Data Storage and I/O Scheduling

Don Porter

Portions courtesy Emmett Witchel and Montek Singh

Today's Lecture

- How do computers store and access bits?
- Review current and emerging storage technologies
  - Hard Disk Drives (HDDs)
  - Solid State Drives (SSDs, aka flash)
- Reasoning about volatility vs. persistence
- Key trade-offs
- How to optimize I/O performance
- Practical miscellany
- Emerging media

OS's view of a storage device

- Simple array of sectors
  - Sectors are usually 512 or 4k bytes
  - Also called Logical Block Addresses (LBAs)
    - Captures virtual address space that device exports to OS

- OS can issue reads/writes to disk as small as one sector/LBA
- Depending on how data is placed on device, can also aggregate into larger requests
  - One contiguous LBA range and operation (read/write) per IO request

Storing Bits in a Computer

- We are used to the idea of just defining variables, reading, writing, etc.
- But internally, how does one actually store data?
  - How is data stored in real life, before computers?

Discretizing Physical Phenomena

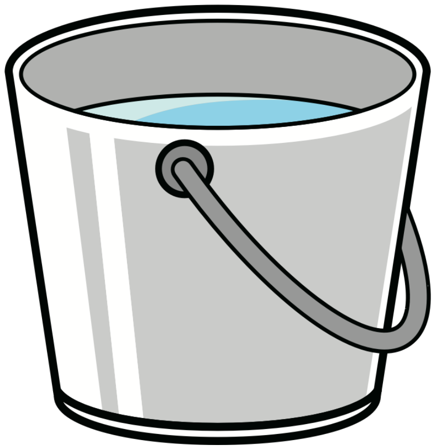
- Key idea: Measure and manipulate some property of a physical medium
- Silly example: I can store a bit in a bucket of water
  - Empty == 0
  - Full == 1
  - Read: measure water with sensor
  - Write: dump or refill with actuator(s)
- Any concerns?
  - What if the bucket has a few drops?
  - What if the bucket has a slow leak?

"Discretize" == round measurement of a continuous quantity (volume) to a discrete value (bit)

https://openclipart.org/detail/325616/bucket

Lessons from the bit bucket

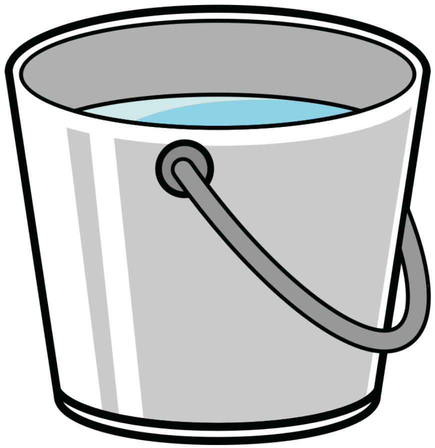
- Rarely perfectly full or empty
  - Rather, need to tolerate some imprecision
  - Better bit bucket encoder:
    - <1/4 full == 0
    - >3/4 full == 1
    - 1/4---3/4 == error



- Damage to the media can flip bits
  - A leaky bucket can shift its value over time
  - Or just evaporation over long enough…

What if I want to store more bits?

- Could use more buckets
  - Need more space (or smaller buckets)
    - Impact of smaller size on precision? Cost?
- Could take finer measurements
    - <1/4 full == 00
    - 1/4---1/2 == 01
    - 1/2---3/4 == 10
    - >3/4 full == 11
  - Impact on cost? Risk of error?



What about write speed?

- What can I do to make writes faster? Downsides?
  - Smaller buckets -> more precise reads
  - More hoses/valves can fill more buckets -> more cost
- Splitting the difference:
  - Expensive filling mechanism attached to a drone that relocates over correct bucket:
    - Save money, increase latency

Key design questions:

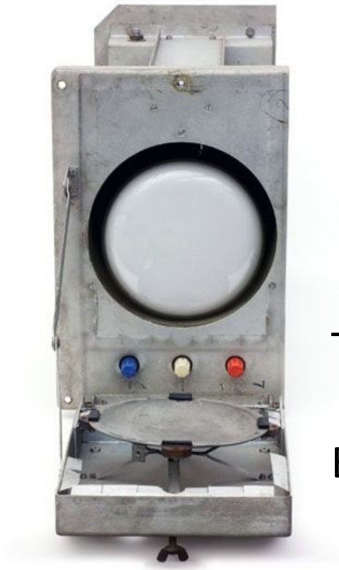
- What is the physical phenomenon?
- How robust is the physical phenomenon to environmental damage, or passage of time?
- How to scale capacity, vs cost?
- Other engineering constraints or performance anomalies?

Key design questions: Bit bucket

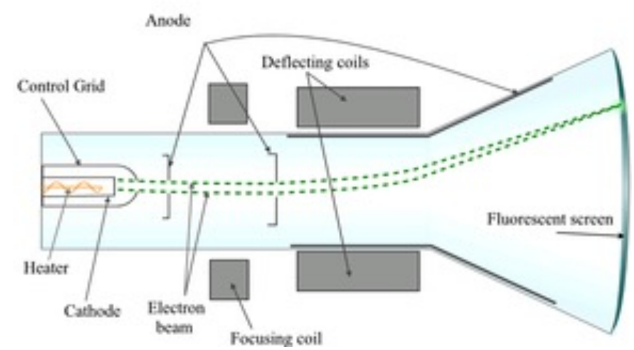
- What is the physical phenomenon?
  - Water volume
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Not terribly robust to physical shock (not good for an apple watch), water evaporates over enough time (prob months)
- How to scale capacity, vs cost?
  - More buckets or finer measurements
- Other engineering constraints or performance anomalies?
  - “Drone solution” introduces delays, but cheaper than per-bucket instrumentation

1947: First Fully Electronic Memory

- Williams-Kilburn Tube
- Media: a Cathode Ray Tube (CRT)
  - Same as old TVs
- CRTs work by shooting electron beam at phosphorescent screen
  - Pixel "glows" for a fraction of second
  - Encode one bit per pixel on/off
- Computer "read" charge level on screen with metal pickup plate
  - Rewrites signal periodically, Called ***refresh***



Top: Computer History Museum
Bottom: Wikipedia



Many media require periodic data refresh

Williams-Kilburn Tube Lessons

- Write speed limited by how fast glow fades
- Precision: Glow has imperfect fading rate
  - Mitigated by writing to nearby, unmeasured space on screen to pull charge out of an “off” pixel
  - Trades capacity for lower latency
- Physical resilience: sensitive to other magnetic fields
  - Required constant recalibration in practice
- Refreshing values uses power, increases cost
  - But again, a common strategy
    - Non-thrifty fix for leaky bucket: periodically measure and top off

Key design questions: Williams-Kilburn

- What is the physical phenomenon?
  - Phosphoresence in a cathode-ray tube
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Sensitive to other magnetic fields
  - Holds data for fraction of second before refresh
- How to scale capacity, vs cost?
  - Bigger screen or more screens; limited by speed of electron beam
- Other engineering constraints or performance anomalies?
  - Can trade some space for higher write speed

1949: Delay-line memory

- Encode data in a looped waveform/signal
  - Media varies: sound through a mercury tube used first
    - High speed of sound in mercury, plus similar acoustic impedance to quartz crystals
    - Audible "hum" from some devices

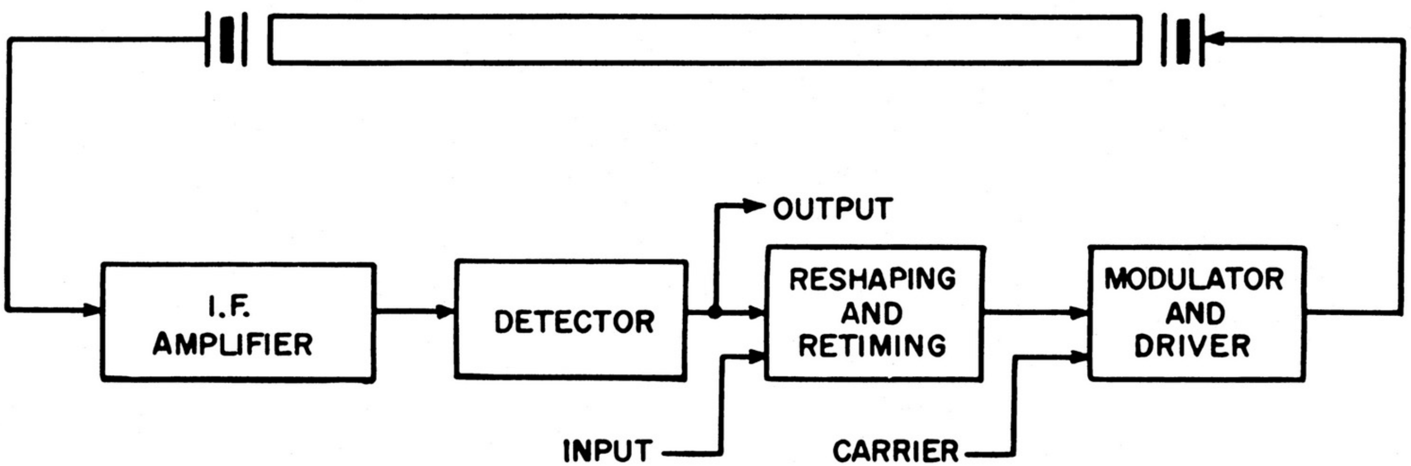
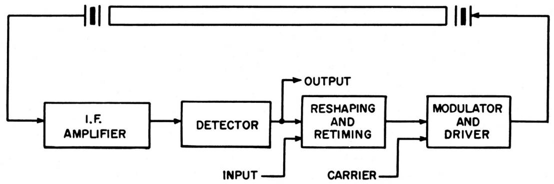


Image Source: wikipedia

Block diagram of the mercury memory system.

Delay-line memory



Block diagram of the mercury memory system.

