CS162
Operating Systems and Systems Programming
Lecture 17

Data Storage to File Systems

October 29, 2019

Prof. David Culler

http://cs162.eecs.Berkeley.edu

Read: A&D Ch 12, 13.1-3.2

Recall: OS Storage abstractions

Key Unix I/O Design Concepts

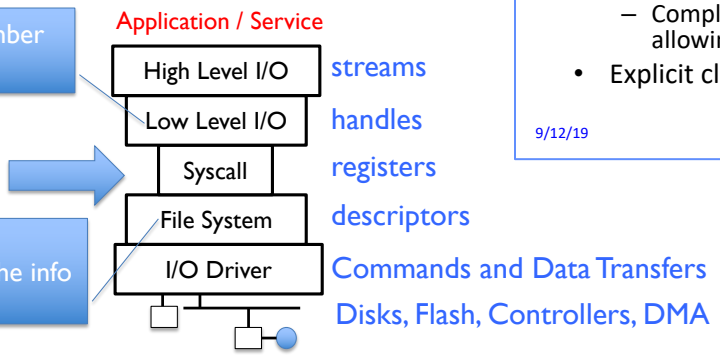
- Uniformity – everything is a file
  - file operations, device I/O, and interprocess communication through open, read/write, close
  - Allows simple composition of programs
    - find | grep | wc …
- Open before use
  - Provides opportunity for access control and arbitration
  - Sets up the underlying machinery, i.e., data structures
- Byte-oriented
  - Even if blocks are transferred, addressing is in bytes
- Kernel buffered reads
  - Streaming and block devices looks the same, read blocks yielding processor to other task
- Kernel buffered writes
  - Completion of out-going transfer decoupled from the application, allowing it to continue
- Explicit close

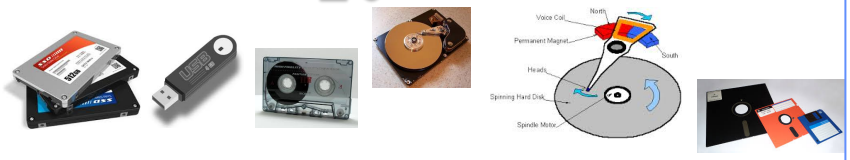
9/12/19

What's below the surface ??

File descriptor number - an int

File Descriptors
- a struct with all the info about the files





9/12/19 cs162 fa19 L5 57

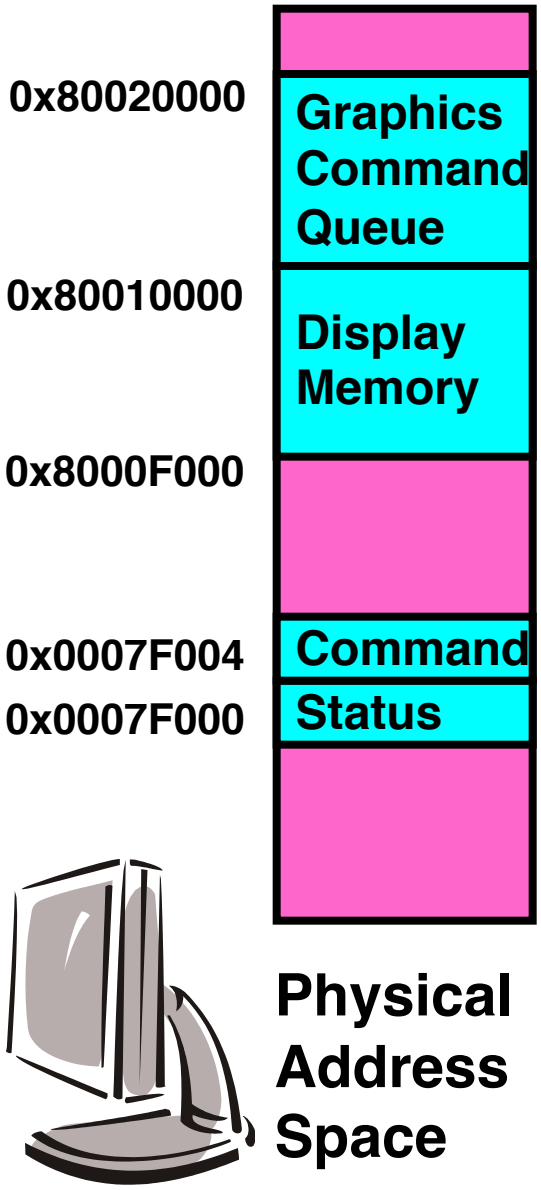
The file system abstraction

- File
  - Named collection of data in a file system
  - POSIX File data: sequence of bytes
    - Could be text, binary, serialized objects, …
  - File Metadata: information about the file
    - Size, Modification Time, Owner, Security info
    - Basis for access control
- Directory
  - "Folder" containing files & Directories
  - Hierachical (graphical) naming
    - Path through the directory graph
    - Uniquely identifies a file or directory
      - /home/ff/cs162/public_html/fa14/index.html
  - Links and Volumes (later)

9/12/19 cs162 fa19 L5 28

Review: Memory-Mapped Display Controller

- Memory-Mapped:
  - Hardware maps control registers and display memory into physical address space
    - » Addresses set by HW jumpers or at boot time
  - Simply writing to display memory (also called the "frame buffer") changes image on screen
    - » Addr: 0x8000F000 — 0x8000FFFF
  - Writing graphics description to cmd queue
    - » Say enter a set of triangles describing some scene
    - » Addr: 0x80010000 — 0x8001FFFF
  - Writing to the command register may cause on-board graphics hardware to do something
    - » Say render the above scene
    - » Addr: 0x0007F004
- Can protect with address translation

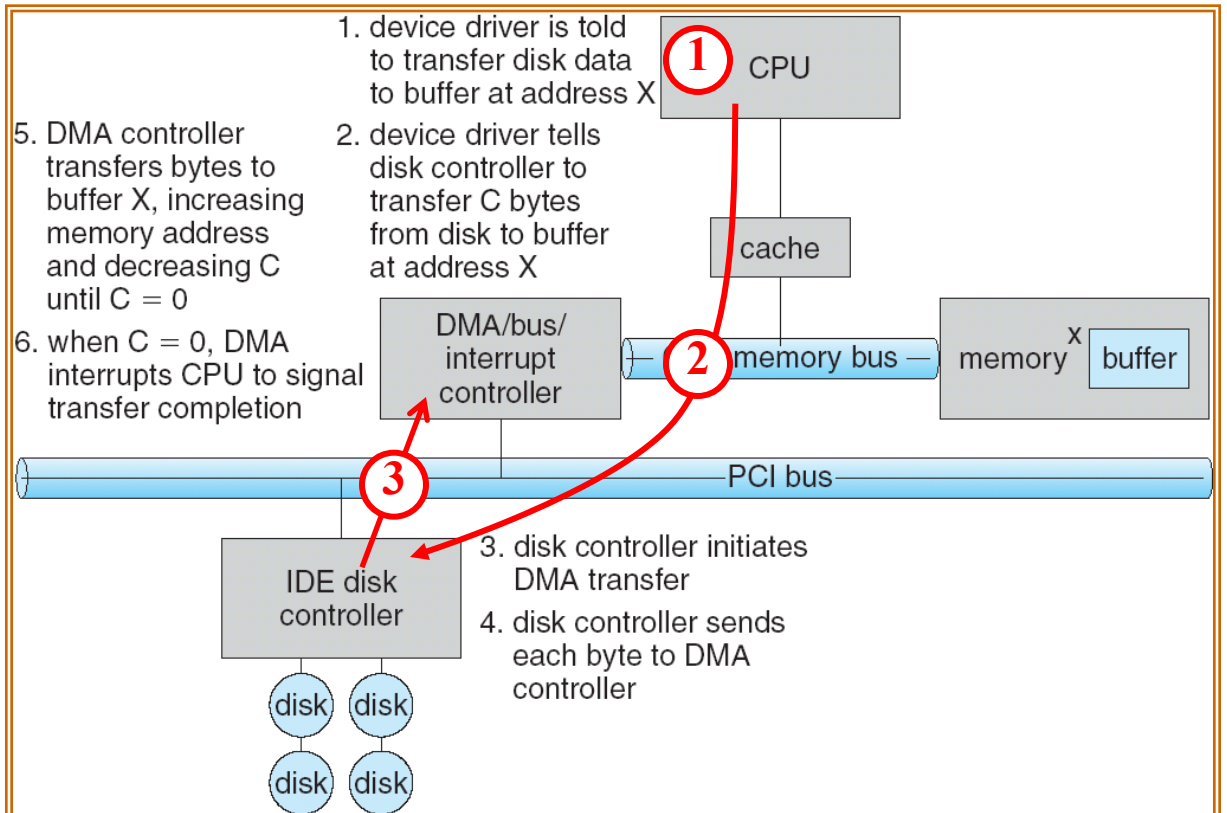


Review: Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size

- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly

- Sample interaction with DMA controller (from OSC book):

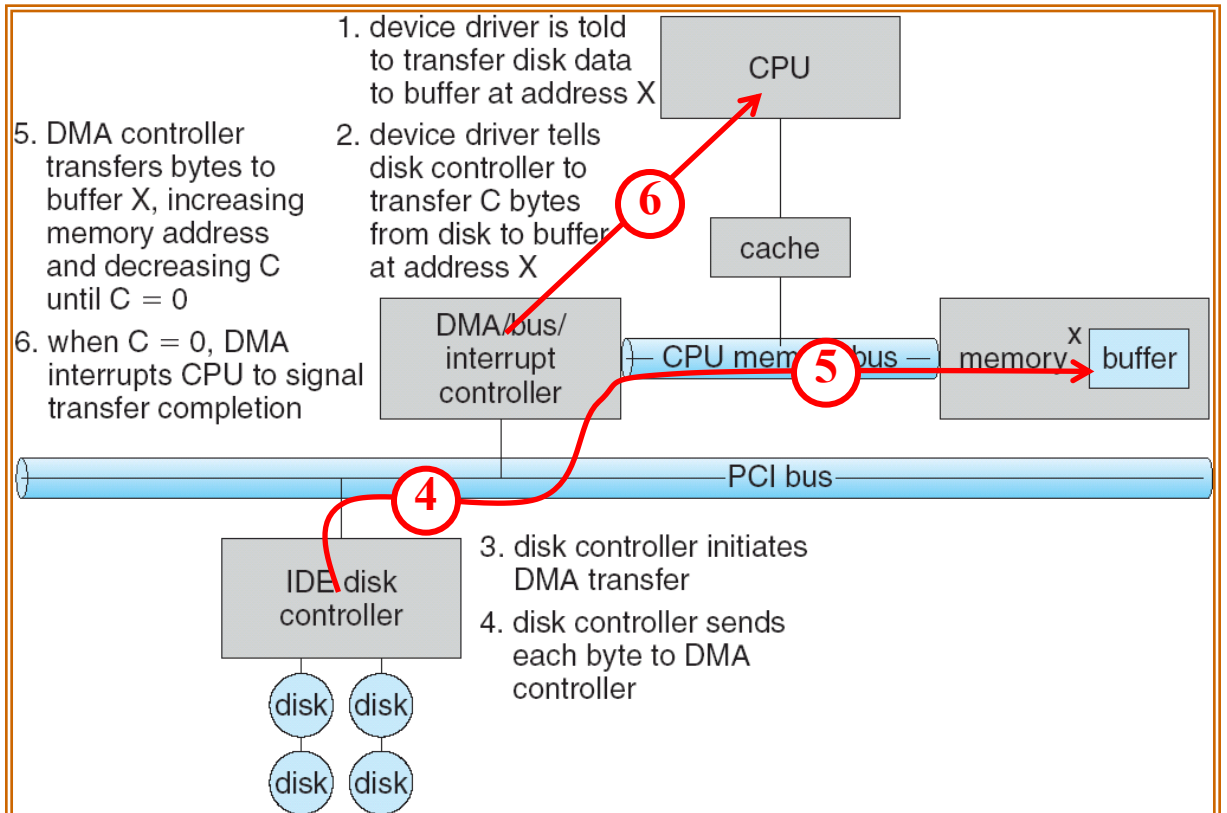


Review: Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size

- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly

- Sample interaction with DMA controller (from OSC book):



Recall: I/O Device Notifying the OS

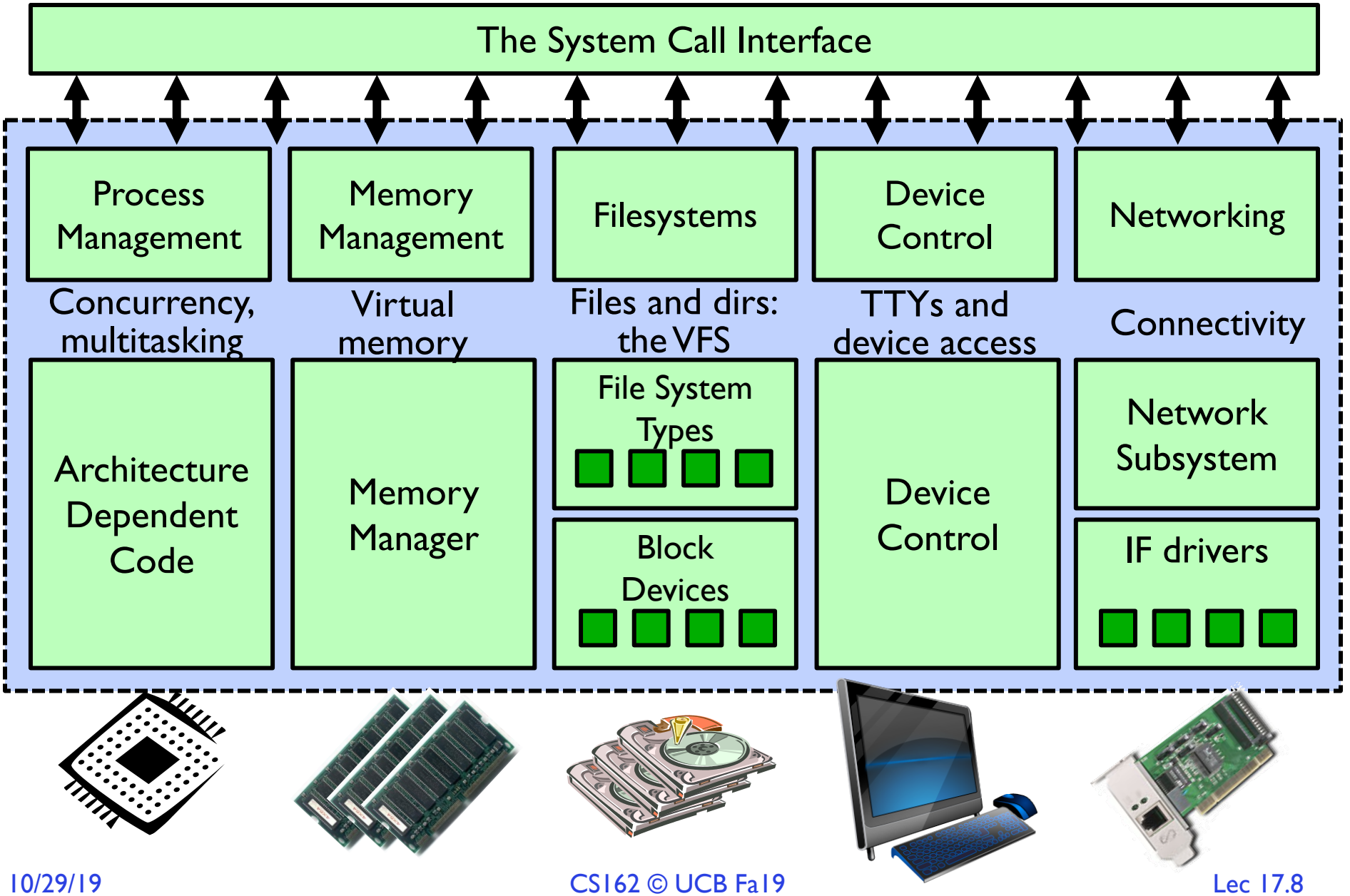
- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - » I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - » Interrupt for first incoming packet
    - » Poll for following packets until hardware queues are empty

