University of Pittsburgh

Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - **All deadlines moved to Monday May 1st at 11:59 pm**
    - But please don't wait to last minute!
  - Homework 11, 12, Bonus Homework
  - Lab 4 and Lab 5
  - Quiz 3 and Quiz 4
  - Project 4 **(no late deadline)**
  - Post-Course Quiz (1 bonus point)

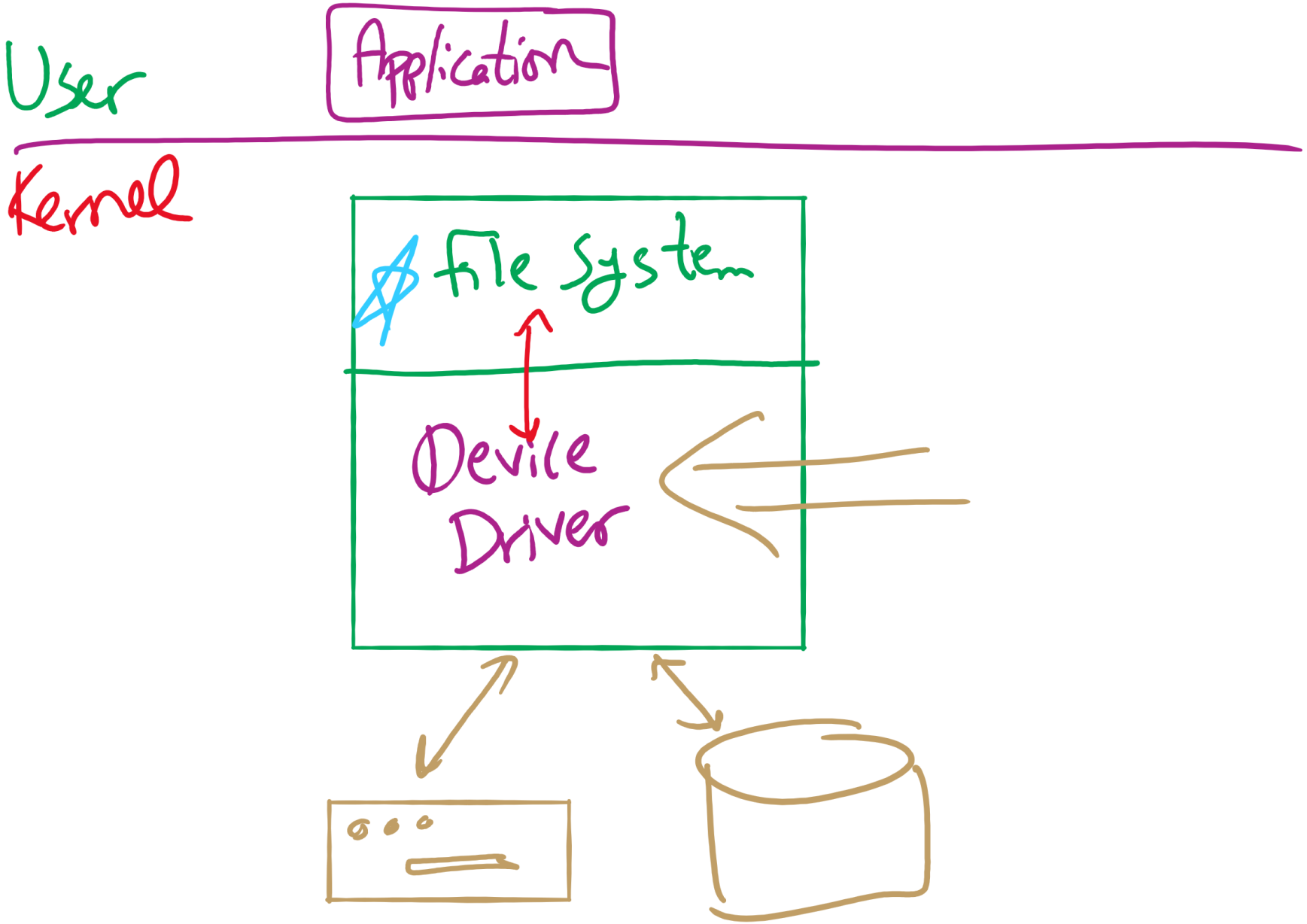
Previous lecture …

- How to allocate disk blocks to files and directories?
  - linked, FAT, indexed, and hybrid
  - file directories
  - free block tracking

This lecture …

- How does a file system handle errors?
- How does a file system hide disk access delays?

Software Layers

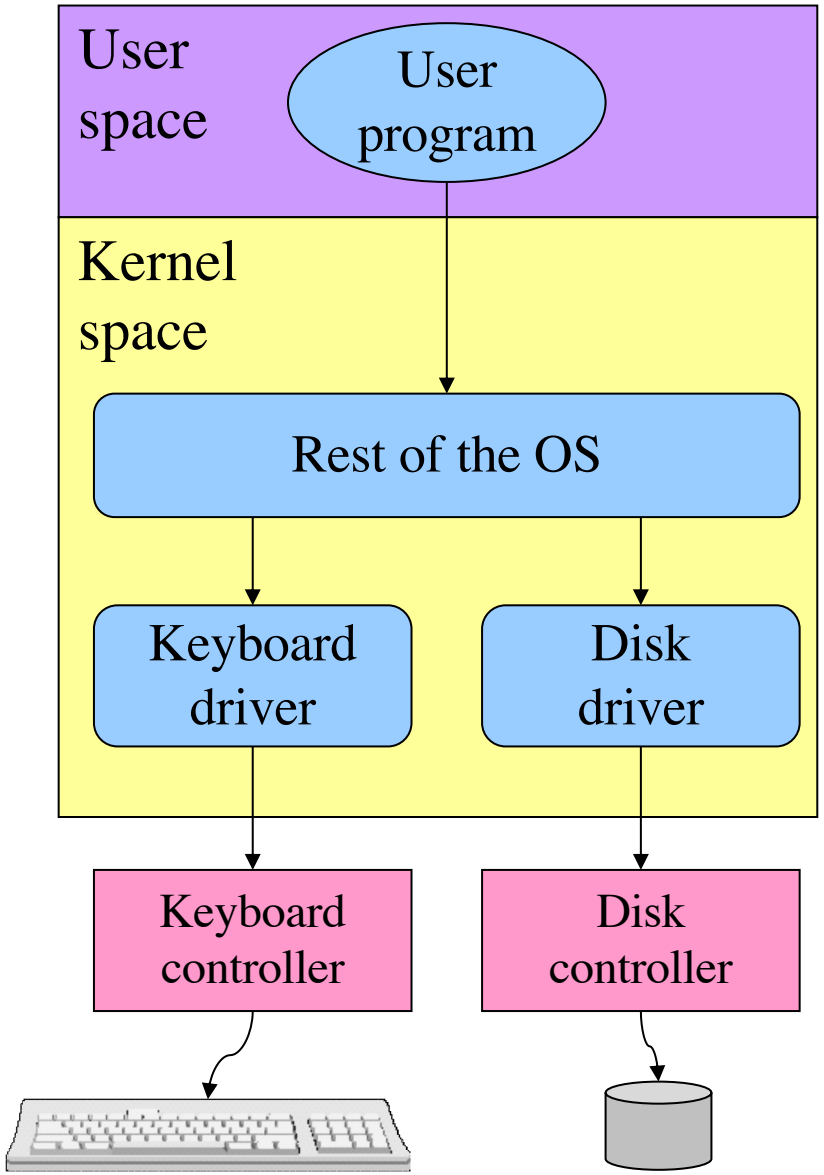


Question of the Day

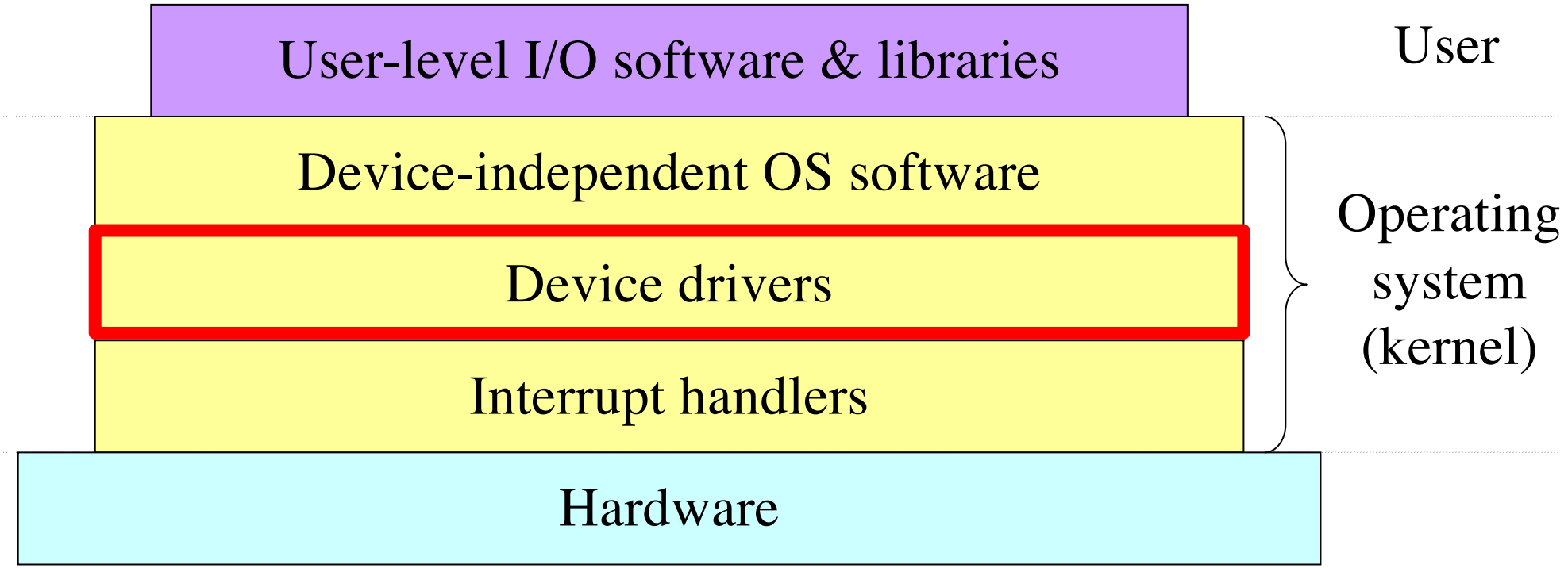
- How does a file system handle errors?
- Answer: Defense in Depth
  - multiple layers of error detection/correction

