Operating Systems
(Honor Track)

IO: General I/O, Disk and SSD

Xin Jin

Spring 2022

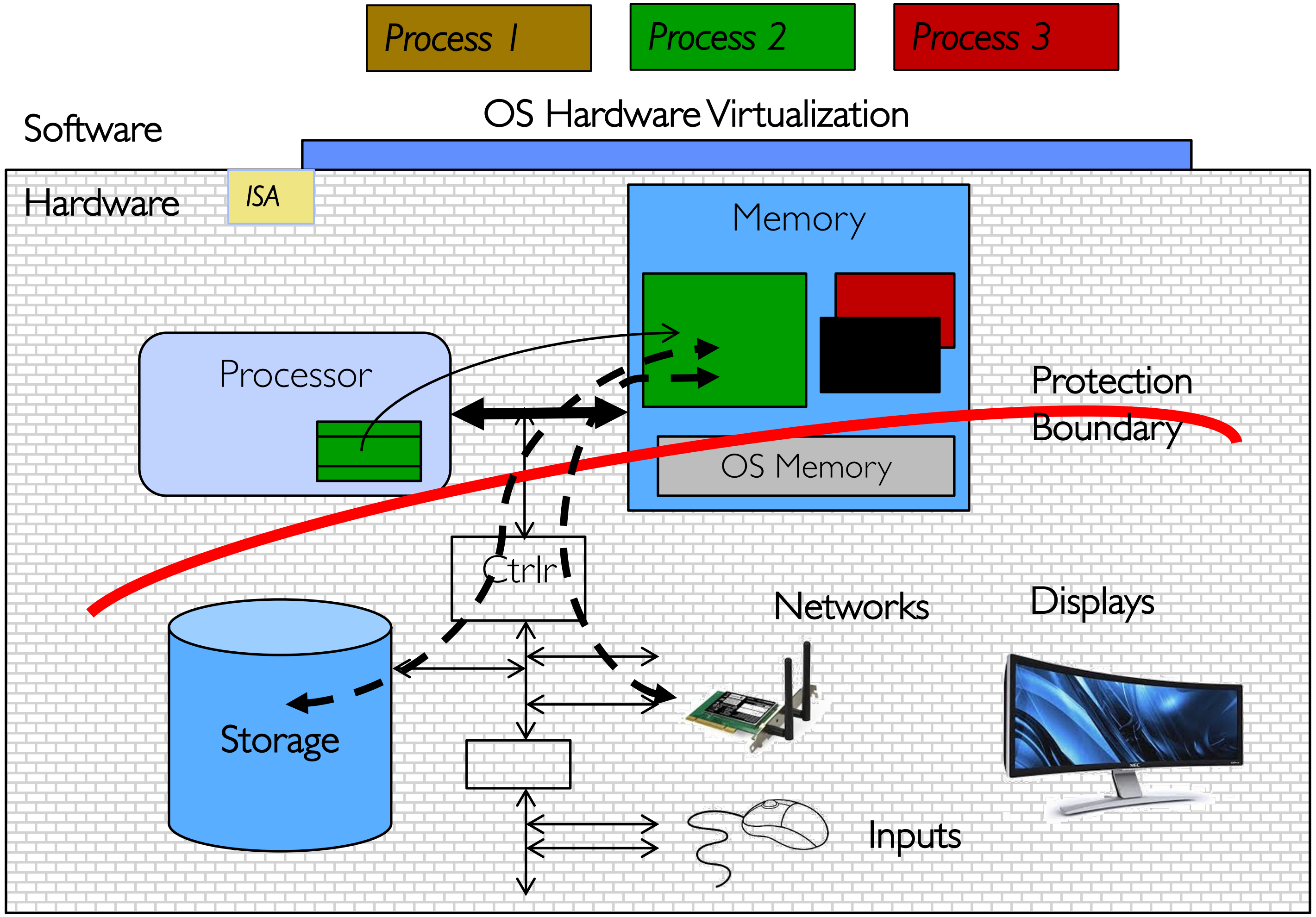
Acknowledgments: Ion Stoica, Berkeley CS 162

Requirements of I/O

- So far, we have studied:
  - Abstractions: the APIs provided by the OS to applications running in a process
  - Synchronization/Scheduling: How to manage the CPU

- What about I/O?
  - Without I/O, computers are useless
  - But… thousands of devices, each slightly different
    - » How can we standardize the interfaces to these devices?
  - Devices unreliable: media failures and transmission errors
    - » How can we make them reliable???
  - Devices unpredictable and/or slow
    - » How can we manage them if we don't know what they will do or how they will perform?

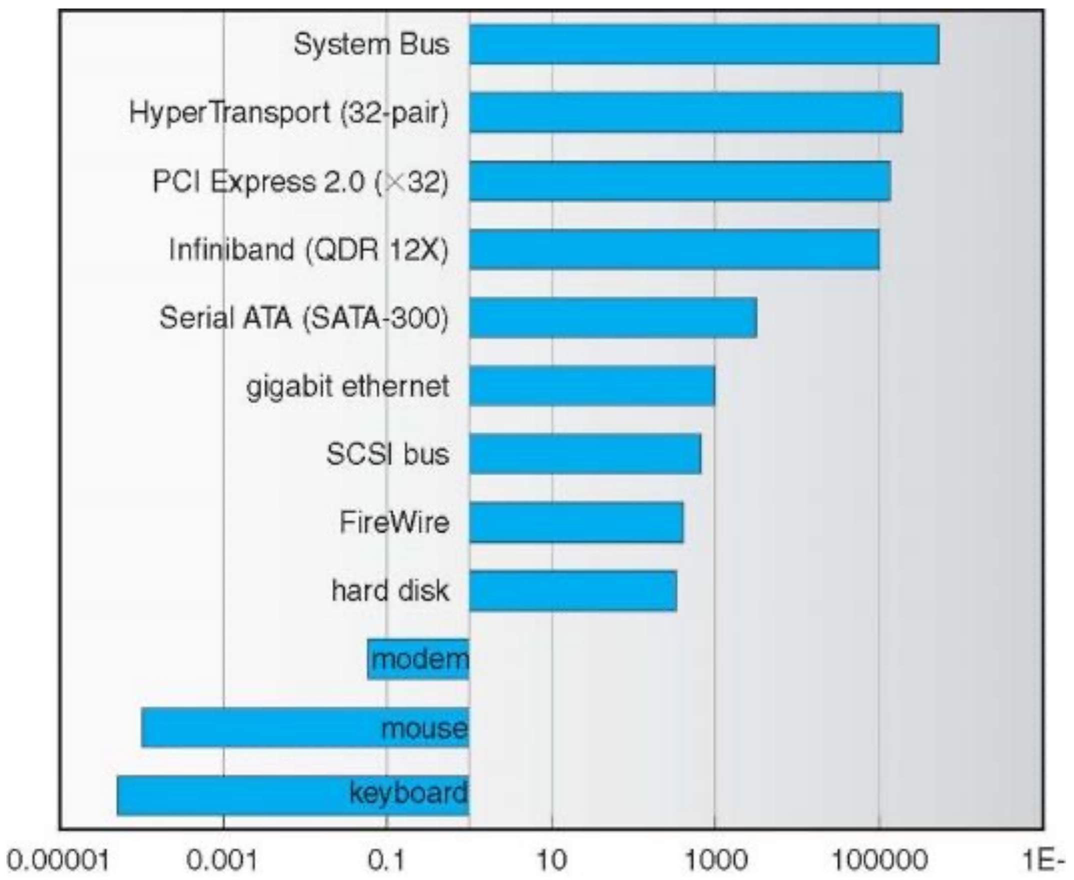
Recall: OS Basics: I/O



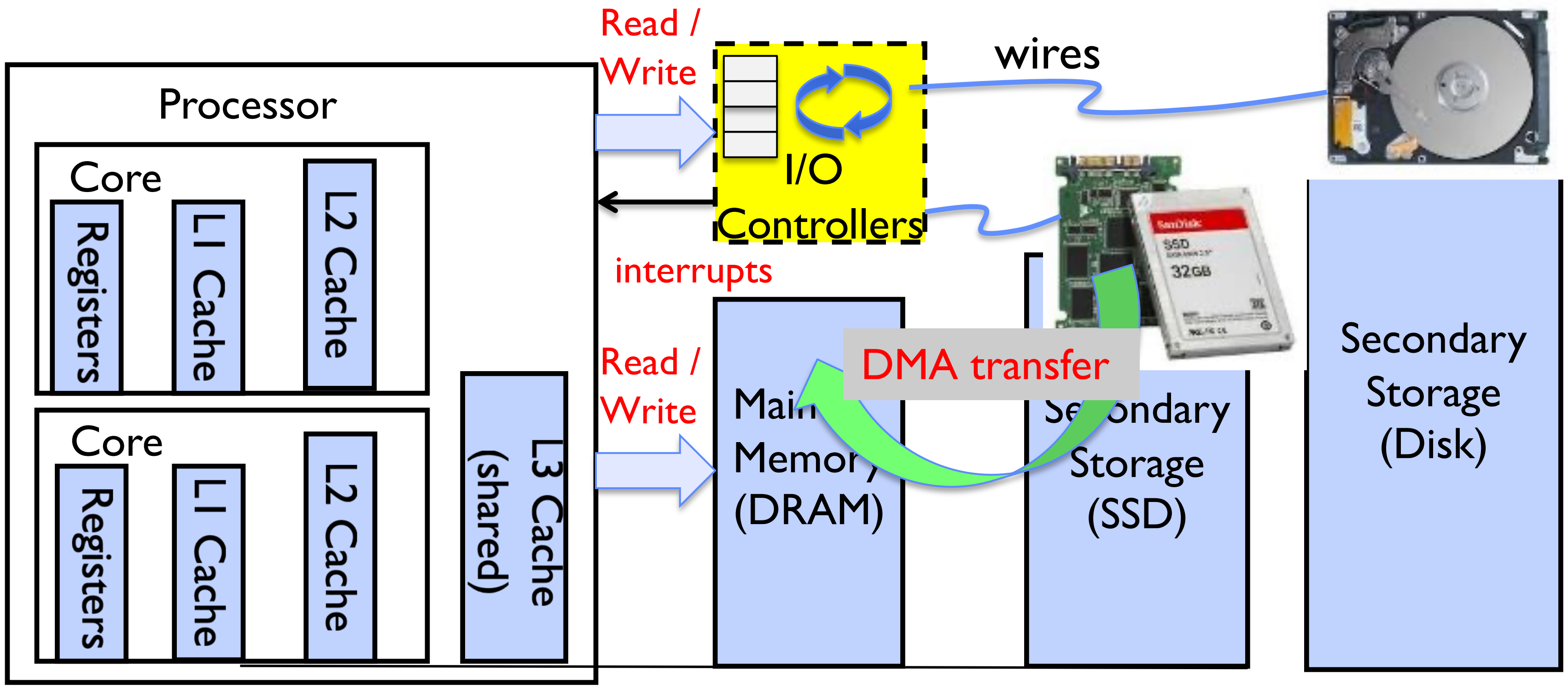
- OS provides common services in form of I/O

Example: Device Transfer Rates in Mb/s (Sun Enterprise 6000)

- Device rates vary over 12 orders of magnitude!!!
- System must be able to handle this wide range
  - Better not have high overhead/byte for fast devices
  - Better not waste time waiting for slow devices

