# Introduction to Cluster Computing:

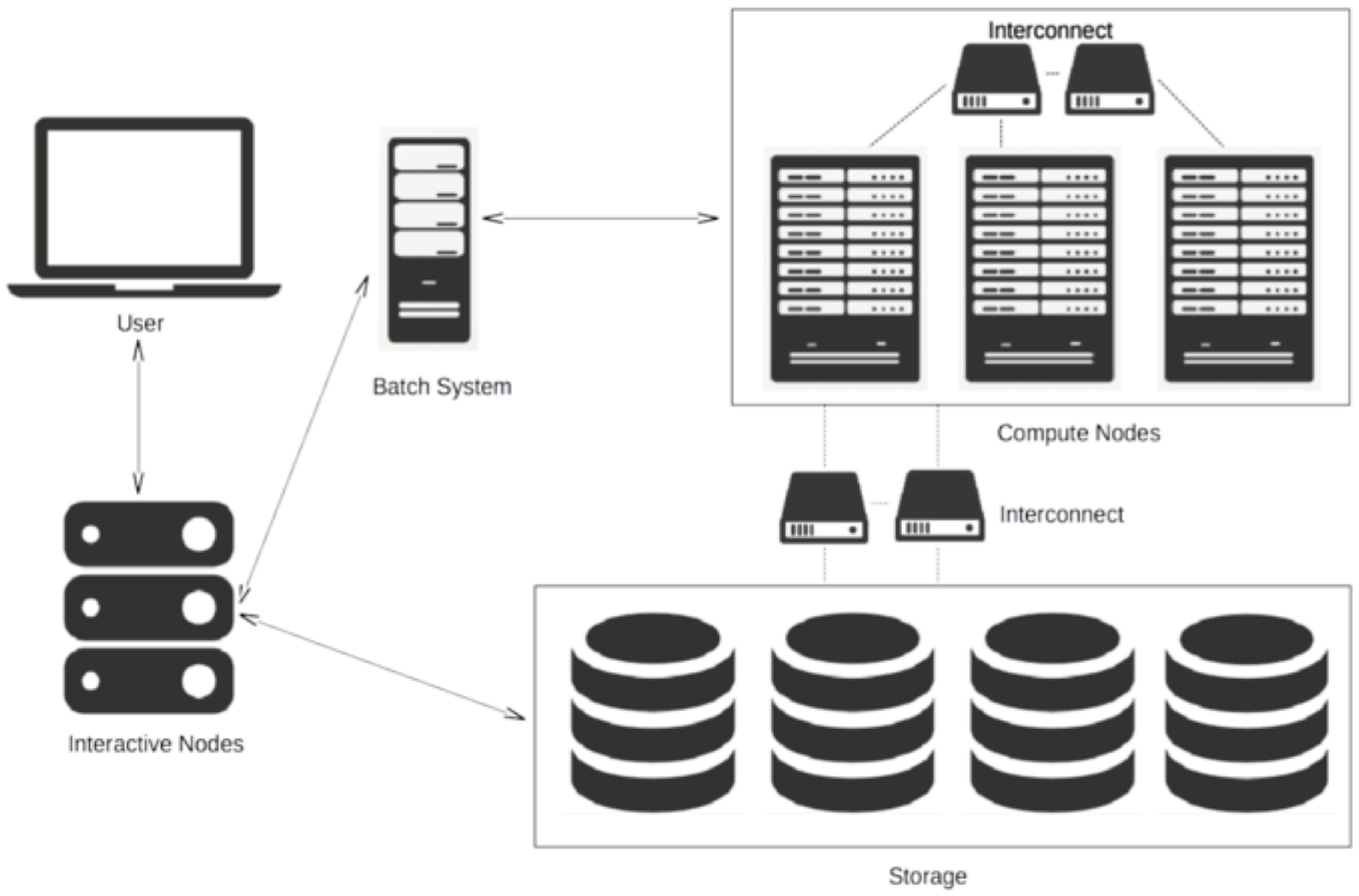
## Linux, shell scripting, queuing systems, cluster architecture

Instructor: Dr. Peggy Lindner (plindner@uh.edu)

Lecture 4 (Compiling & Schedulers)

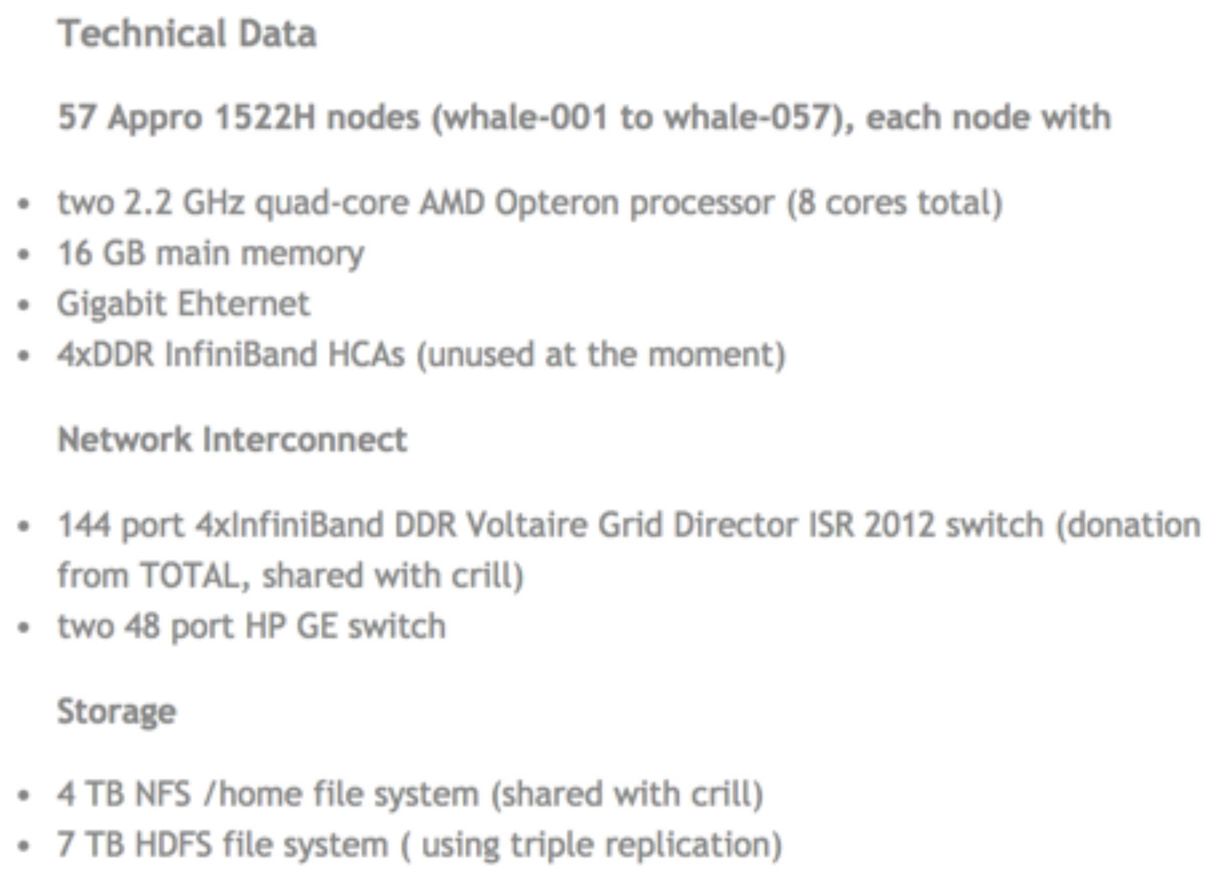
UNIVERSITY of HOUSTON
CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

# Compute Cluster



# Whale Specifications

- http://pstl.cs.uh.edu/resources/whale



**Technical Data**

**57 Appro 1522H nodes (whale-001 to whale-057), each node with**

- two 2.2 GHz quad-core AMD Opteron processor (8 cores total)
- 16 GB main memory
- Gigabit Ehternet
- 4xDDR InfiniBand HCAs (unused at the moment)

