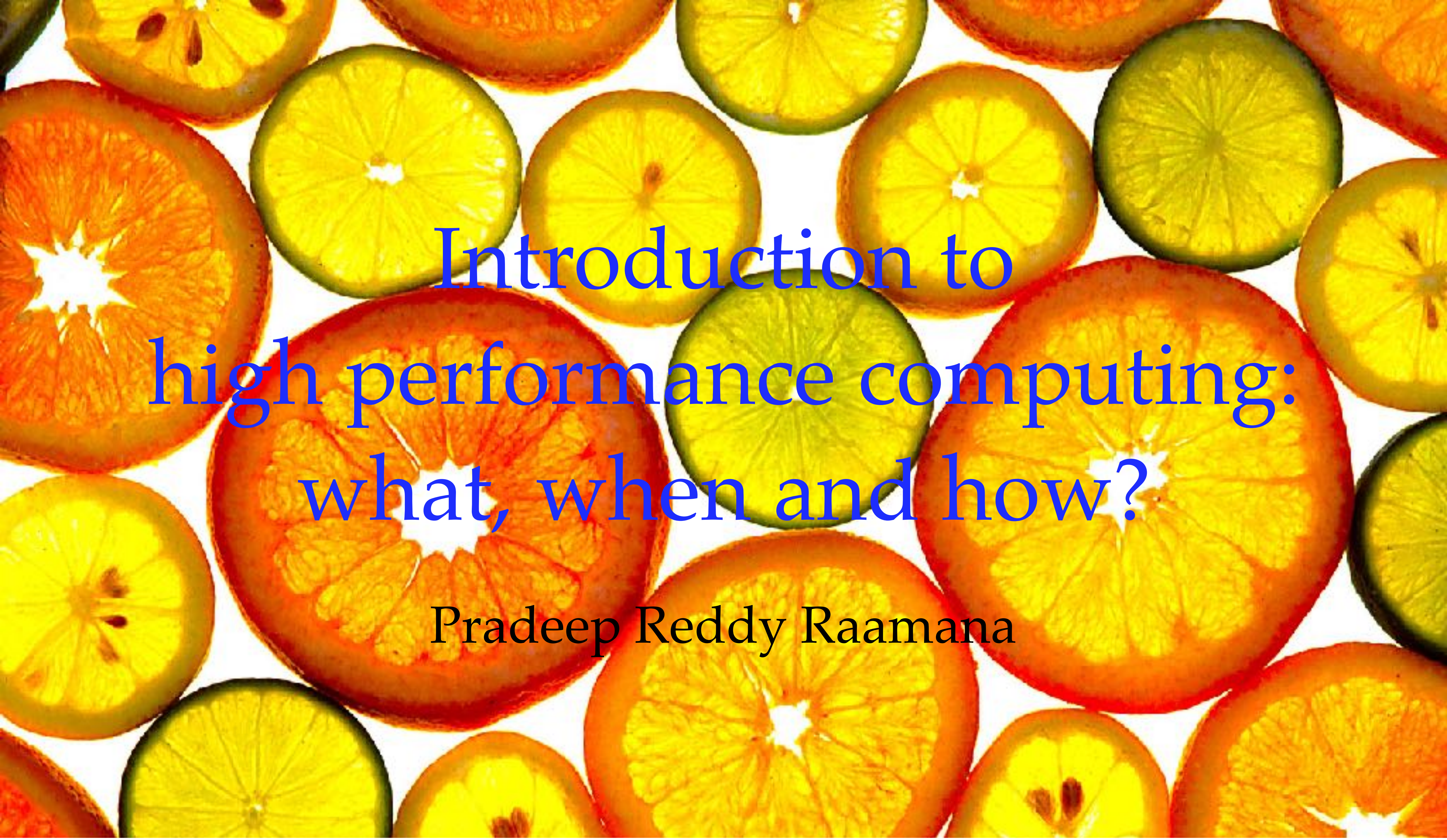


Introduction to high performance computing: what, when and how?

Pradeep Reddy Raamana

100 YEARS
Baycrest

crossinvalidation.com

Follow @raamana_

ONTARIO
BRAIN
INSTITUTE

High Performance Computing

Adam DeConinck
R Systems NA, Inc.

LIFE&ANNUITY
2011 SYMPOSIUM
SEEK KNOWLEDGE. GAIN SOLUTIONS.

r systems
enabling innovation℠

Development of models begins at small scale.

Working on your laptop is convenient, simple.

Actual analysis, however, is *slow.*

Development of models begins at small scale.

Working on your laptop is convenient, simple.

Actual analysis, however, is *slow.*

“Scaling up” typically means a small server or fast multi-core desktop.

Speedup exists, but for very large models, not significant.

Single machines don't scale up forever.

For the largest models, a different approach is required.

High-Performance Computing involves many distinct computer processors working together on the same calculation.

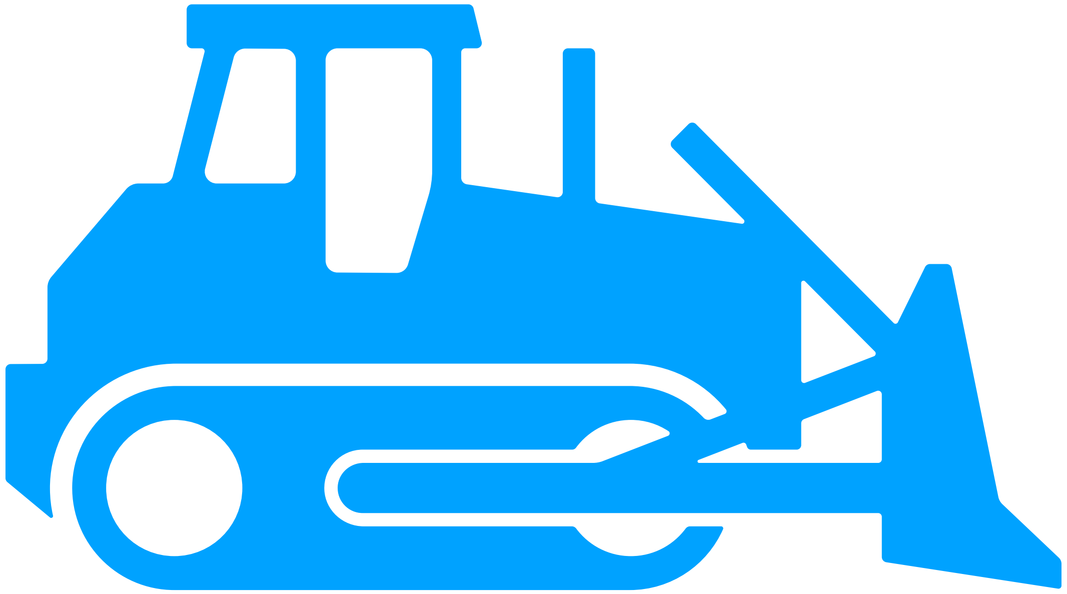
Large problems are divided into smaller parts and distributed among the many computers.

Usually **clusters** of quasi-independent computers which are coordinated by a central scheduler.

What is [not] HPC?

- ✓ Simply a multi-user, shared and smart batch processing system
- ✓ Improves the scale & size of processing significantly
- ✓ With raw power & parallelization
- ✓ Thanks to rapid advances in low cost micro-processors, high-speed networks and optimized software
- ✓ Imagine a big bulldozer!

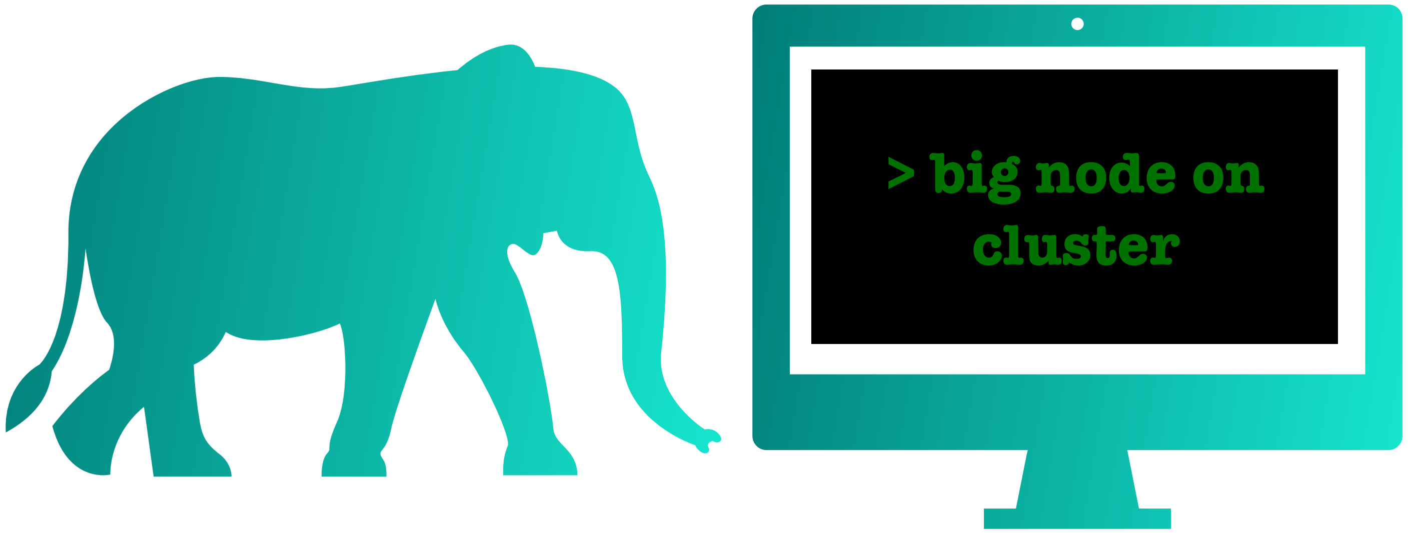
- ✗ does not write your code!
- ✗ does not debug your code!
- ✗ does not speed up your code!
- ✗ does not think for you, or write your paper!



