

RAID Storage, Network File Systems, and DropBox

George Porter

CSE 124

February 24, 2015

* Thanks to Dave Patterson and Hong Jiang

Announcements

- Project 2 due by end of today
 - Office hour today 2-3pm in B275
- Project 3 out

Overview

- Networked file storage is really important
- Used in companies/business/education
- Used in cloud computing environments
- Used by people our daily lives
- Challenges:
 - How to access storage over the network?
 - How to keep it reliable?

We'll start with the single-node case first



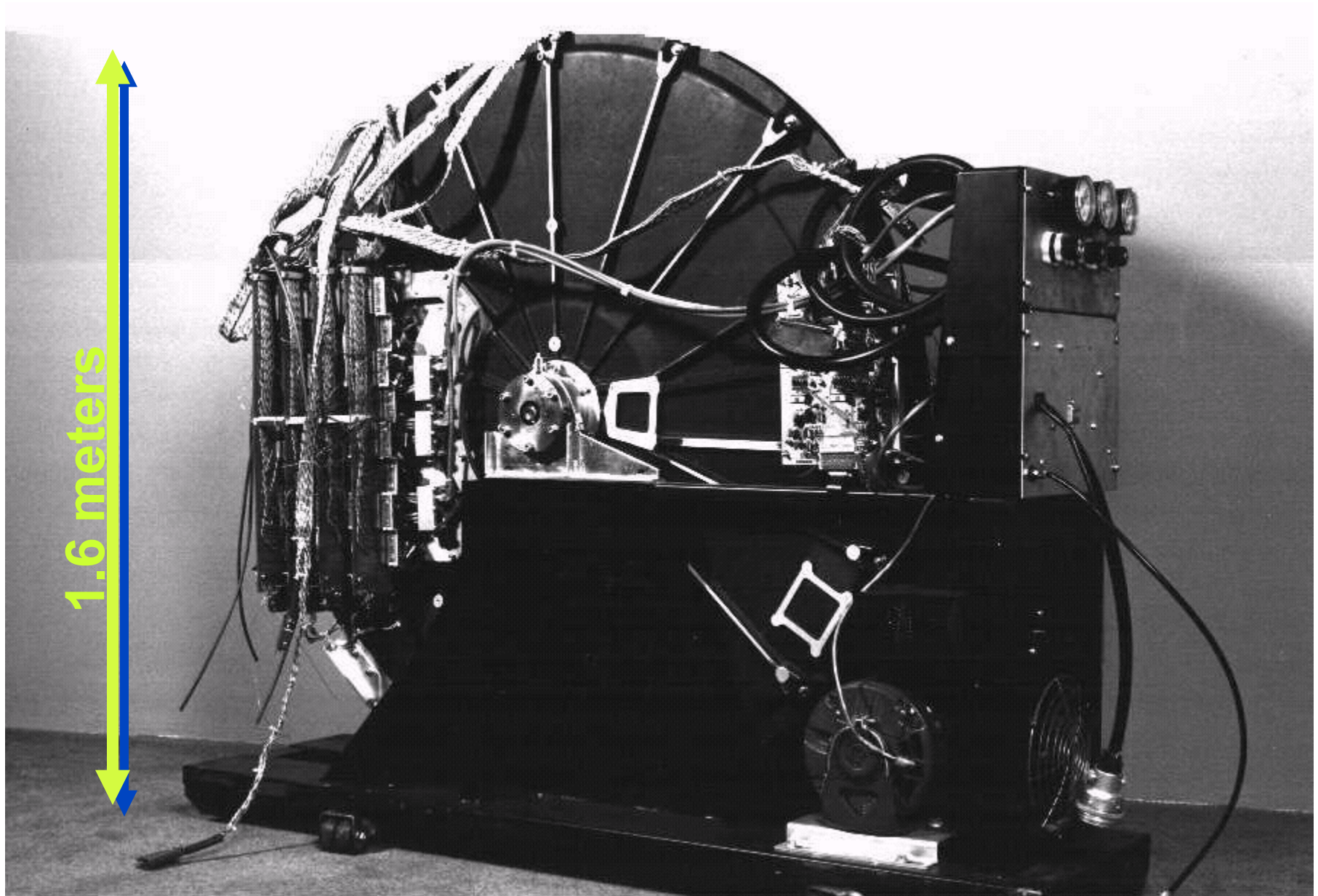
Dropbox

The first HDD (1956)

- IBM 305 RAMAC
- 4 MB
- 50x24" disks
- 1200 rpm
- 100 ms access
- 35k\$/y rent
- Included computer & accounting software
(tubes not transistors)



