



# HPC at the Smithsonian

Introduction to Hydra

# Overview

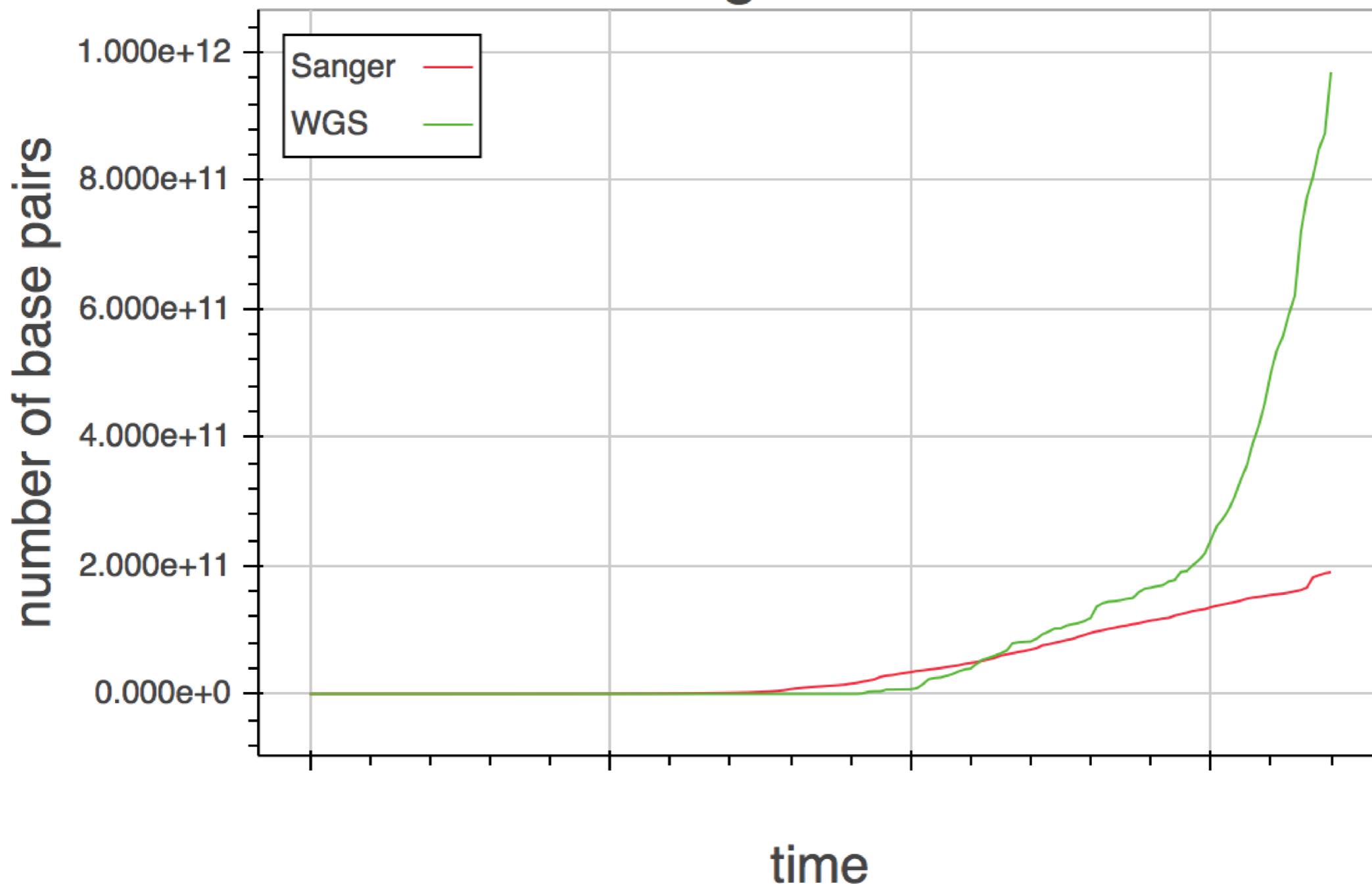
- Why HPC?
- SI Computing Cluster
- Support
- Cluster Architecture
- Submitting Jobs
- Best Practices
- Tips

