I/O Management and Disk Scheduling

Agenda

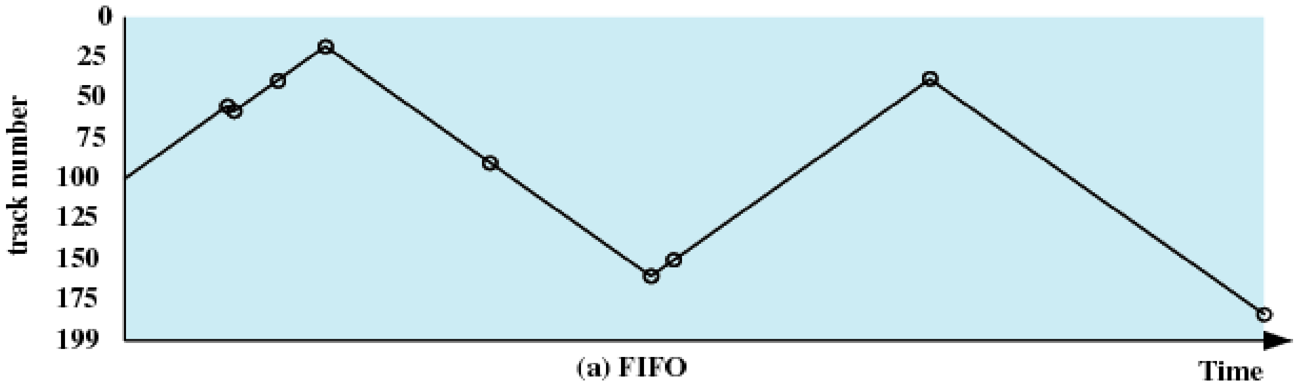
- ✓ **I/O Devices**
- ✓ **Organization of the I/O Function**
- ✓ **Operating System Design Issues**
- ✓ **I/O Buffering**
- **Disk Scheduling**
- **RAID**
- **Disk Cache**
- **Linux I/O**

Disk Scheduling Policies

- **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, we will poor performance

Disk Scheduling Policies

- First-in, first-out (FIFO)
  - Process request sequentially
  - Fair to all processes
  - Approaches random scheduling in performance if there are many processes



(a) FIFO

With FIFO, if there are only a few processes that require access and if many of the requests are to clustered file sectors, then we can hope for good performance.

However, this technique will often approximate random scheduling in performance, if there are many processes competing for the disk.

Disk Scheduling Policies

- Priority
  - Goal is not to optimize disk use **<u>but to meet other objectives</u>**
  - Short batch jobs may have higher priority
  - Provide good interactive response time

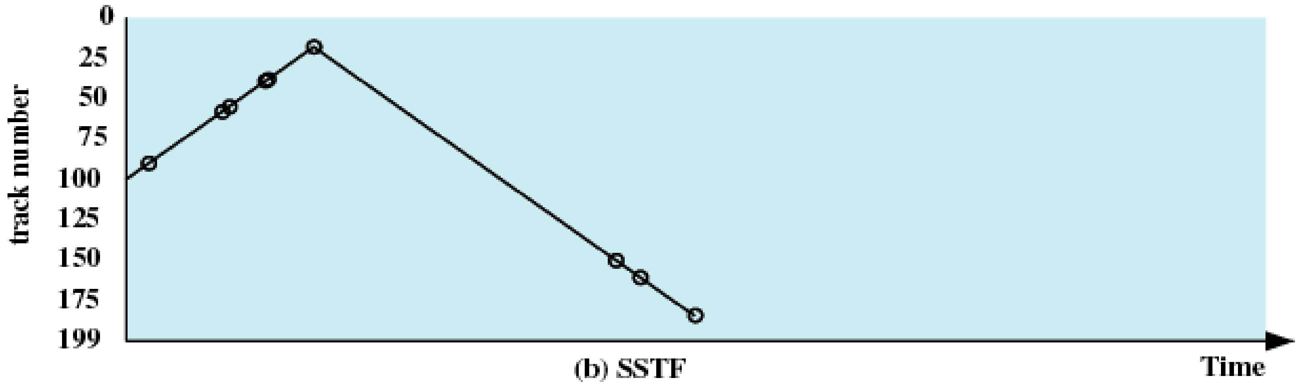
However, longer jobs may have to wait excessively long times.

Furthermore, such a policy could lead to countermeasures on the part of users, who split their jobs into smaller pieces to beat the system.

This type of policy tends to be poor for database systems.

