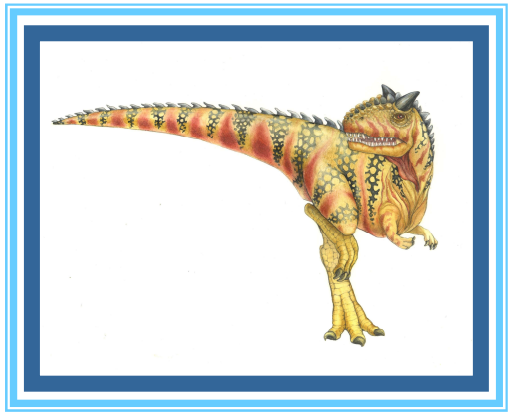
Chapters 13 and 14: File-System



File Concept

- The operating system abstracts from the physical properties of its storage devices to define a **logical storage unit**, the file.
- A file is a named collection of related information that is recorded on secondary storage
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents defined by file's creator
  - Many types
    - **text file,**
    - **source file,**
    - **executable file**

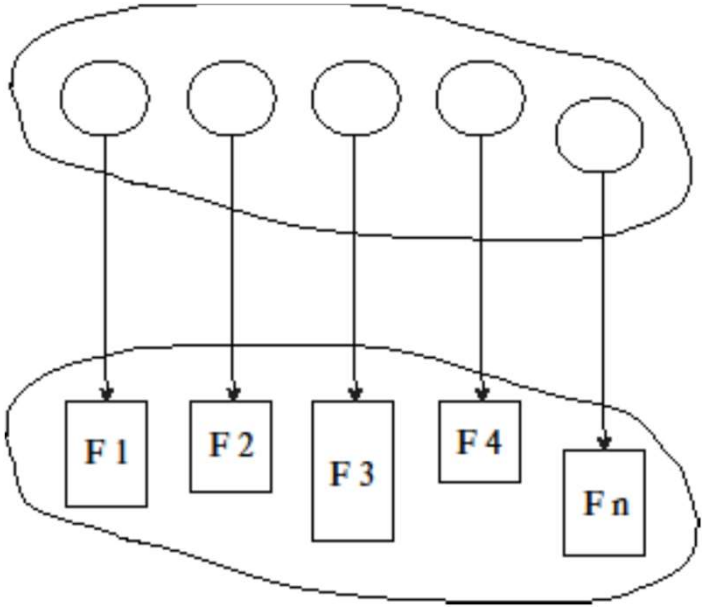
File Attributes

- **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 info Window on Mac OS X

Directory Structure

- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

File Operations

- File is an **abstract data type**
  - **Create**
  - **Write –** at **write pointer** location
  - **Read –** at **read pointer** location
  - **Reposition within file - seek**
  - **Delete**
  - **Truncate**

- To avoid searching the directory for the file entry, many systems require to open the file first
  - ***Open*** **(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory (open file table)
  - ***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

