I/O Management and Disk Scheduling

Chapter 11

Contents

- I/O devices
- Organization of I/O function
- OS design issues
- I/O buffering
- Disk scheduling
- RAID
- Disk cache
- Unix SVR4 I/O
- Windows 2000 I/O

Categories of I/O Devices

- Human readable
  - used to communicate with the user
  - video display terminals
  - keyboard
  - mouse
  - printer

Categories of I/O Devices

- Machine readable
  - used to communicate with electronic equipment
  - disk drives
  - tape drives
  - controllers
  - sensors
  - actuators

Categories of I/O Devices

- Communication
  - used to communicate with remote devices
  - digital line drivers
  - modems
  - network drivers

Differences in I/O Devices

- Data Transfer Rate
- Application
  - disk used to store files must have file-management software
  - disk used to store virtual memory pages needs special hardware to support it
  - terminal used by system administrator may have a higher priority

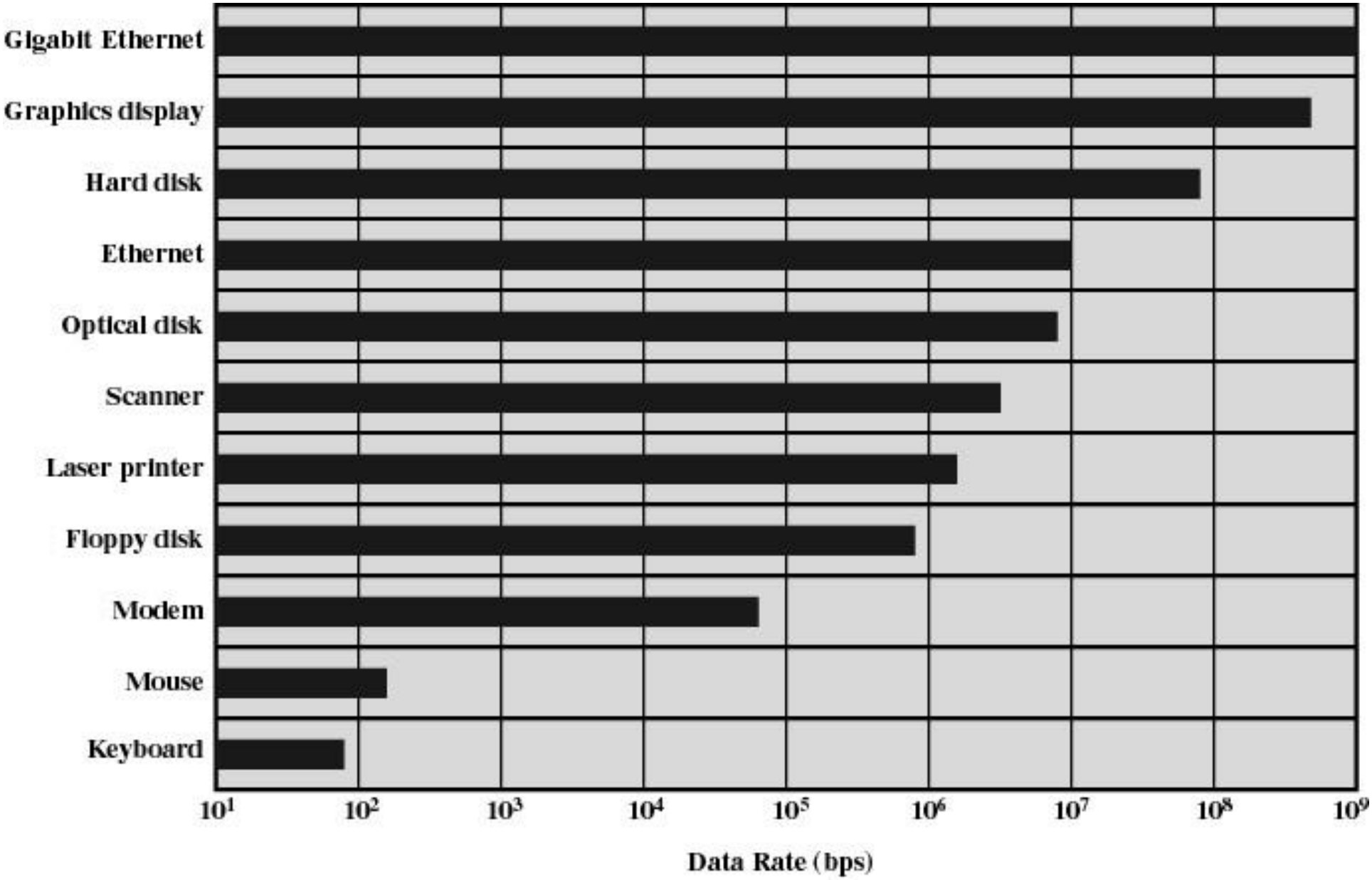


Figure 11.1 Typical I/O Device Data Rates

Differences in I/O Devices

- Complexity of control
- Unit of transfer
  - data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
- Data representation
  - encoding schemes
- Error conditions
  - devices respond to errors differently

Techniques for I/O function

- Programmed I/O
  - process is busy-waiting for the operation to complete
- Interrupt-driven I/O
  - I/O command is issued
  - processor continues executing instructions
  - I/O module sends an interrupt when done

Techniques for I/O function

- Direct Memory Access (DMA)
  - DMA module controls exchange of data between main memory and the I/O device
  - processor interrupted only after entire block has been transferred

Evolution of the I/O Function

- Processor directly controls a peripheral device
  - simple microprocessor-controlled devices
- Controller or I/O module is added
  - processor uses programmed I/O without interrupts
  - processor does not need to handle details of external devices

Evolution of the I/O Function

- Controller or I/O module with interrupts
  - processor does not spend time waiting for an I/O operation to be performed
- Direct Memory Access
  - blocks of data are moved into memory without involving the processor
  - processor involved at beginning and end only

Evolution of the I/O Function

- I/O module is a separate processor
  - I/O channel
  - access main memory for instructions
- I/O processor with its own memory
  - it is a computer in its own right
  - a large set of I/O devices can be controlled
  - a common use is to control communications with interactive terminals

Direct Memory Access

- Takes over control of the system from the CPU to transfer data to and from memory over the system bus
- Cycle stealing is commonly used to transfer data on the system bus
  - processor is suspended just before it needs to use the bus
  - DMA transfers one word and returns control to the processor
    - processor pauses for one bus cycle

How DMA Works?

- Processor sends the following information
  - read or write?
  - address of I/O device involved
  - starting address in memory
  - number of words
- DMA transfers the entire block
- DMA sends an interrupt signal when done

