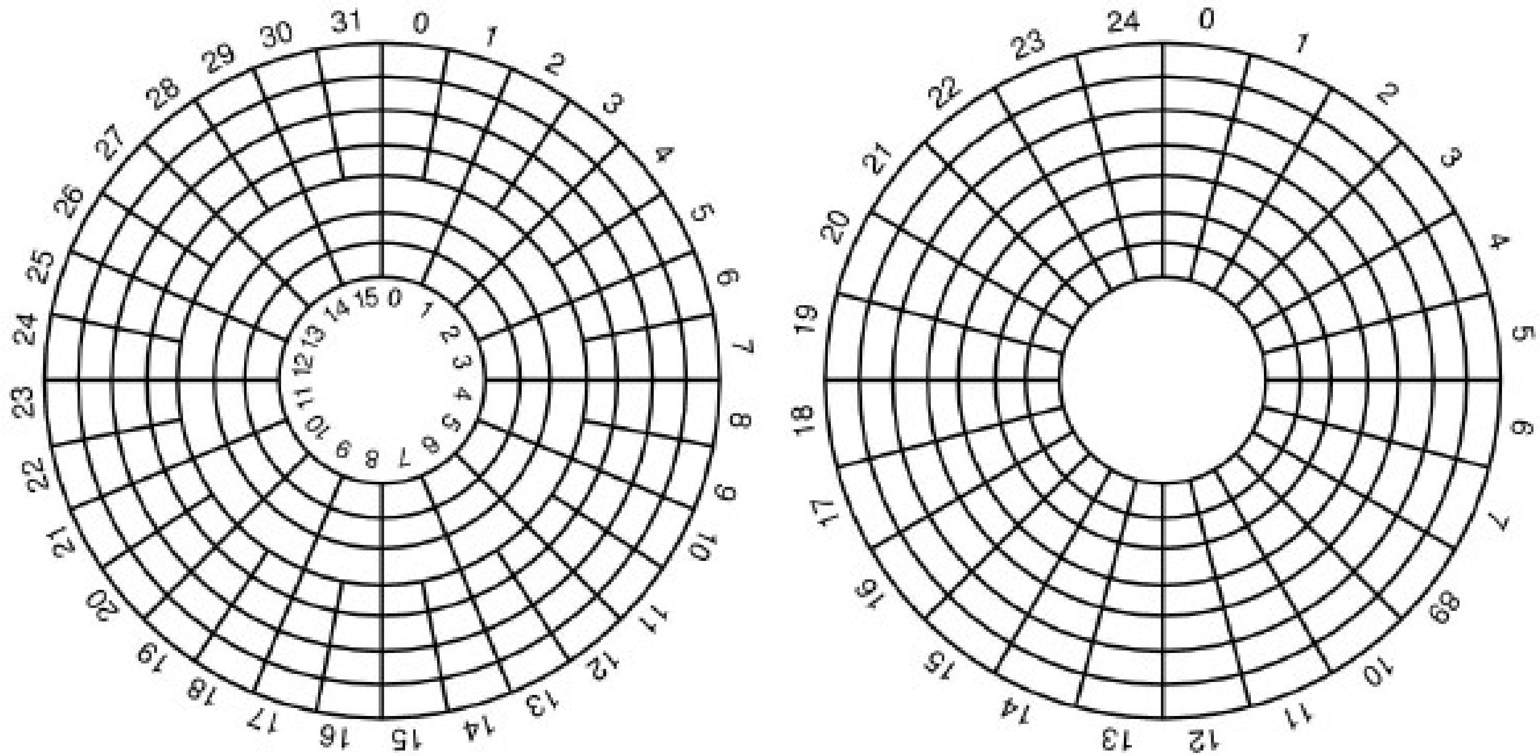
Lec18: Disk I/O Management

Disk Management

- Management and ordering of disk access requests is important:
  - Huge speed gap between memory and disk
  - Disk throughput is extremely sensitive to
    - Request order ⇒ Disk Scheduling
    - Placement of data on the disk ⇒ file system design
  - Disk scheduler must be aware of disk geometry

Disk Geometry



- Physical geometry of a disk with two zones
  - Outer tracks can store more sectors than inner without exceed max information density
- A possible virtual geometry for this disk

Evolution of Disk Hardware

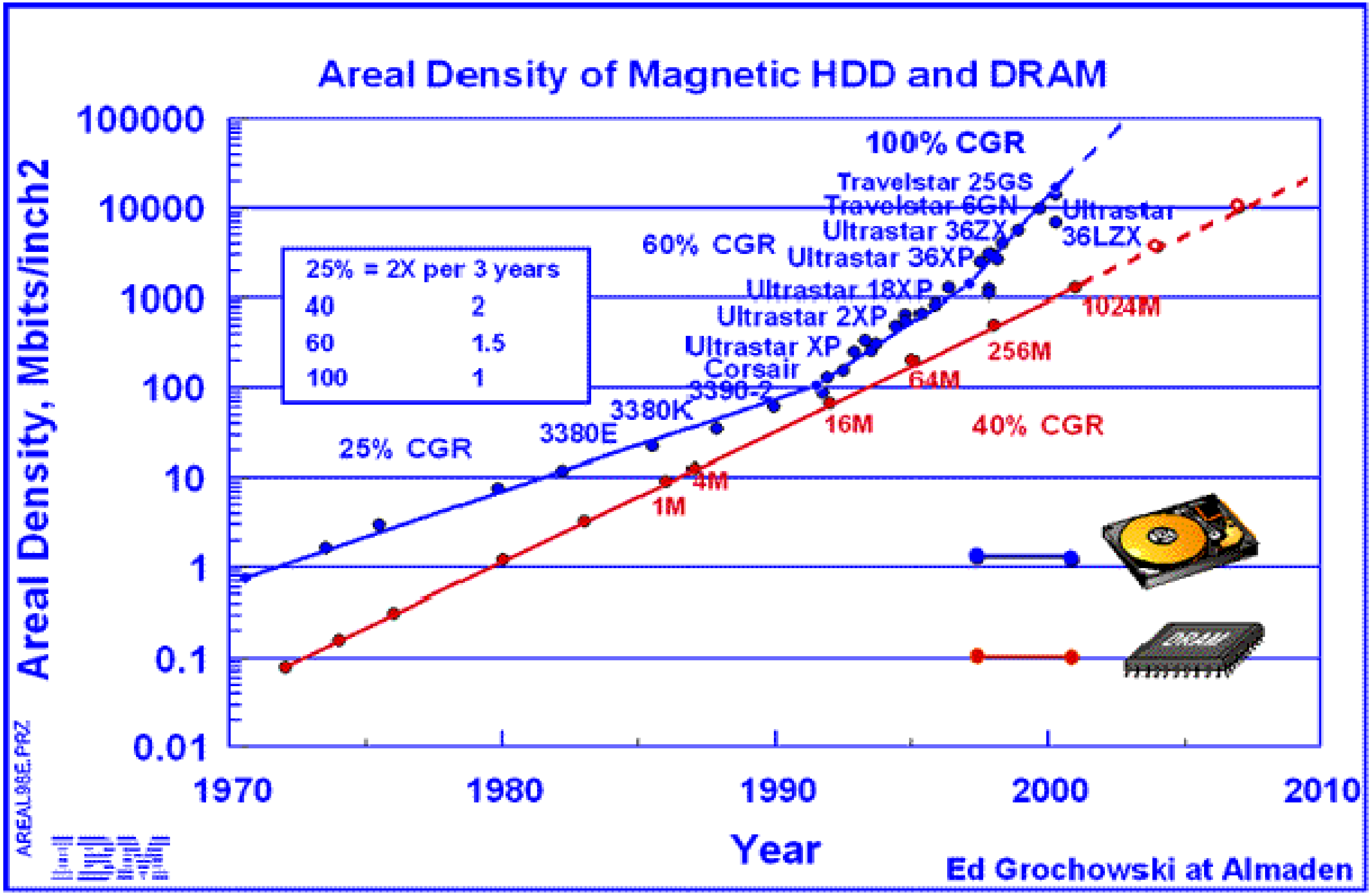
Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

