File Management

File System Architecture

- Field
- Records
- Files
- Database

Field

Roll No.	Name	Address	Telephone	....
865472	Jhony	North East P.	985647234	....
....	.....	.....	.....	.....
.....	.....	.....	.....	.....

Record

FILE

File

- File is a named collection of related information that is recorded on a secondary storage.
- It made of fixed length logical records that allow programs to read and write records rapidly in no particular order.

File Concept

- File system is the most visible aspect of an operating system.
- It consists of 2 parts: collection of files(each storing related data) and a directory structure which organizes and provides all the information about all the files in your system.
- File is the named collection of related information.
- File is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.
- Commonly files represent programs(both source and object forms) and data.

File Attributes

- **Name** – only information kept in human-readable form. Name is usually a string of characters such as example.c. some systems differentiate between upper case and lower case and some don’t.
- **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

File Operations

- File is an **abstract data type. To define a file properly, we need to consider some operations which can be performed on files**
  - **Create**
  - **Write**
  - **Read**
  - **Reposition within file**
  - **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

File Structure

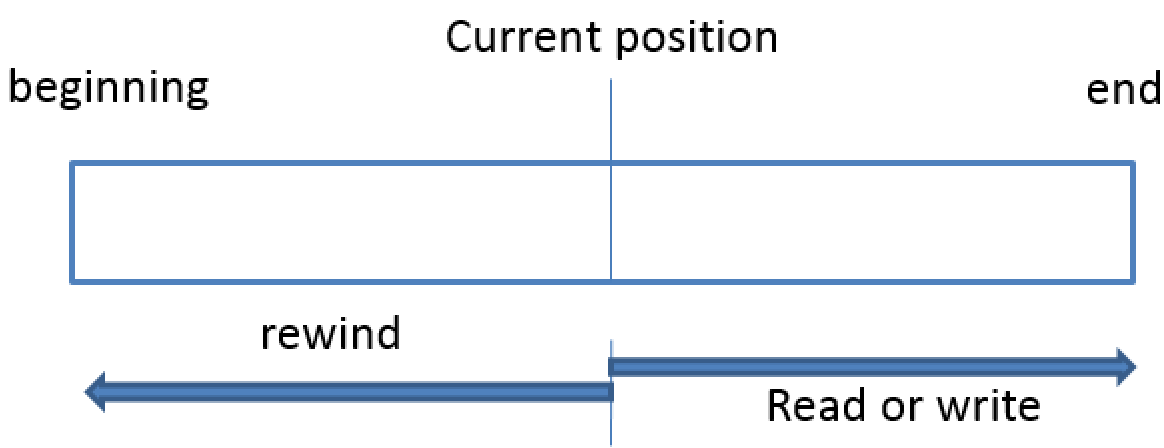
- Internal File Structure
- External File structure
  - Operating system provides various file structures for managing files.
  - Also provides set of special operations for manipulating files with these file structures.

Access Methods

- Files store information and this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways.

Sequential Access

- Information is processed on one order, one record after the other.
- A read operation—*read next*—reads the next portion of the file and automatically
  advances a file pointer, which tracks the I/O location. Similarly, the write operation—*write next*—appends to the end of the file and advances to the end of the newly written material (the new end of file).



Direct Access

- In direct access file is seen as a numbered sequence of blocks or records.
- Immediate access to large amount of information.
- The operations include:
  - *read n,* where *n* is the block number, rather than *read next.*
  - *write n,* where *n* is the block number, rather than *write next.*
  - *position to n,* where *n* is the block number.