DMA

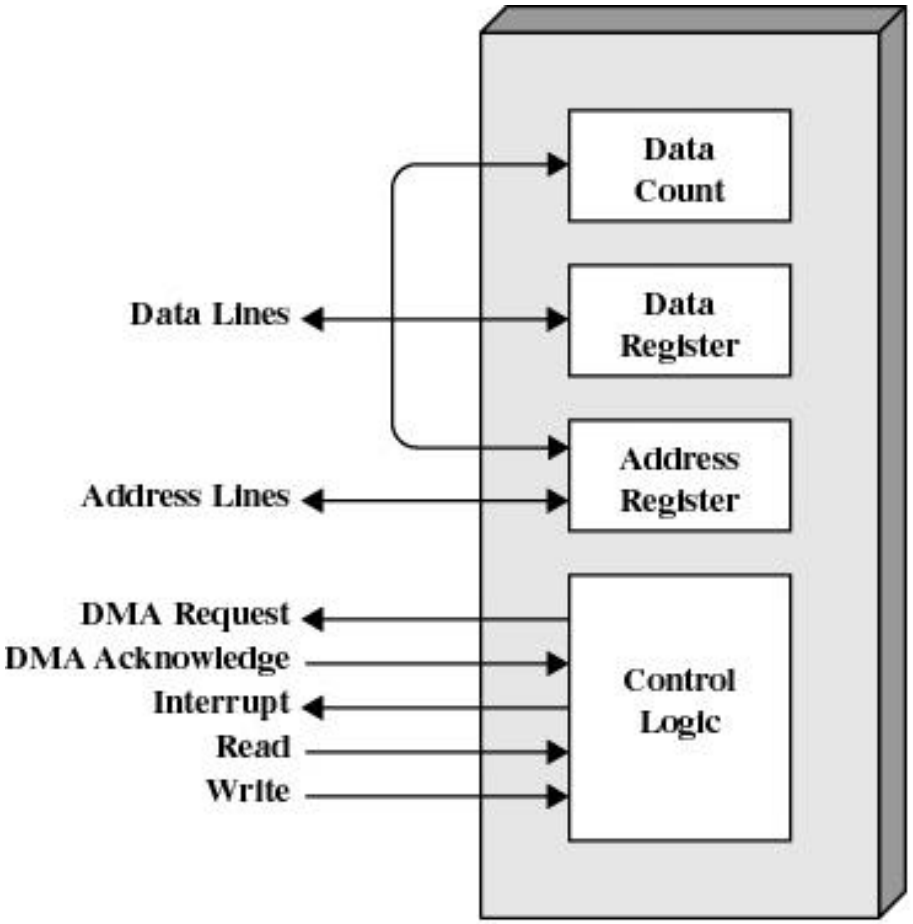


Figure 11.2 Typical DMA Block Diagram

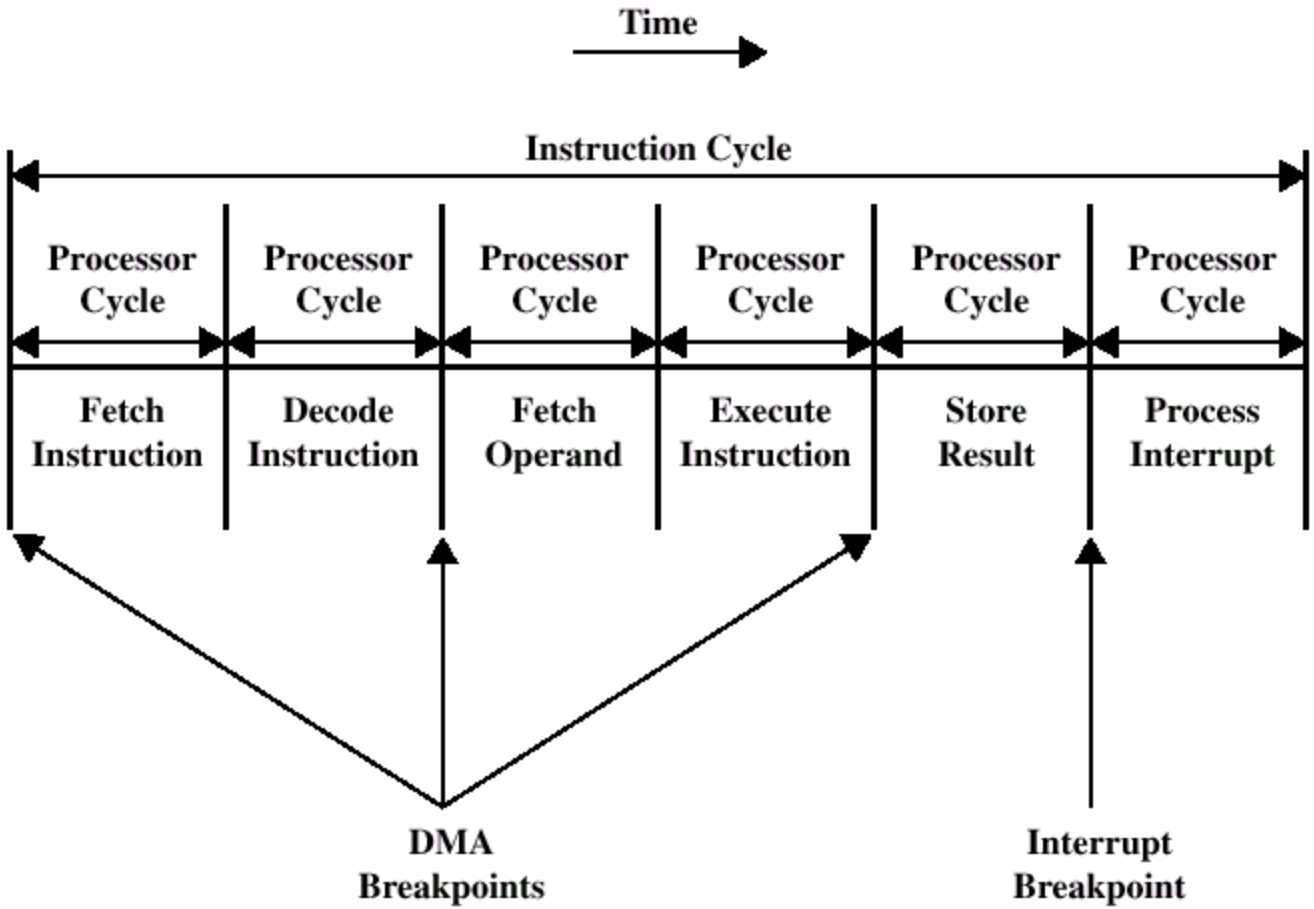
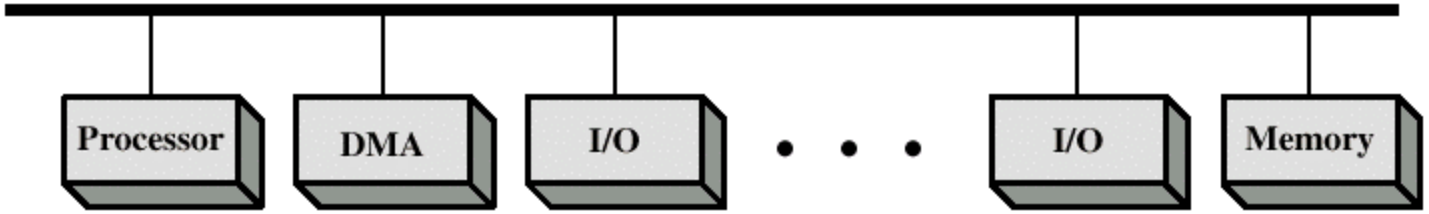


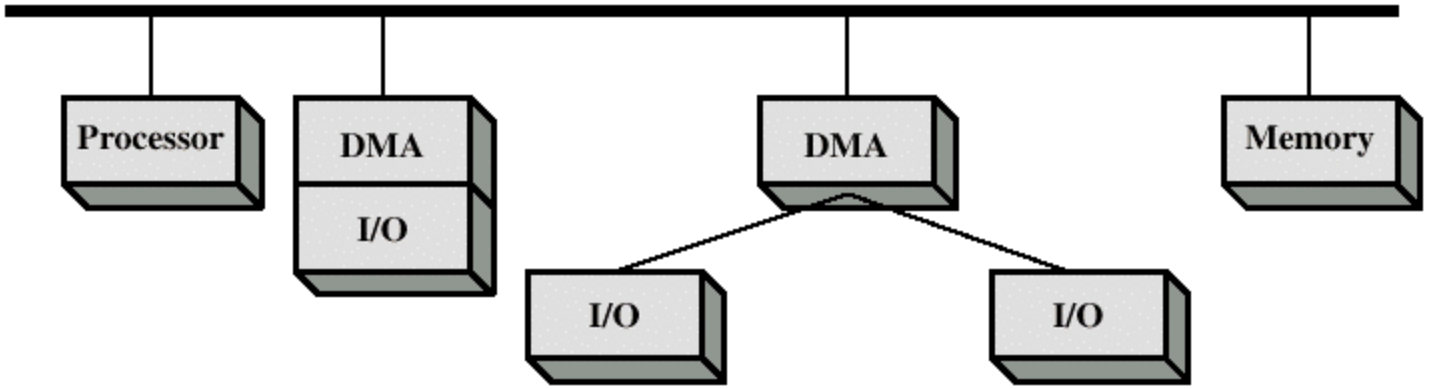
Figure 11.3 DMA and Interrupt Breakpoints During an Instruction Cycle

DMA Configurations

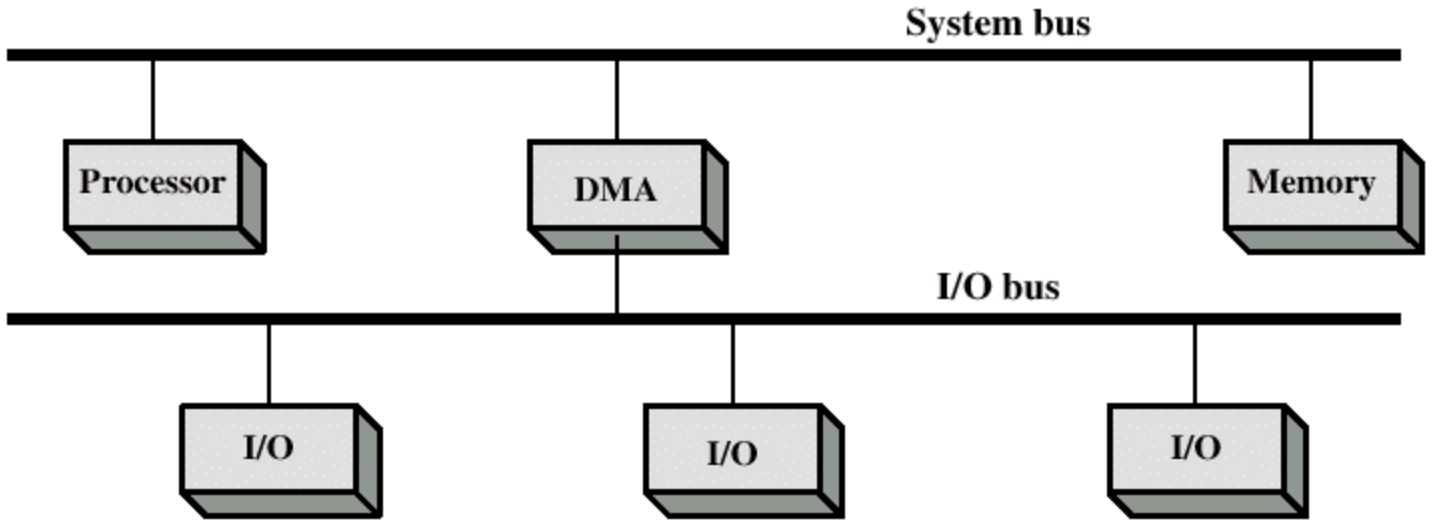
- Single bus, detached DMA
  - all modules share the same system bus
  - inexpensive, but inefficient
- Single bus, integrated DMA-I/O
  - there is a separate path between DMA and I/O modules
- I/O bus
  - only one interface between DMA and I/O modules
  - provides for an easily expandable configuration



