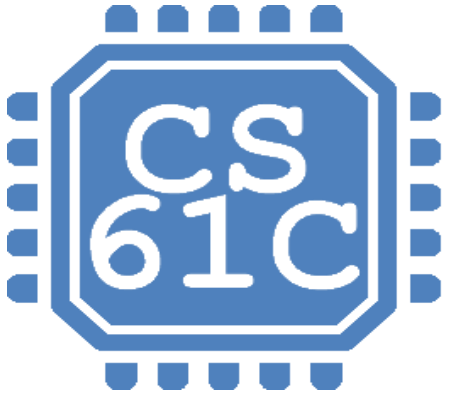
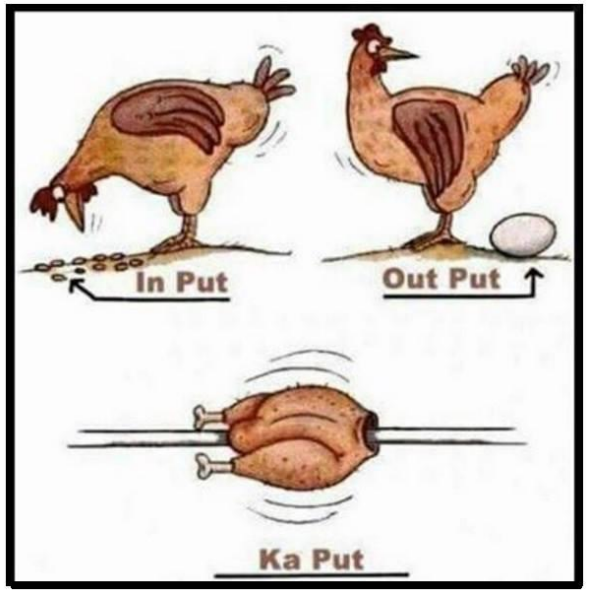


Great Ideas in Computer Architecture

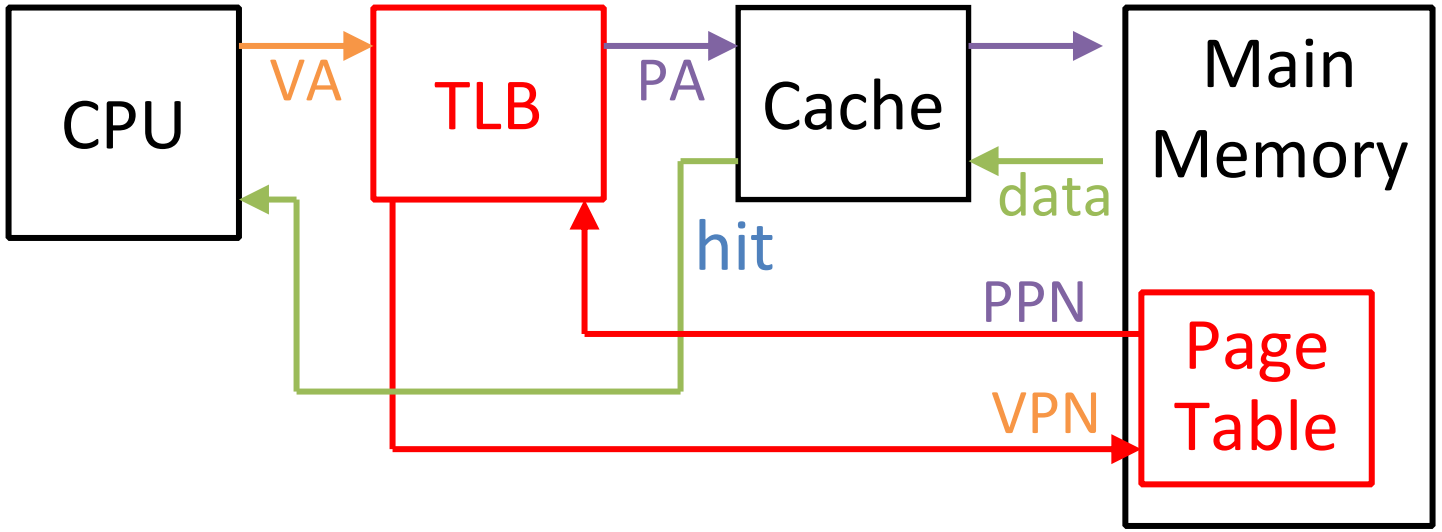
Input/Output

Instructor: Sean Farhat



Review

- Virtual Memory
  - Page Table maps virtual to physical addresses
  - TLB is a cache for the Page Table

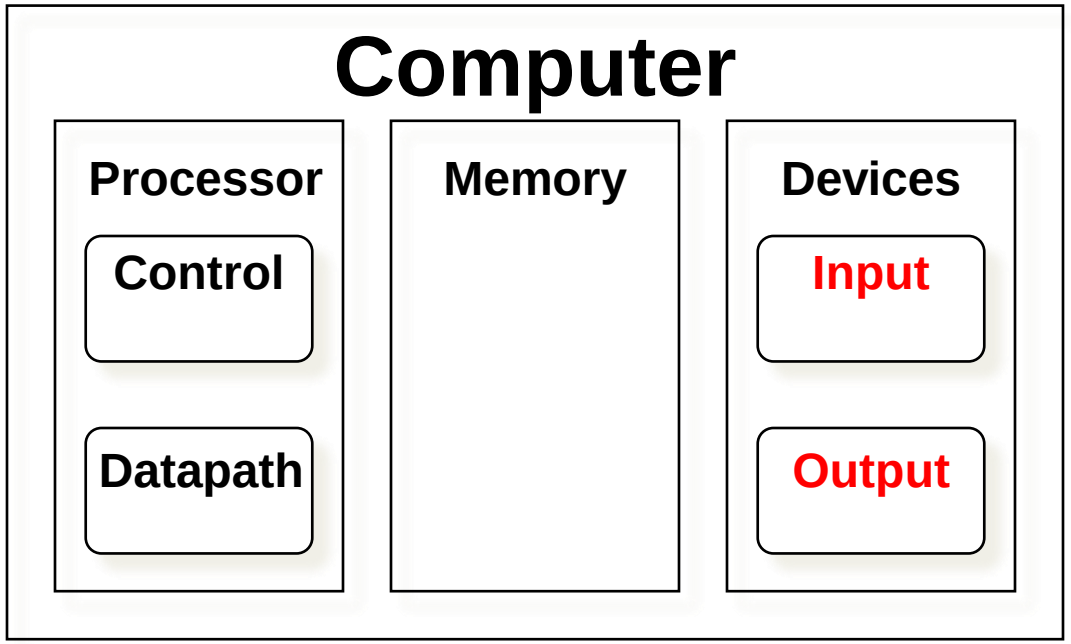


Agenda

- **Input/Output Overview**
- Input/Output Devices
  - Disks
  - SSDs
- Communicating with Memory-Mapped I/O
  - Polling
  - Interrupts
  - DMA
- Bonus I/O Device: GPIO!

Five Components of a Computer

- Components a computer needs to work
  - Control
  - Datapath
  - Memory
  - Input
  - Output

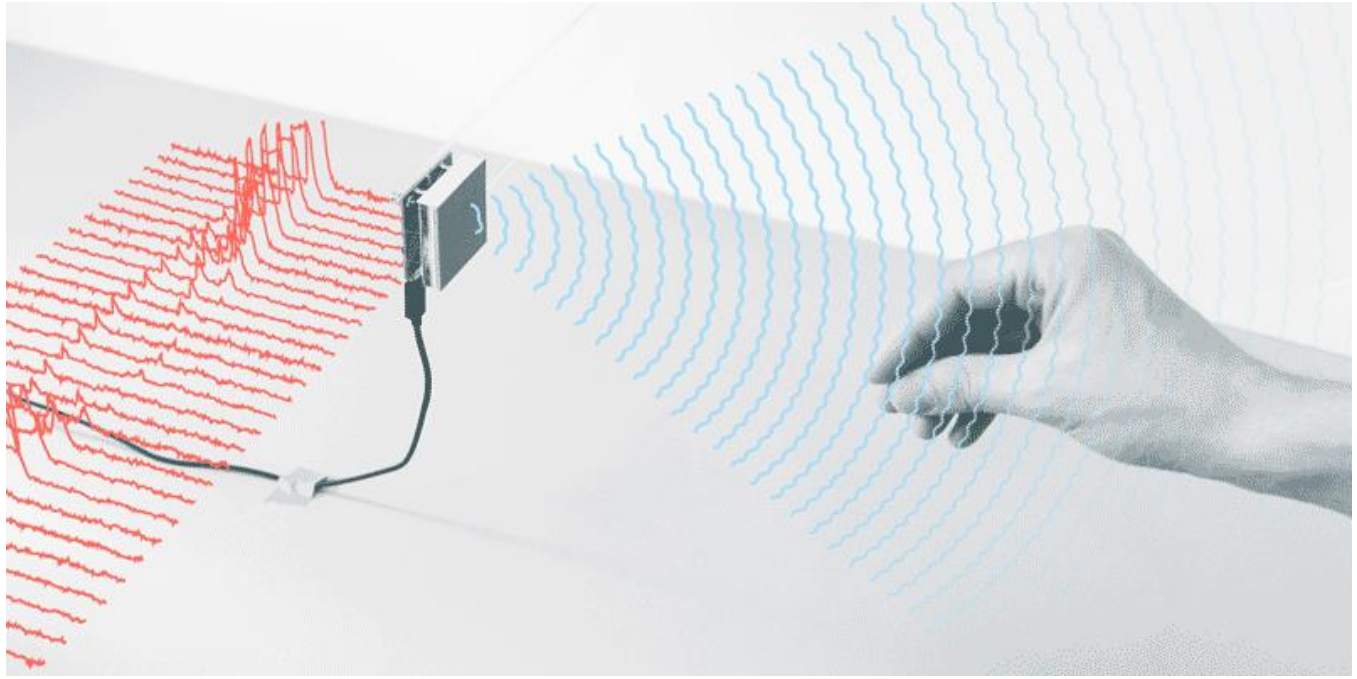


Inputs have many forms

- Keyboard and mouse (USB)
- Touch screen
- Audio input
- Camera
- Motion
- Fingerprint sensor

