Problem Solving With Large Clusters

What's the problem, and what resources do we have?

Game plan for today:

Structure of the course

Overview of parallel and distributed computing

Quick intro to distributed file systems

Do you actually need a cluster?

http://cslu.ohsu.edu/~bedricks/courses/cs624/

- Focus: Conventional vs. distributed algorithm
- Problem: What is the problem? Why is it important?
- Background: What are the conventional algorithms? You may ignore the specifics of the application area.
- Distributed Algorithm: Details, assumptions and advantages
- Evaluation: Experimental paradigm, corpus
- Results: Outcomes, analysis and discussion

Game plan for today:

Structure of the course

Overview of parallel and distributed computing

Quick intro to distributed file systems

Do you actually need a cluster?

The problem:

Many things we might want to do with computers take a long time.

Why?

Trivial answer: they require the computer to do a lot of work.

The problem:

This can be for two main reasons:

1. We are working with a lot of data
2. We have to do a lot of computations on each chunk of data.

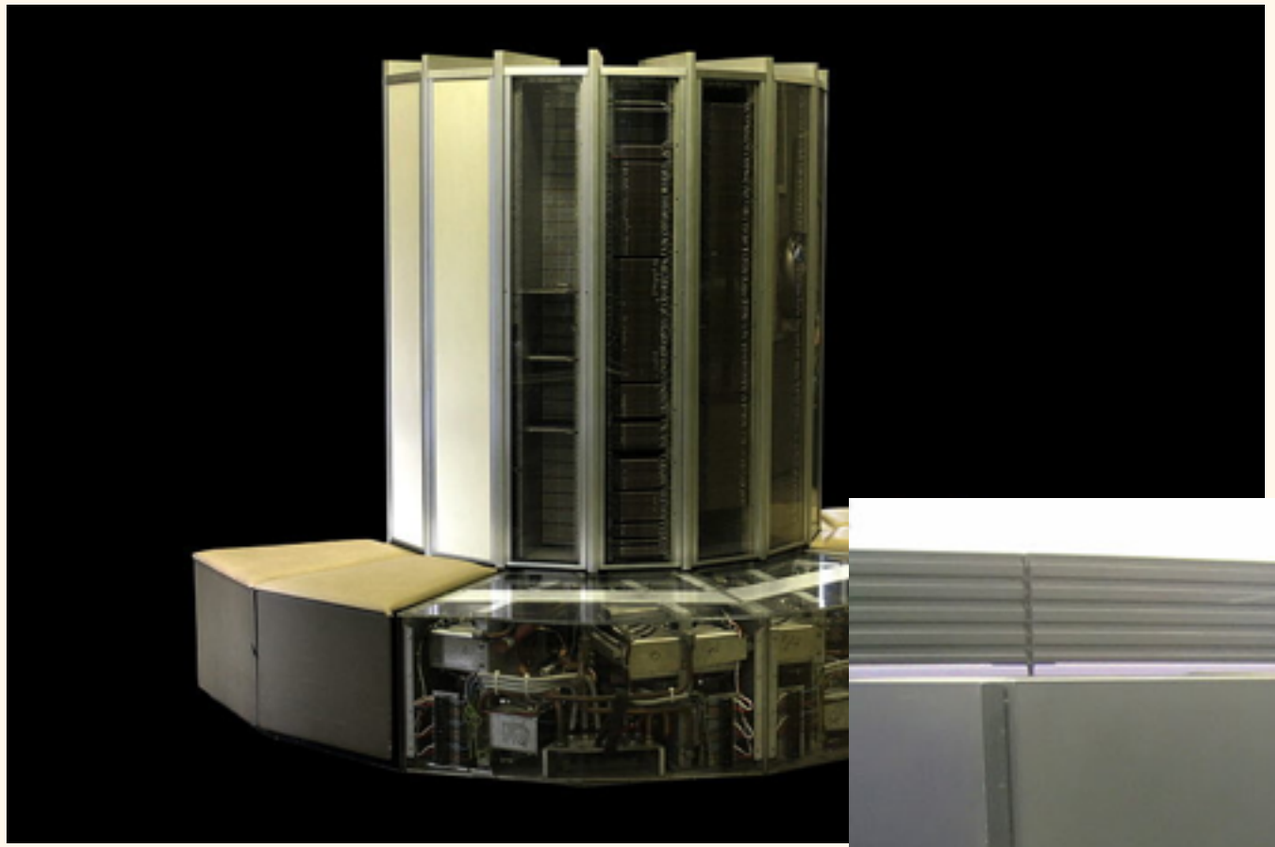
A common solution: split the work up!

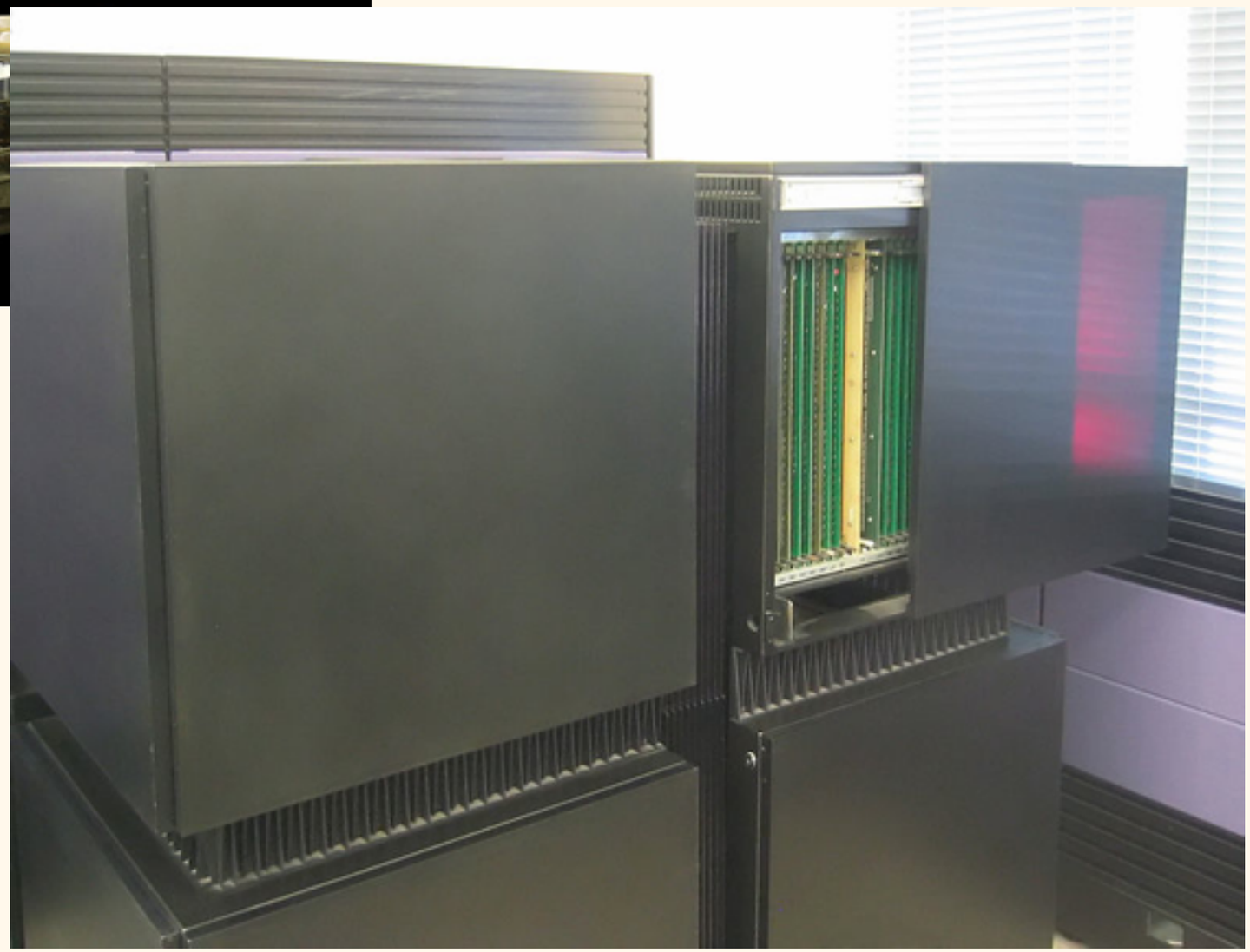
A common solution: split the work up!

1. Do parts of the computation in parallel (less work per processor)

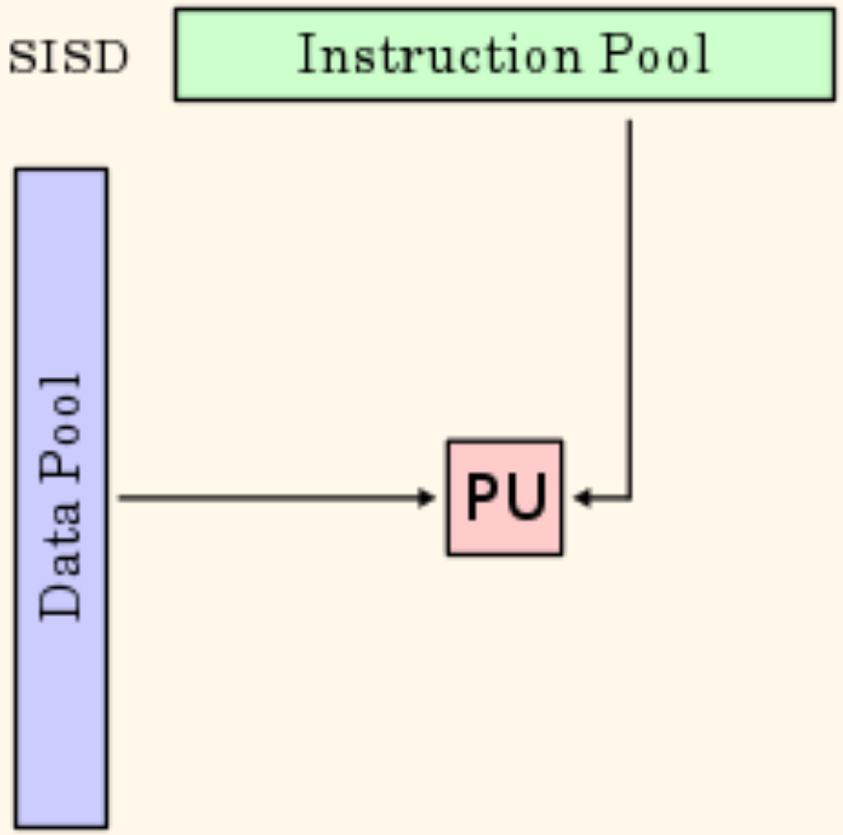
2. Split the data onto multiple computers (less data per processor)

Often, we (try to) do both!

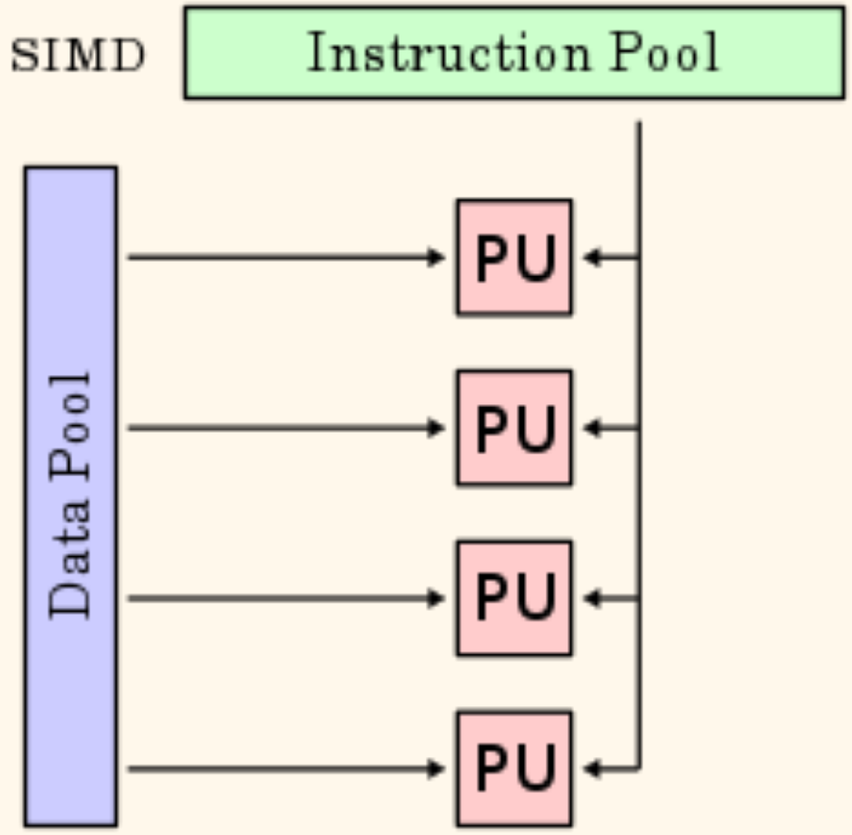




Ultimately, it all comes down to feeding instructions and data to processors:



