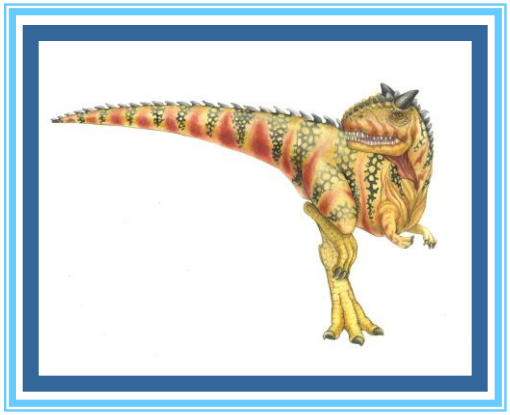
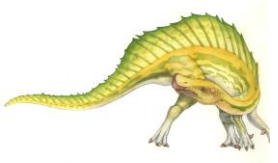
Chapter 13: File-System Interface



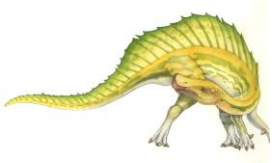
Outline

- File Concept
- Access Methods
- Disk and Directory Structure



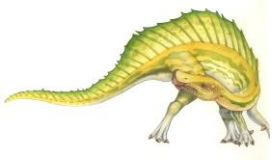
Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures



File Concept

- Contiguous logical address space
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents defined by file's creator
  - Many types
    - **text file,** A text file is a file without any formatting or fonts, and has a .txt extension in Windows.You can create a text file on your computer with a text editor. An example of a text editor is Notepad, which is included with Microsoft Windows.
    - **source file**, A .SOURCE file is a source code file that contains text written in one or more programming languages.
    - **executable file**n executable file is a type of computer file that runs a program or performs a specific operation on a computer. Unlike data files, executable files contain compiled code that the computer's CPU can directly execute. Common examples of executable files include .EXE, .BAT, .COM, and .BIN on IBM-compatible computers, and .DMG and .APP on Apple Mac computers.

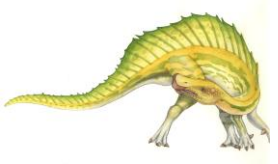


File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

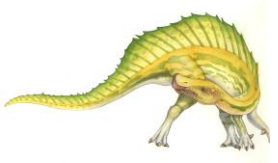
File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open*** **(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- ***Close*** **(F_i)** – move the content of entry F_i in memory to directory structure on disk



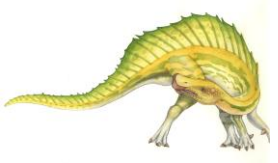
Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information



File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

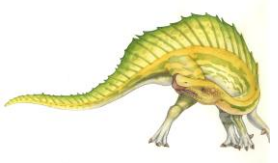


File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- **Complex Structures:**

1- **Formatted document**: File formats that store data in structured, semi-structured, or unstructured forms, such as JSON, XML, SQL, images, and audio

2- **Relocatable load file**: holds code and data suitable for linking with other object files to obtain executable file.



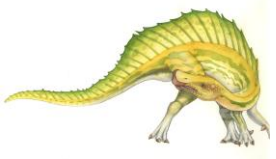
File structure types are:

Text files: Series of characters organized in lines.

Object files: Series of bytes organized into blocks.

Source files: Series of functions and processes.

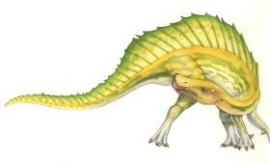
File systems: Methods of organizing and storing files on a disk.
Examples include FAT, NTFS, ext, and HFS.



