Introduction to Operating Systems

CPSC/ECE 3220 Summer 2018

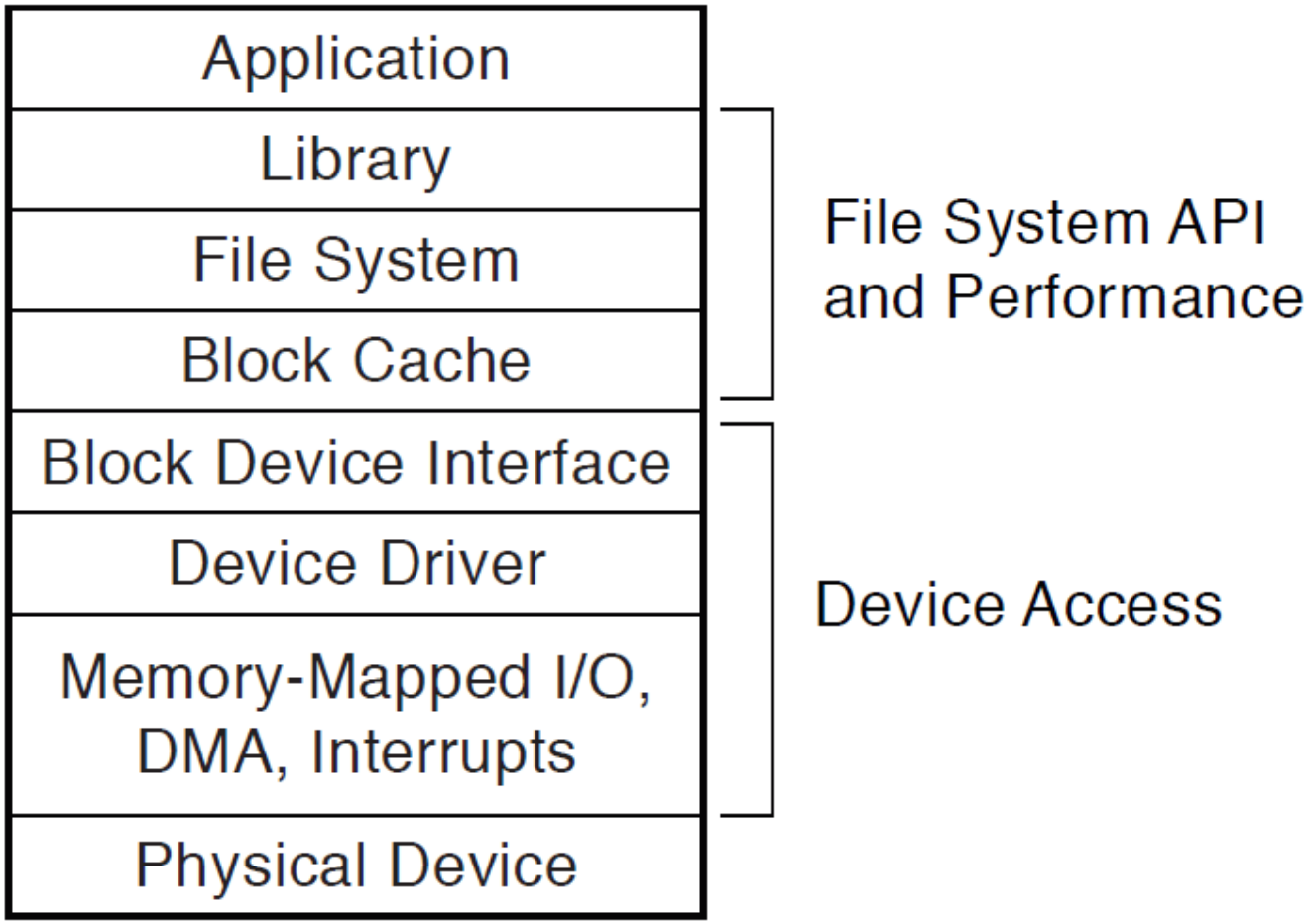
Lecture Notes
OSPP Chapters 11 and 12

(adapted by Mark Smotherman from Tom Anderson's slides on OSPP web site)

Main Points

- File systems
  - Useful abstractions on top of physical devices
- Files
  - Metadata, data, file types, open/close
- Storage hardware characteristics
  - Disks and flash memory
- File system usage patterns
- Naming

