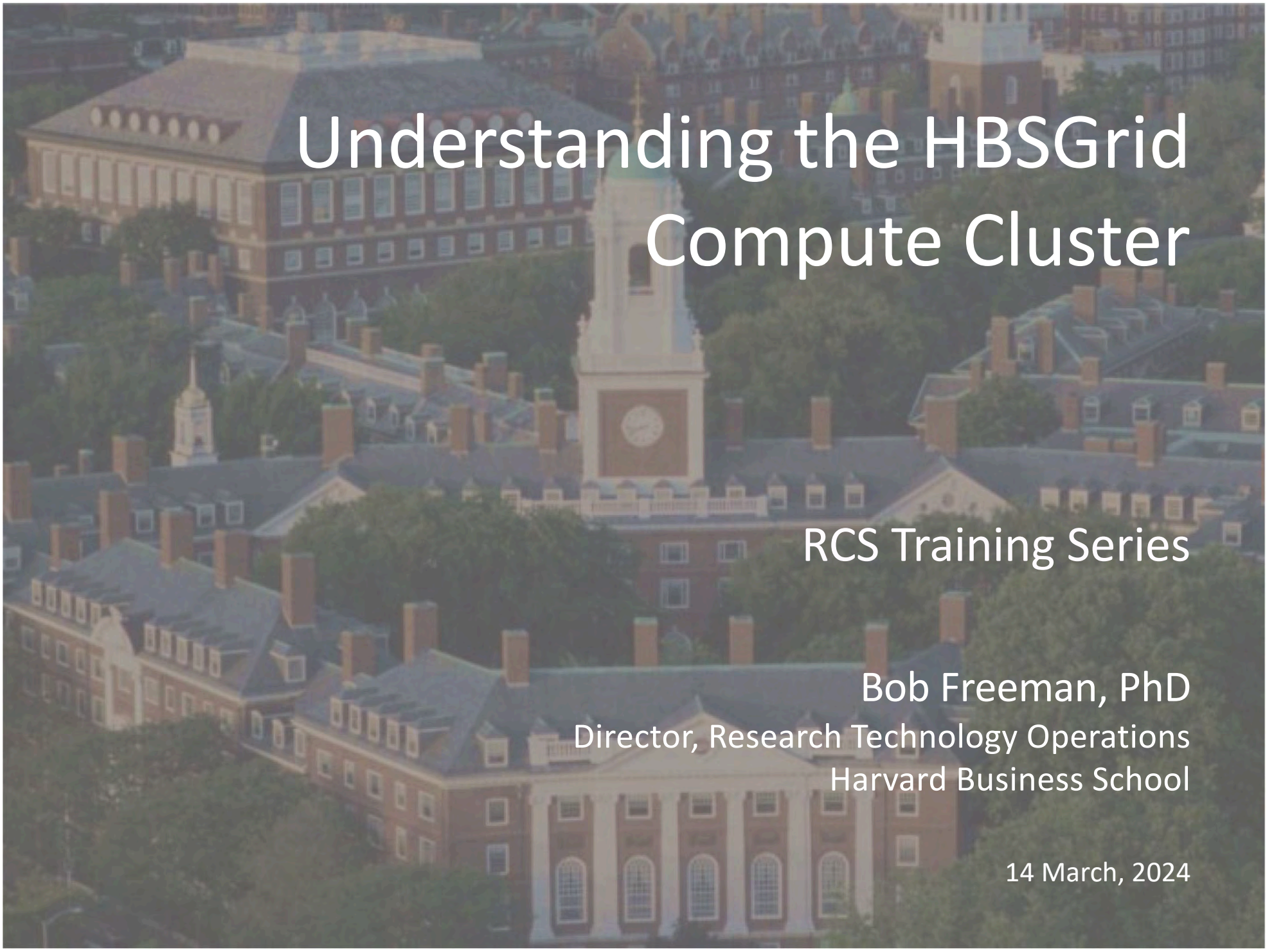


Understanding the HBSGrid Compute Cluster

RCS Training Series

Bob Freeman, PhD
Director, Research Technology Operations
Harvard Business School

14 March, 2024

Today's Training…

Agenda

- Introductions, Q&A
- What do I need to do? Or, how do I use this?
- Where do I store my data?
- What is different than using a desktop/laptop?
- How do I choose RAM sizes and # of cores?
- What if I'm working with large datasets? Or want to use GPUs?
- What do I do if things don't work as expected?

Goals

- Understand the technology you are using
- Learn how to get information on jobs and control them if needed
- Look at RAM & CPU usage from previous jobs to inform choices for future work
- Understand considerations for parallel processing (& GPU usage)

Ultimately… **help you be more efficient and successful with your research!**

We're going assume you've been doing work for a few weeks. No worries if not!

Additions to Training

You will get the most out of this training if:

- You've watched the pre-class video
- You've been working on the cluster – NoMachine or terminal – at least for several days and have run applications or submitted batch jobs

Other topics that frequently are raised:

- (Remote) Editing & switching between laptop/desktop & cluster
- Maintaining project specific settings, modules, packages
- Coding, importing data, & choosing resources
- Handling large data

And some considerations:

- Networking/Firewall
- Secure data environment
- Permissions/Restrictions on capabilities

HBSGrid Cluster & Projects

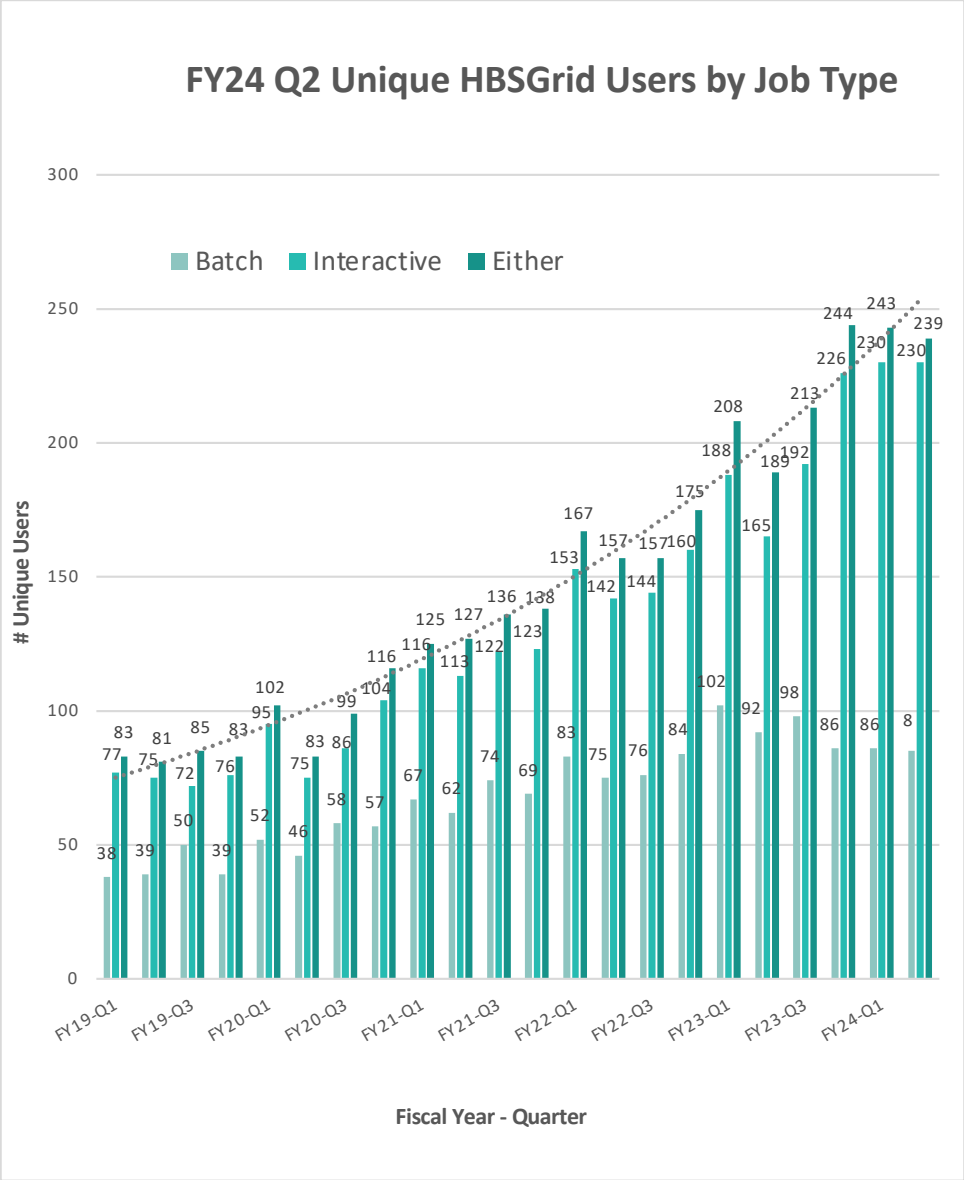
HBSGrid Cluster	
Node Count	16
Total CPUs	684
Total GPUs	5
Total RAM (GB)	9840
Total Storage (TB)	320
Total Accounts	~700

HBSGrid	
Total Projects	472
Active Projects*	221
Total Space Used	- TB
Space Used by Active Projects	172 TB

*defined as modifications made within the last year

Current as of FY23Q4-24Q1

HBSGrid Usage



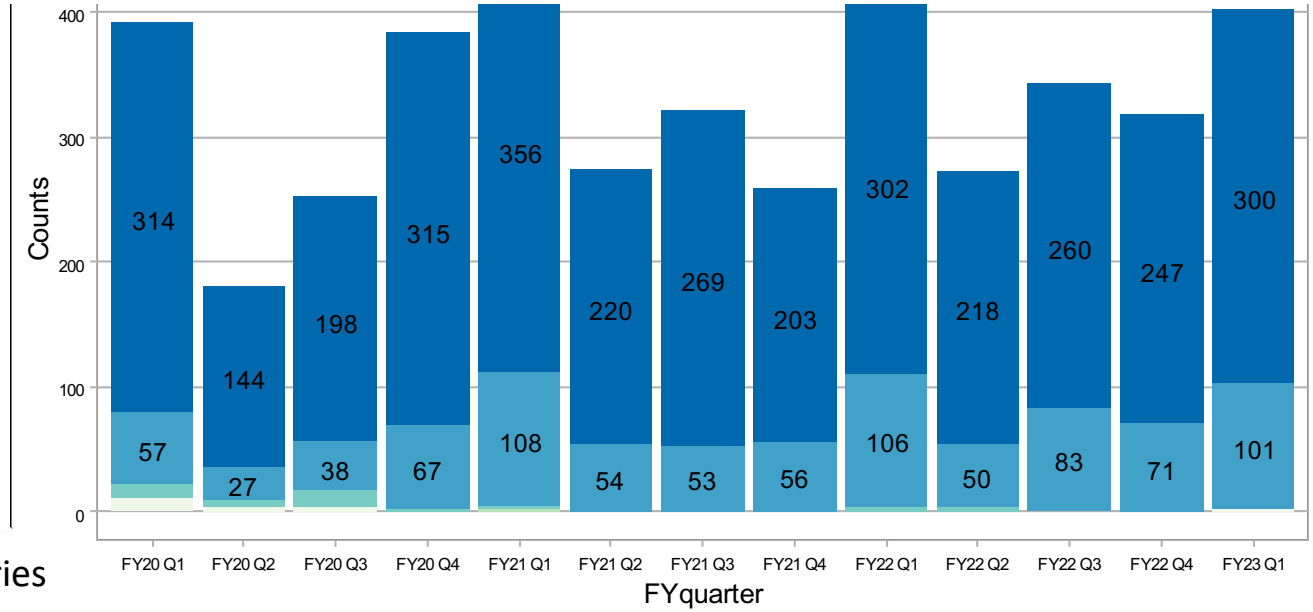
Activity	
Active Accounts	782
Accounts, by Type	Faculty: 172 Doctoral: 145 Staff/RAs: 226 Guest: 196 Other: 38
Unique users active in the last quarter*	175
CPU Utilization**	38.4%
Total Work Accomplished (CPU yrs)	85.1
Max / Mean / Min Interactive Utilization	83% / 39% / 8%

* ran an application or job ** this includes (high-cpu) batch and (low-cpu) interactive sessions

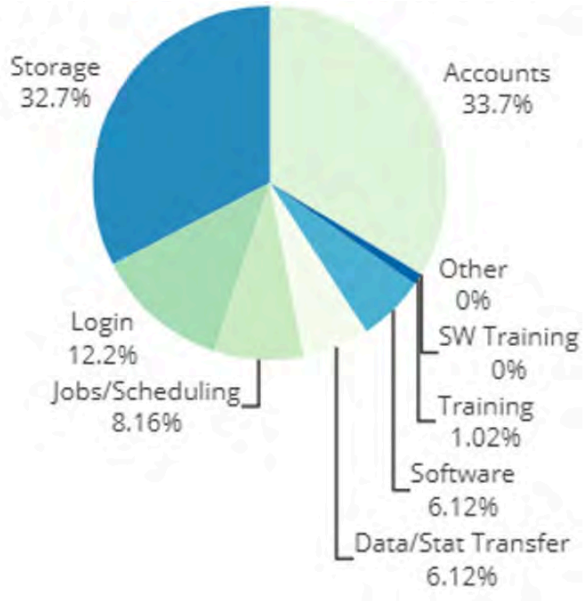
Current as of FY24Q2

Current as of FY23Q1

RCS Research Inbox: Requests & Inquiries

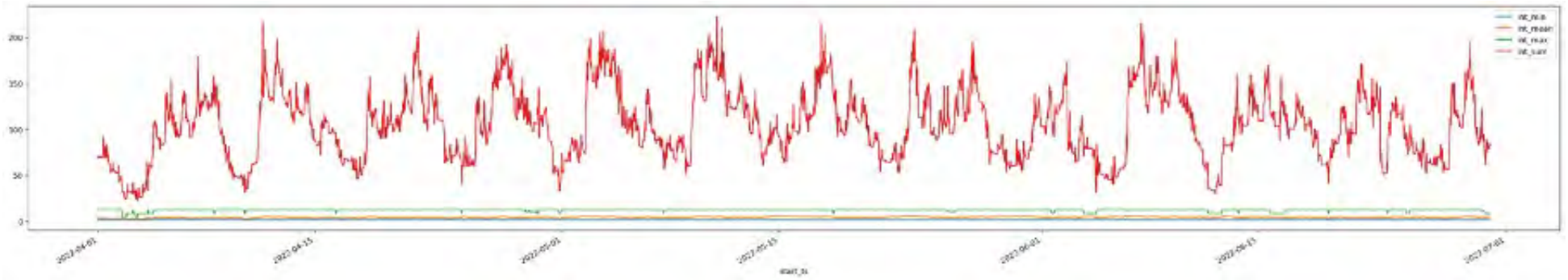


FY23Q1: Research Inbox HBSGrid Inquiries



HBSGrid Cluster Usage

FY22Q4 Interactive Session Core Usage, Cumulative



Method: This graph illustrates the cumulative core usage by interactive sessions over the course of the quarter as determined by the job completion date. Sliding windows of 30 minutes were taken and the number of core in use for interactive was summed and graphed. Periods during maintenance windows were removed in order so that other low use periods could be seen.

