Biowulf: A Beowulf for Bioscience

Steven Fellini
sfellini@nih.gov

Susan Chacko
susanc@helix.nih.gov

staff@helix.nih.gov
Helix Systems Staff, CIT
Oct 14, 2015

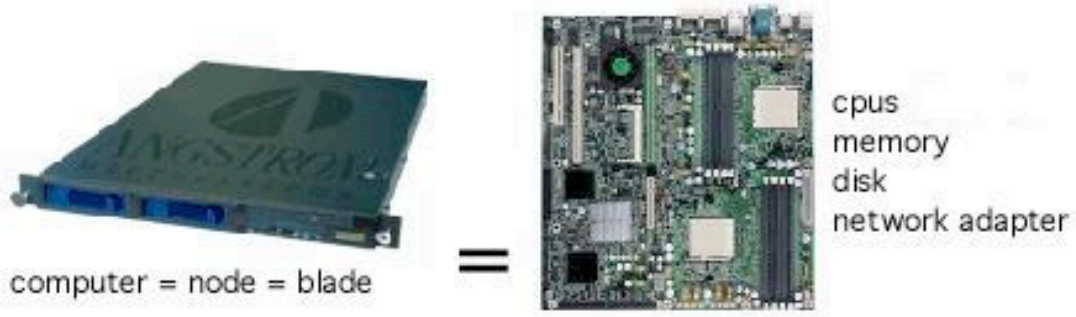
Slides available as *http://hpc.nih.gov/docs/biowulf-seminar-14oct2015.pdf*

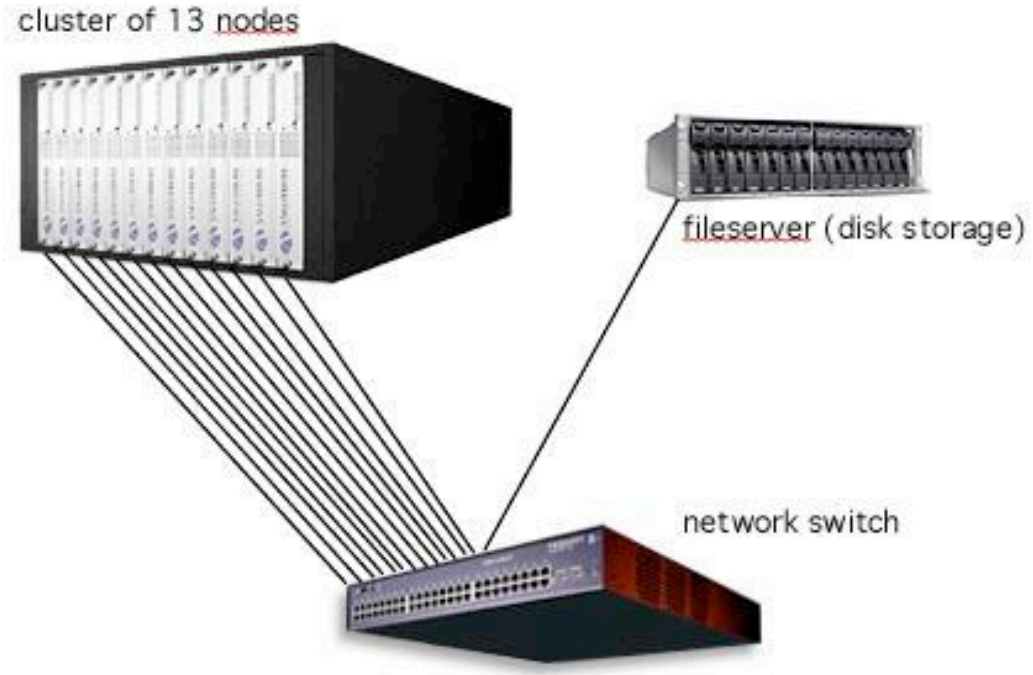
Introduction to Biowulf

- Cluster Basics & Concepts
- Biowulf architecture & hardware configuration
- Why use Clusters/Biowulf?
- Administrivia: Accounts & Passwords, Connecting to Biowulf; Email
- Data Transfer & Storage
- Running Jobs & the Batch System
- Monitoring Jobs

