Pawsey

Using GPUs on Setonix

Setonix Phase 2 Migration

May 2023, Version 1.00

Focus for this Training

Learning Outcomes:

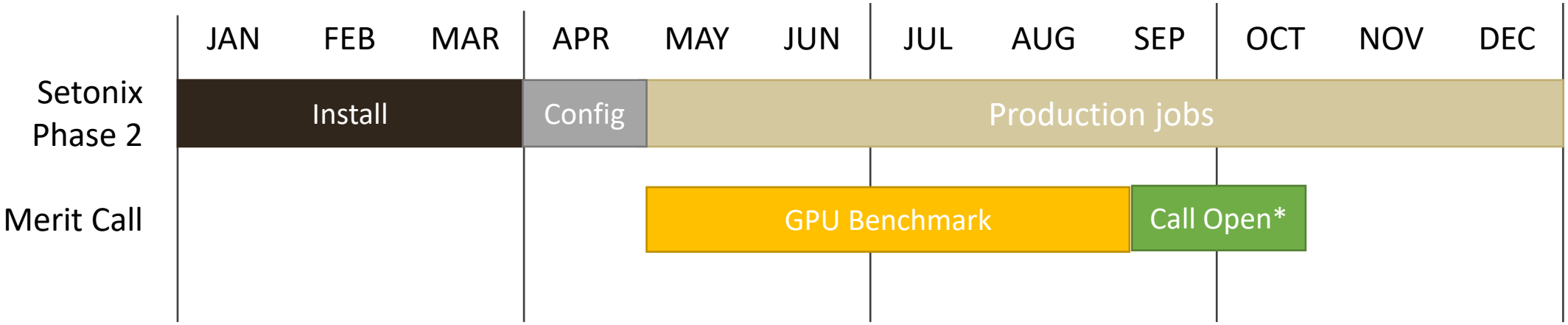
- Review the timing of the availability of Setonix Phase 2 and its impact on Topaz users
- Be aware of the available Setonix compute resources following the Phase 2 upgrade
- Be able to develop a plan to migrate Topaz workflows to Setonix Phase 2 (if needed)
- Be aware of available GPU-enabled programs in the Setonix Phase 2 software stack
- Find out more about the accounting model for the GPU partitions on Setonix Phase 2
- Find out how to schedule and run GPU jobs on Setonix Phase 2

pawsey

Section 1

GPU Migration Overview

GPU Migration Schedule



- Full Setonix GPU allocations are distributed evenly across calendar year 2023: quarters 2, 3, and 4.
- Topaz will run in parallel until later in 2023. Existing Topaz projects can run during the overlap period. However, Topaz should only be used for software that does not yet run on Setonix.
- For more information on the annual allocation call, see Accessing Pawsey Supercomputers.

⚠ **IMPORTANT:** Topaz compute nodes and filesystems (/group, /scratch, /home) will be decommissioned later in 2023.