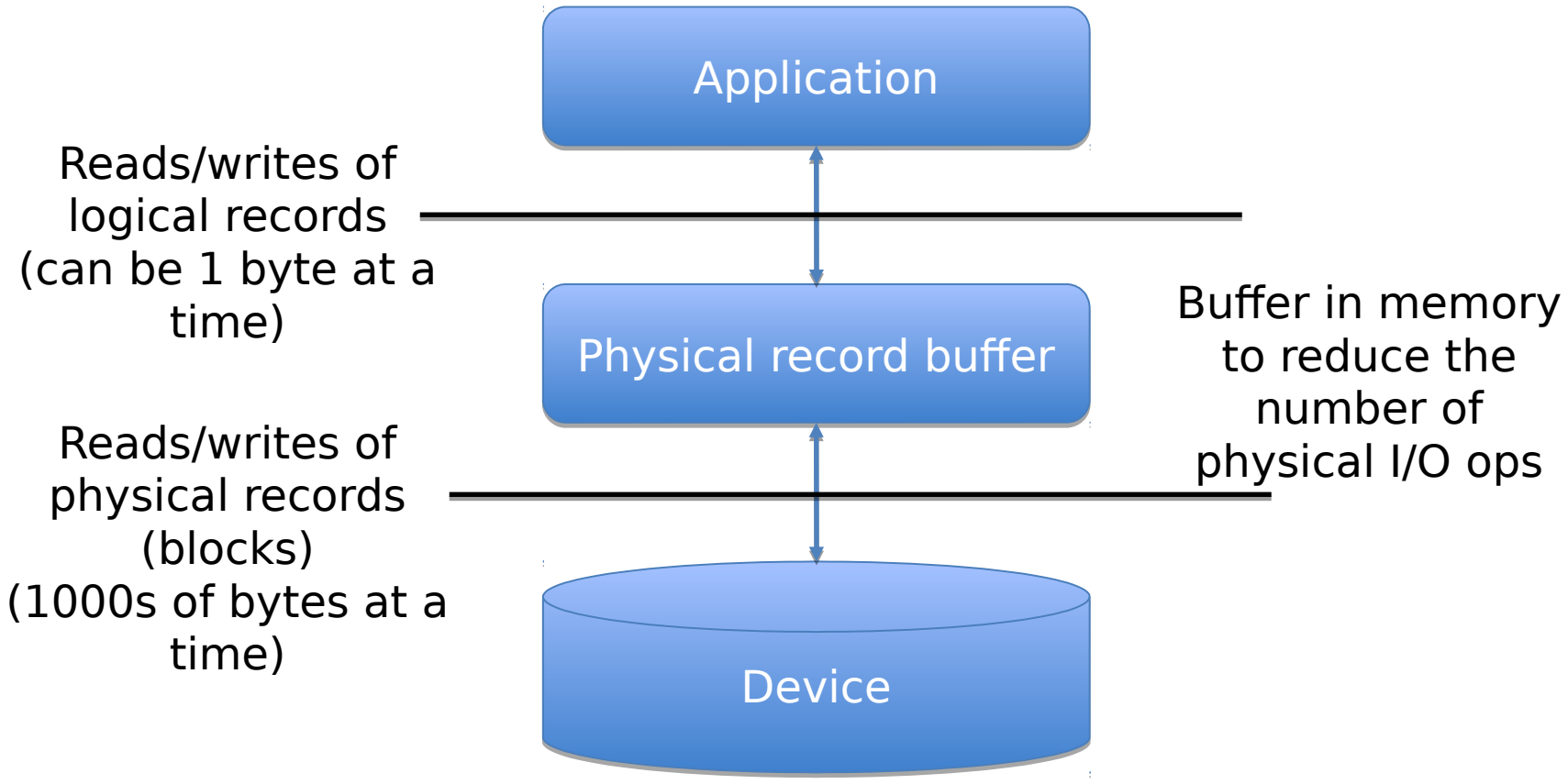
I/O System Layers



File Systems

- Abstraction on top of persistent storage
  - Magnetic disk
  - Flash memory (e.g., USB thumb drive)
- Devices provide
  - Storage that (usually) survives across machine crashes
  - Block level (random) access
  - Large capacity at low cost
  - Relatively slow performance
    - Magnetic disk read takes 10-20M processor instructions

Logical Record Blocking



Note: double-buffering allows processing to overlap transfer of next physical record

File System as Illusionist: Hide Limitations of Physical Storage

- Persistence of data stored in file system:
  - Even if crash happens during an update
  - Even if disk block becomes corrupted
  - Even if flash memory wears out
- Controlled access to shared data
- Naming:
  - Named data instead of disk block numbers
  - Byte addressable data even though devices are block-oriented
  - Directories instead of flat storage
- Performance:
  - Data placement and data structure organization
  - Cached data

File System Abstraction

- File system
  - Persistent, named data
  - Operating system crashes (and disk errors) leave file system in a valid state
- Access control on data
- File: named collection of data
  - Linear sequence of bytes (or a set of sequences or records)
  - Read/write or memory mapped
  - Hierarchical organization (directories, subdirectories)
- Performance
  - Achieve close to the hardware limit in the average case

Files

- Metadata
  - owner, access permissions, timestamps (creation, last written), size, reference count, lock, etc.
- Data
  - May be unstructured or structured:
    - Stream of bytes (even if stored as blocks)
    - Records are collections of related fields, often with a key field used for searching and sorting
  - Alternate data streams (resource forks in Mac OS)

Connection-Oriented Interface

- Explicit open and close operations for files
  - OS creates an internal data structure on open
- Read and write ops use descriptor (a.k.a. handle or stream) to identify the internal data structure
  - No need to reparse file name
- Per-open data structure contains:
  - Access permission under which file was opened
  - Location of file (e.g., inode number)
  - Pointer to current byte or record for sequential access

