Ohio Supercomputer Center

An OH·TECH Consortium Member

Conquering the OSC Batch Environment

or

Why Must I Get in Line?
I Want to Run Now!

Marcio Faerman, Ph.D. (mfaerman@osc.edu)
University of Cincinnati
October 2014

Understanding the Infrastructure

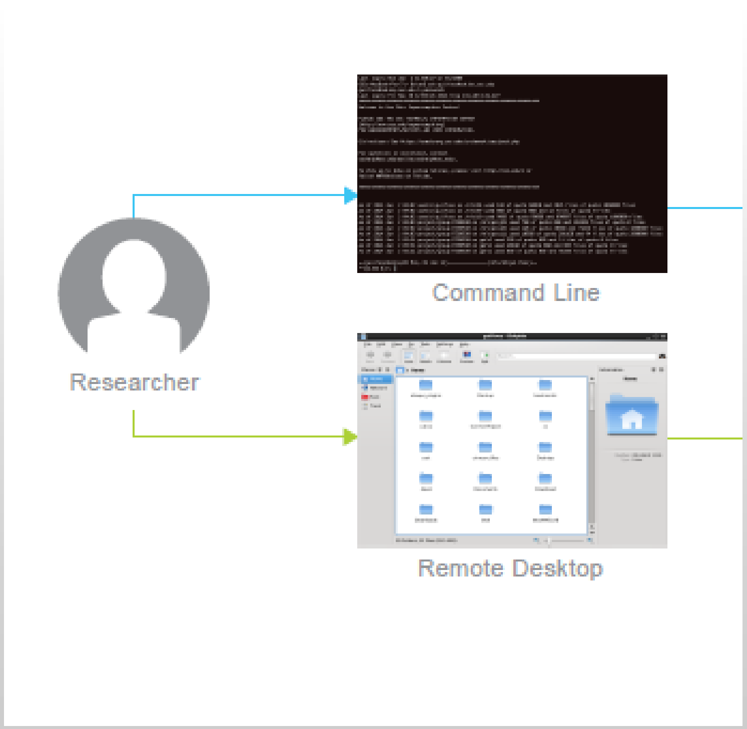
What Can I do?

☺ Many Compute Resources!

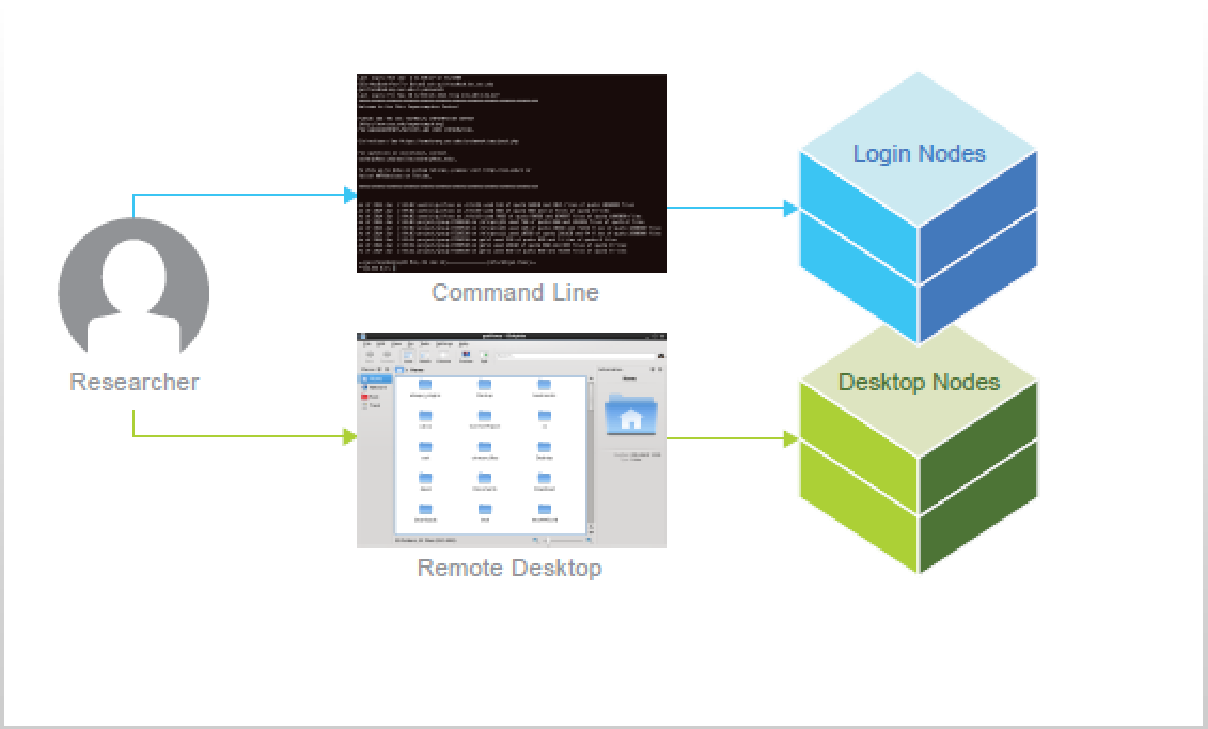
OSC's HPC Clusters:

- Oakley - 8,300 cores
  - Glenn – 3,400 cores
    - Ruby 2014 – 4,800 cores

The User and an OSC Cluster

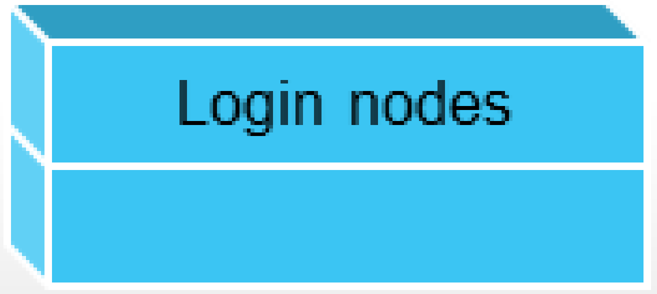


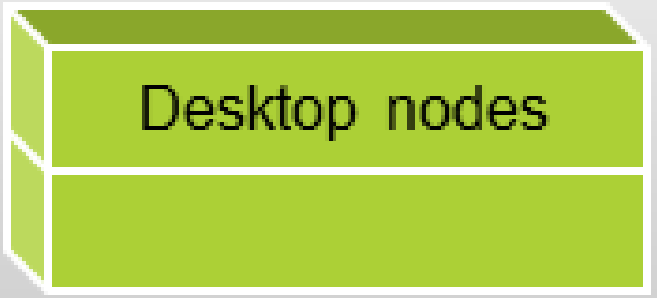
The User and an OSC Cluster



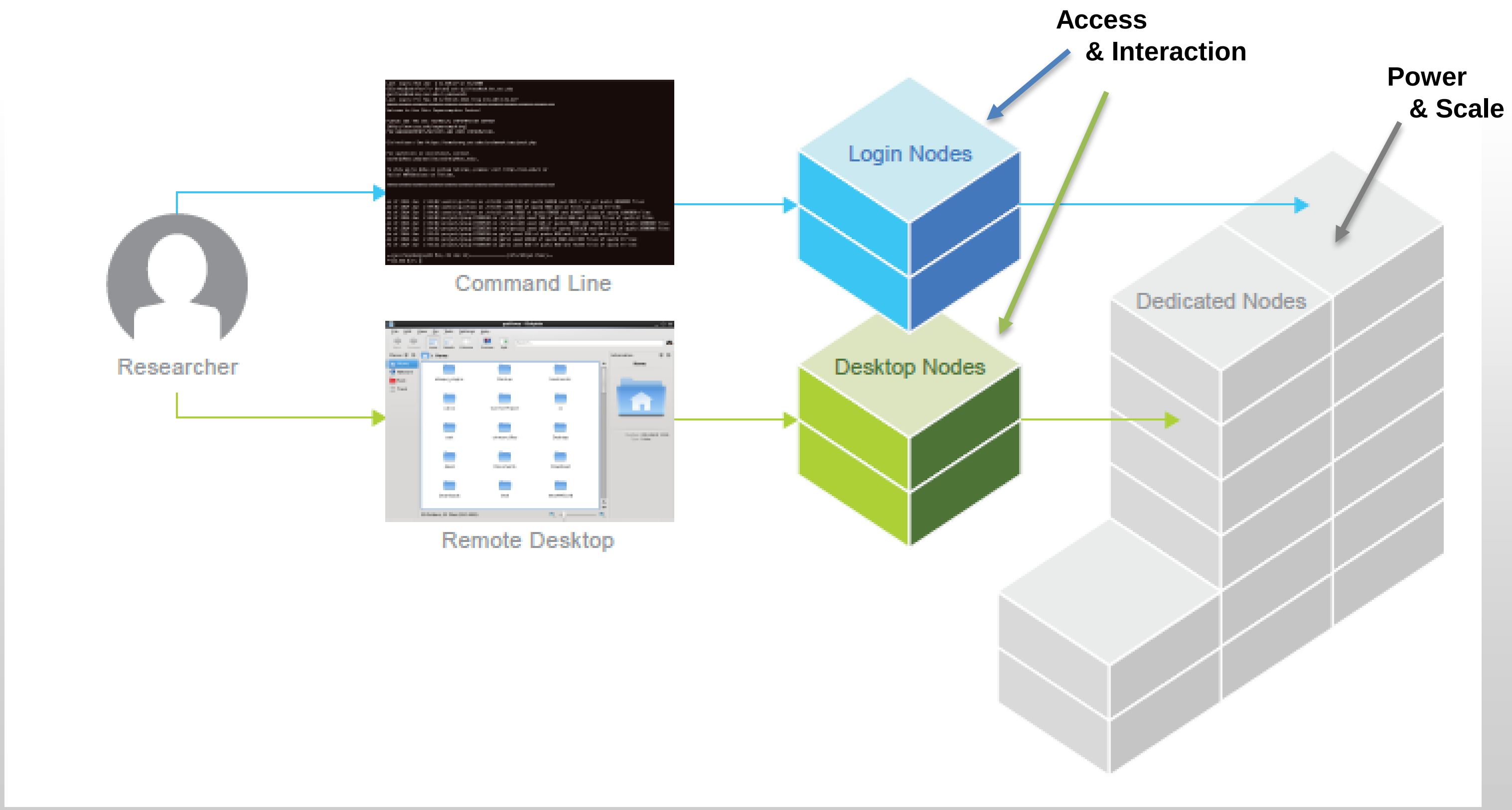
Interacting
The Login and Desktop Nodes

- Purpose
  - Gateway
    - Submit jobs to batch system
  - Interactive Sandbox
    - Edit files
    - Manage your files
    - Interactive work – **small scale**
      - Compiling and some debugging
- Limits
  - 20 minutes CPU time
  - 1GB memory
- Use the batch system for serious computing!

