Disk Scheduling Algorithms

(Chapter 41. Scheduling in Secondary Storage Systems)

Kanlue Zhang

Agenda

- Secondary Storage System
- Disk Organization
- Performance Parameters
- Disk Scheduling Algorithms
- RAID Motivation
- Summary

Secondary Storage Systems

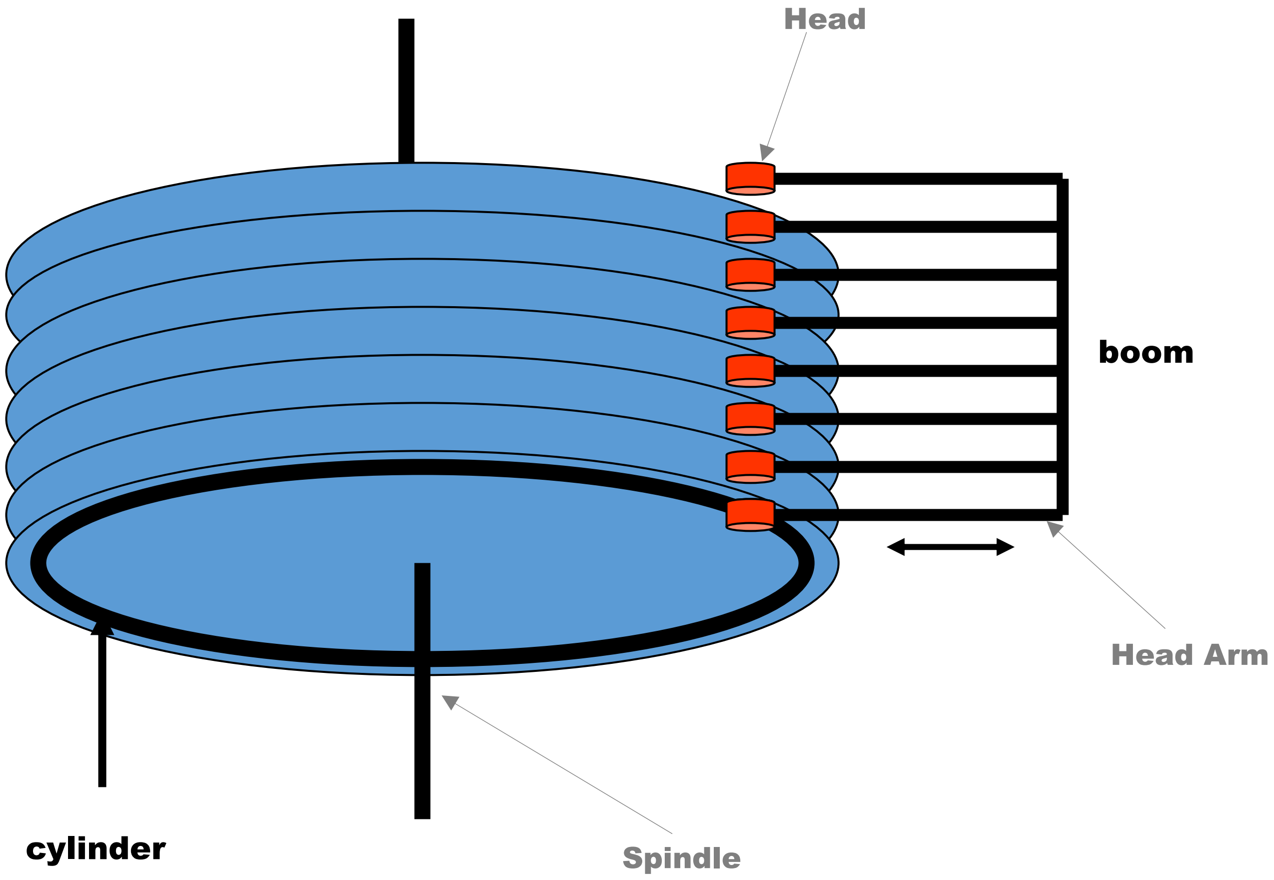
The memory hierarchy in a computer system consists of one to three levels of CPU caches:

- Caches for file and database systems in main memory, the cache associated with a disk array controller;
- An onboard disk cache, disks;
- Automated tape libraries, which are themselves cached.

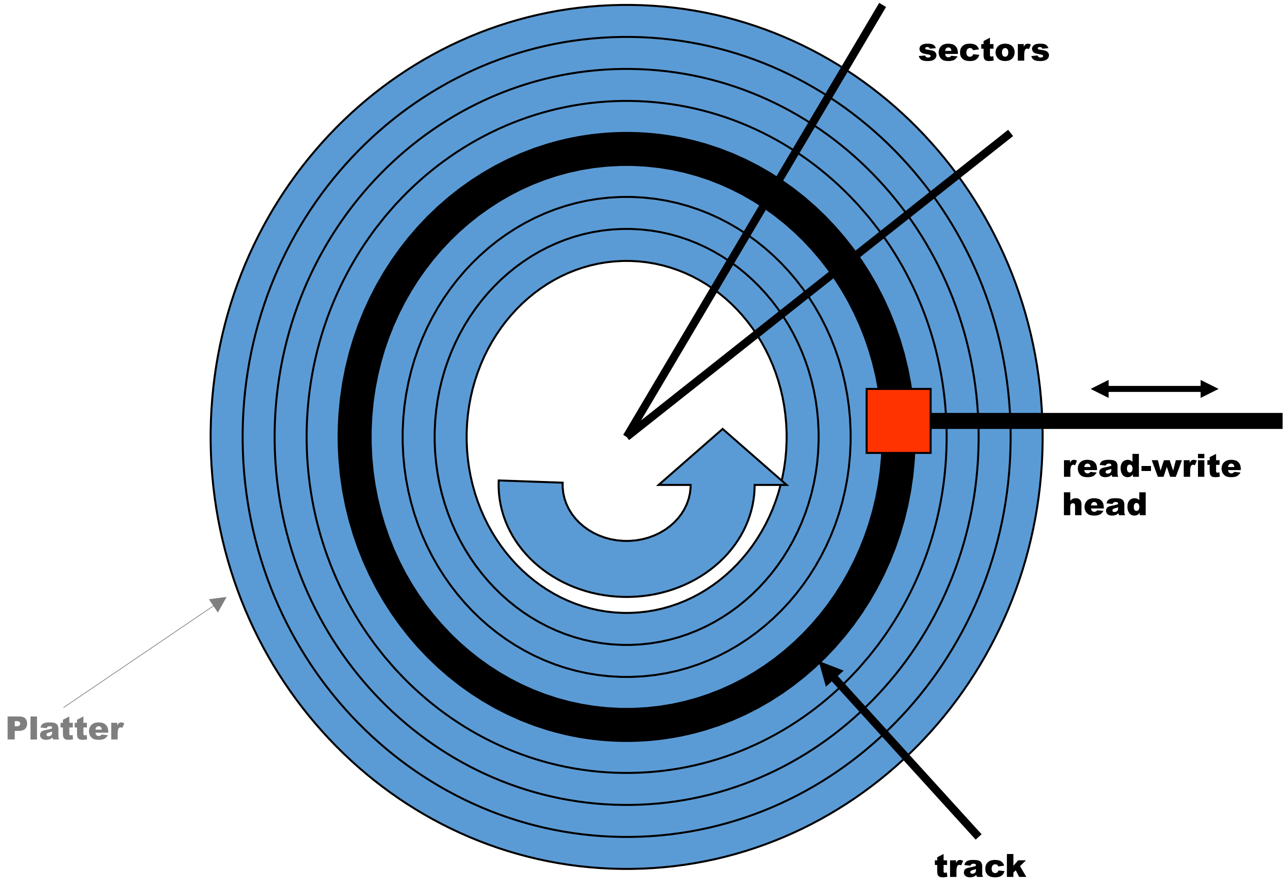
We are concerned here with accesses to database and file systems, which may be satisfied by the buffer in main memory, but in the case of a cache miss, an I/O request is generated for a data block residing on disk.

(This I/O request is intercepted by the disk array controller, which checks its cache in order to satisfy the I/O request. In case of a cache miss an I/O request is issued to the disk and the disk controller checks its cache to satisfy the request. A disk access is initiated if we had misses at all levels.)

