# Input/Output

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

# A Computer is Useless without I/O



- I/O handles persistent storage
  - Disks, SSD memory, etc
- I/O handles user interfaces
  - Keyboard/mouse/display
- I/O handles network

# Basic I/O:
# Devices are Memory



- The memory bus controller ("chipset") doesn't just control the memory...
  - But also controls the PCI I/O devices
- Provides a memory-mapped interface
  - Processor can write & read to control registers to tell the device what to do
  - Processor can write & read data to the device
- The "device driver" runs in the operating system
  - Provides a device independent abstraction to the rest of the OS
  - And user programs have to ask the OS for permission
- Programmed I/O:
  - The CPU is responsible to transfer data to the devices as well as commands

# So What Happens When Data Comes?



- I/O is asynchronous
  - It may occur at any time without coordination with what the OS is doing
- Option 1: Trigger an interrupt
  - Jumps control to the interrupt handler which has to figure out what to do...
- Option 2: Wait for the OS to poll the device
  - Do nothing, its the OS's job to check whether something is available

# More on Interrupt-Driven



- Highly responsive
  - When data comes in, the interrupt triggers
- Interesting efficiency tradeoff:
  - For low rate its ***very*** efficient
    - The computer is doing other things except when data comes
  - For high rates its inefficient
    - Interrupts are relatively expensive!
      You effectively ***have*** to nuke all cache state going to/from the OS
- Common design:
  - Interrupt drive by default
  - But if real fast is needed, poll instead
    - Multi-gigabit Network interfaces

# Working with real devices



- "Memory mapped I/O": Device control/data registers mapped to CPU address space
- CPU synchronizes with I/O device:
  - Polling
  - Interrupts
- "Programmed I/O"
  - CPU execs lw/sw instructions for all data movement to/from devices
  - CPU spends time doing 2 things:
    - Getting data from device to main memory
    - Using data to compute

# Working with real devices



- "Memory mapped I/O": Device control/data registers mapped to CPU address space
- CPU synchronizes with I/O device:
  - Polling
  - Interrupts
- ~~"Programmed I/O"~~: DMA
  - ~~CPU execs lw/sw instructions for all data movement to/from devices~~
  - CPU spends time doing 2 things:
    - ~~Getting data from device to main memory~~
    - Using data to compute

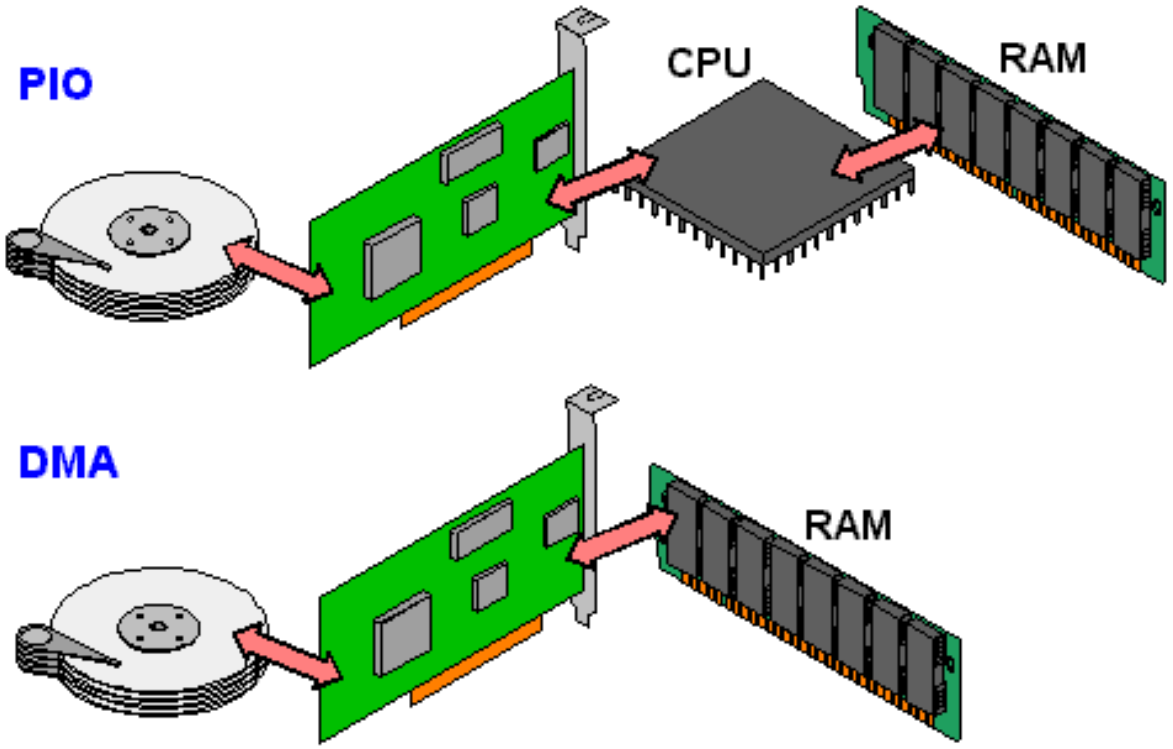
# What's wrong with Programmed I/O?



- Not ideal because …
  - CPU has to execute all transfers, could be doing other work
  - Device speeds don't align well with CPU speeds
  - Energy cost of using beefy general-purpose CPU where simpler hardware would suffice
- Until now CPU has sole control of main memory

# PIO vs. DMA





# Direct Memory Access (DMA)



- Allows I/O devices to directly read/write main memory
- New Hardware: the DMA Engine
- DMA engine contains registers written by CPU:
  - Memory address to place data
  - # of bytes
  - I/O device #, direction of transfer
  - unit of transfer, amount to transfer per burst

# Operation of a DMA Transfer



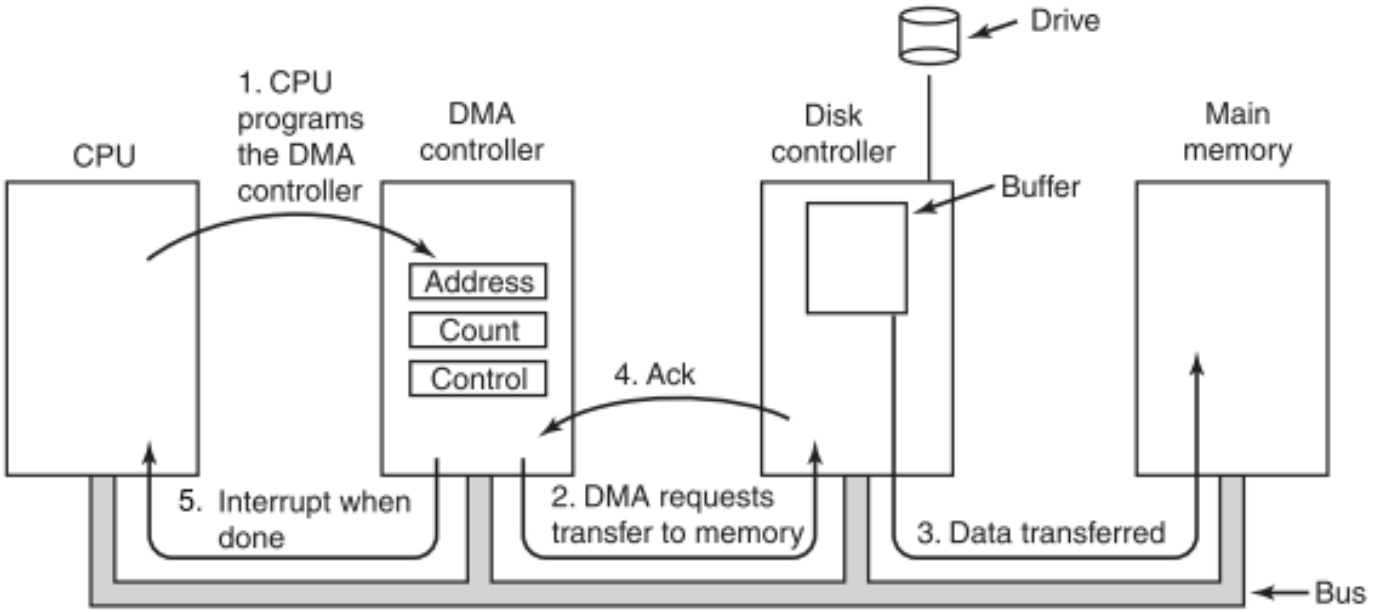



**Figure 5-4.** Operation of a DMA transfer.

[From Section 5.1.4 Direct Memory Access in *Modern Operating Systems* by Andrew S. Tanenbaum, Herbert Bos, 2014]

