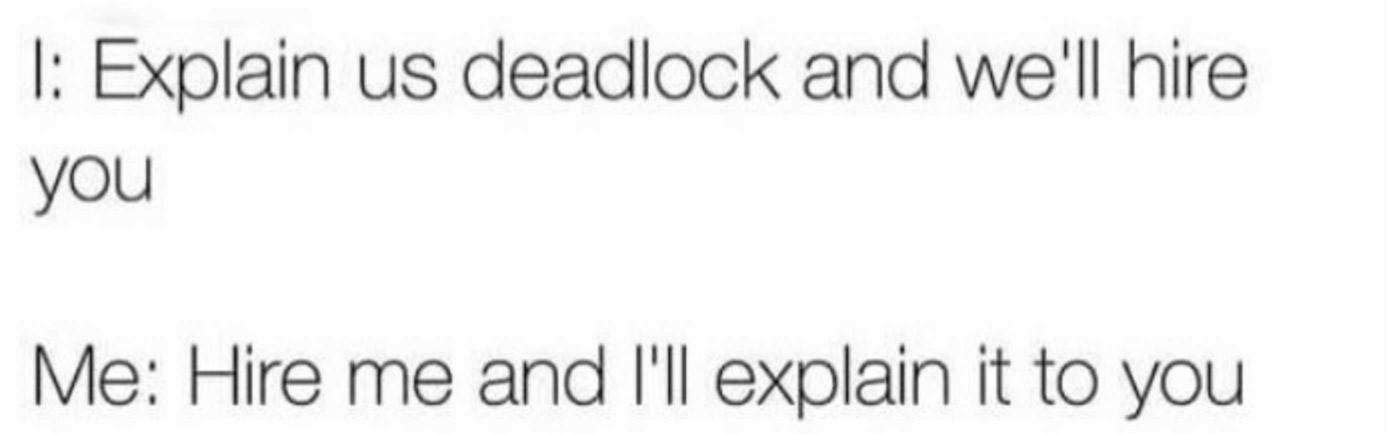
UPE Tutoring:

CS 111 Final Review

Sign-in https://goo.gl/UFpb9T

Slides link available upon sign-in

Deadlocks!



I: Explain us deadlock and we'll hire you

Me: Hire me and I'll explain it to you

Deadlocks

- When two things are waiting for each other to finished before they start
- The Dining Philosopher’s problem
- Caused by hold-and-wait on common resources:
  - Locks/mutexes/semaphores
  - Memory
- Bad dependency management
- 4 Conditions:
  - Mutual Exclusion
  - Incremental Allocation
  - No Pre-emption
  - Circular waiting

Deadlock

```
process A(){
        acquireX();      ← Step1
        acquireY():      ← Step3

        //do stuff

        releaseY();
        releaseX();
}
```

```
        process B(){
Step2 →         acquireY();
                acquireX():

                //do stuff

                releaseX();
                releaseY();
        }
```

Problem

Why is hold and wait a necessary condition for deadlock? Describe one method that could be used to avoid the hold-and-wait condition to thus avoid deadlock.

Problem

Why is hold and wait a necessary condition for deadlock? Describe one method that could be used to avoid the hold-and-wait condition to thus avoid deadlock.

Answer: *Without hold-and-wait, the deadlock couldn't occur since the lock would be released eventually (no holding!) and other processes would not be stuck forever waiting for the lock. One method is to release other held locks if trying to acquire the next lock fails. That way, other processes won't have to wait for the first lock.*

I/O

- Devices and Drivers
  - Operate as interrupt driven
  - Connected to a bus
- Disk I/O
  - Reading and writing data is a big time consumer
  - Direct Memory Access: Allows busses and devices to directly move memory around without having to go through the cpu, though this takes up bus time
  - If device processes want reads/writes, they make a request and block

I/O

- Disk reads can be slow
    - Disk seek and rotation overheads are unwanted
    - Good things:
        - Caching and buffering are friends :)
        - Larger transfers
        - Queued requests
        - Combining nearby requests
        - Write-back (sometimes)
    - Double buffering

Problem

Describe an optimization related to making writes to a file system perform better. When does this optimization help? What complexities does this optimization add to the operating system behavior?

