File Management

Chapter 12, Livro do William Stallings

Chapter 12, Livro do Silberchatz

Sistemas de Operação, 1º Semestre, 2004-2005

Objectives for a File Management System

- Meet the data management needs and requirements 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 or eliminate the potential for lost or destroyed data.
- Provide a standardized set of I/O interface routines.
- Provide I/O support for multiple users.

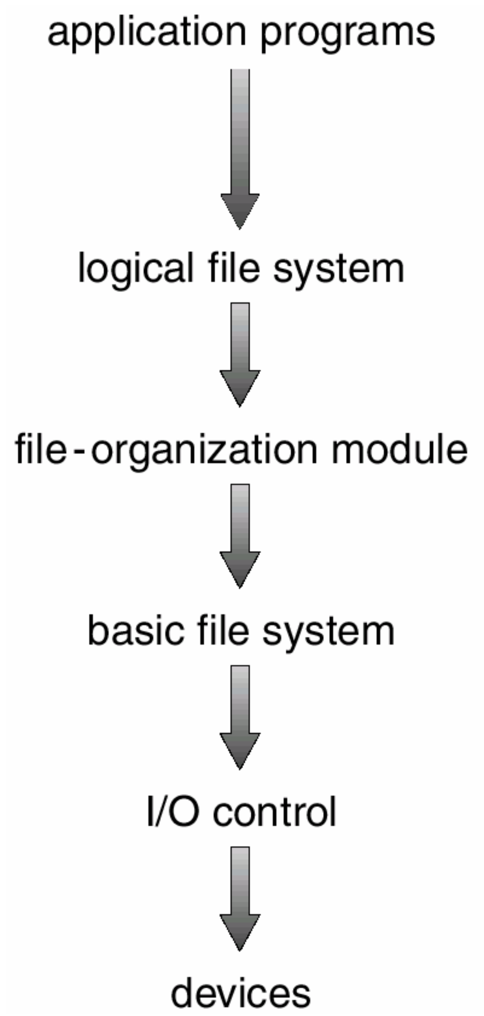
File Management Functions

- Identify and locate a selected file.
- Use a directory to describe the location of all files plus their attributes.
- On a shared system describe user access control.
- Blocking for access to files.
- Allocate files to free blocks.
- Manage free storage for available blocks.

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- **File control block** – storage structure consisting of information about a file.

Layered File System



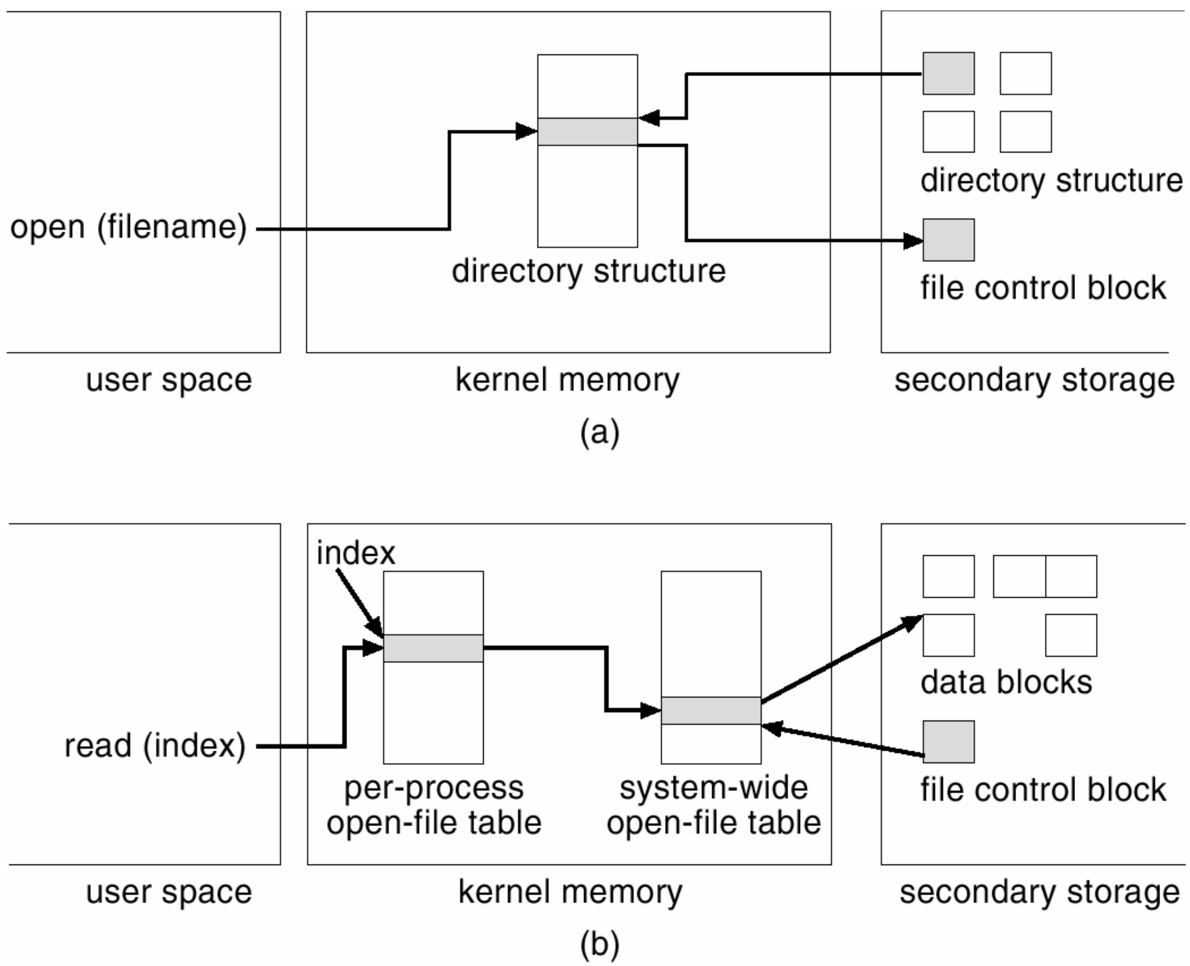
A Typical File Control Block

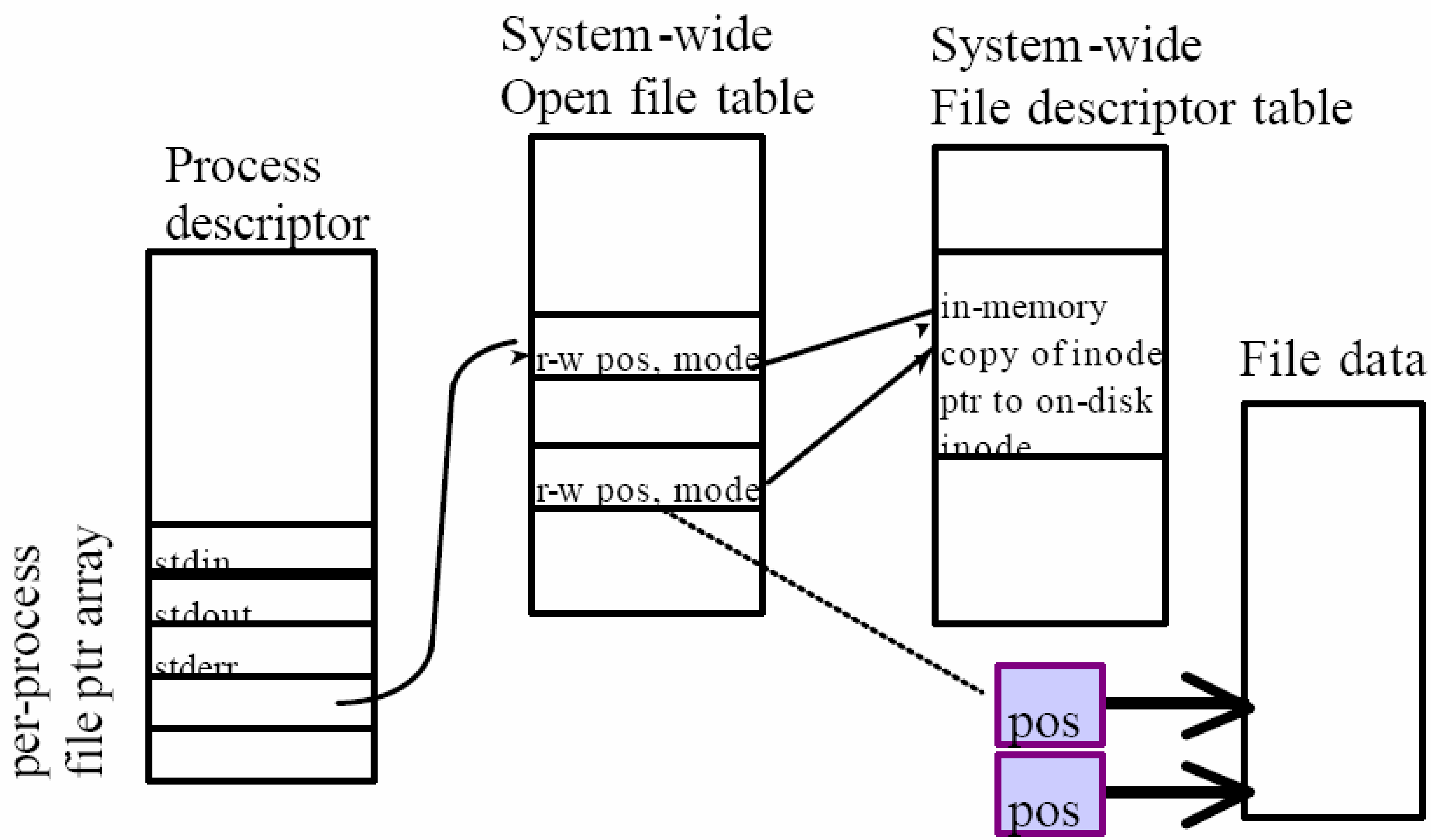
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

In-Memory File System Structures

- The following figures illustrate the necessary file system structures provided by the operating systems.
- Figure (a) refers to opening a file.
- Figure (b) refers to reading a file.

