Pawsey

Submitting and Monitoring Your Job

Setonix Phase I Release

21 June 2022, Version 1.02

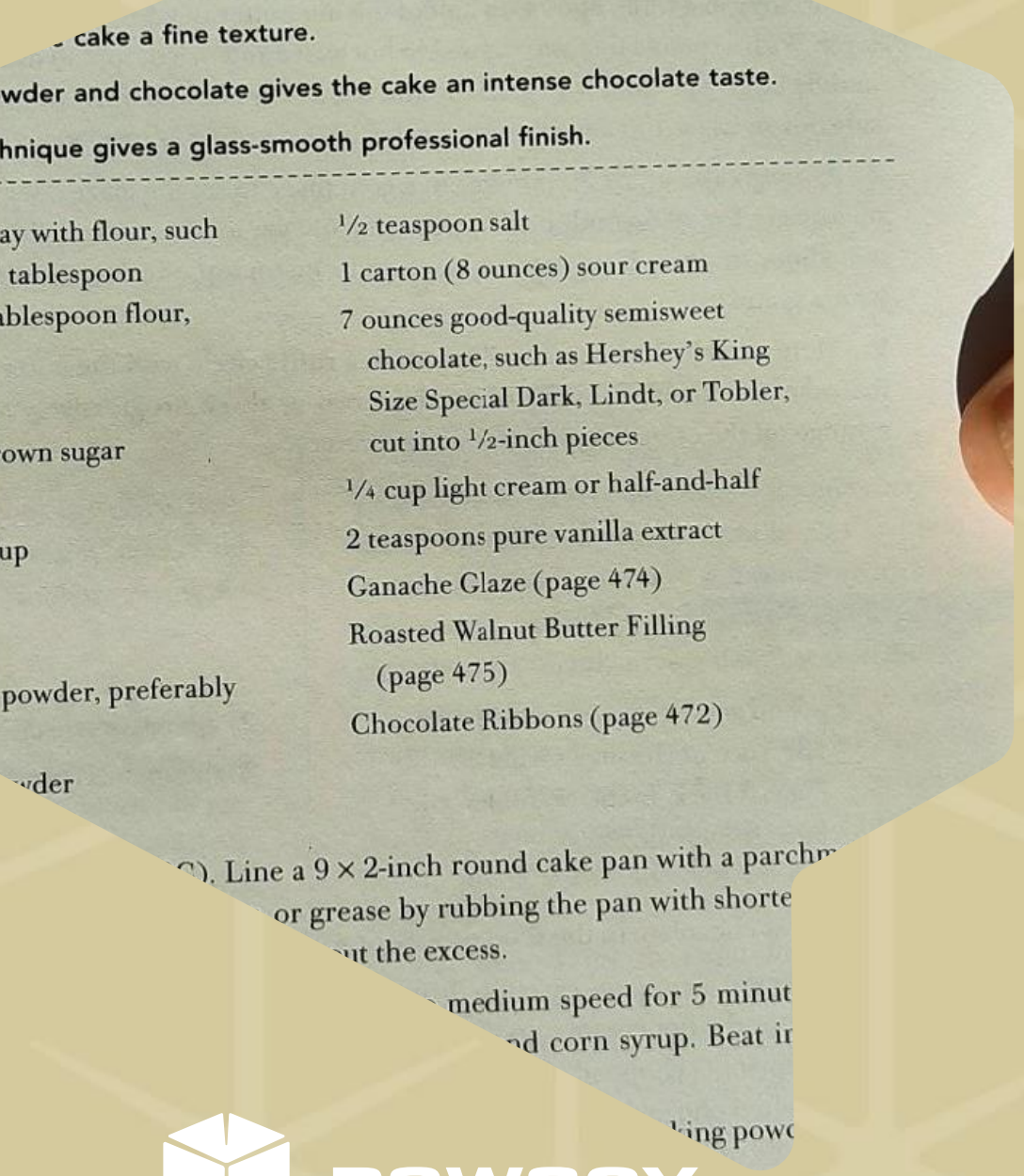
Focus for this Training

Learning Outcomes:

- Summarise high-level changes to the process of submitting and running a job on Setonix
- Understand how usage on the supercomputer is charged against project allocations
- Identify changes to the Slurm scheduler configuration, such as partitions and quality of service levels
- Identify key job script changes to submit and run jobs on Setonix
- Summarise changes to monitoring and managing a job on Setonix
- Discuss the tools, software and reports that can help you monitor and manage a job on Setonix
- Familiarise yourself with best practices for scheduling, running and monitoring a job on Setonix

Core Migration Training Modules:

1. Getting Started with Setonix
2. Supercomputing Filesystems
3. Using Modules and Containers
4. Installing and Maintaining Your Software
5. **Submitting and Monitoring Your Job**
6. Using Data Throughout the Project Lifecycle

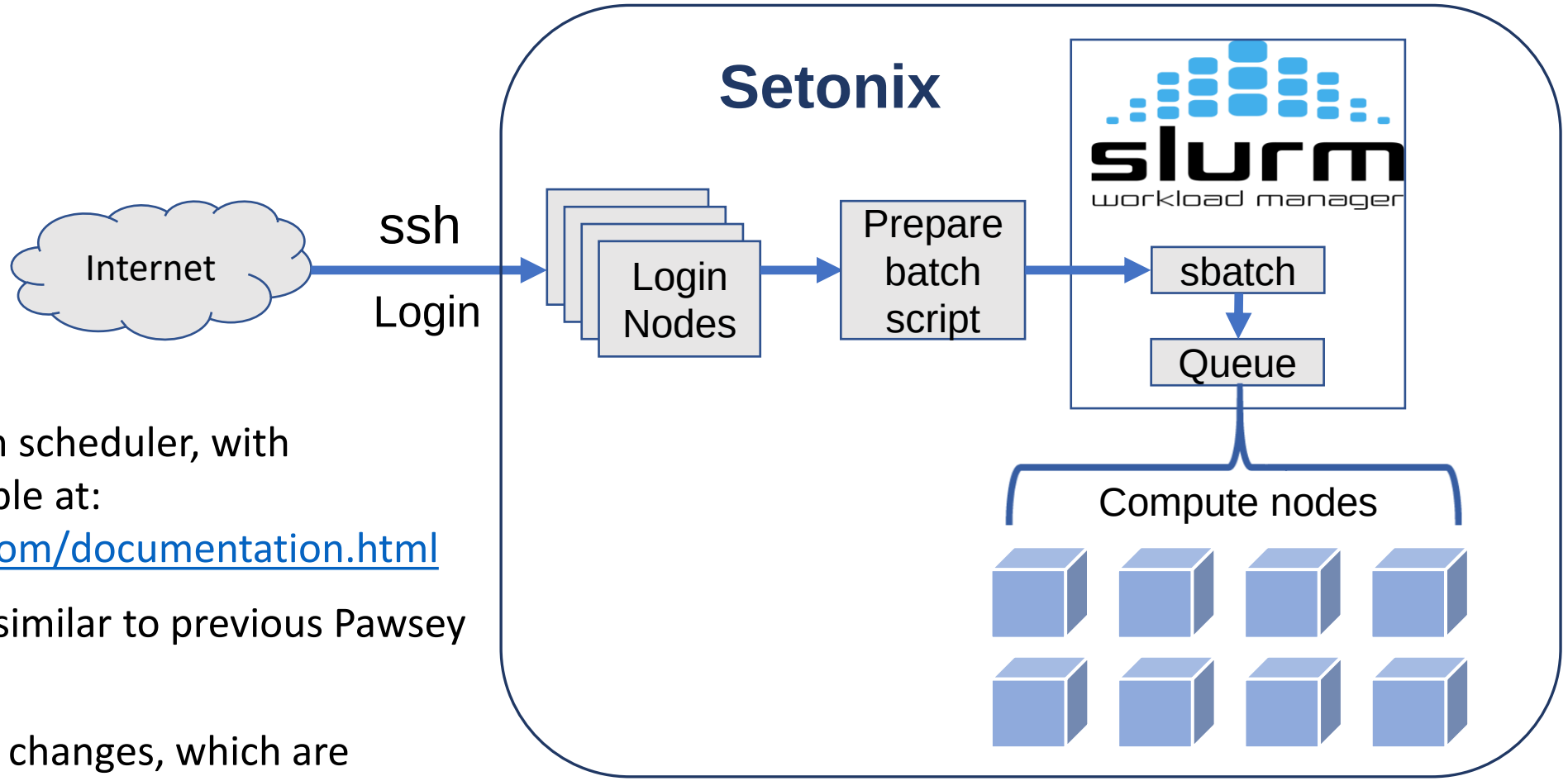


pawsey

Section 1

Overview of Key Scheduler Changes

Scheduling Jobs on Setonix



- Setonix uses the Slurm scheduler, with documentation available at: https://slurm.schedmd.com/documentation.html
- The overall process is similar to previous Pawsey supercomputers.
- There are some minor changes, which are described in the following slides.