Disk Scheduling Policies

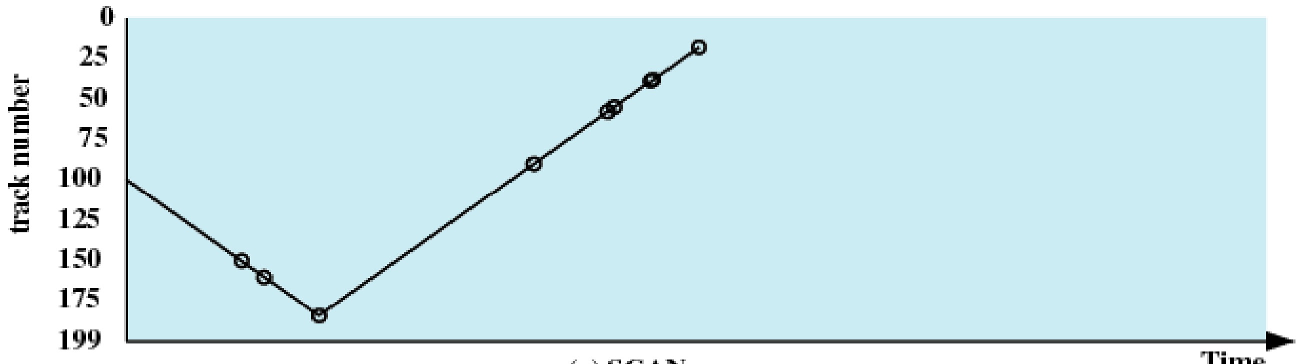
- Shortest Service Time First
  - Select the disk I/O request that requires the least movement of the disk arm from its current position
  - Always choose the minimum Seek time



(b) SSTF

Disk Scheduling Policies

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



(c) SCAN

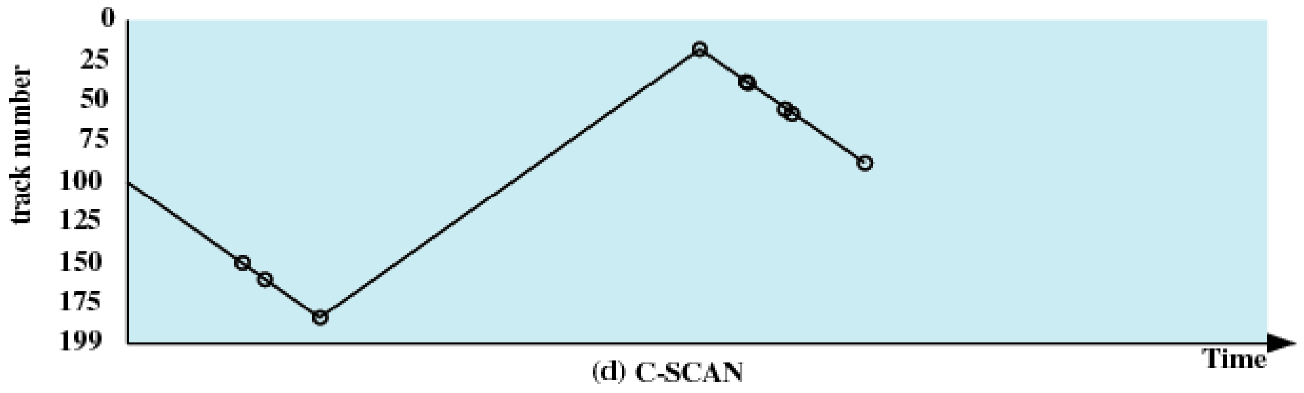
SCAN not exploit locality as well as SSTF.

It is not difficult to see that the SCAN policy favors jobs whose requests are for tracks nearest to both innermost and outermost tracks and favors the latest arriving jobs.

The first problem can be avoided via the C-SCAN policy,
The second problem is addressed by the N-step-SCAN policy.