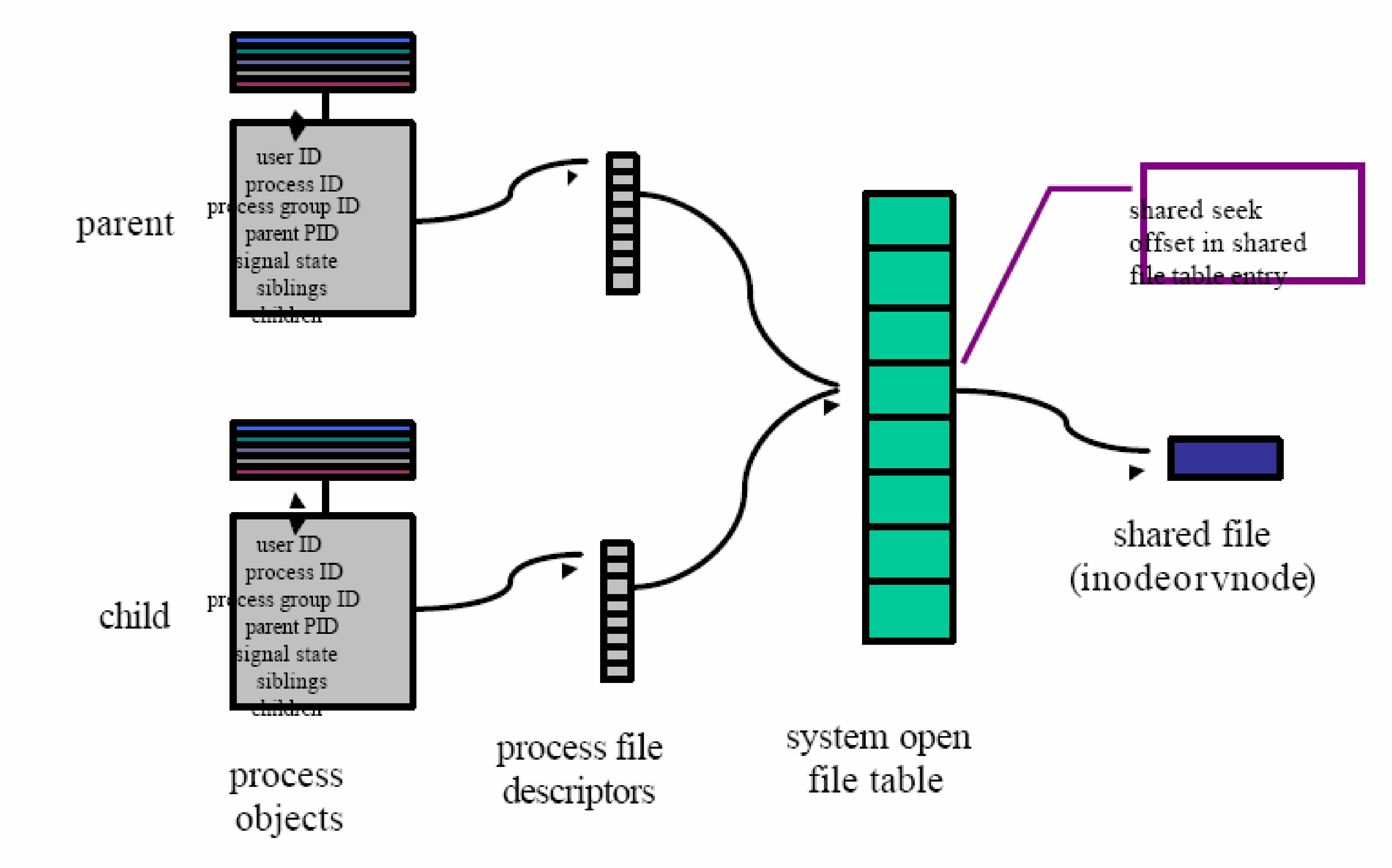
In-Memory File System Structures





System-wide
Open file table
System-wide
File descriptor table
Process
descriptor
r-w pos, mode
r-w pos, mode
in-memory
copy of inode
ptr to on-disk
inode
File data
per-process
file ptr array
stdin
stdout
stderr
pos
pos

Sharing File Descriptors



File Directories

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

	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

Hierarchical, or Tree-Structured Directory

- Master directory with user directories.
- Each user directory may have subdirectories and files as entries.

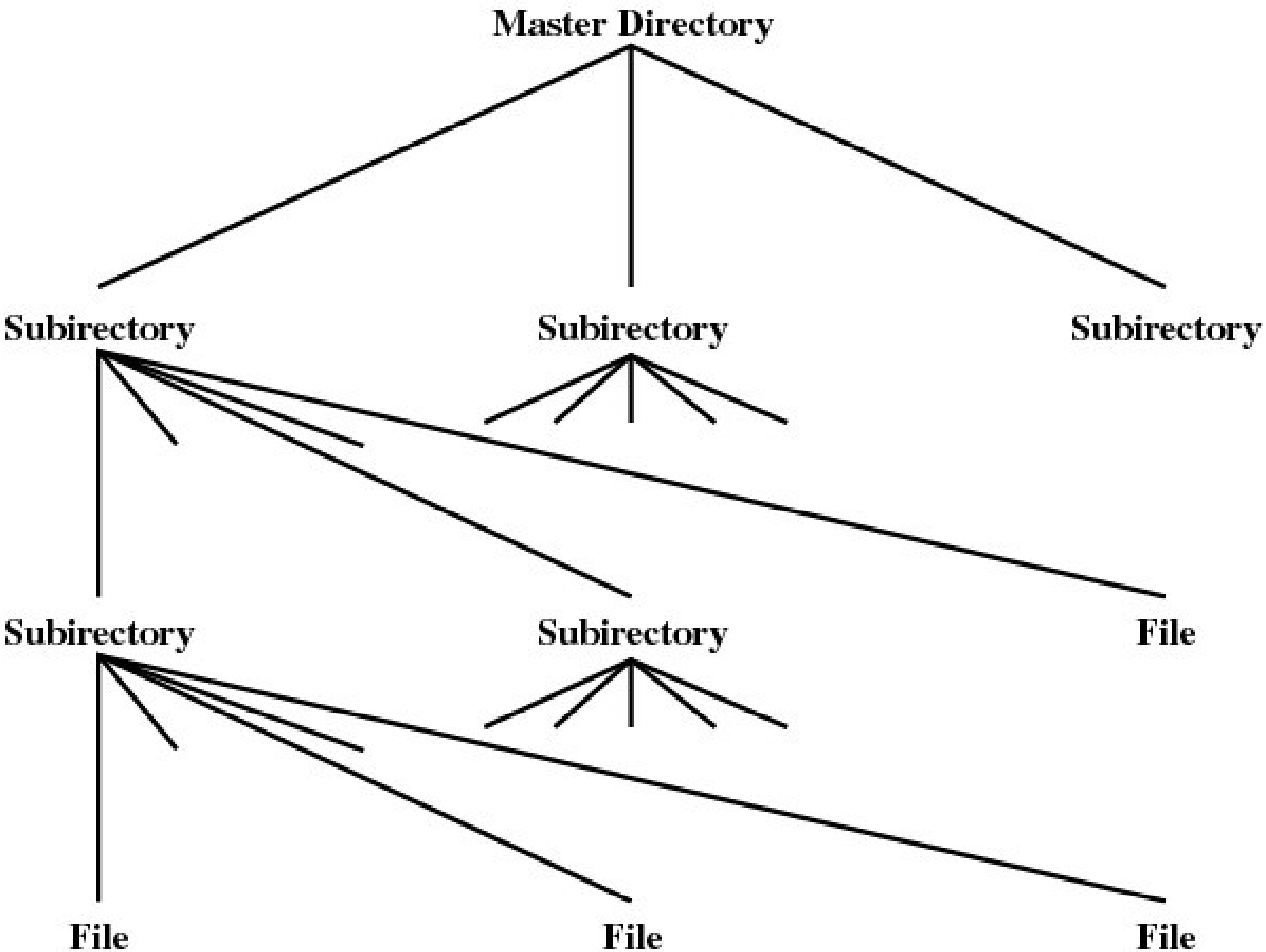


Figure 12.4 Tree-Structured Directory

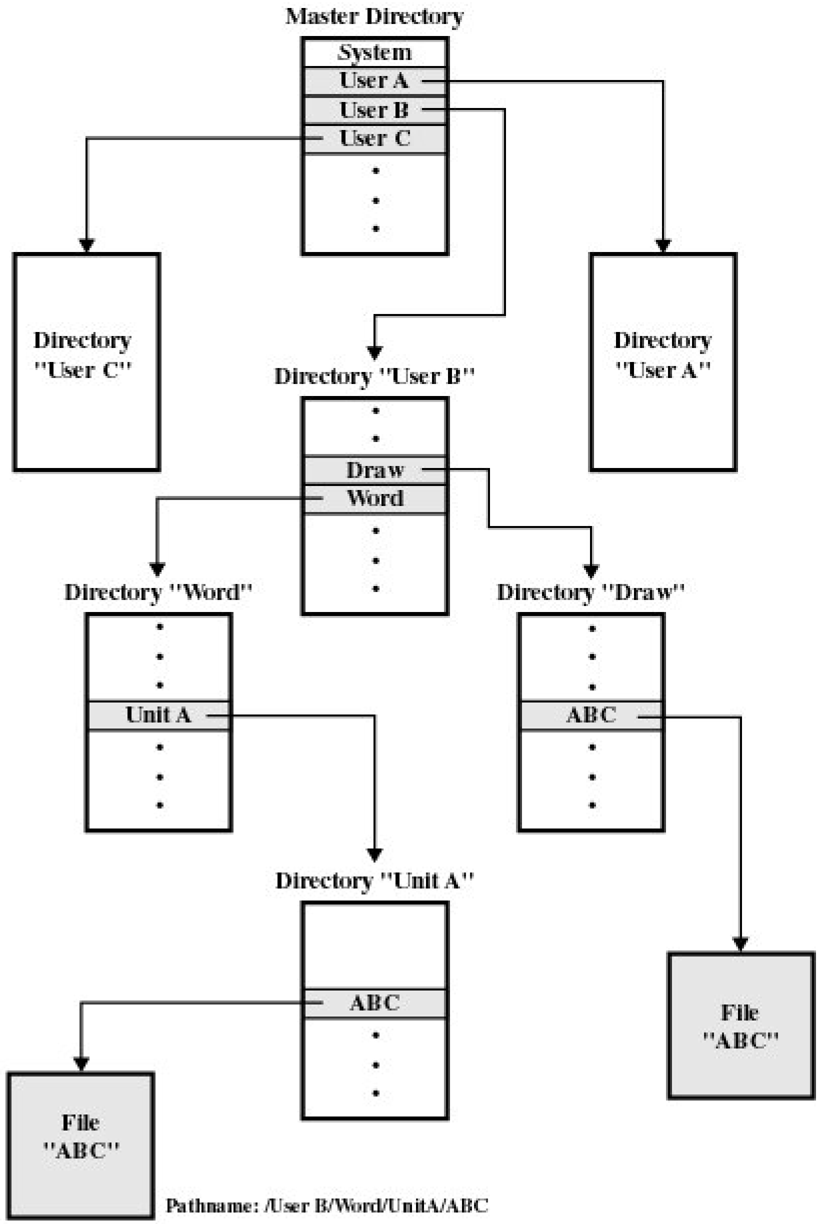
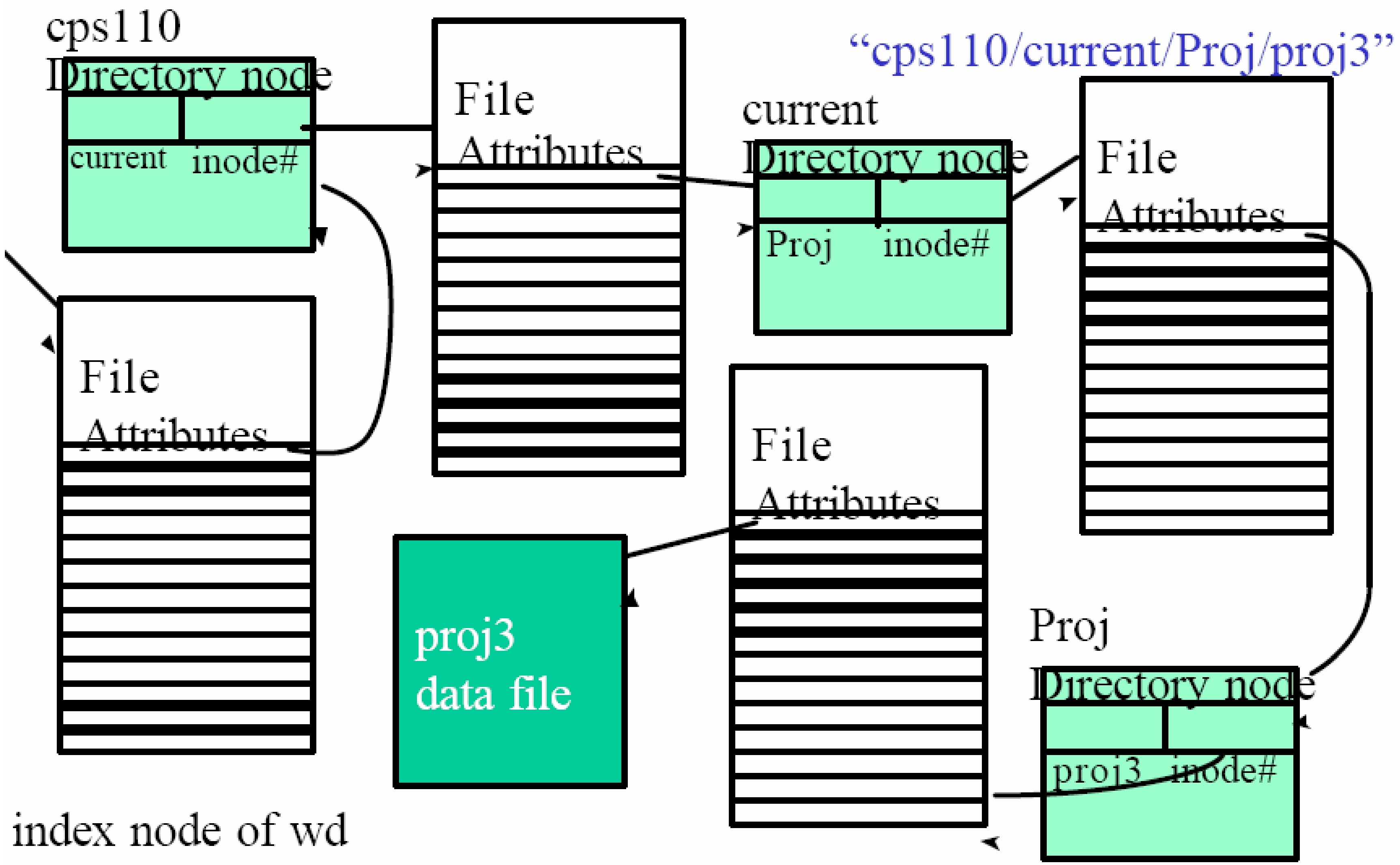


