CS162
Operating Systems and Systems Programming
Lecture 20

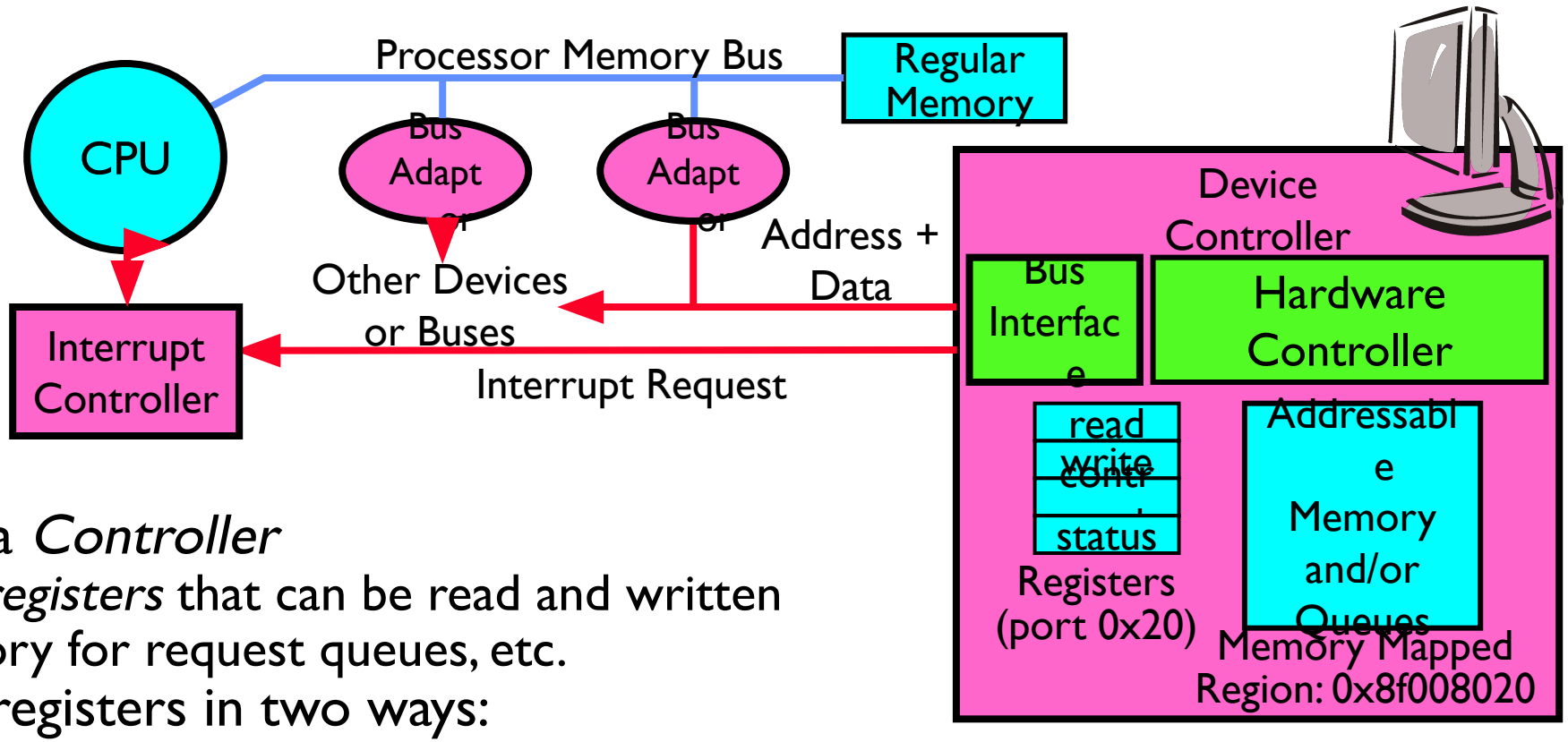
Device Drivers (finish), Storage Devices

November 12th, 2024

Prof. Ion Stoica

http://cs162.eecs.Berkeley.edu

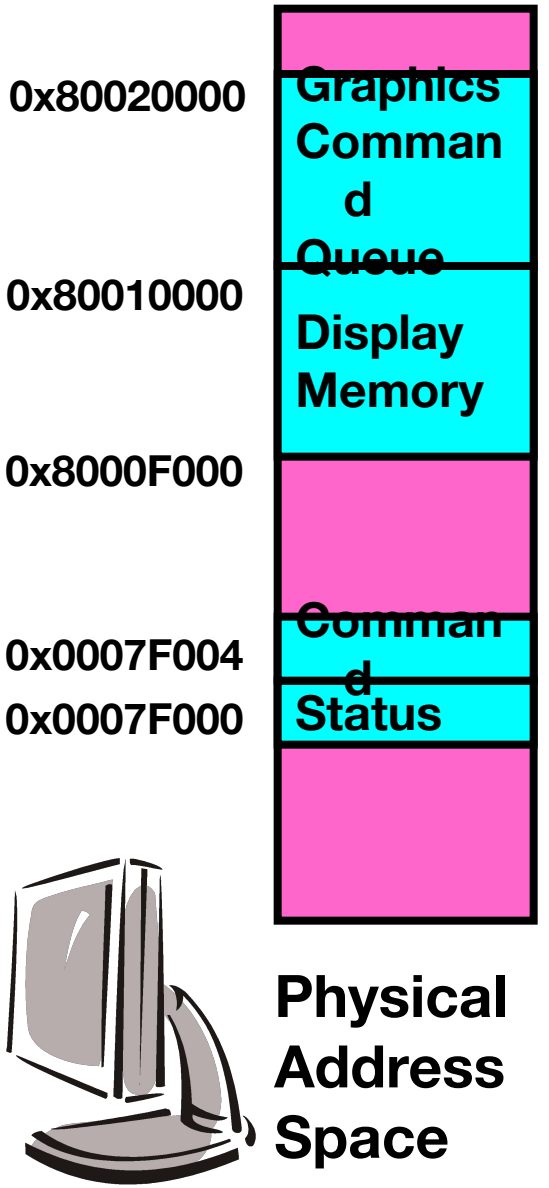
Recall: How does the Processor Talk to the Device?



- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues, etc.
- Processor accesses registers in two ways:
  - Port-Mapped I/O: in/out instructions
    - » Example from the Intel architecture: `out 0x21,AL`
  - Memory-mapped I/O: load/store instructions
    - » Registers/memory appear in physical address space
    - » I/O accomplished with load and store instructions

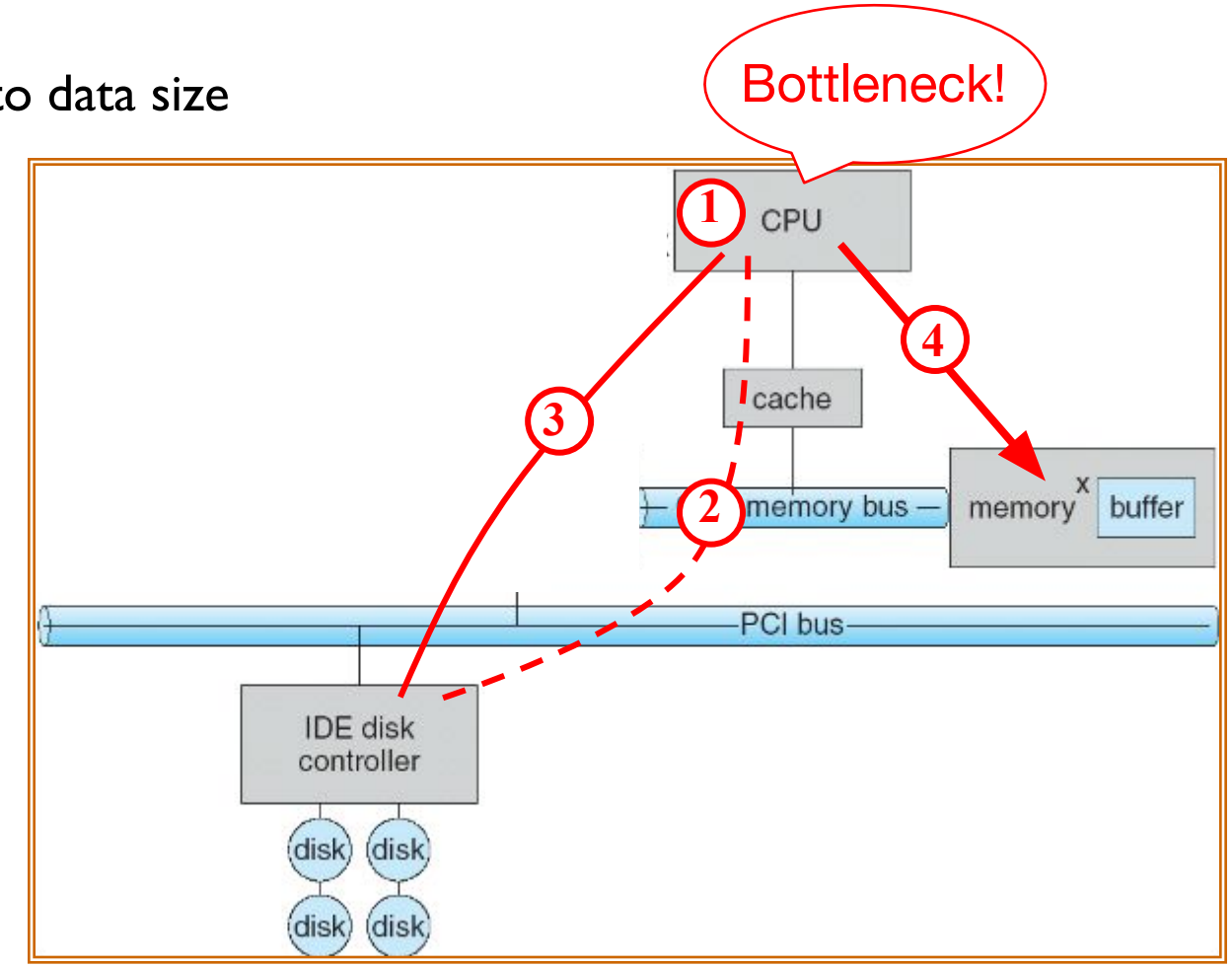
Example Memory-Mapped Display Controller