Grid 3.0

- A User-Interface Overlay to improve the usability of the cluster
- Provides a larger selection of analysis tools, programming languages, and statistical/development environment
- Includes recent Python, R, Stata, and JupyterLab/Notebooks
- *de facto* standard for new users (Apr 2023) & opt-in for others
- Easier to:
  - Launch interactive sessions on the cluster (== interactive jobs to the scheduler)
  - Switch effortlessly between different versions of a program
  - OR Freeze the version you are using in order to promote reproducibility
- For more information:
  - Documentation https://hbs-rcs.github.io/hbsgrid-docs/
  - QuickStart https://hbs-rcs.github.io/hbsgrid-docs/userguide/quickstart/
  - Demo video https://hbs-rcs.github.io/hbsgrid-docs/3.0/#hbsgrid_v3.0_demo

NB: Training information that will be deprecated due to UI Overlay will be indicated in grey background shading.

Important Points

The HBSGrid Computer Cluster is a **great start** for using advance computing resources (ACI). But there are things to remember:

- NB! This is a **shared resource**, which means shared responsibility for fair use & security
- We promote not only fair use, **but also fair access**. Due to limited resources, we encourage
  - Researchers to leverage batch, background processing, to free up resources, and
  - Starting "Interactive" sessions -- work that is truly interactive – with lower CPU counts than normal
- **This is a Level 3 & 4 Secure Data / HRCI environment**
  - Ensure that your compliant with security guidelines, respect the policies / ToS
  - Be cautious with code you have not written and/or have obtained external.
  - **Tight network restrictions are in place: most ports are closed**. Some exceptions (e.g. http/https)
- Job information and control only happens through the terminal*
  - If problems, remember `bjobs` and `bhist` (& your friends at RCS!) in addition to job reports.
  - Do not be afraid! We have a *Compute Grid: Unix Cheatsheet* class!
- If parallel processing via terminal, ensure that you use the correct command sequence:
  - Reserve CPUs as job submission option
  - Use `LSB_DJOB_NUMPROC` environment variable in code
- Do not run >20 jobs that have heavy disk I/O in home or project folders. Please use our performant, SSD `/export/scratch` volume
- Talk to us about workflow and usage questions! Guidance is FREE!
- Perhaps FASRC's Cannon might be more appropriate for the volume & shape of your work

What is the HBSGrid?

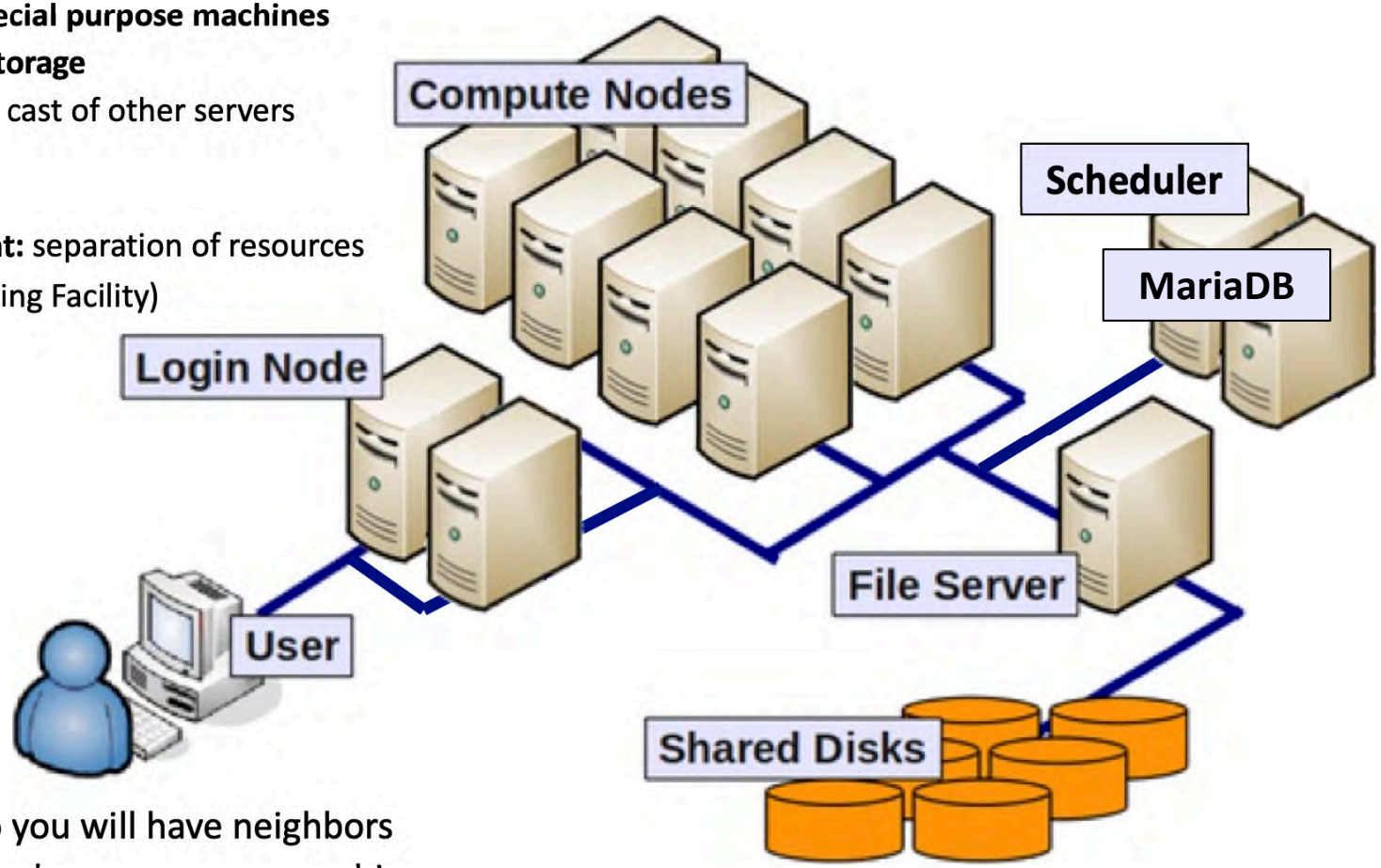
The cluster is a collection of various types of expensive hardware used to create our secure data environment:

A number of tightly interconnected **machines with similar or identical hardware**

- Several high-powered, **special purpose machines**
- Large amount of **shared storage**
- Miscellaneous supporting cast of other servers

And software:

- **User / group management:** separation of resources
- **Scheduler: LSF** (Load Sharing Facility)
- Linux OS (RedHat 7.x)



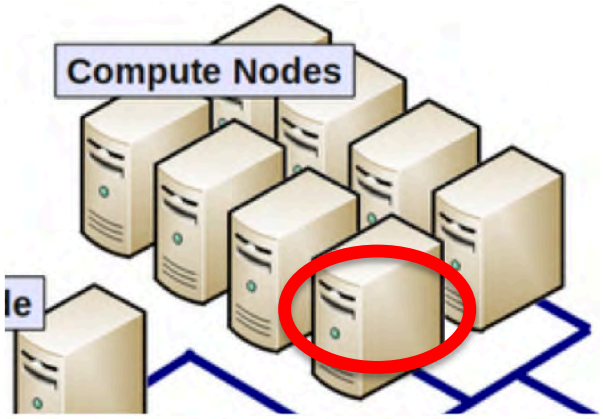
Caveat:

- It's a shared system – so you will have neighbors
- This is not like running code on your own machine.

Key Definitions

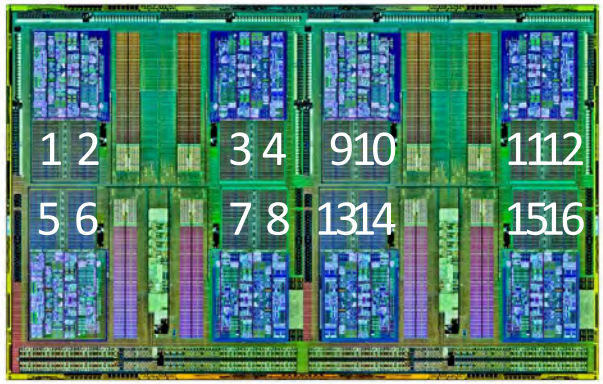
The typical hardware unit is called a **node**

- Same tech that's in a desktop/laptop: CPUs, memory, hard drive, network cards
- But more powerful and more of them compared to a typical desktop
- Nodes are individual hosts with distinct names. E.g...
  - `rhrcscli1`: one of the login nodes
  - `rhrcsnod7`: one of the compute nodes



The basic computational unit in a cluster is a **CPU core**

- Each core runs one process, an *average* job
- Most nodes have 32 cores arranged on 4 CPUs (8 cores /CPU)
- Thus, most nodes run 32 jobs (batch/interactive)



A **typical compute node*** is configured:

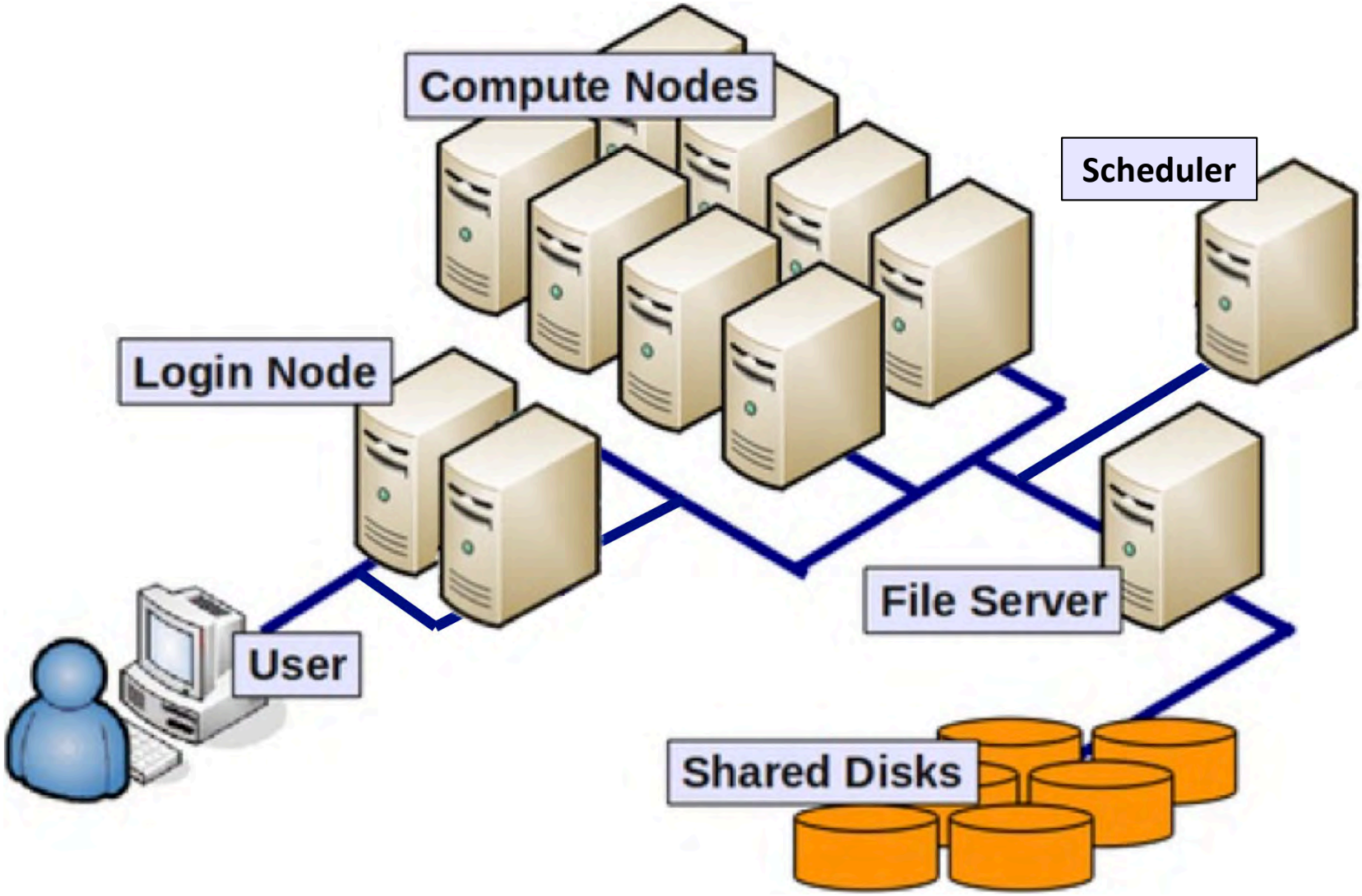
- 32 cores + 256 GB RAM (default ask = 8 GB RAM/core)
- 1 network card to communicate within the data center
- Small, local hard disk/SSD for boot and local /scratch

All cores on a node share all other resources of the node: memory, network bandwidth, etc.