**Network Interconnect**

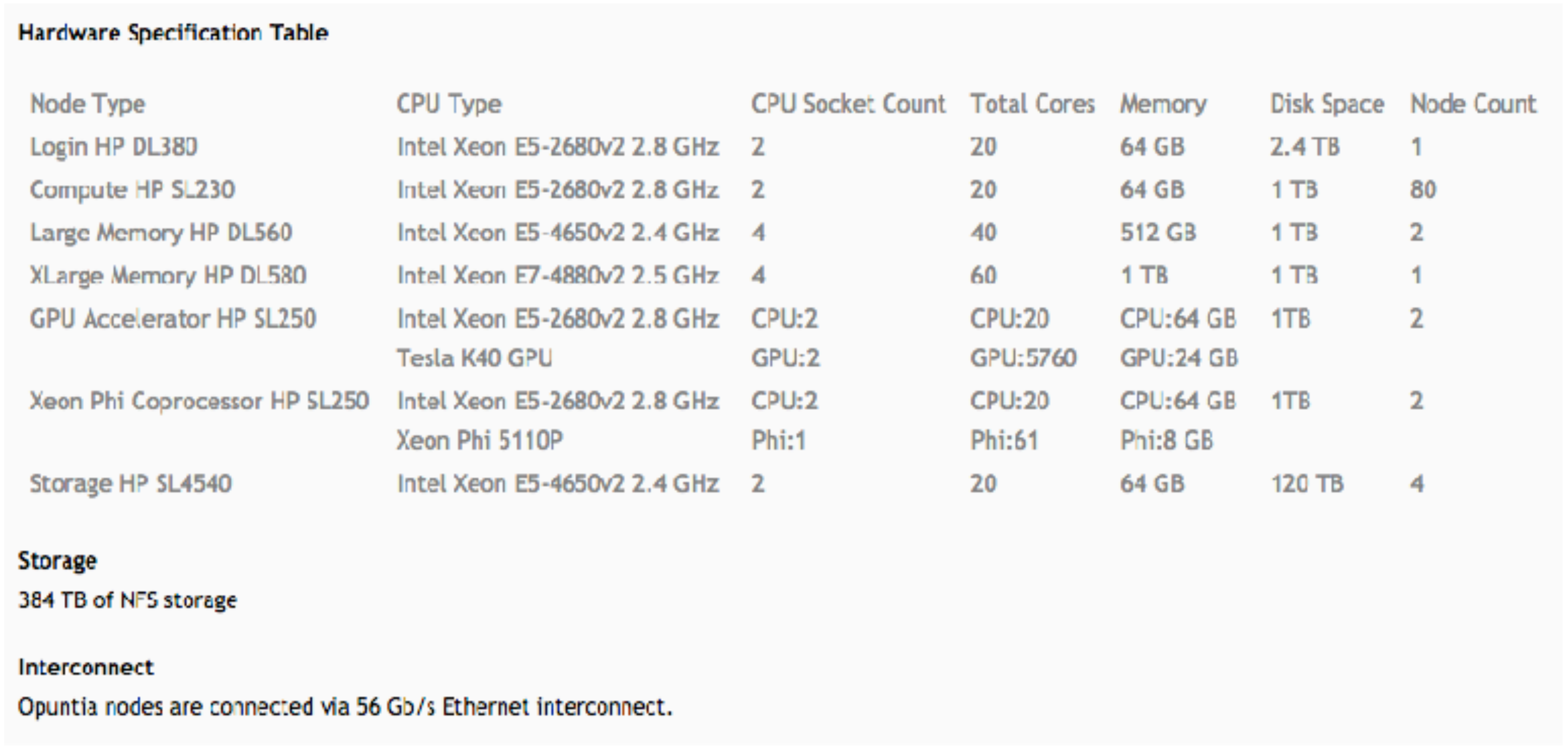
- 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL, shared with crill)
- two 48 port HP GE switch

**Storage**

- 4 TB NFS /home file system (shared with crill)
- 7 TB HDFS file system ( using triple replication)

# Opuntia Specifications

- https://www.cacds.uh.edu/researchresources/hpc/opuntia/



**Hardware Specification Table**

| Node Type | CPU Type | CPU Socket Count | Total Cores | Memory | Disk Space | Node Count |
|---|---|---|---|---|---|---|
| Login HP DL380 | Intel Xeon E5-2680v2 2.8 GHz | 2 | 20 | 64 GB | 2.4 TB | 1 |
| Compute HP SL230 | Intel Xeon E5-2680v2 2.8 GHz | 2 | 20 | 64 GB | 1 TB | 80 |
| Large Memory HP DL560 | Intel Xeon E5-4650v2 2.4 GHz | 4 | 40 | 512 GB | 1 TB | 2 |
| XLarge Memory HP DL580 | Intel Xeon E7-4880v2 2.5 GHz | 4 | 60 | 1 TB | 1 TB | 1 |
| GPU Accelerator HP SL250 | Intel Xeon E5-2680v2 2.8 GHz<br>Tesla K40 GPU | CPU:2<br>GPU:2 | CPU:20<br>GPU:5760 | CPU:64 GB<br>GPU:24 GB | 1TB | 2 |
| Xeon Phi Coprocessor HP SL250 | Intel Xeon E5-2680v2 2.8 GHz<br>Xeon Phi 5110P | CPU:2<br>Phi:1 | CPU:20<br>Phi:61 | CPU:64 GB<br>Phi:8 GB | 1TB | 2 |
| Storage HP SL4540 | Intel Xeon E5-4650v2 2.4 GHz | 2 | 20 | 64 GB | 120 TB | 4 |

**Storage**

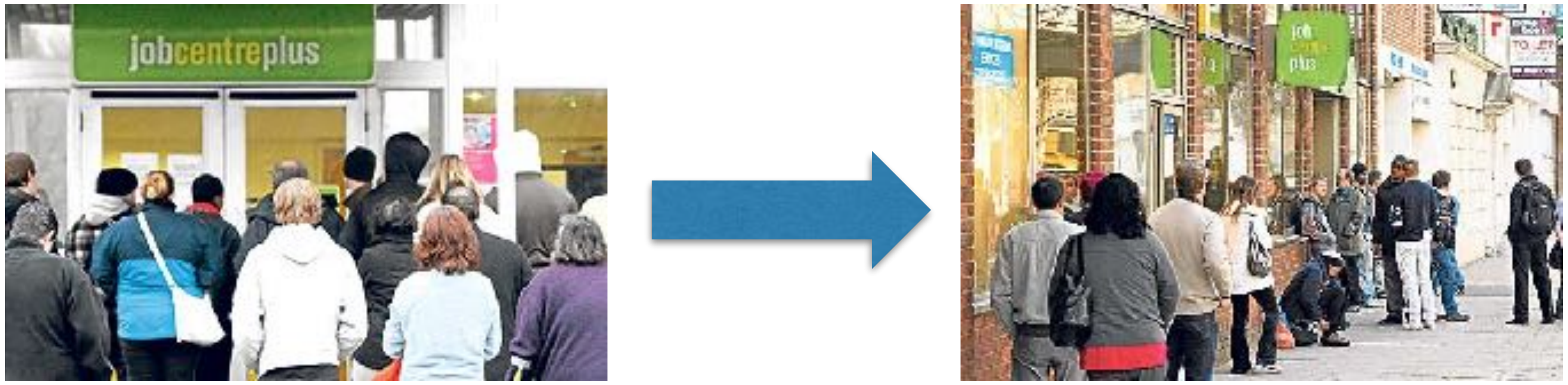
384 TB of NFS storage

**Interconnect**

Opuntia nodes are connected via 56 Gb/s Ethernet interconnect.

# The Queue System (SLURM)

- Simple Linux Utility for Resource Management (aka SLURM) is the resource management service used e.g. on Opuntia Cluster



- Enables you to make more efficient use of your time through scripting computational tasks

- SLURM takes care of running tasks and returns the results

- If the cluster is full, SLURM holds your tasks and runs them when the resources are available

- SLURM ensures fair sharing of cluster resources (policy enforcement)

    - SLURM ensures optimal/efficient use of available resources

UNIVERSITY of **HOUSTON**
CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

## Example Whale cluster

- The clusters consist of a login node (whale.cs.uh.edu) and a number of compute nodes that are configured via units called ``partitions''; e.g. in the whale cluster, the 8-core Opteron Nodes are part of a partition called ``whale``.

- The crill and the whale clusters share home directories, but are otherwise separate. The only access method to both cluster from the outside world is by using ssh.