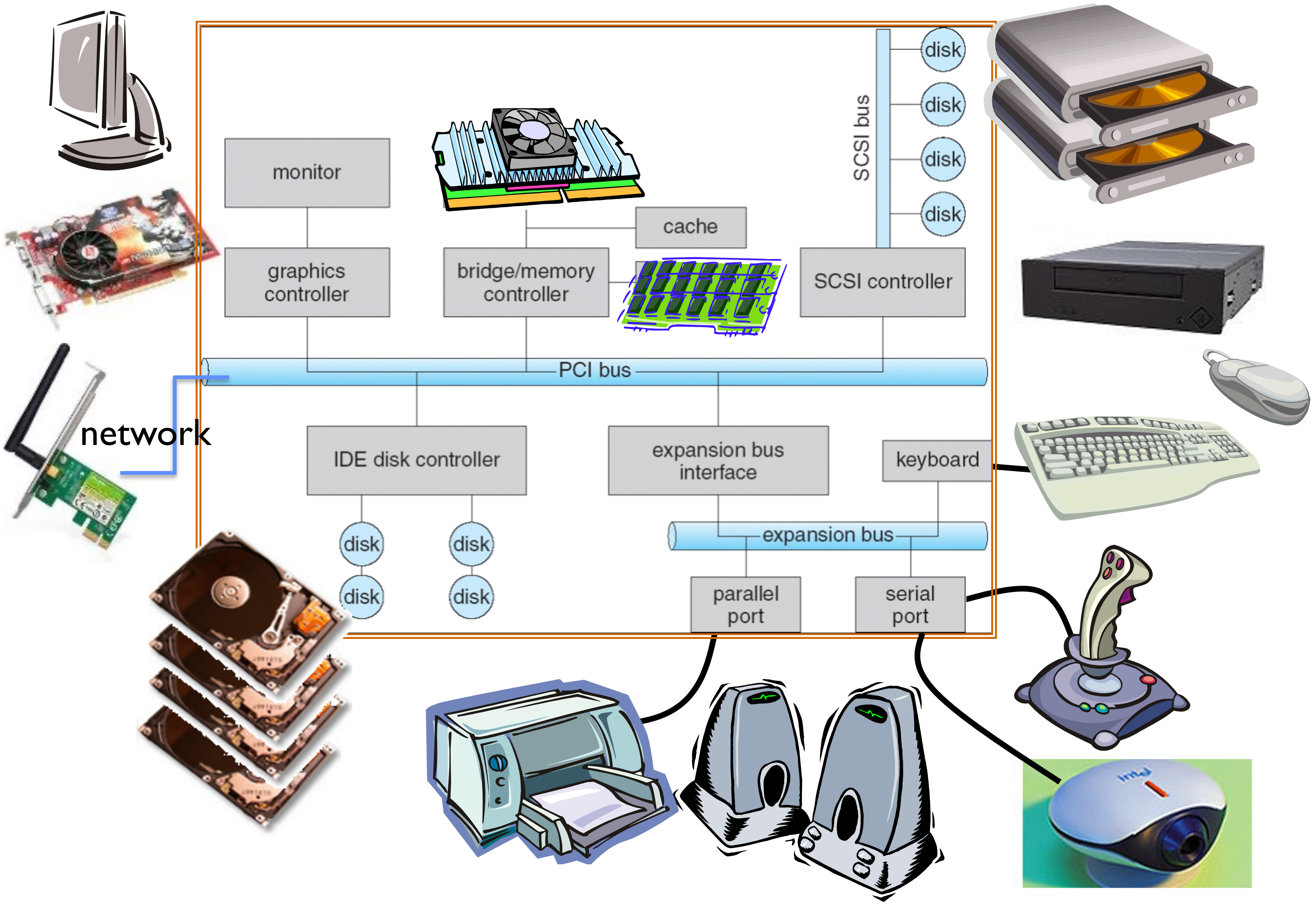


In a Picture

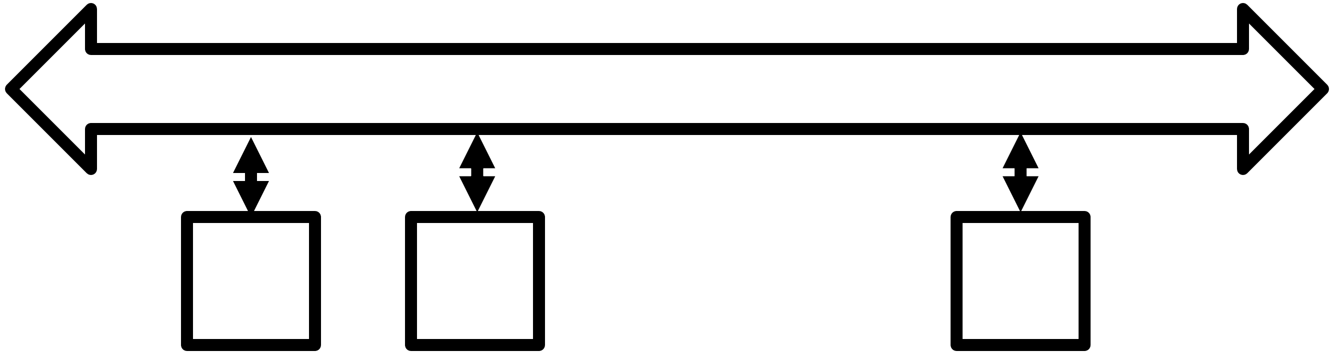


- I/O devices you recognize are supported by I/O Controllers
- Processors accesses them by reading and writing IO registers as if they were memory
  - Write commands and arguments, read status and results

Modern I/O Systems



What's a bus?



- Common set of wires for communication among hardware devices plus protocols for carrying out data transfer transactions
  - Operations: e.g., Read, Write
  - Control lines, Address lines, Data lines
  - Typically, multiple devices
- Protocol: initiator requests access, arbitration to grant, identification of recipient, handshake to convey address, length, data
- Very high BW close to processor (wide, fast, and inflexible), low BW with high flexibility out in I/O subsystem

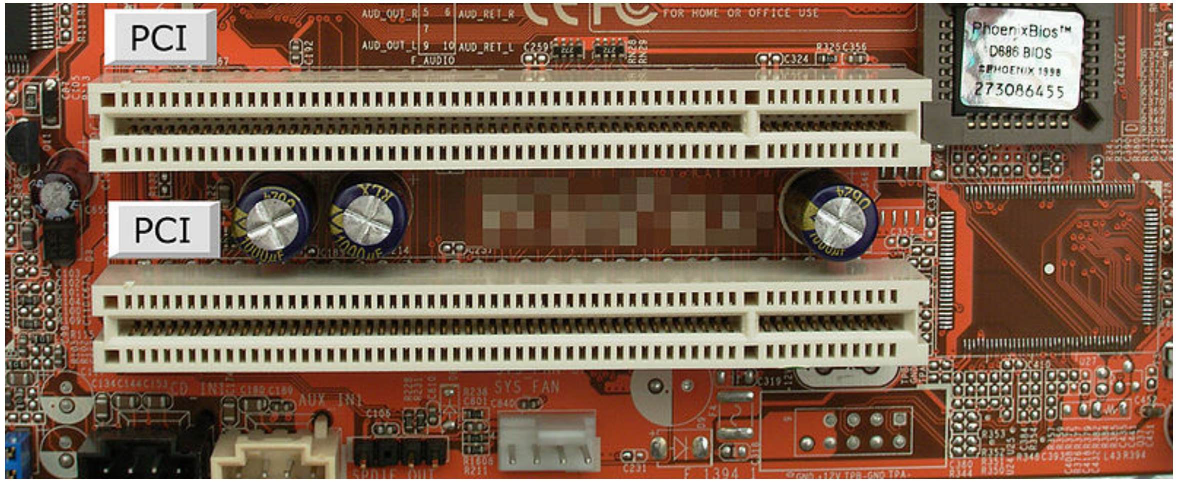
Why a Bus?

- Buses let us connect n devices over a single set of wires, connections, and protocols
  - $O(n^2)$ relationships with 1 set of wires (!)

- Downside: Only one transaction at a time
  - The rest must wait
  - "Arbitration" aspect of bus protocol ensures the rest wait

PCI Bus Evolution

- PCI (Peripheral Component Interconnect) started life out as a bus
- But a parallel bus has many limitations
  - Multiplexing address/data for many requests
  - Slowest devices must be able to tell what’s happening (e.g., for arbitration)

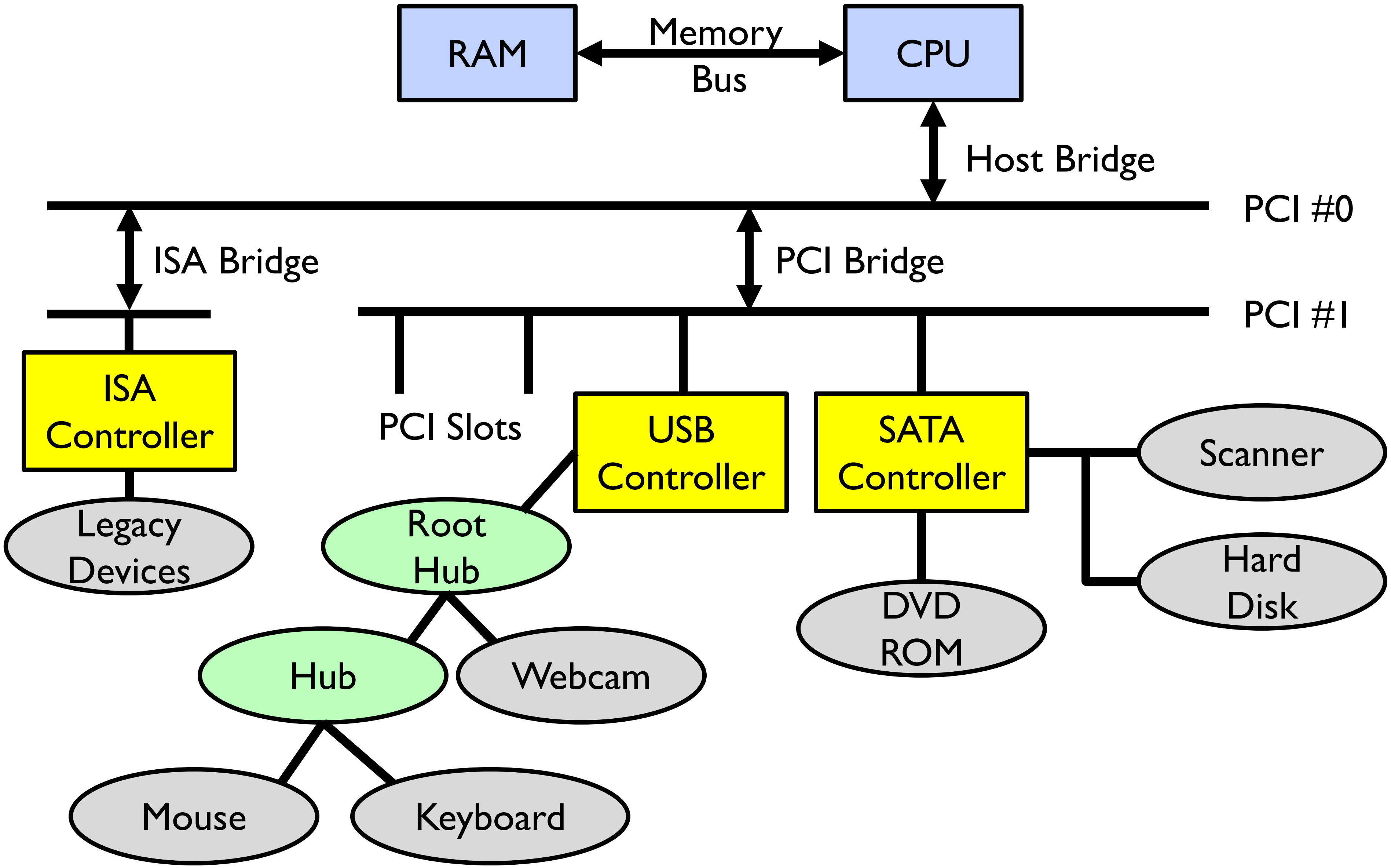


PCI Express "Bus"

