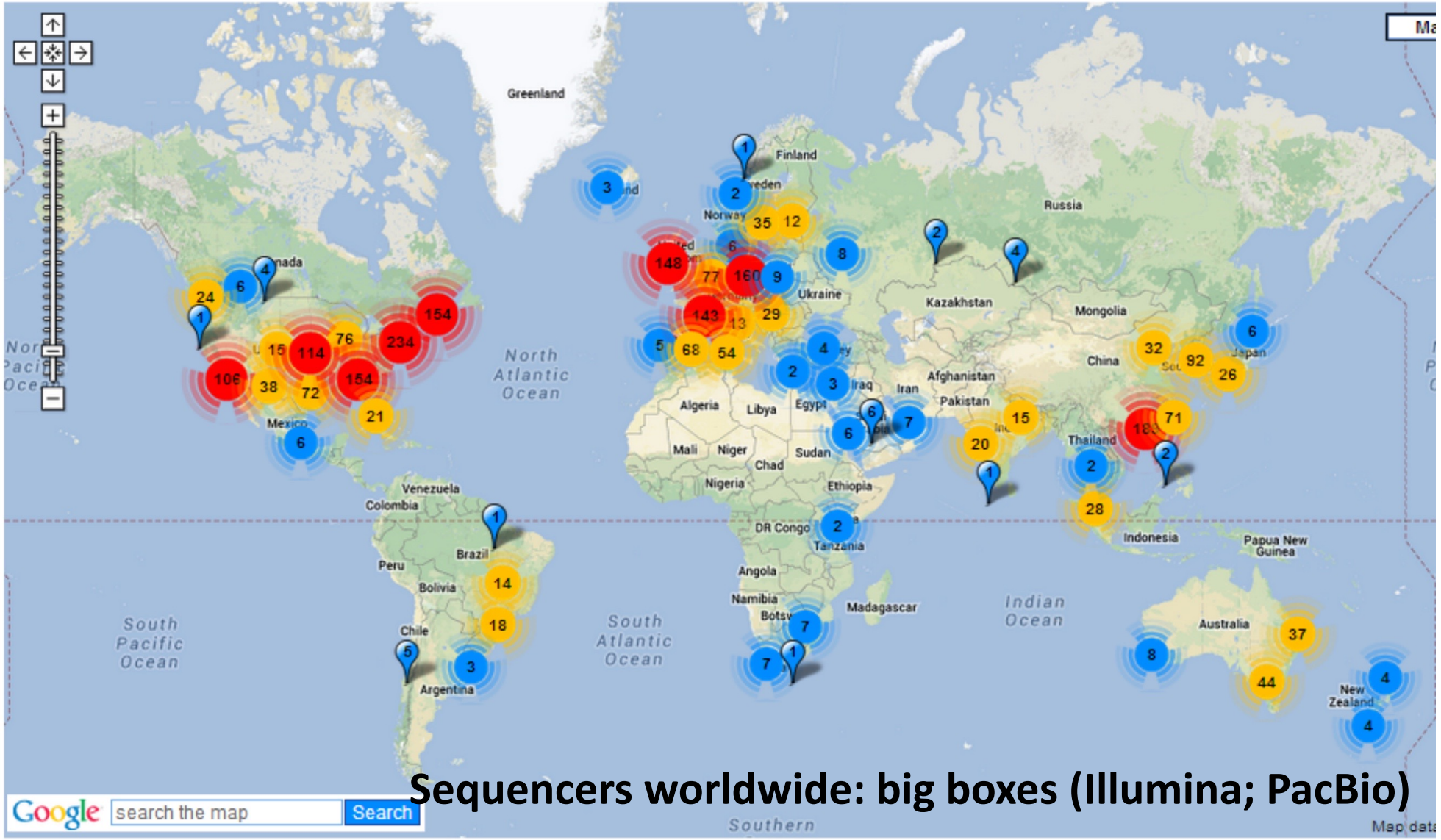
HPC

Is Biology now a data science?

And if so,

what does that mean?

Is Biology now a data science?



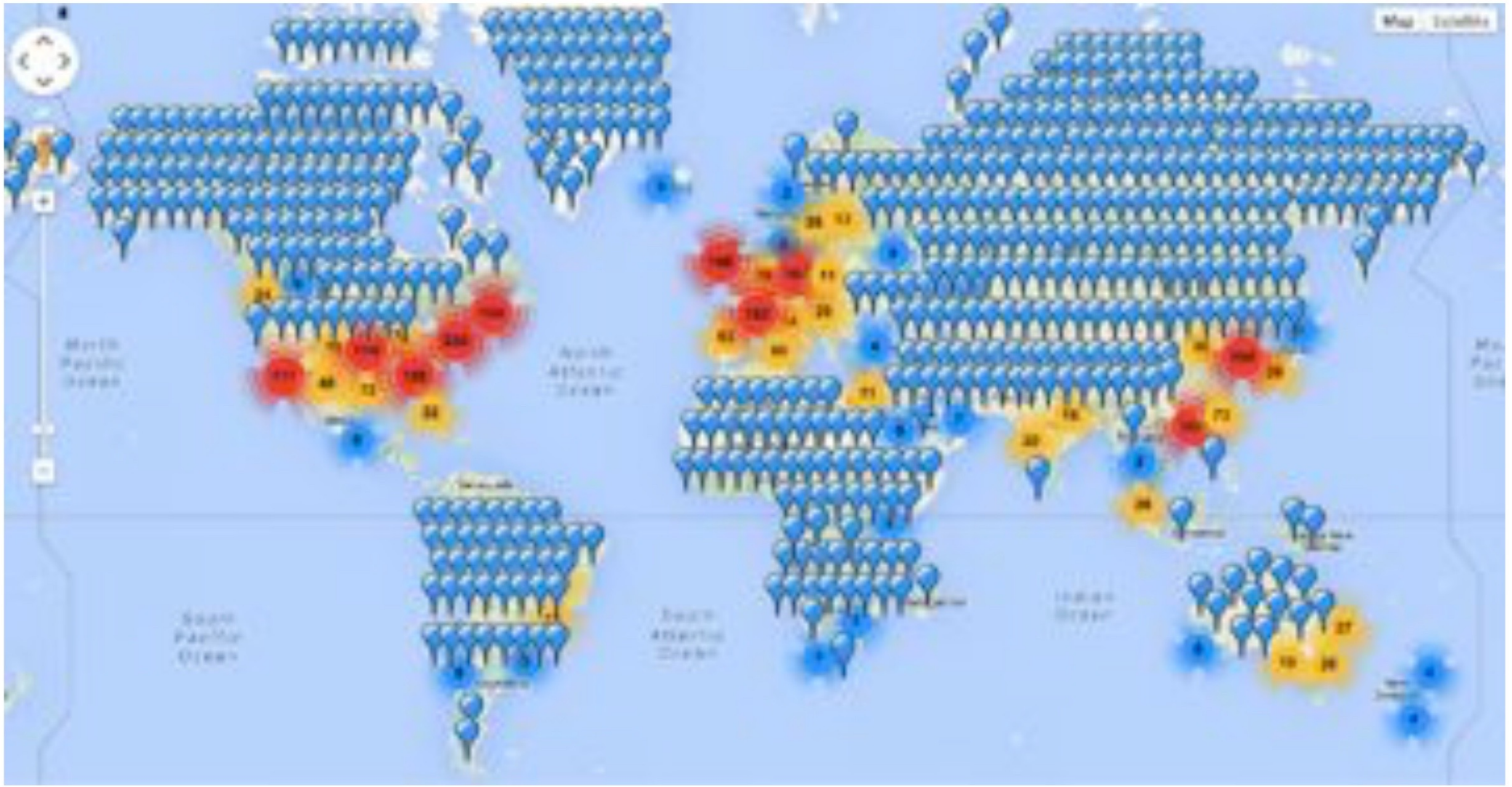
Sequencers worldwide: big boxes (Illumina; PacBio)

Adapted from Mike Schatz

But sequencers are getting smaller; more democratic

MIT Technology Review 2017

But sequencers are getting smaller; more democratic



Is Biology now a data science?

And if so,

what does that mean?

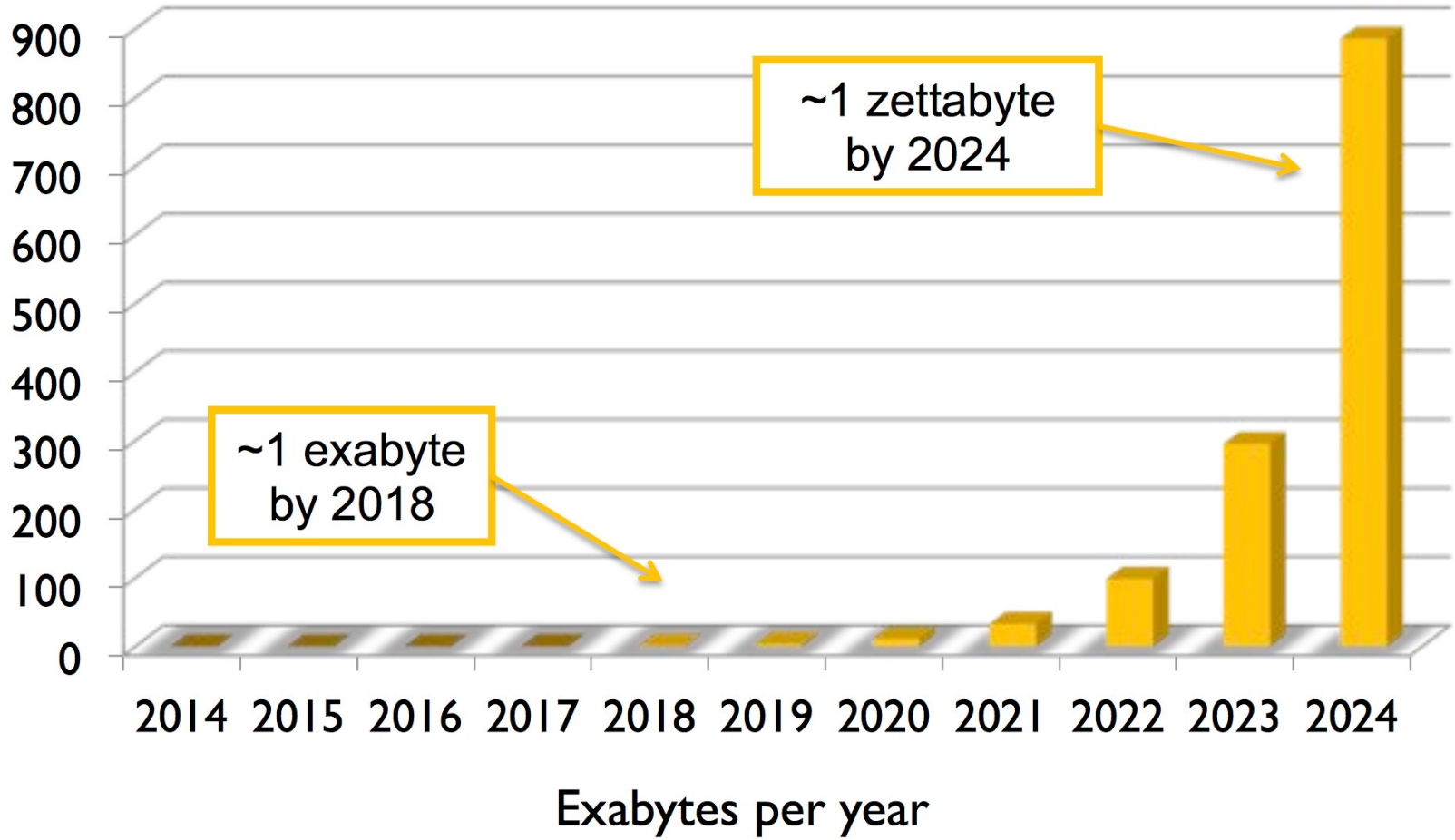
Ubiquitous Recording

Onsite accumulation

High Density Data

Accumulation!!

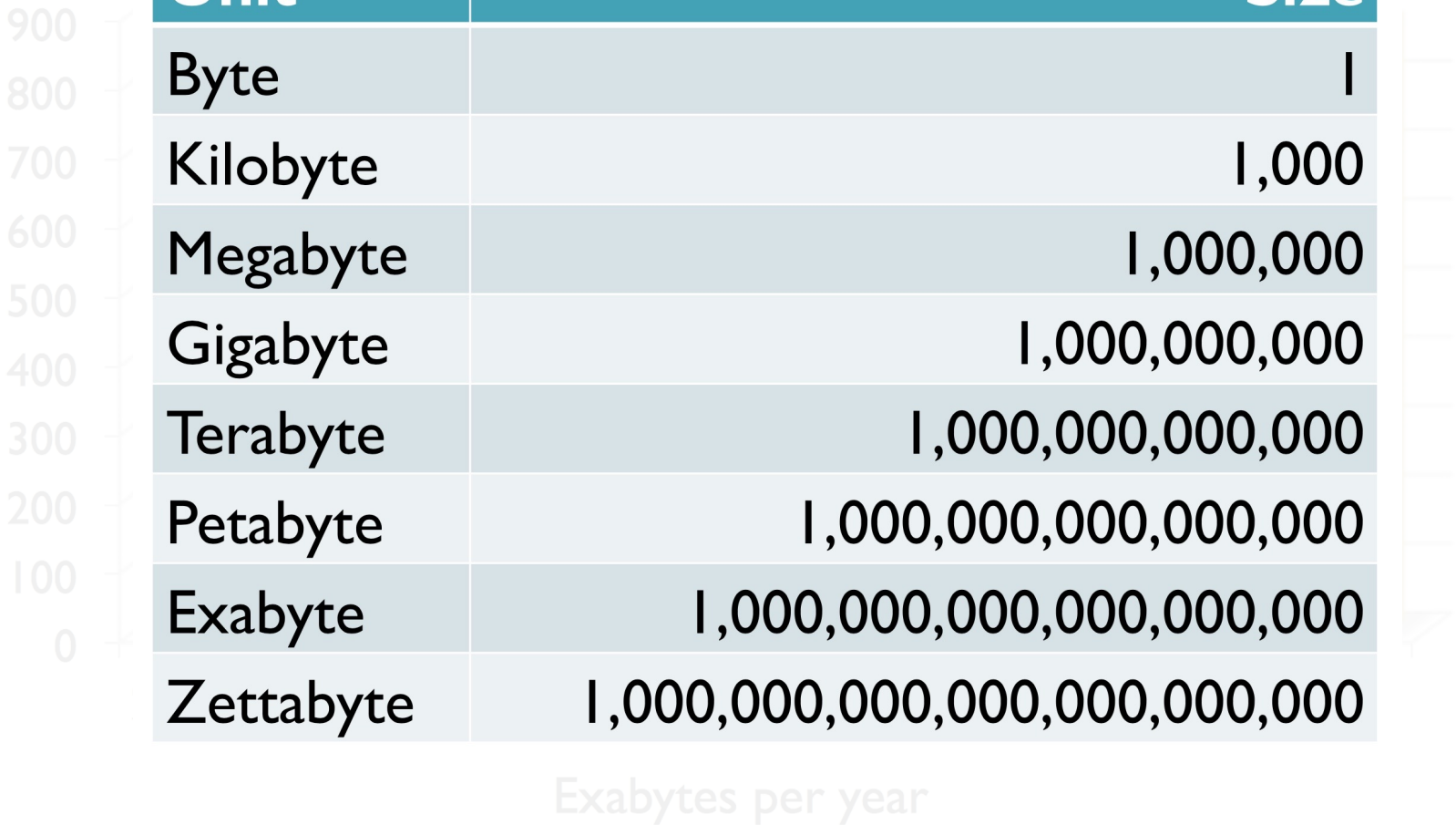
Current world-wide sequencing capacity is growing at ~3x per year!



You can't record that; you can barely compress it

Current world-wide sequencing capacity is growing at ~3x per year!