- The login nodes are to be used for editing, compiling and similar activities. They are not to be used for running jobs such as parallel programs. Program runs are submitted through the SLURM scheduler.

# SLURM Commands

| Command(s) | Description |
|---|---|
| **squeue** | Check status of all jobs |
| **sbatch ./myjob** | Submit batch jobs |
| **squeue -u $USER** | Check status of just your jobs |
| **srun - -pty /bin/bash -l** | Submit an interactive job |
| **sun - - x11=first - -pty /bin/bash -l** | Submit an interactive job **with X11 support** |
| **scontrol hold jobID** | Put a job on hold (before it starts) |
| i.e. qhold 12345 | |
| **scontrol release jobID** | Release a job from hold status |
| **scancel #jobID** | Delete a job, running or not |

# Partition Configuration

```
$ scontrol show partition
  PartitionName=whale
     AllowGroups=users AllowAccounts=ALL AllowQos=ALL
     AllocNodes=ALL Default=YES QoS=N/A
     DefaultTime=NONE DisableRootJobs=NO
  ExclusiveUser=NO GraceTime=0 Hidden=NO
     MaxNodes=UNLIMITED MaxTime=01:00:00 MinNodes=1
  LLN=NO MaxCPUsPerNode=UNLIMITED
     Nodes=whale-0[00-49]
     PriorityJobFactor=1 PriorityTier=1 RootOnly=NO
  ReqResv=NO OverSubscribe=YES:2
     OverTimeLimit=NONE PreemptMode=OFF
     State=UP TotalCPUs=400 TotalNodes=50
  SelectTypeParameters=NONE
     DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

# Indivudual Node configuration

```
$ scontrol show node whale-054
 NodeName=whale-054 Arch=x86_64 CoresPerSocket=4
    CPUAlloc=0 CPUErr=0 CPUTot=8 CPULoad=0.01
    AvailableFeatures=(null)
    ActiveFeatures=(null)
    Gres=(null)
    NodeAddr=whale-054 NodeHostName=whale-054 Version=17.02
    OS=Linux RealMemory=1 AllocMem=0 FreeMem=6977 Sockets=2 Boards=1
    State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=10 Owner=N/A
 MCS_label=N/A
    Partitions=short,high
    BootTime=2016-08-12T10:01:20 SlurmdStartTime=2017-09-12T15:00:20
    CfgTRES=cpu=8,mem=1M
    AllocTRES=
    CapWatts=n/a
    CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
    ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

# Job Queues on Opuntia

| Queue Name | Description |
|---|---|
| short | Queue to run short jobs ( =< 4 hours) |
| medium | Queue to run medium jobs ( 1 week allowed) |
| long | Queue to run very long jobs (2 weeks allowed) |
| gpu | Queue to run gpu jobs on K 40 equipped nodes  (24 hrs allowed) |
| | |
| | |

Note phi and gpu queue are not available/functional on the surrogate cluster

# Queue Information

```
$ [plindner@opuntia ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3096739    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096740    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096741    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096742    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096743    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096744    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096745    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096746    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096747    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096748    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096749    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
           3096750    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096751    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096752    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096753    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096754    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096755    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096756    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096757    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096758    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
          3096759    medium Phon_4.2   acduke PD       0:00      4 (QOSMaxCpuPerUserLimit)
```

# Queue information

```
[plindner@opuntia ~]$ sinfo -all
Thu Sep 14 13:50:17 2017
PARTITION AVAIL  TIMELIMIT   JOB_SIZE ROOT OVERSUBS     GROUPS  NODES       STATE
NODELIST
batch*       up 14-00:00:0 1-infinite   no       NO        all      1    drained*
compute-2-27
batch*       up 14-00:00:0 1-infinite   no       NO        all      6       mixed
compute-0-1,compute-2-[9-10,18,32,39]
batch*       up 14-00:00:0 1-infinite   no       NO        all     64   allocated
compute-0-[2-19,23-35,37-39],compute-2-[0-8,11-17,19,21-26,28-30,35-38]
batch*       up 14-00:00:0 1-infinite   no       NO        all     11        idle
compute-0-[20-22,36],compute-2-[20,31,33-34,40-42]
short        up    4:00:00 1-infinite   no       NO        all      1    drained*
compute-2-27
short        up    4:00:00 1-infinite   no       NO        all      7       mixed
compute-0-[0-1],compute-2-[9-10,18,32,39]
short        up    4:00:00 1-infinite   no       NO        all     64   allocated
compute-0-[2-19,23-35,37-39],compute-2-[0-8,11-17,19,21-26,28-30,35-38]
short        up    4:00:00 1-infinite   no       NO        all     11        idle
compute-0-[20-22,36],compute-2-[20,31,33-34,40-42]
medium       up 7-00:00:00 1-infinite   no       NO        all      1    drained*
compute-2-27
medium       up 7-00:00:00 1-infinite   no       NO        all      6       mixed
compute-0-1,compute-2-[9-10,18,32,39]
medium       up 7-00:00:00 1-infinite   no       NO        all     64   allocated
compute-0-[2-19,23-35,37-39],compute-2-[0-8,11-17,19,21-26,28-30,35-38]
medium       up 7-00:00:00 1-infinite   no       NO        all     11        idle
compute-0-[20-22,36],compute-2-[20,31,33-34,40-42]
```

# HPC Software Available on Opuntia

- Proprietary & Open-Source Software:
  - Examples:
    - Amber, Autodock, Bwa, NwChem, Espresso, Octave, Matlab, Gromacs, Lammps, NAMD , R, NAG, MKL , ASE, GSL, OpenMPI, Intel, Gnu and PGI compilers ….
    - and much more…
  - Proprietary software may be limited to licensed users e.g. VASP
  - Open-source software is accessible to all

- PI could request for installation of additional software
  - PI's licensed propriety application can be installed too

- Software environment management via "**modules**"
  - Allows software to be accessed via an assigned module
  - Dynamic modification of a user's environment

# Computing Environment Setup Using Modules

- The module system is used to make software and related settings available easily

- Software environments can be loaded and unloaded dynamically

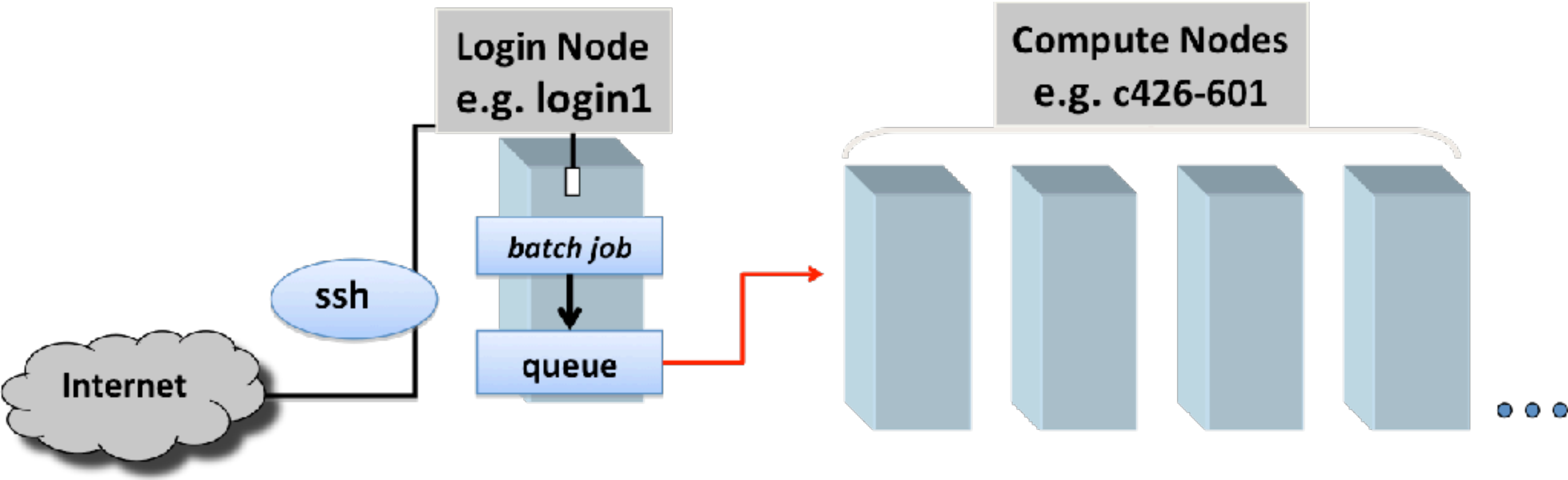
- To get a list of available software:
  ```
  module   avail
    (also useful: module list)
  ```