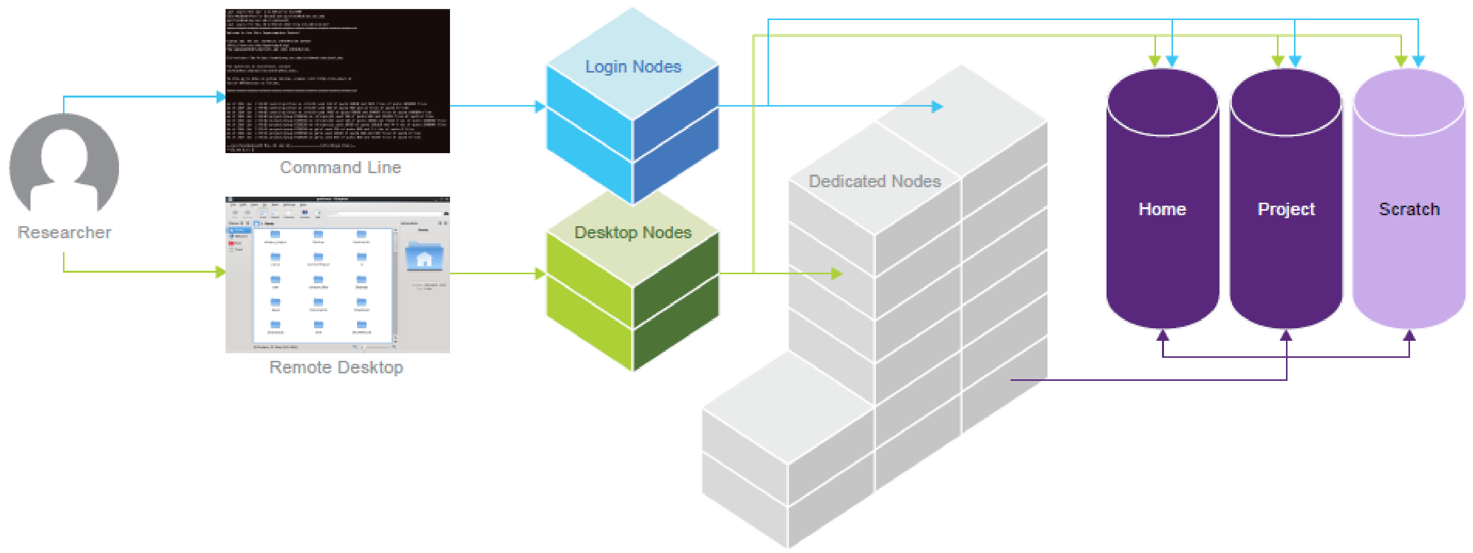




The User and an OSC Cluster

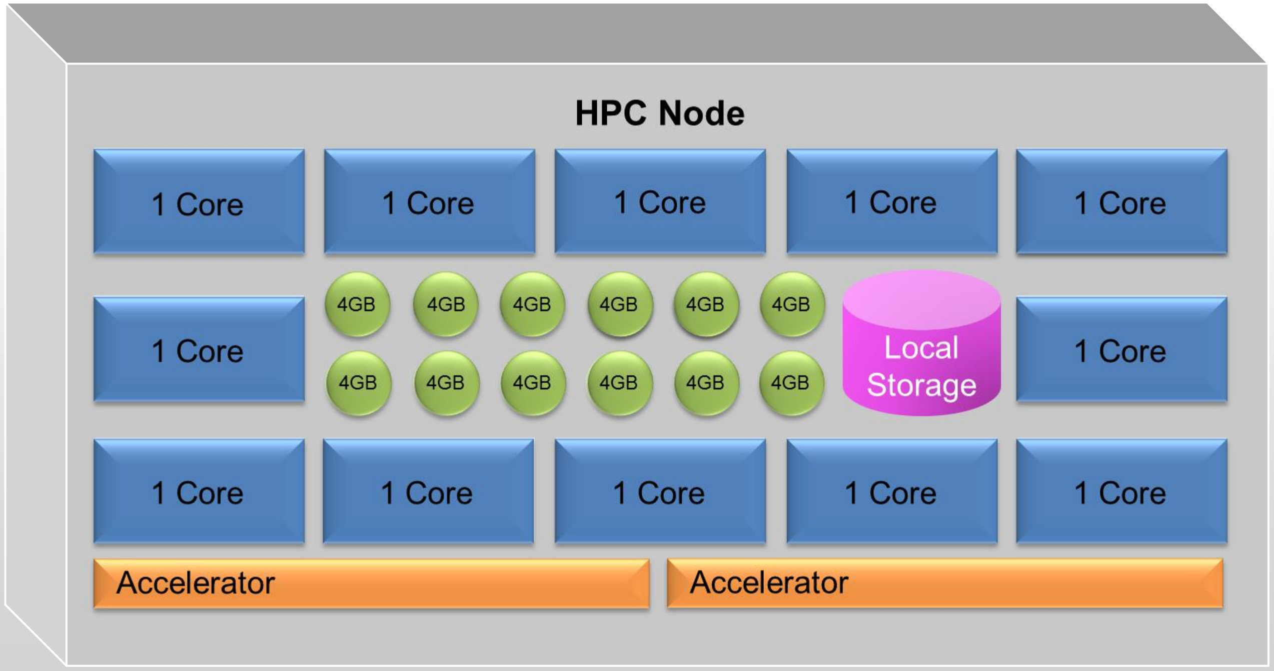


Organization of an OSC Cluster



The Node Components

- Processors
- Memory
- Storage
- Special HW
  - GPUs
  - Accelerators



OSC Computational Capacity

	Oakley System (2012)	Glenn System (Phase III, 2014)
Theoretical Peak Performance	88.6 TF +65.5 TF (GPU) ~154 TF	53TF +6 TF (GPU) ~60 TF
Number of Nodes	692	426
Cores Per Node	12 cores/node	8 cores/node
Number of CPU Cores	8304	3408

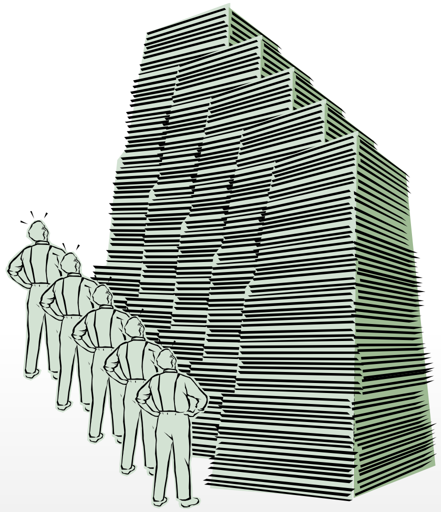
OSC Computational Capacity

	Ruby System (2014)	**Oakley System (2012)**	**Glenn System (Phase III, 2014)**
Theoretical Peak Performance	96 TF +28.6 TF (GPU) +20 TF (Xeon Phi) ~144 TF	88.6 TF +65.5 TF (GPU) ~154 TF	53TF +6 TF (GPU) ~60 TF
Number of Nodes	240	692	426
Cores Per Node	20 cores/node	12 cores/node	8 cores/node
Number of CPU Cores	4800	8304	3408

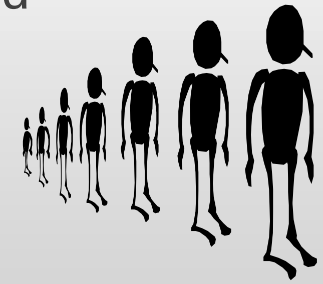
Understanding the Infrastructure

What Can I do? **When** Can I do it?

- Many Compute Resources ☺
- Many users 😐
  - Crowd brings much more processes to run
    - Than computer processors available
  - Not everyone is able to run at the same time ☹
    - Even though we wish you could

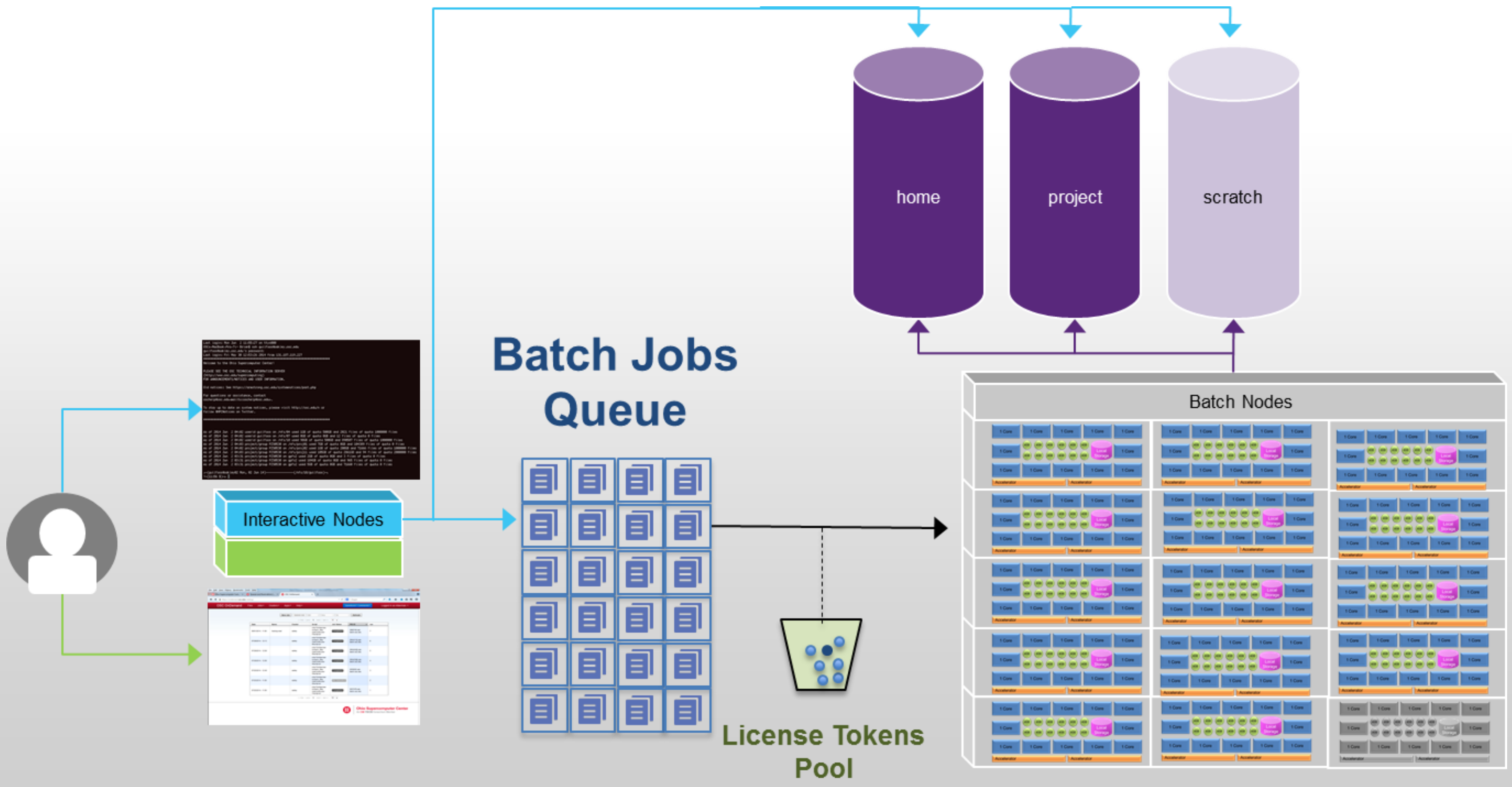


- What to do?
  - Let's get folks in line



- ***The only access to significant resources on the HPC machines is through the batch job requests***