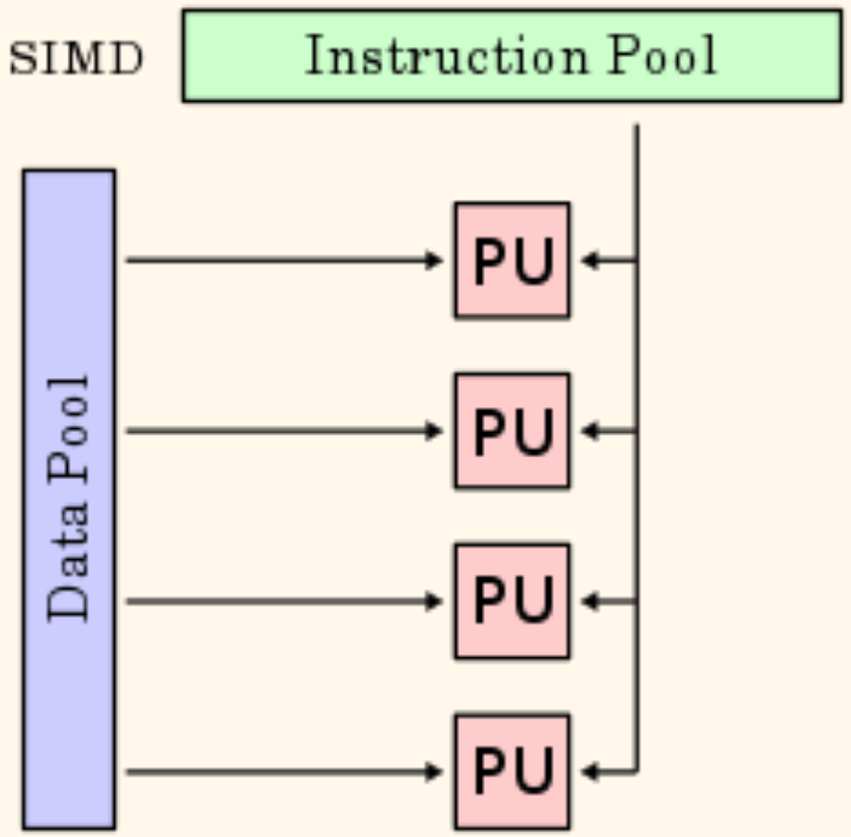
Single Instruction, Single Data



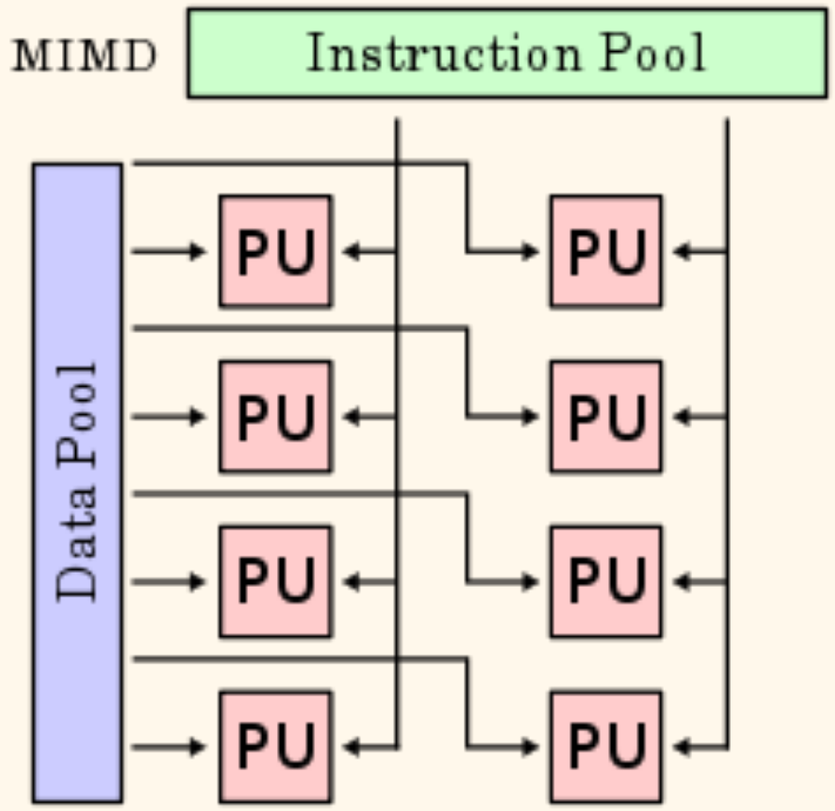
Single Instruction, Multiple Data

Most modern CPUs are SIMD (SSE3, etc.)...

Ultimately, it all comes down to feeding instructions and data to processors:



Single Instruction, Multiple Data

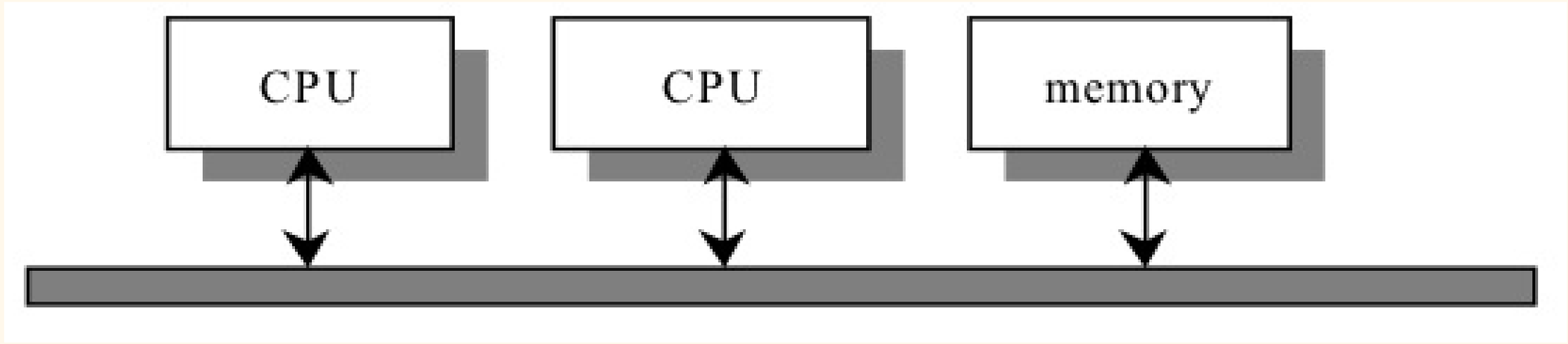


Multiple Instruction, Multiple Data

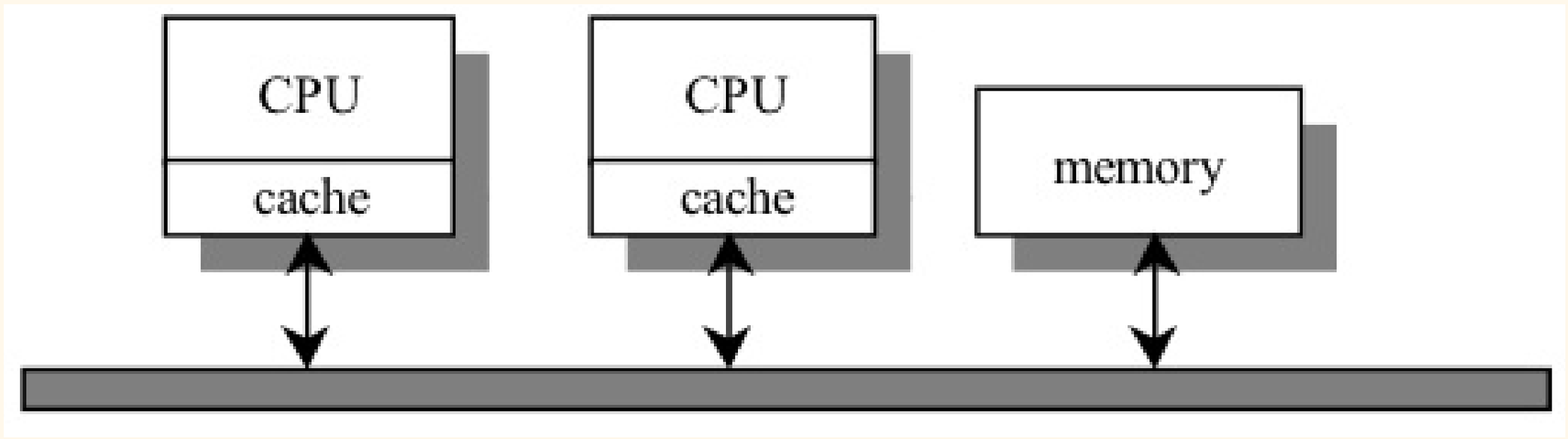
Some architectures are MIMD, e.g. Intel "Xeon Phi" and most modern parallel machines.

A single computer can have more than one CPU...

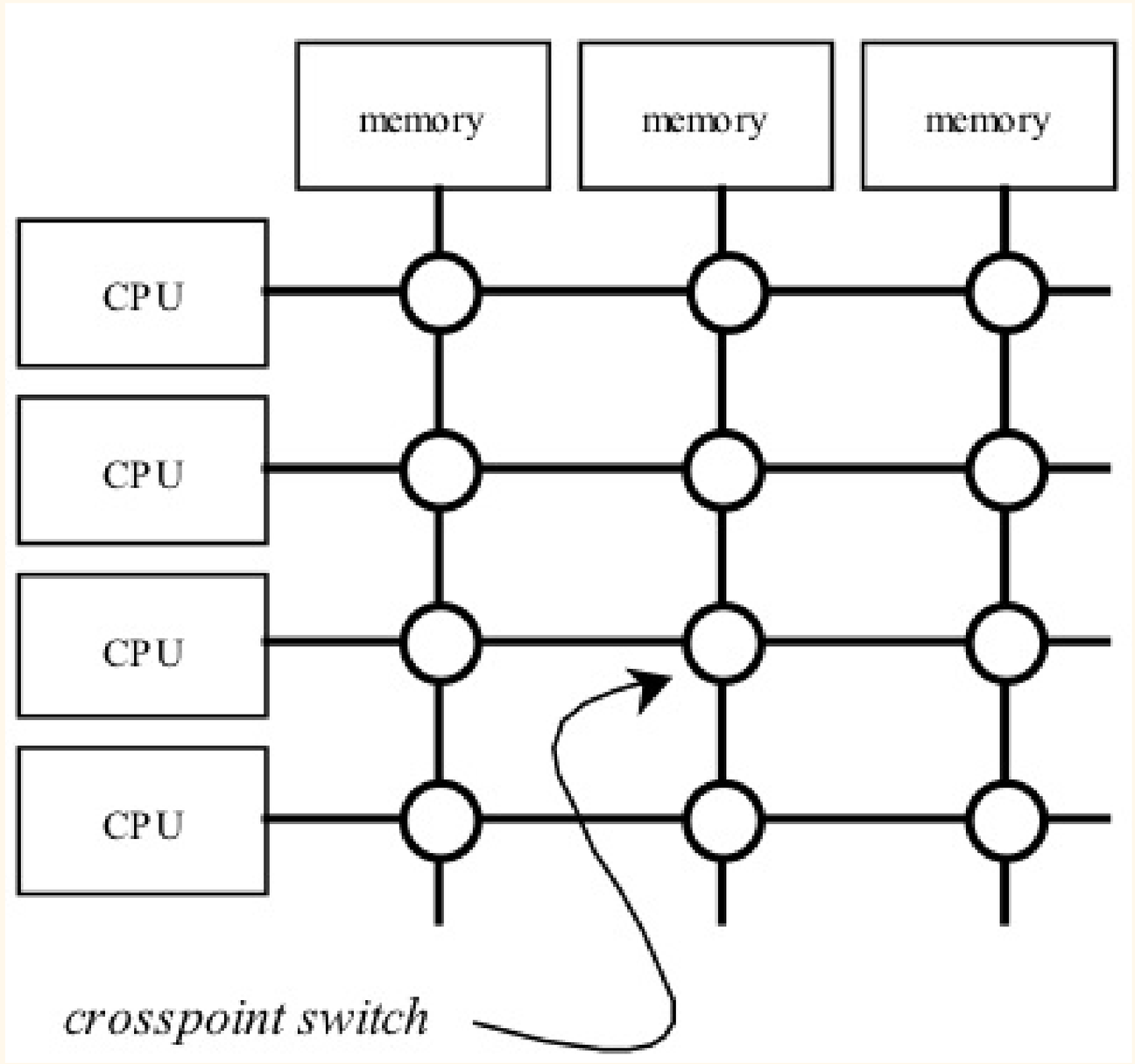
As a single bus:



As a single bus w/ cache:

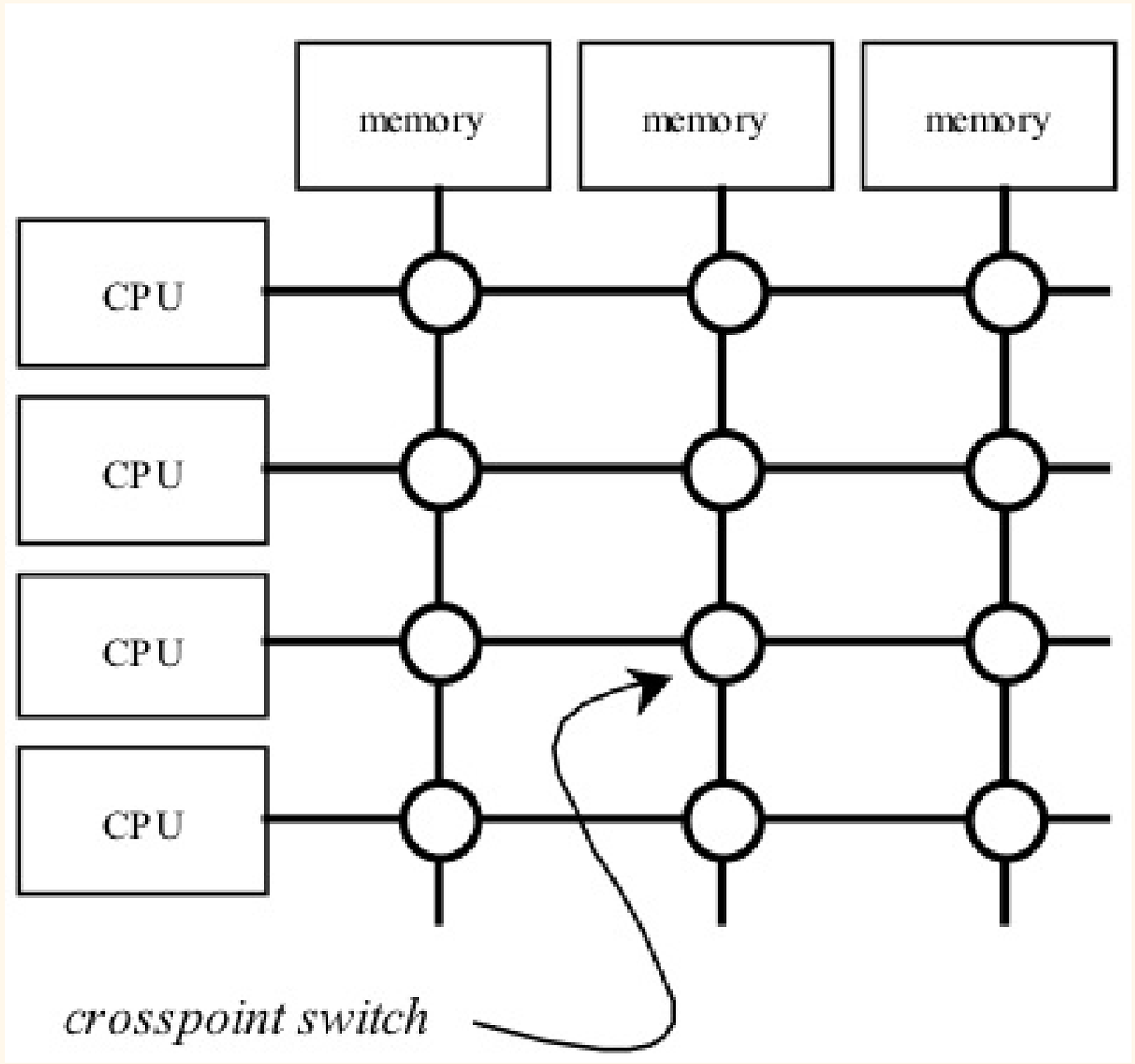


The question becomes: how to share memory across many CPUs?



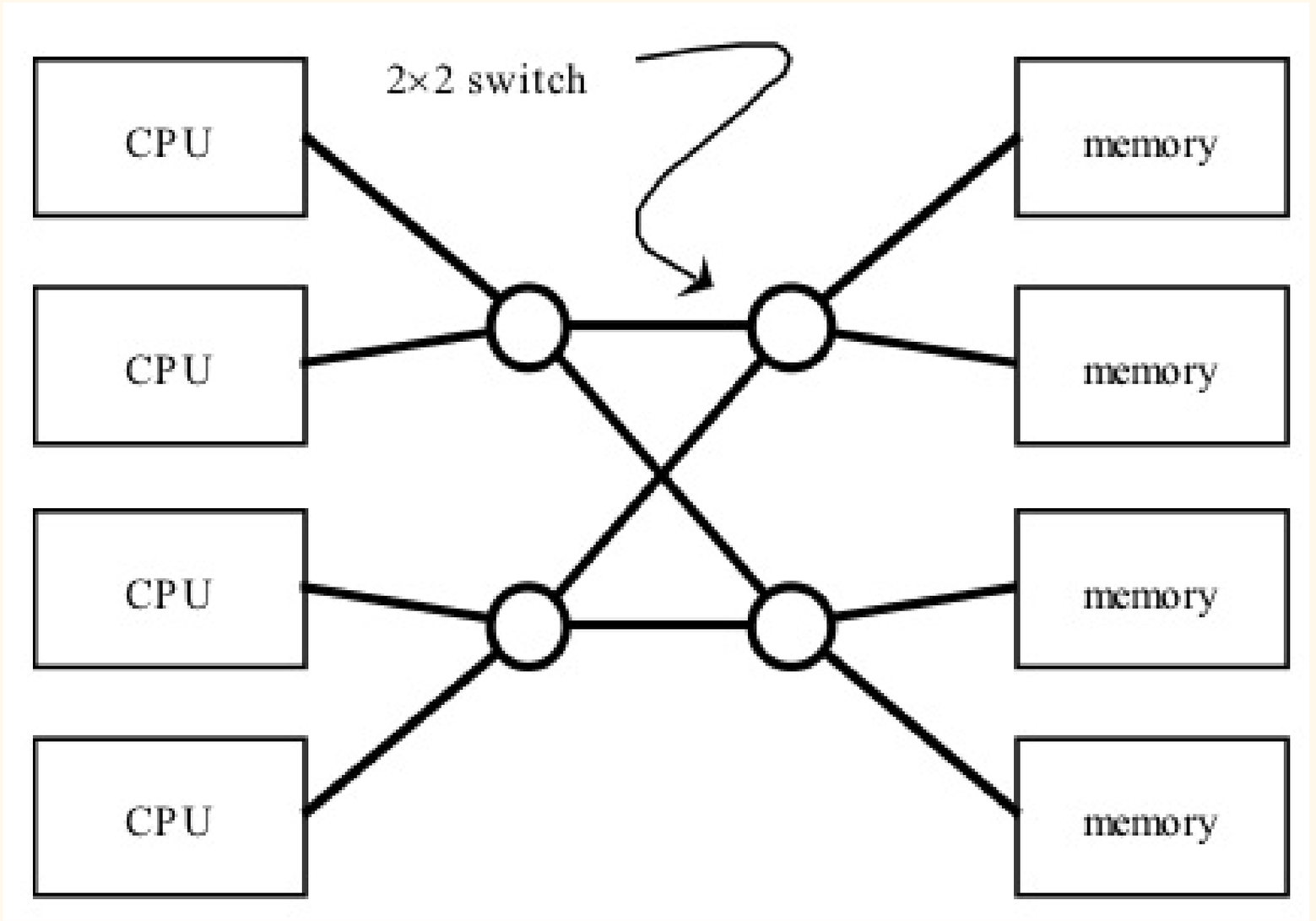
A crossbar topology is simple, but has many expensive* swtiches.

The question becomes: how to share memory across many CPUs?



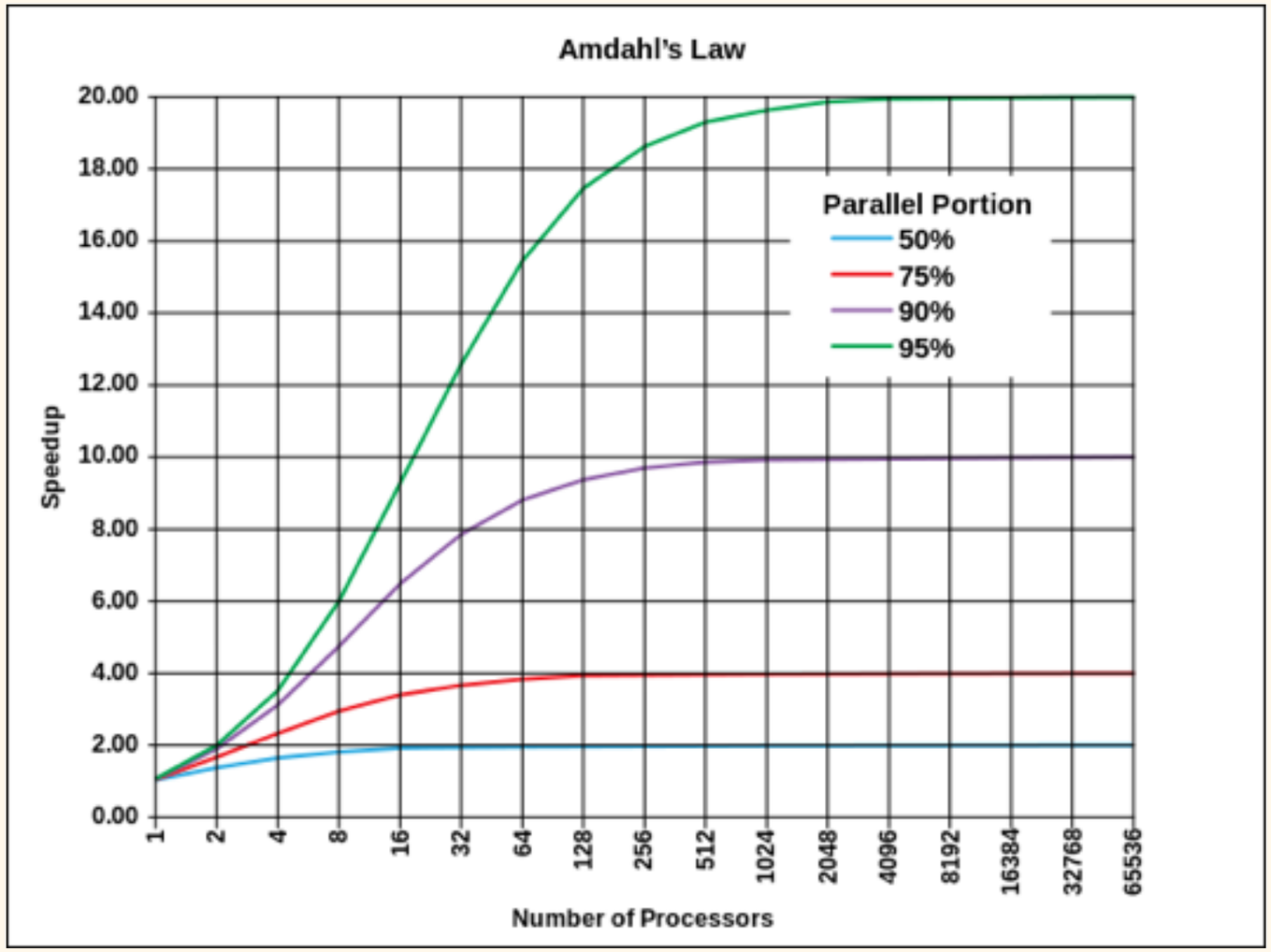
Expensive in terms of both time and silicon!

The question becomes: how to share memory across many CPUs?



Non-uniform memory access (NUMA) is a common compromise (Intel Nehalem, Westmere, etc.).

In the real world, there is never a linear speedup with an increase in CPUs.



$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right)$$

T: time taken

n: num. threads

B: proportion of algorithm that is strictly serial.

Amdahl's law states that the maximum speedup is related to the fraction of a program's work that is serial.

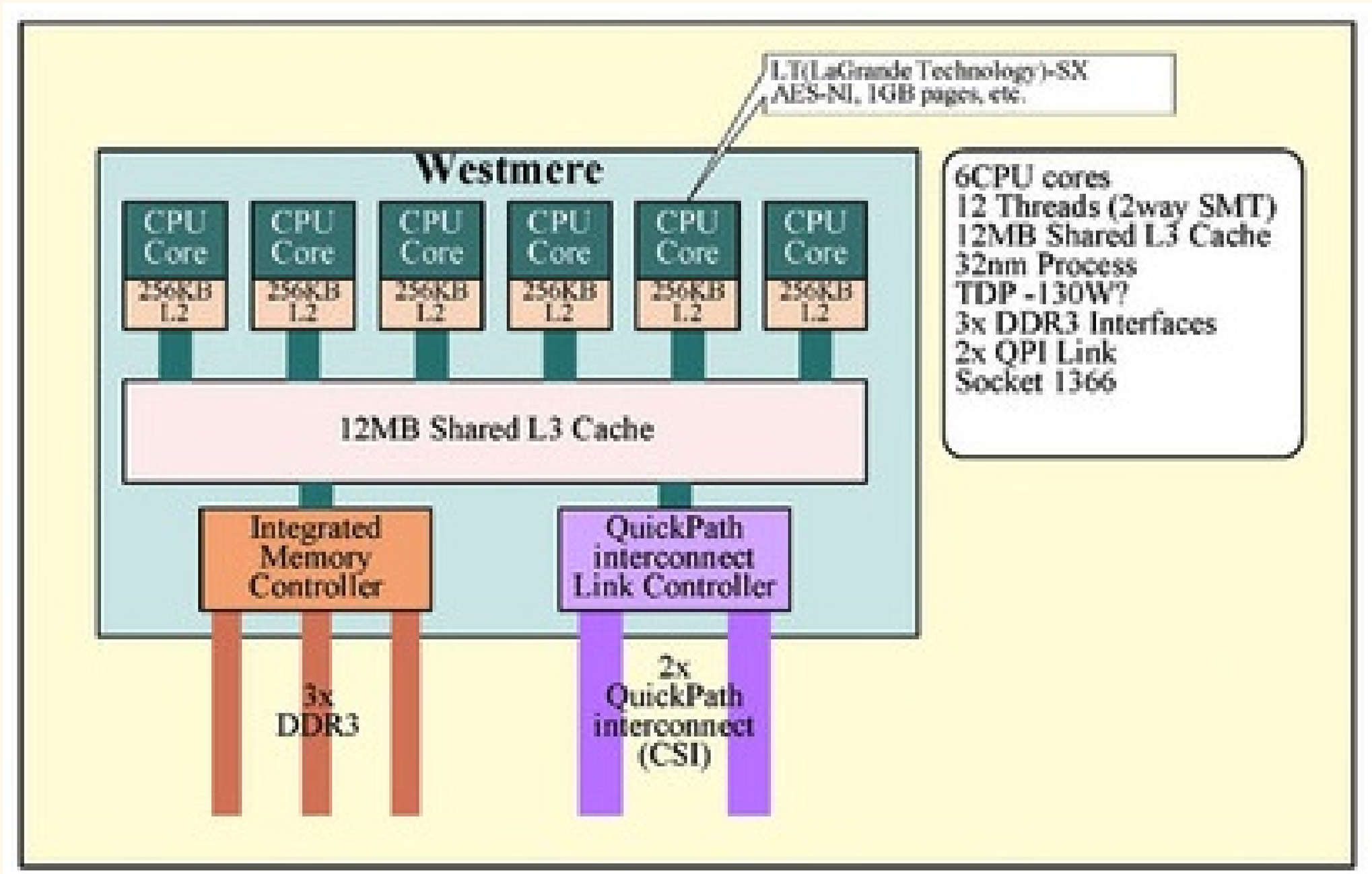
Other holdups include cache stalls, disk latency, etc. etc.