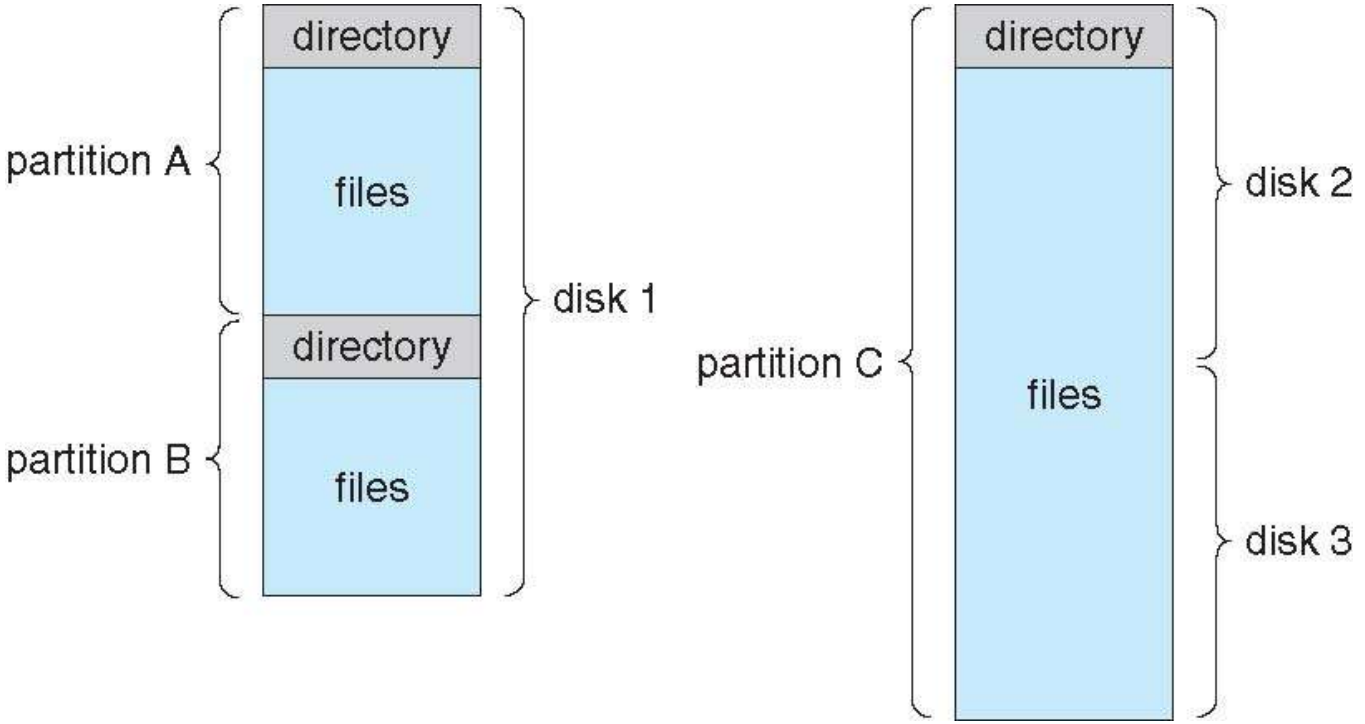
Access Methods

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

Disk Structure

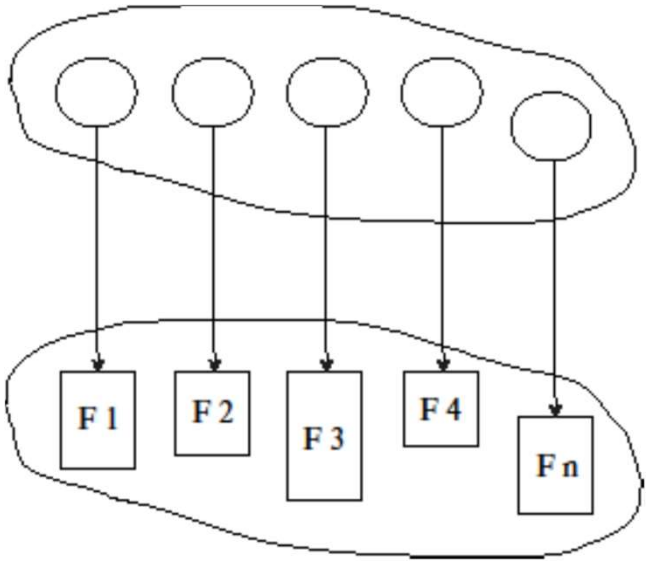
- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **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

A Typical File-system Organization



Directory Structure

- A collection of nodes containing information about all files



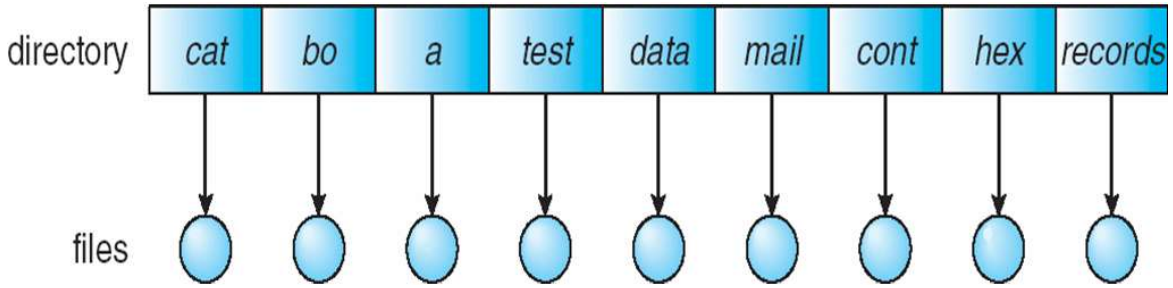
- Both the directory structure and the files reside on disk

Operations Performed on Directory

- The directory can be viewed as a symbol table that translates file names into their directory entries
- Operations:
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

Single-Level Directory

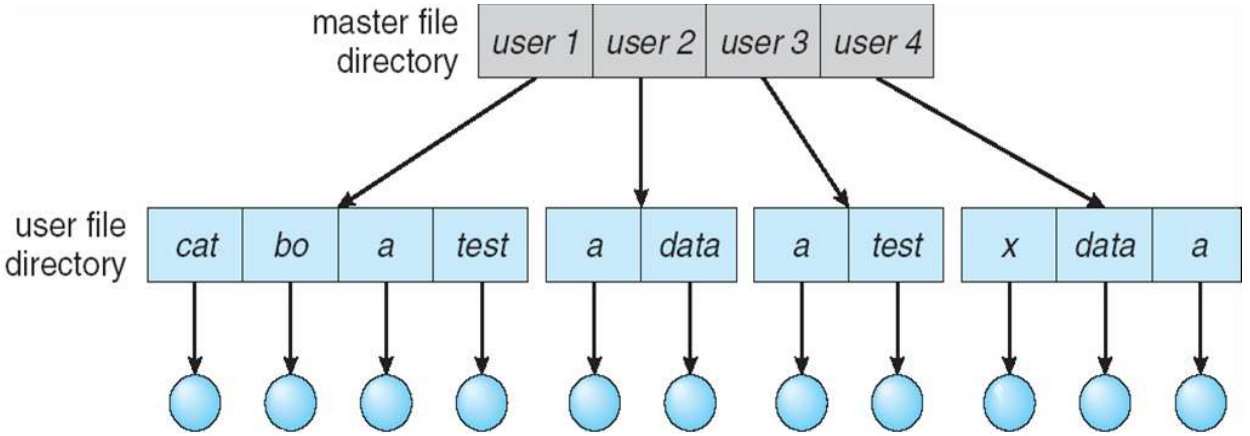
- A single directory for all users
- All files are contained in the same directory, which is easy to support and understand



