

# Using a cluster effectively

## Scheduling and Job Management

- Log into cedar.computecanada.ca:
  - ssh -X [yourusername@cedar.computecanada.ca](mailto:yourusername@cedar.computecanada.ca)
  - use putty if you are working in windows
- Copy the working directory to your own and go into it.
  - cp -r /home/kamil/workshop\_public/2017/scheduling .
  - cd scheduling
- You can find a copy of the slides and materials for this workshop in the following link  
<https://goo.gl/eXyHhL>

# Upcoming ARC Training Sessions

<b>October 25</b> 10am - 11pm MDT	<b>Machine Learning Using Jupyter Notebooks on Graham</b>
<b>November 1</b> 11am – 1 pm MDT	<b>Introduction to Classical Molecular Dynamics Simulations</b>
<b>November 21</b> 11am – 1 pm MDT	<b>Exploring Containerization with Singularity</b>
<a href="https://www.westgrid.ca/events/westgrid-training-events">https://www.westgrid.ca/events/westgrid-training-events</a>	



# Scheduling and Job Management 1

Using a cluster effectively



# Presentation contents

Scheduling Theory

Basic Job submission

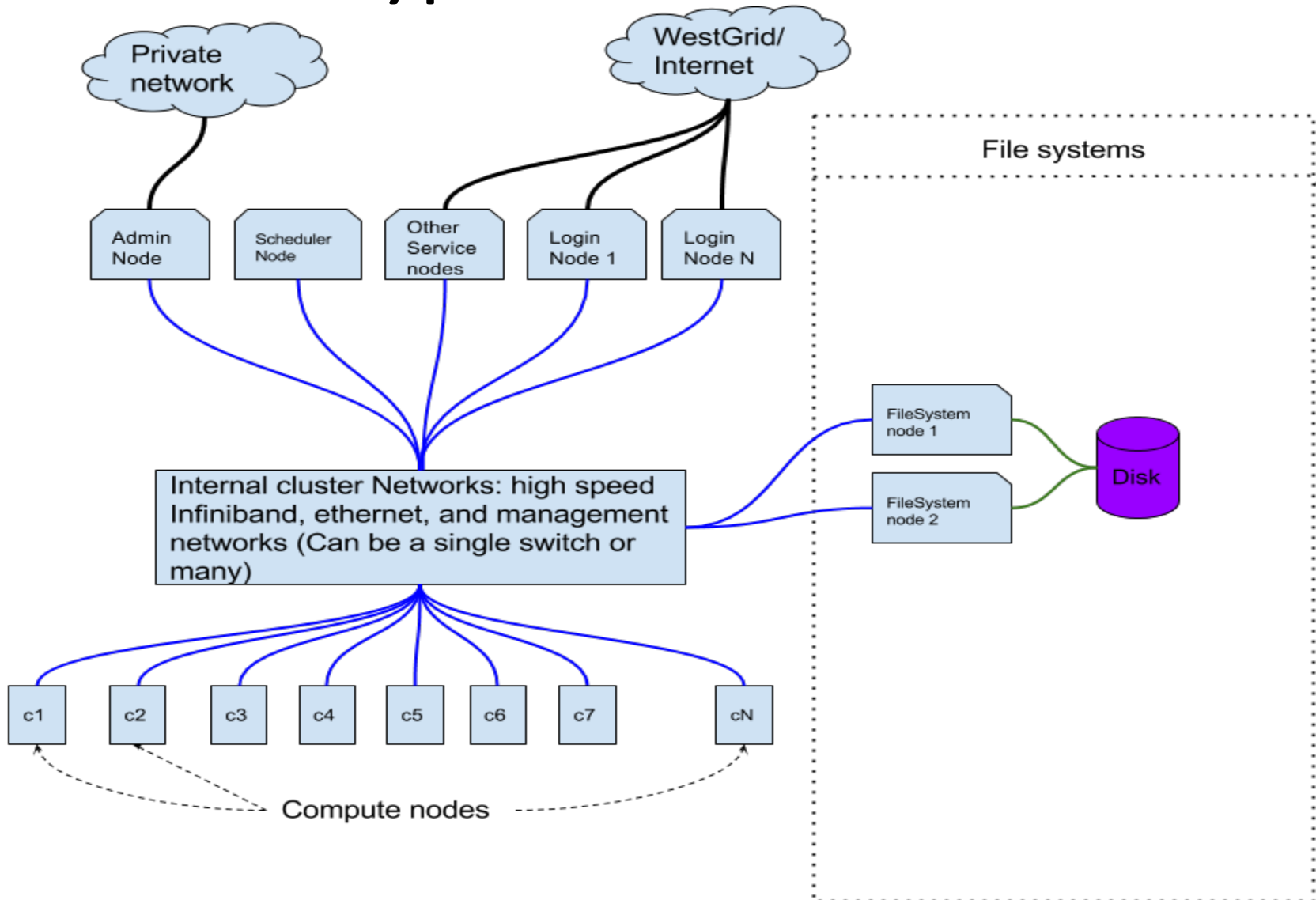
Parallel computing and Job submission



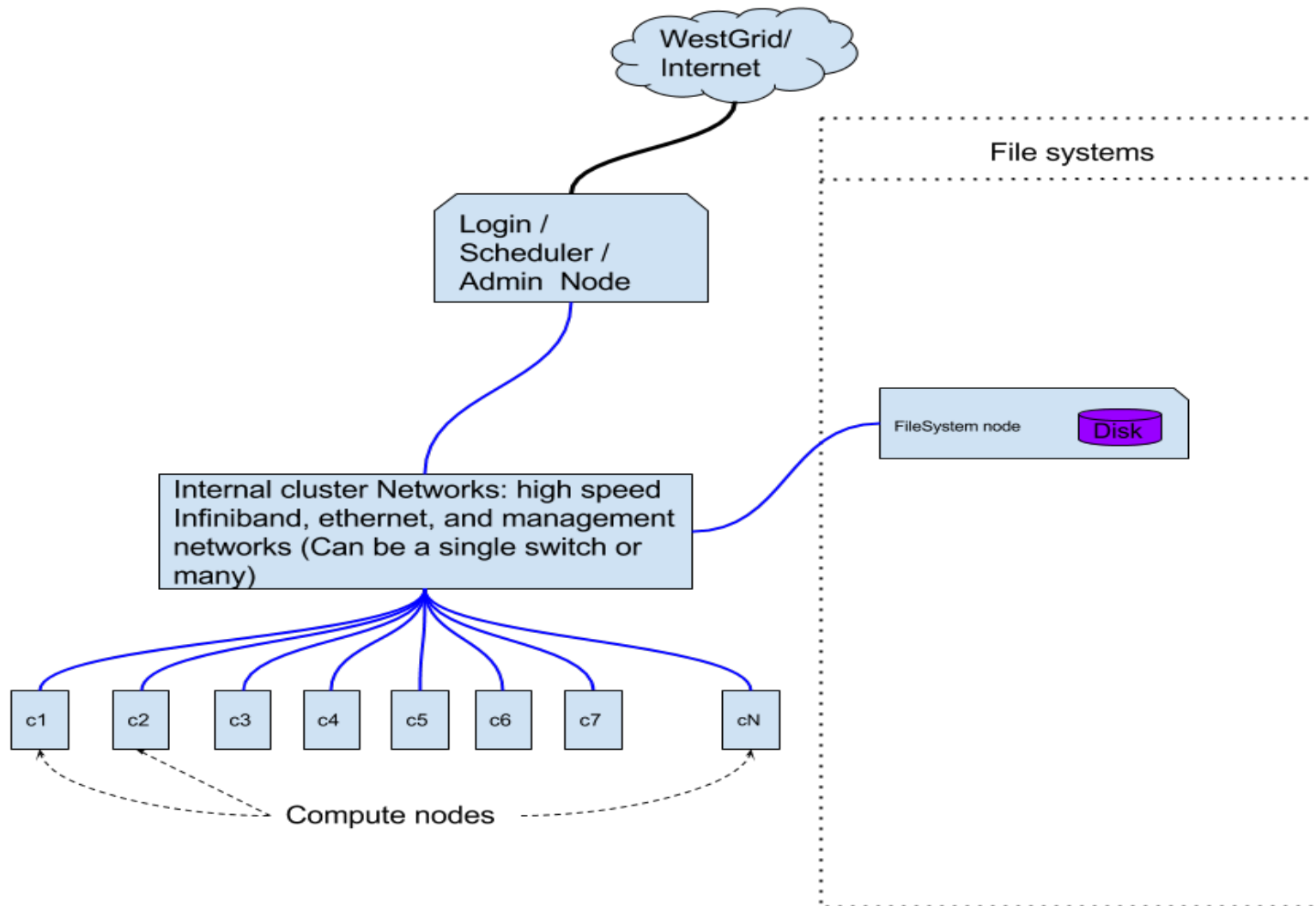
# Batch Scheduling

- Is not used when you need a service for example a webserver that runs all the time.
- Is preferred when you have one or more jobs (simulations) that need to be run and you wish to get the results back sometime in the future.
- Your job automatically started by the scheduler when enough resources are available, and you get results back, you may be notified when your job starts and finishes.

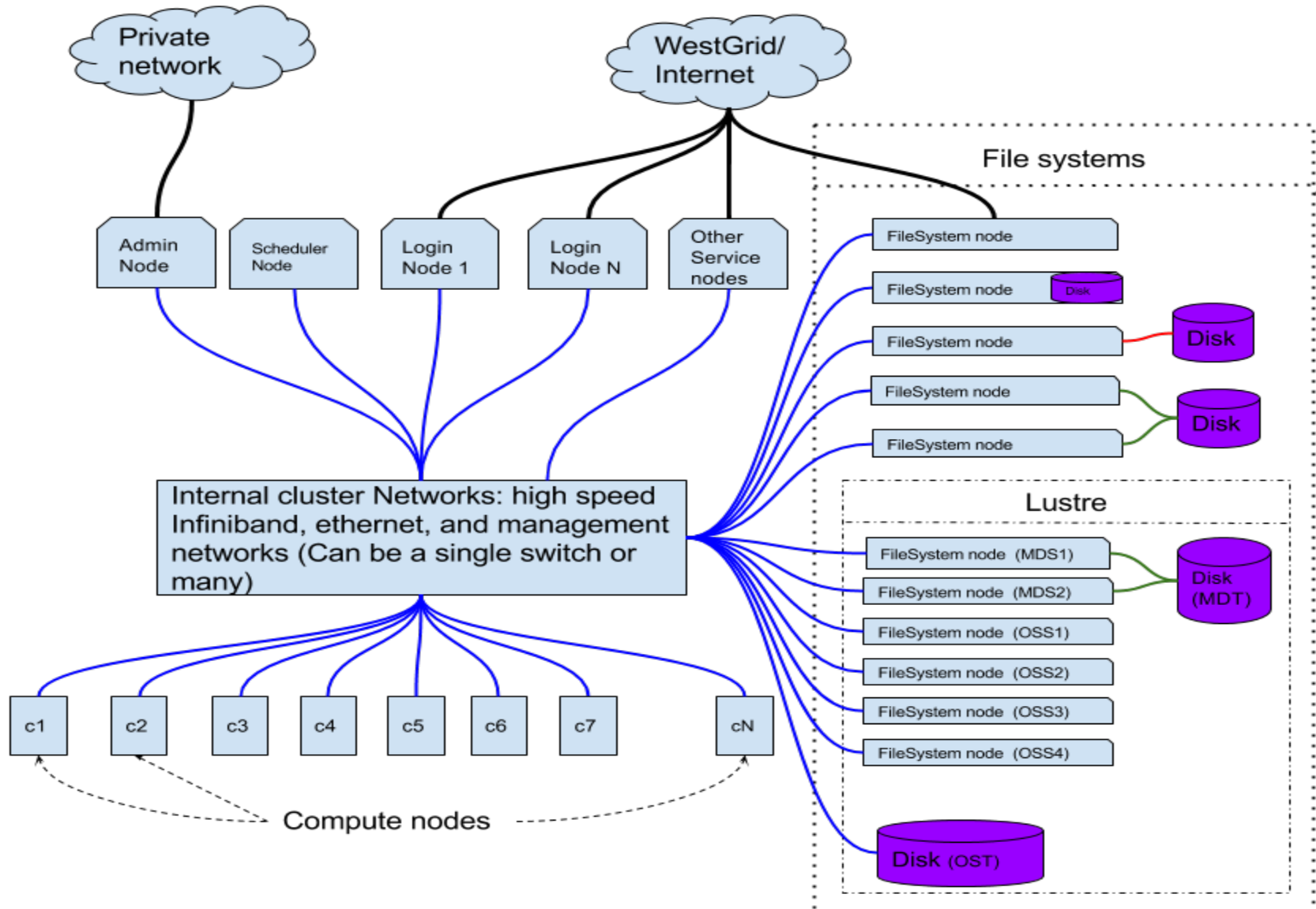
# Typical HPC Cluster



# Typical small HPC Cluster



# Bigger HPC Cluster



# Goals of scheduling

- Fairness and policy
- Efficiency / Utilization / Throughput
- Minimize turnaround

# Fairness and policy

- Does not necessarily mean everyone or every group gets the same usage.
- An important science project may get larger allocation.
- Scheduler fairly allocates according to usage policy

# Efficiency, Utilization and Throughput

- We want all resources cpus, gpus, memory, disk, software licenses, bandwidth, and more to be all used as much as possible.
- How many gaps are there in scheduling between jobs.

# Minimize turnaround

- Goal here is return an answer or result to a user as fast as possible
- Important to users which use iterative process to their goal.
- Minimize time to scientific discovery



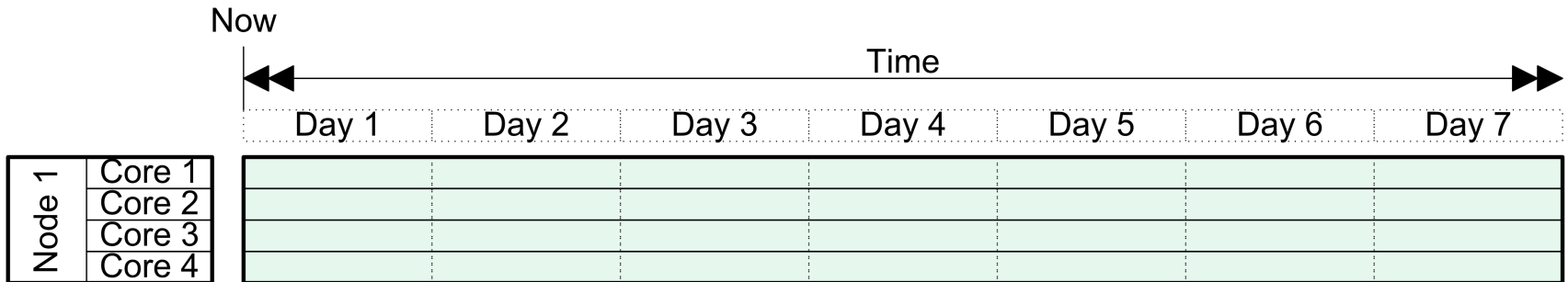
# Some insights

- The shorter the walltime which is the maximum time a job will run before being killed, the better we can meet the 3 goals of scheduling.
- Jobs using large amount of resources per job result in a reduction of fairness, efficiency, responsiveness of the scheduling system.
- The more nodes we have the better we can meet these goals.

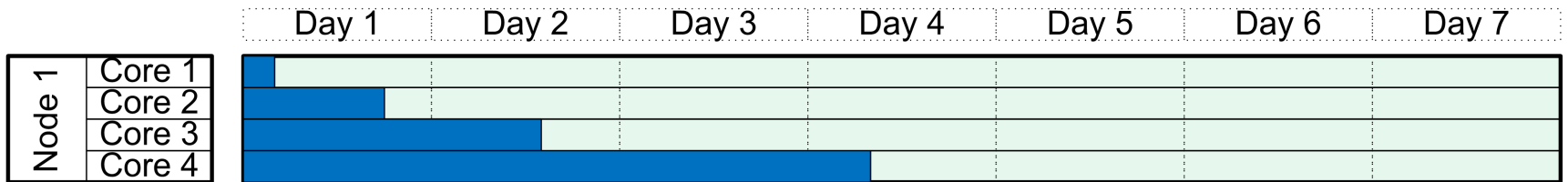
# Advantages of Large Clusters

- Larger clusters are more fair, efficient, responsive just by being larger.
- Larger clusters are capable of running larger jobs expanding capability, but if larger jobs are run exclusively we lose the advantage of a large cluster.
- Shared resources such as WestGrid are better and are used more efficiently than multiple small clusters. The larger the scope of shared resources the better.