- 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



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

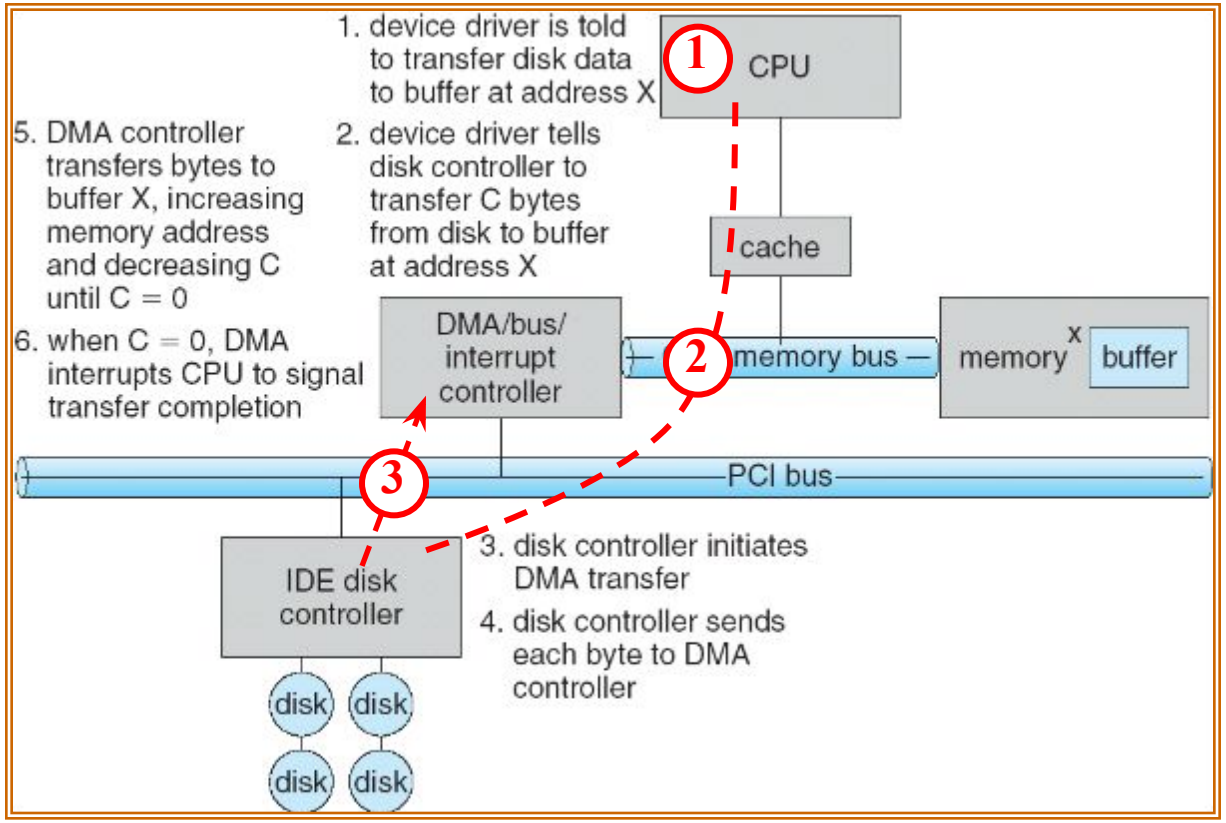


Transferring Data To/From Controller

- 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):

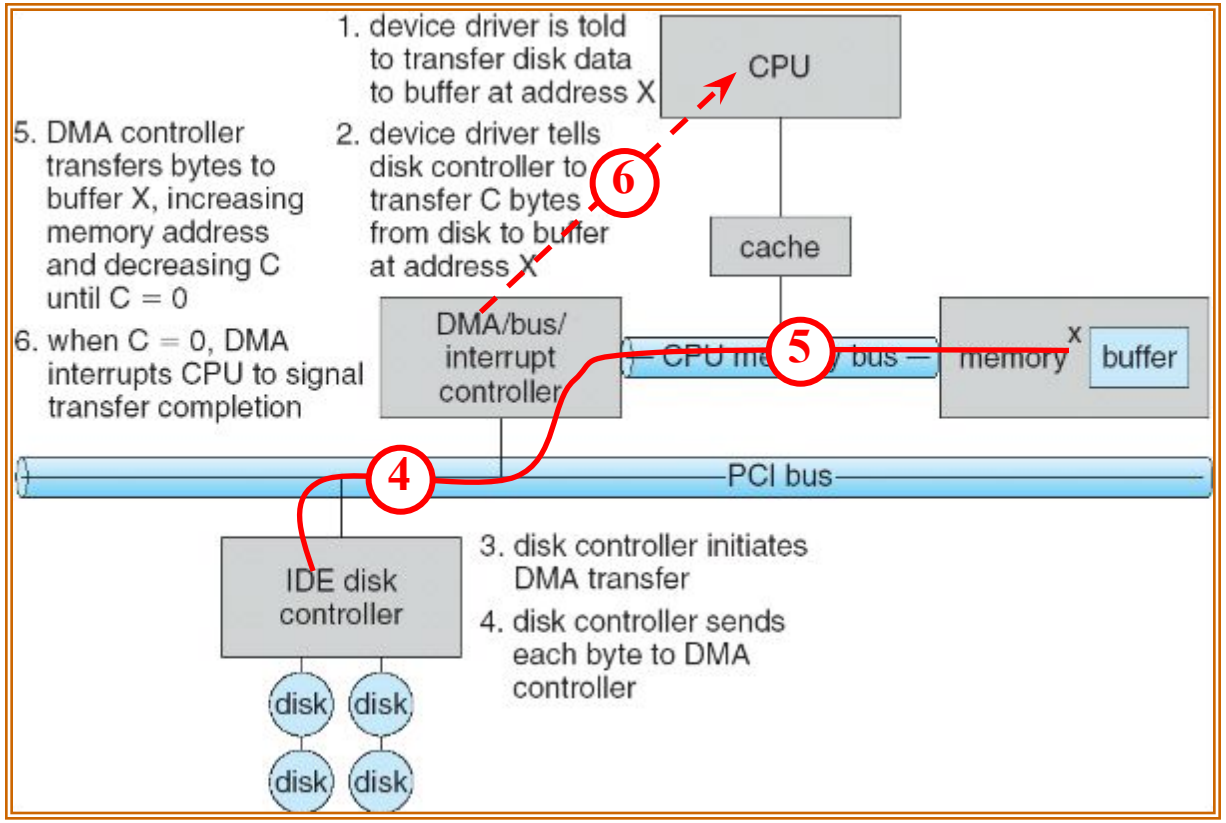


Transferring Data To/From Controller

- 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

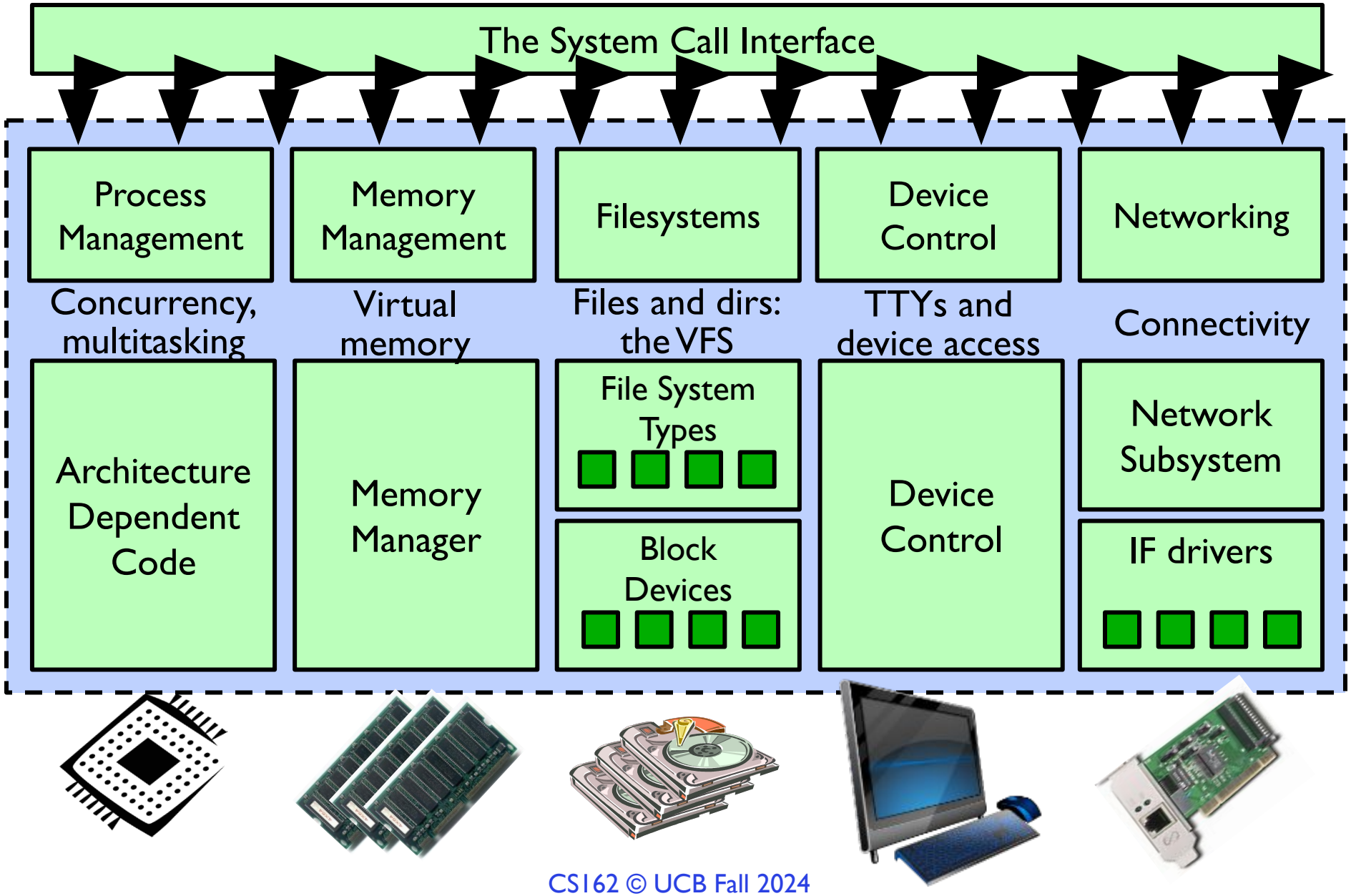
- Sample interaction with DMA controller (from OSC book):