- Naming problem
  - File names must be unique, even in multi-user environment

Two-Level Directory

- Separate directory for each user

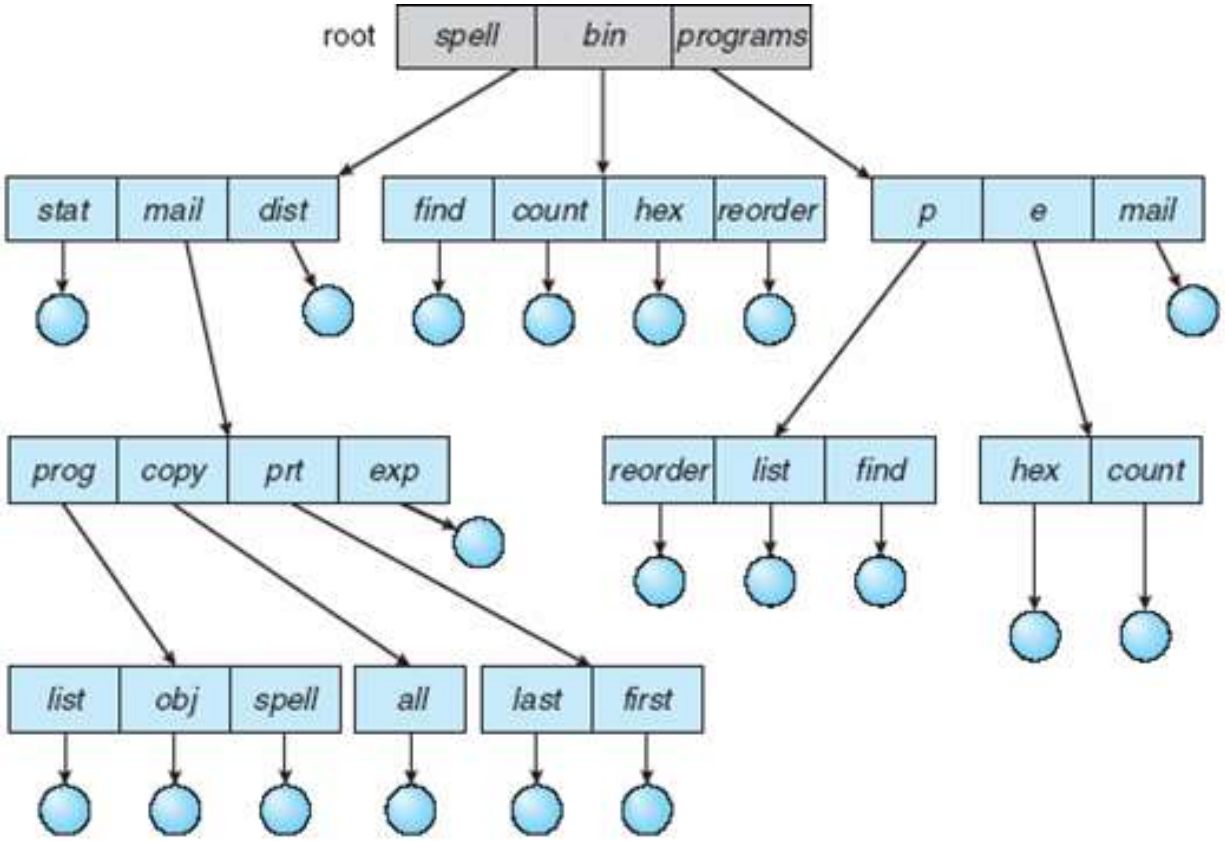


- Path name
- Can have the same file name for different user
- Efficient searching
- User File Directory (UFD) and Master File Directory (MFD)
  - When a user refers to a particular file, only his own UFD is searched
  - Users can access other files (if permitted) using their path names.
  - Example: /user1/test

Tree-Structured Directories

- Two-level directory can be generalized to a tree of arbitrary height
  - This allows users to create their own subdirectories and to organize their files accordingly
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
  - A directory is simply another file, but it is treated in a special way

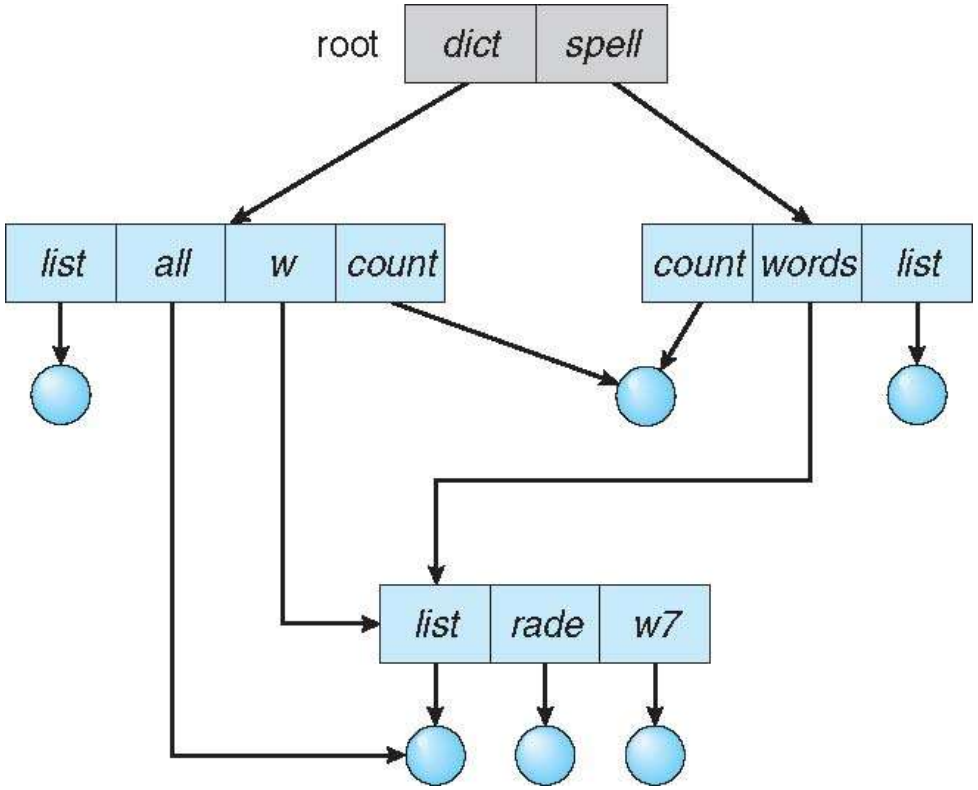
Tree-Structured Directories



- Current Directory
- Absolute and Relative path name
  - Example: spell/mail/prt/first