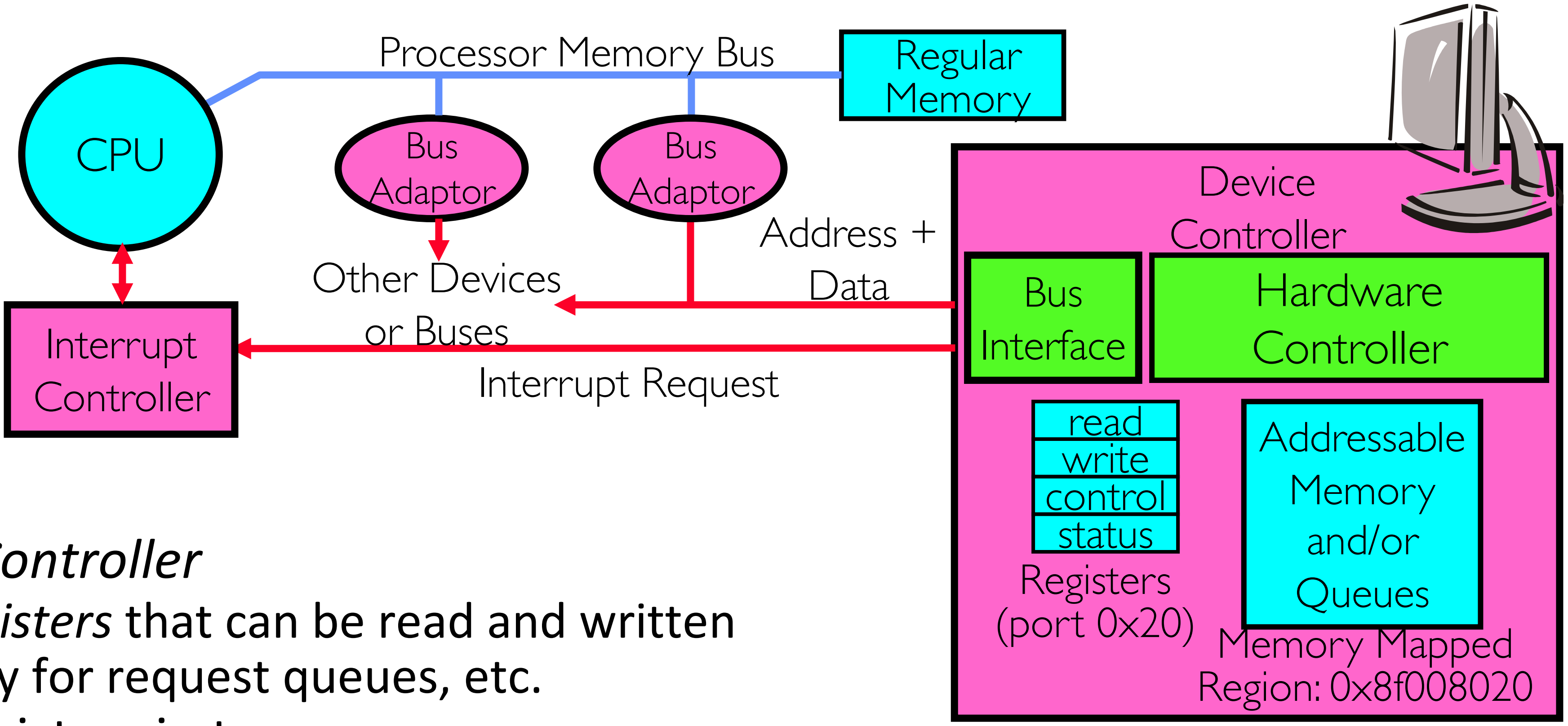
- No longer a parallel bus
- Really a **collection of fast serial channels** or "lanes"
- Devices can use as many as they need to achieve a desired bandwidth
- Slow devices don't have to share with fast ones
  - Space multiplexing vs. time multiplexing

- One of the successes of device abstraction in Linux was the ability to migrate from PCI to PCI Express
  - The physical interconnect changed completely, but the old API still worked

Example: PCI Architecture



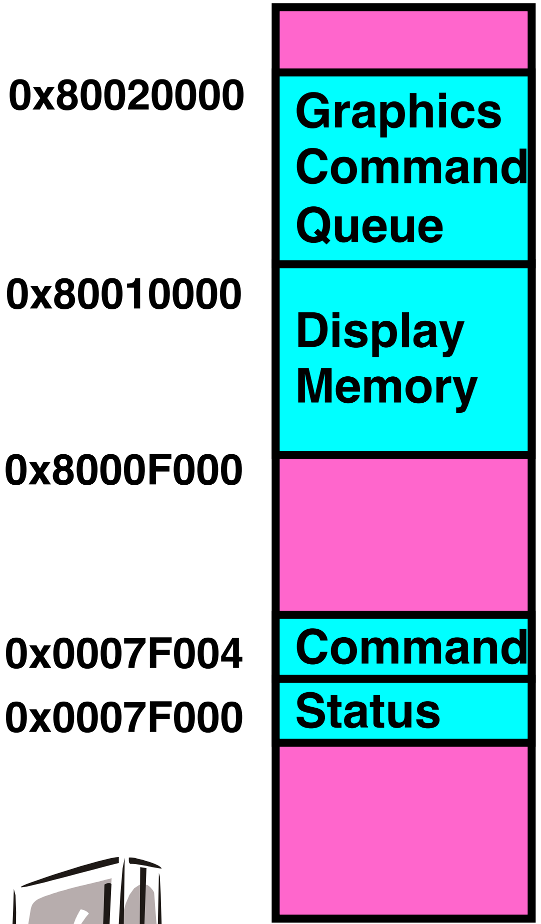
How does the Processor Talk to the Device?

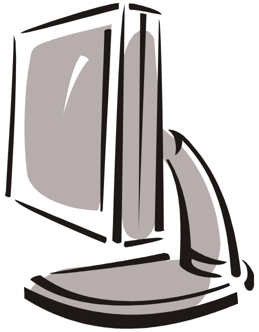


- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues, etc.
- Processor accesses registers in two ways:
  - Port-Mapped I/O: in/out instructions
    - » Example from the Intel architecture: `out 0x21,AL`
  - Memory-mapped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

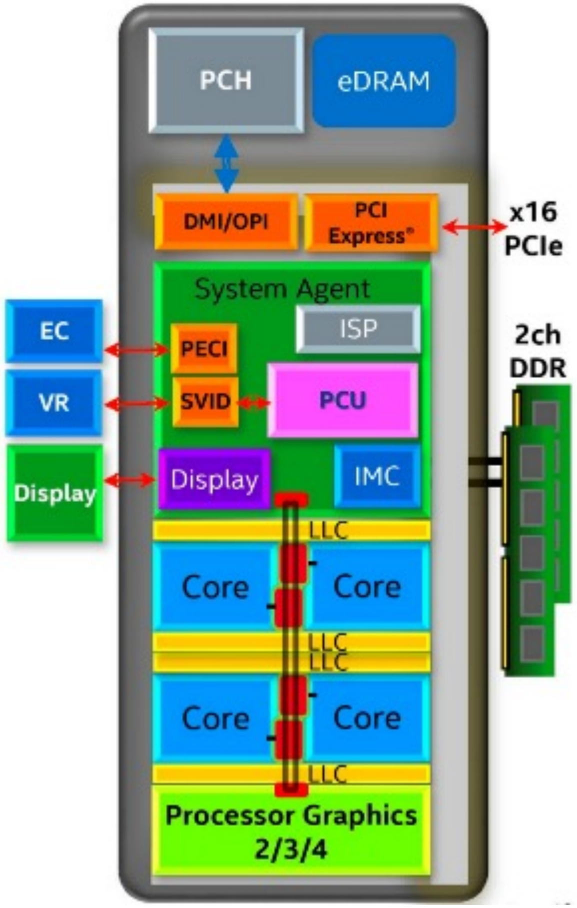




Physical Address Space

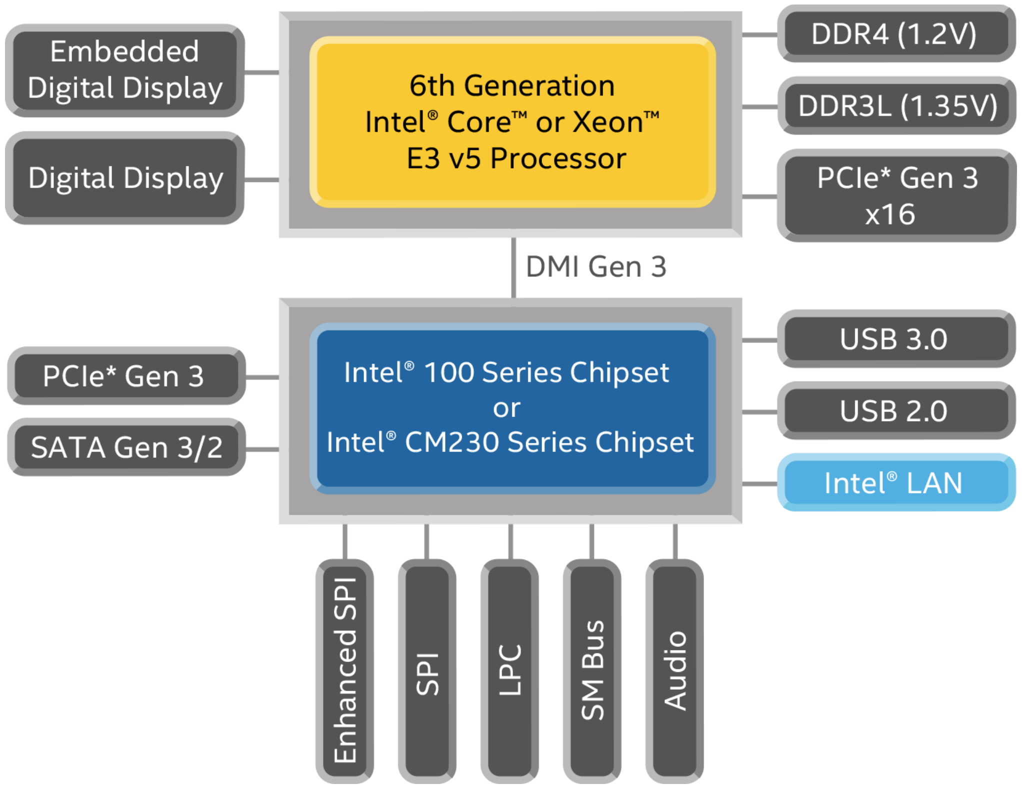
There's more than just a CPU in there!

