CSC369: Operating Systems

Fall 2014

Andrew Petersen

I hear you. Sorry about A1!

A1 marks need a review from me, and I anticipate that will happen late this week.

If you're thinking about quizzes -- everything has been returned in lab.

Thank you.

... but you're a little early.

Outline

- Errors and Recovery
  - Logs and Journaling
- Sharing: Permissions
- Caching and Mapped Memory

Removing a File

- Consider the steps required to remove a file
  - Remove the directory entry.
  - Update the inode.
  - Potentially free data blocks and the inode.
- What happens if the disk is turned off during this process?

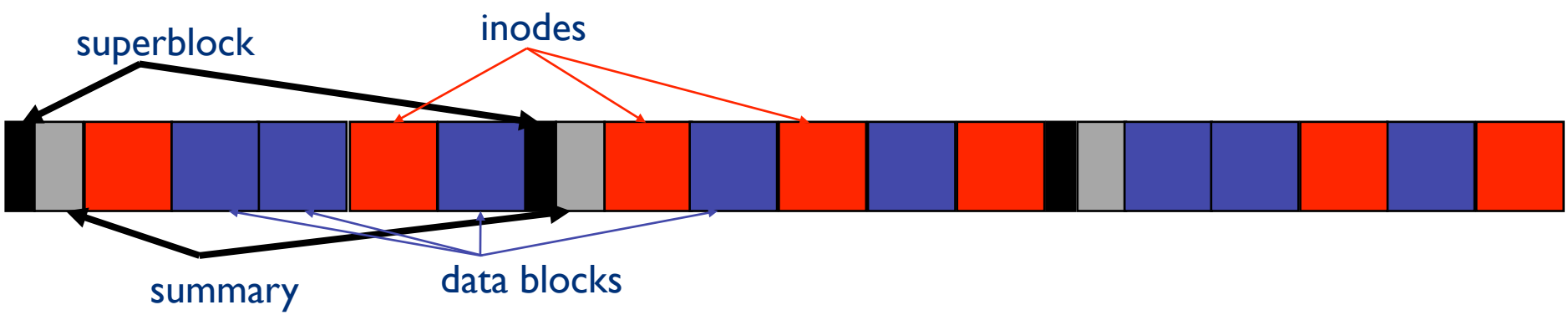
Journaling

- Keep a log of all updates to the disk.
- Write to this log before making any changes.
- Confirm the changes after they occur.

- If there is a crash, the log can be checked to make sure that the system is in a consistent state.

Log Structured File System (LFS)

- Developed by Ousterhout in 1989
- Idea: Write ***all*** file system data in a continuous log
- Uses inodes and directories from FFS
- Needs an inode map to find the inodes
- Cleaner reclaims space from overwritten or deleted blocks.



LFS Reads

- If the writes are easy, what happens to the reads?
- To read a file from disk:
  1. Read the superblock to find the index file
  2. Read the index file and find the inode-map
  3. Get the file's inode
  4. Use the inode as usual to find the file's data blocks
- Remember, we expect reads to hit in memory most of the time … after we open the file in the first place.

Other Errors

- Latent Sector Errors: damage to the disk or the content of the disk
- Corruption: by definition, errors that are not detectable by the disk itself

