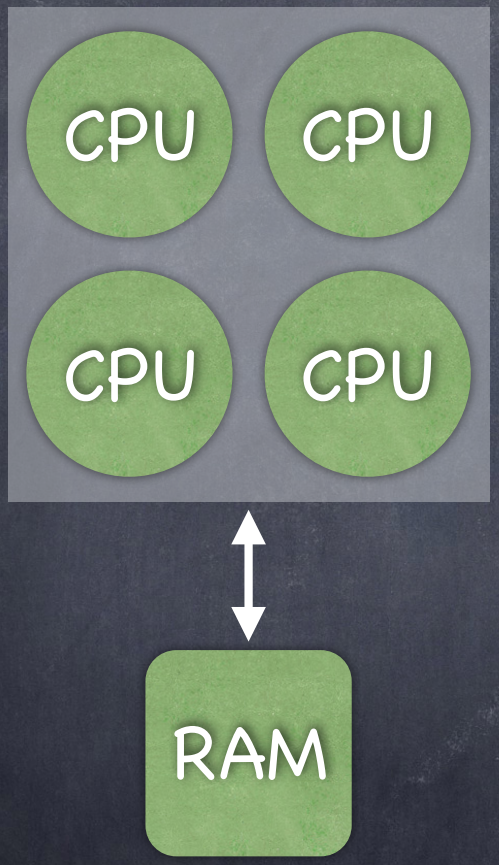
CIBR Parallel Computing Mini-workshop

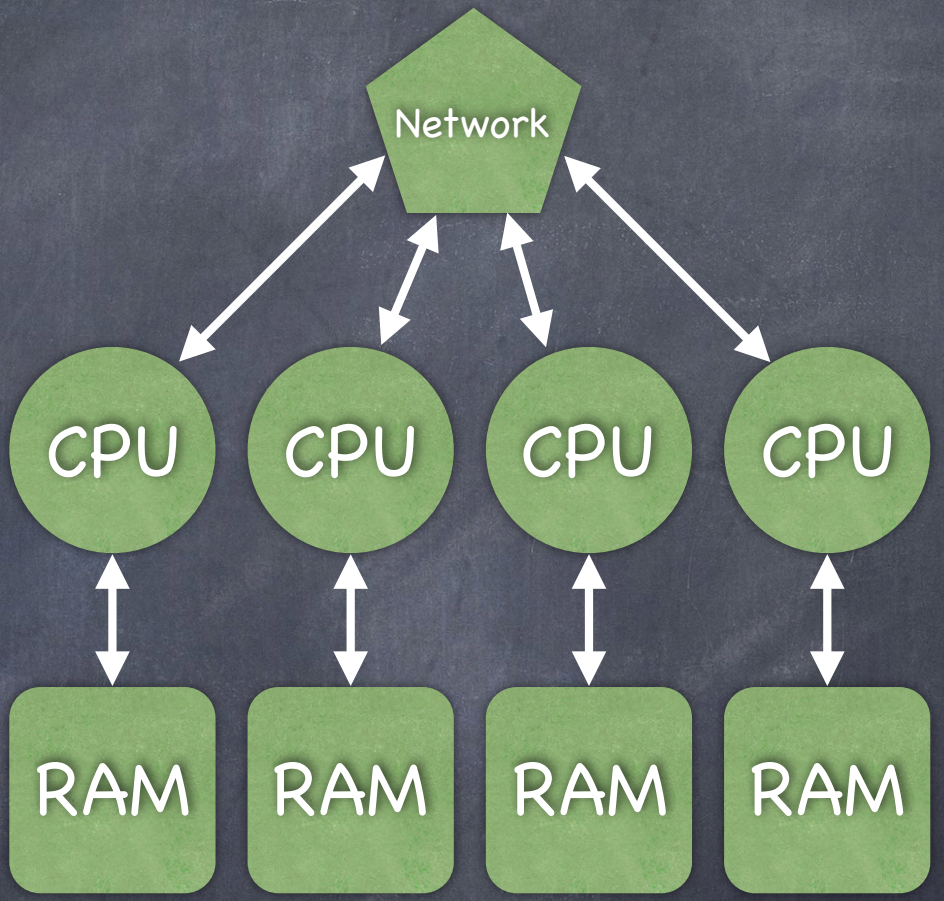
Slides available at:
http://blake.bcm.edu/emanwiki/CIBRClusters

Prof. Steven Ludtke
N420, sludtke@bcm.edu

Part 1
Cluster Architecture

Shared or Distributed ?