# DMA: Incoming Data



- Receive interrupt from device
- CPU takes interrupt, begins transfer
  - Instructs DMA engine/device to place data @ certain address
    - Even smarter devices can dispense with this first interrupt:
      Agreement with O/S that a particular hunk of memory is writeable by the device,
      Device just starts writing where it is supposed to
- Device/DMA engine handle the transfer
  - CPU is free to execute other things
- Upon completion, Device/DMA engine interrupt the CPU again

# DMA: Outgoing Data



- CPU decides to initiate transfer, confirms that external device is ready
- CPU begins transfer
  - Instructs DMA engine/device that data is available @ certain address
- Device/DMA engine handle the transfer
  - CPU is free to execute other things
- Device/DMA engine interrupt the CPU again to signal completion

# DMA: Some new problems



- Where in the memory hierarchy do we plug in the DMA engine? Two extremes:
  - Between L1 and CPU:
    - Pro: Free coherency
    - Con: Trash the CPU's working set with transferred data
  - Between Last-level cache and main memory:
    - Pro: Don't mess with caches
    - Con: Need to explicitly manage coherency
- Or just treat like another node in a multiprocessor
  - Cache-coherence is supported by most modern multiprocessors

# DMA: Some new problems



- How do we arbitrate between CPU and DMA Engine/Device access to memory? Three options:
  - Burst Mode
    - Start transfer of data block, CPU cannot access memory in the meantime
  - Cycle Stealing Mode
    - DMA engine transfers a byte, releases control, then repeats - interleaves processor/ DMA engine accesses
  - Transparent Mode
    - DMA transfer only occurs when CPU is not using the system bus

# DMA: Cool Interactions with Virtual Memory...



- We want to do high performance I/O for a ***unprivileged process***
  - Normally I/O requires context switching to the Operating System
- Idea: Map the DMA'ed regions into the user program's memory space!
  - High performance disk: "Memory map" the file. VM says "yeah, the file is in memory here", when you actually want to read it swaps it in from disk using a DMA transfer
  - High performance memory: "User-level buffers". The network card's I/O buffers are mapped into the user process's memory
- Result is now I/O operations don't even need to ask the OS once things are set up

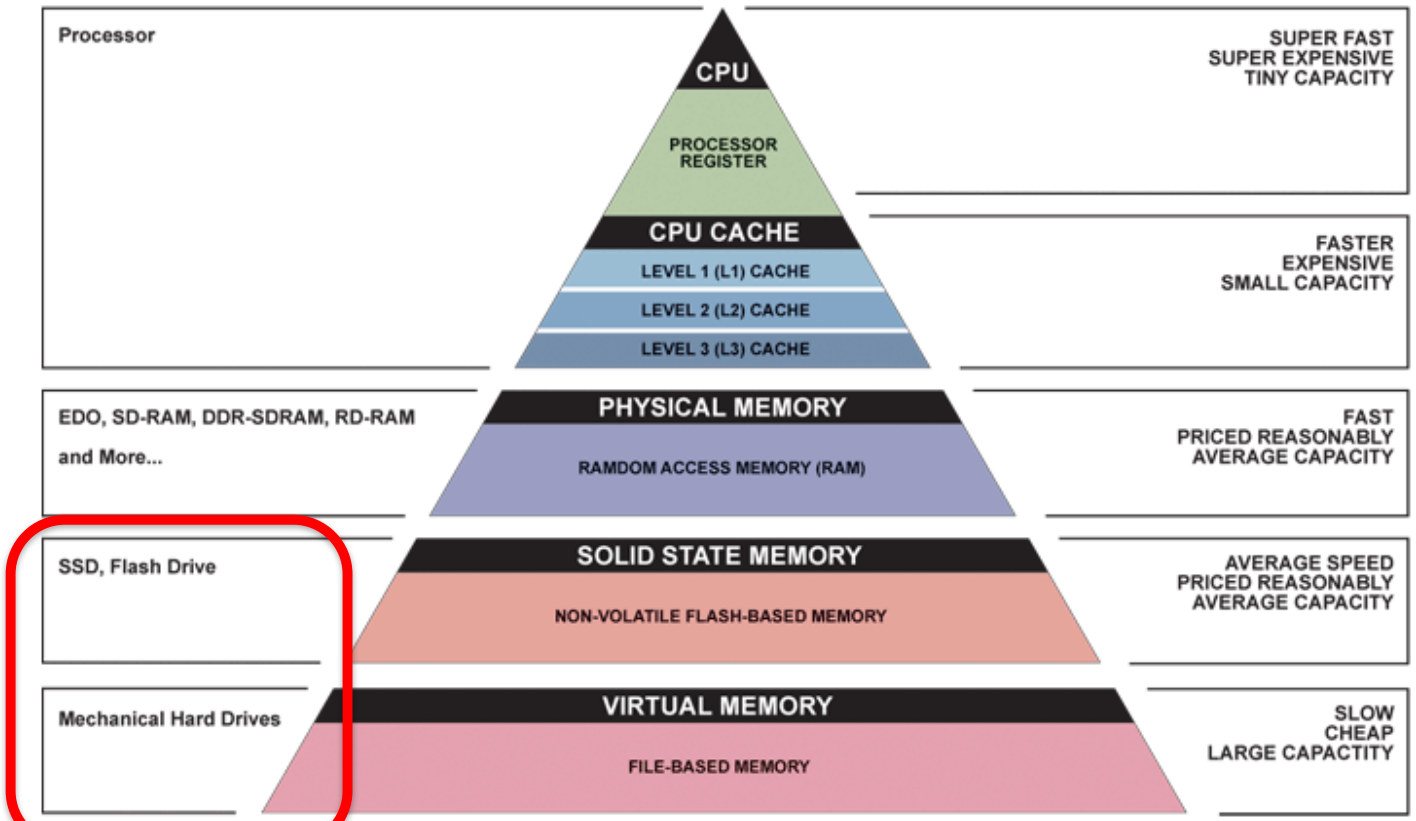
# Announcements



- Project 5-1 is out now
  - 5-2 out soon, both due 4:30 at 23:59.
  - Start ***now***: The big problem is making sure you get your environment working.
- HW5 due next Friday
- Guerilla section next thursday