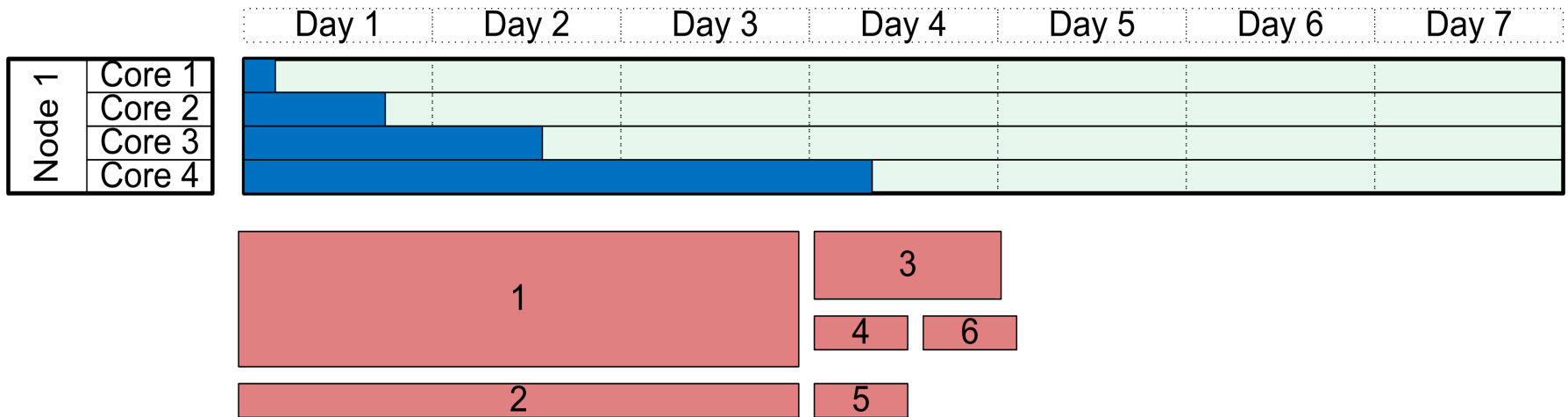
# Visualizing single node cluster



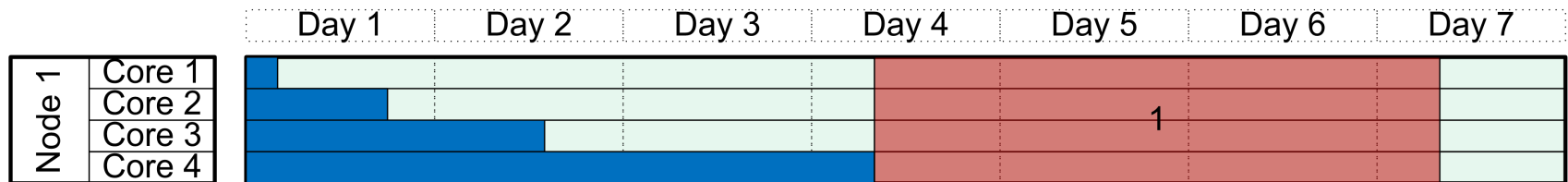
# Running jobs



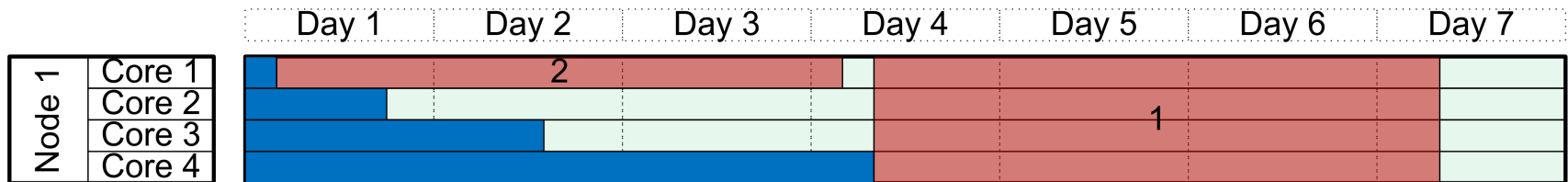
# Scheduling jobs in order of priority



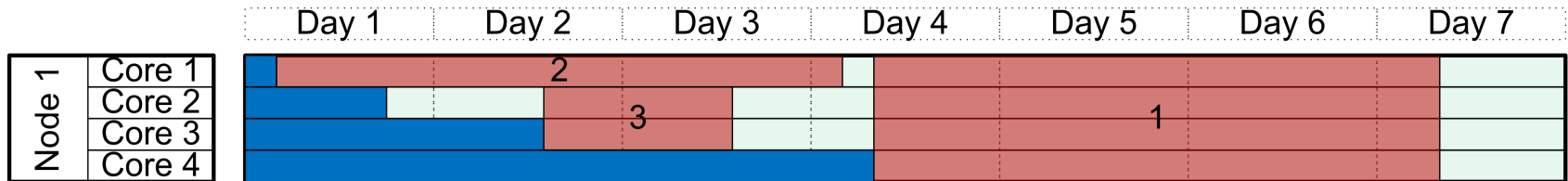
# Scheduling jobs in order of priority



# Scheduling jobs in order of priority

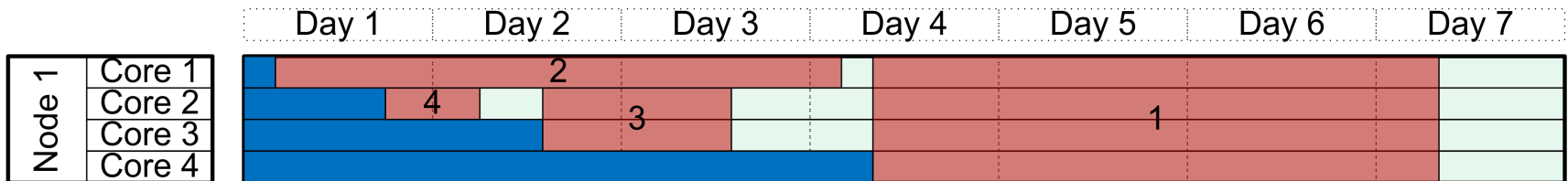


# Scheduling jobs in order of priority

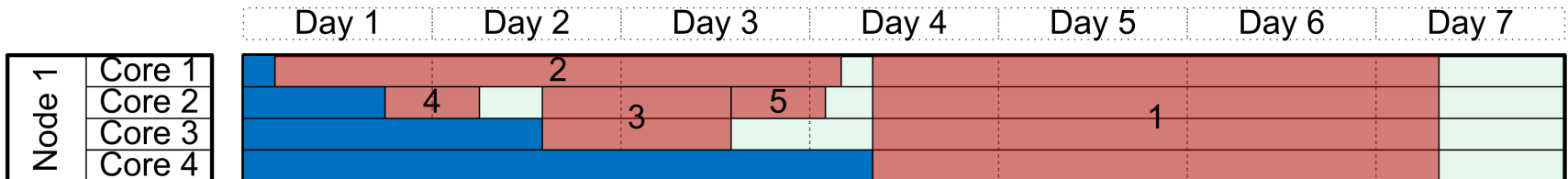




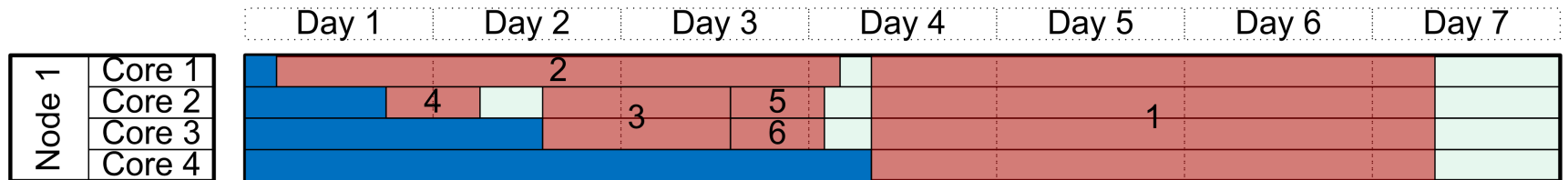
# Scheduling jobs in order of priority



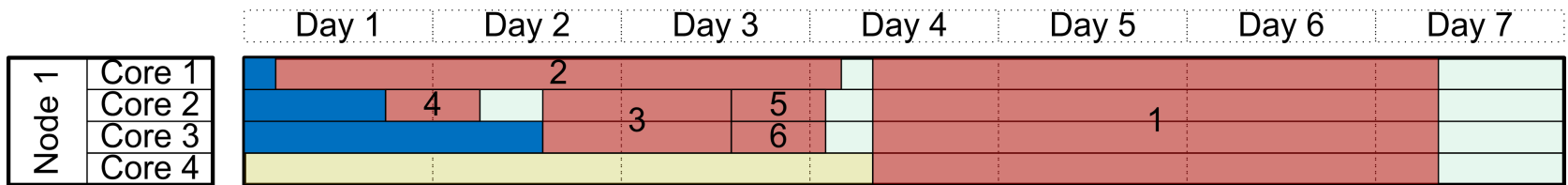
# Scheduling jobs in order of priority



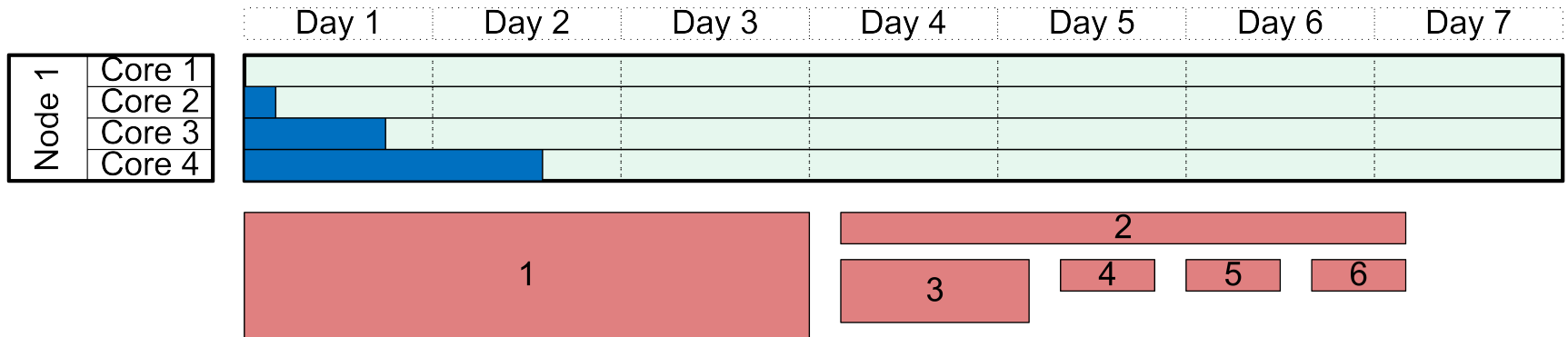
# Scheduling jobs in order of priority



# A Job finishes early



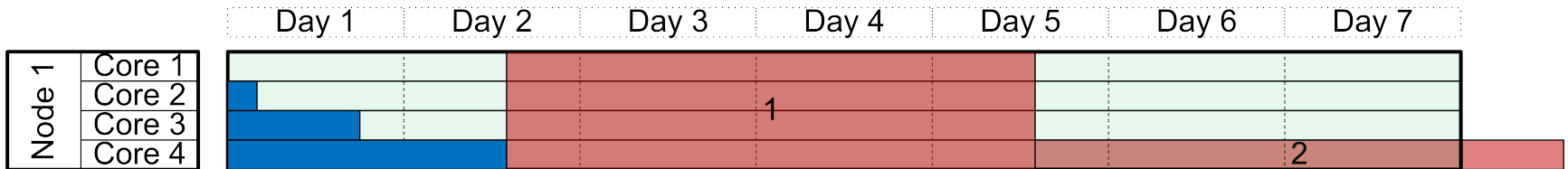
# Jobs are rescheduled



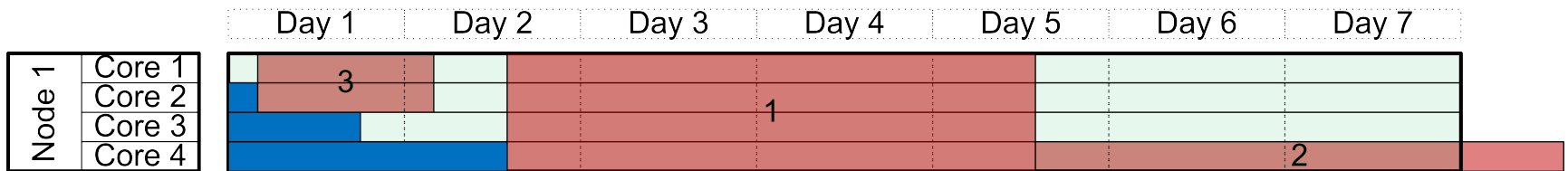
# Jobs are rescheduled



# Jobs are rescheduled

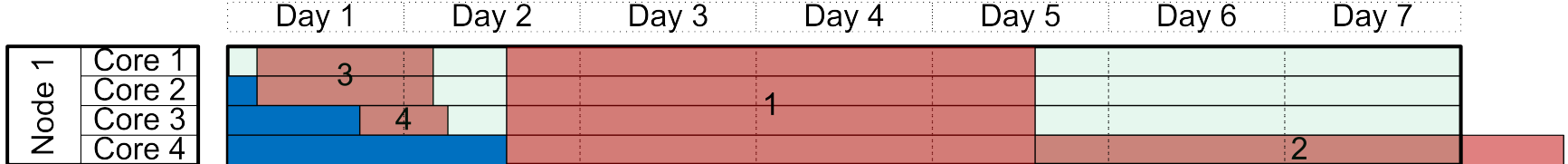


# Jobs are rescheduled

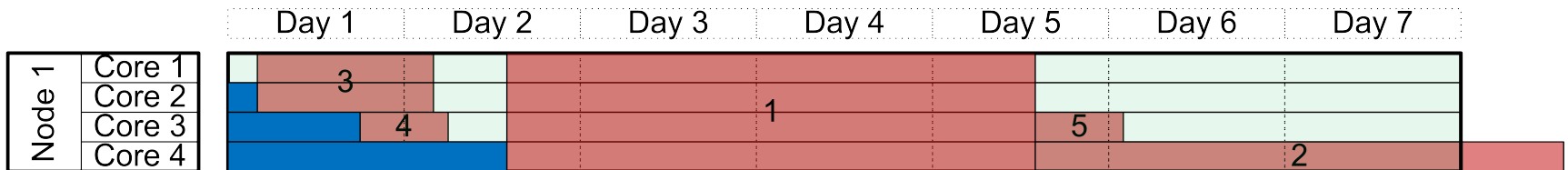




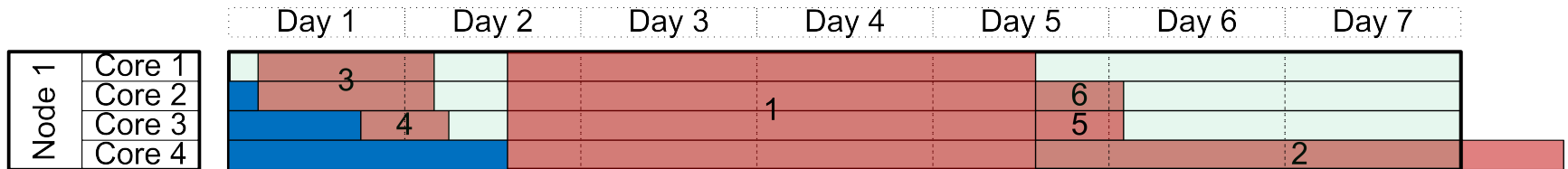
# Jobs are rescheduled



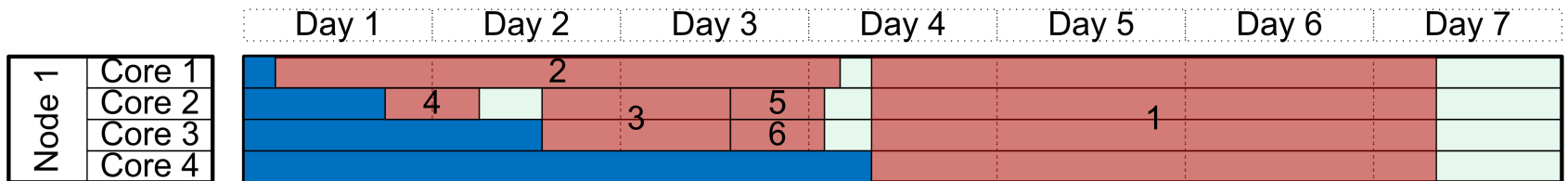
# Jobs are rescheduled



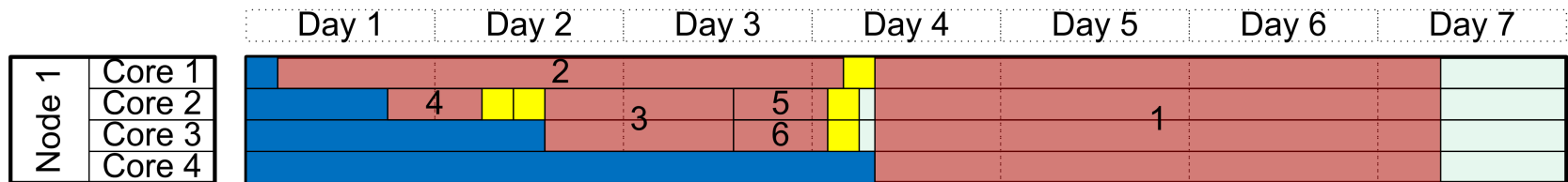
# Jobs are rescheduled



# Single node cluster



# Short serial jobs and Backfill



# Myths

If there is a large number of jobs in the queue my job will not run quickly.

- Most of the time these jobs belong to users with very low priority, because they are running a large number of jobs.
- Most of these jobs may not be capable of running as number of running jobs per user may be limited.
- The cluster may have empty processors available for immediate use.
- Deciding if a cluster is busy by number of queued jobs does not work.

It is better not to submit too many jobs at a time so that other users can run.

- The scheduling system is more efficient if you submit your jobs earlier, as long as you don't go over the usage limits.
- Fairness is insured by the scheduling system.

# Tips

- Make sure your job can run on the resources available on the cluster.
- Look at the state of cluster/account/Jobs and how to get the information.
- If the cluster is empty and you are able to run shorter jobs to evade the limits.

# Basic Job submission



# Submitting a Job

- If you have a program that you wish to run you need to figure out the resource requirements of your Job. These requirements include:
  - walltime: maximum length of time your will take to run
  - number of cpus, memory, nodes, gpus
  - The partition you are submitting to.
- The command to submit your job is sbatch, although sbatch allows you to specify your requirements on the command line, however you should put your requirements in a job script.
- `sbatch jobscript.sh`

# Simple slurm job script

```
#!/bin/bash
```

```
#SBATCH --ntasks=1
```

```
#SBATCH --nodes=1
```

```
#SBATCH --time=0-00:02
```

```
#SBATCH --mail-type=ALL
```

```
#SBATCH --mail-user=no.email@ubc.ca
```

```
#SBATCH -o my-output-file-%j.out
```

```
#SBATCH --job-name=my-named-job
```

```
sleep 1000; # Replace with a line running code
```

# Basic Slurm script commands

Slurm script command	Description
<code>#!/bin/bash</code>	Sets the shell that the job will be executed on the compute node
<code>#SBATCH --ntasks=1</code> <code>#SBATCH --n1</code>	Requests for 1 processors on task, usually 1 cpu as 1 cpu per task is default.
<code>#SBATCH --time=0-05:00</code> <code>#SBATCH -t 0-05:00</code>	Sets the maximum runtime of 5 hours for your job
<code>#SBATCH --mail-user= &lt;email&gt;</code>	Sets the email address for sending notifications about your job state.
<code>#SBATCH --mail-type=BEGIN</code> <code>#SBATCH --mail-type=END</code> <code>#SBATCH --mail-type=FAIL</code> <code>#SBATCH --mail-type=REQUEUE</code> <code>#SBATCH --mail-type=ALL</code>	Sets the scheduling system to send you email when the job enters the following states: BEGIN,END,FAIL,REQUEUE,ALL
<code>#SBATCH --job-name=my-named-job</code>	Sets the Jobs name

# Slurm Jobs and steps

- Unlike PBS slurm jobs can have multiple steps
- Each of these steps is like a job and may have different resources used in it.
- Use the command `srun` to carry out each step
- `Srun` has a similar syntax to `sbatch`
- You can have prologue and epilogue per step.

# Interactive Jobs

- One can ask for an interactive Job to run a program on the cluster and interact with it while it is running.
- Interactive jobs are useful for debugging.
- To request and use an interactive job is a multi step process.
- We can request an allocation of resources with the salloc command
  - `salloc --ntasks=1 --nodes=1 --time=0-01:20`
- We actually proceed to open a shell on the inside of the allocated job
  - `srun --pty -p interact bash`
- Please make sure to only run the job on the processors assigned/ allocated to your job. This will happen automatically if you use srun, but not if you just ssh from the headnode.

# SLURM Environment Variables

Environment Variable	Description
SLURM_JOB_NAME	User specified job name
SLURM_JOB_ID	Unique slurm job id
SLURM_NNODES	Number of nodes allocated to the job
SLURM_NTASKS	Number of tasks allocated to the job
SLURM_ARRAY_TASK_ID	Array index for this job
SLURM_ARRAY_TASK_MAX	Total number of array indexes for this job
SLURM_MEM_PER_CPU	Memory allocated per CPU
SLURM_JOB_NODELIST	List of nodes on which resources are allocated to Job
SLURM_JOB_CPUS_PER_NODE	Number of CPUs allocated per Node
SLURM_JOB_PARTITION	List of Partition(s) that the job is in.
SLURM_JOB_ACCOUNT	Account under which this job is run.

Running basic Jobs

**BREAK FOR PRACTICE**

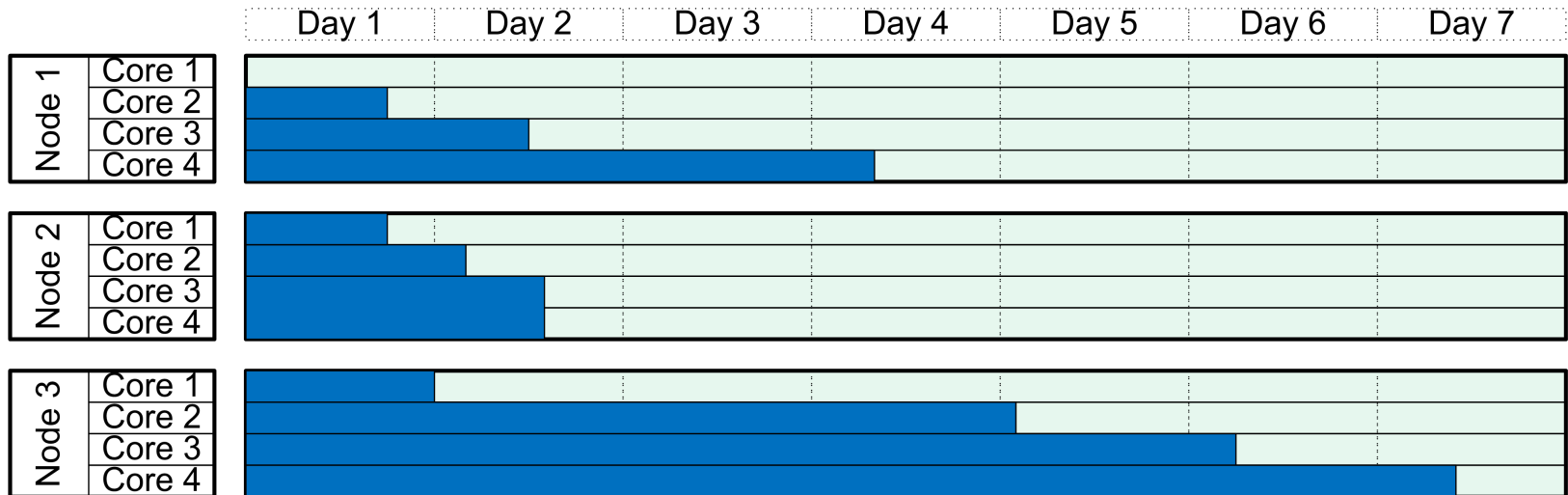
# Jobs Types: Parallelism

- Many Serial Jobs
- Message Passing (MPI)
- Single node multi-core (OpenMP, Gaussian)
- Hybrid/ Advanced

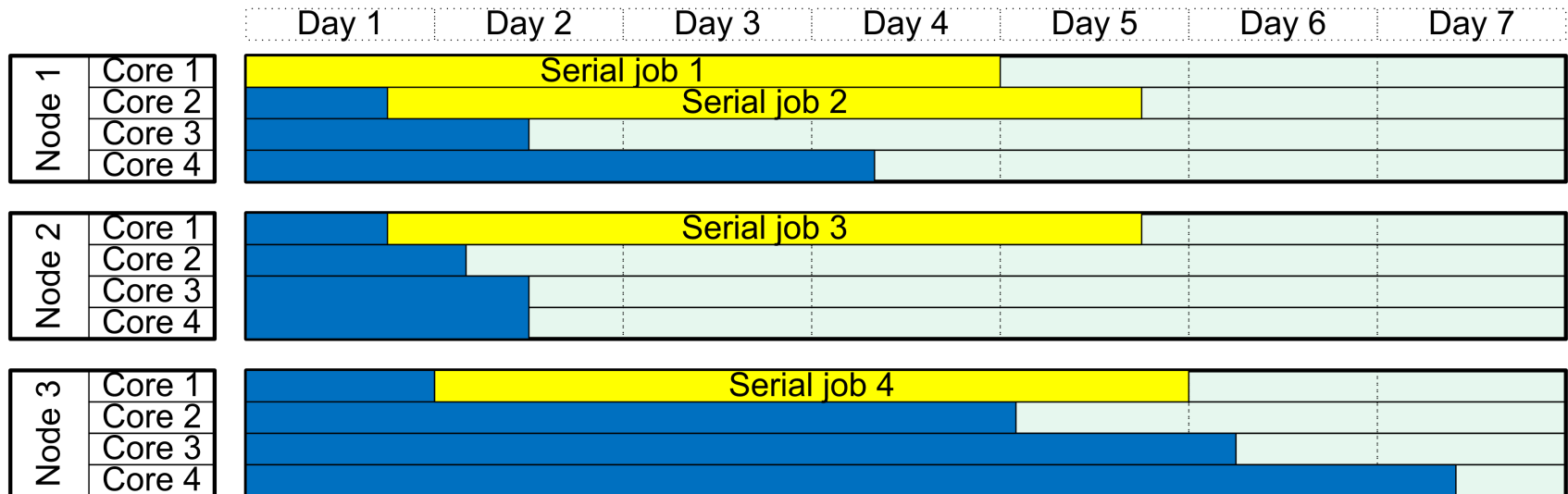
	1 Nodes	N Nodes
1 cpu	Serial	MPI
X cpus	OpenMP	Hybrid



# Visualizing Multinode cluster



# Many Serial Jobs



# Many Serial Jobs

- Use 1 cpu per job
- Easiest and most efficient to schedule
- Excellent scaling linear speedup
- Example job would be a parameter searches
- In your slurm file one can ask for a serial job with:
- `#SBATCH --ntasks=1`

# Slurm Serial Job Example

```
#!/bin/bash
```

```
#SBATCH --ntasks=1
```

```
#SBATCH --time=0-00:02
```

```
#SBATCH --mail-type=ALL
```

```
#SBATCH --mail-user=no.email@ubc.ca
```

```
#SBATCH -o my-output-file-%j.out
```

```
#SBATCH --job-name=my-named-job
```

```
sleep 1000; # Replace with a line running code
```

# Tips for running more Serial Jobs

- Submit shorter serial jobs
- Many short serial jobs will run before larger job
- Checkpoint longer jobs and submit them as short jobs, this will also save you when the cluster suffers hardware or power failure.