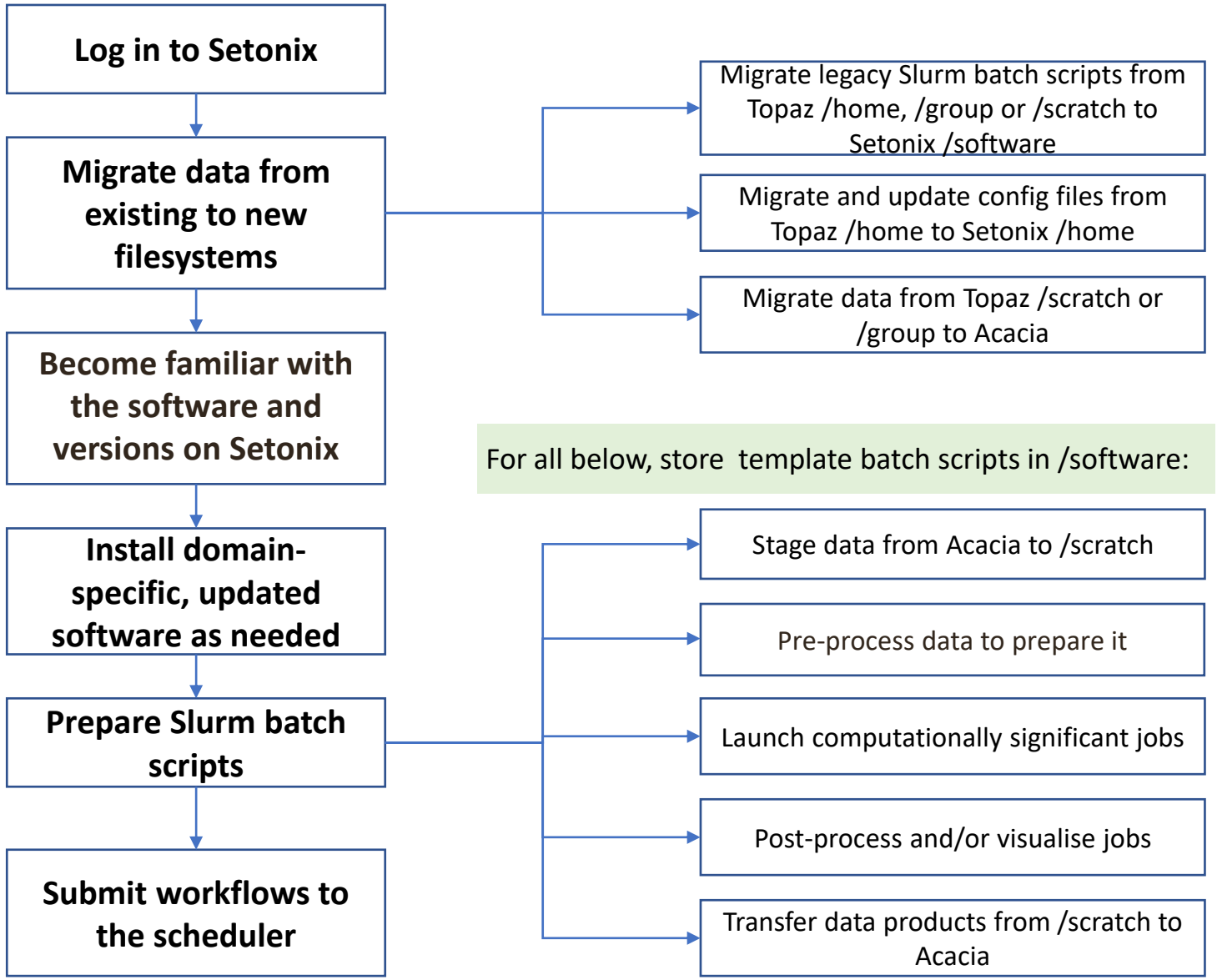
Migration Pathway for Topaz projects

Overview

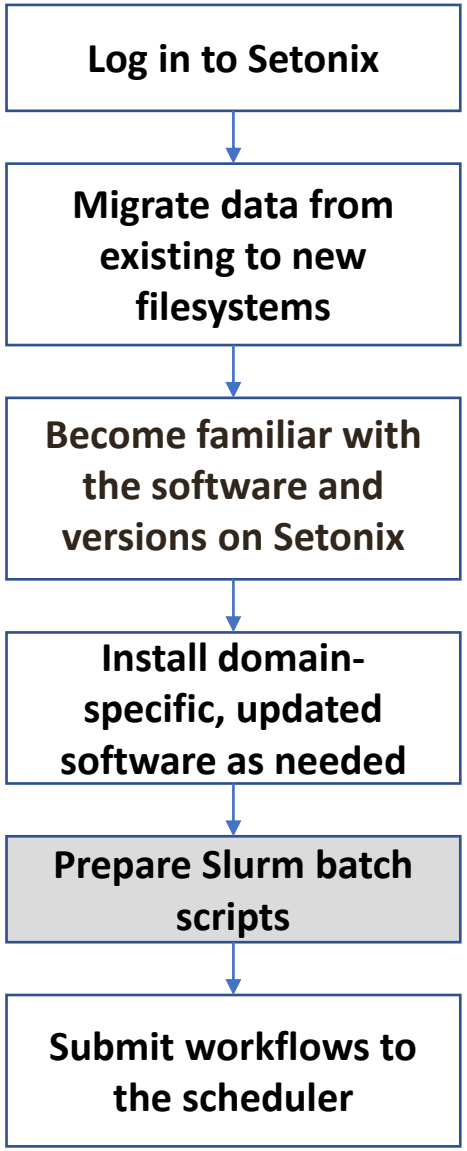
This flowchart shows key steps for migrating to Setonix from Topaz.

Best Practices

- Move important data. Delete data that is no longer needed.
- Do not transfer software. Reinstall or use new versions.
- Set up environments in Slurm batch scripts, not login scripts.
- Consider watching Using Supercomputers for details.



Use Case: Migrating a GPU Workflow from Topaz



Overview

The process of migration is similar to Phase 1 migration of CPUs from Magnus and Zeus to Setonix.

Key Changes

- The GPU nodes have 64 cores and 256 GB of memory per node.
- There are 4 AMD MI250X GPU cards or 8 logical GPUs (or GCDs) available per node.

Example single GPU job

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

#SBATCH --account=project-gpu
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --gpus-per-task=1
#SBATCH --time=24:00:00