Figure 12.5 Example of Tree-Structured Directory

Hierarchical, or Tree-Structured Directory

- Files can be located by following a path from the root, or master, directory down various branches
  - This is the pathname for the file
- Can have several files with the same file name as long as they have unique path names



"cps110/current/Proj/proj3"
cps110
Directory node
current
inode#
File
Attributes
current
Directory node
Proj
inode#
File
Attributes
File
Attributes
File
Attributes
proj3
data file
Proj
Directory node
proj3
inode#
index node of wd

Secondary Storage Management

Secondary Storage Management

- Space must be allocated to files
- Must keep track of the space available for allocation

- **Preallocation versus Dynamic Allocation.**

Preallocation

- Need the maximum size for the file at the time of creation
- Difficult to reliably estimate the maximum potential size of the file
- Tend to overestimated file size so as not to run out of space

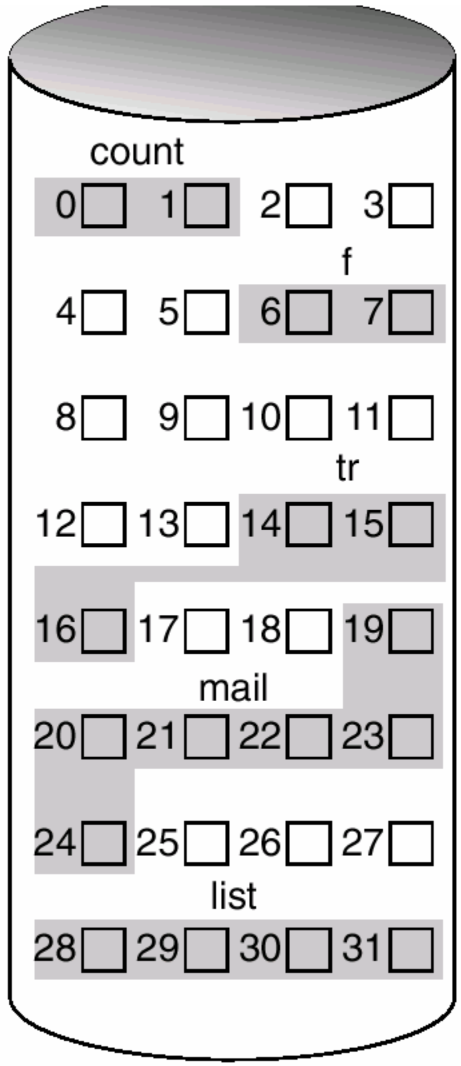
File Allocation Methods

- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

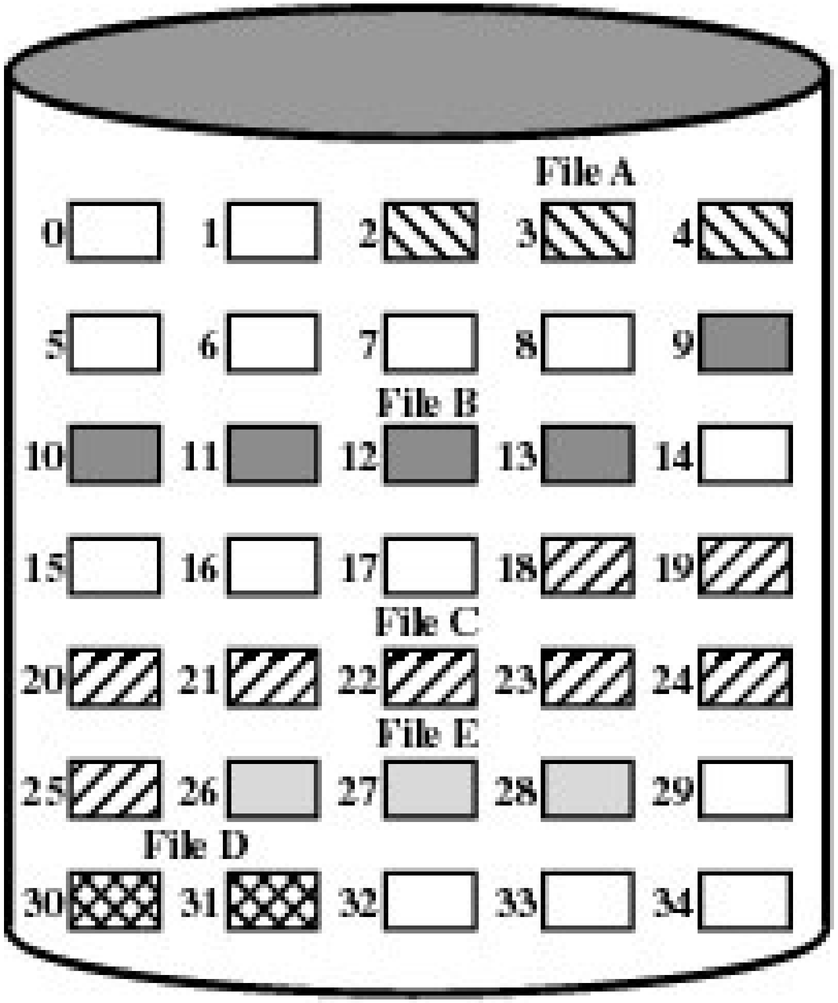
- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.
- External fragmentation will occur.

Contiguous Allocation



directory

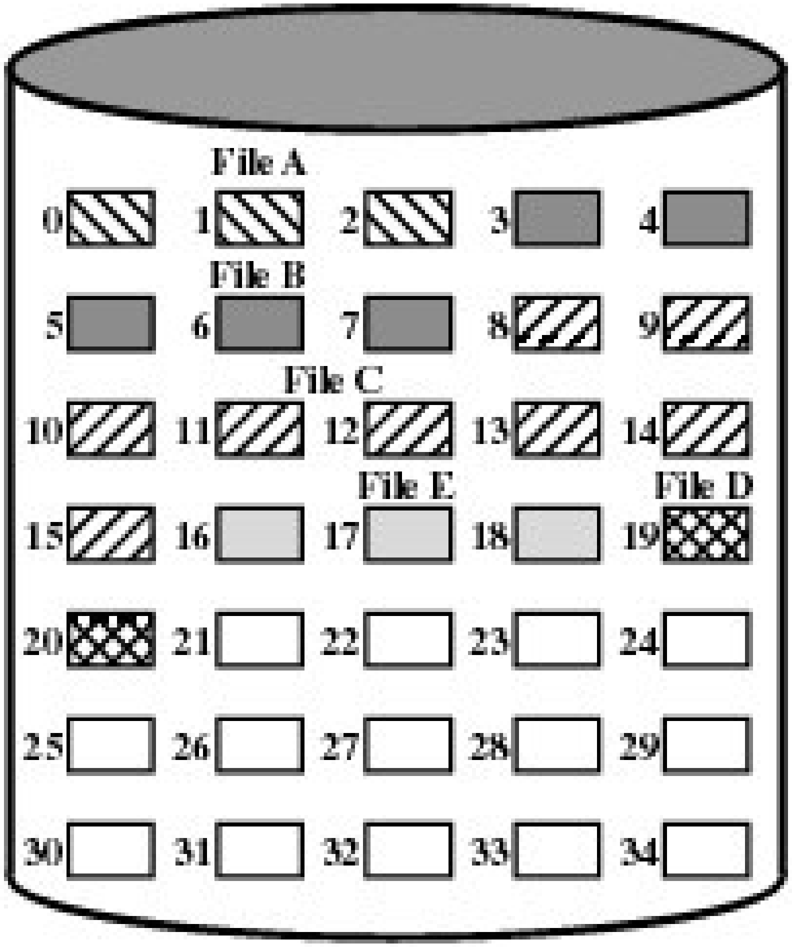
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



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



File Allocation Table

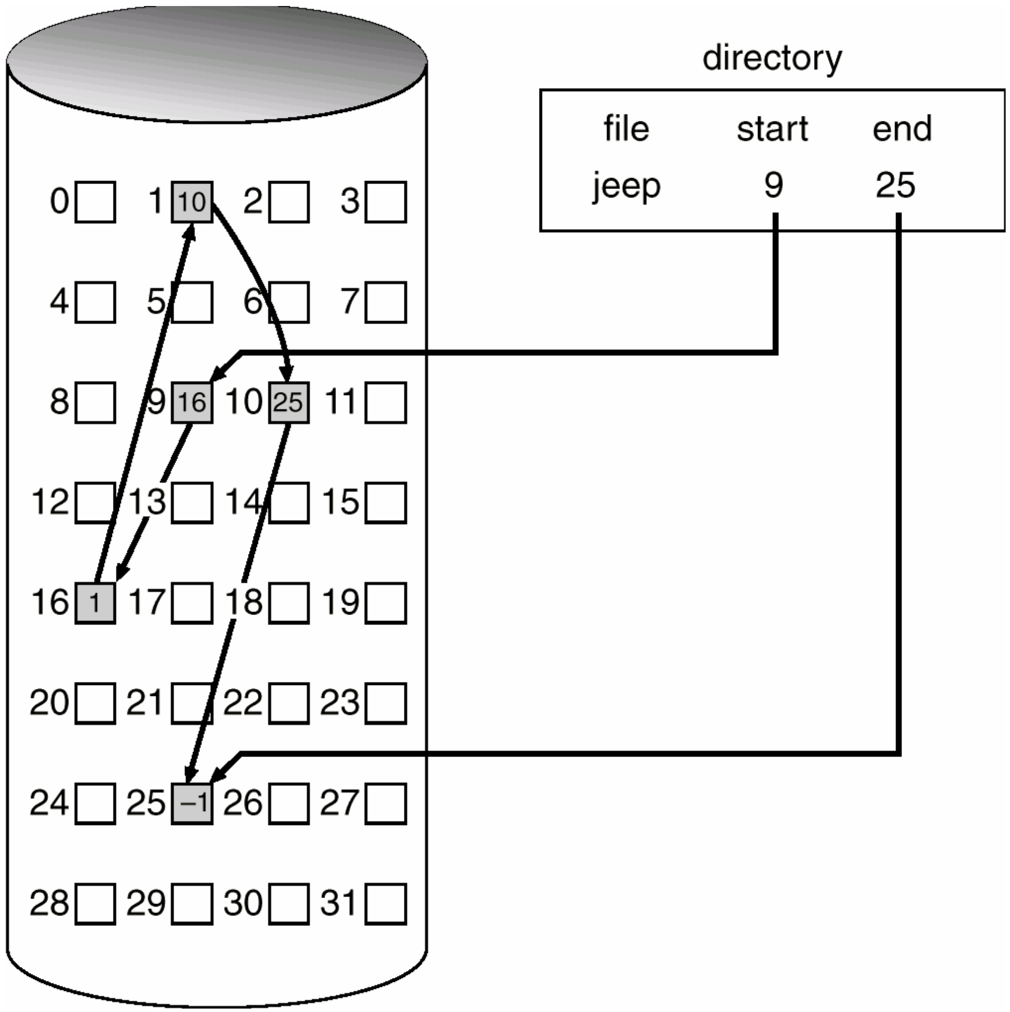
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