Unit	Size
Byte	1
Kilobyte	1,000
Megabyte	1,000,000
Gigabyte	1,000,000,000
Terabyte	1,000,000,000,000
Petabyte	1,000,000,000,000,000
Exabyte	1,000,000,000,000,000,000
Zettabyte	1,000,000,000,000,000,000,000



DNA sequencing as an ephemeral sensing technology

Perspective

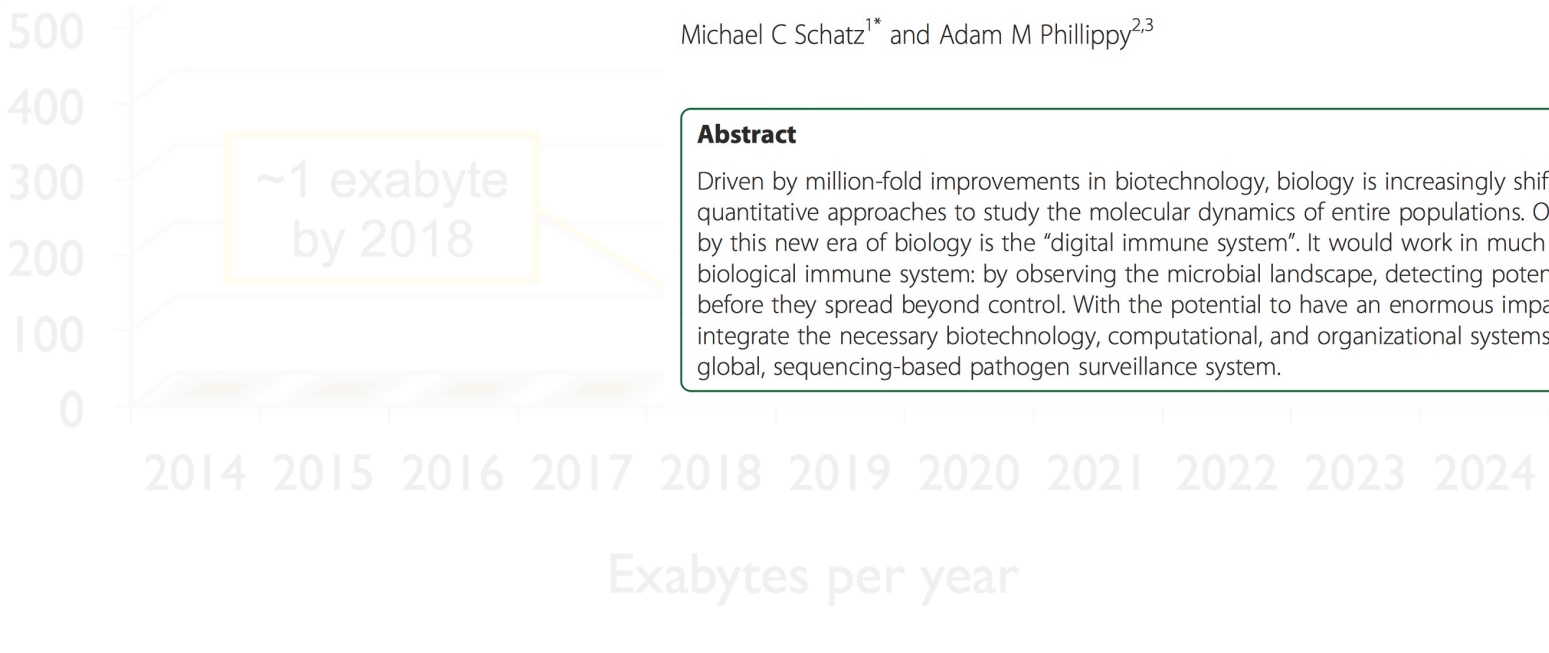
A vision for ubiquitous sequencing

Yaniv Erlich[1,2]

[1]Department of Computer Science, Columbia University, New York, New York 10027, USA; [2] New York, New York 10013, USA

Genomics has recently celebrated reaching the $1000 genome milestone, making affo
With this goal successfully completed, the next goal of the sequencing revolution can be
sequencing devices that are manufactured for real-time applications and deployed in lar
part of this manuscript envisions applications that will benefit from moving the sequenc
mains. In the second part, the manuscript outlines the critical barriers that need to be add
ubiquitous sequencing sensors.

owing at ~3x per year!



Schatz and Phillippy *GigaScience* 2012, **1**:4
http://www.gigasciencejournal.com/content/1/1/4

(GIGA)n SCIENCE

COMMENTARY Open Access

The rise of a digital immune system

Michael C Schatz[1*] and Adam M Phillippy[2,3]

Abstract

Driven by million-fold improvements in biotechnology, biology is increasingly shifting towards high-resolution, quantitative approaches to study the molecular dynamics of entire populations. One exciting application enabled by this new era of biology is the "digital immune system". It would work in much the same way as an adaptive, biological immune system: by observing the microbial landscape, detecting potential threats, and neutralizing them before they spread beyond control. With the potential to have an enormous impact on public health, it is time to integrate the necessary biotechnology, computational, and organizational systems to seed the development of a global, sequencing-based pathogen surveillance system.

DNA sequencing as an ephemeral sensing technology

Perspective

A vision for ubiquitous sequencing

Yaniv Erlich[1,2]

[1]Department of Computer Science, Columbia University, New York, New York 10027, USA; [2] New York, New York 10013, USA

Genomics has recently celebrated reaching the $1000 genome milestone, making affo
With this goal successfully completed, the next goal of the sequencing revolution can be
sequencing devices that are manufactured for real-time applications and deployed in lar
part of this manuscript envisions applications that will benefit from moving the sequenc
mains. In the second part, the manuscript outlines the critical barriers that need to be add
ubiquitous sequencing sensors.

owing at ~3x per year!

Schatz and Phillippy *GigaScience* 2012, **1**:4
http://www.gigasciencejournal.com/content/1/1/4

(GIGA)n SCIENCE

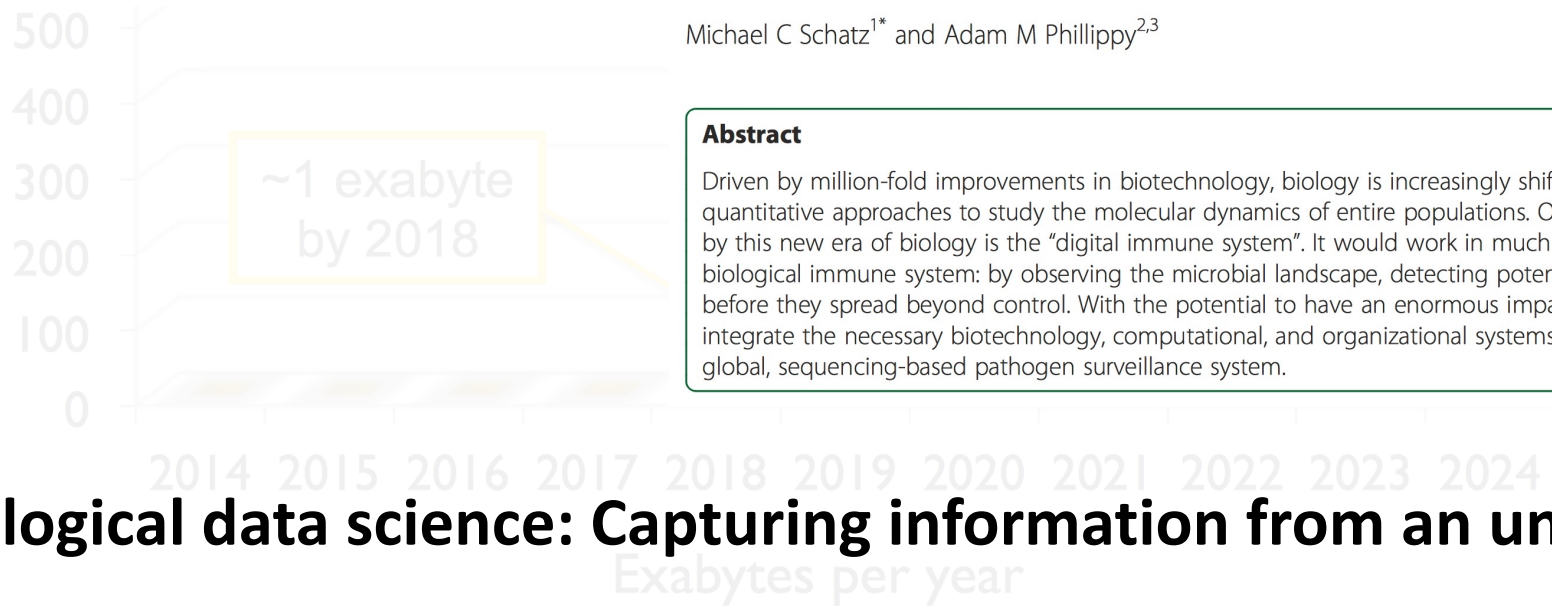
COMMENTARY Open Access

The rise of a digital immune system

Michael C Schatz[1*] and Adam M Phillippy[2,3]

Abstract

Driven by million-fold improvements in biotechnology, biology is increasingly shifting towards high-resolution, quantitative approaches to study the molecular dynamics of entire populations. One exciting application enabled by this new era of biology is the "digital immune system". It would work in much the same way as an adaptive, biological immune system: by observing the microbial landscape, detecting potential threats, and neutralizing them before they spread beyond control. With the potential to have an enormous impact on public health, it is time to integrate the necessary biotechnology, computational, and organizational systems to seed the development of a global, sequencing-based pathogen surveillance system.



A vision of biological data science: Capturing information from an unrecordable stream

Agenda:

1. Talking

* Museum Resources / Quotas/ Website
* Be Nice Rules
* Parallel topics: embarrassingly parallel; threads; MPI

Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)

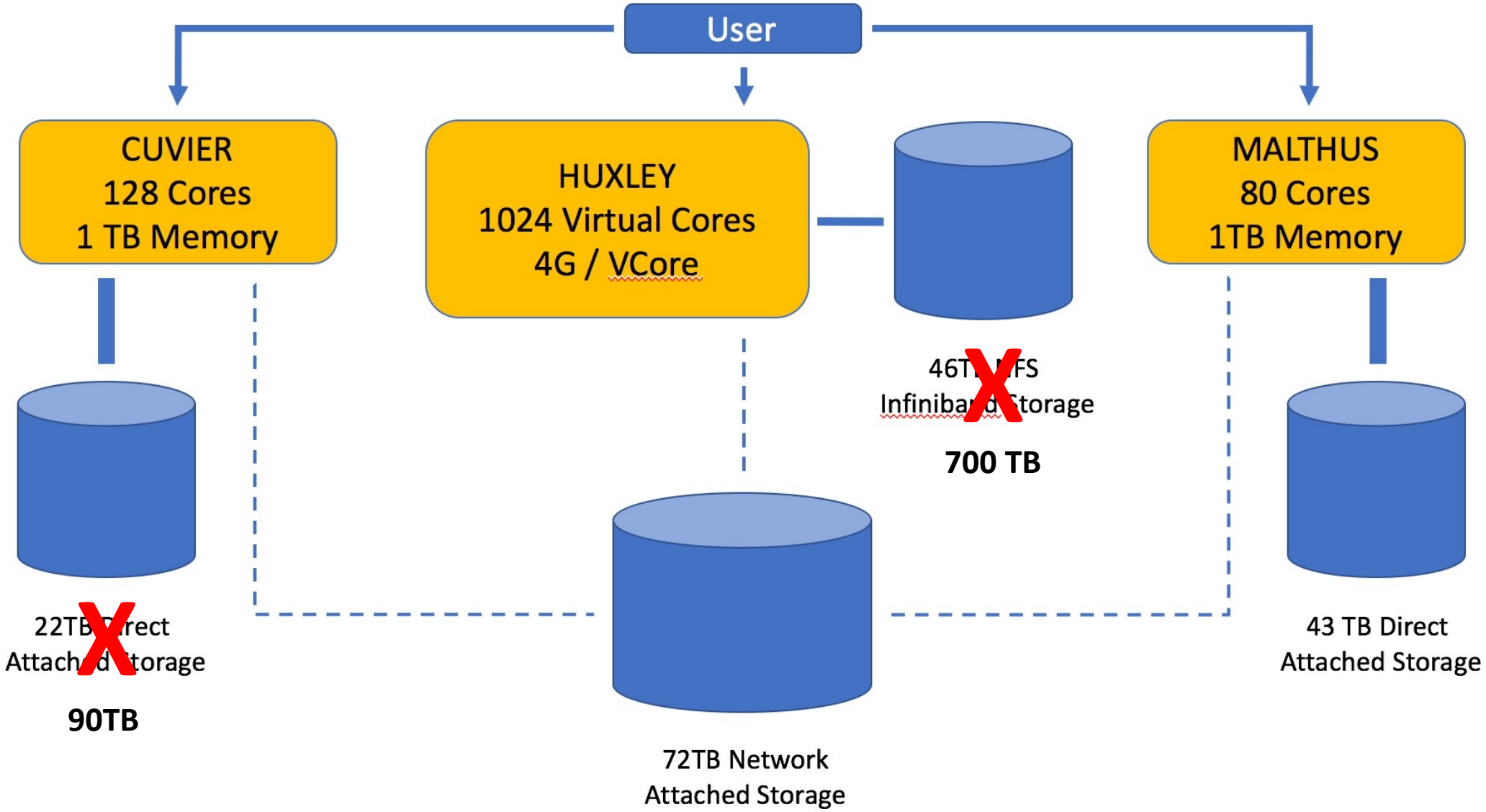
Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)
3. More Talking
    * Where does genomics fit in big data science?

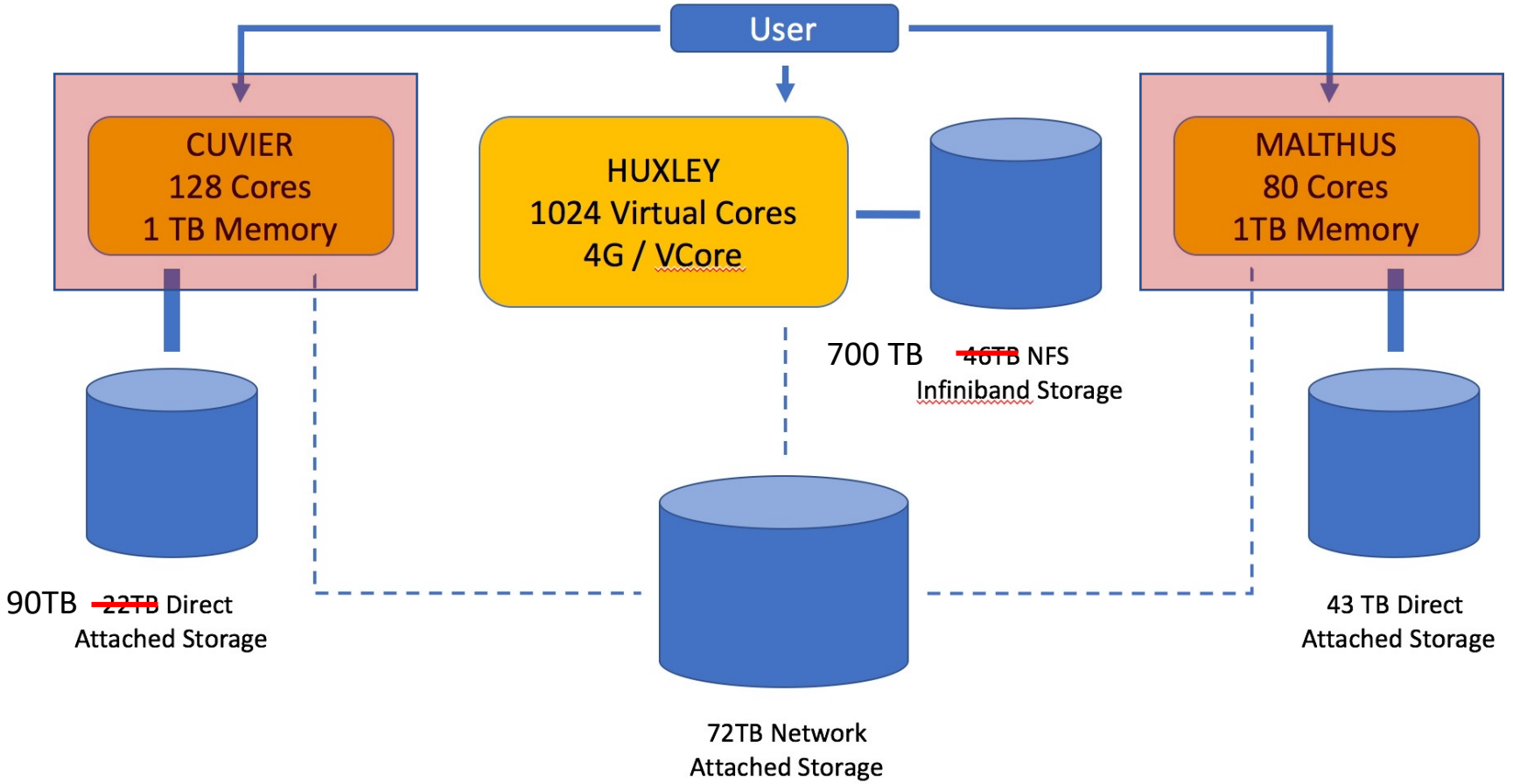
Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)
3. More Talking
    * Where does genomics fit in big data science?
4. More Doing (BONUS, if we make it)
    * GNU Parallel
    * auto-generating PBS scripts for complex workflows