Problem

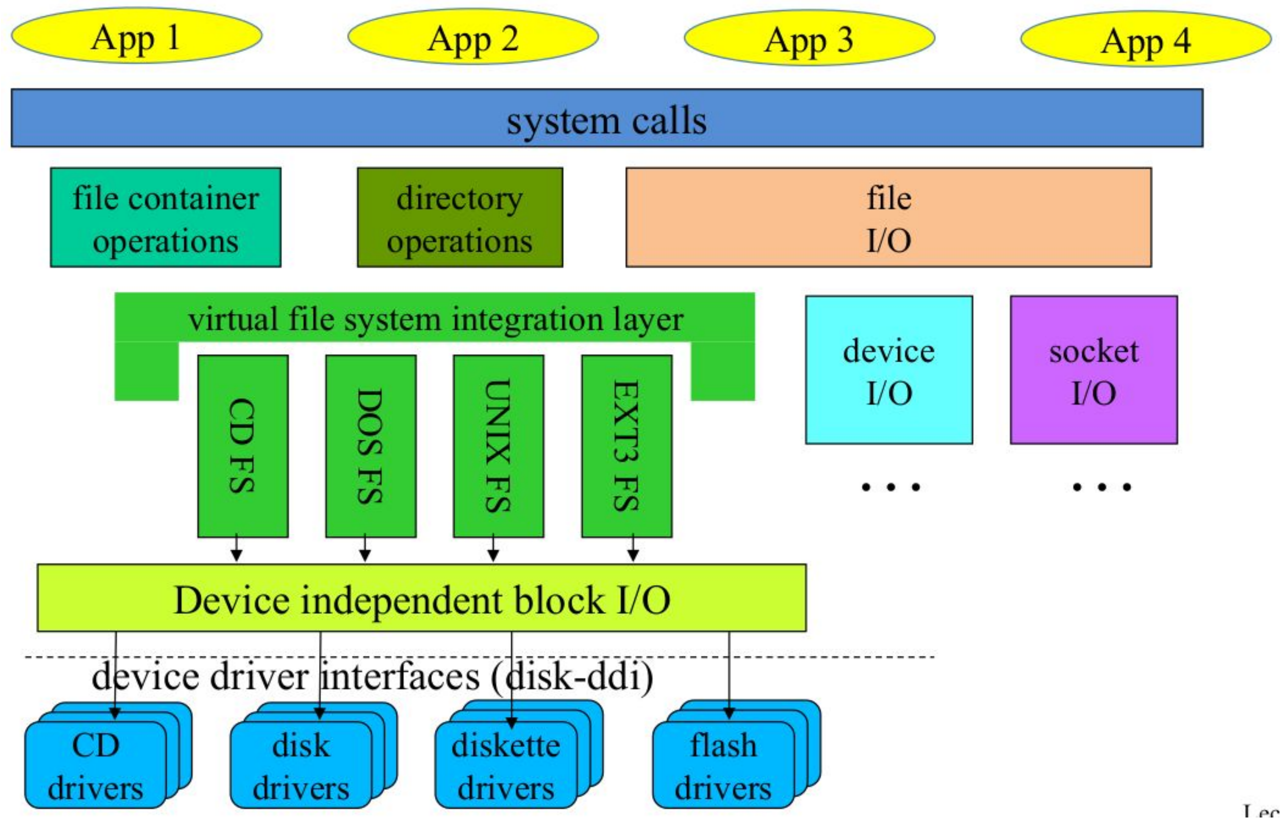
Describe an optimization related to making writes to a file system perform better. When does this optimization help? What complexities does this optimization add to the operating system behavior?

Answer: *There are many right answers. One solution - caches and buffering. Helps reduce read/write disk operations. Makes OS more complicated since it has to manage when to actually interact with the disk.*

File System Abstractions

- File System API: interface between **user and kernel** (syscall)
    - File Container Operations: manipulate file as objects (ignore contents of file)
        - `stat(2), chmod(2), ...`
    - Directory Operations: organization of file system hierarchy
        - `mkdir(2), getcwd(2), ...`
    - File I/O Operations: contents of the file
        - `open(2), close(2), read(2), write(2), lseek(2), ...`