10 years later

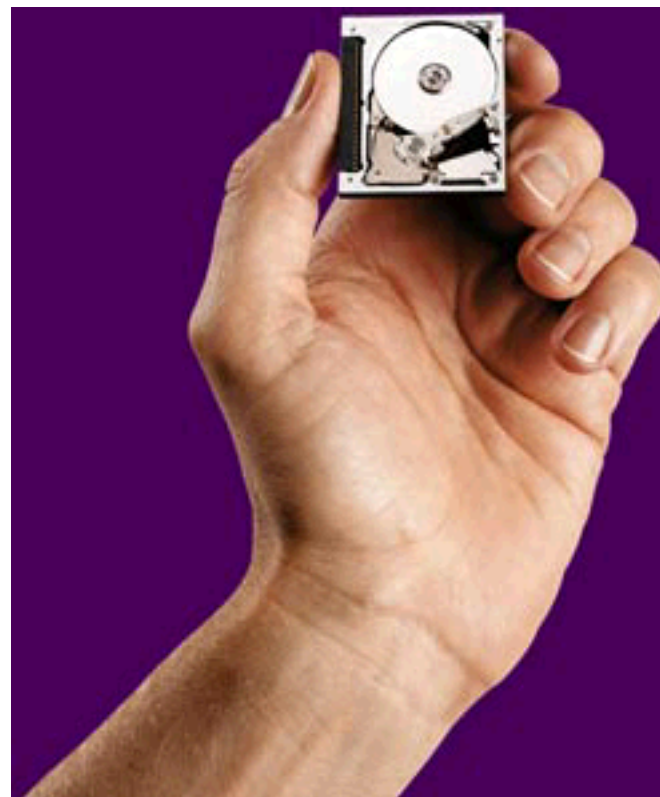


Transportation of HDD

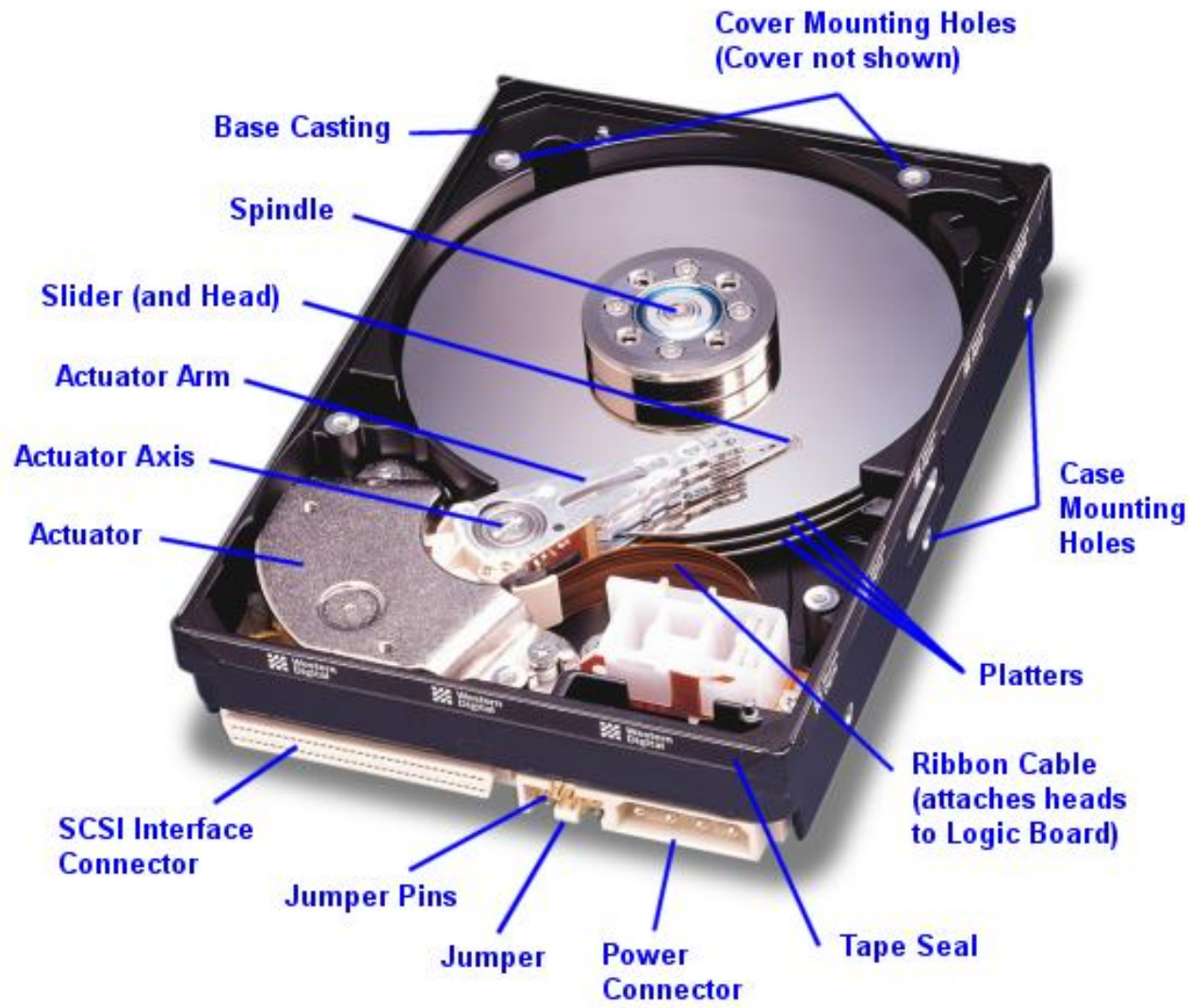


1 inch disk drive!

