I/O Management and Disk Scheduling

Chapter 11

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
  - actuators

Categories of I/O Devices

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

Differences in I/O Devices

- ✓ Data Transfer Rate
- ✓ Application-specific
  - disk used to store files must have file-management software
  - disk used to store virtual memory pages depends on virtual memory hardware; I/O ops may be scheduled differently than for disks used for file storage
  - terminal used by system administrator may have a higher priority

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: character encoding, parity may be different
- ✓ Error conditions
  - devices respond to errors differently

Techniques for Performing I/O

- ✓ 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 Performing I/O

- ✓ 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
- ✓ 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

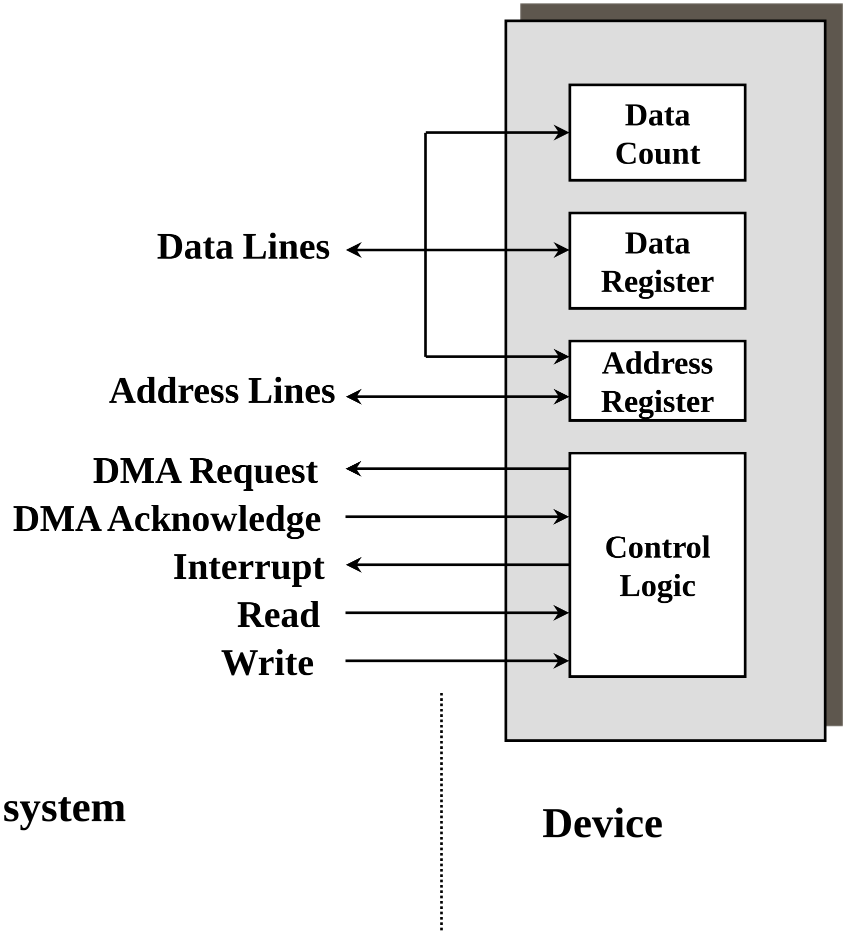
Evolution of the I/O Function

- ✓ I/O channel
  - I/O module is a separate processor
  - Uses computer’s main memory
- ✓ I/O processor
  - I/O module is a processor with its own local memory
  - It’s a computer in its own right

Direct Memory Access

- ✓ Takes control of the system form the CPU to transfer data to and from memory over the system bus
- ✓ Cycle stealing is used to transfer data on the system bus
- ✓ The instruction cycle is suspended so data can be transferred
- ✓ The CPU pauses one bus cycle
- ✓ No interrupts occur
  - does not need to save context