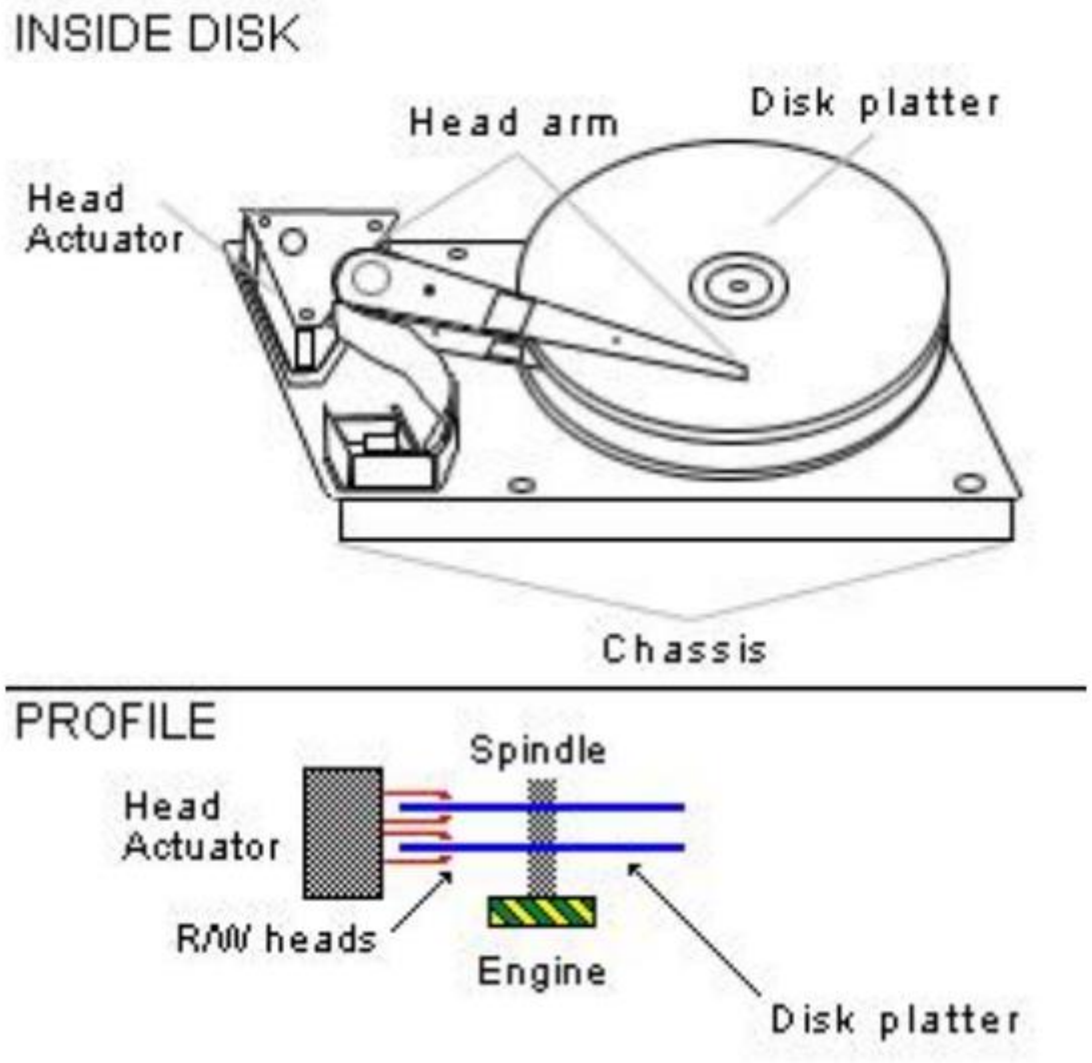
Disk Organization



Disk Organization



Disk Organization

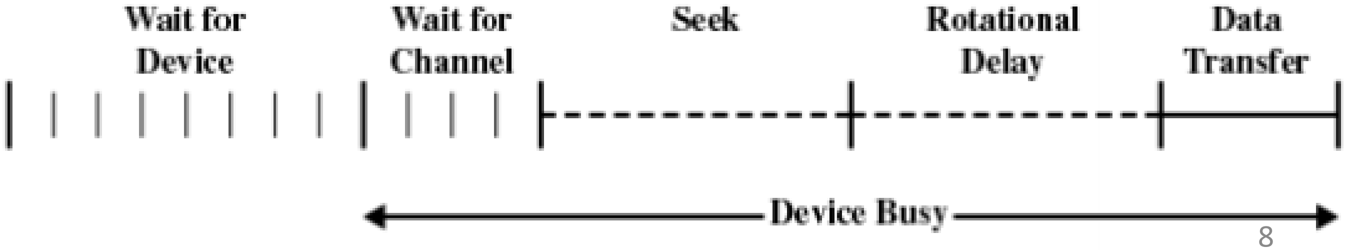


Disk Organization

- When the disk drive is operating, the disk is rotating at constant speed
- To read or write, the disk head must be positioned on the desired track and at the beginning of the desired sector
- Once the head is in position, the read or write operation is then performed as the sector moves under the head

Performance Parameters

- **Seek time** is the time it takes to position the head on the desired track
- **Rotational delay** or **rotational latency** is the additional time it takes for the beginning of the sector to reach the head once the head is in position
- **Transfer time** is the time for the sector to pass under the head



Performance Parameters

- Access time = seek time + rotational latency + transfer time
- Efficiency of a sequence of disk accesses strongly depends on the order of the requests
- Adjacent requests on the same track avoid additional seek and rotational latency times
- Loading a file as a unit is efficient when the file has been stored on consecutive sectors on the same track/cylinder of the disk

Performance

- Seek time is the reason for differences in performance
  - For a single disk there will be a number of I/O requests
  - If requests are selected randomly, get poor performance

- Disk Scheduling Algorithms are used to reduce the total seek time of any request.

In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down.

Disk Scheduling

- I/O request issues a system call to the OS.
  - If desired disk drive or controller is available, request is served immediately.
  - If busy, new request for service will be placed in the queue of pending requests.
  - When one request is completed, the OS has to choose which pending request to service next.

Disk Scheduling Algorithms

At-A-Glance

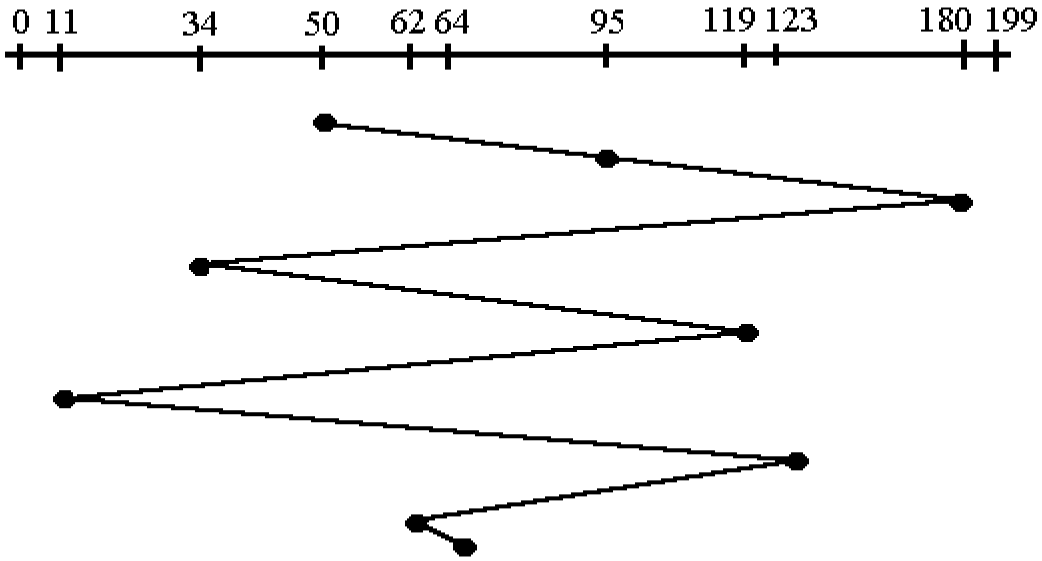
- First Come-First Serve (FCFS)
- Shortest Seek Time First (SSTF)
- Elevator (SCAN)
- Circular SCAN (C-SCAN)
- C-LOOK

Disk Scheduling - FCFS

- **First Come First Serve**
  - Process request sequentially
  - Fair to all processes
  - Approaches random scheduling in performance if there are many processes

Disk Scheduling - FCFS

- Given the following queue:
  - 95, 180, 34, 119, 11, 123, 62, 64
  - with the Read-write head initially at the track 50
  - and the tail track being at 199



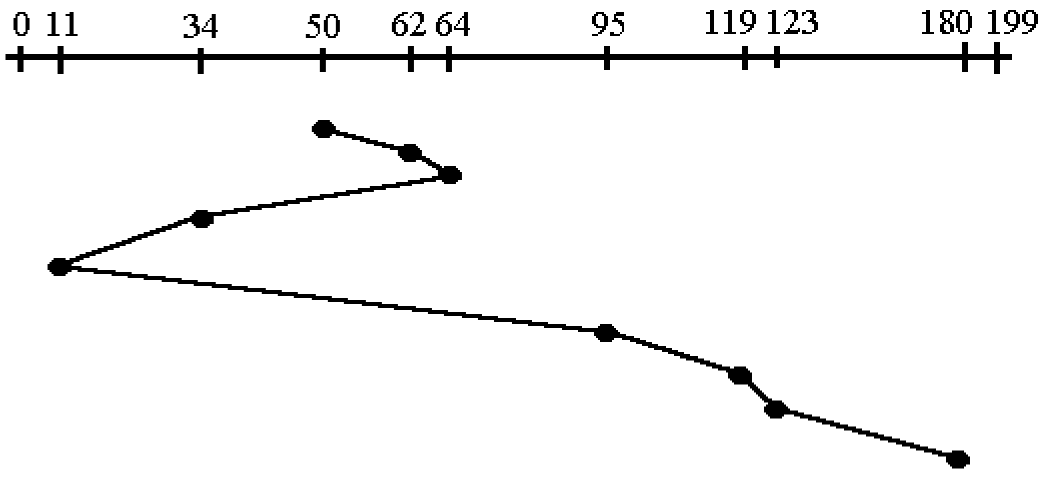
- |50-95|+|95-180|+|180-34|+|34-119|+|119-11|+|11-123|+|123-62|+|62-64|
- **Total:** 45+85+146+85+108+112+61+2 = 640

Disk Scheduling - SSTF

- **Shortest Seek Time First (SSTF)**
  - Select the disk I/O request that requires the least movement of the disk arm from its current position
  - Always choose the minimum seek time
  - Requests for tracks far away from the current position may never be served, if requests for closer tracks are issued continuously

Disk Scheduling - SSTF

- In this case request is serviced according to next shortest distance.