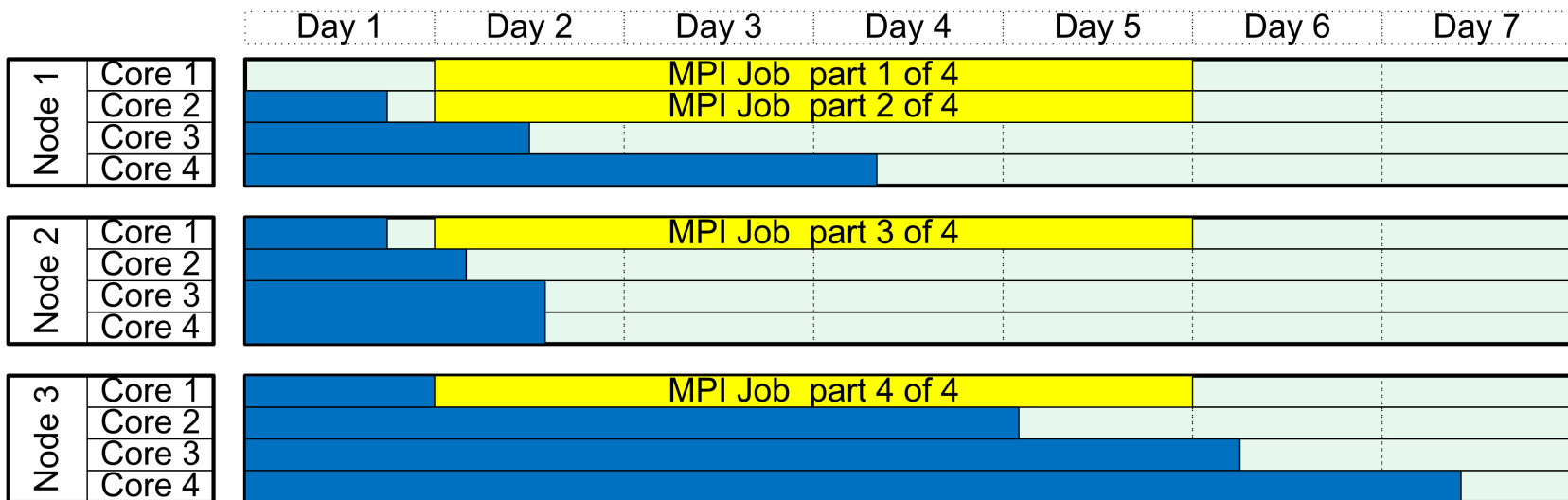
# Job array

- Job arrays are used when you have need to submit a large number of Jobs using the same job script.
- There is a naming convention for jobs in array, which is useful as you don't need to remember a large number of unique job ids or job names: jobname[0]
- Job arrays are preferred as they don't require as much computation by the scheduling system to schedule, as they are evaluated as a group instead of individually. Ask for a job array in one of the following ways:
  - #SBATCH --array=0-99
    - job array 100 jobs numbered 0 -99
  - #SBATCH --array=1,2,3,5,7
    - Job array with 5 jobs with indexes [1,2,3,5,7]
  - #SBATCH --array=0-99%5
    - job array 100 jobs numbered 0 -99 with a maximum of 5 running at any time

# Job array sample script

```
#!/bin/bash
#SBATCH --ntasks=1                # Number of cores/tasks
#SBATCH --time=0-00:02            # Runtime in D-HH:MM
#SBATCH --job-name=my-array-job   # Sets the Jobs name
#SBATCH --array=1-12             # Ask for an Job array of
12 tasks
echo "This jobs name is:  $SLURM_JOB_NAME"
echo "This jobs jobid is:  $SLURM_JOB_ID"
echo "This jobs taskid is: $SLURM_ARRAY_TASK_ID"
sleep 30
hostname
```

# MPI job





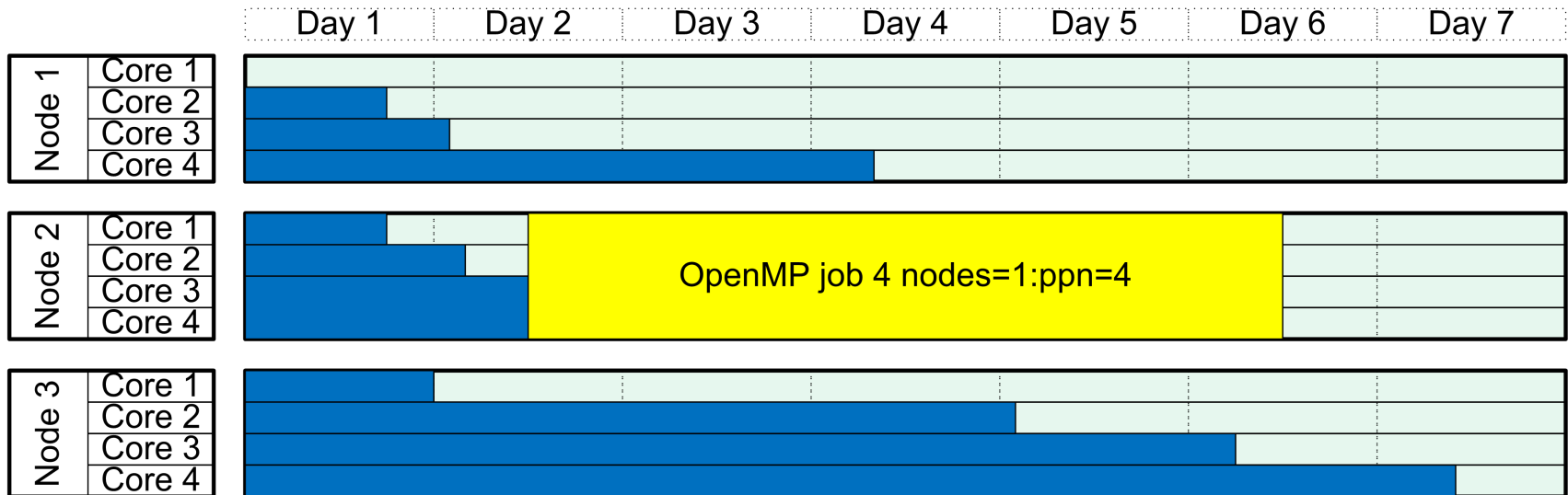
# MPI Jobs

- Use the network for message passing
- Each job uses multiple cpus each of which can be on a different node.
- Each process uses a different memory address space
- More difficult to write parallel code than OpenMP as deadlocks are more common.
- Can scale higher than OpenMP as clusters are typically larger than even large SMP machines

# MPI Job Submission

- This type of job can have its processes running on any node, multiple processes can run on a single node.
- `#SBATCH --ntasks=X`

# Single node multi-core job (OpenMP, Gaussian, Threads)



# Single node muti-core job

- All the threads must run on a single node.
- The threads share a single memory address space
- Can compile serial and parallel executables from the same source code
- OpenMP is one of the easiest methods of parallel programing, can be done incrementally.

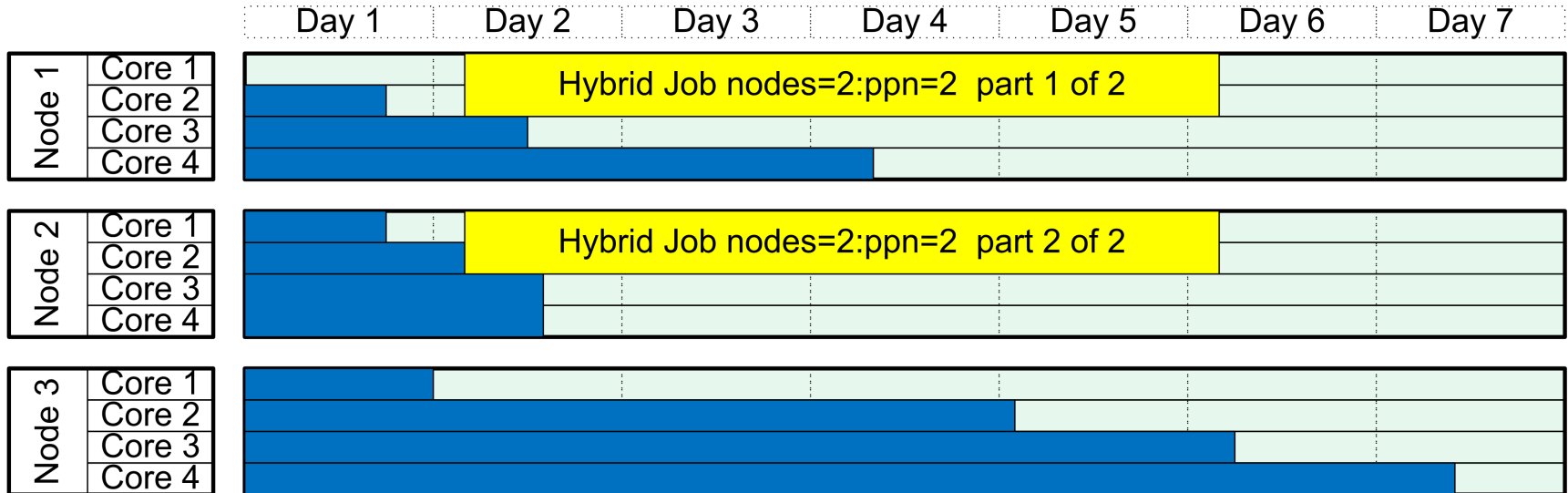
# OpenMP job submission

- This type of job must have its thread running on one node, sharing the same memory.
- Communication between parts of the job is done via memory
- `#SBATCH --cpus-per-task=X`
- One can ask the program to run a number of threads via an environment variable:
  - `export OMP_NUM_THREADS=8`
- Usually set it to the requested cores:
  - `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`

# Tips for running OpenMP Jobs

- Check the state of the cluster to see if your job will run quickly.
- If you have a number of OpenMP style jobs you should consider running longer jobs using less cpus per job instead.
  - It is faster and more efficient to schedule single/smaller processor jobs.
  - This advice may not apply when you need other resources like large amount of RAM per job.

# Hybrid Job



# Why use a hybrid job

- It's possible to combine OpenMP and MPI for running on clusters of SMP machines
- Need more memory or other resource than is available per core.
- Advanced systems of running parallel jobs can utilize resources more efficiently. Communication between cores is faster than between distant nodes. These systems include Chapel language as well as Partitioned global address space languages (PGAS) such as Unified Parallel C, Co-array Fortran.



# Slurm script commands

Slurm script command	Description
#SBATCH --ntasks=X	Requests for X tasks. When cpus-per-task=1 (and this is the default) this requests X cores. When not otherwise constraint these CPUs may be running on any node
#SBATCH --nodes=X	Request that a minimum of X nodes be allocated to this job
#SBATCH --nodes=X-Y	Request that a minimum of X nodes and a maximum of Y nodes be allocated to this job
#SBATCH --cpus-per-task=X	Request that a minimum of X CPUs per task be allocated to this job
#SBATCH --tasks-per-node=X	Requests minimum of X task be allocated per node

# Slurm script commands

Slurm script commands	Description of effects
<code>#SBATCH --ntasks=1</code> <code>#SBATCH --cpus-per-task=1</code>	Requests 1 CPU (Serial) <code>cpus-per-task</code> is set to 1 by default and may be omitted.
<code>#SBATCH --cpus-per-task=X</code> <code>#SBATCH --ntasks=1</code> <code>#SBATCH --nodes=1</code>	Requests for X CPUs in 1 task on 1 node (OpenMP) Both <code>ntasks</code> and <code>nodes</code> are set to 1 by default and may be omitted
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --tasks-per-node=X</code> <code>#SBATCH --cpus-per-task=1</code>	Requests for X CPUs and tasks on 1 node (OpenMP) <code>cpus-per-task</code> is set to 1 by default and may be omitted.
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --nodes=1</code> <code>#SBATCH --cpus-per-task=1</code>	Requests for X CPUs and tasks on 1 node (OpenMP) <code>cpus-per-task</code> is set to 1 by default and may be omitted.

# Slurm script commands

Slurm script commands	Description of effects
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --cpus-per-task=1</code>	Requests X CPUs and tasks (MPI) <code>cpus-per-task</code> is set to 1 by default and may be omitted.
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --ntasks-per-node=Y</code> <code>#SBATCH --cpus-per-task=1</code>	Requests for X CPUs and tasks with Y CPUs and tasks per node (MPI) <code>cpus-per-task</code> is set to 1 by default and may be omitted.
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --nodes=1</code> <code>#SBATCH --cpus-per-task=1</code>	Requests for X CPUs and tasks on the same node, <code>cpus-per-task</code> is set to 1 by default and may be omitted.
<code>#SBATCH --ntasks=X</code> <code>#SBATCH --nodes=1</code> <code>#SBATCH --cpus-per-task=1</code>	Requests for X CPUs and tasks on the 1 node <code>cpus-per-task</code> is set to 1 by default and may be omitted.

Serial, mpi, openmp, hybrid, jobarrays

**BREAK FOR PRACTICE**

**QUESTIONS?**

# Upcoming ARC Training Sessions

<b>October 25</b> 10am - 11pm MDT	<b>Machine Learning Using Jupyter Notebooks on Graham</b>
<b>November 1</b> 11am – 1 pm MDT	<b>Introduction to Classical Molecular Dynamics Simulations</b>
<b>November 21</b> 11am – 1 pm MDT	<b>Exploring Containerization with Singularity</b>
<a href="https://www.westgrid.ca/events/westgrid-training-events">https://www.westgrid.ca/events/westgrid-training-events</a>	

# Scheduling and Job Management 2

Using a cluster effectively

# Presentation contents

Job submission part 2

Understanding Jobs



# Slurm Jobs and memory

It is very important to specify memory correctly

- If you don't ask for enough and your job uses more ,your job will be killed.
- If you ask for too much, it will take a much longer time to schedule a job, and you will be wasting resources.
- If you ask for more memory than is available on the cluster your job will never run. The scheduling system will not stop you from submitting such a job or even warn you.
- If you don't know how much memory your jobs will need ask for a large amount in your first job and run:
  - `sstat --format=AveCPU,MaxRSS,MaxVMSize,JobID -j <jobid>`
- In the MaxRSS, you should see how much memory your job used.
- If you don't specify any memory then your job will get a very small default maximum memory.

# Slurm Jobs and memory

- Always ask for slightly less than total memory on node as some memory is used for OS, and your job will not start until enough memory is available.
- You may specify the maximum memory available to your job in one of 2 ways.
  - Ask for a total memory used by your jobs (MB)
    - `#SBATCH --mem=4000`
  - Ask for memory used per process/core in your job (MB)
    - `#SBATCH --mem-per-cpu=2000`

# Slurm jobs and GPUS

- To request GPU use the following syntax
  - **#SBATCH --gres=gpu:1**
- Modern slurm scheduling programs recognize GPUs as well as the state of the GPU.
- To request a large gpu node on cedar
  - **#SBATCH --gres=gpu:lgpu:4**

# Software licenses

- Sometimes not only cluster hardware is required to be scheduled for a job but other resources as well, such as software licenses, telescope or other instrument time.
- To request software licenses:
  - `#SBATCH --licenses=sas:2`

# Slurm script commands

PBS script command	Description
#SBATCH --mem=4000	Requests 4000 MB of memory in total
#SBATCH --mem-per-cpu=4000	Requests 4000 MB of memory per cpu
#SBATCH --licenses=sas:2	Requests 2 SAS licenses
#SBATCH --gres=gpu:1	Requests that your job get 1 GPU allocated per node
#SBATCH --exclusive	Requests that your job run only on nodes with no other running jobs
#SBATCH --dependency=after:job_id1	Requests that the the job start after job (jobid1) has <b>started</b>
#SBATCH --dependency=afterany:job_id1, job_i2	Requests that the the job start after ether job (jobid1) or job (jobud2) has <b>finished</b>
#SBATCH --dependency=afterok:job_id1	Requests that the the job start after job (jobid1) has <b>finished successfully</b>

Memory, Features, Software licenses , Partitions

**BREAK FOR PRACTICE**

# Job Submission Requiring Exclusive Access

- Sometimes there is a need for exclusive access to guarantee that no other job will be running on the same nodes as your job such as during debugging.
- To guarantee that the job will only run on nodes without other jobs you own use:
  - **#SBATCH --exclusive**
- Your research group may get charged for using the whole node and not just the resources requested, and it may take a long time to gather resources needed for these special jobs.

# Job submission multiple projects

- If you are part of two different Compute Canada projects and are running jobs for both, you need to specify the accounting group for each project so that the correct priority of the job can be determined and so that the usage is “charged” to the correct group.
- In order to specify an accounting group for a Job use:
  - **#SBATCH --account=accounting\_group**
- You can see your accounting group information with the “sacctmgr show user <username> withassoc” command.



# Job dependencies

- If you want one job to start one after another finishes use the
  - **#SBATCH --dependency=afterok:job\_id1**
- If one can break apart a long job into several shorter jobs then the shorter jobs will often be able to be ran faster. This is also the technique to use if the required job runtime is longer than the maximum walltime allowed on the cluster.
  - **job1id=\$(sbatch anwser-q24.1.sh | awk '{print \$4}')**
  - **sbatch --dependency=aftercorr:\$job1id anwser-q24.2.sh**

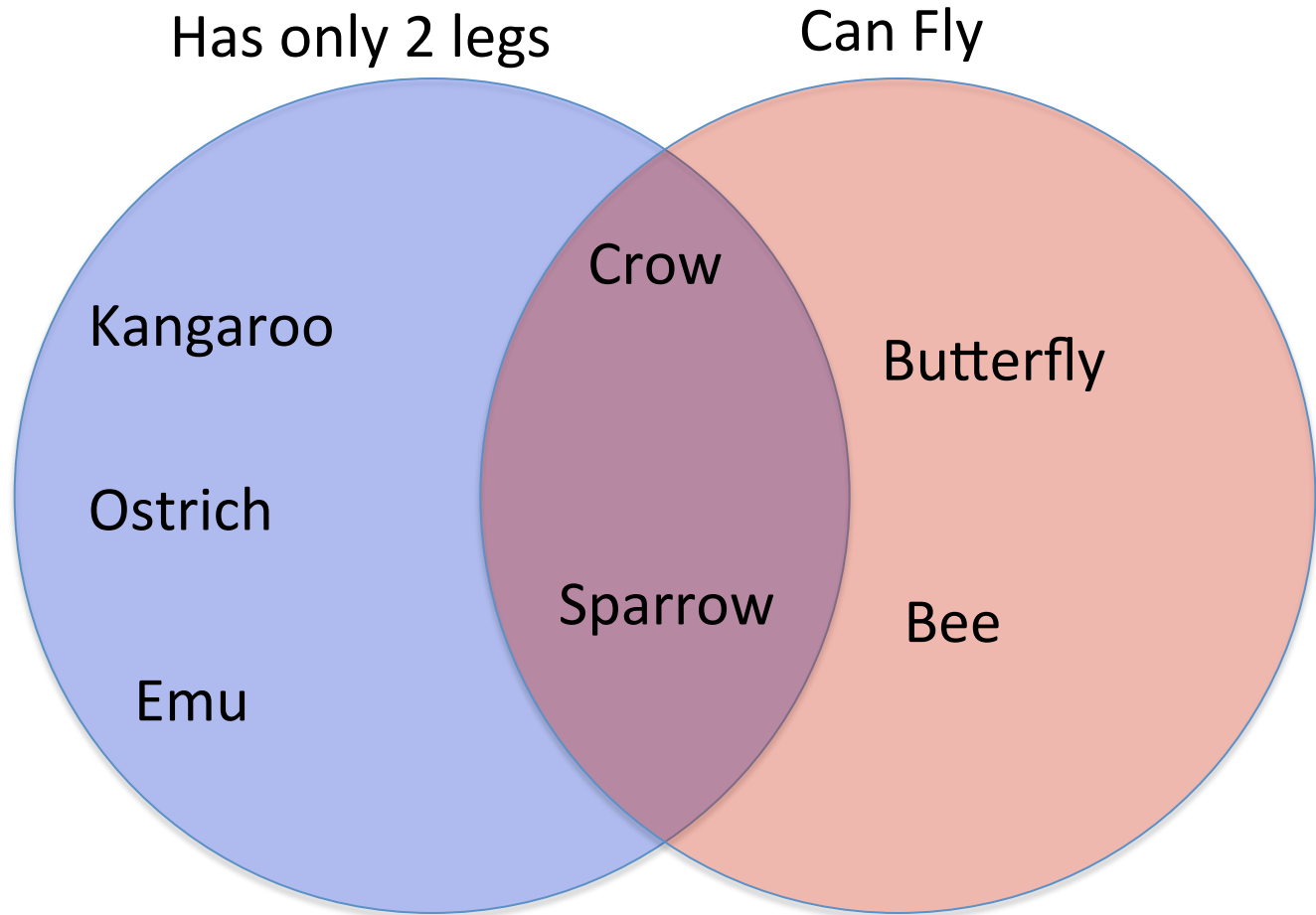
# Temporary available local storage

- Some software like Gaussian needs to make many small reads and writes to disk. The cluster (lustre) file system cannot do this well and this becomes a performance problem for the job and the cluster its running on.
- Each node has local disk, that is shared by all jobs running on the node. One specifies the requests the local storage via “#PBS -l file=1000mb”.
- There is a directory created for each job when it is run. When the job finished this directory is automatically erased. The directory name is \$TMPDIR. A example of using the temporary local storage:
  - #SBATCH --tmp=200G  
cd \$SLURM\_TMPDIR  
<run my job >  
mkdir my\_new\_dir  
cp <file I wish to save> my\_new\_dir/

# Partitions

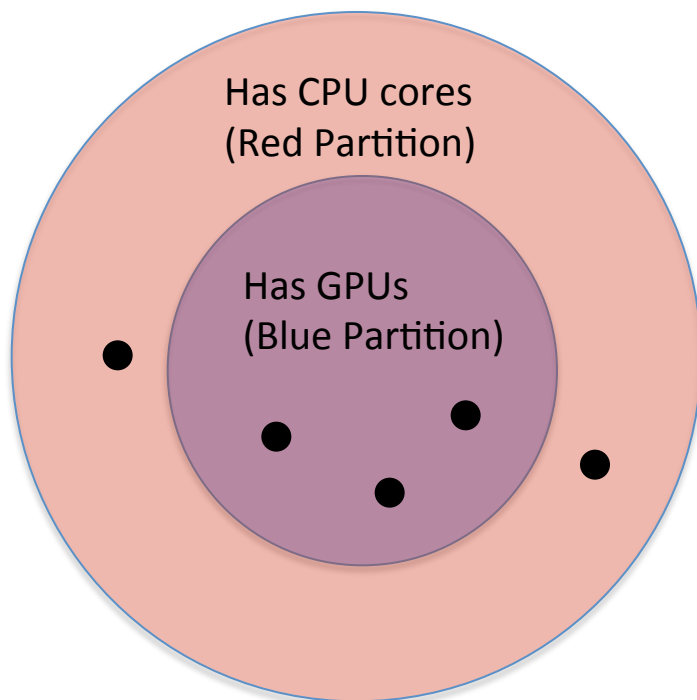
- Your job will automatically be assigned
- Somewhat like queues or classes in pbs/torque and moab.
- A job can be in multiple partitions simultaneously, and can have multiple a per partition priorities.
- A node can be in multiple partitions simultaneously

# Venn Diagram



# Partition Venn Diagram

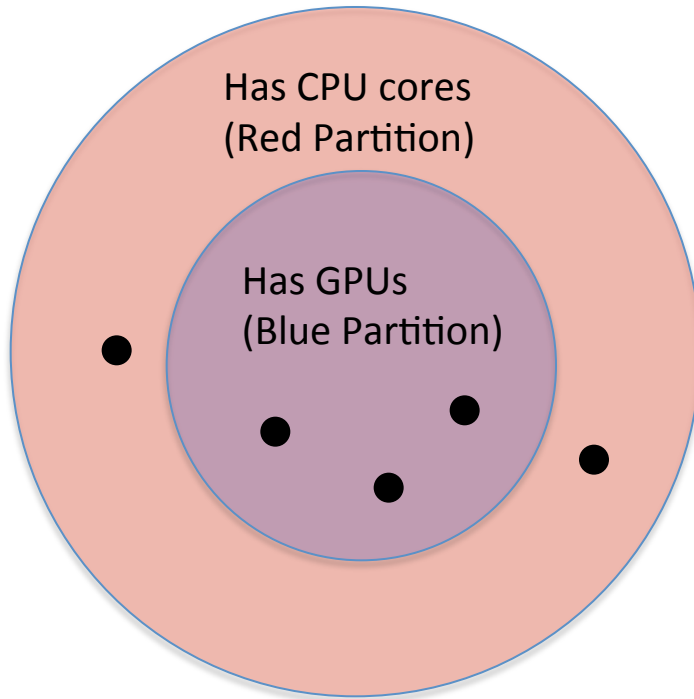
(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.