Acyclic-Graph Directories

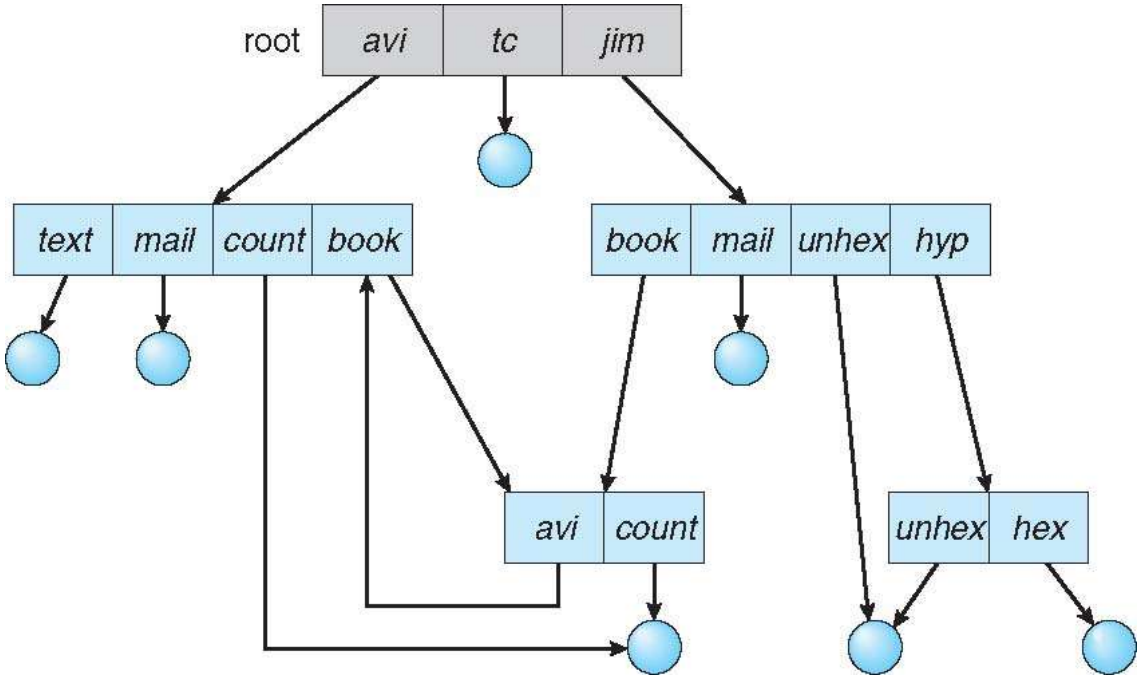
- Have shared subdirectories and files which exist in the file system in two (or more) places at once
- Example



Acyclic-Graph Directories (Cont.)

- Important: a shared file (or directory) is not the same as two copies of the file
  - With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other
- Implementation
  - Create a new directory entry called a **link**, i.e., a pointer to another file or subdirectory
  - Duplicate all information about shared file or directory in both sharing directories
    - Problem: maintaining consistency when a file is modified

General Graph Directory



- Problem: We want to avoid searching any component twice, for reasons of correctness as well as performance.
  - A poorly designed algorithm might result in an infinite loop

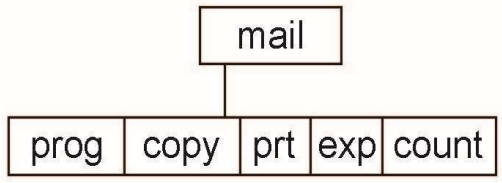
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to files not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

Current Directory

- Can designate one of the directories as the current (working) directory
  - `cd /spell/mail/prog`
  - `type list`
- Creating and deleting a file is done in current directory
- Example of creating a new file
  - If in current directory is `/mail`
  - The command

    mkdir <dir-name>
  - Results in:



  - Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Protection

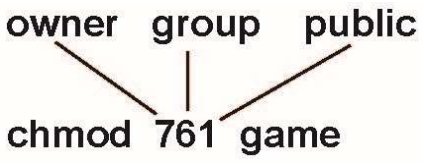
- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

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.



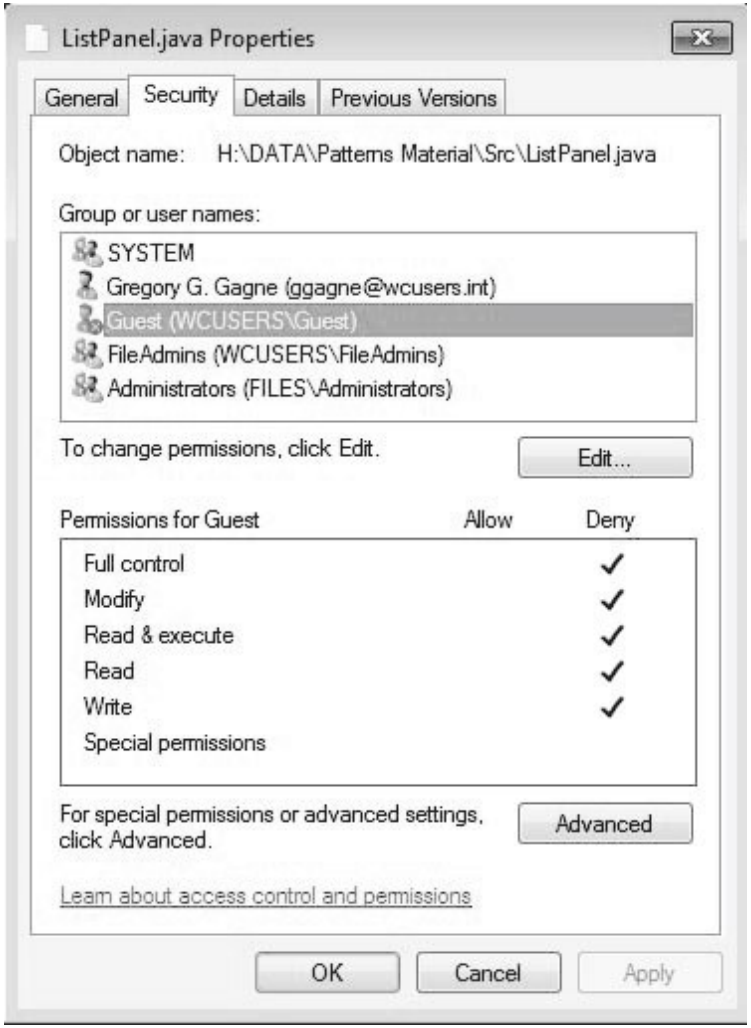
- Attach a group to a file

```
chgrp       G       game
```