There are also the dreaded “fallacies of distributed computing” to keep in mind...

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

So, what do we have at CSLU?

The OHSU *bigbird* cluster uses Westmere CPUs; 2x per node.



The bigbird cluster has 56 accessible nodes (`bigbirdXX.cslu.ohsu.edu`).

Each node uses Westmere CPUs; 2x per node (total of 24 logical cores).

Each node has 48 GB of RAM, and all share a large distributed file system.

The bigbird cluster *also* has 8 additional nodes (`bigbird61-68.cslu.ohsu.edu`).

Each node uses "Gulftown" CPUs; 2x per node (total of 16 logical cores).

Each node has 16 GB of RAM, and all are equipped with solid-state storage.

Game plan for today:

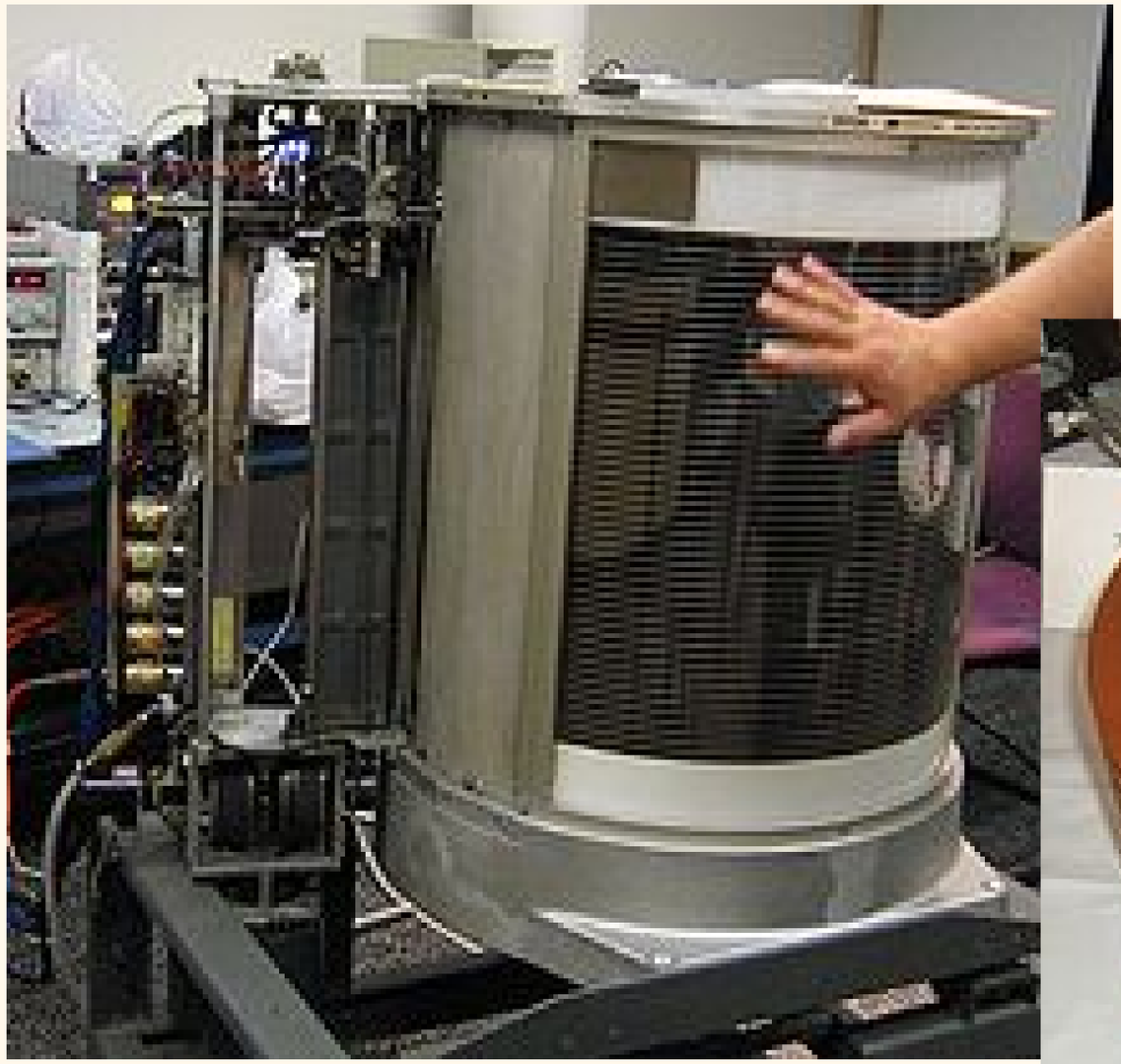
Structure of the course

Overview of parallel and distributed computing

Quick intro to distributed file systems

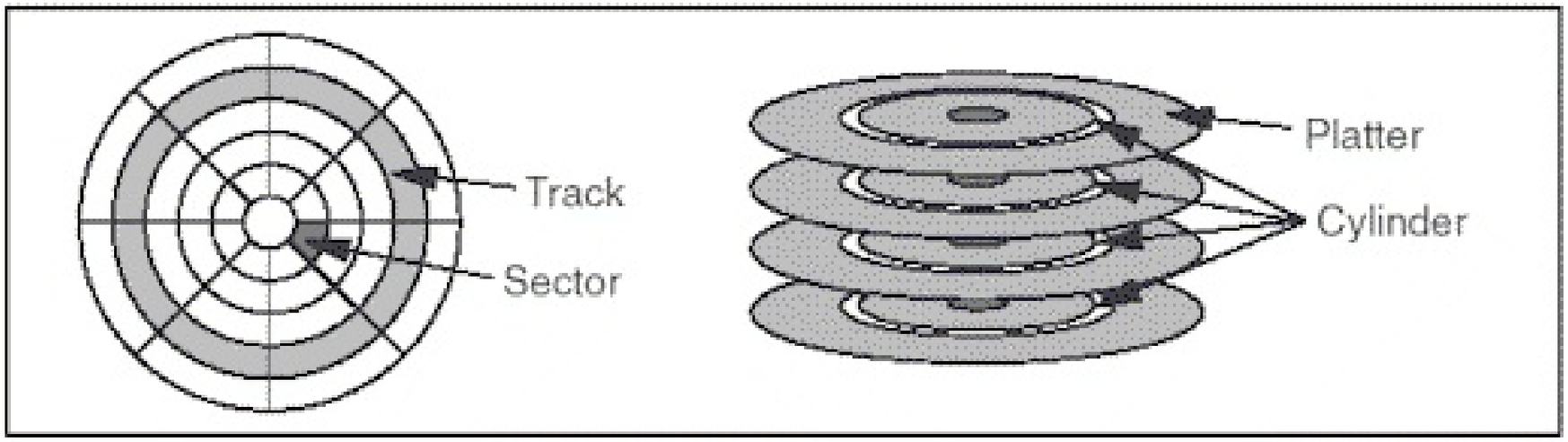
Do you actually need a cluster?

Let’s talk about storage.



Most hard drives are still mechanical.

- Hard disk has a number of disks
- Each disk segmented into tracks and sectors



- Disk speed: access time = seek time + latency time
  - Seek time: Time required to bring the head to the track
  - Latency time: Time required for the sector to reach the head
  - Platters spin about 7k to 15k rpm
  - Disk-to-buffer about 1Gbits/s, depends on track

Modern SSDs avoid these problems, but introduce others: Cost, limited life-span, etc.

“Most hard drives are still mechanical.”

Is this still true?

In your laptop: no.

In the data center, a qualified “yes.”

In the data center, a qualified “yes.”

SSDs are:

Much more expensive ($/gigabyte)

Relatively fragile

A *whole lot* faster*

(depending on your workload, etc.)

Key findings:

Temperature-sensitive

Lots of manufacturing variability

Failure types and frequencies depend highly on use patterns

Caching, buffering, and wear-reduction makes diagnosis tricky.

A Large-Scale Study of Flash Memory Failures in the Field

Justin Meza
Carnegie Mellon University
meza@cmu.edu

Qiang Wu
Facebook, Inc.
qwu@fb.com

Sanjeev Kumar
Facebook, Inc.
skumar@fb.com

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

ABSTRACT

Servers use flash memory based solid state drives (SSDs) as a high-performance alternative to hard disk drives to store persistent data. Unfortunately, recent increases in flash density have also brought about decreases in chip-level reliability. In a data center environment, flash-based SSD failures can lead to downtime and, in the worst case, data loss. As a result, it is important to understand flash memory reliability characteristics over flash lifetime in a realistic production data center environment running modern applications and system software.

This paper presents the first large-scale study of flash-based SSD reliability in the field. We analyze data collected across a majority of flash-based solid state drives at Facebook data centers over nearly four years and many millions of operational hours in order to understand failure properties and trends of flash-based SSDs. Our study considers a variety of SSD characteristics, including: the amount of data written to and read from flash chips; how data is mapped within the SSD address space; the amount of data copied, erased, and discarded by the flash controller; and flash board temperature and bus power.

Based on our field analysis of how flash memory errors manifest when running modern workloads on modern SSDs, this paper is the first to make several major observations: (1) *SSD* failure rates do *not* increase monotonically with *flash chip* wear; instead they go through several distinct periods corresponding to how failures emerge and are subsequently detected, (2) the effects of read disturbance errors are *not* prevalent in the field, (3) sparse logical data layout across an SSD's physical address space (e.g., non-contiguous data), as measured by the amount of metadata required to track logical address translations stored in an SSD-internal DRAM buffer, can greatly affect SSD failure rate, (4) higher temperatures lead to higher failure rates, but techniques that throttle SSD operation appear to greatly *reduce* the negative reliability impact of higher temperatures, and (5) data written by the operating system to flash-based SSDs does *not* always accurately indicate the amount of wear induced on flash cells due to optimizations in the SSD controller and buffering employed in the system software. We hope that the findings of this first large-scale flash memory reliability study can inspire others to develop other publicly-available analyses and novel flash reliability solutions.

SIGMETRICS'15, June 15–19, 2015, Portland, OR, USA.
ACM 978-1-4503-3486-0/15/06.
http://dx.doi.org/10.1145/2745844.2745848.

Categories and Subject Descriptors

B.3.4. [**Hardware**]: Memory Structures—*Reliability, Testing, and Fault-Tolerance*

Keywords

flash memory; reliability; warehouse-scale data centers

1. INTRODUCTION

Servers use flash memory for persistent storage due to the low access latency of flash chips compared to hard disk drives. Historically, flash capacity has lagged behind hard disk drive capacity, limiting the use of flash memory. In the past decade, however, advances in NAND flash memory technology have increased flash capacity by more than 1000×. This rapid increase in flash capacity has brought both an increase in flash memory use and a decrease in flash memory reliability. For example, the number of times that a cell can be reliably programmed and erased before wearing out and failing dropped from 10,000 times for 50 nm cells to only 2,000 times for 20 nm cells [28]. This trend is expected to continue for newer generations of flash memory. Therefore, if we want to improve the operational lifetime and reliability of flash memory-based devices, we must first fully understand their failure characteristics.