Sackler Genomics Compute Resources Overview

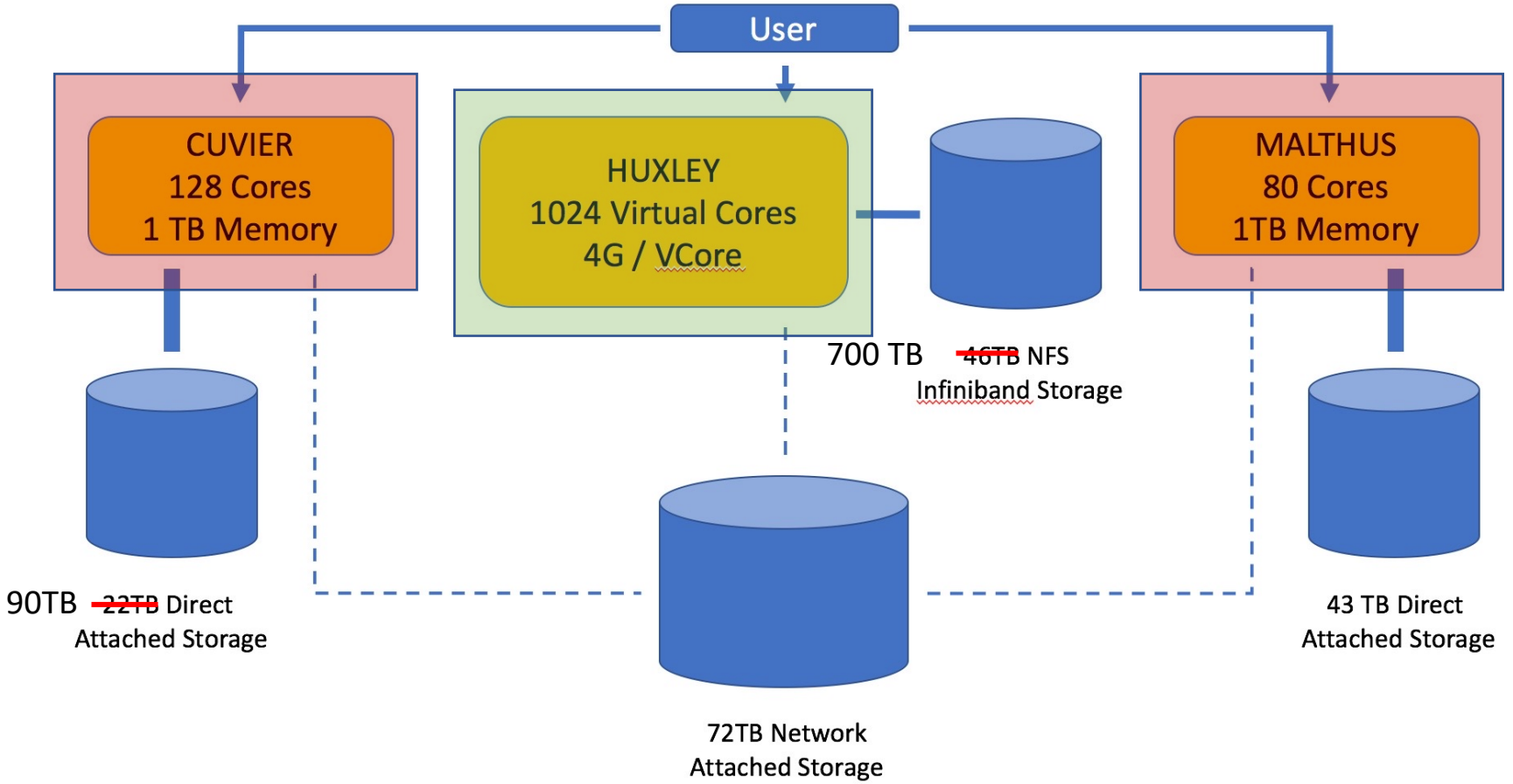


Shared Memory Server



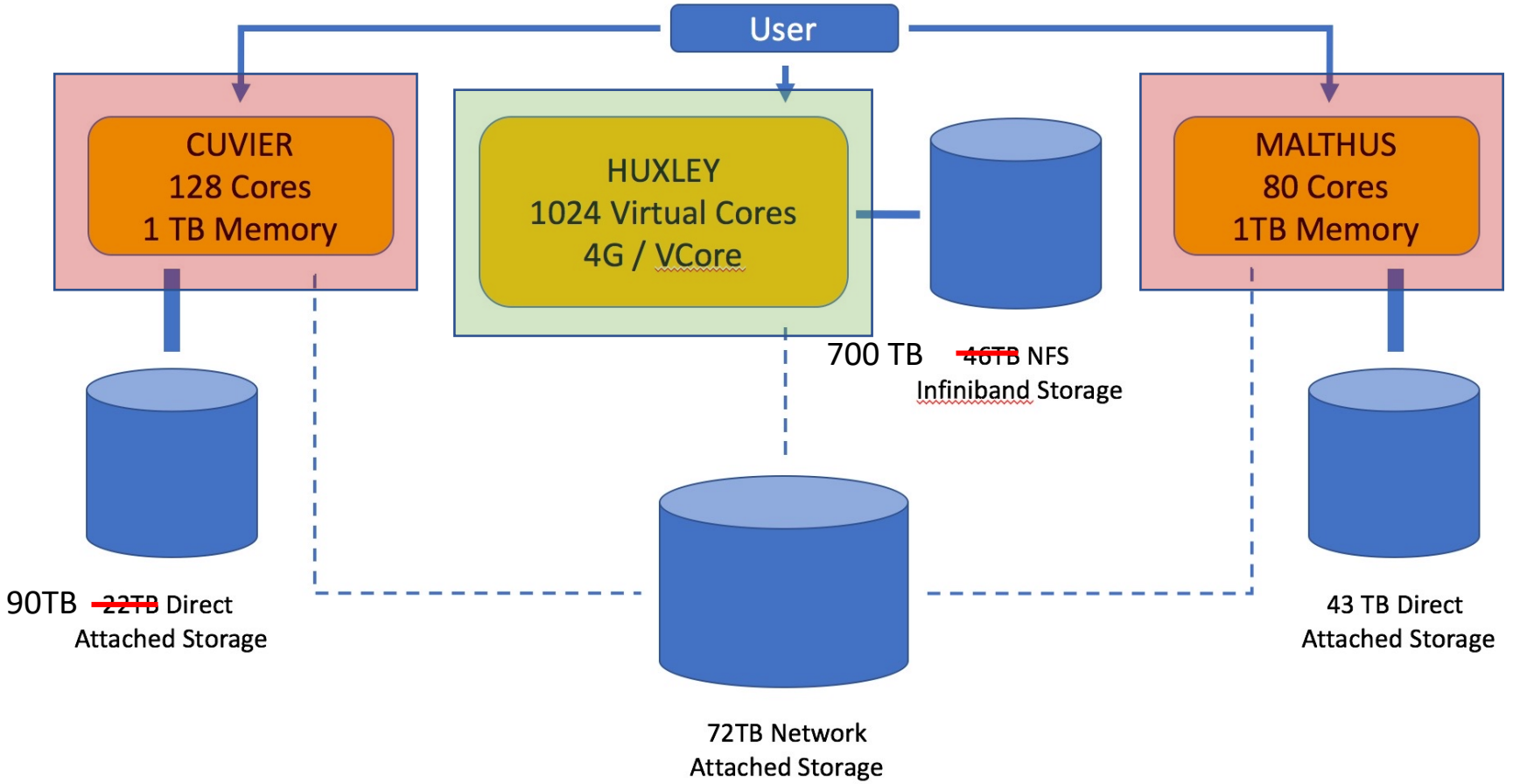
High Performance Compute Cluster

Shared Memory Server



High Performance Compute Cluster

Shared Memory Server good for memory intensive assemblies, etc



High Performance Compute Cluster good for CPU intensive blasts, etc

CPU

Shared Memory Cuvier

- 1 node
- 64 CPUs
- 128 Hyperthreaded cores
- 1TB/node

HPC Huxley

- 16 nodes
- 512 CPUs
- 1024 Hyperthreaded cores
- 256GB / node

CPU

Shared Memory Cuvier

- 1 node
- 64 CPUs
- 128 Hyperthreaded cores
- 1TB/node

PBS quotas

- AMNH: 16 cores
- Visiting: 8 cores

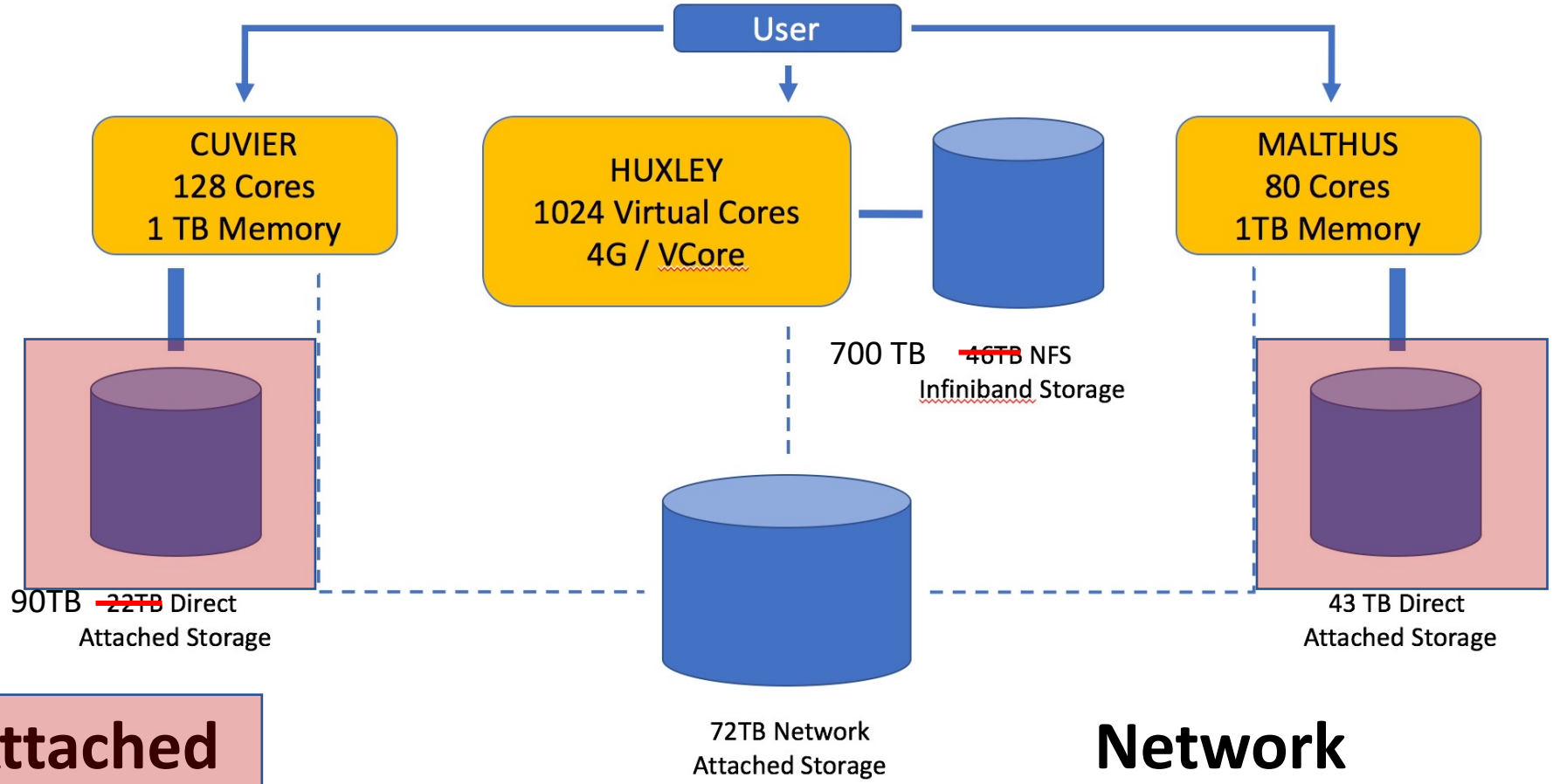
HPC Huxley

- 16 nodes
- 512 CPUs
- 1024 Hyperthreaded cores
- 256GB / node

PBS quotas

- AMNH: 128 cores
- Visiting: 32 cores

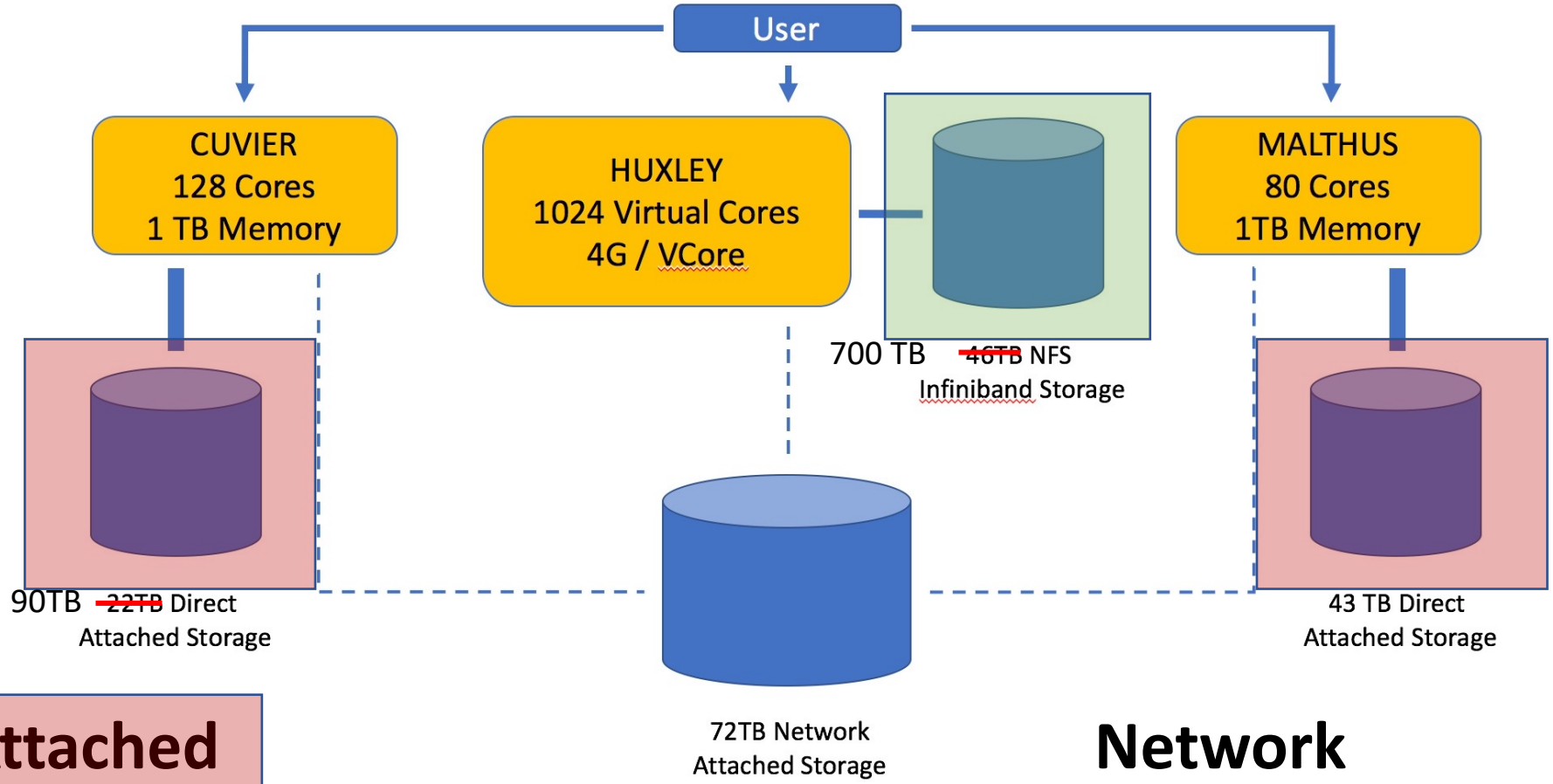
HPC Network File System: Fast for a network (Infinband)



Direct Attached Storage: as fast as you can get

Network Attached Storage: slow but large

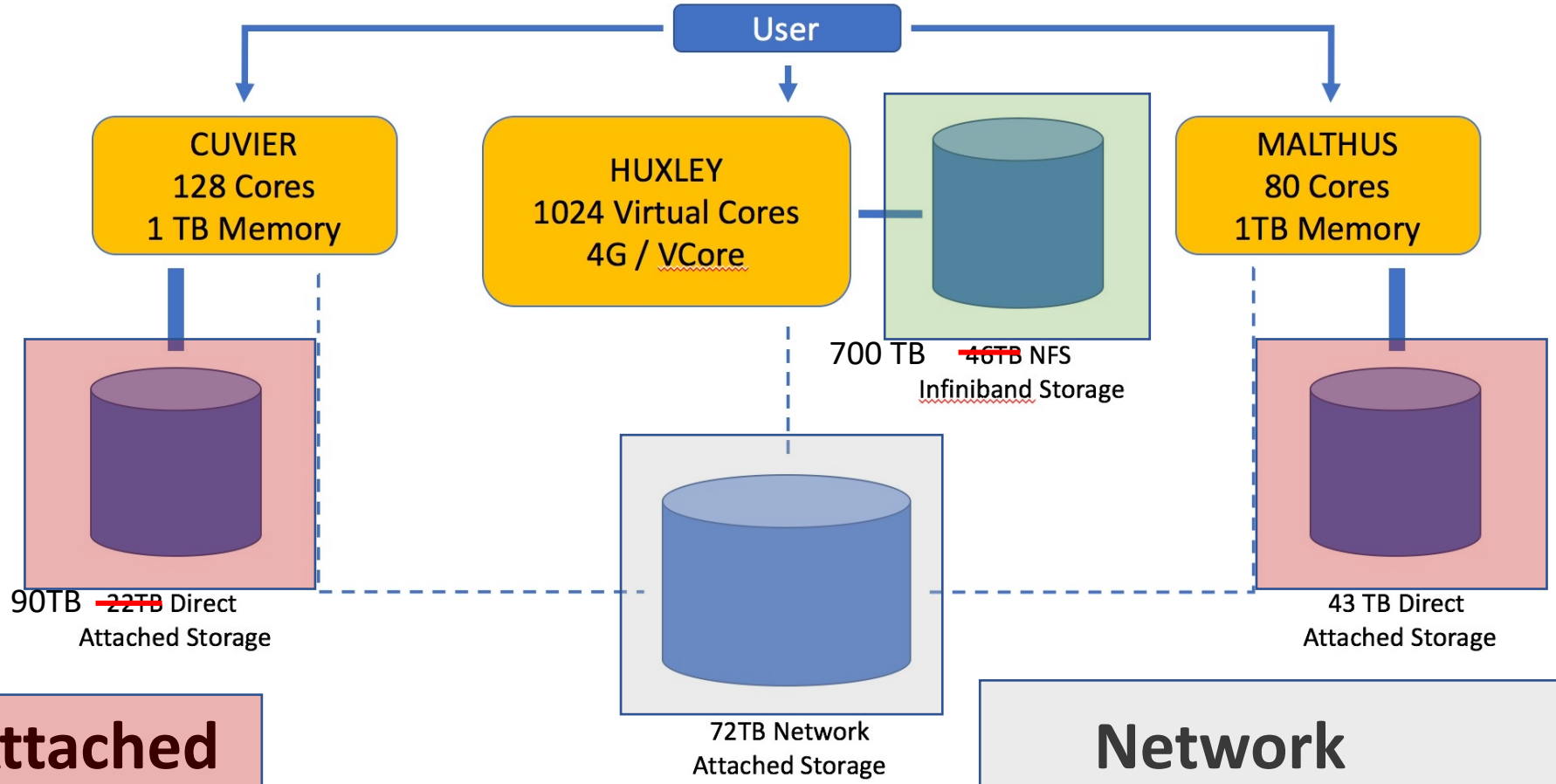
HPC Network File System: Fast for a network (Infinband)