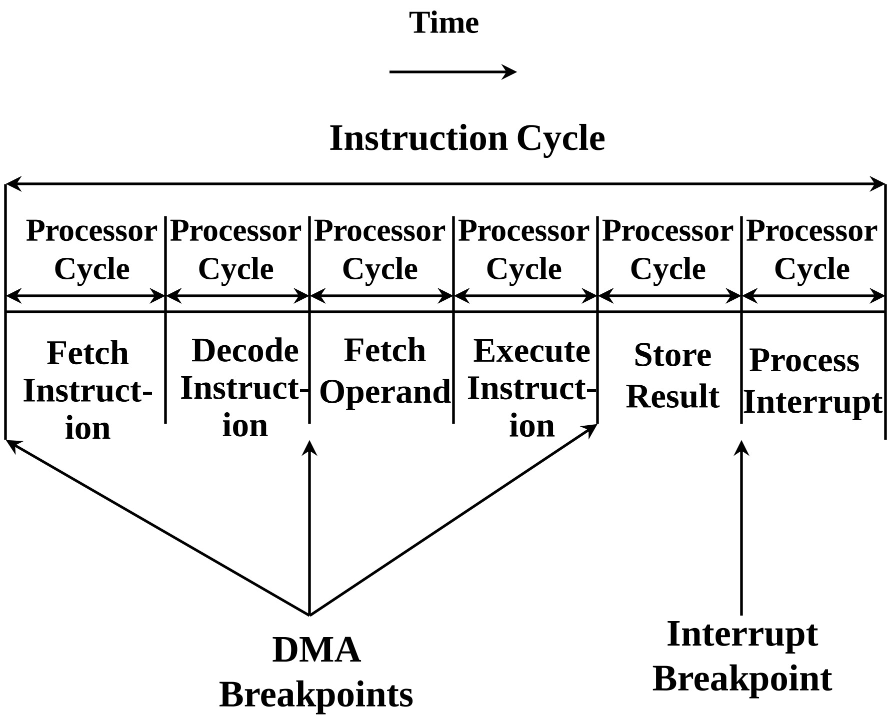
Typical DMA Block Diagram



Direct Memory Access

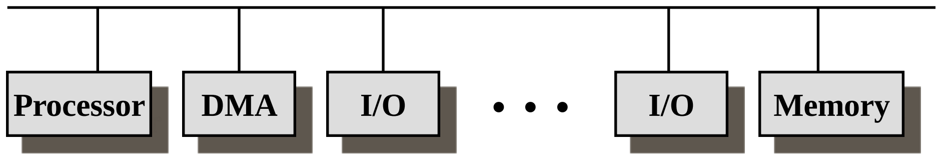
- ✓ Cycle stealing causes the CPU to execute more slowly
- ✓ Number of required busy cycles can be cut by integrating the DMA and I/O functions
- ✓ Try to use path between DMA module and I/O module that does not include the system bus