Things to Note

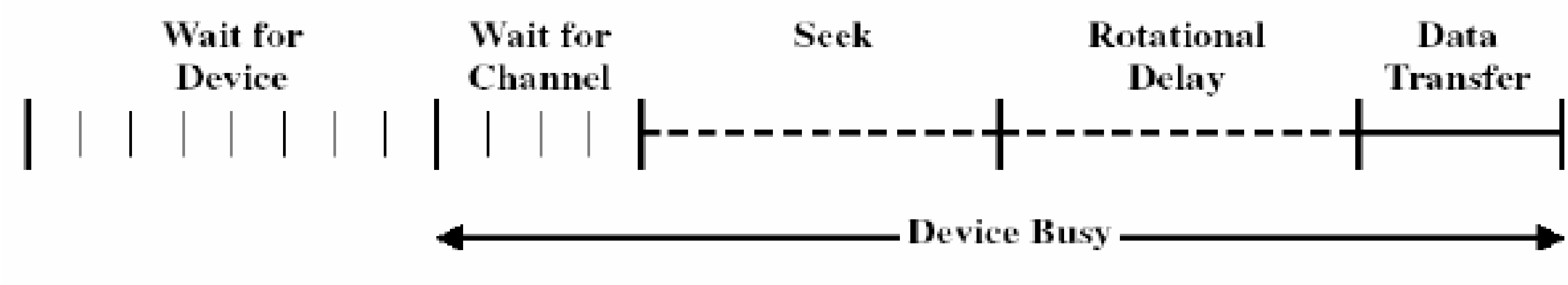
- Average seek time is approx 12 times better
- Rotation time is 24 times faster
- Transfer time is 1300 times faster
  - Most of this gain is due to increase in density
- Represents a gradual engineering improvement

Storage Capacity is 50000 times greater



Disk Performance

- Disk is a moving device ⇒ must be positioned correctly for I/O
- Execution of a disk operation involves
  - Wait time: the process waits to be granted device access
    - Wait for device: time the request spend in wait queue
    - Wait for channel: time until a shared I/O channel is available
  - Access time: time hardware need to position the head
    - Seek time: position the head at the desire track
    - Rotational delay (latency): spin disk to the desired sector
  - Transfer time: sectors to be read/written rotate below head



Estimating Access Time

- *Seek time* T_s: Moving the head to the required track
  - not linear in the number of tracks to traverse:
    - ➜ startup time
    - ➜ settling time
  - Typical average seek time: a few milliseconds
- *Rotational delay:*
  - rotational speed, r, of 5,000 to 10,000rpm
  - At 10,000rpm, one revolution per 6ms $\Rightarrow$ average delay 3ms
- *Transfer time:*

  to transfer b bytes, with N bytes per track: $$T = \frac{b}{rN}$$

Total average access time: $$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

A Timing Comparison

- $T_s = 2$ ms, $r = 10,000$ rpm, 512B sect, 320 sect/track
- Read a file with 2560 sectors (= 1.3MB)
- File stored compactly (8 adjacent tracks):

Read first track	
Average seek	2ms
Rot. delay	3ms
Read 320 sectors	6ms
	11ms

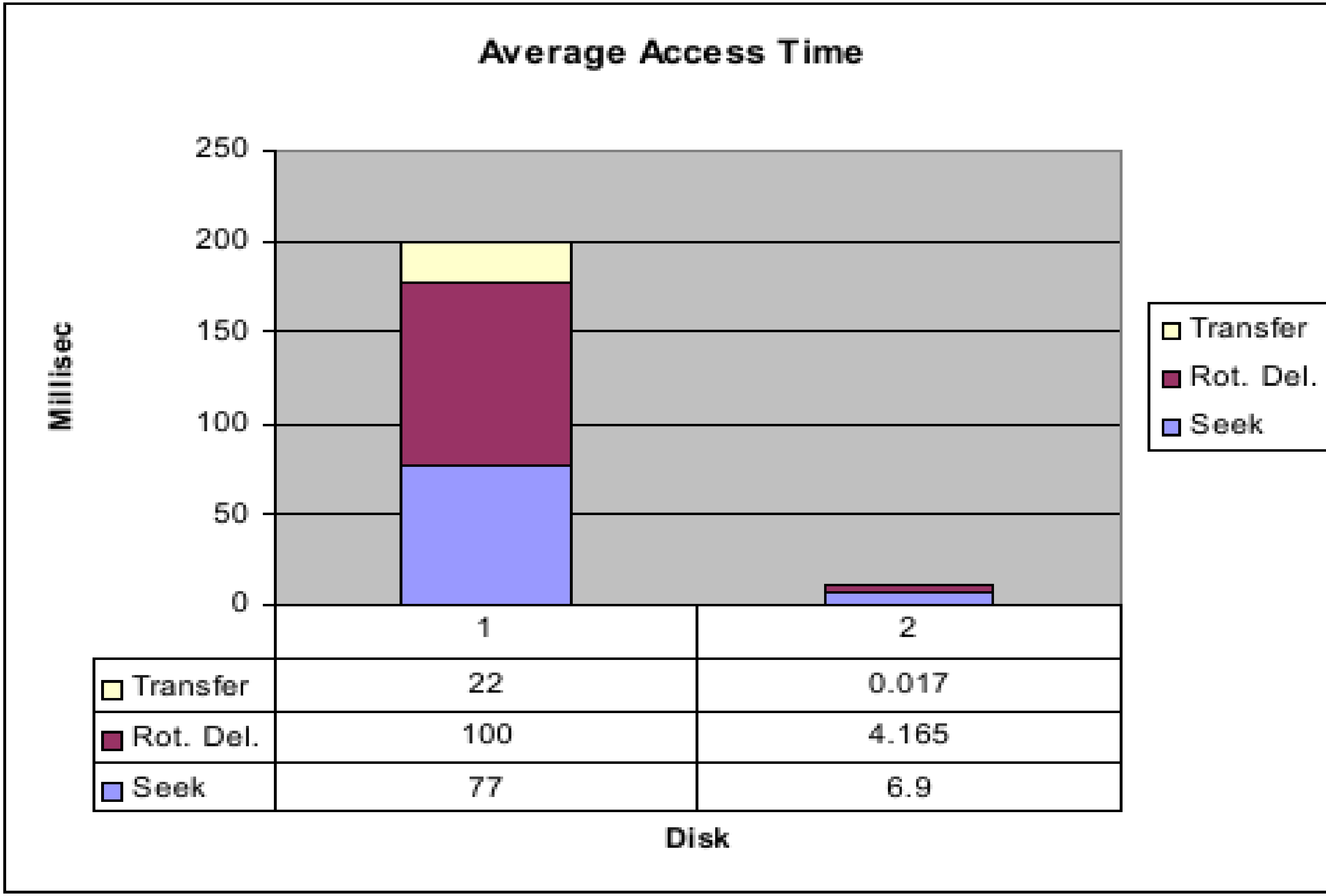
$\Rightarrow$ All sectors: $11 + 7 * 8 = 67\ ms$

- Sectors distributed randomly over the disk:

Read any sector	
Average seek	2ms
Rot. delay	3ms
Read 1 sector	0.01875ms
	5.01875ms

$\Rightarrow$ All: $2560 * 5.01875 = 20,328ms$

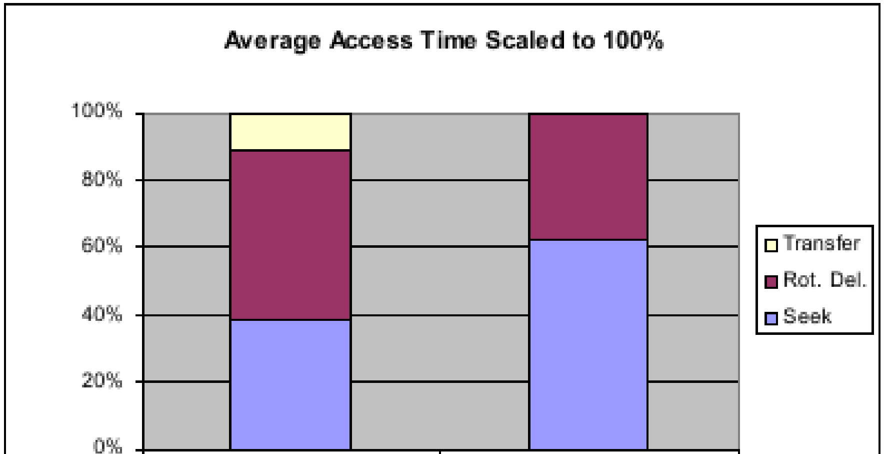
Disk Comparative Performance



	1	2
Transfer	22	0.017
Rot. Del.	100	4.165
Seek	77	6.9

Disk Performance is Entirely Dominated by Seek and Rotational Delays

- Will only get worse as capacity increases much faster than increase in seek time and rotation speed
  - Note it has been easier to spin the disk faster than improve seek time
- Operating System should minimise mechanical delays as much as possible