UNIX Data Structures for Open Files

per-process

system-wide

on disk

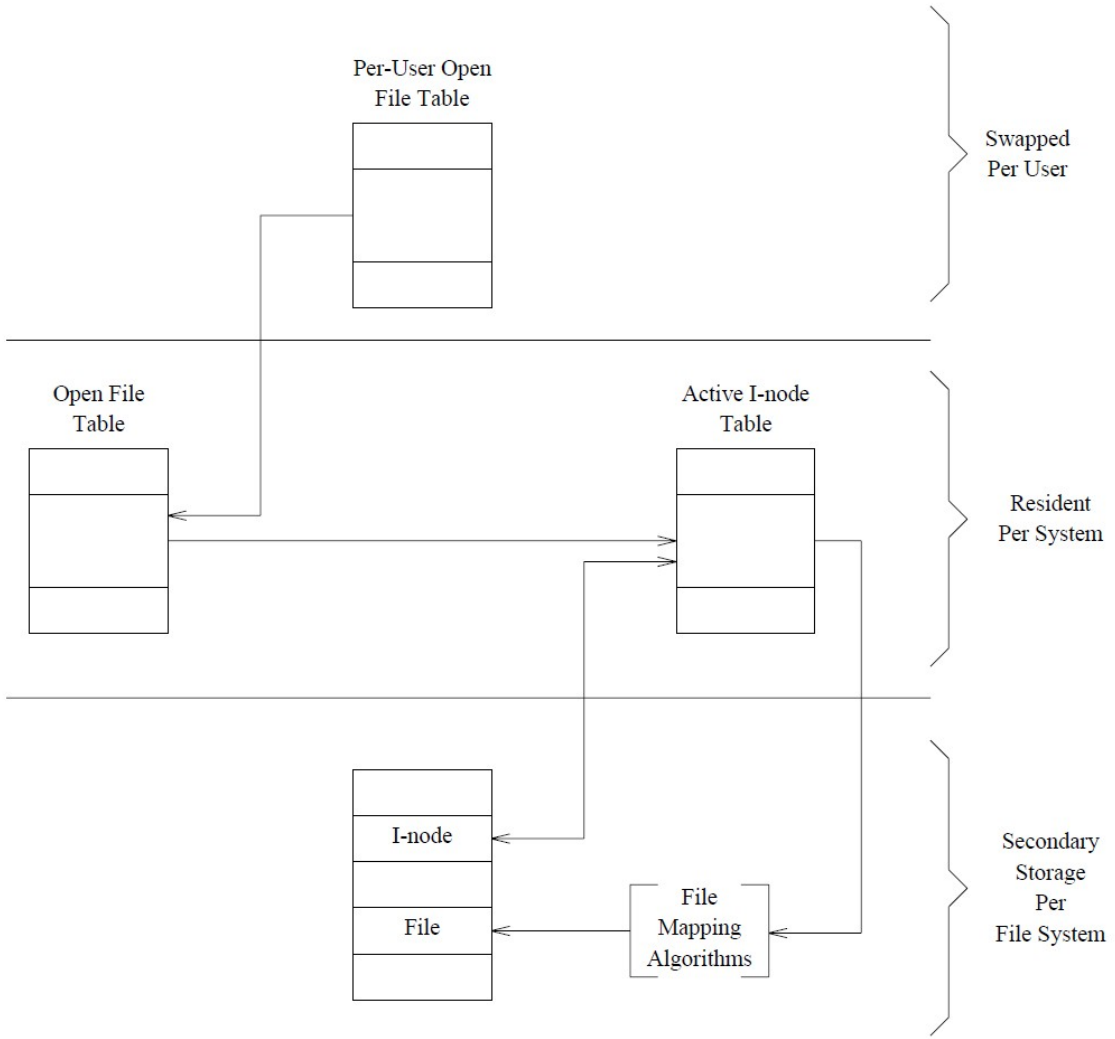


Diagram from K. Thompson, “UNIX Implementation,” Bell System Tech. Jrnl., 1978

UNIX Data Structures for Open Files (2)

