# Intermediate SCC Usage

Research Computing Services

Katia Bulekova

BOSTON UNIVERSITY

# Learning Objectives

- Understanding Cluster Structure
  - Buy-in vs. Shared nodes
  - Hardware architecture
  - Resources request
- Learning how to retrieve and understand information about past jobs
- How to run parallel Jobs on the SCC
- Running multiple jobs on the cluster
  - Array jobs
  - Executing multiple scripts within a single job
- Executing dependent jobs

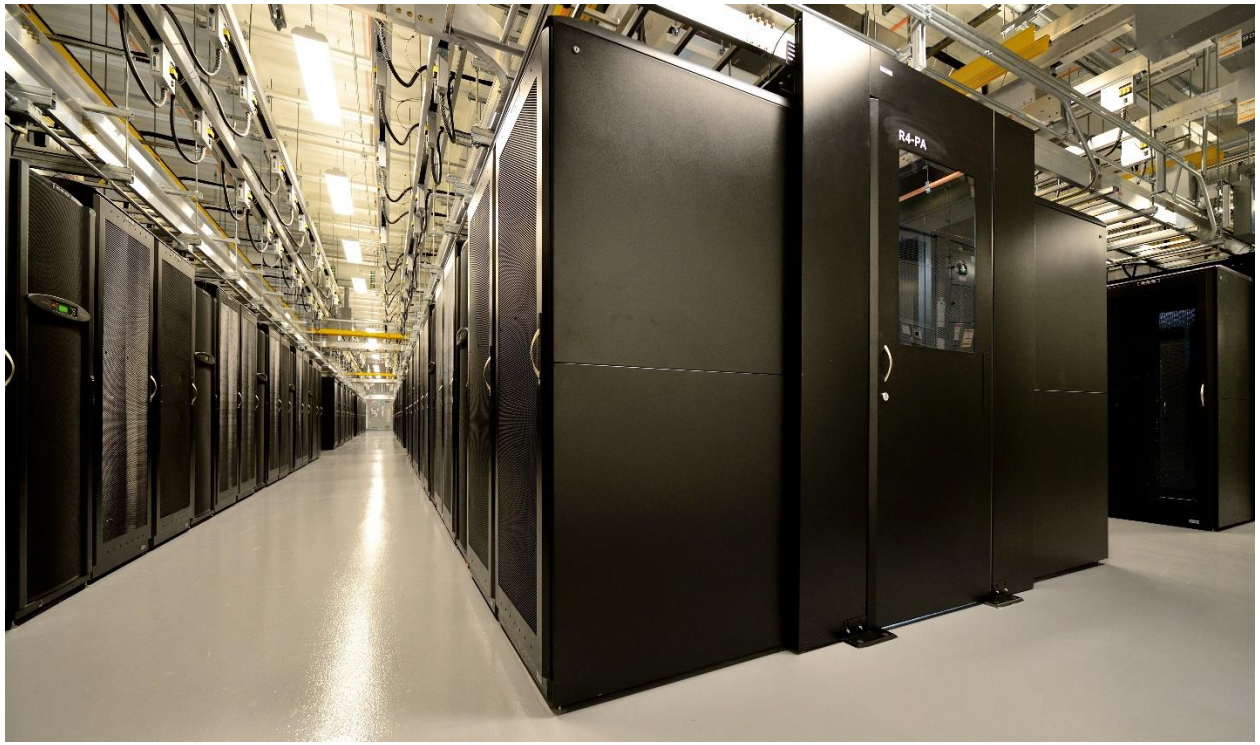
# Shared Computing Cluster

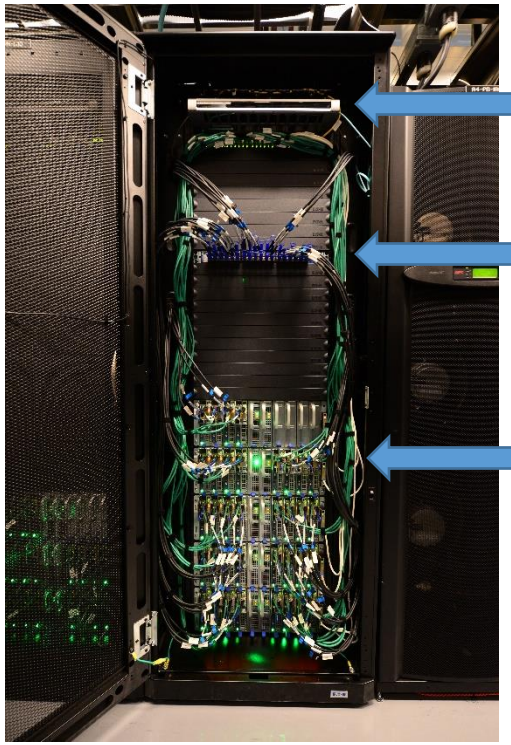
- Shared - transparent multi-user and multi-tasking environment