Chip-scale Features of 2015 x86 (Sky Lake)



- Significant pieces:
  - Four OOO cores with deeper buffers
    - » Intel MPX (Memory Protection Extensions)
    - » Intel SGX (Software Guard Extensions)
    - » Issue up to 6 μ-ops/cycle
  - GPU, System Agent (Mem, Fast I/O)
  - Large shared L3 cache with on-chip ring bus
    - » 2 MB/core instead of 1.5 MB/core
    - » High-BW access to L3 Cache
- Integrated I/O
  - Integrated memory controller (IMC)
    - » Two independent channels of DRAM
  - High-speed PCI-Express (for Graphics cards)
  - Direct Media Interface (DMI) Connection to PCH (Platform Control Hub)

Sky Lake I/O: PCH



Sky Lake
System Configuration

- Platform Controller Hub
  - Connected to processor with proprietary bus
    - » Direct Media Interface (DMI)
- Types of I/O on PCH:
  - USB, Ethernet
  - Thunderbolt 3
  - Audio, BIOS support
  - More PCI Express (lower speed than on Processor)
  - SATA (for Disks)

Operational Parameters for I/O

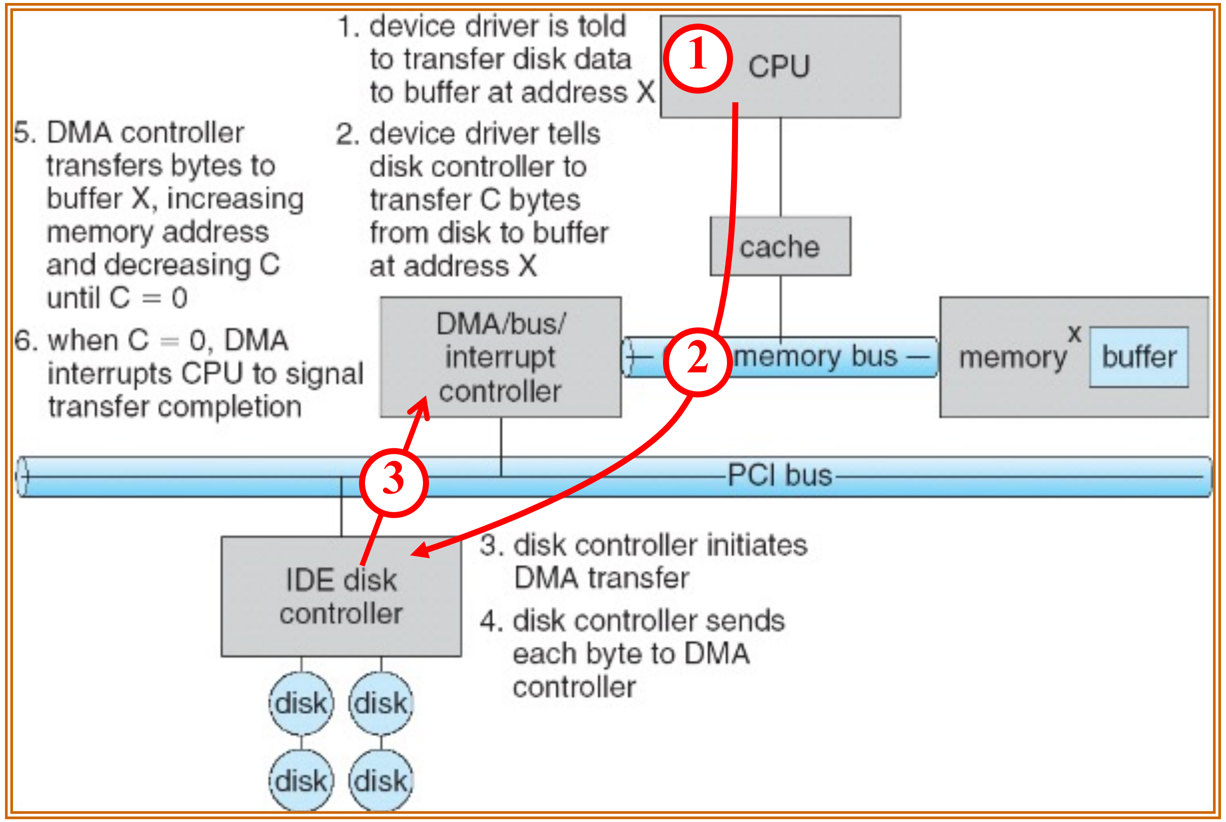
- Data granularity: Byte vs. Block
  - Some devices provide single byte at a time (e.g., keyboard)
  - Others provide whole blocks (e.g., disks, networks, etc.)

- Access pattern: Sequential vs. Random
  - Some devices must be accessed sequentially (e.g., tape)
  - Others can be accessed "randomly" (e.g., disk, cd, etc.)
    - » Fixed overhead to start transfers
  - Some devices require continual monitoring
  - Others generate interrupts when they need service (e.g., keyboard, network card)

- Transfer Mechanism: Programmed IO and DMA (Directed Memory Access)

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:

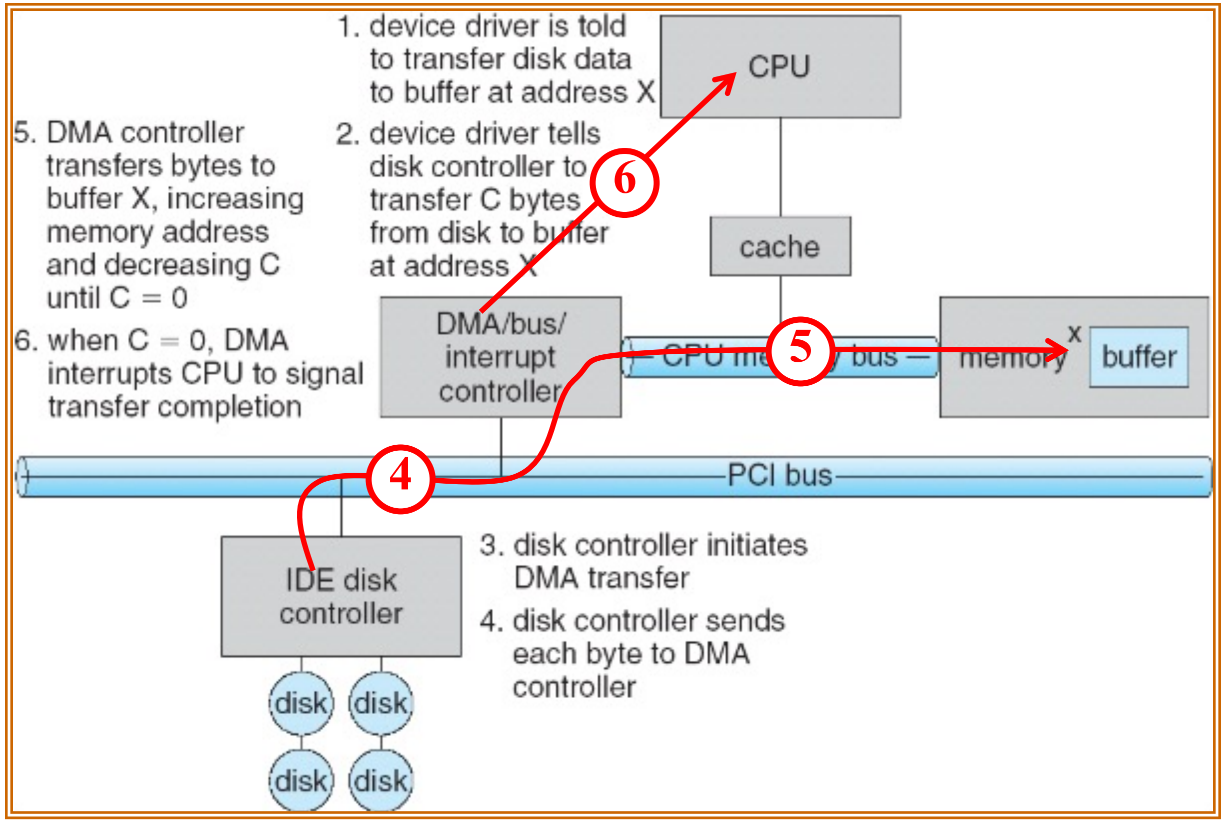


Transferring Data To/From Controller

- 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

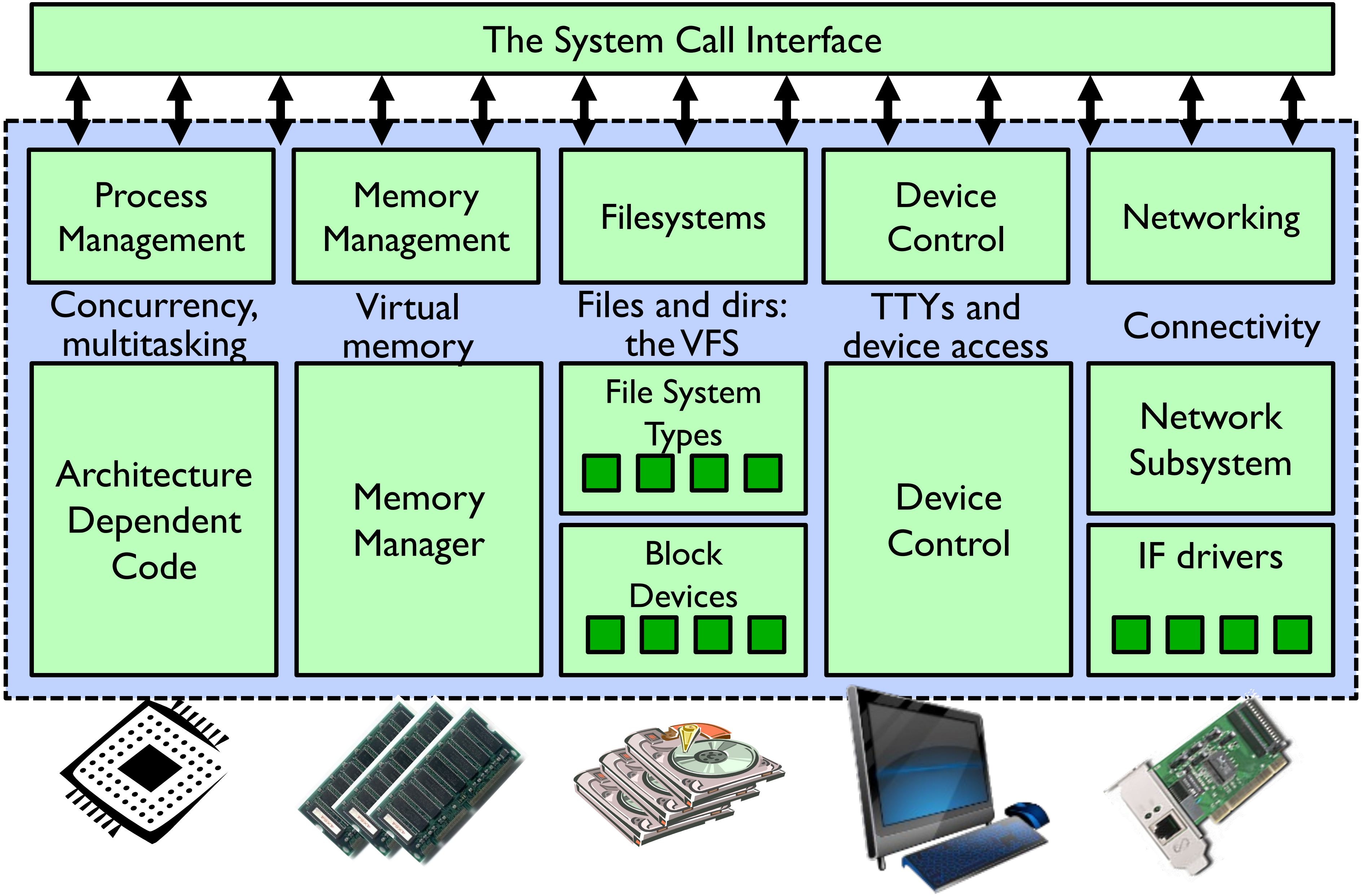
- Sample interaction with DMA controller:



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

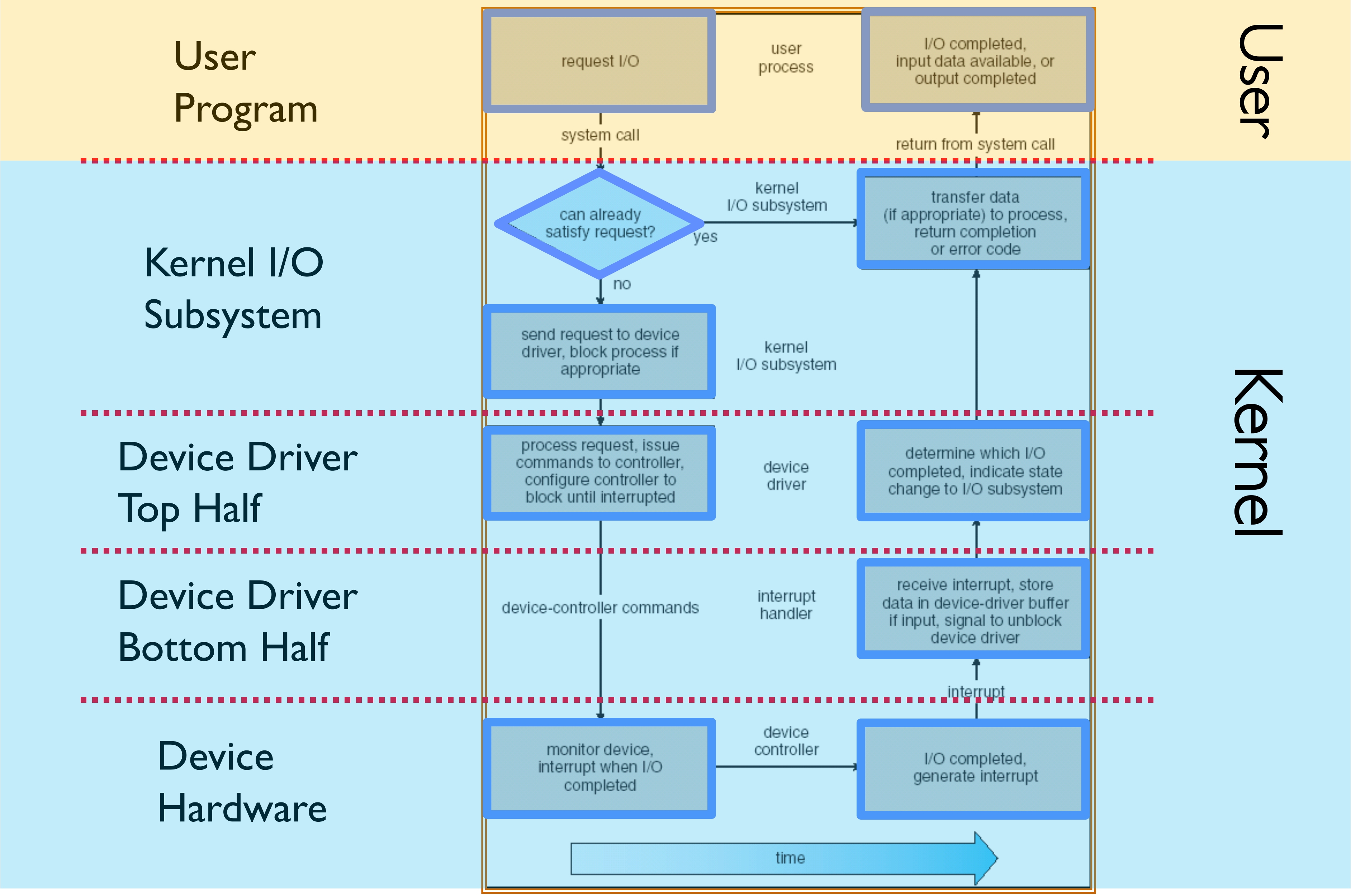


Device Drivers

- 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()`
    - » 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

Life Cycle of An I/O Request



The Goal of the I/O Subsystem

- Provide Uniform Interfaces, Despite Wide Range of Different Devices
  - This code works on many different devices:

```
FILE fd = fopen("/dev/something", "rw");
for (int i = 0; i < 10; i++) {
  fprintf(fd, "Count %d\n", i);
}
close(fd);
```

  - Why? Because code that controls devices ("device driver") implements standard interface
- We will try to get a flavor for what is involved in actually controlling devices in rest of lecture
  - Can only scratch surface!

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?

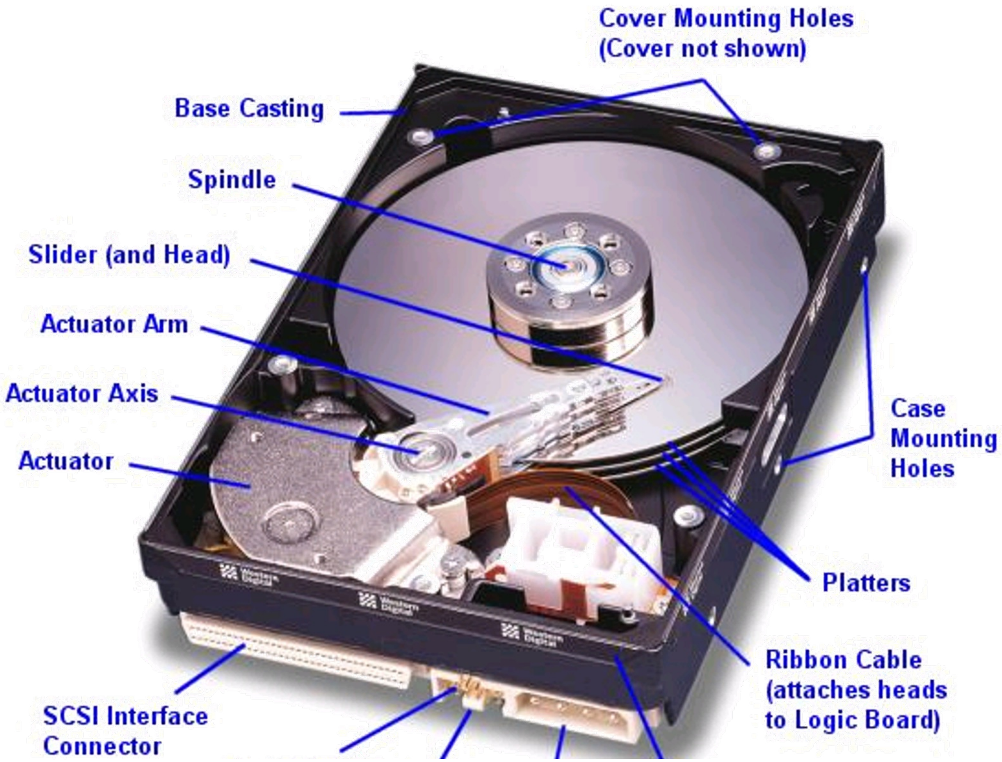
- Blocking Interface: "Wait"
  - When request data (e.g. `read()` system call), put process to sleep until data is ready
  - When write data (e.g. `write()` system call), put process to sleep until device is ready for data
- Non-blocking Interface: "Don't Wait"
  - Returns quickly from read or write 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
  - Slow performance for random access
  - Better performance for sequential access

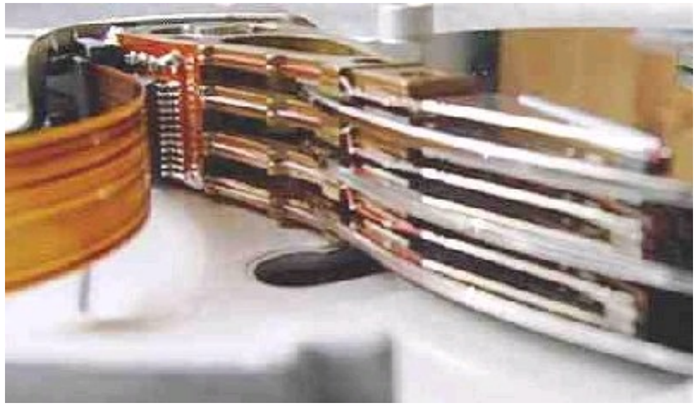
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Ability to store data degrades with the number of writes

Hard Disk Drives (HDDs)