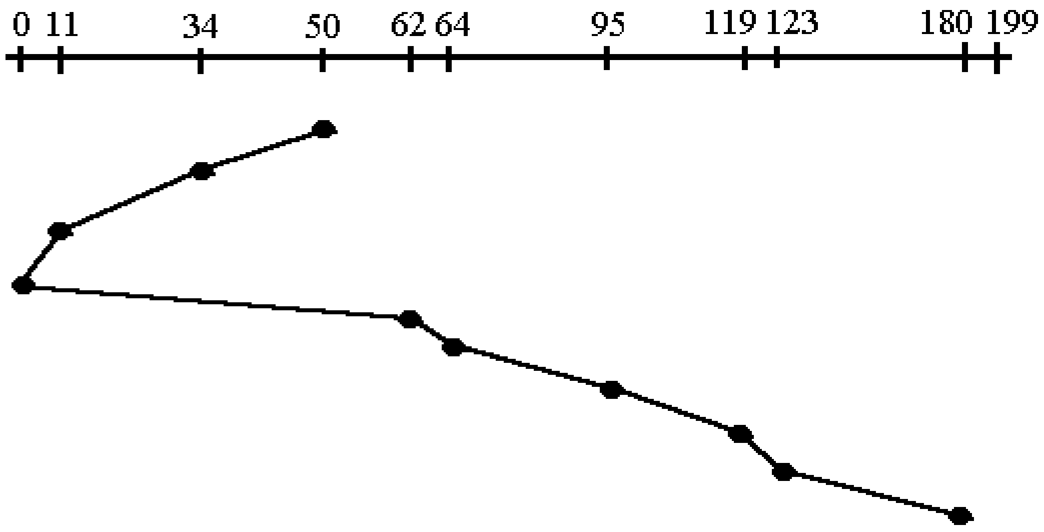
- Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34.
- The process would continue until all the process are taken care of.
- **Total:** 236

Disk Scheduling - SCAN

- **SCAN (aka Elevator Algorithm)**
    - Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
    - Direction is reversed

Disk Scheduling - SCAN

- This approach works like an elevator does.



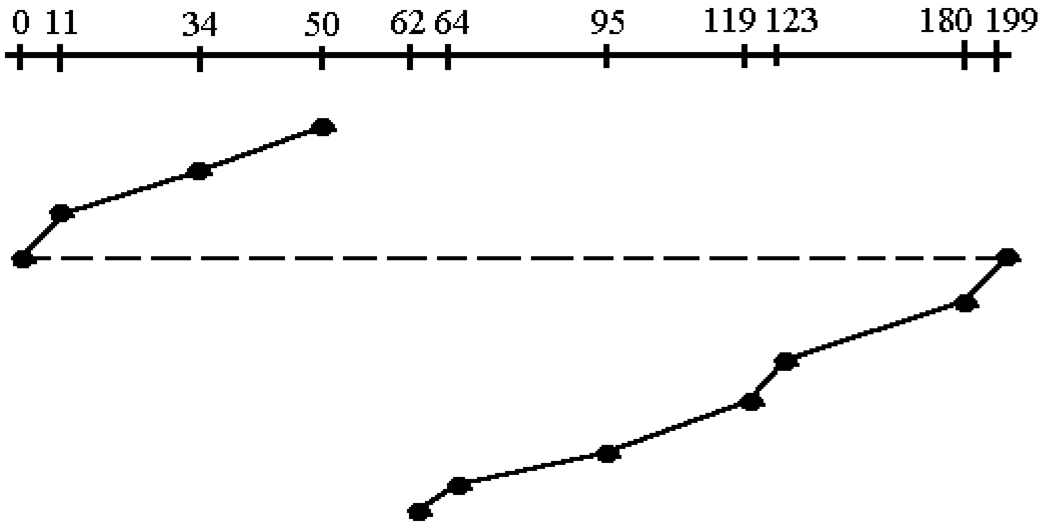
- It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down.
- If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.
- **Total:** 230

Disk Scheduling – C-SCAN

- **C-SCAN**
  - Restricts scanning to one direction only
  - When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again
  - In case of repetitive requests to one track, we will see “arm stickiness” in SSTF, SCAN, C-SCAN

Disk Scheduling – C-SCAN

- Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system.
- Once it hits the bottom or top it jumps to the other end and moves in the same direction.
- Keep in mind that the huge jump doesn't count as a head movement.

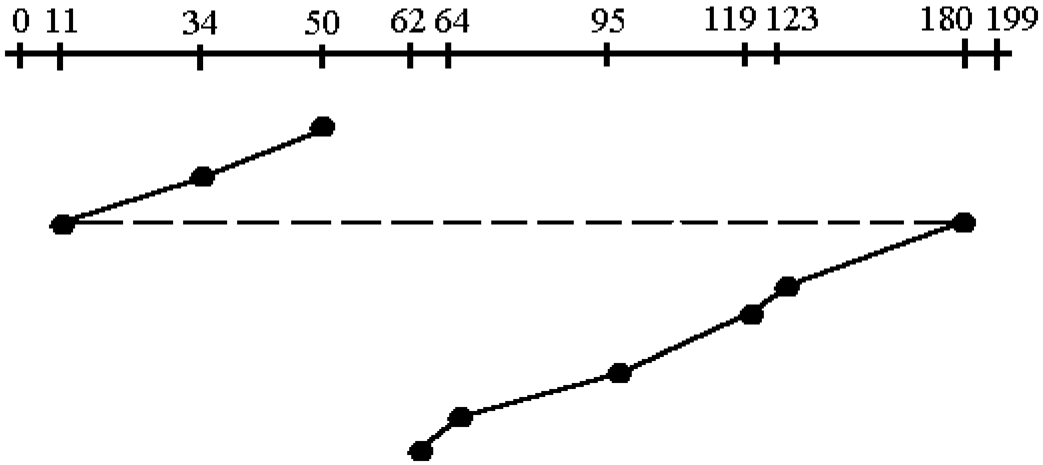


- The total head movement for this algorithm is only 187 track;
- But it is still not the best …

Disk Scheduling – C-LOOK

- **C-LOOK**
- This is just an enhanced version of C-SCAN.
- In this the scanning doesn't go past the last request in the direction that it is moving.
- It too jumps to the other end but not all the way to the end. Just to the furthest request.

Disk Scheduling – C-LOOK



- C-SCAN had a total movement of 187, but this scan (C-LOOK) reduced it down to 157 tracks.