Access Methods

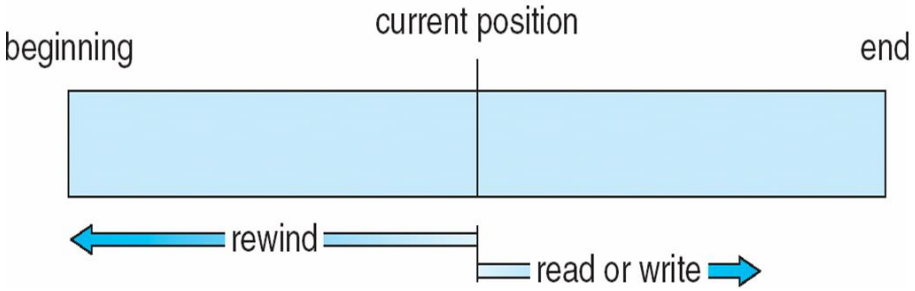
- A file is fixed length **logical records**
- **Sequential Access**
- **Direct Access**
- **Other Access Methods**

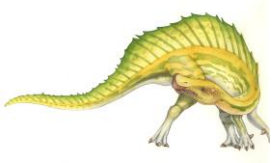
- **Sequential files** store data in a linear order, suitable for batch processing.
- **Indexed sequential files** add indexes for faster access. Direct access files allow immediate access to any data record.
- **Hierarchical structures**, such as those found in databases, organize data in a tree-like format.
- The choice of file structure depends on access patterns and data requirements, balancing speed, storage space, and complexity



Sequential Access

- Operations
  - **`read next`**
  - **`write next`**
  - **`Reset`**
  - no read after last write (rewrite)
- Figure



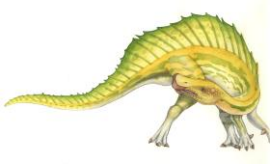


Direct Access

- Operations
  - **`read n`**
  - **`write n`**
  - **`position to n`**
    - **`read next`**
    - **`write next`**
    - **`rewrite n`**

  n = **relative block number**

- Relative block numbers allow OS to decide where file should be placed

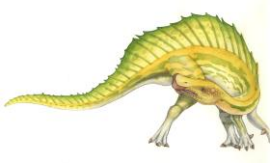


Other Access Methods

- Can be other access methods built on top of base methods
- General involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider Universal Produce Code (UPC code) plus record of data about that item)
- If the index is too large, create an in-memory index, which an index of a disk index
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS

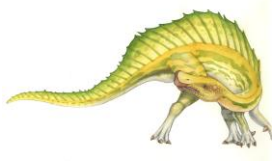
Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **volume**
- Each volume containing a file system also tracks that file system's info in **device directory** or **volume table of contents**
- In addition to **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer



Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system



Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

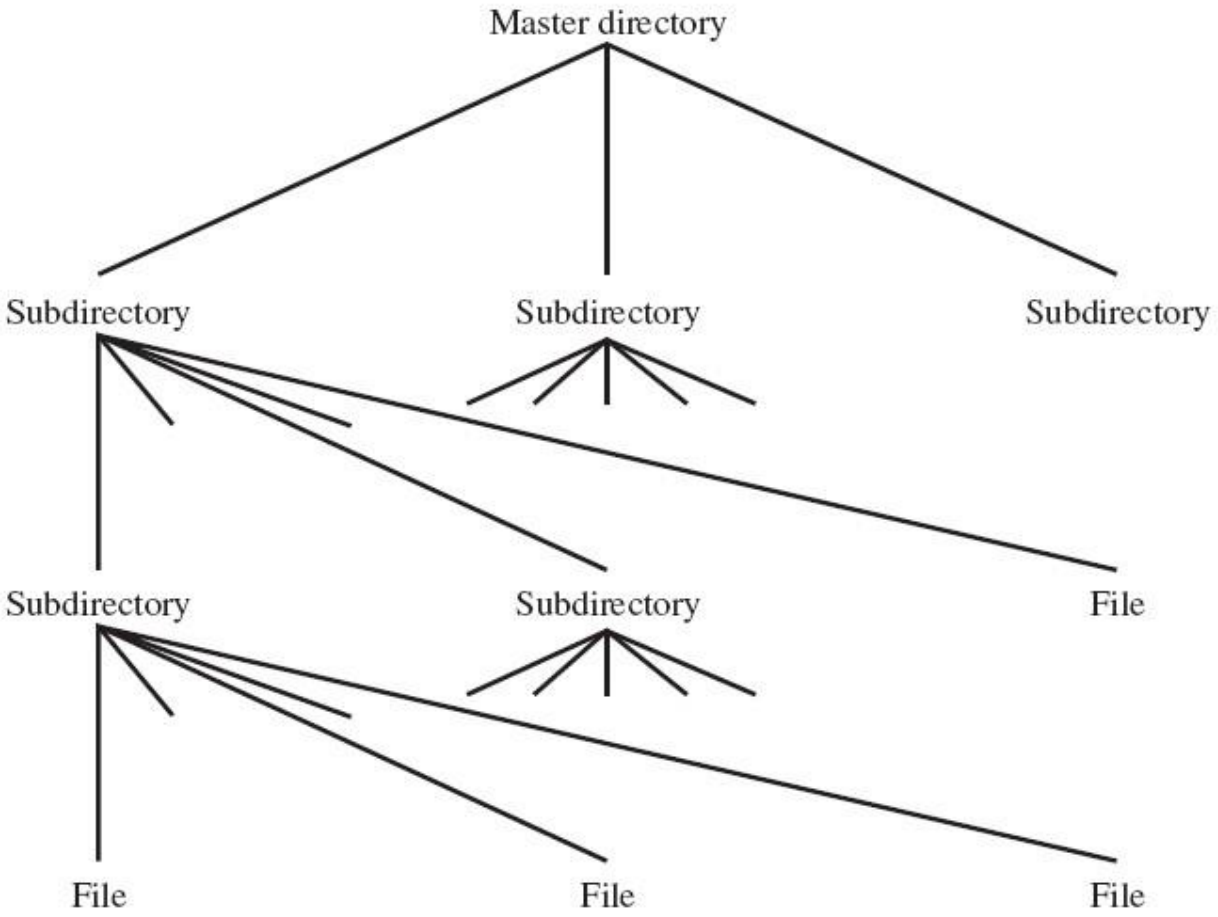
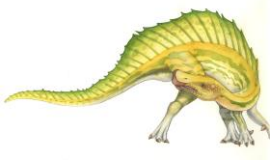


Figure Tree-Structured Directory

Protection

- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **None**
  - **Knowledge**
  - **Write**
  - **Execute**
  - **updating**
  - **Append**
  - **Change protection**
  - **Delete**
  - **List**

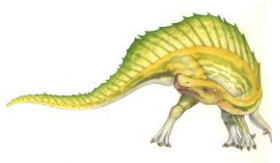


Access Rights

Access Rights

- The following list is representative of **access rights** that can be assigned to a particular user for a particular file:

- **None:** The user may not even learn of the existence of the file, much less access it. To enforce this restriction, the user would not be allowed to read the user directory that includes this file.
- **Knowledge:** The user can determine that the file exists and who its owner is. The user is then able to petition the owner for additional access rights.
- **Execution:** The user can load and execute a program but cannot copy it. Proprietary programs are often made accessible with this restriction.
- **Reading:** The user can read the file for any purpose, including copying and execution. Some systems are able to enforce a distinction between viewing and copying. In the former case, the contents of the file can be displayed to the user, but the user has no means for making a copy.

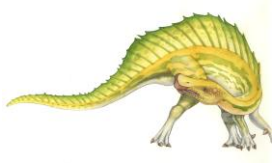


Access Rights

- **Appending:** The user can add data to the file, often only at the end, but cannot modify or delete any of the file's contents. This right is useful in collecting data from a number of sources.
- **Updating:** The user can modify, delete, and add to the file's data. This normally includes writing the file initially, rewriting it completely or in part, and removing all or a portion of the data. Some systems distinguish among different degrees of updating.
- **Changing protection:** The user can change the access rights granted to other users. Typically, this right is held only by the owner of the file. In some systems, the owner can extend this right to others. To prevent abuse of this mechanism, the file owner will typically be able to specify which rights can be changed by the holder of this right.
- **Deletion:** The user can delete the file from the file system.