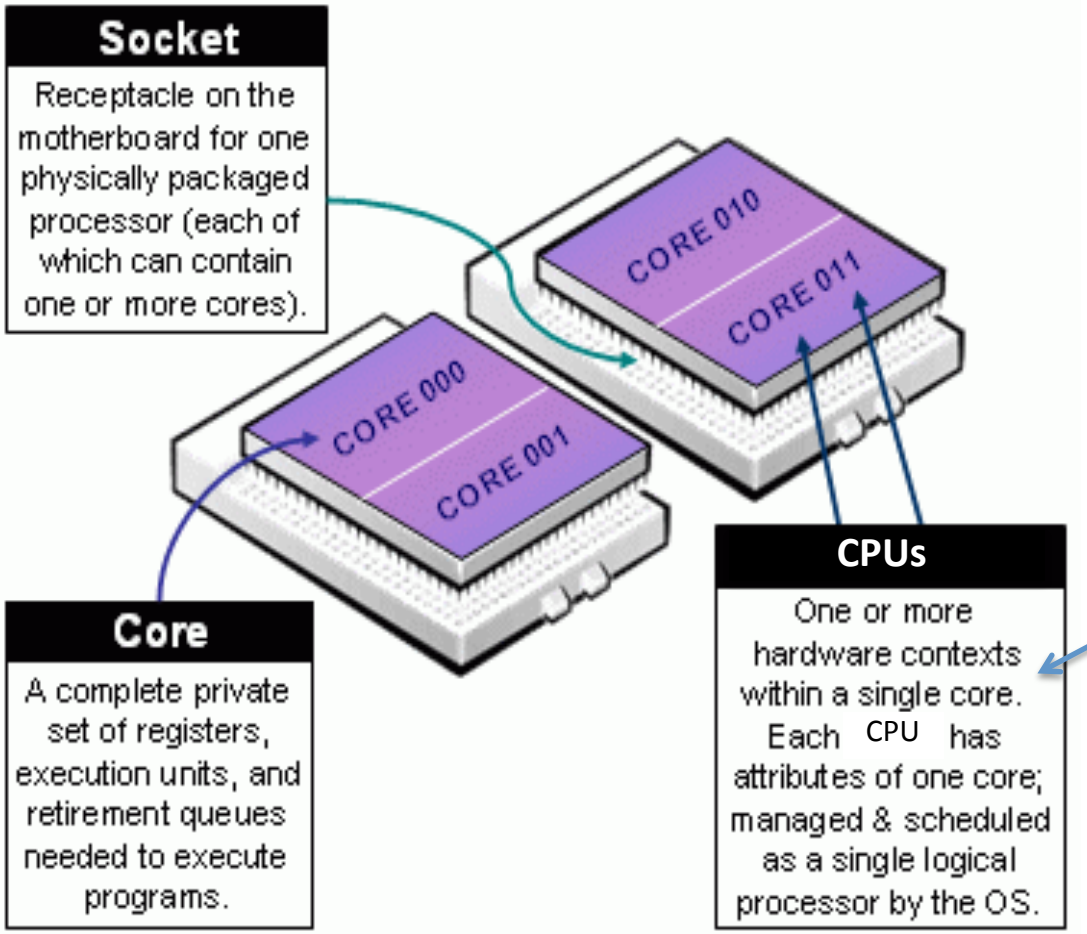
Cluster Basics
& Biowulf

Cluster Basics

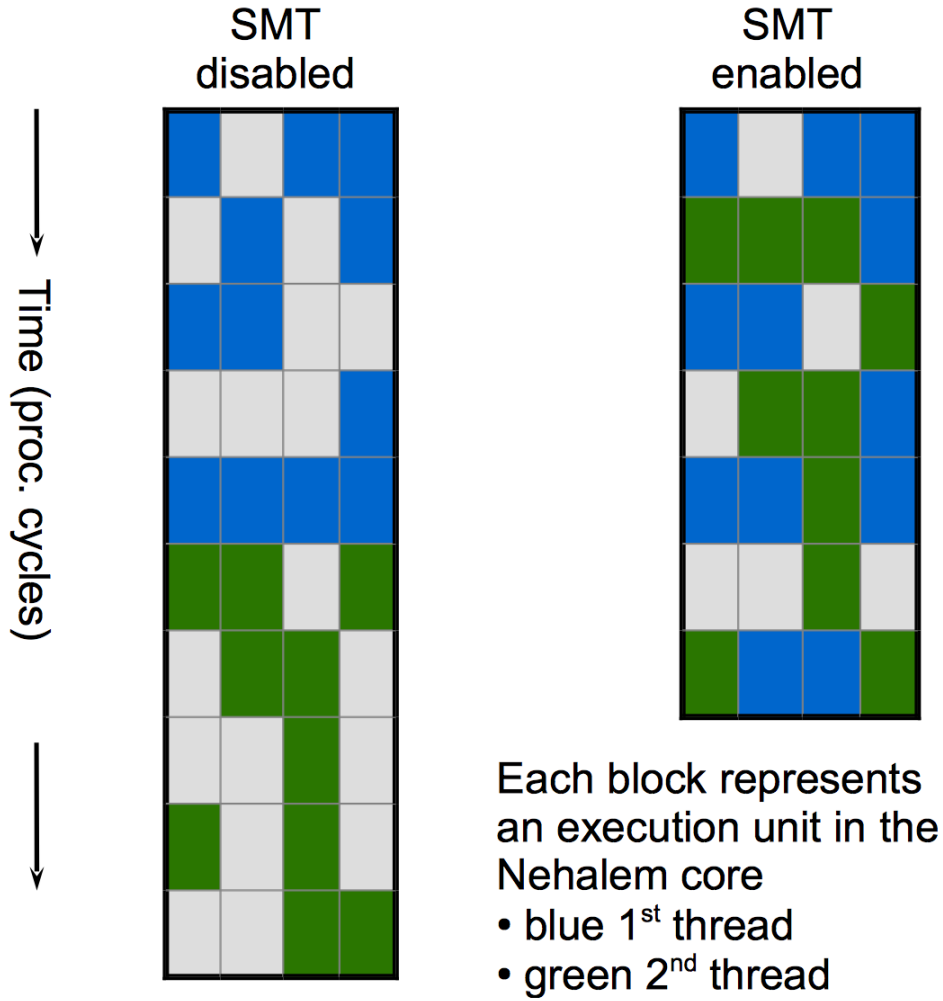




Hardware Terminology



Hyper-threading

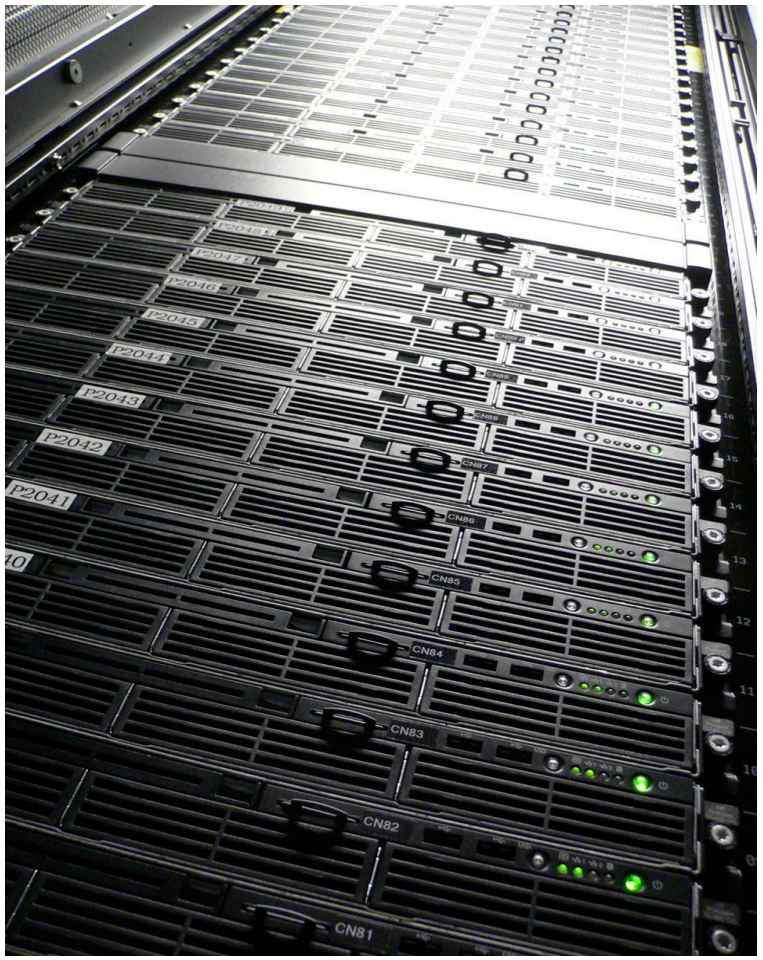


Each block represents an execution unit in the Nehalem core

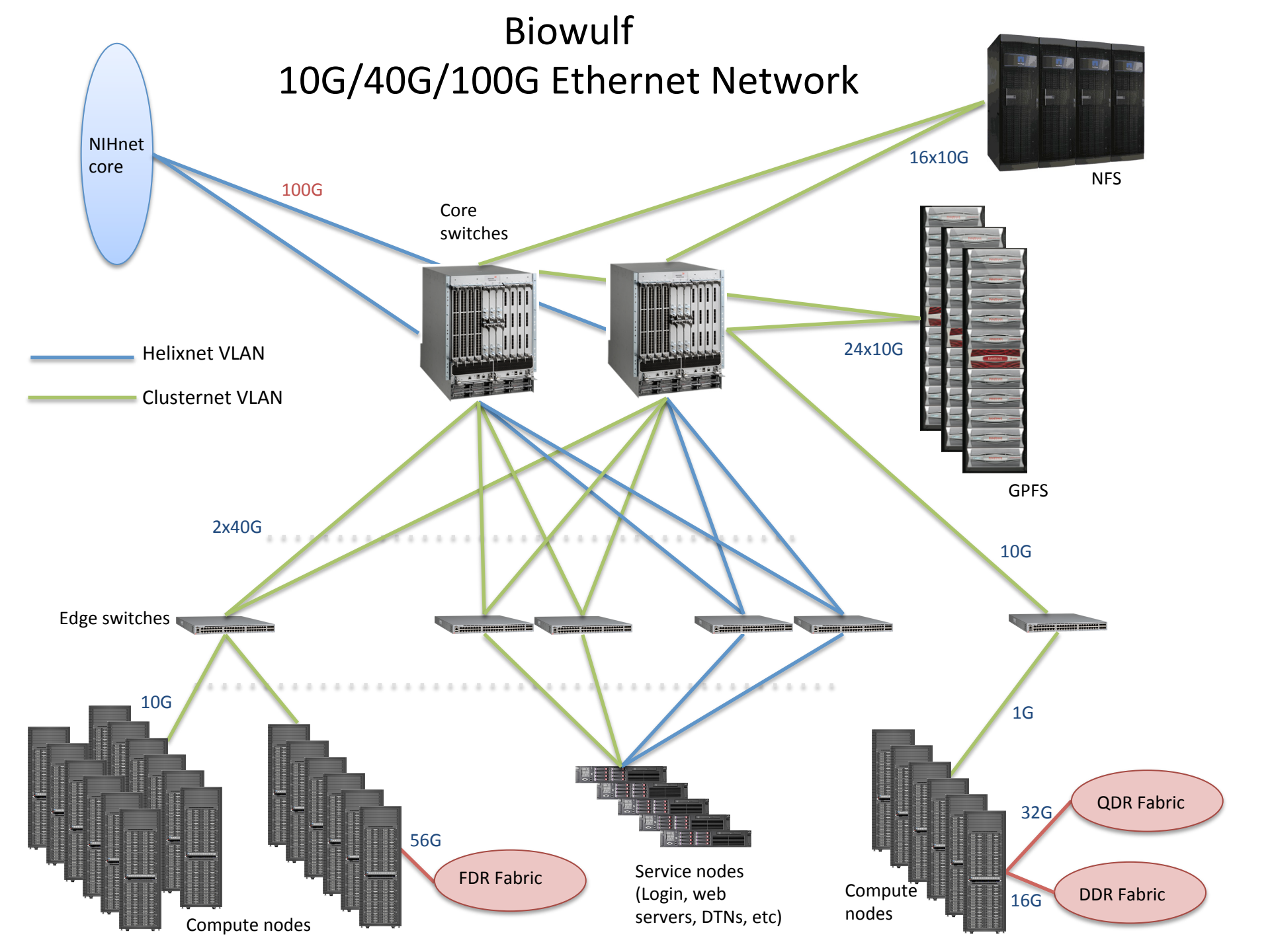
- blue 1st thread
- green 2nd thread

The NIH HPC Core Facility (Biowulf)

- Central scientific compute resource managed by CIT Helix Staff
- Funded by NIH Management Fund
- Available to all NIH intramural scientists
- Production facility: high availability, data integrity
- Large scale: 28,000+ processor cores; Petabytes storage capacity
- Enabling research not otherwise possible
- General purpose scientific computing (not dedicated to any one application type)
- Cited by 150+ publications/year



Biowulf
10G/40G/100G Ethernet Network



Compute Nodes

# nodes	processor, cores per node	memory	network
324	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10Gb/s Ethernet
192	Intel E5-2650v2 16 x 2.6 GHz	64 GB	56 Gb/s FDR Infiniband 10Gb/s Ethernet
24	Intel E5-2650v2 16 x 2.6 GHz **2 x Nvidia K20x**	128 GB	56 Gb/s FDR Infiniband 10Gb/s Ethernet
4	Intel E5-4620v2 32 x 2.6 GHz	1 TB	10Gb/s Ethernet
362	Intel X5660 12 x 2.8 GHz	24 GB	1 Gb/s Ethernet
302	Intel E5-5550 16 x 2.67 GHz	24GB	1 Gb/s Ethernet
218	Intel E5462 8 x 2.8 GHz (hyper-threading not enabled)	8 GB	16 Gb/s DDR Infiniband 1 Gb/s Ethernet

Buy-In Nodes

# nodes	processor, cores per node	memory	network	funding
86	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10Gb/s Ethernet	NIDDK
224	Intel E5-2670 16 x 2.6 GHz	128 & 64 GB	1 Gb/s Ethernet	CCR (NCI)
64	Intel E5-2670 16 x 2.6 GHz	64 & 32 GB	32 Gb/s QDR Infiniband 1 Gb/s Ethernet	NIMH
206	Intel E5-2670 16 x 2.6 GHz	64 & 32 GB	32 Gb/s QDR Infiniband 1 Gb/s Ethernet	LCP (NIDDK)

- Priority given to funding Lab/IC
- Available via *quick* queue to all NIH users

28,968 Compute Cores

57,936 CPUs

http://hpc.nih.gov

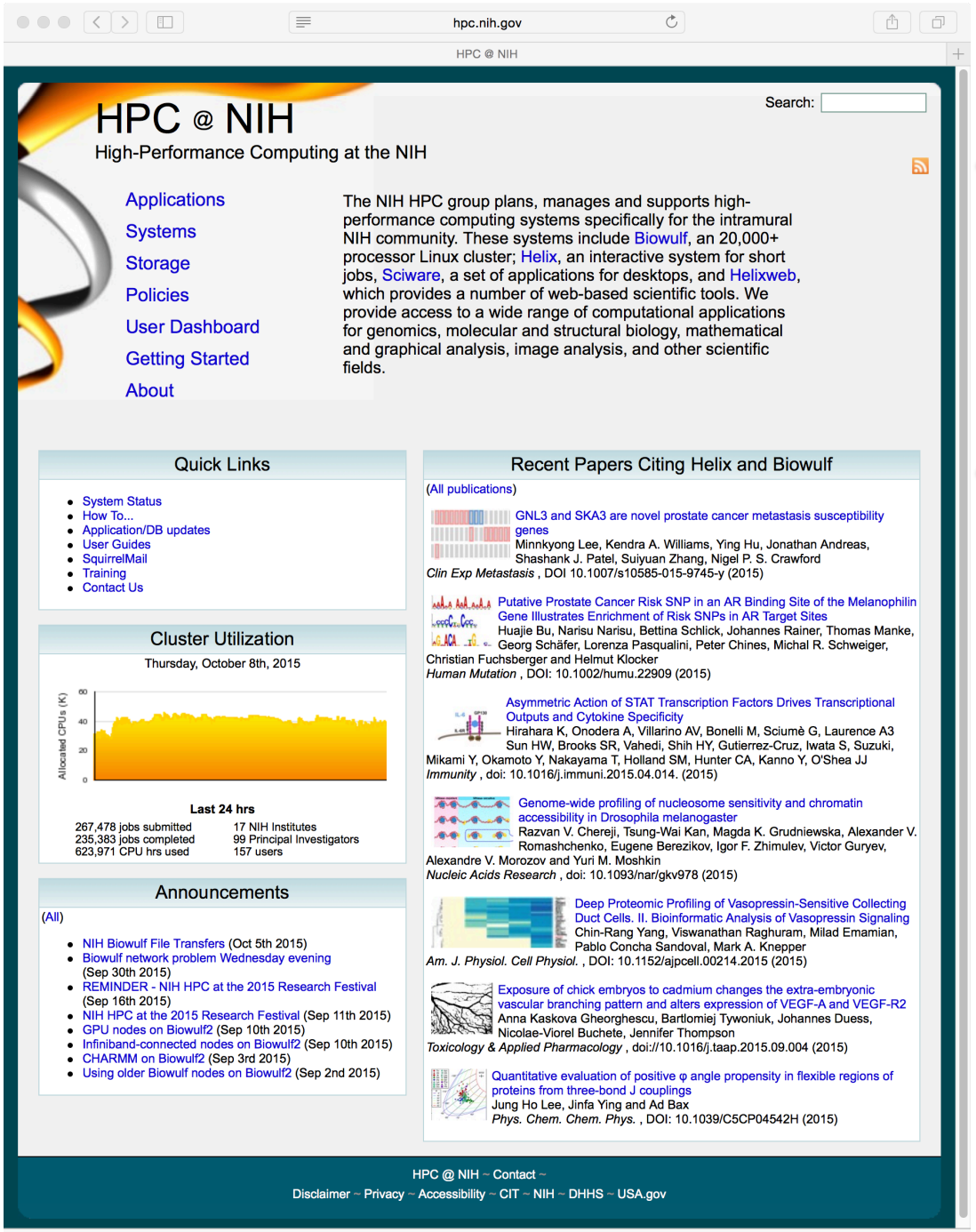
http://hpc.nih.gov/apps

User Guides

Announcements

RSS

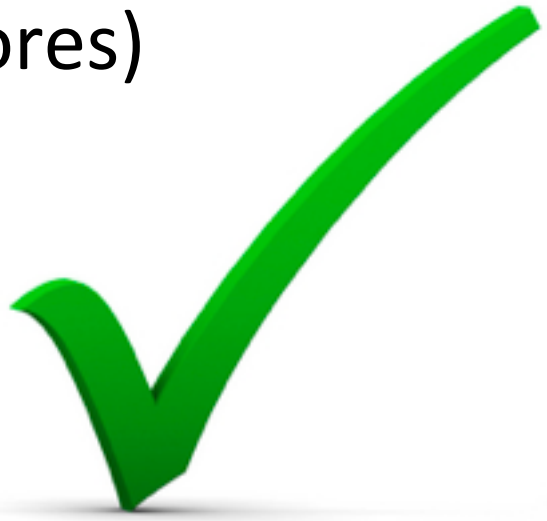
http://hpc.nih.gov/research



Why would you want to use Biowulf?

- Large numbers of jobs (bioinformatics on thousands of sequences)
- Large-memory jobs (genome assemblies)
- Parallel jobs with high-cpu demands (e.g., molecular dynamics on 1024 cores)
- Terabytes of data to process

… in other words, LARGE SCALE



Unsuitable for Biowulf…(or, why bother?)

- Small numbers of jobs
- One dependent job after the other
- Interactive jobs, graphics etc.

… in other words, desktops can be pretty powerful! (Helix too)

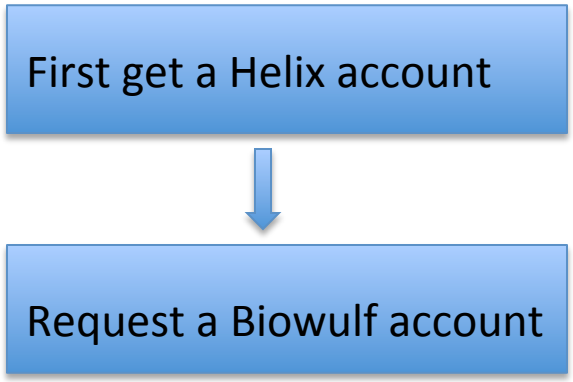
Class survey

- Level of Unix/Linux experience?
- Helix account?
- Biowulf account?
- Your research application?

Accounts, passwords etc.

Accounts

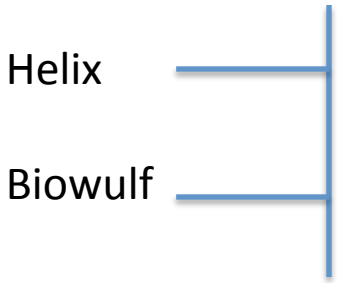
No sharing of accounts!



http://helix.nih.gov/Documentation/accounts.html
$33/month

https://helix.nih.gov/accounts/account_request.php
No additional charges
No usage charges

Accounts



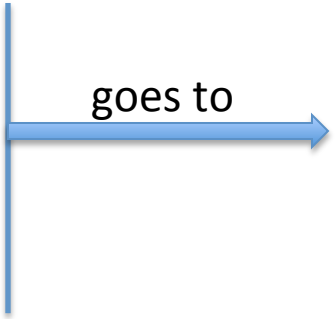
Same username (NIH login username)
Same password (NIH login password)
Same /home area
Same /data area

Need not have the same shell, but we recommend ‘bash’ for both.