Raamana

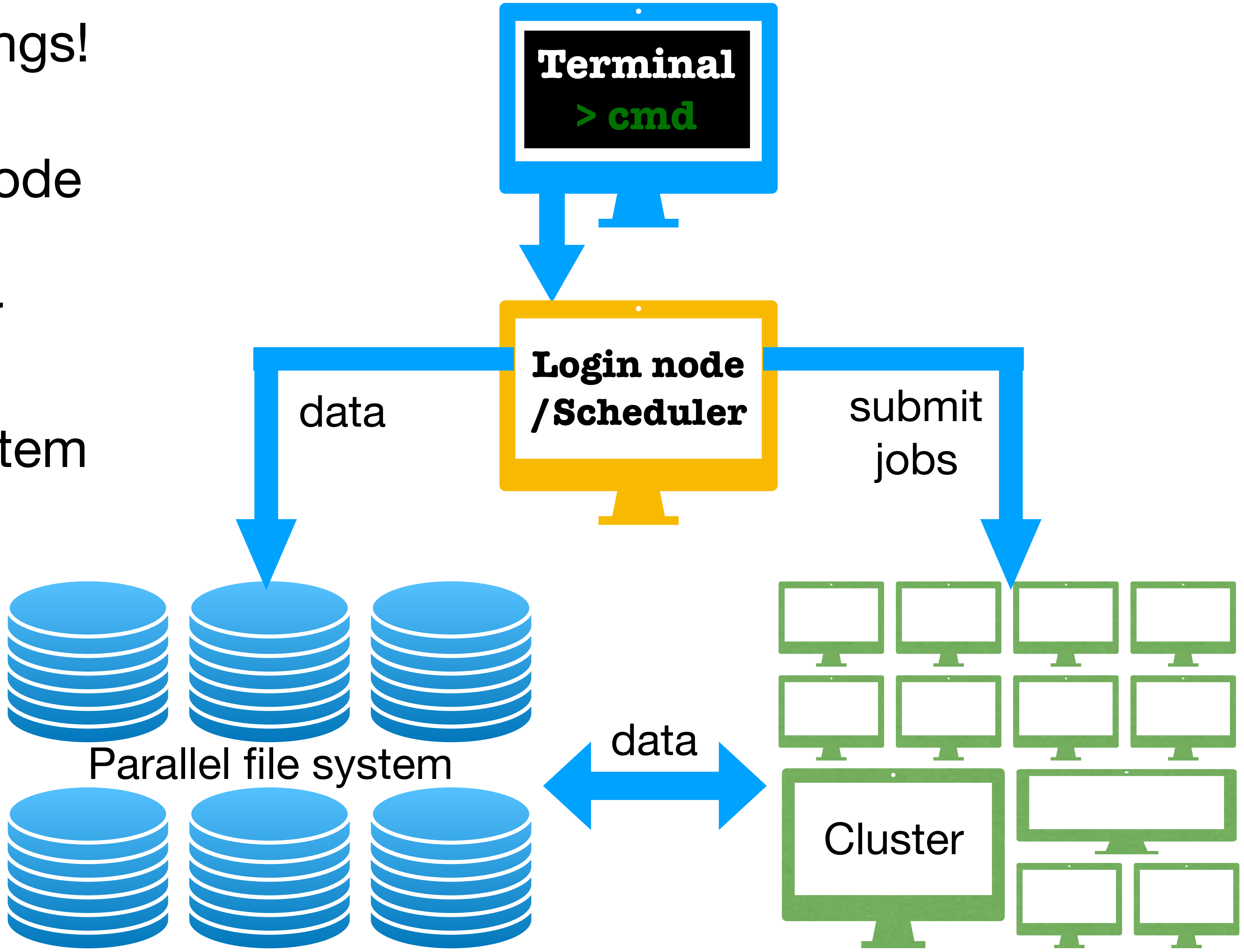
When to **use** HPC?

- When the **task is too big** (memory) to fit on your own desktop computer!
- When you have **many small tasks** with different parameters!
  - same task, many different subjects or conditions etc.
  - same **pattern** of computing, if not same task.
- When your jobs ran too long (months!)
- Need > 1 terabyte of disk space
- High-speed data access - really high!
- No downsides in using it in most cases!!



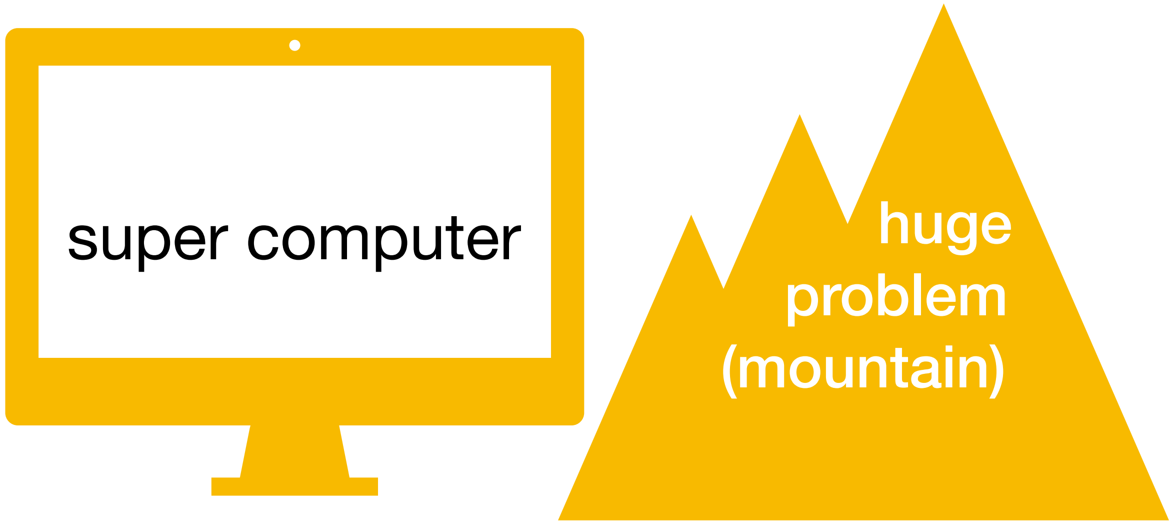
Components of HPC cluster

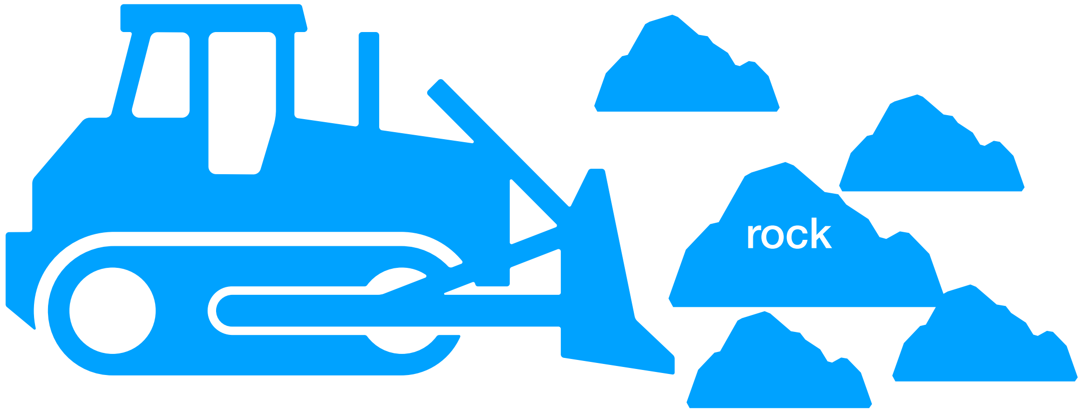
- Just 3 things!
  - Headnode
  - Cluster
  - Filesystem



Is HPC a supercomputer?