Device drivers

- Device drivers go between device controllers and rest of OS
  - Drivers **standardize interface** to widely varied devices
- Device drivers **communicate** with controllers over bus
  - Controllers communicate with devices themselves



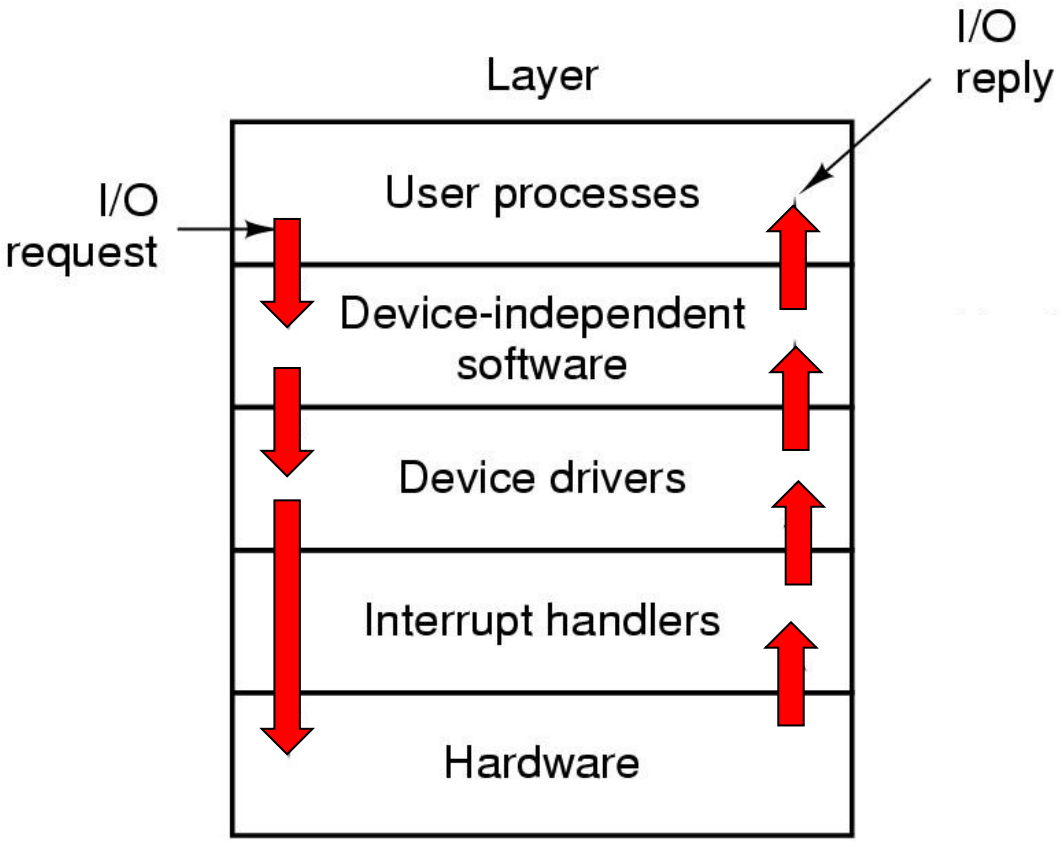
Layers of I/O software



Device Driver goals

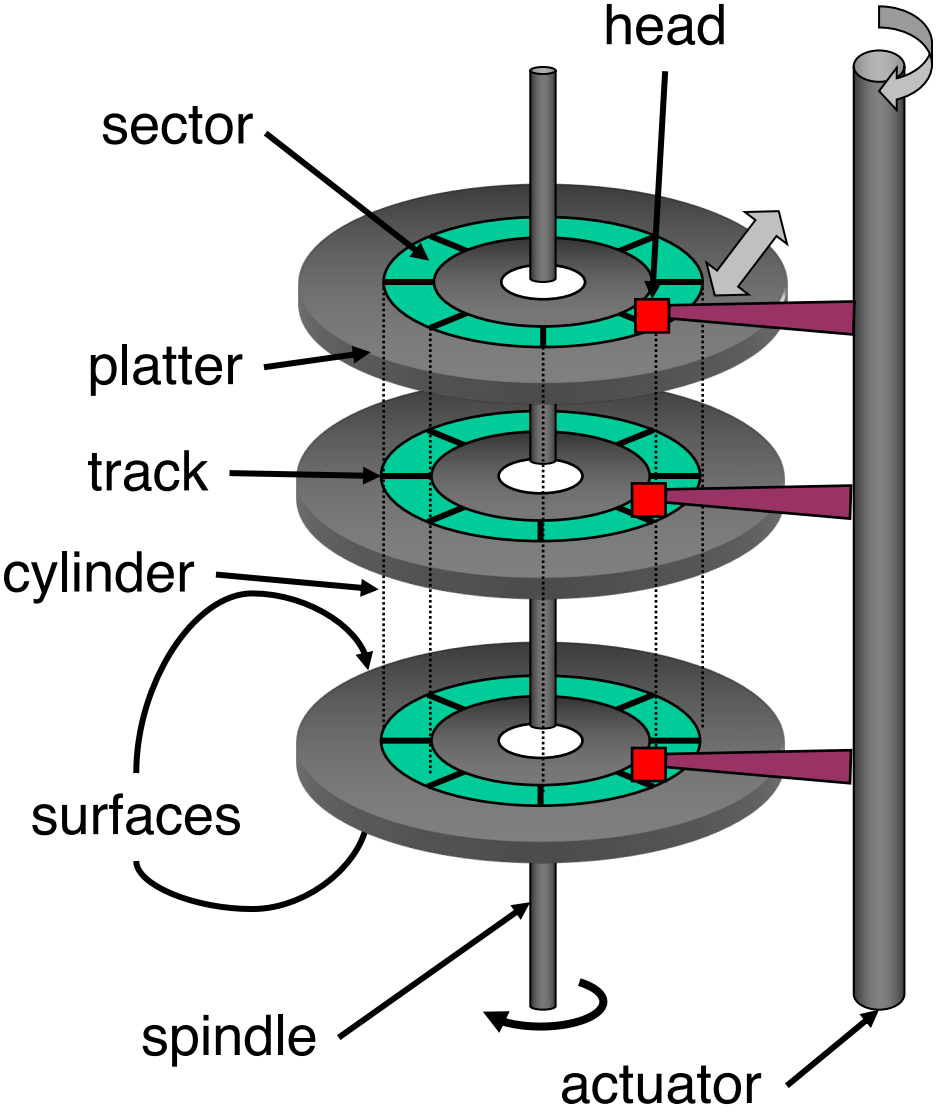
- Device independence
  - Programs can access any I/O device
  - No need to specify device in advance
- Uniform naming
  - Name of a file or device is a string or an integer
  - Doesn't depend on the machine (underlying hardware)
- **Error handling**
  - **Done as close to the hardware as possible**
  - **Isolate from higher-level software**
- Synchronous vs. asynchronous transfers
  - Blocked transfers vs. interrupt-driven
- Buffering
  - Data coming off a device cannot be stored in final destination right away
- Arbitration of device access
  - Sharable vs. dedicated devices