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

Kernel Device Structure



Recall: Want Standard Interfaces to Devices

- Block Devices: *e.g.* disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Commands include `open()`, `read()`, `write()`, `seek()`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character Devices: *e.g.* keyboards, mice, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing
- Network Devices: *e.g.* Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface
  - Unix and Windows include socket interface
    - » Separates network protocol from network operation
    - » Includes `select()` functionality
  - Usage: pipes, FIFOs, streams, queues, mailboxes

How Does User Deal with Timing?

- Blocking Interface: "Wait"
  - When request data (e.g. `read()` system call), put process to sleep until data is ready
  - When write data (e.g. `write()` system call), put process to sleep until device is ready for data
- Non-blocking Interface: "Don't Wait"
  - Returns quickly from read or write request with count of bytes successfully transferred
  - Read may return nothing, write may write nothing
- Asynchronous Interface: "Tell Me Later"
  - When request data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
  - When send data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

Storage Devices

- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access (except for SMR – later!)
  - Slow performance for random access
  - Better performance for sequential access

- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Wear patterns issue

Ways of Measuring Performance

Latency - time to complete a task

Measured in units of time (s, ms, us, …, hours, years)

Throughput or *Bandwidth* – rate at which tasks are performed

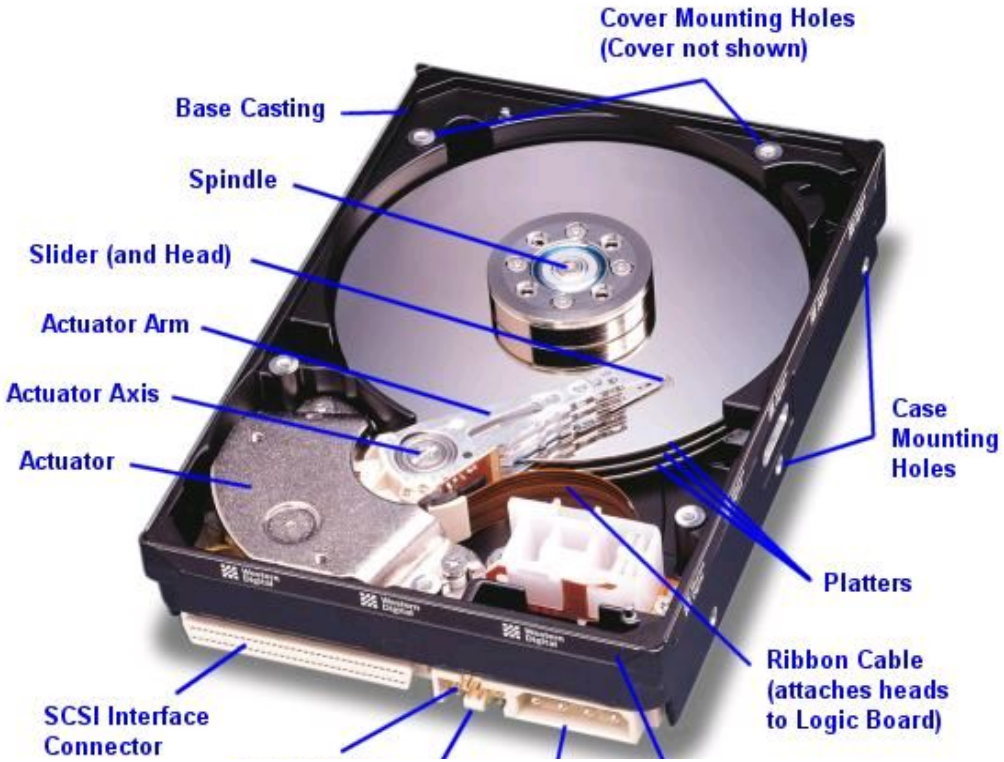
Measured in units of things per unit time (ops/s, GFLOP/s)

Start up or Overhead – time to initiate an operation

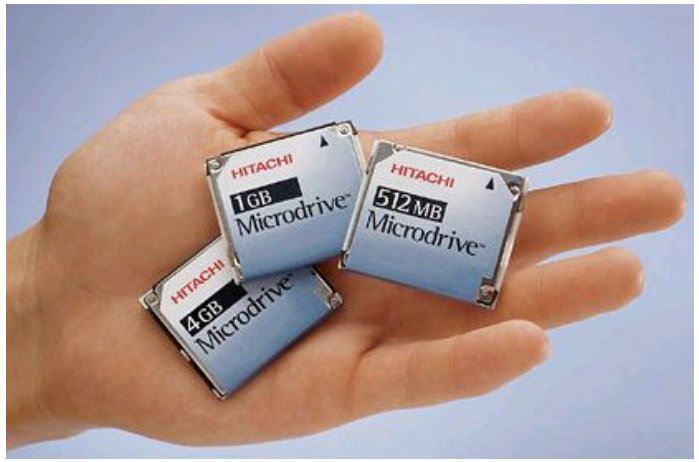
Most I/O operations are roughly linear in *b* bytes

– Latency(b) = Overhead + b/TransferCapacity

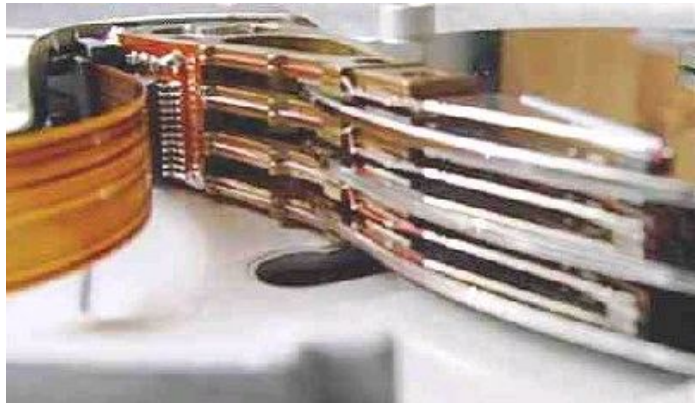
Hard Disk Drives (HDDs)