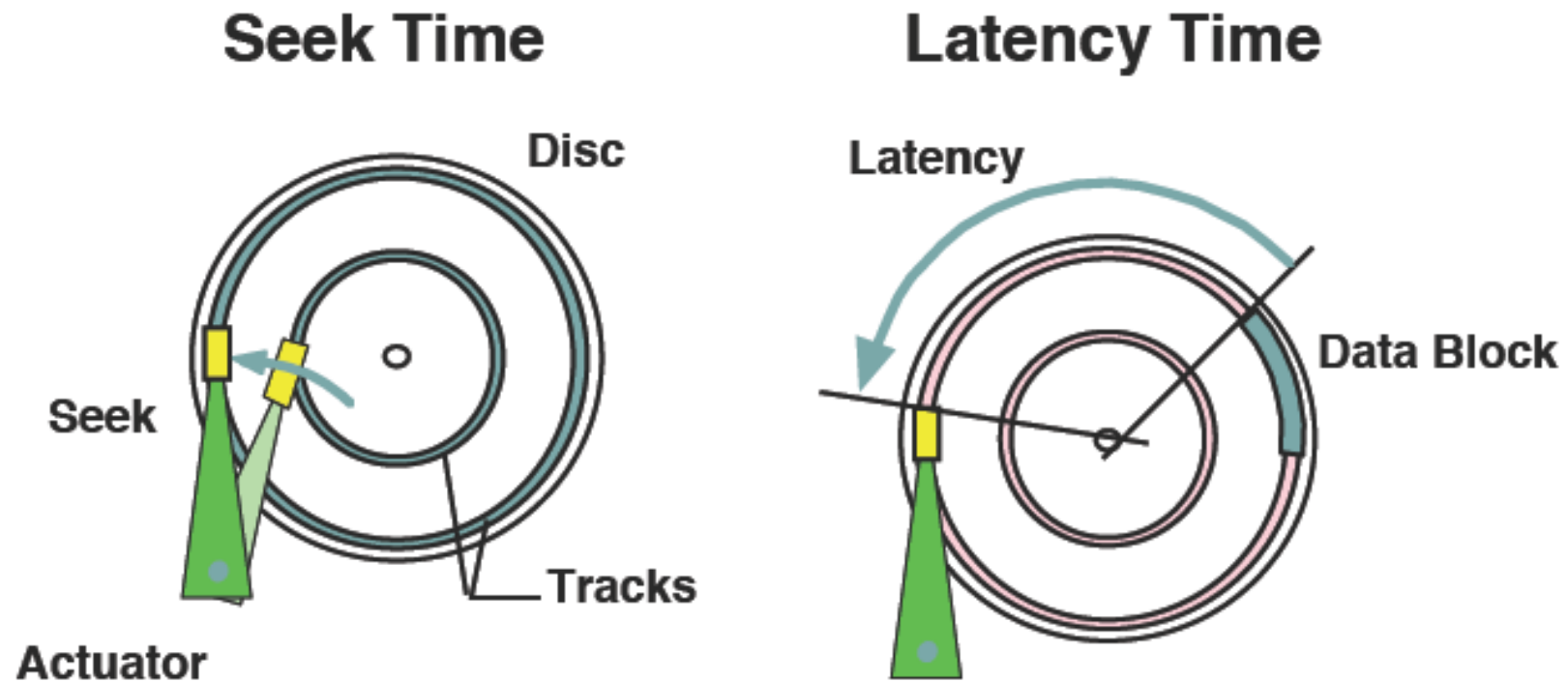
- 2000 IBM MicroDrive:
 - 1.7" x 1.4" x 0.2"
 - 1 GB, 3600 RPM, 5 MB/s, 15 ms seek
 - Digital camera, PalmPC?
- 2006 MicroDrive
 - 8 GB, 50 MB/s!



The internal look of HDD (now)



Data access of HDD



$\text{Access Time} = \text{Seek Time} + \text{Rotational Delay} + \text{Transfer Time}$

Redundant Array of Inexpensive Disks (RAID): 1987-1993



The RAID paper

- D. A. Patterson, G. Gibson, and R. H. Katz, "*A case for redundant arrays of inexpensive disks (RAID)*," in SIGMOD'88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, 1988, vol. 17, no. 3, pp. 109-116.

[PDF] [A Case for Redundant Arrays of Inexpensive Disks -](http://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf)

www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf

File Format: PDF/Adobe Acrobat - Quick View

by DA Patterson **Cited by 2804** Related articles

- One of the important publications in computer science.
[http://en.wikipedia.org/wiki/
List_of_important_publications_in_computer_science](http://en.wikipedia.org/wiki/List_of_important_publications_in_computer_science)
- EMC, HP, IBM, NetApp... have produced so many RAID-related storage products.

Better Storage?