The Batch Jobs Queue

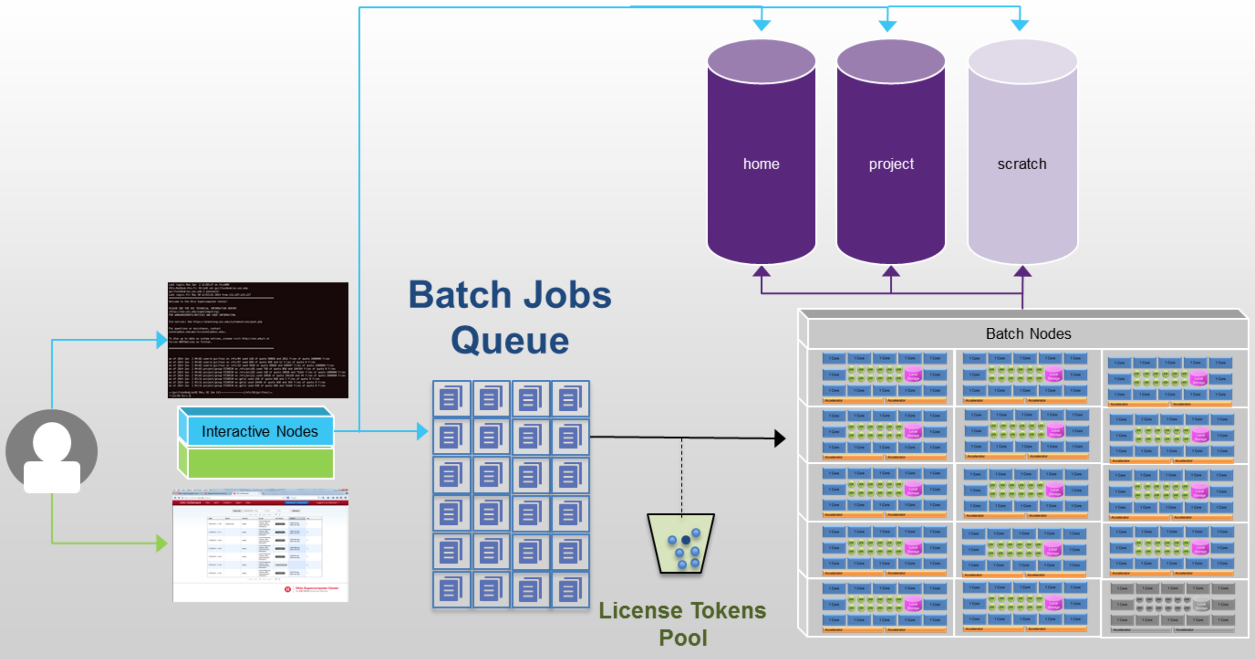


What Else Needed

to Make Through the Queue and Run

More to consider in addition to just compute nodes?

- Number of Cores,
- Memory,
- Software
  - Availability,
  - Licenses
- Special Resources,
  - Accelerators,
  - GPUs
- Storage,
  - Access permissions,
  - Space availability
- Priority Policies
- Resource Limits
- RUs (Resource Units)



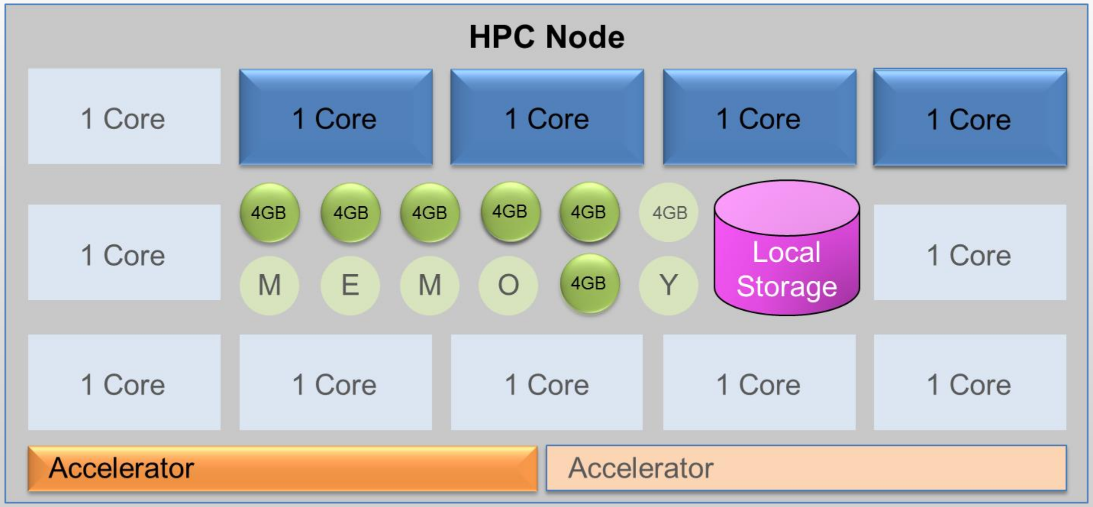
Idea Behind Batch Processing

- System runs the job when **Resources** become available
- **Batch Script** Requests Resources
  - **What** will be **needed**
  - **How Long**
- Put keyboard input into **Batch Script**
- Screen output goes into a **log file** (or files)

- Very efficient in terms of resource utilization
- Requires more preparation than interactive processing

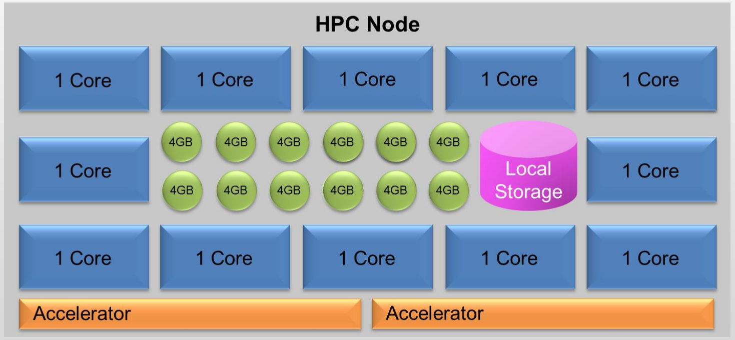
Scheduling Policies

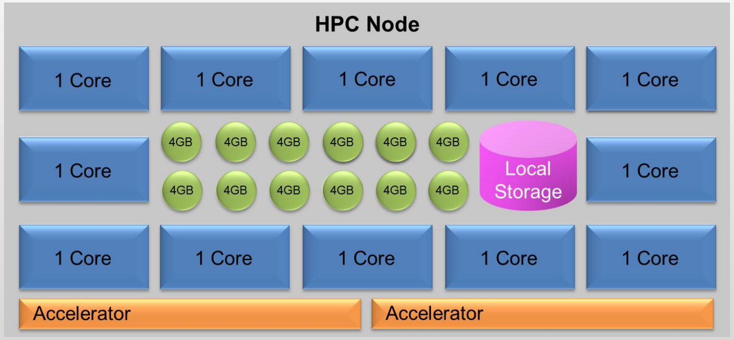
- Serial jobs requesting less than a full node
  - May **share** a node with **other jobs**



Scheduling Policies

- Parallel jobs are always allocated (and charged for) whole nodes





- Note: Serial jobs requiring more than the default amount of memory per core are charged extra

Hardware Characteristics

	# of nodes	# of cores per node (ppn)	Memory	Temporary file space
Oakley (standard)	690	12	48 GB	812 GB
Oakley (bigmem)	8	12	192 GB	812 GB
Oakley (hugemem)	1	32	1 TB	812 GB
Glenn (newdual)	400	8	24 GB	392 GB

Walltime and Processor Limits per Job Oakley

- Serial jobs
  - Request 1 node and up to 12 processor cores
  - 168 hour limit (1 week)
  - Exceptions possible, up to 2 weeks
- Parallel jobs
  - Request multiple nodes and up to 2040 processor cores
  - 96 hour limit (4 days)
- Huge memory node
  - Request 1 node and 32 processor cores
  - 48 hour limit

Limits per User and Group

- User
  - Up to 128 concurrently running jobs and/or
  - Up to 2048 cores in use
- Group
  - Up to 192 concurrently running jobs and/or
  - Up to 2048 cores in use
- Excess jobs wait in queue until other jobs exit
- No more than 1000 jobs per user in the system at once