Western Digital Drive
http://www.storagereview.com/guide/



IBM/Hitachi Microdrive

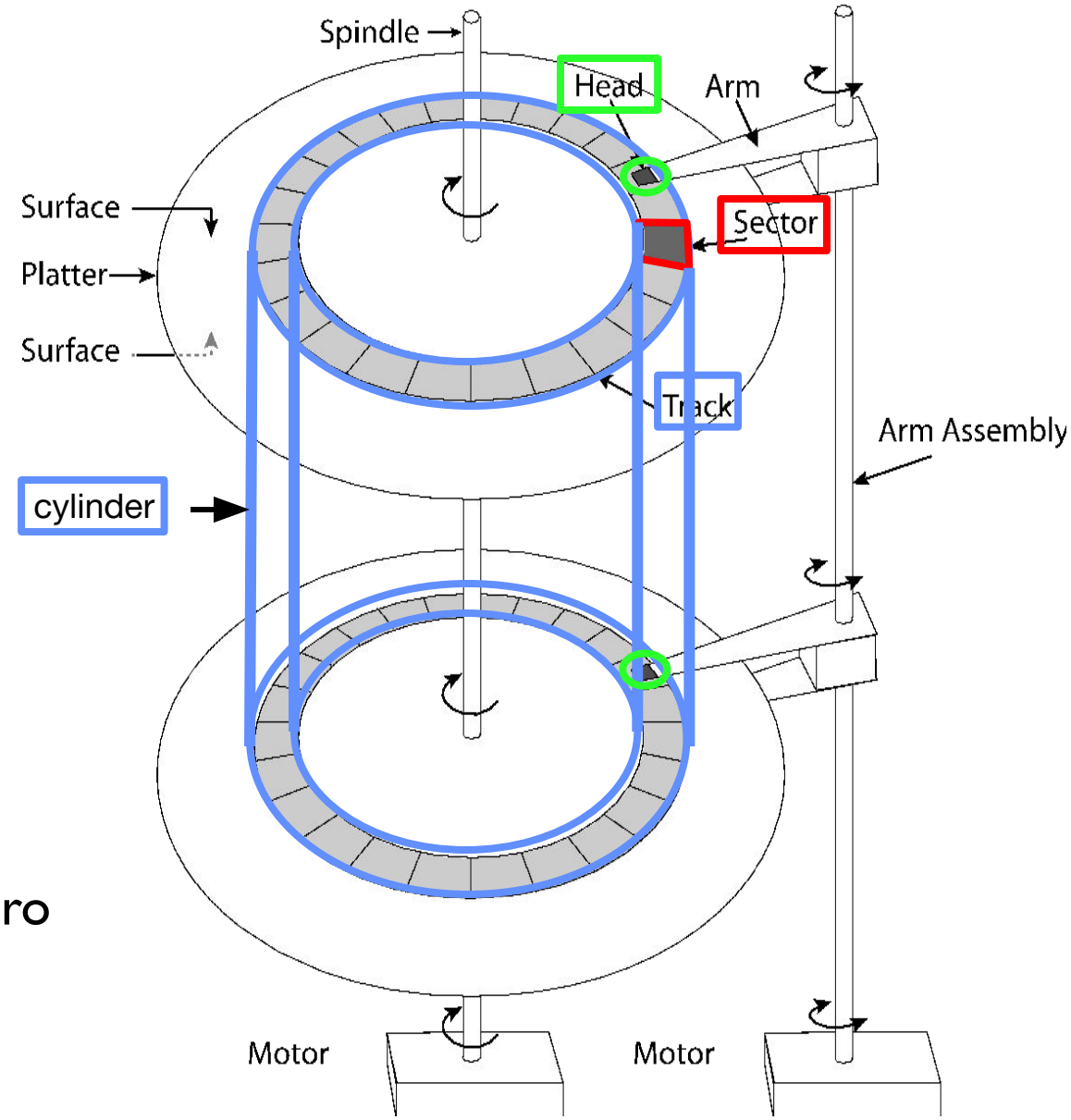


Read/Write Head
Side View

IBM Personal Computer/AT (1986)
- 30 MB hard disk - $500
- 30-40ms seek time
- 0.7-1 MB/s (est.)

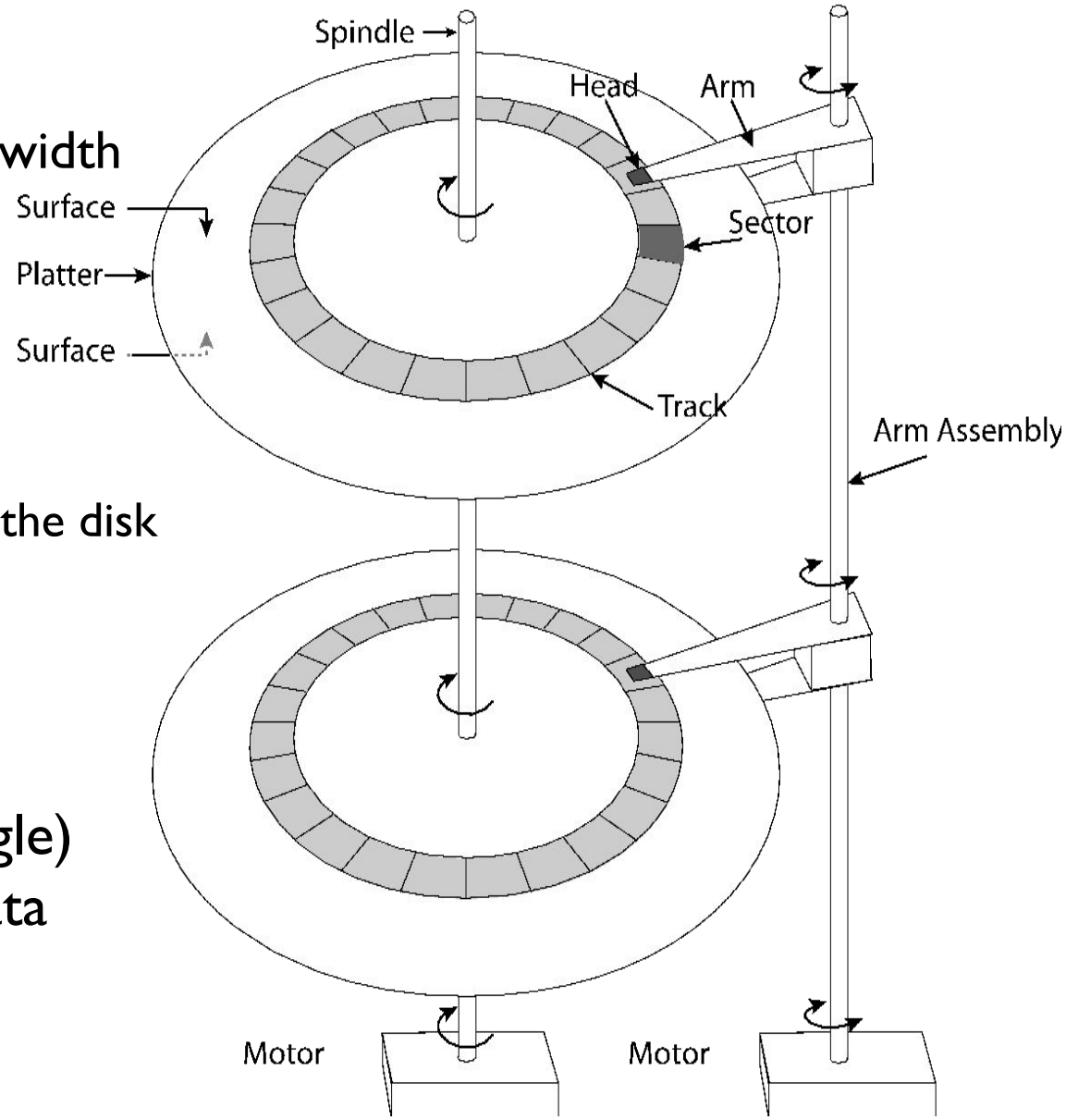
The Amazing Magnetic Disk

- Unit of Transfer: **Sector** (512B or 4096B)
  - 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)



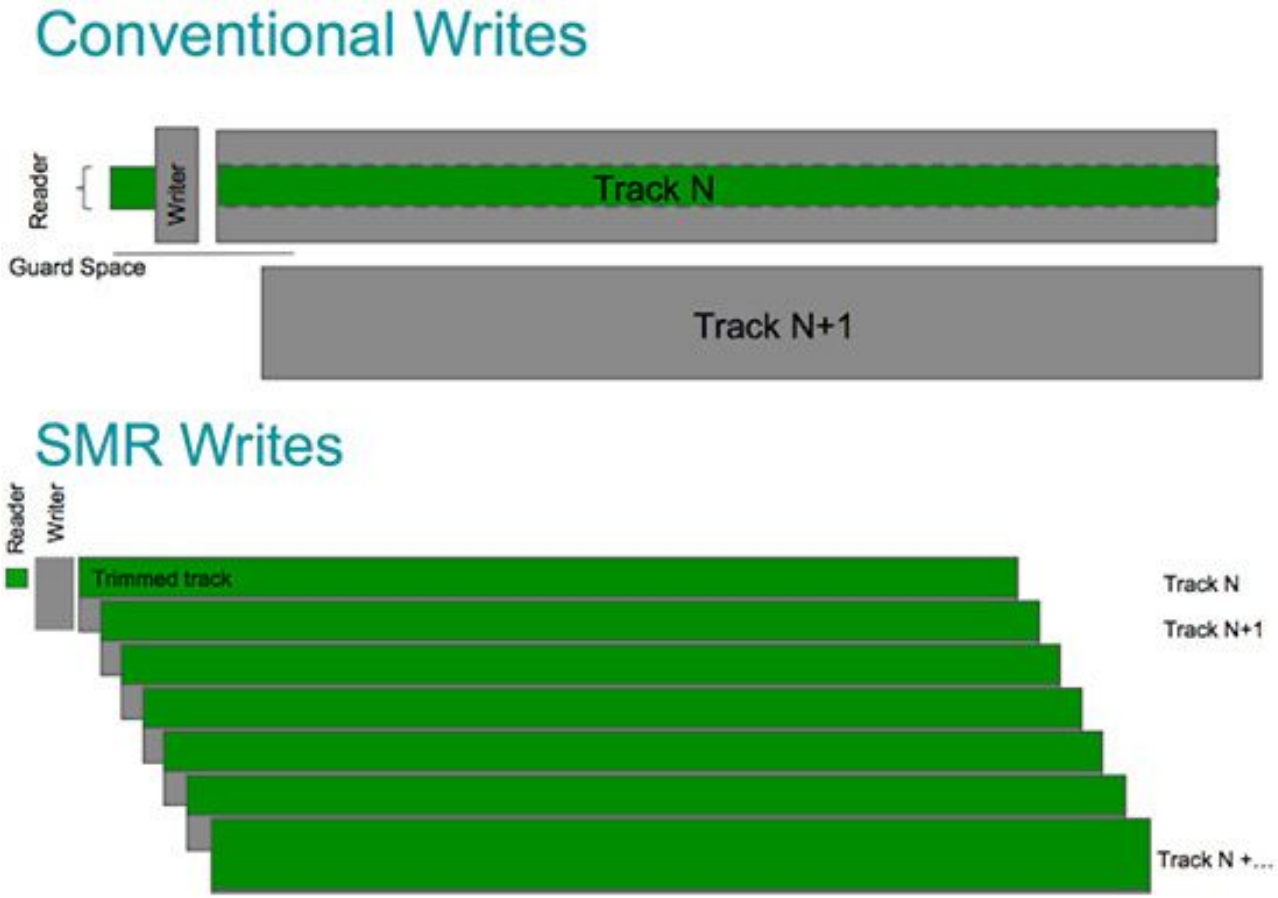
The Amazing Magnetic Disk

- 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
- OS Unit of Transfer: Block
  - Typically, more than one Sector
  - Example: 4KB, 16KB
- Disks so big that some companies (like Google) reportedly only use part of disk for active data
  - Rest is archival data