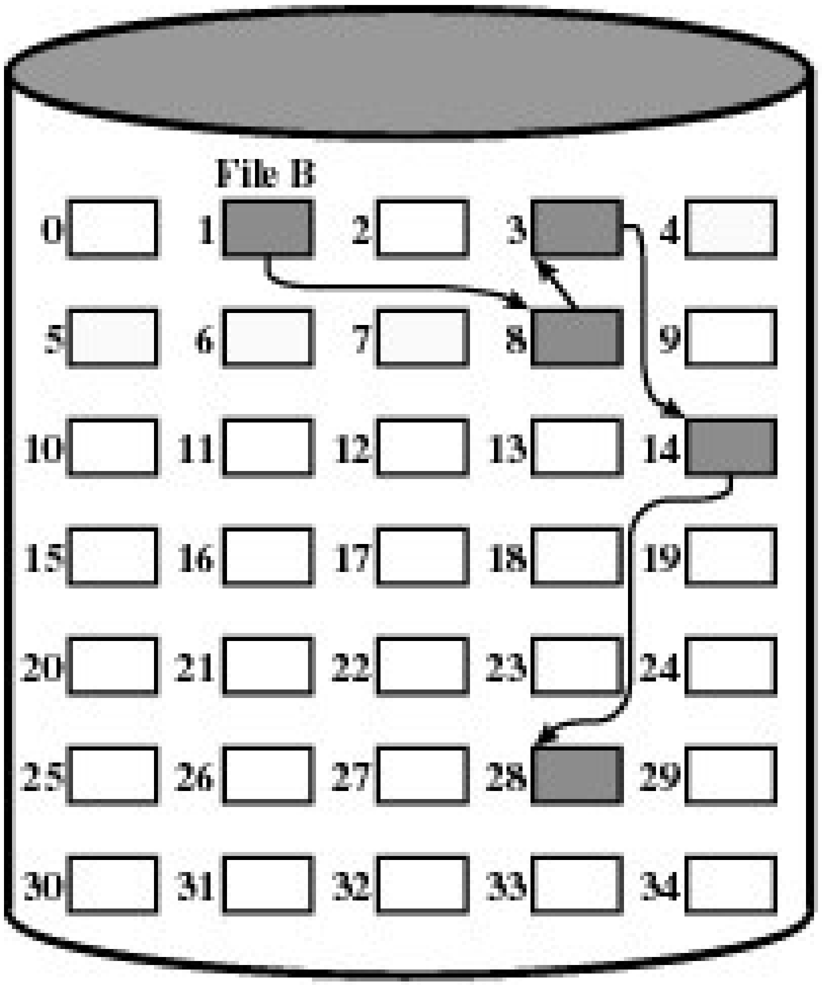
Figure 12.8 Contiguous File Allocation (After Compaction

Linked/Chained Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple – need only starting address
- Free-space management system – no waste of space
- No external fragmentation
- No accommodation of the principle of locality
- No random access
- Ex: File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked/Chained Allocation