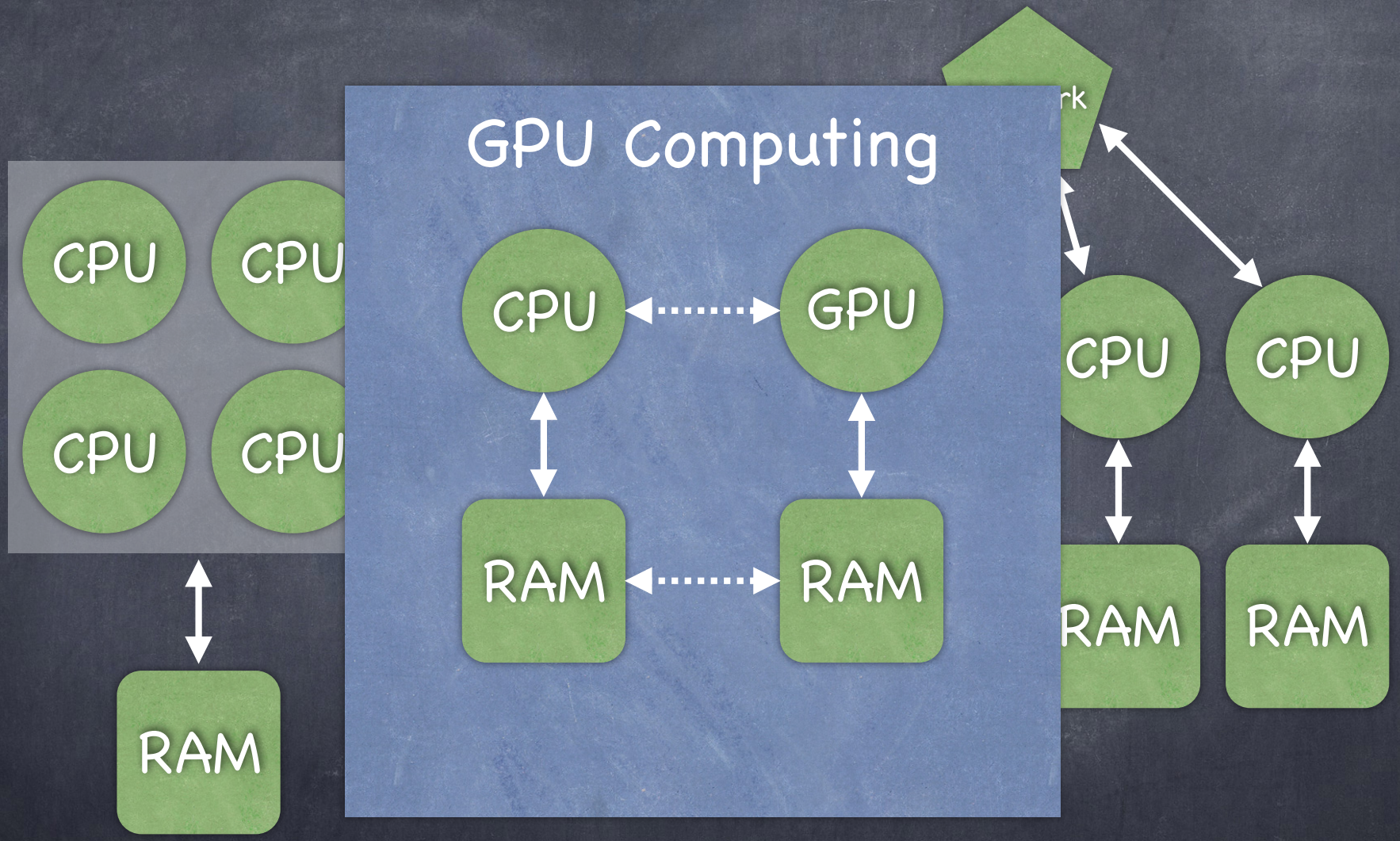




Easy Parallelism

Inexpensive

Shared or Distributed ?



Easy Parallelism

Inexpensive

Units

- FLOPS/s - FLoating point OPerationS per second
- Kilo - 10^3 - 1000 (or 1024)
- Mega - 10^6 - 1,000,000
- Giga - 10^9 - 1,000,000,000
- Tera - 10^{12} - 1,000,000,000,000
- Peta - 10^{15} - 1,000,000,000,000,000

SMP or Distributed

S 1964, CDC 6600, $60m (2012 $), 500 KFlops, 1 CPU

S 1977, CRAY 1, $33m, 80 MFlops, 1 CPU

iPhone4S

S 1984, CRAY XMP, $25m, 800 MFlops, 4 CPUs - vector

D 1987, CM-2, $22m, 6 GFlops, 65,536 CPUs, 2048 MPU

S 1996, Origin 2000, ~$3m, 10 GFlops, 32 CPUs SMP

iPhoneX

D 2005, Cluster, $0.4m, 900 GFlops, 106 nodes, 212 cores

? 2011, Cluster, $0.22m, 5.5 TFlops, 48 nodes, 576 cores

? 2014, Cluster, $0.29m, 40 TFlops, 40 nodes, 960 cores

! 2015, Tesla K80 GPU, $0.005m, 5.6 TFlops, ~5000 cores

2018, Titan V GPU, $0.003m, 110 TFlops, ~6000 cores

Relative Scaling

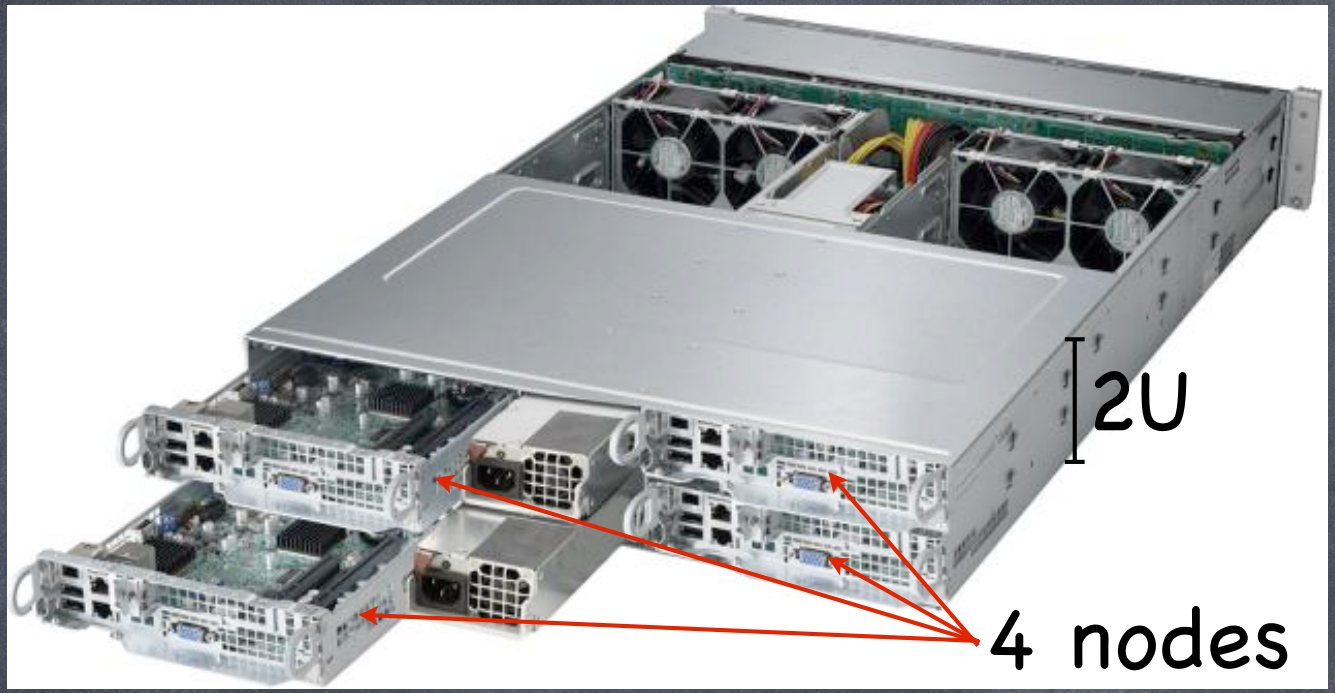
	1996	2006	2016
CPU (workstation)	0.1 GFLOPS/s	20 GFLOPS/s	1000 GFLOPS/s
GPU		50 GFLOPS/s	10,000 GFLOPS/s
Disk Capacity	1 GB	100 GB	10 TB
Disk I/O	8 MB/s	80 MB/s	600 - 1500 MB/s
Network	10 Mb/s	100 Mb/s	1000 Mb/s

Relative Scaling

	1996	2006	2016	
CPU (workstation)	0.1 GFLOPS/s	20 GFLOPS/s	1000 GFLOPS/s	
GPU		50 GFLOPS/s	10,000 GFLOPS/s	100x in 10 years
Disk Capacity	1 GB	100 GB	10,000 GB	
Disk I/O	8 MB/s	80 MB/s	600 - 1500 MB/s	10x in 10 years
Network	10 Mb/s	100 Mb/s	1000 Mb/s	

Typical Rack 42U

Cluster Hardware



- 1 Chassis:
  - 4 nodes
    - 2 processors/node:
      - 16 cores/processor
    - 96 GB RAM/node
    - 2 TB Hard Drive/node
  - 10 Gb ethernet/IB

- 128 cores
  - 16 TFLOPS (peak)
  - 100 GB/sec RAM
- 384 GB RAM (3GB/core)

- $40,000 ($312/core)

Cluster Hardware

- 1 Rack:
  - 20 * 2U ->
    - $40,000 * 20 -> $800k + ~$30k (rack, etc.)
    - 20*128 cores -> 2560 cores
      - 300 TFLOPS Peak
      - ~30 KW
        - 30 KW * 8700 hr/yr = 260 MWH/yr
        - ~$30,000/yr electric bill
        - A/C bill !?