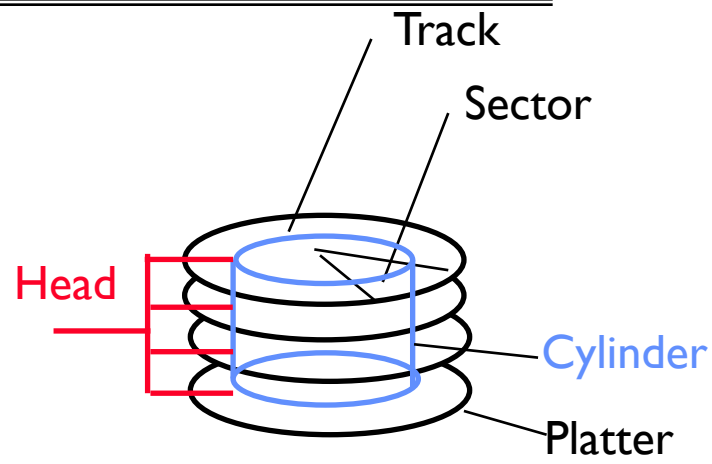
Shingled Magnetic Recording (SMR)

- Overlapping tracks yields greater density, capacity
- Restrictions on writing, complex DSP for reading



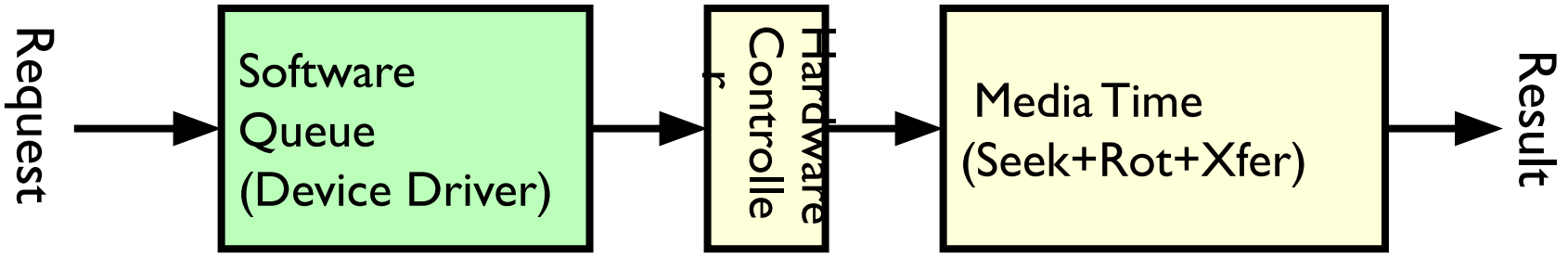
Roof shingles

Magnetic Disk Performance



- Cylinders: all the tracks under the head at a given point on all surfaces
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper cylinder
  - 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

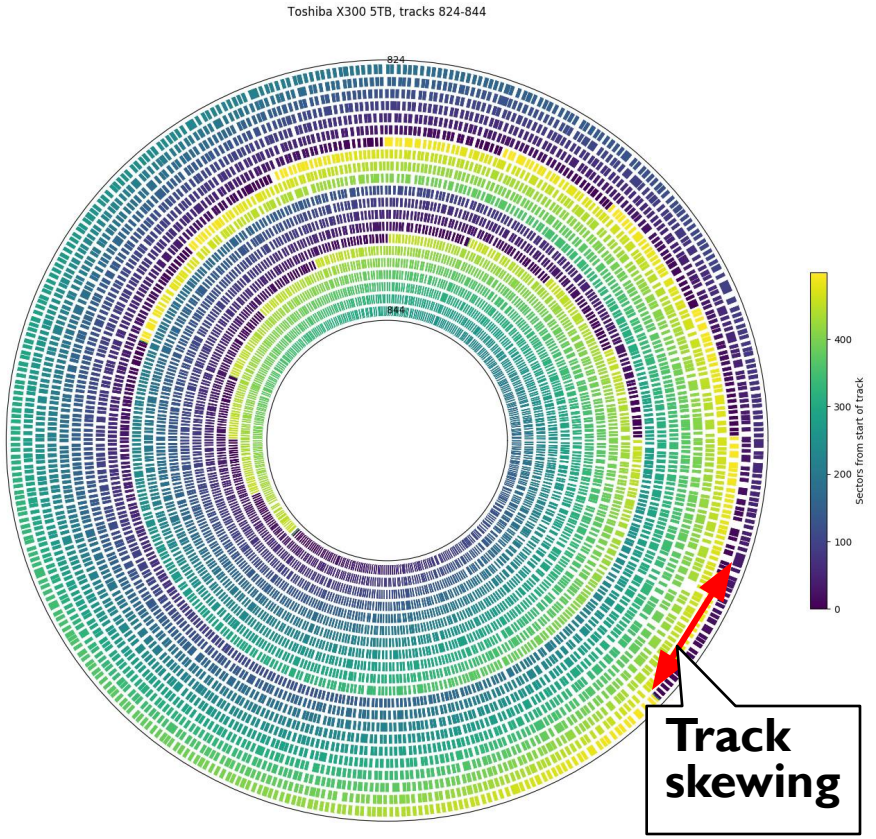
Parameter	Info/Range
Space/Density	Space: 18TB (Seagate), 9 platters, in 3½ inch form factor! **Areal Density: >= 1 Terabit/square inch (PMR, Hellium…)**
Average Seek Time	Typically 4-6 milliseconds
Average Rotational Latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 4-8 milliseconds
Controller Time	Depends on controller hardware
Transfer Time	Typically, 50 to 270 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 faster), now slower

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
    Avg time to find block = ½ × 8ms = 4ms
  - 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 block
- Read block from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.082ms) = 9.082ms
  - Approx 9ms to fetch/put data: 4096 bytes/9.082×10^{-3} s ≅ 451KB/s
- Read block from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.082ms) = 4.082ms
  - Approx 4ms to fetch/put data: 4096 bytes/4.082×10^{-3} s ≅ 1.03MB/s
- Read next block on same track:
  - Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s ≅ 50MB/sec
- Key to using disk effectively 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 th
- 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



Dark blue: beginning of the track

When is Disk Performance Highest?

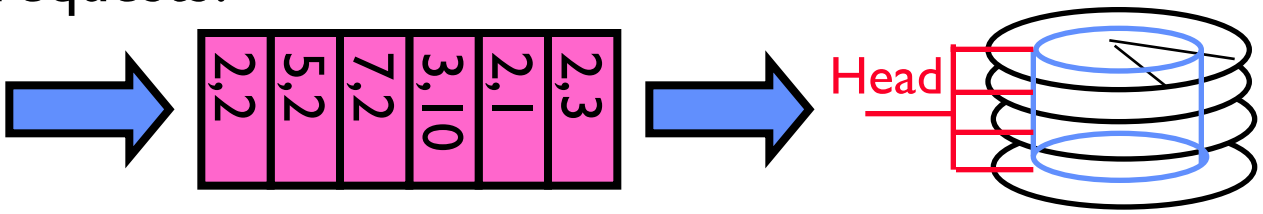
- When there are big sequential reads, or
- When there is so much work to do that they can be piggy backed (see next)

- It is OK to be inefficient when things are mostly idle
- Bursts are both a threat and an opportunity
- <your idea for optimization goes here>
  - Waste space for speed?

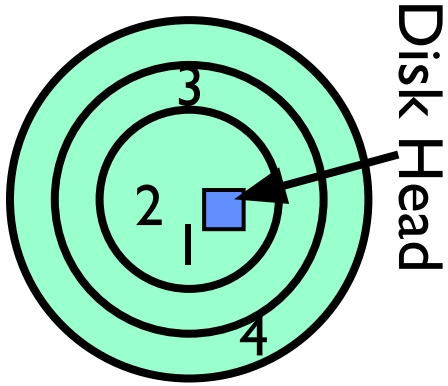
- Other techniques:
  - Reduce overhead through user level drivers
  - Reduce the impact of I/O delays by doing other useful work in the meantime

Disk Scheduling (1/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?