Recall: Device Drivers

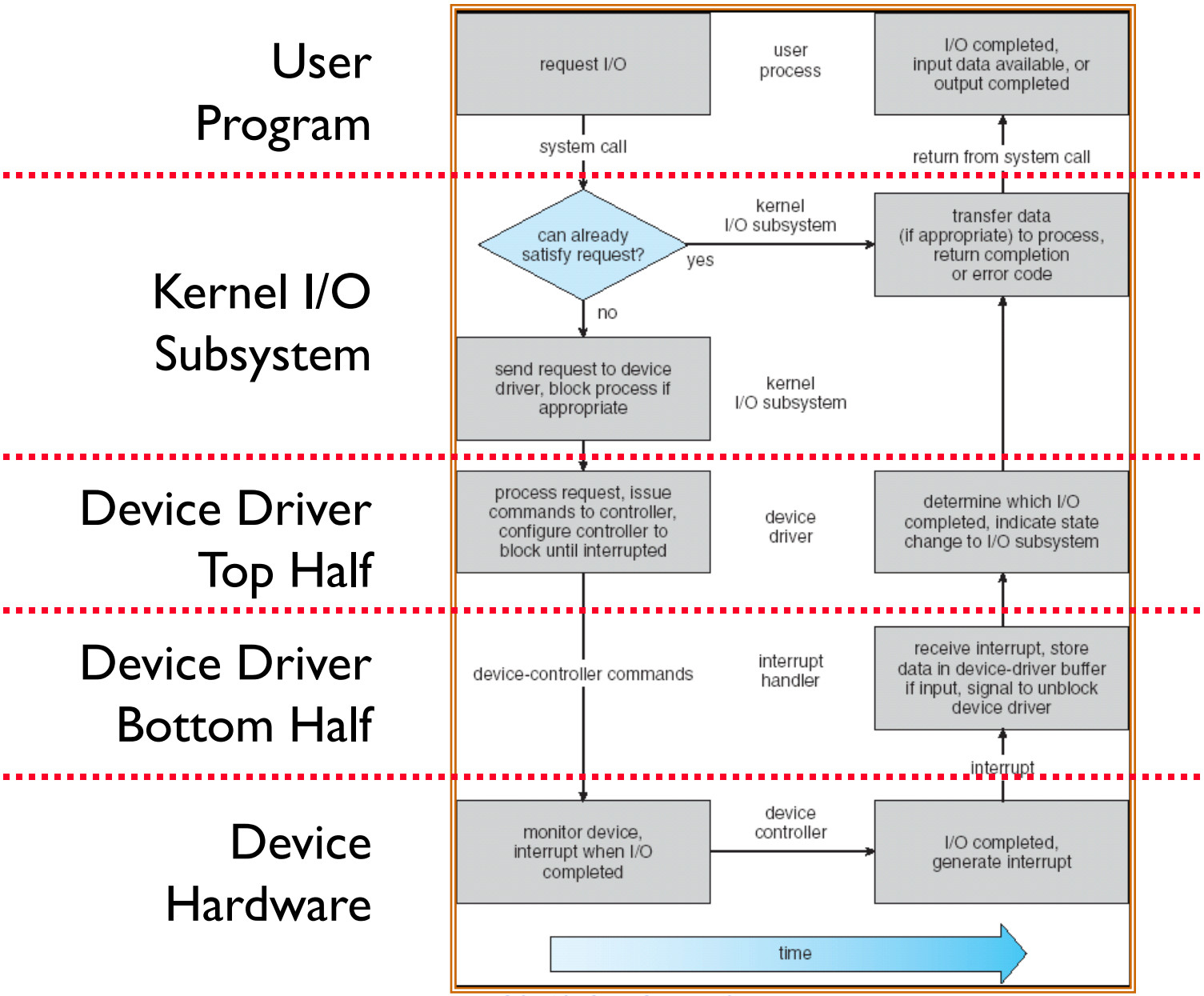
- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - Special device-specific configuration supported with the `ioctl()` system call

- Device Drivers typically divided into two pieces:
  - Top half: accessed in call path from system calls
    - » implements a set of standard, cross-device calls like `open()`, `close()`, `read()`, `write()`, `ioctl()`, `strategy()`
    - » This is the kernel's interface to the device driver
    - » Top half will *start* I/O to device, may put thread to sleep until finished
  - Bottom half: run as interrupt routine
    - » Gets input or transfers next block of output
    - » May wake sleeping threads if I/O now complete

Recall: Kernel Device Structure



Review: Life Cycle of An I/O Request

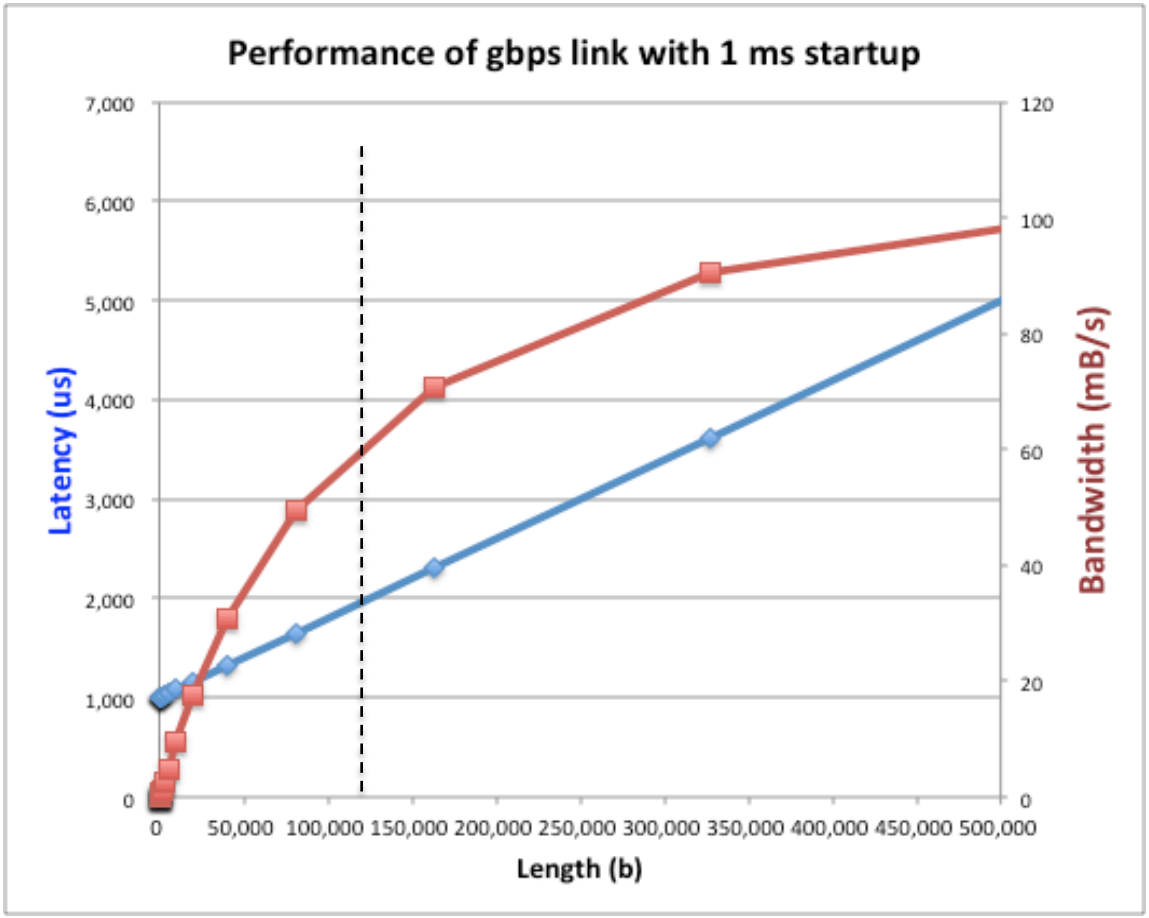


Basic I/O Performance Concepts

- *Response Time* or *Latency*: Time to perform an operation(s)
- *Bandwidth* or *Throughput*: Rate at which operations are performed (op/s)
  - Files: MB/s, Networks: Mb/s, Arithmetic: GFLOP/s
- *Start up* or "Overhead": time to initiate an operation
- Most I/O operations are roughly linear in *b* bytes
  - Latency(**b**) = Overhead + **b** / TransferCapacity

Example (Fast Network or SSD)

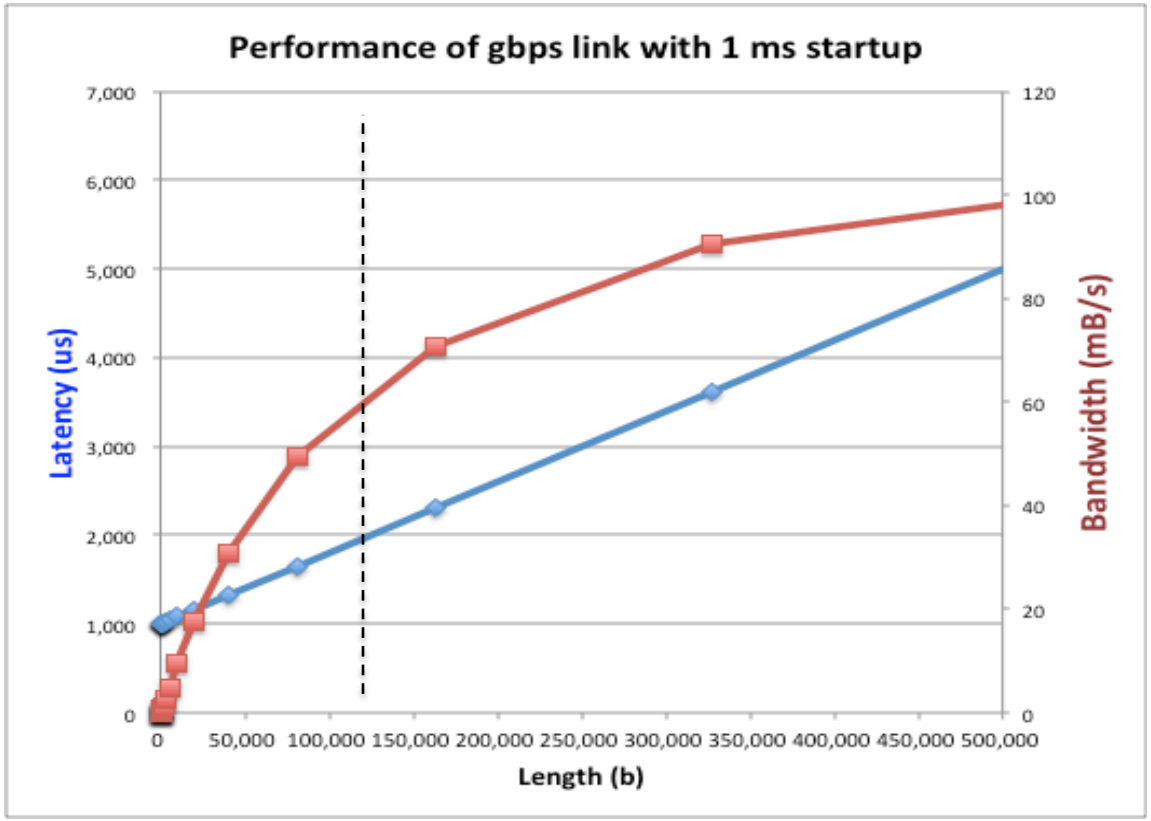
- Consider: 1 Gb/s link (B = 125 MB/s) w/ startup cost S = 1 ms



  - Latency(b) = S + b/B
  - Bandwidth = b/(S + b/B) = B*b/(B*S + b) = B/(B*S/b + 1)

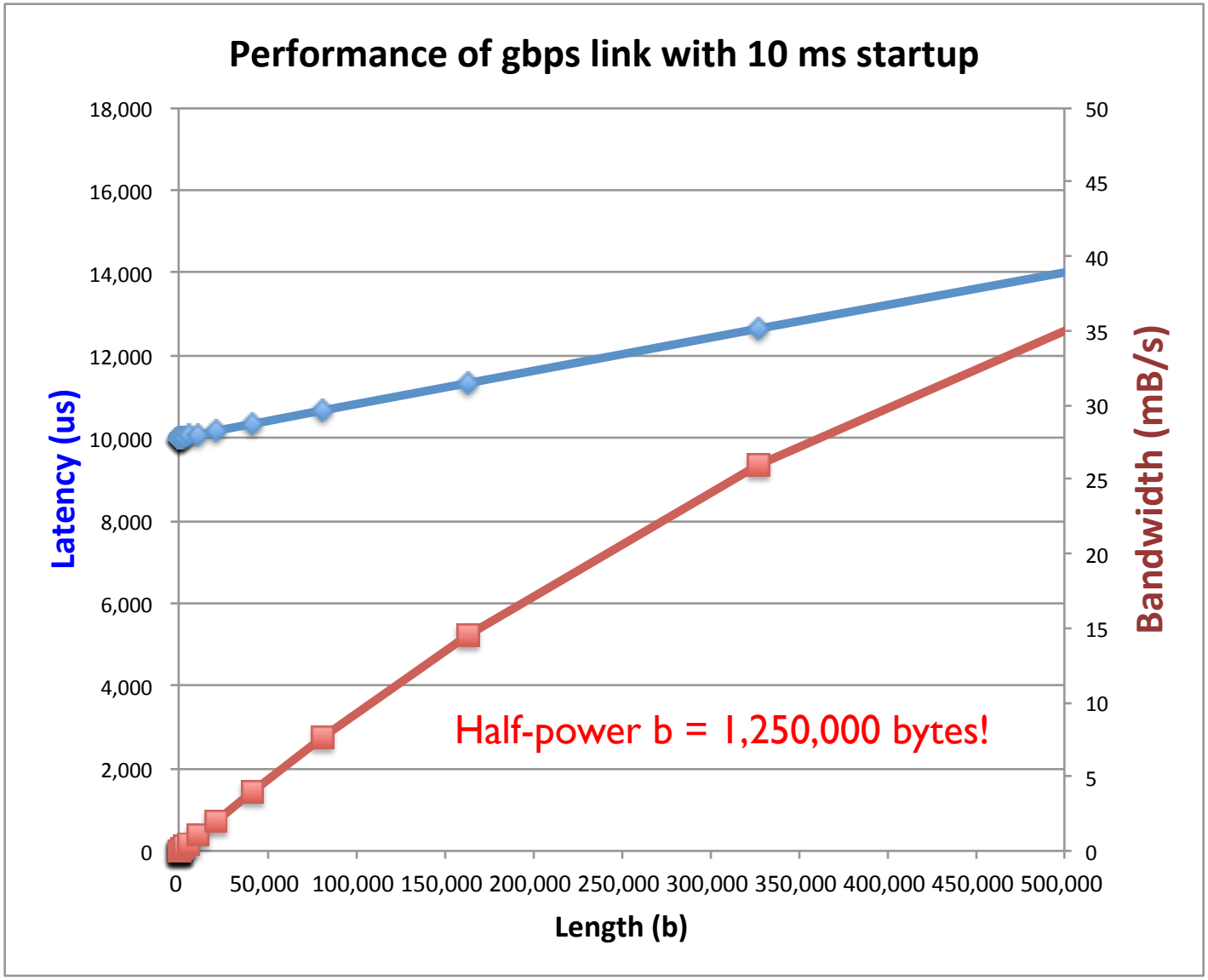
Example (Fast Network or SSD)

- Consider a 1 Gb/s link (B = 125 MB/s) w/ startup cost S = 1 ms



  - Half-power Bandwidth ⇒ B/(B*S/b + 1) = B/2
  - Half-power point occurs at b=S*B= 125,000 bytes

Example: at 10 ms startup (like Disk)



What Determines Peak BW for I/O ?

