



Disks and I/O Scheduling

Don Porter

Portions courtesy Emmett Witchel



Quick Recap

- CPU Scheduling
 - Balance competing concerns with heuristics
 - What were some goals?
 - No perfect solution
- Today: Block device scheduling
 - How different from the CPU?
 - Focus primarily on a traditional hard drive
 - Extend to new storage media



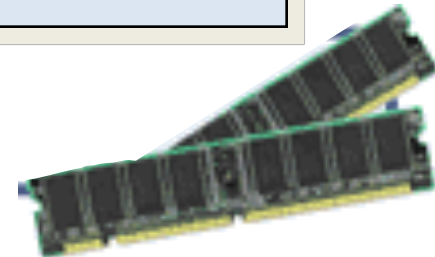
Disks: Just like memory, but different

- Why have disks?
 - Memory is small. Disks are large.
 - Short term storage for memory contents (e.g., swap space).
 - Reduce what must be kept in memory (e.g., code pages).
 - Memory is volatile. Disks are forever (?!)
 - File storage.



	GB/dollar	dollar/GB
RAM	0.013(0.015,0.01)	\$77(\$68,\$95)
Disks	3.3(1.4,1.1)	30¢ (71¢,90¢)

Capacity : 2GB vs. 1TB
2GB vs. 400GB
1GB vs 320GB





OS's view of a disk

- Simple array of blocks
 - Blocks are usually 512 or 4k bytes



A simple disk model

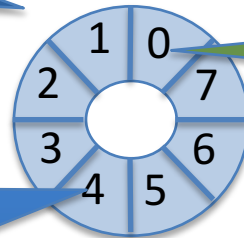
- Disks are slow. Why?
 - Moving parts \ll circuits
- Programming interface: simple array of sectors (blocks)
- Physical layout:
 - Concentric circular “tracks” of blocks on a platter
 - E.g., sectors 0-9 on innermost track, 10-19 on next track, etc.
 - Disk arm moves between tracks
 - Platter rotates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



Disk Model

Each block on
a sector

Disk spins at a
constant speed.
Sectors rotate
underneath head.