Thus, how you use these resources affects the other 31 jobs on that compute node

*We have other differently-shaped nodes used for specific purposes

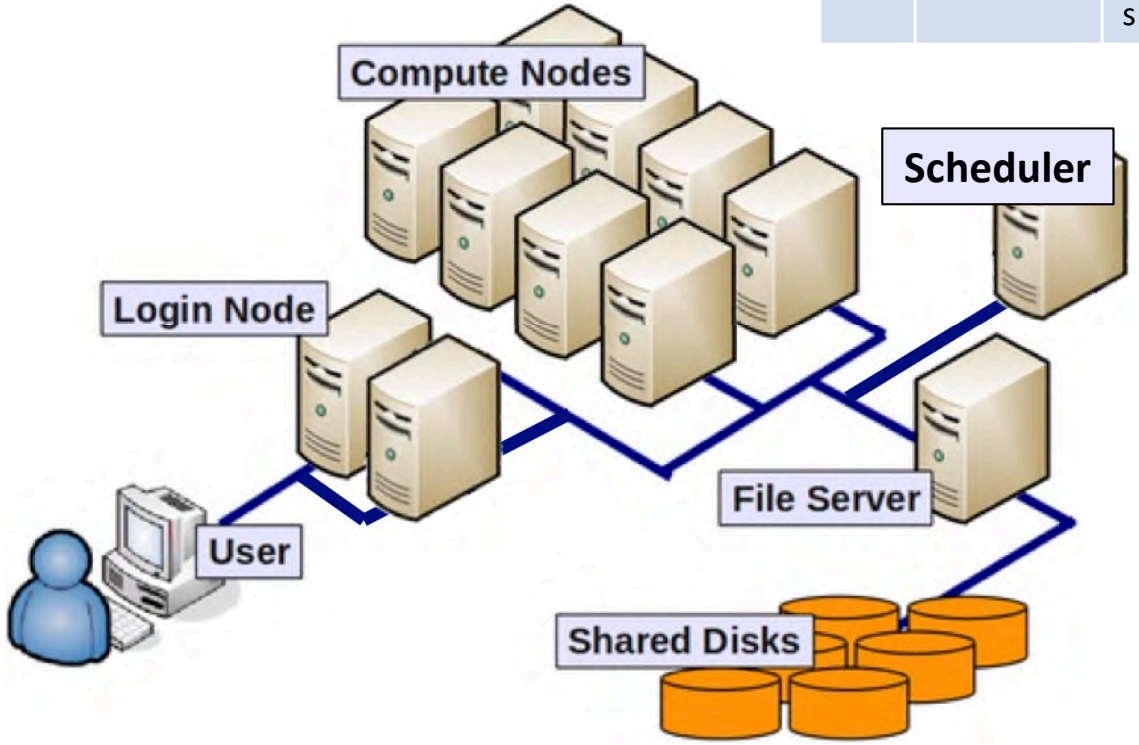
A Closer Look…



A Closer Look…

Count	Role	CPU	RAM	Other
2	Login nodes	16 cores	128 GB	
14	General purpose	32 cores	256 GB	
1	GPU node	64 cores	256 GB	5 Tesla V100s
4	High memory	56 cores	1500 GB	
1	SAS node	16 cores	128 GB	Large local disk for file streaming

Count	Role	CPU	RAM	Other
2	Queue nodes	16 cores	128 GB	Redundant



A Closer Look…

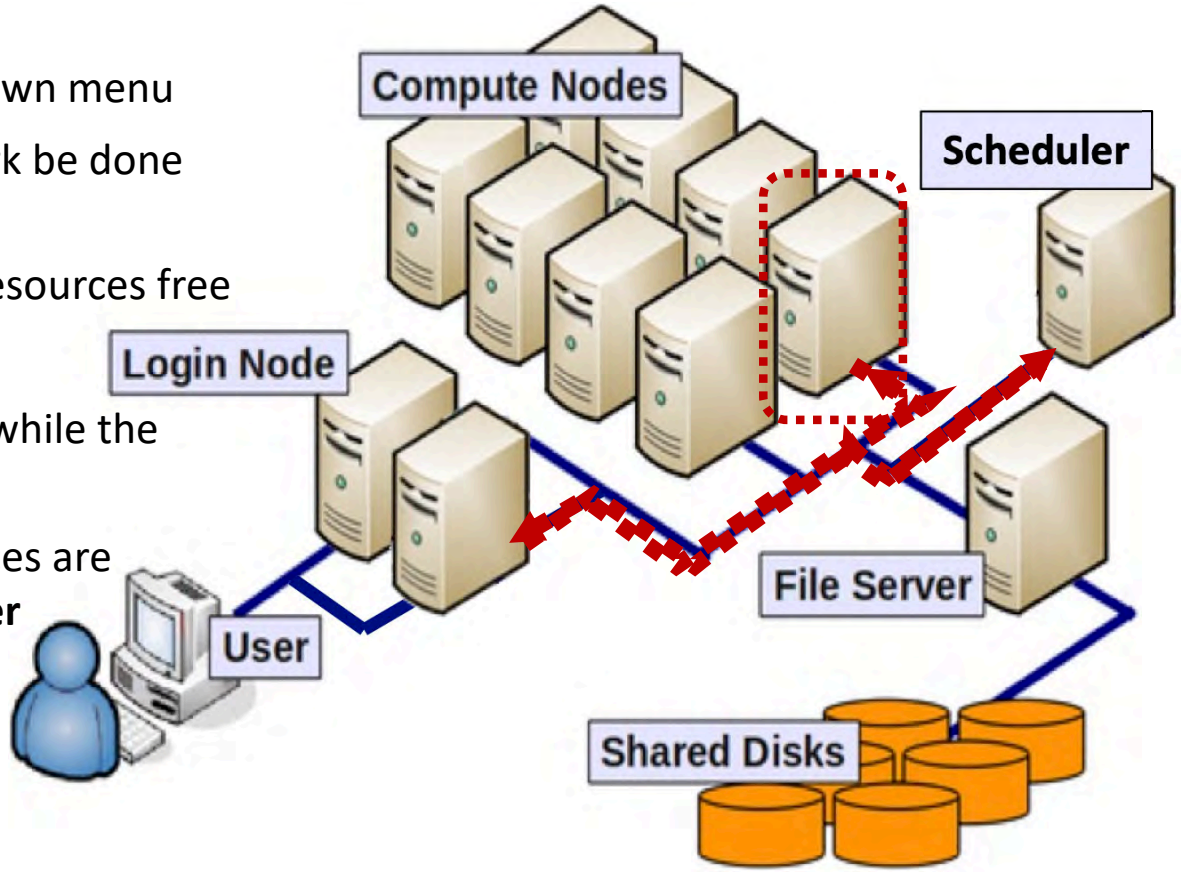
Using a cluster is very different from your desktop/laptop

- You own, manage, & control all the resources on your local machine,
- The scheduler must understand your needs, find the right spot, and handle the logistics for you.
- A good analogy - taking your car (w/ passengers) on the ferry to Martha's Vineyard.

The flow…

1. Run program from Application pull-down menu
2. Scheduler is contacted to request work be done (interactive program)
3. Scheduler finds compute node with resources free
4. Job is sent to compute node*
5. Compute node starts executing code while the display still appears on the login
6. Program continues to run and resources are unavailable **until explicitly quit by user**

* only if resource limits aren't hit



Getting Work Done…

All work on the cluster is thought of as jobs

job == unit of work that the scheduler handles

Your work can be divided into two types:

- *Interactive* == you, a person, is working directly with the program
  - Can be done via GUI - typical point, click, and type programs
  - Or via terminal, shell, console: terminal/SSH login or a terminal session inside NoMachine
- *Batch/background* == you submit a job to run independently in the background & is self-contained (has needed inputs), writing out data to files or logs; it is (usually) error free
  - Code is usually written and tested interactively
  - Is then submitted as a job to run once, multiple times, etc on compute nodes

Getting Work Done…

Interactive

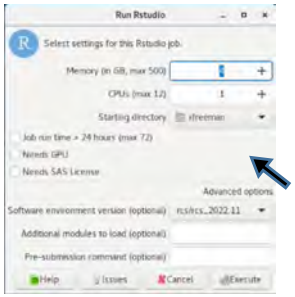
1. "Launchers" in Grid-3 (UI Overlay)
    1. This supersedes the "wrapper scripts" on login nodes. These had been used to help users execute applications easily & not on the login nodes
2. In a terminal window (either via terminal SSH or in NoMachine/Gnome)…
    - Executing a command-line "wrapper script" without naming a file

```
Rstudio-5g
xstata-se-10g   # xstata- does GUI, stata- does terminal
```

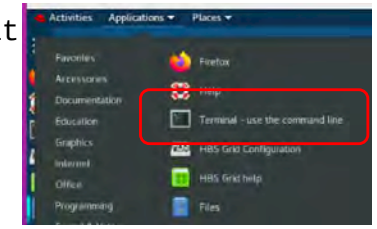
    - Executing an LSF command that includes the `-Is` or `-Ip` flag and is sent to an interactive queue

```
module load rcs
bsub -q long_int -Is -M 5000 –hl rstudio &   # assumes 1 core & 5-8 GB default
```

queue RAM program



Start application…



Run terminal app…

Batch

1. Cannot be done through a standard GUI (point-and-click). Yet.
2. In a terminal window…
    - Executing a command-line "wrapper script"* and naming a file. (Note: not all wrappers have this.)

```
matlab-5g myfile.m
```

    - Executing an LSF command that omits the `-Ip` or `-Is` flag and is sent to a batch queue

```
module load matlab
bsub -q long -M 5000 -hl matlab -r "myfile"  # again assume 1 core and 5-8 GB default RAM
```

queue RAM program

* Exceptions: `spyder3-5g somefile.py` opens the file for interactive editing

https://hbs-rcs.github.io/hbsgrid-docs/commandline/

Items deprecated, but still useful, in Grid-3.0

Working on a Cluster: Mental Model Transitions

There are different phases/mental models when transitioning from desktops/laptops:

1. *(Start) Everything works, as is:*
    a. You control all the RAM, cores, and run time
    b. Disk is local and fast (mostly)
    c. You control software choices and versions

2. *(Novice user) Everything is new, working to piece together new (old) work patterns:*
    a. GUI offers most applications and are easy use; but one cannot update/modify centrals ones
    b. Gauging the RAM & core sizes needed to "do work" for each job is foreign & hard to judge*
    c. Remember that disk / folders are across the network on networked volumes (mounted filesystems) and not local to the machine, causing some delays
    d. Must take different measures to do any troubleshooting

3. *(Experienced user) More familiarity with resource asks and leverage capabilities:*
    a. Viewing LSF job logs on recent jobs to understand & diagnose crashes
    b. Viewing LSF job logs on recent jobs to understand RAM and core/CPU usage
    c. Using LSF commands to ask for wider variety of and more flexible run configurations
    d. Using software modules to access all available software titles and versions

*though you're equipped to make good guestimates!

Transitioning to a Cluster

Additional things to consider when moving from local to remote computing:

1. Be familiar with the topology & technologies of the environment
   - Understanding how tools work ensures less frustration and better chances of success.
2. Use thoughtful coding practices to promote transparency & reusability
   - Write code and scripts in small, reusable chunks, instead of long, monolithic files
   - Things will break and you'll make mistakes. It is easier to repeat small tasks.
   - **Use logging capabilities** in published modules & practices. Or at least basic print statements.
   - **Don't use remote editing capabilities that run code on the login nodes** (e.g. VSCode)
3. Make use of good data management practices as early as possible
   - Working on a team on years-long projects requires discipline and communication.
   - Use a good project structure
   - Document your work, your code, your data
4. Think about using version control and automating work. (Seriously!)
   - Let the version control tools manage the changes & the reasons; don't keep them in your head
   - Start simply. Only add complexity if you need it and are ready to do so
5. Think / work smartly to streamline and accelerate your workflows
   - Leverage the capabilities: use SSD scratch, switch to batch jobs, scale your code across multiple cores, etc
   - Understand & eliminate/avoid bottlenecks (e.g. importing/exporting CSV)

Repetition is the nature of and the *re* in *research*.

Less manual work means more time doing the things we enjoy the most.