- Computing - heterogeneous environment:
  - interactive jobs
  - single processor and parallel jobs
  - graphics job

- Cluster - a set of connected via a fast local area network computers; job scheduler coordinates work loads on each node

# Shared Computing Cluster



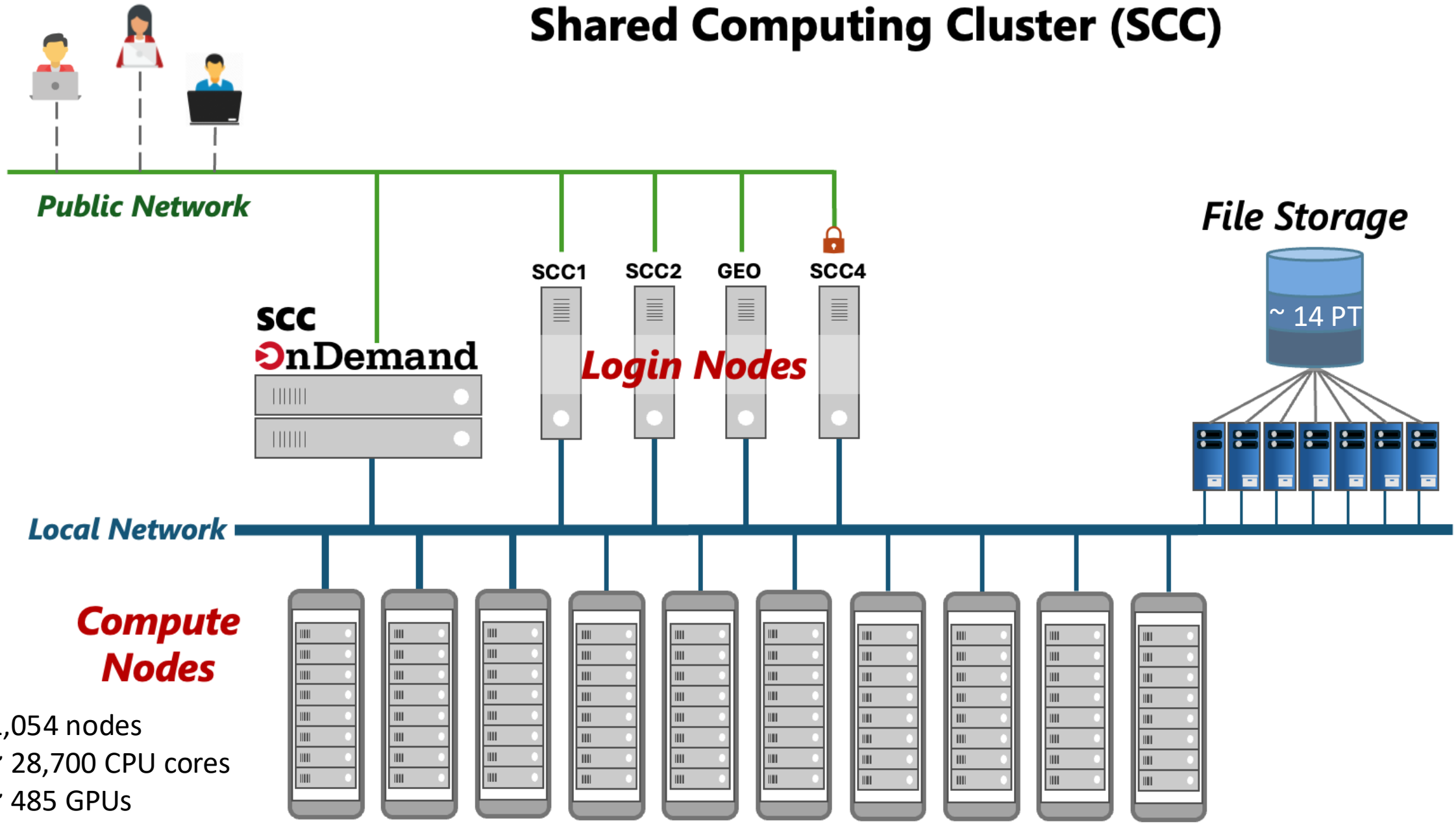
Ethernet

Infiniband

Compute Nodes

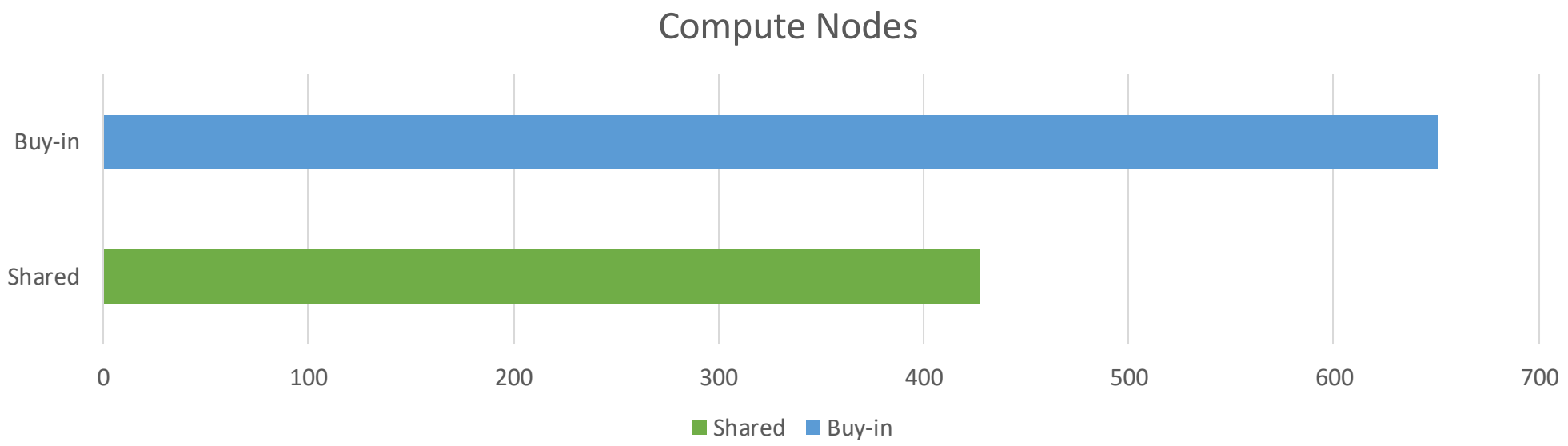
Rear View

# Shared Computing Cluster (SCC)




1,054 nodes
~ 28,700 CPU cores
~ 485 GPUs

# SCC buy-in resources




- All buy-in nodes have a hard limit of 12 hours for non-member jobs.
- Setting time limit for a job larger than 12 hours automatically excludes all buy-in nodes from the available resources

# SCC resources

- Processors: *Intel and AMD*
- CPU Architecture: *sandybridge, ivybridge, haswell, broadwell, knl, epyc, skylake, cascadelake, icelake, sapphirerapids*
- Ethernet connection: *1 or 10 Gbps*
- Infiniband: *EDR, FDR, QDR, HDR ( or none )*
- GPUs: *NVIDIA K40m, P100, V100, A100, A40, A6000, L40s etc. + GPUs for visualization*
- Number of cores: *8,16, 20, 28, 32, 36, 48, 64, 96*
- Memory (RAM): *128 GB – 1TB*
- Scratch Disk: *244GB – 886GB*

Technical Summary:

http://www.bu.edu/tech/support/research/computing-resources/tech-summary/

# SCC: batch jobs

Script organization:

**Script interpreter**

**Scheduler Directives**

**Commands to execute**

```
#!/bin/bash -l

#Time limit
#$ -l h_rt=12:00:00

#Project name
#$ -P krcs

#Send email-report at the end of the job
#$ -m e

#Job name
#$ -N myjob

#Load modules:
module load python/3.12.4

#Run the program
python myscript.py
```

# SCC: batch jobs

Script organization:

Execute login shell
(for proper interpretation of the module commands)

Script interpreter

Scheduler Directives

Commands to execute

```
#!/bin/bash -l

#Time limit
#$ -l h_rt=12:00:00

#Project name
#$ -P krcs

#Send email-report at the end of the job
#$ -m e

#Job name
#$ -N myjob

#Load modules:
module load python/3.12.4

#Run the program
python myscript.py
```