In the past, a large body of prior work examined the failure characteristics of flash cells in controlled environments using small numbers e.g., tens) of raw flash chips (e.g., [36, 23, 21, 27, 22, 25, 16, 33, 14, 5, 18, 4, 24, 40, 41, 26, 31, 30, 37, 6, 11, 10, 7, 13, 9, 8, 12, 20]). This work quantified a variety of flash cell failure modes and formed the basis of the community's understanding of flash cell reliability. Yet prior work was limited in its analysis because these studies: (1) were conducted on small numbers of raw flash chips accessed in adversarial manners over short amounts of time, (2) did not examine failures when using real applications running on modern servers and instead used synthetic access patterns, and (3) did not account for the storage software stack that real applications need to go through to access flash memories. Such conditions assumed in these prior studies are substantially different from those experienced by flash-based SSDs in large-scale installations in the field. In such large-scale systems: (1) real applications access flash-based SSDs in different ways over a time span of *years*, (2) applications access SSDs via the storage software stack, which employs various amounts of buffering and hence affects the access pattern seen by the flash chips, (3) flash-based SSDs employ aggressive techniques to reduce device wear and to correct errors, (4) factors in platform design, including how many SSDs are present in a node, can affect the access patterns to SSDs, (5) there can be significant variation in reliability due to the existence of a very large number of SSDs and flash chips. All of these real-world conditions present in large-scale

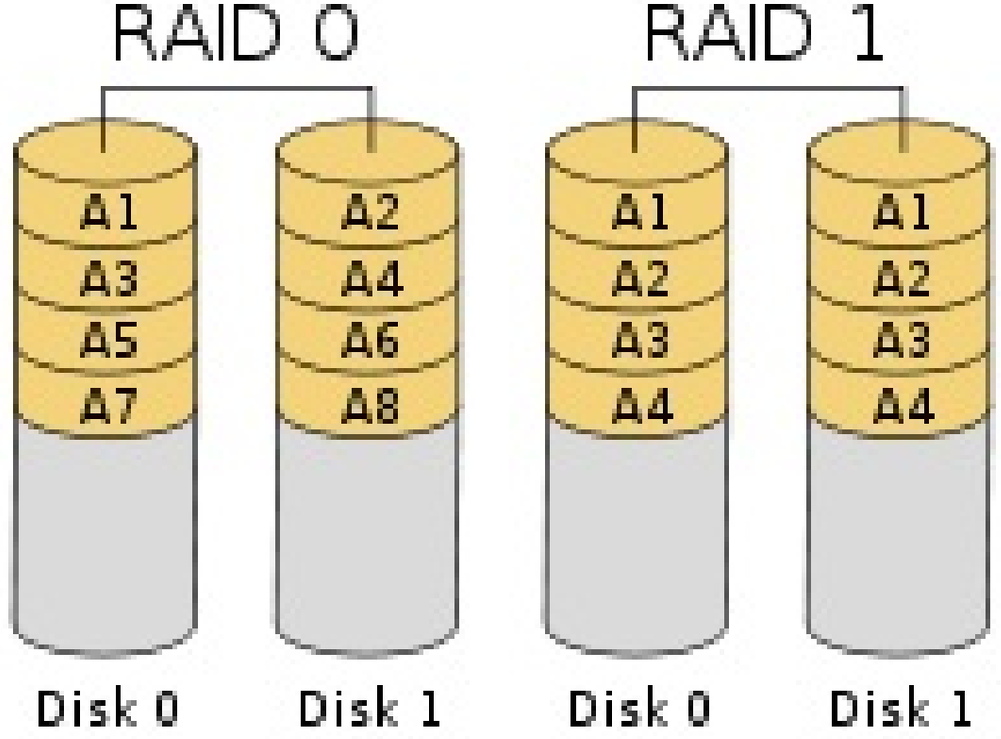
There are many consequences of the mechanical nature of hard disks:

Reading/writing a small number of large files is far faster than reading/writing a large number of small files.

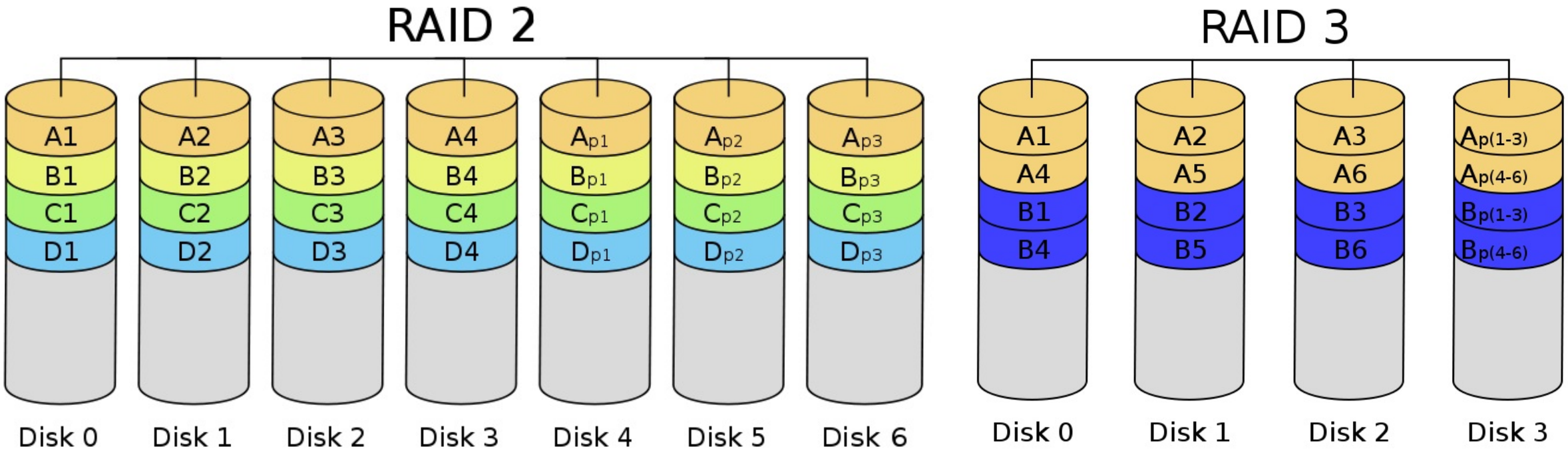
+Δ Moving parts -> +Δ things that can break.

(Patterson, Gibson and Katz, 1987)

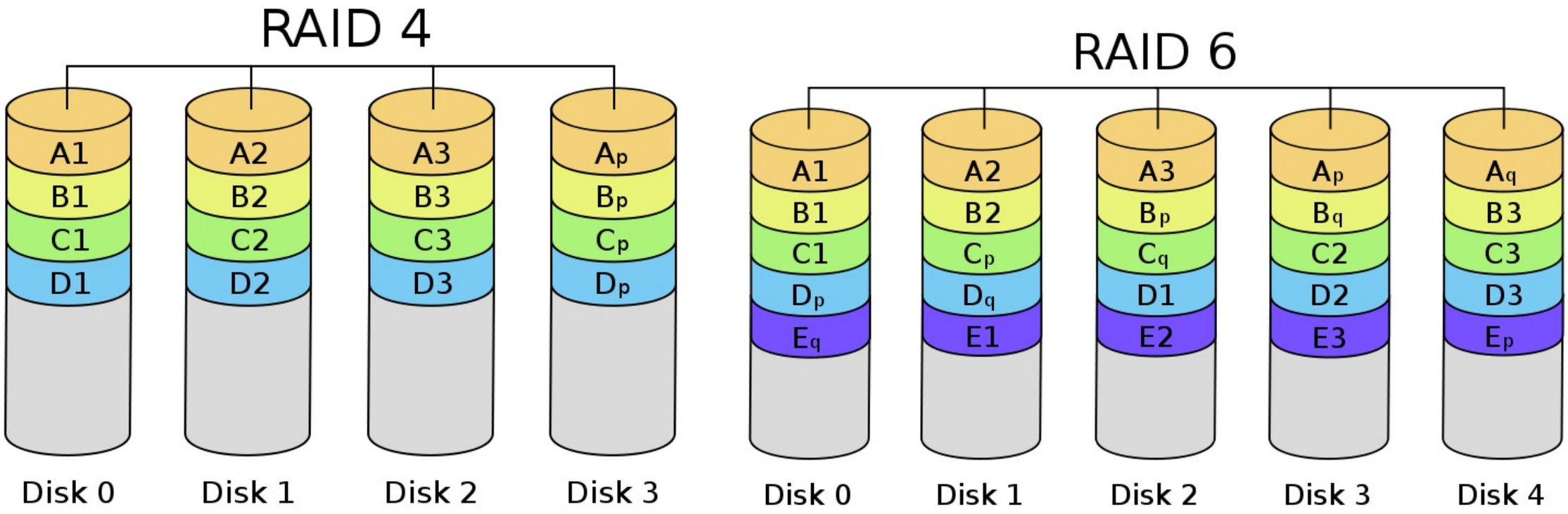
- Cost-effective to build capacity with many cheaper disks
- Divide the file into stripes, saved on independent disks
- Better performance by putting all the disks to work
- Compensate for higher failure rates with redundancy or parity
- RAID0: block level striping, zero redundancy, read nX
- RAID1: full mirroring, read nX, write 1x



- RAID2: bit-level, parity, sync-ed spindles
- RAID3: byte-level, parity, sync-ed spindles

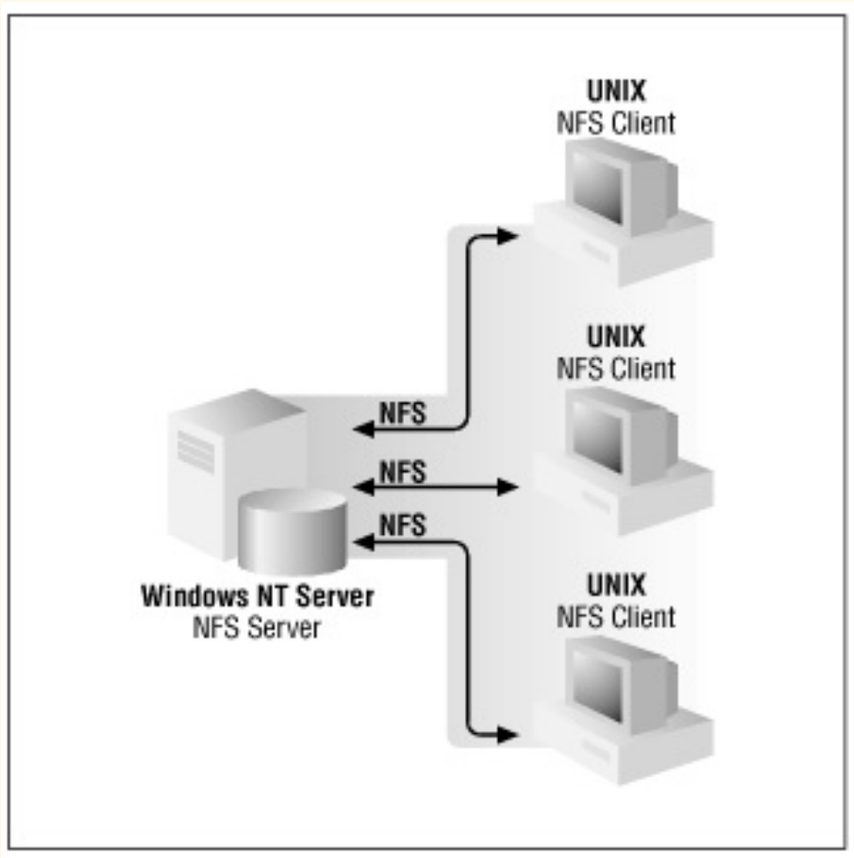


- RAID4: block-level, dedicated parity
- RAID6: block-level, doubly distributed parity



That's all well and good if you've only got one machine...

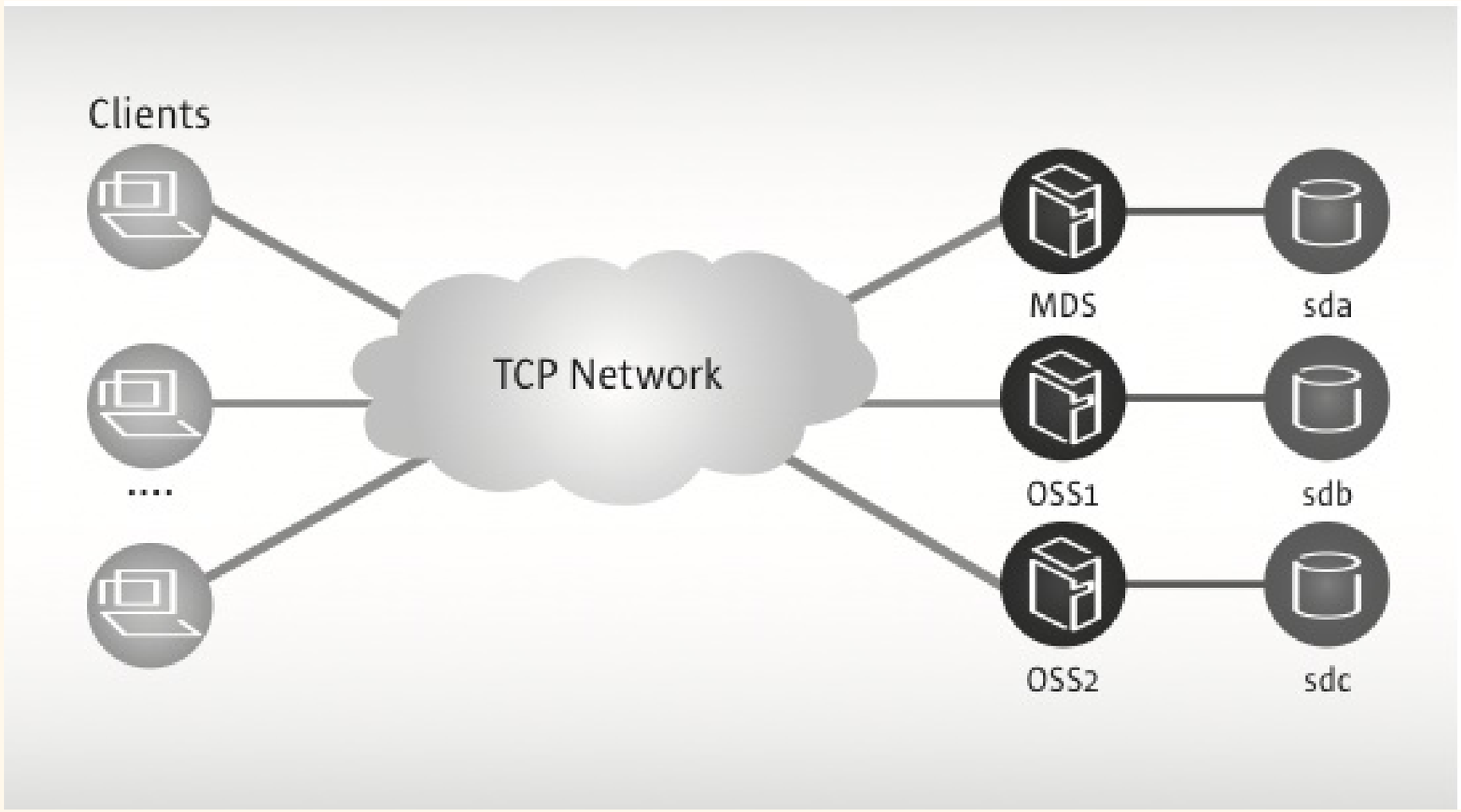
... but what if you need to share a disk array with more than one machine, over a network?