# Why High Performance Computing (HPC)?

# Why HPC?



increase in genetic data

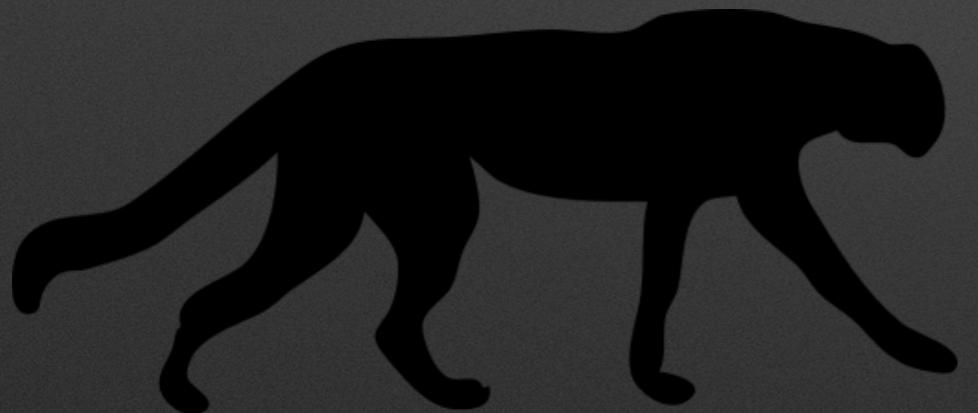
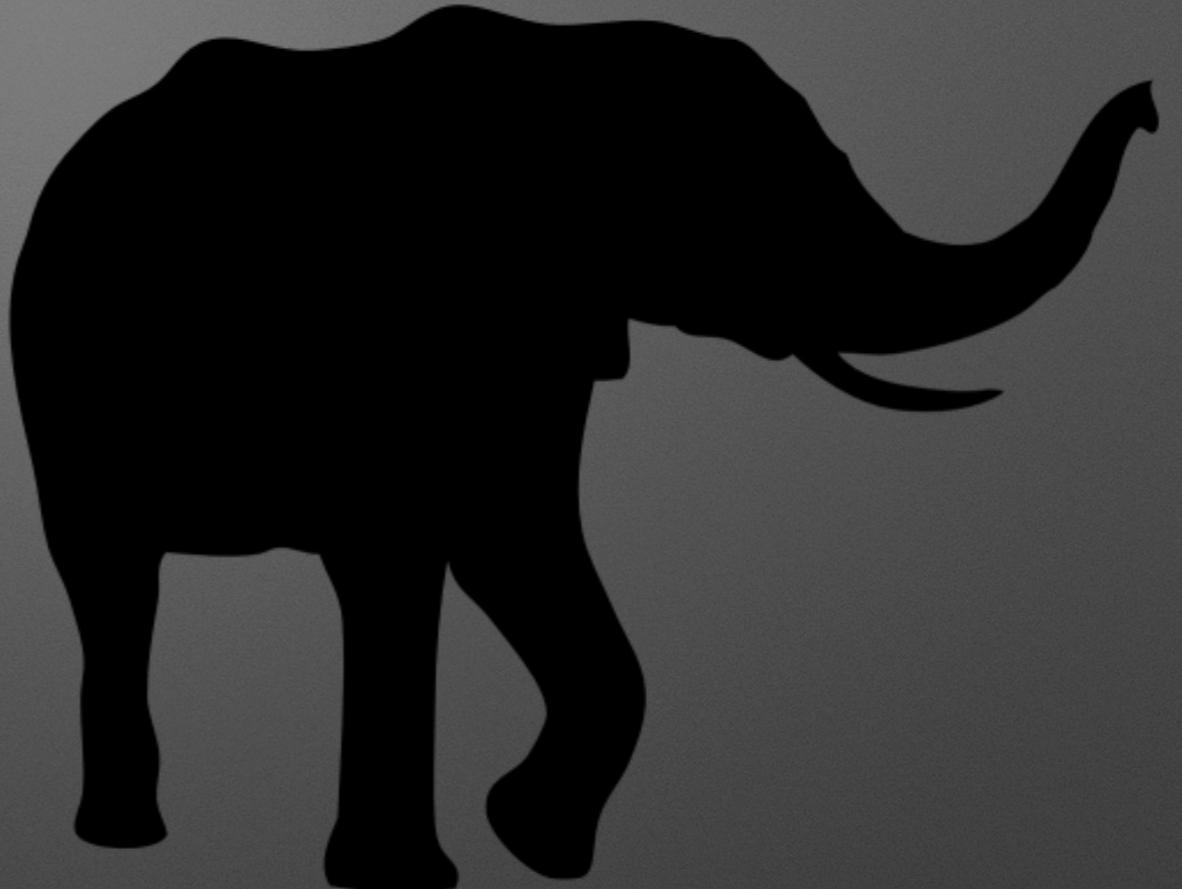