DMA and Interrupt Breakpoints

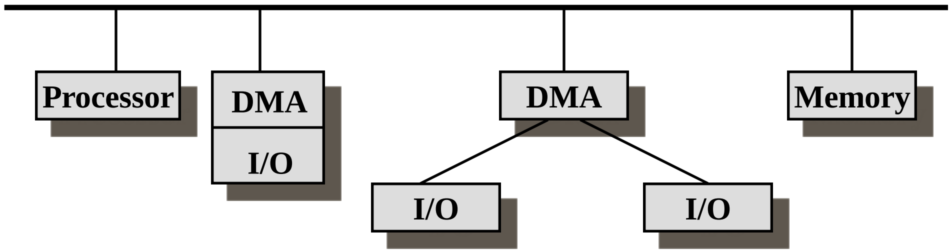


Breakpoints where CPU can be suspended to let the DMA module use the buss

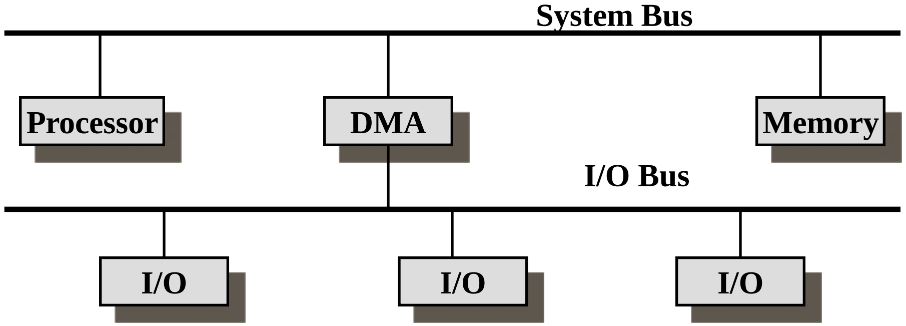
Single-bus, Detached DMA



Single-bus, Integrated DMA-I/O



I/O Bus



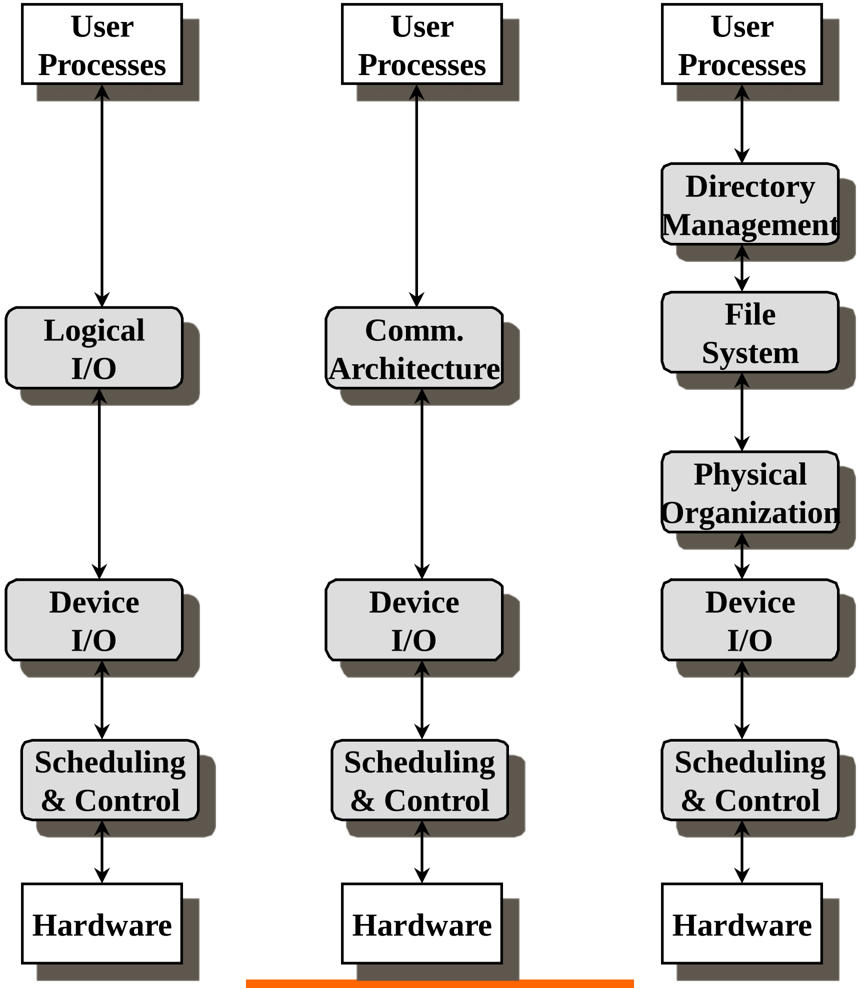
Operating System Design Objectives

- ✓ I/O is extremely slow compared to main memory
- ✓ Use of multiprogramming allows that some processes will be waiting on I/O while another process executes
- ✓ I/O cannot keep up with processor speed
- ✓ Swapping is used to bring in additional Ready processes, which is an I/O operation
- ✓ Efficiency of I/O is an important issue, since this is a bottleneck

Operating System Design Objectives

- ✓ 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 and upper levels see devices in general terms such as Read, Write, Open, and Close
- ✓ Generality is an important issue

A Model of I/O Organization



Local peripheral device

Communications port

Device with a File System

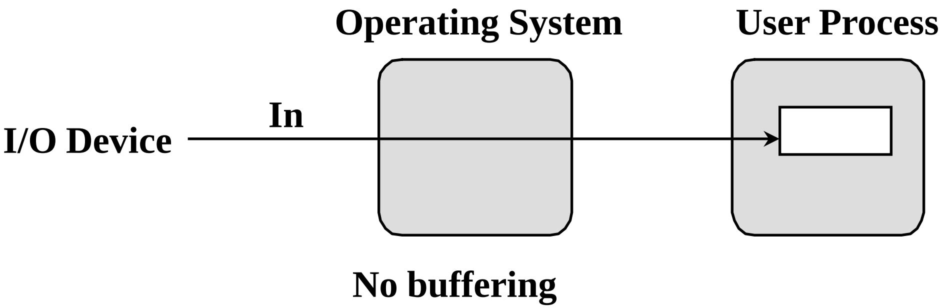
I/O Buffering

- ✓ Reasons for buffering: to find a solution to these problems:
  - Processes must wait for I/O to complete before proceeding
  - Certain pages must remain in main memory during I/O – interferes with page replacement

I/O Buffering

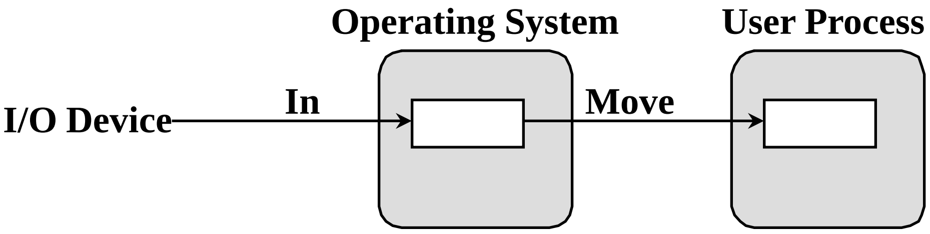
- ✓ Block-oriented
  - information is stored in fixed sized blocks
  - transfers are made a block at a time
  - used for disks and tapes
- ✓ Stream-oriented
  - transfer information as a stream of bytes
  - used for terminals, printers, communication ports, mouse, and most other devices that are not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

No Buffering



Single Buffer

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



Single buffering

Single Buffer

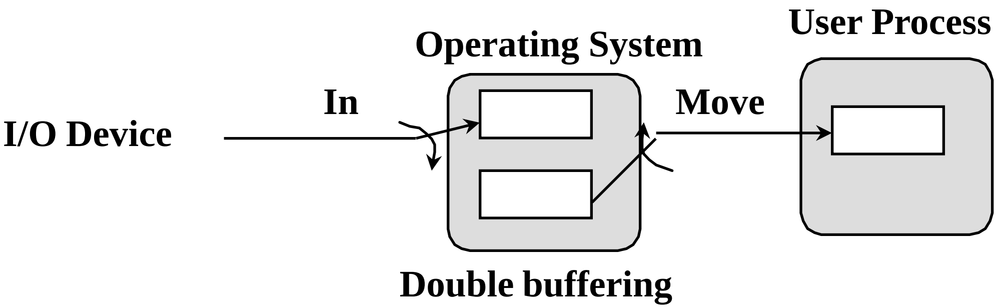
- ✓ *Block-oriented I/O:*
  - user process can work on one block of data while next block is being read in
  - process waiting for I/O can be swapped out, since input is taking place in system memory, not user memory
  - operating system keeps track of assignment of system buffers to user processes
  - output is accomplished by the user process writing a block to the buffer and later actually written out

Single Buffer

- *Stream-oriented:*
  - used one line at a 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