Why Do I care? I'm bored!

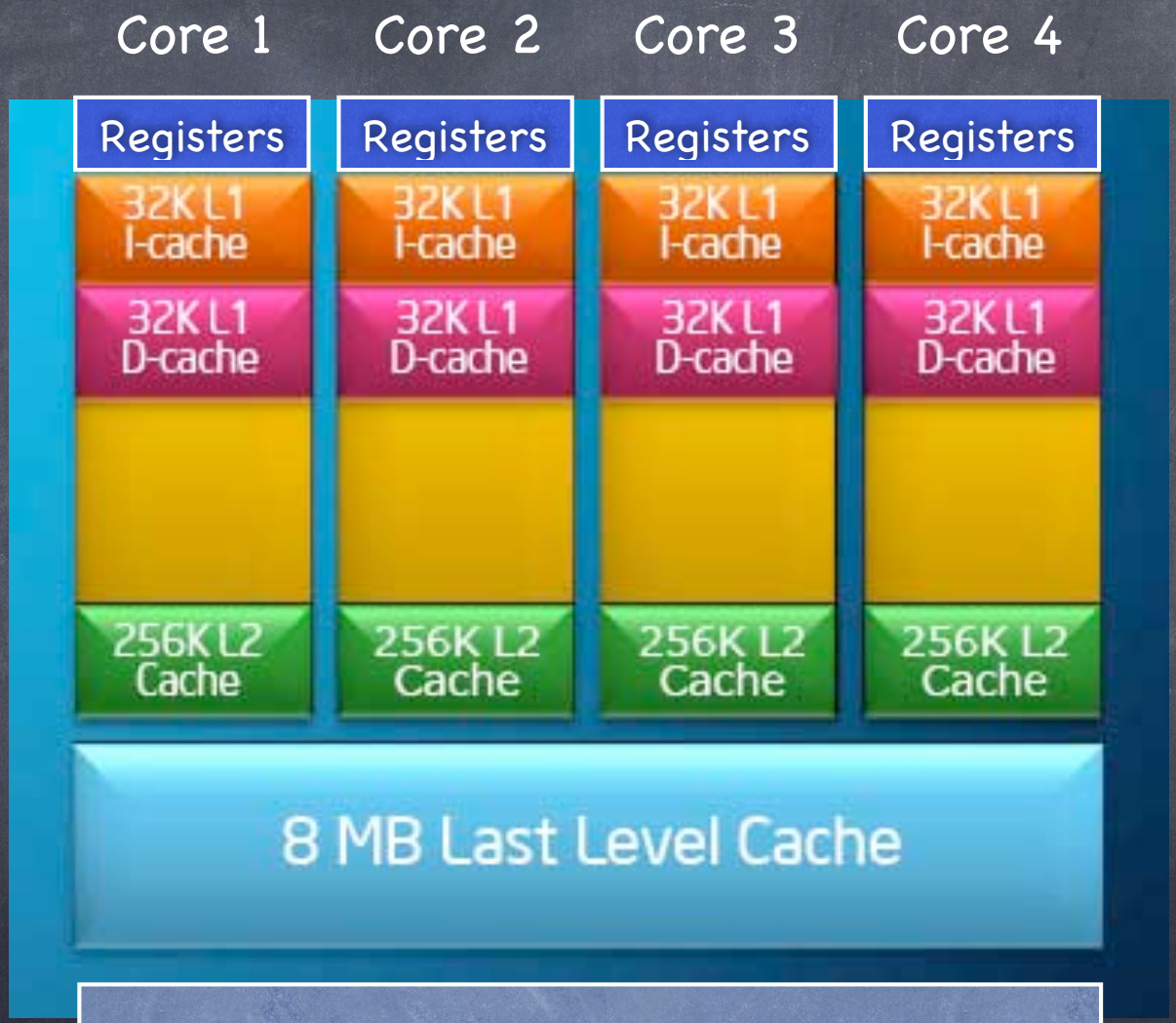
- 7.2. Review Criteria (XSEDE, FREE CLUSTER TIME)
- 1. Appropriateness of Methodology: ...
- 2. Appropriateness of Research Plan: ...
- 3. Efficient Use of Resources: The resources selected should be used as efficiently as is reasonably possible and in accordance with the recommended use guidelines of those resources. For computational resources, performance and parallel scaling data should be provided along with a discussion of optimization and/or parallelization work to be done to improve the applications. If the reviewers conclude that the request is more appropriate on XSEDE resources other than those requested, they may recommend an allocation on those other resources instead.

Comparison of Languages

Loop/Array/Math Benchmark

Language	Time
C++ (-O2)	1
C++ (no opt)	2
Javascript (JIT)	2
Java	5.1
Python	16.5
Perl	24.6
PHP	55.6

