UNIT -6

INPUT/OUTPUT Management and Disk Scheduling

Roadmap

I/O Devices

- Organization of the I/O Function
- Operating System Design Issues
- I/O Buffering
- Disk Scheduling
- Raid
- Disk Cache

Categories of I/O Devices

- Difficult area of OS design
  - Difficult to develop a consistent solution due to a wide variety of devices and applications
- Three Categories:
  - Human readable
  - Machine readable
  - Communications

(1) Human readable

- Devices used to communicate with the user
- Printers and terminals
  - Video display
  - Keyboard
  - Mouse etc

(2) Machine readable

- Used to communicate with electronic equipment like..
  - Disk drives
  - USB keys
  - Sensors
  - Controllers
  - Actuators

(3) Communication

- Used to communicate with remote devices
  - Digital line drivers
  - Modems

An actuator is an output device for the conversion of supply energy into useful work. The output signal is controlled by the control system, and the actuator responds to the control signals via the control element.

Roadmap

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

Techniques for performing I/O

Q-1: List and briefly define three techniques for performing I/O.

Following summarized three techniques for performing I/O:

- **Programmed I/O:-**
  - The processor issues an I/O command, on behalf of a process, to an I/O module;

- **Interrupt-driven I/O:-**
  - The processor issues an I/O command on behalf of a process.
  - If the I/O instruction from the process is nonblocking, then the processor continues to execute instructions from the process that issued the I/O command.
  - Else the next instruction that the processor executes is from the OS, which will put
  - the current process in a blocked state and schedule another process.

Techniques for performing I/O

- **Direct memory access (DMA):-**
  - A DMA module controls the exchange of data between main memory and an I/O module.
  - The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

Techniques for performing I/O

Table 11.1 indicates the relationship among these three techniques

Table 11.1 I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Roadmap

- I/O Devices
- Organization of the I/O Function
- **I/O Buffering**
- Disk Scheduling
- Raid
- Disk Cache

I/O Buffering

- Processes must wait for I/O to complete before proceeding
  - To avoid deadlock certain pages must remain in main memory during I/O
- It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made. This technique is known as buffering.

Block-oriented Buffering

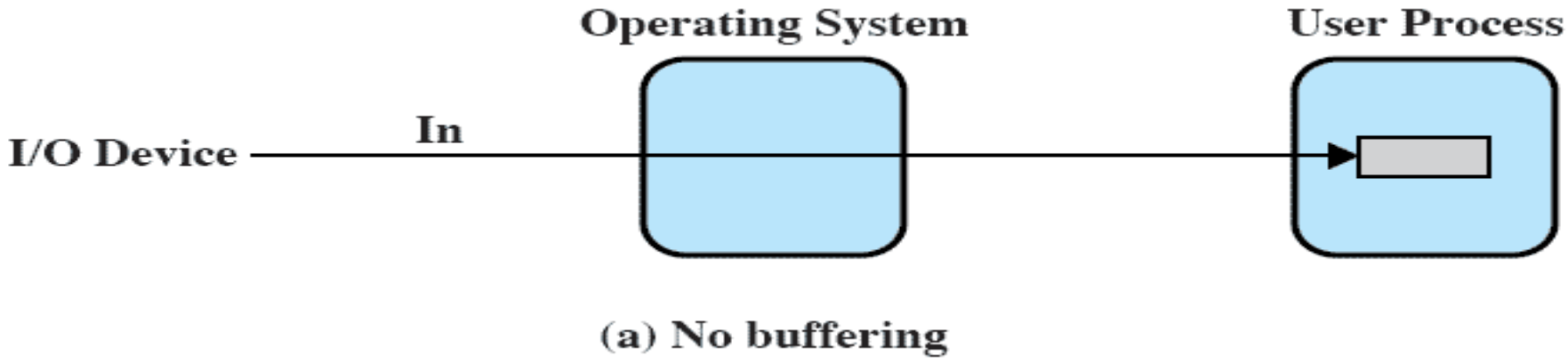
- Information is stored in fixed sized blocks
- Transfers are made a block at a time
  - Can reference data by block number
- Used for disks and USB keys

Stream-Oriented Buffering

- Transfer information as a stream of bytes
- Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage

No Buffer

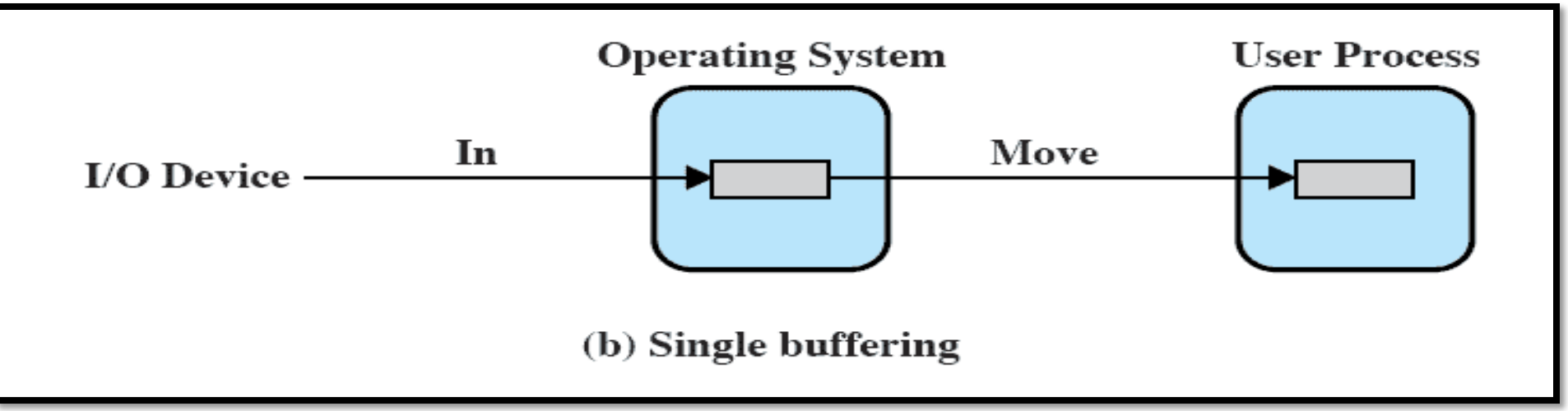
- Without a buffer, the OS directly access the device as and when it needs



(a) No buffering

Single Buffer

- Operating system assigns a buffer in main memory for an I/O request



Block Oriented Single Buffer

For block-oriented devices,

- Input transfers are made to the system buffer.
- When the transfer is complete, the process moves the block into user space and immediately requests another block.

Called **reading ahead**, or **anticipated input**;

- it is done in the expectation that the block will eventually be needed.

Often this is a reasonable assumption most of the time because data are usually accessed sequentially.

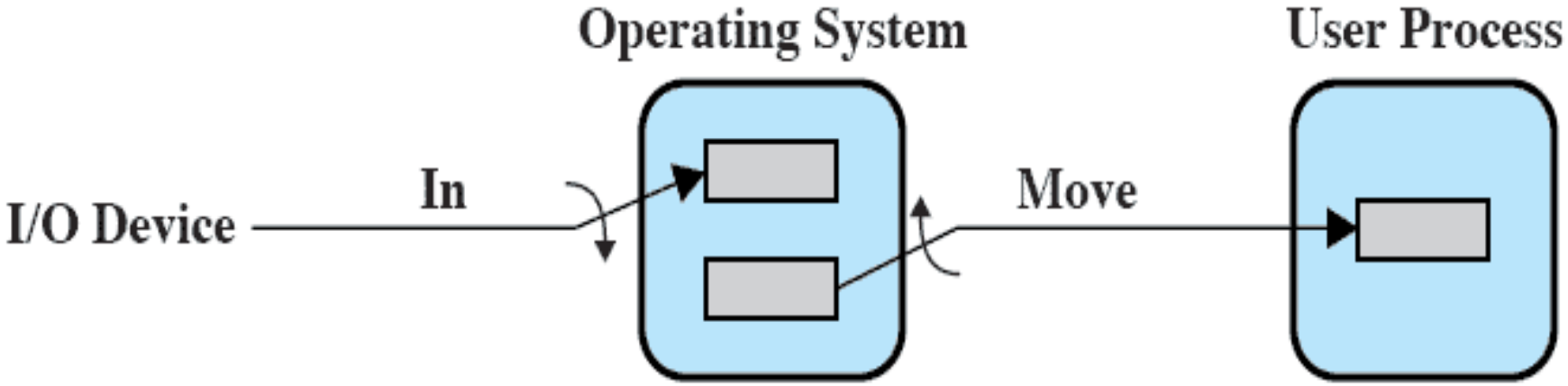
- Only at the end of a sequence of processing will a block be read in unnecessarily.

Stream-oriented Single Buffer

- The single buffering scheme can be used in a line-at-a-time fashion or a byte-at-a-time fashion.
  - Line-at-a-time operation is appropriate for scroll-mode terminals (sometimes called dumb terminals).
  - Byte-at-a-time operation is used on where each keystroke is significant, or for peripherals such as sensors and controllers.

Double Buffer

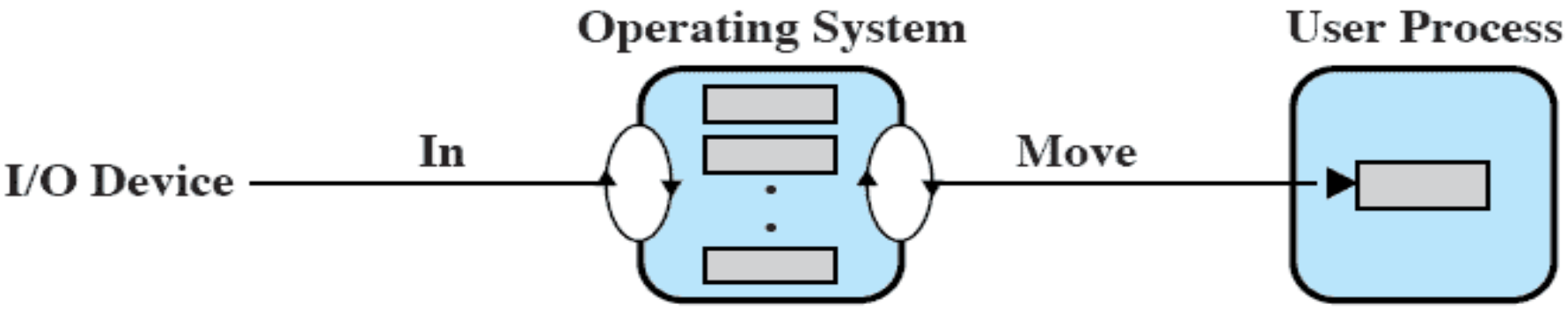
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



(c) 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



(d) Circular buffering

Buffer Limitations

- Buffering is a technique that smoothes out peaks in I/O demand.

- However, no amount of buffering will allow an I/O device to keep pace with a process indefinitely when the average demand of the process is greater than the I/O device can service.
  - Even with multiple buffers, all of the buffers will eventually fill up and the process will have to wait after processing each chunk of data.

- However, in a multiprogramming environment, when there is a variety of I/O activity and a variety of process activity to service, buffering is one tool that can increase the efficiency of the operating system and the performance of individual processes.

Roadmap

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

Disk Performance Parameters

Q-2: Briefly define the disk scheduling policies with example.

- When the disk drive is operating, the disk is rotating at constant speed.

- To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track.

- Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed-head system.

Disk Performance Parameters

Access Time = (SEEK TIME + ROTATIONAL DELAY)

- **Seek time:-**
  - On a movable-head system, the time it takes to position the head at the track is Known as **seek time.**
- **Rotational delay** or **rotational latency:-**
  - The time its takes for the beginning of the sector to reach the head is known as **rotational delay.**
- Seek time + rotational delay = **access time,**
  - The time it takes to get into position to read or write.
- **Transfer Time:-**
- Once the head is in position then the time taken to transfer the data is transfer time.

Disk Performance Parameters

- The actual details of disk I/O operation depend on many things
  - A general timing diagram of disk I/O transfer is shown here.