You may get locked due to inactivity on any system: send email to staff@helix.nih.gov to get unlocked.
(soon: unlock yourself at https://helix.nih.gov/unlock)

Email

Mail from:
- Biowulf batch system
- Biowulf sysadmins
- Notifications
- Helix announcements



username@helix.nih.gov

If you're not reading your Helix email, forward it!!!!

Hands-on
Connecting to Biowulf

Macs/Linux: ssh *your-username*@biowulf.nih.gov

Windows: open a putty session to biowulf.nih.gov

- Use your own username/password.
- If you don't have an account, ask the instructor for a student account.

Use of Biowulf's login node

- Submitting jobs
- Editing/compiling code
- File management
- File transfer
- **Brief** testing of code or debugging (under 20 minutes cpu time)

Do not run compute jobs of any kind on the login node!

root Today at 10:26 AM

To: [redacted]@biowulf2.nih.gov

No compute intensive processes on Biowulf login node

Compute or large memory jobs are not allowed on the Biowulf login node.
Such jobs can hang the login node and/or seriously affect all users.
If you persist in running such jobs, your account will be locked.

Please use batch or allocate an interactive node if needed. Here is
how easy it is to get a dedicated interactive session:

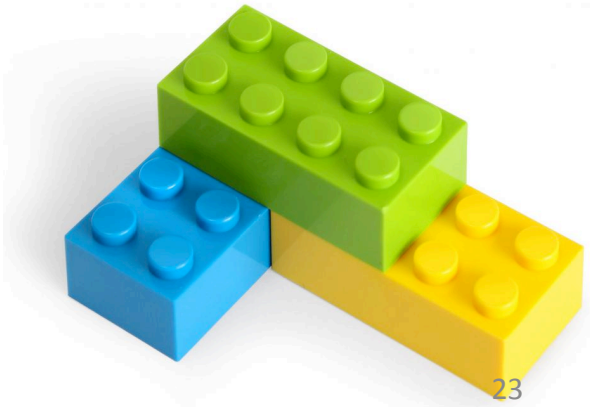
% sinteractive

Process terminated:

```
 PID USER            NI       VSZ       RSS S %CPU %MEM       TIME COMMAND
23583 [redacted]       0   5973256   5463320 R 96.9  4.4   00:01:44 bedtools sort -i KZ946_CLP.bed
```

Environment Modules

- Dynamically set up environments (paths, library locations, etc.) for different applications
- One easy command for setup (not shell-dependent)
- http://hpc.nih.gov/apps/modules.html



wiseGEEK

What modules exist?

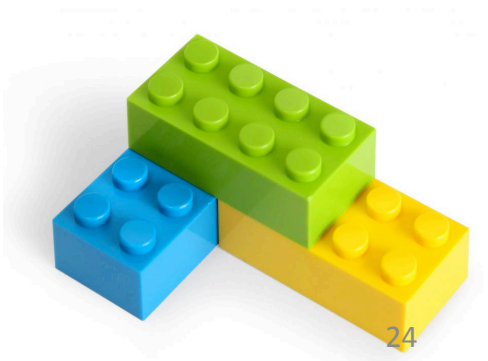
module avail

module avail appname

Examine a module

module display appname

module display appname/version



wiseGEEK

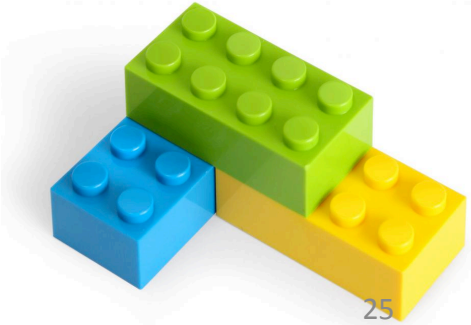
Load a module

module load appname

module load appname/version

See loaded modules

module list



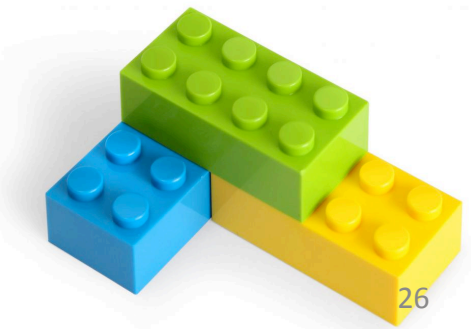
wiseGEEK

Unload a module

module unload appname

Personal modules

module load use.own



wiseGEEK

Modules

To see default versions of all apps, use:
module –d avail

For a case-insensitive module search, use:
module spider

Example:

```
[biowulf2 ~]$ module spider xplor-nih

-----------------------------
  Xplor-NIH: Xplor-NIH/2.39
-----------------------------
```

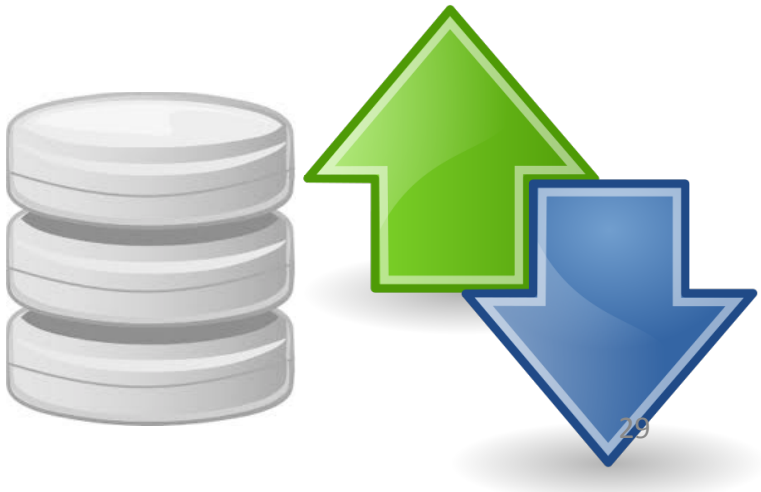
Hands-on (modules)

```
module avail
module avail gromacs
```

(or any application of your choice)

```
module load samtools
module list
echo $PATH
module display bedtools
```

Data transfer and storage



Transferring files

http://hpc.nih.gov/docs/transfer.html

- Map your Helix /home or /data area on your desktop.
- GUI File transfer clients (WinSCP etc.)
- Commandline (`scp`, `sftp`,`bbcp`)
- Remember: Windows files have Ctrl-M characters as linefeeds (`dos2unix to fix the file`)

Web Proxy

- Allows limited Internet access *from compute nodes*
- Transparent for programs using HTTP_PROXY, HTTPS_PROXY, RSYNC_PROXY or FTP_PROXY environment variables
- Transfers through Data Transfer Nodes
- wget, curl, lftp, rsync, git clone
- (NCBI SRAToolkit, Hisat)

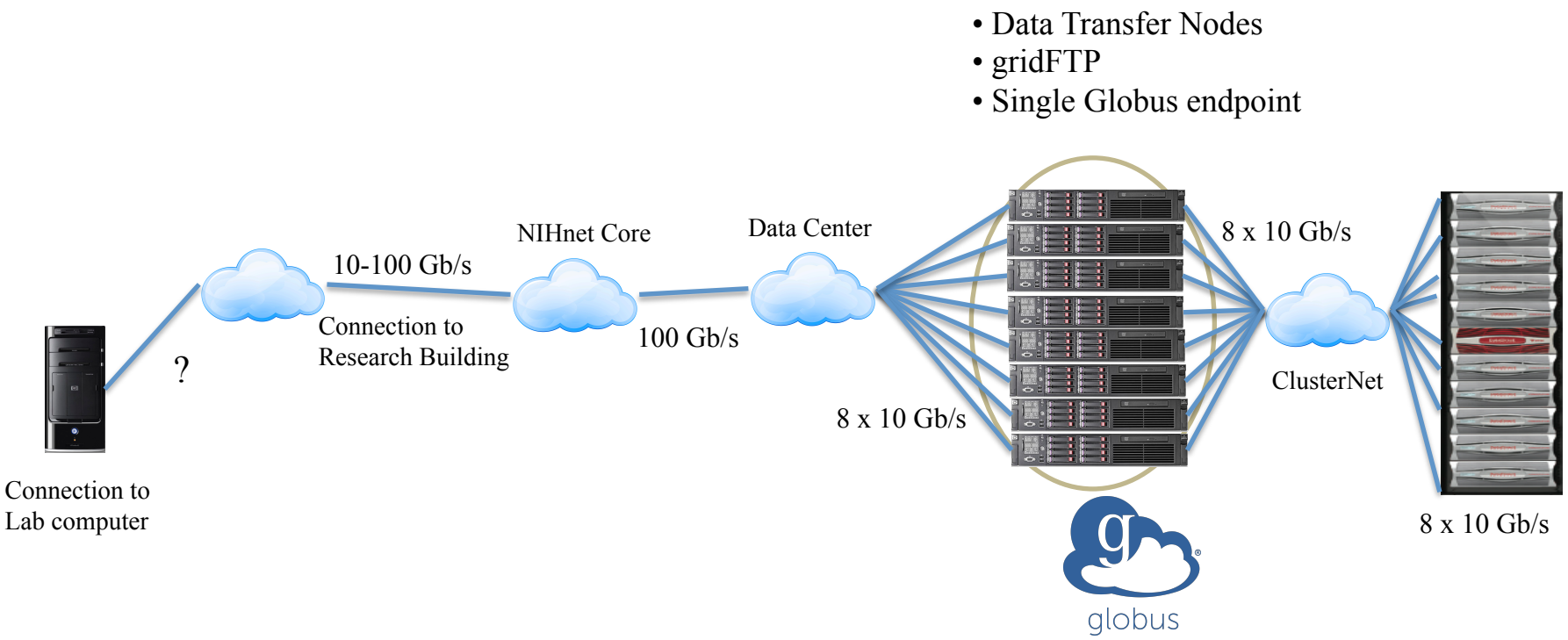
Transferring files

https://hpc.nih.gov/docs/globus.html

Sharing data with collaborators

https://hpc.nih.gov/docs/sharing_data.html

High Performance Data Transfers



Disk storage

	Location	Creation	Backups?	Amount of space	Accessible from (*)
/home	fileservers	with Helix account	snapshots & tape	8 GB (quota)	B, H, C
/data	fileservers	with Biowulf account	snapshots	100 GB (quota)	B, H, C
/scratch	fileservers	created by user	No	20 TB shared by all users	B, H
/lscratch	local disk	created by batch job	No	varies, up to 800 GB shared	C

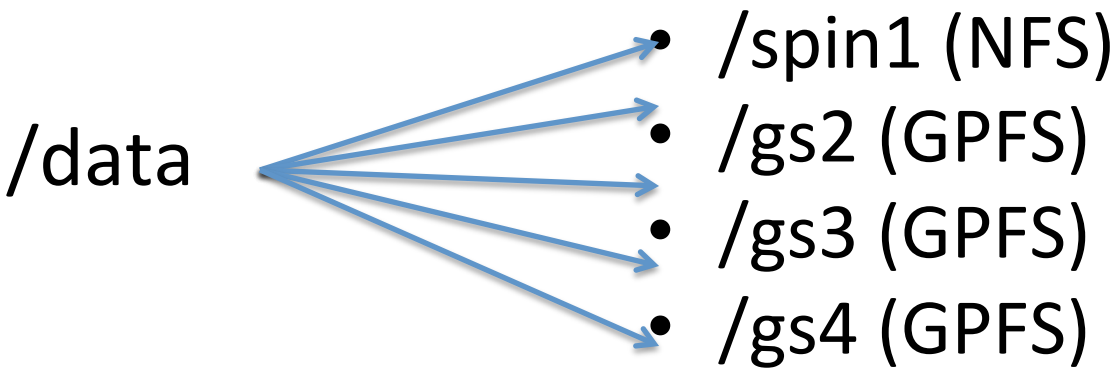
(*) H = helix, B = biowulf login node, C = biowulf computational nodes

Disk storage

- *checkquota* command to check usage
- Login node: use /scratch, ***not*** /tmp or /var/tmp
- Quota increases for /data – request at https://hpc.nih.gov/nih/storage_request.html
- Shared (group) directory – request at https://hpc.nih.gov/nih/shared_data_request.html

Biowulf FileSystems

Use /data/username, ***not*** /gs1/users/username



Remember: reference paths as /data/…, ***not*** /gs1…, /gs2/… or /spin1/…

4.8 Petabytes (GPFS Parallel Filesystems)

Hands-on (disk storage)

Simple commands

```
ls  -l /home/$USER                          (check contents of your /home area)
ls  -l /data/$USER                          (check contents of your /data area)
checkquota                                  (check your disk space usage/allocation)
mkdir /scratch/$USER                        (create a directory in /scratch)
cp /home/$USER/somefile /scratch/$USER/
                                            (copy a file to your /scratch directory)
```