# Why HPC?



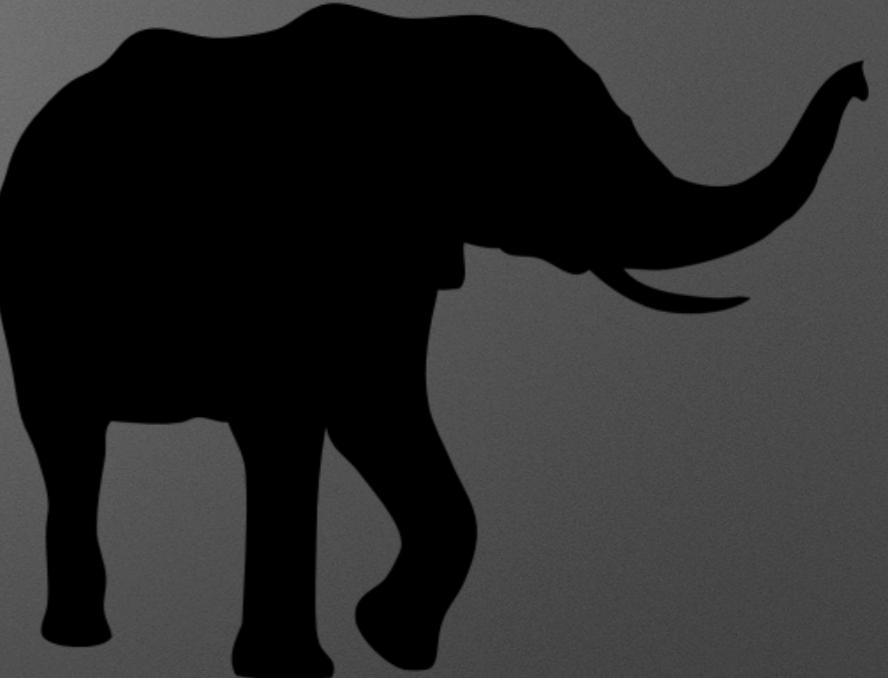
Size

&



Speed

# Why HPC?



- Size: some analyses use too much memory to be completed on a standard desktop/laptop, e.g. *de novo* genome assembly
- Speed: all about parallel computing! A problem that might take months on a desktop can take hours on a HPC, e.g. BLAST searches

# SI Computing Cluster

# Hydra



- Smithsonian HPC cluster
- 3,950 CPUs, 26 TB of RAM
- 16 high memory nodes (14 512GB; 2 1TB RAM; 1 2TB RAM)
- Many high CPU nodes with 64 CPUs



# Support

# <https://confluence.si.edu/HPC>

Confluence Spaces People Create

Pages / ... / Hydra and Lattice

Hydra Hardware

Created by Frandsen, Paul, last modified on Jan 21, 2015

Hydra

Hydra is the main computing cluster administered by the Office of Research Information Services. It consists of 10 racks of servers. The vast majority of these servers are general purpose servers. Due to a recent upgrade, it also contains several high CPU servers.

Server type	RAM per server	CPUs per server	CPU type	RAM per CPU	Number in cluster	Infiniband
Sun Fire X2200	16 GB	8	AMD Opteron	2 GB	175	no
HP DL165-G7	32 GB	16	AMD Opteron	2 GB	24	yes
Dell R415	32 GB	12	AMD Opteron	2.7 GB	24	yes
Dell R715	64 GB	24	AMD Opteron	2.7 GB	10	yes
Dell R815 (high CPU)	256 GB	64	AMD Opteron	4 GB	16	yes*

\*Four R815 servers in rack 0 are not configured with infiniband

Total number of CPUs: 3,336

Total RAM: 9,072 GB

Note: these numbers do not include the memory and CPUs on login nodes.

Lattice

Lattice is a separate (for now) cluster from Hydra that consists of both high CPU servers and high memory servers. The high memory servers are ideal for tasks such as *de novo* transcriptome or genome assembly, while the high CPU servers are well suited to analyses that can be run massively parallel such as reciprocal blast searches.

Server type	RAM per server	CPUs per server	CPU type	RAM per CPU	Number in cluster	Infiniband
Dell R815 (high CPU)	256 GB	64	AMD Opteron	4 GB	3	no
Dell R820 (high memory)	512 GB	24	Intel Xeon	20 GB	2	yes, software not configured

Total number CPUs: 240

Total RAM: 1,792 GB

Note: these numbers do not include the memory and CPUs on login nodes.