- Bits encoded using waveform + time
  - **Sequential access** (not random)
    - Must wait for wave to circle around to desired bit
  - Capacity determined by number of tubes x length of tube
    - Longer tubes increase access time (and, to a lesser degree, cost)
    - More tubes increase cost
- Volatile memory: Bits lost once powered off
- Lots of engineering effort to deal with environmental variation (e.g., temperature, clock variations)
  - Mercury later replaced with magnetism, quartz, and electric delay lines for faster, lower variance
  - Not used today; I believe largely because of clock variance

Historical Shout-Out

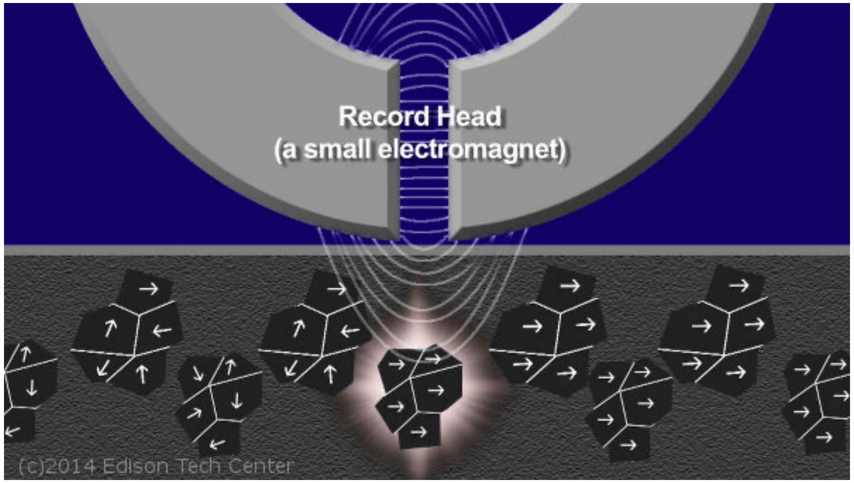
- Delay-lines were originally used in radar applications
  - Delay-line memory invented by Presper Eckert, who worked first in radar, then computers
  - Patented in 1947 by Eckert and John Mauchly
  - Used in 2^{nd} real computer: EDSAC
    - 16 delay lines, 560 bits each
- Today, ACM/IEEE award for major contributions to the field of computer architecture is named for them: the Eckert-Mauchly award
  - Given in 1983 to Kilburn
  - And in 2004 to our own Fred Brooks

Key design questions: Delay-line memory

- What is the physical phenomenon?
  - Sound wave in mercury
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Susceptible to acoustic interference, requires constant refreshing
- How to scale capacity, vs cost?
  - More tubes or longer tubes
- Other engineering constraints or performance anomalies?
  - Clock variance is a bummer, sequential access within a tube

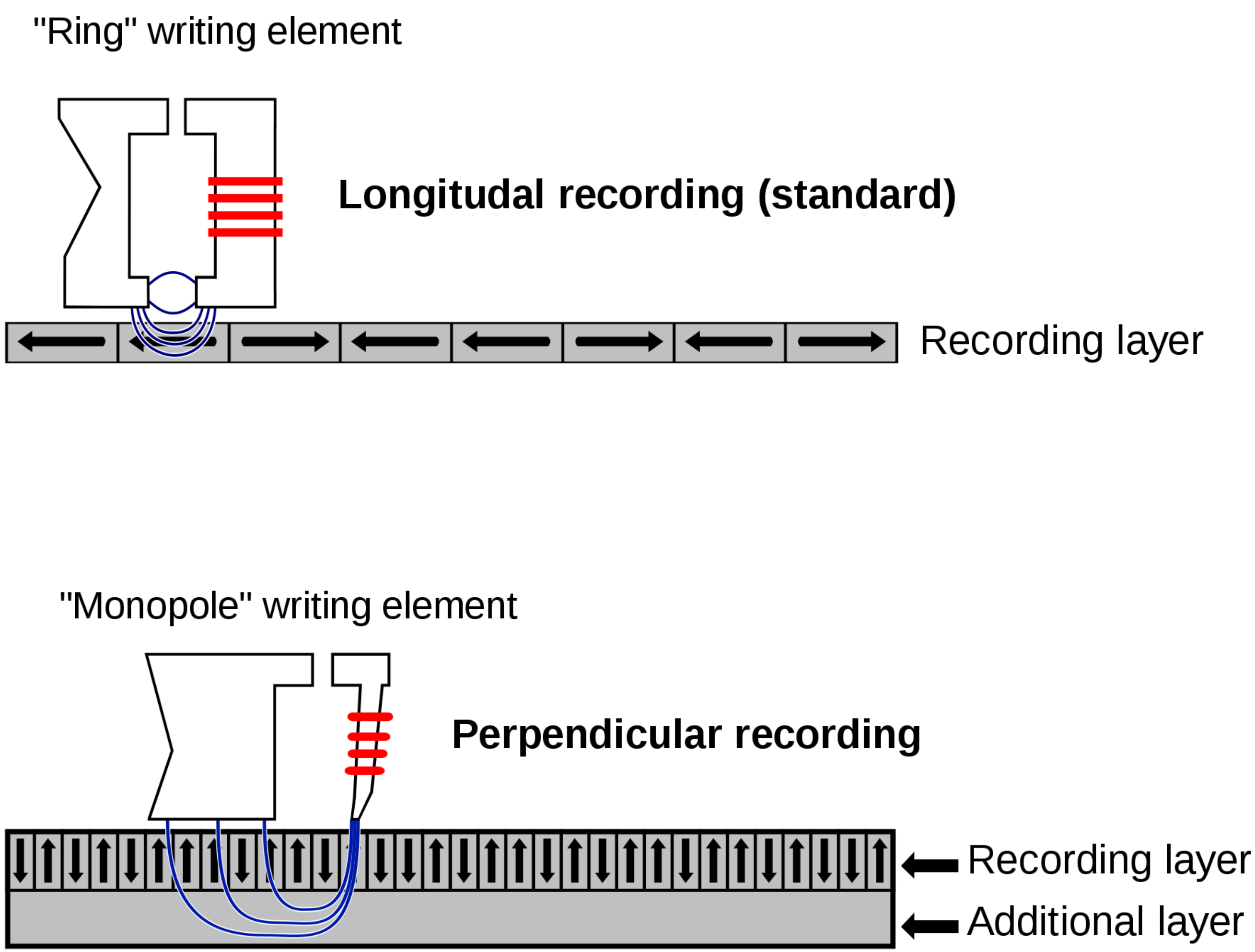
Next Big Idea: Magnetic Recording

- A powerful electromagnet can change the polarity of some materials, such as iron oxide

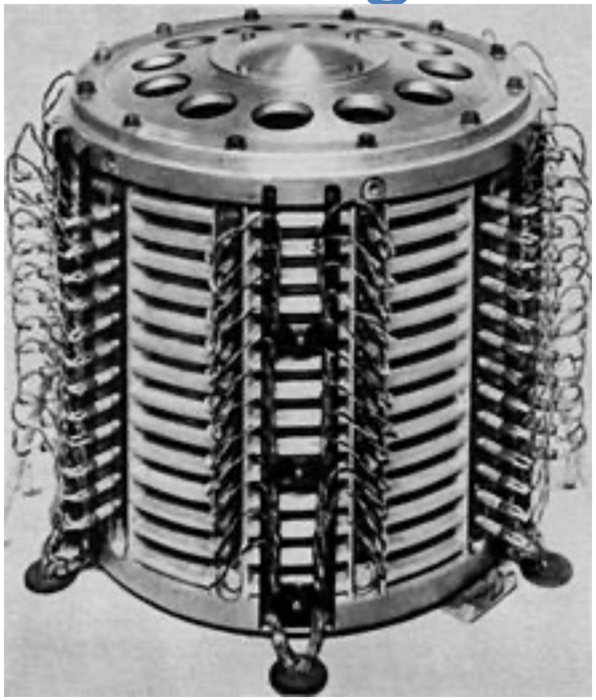


- Media can hold polarity arrangement for decades
  - All materials eventually succumb to entropy
  - Refresh data every 50 years, rather than every .5 seconds!!

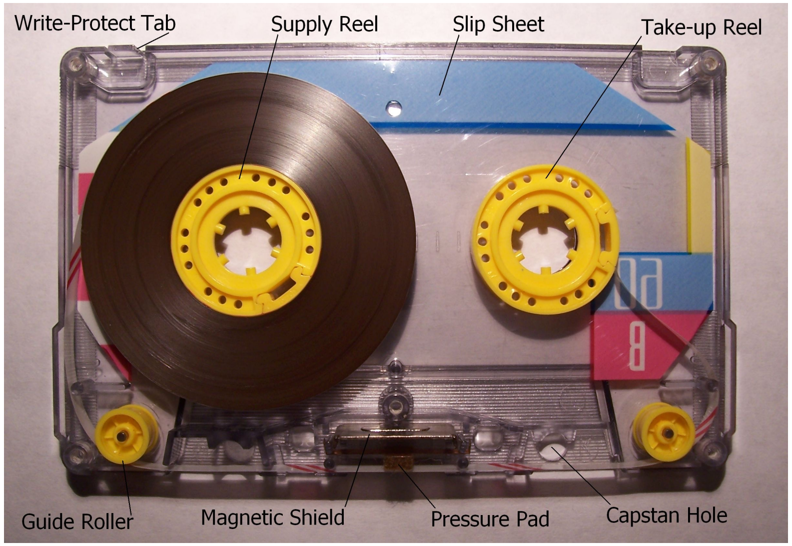
Another illustration, from wikipedia



Magnetic Recording over time



Drum memory, 1932-60s
From wikipedia



Cassette tape
From wikipedia

Hard disk
From wikipedia

Many, many variants over time

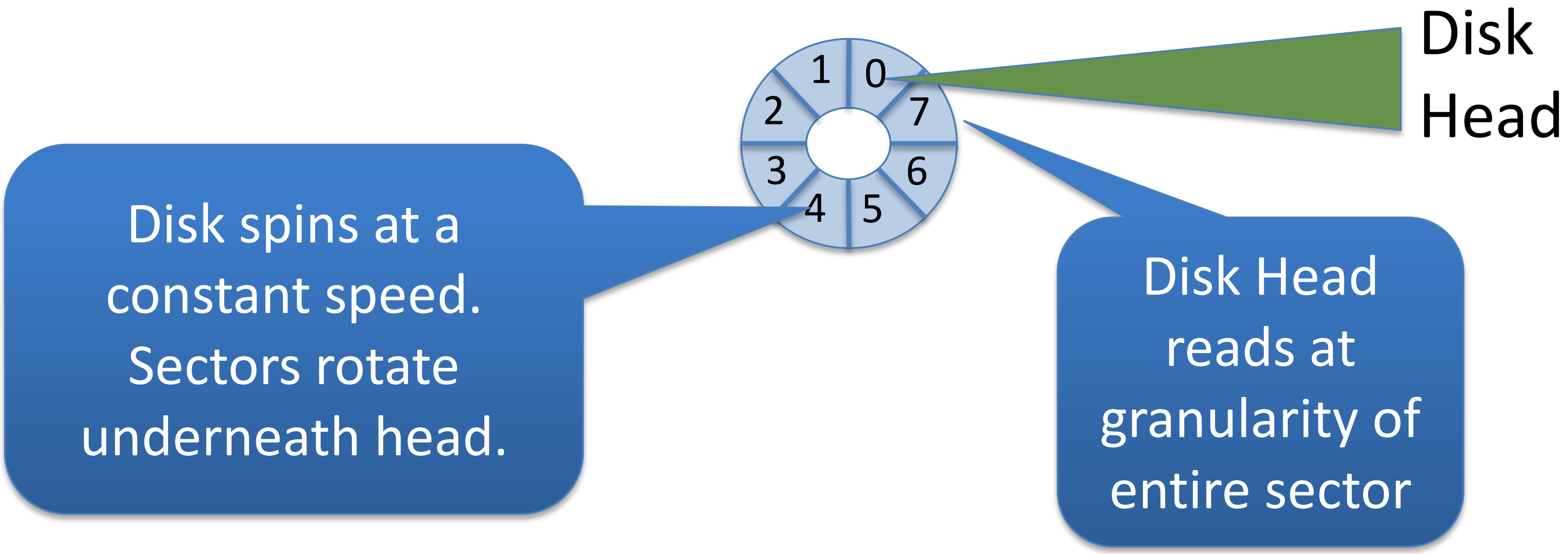
More on magnetic recording

- A *ton* of engineering to increase precision (and capacity)
  - Major cost in the encoding *head* that does encoding/decoding
  - Most designs have a small number of heads and move media under the head (e.g., spooling tape under the head)
- Magnetic recording is susceptible to being "erased" by adjacent magnetic fields
  - Engineered to resist weaker magnetic fields
  - Magnetic data loss typically requires a powerful magnet

