Unit-4 part-2
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.

- Desirable properties include
  - **Long-term existence:**
    - Files are stored on disk or other secondary storage and do not disappear when a user logs off.
  - **Sharable between processes:**
    - Files have names and can have associated access permissions that permit controlled sharing.
  - **Structure:**
    - Depending on the file system, a file can have an internal structure that is convenient for particular applications.
    - In addition, files can be organized into hierarchical or more complex structure to reflect the relationships among files.

FILE MANAGEMENT

File management system consists of system utility programs that run as privileged applications concerned 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 STRUCTURE-TERMS

Four terms are in common use when discussing file

- Field
- Record
- File
- Database

Fields and Records

- A **field** is the basic element of data.
  - It is characterized by its length and data type (e.g. ASCII string, decimal).
  - Depending on the file design, fields may be fixed length or variable length.
- A **record** is a collection of related fields that can be treated as a unit by some application program.

FILE AND DATABASE

- *A* **file** is a collection of similar records.
  - The file is treated as a single entity by users and applications and may be referenced by name.
  - Files have file names and may be created and deleted.
  - Access control restrictions usually apply at the file level.

- *A* **database** is a collection of related data.
  - Explicit relationships exist among elements of data
  - The database itself consists of one or more types of files.
  - Usually, there is a separate database management system that is independent of the operating system, although that system may make use of some file management programs.

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 SYSTEM ARCHITECTURE

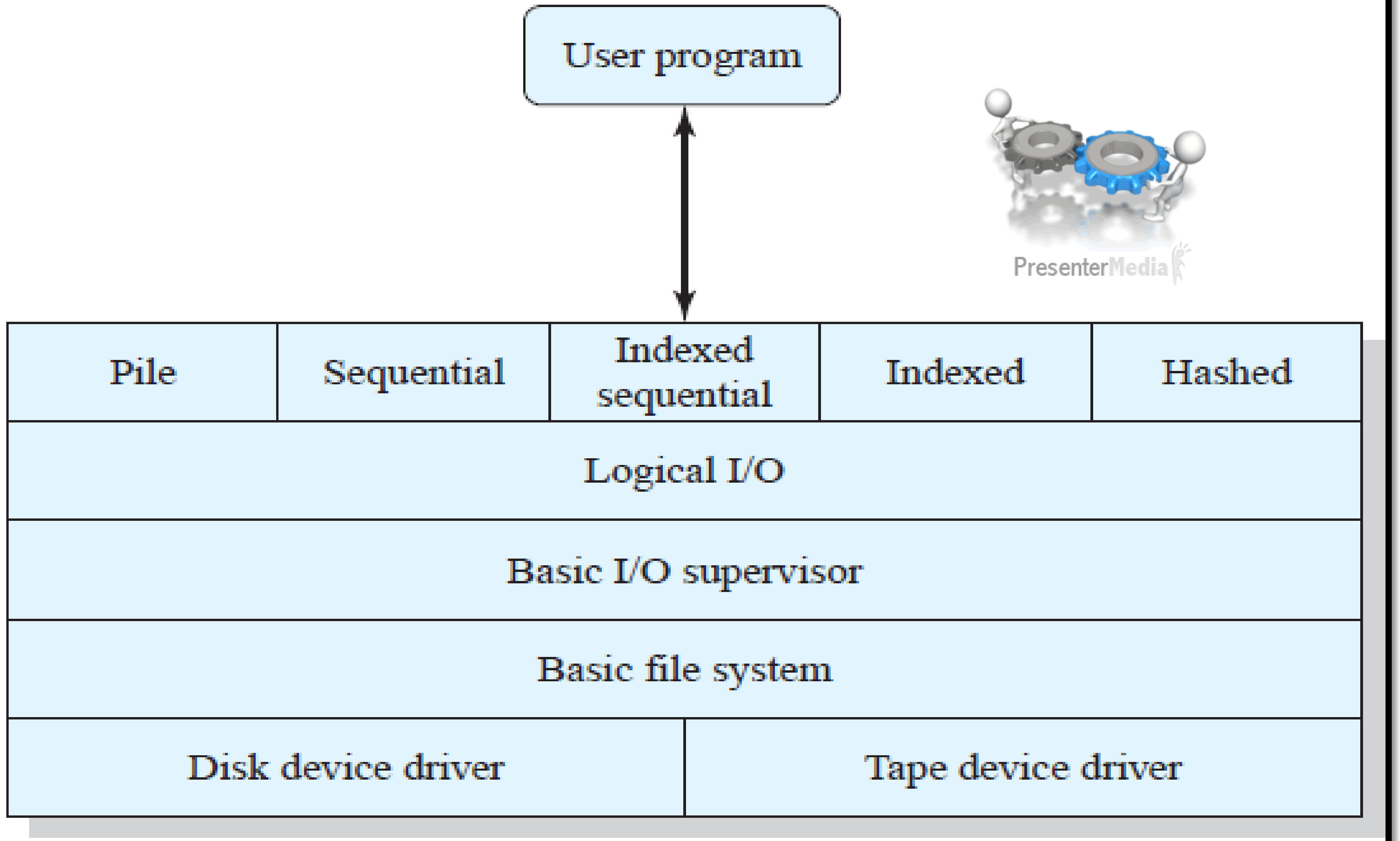


Figure 12.1 **File System Software Architecture**

File System Architecture

- One way of getting a feel for the scope of file management is to look at a depiction of a typical software organization, as suggested in Figure 12.1.
- Different systems will be organized differently, but this organization is reasonably representative.

DEVICE DRIVERS

device drivers communicate at the lowest level directly with peripheral devices or their controllers or channels.

- A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request.
- For file operations, the typical devices controlled are disk and tape drives.
- Device drivers are usually considered to be part of the operating system.

Basic File System

basic file system, or **the physical I/O level.**

This is the primary interface with the environment outside of the computer system.

It deals with blocks of data that are exchanged with disk or tape systems.

BASIC I/O SUPERVISOR

- Basic I/O supervisor is responsible for all file I/O initiation and termination.
- Control structures are maintained that deal with
  - device I/O,
  - scheduling, and
  - file status.

LOGICAL I/O

- **Logical I/O** enables users and applications to access records.
- Whereas the basic file system deals with blocks of data,
  - the logical I/O module deals with file records.
- Logical I/O provides a general-purpose record I/O capability
  - and maintains basic data about files.

Access Method

- Closest to the user
- Reflect different file structures
- Provides a standard interface between applications and the file systems and devices that hold the data
- Access method varies depending on the ways to access and process data for the device.