Like Be the first to like this

No labels

## Sysadmin support:

**DJ Ding, OCIO**

**Sylvain Korzennik, SAO**

**Rebecca Dikow, OCIO**

**SI-HPC-admin@si.edu**

## Application support:

**Rebecca Dikow, OCIO**

**Matt Kweskin, LAB**

**Vanessa González, GGI**

**Mirian Tsuchiya, OCIO**

**Mike Trizna, OCIO**

**SI-HPC@si.edu**

# Cluster Architecture

# Cluster Architecture

- Many computers connected to each other via an interconnection network (interconnect) with software that controls where jobs are run
- Each computer in the cluster is called a node
  - Login nodes: users log into these to access the cluster and launch jobs, they perform no real computation (`hydra-login01.si.edu`, `hydra-login02.si.edu`)
  - Compute nodes: actually run your program
  - Head node: runs the software that controls the cluster and the jobs

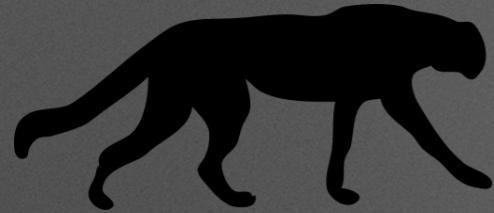
# Queuing System/Job Scheduler

- Software that schedules the jobs and submits them to the compute nodes.
- Hydra uses the “(Sun) Grid Engine” queuing system.
- Most jobs should be run in batch mode. To submit jobs, you use the command “qsub”
- There is also an interactive mode for short jobs. To initiate, type “qrsh”

# Submitting Jobs

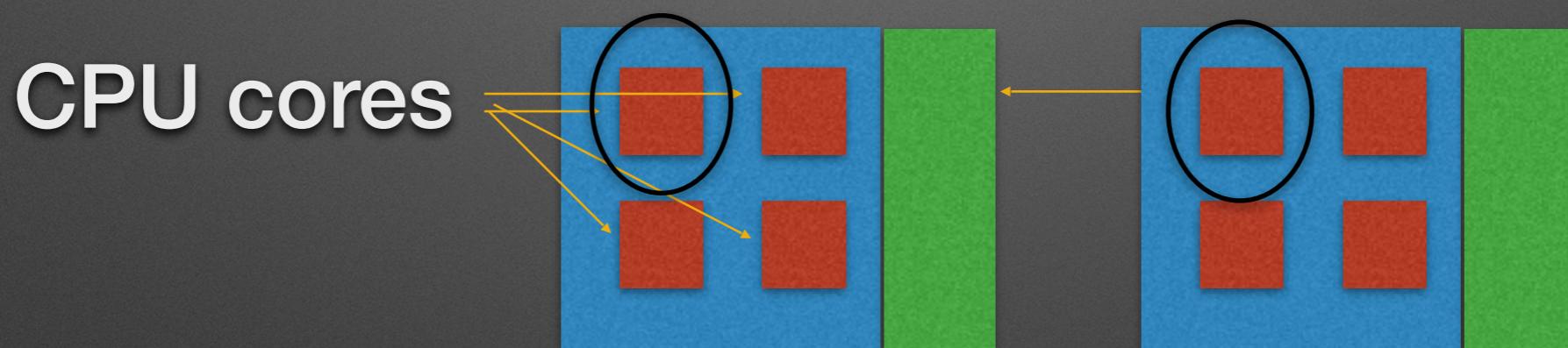
# Job Submission: You need to know...

- Number of CPUs/type of parallel environment
- Time your job will take
- Memory you would like to reserve
- Job specific information (module required, job specific commands)

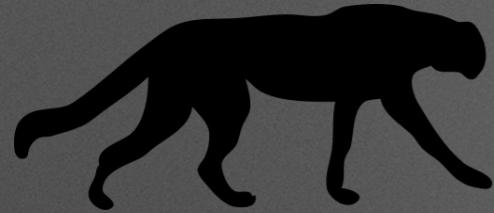


# Number of CPUs: Parallelization

- Naive parallelization (embarrassingly parallel)- one large job easily split into separate jobs.