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

IBM/Hitachi Microdrive

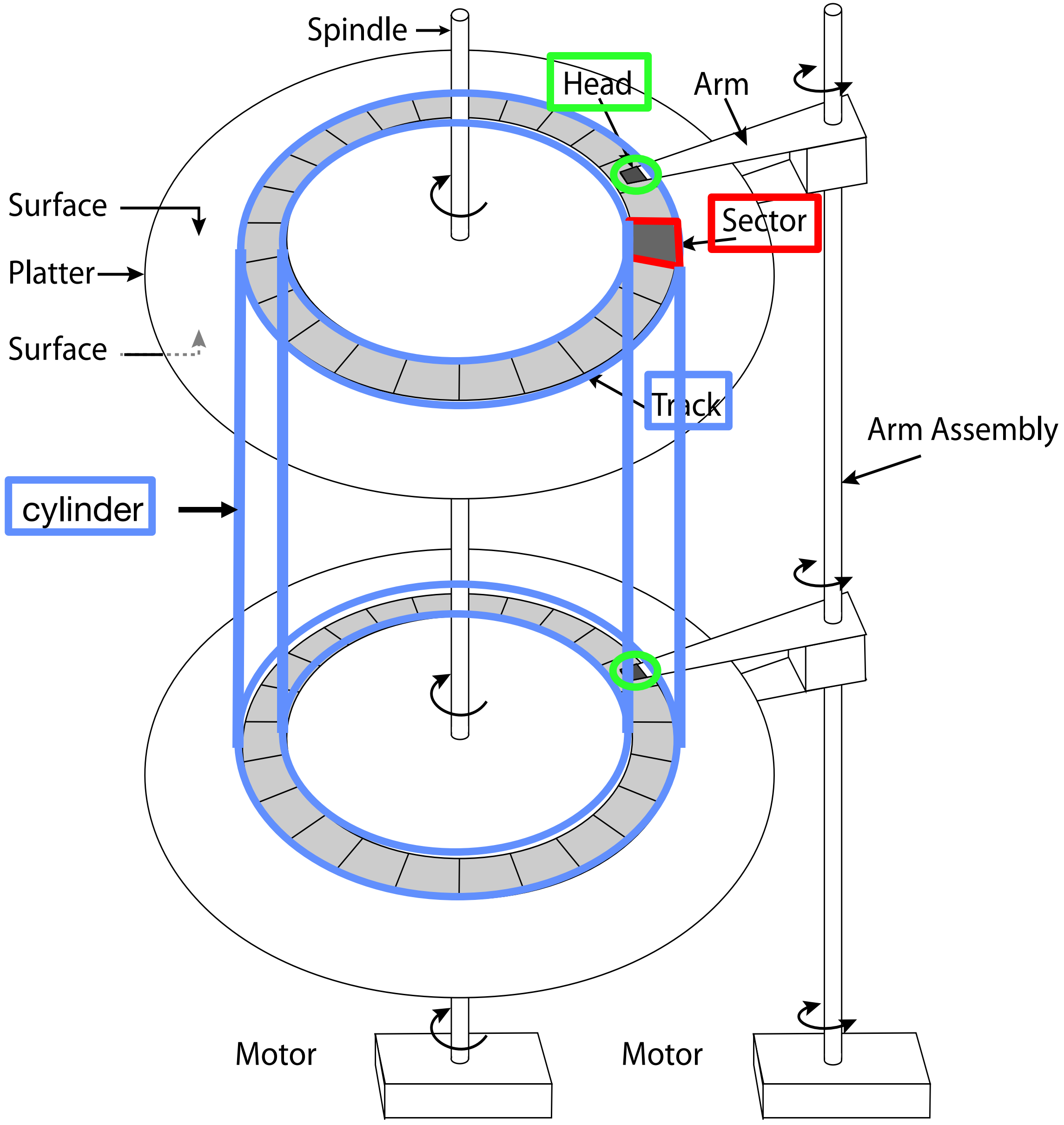


Read/Write Head
Side View

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

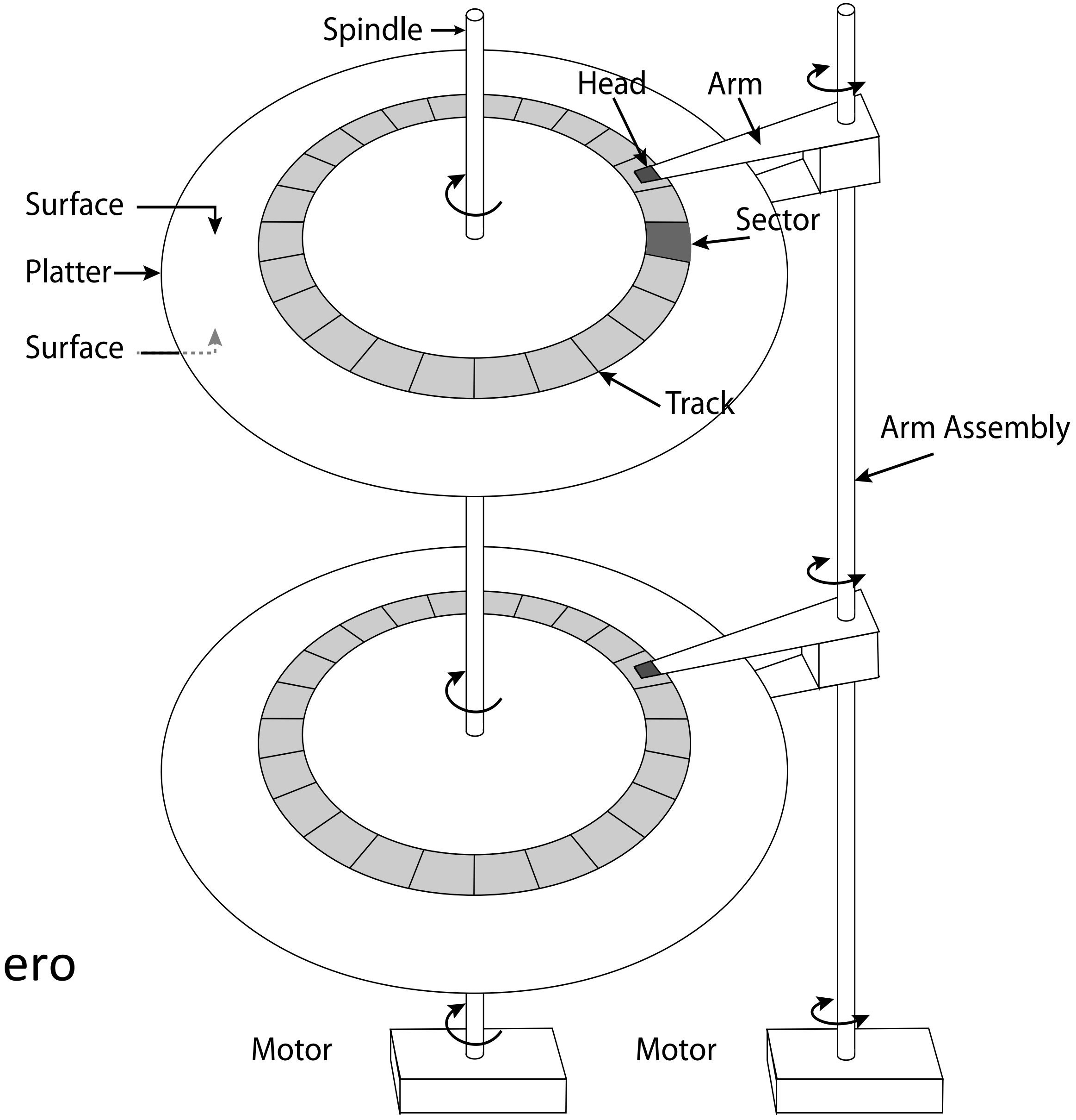
The Amazing Magnetic Disk

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



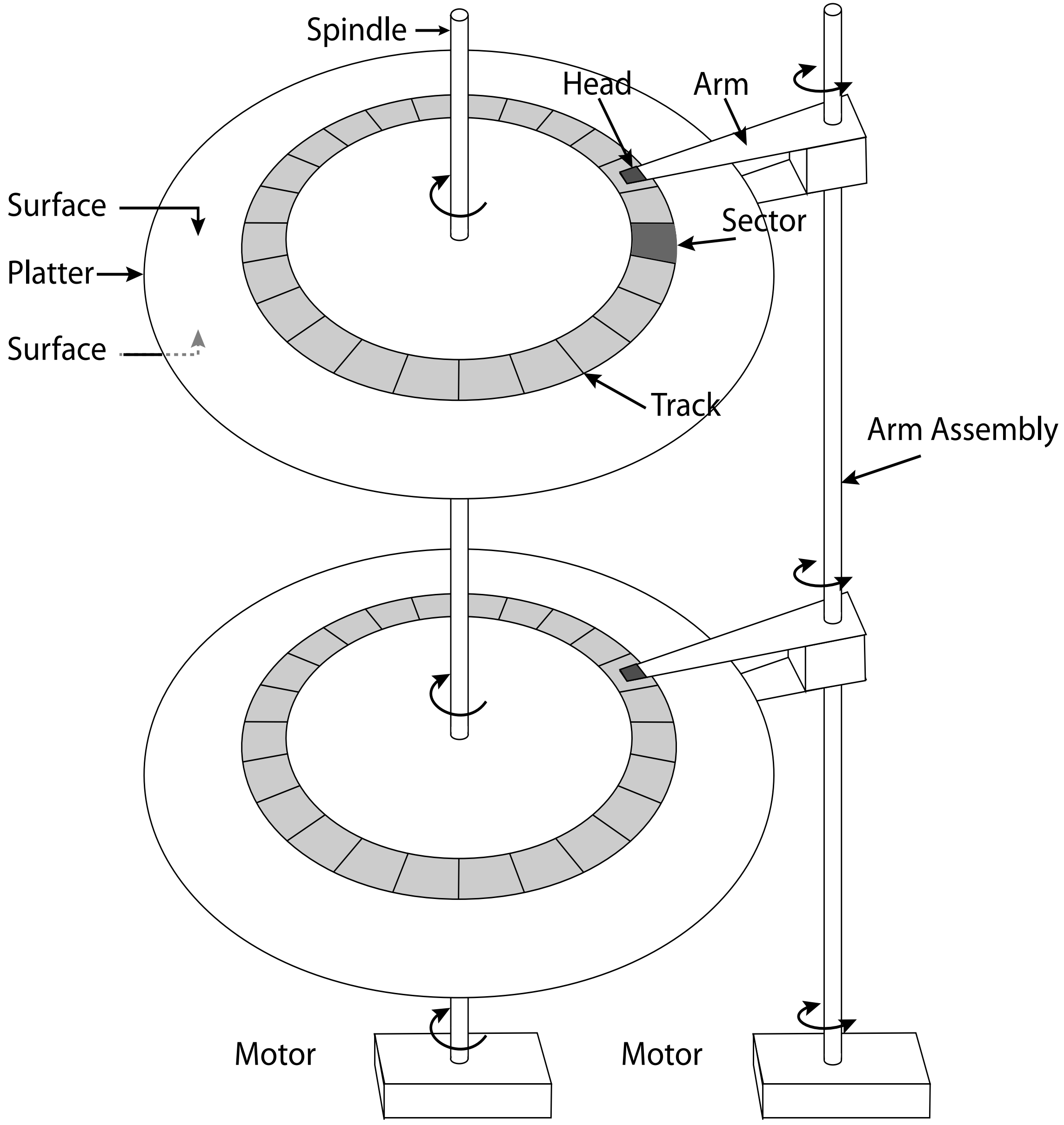
The Amazing Magnetic Disk

- 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)