- Sequential Access

  read next
  write next
  reset

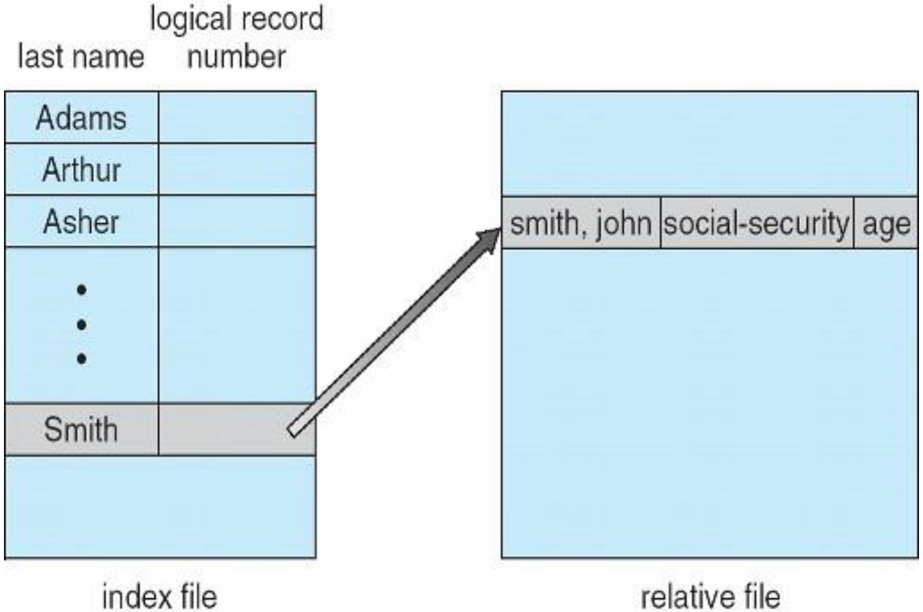
- Direct Access

  read *n*
  write *n*
  position to *n*
      read next
      write next

n = relative block number

Indexed Access Method

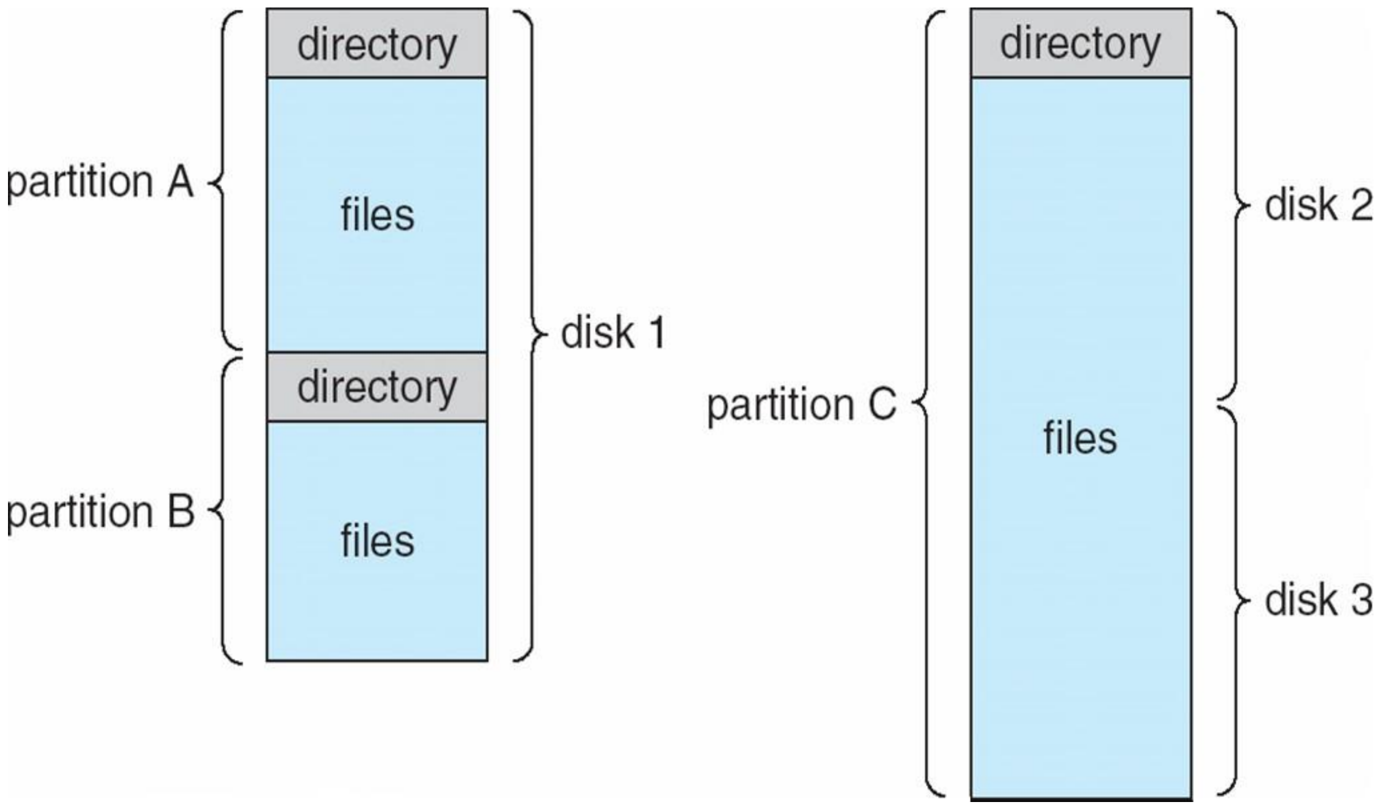
- An *index* is associated with the file containing pointers to the blocks.
- To find the record in the file, we search the index and then use the pointer to access the file directly and to find the desired record.