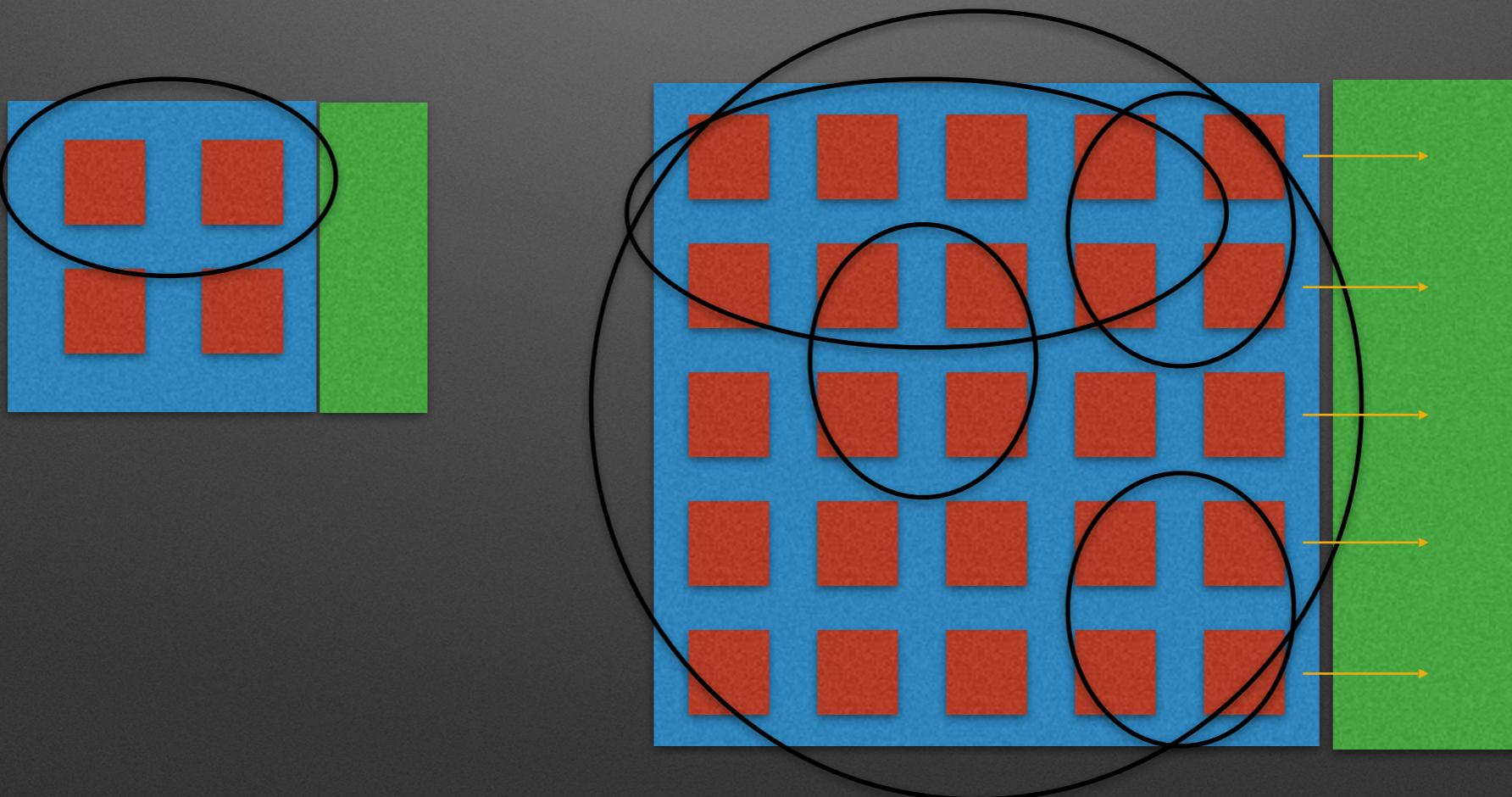


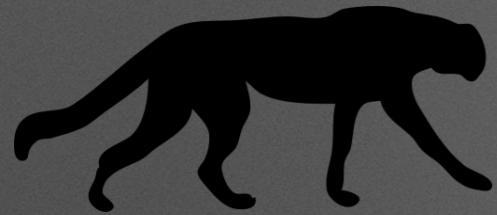
- E.g. many sequential blast searches at once, splitting bootstrap reps.