per-process

system-wide

copy inode info into kernel memory

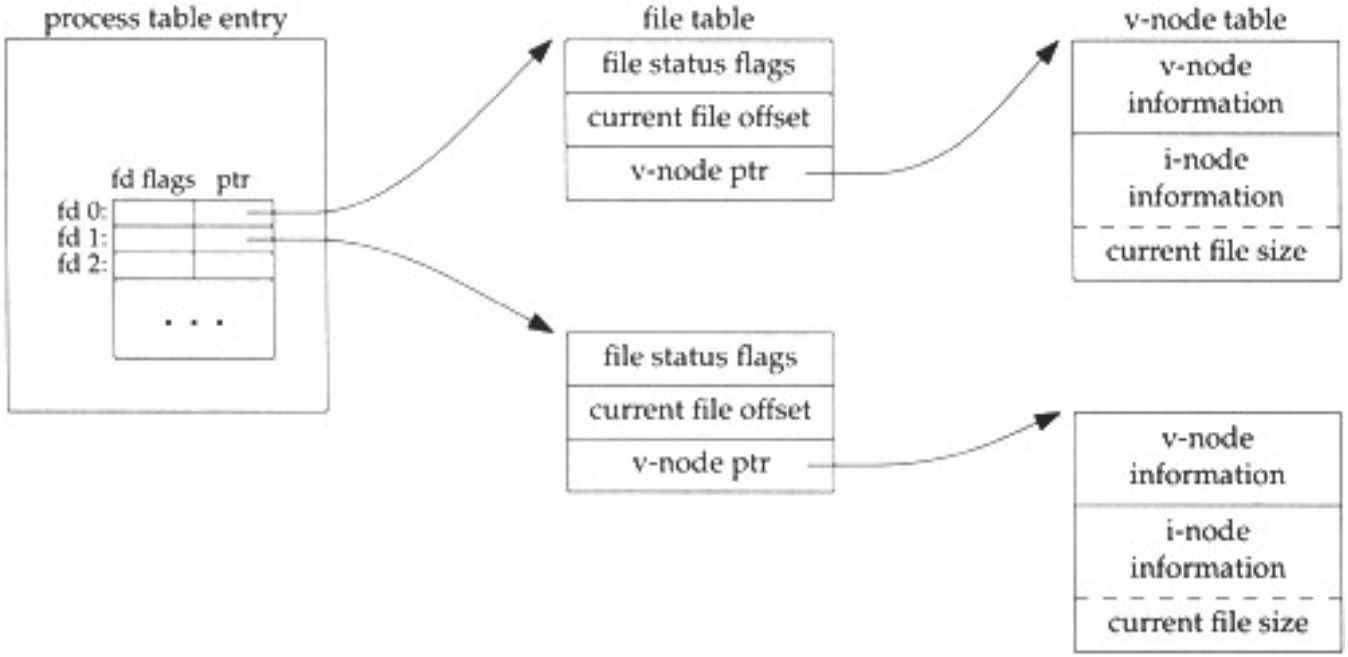


Diagram from W. Richard Stevens, Chapter 3, “UNIX File I/O,” Advanced Programming in the UNIX® Environment, 2003

UNIX Data Structures for Open Files (3)

Two processes open the same file – each has its own current file pointer (“offset”)

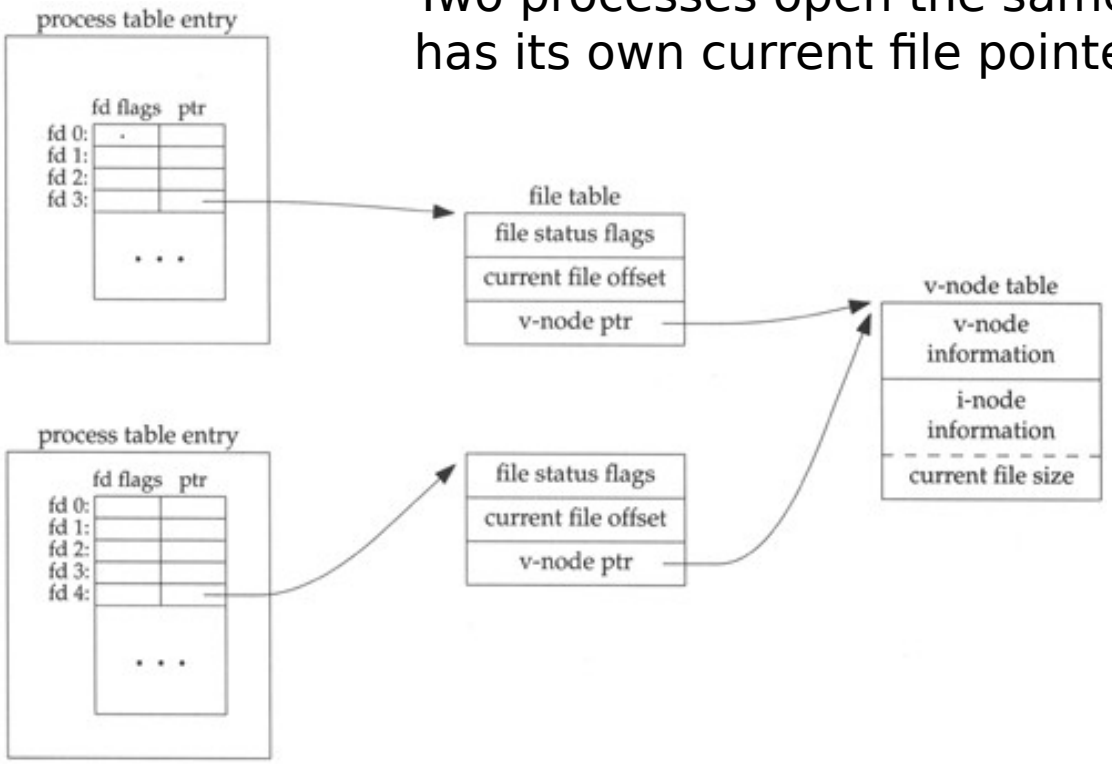


Diagram from W. Richard Stevens, Chapter 3, “UNIX File I/O,” Advanced Programming in the UNIX® Environment, 2003