srun –c 8 --gpu-bind=closest ./program
```

⚠ **IMPORTANT:** More detail and examples are provided in the following sections.

pawsey

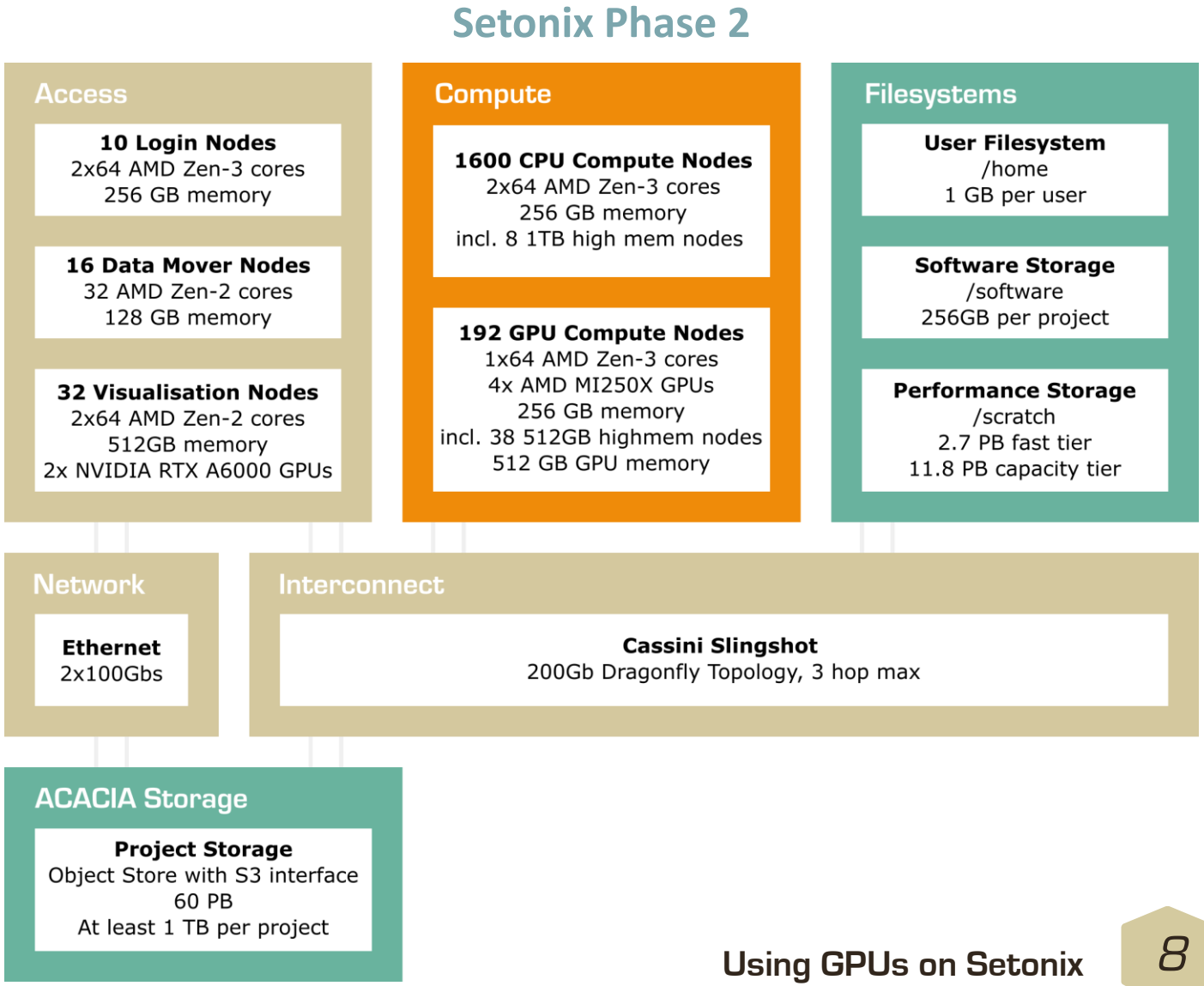
Section 2

GPUs on Setonix

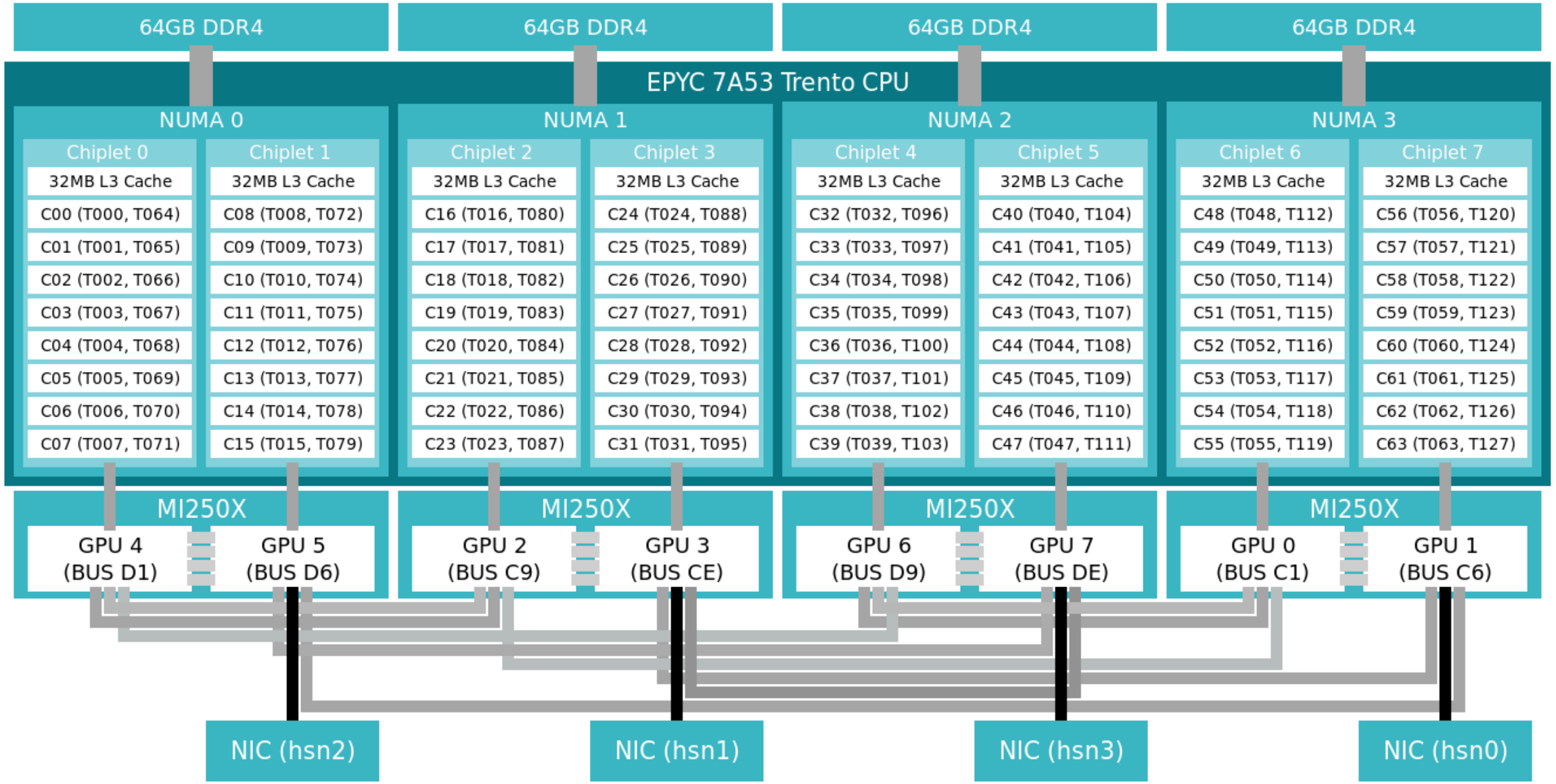
Setonix Phase 2 Hardware Overview

Phase 2 provides:

- 192 GPU compute nodes, each with:
  - 4 x MI250X GPUs, which provide 8 logical GPU devices per node
  - 512 GB of total GPU memory
  - 1 x 64 core AMD Trento CPU
  - 256 GB of CPU memory, with 38 high memory nodes containing 512 GB
  - 4x 200Gb connections to interconnect
- 1088 additional CPU compute nodes
- Additional login, data mover and visualisation nodes
- Upgraded 200Gb interconnect



Setonix Phase 2 Node Architecture

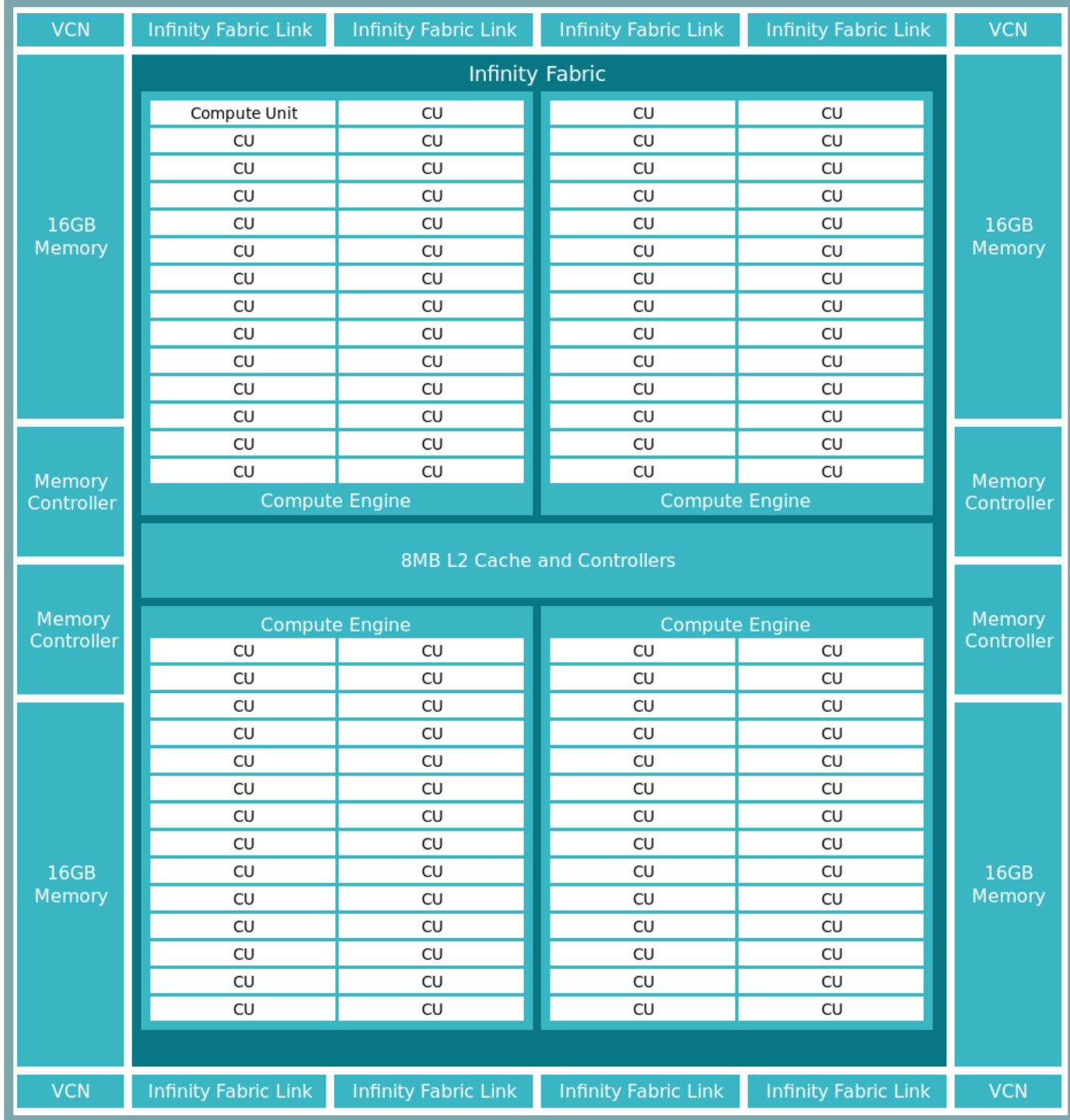


Setonix Phase 2 GPU Architecture

- Each Setonix GPU node has 4 MI250X cards each with 2 logical GPUs corresponding to a CDNA-2 Graphics Compute Die (GCD).
- One GCD is shown here, with:
  - 110 Compute Units (CU)
  - 64 stream processors per CU
  - 4 matrix cores per CU
  - 64 GB of HBMe memory
- For more information see:

  https://www.amd.com/en/technologies/cdna2

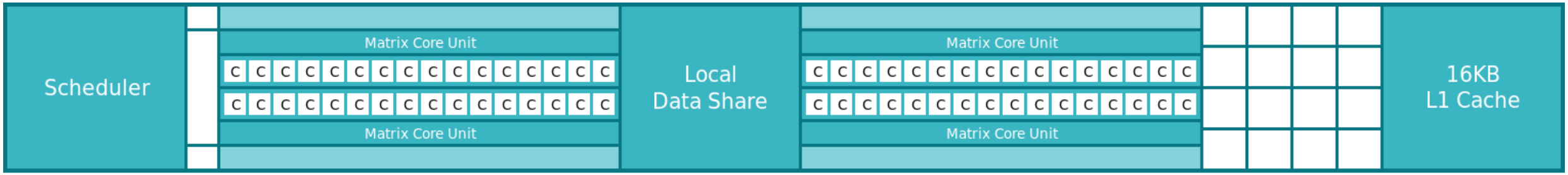
For comparison, a Topaz gpuq node had 2 Nvidia V100 GPUs.



CDNA-2 Compute Unit

- Shader cores (C) are used by standard HIP APIs and kernels.
- Matrix cores (similar to NVIDIA tensor cores):
  - Perform accelerated small matrix multiplications in half, single, and double precision, and
  - Require specialised APIs and/or libraries, such as rocWMMA.
- Wavefronts (similar to NVIDIA warps) are 64 lanes wide.
- For more information see:

  https://www.amd.com/en/technologies/cdna2