	1	2
Transfer	22	0.017
Rot. Del.	100	4.165
Seek	77	6.9

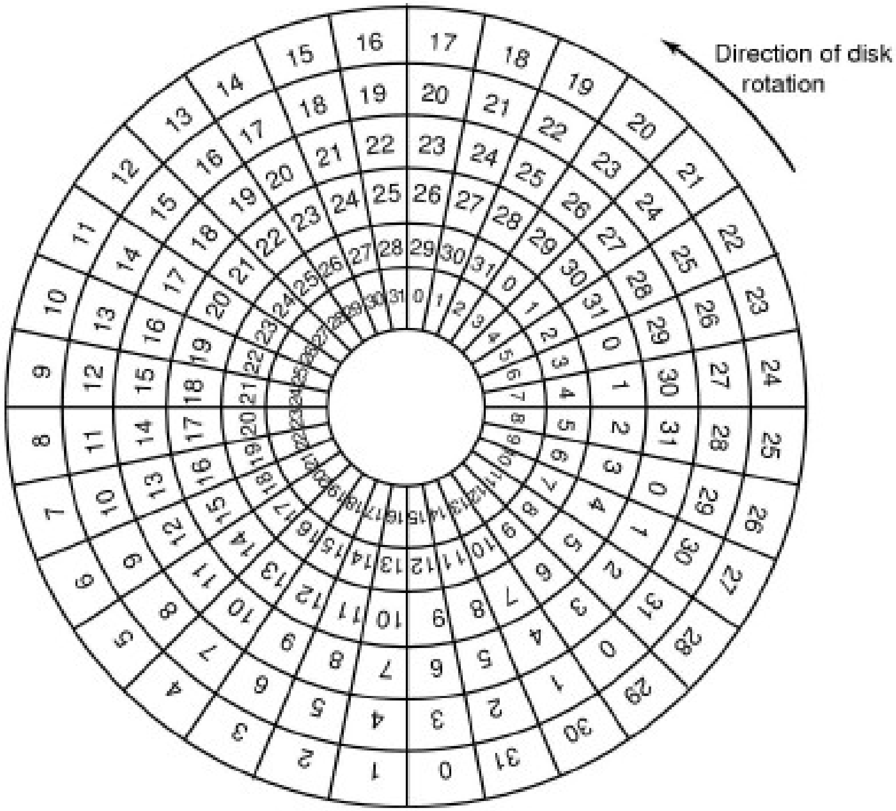
Low-level Disk Formatting

Preamble	Data	ECC

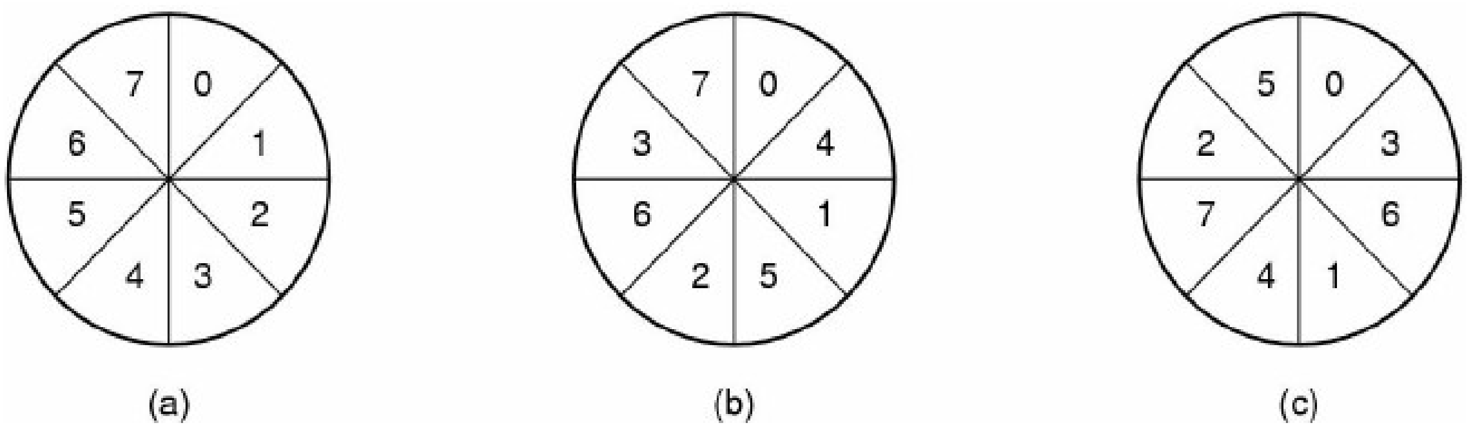
A disk sector

Low-level Disk Formatting

- When reading sequential blocks, the seek time can result in missing block 0 in the next track
- Disk can be formatted using a cylinder skew to avoid this



Low-Level Disk Formatting



- Issue: After reading one sector, the time it takes to transfer the data to the OS and receive the next request results in missing reading the next sector
- To overcome this, we can use interleaving
  - a) No interleaving
  - b) Single interleaving
  - c) Double interleaving

Low-Level Disk Formatting

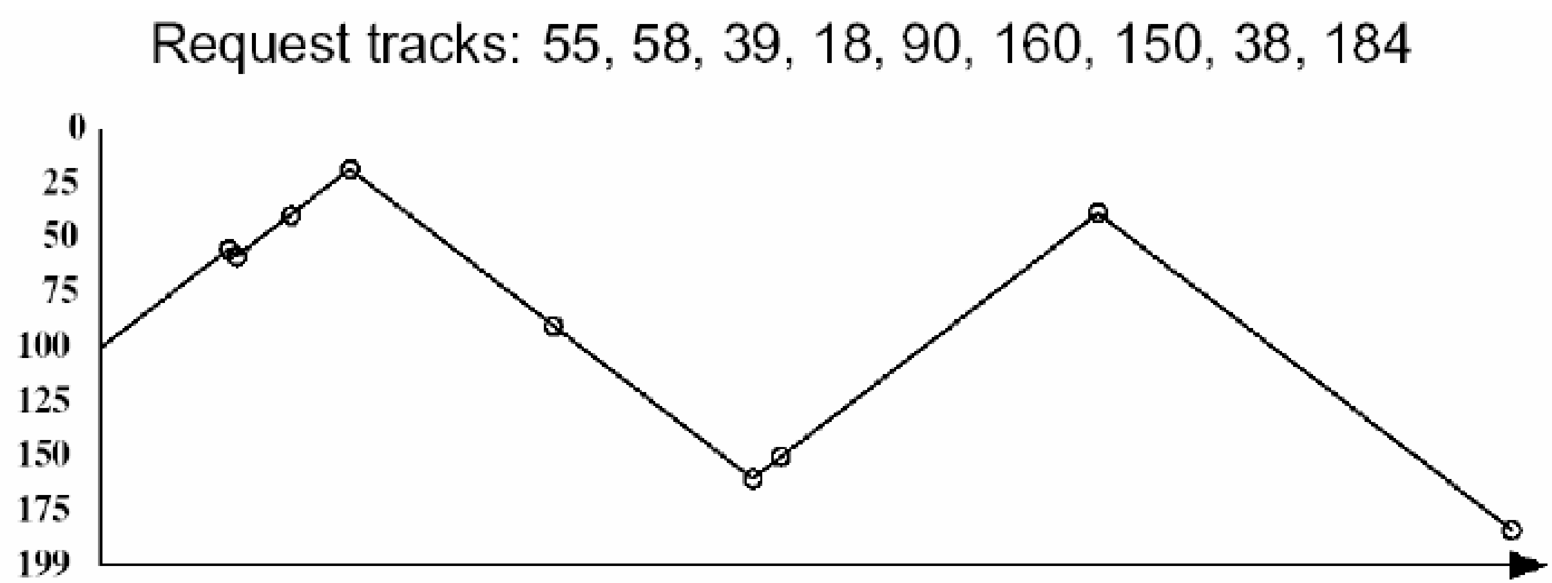
- Modern drives overcome interleaving type issues by simply reading the entire track (or part thereof) into the on-disk controller and caching it.

Disk Arm Scheduling Algorithms

- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
  - For a single disk, there will be a number of I/O requests
  - Processing them in random order leads to worst possible performance

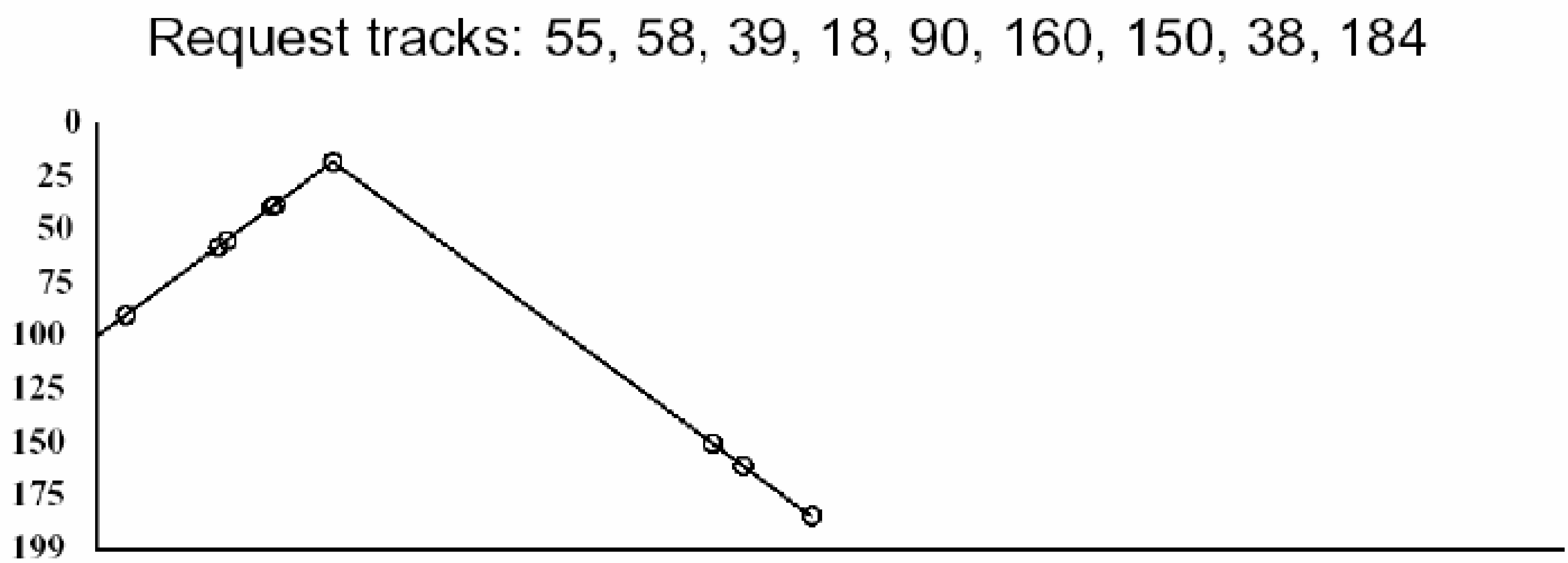
First-in, First-out (FIFO)