FILE MANAGEMENT FUNCTIONS

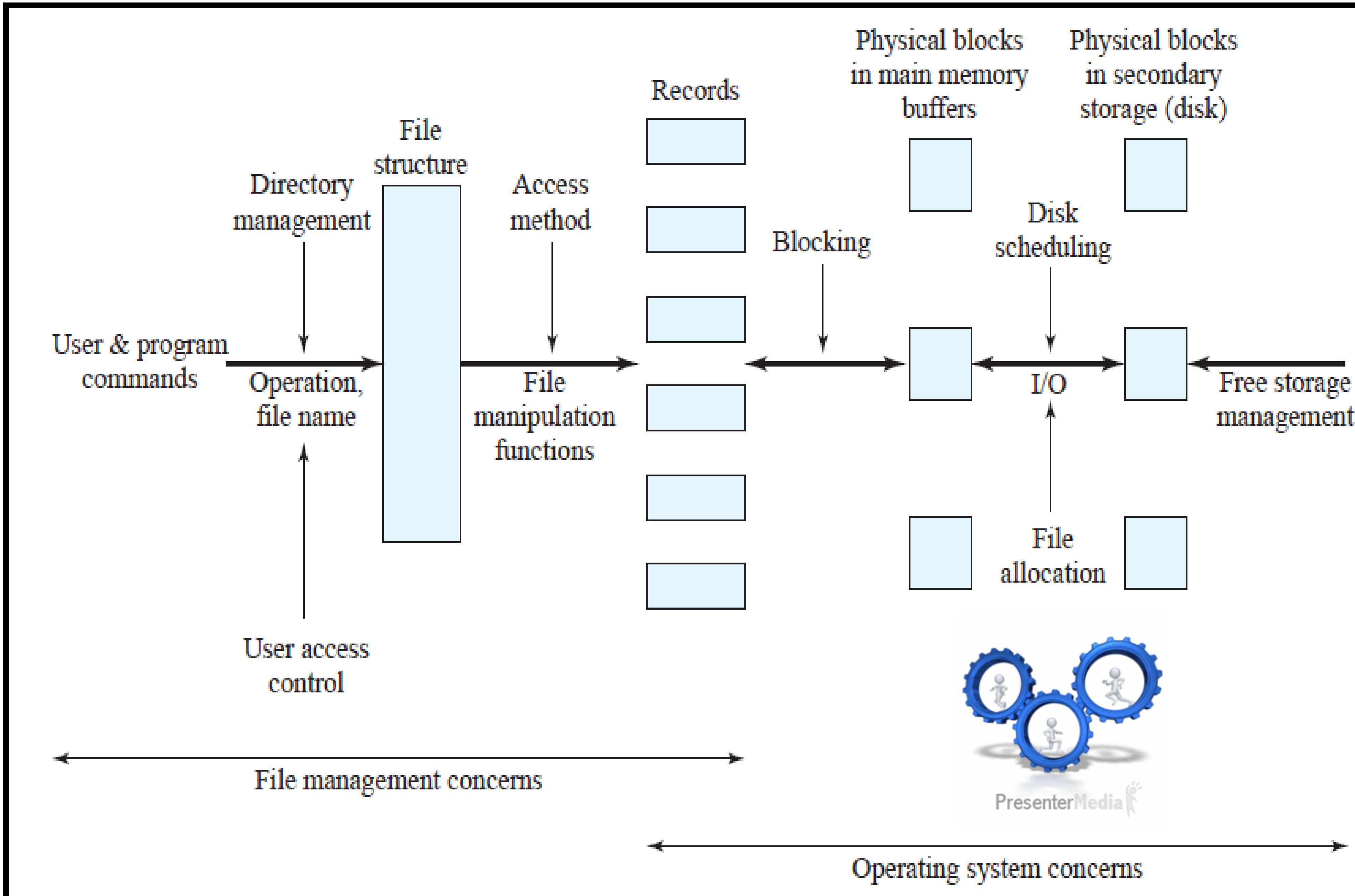


Figure 12.2 Elements of File Management

ROADMAP

Overview

➡ **File organisation and Access**

File Directories

File Sharing

Record Blocking

Secondary Storage Management

FILE ORGANIZATION

File Management Referring to the logical structure of records

- Physical organization discussed later

Determined by the ***way*** in which files are accessed

Criteria for File Organization

- Important criteria include:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

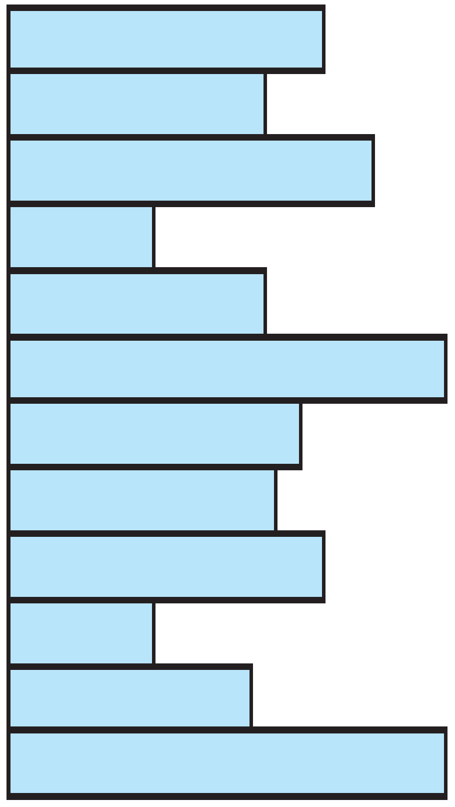
FILE ORGANISATION TYPES

Many exist, but usually variations of:

- Pile
- Sequential file
- Indexed sequential file
- Indexed file
- Direct, or hashed, file

THE PILE

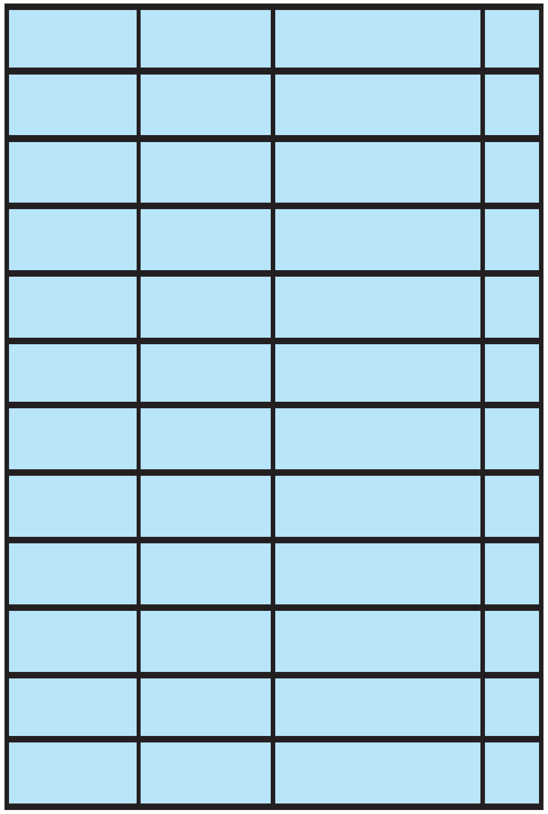
- The least-complicated form of file organization may be termed the pile.
- Data are collected in the order in which they arrive.
- Each record consists of one burst of data.
- The purpose of the pile is simply to accumulate the mass of data and save it.
- Records may have different fields, or similar 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.