# Computer Memory Hierarchy






▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

**Today**

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

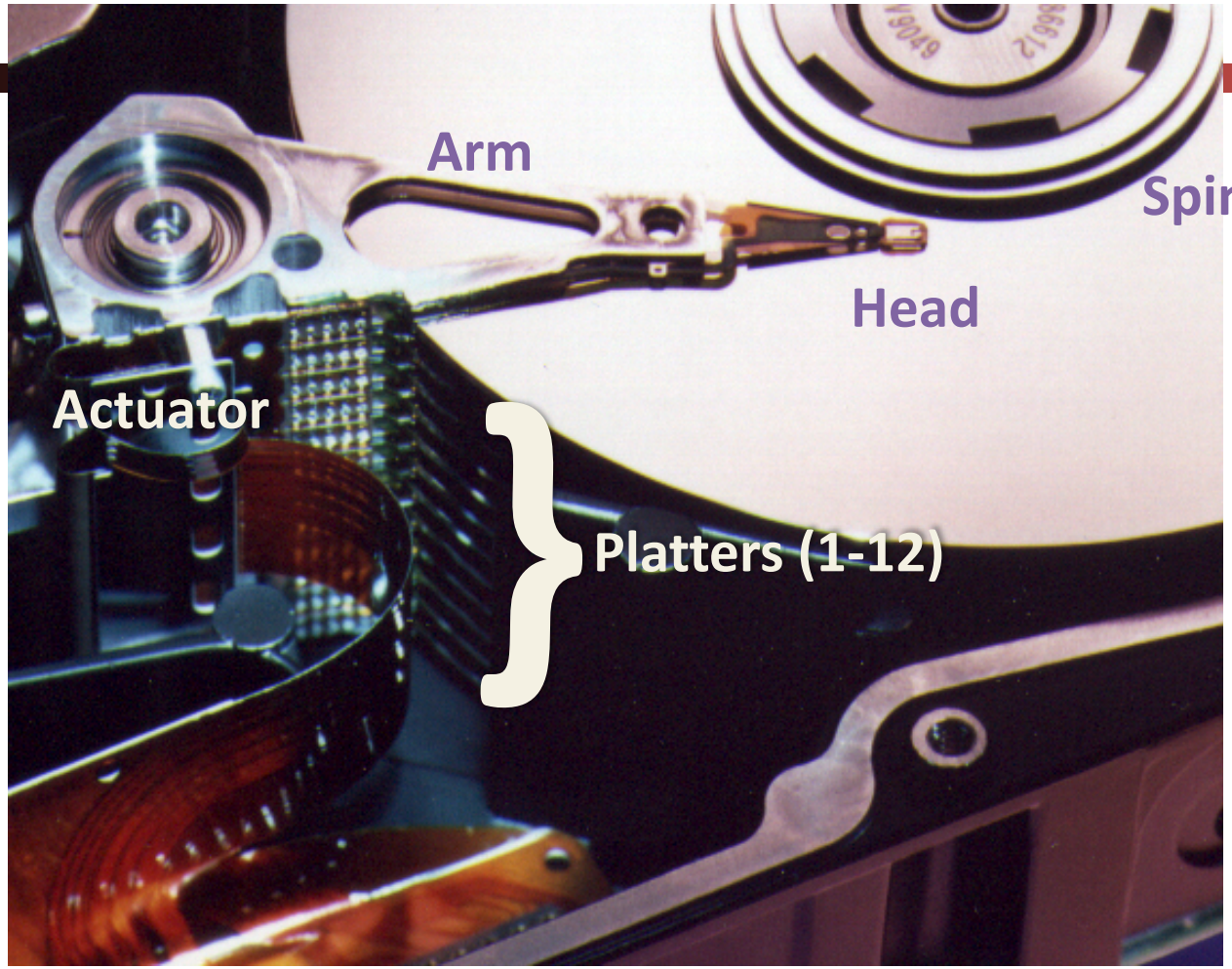
# Magnetic Disk – common I/O device



- A kind of computer memory
  - Information stored by magnetizing ferrite material on surface of rotating disk
    - Similar to tape recorder except digital rather than analog data
- A type of non-volatile storage
  - Retains its value without applying power to disk.
- Two Types of Magnetic Disk
  - Hard Disk Drives (HDD) – faster, more dense, non-removable.
  - Floppy disks – slower, less dense, removable (now replaced by USB "flash drive").
- Purpose in computer systems (Hard Drive):
  - Working file system + long-term backup for files
  - Secondary "backing store" for main-memory. Large, inexpensive, slow level in the memory hierarchy (virtual memory)

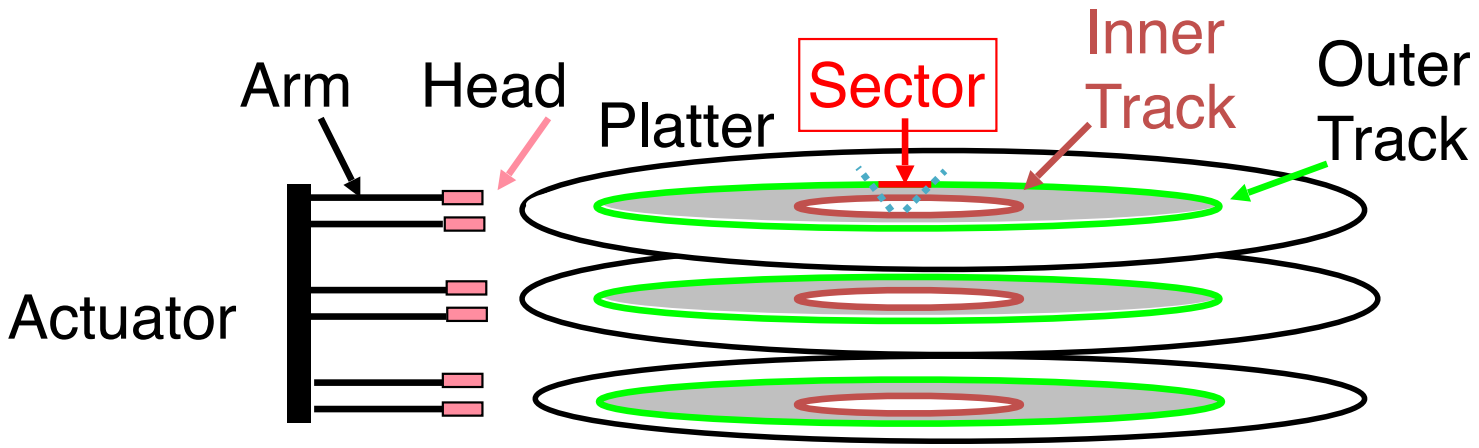
# Photo of Disk Head, Arm, Actuator





# Disk Device Terminology






- Several platters, with information recorded magnetically on both surfaces (usually)
- Bits recorded in tracks, which in turn divided into sectors (e.g., 512 Bytes)
- Actuator moves head (end of arm) over track ("seek"), wait for sector rotate under head, then read or write

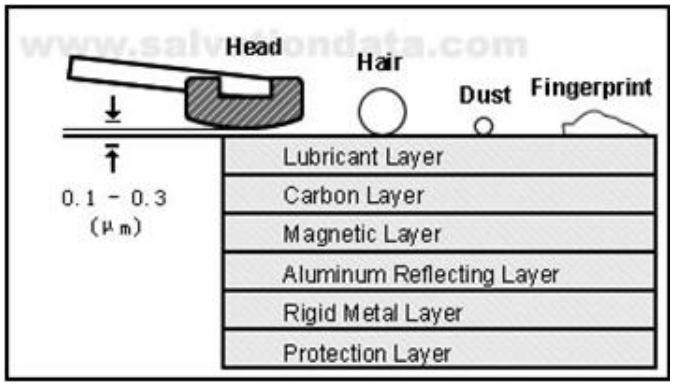
Video of hard disk in action

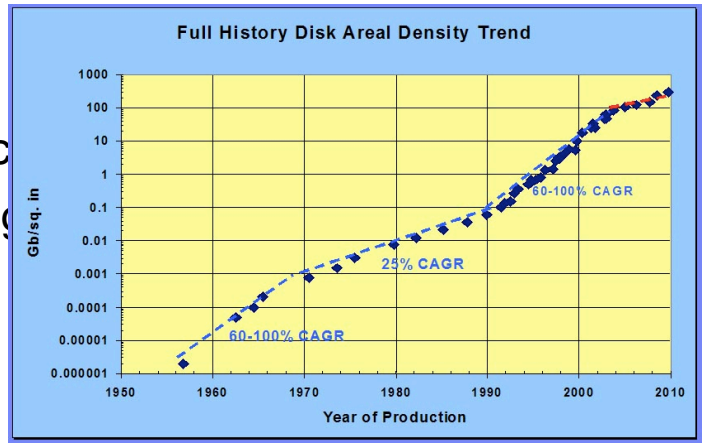
# Hard Drives are Sealed



- The closer the head to the disk, the smaller the "s denser the recording.
  - Measured in Gbit/in^2
  - ~900 Gbit/in^2 is state of the art
  - Started out at 2 Kbit/in^2
  - ~450,000,000x improvement in ~60 years
- Disks are sealed to keep the dust out.
  - Heads are designed to "fly" at around 3-20nm above the surfac
  - 99.999% of the head/arm weight is supported by the air bearing between the disk and the head
- Some drives are even sealed with Helium
  - Lower drag than air

