



COS 318: Operating Systems

Storage and File System

Vivek Pai

Computer Science Department

Princeton University

<http://www.cs.princeton.edu/courses/archive/fall11/cos318/>

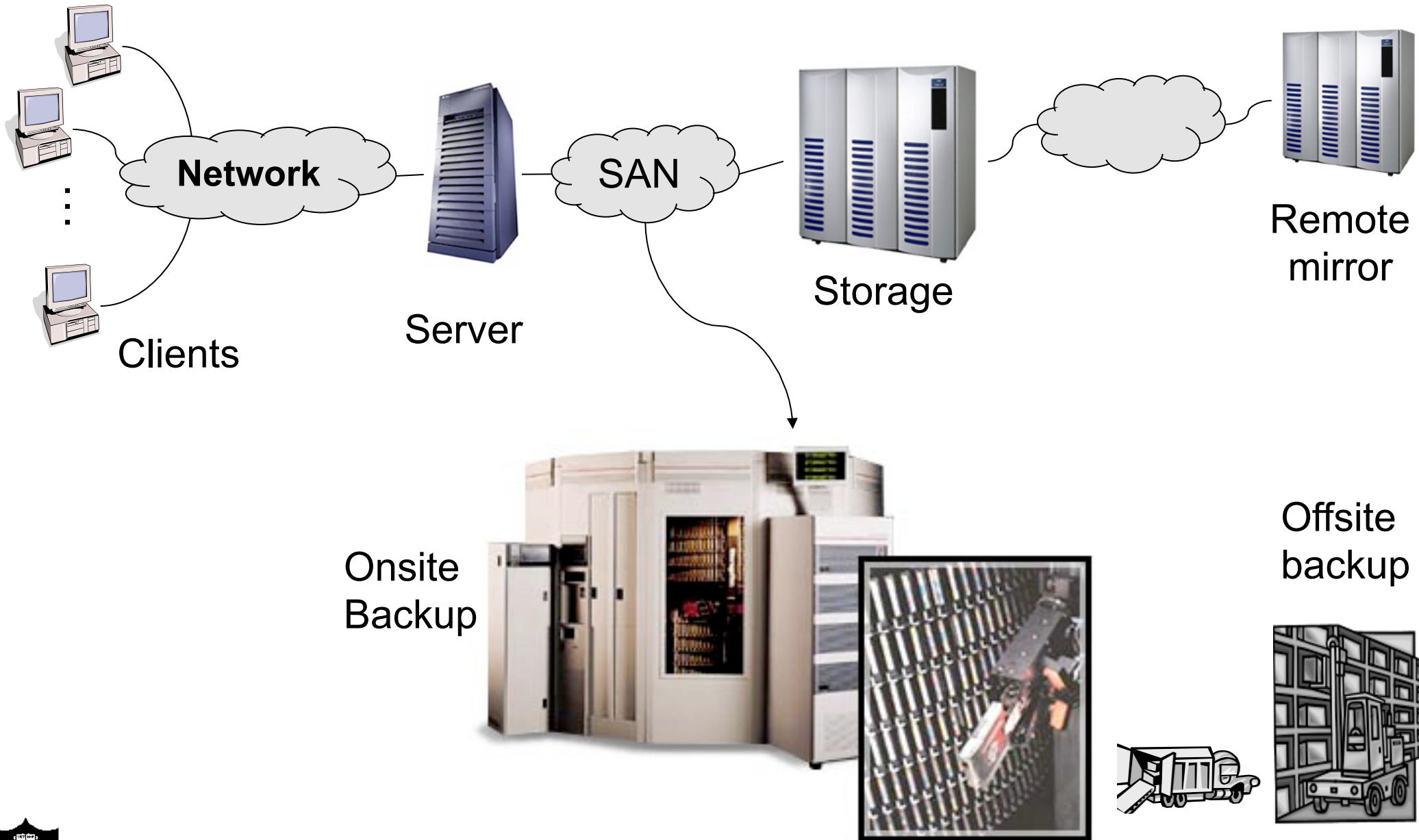


Topics

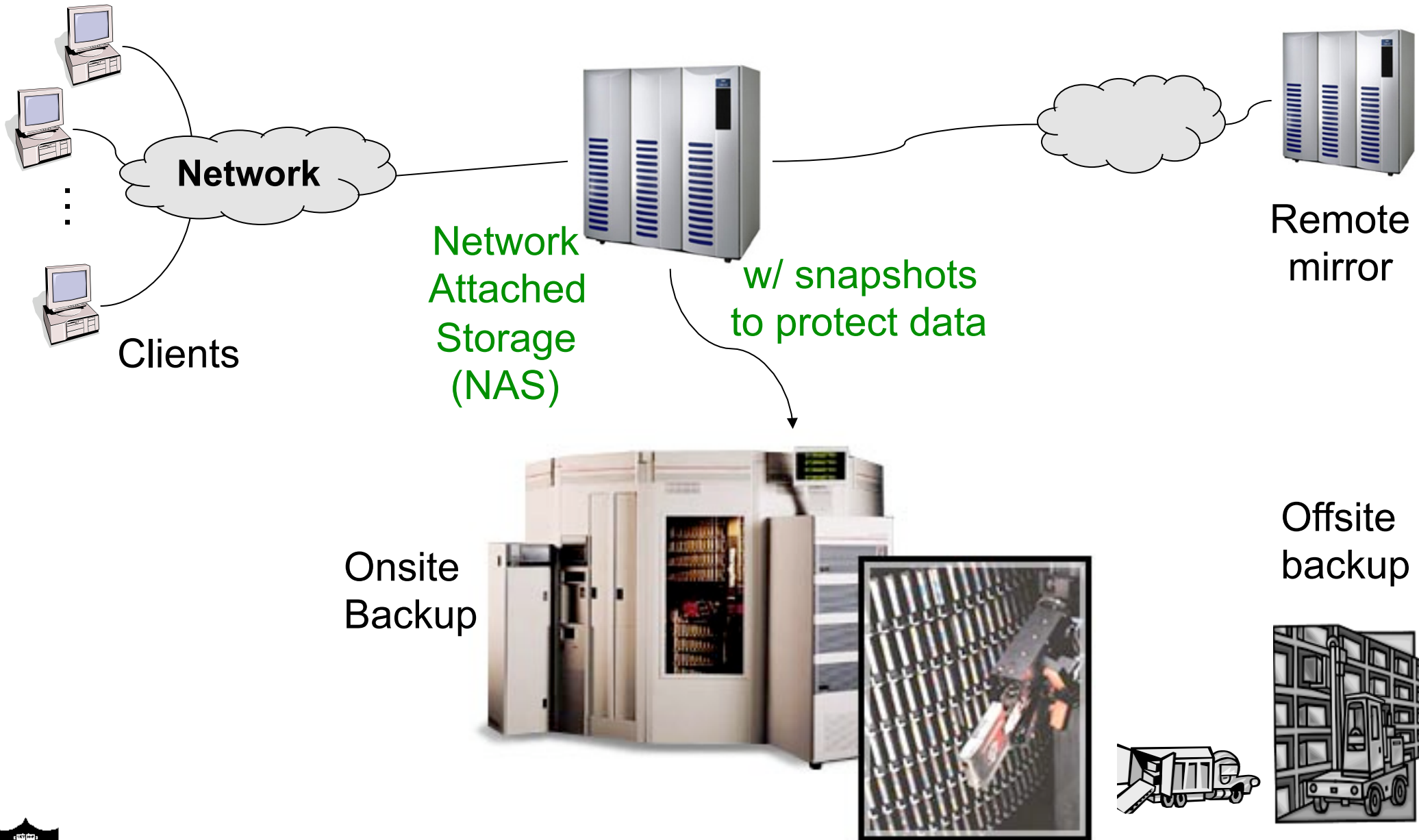
- ◆ Storage hierarchy
- ◆ File system abstraction
- ◆ File system operations
- ◆ File system protection