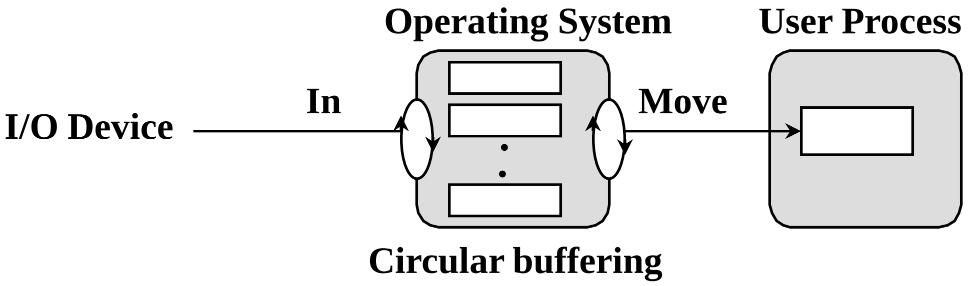
- ✓ Use two system buffers instead of one
- ✓ A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



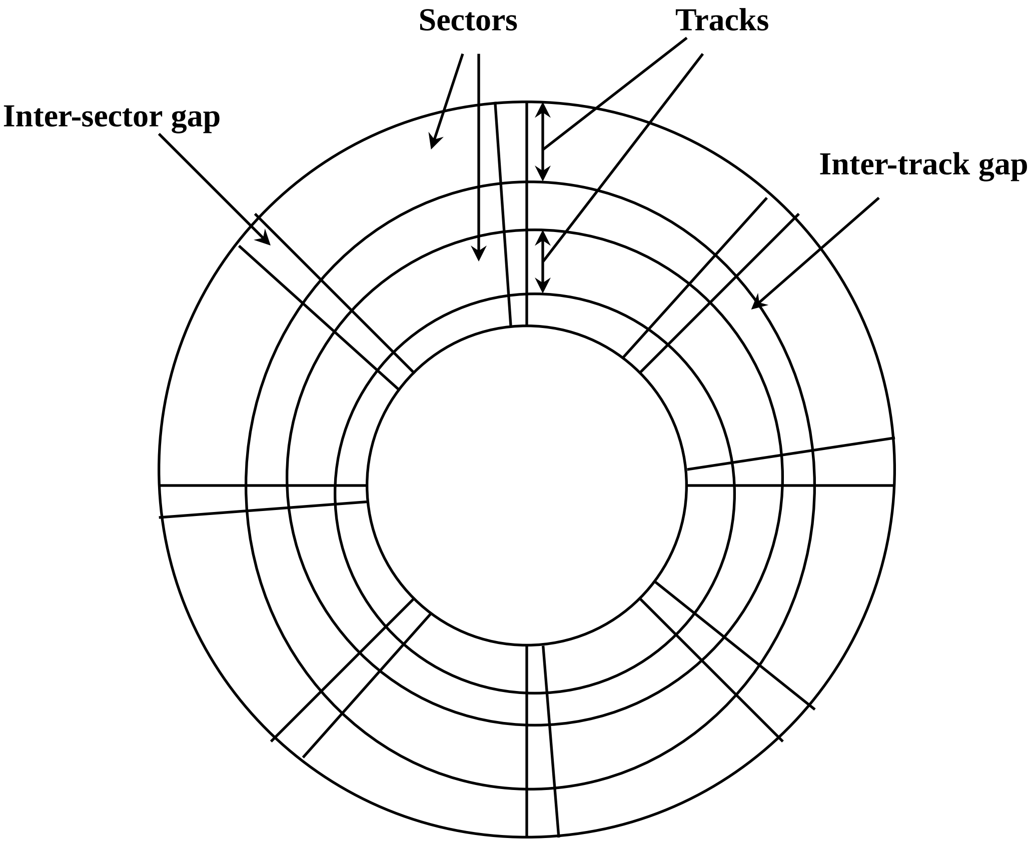
Double buffering

Circular Buffer

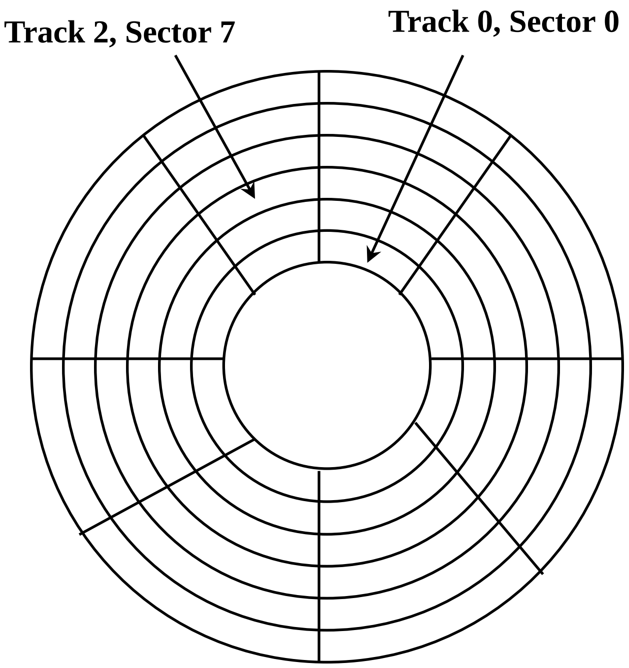
- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process



Disk Data Layout



Disk Layout Using Constant Angular Velocity



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
- ✓ *Seek time*
  - time it takes to position the head at the desired track
- ✓ *Rotational delay* or rotational latency
  - time its takes until desired sector is rotated to line up with the head