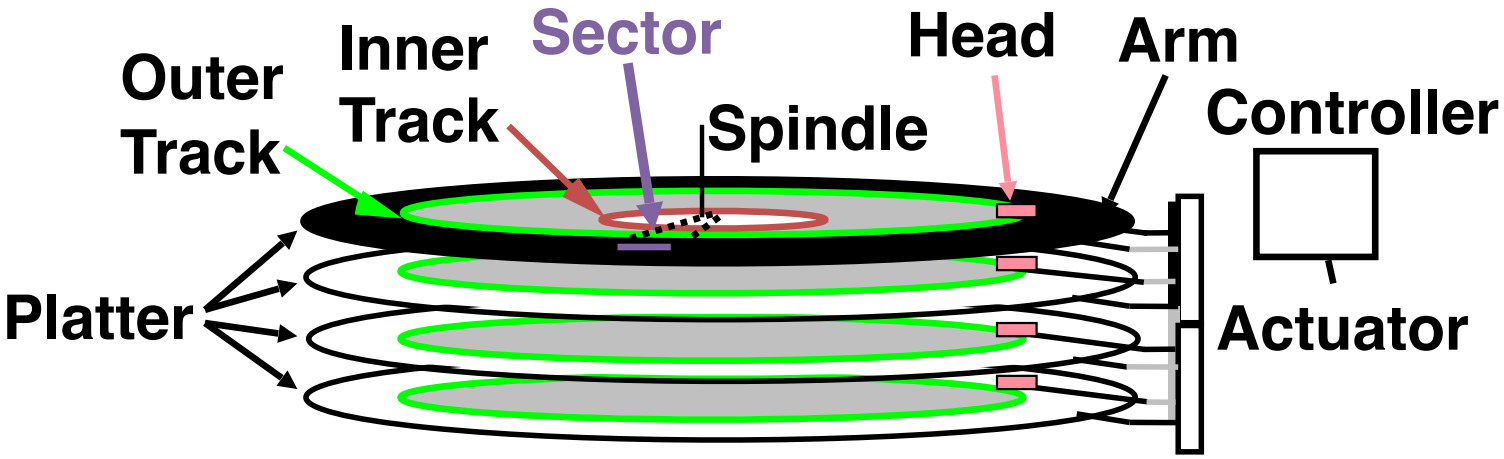
3-20nm



# Disk Device Performance (1/2)






- **Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead**
  - Seek Time = time to position the head assembly at the proper cylinder
  - Rotation Time = time for the disk to rotate to the point where the first sectors of the block to access reach the head
  - Transfer Time = time taken by the sectors of the block and any gaps between them to rotate past the head

# Disk Device Performance (2/2)



- Average values to plug into the formula:
- Rotation Time: Average distance of sector from head?
  - 1/2 time of a rotation
    - 7200 Revolutions Per Minute ⇒ 120 Rev/sec
    - 1 revolution = 1/120 sec ⇒ 8.33 milliseconds
    - 1/2 rotation (revolution) ⇒ 4.17 ms
- Seek time: Average no. tracks to move arm?
  - Number of tracks/3 (see CS186 for the math)
  - Then, seek time = number of tracks moved × time to move across one track

# But wait!

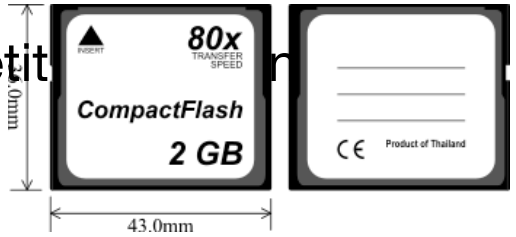


- Performance estimates are different in practice:
- Many disks have on-disk caches, which are completely hidden from the outside world
  - Previous formula completely replaced with on-disk cache access time

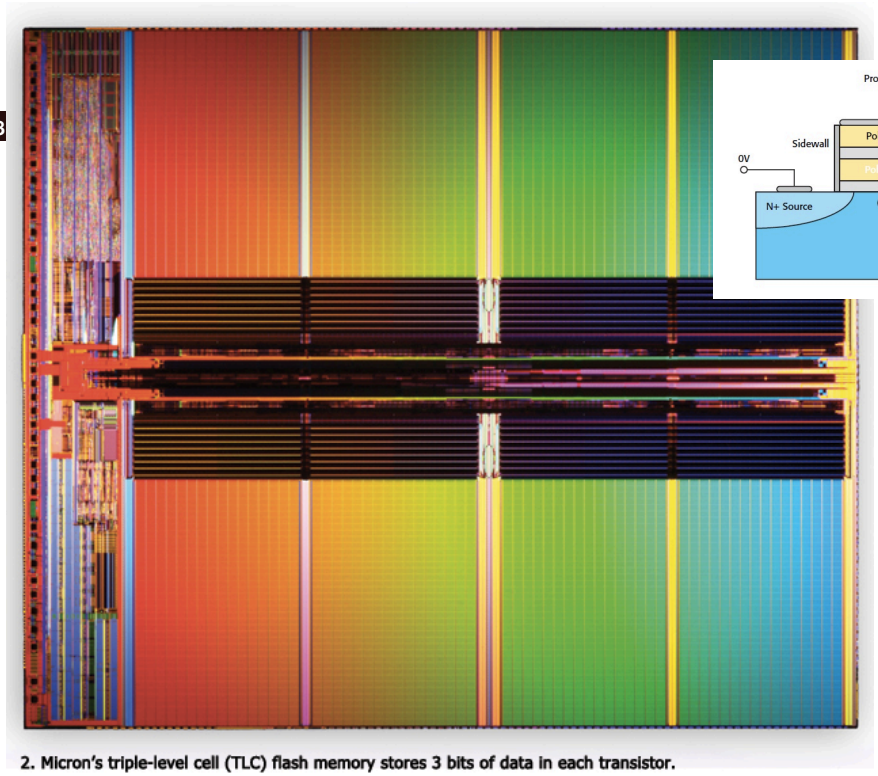
# Where does Flash memory come in?



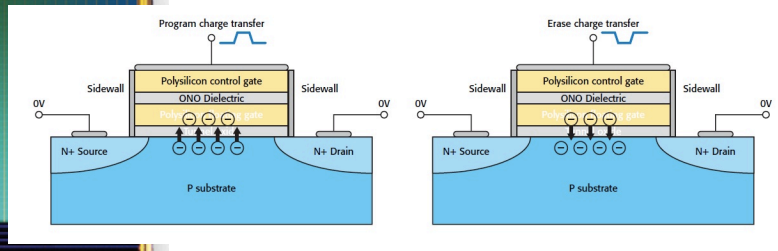
- ~10 years ago: Microdrives and Flash memory (e.g., CompactFlash) went head-to-head
  - Both non-volatile (retains contents without power supply)
  - Flash benefits: lower power, no crashes (no moving parts, need to spin µdrives up/down)
  - Disk cost = fixed cost of motor + arm mechanics, but actual [illegible]t very low
  - Flash cost = most cost/bit of flash chips
- Over time, cost/bit of flash came down, became cost competit[illegible]n storage

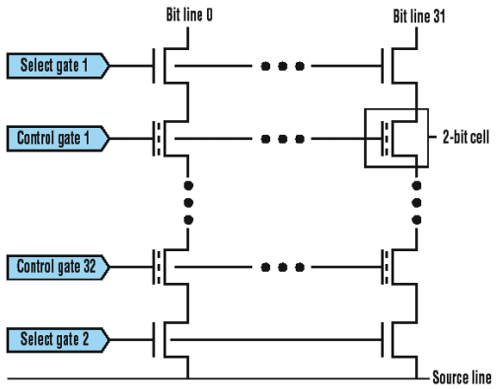



# Flash Memory / SSD Technology



2. Micron's triple-level cell (TLC) flash memory stores 3 bits of data in each transistor.




In the basic functional block used in multilevel NAND flash memories, 32 rows of bit lines and 32 control-gate lines form a building block that's repeated many times to form the memory array. The select gate lines are used with the control gate lines to control access to the array.