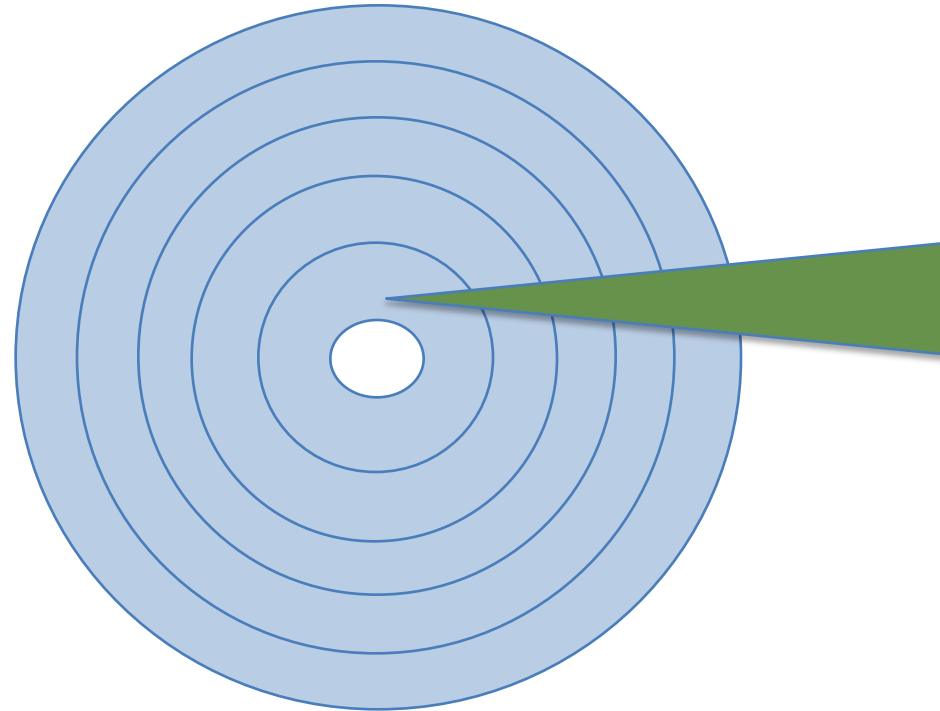


Disk
Head

Disk Head
reads at
granularity of
entire sector



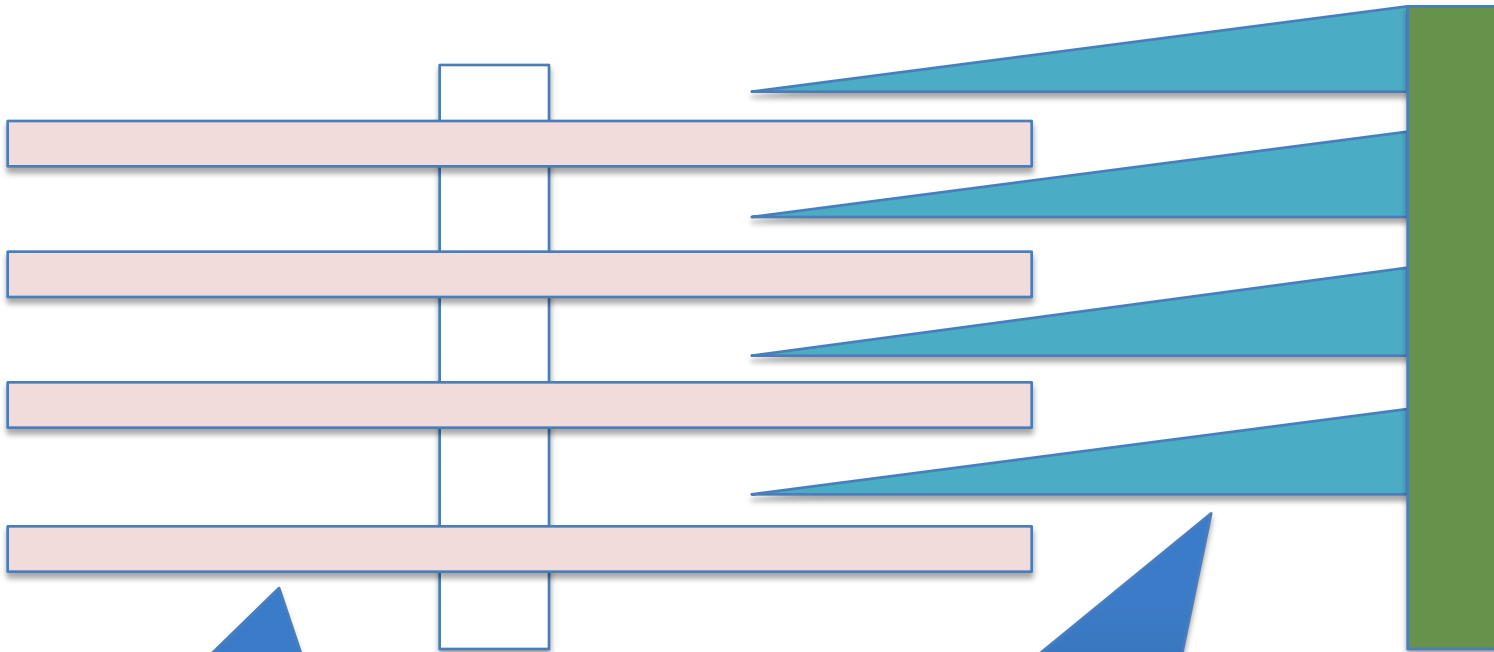
Many Tracks



Disk
Head



Several (~4) Platters



Platters spin
together at same
speed

Each platter has a head;
