# COS 318: Operating Systems

# Storage and File Hierarchy

Kai Li and Andy Bavier
Computer Science Department
Princeton University

(http://www.cs.princeton.edu/courses/cos318/)

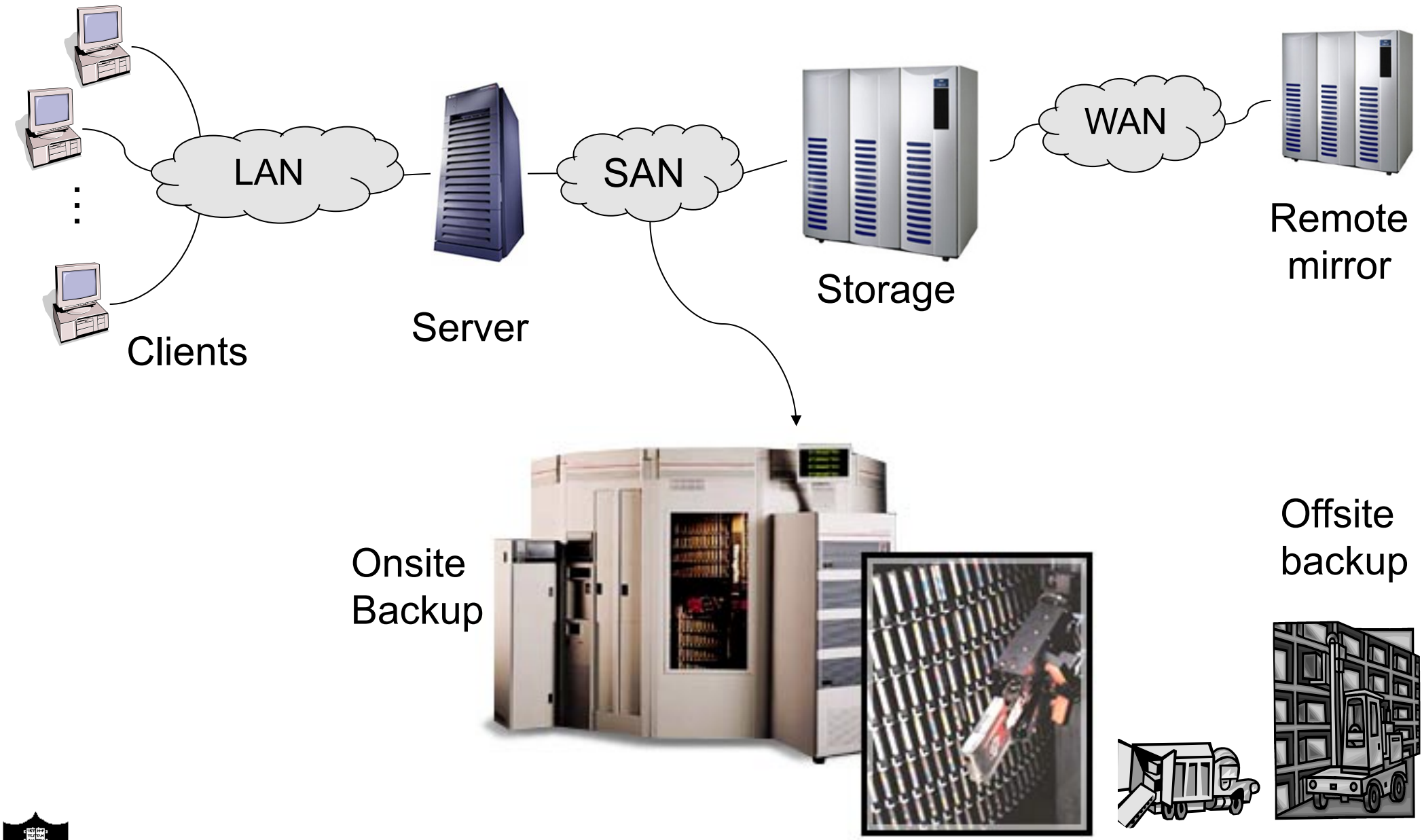
# Project 5

- New extension to December 8, Sunday 11:55pm
- Tonight: Q&A session in CS 105 7:30-830pm