A simple disk model

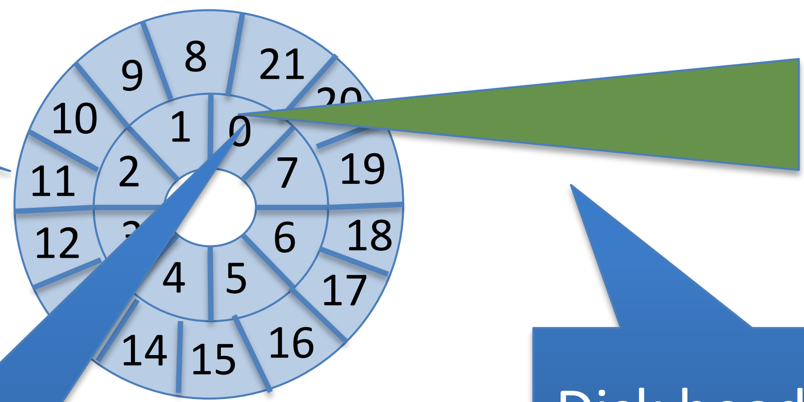
- Disks are slow. Why?
  - Moving parts << circuits
- Programming interface: simple array of sectors
  Physical layout:
  - Concentric circular “tracks” of sectors on a platter
  - E.g., sectors 0-9 on innermost track, 10-19 on next track, etc.
  - Disk arm moves between tracks
  - Platter rotates under disk head to align w/ requested sector

Disk Model


1
0
2
7
3
6
4
5
Disk Head
Disk spins at a constant speed. Sectors rotate underneath head.
Disk Head reads at granularity of entire sector

Disk Model

Concentric **tracks**

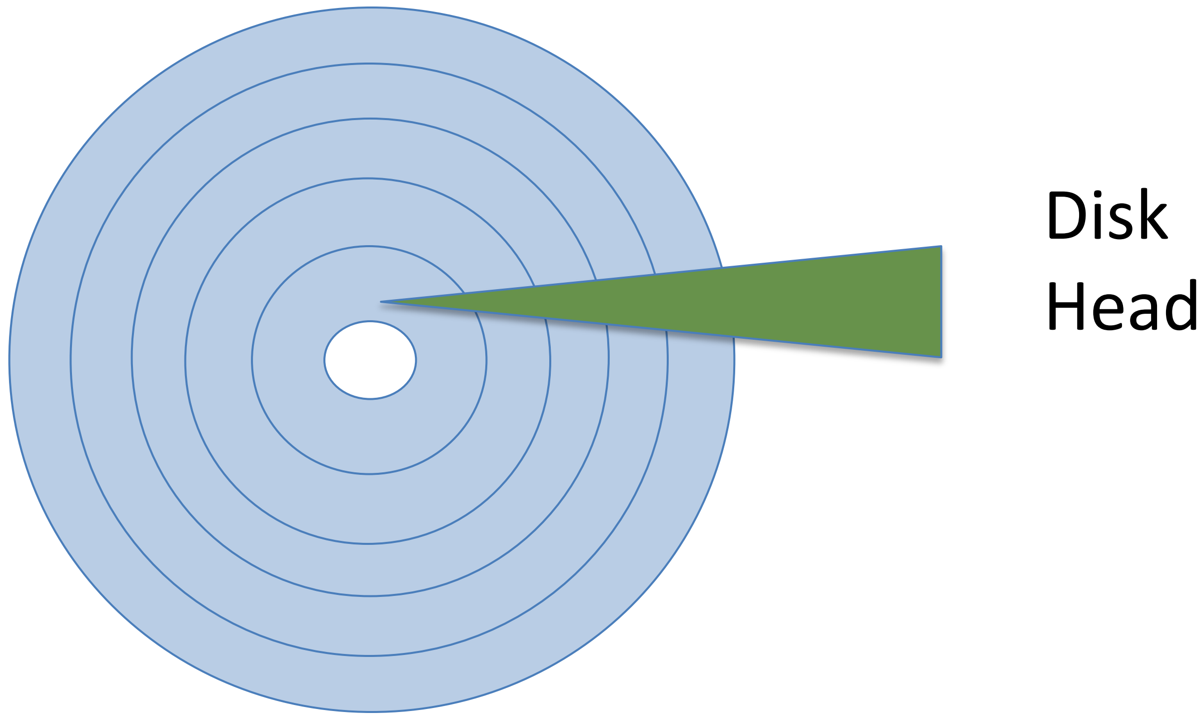


Disk
Head

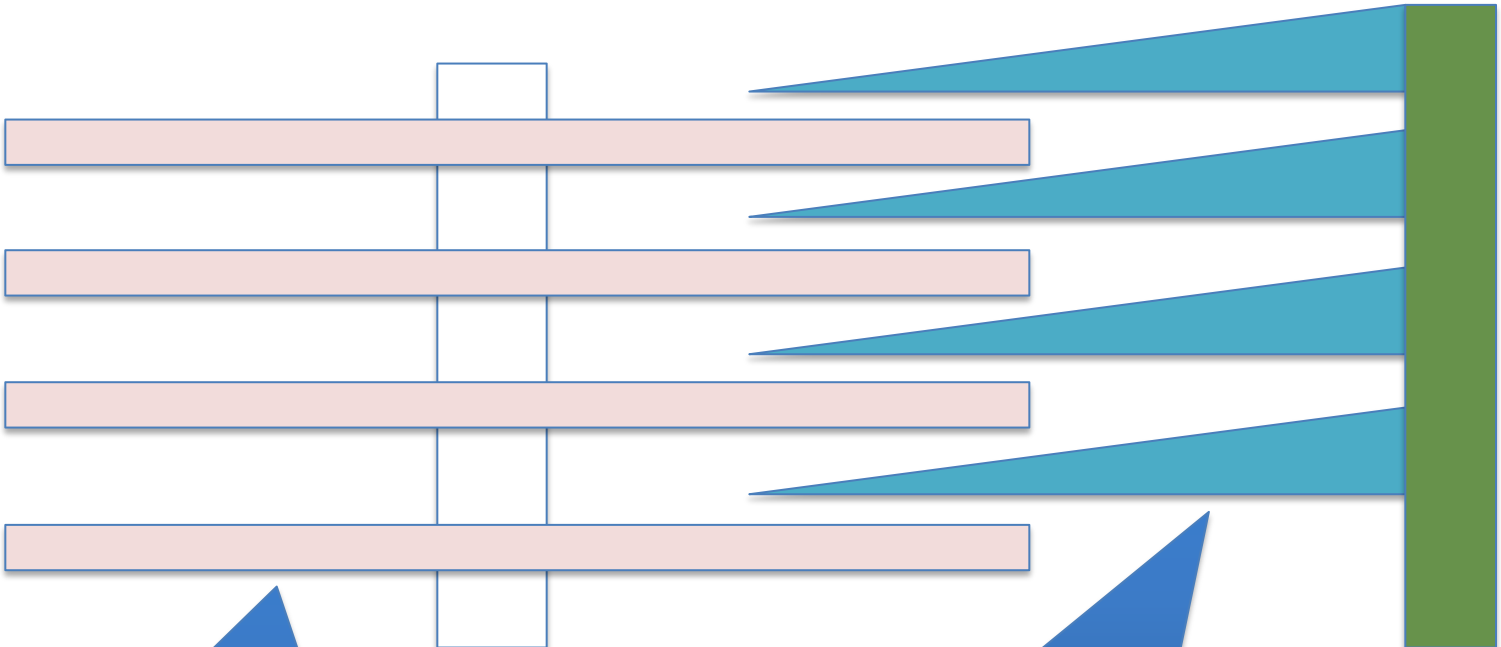
Gap between 7 and 8 accounts for seek time

Disk head **seeks** to different tracks

Many Tracks



Several (~4) Platters



Platters spin together at same speed

Each platter has a head; All heads seek together

Implications of multiple platters

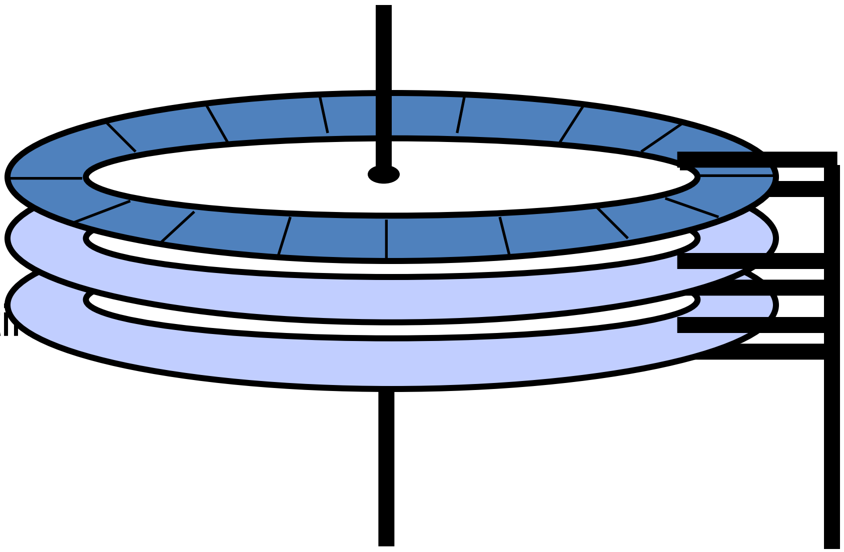
- Blocks actually striped across platters
- Also, both sides of a platter can store data
  - Called a **surface**
  - Need a head on top and bottom
- Example:
  - Sector 0 on platter 0 (top)
  - Sector 1 on platter 0 (bottom, same position)
  - Sector 2 on platter 1 at same position, top,
  - Sector 3 on platter 1, at same position, bottom
  - Etc.
  - 8 heads can read all 8 sectors simultaneously

Real Example

- Seagate 73.4 GB Fibre Channel Ultra 160 SCSI disk

- Specs:
  - 12 Platters
  - 24 Heads
  - Variable # of sectors/track
  - 10,000 RPM
    - Average latency: 2.99 *ms*
  - Seek times
    - Track-to-track: 0.6/0.9 *ms*
    - Average: 5.6/6.2 *ms*
    - Includes acceleration and settle ti
  - 160-200 MB/s peak transfer rate
    - 1-8K cache

- 12 Arms
- 14,100 Tracks
- 512 bytes/sector



Disks: Technology Trends

- Disks are getting denser
  - More bits/square inch → small disks with large capacities
- Disks are getting cheaper
  - Well, in $/byte – a single disk has cost at least $50-100 for 20 years
  - 2x/year since 1991
- Disks are getting faster
  - Seek time, rotation latency: 5-10%/year (2-3x per decade)
  - Bandwidth: 20-30%/year (~10x per decade)
  - This trend is really flattening out on commodity devices; more apparent on high-end
- Increasingly esoteric constraints to increase density
  - Shingled Magnetic Recording, Interlaced Magnetic Recording, Heat-Assisted Magnetic Recording, etc.

Overall: Capacity improving much faster than perf.