Inputs have many forms

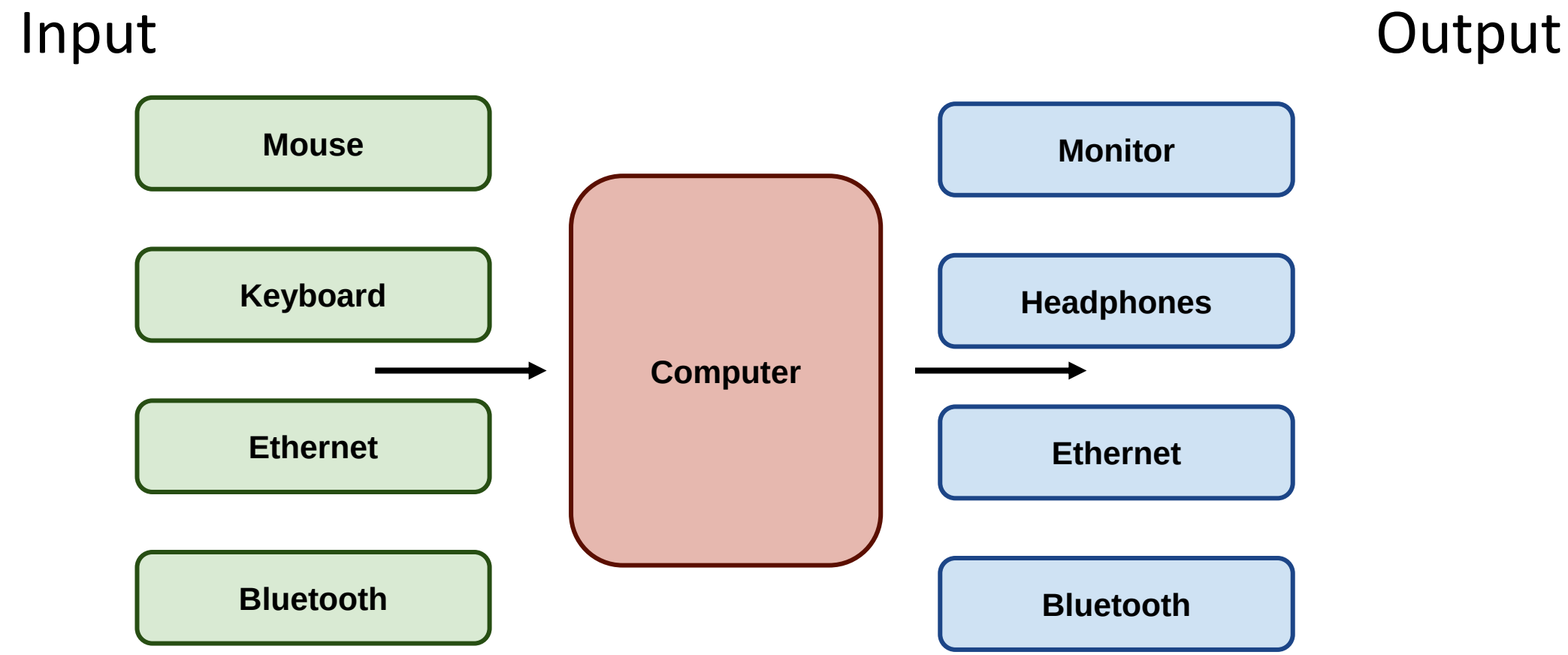
- Tiny Radar (Pixel 4)



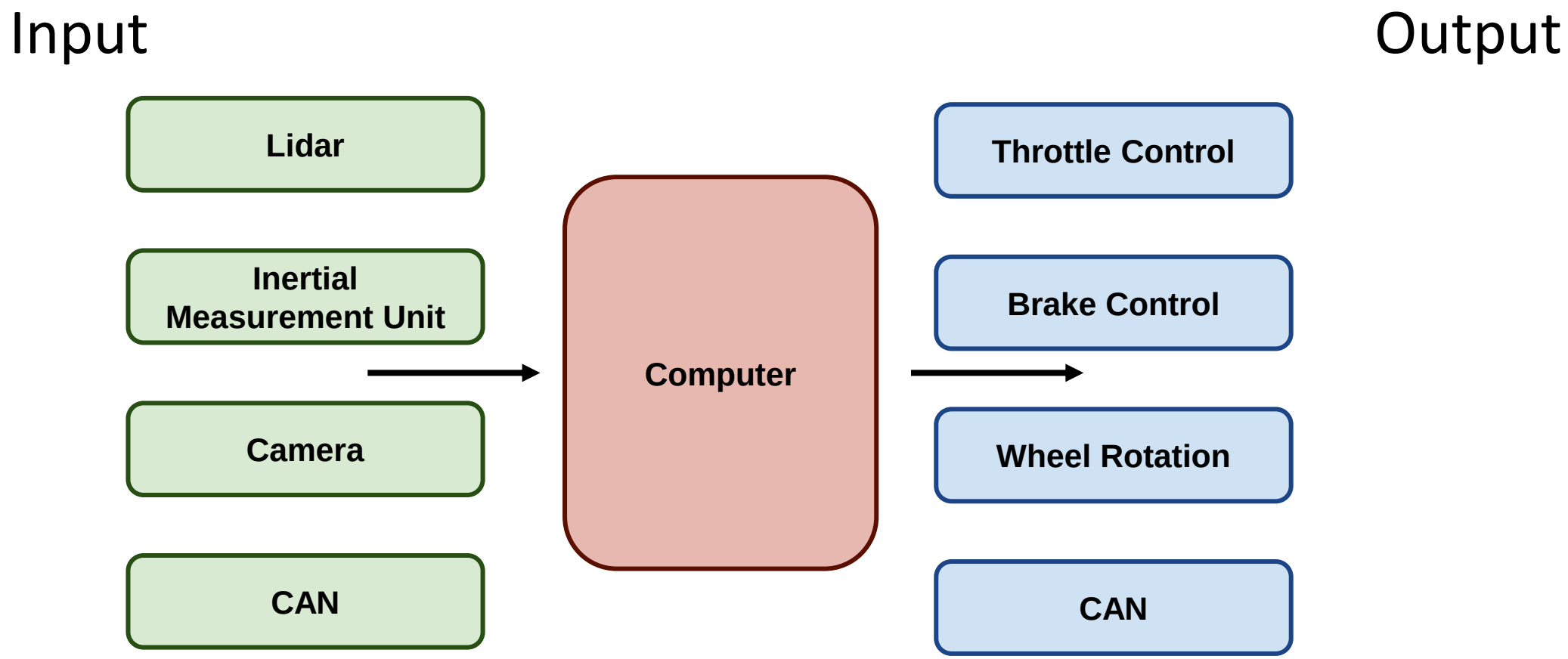
Outputs have many forms too

- Displays
- Audio output
- Printers
- LEDs
- RFID Reader

General-Purpose Computer



Cyber-Physical System



Computers without Input or Output

Imagine a laptop

- Without a screen
- Without a keyboard or touchpad
- Without WiFi or Ethernet
- Without USB ports

What is it good for?

- A computer without I/O is like a car without wheels
  - Interesting technology, but gets you nowhere

I/O Device Examples and Speeds

- I/O speeds: *8 orders of magnitude* between keyboard and graphics

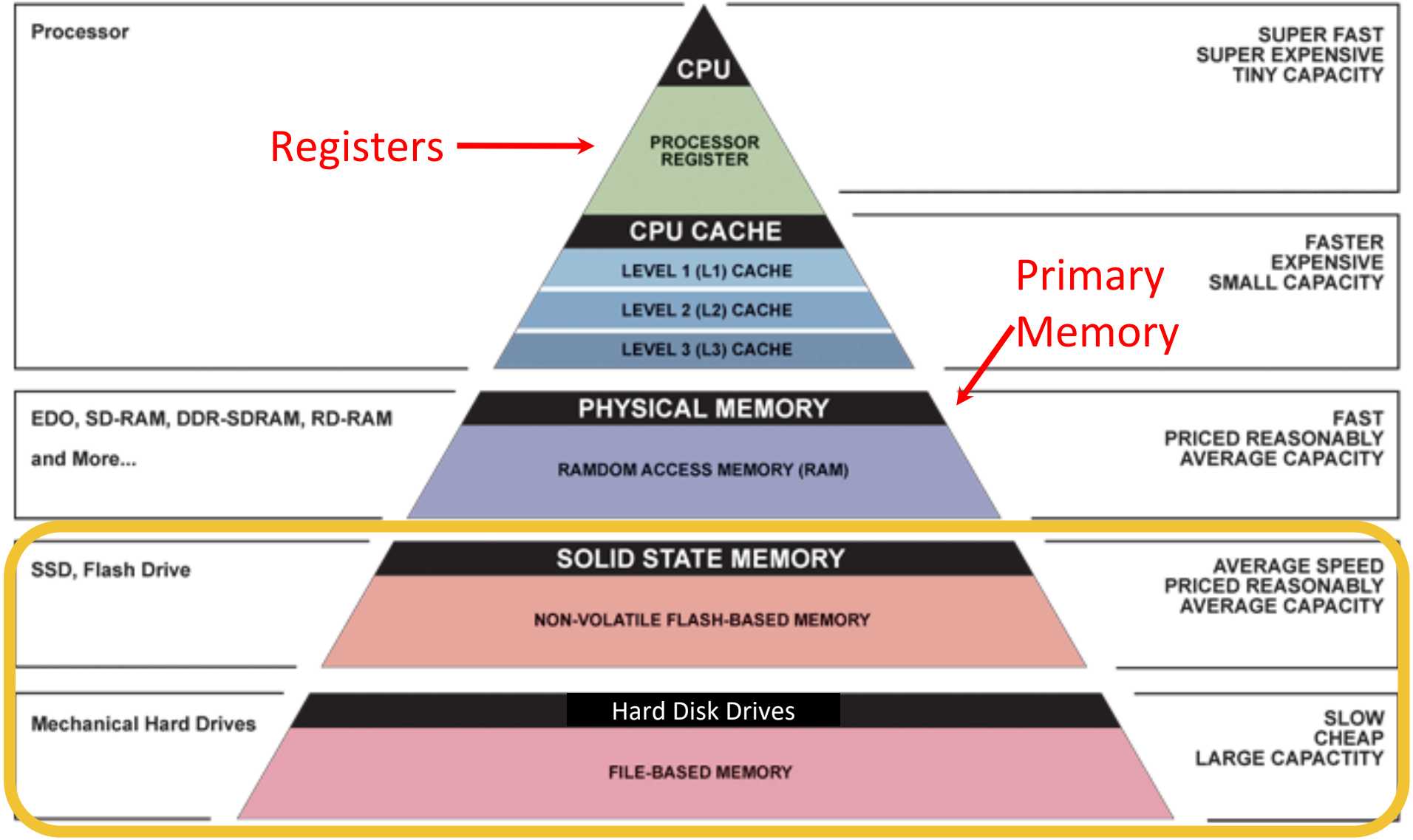
Device	Behavior	Partner	Data Rate (Kb/s)
Keyboard	**Input**	**Human**	**0.2**
Mouse	Input	Human	0.4
Microphone	Input	Human	700.0
Bluetooth	Input or Output	Machine	20,000.0
Hard disk drive	Storage	Machine	100,000.0
Wireless network	Input or Output	Machine	300,000.0
Solid state drive	Storage	Machine	500,000.0
Wired LAN network	Input or Output	Machine	1,000,000.0
Graphics display	**Output**	**Human**	**3,000,000.0**

When discussing transfer rates, use SI prefixes (10^x) & bit/second

Agenda

- Input/Output Overview
- **Input/Output Devices**
  - **Disks**
  - SSDs
- Communicating with Memory-Mapped I/O
  - Polling
  - Interrupts
  - DMA
- Bonus I/O Device: GPIO!