# Partition Venn Diagram

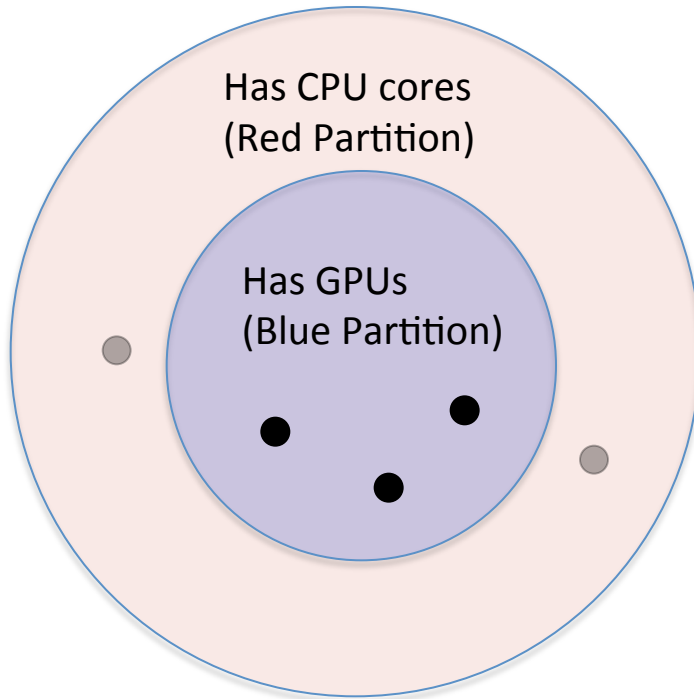
(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.

# Partition Venn Diagram

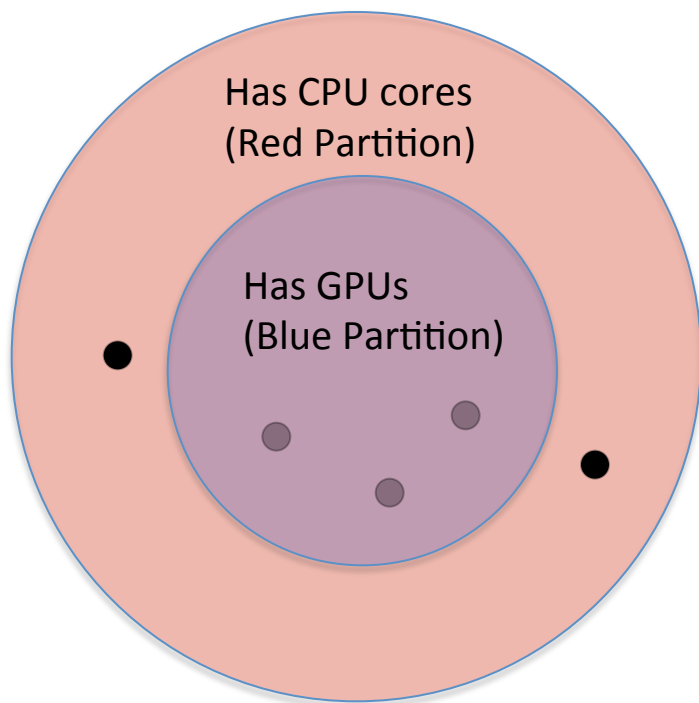
(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.

# Partition Venn Diagram

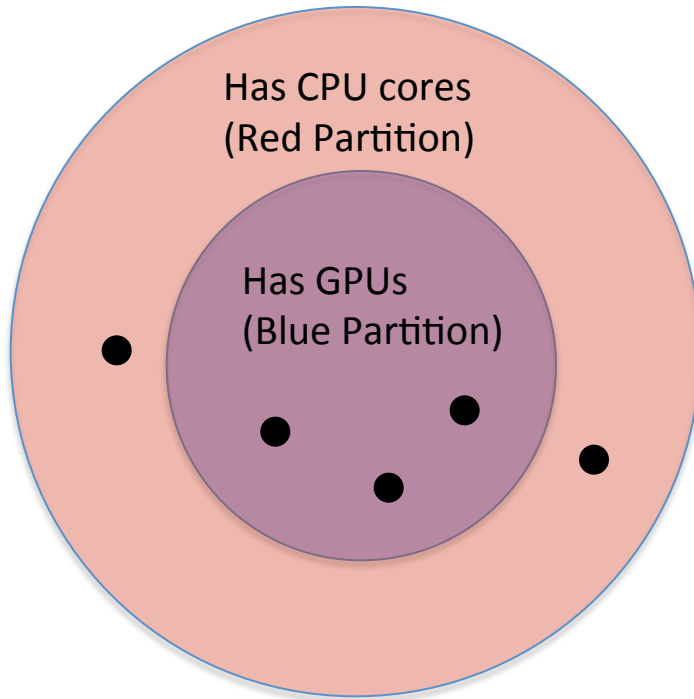
(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs (In the red partition but not in the blue)
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.



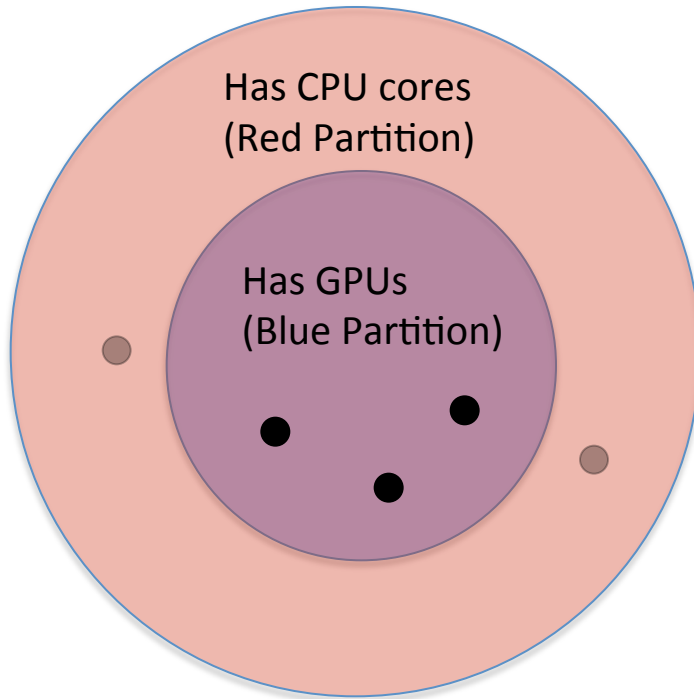
# Partition Venn Diagram



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.

# Partition Venn Diagram

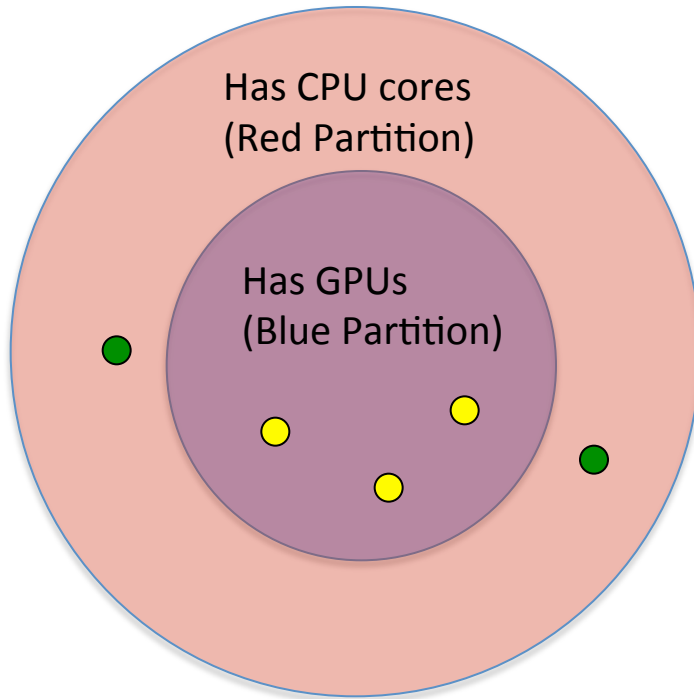
(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
  - The two nodes with no gpu in the red partition may be idle but a job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority blue jobs.

# Partition Venn Diagram

(on a 5 node imaginary cluster)



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
- In the case that the two nodes with no gpus in the red partition may be idle(green) and 3 nodes with gpus may be busy.
  - A job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority jobs in the blue partition.

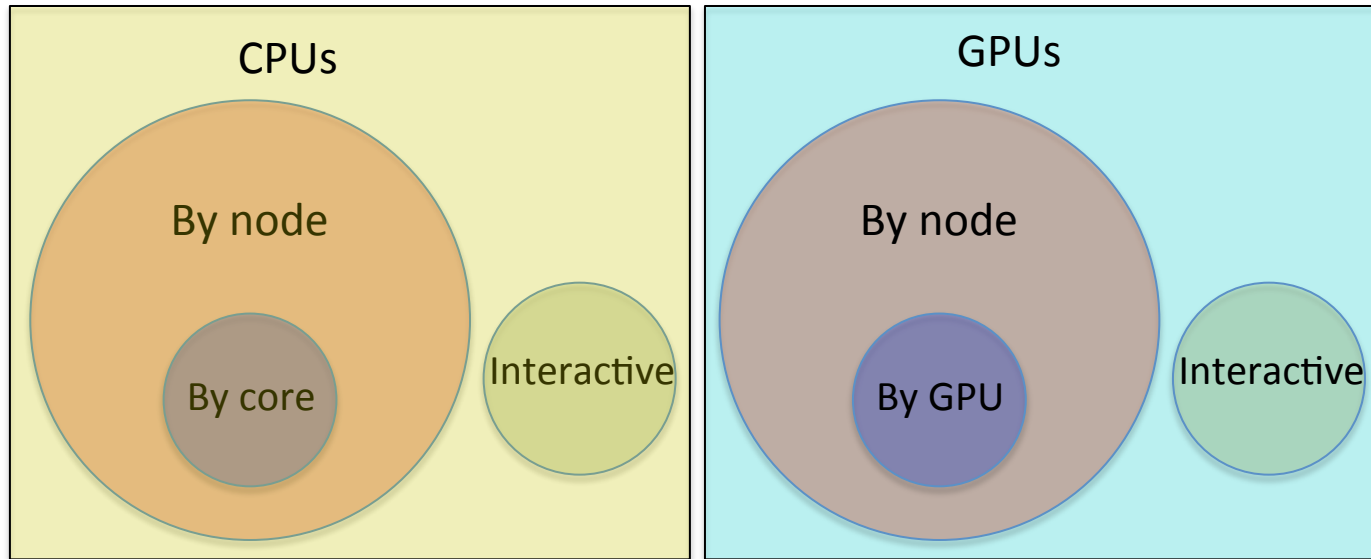
# Node types on Cedar

Total Mem TB	Cores	Memory	GPUS	Number of Nodes	Partition type
1/8	32	4GB/core		576	cpubase
1/4	32	8GB/core		182	cpubase
1/2	32	16GB/core		24	cpularge
1.5	32	48GB/core		24	cpularge
3	32	96GB/core		4	cpularge
1/8	24	32GB/GPU	4	114	gpubase
1/4	24	64GB/GPU	4	132	gpularge

# Node types on Graham

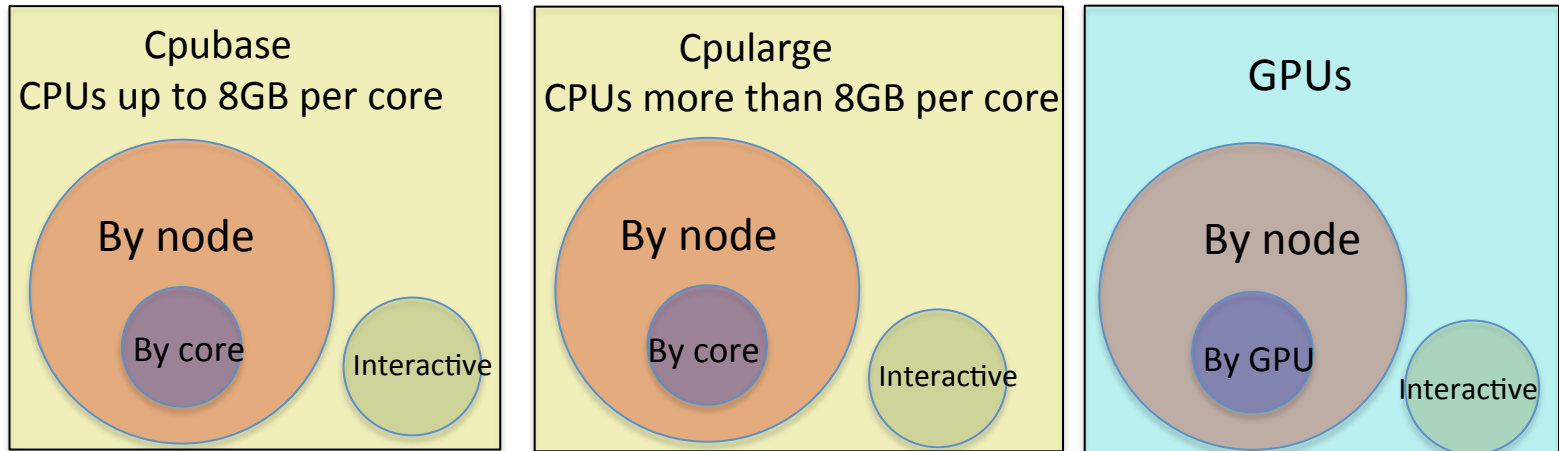
Total Mem TB	Cores	Memory	GPUS	Number of Nodes	Partition Type
1/8	32	4GB/core		800	cpubase
1/4	32	8GB/core		55	cpubase
1/2	32	16GB/core		24	cpularge
3	32	96GB/core		3	cpularge
1/8	32	32GB/GPU	4	114	gpubase

# Partitions on Cedar and Graham



- Separate partitions for GPUs and CPU request
- Nodes that are in the by core partition are also in the by node partition, the reverse is not always true.
- There are separate interactive (testing) partitions with dedicated nodes for interactive usage.

# Partitions on Cedar and Graham



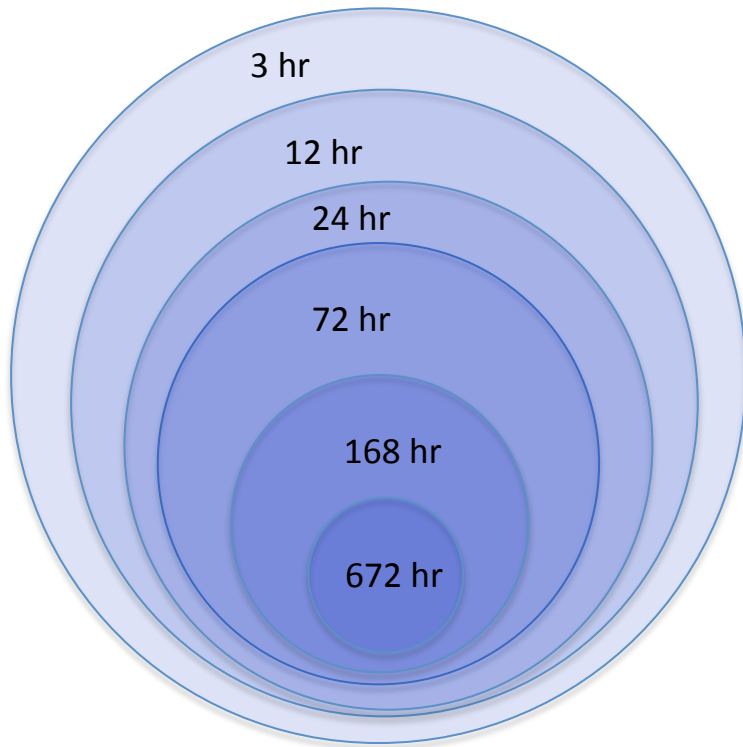
- Separate partitions for large memory Nodes and jobs that have more than 8 GB RAM and smaller memory nodes and jobs.
  - This is done to disallow low memory jobs from stopping a large memory job from running quickly on the few expensive large memory nodes we have.

# Partitions why the complexity?

- If we allowed serial jobs to run on all nodes, the chances that there was a node that had all 32 cores not used or coming to an end soon would be very small.
  - if  $\frac{1}{2}$  the cluster was empty and the job distributed randomly the chances a any particular node to be empty =  $\frac{1}{2^{32}} = \frac{1}{4,294,967,296}$
- As a consequence whole node jobs would in practice all have to wait (max walltime) time to start regardless of priority.
- If the whole cluster only allows allocation to jobs by node jobs by core will not run or people would ask for a node and use a single core.



# Partitions on Cedar and Graham



- There are partitions based upon how long the maximum walltime your job has.
- Your job ends up in the shortest walltime partition that has a longer walltime than your job
- The shorter walltime partitions include all the nodes of longer walltime partitions.

# Maximum job walltime partition limit

- A high maximum walltime is not necessary a good thing, clusters that allow high walltime jobs take longer for jobs to start to run, and are less “fair”.
- There are advantages to running shorter jobs, such as how quickly your job can be started.
- The longer and larger a job is the greater the chances of experiencing hardware failure, minimize this through check pointing.
- Part of the resources of a cluster is dedicated for shorter jobs.
- Part of CC clusters are dedicated to whole node parallel jobs, other jobs with a short walltime of under 12 hours can run in this part at a reduced priority compared to whole node parallel jobs.

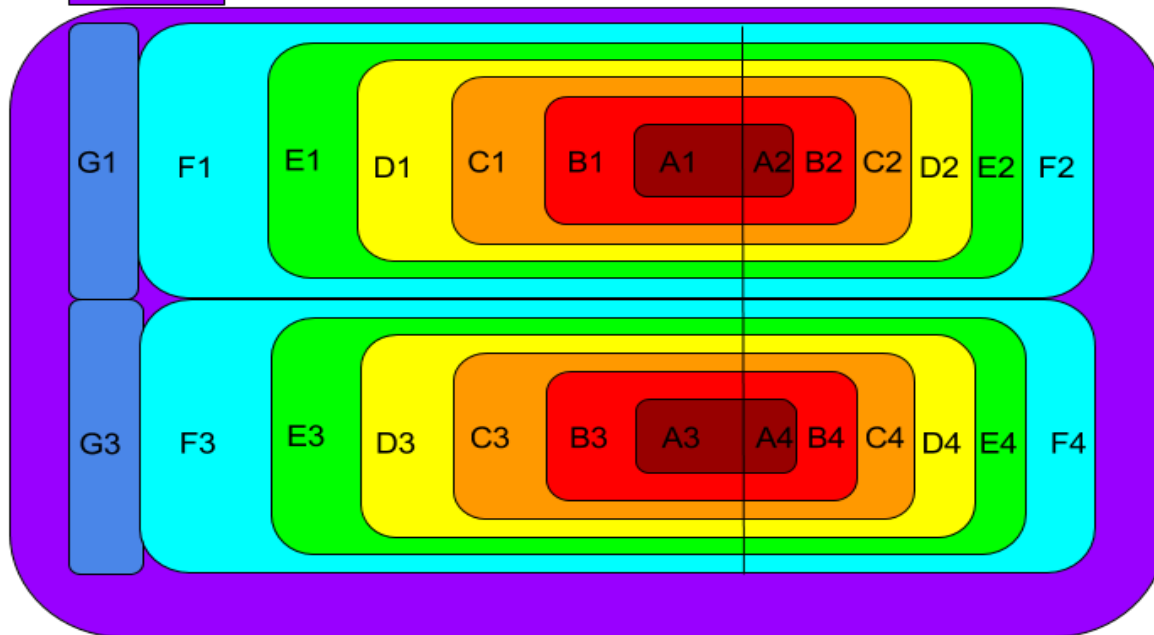
Partition name	Maximum walltime
*_b1	3 hours
*_b2	12 hours
*_b3	1 day
*_b4	3 days
*_b4	7 days
*_b6	28 days

# Partitions why the complexity?

- Some jobs need to run a long time
  - Commercial code that does not checkpoint
  - Checkpoints can take a very long time
- If we allow all nodes to run long walltime jobs
  - It would take a long time for resources to become available, researchers that need to run short jobs and analyze the result before running another would find the system unusable.
  - People that can divide their work arbitrarily would run long walltime jobs as they have already waited a long time for their job to start, making the situation worse.
- CC has dealt with the situation in the past by having different clusters each has different walltimes. But there are not enough clusters to do this anymore.
- The solution of concentric partitions on larger cluster allows us to more efficiently address diverse user needs.