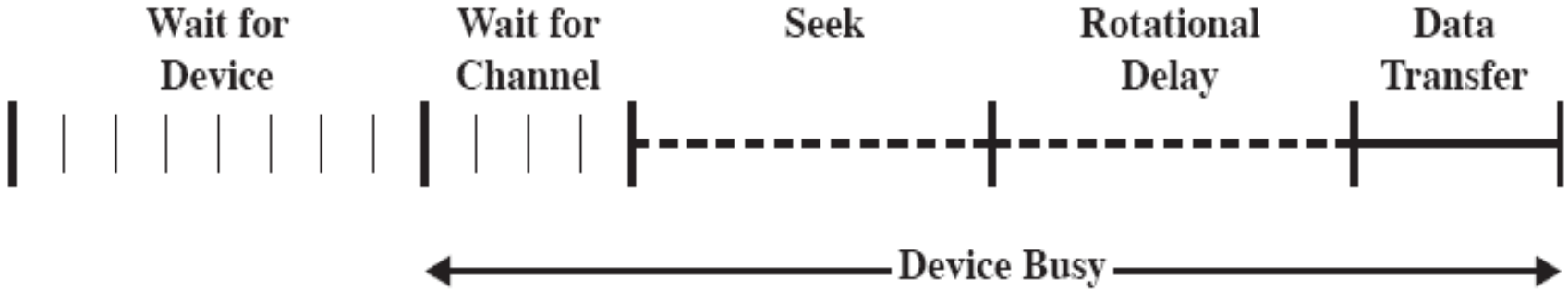


Figure 11.6 Timing of a Disk I/O Transfer

Disk Scheduling Policies

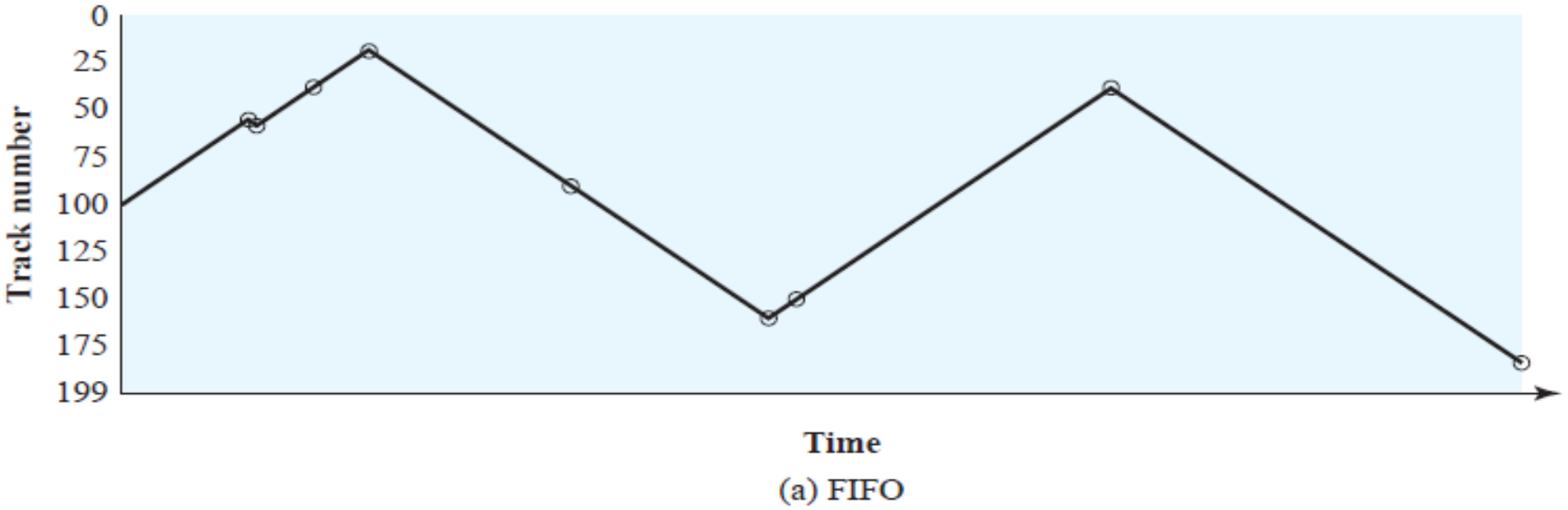
- **RANDOM SHCHEDULING:-**
  - If we selected items from the queue in random order, then we can expect that the tracks to be visited will occur randomly, giving poor performance.
  - This is called **random scheduling**

- To compare various schemes, consider a disk head is initially located at track 100.
  - assume a disk with 200 tracks and that the disk request queue has random requests in it.
- The requested tracks, in the order received by the disk scheduler, are
  - 55, 58, 39, 18, 90, 160, 150, 38, 184.

First-in, first-out (FIFO)

- The simplest form of scheduling is first-in-first-out (FIFO) scheduling, which processes items from the queue in sequential order.
- This strategy has the advantage of being fair, because every request is honored in the order received.
- 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

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



(a) FIFO

Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

FIFO - PERFORMANCE

(a) FIFO (starting at track 100)

Next track accessed	Number of tracks traversed
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
Average seek length	55.3

Table 11.2 Comparison of Disk Scheduling Algorithms

- Figure 11.7a illustrates the disk arm movement with FIFO.
  - This graph is generated directly from the data in Table 11.2a

Shortest Service Time First

- The SSTF policy is to 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.

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

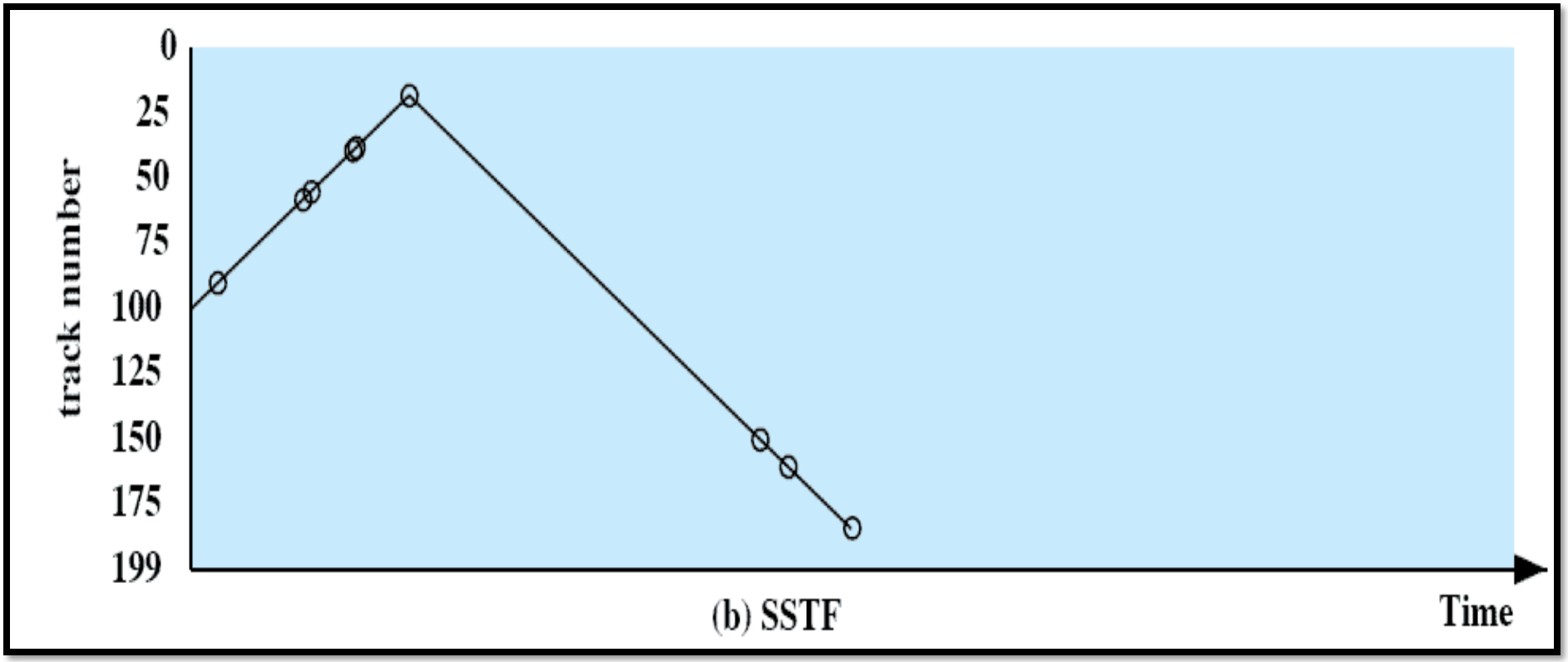


Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

Shortest Service Time First

(b) SSTF (starting at track 100)

Next track accessed	Number of tracks traversed
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
Average seek length	27.5

Table 11.2 Comparison of Disk Scheduling Algorithms

- Always choosing the minimum seek time does not guarantee that the average seek time over a number of arm movements will be minimum.
- However, this should provide better performance than FIFO.
- Because the arm can move in two directions, a random tie-breaking algorithm may be used to resolve cases of equal distances.

SCAN

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

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

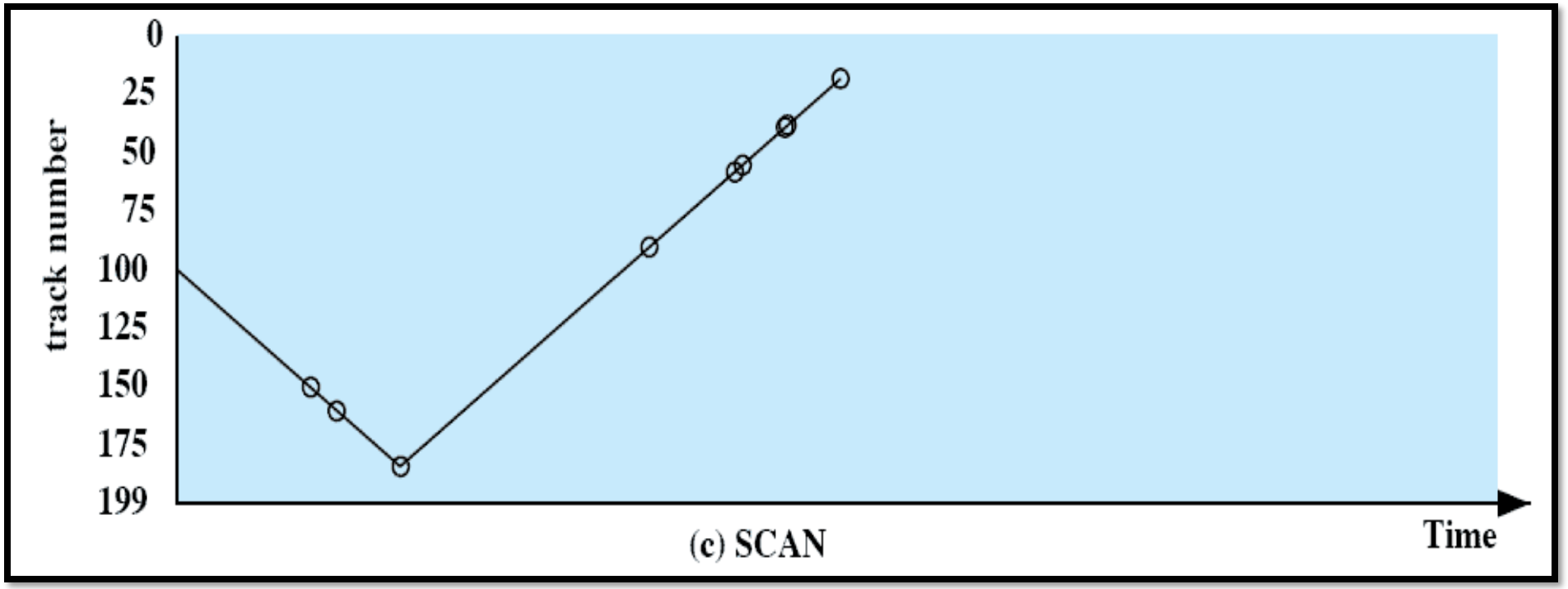


Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

SCAN

(c) SCAN (starting at track 100, in the direction of increasing track number)

Next track accessed	Number of tracks traversed
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
Average seek length	27.8

Table 11.2 Comparison of Disk Scheduling Algorithms

- The arm is required to move in one direction only until there are no more requests in that direction.
- The service direction is then reversed and the scan proceeds in the opposite direction, again picking up all requests in order.