Anatomy of an I/O request

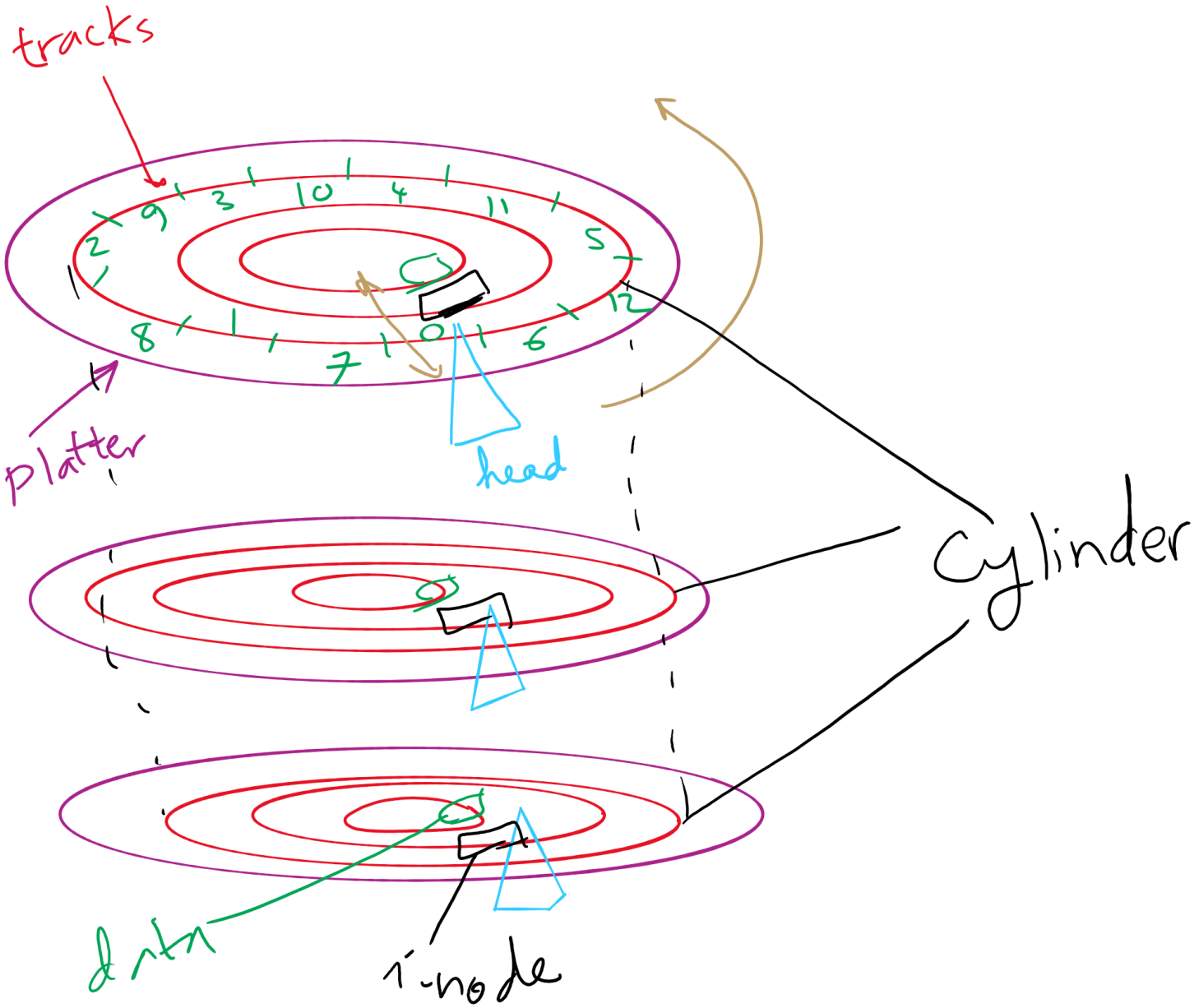


Disk drive structure

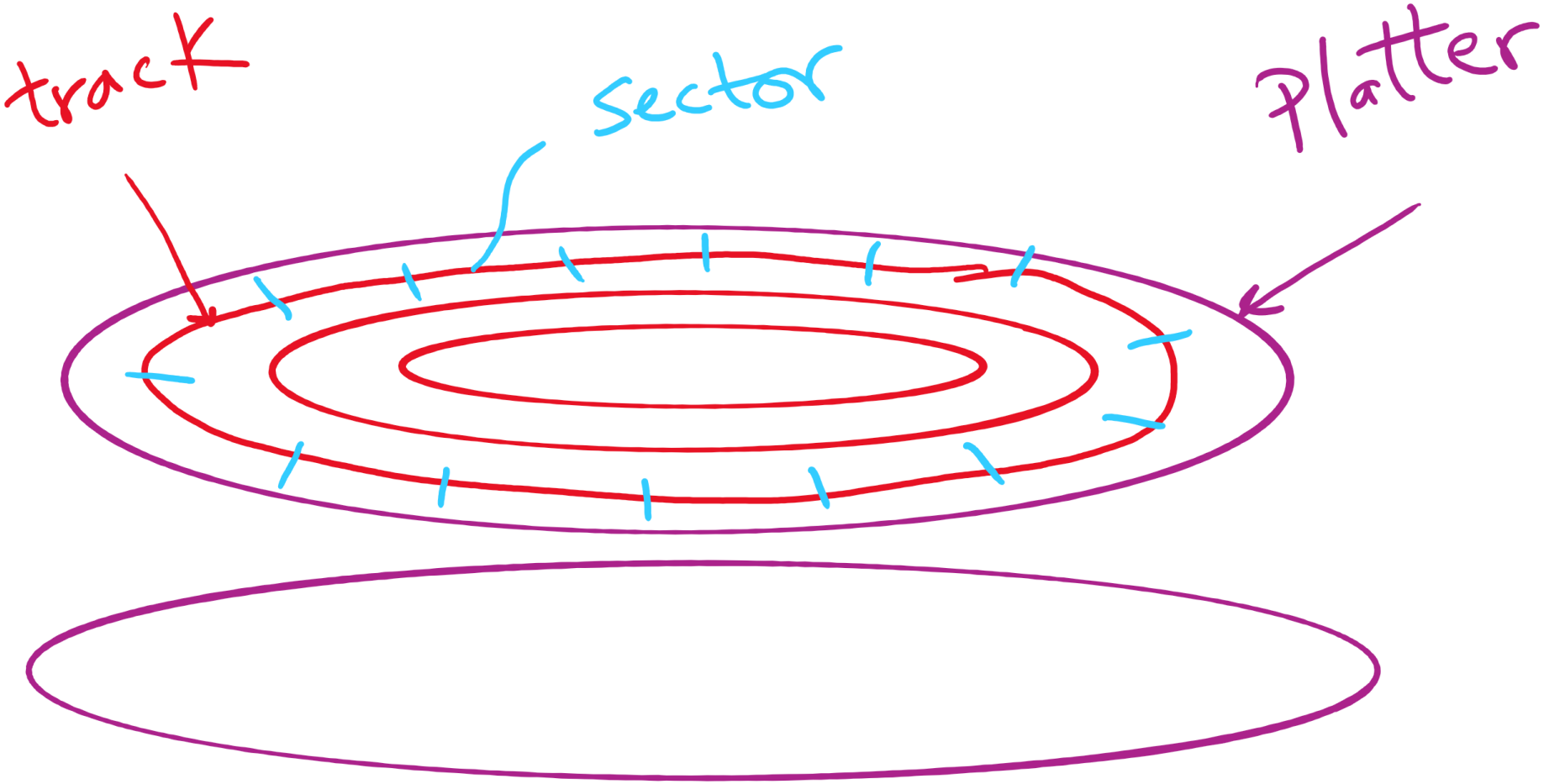
- Data stored on surfaces
  - One or more platters per disk
  - Up to two surfaces per platter
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison



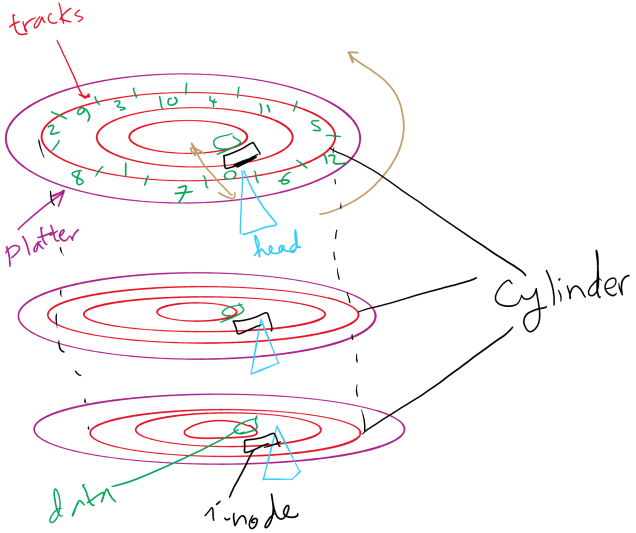
Disks, cylinders, cylinder groups



Disk Sector



Disk Size



$$\text{Sectors/disk} = \frac{\text{Sectors}}{\text{track}} * \frac{\text{tracks}}{\text{cylinder}} * \frac{\text{cylinders}}{\text{disk}}$$