Disk Scheduling Policies

- 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



(d) C-SCAN

Disk Scheduling Policies

- N-step-SCAN
  - Segments the disk request queue into subqueues of length N
  - Subqueues are processed one at a time, using SCAN
  - New requests added to other queue when queue is processed
- FSCAN
  - Two queues
  - One queue is empty for new requests

Disk Scheduling Algorithms

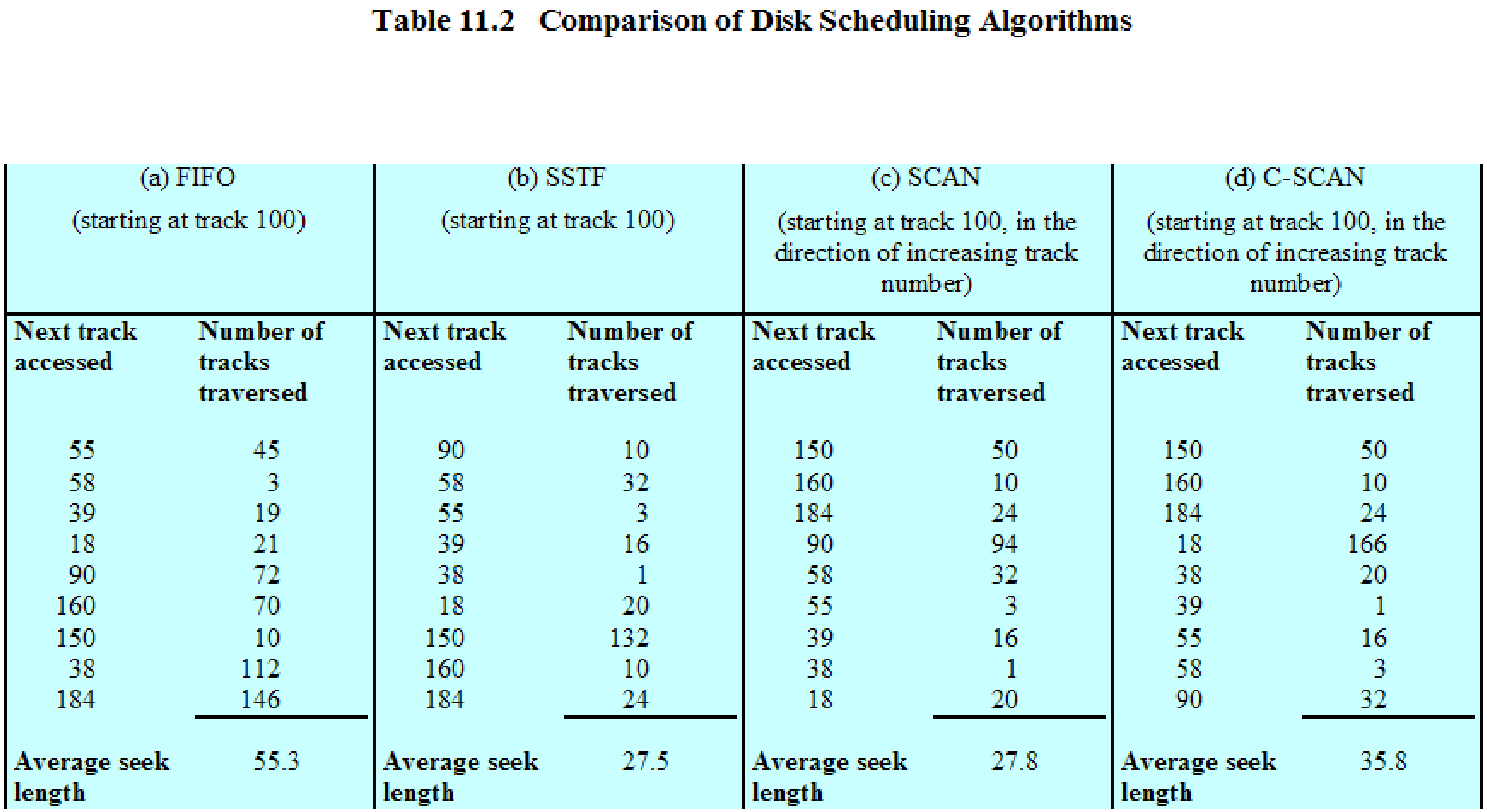