This latter refinement is sometimes referred to as the LOOK policy.

The SCAN policy favors jobs whose requests are for tracks nearest to both innermost and outermost tracks and favors the latest-arriving jobs.

C-SCAN

- The C-SCAN (circular SCAN) policy restricts scanning to one direction only.
  - Thus, 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.
- This reduces the maximum delay experienced by new requests.

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

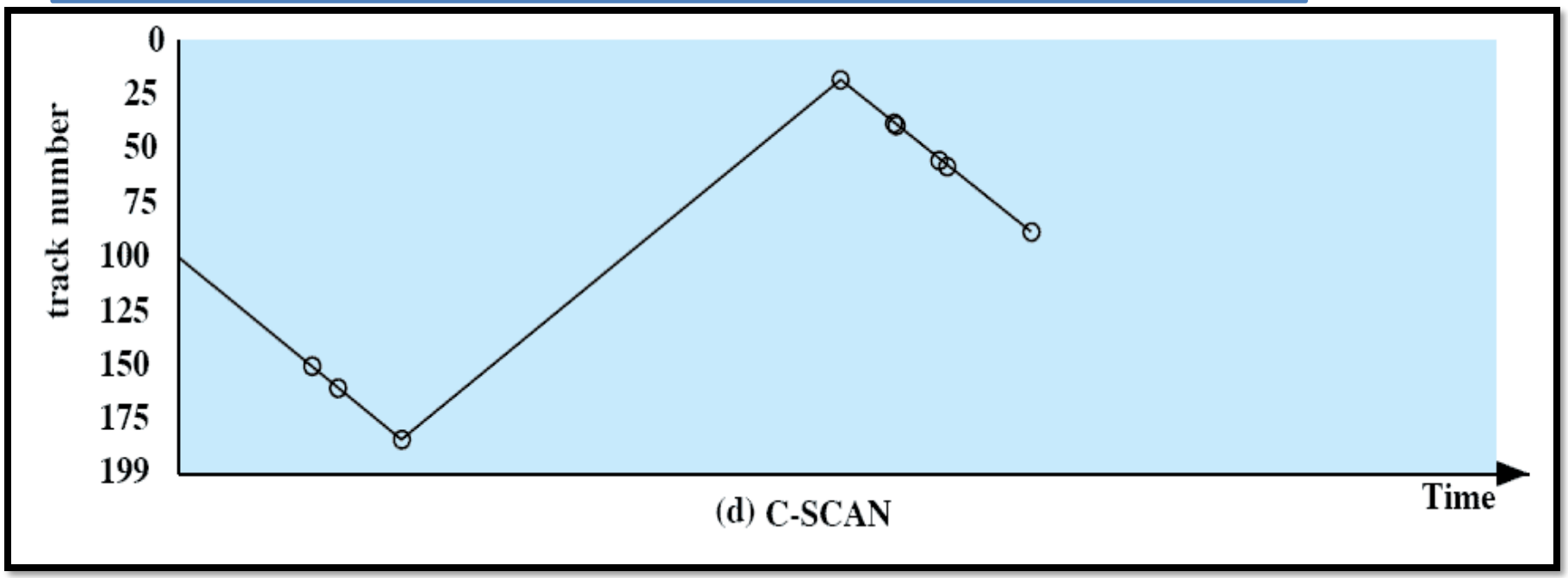


Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

C-SCAN

(d) C-SCAN (starting at track 100, in the direction of increasing track number)

Next track accessed	Number of tracks traversed
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
Average seek length	35.8

Table 11.2 Comparison of Disk Scheduling Algorithms

Performance Compared

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

Disk Scheduling Algorithms

Table 11.3 Disk Scheduling Algorithms

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

Roadmap

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

RAID

Q-3: Briefly define the seven RAID levels.

- Redundant Array of Independent Disks

- The RAID scheme consists of seven levels, zero through six.
- These levels do not imply a hierarchical relationship but designate different design architectures that share three common characteristics:
  1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
  2. Data are distributed across the physical drives of an array in a scheme known as striping, described subsequently.
  3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

RAID

Parity data is used by some RAID levels to achieve redundancy. using XOR function to reconstruct the missing data.

For example, suppose two drives in a three-drive RAID 51 array contained the following data:

Drive 1: **01101101**

Drive 2: **11010100**

To calculate parity data for the two drives, an XOR is performed on their data:

01101101

XOR **11010100**

10111001

The resulting parity data, **10111001**, is then stored on Drive 3.

If Drive 2 were to fail, its data could be rebuilt using the XOR results of the contents of the two remaining drives, Drive 1 and Drive 3:

Drive 1: **01101101**

Drive 3: **10111001** as follows:

10111001

XOR **01101101**

11010100

The result of that XOR calculation yields Drive 2's contents. **11010100** is then stored on Drive 2, fully repairing the array.

In the case of a RAID 3 array of 12 drives, 11 drives participate in the XOR calculation shown above and yield a value that is then stored on the dedicated parity drive.

RAID 0 - Stripped

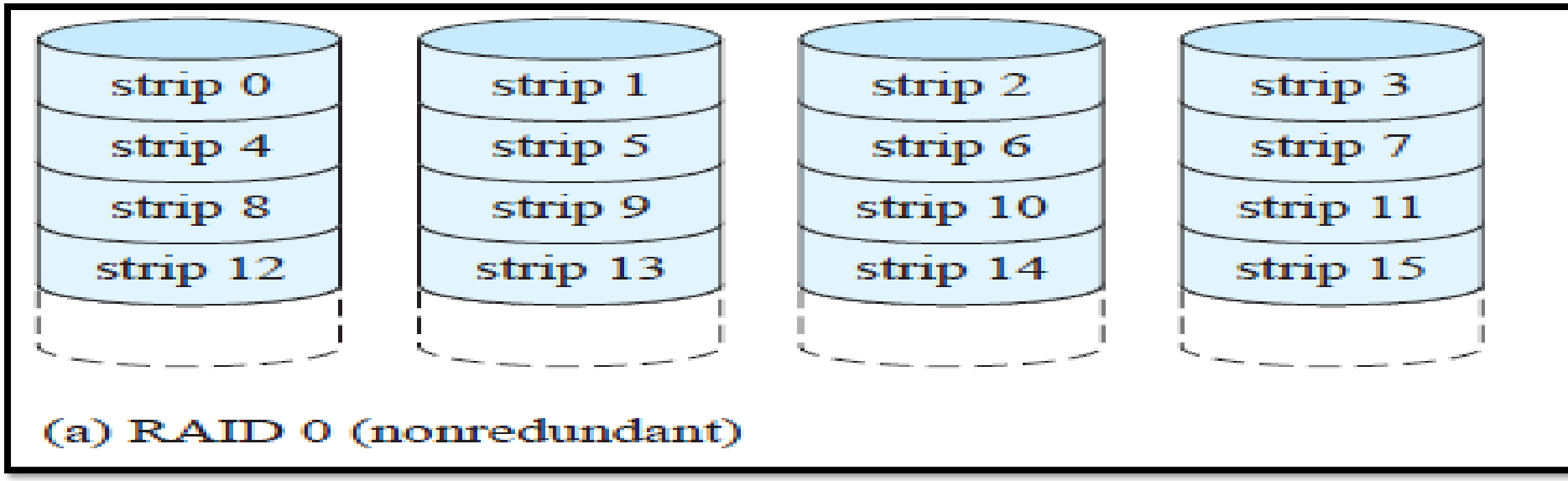


Figure 11.8 RAID Levels

RAID level 0 is not a true member of the RAID family, because it does not include redundancy.

The advantage of this layout is that if a single I/O request consists of multiple logically contiguous strips, then up to n strips for that request can be handled in parallel, greatly reducing the I/O transfer time.

RAID 1 - Mirrored

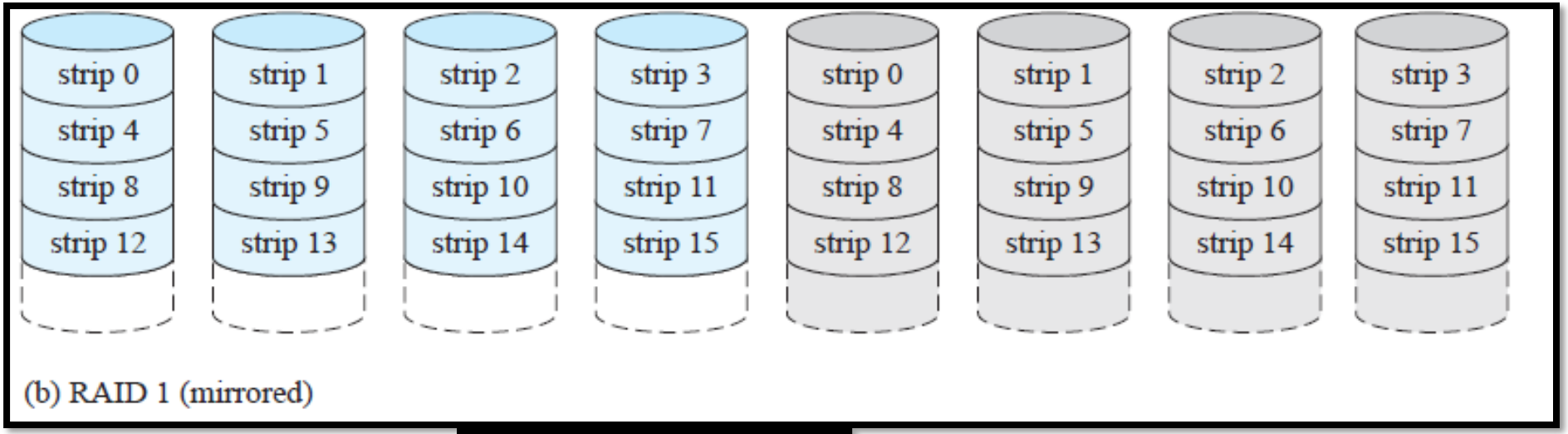


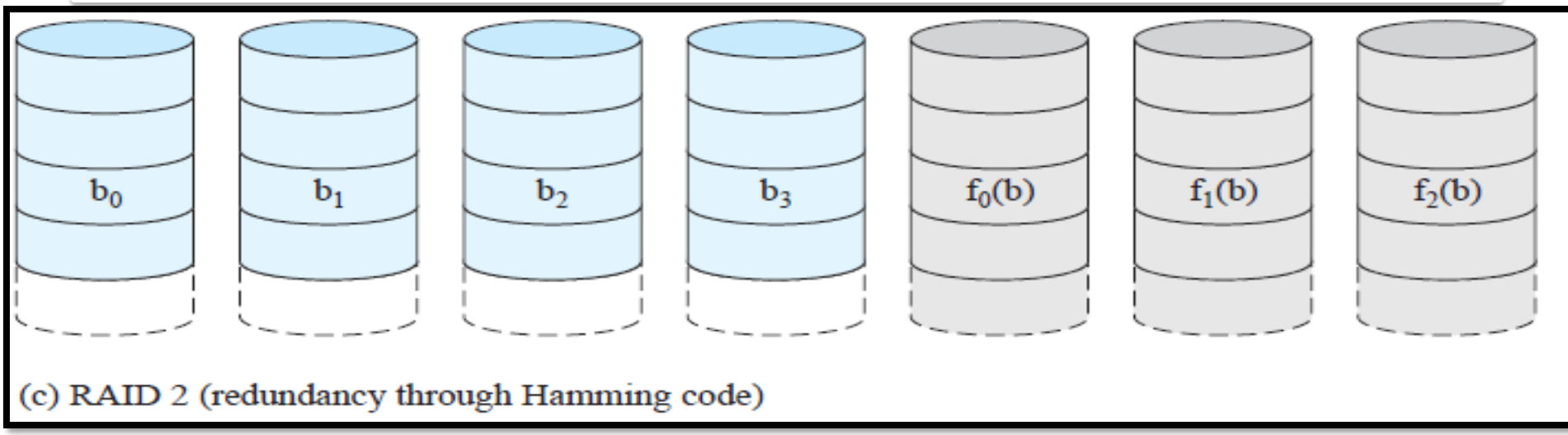
Figure 11.8 RAID Levels

- Each logical strip is mapped to two separate physical disks so that every disk in the array has a mirror disk that contains the same data.
- A read request can be serviced by either of the two disks that contains the requested data, whichever one involves the minimum seek time plus rotational latency.