All heads seek together



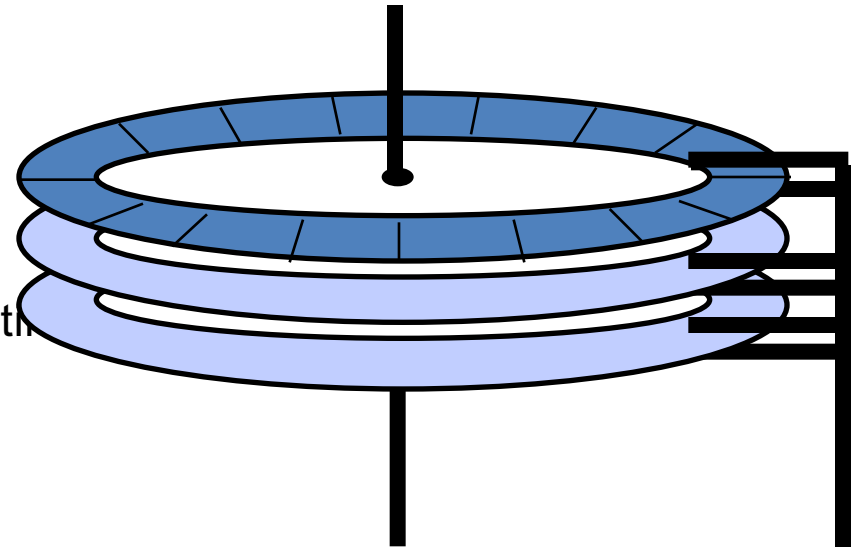
Implications of multiple platters

- 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 times
 - 160-200 MB/s peak transfer rate
 - 1-8K cache
- 12 Arms
 - 14,100 Tracks
 - 512 bytes/sector