Disk Performance Parameters

- ✓ *Access time*
  - sum of seek time and rotational delay
  - the time it takes to get in position to read or write
- ✓ 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

Disk Scheduling Policies

- ✓ Seek time is the main reason for differences in performance
- ✓ For a single disk there can be a number of outstanding I/O requests
- ✓ If requests are selected randomly, we will get the worst possible performance
- ✓ The goal of disk scheduling is to process these requests so as to lower seek time

Disk Scheduling Policies

- ✓ <u>First-in, first-out (FIFO)</u>
  - process requests sequentially
  - fair to all processes
  - 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

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
  - possibility of starvation since a job may never regain the head of the line

Disk Scheduling Policies

- ✓ Shortest Service 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

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

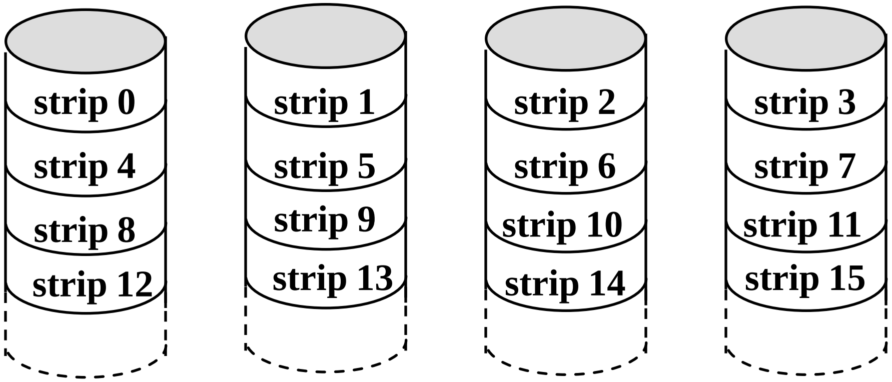
Disk Scheduling Policies

- ✓ 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 subqueues of length N
  - subqueues are processed one at a time, using SCAN
  - new requests added to other queues when the current queue is processed
- ✓ FSCAN
  - two queues
  - one queue is used for new requests

RAID 0 (non-redundant)



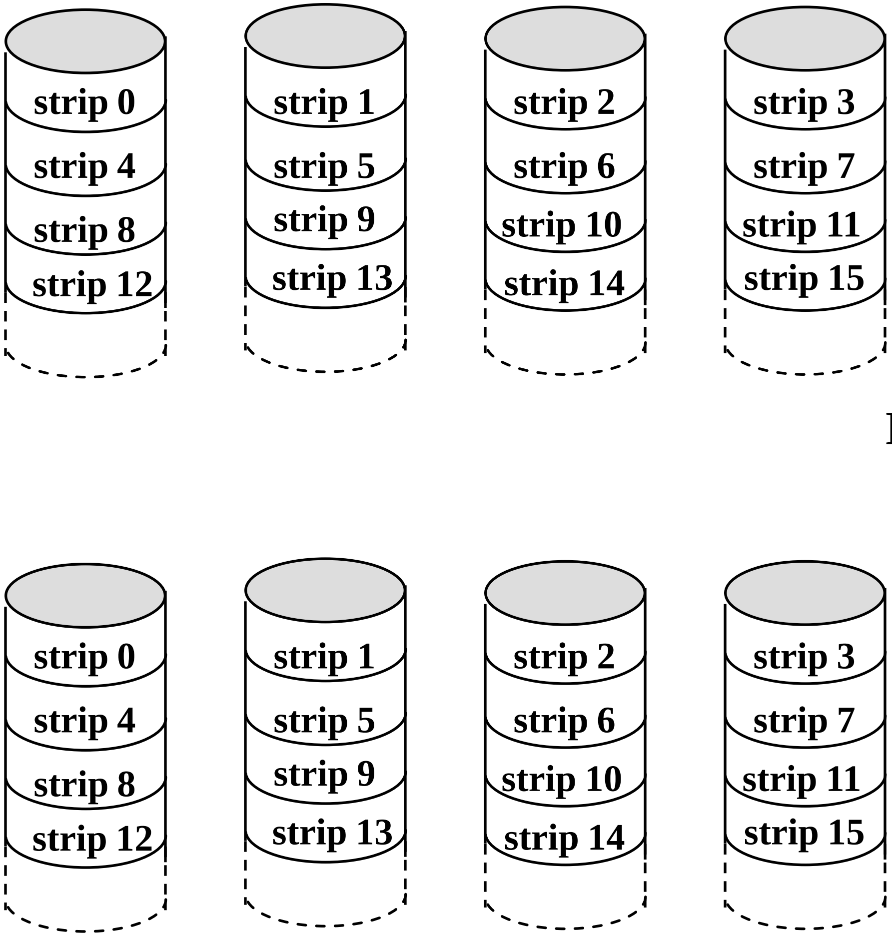
strip 0	strip 1	strip 2	strip 3
strip 4	strip 5	strip 6	strip 7
strip 8	strip 9	strip 10	strip 11
strip 12	strip 13	strip 14	strip 15

Performance:

I/O request rate: Excellent
Data transfer rate: Excellent

Several strips can be transferred in 1 I/O request

RAID 1 (mirrored)



strip 0 strip 1 strip 2 strip 3
strip 4 strip 5 strip 6 strip 7
strip 8 strip 9 strip 10 strip 11
strip 12 strip 13 strip 14 strip 15