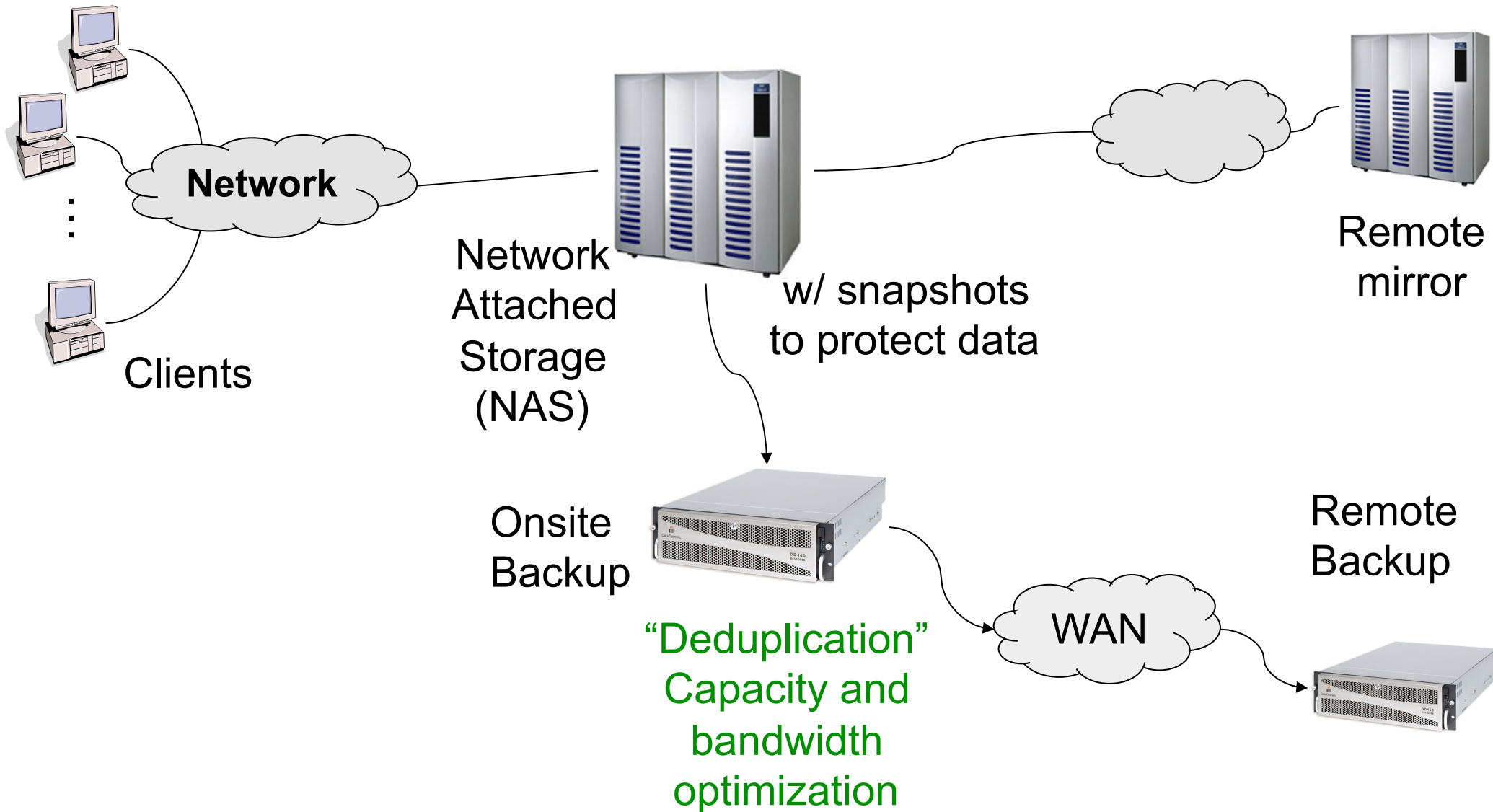
Traditional Data Center Storage Hierarchy