(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

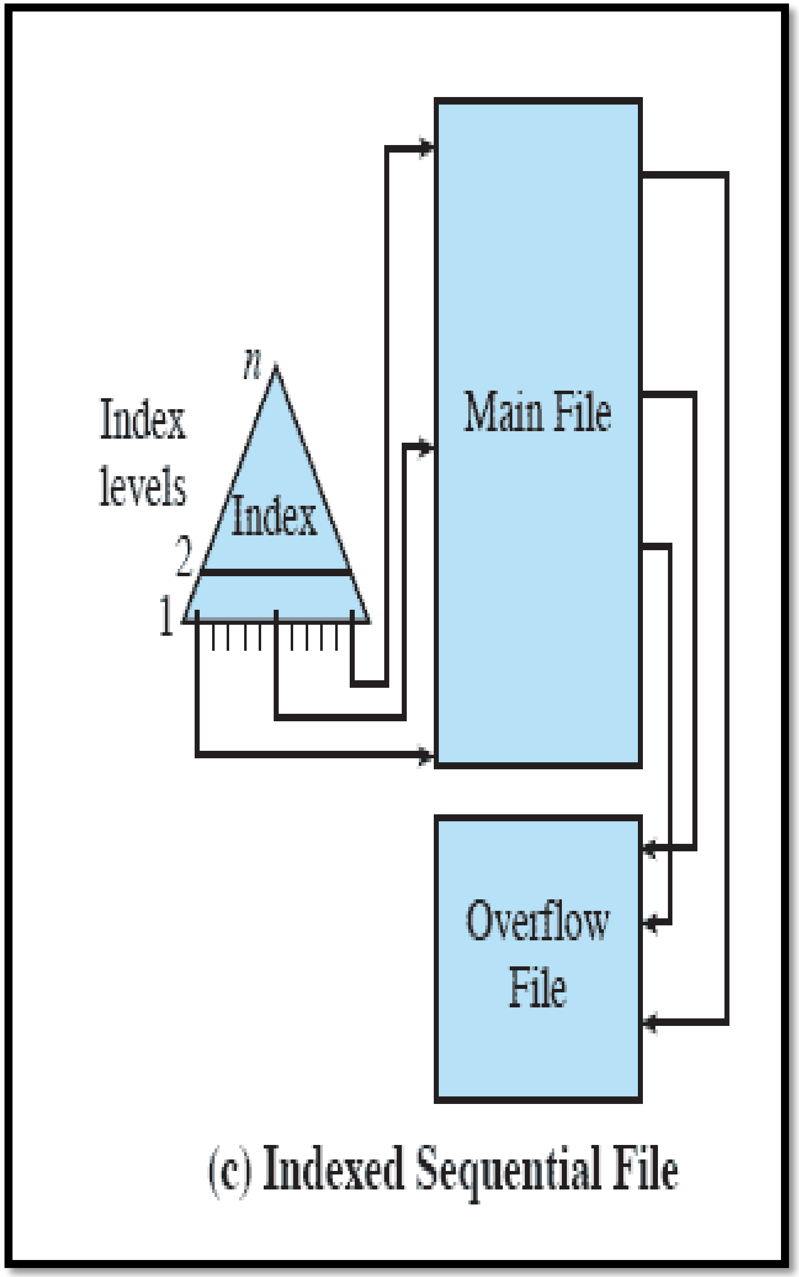
INDEXED SEQUENTIAL FILE

- 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

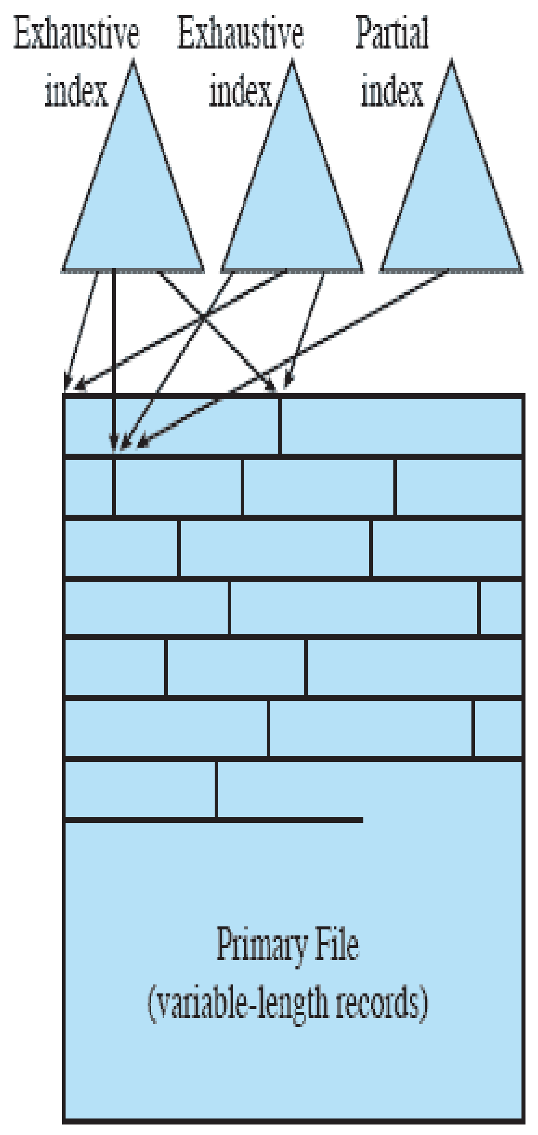
INDEXED FILE

- 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

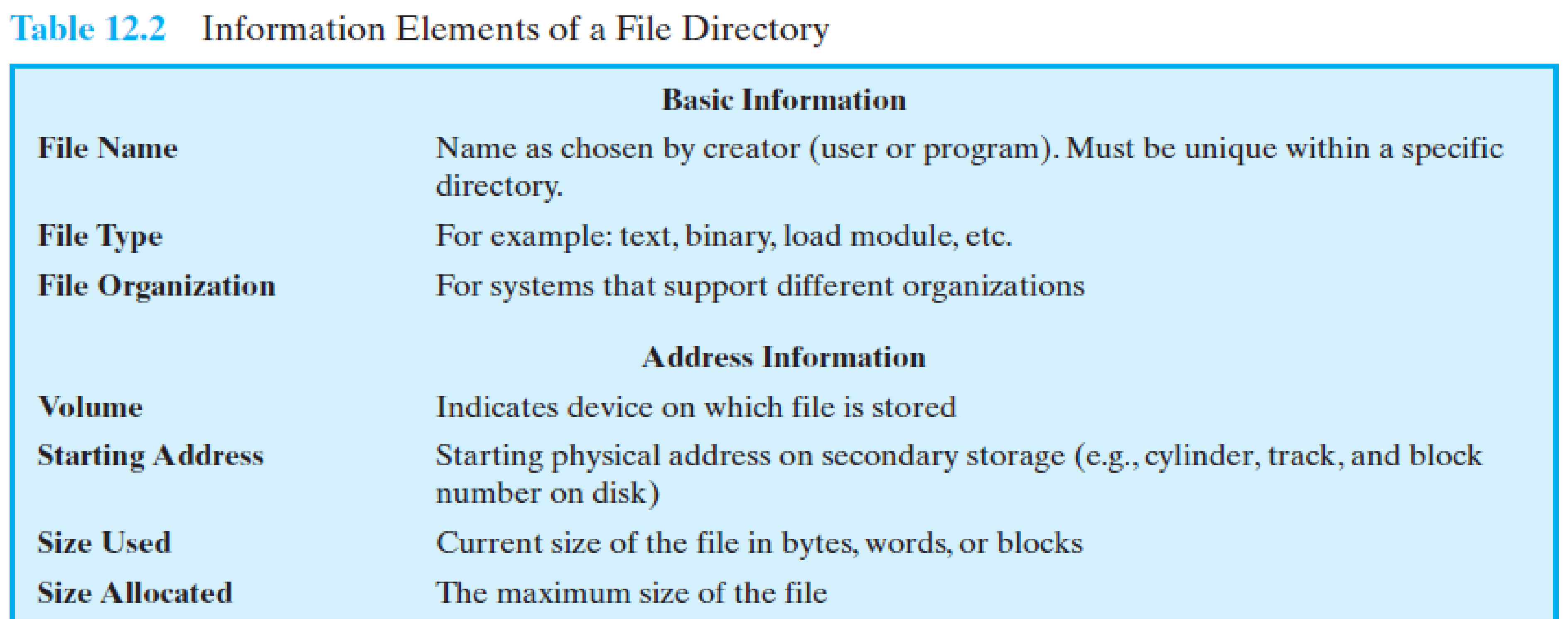


Table 12.2 Information Elements of a File Directory

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.

TWO-LEVEL SCHEME FOR A DIRECTORY

•In this case, there is one directory for each user, and a master directory.

- The master directory has an entry for each user directory, providing address and access control information.
- Each user directory is a simple list of the files of that user.

• This arrangement means that names must be unique only within the collection of files of a single user, and that the file system can easily enforce access restriction on directories.

• However, it still provides users with no help in structuring collections of files.

HIERARCHICAL, OR TREE-STRUCTURED DIRECTORY

- There is a master directory, which has under it a number of user directories.

- Each of these user directories, in turn, may have subdirectories and files as entries.
- The simplest approach is to store each directory as a sequential file.

- When directories may contain a very large number of entries, such an organization may lead to unnecessarily long search times.
  - If so, a hashed structure is preferred.