Copy the scripts/data for this class to your own area

```
cp -r /data/classes/biowulf/   /data/$USER/biowulf-class
    (you should now have a directory /data/$USER/biowulf-class)
```

Snapshots

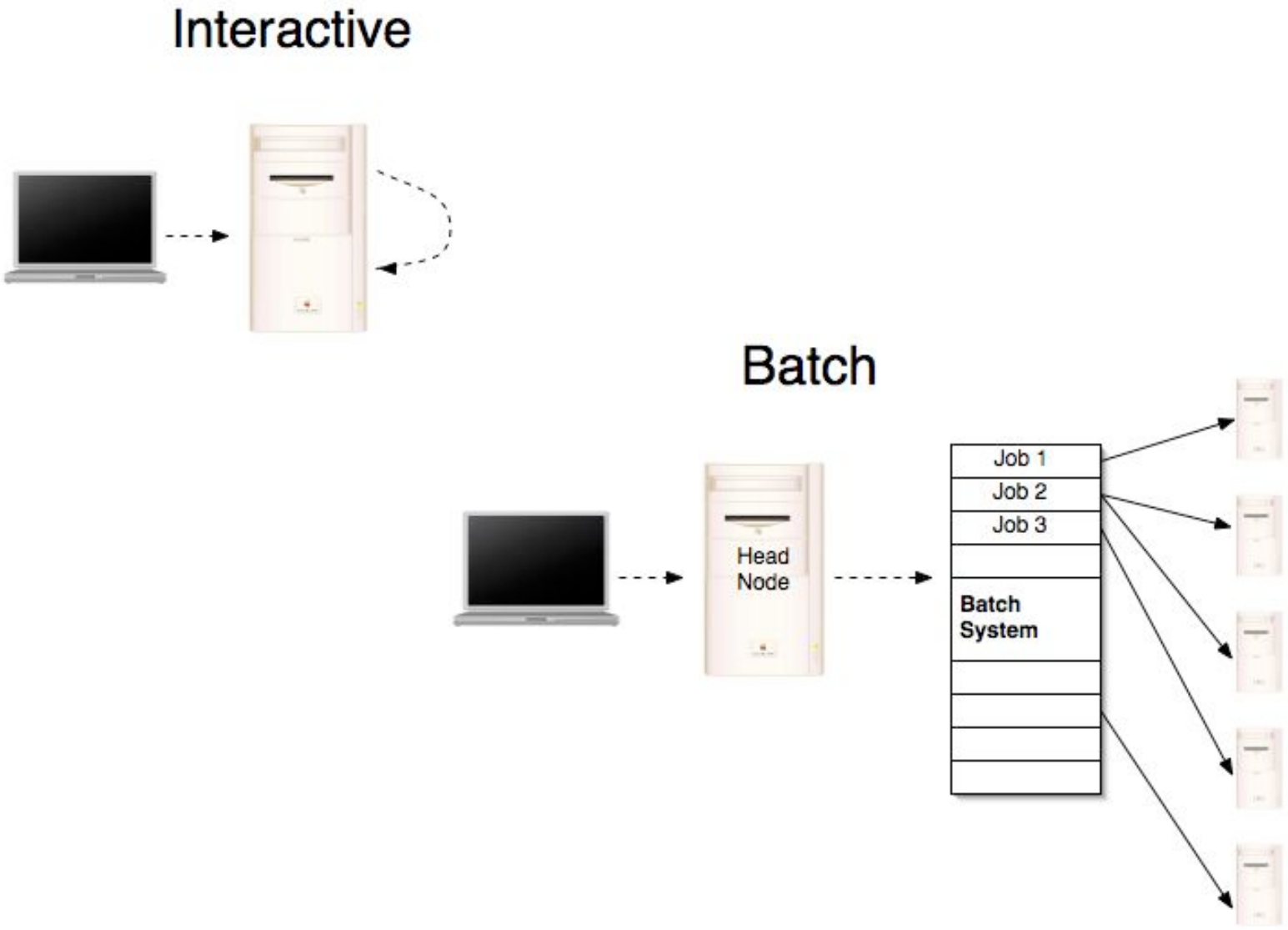
/home	6 daily, 6 nightly, 8 weekly
/data	2 daily, 1 weekly
/scratch	no snapshots

Hands-on (snapshots)

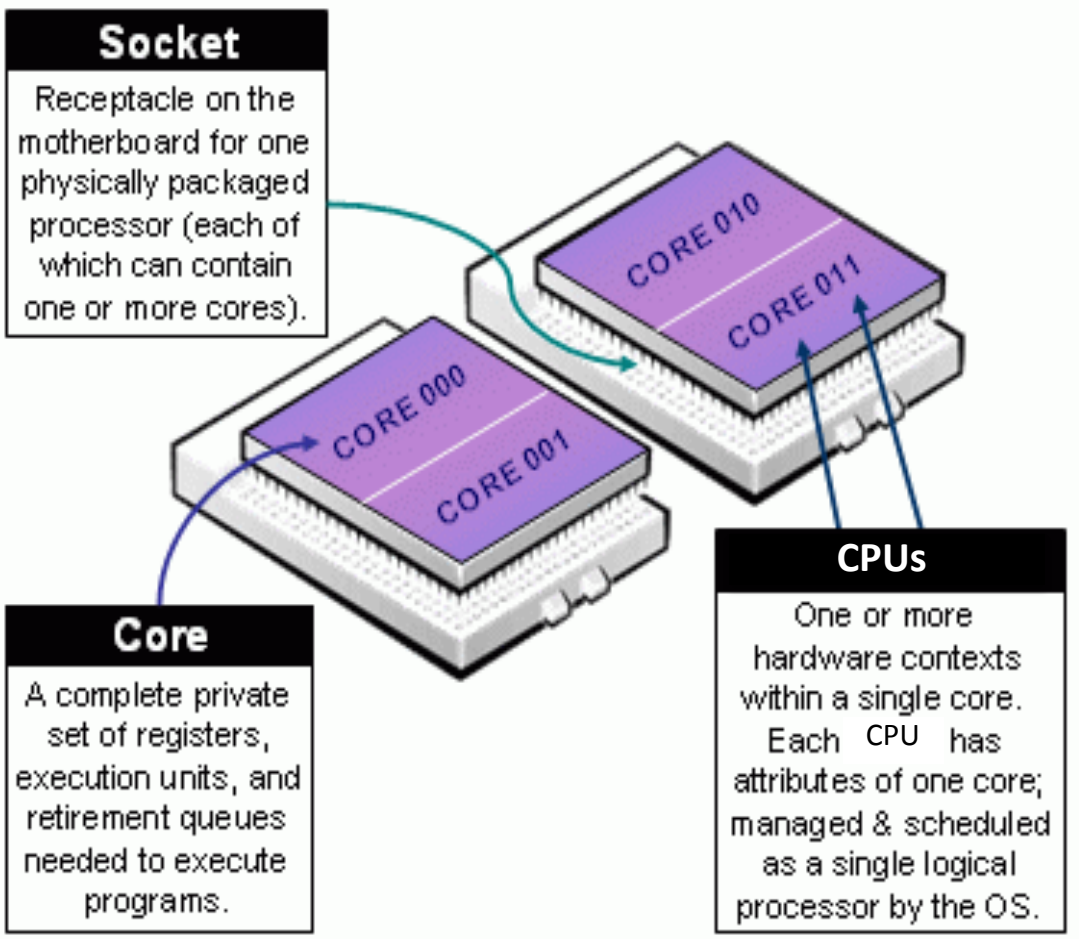
```
cd                                  (cd to your /home area)
ls –a
        (can you see the .snapshot directory?)
cd .snapshot                        (cd to the snapshot in that area)
ls                                  (see what snapshots are available)
cd Nightly.[date]                   (cd to a specific snapshot)
cp somefile /home/$USER             (copy a file from that snapshot to /home)

cp –p somefile /home/$USER          (to preserve timestamps and permissions)
```

The Batch System: SLURM

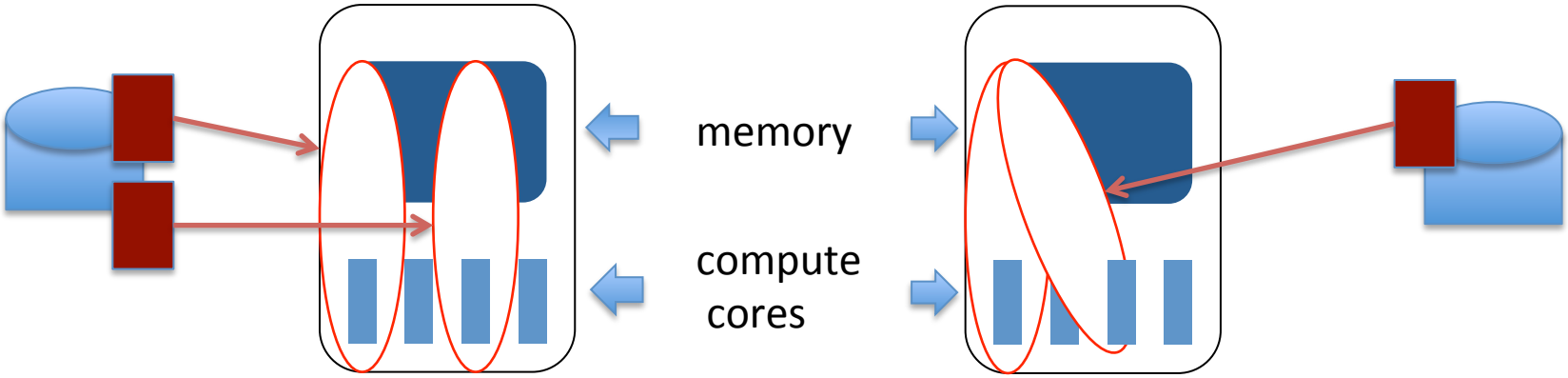
Concepts 1: Interactive vs. Batch



Concepts 2: SLURM Hardware Terminology



Concepts 3: Processes & Threads



2 programs (files) execute as 2 processes *(total number of threads = 2)*

1 program (file) executes as 1 process running 2 threads each sharing the process's memory *(total number of threads = 2)*

"A process has a self-contained execution environment; each process has its own memory space. Processes are often seen as synonymous with programs or applications."

"Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files."

SLURM calls processes "tasks"

Concepts 4: Binding

SLURM uses the Linux kernel to bind process threads to cores

Example:

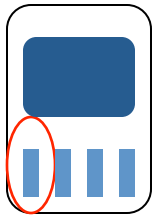
- 16-core node (32 CPUs)
- Submit jobs that allocate 1 core (2 CPUs)

Job 1: Application runs 1 thread, Result: 1 thread runs on 1 core (50% utilization of the core)

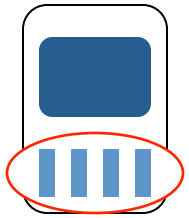
Job 2: Application runs 16 threads, Result: 16 threads run on 1 core! (800% overloading of the core)

Kinds of Jobs on a Cluster

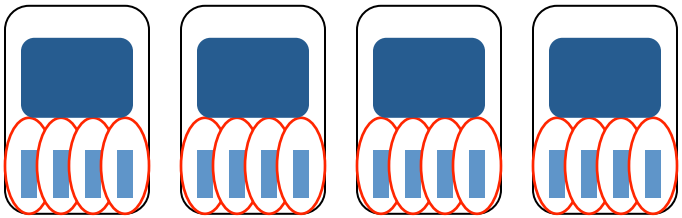
- Single-threaded apps



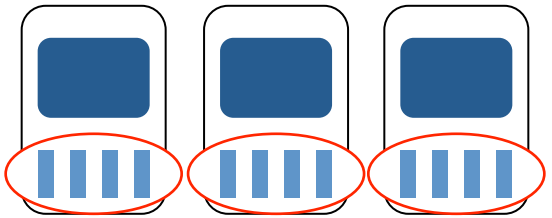
- Shared-memory parallel apps (multi-threaded)



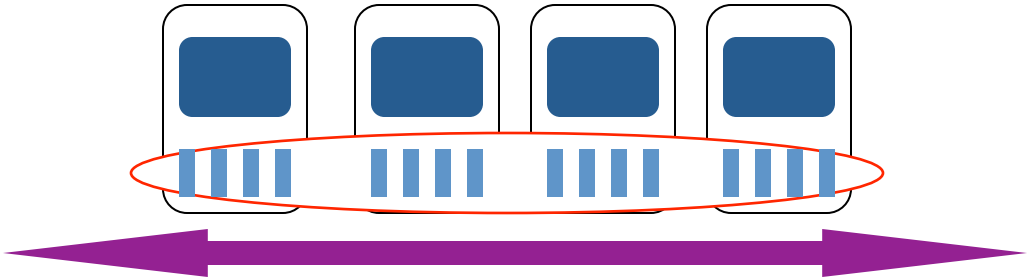
- Swarm of single-threaded processes



- Swarm of multi-threaded processes



- Distributed-memory parallel apps



Jobs on the Biowulf Cluster

- Most (almost all) jobs should be run in batch
- Two ways to submit batch jobs
  - sbatch
  - swarm
- Default compute allocation = 1 physical core
  = 2 CPUs in Slurm notation
- Default Memory Per CPU = 2 GB
  Therefore, default memory allocation = 4 GB