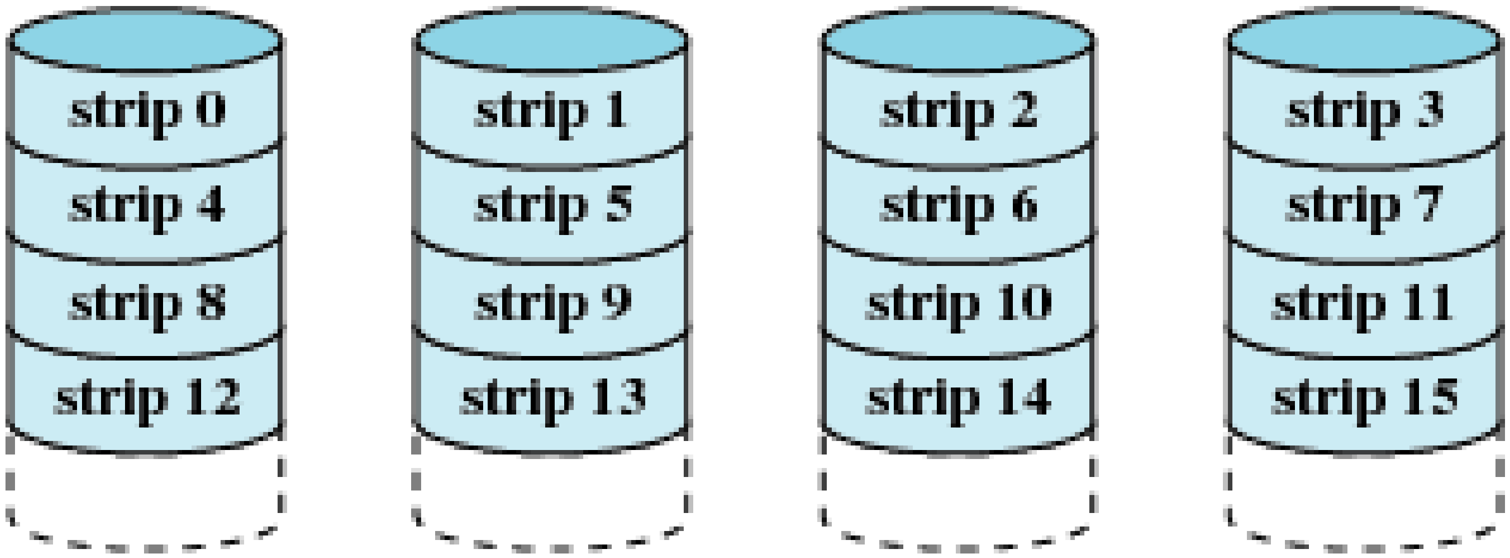
Table 11.2 Comparison of Disk Scheduling Algorithms

(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

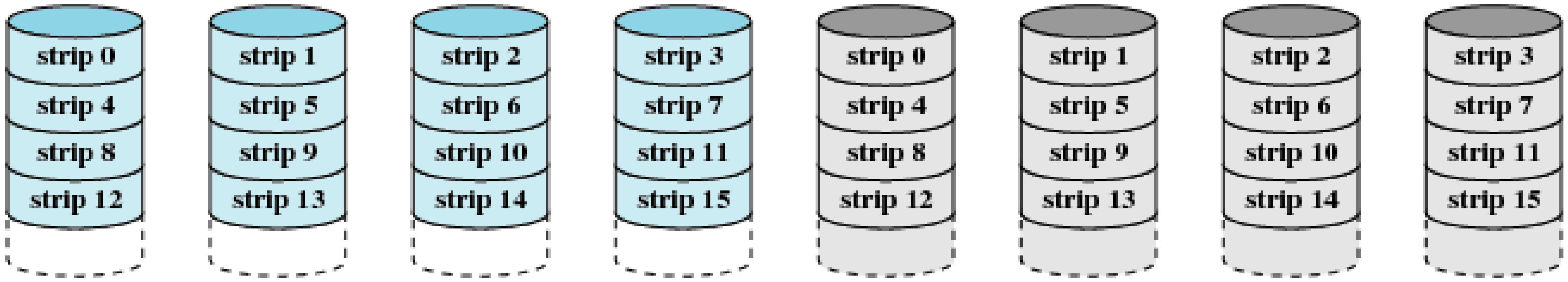
- **Redundant** Array of Independent Disks
- Set of physical disk drives viewed by the operating system as a single logical drive
- Data are distributed across the physical drives of an array
- **Redundant** disk capacity is used to store parity information

RAID 0 (non-redundant)