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.

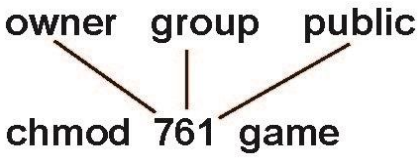


Access Lists and Groups in Unix

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- Ask manager to **create a group** (unique name), say **G**, and add some users to the group.
- For a file (say ***game***) or subdirectory, define an appropriate access.



- Attach a group to a file

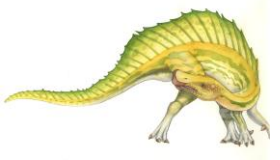
```
chgrp    G    game
```

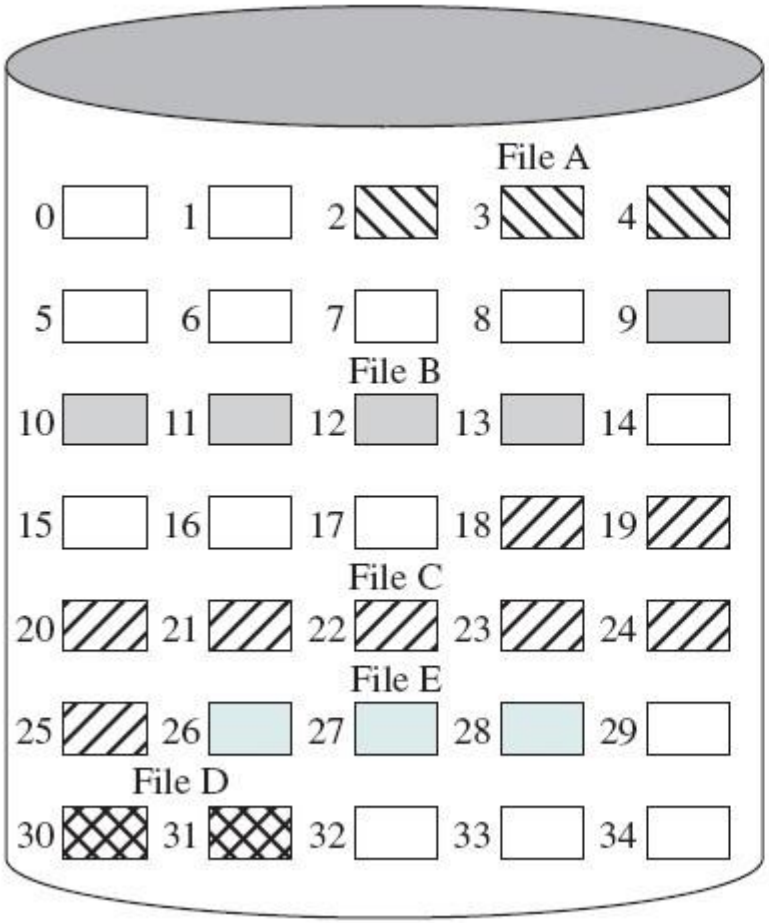
file allocation

- **File Allocation methods** Having looked at the issues of preallocation versus dynamic allocation and portion size, we are in a position to consider specific file allocation methods. Three methods are in common use: contiguous, chained, and indexed. Table 2 summarizes some of the characteristics of each method.
- With **contiguous allocation**, a single contiguous set of blocks is allocated to a file at the time of file creation (see Figure 6). Thus, 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.