- A write request requires that both corresponding strips be updated, but this can be done in parallel.
  - Thus, the write performance is dictated by the slower of the two writes

- Recovery from a failure is simple.
  - When a drive fails, the data may still be accessed from the second drive.

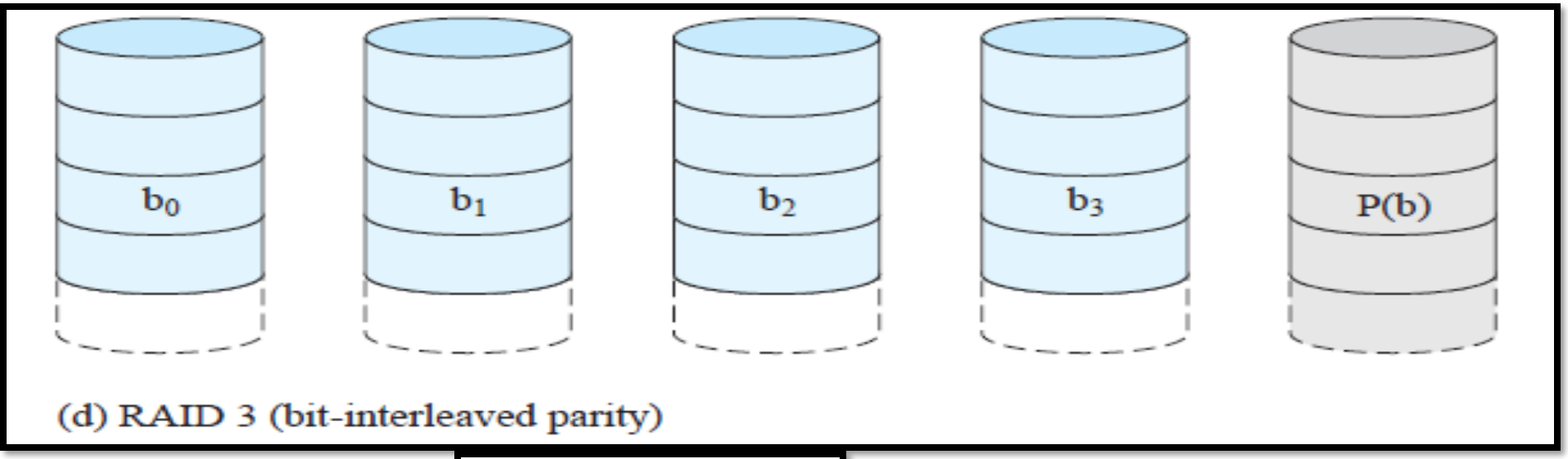
RAID 2 (Using Hamming code)



(c) RAID 2 (redundancy through Hamming code)

- In a parallel access array, all member disks participate in the execution of every I/O request.
- Typically, the spindles of the individual drives are synchronized so that each disk head is in the same position on each disk at any given time.
- As in the other RAID schemes, data striping is used.
- In the case of RAID 2 and 3, the strips are very small, often as small as a single byte or word.
- With RAID 2, an error-correcting code is calculated across corresponding bits on each data disk, and the bits of the code are stored in the corresponding bit positions on multiple parity disks.
- Typically, a Hamming code is used, which is able to correct single-bit errors and detect double-bit errors.

RAID 3 bit-interleaved parity



(d) RAID 3 (bit-interleaved parity)

Figure 11.8 RAID Levels

RAID 3 is organized in a similar fashion to RAID 2.

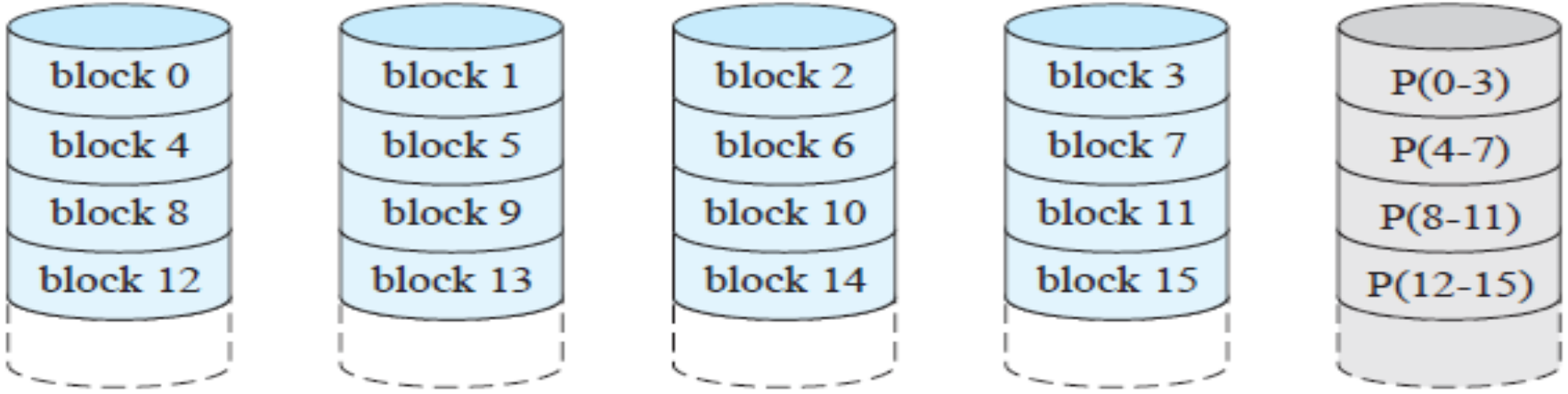
The difference is that RAID 3 requires only a single redundant disk, no matter how large the disk array.

RAID 3 employs parallel access, with data distributed in small strips.

Instead of an error-correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.

RAID 4 Block-level parity

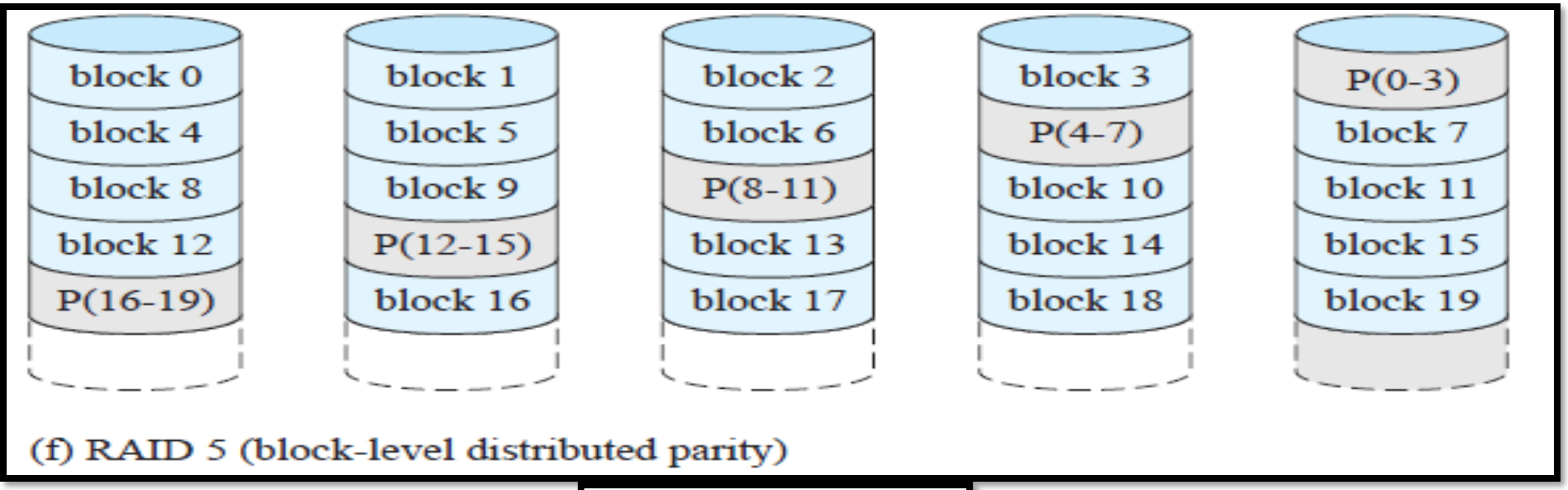
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk
- The parity bits are stored in the corresponding strip on the parity disk.



(e) RAID 4 (block-level parity)

Figure 11.8 RAID Levels

RAID 5 Block-level Distributed parity



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

Figure 11.8 RAID Levels

RAID 5 is organized in a similar fashion to RAID 4.

RAID 5 distributes the parity strips across all disks.

- A typical allocation is a round-robin scheme
- For an n-disk array, the parity strip is on a different disk for the first n stripes, and the pattern then repeats.

The distribution of parity strips across all drives avoids the potential I/O bottleneck of the single parity disk found in RAID 4.

RAID 6 Dual Redundancy

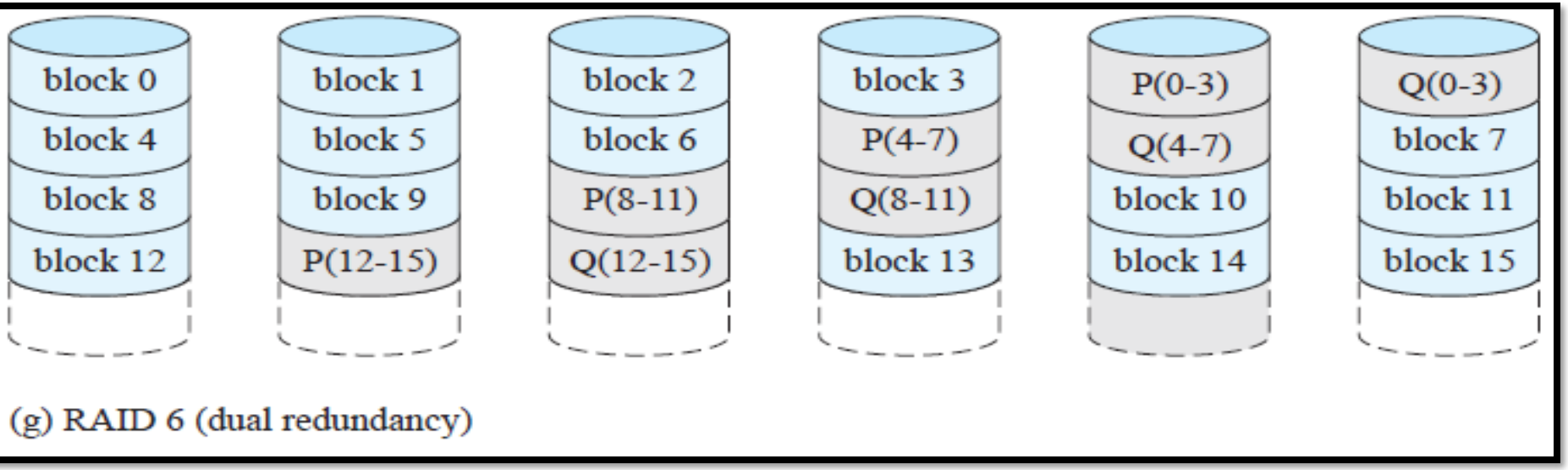


Figure 11.8 RAID Levels

Two different parity calculations are carried out and stored in separate blocks on different disks.