# Number of CPUs: Parallelization

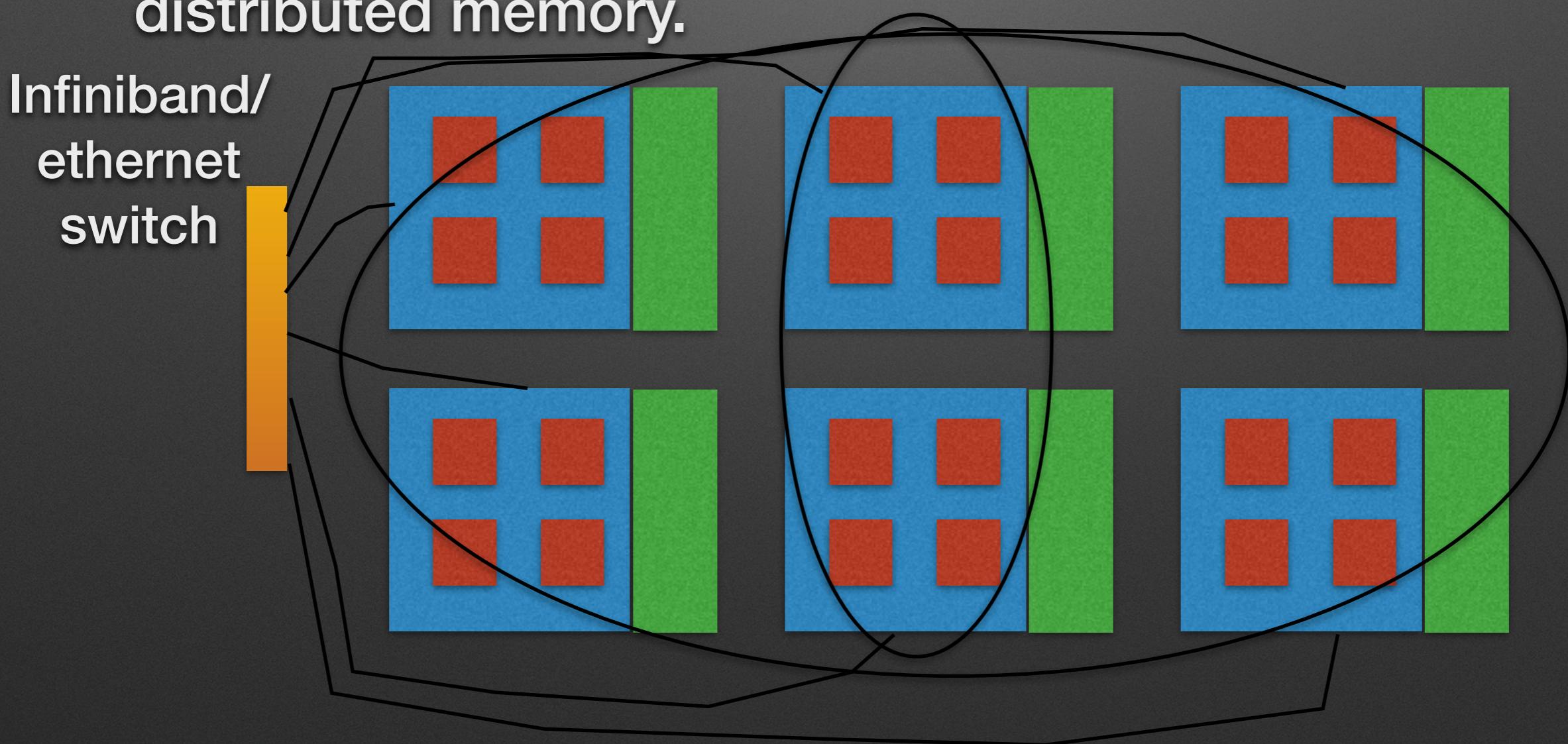
- Multi-threading (pthreads, OpenMP)-multiple CPUs on a single node. This method uses shared memory.