If You're New to Compute Clusters

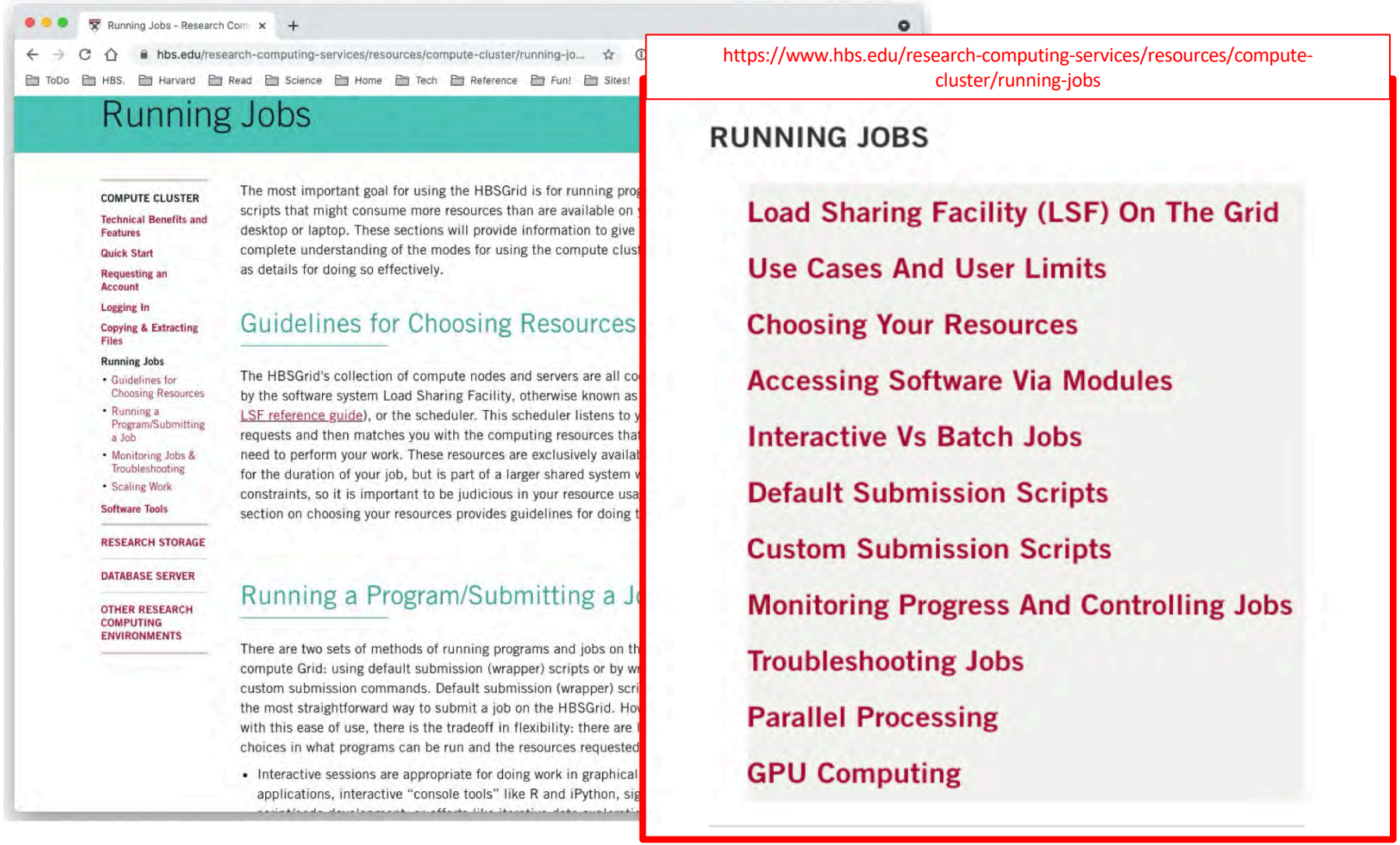
Don't be fooled! This is not a "beefier" version of your local machine. You will be more successful if:

1. Take time to review the Quick Start and other documentation.
   - Understanding how tools work ensures less frustration and better chances of success.
2. Be mindful when choosing program options (RAM, cores, etc)
   - More is not better
   - When you over-ask and not use the resources, other people are waiting for you to finish
3. Develop (good) habits to promote the stability & utility of the cluster:
   - Exit out of (interactive) applications when pausing work for several hours or more
   - Log out of NoMachine when possible (instead of disconnecting)
   - Please use Google Chrome instead of Firefox when using NoMachine
   - Don't overload directories with tens of thousands of files (or more!)
   - Avoid running or debugging code on the login nodes, including remote VSCode sessions
   - Review your past memory / core usage and adjust your future job resource asks appropriately
4. Include info in your requests for assistance that help paint a picture of your problem
5. Briefly review the Terms of Service, Policies, & Acceptable Use

Cluster Anti-Patterns

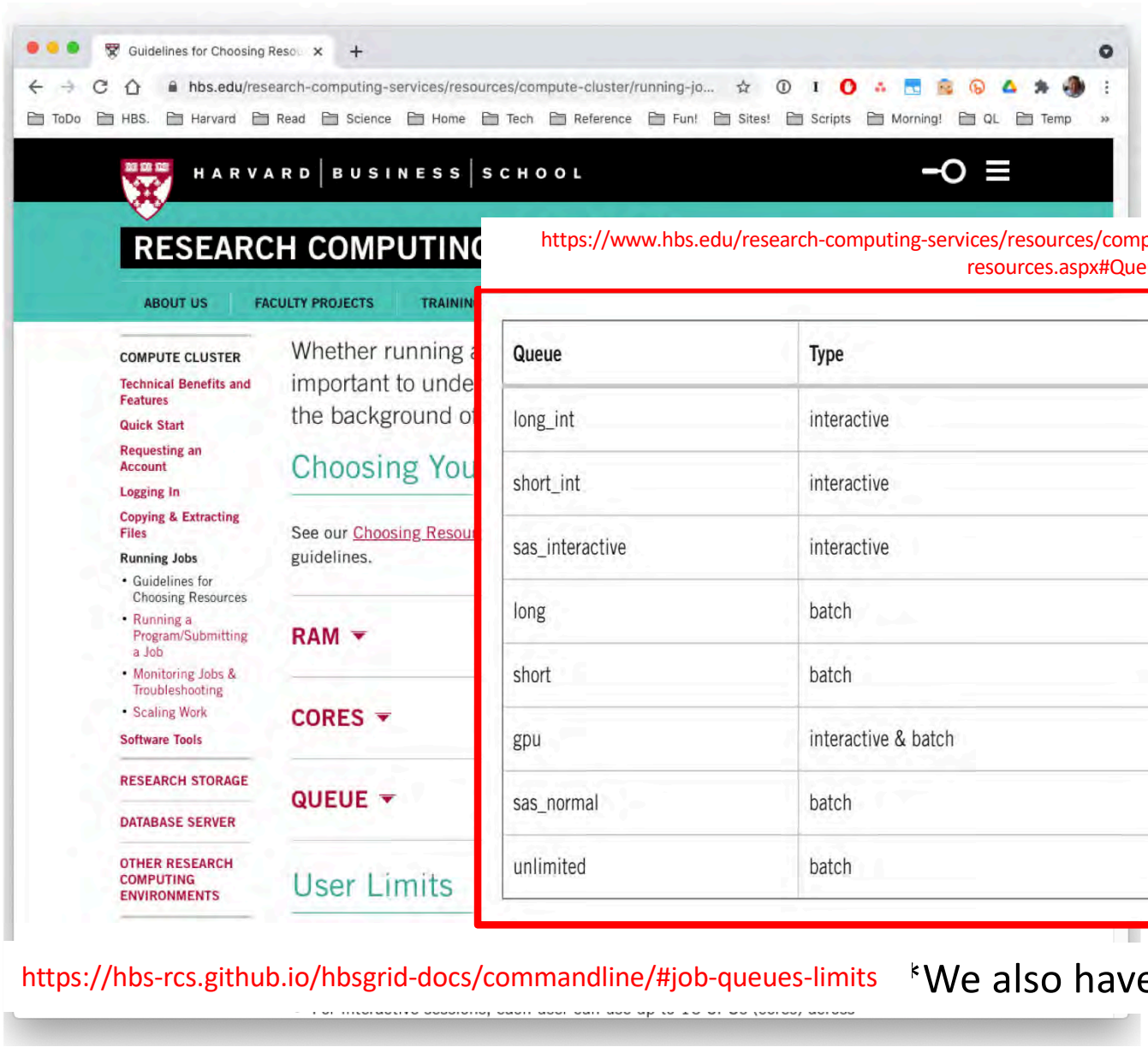
- Using Sudo or anything for elevated privileges
- Using more core or processors or threads and a number of cores you've requested
- Asking for large amounts of RAM and using very little
- Asking for a large numbers of cores, and using a fraction of them
- String more than 10,000 files in one directory
- Making calls to a network or Internet resource without pausing and two frequently
- Importing an exporting day as text every single session
- Creating a top-level folder on scratch or Globus, and leaving it readable to the world
- Starting interactive jobs with large ram or core asks, and leaving them idle for more than several hours or overnight
- Using the same NoMachine session day after day for a week or more

Getting Work Done…



https://www.hbs.edu/research-computing-services/resources/compute-cluster/running-jobs

Getting Work Done…



https://www.hbs.edu/research-computing-services/resources/compute-cluster/running-jobs/guidelines-for-choosing-resources.aspx#Queue

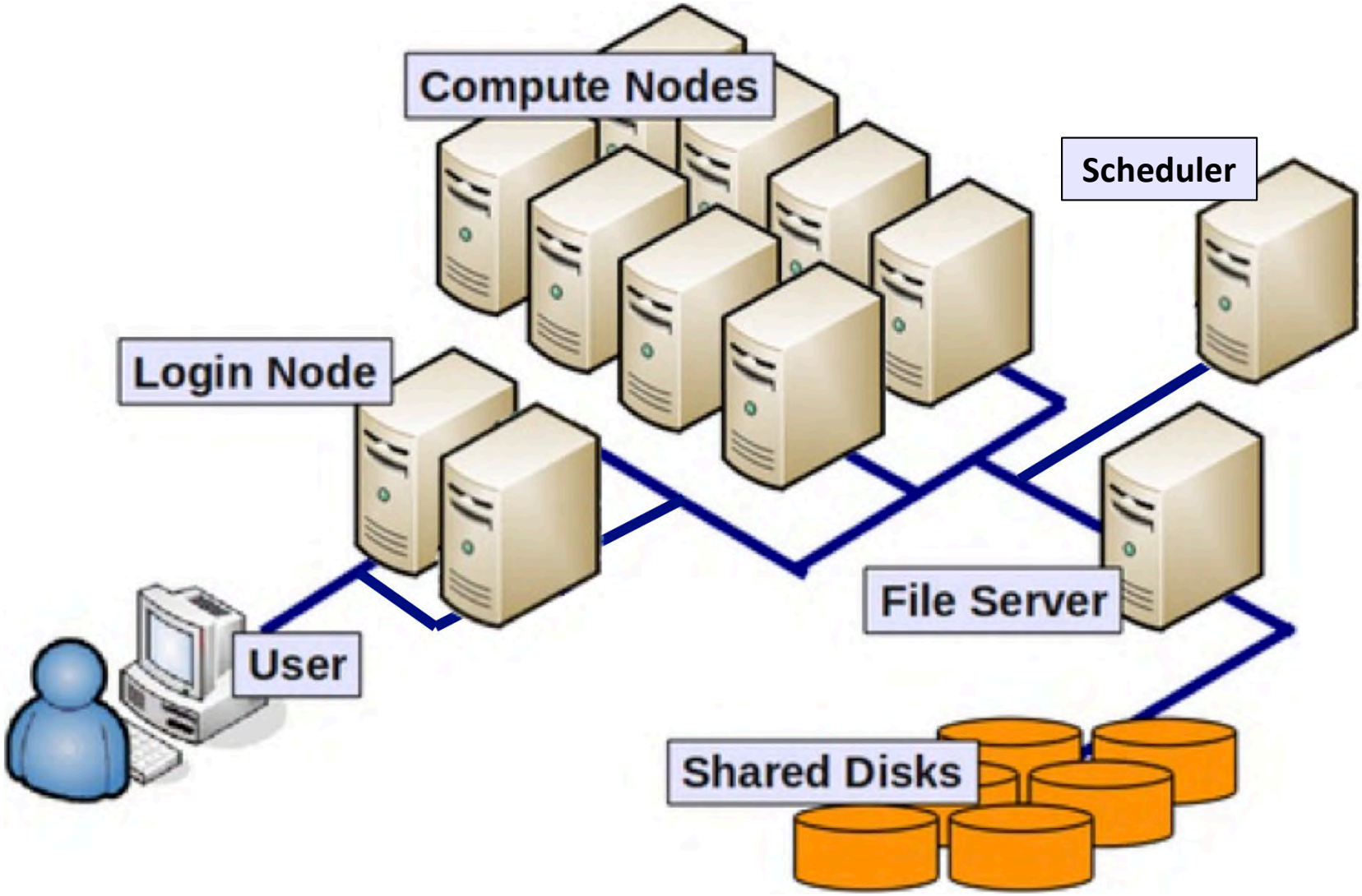
Queue	Type	Length	Max Cores/Job
long_int	interactive	3 days	4
short_int	interactive	1 day	12
sas_interactive	interactive	no limit	4
long	batch	7 days	12
short	batch	3 days	16
gpu	interactive & batch	no limit	4
sas_normal	batch	no limit	4
unlimited	batch	no limit	4 (for now)

https://hbs-rcs.github.io/hbsgrid-docs/commandline/#job-queues-limits

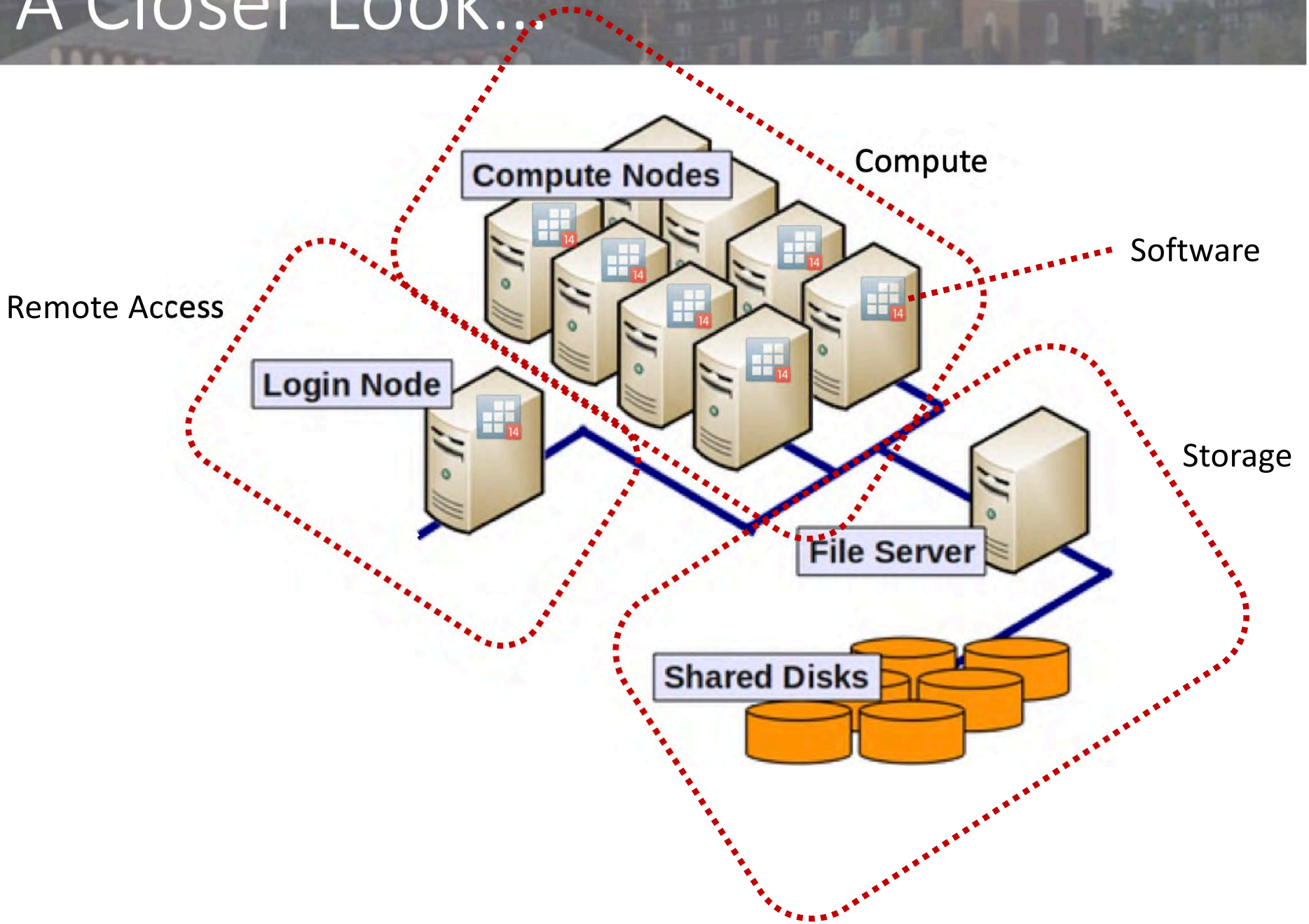
We also have a *mini* and *micro* queue!