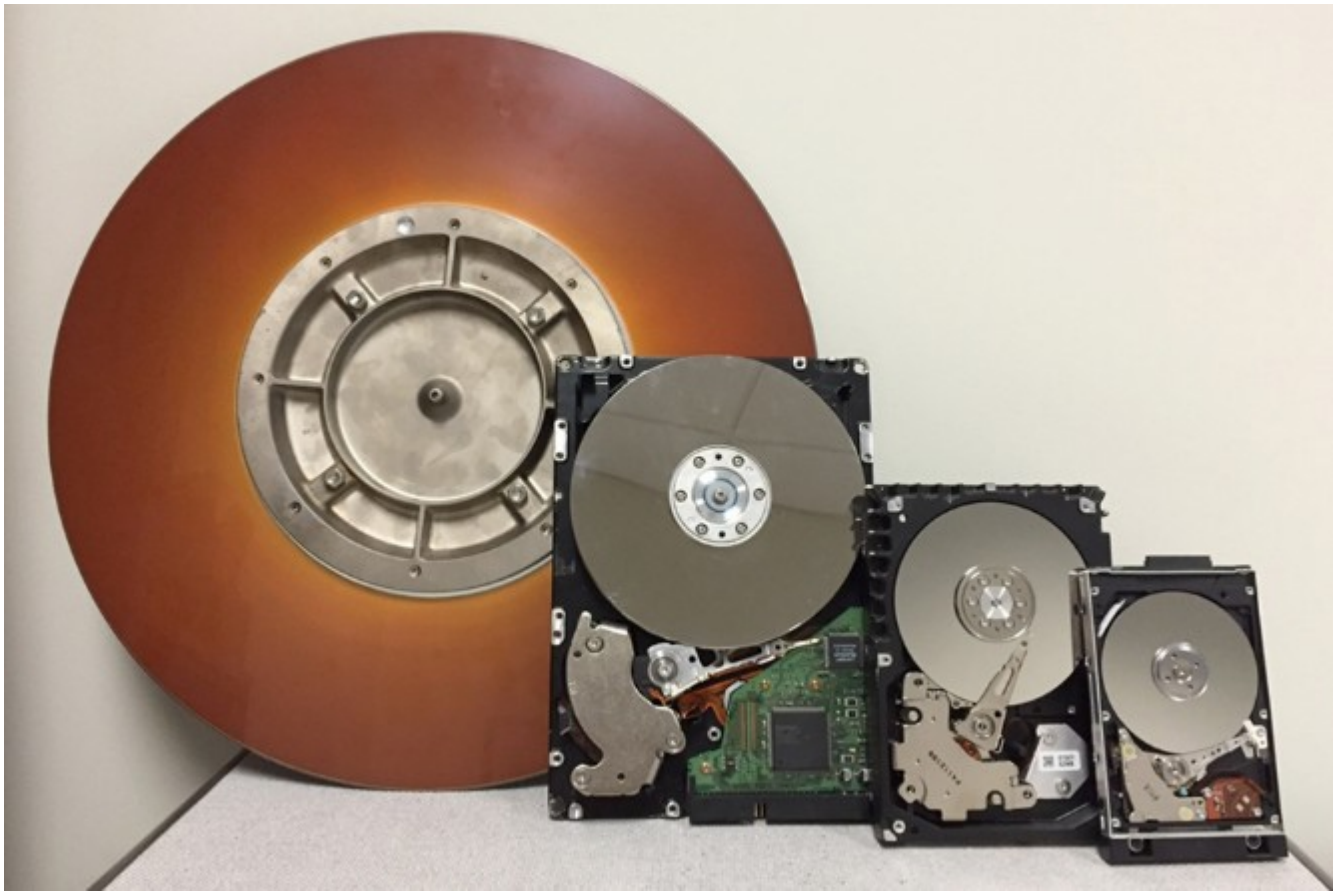
UNIX File System API

- create, link, unlink, createdir, rmdir
  - Create file, link to file, remove link
  - Create directory, remove directory
- open, close, read, write, seek
  - Open/close a file for reading/writing
  - Seek resets current position
- fsync
  - File modifications can be cached
  - fsync forces modifications to disk (like a memory barrier)