(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-I/O



(c) I/O bus

Figure 11.4 Alternative DMA Configurations

OS Design Objectives

- Efficiency
  - Extremely slow compared with CPU
  - Use of multiprogramming allows for some processes to be waiting on I/O while another process executes
  - Swapping is used to bring in additional Ready processes to keep the processor busy
  - Disk I/O is important to improve the efficiency of the I/O

OS Design Objectives

- Generality
  - Desirable to handle all I/O devices in a uniform manner
  - Hide most of the details of device I/O in lower-level routines so that processes see devices in terms of general functions such as Read, Write, Open, and Close

I/O Buffering

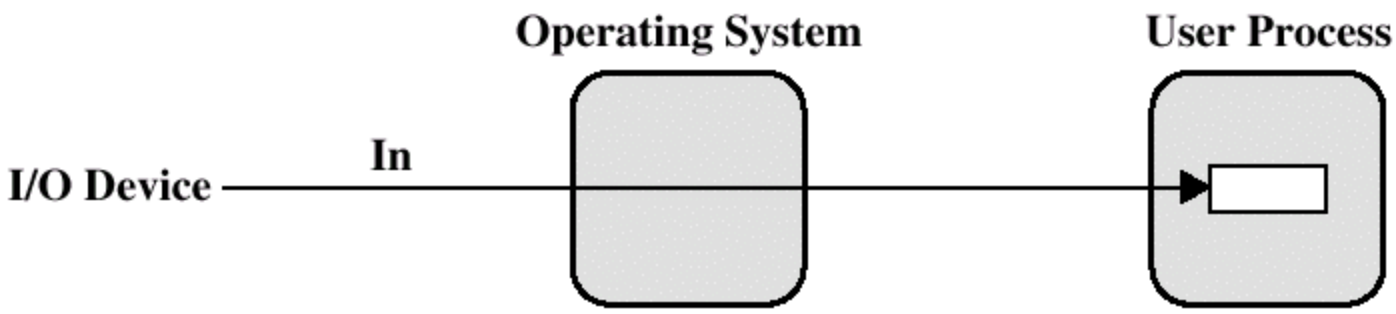
- Perform input transfers in advance of requests and perform output transfers sometime after the request is made
- Schemes
  - single buffering
  - double buffering
  - circular buffering

I/O Buffering

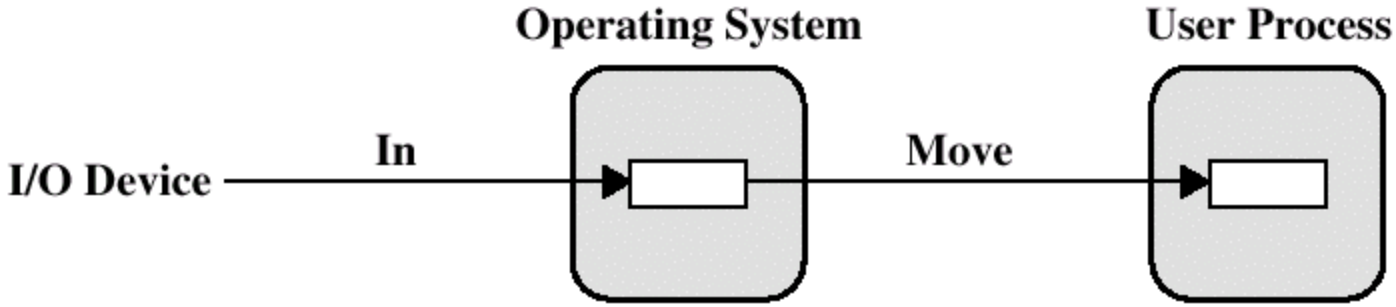
- Types of I/O devices
  - Block-oriented devices
    - information is stored in fixed sized blocks
    - transfers are made a block at a time
    - used for disks and tapes
  - Stream-oriented devices
    - transfer information as a stream of bytes
    - used for terminals, printers, communication ports, mouse, and most other devices that are not secondary storage

Single Buffer

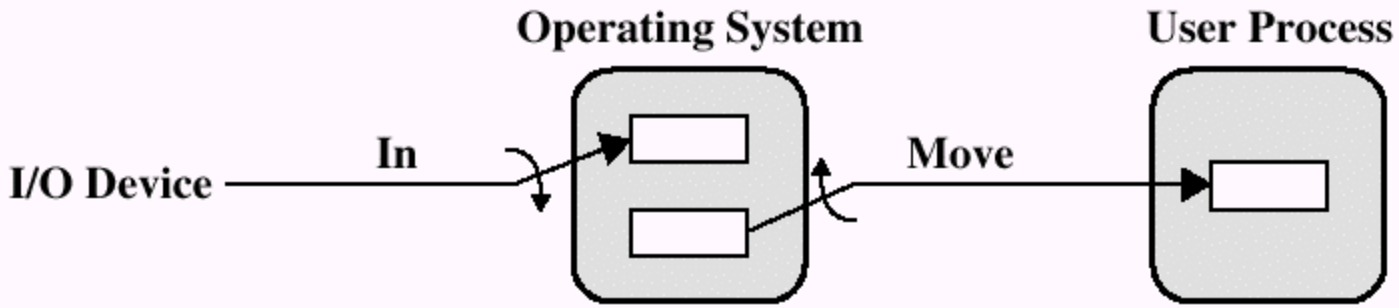
- OS assigns a buffer in main memory for an I/O request
- Block-oriented
  - input transfers made to buffer
  - block moved to user space when needed
  - another block is moved into the buffer
    - read ahead



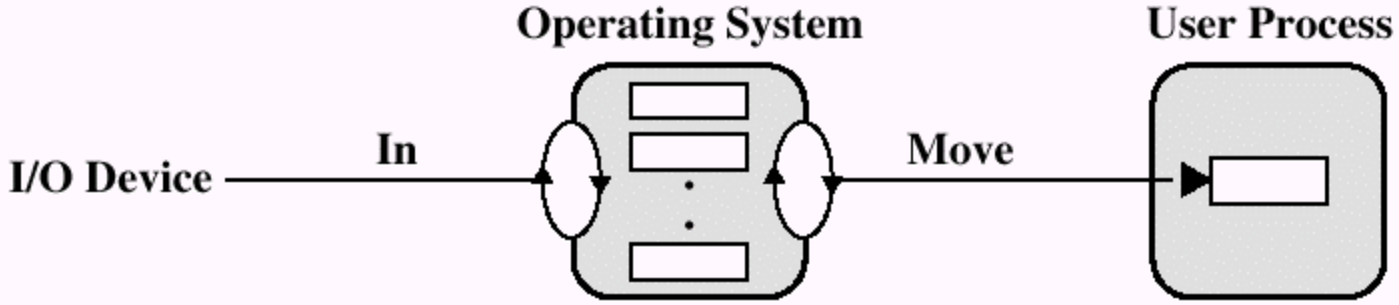
(a) No buffering



(b) Single buffering



(c) Double buffering



(d) Circular buffering

Figure 11.6 I/O Buffering Schemes (input)

Single Buffer

- Block-oriented
  - user process can process one block of data while next block is read in
  - swapping can occur since input is taking place in system memory, not user memory
  - operating system keeps track of assignment of system buffers to user processes
  - output is accomplished by the user process writing a block to the buffer and later actually written out

Single Buffer

- Stream-oriented
  - used a line at time
  - user input from a terminal is one line at a time with carriage return signaling the end of the line
  - output to the terminal is one line at a time

Double Buffer

- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer

Circular Buffer

- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process

Disk Performance Parameters

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector
- Disk I/O time
  - *queuing time* (wait for device) +
  - *channel waiting time* (wait for channel) +
  - *seek time* +
  - *rotational delay (latency)* +
  - *data transfer time*