Key design questions: Magnetic Hard Disks

- What is the physical phenomenon?
  - Magnetic polarity
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Susceptible to strong magnetic interference
  - Sensitive to physical disturbance (e.g., dropping or shaking it)
  - Data lasts for decades without a refresh
- How to scale capacity, vs cost?
  - More surfaces
  - More precision encoding (smaller surface area)
    - Heads not independent (too costly)
- Other engineering constraints or performance anomalies?
  - Latency for head movement

Dynamic RAM (DRAM)

- Encode data as charge in capacitors
  - “high charge” == 1, “low charge” == 0
- Reading the charge also discharges it, requiring a read to re-write the data
  - Charge leaks out after 1-10 seconds – not persistent
  - Thus, requires periodic refresh
- Circuits relatively cheap to replicate at scale
  - Relatively uniform access times across cells (no “drone”)
  - Power seems to be bottleneck to capacity

Attacks on DRAM (1)

- Cold boot attack: Dump a computer’s RAM contents
  – Without administrator account on OS
  – Requires physical possession of the computer
- Recall: Data retention a function of temperature
  – Longer when colder
- Literally put a laptop in the freezer
  – Leverage seconds of retention while unplugged to quickly take out DRAM and plug into another computer

Attacks on DRAM (2)

- Rowhammering: Flip bits in memory you don't have access to
  - E.g., in the kernel or another process's address space
- Observation: Frequent charge/discharge cycles can cause disturb charge in adjacent cells
- Idea: Repeatedly read your own memory
  - Flip bits in adjacent cells
- Leverage this to gain privilege

Key design questions: DRAM

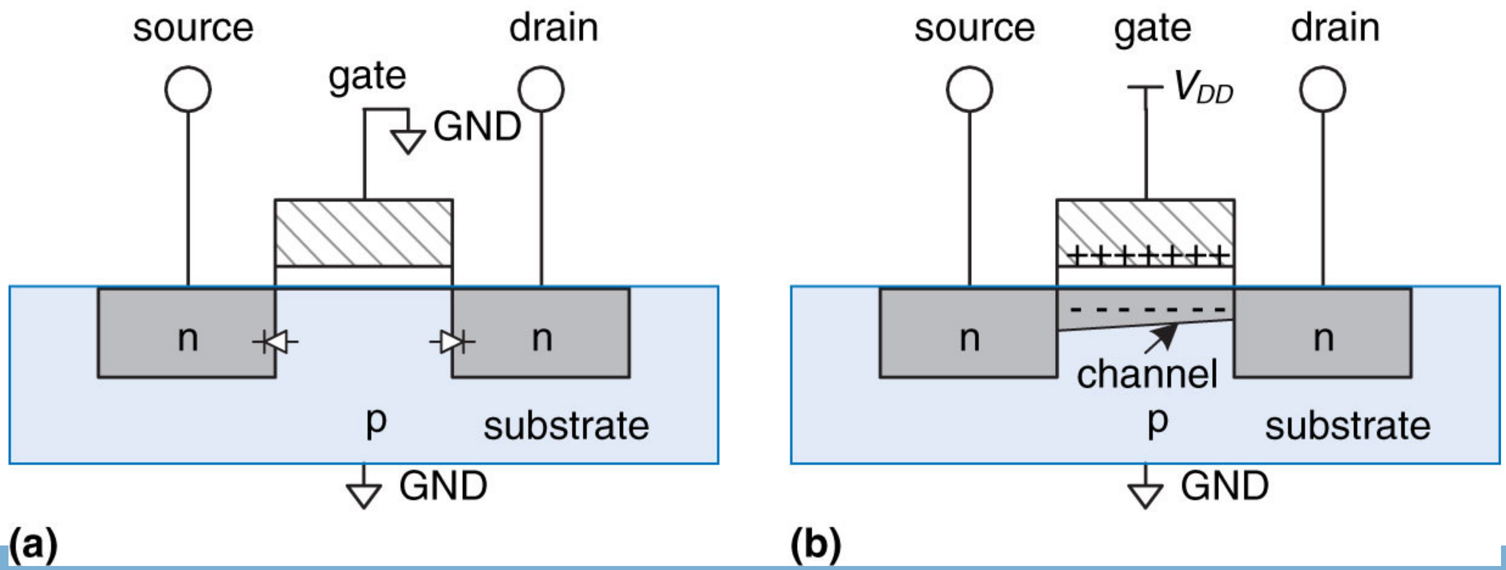
- What is the physical phenomenon?
  - Charge in a capacitor
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Susceptible to electrical disturbance
  - Data lasts for a few seconds
- How to scale capacity, vs cost?
  - Replicate circuits
- Other engineering constraints or performance anomalies?
  - Some performance variance from multiplexing

Flash Memory (aka EEPROM)

- A capacitor that holds its charge longer than DRAM
  - ~10 yrs fully unplugged
- Low-level physics is complex, based on quantum effects, I'll give intuition

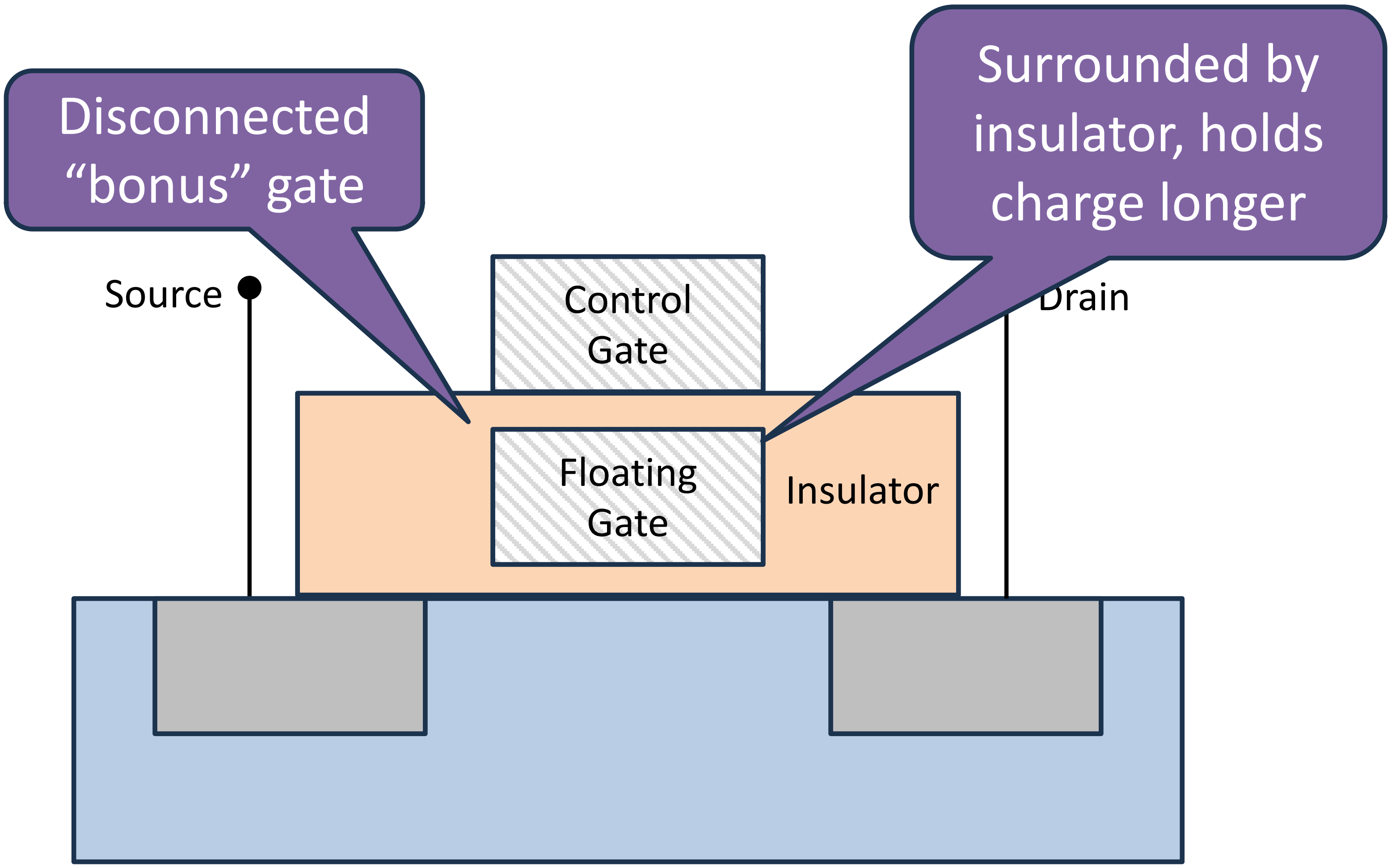
nMOS Transistors (stolen from COMP 411)

- Gate = 0
  - OFF = disconnect
    - no current flows between source & drain
- Gate = 1
  - ON= connect
    - current can flow between source & drain
    - positive gate voltage draws in electrons to form a channel

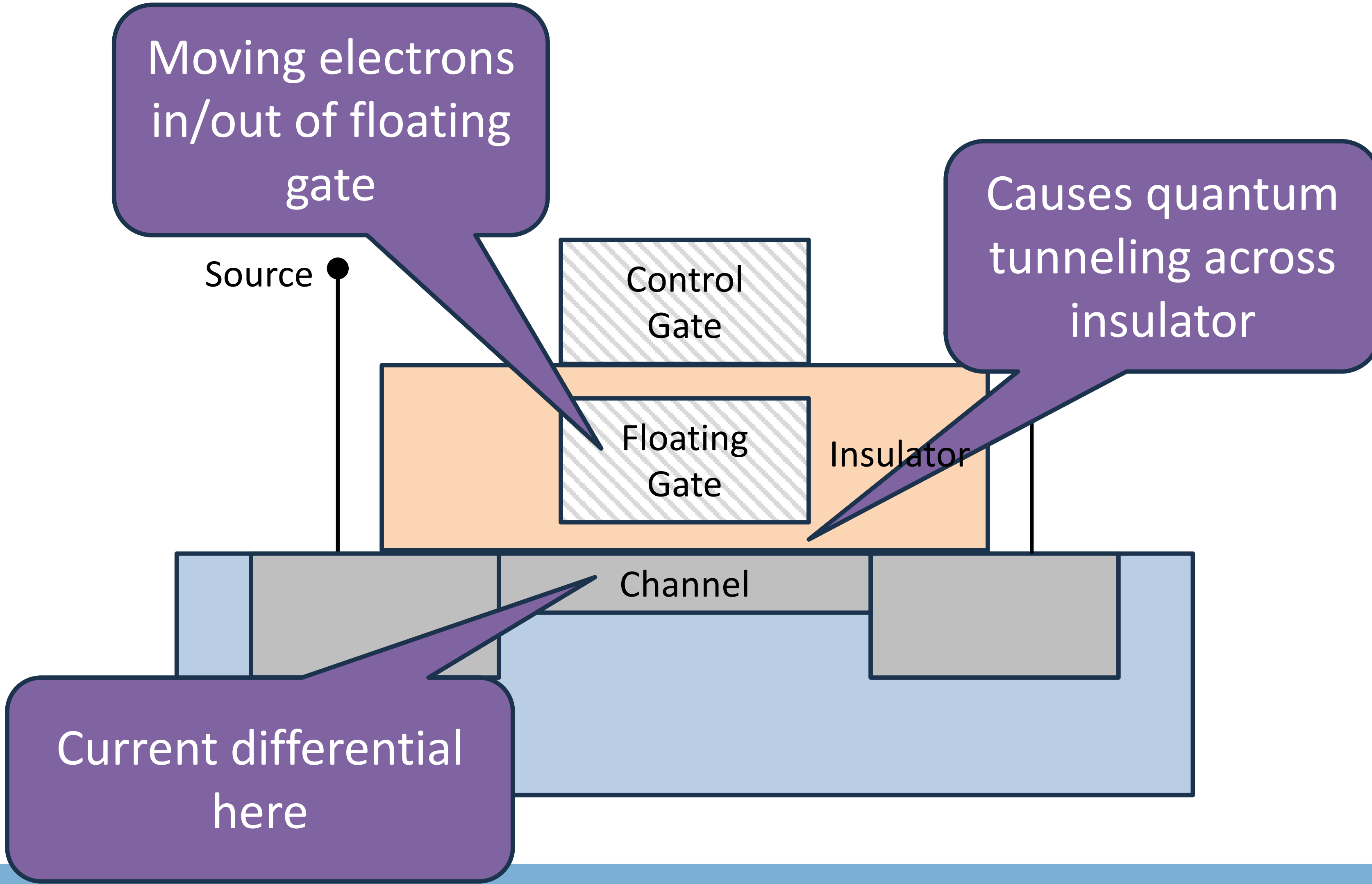


nMOS transistor operation (from Harris and Harris)