3 Key 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



Observations

- 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



Example formula

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

- $\text{Time} = (\Delta\text{tracks} * s) + (\Delta\text{sectors} * r) + i$
- Note: $\Delta\text{sectors}$ 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)



The Disk Scheduling Problem: Background

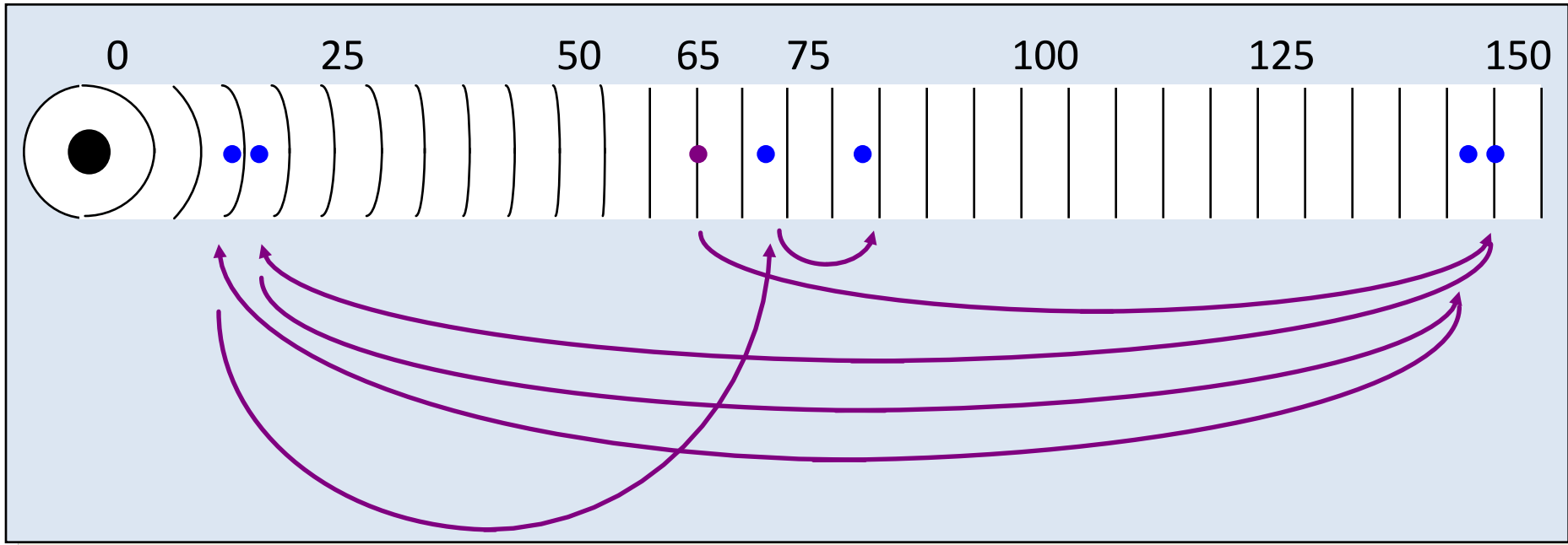
- Goals: Maximize disk throughput
 - Bound latency
- Between file system and disk, you have a queue of pending requests:
 - Read or write a given logical block address (LBA) range
- You can reorder these as you like to improve throughput
- What reordering heuristic to use? If any?
- Heuristic is called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