- To clear-out all added modules:
  ```
  module    purge
  ```

- To clear-out all  specific modules:
  ```
  module    rm  module1 module2 ….
  module   rm matlab
  ```

- Now add intel compiler and  MPI runtime (necessary for certain types of parallel programs)
  ```
  module    add   intelmpi
  ```

# Using sbatch with a Job Script



- Syntax/Format:
  - **sbatch [sbatch params] [job script file]**

- Examples:
  - **sbatch myjob**

# SLURM Batch Job Script File

- Very simple example:
- Serial job requesting 1 CPU, 2 GB memory, 60 hours of walltime

```
#!/bin/bash
#SBATCH  -J   jobname
#SBATCH  -o   jobname.o%j
#SBATCH –-mem=2gb
#SBATCH  -t 60:00:00

./my.program.exe [arguments for my program]
```

## Using Parallel Environment to Manage CPU Core Availability

- #SBATCH -N 1 -n 20 --use only 1 node and use a total of 20 processes, 20 CPUs per node.
- #SBATCH -N 2 -n 20 --use only 2 nodes and use a total of 20 processes, 10 CPUs per node.
- #SBATCH -N 2 -n 40 --use only 2 nodes and use a total of 40 processes, 20 CPUs per node.
- #SBATCH -N 4 -n 80 --use only 4 nodes and use a total of 80 processes, 20 CPUs per node.

# Sbatch Notification Parameters

- #SBATCH --mail-user=[e-mail address]
  - e-mail address can be a list of email addresses
  - separated by commas.

```
#SBATCH --mail-type=begin  # email me when the job starts

#SBATCH --mail-type=end    # email me when the job finishes

#SBATCH --mail-type=all    # email me when the job starts & finishes
```

```
#!/bin/bash
#SBATCH   -o  jobname.o%j  -J o  jobname
#SBATCH --mail-user=jerry@uh.edu
#SBATCH --mail-type=begin  # email me when the job starts
#SBATCH --mail-type=end    # email me when the job finishes

./my.program.exe [arguments for my program]
```

# Handling Output Files

- When a job is started it takes its job name from the script file that was submitted.
  - The standard output and error output are sent to file named jobname.o987349  (in your job directory )

-

- **#SBATCH  -o**  jobname.o%j name to be used for the job.

- **#SBATCH  -J**  jobname name to be used for the job.

# Using Sbatch with a Job Script

- Syntax/Format:
  - **sbatch [sbatch params] [script file]**

- More examples on using parameters at the command line:
  - `sbatch -J test1   myjob.sh`
  - `sbatch -N 2 -n 16 -o test2.out -J test2 myjob.sh`
  - `sbatch -t 02:00:00 --mem=4gb myjob.sh`
  - `sbatch -n 32 --mem=64gb   myjob.sh`
  - `sbatch -p gpu  -N 1 -n 20  my_gpu_job.sh`

- To see all available sbatch options, just run sbatch --help

# Compiling Programs

- Typically done on the headnode/login node
  - Except for large compilation tasks which should ran as an interactive job instead

- CPU programs (compiled to execute on central processing unit)
  - Serial Programs
  - Parallel  Programs

- GPU programs (compiled to execute on graphics processing unit)
  - GPGPU Programs

## Compiling Programs: CPU Serial Program

- Example 1 Program in file: gethostname.c

```
#include <stdio.h>
#include <sys/utsname.h>
int main ( )
{
     struct utsname uts;
   uname (&uts);
    printf ("Process on node %s.\n",
   uts.nodename);
     return 0;
}
```

- To compile:

1. choose compiler (e.g. module  add  intel)
2. compile

```
$ gcc  gethostname.c  -o   gethostname.exe
```

# SLURM script for serial program (1 CPU)

- Write your job script, e.g. job.gethostname.bash.slurm

```
#!/bin/bash
#SBATCH  -o gethost.out
#SBATCH  -t 00:25:00
#SBATCH  --mail-user=JohnDoe@gmail.com
#SBATCH --mail-type=all

### Run gethostname

time   ./gethostname.exe
```

- submit job into job queue

```
$ sbatch   job.gethostname.bash.slurm
```

## Compiling Programs: Parallel Programm MPI

- Example 1 Program in file: hello_world.mpi.c

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d"
           " out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

- To compile:
  1.choose compiler `module   add   openmpi`
  2.compile

```
$ mpicc hello_world.mpi.c -o hello_world.mpi.o
```

# SLURM script for parallel program

- Write your job script, e.g. job.hello_world_mpi.bash.slurm

```
#!/bin/bash
#SBATCH -n 2
#SBATCH -t 0:05:05 # Runtime in D-HH:MM
#SBATCH --mem=1000 # Memory pool for all cores (see also --mem-per-
cpu)
#SBATCH -o hostname.out
#SBATCH -e hostname.err
#SBATCH --mail-type=END # Type of email notification-
BEGIN,END,FAIL,ALL
#SBATCH --mail-user=jerry@uh.edu
# Email to which notifications will be sent

##set up your environment


module add openmpi

mpirun -np  4 ./hello_world.mpi.o > output.from.txt
```

- submit job into job queue

```
$ sbatch   job.gethostname.bash.slurm
```

# SLURM Script for GPGPU Program

Give me a node with this kind of gpu

Example GPU Job Script

```
#!/bin/bash
#SBATCH -n 1
#SBATCH -t 0:30:05  -p gpu
#SBATCH --mem=10gb
#SBATCH -o  GpuJob.out
#SBATCH --mail-type=all
#SBATCH --mail-user=jerry@uh.edu


    ##set up your environment

    module add cuda-toolkit
    time    ./sortkeys_basic_thrust.exe
```

Load CUDA environment

See job filename: job.gpu.bash or job.gpu.sort.bash.slurm

## SLURM Script for NAMD2 Job Parallel MPI

```
#!/bin/bash
#SBATCH -n 32 -N 4
#SBATCH -t 0:30:05
#SBATCH --mem=10gb
#SBATCH -o NAMDjob.out  -J NAMDjob
#SBATCH --mail-type=all
#SBATCH --mail-user=jerry@uh.edu




module load namd

mpirun   namd2  apoa1.namd
```

See job filename:    job.namd.mpi.bash.slurm

# Job allocations

- ***sbatch*** always creates a new resource allocation when it is invoked, executes a job script on one of the allocated nodes (master node), then releases the allocation once the script terminates. An additional feature of sbatch is that it will parse this script at job submission time for lines that begin with #SBATCH and contain sbath options in lieu of command line arguments.

- ***srun*** may or may not create an allocation, depending on how it is invoked. If it is invoked on the command line of a login node, then it will create a new allocation and execute the command following srun. If it is invoked in a batch script, it will simply run a task on the current allocation. Likewise, srun may be given a --jobid argument that tells it to run the task as part of the given job, on the specified job's own allocation.