⚠ **IMPORTANT:** Avoid running parallel programs on login nodes.

Job Execution on Setonix

- Slurm batch scripts are used, similar to previous Pawsey supercomputers.
- **Significant** changes include:
  - Nodes have shared access by default
  - Exclusive access is available via the `--exclusive` directive
  - The number of physical cores per node is now 128.
  - New filesystems have replaced the previous filesystems.
  - Partitions and quality-of-service levels have been simplified.
  - Job types previously run on Magnus and Zeus will run on Setonix.
  - There is a hard limit on project usage at 150%.

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration
#SBATCH --exclusive

module load module/version

srun ./program
```

pawsey

Section 2

Setonix Accounting Model

What is an accounting model?

The accounting model determines how researchers are charged for allocation usage.

- *Service Unit (SU)* is the unit of measure for consumable supercomputing resources.
- Typically, this is the *hourly usage of CPU cores* of a supercomputer.
- At Pawsey, 1 SU is equivalent to 1 hour use of 1 physical CPU core.
- Cost of a job (SU): number of CPU cores requested (CPU) × wall time (h).

Examples

1 SU = 1 hour use of 1 CPU core
= ½ hour use of 2 CPU cores

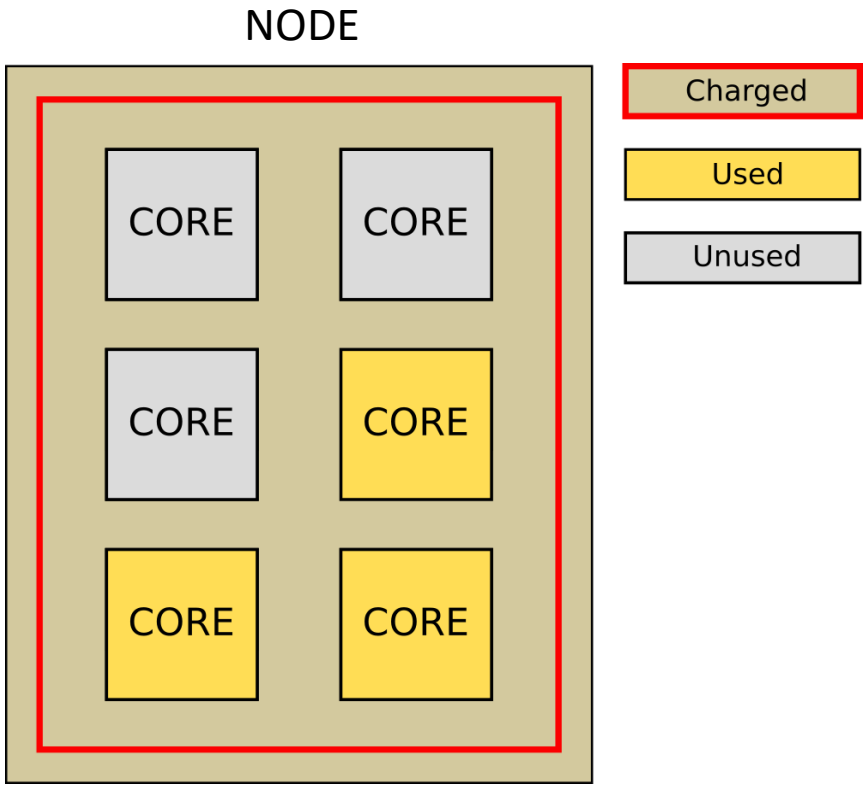
576 SU = 24 hours on 1 Magnus node (24 CPU cores)
= 4.5 hours on 1 Setonix node (128 CPU cores)

The Previous Accounting Model: Magnus

- Consumable resource: hourly usage of CPU cores
- 1 SU = 1 hour use of 1 CPU core

Exclusive node usage:

- Resources are allocated and charged at a compute node granularity.
- At any time, at most one job has access to cores in a node.
- If a job doesn't use all the cores in a node, the consumable resource is wasted.
- Hence, time on idle cores is also charged.
- Cost of a job (SU): 24 × nodes × wall time.

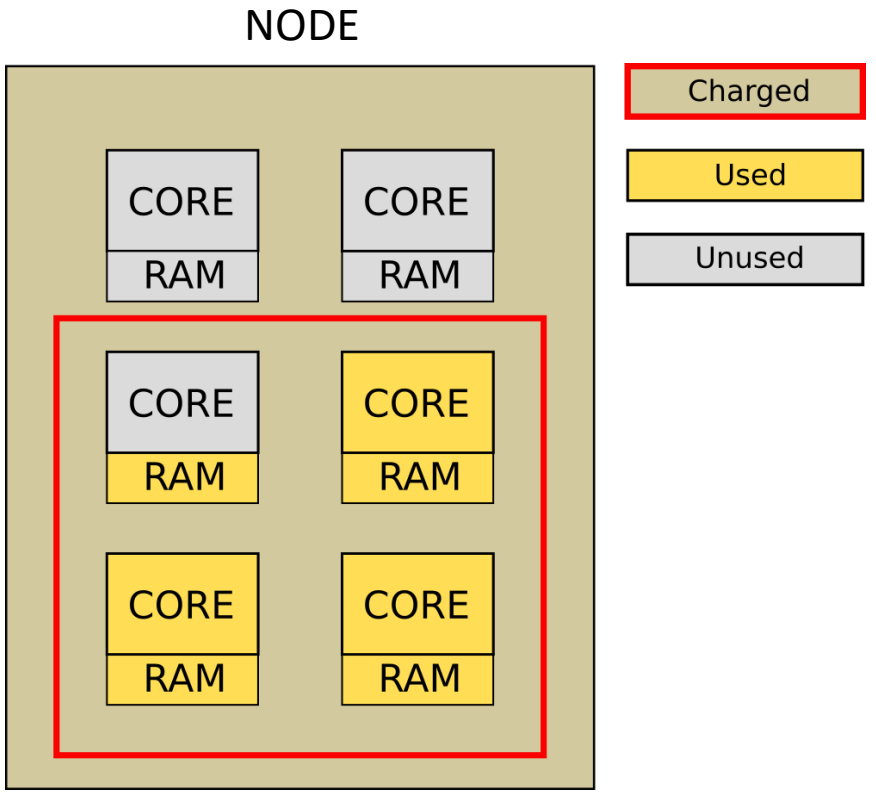


The New Accounting Model: Setonix

- Consumable resource: hourly usage of CPU cores (CPUs)
- 1 SU = 1 hour use of 1 CPU core

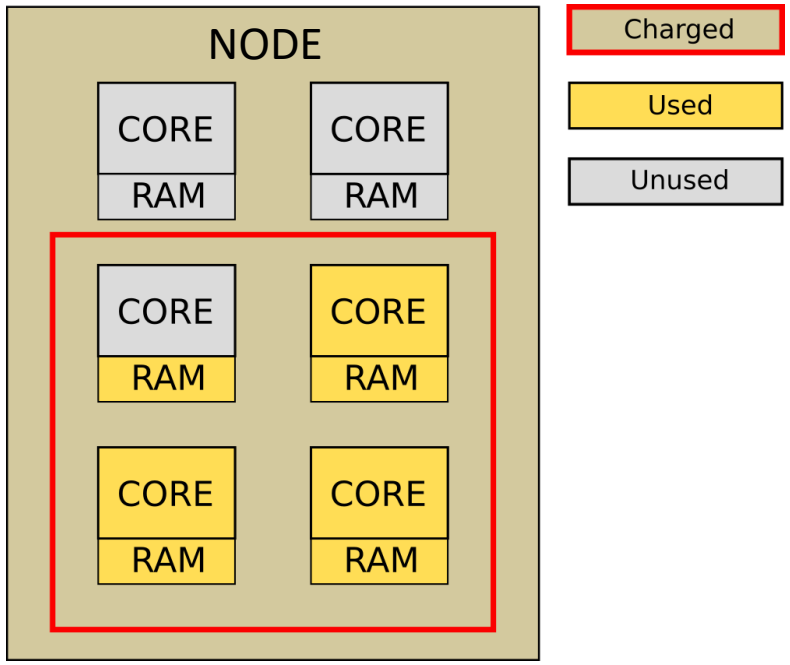
Proportional node usage:

- Resources are allocated at a *sub-node granularity*.
- Multiple jobs can run on the same node.
- A job is charged for the largest fraction of resources used.
- Random Access Memory (RAM) consumption is mapped to CPU consumption.
- RAM consumption of a job may affect other jobs on the same node.
- Cost of a job (SU): largest fraction × nodes × wall time.
  - Minimum: 1 SU per hour
  - Maximum: 128 SU per hour per node

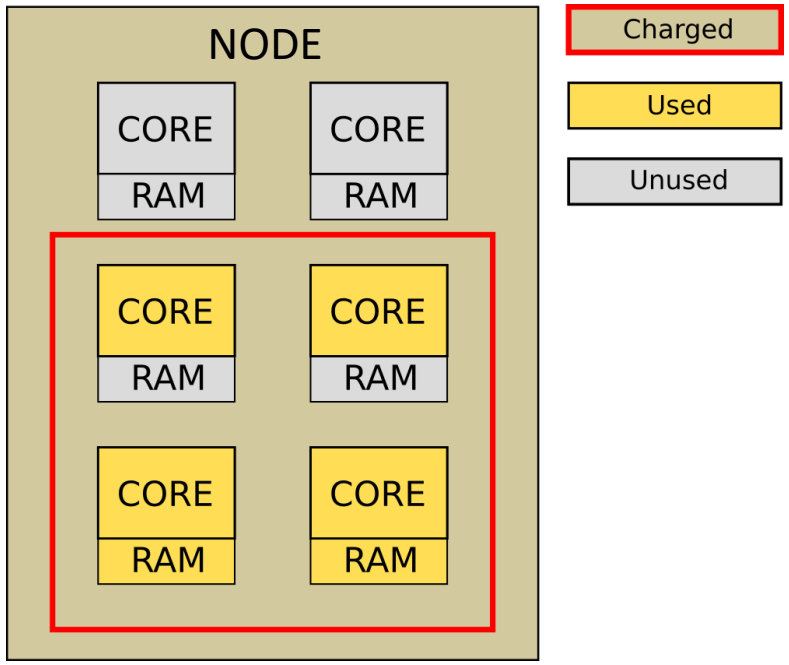


Setonix Accounting Model Examples

Examples of Setonix proportional node usage



Example 1: RAM proportion (2/3) is bigger than CPU cores proportion (1/2).



Example 2: CPU cores proportion (2/3) is bigger than RAM proportion (1/3).

Comparing Node Access

Exclusive Access

- Provides all available cores and memory on the node
- More predictable performance
- Process and thread pinning supported
- Allocation charged for full usage of nodes

Shared Access

- Other jobs may run on the node at the same time
- Less predictable performance with other jobs running on the node
- Currently sub-optimal process and thread pinning
- Allocation charged only for partial usage of node

NOTE: Support for process and thread pinning for shared jobs will improve in future system updates.

Checking Project Allocation Use

- Use the tool `pawseyAccountBalance` to see the amount of service units left to your project.