- No and Yes
- Supercomputers —> a single very-super-large task
- HPC —> many small tasks
- by “high”, we typically mean “large amount” of performance

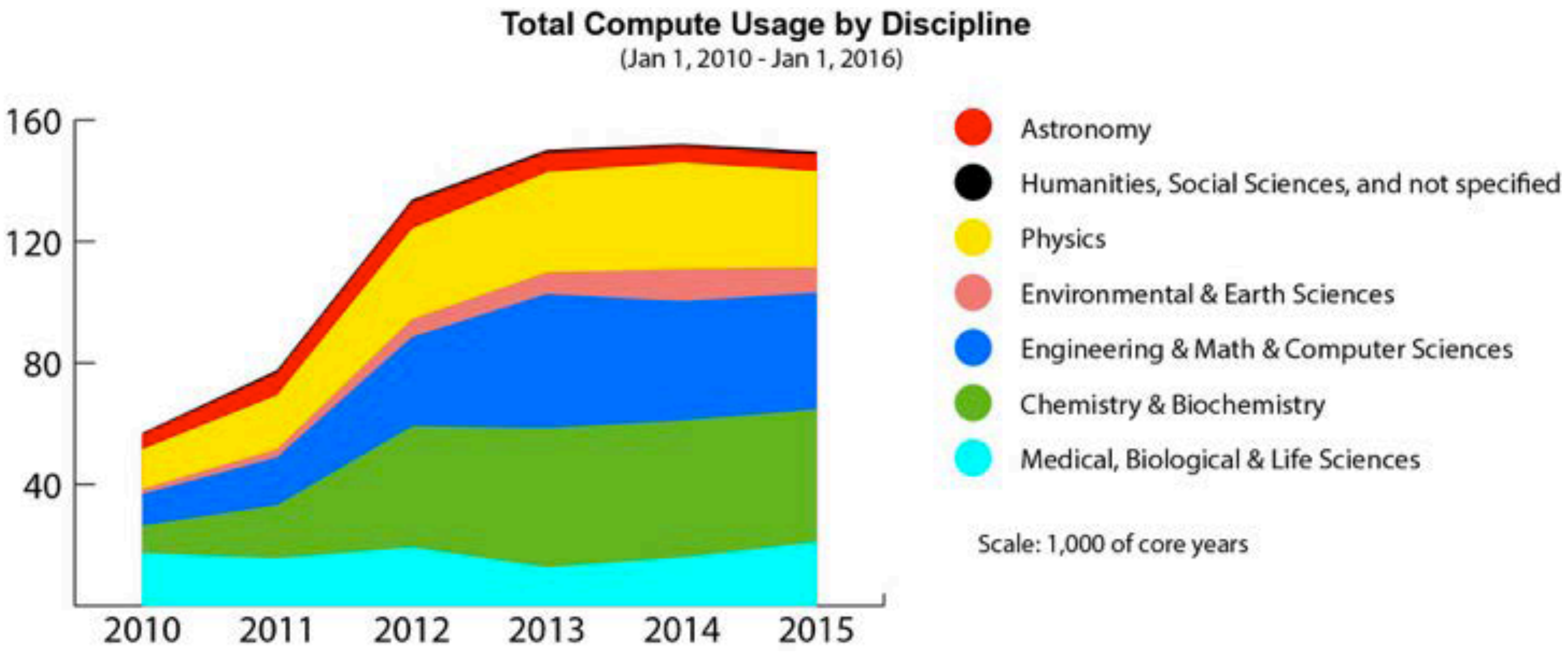




Benefits of HPC cluster

- Cost-effective
  - Much cheaper than a super-computer with the same amount of computing power!
  - When the supercomputer crashes, everything crashes!
  - When a single/few nodes in HPC fail, cluster continues to function.
- Highly scalable
  - Multi-user shared environment: not everyone needs all the computing power all the time.
  - higher utilization: can accommodate variety of workloads (#CPUs, memory etc), at the same time.
  - Can be expanded or shrunk, as needed.

HPC usage is growing



© compute canada, tech briefing, 2016.

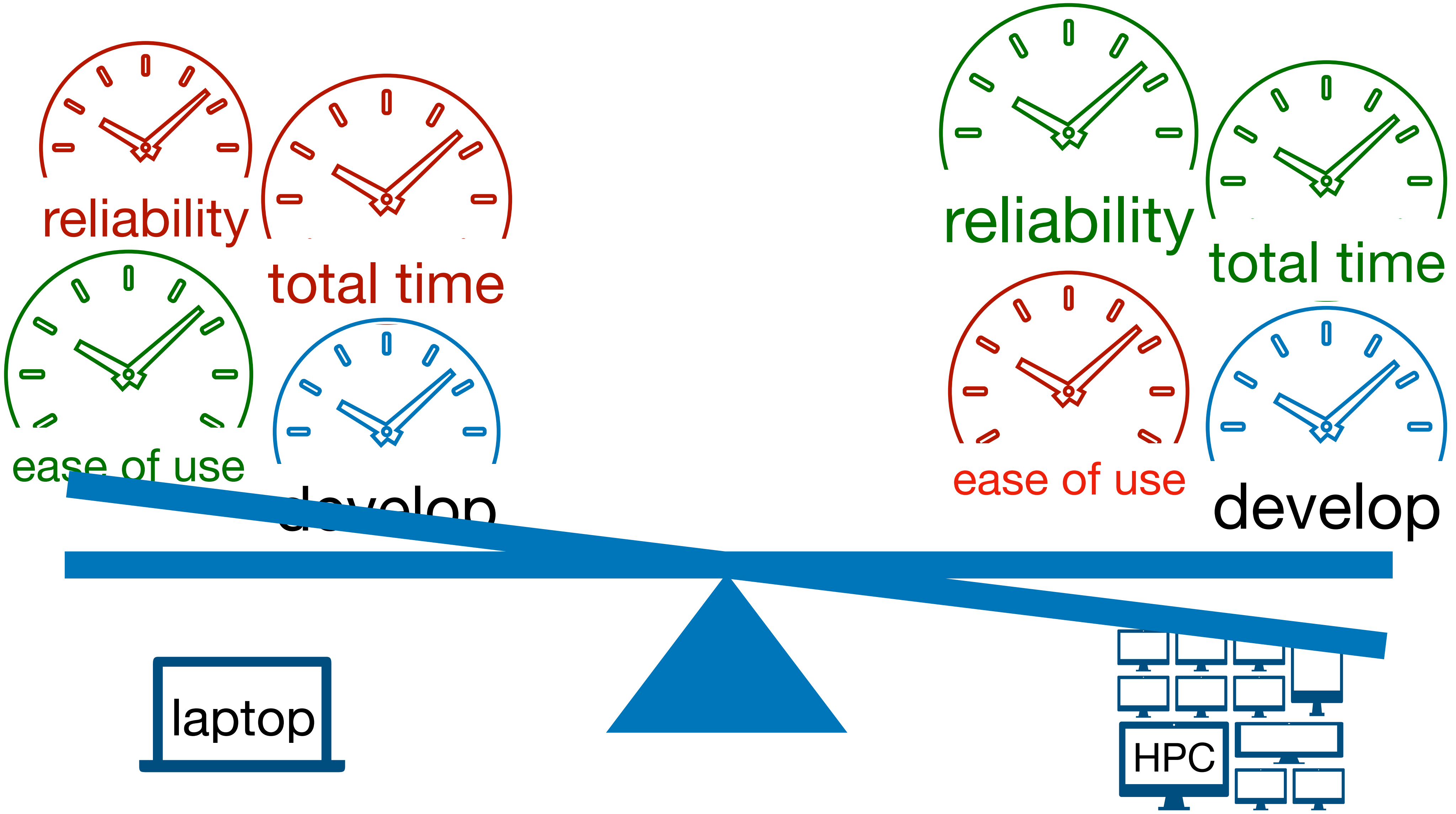
When to avoid HPC?

- When interaction is a big part!
- When visualization is a big part!
- When you are still “improving” algorithm
- Debugging, profiling and optimizing code
- BUT, sometimes you need to deploy on big nodes to test them. Then its necessary.
- PS: interaction and visualization are both possible - just need more effort to setup.

```
> interact

still waiting …
```

```
> visualize
display missing..
error!
```

Should I use HPC?



reliability

total time

ease of use

develop

laptop

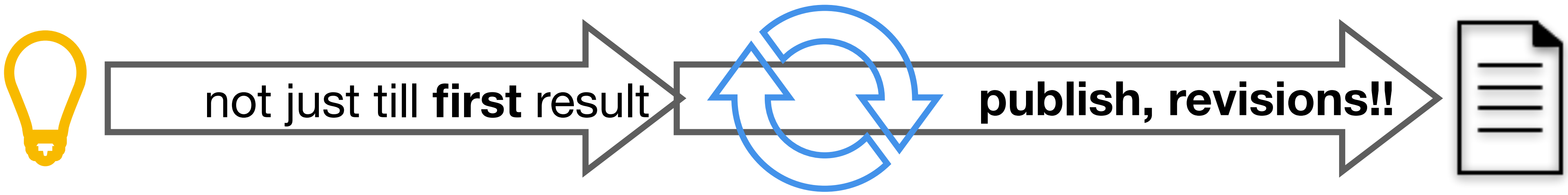
reliability

total time

ease of use

develop

HPC



not just till **first** result

publish, revisions!!