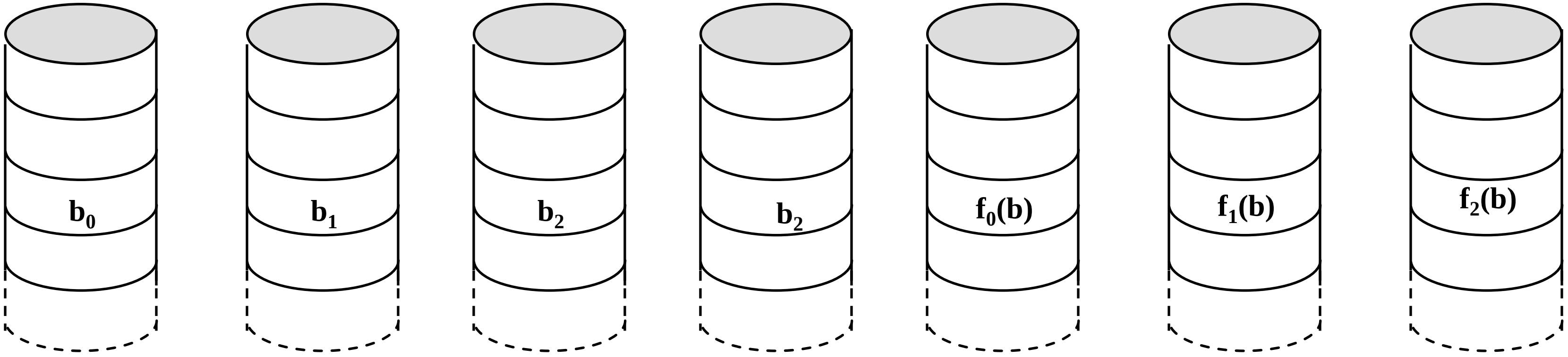
strip 0 strip 1 strip 2 strip 3
strip 4 strip 5 strip 6 strip 7
strip 8 strip 9 strip 10 strip 11
strip 12 strip 13 strip 14 strip 15

Mirroring improves Fault Tolerance

Performance:
- **I/O request rate: good**
- **Data transfer rate: good**

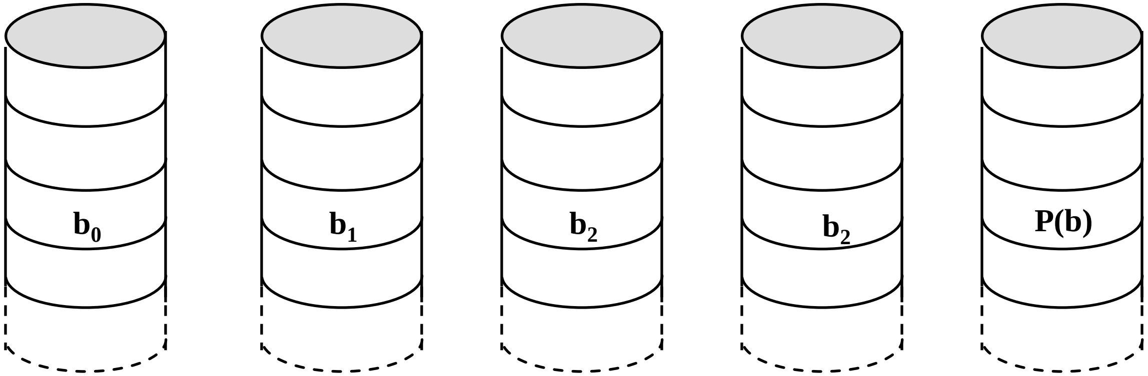
RAID 2 (redundancy through Hamming code)

Hamming code corrects 1-bit errors; detects 2 bit errors



Disk heads are synchronized, so that they are at the same place on each disk.
All disks are working on the same I/O request, so 1 I/O at a time!

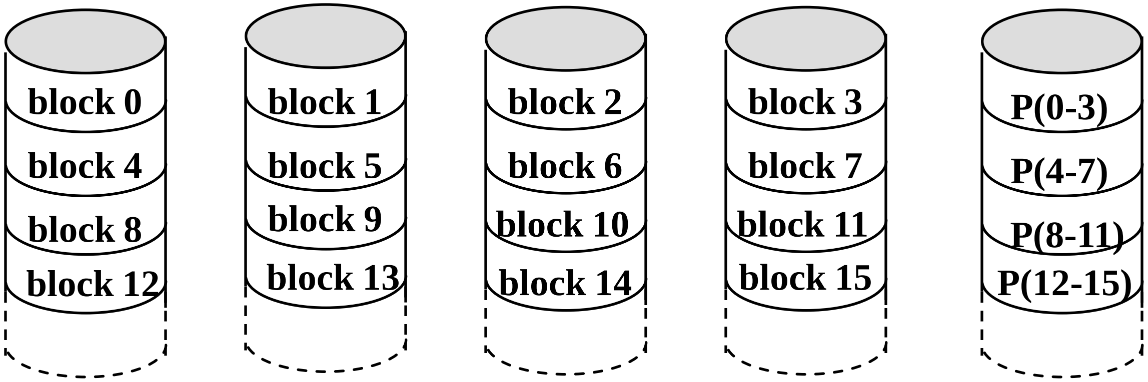
RAID 3 (bit-interleaved parity)