```
$ pawseyAccountBalance -p pawsey0001
Compute Information
-------------------
          Project ID     Allocation          Usage     % used
          ----------     ----------          -----     ------
          pawsey0001          25000          16395       65.6
```

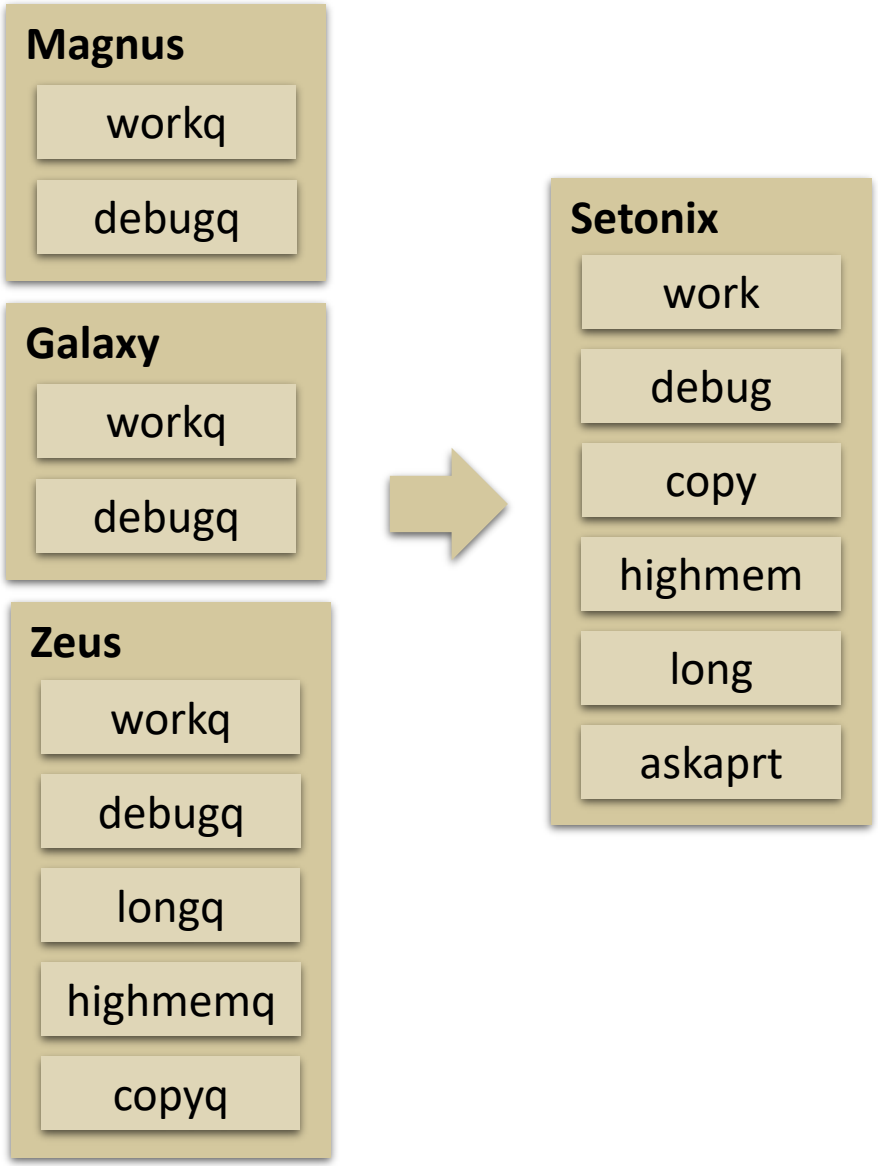
- Your allocation consumption is capped at 150% usage.
- If you reach the hard limit, your jobs will be assigned the *exhausted* quality of service (QOS) and won't run until the next quarter.
- More details @ Job Scheduling

pawsey

Section 3

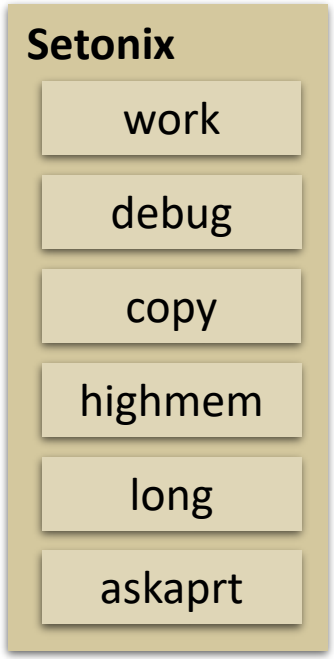
Overview of Slurm Configurations

Changes to Slurm Partitions



- Partitions have been simplified, and the letter "q" has been removed for clarity.
- The `copy`, `highmem`, and long partitions are now available directly on Setonix.
- The `askaprt` partition supports the operational workload of the ASKAP radio telescope.

Overview of New Slurm Partitions



`work` partition **(316 nodes, 128 cores per node)**

- Default partition, used for regular compute jobs
- 230 GB available memory per node, which is 1.8 GB per core

`debug` partition **(8 nodes, 128 cores per node)**

- Intended for short interactive jobs
- Use for benchmarking, testing, debugging, profiling and development

`copy` partition **(8 nodes, 32 cores per node)**

- Use for data transfers to/from Setonix
- 89 GB available memory per node, which is approximately 2.78 GB per core

`highmem` partition **(8 nodes, 128 cores per node)**

- Use for large memory workflows that require more than 230 GB RAM per node
- Supports up to 980 GB of available memory per node, which is 7.65 GB per core

`long` partition **(8 nodes, 128 cores per node)**

- Use for jobs that require wall times between 24 and 96 hours

`askaprt` partition **(180 nodes, 128 cores per node)**

- Supports the operational workflow of the ASKAP radio telescope

SLURM: work Partition

- The work partition contains most of the CPU nodes in Setonix. Use this partition for production jobs on the system.
- It is selected by default, but it is recommended to include it as a SLURM directive:

  ```
  #SBATCH --partition=work
  ```

- The maximum wall time that can be requested is 24 hours.
- Use a shorter wall time if you know your job needs less than 24 hours.
- Checkpoint-restart features in software can split longer jobs into multiple jobs that run for less than 24 hours.

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

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=24:00:00

module load module/version

srun ./program
```

SLURM: debug Partition

- The debug partition is used for:
  - Testing Slurm batch scripts
  - Debugging software
  - Code development
- It can be selected using SLURM directives:

  `#SBATCH --partition=debug`
- The maximum wall time request is 1 hour.