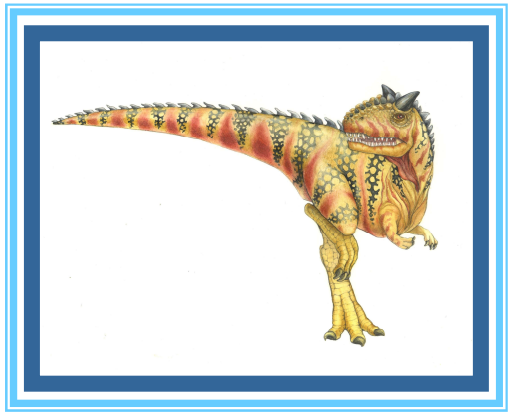
A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Windows 7 Access-Control List Management



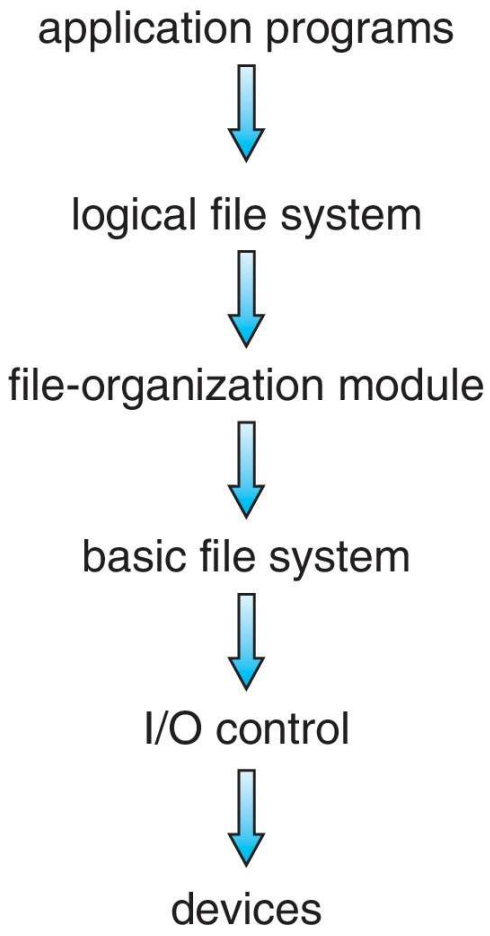
Chapter 14: File System Implementation



File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** (**FCB**) – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

Layered File System



File-System Operations

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

File Control Block (FCB)

- OS maintains **FCB** per file, which contains many details about the file
  - Typically, inode number, permissions, size, dates
  - Example

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

Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

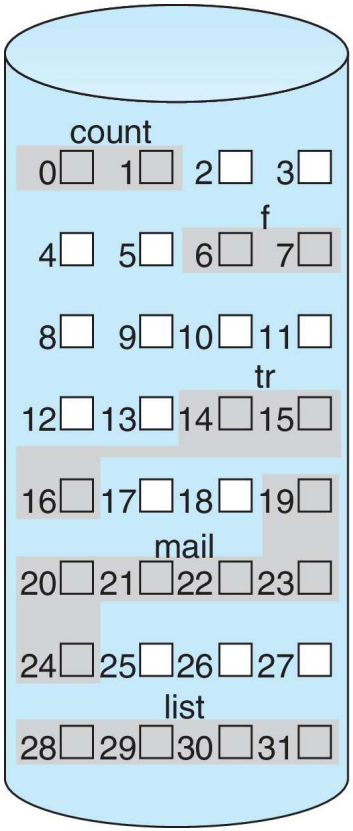
Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous
  - Linked
    - File Allocation Table (FAT) – MS DOS and Windows
  - Indexed - Unix

Contiguous Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
- Each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include:
    - Finding space on the disk for a file,
    - Knowing file size,
    - External fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation (Cont.)