Evaluation: how many tracks head moves across¹⁵



I/O Scheduling Algorithm 1: FCFS

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



FCFS: Moves head 550 tracks



I/O Scheduling Algorithm 2: SSTF

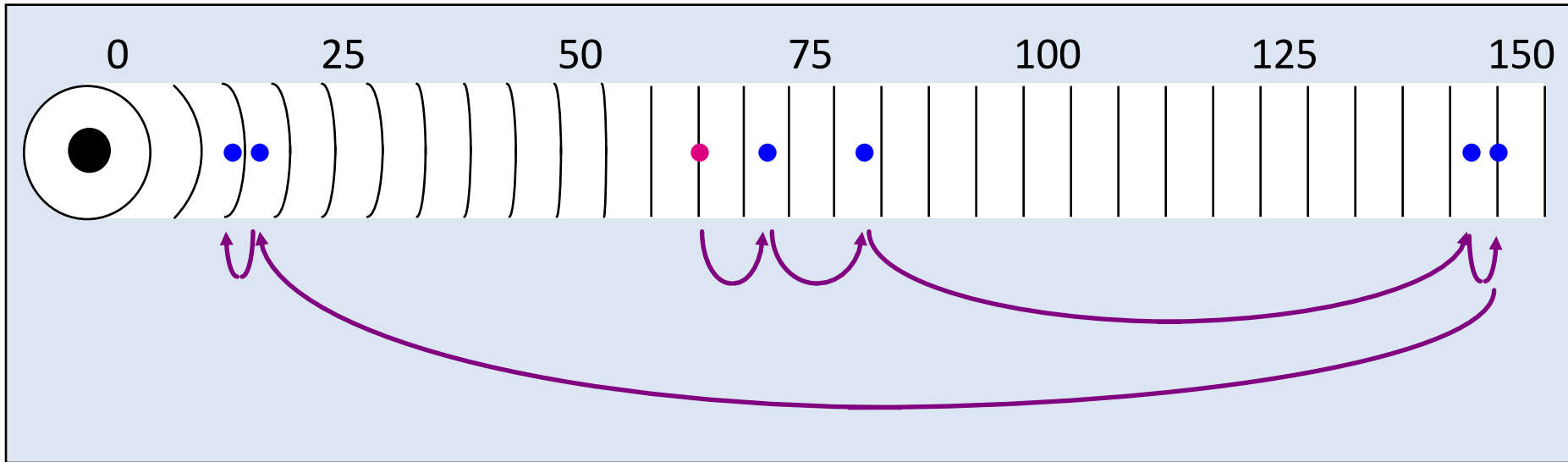
- Greedy scheduling: *shortest seek time first*

– 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)



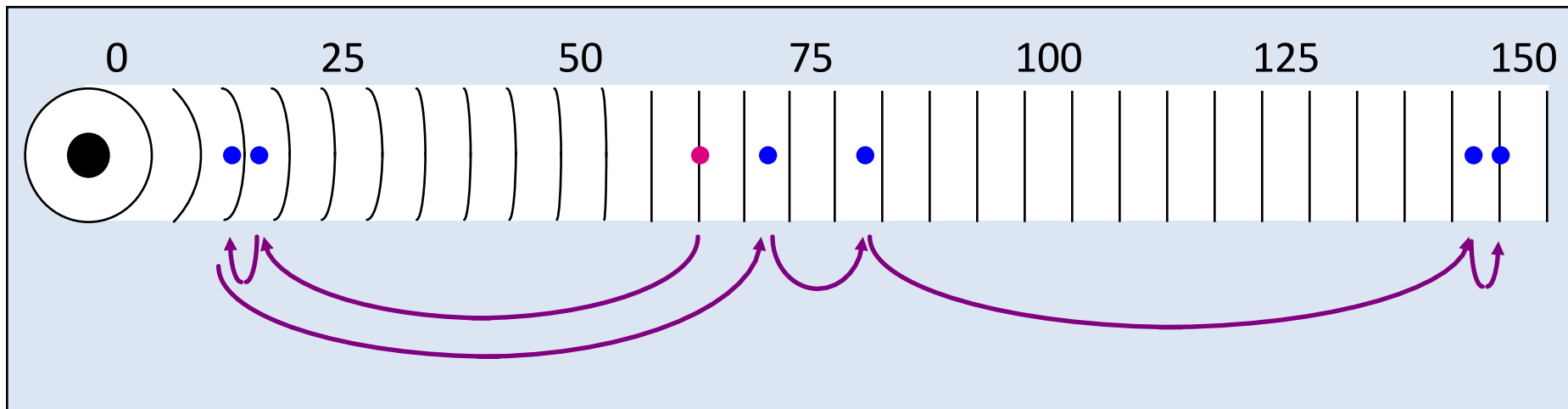
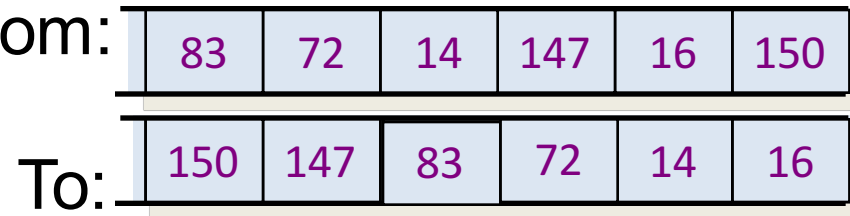
Other problems with greedy?