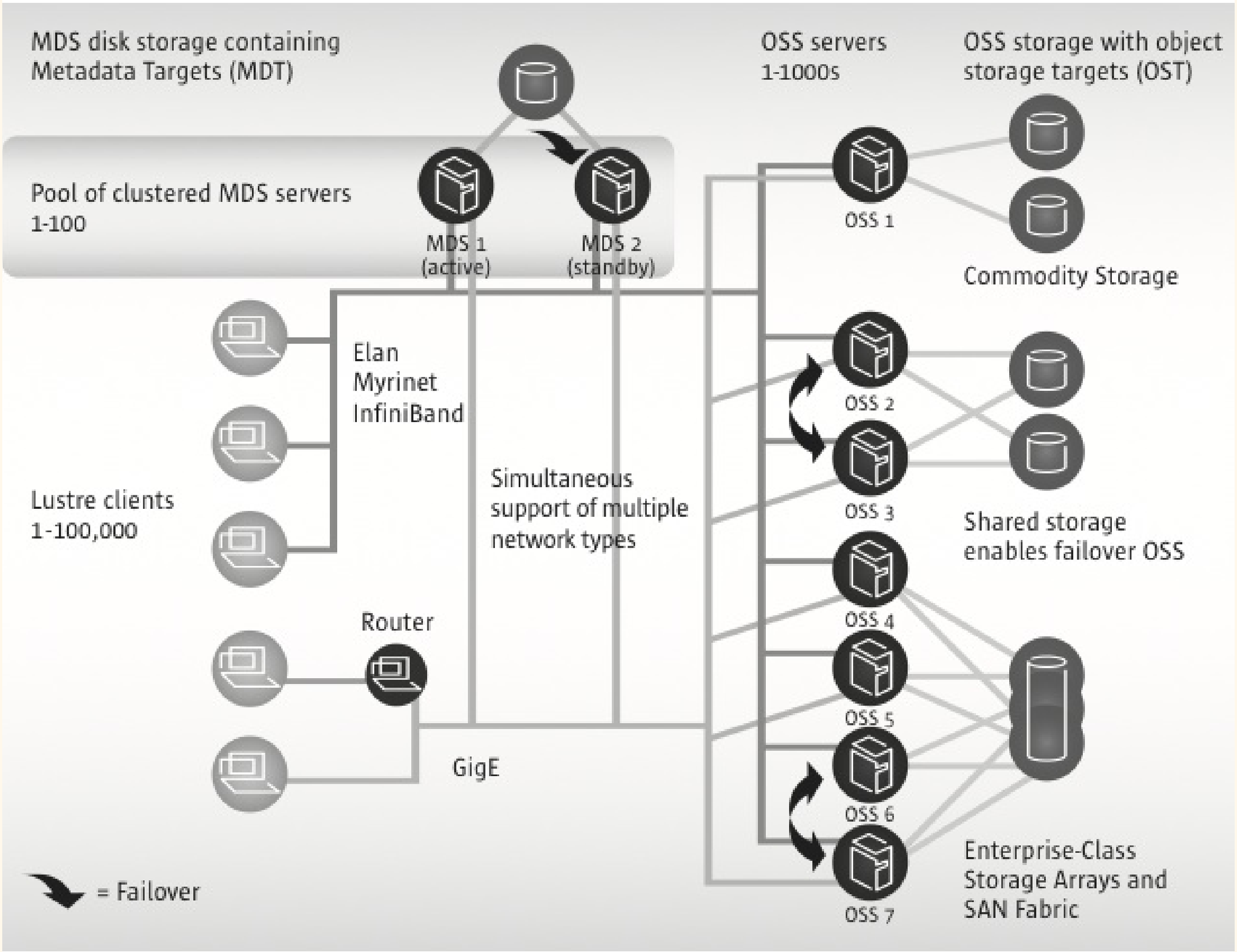


NFS gives file-level access, with server-side caching and coherency.

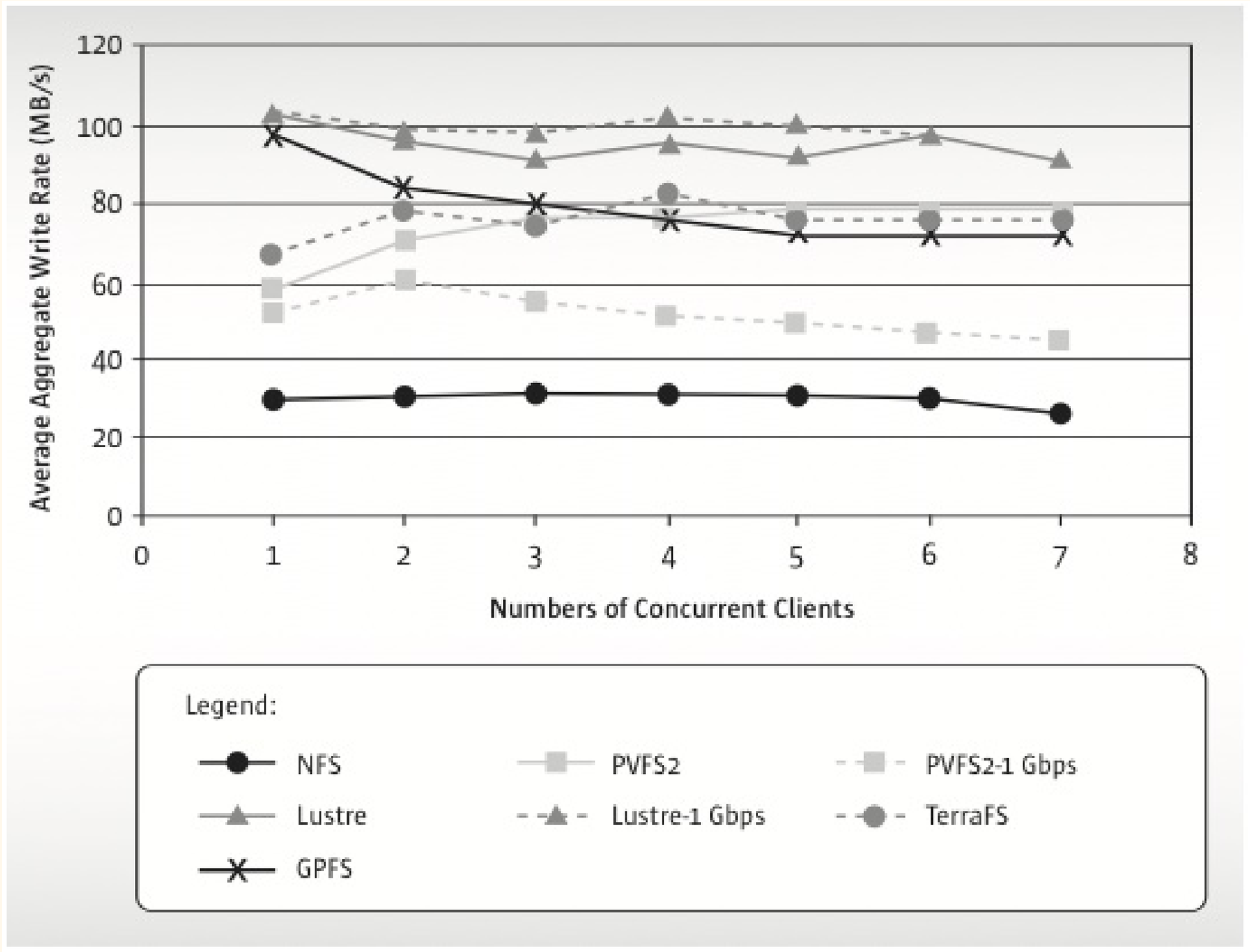
Other network file systems provide block-level access, etc.

Our cluster uses the Lustre distributed network file system:

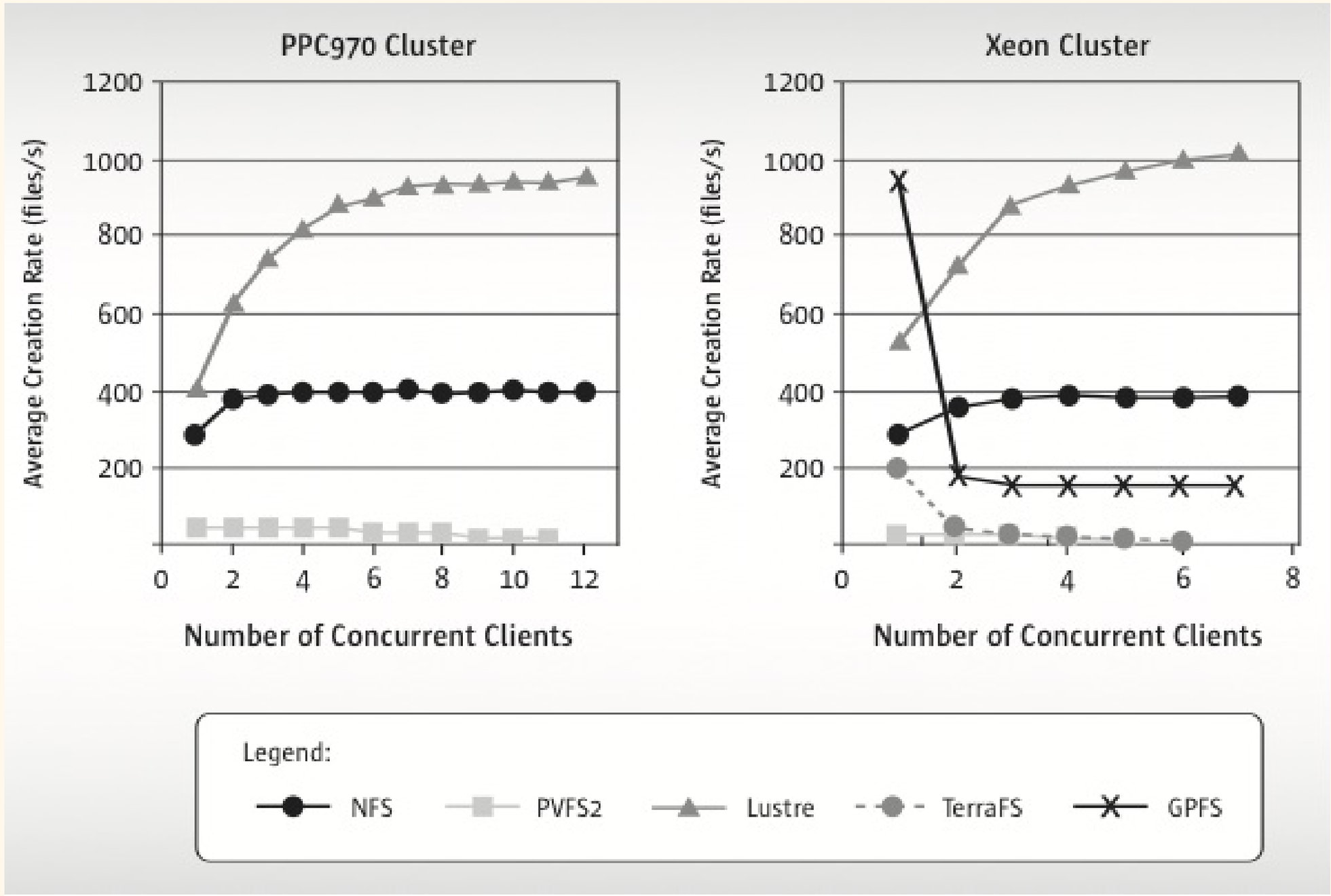



MDS disk storage containing
Metadata Targets (MDT)
Pool of clustered MDS servers
1-100
MDS 1
(active)
MDS 2
(standby)
OSS servers
1-1000s
OSS storage with object
storage targets (OST)
OSS 1
Commodity Storage
Elan
Myrinet
InfiniBand
Lustre clients
1-100,000
Simultaneous
support of multiple
network types
OSS 2
OSS 3
Shared storage
enables failover OSS
OSS 4
OSS 5
OSS 6
OSS 7
Router
GigE
Enterprise-Class
Storage Arrays and
SAN Fabric
= Failover

Lustre holds up well under concurrent load:



Lustre holds up well under concurrent load:



18 TB Lustre system (/l2/users/userid)

- MDS: 2 x 4-core CPUs @ 3GHz, 16 GB

- MDT: 15k rpm, 400GB x 15 (6 TB)

- ODS (5): 2 x 4-core CPUs @ 3GHz, 16 GB
- ODT: 7.2k rpm, 1 TB x 6
- Note: Limited backup

Future upgrades are planned!

We'll be talking more about distributed file systems/stores throughout the course.

Systems such as Lustre are extensions of traditional file systems...

... but for truly large data collections, the file system model can be inadequate.

File systems such as the Google File System (GFS) and the Hadoop File System (HDFS) can offer more scalability and reliability.