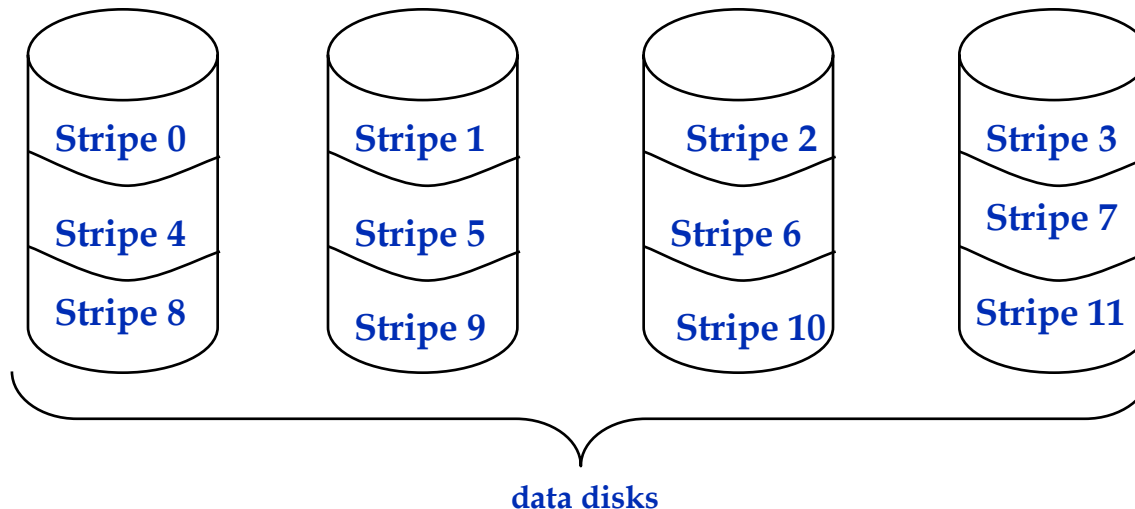
- Capacity?
- Performance?
- Availability?
-

RAID introduction

- A RAID is a Redundant Array of Inexpensive Disks.
 - In industry, “I” is for “Independent”
 - The alternative is SLED, single large expensive disk
- Disks are small and cheap, so it’s easy to put lots of disks (10s to 100s) in one box for increased storage, performance, and availability.
- The RAID box with a RAID controller looks just like a SLED to the computer. Data plus some redundant information is Striped across the disks in some way.
- How that Striping is done is key to performance and reliability----Different RAID levels 0-5, 6...

RAID0

- Level 0 is non-redundant disk array
- Files are Striped across disks, no redundant info
- High read throughput
- Best write throughput (no redundant info to write)
- Any disk failure results in data loss
- Reliability worse than SLED



Array Reliability

- **Reliability of N disks = Reliability of 1 Disk $\div$ N**

50,000 Hours $\div$ 70 disks = 700 hours

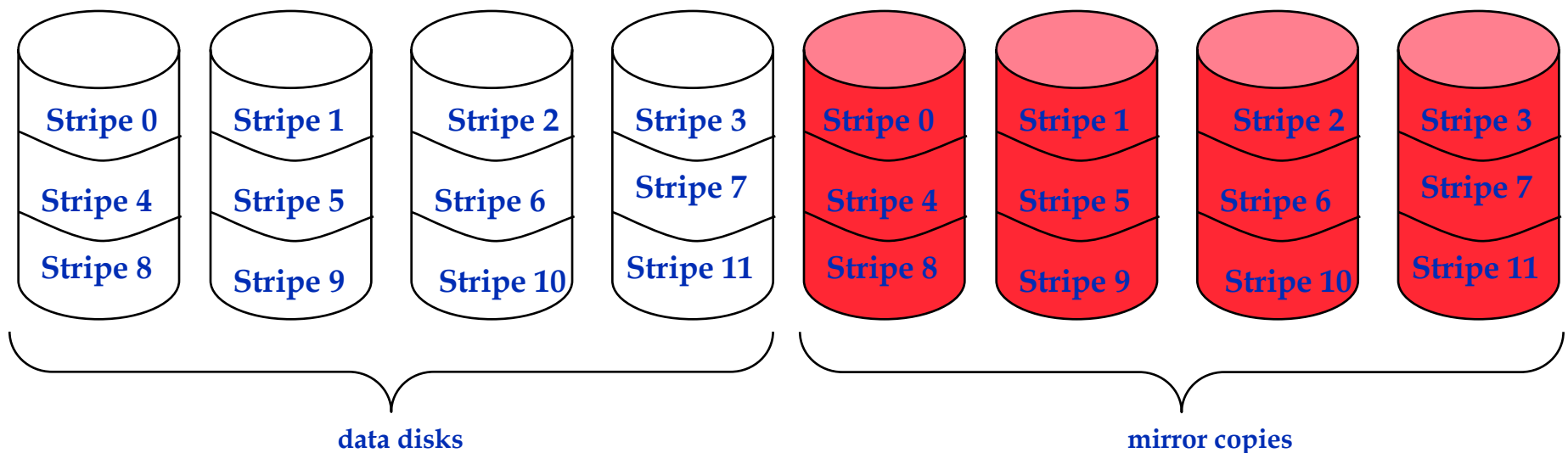
Disk system MTTF: Drops from 6 years to 1 month!