Direct Attached Storage: as fast as you can get

Network Attached Storage: slow but large

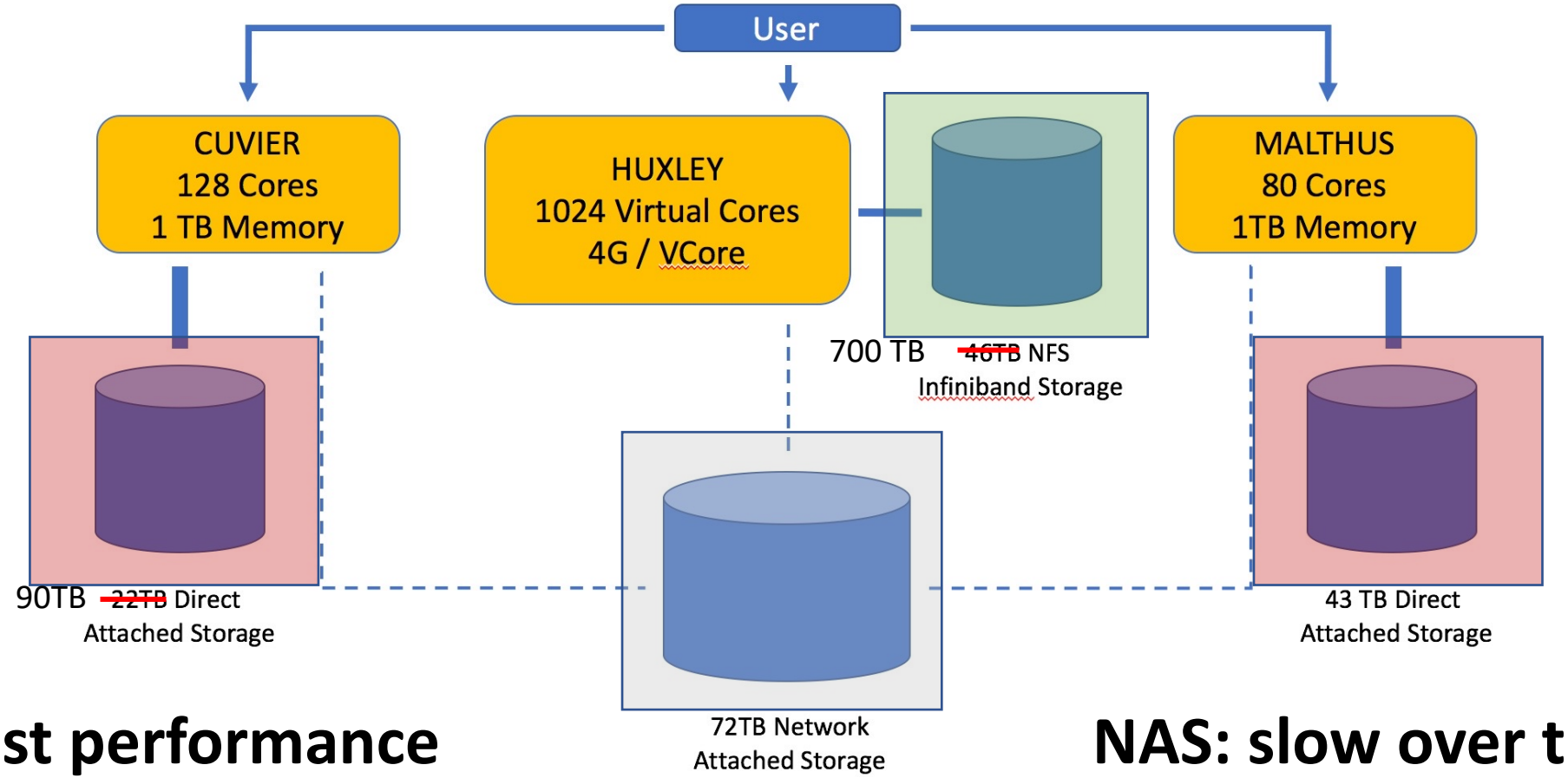
HPC Network File System: Fast for a network (Infinband)



Direct Attached Storage: as fast as you can get

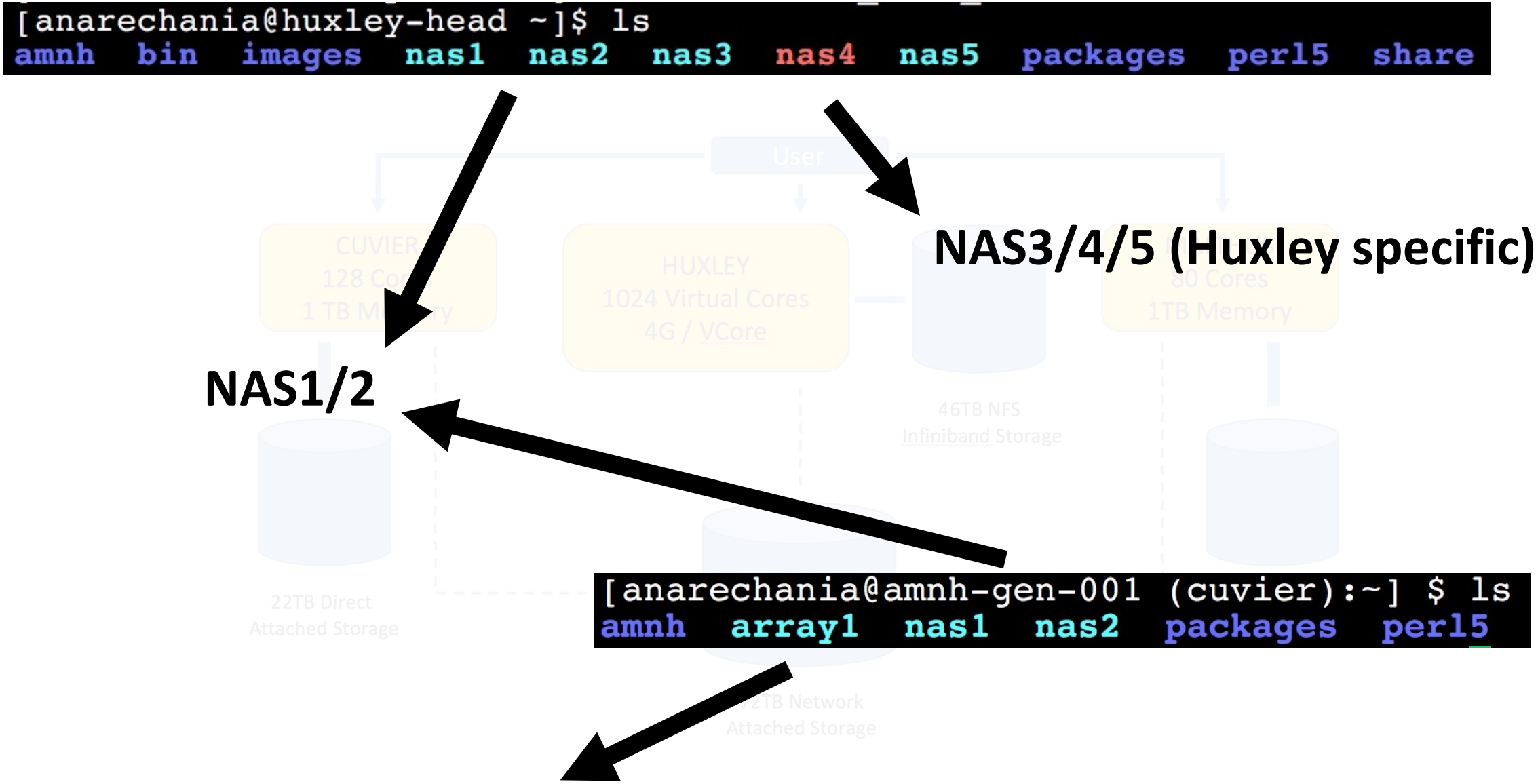
Network Attached Storage: slow but large

HPC NFS: All data for Huxley HPC jobs must be available here



DAS: Best performance for manipulating large BAM/fastq files

NAS: slow over the network. Ok for small reconstructions / blasts


[anarechania@huxley-head ~]$ ls
amnh bin images nas1 nas2 nas3 nas4 nas5 packages perl5 share
NAS3/4/5 (Huxley specific)
NAS1/2
[anarechania@amnh-gen-001 (cuvier):~] $ ls
amnh array1 nas1 nas2 packages perl5
Array1 (Cuvier Specific)

Storage

NAS1/2

- 72TB RAID storage

PBS quotas

- AMNH: 500G
- Visiting: 500G

NAS3/4/5 (Huxley)

- 700TB RAID storage
- 128GB SSD / node

PBS quotas

- AMNH: 250G
- Visiting: 250G

Array1 (Cuvier)

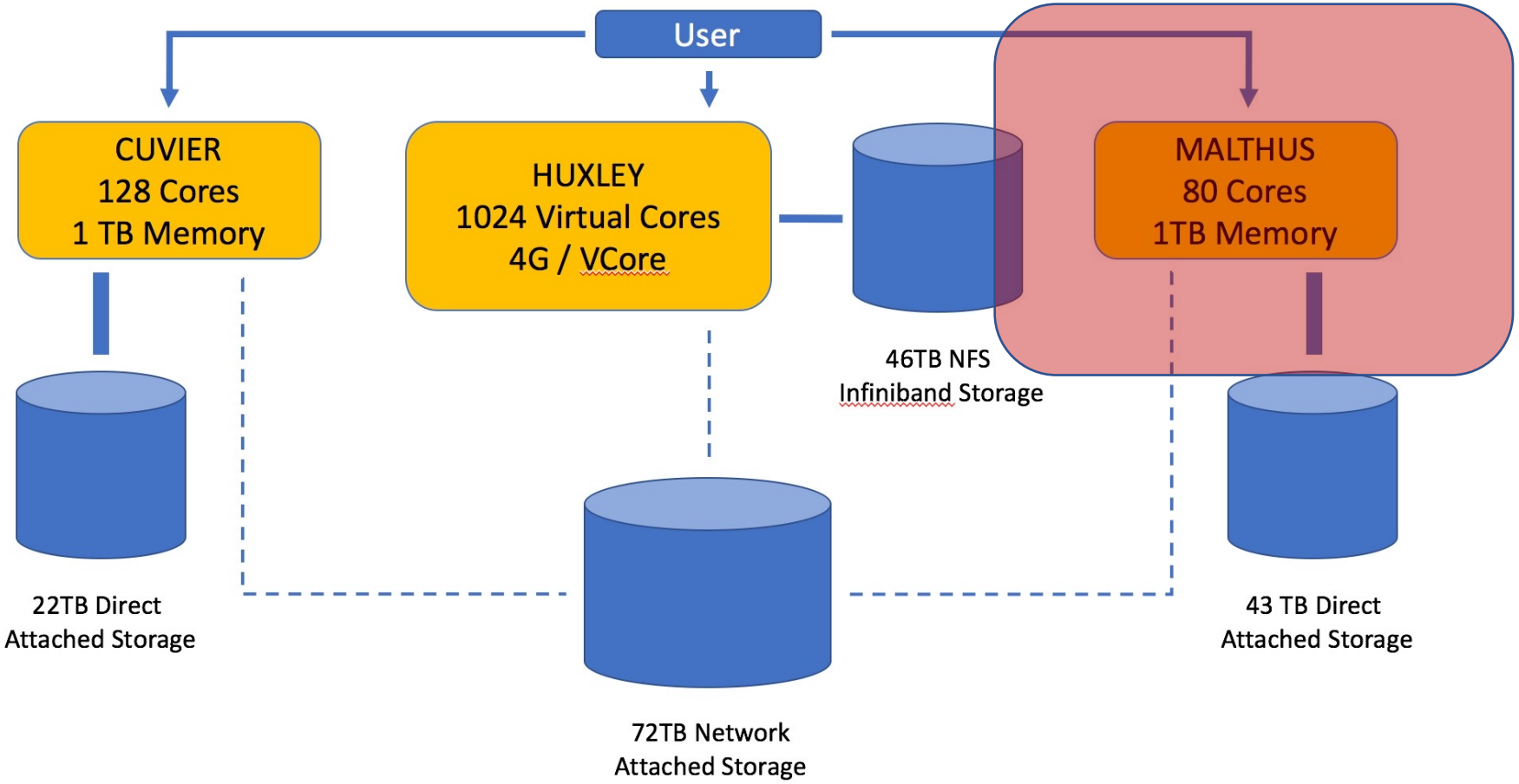
- 92T Direct Attached Storage

Software on Huxley: login and look

```
$> ssh anarechania@huxley-master.pcc.amnh.org

$> module avail

            A LOT

$> which raxml
$> module load RAxML-8.2.11
$> which raxml
```

Can make permanent by adding a load statement to .bashrc; let's try…

What about malthus?



Malthus is essentially a free for all

It’s unscheduled, and has no CPU or disk space quotas.

Development box

```
[anarechania@malthus ~]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_malthus-lv_root
                       50G   20G   28G  42% /
tmpfs                 505G  8.0K  505G   1% /dev/shm
/dev/sda1             477M   67M  385M  15% /boot
/dev/mapper/vg_malthus-LogVol03
                      2.0T  1.3T  648G  67% /data
/dev/mapper/vg_malthus-lv_home
                      493G  393G   75G  84% /home
/dev/mapper/vg_malthus-LogVol04
                       11T  8.2T  1.4T  87% /scratch
/dev/sdb1             9.7T  8.6T  668G  93% /array
gen-nas.internal.amnh.org:/nas1
                       36T   21T   16T  57% /nas1
gen-nas.internal.amnh.org:/nas2
                       36T   11T   26T  29% /nas2
```

Malthus is essentially a free for all

It’s unscheduled, and has no CPU or disk space quotas.

Development box

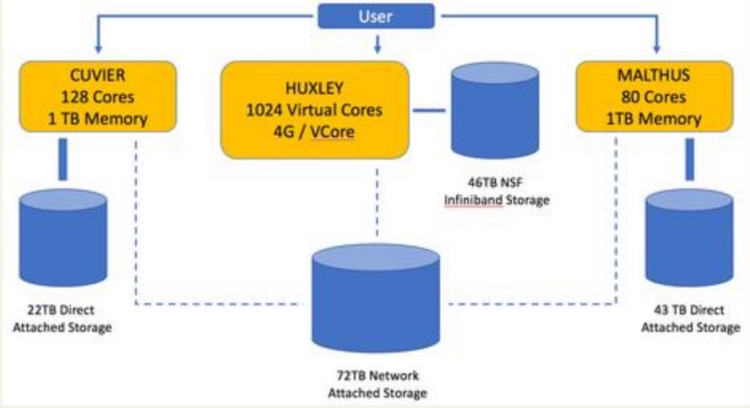
```
[anarechania@malthus ~]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_malthus-lv_root
                       50G   20G   28G  42% /
tmpfs                 505G  8.0K  505G   1% /dev/shm
/dev/sda1             477M   67M  385M  15% /boot
/dev/mapper/vg_malthus-LogVol03
                      2.0T  1.3T  648G  67% /data
/dev/mapper/vg_malthus-lv_home
                      493G  393G   75G  84% /home
/dev/mapper/vg_malthus-LogVol04
                       11T  8.2T  1.4T  87% /scratch
/dev/sdb1             9.7T  8.6T  668G  93% /array
gen-nas.internal.amnh.org:/nas1
                       36T   21T   16T  57% /nas1
gen-nas.internal.amnh.org:/nas2
                       36T   11T   26T  29% /nas2
```

http://museum.internal.amnh.org/Sackler-Institute-for-Comparative-Genomics/SICG-Bioinformatics

SICG Bioinformatics

Welcome to the home for bioinformatics at SICG. We collaborate with a number of research groups across the museum on projects with a genomic and/or computational component. We also offer workshops on bioinformatics best practices and consult with researchers on their computational problems. One of our major goals is to provide a significant compute infrastructure open to much of the museum research community.

Please explore what we have to offer, and feel free to get in touch.



THE BE NICE RULES:

1. Do NOT park your data on Sackler machines
    * Working storage only!
    * Remove intermediate files! (SAM/BAM, etc)
    * NOT BACKED UP!

THE BE NICE RULES:

1. Do NOT park your data on Sackler machines
    * Working storage only!
    * Remove intermediate files! (SAM/BAM, etc)
    * NOT BACKED UP!
2. Do NOT overload PBS
    * Design each job to take at least 10 minutes
    * Use job arrays!

THE BE NICE RULES:

1. Do NOT park your data on Sackler machines
    * Working storage only!
    * Remove intermediate files! (SAM/BAM, etc)
    * NOT BACKED UP!
2. Do NOT overload PBS
    * Design each job to take at least 10 minutes
    * Use job arrays!
3. Do NOT overload I/O on NAS1/2/3/4/5
    * On Huxley if infiniband NFS is too slow, copy data to local SSD
    * On Cuvier/Malthus run I/O intensive apps from DAS not NAS.

THE BE NICE RULES:

1. Do NOT park your data on Sackler machines
    * Working storage only!
    * Remove intermediate files! (SAM/BAM, etc)
    * NOT BACKED UP!
2. Do NOT overload PBS
    * Design each job to take at least 10 minutes
    * Use job arrays!
3. Do NOT overload I/O on NAS1/2/3
    * On Huxley if infiniband NFS is too slow, copy data to local SSD
    * On Cuvier/Malthus run I/O intensive apps from DAS not NAS.
4. Think hard about what you're doing!
    * Memory intensive apps like assembly → Cuvier/Malthus
    * CPU intensive → Huxley
    * Consider whether your threaded programs work well with hyperthreads; otherwise count 1 physical core for every 2 hyperthreaded cores.