Disk Structure

- As the number of files increase the issue of managing the files becomes an issue.
- The organization is done in two parts, Firstly:
  - Disk can be subdivided into partitions
  - Disks or partitions can be RAID protected against failure
  - Disk or partition can be formatted with a file system
  - Partitions also known as minidisks, slices or volume
- Secondly, Each volume containing file system also tracks that file system’s info in device directory or volume table of contents.

A Typical File-system Organization

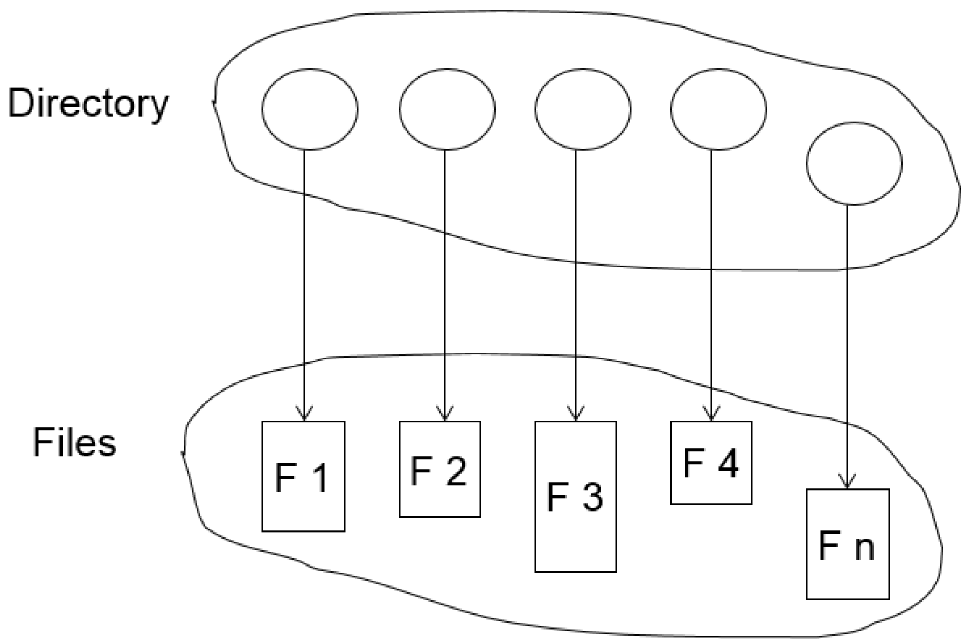