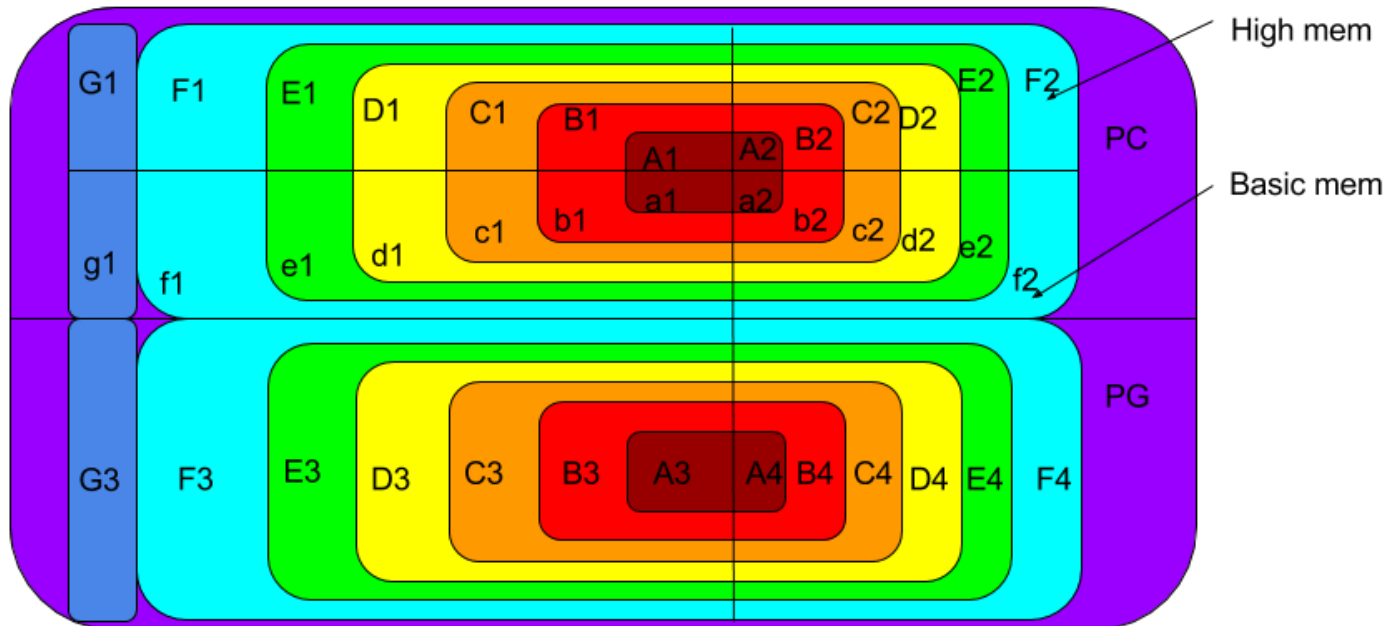
# Partitions on Cedar and Graham

Walltime	Whole node cpu	By core cpu	Whole node gpu	By gpu
768 hr	A1 + A2	A2	A3 + A4	A4
168 hr	B1 + B2	B2	B3 + B4	B4
72 hr	C1 + C2	C2	C3 + C4	C4
24 hr	D1 + D2	D2	D3 + D4	D4
12 hr	E1 + E2	E2	E3 + E4	E4
3 hr	F1 + F2	F2	F3 + F4	F4
interactive	G1		G3	
Preemptable				



# Partitions on Cedar and Graham

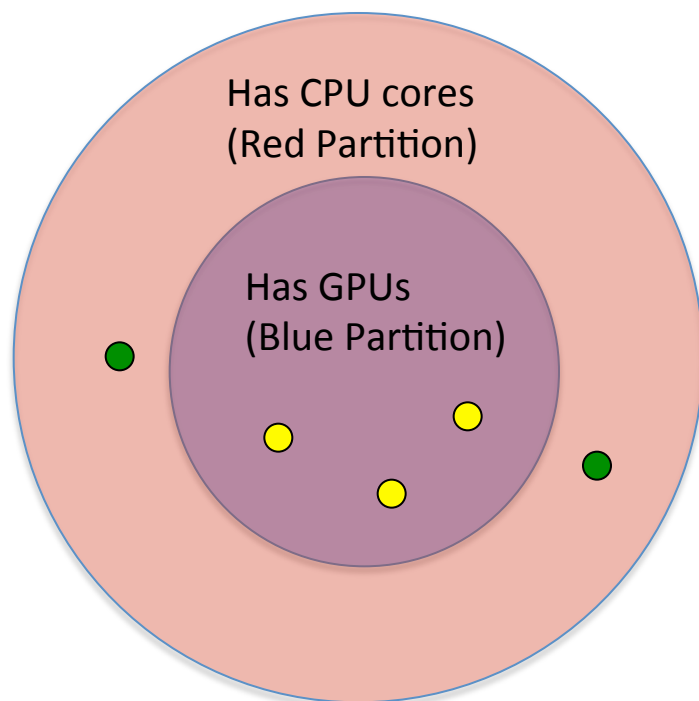
Walltime	Whole node cpu	By core cpu	Whole node gpu	By gpu
768 hr	A1 + A2	A2	A3 + A4	A4
168 hr	B1 + B2	B2	B3 + B4	B4
72 hr	C1 + C2	C2	C3 + C4	C4
24 hr	D1 + D2	D2	D3 + D4	D4
12 hr	E1 + E2	E2	E3 + E4	E4
3 hr	F1 + F2	F2	F3 + F4	F4
Low Priority Backfill Short Jobs	F1 + F2 + f1 + f2		F3 + F4	
interactive	G1 + g1		G3	
Preemptable	PC		PG	



(CC script)

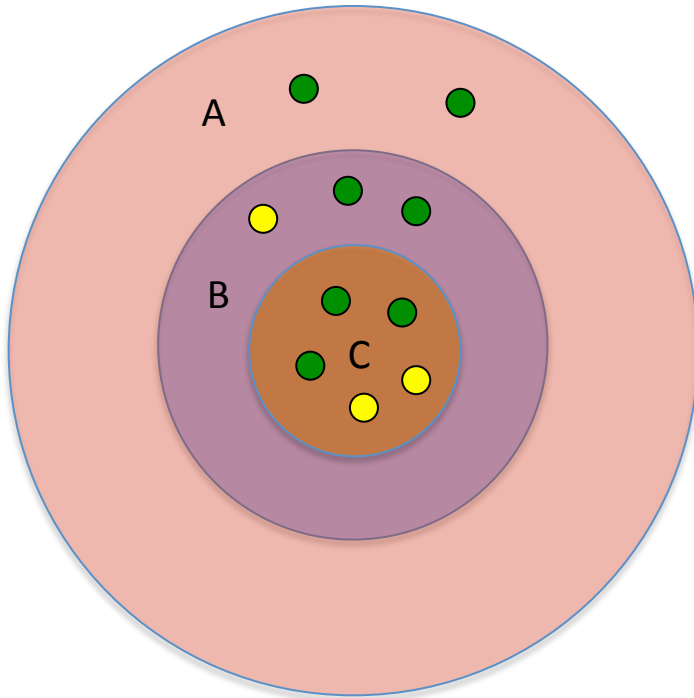
Node type	Max walltime					
	3 hr	12 hr	24 hr	72 hr	168 hr	672 hr
-----						
Number of Queued Jobs by partition Type (by node:by core)						
-----						
Regular	1:15	2:31	2:145	11:187	86:69	3:2
Large Mem	0:0	0:0	0:0	0:0	0:1	0:1
GPU	0:1	0:526	10:10	0:0	189:4	0:0
-----						
Number of Running Jobs by partition Type (by node:by core)						
-----						
Regular	60:6	4:2	45:836	5:90	11:1065	1:4
Large Mem	0:0	0:0	0:0	0:0	0:0	1:0
GPU	0:20	2:10	13:2	0:0	0:0	0:3
-----						
Number of Idle nodes by partition Type (by node:by core)						
-----						
Regular	0:0	0:0	0:0	0:0	0:0	0:0
Large Mem	3:1	0:0	0:0	0:0	0:0	0:0
GPU	17:1	11:1	0:0	0:0	0:0	0:0
-----						
Total Number of nodes by partition Type (by node:by core)						
-----						
Regular	851:411	821:391	756:346	636:276	180:100	90:50
Large Mem	27:12	24:11	24:11	20:3	3:2	2:1
GPU	156:78	144:72	116:58	104:52	13:12	13:12
-----						

# Partitions and priority



- Black dots are nodes
- In this example we have:
  - 5 nodes with CPUs (Red partition)
  - 3 nodes with GPUs (Blue partition)
  - 2 nodes have CPUs but not GPUs
- A Job that requires CPUs (red partition) can run on any of the 5 nodes
- A job that requires GPUS (blue partition) can run on any of the 3 nodes.
- In the case that the two nodes with no gpus in the red partition may be idle(green) and 3 nodes with gpus may be busy.
  - A job that requires a GPU node (from the blue partition) will be unable to start if no GPU nodes are idle. A job that requires CPUs only (Red partition) will be able to start immediately, even when there are higher priority jobs in the blue partition.

# Partitions and priority example

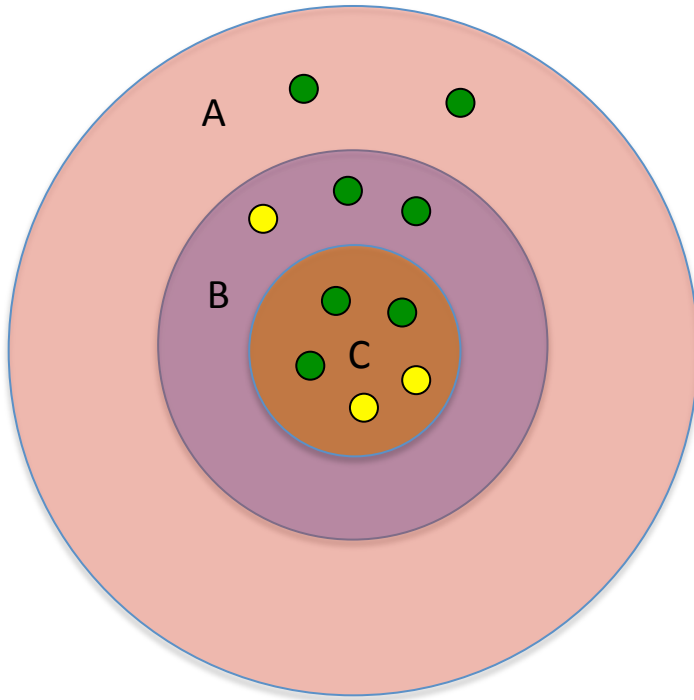


- Partition A has 3 hour walltime and includes all the nodes of this type on the cluster
- Partition B is the largest partition that your job can run in.
- Partition C is a subset of partition B and contains jobs that have a longer walltime and nodes that can run those jobs.
- Each small green circle represents an idle node
- Each small yellow circle represents a busy node

● Idle node  
● Busy node



# Partitions and priority example



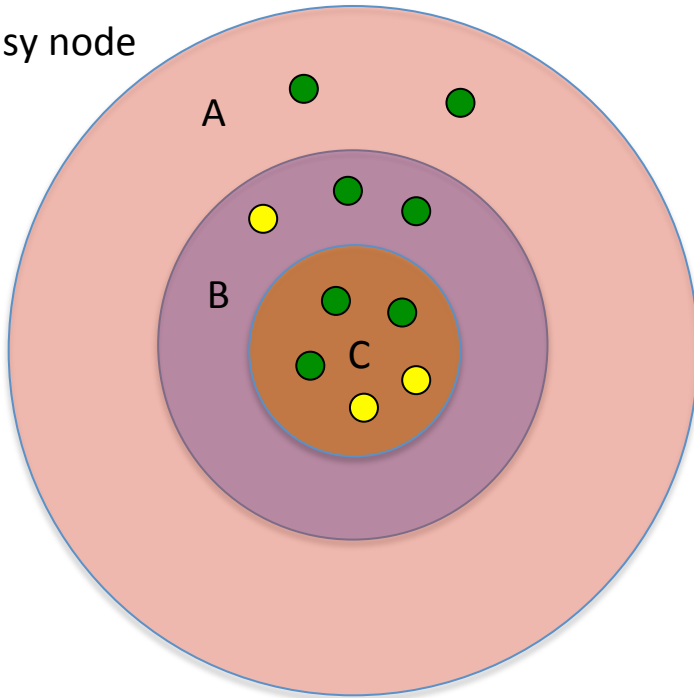
Lets assume we have 3 jobs:

- Highest priority job (1) in partition C that requires 4 nodes.
- 2<sup>nd</sup> highest job in partition job (2) in partition A that requires 5 nodes.
- Our job in partition B that requires 2 nodes

- Idle node
- Busy node

# Partitions and priority example

- Idle node
- Busy node

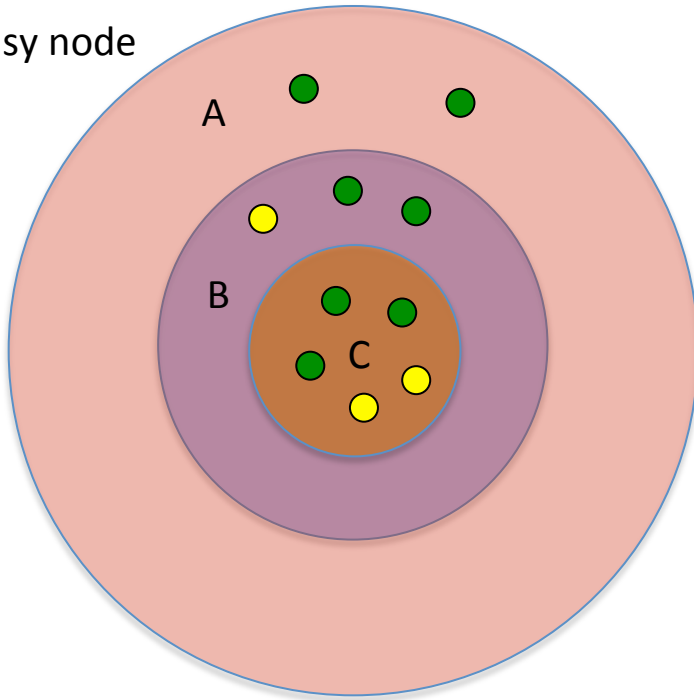


- Highest priority job (1) in partition C that requires 4 nodes.
- 2<sup>nd</sup> highest job (2) in partition A that requires 5 nodes.
- Our job (3) in partition B that requires 2 nodes

- Job 1 cannot run as there are only 3 idle nodes in partition C.
  - A reservation is created for the idle nodes in partition C and the first of the busy nodes that will become available.
- Job 2 likely cannot run either as it needs one of the nodes reserved by job 1, and unless job 2 can finish before job 1 starts it will not be able to run.
- Job 3 will likely not run as well because it requires resources (nodes) that are reserved by other higher priority jobs.

# Partitions and priority example

- Idle node
- Busy node

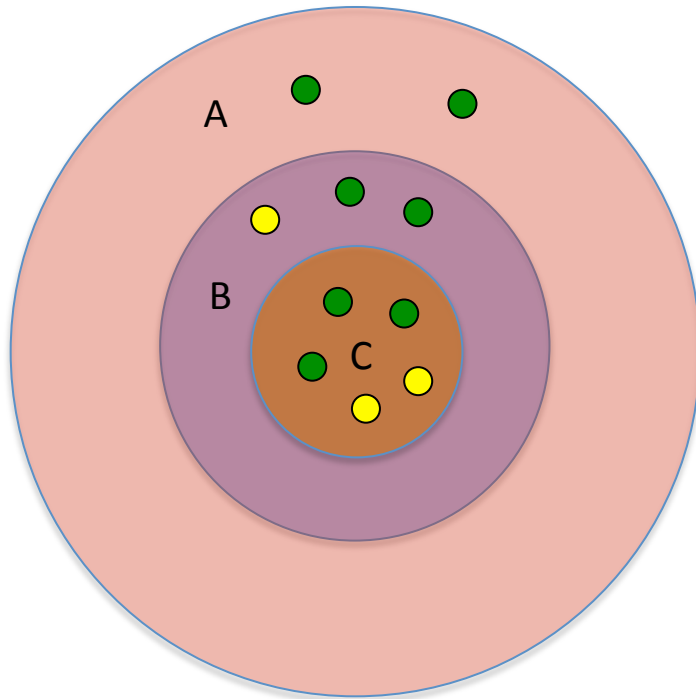


- Highest priority job (1) in partition C that requires 4 nodes.
- 2<sup>nd</sup> highest job (2) in partition A that requires 5 nodes.
- Our job (3) in partition B that requires 2 nodes

This cluster is 70% idle and jobs cannot run why?

- The example cluster is small and the jobs are large in comparison
- There are no short single node jobs that can fill in these empty nodes.
- This example was created to show a worse case scenario

# Partitions and priority lessons learned



- Submit smaller, shorter jobs
- When looking at priority and why your job is not running, look at the priority of other jobs in the partitions that are either a subset or superset of your job.
- The situation in Compute Canada will get better when Niagara is up as that system is designed for large jobs. The types of jobs on Cedar and Graham will become less diverse and we will be better able to efficiently schedule similar and smaller jobs on Graham and Cedar.

● Idle node  
● Busy node

# Slurm script commands

PBS script command	Description
#SBATCH --mem=4000	Requests 4000 MB of memory in total
#SBATCH --mem-per-cpu=4000	Requests 4000 MB of memory per cpu
#SBATCH --licenses=sas:2	Requests 2 SAS licenses
#SBATCH --gres=gpu:1	Requests that your job get 1 GPU allocated per node
#SBATCH --exclusive	Requests that your job run only on nodes with no other running jobs
#SBATCH --dependency=after:job_id1	Requests that the the job start after job (jobid1) has <b>started</b>
#SBATCH --dependency=afterany:job_id1, job_i2	Requests that the the job start after ether job (jobid1) or job (jobud2) has <b>finished</b>
#SBATCH --dependency=afterok:job_id1	Requests that the the job start after job (jobid1) has <b>finished successfully</b>

# Slurm script commands

PBS script command	Description
#SBATCH --account=acc_name	To submit a job to a specific accounting group such as RAC/RAS allocation or different role
#SBATCH --tmp=200G	Asks for 200Gb of temporary disk space
#SBATCH --constraint=blue	To ask for a node feature or constraint set by cluster admin. Here we are looking for “blue” nodes.
#SBATCH --partition=partition_name	To ask for the job to run in a specific partition or queue by name, (unlike Moab there can be more than 1 partition per Job)
--prolog=<executable>	Run by srun only, runs the executable before the step
--epilog=<executable>	Run by srun only, runs the executable after the step finishes

# SLURM Environment Variables

Environment Variable	Description
SLURM_JOB_NAME	User specified job name
SLURM_JOB_ID	Unique slurm job id
SLURM_NNODES	Number of nodes allocated to the job
SLURM_NTASKS	Number of tasks allocated to the job
SLURM_ARRAY_TASK_ID	Array index for this job
SLURM_ARRAY_TASK_MAX	Total number of array indexes for this job
SLURM_MEM_PER_CPU	Memory allocated per CPU
SLURM_JOB_NODELIST	List of nodes on which resources are allocated to Job
SLURM_JOB_CPUS_PER_NODE	Number of CPUs allocated per Node
SLURM_JOB_PARTITION	List of Partition(s) that the job is in.
SLURM_JOB_ACCOUNT	Account under which this job is run.

Job submission practice

**BREAK FOR PRACTICE**



# Getting information on your Job

Command	What its used for
<code>squeue -u &lt;username&gt;</code>	List all current jobs for a user
<code>squeue -u &lt;username&gt; -t PENDING</code>	List all pending jobs for a user
<code>squeue -u &lt;username&gt; -t RUNNING</code>	List all running jobs for a user
<code>Squeue -p &lt;partitionname&gt;</code>	List all the jobs in a partition
<code>scontrol show job &lt;jobid&gt;</code>	List information on Job
<code>scontrol show jobid -dd &lt;jobid&gt;</code>	List detailed information on Job
<code>Squeue -o "%.18i %.30P %.8j %.8u %.2t %.8p %.10M %.6D %R "</code>	Formatted output of squeue: we added priority and made the partition field bigger (30 characters)

---

# Getting information on your Job

Command	What its used for
<code>sstat -- format=AveCPU,MaxRSS,MaxV MSize,JobID -j &lt;jobid&gt;</code>	List info resource used by your completed job : average cpu time, Max memory, Max virtual memory, JobId
<code>sacct -u &lt;username&gt; -- format=JobID,JobName,AveCPU ,MaxRSS,MaxVMSize,JobID,Elap sed</code>	List resources used by all jobs of a user
<code>sprio</code>	List job priority information

---

# queue

```
kamil@zeno ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST	REASON
2020_1	mem12_sho	my-array	kamil	R	0:04	1	zeno001	
2020_4	mem12_sho	my-array	kamil	R	0:04	1	zeno001	
2019	mem12_sho	my-named	judy	R	0:11	1	zeno001	

# Queue command for user

## Queue -u \$USER

```
[kamil@zeno ~]$ queue -u kamil
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
2025	mem12_sho	anwser-q	kamil	R	0:01	1	zeno001
597520	cpubase_b	aln_ERR1	kamil	PD	0:00	1	(Dependency)
597540	cpubase_b	aln_SRR9	kamil	PD	0:00	1	(Dependency)
598316	cpubase_b	aln_SRR2	kamil	PD	0:00	1	(DependencyNeverS
598324	cpubase_b	aln_SRR2	kamil	PD	0:00	1	(DependencyNeverS

# Queue command for queued jobs

## queue -u <username> -t PENDING

```
[kamil@zeno ~]$ queue -u kamil -t pending
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(REASON)
597520	cpubase_b	aln_ERR1	kamil	PD	0:00	1	(Dependency)
597540	cpubase_b	aln_SRR9	kamil	PD	0:00	1	(Dependency)
598316	cpubase_b	aln_SRR2	kamil	PD	0:00	1	(DependencyNeverS
598324	cpubase_b	aln_SRR2	kamil	PD	0:00	1	(DependencyNeverS
619783	cpubase_b	ala1805S	kamil	PD	0:00	1	(Priority)
617318	cpubase_b	Pseudomo	kamil	PD	0:00	1	(Resources)
617319	cpubase_b	Pseudomo	kamil	PD	0:00	1	(Resources)

# queue -u <username> -t RUNNING

```
[kamil@cedar ~]$ queue -u kamil -t running
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(REASON)
2026	mem12_sho	anwser-q	kamil	R	0:02	1	zeno001
620930	cpubase_b	HRAGR001	kamil	R	23:58	1	cdr57
617805	cpubase_b	Ro:0	kamil	R	9:44:23	4	cdr[72,88,92,95]
584942	cpubase_b	runmpi.s	kamil	R	2-11:09:29	4	cdr[81-83,98]
574866	cpubase_b	Ro:-0.08	kamil	R	2-22:21:17	5	cdr[77,79-80,84,9
618505	cpubase_b	Bowtie2_	kamil	R	9:42:10	1	cdr215

# Jobs by partition

`queue -p <partitionname>`

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(P
535639	cpubase_b	AE17631.	kamil	PD	0:00	1	(Resource
591830	cpubase_b	bz.sh	ermin	PD	0:00	1	(Resource
615762	cpubase_b	AE21380.	kamil	PD	0:00	1	(Resource
401219	cpubase_b	CTD095.s	john	PD	0:00	1	(Resource
491576	cpubase_b	gen3x1s8	judy	R	2-08:04:59	1	cdr747
535638	cpubase_b	AE17594.	kamil	R	1-11:46:03	1	cdr101
491574	cpubase_b	gen3x1s6	masao	R	4-20:06:44	1	cdr79
491575	cpubase_b	gen3x1s7	masao	R	4-20:06:44	1	cdr85