Storage Devices

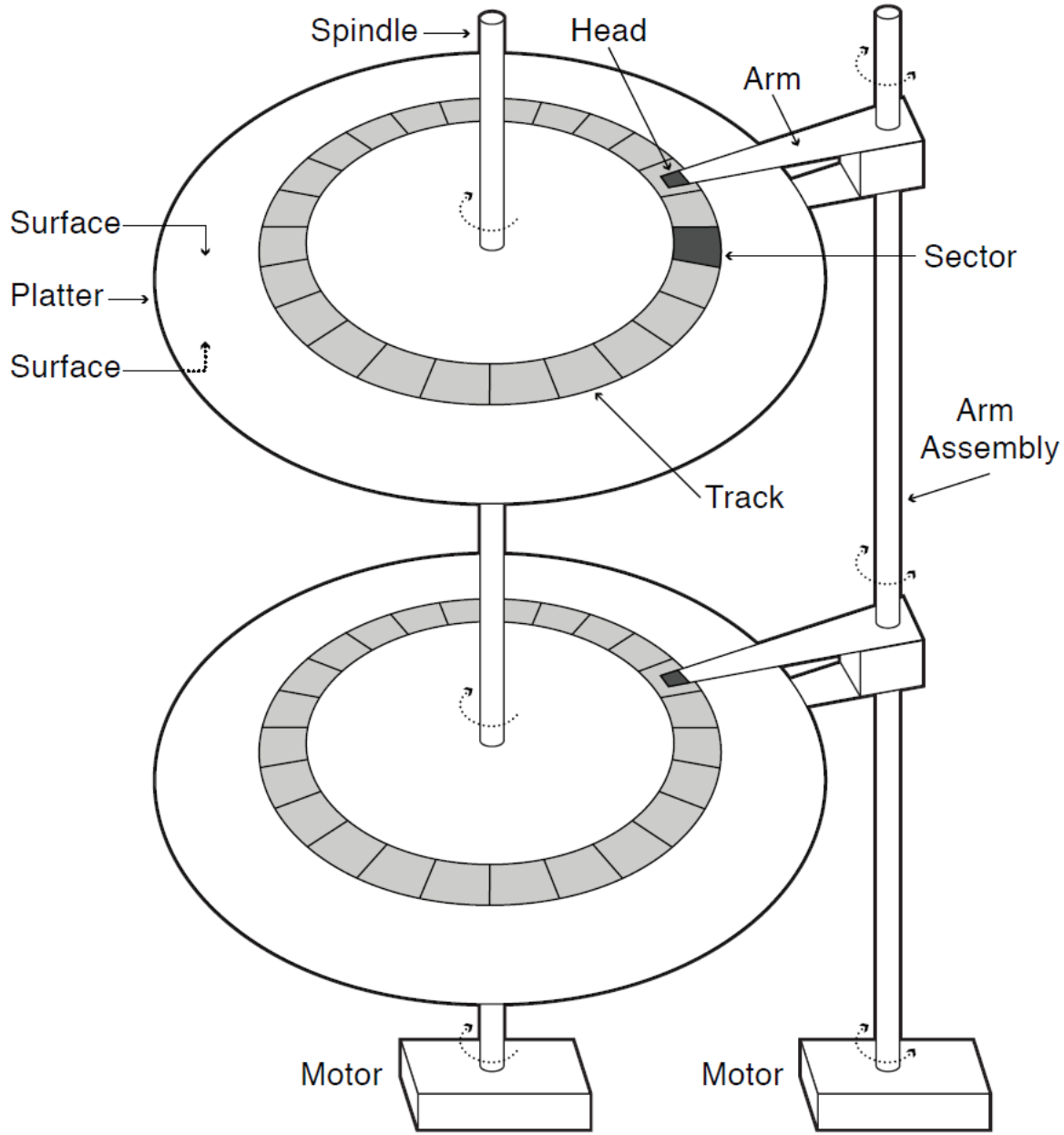
- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for streaming access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (50x disk)
  - Block level random access
  - Good performance for reads; worse for random writes

Magnetic Disk



Left-to-right:

- Single 14-inch disk platter, 16 MB removable cartridge disk for 1980s rack-mount Ampex DFR-900 series HDD, 3600 rpm, 30 ms avg. seek time
- Three 5.12-inch (5.25-inch FF) platters, 19.2 GB internal drive, Quantum Bigfoot TS19A011, 4000 rpm, 10.5 ms seek
- Four 3.74-inch (3.5-inch FF) platters, likely 80 GB internal drive, Seagate Barracuda, 7200 rpm, 8 ms seek
- Single 2.56-inch (2.5-inch FF) platter, 10.06 GB internal drive, IBM Travelstar 07N5138, 4200 rpm, 12 ms seek


Spindle
Head
Arm
Surface
Platter
Surface
Sector
Arm
Assembly
Track
Motor
Motor

Disk Tracks

- ~ 1 micron wide
  - Wavelength of light is ~ 0.5 micron
  - Resolution of human eye: 50 microns
  - 100K tracks on a typical 2.5” disk
- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)
- Track length varies across disk
  - Outside: More sectors per track, higher bandwidth
  - Disk is organized into regions of tracks with same # of sectors/track
  - Only outer half of radius is used
    - Most of the disk area in the outer regions of the disk

Sectors

Sectors contain sophisticated error correcting codes

- Disk head magnet has a field wider than track
- Hide corruptions due to neighboring track writes

- Sector sparing
  - Remap bad sectors transparently to spare sectors on the same surface
- Slip sparing
  - Remap all sectors (when there is a bad sector) to preserve sequential behavior
- Track skewing
  - Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

Disk Performance

- Disk Latency =
  - Seek Time + Rotation Time + Transfer Time
  - Seek Time: time to move disk arm over track (1-20ms)
    - Fine-grained position adjustment necessary for head to “settle”
    - Head switch time ~ track switch time (on modern disks)
  - Rotation Time: time to wait for disk to rotate under disk head
    - Disk rotation: 4 – 15ms (depending on price of disk)
    - On average, only need to wait half a rotation
  - Transfer Time: time to transfer data onto/off of disk
    - Disk head transfer rate: 50-100MB/s (5-10 usec/sector)
    - Host transfer rate dependent on I/O connector (USB, SATA, …)

Toshiba Disk (2008)

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle speed	7200 RPM
Average seek time read/write	10.5 ms/ 12.0 ms
Maximum seek time	19 ms
Track-to-track seek time	1 ms
Transfer rate (surface to buffer)	54–128 MB/s
Transfer rate (buffer to host)	375 MB/s
Buffer memory	16 MB
Power	
Typical	16.35 W
Idle	11.68 W

Question

- How long to complete 500 random disk reads, in FIFO order?