- These errors tend to *not* be uniformly random

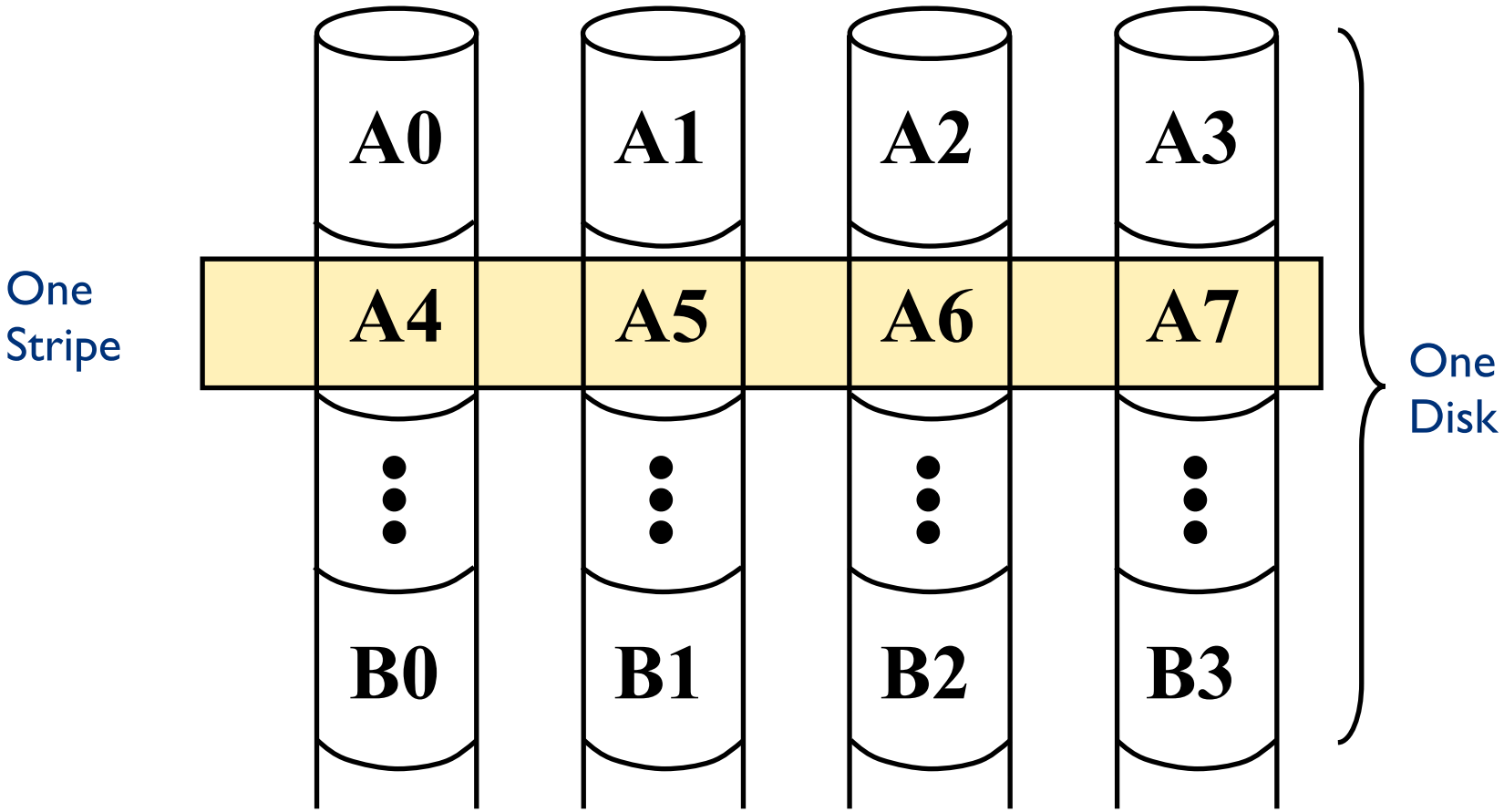
Solutions

- Checksums: the use of extra bits to identify and reverse limited LSEs.
  - The text has an excellent discussion of these, so we will not cover them further.
- Redundancy: replication of large chunks of data in case of disk failure.

RAID

- Redundant Array of Inexpensive Disks (RAID)
  - A storage system, not a file system
  - Patterson, Katz, and Gibson (Berkeley, '88)
- Idea: Use many disks in parallel to increase storage bandwidth, improve reliability
  - Files are striped across disks
  - Each stripe portion is read/written in parallel
  - Bandwidth increases with more disks
  - Better throughput for large requests