Quad Core Cache Structure



Full Speed

Extra Super Speedy

Super Speedy

Just Speedy

Ouch

DISK & Network

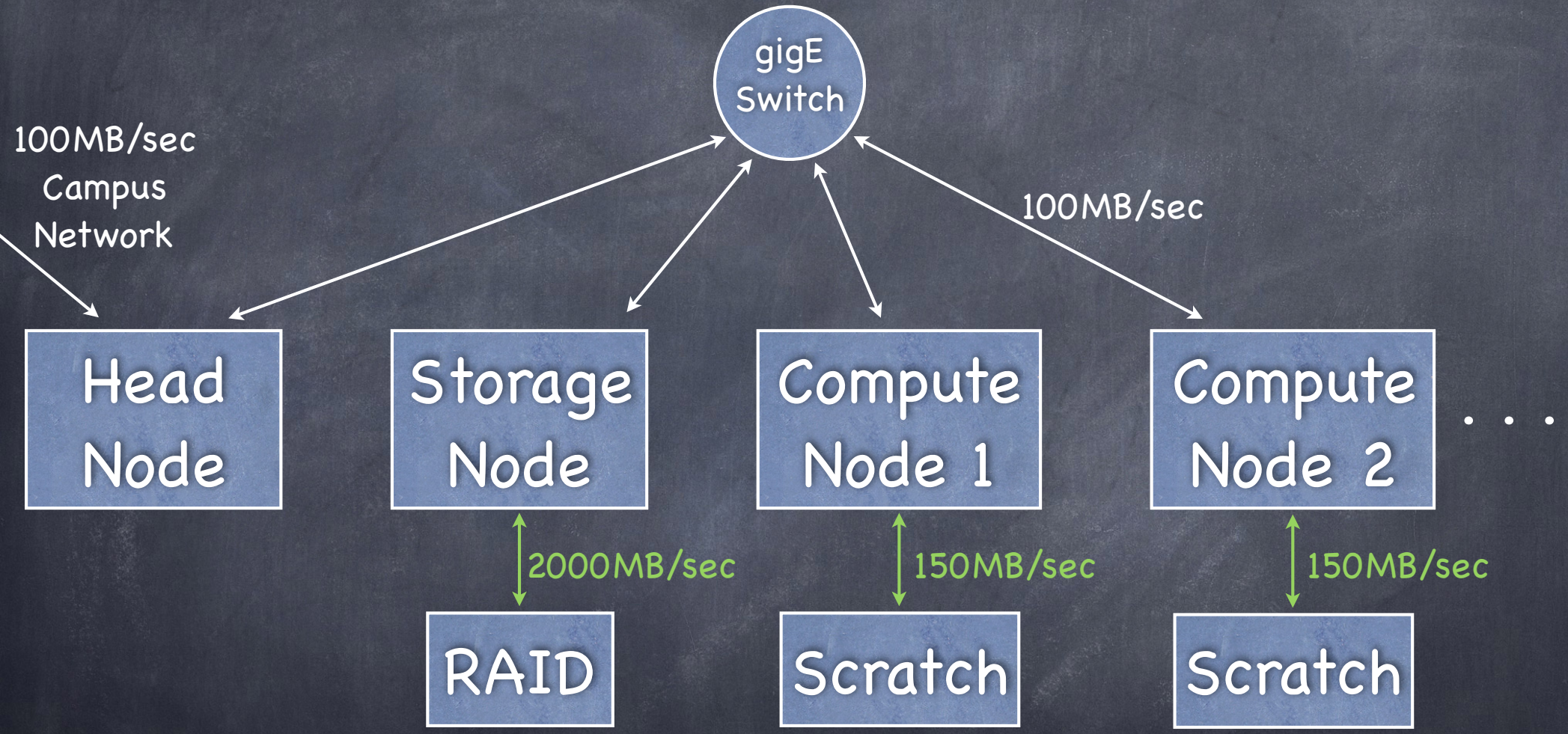
RAM

Meh

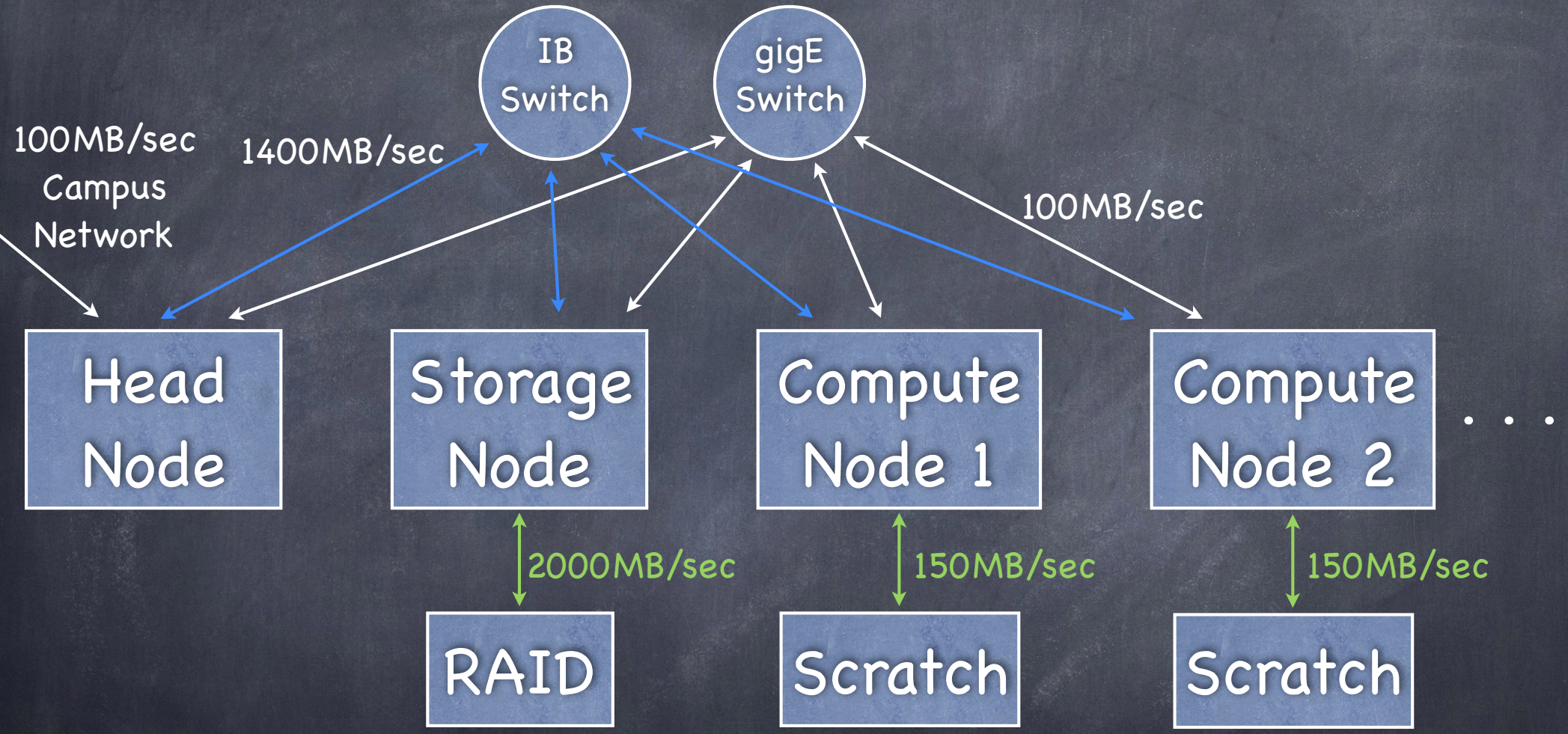
Speed

- 300,000 MIPS - (Million instructions per second) current peak capabilities of a single CPU (with multiple cores)
- 100,000 MB/sec - Level 1 cache memory bandwidth (32 kbytes/core)
- 50,000 MB/sec - Level 2 cache memory bandwidth (256 kbytes/core)
- 35,000 MB/sec - Level 3 cache memory bandwidth (8000 kbytes/CPU)
- 20,000 MB/sec - RAM (typical DDR3 dual channel)
- 8,000 MB/sec - PCIe x16 (2.0)
- 1,500 MB/sec - 12 drive RAID6 with PCIe controller or PCIe SSD
- 800 MB/sec - QDR Infiniband
- 150 MB/sec - Typical sequential disk read bandwidth for one drive
- 100 MB/sec - Gigabit network

Hypothetical Cluster



Hypothetical Cluster



Part 2
Parallelism

Simple Task

- Take a 20 GB sequence and locate all of the TATA boxes within it.
  - Choice of language ?
  - Run on a cluster ?
  - Multiple cores ?
  - How long will it take to run ?
  - How to make it faster ?

Simple Task

- Take a 20 GB sequence and locate all of the TATA boxes within it.
- 20-120 s to read from disk, 200+ s to read on net
- computation minimal
  - Choice of language ? Doesn't matter
  - Run on a cluster ? No
  - Multiple cores ? No
  - How long will it take to run ? Disk/Net limited
  - How to make it faster ? Faster disk on local machine

Another Task

- You have 500, 4096x4096 pixel floating point images. You need to apply a (Fourier) low-pass filter to all of them
- Run on multiple cores ?
- Run on a cluster ?