The Virtual File System Layer



File System Layer

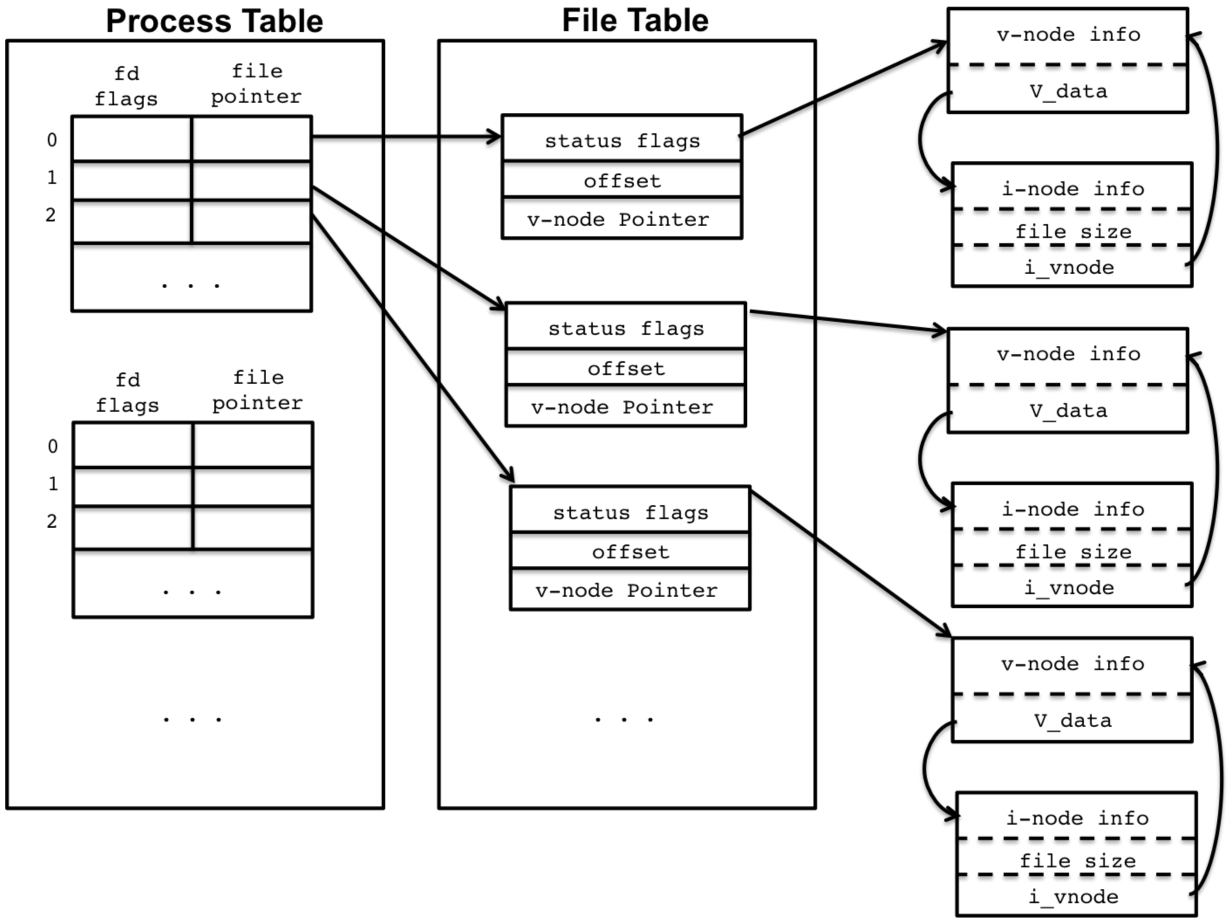
- **Virtual File System Layer**: uniform interface for different **file system implementations** (inside kernel)
  - E.g. DOS, ext4, NFS, procfs, tmpfs, ...
  - Each file system implemented by a kernel module
  - All file system modules implement the same basic operations
- **Block I/O Layer**: uniform interface for **Block I/O** (interact with hardware)
  - Make all disks look the same
    - Standard operations on block device
    - Map logical block numbers to device addresses
    - Encapsulate all the particulars of device support
  - Advantage
    - Unified buffer cache for disk data
    - Automatic buffer management: allocation, deallocation, automatic write-back

File System Layer Design

- Why Virtual File System
  - Multiple file systems
    - Different storage devices, different services, different purposes
    - On seasnet: `$ mount   #` shows currently mounted file systems
- Page Cache: In memory cache representing structures on disk
  - Caches file control structures in memory
  - File access exhibits reference locality
  - Common cache reduce number of disk accesses
  - Shared among different processes
    - Some structures are private to processes: file offsets (cursor), ...
- Fun with linux disk cache: https://www.linuxatemyram.com/play.html