RAID 0: Disk Striping

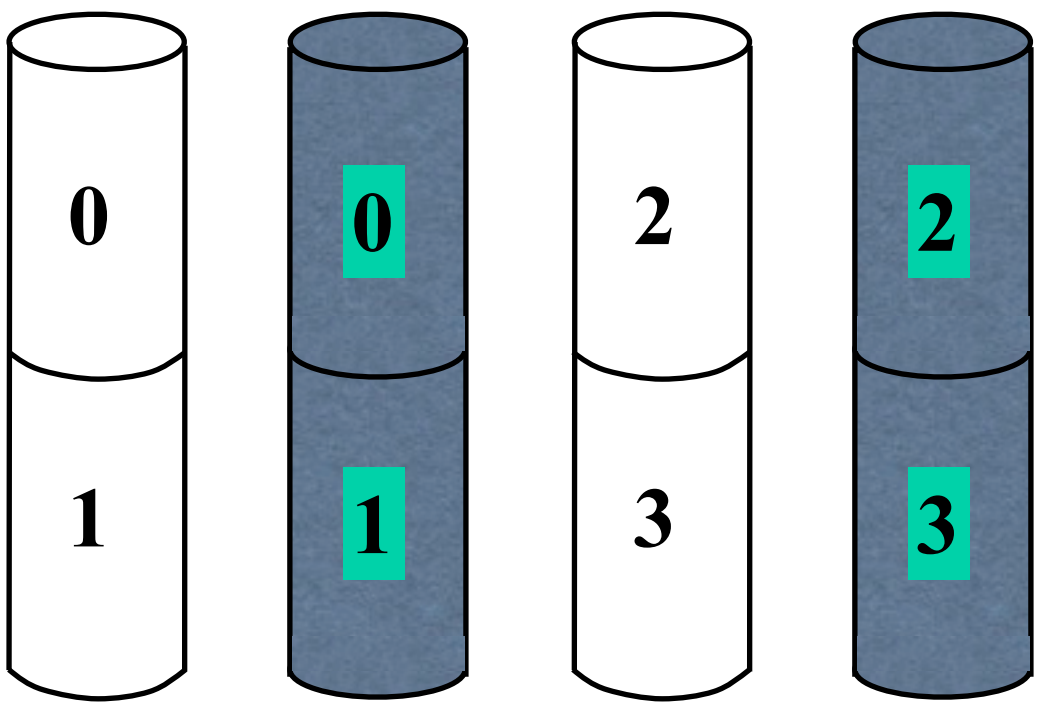


RAID 0 Challenge

- Reliability
  - More disks increases the chance of media failure (MTBF)
- Turn reliability problem into a feature
  - Add an extra disk for each block?
  - Or use one disk to store parity data?
    - XOR of all data blocks in stripe
    - Can recover any data block from all others + parity block
  - Hence "redundant" in name
  - Introduces overhead, but, hey, disks are "inexpensive"