Over full project timeline

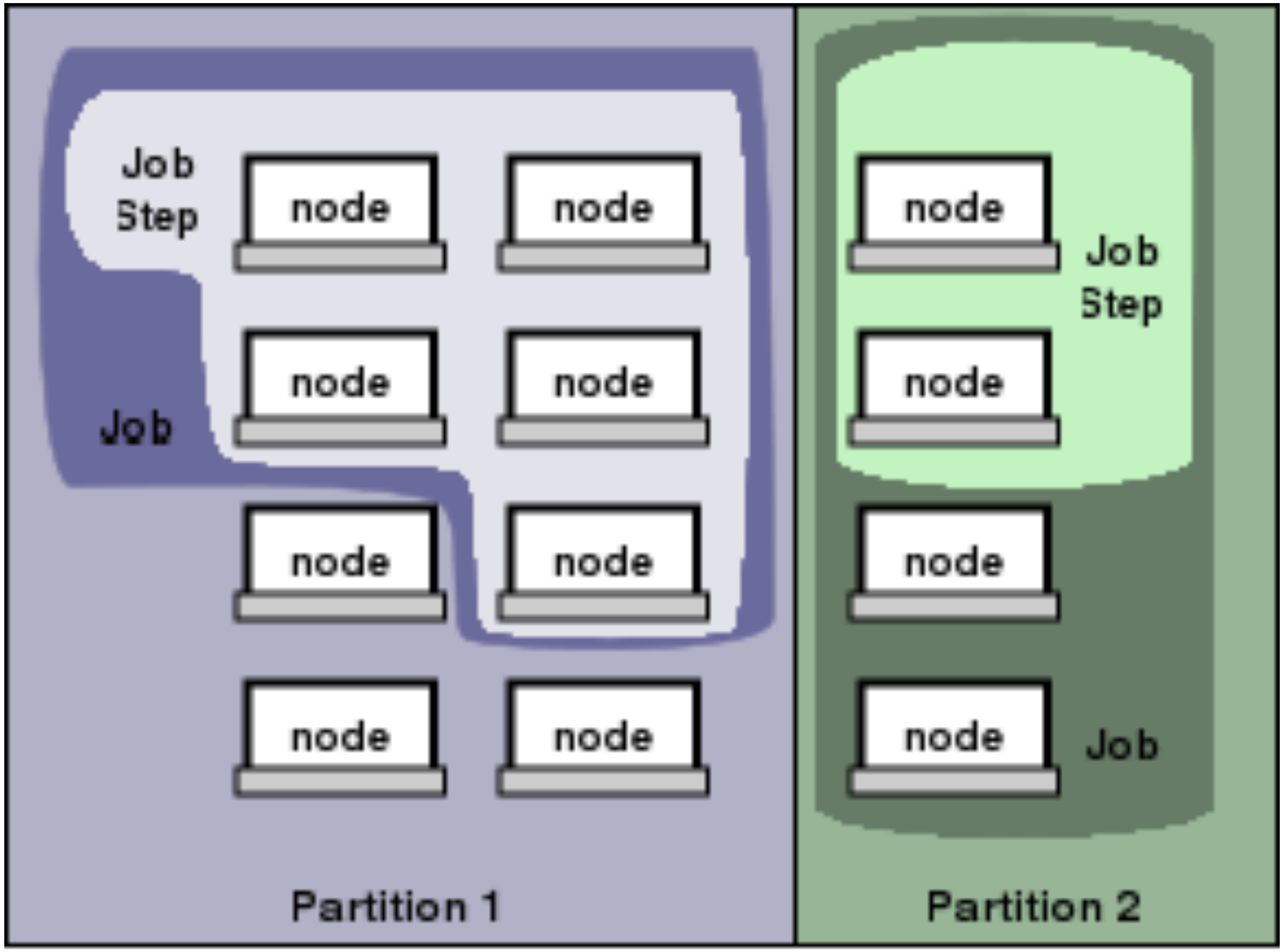
Batch processing

- Self-explanatory!
  - process a batch of jobs, in sequence!
  - non-interactive, to reduce idle time.
  - let’s face it: humans are slow!!
- Reduces startup & shutdown times, when run separately.
- Efficient use of resources (run when systems are idle)

Scheduler

- Allocates jobs to nodes (i.e. time on resources available)
- Applies priorities to jobs, according to policies and usage
- Enforces limits on usage (restricts jobs to its spec)
- Coordinates with the resource manager (accounting etc)
- Manages different queues within a cluster
    - customized with different limits on memory, CPU speed and number of parallel tasks etc.
- Manages dependencies (different “steps” within the same job)!

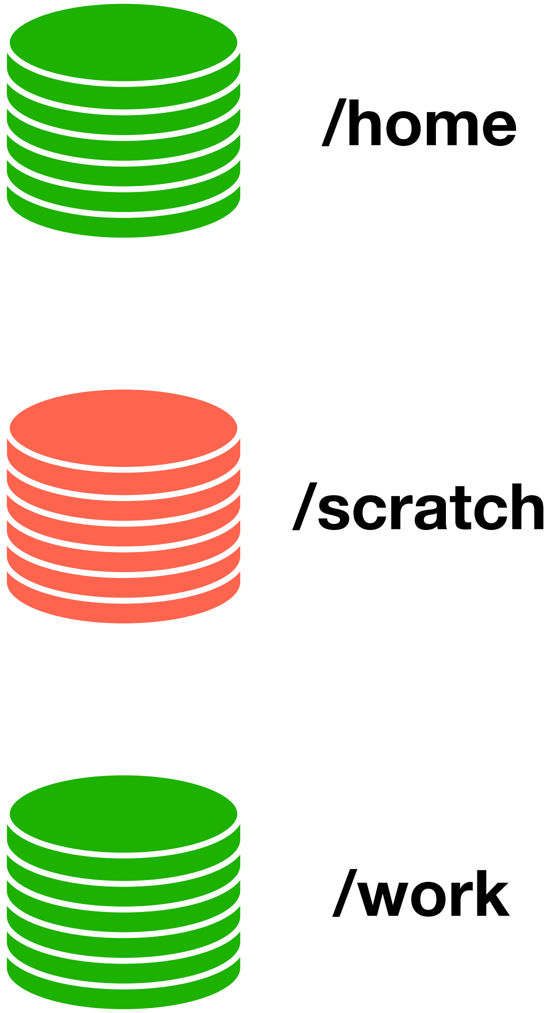
Terminology



© SchedMD, SLURM

File-system

- Major roles:
  - reduce latency in read/write
  - perform regular backup
- Enable concurrent access
  - to all nodes
  - to all users
- Amazing engineering behind!



Types of schedulers

	SGE	Torque/PBS	SLURM
Variables	Easy	Hard	**Easier**
Features	Reasonable	Reasonable	**Many**
Support	Low	Low	**Well-supported**
Administration	Not hard	Hard	**Easy**
Scalability	Medium	Low	**High (millions of jobs)**
Popularity	Okay	Not very	**Highly popular**

*Raamana’s personal opinion

Job Priority

Factor	Impact
First-come first-serve	jobs waiting longer get priority
Size of resources requested	smaller get priority, typically
	“whole node” jobs are preferred (over partial use)
Fair-share	groups/users with lesser usage get priority
Resource allocation	users or groups with prior allocations get higher priority. Must be set-up in advance.
*Terms and conditions apply	e.g. dependences, availability of resources, type of queue requested etc

https://docs.computecanada.ca/wiki/Job_scheduling_policies

Resource specification

Resource	SLURM	SGE
number of nodes	`-N [min[-max]]`	`N/A`
number of CPUs	`-n [count]`	`-pe [PE] [count]`
memory (RAM)	`--mem [size[units]]`	`-l mem_free=[size[units]]`
total time (wall clock limit)	`-t [days-hh:mm:ss] OR -t [min]`	`-l h_rt=[seconds]`
export user environment	`--export=[ALL | NONE | variables]`	`-V`
naming a job (important)	`--job-name=[name]`	`-N [name]`
output log (stdout)	`-o [file_name]`	`-o [file_name]`
error log (stderr)	`-e [file_name]`	`-e [file_name]`
join stdout and stderr	`by default, unless -e specified`	`-j yes`
queue / partition	`-p [queue]`	`-q [queue]`
script directive (inside script)	`#SBATCH`	`#$`
job notification via email	`--mail-type=[events]`	`-m abe`
email address for notifications	`--mail-user=[address]`	`-M [address]`

Useful glossary:
https://www.computecanada.ca/research-portal/accessing-resources/glossary/