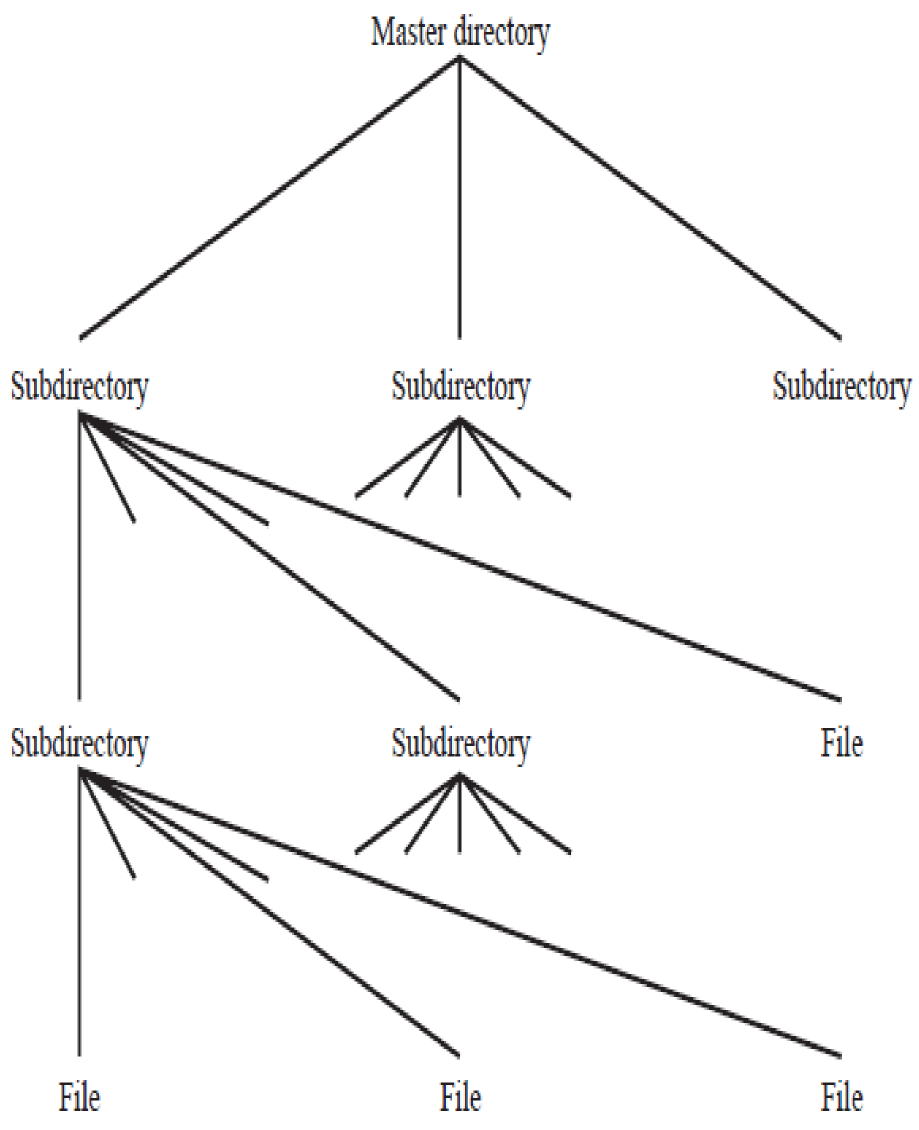


Figure 12.4 Tree-Structured Directory

NAMING

- Users need to be able to refer to a file by a symbolic name.
  - Each file in the system must have a unique name in order that file references be unambiguous.
  - But it is an unacceptable burden on users to require that they provide unique names, especially in a shared system.

- The use of a tree-structured directory minimizes the difficulty in assigning unique names.
  - Any file in the system can be located by following a path from the root or master directory down various branches until the file is reached.
  - The series of directory names, culminating in the file name itself, constitutes a pathname for the file.

EXAMPLE OF TREE-STRUCTURED DIRECTORY

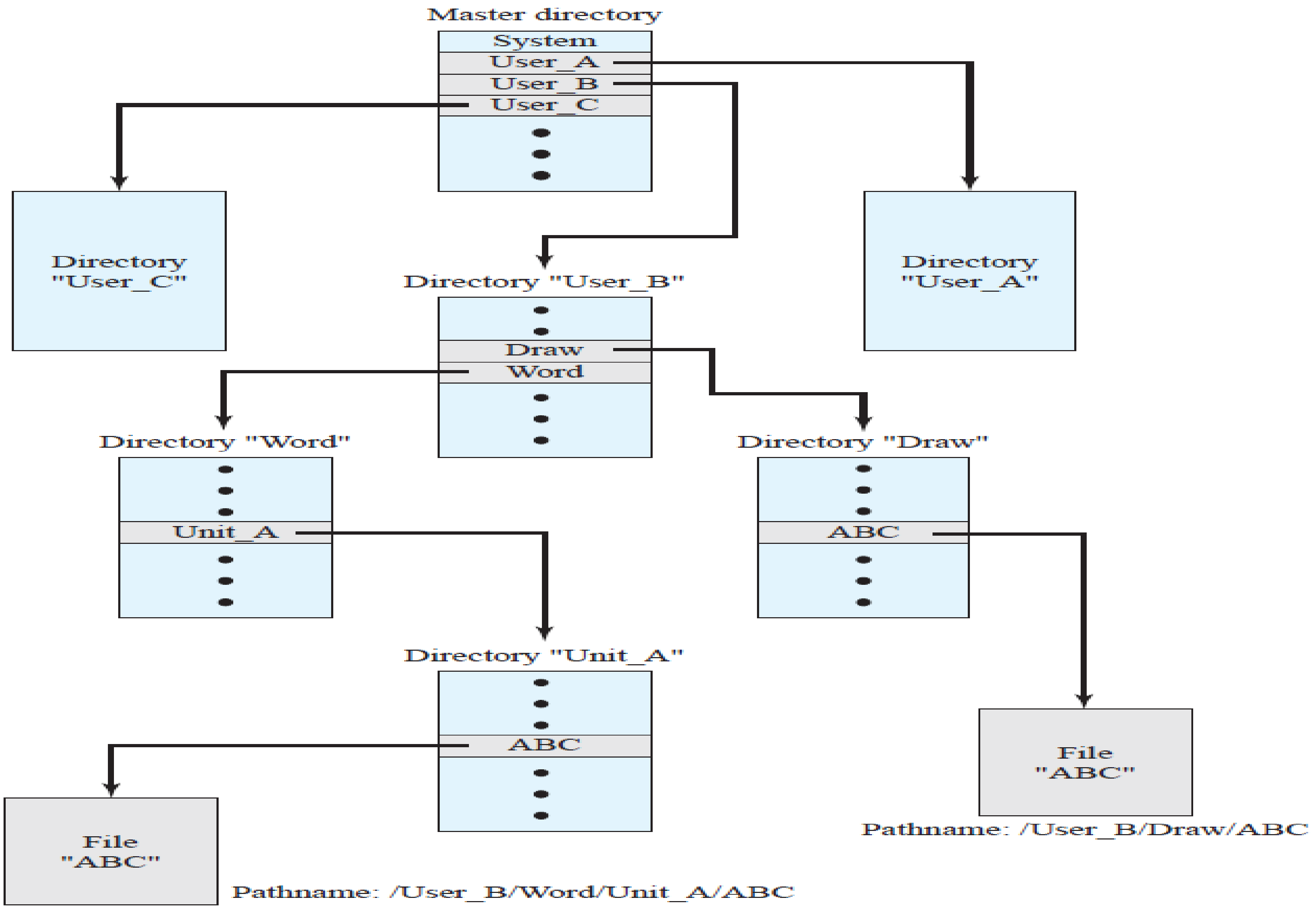


Figure 12.5 Example of Tree-Structured Directory

WORKING DIRECTORY

- Stating the full pathname and filename is awkward and tedious
- An interactive user or a process has associated with it a current directory, often referred to as the **working directory**.
- Files are then referenced relative to the working directory.
- For example, if the working directory for user B is “Word,”
- Then the pathname Unit_A /ABC is sufficient to identify the file

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.