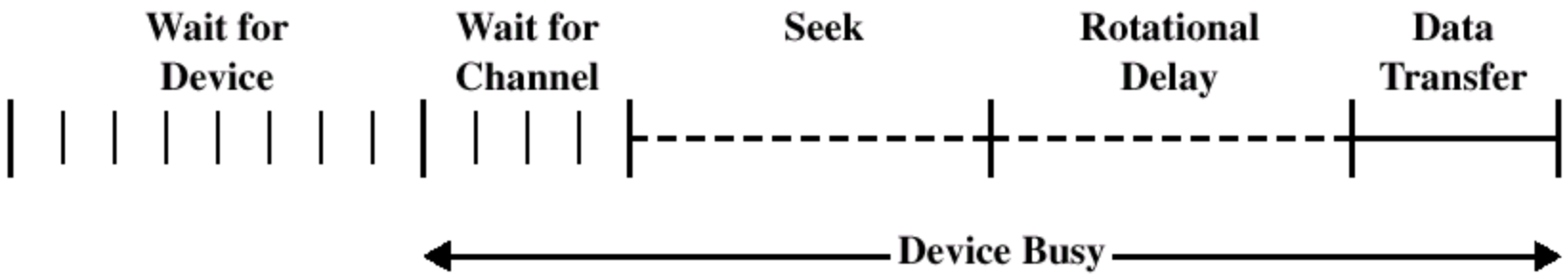


Figure 11.7 Timing of a Disk I/O Transfer

Disk Performance Parameters

- Seek time
  - time it takes to position the head at the desired track
  - $T_s = m \ ? \ n + s$
    - T_s = estimated seek time, n = number of tracks traversed, m = constant that depends on the disk drive, s = startup time
    - inexpensive disk : m = 0.3, s = 20ms
    - expensive disk : m = 0.1, s = 3ms

Disk Performance Parameters

- Rotational delay or rotational latency
  - time its takes until desired sector is rotated to line up with the head
  - $T_r = 1 / (2r)$
    - T_r = average rotational delay time, r = rotation speed in revolutions per second
    - disk : 3600 rpm, 16.7 ms/rot, T_r = 8.3 ms ;
    - floppy : 300~600 rpm, 100~200 ms/rot, T_r = 50~100 ms

Disk Performance Parameters

- Transfer time
  - time its takes while desired sector moves under the head
  - $T_t = b / (rN)$
    - T_t = transfer time
    - b = number of bytes to be transferred
    - N = number of bytes on a track
    - r = rotation speed in revolutions per second

Disk Performance Parameters

- Access time
  - sum of seek time and rotational delay
  - the time it takes to get in position to read or write
- Total Access Time
  - $T_a = T_s + T_r + T_t = T_s + 1/(2r) + b/(rN)$

Timing Comparison

- Data transfer occurs as the sector moves under the head
- Data transfer for an entire file is faster when the file is stored in the same cylinder and in adjacent sectors
- Illustrate the danger of relying on average values

Timing Comparison

- Disk specification
  - average seek time : 20 ms
  - a transfer rate : 1MB/s
  - 512-byte sectors with 32 sectors per track
  - suppose reading a file consisting 2048 sectors for a total of 1MB : 64 tracks ? 32 sectors/track = 2048 sectors
  - $T_t = b / (rN) = 1024 * 1024 / (r * 512 * 32)$
  - ? $r = 64$

Timing Comparison

- Sequential organization (single surface)
  - first track = 45 ms = 20 ms (average seek) + 8.3 ms (rotational delay) + 16.7 ms (read 32 sectors)
  - each succeeding 63 tracks = 25 ms = 8.3 ms (rotational delay) + 16.7 ms (read 32 sectors)
  - total time = 45 ms + 63 tracks * 25 ms = 1620 ms = 1.6 sec

Timing Comparison

- Random access (single surface)
  - each sector = 28.8 ms = 20 ms (average seek) + 8.3 ms (rotational delay) + 0.5 ms (read 1 sector)
  - total time = 2048 sectors * 28.8 ms = 58982 ms = 59 sec

Timing Comparison

- Sequential parallel access
  - stored on same cylinders and same tracks
  - disk consists of 8 surfaces
  - first cylinder(8 tracks) = 45 ms = 20 ms (average seek) + 8.3 ms (rotational delay) + 16.7 ms (read 32 sectors)
  - each succeeding 7 cylinders = 25ms = 8.3ms (rotational delay) + 16.7 ms (read 32 sectors)
  - total time = 45ms + 7 * 25ms = 220 ms

Disk Scheduling Policies

- Seek time is the reason for the difference in performance
  - need to reduce the average seek time
  - OS maintains a queue of requests for each I/O devices
    - For a single disk there will be a number of I/O requests from processes in the queue

Disk Scheduling Policies

- Assume a disk with 200 tracks
- Starting at track 100 in the direction of increasing track number
- Requested tracks in the order received : 55, 58, 39, 18, 90, 160, 150, 38, 184

Disk Scheduling Policies

- Random scheduling
- First-in, first-out (FIFO)
- Priority
- Last-in, first-out
- Shortest Service Time First
- SCAN
- C-SCAN
- N-step-SCAN
- FSCAN

Disk Scheduling Policies

- Random scheduling
  - select requests from queue in random order
  - the worst possible performance
  - useful as a benchmark against which to evaluate other techniques

Disk Scheduling Policies

- First-in, first-out (FIFO)
  - process request sequentially
  - fair to all processes
  - if there are only a few processes that require access and if many of the requests are to clustered, good performance can be hoped
  - approaches random scheduling in performance if there are many processes

Disk Scheduling Policies

- Priority
  - goal is not to optimize disk use but to meet other objectives
  - short batch jobs may have higher priority
    - provide good interactive response time
  - longer jobs may have to wait long

Disk Scheduling Policies

- Last-in, first-out
  - good for transaction processing systems
    - the device is given to the most recent user so there should be little arm movement
      - improve throughput and reduce queue length
  - possibility of starvation since a job may never regain the head of the line

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 to incur the minimum Seek time

Disk Scheduling Policies

- SCAN
  - arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
  - direction is reversed

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

Disk Scheduling Policies

- N-step-SCAN
  - segments the disk request queue into sub-queues of length N
  - sub-queues are processed one at a time, using SCAN
  - new requests added to other queue when queue is processed
  - with large value of N, this is similar to SCAN
  - with N = 1, this is same as FIFO

Disk Scheduling Policies

- FSCAN
  - two queues
  - when a scan begins, all of the requests are in one of the queues, with the other empty
  - during the scan, all new requests are put into the other queue

Table 11.3 Disk Scheduling Algorithms [WIED87]

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

RAID(Redundant Array of Independent Disks)

- Why RAID?
  - big speed gap between CPU and disk
  - why not having a parallelism in disks?
    - Redundant Array of Independent Disks
  - RAID
    - many SCSI disks with RAID SCSI controller
    - organizations defined by Patterson et al.
      - level 0 through level 6
  - SLED(Single Large Expensive Disk)

RAID(Redundant Array of Independent Disks)

- Array of disks that operate independently and in parallel
  - distribute the data on multiple disks
    - single I/O request can be executed in parallel
  - replaces large-capacity disk drives with multiple smaller-capacity drives
    - improves I/O performance and allows easier incremental increases in capacity

Characteristics of RAID

- RAID is a set of physical disk drives viewed by OS as a single logical drive
- Data are distributed across the physical drives of an array
- Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure

Table 11.4 RAID levels

Category	Level	Description	I/O Request Rate (Read/Write)	Data Transfer Rate (Read/Write)	Typical Application
Striping	0	Nonredundant	Large strips: Excellent	Small strips: Excellent	Applications requiring high performance for noncritical data
Mirroring	1	Mirrored	Good/Fair	Fair/Fair	System drives; critical files
Parallel access	2	Redundant via Hamming code	Poor	Excellent	
	3	Bit-interleaved parity	Poor	Excellent	Large I/O request size applications, such as imaging, CAD
Independent access	4	Block-interleaved parity	Excellent/Fair	Fair/Poor	
	5	Block-interleaved distributed parity	Excellent/Fair	Fair/Poor	High request rate, read-intensive, data lookup
	6	Block-interleaved dual distributed parity	Excellent/Poor	Fair/Poor	Applications requiring extremely high availablity