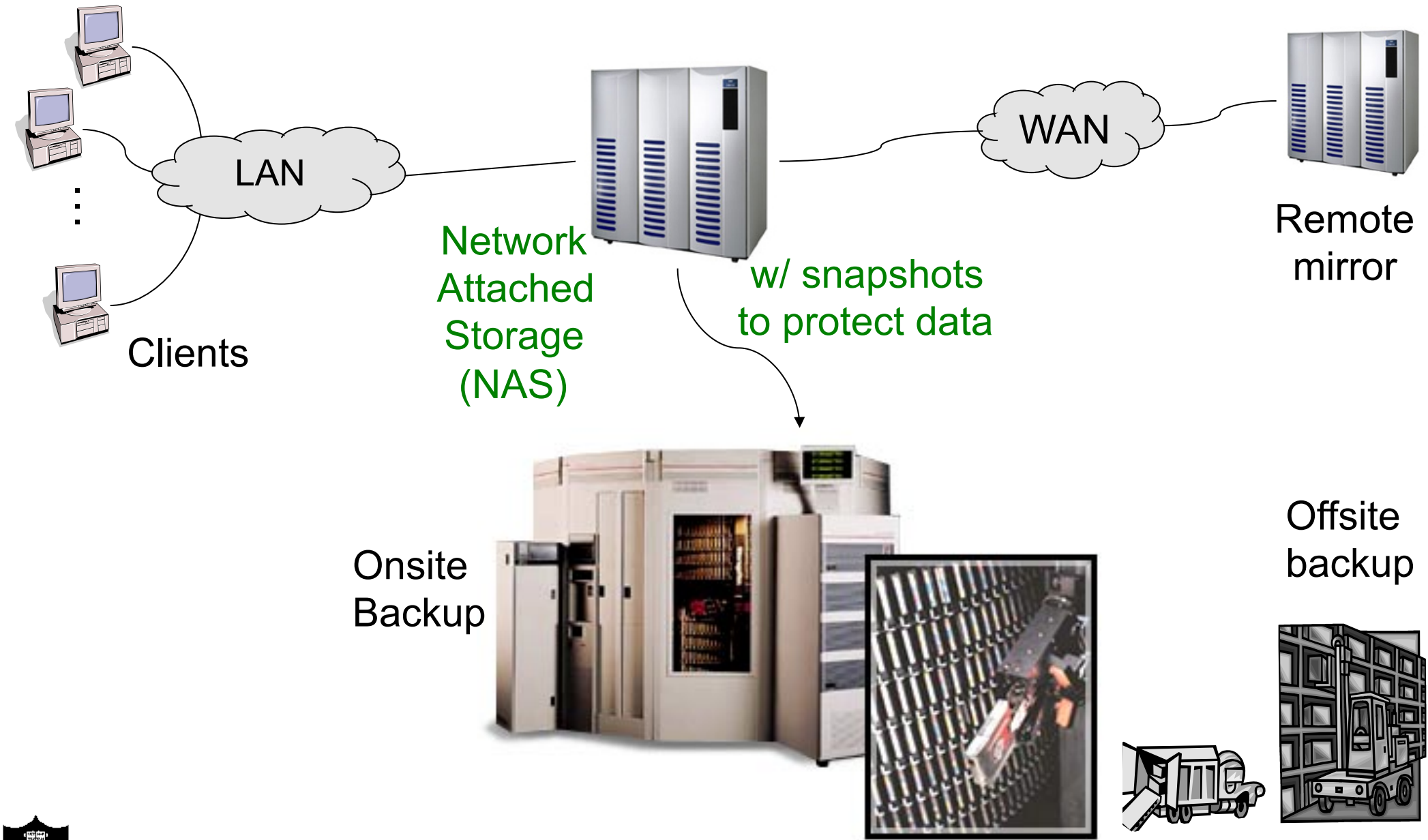
# Topics

- Storage hierarchy
- File system abstraction
