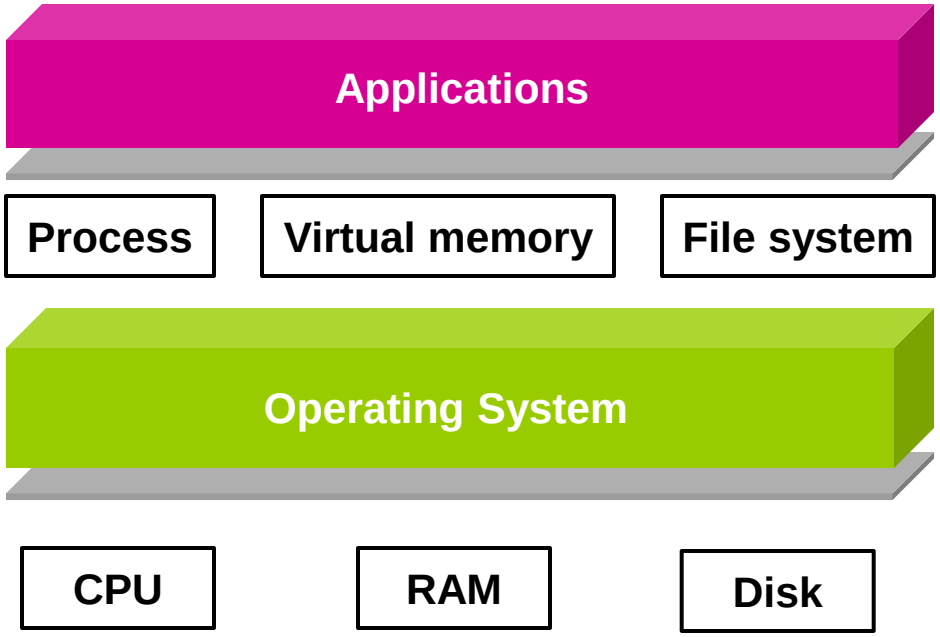
# CS 153
# Design of Operating Systems

## Winter 2016

Lecture 19: File Systems

# OS Abstractions



# File Systems

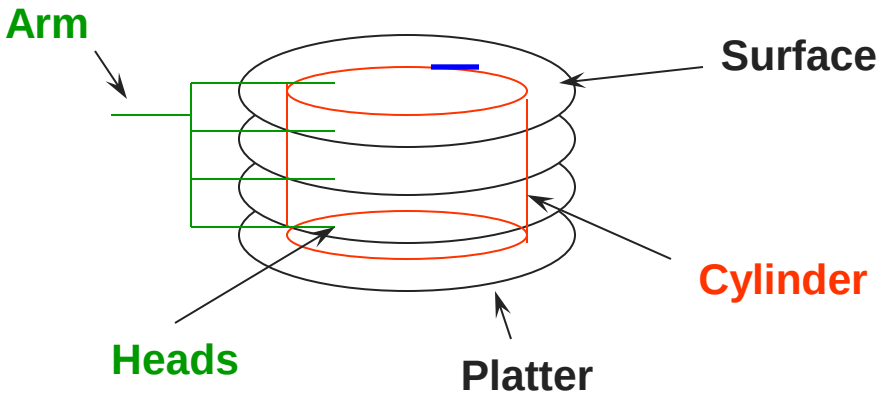
- First we'll discuss properties of physical disks
  - Structure
  - Performance
  - Scheduling
- Then we'll discuss how we build file systems on them
  - Files
  - Directories
  - Sharing
  - Protection
  - File System Layouts
  - File Buffer Cache
  - Read Ahead