Another Task

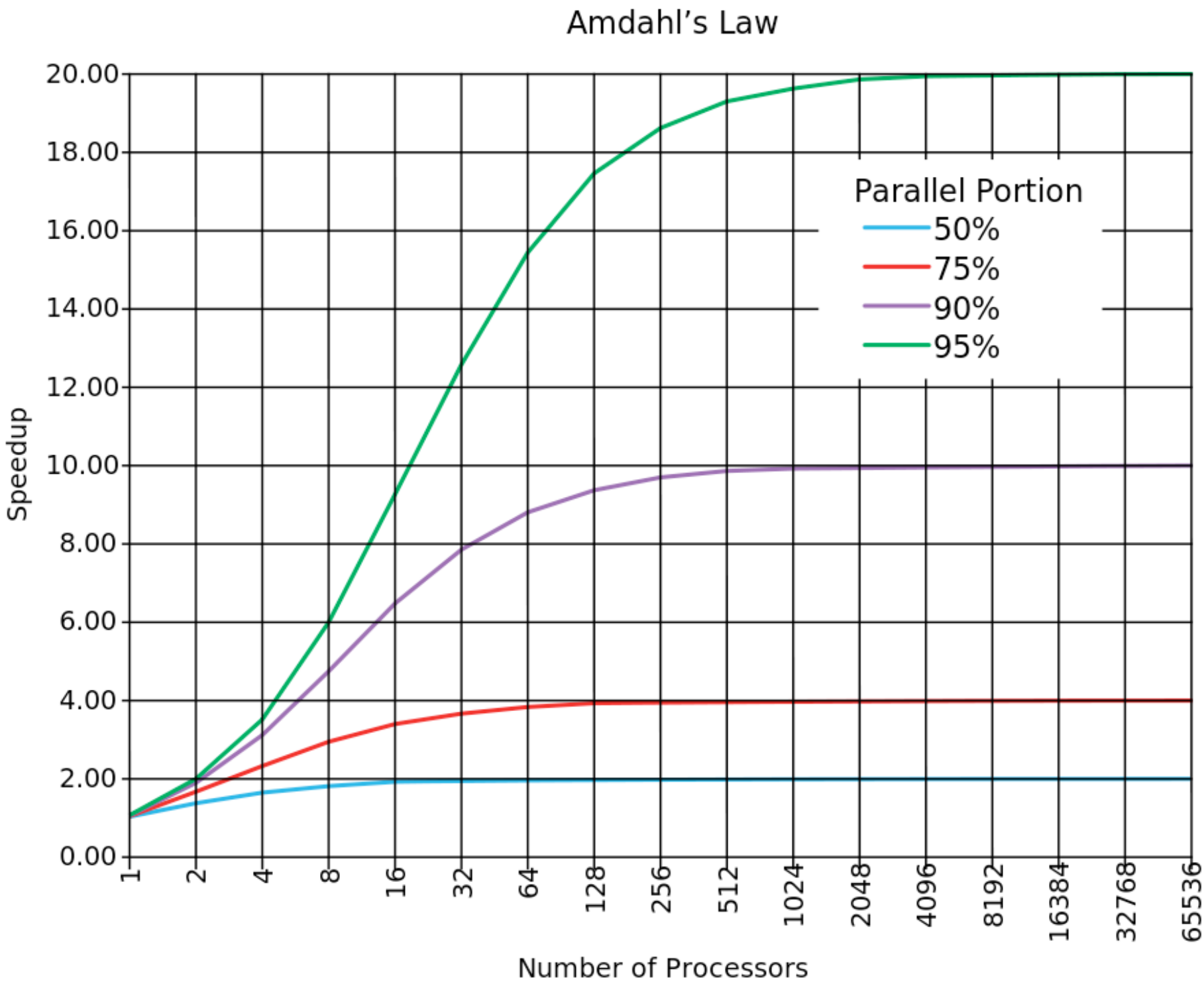
- You have 500, 4096x4096 pixel floating point images. You need to apply a (Fourier) low-pass filter to all of them
  - Read -> FFT -> multiply -> IFT -> Write
  - Image size: 64 MB
  - Total time for one image on desktop PC: ~3.5 sec
- Run on multiple cores ? Yes. I/O time 0.05 - 0.5 sec
- Run on a cluster ? Maybe, depends on next step

Slightly Trickier

- Iterative Image Alignment - You have a set of 1000, 256x256 images:
  - average all images together
  - align each image to the average
  - repeat 10x
- How to handle communications ?

Amdahls Law

- Speedup achievable with many processors is limited by the non-parallel portions of the task:
- S=1/(B+(1-B)/n)
- B=fraction of the code which cannot run in parallel
- n=number of processors


Amdahl's Law
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Speedup
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Slightly Trickier

- Iterative Image Alignment - You have a set of 1000, 256x256 images:
  - average all images together
  - align each image to the average
  - repeat 10x
- How to handle communications ?

Slightly Trickier

- average all images together
  - All images on 1 node ? (serial !)
  - How else to handle ?
- align each image to the average
  - Each node needs:
    - the reference
    - 1 or more images to align

Coarse vs Fine

- Coarse-grained parallelism
  - Tasks are completely independent (may have shared input data)
  - Example: filter 1000 images
- Fine-grained parallelism
  - Tasks need to communicate between each other continuously
  - Example: Matrix inversion

Example

- You have 200 sequences and wish to run a multiple sequence alignment against a set of 20 shorter reference sequences. How to parallelize ?

Coarse Grained?

- Each of the 200 sequences to one processor, which computes all of the 20 alignments for that sequence
- Advantages:
  - Very coarse, easy to distribute
  - Potentially 'perfectly' parallel
- Disadvantages:
  - Only works if you have at most 200 cores
  - If the 200 sequences vary significantly in length, total time will be limited by the longest sequence