- File system protection

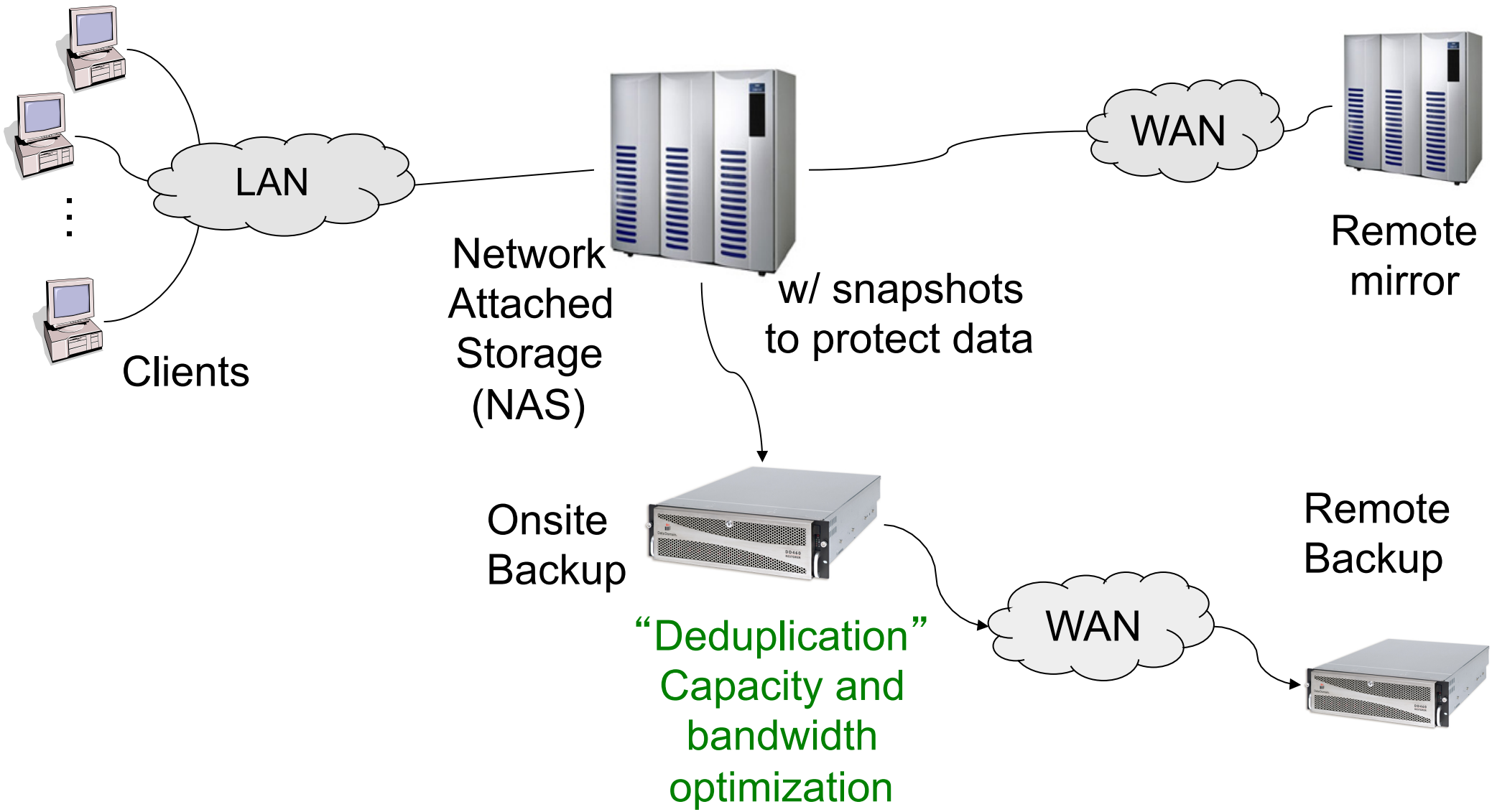
# Traditional Data Center Storage Hierarchy