Evolved Data Center Storage Hierarchy

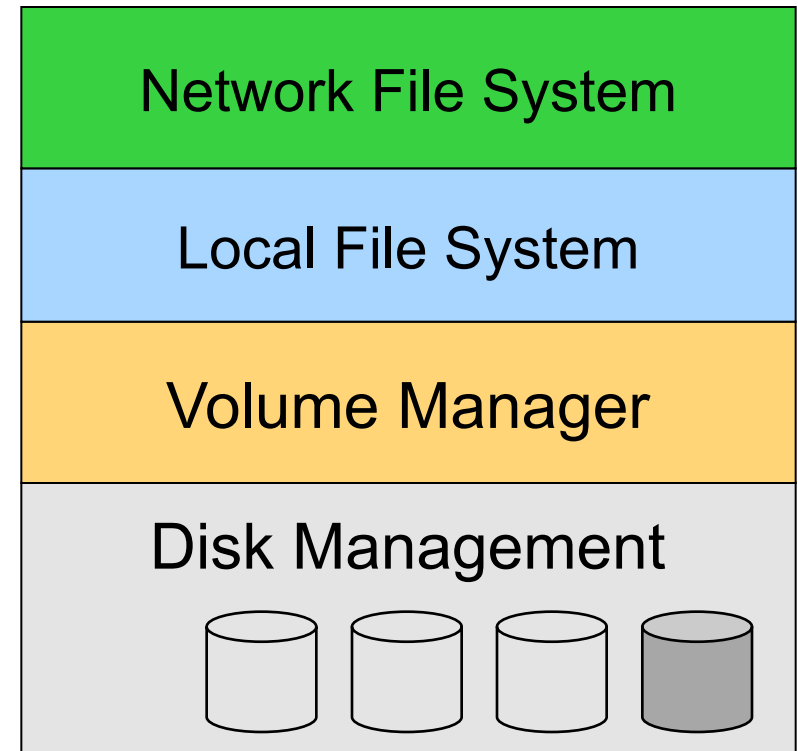


Modern Data Center Storage Hierarchy



File System Layers and Abstractions

- ◆ Network file system maps a network file system protocol to local file systems
 - NFS, CIFS, DAFS, etc
- ◆ Local file system implements a file system on blocks in volumes
 - Local disks or network of disks
- ◆ Volume manager maps logical volume to physical disks
 - Provide logical unit
 - RAID and reconstruction
- ◆ Disk management manages physical disks
 - Sometimes part of volume manager
 - Drivers, scheduling, etc



Volume Manager

◆ What and why?

- Group multiple disk partitions into a logical disk volume
 - No need to deal with physical disk, sector numbers
 - To read a block: `read(vol#, block#, buf, n);`
- Volume can include RAID, tolerating disk failures
 - No need to know about parity disk in RAID-5, for example
 - No need to know about reconstruction
- Volume can provide error detections at disk block level
 - Some products use a checksum block for 8 blocks of data
- Volume can grow or shrink without affecting existing data
- Volume can have remote volumes for disaster recovery
- Remote mirrors can be split or merged for backups

◆ How to implement?

- OS kernel: Veritas (for SUN and NT), Linux
- Disk subsystem: EMC, Hitachi, IBM



Block Storage vs. Files

Disk abstraction

- ◆ Block oriented
- ◆ Block numbers
- ◆ No protection among users of the system
- ◆ Data might be corrupted if machine crashes

File abstraction

- ◆ Byte oriented
- ◆ Named files
- ◆ Users protected from each other
- ◆ Robust to machine failures