- **Arrays (without redundancy) too unreliable to be useful!**

Hot spares support reconstruction in parallel with access: very high media availability can be achieved

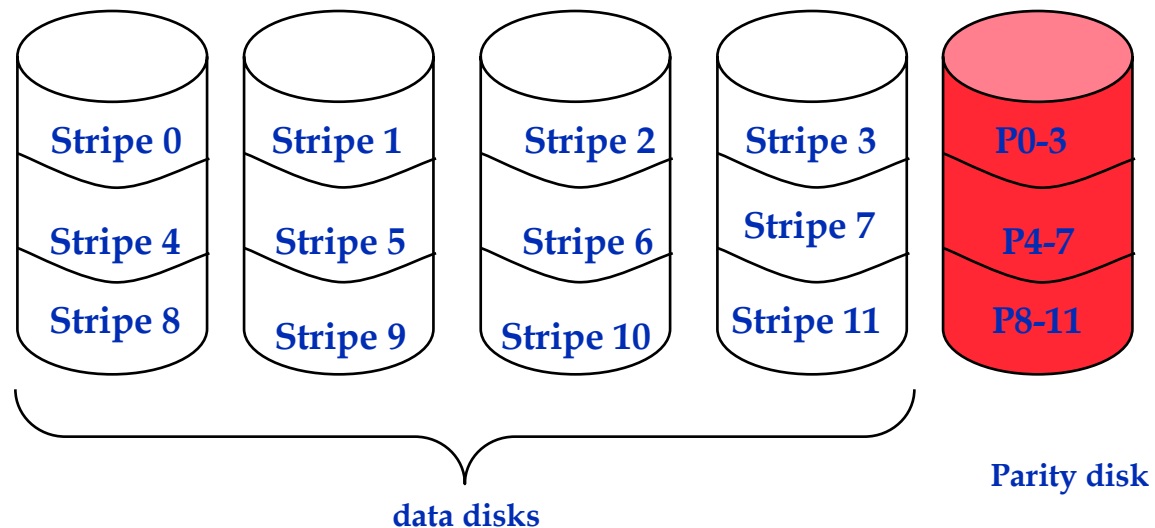
RAID1

- Mirrored Disks, data is written to two places
- On failure, just use surviving disk
- On read, choose fastest to read
- Write performance is same as single drive, read performance is 2x better
- Expensive



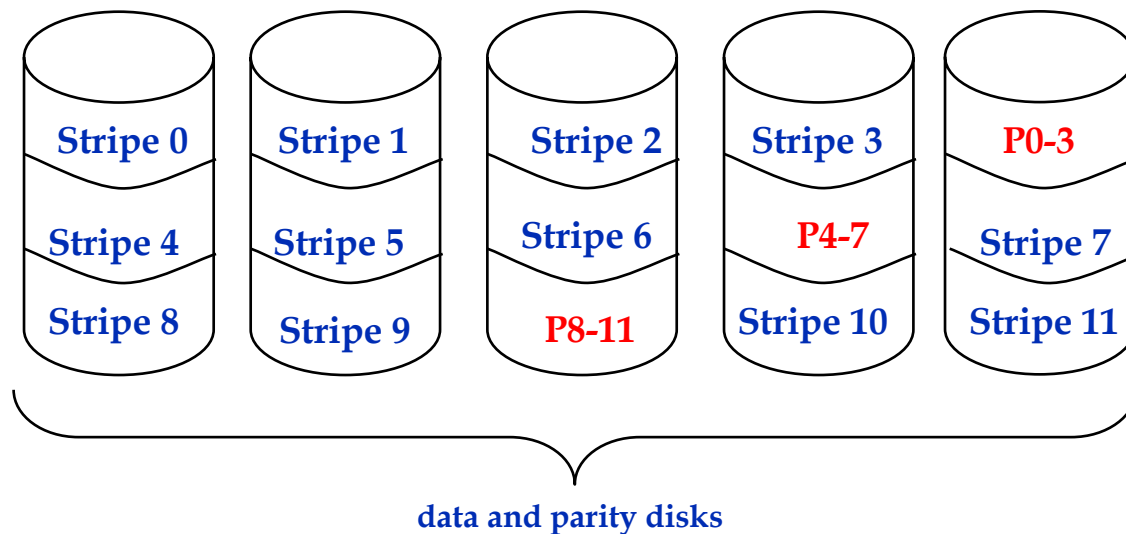
RAID4

- Block-level parity with Stripes
- A read accesses the appropriate data disk
- A write accesses all data disks plus the parity disk
 - Why?
- Heavy load on the parity disk



RAID5

- Block Interleaved Distributed Parity
- Like parity scheme, but distribute the parity info over all disks (as well as data over all disks)
- Better read performance, large write performance
- What happens when a single disk fails?

