hint=nomultithread means ignore hyperthreading / hypercores / SMT
    - just use physical cores 0-127 and not hypercores 128-255

Why not run MPI processes on all the cores?

- Two main use cases:
  - you need more than 2GiB per core
  - you are using OpenMP threading in addition to MPI

- For hybrid MPI/OpenMP
  - set OMP_NUM_THREADS equal to *cpus-per-task*
    - e.g. export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
  - `export OMP_PLACES=cores` to ensure correct binding of threads

- Can all get a bit complicated
  - if in doubt run the “xthi” program (module load xthi)
  - prints out nodes and the binding of each process / thread within each node
    - either as a number (“8”), a list (“3,131”) or a range (“0-7”).

srun

- Can issue multiple srun's in a single job
  - single job + many srun's may be better than many small batch jobs
    - benchmarking
    - simple taskfarms with multiple concurrent sruns

- Approach
  - specify minimal resource requirements at submit time
    - e.g. #SBATCH --nodes=4
  - specify all other parameters as arguments to srun, e.g.

```
srun --nodes=2 –ntasks=256 --tasks-per-node=128 --cpus-per-task=1 \
    --distribution=block:block --hint=nomultithread mympiprogram
```

benchmarking

- Single batch job requesting maximum number of nodes

- Many sequential sruns executing on subset of nodes
  - e.g. scaling run on 128, 256, 512, 1024, ... processes

- Advantages
  - much less queueing
  - all jobs run on the same nodes so performance more reproducible

- Disadvantages
  - you pay for unused resources

Taskfarm

- OK, how about:

```
srun ... mympiprogram dataset1
srun ... mympiprogram dataset2
srun ... mympiprogram dataset3
srun ... mympiprogram dataset4

# Incorrect! – all these run sequentially
```

Need to run them in the backround

```
 srun ... mympiprogram dataset1  &
 srun ... mympiprogram dataset2  &
 srun ... mympiprogram dataset3  &
 srun ... mympiprogram dataset4  &

# Incorrect: “Job finishes after the all the
# commands in script have been executed”.
# Final srun returns immediately, script
# reaches end and finishes, srun’s all killed.
```

Multiple aprun’s in the background (ii)

```
srun ... mympiprogram dataset1 &
srun ... mympiprogram dataset2 &
srun ... mympiprogram dataset3 &
srun ... mympiprogram dataset4

# Incorrect: script finishes when dataset4
# finishes, but other dataset may still be
# running at that time so will be killed!
```

Run them in the backround and wait…

```
 srun ... mympiprogram dataset1 &
 srun ... mympiprogram dataset2 &
 srun ... mympiprogram dataset3 &
 srun ... mympiprogram dataset4 &

 wait

# Correct! “wait” blocks until all spawned
# processes are complete
# Here, srun acts like a mini-scheduler
```

This only works for full nodes

- If you wanted **8** jobs each using **64** processes each:

```
srun ... mympiprogram dataset1 &  # run with 64 tasks
...
srun ... mympiprogram dataset8 &  # run with 64 tasks
```

- srun assigns entire node resources (including memory) to each program
  - 5th srun will block until one of first 4th finishes even though CPU-cores are available
- Solution: specify memory requirements (2 GiB/process with headroom)

```
srun --nodes=1 --ntasks=64 --tasks-per-node=64 --cpus-per-task=1 \
     --distribution=block:block --hint=nomultithread \
     --exact --mem=1500M mympiprogram dataset1 &
```

Tips and tricks (i)

- Useful to have print statements appear in log files ASAP

```
srun --unbuffered ...
```

- Interactive jobs
  - allow you to do realtime experiments with many sruns but a single sbatch
  - or debug your batch script to check that all commands are correct
- Essentially submit a job that just runs a terminal

```
user@ln01:work$ srun --nodes=2 --exclusive --time=00:20:00 \
                --partition=standard --qos=short --account=[budget] \
                --pty /bin/bash
user@nid001261:work$
```

  - to fully debug scripts may require a clean environment: `--export=none`
- Subsequent srun commands require: `--oversubscribe`

Tips and tricks (ii)

- May want to run a bash system command on every compute node
  - e.g. monitor memory usage, CPU load, ..

- Approach
  - put it in an executable shell script

```
#!/bin/bash
echo -n “running on node: $(hostname)”
top -b -n 1 # Monitor running processes
```

- Run one copy per node

```
srun --tasks-per-node=1 --ntasks=$SLURM_NNODES --nodes=$SLURM_NNODES ./top.sh
```

epcc

Questions?

☺