Charging Algorithm

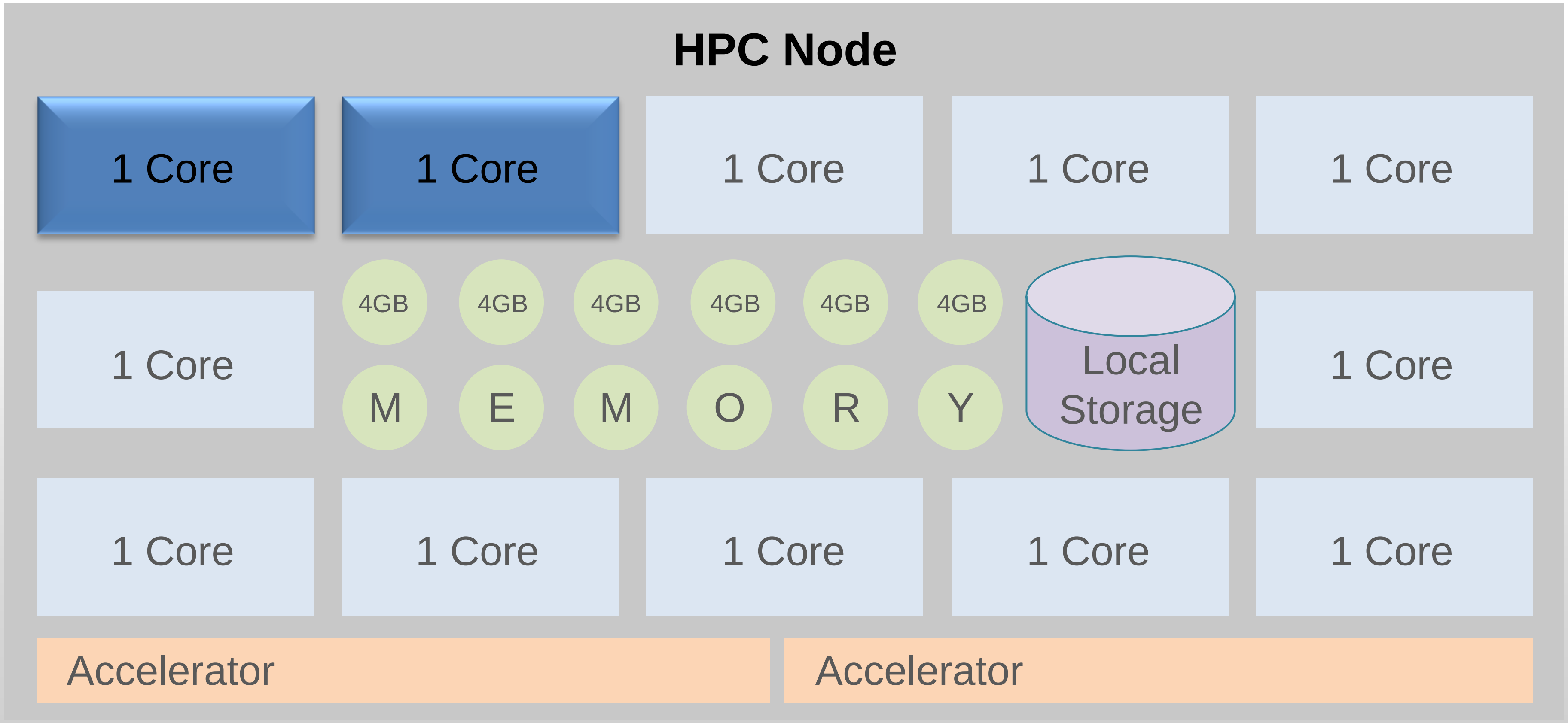
- Charges are in resource units (RUs)
- 1 RU = 10 CPU hours
- Serial job (1 node)
  - CPU hours = # of cores (ppn) requested * walltime used
  - Ex: nodes=1:ppn=12, 1.5 hours walltime used => 1.8 RUs
- Parallel job (2 or more nodes)
  - Charged for whole nodes regardless of ppn requested
  - CPU hours =
  - # of nodes requested * # or cores on node * walltime used
  - Ex: nodes=10:ppn=1, Oakley (12 cores/node), 1.5 hours walltime used => 18 RUs

Memory Containers (not on Glenn)

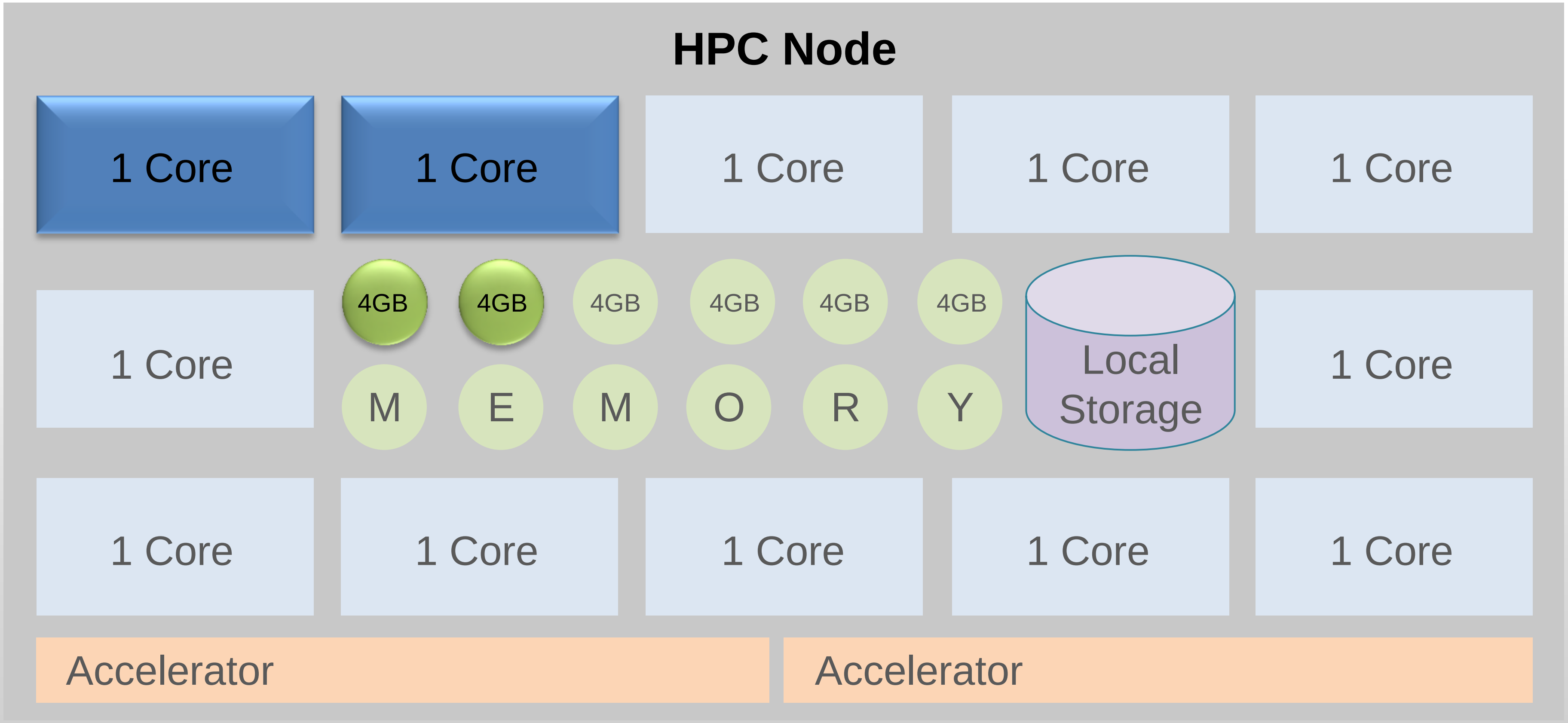
Now memory counts!

- Nodes=1:ppn=1,mem=12GB
  – Such requests didn't work properly before
- Change rolled out in October, 2013
  – Jobs allocated 4GB per core if explicit memory request not included
    - Effective Cores = memory / memory per core
    - **Charge** for **Effective Cores**

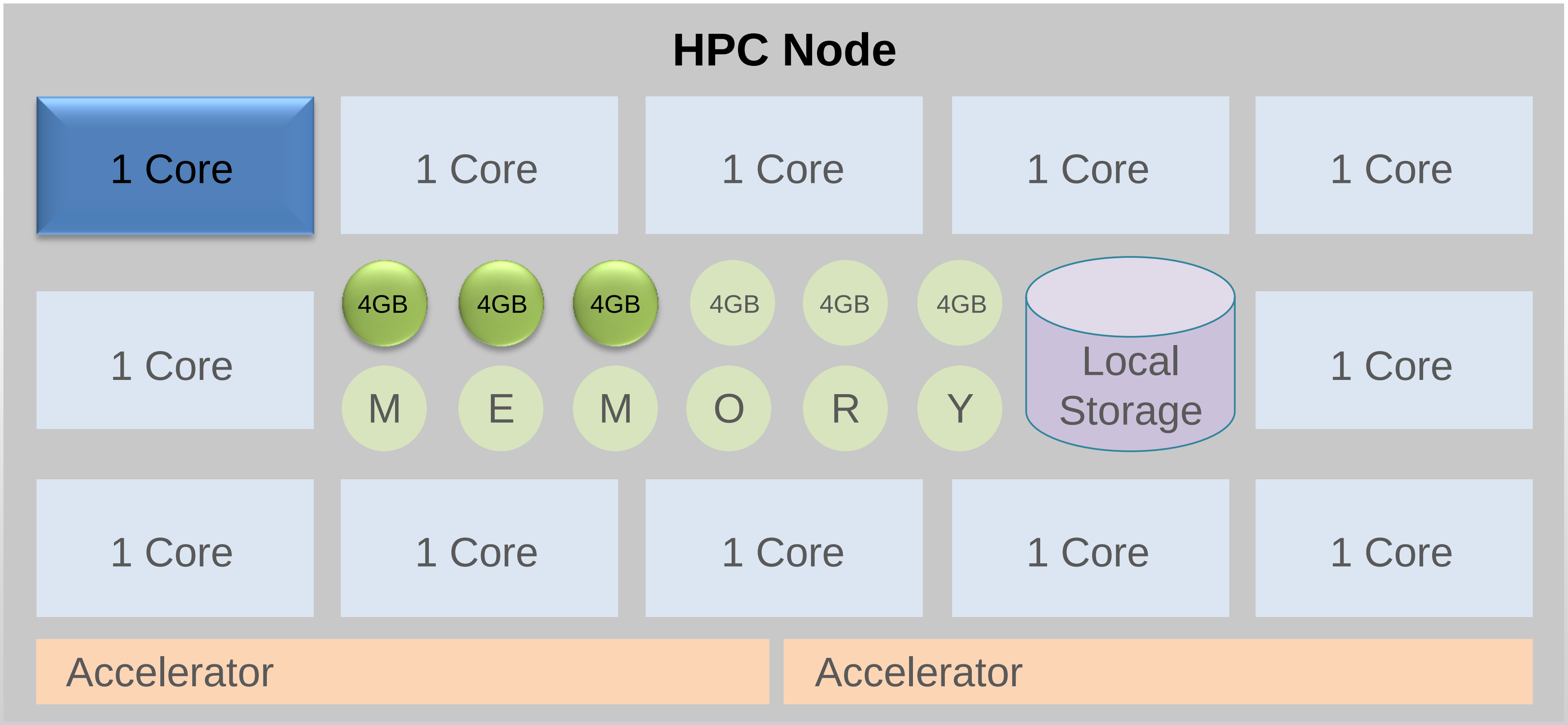
Serial Request (nodes=1:ppn=2)