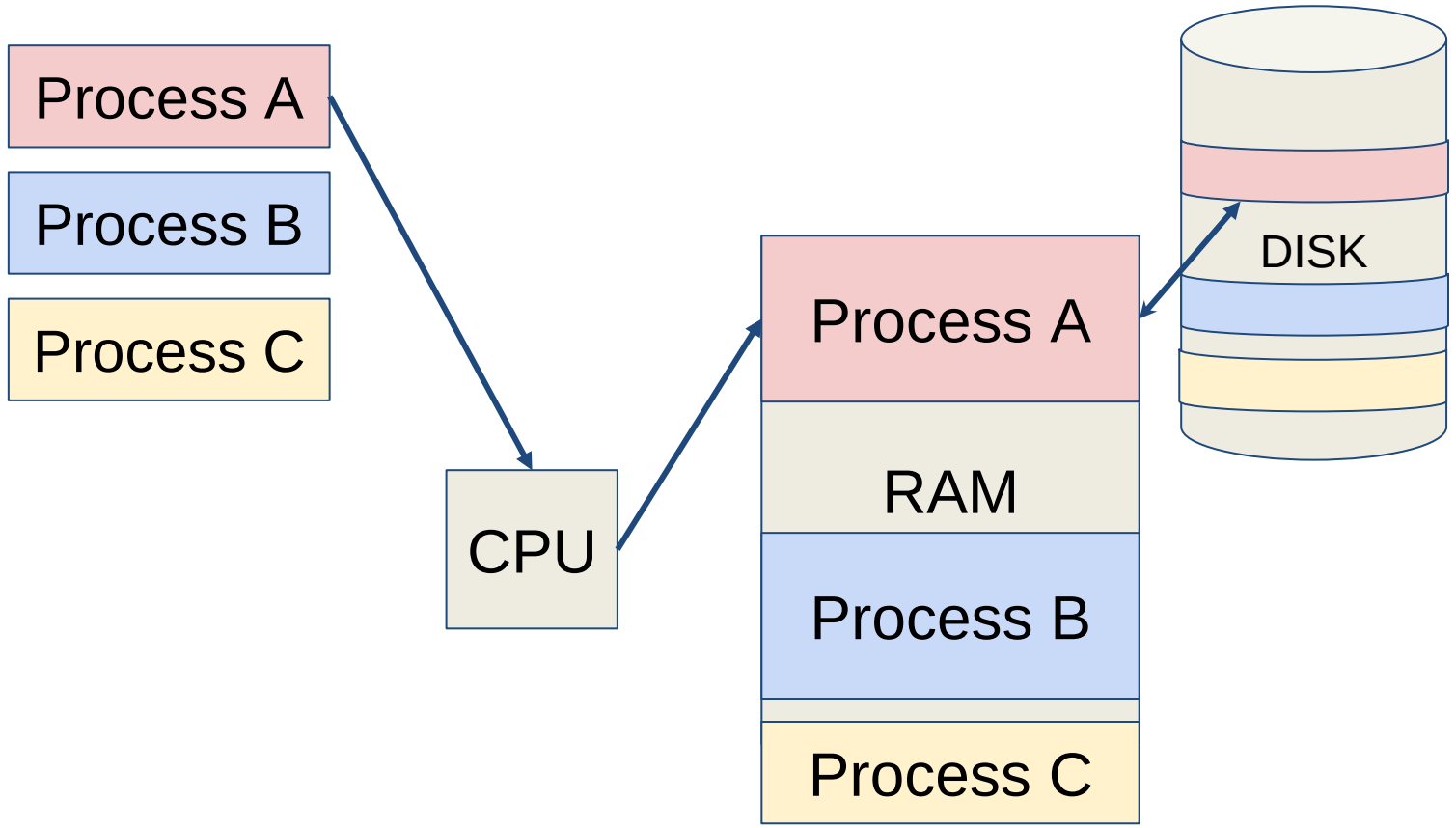
Great Idea #3: Principle of Locality/ Memory Hierarchy



Discussion

- Are disks I/O or Memory?
- What qualities might make them one or the other?

Disks are I/O

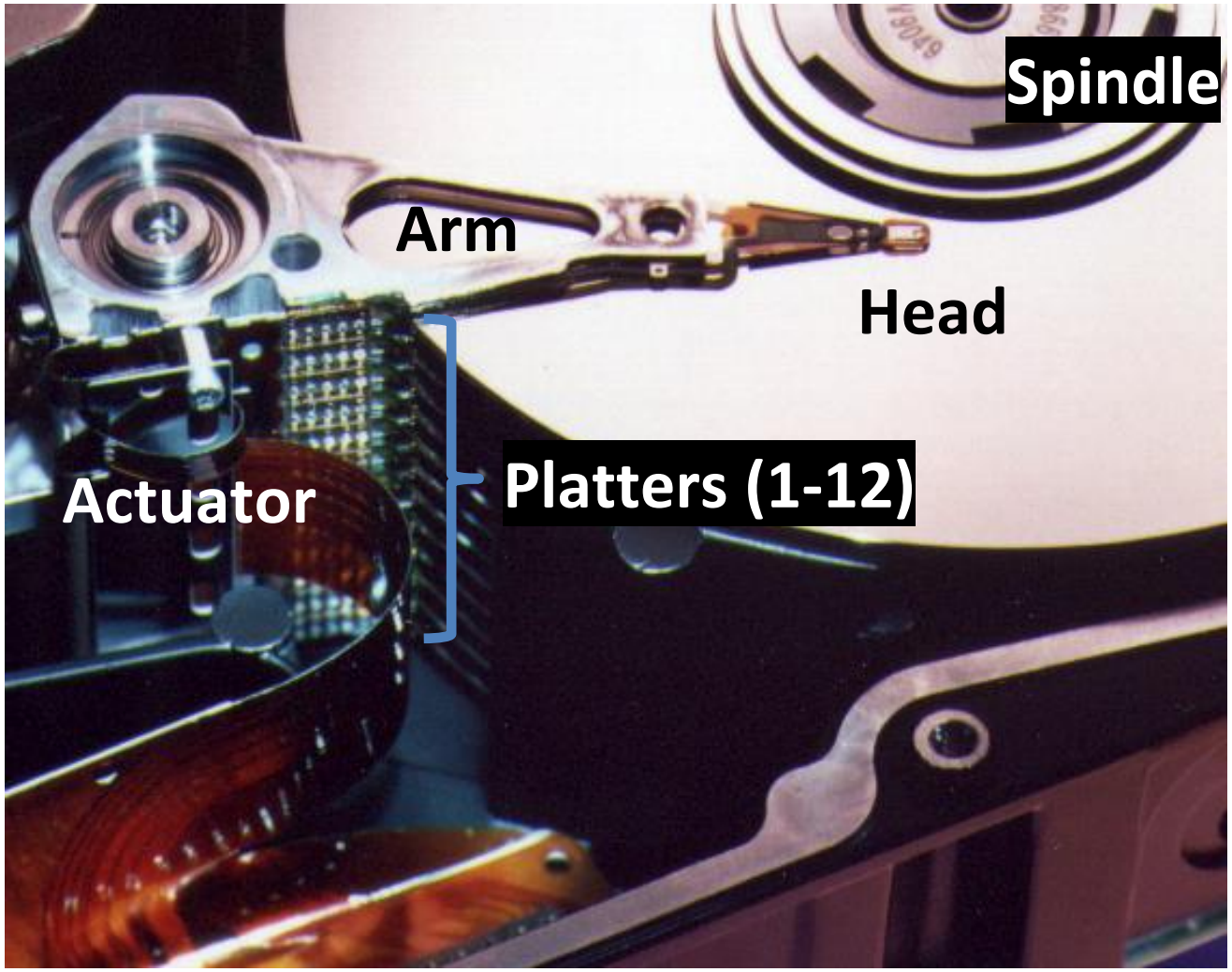


- Disk is never accessed as memory
  - Pages are loaded from disk into RAM, where they become memory
  - The computer can run with the disk removed (if everything is in RAM)