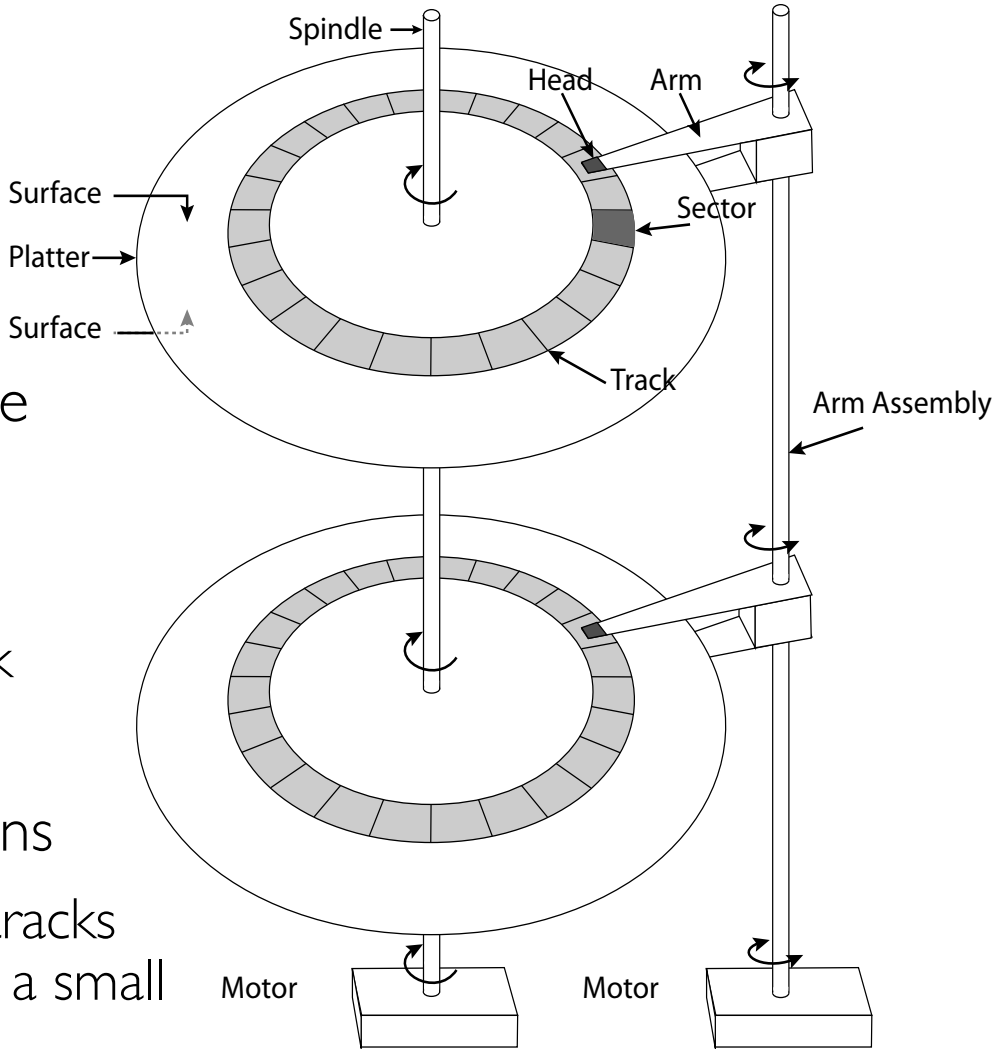
- Bus Speed
  - PCI-X: 1064 MB/s = 133 MHz x 64 bit (per lane)
  - ULTRA WIDE SCSI: 40 MB/s
  - Serial Attached SCSI & Serial ATA & IEEE 1394 (firewire): 1.6 Gb/s full duplex (200 MB/s)
  - USB 3.0 – 5 Gb/s
  - Thunderbolt 3 – 40 Gb/s

- Device Transfer Bandwidth
  - Rotational speed of disk
  - Write / Read rate of NAND flash
  - Signaling rate of network link

- Whatever is the bottleneck in the path…

The Amazing Magnetic Disk

- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
  - Heads position on cylinders

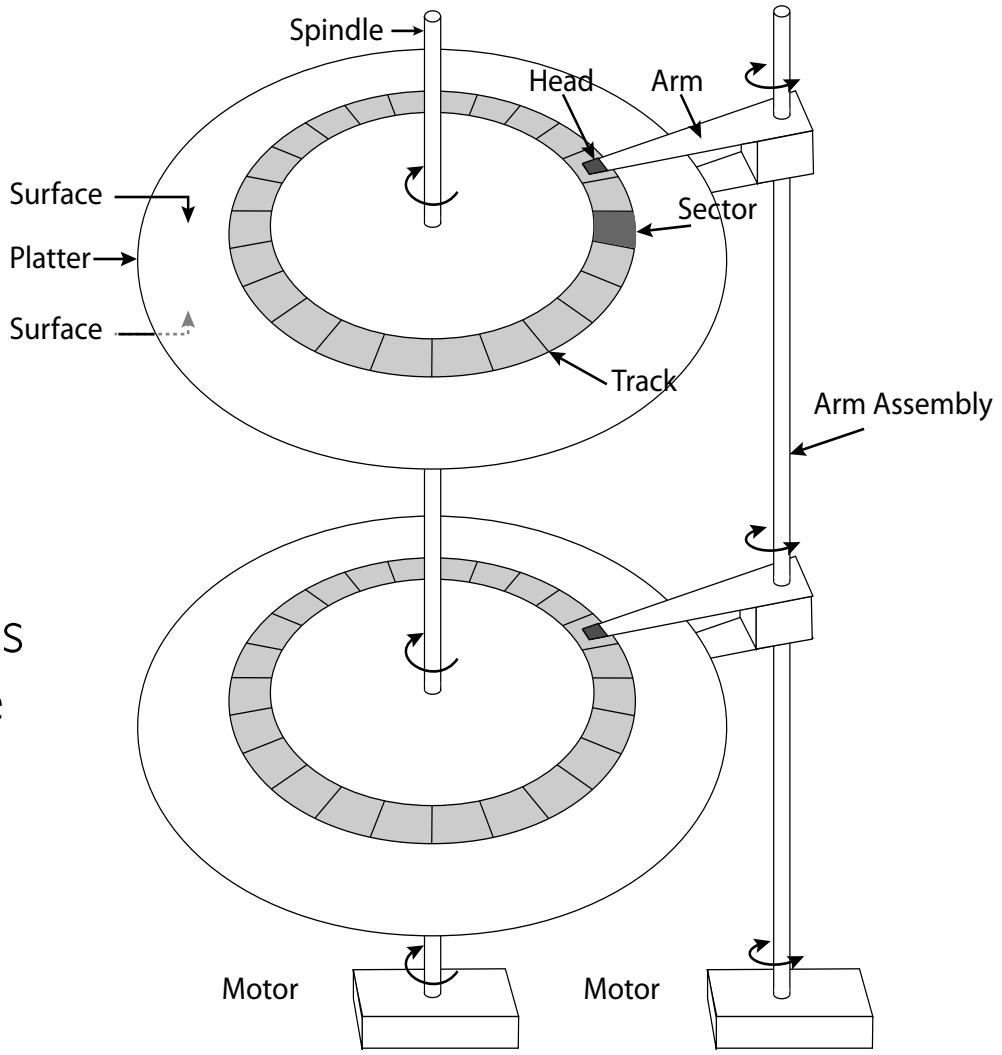


- Disk Tracks ~ 1μm (micron) wide
  - Wavelength of light is ~ 0.5μm
  - Resolution of human eye: 50μm
  - 100K tracks on a typical 2.5" disk

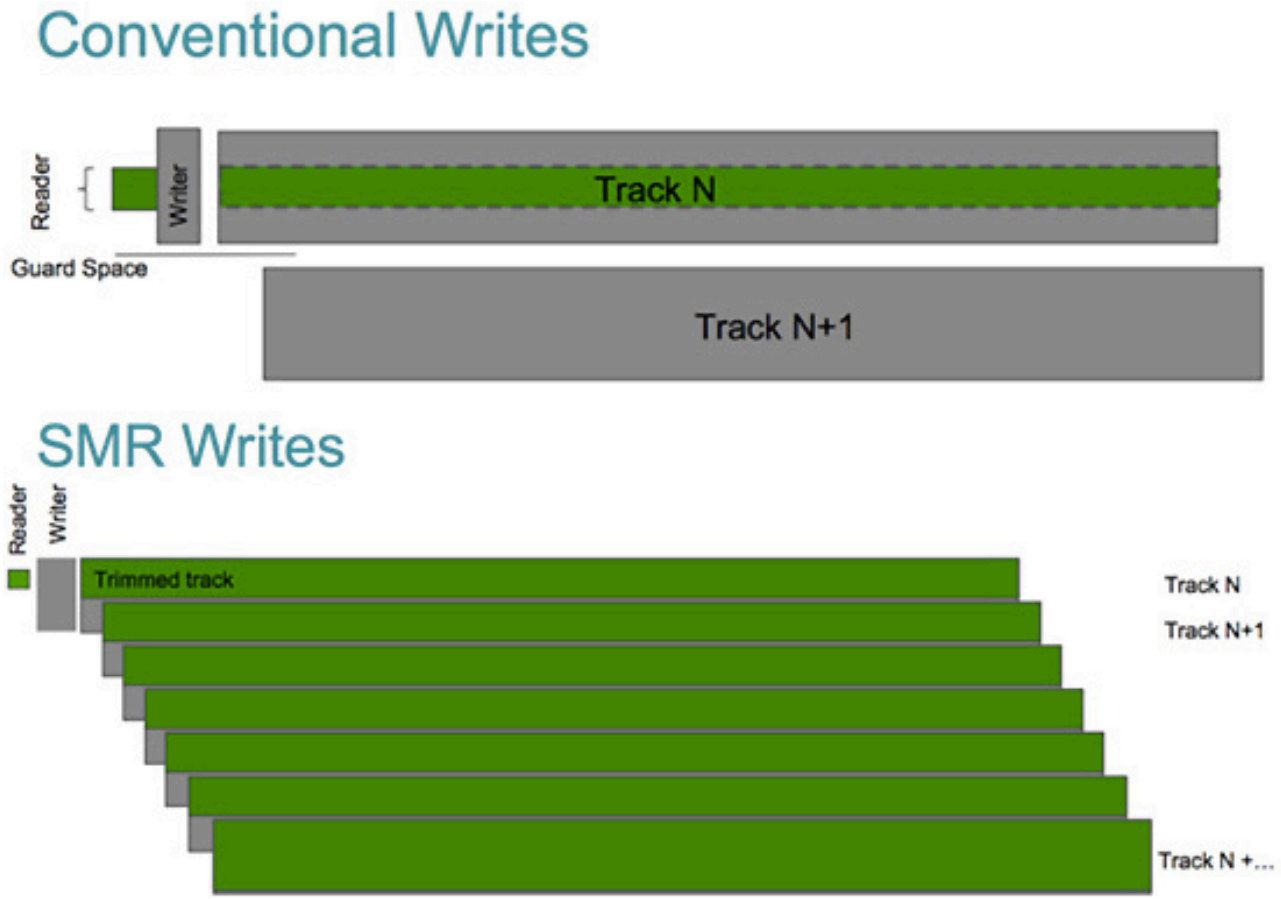
- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)

The Amazing Magnetic Disk

- 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
- Disks so big that some companies (like Google) reportedly only use part of disk for active data
  - Rest is archival data

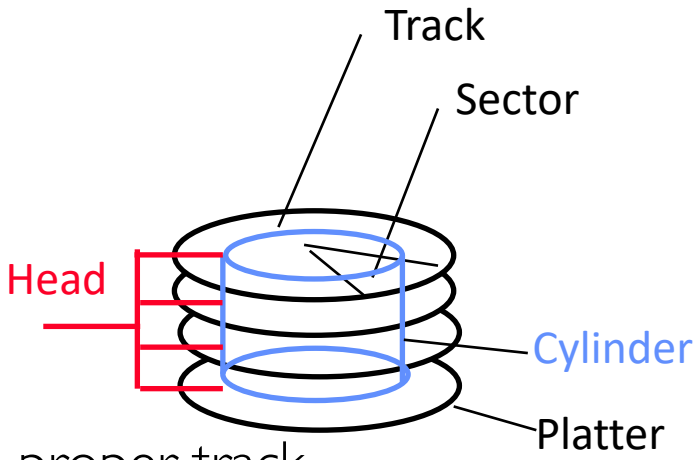


Shingled Magnetic Recording (SMR)



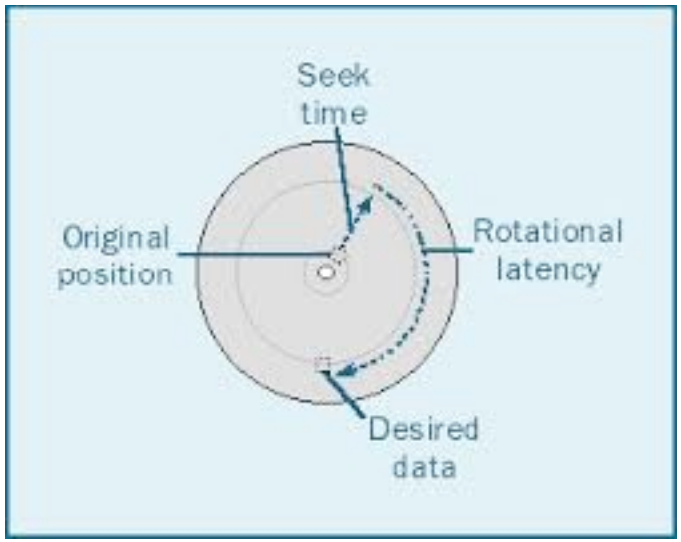
- Overlapping tracks yields greater density, capacity
- Restrictions on writing, complex DSP for reading
- Examples: Seagate (8TB), Hitachi (10TB)

Magnetic Disks - Performance



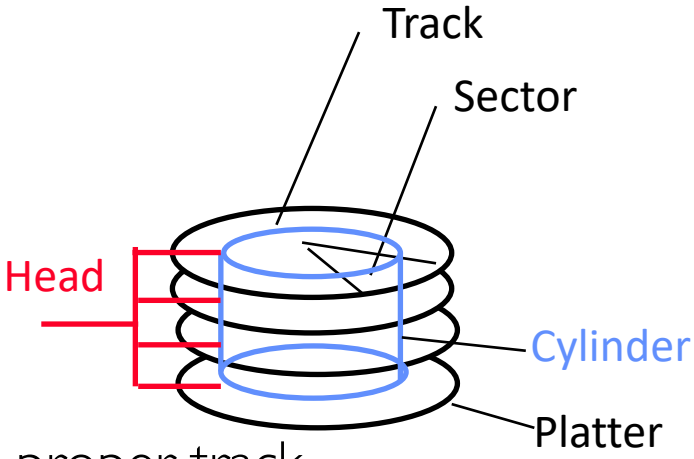
- Cylinders: all the tracks under the head at a given point on all surface

- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head



Seek time = 4-8ms
One rotation = 1-2ms
(3600-7200 RPM)