- “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:

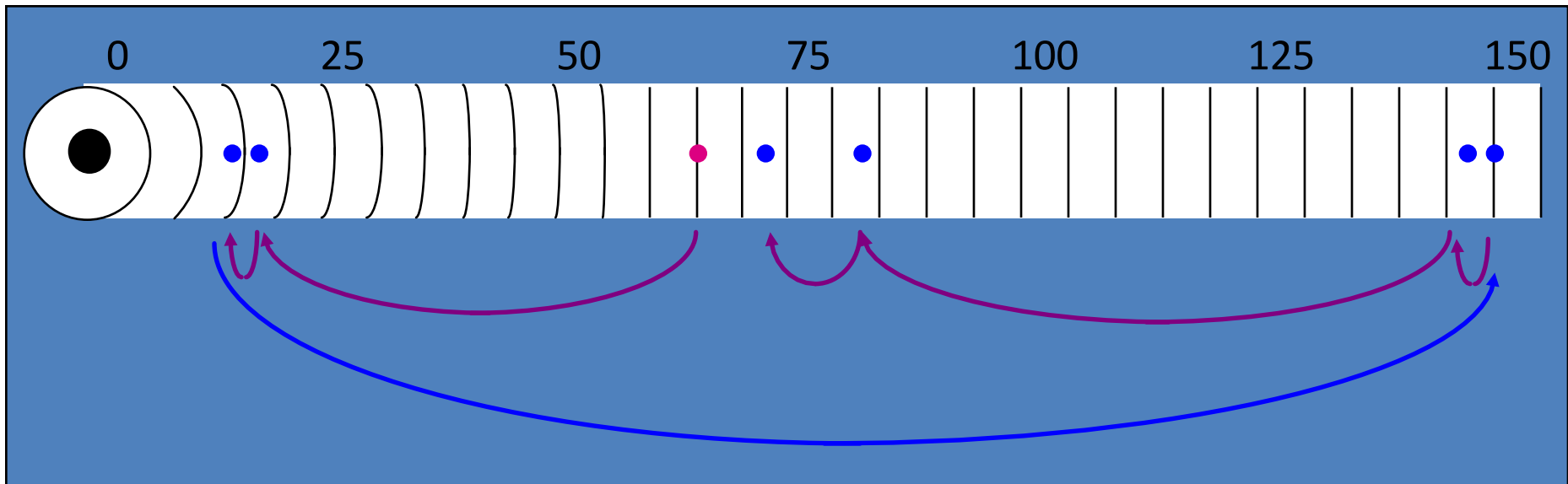


SCAN: 187 tracks (vs. 221 for SSTF)



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)



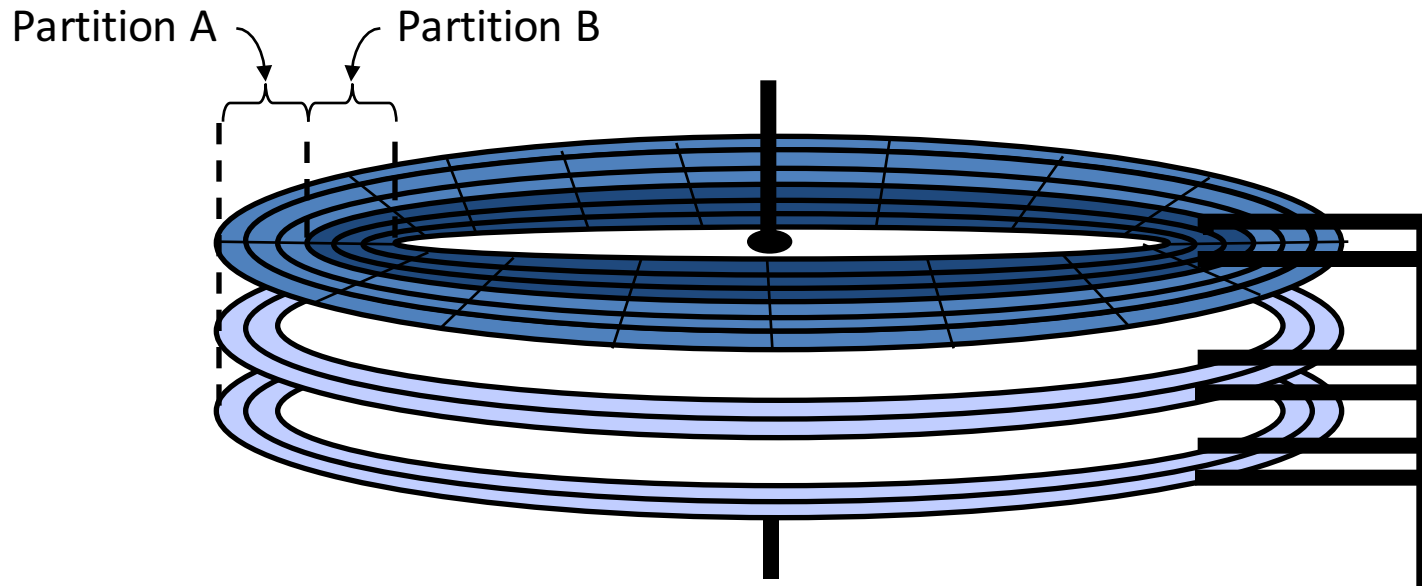
Scheduling Checkpoint

- 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 4!