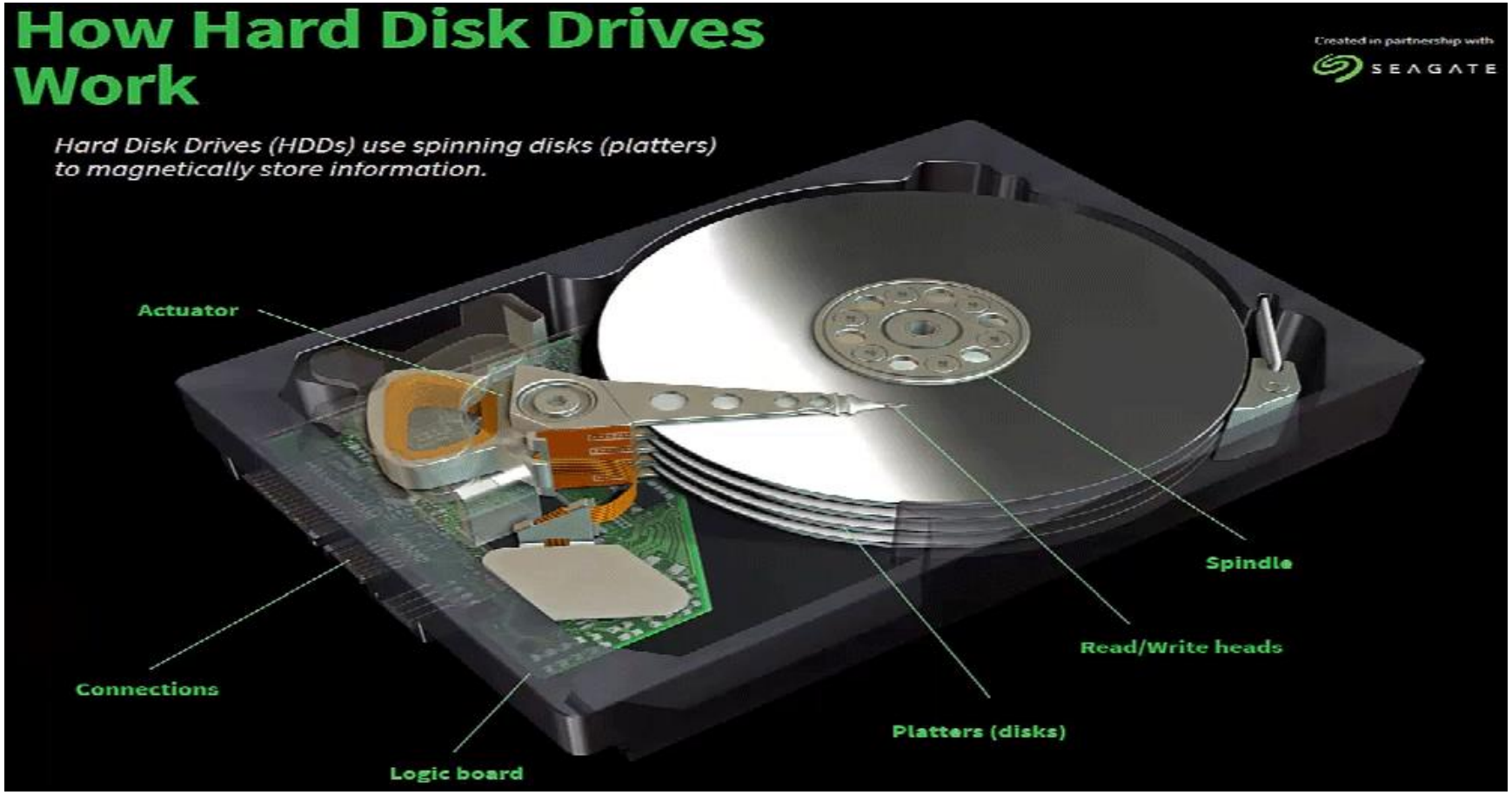
Disks

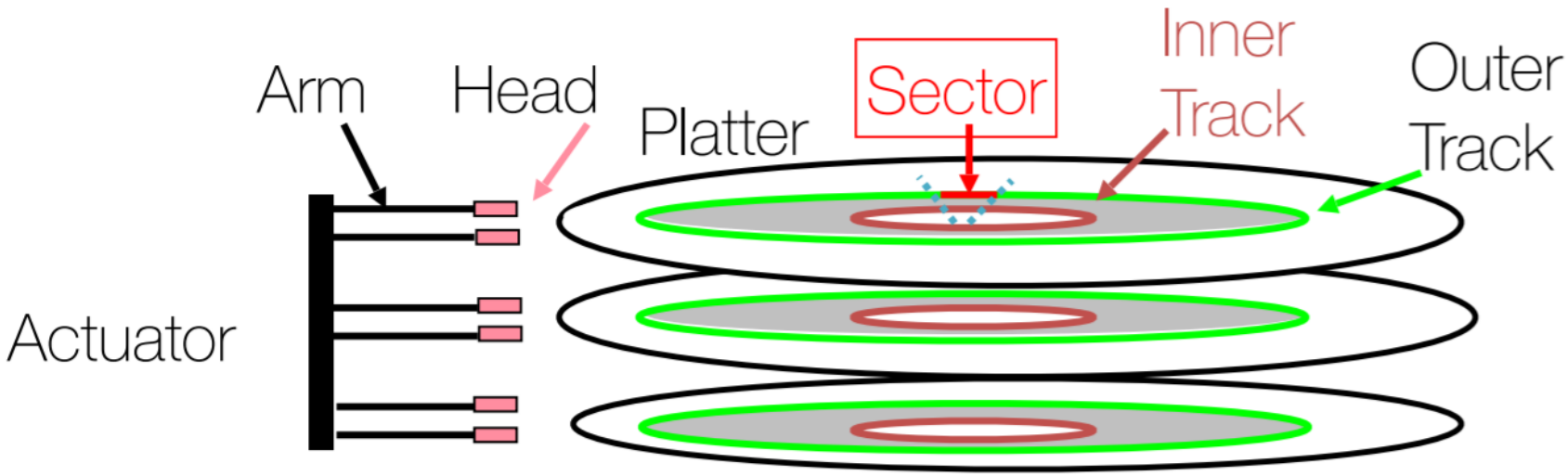
- Nonvolatile storage
  - Retains its value *without power*
  - Long-term, inexpensive storage for files
- Two Types:
  - Magnetic Drives (Mechanical Drive and Floppy Disk)
    - Information stored by magnetizing ferrite material on surface of rotating disk
  - Flash Drives (Solid State Drives and Thumb Drives)
    - Information stored by trapping charge in a semiconductor and MOSFET based dual-gate transistor

Photo of Hard Drive Internals

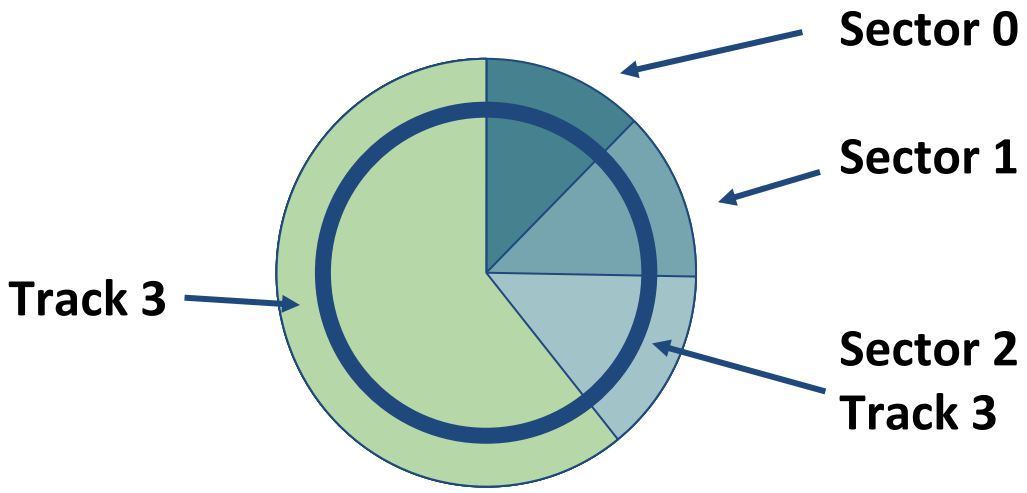


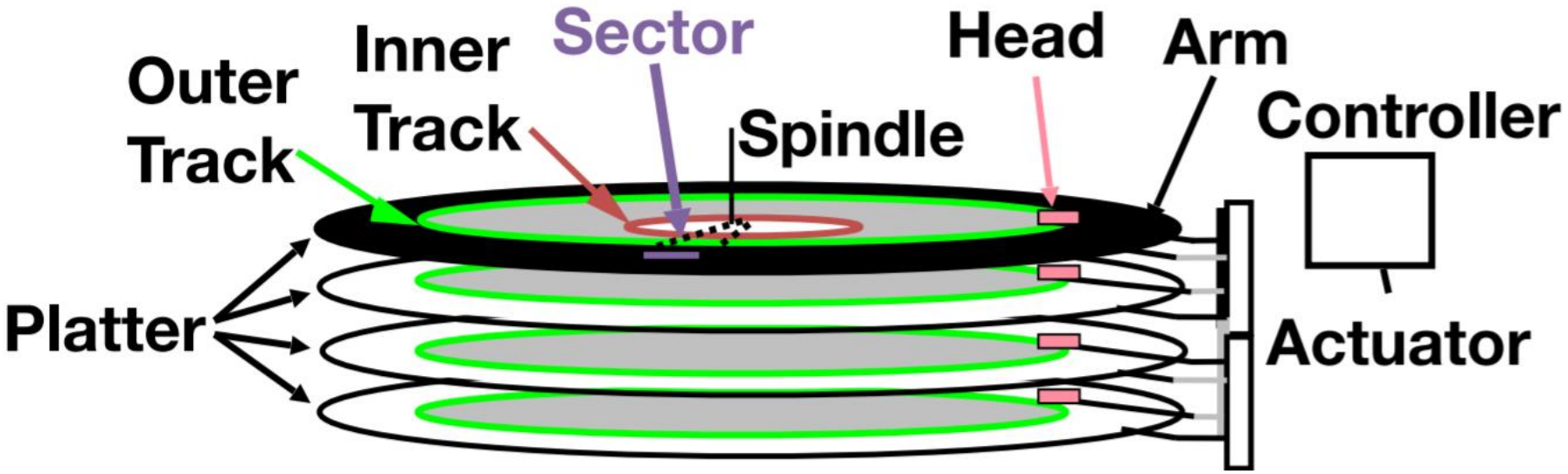
Hard Drive in Action





- Several platters with information recorded magnetically on both surfaces (usually)
  - Bits recorded in *tracks*, which in turn are divided into *sectors* (usually 4 kB)





- Reading and writing from the disk:
  - *Actuator* moves *head* (end of *arm*) over track (*"seek"*), wait for sector to rotate under head, then read or write

Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead

- Seek Time - time to position head to correct track
- Rotation Time - time for disk to rotate to proper sector
- Transfer Time - time for data to rotate under the head

Disk Device Performance

- To find performance, we can estimate some values
- Rotation Time: Average distance of sector from head?
  - **Average rotation time =** 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 ~16,000,000 instructions...
- Seek time: Average number of tracks to move arm?
  - Number of tracks/3 (see CS186 for the math)
  - **Average seek time =** average number of tracks moved × time to move across one track

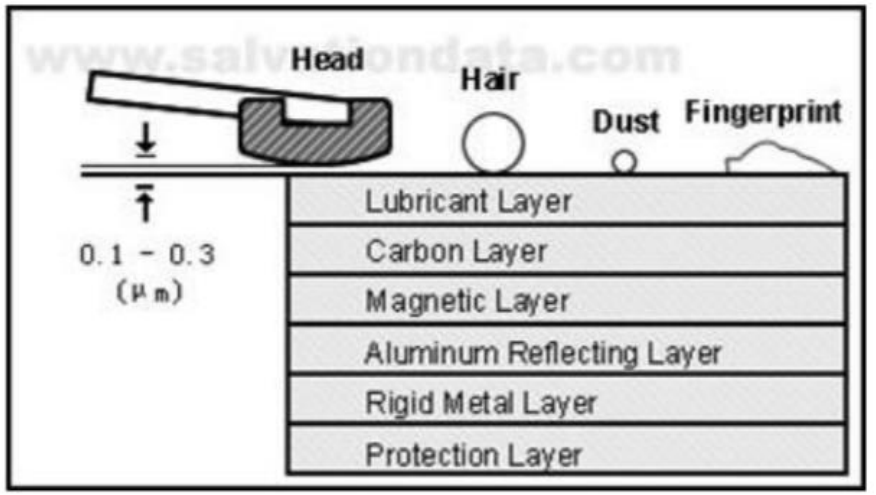
Faster Hard Drives

- Performance estimates are different in practice…
  - Many disks have on-disk caches
    (totally hidden from the rest of the computer)

L1 cache -> L2 cache -> L3 cache -> RAM -> Disk Cache -> Disk

HDD Difficulties

Fast-spinning disk with moving armatures. What could go wrong?



Head Crash

- Literally the head hitting into the disk surface

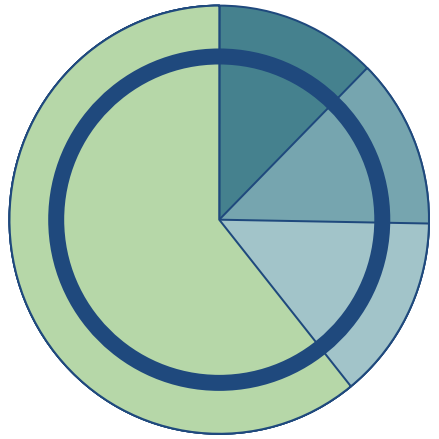
Fragmentation

- To spend less seek and rotation time, we want data to be contiguous whenever possible

- Fragmentation occurs when files are not stored contiguously on the disk

- Can “defrag” to speed up disk!

