

# Introduction to File Systems

COMP 321

Dave Johnson



COMP 321

Copyright © 2025 David B. Johnson

Page 1

1

## Hard Disks, Big(ger) and Small(er)



5.25" "full-height" vs. a  
more modern 2.5" hard disk

Photo: Wikimedia Commons and Ian Wilson

COMP 321

Copyright © 2025 David B. Johnson

Page 2

2

## Really Old Hard Disks



14" disk drives with removable disk packs (sort of like a floppy disk, but way before that time)

The fixed electronics was the most expensive part

Photo: Wikimedia Commons and Deutsche Fotothek

COMP 321

Copyright © 2025 David B. Johnson

Page 3

3

## Hard Disk Internals



A modern 3.5" drive

... but they're all basically the same inside

Photo: Amazon.com and Western Digital (WD Blue WD20EZRX 2 TB 3.5" SATA Hard Drive)

COMP 321

Copyright © 2025 David B. Johnson

Page 4

4

## The Same Basic Thing in Many Sizes

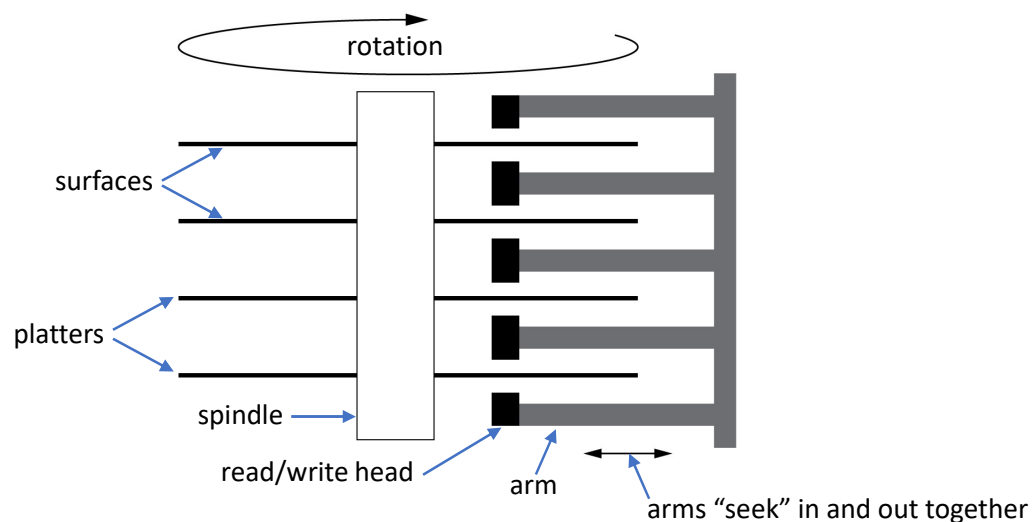


Photos: Wikimedia Commons

IBM and Hitachi “Microdrive” — 1 inch in diameter, but still the same inside

5

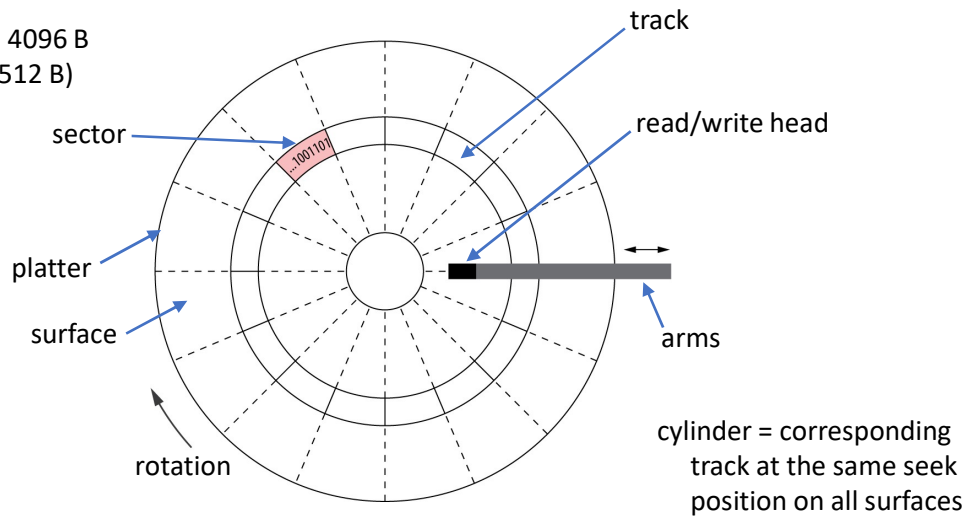
## Hard Disk (Side View)



6

## Hard Disk (Top View)

sector size = 4096 B  
(older disks 512 B)