- Thus, a RAID 6 array whose user data require N disks consists of N+2 disks.

P and Q are two different data check algorithms.

- One of the two is the exclusive-OR calculation used in RAID 4 and 5.
- The other is an independent data check algorithm.

This makes it possible to regenerate data even if two disks containing user data fail.

File Management

Roadmap

- **Overview**
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management

Files

- In most applications, the file is the central element.
- From the user's point of view, one of the most important parts of an operating system is the file system.
- The file system provides the resource abstractions typically associated with secondary storage.

File Management

- File management system consists of system utility programs that run as privileged applications **CO**ncerned with secondary storage

Typical Operations

- Any file system provides not only a means to store data organized as files, but a collection of functions that can be performed on files.
- Typical operations include the following:
  - **Create:** A new file is defined and positioned within the structure of files.
  - **Delete:** A file is removed from the file structure and destroyed.
  - **Open:** An existing file is declared to be "opened" by a process, allowing the process to perform functions on the file.
  - **Close:** The file is closed with respect to a process, so that the process no longer may perform functions on the file, until the process opens the file again.
  - **Read:** A process reads all or a portion of the data in a file.
  - **Write:** A process updates a file, either by adding new data that expands the size S of the file or by changing the values of existing data items in the file.

File Management Systems

- A file management system is the set of system software that provides services to users and applications in the use of files.
  - Typically, the only way that a user or application may access files is through the file management system.

- This relieves the user or programmer of the necessity of developing special-purpose software for each application
- And provides the system with a consistent, well-defined means of controlling its most important asset.

Objectives for a File Management System

- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Minimize lost or destroyed data
- Provide a standardized set of I/O interface routines to user processes
- Provide I/O support for multiple users (if needed)

File Management Functions

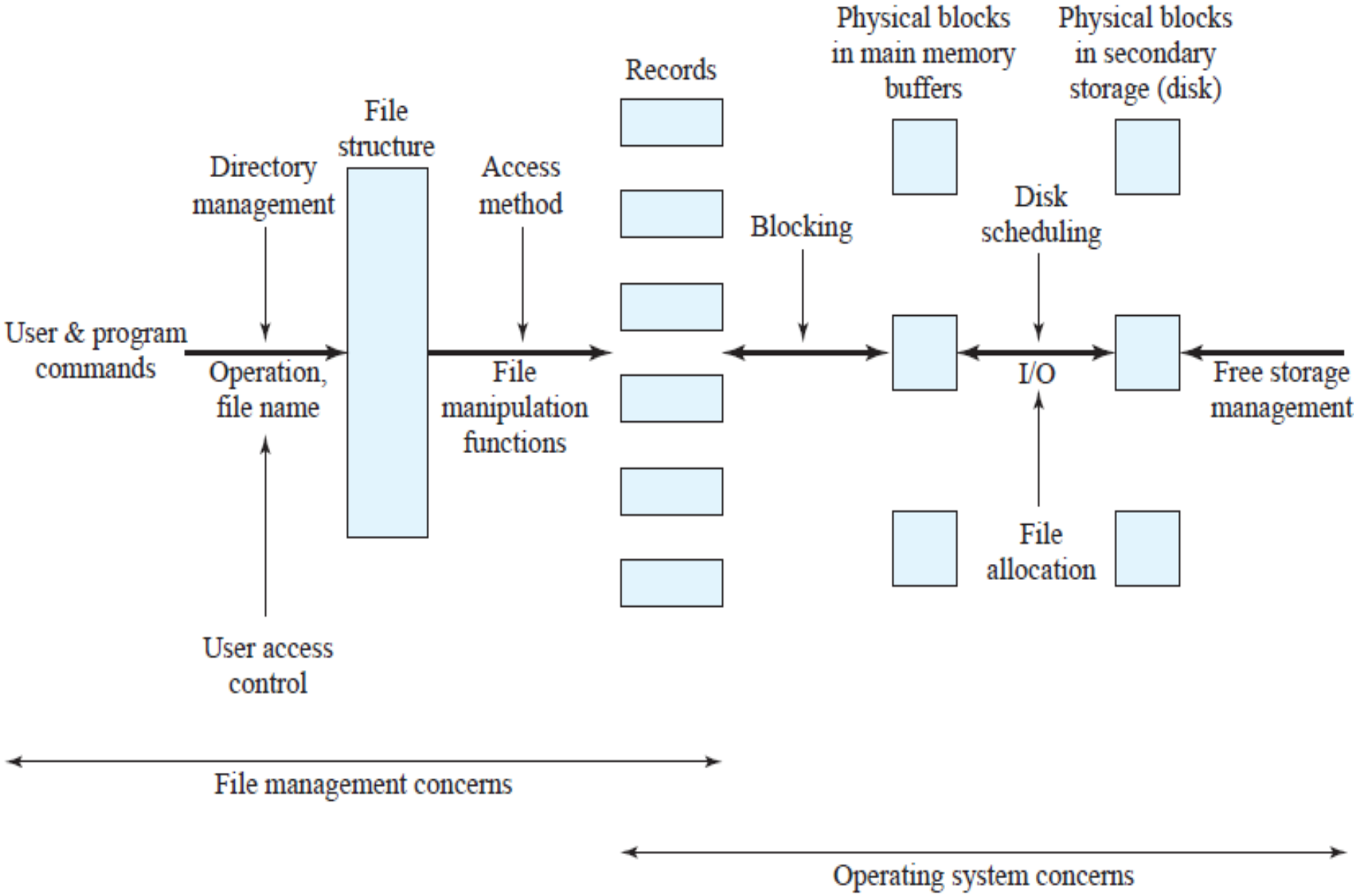


Figure 12.2 Elements of File Management

Roadmap

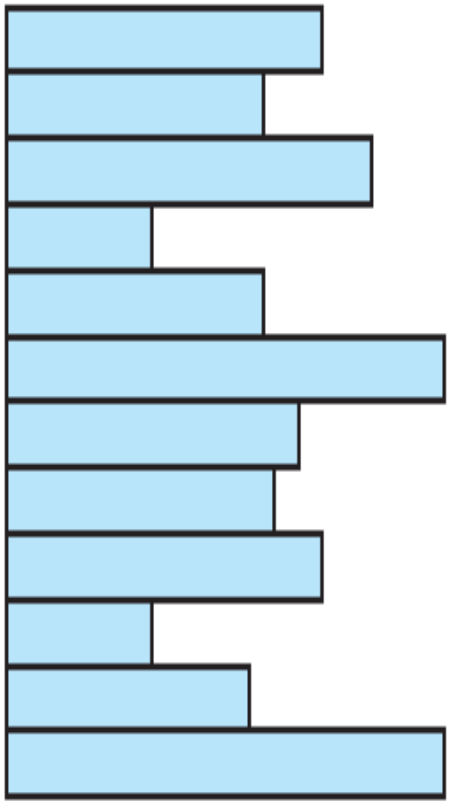
- Overview
- **File organisation and Access**
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management

File Organisation Types

- Many exist, but usually variations of:
  - Pile
  - Sequential file
  - Indexed sequential file
  - Indexed file
  - Direct, or hashed, file

The Pile

- The least-complicated form of file organization may be termed the pile.
- Data are collected in the order in which they arrive.
- Each record consists of one burst of data.
- The purpose of the pile is simply to accumulate the mass of data and save it.
- Records may have different fields, or similar fields in different orders.
- Because there is no structure to the pile file, record access is by exhaustive search.
  - ie , to find a record that contains a particular field with a particular value, it is necessary to examine each record in the pile until the desired record is found or the entire file has been searched.
  - To find all records that contain a particular field or contain that field with a particular value, then the entire file must be searched.

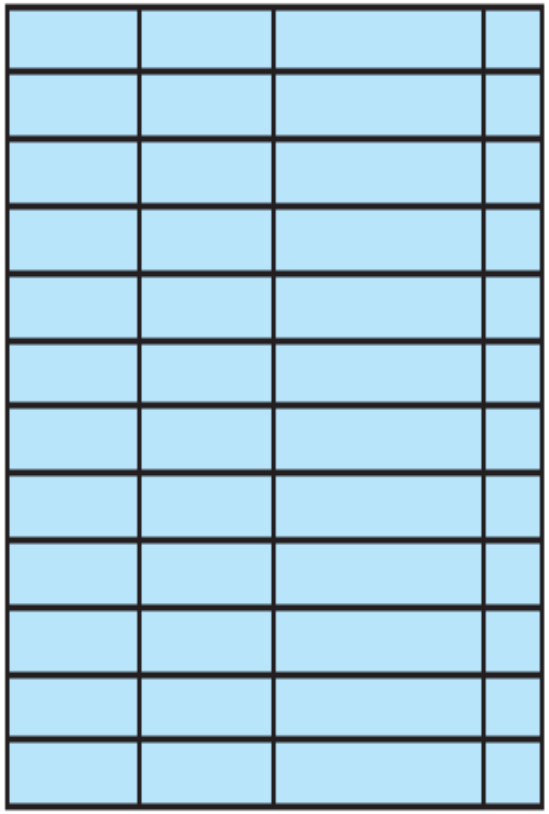


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

The Sequential File

- The most common form of file structure.
- A fixed format is used for records.
- All records are of the same length, consisting of the same number of fixed-length fields in a particular order.
  - Because the length and position of each field are known, only the values of fields need to be stored;
  - The field name and length for each field are attributes of the file structure.
- The key field uniquely identifies the record;
  - Thus key values for different records are always different.
  - The records are stored in key sequence: alphabetical order for a text key, and numerical order for a numerical key.



Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

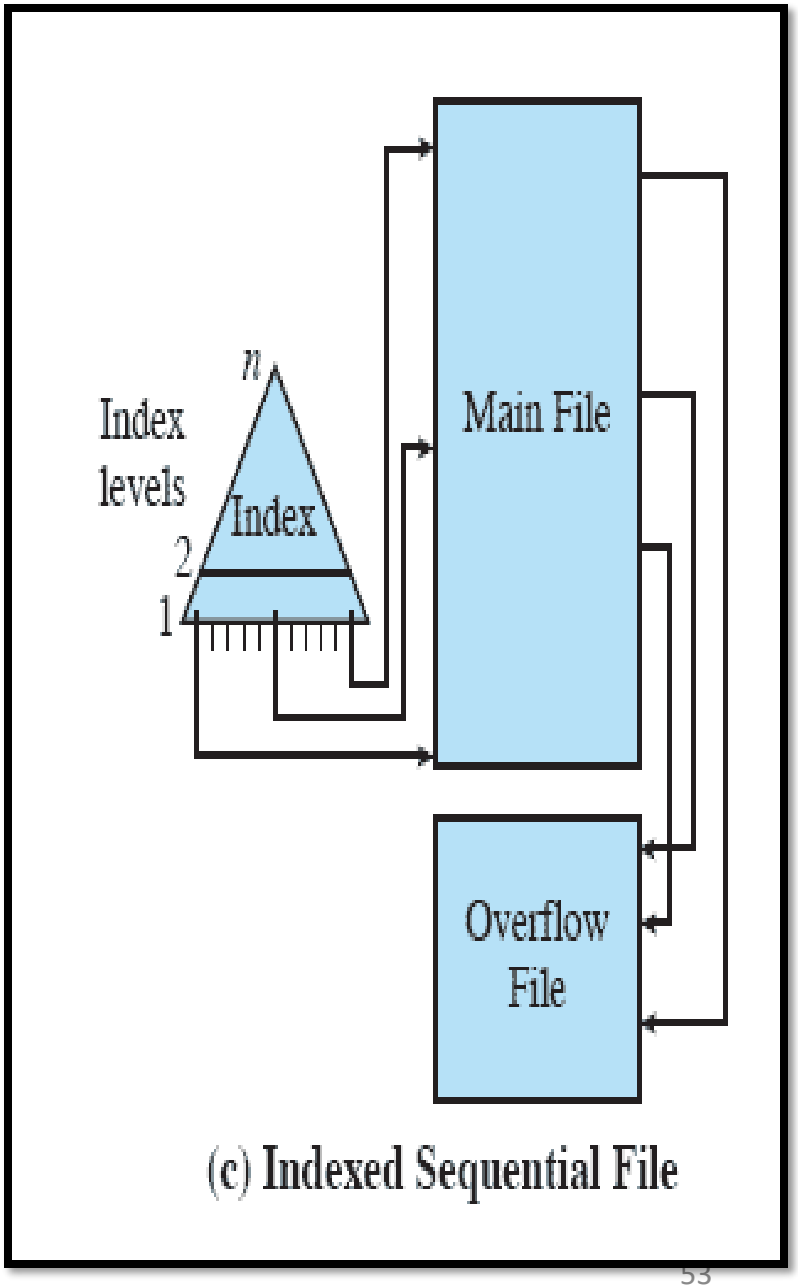
Indexed Sequential File

- The indexed sequential file maintains the key characteristic of the sequential file:
  - Records are organized in sequence based on a key field.
- Two features are added:
  - An index to the file to support random access,
  - And an overflow file.