GFS was invented at Google to store their web search index:

- Fault-tolerance
- Implemented at user-level, provides location-awareness
- Assumptions: high sustained bandwidth > low latency
  - Large files are typical
  - Large streaming reads and small random reads
  - Large sequential writes and small random writes
- No file or directory aliases (hard or soft links)
- Clients can concurrently append to a file efficieintly

Google File System

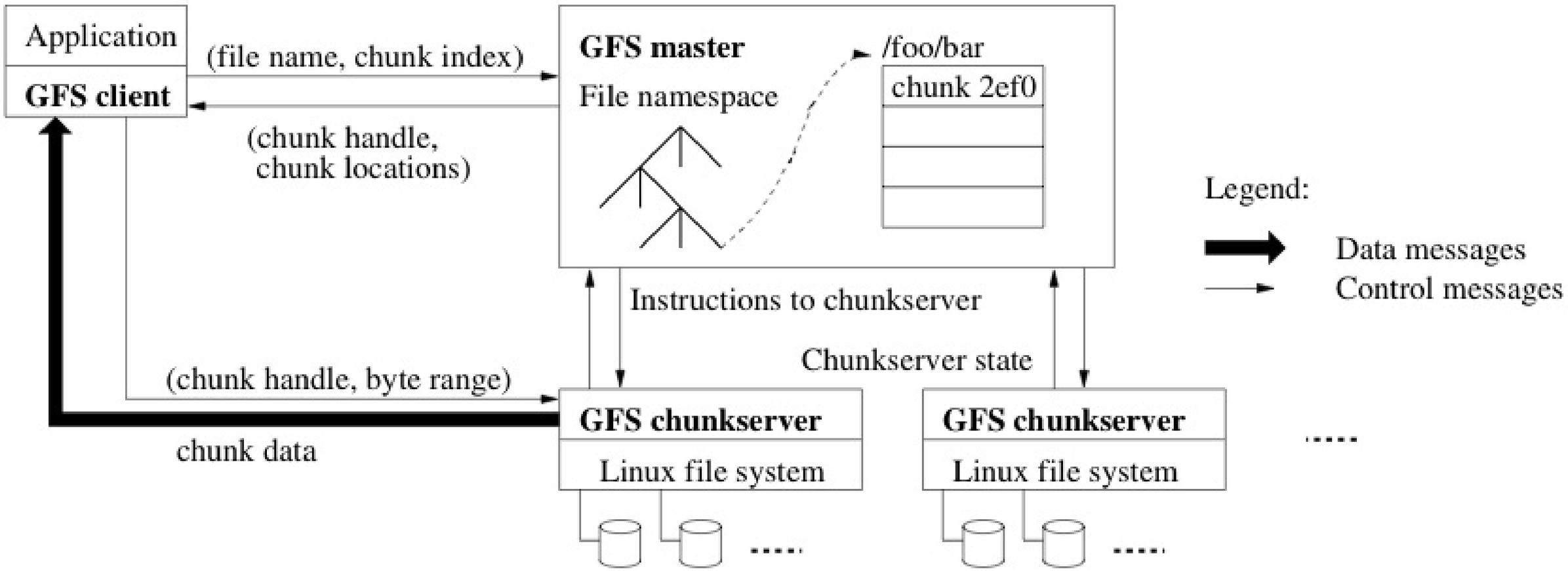


Figure 1: GFS Architecture

- Single master, multiple chunkservers
- Files are divided up into chunk, ID-ed by an addres
- Chunkservers manage chunks like local files
- Chunk data replicated for reliability

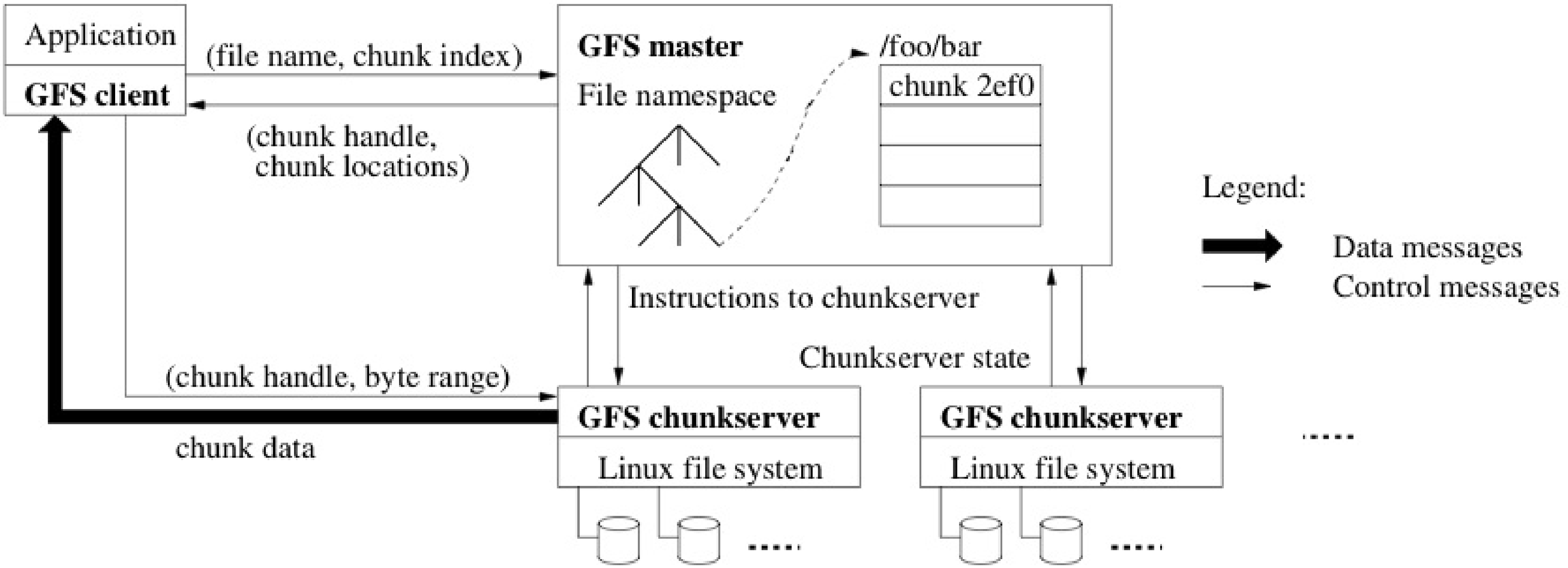


Figure 1: GFS Architecture

HDFS is essentially an open-source implementation of GFS.

We'll be talking more about distributed file systems/stores throughout the course!

Execute instruction	1 ns
Fetch from L1 cache	0.5 ns
Branch misprediction	5 ns
Fetch from L2 cache	7 ns
Mutex lock/unlock	25 ns
Fetch from main memory	100 ns
Send 2kb over 1Gbps network	20,000 ns
Read 1mb sequentially from memory	250,000 ns
Fetch from new disk location (seek)	8,000,000 ns (20 ms)
Read 1mb sequentially from disk	20,000,000 ns (20 ms)
Roundtrip packet from US to Europe	150,000,000 ns (150 ms)

http://norvig.com/21-days.html

Game plan for today:

Structure of the course

Overview of parallel and distributed computing

Quick intro to distributed file systems

Do you actually need a cluster?

Do you actually need a cluster?

Single computers these days have a *lot* of memory...

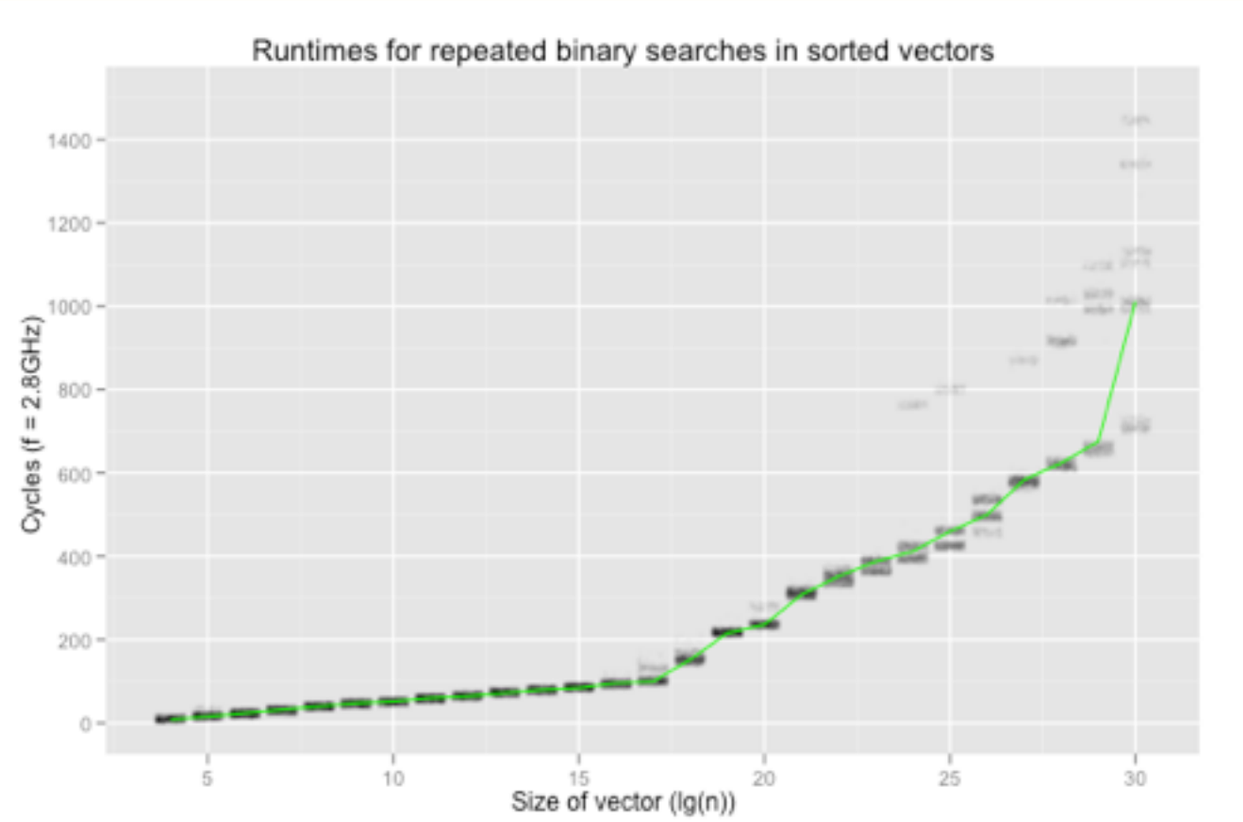
... and also multiple CPU cores.

Do you actually *need* to fit it all into memory at once?

Is your problem I/O-bound, or CPU-bound?

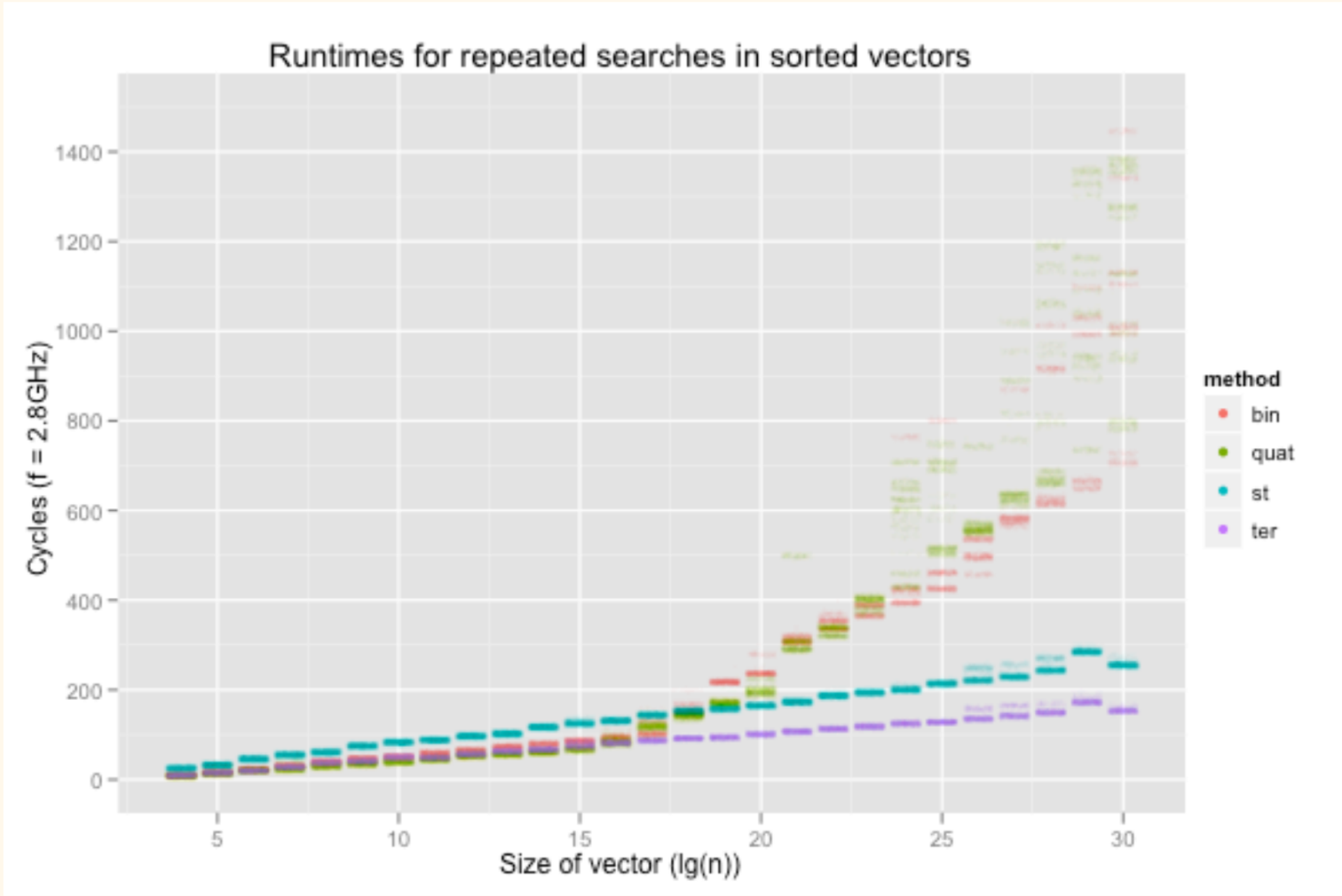
If it's CPU-bound, have you used a profiler?