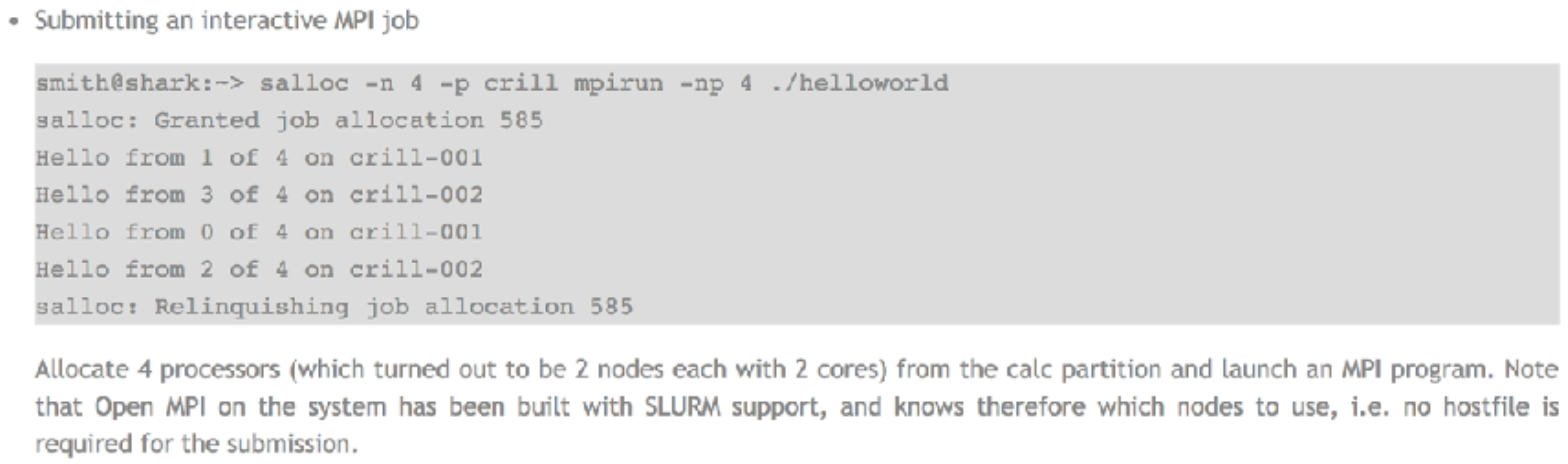
- ***salloc*** always creates a new resource allocation when it is invoked, but doesn't necessarily run any tasks on the allocated nodes. The typical use case of salloc is to create an allocation in order to run a series of subsequent srun commands either through an interactive bash session, or a script which runs from the login node. It releases the allocation after the script or bash session terminates. This use case is not supported on Opuntia, so salloc is of limited use there.

## Using an Interactive Job *srun*

- Special kind of batch job
- Useful for debugging applications, short tests, or for computational steering

```
srun -n 8 -t 02:00:00 --pty /bin/bash -l

srun -N 2 -n 8 -t 02:00:00 --pty /bin/bash -l

srun -n 8 -p gpu -t 02:00:00 --pty /bin/bash -l
```

# salloc

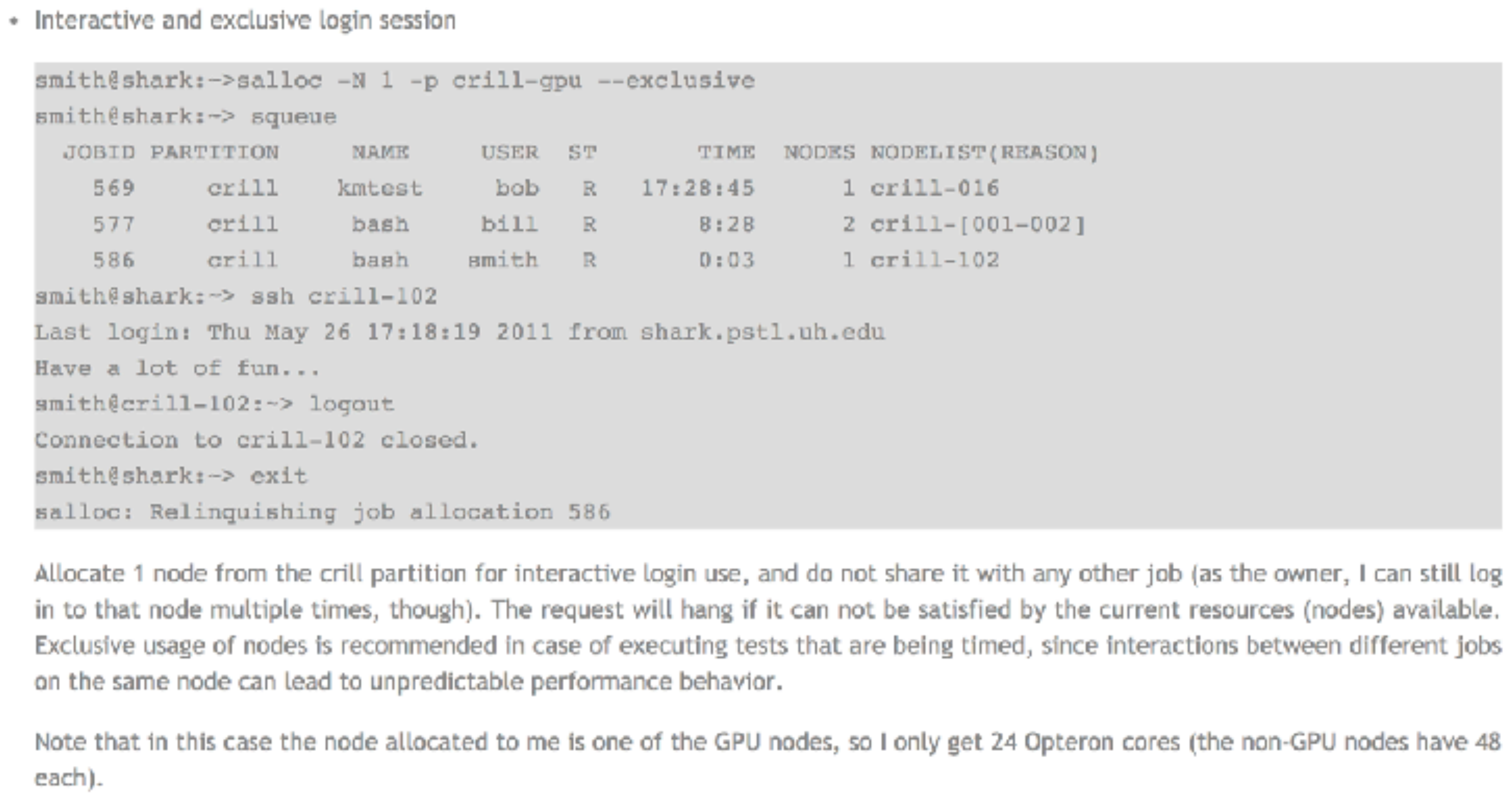


- Submitting an interactive MPI job

```
smith@shark:~> salloc -n 4 -p crill mpirun -np 4 ./helloworld
salloc: Granted job allocation 585
Hello from 1 of 4 on crill-001
Hello from 3 of 4 on crill-002
Hello from 0 of 4 on crill-001
Hello from 2 of 4 on crill-002
salloc: Relinquishing job allocation 585
```

Allocate 4 processors (which turned out to be 2 nodes each with 2 cores) from the calc partition and launch an MPI program. Note that Open MPI on the system has been built with SLURM support, and knows therefore which nodes to use, i.e. no hostfile is required for the submission.

# salloc -continued



- Interactive and exclusive login session

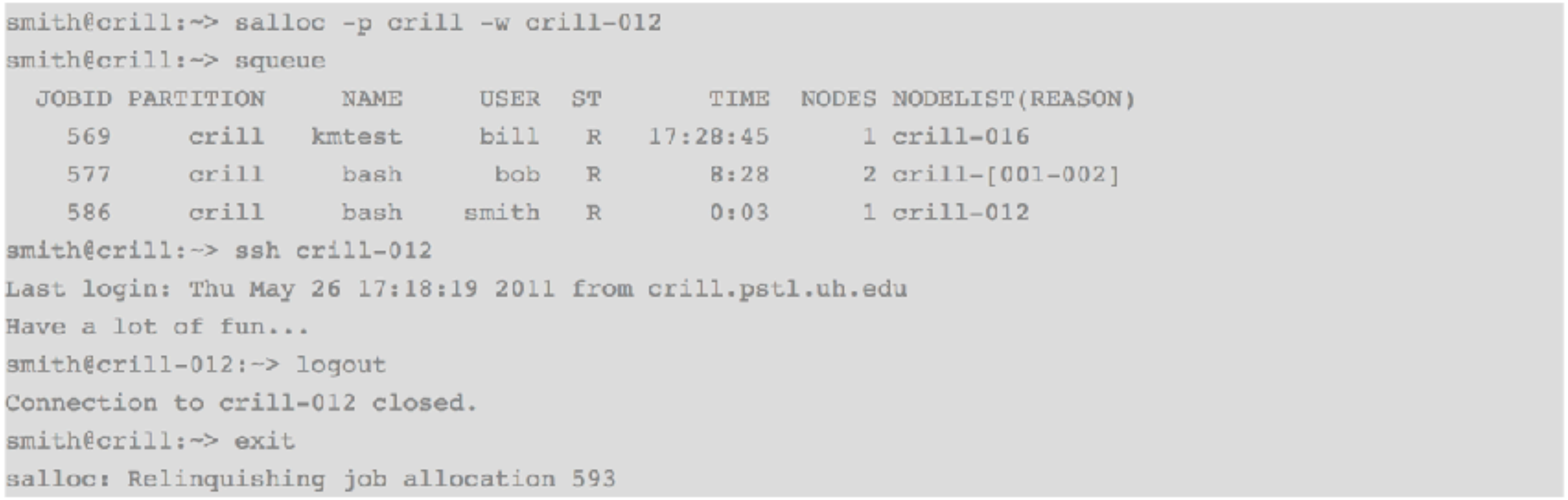
```
smith@shark:~>salloc -N 1 -p crill-gpu --exclusive
smith@shark:~> squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    569     crill   kmtest      bob   R   17:28:45      1 crill-016
    577     crill     bash     bill   R       8:28      2 crill-[001-002]
    586     crill     bash    smith   R       0:03      1 crill-102
smith@shark:~> ssh crill-102
Last login: Thu May 26 17:18:19 2011 from shark.pstl.uh.edu
Have a lot of fun...
smith@crill-102:~> logout
Connection to crill-102 closed.
smith@shark:~> exit
salloc: Relinquishing job allocation 586
```