Node specification

Resource	SLURM
restrict to particular nodes	`--nodelist=intel[1-5]`
exclude certain nodes	`--exclude=amd[6-9]`
based on features (tags)	`--constraint=“intel&gpu”`
to a specific partition or queue	`--partition intel_gpu`
based on number of cores/threads	`--extra-node-info=<sockets[:cores[:threads]]>`
type of computation	`--hint=[compute_bound,memory_bound,multithread]`
contiguous	`--contiguous`
CPU frequency	`--cpu-freq=[Performance,Conservative,PowerSave]`

Useful glossary:
https://www.computecanada.ca/research-portal/accessing-resources/glossary/

Making a job from a script

```
> sbatch -n 1 --mem=4G -t 2:00 -o my_output.txt
  python /home/quark/script.py
```

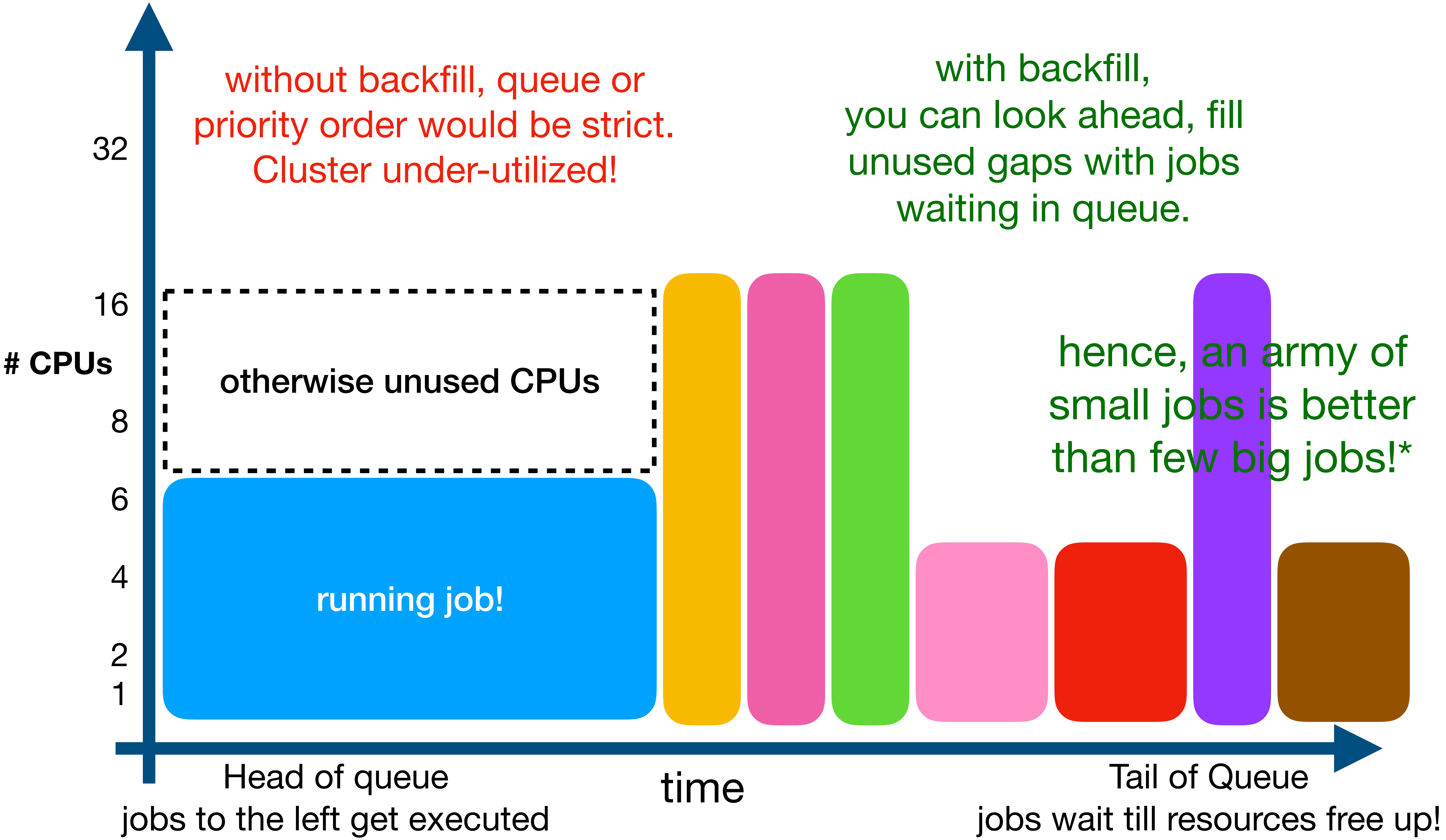
```
#!/bin/bash
#SBATCH -p general         # which partition/queue
#SBATCH -N 1               # number of nodes
#SBATCH -n 1               # number of cores
#SBATCH --mem=4G           # total memory
#SBATCH -t 0-2:00          # time (D-HH:MM)
#SBATCH -o my_output.txt

# command to invoke your script
python /home/quark/script.py
R CMD BATCH stats.R
matlab -nodesktop -r matrix.m
```

Recommended

Invoking script from shell

Language / environment	Shell command
shell script	`bash script.sh`
python	`python script.py`
matlab	`matlab -nodesktop -r script.m`
R	`R CMD BATCH script.R`

Being precise and smaller is wiser! (backfill policy)



without backfill, queue or priority order would be strict. Cluster under-utilized!
with backfill, you can look ahead, fill unused gaps with jobs waiting in queue.
32
16
# CPUs
8
6
4
2
1
otherwise unused CPUs
running job!
hence, an army of small jobs is better than few big jobs!*
Head of queue
jobs to the left get executed
time
Tail of Queue
jobs wait till resources free up!

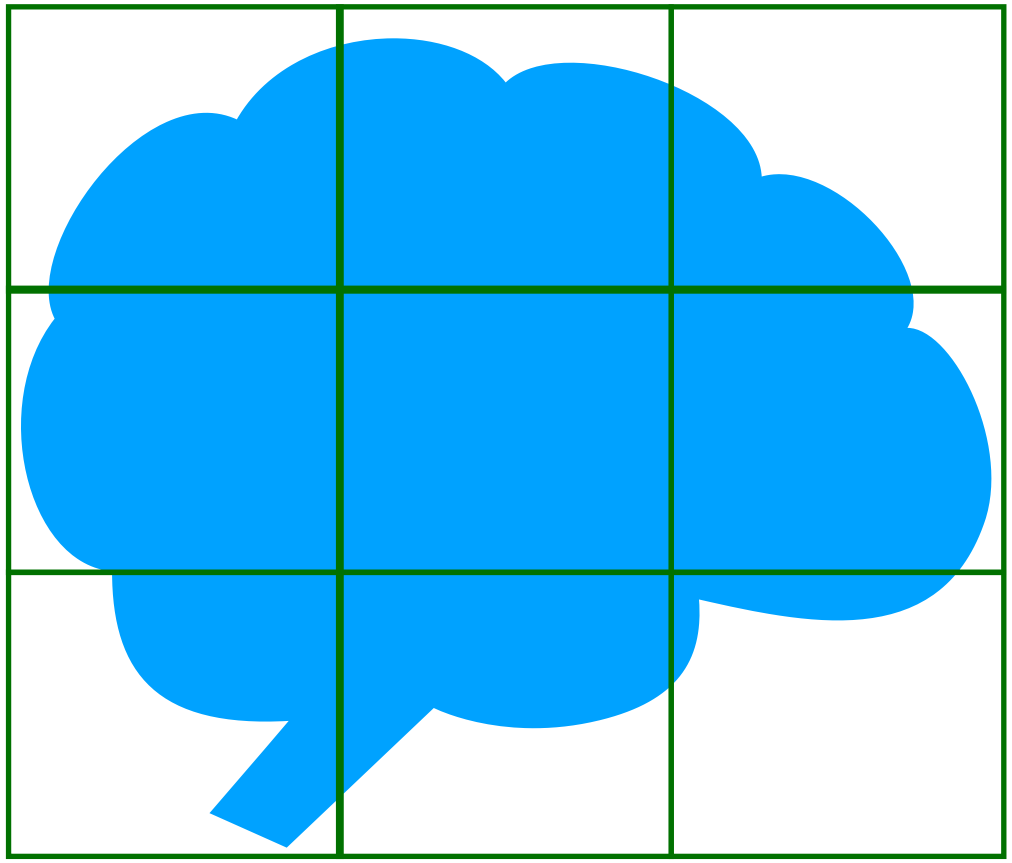
*backfill might not always be enabled. Assumes users specified resources!