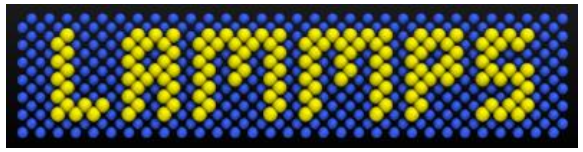


pawsey

Section 3

GPU Software Available on Setonix

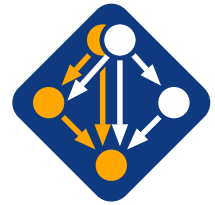
GPU Software Support: Applications and Libraries



- There is a wide range of software with varying levels of AMD GPU capability.
- Pawsey is actively adding modules for GPU-enabled builds of popular software, as available.
- Check with your software vendor or documentation for details.

GPU Software Support: Package Deployment

- GPU-enabled modules have the **`-amd-gfx90a`** suffix.
- Bare-metal builds are available with full GPU functionality of the software.
- Singularity enables running GPU-accelerated containers.
  - Containers are available as modules, but there are some limitations:
    - Containers can currently scale only up to 1 node.
    - Some containers can currently use only a single GPU.
  - The AMD Infinity Hub has a collection of containerised applications: https://www.amd.com/en/technologies/infinity-hub
  - The E4S initiative provides containerised build tools and application stacks for HPC: https://e4s-project.github.io



AMD Infinity Hub

pawsey

Section 4

GPU Accounting on Setonix

Setonix GPU Accounting

- GPU allocations are separate to CPU allocations in SLURM.
- Use the **`-gpu`** suffix on your usual account name when running GPU jobs:

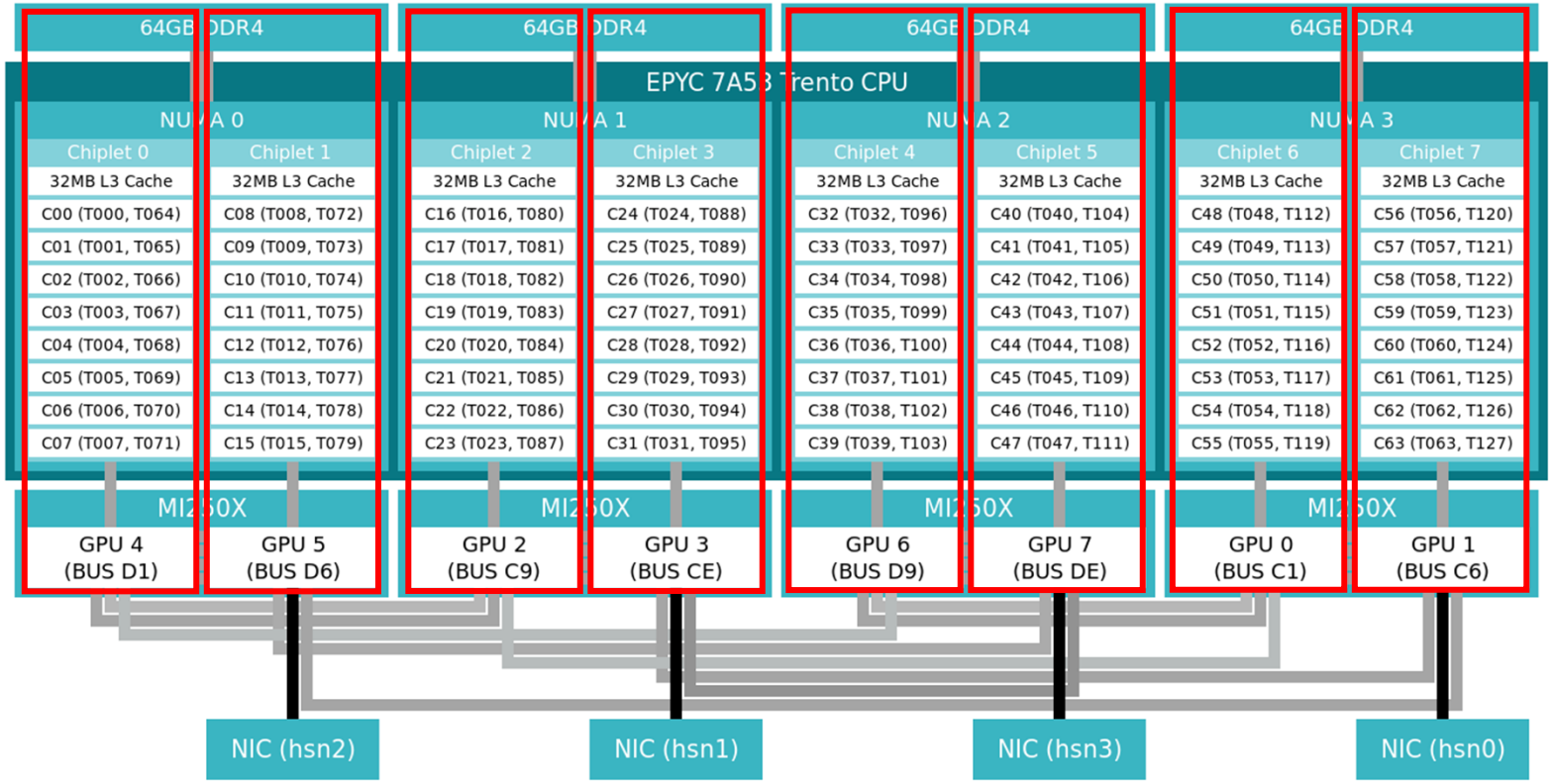
```
#SBATCH --account=project-gpu
```

- Use this only for the Slurm GPU request; use the usual project name for files.
- GPU nodes consume 512 SU per hour.
  - For comparison, CPU nodes consume 128 SU per hour.
  - The difference is based on energy consumption.
- The **`pawseyAccountBalance`** tool can be used to check GPU allocation usage:

```
$ pawseyAccountBalance -p project-gpu
```

Accounting for Shared Access on GPU Nodes

- There are 8 cores and 32GB of memory associated with each logical GPU (or GCD).
- Slurm will account for the largest proportion of the node resources you request.
- 8 cores should be requested per GPU for optimal scheduling.



Setonix GPU Accounting Examples

- With exclusive access, nodes will be charged 512 SU per node per hour regardless of utilisation.
- With shared access, nodes will be charged based on the largest proportion used.
- Use 8 cores and less than 32 GB per GPU for optimal shared access.
- Each logical GPU has 64 GB of global HBMe memory included.

Mode	GPUs	Cores	Memory	SU per hour
Exclusive	8	64	256 GB	512 SU
Exclusive	8	1	1 GB	512 SU
Shared	1	8	32 GB	64 SU
Shared	4	32	128 GB	256 SU
Shared	2	1	1 GB	128 SU
Shared	1	1	128 GB	256 SU
Shared	1	32	1 GB	256 SU

pawsey

Section 5

Scheduling Jobs on Setonix

New GPU Slurm Partitions

gpu partition (134 nodes)

- Used for production GPU jobs
- 64 cores and 8 logical GPUs (i.e. GCD) per node
- Up to 256 GB per node or 32 GB per logical GPU
- Up to 24-hour walltime

gpu-dev partition (20 nodes)

- Used for debugging GPU jobs or developing GPU software
- Same node configuration as gpu partition, apart from 4-hour maximum walltime
- Not for production work, to ensure availability of nodes for debugging and development
- Some of these nodes may be repurposed to the gpu partition based on usage over time

gpu-highmem partition (38 nodes)

- Used for production GPU jobs with higher CPU memory requirements
- Similar node configuration as gpu partition, except:
  - Up to 512 GB per node or 32 GB per logical GPU (i.e. GCD)

Requesting GPUs in Slurm: Exclusive Node Access

For exclusive GPU jobs, the Slurm batch scripts should additionally specify:

- A project account with the -gpu suffix
- A partition with GPUs: gpu, gpu-dev or gpu-highmem
- The number of GPUs per task
- Exclusive access

⚠ **IMPORTANT:** **`--gpus-per-task`** or **`--gpus-per-node`** is recommended, but **`--gres=gpu:#`** is also usable.

Example exclusive GPU job

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

#SBATCH --account=project-gpu
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --gpus-per-task=8
#SBATCH --time=24:00:00
#SBATCH --export=NONE
#SBATCH --exclusive

module load module/version

srun -c 1 -gpu-bind=closest ./program
```