Allocate 1 node from the crill partition for interactive login use, and do not share it with any other job (as the owner, I can still log in to that node multiple times, though). The request will hang if it can not be satisfied by the current resources (nodes) available. Exclusive usage of nodes is recommended in case of executing tests that are being timed, since interactions between different jobs on the same node can lead to unpredictable performance behavior.

Note that in this case the node allocated to me is one of the GPU nodes, so I only get 24 Opteron cores (the non-GPU nodes have 48 each).

# salloc -continued

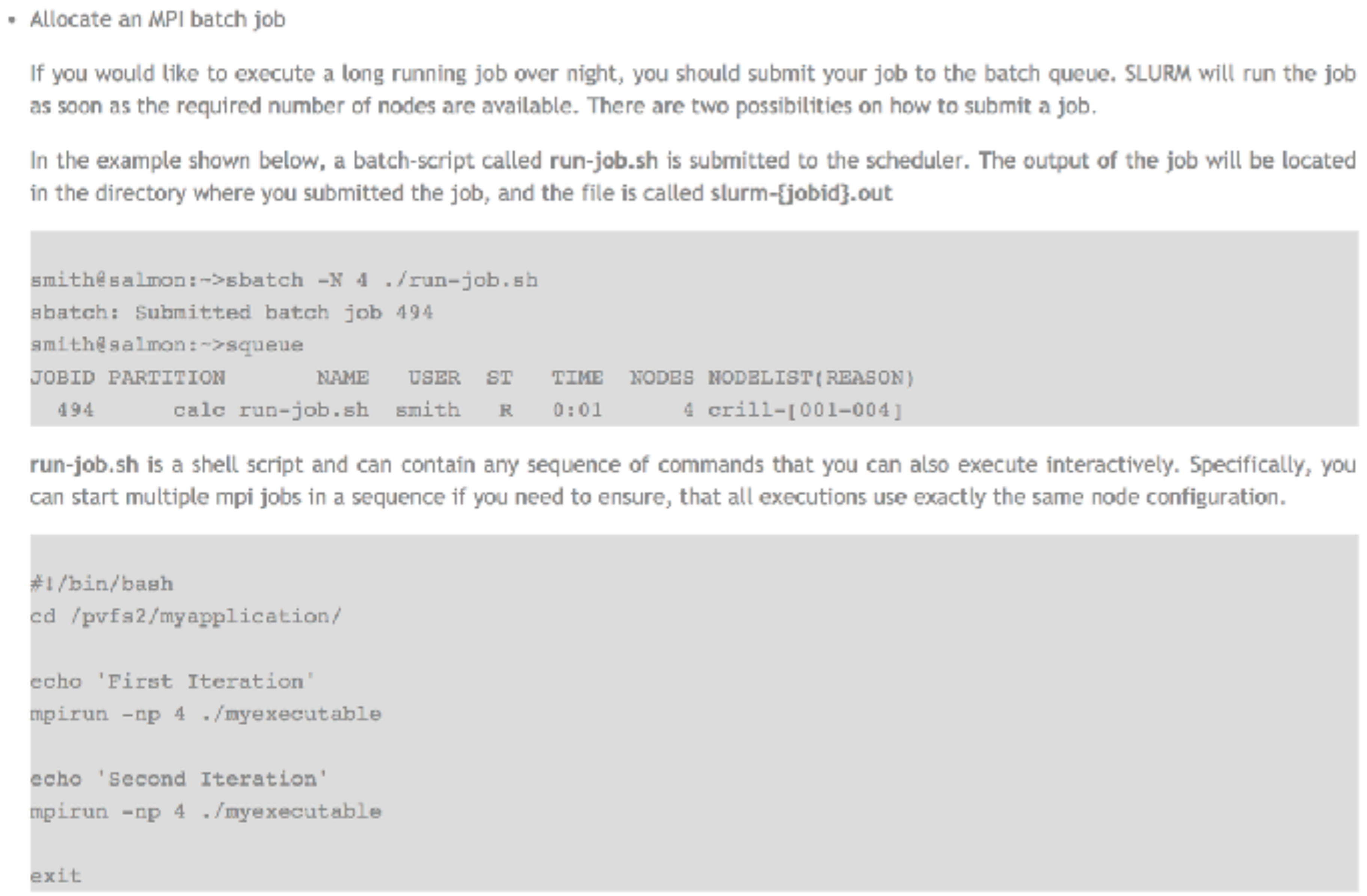
- Allocating a specific node



```
smith@crill:~> salloc -p crill -w crill-012
smith@crill:~> squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    569     crill   kmtest     bill   R   17:28:45      1 crill-016
    577     crill     bash      bob   R       8:28      2 crill-[001-002]
    586     crill     bash    smith   R       0:03      1 crill-012
smith@crill:~> ssh crill-012
Last login: Thu May 26 17:18:19 2011 from crill.pstl.uh.edu
Have a lot of fun...
smith@crill-012:~> logout
Connection to crill-012 closed.
smith@crill:~> exit
salloc: Relinquishing job allocation 593
```

Allocate crill node 012 explicitly. May be shared with other users unless it already has been allocated exclusively.

# salloc -continued



- Allocate an MPI batch job

  If you would like to execute a long running job over night, you should submit your job to the batch queue. SLURM will run the job as soon as the required number of nodes are available. There are two possibilities on how to submit a job.

  In the example shown below, a batch-script called **run-job.sh** is submitted to the scheduler. The output of the job will be located in the directory where you submitted the job, and the file is called **slurm-{jobid}.out**

```
smith@salmon:~>sbatch -N 4 ./run-job.sh
sbatch: Submitted batch job 494
smith@salmon:~>squeue
JOBID PARTITION       NAME   USER  ST   TIME  NODES NODELIST(REASON)
  494       calc run-job.sh  smith   R   0:01      4 crill-[001-004]
```

  **run-job.sh** is a shell script and can contain any sequence of commands that you can also execute interactively. Specifically, you can start multiple mpi jobs in a sequence if you need to ensure, that all executions use exactly the same node configuration.

```
#!/bin/bash
cd /pvfs2/myapplication/

echo 'First Iteration'
mpirun -np 4 ./myexecutable

echo 'Second Iteration'
mpirun -np 4 ./myexecutable

exit
```

UNIVERSITY of HOUSTON
CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

# Killing Jobs

- One, or a few jobs:
  - scancel  [jobID]  [jobID]  [jobID] ...