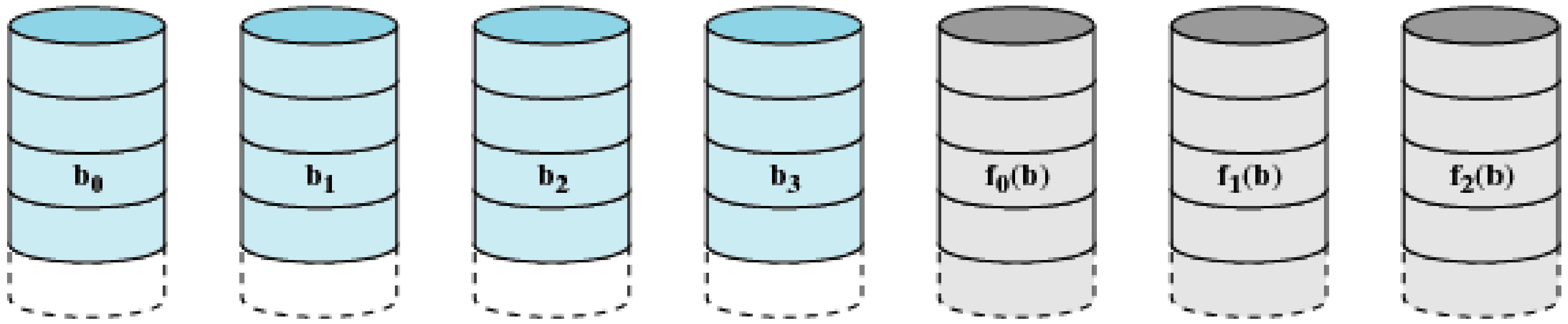
(a) RAID 0 (non-redundant)

RAID 1 (mirrored)



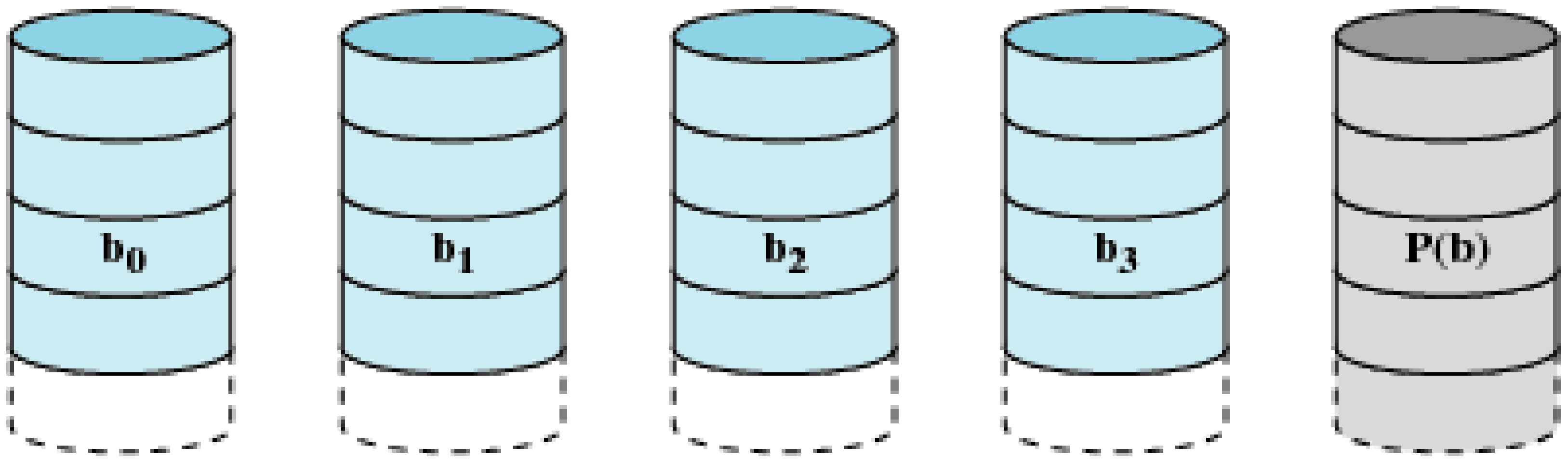
(b) RAID 1 (mirrored)

RAID 2 (redundancy through Hamming code)