# Squeue queries

```
squeue -o "%.18i %.30P %.8j %.8u %.2t %.8p %.10M %.6D %R " -u <username>
```

```
[kamil@cedar5 test]$ squeue -o "%.18i %.30P %.8j %.8u %.2t %.8p %.10M %.6D %R " -u kamil
```

JOBID	PARTITION	NAME	USER	ST	PRIORITY	TIME	NODES	NODELIST	REASON
597520	<b>cpubase_bycore_b1,cpubackfill</b>	aln_ERR1	kamil	PD	<b>0.001164</b>	0:00	1	(Depend	
597540	<b>cpubase_bycore_b1,cpubackfill</b>	aln_SRR9	kamil	PD	<b>0.001164</b>	0:00	1	(Depend	
597592	<b>cpubase_bycore_b1,cpubackfill</b>	aln_SRR5	kamil	PD	<b>0.001164</b>	0:00	1	(Depend	
597593	<b>cpubase_bycore_b1,cpubackfill</b>	aln_SRR8	kamil	PD	<b>0.001164</b>	0:00	1	(Depend	



# scontrol show job <jobid>

```
[kamil@zeno ~]$ scontrol show job 2026
JobId=2026 JobName=anwser-q3.sh
  UserId=kamil(1005) GroupId=slurmteam(1007) MCS_label=N/A
  Priority=38885 Nice=0 Account=team1 QOS=mem12_short
  JobState=COMPLETED Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:00:31 TimeLimit=00:02:00 TimeMin=N/A
  SubmitTime=2017-03-22T13:51:02 EligibleTime=2017-03-22T13:51:02
  StartTime=2017-03-22T13:51:02 EndTime=2017-03-22T13:51:33 Deadline=N/A
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  Partition=mem12_short AllocNode:Sid=zeno:31494
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=zeno001
  BatchHost=zeno001
  NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=1,mem=1948M,node=1
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryCPU=1948M MinTmpDiskNode=0
  Features=(null) Gres=(null) Reservation=(null)
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/home/kamil/anwser-q3.sh
  WorkDir=/home/kamil
  StdErr=/home/kamil/slurm-q1-2026.err
  StdIn=/dev/null
```

# Priority sprio

JOBID	PRIORITY	AGE	FAIRSHARE	PARTITION	TRES
130976	7088	2500	0	625	cpu=2526,mem=1437
167003	6150	2500	0	1250	cpu=2008,mem=392
195802	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195809	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195810	4996086	2500	4991771	833	cpu=469,mem=45,gres/
205281	8206	2500	0	625	cpu=1875,mem=1800,gr
205290	6408	2500	0	625	cpu=1875,mem=2,gres/
544814	23534	1741	21571	208	cpu=13,mem=2
544815	23534	1741	21571	208	cpu=13,mem=2
617580	24194	373	22768	1042	cpu=10,mem=2
617581	24194	373	22768	1042	cpu=10,mem=2

# Demonstration on cluster

- SSH cluster and show all the following commands and how to interpret them
  - `queue`
  - `queue -u $USER`
  - `queue -t pending`
  - `queue -t running`
  - `queue -p <partition>`
  - `queue` (custom format)
  - `scontrol show job <jobid>`
  - `Sprio -n`

Job information practice

**BREAK FOR PRACTICE**

**QUESTIONS?**

# Upcoming ARC Training Sessions

<b>October 25</b> 10am - 11pm MDT	<b>Machine Learning Using Jupyter Notebooks on Graham</b>
<b>November 1</b> 11am – 1 pm MDT	<b>Introduction to Classical Molecular Dynamics Simulations</b>
<b>November 21</b> 11am – 1 pm MDT	<b>Exploring Containerization with Singularity</b>
<a href="https://www.westgrid.ca/events/westgrid-training-events">https://www.westgrid.ca/events/westgrid-training-events</a>	

# Scheduling and Job Management 3

Using a cluster effectively

# Presentation contents

Priority, Allocations and Fairshare  
Cluster limits, Reservations and Topology  
Getting information on your Cluster  
Trouble shooting your jobs



# Priority

- Can only be positive in slurm.
- Only relative priority matters.
- Jobs with highest or least negative priority get reservation to run first.
- Highest priority job may not run first.  
A job which is using a small amount of resources that are in great supply may easily run before a high priority job requesting scarce or already used resources.
- In Compute Canada priority is determined per group via “fairshare” and how long your job sits in the queue
- “sprio” will show priority of your job

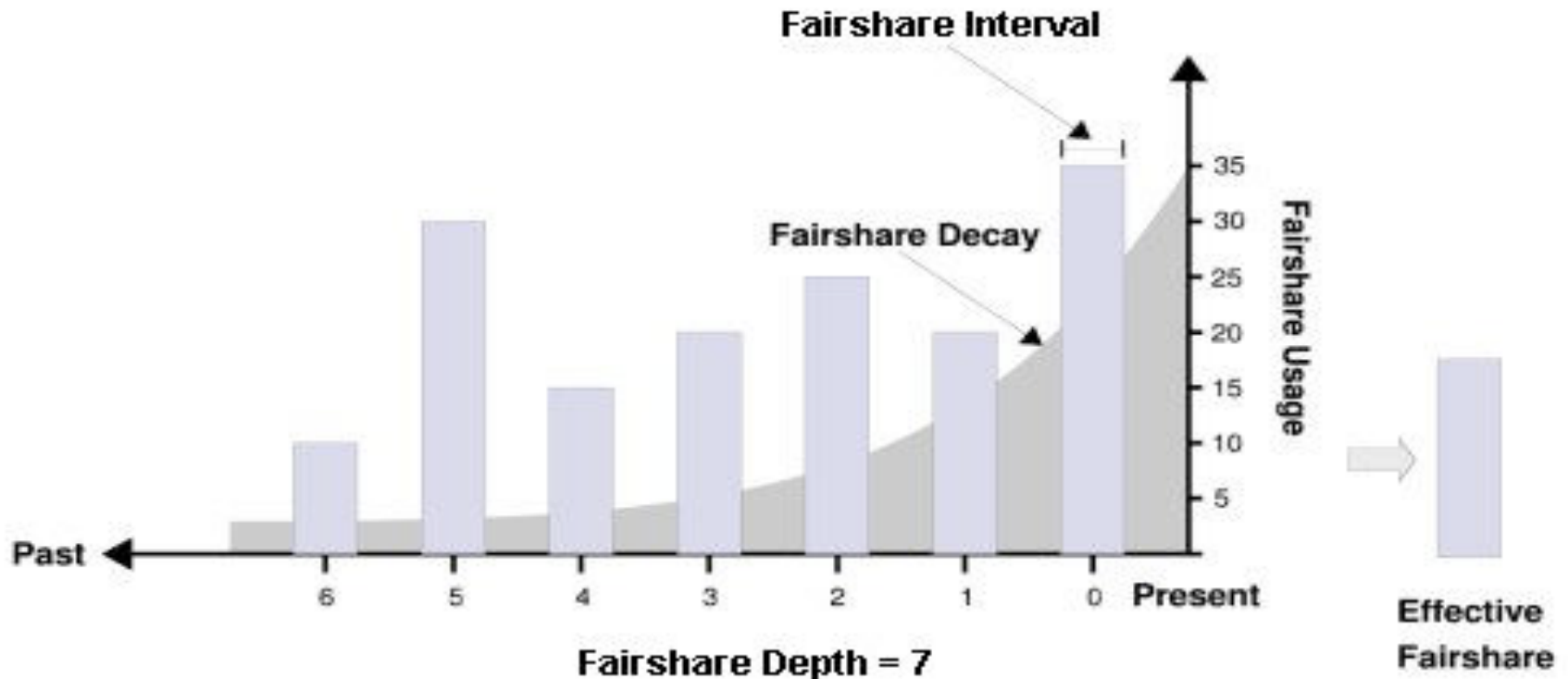
# Priority sprio

JOBID	PRIORITY	AGE	FAIRSHARE	PARTITION	TRES
130976	7088	2500	0	625	cpu=2526,mem=1437
167003	6150	2500	0	1250	cpu=2008,mem=392
195802	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195809	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195810	4996086	2500	4991771	833	cpu=469,mem=45,gres/
205281	8206	2500	0	625	cpu=1875,mem=1800,gr
205290	6408	2500	0	625	cpu=1875,mem=2,gres/
544814	23534	1741	21571	208	cpu=13,mem=2
544815	23534	1741	21571	208	cpu=13,mem=2
617580	24194	373	22768	1042	cpu=10,mem=2
617581	24194	373	22768	1042	cpu=10,mem=2

# Fairshare

- Fairshare is a mechanism that allows historical resource utilization information to be incorporated into job feasibility and priority decisions
- In SLURM priority ranges from 1 to 0
- In Compute Canada fairshare compares your group's target usage to your group's actual usage during a time period. If your group has used less than your group share you are given higher priority.

# Fairshare



- Fair share usage is weighted by when the usage occurred recent usage is more important than usage at the end of the period

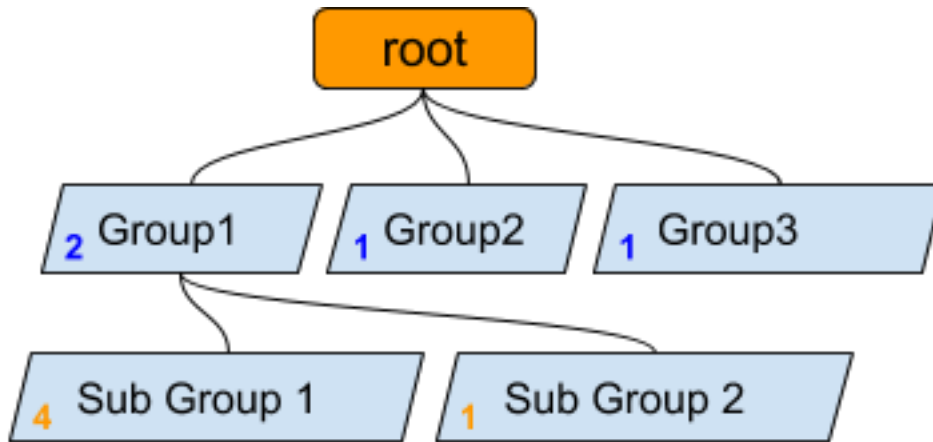
# Fairshare tree CPUs and GPUs and Equivalents

- We use GPU or CPU equivalent resources in all our calculations.
  - If your job uses all (memory/disk/any other resource) on a node and half the CPUs the scheduling system will “charge” or use in its calculations as if you used all the CPUs on that node.
- Separate accounting groups for CPUs and GPUs
- For GPU jobs we only count number of GPU used or GPU equivalent in terms of other resources.

# Fairshare trees

- It is possible for project leader to divide the target allocations of resources for the group.
- Your priority is determined by a combination of your group's usage compared to your group's target usage, as well as your subgroups usage compared to subgroup target share as well as your individual usage in the group compared to your individual target in the group.
- The priority of anyone's job will primarily be influenced by the top of the tree rather than the subgroups/individual usage.

# Fairshare tree Basics






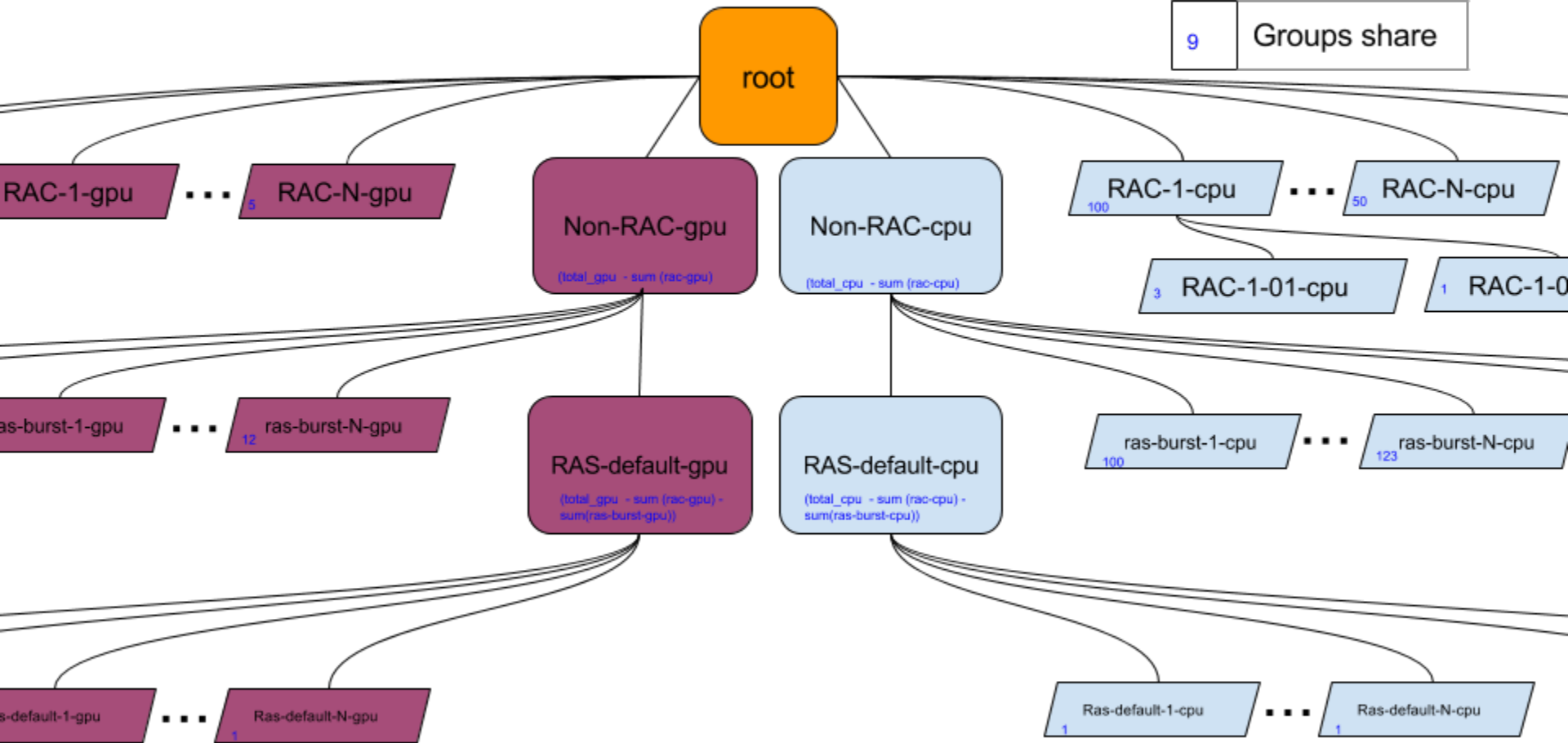
- Group 1 has normalized share 50%  
 $= (2/(2+1+1))$
- Subgroup 1 has normalized share 40%  
 $= (4/(4+1) * \text{Group 1 share})$   
 $= (4/(4+1) * (2/(2+1+1)))$

- Fairshare trees shares are different shares on different levels.
- Fairshare tree shares don't mean anything other than meaning we give them if they add up to 100 then its percent if they add up to the number of cores then each share is expressed in cores
- In SLURM shares have to be integers
- Usage in a group is includes the usage by sub groups
- Normalized shares are the fraction of the system that the group or user receives

# CC Slurm Fairshare tree

Version 1.5 (subgroups)

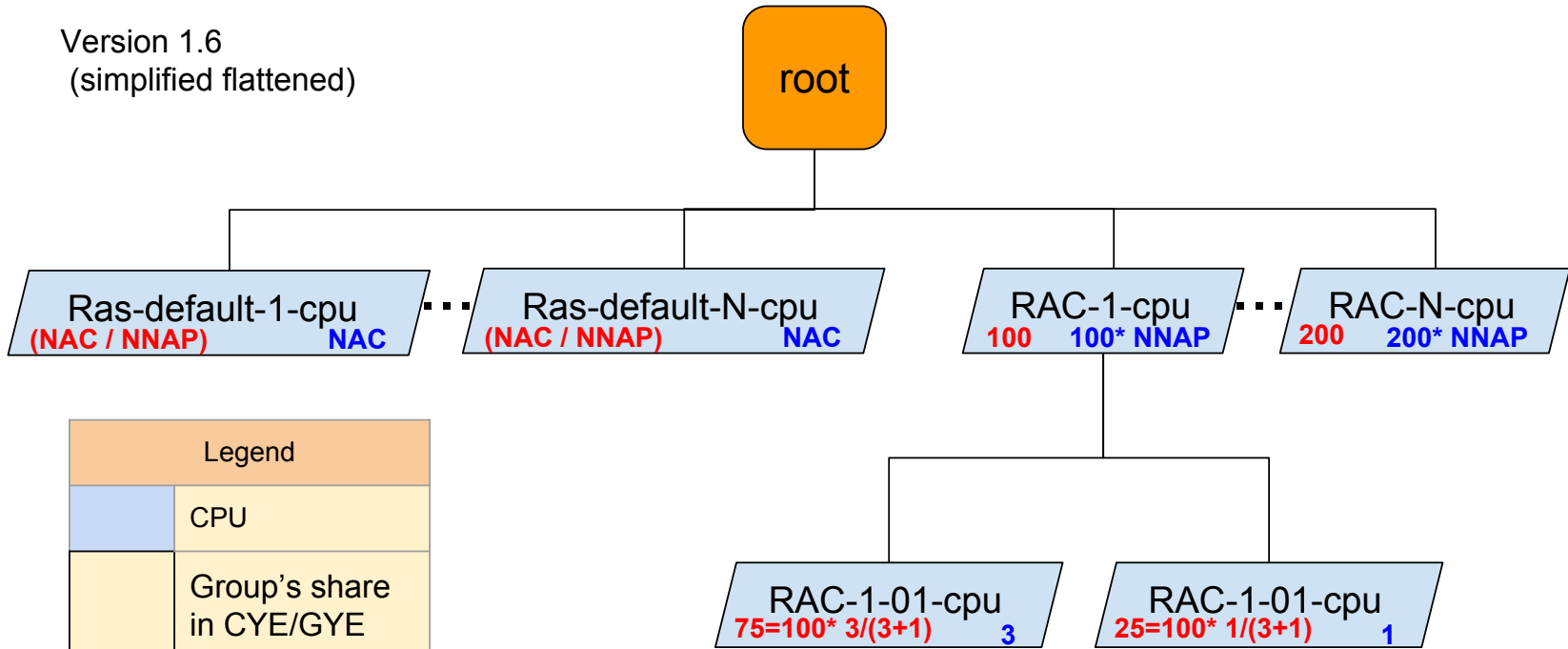
Legend	
	CPU
	GPU
	Expired
9	Groups share





# Upcoming CC Slurm Fairshare tree

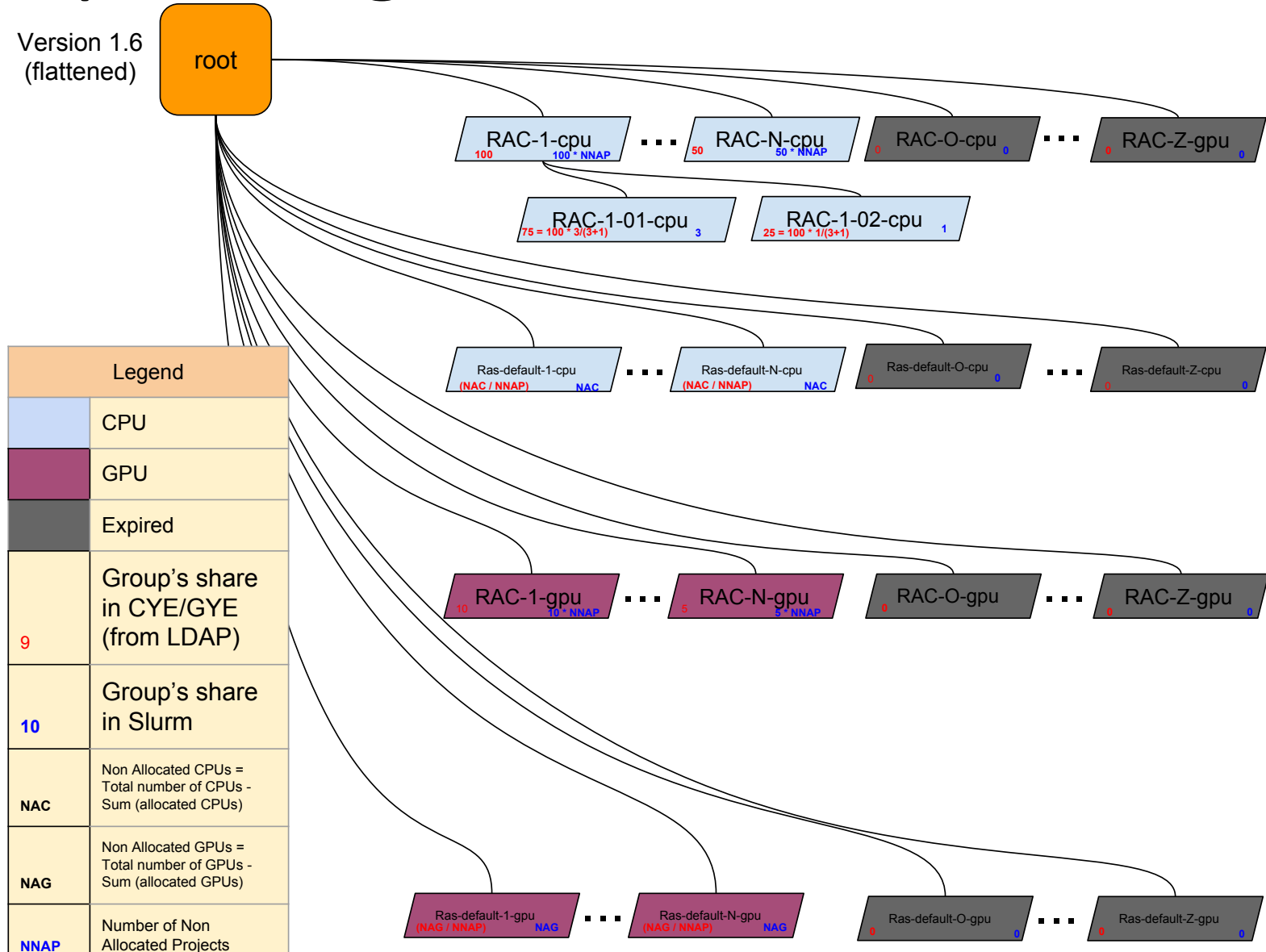
Version 1.6  
(simplified flattened)