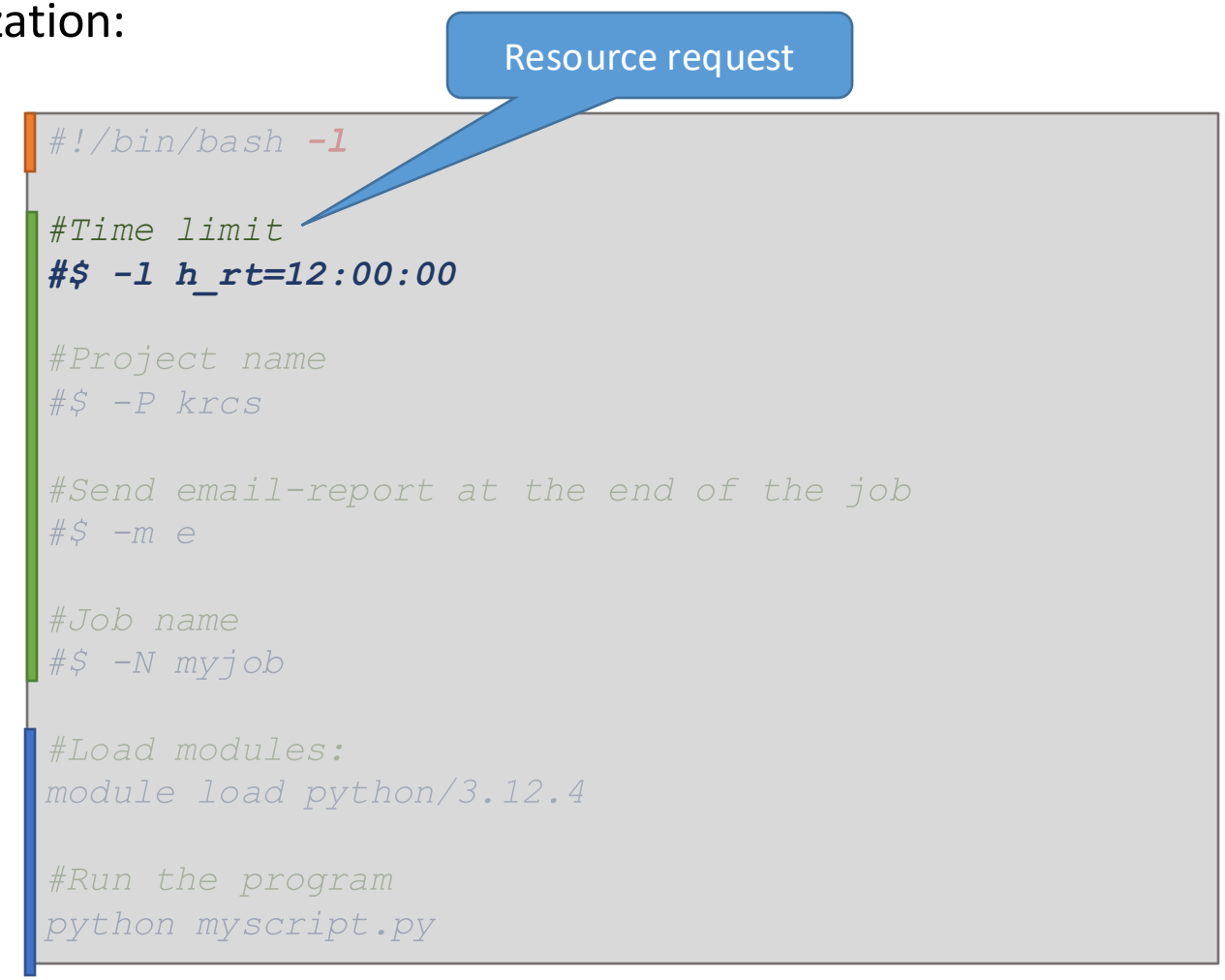
# SCC: batch jobs

Script organization:

Script interpreter

Scheduler Directives

Commands to execute



# SCC Resources

**All purpose** nodes:

can run single-processor jobs and parallel jobs (up to 720 hours)

**Whole node** queues (8, 16, 28, 32 and 36 cores):

only jobs that request a whole node will run on them (up to 240 hours)

**GPU** nodes:

only jobs requesting GPU(s) will run on these nodes (up to 48 hours)

**MPI** queues:

only for jobs requesting multiple nodes (up to 120 hours)