- Process requests as they come
- Fair (no starvation)
- Good for a few processes with clustered requests
- Deteriorates to random if there are many processes



Shortest Seek Time First

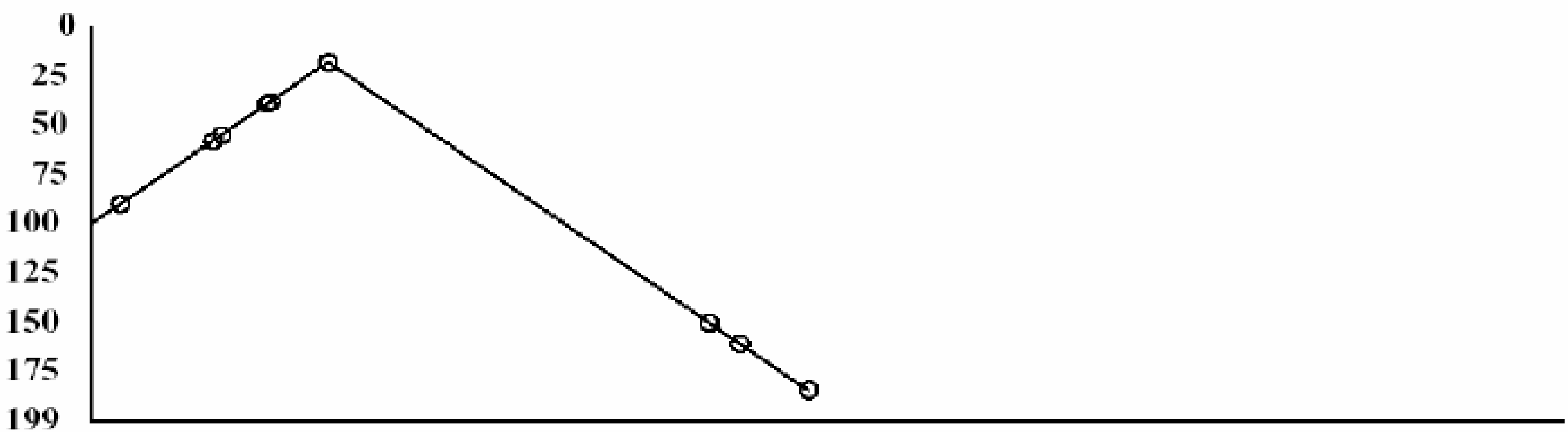
- Select request that minimises the seek time
- Generally performs much better than FIFO
- May lead to starvation



Elevator Algorithm (SCAN)

- Move head in one direction
  - Services requests in track order until it reaches the last track then reverses direction
- Better than FIFO, usually worse than SSTF
- Avoids starvation
- Makes poor use of sequential reads (on down-scan)

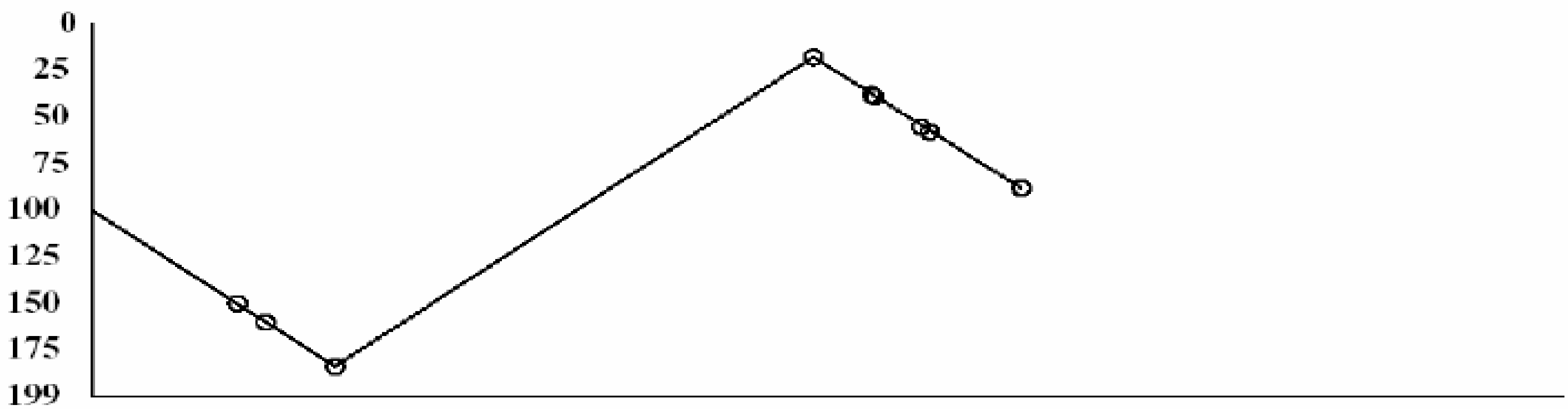
Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



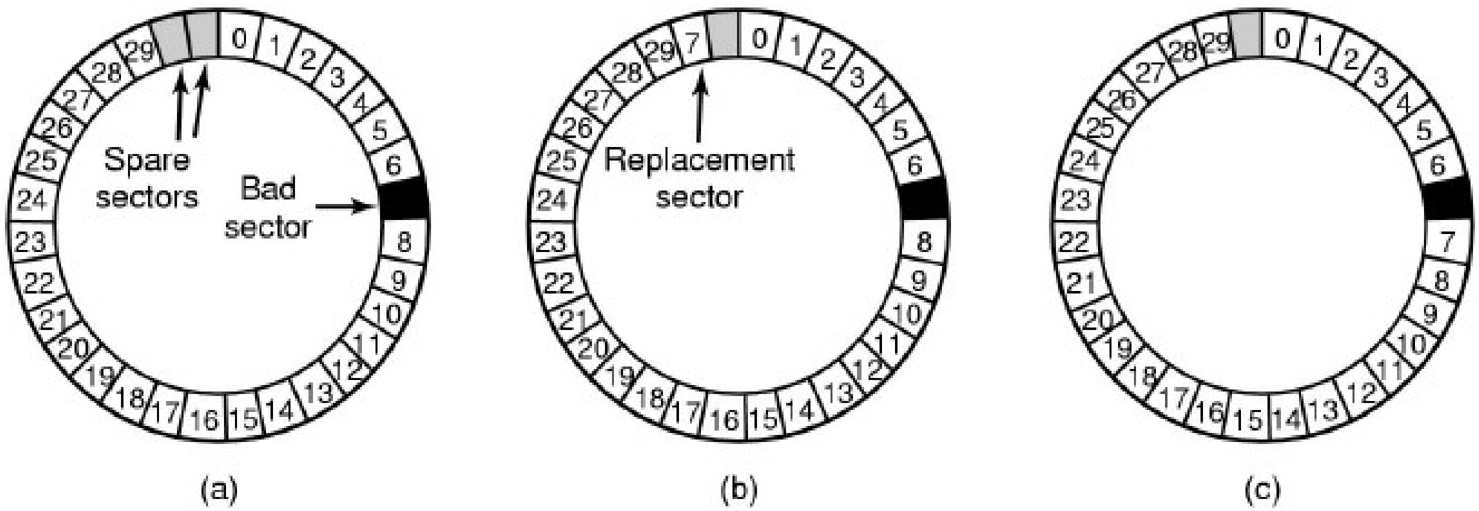
Modified Elevator (Circular SCAN, C-SCAN)

- Like elevator, but reads sectors in only one direction
  - When reaching last track, go back to first track non-stop
- Better locality on sequential reads
- Better use of read ahead cache on controller
- Reduces max delay to read a particular sector