⚠ **IMPORTANT:** For a responsive partition, it is important that there are nodes available. Therefore, do not run production jobs in the debug partition.

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

#SBATCH --account=project
#SBATCH --partition=debug
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=01:00:00

module load module/version

srun ./program
```

SLURM: copy Partition

- The copy partition is used for:
  - Storing and retrieving files from Acacia object storage
  - Transferring files directly between the `/scratch` filesystem and storage external to Pawsey
  - Migrating files from the previous generation /oldgroup and /oldhome filesystems to the new filesystems
  - Note that the copyq partition on Zeus should be used to copy data to /newscratch
- It can be selected using SLURM directives:

  `#SBATCH --partition=copy`
- The maximum wall time request is 48 hours.
- Use `#SBATCH --mem=`*`memory`* to request additional memory if needed.

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

#SBATCH --account=project
#SBATCH --partition=copy
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=48:00:00

module load module/version

srun ./program
```

SLURM: highmem Partition

- Some jobs may need more memory than the 230 GB available on the compute nodes in the work partition.
- Specify the highmem partition using Slurm directives:

  `#SBATCH --partition=highmem`

- Specify the memory usage up to 980 GB with room for the operating system:

  `#SBATCH --mem=980GB`

- Note that there are only 8 nodes with high memory, and jobs may take longer to start running if they are in demand.
- Consider implementing distributed memory parallelism to distribute the computation across the memory of multiple nodes.

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

#SBATCH --account=project
#SBATCH --partition=highmem
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --mem=980GB
#SBATCH --time=24:00:00

module load module/version

srun ./program
```

⚠ **IMPORTANT:** Only use highmem if the extra memory is needed.

SLURM: `long` Partition

- Specify the `long` partition using Slurm directives:

  `#SBATCH --partition=long`

- Specify the wall time up to 96 hours:

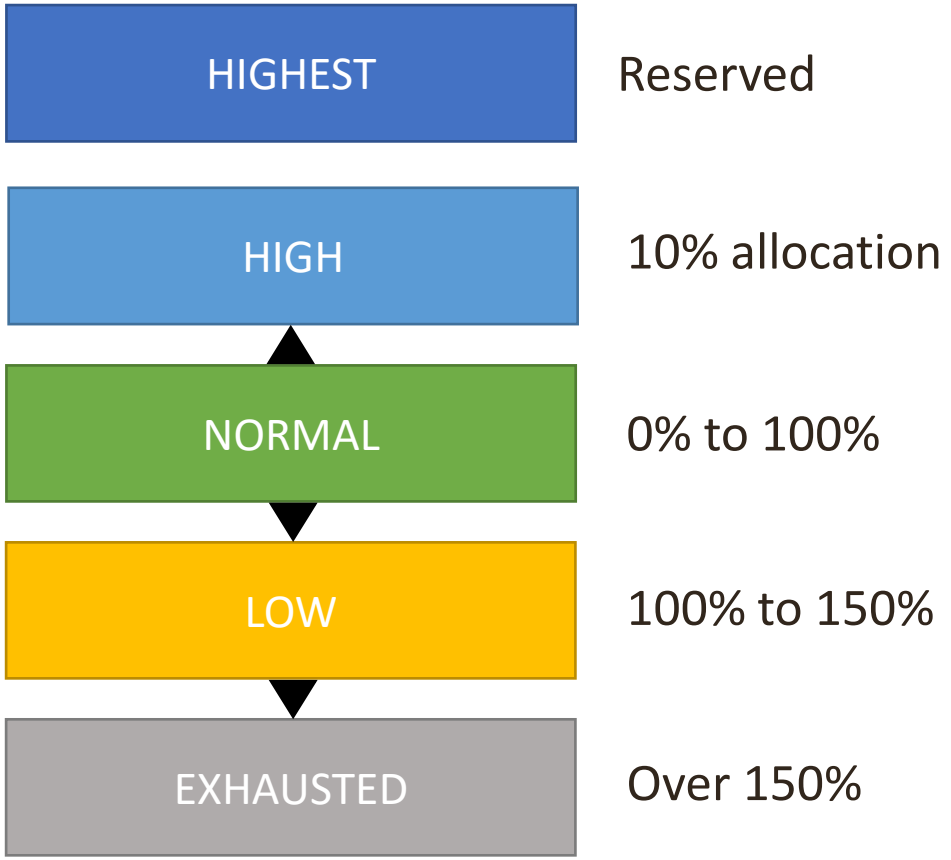
  `#SBATCH --time=96:00:00`

- There are **only** 8 nodes available for long running jobs, to minimise risk of lost compute time.
- Long-running jobs may take longer to start running.
  - Smaller pool of nodes are available.
  - Increased job duration means jobs end less frequently.
- Consider implementing checkpointing or distributed memory parallelism.

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

#SBATCH --account=project
#SBATCH --partition=long
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=96:00:00

module load module/version

srun ./program
```

SLURM: Quality-of-Service (QoS) Levels

- Quality-of-service levels alter the priority of jobs:
  - Highest: Reserved for critical work
  - High: Self-selected for 10% of allocation
  - Normal: Default level for all projects
  - Low: Projects over 100% quarterly usage
  - Exhausted: Projects over 150% quarterly usage
- Higher priority is user managed. Select using:
  - The `--qos=high` argument to sbatch, or
  - The #SBATCH `--qos=high` SLURM directive
- Exhausted projects can no longer run jobs.
- Quality-of-service levels reset each quarter.



Summary of Key Changes

Feature	Pre-Setonix	Setonix	Benefits, impacts and considerations
Partitions	Multiple systems	Single system	More concise partition names, available on the same system.
Access cut-off	Unlimited access at low priority	150%	Prevents significant overuse of the system once an allocation is exhausted each quarter.
Shared Access	Not available on Magnus	Available on Setonix	More efficient use of allocation, proportional node charging based on core and memory usage.

pawsey

Section 4

Create your Slurm Batch Script

Slurm Batch Script Overview

- No major changes, same format
- Slurm directives
  - Specify to the scheduler all of the resources that the job needs to run
  - Are used by scheduler when you submit a job
- Environment Setup
  - Activating software for use in the job
- Job launch
  - Executing commands when the job runs

Slurm batch scripts were previously called job scripts.

Start of Script

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

Slurm Directives

```
#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration
```

Environment Setup

```
module load module/version
```

Job Launch

```
srun ./program
```

Slurm Batch Script Directives: Questions to ask yourself.

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration
#SBATCH --exclusive

module load module/version

srun ./program
```

- Which project will the job be accounted under?
- Which partition is appropriate for the job?
- What type of parallelism does the code use?
  - How many nodes are required?
  - How many cores are required?
- How much memory per node is required?
- How much time will the job require?
  - Use the `long` partition for jobs over 24 hours (up to 96 hours)
- Is exclusive node access required?
- Is hyperthreading needed? (⚠ Talk to Pawsey staff first)

Grouping Tasks for Shared Access

- Node access is shared by default on Setonix.
- Slurm may place tasks across various nodes by default.
  - This is not an issue if your tasks do not communicate
- If your job relies on communication between tasks, we recommend ensuring they are allocated on the same node(s).
- Ensure the following SLURM directive is set:

  `#SBATCH --ntasks-per-node=`*`tasks-per-node`*
- Alternatively, use:

  `#SBATCH --nodes=`*`nodes`*

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration

module load module/version

srun ./program
```

Requesting Exclusive Usage

- Nodes are now shared by default. This means that exclusive use must be specifically requested.
- There may be use cases that do not use all the cores or memory of a node, but still want exclusive access to the nodes.
  - An example is a code bound by memory bandwidth.
- Use the following SLURM directive to disable node sharing:

  `#SBATCH --exclusive`

- In exclusive usage the job accounting will charge for the entire node.