Similar to RAID 2, but parity bit is used. Can correct 1-bit errors

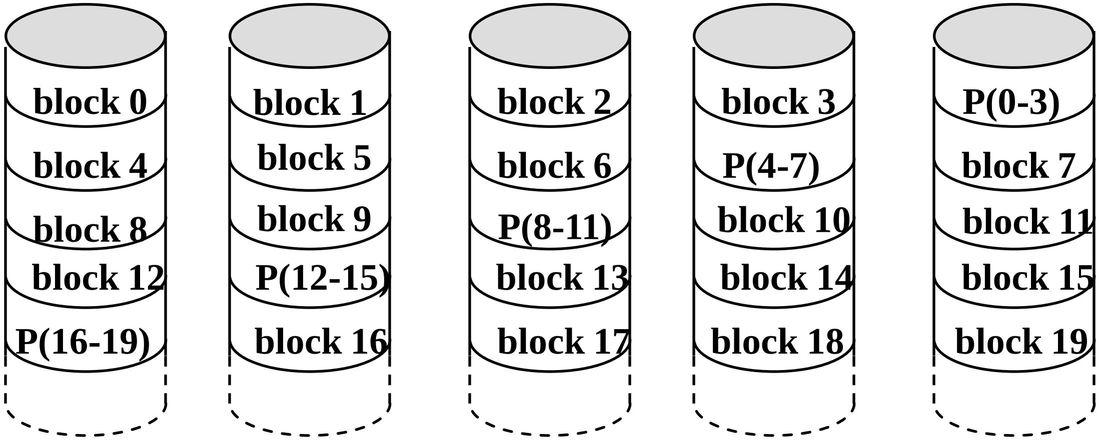
RAID 2 & 3: only one I/O request can be performed at a time,
hence poor I/O request rate.
Very good data transfer rate!

RAID 4 (block-level parity)



Parity, like RAID 3
However, each disk works independently,
so multiple I/O requests can be processed at the same time

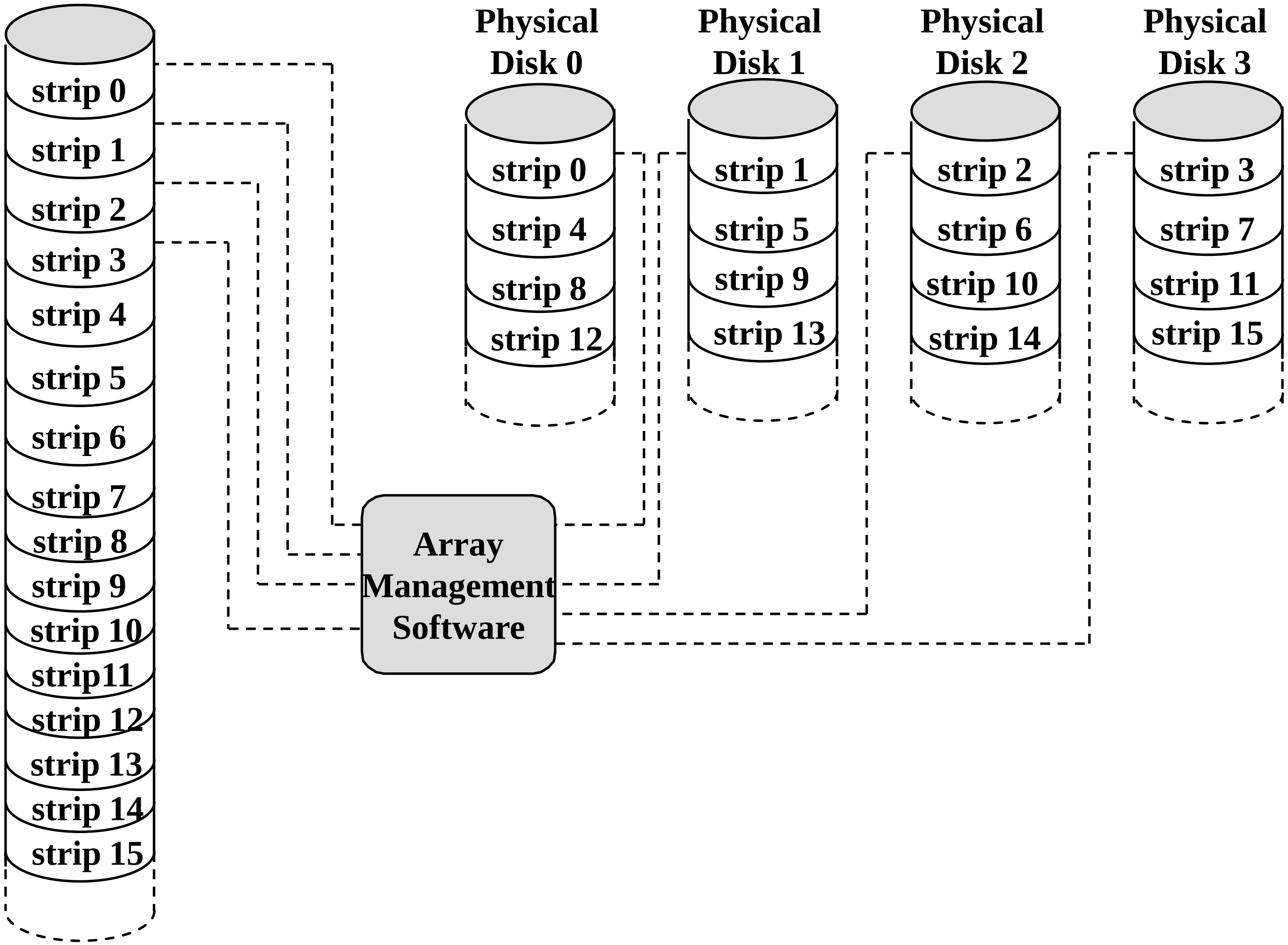
RAID 5 (block-level distributed parity)



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

Like RAID 4, but parity info is distributed across all disks.
Presumably, avoids bottleneck presented by a single parity disk.