Table 2 File Allocation Methods

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or Variable Size Portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion Size	Large	Small	Small	Medium
Allocation Frequency	Once	Low to high	High	Low
Time to Allocate	Medium	Long	Short	Medium
File Allocation Table Size	One entry	One entry	Large	Medium

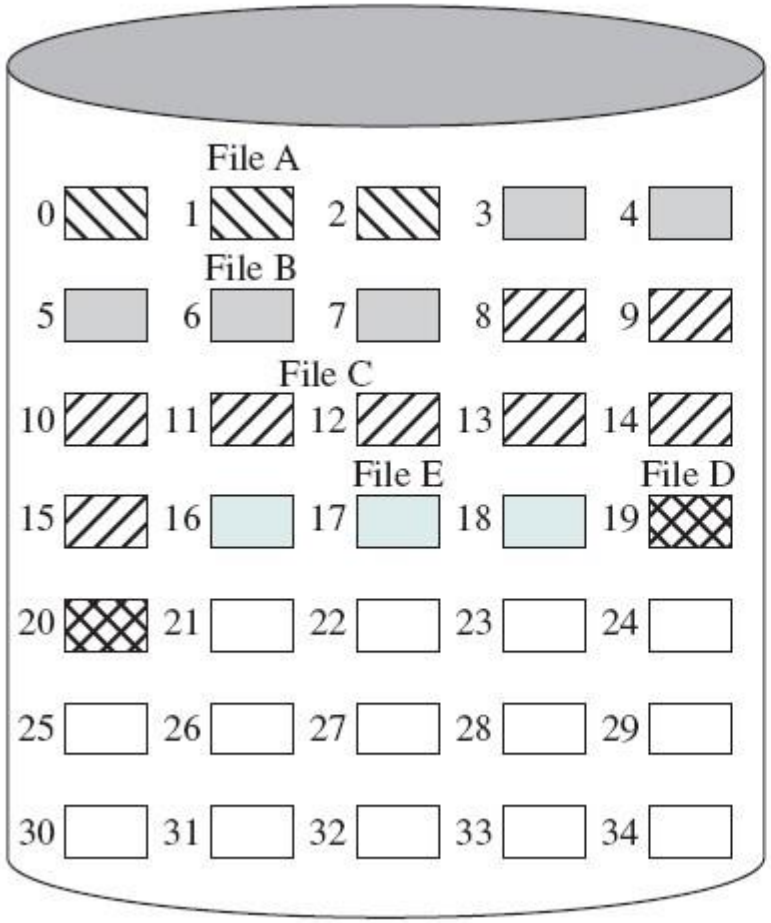




File allocation table

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 6 Contiguous File Allocation



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 7 Contiguous File Allocation (After Compaction)

allocation methods

- At the opposite extreme from contiguous allocation is
- II **chained allocation** (see Figure 8). Typically, allocation is on an individual block basis. **Each block contains a pointer to the next block in the chain**. Again, the file allocation table needs just a single entry for each file, showing the starting block and the length of the file.

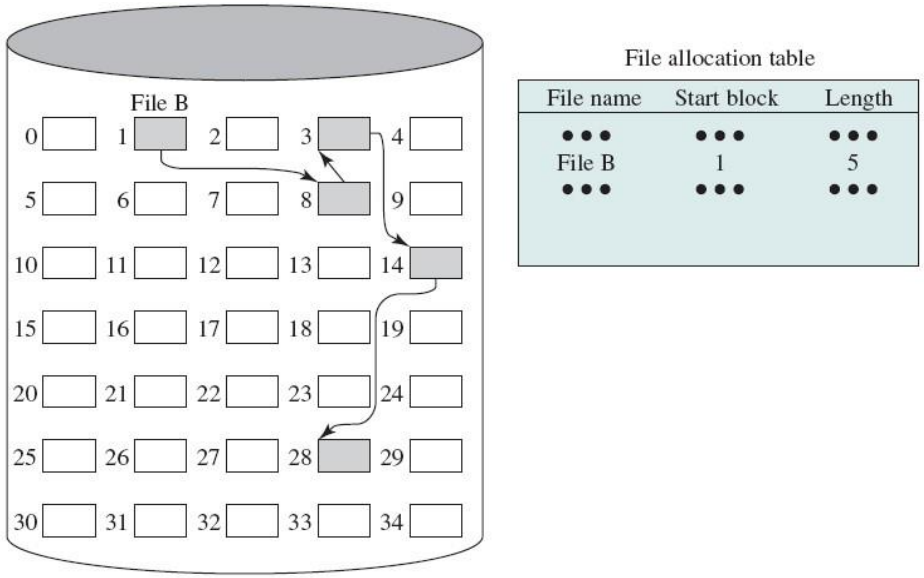
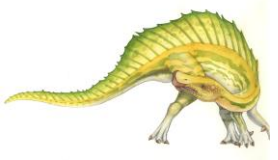


Figure 8 Chained Allocation



allocation methods

- One consequence of chaining, as described so far, is there is no accommodation of the principle of locality. Thus, 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. To overcome this problem, some systems periodically consolidate files (see Figure 9).