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration
#SBATCH --exclusive

module load module/version

srun ./program
```

Setting the Environment

- Check the availability of modules, and update the versions in your Slurm batch script environment.
  - Versions should be provided when loading modules.
- For software provided by the module system, the module already includes all dependencies.
  - This is different from previous systems, which loaded additional dependencies automatically as separate modules.
- Check hard-coded paths to ensure they are correct.
  - For software paths, $MYGROUP may need to be replaced with $MYSOFTWARE (or change `/group` to `/software`).
  - If you previously used $MYGROUP to run jobs, now use $MYSCRATCH instead (or change `/group` to `/scratch`).
- Include OpenMP environment variables as usual if required.
  - More details @ OpenMP

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration

module load module/version
export VARIABLE=value

srun ./program
```

Job Launching

- `srun` will use the configuration from the directives by default:

  `srun ./program`

- For jobs where parts of the workflow require a subset of the resources requested, these defaults can be overridden:

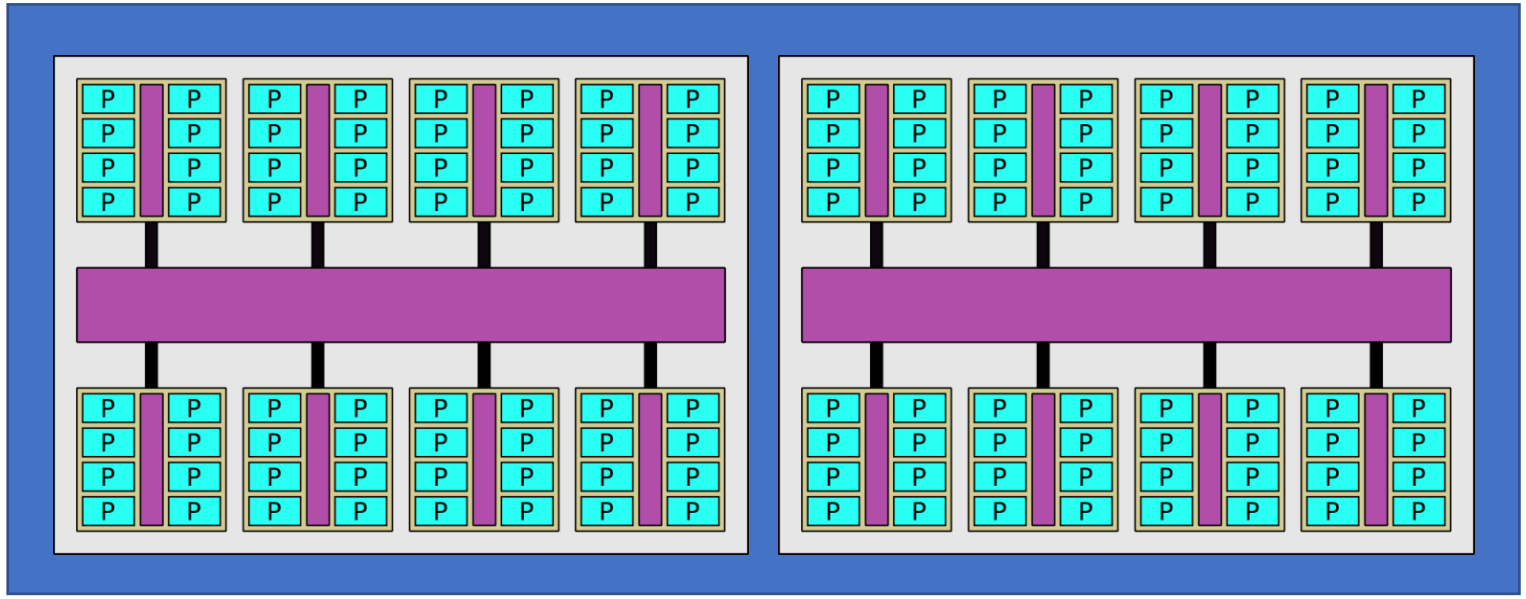
  `srun -N nodes -n tasks –c threads ./program`

- For GPU jobs, continue to use Topaz during Phase 1
- More details @ Example Workflows

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

#SBATCH --account=project
#SBATCH --partition=partition
#SBATCH --ntasks=tasks
#SBATCH --ntasks-per-node=tasks-per-node
#SBATCH --cpus-per-task=threads
#SBATCH --time=duration

module load module/version

srun -N nodes -n tasks –c threads ./program
```

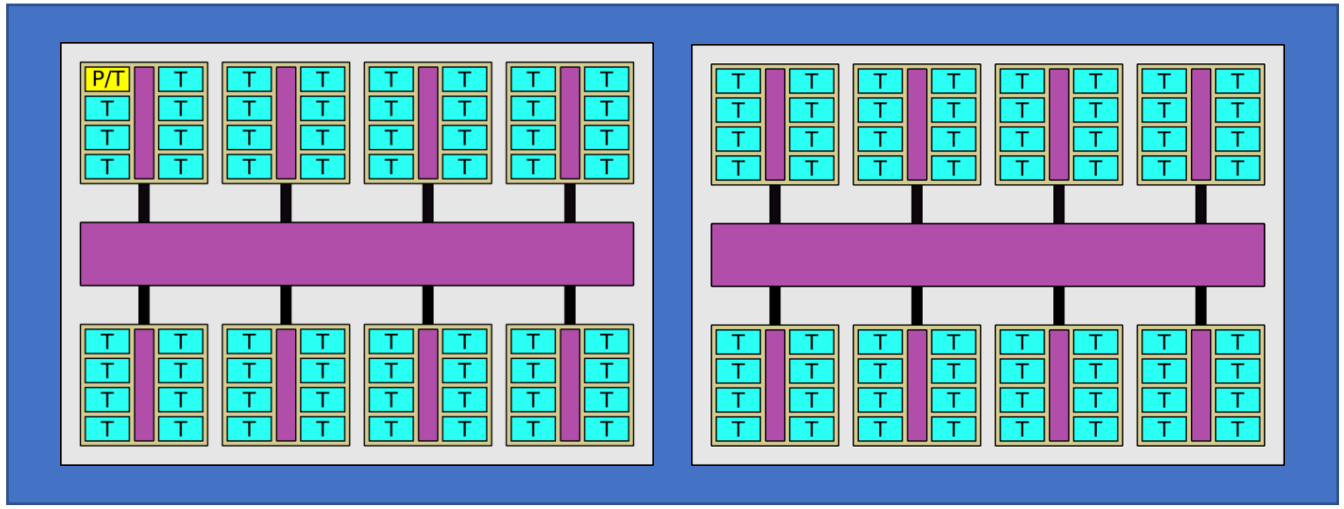
Job Launching: Exclusive Access - MPI Example

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

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=512
#SBATCH --ntasks-per-node=128
#SBATCH --time=24:00:00
#SBATCH --exclusive

module load module/version

srun ./program
```



- The script requests 4 nodes (512/128). For clarity, of the four nodes requested, one node containing two sockets is shown.
- One process (P) is launched per core (blue box).
- In this example, all cores are used.

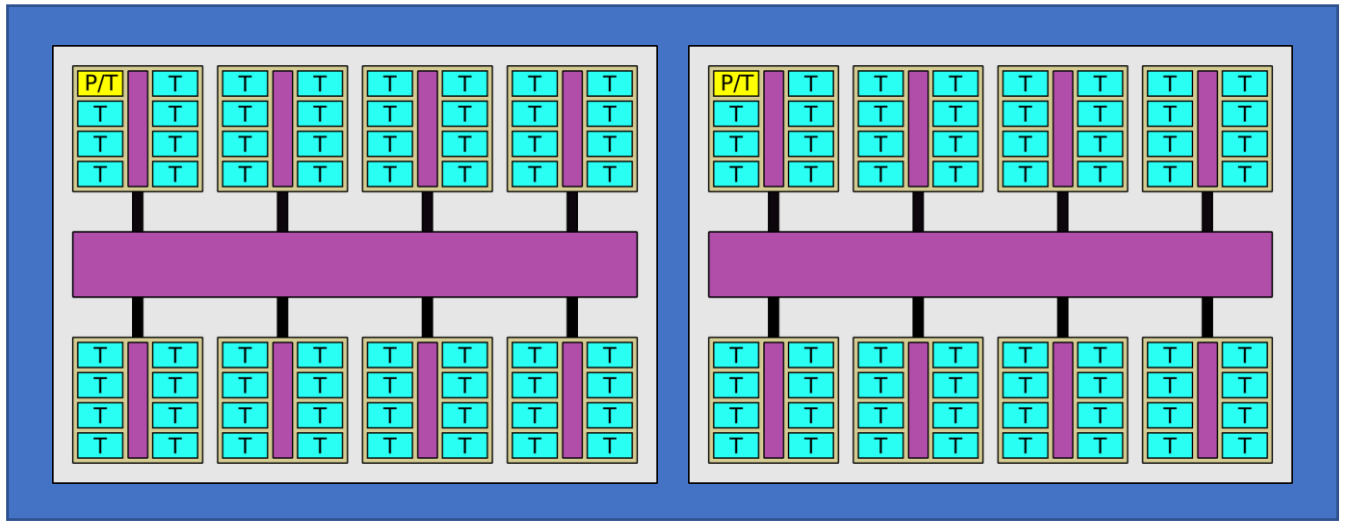
Job Launching: Exclusive Access - OpenMP example