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



Error Handling



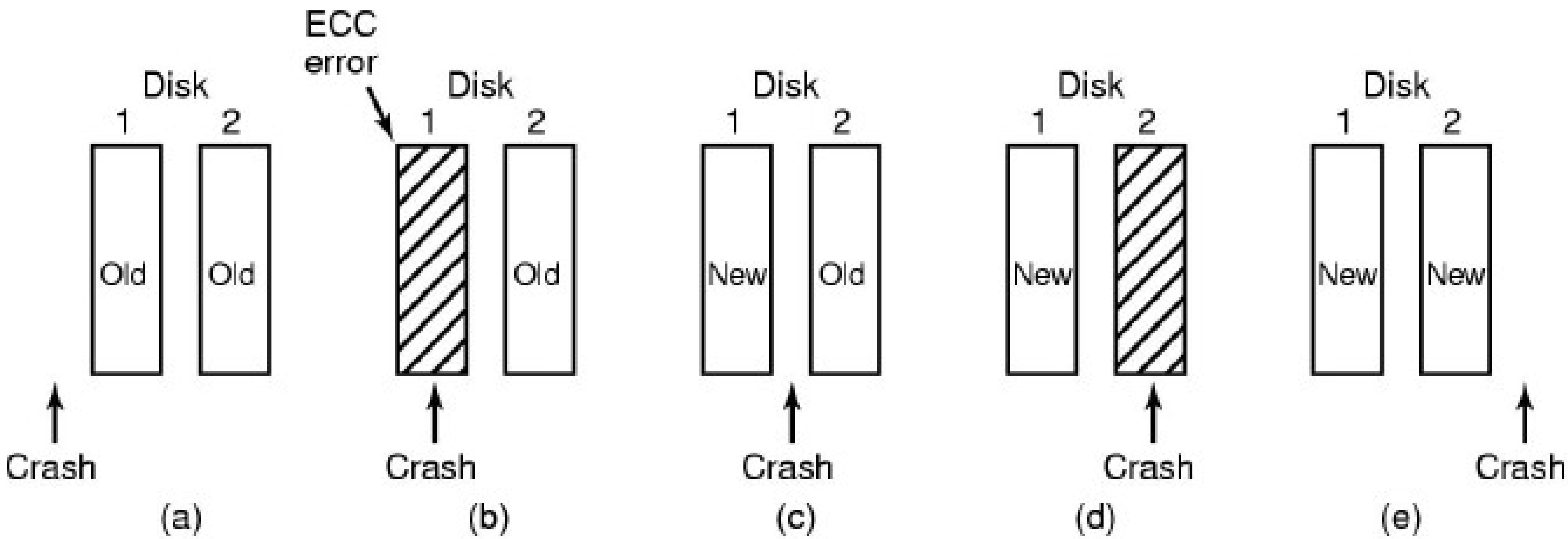
a) A disk track with a bad sector

b) Substituting a spare for the bad sector

c) Shifting all the sectors to bypass the bad one

- Bad blocks are usually handled transparently by the on-disk controller

Implementing Stable Storage



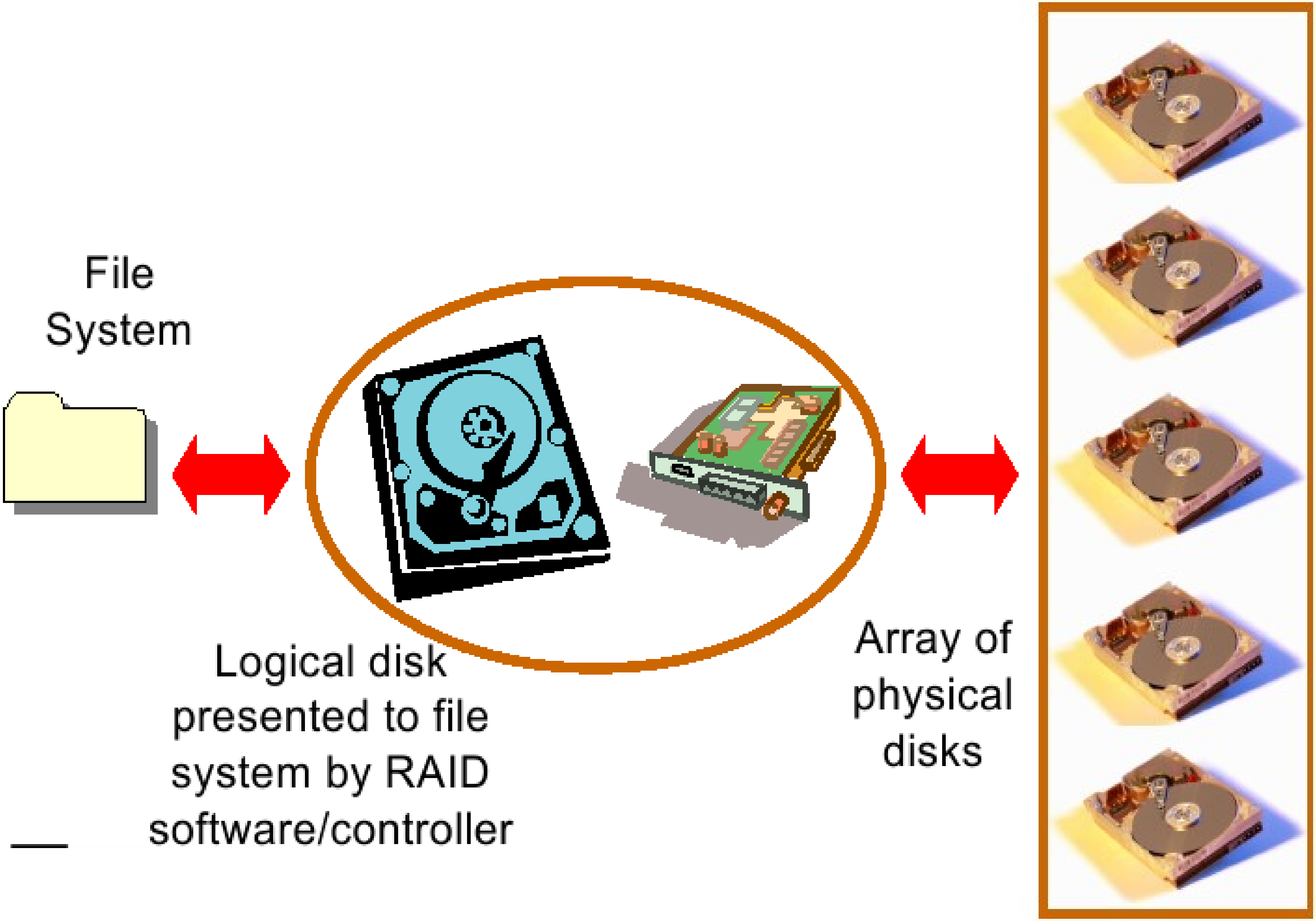
- Use two disks to implement stable storage
  - Problem is when a write (update) corrupts old version, without completing write of new version
  - Solution: Write to one disk first, then write to second after completion of first

RAID

- Redundant Array of Inexpensive Disks
  - Industry tends to use “Independent Disks” ☺
- Idea:
  - Use multiple disks to parallelize Disk I/O for better performance
  - Use multiple redundant disks for better availability
- Alternative to a Single Large Expensive Disk (SLED)

RAID Level

- Various configurations of multiple disks are termed a RAID Level
  - Note the Level, does not necessarily imply that one configuration is above or below another.
- We will look at RAID Levels 0 to 5
- All instances of RAID present a single logical disk to the file system.


File System
Logical disk presented to file system by RAID software/controller
Array of physical disks