Integrated technologies of login,
storage, & compute

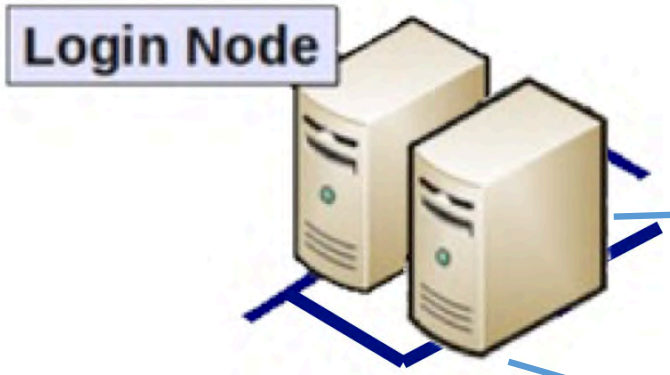
A Closer Look…

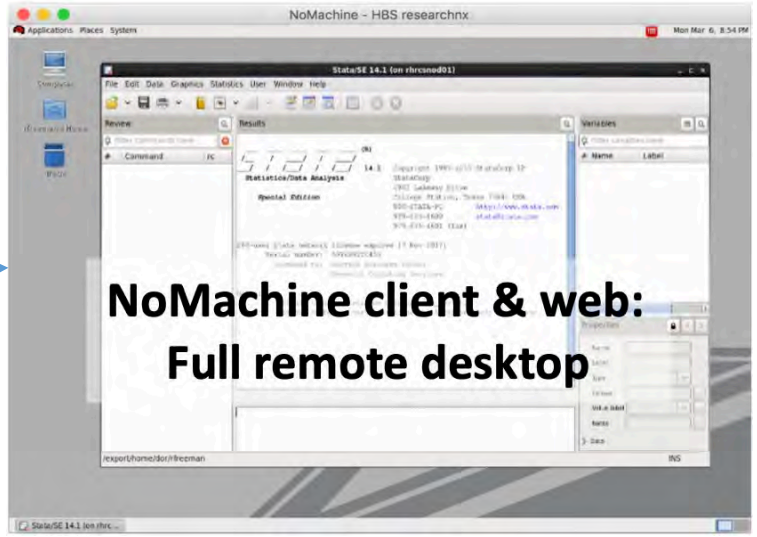


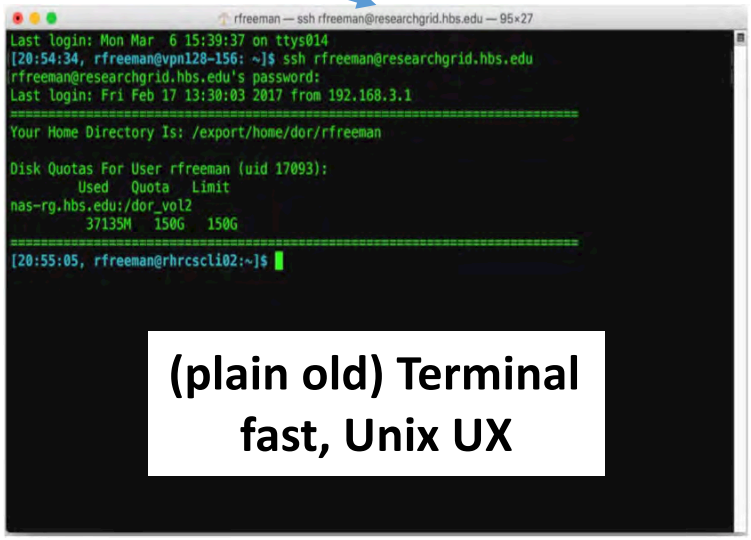
A Closer Look…



A Closer Look…Remote Access



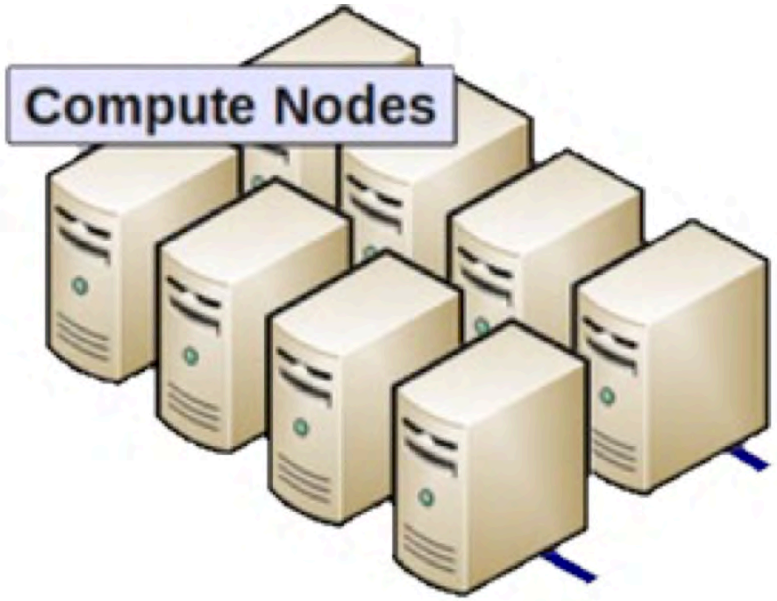




VPN is required off-campus and on HBSGuest WiFi

https://hbs-rcs.github.io/hbsgrid-docs/#quick-start

Finite Limits of the HBSGrid…



CPU/cores:

32 cores * 12 boxes	576 cores
6 special nodes	296
	876 cores total

RAM:

18 nodes	=	256 GB RAM ea
4 nodes	=	1.5 TB RAM ea

Resource limits:

Interactive:	18 cores max over 3 simult. sessions
Batch:	150 cores max over all jobs
No RAM limits	

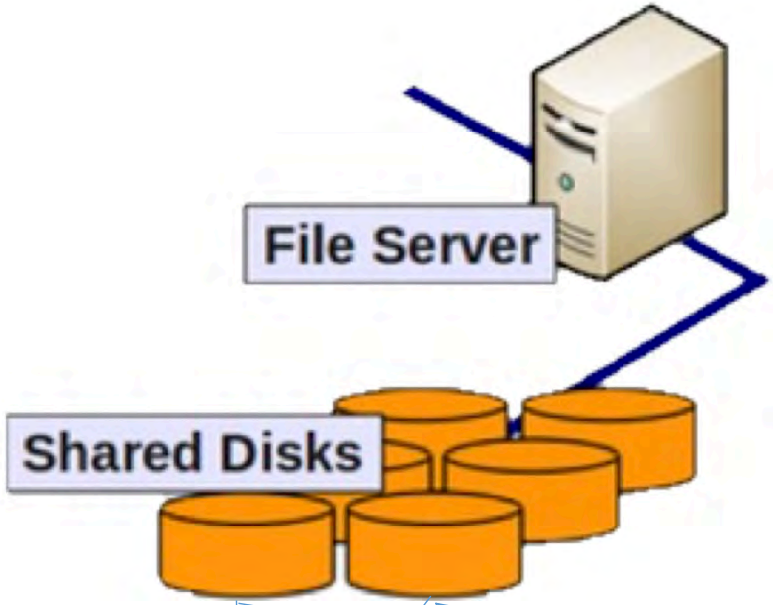
Remember, this is for use by >180 faculty, >120 doctoral students, >60 RAs, & > 100 guests!

Do talk to us if you need a temporary increase of your allocation for any reason…

- Holidays are coming & you want to leave Santa beautiful graphs instead of cookies
- Conference or presentation
- Deadline for faculty or collaborator

https://hbs-rcs.github.io/hbsgrid-docs/commandline/#job-queues-limits
https://hbs-rcs.github.io/hbsgrid-docs/menulaunch/#system-resources-limits

A Closer Look…At Storage (Internal)



Use for solo & project work: snapshots every hour, and quotas enforced.

/export/home/dor/jharvard/

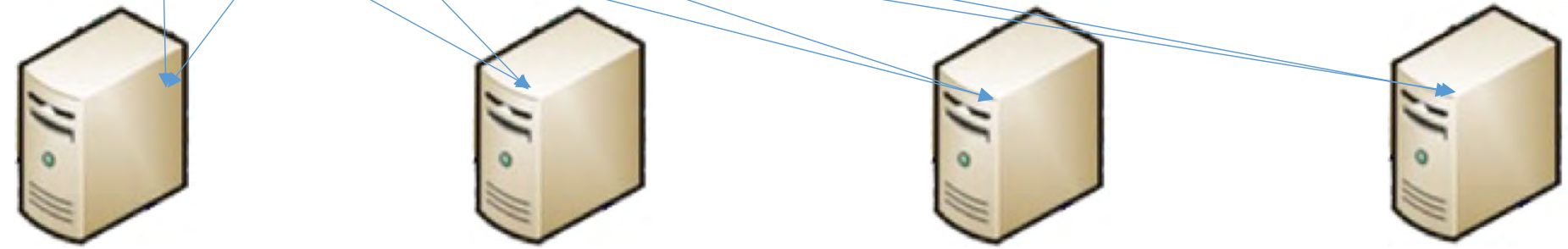
/export/projects/jharvard_taxes/

Use for temp work: No backup, not part of quota, & mind folder permissions / files older than 60d

/export/scratch

Use "Create Scratch Folder" script to help with setup

* Level4 data area is more restricted

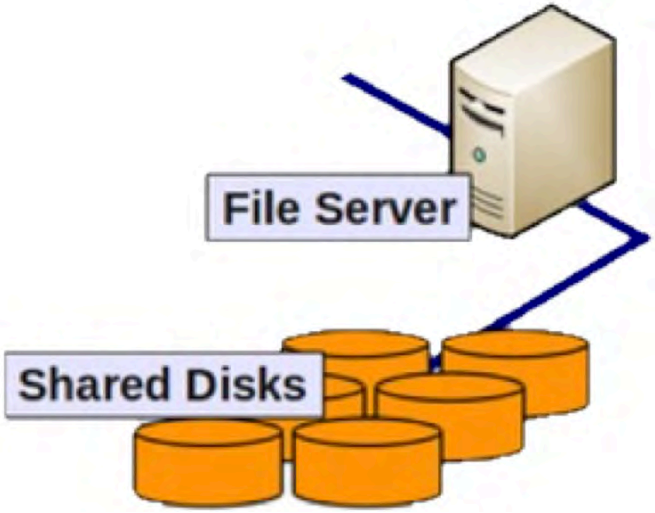


rhrcscli1 rhrcsnod1 rhrcsnod2 rhrcsnod3

Internally, the storage filesystems looks the same* from every node, incl. login nodes!

https://hbs-rcs.github.io/hbsgrid-docs/syncfiles/#hbs-grid-storage-overview

A Closer Look…At Storage (External)



From outside the cluster…Two methods to access storage

1) (Mac) Mounting volumes in Finder / (PC) Mapping drives in File Explorer – NOT RECOMMENDED

- Uses SMB/CIFS protocol
- Allows one to use files on research storage from Desktop/laptop
- Can be slow due to protocol and GUI overhead; & cause permissions issues
- May not be permitted for L4 sensitive data

```
smb://research.hbs.edu/projects/my_big_project
\\research.hbs.edu\projects\my_big_project
```

2) Using FileZilla, rsync, scp, SecureFX -- RECOMMENDED

- Programs using SSH or SFTP protocols
- Method to transfer uni- or bi-directionally (not use in-place)
- Connection goes though login node and is usually quite fast

```
sftp://hbsgrid.hbs.edu:/export/projects/my_big_project
rsync –av ~/my_big_project/*.txt hbsgrid.hbs.edu:/export/projects/my_big_project/
```

There are other methods available (Globus, RClone, NoMachine attached devices, etc.), and **We strongly discourage using wireless for large data transfers or large file, in-place work**

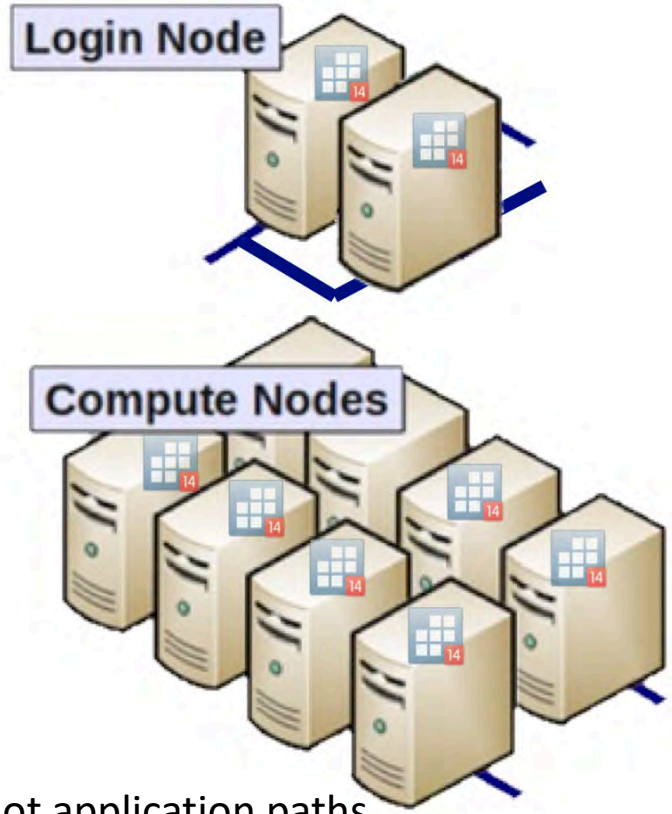
https://hbs-rcs.github.io/hbsgrid-docs/syncfiles/#transfer-data-fromto-local-storage
https://hbs-rcs.github.io/hbsgrid-docs/syncfiles/#transfer-data-fromto-cloud-storage

A Closer Look…at Software

Quite a large number of software titles!