- NMOS transistor with an additional conductor between gate and source/drain which "traps" electrons. The presence/absence is a 1 or 0
- Memory cells can withstand a limited number of program-erase cycles. Controllers use a technique called *wear leveling* to distribute writes as evenly as possible across all the flash blocks in the SSD.

# Flash and Latency...



- Flash bandwidth is ~ to spinning disk
  - And spinning disk is still a lot bigger storage/$ and storage/cc
- But Flash's big party trick: ***no seek time!***
  - No additional latency for random access vs sequential access
- This is huge:
  - Operating system boot ***should be*** as linear as possible
    - Apple spent a huge amount of effort to optimize this on macs...
    - And when I swapped a spinning disk for an SSD, boot times cut in half...

# What did Apple put in its iPods?



Toshiba flash
2 GB


Samsung flash
16 GB


Toshiba 1.8-inch HDD
80, 120, 160 GB


Toshiba flash
32, 64, 128 GB


shuffle

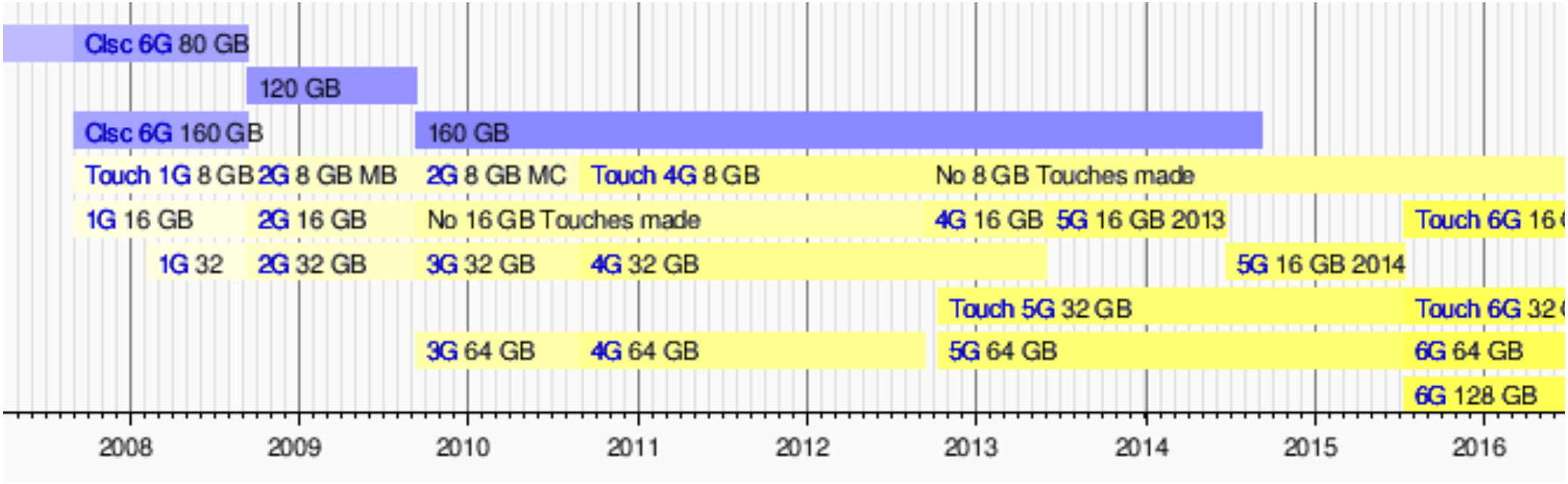
nano


classic

touch

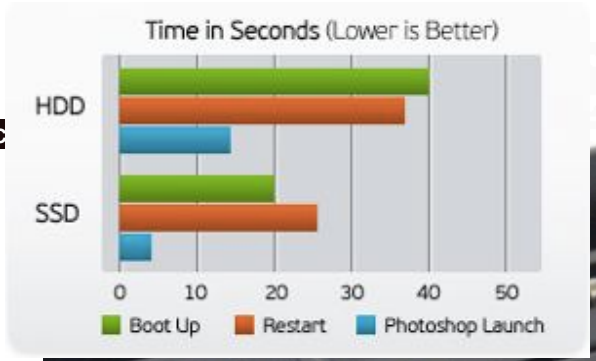
# And How Things Change...
# iPod Classic v Touch from Wikipedia





# Flash Memory in Smart Phones

# Flash Memory in Laptops – Solid State Drive (SSD)



# iClicker Question



- We have the following disk:
  - 15000 Cylinders, 1 ms to cross 1000 Cylinders
  - 15000 RPM = 4 ms per rotation
  - Want to copy 1 MB, transfer rate of 1000 MB/s
  - 1 ms controller processing time
- What is the access time using our model?

**Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Processing Time**

| A | B | C | D | E |
|---|---|---|---|---|
| 10.5 ms | 9 ms | 8.5 ms | 11.4 ms | 12 ms |

# iClicker Question



- We have the following disk:
  - 15000 Cylinders, 1 ms to cross 1000 Cylinders
  - 15000 RPM = 4 ms per rotation
  - Want to copy 1 MB, transfer rate of 1000 MB/s
  - 1 ms controller processing time
- What is the access time?

Seek = # cylinders/3 * time = 15000/3 * 1ms/1000 cylinders = 5ms

Rotation = time for ½ rotation = 4 ms / 2 = 2 ms

Transfer = Size / transfer rate = 1 MB / (1000 MB/s) = 1 ms

Controller = 1 ms

Total = 5 + 2 + 1 + 1 = 9 ms

# Networks: Talking to the Outside World

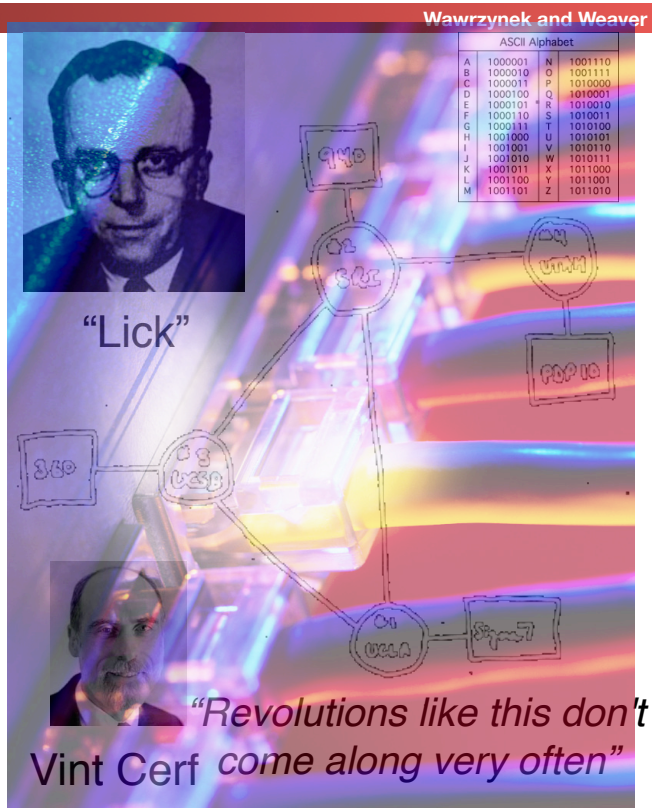


- Originally sharing I/O devices between computers
  - E.g., printers
- Then communicating between computers
  - E.g., file transfer protocol
- Then communicating between people
  - E.g., e-mail
- Then communicating between networks of computers
  - E.g., file sharing, www, …
- Then turning multiple cheap systems into a single computer
  - Warehouse scale computing