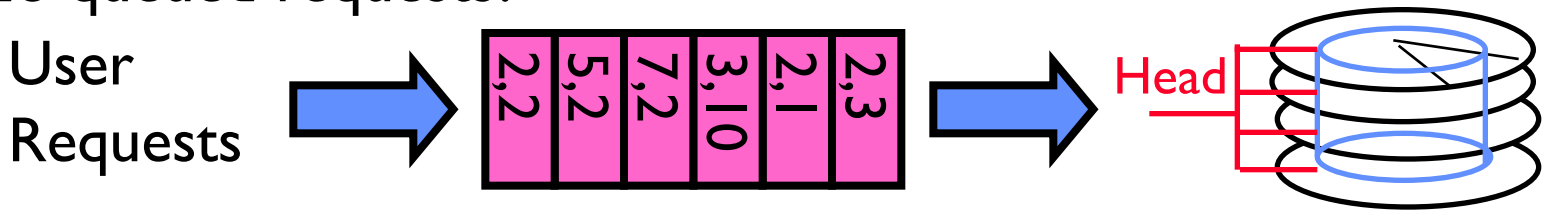


- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks
- SSTF: Shortest seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, today must include rotational delay in calculation, since rotation can be as long as seek
  - Con: SSTF good at reducing seeks, but may lead to starvation

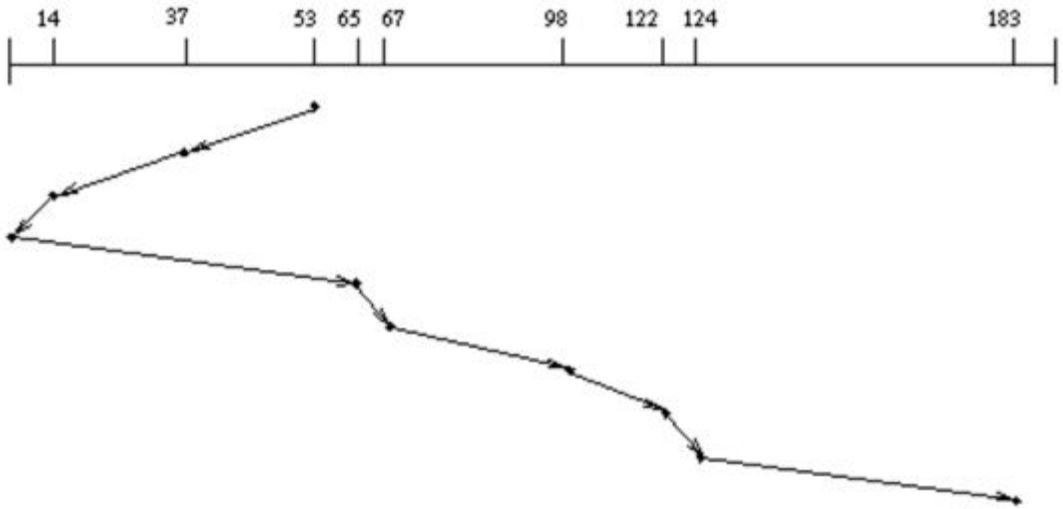


Disk Scheduling (2/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?

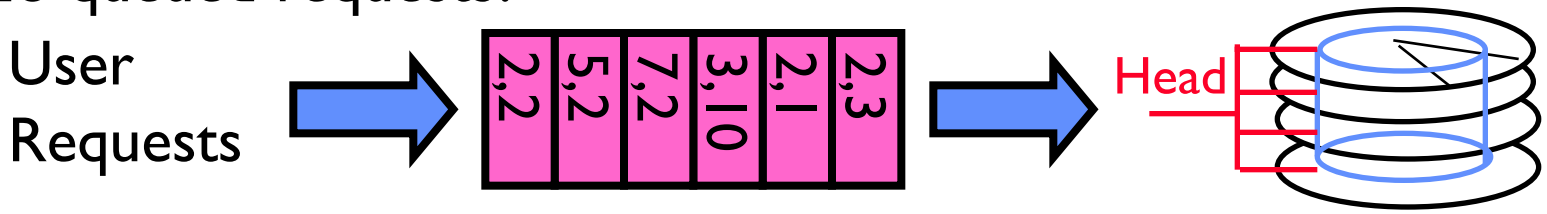


- SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel
  - No starvation, but retains flavor of SSTF

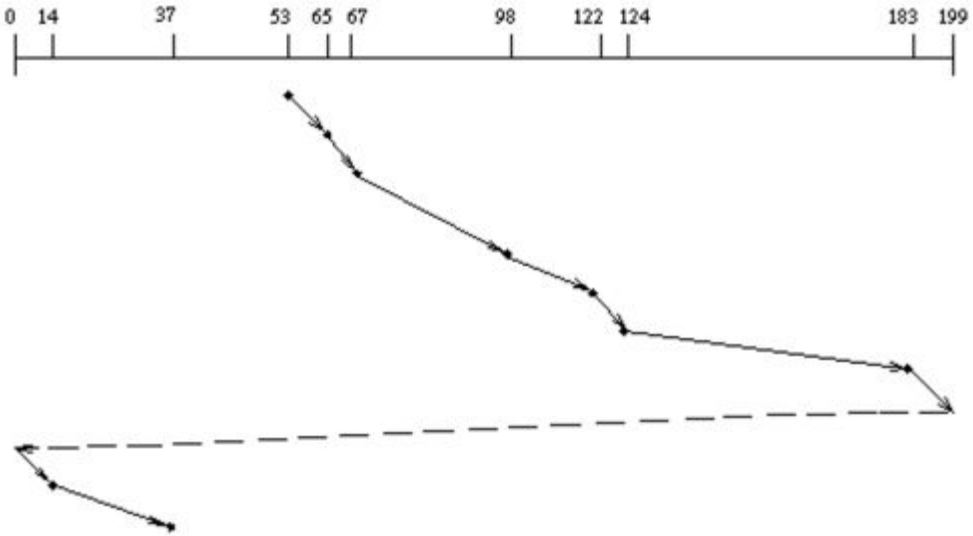


Disk Scheduling (3/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?



- C-SCAN: Circular-Scan: only goes in one direction
  - Skips any requests on the way back
  - Fairer than SCAN, not biased towards pages in middle



Example of Current HDDs

- Seagate Exos X24 (2023)
  - 24 TB hard disk
    - » 10 platters, 20 heads
    - » 1.26 TB/in^2
    - » Helium filled: reduce friction and power
  - 4.16 ms average seek time
  - 4096 byte physical sectors
  - 7200 RPMs
  - Dual 6 Gbps SATA /12Gbps SAS interface
    - » 285MB/s MAX transfer rate
    - » Cache size: 512MB
  - Price: $ 479 (~ $0.02/GB)

800K x

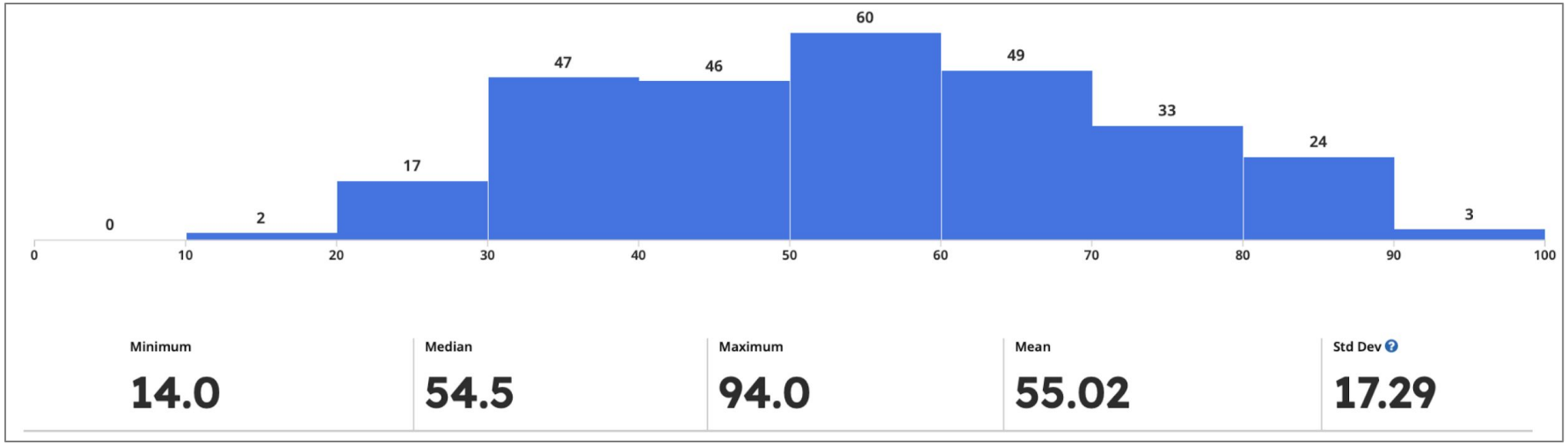
385 x

10 x

850K x

- IBM Personal Computer/AT (1986)
  - 30 MB hard disk
  - 30-40 ms average seek time
  - 0.7-1 MB/s (est.)
  - Price: $500 ($17K/GB)

Administrivia



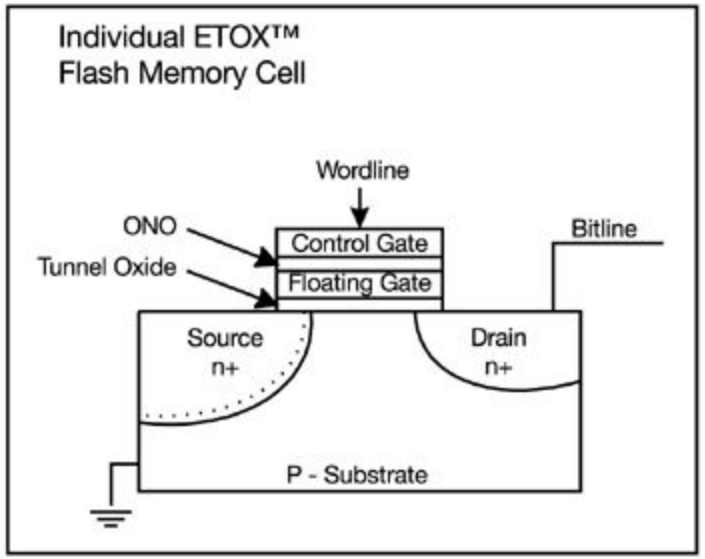
- Midterm 2 results:
  - Better than M1, but we want to do better
  - Remember, we are grading on a curve
  - We are still seeing a strong positive correlation between people attending the lectures and discussion sections, on one hand, and their scores, on the other hand…
- Project 2 code and final report due by Wednesday, Nov 13
- HW 5 released

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!