Requesting GPUs in Slurm: Shared Node Access

For shared GPU jobs the Slurm batch scripts should additionally specify:

- A project account with the -gpu suffix
- A partition with GPUs: gpu, gpu-dev or gpu-highmem
- The number of GPUs per task

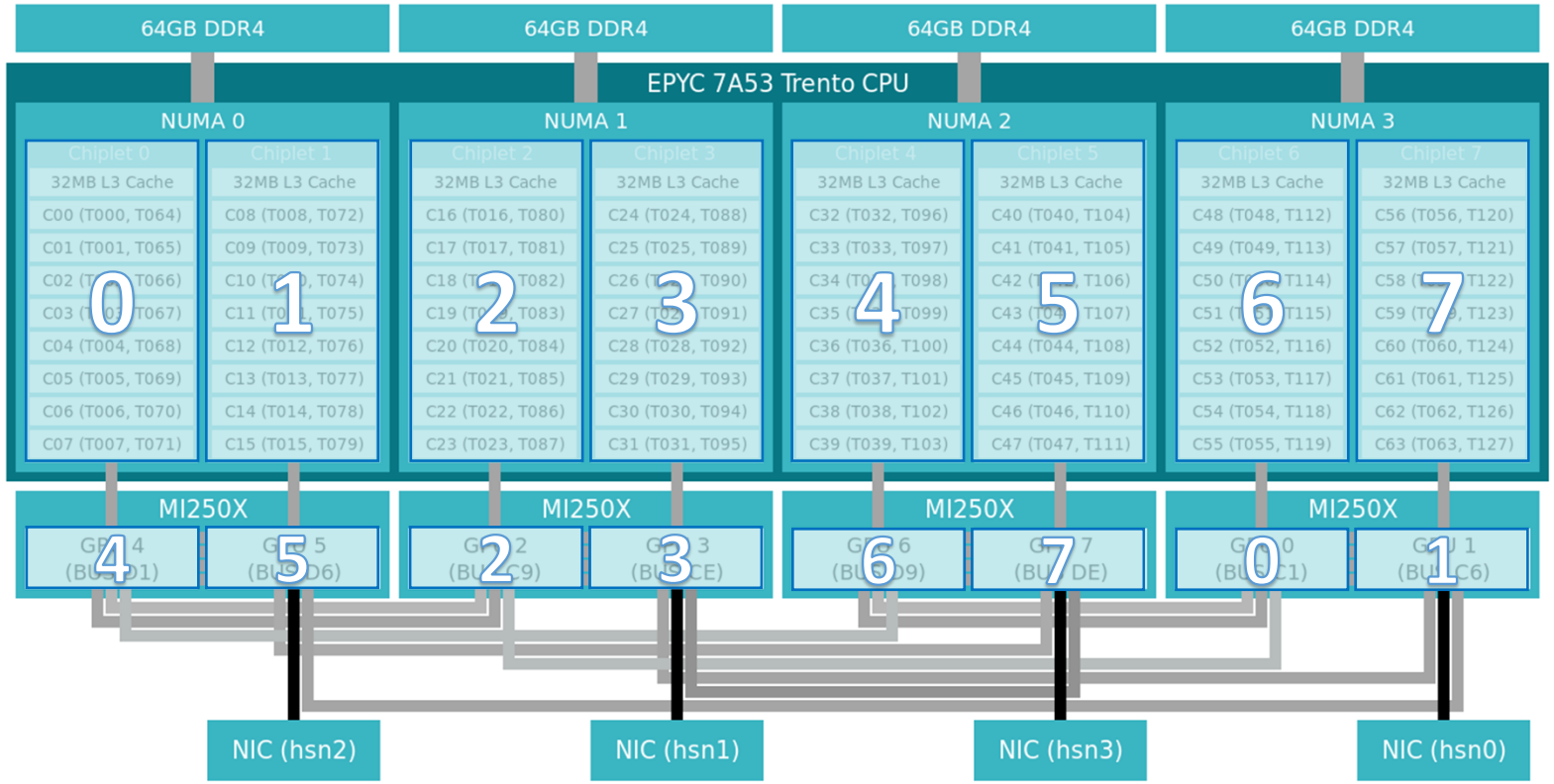
It is best practise to request 8 CPU cores for each GPU in a shared GPU Slurm batch script.