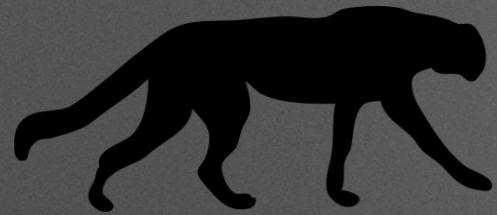




# Number of CPUs: Parallelization

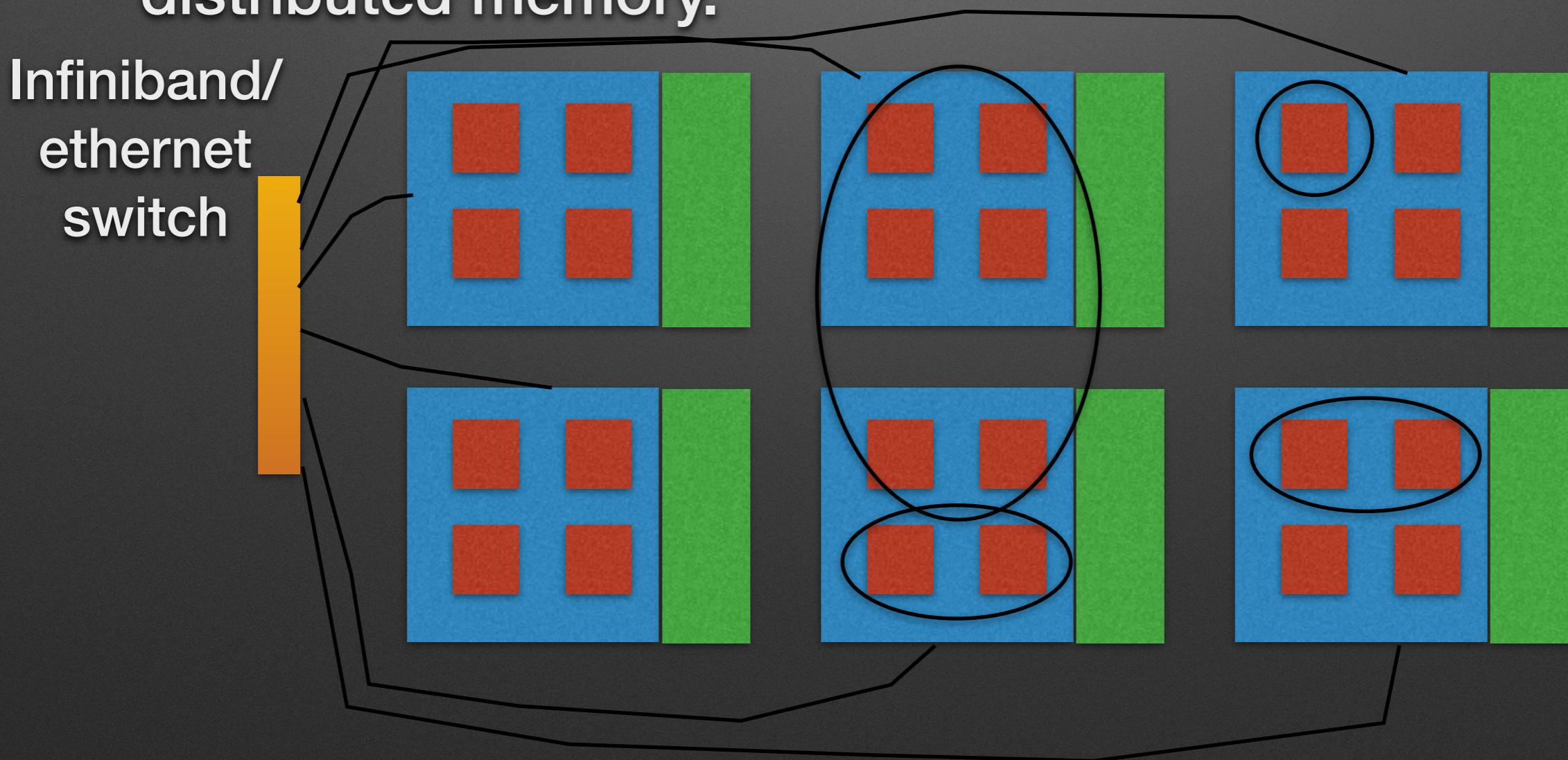
- MPI (Message Passing Interface)-multiple CPUs across multiple nodes. Messages are passed between nodes via the interconnect. This method uses distributed memory.

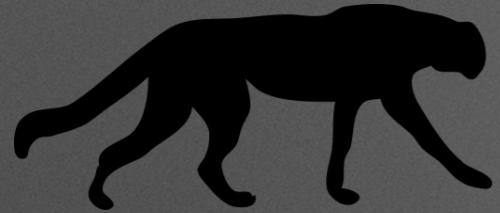




# Number of CPUs: Parallelization

- MPI (Message Passing Interface)-multiple CPUs across multiple nodes. Messages are passed between nodes via the interconnect. This method uses distributed memory.



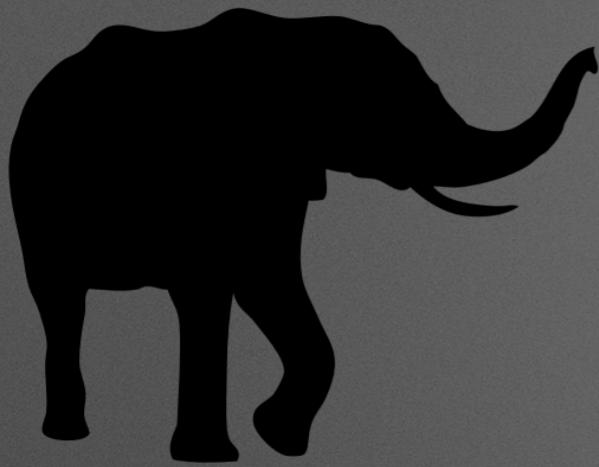


# Number of CPUs: Parallelization

- How do I know which one to use?
- *This is determined by the type of program you wish to run. Check the program documentation or check with us.*

# Time

- The cluster needs to be told estimates of run time
- Short jobs: 7 hours
- Medium: 3 days
- Long: 30 days
- Unlimited (there are very few of these...anything submitted to this queue should have some checkpointing capability)