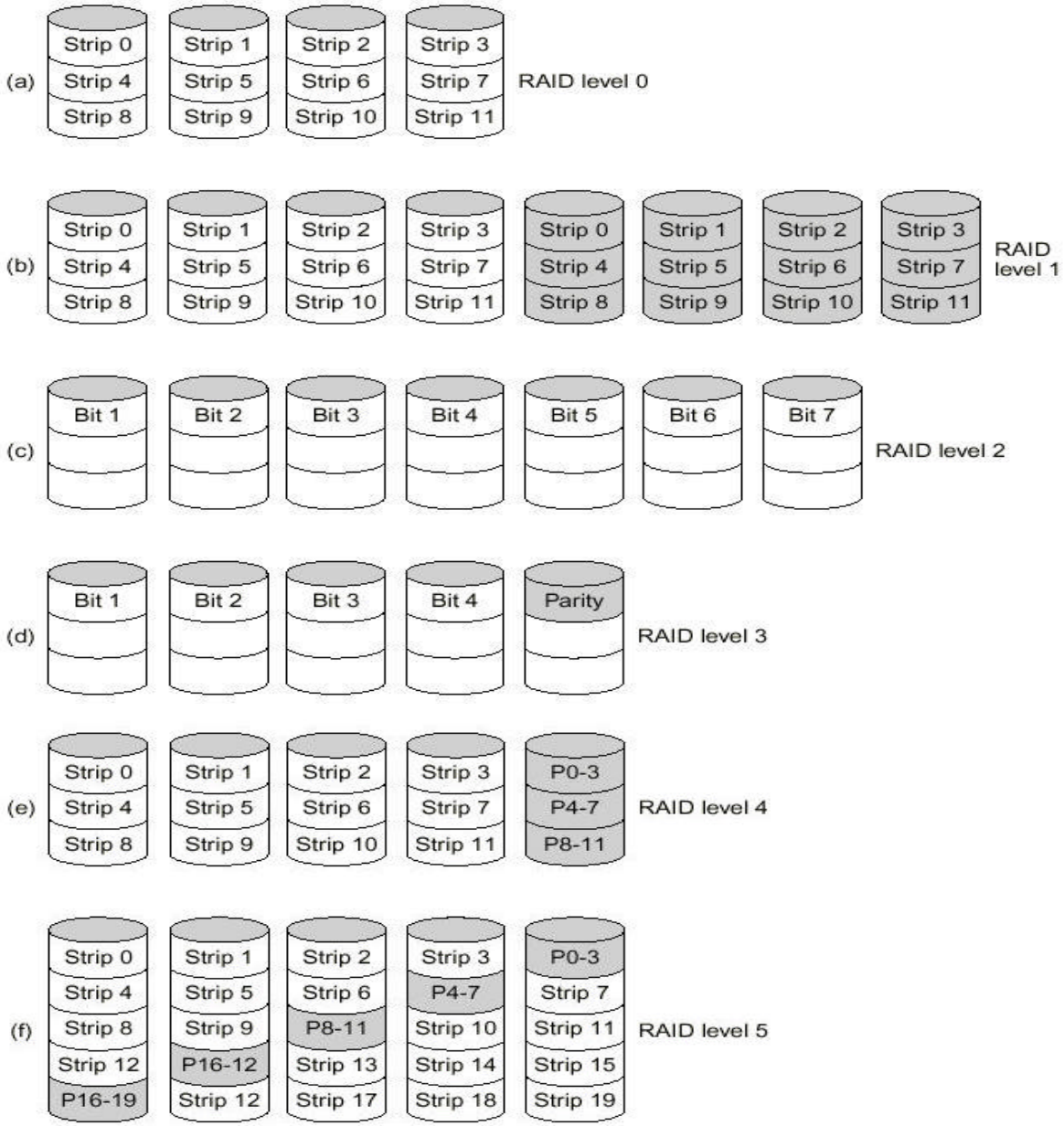
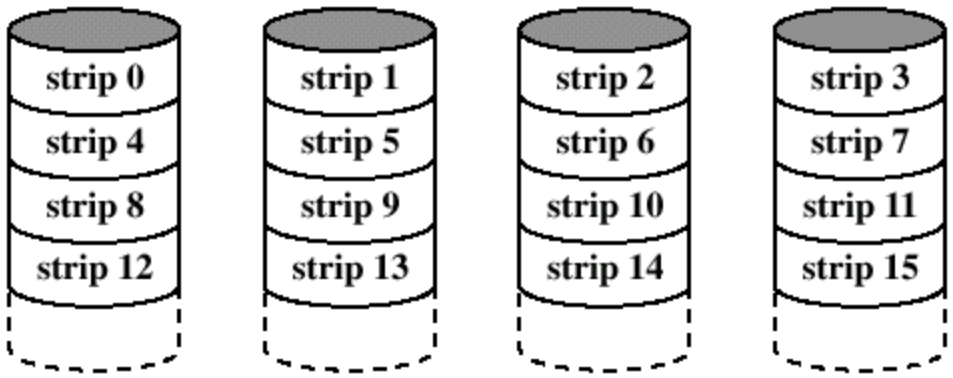


Figure 2-23. RAID levels 0 through 5. Backup and parity drives are shown shaded.

RAID 0 (non-redundant)

- a *logical disk* is divided into strips
- *strips* : physical blocks, sectors, or some other unit
- writes consecutive strips over the drives in round robin way
- a *stripe* : a set of logically consecutive strips that maps exactly one strip to each array member

RAID 0 (non-redundant)



(a) RAID 0 (non-redundant)

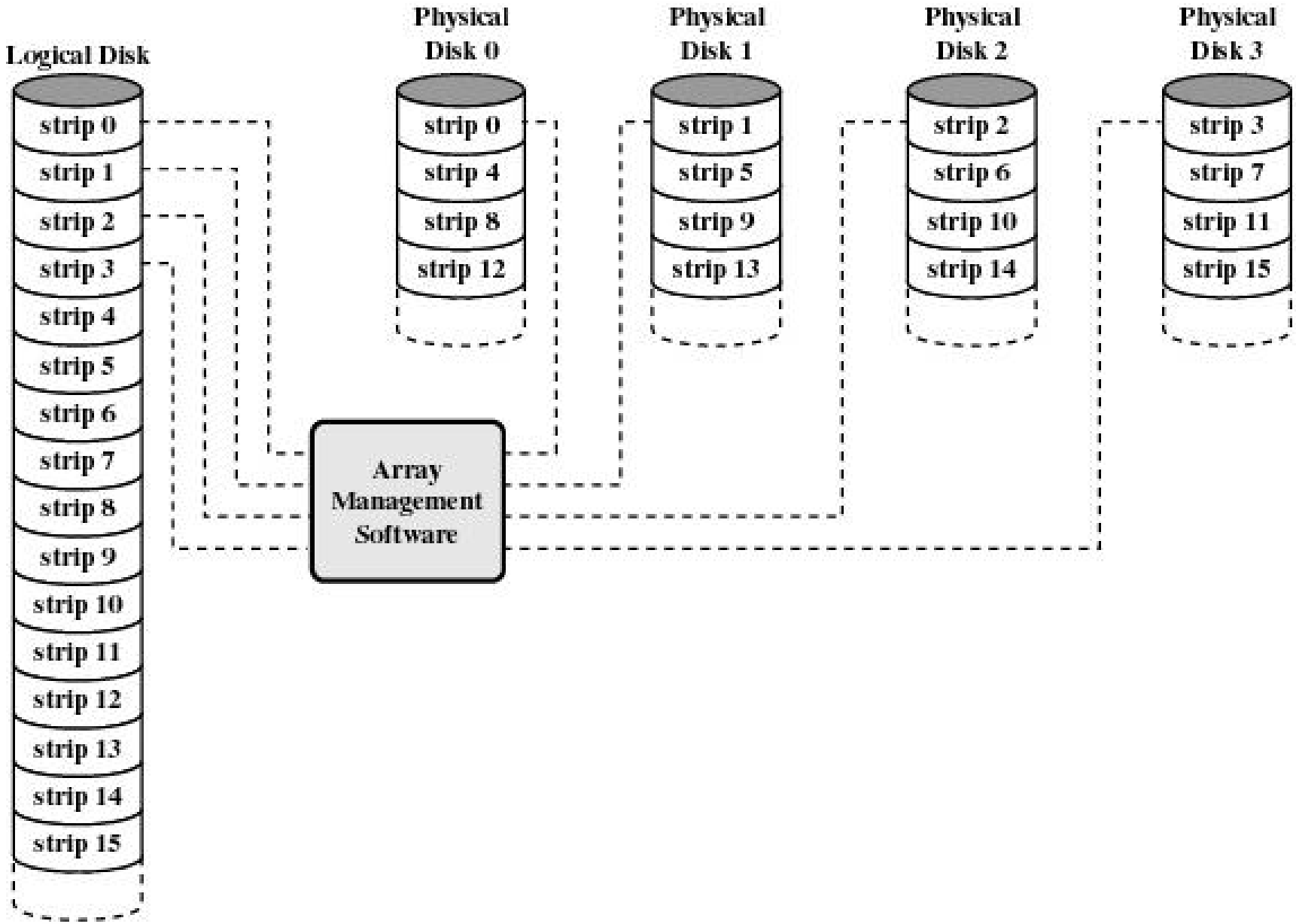
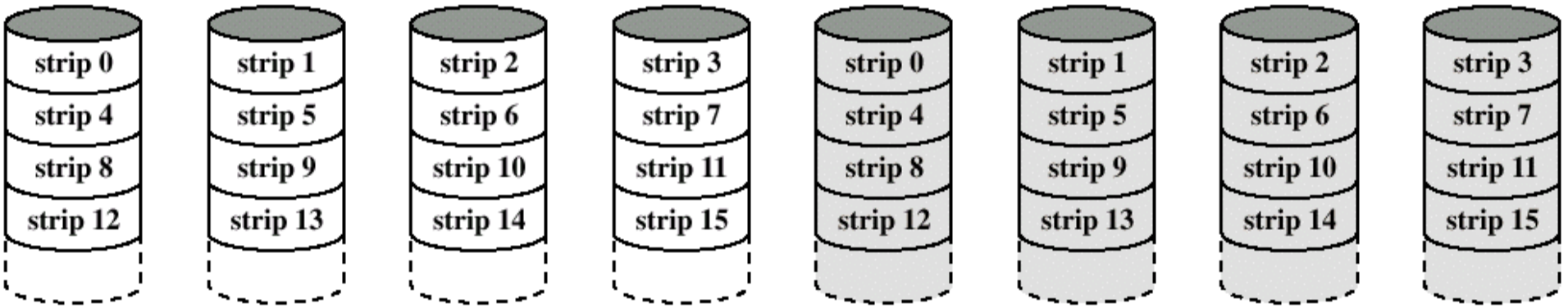