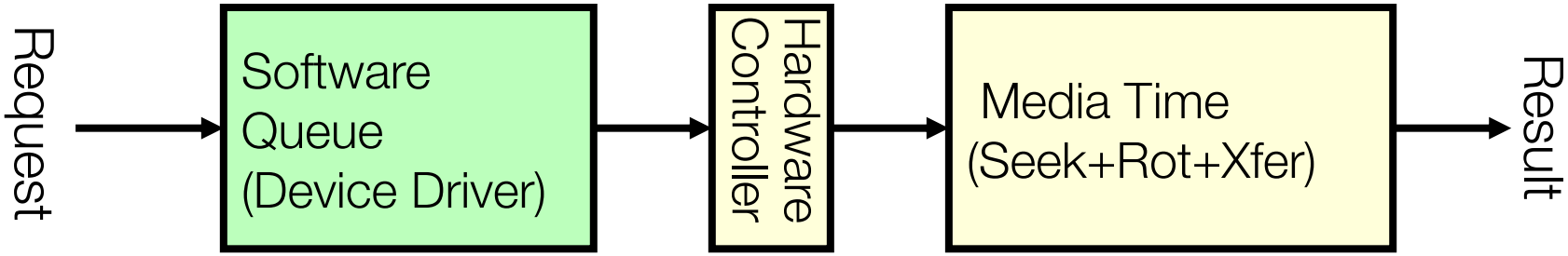
Magnetic Disks - Queuing



- Cylinders: all the tracks under the head at a given point on all surface

- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track
  - Rotational latency: wait for desired sector to rotate under r/w head
  - Transfer time: transfer a block of bits (sector) under r/w head

Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



Typical Numbers for Magnetic Disk

Parameter	Info / Range
Space/Density	Space: 14TB (Seagate), 8 platters, in 3½ inch form factor! Areal Density: **≥ 1Terabit/square inch! (PMR,** Helium, …)
Average seek time	Typically 4-6 milliseconds. Depending on reference locality, actual cost may be 25-33% of this number.
Average rotational latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 8-4 milliseconds
Controller time	Depends on controller hardware
Transfer time	Typically 50 to 250 MB/s. Depends on: • Transfer size (usually a sector): 512B – 1KB per sector • Rotation speed: 3600 RPM to 15000 RPM • Recording density: bits per inch on a track • Diameter: ranges from 1 in to 5.25 in
Cost	Used to drop by a factor of two every 1.5 years (or even faster); now slowing down

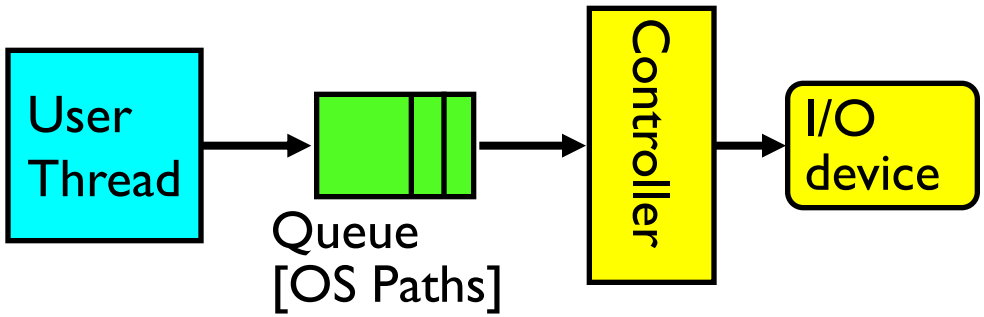
Disk Performance Example

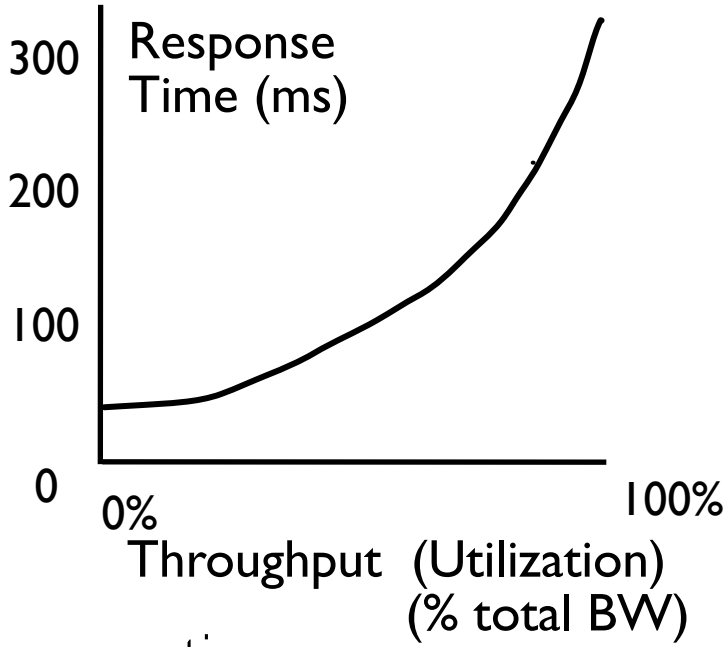
- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for rotation: 60000 (ms/min) / 7200(rev/min) ~= 8ms
  - Transfer rate of 50MByte/s, block size of 4Kbyte ⇒ 4096 bytes/50×10^6 (bytes/s) = 81.92×10^{-6} sec ≅ 0.082 ms for 1 sector
- Read block from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.082ms) = 9.082 ms
  - Approx 9ms to fetch/put data: 4096 bytes/9.082×10^{-3} s ≅ 450 KB/s
- Read block from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.082ms) = 4.082 ms
  - Approx 4ms to fetch/put data: 4096 bytes/4.082×10^{-3} s ≅ 1.0 MB/s
- Read next block on same track:
  - Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s ≅ 50MB/sec
- Key to using disk effectively (especially for file systems) is to minimize seek and rotational delays

(Lots of) Intelligence in the Controller

- 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
- …

I/O Performance





Response Time = Queue + I/O device service time

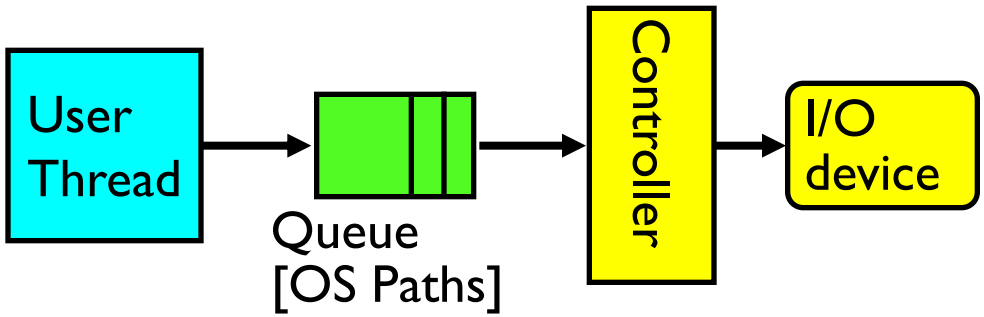
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)

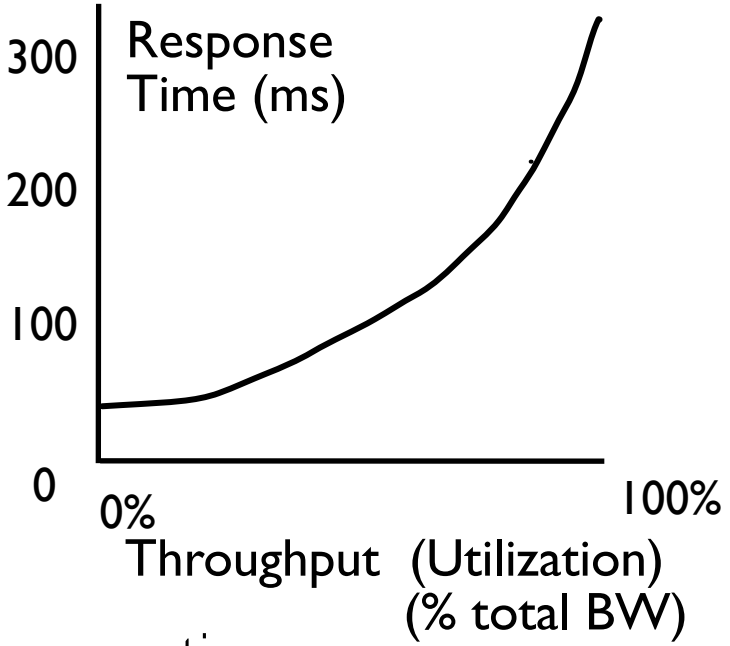
# of ops

time per op

Fixed overhead

I/O Performance





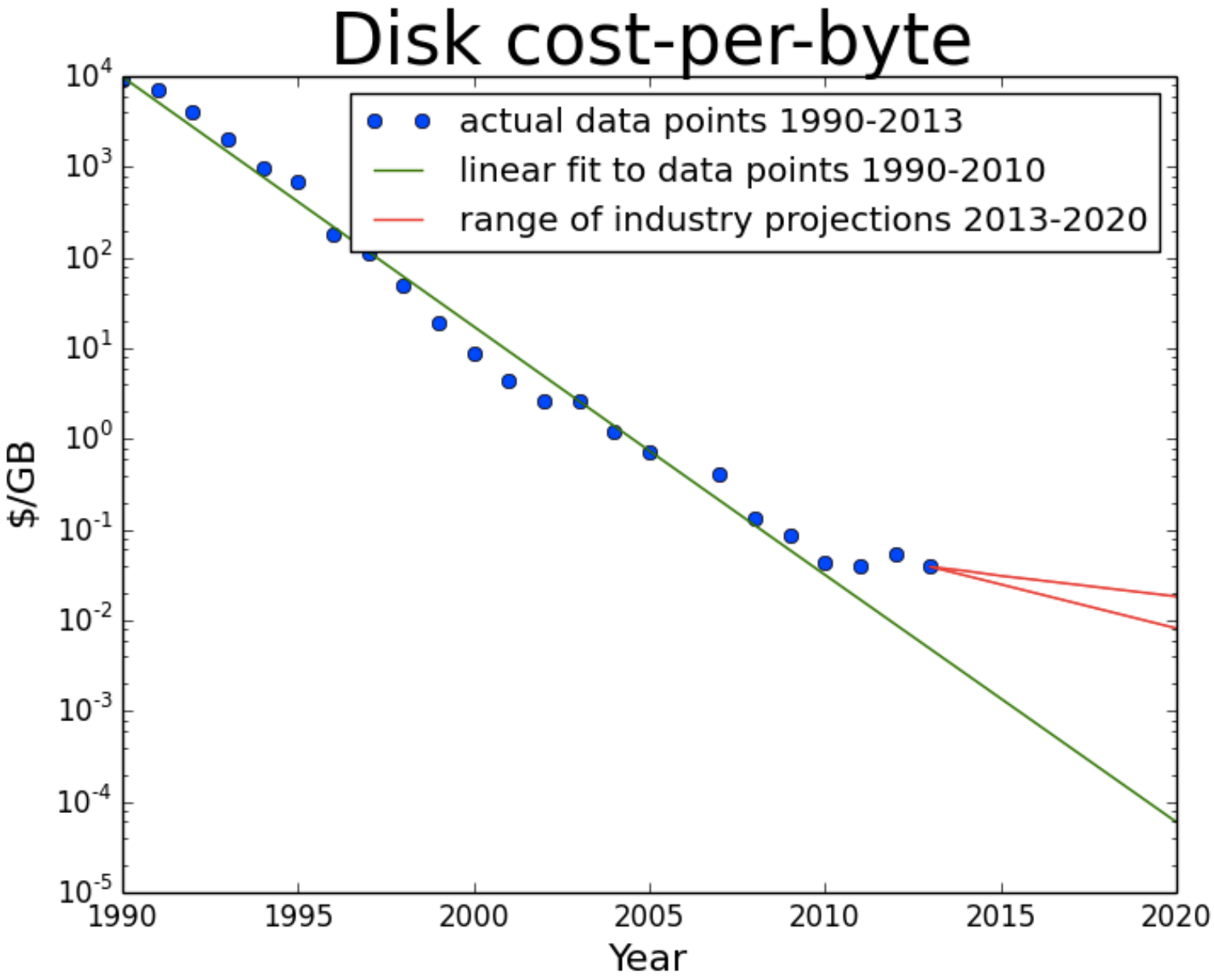
Response Time = Queue + I/O device service time

- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Effective BW per op = transfer size / response time
    - » EffBW(n) = n / (S + n/B) = B / (1 + SB/n)
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
    - » I/O device service time
- Queuing behavior:
  - Can lead to large increases of latency as utilization increases

I/O Scheduling Discussion

- What happens when two processes are accessing storage in different regions of the disk ?
- What can the driver do?
- How can buffering help?
- What about non-blocking I/O?
- Or threads with blocking I/O?
- What limits how much reordering the OS can do?

Hard Drive Prices over Time

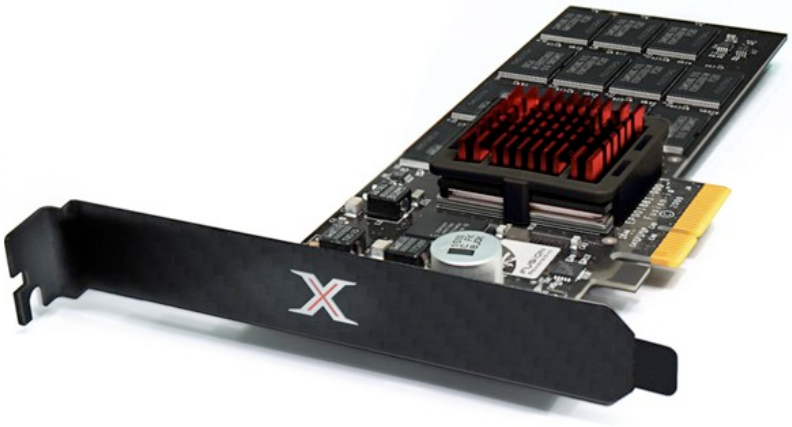


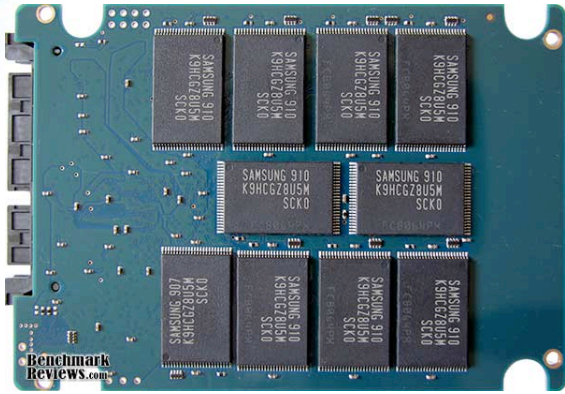
Example of Current HDDs

- Seagate Exos X14 (2018)
  - 14 TB hard disk
    - » 8 platters, 16 heads
    - » Helium filled: reduce friction and power
  - 4.16ms average seek time
  - 4096 byte physical sectors
  - 7200 RPMs
  - 6 Gbps SATA /12Gbps SAS interface
    - » 261MB/s MAX transfer rate
    - » Cache size: 256MB
  - Price: $615 (< $0.05/GB)

- IBM Personal Computer/AT (1986)
  - 30 MB hard disk
  - 30-40ms seek time
  - 0.7-1 MB/s (est.)
  - Price: $500 ($17K/GB, 340,000x more expensive !!)