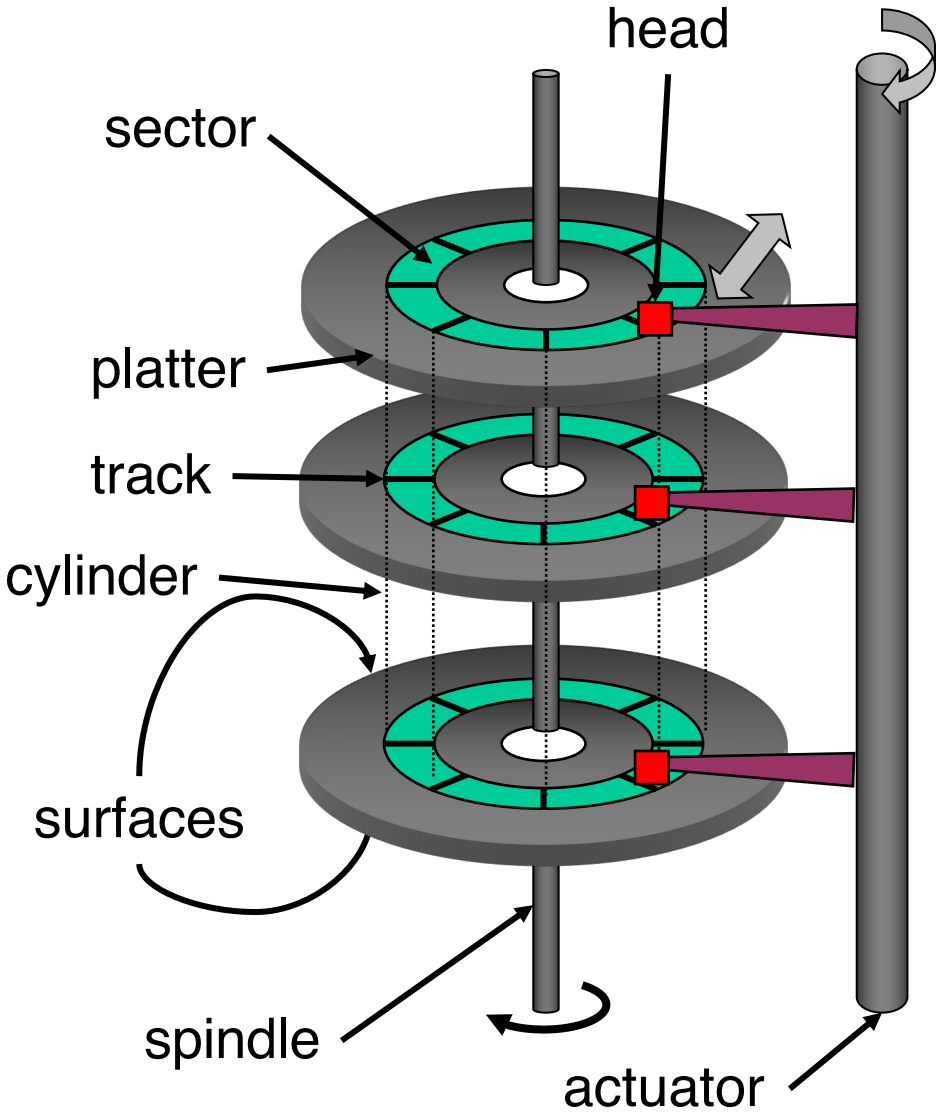
$$\text{disk capacity (bytes/disk)} = \frac{\text{bytes}}{\text{sector}} * \frac{\text{sectors}}{\text{disk}}$$

What's in a disk request?

- Time required to read or write a disk block determined by **3 factors**
  - **Seek** time: move disk arm to track
  - **Rotational** delay
    - Average delay = 1/2 rotation time
    - Example: one rotation in 10ms → average rotational delay = 5ms
  - Actual **transfer** time
    - Transfer time = time to rotate sector(s) under the heads
    - Example: one rotation in 10ms, 200 sectors/track
      - 10/200 ms = 0.05ms transfer time **per sector**
- Seek time dominates, with rotation time close

Intelligent Seek (IntelliSeek)

- Sometimes we don't need to move disk arm at max speed during seek time
- Adjust disk arm speed so that it reaches track right before needed sector is under the head

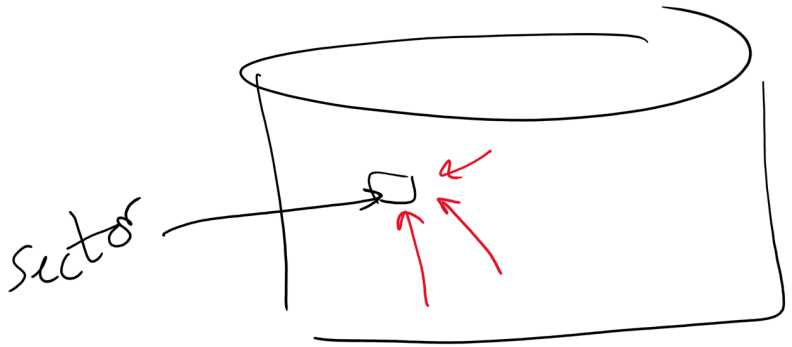


Disk drive specifics

	IBM 360KB floppy	WD 18GB HD
Cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (average)
Sectors per disk	720	35742000
Bytes per sector	512	512
Capacity	360 KB	18.3 GB
Seek time (minimum)	6 ms	0.8 ms
Seek time (average)	77 ms	6.9 ms
Rotation time	200 ms	8.33 ms
Spinup time	250 ms	20 sec
Sector transfer time	22 ms	17 μsec

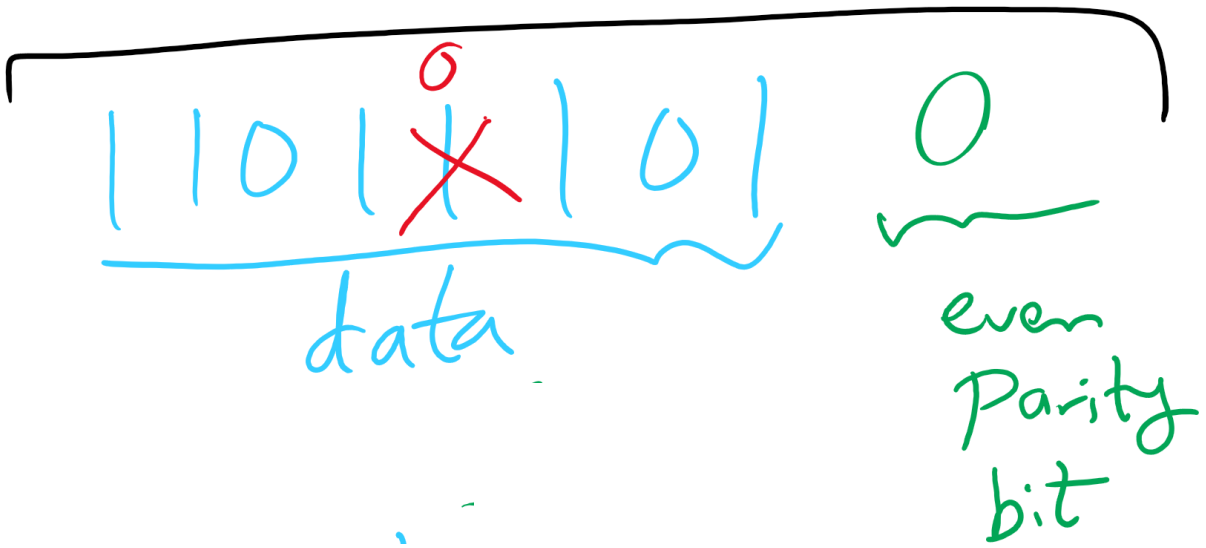
Structure of a disk sector

- Preamble contains information about the sector
  - Sector number & location information
- Data is usually 256, 512, or 1024 bytes
- ECC (**Error Correcting Code**) is used to detect & correct minor errors in the data