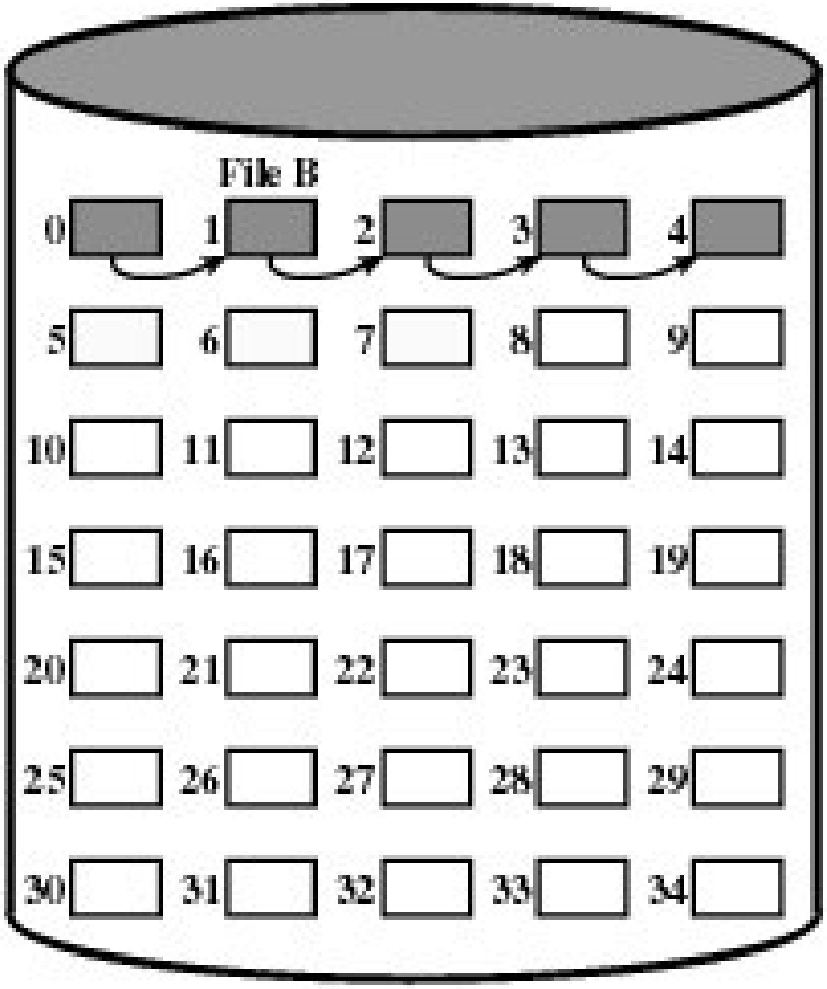




File Allocation Table

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

Figure 12.9 Chained Allocation

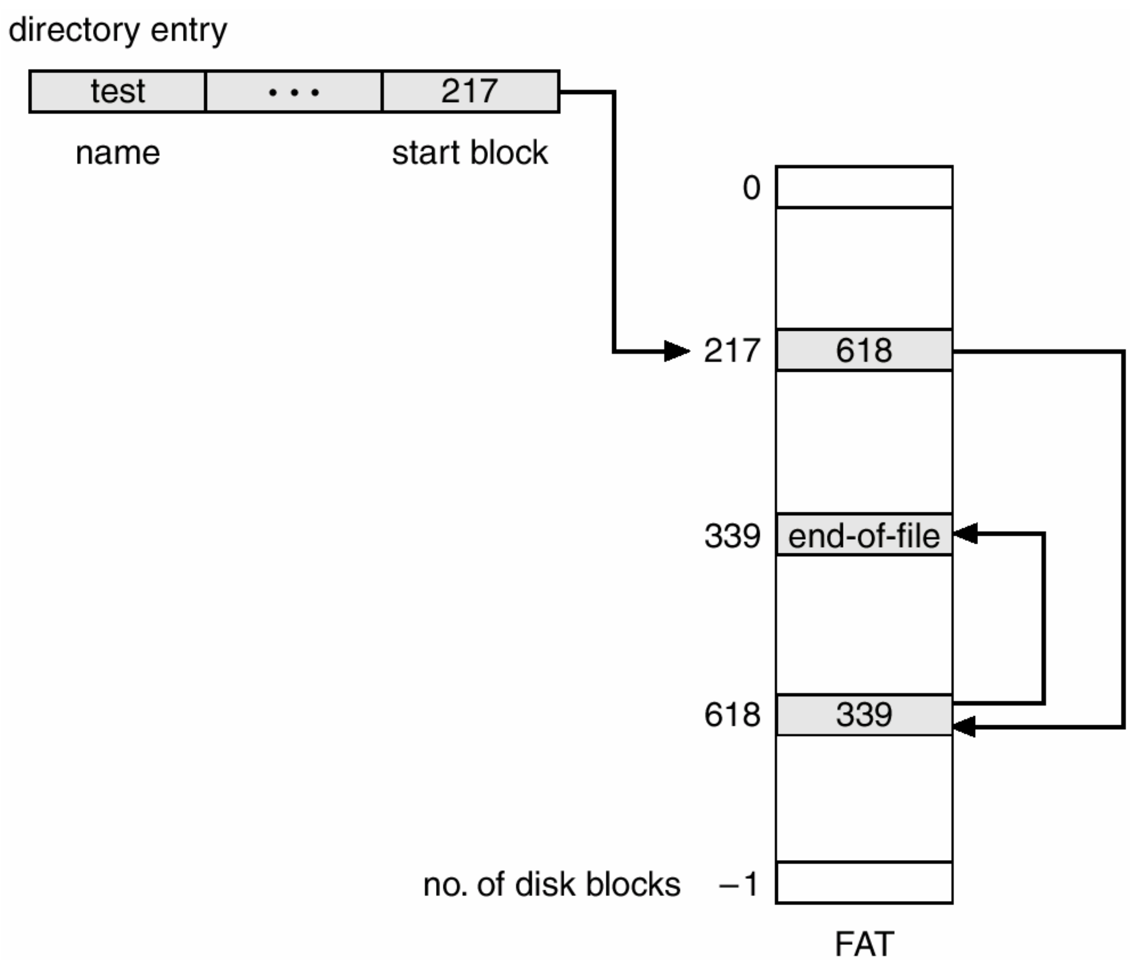


File Allocation Table

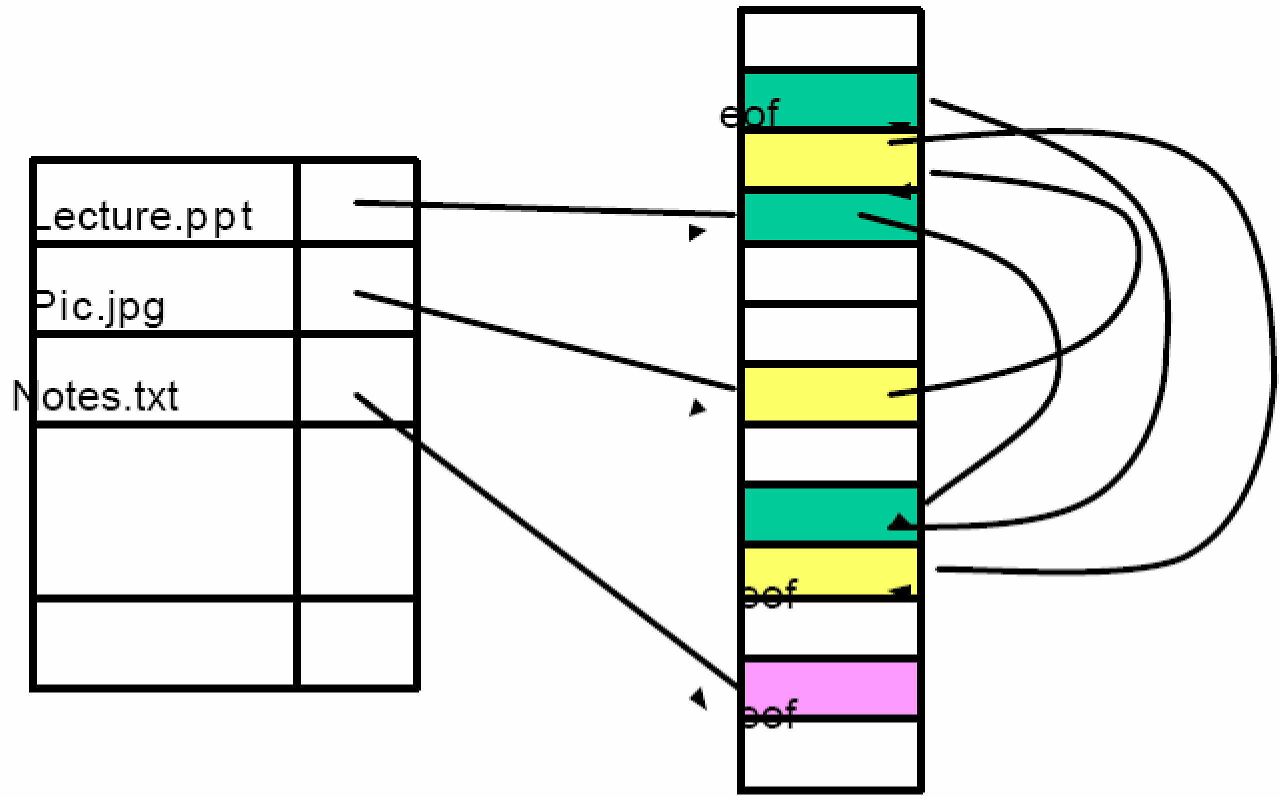
File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (after consolidation)

File-Allocation Table

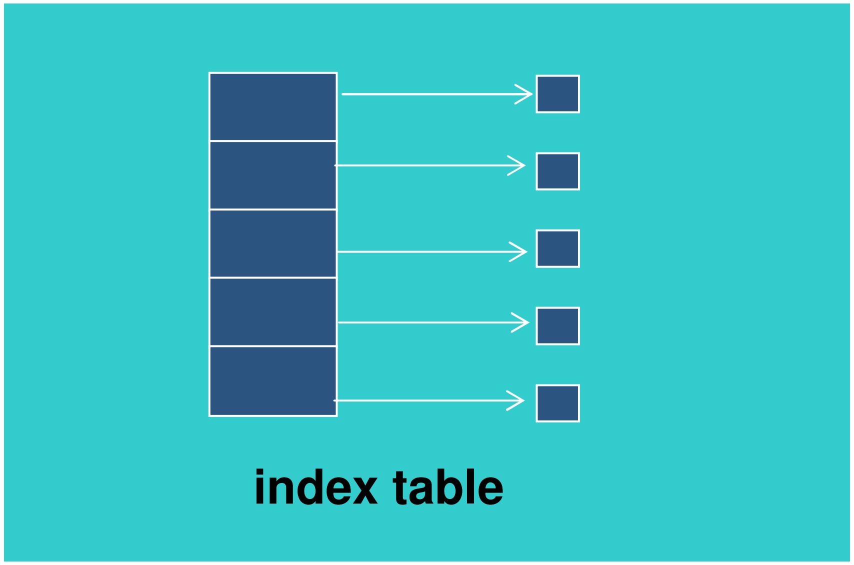


FAT: File Allocation Table

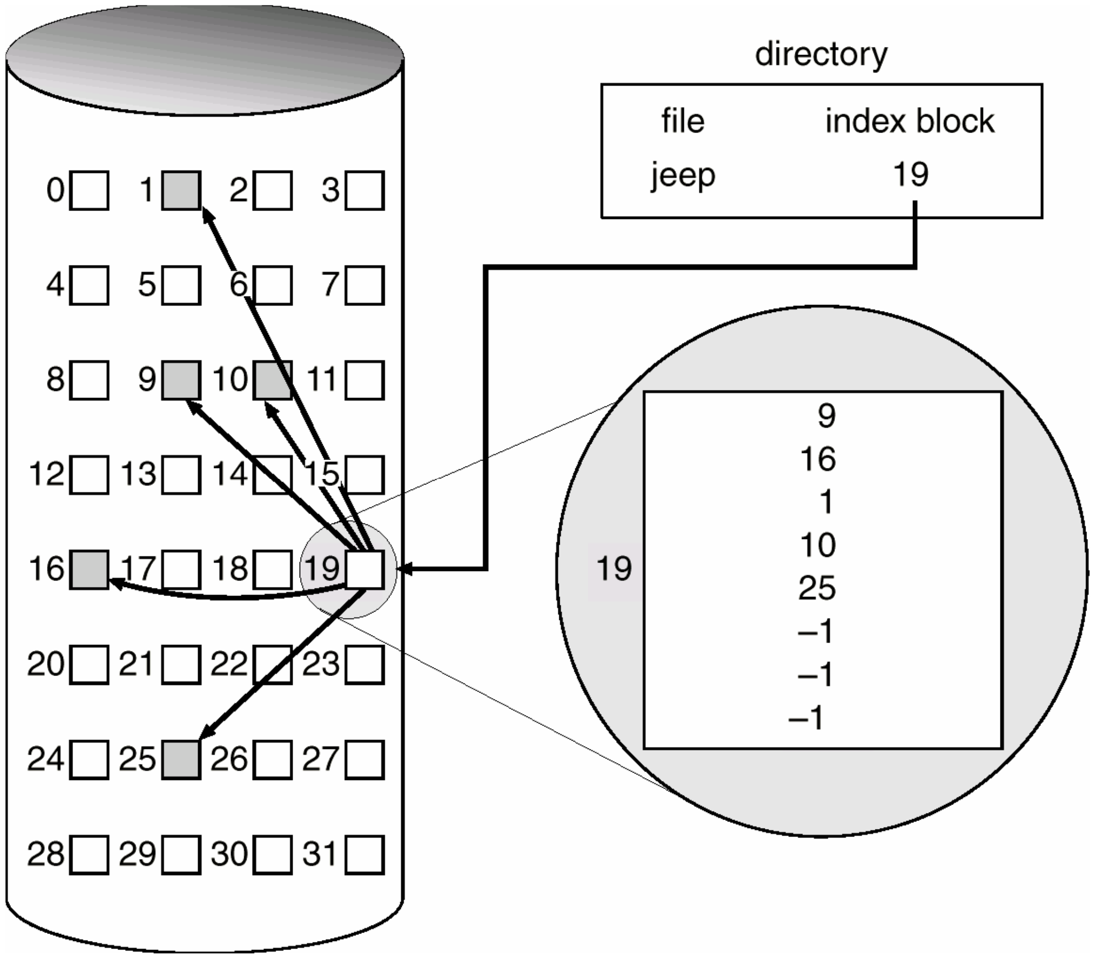


Indexed Allocation

- Brings all pointers together into the *index block.*
- Logical view.



Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

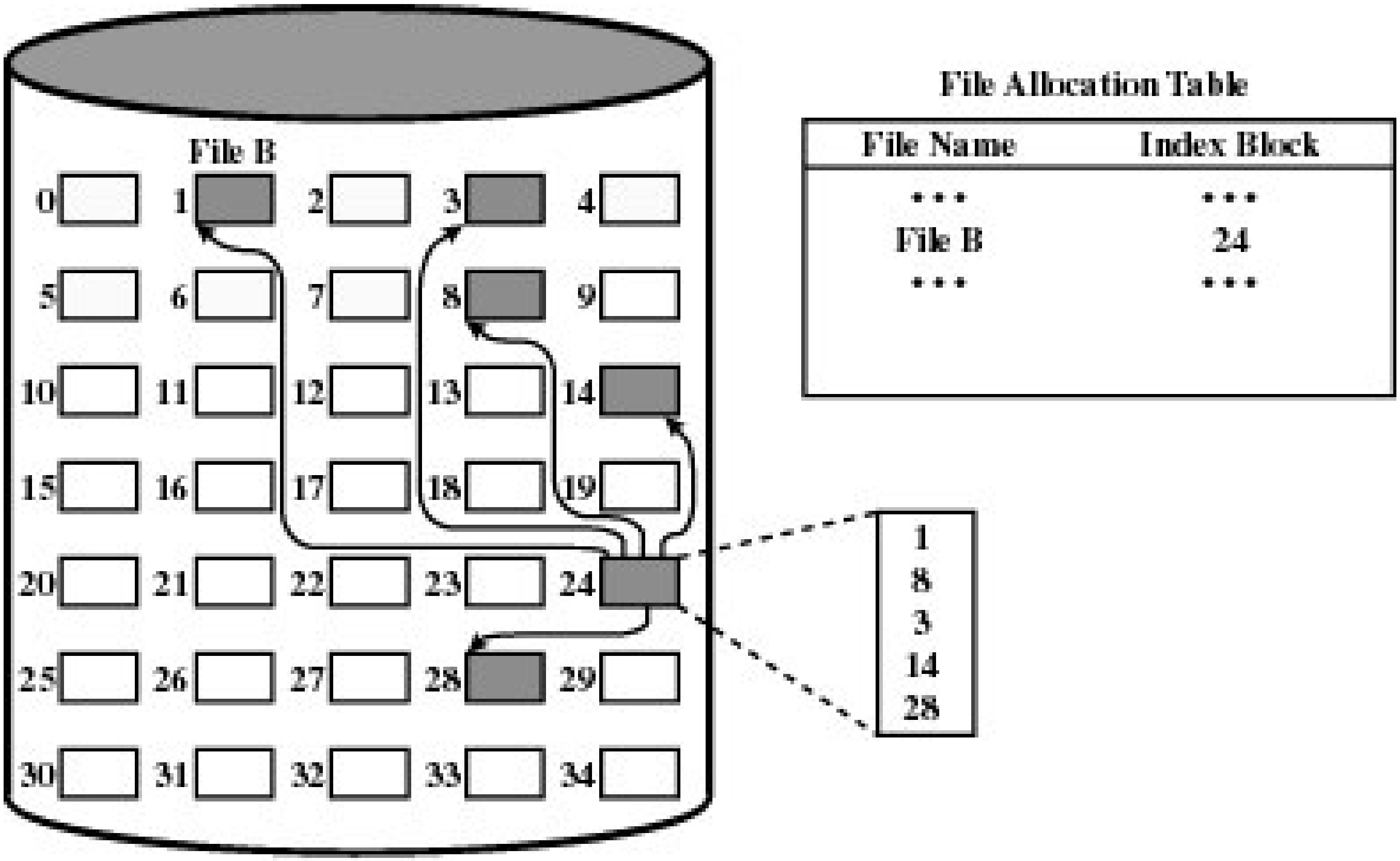


Figure 12.11 Indexed Allocation with Block Portions

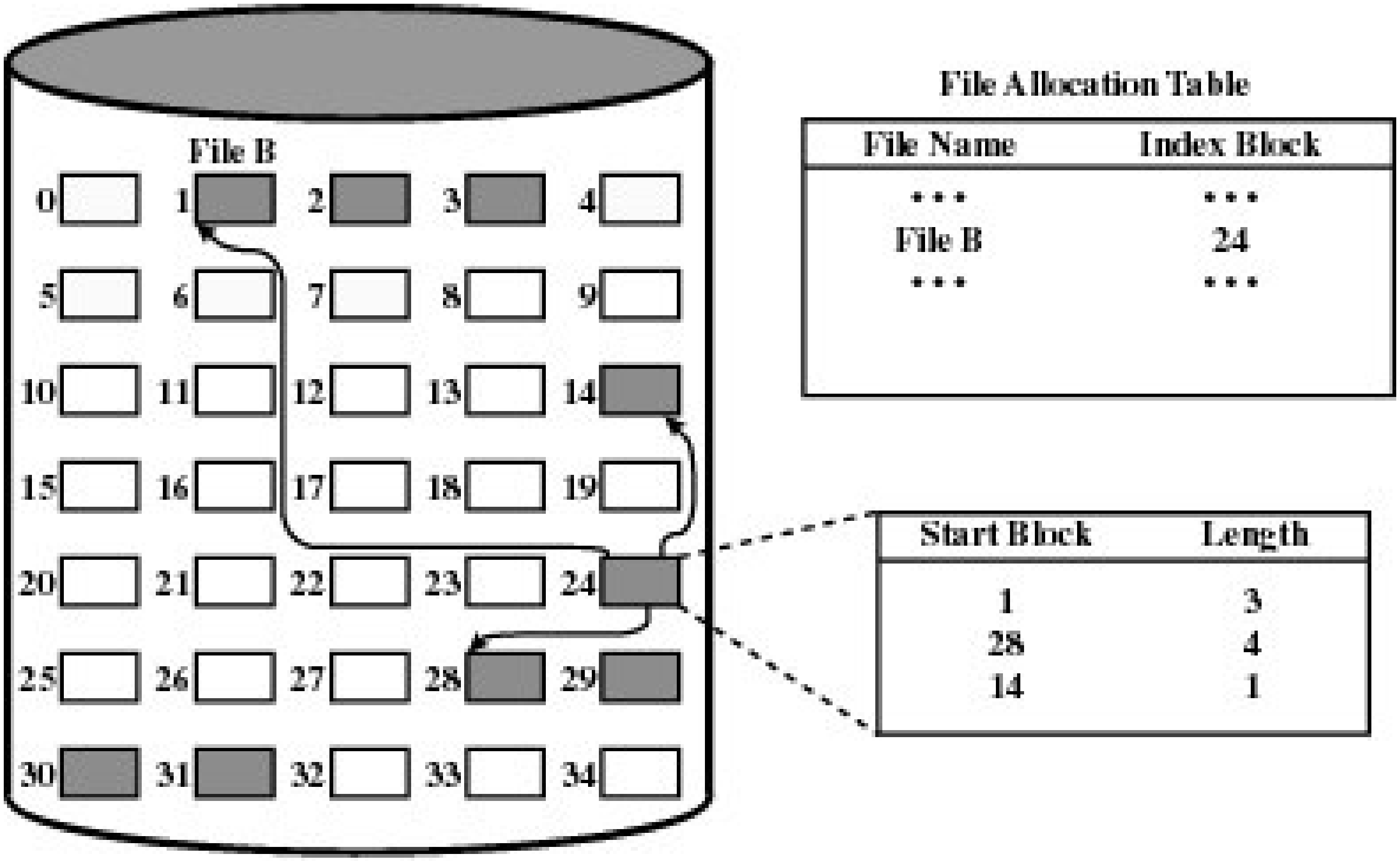
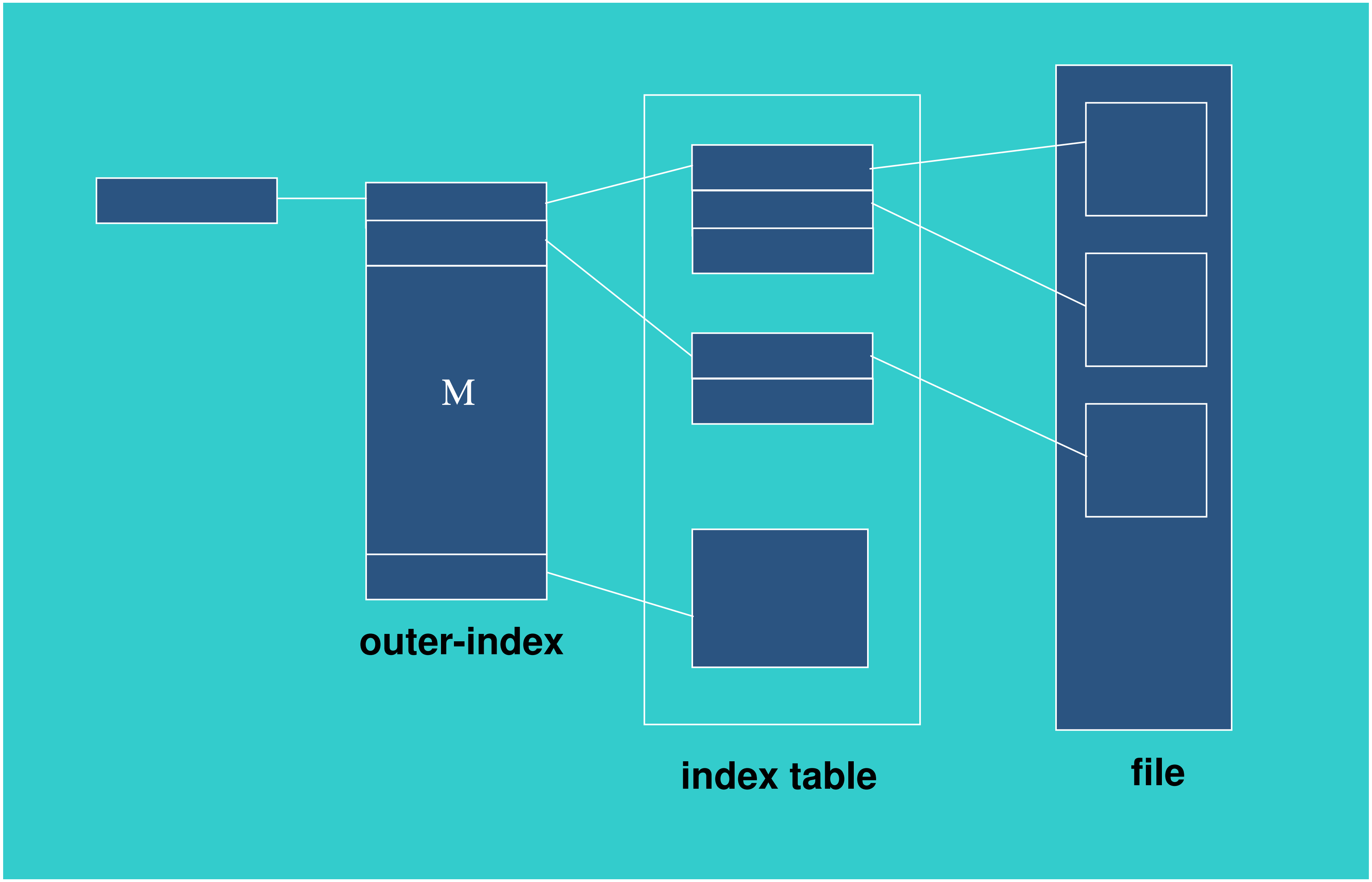


Figure 12.12 Indexed Allocation with Variable-Length Portions

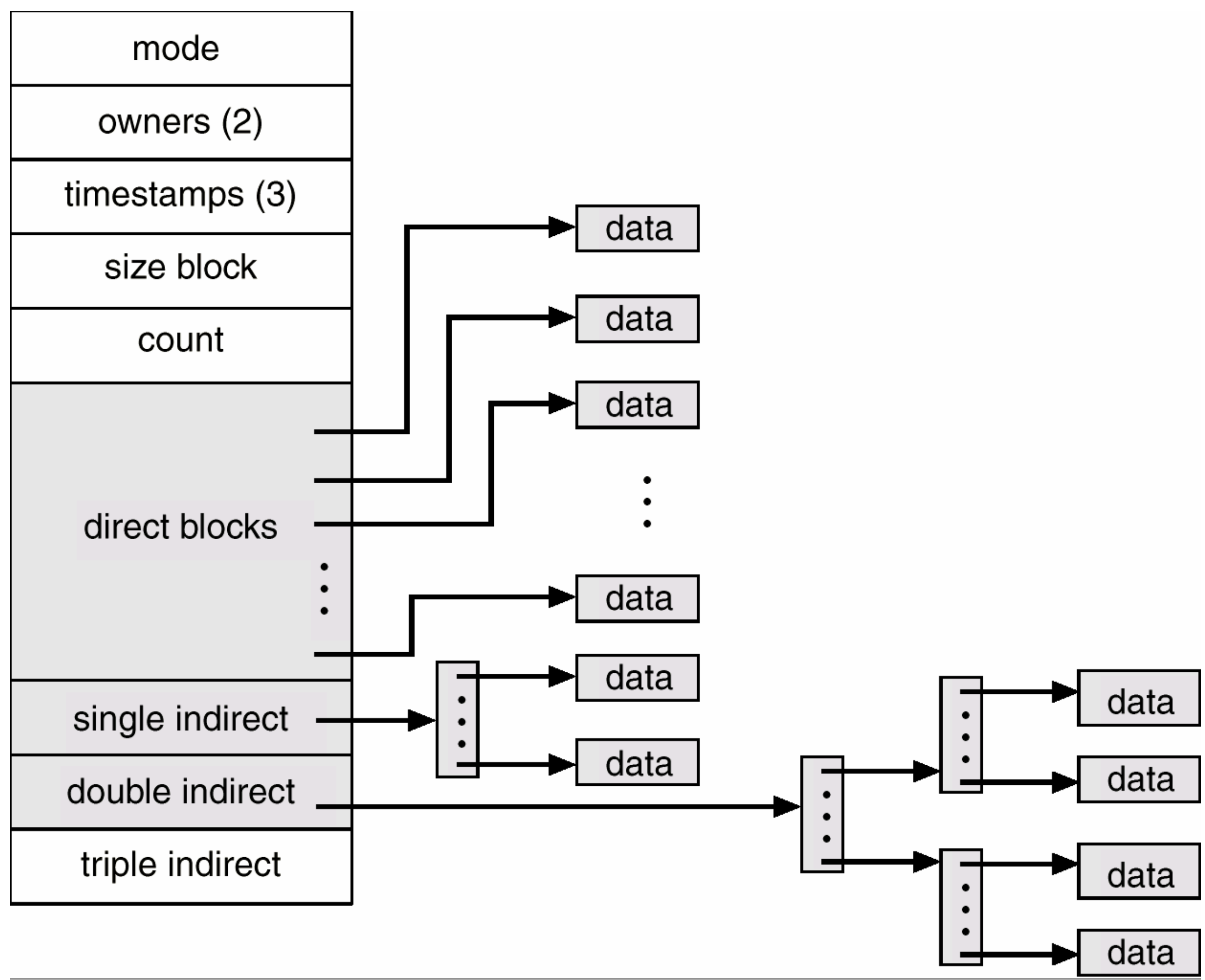
Indexed Allocation – Mapping



File Allocation Methods

	Contiguous	Chained	Indexed	
Pre-Allocation?	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

Combined Scheme: UNIX (4K bytes per block)