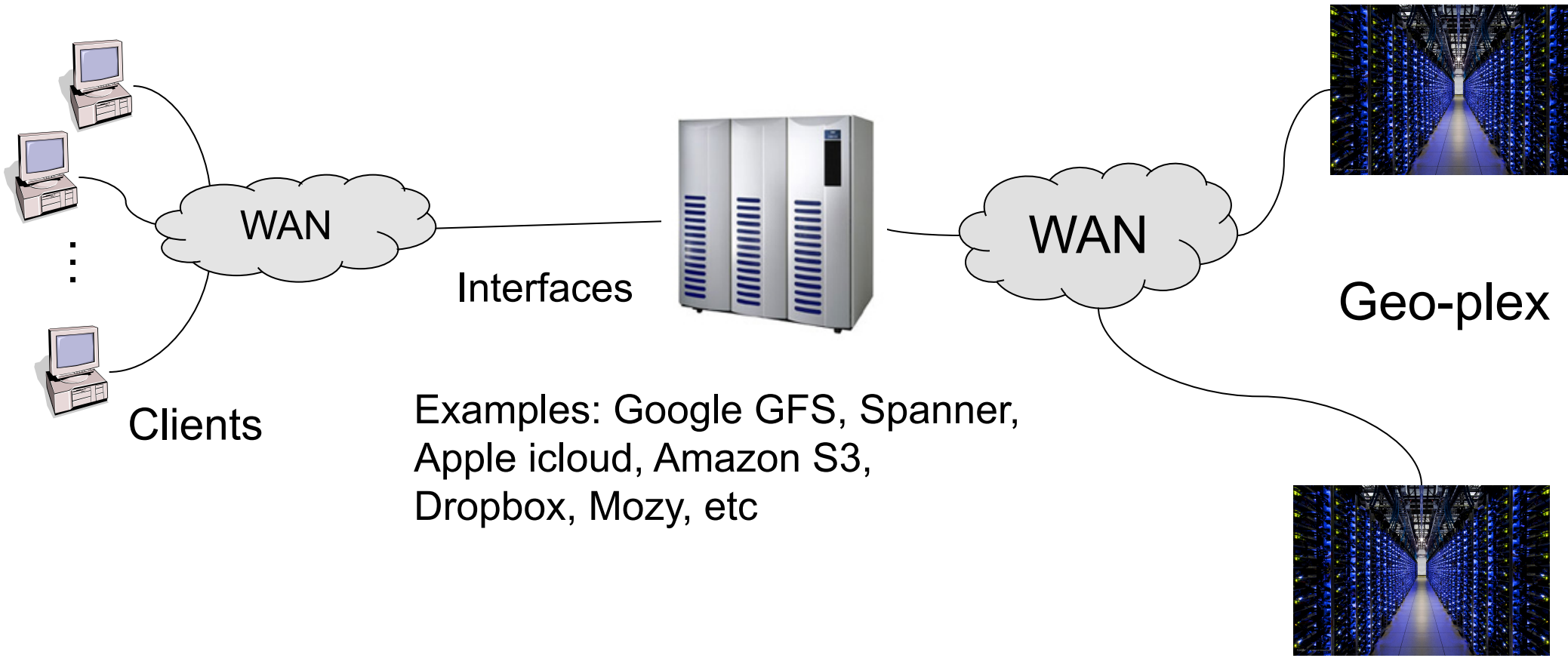
# Evolved Data Center Storage Hierarchy



# New Data Center Storage Hierarchy

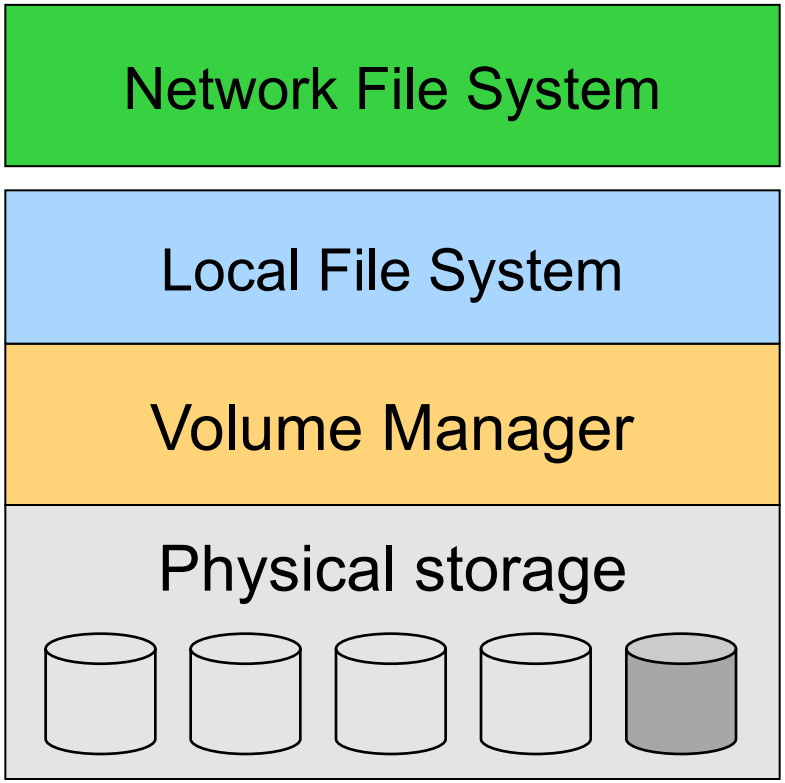


# "Public Cloud" Storage Hierarchy



# Revisit File System Abstractions

- Network file system
  - Map to local file systems
  - NFS, CIFS, etc
- Local file system
  - Implement file system abstraction on block block storage
- Volume manager
  - Logical volume of block storage
  - Map to physical storage
  - RAID and reconstruction
- Physical storage
  - Previous lectures




# Volume Manager

- Group multiple storage partitions into a logical volume
  - Grow or shrink without affecting existing data
  - Virtualization of capacity and performance
- Reliable block storage
  - Include RAID, tolerating device failures
  - Provide error detections at block level
- Remote abstraction
  - Block storage in the cloud
  - Remote volumes for disaster recovery
  - Remote mirrors can be split or merged for backups
- How to implement?
  - OS kernel: Windows, OSX, Linux, etc.
  - Storage subsystem: EMC, Hitachi, HP, IBM, NetApp

# Block Storage vs. Files

Disk/Volume abstraction

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

- Support file systems, database systems, etc.

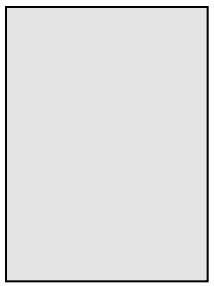
File abstraction

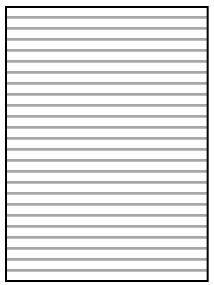
- Byte oriented
- Named files
- Users protected from each other
- Robust to machine failures