Question



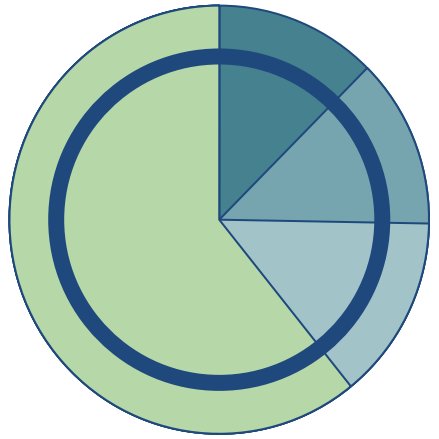
What is the average access time for a disk with the following qualities:

- 900 Tracks with 3 ms for the head to cross all 900
- 10000 RPM
- 1 ms controller processing time
- Copy 1 MB, transfer rate of 1000 MB/s

Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead

A	B	C	D	E
4	6	8	9	11

Question



What is the average access time for a disk with the following qualities:

- 900 Tracks with 3 ms for the head to cross all 900
  -> **1 ms** for ⅓ of tracks
- 10000 RPM -> 166 R/s -> **3 ms** for ½ rotation
- **1 ms** controller processing time
- Copy 1 MB, transfer rate of 1000 MB/s -> **1 ms**

Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead

A	B	C	D	E
4	6	8	9	11

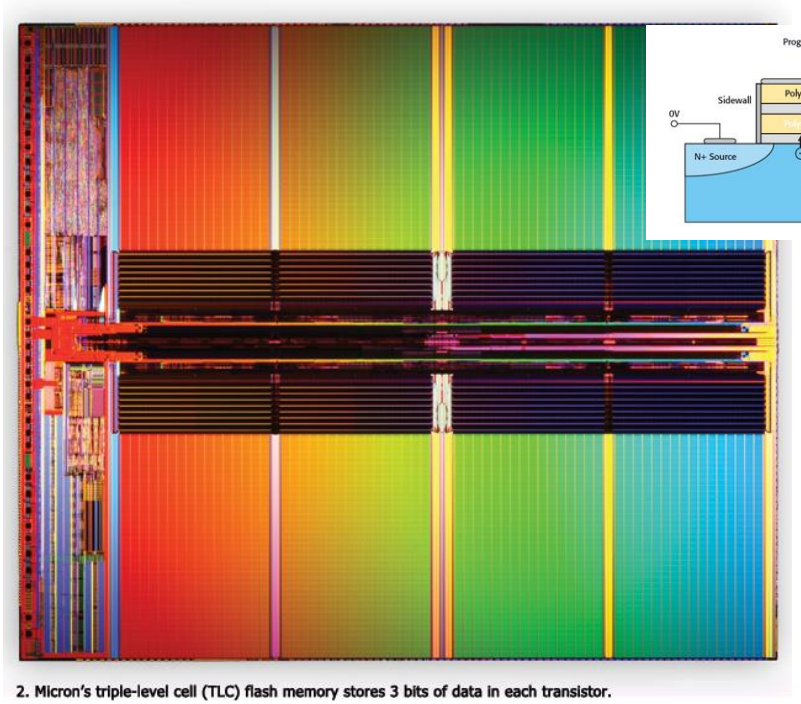
Agenda

- Input/Output Overview
- **Input/Output Devices**
  - Disks
  - **SSDs**
- Communicating with Memory-Mapped I/O
  - Polling
  - Interrupts
  - DMA
- Bonus I/O Device: GPIO!

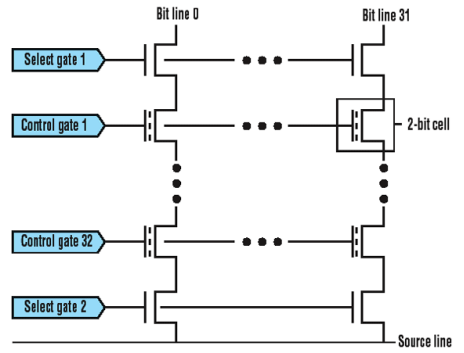
What about Solid State Drives?

- ~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 magnetic media cost very low
  - Flash cost = most cost/bit of flash chips
- Cost/bit of flash came down, became cost competitive for dense 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 capacitor between gate and source/drain which "traps" electrons. The presence/absence is a 1 or 0

Flash Memory Problems

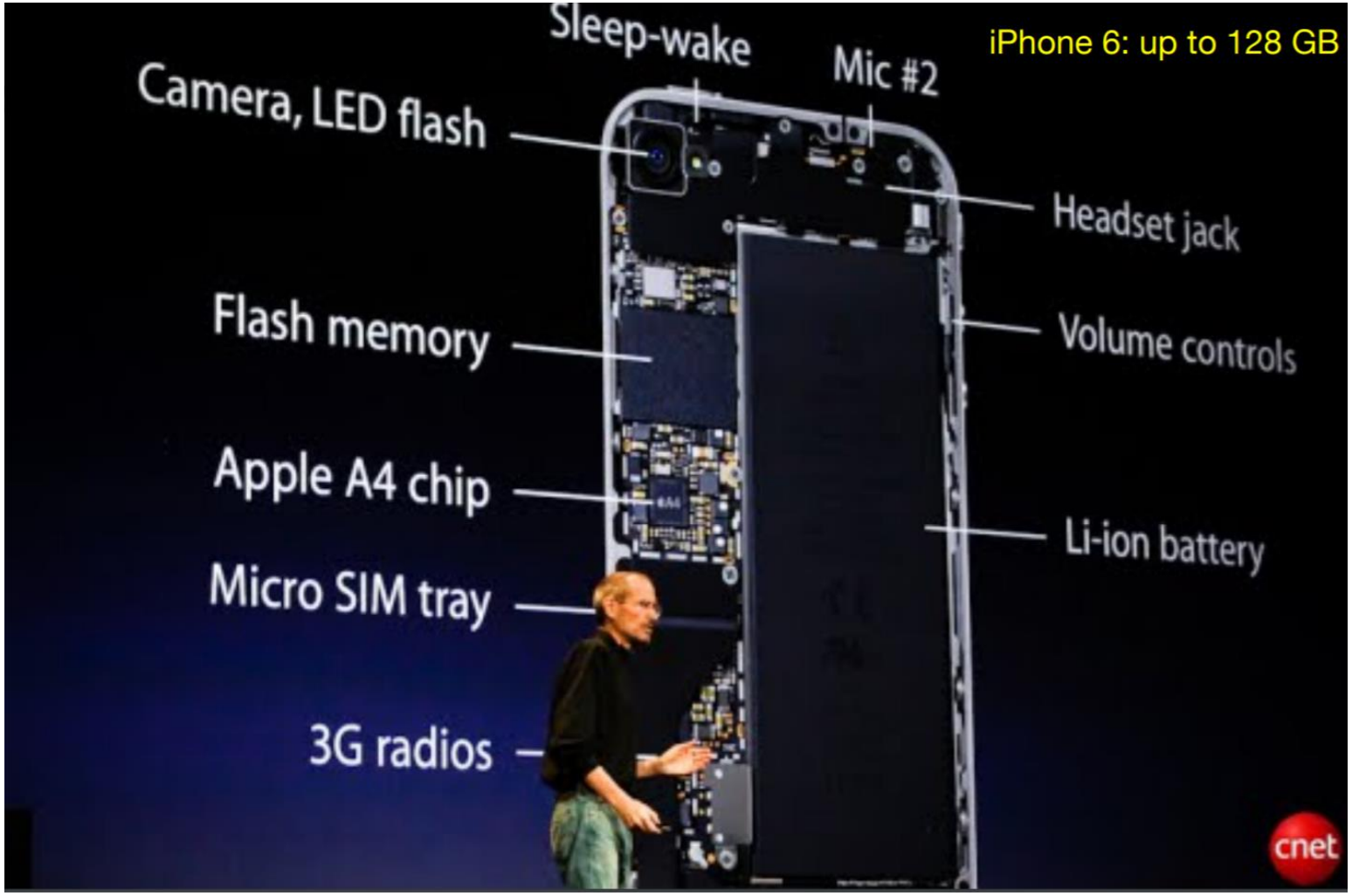
- Flash has a limited number of program-erase cycles
  - 10,000 - 100,000 writes total…

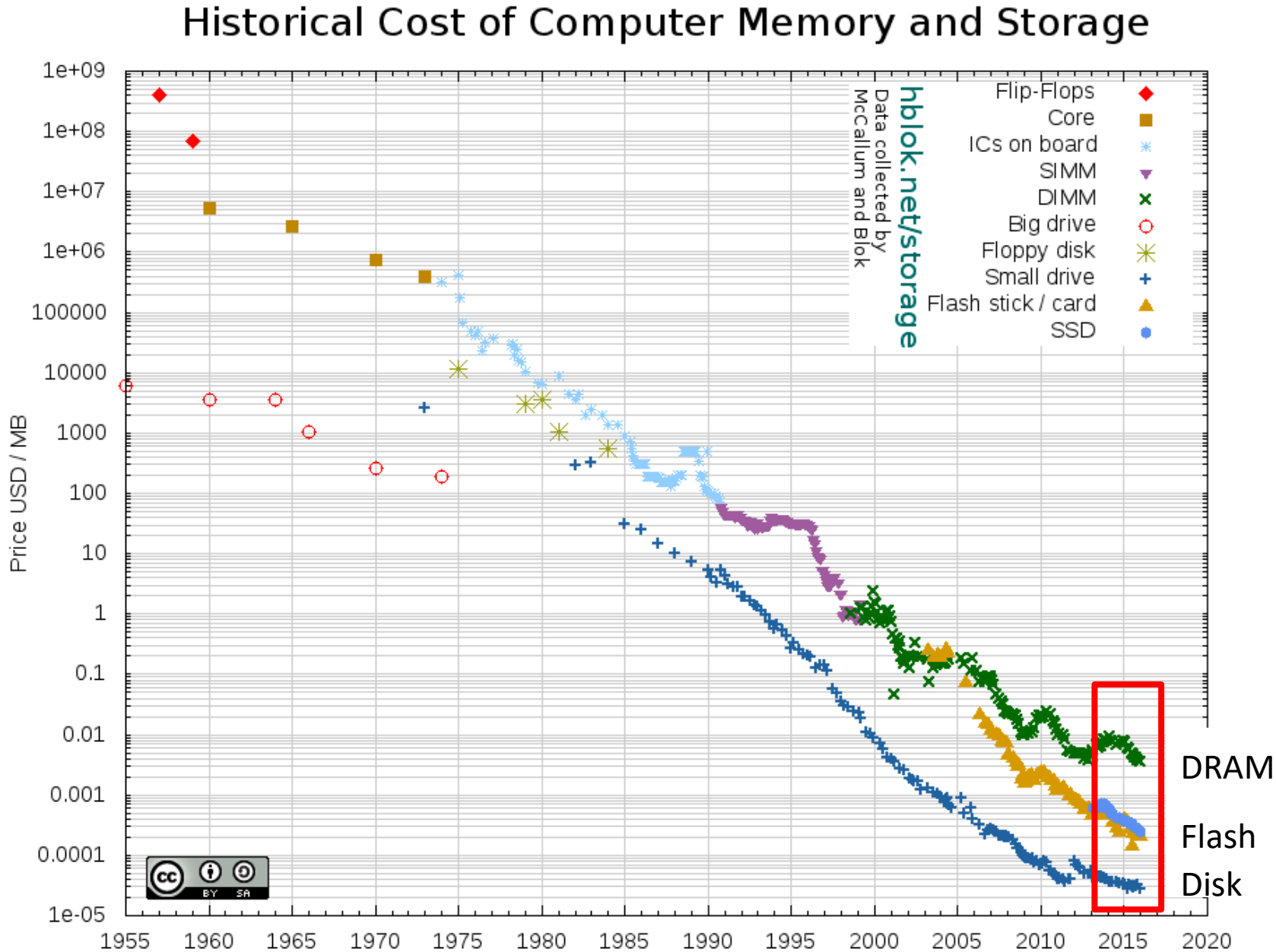
- Solutions:
  - Apply wear leveling to distribute writes evenly
    - Move frequently written data around
    - Create a map from addresses to real locations

  - Lie about how much space you have
    - Sell a 100 GB SSD that secretly has 120 GB
    - Slowly move memory over as sections die