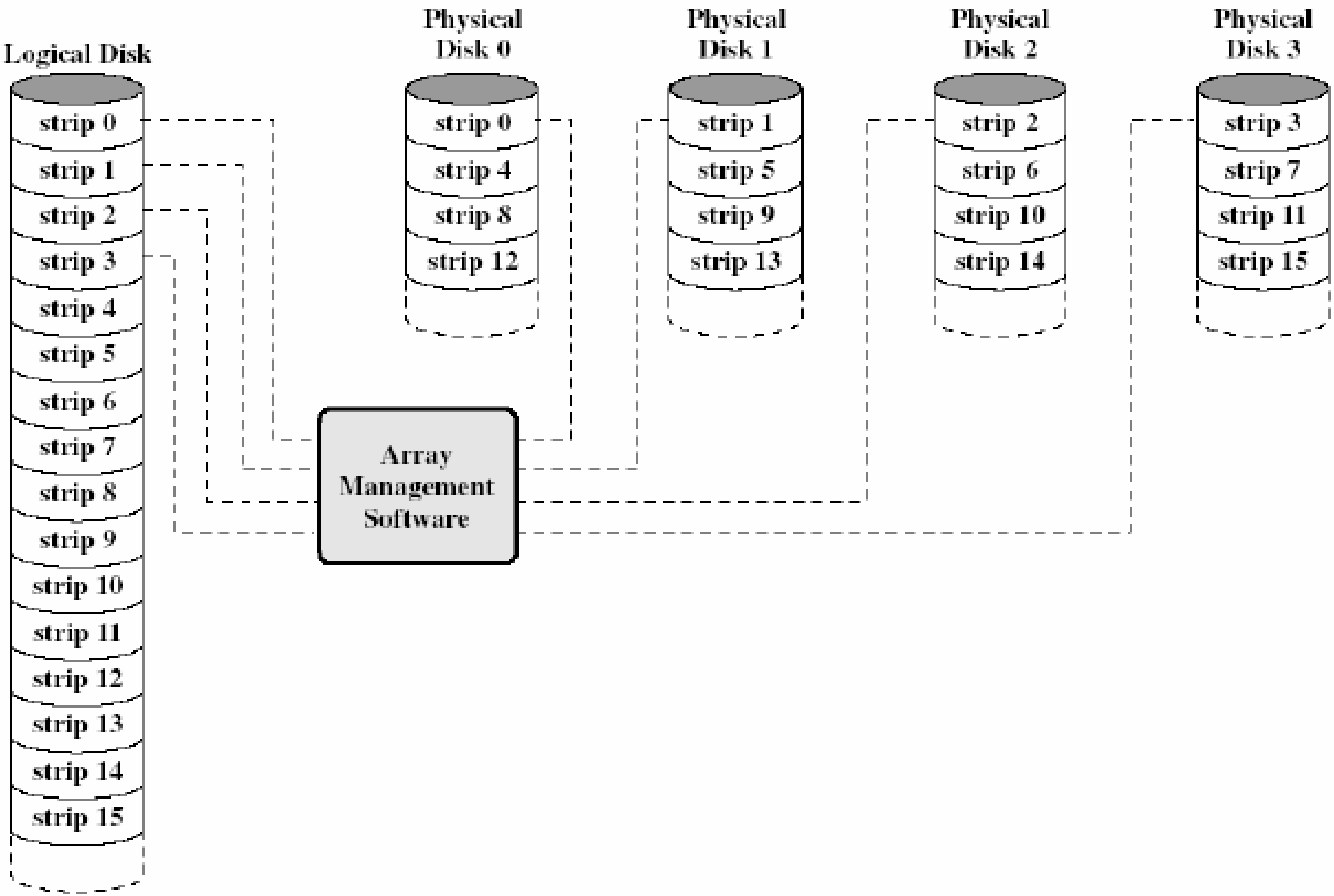


Figure 11.10 Data Mapping for a RAID Level 0 Array [MASS97]

RAID 0

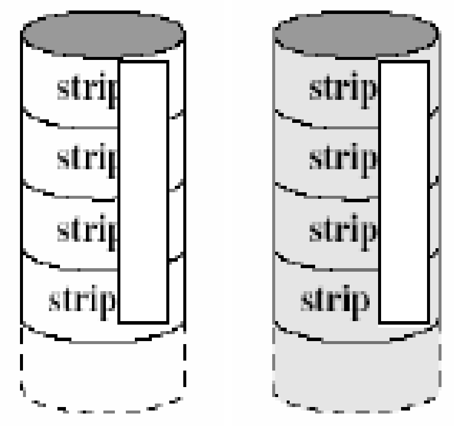
- Logical Disk divided into strip(e)s
  - Stripe = a fixed number of sectors
  - First strip written to disk 0
  - Consecutive strips written to different disk in the array in round-robin fashion
- Splits I/O workload across several disks
  - Best with many independent request streams
    - Avoids hotspots on a single disk
- Increases bandwidth available to/from the logical disk.

RAID 0

- Not really true RAID
  - No redundancy
- RAID 0 is less reliable than a SLED
  - Example: Assume MTBF of 10000 hours
  - MTBF of the array is MTBF divided by the number of disks
    - A 4 disk array would have an MTBF of 2500 hours

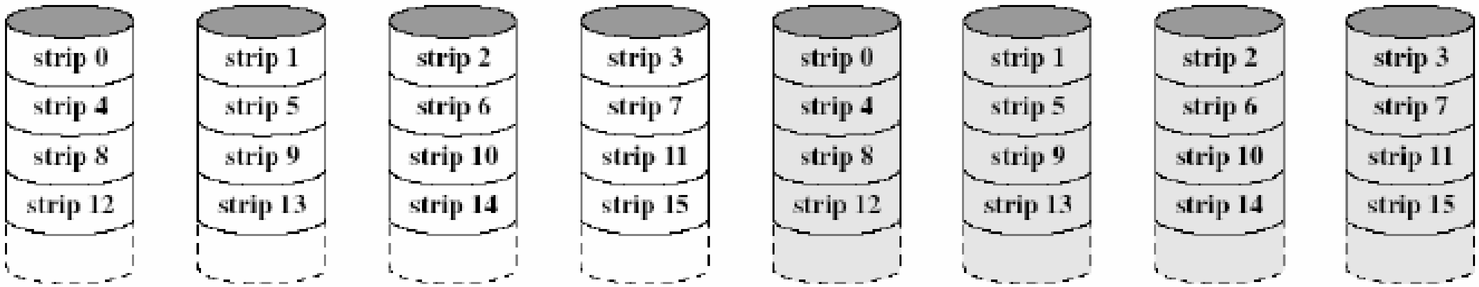
RAID 1

- Each strip is written to two disks
  - Also termed Mirroring (true RAID 1)
- Provides redundancy
  - If disk fails, we can use the copy
- Read performance can double
  - To fetch some blocks, we send half the requests to one disk set, and the other half to the other
- Write performance stays the same
  - A logical write results in two parallel writes to real disks



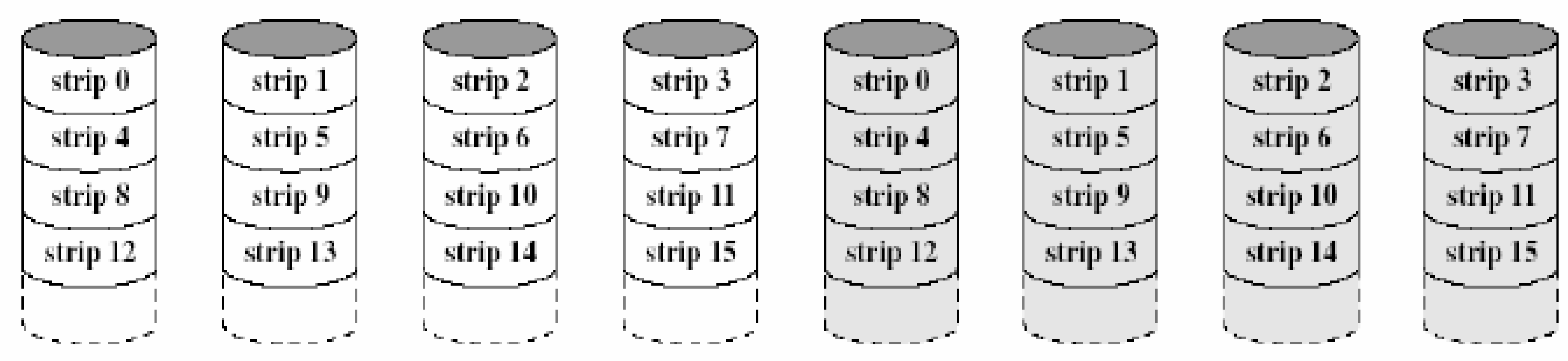
RAID 0+1,1+0, 01, 10

- With striping, sometimes termed RAID 0+1,1+0, 01, 10
- Diagram RAID 0+1
  - Two striped sets (RAID 0)
  - Mirror the two sets (RAID 1)
- Alternative RAID 1+0