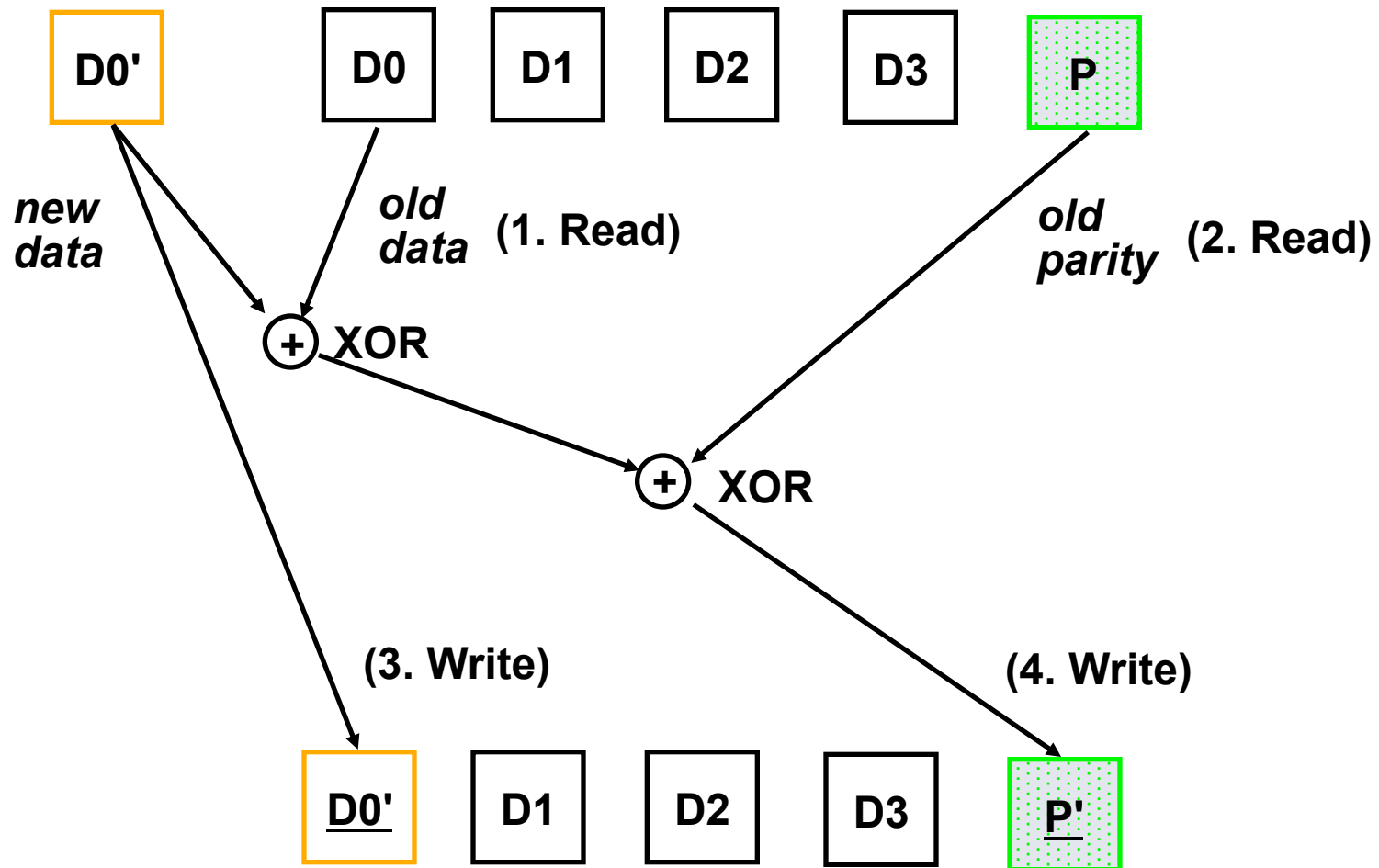


Problems of Disk Arrays:

Small Writes

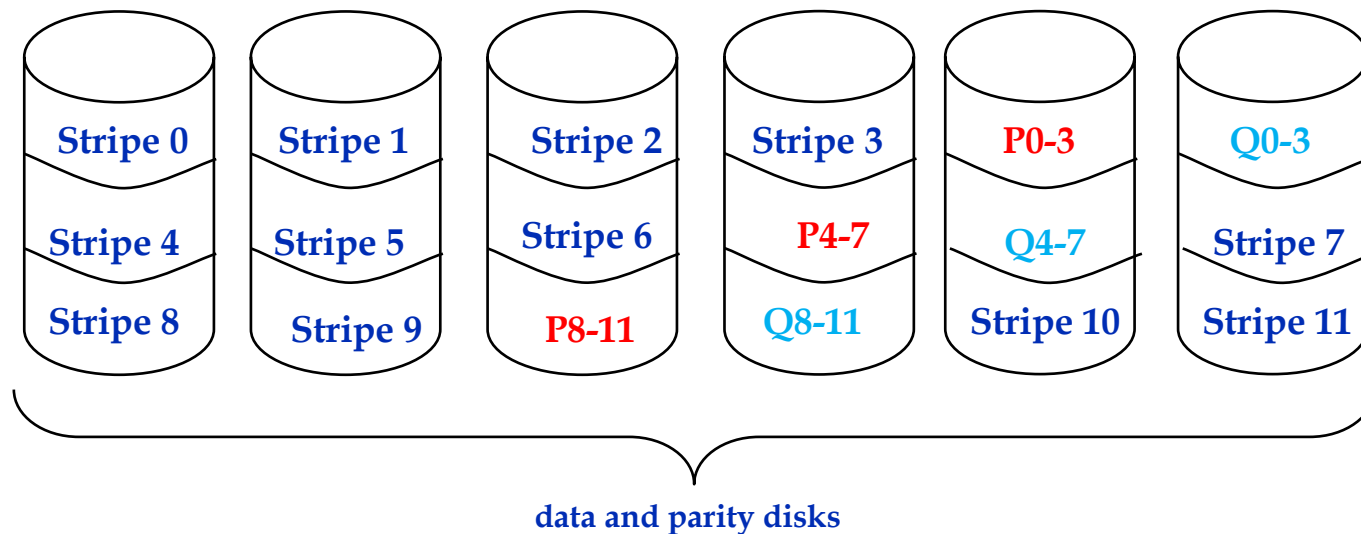
RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