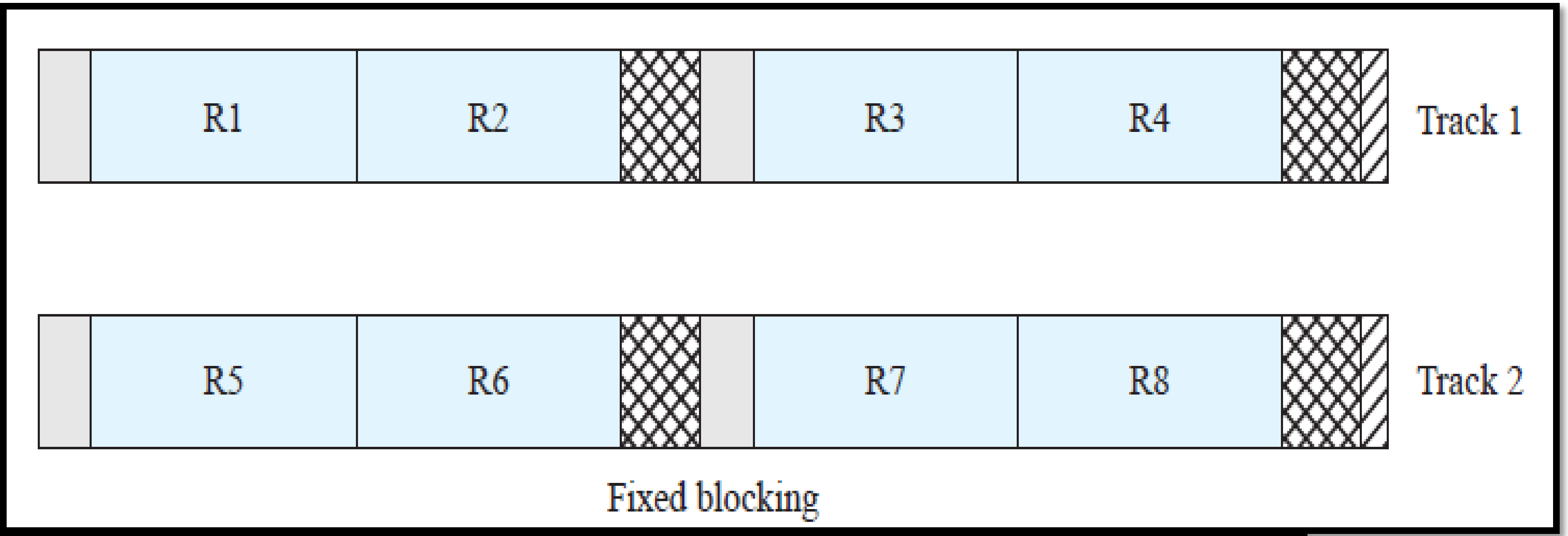
Three approaches are common

- 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***



🟦	Data	▧	Waste due to record fit to blocksize
⬜	Gaps due to hardware design	▦	Waste due to blocksize constraint from fixed record size
▨	Waste due to block fit to track size		

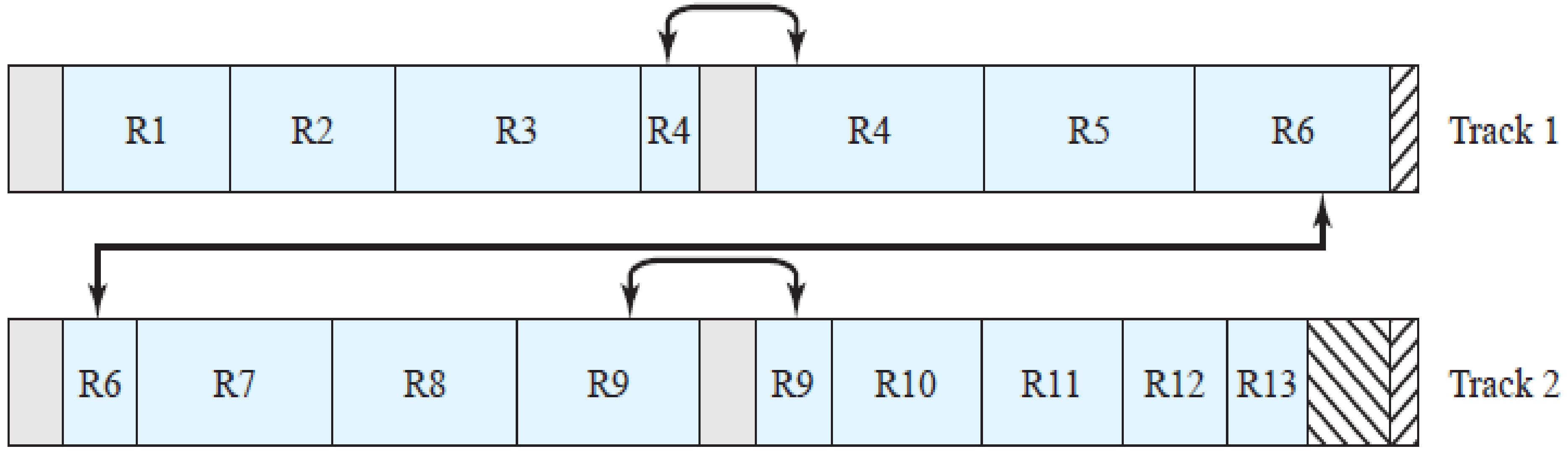
Figure 12.6 Record Blocking Methods [WIED87]

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

🟦	Data	▧	Waste due to record fit to blocksize
⬜	Gaps due to hardware design	▦	Waste due to blocksize constraint from fixed record size
▨	Waste due to block fit to track size		

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.