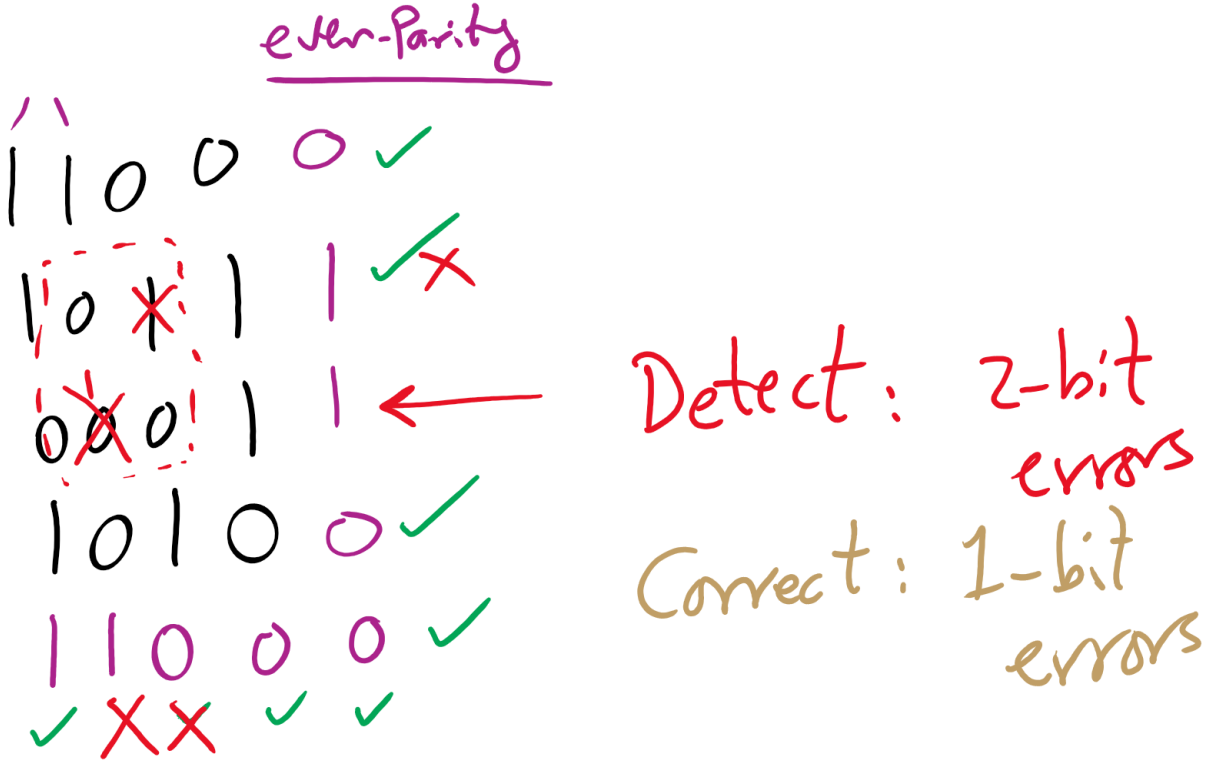


Parity Bit

- One parity bit can detect single-bit errors

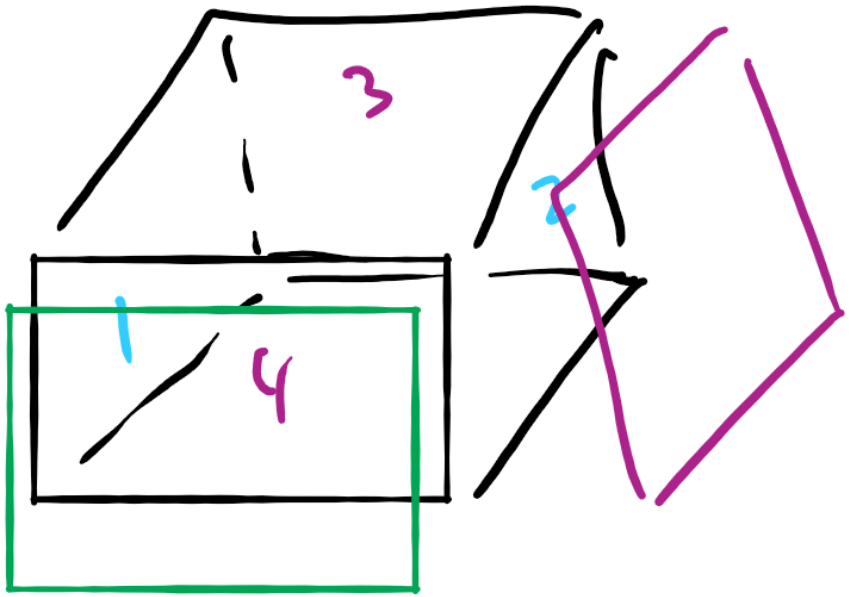


Two-Dimensional Parity Bits



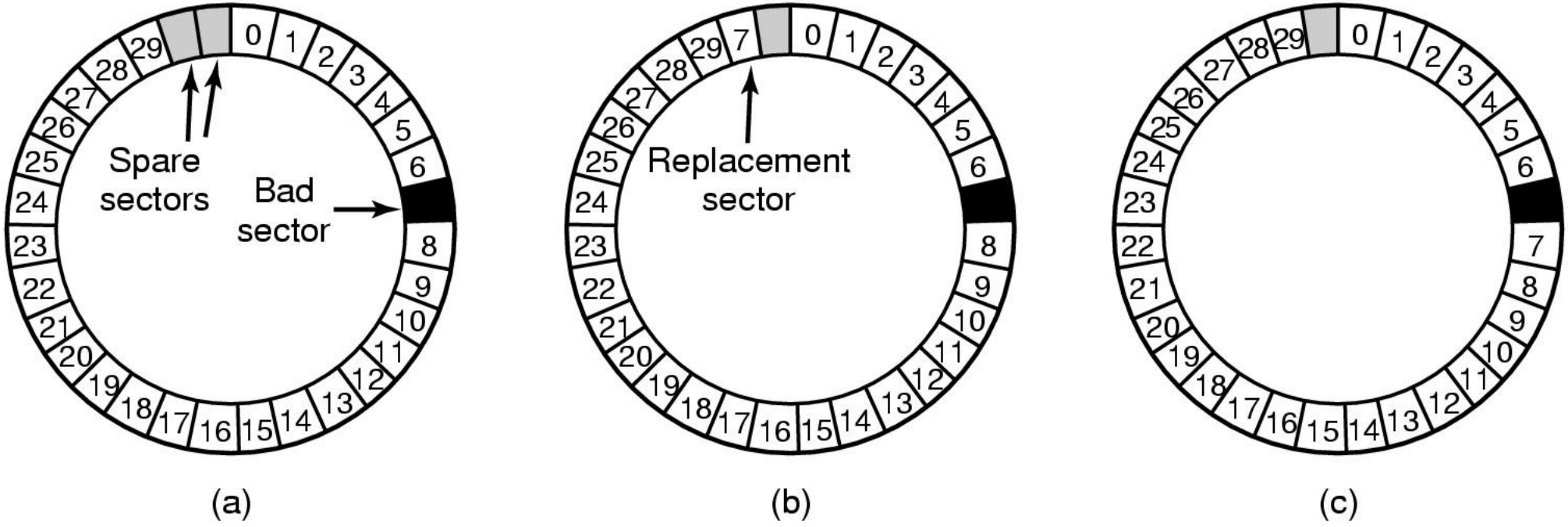
Three-Dimensional Parity Bits

- Detect up to 3-bit errors
- Correct up to 2-bit errors

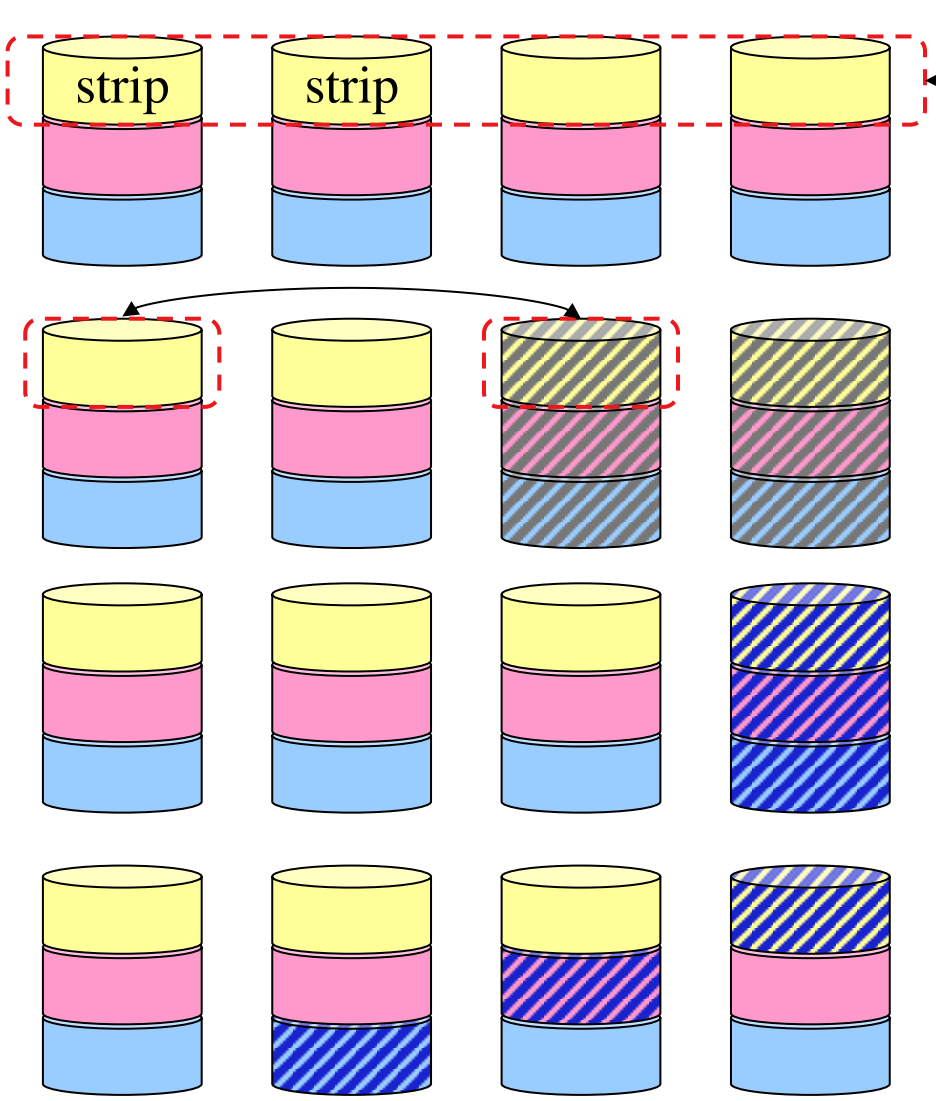


When good disks go bad…

- Disks have **defects**
  - In 3M+ sectors, this isn't surprising!
- ECC helps with errors, but sometimes this isn't enough
- Disks keep **spare sectors** (normally unused) and remap bad sectors into these spares
  - If there's time, the whole track could be **reordered**…



What if an entire disk goes bad?