directory

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

Extent-Based Systems

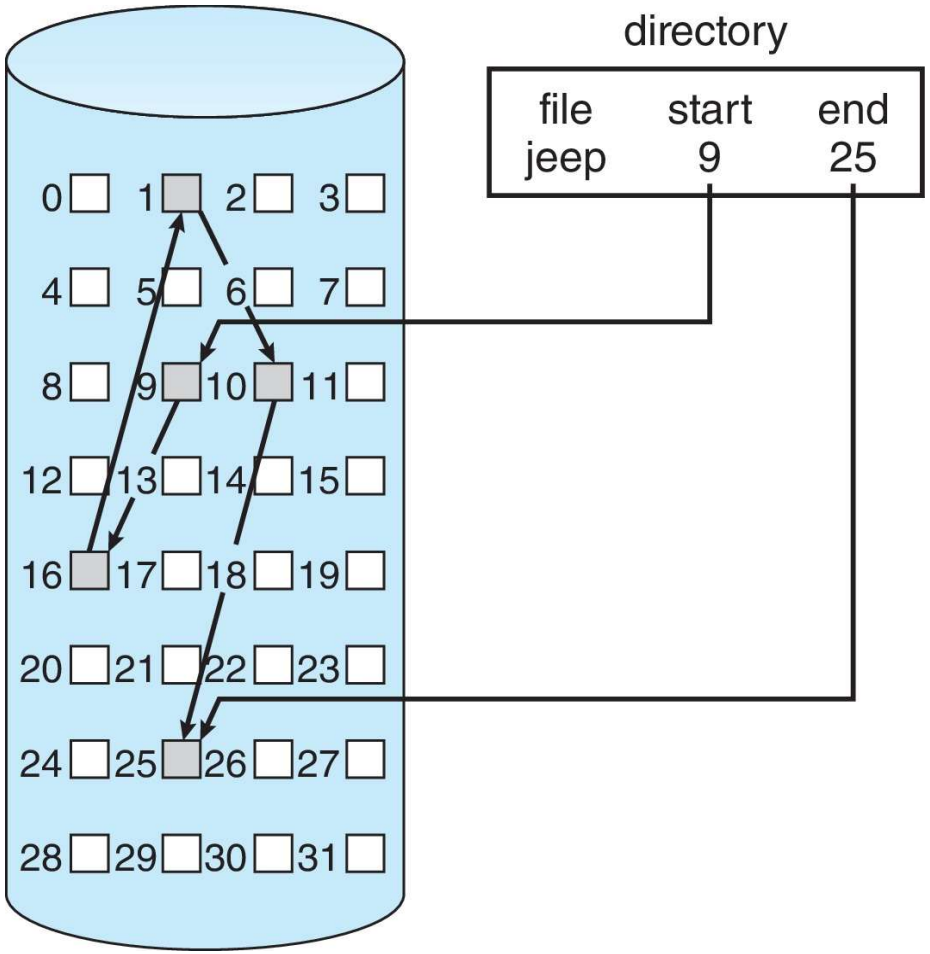
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Linked Allocation

- Each file is a linked list of blocks
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks

Linked Allocation Example

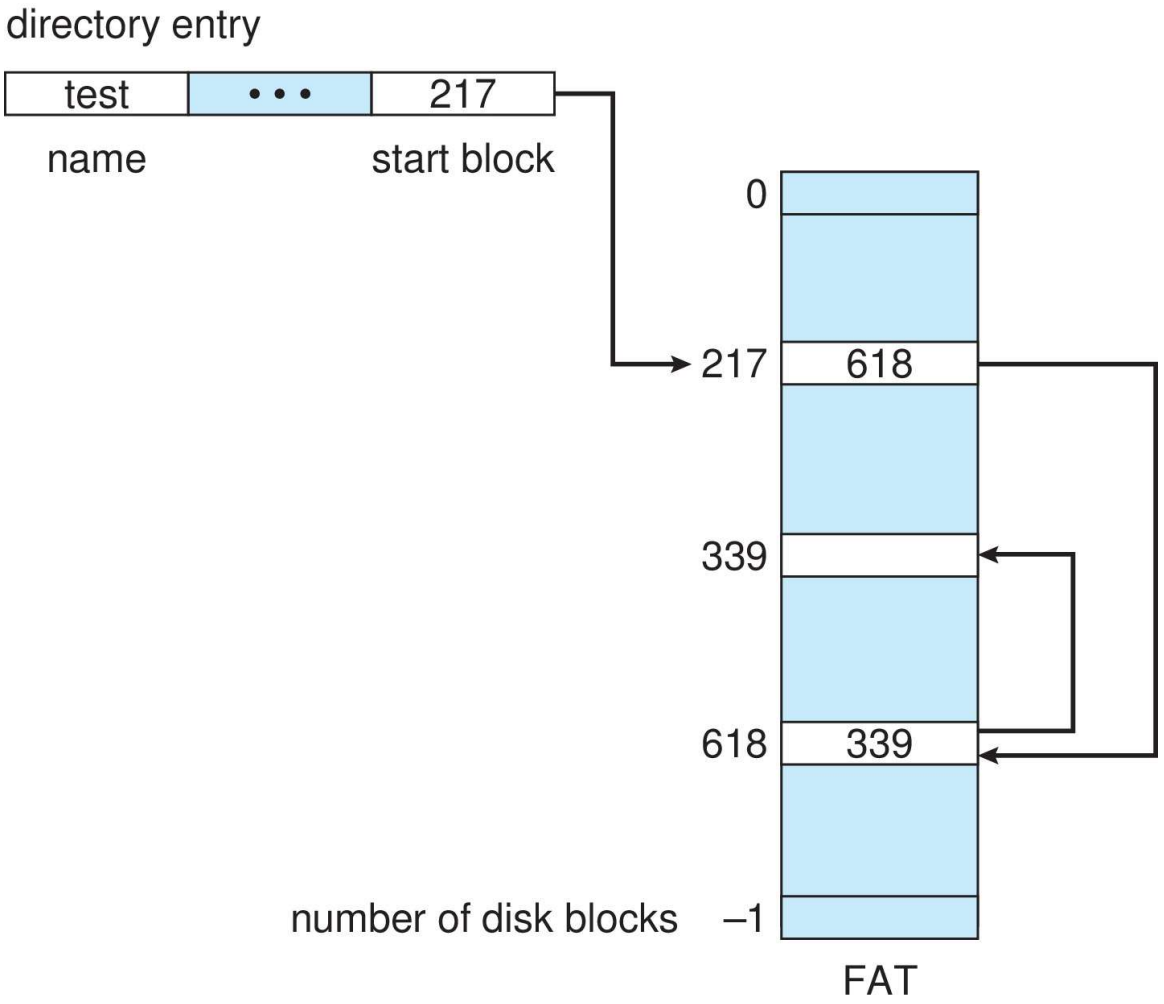
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- Scheme