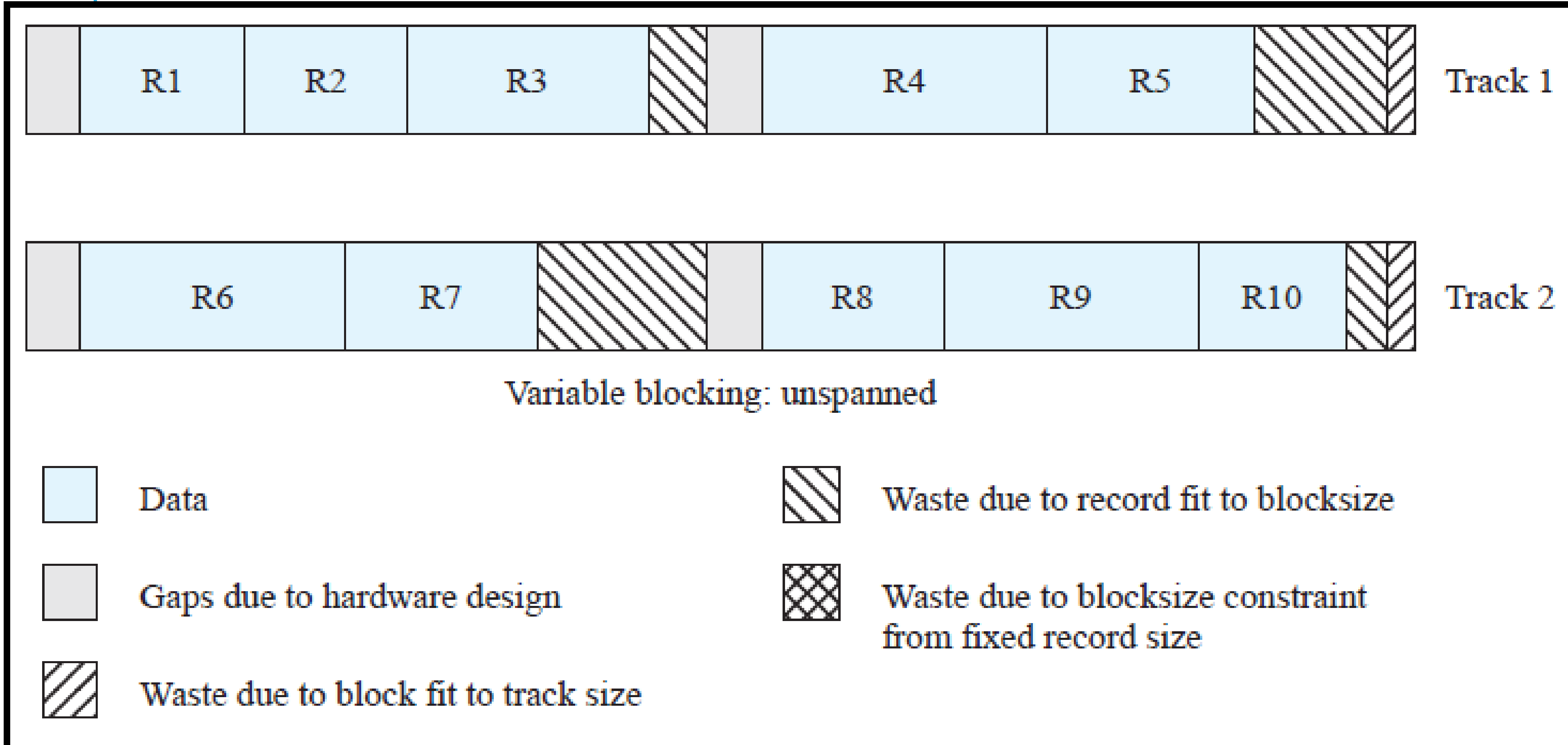

R1
R2
R3
R4
R5
Track 1
R6
R7
R8
R9
R10
Track 2
Variable blocking: unspanned
Data
Gaps due to hardware design
Waste due to block fit to track size
Waste due to record fit to blocksize
Waste due to blocksize constraint from fixed record size

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.

PORTION SIZE

Two extremes:

- Portion large enough to hold entire file is allocated
- Allocate space one block at a time

Trade-off between efficiency from the point of view of a single file, or the overall system efficiency

FILE ALLOCATION METHOD

Three methods are in common use:

- contiguous,
- chained, and
- indexed.

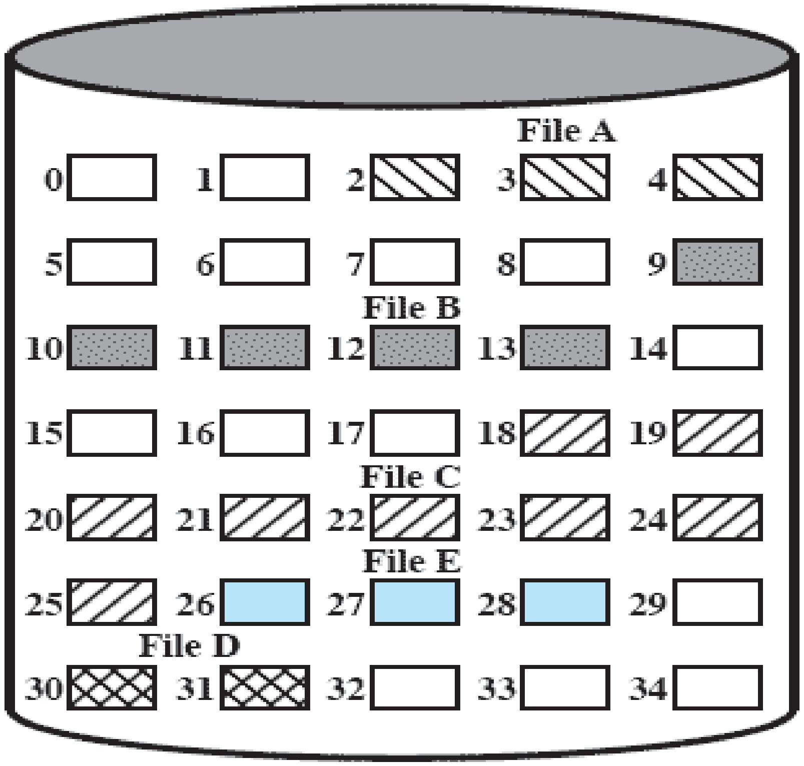
CONTIGUOUS ALLOCATION

- 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

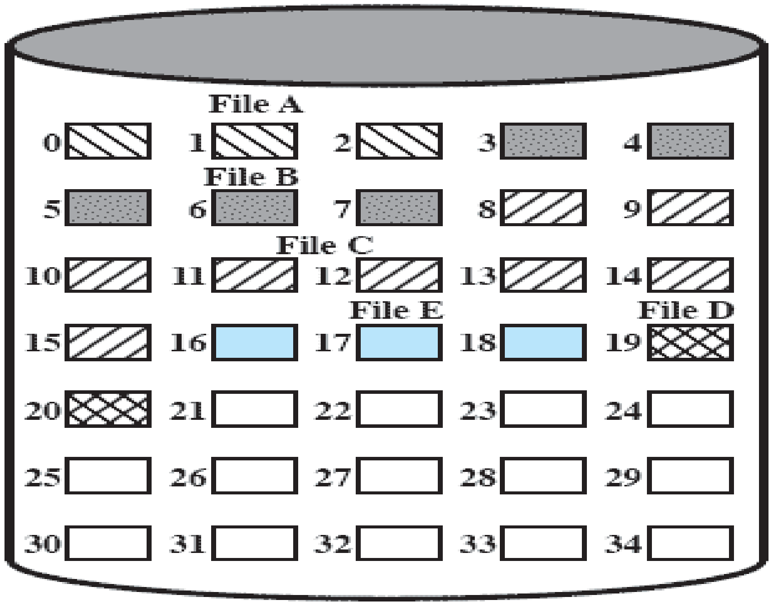


File Allocation Table

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.

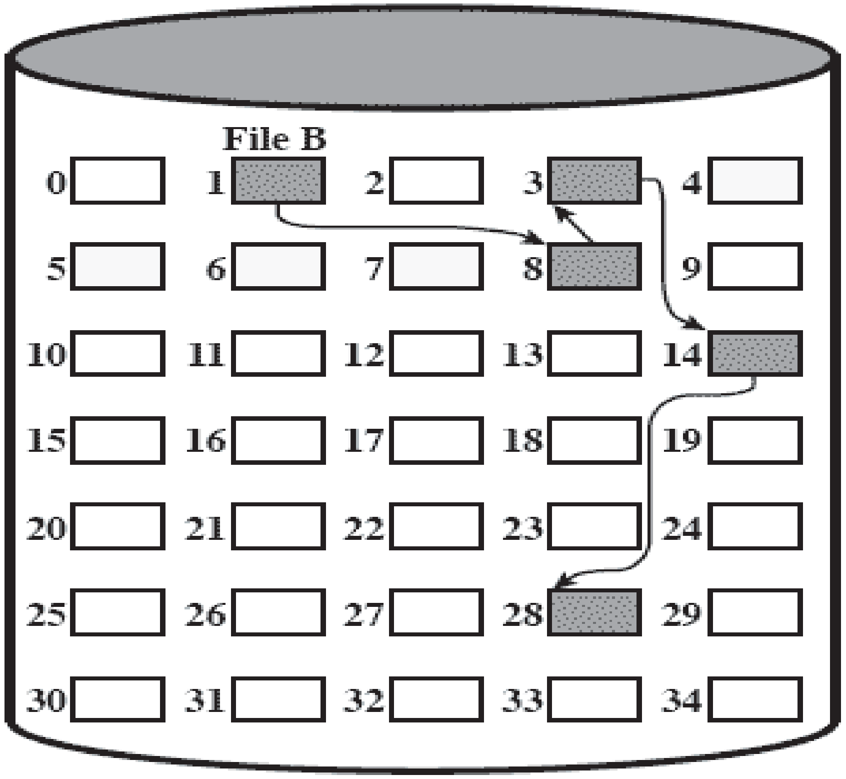
Although preallocation is possible, it is more common simply to allocate blocks as needed.