Example shared GPU job

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

#SBATCH --account=project-gpu
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --gpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --export=NONE

module load module/version

srun –c 8 –gpu-bind=closest ./program
```

CPU-GPU Binding for Optimal Performance

- Each logical GPU (or GCD) is connected to a chiplet of 8 cores.
- The numerical ordering of the cores is different from the ordering of the attached GPUs.
- For best performance, ensure appropriate placement of processes and threads.



Checking GPU Availability

- The **`rocm-smi`** program can be used to check available GPUs for processing on a node.
- A subset of GPUs may be available with shared access depending on the resources requested.
- The hexadecimal Bus IDs are consistent across all GPU nodes.
- The GPU IDs may vary depending on the job requested:

```
$ rocm-smi --showhw

======================== ROCm System Management Interface ========================
============================ Concise Hardware Info ===============================
GPU  DID   GFX RAS   SDMA RAS  UMC RAS   VBIOS           BUS
0    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C1:00.0
1    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C6:00.0
2    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C9:00.0
3    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:CE:00.0
4    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D1:00.0
5    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D6:00.0
6    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D9:00.0
7    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:DE:00.0
==============================================================================
========================== End of ROCm SMI Log ===============================
```

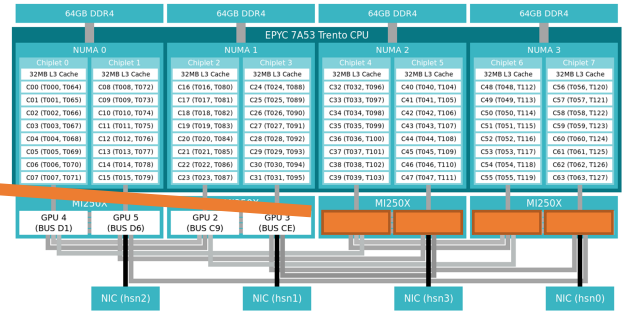
GPU ID(s)

GPU Bus ID(s)

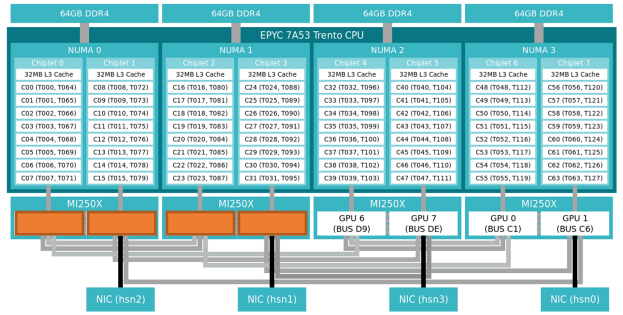
Checking GPU Availability

- **`--gpus-per-task`** and **`--gpus-per-node`** will restrict the available devices to the job.
- For example, the following is the rocm-smi output from two jobs sharing the same GPU node:

```
======================= ROCm System Management Interface =======================
============================ Concise Hardware Info ==============================
GPU  DID   GFX RAS   SDMA RAS  UMC RAS   VBIOS              BUS
0    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C1:00.0
1    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C6:00.0
2    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D9:00.0
3    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:DE:00.0
==============================================================================
============================ End of ROCm SMI Log ===============================
```

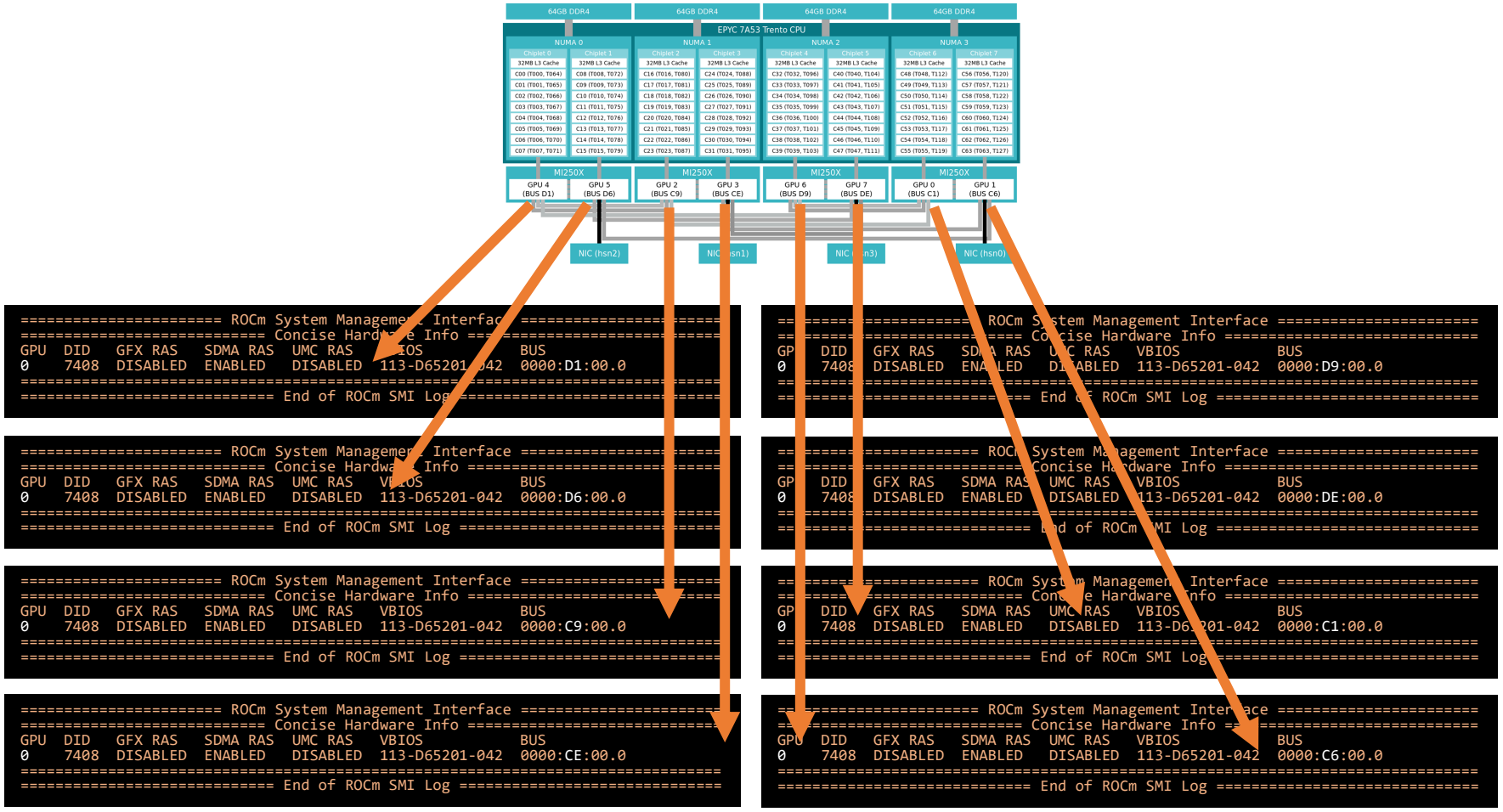