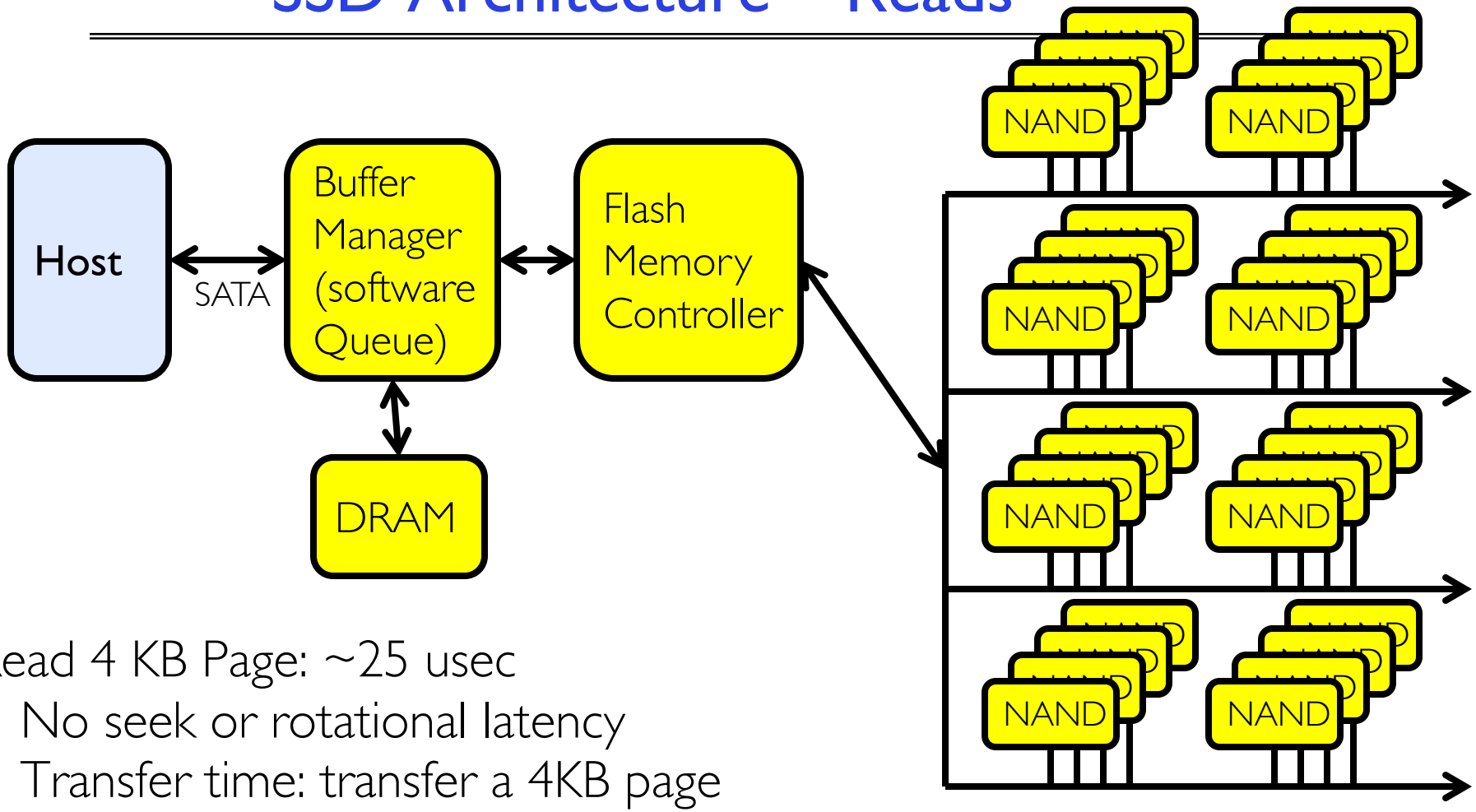
Solid State Disks (SSDs)





- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)
- 2009 – Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 "pages" per memory block
  - Trapped electrons distinguish between 1 and 0
- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited "write cycles"
- Rapid advances in capacity and cost ever since!

SSD Architecture – Reads

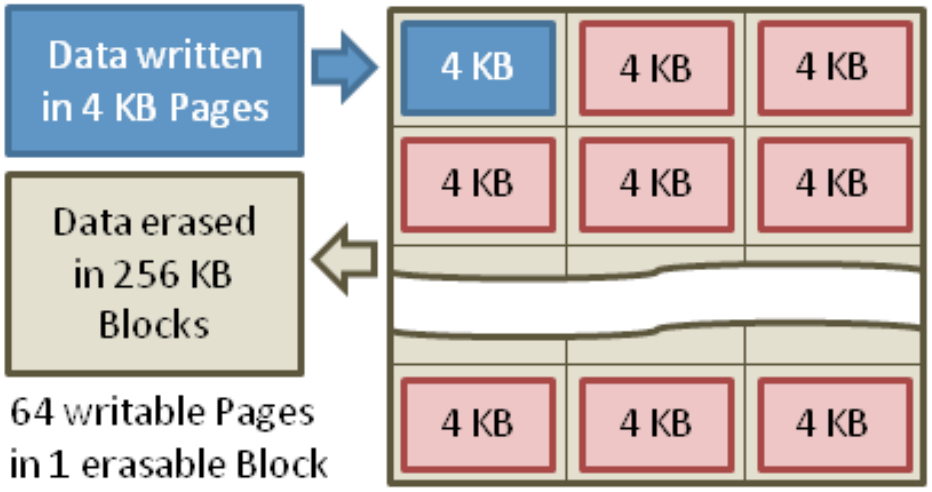


Read 4 KB Page: ~25 usec

- No seek or rotational latency
- Transfer time: transfer a 4KB page
  - » SATA: 300-600MB/s => ~4 x10^3 b / 400 x 10^6 bps => 10 us
- Latency = Queuing Time + Controller time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes

- Writing data is complex! (~200µs – 1.7ms)
  - Can only write empty pages in a block
  - Erasing a block takes ~1.5ms
  - Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes



Typical NAND Flash Pages and Blocks

https://en.wikipedia.org/wiki/Solid-state_drive

SSD Architecture - Writes

- SSDs provide same interface as HDDs to OS – read and write chunk (4KB) at a time
- But can only overwrite data 256KB at a time!
- Why not just erase and rewrite new version of entire 256KB block?
  - Erasure is very slow (milliseconds)
  - Each block has a finite lifetime, can only be erased and rewritten about 10K times
  - Heavily used blocks likely to wear out quickly

Solution – Two Systems Principles

1. Layer of Indirection
   - Maintain a *Flash Translation Layer (FTL)* in SSD
   - Map virtual block numbers (which OS uses) to physical page numbers (which flash mem. controller uses)
   - **Can now freely relocate data w/o OS knowing**
2. Copy on Write
   - Don't overwrite a page when OS updates its data
   - Instead, write new version in a free page
   - Update FTL mapping to point to new location

Flash Translation Layer

- No need to erase and rewrite entire 256KB block when making small modifications
- SSD controller can assign mappings to spread workload across pages
  - *Wear Levelling*

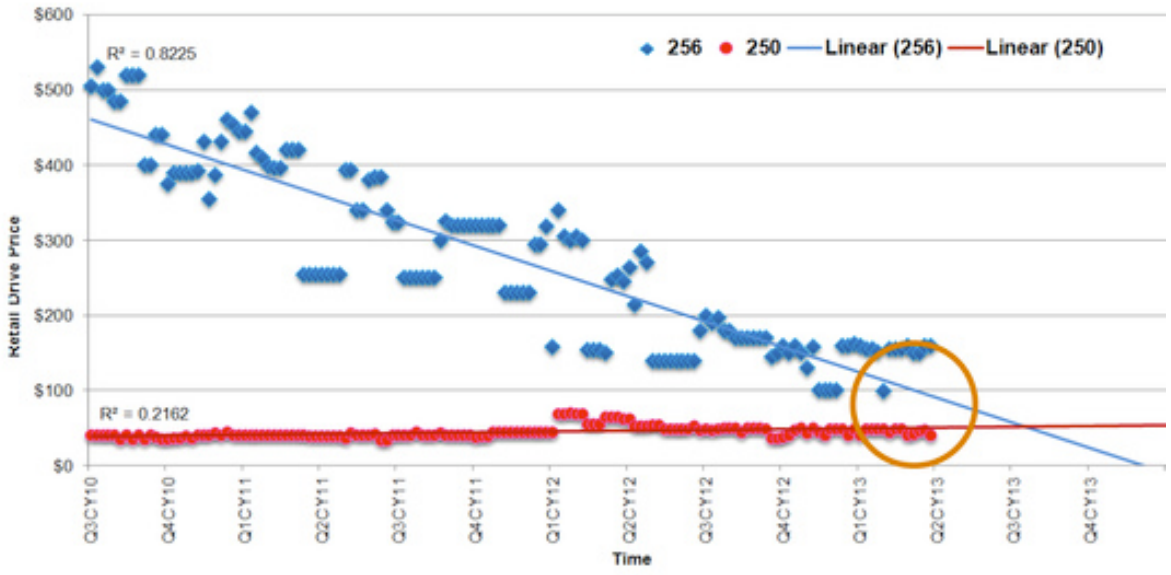
- What to do with old versions of pages?
  - *Garbage Collection* in background
  - Erase blocks with old pages, add to free list

Some "Current" 3.5in SSDs

- Seagate Nytro SSD: 15 TB (2017)
  - Dual 12Gb/s interface
  - Seq reads 860MB/s
  - Seq writes 920MB/s
  - Random Reads (IOPS): 102K
  - Random Writes (IOPS): 15K
  - Price (Amazon): $6325 ($0.41/GB)

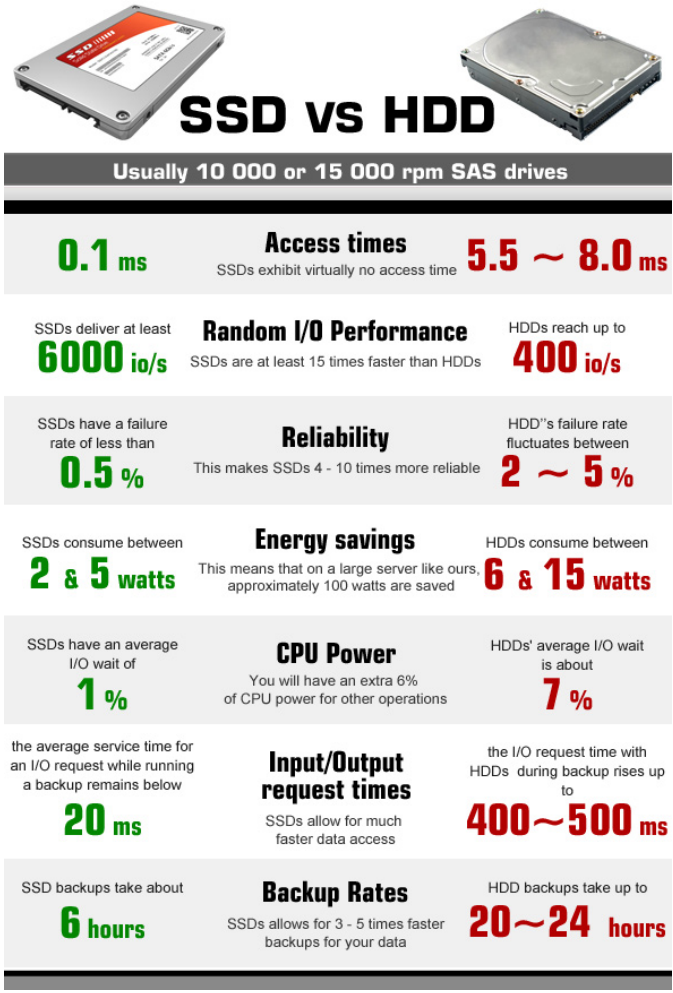
- Nimbus SSD: 100 TB (2019)
  - Dual port: 12Gb/s interface
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - *Unlimited writes for 5 years!*
  - Price: ~ $50K? ($0.50/GB)

HDD vs SSD Comparison



Price Crossover Point for HDD and SSD

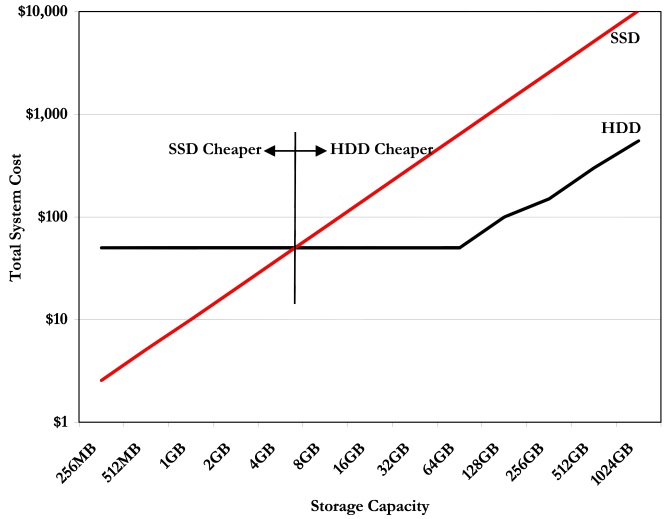
	2012	2013	2014	2015E	2016F	2017F
HDD	0.09	0.08	0.07	0.06	0.06	0.06
2.5" SSD	0.99	0.68	0.55	0.39	0.24	0.17



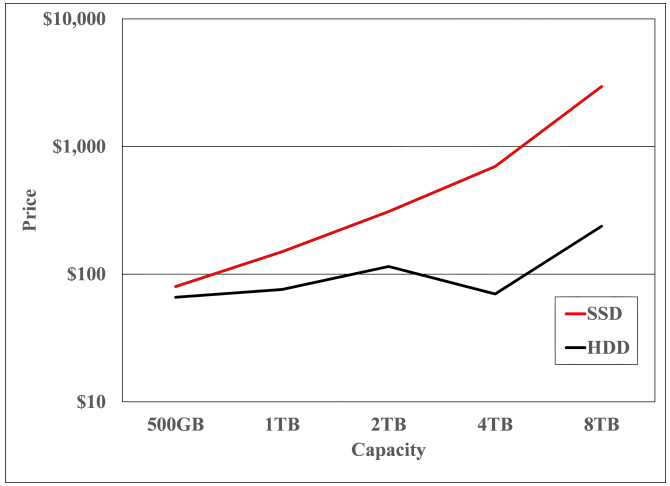
SSD prices drop much faster than HDD

SSD Summary

- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - Small storage (0.1-0.5x disk), expensive (3-20x disk)
    - » Hybrid alternative: combine small SSD with large HDD



2007 perspective (Storage Newsletter)



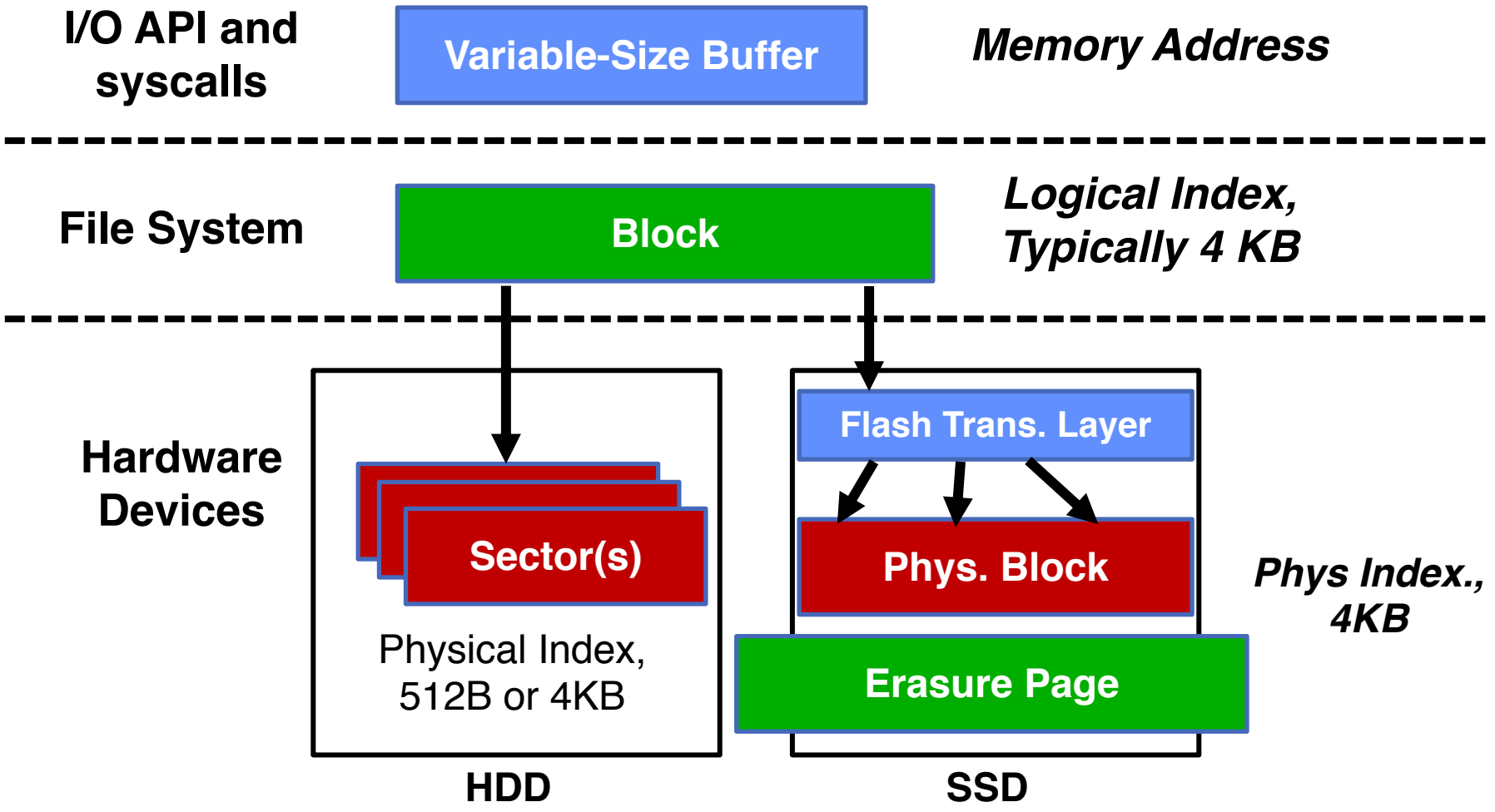
2019 perspective

SSD Summary

- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - ~~Small storage (0.1-0.5x disk)~~, expensive (3-20x disk)
    - » Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - » Controller garbage collection (GC) algorithms have major effect on performance
  - Limited drive lifetime
    - » 1-10K writes/page for MLC NAND
    - » Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly

No longer true!