RAID6

- Level 5 with an extra parity
- Can tolerate two failures
- What are the odds of having two concurrent failures?
- May outperform Level-5 on reads, slower on writes



Comparison of RAIDs

RAID Levels	Capacity	Storage Efficiency	Availability	Ran. Read	Ran. Write	Seq. Read	Seq. Write
0	$S * N$	100%	*	*****	*****	*****	*****
1	$S * N/2$	50%	*****	***	***	**	**
4	$S * (N-1)$	$(N-1) / N$	***	*****	**	*****	**
5	$S * (N-1)$	$(N-1) / N$	***	*****	**	*****	***
6	$S * (N-2)$	$(N-2) / N$	*****	*****	*	*****	**

Note: **S** indicates the capacity of a single disk, **N** indicates the number of the disks in a RAID set.

Distributed File Systems

Distributed File Systems

- Goal: transparent access to remote files
 - Access remote files as if they were stored on local hard drive
 - Why would you want this?
 - What are some of the hard issues?
- Examples:
 - NFS: Sun's Network File System
 - AFS: Andrew File System
 - Coda: CMU research project for mobile clients (now available in Linux)
 - xFS: Berkeley research project stressing “serverless” design

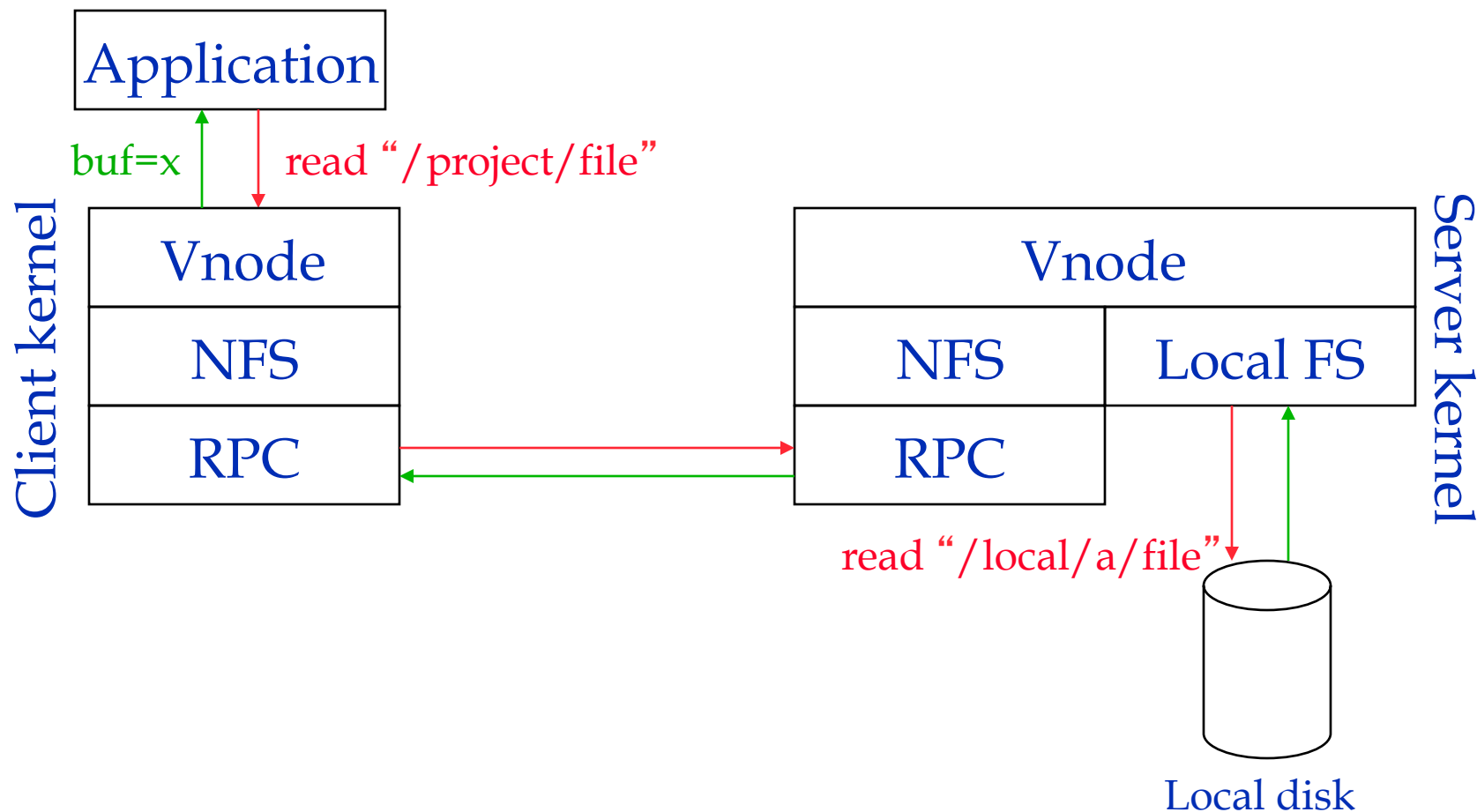
Distributed File Systems: Motivation

- Centralized administration
 - E.g., upgrades, backups, additional storage
- Same file system independent of physical machine
 - Important distributed system mantra: *location independence*
- Incremental scalability
 - Do not give everyone 20 GB disk if average user needs 1 GB
 - Add disks to central server rather than desktops