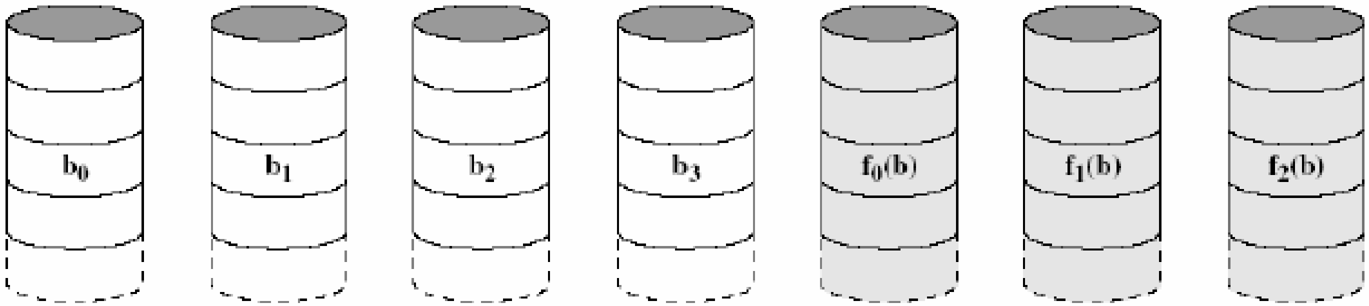
RAID 1

- However
  - RAID 1 requires twice as many disks



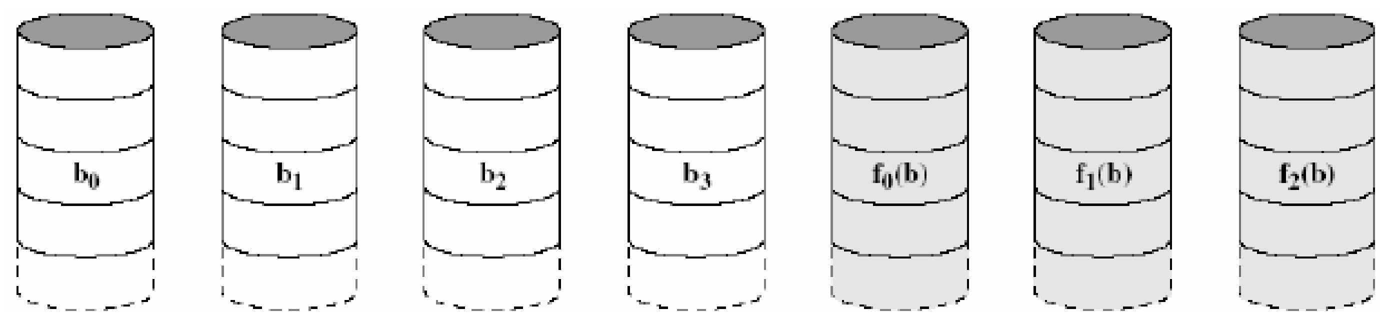
RAID 2

- Example: split data into 4-bit nibbles
- Write each bit to a separate disk
  - Use synchronised spindles to ensure each bit is available at the same time
- Additionally, write 3 Hamming code (ECC) bits to 3 extra disks
  - Hamming code can correct a single bit error



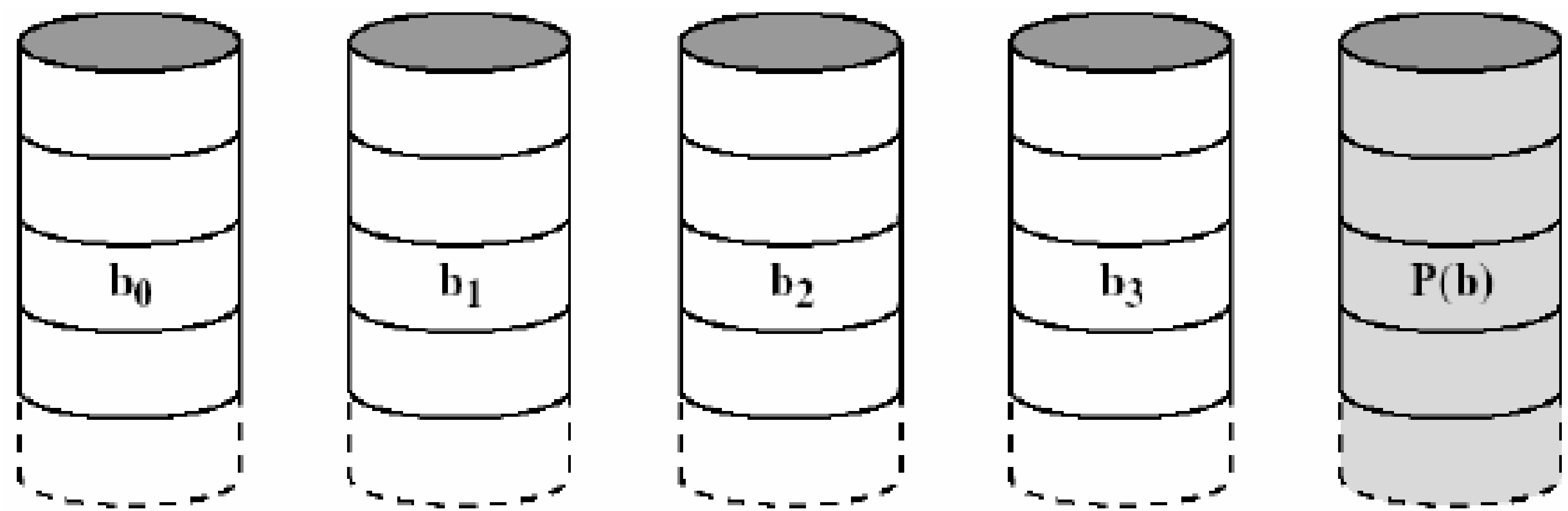
RAID 2

- Makes more sense with more drives
  - 38 drives (32-bit words, with 6-bit ECC)
  - Still 19% storage overhead
- Disadvantage – needs synchronised spindles
- Not used



RAID 3

- Like RAID 2, but instead of ECC, use a single parity bit.
- Can only detect a single error, not correct it
  - Unless we know which bit is wrong



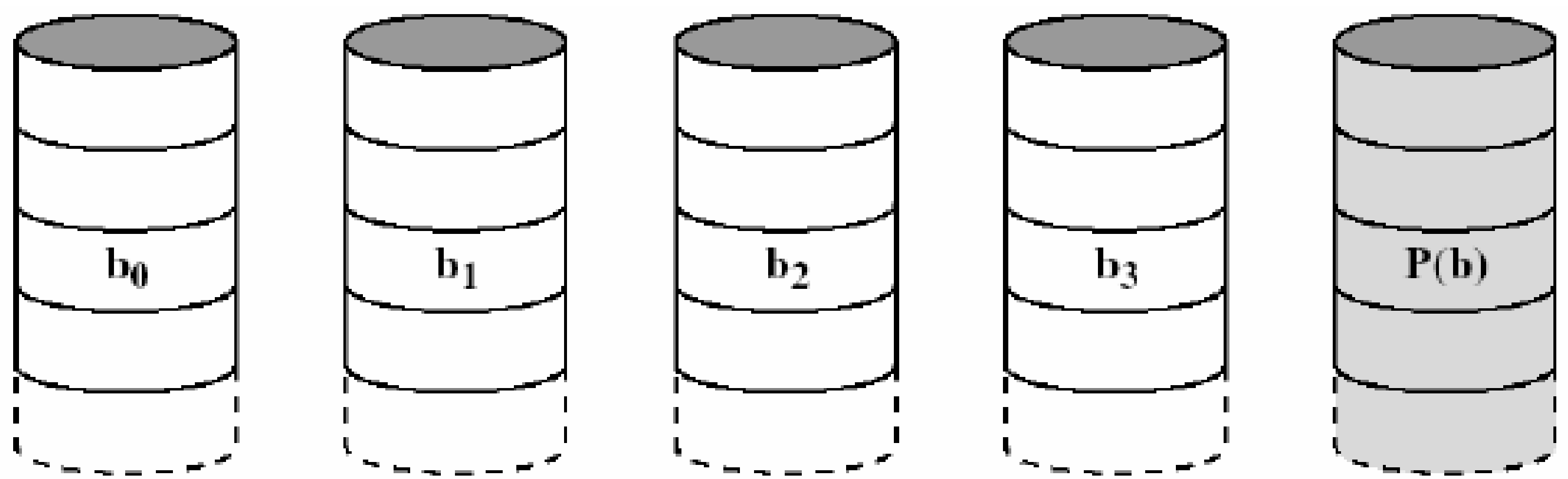
Quick Look At Parity

Disk 1	Disk 2	Disk 3	Disk 4	Parity
1	0	1	0	$\boxed{0}$

What is the lost bit?