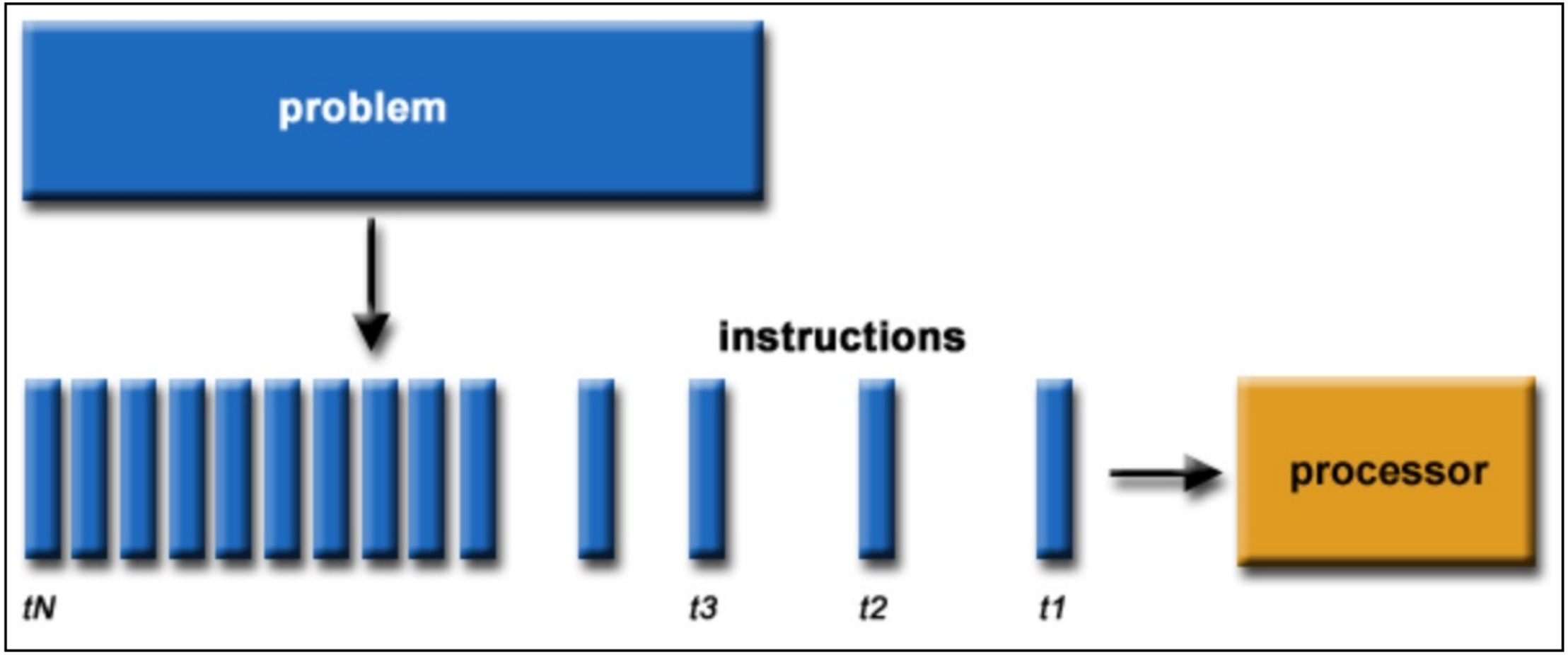
What is Parallel Computing?

What is Parallel Computing?

The only goal in parallel computing is to bring down the WALLCLOCK.

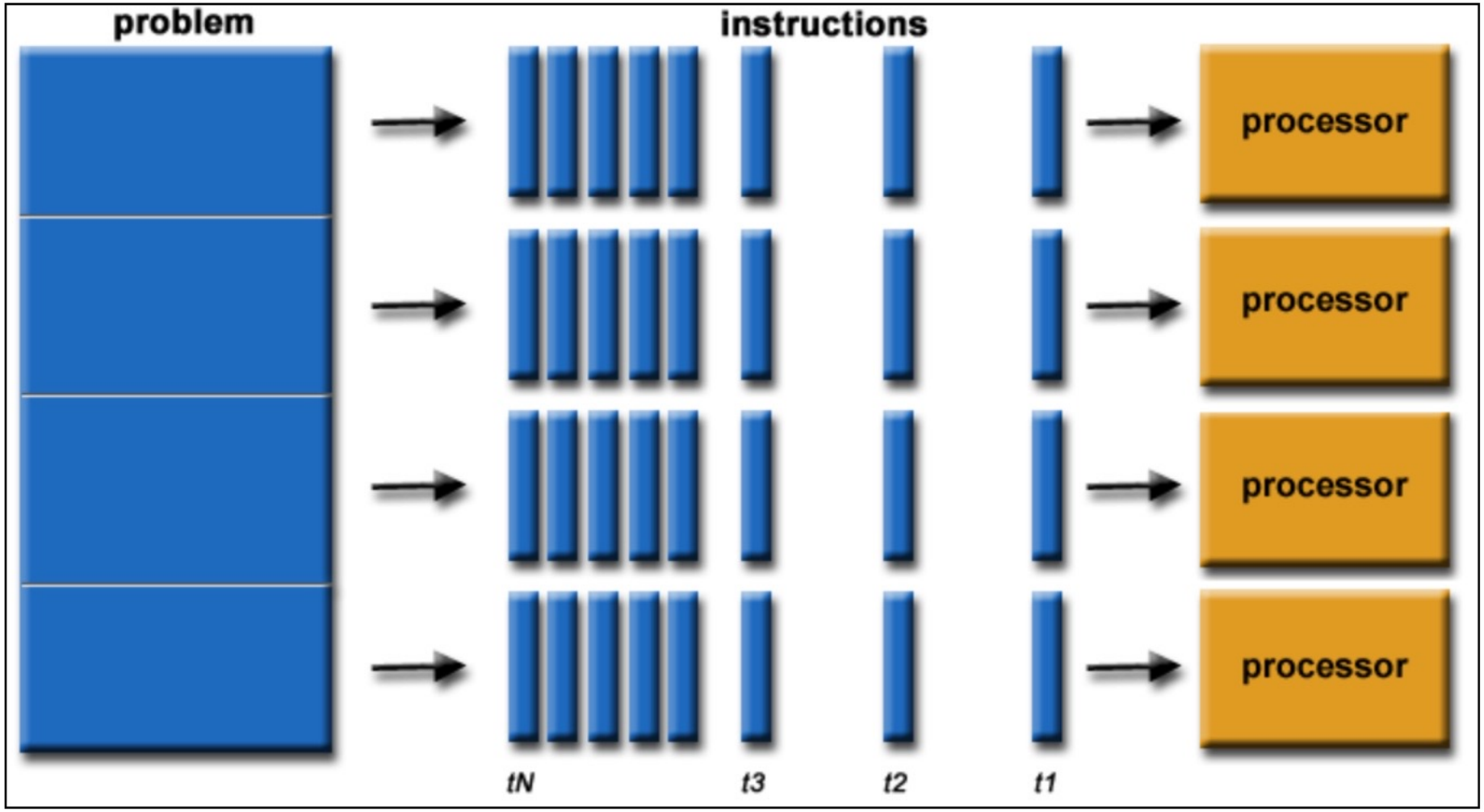
WALLCLOCK = The total amount of time your job runs regardless of what resources it is running on.

This is serial…



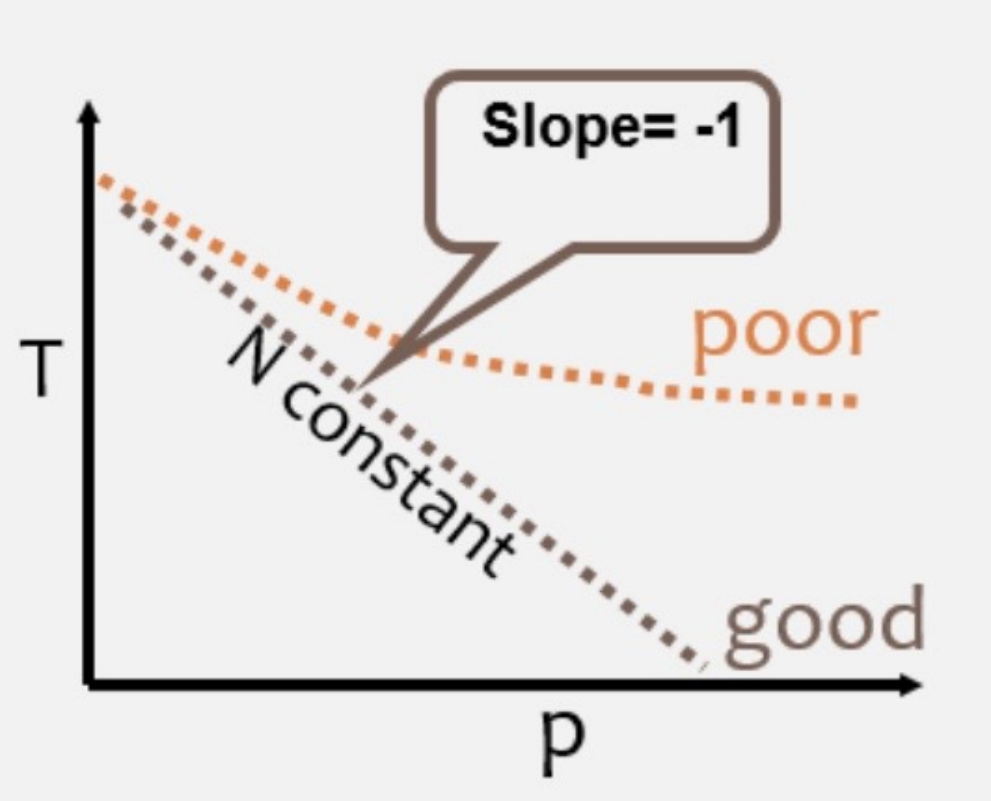
Adapted from Blaise Barney: https://computing.llnl.gov/tutorials/parallel_comp/

Concurrency: a problem split onto separate processors running at the same time



Concurrency: a problem split onto separate processors running at the same time

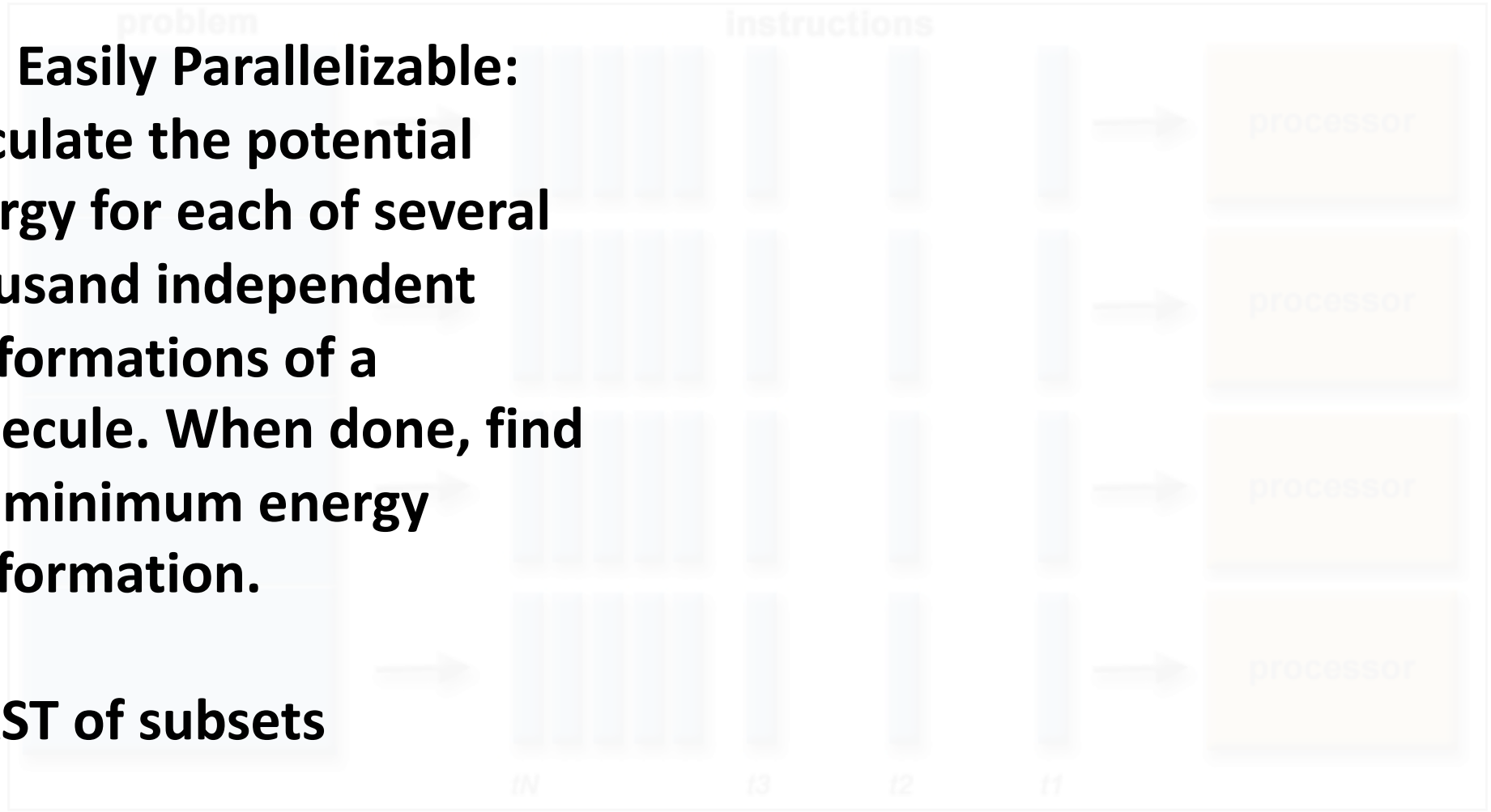
Scalability: does adding processors give you a proportional speedup in compute time?



Does your problem suffer from data dependence?

Easily Parallelizable:
Calculate the potential energy for each of several thousand independent conformations of a molecule. When done, find the minimum energy conformation.

BLAST of subsets



Does your problem suffer from data dependence?

Easily Parallelizable:

Calculate the potential energy for each of several thousand independent conformations of a molecule. When done, find the minimum energy conformation.

BLAST of subsets

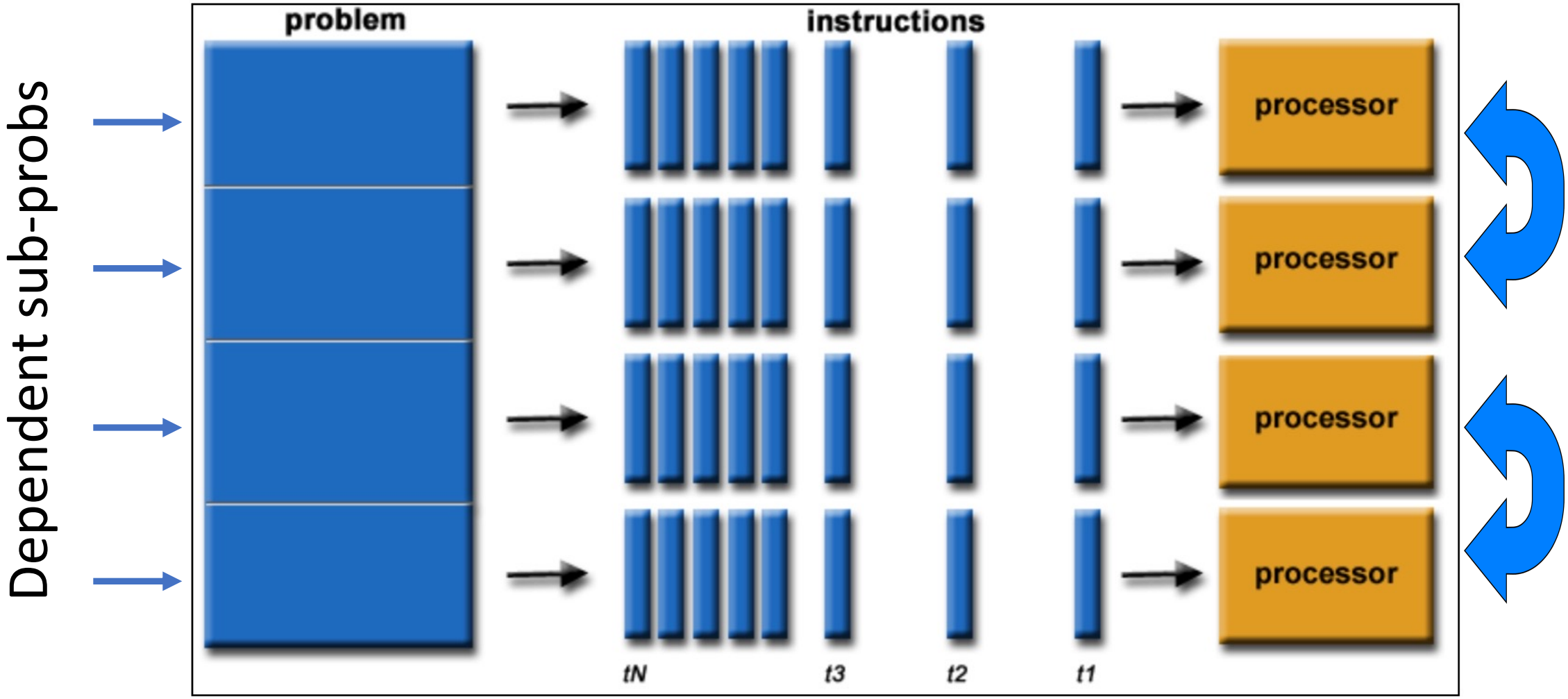
Impossible:

Calculation of the Fibonacci series (0,1,1,2,3,5,8,13,21,...) by use of the formula:
F(n) = F(n-1) + F(n-2)

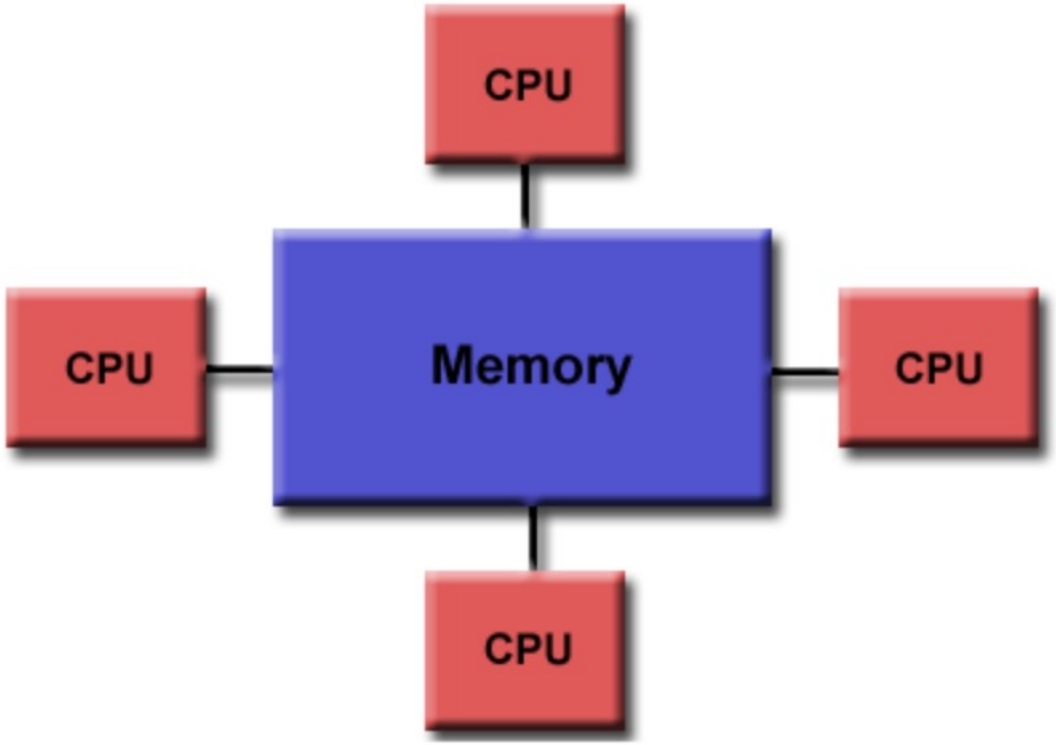
F(n) requires the prior calculation of F(n-1) and F(n-2)

The shortest path thru a DeBruijn graph.