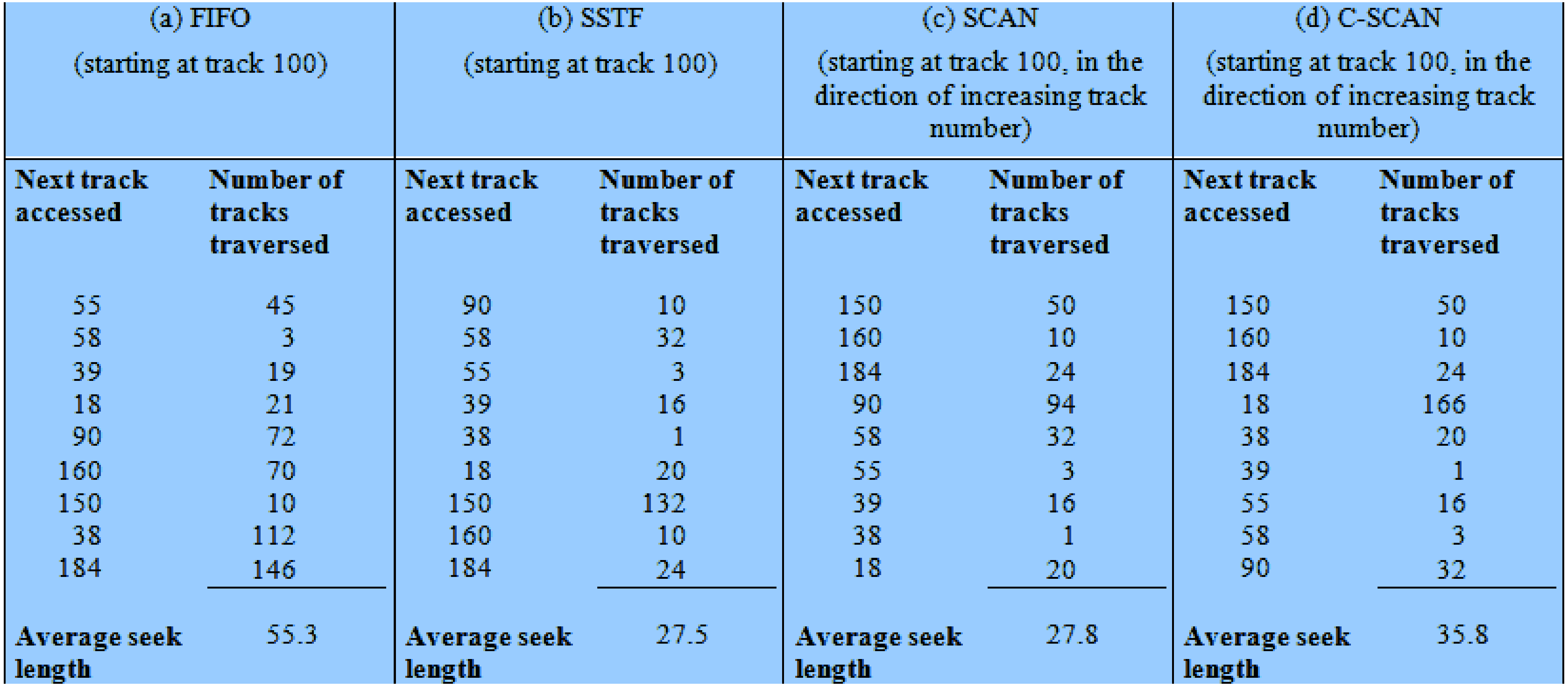
Disk Scheduling Algorithms

Comparisons

- (640) - First Come-First Serve (FCFS)
- (236) - Shortest Seek Time First (SSTF)
- (230) - Elevator (SCAN)
- (187) - Circular SCAN (C-SCAN)
- **(157) - C-LOOK**

Another Example

- Example: 55, 58, 39, 18, 90, 160, 150, 38, 184



(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

RAID Motivation

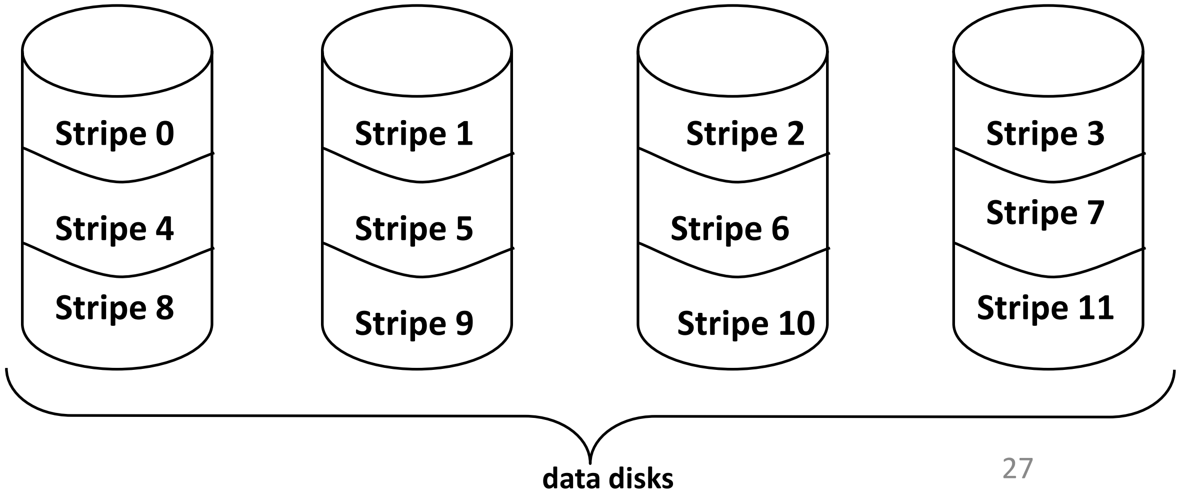
- We can use multiple disks for improving performance
  - By striping files across multiple disks (placing parts of each file on a different disk), parallel I/O can improve access time
- Striping reduces reliability
  - What’s the mean time between failures of 100 disks, assuming T as the mean time between failures of one disk?
  - The mean time between failures of 100 disks = 1/100 times of the mean time between failures of one disk
- So, we need striping for performance, but we need something to help with reliability
- To improve reliability, we can add redundant data to the disks, in addition to striping

RAID

- A RAID is a Redundant Array of Inexpensive Disks
  - “I” is also for “Independent”
  - The alternative is SLED, single large expensive disk
- Disks are small and cheap, so it’s easy to put lots of disks (10s to 100s) in one box for increased storage, performance, and availability
- The RAID box with a RAID controller looks just like a SLED to the computer
- Data plus some redundant information is striped across the disks in some way
- How that striping is done is key to performance and reliability.

RAID 0

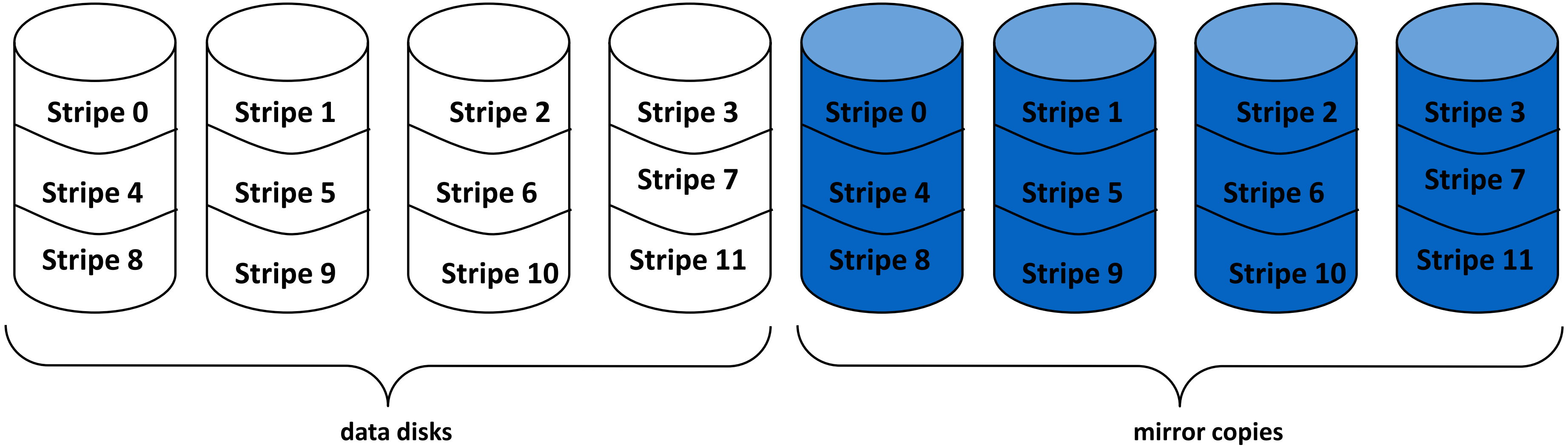
- Level 0 is nonredundant disk array
- Files are *striped* across disks, no redundant info
- High read throughput
- Best write throughput (no redundant info to write)
- Any disk failure results in data loss
  - Reliability worse than SLED



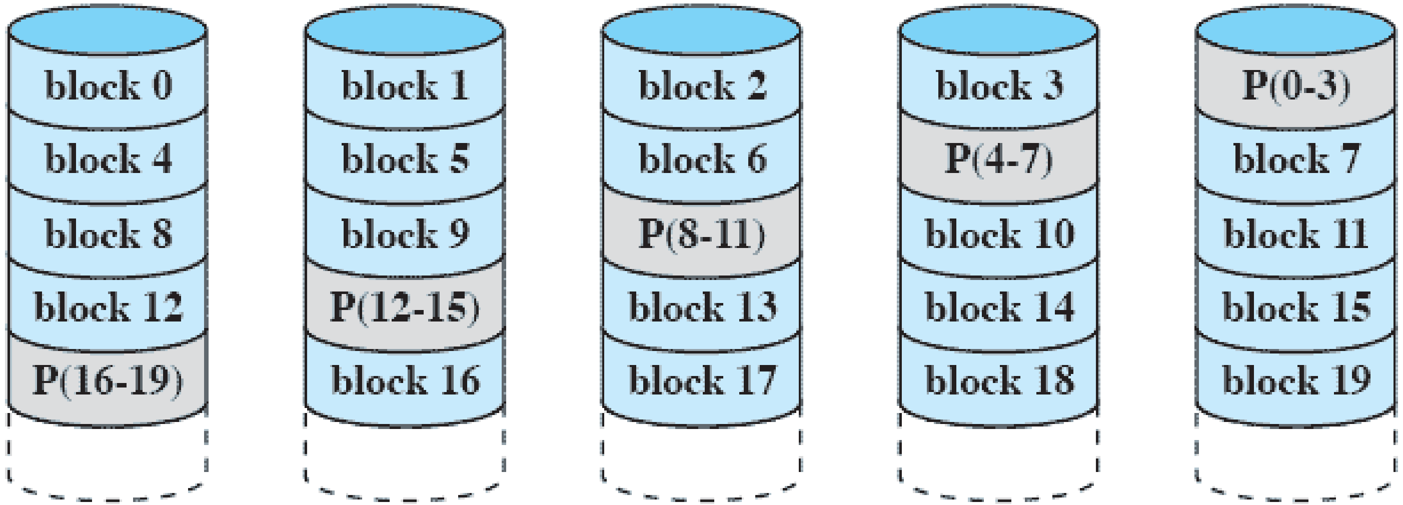
RAID 1

- Mirrored Disks
- Data is written to two places
  - On failure, just use surviving disk
- On read, choose fastest to read
  - Write performance is same as single drive, read performance is 2x better
- Replication redundancy is expensive