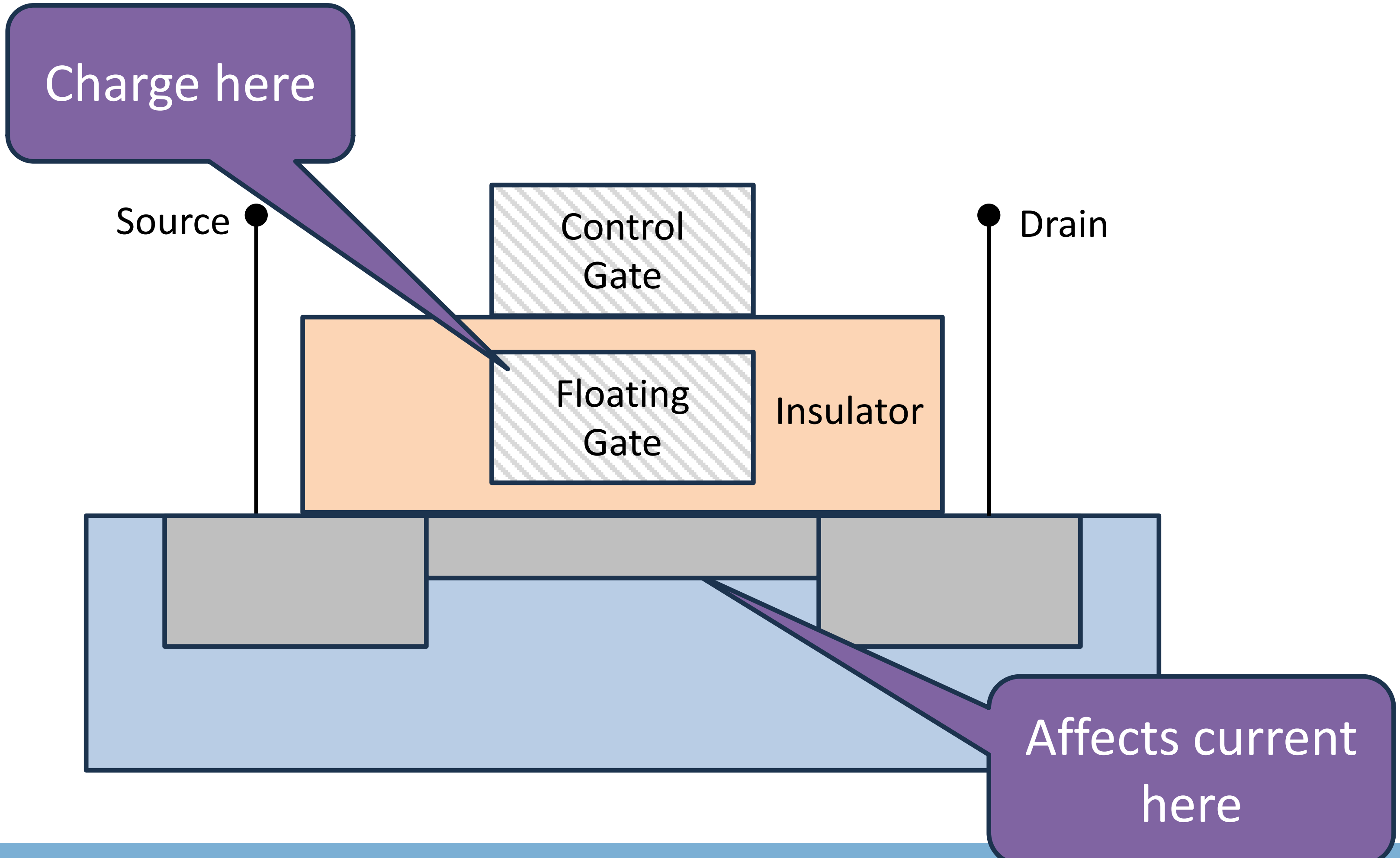
nMOS Transistor -> Flash



Manipulating Flash



Reading Flash



Flash details

- Like DRAM, cheap enough to replicate (nearly) everything – no expensive sensor to move around
- Program/erase via high current on control gate
  - No charge == 1
  - Program (add electrons to floating gate) to convert to a 0
  - Erase (remove electrons from flt gate) to pull back to a 1
- Sense charge level via current measurement of control gate

Flash Performance Caveat

- Erasing 7x slower than programming
- Trick: Hide erase cost by adding more cells than advertised
  - E.g., an 800 GB SSD may actually have 1 TB of cells
- For engineering reasons, write in KB blocks,
  - Erase in MB *erase blocks*
- Add a ”page table” to redirect writes for a logical block address (LBA) to a new location on each write
  - Called a Flash Translation Layer (FTL)

The FTL and SSD firmware

- The FTL is firmware (really software) that runs inside an SSD
- An SSD is really a small embedded computer
  - Often with a few ARM cores and 100s MB of DRAM
  - Runs software to implement the translation table
  - And manage actual reads/writes of flash cells
- In the background, FTL erases blocks
  - Try to go for mostly stale contents, then recycle space
  - If some live data in erase block, must first copy elsewhere

No longer a simple machine to reason about…

The TRIM command

- Indicates to the device a logical block (sector) is free
  - So FTL can avoid copying junk contents when erasing block
  - Abstraction introduced for flash (useful elsewhere)
- As opposed to leaving junk in place until overwrite
  - Harmless on disks, adds overhead on flash
- TRIM issued by file system to device (more soon…)

Scaling Flash

- Replication is one common strategy
  - New technology: 3D stacked flash
- Another is increasing precision
  - Rather than just measuring charged, not charged, measure charge more finely
  - And charge more carefully
  - This is Single Level Cell (1 bit/cell)
    - vs. Multi-Level Cell (2 bits) vs TLC (3 bits/cell) vs QLC (4 bits/cell) vs PLC (5 bits/cell)
  - Trade write speed (and endurance) for capacity
- Same capacity in SLC will be faster, last longer, (and cost more) than in MLC…PLC

Flash Endurance Caveats

- Over time, a cell wears out
  - Sending electrons across the insulator damages the insulator
  - Around 10,000 program/erase cycles, but varies
    - Reduced by higher precision encoding
    - Repeatedly adding electrons hastens wear
- Over time, electrons get stuck in the floating gate
  - Becomes more difficult to erase
  - And causes skew (bit flips) in higher precision encodings

Flash Wearout: A Real Thing

DESIGNLINES | AUTOMOTIVE DESIGNLINE

Flash Wearout Drives Tesla Recall

Overly large display may be too much for even automotive-qualified eMMC

By Gary Hilson 02.08.2021 3

M1 Mac Users Report Excessive SSD Wear

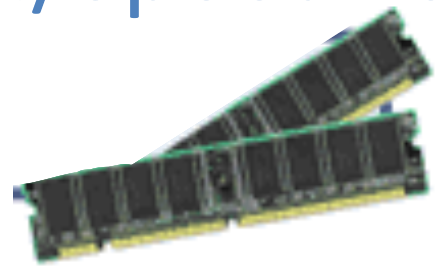
Tuesday February 23, 2021 7:07 am PST by Hartley Charlton

Over the past week, some M1 Mac users have been reporting alarming SSD health readings, suggesting that these devices are writing extraordinary amounts of data to their drives (via *iMore*).

Key design questions: Flash

- What is the physical phenomenon?
  - Charge in a floating gate
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Limited writes, but high tolerance for physical damage
  - Data lasts for about 10 years
- How to scale capacity, vs cost?
  - Replicate circuits *and* increase precision
- Other engineering constraints or performance anomalies?
  - Large erase blocks -> Copying/erasing in FTL

Big picture: Cost/Speed vs Size



- In 2023:

	Cost:	Max Capacity Sold in one dev
– DRAM:	$1.94 / GB	32 GB
– SSD:	$0.03/GB	16 TB
– HDD:	$0.02/GB	22 TB

- Note: Cost picked based on first non-sponsored hit on newegg.com (i.e., random sample), capacity based on maximum available

Today's Lecture

- How do computers store and access bits?
- Review current and emerging storage technologies
  - Hard Disk Drives (HDDs)
  - Solid State Drives (SSDs, aka flash)
- Reasoning about volatility vs. persistence
- Key trade-offs
- How to optimize I/O performance
- Practical miscellany
- Emerging media

How to Optimize Storage Performance?

- Two key techniques:
  - Large IOs and locality on device
  - I/O scheduling

OS's view of a storage device

- Simple array of sectors
  - Sectors are usually 512 or 4k bytes
  - Also called Logical Block Addresses (LBAs)
    - Captures virtual address space that device exports to OS
- OS can issue reads/writes to disk as small as one sector
- Depending on how data is placed on device, can also aggregate into larger requests
  - One contiguous LBA range and operation (read/write) per IO request

The Case for Larger I/Os

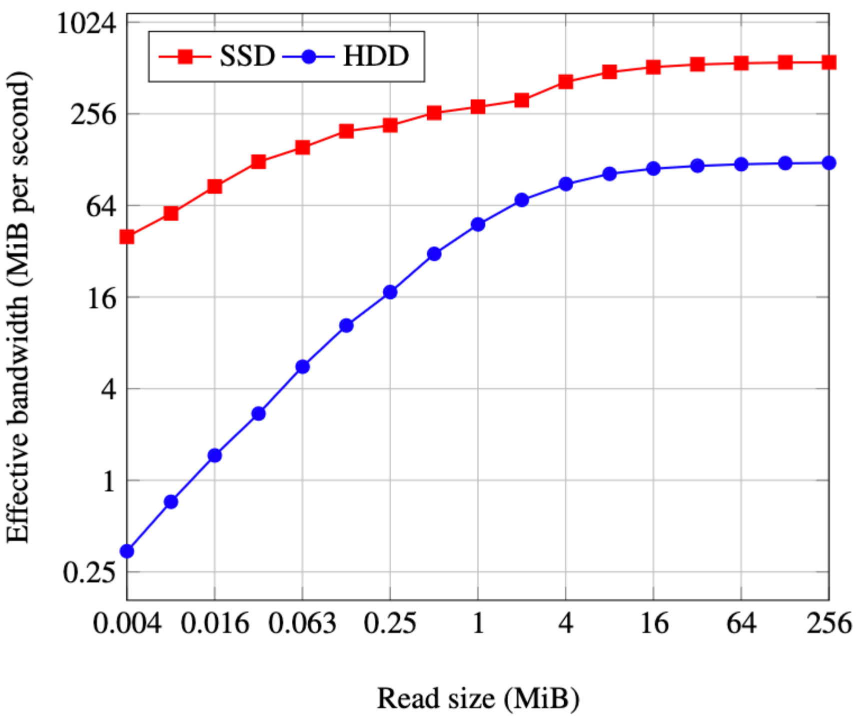


Figure 1: Effective bandwidth vs. read size (higher is better). Even on SSDs, large I/Os can yield an order of magnitude more bandwidth than small I/Os.