FAT Allocation Method

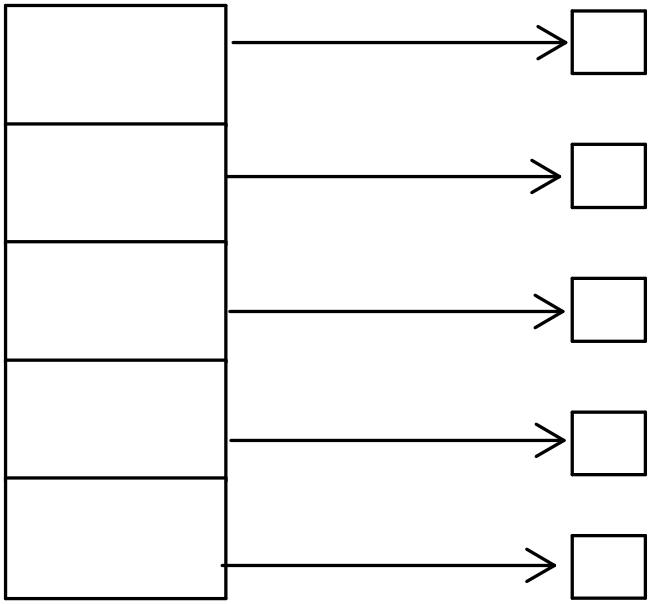
- Beginning of volume has table, indexed by block number
- Much like a linked list, but faster on disk and cacheable
- New block allocation simple