**VirtualGL** nodes:

for interactive graphics jobs (up to 48 hours)

# Request resources: single node parallelization

*An example of resource request for multiple cores (to run parallel codes):*

```
#$ -pe omp 4
```

Recommended values to select multiple CPU cores:

4, 8, 16, 28, 32, 36 # *shared* nodes

4, 8, 16, 20, 28, 32 # *buy-in* nodes

# Request resources: multi-node parallelization

*An example of resource request for an MPI job:*

```
#$ -pe mpi_28_tasks_per_node 56
```

3 MPI queues:

- two queues with 28-core nodes  (896 total cores limit)
- 64-core nodes ( 1024 total cores limit)

# Request resources: GPU jobs

*An example of resource request for a GPU job:*

```
#$ -l gpus=1
```

All shared GPU nodes have 2 or 4 GPUs per node

Buy-in queues have 1, 2, 4, 5, 8, and 10 GPUs per node

GPUs architecture: K40m, P100, V100, A100, A40, A6000, L40 (shared nodes)

**qgpus -v** # displays information about current GPUs on the SCC

# Request resources: GPU jobs

*Example of resource request for a GPU job, with additional restrictions:*

```
# GPU capability ( current selection: 3.5, 6.0, 7.0, 7.5, 8.0, 8.6, 8.9)
#$ -l gpus=1
#$ -l gpu_c=6.0     # request GPU with at least 6.0 compute capability
#$ -l gpu_memory=40G
# GPU type ( see qgpus output for current selection)
#$ -l gpus=1
#$ -l gpu_type=P100     # request specific GPU type
```

# Hardware Architecture

- Processors: *Intel and AMD*
- CPU Architecture: *sandybridge, ivybridge, haswell, broadwell, knl, epyc skylake, cascadelake, icelake, sapphirerapids*

There are 3 compilers available on the SCC: **GNU** (gcc/llvm), **PGI** and **Intel**

A program compiled with options that optimize performance for a newer CPU architecture may be unable to run on older compute nodes.

```
#$ -l avx
#$ -l avx2
```

https://www.bu.edu/tech/support/research/software-and-programming/programming/compilers/

# Resource request: Memory

Script interpreter

Scheduler Directives

```
#!/bin/bash -l

#Time limit
#$ -l h_rt=12:00:00

#Memory request
#$ -pe omp 28
#$ -l mem_per_core=8GB
```

https://www.bu.edu/tech/support/research/system-usage/running-jobs/resources-jobs/#memory
https://www.bu.edu/tech/support/research/system-usage/running-jobs/batch-script-examples/#MEMORY

# SCC: Job Memory usage

Checking the status of a batch job

```
scc1 % qstat -u <userID>
```

List only running jobs

```
scc1 % qstat -u <userID> -s r
```

Get job information:

```
scc1 % qstat -j <jobID>
```

# SCC: Job Memory usage


**job_number:** **596557**
exec_file: job_scripts/596557
submission_time: Mon Sep 11 10:11:04 2017
owner: ktrn
. . .
**sge_o_workdir:** **/projectnb/krcs/projects/**
sge_o_host: scc4
account: sge
cwd: /projectnb/krcs/projects/chamongrp
**. . .**
**hard resource_list:** **no_gpu=TRUE,h_rt=172800**
soft resource_list: buyin=TRUE
env_list: PATH=/usr/java/default/jre/bin:/usr/java/default/bin
**script_file:** **job.qsub**
**parallel environment: omp16 range: 16**
project: krcs

**usage 1:** cpu=00:13:38, mem=813.90147 GBs, io=0.01024, vmem=1.013G, **maxvmem=1.013G**
scheduling info: (Collecting of scheduler job information is turned off)

# SCC: Memory and cpu core usage

1. Login to the compute node

```
scc1 % ssh scc-ca1
```

2. Run *top* command

```
scc1 % top -u <userID>
```

```
scc-c01 ~ % top -u koleinik
top - 10:20:02 up 16 days, 23:48,  1 user,  load average: 11.93, 11.64, 6.25
Tasks: 463 total,   2 running, 461 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  0.1%sy,  0.0%ni, 97.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  131914288k total, 80273064k used, 51641224k free,   447720k buffers
Swap:  8388604k total,        0k used,  8388604k free, 77855288k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
53248 koleinik  20   0 1028m  22m 1020 R 1597.5  0.0 126:02.65 sim
53469 koleinik  20   0 13392 1444  852 R  1.9  0.0   0:00.01 top
53216 koleinik  20   0  9192 1440 1192 S  0.0  0.0   0:00.01 596557
53367 koleinik  20   0 92876 1800  832 S  0.0  0.0   0:00.00 sshd
53368 koleinik  20   0  9676 1828 1376 S  0.0  0.0   0:00.01 bash
```