# Memory use

- *Sylvain: “Memory is both an expensive and scarce resource.”*
- Important to give a good estimate!
- How to estimate?
  - Check with others, program documentation
  - Monitor test runs, try smaller run and check how memory use scales up

# **qsub flags**

- How you tell the scheduler about your job
- -cwd runs job from current working directory
- -j y joins the stderr to the stdout
- -o sampleOut.log if you have chosen to join stderr and stdout, your job will write these to the sampleOut.log file
- -N The name of your job. Use something informative!
- -m abe, -M example@si.edu email with job status

# **qsub flags**

- Parallel environment (-pe)
  - **MPI (CPUs spread across nodes)** -pe orte 10 or  
-pe mpich 10
  - **Threads (multiple CPUs, one node)** -pe mthread  
16
- **Run Time** -q <q-name> or -l s\_cpu <time>

# qsub flags

- Memory use specification:

- `-l memres=4G, h_data=4G, h_vmem=4G`
- `memres`: “Memory Reserved” This is for the cluster to track memory use and prevent other from grabbing memory you will need. Corollary, it prevents other from using that memory. *This is a shared resource.*
- `h_data, h_vmem`: Tells the system to terminate your job if it uses too much RAM
- For large memory jobs, you must indicate that you wish to use the high memory queue by adding `-l himem` this can be added to the other variable using the `-l` flag.



MEMORY IS PER CPU NOT PER JOB



- 16 threads with 4GB RAM each means 64GB total

# Modules

- **Modules are a way to set your system paths and environmental variables for use of a particular program or pipeline.**
- **You can view available modules with the command:**  
\$ module avail
- **To find out more about a module you can use:**  
\$ module whatis <module\_name>  
\$ module help <module\_name>  
\$ module show <module\_name>
- **To load a module:**  
\$ module load <module\_name>

# QSub Generator

The screenshot shows a web browser window titled "hydra-3.si.edu" displaying the "QSub Generation Utility". The interface is a form-based generator for creating qsub scripts. It includes fields for specifying job parameters like time limit, memory, and number of CPUs; selecting PE type (serial, MPI, multi-thread); choosing shell (bash, sh, csh); adding modules; entering job-specific commands; and defining additional options like job name, log files, and email notifications. Below the form, a code editor displays the generated qsub script, which includes comments for parameters, modules, and commands, along with specific echo statements for tracking the job's start and completion. At the bottom, there are buttons for "Check if OK" and "Save it", and a small footer note about the QSub generator PHP version.

Specify the amount of:

CPU time  or  time limit;  
Memory  GB

Select the type of PE:  serial  MPI (orte)  MPI (mpich)  multi-thread  
Number of CPUs

Select the job's shell:  bash  sh  csh

Select which modules to add:

select from the list, or start typing

Job specific commands:   
e.g. myprogram -p \$NSLOTS -o myoptions

Additional options:

Job Name   
Log File Name   
Err File Name   
Change to cwd  Join stderr & stdout  Send email notifications  
Email

This is the corresponding qsub script:

```
# /bin/csh
# -----Parameters----- #
#
# -----Modules----- #
#
# -----Your Commands----- #
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
#
#
echo = `date` job $JOB_NAME done
```

This page is ready!

©2015 Smithsonian Institution HPC - OCIO (SGU/PF)  
QSub generator PHP ver 1.0/1/150522 IS ver 1.0/1/150522

<https://hydra-4.si.edu/tools/QSubGen/>

# Best Practices

# Best use practices

- Ramp up your use slowly to monitor the resources used by your jobs (CPU, memory, disk space)
- Run all analyses through `qsub`, i.e. NO analyses other than simple scripting/moving around should be done on the login nodes.
- Upload your files directly to the /pool drives, not your home directory.
- Check available disk space before you upload large files.
  - do this with:  
`$ df -h /pool/genomics` (**df=disk free**)
- Cleanup disk use following analysis
  - Remove files after transferring them back
  - Avoid leaving lots of small files (use `tar!!`)
- Understand the usage requirements of the programs that you are running and use reasonable `-l` flags for memory and compute time.

# Transferring files

- You can use scp, sftp or rsync
- Important!
  - Do NOT store large files on your home directory.
  - When given a user account, you will also be given a folder to place your files (e.g. /pool/genomics/USER).

# Tips

# Helpful commands

- Show info for all of your running jobs:  
\$ qstat -u \$USER -s r
- Display info for a specific job: \$ qstat -j <job\_number>
- Check queue usage: \$ qstat -g c
- Check job stats after complete: \$ qacct -j <job\_number>

# Helpful commands

- **Delete a job:** \$ qdel <job\_number>
- **Delete all of your jobs:** \$ qdel -u \$USER