File-Allocation Table

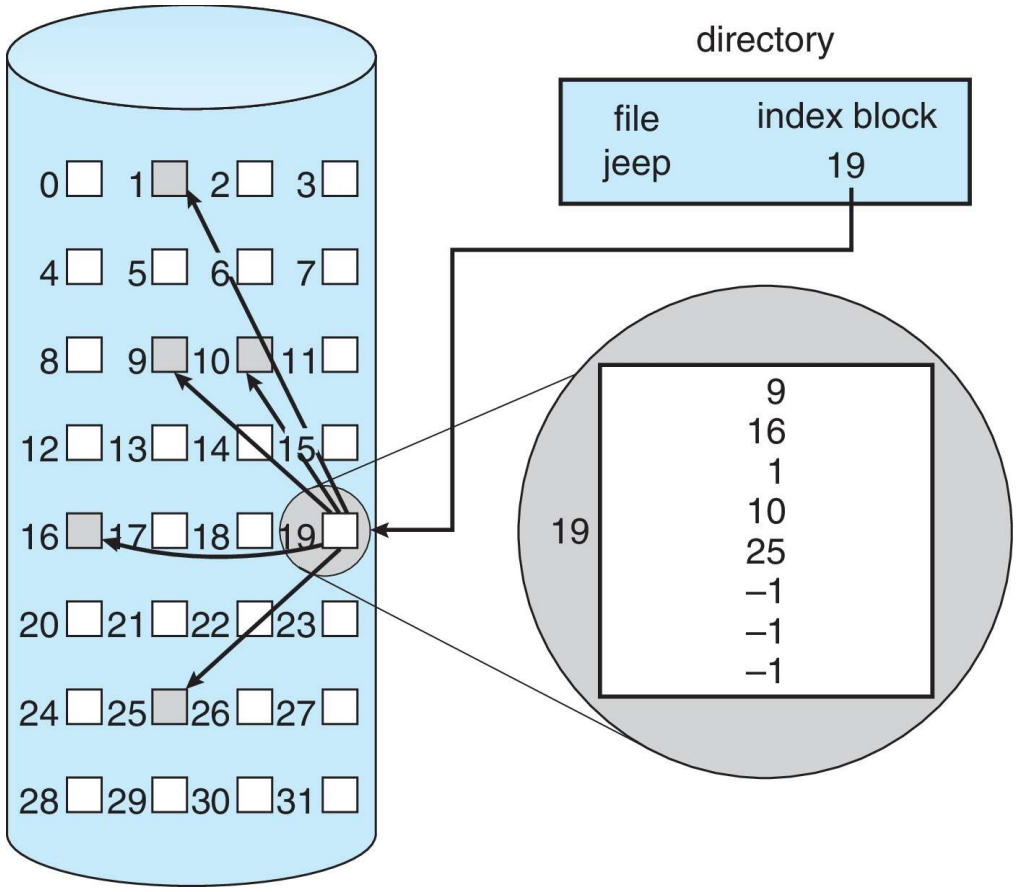


Indexed Allocation Method

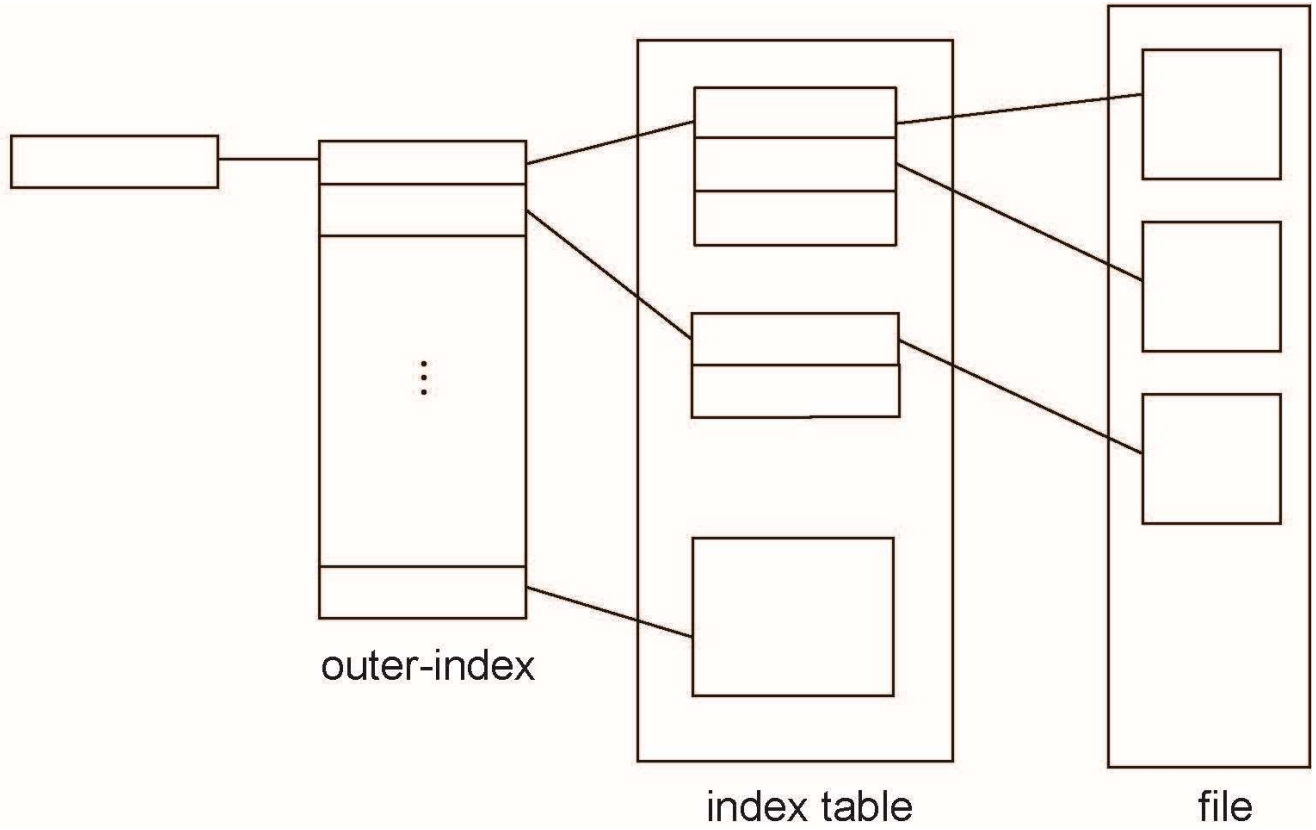
- Each file has its own **index block**(s) of pointers to its data blocks
- Logical view



Example of Indexed Allocation

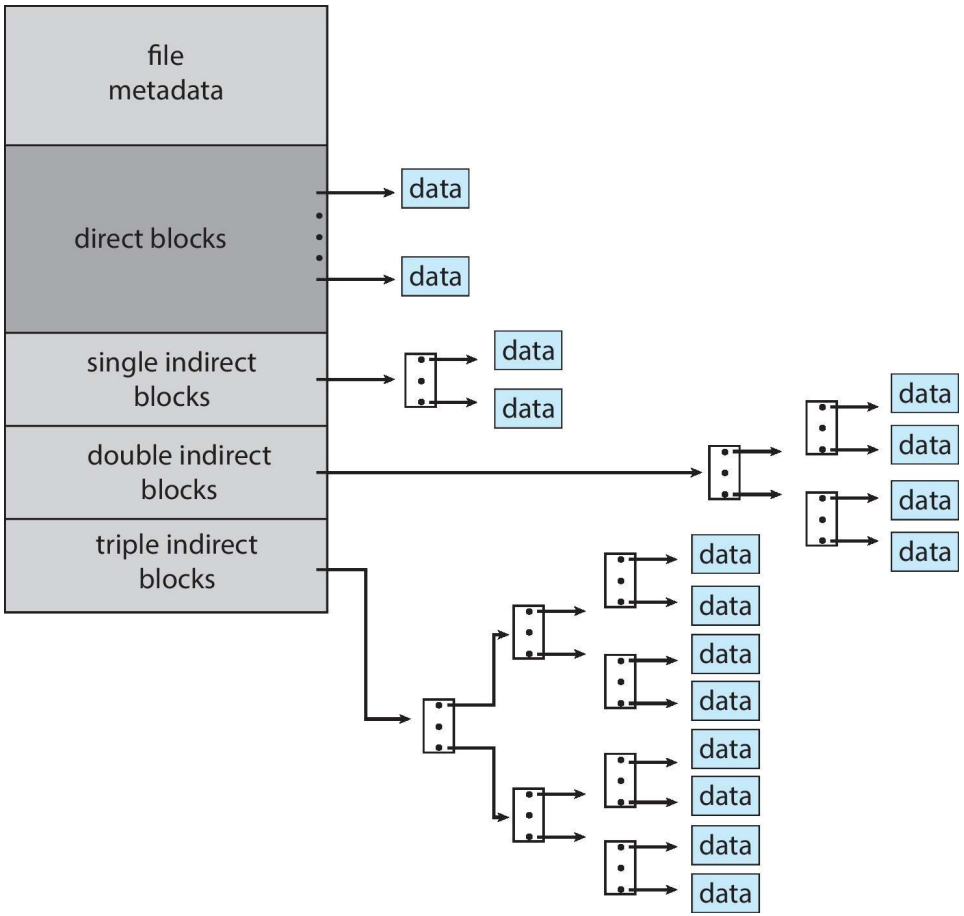


Indexed Allocation – Two-Level Scheme



Combined Scheme : UNIX UFS

- 4K bytes per block, 32-bit addresses



- More index blocks than can be addressed with 32-bit file pointer