- One process (P) per node, the process spawning 128 OpenMP threads (T)

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

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=128
#SBATCH --time=24:00:00
#SBATCH --exclusive

module load module/version

# OpenMP environment variables
export
OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export OMP_PROC_BIND=close/spread
export OMP_PLACES=cores/threads

srun ./program
```

Job Launching: Exclusive Access - Hybrid MPI + OpenMP Example

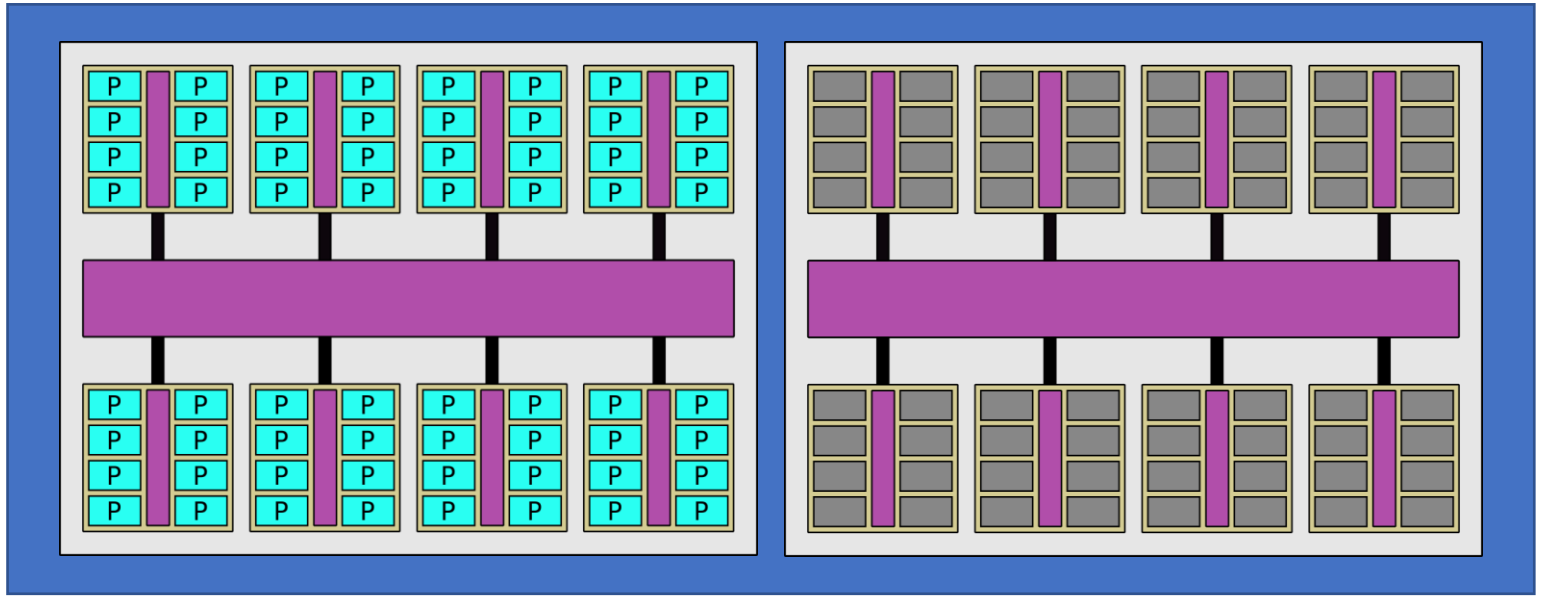


- One process per socket, each spawning 64 OpenMP threads.
- The script requests 4 nodes (8/2). For clarity, one node is shown.

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

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=64
#SBATCH --time=24:00:00
#SBATCH --exclusive

module load module/version

# OpenMP environment variables
export
OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export OMP_PROC_BIND=close/spread
export OMP_PLACES=cores/threads

srun ./program
```

Job Launching: Shared Access - MPI Example

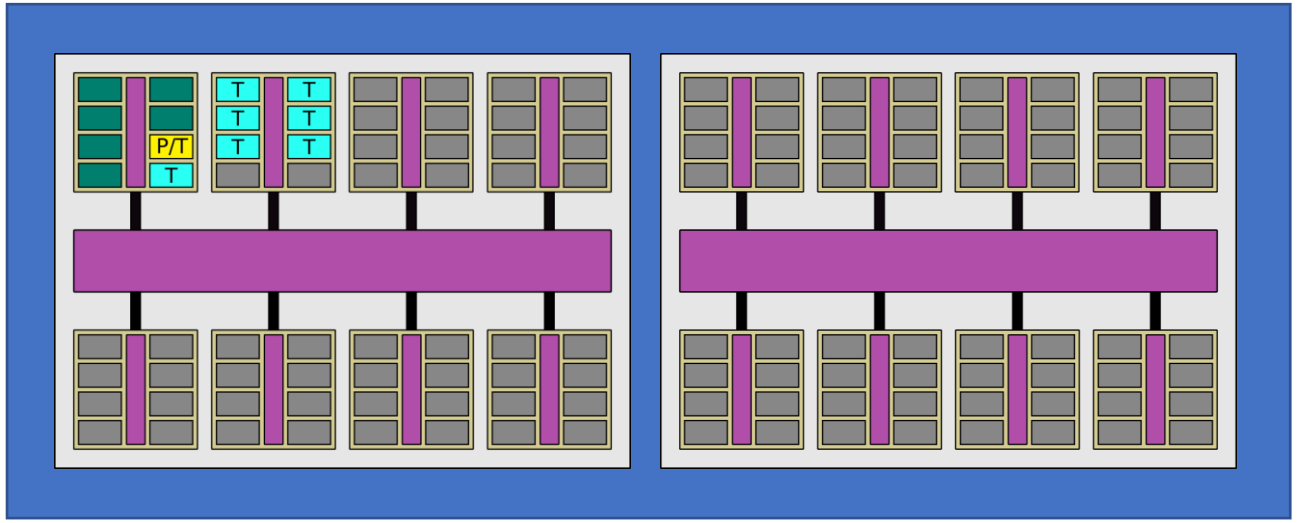
```
#!/bin/bash --login

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=64
#SBATCH --ntasks-per-node=64
#SBATCH --ntasks-per-socket=64
#SBATCH --time=24:00:00

module load module/version

srun –m block:block:block
./program
```



- Using `--ntasks-per-socket=64` confines the job to a single socket, which is optimal when sharing node.
- The `-m block:block:block` ensures tasks are packed together.
- Try to use multiples of 8 MPI tasks for best L3 cache utilisation.

Job Launching: Shared Access - OpenMP Example

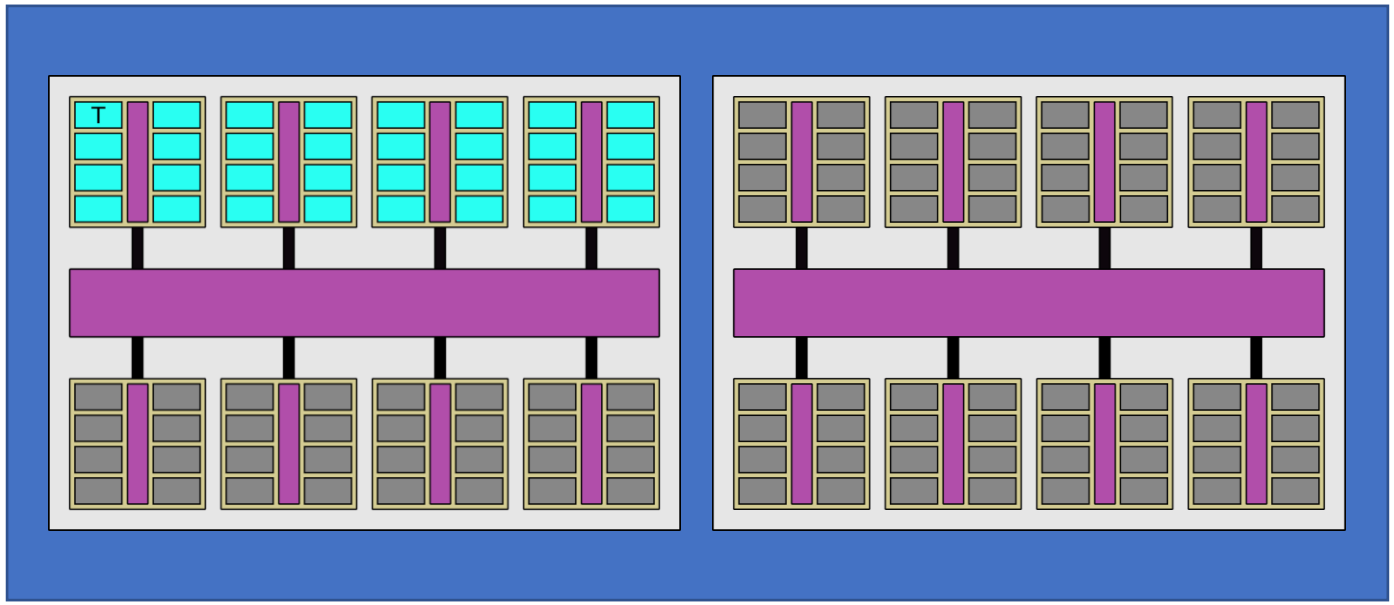
```
#!/bin/bash --login

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --tasks-per-node=1
#SBATCH --ntasks-per-socket=1
#SBATCH --cpus-per-task=8
#SBATCH --time=24:00:00

module load module/version

# OpenMP environment variables
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun -m block:block:block ./program
```



- Another job has used 6 cores, so this job is spread over two different chiplets. If the other job used 8 cores, this job would be fully allocated on the second chiplet.