- The index provides a lookup capability to reach quickly the vicinity of a desired record.

- When a new record is to be inserted into the file, it is added to the overflow file.

- The record in the main file is updated to contain a pointer to the new record in the overflow file.



(c) Indexed Sequential File

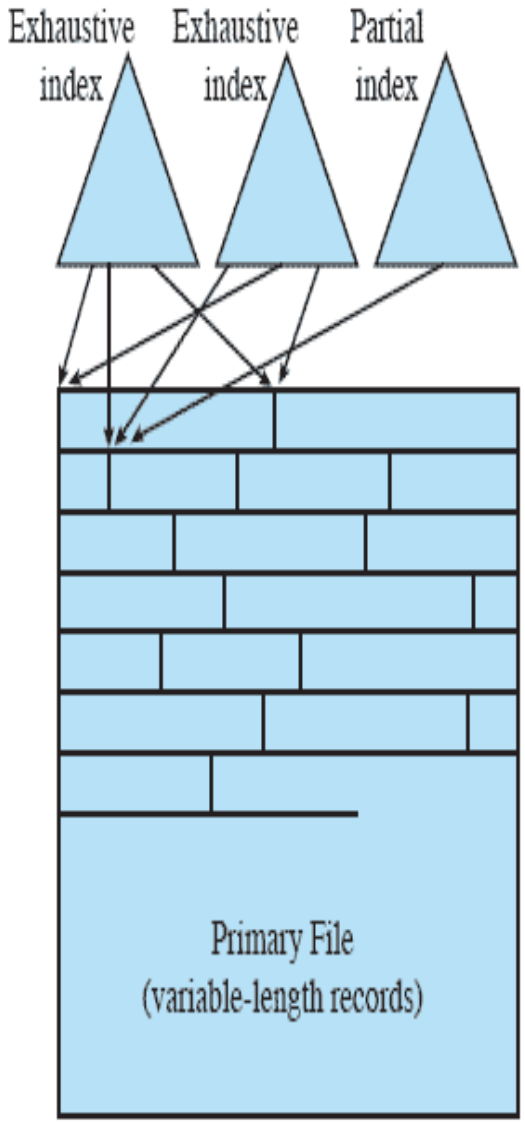
Indexed File

- In the general indexed file, the concept of sequentiality and a single key are abandoned.

- Records are accessed only through their indexes.
  - Now no restriction on the placement of records as long as a pointer in at least one index refers to that record.
  - variable-length records can be employed.

- Two types of indexes are used.
  - An exhaustive index contains one entry for every record in the main file. The index itself is organized as a sequential file for ease of searching.
  - A partial index contains entries to records where the field of interest exists.

- When a new record is added to the main file, all of the index files must be updated.



(d) Indexed File

Direct, or hashed, file

- Exploits the capability found on disks to access directly any block of a known address.

- A key field is required in each record.
  - But there is no concept of sequential ordering.

- The direct file makes use of hashing on the key value.

- Direct files are often used where very rapid access is required, where fixed length records are used, and where records are always accessed one at a time.

Roadmap

- Overview
- File organisation and Access
- **File Directories**
- File Sharing
- Record Blocking
- Secondary Storage Management

Contents

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system
- Provides mapping between file names and the files themselves

Directory Elements

Directory Elements: Basic Information

- File Name
  - Name as chosen by creator (user or program).
  - Must be unique within a specific directory.
- File type
- File Organisation
  - For systems that support different organizations

Table 12.2 Information Elements of a File Directory

	Basic Information
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
	Address Information
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file

Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Simple Structure for a Directory

- The method for storing the previous information varies widely between systems
- Simplest is a list of entries, one for each file
  - Sequential file with the name of the file serving as the key provides no help in organizing the files
  - Forces user to be careful not to use the same name for two different files

Operations Performed on a Directory

- A directory system should support a number of operations including:
- **Search**:
  - When a user or application references a file, the directory must be searched to find the entry corresponding to that file.

- **Create file:**
  - When a new file is created, an entry must be added to the directory.

- **Delete file:**
  - When a file is deleted, an entry must be removed from the directory.

- **List directory:**
  - All or a portion of the directory may be requested. Generally, this request is made by a user and results in a listing of all files owned by that user, plus some of the attributes of each file

- **Update directory:**
  - Because some file attributes are stored in the directory, a change in one of these attributes requires a change in the corresponding directory entry.

Roadmap

- Overview
- File organisation and Access
- File Directories
- **File Sharing**
- Record Blocking
- Secondary Storage Management

File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

Access Rights

- A wide variety of access rights have been used by various systems
  - often as a hierarchy where one right implies previous
- None
  - User may not even know of the files existence
- Knowledge
  - User can only determine that the file exists and who its owner is
- Execution
  - The user can load and execute a program but cannot copy it
- Reading
  - The user can read the file for any purpose, including copying and execution

Access Rights

- Appending
  - The user can add data to the file but cannot modify or delete any of the file's contents
- Updating
  - The user can modify, delete, and add to the file's data.
- Changing protection
  - User can change access rights granted to other users
- Deletion
  - User can delete the file

User Classes

- One user is designated as owner of a given file, usually the person who initially created a file.
- The owner has all of the access rights listed previously and may grant rights to others.
- Access can be provided to different classes of users:

- **Specific user:** Individual users who are designated by user ID.

- **User groups:** A set of users who are not individually defined.
  - The system must have some way of keeping track of the membership of user groups.

- **All:** All users who have access to this system. These are public files.

Simultaneous Access

- User may lock entire file when it is to be updated
- User may lock the individual records during the update
- Mutual exclusion and deadlock are issues for shared access

Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- **Record Blocking**
- Secondary Storage Management

Blocks and records

- Records are the logical unit of access of a structured file
  - But blocks are the unit for I/O with secondary storage
- For I/O to be performed, records must be organized as blocks.
- Three approaches are common
  - Fixed length blocking
  - Variable length spanned blocking
  - Variable-length unspanned blocking

Fixed Blocking

- Fixed-length records are used, and an integral number of records are stored in a block.
- Unused space at the end of a block is ***internal fragmentation***

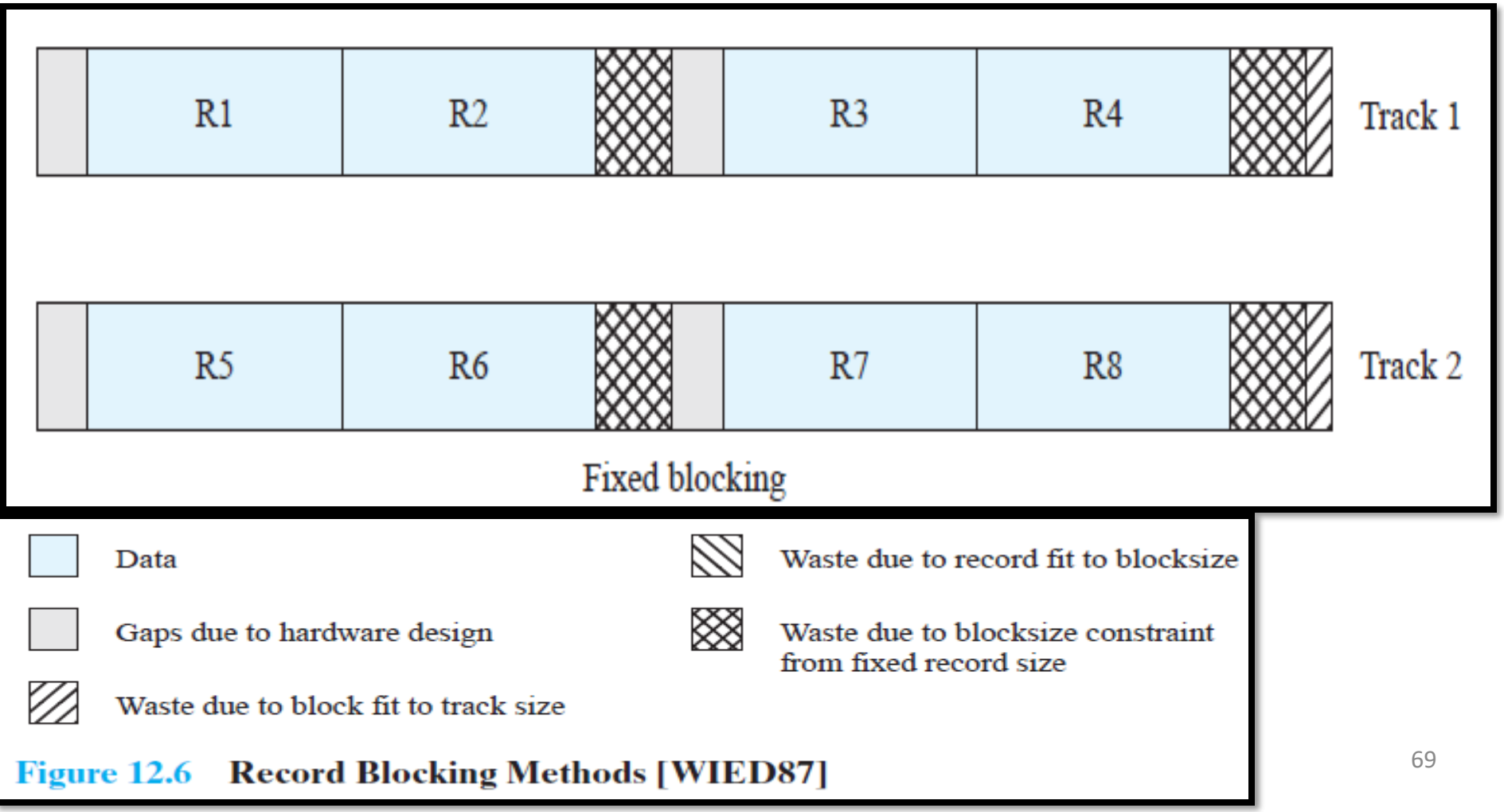
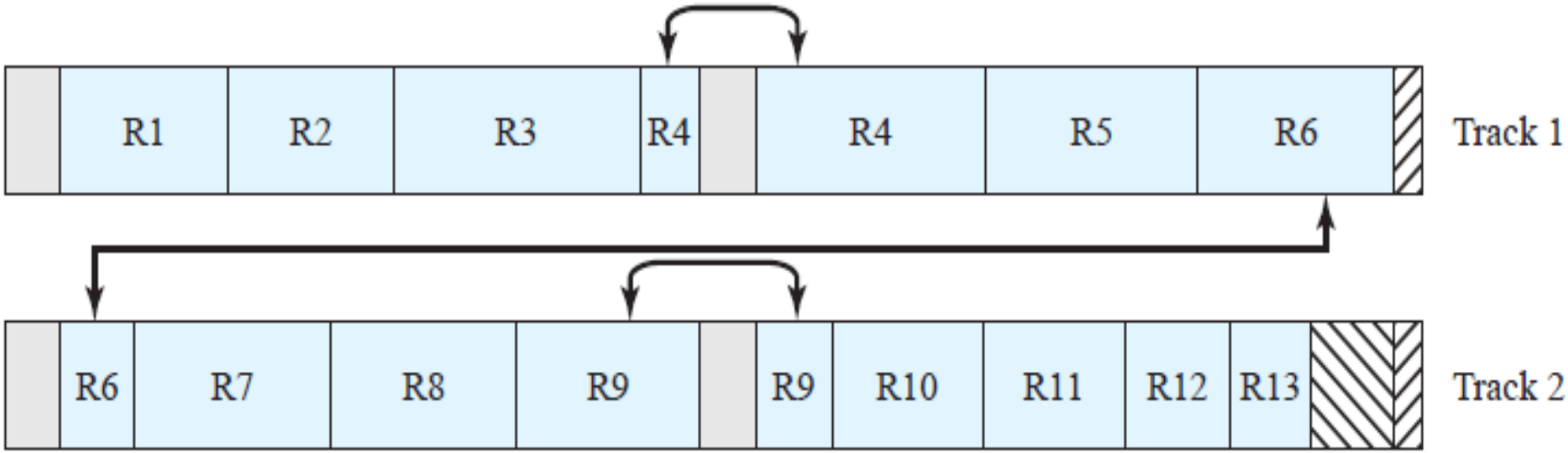