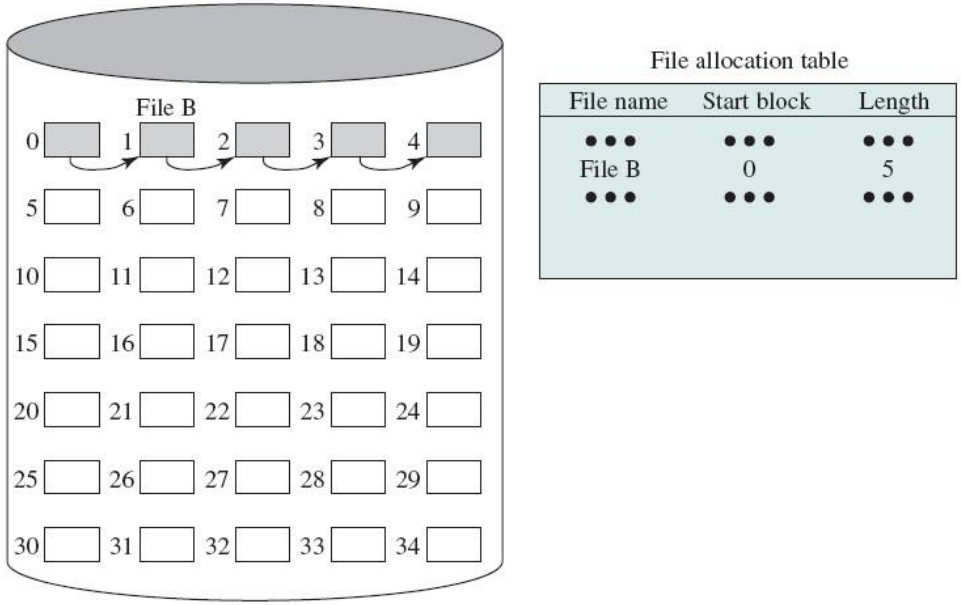


Figure 9 Chained Allocation (After Consolidation)

allocation methods

- **III Indexed allocation** 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. Allocation may be on the basis of either fixed-size blocks (see Figure 10) or variable-size portions (see Figure 11).
- 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 supports both sequential and direct access to the file and thus is the most popular form of file allocation.