- Threads placement may vary from job to job and the current usage of resources allocated to other jobs on the node (greyed out region), as shown by the two diagrams.
- The `-m block:block:block` ensures threads are packed together.
- Try to use multiples of 8 OpenMP threads for best L3 cache utilisation.

Job Launching: Shared Access - Memory Example




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

#SBATCH --account=project
#SBATCH --partition=work
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=57GB
#SBATCH --time=24:00:00

module load module/version

srun –m block:block:block ./program
```

- Memory requests are pinned to cores, resulting the allocation of cores.

Summary of Key Slurm Batch Script Changes

Change	New	Impacts / Considerations
Resource request	128 cores available per node	24 cores on Magnus
Memory request	230 GB available memory per node	~59 GB available on Magnus
Cache layout	L3 cache domains for groups of 8 cores	Only per-socket cache domains on Magnus

pawsey

Section 5

Submit and Monitor Your Job on Setonix

The Process of Scheduling a Production Job on Setonix

Prepare

- Before submitting a job, prepare a directory on `/scratch`

Test (optional, strongly recommended)

- For the first time running a script, perform scaling tests in the debug partition.

Submit

- Submit your job from the `/scratch` filesystem using `sbatch`

Monitor

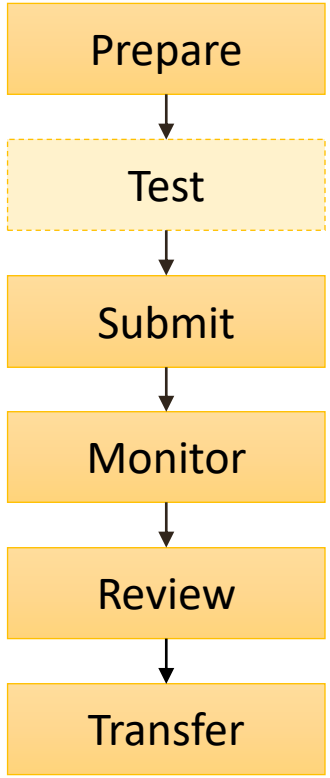
- Check your jobs have submitted correctly using `squeue`
- Do something else while the jobs queue and run.

Review

- Review the output data from your jobs.

Transfer

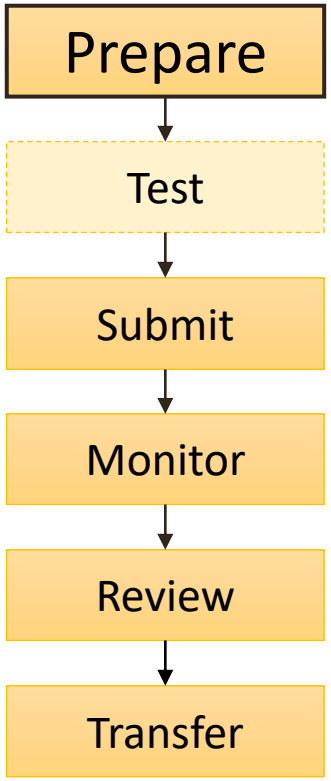
- Transfer files from the /scratch filesystem once they are no longer needed for active jobs.



Prepare to Run the Job

Before submitting a job:

- Ensure the correct versions of any required software are available.
  - See Migration Training Modules 3: Using Modules and Containers and Migration Training Module 4: Installing and Maintaining Your Software for more details on using and installing software
- Create a working directory for the job on `/scratch`
- Create or copy the Slurm batch script to the working directory.
  - It is useful to have a collection of template Slurm batch scripts.
  - These should be located in your /software directory.
- Prepare input data (use sub-directories if needed).
  - See Migration Training Module 2: Supercomputing Filesystems for details on transferring data directly to `/scratch` from local machines.
  - See Migration Training Module 6: Using Data Throughout the Project Lifecycle for details on staging data to /scratch from the Acacia object store.
- Create directories for output data if needed.



For Migration Training Modules, see:

- Materials, and
- Recordings

(Optional, but Recommended) Test the Slurm Batch Script

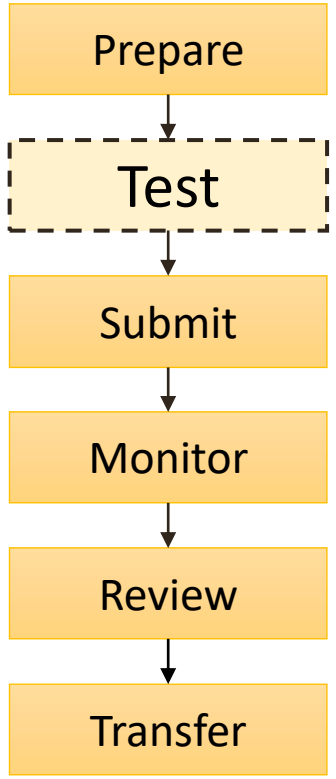
- The first time running a Slurm batch script, prepare a small test case:
  - Reduce the number of time steps
  - Reduce the run time or execution time of the program
  - Reduce the parameter space or amount of data processed
  - Reduce the number of nodes needed for the job
- Perform a short test run (5-10 minutes) in the debug partition:

```
sbatch testScript.slurm
```

- (Optional) Override Slurm directives to test a job in the debug partition:

```
sbatch --partition debug --time 00:15:00 jobscript.slurm
```

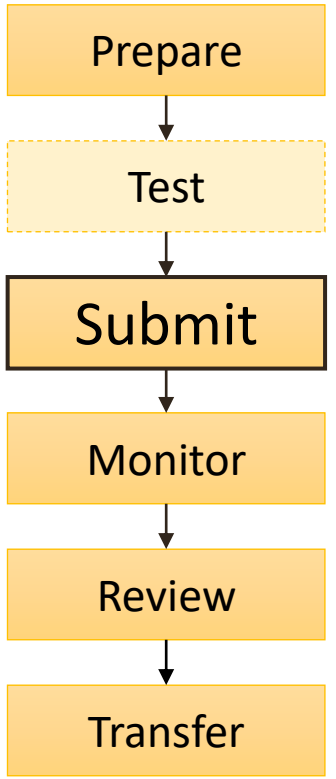
- We recommend trying multiple resource configurations and requests to identify the best scaling for your workflow.
  - Use multiples of 8 cores for scaling tests.
  - Test with exclusive nodes access for predictable performance of partial node jobs.



⚠ **IMPORTANT:** Overriding Slurm directives is not recommended for production jobs.

Submit the Job