Submitting a simple serial job

```
#!/bin/bash
#
# this file is myjob.sh
#
#SBATCH --job-name MyJob
#SBATCH --mail-type BEGIN,END
#
myprog -n 100 < infile
```

```
% sbatch myjob.sh
```

NOTES:

(1) sbatch exports your shell environment to the batch job

(2) Command line options can also be used as directives within script (#SBATCH)

Output/error files

Slurm combines stdout and stderr into a single file called slurm-<job number>.out

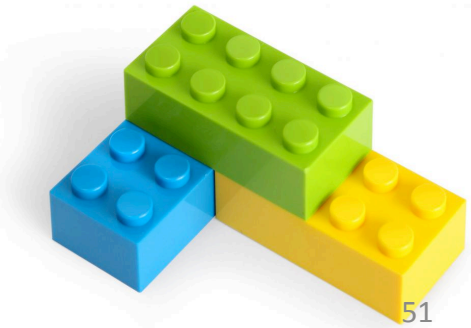
Default location: submission directory

Can be reassigned with

```
--error /path/to/dir/filename
--output /path/to/dir/filename
```

Using modules in batch scripts

```
#!/bin/bash

# load the latest version of bsoft
module load bsoft

# load a particular version of bsoft
# module load bsoft/1.8

cd /data/user/myjob
bimg -verbose 7 -truncate 0,100 input.img output.img
```



wiseGEEK

Other slurm commands

- squeue
- sjobs
- scancel
- scontrol

Allocating compute resources

- --mem
- --mem-per-cpu
- --ntasks
- --cpus-per-task
- --ntasks-per-core
- --gres

Special case only…

- --exclusive
- --constraint

Single-threaded app

Required		Command	Allocated	
CPU	Memory		CPU	Memory
1	< 4 GB	`sbatch jobscript`	2	4 GB
1	***M*** GB (***M*** > 4)	`sbatch --mem=`***M***`g jobscript`	2	***M*** GB

Hands-on
Submitting a single batch job

```
cd /data/$USER/biowulf-class/plink
more plink.bat
sbatch plink.bat
squeue -u username
sjobs
```

- Which node is your job running on?
- Find the output file: Examine it for errors.
- How many cores/CPUs were allocated?
- How many cores/CPUs were used?

Multi-threaded app

Required		Command	Allocated	
CPU	Memory		CPU	Memory
C threads on C CPUs	***M*** GB	`sbatch --cpus-per-task=`***C*** `--mem=`***M***`g jobscript`	***C*** (rounded up to nearest 2)	***M*** GB

Default memory allocation for ***C*** cpus would be (***C****2) GB

Multi-threaded app

Use $SLURM_CPUS_PER_TASK inside the script

```
#!/bin/bash

cd /data/mydir
module load hmmer
hmmsearch --cpu $SLURM_CPUS_PER_TASK globins4.hmm /fdb/fastadb/nr.aa.fas
```

```
#!/bin/bash
cd /data/mydir
module load mapsplice
mapsplice.py -1 Sp_ds.10k.left.fq -2 Sp_ds.10k.right.fq -p $SLURM_CPUS_PER_TASK
```

Hands-on (multi-threaded Tophat)

```
cd /data/$USER/biowulf-class/bowtie
sbatch --cpus-per-task=8 –mem=5g tophat.bat
```

Use ‘sjobs’ to see if the job is running.

Look at the tophat.bat script to see how the number of CPUs (threads) is specified to tophat. Why is this important?

Auto-threading apps

```
sbatch –-exclusive --cpus-per-task=32 jobscript
```

will give you a node with 32 CPUs.

Interactive Batch Jobs

Reasons to use interactive nodes

- Testing/debugging cpu-intensive code
- Pre/post-processing of data
- Graphical application
- GUI interface to application

Reasons *not* to use interactive nodes

- Easier than setting up a batch job

Interactive jobs

```
[biowulf2 ~]$ sinteractive
salloc.exe: Granted job allocation 22374
[cn0004 ~]$ exit
exit
salloc.exe: Relinquishing job allocation 22374
```

```
[biowulf2 somedir]$ sinteractive --cpus-per-task=4 --mem=8g
salloc.exe: Granted job allocation 22375
[cn0004 somedir]$
```

Note: your environment will be exported to the job by default.

Interactive Batch Jobs

- The maximum allowable number of cores allocated to interactive jobs is 32 (64 CPUs).
- By default, the job will be terminated after 8 hours. You may extend this time to a maximum of 36 hours by using the ”--time" attribute to the `sinteractive` command.

Hands-on (interactive batch jobs)

```
sinteractive
```

Once you are logged into a node:

```
cd /data/$USER/biowulf-class/freebayes
module load freebayes
freebayes --fasta-reference PyYM_genome.fasta N67.bam
exit
```

Requeue Job?

```
sbatch –requeue          ← default
```

or

```
sbatch --no-requeue
```

Deleting Batch Jobs

```
scancel <jobid> <jobid> …
```

Options

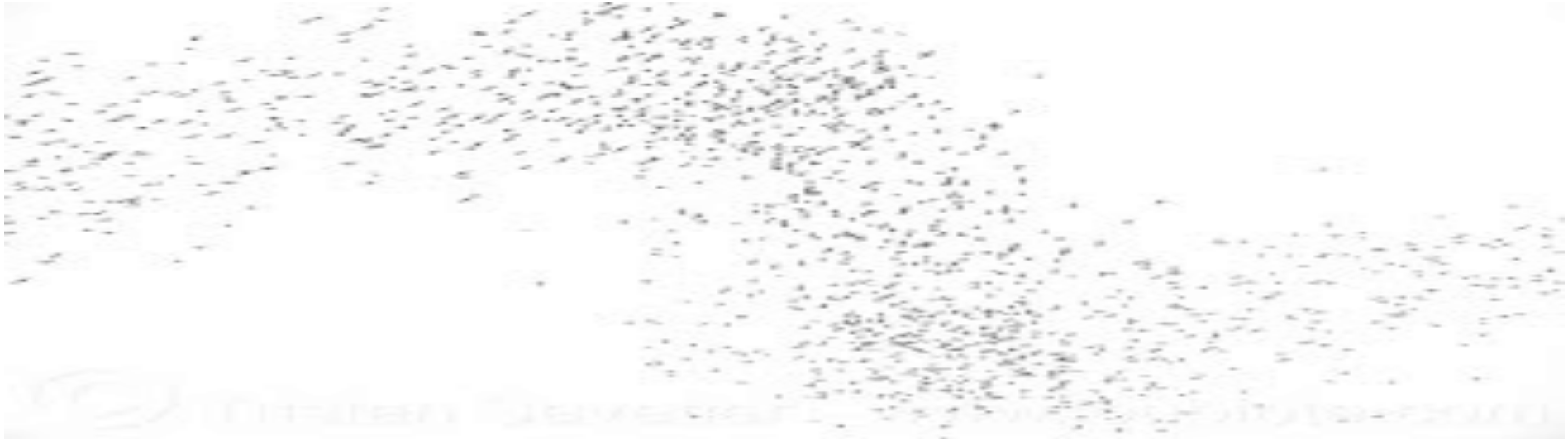
```
--user
--name
--state
--nodelist
```

To delete this swarm…

```
3479375_0      norm    swarm     joeu  R 2-04:15:47      1 cn0175
3479375_5      norm    swarm     joeu  R 2-04:15:47      1 cn0297
3479375_6      norm    swarm     joeu  R 2-04:15:47      1 cn0297
3479375_9      norm    swarm     joeu  R 2-04:15:47      1 cn0304
3479375_10      norm    swarm     joeu  R 2-04:15:47      1 cn0304
3479375_11      norm    swarm     joeu  R 2-04:15:47      1 cn0257
3479375_12      norm    swarm     joeu  R 2-04:15:47      1 cn0257
3479375_13      norm    swarm     joeu  R 2-04:15:47      1 cn0257
```

```
$ scancel 3479375
```

Swarm



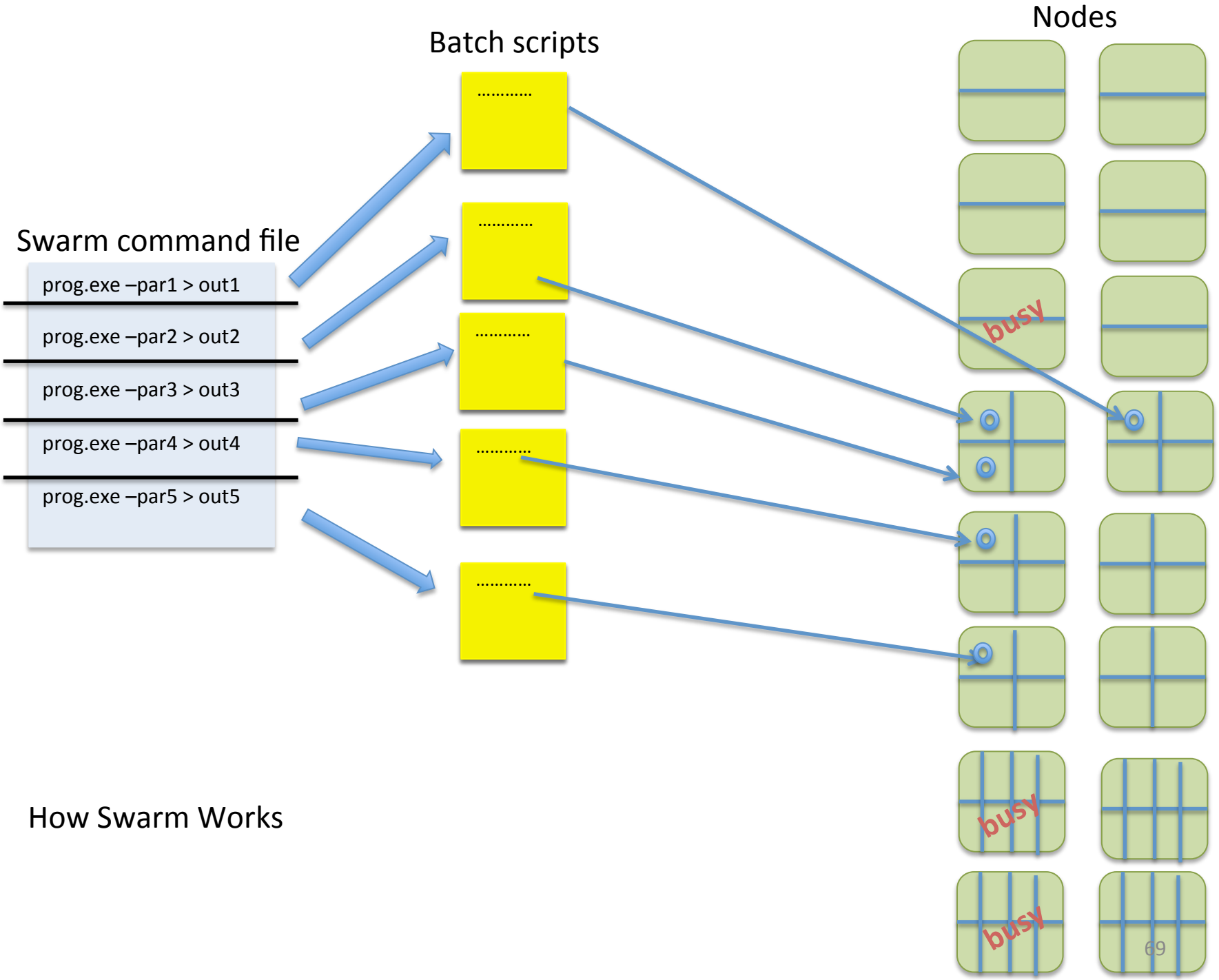
Submitting swarms of jobs with *swarm*

```
#
# this file is cmdfile
#
myprog -param a < infile-a > outfile-a
myprog -param b < infile-b > outfile-b
myprog -param c < infile-c > outfile-c
myprog -param d < infile-d > outfile-d
myprog -param e < infile-e > outfile-e
myprog -param f < infile-f > outfile-f
myprog -param g < infile-g > outfile-g
```

```
% swarm –f cmdfile
```

What does swarm do, exactly?

- Creates a subjob for each line in your swarm command file
- Submits all subjobs as a single jobarray to slurm.
- Single jobid regardless of number of commands.