Figure 12.6 Record Blocking Methods [WIED87]

Variable Length Spanned Blocking

- Variable-length records are used and are packed into blocks with no unused space.
- Some records may span multiple blocks
  - Continuation is indicated by a pointer to the successor block



Variable blocking: spanned

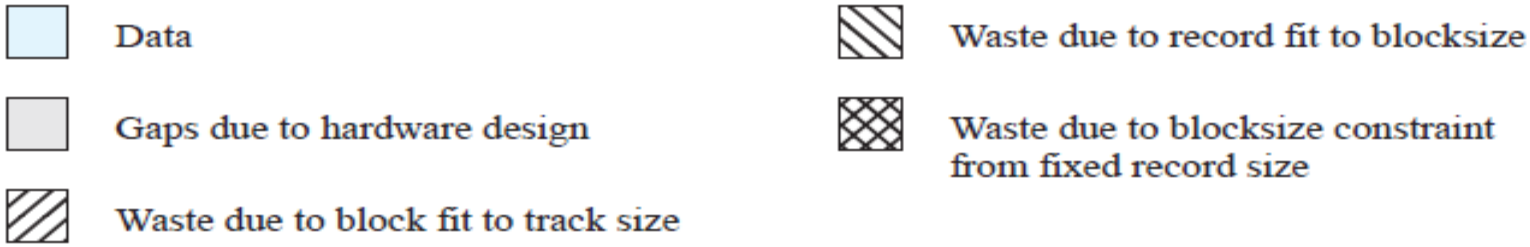


Figure 12.6 Record Blocking Methods [WIED87]

Variable-length unspanned blocking

- Uses variable length records without spanning
- Wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.

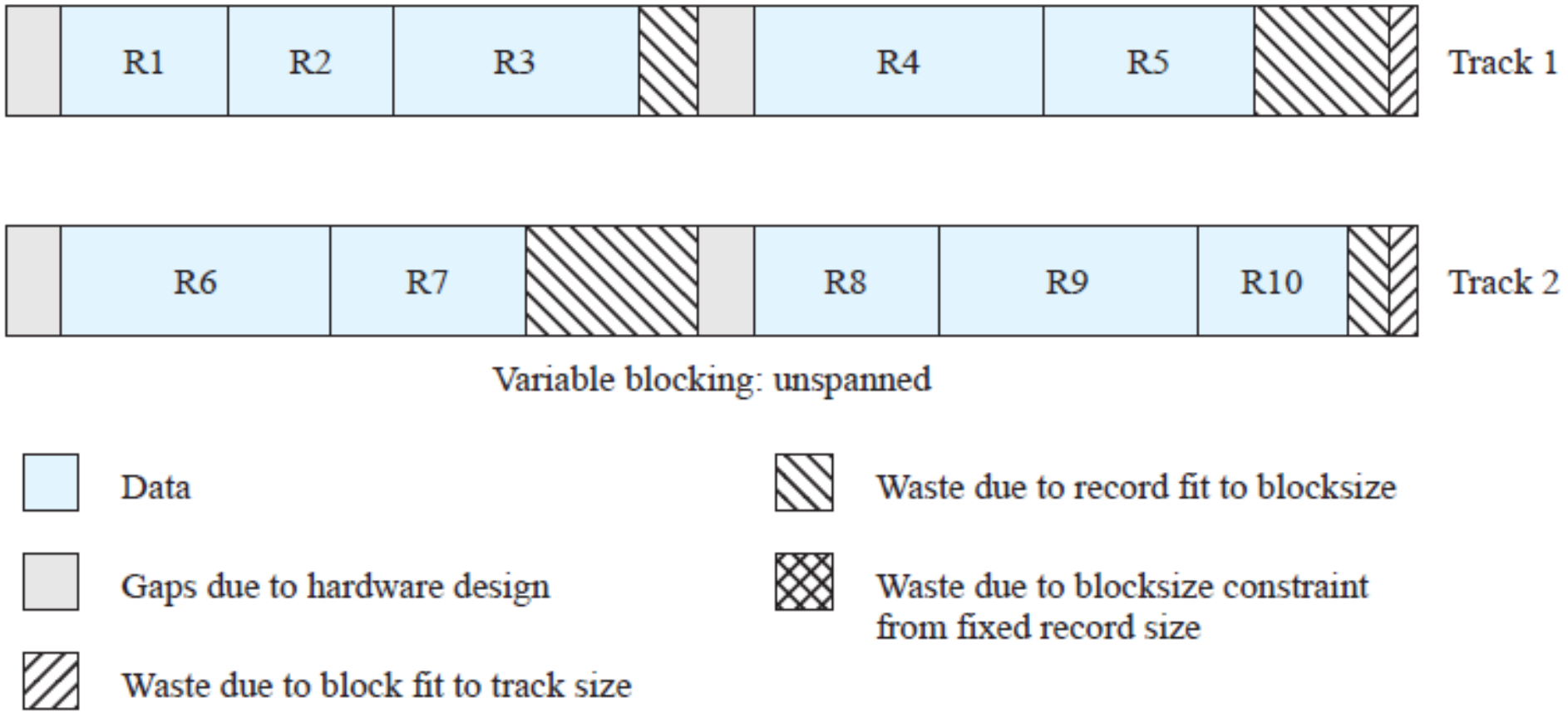


Figure 12.6 Record Blocking Methods [WIED87]

Q-5 Ends.

Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- **Secondary Storage Management**

Secondary Storage Management

On secondary storage, a file consists of a collection of blocks.

- The operating system or file management system is responsible for allocating blocks to files.

This raises two management issues.

- First, space on secondary storage must be allocated to files,
- second, it is necessary to keep track of the space available for allocation.

File Allocation Method

- Three methods are in common use:
  - contiguous,
  - chained, and
  - indexed.

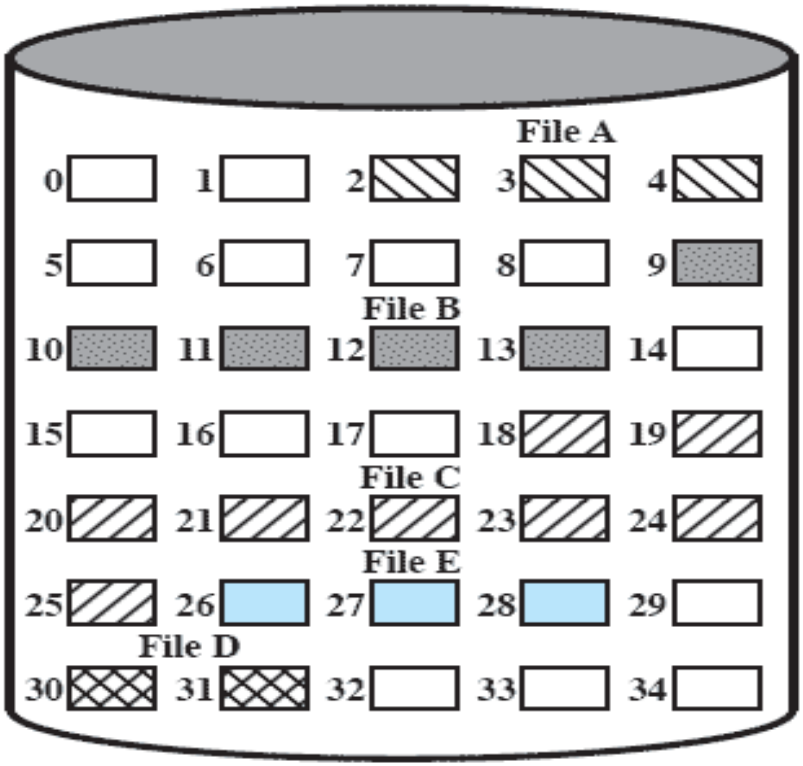
Contiguous Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation.
- This is a preallocation strategy, using variable-size portions.

- The file allocation table needs just a single entry for each file, showing the starting block and the length of the file.

- Contiguous allocation is the best from the point of view of the individual sequential file.
  - Multiple blocks can be read in at a time to improve I/O performance for sequential processing.
  - It is also easy to retrieve a single block.

Contiguous File Allocation

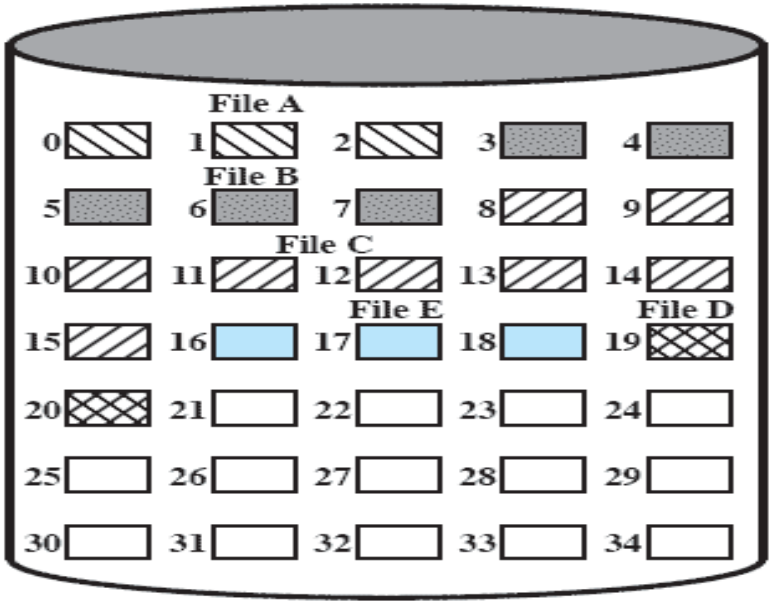


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.7 Contiguous File Allocation

External fragmentation



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length.

From time to time, it will be necessary to perform a compaction algorithm to free up additional space on the disk.

Also, with preallocation, it is necessary to declare the size of the file at the time of creation, with the problems mentioned earlier.

Chained Allocation

Typically, allocation is on an individual block basis.

- Each block contains a pointer to the next block in the chain.

The file allocation table needs just a single entry for each file, showing the starting block and the length of the file.