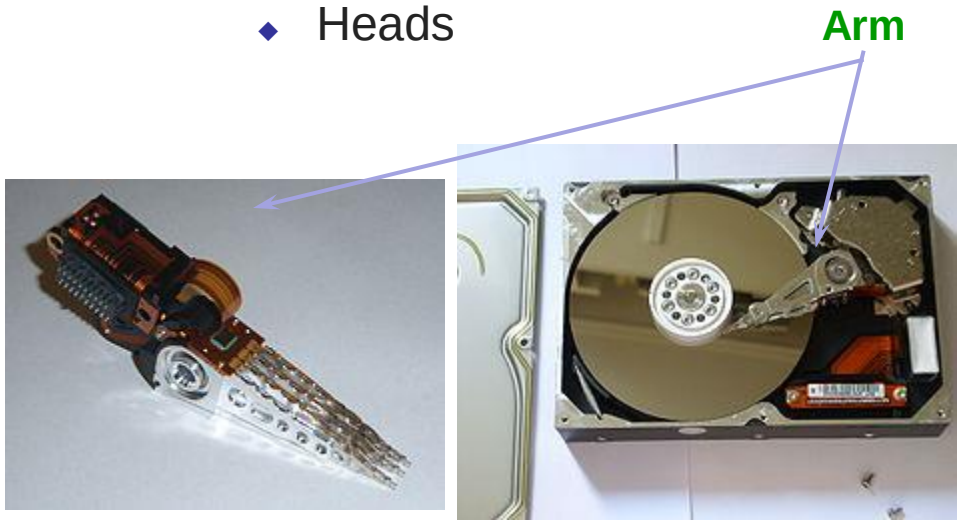
# Disks and the OS

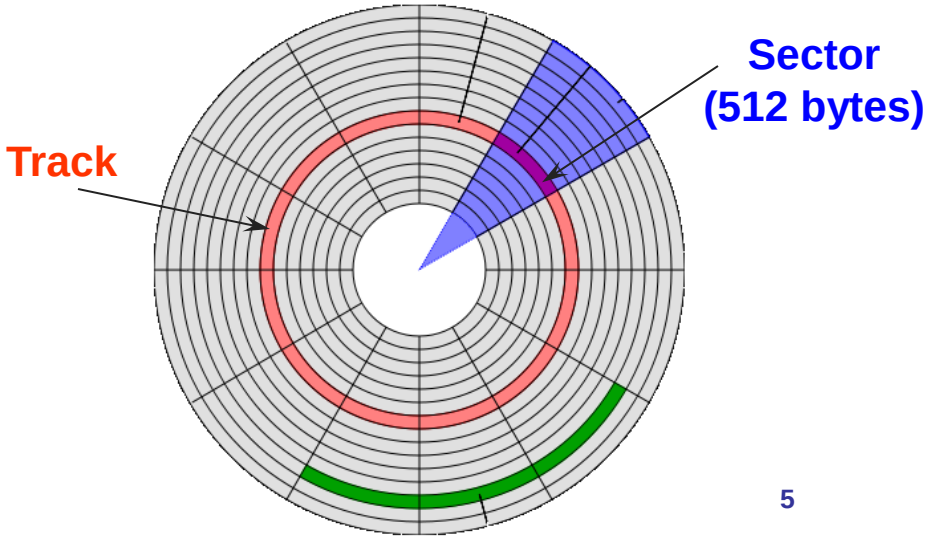
- Disks are messy physical devices:
  - Errors, bad blocks, missed seeks, etc.
- OS's job is to hide this mess from higher level software
  - Low-level device control (initiate a disk read, etc.)
  - Higher-level abstractions (files, databases, etc.)

# Physical Disk Structure

- Disk components
  - Platters
  - Surfaces
  - Tracks
  - Sectors
  - Cylinders
  - Arm
  - Heads



# Disk Interaction

- Specifying disk requests requires a lot of info:
  - Cylinder #, surface #, track #, sector #, transfer size…
- Older disks required the OS to specify all of this
  - The OS needed to know all disk parameters
- Modern disks are more complicated
  - Not all sectors are the same size, sectors are remapped, etc.
- Current disks provide a higher-level interface (SCSI)
  - The disk exports its data as a logical array of blocks [0…N]
    - » Disk maps logical blocks to cylinder/surface/track/sector
  - Only need to specify the logical block # to read/write
  - But now the disk parameters are hidden from the OS

# Disks Heterogeneity

- Seagate Barracuda 3.5" (workstation)
  - capacity: 250 - 750 GB
  - rotational speed: 7,200 RPM
  - sequential read performance: 78 MB/s (outer) - 44 MB/s (inner)
  - seek time (average): 8.1 ms
- Seagate Cheetah 3.5" (server)
  - capacity: 73 - 300 GB
  - rotational speed: 15,000 RPM
  - sequential read performance: 135 MB/s (outer) - 82 MB/s (inner)
  - seek time (average): 3.8 ms
- Seagate Savvio 2.5" (smaller form factor)
  - capacity: 73 GB
  - rotational speed: 10,000 RPM
  - sequential read performance: 62 MB/s (outer) - 42 MB/s (inner)
  - seek time (average): 4.3 ms