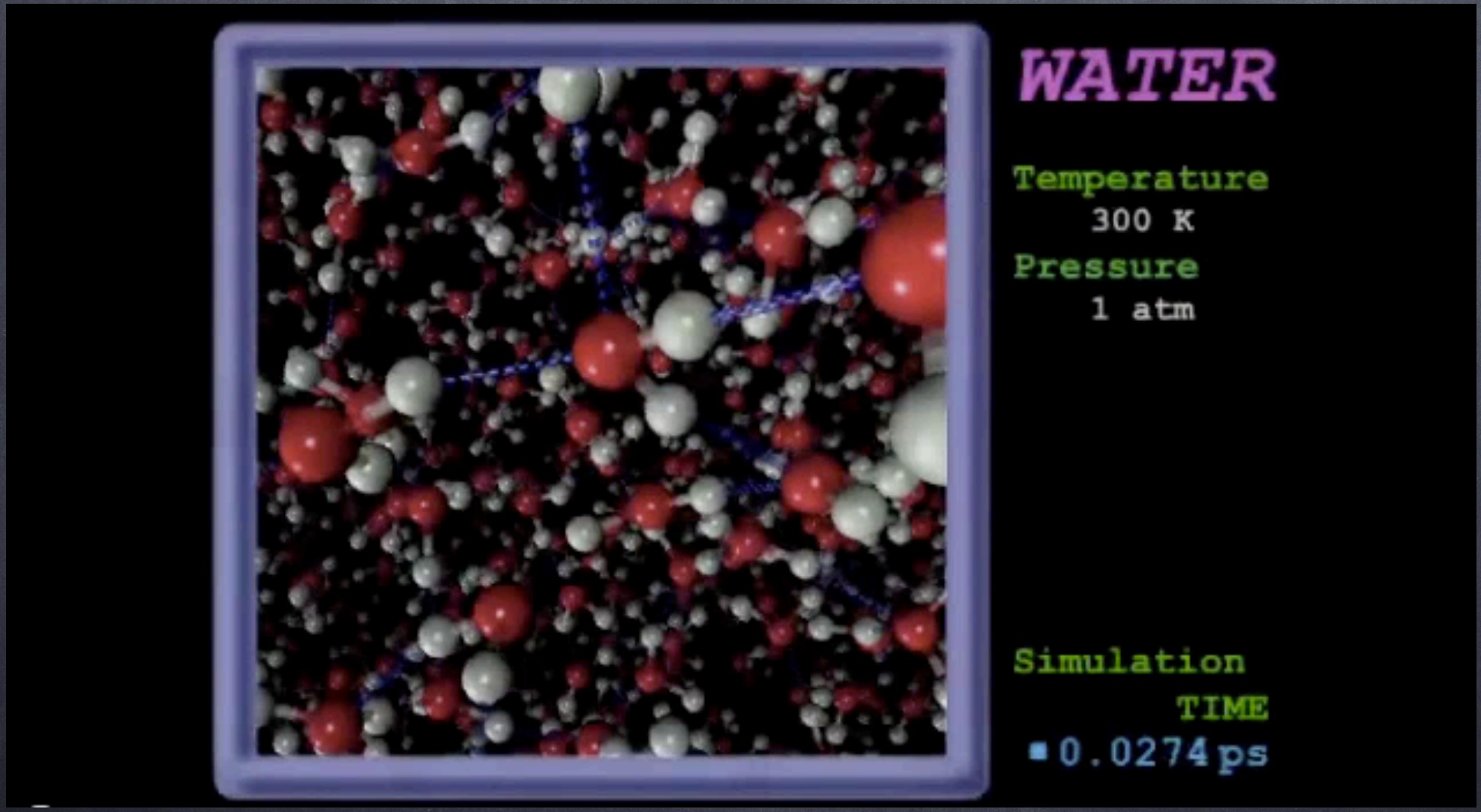
Fine Grained

- Tackle 1 sequence and 1 reference at a time. Each processor helps compute the local score
- Advantages:
  - Fine grained - more uniformly scalable
- Disadvantages
  - May be VERY inefficient due to communications bottlenecks

Intermediate Approach

- Split the overall process into 200*20 = 4000 individual alignment tasks, and send one to each core as it becomes available
- Advantages
  - Each task independent, so still 'perfectly' parallel
  - Parallelizable up to 4000 cores
- Disadvantages
  - May still have some inefficiencies with differing sequence lengths, particularly for large number of processors

MD Simulations



MD Simulations

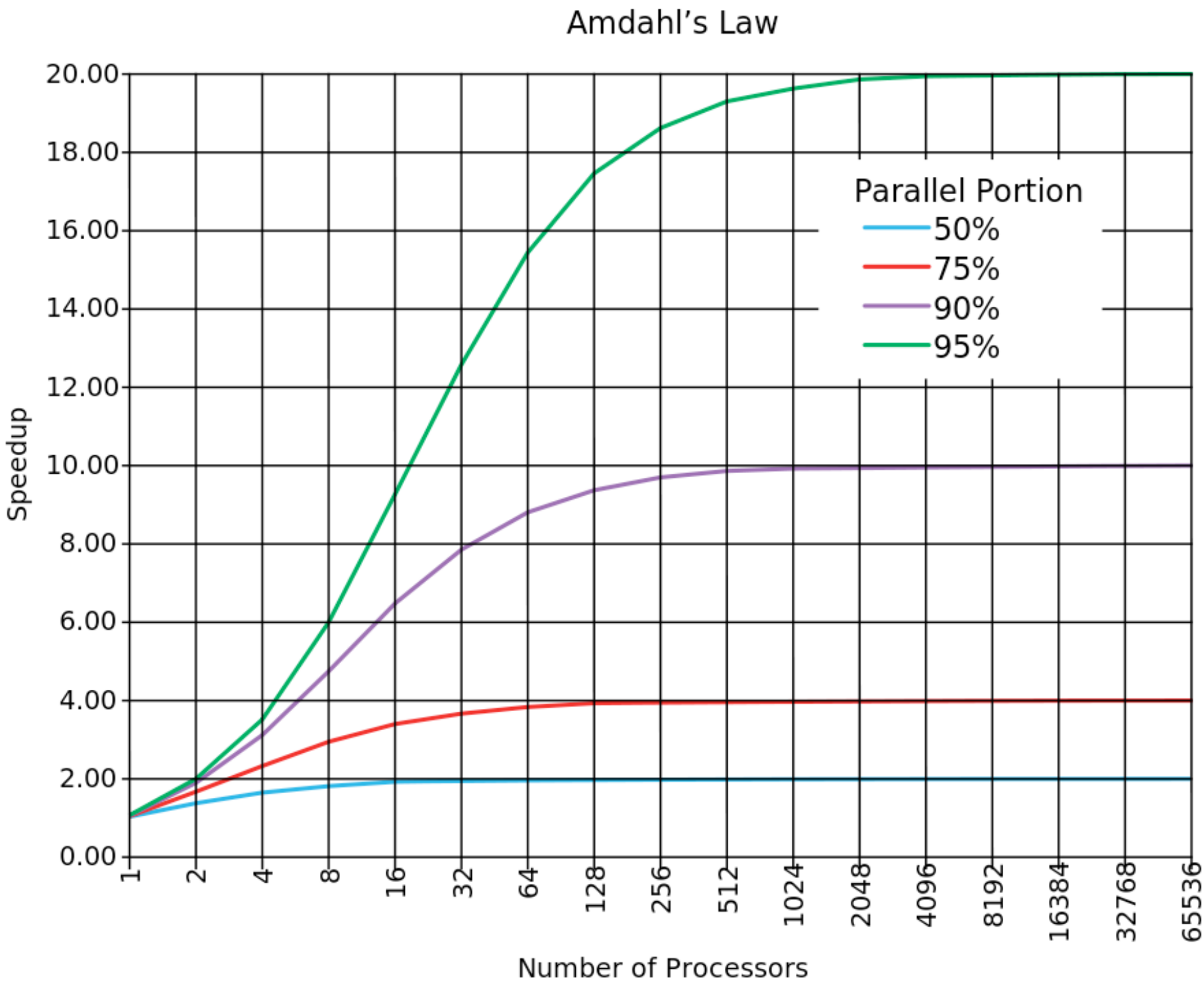
- VERY high CPU/Disk ratio
- Long-time single simulation
- Many short-time simulations (folding@home)

Native petaFLOPS threshold	Date crossed	Fastest Supercomputer at Date Crossed[Note 1]
1.0	September 16, 2007	0.2806 petaFLOP BlueGene/L[105]
2.0	May 7, 2008	0.4782 petaFLOP BlueGene/L[106]
3.0	August 20, 2008	1.042 petaFLOP Roadrunner[107]
4.0	September 28, 2008	1.042 petaFLOP Roadrunner[107]
5.0	February 18, 2009	1.105 petaFLOP Roadrunner[108]
6.0	November 10, 2011	8.162 petaFLOP K computer[109]