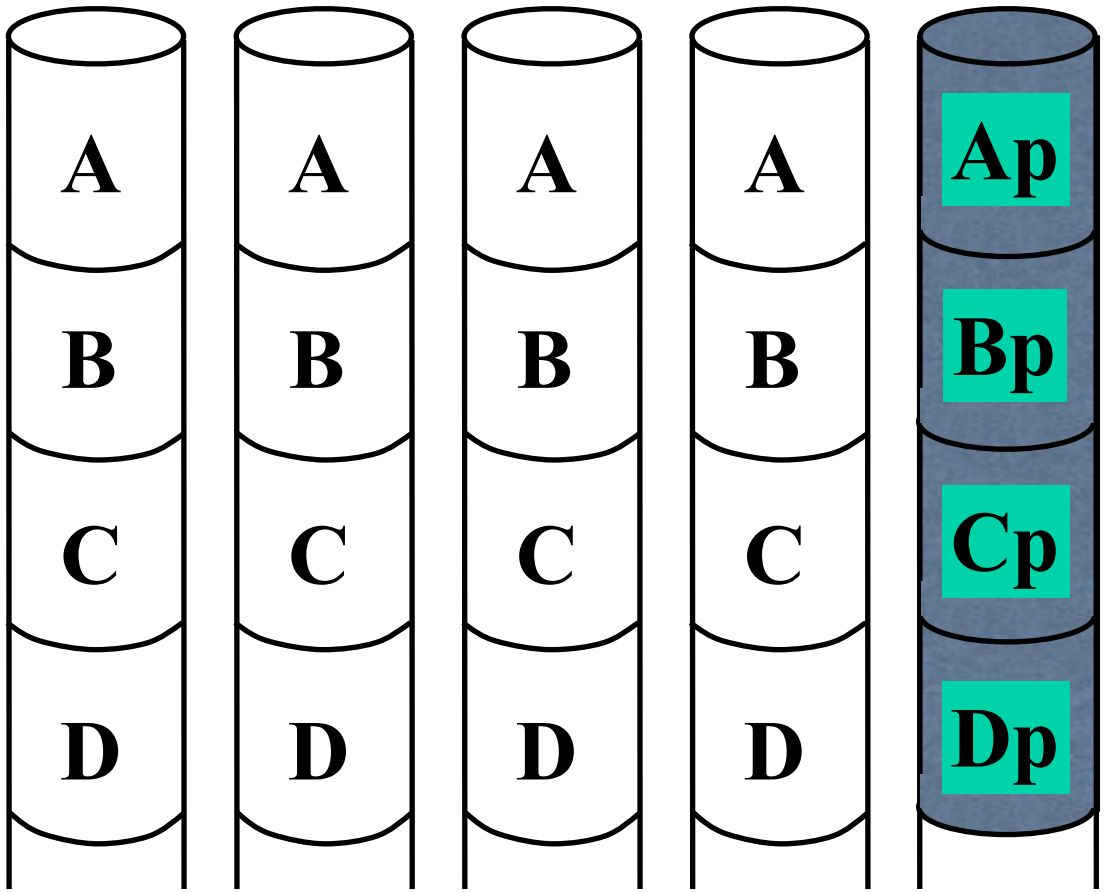
RAID Level 1: Mirroring

- Redundancy via replication, two (or more) copies
  - mirroring, shadowing, duplexing, etc.
- Write both, read either



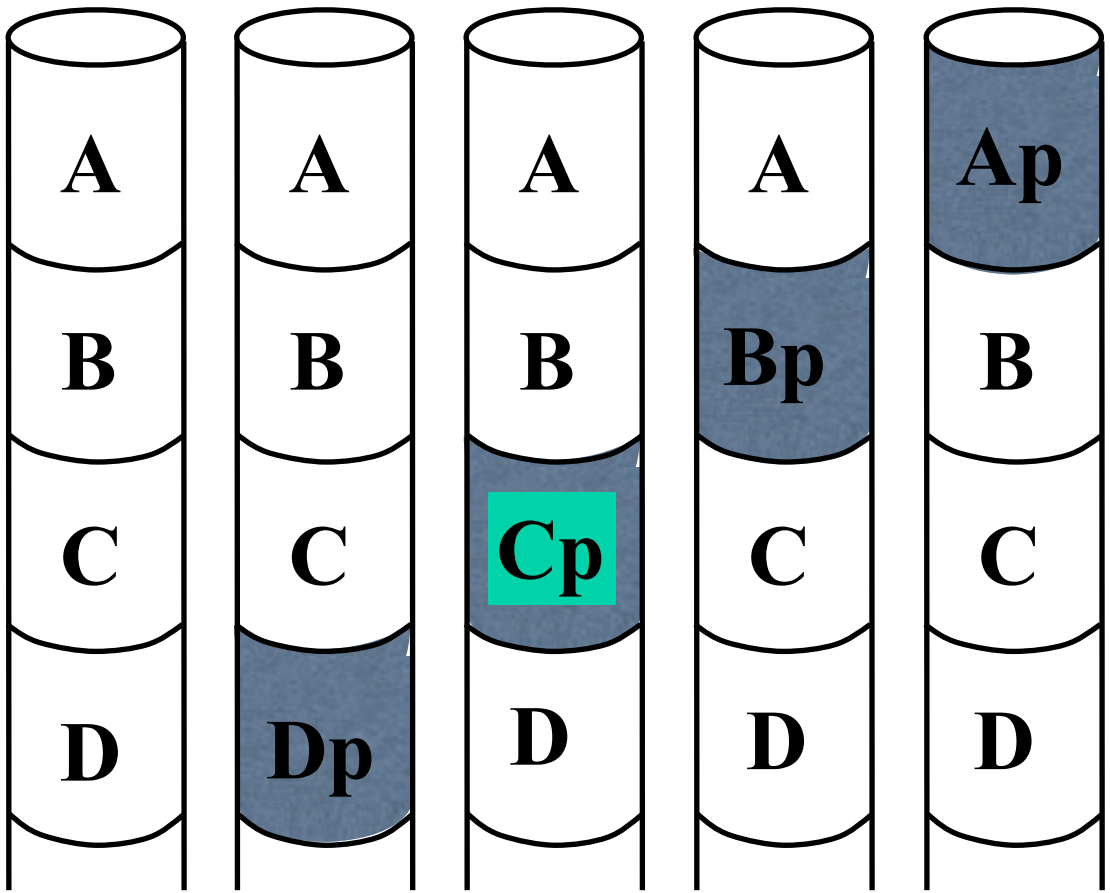
RAID 2&3: Parity Disks

- Both:
  - Very small stripe unit (single byte or word)
  - All writes update parity
  - Can correct single-bit errors
- Level 2:
  - #parity disks = Log_2 (#data disks)
  - This is overkill
- Level 3:
  - One extra disk