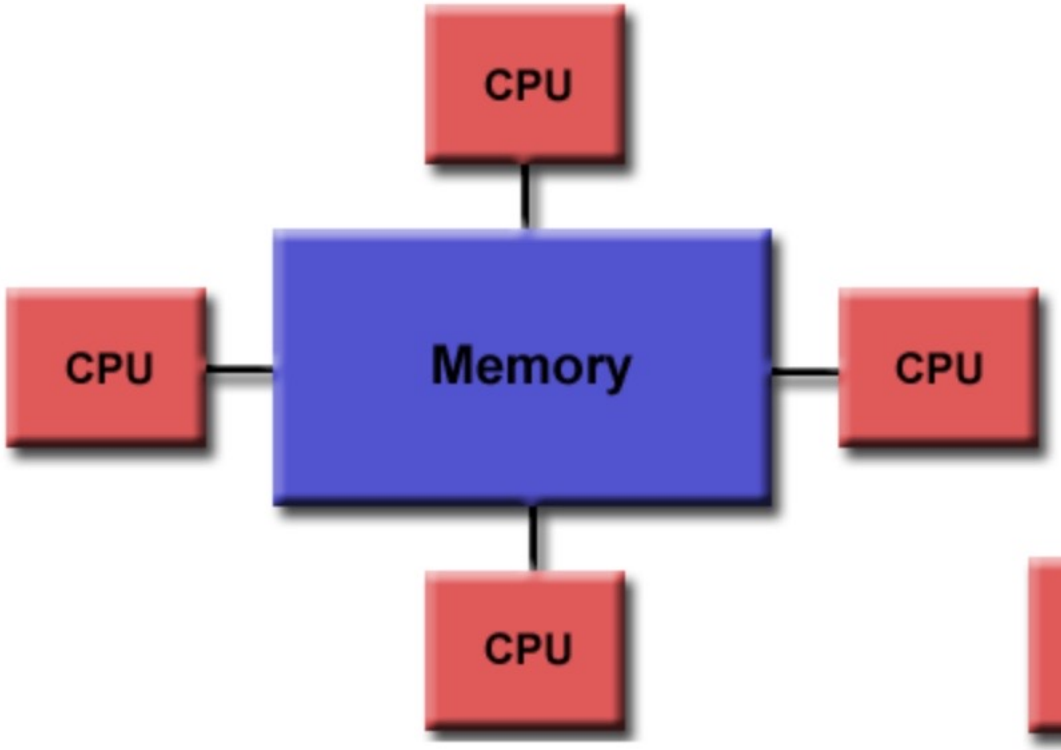
Often, sub-problems are dependent and need to update each other…

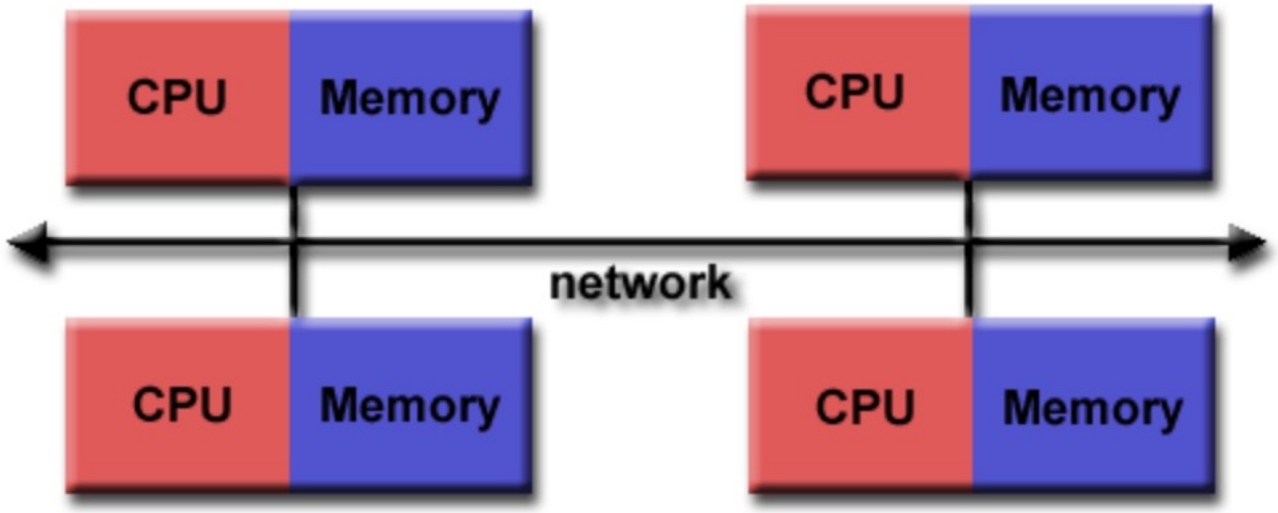


This can happen using threads if the architecture is shared memory…

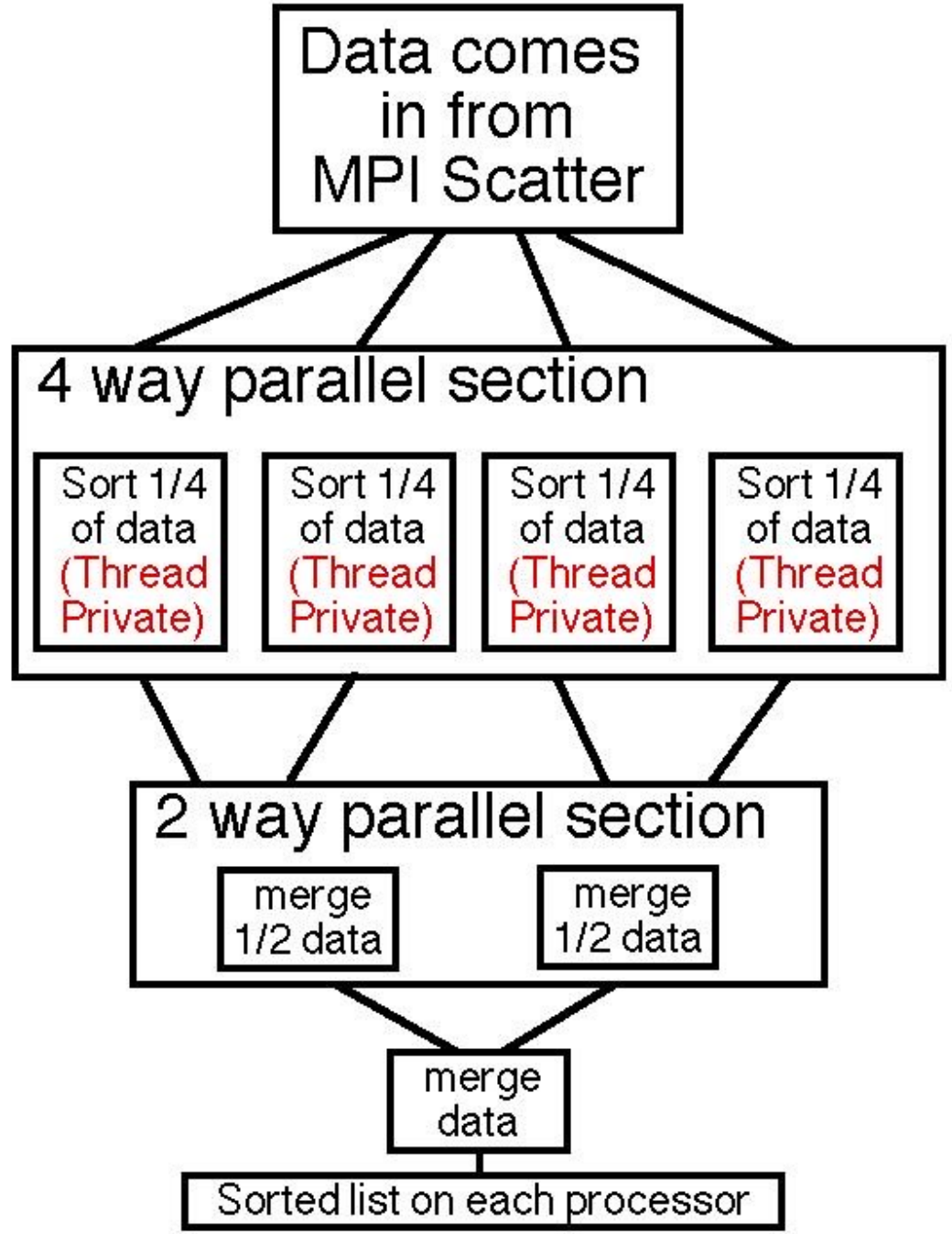


Or using MPI if the architecture is distributed as in an HPC…



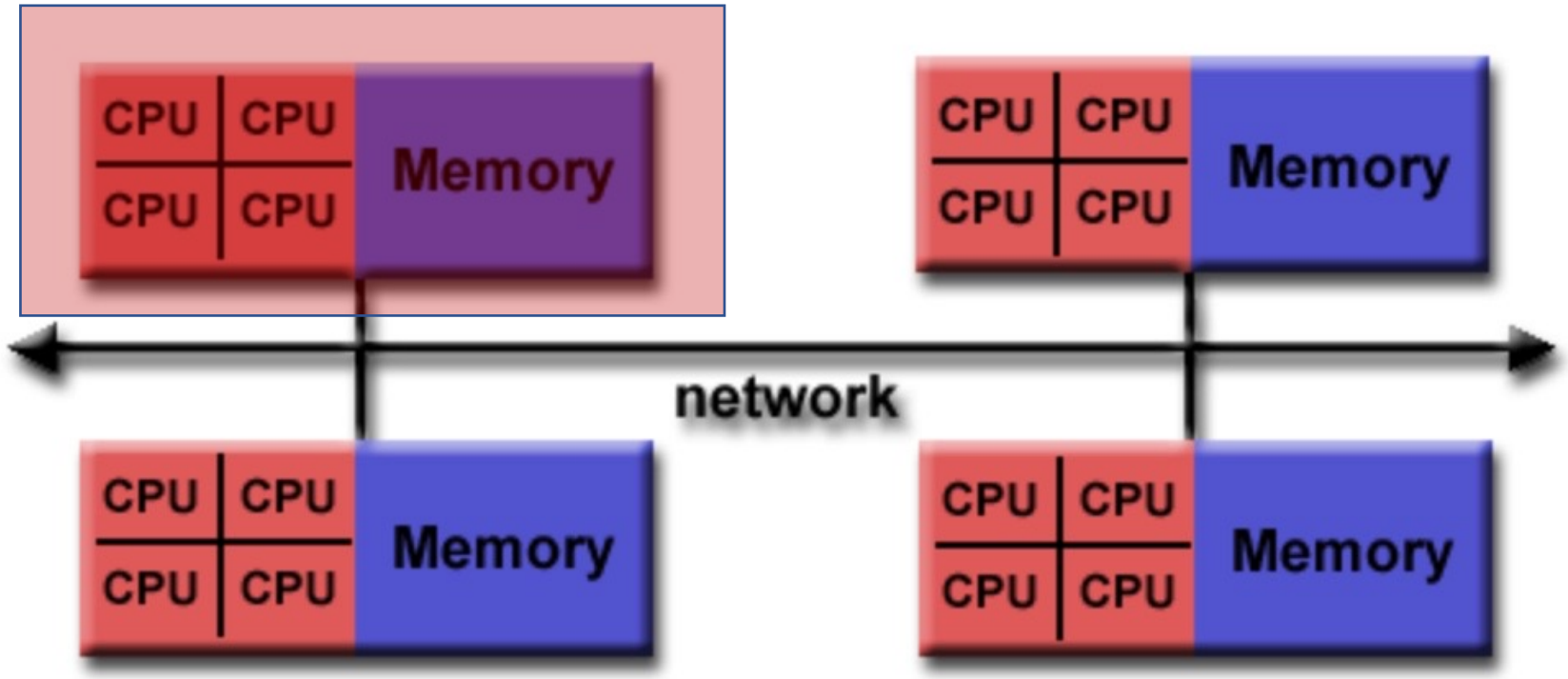


MPI Enabled Parallel Merge Sort



Generally True for All Parallel Computing:

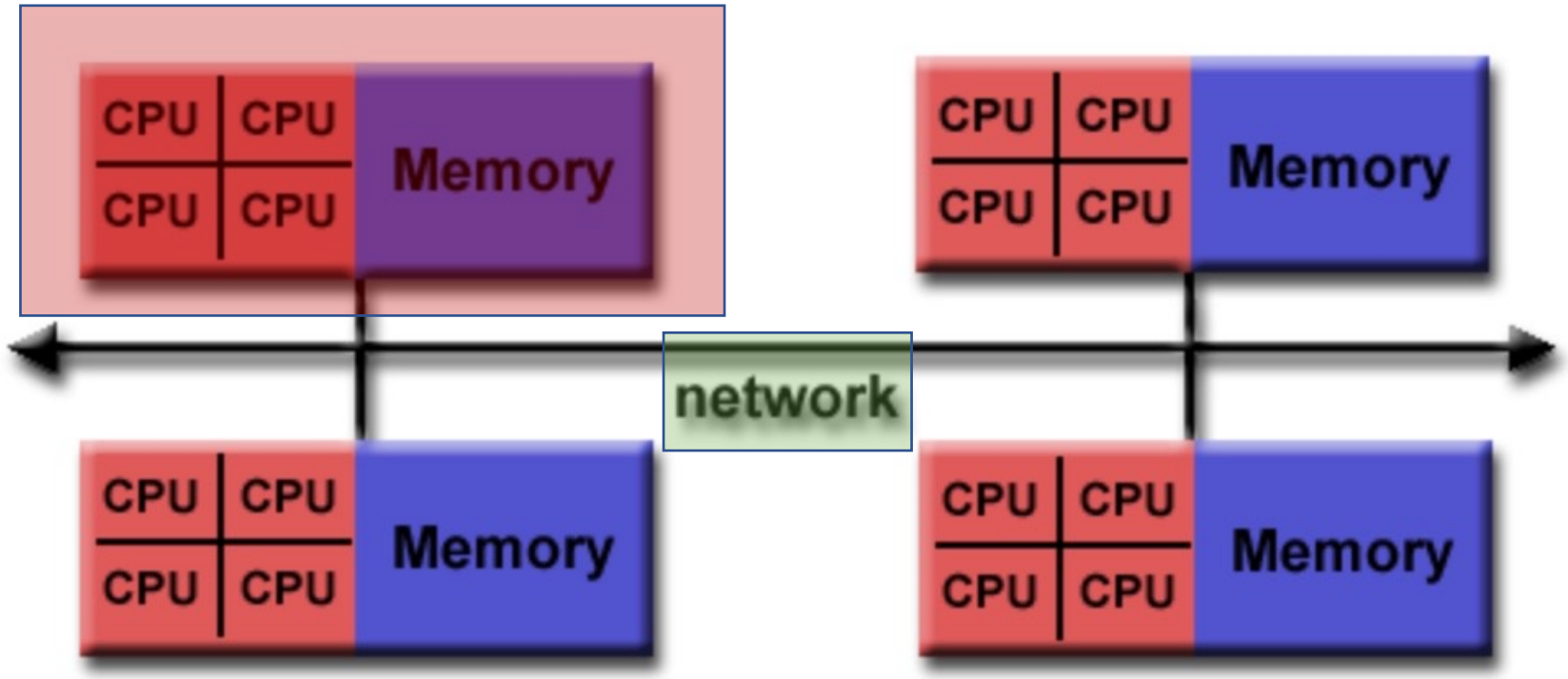
CPU is FAST



Generally True for All Parallel Computing:

CPU is FAST

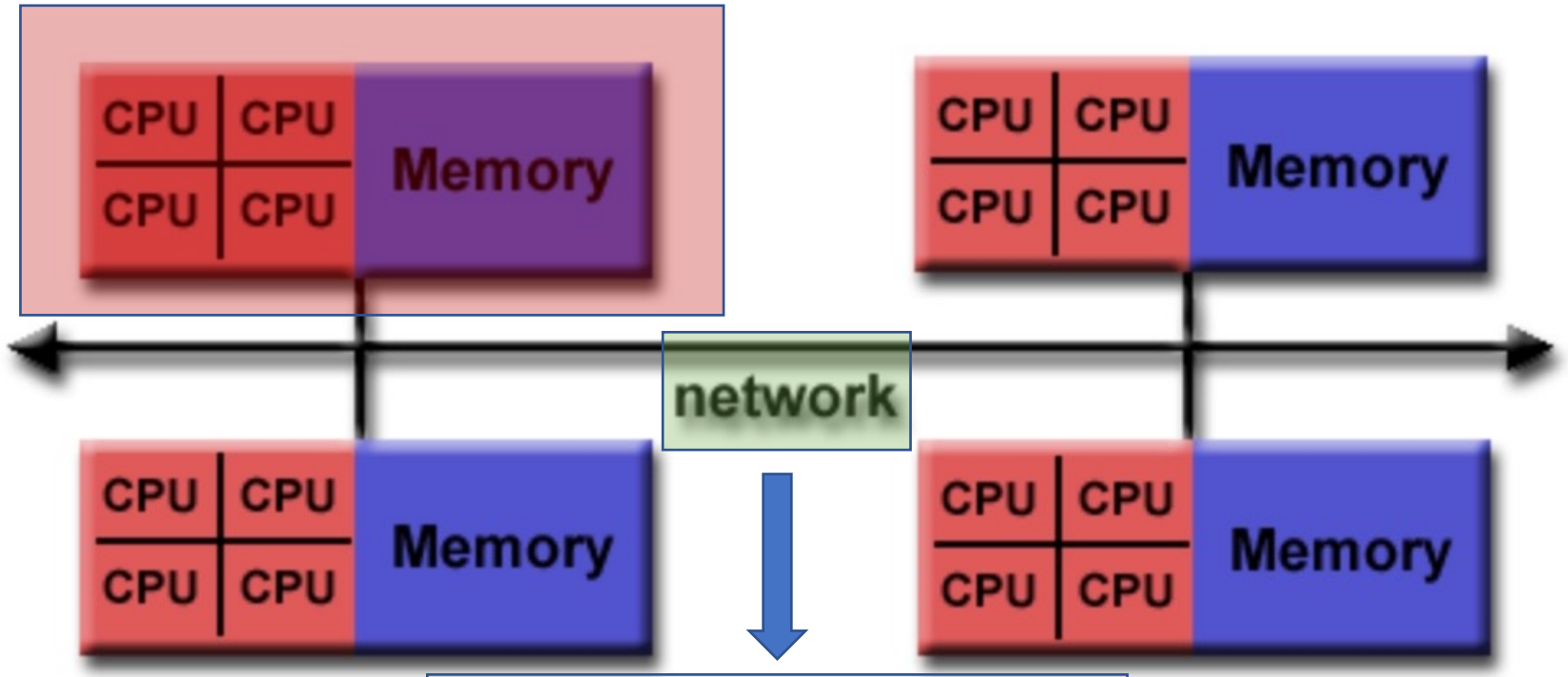
Communication is slow



Generally True for All Parallel Computing:

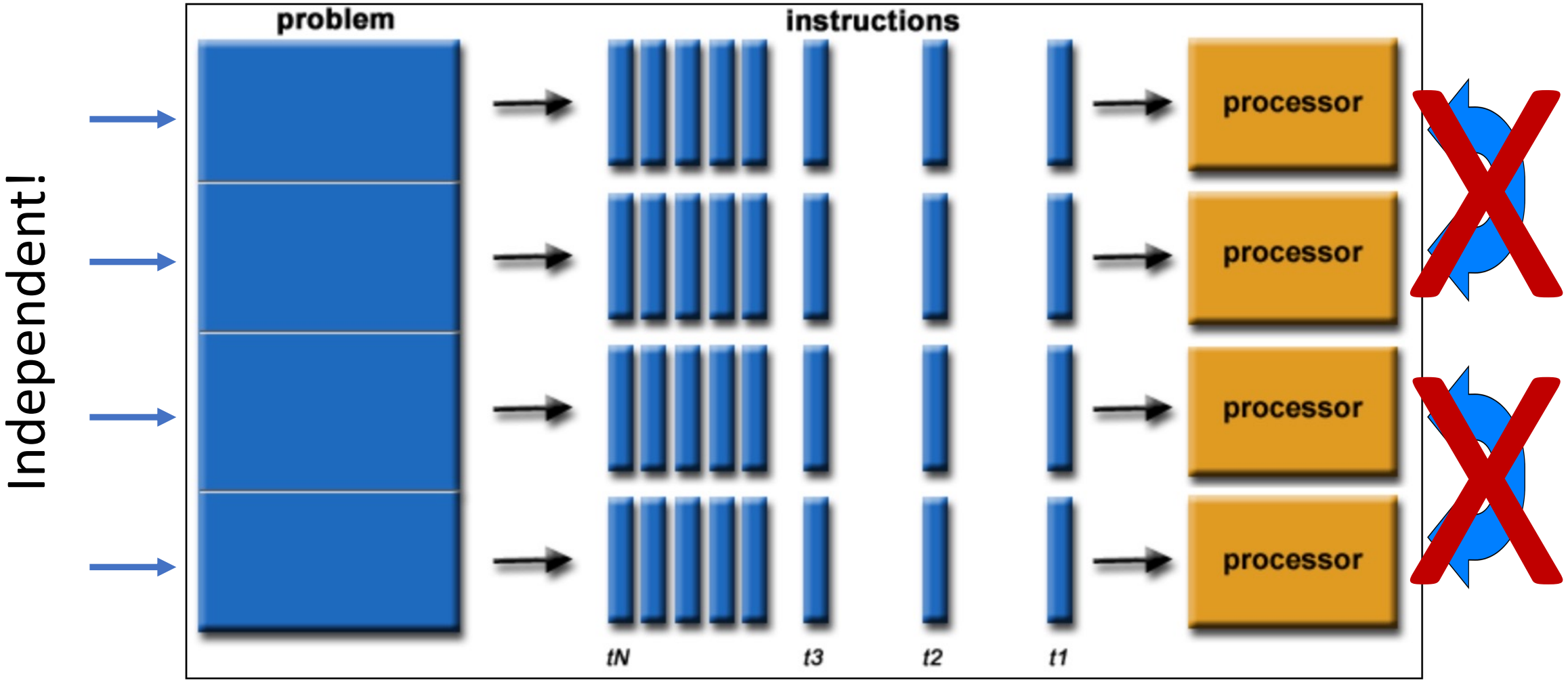
CPU is FAST

Communication is slow



I/O to Disk is so so so slow

If all sub-problems are independent, you're in luck!: Embarrassingly Parallel.



If all sub-problems are independent, you're in luck!:
Embarrassingly Parallel.

Bioinformatics is FILLED with embarrassingly parallel problems!

Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)
3. More Talking
    * Where does genomics fit in big data science?
    * Specific QA with Sajesh Singh
4. More Doing (BONUS, if we make it)
    * GNU Parallel
    * auto-generating PBS scripts for complex workflows

A prime example is blast. Setup for blast array job…

We’re going to blast all the dicty transcripts against the human genome!

Why?

A prime example is blast. Setup for blast array job…

```
$> cp -r /opt/AMNH/cluster/sample_scripts/hpc_class/ /home/anarechania/nas3
$> cd /home/anarechania/nas3
$> cd blast_exercise
$> module load ncbi-blast-2.6.0+
$> makeblastdb -in GRCh38_latest_genomic.fna -parse_seqids -dbtype nucl
```

Now we need to write a PBS script to submit our blast to the cluster.
But what is PBS???

PBS (Portable Batch Scheduler)

- A scheduler keeps a cluster from getting overwhelmed
- Designed to allot resources to incoming jobs in a fair way (fair share) given user quotas and privileges

PBS (Portable Batch Scheduler)

- A scheduler keeps a cluster from getting overwhelmed
- Designed to allot resources to incoming jobs in a fair way (fair share) given user quotas and privileges

Now I’m going to show you three bewildering tables. Don’t worry!

We’ll highlight only the few things you really need.

Table 1: Basic PBS Commads

Job Control

- qsub: Submit a job
- qdel: Delete a batch job
- qsig: Send a signal to batch job
- qhold: Hold a batch job
- qrls: Release held jobs
- qrerun: Rerun a batch job
- qmove: Move a batch job to another queue

Job Monitoring

- qstat: Show status of batch jobs
- qselect: Select a specific subset of jobs

Node Status

- pbsnodes: List the status and attributes of all nodes in the cluster.

Miscellaneous

- qalter: Alter a batch job
- qmsg: Send a message to a batch job

Table 1: Basic PBS Commads

Job Control

- qsub: Submit a job → **Submit your job!**
- qdel: Delete a batch job → **Delete your job!**
- qsig: Send a signal to batch job
- qhold: Hold a batch job
- qrls: Release held jobs
- qrerun: Rerun a batch job
- qmove: Move a batch job to another queue

Job Monitoring

- qstat: Show status of batch jobs → **See what everyone's up to.**
- qselect: Select a specific subset of jobs

Let's try: qstat; qstat -a; qstat -f <jobid>

Node Status

- pbsnodes: List the status and attributes of all nodes in the cluster.

Miscellaneous

- qalter: Alter a batch job
- qmsg: Send a message to a batch job

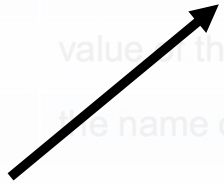
Table 2: PBS Variables. Stuff you can use in your submit scripts.

Variable	Description
$PBS_ENVIRONMENT	set to PBS_BATCH to indicate that the job is a batch job; otherwise, set to PBS_INTERACTIVE to indicate that the job is a PBS interactive job
$PBS_JOBID	the job identifier assigned to the job by the batch system
$PBS_JOBNAME	the job name supplied by the user
$PBS_NODEFILE	the name of the file that contains the list of nodes assigned
$PBS_QUEUE	the name of the queue from which the job is executed
$PBS_O_HOME	value of the HOME variable in the environment in which qsub was executed
$PBS_O_LANG	value of the LANG variable in the environment in which qsub was executed
$PBS_O_LOGNAME	value of the LOGNAME variable in the environment in which qsub was executed
$PBS_O_PATH	value of the PATH variable in the environment in which qsub was executed
$PBS_O_MAIL	value of the MAIL variable in the environment in which qsub was executed
$PBS_O_SHELL	value of the SHELL variable in the environment in which qsub was executed
$PBS_O_TZ	value of the TZ variable in the environment in which qsub was executed
$PBS_O_HOST	the name of the host upon which the qsub command is running
$PBS_O_QUEUE	the name of the original queue to which the job was submitted
$PBS_O_WORKDIR	the absolute path of the current working directory of the qsub command

Table 2: PBS Variables. Stuff you can use in your submit scripts.

Variable	Description
$PBS_ENVIRONMENT	set to PBS_BATCH to indicate that the job is a batch job; otherwise, set to PBS_INTERACTIVE to indicate that the job is a PBS interactive job
$PBS_JOBID	the job identifier assigned to the job by the batch system
$PBS_JOBNAME	the job name supplied by the user
$PBS_NODEFILE	the name of the file that contains the list of nodes assigned
$PBS_QUEUE	the name of the queue from which the job is executed
$PBS_O_HOME	value of the HOME variable in the environment in which qsub was executed
$PBS_O_LANG	value of the LANG variable in the environment in which qsub was executed
$PBS_O_LOGNAME	value of the LOGNAME variable in the environment in which qsub was executed
$PBS_O_PATH	value of the PATH variable in the environment in which qsub was executed
$PBS_O_MAIL	value of the MAIL variable in the environment in which qsub was executed
$PBS_O_SHELL	value of the SHELL variable in the environment in which qsub was executed
$PBS_O_TZ	value of the TZ variable in the environment in which qsub was executed
$PBS_O_HOST	the name of the host upon which the qsub command is running
$PBS_O_QUEUE	the name of the original queue to which the job was submitted
$PBS_O_WORKDIR	the absolute path of the current working directory of the qsub command

The only variable I use all the time: keeps results out of your home directory. All products go into the current working directory.



$PBS_O_WORKDIR

Table 2: PBS Variables. Stuff you can use in your submit scripts.

Variable	Description
$PBS_ENVIRONMENT	set to PBS_BATCH to indicate that the job is a batch job; otherwise, set to PBS_INTERACTIVE to indicate that the job is a PBS interactive job
$PBS_JOBID	the job identifier assigned to the job by the batch system
$PBS_JOBNAME	the job name supplied by the user
$PBS_NODEFILE	the name of the file that contains the list of nodes assigned
$PBS_QUEUE	the name of the queue from which the job is executed
$PBS_O_HOME	value of the HOME variable in the environment in which qsub was executed
$PBS_O_LANG	value of the LANG variable in the environment in which qsub was executed
$PBS_O_LOGNAME	value of the LOGNAME variable in the environment in which qsub was executed
$PBS_O_PATH	value of the PATH variable in the environment in which qsub was executed
$PBS_O_MAIL	value of the MAIL variable in the environment in which qsub was executed
$PBS_O_SHELL	value of the SHELL variable in the environment in which qsub was executed
$PBS_O_TZ	value of the TZ variable in the environment in which qsub was executed
$PBS_O_HOST	the name of the host upon which the qsub command is running
$PBS_O_QUEUE	the name of the original queue to which the job was submitted
$PBS_O_WORKDIR	the absolute path of the current working directory of the qsub command

The only variable I use all the time: keeps results out of your home directory. All products go into the current working directory.

A key variable when doing job arrays. Stands in for a job index, usually 1,2,3,4,5….

$PBS_O_WORKDIR

$PBS_ARRAY _INDEX

Table 3: PBS Directives. Stuff you need in your submit scripts.

Assign all environmental variables to a batch job	#PBS -V
Set the job name	#PBS -N <JOBNAME>
Run in the queue named "batch"	#PBS -q batch
Use the bourne shell	#PBS -S /bin/bash
Send email when the job is completed	#PBS -m abe; #PBS -M username@amnh.org
Set destination for standard error	#PBS -e /home/username/HPC_RUNS/<ERROR FILE NAME>
Set destination for standard out	#PBS -o /home/username/HPC_RUNS/<OUTPUT FILE NAME>
Specify the number of cpus for your job	#PBS -l select=1:ncpus=64
Specify memory you expect to use. Use units 'mb' or 'gb'.	#PBS -l mem=256m
Specify anticipated run-time, where walltime=HH:MM:SS	#PBS -l walltime=08:00:00

Table 3: PBS Directives. Stuff you need in your submit scripts.

Assign all environmental variables to a batch job	#PBS -V	→ Job inherits your env: always a good idea
Set the job name	#PBS -N <JOBNAME>	→ Your job label in qstat
Run in the queue named "batch"	#PBS -q batch	→ Direct to a queue; now 1; future more?
Use the bourne shell	#PBS -S /bin/bash	→ Because what other shell is there, really?
Send email when the job is completed	#PBS -m abe; #PBS -M username@amnh.org	
Set destination for standard error	#PBS -e /home/username/HPC_RUNS/<ERROR FILE NAME>	
Set destination for standard out	#PBS -o /home/username/HPC_RUNS/<OUTPUT FILE NAME>	
Specify the number of cpus for your job	#PBS -l select=1:ncpus=64	→ Always set!
Specify memory you expect to use. Use units 'mb' or 'gb'.	#PBS -l mem=256m	→ Important only if your job is low CPU but high mem
Specify anticipated run-time, where walltime=HH:MM:SS	#PBS -l walltime=08:00:00	→ Default is 1 hour. But who only goes For 1 hr?? So set!

Let’s write a PBS script for a serial blast submit together…

Let’s write a PBS script for a serial blast submit together…

```
#!/bin/bash
#PBS -V
#PBS -q batch
#PBS -S /bin/bash
#PBS -N serial-blast
#PBS -l ncpus=1
#PBS -l walltime=00:10:00
cd $PBS_O_WORKDIR
module load ncbi-blast-2.6.0+
date
blastn -query /home/anarechania/nas3/hpc_class/blast_exercise/dicty_rna.fa \
-db /home/anarechania/nas3/hpc_class/blast_exercise/GRCh38_latest_genomic.fna \
-out /home/anarechania/nas3/hpc_class/blast_exercise/serial.blastout
date
```

Since blast queries are independent, let's parallelize using job arrays!