RAID 1


Stripe 0
Stripe 1
Stripe 2
Stripe 3
Stripe 4
Stripe 5
Stripe 6
Stripe 7
Stripe 8
Stripe 9
Stripe 10
Stripe 11
Stripe 0
Stripe 1
Stripe 2
Stripe 3
Stripe 4
Stripe 5
Stripe 6
Stripe 7
Stripe 8
Stripe 9
Stripe 10
Stripe 11
data disks
mirror copies

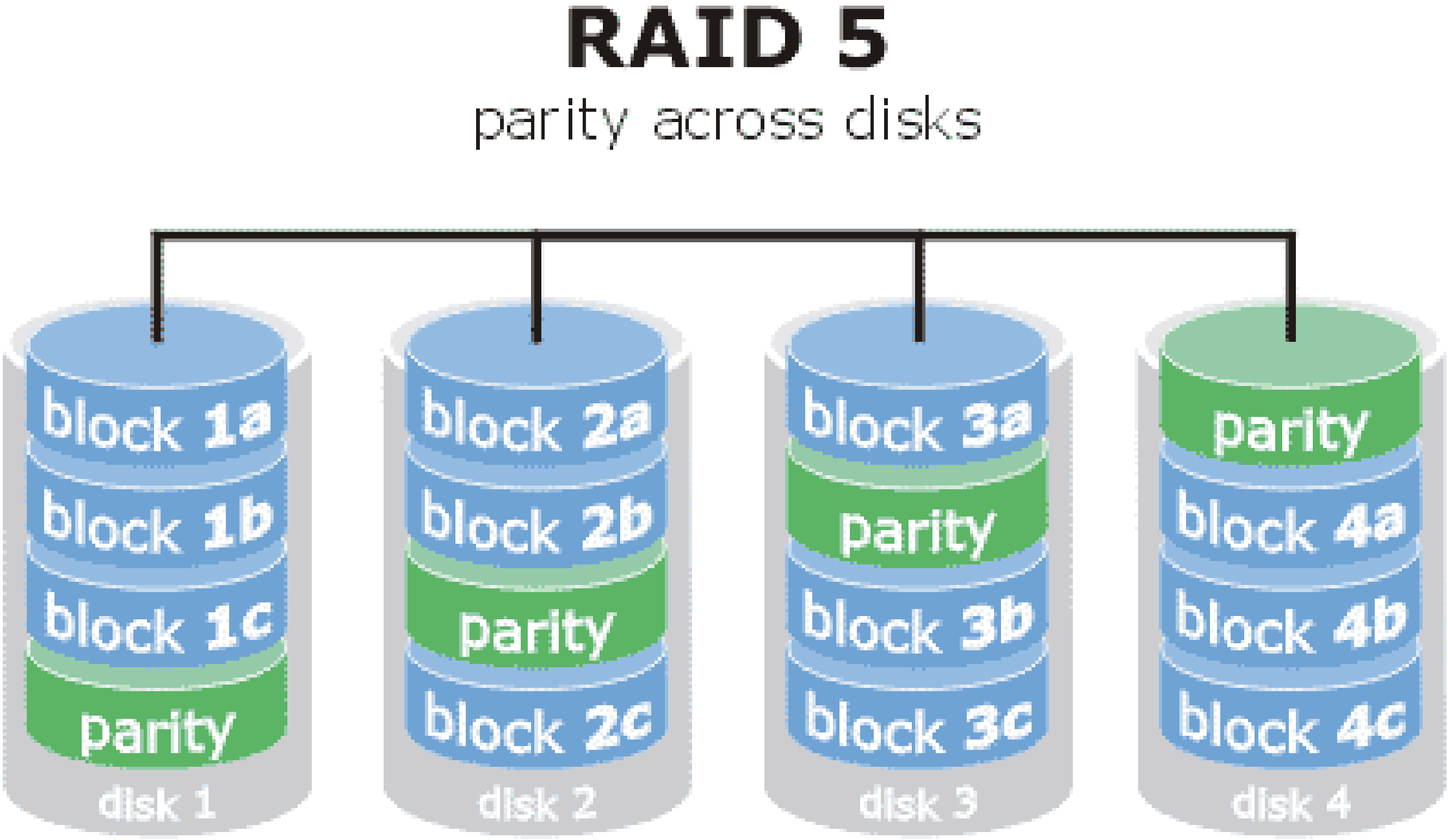
RAID 5 (block-level distributed parity)



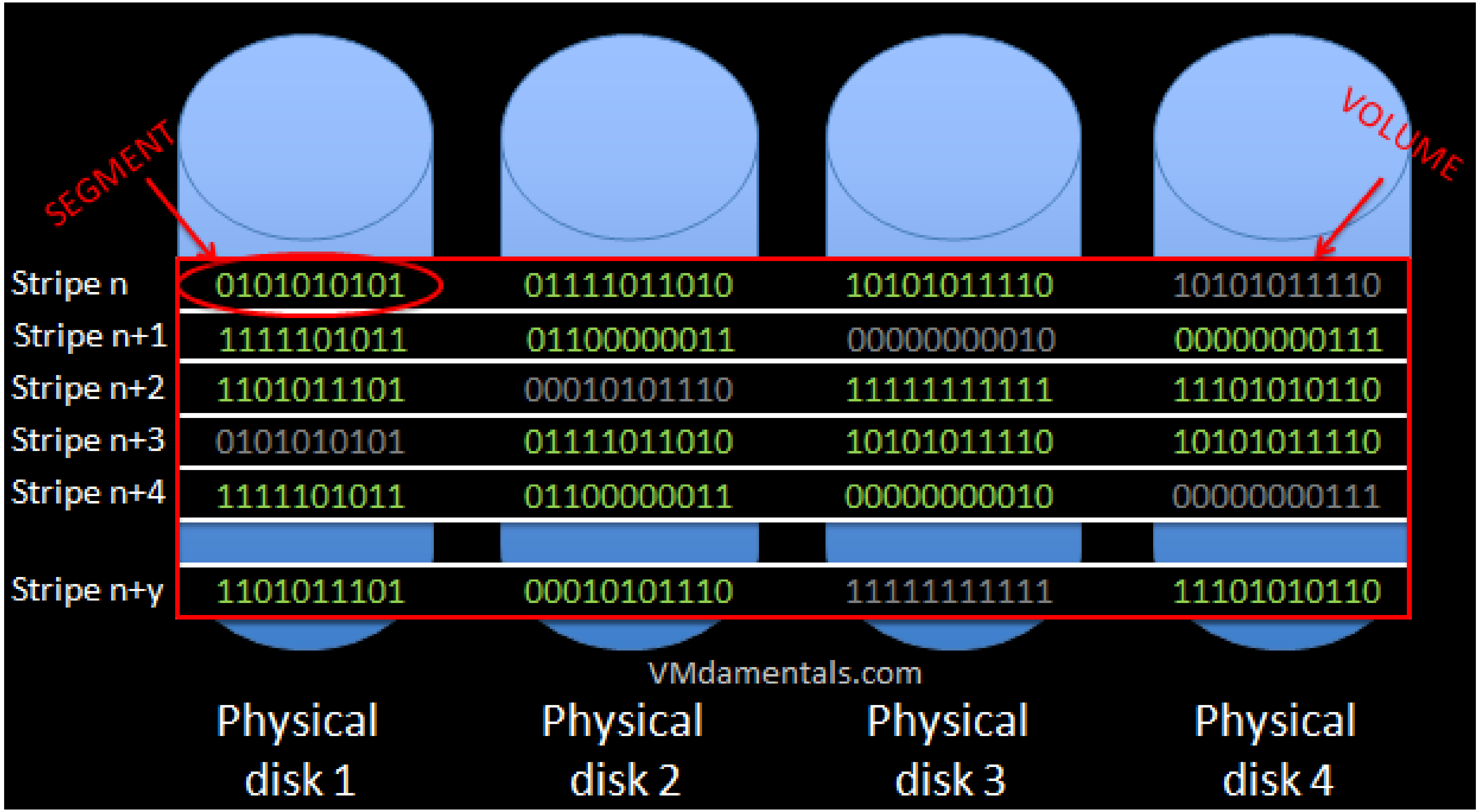
block 0	block 1	block 2	block 3	P(0-3)
block 4	block 5	block 6	P(4-7)	block 7
block 8	block 9	P(8-11)	block 10	block 11
block 12	P(12-15)	block 13	block 14	block 15
P(16-19)	block 16	block 17	block 18	block 19