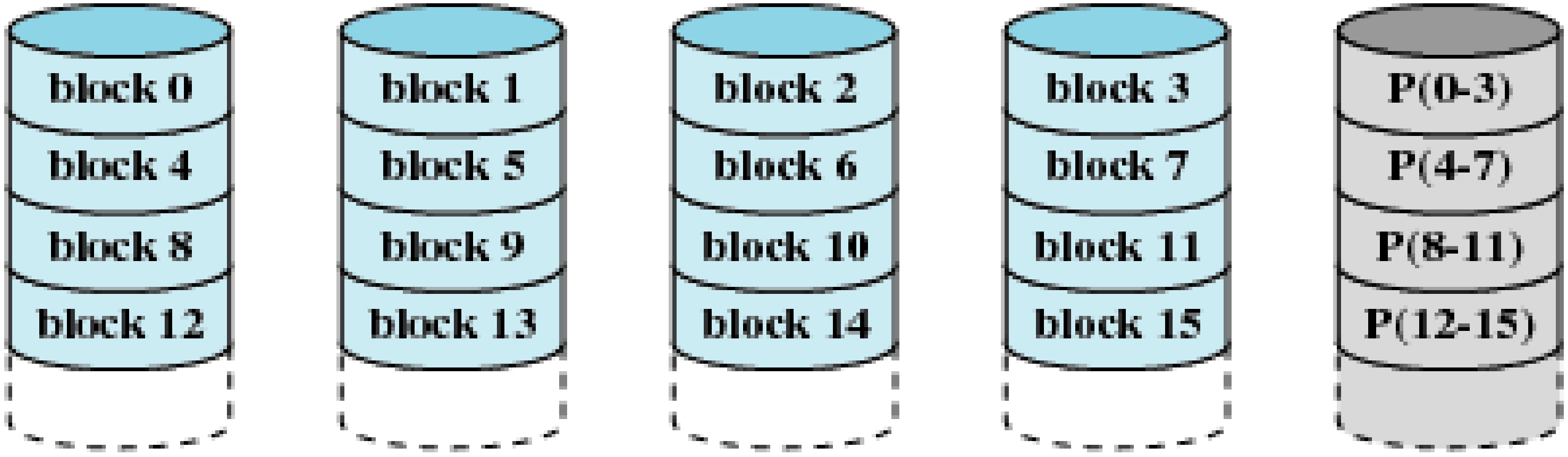
(c) RAID 2 (redundancy through Hamming code)

RAID 3 (bit-interleaved parity)



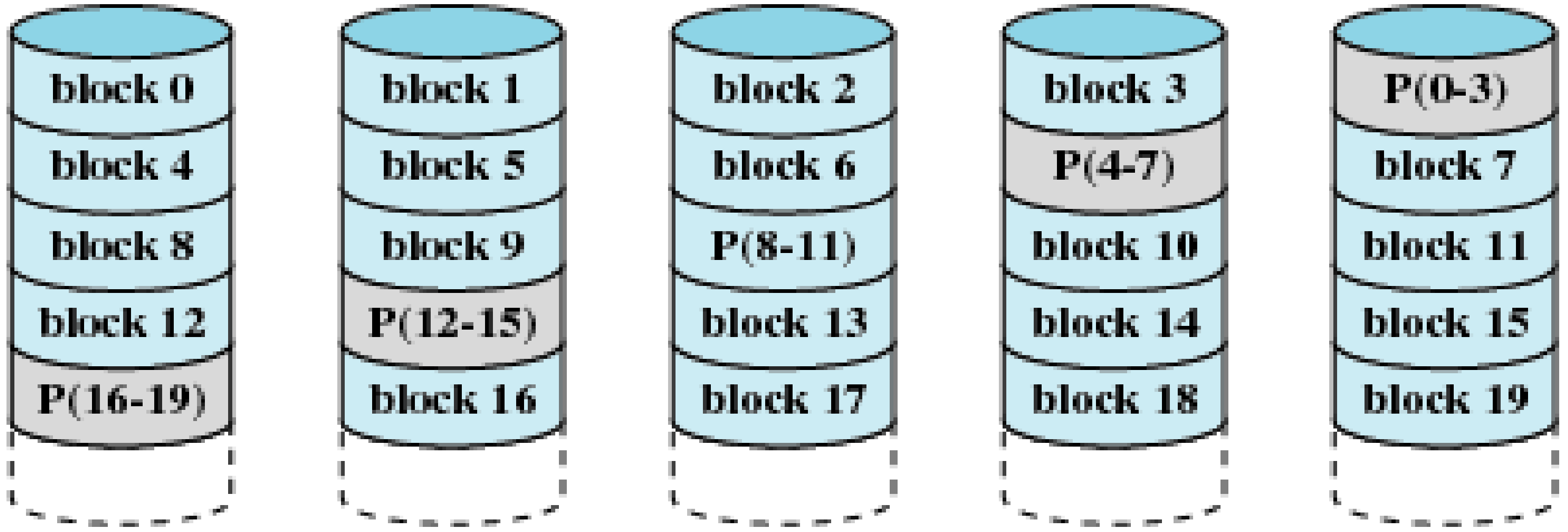
(d) RAID 3 (bit-interleaved parity)

RAID 4 (block-level parity)