Serial Request (nodes=1:ppn=2)
Implies Memory Limit of 8GB

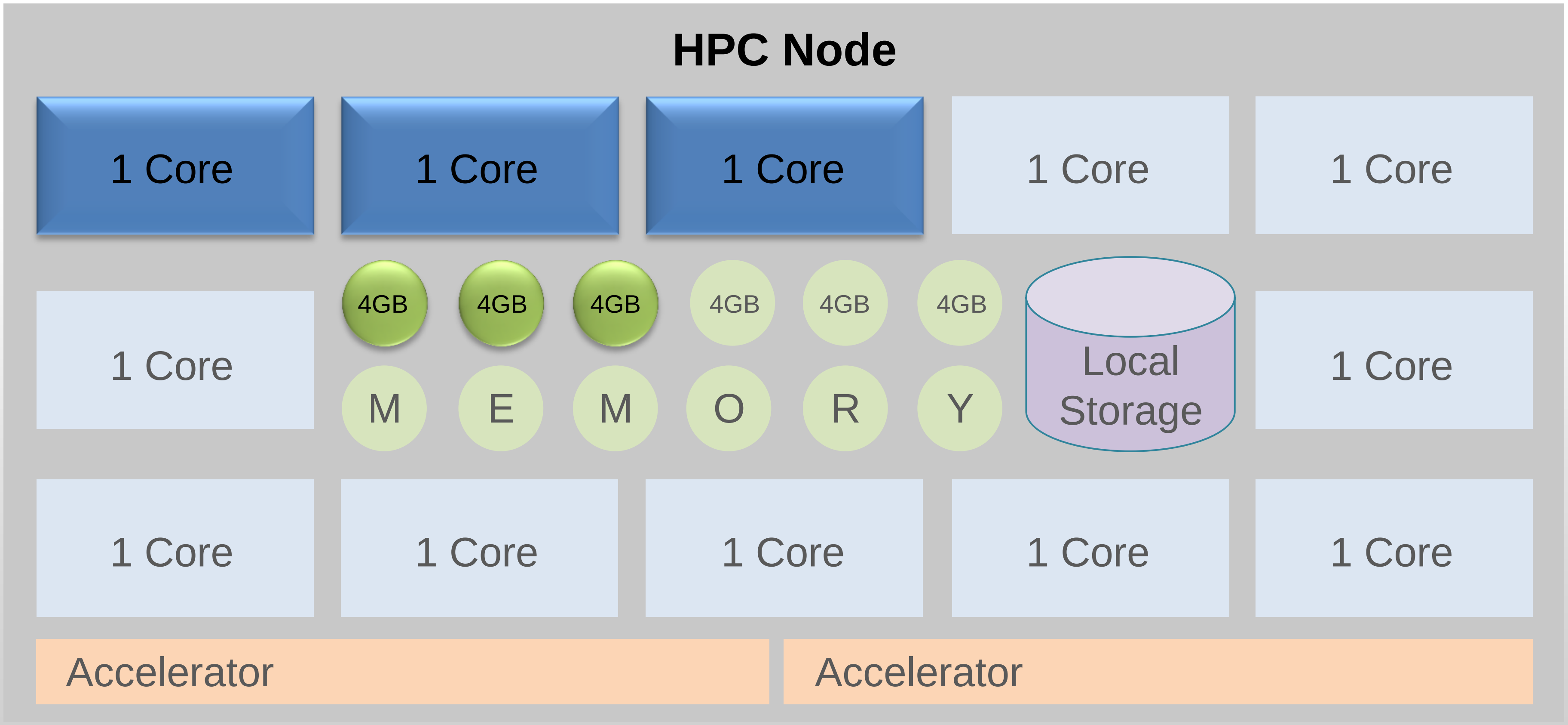


Serial Request (nodes=1)
1 core (ppn=1), Memory (12 GB)



Actual Charge: 3 Effective Cores

Memory (12 GB)



Priority Scheduling

- Scheduling is not strictly first-come first-serve
- Many factors involved in priority calculation
  - Length of time job has been waiting
  - Processor count requested
  - “Fair share” – reduced priority
    - How much computing user has done over last few days
    - How much user’s group has done over last few days
  - Penalty for projects with large negative RU balances

Scheduling Algorithm

- Scheduler runs as many top priority jobs as possible
- Scheduler identifies highest priority job that cannot currently be run
  - Finds time in future to reserve for it
- Backfill
  - Scheduler backfills as many lower priority jobs as reserved resources permit
  - Small jobs are most likely to fit into scheduling holes
- Keeps overall utilization of system high
- Allows reasonable turnaround time for high priority jobs

More on Scheduling

- Highest priority does not mean a job will run immediately
  - Must free up enough resources (processors and memory) to run it

- Debugging
  - Small number of nodes set aside during the day
  - Walltime limit of 1 hour or less

Preparing to Run a Batch Job

- Choose a cluster
- Compile and debug your code in an interactive session
  - Use login node to the extent practical
  - Not applicable if using system-installed software
- Determine resource requirements
  - nodes, memory, walltime, software licenses
- Create a batch script for the job
  - Script can have any valid filename
- Submit the job
- Job gets queued

Batch Script Overview

```
#PBS -N serial_fluent
#PBS -l walltime=1:00:00
#PBS -l nodes=1:ppn=1
#PBS -j oe
#PBS -l software=fluent+1
```

PBS headers

```
# Set up the FLUENT environment
module load fluent


# Move to directory job was submitted from
cd $PBS_O_WORKDIR


# Run fluent
fluent 3d -g < run.input
```

Executable commands

Put all this into a text file!

PBS (Batch) Options

- May appear on command line
- May appear at beginning of batch script
  - Before first executable line
  - Preceded by **`#PBS`**
- Resource requests
- Job name
- Output log preferences
- Mail options

Useful Options for Resource Requests

`-l nodes=`*`numnodes`*`:ppn=`*`numprocs`*	Number of nodes and processors per node. Can also specify gpus.
	`-l nodes=1:ppn=1`
	`-l nodes=5:ppn=12`

`-l mem=`*`amount`*	(optional - rarely needed) Request total amount of memory. `-l mem=192GB`
`-l walltime=`*`time`*	Total walltime limit in seconds or hours:minutes:seconds. `-l walltime=10:00:00`
`-l software=`*`package[+N]`*	(optional) Request use of N licenses for package. See software documentation for details. `-l software=abaqus+5`

- http://ondemand.osc.edu
- `ssh oakley.osc.edu`

Other Useful Options

`-N` *`jobname`*	Name you give the job
`-j oe`	Redirect stderr to stdout – get one log file rather than two.
`-m bea`	Mail options – send mail when job begins, ends, or aborts. Specify any combination of b, e, a.
`-M <my-email-address>`	Send logs to alternative email addresses

Batch Execution Environment

- Batch jobs begin execution in home directory
  - Even if you submit job from another directory
  - To get to directory submitted from:
    - `cd $PBS_O_WORKDIR`
- Environment identical to what you get when you log in
  - Same shell (unless you request a different one)
  - Same modules loaded
  - Appropriate “dot-files” executed
  - Must load any modules you need

Submitting a Job and Checking Status

- Command to submit a job
  - `qsub` *`script_file`*
- Response from PBS (example)
  - `123456.oak-batch.osc.edu`
- Show status of batch jobs (example)
  - `qstat -a 123456`
  - `qstat –u usr1234`
  - `qstat -f 123456`

Waiting for Your Job to Complete

- Job runs when resources become available
  - Optionally receive email when job starts
- Deleting a job
  - **`qdel 123456`**
  - Works for queued or running job

- `cp ~mfaerman/OSC-Batch-Training.tar.gz ~`
- `tar xzvf OSC-Batch-Training.tar.gz`

Monitoring a Running Job

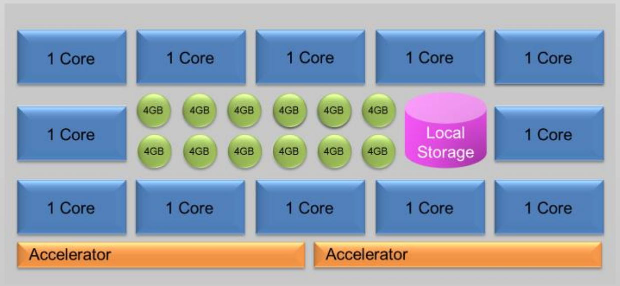
- To see the job output log (**`stdout`** and/or **`stderr`**)
  - **`qpeek 123456`**
  - See documentation for options
- To see resource utilization on nodes allocated to job
  - pdsh (Oakley)
  - all (Glenn)
  - See documentation
- Graphical representation of resource utilization
  - OSC OnDemand (ondemand.osc.edu)
  - Jobs → Active Jobs → Job Status

Considerations for Parallel Jobs

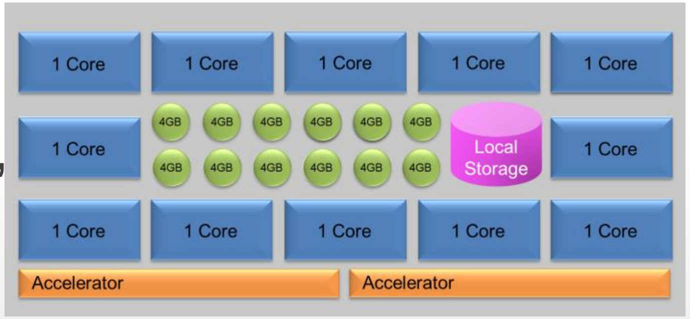
- Multiple Threads per process
  - Share single memory space
  - Leverage multiple cores within same node
  - OpenMP most common approach
- Multiple Processes on multiple nodes
  - Separate memory spaces
  - Data exchanged through messages
  - Message-Passing Interface (MPI) most common approach
- Multi-level parallelism may involve hybrid models
  - Multithreading
  - Message Passing
  - Accelerators
    - GPUS
    - Xeon Phi

$TMPDIR – The FASTEST (scratch)

- Data or executable files so large do not fit home directories.
- The `/tmp` directory offers a huge amount of **temporary** disk space (315TB in total)
  - **Much Faster** than `$HOME` disk since it is on **local disk** (not NFS-mounted).
- For each batch job – stored in the environment variable `TMPDIR`



$TMPDIR – The FASTEST (scratch)



- In the batch file the user should
  - copy all files needed to `$TMPDIR`,
  - `cd` to `$TMPDIR`,
  - run your code, and
  - finally bring needed output back files to your `$HOME` area.
- “clean-up” not needed
  - `$TMPDIR` directory and all its files are **deleted** when the job ends.

pbsdcp – Distributed Copy for Parallel Jobs

- $TMPDIR directory is not shared across nodes!
- When a parallel job starts running on multiple nodes, each node has its own $TMPDIR.
- Use pbsdcp when copying files to directories not shared between nodes (e.g. /tmp or $TMPDIR)
  - Distributed copy command
  - Two modes:
    - -s scatter mode (default)
    - -g gather mode

pbsdcp – Distributed Copy for Parallel Jobs

- Note: In gather mode, if files on different nodes have the same name, they will overwrite each other.
  - Using the -g (gather mode), the file names should have the form outfile001, outfile002, etc., with each node producing a different set of files.