Stripe

RAID 0
(Redundant Array of Inexpensive Disks

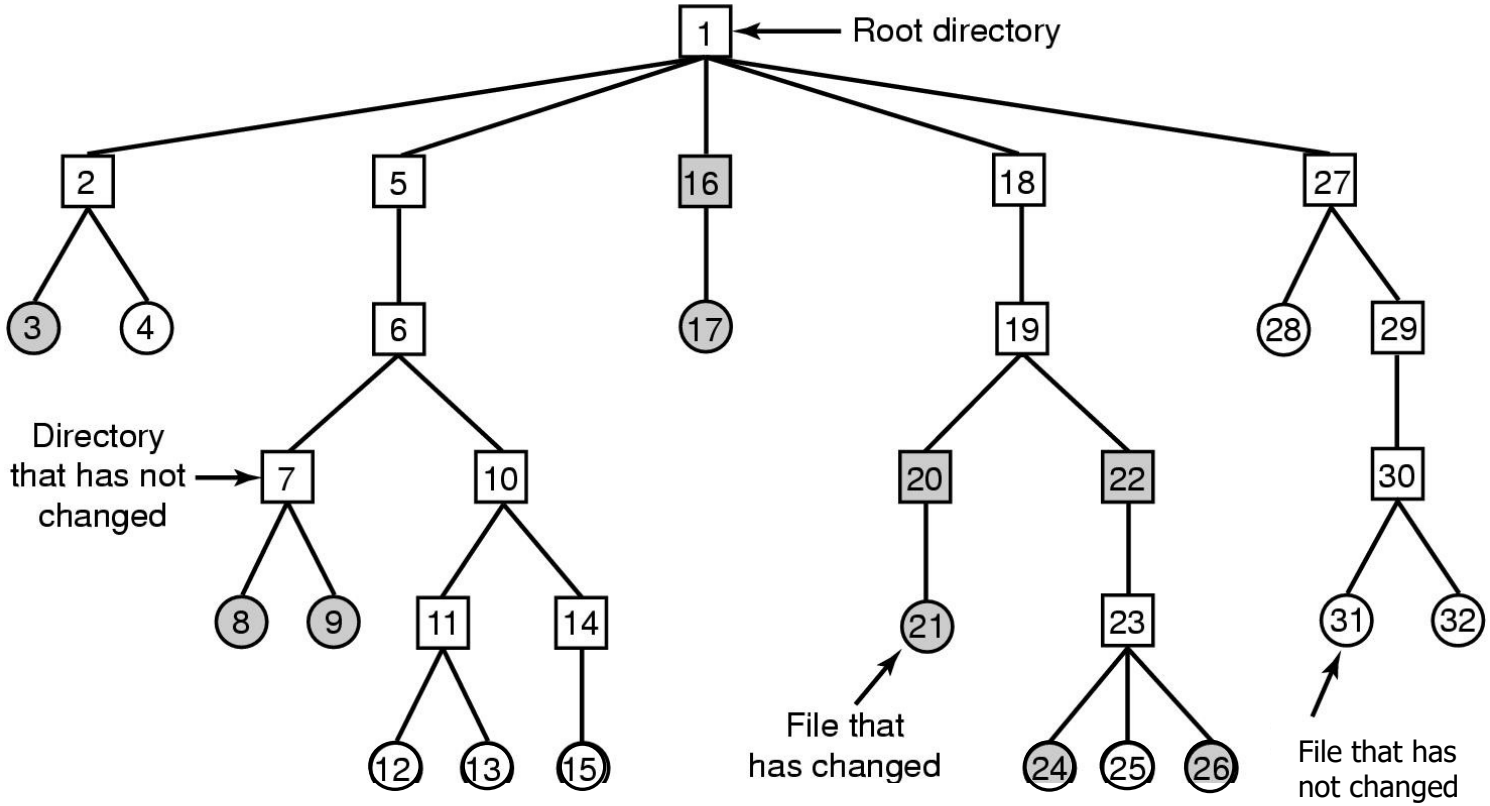
RAID 1
(Mirrored copies)

RAID 4
(Striped with parity)

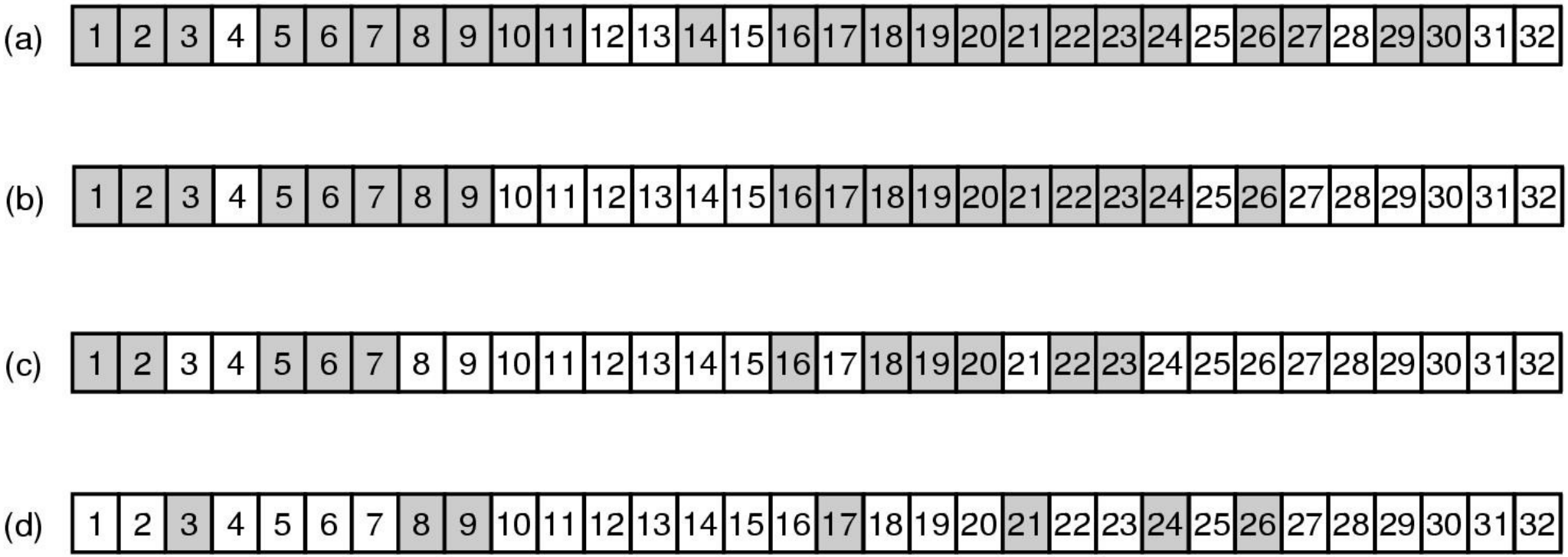
RAID 5
(Parity rotates through disks)

What if RAID cannot mask an error?

- **Solution**: Backing up a file system (aka **dumping)**
- **Expensive** operation → done **incrementally**
- Track items modified since last dump (shaded)
  - Numbers are i-node numbers



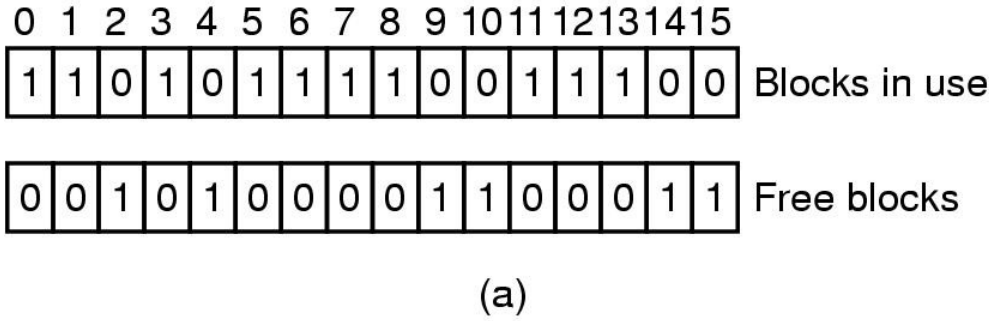
Incremental dump using bitmaps



What if errors happen in metadata?