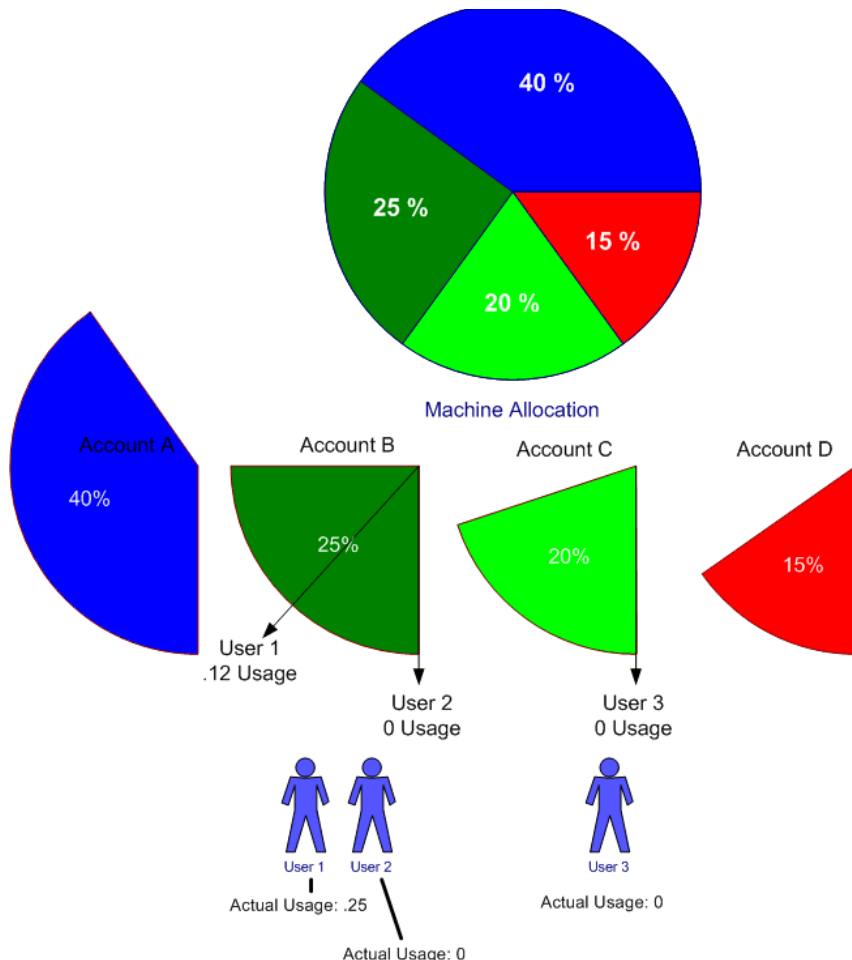
Legend	
	CPU
9	Group's share in CYE/GYE (from LDAP)
10	Group's share in Slurm
NAC	Non Allocated CPUs = Total number of CPUs - Sum (allocated CPUs)
NNAP	Number of Non Allocated Projects

# Upcoming CC Slurm Fairshare tree

Version 1.6  
(flattened)



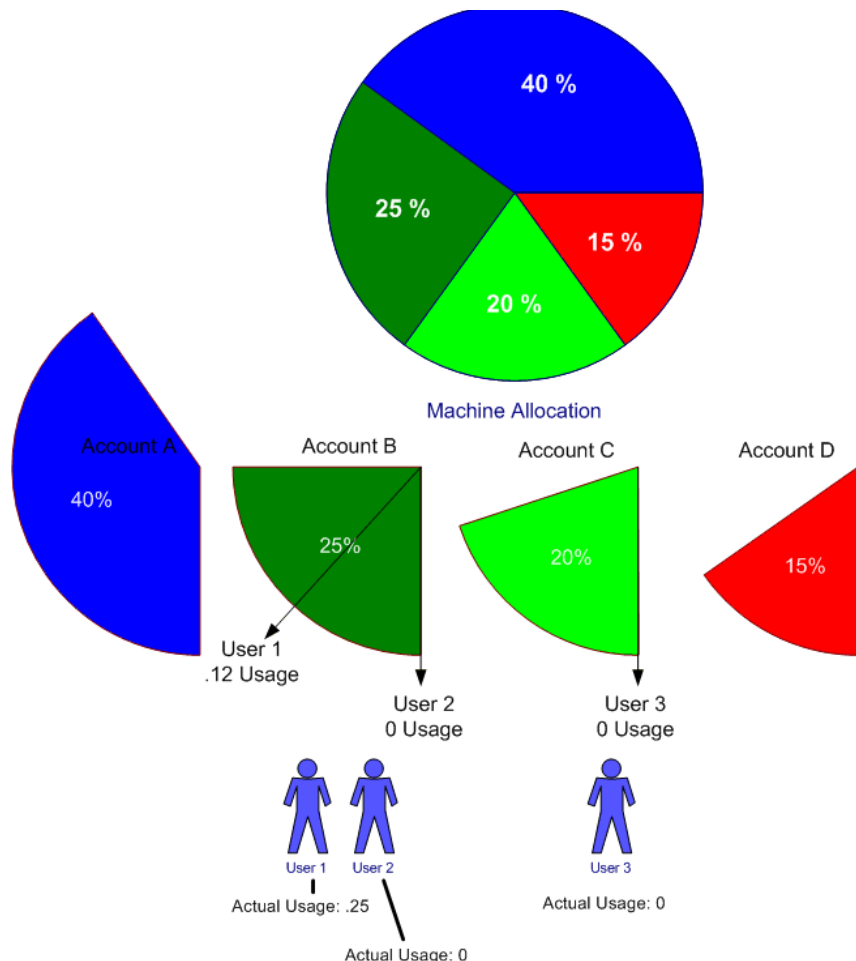
# Multi level fairness



“Another layer of "fairness" is necessary however, one that factors in the usage of other users drawing from the same account. This allows a job's fair-share factor to be influenced by the computing resources delivered to jobs of other users drawing from the same account.”

# Effective usage

(No longer used by CC)



- No longer used in FS calculations in the new “fair tree fairshare tree” algorithm CC uses but its still reported by scheduling system.
- An individual who has not run any Jobs will have a nonzero effective usage if his group or its parent group has been running jobs.
- Was used by the standard fairshare tree slurm algorithm to achieve Multi level fairness.
- Effective usage takes into account the Effective usage of the parent group as well as the actual usage of the individual.

[https://slurm.schedmd.com/priority\\_multifactor.html](https://slurm.schedmd.com/priority_multifactor.html)

# “Fair tree” Fairshare tree priority algorithm

- Algorithm works by calculating “level fairshare” which is:  $LF = \text{Share} / \text{Use}$  at each level of the fairshare tree.
- Orders all the accounts and users in the level.
  - For each account and user in the tree, calculates the level fairshare and does the same at the next level
- Traverse the tree and order/rank/number all user accounts.
  - Use zero based counting here, first user will be 0
- Priority is given in the following formula: 
$$P = \frac{(\text{UserCount} - \text{UserRank})}{(\text{UserCount})}$$

ex: If there are 3 users the priority of the middle user

$$P = \frac{(\text{UserCount} - \text{UserRank})}{(\text{UserCount})} = \frac{(3-1)}{(3)} = \frac{2}{3} = 0.67$$

- More information available here: [https://slurm.schedmd.com/SUG14/fair\\_tree.pdf](https://slurm.schedmd.com/SUG14/fair_tree.pdf)

# Priority

- Job priority is the sum of all the weighted sum of all the factors that have been enabled.
- $\text{Job\_priority} = (\text{PriorityWeightAge} * \text{age\_factor}) + (\text{PriorityWeightFairshare} * \text{fair-share\_factor}) + (\text{PriorityWeightPartition} * \text{partition\_factor}) + \text{other stuff}$
- This allows us to give greater priority to jobs that have been waiting in the queue a long time and determine how important that is relative to fairshare priority.
- Without an age factor a larger job by a user with a small allocation could never run.

# Group's Status: "sshare"

```
[kamil@cedar5 workshop_test]$ sshare | egrep "(--|Account|^root|no_rac_|ras_b|cc-debug|kamil_)"
```

Account	User	RawShares	NormShares	RawUsage	NormUsage	EffectvUsage	FairShare
root			1.000000	56519806629365289		1.000000	0.500000
no_rac_cpu		3083	0.123572	54311297258252622	0.960925	0.960925	0.004562
ras_basic_cpu		3083	0.123532	54311297258252622	0.960925	0.960925	0.004554
cc-debug_cpu		1	0.000031	120455	0.000000	0.000237	0.004554
cc-debug_cpu	kamil	1	0.000000	0	0.000000	0.000001	0.004554
def-kamil_cpu		1	0.000031	46106596622	0.000001	0.000238	0.004470
def-kamil_cpu	kamil	1	0.000031	46106596622	0.000001	0.000238	0.004470
no_rac_gpu		75	0.003006	842007112518017	0.014898	0.014898	0.032224
ras_basic_gpu		75	0.002967	842007112518017	0.014898	0.014898	0.030781
cc-debug_gpu		1	0.000001	37224	0.000000	0.000004	0.030781
cc-debug_gpu	kamil	1	0.000000	0	0.000000	0.000000	0.030781
def-kamil_gpu		1	0.000001	37555979258	0.000001	0.000004	0.016416
def-kamil_gpu	kamil	1	0.000001	37555979258	0.000001	0.000004	0.016416

# Group's Status: "sshare -l"

```
[kamil@cedar5 workshop_test]$ sshare -l | egrep "(--|Account|^root|no_rac_|ras_b|cc-debug|kamil_)"
```

Account	User	RawShares	NormShares	RawUsage	NormUsage	EffectvUsage	FairShare	LevelFS
root			0.000000	639083114320110		1.000000		
no_rac_cpu		1320	0.043194	404703982221822	0.633257	0.633257		0.068209
ras_basic_cpu		1320	0.999243	404703982221822	0.633257	1.000000		0.999243
cc-debug_cpu		1	0.000236	1273287234	0.000002	0.000003		75.104409
cc-debug_cpu	kamil	1	0.004386	0	0.000000	0.000000	0.026537	inf
def-kamil_cpu		1	0.000236	0	0.000000	0.000000		inf
def-kamil_cpu	kamil	1	1.000000	0	0.000000	0.000000	0.486678	inf
no_rac_gpu		65	0.002127	6883285083841	0.010771	0.010771		0.197479
ras_basic_gpu		65	0.984848	6883285083841	0.010771	1.000000		0.984848
cc-debug_gpu		1	0.000236	12668	0.000000	0.000000		128389.386733
cc-debug_gpu	kamil	1	0.004386	0	0.000000	0.000000	0.508693	inf
def-kamil_gpu		1	0.000236	0	0.000000	0.000000		inf
def-kamil_gpu	kamil	1	1.000000	0	0.000000	0.000000	0.973463	inf



# Priority

## sprio -n

```
kamil@cedar5 test]$ sprio | head
```

```
[kamil@cedar5 workshop_test]$ sprio -n
```

JOBID	PRIORITY	AGE	FAIRSHARE	PARTITION	TRES
130976	0.00000165	1.0000000	0.0000000	0.2500000	cpu=0.17,mem=0.10
167003	0.00000143	1.0000000	0.0000000	0.5000000	cpu=0.13,mem=0.03
195802	0.00116324	1.0000000	0.9983542	0.3333333	cpu=0.03,mem=0.00,gr
195804	0.00116324	1.0000000	0.9983542	0.3333333	cpu=0.03,mem=0.00,gr
195807	0.00116324	1.0000000	0.9983542	0.3333333	cpu=0.03,mem=0.00,gr
195809	0.00116324	1.0000000	0.9983542	0.3333333	cpu=0.03,mem=0.00,gr
195810	0.00116324	1.0000000	0.9983542	0.3333333	cpu=0.03,mem=0.00,gr

# Priority sprio

JOBID	PRIORITY	AGE	FAIRSHARE	PARTITION	TRES
130976	7088	2500	0	625	cpu=2526,mem=1437
167003	6150	2500	0	1250	cpu=2008,mem=392
195802	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195809	4996086	2500	4991771	833	cpu=469,mem=45,gres/
195810	4996086	2500	4991771	833	cpu=469,mem=45,gres/
205281	8206	2500	0	625	cpu=1875,mem=1800,gr
205290	6408	2500	0	625	cpu=1875,mem=2,gres/
544814	23534	1741	21571	208	cpu=13,mem=2
544815	23534	1741	21571	208	cpu=13,mem=2
617580	24194	373	22768	1042	cpu=10,mem=2
617581	24194	373	22768	1042	cpu=10,mem=2

# Multiple allocations/accounting groups

- Occurs when group gets a RAC (Resource Allocation Committee ) allocation and therefore a new allocation that becomes the default allocation.
- Occurs when a user is part of multiple Compute Canada research groups. One can select the default allocation, even a default allocation per cluster and send an email to [support@westgrid.ca](mailto:support@westgrid.ca).
- In order to specify a accounting group to charge and figure out the priority use the following example in your job submission script.
  - **#SBATCH --account=accounting\_group\_name**

# Allocations

- What does an allocation usually mean?
  - If you request average resources continually through the time period and run jobs, you are guaranteed to get at least your allocated resources over the time period (year).
- What if I have not applied for an allocation?
  - you have a default allocation

# Allocations

- It is impossible for an allocation to be defined as: “Any time you ask for the resources allocated you will receive them”.
  - If 2 users are given 50% of a cluster each, and both don’t start running jobs until the 6th month they both cannot get the same cluster
- Unless an extraordinary situation exist allocation will not mean that the specified resources are available sitting idle.
  - Funding agencies don’t like to see resources sitting idle
  - An example of a extraordinary situation would be an Tsunami warning center which may need to have an allocation sitting idle so that when a earthquake occurs they can compute which beaches get hit and concentrate first responder resources to save lives.

# Allocations in Compute Canada

- Compute Canada (CC) Resource Allocation Committee (RAC) is a Committee of researchers that evaluate proposed allocations on the basis of scientific merits and resources available. There is also a preliminary technical evaluation which evaluates the application on technical merits, job requirements. The technical evaluation reports its findings and recommendations to the RAC.
- Allocations are done yearly, the RAC call for proposals goes out every September.
- For more information see: [https://www.westgrid.ca/support/accounts/resource\\_allocations](https://www.westgrid.ca/support/accounts/resource_allocations)

# Getting information on you and your group

Command	What its used for
sacctmgr list Users USERS=<username>	List user and their default account (accounting group)
sacctmgr show user <username> withassoc	List user and their default account (accounting group) and shows more extensive information
sshare	Shows usage info for user usage and priority.
sshare -l	Shows even more info for user usage and priority.

Priority for your job

Compare it to other job

Fairshare target allocation to your group

Your groups usage by members

**BREAK FOR PRACTICE**

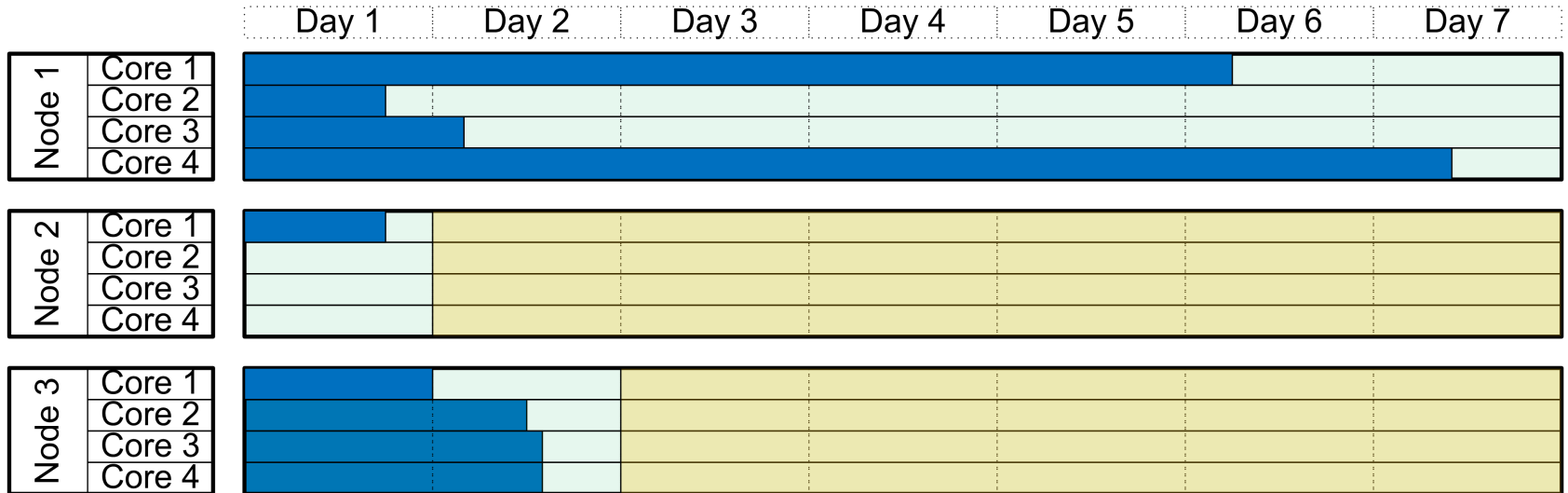


# Usage limits on a cluster

There are 2 types of usage limits:

- Usage limits that prevent the scheduling system from being overloaded.
- Usage limits that prevent a user from monopolizing the cluster
  - by starting jobs on all resources of a cluster which will run for a long period of time.
  - By starting jobs that last a very long time

# Reservations

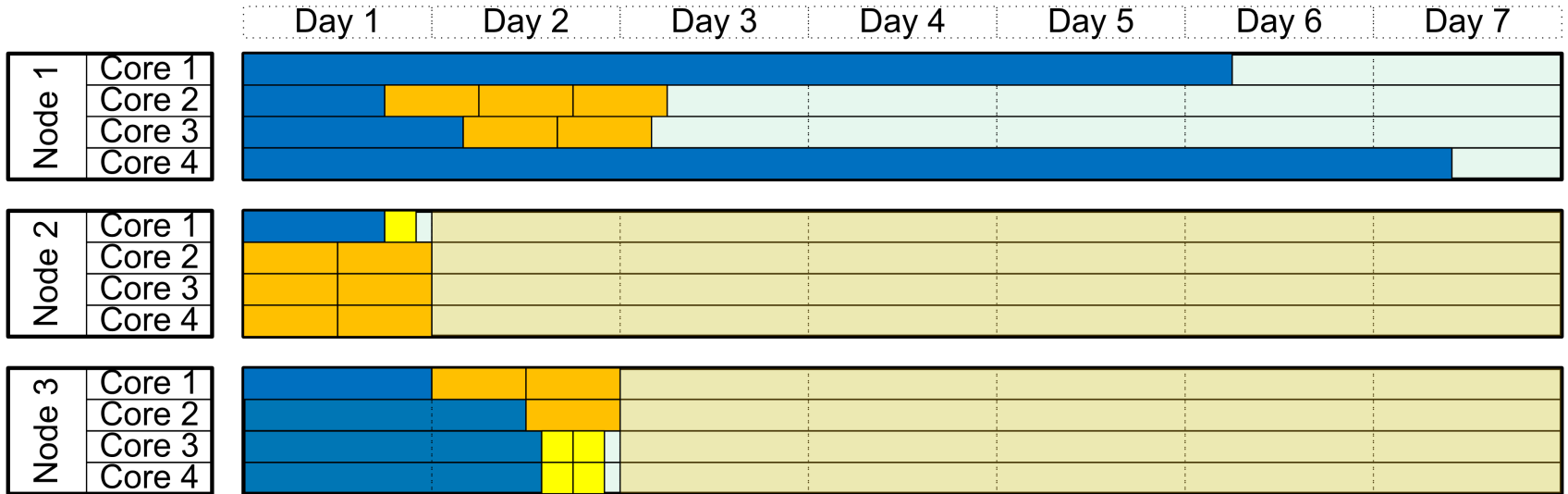


Special Reservation

# Reservations

- Used for many purposes
  - Used to schedule outages: Security patch that requires an reboot
  - Used to reserve resources for special occasions, such as a workshop
  - Each job also creates reservations
- One can see reservations on a cluster via “scontrol show reservations” command

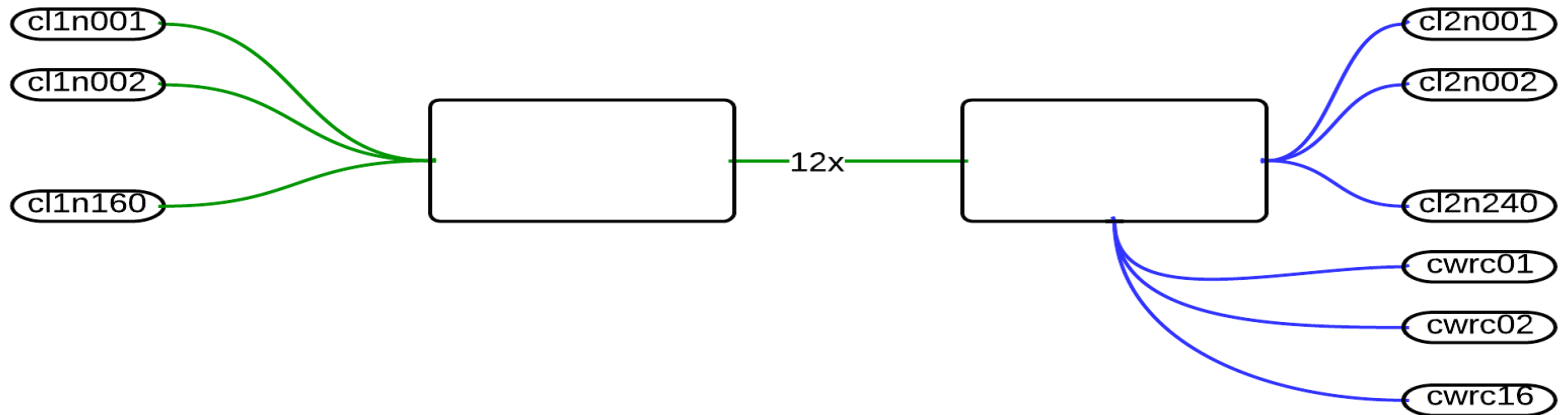
# Reservations and short serial jobs



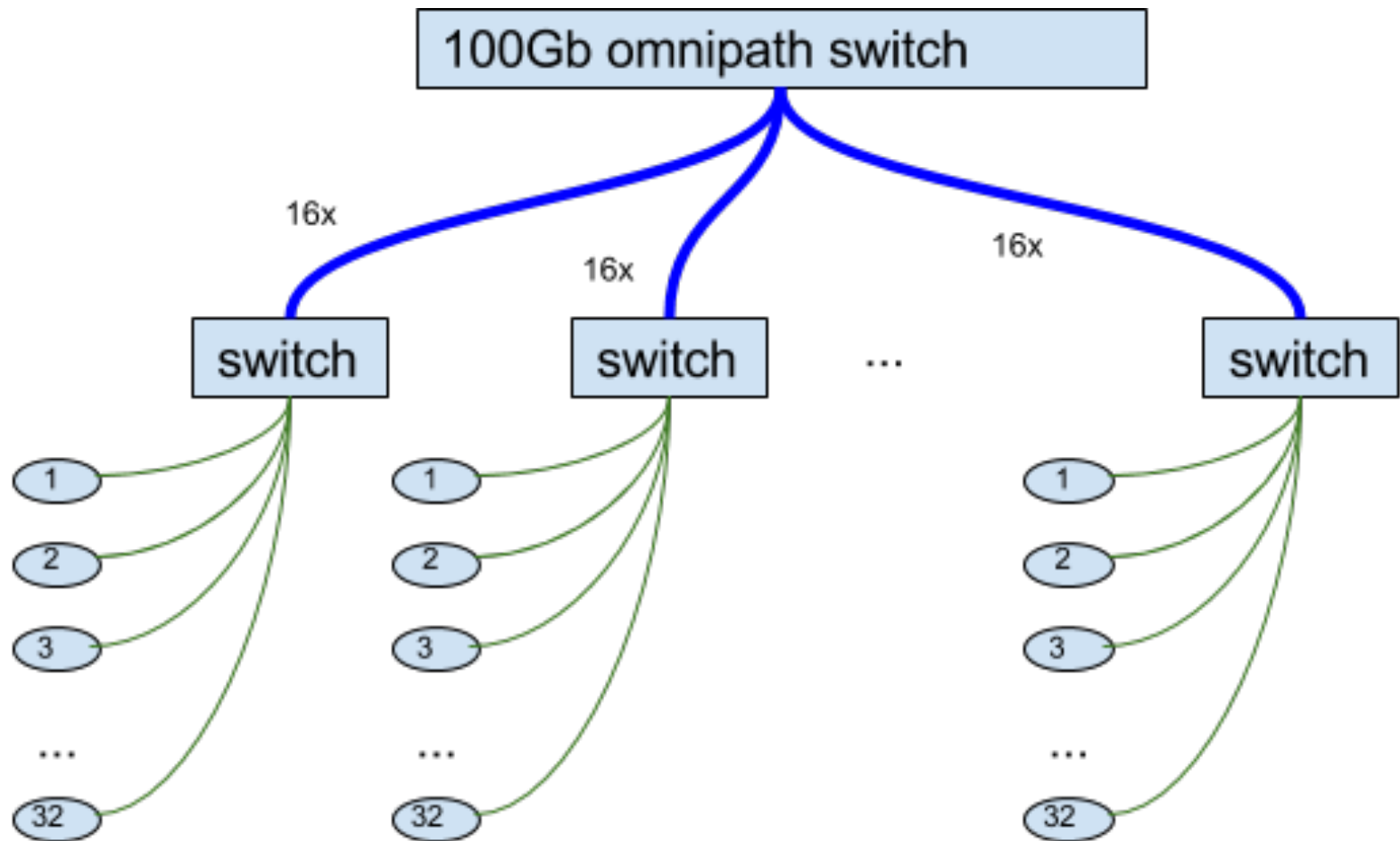
# Topology

- As more devices are added to a system the ability to have high bandwidth and low latency communication between every device to every other device becomes at first expensive and then impossible.
- This effect is true between cores on a chip, memory on a machine, chips on boards, gpus, as well as nodes in a cluster.
- The workaround is topology, only certain set resources are connected with high bandwidth, low latency, non blocking connections with each other, but the connection to other resources of lower bandwidth, higher latency, larger blocking factor.
- The result is that jobs running on certain sets of resources are faster than running on others, and the scheduling system needs to take this into account.
- This problem will be a much bigger in the future.