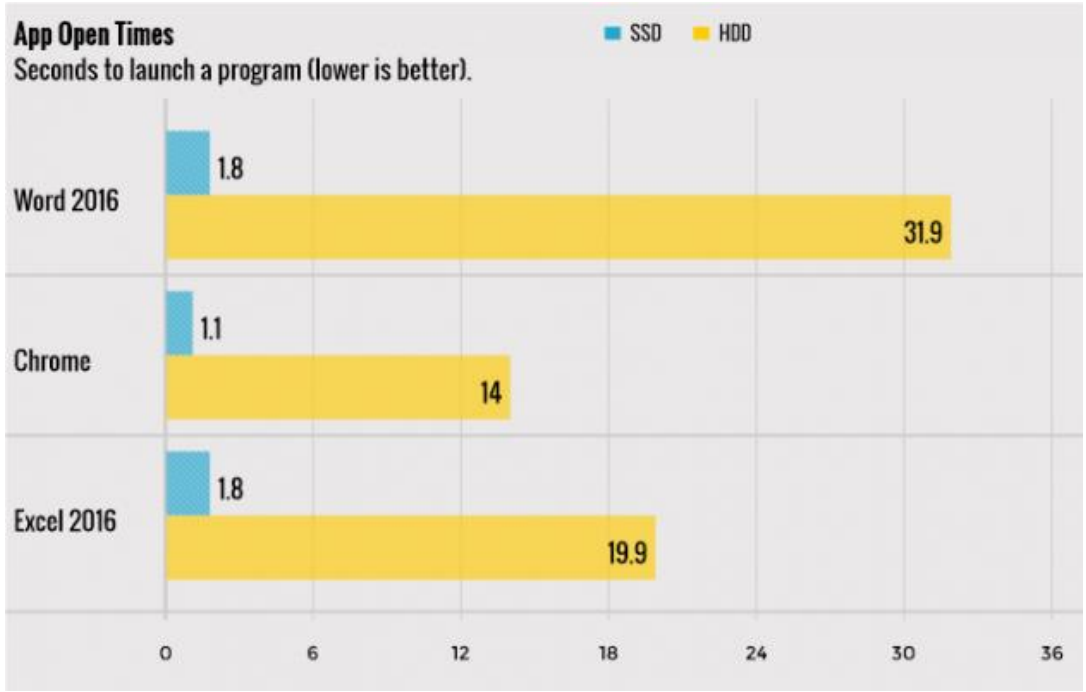
Original iPods

Smartphones Use Flash Memory




Historical Cost of Computer Memory and Storage
Price USD / MB
1e+09
1e+08
1e+07
1e+06
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
1e-05
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
hblok.net/storage
Data collected by McCallum and Blok
Flip-Flops
Core
ICs on board
SIMM
DIMM
Big drive
Floppy disk
Small drive
Flash stick / card
SSD
DRAM
Flash
Disk

SSD's Are Fast and Small



Comparing Drive Types

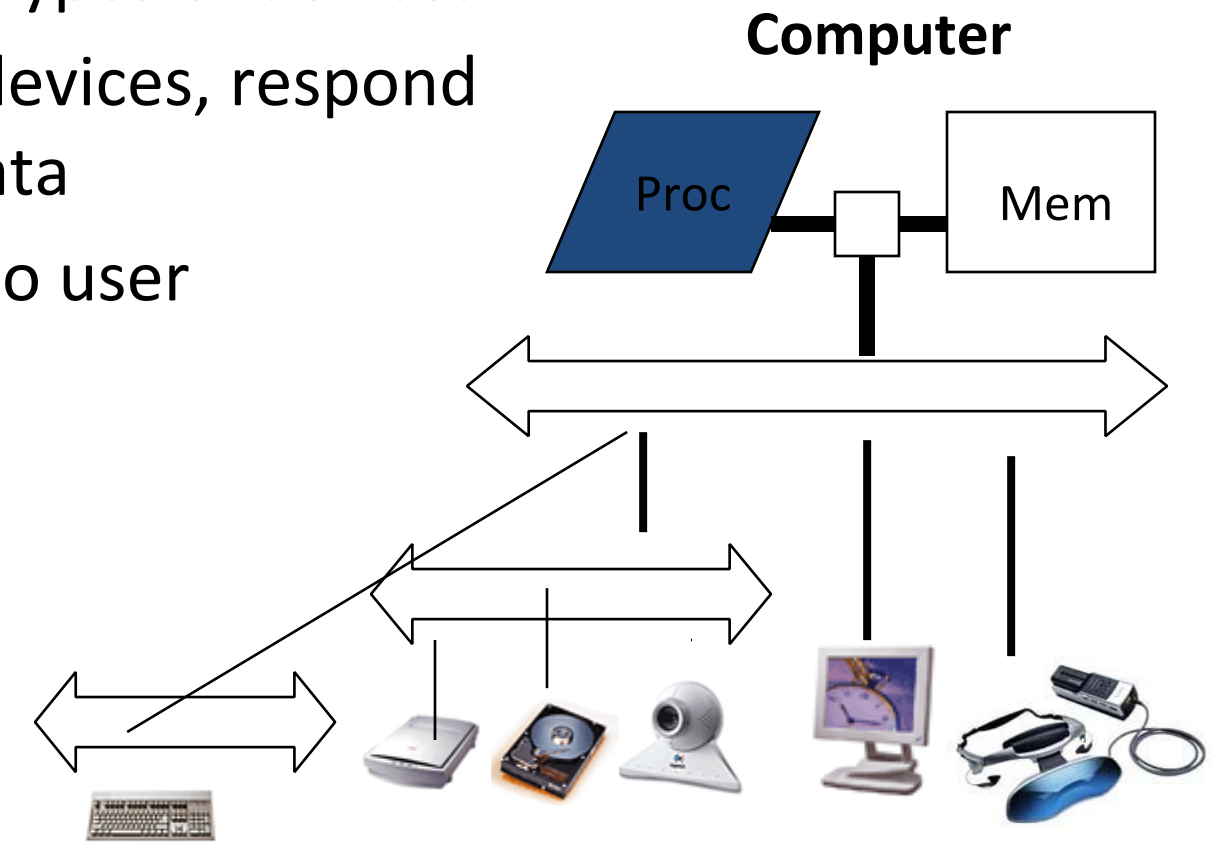
	HDD	Flash-based SSD
Durability and Noise	Loud and susceptible to shock	No moving parts!
Access Time	~ 12 ms ≈ 30M clock cycles	~ 0.1 ms ≈ 250K clock cycles
Relative Power	1	1/3
Cost	~ $0.017 / GB	~ $0.099 / GB
Capacity	500GB - 4TB	128GB - 500GB
Other Problems	Fragmentation	Limited Writes
Lifespan	5-10 years	3-5 years

Agenda

- Input/Output Overview
- Input/Output Devices
  - Disks
  - SSDs
- **Communicating with Memory-Mapped I/O**
  - **Polling**
  - Interrupts
  - DMA
- Bonus I/O Device: GPIO!

What do we need for I/O to work?

1) A way to connect many types of devices
2) A way to control these devices, respond to them, and transfer data
3) A way to present them to user programs so they are useful

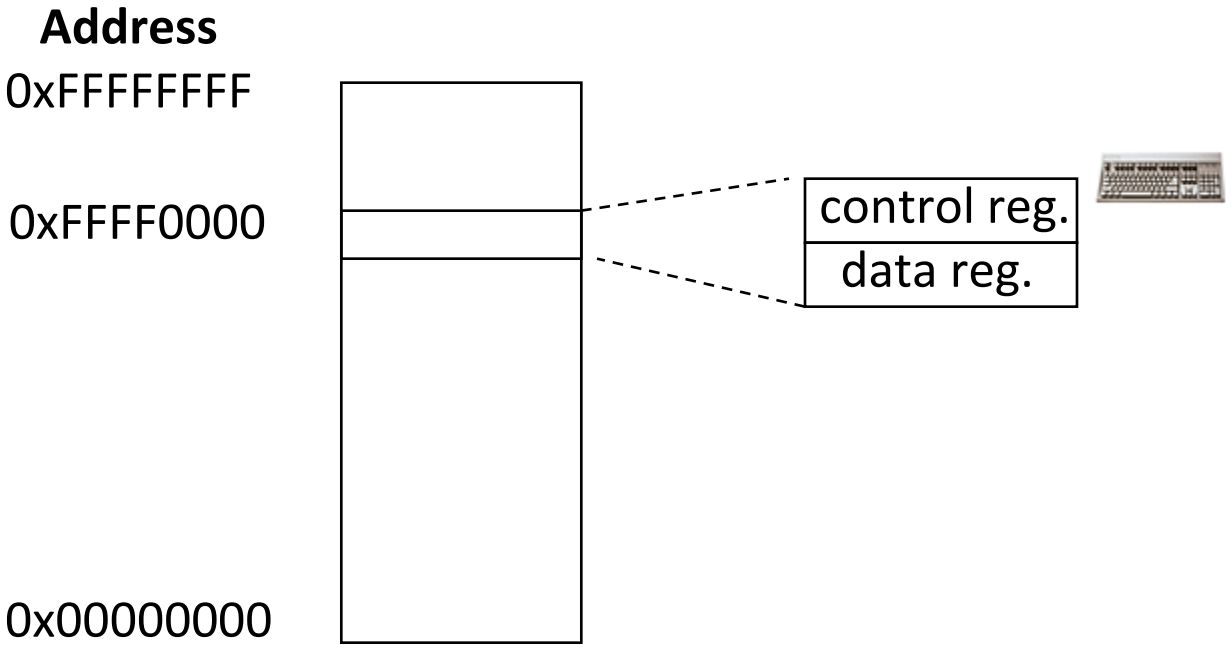


Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Some processors have special input and output instructions (CISC!)
- Alternative model (used by RISC-V and ARM):
  - Use loads for input, stores for output (in small pieces)
  - Called *Memory Mapped Input/Output* (MMIO)
  - A portion of the address space dedicated to communication paths to Input or Output devices
    (no actual memory there)

Memory Mapped I/O

- Certain physical addresses are not regular memory
- Instead, they correspond to registers in I/O devices
- Loads and stores read/write to the device!