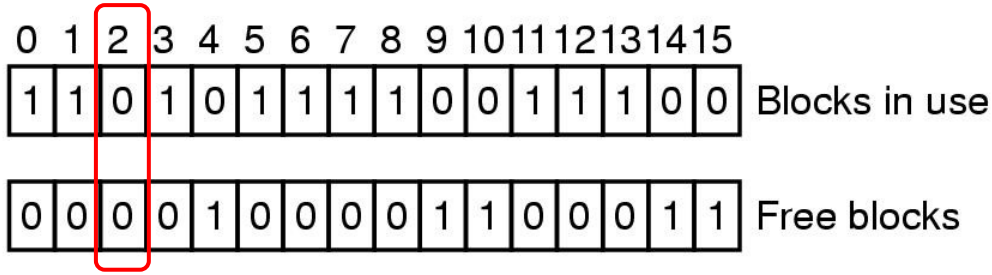
- Reason: power failure during an operation
- file system consistency check

Consistent

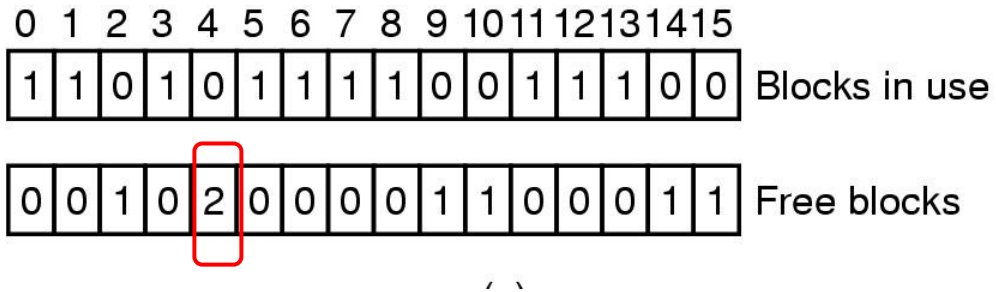


(a)

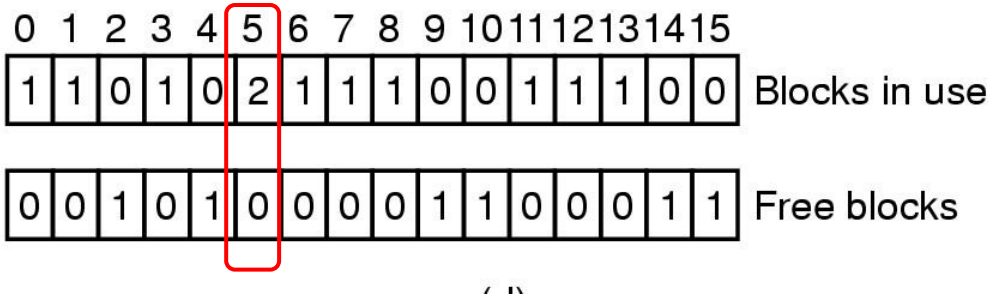
Missing ("lost") block



(b)



Duplicate block in free list

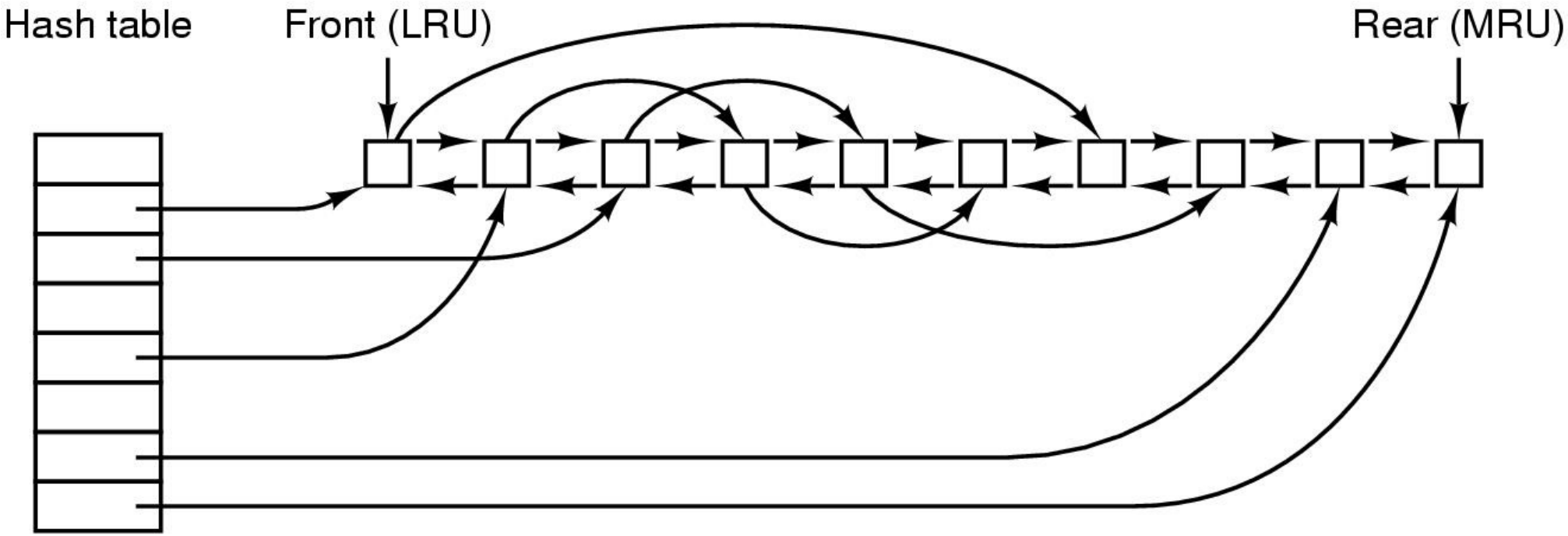


Duplicate block in two files

Problem of the Day – Part 2

- How does a file system hide disk access delays?
- Answer:
  - Caching
  - Optimize the writes