allocation methods

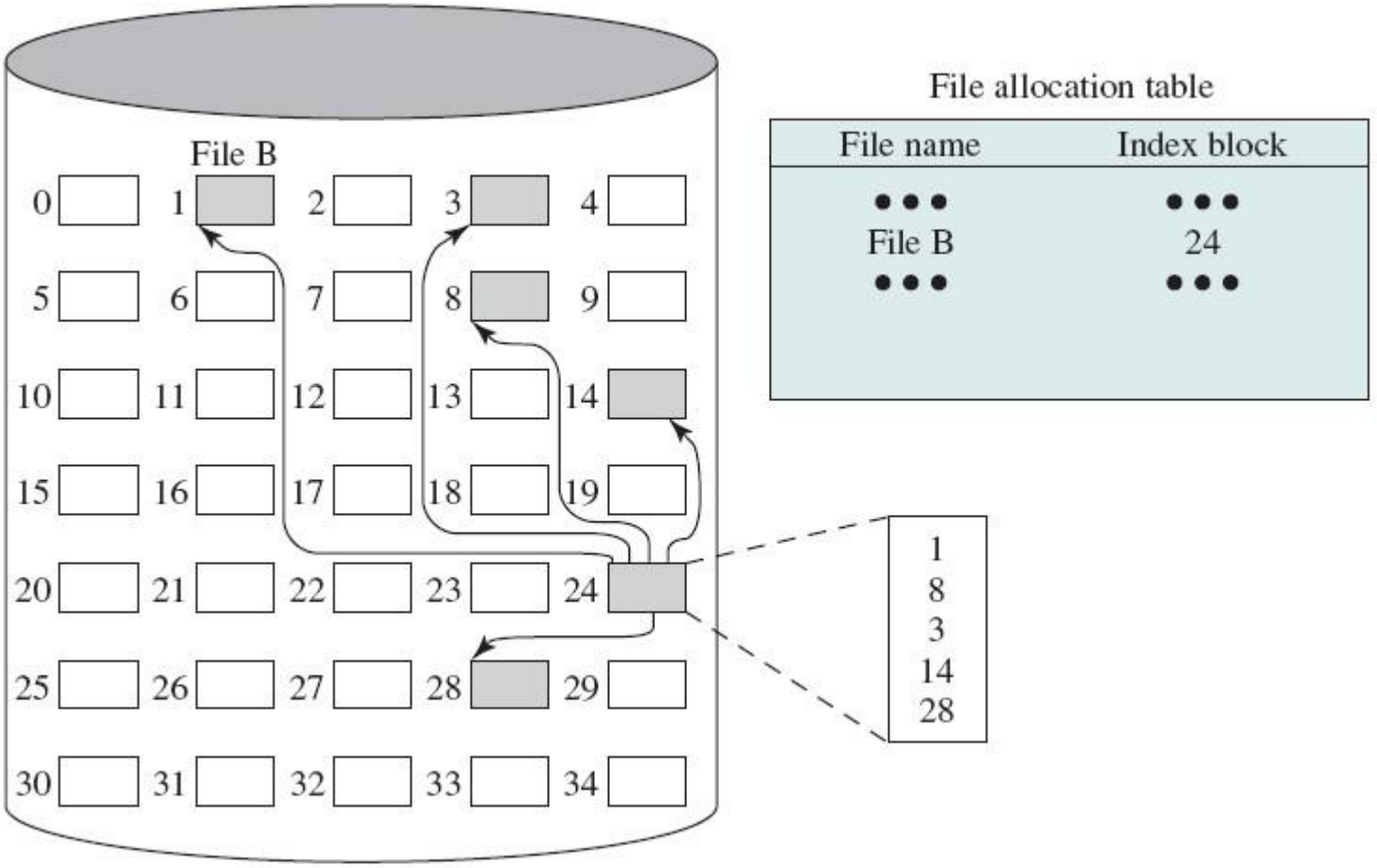


Figure 10 Indexed Allocation with Block Portions

allocation methods

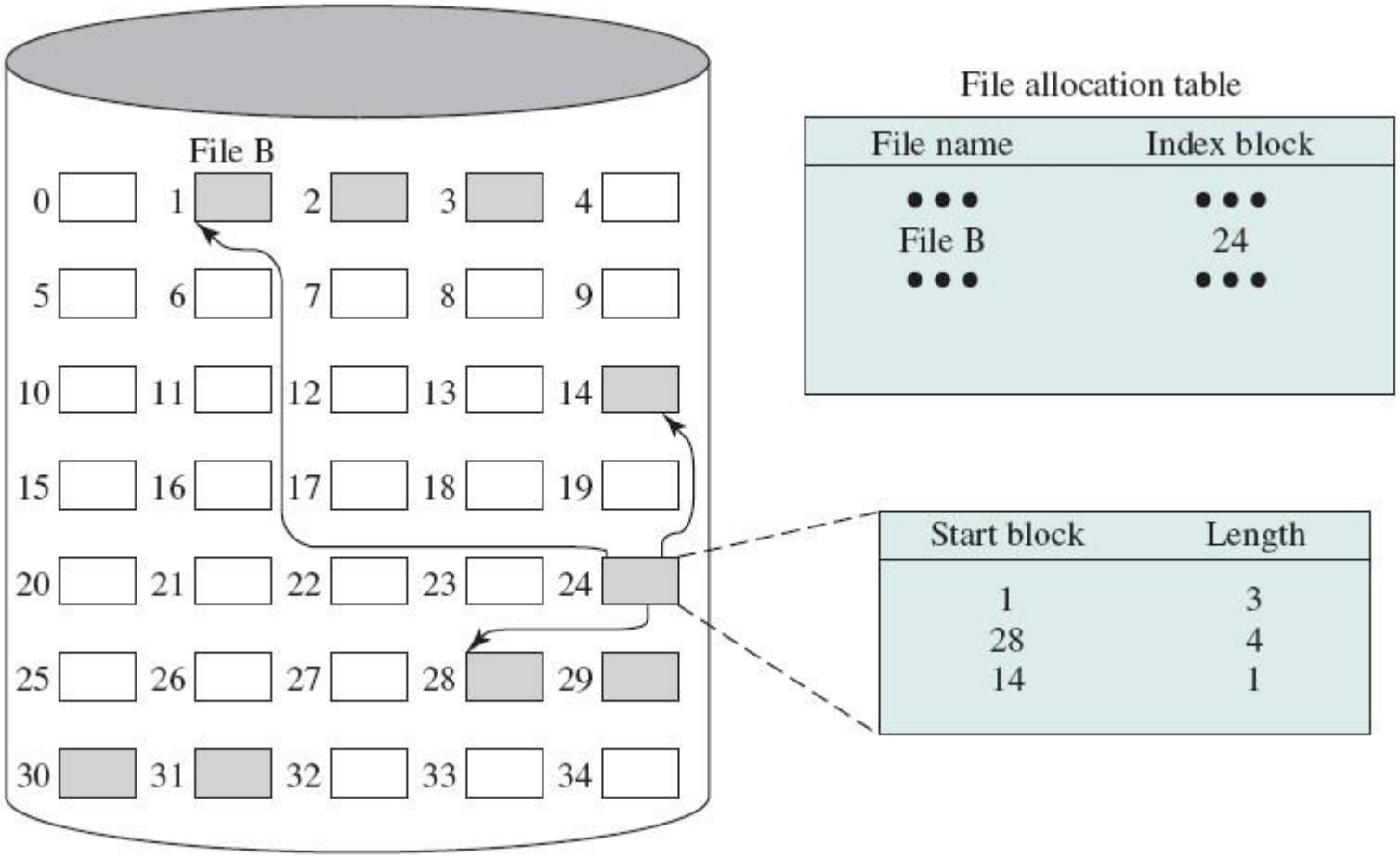


Figure 11 Indexed Allocation with Variable-Length Portions

widows NTFS example

- **WINDOWS FILE SYSTEM**
- The developers of Windows NT designed a new file system, the New Technology File System (NTFS), which is intended to meet high-end requirements for workstations and servers. Examples of high-end applications include the following:
  - Client/server applications such as file servers, compute servers, and database servers
  - Resource-intensive engineering and scientific applications
  - Network applications for large corporate systems This section provides an overview of NTFS.

windows NTFS file system

NTFS Volume and File Structure

- NTFS makes use of the following disk storage concepts:
  - **Sector:** The smallest physical storage unit on the disk. The data size in bytes is a power of 2 and is almost always 512 bytes.
  - **Cluster:** One or more contiguous (next to each other on the disk) sectors. The cluster size in sectors is a power of 2.
  - **Volume:** A logical partition on a disk, consisting of one or more clusters and used by a file system to allocate space. At any time, a volume consists of file system information, a collection of files, and any additional unallocated space remaining on the volume that can be allocated to files.
- A volume can be all or a portion of a single disk, or it can extend across multiple disks. If hardware or software RAID 5 is employed, a volume consists of stripes spanning multiple disks. The maximum volume size for NTFS is 2^{64} clusters.

End of Chapter 13