Question

- How long to complete 500 random disk reads, in FIFO order?
  - Seek: average 10.5 msec
  - Rotation: average 4.15 msec
  - Transfer: 5-10 usec
- 500 * (10.5 + 4.15 + 0.01)/1000 = 7.3 seconds

Question

- How long to complete 500 sequential disk reads?

Question

- How long to complete 500 sequential disk reads?
  - Seek Time: 10.5 ms (to reach first sector)
  - Rotation Time: 4.15 ms (to reach first sector)
  - Transfer Time: (outer track)

    500 sectors * 512 bytes / 128MB/sec = 2ms

Total: 10.5 + 4.15 + 2 = 16.7 ms

Might need an extra head or track switch (+1ms)

Track buffer may allow some sectors to be read off disk out of order (-2ms)

Question

- How large a transfer is needed to achieve 80% of the max disk transfer rate?

Question

- How large a transfer is needed to achieve 80% of the max disk transfer rate?

  Assume x rotations are needed, then solve for x:

  0.8 (10.5 ms + (1ms + 8.5ms) x) = 8.5ms x

Total: x = 9.1 rotations, 9.8MB

Question

- How long to complete 500 random disk reads, in any order?

Question

- How long to complete 500 random disk reads, in any order?
  - Disk seek: 1ms (most will be short)
  - Rotation: 4.15ms
  - Transfer: 5-10usec
- Total: 500 * (1 + 4.15 + 0.01) = 2.2 seconds
  - Would be a bit shorter with R-CSCAN
  - vs. 7.3 seconds if FIFO order

Question

- How long to read all of the bytes off of a disk?

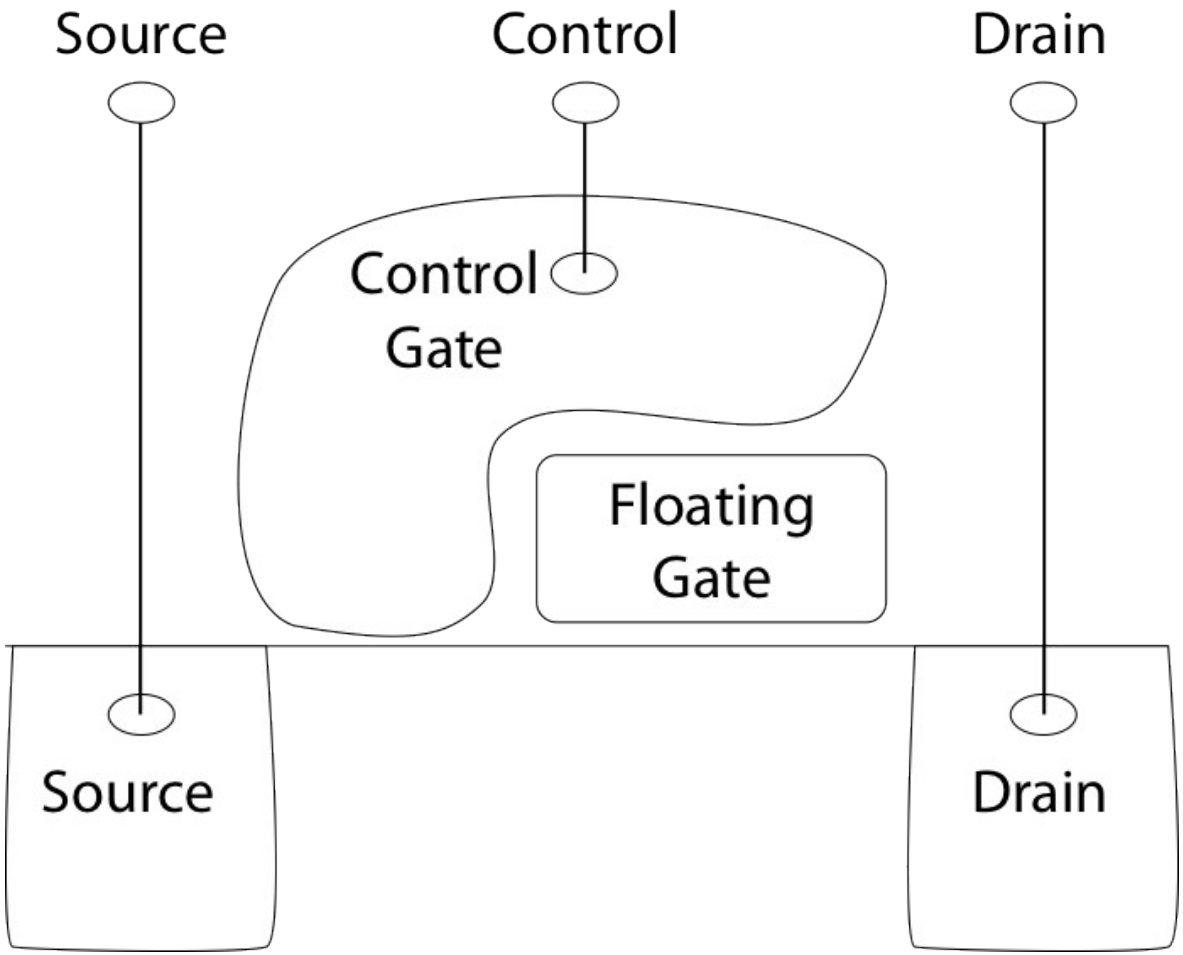
Question

- How long to read all of the bytes off of a disk?
  - Disk capacity: 320GB
  - Disk bandwidth: 54-128MB/s
- Transfer time =