Memory Maps

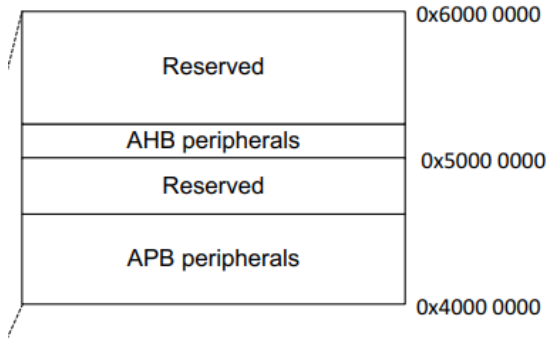
Example from EECS 149

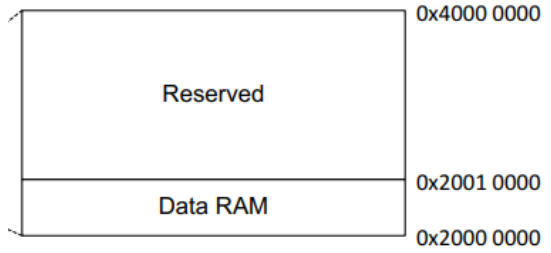
(nRF52832 microcontroller, designed for embedded applications)

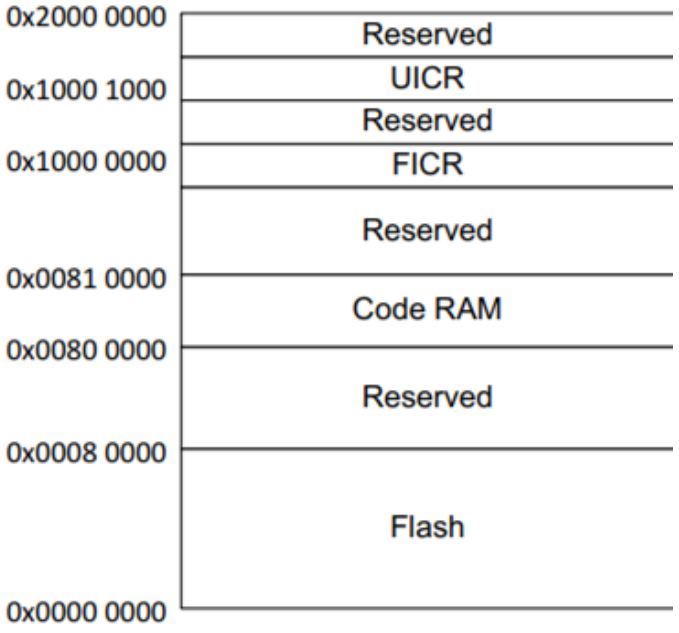
Flash from 0x0 - 0x800000

RAM at 0x20000000

Memory-mapped peripherals start at 0x40000000







Processor-I/O Speed Mismatch

- 1 GHz microprocessor can execute 1 billion load or store instr/sec (32,000,000 Kb/s data rate)
  - **Recall:** I/O devices data rates range from less than 1 Kb/s to more than 1,000,000 Kb/s
- *Input:* Device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- *Output:* Device may not be ready to accept data as fast as processor stores it

Processor Checks Status Before Acting

- Path to a device generally has 2 registers:
  - *Control Register* says it's OK to read/write (I/O ready)
  - *Data Register* contains data

1) Processor reads from control register in a loop, waiting for device to set *Ready bit* (0 → 1)
2) Processor then loads from (input) or writes to (output) data register
   - Resets Ready bit of control register (1 → 0)

- This process is called *"Polling"*
  - Can either loop wait, or if timing is known, poll at frequency

I/O Example (Polling in Assembly)

- **Input:** Read from keyboard into **a0**, MMIO at **t0**

```
            lui   t0, 0xffff # ffff0000
Waitloop:   lw    t1, 0(t0) # control reg
            andi  t1, t1,0x1
            beq   t1, x0, Waitloop
            lw    a0, 4(t0) # data reg
```

- **Output:** Write to display from **a0**

```
            lui   t0, 0xffff # ffff0000
Waitloop:   lw    t1, 8(t0) # control reg
            andi  t1, t1, 0x1
            beq   t1, x0, Waitloop
            sw    a0,12($t0) # data reg
```

Cost of Polling?

- Processor specs: 1 GHz clock, 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)
- Determine % of processor time for polling:
  - **Mouse:** Polled 30 times/sec so as not to miss user movement
  - **Hard disk:** Transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.

% Processor time to poll

- Mouse polling:
  - *Time taken:* 30 [polls/s] × 400 [clocks/poll] = 12K [clocks/s]
  - *% Time:* 1.2×10^4 [clocks/s] / 10^9 [clocks/s] = 0.0012%
  - Polling mouse has little impact on processor
- Disk polling:
  - *Freq:* 16 [MB/s] / 16 [B/poll] = 1M [polls/s]
  - *Time taken:* 1M [polls/s] × 400 [clocks/poll] = 400M [clocks/s]
  - *% Time:* 4×10^8 [clocks/s] / 10^9 [clocks/s] = 40%
  - Unacceptable!
- **Problems:** polling, accessing small chunks

Agenda

- Input/Output Overview
- Input/Output Devices
  - Disks
  - SSDs
- **Communicating with Memory-Mapped I/O**
  - Polling
  - **Interrupts**
  - DMA
- Bonus I/O Device: GPIO!

Alternatives to Polling?

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- **Solution:** Use *exception* mechanism to help trigger I/O, then *interrupt* program when I/O is done with data transfer

Common Exceptions

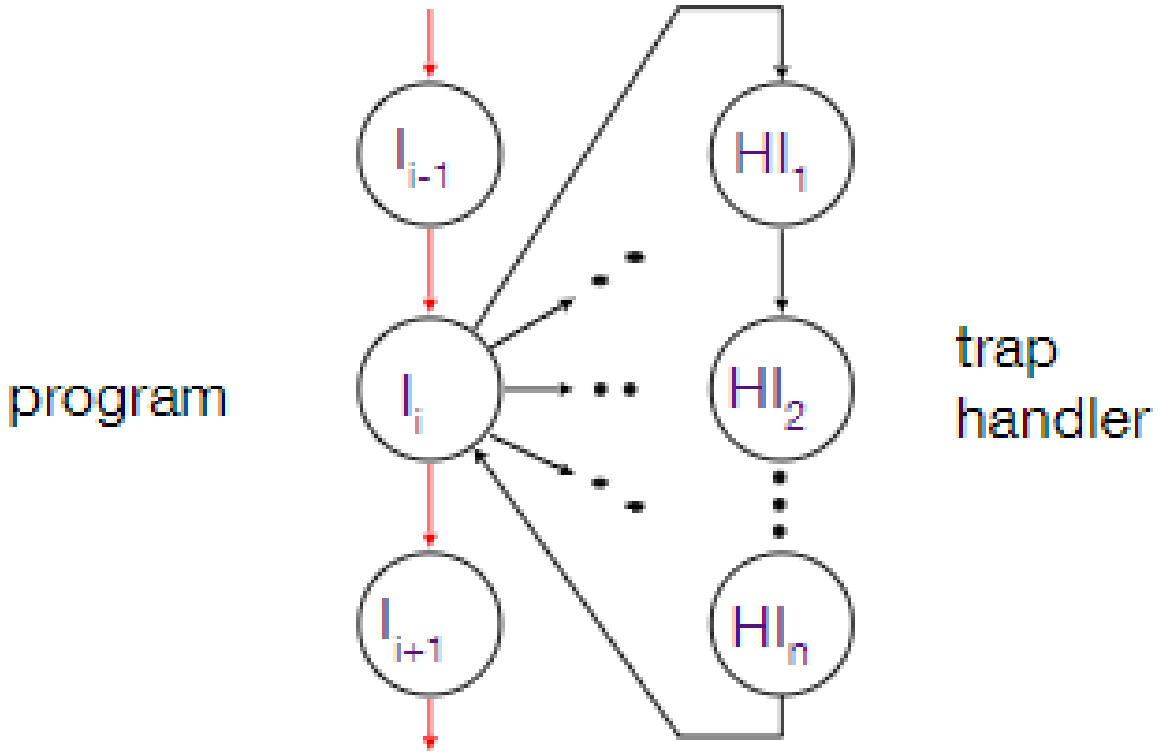
What's the most common exception we're used to as C programmers?

`Segmentation fault (core dumped)`

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently
- *Exception*
  - Arises within the CPU (e.g. syscall, invalid memory access, illegal instruction, overflow error)
  - Synchronous so must be handled *precisely* on instruction that raised exception
- *Interrupt*
  - From an external I/O controller
  - Asynchronous to current program
  - Can handle whenever it is convenient, though don't wait too long
- *Trap*
  - Action of servicing interrupt or exception by hardware jump to handler code
- Dealing with these without sacrificing performance is difficult!

Traps/Interrupts/Exceptions



- An external or internal event that needs to be processed – by another program – the OS. The event is often unexpected from the original program's point of view

Precise Traps

- Trap handler's view of machine state is that every instruction prior to the trapped one has completed, but that no instruction after the trap has executed
- The handler should return by restoring user registers and jumping back to interrupted instruction
- Doesn't need to understand any context! (pipeline, what program was doing, etc)

Exception Steps

1. Unexpected event occurs
2. Save the PC and registers
3. Change PC to trap handler
4. Execute handler code
5. Restore registers and PC

Problems: what about our cache…?

How to Save Program "State"?

- Utilize special "hardware" registers called Control and Status Registers (CSR)
- Put PC in "sepc" CSR, reason in "scause" CSR for trap handler to use
- "sscratch" CSR is points to where we should save registers and PC
- Address of trap handler given by "stvec" CSR
- Special instruction to atomically use CSRs
  - csrrw rd, rs, csr
  - Read old value of csr and put into rd. If rs != 0, put new val in csr

Exceptions in a Pipeline

- Another kind of pipeline hazard
- Consider overflow on `fadd.s` in EX stage

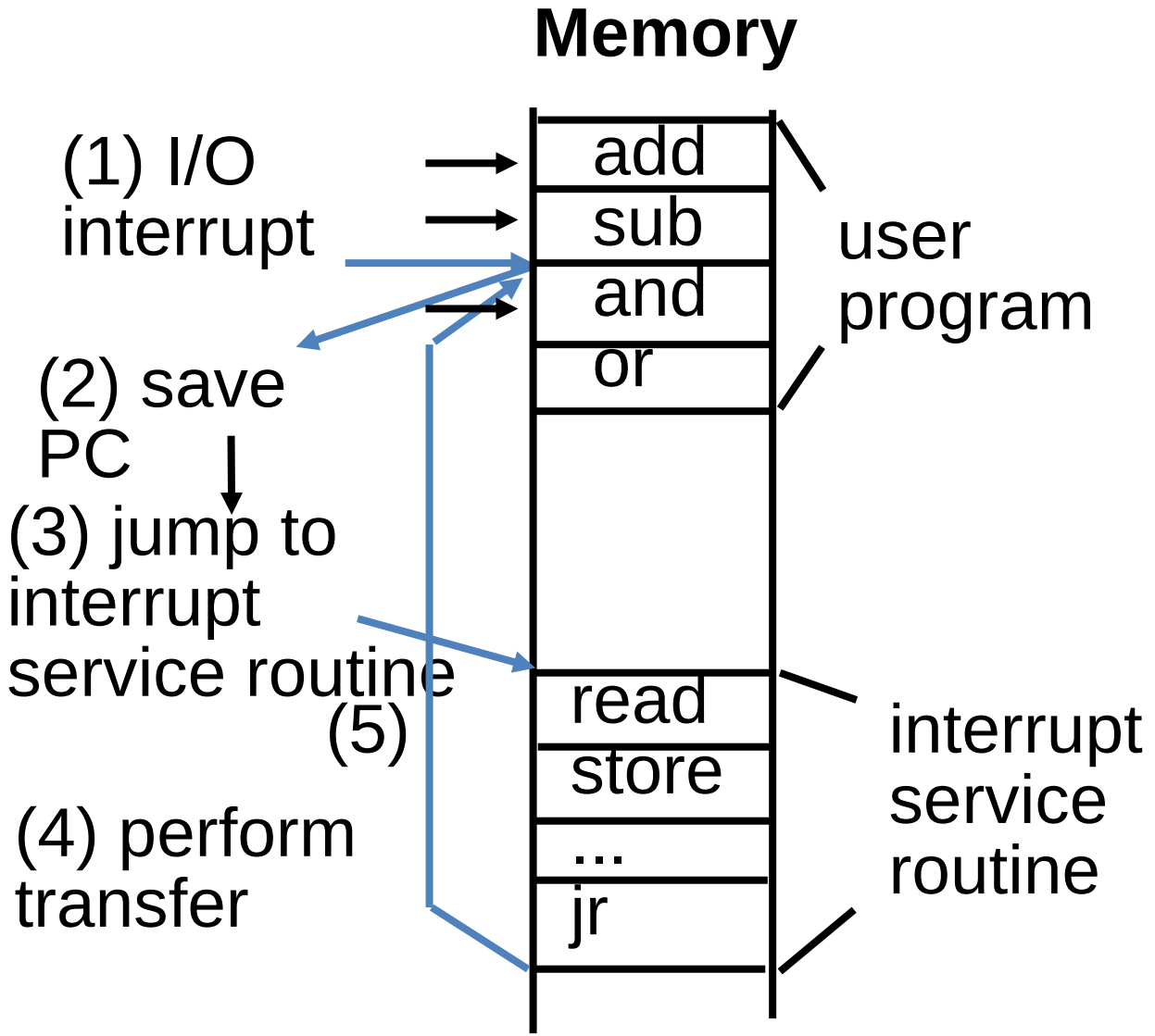
```
fadd.s x1, x2, x1
```

  1) Prevent `x1` from being clobbered
  2) Complete previous instructions
  3) Flush `fadd` and subsequent instructions
  4) Save PC and registers
  5) Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