- 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

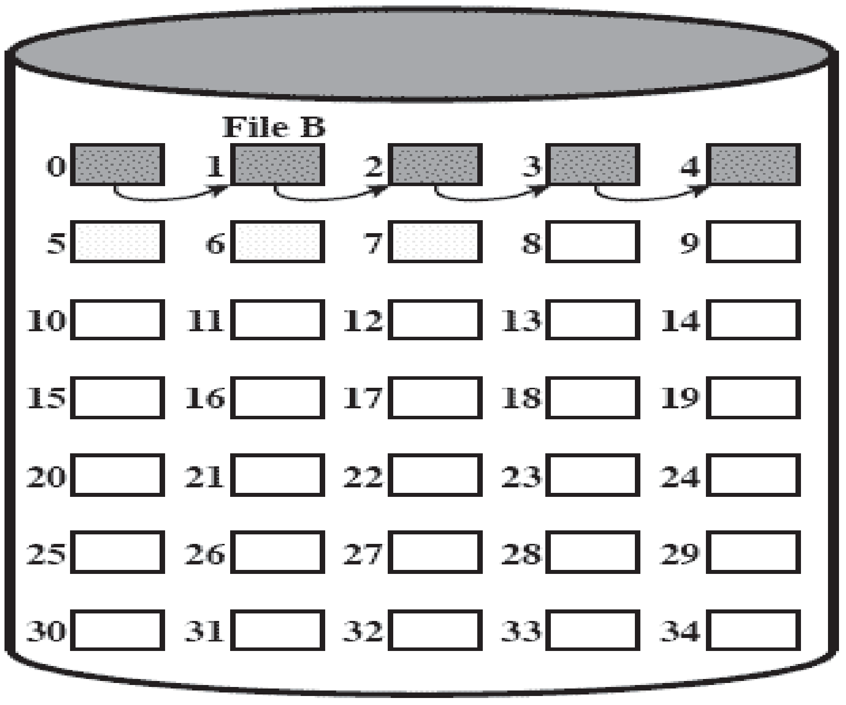


File Allocation Table

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

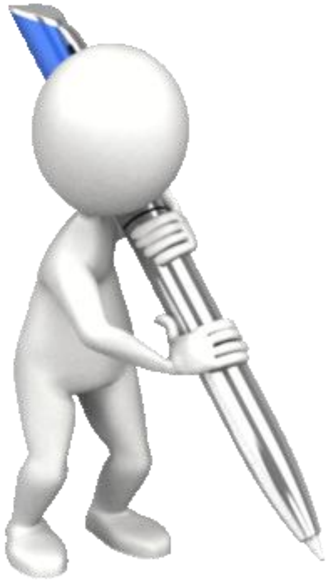
- fixed-size blocks or
- variable-size portions

Allocation by blocks eliminates external fragmentation,

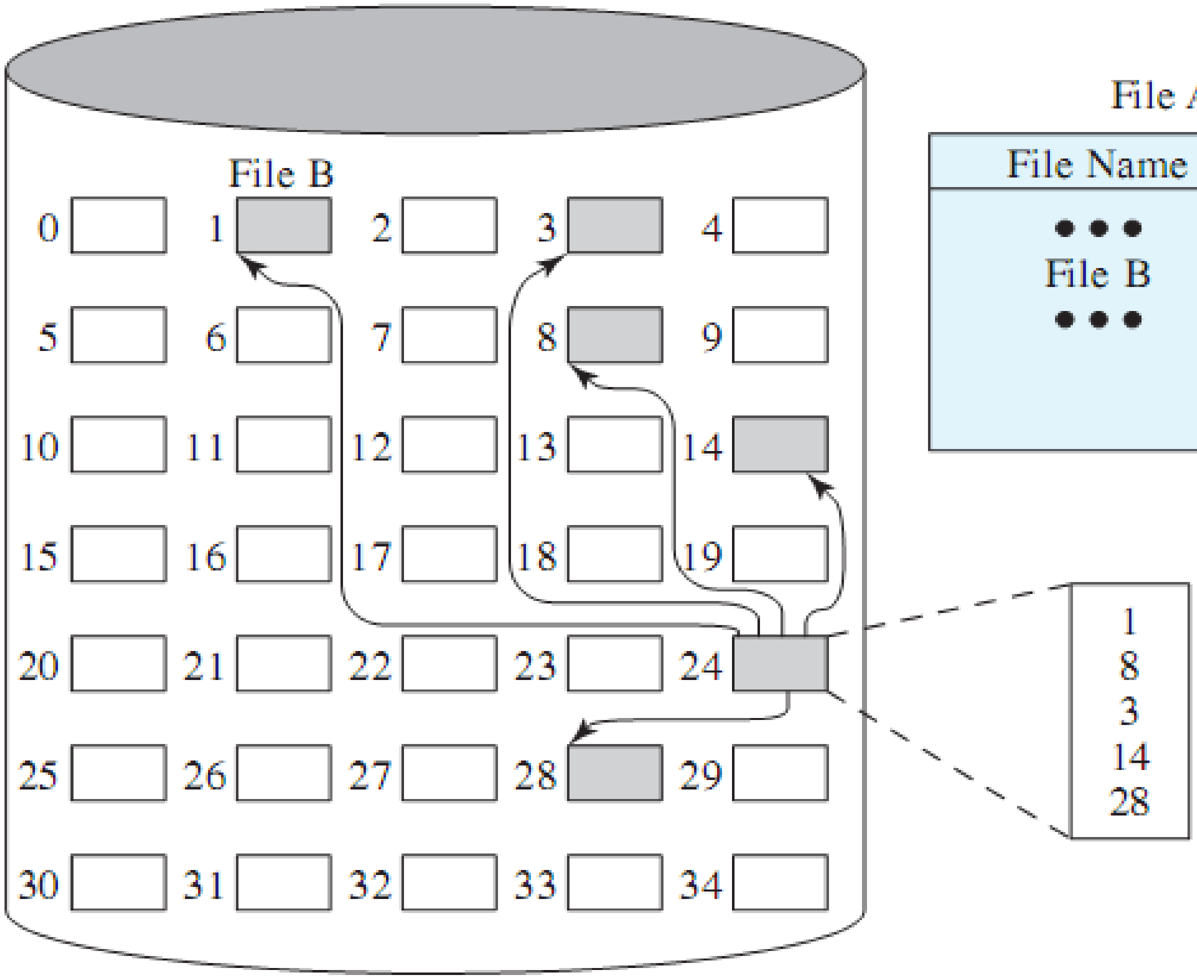
- whereas allocation by variable-size portions improves locality.