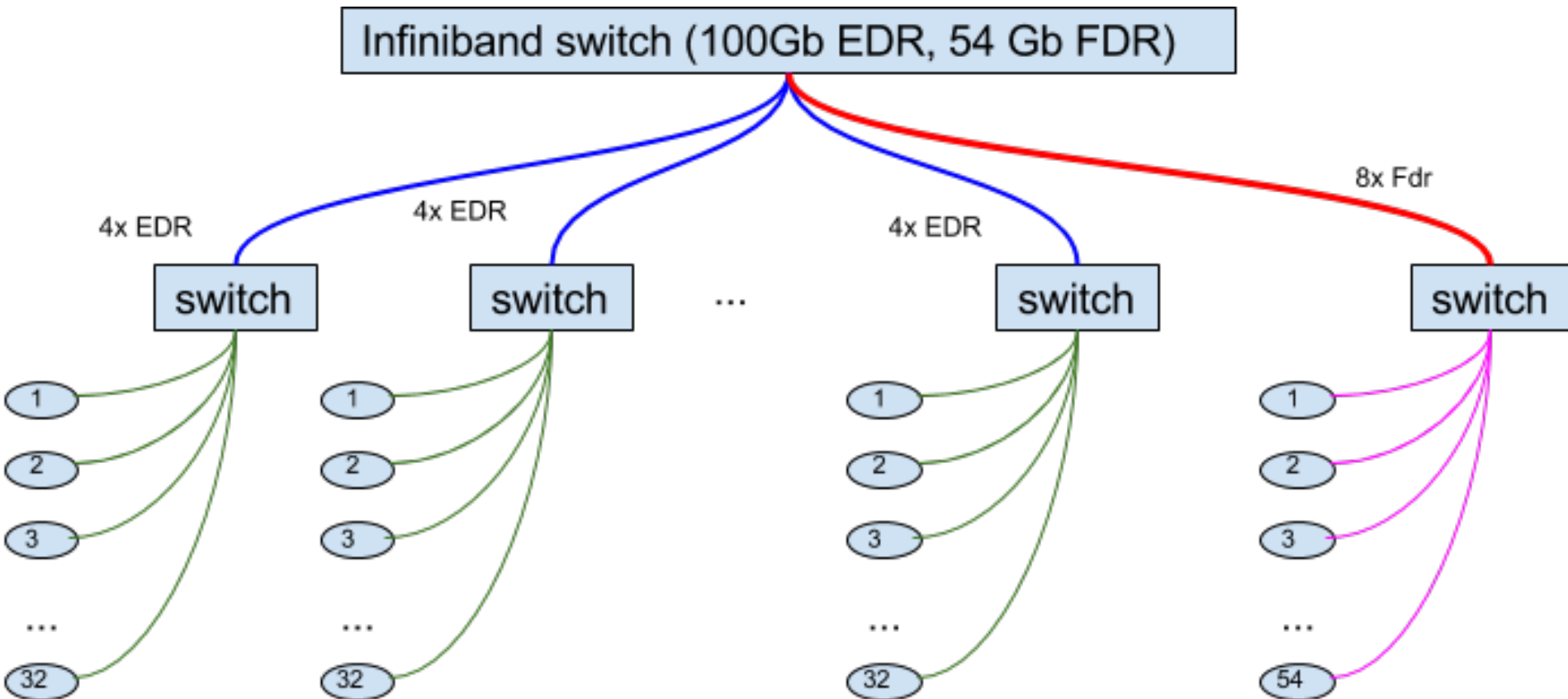
# Topology on older cluster



# Topology on Cedar

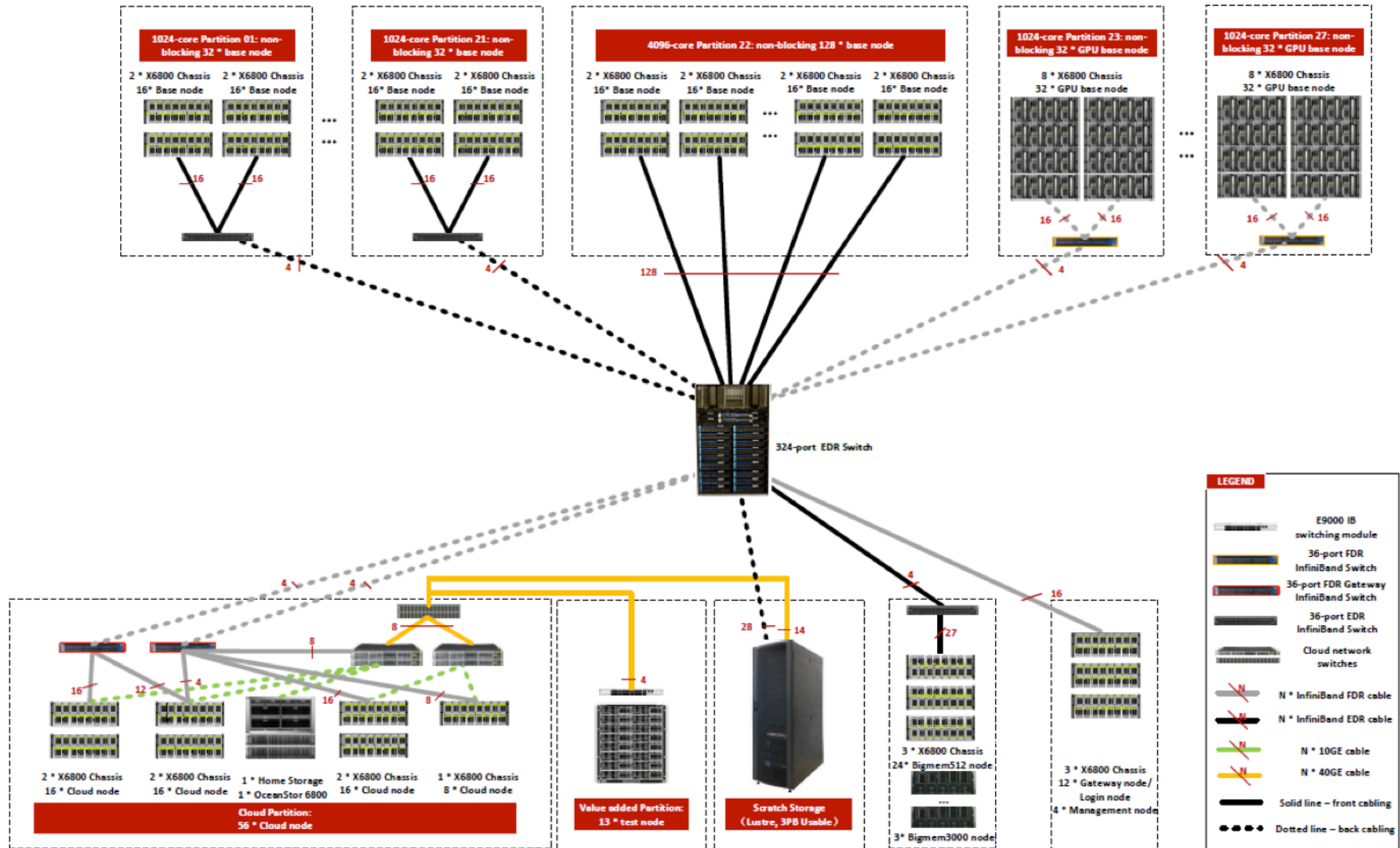


# Topology on Graham

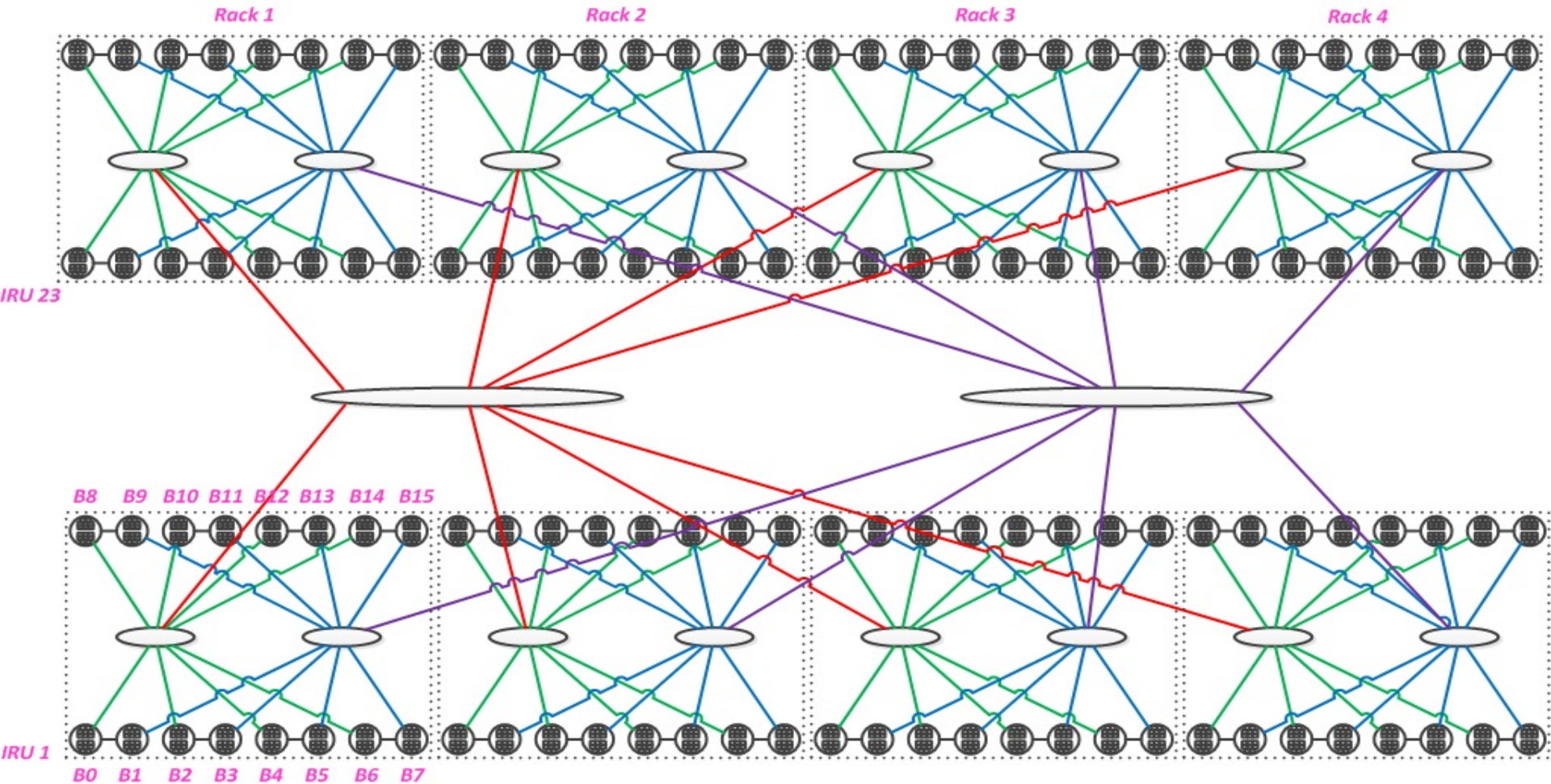




# Interconnect network on Graham










# Topology on Hungabee



Cost of on socket communication = 10  
 Cost of inter-socket communication  
 or to hub chip on blade += 3  
 Cost of going off blade += 20  
 Cost each black line += 7  
 Cost of each colored line += 7.5

	Socket
	Blade
	IRU

	10		$55 = (10 + 3 + 20) + 2 * (7.5) + 7$
	$13 = 10 + 3$		$63 = (10 + 3 + 20) + 4 * (7.5)$
	$40 = (10 + 3 + 20) + 7$		$70 = (10 + 3 + 20) + 4 * (7.5) + 7$
	$48 = (10 + 3 + 20) + 2 * (7.5)$		

# Topology on Hungabee

- Communication between cores and memory on hungabee's uv1000 compute node is faster and more abundant on adjacent connected resources than on the other side of the machine. The scheduling system needs to take this into account and schedule your jobs to runs on adjacent/connected resources.
- The topology of hungabee uv1000 machine is strange, odd even blade pairs, all blades in a chassis, all even and all odd blades are connected to each other more closely than other combinations.
- The topology results in strange effects, a job using 2 of 128 blades will stop a job requiring  $\frac{1}{2}$  of the machine (64 blades from running), but will not stop a 66 blade job from starting, the reverse is also true: a 64 blade job will stop a 2 blade job from starting but not a 3 blade job.
- The only way to know if your job should be starting but isn't is to take the "mdiag -n" or "jobinfo -n" output and compare it to topology diagram and see if there is enough empty resources, appropriately connected for your job to start.
- **Tip:** Don't have your jobs ask for  $\frac{1}{2}$  the machine, use less than  $\frac{1}{2}$  or slightly more, and it will be scheduled quicker.

# Getting information on your Cluster

# Sinfo -R

- Shows Nodes that are down and the reason why usually some error.

```
[kamil@cedar5 projects]$ sinfo -R | head -12
```

REASON	USER	TIMESTAMP	NODELIST
Not responding	root	2017-06-23T14:10:54	cdr[137-139,147,270]
batch job complete f	root	2017-08-20T05:36:07	cdr811
Not responding	slurm	2017-08-29T02:41:01	cdr119
Prolog error	root	2017-08-27T14:31:25	cdr47
batch job complete f	root	2017-08-23T01:36:00	cdr52
batch job complete f	root	2017-08-17T14:07:09	cdr[53,62]
Epilog error	root	2017-07-25T16:39:47	cdr61

# sinfo --states=idle

- Shows idle nodes and partitions (When a node is in multiple partitions it shows it multiple times)

```
kamil@cedar5 projects]$ sinfo --states=idle | head -15
```

```
PARTITION    AVAIL TIMELIMIT NODES STATE NODELIST
```

```
cpubase_interac  up 12:00:00  7 idle cdr[552,556,682,693,695-696,848]
```

```
cpubase_bycore_b1 up 3:00:00 17 idle  
cdr[358,362,365-367,369-374,377-379,381-382,384]
```

```
cpubase_bycore_b2 up 12:00:00  0 n/a
```

```
cpubase_bycore_b3 up 1-00:00:00  0 n/a
```

```
cpubase_bycore_b4 up 3-00:00:00  0 n/a
```

```
cpubase_bycore_b5 up 7-00:00:00  0 n/a
```

```
cpubase_bycore_b6 up 28-00:00:0  0 n/a
```

```
cpubase_bynode_b1* up 3:00:00 66 idle  
cdr[358,362,365-367,369-374,377-379,381-382,384,391,413,497,501,504,510,542,555,560,563,568,579,598,600,612,615,626,631,644,648,652,654,657,667,669,676,684,711,716-717,721,724-725,729,731-732,735,739,744,758,761,774,778,785,805-806,808,837,855]
```

# Partition Stats

(CC script)

Node type	Max walltime					
	3 hr	12 hr	24 hr	72 hr	168 hr	672 hr
Number of Queued Jobs by partition Type (by node:by core)						
Regular	1:15	2:31	2:145	11:187	86:69	3:2
Large Mem	0:0	0:0	0:0	0:0	0:1	0:1
GPU	0:1	0:526	10:10	0:0	189:4	0:0
Number of Running Jobs by partition Type (by node:by core)						
Regular	60:6	4:2	45:836	5:90	11:1065	1:4
Large Mem	0:0	0:0	0:0	0:0	0:0	1:0
GPU	0:20	2:10	13:2	0:0	0:0	0:3
Number of Idle nodes by partition Type (by node:by core)						
Regular	0:0	0:0	0:0	0:0	0:0	0:0
Large Mem	3:1	0:0	0:0	0:0	0:0	0:0
GPU	17:1	11:1	0:0	0:0	0:0	0:0
Total Number of nodes by partition Type (by node:by core)						
Regular	851:411	821:391	756:346	636:276	180:100	90:50
Large Mem	27:12	24:11	24:11	20:3	3:2	2:1
GPU	156:78	144:72	116:58	104:52	13:12	13:12

# Getting information on your Cluster

Command	What its used for
<code>sinfo --states=idle</code>	Show idle node on cluster
<code>sinfo -R</code>	Show down, drained and draining nodes and their reason
<code>sinfo --Node --long</code>	Show detailed node info.
<code>scontrol show reservation</code>	Shows reservations on the cluster
<code>partition-stats</code>	Compute Canada script to show jobs and nodes by partition

```
scontrol create reservation  
user=root starttime=now  
    duration=infinite  
flags=maint  
nodes=<nodeid>
```



Cluster information

**BREAK FOR PRACTICE**

# Why does my job not run?

- List of reasons your job is not running in order of probability.
  1. There is a problem with the job
  2. The Job is blocked
  3. Other jobs have greater priority
  4. Resources are not available
  5. There is a problem with the scheduling system or cluster.

# Common Problems

- The Job request more resources than are available on the system or node or practical to run on the system.
- ex)
  - You can request 10,000 cores on cedar
  - Request more than 3TB of RAM per node
  - Request 5 nodes each with 2TB per node

# Problem with my job

1. Is the Job blocked? “`squeue -u <user name>`”
  - Find out more? “`scontrol show jobid -dd <jobid>`”
2. Is the Job on hold? Are there dependencies?

# Is there a problem with my job?

3. What is my jobs priority? Compare it to other jobs on cluster run: “sprio”

If you have much lower priority find out why:

use: “sshare”

- Wait until priority improves over time.
- Ask fellow group members to run less.
- Ask for your professor to apply for a RAC allocation.

# Is there a problem with the cluster?

4. If you have high priority and your job is queued check to see if the resources are available
  - a. Use “partition-stats” to see if there are enough resources available on enough nodes to start your job. Check the WestGrid webpage to see if there is an outage scheduled.

# Is there a problem with cluster

## 5. Is there a reservation or system outage

- Check the Compute Canada webpage / MOTD on the system to see if there is an outage scheduled.
- Check for an reservation on the system “scontrol show reservation”

# Send email to `support@comptecanada.ca`

- Make sure you always include the following at the beginning of the email
  - Name of the cluster, jobid , userid
  - The location of the jobscript you submitted.
  - Any output or error of the job run.
  - Also make sure the name of the cluster is in the subject, ex: “job 123456 fails to run on the Cedar cluster”
- Brief but complete description of the problem.
- You should try to include the output of any commands like those described in the talk earlier. Please include any output of commands that you have run which convinced you there is a problem. A lot of these commands give the state of the job or cluster at the moment and this way we can analyze the situation as you saw it.



# Scheduling in the future

- Many more levels of topology
- Enforcing exclusivity with granularity
- Data movement, backups, recovery, latency, bandwidth, move job to data not data to job.
- Failure tolerant jobs and scheduling
- Power aware jobs and scheduling
- Scheduling provisioning of nodes
- Scheduling VMs and containers.
- Cloud /Grid scheduling including both batch jobs and services on the same system, virtual network management, all the points above in a integrated system

**QUESTIONS?**

# Upcoming ARC Training Sessions

<b>October 25</b> 10am - 11pm MDT	<b>Machine Learning Using Jupyter Notebooks on Graham</b>
<b>November 1</b> 11am – 1 pm MDT	<b>Introduction to Classical Molecular Dynamics Simulations</b>
<b>November 21</b> 11am – 1 pm MDT	<b>Exploring Containerization with Singularity</b>
<a href="https://www.westgrid.ca/events/westgrid-training-events">https://www.westgrid.ca/events/westgrid-training-events</a>	