Job arrays require the PBS directive -J

and

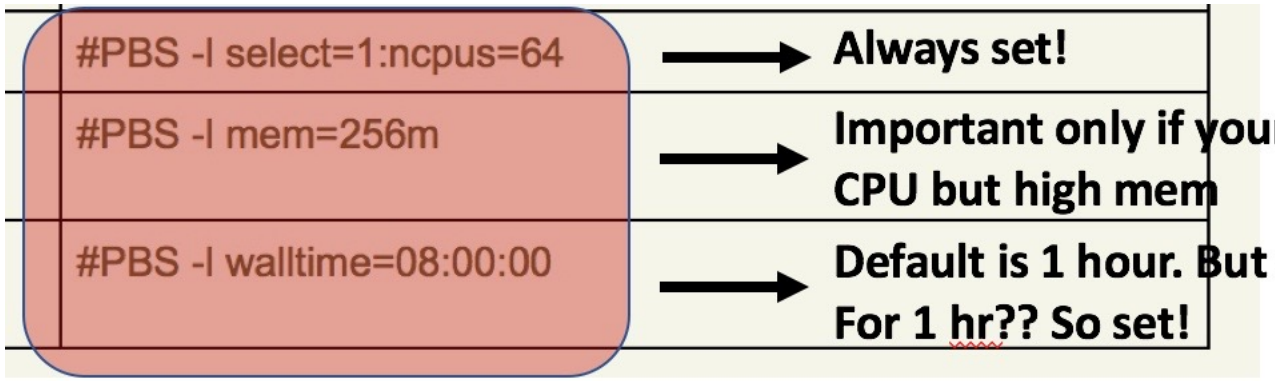
The PBS variable $PBS_ARRAY_INDEX

```
[anarechania@huxley-head blast_exercise]$ ls dicty_rna.fa.* | less
dicty_rna.fa.1
dicty_rna.fa.2
dicty_rna.fa.3
```

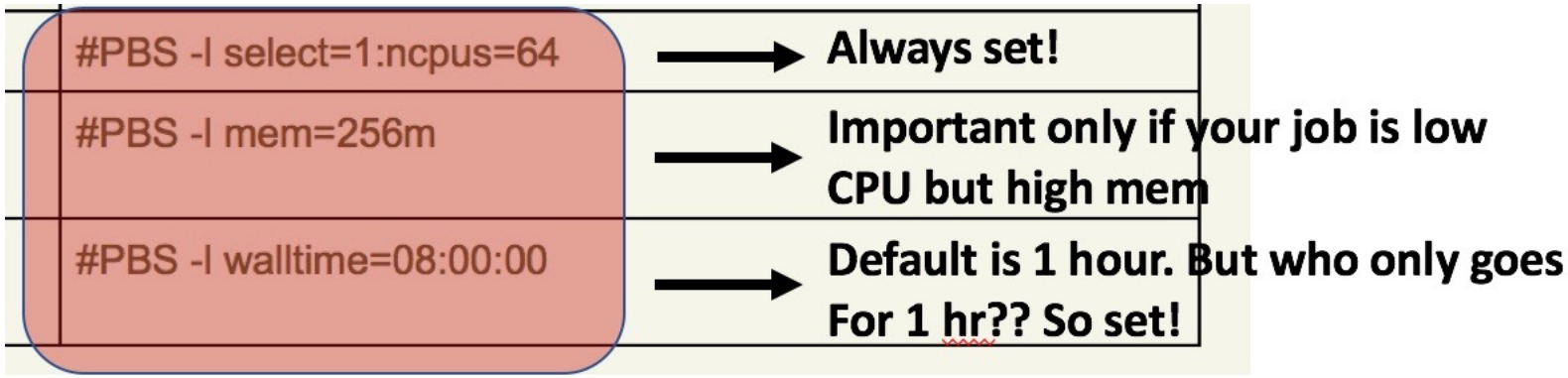
Since blast queries are independent, let's parallelize using job arrays!

```
#!/bin/bash
#PBS -V
#PBS -q batch
#PBS -S /bin/bash
#PBS -N array-blast
#PBS -l ncpus=1
#PBS -l walltime=00:10:00
#PBS -J 1-3
cd $PBS_O_WORKDIR
module load ncbi-blast-2.6.0+
Date
blastn -query /home/anarechania/nas3/hpc_class/blast_exercise/dicty_rna.fa.$PBS_ARRAY_INDEX \
-db /home/anarechania/nas3/hpc_class/blast_exercise/GRCh38_latest_genomic.fna \
-out /home/anarechania/nas3/hpc_class/blast_exercise/serial.blastout.$PBS_ARRAY_INDEX \
date
```

The -l directive is at the heart of PBS (primary way to control resource)



The -l directive is at the heart of PBS (primary way to control resource)



#PBS -l select=X:ncpus=Y:mem=Z

Where
X = the number of nodes or units of the resources required
Y = the number of cores (individual processors) required on each node
Z = the amount of memory (typically in mb or gb) required on each node

The -l directive is at the heart of PBS (primary way to control resource)

As outlined above, an exhaustive select statement for a **complete node** on Huxley would look like this:

#PBS -l select=1:ncpus=64:mem=256gb

The -l directive is at the heart of PBS (primary way to control resource)

As outlined above, an exhaustive select statement for a **complete node** on Huxley would look like this:

#PBS -l select=1:ncpus=64:mem=256gb

Some parallel software can process data across nodes. For example, if you want **two full nodes** on Huxley, you'd specify:

#PBS -l select=2:ncpus=64:mem=256gb

The -l directive is at the heart of PBS (primary way to control resource)

As outlined above, an exhaustive select statement for a **complete node** on Huxley would look like this:

#PBS -l select=1:ncpus=64:mem=256gb

Some parallel software can process data across nodes. For example, if you want **two full nodes** on Huxley, you'd specify:

#PBS -l select=2:ncpus=64:mem=256gb

MPI jobs often require even more specific resource requests. To reserve **two nodes for an MPI process**:

#PBS -l select=2:ncpus=64:mpiprocs=64

where mpiprocs specifies the number of processors allocated to MPI.

Now that we know the magic of -l, let's build a raxml tree with threads!

```
raxml -T 64 -m PROTGAMMAJTTF -f d -N 1 -n best -p 1977 \
-s
/home/anarechania/nas3/hpc_class/raxml_exercise/data/A0JC77.phy
```

Now that we know the magic of -l, let's build a raxml tree with threads!

```
#!/bin/bash
#PBS -V
#PBS -q batch
#PBS -S /bin/bash
#PBS -N threads-raxml
#PBS -l ncpus=64
#PBS -l walltime=00:10:00
cd $PBS_O_WORKDIR
module load RAxML-8.2.11
date
raxml -T 64 -m PROTGAMMAJTTF -f d -N 1 -n best -p 1977 \
-s /home/anarechania/nas3/hpc_class/raxml_exercise/data/A0JC77.phy
date
```

Let’s try doing a single end assembly with MPI abyss…

```
abyss-pe np=16 k=75 v=-v \
se=/home/anarechania/nas3/hpc_class/abyss_exercise/27924_S15_
L001_R2_001.fastq \
name=test "unitigs”
```

#PBS -l select=2:ncpus=64:mpiprocs=64

Let’s try doing a single end assembly with MPI abyss…

```
#!/bin/bash
#PBS -V
#PBS -q batch
#PBS -S /bin/bash
#PBS -N abyss-mpi
#PBS -l select=1:ncpus=16:mpiprocs=16
#PBS -l walltime=1:00:00
cd $PBS_O_WORKDIR
module load openmpi-2.1.1 abyss-2.0.2
date
abyss-pe np=16 k=75 v=-v \
se=/home/anarechania/nas3/hpc_class/abyss_exercise/27924_S15_L001_R2_001.fastq \
name=test "unitigs”
date
```

Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)
3. More Talking
    * Where does genomics fit in big data science?
    * Specific QA with Sajesh Singh
4. More Doing (BONUS, if we make it)
    * GNU Parallel
    * auto-generating PBS scripts for complex workflows

REVIEW

doi:10.1038/nature24286

DNA sequencing at 40: past, present and future

Jay Shendure[1,2], Shankar Balasubramanian[3,4], George M. Church[5], Walter Gilbert[6], Jane Rogers[7], Jeffery A. Schloss[8] & Robert H. Waterston[1]

BOX 2

The milestones listed below correspond to key developments in the availability of new reference genomes, new sequencing-related computational tools and the applications of DNA sequencing in new ways or to new areas. These are large topics, and we apologize for any omissions.

Genome milestones

1977: Bacteriophage ΦX174 (ref. 72)
1982: Bacteriophage lambda[13]
1995: *Haemophilus influenzae*[26]
1996: *Saccharomyces cerevisiae*[27]
1998: *Caenorhabditis elegans*[28]
2000: *Drosophila melanogaster*[32]
2000: *Arabidopsis thaliana*[146]
2001: *Homo sapiens*[29–31]
2002: *Mus musculus*[147]
2004: *Rattus norvegicus*[148]
2005: *Pan troglodytes*[149]
2005: *Oryza sativa*[150]
2007: *Cyanidioschyzon merolae*[126]
2009: *Zea mays*[151]
2010: Neanderthal[88]
2012: Denisovan[145]
2013: The HeLa cell line[152,153]
2013: *Danio rerio*[154]
2017: *Xenopus laevis*[155]

Computational milestones

1981: Smith–Waterman[156]
1982: GenBank (https://www.ncbi.nlm.nih.gov/genbank/statistics/)
1990: BLAST[16]
1995: TIGR assembler[24]
1996: RepeatMasker
1997: GENSCAN[157]
1998: phred, phrap, consed[22]
2000: Celera assembler[25]
2001: Bioconductor
2001: EULER[74]
2002: BLAT[158]
2002: UCSC Genome Browser[159]
2002: Ensembl[160]
2005: Galaxy[161]
2007: NCBI Short Read Archive
2008: ALLPATHS[162]
2008: Velvet[75]
2009: Bowtie[83]
2009: BWA[82]
2009: SAMtools[84]
2009: BreakDancer[163]
2009: Pindel[164]
2009: TopHat[115]
2010: SOAPdenovo[165]
2010: GATK[85]
2010: Cufflinks[116]
2011: Integrated Genomics Viewer[166]
2013: HGAP/Quiver[167]
2017: Canu[81]

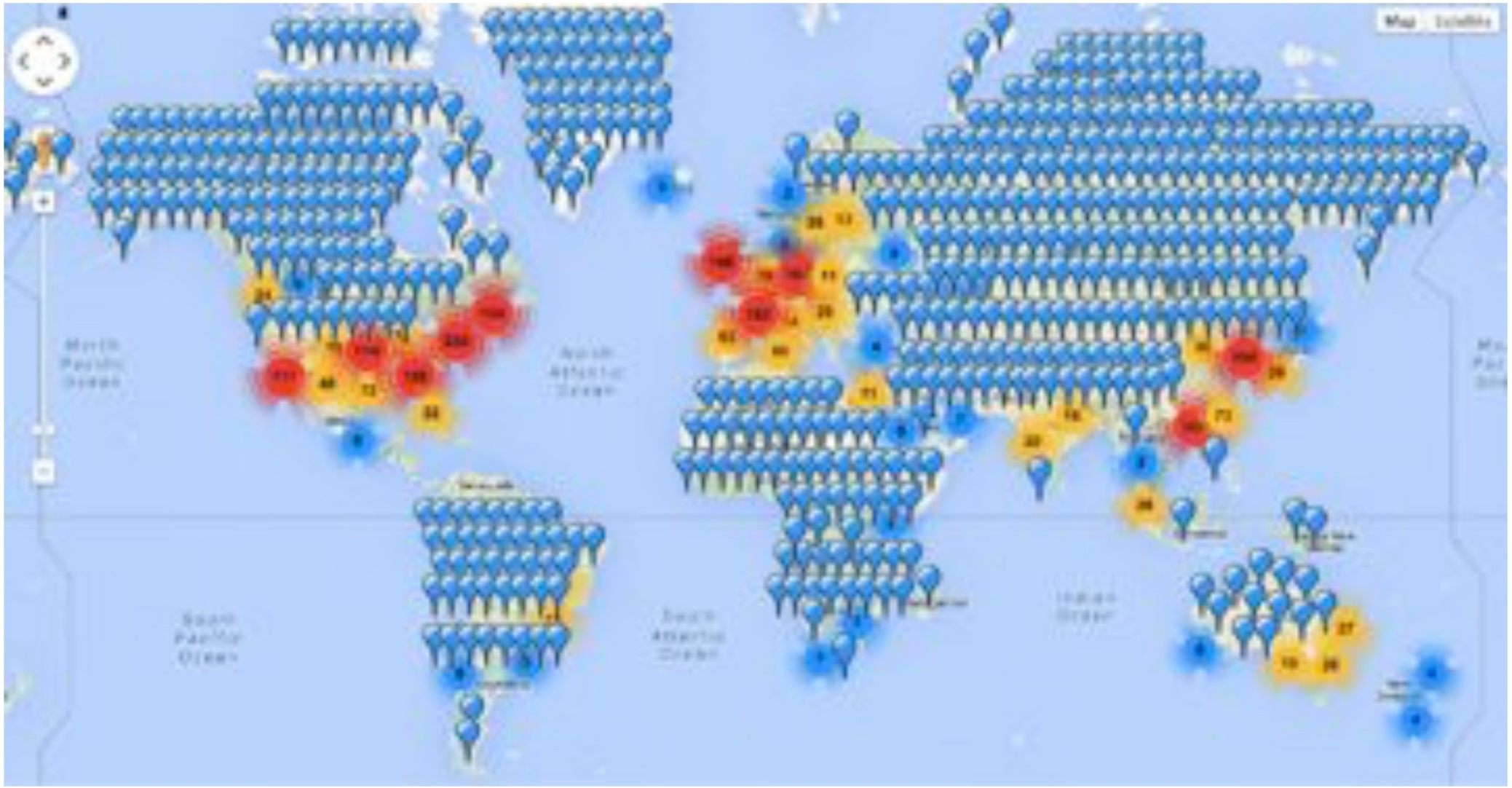
Application milestones

1977: Genome sequencing[72]
1982: Shotgun sequencing[13]
1983, 1991: Expressed sequence tags[107,108]
1995: Serial analysis of gene expression[109]
1998: Large-scale human SNP discovery[168]
2004: Metagenome assembly[122]
2005: Bacterial genome resequencing with NGS[40,41]
2007: Chromatin immunoprecipitation followed by sequencing (ChIP–seq) using NGS[117]
2007–2008: Human genome and cancer genome resequencing using NGS[55,90–92]
2008: RNA-seq using NGS[110–114]
2008: Chromatin accessibility using NGS[118]
2009: Exome resequencing using NGS[97]
2009: Ribosome profiling using NGS[119]
2010: Completion of Phase I of the 1000 Genomes Project[98]
2010: *De novo* assembly of a large genome from short reads[169]
2011: Haplotype-resolved human genome resequencing using NGS[170,171]
2016: Human genome *de novo* assembly with PacBio[172]
2017: Human genome *de novo* assembly with nanopore[64]

Computational milestones

1981: Smith–Waterman[156]
1982: GenBank (https://www.ncbi.nlm.nih.gov/genbank/statistics/)
1990: BLAST[16]
1995: TIGR assembler[24]
1996: RepeatMasker
1997: GENSCAN[157]
1998: phred, phrap, consed[22] → **High throughput sanger shotgun sequencing**
2000: Celera assembler[25]
2001: Bioconductor
2001: EULER[74]
2002: BLAT[158]
2002: UCSC Genome Browser[159]
2002: Ensembl[160] → **Development burst 1.**
2005: Galaxy[161]
2007: NCBI Short Read Archive → **Next generation sequencing: 454; Solexa**
2008: ALLPATHS[162]
2008: Velvet[75]
2009: Bowtie[83]
2009: BWA[82]
2009: SAMtools[84]
2009: BreakDancer[163]
2009: Pindel[164]
2009: TopHat[115]
2010: SOAPdenovo[165]
2010: GATK[85]
2010: Cufflinks[116] → **Development burst 2.**
2011: Integrated Genomics Viewer[166]
2013: HGAP/Quiver[167] → **3rd generation real time sequencing: PacBio; nanopore**
2017: Canu[81]

Development Burst 3??



BIG DATA

Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001

1 ZettaB is 1 Billion TBs

Stephens 2015

NAS1/2

- **72TB RAID storage**

NAS3 (Huxley)

- **700TB RAID storage**
- **128GB SSD / node**

Array1 (Cuvier)

- **90T Direct Attached Storage**

Agenda:

1. Talking
    * Museum Resources / Quotas/ Website
    * Be Nice Rules
    * Parallel topics: embarrassingly parallel; threads; MPI
2. Doing
    * PBS script for a blast job (serial subdivided submissions; array job)
    * PBS script for RAxML reconstructions (threaded job for a gene family)
    * PBS script for ABYSS assemblies (MPI job)
3. More Talking
    * Where does genomics fit in big data science?
    * Specific QA with Sajesh Singh
4. More Doing (BONUS, if we make it)
    * GNU Parallel
    * auto-generating PBS scripts for complex workflows

Make your life so much easier with GNU Parallel…

Swiss army knife of embarrassingly parallel bioinformatics.

Two examples:

1. gzip/home/anarechania/array1 many files, but nicely!
2. Parallelize piped or composed commands: a blast to a blast parse across all three query input partitions.

Make your life so much easier with GNU Parallel…

```
$> cd gnu_parallel_exercise
$> for i in {1..10}; do gzip dicty_genome.$i.fa & done
$> gzip –d *
$> for i in {1..10}; do echo $i; sem –j2 gzip dicty_genome.$i.fa; done
```

```
$> parallel -j3 'blastn -query ../blast_exercise/dicty_rna.fa.{}
-db ../blast_exercise/GRCh38_latest_genomic.fna | grep -c "No hits
found" > {}.out' ::: {1..3}
```

Autogenerate multiple pbs scripts and submit independent calcs in parallel

You may have noticed that the raxml directory contains 5 alignments…

Use a perl program in a bash loop to write and submit a PBS script for every alignment.

Autogenerate multiple pbs scripts and submit independent calcs in parallel

```
File Edit Options Buffers Tools Help
#!/usr/bin/perl

use Getopt::Std;

my %opts = ();
getopts ('e:o:l:w:m:b:n:', \%opts);
my $resources = $opts{'l'};
my $wall = $opts{'w'};
my $modules = $opts{'m'};
my $executable = $opts{'b'}; #delimit multiple executable commands with semicolons
my $name = $opts{'n'};

# build the pbs submit script
open (P, ">pbs.sh\n");
print P "#!/bin/bash\n";
print P "#PBS -V\n";
print P "#PBS -q batch\n";
print P "#PBS -S /bin/bash\n";

print P "#PBS -N $name\n";
print P "#PBS -l $resources\n";
print P "#PBS -l $wall\n";

print P "cd \$PBS_O_WORKDIR\n";
print P "module load $modules\n";

if ($executable =~m/\;/){
    my @executable = split (/\;/, $executable);
    foreach my $exe (@executable){
        print P "$exe\n";
    }
}
else{
    print P "$executable\n";
}
close (P);

`qsub pbs.sh`;
```

Autogenerate multiple pbs scripts and submit independent calcs in parallel

```
for i in `cat list`
do
       mkdir $i
       cd $i
       generic_pbs.pl -l ncpus=64 -w walltime=1:00:00 -m RAxML-8.2.11 -n $i \
       -b 'raxml -T 64 -m PROTGAMMAJTTF -f d -N 1 -n '$i' -p 1977 \
       -s /home/anarechania/nas3/hpc_class/auto_gen_exercise/data/'$i'' \
       cd ../
done
```