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





Disks: Technology Trends

- Disks are getting smaller in size
 - Smaller → spin faster; smaller distance for head to travel; and lighter weight
- 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

Overall: Capacity improving much faster than perf.



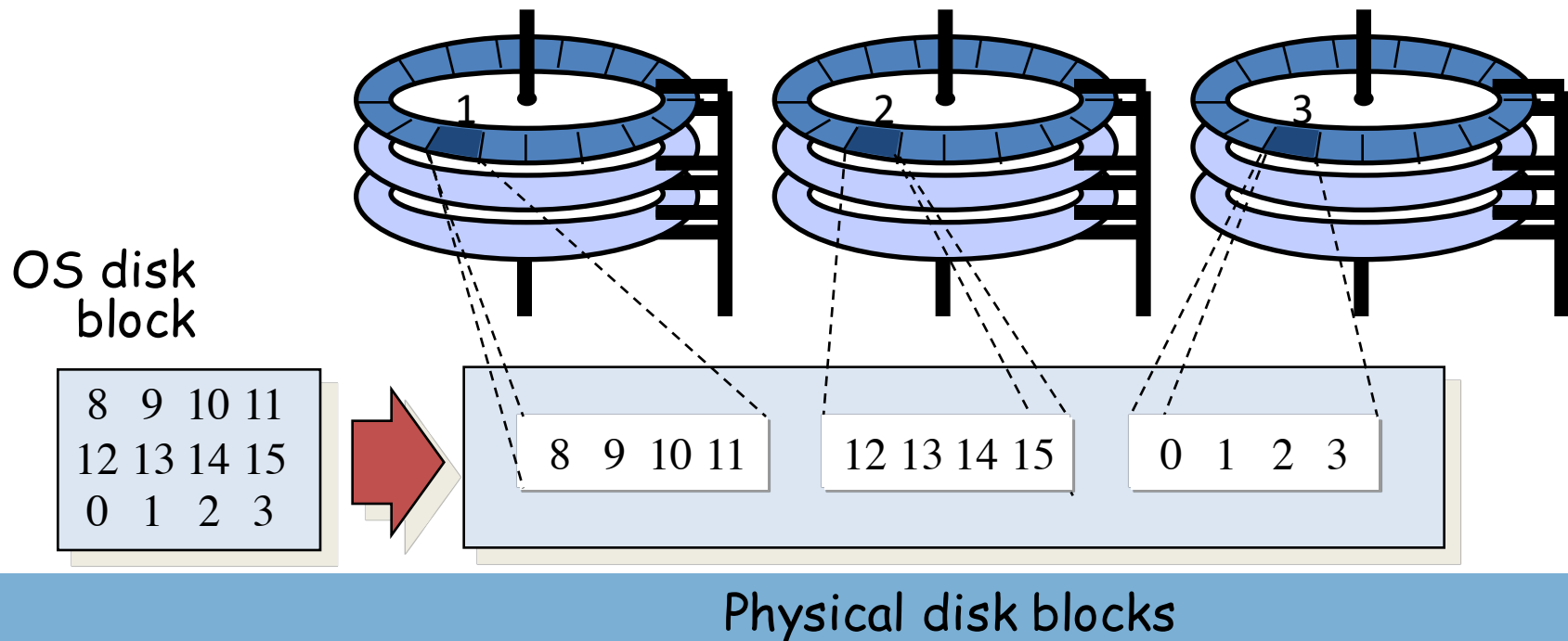
Parallel performance with disks

- 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

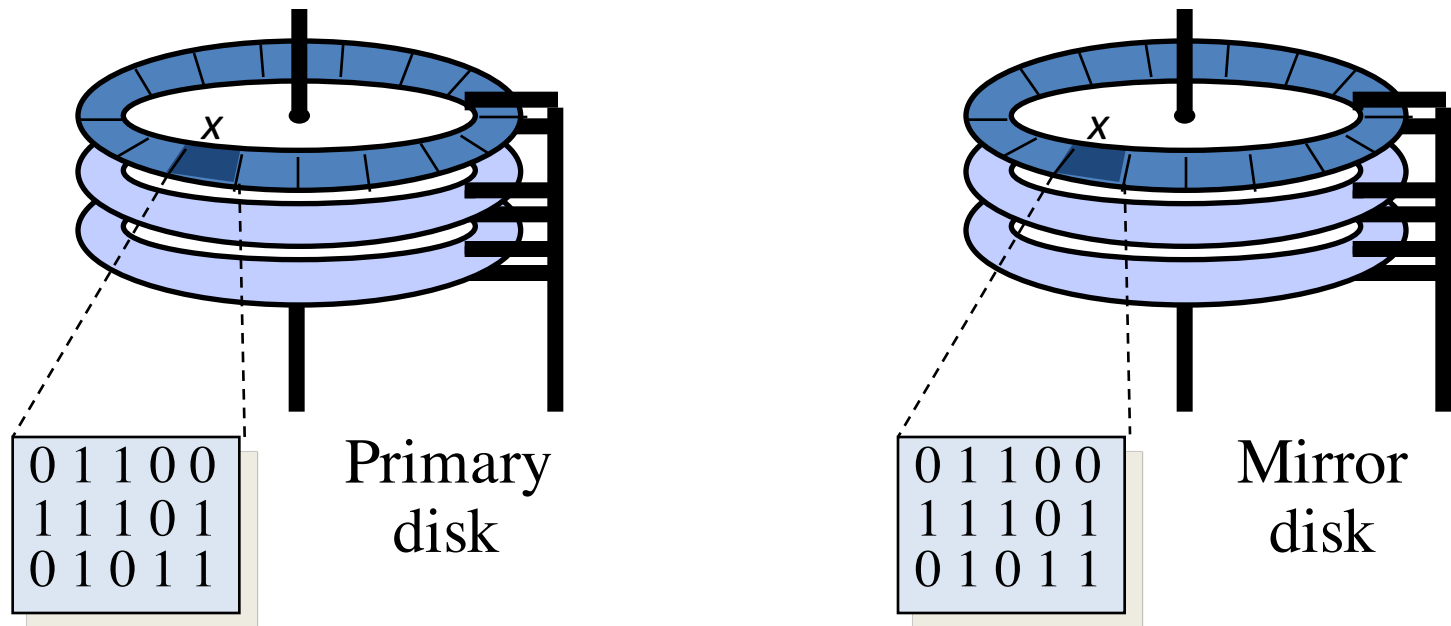
- Blocks broken into sub-blocks that are stored on separate disks
 - similar to memory interleaving
- Provides for higher disk bandwidth through a larger effective block size