From Storage to File Systems



User vs. System View of a File

- User's view:
  - Durable Data Structures
- System's view (system call interface):
  - Collection of Bytes (UNIX)
  - Oblivious to specific data structures user wants to store
- System's view (inside OS):
  - File is a collection of blocks

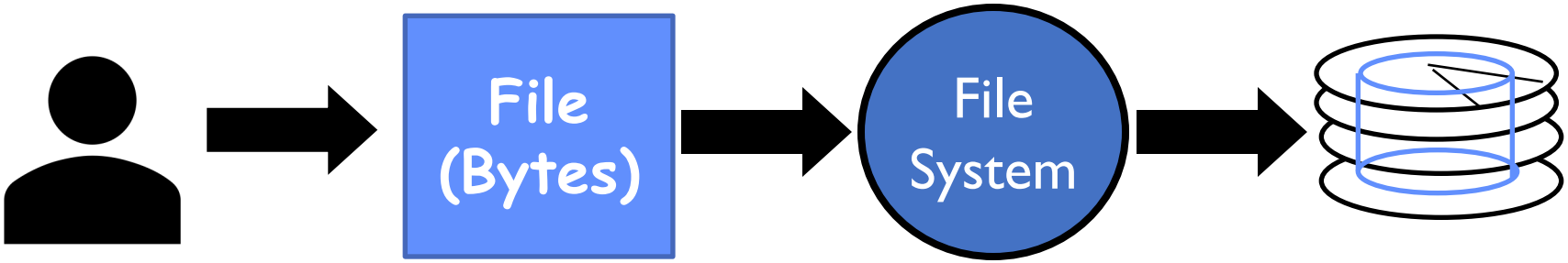
Building a File System

- Classic OS situation

  Take limited hardware interface (array of blocks) and provide a more convenient/useful interface with:

  – Naming: Find file by name, not block numbers
    » Organize file names with *directories*
  – Organization: Map files to blocks
  – Protection: Enforce access restrictions
  – Reliability: Keep files intact despite crashes, hardware failures, etc.

Translating from User to Sys. View



What happens if user says: "give me bytes 2 – 12?"

- Fetch block corresponding to those bytes
- Return just the correct portion of the block

- What about writing bytes 2 – 12?
  - Fetch block, modify relevant portion, write out block

Everything inside file system is in terms of whole-size blocks

- Actual disk I/O happens in blocks
- **`read`**/**`write`** smaller than block size needs to translate and buffer

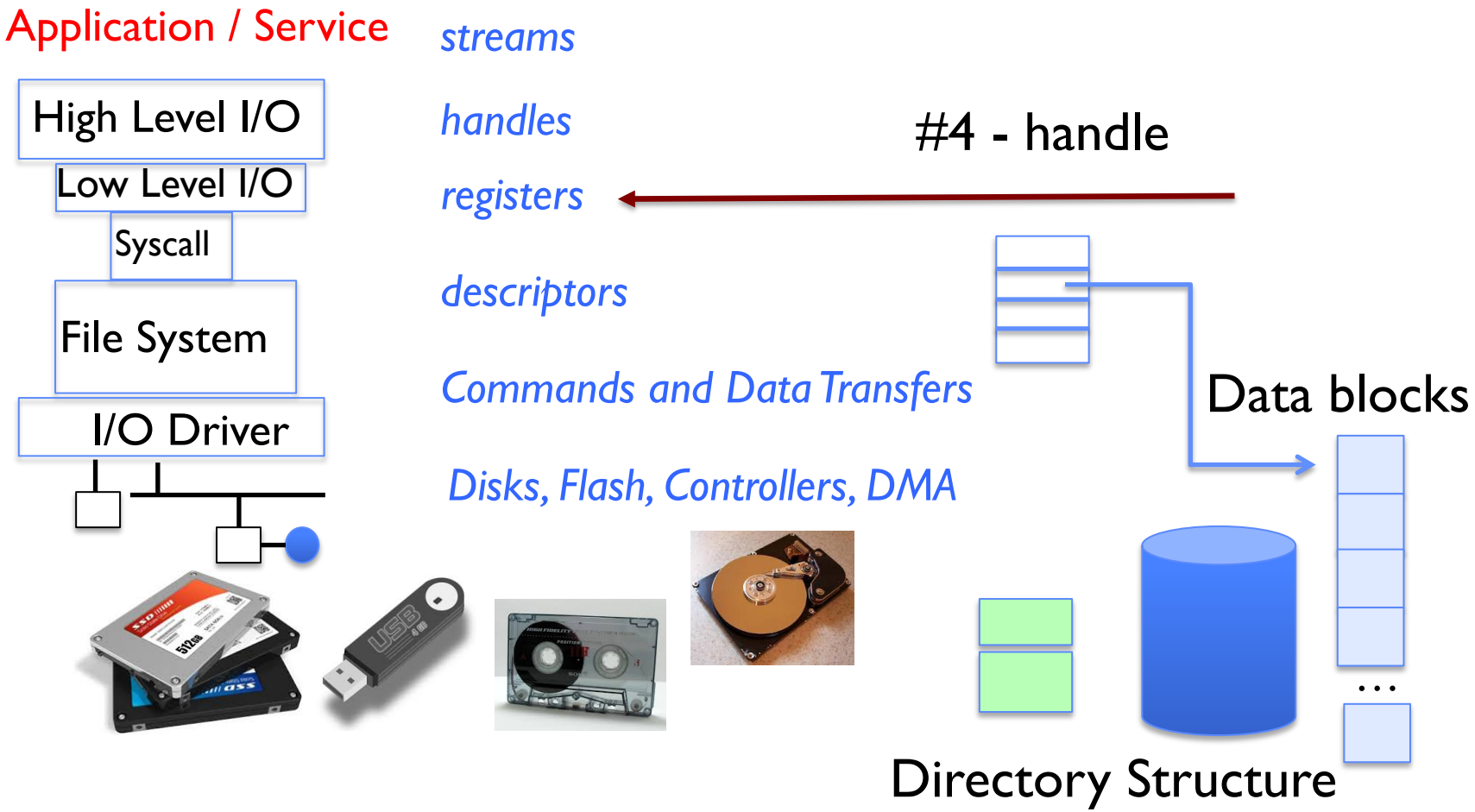
What does the file system need?

- Track free disk blocks
  - Need to know where to put newly written data
- Track which blocks contain data for which files
  - Need to know where to read a file from
- Track files in a directory
  - Find list of file's blocks given its name
- Where do we maintain all of this?
  - Somewhere on disk

Data Structures on Disk

- Bit different than data structures in memory
- Access a block at a time
  - Can't efficiently read/write a single word
  - Have to read/write full block containing it
  - Ideally want **sequential** access patterns
- Durability
  - Ideally, file system is in meaningful state upon shutdown
  - This obviously isn't always the case…

I/O & Storage Layers

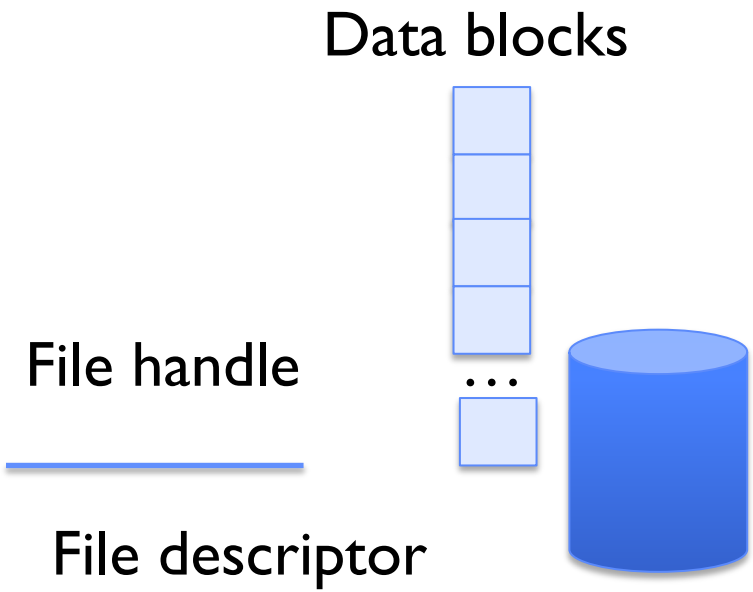


Designing a File System …

- What factors are critical to the design choices?
- Durable data store => it's all on disk
- (Hard) Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to allocate / free blocks
  - Such that access remains efficient

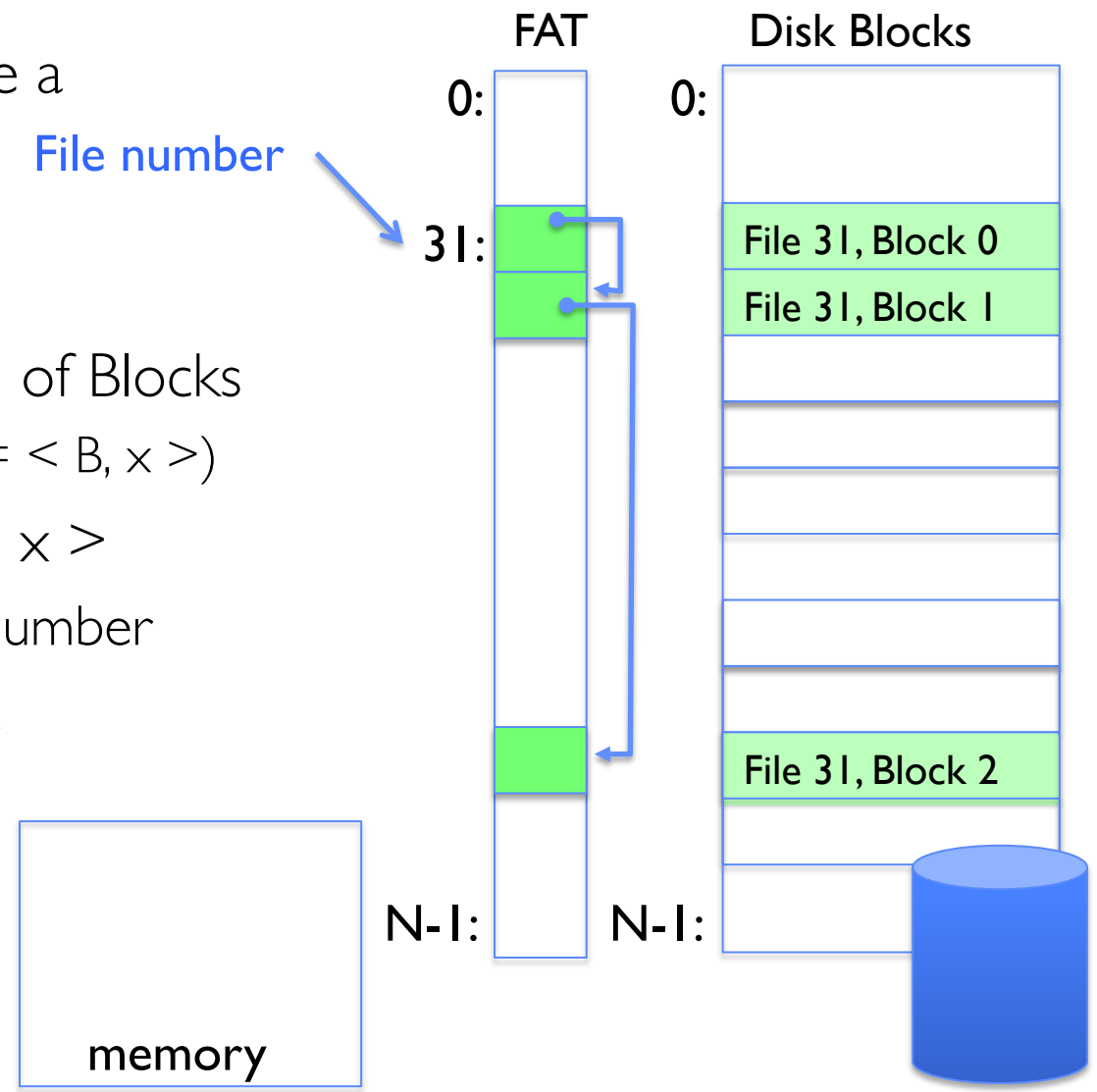
File

- Named permanent storage
- Contains
  - Data
    - » Blocks on disk somewhere
  - Metadata (Attributes)
    - » Owner, size, last opened, …
    - » Access rights
      - R, W, X
      - Owner, Group, Other (in Unix systems)
      - Access control list in Windows system