$PFSDIR

- Large, Complex Data Structures
  - Spawning multiple nodes
  - Good candidates for $PFSDIR

- Utilizing Data Driven Software
  - MPI/IO
  - HDF5
  - NetCDF

- Removed when job terminates

PBS Information Variables

- PBS has a number of built-in environment variables that preserve job information:
  - `PBS_O_HOST` = hostname of machine running PBS
  - `PBS_O_QUEUE` = starting queue your job was put in
  - `PBS_QUEUE` = queue your job was executed in
  - `PBS_JOBID` = JID of your job
  - `PBS_JOBNAME` = “internal” name you gave job
  - `PBS_NODEFILE` = name of the file containing list of nodes your job used

- The next two slides show an example batch script and corresponding log depicting access to these PBS variables

Batch Script Reporting PBS Environment Information

```
#PBS -l walltime=1:00
#PBS -N print-env-var
#PBS -j oe
#PBS -m bae
#PBS -S /bin/bash


set -x
cd $PBS_O_WORKDIR
qstat -u $USER -rn
echo $PBS_O_HOST
echo $PBS_O_QUEUE
echo $PBS_QUEUE
echo $PBS_JOBID
echo $PBS_JOBNAME
cat $PBS_NODEFILE
```

Batch Log Reporting PBS Environment Information

```
+ cd /nfs/15/mfaerman/Training-UC/PBS-Environment
+ qstat -u mfaerman -rn

oak-batch.osc.edu:15001:
                                                                         Req'd  Req'd    Elap
Job ID                Username    Queue    Jobname          SessID NDS   TSK    Memory Time   S Time
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ ----- - -----
3109574.oak-batc      mfaerman    serial   print-env-var      4250     1      1    4gb 00:01 R    --
   n0678/0
+ echo oakley02.osc.edu
oakley02.osc.edu
+ echo batch
batch
+ echo serial
serial
+ echo 3109574.oak-batch.osc.edu
3109574.oak-batch.osc.edu
+ echo print-env-var
print-env-var
+ cat /var/spool/batch/torque/aux//3109574.oak-batch.osc.edu
n0678
```

Parallel Jobs
Script Issues

- Script executes just on the first node assigned to the job
- But how about my other nodes?
  - Use `mpiexec` to
    - Run copies of a program or command
    - On multiple nodes
- Software that also provides multi-node execution
  - `pbsdcp` (parallel file copy)
  - Some application software installed by OSC

Job Output

- Get your results when the job finishes
  - Optionally receive email when job ends
- Screen output ends up in file ***`job_name.ojobid`***
  - Copied to your working directory when job ends
  - Example: **`testjob.o1234567`**

Exercise

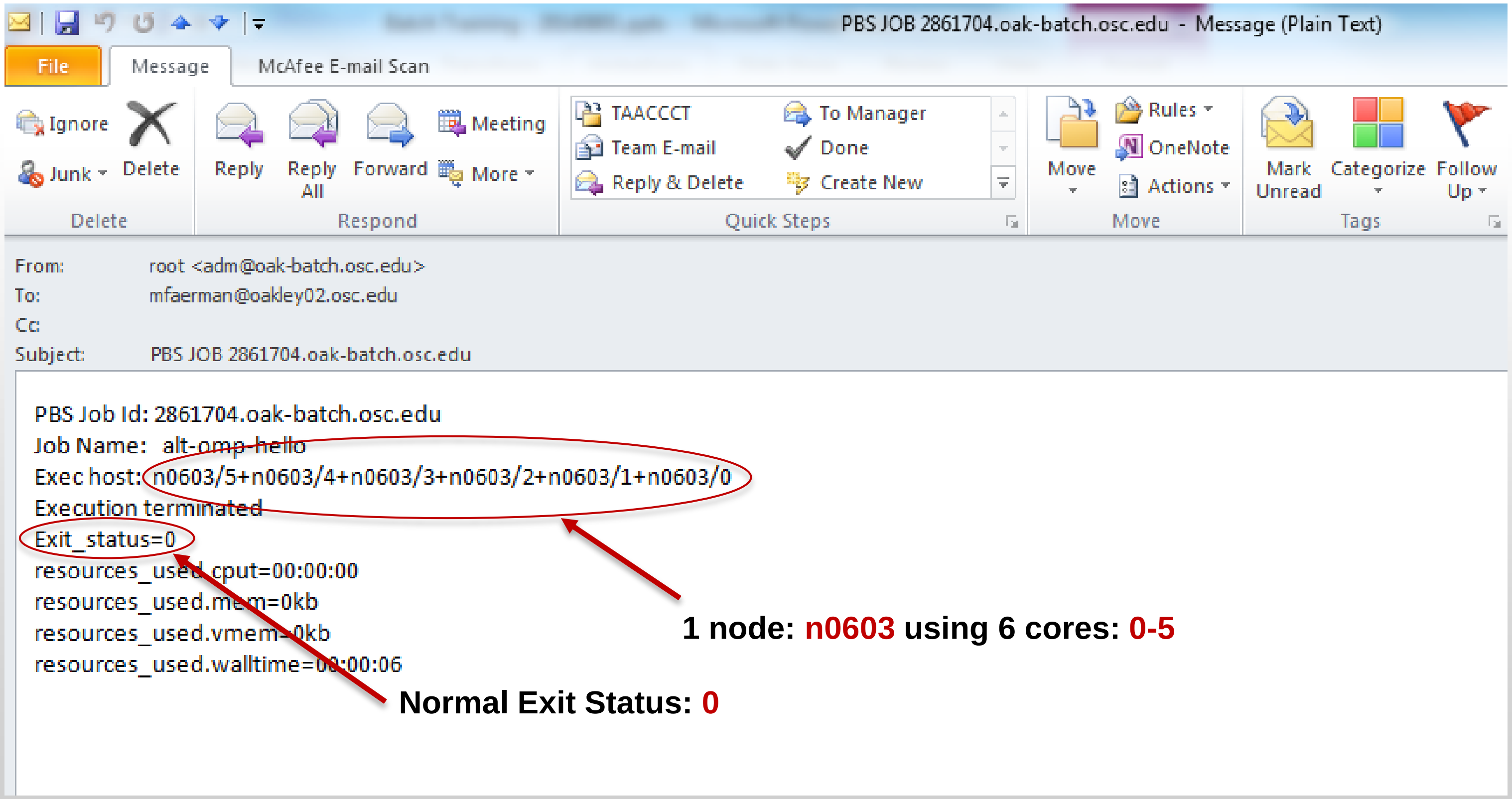
- Create and submit a serial job
  - Batch script is a text file – many options for creating
  - Select appropriate PBS headers – again, many options
  - Have the job print out the hostname and working directory, then sleep for 10 minutes
    - **`hostname; pwd; sleep 600`**
- Check the status of the job using **`qstat`**
- Check the job using OnDemand
- Take a peek at the output log using **`qpeek`**
- Optional: delete the job using **`qdel`**
- Find and display the output log(s)

OMP Job qstat sample

```
[mfaerman@oakley01 Simple_OMP_Job]$ qstat -u mfaerman

oak-batch.osc.edu:15001:
                                                                              Req'd  Req'd    Elap
Job ID                    Username    Queue    Jobname          SessID NDS   TSK    Memory Time   S Time
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ ----- - -----
2861061.oak-batc          mfaerman     serial   omp-hello            --      1     12   48gb 00:10 Q    --
[mfaerman@oakley01 Simple_OMP_Job]$
```

Sample of end of execution e-mail

Some useful information



From: root <adm@oak-batch.osc.edu>
To: mfaerman@oakley02.osc.edu
Cc:
Subject: PBS JOB 2861704.oak-batch.osc.edu

```
PBS Job Id: 2861704.oak-batch.osc.edu
Job Name:   alt-omp-hello
Exec host: n0603/5+n0603/4+n0603/3+n0603/2+n0603/1+n0603/0
Execution terminated
Exit_status=0
resources_used.cput=00:00:00
resources_used.mem=0kb
resources_used.vmem=0kb
resources_used.walltime=00:00:06
```

1 node: n0603 using 6 cores: 0-5

Normal Exit Status: 0

MPI Job qstat sample

```
[mfaerman@oakley02 Simple_MPI_Job]$ qstat -u mfaerman

oak-batch.osc.edu:15001:
                                                                         Req'd  Req'd   Elap
Job ID               Username    Queue    Jobname          SessID NDS   TSK    Memory Time  S Time
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ ----- - -----
2861557.oak-batc     mfaerman    parallel mpi-hello            --     4     48     --  00:10 Q   --
[mfaerman@oakley02 Simple_MPI_Job]$
```

Problems with Jobs Not Starting

- My job didn't start at all—why?
  - Are you logged on to correct machine?
  - Tricky part about shared storage is that all machines use same home directories
- Why is my job being held?
  - Check technical web pages (http://www.osc.edu/supercomputing)
  - Has a downtime been announced?
    - Scheduler will not run jobs that cannot finish before downtime

Problems with Jobs Failing after Starting

- My job quit before it finished—why?
  - Check for file ending with `.o`***`jobid`***
  - Study errors listed
    - Are errors from batch script?
      - oschelp may be of assistance
    - Are errors from programming problem?
      - oschelp can't really debug programs for users
- My job died with a segmentation fault—why?
  - Usually sign of trying to access an array out of bounds
  - Usually sign of a programming problem

Job Arrays

- Submission of many similar jobs
  - With single qsub

- unique $PBS_ARRAYID,
  - Parameterizes job behavior in array.
    - Input argument to an application
    - Part of a file name.

Job Array Script Example

```
#PBS -N test-array
#PBS -l walltime=00:00:30
#PBS -l nodes=1:ppn=1
#PBS -t 1-3,10,20
#PBS -j oe
#PBS -S /bin/bash

set -x

cd $PBS_O_WORKDIR

echo $PBS_ARRAYID

myprogram < data${PBS_ARRAYID}.in > data${PBS_ARRAYID}.out
```

How to qstat the whole Job Array

```
bash-4.1$ qstat -t '2862849[]'
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2862849[1].oak-batch       test-array-1     mfaerman                0 Q serial
2862849[2].oak-batch       test-array-2     mfaerman                0 Q serial
2862849[3].oak-batch       test-array-3     mfaerman                0 Q serial
2862849[10].oak-batch      test-array-10    mfaerman                0 Q serial
2862849[20].oak-batch      test-array-20    mfaerman                0 Q serial
```

How to qstat **specific jobs** in Job Array

```
bash-4.1$ qstat -t '2862849[1]'
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2862849[1].oak-batch       test-array-1     mfaerman                0 Q serial

bash-4.1$ qstat -t '2862849[2]'
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2862849[2].oak-batch       test-array-2     mfaerman                0 Q serial
```