# The Internet (1962) www.computerhistory.org/internet_history



- History
  - 1963: JCR Licklider, while at DoD's ARPA, writes a memo describing desire to connect the computers at various research universities: Stanford, Berkeley, UCLA, ...
  - 1969 : ARPA deploys 4 "nodes" @ UCLA, SRI, Utah, & UCSB
  - 1973 Robert Kahn & Vint Cerf invent TCP, now part of the Internet Protocol Suite
- Internet growth rates
  - Exponential since start
  - But finally starting to hit human scale limits although lots of silicon cockroaches...




www.greatachievements.org/?id=3736
en.wikipedia.org/wiki/Internet_Protocol_Suite

# The World Wide Web (1989)

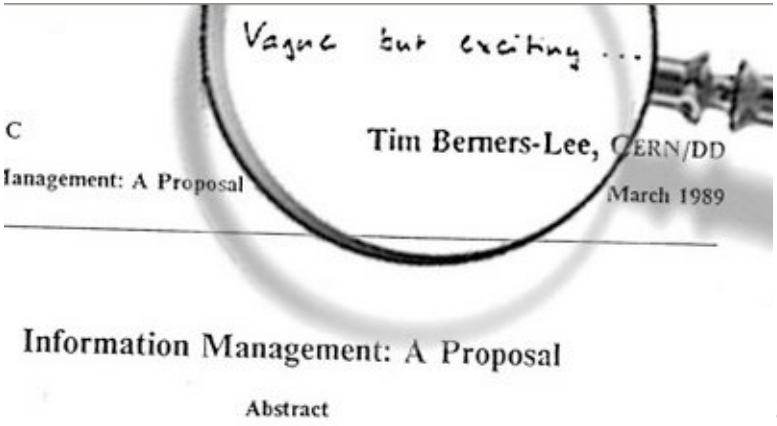
**en.wikipedia.org/wiki/History_of_the_World_Wide_Web**



- "System of interlinked hypertext documents on the Internet"
- History
  - 1945: Vannevar Bush describes hypertext system called "memex" in article
  - 1989: Sir Tim Berners-Lee proposed and implemented the first successful communication between a Hypertext Transfer Protocol (HTTP) client and server using the internet.
  - 1993: NCSA Mosaic: A graphical HTTP client
  - ~2000 Dot-com entrepreneurs rushed in, 2001 bubble burst
- Today : Access anywhere!

Tim Berners-Lee

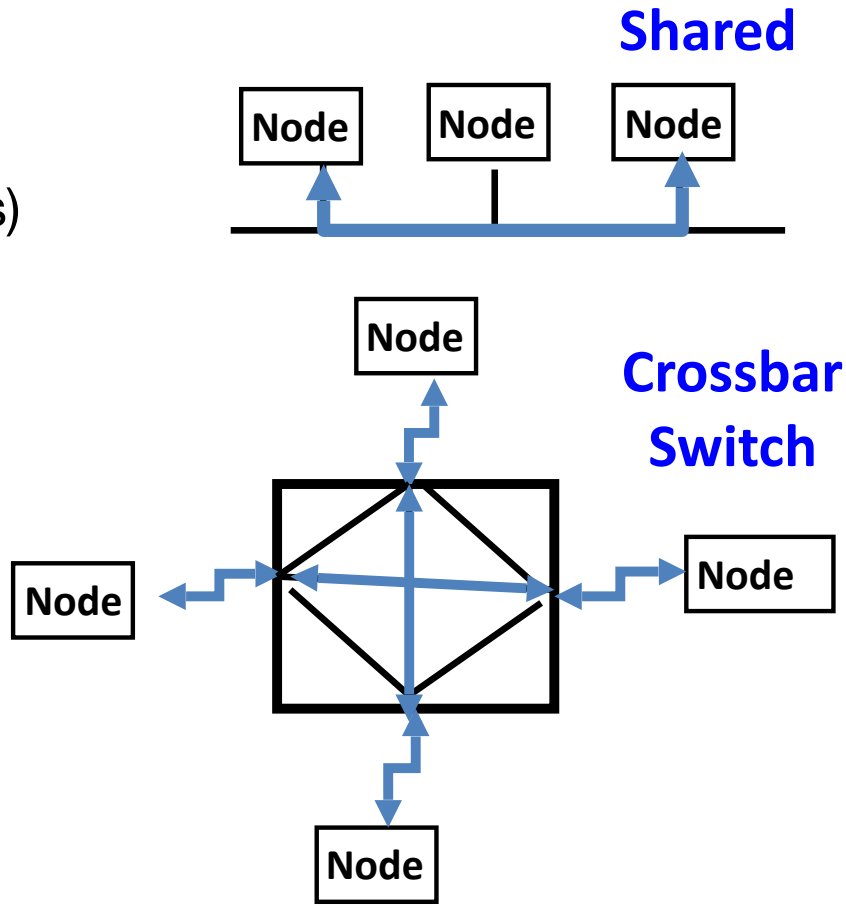
World's First web server in 1990



# Shared vs. Switch-Based Networks



- Shared vs. Switched:
  - Shared: 1 at a time (CSMA/CD)
  - Switched: pairs ("point-to-point" connections) communicate at same time
- Aggregate bandwidth (BW) in switched network is many times that of shared:
  - point-to-point faster since no arbitration, simpler interface
- Wired is almost always switched
- Wireless is by definition shared



# Shared Broadcast

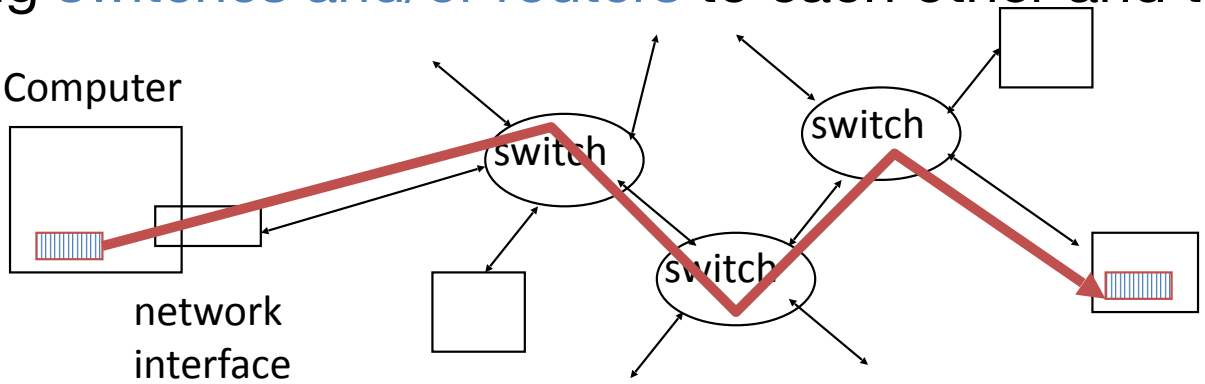


- Old-school Ethernet and Wireless
  - It doesn't just share but all others can see the request?
- How to handle access:
  - Old when I was old skool: Token ring
    - A single "token" that is passed around
  - Ethernet:
    - Listen and send
    - Randomized retry when someone else interrupts you
  - Cable Modem:
    - "Request to send": small request with a listen and send model
    - Big transfers then arbitrated

# What makes networks work?



- links connecting switches and/or routers to each other and to computers or devices