Unix Inodes

File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

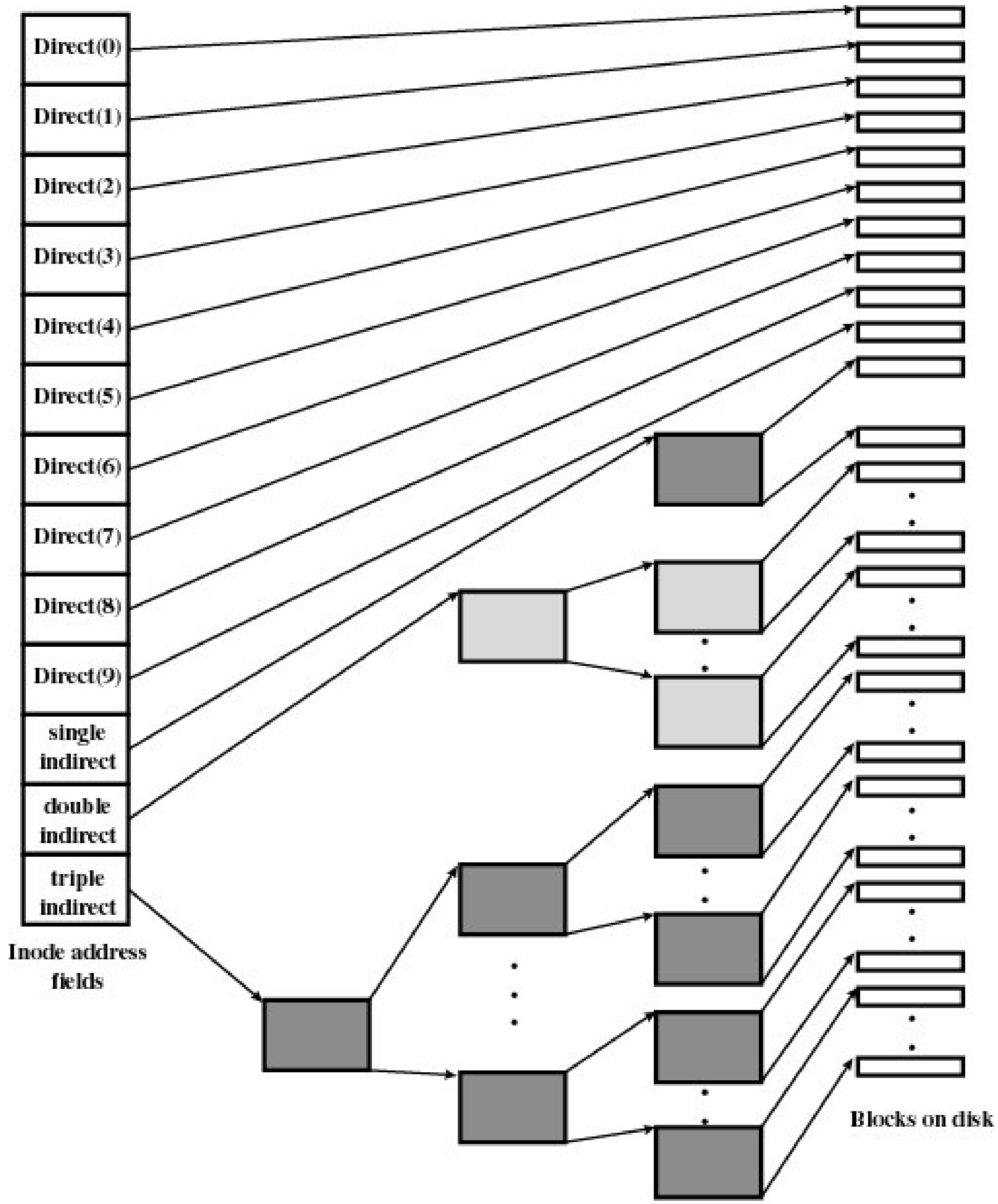
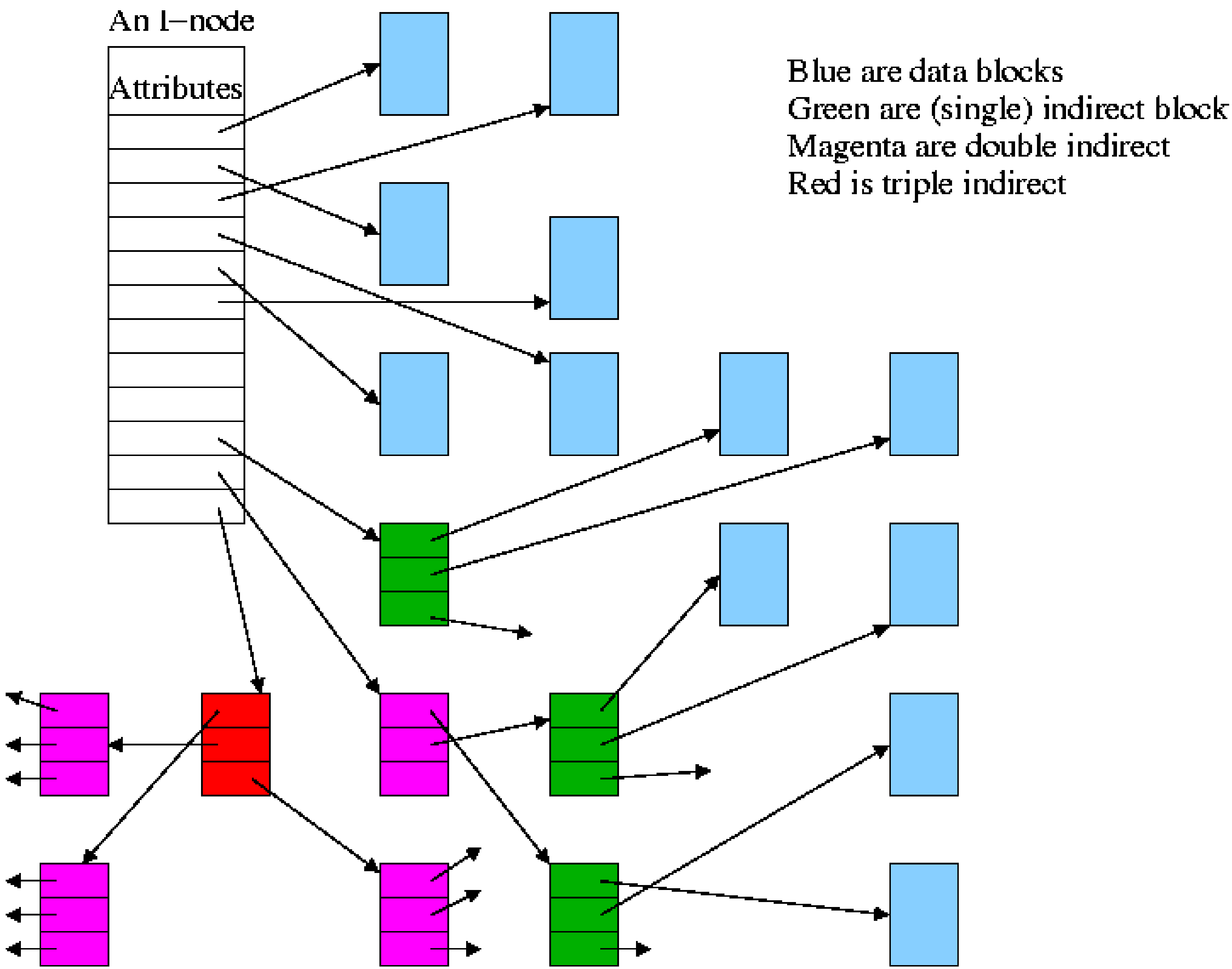


Figure 12.13 UNIX Block Addressing Scheme


An I−node
Attributes
Blue are data blocks
Green are (single) indirect block
Magenta are double indirect
Red is triple indirect

Capacity of a UNIX File

Level	Number of Blocks	Number of Bytes
Direct	10	10K
Single Indirect	256	256K
Double Indirect	$256 \times 256 = 65K$	65M
Triple Indirect	$256 \times 65K = 16M$	16G

Blocos de 1 k

Ponteiro: 32 bits

Capacity of a UNIX File

Level	Number of Blocks	Number of Bytes
Direct	10	40k
Single Indirect	1024	4M
Double Indirect	1024x1024=1M	4G
Triple Indirect	$256 \times 65K = 16M$	4T