- Kill all of your jobs:
  - scancel  -u  $USER

- Kill all of your queued jobs:
  - scancel  -u  $USER    -t   PENDING

- Kill all of your running jobs:
  - scancel  -u  $USER    -t    RUNNING

# SLURM Script for Executing Matlab script

## Using Proprietary Matlab runtime (sample I)

```
#!/bin/bash

#SBATCH -n 8 -N 1

#SBATCH -t  0:30:05

#SBATCH --mem=10gb

#SBATCH -o matlabjob.out  -J matlabjob

#SBATCH --mail-type=all

#SBATCH --mail-user=jerry@uh.edu



##set up your environment



module add matlab



matlab << EOF

a = 10;  b = 20; c = 30;

d = sqrt((a + b + c)/pi)

exit

EOF
```

See job filename:    job.matlab1.bash.slurm

# SLURM Script for Executing Matlab script

## Using Proprietary Matlab runtime (sample II)

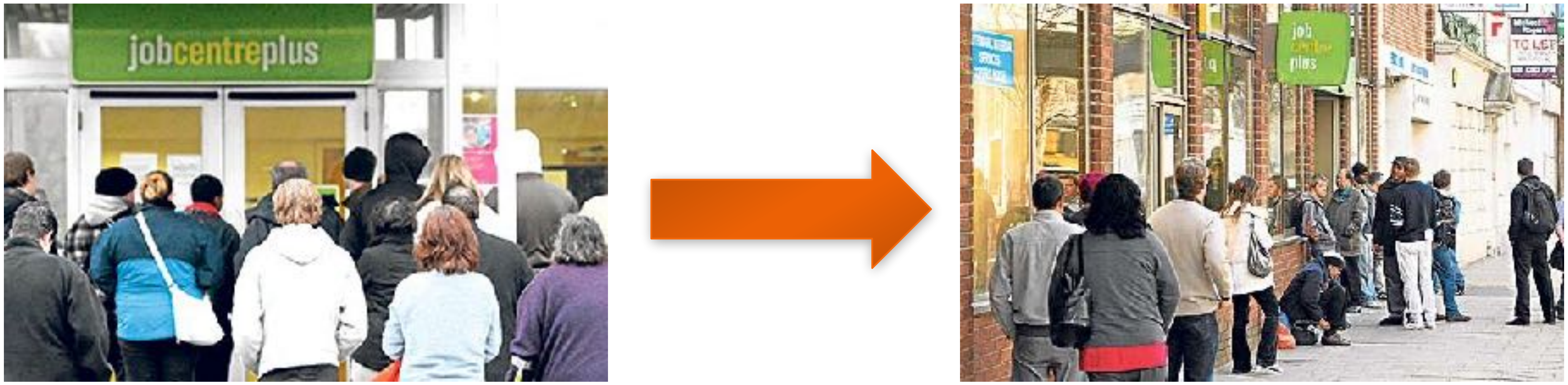
```
#!/bin/bash

#SBATCH -n 8 -N 1

#SBATCH -t 0:30:05

#SBATCH --mem=10gb

#SBATCH -o matlabjob.out  -J matlabjob

#SBATCH --mail-type=all

#SBATCH --mail-user=jerry@uh.edu



module add matlab



## run matlab program compute pseudo inverse for  100  matrices of size 400x400

time    matlab    < matrix_inversion100_matlab.m
```

See job filename:    job.matlab2.bash.slurm

UNIVERSITY of HOUSTON
CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

# The Queue System (PBS)

- Portable Batch System (aka PBS) is the <u>resource management service</u> used on the Maxwell Cluster
- Enables you to make more efficient use of your time through scripting computational tasks
- PBS takes care of running these tasks and returning the results
- If the cluster is full, PBS holds your tasks and runs them when the resources are available
- PBS ensures fair sharing of cluster resources (policy enforcement)
    - PBS ensures optimal/efficient use of available resources



# PBS Commands

| Command(s) | Description |
|---|---|
| **qstat** | Check status of all jobs |
| **qsub ./myjob** | Submit batch jobs |
| **qstat -u $USER** | Check status of just your jobs |
| **qsub -I** | Submit an interactive job |
| **qhold jobID**<br>i.e. qhold 12345 | Put a job on hold (before it starts) |
| **qrls jobID** | Release a job from hold status |
| **qdel #jobID** | Delete a job, running or not |

# Using qsub with a Job Script

- Syntax/Format:
  - **qsub [qsub params] [job script file]**

- Examples:
  - **qsub myjob**

# PBS Batch Job Script File

- Very simple example:
- Serial job requesting 1 CPU, 2 GB memory, 60 hours of walltime

```
#!/bin/bash
#PBS   -N   jobname
#PBS   -l  mem=2gb,walltime=60:00:00
#PBS   -j  y

cd $PBS_O_WORKDIR

./my.program.exe [arguments for my program]
```

## Using Parallel Environment to Manage Memory and CPU Core Availability

- #PBS -l nodes=16:ppn=1 --use only one CPU core per node and use a total of 16 processes.

- #PBS -l nodes=8:ppn=2 --use only 2 CPU cores per node and use a total of 16 processes

- #PBS -l nodes=4:ppn=4 --use only 4 CPU cores per node and use a total of 16 processes

- #PBS -l nodes=8:ppn=8 --use only 8 CPU cores per node and use a total of 64 processes

# qsub Notification Parameters

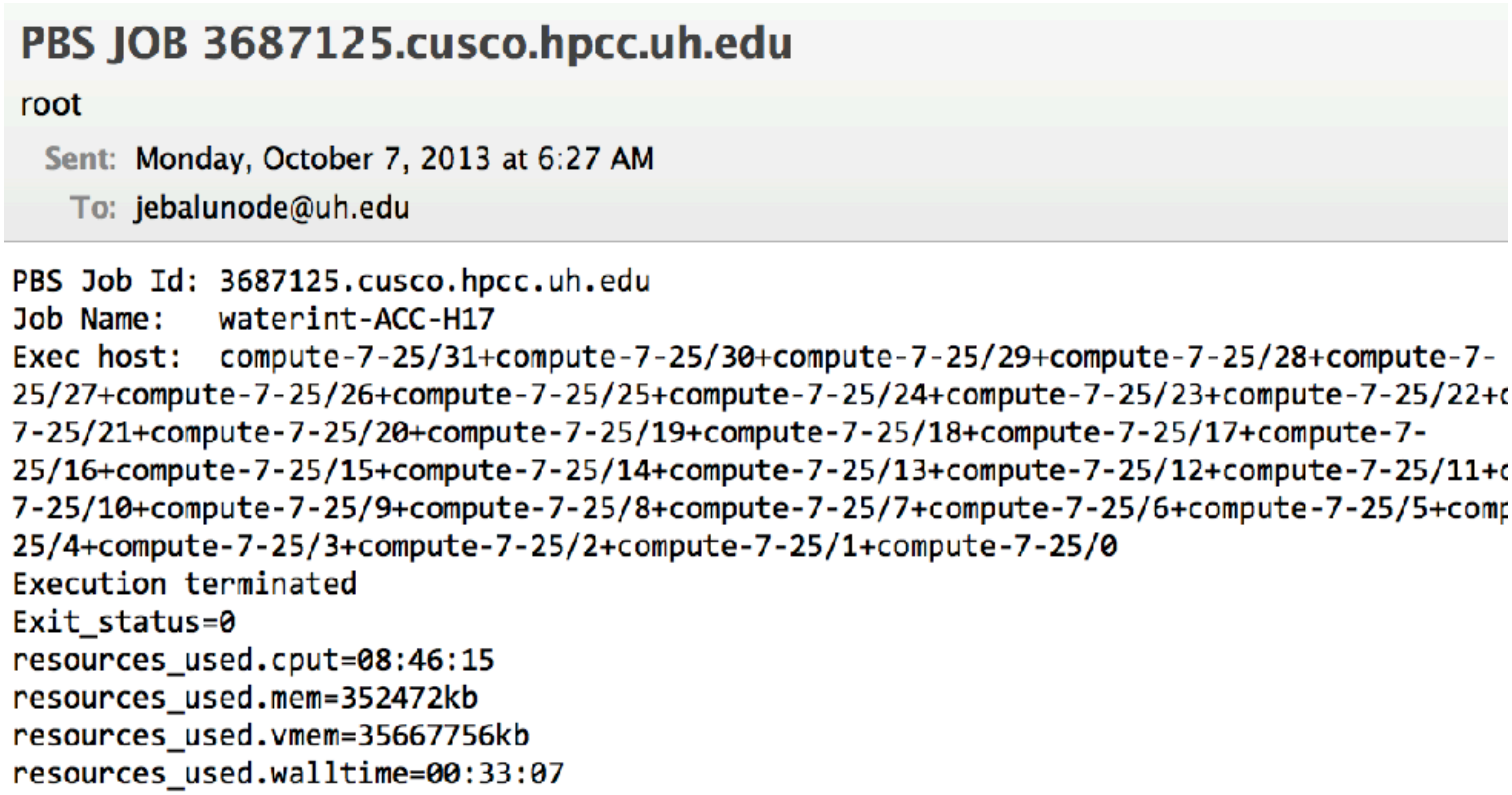
- #PBS  -M [e-mail address]
  - e-mail address can be a list of email addresses
  - separated by commas.