Splitting your workflow

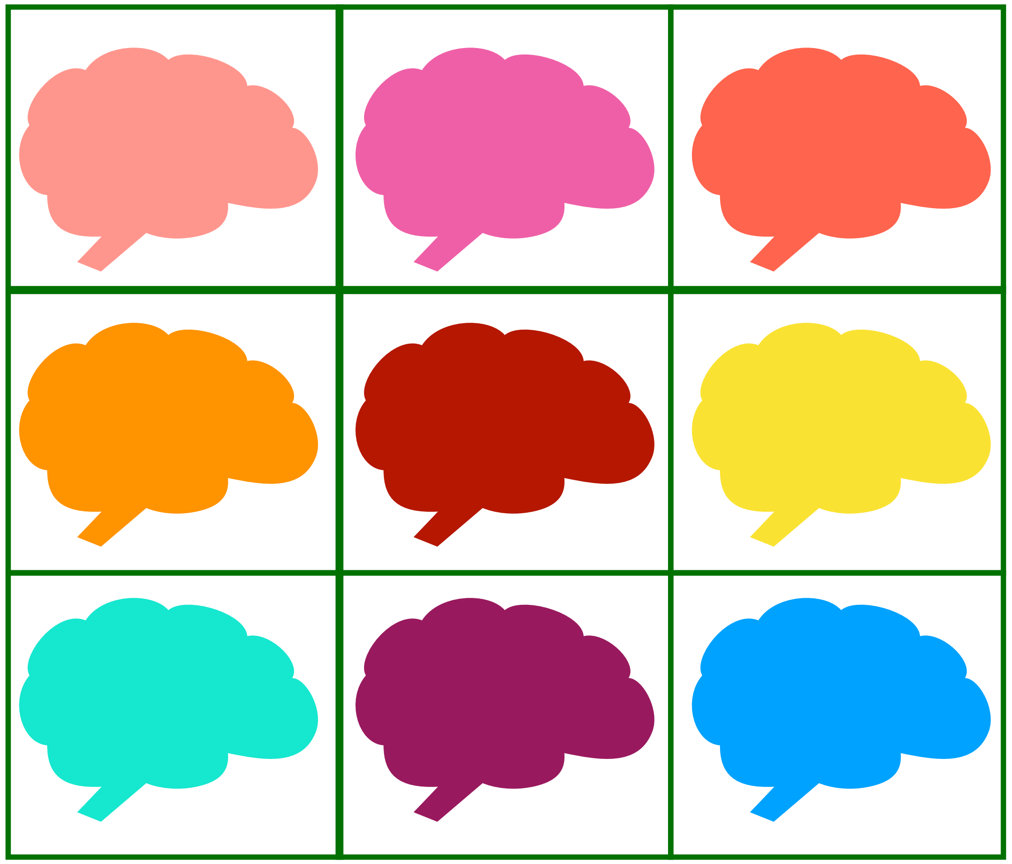
- Some tasks are highly parallel
  - painting different walls
- Some tasks have to wait for others
  - Installing roof, needs all walls built first.

Slicing Total Processing

Data parallelism



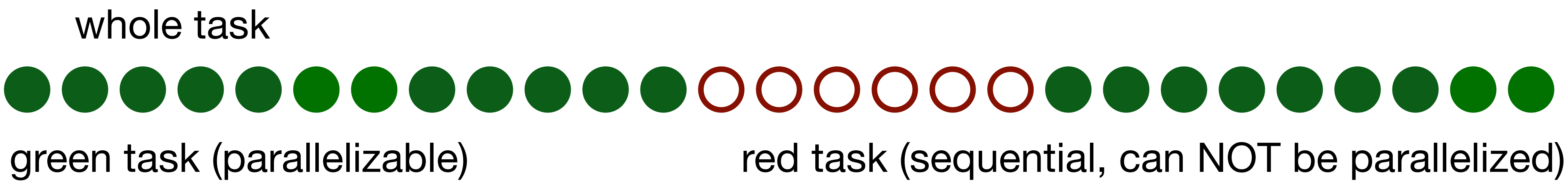
Task parallelism



you can do both, although with varying returns!

Note: these are mostly embarrassingly parallel!

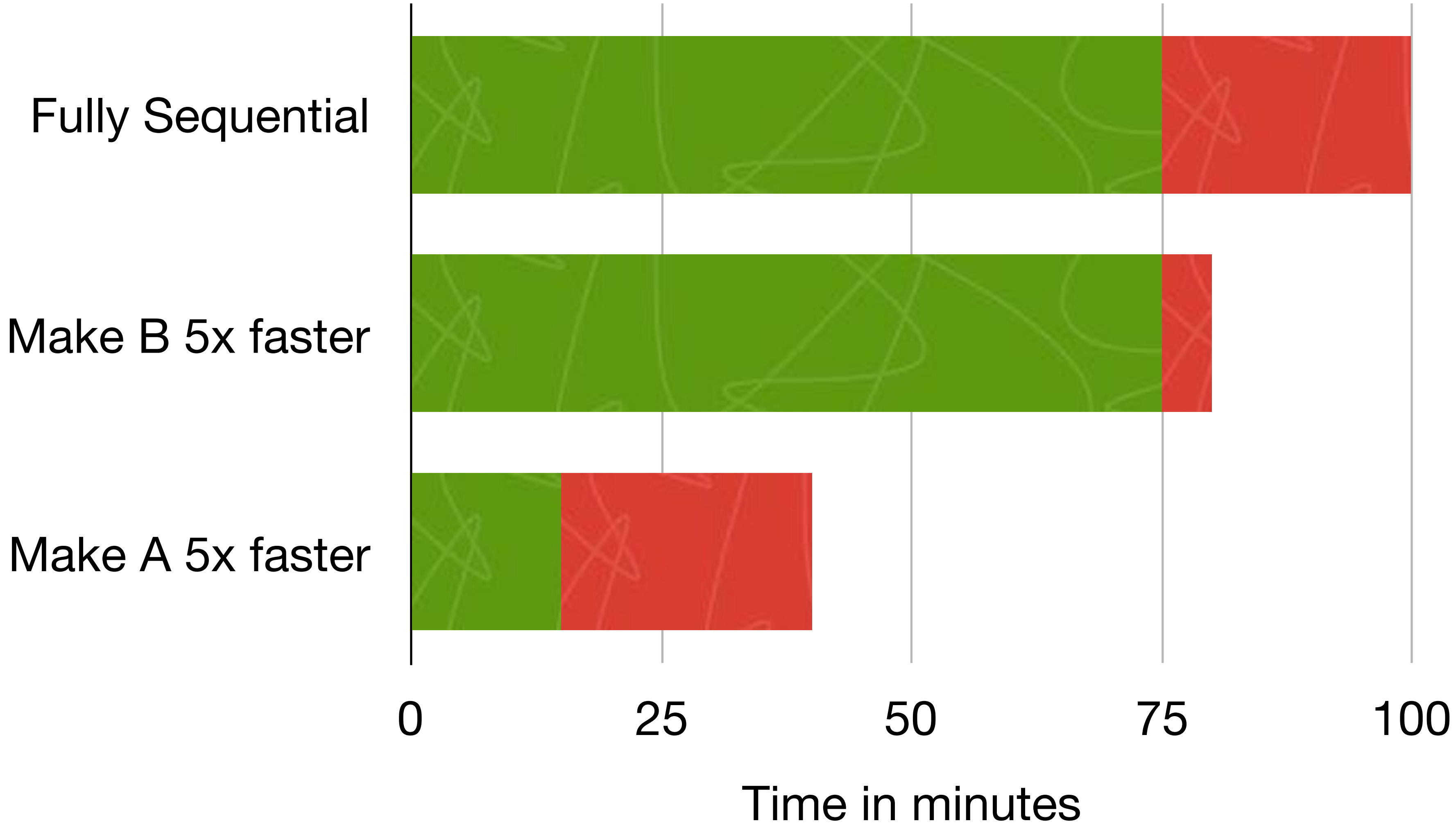
Where to speed up?