How Swarm Works

Swarm of single-threaded processes

Required		Command	Allocated	
CPU	Memory		CPU	Memory
1 per process	< 1.5 GB per process	`swarm -f swarmfile`	2 per process	1.5 GB per process
1 per process	***G*** GB (***G*** > 1.5)	`swarm -g G -f swarmfile`	2 per process	***G*** GB per process
1 per process	***G*** GB (***G*** > 1.5)	`swarm -g G -p P -f swarmfile`	1 per process	***G*** GB per process

Swarm of multi-threaded processes

Required		Command	Allocated	
CPU	Memory		CPU	Memory
T threads on ***T*** CPUs per process	***G*** GB per process	`swarm -t T -g G -f swarmfile`	***T*** per process	***G*** GB per process

Default memory allocation: ***T****1.5 GB per process

Swarm of multi-threaded processes

Use $SLURM_CPUS_PER_TASK inside batch script

```
mapsplice.py  -1 seq1a.fq  -2 seq1b.fq  –p $SLURM_CPUS_PER_TASK
mapsplice.py  -1 seq2a.fq -2 seq2b.fq   –p $SLURM_CPUS_PER_TASK
mapsplice.py  -1 seq3a.fq -2 seq3b.fq   –p $SLURM_CPUS_PER_TASK
etc
```

Swarm of auto-threading processes

swarm -t auto

-> implies --exclusive

swarms

- Single job array created
- The number of subjobs = number of commands
- Entire swarm will be placed on one partition only
- Use semi-colons (;) to separate multiple commands on one line
- *swarm* expects bash syntax
- *swarmdel* command (see "*man swarmdel*")
- Create complex/long swarm command files with scripts

Sample swarm-created batch scripts

```
#!/bin/bash

# When started, migrate the directory name to something more pleasing
d=/spin1/swarm/user/${SLURM_ARRAY_JOB_ID}
while [[ -d /spin1/swarm/user/I762VwVu ]]; do
  mv /spin1/swarm/user/I762VwVu ${d} 2>/dev/null
  mv /spin1/swarm/user/I762VwVu.batch ${d}.batch 2>/dev/null
done

z=$(printf '%02d' ${SLURM_ARRAY_TASK_ID})
/bin/bash ${d}/cmd.${z}
exitcode=$?
rm ${d}/cmd.${z}
exit $exitcode
```

```
source /usr/local/lmod/lmod/lmod/init/bash
module load R
 R –vanilla –args59250 > output59250
```

Just FYI. You won't normally need to look at these

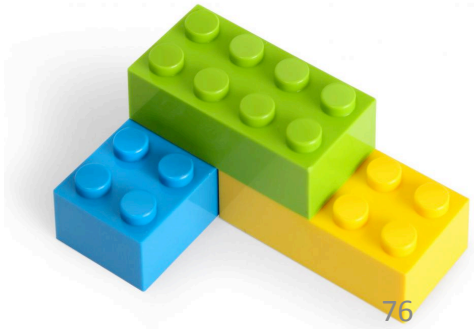
Using modules in swarm scripts

Method 1:

```
bimg -verbose 7 -truncate 0,100 input1.img output1.img
bimg -verbose 7 -truncate 0,100 input2.img output2.img
bimg -verbose 7 -truncate 0,100 input3.img output3.img
```

Submit with:

```
swarm –f swarmfile --module bsoft/1.8.2
```



wiseGEEK

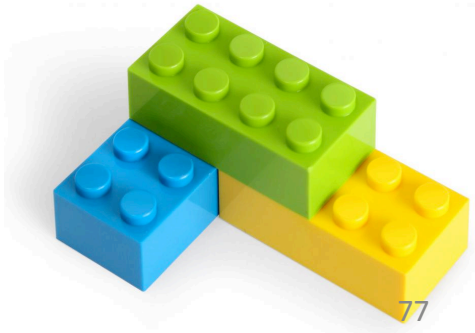
Using modules in swarm scripts

Method 2:

```
module load bsoft/1.8.2; bimg -verbose 7 -truncate 0,100 input1.img output1.img
module load bsoft/1.8.2; bimg -verbose 7 -truncate 0,100 input2.img output2.img
module load bsoft/1.8.2; bimg -verbose 7 -truncate 0,100 input3.img output3.img
```

Submit with:

swarm –f swarmfile



wiseGEEK

Hands-on (swarm)

```
cd /data/$USER/biowulf-class/swarm
```

- Use 'more' to examine the file blat.swarm
- '`wc -l blat.swarm`' will give you the # of commands

```
swarm -f blat.swarm --module blat
```

- How many jobs were created?
- Use '`sjobs`' to watch your jobs.
- Use '`ls`' and '`more`' to examine the output after the swarm is done.

Swarm options

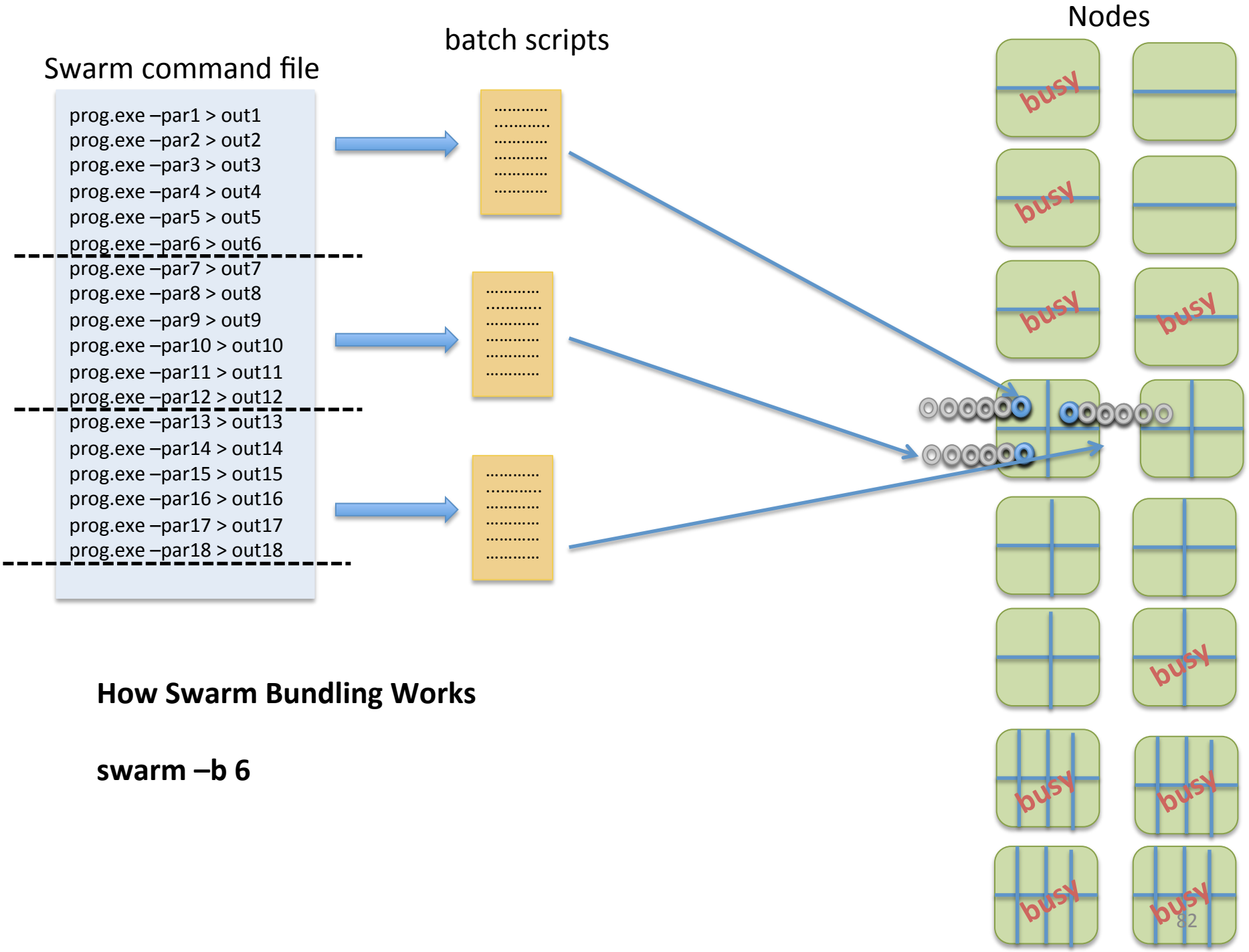
```
$ swarm --help
Usage: swarm [swarm options] [qsub options]

  Usage: swarm [swarm options] [sbatch options]

  -f,--file [file]          name of file with list of command lines to execute,
                            with a single command line per subjob

  -g,--gb-per-process       gb per process (can be fractions of GB, e.g. 3.5)
  [float]

  -t,                       threads per process (can be an integer or the word
  --threads-per-process     auto).  This option is only valid for multi-
  [int]/"auto"              threaded swarms (-p 1).

  -p,                       processes per subjob (default = 1).  This option is
  --processes-per-subjob    only valid for single-threaded swarms (-t 1).
  [int]

  --noht                    don't use hyperthreading, equivalent to slurm option
                            --threads-per-core=1

  -b,--bundle [int]         bundle more than one command line per subjob and run
                            sequentially

  --usecsh                  use tcsh as the shell instead of bash
  -m, --module              provide a list of environment modules to load
                            prior to execution (comma delimited)
```

Swarm options

```
$ swarm --help [contd]
--singleout             concatenate STDOUT and STDERR to single files
                          NOTE: This option is NOT appropriate for swarms of
                          very short running commands, as the process of
                          concatenating output can outlast command execution!

  --logdir              directory to which .o and .e files are to be written
                        (default is current working directory)


sbatch options:

  -J, --job-name [str]  set the name of the job
  --dependency [str]    set up dependency (i.e. run swarm before or after)
  --time [str]          change the walltime for each subjob (default is
                        04:00:00, or 4 hours)
  -L,--licenses [str]   obtain software licenses (e.g. --licenses=matlab)
  --partition [str]     change the partition (default is norm)
  --gres [str]          set generic resources for swarm
  --qos [str]           set quality of service for swarm

Other sbatch options

  --sbatch [string]     add sbatch-specific options to swarm.  These options
                        will be added last, which means that swarm options
                        for allocation of cpus and memory take precedence.
```

Swarm bundles

- For large swarms or swarms with short-lived threads (under 5-10 minutes)
- `swarm –b 20 –f commandfile`
- creates a “bundle” (queue) of 20 processes *per core*
- 8000 lines (8000 processes), -b 20 => 25 jobs (16-core)
- Bundles *required* for command files with >10,000 lines



How Swarm Bundling Works

swarm –b 6

Swarms of multi-threaded programs

- `swarm -t 8 -f commandfile`
- Use $SLURM_CPUS_PER_TASK within script to specify the number of threads
- Default value of 1

Specifying memory requirements to swarm

- `swarm –g 8 –f commandfile`
- Swarm will reserve 8 GB of memory *per process*
- Default value of 3 GB

Swarm with licenses

```
swarm -f swarmfile -license=matlab
```

Caveat: not working correctly yet!

Large Memory Swarms (max 1 TB)

```
swarm -f swarmfile -g 128
```

Hands-on (swarm bundling)

```
cd /data/$USER/biowulf-class/swarm
swarm –b 4 –g 4 –f blat.swarm ––module blat
```

How many subjobs were produced?