- #PBS  -m bea or -m be or -m e
  - send an email when the job **b**egins, **e**nds, or is **a**borted.

**Example:**

```
#!/bin/bash
#PBS   -N   jobname
#PBS   -j  y
#PBS  -M   jerry@uh.edu
#PBS  -m   abe

cd $PBS_O_WORKDIR

./my.program.exe [arguments for my program]
```

# Sample Notification Email



**PBS JOB 3687125.cusco.hpcc.uh.edu**

root

Sent: Monday, October 7, 2013 at 6:27 AM

To: jebalunode@uh.edu

```
PBS Job Id:  3687125.cusco.hpcc.uh.edu
Job Name:    waterint-ACC-H17
Exec host:   compute-7-25/31+compute-7-25/30+compute-7-25/29+compute-7-25/28+compute-7-
25/27+compute-7-25/26+compute-7-25/25+compute-7-25/24+compute-7-25/23+compute-7-25/22+c
7-25/21+compute-7-25/20+compute-7-25/19+compute-7-25/18+compute-7-25/17+compute-7-
25/16+compute-7-25/15+compute-7-25/14+compute-7-25/13+compute-7-25/12+compute-7-25/11+c
7-25/10+compute-7-25/9+compute-7-25/8+compute-7-25/7+compute-7-25/6+compute-7-25/5+comp
25/4+compute-7-25/3+compute-7-25/2+compute-7-25/1+compute-7-25/0
Execution terminated
Exit_status=0
resources_used.cput=08:46:15
resources_used.mem=352472kb
resources_used.vmem=35667756kb
resources_used.walltime=00:33:07
```

# Handling Output Files

- When a job is started it takes its job name from the script file that was submitted.
  - The standard output and error output are sent to files named jobname.o987349 and jobname.e987349 (in your job directory )

- The following parameters modify this behavior:
  - -e [path] standard error output file.
  - -o [path] standard output file
  - -j y merge the error and standard output

- **#PBS -N** jobname name to be used for the job.

# Using qsub with a Job Script

- Syntax/Format:
  - **qsub [qsub params] [script file]**

- More examples on using parameters at the command line:
  - `qsub -N  test1   myjob.sh`
  - `qsub -l nodes=2:ppn=2  -o test2.out -N test2 myjob.sh`
  - `qsub  -l walltime=02:00:00,mem=4gb myjob.sh`
  - `qsub  -l nodes=2:ppn=32,mem=64gb   myjob.sh`
  - `qsub  -q gpu -l nodes=1:ppn=4  my_gpu_job.sh`

- To see all available qsub options, just run qsub --help

# Using an Interactive Job

- Special kind of batch job
- Useful for debugging applications, short tests, or for computational steering

Capital "i"

Small L "l"

```
qsub -I

qsub -I -l  walltime=2:00:00

qsub -I -l walltime=2:00:00,nodes=2:ppn=8

qsub -I -l  walltime=2:00:00  -q gpu
```

## Job Queues

| Queue Name | Description |
|---|---|
| short | Queue to run short jobs ( =< 4 hours) |
| medium | Queue to run medium jobs ( 1 week allowed) |
| long | Queue to run very long jobs (2 weeks allowed) |
| gpu | Queue to run gpu jobs on GTX 570 equipped nodes  (2 weeks allowed) |
| gpu-tesla | Queue to run gpu jobs on Testla C2075 equipped nodes  (2 weeks allowed) |
| | |

USE “qstat -q” to probe the  queues installed
Note gpu-tesla and gpu queue are not available/functional on the surrogate cluster

## PBS Script for Serial Program
1 CPU

- Example  of a Serial Job Script

filename:  job.gethostname.bash

```
#!/bin/bash
#PBS -N gethost
#PBS  -l mem=8gb,walltime=00:25:00
#PBS  -l nodes=1:ppn=1
#PBS  -M JohnDoe@gmail.com
#PBS -m bea

cd $PBS_O_WORKDIR

### Run gethostname

time   ./gethostname.exe
```

To submit the  job run:
```
qsub job.gethostname.bash
```

# PBS Script for MPI Program

```
#!/bin/bash
#PBS -N gethostmpi
#PBS -l nodes=2:ppn=8,pmem=1gb
#PBS -S /bin/bash
#PBS -l walltime=00:05:00
#PBS -M monkeybrain@sc.edu
#PBS -m bea


cd $PBS_O_WORKDIR
##set up your environment
source /etc/profile.d/modules.sh


module add openmpi


mpirun –np  16 ./gethostname.mpi.exe > output.from.txt
```

See job filename:   job.gethostname.mpi.bash

To submit the job run:
```
qsub job.gethostname.mpi.bash
```

# PBS Script for GPGPU Program

Give me a node with this kind of gpu

Example GPU Job Script

```
    #!/bin/bash
    #PBS -q gpu
    #PBS -N sortrandomkeys
    #PBS -l walltime=00:05:00,nodes=1:ppn=4
    #PBS -M monkeybrain@gmail.com
    #PBS –m bea

     cd $PBS_O_WORKDIR
    ##set up your environment
     source /etc/profile.d/modules.sh
    module add cuda-toolkit
    time    ./sortkeys_basic_thrust.exe
```

Load CUDA environment

See job filename: job.gpu.bash or job.gpu.sort.bash

# PBS Script for Executing Matlab script

## Using Proprietary Matlab runtime (sample I)

```
#!/bin/bash
#PBS -N matlabjob
#PBS -l nodes=1:ppn=1,pmem=1gb
#PBS -S /bin/bash
#PBS -l walltime=00:05:00

cd $PBS_O_WORKDIR

##set up your environment
source /etc/profile.d/modules.sh
module add matlab

matlab << EOF
a = 10;  b = 20; c = 30;
d = sqrt((a + b + c)/pi)
exit
EOF
```

See job filename:    job.matlab1.bash

# PBS Script for Executing Matlab script

## Using Proprietary Matlab runtime (sample II)

```
#!/bin/bash
#PBS -N matlabjob
#PBS -S /bin/bash
#PBS -l walltime=20:05:00,pmem=1gb,nodes=1:ppn=4
#PBS -M monkeybrain@uh.edu
#PBS -m bea
##set up your environment
cd  $PBS_O_WORKDIR
source /etc/profile.d/modules.sh
module add matlab


## run matlab program compute pseudo inverse for  100  matrices of size 400x400
 time    matlab    < matrix_inversion100_matlab.m
```

See job filename:    job.matlab2.bash

## PBS Script for Executing R script

```
#!/bin/bash
#PBS -N R-job
#PBS -S /bin/bash
#PBS -l walltime=0:05:00,pmem=1gb,nodes=1:ppn=1
#PBS -M monkeybrain@uh.edu
#PBS -m bea
##set up your environment
cd  $PBS_O_WORKDIR
source /etc/profile.d/modules.sh
module add R


## run R program
 R –vanilla < sample.r
```

See job filename:    job.R.bash

## PBS Script for NAMD2 Job
## Parallel MPI

```
#!/bin/bash
#PBS -l nodes=1:ppn=4
#PBS -l walltime=00:05:00
#PBS -l pmem=1gb
#PBS -N  namd
#PBS –V

cd $PBS_O_WORKDIR



module load namd

mpirun –v  namd2  apoa1.namd
```

See job filename:    job.namd.mpi.bash

# Killing Jobs

- One, or a few jobs:
  - qdel  [jobID]  [jobID]  [jobID] ...

- Kill all of your jobs:
  - qselect  -u  $USER  |  xargs qdel

- Kill all of your queued jobs:
  - qselect  -u $USER  -s  Q |  xargs qdel

- Kill all of your running jobs:
  - qselect  -u  $USER  -s  R  |  xargs qdel