COMP 321

Copyright © 2025 David B. Johnson

Page 7

7

## Reading or Writing the Disk

***Hardware can read or write only in granularity of whole sectors***

### ***Steps in reading or writing the disk***

- Disk just keeps spinning in same direction at the same speed continuously
  - (Except it can be stopped spinning to save power, but it's very slow to stop and restart spinning and become stable again)
- The disk must seek to the correct cylinder (move set of arms in or out)
- Start paying attention to correct head and thus correct surface (zero time)
- Wait for the beginning of the sector to rotate up to the head
- Transfer the bits of the sector, one by one, off of/onto the surface

COMP 321

Copyright © 2025 David B. Johnson

Page 8

8

## Performance of Reading or Writing the Disk

***The disk must seek to the correct cylinder (move the set of arms in or out)***

- Requires time for arms to start moving and to stop moving
- Requires time to heads based on distance (number of cylinders)

***Wait for the beginning of the sector to rotate up to the head***

- Depends only on the rotational speed (RPM) of the disk
- On **average**, this takes **one half** of time for a single revolution

***Transfer the bits of the sector, one by one, off of/onto the surface***

- Also depends on the number of sectors per track (i.e., one full rotation time divided by the number of sectors per track)

## Some Example Hard Disk Characteristics

|                        | <u>Old</u> | <u>Common</u> | <u>Fast</u> |
|------------------------|------------|---------------|-------------|
| Average seek time      | 30 ms      | 9 ms          | 4 ms        |
| Rotational speed       | 3600 RPM   | 5400 RPM      | 10,000 RPM  |
| Full revolution time   | 16.66 ms   | 11.11 ms      | 6 ms        |
| Average rotation delay | 8.33 ms    | 5.55 ms       | 3 ms        |

## Solid State Drives (SSDs)

*No moving parts (not mechanical, all electronic)*

Two example 1 TB SSDs:



Photos: Amazon.com and Samsung (860 EVO 2.5" SATA interface and 970 EVO M.2 NVMe interface)

## Solid State Drives (SSDs)

- Many attach to the computer with same interface as a “traditional” disk drive
- Based on “flash memory” non-volatile storage
- Very different performance characteristics than with traditional hard disks
- Internally, storage is still organized in sectors (usually there called “pages”)
- Pages are grouped into “blocks” of, e.g., 32 to 128 pages
- Reading is easy, but writing requires the entire block to be erased first (slow)
- Blocks can only be erased a limited (but large) number of times
- Internal drive algorithms for block wear leveling, page remapping

***Now common, but file systems still generally designed around traditional disks***

## Types of File Access Patterns

### ***Sequential access to data within a file***

- System remembers an open file's current position (offset) within the file
- Read or write by  $N$  bytes advances file's current position by  $N$  bytes
- "Rewind" operation sets file's current position to the beginning of file

### ***Random (or direct) access to data within a file***

- Every individual read or write can specify the file position to use
- Or can explicitly set (e.g., "`lseek()`") to a new position in the file anytime
  - Relative to the beginning of the file
  - Relative to the open file's current position in the file
  - Relative to the end of the file

## On-Disk File System Data Structures

***The file system certainly keeps some data in memory as it runs***

***But hard disks (and thus file systems) can be very, very large***

- A hard disk can store orders of magnitude more data than physical memory
- And there can be many (many!) individual files on the disk
- Impossible in general to store everything in memory

***And data structures are also needed on-disk so they are permanent***

- Don't want to lose all data on a crash or power failure
- Also don't want to lose even some kinds of data (such as keeping the data of all files on disk but losing the **names** of the files on a crash or power failure)

## Guidelines for On-Disk File System Data Structures

- On-disk file system data structures are “permanent” (good and bad)
  - With memory, crash and reboot gives you fresh new data structures
  - But data structures on disk are still there after a crash and reboot
- A tradeoff between simplicity and performance
  - Want good performance, but be careful not to mess things up
- Put “related” things “near” each other on disk
  - Classical disk scheduling no longer fully works, but it still helps
- ***Can’t use memory pointers (memory addresses) on disk***
  - Pointers recorded on disk no longer mean the same thing when read back
  - Must link things together on disk some other way

## File System “Blocks” vs. Hardware Disk Sectors

- Disk sectors are relatively small (512 bytes or 4096 bytes)
- File systems generally treat multiple sectors together as a file system “block”
- Power of 2 multiple, e.g., 2 consecutive sectors, 8 consecutive sectors, ...
- ***Advantages***
  - Lowers bookkeeping overhead: keeping track of individual free blocks, keeping track of the blocks that belong to each file
  - Better performance, since keeps sectors of a file’s data more “together”
  - Larger total file size possible for a given number of bits representing block numbers (e.g., 16-bit block number means only 65,535 **blocks** can be used)
- ***Disadvantage***
  - Slight increase in internal fragmentation within each file