RAID Levels 4-6

- Levels 4-6 introduce independence:
  - Each disk may be writing different data
- Level 5 removes parity disk bottleneck
  - Distributes parity bits across all data disks
- Level 6 tolerates 2 errors



Exercise

- What does it mean for a file to be "deleted"?
- How do you know that something is deleted?
- As an investigator, what would you do to recover information from a disk?

A Favorite ...

... quote, from the article on deleting information from a drive:

"If the data is very sensitive and is stored on floppy disk, it can best be destroyed by removing the media from the disk liner and burning it, or by burning the entire disk, liner and all (most floppy disks burn remarkably well - albeit with quantities of oily smoke - and leave very little residue)."

Stop Identity Theft in 10 Minutes!

DISKSTROYER™

The World's Only Do-It-Yourself
Hard Disk Destruction Kit
What have *you* got to lose?

Now Revised & Expanded!
Now more than twice as many
tool pieces than the original kit!

Outline

- Errors and Recovery
  - Logs and Journaling
- Sharing: Permissions
- Caching and Mapped Memory

File Buffer Cache

- Fortunately, applications exhibit locality for reading and writing files

- Idea: Cache file blocks in memory to capture locality
  - This is called the file buffer cache
  - Cache is system wide, used and shared by all processes
  - Even a 4 MB cache can be very effective
- Issues
  - The file buffer cache competes with VM (tradeoff here)
  - Like VM, it has limited size
  - Need replacement algorithms again (LRU is common)

Caching Writes

- On a write, some applications expect that data makes it through the buffer cache and onto the disk
  - As a result, writes are often slow even with caching
- Several ways to compensate for this
  - write-behind
    - Maintain a queue of uncommitted blocks
    - Periodically flush the queue to disk
    - Unreliable and may break program expectations
  - Battery backed-up RAM (NVRAM)
    - As with write-behind, but maintain queue in NVRAM
    - Expensive
  - Log-structured file system
    - Always write contiguously at the end of the previous write

Read Ahead

- Many file systems implement read ahead
  - FS predicts that the process will request next block
  - FS goes ahead and requests it from the disk
  - This can happen while the process is computing on previous block
    - Overlap I/O with execution
  - When the process requests block, it will be in cache
  - Compliments the on-disk cache, which also is doing read ahead

- For sequentially accessed files, can be a big win
  - Unless blocks for the file are scattered across the disk
  - File systems try to prevent that, though (during allocation)

Mapped Memory Exercise

Outline

- Errors and Recovery
  - Logs and Journaling
- Sharing: Permissions
- Caching and Mapped Memory

File Sharing

- File sharing is incredibly important for getting work done
  - Basis for communication and synchronization
  - Uh-oh ... there's that word again ...

- Two key issues when sharing files
  - Semantics of concurrent access
    - What happens when one process reads while another writes?
    - What happens when two processes open a file for writing?
  - Protection

Protection

- File systems must implement some kind of protection system
  - Who can access a file?
  - How they can access it?
  - More generally…
    - Objects are "what", subjects are "who", actions are "how"

- A protection system dictates whether a given action performed by a given subject on a given object should be allowed
  - You can read and/or write your files, but others cannot
  - You can read "/etc/motd", but you cannot write it

Types of Access

- None
- Knowledge
- Execution
- Reading
- Appending
- Updating
- Changing Protection
- Deletion