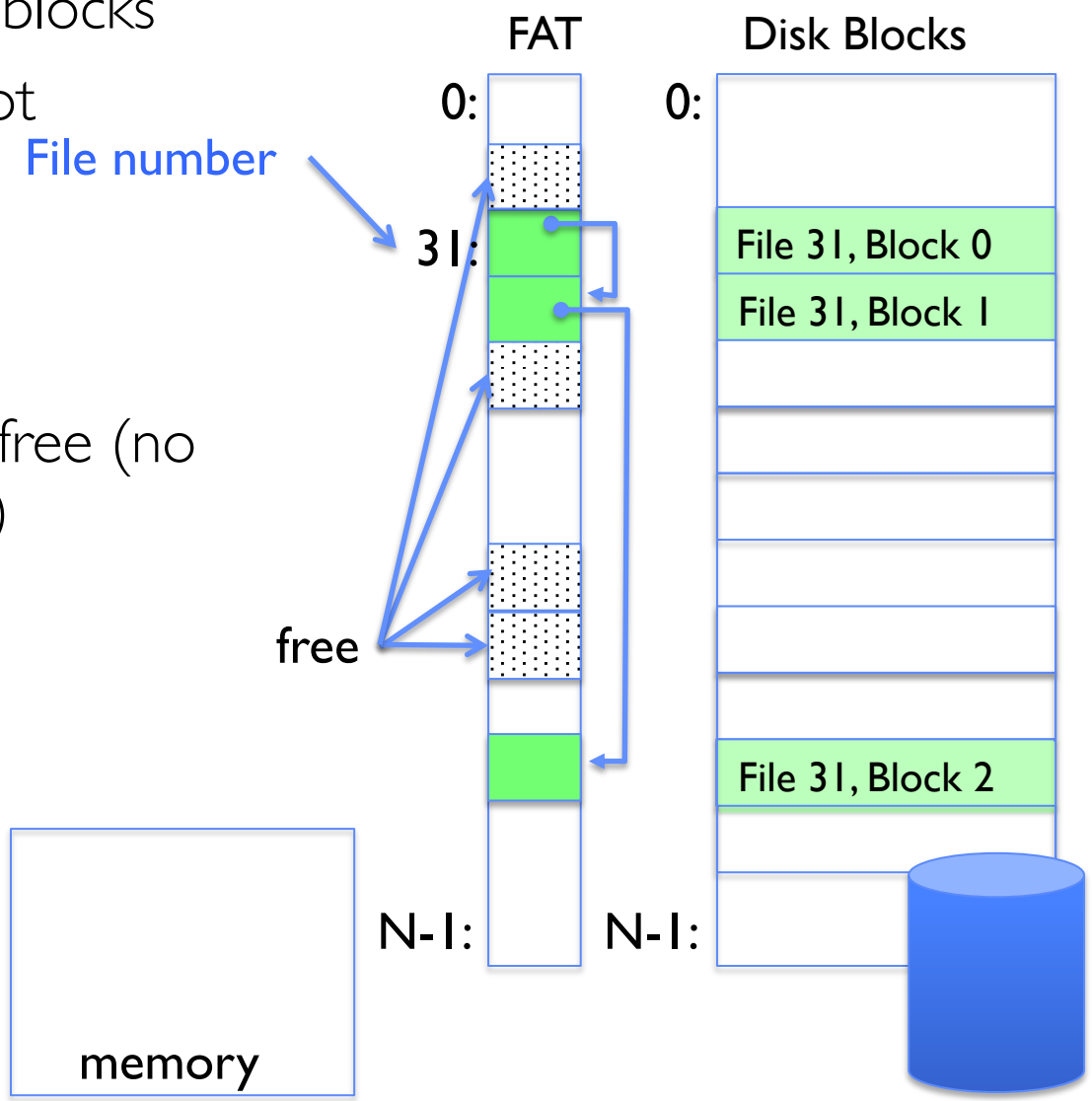
Our first filesystem: FAT (File Allocation Table)

- MS-DOS, 1977, still widely used filesystem (thumb drives, boot, …)
- Assume (for now) we have a way to translate a path to a "file number"
  - i.e., a directory structure
- Disk Storage is a collection of Blocks
  - Just hold file data (offset o = < B, x >)
- Example: file_read 31, < 2, x >
  - Index into FAT with file number
  - Follow linked list to block
  - Read the block from disk into memory



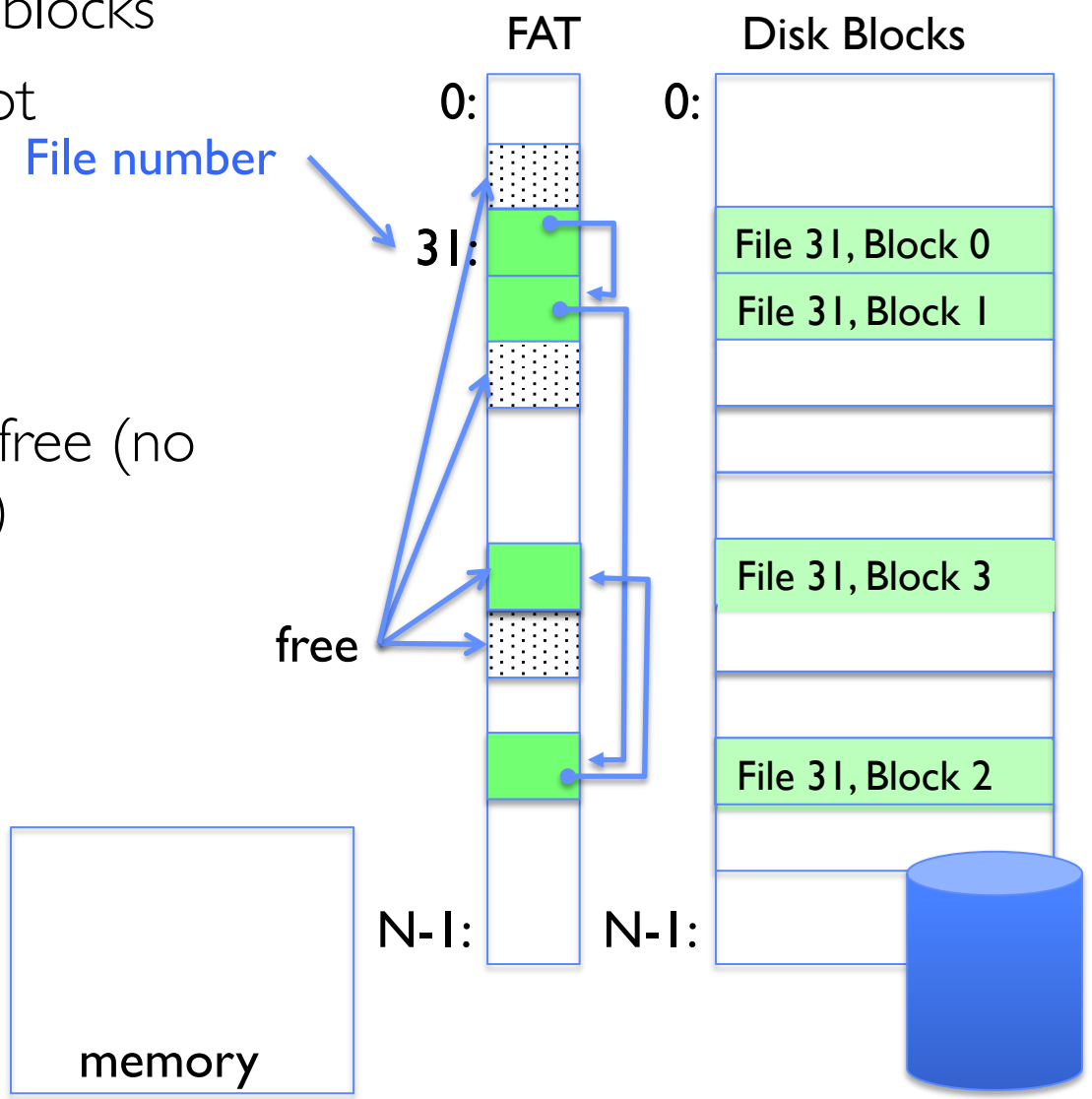
FAT Properties

- File is collection of disk blocks
- FAT is linked list 1–1 with blocks
- File Number is index of root of block list for the file
- File offset (o = < B, x >)
- Follow list to get block #
- Unused blocks ⇔ Marked free (no ordering, must scan to find)



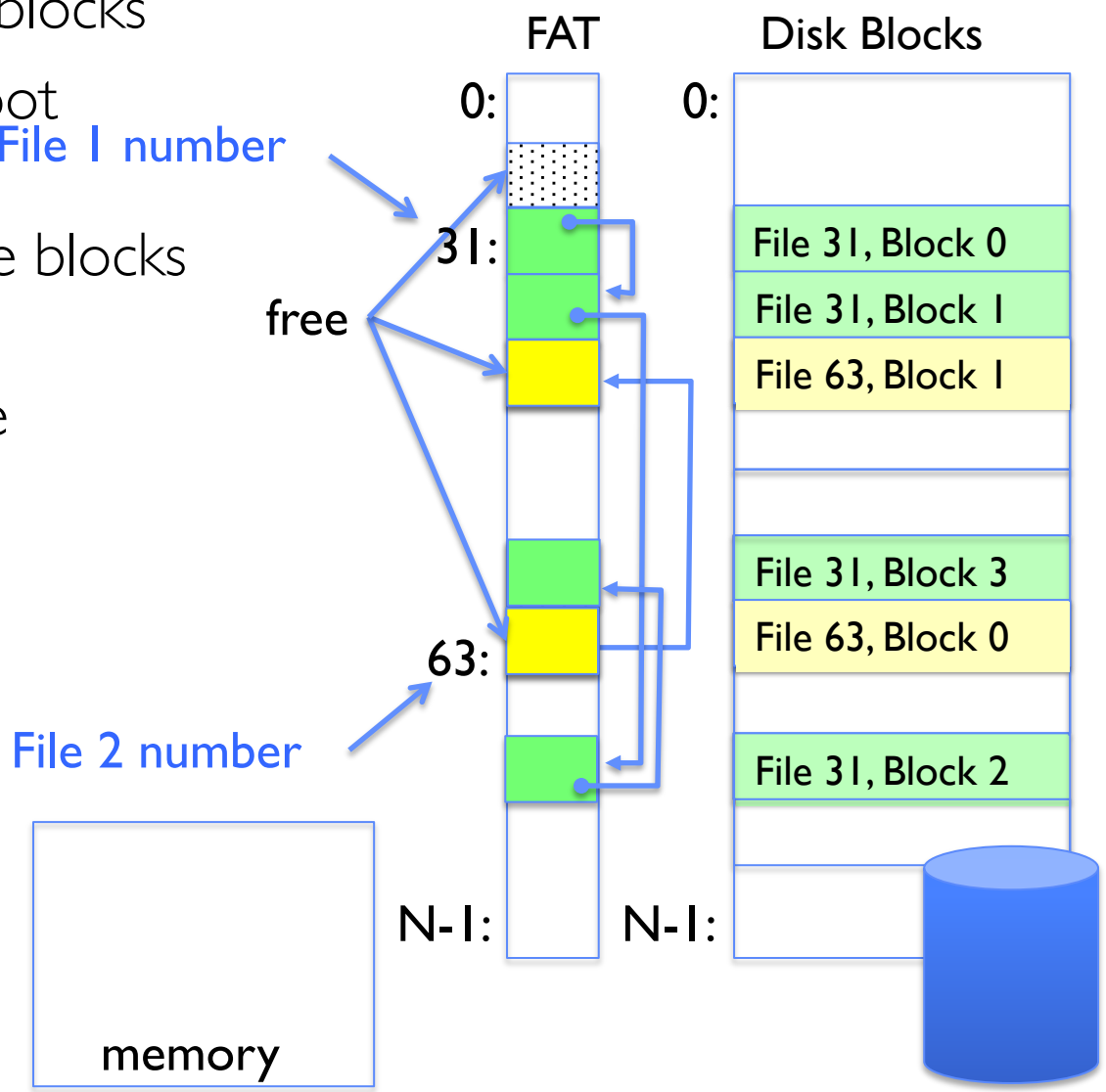
FAT Properties

- File is collection of disk blocks
- FAT is linked list **1–1** with blocks
- File Number is index of root of block list for the file
- File offset (o = < B, x >)
- Follow list to get block #
- Unused blocks ⇔ Marked free (no ordering, must scan to find)
- Ex: file_write(31, < 3, y >)
  - Grab free block
  - Linking them into file



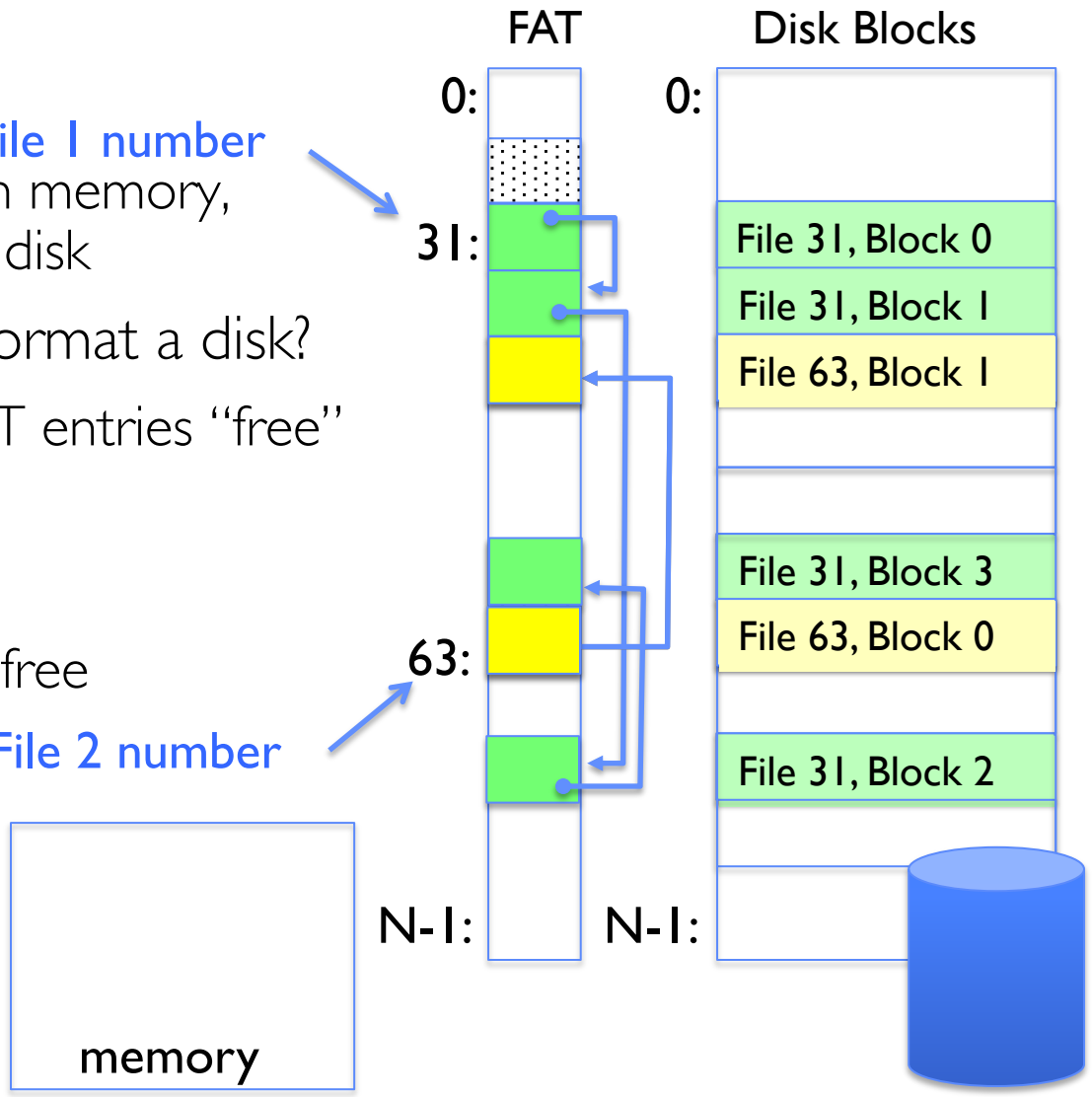
FAT Properties

- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- Grow file by allocating free blocks and linking them in
- Ex: Create file, write, write