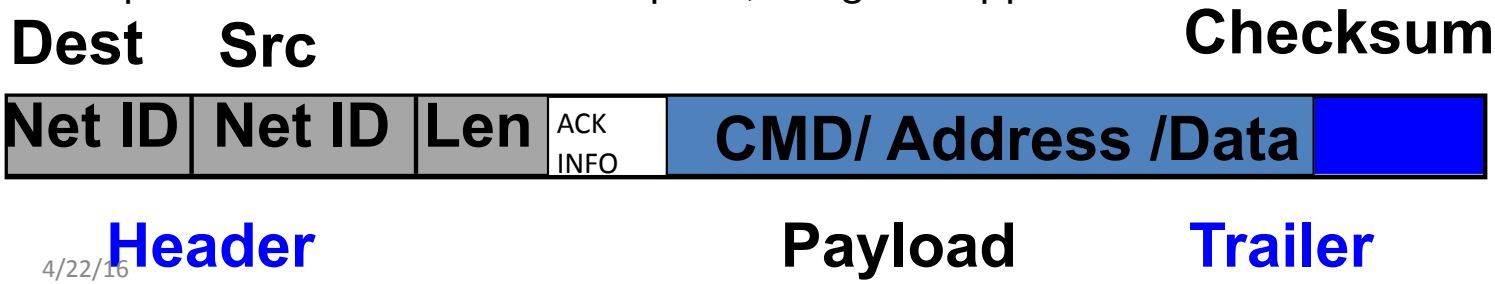



- ability to name the components and to route packets of information - messages - from a source to a destination
- Layering, redundancy, protocols, and encapsulation as means of abstraction (61C big idea)

# Software Protocol to Send and Receive



- SW Send steps
  - 1: Application copies data to OS buffer
  - 2: OS calculates checksum, starts timer
  - 3: OS sends DMA request to network interface HW and says start
- SW Receive steps
  - 3: Network interface copies data from network interface HW to OS buffer, triggers interrupt
  - 2: OS calculates checksum, if OK, send ACK; if not, delete message (sender resends when timer expires)
  - 1: If OK, OS copies data to user address space, & signals application to continue



# ***Protocols*** for Networks of Networks?



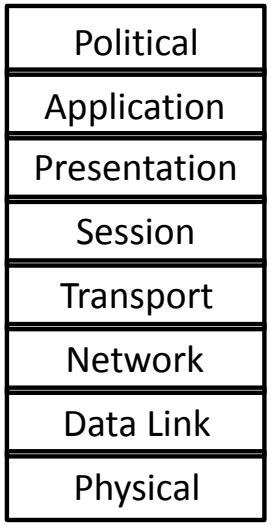
What does it take to send packets across the globe?

- Bits on wire or air
- Packets on wire or air
- Delivery packets within a single physical network
- Deliver packets across multiple networks
- Ensure the destination received the data
- Create data at the sender and make use of the data at the receiver

# The OSI 7 Layer Network Model



- A conceptual "Stack"
  - Physical Link: eg, the wires/wireless
  - Data Link: Ethernet
  - Network Layer: IP
  - Transport Layer: TCP/UDP
  - Session Layer/Presentation Layer/Application Layer
  - Political Layer: "Feinstein/Burr 'thou shalt not encrypt'"
    - Nick is starting to spend way too much time on "layer 8" issues



# Protocol Family Concept



- Protocol: packet structure and control commands to manage communication
- Protocol families (suites): a set of cooperating protocols that implement the network stack
- Key to protocol families is that communication occurs logically at the same level of the protocol, called peer-to-peer…
  …but is implemented via services at the next lower level
- Encapsulation: carry higher level information within lower level "envelope"

# Inspiration…



- CEO A writes letter to CEO B
  - Folds letter and hands it to assistant
- Assistant:
  - Puts letter in envelope with CEO B's full name
  - Takes to FedEx
- FedEx Office
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- FedEx delivers to other company

***Dear John,***

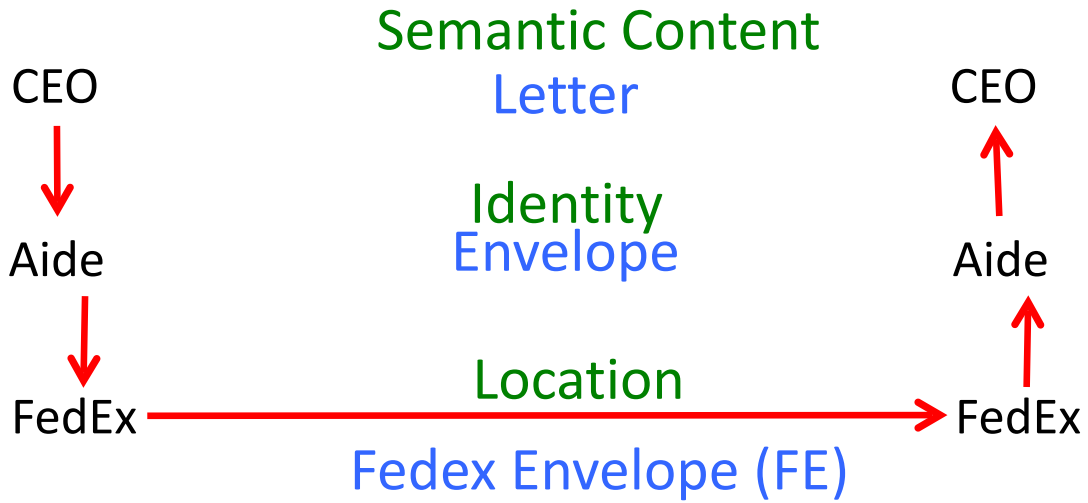
***Your days are numbered.***

***--Pat***

# The Path of the Letter

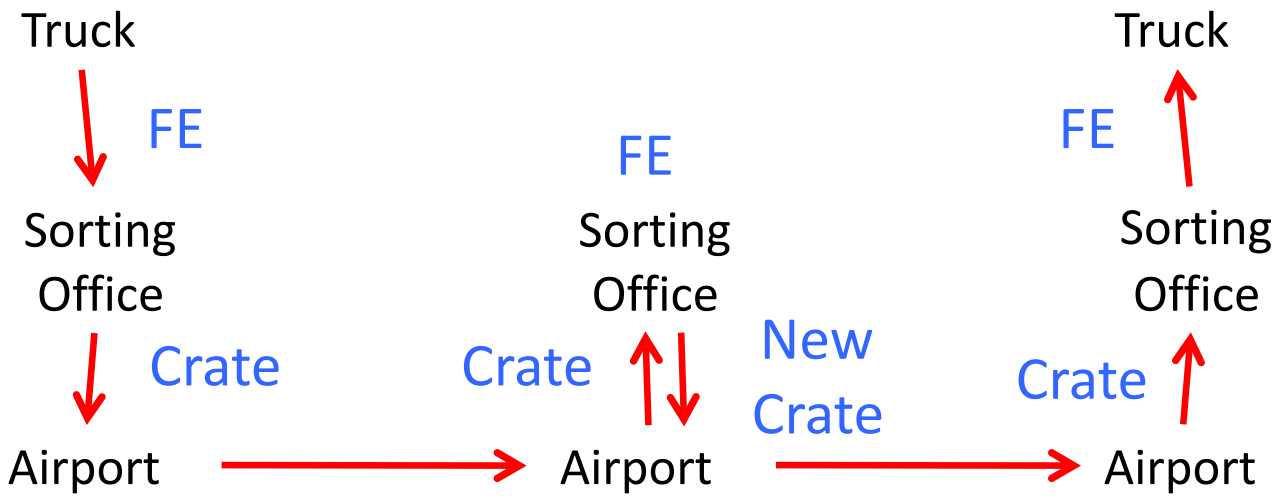


"Peers" on each side understand the same things
No one else needs to
Lowest level has most packaging