- Unix provides only Read/Write/Execute permissions

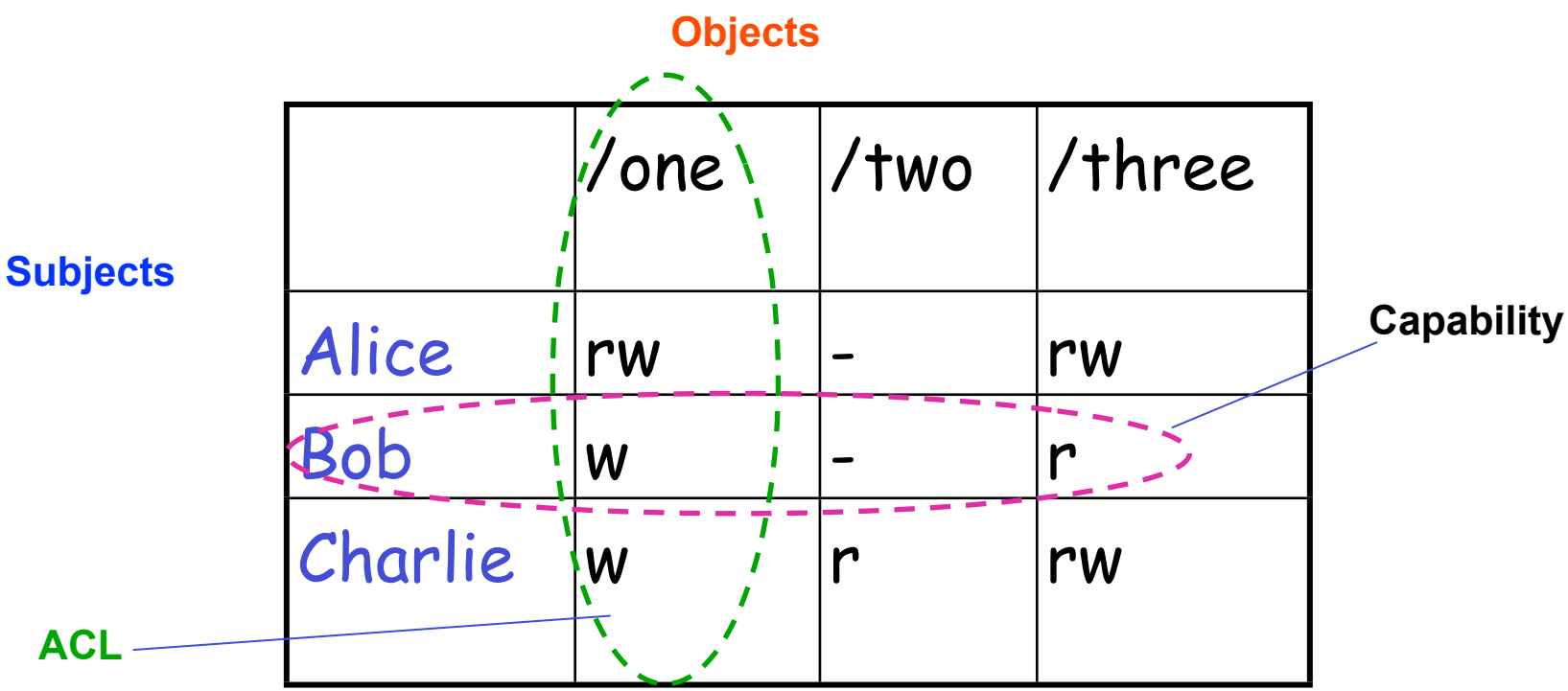
Representing Protection

Access Control Lists (ACL)

- For each object, maintain a list of subjects and their permitted actions

Capabilities

- For each subject, maintain a list of objects and their permitted actions



ACLs and Capabilities

- The approaches differ only in how the table is represented
  - What does UNIX use?

- Capabilities are easier to transfer
  - They are like keys and can be handed off
- In practice, ACLs are easier to manage
  - Object-centric (each file has its own ACL), so easy to grant or revoke
  - To revoke capabilities, we have to keep track of all subjects that have the capability
- ACLs have a problem when objects are heavily shared
  - The ACLs become very large
  - How could we mitigate this?

Summary

- The main abstraction is the file
  - Files have a basic set of operations
- For organizational purposes, we use a special type of file called a directory
  - This leads to the ideas of a path and working directory
  - Files may need to reside in different places, so we create links to keep track of the various places a file might reside
  - We can mount filesystems together within a single namespace
- We must consider sharing, which leads to a need for protection
  - ACLs vs. capabilities
- All file operations must begin by locating the file
  - Path search is very expensive and may be cached
- Inodes separate logical and physical location but add to the cost of path search