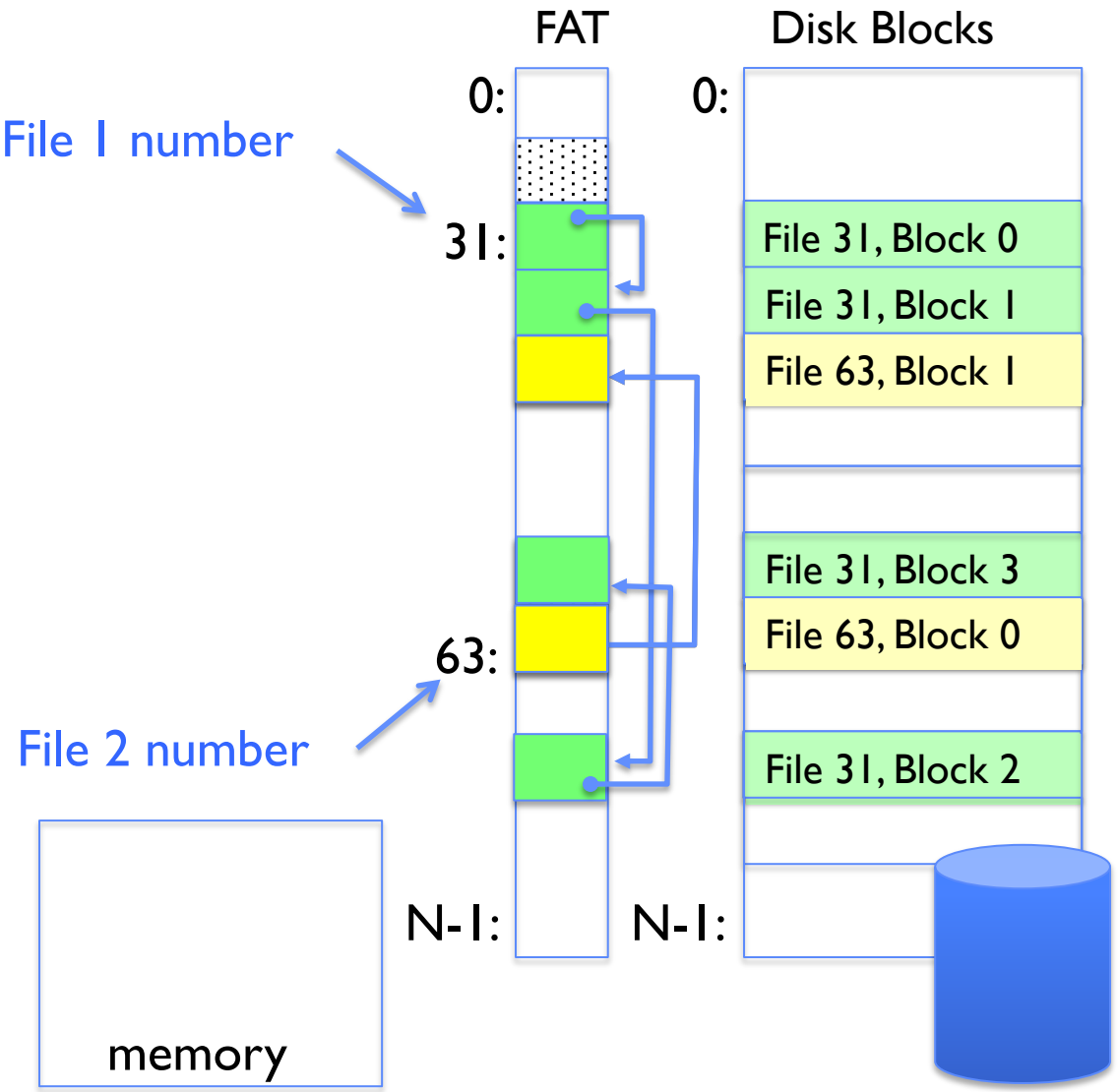
FAT Assessment

- *FAT32 (32 instead of 12 bits) used in Windows, USB drives, SD cards, …*
- Where is FAT stored?
  - On Disk, on boot cache in memory, second (backup) copy on disk
- What happens when you format a disk?
  - Zero the blocks, Mark FAT entries "free"
- What happens when you quick format a disk?
  - Mark all entries in FAT as free
- *Simple*
  - Can implement in device firmware

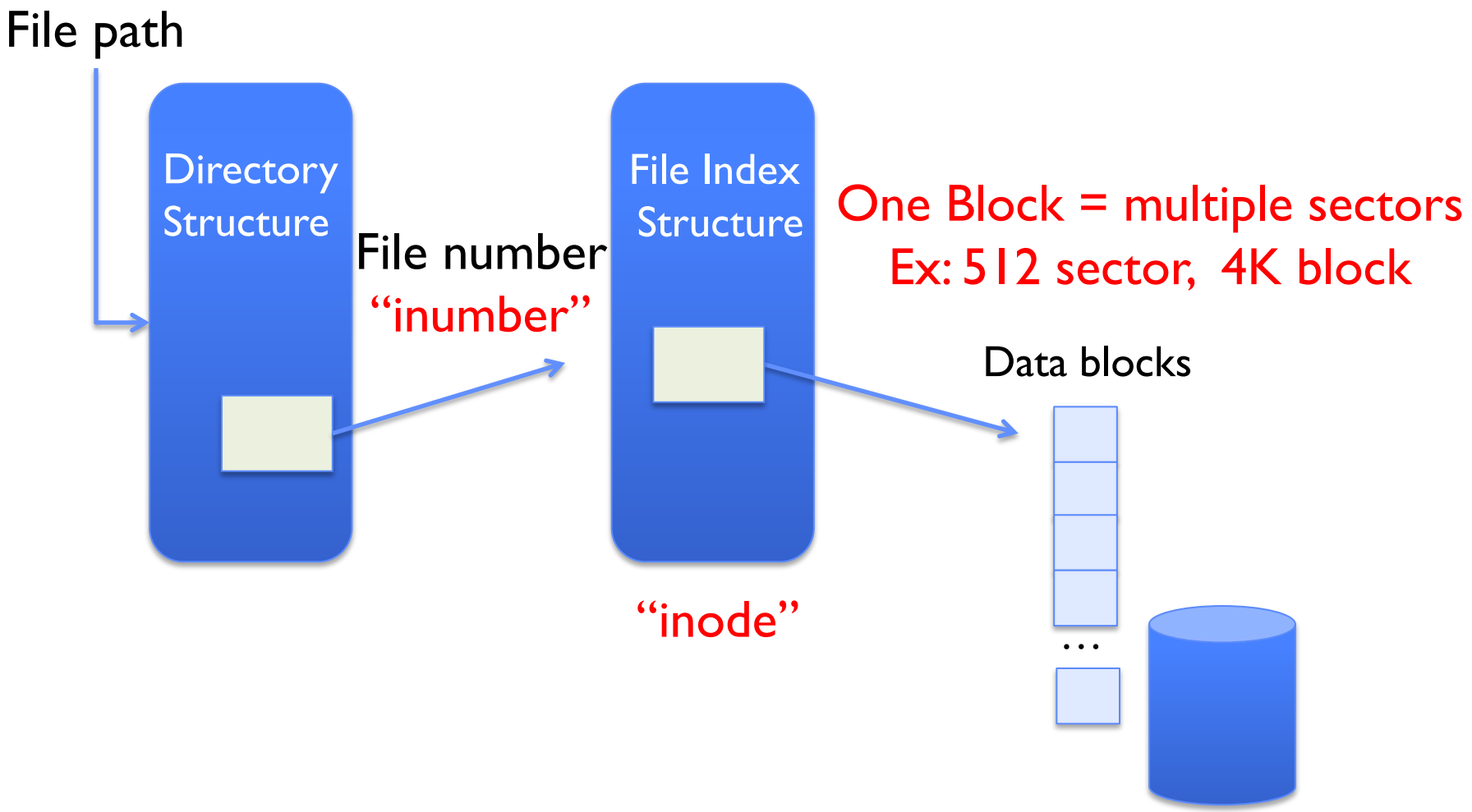


FAT Assessment – Issues

- Time to find block (large files) ??
- Block layout for file ???
- Sequential Access ???
- Random Access ???
- Fragmentation ???
  - MSDOS defrag tool
- Small files ???
- Big files ???



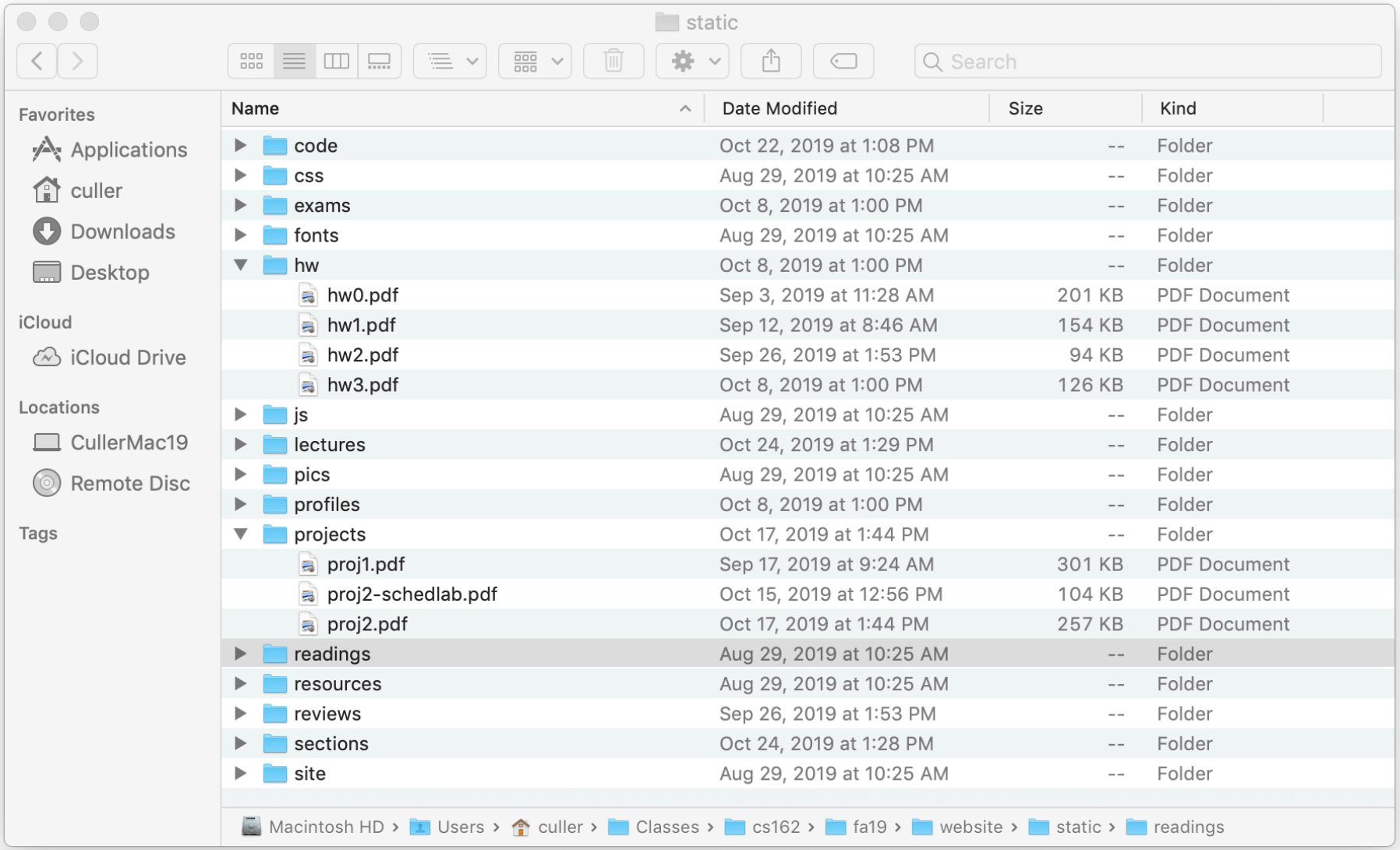
Components of a File System



Components of a file system

- Open performs *Name Resolution*
  - Translates pathname into a "file number"
    - » Used as an "index" to locate the blocks
  - Creates a file descriptor in PCB within kernel
  - Returns a "handle" (another integer) to user process
- Read, Write, Seek, and Sync operate on handle
  - Mapped to file descriptor and to blocks
- 4 components: directory, index structure, blocks, free space map

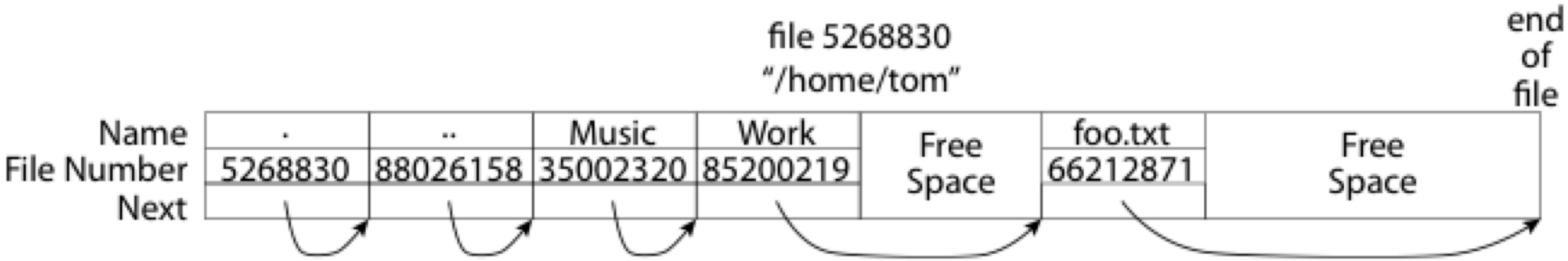
Directories



Directory

- Basically a hierarchical structure
- Each directory entry is a collection of
  - Files
  - Directories
    - » links to other entries
- Each has a name and attributes
  - Files have data
- Links (hard links) make it a DAG, not just a tree
  - Softlinks (aliases) are another name for an entry

What about the Directory - FAT?



- Essentially a file containing
  <file_name: file_number> mappings
- Free space for new entries
- In FAT: file attributes are kept in directory (!!!)
- Each directory a linked list of entries
- Where do you find root directory ("/")?

FAT Directory Structure (cont'd)

- How many disk accesses to resolve "/my/book/count"?
  - Read in file header for root (fixed spot on disk)
  - Read in first data block for root
    - » Table of file name/index pairs. Search linearly – ok since directories typically very small
  - Read in file header for "my"
  - Read in first data block for "my"; search for "book"
  - Read in file header for "book"
  - Read in first data block for "book"; search for "count"
  - Read in file header for "count"
- Current working directory: Per-address-space pointer to a directory used for resolving file names
  - Allows user to specify relative filename instead of absolute path (say CWD="/my/book" can resolve "count")

Many Huge FAT Security Holes!

- FAT has no access rights
- FAT has no header in the file blocks
- Just gives an index into the FAT
  - (file number = block number)

Summary (1/3) - Devices

- I/O Devices Types:
  - Many different speeds (0.1 bytes/sec to GBytes/sec)
  - Different Access Patterns:
    - » Block Devices, Character Devices, Network Devices
  - Different Access Timing:
    - » Blocking, Non-blocking, Asynchronous
- I/O Controllers: Hardware that controls actual device
  - Processor Accesses through I/O instructions, load/store to special physical memory
- Notification mechanisms
  - Interrupts
  - Polling: Report results through status register that processor looks at periodically
- Device drivers interface to I/O devices
  - Provide clean Read/Write interface to OS above
  - Manipulate devices through PIO, DMA & interrupt handling
  - Three types: block, character, and network

Summary (2/3) - Storage

- Disk Performance:
  - Queuing time + Controller + Seek + Rotational + Transfer
  - Rotational latency: on average ½ rotation
  - Transfer time: spec of disk depends on rotation speed and bit storage density
- Devices have complex interaction and performance characteristics
  - Response time (Latency) = Queue + Overhead + Transfer
    - » Effective BW = BW * T/(S+T)
  - HDD: Queuing time + controller + seek + rotation + transfer
  - SDD: Queuing time + controller + transfer (erasure & wear)
- Systems (e.g., file system) designed to optimize performance and reliability
  - Relative to performance characteristics of underlying device
- Bursts & High Utilization introduce queuing delays

Summary (3/3) - File Systems

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for size, access and usage patterns
  - Maximize sequential access, allow efficient random access
  - Projects the OS protection and security regime (UGO vs ACL)
- Naming: translating from user-visible names to actual sys resources
  - Directories provide naming for local file systems
  - Linked or tree structure stored in files
- Components: directory, index, storage blocks, free list
- File Allocation Table (FAT) – simple and primitive
  - I-number = FAT index, FAT **1–1** with disk blocks, file is singly-link list of FAT=Blocks, directory is essentially file of <name, index, attributes> at known location
  - Linear search – for block, for i-number