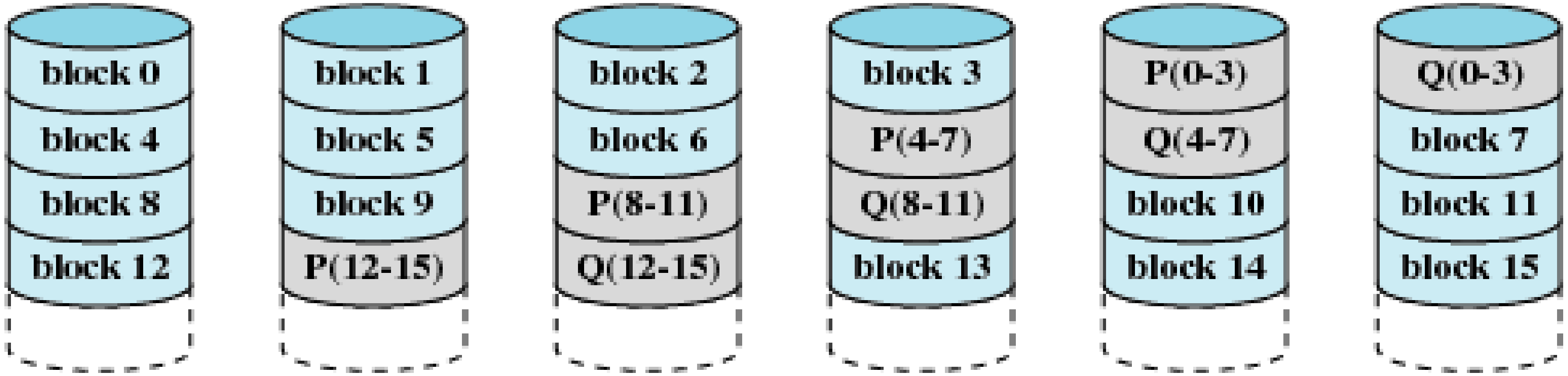
(e) RAID 4 (block-level parity)

RAID 5 (block-level distributed parity)



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

RAID 6 (dual redundancy)



(g) RAID 6 (dual redundancy)