The Flash Cell

Encode bit by trapping electrons into a cell



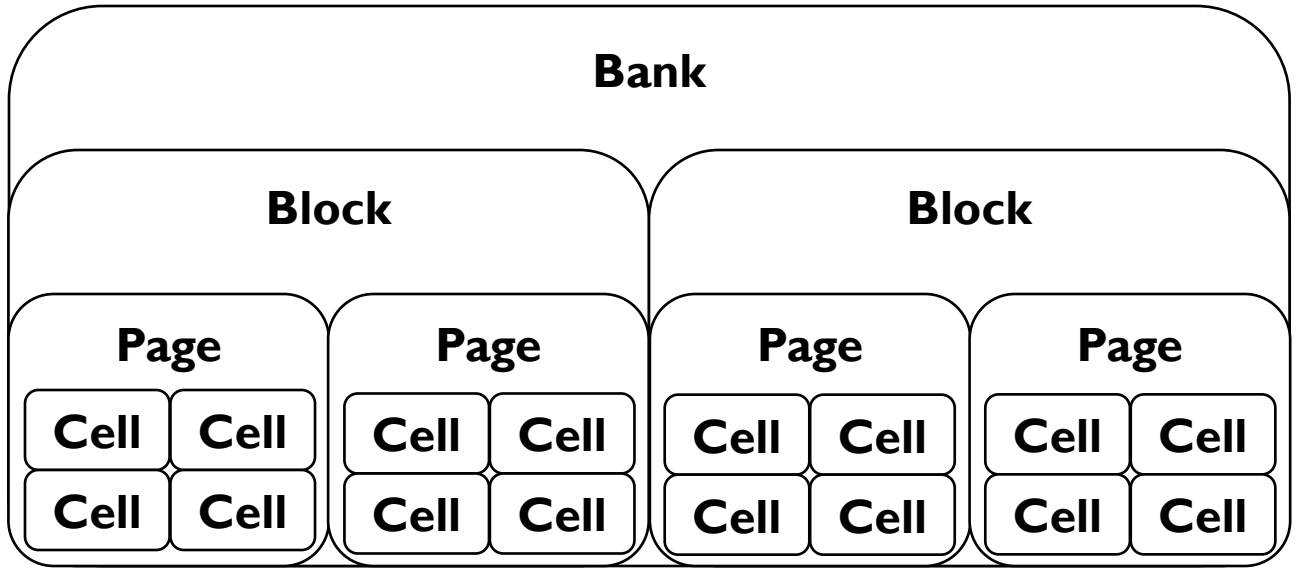
Single-level cell (SLC)

- Single bit is stored within a transistor
- Faster, more lasting (50k to 100k writes before wear out)

Multi-level cell (MLC)

- Two/three bits are encoded into different levels of charge
- Wear out much faster (1k to 10k writes)

Of banks, blocks, cells



Flash chips organized in banks
- Banks can be accessed in parallel

Blocks 128 KB/256KB
- (64 to 258 pages)

Pages Few KB

Cells 1 to 4 bits

Distinction between blocks and pages important in operations!

Low-level flash operations

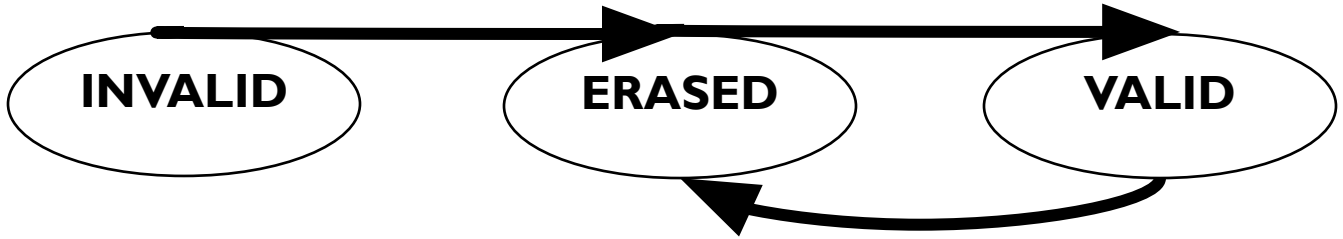
How do you read?

- Chip supports reading *pages*
- **10** microseconds, independently of the previously read page

What about writing? More complicated!

- Must first *erase the block*
  - » Erase quite expensive (milliseconds)
- Once block has been erased, can then *program a page*
  - » Change 1s to 0s within a page.
  - » **100-200** microseconds.
- Blocks can only be erased a limited number of times!

Low-level flash operations



		`iiii`	*Initial: pages in block are invalid* (`i`)
Erase()	→	`EEEE`	*State of pages in block set to erased* (`E`)
Program(0)	→	`VEEE`	*Program page 0; state set to valid* (`V`)
Program(0)	→	**error**	*Cannot re-program page after programming*
Program(1)	→	`VVEE`	*Program page 1*
Erase()	→	`EEEE`	*Contents erased; all pages programmable*

Low-level flash operations

Assume block of 4 pages. All valid.
Want to write Page 0

Page 0	Page 1	Page 2	Page 3
00011000	11001110	00000001	00111111
VALID	VALID	VALID	VALID

Step 1: erase full block

Page 0	Page 1	Page 2	Page 3
11111111	11111111	11111111	11111111
ERASED	ERASED	ERASED	ERASED

Step 2: program page 0

Page 0	Page 1	Page 2	Page 3
00000011	11111111	11111111	11111111
VALID	ERASED	ERASED	ERASED

SSD Architecture

Recall that SSDs uses low-level Flash operations to provide same interface as HDD

- read and write chunk (4KB) at a time

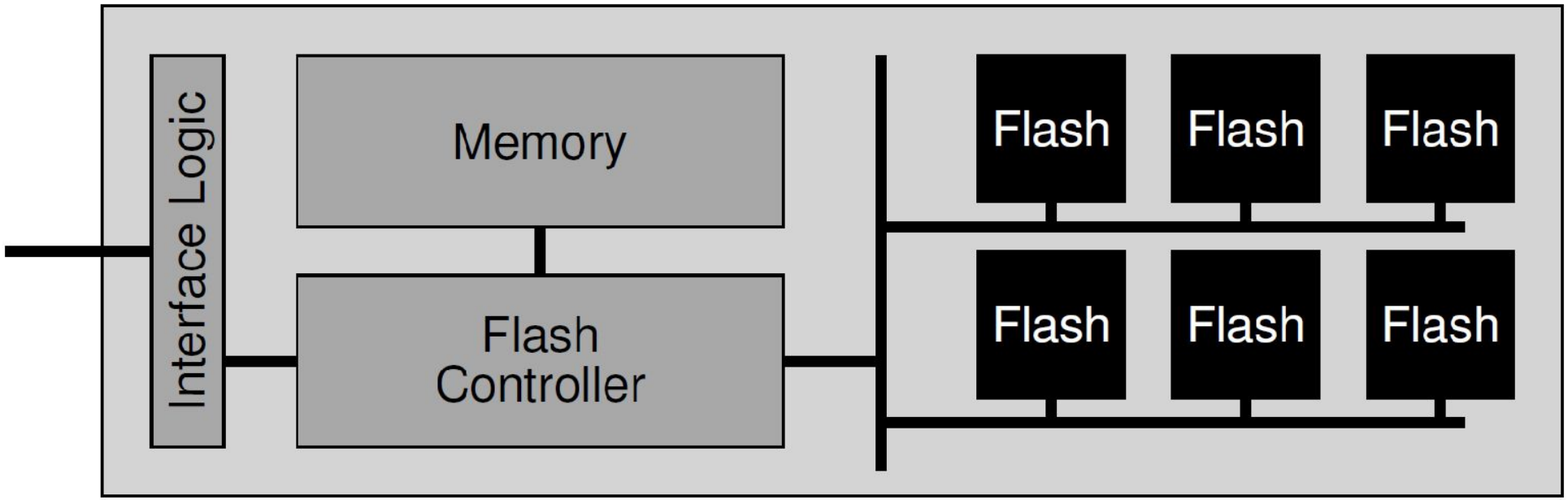
Reads are easy, but for writes, can only overwrite data one block (256KB) at a time!

Why not just erase and rewrite new version of entire 256KB block?