# Disk Performance

- What does disk performance depend upon?
  - Seek – moving the disk arm to the correct cylinder
    - » Depends on how fast disk arm can move (increasing very slowly)
  - Rotation – waiting for the sector to rotate under the head
    - » Depends on rotation rate of disk (increasing, but slowly)
  - Transfer – transferring data from surface into disk controller electronics, sending it back to the host
    - » Depends on density (increasing quickly)
- When the OS uses the disk, it tries to minimize the cost of all of these steps
  - Particularly seeks and rotation

# Disk Scheduling

- Because seeks are so expensive (milliseconds!), OS schedules requests that are queued waiting for the disk
  - FCFS (do nothing)
    - » Reasonable when load is low
    - » Does nothing to minimize overhead of seeks
  - SSTF (shortest seek time first)
    - » Minimize arm movement (seek time), maximize request rate
    - » Favors middle blocks, potential starvation of blocks at ends
  - SCAN (elevator)
    - » Service requests in one direction until done, then reverse
    - » Long waiting times for blocks at ends
  - C-SCAN
    - » Like SCAN, but only go in one direction (typewriter)

# Disk Scheduling (2)

- In general, unless there are request queues, disk scheduling does not have much impact
  - Important for servers, less so for PCs
- Modern disks often do the disk scheduling themselves
  - Disks know their layout better than OS, can optimize better
  - Ignores, undoes any scheduling done by OS

# File Systems

- File systems
  - Implement an abstraction (files) for secondary storage
  - Organize files logically (directories)
  - Permit sharing of data between processes, people, and machines
  - Protect data from unwanted access (security)

# Files

- A file is a sequence of bytes with some properties
  - Owner, last read/write time, protection, etc.
- A file can also have a type
  - Understood by the file system
    - » Block, character, device, portal, link, etc.
  - Understood by other parts of the OS or runtime libraries
    - » Executable, dll, souce, object, text, etc.
- A file's type can be encoded in its name or contents
  - Windows encodes type in name
    - » .com, .exe, .bat, .dll, .jpg, etc.
  - Unix encodes type in contents
    - » Magic numbers, initial characters (e.g., #! for shell scripts)

# Basic File Operations

Unix

- creat(name)
- open(name, how)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)

NT

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- CopyFile(name)
- MoveFile(name)

# File Access Methods

- Different file systems differ in the manner that data in a file can be accessed
  - Sequential access – read bytes one at a time, in order
  - Direct access – random access given block/byte number
  - Record access – file is array of fixed- or variable-length records, read/written sequentially or randomly by record #
  - Indexed access – file system contains an index to a particular field of each record in a file, reads specify a value for that field and the system finds the record via the index (DBs)
- Older systems provide more complicated methods
- What file access method do Unix, Windows provide?

# Directories

- Directories serve two purposes
  - For users, they provide a structured way to organize files
  - For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, …)
- Most file systems support the notion of a current directory
  - Relative names specified with respect to current directory
  - Absolute names start from the root of directory tree

# Directory Internals

- A directory is a list of entries
  - <name, location>
  - Name is just the name of the file or directory
  - Location depends upon how file is represented on disk
- List is usually unordered (effectively random)
  - Entries usually sorted by program that reads directory
- Directories typically stored in files
  - Only need to manage one kind of secondary storage unit

# Basic Directory Operations

Unix

- Directories implemented in files
  - Use file ops to create dirs
- C runtime library provides a higher-level abstraction for reading directories
  - opendir(name)
  - readdir(DIR)
  - seekdir(DIR)
  - closedir(DIR)

Windows

- Explicit dir operations
  - CreateDirectory(name)
  - RemoveDirectory(name)
- Very different method for reading directory entries
  - FindFirstFile(pattern)
  - FindNextFile()

# Path Name Translation

- Let's say you want to open "/one/two/three"
- What does the file system do?
  - Open directory "/" (well known, can always find)
  - Search for the entry "one", get location of "one" (in dir entry)
  - Open directory "one", search for "two", get location of "two"
  - Open directory "two", search for "three", get location of "three"
  - Open file "three"
- Systems spend a lot of time walking directory paths
  - This is why open is separate from read/write
  - OS will cache prefix lookups for performance
    - » /a/b, /a/bb, /a/bbb, etc., all share "/a" prefix

# File Sharing

- File sharing is important for getting work done
  - Basis for communication between processes and users

- 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

# Summary

- Files
  - Operations, access methods
- Directories
  - Operations, using directories to do path searches
- Sharing

# Next time…

- File system optimizations