Security - Cryptography

- Altering bits in a controlled way which improves the security of a system
- Symmetric - use a key both parties have to decrypt and encrypt
  - If the key gets out… you're "got". (ex. Heartbleed controversy)
  - DES vs AES
- Asymmetric - Public and private keys (ex. RSA, Elliptic Curve)
  - I can sign with your public and you are the only one who can decrypt with private
  - I can sign with my private and you can decrypt knowing the message is from me
    - Windows Update Example
- Hashing Functions
  - Passwords, as before
  - Message Integrity
  - "Proof of work"

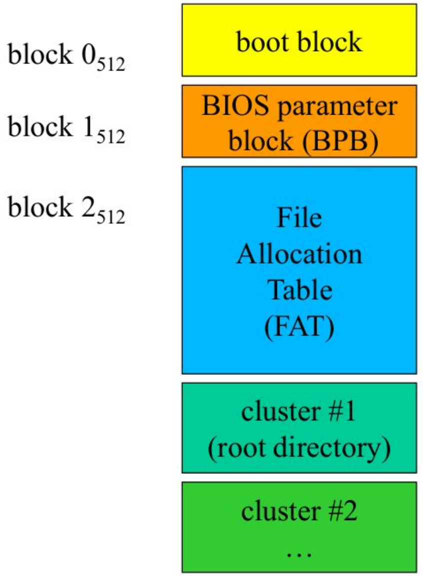

Process Table
fd flags
file pointer
0
1
2
. . .
fd flags
file pointer
0
1
2
. . .
. . .
File Table
status flags
offset
v-node Pointer
status flags
offset
v-node Pointer
status flags
offset
v-node Pointer
. . .
v-node info
V_data
i-node info
file size
i_vnode
v-node info
V_data
i-node info
file size
i_vnode
v-node info
V_data
i-node info
file size
i_vnode

DOS FAT File System

- Divide disk into “clusters”
- FAT (File Allocation Table): contains the number of next cluster in file
  - Each entry corresponds to a cluster
- Characteristics
  - To access (seek) **n**th “cluster” of a file: O(**n**)
  - Size of FAT determines the max files size
  - No support for sparse files
    - Requires all “clusters” to be present

    -
      ```
      $ dd if=/dev/zero of=sparsefile bs=1 count=0 seek=128G
      $ ls -hl sparsefile
      $ du -h sparsefile
      ```

DOS FAT File System

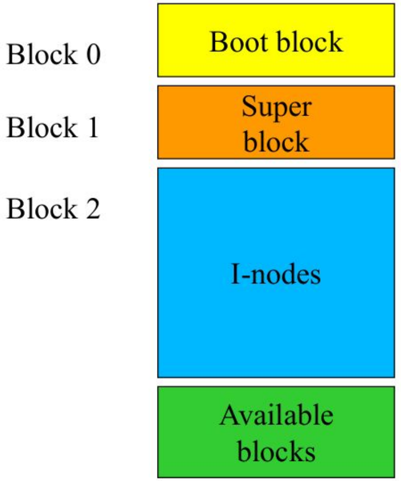

block 0_{512}
block 1_{512}
block 2_{512}
boot block
BIOS parameter block (BPB)
File Allocation Table (FAT)
cluster #1 (root directory)
cluster #2
…

Cluster size and FAT length are specified in the BPB

Data clusters begin immediately after the end of the FAT

Root directory begins in the first data cluster

Unix File System



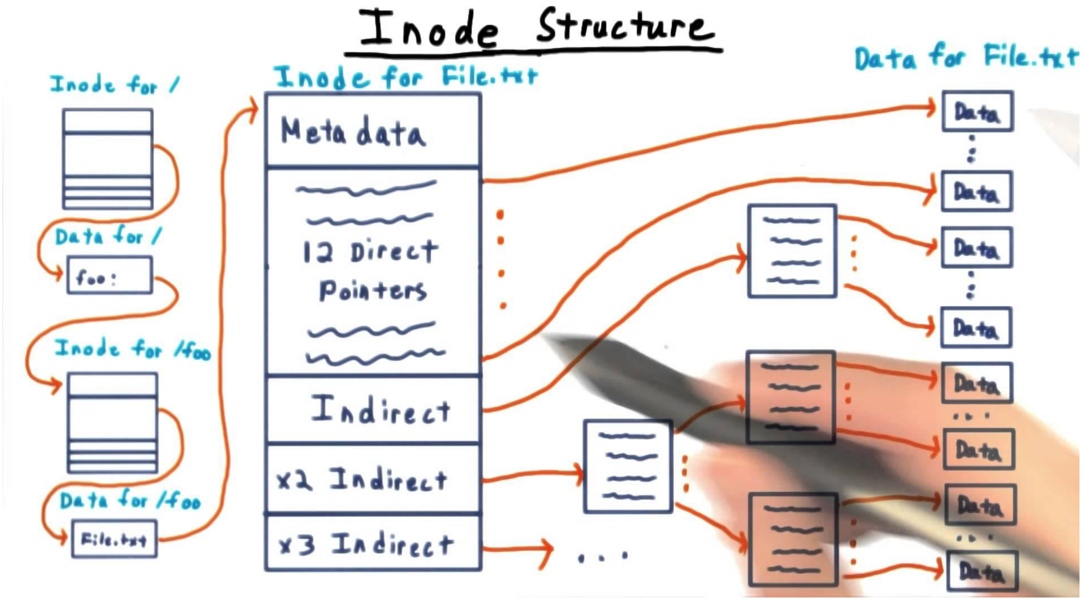
Block size and number of I-nodes are specified in super block