Figure 11.10 Data Mapping for a RAID Level 0 Array [MASS97]

RAID 1 (mirrored)

- on a write, every strip is written twice
- on a read, either copy can be used
  - read performance can be up to twice as good
- fault-tolerance is excellent

RAID 1 (mirrored)

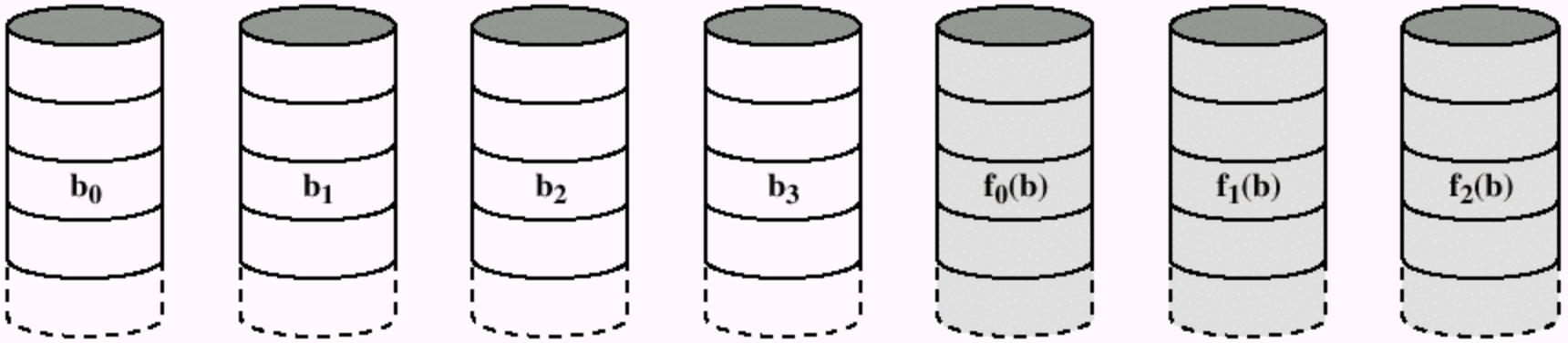


(b) RAID 1 (mirrored)

RAID 2 (redundancy through Hamming code)

- works on a word basis + Hamming code
  - splitting each byte into a pair of 4-bit nibbles
  - adding 3 parity bits
  - 7 bits are spread over the 7 drives
- performance is good
  - in one sector time, it could write 4 sectors
  - losing one drive do not cause any problem
- all the drives must be synchronized
  - on a single write, all data disks and parity disks must be accessed

RAID 2 (redundancy through Hamming code)



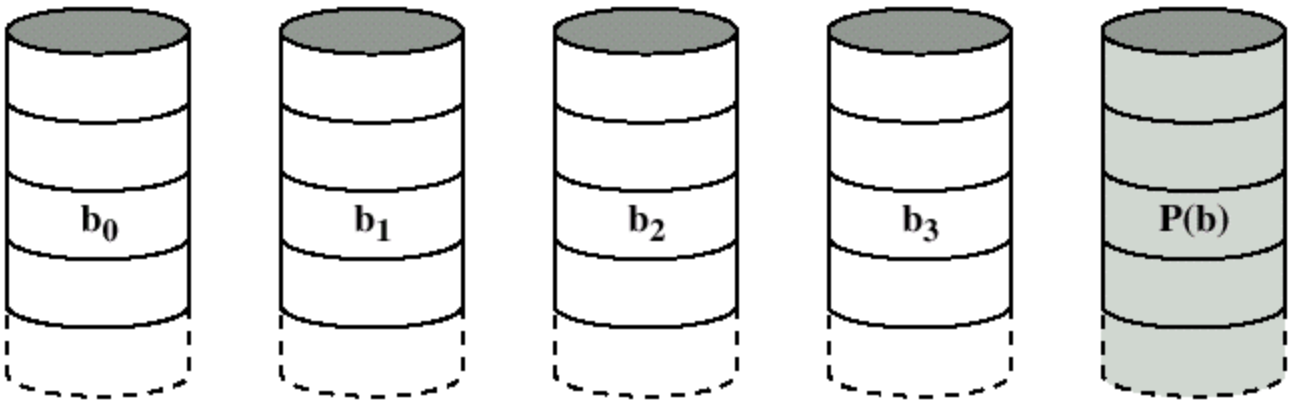
(c) RAID 2 (redundancy through Hamming code)

Figure 11.9 RAID Levels (page 1 of 2)

RAID 3 (bit-interleaved parity)

- similar to RAID 2 : requires only single redundant disk
- a parity bit is generated by exclusive-or of across corresponding bits on each data disk
  - $X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$
  - $X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$

RAID 3 (bit-interleaved parity)



(d) RAID 3 (bit-interleaved parity)

RAID 4 (block-level parity)

- work with strips with a strip-for-strip parity written onto an extra drive
  - all the strips are EXCLUSIVE ORed together
- if a drive crashes, the lost bytes can be recomputed from the parity drive
- performs poorly for small updates
  - need to recalculate the parity every time
- parity drive may become a bottleneck

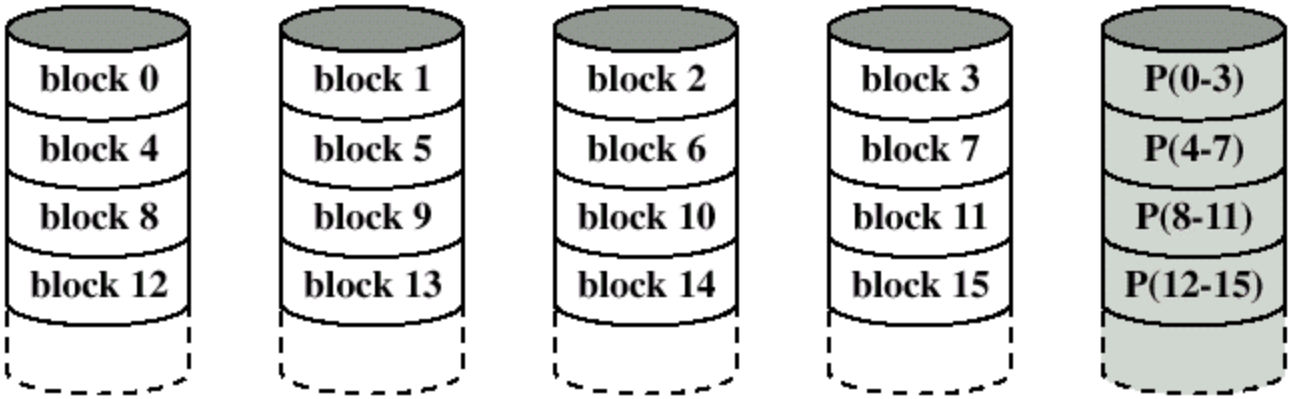
RAID 4 (block-level parity)

- use an independent access technique
  - $X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$
  - $X4'(i) = X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i)$

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

    $= X4(i) \oplus X1(i) \oplus X1'(i)$

RAID 4 (block-level parity)