From Conway et al, “Filesystems Fated for Senescence?...”FAST ‘17

Large IOs

- Regardless of device or internals,
  - Fewer, big IO requests >> more small ones
- How to ensure large requests?
  - Heavily affected by placement on device by file system
    - (a topic for next lecture)
  - Or hold requests in memory for a bit, to see if they can be combined
    - Very common optimization to delay writes for a few seconds

The Disk Scheduling Problem: Background

- Goals: Maximize disk throughput
  - Bound latency
- Between file system and disk, you have a queue of pending requests:
  - Read or write a contiguous logical block address (LBA) range
- You can reorder these as you like to improve throughput
- What reordering heuristic to use? If any?
- Heuristic is called the **IO Scheduler**
  - Or "Disk Scheduler" or "Disk Head Scheduler"

Let's Start with Hard Disks

- Latency of a given operation is a function of current disk arm and platter position
- Each request changes these values
- Idea: build a model of the disk
  - Maybe use delay values from measurement or manuals
  - Use simple math to evaluate latency of each pending request
  - Greedy algorithm: always select lowest latency

3 Key HDD Latencies

- I/O delay: time it takes to read/write a sector
- Rotational delay: time the disk head waits for the platter to rotate desired sector under it
  - Note: disk rotates continuously at constant speed
- Seek delay: time the disk arm takes to move to a different track

Example formula

- s = seek latency, in time/track
- r = rotational latency, in time/sector
- i = I/O latency, in seconds

- Time = (Δtracks * s) + (Δsectors * r) + I
- Note: Δsectors must factor in position after seek is finished. Why?

Example read time:
seek time + *latency* + *transfer time*
(5.6 *ms* + 2.99 *ms* + 0.014 *ms*)

Practical Simplification

- Most hard disks don't export low-level geometry
  - But LBA layout (mostly) sequential on disk
- In practice, use LBA distance as proxy for track distance
  - I.e., we may not know exactly how many tracks an IO request crosses in practice, but we can assume bigger gaps in LBA space correspond to more tracks to cross

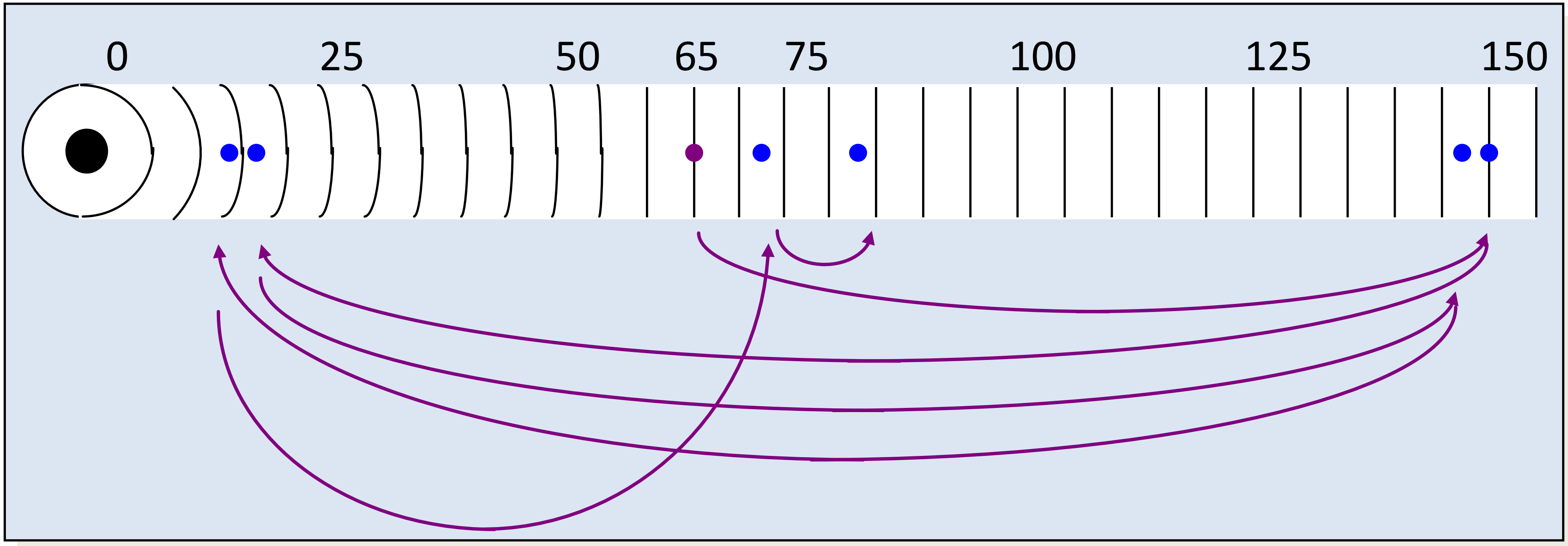
I/O Scheduling Algorithm 1: FCFS

- Assume a queue of requests exists to read/write tracks:

83	72	14	147	16	150

and the head is on track 65

... Incoming reqs → Next req



FCFS: Moves head 550 tracks

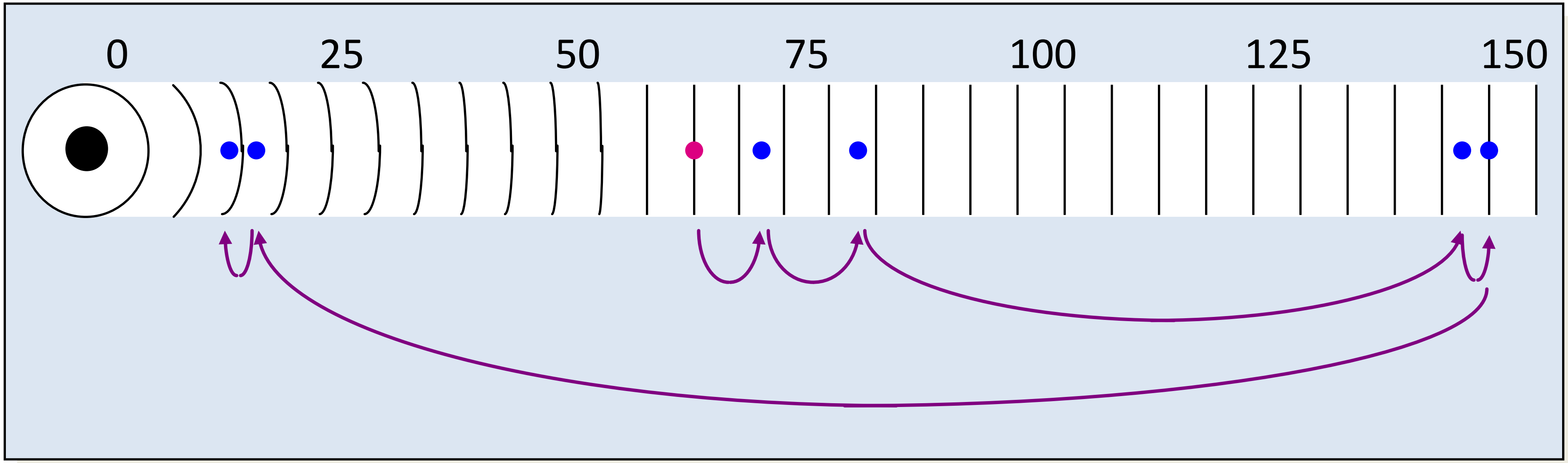
I/O Scheduling Algorithm 2: SSTF

- Greedy scheduling: *shortest seek time first*
  - Rearrange queue from:

83	72	14	147	16	150

To:

14	16	150	147	82	72



SSTF scheduling results in the head moving 221 tracks
Can we do better?

SSTF: 221 tracks (vs 550 for FCFS)

Other problems with greedy?

- "Far" requests will starve
  - Assuming you reorder every time a new request arrives
- Disk head may just hover around the "middle" tracks

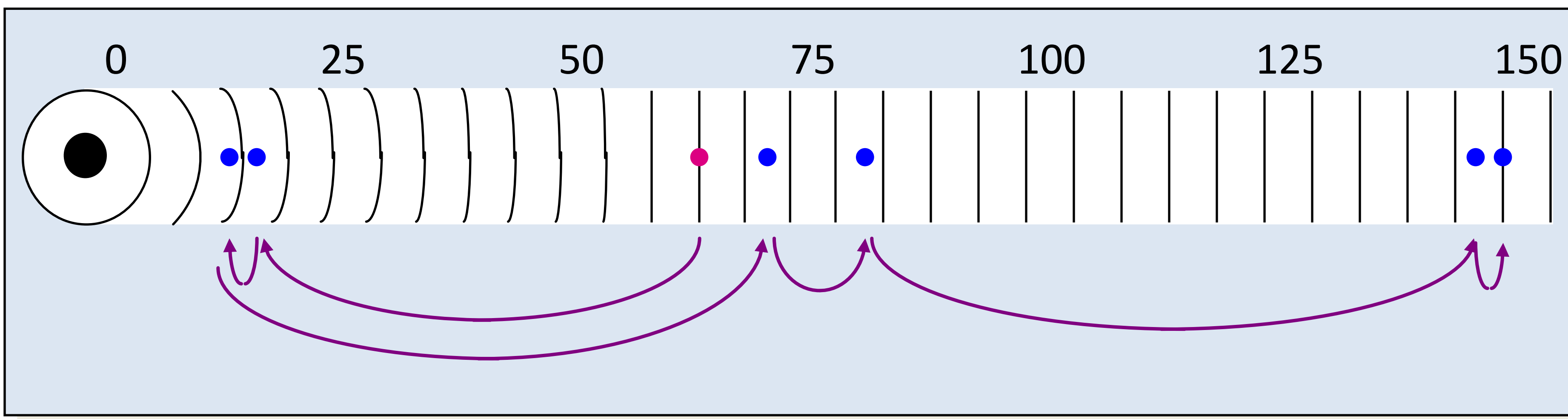
I/O Scheduling Algorithm 3: SCAN

- Move the head in one direction until all requests have been serviced, and then reverse.
- Also called Elevator Scheduling
- Rearrange queue from:

83	72	14	147	16	150

To:

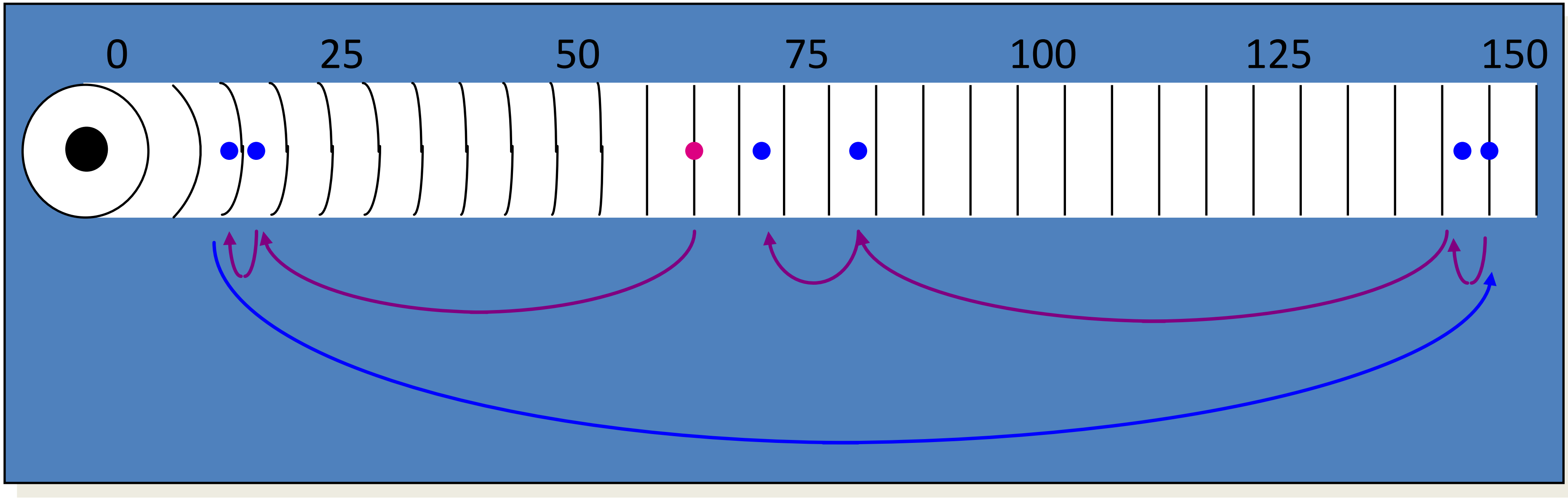
150	147	83	72	14	16



SCAN: 187 tracks (vs. 221 for SSTF)

I/O Scheduling Algorithm 4: C-SCAN

- Circular SCAN: Move the head in one direction until an edge of the disk is reached, and then reset to the opposite edge



- Marginally better fairness than SCAN

C-SCAN: 265 tracks (vs. 221 for SSTF, 187 for SCAN)

Scheduling Checkpoint

- SCAN seems most efficient for these examples
  - C-SCAN offers better fairness at marginal cost
  - Your mileage may vary (i.e., workload dependent)
- File systems would be wise to place related data ”near” each other
  - Files in the same directory
  - Blocks of the same file
- You will explore the practical implications of this model in Lab 5!

So what about IO scheduling for SSDs?

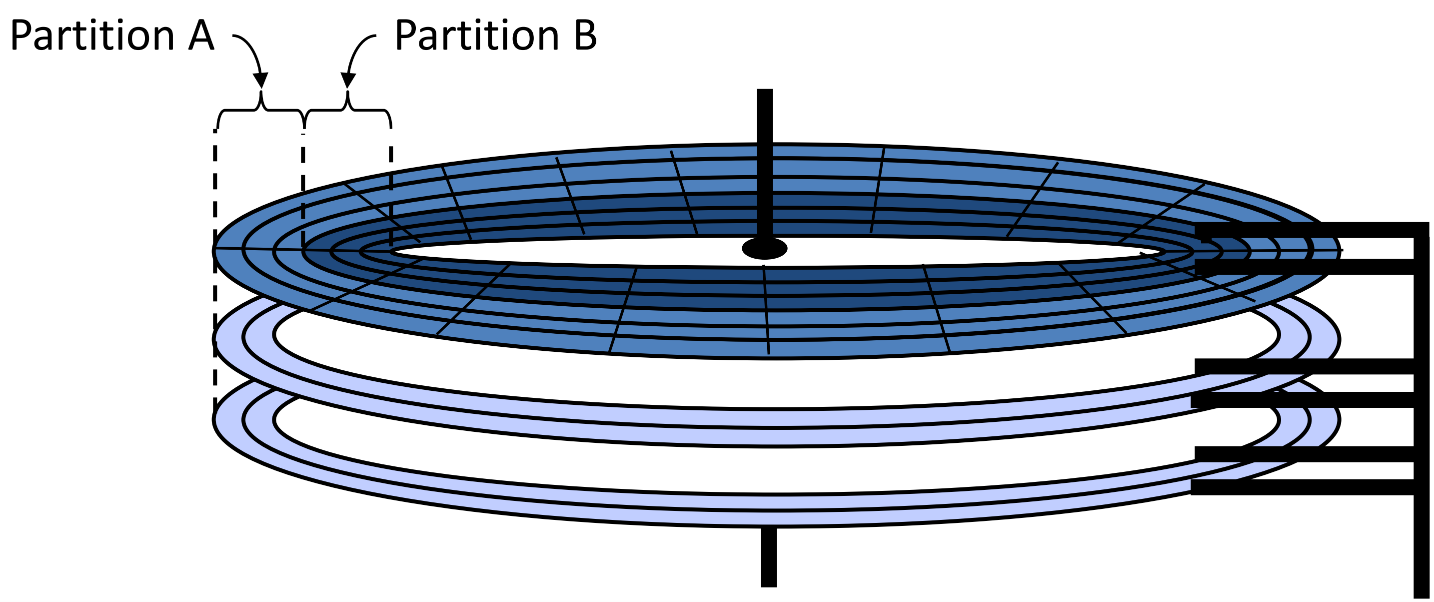
- Elevator scheduler is pointless on an SSD
  - LBA locality across requests irrelevant
  - Larger requests do help
- SSDs often use a no-op scheduler in practice
  - Or simply delay IO requests to combine
- Possible room for improvement in future
  - Devices themselves still evolving rapidly

Today's Lecture

- How do computers store and access bits?
- Review current and emerging storage technologies
  - Hard Disk Drives (HDDs)
  - Solid State Drives (SSDs, aka flash)
- Reasoning about volatility vs. persistence
- Key trade-offs
- How to optimize I/O performance
- Practical miscellany
- Emerging media

Disk Partitioning

- Multiple file systems can share a disk: **Partition** space
- Disks are typically partitioned to minimize the maximum seek time
  - A partition is a collection of cylinders
  - Each partition is a logically separate disk

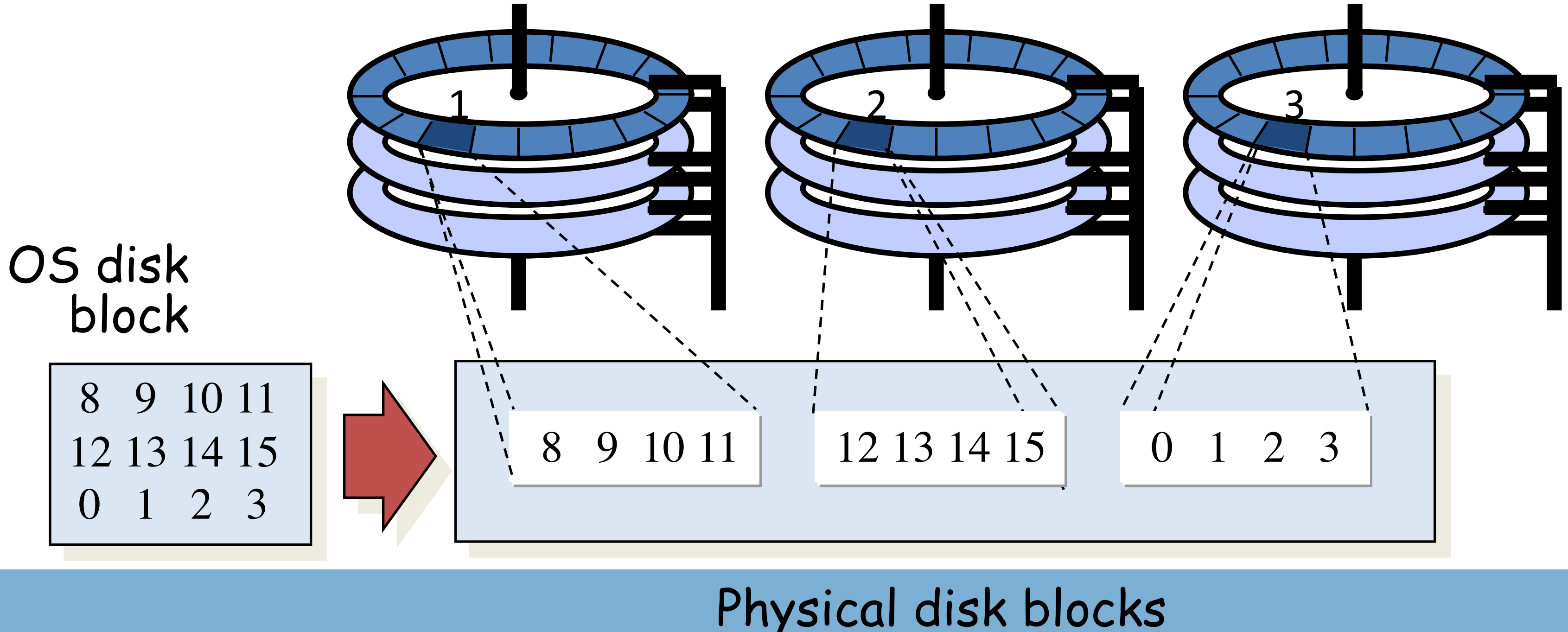


Parallel performance with disks

- Idea: Use more of them working together
  - Just like with multiple cores
- Redundant Array of Inexpensive Disks (RAID)
  - Intuition: Spread logical blocks across multiple devices
  - Ex: Read 4 LBAs from 4 different disks in parallel
- Does this help throughput or latency?
  - Definitely throughput, can construct scenarios where one request waits on fewer other requests (latency)
- It can also protect data from a disk failure
  - Transparently write one logical block to 1+ devices