Blocos de 4 k

Ponteiro: 32 bits

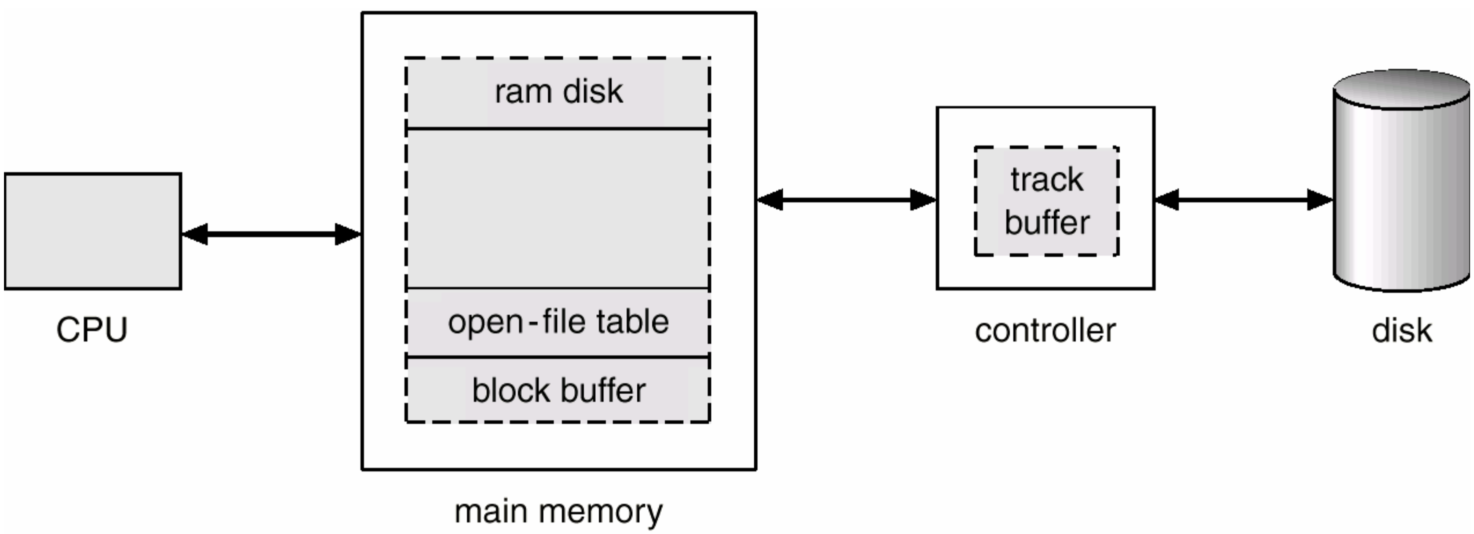
Go to QUIZ#10

Efficiency and Performance

- **Efficiency** dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry

- **Performance**
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

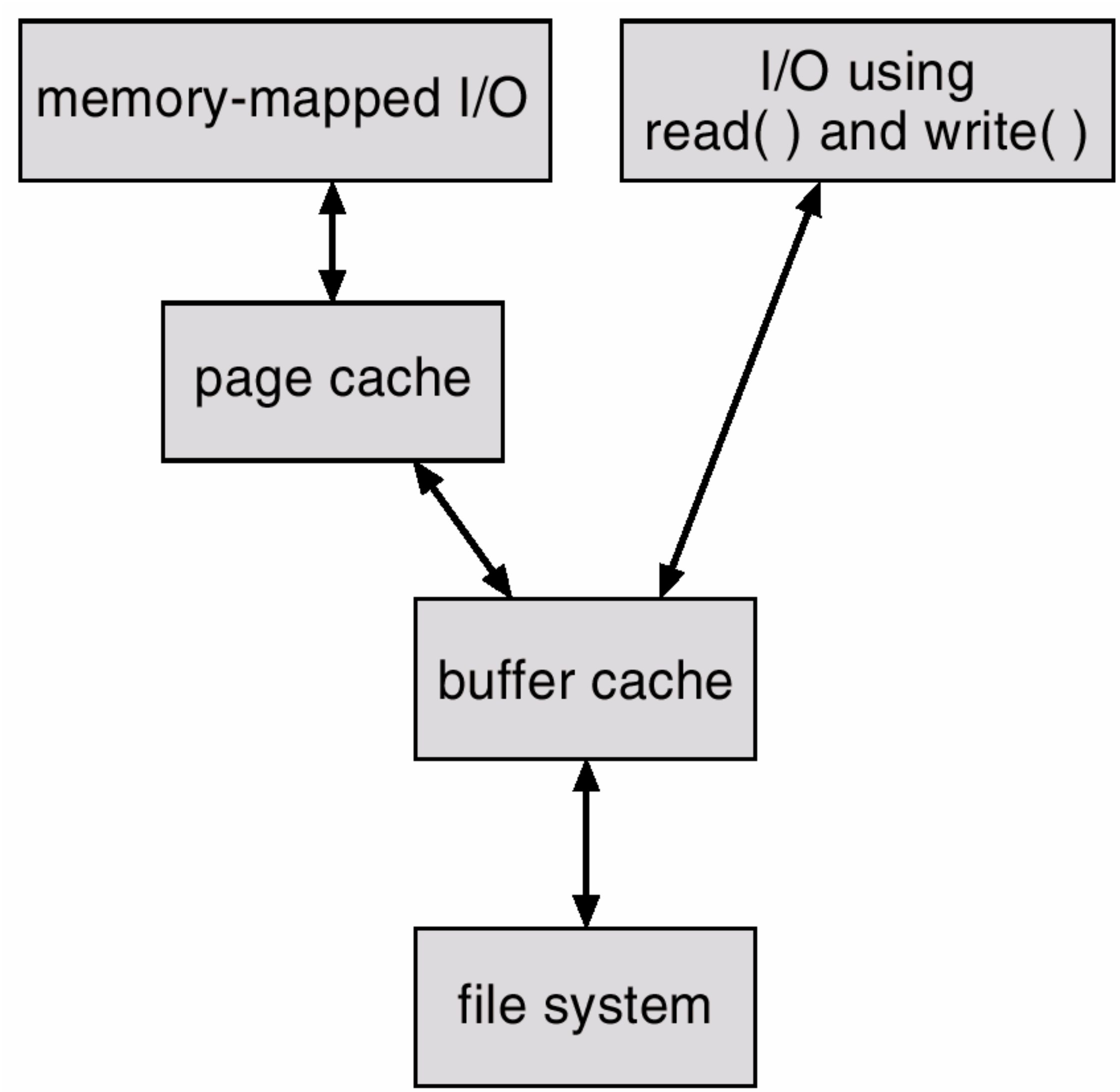
Various Disk-Caching Locations



Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques.
- Memory-mapped I/O uses a page cache.
- Routine I/O through the file system uses the buffer (disk) cache.
- This leads to the following figure.

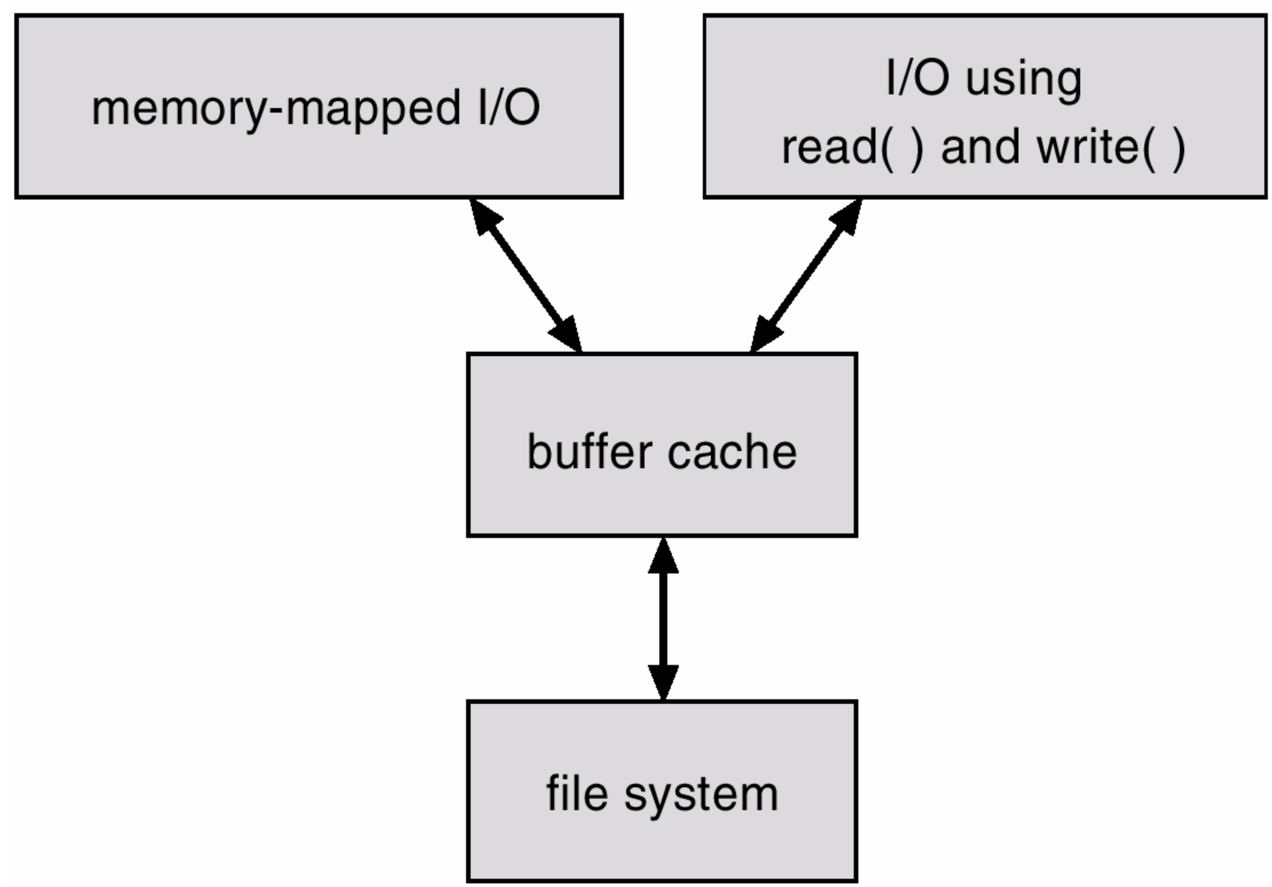
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.

I/O Using a Unified Buffer Cache



Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.

Log Structured File Systems

- **Log structured** (or journaling) file systems record each update to the file system as a **transaction**.
- All transactions are written to a **log**. A transaction is considered **committed** once it is written to the log. However, the file system may not yet be updated.
- The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log.
- If the file system crashes, all remaining transactions in the log must still be performed.

Windows 2000 File System

- Key features of NTFS
  - Recoverability
  - Security
  - Large disks and large files
  - Multiple data streams
  - General indexing facility

Windows NTFS Attributes

Attribute Type	Description
Standard information	Includes access attributes (read-only, read/write, etc.); time stamps, including when the file was created or last modified; and how many directories point to the file (link count).
Attribute list	A list of attributes that make up the file and the file reference of the MFT file record in which each attribute is located. Used when all attributes do not fit into a single MFT file record.
File name	A file or directory must have one or more names.
Security descriptor	Specifies who owns the file and who can access it.
Data	The contents of the file. A file has one default unnamed data attribute and may have one or more named data attributes.
Index root	Used to implement folders.
Index allocation	Used to implement folders.
Volume information	Includes volume-related information, such as the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or folder.

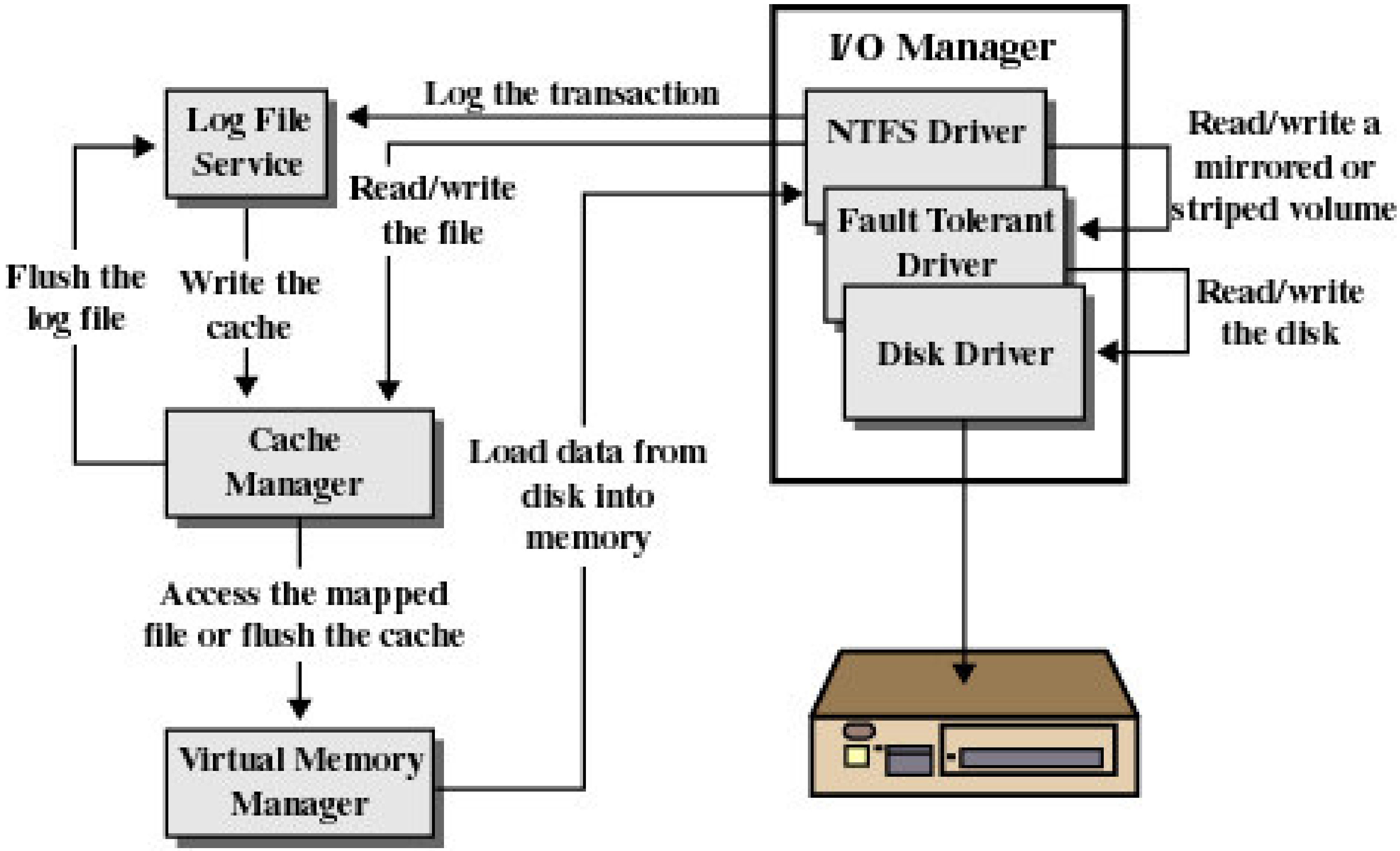


Figure 12.15 Windows NTFS Components [CUST94]