Top command will give you a listing of the processes running as well as memory an CPU usage

3. Exit from the compute node

```
scc1 % exit
```

# SCC: completed jobs report (*qacct*)

*qacct* - query the accounting system

```
scc1 % qacct -j 596557
```

query the job by ID

```
scc1 % qacct -j -d 3 -o ktrn
```

query the job by the time of execution

# SCC: completed jobs report (*qacct*)

**qname p100**
**hostname scc-c11.scc.bu.edu**
group scv
owner ktrn
project krcs
jobname myjob
**jobnumber 551947**
qsub_time Wed Sep 6 20:08:56 2017
**start_time Wed Sep 6 20:09:37 2017**
**end_time Wed Sep 6 23:32:29 2017**
granted_pe NONE
**slots 1**
**failed 0**
**exit_status 0**
cpu 11232.780
mem 611514.460
io 14.138
iow 0.000
**maxvmem 71.494G**
arid undefined

# SCC: Array jobs

An array job executes independent copy of the same job script. The number of tasks to be executed is set using *-t* option to the qsub command, .i.e:

The script below will submit an array job consisting of 10 tasks, numbered from 1 to 10. The batch system sets up ***SGE_TASK_ID*** environment variable which can be used inside the script to pass the task ID to the program:

```
#!/bin/bash -l

#$ -P myproject
#$ -l h_rt=12:00:00
#$ -N myjob
#$ -t 1-10

module python3/3.10.12
python my_script.py $SGE_TASK_ID
```

# SCC: Environment variables

SGE automatically sets several variables, and you can also leverage standard Linux environment variables to control your job's execution.

**SGE-Specific Variables**

$JOB_ID: The unique ID assigned to your job by SGE. Essential for monitoring and interacting with a specific job.

$JOB_NAME: The name of your job, as specified with the -N option in qsub.

$NSLOTS: The number of slots (CPUs) allocated to your job across all hosts.

$SGE_TASK_ID: Task ID

**General Linux Variables**

$PATH: Defines the directories where the shell looks for executable commands.

$HOME: The user's home directory.

$TMPDIR: A temporary directory that can be used by jobs to write temporary files. Using this can improve performance by reducing network I/O and prevent issues with shared storage.

# SCC: Accessing Environment variables in code

Python:

```
import os
num_cores_requested = int( os.getenv('NSLOTS') )
```

R:

```
num.cores <- as.numeric(Sys.getenv('NSLOTS'))
```

MATLAB:

```
num_cores= str2num( getenv('NSLOTS') )
```

# SCC: Running short scripts (Python example)

```
#!/bin/bash -l

module load python3/3.10.12

# Run same program for different parameters many times:
for i in `seq 0 4`;
do
        python short.py $i outfile_${i}.txt
done
```

# SCC: Running short scripts (R example)

```
#!/bin/bash -l

module load R/4.4.3

# Run same program for different parameters many times:
for i in `seq 0 4`;
do
       Rscript short_r.R $i outfile_${i}.txt
done
```

# SCC: Job dependency

Some jobs may be required to run in a specific order. For this application, the job dependency can be controlled using "-hold_jid" option:

```
scc1 % qsub -N job1 script1
scc1 % qsub -N job2 -hold_jid job1 script2
scc1 % qsub -N job3 -hold_jid job2 script3
```

A job might need to wait until the remaining jobs in the group have completed (aka post-processing).

In this example, lastjob won't start until job1, job2, and job3 have completed.

```
scc1% qsub -N job1 script1
scc1% qsub -N job2 script2
scc1% qsub -N job3 script3
scc % qsub -N lastJob -hold_jid "job*" script4
```

https://www.bu.edu/tech/support/research/system-usage/running-jobs/advanced-batch/#depend

# SCC: Links

Research Computing website: http://www.bu.edu/tech/support/research/

RCS software: http://sccsvc.bu.edu/software/

RCS examples: http://rcs.bu.edu/examples/

RCS Tutorial Evaluation: http://scv.bu.edu/survey/tutorial_evaluation.html

Please contact us at help@scc.bu.edu if you have any problem or question