How to **Remove** a specific job from Job Array

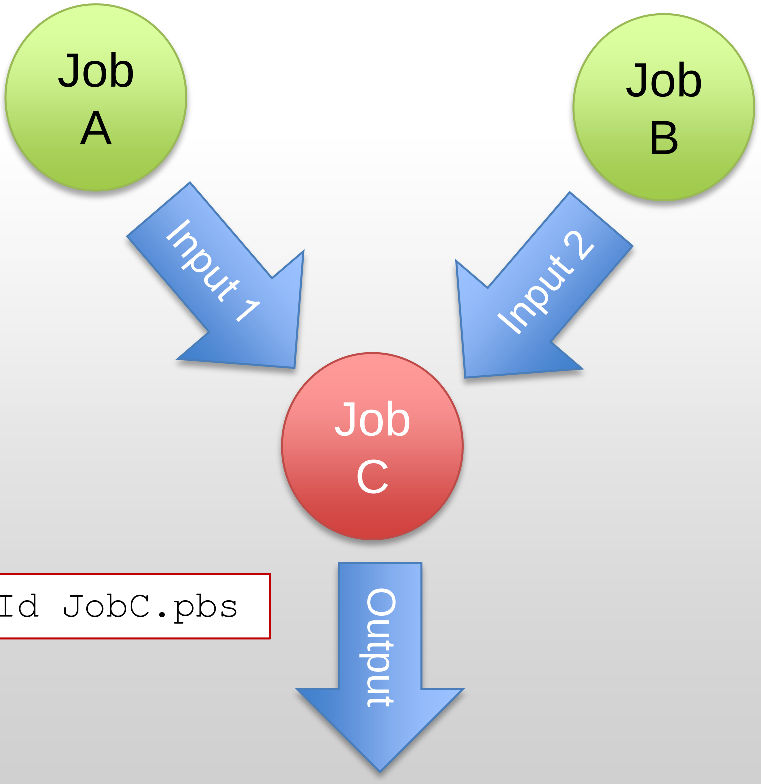
```
bash-4.1$ qstat -t '2862849[]'
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2862849[1].oak-batch       test-array-1     mfaerman               0 Q serial
2862849[2].oak-batch       test-array-2     mfaerman               0 Q serial
2862849[3].oak-batch       test-array-3     mfaerman               0 Q serial
2862849[10].oak-batch      test-array-10    mfaerman               0 Q serial
2862849[20].oak-batch      test-array-20    mfaerman               0 Q serial

bash-4.1$ qdel -t 2 '2862849[]`

bash-4.1$ qstat -t '2862849[]'
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2862849[1].oak-batch       test-array-1     mfaerman               0 Q serial
2862849[3].oak-batch       test-array-3     mfaerman               0 Q serial
2862849[10].oak-batch      test-array-10    mfaerman               0 Q serial
2862849[20].oak-batch      test-array-20    mfaerman               0 Q serial
bash-4.1$
```

Job Dependency

- Example:
  - Job C must not start before
  - Jobs A and B terminate

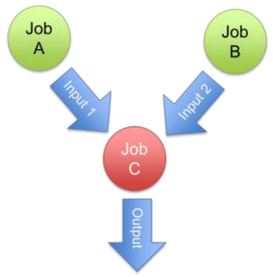
- Several conditional options available

```
qsub -W depend=afterany:$JobA_Id:$JobB_Id JobC.pbs
```



Job Dependency

Submission Example



```
[mfaerman@oakley02 Alt_OMP_Job]$ qsub -W depend=afterany:2865505:2865506 alt-omp-hello.pbs
2865507.oak-batch.osc.edu
[mfaerman@oakley02 Alt_OMP_Job]$ qstat -u mfaerman

oak-batch.osc.edu:15001:
                                                                          Req'd  Req'd    Elap
Job ID               Username    Queue    Jobname          SessID NDS   TSK    Memory Time   S  Time
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ ----- - -----
2865505.oak-batc     mfaerman    serial   alt-omp-hello        --     1      6   24gb 00:05 Q   --
2865506.oak-batc     mfaerman    serial   alt-omp-hello        --     1      6   24gb 00:05 Q   --
2865507.oak-batc     mfaerman    serial   alt-omp-hello        --     1      6   24gb 00:05 H   --
```

Licenses and Tokens
Abaqus Example

```
#PBS -N my_job
#PBS -l walltime=00:30:00
#PBS -l nodes=1:ppn=1
#PBS -l software=abaqus+5
module load abaqus
abaqus job=<abaqus_job> input=<input_file> interactive
```

- An Abaqus job needs T tokens to run
  - T = int(5 x C^0.422), where
  - C = total number of cores requested

- Tokens checked out from OSC token-based license pool

Cores (nodes x ppn each):	1	2	3	4	6	8	12	16	24	32	48
Tokens needed:	5	6	7	8	10	12	14	16	19	21	25

Abaqus Job Example

```
#PBS -l walltime=1:00:00
#PBS -l nodes=2:ppn=12
#PBS -N my_abaqus_job
#PBS -l software=abaqus+19
#PBS -j oe
#
# The following lines set up the ABAQUS environment
#
module load abaqus
#
# Move to the directory where the job was submitted
#
cd $PBS_O_WORKDIR
cp *.inp $TMPDIR/
cd $TMPDIR
#
# Run ABAQUS, note that in this case we have provided the names of the input files explicitly
#
abaqus job=test input=<my_input_file_name1>.inp cpus=24 interactive
#
# Now, move data back once the simulation has completed
#
mv * $PBS_O_WORKDIR
```

Considerations for Parallel Jobs

- Multiple Threads per process
  - Share single memory space
  - Leverage multiple cores within same node
  - OpenMP most common approach

- Multiple Processes on multiple nodes
  - Separate memory spaces
  - Data exchanged through messages
  - Message-Passing Interface (MPI) most common approach

- Multi-level parallelism may involve hybrid models
  - Multithreading
  - Message Passing
  - Accelerators
    - GPUS
    - Xeon Phi

Hybrid MPI, OpenMP Job Script

6 threads/process, 4 MPI processes, 2 nodes

```
#PBS -N hybrid-mpi-omp-2x4d2
#PBS -l walltime=00:01:00
#PBS -l nodes=2:ppn=12
#PBS -j oe
#PBS -m bae
#PBS -S /bin/bash

module swap intel gnu
set -x

export OMP_NUM_THREADS=6
export MV2_ENABLE_AFFINITY=0

cd $PBS_O_WORKDIR
pwd

# Compile in $PBS_O_WORKDIR, printed above.
mpicc -O2 -fopenmp hello-hybrid.c -o hello-hybrid

# Copy executable to all nodes
pbsdcp $PBS_O_WORKDIR/hello-hybrid $TMPDIR
mpiexec -npernode 2 $TMPDIR/hello-hybrid
```

MPI-OpenMP Sample Output

6 threads/process, 4 MPI processes, 2 nodes

```
[mfaerman@oakley02 Hybrid-MPI-OpenMP]$ grep Hello hybrid-mpi-omp-2x4d2.o2879820
Hello from thread 0 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
Hello from thread 3 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 4 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
Hello from thread 0 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 0 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 5 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 3 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
Hello from thread 4 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 1 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 3 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 2 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 1 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 2 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 2 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 5 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 5 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 4 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 4 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 0 out of 6 from process 3 out of 4 on n0401.ten.osc.edu
Hello from thread 1 out of 6 from process 2 out of 4 on n0401.ten.osc.edu
Hello from thread 3 out of 6 from process 1 out of 4 on n0599.ten.osc.edu
Hello from thread 5 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
Hello from thread 2 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
Hello from thread 1 out of 6 from process 0 out of 4 on n0599.ten.osc.edu
```

Batch Specifics

- 8 Large Memory (192 GB) nodes on Oakley ("bigmem").
  - `#PBS -l mem=192GB`
- Huge Memory node ("hugemem"), with 1 TB of RAM and 32 cores
  - `#PBS -l nodes=1:ppn=32.`
  - This node is only for serial jobs, must request the entire
  - Walltime limit of 48 hours for jobs on this node.
- GPU jobs may request any number of cores and either 1 or 2 GPUs.

Interacting with OSC Nodes

- Login Nodes
  - Just ssh to cluster login nodes
  - Limited time and computational resources

- OnDemand Portal
  - Easy access to Graphic User Interface (GUI) software
    - Just open a VNC App
      - Desktops
      - Applications

Interacting with a Batch Job

- Yes – you wait in line to run your job

- But once you get out of the queue:
  - You have access to the batch nodes
  - Can actually **interact** with them
    - For instance, using VNC

    - Further information available at:
      https://www.osc.edu/documentation/howto/use-vnc-in-a-batch-job

Interactive Batch Jobs

- Useful for debugging parallel programs
- Running a GUI program too large for login or desktop nodes.
- Resource limits (memory, CPU) same as batch limits
- Generally invoked without a script, for example:

```
qsub -I -X -l nodes=2:ppn=12 -l walltime=1:00:00
```

  - The `-I` flag indicates job is interactive
  - The `-X` flag enables X11 forwarding
  - Need X11 server running on your computer to use X11 forwarding [see more]

Starting your VNC server

Option 1: Interactive Shell

- In your job submission, request:
  - Entire GPU node,
    - GPUs used to accelerate visualization

```
qsub -I -l nodes=1:ppn=12:gpus=2:vis
```

- Your job will still be **queued** just like any job

```
qsub: waiting for job 123456.opt-batch.osc.edu to start
```

- When the job runs, you'll see the following line:

```
qsub: job 123456.opt-batch.osc.edu ready
```

- You now have an **Interactive Shell**
  - On one of the GPU nodes

⚠ Interactive PBS Shell – An Important Note ⚠

- If the load is high,
  - ➡ Your job may wait for hours in the queue

- A walltime limit ≤ 1 hour recommended
  - – As job can run on nodes reserved for debugging

Starting your VNC server

Option 1: Interactive Shell

- Start the VNC server

```
module load virtualgl
module load turbovnc
vncserver
```

- May ask to setup password
  - To secure VNC session from unauthorized connections
  - We recommend a strong password
- The output of this command is important

```
New 'X' desktop is n0302.ten.osc.edu:1
```

  - Tells where to point client to access desktop
    - Host Name (before the `:`)
    - Display # (after the `:`)

Starting your VNC server

Option 2: Non-Interactive Batch Job

- Less Friendly
  - Use `qpeek` to verify the output of `vncserver`
    - Host Name
    - Display #

- More Robust
  - Can go away (no “baby-sitting” of interactive prompt)
    - System **notifies by email** when desktop is available
  - If connection to OSC is unstable and intermittent
    - VNC server survives disconnection

Starting your VNC server

Option 2: Non-Interactive Batch Job

Script Sample:

```
#PBS -l nodes=1:ppn=12:gpus=2:vis
#PBS -l walltime=00:15:00
#PBS -m b
#PBS -N VNCjob
#PBS -j oe

module load virtualgl
module load turbovnc

vncserver

sleep 100

vncpid=`pgrep -s 0 Xvnc`

while [ -e /proc/$vncpid ]; do sleep 0.1; done
```