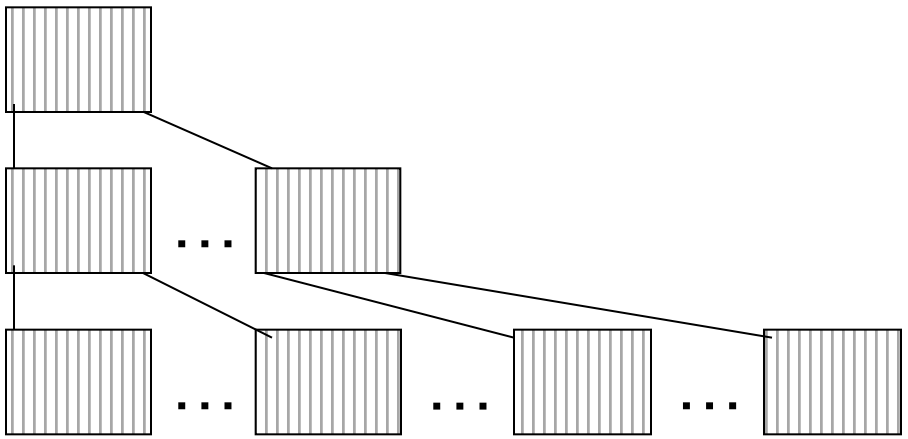
- Emulate block storage interface

# File Structures

- Byte sequence
  - Read or write N 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
  - Record
  - Tree
  - An Unix executable file
    - header: magic number, sizes, entry point, flags
    - text
    - data
    - relocation bits
    - symbol table
- Devices
- Everything else in the system

# File Operations

- Operations for “sequence of bytes” files
  - Create: create a file (mapping from a name to a file)
  - Delete: delete a file
  - 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 challenges
  - Few disk accesses
  - Minimal space overhead

# Access Patterns

- Sequential (the common pattern)
  - File data processed sequentially
  - Example: Editor writes out a file
- Random access
  - Access a block in file directly
  - Example: Read a message in an inbox file
- 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
- Implement in software, may cause I/Os

# File System vs. Virtual Memory

- Similarity
  - Location transparency
  - Oblivious to size
  - Protection
- File system is easier than VM
  - File system mappings can be slow
  - Files are dense and mostly sequential
  - Page tables deal with sparse address spaces and random accesses
- File system is more difficult than VM
  - Each layer of translation causes potential I/Os
  - Memory space for caching is never enough
  - File size range vary: many < 10k, some > GB
  - Implementation must be reliable

# Protection Policy vs. Mechanism

- Policy is about what
- Mechanism is about how
- A protection system is the mechanism to enforce a security policy
  - Same set of choices, no matter what policies
- A security policy defines acceptable behaviors and unacceptable behaviors
  - Example security policies:
    - Each user can only allocate 4GB of disk storage
    - No one but root can write to the password file
    - A user is not allowed to read others' mail files

# Protection Mechanisms

- Authentication
  - Identity check
    - Unix: password
    - Credit card: last 4 digits of credit card # + SSN + zipcode
    - Airport: driver's license or passport
- 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

# Authentication

- Usually done with passwords
  - Relatively weak, because you must remember them
- Passwords are stored in an encrypted form
  - Use a "secure hash" (one way only)
- Issues
  - Passwords should be obscure, to prevent "dictionary attacks"
  - Each user has many passwords
- Alternatives?

# Protection Domain

- Once identity known, provides rules
  - E.g. what is Bob allowed to do?
- Protection matrix: domains vs. resources

| | File A | Printer B | File C |
|---|---|---|---|
| Domain 1 | R | W | RW |
| Domain 2 | RW | W | … |
| Domain 3 | R | … | RW |

# By Columns: Access Control Lists (ACLs)

- Each object has a list of <user, privilege> pairs
- ACL is simple, implemented in most systems
  - Owner, group, world
- Implementation considerations
  - Stores ACLs in each file
  - Use login authentication to identify
  - Kernel implements ACLs
- Any issues?

# By Rows: Capabilities

- For each user, there is a capability list
  - A lists of <object, privilege> pairs
- Capabilities provide both naming and protection
  - Can only “see” an object if you have a capability
- Implementation considerations
  - Architecture support
  - Capabilities stored in the kernel
  - Capabilities stored in the user space in encrypted format
- Issues?

# 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 there is a bug, the entire system could 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

# No Perfect Protection System

- Only make it difficult to do bad things
  - It cannot prevent bad things
- There are always ways to defeat
  - burglary, bribery, blackmail, bludgeoning, etc.
- Every system has holes

# Summary

- Storage hierarchy can be complex
  - Reliability, security, performance and cost
  - Many things are hidden
- Primary storage
  - Volume of block storage
  - Local file system
  - Network file system
- Protection
  - ACL is the default in file systems
  - More protection is needed because we are in the cloud