```
$ cd $MYSCRATCH
$ ls
test.sh
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up 1-00:00:00      1 drain$ nid001145
work*        up 1-00:00:00      1  maint nid001070
work*        up 1-00:00:00     10  down* nid[001071,001077,…
work*        up 1-00:00:00      1   resv nid001214
work*        up 1-00:00:00    303   idle nid[001008-001069,…
long         up 4-00:00:00      8   idle nid[001316-001323]
copy         up 2-00:00:00      8   idle dm[01-08]
…
askaprt      up 1-00:00:00    176   idle nid[001324-001487,001492-001503]
debug        up    1:00:00      1    mix nid001006
debug        up    1:00:00      7   idle nid[001000-001005,001007]
highmem      up 1-00:00:00      8   idle nid[001504-001511]
$ sbatch test.sh
  Submitted batch job 4889
$ ls
Introductory-Supercomputing  kmeans-implementations  slurm-4889.out  test_matmul  test.sh
```



(Optional) Use Interactive Sessions

- It is possible to run serial or parallel jobs interactively through Slurm. This is a very useful feature when:
  - Debugging
  - Compiling
  - Pre-processing/post-processing
  - Running GUI applications
- You can use the `salloc` command to obtain an interactive session with active resources over a Slurm job allocation. The Slurm directives that would normally be provided in the batch script must instead be given as options to the `salloc` command.
- More information available @ Job Scheduling

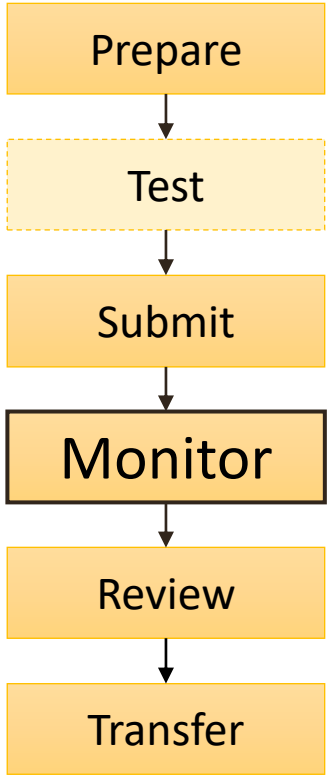
```
$ salloc -p work
salloc: Granted job allocation 4861
salloc: Waiting for resource configuration
salloc: Nodes nid001008 are ready for job
```

Monitor the Job: Before it runs

Use the `squeue` command to check the list of queued jobs.

- The command `squeue --me` will show only your jobs.
- By default, the command shows basic information, such as job ID, user, account billed, partition, start and end times.

The command `sprio` shows what makes up the priority of your job.



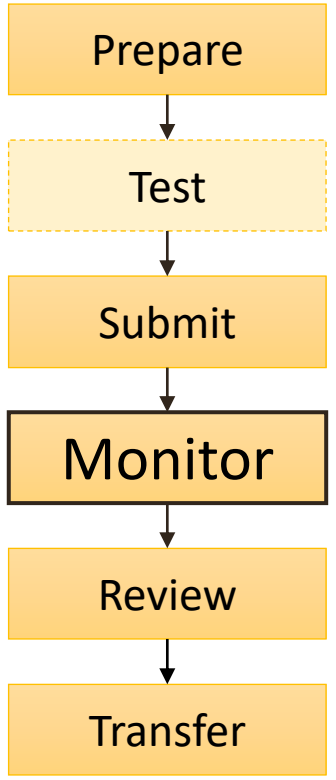

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4861      work interact username PD       0:00      1 Priority
              4853      work interact username PD       0:00      1 Priority
$ sprio -j 40006
 JOBID PARTITION   PRIORITY       SITE        AGE  FAIRSHARE  PARTITION        QOS
 40006 askaprt        75443          0          0        444          0      75000
```

Monitor the Job: While it is running

You still use `squeue`. In addition, you can use the `sacct` command.

Check output files are starting to be written as expected.

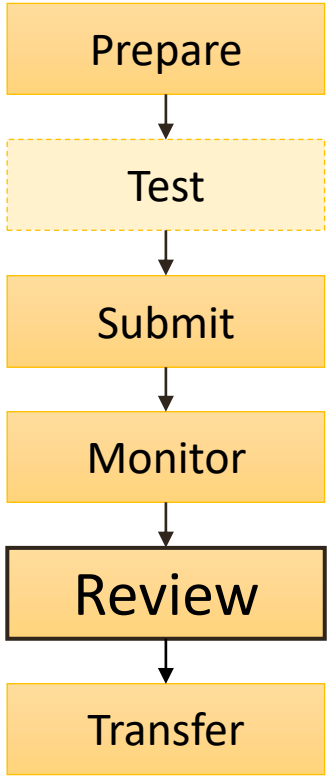
```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4861      work interact username  R       0:05      1 nid001008
              4853      work interact username  R      17:53      1 nid001008
```



Review the Job After it Completes

- You must use `sacct` to query job information. Pass the job ID using the `-j` option. Check the man page for the fields that can be requested.
- Did you receive any error codes or error messages in the job output?

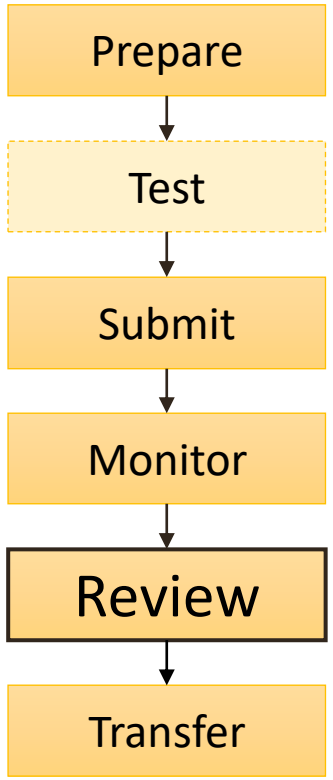
```
$ sacct -j 40006
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
40006        sillyallo+    askaprt pawsey0001          2  COMPLETED      0:0
40006.batch       batch            pawsey0001          2  COMPLETED      0:0
40006.extern     extern            pawsey0001          2  COMPLETED      0:0
40006.0        hostname            pawsey0001          2  COMPLETED      0:0
```



Review the Job Efficiency

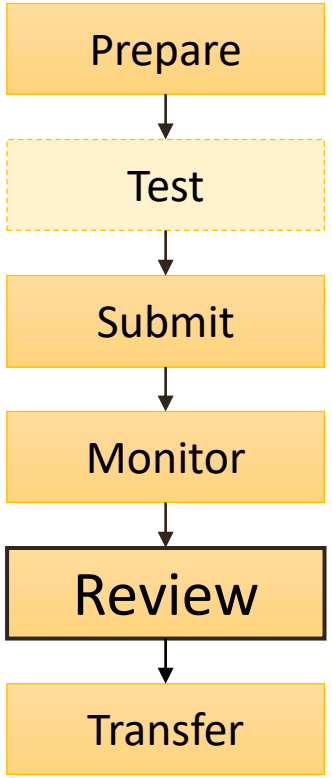
- The `seff` executable displays the resource usage of a job in a human-readable way.

```
$ seff 40006
Job ID: 40006
Cluster: setonix
User/Group: username/username
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 00:03:28 core-walltime
Job Wall-clock time: 00:01:44
Memory Utilized: 548.00 KB
Memory Efficiency: 0.03% of 1.86 GB
```



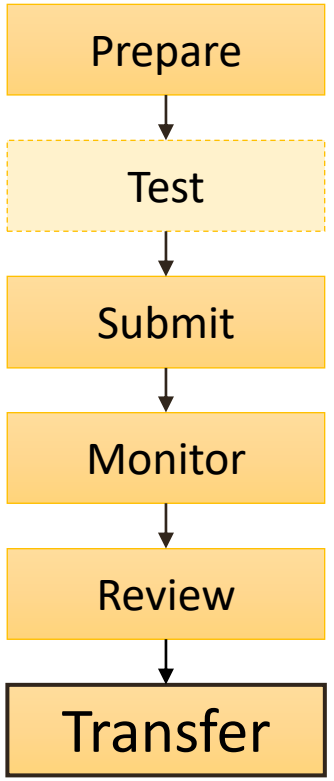
Review the Job Output

- Is the output from your job what you expected? For example:
  - Are all the files you expected present?
  - Is the amount of data as expected?
  - Were all checkpoints completed?
  - Is the data that was written correct?
- Check Slurm batch scripts, directory permissions, software installations and input parameters if things have not worked as expected.



Transfer the Files

- Transfer files from the `/scratch` filesystem once they are no longer needed for active jobs.
- Select the destination for the data, which can include:
  - Pawsey's Acacia object storage
  - Institutional data storage infrastructure
  - Workstations, laptops, or other media
- See Migration Training Module 6: Using Data Throughout the Project Lifecycle, for more information on using Acacia object storage.
- See Migration Training Module 2: Supercomputing Filesystems, for more information on using the Setonix data mover nodes to transfer files to other external filesystems.



For Migration Training Modules, see Materials and Recordings.

How do I get help?

Migration Documentation & Migration Guides

- Setonix Migration Guide
- Setonix User Guide
- Supercomputing Documentation

Migration Training Materials & Video Recordings

- Upcoming Migration Training
- Recordings: Pawsey YouTube Setonix Migration Phase 1 Playlist
- Materials: Setonix Migration Training Materials (PDFs)

Help Desk

- Help Desk
- Email: help@pawsey.org.au

pawsey

Thank you for attending!

Please complete this short survey:

https://www.surveymonkey.com/r/Y3YFYHQ