I-node #1 (traditionally) describes the root directory

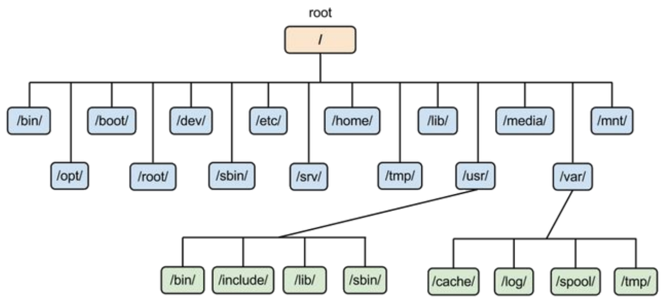
Data blocks begin immediately after the end of the I-nodes.

Unix File System

- Inode: **fixed size metadata** that stores attributes and disk block locations of the object
  - `$ ls -il file  # shows inode number of file`
  - Block pointers: direct pointers, indirect pointer
    - First ~10 blocks can be found without extra I/O
    - 1-3 extra I/O per thousand pages
    - Largest possible file
      - Each direct pointer: 1 block
      - Each indirect pointer: 1024 blocks (assuming each block can hold 1024 block pointers)
      - Each doubly/triply indirect pointer, ...
    - Support sparse file: with empty block pointers (similar to NULL pointer)


Inode Structure
Inode for /
Data for /
foo:
Inode for /foo
Data for /foo
File.txt
Inode for File.txt
Meta data
12 Direct Pointers
Indirect
x2 Indirect
x3 Indirect
Data for File.txt
Data

File System Hierarchy



- Directory: s special kind of file
  - Contains multiple directory entries: name, pointer to the inode
  - One entry is special file `..` : parent directory
- Navigate File System
  - Absolute path
    - `$ namei /etc/systemd/system/default.target`
  - Relative path
    - `$ namei ../../`
- Why can't we open a file by inode?
  - Bypass Unix file permission check
    - `654321  /a/b   --wx------` (directory not readable)
    - `123456  /a/b/f -rwx------`

Hard Link vs. Symbolic Link

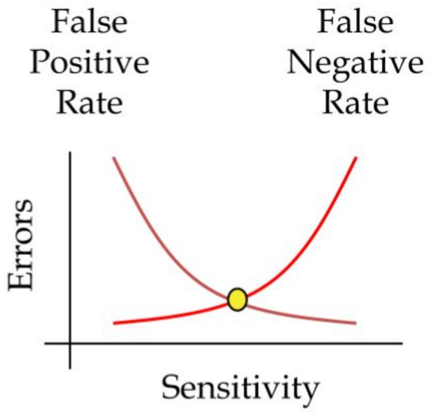
- Hard link: file exists under multiple names
  - Appear exactly the same as a normal file
  - In **inode**, there is a field storing the link count
    - When a reference to this file is removed, link count reduce by 1
    - When link count reduce to 0, the space underlying the file is freed
- Symbolic link: an **alias** to another file (just contains a pathname)
  - A special kind of file: `$ stat symlink  # shows symlink is a symbolic link`
  - Not a reference to the inode
    - Does not prevent deletion (does not affect *inode link count* of linked file)

Security - Basics

- Three Main Topics
  - Authentication - is this who he claims to be?
  - Authorisation - do we trust this person / allow him to do this action?
  - Cryptography - how can we maintain confidentiality and integrity
- Three Goals
  - Confidentiality - other people should not be able to see this
  - Integrity - you get the message I send you
  - Availability - This resource is available as specified by its interface, no one blocks it.
- The OS
  - If the OS is compromised/you can’t trust it… you’re “got”

Security - Authentication

- What you know
  - Passwords and some Challenge Response
  - Salt -> Hash
    - Why? Dictionary Attacks
- What you have
  - Phones, dongles Challenge Response
- Who you are
  - Biometric Auth.
    - False positive, False negative
    - Tweaking? Crossover Error Rate Point.
- Multi-Factor, use advantages of multiple methodologies.