Data Mapping for RAID Level 0 Array



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

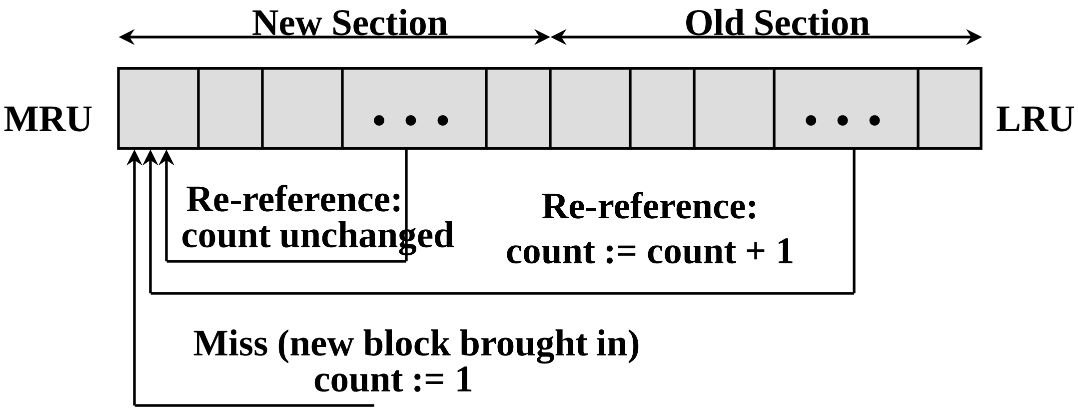
Least Recently Used

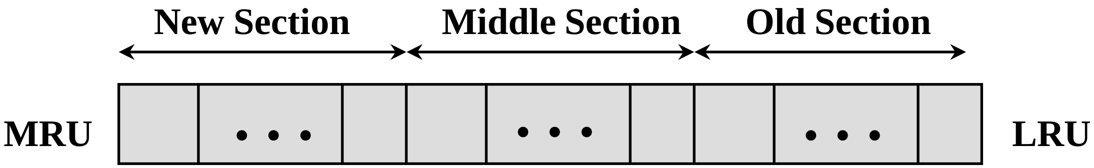
- ✓ The block on the bottom of the stack is removed when a new block is brought in
- ✓ Blocks don’t actually move around in main memory
- ✓ A stack of pointers is used

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 is accessed
- ✓ <u>Problem</u>: Some blocks may be referenced many times in a short period of time and then not needed any more

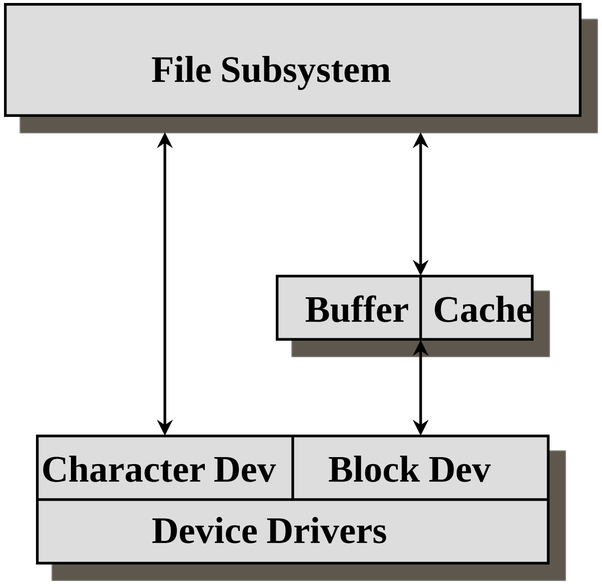
Frequency-based Replacement





Refinement: Use of three sections

UNIX I/O Structure



File Subsystem

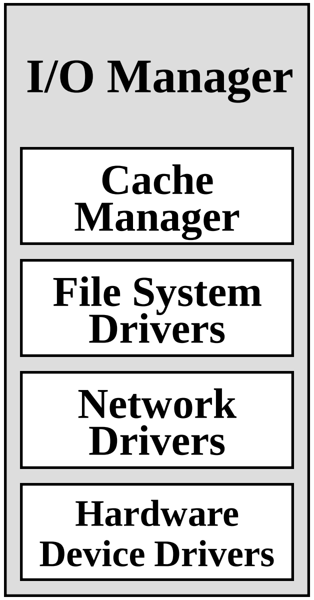
Buffer Cache

Character Dev

Block Dev

Device Drivers

Windows NT 4.0 I/O Manager



I/O Manager

Cache Manager

File System Drivers

Network Drivers

Hardware Device Drivers