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



File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

Figure 12.11 **Indexed Allocation with Block Portions**

INDEXED ALLOCATION WITH VARIABLE LENGTH PORTIONS

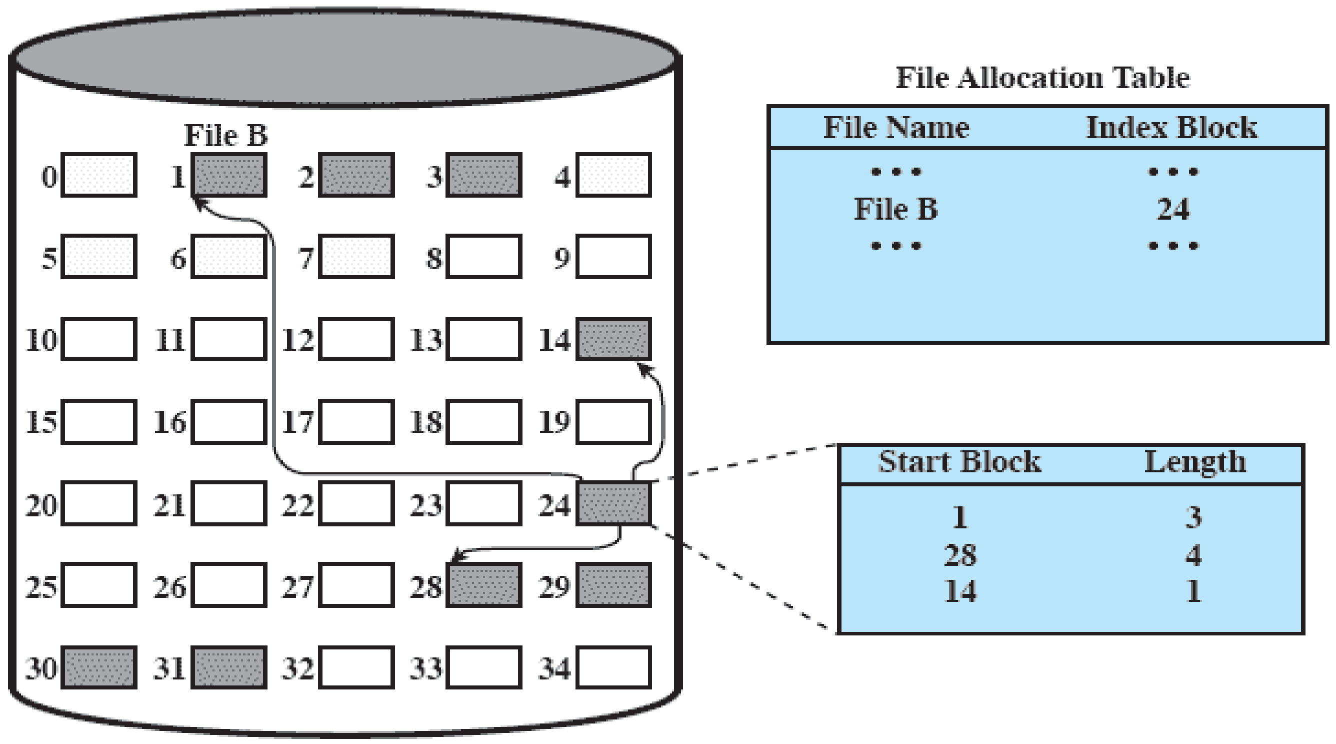


Figure 12.12 Indexed Allocation with Variable-Length Portions

FREE SPACE MANAGEMENT

Just as allocated space must be managed, so must the unallocated space

To perform file allocation, we need to know which blocks are available.

We need a disk allocation table in addition to a file allocation table.

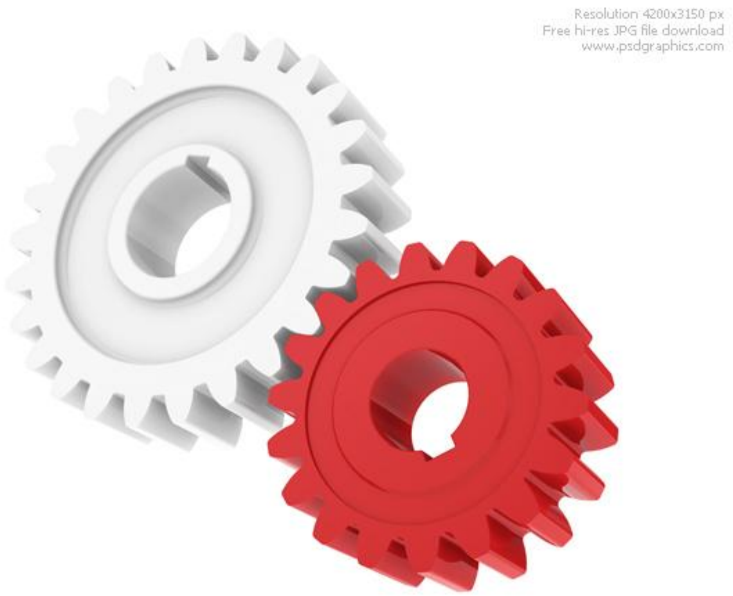
A number of techniques that have been implemented.

(1) Bit Tables

(2) Chained Free Portions

(3) Indexing

(4) Free Block List



(1)BIT TABLES

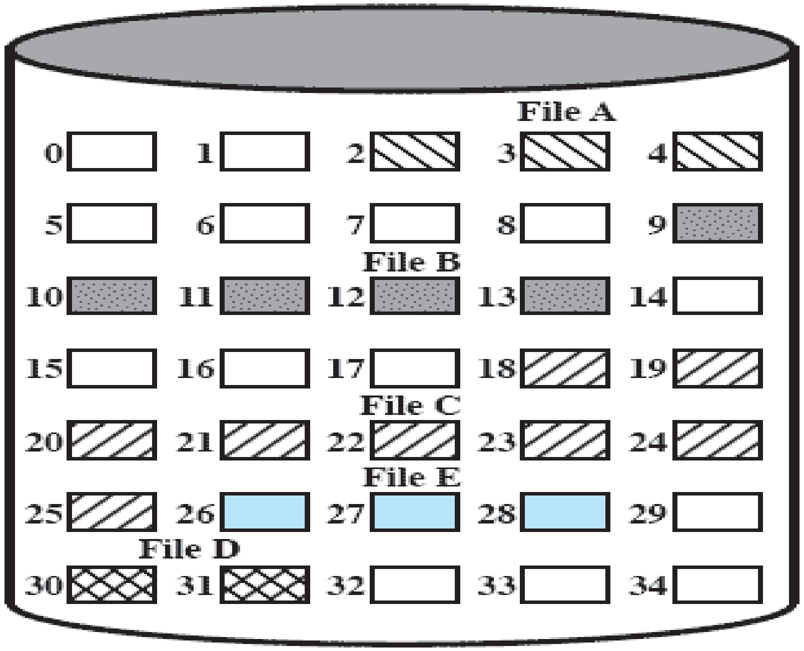
This method uses a vector containing one bit for each block on the disk.

- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use.

A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks.

- Thus, a bit table works well with any of the file allocation methods outlined.
- Another advantage is that it is as small as possible.

(1)Bit Tables



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

For example, for the disk layout of Figure 12.7, a vector of length 35 is needed and would have the following value:
00111000011111000011111111111011000

(2)CHAINED FREE PORTIONS

- The free portions may be chained together by using a pointer and length value in each free portion.
- This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first portion.

- This method is suited to all of the file allocation methods.
  - If allocation is a block at a time, simply choose the free block at the head of the chain and adjust the first pointer or length value.
  - If allocation is by variable-length portion, a first-fit algorithm may be used:
  - The headers from the portions are fetched one at a time to determine the next suitable free portion in the chain.
  - Again, pointer and length values are adjusted.

- This method has its own problems.
  - After some use, the disk will become quite fragmented and many portions will be a single block long.
  - Also note that every time you allocate a block, you need to read the block first to recover the pointer to the new first free block before writing data to that block.
  - If many individual blocks need to be allocated at one time for a file operation, this greatly slows file creation
  - Similarly, deleting highly fragmented files is very time consuming.

(3)INDEXING

Treats free space as a file and uses an index table as it would for file allocation

For efficiency, the index should be on the basis of variable-size portions rather than blocks.

- Thus, there is one entry in the table for every free portion on the disk.

This approach provides efficient support for all of the file allocation methods.

(4)FREE BLOCK LIST

- In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved portion of the disk.
- Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number, so the size of the free block list is 24 or 32 times the size of the corresponding bit table and thus must be stored on disk rather than in main memory.

VOLUMES

A collection of addressable sectors in secondary memory that an OS or application can use for data storage.

The sectors in a volume need not be consecutive on a physical storage device;

- instead they need only appear that way to the OS or application.

A volume may be the result of assembling and merging smaller volumes.