File Structure Alternatives

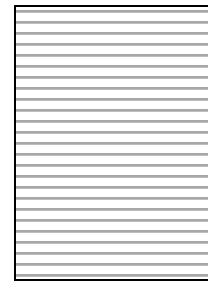
◆ Byte sequence

- Read or write a number of bytes
- Unstructured or linear



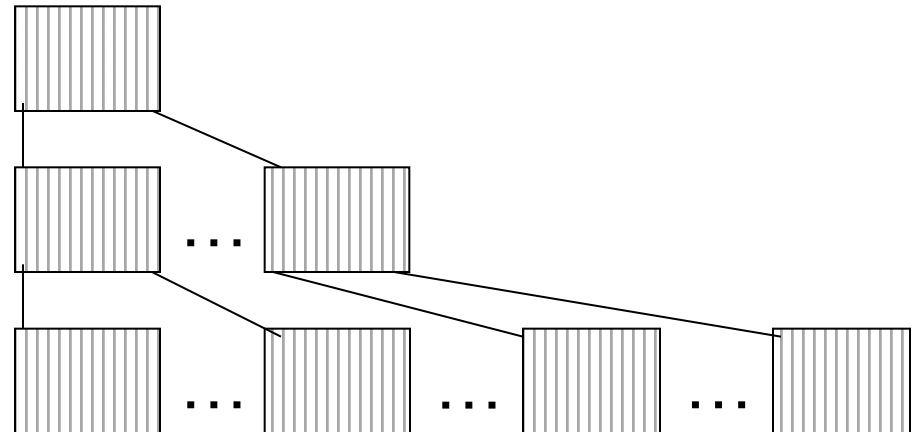
◆ Record sequence

- Fixed or variable length
- Read or write a number of records



◆ Tree

- Records with keys
- Read, insert, delete a record (typically using B-tree)



File Types

- ◆ ASCII
- ◆ Binary data
 - Records, trees
 - Documents, music, video, etc.
 - An Unix executable file
 - header: magic number, sizes, entry point, flags
 - text
 - data
 - relocation bits
 - symbol table
- ◆ Devices



File Operations

- ◆ Operations for “sequence of bytes” files
 - Create: create a mapping from a name to bytes
 - Delete: delete the mapping
 - Open: authentication
 - Close: finish accessing a file
 - Seek: jump to a particular location in a file
 - Read: read some bytes from a file
 - Write: write some bytes to a file
 - A few more on directories: talk about this later
- ◆ Implementation goal
 - Operations should have as few disk accesses as possible and have minimal space overhead



Access Patterns

- ◆ Sequential (the common pattern)
 - File data processed sequentially
 - Examples
 - Editor writes out a new file
 - Compiler reads a file
- ◆ Random access
 - Address a block in file directly without passing through predecessors
 - Examples:
 - Data set for demand paging
 - Read a message in an inbox file
 - Databases
- ◆ Keyed access
 - Search for a record with particular values
 - Usually not provided by today's file systems
 - Examples
 - Database search and indexing



VM Page Table vs. File System Metadata

Page table

- ◆ Manage the mappings of an address space
- ◆ Map virtual page # to physical page #
- ◆ Check access permission and illegal addressing
- ◆ TLB does all in one cycle

File metadata

- ◆ Manage the mappings of files
- ◆ Map byte offset to disk block address
- ◆ Check access permission and illegal addressing
- ◆ All implement in software and may cause disk accesses



File System vs. Virtual Memory

◆ Similarity

- Location transparency
- Oblivious to size
- Protection

◆ File system is easier than VM

- CPU time to do file system mappings is not a big deal
- Files are dense and mostly sequential
- Page tables deal with sparse address spaces and random accesses

◆ File system is harder than VM

- Each layer of translation causes potential disk accesses
- Memory space for caching is never enough
- Range very extreme: many < 10k, some > GB
- Implementation must be very reliable



Protection Policy vs. Mechanism

- ◆ Policy is about what and mechanism is about how
- ◆ A protection system is the mechanism to enforce a security policy
 - Roughly the same set of choices, no matter what policy
- ◆ A security policy delineates what acceptable behavior and unacceptable behavior
 - Example security policies:
 - Each user can only allocate 40MB of disk
 - No one but root can write to the password file
 - You cannot read my mail



Protection Mechanisms

◆ Authentication

- Make sure we know who we are talking to
 - Unix: password
 - Credit card companies: social security # + mom's name
 - Bar's: driver's license
- Theme?