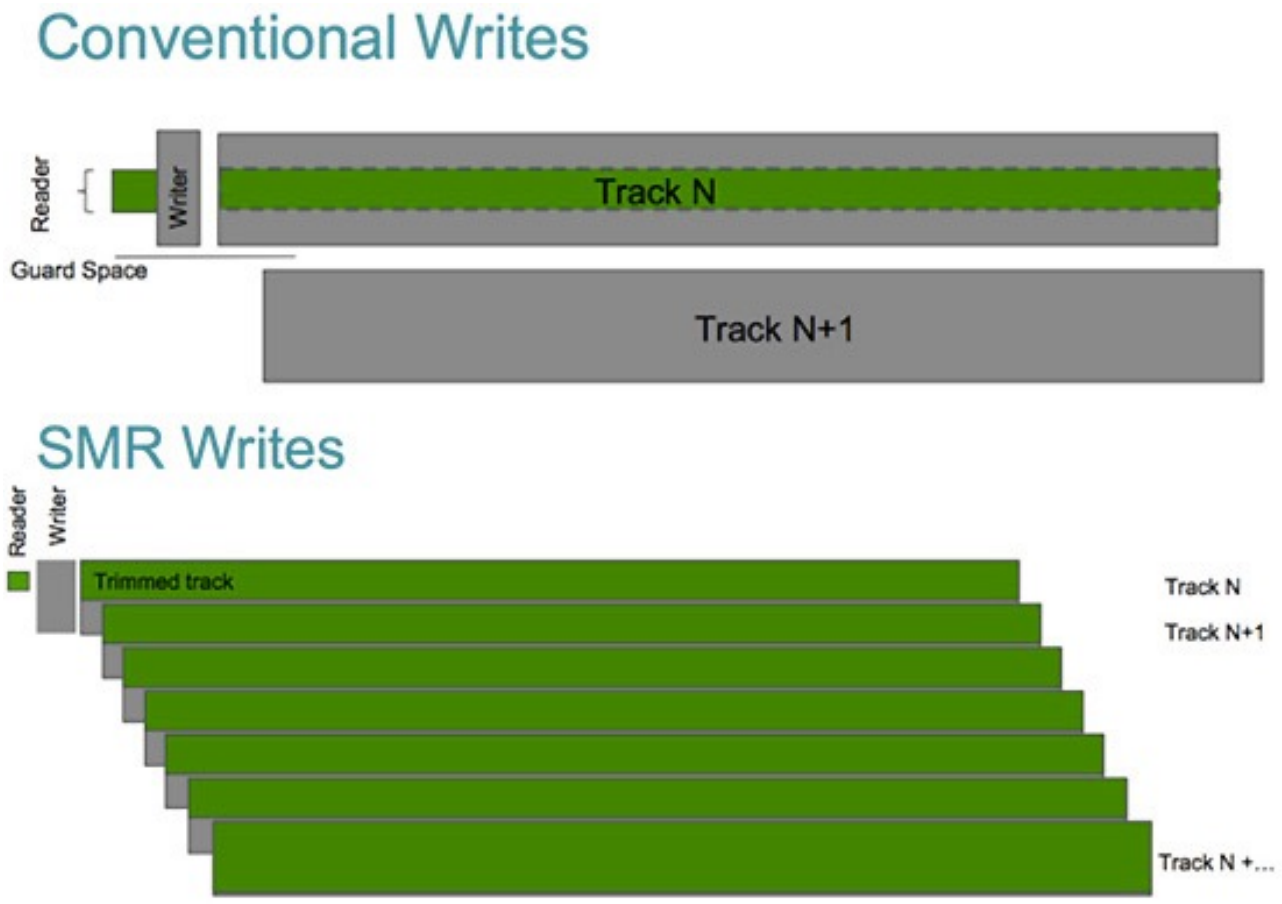
The Amazing Magnetic Disk

- 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

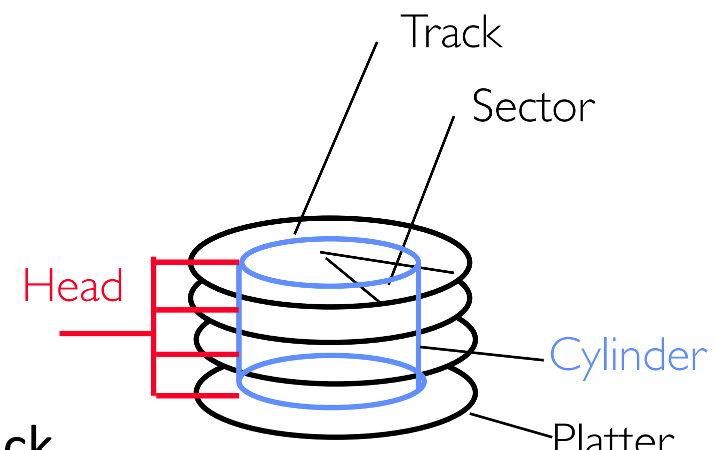


Shingled Magnetic Recording (SMR)

- Overlapping tracks yields greater density, capacity
- Restrictions on writing, complex DSP (Digital Signal Processing) for reading

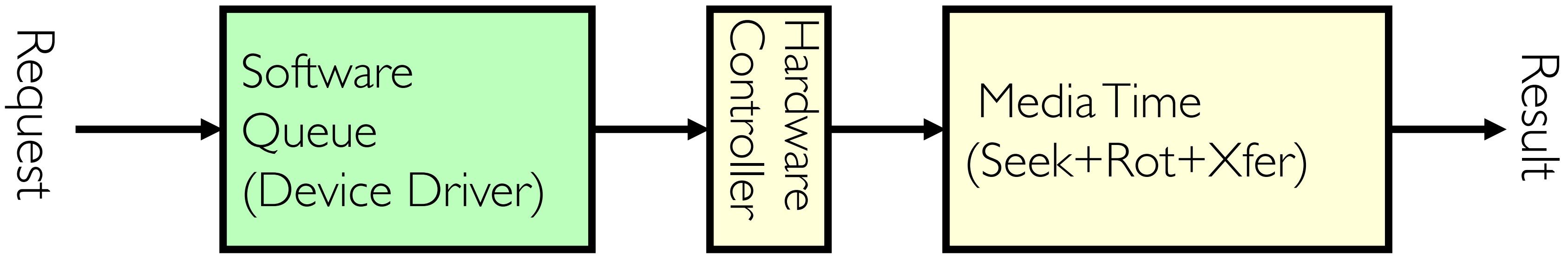


Magnetic Disks



- 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 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: ≥ **1 Terabit/square inch! (PMR, Helium, …)**
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 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 faster), now slowing down

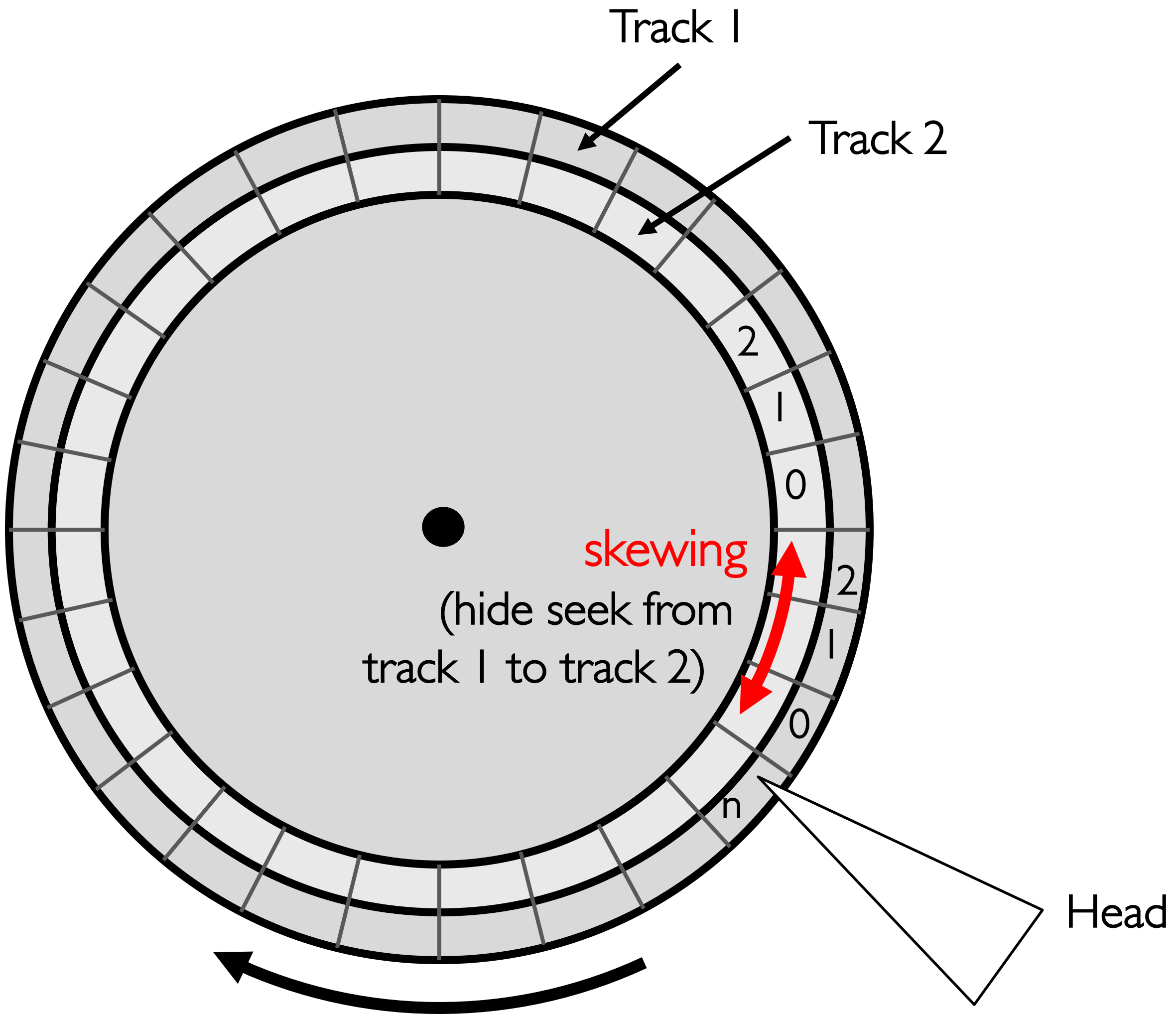
Disk Performance Example

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms
  - 7200RPM $\Rightarrow$ Time for rotation: 60000 (ms/min) / 7200(rev/min) ~= 8ms
  - Transfer rate of 50MByte/s, block size of 4Kbyte $\Rightarrow$
    4096 bytes/(50×10^6 (bytes/s)) = 81.92×10^{-6} sec $\cong$ 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 $\cong$ 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 $\cong$ 1.03MB/s
- Read next block on same track:
  - Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s $\cong$ 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
  - 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

Track Skewing



Example of Current HDDs

- Seagate Exos X18 (2020)
  - 18 TB hard disk
    - » 9 platters, 18 heads
    - » 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
    - » 270MB/s MAX transfer rate
    - » Cache size: 256MB
  - Price: $ 562 (~ $0.03/GB)

600K ×

10 ×

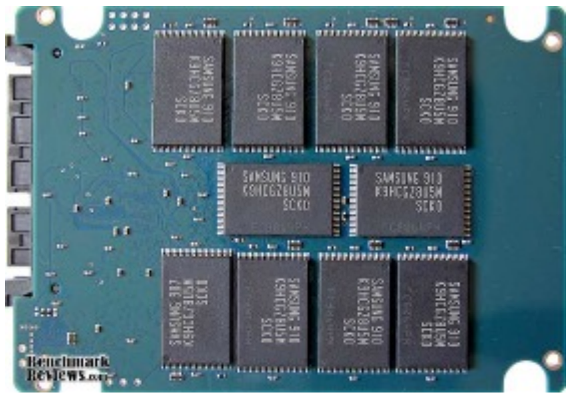
300 ×

567K ×

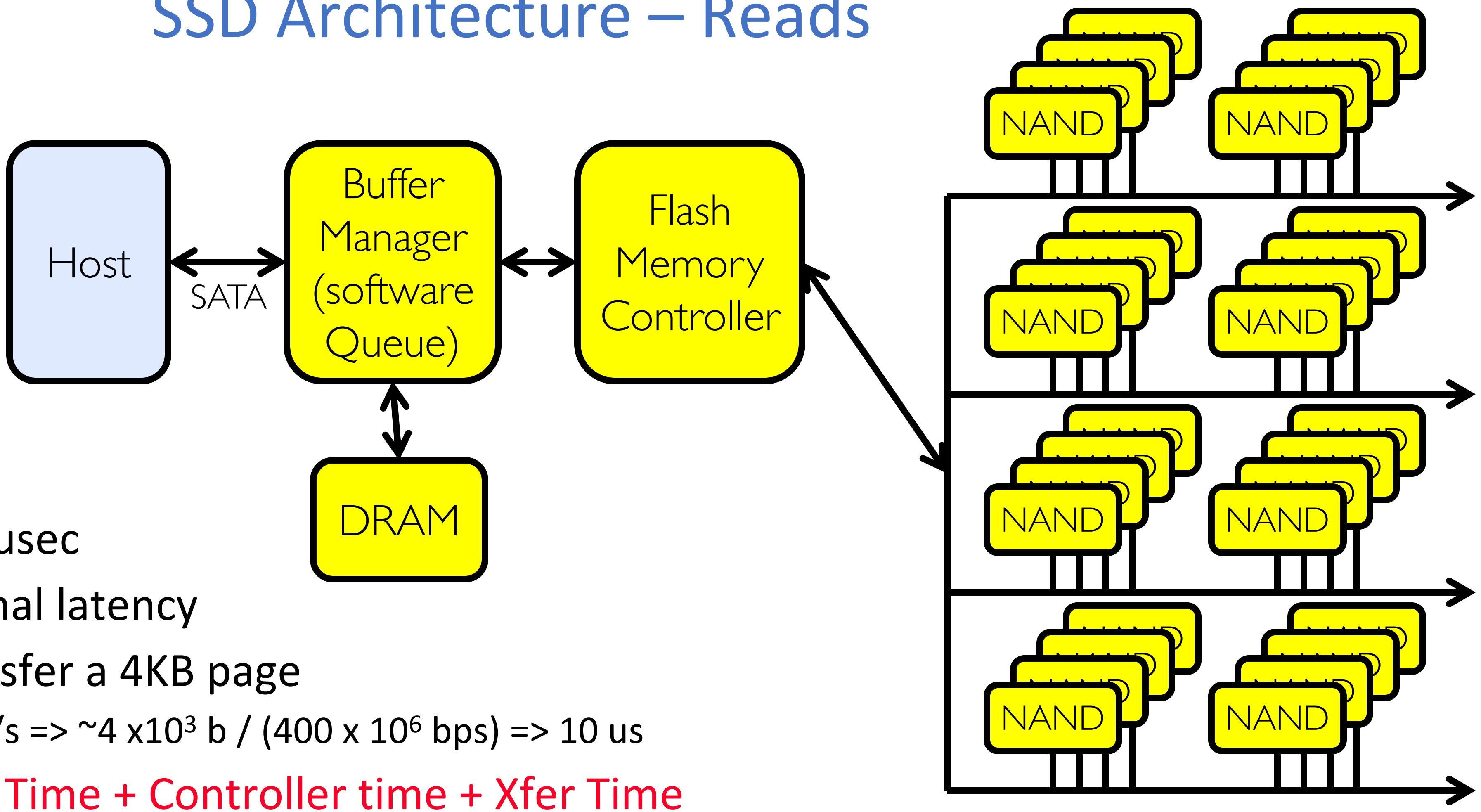
- IBM Personal Computer/AT (1986)
  - 30 MB hard disk
  - 30-40 ms seek time
  - 0.7-1 MB/s (est.)
  - Price: $500 ($17K/GB)