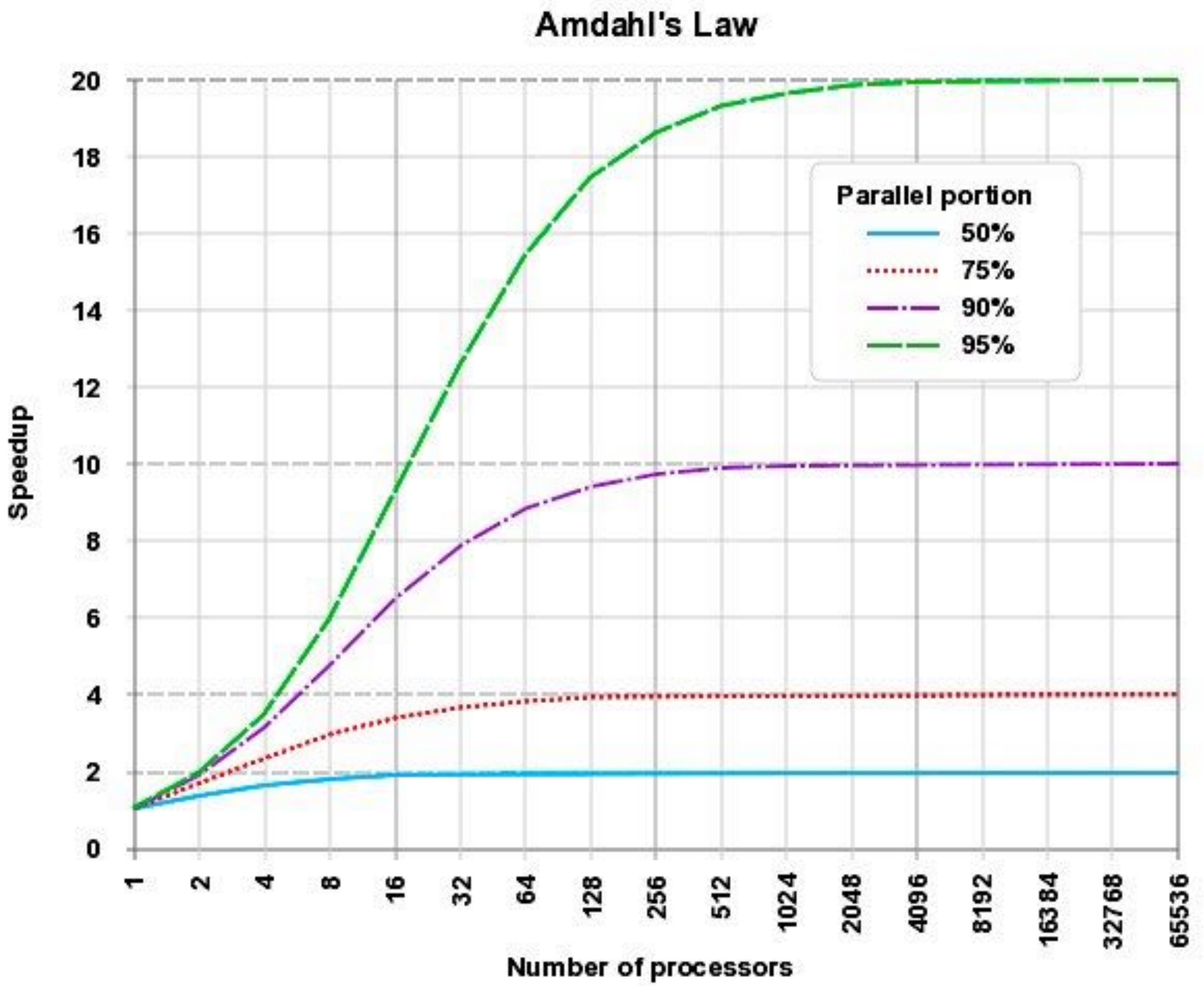
whole task

green task (parallelizable)

red task (sequential, can NOT be parallelized)



Limits on speed up



© wikipedia

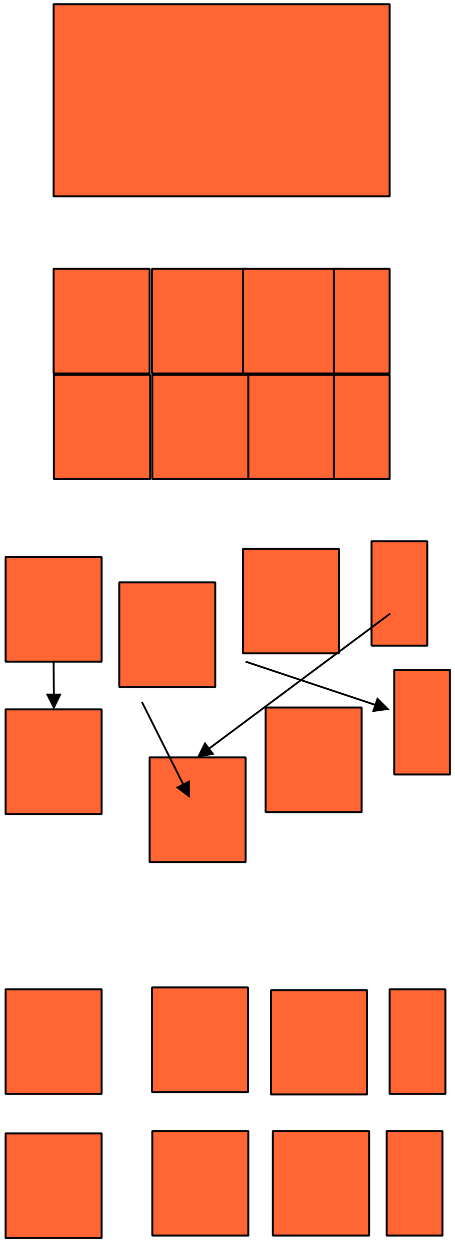
High-Performance Computing involves many distinct computer processors working together on the same calculation.

Large problems are divided into smaller parts and distributed among the many computers.

Usually **clusters** of quasi-independent computers which are coordinated by a central scheduler.

Applications must be written specifically to take advantage of distributed computing.

- Explicitly split your problem into smaller “chunks”
- “Message passing” between processes
- Entire computation can be slowed by one or two slow chunks
- Exception: “embarrassingly parallel” problems
- Easy-to-split, independent chunks of computation
- Thankfully, many useful models fall under this heading. (e.g. stochastic models)



“Embarrassingly parallel” = No inter-process communication

Checklist: before you submit

- Test and debug your code locally
  - starting with each of the small parts of the pipeline
  - whether they are integrated well
  - reduce redundancy, choosing right output formats etc
  - sloppy testing and debugging could cost you a lot, later on!!
- Test your environment
  - Run the job locally on the headnode or login node
  - If not, you can request an interactive job
- Do I have enough disk space?
- Chalk out job requirements in speed, walltime, RAM, number of jobs etc
  - to reduce the total processing time (at the level of dataset and experiment)
  - You many need to select appropriate queue or partition to match your needs and specifications (otherwise you might wait in line forever
- Decide on whether to insert checkpoint logic and code
- Decide on whether to insert Profiling code (measure its effective speed in different parts of pipeline)
- Decide on whether to retain intermediate or scratch outputs?

Checklist: before you submit

- Always try to specify resources!
  - defaults are not necessarily the best for your need!
  - job gets scheduled quickly, choosing right queue/specs.
  - reduces the trial and error to get to the right nodes w/ resources
  - reduces wastage - don’t take up 8 CPUs and 32Gb to print(“Hello, World!”)
  - “Know your job” well (profiling!)
- Save the job specifications to a file (do not rely on shell history)
- Estimate requirements precisely, but be conservative in requesting - add 10-20%
  - If your matrix needs 2.5GB, specify 4GB for job.
  - Remember OS on the nodes takes up some RAM - so if the node physically has 32GB, it needs a 2-3GB to run and stay alive. Only jobs requiring less than 30GB will be scheduled to it. Jobs requiring exactly 32GB will be sent to nodes with more than RAM (64 or 128GB)

Checklist: profile your job

- Many tools are available in Linux to “profile” the memory usage and time usage for different parts of your program.
  - `top / htop`
  - `free / vmstat`
  - `time`
- Plugins to IDEs for your language
- Explicit profiling is typically not necessary!
  - you know your job during development.

- Check the file sizes created during “trial” runs
  - Keep only what is necessary, after testing!
  - You don’t need to specify disk space, but need to ensure you won’t create more files exceeding your quota (aggregate over all jobs)
  - tools: `quota, du` or `df -h`
- If you are unable to profile on your desktop,
  - request an interactive job!
  - Once obtained, acts like your desktop!
  - Need to think about whether you need a display, when you run jobs!

Checklist: during execution

- Regularly check on job status
  - because jobs fail! Many reasons! It sucks. It hurts. No matter how well you tested your code!
  - Some factors (like network, file system and weather) are not in your control.
  - Better to accept failures, and reduce the time to resubmit them.
- Hence **checkpointing** is important!
  - so you reuse what was finished already before failure!
- You may need to write scripts to get an accurate estimate of status of processing!
  - as your pipeline can be complicated
  - rely on files written to disk, than text output in a log
  - unless you designed it that way

Checklist: after execution

- Check various things!!
  - Check file sizes (file being there doesn't mean it has data)
  - Visualize them (data present doesn't mean its accurate)
  - Sweep across all jobs!
  - Check disk usage.

- Track usage:
  - memory, walltime, disk I/O etc.
  - to optimize job specs next time
  - as it's never a one time thing!
- Again, scripts can help
  - automating this process
  - mad shell skills also help.

Checklist

Before	Test and profile code, locally!	*Run a test job to test environment & config*	Chalk out requirements
During	Look for any failures!	Monitor usage!	Resubmit, correcting any simple mistakes
After	Check logs & outputs	Assume the worst!	Visualize and verify, do not assume!
Do	Automate checks when possible!	Identify areas for optimization (repkg)	Regular cleanups (shared file systems)
Avoid	Don’t create too many small files!	Avoid ASCII (text) format for large files	Relative paths (use absolute paths)
	Don’t use MS Word (hidden characters): use text editor or vi		

Data transfer tools

Task	Recommended	Alternatives
download	wget URL	browser and scp
synchronize	rsync -av /src server:/dest	scp (batch, simplest) sftp (interactive) bbcp (parallel; large sizes)
reduce size	tar -cvf (create / zip) tar -xvf (extract/unzip)	zip unzip
software	FileZilla(desktop to cluster) Globus (between clusters)	WinSCP (windows) FireFTP (cross-platform) Transmit, Fugu (Mac)

When in doubt, don’t delete data.