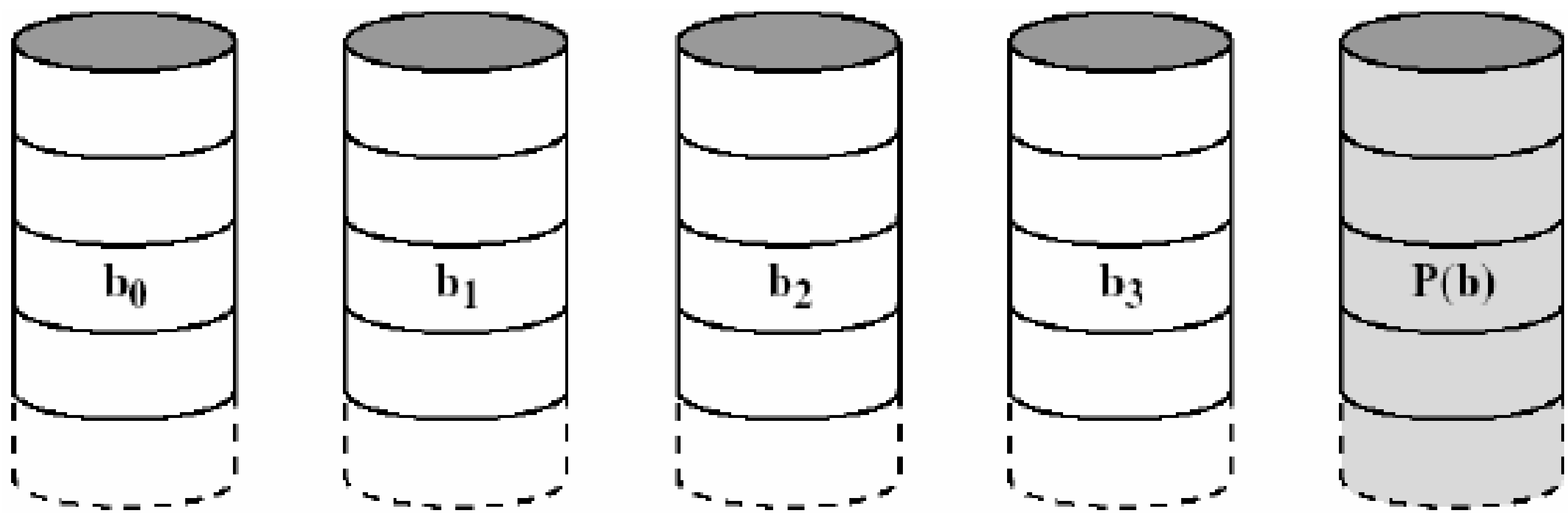
RAID 3

- Fortunately, if a disk fails, we know which bit is "wrong" and can use the parity bit to recover it
- Advantage:
  - Only need a single extra disk to implement RAID 3
- Can handle failure of complete disk



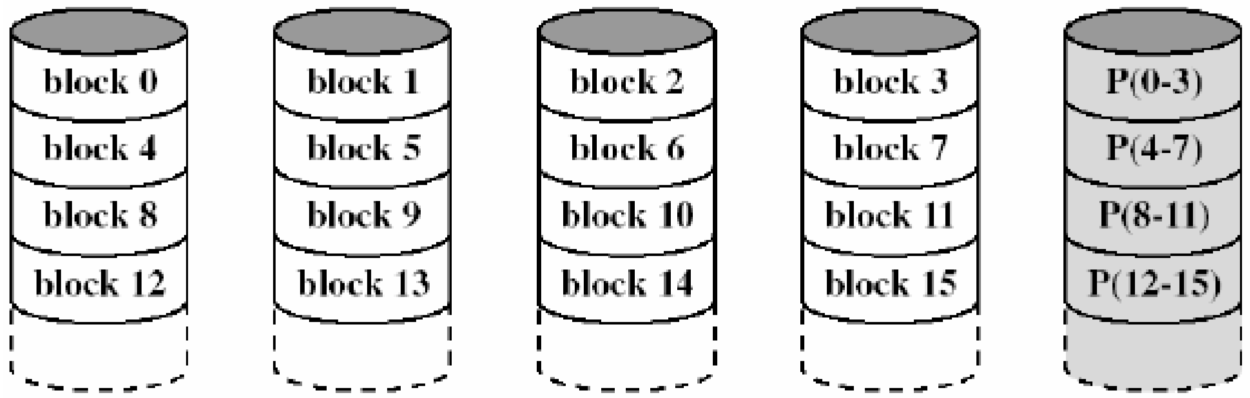
RAID 3

- Disadvantage:
  - Synchronised spindles
  - Fast for reading contiguous data, but does not improve performance for independent small requests
    - Each drive seeks together



RAID 4

- Parity computed on a block basis
  - Block 0-3 XOR'd to gether to generate a parity block
    - P block(x) = Block0(x) ⊗ Block1(x) ⊗ Block2(x) ⊗ Block3(x)
  - Parity stored on an extra disk
- Only needs one extra disk to implement
- Can handle failure of a single disk



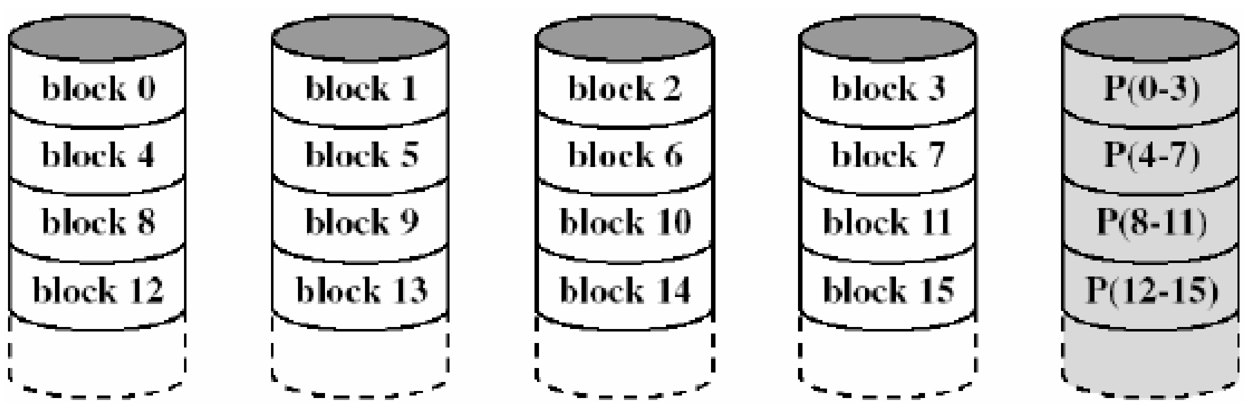
Examining the first byte in each block

	Byte 0
Block 0	011010011
Block 1	111111010
Block 2	010000001
Block 3	001010100
Parity	111111100

What is the lost byte?

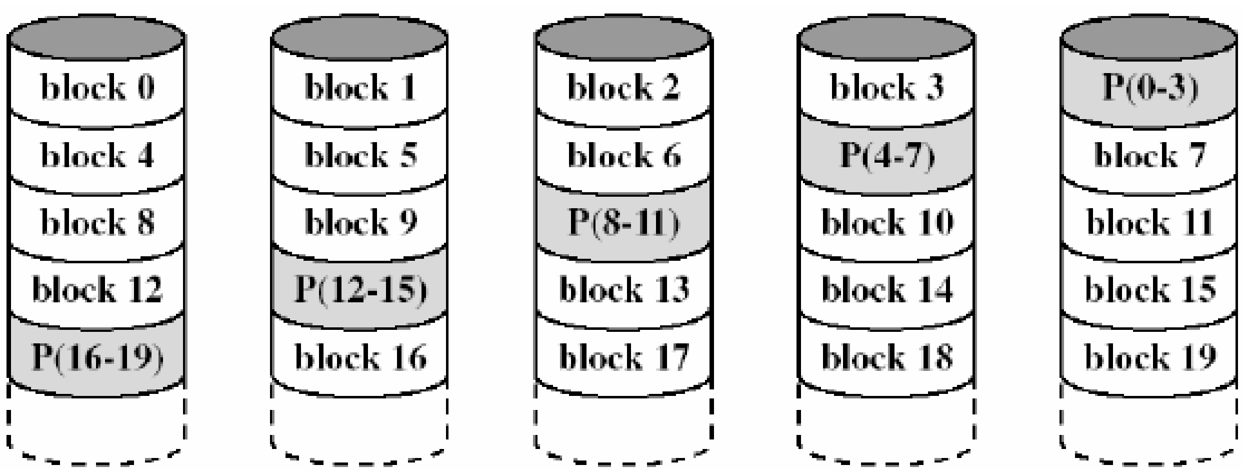
RAID 4

- Does not require synchronised spindles
- Can parallelised many independent request
- Small updates are a problem
  - Requires two reads (old block + parity) and two writes (new block + parity) to update a disk block
  - Parity disk may become a bottleneck



RAID 5

- Like RAID 4, except we distribute the parity on all disks
- Avoids parity disk updates becoming a bottleneck
- Update performance still less than a single disk
- Reconstruction after failure is tricky



Summary

- RAID 0 provides performance improvements, but no availability improvement
- RAID 1 (01,10) provides performance and availability improvements but expensive to implement (double the number of disks)
- RAID 5 is cheap (single extra disk), but has poor write update performance
- Others (2 & 3) are not used

HP AutoRAID

- Active data used RAID 1
  - Good read and write performance
- Inactive data uses RAID 5
  - Rarely accessed, RAID 5 provides low storage overheads
- Adaptive Storage
  - Empty disk uses entirely RAID 1, as disk fills, data incrementally converted to RAID 5 to increase available capacity
  - Data updates convert data back to RAID 1

HP AutoRAID

- On-line array expansion
  - New disks can be added and system rebalances
  - New Disks can be an arbitrary size
- Active Hot Spare
  - The hot spare is used for mirroring until needed.

HP AutoRAID

- If you interested in the details see John Wilkes, Richard Golding, Carl Staelin and Tim Sullivan. "The HP AutoRAID hierarchical storage system", ACM Trans. Comput. Syst., Vol 14(1), 1996