(f) RAID 5 (block-level distributed parity)

RAID 5 (block-level distributed parity)



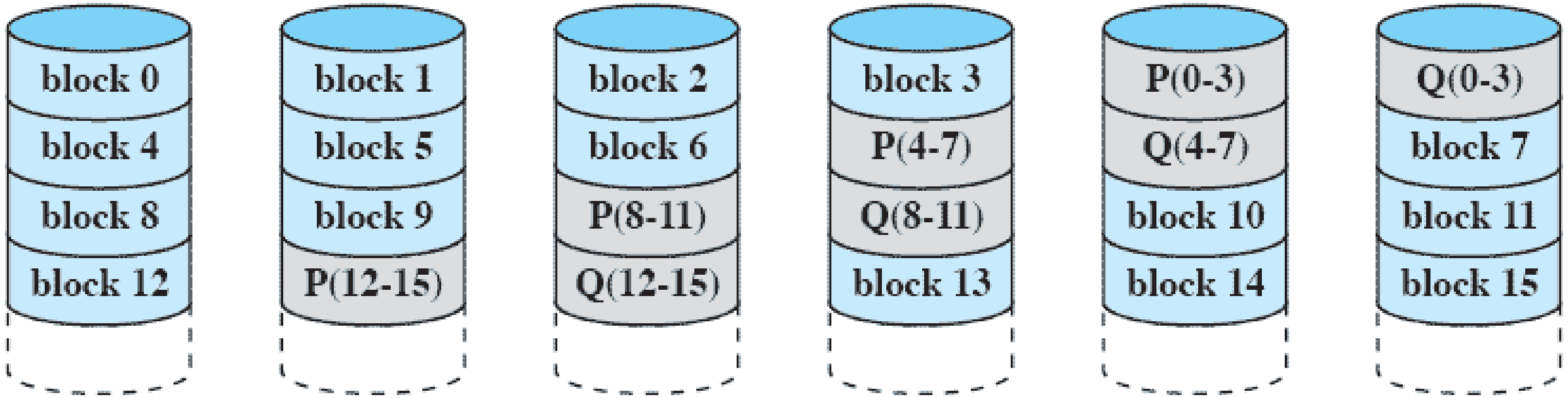
RAID 5 ("Write Penalty")



	Physical disk 1	Physical disk 2	Physical disk 3	Physical disk 4
Stripe n	0101010101	01111011010	10101011110	10101011110
Stripe n+1	1111101011	01100000011	00000000010	00000000111
Stripe n+2	1101011101	00010101110	11111111111	11101010110
Stripe n+3	0101010101	01111011010	10101011110	10101011110
Stripe n+4	1111101011	01100000011	00000000010	00000000111
Stripe n+y	1101011101	00010101110	11111111111	11101010110

http://vmdamentals.com/?p=3200

RAID 6 (dual redundancy)



block 0	block 1	block 2	block 3	P(0-3)	Q(0-3)
block 4	block 5	block 6	P(4-7)	Q(4-7)	block 7
block 8	block 9	P(8-11)	Q(8-11)	block 10	block 11
block 12	P(12-15)	Q(12-15)	block 13	block 14	block 15

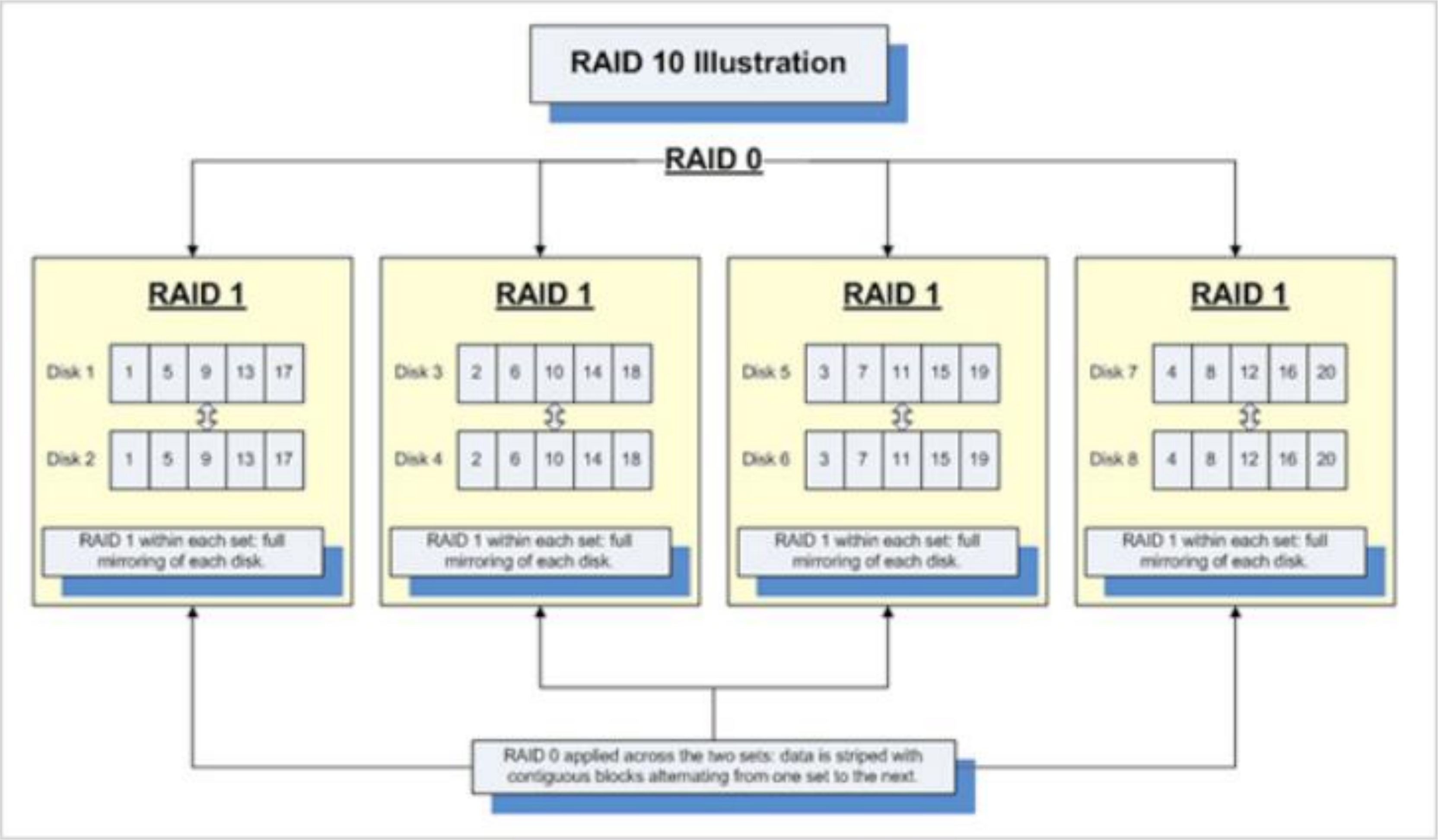
(g) RAID 6 (dual redundancy)

- Level 5 with an extra parity bit, which is generated with a different and independent data check algorithm

- Can tolerate two failures

RAID 1+0

RAID 10 is sometimes also called RAID 1+0



Disk 1	1	5	9	13	17
Disk 2	1	5	9	13	17
Disk 3	2	6	10	14	18
Disk 4	2	6	10	14	18
Disk 5	3	7	11	15	19
Disk 6	3	7	11	15	19
Disk 7	4	8	12	16	20
Disk 8	4	8	12	16	20

RAID 1 within each set: full mirroring of each disk.

RAID 1 within each set: full mirroring of each disk.

RAID 1 within each set: full mirroring of each disk.

RAID 1 within each set: full mirroring of each disk.