When in doubt, email the admins!

https://docs.computecanada.ca/wiki/Globus

Data management plan!

- Calculate size of data you’ll produce from test runs.
- What do you need to “keep”, and how long?
- When is data “final” and needs to be backed up?
- What is scratch and deletable, and what is not?
- Is the “intermediate” data easy & quick to regenerate?
    - If so, should you even store it?

Should I build a pipeline?

	Yes 4/6?	
• Can few parts of my project be automated? Together?	Yes	No
• Do I repeat this processing/analysis? More than twice?	Yes	No
• Even if they are repeated in a different manner, can I capture the variations in logic?	Yes	No
• Are there delays due to human involvement?	Yes	No
• Is it difficult to redo this on a different dataset or by others?	Yes	No
• Are there concerns of reproducibility in my analysis?	Yes	No

My thesis: “Most things can be automated!”

Building pipelines

- We usually need to stitch together a diverse array of tools (AFNI, FSL, Python, R etc) to achieve a larger goal (build a pipeline)
    - They are often written in different programming languages (Matlab, C++, Python, R etc)
    - Mostly compiled, and no APIs
    - To reduce your pain, you can use `bash` or Python to develop a pipeline.
    - If it's neuroimaging-specific, check `nipy` also
- So, learning a bit of bash/Python really helps!
- be warned, bash is not super easy, but very helpful for relatively **straightforward pipelines**!

no heavy logic? Use:

BASH
THE BOURNE-AGAIN SHELL

When in doubt, use:

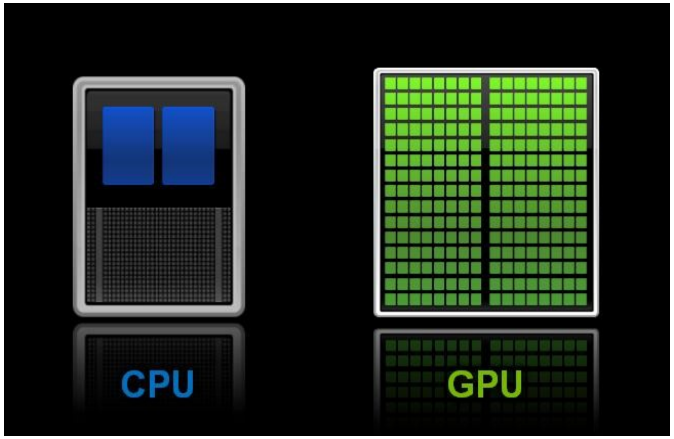
python™

HPC Skills

- Learning Linux goes a long way.
  - Most HPC clusters are in Linux!
  - It is reliable and free.
  - Great to build pipelines
- Understanding of scheduling
- Command-line skills
  - batch processing is king!!
  - human interaction is slow!
- Scripting in bash/python
  - to stitch together routine or repetitive tasks into a pipeline!

Recent developments: GPUs

Graphics processing units



- CPU: complex, general-purpose processor
- GPU: highly-specialized parallel processor, optimized for performing operations for common graphics routines
- Highly specialized → many more “cores” for same cost and space
  - Intel Core i7: 4 cores @ 3.4 GHz: $300 = $75/core
  - NVIDIA Tesla M2070: 448 cores @ 575 MHz: $4500 = $10/core
- Also higher bandwidth: 100+ GB/s for GPU vs 10-30 GB/s for CPU
- Same operations can be adapted for non-graphics applications: “GPGPU”

Image from http://blogs.nvidia.com/2009/12/whats-the-difference-between-a-cpu-and-a-gpu/

External resources

- One solution to handling complexity: **outsource it!**
- Historical HPC facilities: universities, national labs
  - Often have the most absolute compute capacity, and will sell excess capacity
  - Competition with academic projects, typically do not include SLA or high-level support
- Dedicated commercial HPC facilities providing “on-demand” compute power.

External HPC

- Outsource HPC sysadmin
- No hardware investment
- Pay-as-you-go
- Easy to migrate to new tech

Internal HPC

- Requires in-house expertise
- Major investment in hardware
- Possible idle time
- Upgrades require new hardware

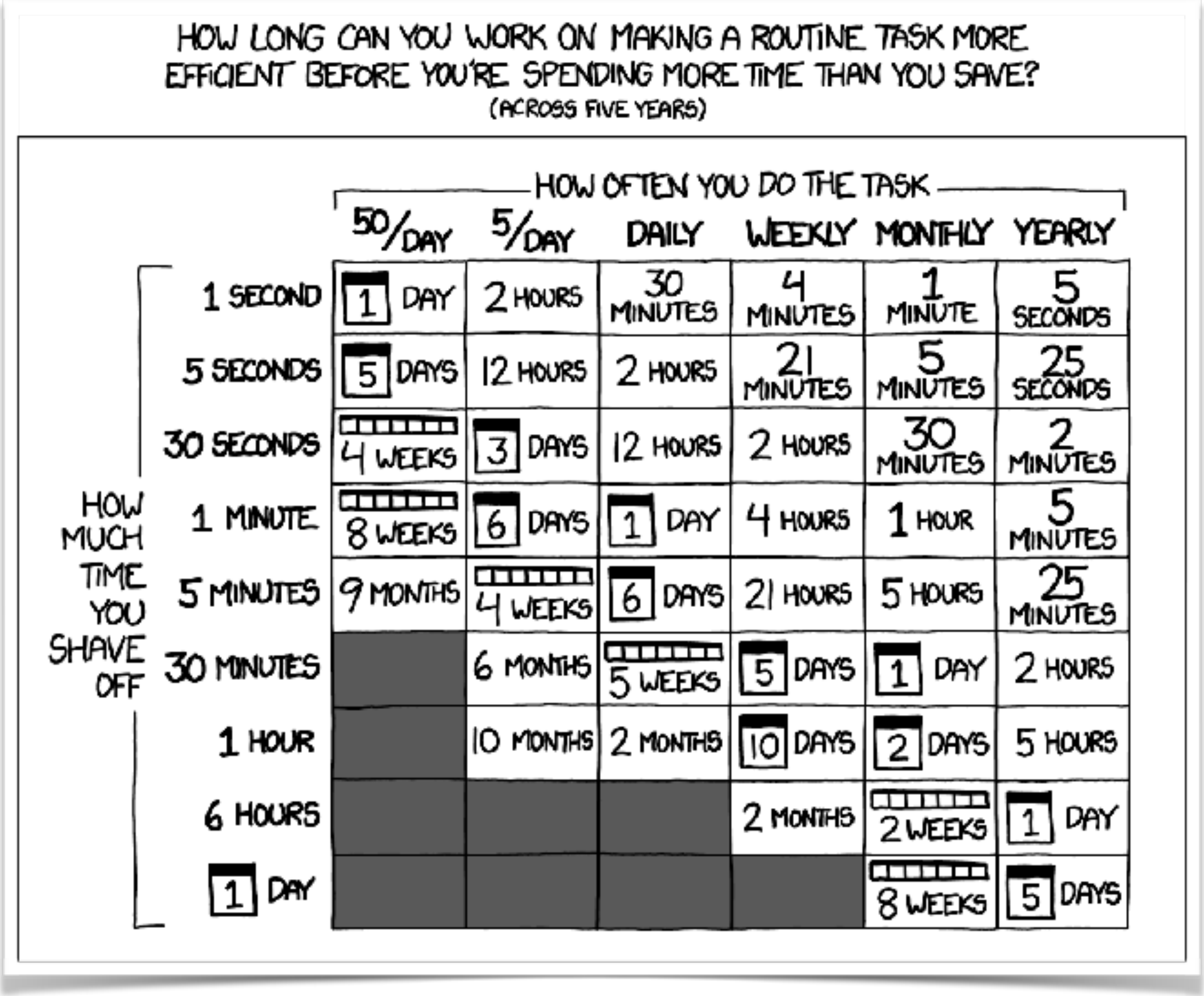
Internal HPC	External HPC
No external contention	No guaranteed access
All internal—easy security	Security arrangements complex
Full control over configuration	Limited control of configuration
Simpler licensing control	Some licensing complex
Requires in-house expertise	Outsource HPC sysadmin
Major investment in hardware	No hardware investment
Possible idle time	Pay-as-you-go
Upgrades require new hardware	Easy to migrate to new tech

“The Cloud”

- “Cloud computing”: virtual machines, dynamic allocation of resources in an external resource
- Lower performance (virtualization), higher flexibility
- Usually no contracts necessary: pay with your credit card, get 16 nodes
- Often have to do all your own sysadmin
- Low support, high control

Submit your questions now!

I will batch process them in queue.



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK					
	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY					8 WEEKS	5 DAYS

Any questions before we move on to a hands-on demo?