More on Biowulf/Slurm Resources

Partitions (Queues)

```
$ batchlim

Max Jobs Per User: 4000
Max Array Size: 1001
```

Partition	MaxCPUsPerUser	DefWalltime	MaxWalltime
norm	1024	04:00:00	10-00:00:00
b1	1024	06:00:00	10-00:00:00
interactive	64	08:00:00	1-12:00:00
quick	1024	02:00:00	02:00:00
largemem	128	04:00:00	10-00:00:00
ibfdr	1024	10-00:00:00	10-00:00:00
gpu	128	10-00:00:00	10-00:00:00
unlimited	128	UNLIMITED	UNLIMITED
ccr	1536	04:00:00	10-00:00:00
lcp	9999	10-00:00:00	10-00:00:00
niddk	1024	04:00:00	10-00:00:00
nimh	1024	04:00:00	10-00:00:00

Show available compute resources

```
$ freen

                                          ........Per-Node Resources........
Partition     FreeNds       FreeCPUs       Cores CPUs    Mem   Disk    Features
-------------------------------------------------------------------------------
norm*        57/310       3152/9920          16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g
unlimited    6/14         350/448            16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g
niddk        49/86        1568/2752          16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,niddk
ibfdr        0/192        58/6144            16    32     62g   800g   cpu32,core16,g64,ssd800,x2650,ibfdr
gpu          0/20         6/640              16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,gpuk20x
gpu          0/4          0/128              16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,gpuk20x,acemd
gpu          13/16        156/192            12    12     47g   600g   cpu12,core12,g48,sata600,x5650,gpum2050,1g
largemem     0/4          152/256            32    64   1009g   800g   cpu64,core32,g1024,ssd800,x4620,10g
nimh         63/64        2016/2048          16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,nimh
ccr          25/96        840/3072           16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,ccr
ccr          49/64        1568/2048          16    32    125g   400g   cpu32,core16,g128,sata400,x2600,1g,ccr
ccr          29/64        1488/2048          16    32     62g   400g   cpu32,core16,g64,sata400,x2600,1g,ccr
lcp          31/103       992/3296           16    32     31g   400g   cpu32,core16,g32,sata400,x2600,ibqdr,lcp
lcp          61/103       1952/3296          16    32     62g   400g   cpu32,core16,g64,sata400,x2600,ibqdr,lcp
quick        49/86        1568/2752          16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,niddk
quick        25/96        840/3072           16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,ccr
quick        63/64        2016/2048          16    32    125g   800g   cpu32,core16,g128,ssd800,x2650,10g,nimh
quick        49/64        1568/2048          16    32    125g   400g   cpu32,core16,g128,sata400,x2600,1g,ccr
quick        29/64        1488/2048          16    32     62g   400g   cpu32,core16,g64,sata400,x2600,1g,ccr
b1           214/320      4000/5120           8    16     23g   200g   cpu16,core8,g24,sata200,e2666,1g
b1           6/16         100/256             8    16     70g   200g   cpu16,core8,g72,sata200,e2666,1g
b1           11/16        176/256             8    16     70g   100g   cpu16,core8,g72,sata100,x5550,1g
b1           20/24        640/768            16    32    252g   400g   cpu32,core16,g256,sata400,e2670,1g
ibddr        121/250      968/2000            8     8      7g   100g   cpu8,core8,g8,sata100,e5462,ibddr
```

Submitting Jobs to Partitions

```
sbatch --partition=quick jobscript
```

```
swarm --partition=quick swarmfile
```

CPU limits in partitions are cumulative, example for NIDDK user:
niddk (1024 cpus) + norm (1024 cpus) + quick (1024 cpus) = 3072 cpus

Coming soon…

```
$ sbatch --partition niddk,norm ... [for buy-in users]
```

“Unlimited” Partition

```
sbatch --partition=unlimited …
```

- No walltime limit
- Only 14 nodes (448 CPUs) available

Walltimes

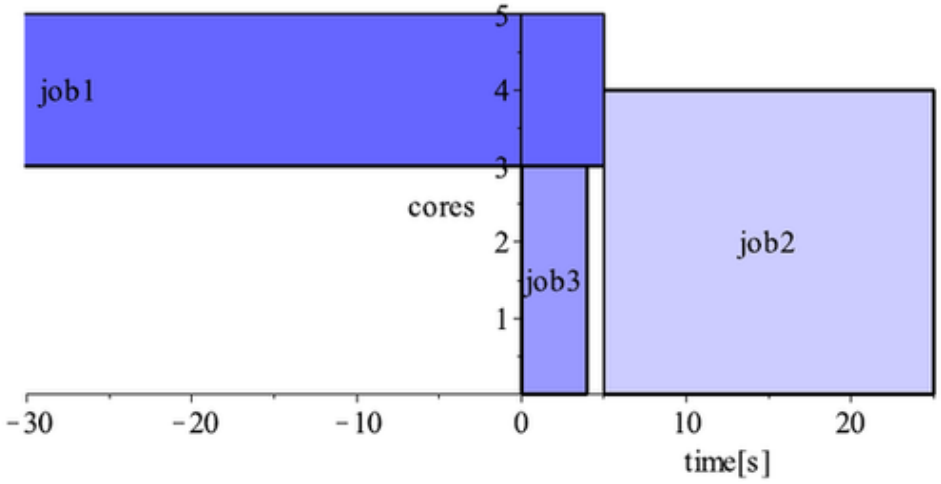
- Required for all jobs
- Default 4 hr walltime in norm (default) partition
- Specify walltime with

```
--time=d-hh:mm:ss
```

e.g. --time=12:00:00 (12 hrs)
--time=36:00:00 (36 hrs)
--time=2-12:00:00 (2 days, 12 hrs)

Why accurate walltimes are important

- Backfill scheduling



Speaking of scheduling…

- Priority = FairShare+Age+JobSize+Partition+QOS
- Fairshare half life of 7 days
- `sprio` command reports priorities of pending (queued) jobs
- Backfill job scheduling for parallel jobs

Local disk on node

- Local disk is now /lscratch (/scratch is central shared space)
- ‘freen’ will report disk size as a Feature (eg, ssd800 = 800GB SSD disk)
- Jobs which need to use /lscratch must explicitly allocate space:

```
sbatch --gres=lscratch:500 …
```

GB

- Read/write to /lscratch/$SLURM_JOBID (/lscratch directory not writable)
- /lscratch/$SLURM_JOBID deleted at end of job

Hands-on (lscratch)

```
sinteractive --gres=lscratch:2
                    (Requesting 2 GB of local scratch)
```

Once you are logged into a node:

```
cd /lscratch/$SLURM_JOBID; ls -l
cp -r /data/$USER/biowulf-class/freebayes/ .
module load freebayes
cd freebayes
freebayes --fasta-reference PyYM_genome.fasta N67.bam
exit
```

Question: once you exit the job, can you access the files on /lscratch on the node?

What happens if you try to write to /lscratch, instead of /lscratch/$SLURM_JOBID?

Allocating GPUs

```
sbatch —-partition=gpu --gres=gpu:k20x:1 jobscript

sbatch —-partition=gpu --gres=gpu:k20x:2 jobscript
```

Resource Name

Resource Type

Count
Required even if 1
(Max of 2)

```
--gres=gpu:k20x:1
--gres=gpu:m2050:1
```

Large Memory Jobs (>128 GB)

```
sbatch --mem=768g …
```

- Four large memory nodes
- Use large memory nodes for their memory, not for their (64) cores

Licensed Software

```
[biowulf2 ~]$ licenses

 License                      Total   Free
--------------------------   -----   ----
IDL                             20     14
Matlab                          24     18
Bioinformatics Toolbox           3      2
Compiler                         1      1
Curve Fitting Toolbox            3      2
…etc…
```

Request a license with

```
sbatch --license=matlab,matlab-stat
```

```
swarm –f swarmfile --license=matlab
```

Hands-on (swarm vs sbatch)

```
cd /data/$USER/biowulf-class/R
```

Compare the files R.bat and swarm.cmd

How would you submit each one?

Monitoring Batch Jobs

- Things to monitor
  - Core (CPU) utilization
  - Memory

- Tools for monitoring
  - `sjobs, squeue`
  - `jobload`
  - `jobhist`

- *Users* (you) are responsible for monitoring their jobs

Monitoring – show all jobs

```
[susanc@biowulf2]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             22340     ibfdr stmv-204   susanc PD       0:00    128 (Dependency)
             22353     ibfdr meme_sho   susanc PD       0:00    128 (Dependency)
             22339     ibfdr stmv-102   susanc PD       0:00     64 (Dependency)
             22352     ibfdr meme_sho   susanc PD       0:00     64 (Dependency)
```

```
[susanc@biowulf2]$ sjobs
                                                        ................Requested............................
User       JobId    JobName    Part  St      Runtime   Nodes  CPUs     Mem        Dependency     Features    Nodelist
susanc     22344    meme_short ibfdr  R        22:30   1      32   1.0GB/cpu                     (null)      cn0414
susanc     22335    stmv-64    ibfdr  R         0:54   4     128   1.0GB/cpu                     (null)      cn[0413,0415-0417]
susanc     22336    stmv-128   ibfdr  PD        0:00   8     128   1.0GB/cpu   afterany:22335    (null)      (Dependency)
susanc     22337    stmv-256   ibfdr  PD        0:00   16    256   1.0GB/cpu   afterany:22336    (null)      (Dependency)
susanc     22338    stmv-512   ibfdr  PD        0:00   32    512   1.0GB/cpu   afterany:22337    (null)      (Dependency)
```

Monitoring – jobload

```
$ jobload -u muser
         JOBID          TIME    NODES    CPUS   THREADS    LOAD              MEMORY
                                         Alloc   Running                Used/Alloc
     170581_276   3-14:22:51   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_277   3-14:20:51   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_282   3-14:07:52   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_283   3-14:07:52   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_284   3-14:06:52   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_285   3-14:06:52   cn0343       2         2    100%    294.6 MB/1.5 GB
     170581_286   3-14:01:52   cn0343       2         2    100%    294.6 MB/1.5 GB
...
USER SUMMARY
      Jobs: 256
     Nodes: 256
      CPUs: 512
 Load Avg: 100%
```

```
$ jobload -u suser
         JOBID          TIME    NODES    CPUS   THREADS    LOAD              MEMORY
                                         Alloc   Running                Used/Alloc
        211880       2:08:46   cn0499      32        16     50%     2.6 GB/31.2 GB
                     2:08:46   cn0500      32        16     50%     2.4 GB/31.2 GB
                     2:08:46   cn0501      32        16     50%     2.4 GB/31.2 GB
...
JOB SUMMARY: {Nodes: 48, CPUs: 1536, Load Avg: 50.0}

USER SUMMARY
      Jobs: 1
     Nodes: 48
      CPUs: 1536
 Load Avg: 50%
```

jobhist

```
[biowulf2 ~]$ jobhist 22343

JobId              : 22343
User               : susanc
Submitted          : 20150522 08:59:54
Submission Dir     : /spin1/users/susanc/meme
Submission Command : sbatch --depend=afterany:22342 --partition=ibfdr --ntasks=2 --ntasks-per-core=1
                     meme_short.slurm

Partition     State       AllocNodes     AllocCPUs      Walltime          MemReq       MemUsed    Nodelist
     ibfdr   COMPLETED           1            32          01:43:58       1.0GB/cpu        0.7GB    cn0414
```

Note: RUNNING jobs will always show MemUsed = 0.0GB

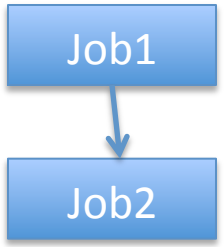
Hands-on (monitoring)

```
cd /data/$USER/biowulf-class/bowtie
sbatch --cpus-per-task=16 bowtie.bat
```

- Use ‘jobload –u username’ to check the status of your running job. How much memory is it using?

- After the job finishes, use ‘jobhist jobnumber’ and find out how much memory it used.

Job Dependencies

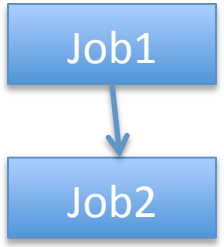



```
biowulf% sbatch Job1.bat
1111.biobos

biowulf% sbatch --depend=afterany:1111 Job2.bat
1112.biobos

biowulf% sjobs

[~]$ sjobs
User     JobId    JobName   Part  St  Reason      Runtime  Walltime  Dependency       Nodelist
================================================================================================
teacher  1111     Job1.bat  norm  R   ---            0:17   4:00:00                    cn0035
teacher  1112     Job2.bat  norm  PD  Dependency     0:00   4:00:00 afterany:1111
================================================================================================
cpus running = 2
cpus queued = 1
jobs running = 1
jobs queued = 1
```

Job Dependencies

```
--depend=afterany     (after all of the jobs have exited, with or without errors)
         afterok      (after all of the jobs have exited with no errors)
         afternotok   (after all of the jobs have exited with errors)
         after       (after all of the jobs have started)
```

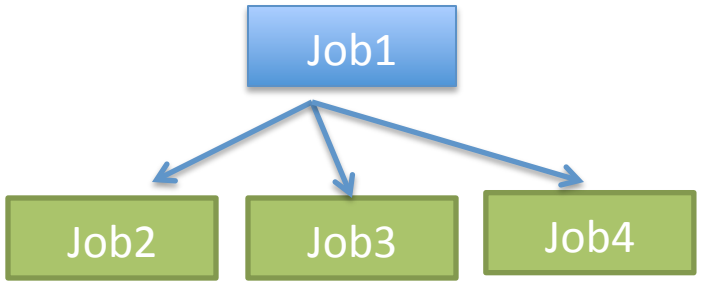
IMPORTANT: Exit status = exit status of shell = exit status of last command



Job Dependencies

Making several jobs dependent on a single job.

```
biowulf% sbatch  Job1.bat
1111

biowulf% sbatch --depend=afterany:1111   Job2.bat
1112

biowulf% sbatch --depend=afterany:1111   Job3.bat
1113
```

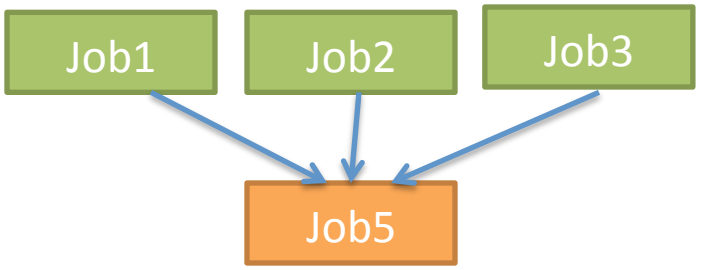


Job Dependencies

Making one job dependent on several other jobs.

```
biowulf% sbatch --depend=afterany:1111:1112:1113  Job3.bat
1114
```