Login nodes: OK for some low-memory / low-CPU programs:

- Text editor
- Emacs
- File browser
- GitKraken



All compute-/ RAM-intensive work on compute nodes!

- MATLAB
- R
- Python
- SAS
- Mathematica
- Stata
- Stat Transfer

If not using wrapper scripts (see next), use software modules, not application paths.

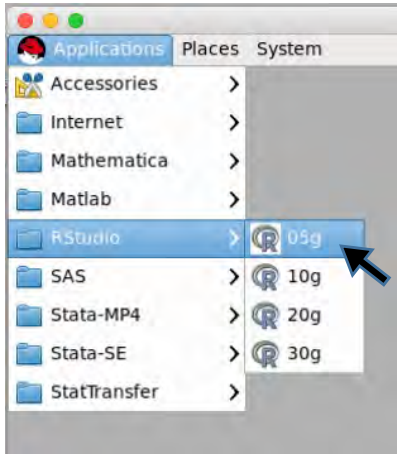
- Can install your own software
- Software modules now implemented
  - Running Jobs, brief details; Software section, reference info
  - Don't need to know program paths
  - Can use software modules with (before) wrapper scripts

https://hbs-rcs.github.io/hbsgrid-docs/environments/#environment-versions
https://hbs-rcs.github.io/hbsgrid-docs/environments/#select-terminal-environment
https://www.hbs.edu/research-computing-services/resources/compute-cluster/selectingsoftware.aspx#modules
https://hbs-rcs.github.io/hbsgrid-docs/software/#installing-compatible-software-on-the-hbsgrid

A Closer Look…at Software

Wrapper scripts: RAM and (mostly single) core presets on login nodes

- Command-line wrappers
- Drop down menus




```
# both are same, defaults to 5 GB RAM
Rstudio-5g
Rstudio

# -5g -10g -20g are the options
Rstudio-10g

# same; if not specified, -n = 1 number of cores
matlab    -n2
matlab-5g –n2

# runs as batch: input & output files are named
matlab-5g -n2 myscript.m output.txt
```

Items deprecated with Grid-3.0

Same result by writing custom LSF commands

queue cores RAM program

```
bsub -q long_int -n 1 -Is -R "rusage[mem=5120]" -M 5120 -hl /usr/local/app/conda-R/conda-R-5.1/bin/rstudio
bsub -q long_int -n 1 -Is -app R-5g -hl rstudio          # relying on symlinks on compute nodes
bsub -q long_int -n 1 -Is -M 5120 -hl rstudio            # via symlinks or w/ prior module load
bsub -q long_int -Is rstudio                  # example of easiest submit == assume 1 core, 5 – 8 GB RAM
```

Default wrapper scripts can be avoided by using direct LSF commands and options.
See our website http://grid.rcs.hbs.org/custom-submission-scripts
FAQ on more RAM for sessions: https://grid.rcs.hbs.org/faq/compute-i-need-interactive-session-more-ram-how-do-i-do

A Closer Look…at Software Modules

Software Modules are now the new way to access software titles in your jobs

- Currently opt-in (through September)
- Flexible usage of application versions
- Does not require hard-coding paths
- Allows flexibility of RCS & HBS IT to roll out improvements

Wrapper scripts are evolving to make use of software modules

- No longer hard-coded to specific versions
- Will allow more flexibility if modules used before wrappers (command-line)
- Will also evolve to handle GUI user's application version preferences

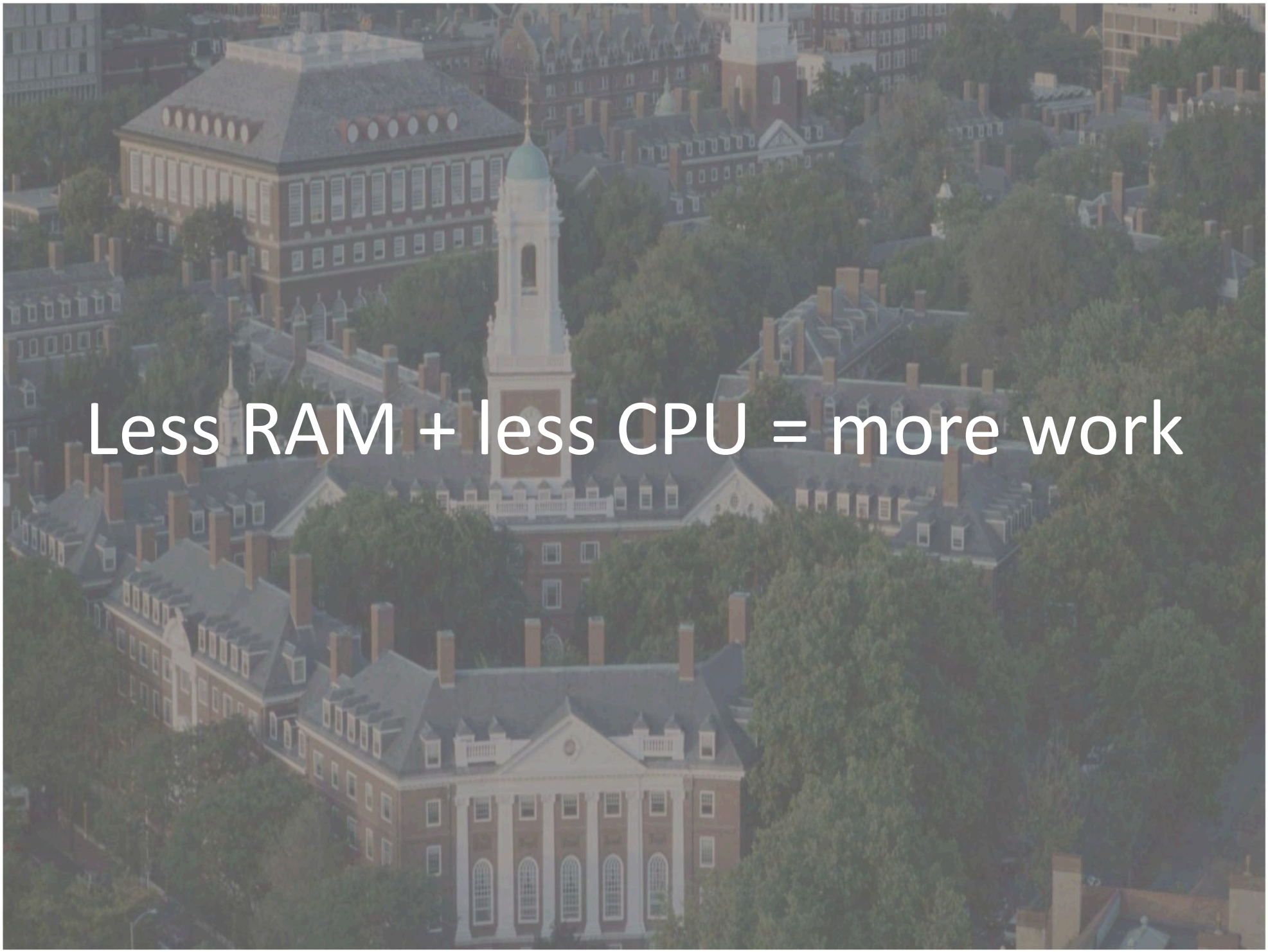
A few commands to remember

`module avail`	what titles/versions are available
`module load python`	loads default python version
`module load python/3.7`	loads specific python version

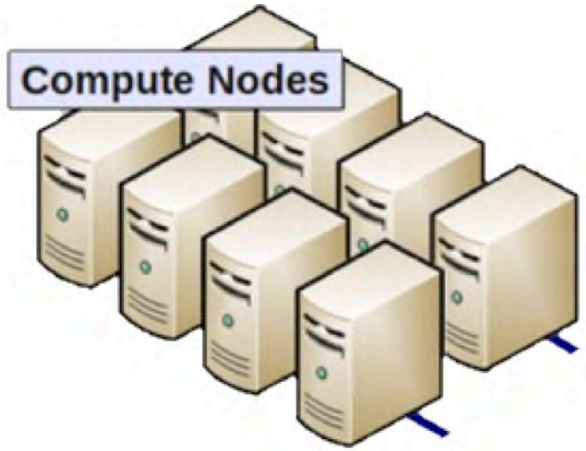
Please see our website for more information…

- Brief information for Running Jobs: Accessing Software via Modules
- Detailed info: Software Modules (in Software)

https://hbs-rcs.github.io/hbsgrid-docs/environments/#environment-versions
https://hbs-rcs.github.io/hbsgrid-docs/environments/#select-terminal-environment
https://www.hbs.edu/research-computing-services/resources/compute-cluster/selectingsoftware.aspx#modules


Less RAM + less CPU = more work

What are you working with?



CPU/cores:
32 cores * 10 boxes = 336 cores
+ 288 special nodes = 624 cores total

RAM:
11 nodes = 256 GB RAM ea
4 nodes = 1.5 TB RAM ea

resource limits: 150 cores max, batch
18 cores max, interactive
& 3 concurrent sessions
no RAM limits

Going beyond your resource limit causes PEND problems:

```
[jharvard@rhrcscli1:~]$ bjobs -w
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
144795  jharvard RUN   interactive rhrcscli01  4*rhrcsnod08 /usr/local/bin/jobstarter_xstata-mp4.pl xstata-mp4 Mar  7 10:40
144796  jharvard RUN   interactive rhrcscli01  4*rhrcsnod07 /usr/local/bin/jobstarter_xstata-mp4.pl xstata-mp4 Mar  7 10:44
144797  jharvard RUN   interactive rhrcscli01  4*rhrcsnod07 /usr/local/bin/jobstarter_xstata-mp4.pl xstata-mp4 Mar  7 10:44
144798  jharvard PEND  interactive rhrcscli01     -        /usr/local/bin/jobstarter_xstata-se.pl xstata-se Mar  7 10:44
[jharvard@rhrcscli1:~]$
```

Is it really worth it? Do you really need this?
Do you need all that RAM?
Do you need all that CPU?

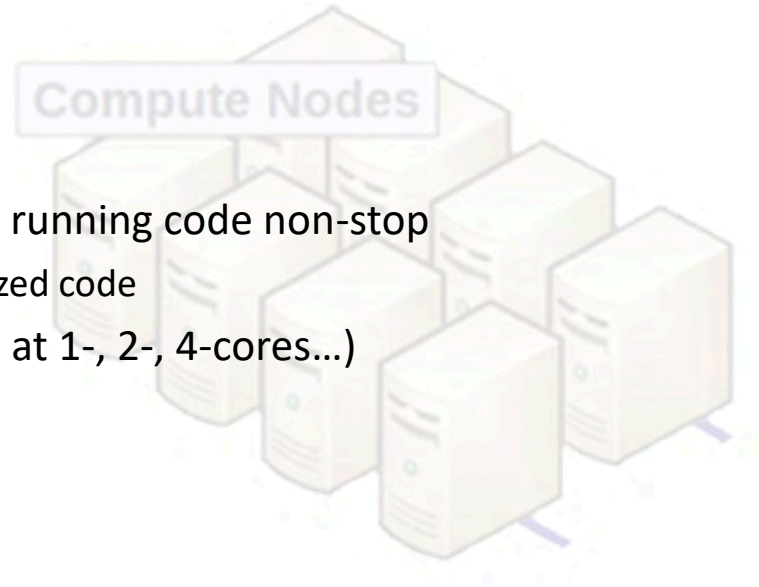
Methods for Selecting CPU & RAM

Starting points:

- Summarized on our Guidelines for Choosing Resources web page
- Likely you can make a very good guestimate by looking into a few things
- **Wild guesses are discouraged, and More is not better! This prevents users from getting onto the system**

CPU:

- Be thoughtful on how many cores to use– others are in line behind you to use the resources
- Use fewer cores (e.g. 1) for interactive work, especially if you are …
  - Writing code
  - Doing initial testing / debugging
  - Exploring data / visualizations
  - Or generally noodling around
- Use more cores if your application or code is parallelized AND running code non-stop
  - **NB! R and Python can only use 1 core** unless you write parallelized code
- If parallelizing code, do basic scaling tests (compare run times at 1-, 2-, 4-cores…)



(NoMachine/Grid-3.0) Applications > Systems > HBSGrid Job Monitor
https://hbs-rcs.github.io/hbsgrid-docs/trouble/#troubleshooting-jobs-and-resources

Methods for Selecting CPU & RAM

RAM

- Be thoughtful on your RAM ask! **RAM wastage is our biggest problem in blocking research**
- If it works on your local machine, ask for that amount of RAM
- If importing text (CSV) files, you might need up to ~5X the size of the text file when compressed
    - **We highly discourage ongoing reading and writing CSV/text files as the main way to work with data.** Not only is it time consuming, but significantly more RAM needed for this pattern than with binary files. We recommend StatTransfer as an efficient way to convert text data into binary format.
- If loading in binary data files, ask for the size of the file ≤ 1.2 - 1.5x (extra RAM for new work)
    - Go on the lower side if your RAM ask is > 50 GB RAM
- Review from past jobs your MAX MEM usage (as shown in a few slides), & select best memory footprint
- Give yourself 20% extra RAM for wiggle room!
- Use LSF job submission options to match more closely memory usage