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



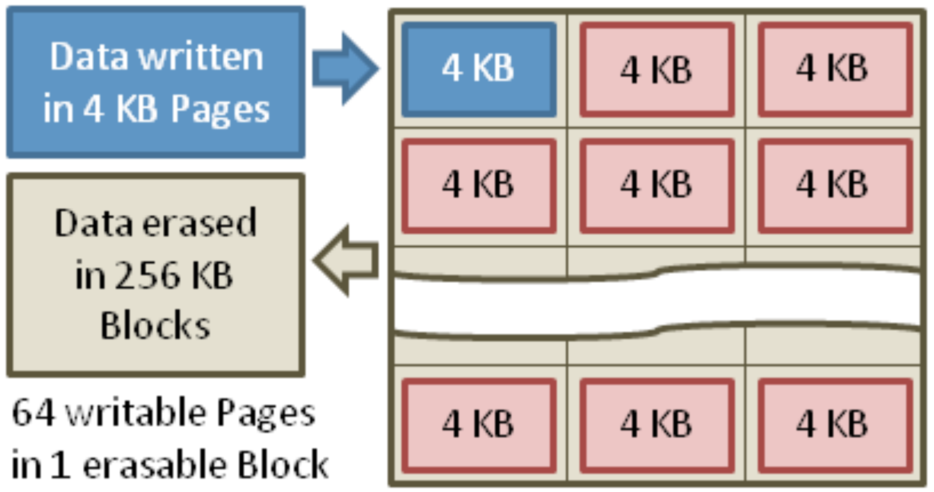
Read 4 KB Page: ~25 usec

– 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 memory 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 (this is slow as we need to erase page first!)
   - Instead, write new version in a free page
   - Update FTL mapping to point to new location

Flash Translation Layer

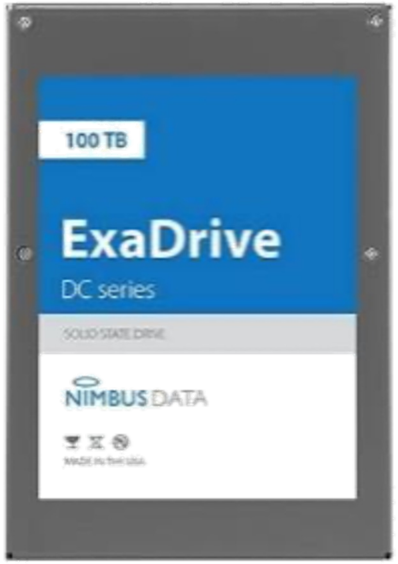
- No need to erase and rewrite entire 256KB block when making small modifications

- SSD controller can assign mappings to spread workload across pages
  - *Wear Levelling*

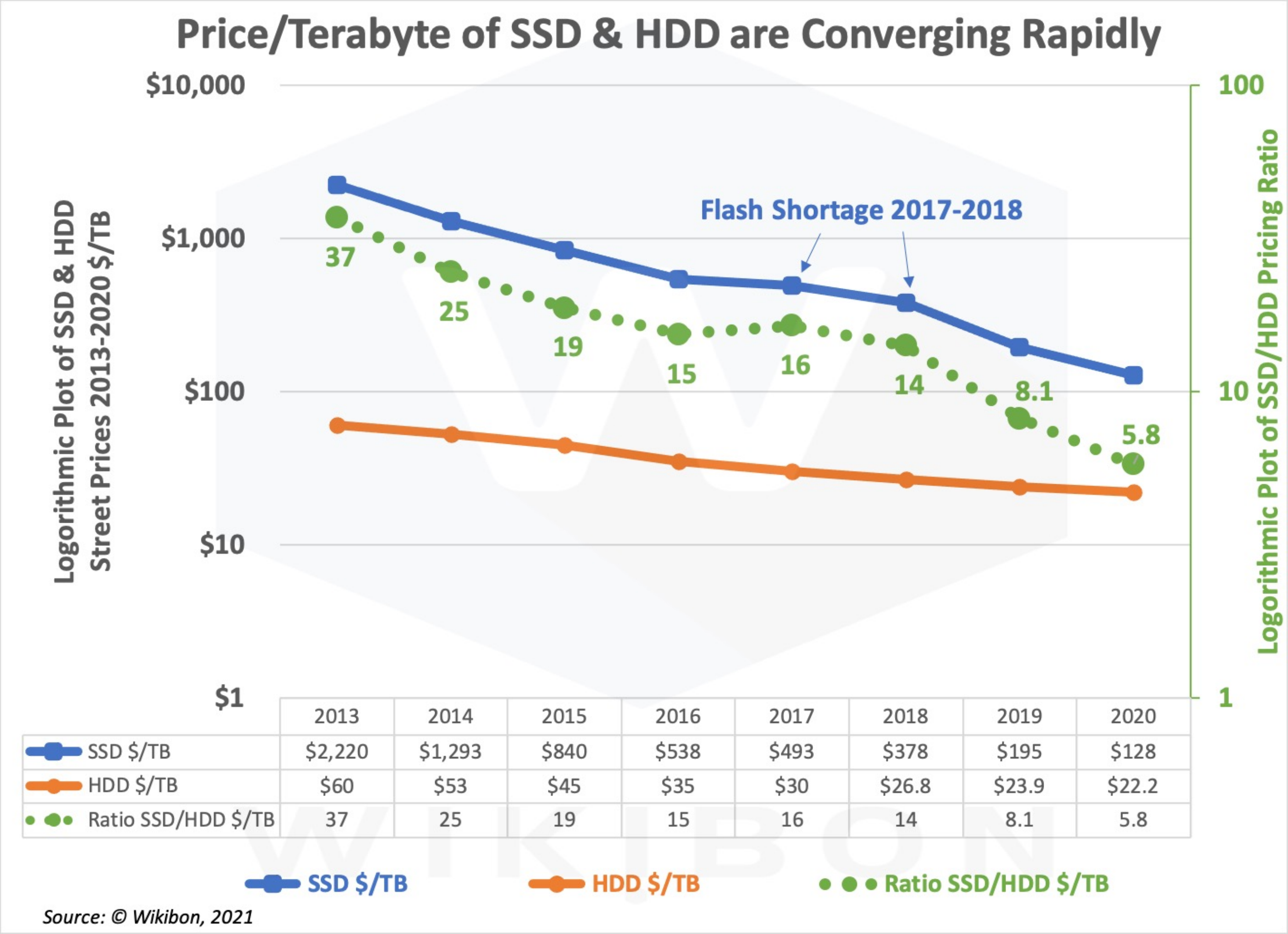
- What to do with old versions of pages?
  - *Garbage Collection* in background
  - Erase blocks with old pages, add to free list

Some "Current" (large) 3.5in SSDs

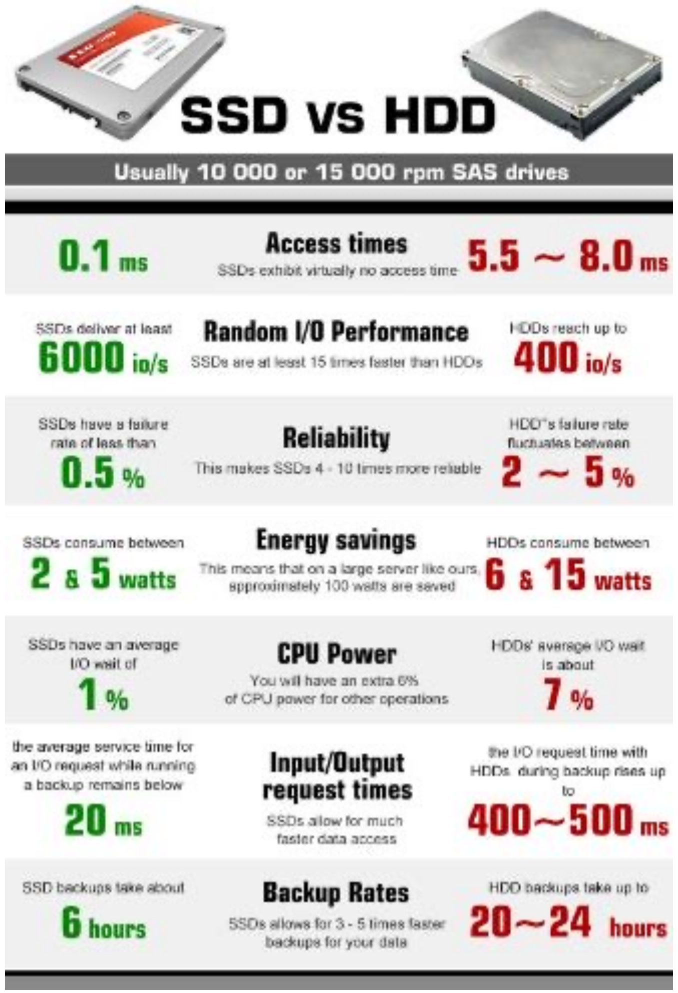
- Seagate Exos SSD: 15.36TB (2017)
  - Dual 12Gb/s interface
  - Sequential reads: 860MB/s
  - Sequential writes: 920MB/s
  - Random Reads (IOPS): 102K
  - Random Writes (IOPS): 15K
  - Price (Amazon): $5495 ($0.36/GB)
- Nimbus SSD: 100TB (2019)
  - Dual port: 12Gb/s interface
  - Sequential reads/writes: 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)



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

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

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 multi-level cell (MLC) NAND
    - » Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly!

No longer true!

Conclusion (1/2)

- 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

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)
- Systems (e.g., file system) designed to optimize performance and reliability
  - Relative to performance characteristics of underlying device