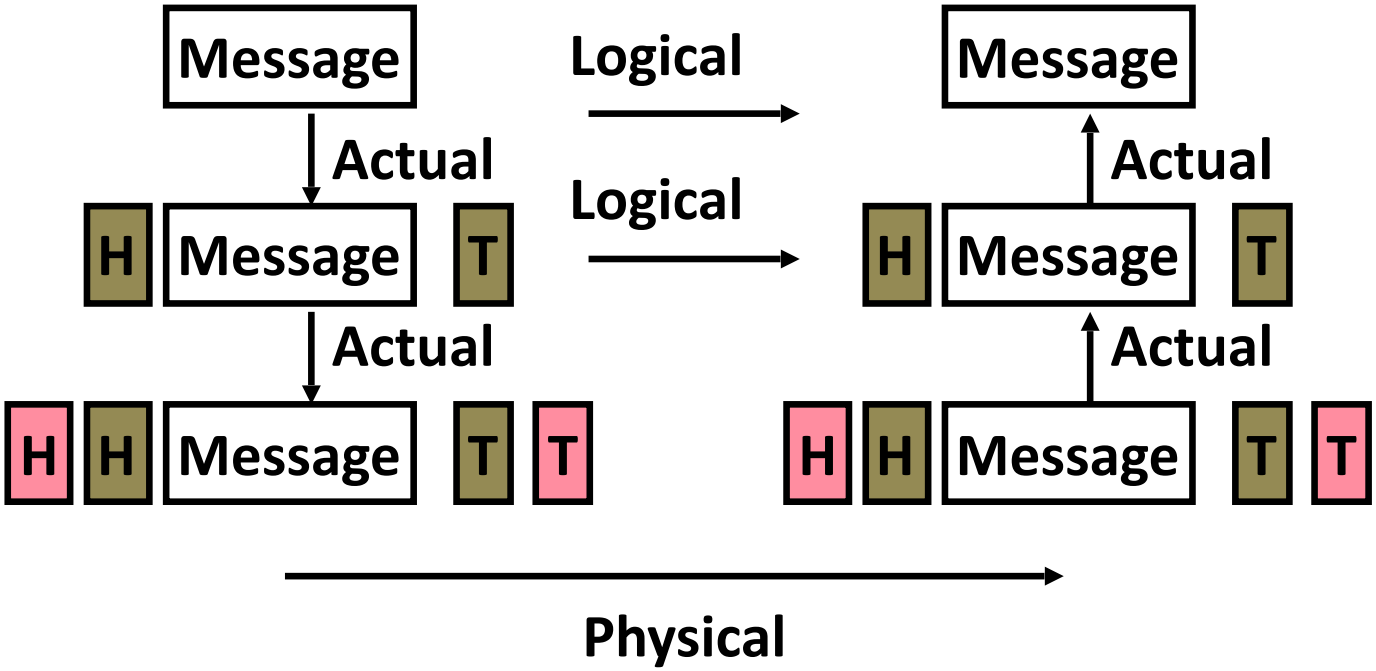
# The Path Through FedEx





# Protocol Family Concept






*Each lower level of stack "encapsulates" information from layer above by adding header and trailer.*

# Most Popular Protocol for Network of Networks

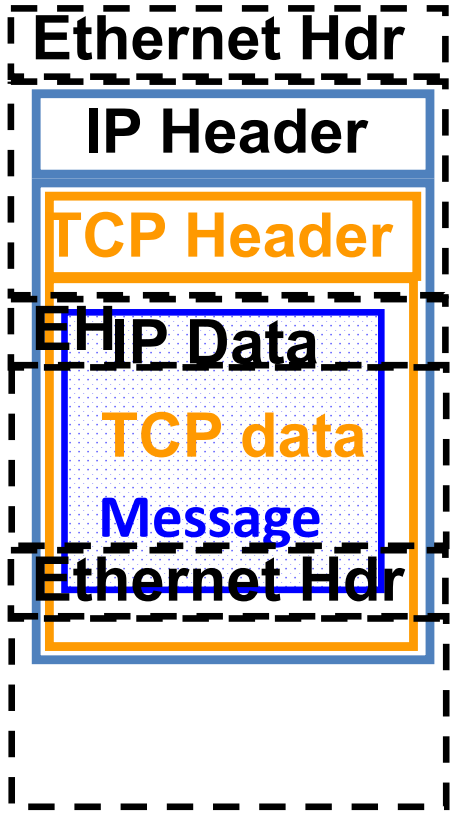


- Transmission Control Protocol/Internet Protocol (TCP/IP)
- This protocol family is the basis of the Internet, a WAN (wide area network) protocol
  - IP makes best effort to deliver
    - Packets can be lost, corrupted
      - But corrupted packets should be turned into lost packets
  - TCP guarantees ***reliable, in-order*** delivery
  - TCP/IP so popular it is used even when communicating locally: even across homogeneous LAN (local area network)

# TCP/IP packet, Ethernet packet, protocols



- Application sends message
  - **TCP breaks into 64KiB segments, adds 20B header**
  - **IP adds 20B header, sends to network**
  - **If Ethernet, broken into 1500B packets with headers, trailers**



# TCP and UDP
# The Two Internet Transfer Protocols



- TCP: Transmission Control Protocol
  - Connection based
    - SYN->SYN/ACK->ACK 3-way handshake
  - In order & reliable
    - All data is acknowledged
    - Programming interface is streams of data
- UDP: Universal Datagram Protocol
  - Datagram based
    - Just send messages
  - Out of order & unreliable
    - Datagrams can arrive in any order or be dropped (but not corrupted)
  - Needed for realtime protocols

# And Switching Gears: GPIO



- We see how to do high performance I/O
  - CPU has data it wants to send in main memory
  - Configures device & DMA controller to initiate transfer
    - Device then receives the data through DMA
- We have moderate bandwidth, flexible I/O
  - Universal Serial Bus is really a lightweight network for your slower peripheral devices
- But what about human scale?
  - With people, we only need to react in milliseconds to hours

# Reminder: Amdahl's Law and Programming Effort

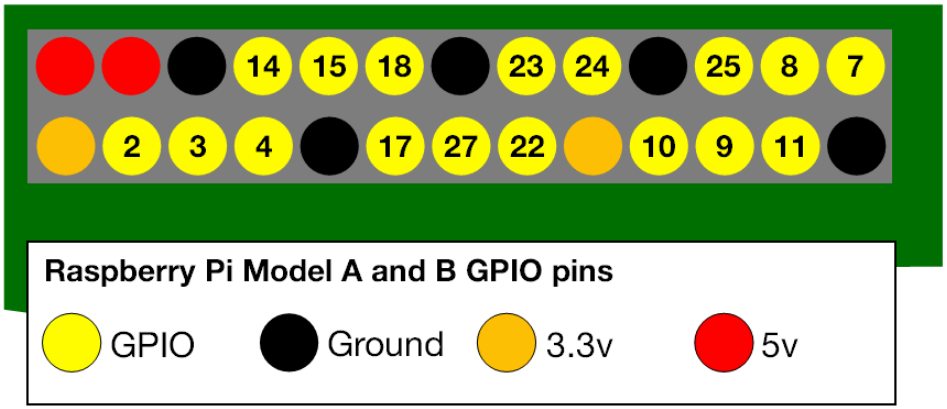


- Don't optimize where you don't need to
  - And if I only need to react at kHz granularity...
    But my processor is a GHz...
- I have 1 million clock cycles to actually decide what to do!
- So lets provide a simple interface

# Raspberry Pi GPIO



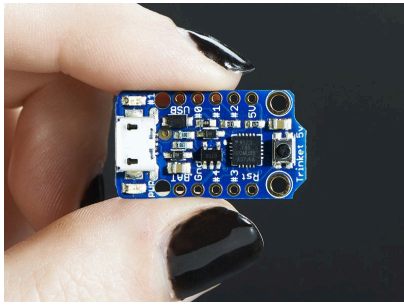
- A set of physical pins hooked up to the CPU
  - The CPU can write and read these pins as memory, like any other I/O device
- But that is a low level pain for us humans...
  - So the Linux instillation provides "files" that can access GPIO
  - You can literally write a 1 or a 0 to a pin or read the value at a pin
- Plus faster & still simple APIs



# Using GPIO



- There are a lot of add-on cards...
  - EG, ones for controlling servos
- Or you can build your own
- Combined with USB provides very powerful glue...
- Similarly some even smaller devices:
  - Adafruit "Trinket": 8 MHz 8-bit microcontroller, 5 GPIO pins



# “And in conclusion…”



- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- Polling vs. Interrupts
- DMA to avoid wasting CPU time on data transfers
- Disks for persistent storage, replaced by flash
- Network for communicating to the rest of the planet