(NoMachine/Grid-3.0) Applications > Systems > HBSGrid Job Monitor
https://hbs-rcs.github.io/hbsgrid-docs/trouble/#troubleshooting-jobs-and-resources

Methods for Selecting RAM

CPU & RAM

- If parts of your program have different requirements, break it up into smaller pieces, each with its own CPU/RAM need
- For large RAM asks (> 50 - 100 GB), reconsider your approach by:
  - Loading less data (fewer rows or columns) into memory
  - Optimizing the data types
  - Chunking the data (automatically) as its loaded
  - Using a different data framework that is more memory efficient / parallelized

```
# One Unix command to rule them all...
#
[jharvard@rhrcscli1:~]$ bjobs -l    | grep -E "Application|IDLE|MAX"

Job <144795>, User <jharvard>, Project <XSTATA>, Application <stata-mp4-20g>, S
IDLE_FACTOR(cputime/runtime):   0.01
MAX MEM: 56 Mbytes;  AVG MEM: 49 Mbytes

# Or for previously run job
#
[jharvard@rhrcscli1:~]$ bhist -a -l –n 5 | grep -E "Application|IDLE|MAX"

Job <144695>, User <jharvard>, Project <XSTATA>, Application <stata-mp4-10g>, S
IDLE_FACTOR(cputime/runtime):   0.05
MAX MEM: 159 Mbytes;  AVG MEM: 143 Mbytes
```

https://www.hbs.edu/research-computing-services/resources/compute-cluster/running-jobs/guidelines-for-choosing-resources.aspx

Methods for Selecting RAM

If you don't really know where to start… Each language has commands that will give you the memory usage of your data while loaded (in memory):

Stata:
`. memory`

`grand total` indicates *used* and *allocated* (!)

https://www.stata.com/manuals14/dmemory.pdf

Python:
guppy module for total program and object information:

```
from guppy import hpy
h = hpy()
print h.heap()
Total size = 19909080 bytes.
```

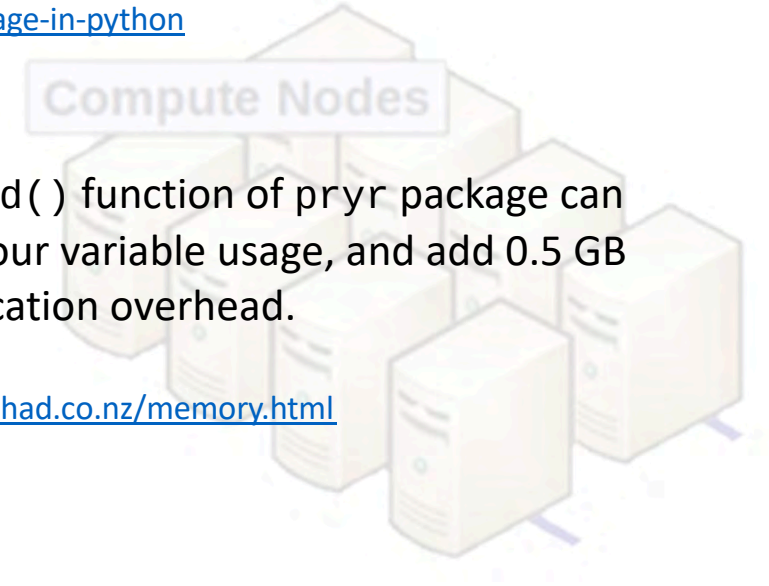
https://www.pluralsight.com/blog/tutorials/how-to-profile-memory-usage-in-python

MATLAB:
`memory` function only available on Windows. Others, use `monitor_memory_whos.m` function to determine variable usage, and add 0.5 GB for application overhead.

https://www.mathworks.com/matlabcentral/answers/97560-how-can-i-monitor-how-much-memory-matlab-is-using

R:
`mem_used()` function of `pryr` package can inform your variable usage, and add 0.5 GB for application overhead.

http://adv-r.had.co.nz/memory.html



Looking at Job Information: Info and Troubleshooting

A Closer Look…

All job info and management must be done through the terminal. Basic commands:

- Get current job information

```
bjobs                                   # PEND, RUN, & SUSP
bjobs -l JOBID                          # long format
```

- Get historic information from past week or beyond

```
bhist                                   # anything recent
bhist -l JOBID                          # long format
bhist -a –n5 –l -S 2017/09/01,          # your jobs submitted since 9/1/17
```

- Kill a job or all jobs

```
bkill JOBID                             # one job
bkill 0                                 # all jobs
```

- How busy is the cluster?

```
bjobs -u all                            # see all users
bqueues && bhosts && lsload             # show cluster load
```

- Current or past RAM usage

```
bjobs -l    | grep -E "Application|IDLE|MAX"     # for current jobs
bhist -a -l | grep -E "Application|IDLE|MAX"     # for current & past jobs
```

Job state and error information is not too complex

Also ”HBSGrid Job Information” menu item in Grid3

Nota bene! Job reports via email are not the answer to your questions!

https://hbs-rcs.github.io/hbsgrid-docs/trouble/#troubleshooting-jobs-and-resources

Details for Current Jobs…

```
[16:57:27, rfreeman@rhrcscli01:~]$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
144767  rfreema RUN   interactiv rhrcscli01  rhrcsnod02  *n/rstudio Mar  6 16:44

[16:57:30, rfreeman@rhrcscli01:~]$ bjobs -l 144767

Job <144767>, User <rfreeman>, Project <R>, Application <R-5g>, Status <RUN>, Q
                     ueue <interactive>, Interactive mode, Command </usr/local/
                     apps/R/rstudio/v0.98.493/bin/rstudio>
Mon Mar  6 16:44:29: Submitted from host <rhrcscli01>, CWD <$HOME>;
Mon Mar  6 16:44:29: Started on <rhrcsnod02>;
Mon Mar  6 16:57:34: Resource usage collected.
                     The CPU time used is 37 seconds.
                     IDLE_FACTOR(cputime/runtime):   0.04
                     MEM: 169 Mbytes;  SWAP: 2.2 Gbytes;  NTHREAD: 13
                     PGID: 12778;  PIDs: 12778
                     PGID: 12779;  PIDs: 12779 12781 12793


 MEMORY USAGE:
 MAX MEM: 169 Mbytes;  AVG MEM: 159 Mbytes

 SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

 RESOURCE REQUIREMENT DETAILS:
 Combined: select[type == local] order[r15s:pg] rusage[mem=5120.00]
 Effective: select[type == local] order[r15s:pg] rusage[mem=5120.00]
```

Details for Past Jobs (Mostly)…

```
[17:16:34, rfreeman@rhrcscli01:~]$ bhist -a -S 2021/12/01,
Summary of time in seconds spent in various states:
JOBID   USER    JOB_NAME  PEND    PSUSP   RUN     USUSP   SSUSP   UNKWN   TOTAL
155256  rfreema *in/bash  0       0       1121    0       0       0       1121
155948  rfreema *in/bash  0       0       21358   0       0       0       21358
157290  rfreema MATLAB    0       0       376     0       0       0       376
157396  rfreema *in/bash  0       0       30743   0       0       0       30743
157400  rfreema MATLAB    0       0       595     0       0       0       595
157520  rfreema MATLAB    0       0       1178    0       0       0       1178
157522  rfreema *rstudio  0       0       625     0       0       0       625
157523  rfreema *rstudio  0       0       21      0       0       0       21
157524  rfreema *rstudio  2       0       32      0       0       0       34

[10:58:03, rfreeman@rhrcscli02:~]$ bhist -l 157400

Job <157400>, Job Name <MATLAB>, User <rfreeman>, Project <MATLAB>, Application
                     <matlab-5g>, Interactive pseudo-terminal mode, Command <m
                     atlab_2017a -r "LASTN = maxNumCompThreads(4);">
Wed Aug 23 12:08:23: Submitted from host <rhrcscli02>, to Queue <interactive>,
                     CWD <$HOME>, 4 Processors Requested;
Wed Aug 23 12:08:23: Dispatched to 4 Hosts/Processors <4*rhrcsnod01>, Effective
                     RES_REQ <select[type == local] order[r15s:pg] rusage[mem=
                     5120.00] >;
Wed Aug 23 12:08:23: Starting (Pid 28541);
Wed Aug 23 12:18:18: Done successfully. The CPU time used is 142.2 seconds;
Wed Aug 23 12:18:18: Post job process done successfully;

MEMORY USAGE:
MAX MEM: 634 Mbytes;  AVG MEM: 537 Mbytes

Summary of time in seconds spent in various st
  PEND     PSUSP    RUN      USUSP    SSUSP
  0        0        595      0        0
```

Useful for scaling tests!

```
Efficiency = CPU time / (RUN * # Processors Requested
           = 142.2    / (595 * 4 )
      0.06 =
-->      6%
```

Details for Past Jobs (Mostly)…

```
[10:58:27, rfreeman@rhrcscli02:~]$ bhist -l 157396

Job <157396>, User <rfreeman>, Project <default>, Interactive pseudo-terminal s
                     hell mode, Command </bin/bash>
Wed Aug 23 11:44:51: Submitted from host <rhrcscli01>, to Queue <interactive>,
                     CWD <$HOME>, Requested Resources <rusage[mem=1000]>, Speci
                     fied Hosts <rhrcsnod07>;


 RUNLIMIT
 1440.0 min of rhrcscli01
Wed Aug 23 11:44:51: Dispatched to <rhrcsnod07>, Effective RES_REQ <select[type
                      == any] order[r15s:pg] rusage[mem=1000.00] >;
Wed Aug 23 11:44:51: Starting (Pid 21766);
Wed Aug 23 20:17:14: Exited by signal 9. The CPU time used is 191.2 seconds;
Wed Aug 23 20:17:14: Completed <exit>; TERM_EXTERNAL_SIGNAL: job killed by a si
                     gnal external to LSF;

MEMORY USAGE:
MAX MEM: 4 Mbytes;  AVG MEM: 3 Mbytes

Summary of time in seconds spent in various s
  PEND     PSUSP    RUN      USUSP    SSUSP
  0        0        30743    0        0
```

```
Efficiency = CPU time / (RUN * # Processors Requested
           = 191.2    / (30743 * 1 )
     0.01 ~=
-->    ~1%
```

Resources consumed by low efficiency "work" account for > 70% of the hbsgrid usage. If not actively using the resources, exit programs to release those resources for others to use!

Basic Troubleshooting

Before seeking help, take some basic steps to ascertain what is going on with your job:

- If running interactive jobs from NoMachine/Gnome or wrapper scripts, there are no log files, so `bjobs` & `bhist` commands must be used
- Use `bjobs -l` to query details from LSF. Look towards bottom for errors:
  - Is your job waiting for space (`Resources`)?
  - Will your job ever run (`Dependency`)?
  - Is there an error code or message?
- Otherwise, check your log files:
  - Was a *.log file generated? What does it say?
  - If using custom submission commands, you did specify both `-o` and `-e` output files, yes?
  - bsub -q gpu -gpu - -R ”rusage[mem=5000]” -n 3 -M 6000 –hl -o myfile_%J.out -e myfile_%J.err \
  - -B -N python myscript.py 1 3
  - Message about `Pre-emption`, `Timeout`, or `Failure`?
  - The last error in the log is usually not the problem. The first one is!
- Did you request e-mail messages for your jobs with `-u youremail@hbs.edu`?
- If you got a job report through email, did you understand the error?
- Is your job submission script formatted properly?
- Are you using the correct software? Modules? Possible software/library conflicts?

https://www.hbs.edu/research-computing-services/resources/compute-cluster/running-jobs/monitoring.aspx

Getting Help

RCS Website & Documentation -- only authoritative source

https://www.hbs.edu/research-computing-services/resources/
https://rcs-hbs.github.io/grid_docs/

Submit a help request

Form at https://www.hbs.edu/research-computing-services/help/online-requests.aspx
Email to research@hbs.edu

Best way to help us to help you? Give us...