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.*

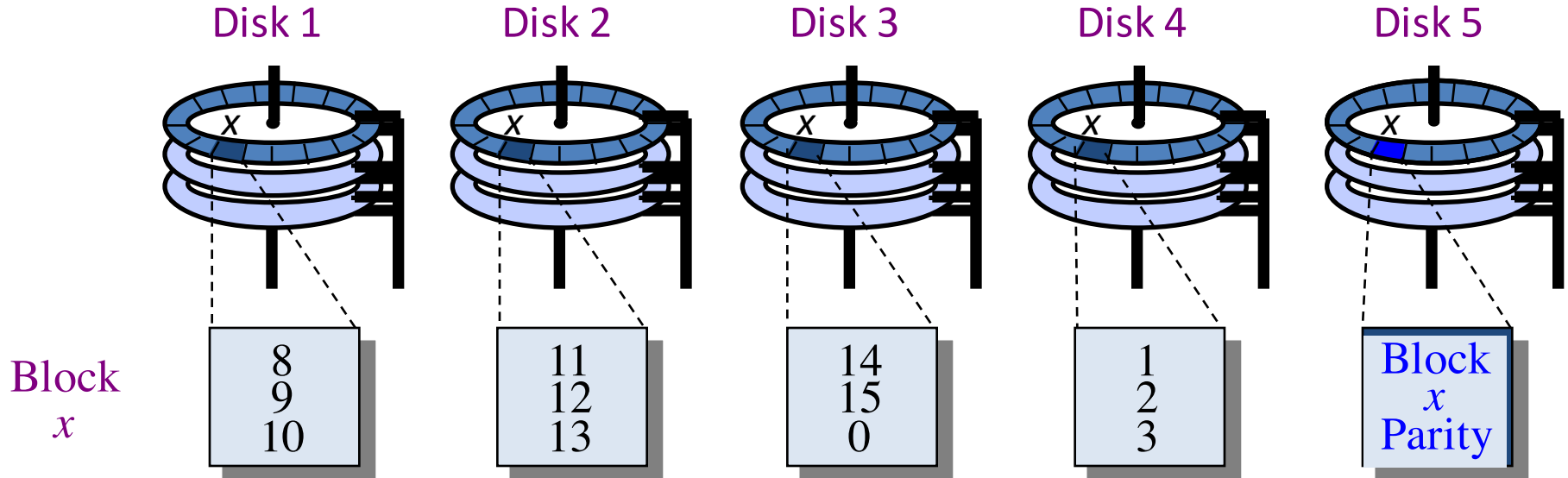


Can lose one disk without losing data



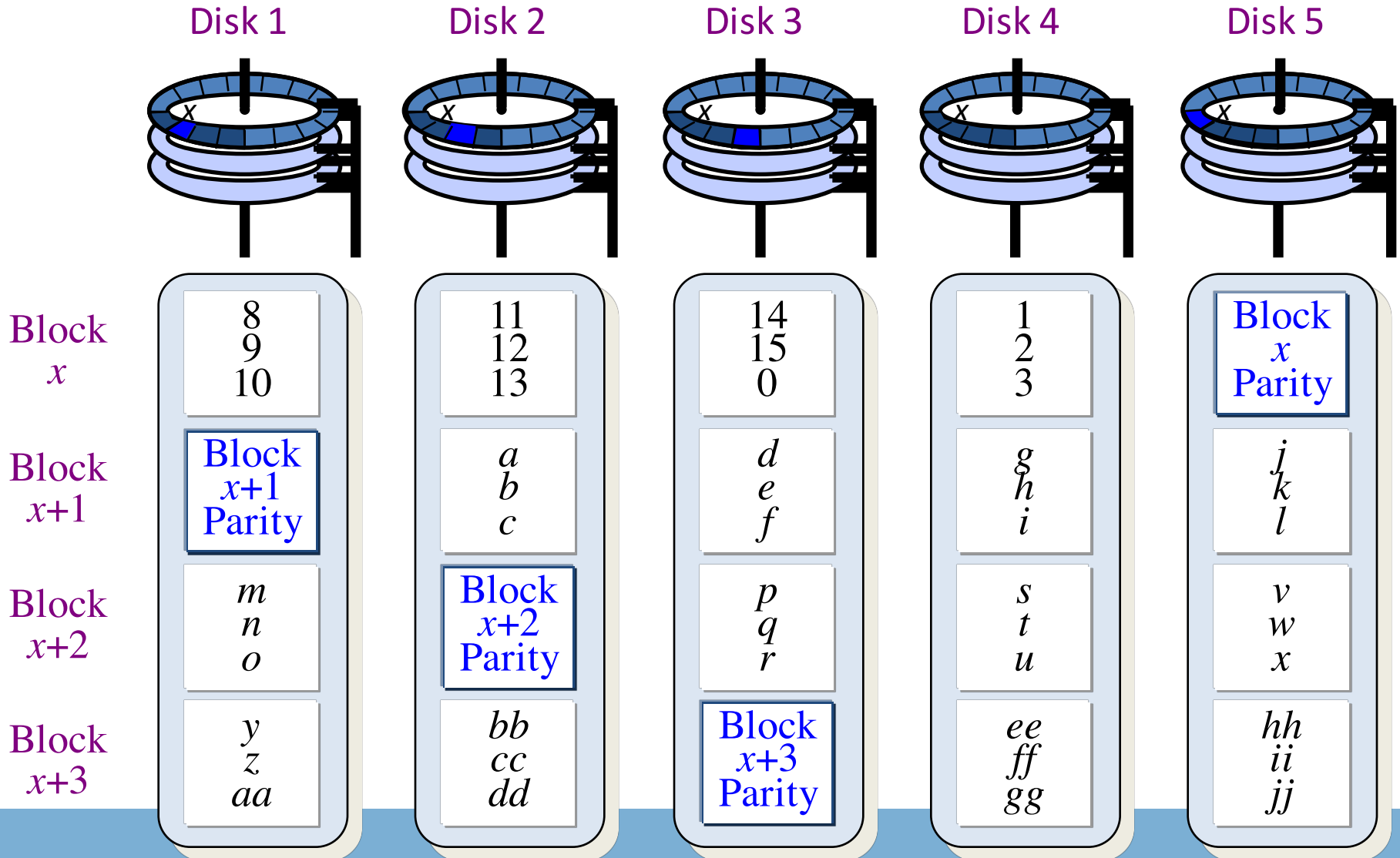
RAID 5: Performance and Redundancy

- 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





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



Summary

- Understand disk performance model
 - Will explore more in Lab 4
- Understand I/O scheduling algorithms
- Understand RAID