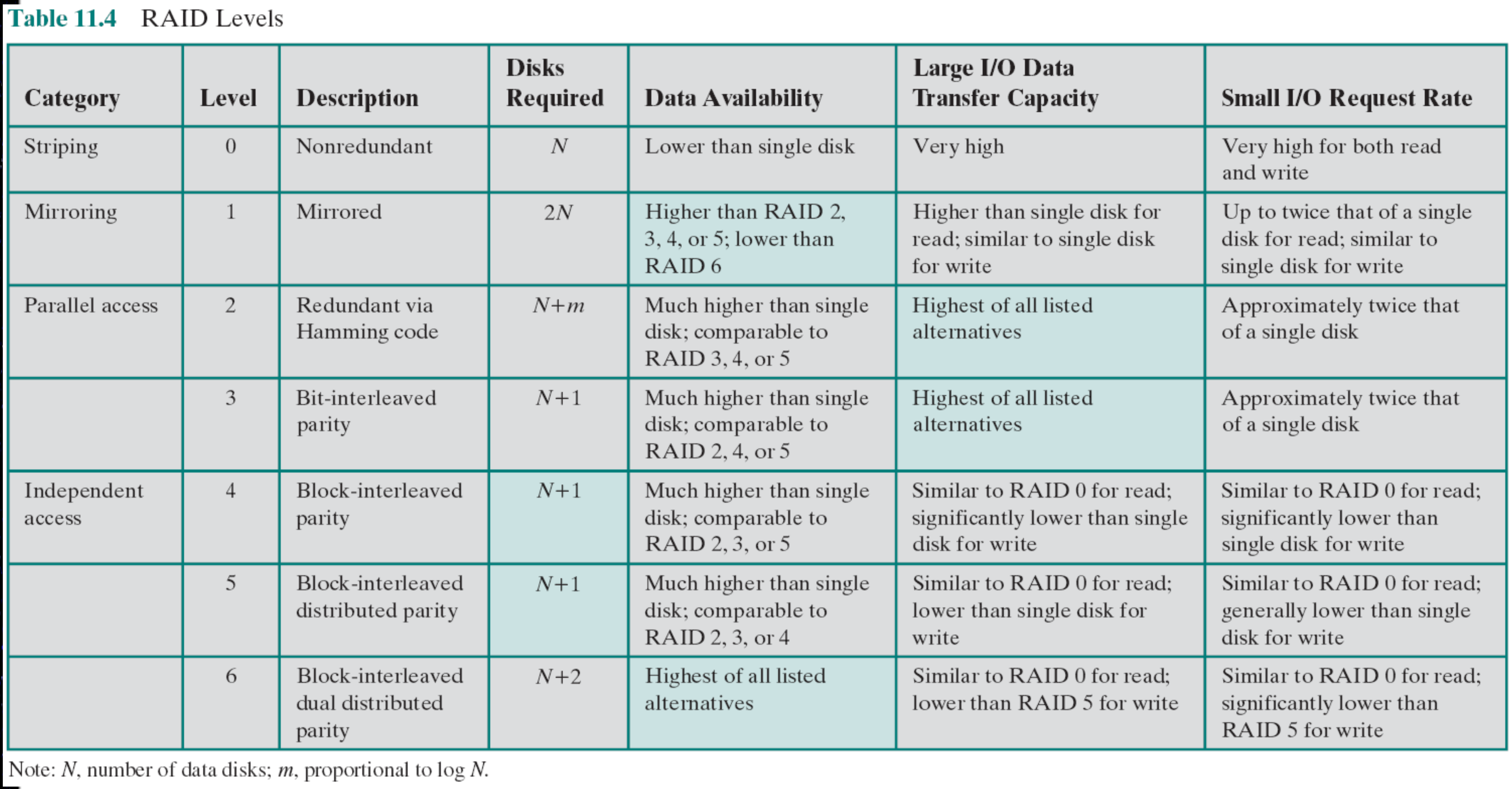


Table 11.4 RAID Levels

Category	Level	Description	Disks Required	Data Availability	Large I/O Data Transfer Capacity	Small I/O Request Rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N+m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N+1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N+1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N+1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N+2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

Note: N, number of data disks; m, proportional to log N.

Disk Cache

- Buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk

Least Recently Used

- The block that has been in the cache the longest with no reference to it is replaced
- The cache consists of a stack of blocks
- Most recently referenced block is on the top of the stack
- When a block is referenced or brought into the cache, it is placed on the top of the stack

Least Recently Used

- The block on the bottom of the stack is removed when a new block is brought in
- Blocks don’t actually move around in main memory
- A stack of pointers is used

Least Frequently Used

- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block accessed
- Block with smallest count is selected for replacement
- **Some blocks may be referenced many times in a short period of time and the reference count is misleading**

To overcome this difficulty with LFU, a technique known as frequency-based replacement

Look text page 504

Linux I/O

- Elevator scheduler
  - Maintains a single queue for disk read and write requests
  - Keeps list of requests sorted by block number
  - Drive moves in a single direction to satisfying each request

Linux I/O

- Deadline scheduler
  - Uses three queues
    - Incoming requests
    - Read requests go to the tail of a FIFO queue
    - Write requests go to the tail of a FIFO queue
  - Each request has an expiration time

Linux I/O

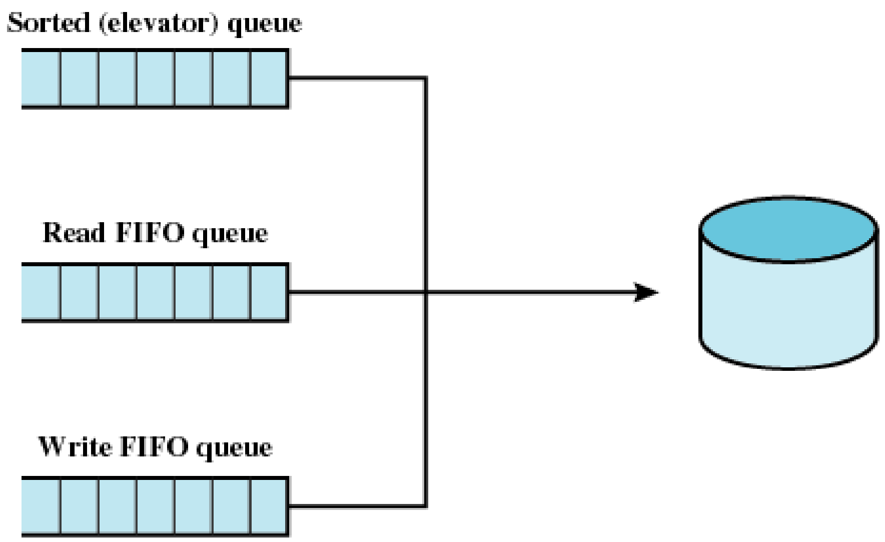


Figure 11.14 The Linux Deadline I/O Scheduler

Sheet 3

Review Questions page 517

Problem

1, 2, 3,

7, 8, 9, 10, 12