Distributed File System Issues

- Semantic transparency and performance transparency:
 - Naming: Do not change file names in moving from machine to machine
 - Caching: approximate local performance
 - Availability: remote server crash (fate sharing)
 - Security: protect sensitive information
 - Scale: how large can system grow?
 - In terms of storage and user base

Simplified Access Model Example



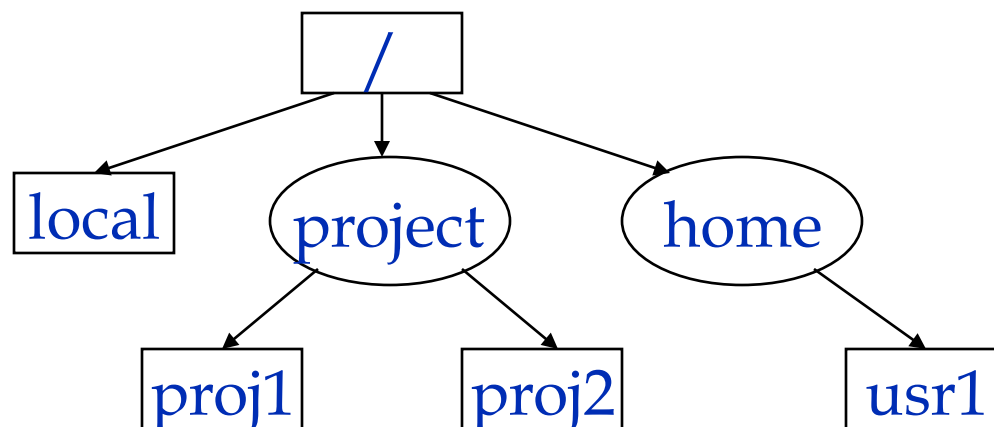
Performance

- How to make distributed file access approximate the performance of local file access?

Performance

- Network latency and limited bandwidth make it difficult to match local performance
 - But network bandwidth is surpassing disk bandwidth
 - Storage area networks, iSCSI
- How to make distributed file access approximate the performance of local file access?
 - Caching: take advantage of locality
 - Both spatial and temporal*
 - What issues are introduced by caching?

Distributed File System Structure



- Perform *mount* operation to attach remote file system into local namespace
 - E.g., /project/proj1 actually a file on remote machine (maps to server.cs.ucsd.edu:/local/a/project/proj1)

UNIX File Usage

- Most files are small ($< 10k$)
- Reads outnumber writes ($\sim 6:1$)
- Sequential access is common
- Files remain open for short period of time
 - $75\% < .5s$, $90\% < 10s$
- Most files accessed by exactly one user
 - Most shared files written by exactly one user
- Temporal locality: recently accessed files likely to be accessed again in near future
- Most bytes/files are short lived

Building a Distributed File System

- Debate in late 1980's, early 1990's:
 - Stateless vs. stateful file server
- NFS: stateless server
 - Only store contents of files + soft state (for performance)
 - Crash recovery simple operation
 - All RPCs idempotent (no state)
 - At least once RPC semantics sufficient
 - Server unaware of users accessing files
- Clients have to check with server periodically for the uncommon case
 - Where directory/file has been modified

Server Caching

- Cache read results, writes, directory operations
- Write-through vs. write-back cache?
- Pros/cons?

NFS Server Caching

- Cache read results, writes, directory operations
- Write-through cache vs. write-back cache?
 - Write through: Each update written to disk immediately
 - When write operation returns, client is guaranteed stable update
- Pros:
 - Stateless (easy to implement), no data lost on crash
- Cons:
 - Slow: client must wait for disk write

NFS Client Caching

- Clients cache read, writes, and directory ops
 - What if multiple people updating the same file at the same time? Consistency problems
- NFS approach:
 - Server maintains last modification time/per file
 - Client remembers time it initially retrieved data
 - On file access, client checks timestamp against server (every 3-30 seconds)

Lots of unnecessary timestamp checking

How long to set the timeout? What is the tradeoff?

NFS Replication

- As originally specified, NFS did not support data replication
- More recent versions of NFS support replication via a mechanism called Automounter
 - Allows remote mount points to be specified using a set of servers
 - However, manually propagate modifications to replicas
- Intended primarily for READ-ONLY files

NFS Security

- NFS uses underlying Unix file protection on servers for access checks
- In early NFS, mutual trust assumed among all participating machines
 - User identity determined by client machine and accepted without further server validation
- More recent versions of NFS use DES-based mutual authentication to provide a higher level of security
 - File data in RPC packets is not encrypted → NFS is still vulnerable