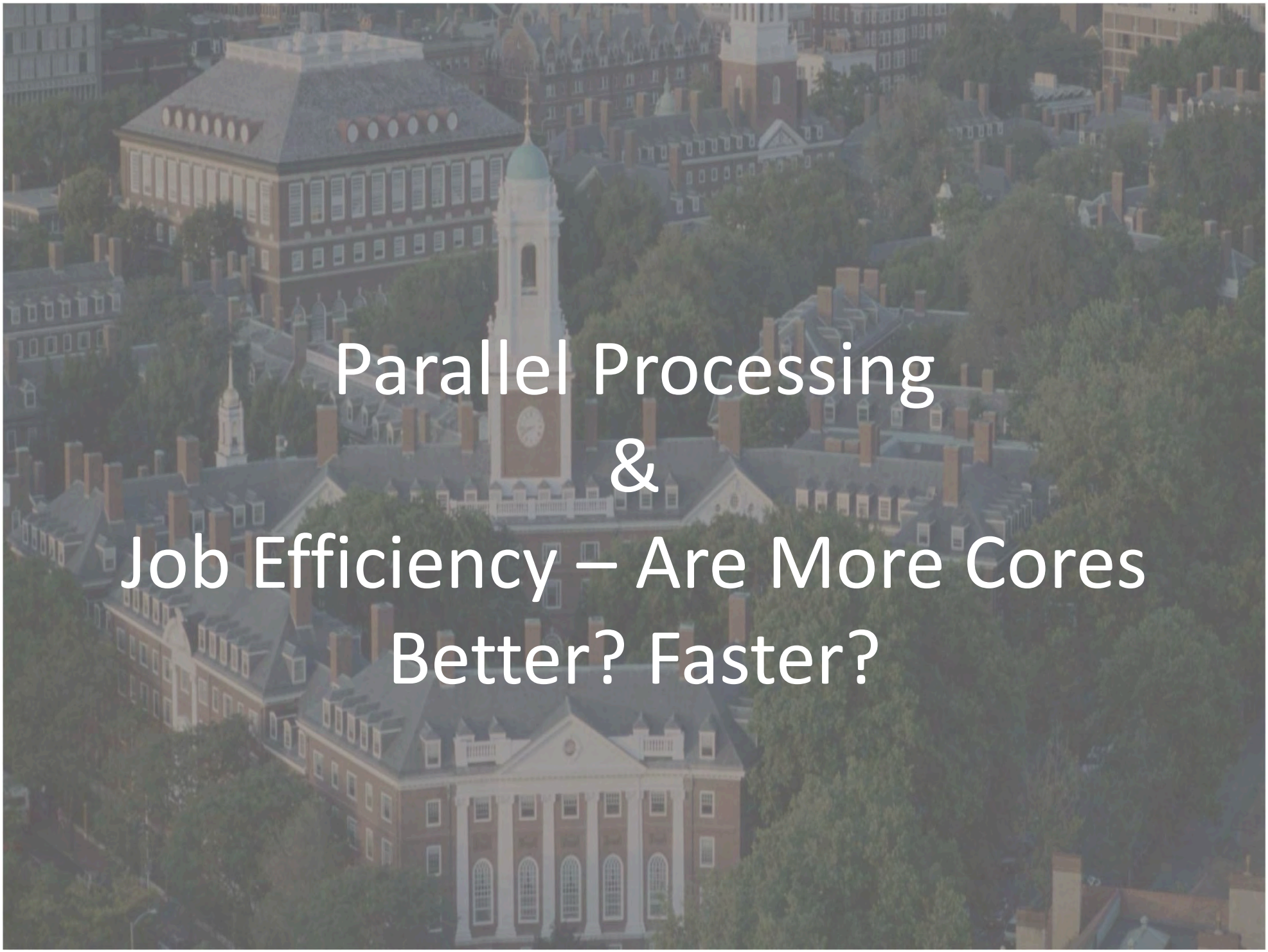
- Description of problem
- Additional info (login/batch? queue? JobIDs? How connected?)
- **Error messages & screen shots**
- Steps to Reproduce (1., 2., 3...) **including command(s) used**
- Actual results
- Expected results

Examples:

√ Hellooo RCS! Hoping you can help. My RStudio interactive session in NoMachine crashed, and I'm not entirely sure why. I tried the `bjobs` command in the unix window but it said "`no jobs found`". bhist gave me a few, as I've been multitasking. When doing `bhist -l` on one, it said `error -127`. Is this the one? What does that mean?

X It's been a long day, folks. Stata isn't working. Help!

https://www.hbs.edu/research-computing-services/help/


Parallel Processing
&
Job Efficiency – Are More Cores
Better? Faster?

Parallel Processing…

We've mostly been discussing **explicit parallelization**:

User-written code that leverages more than 1 CPU/core

There's also **implicit parallelization**:

Code (usually commercial) that has been written to use multiple cores automatically

And we also have "**pleasantly parallelizing**":

Breaking a task into smaller independent pieces, which are submitted as jobs

1 task == 1 job

If tasks are almost identical, one can use job arrays to treat them as one unit

We highly recommend:

1a. Reviewing our Parallel Processing web pages for general overview

1b. Review the code examples there for your language / analysis environment

2a. Review the Parallelization slides for ways to parallelize

2b. Also follow the guidelines there for coding, testing, and executing your work

Nota bene! It is very easy for parallel code to get out of hand, potentially causing problems with other people's jobs! Test and execute cautiously & carefully!

https://hbs-rcs.github.io/hbsgrid-docs/tutorials/scaling-work/

Example: Stata Parallelization

Stata offers a 293-page report on its parallelization efforts. They are pretty impressive. However:

With multiple cores, one might expect to achieve the theoretical upper bound of doubling the speed by doubling the number of cores—2 cores run twice as fast as 1, 4 run twice as fast as 2, and so on. **However, there are three reasons why such perfect scalability cannot be expected:** 1) some calculations have parts that cannot be partitioned into parallel processes; 2) even when there are parts that can be partitioned, determining how to partition them takes computer time; and 3) multicore/multiprocessor systems only duplicate processors and cores, not all the other system resources.

Stata/MP achieved **75% efficiency overall and 85% efficiency among estimation commands**.

Speed is more important for problems that are quantified as large in terms of the size of the dataset or some other aspect of the problem, such as the number of covariates. On large problems, Stata/MP with **2 cores runs half of Stata's commands at least 1.7 times faster than on a single core. With 4 cores, the same commands run at least 2.4 times faster than on a single core**.

https://hbs-rcs.github.io/hbsgrid-docs/tutorials/scaling-work/#introduction_2 #STATA

1 Summary

Stata/MP[1] is the version of Stata that is programmed to take full advantage of multicore and multiprocessor computers. It is exactly like Stata/SE in all ways except that it distributes many of Stata's most computationally demanding tasks across all the cores in your computer and thereby runs faster—much faster.

In a perfect world, software would run 2 times faster on 2 cores, 3 times faster on 3 cores, and so on. Stata/MP achieves about 75% efficiency. It runs 1.7 times faster on 2 cores, 2.4 times faster on 4 cores, and 3.2 times faster on 8 cores (see figure 1). Half the commands run faster than that. The other half run slower than the median speedup, and some of those commands are not sped up at all, either because they are inherently sequential (most time-series commands) or because they have not been parallelized (graphics, `mixed`).

In terms of evaluating average performance improvement, commands that take longer to run—such as estimation commands—are of greater importance. When estimation commands are taken as a group, Stata/MP achieves an even greater efficiency of approximately 85%. Taken at the median, estimation commands run 1.9 times faster on 2 cores, 3.1 times faster on 4 cores, and 4.1 times faster on 8 cores. Stata/MP supports up to 64 cores.

This paper provides a detailed report on the performance of Stata/MP. Command-by-command performance assessments are provided in section 8.

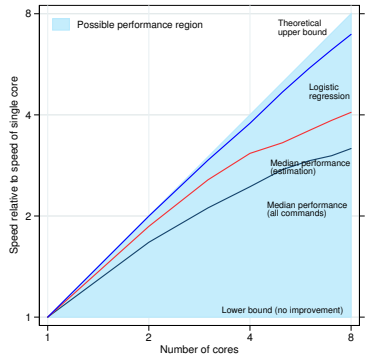


Figure 1. **Performance of Stata/MP.** Speed on multiple cores relative to speed on a single core.

This parallelization benefit is mostly realized in batch mode … most of interactive Stata is waiting for user input (or left idle), as CPU efficiency is typically < 5% - 10%

Explicit Parallelization…*Nota Bene!*

- **By default, R, Python, Perl, and MATLAB* are not multithreaded** … so do not ask for or try to use more than 1 core/CPU!!
- For all these programs, you cannot use the drop-down GUI menus / wrapper scripts.
- Use command-line wrappers or write your own custom LSF submission scripts, and you must set the # of CPUs/core dynamically! DO NOT USE STATIC VALUES!
- For **R**, you can use appropriate routines with `Rparallel` or `Rfutures`
  - Now part of base-R, and includes `Rforeach`, `RdoMC`, or `Rsnow`
- For **Python**, you can use the multiprocessing library (or many others)
- For Perl, there's threads or `Parallel::ForkManager`
- **MATLAB** has `parpool`, and do not set the worker thread count in GUI settings

```
# R example (parallel.R)
library(doMC)
mclapply(seq_len(), run2, mc.cores = Sys.getenv('LSB_DJOB_NUMPROC'))
```

```
bsub –q short –n 4 -app R-5g –hl R CMD BATCH parallel.R     # custom submission command
```

```
# MATLAB example (parallel.m)
hPar = parpool( 'local' , str2num( getenv('LSB_DJOB_NUMPROC') ) );
…
```

```
matlab-5g –n4 parallel.m       # uses command-line wrapper
```

https://hbs-rcs.github.io/hbsgrid-docs/tutorials/scaling-work/

Explicit Parallelization…

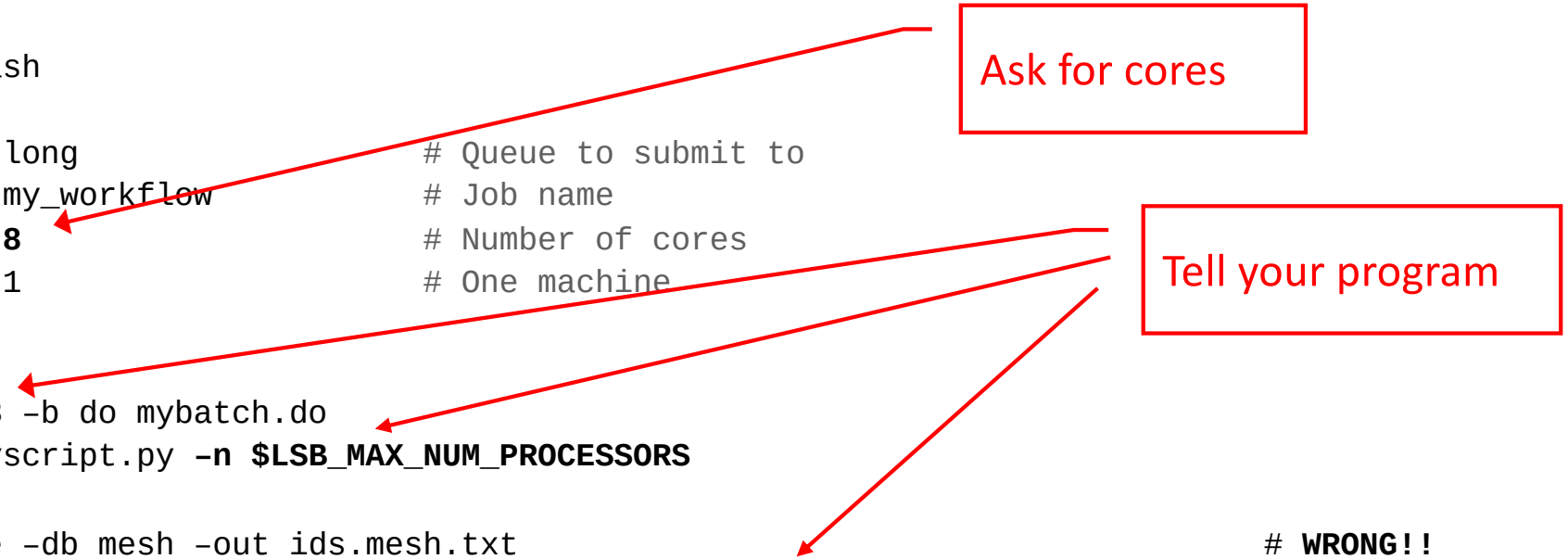
In order to run in parallel, programs (code) must be explicitly programmed to do so. Thus, requesting cores from the scheduler does not automagically parallelize your code.

```
#!/bin/bash
#
#BSUB -q long                   # Queue to submit to
#BSUB -J my_workflow            # Job name
#BSUB -n 8                      # Number of cores
#BSUB -N 1                      # One machine
...

stata-mp8 –b do mybatch.do
python myscript.py –n $LSB_MAX_NUM_PROCESSORS

mycompare –db mesh –out ids.mesh.txt                                  # WRONG!!
mycompare –db mesh –out ids.mesh.txt –num_threads 8                   # WRONG!!
mycompare –db mesh –out ids.mesh.txt –num_threads $LSB_MAX_NUM_PROCESSORS  # YES!!

-----
# now let's submit this parallel job..
bsub < my_job_script.sh                # have it parse the #BSUB directives
# OR
bsub –n 4 –q super_long < my_job_script.sh  # command options override #BSUBs
```




Big Data / Workflows

Big Data & Batch

Attendees usually ask about specific workflow topics. Here are several:

- Big Data Handling in R seminar: Materials from our latest run are at the RCS Large Data in R repo.
- Big Data Handling in Python seminar: materials at https://github.com/fasrc/PythonDataHandling
- For those learning to transition to batch jobs OR scaling work with large #s of jobs, please see our Scaling Up Working/Transitioning GitHub repo. If there is enough interest, I may re-run this seminar, along with a Troubleshooting Jobs one. Let us know!

We also highly recommend tutorials from DataCamp and other reputable online training organizations

Big Data - FYI

We also make the following recommendations

- Code, test, and debug using a small, sampled subset of your data if possible before trying to work on the larger dataset
- Know that any friction points or inefficiencies in your workflow likely will be magnified when working with large datasets.
- One may need to reassess patterns of analysis and workflows
- Research memory usage when using modules and packages that others have written, as not all frameworks are created equal!
    - E.g., when parsing XML, xml.etree.ElementTree is more memory efficient than xml.dom.minidom, which tries to load the entire DOM into memory.
- Walking the filesystem and importing data may no longer be appropriate. Consider using more appropriate, modern formats with helper libraries, such as parquet-formatted files and pyarrow (Python) / arrow (R).
- One might also see significant improvement if using DuckDB, which can use parquet-file metadata more effectively that Python's pandas or R.


Final Points

Important Points

The HBSGrid Compute Cluster and our research computing environment is a **great starting point** for using high-end computing resources (sometimes called advanced cyberinfrastructure). Here are some helpful points:

- The technologies, though similar to laptops and desktops, are used very differently. You will be more successful in your work if you understand better how they work.
- **Submitting jobs with reasonable estimates on # of cores and RAM requirements both will increase your productivity and also will enable more people to do their work.**
- Troubleshooting begins with understanding how the scheduler (LSF) handled your job(s). Use bjobs / bhist (or "HBSGrid Job Monitor) with appropriate options to get these details.
- Not all code is parallelized, and so (in general) asking for more CPU cores will not automagically make things run faster.
- If running parallelized codes, the run pattern is 1) ask the scheduler for the cores, and 2) tell your code via a dynamic variable how many cores have been scheduled for you.
- Check our website(s) for documentation.
- The RCS team is here to help for planning, consultations, and assistance beyond our website docs. It's free, we're friendly, and we like hearing about your research!

Research Computing Services

- Please talk to your peers, and … **We wish you success in your research!**

- https://www.hbs.edu/research-computing-services/
- https://www.hbs.edu/research-computing-services/resources/
- https://www.hbs.edu/research-computing-services/training/

- Follow us @hbs_rcs

- Fill out attendance form & survey at **http://bit.ly/grid_training_survey**