All of our clusters combined
total ~0.1 Petaflops

Questions to Ask Yourself

- Total time required for I/O
- Possible to share data?
- Total time required for processing
- Memory usage
- Interprocess communication


Amdahl's Law
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Speedup
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Where to Compute

	Cost	Best for	Problems
CIBR Clusters	Free+$ (co-op)	Moderate jobs, rapid access	Limited capacity, no GPU
XSEDE/TACC	Free (application)	Large jobs or smaller jobs less urgency	Application, capacity limit, queue length
Amazon EC2	$$ - $$$	Urgent or very infrequent jobs	Expensive, significant setup, storage issues
Local Workstation	$$$ up front $ long term	Large data + moderate compute	Limited compute capacity

Where to Store

	<10 TB ~150 MB/s	<50 TB ~1500 MB/s	<1 PB	Backup
Local Workstation	$			
Local Workstation (RAID)	$$	$		
Local Storage (NAS)	$$	$-$$ (but slow via net)	$	$
CIBR (cluster use only)	Free (limited)	Free (limited)		
IT Storage	???	???	???	???
Cloud		$$$ (slow outside cld)	$$$	$$

Part 3
Using Clusters

The Command-Line

If you are not comfortable with the UNIX command-line you will be forever limited to applications pre-configured for you by others (mostly in the cloud).

Simple Unix Commands

- ssh - log in to a remote computer
- passwd - Change your password. Beware on clusters!
- man - manual for other commands
- ls - list current directory (folder)
- cd - change directory
- pwd - print working directory (where are you now)
- cp - copy a file
- mv - rename a file
- rm - delete a file
- cat, more, less - display text files on the terminal
- nano - text editor, easy, almost ubiquitous
- vi/vim - text editor, difficult to use, but ubiquitous

Other Important Facts

- echo $<NAME> - show a single environment variable
- set - show all environment variables
- export <NAME>=<VALUE>
- $PATH - list of directories containing programs to run
- $HOME - home directory. Also "~"
- bash - typical default shell, others: csh, tcsh, zsh, ...
- files starting with '.' are hidden, use 'ls -al'

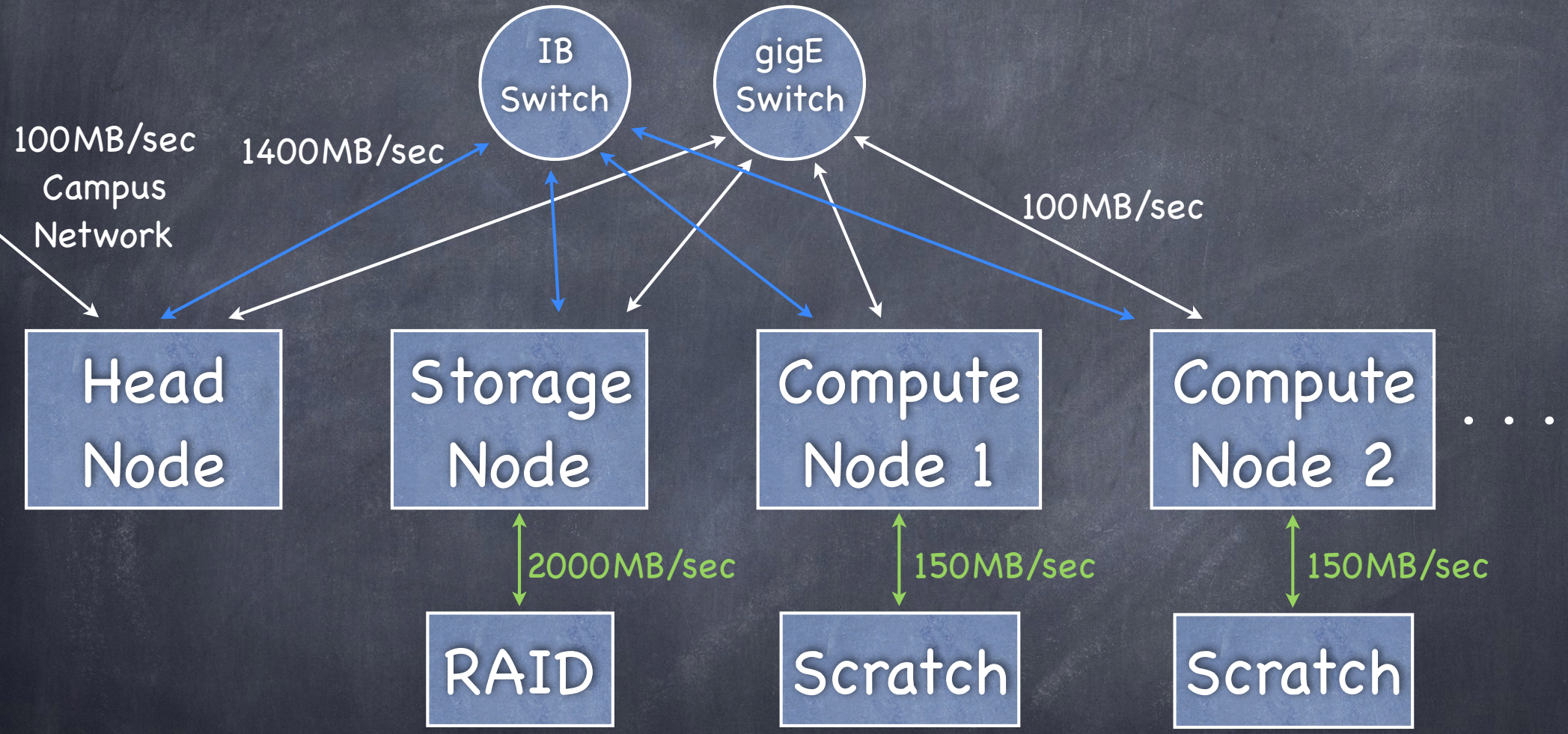
SSH

- Virtually all clusters will be accessed using SSH
- ssh <hostname>
- ssh-keygen - to configure easier access/secure clusters
- password for the machine
- passphrase for the key (may be empty)
- some machines require both!
- $HOME/.ssh - where keys/config live, both ends

SSH

- in $HOME/.ssh
  - id_rsa - private! On machine logging in FROM
  - id_rsa.pub - copy contents to authorized_keys on machine you want to log in TO
  - authorized_keys - a list of id_rsa.pub lines for all of the machines you permit access FROM
- ssh-agent - Permits keys with non-empty passphrases, but saves you from typing the passphrase over and over

Hypothetical Cluster



Linux Clusters

- Every cluster is different. READ POLICIES!
- When you log into a cluster, you are logging in to a head node
- head nodes are used for compiling/installing software, configuring your account, and queueing jobs. NEVER run a job directly on the head-node.
- Compute nodes are allocated, and jobs run by a BQS (Batch Queuing System)
- Some clusters permit direct login to compute nodes via SSH. This is for checking/debugging jobs, NOT running them!

How to

Interrogate Your Cluster !