  Disk capacity / average disk bandwidth

  ~ 3500 seconds (1 hour)

Flash Memory



Flash Memory

- Writes must be to “clean” cells; no update in place
  - Large block erasure required before write
  - Erasure block: 128 – 512 KB
  - Erasure time: Several milliseconds
- Write/read page (2-4KB)
  - 50-100 usec

Flash Drive (2011)

Size	
Capacity	300 GB
Page Size	4KB
Performance	
Bandwidth (Sequential Reads)	270 MB/s
Bandwidth (Sequential Writes)	210 MB/s
Read/Write Latency	75 μs
Random Reads Per Second	38,500
Random Writes Per Second	2,000 (2,400 with 20% space reserve)
Interface	SATA 3 Gb/s
Endurance	
Endurance	1.1 PB (1.5 PB with 20% space reserve)
Power	
Power Consumption Active/Idle	3.7 W / 0.7 W

Question

- Why are random writes so slow?
  - Random write: 2000/sec
  - Random read: 38500/sec

Flash Translation Layer

- Flash device firmware maps logical page # to a physical location
  - Garbage collect erasure block by copying live pages to new location, then erase
    - More efficient if blocks stored at same time are deleted at same time (e.g., keep blocks of a file together)
  - Wear-leveling: only write each physical page a limited number of times
  - Remap pages that no longer work (sector sparing)
- Transparent to the device user

File System – Flash

- How does Flash device know which blocks are live?
  - Live blocks must be remapped to a new location during erasure
- TRIM command
  - File system tells device when blocks are no longer in use

Device Intelligence

- Disk and flash devices now have simple CPUs
  - Disk head scheduling, remapping, SSD write leveling
- Historical trends based on technological ratios
  - I/O channels and disk controllers in 1960s had a fair amount of processing power since the main CPU was expensive
    - E.g., to reduce load on CPU, channel could independently search for a record within a file when given a search key
  - When CPU costs decreased, I/O devices became dumb so that you could build a cheaper overall system => OS file system did more work
  - Now, CPUs are very cheap so device manufacturers include more intelligence on devices

File System Workload

- File sizes
  - Are most files small or large?
  - Which accounts for more total storage: small or large files?

File System Workload

- File sizes
  - Are most files small or large?
    - SMALL
  - Which accounts for more total storage: small or large files?
    - LARGE

File System Workload

- File access
  - Are most accesses to small or large files?
  - Which accounts for more total I/O bytes: small or large files?

File System Workload

- File access
  - Are most accesses to small or large files?
    - SMALL
  - Which accounts for more total I/O bytes: small or large files?
    - LARGE

File System Workload

- How are files used?
  - Most files are read/written sequentially
  - Some files are read/written randomly
    - Ex: database files, swap files
  - Some files have a pre-defined size at creation
  - Some files start small and grow over time
    - Ex: program stdout, system logs

File System Design

- For small files:
  - Small blocks for storage efficiency
  - Concurrent ops more efficient than sequential
  - Files used together should be stored together
- For large files:
  - Storage efficient (large blocks)
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation
  - Whether file will become small or large
  - Whether file is persistent or temporary
  - Whether file will be used sequentially or randomly

File System Abstraction

- Directory
  - Group of named files or subdirectories
  - Mapping from file name to file metadata location
- Path
  - String that uniquely identifies file or directory
  - Ex: /web/home/mark/public_html/3220.html
- Links
  - Hard link: link from name to metadata location
  - Soft link: link from name to alternate name
- Mount
  - Mapping from name in one file system to root of another

Directory Levels

	System-wide	Per-device
Flat		CP/M
Two-level	OS/360 – top-level is user name	
Tree-structured	UNIX	Windows

Path Names

- Absolute path
  - Fully qualified
  - Starts at root (“/” in Unix systems)
- Relative path
  - Partially qualified, ex: public_html/3220.html
  - Unqualified, ex: 3220.html
  - Starts at current working directory
  - . = this directory, .. = parent directory

File Types

- Extension identifier (.c, .o)
  - Can be used to specify usage or structure
  - OS can associate a specific application with an extension
- Magic number within file
  - Identify usage or structure
  - First four bytes for ELF file are: 177 E L F

File Types (2)

- Regular file
  - Text (e.g., ASCII)
  - Binary
- Directory
  - Writes must be restricted to preserve structure
- Special file
  - Maps physical I/O device to the file system

File Types (3)

- UNIX philosophy as stated by Ritchie and Thompson (inventors):
  - “the structure of files is controlled by the programs that use them, not by the system”
- Alternate approach, e.g., IBM z/OS
  - “An access method defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, macros to define and process data sets, and utility programs to process data sets.”
    - Ex: Virtual Sequential Access Method (VSAM)
    - Ex: Queued Sequential Access Method (QSAM)