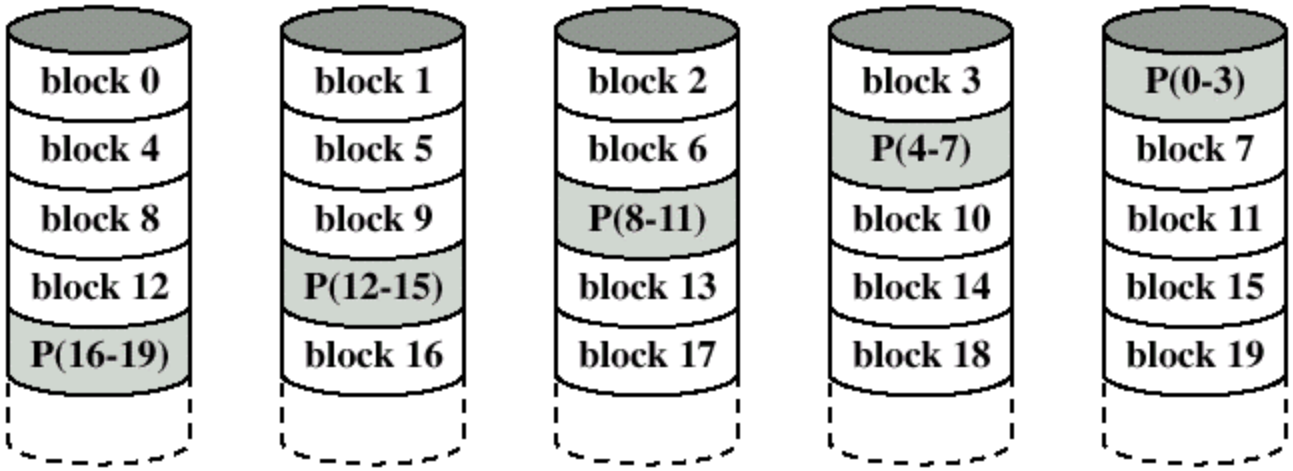


(e) RAID 4 (block-level parity)

RAID 5 (block-level distributed parity)

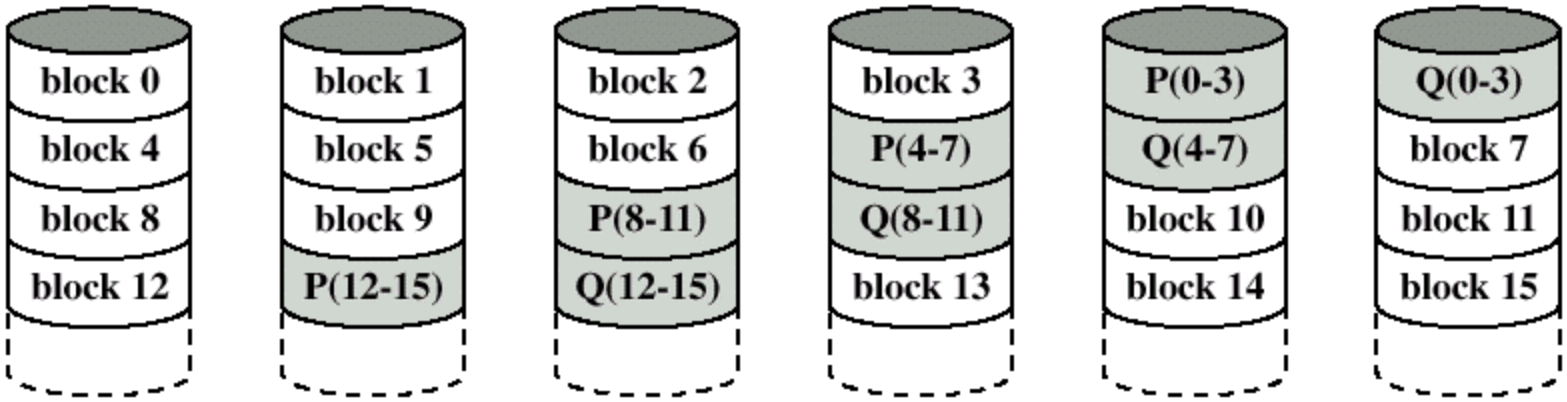
- distributing the parity bits uniformly over all the drives

RAID 5 (block-level distributed parity)



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

RAID 6 (dual redundancy)



(g) RAID 6 (dual redundancy)

Figure 11.9 RAID Levels (page 2 of 2)

Disk Cache

- Buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk
- When an I/O request is made, a check is made to determine if the sector is in the disk cache

Replacement Policies

- When a new sector is brought into the disk cache, one of the existing blocks must be replaced
  - Least Recently Used
  - Least Frequently Used

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
  - When a block is referenced or brought into the cache, it is placed on the top of the stack
  - Most recently referenced block is on the top of the stack
  - 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 then not needed any more
    - frequency-based replacement technique

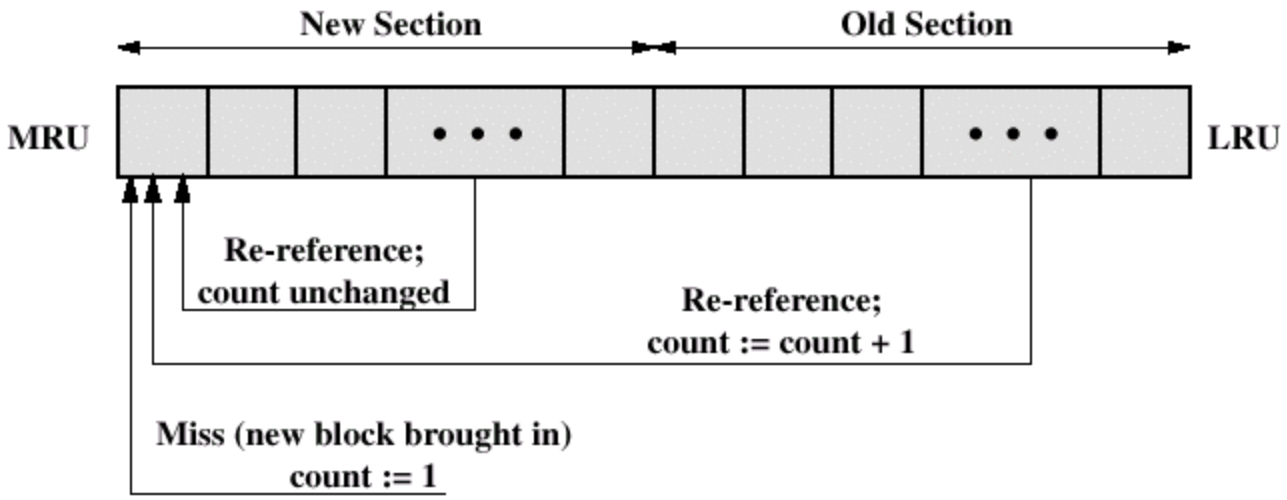
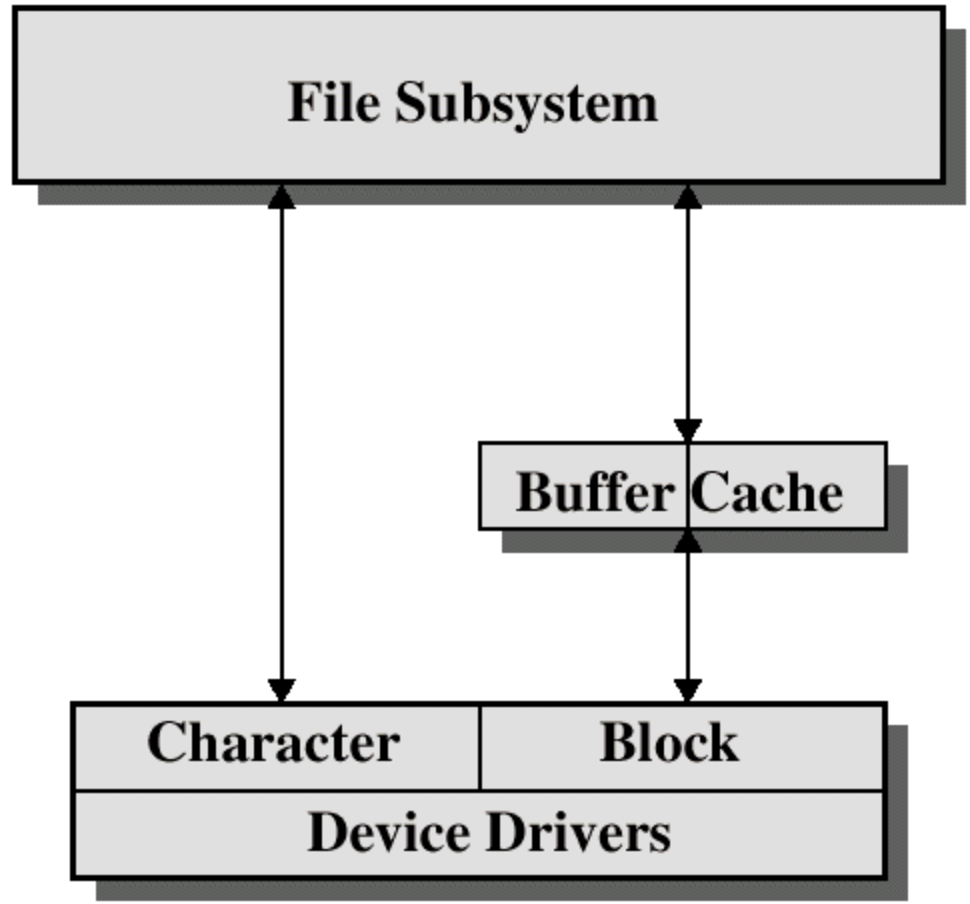