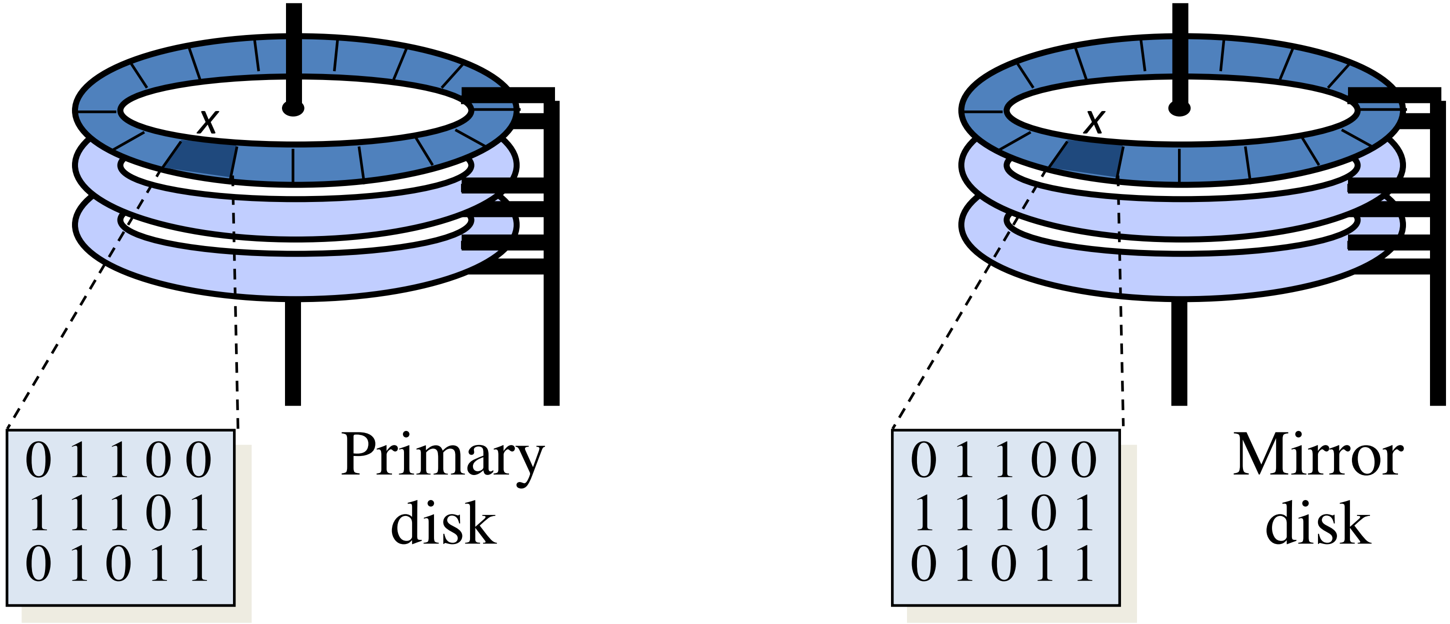
Disk Striping: RAID-0

- Blocks broken into sub-blocks that are stored on separate disks
  - similar to memory interleaving
- Provides for higher disk bandwidth through a larger effective block size



RAID 1: Mirroring

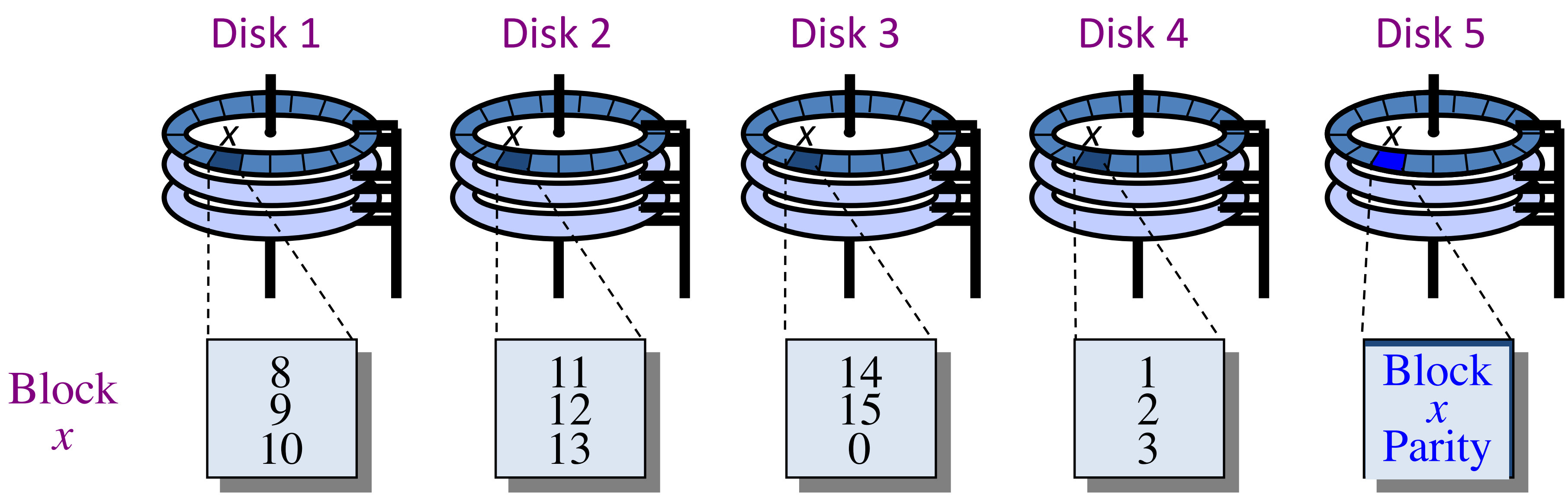
- To increase the reliability of the disk, redundancy must be introduced
  - Simple scheme: *disk mirroring (RAID-1)*
  - *Write to both disks, read from either.*



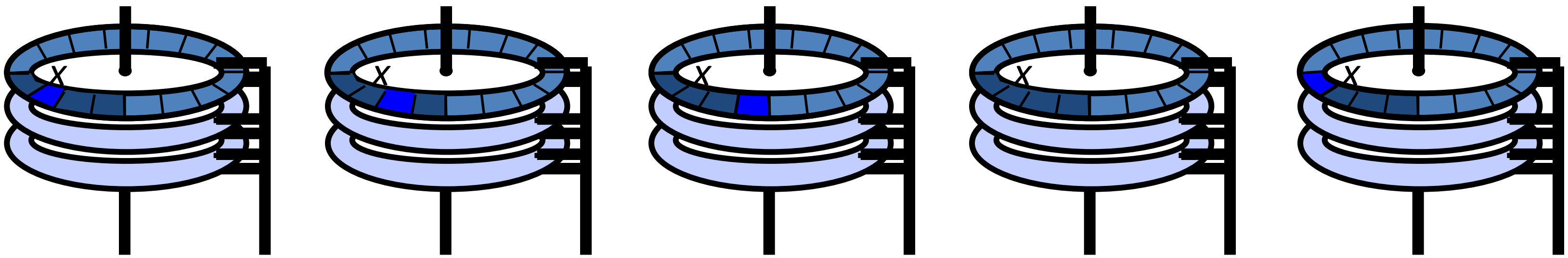
Can lose one disk without losing data

RAID 5: Performance and Redundancy

- Idea: Sacrifice one disk to store the parity bits of other disks (e.g., xor-ed together)
- Still get parallelism
- Can recover from failure of any one disk
- Cost: Extra writes to update parity



RAID 5: Interleaved Parity



	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
Block x	8 9 10	11 12 13	14 15 0	1 2 3	Block x Parity
Block x+1	Block x+1 Parity	*a* *b* *c*	*d* *e* *f*	*g* *h* *i*	*j* *k* *l*
Block x+2	*m* *n* *o*	Block x+2 Parity	*p* *q* *r*	*s* *t* *u*	*v* *w* *x*
Block x+3	*y* *z* *aa*	*bb* *cc* *dd*	Block x+3 Parity	*ee* *ff* *gg*	*hh* *ii* *jj*

Other RAID variations

- Variations on encoding schemes, different trades for failures and performance
  - See wikipedia
  - But 0, 1, 5 are the most popular by far
- More general area of **erasure coding**:
  - Store k logical blocks (message) in n physical blocks (k < n)
  - In an optimal erasure code, recover from any k/n blocks
  - Xor parity is a (k, k+1) erasure code
  - Gaining popularity at data center granularity

Where is RAID implemented?

- Hardware (i.e., a chip that looks to OS like 1 disk)
  - +Tend to be reliable (hardware implementers test)
  - +Offload parity computation from CPU
    - Hardware is a bit faster for rewrite intensive workloads
  - -Dependent on card for recovery (replacements?)
  - -Must buy card (for the PCI bus)
  - -Serial reconstruction of lost disk
- Software (i.e., a “fake” disk driver)
  - -Software has bugs
  - -Ties up CPU to compute parity
  - +Other OS instances might be able to recover
  - +No additional cost
  - +Parallel reconstruction of lost disk

Most PCs have “fake” HW RAID: All work in driver

Word to the wise

- RAID is a good idea for protecting data
  - Can safely lose 1+ disks (depending on configuration)
- But there is another weak link: The power supply
  - I have personally had a power supply go bad and fry 2/4 disks in a RAID5 array, effectively losing all of the data

RAID is no substitute for backup to another machine

Today's Lecture

- How do computers store and access bits?
- Review current and emerging storage technologies
  - Hard Disk Drives (HDDs)
  - Solid State Drives (SSDs, aka flash)
- Reasoning about volatility vs. persistence
- Key trade-offs
- How to optimize I/O performance
- Practical miscellany
- Emerging media

A Few Emerging Media

- Byte-addressable, non-volatile RAM
  - Resistive media: Memristor
    - Phase-Change Memory
- Capacity-optimized storage
  - Glass

Example: Phase-Change Memory (PCM)

- Chalcogenide glass: 2 chemical states as solid
  - Crystalline or amorphous solid
  - Depends on how fast it cools
  - Each state has different electrical resistivity and different optical refraction
- Optical refraction is how CDs, DVDs, Blu-Ray encode bits
- PCM uses resistivity
- Slower to write (must melt and cool cell)
  - Than to read (just measure current)
  - Overall performance <10x slower than DRAM, uses less power
- Retention projected at 300 years; no refresh needed
- Heating element does wear out (100m writes)
  - Bytes still readable

Key design questions: PCM

- What is the physical phenomenon?
  - Electrical resistivity of chalcogenide glass
- How robust is the physical phenomenon to environmental damage, or passage of time?
  - Limited writes, but high tolerance for physical damage
  - Data estimated to last 300 years (but limited experience)
- How to scale capacity, vs cost?
  - Replicate circuits; not clear if precision can be raised
- Other engineering constraints or performance anomalies?
  - Limited write endurance, but last value retained
  - Writes slower than reads

Memristors

- Several different materials under study manipulate resistivity
  - Generically called Memristor
    - ReRAM
    - Spin-Transfer Torque memory
- Active area of development, moving quickly
  - Motivated in part by difficulty scaling refresh of DRAM
    - Current prototypes slower than DRAM, within an order of magnitude
  - Will probably change during your career!

Microsoft's Project Silica

- Focus on very, very long-term storage
  - Example: Archiving UNC student data for >5 yrs ago
    - Don't need immediately, but occasional requests for a transcript
- Encode data with laser-etched, quartz glass
  - Write once, read many times
  - Very high endurance, retention
  - Cheap material
    - Some cost to move material to sensor (robot)
  - No cost to retain other than space
  - Too slow to try to compete with RAM

Silica Video

Editorial: Where is all this going?

- My personal guess: Market will segment
  - Really fast storage (compete with DRAM on speed)
  - Cheap, bulk storage (compete on total $/GB)
    - HDDs mostly moving in this direction
    - Some flash companies trying to compete here too
    - Costs likely to include operating $$ and carbon
- HDD and SSD internals will also get esoteric
  - Conventional scaling techniques have plateaued
  - Interesting experiments with new encoding techniques
  - And new constraints on updates in place (vs copying)
- Hybrid devices increasingly common

Summary

- Know the 4 key questions to ask about any key storage technology
  - Familiarity with HDD and Flash in particular
- Understand how to get good performance from storage
  - Large IOs (always), LBA locality (HDDs)
- Understand I/O scheduling algorithms
- Understand RAID, partitioning, TRIM