◆ Authorization

- Determine if x is allowed to do y
- Need a simple database

◆ Access enforcement

- Enforce authorization decision
- Must make sure there are no loopholes
- This is difficult



Authentication

- ◆ Usually done with passwords
 - This is usually a relatively weak form of authentication, since it's something that people have to remember
 - Often based on things like girlfriend/boyfriend name
- ◆ Passwords should not be stored in a directly-readable form
 - Use some sort of one-way-transformation (a “secure hash”) and store that
 - In /etc/passwd or /etc/shadow, gibberish associated with each name. That is the password
- ◆ Problem: to prevent guessing (“dictionary attacks”) passwords should be long and obscure
 - Unfortunately easily forgotten and usually written down
- ◆ What are the alternatives?



Protection Domain

- ◆ Once identity known, what is Bob allowed to do?
 - More generally: must be able to determine what each “principal” is allowed to do with what
- ◆ Can be represented as an “protection matrix” with one row per domain, one column per resource
- ◆ What are the pros and cons of this approach?

	File A	Printer B	File C
Domain 1	R	W	RW
Domain 2	RW	W	...
Domain 3	R	...	RW



Access Control Lists (ACLs)

- ◆ By column: For each object, indicate which users are allowed to perform which operations
 - In most general form, each object has a list of <user,privilege> pairs
- ◆ Access control lists are simple, and are used in almost all file systems
 - Owner, group, world
- ◆ Implementation
 - Stores ACLs in each file
 - Use login authentication to identify
 - Kernel implements ACLs
- ◆ What are the issues?



Capabilities

- ◆ By rows: For each user, indicate which files may be accessed and in what ways
 - Store a lists of <object, privilege> pairs which each user.
 - Called a *Capability List*
- ◆ Capabilities frequently do both naming and protection
 - Can only “see” an object if you have a capability for it.
 - Default is no access
- ◆ Implementation
 - Capability lists
 - Architecture support
 - Stored in the kernel
 - Stored in the user space but in encrypted format
 - Checking is easy: no enumeration
- ◆ Issues with capabilities?



Access Enforcement

- ◆ Use a trusted party to
 - Enforce access controls
 - Protect authorization information
- ◆ Kernel is the trusted party
 - This part of the system can do anything it wants
 - If it has a bug, the entire system can be destroyed
 - Want it to be as small & simple as possible
- ◆ Security is only as strong as the weakest link in the protection system



Some Easy Attacks

◆ Abuse of valid privilege

- On Unix, super-user can do anything. Read your mail, send mail in your name, etc.
- If you delete the code for COS318 project 5, your partner is not happy

◆ Spoiler/Denial of service (DoS)

- Use up all resources and make system crash
- Run shell script to: “while(1) { mkdir foo; cd foo; }”
- Run C program: “while(1) { fork(); malloc(1000)[40] = 1; }”

◆ Listener

- Passively watch network traffic. Will see anyone’s password as they type it into telnet. Or just watch for file traffic: Will be transmitted in plaintext.



No Perfect Protection System

- ◆ Protection can only increase the effort needed to do something bad
 - It cannot prevent bad things from happening
- ◆ Even assuming a technically perfect system, there are always ways to defeat
 - burglary, bribery, blackmail, bludgeoning, etc.
- ◆ Every system has holes
 - It just depends on what they look like



Summary

- ◆ Storage hierarchy is complex
 - Reliability, security, performance and cost
 - Many things are hidden, but the world is becoming tapeless
- ◆ Primary
 - Network file system
 - Local file system
 - Volume manager
- ◆ Protection
 - We basically live with access control list
 - More protection is needed in the future