Starting your VNC server

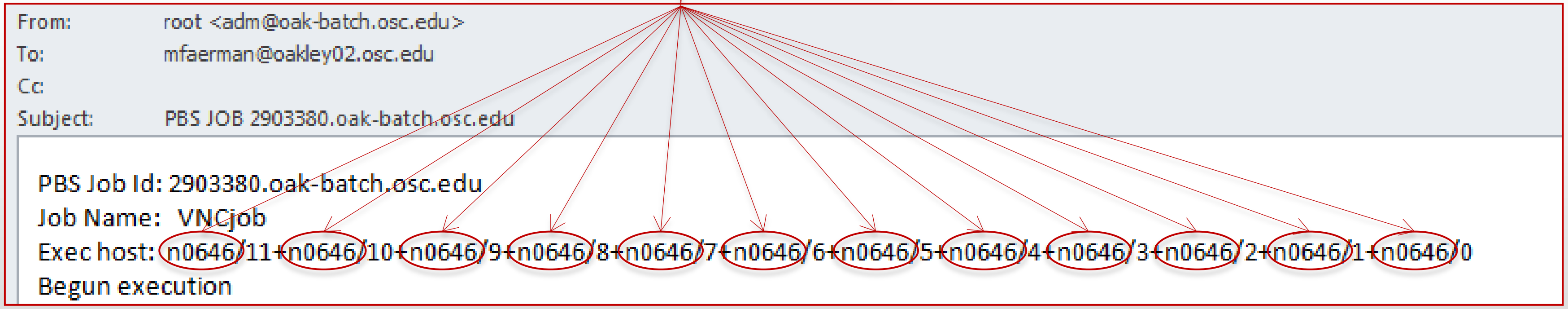
Option 2: Non-Interactive Batch Job

```
-bash-4.1$ vncpasswd
Password:

-bash-4.1$ qsub int-nogpus.pbs
3092450.oak-batch.osc.edu
```

Starting your VNC server

Option 2: Non-Interactive Batch Job

- Script submission sends an email when job has started
  - Includes the host (node) name: "`n0646`"



```
From:       root <adm@oak-batch.osc.edu>
To:         mfaerman@oakley02.osc.edu
Cc:
Subject:    PBS JOB 2903380.oak-batch.osc.edu

PBS Job Id: 2903380.oak-batch.osc.edu
Job Name:   VNCjob
Exec host:  n0646/11+n0646/10+n0646/9+n0646/8+n0646/7+n0646/6+n0646/5+n0646/4+n0646/3+n0646/2+n0646/1+n0646/0
Begun execution
```

- Use qpeek to check output of vncserver on a login node:
  - The `display#` is virtually always "`1`"

```
[mfaerman@oakley02]$ qpeek 2903380
New 'X' desktop is n0646.ten.osc.edu:1

Starting applications specified in /nfs/15/mfaerman/.vnc/xstartup.turbovnc
Log file is /nfs/15/mfaerman/.vnc/n0646.ten.osc.edu:1.log
```

Connecting to your VNC server

- In both Interactive an Batch options
- Cluster compute nodes not directly accessible

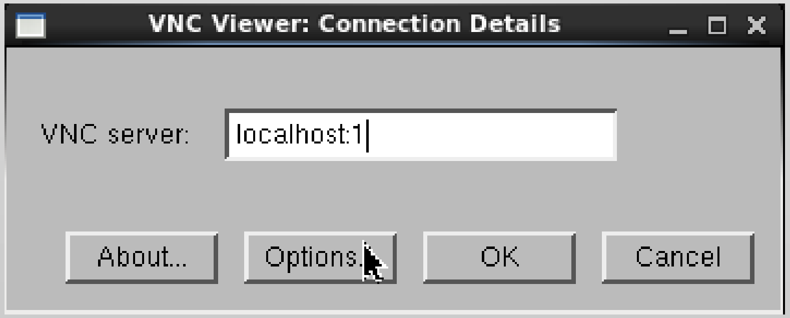
- Must log into login node
  - Allow VNC client to "tunnel" through SSH to compute node.
    - The method of doing so *may vary on client software*.

Linux/MacOS example to Oakley

Manually create an SSH tunnel

```
ssh -L 5901:n0646.ten.osc.edu:5901 mfaerman@oakley.osc.edu
```

- Issue this command in new terminal window on your local machine, creating a new connection to Oakley.
- Open your VNC client and connect to "localhost:1"
  - This will tunnel to the correct node on Oakley


VNC Viewer: Connection Details

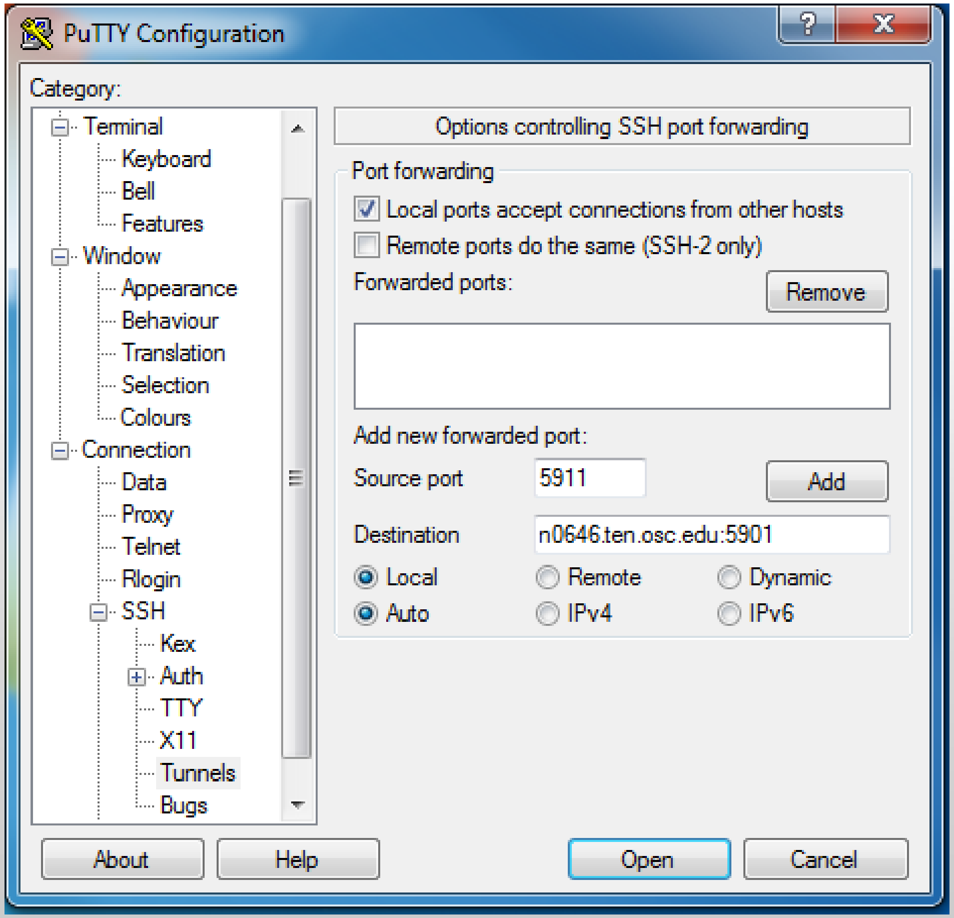
VNC server: localhost:1

Putty/Windows example to Oakley

- Enable X11 Forwarding
- At SSH Tunnels settings
  - Pick Source port
    - Between 5911 and 5999
  - Set Destination
    - From `vncserver` output

```
<Host Name>:<5900+display#>
```

  - Click “Add” button
- SSH to cluster login node
  - Where `vncserver` is running



PuTTY Configuration

Category: Terminal > Keyboard, Bell, Features; Window > Appearance, Behaviour, Translation, Selection, Colours; Connection > Data, Proxy, Telnet, Rlogin, SSH > Kex, Auth, TTY, X11, Tunnels, Bugs

Options controlling SSH port forwarding

Port forwarding
- [✓] Local ports accept connections from other hosts
- [ ] Remote ports do the same (SSH-2 only)

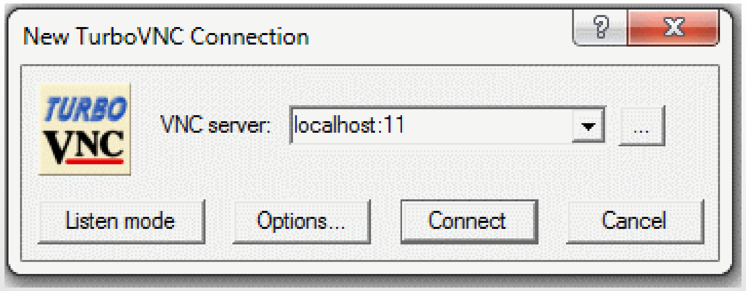
Forwarded ports:

Add new forwarded port:
- Source port: 5911
- Destination: n0646.ten.osc.edu:5901
- [● Local] Remote Dynamic
- [● Auto] IPv4 IPv6

VNC Client

Windows Example

- Enter `localhost:[port]`
  - Replacing `[port]` with the port between 11-99 chosen earlier.
- TurboVNC is recommended



New TurboVNC Connection

VNC server: localhost:11

- If you've set up a VNC password you will be prompted for it now
- A desktop display should pop up now if everything is configured correctly.

Further Considerations

- Advanced Reservations
  - Known Start Time
  - Interactive Sessions
  - Reservations are charged

- Condo Model
  - Shared cost
    - Users and OSC
    - Purchase or Rental
  - Win-Win Framework
    - Skip the line!
    - Exclusive access to user dedicated resources
    - Operational Costs Reduction

For More Information

- www.osc.edu/supercomputing/batch-processing-at-osc
- Contact oschelp@osc.edu with any questions or problems
- Additional example code available:
  - `cp -bvia ~mfaerman/Training-UC/ .`

Marcio Faerman
mfaerman@osc.edu
614-292-2819

Ohio Supercomputer Center

An OH·TECH Consortium Member

Additional Infrastructure Details

Login Nodes – Configuration

- Oakley
  - 2 general-purpose login nodes
  - 12 cores, 124 GB memory each
  - Connect to oakley.osc.edu
- Glenn
  - 4 general-purpose login nodes
  - 8 cores, 32 GB memory each
  - Connect to glenn.osc.edu

Compute Nodes – Oakley

- 684 standard nodes
  - 12 cores per node
  - 48 GB memory (4GB/core)
  - 812 GB local disk space
- 8 large memory nodes
  - 12 cores per node
  - 192 GB memory (16GB/core)
  - 812 GB local disk space
- Network
  - Nodes connected by 40Gbit/sec Infiniband network (QDR)

Special Resources

- GPU computing
  - 128 NVIDIA Tesla M2070 GPUs
  - 64 of the standard nodes have 2 GPUs each
- 1 huge memory node
  - 32 cores
  - 1 TB memory
- Intel Xeon Phi accelerators (Ruby cluster)
  - 8 nodes, each with one Phi card
  - limited-access test cluster

Compute Nodes – Glenn

- 634 standard nodes
  - 8 cores per node
  - 24 GB memory (3GB/core)
  - 393 GB local disk space
- Network
  - Nodes connected by 20Gbit/sec Infiniband network (DDR)

Special Resources – Glenn

- GPU computing
  - 18 NVIDIA Quadro Plex S4 systems
  - Each Quadro Plex S4 has 4 Quadro FX GPUs
  - 36 of the standard nodes have 2 GPUs each