- Erasure is very slow, i.e., 1.5 ms
- Each block has a finite lifetime, can only be erased and rewritten about 10K times
- Heavily used blocks likely to wear out quickly

Rule of thumb: writes 10x slower than reads; erasure 10x slower than writes

SSD Architecture (Simplified)



Flash Translation Layer (FTL)

Add a layer of indirection: the flash translation layer

- Translates request for logical blocks (device interface) to low-level Flash blocks and pages

Reduce write amplification

- Ratio of the total write traffic in bytes issues to the flash chip by the FTL divided by the total write traffic issued by the OS to the device

Avoid wear out

- A single block should not be erased too often

FTL – Two Systems Principles

FTL uses *indirection* and copy-on-write

Maintains mapping tables in DRAM

- 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

Copy on Write/ Log-structured FTL

- 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

FTL Example

a0, a1: virtual block numbers (used by OS)

Initial State

Mapping Table:

Block 0		Block 1	
E	E	E	E
0	1	0	1

Write(a0)

Mapping Table
a0 → 0,0

Block 0		Block 1	
a0			
V	E	E	E

Write(a1)

Mapping Table:
a0 → 0,0 / a1 → 0,1

Block 0		Block 1	
a0	a1		
V	V	E	E

Write(a1)

Mapping Table:
a0 → 0,0 / a1 → 1,0

Block 0		Block 1	
a0	a1	a1	
V	V	V	E

Write(a0)

Mapping Table:
a0 → 1,1 / a1 → 1,0

Block 0		Block 1	
a0	a1	a1	a0
V	V	V	V

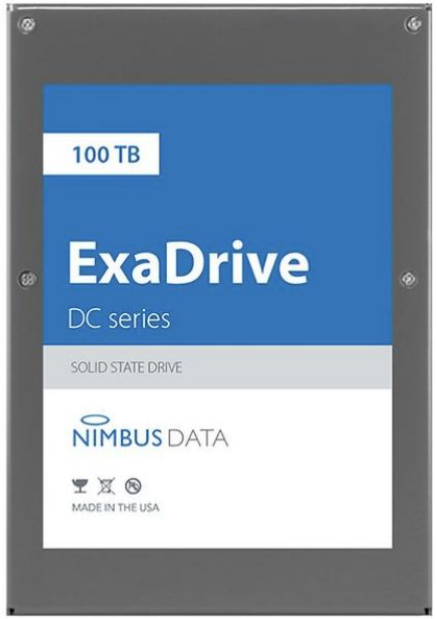
Garbage Collect

Mapping Table:
a0 → 1,1 / a1 → 1,0

Block 0		Block 1	
		a1	a0
E	E	V	V

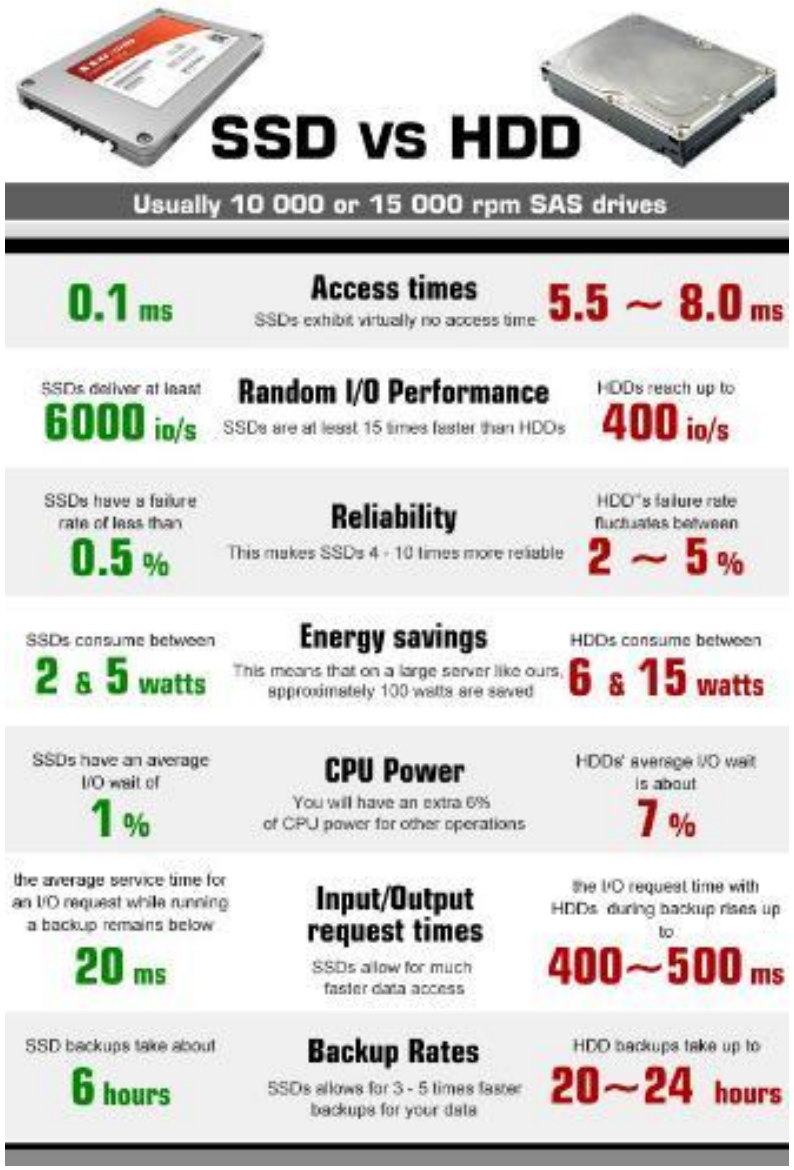
Some "Current" (large) 3.5in SSDs

- Nimbus SSD: 100TB
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - *Unlimited writes for 5 years!*
  - Price: ~ $40K ($0.4/GB)
    - » However, 50TB drive costs $12500 ($0.25/GB)



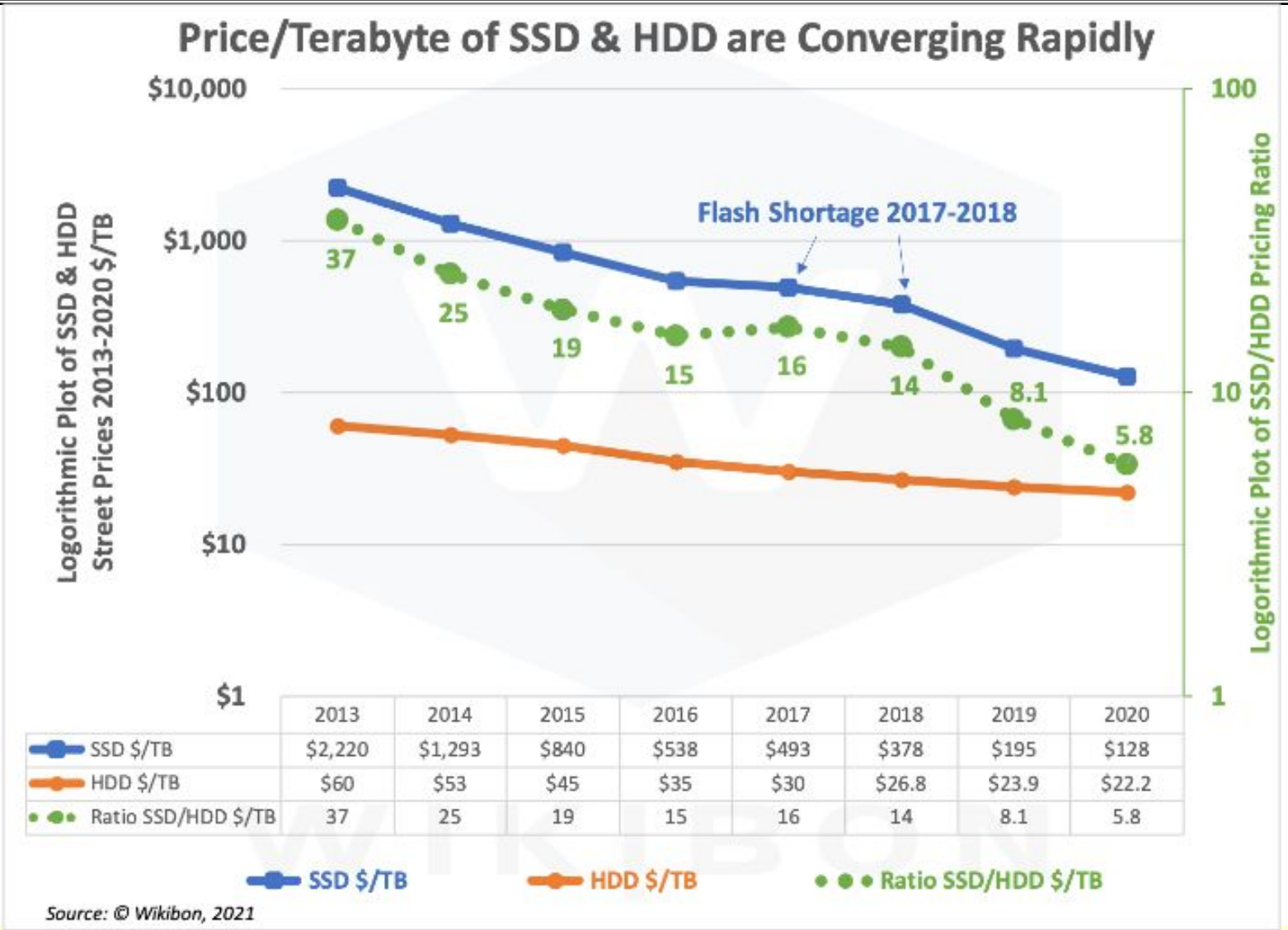
- Samsung 870 QVO
  - Up to 8TB
  - Seq reads 560MB/s
  - Seq writes 530MB/s
  - Random read/write Ops (IOPS): 98K/88K
  - Price (Amazon): $602 ($0.075/GB)

HDD vs. SSD Comparison



HDD	SDD
Require seek + rotation	No seeks
Not parallel (one head)	Parallel
Brittle (moving parts)	No moving parts
Random reads take 10s milliseconds	Random reads take 10s microseconds
Slow (Mechanical)	Wears out
Cheap/large storage	Expensive/smaller storage

HDD vs. SSD Comparison



	2013	2014	2015	2016	2017	2018	2019	2020
SSD $/TB	$2,220	$1,293	$840	$538	$493	$378	$195	$128
HDD $/TB	$60	$53	$45	$35	$30	$26.8	$23.9	$22.2
Ratio SSD/HDD $/TB	37	25	19	15	16	14	8.1	5.8

SSD prices drop faster than HDD

Conclusion (1/2)

- Notification mechanisms
  - Interrupts
  - Polling: Report results through status register that processor looks at periodically
- Direct Memory Access (DMA)
  - Permit devices to directly access memory
  - Free up processor from transferring every byte

Conclusion (2/2)

- 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
  - SSD: Queuing time + controller + transfer (erasure & wear)