Figure 11.11 Frequency-Based Replacement

UNIX SVR4 I/O Structure



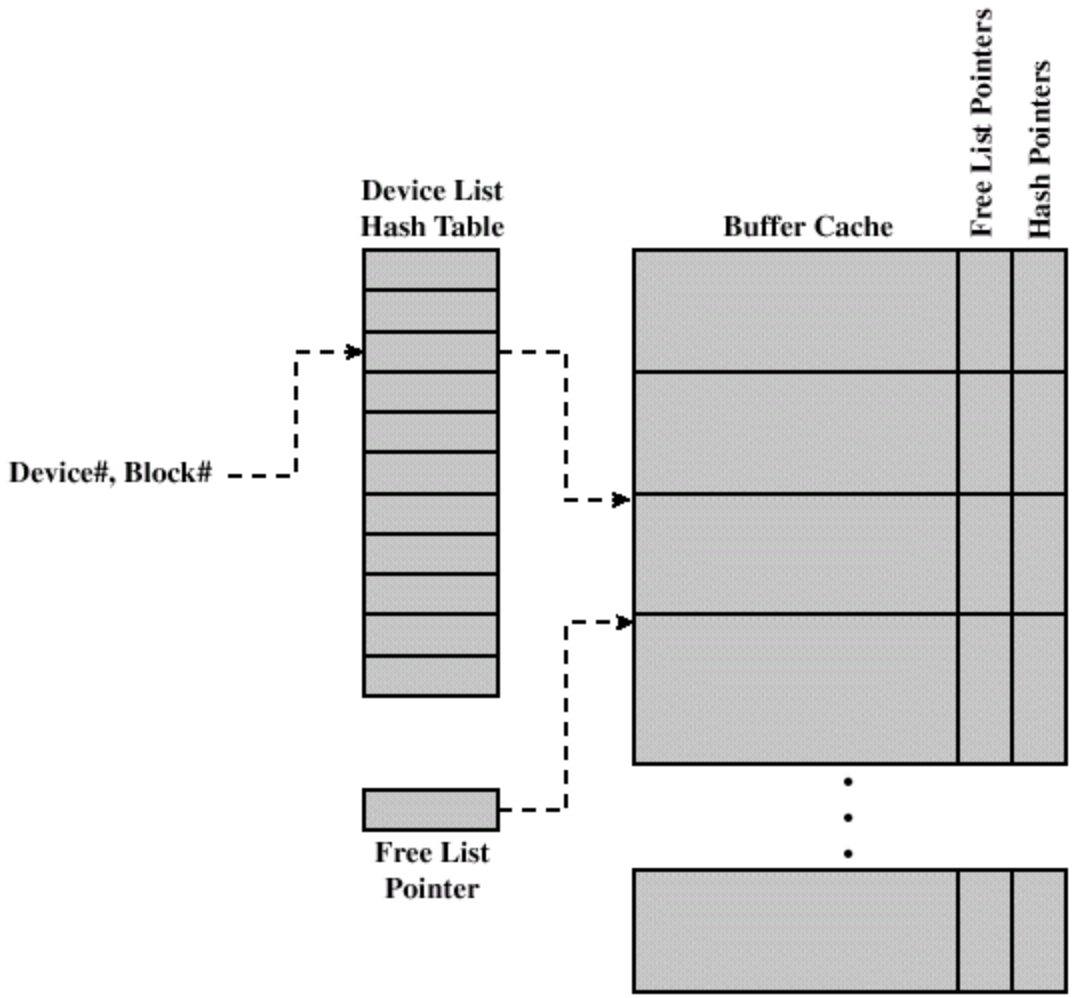


Figure 11.15 Unix buffer cache organization

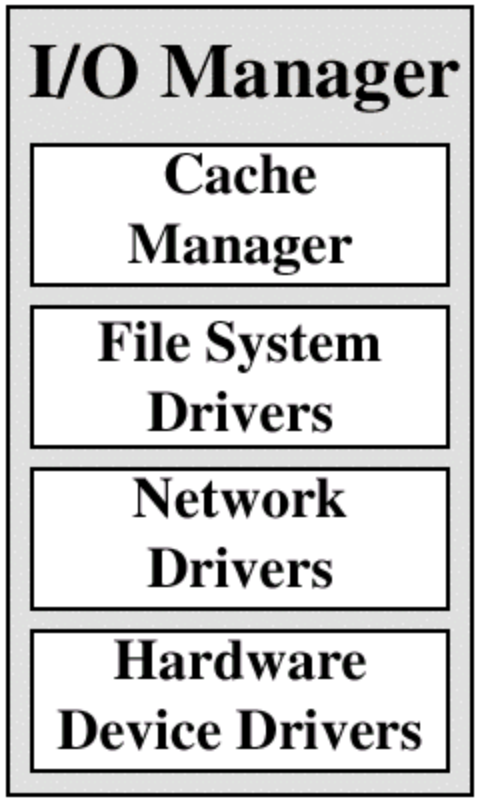
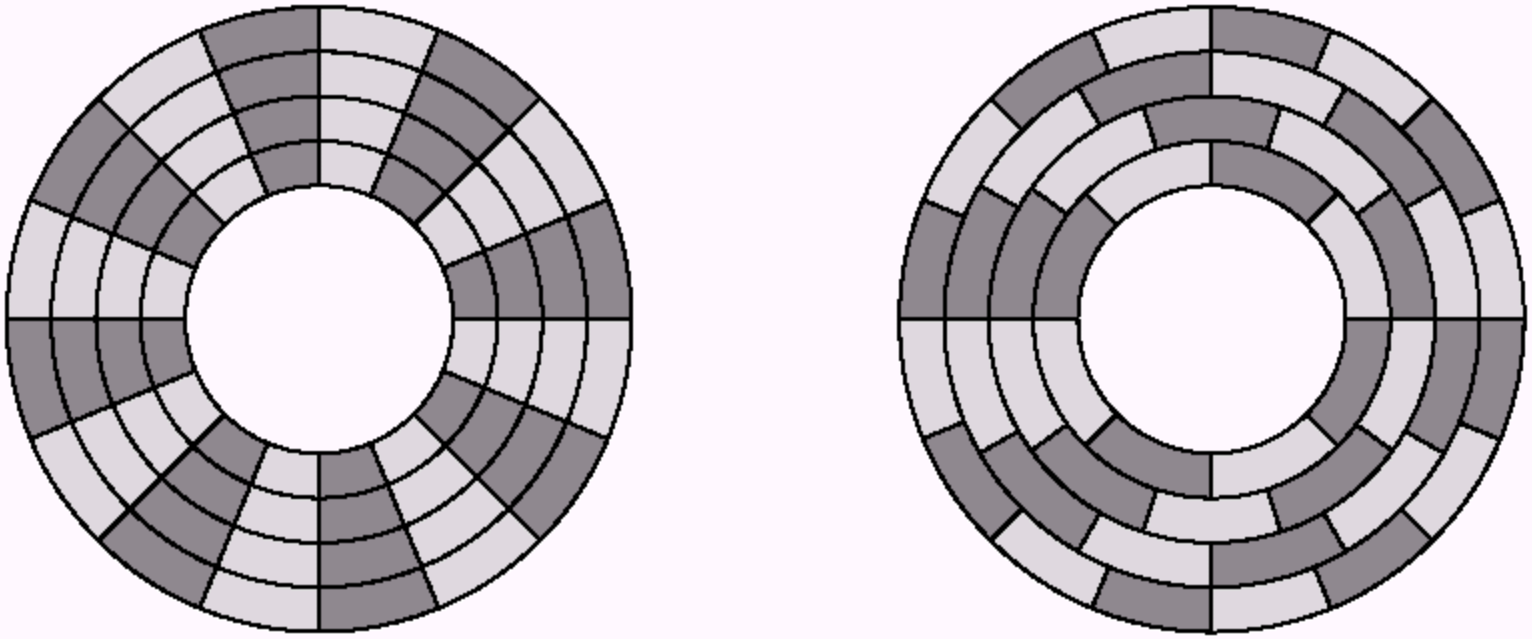


Figure 11.16 Windows 2000 I/O Manager



(a) Constant angular velocity

(b) Constant linear velocity

Figure 11.20 Comparison of Disk Layout Methods