Don’t forget about caching...



http://www.pvk.ca/Blog/2012/07/30/binary-search-is-a-pathological-case-for-caches/

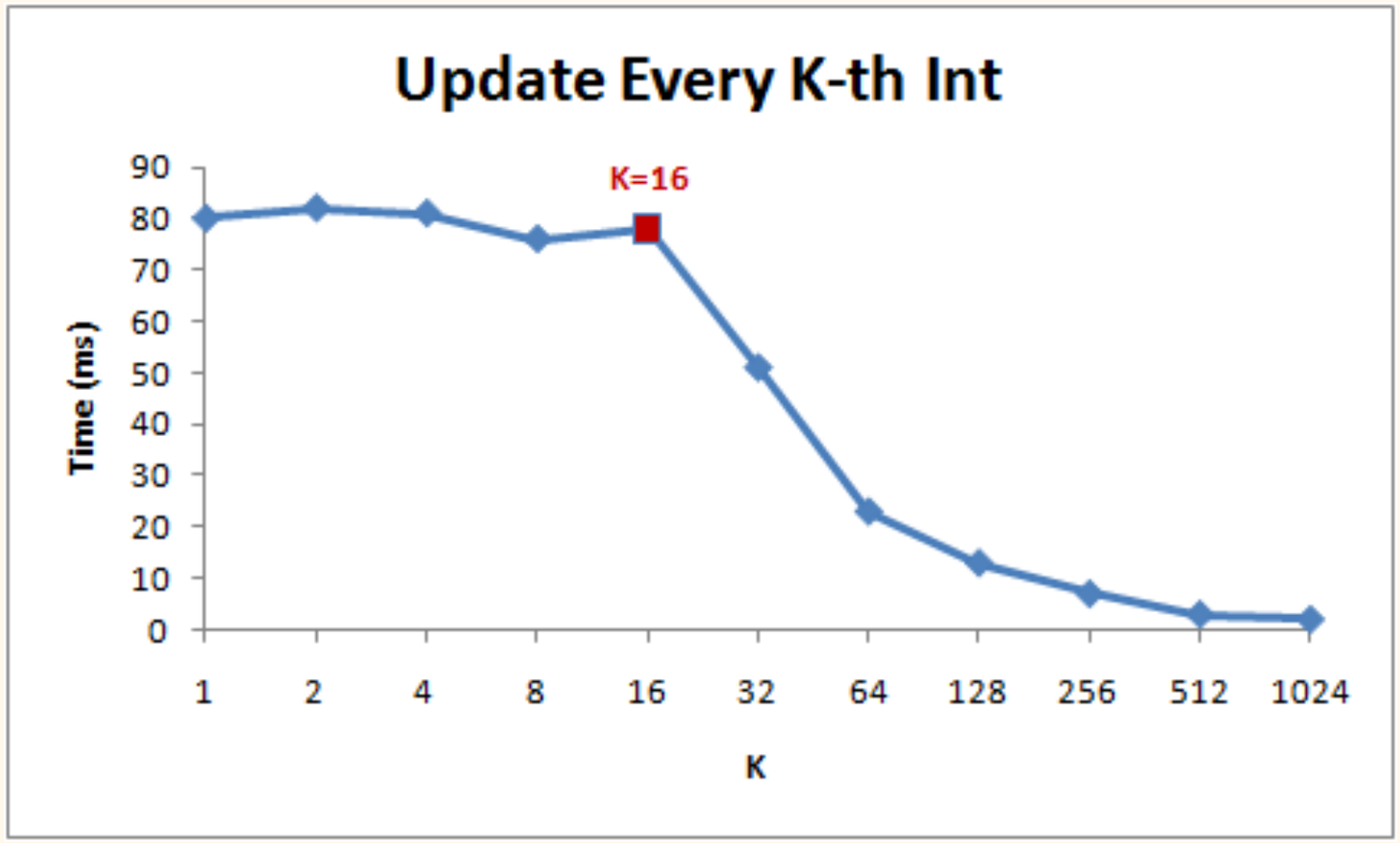
Don’t forget about caching...



http://www.pvk.ca/Blog/2012/07/30/binary-search-is-a-pathological-case-for-caches/

Don’t forget about caching...

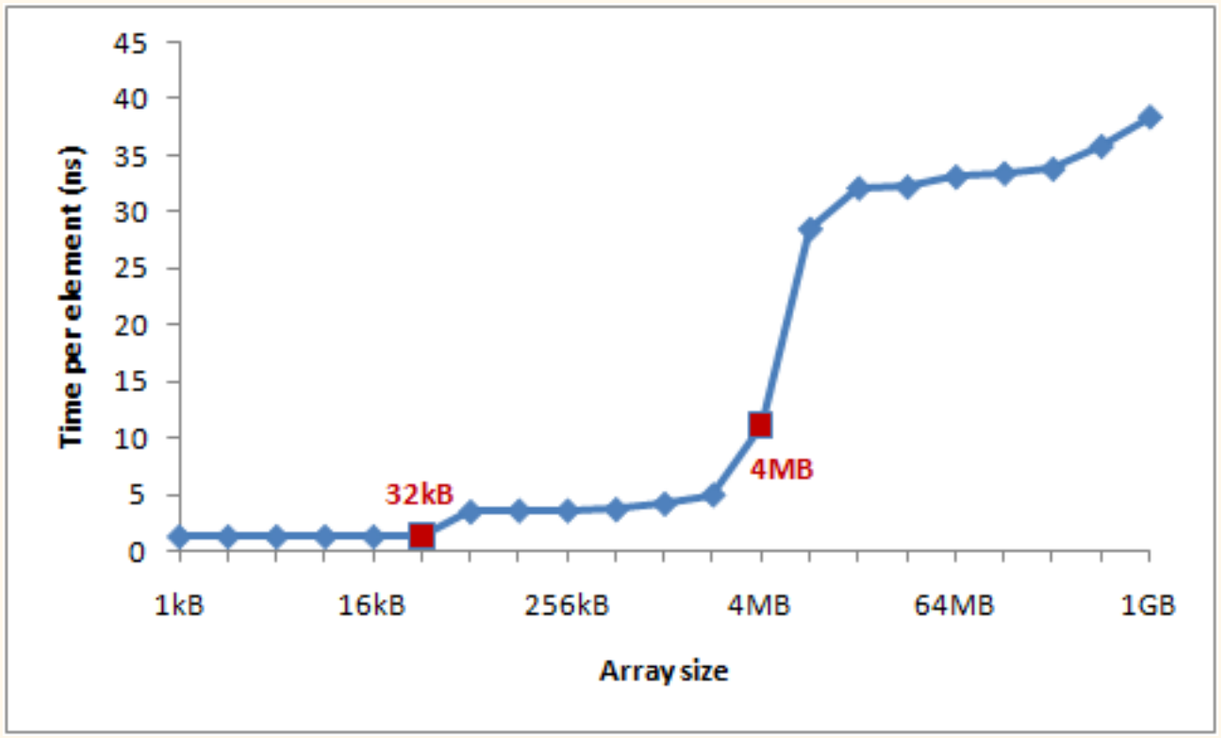
```
for (int i = 0; i < arr.Length; i += K) arr[i] *= 3;
```



http://igoro.com/archive/gallery-of-processor-cache-effects/

Don’t forget about caching...

```
int steps = 64 * 1024 * 1024; // Arbitrary number of steps
int lengthMod = arr.Length - 1;
for (int i = 0; i < steps; i++)
{
    arr[(i * 16) & lengthMod]++; // (x & lengthMod) is equal to (x % arr.Length)
}
```



http://igoro.com/archive/gallery-of-processor-cache-effects/

... and branch prediction!

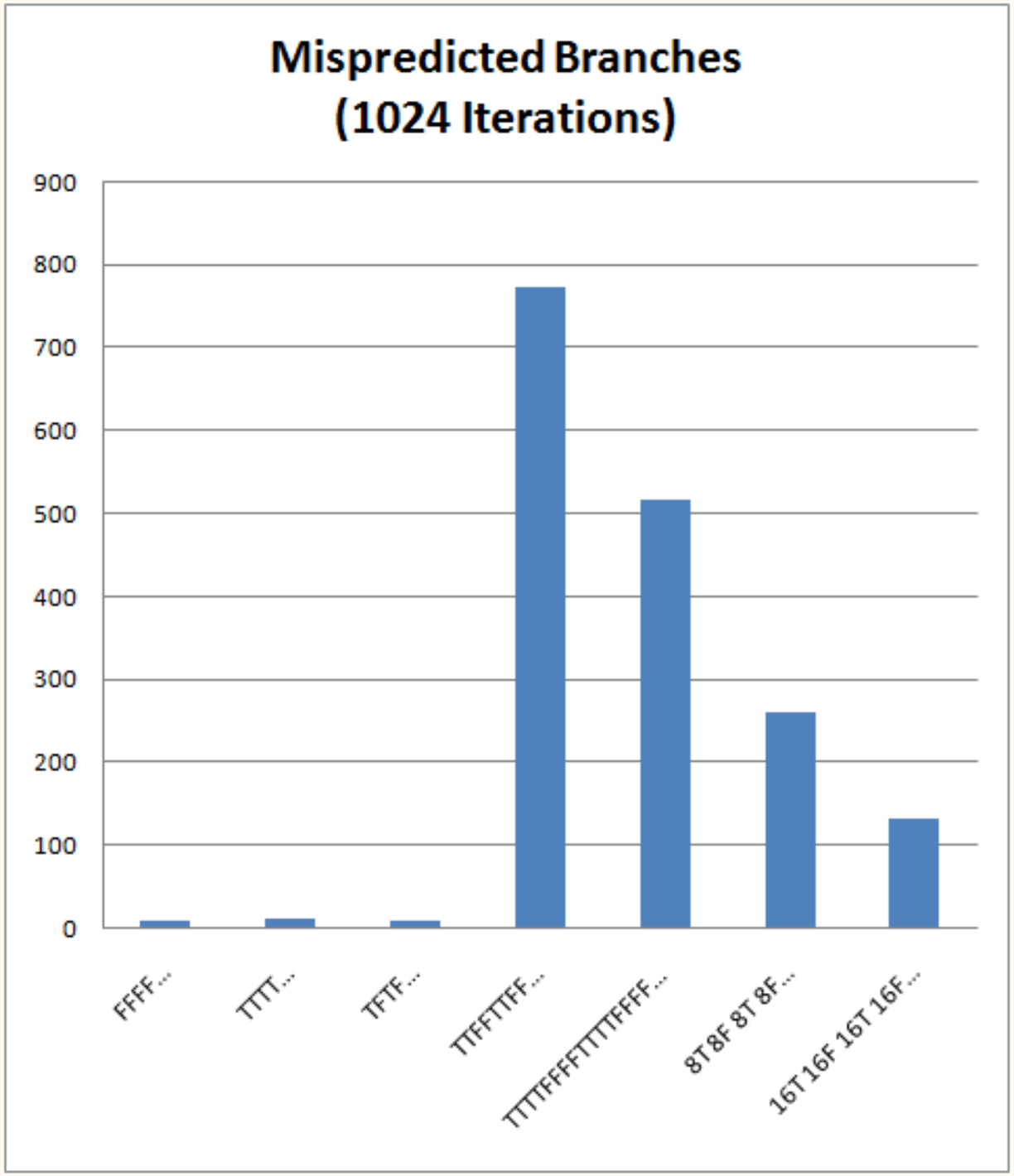
```
for (int i = 0; i < max; i++) if (<condition>) sum++;
```

Condition	Pattern	Time (ms)
(i & 0x80000000) == 0	T repeated	322
(i & 0xffffffff) == 0	F repeated	276
(i & 1) == 0	TF alternating	760
(i & 3) == 0	TFFFTFFF...	513
(i & 2) == 0	TTFFTTFF...	1675
(i & 4) == 0	TTTTFFFFTTTTFFFF...	1275
(i & 8) == 0	8T 8F 8T 8F ...	752
(i & 16) == 0	16T 16F 16T 16F ...	490

http://igoro.com/archive/fast-and-slow-if-statements-branch-prediction-in-modern-processors/

... and branch prediction!

```
for (int i = 0; i < max; i++) if (<condition>) sum++;
```



http://igoro.com/archive/fast-and-slow-if-statements-branch-prediction-in-modern-processors/

Do you actually need a cluster?

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Execute instruction	1 ns
Fetch from L1 cache	0.5 ns
Branch misprediction	5 ns
Fetch from L2 cache	7 ns
Mutex lock/unlock	25 ns
Fetch from main memory	100 ns
Send 2kb over 1Gbps network	20,000 ns
Read 1mb sequentially from memory	250,000 ns
Fetch from new disk location (seek)	8,000,000 ns (20 ms)
Read 1mb sequentially from disk	20,000,000 ns (20 ms)
Roundtrip packet from US to Europe	150,000,000 ns (150 ms)

http://norvig.com/21-days.html