Cluster Resources

- RTFM (http://blake.bcm.edu/CIBRClusters)
- more /etc/hosts
- df -h
- mount
- ifconfig or 'ip addr'
- /proc filesystem (cpuinfo, meminfo)
- 'qstat -q' or sinfo
- Filesystem speed ?
- dd if=/dev/zero of=tst bs=1M count=2000; sync; rm tst
- Some clusters have a 'module' system (TACC)

Subsystems

- BQS (Batch Queuing System)
  - SLURM
  - PBS (OpenPBS, Torque, etc.)
  - SGE (Sun Grid Engine)
  - HTCondor (UW)
- Parallelizing programs
  - pthreads
  - OpenMP
  - MPI

OpenPBS/Torque

(CIBR prism cluster)

- Edit batch script
- Submit job (qsub)
  - to a specific queue (qstat -q)
- Job waits in queue (qstat -a)
- Nodes allocated ($PBS_NODEFILE)
- Script run on the first node ($PBS_O_WORKDIR)
- Cleanup/logfiles
  - Kill a bad job (qdel)
- Accounting updated (resources used)

Batch Script

```
#!/bin/bash
#
# This is an example PBS/Torque script
# modify the number of nodes, ppn (processors per node), and walltime
#

#PBS -l nodes=2:ppn=12
#PBS -l walltime=2:00:00

cd $PBS_O_WORKDIR

YOUR COMMANDS HERE
```

qsub -q <queuename> myscript.pbs

SLURM

(CIBR sphere cluster)

- Edit batch script
- Submit job (sbatch)
  - to a partition (sinfo, scontrol show partition)
- Job waits in queue (squeue)
- Nodes allocated ($SLURM_JOB_NODELIST)
- Script run on the first node ($SLURM_SUBMIT_DIR)
- or run tasks (srun)
- Cleanup/logfiles
  - Kill a bad job (scancel)

Batch Script

```
#!/bin/sh
#SBATCH --time=1:00 -n48 -p debug

cd /home/stevel/test

srun -c 1 -n 48 python test.py
```

test.py

```
import time
import socket
import sys
import os

t=time.localtime()
print "{}: {}\t{:02d}:{:02d} - ".format(os.getenv("SLURM_PROCID"),
socket.gethostname(),t.tm_min,t.tm_sec),
time.sleep(10)
t=time.localtime()
print "{:02d}:{:02d}".format(t.tm_min,t.tm_sec)
```

Efficient Cluster Use

- Many jobs can only work on full nodes. TACC clusters only allocate whole nodes at a time.
- If you can run your job multithreaded, do that. Use one node per job, using all threads available.
- If you have a lot of small jobs to run which take a similar amount of time, consider grouping them into node-sized sets, and queuing whole-node jobs.
- NEVER launch hundreds of single-core jobs all at the same time.

Being a good citizen

- Virtually all clusters prohibit running programs that automatically queue cluster jobs
- CIBR clusters need to balance big jobs and small jobs
  - When sphere is busy, single user can have at most 10 nodes (240 cores)
  - If idle for a few hours, ok to use up to 240 more, but only for short (<8 hour) jobs
- Every cluster will have its own policies. Read them!

Part 4
Parallel Programming
(a very quick introduction)

Parallel programming

- OpenMP
- pthreads
- MPI
- Other niche systems...

OpenMP

- Very good speedups with limited effort

  (somewhat steep learning curve)
- Same code can compile parallel and serial
- One node only (SMP - symmetric multiprocessing)
- Needs to be part of the compiler (available in gcc)
- ie - no python/ruby

- http://www.openmp.org

pthreads

- Only one node at a time (SMP)
- shared memory -> easy communications
- Somewhat painful to program
  - Synchronization issues
  - May be limits in some languages (Python, Ruby)
- Available in multiple programming languages

pthreads

Python example:

```
from threading import Thread
import time,sys

def func(n):
        for i in range(10):
                time.sleep(1)
                print(n,i)
                sys.stdout.flush()

threads=[Thread(target=func,args=[i]) for i in range(4)]

for t in threads:
        t.start()
        time.sleep(0.1)
```

MPI

- MPI: Message Passing Interface
- Written by computer scientists for computer scientists
- Operates on distributed processors
- Bindings for many languages
- Explicit interprocess communication via messages
- All nodes run the same program
- Communications problems are common
- Zero fault tolerance

MPI

- Many variants - OpenMPI, MPICH, Intel MPI,...

- mpicc - MPI aware C compiler
- which mpicc - identify which MPI installation
- mpirun - convenient program launching tool
  - Runs the exact same program on each processor!
- On most clusters, automatically talks to BQS

MPI

- Standard API, mpi4py different syntax
  - Outline of one strategy for MPI program:
  - MPI_Init() - Initialize MPI on all nodes
  - MPI_Barrier() - Synchronize nodes
  - MPI_Comm_rank() - Identify CPU (rank)
  - rank 0:
    - coordinate processing, perhaps do some
  - rank 1-n:
    - perform work assigned by rank 0
  - MPI_finalize() - clean everything up

MPI

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int rank;
  char hostname[256];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  gethostname(hostname,255);

  printf("Hello world!  I am process number: %d on host %s\n", rank, hostname);

  MPI_Finalize();

  return 0;
}
```

https://hpcc.usc.edu/support/documentation/examples-of-mpi-programs/

SLURM

```
#!/bin/sh
#SBATCH --time=18:00:00 -n48 -p bynode

cd /home/stevel/test2
mpirun ./testmpi
```

https://hpcc.usc.edu/support/documentation/examples-of-mpi-programs/

Improved Sieve

```
from time import time
from math import *
t0=time()

N0=2
N1=10000000
primes=set(range(N0,N1))
for i in range(2,int(sqrt(N1)+1)):
  start=N0-N0%i
  if start<i*2 : start=i*2
  bad=range(start,N1+i,i)
  primes.difference_update(bad)

t1=time()
primes=list(sorted(primes))
print(primes[:10],primes[-10:])
print(t1-t0)
```

MPI Sieve

```
from mpi4py import MPI
from time import time
import sys
from math import *

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
if size==1 : sys.exit(1)

if rank == 0:
    print("Starting")
    t0=time()
    primes=set()
    for i in range(1,size):
        primes.update(comm.recv(source=i, tag=1))
    t1=time()
    primes=list(sorted(primes))
    print(primes[:10],primes[-10:])
    print(t1-t0)
else:
    N0=(rank-1)*10000000
    N1=N0+10000000
    primes=set(range(N0,N1))
    for i in range(2,int(sqrt(N1)+1)):
        start=N0-N0%i
        if start<i*2 : start=i*2
        bad=range(start,N1+i,i)
        primes.difference_update(bad)
    comm.send(primes,dest=0,tag=1)
```

Other systems?

- Many other language dependent systems
- May not be broadly supported on 'big iron' clusters
- Sysops may be hostile to use of anything but MPI

Where to Learn More

- Passing interest
  - Youtube has many good videos
- Somewhat interested
  - TACC offers multi-day workshops on parallelism
- Really Committed
  - Rice offers Comp 422, a full semester course on parallel computing
  - iTunesU - full courses (eg - Stanford)