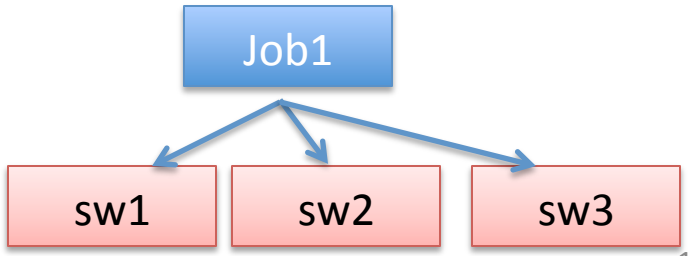
```
Biowulf% sjobs
User      JobId    JobName    Part  St  Reason      Runtime  Walltime  Dependency
Nodelist
================================================================================
teacher  3996129  Job1.bat   norm  R   ---            0:17   4:00:00                  cn0035
teacher  3996143  Job2.bat   norm  PD  Dependency     0:00   4:00:00 afterany:1111,1112,1113
================================================================================
cpus running = 2
cpus queued = 1
jobs running = 1
jobs queued = 1
```



Job Dependencies

Making a swarm dependent on one job.

```
biowulf% swarm -g 10 -f swarm.cmd --depend=afterany:5322129

biowulf% sjobs

User     JobId           JobName   Part  St  Reason      Runtime Dependency       Nodelist
=============================================================================================
teacher  3996455         Job1.bat  norm  R   ---            1:                    cn0046
teacher  3996481_[0-11]  swarm     norm  PD  Dependency    0:00 afterany:5322129
=============================================================================================
cpus running = 2
cpus queued = 12
jobs running = 1
jobs queued = 12
```



Hands-on (job dependencies)

```
cd /data/$USER/biowulf-class/dependencies
sbatch Job1.bat
```

(select or note down the job number)

```
sbatch --depend=afterany:Job1num  Job2.bat
```

(select or note down the job number)

```
sjobs
```

(what is the state of the second job?)

```
sbatch --depend=afterany:Job1num:Job2num Job3.bat
sjobs
```

(does it show the dependency?)

```
swarm -g 2 --depend=afterany:Job3num -f swarm.cmd
```

Matlab on Biowulf

- Can be multi-threaded.

http://biowulf.nih.gov/apps/matlab.html

- License-limited

http://helixweb.nih.gov/nih/license_status

Parallel jobs

- The original driving force driving compute clusters
- N-communicating processes running on N-cores
- Shared-memory (single-node) vs. distributed-memory (multi-node)
- Message-passing libraries (MPI)
- Communication and scalability
- High-performance interconnects

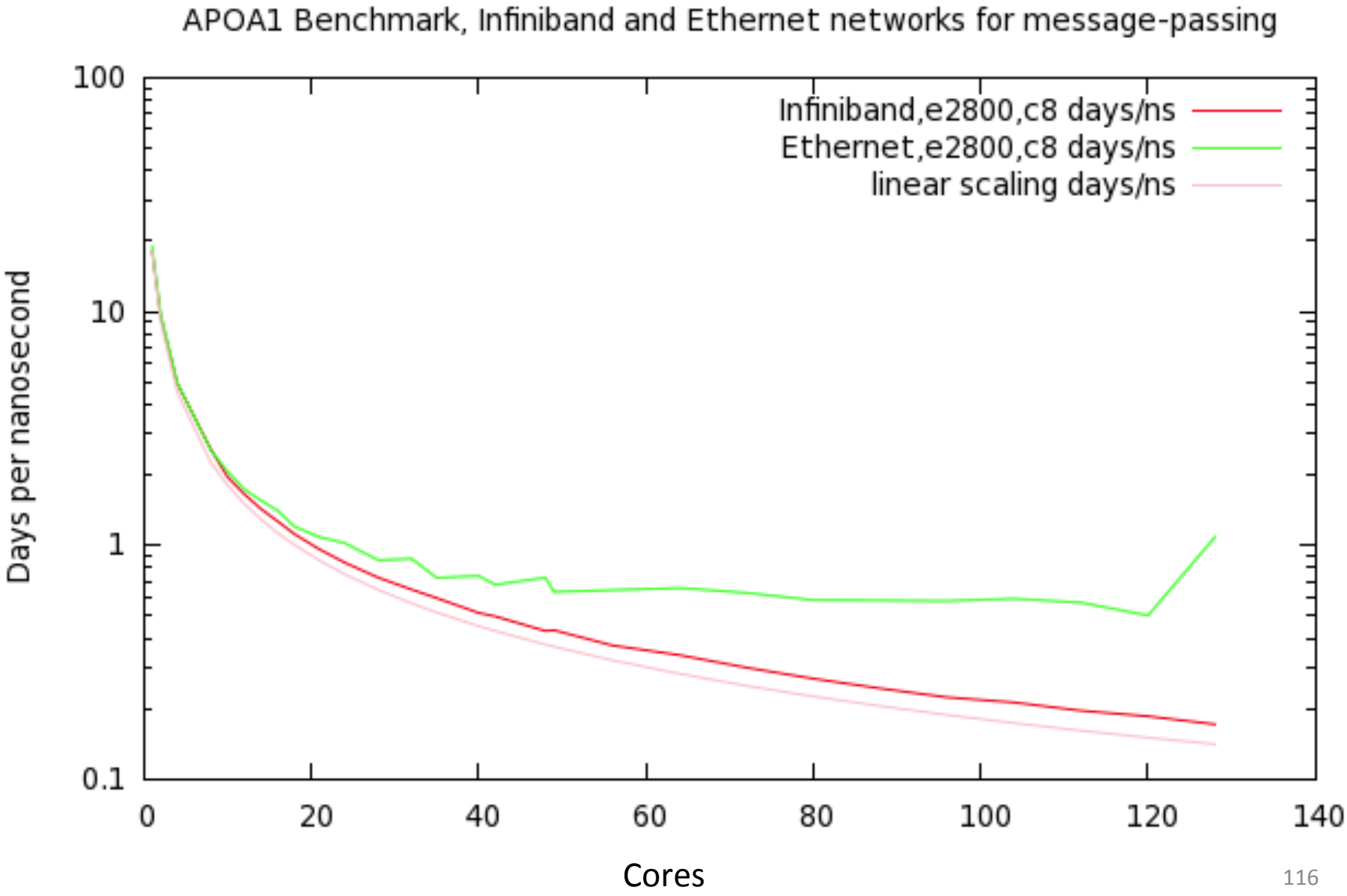
High-performance Infiniband Networks

- Parallel (MPI) applications only
- Low latency more important than bandwidth
- Bypass TCP/IP stack
- Requires compilation against special libraries
- Benchmarking is essential

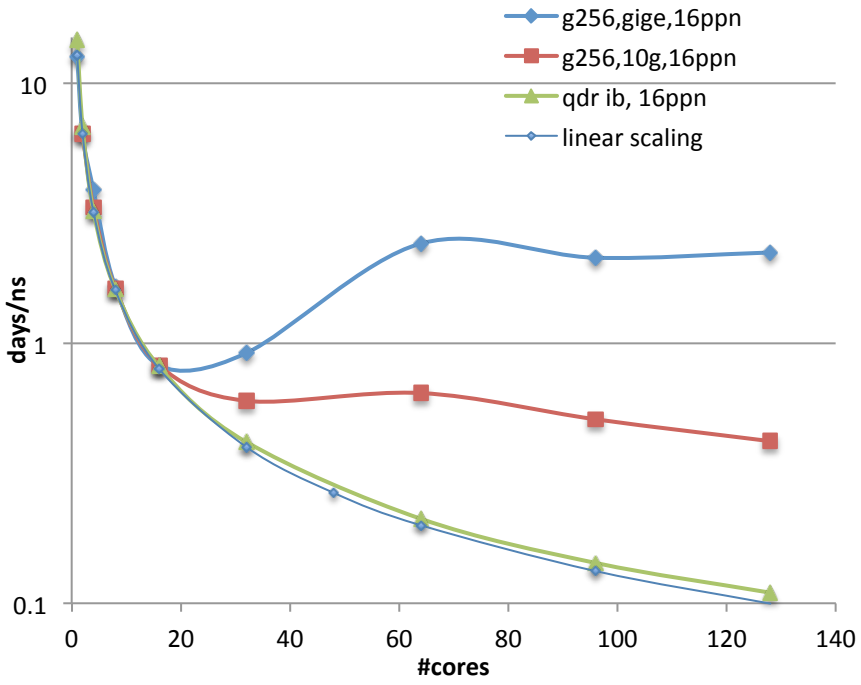
Biowulf Cluster Networks

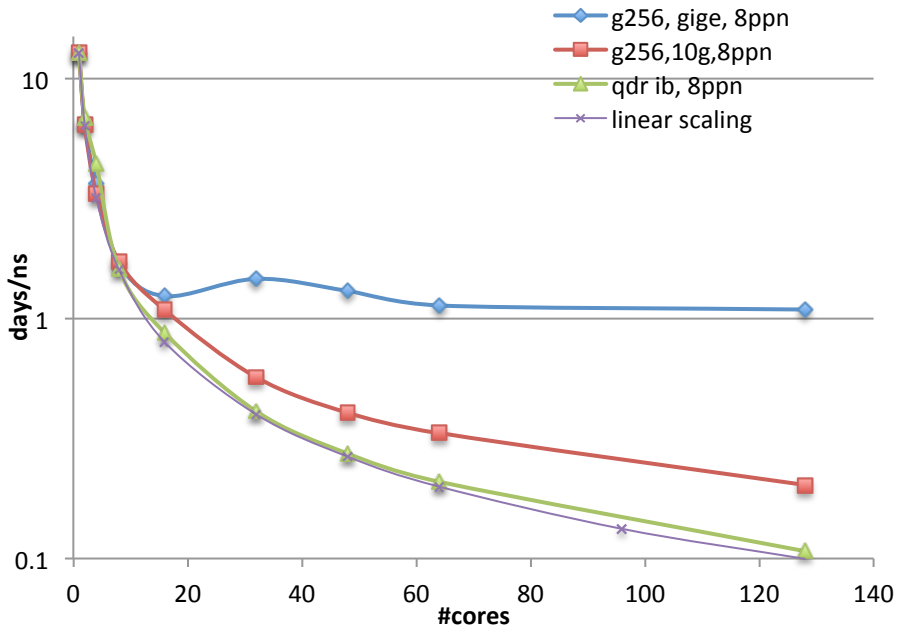
	Ethernet	**Infiniband (FDR)**	**Infiniband (QDR)**	**Infiniband (DDR)**
Bandwidth (full duplex)	10 Gb/s, 1 Gb/s	56 Gb/s	32 Gb/s	16 Gb/s
Latency	“high”	“low”	“low”	“low”
TCP/IP?	Yes	No	No	No
# nodes	All	216	206	250
# cores	All	3456	3296	2000

NAMD Benchmarks



NAMD Benchmarks: 10G vs QDR
Effects of processes-per-node





Parallel (MPI) process

Many parallel apps do better with --ntasks-per-core=1 (ignore hyperthreading)

Required		Command	Allocated	
CPU	Memory		CPU	Memory
C	< 4 GB per MPI process	`sbatch --partition=ibfdr --ntasks=`***C*** `[--constraint=nodetype ] --exclusive --ntasks-per-core=1 jobscript`	2* ***C*** (1 per MPI process) (rounded up to nearest 2 if necess.)	4 GB per MPI process
C	***G*** GB per MPI process	`sbatch --partition=ibfdr --ntasks=`***C*** `[--constraint=nodetype] --exclusive --mem-per-cpu=`***G*** `--ntasks-per-core=1 jobscript`	2*C (1 per MPI process) (rounded up to nearest 2 if necess.)	2*G GB per MPI process

Parallel (MPI) process

Use $SLURM_NTASKS within the script

```
#!/bin/bash

module load meep/1.2.1/mpi/gige
cd /data/username/mydir

meme mini-drosoph.s -oc meme_out -maxsize 600000 -p $SLURM_NTASKS
```

OpenMPI has been built with Slurm support, so $np need not be specified.

```
#!/bin/bash

# this file is called meep.par.bat

# module for ethernet runs
module load meep/1.2.1/mpi/gige

cd /data/$USER/mydir
mpirun `which meep-mpi` bent_waveguide.ctl
```

Infiniband Jobs

```
sbatch --partition=ibfdr
       [--constraint=x2650]
       --ntasks=64
       --ntasks-per-core=1
       --exclusive
         jobscript
```

```
--partition=ibfdr (FDR, 56 Gb/s)
--partition=ibddr (DDR, 16 Gb/s)
--partition=lcp   (QDR, 32 Gb/s)
```

Demo: NAMD on Biowulf

- Parallel program using MPI
- Typically low memory requirements (< 1GB/ core)
- Benefits from high-performance, low-latency network.

http://biowulf.nih.gov/apps/namd.html

Demo

NAMD Scalable Molecular Dynamics

staff@biowulf.nih.gov