RAID 0 applied across the two sets: data is striped with contiguous blocks alternating from one set to the next.

source: http://www.illinoisdataservices.com/raid-10-data-recovery.html

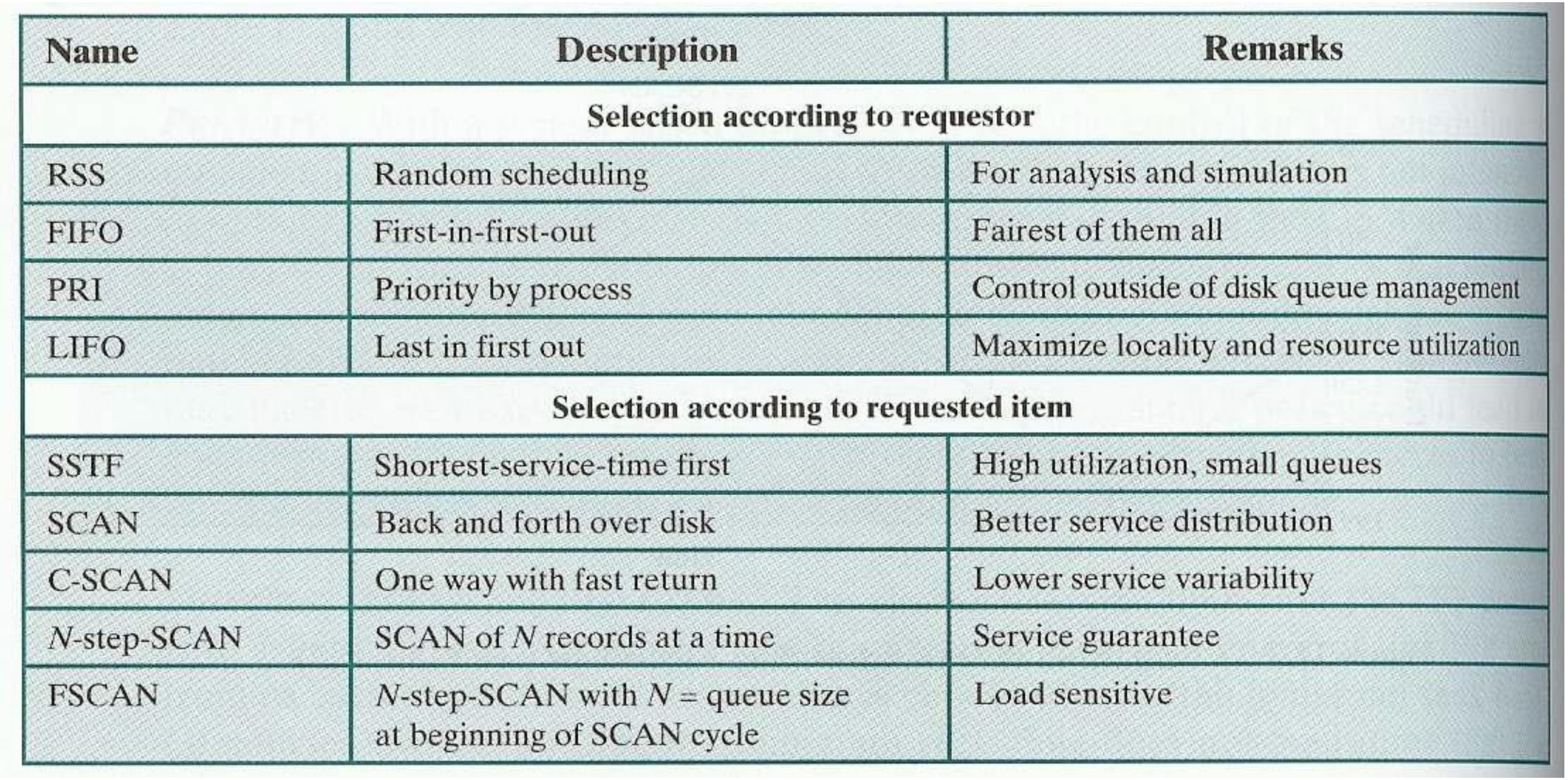
Summary

- Disk Organization
- Performance Parameters
- Disk Scheduling Algorithms
- RAID Motivation

Thank you!

Kanlue Zhang

(05/19/2015)



Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First-in-first-out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest-service-time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of *N* records at a time	Service guarantee
FSCAN	*N*-step-SCAN with *N* = queue size at beginning of SCAN cycle	Load sensitive

Parity and Hamming Codes

- What do you need to do in order to detect and correct a one-bit error ?
  - Suppose you have a binary number, represented as a collection of bits: <b3, b2, b1, b0>, e.g. 0110
- Detection is easy
- Parity:
  - Count the number of bits that are on, see if it’s odd or even
    - EVEN parity is 0 if the number of 1 bits is even
  - Parity(<b3, b2, b1, b0 >) = $P0 = b0 \oplus b1 \oplus b2 \oplus b3$
  - Parity(<b3, b2, b1, b0, p0>) = 0 if all bits are intact
  - Parity(0110) = 0, Parity(01100) = 0
  - Parity(11100) = 1 => ERROR!
  - Parity can detect a single error, but can’t tell you which of the bits got flipped

Hamming Code History

- In the late 1940's Richard Hamming recognized that the further evolution of computers required greater reliability, in particular the ability to not only detect errors, but correct them. His search for error-correcting codes led to the Hamming Codes, perfect 1-error correcting codes, and the extended Hamming Codes, 1-error correcting and 2-error detecting codes.

Parity and Hamming Codes

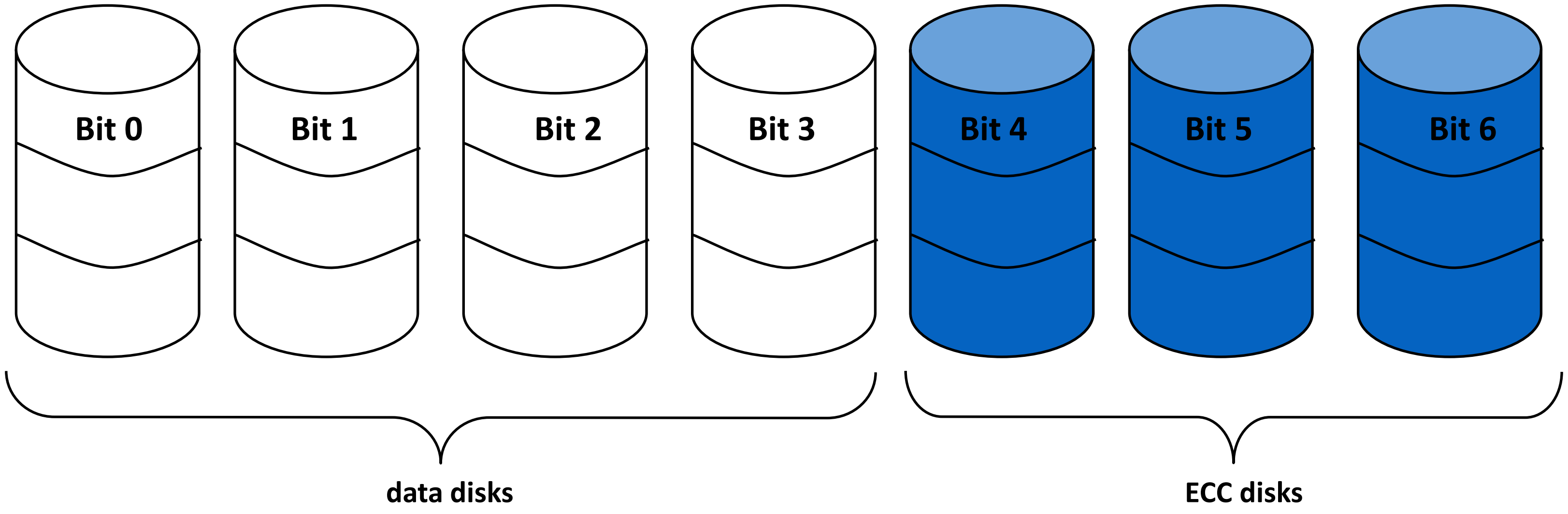
- Detection and correction require more work
- Hamming codes can detect & correct single bit errors
- [7,4] binary Hamming Code
  - $h0 = b0 \oplus b1 \oplus b3$
  - $h1 = b0 \oplus b2 \oplus b3$
  - $h2 = b1 \oplus b2 \oplus b3$
  - H0(<1101>) = 0
  - H1(<1101>) = 1
  - H2(<1101>) = 0
  - Hamming(<1101>) = <b3, b2, b1, h2, b0, h1, h0> = <1100110>
  - If a bit is flipped, e.g. <1110110>
  - $a = h0 \oplus b0 \oplus b1 \oplus b3 = 1$
  - $b = h1 \oplus b0 \oplus b2 \oplus b3 = 0$
  - $c = h2 \oplus b1 \oplus b2 \oplus b3 = 1$
  - abc = <101>, the 5th bit is in error and switch it

Extended [8,4] binary Hamm. Code

- As with the [7,4] binary Hamming Code:
    - $h0 = b0 \oplus b1 \oplus b3$
    - $h1 = b0 \oplus b2 \oplus b3$
    - $h2 = b1 \oplus b2 \oplus b3$
- Add a new bit p such that
    - $p = b0 \oplus b1 \oplus b2 \oplus b3 \oplus h0 \oplus h1 \oplus h2$. i.e., the new bit makes the XOR of all 8 bits zero. p is called a parity check bit.
    - Assume at most 2 errors:
        - If parity check passes and abc = 000, the system is correct;
        - If parity check passes and abc ≠ 000, the system has two errors;
        - If parity check fails, there is one error and abc indicates which bit is in error.