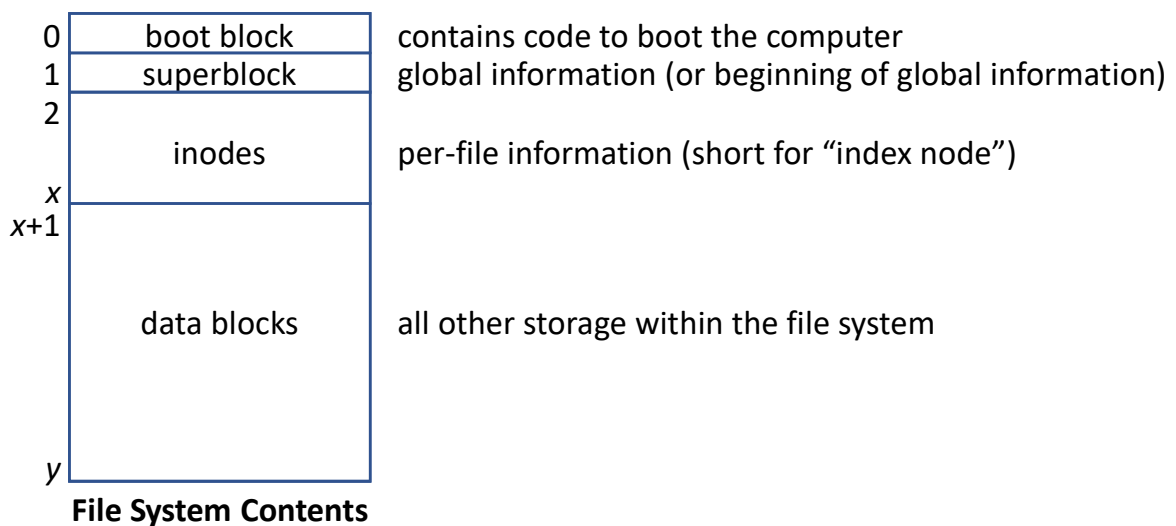
## Overall Types of On-Disk Data Structures

### *Two general types of on-disk file system data structures*

- **Per-file** information
  - Information about what blocks store the data of that file
  - The file's owner and protection information
  - Information such as the creation time and last modified time
  - The name of the file?
- **Global** information
  - Total size of the file system (number of blocks)
  - Information necessary for finding per-file information for a given file
  - The list of free blocks

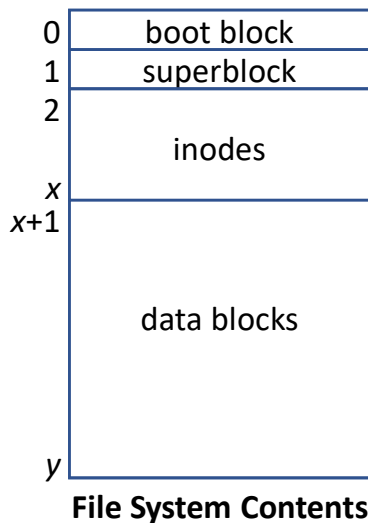
17

## Simple Example: Classical Unix Format



18

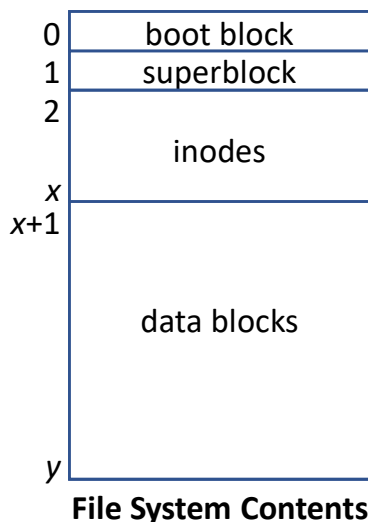
## Consequences of this Format



### Advantages

- Can identify any block by its block number
- Block 0 is not really part of the file system, so 0 can be used in the file system as a special value (e.g., like NULL)
- Can identify any inode by its inode number
  - Example with 4096 blocksize, 256 inode size
  - Exactly 16 inodes per block of inodes
  - Easy to calculate where any inode is, given its inode number

## Consequences of this Format



### Disadvantages

- All inodes must be preallocated when the file system is “formatted”
  - Must decide in advance how many inodes
  - Wastes space if too many are preallocated
  - Or can’t make a new file if not enough inodes are preallocated, even if free data blocks are available
- The inode for a file can be a long seek away from the data blocks of the file’s contents
- ***These disadvantages can all be fixed, but don’t worry about them now***