I/O Interrupt

- An I/O interrupt is like an exception except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- “Asynchronous” with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - *I/O interrupt does not prevent any instruction from running to completion*

Interrupt-Driven Data Transfer



Context Switching

- The trap infrastructure generalize to switching between processes as well
- OS interrupts process at regular intervals, saves state of process, then switches to another
- Involves a little more work than handling an I/O interrupt, though base idea is the same
- More in CS 162

Interrupt-Driven I/O Example (1/2)

- Assume the following system properties:
  - 500 clock cycle overhead for each transfer, including interrupt
  - Disk throughput of 16 MB/s
  - Disk interrupts after transferring 16 B
  - Processor running at 1 GHz
- If disk is active 5% of program, what % of processor is consumed by the disk?
  - 5% × 16 [MB/s] / 16 [B/inter] = 50,000 [inter/s]
  - 50,000 [inter/s] × 500 [clocks/inter] = 2.5×10^7 [clocks/s]
  - 2.5×10^7 [clocks/s] / 10^9 [clock/s] = **2.5% busy**

Interrupt-Driven I/O Example (2/2)

- 2.5% busy (interrupts) much better than 40% (polling) but still not good enough…
- Especially if I have multiple devices

Agenda

- Input/Output Overview
- Input/Output Devices
  - Disks
  - SSDs
- **Communicating with Memory-Mapped I/O**
  - Polling
  - Interrupts
  - **DMA**
- Bonus I/O Device: GPIO!

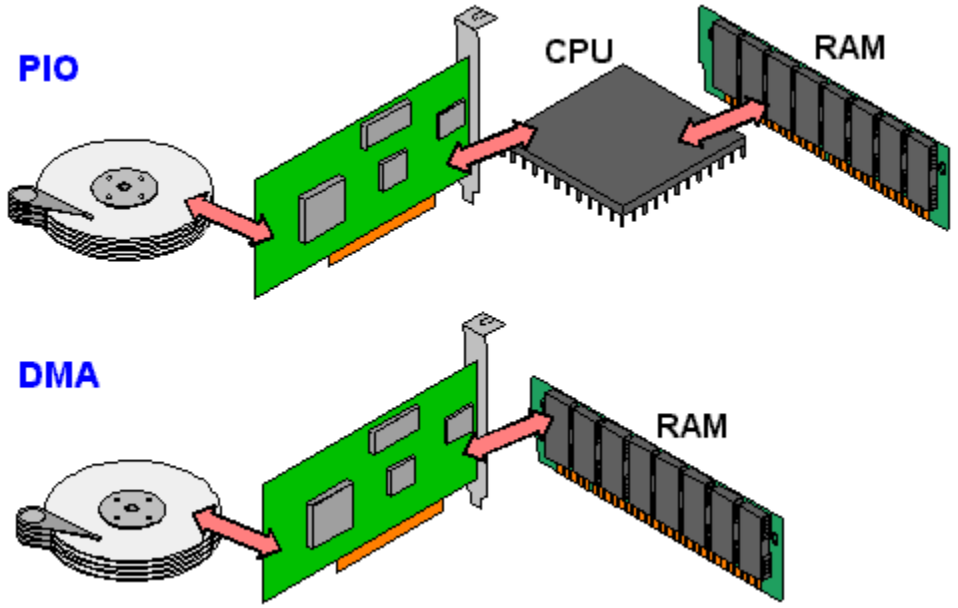
Polling vs. Interrupts

- Low data rate (e.g. mouse, keyboard)
  - Use interrupts to avoid “waiting” for data
  - Overhead of interrupts ends up being low
- High data rate (e.g. network, disk)
  - Start with interrupts
  - Once you start getting data…
    - Then switch to polling! Keep grabbing data until it stops
    - Processor still doing a lot of busywork for data transfer, not ideal…

Interrupt-Driven I/O Example (2/2)

- **Real Solution:** *Direct Memory Access (DMA)* mechanism
  - Device controller transfers data directly to/from memory without involving the processor
  - Only interrupts once per page (large!) once transfer is done

Programmed I/O (PIO) vs. DMA



DMA Analogy (1/2)

- Let's say you are busy doing work but are getting hungry
- You can go to the restaurant, order there, and keep asking them "Is it ready yet?"
  - Polling
- You can order online, wait for them to tell you your order is ready, go pick it up, and return
  - Interrupts

DMA Analogy (2/2)

- New idea: you can just get a delivery service to handle it all for you!
- All you have to do is pick it up when the delivery driver interrupts you to say your food is here!

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

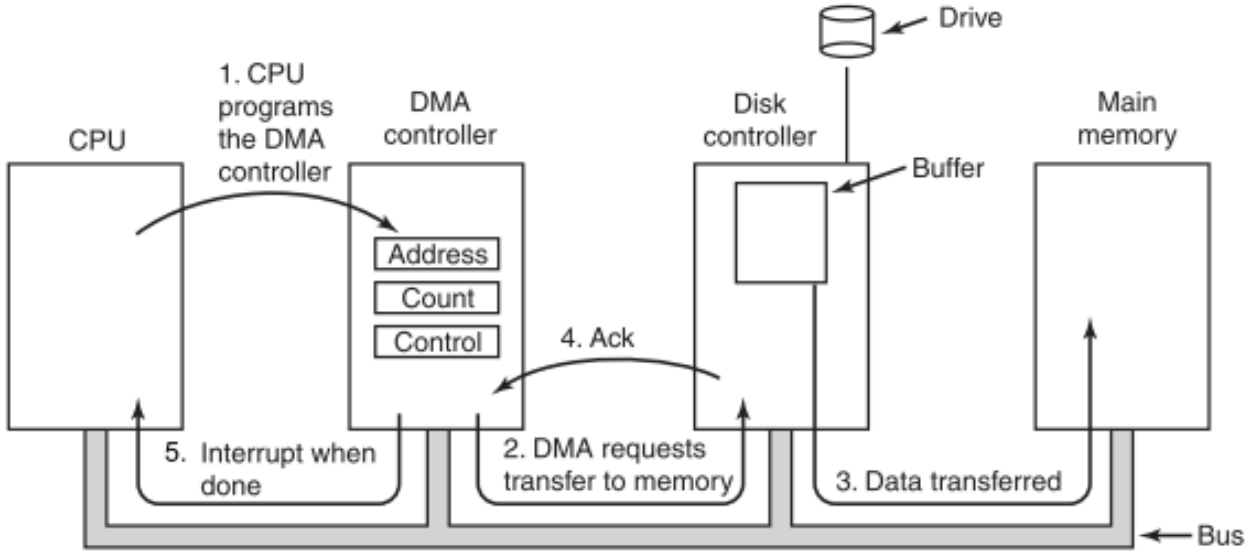


Figure 5-4. Operation of a DMA transfer.

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

DMA: Incoming Data

1. Receive interrupt from device
2. CPU takes interrupt, begins transfer
   - Instructs DMA engine/device to place data @ certain address
3. Device/DMA engine handle the transfer
   - CPU is free to execute other things
4. Upon completion, Device/DMA engine interrupt the CPU again

DMA: Outgoing Data

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

DMA Percent Overhead

- Assume the following system properties:
  - 500 clock cycle overhead for each transfer, including interrupt
  - Disk throughput of 16 MB/s
  - DMA interrupts after transferring 4 KB
  - Processor running at 1 GHz
- What percent of the CPU is occupied now?
  - *Freq:* 16 [MB/s] / 4 [KB/interrupt] = 4000 [interrupt /s]
  - *Time taken:* 4000 interrupt × 500 [clocks] = 2M [clocks/s]
  - *% Time:* 2×10^6 [clocks/s] / 10^9 [clocks/s] = 0.2%

DMA Problems

- If 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 and CPU Sharing Memory?

- How do we arbitrate between the CPU and DMA Engine accessing memory?
  - 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 accesses between the two
  - Transparent Mode
    - DMA transfer only occurs when CPU is not using the system bus

Agenda

- Input/Output Overview
- Input/Output Devices
  - Disks
  - SSDs
- Communicating with Memory-Mapped I/O
  - Polling
  - Interrupts
  - DMA
- **Bonus I/O Device: GPIO!**

Embedded Input/Output

- The simplest way for a computer to interact with the physical world is through digital signals it can read and write

- A lot of embedded devices are simply a combination of buttons and LEDs
  (see TV remote control)

General Purpose I/O (GPIO)

- A set of pins on a computer chip which can be set to high or low (digital inputs and outputs)
  - 1-bit wires in Logisim
- Read them to get a 0 or 1
  - Is the button pressed or not?
- Write them to set to high (1) or low (0)

LED Blinking Code

```
void main(void) {
   const unsigned int LED_NUM  = 1;
   // Sets the LED pin to be an output
   *((volatile unsigned int*)(0x400DC400)) |= (1 << LED_NUM);
   while(1) {
       volatile int i;
       // LED on
       *((volatile unsigned int*)(0x400DC000 + ((1 << LED_NUM) << 2))) = 0x00;
       for (i=0; i<400000; i++);
       // LED off
       *((volatile unsigned int*)((0x400DC000) + ((1 << LED_NUM) << 2))) = 0xFF;
       for (i=0; i<400000; i++);
   }
}
```

Memory Mapped I/O Address!

Compiler: don't optimize me!!!

Delay for a while with "polling"

Agenda

- Input/Output Overview
- Input/Output Devices
  - Disks
  - SSDs
- Communicating with Memory-Mapped I/O
  - Polling
  - Interrupts
  - DMA
- Bonus I/O Device: GPIO!
- **Summary**

Summary

- I/O gives computers their 5 senses + long term memory
  - I/O speed range is 8 orders of magnitude (or more!)
- Input/Output devices are many and varied
  - Hard drive disks
  - Solid state drives
  - GPIO
- Communicate with devices over memory-mapped I/O
  - Polling, Interrupts, or DMA