```
======================= ROCm System Management Interface =======================
============================ Concise Hardware Info ==============================
GPU  DID   GFX RAS   SDMA RAS  UMC RAS   VBIOS              BUS
0    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:C9:00.0
1    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:CE:00.0
2    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D1:00.0
3    7408  DISABLED  ENABLED   DISABLED  113-D65201-042  0000:D6:00.0
==============================================================================
============================ End of ROCm SMI Log ===============================
```



Checking GPU Availability

- **`--gpus-per-task`** and **`--gpus-per-node`** will restrict the available devices to the job.
- This example has exclusive access and **`--gpus-per-task=1`**:



Restricting GPU visibility with wrapper scripts

- The **`ROCR_VISIBLE_DEVICES`** environment variable restricts the GPUs available to the HIP runtime.
- A wrapper script can be used in Slurm batch scripts to limit GPU visibility on a per-process basis:

```
wrapper=selectGPU_${SLURM_JOBID}
cat << EOF > $wrapper
#!/bin/bash

export ROCR_VISIBLE_DEVICES=\$SLURM_LOCALID
exec \$*
EOF
chmod +x ./$wrapper
```

- The wrapper script can be provided in place of your program with the program as an argument when using **`srun`**.

Binding CPU processes

- The ordering of the CPU processes can be controlled with the **`--cpubind`** option for srun.
- The **`generate_CPU_BIND.sh`** script is available to generate the bindings:

```
CPU_BIND=$(generate_CPU_BIND.sh map_cpu)
```

- These bindings can then be provided to **`srun`**:

```
srun -c 8 --cpu-bind=${CPU_BIND} ./$wrapper $theProgram
```

- Refer to the documentation for a step-by-step guide:
  - Example Slurm Batch Scripts for Setonix on GPU Compute Nodes
- Try with and without binding to compare the performance.
- Attend an upcoming migration Ask. Me. Anything. (AMA) session (Events page) or submit a Pawsey Help Desk ticket for assistance.

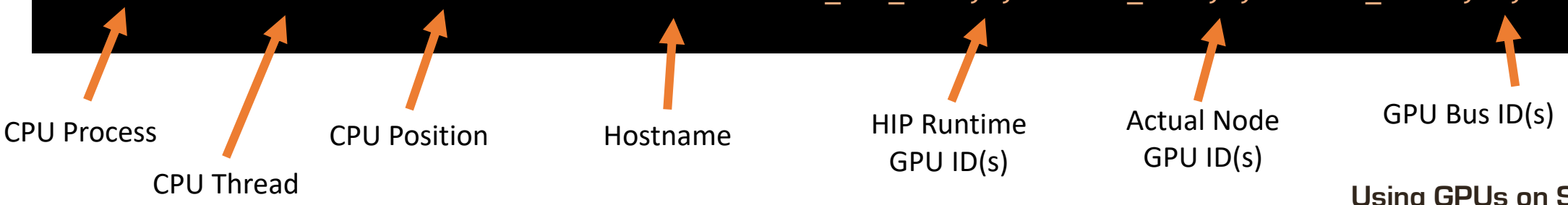
Checking Process and Thread Placement

- Use a parallel hello world program to check which core and GPU(s) are being used.
- The **`hello_jobstep`** program is available courtesy of ORNL:

```
$ git clone https://github.com/PawseySC/hello_jobstep.git
$ cd hello_jobstep
$ module load PrgEnv-cray craype-accel-amd-gfx90a rocm
$ make
```

- Run the program in place of your usual code to understand the placement:

```
$ export OMP_NUM_THREADS=1; srun -N 1 -n 3 -c 8 ./hello_jobstep | sort -n
MPI 000 - OMP 000 - HWT 002 - Node nid002920 - RT_GPU_ID 0,1,2 - GPU_ID 0,1,2 - Bus_ID c9,d1,d6
MPI 001 - OMP 000 - HWT 009 - Node nid002920 - RT_GPU_ID 0,1,2 - GPU_ID 0,1,2 - Bus_ID c9,d1,d6
MPI 002 - OMP 000 - HWT 017 - Node nid002920 - RT_GPU_ID 0,1,2 - GPU_ID 0,1,2 - Bus_ID c9,d1,d6
```



How do I get help?

Migration Documentation & Migration Guides

- Setonix GPU Partition Quick Start
- Setonix Migration Guide
- Setonix User Guide
- Supercomputing Documentation

Migration Training Materials & Video Recordings

- Upcoming Migration Training
- Recordings: Pawsey YouTube Setonix Migration Phase 2 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/PLLNSQT