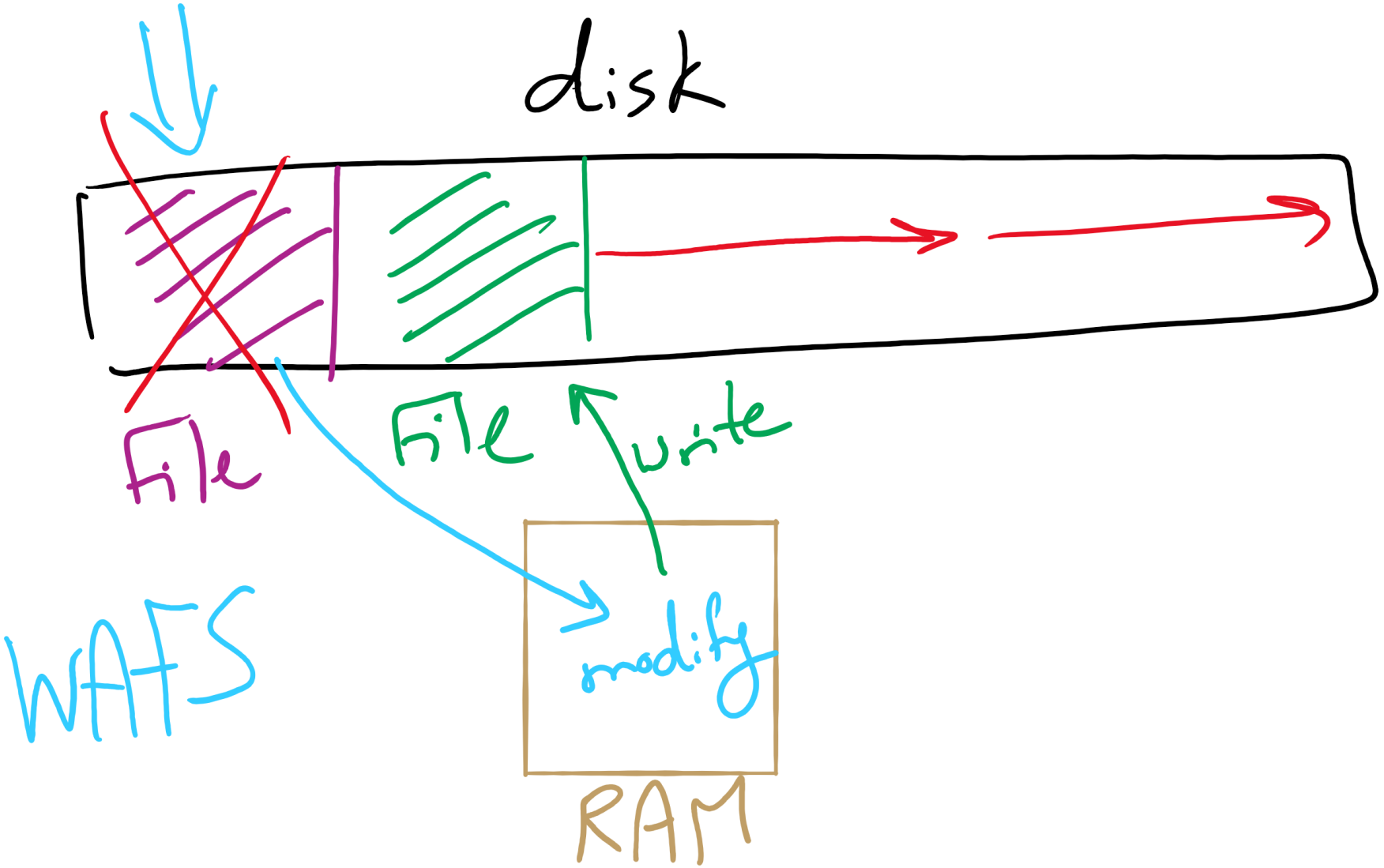
File block cache data structures



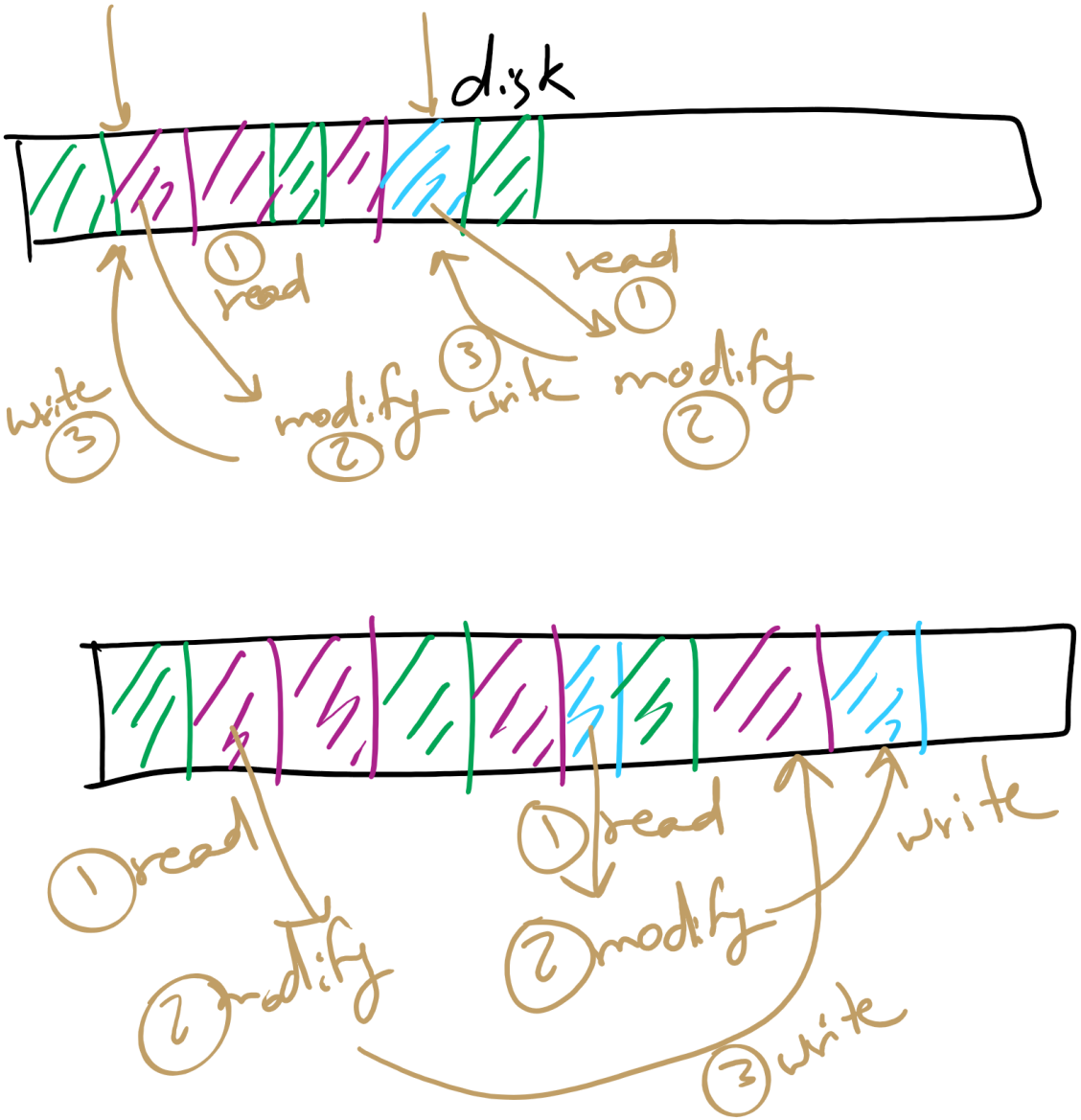
Log-structured file systems

- Trends in disk & memory
  - Faster CPUs
  - Larger memories
- Result
  - More memory -> disk caches can also be larger
  - Increasing number of read requests can come from cache
  - Thus, most disk accesses will be writes
- LFS structures entire disk as a log
  - All writes initially buffered in memory
  - Periodically write these to the end of the disk log
  - When file opened, locate i-node, then find blocks
- Issue: what happens when blocks are deleted?

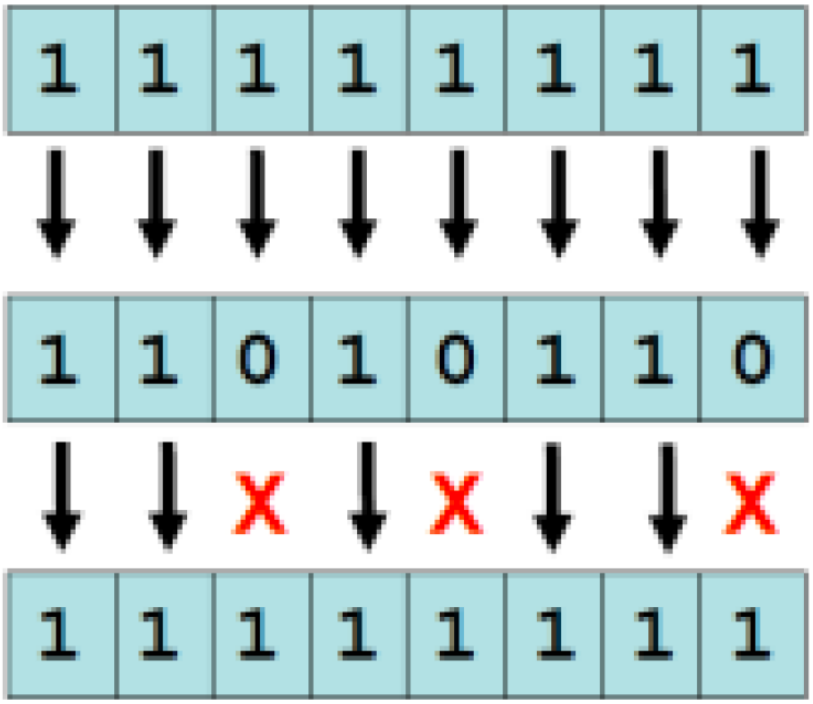
Log Structured File System



Log Structured File System



Flash File System



Start off with all 1's

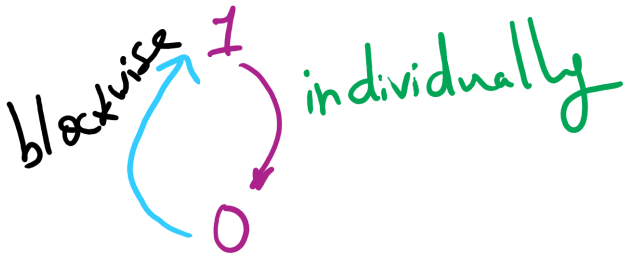
Programming from 1's to 0's is allowed

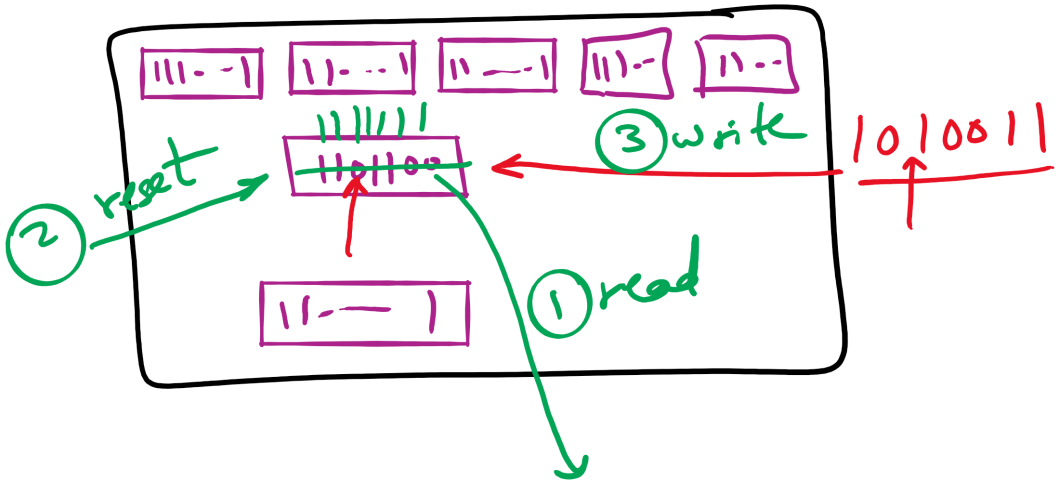
Programming from 0's to 1's is not allowed

Wear Leveling

Count total writes per flash sector and attempt to balance across the whole disk

Wear leveling for SSDs





1 → 0 : individually

0 → 1 : read block → reset block
→ write block

Wear leveling

read block → write to a new block