Security - Authorisation

- Subject, Object, Access.
- Reference monitor - handles the algorithms to figure out who can access what when.
- Access Control List
  - Each obj. Has a list of subjects who can access.
  - Stored by files (ex: linux RWX bits for UGO)
- Capabilities
  - Each subj. Has a list of objects he can access. (ex: Android Permission Label)
  - Usually stored in OS (think Process Control Block)
- Each has Pros and Cons, and make up for where the other lacks.
- Other: Role Based Action Control

Remote Data Access: Goals and Benchmarks

Basic Goal: Client accesses data held in a remote location in the same manner that they access local data.

Benchmarks

- Transparency: Remote data access being indistinguishable from local data access
- Performance: Speed and scalability
- Cost: Capital cost and operational cost
- Capacity: Space available to the clients
- Availability: Clients can query for files at any time.

Client/Server

- Types of Client/Server models
  - Peer-to-peer:
    - No set server, each peer can potentially act as a client or a server
  - Thin client:
    - Client is very lightweight, relies on a server to do most of the work
  - Cloud services:
    - Servers are opaque to clients, clients access services provided by the servers

Remote file transfer

- Relying on non-OS based protocols to query for data from the server
- Pros
  - Requires no OS support
- Cons
  - Latency
  - No transparency

Remote Disk Access

- The remote server is seen as a local disk
- Typical architectures:
    - Storage Area Network (SCSI over Fibre Channel): fast, expensive, moderately scalable
    - iSCSI (SCSI over ethernet): moderate performance, cheap, scalable
- Pros
    - Transparency
    - Leaves performance/reliability/availability problems to the server
- Cons
    - Inefficient fixed partition space allocation
    - Can’t support file sharing amongst clients
    - Message losses due to network errors become file system errors

Remote File Access

- The remote server is seen as a local file system
- Pros
    - Transparency
    - Functional encapsulation
    - Supports multi-client file sharing
    - Good performance
- Cons
    - Requires implementation in the OS
    - Introduces complexities to client/server
- Compared to Distributed File Systems
    - Remote file access is more simple
    - Distributed file system has higher performance and scalability

Security for Remote File Systems

- Issues
    - Privacy and integrity of data on the network
        - Solution: Encrypt the data sent over the network
    - Authentication of remote users
        - Solutions:
            - Anonymous Access: No authentication required for reads
            - P2P: Leave the authentication up to the clients
            - Server Authentication: Server authenticates clients
            - Domain Authentication: Trusted third party handles authentication

Reliability and Availability

- Reliability
    - Implemented via redundancy (Mirroring, Parity, Erasure Coding)
    - Important to automatically recover after failure
- Availability and Fail-Over
    - Fail-Over: Being able to detect failed servers and transferring requests to a working server
    - Failure Detection can be done either at client end or server end
    - Network protocols that do not save state make fail-over easy

Remote File System Performance

- Disk Bandwidth Implications
  - Single server, multiple clients = limited throughput
  - Striping files across multiple servers increases scalable output
- Network delay/packet-loss decreases file system throughput
- Cost of Reads/Writes
  - Use caching to improve reads and writes at the cost of consistency
- Cost of Mirroring
  - Mirroring can be done by the server or by the client
- Direct Data Path: Client directly connected to the storage servers, improves throughput, latency, and scalability

Improving remote file systems

- Improving availability
  - Improving failure detection, fail-over speed, and recovery speed
- Improving speed
  - Minimize messages sent over the network (expensive)
- Improving scalability
  - Avoid single control points
  - Separate data plane and control plane (Direct Data Path)
  - Avoid consensus-based protocols

Good luck!

Sign-in	https://goo.gl/UFpb9T
Slides	https://goo.gl/zytWzx

Questions? Need more help?

- Come up and ask us! We'll try our best.
- UPE offers daily computer science tutoring:
  - Location: ACM/UPE Clubhouse (Boelter 2763)
  - Schedule: https://upe.seas.ucla.edu/tutoring/
- You can also post on the Facebook event page.