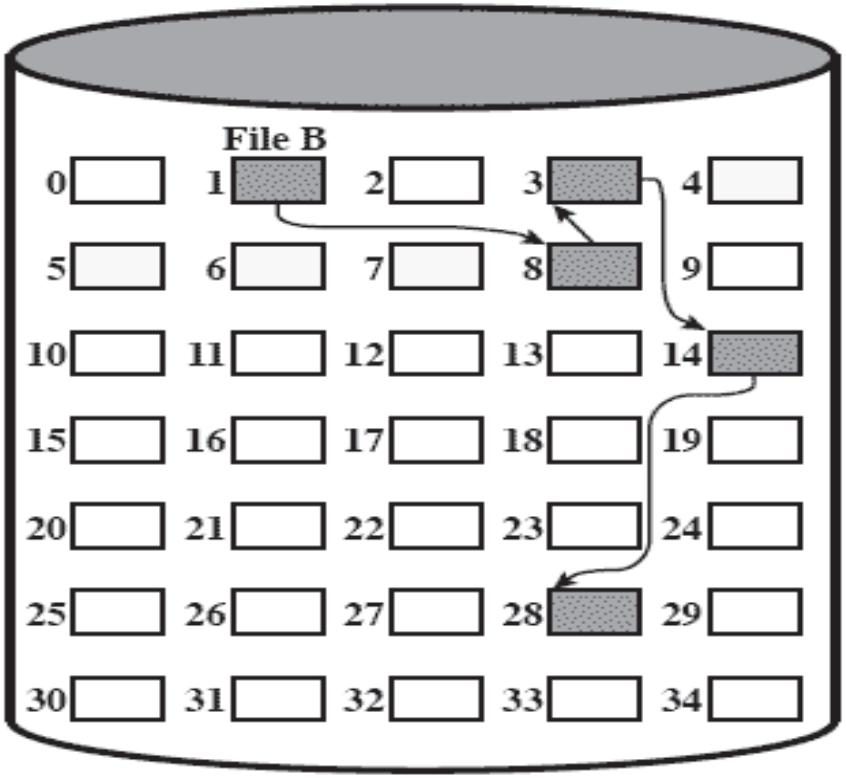
Although preallocation is possible, it is more common simply to allocate blocks as needed.

- The selection of blocks is now a simple matter: any free block can be added to a chain.
- There is no external fragmentation to worry about because only one block at a time is needed.

This type of physical organization is best suited to sequential files that are to be processed sequentially.

- To select an individual block of a file requires tracing through the chain to the desired block.

Chained Allocation

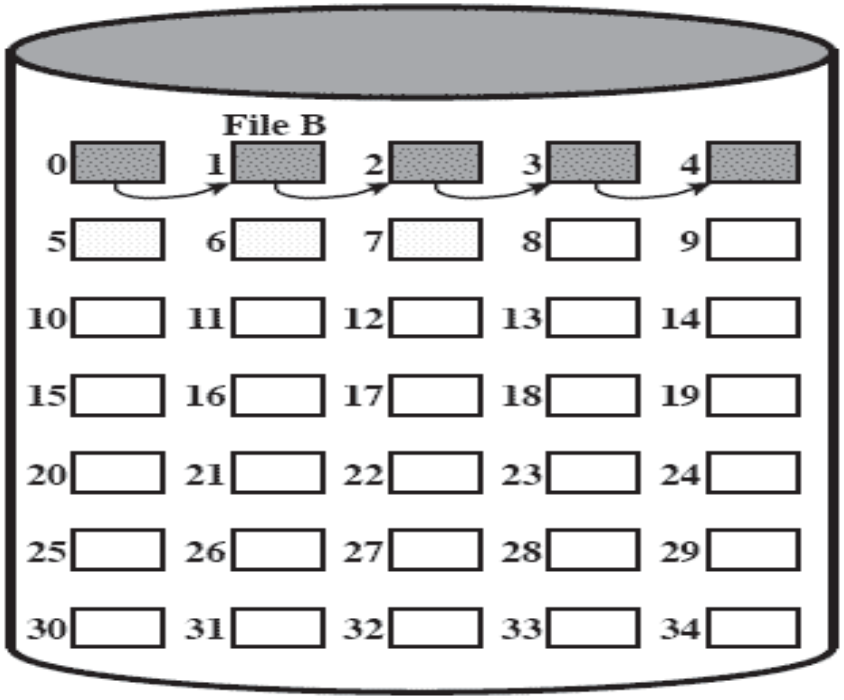


File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.9 Chained Allocation

Chained Allocation Consolidation



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (After Consolidation)

Chained Allocation Consolidation

- One consequence of chaining, is that there is no accommodation of the principle of locality.

- If it is necessary to bring in several blocks of a file at a time, as in sequential processing, then a series of accesses to different parts of the disk are required.
  - This is perhaps a more significant effect on a single-user system but may also be of concern on a shared system.
  - To overcome this problem, some systems periodically consolidate files

Indexed Allocation

This addresses many of the problems of contiguous and chained allocation.

In this case, the file allocation table contains a separate one-level index for each file;

- the index has one entry for each portion allocated to the file.

Typically, the file indexes are not physically stored as part of the file allocation table.

- Rather, the file index for a file is kept in a separate block, and the entry for the file in the file allocation table points to that block.

Indexed Allocation Method

Allocation may be on the basis of either

- fixed-size blocks or
- variable-size portions

Allocation by blocks eliminates external fragmentation,

- whereas allocation by variable-size portions improves locality.

In either case, file consolidation may be done from time to time.

- File consolidation reduces the size of the index in the case of variable-size portions, but not in the case of block allocation.

Indexed allocation with Block Portions

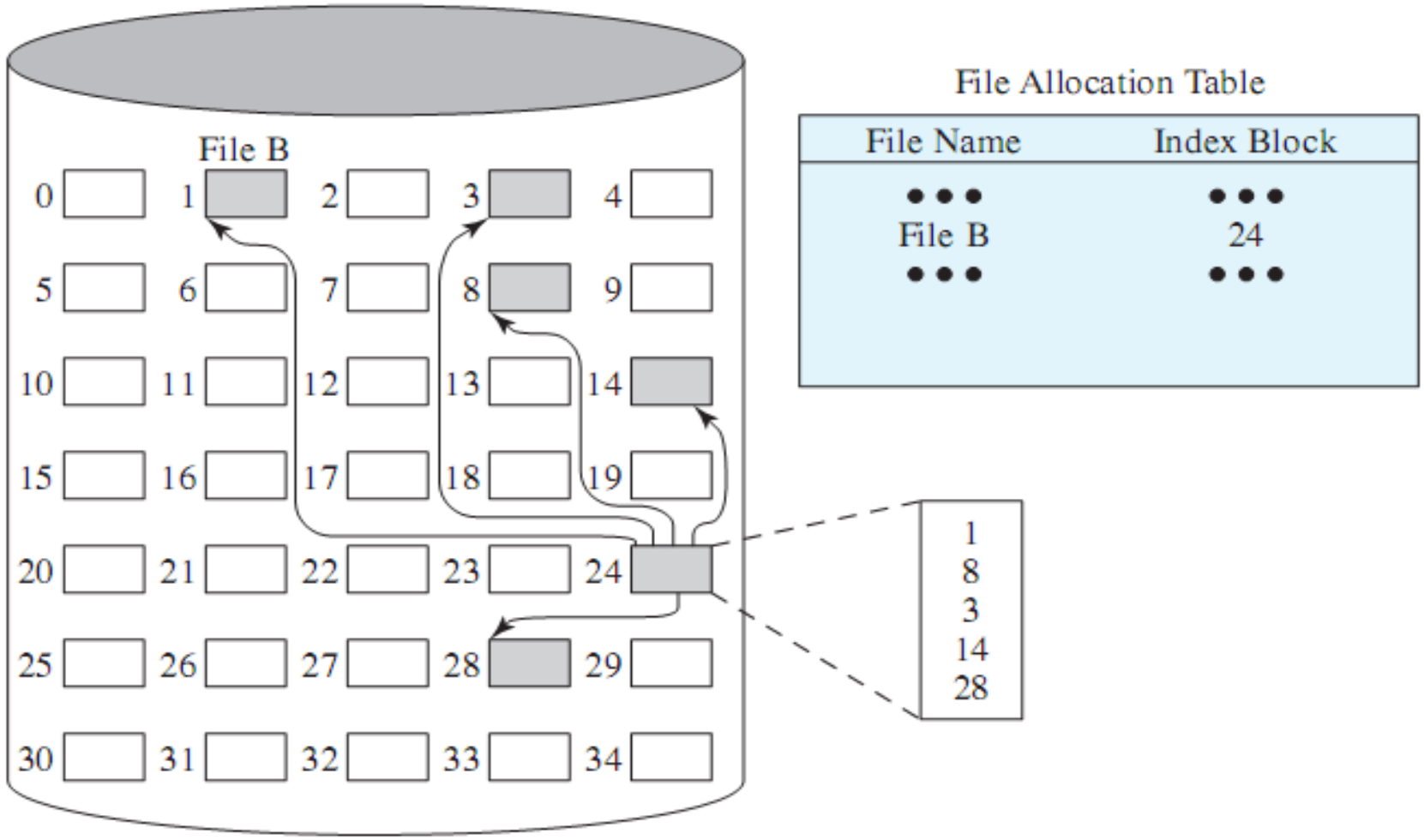


Figure 12.11 Indexed Allocation with Block Portions

Indexed Allocation with Variable Length Portions

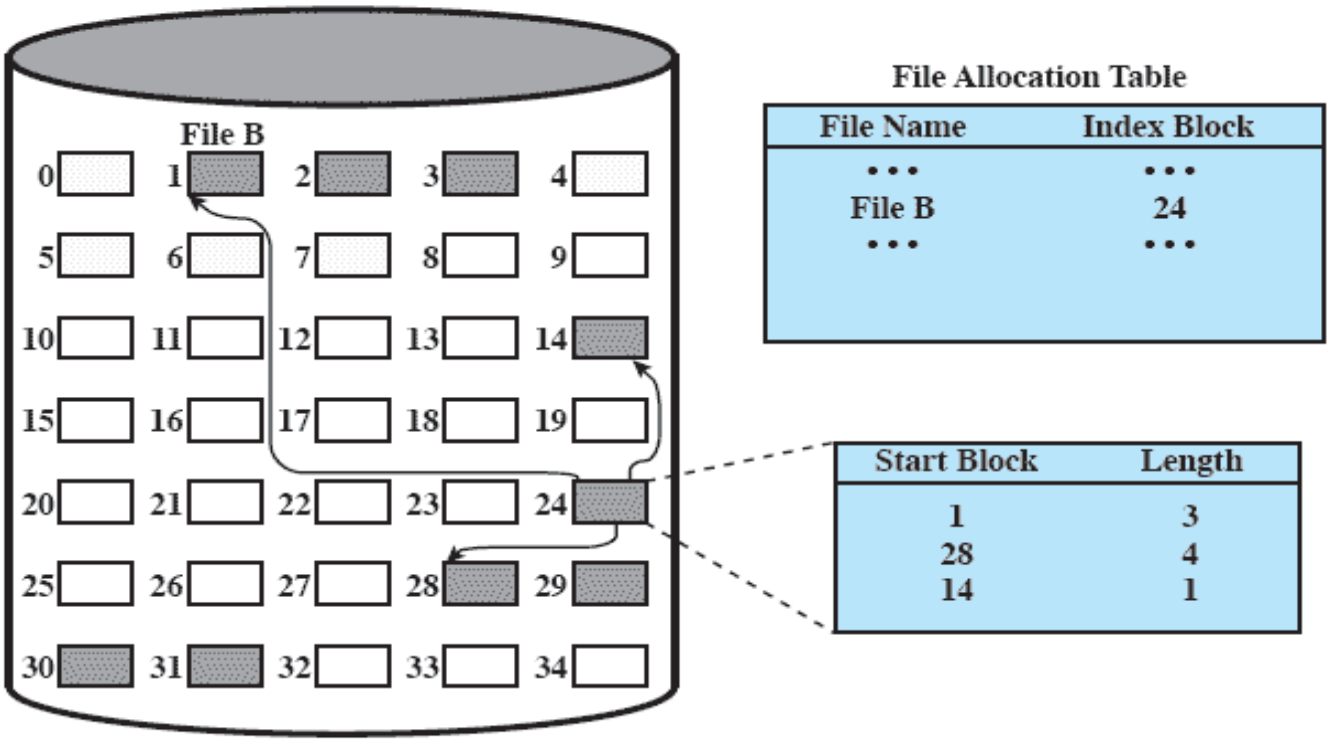


Figure 12.12 Indexed Allocation with Variable-Length Portions

UNIT 5 COMPLETED