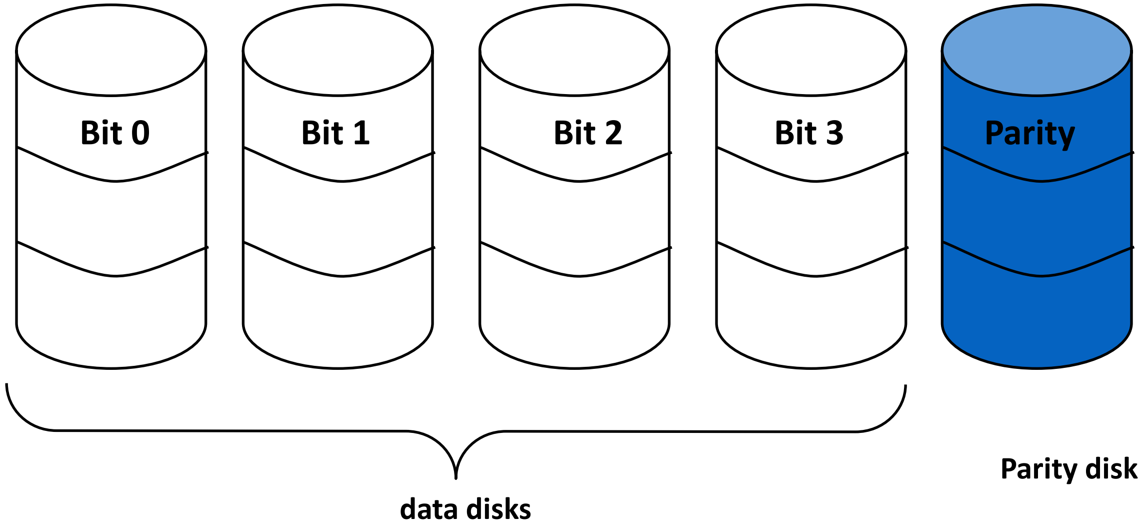
Raid Level 2

- Bit-level *striping* with Hamming (ECC) codes for error correction
- All 7 disk arms are synchronized and move in unison
- Complicated controller
- Single access at a time
- Tolerates only one error

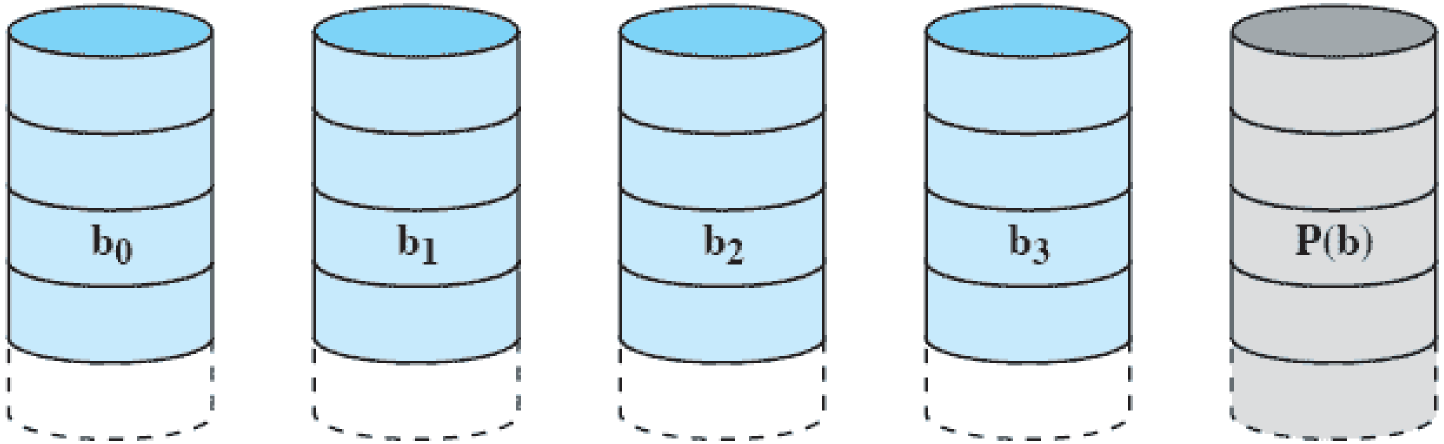


Raid Level 3

- Use a parity disk
  - Each bit on the parity disk is a parity function of the corresponding bits on all the other disks
- A read accesses all the data disks
- A write accesses all data disks plus the parity disk
- On a disk failure, read remaining disks plus parity disk to compute the missing data



RAID 3



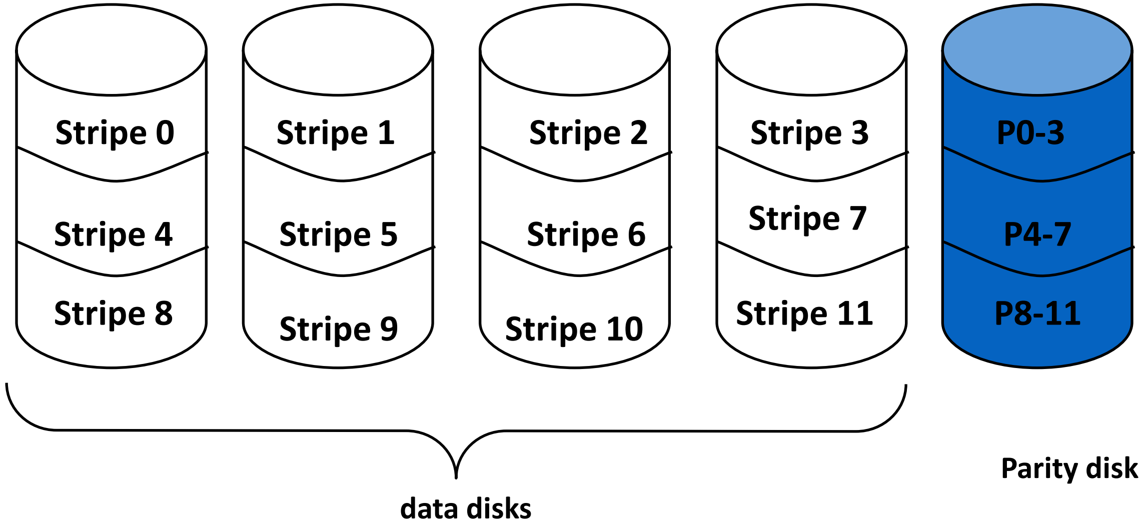
(d) RAID 3 (bit-interleaved parity)

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

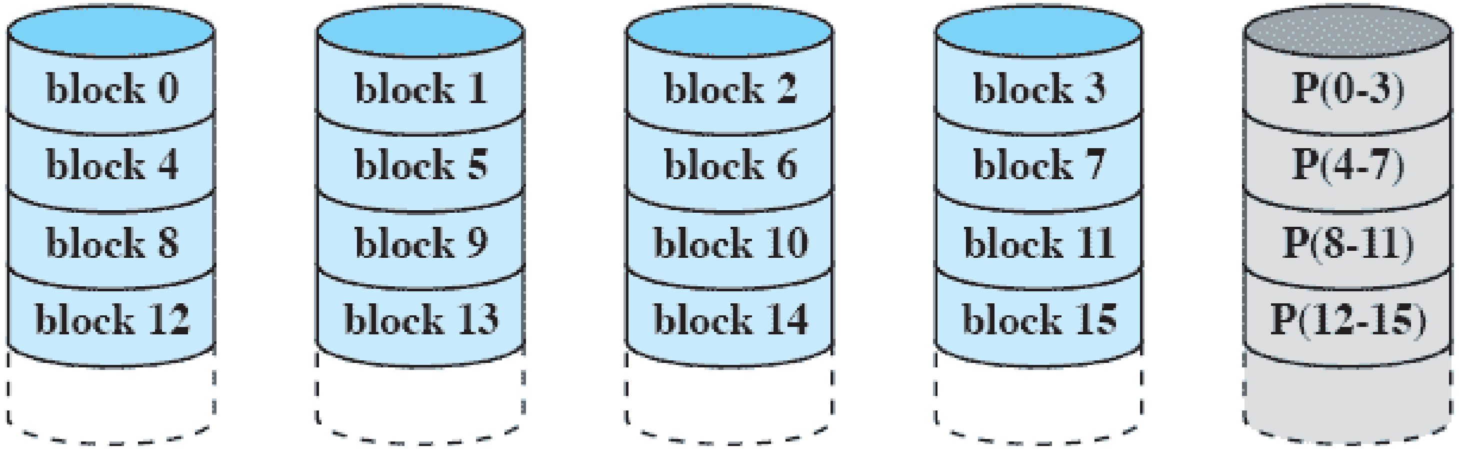
If drive X1 has failed. How to recover it?

Raid Level 4

- Combines Level 0 and 3 – block-level parity with Stripes
- A read accesses all the data disks
- A write accesses all data disks plus the parity disk



RAID 4 (block-level parity)



(e) RAID 4 (block-level parity)

$X4(i) = X3(i) \oplus X2(i) \oplus \mathbf{X1(i)} \oplus \mathbf{X0(i)}$

How to execute a write operation, for instance on drive X1?

Heavy load on the parity disk