Operations Performed on Directory

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

Directory Structure

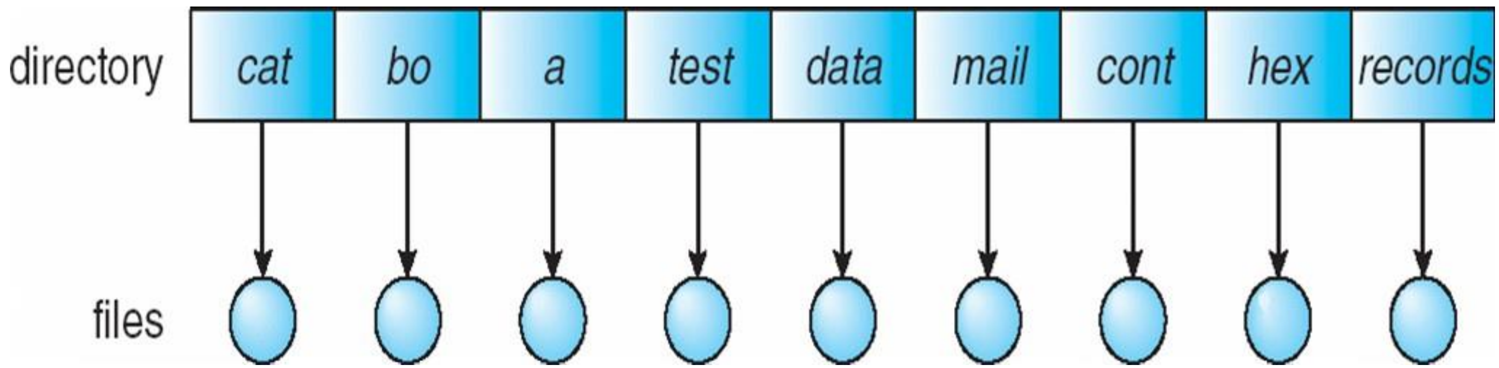
- The directory contains information for all files.



- Both the directory structure and the files reside on disk

Single-Level Directory

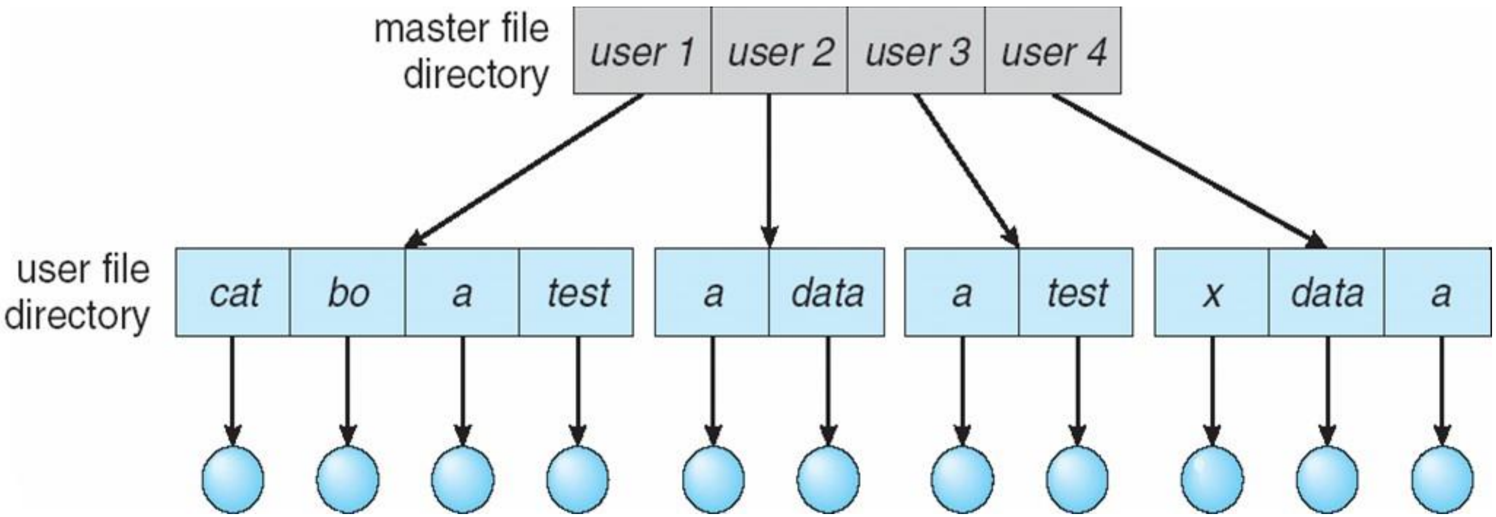
- A single directory for all users



- Easy to support and understand.
- Limitation:
- When number of files increases or when the system has more than one user, then Naming problem occurs. All files should have unique names.

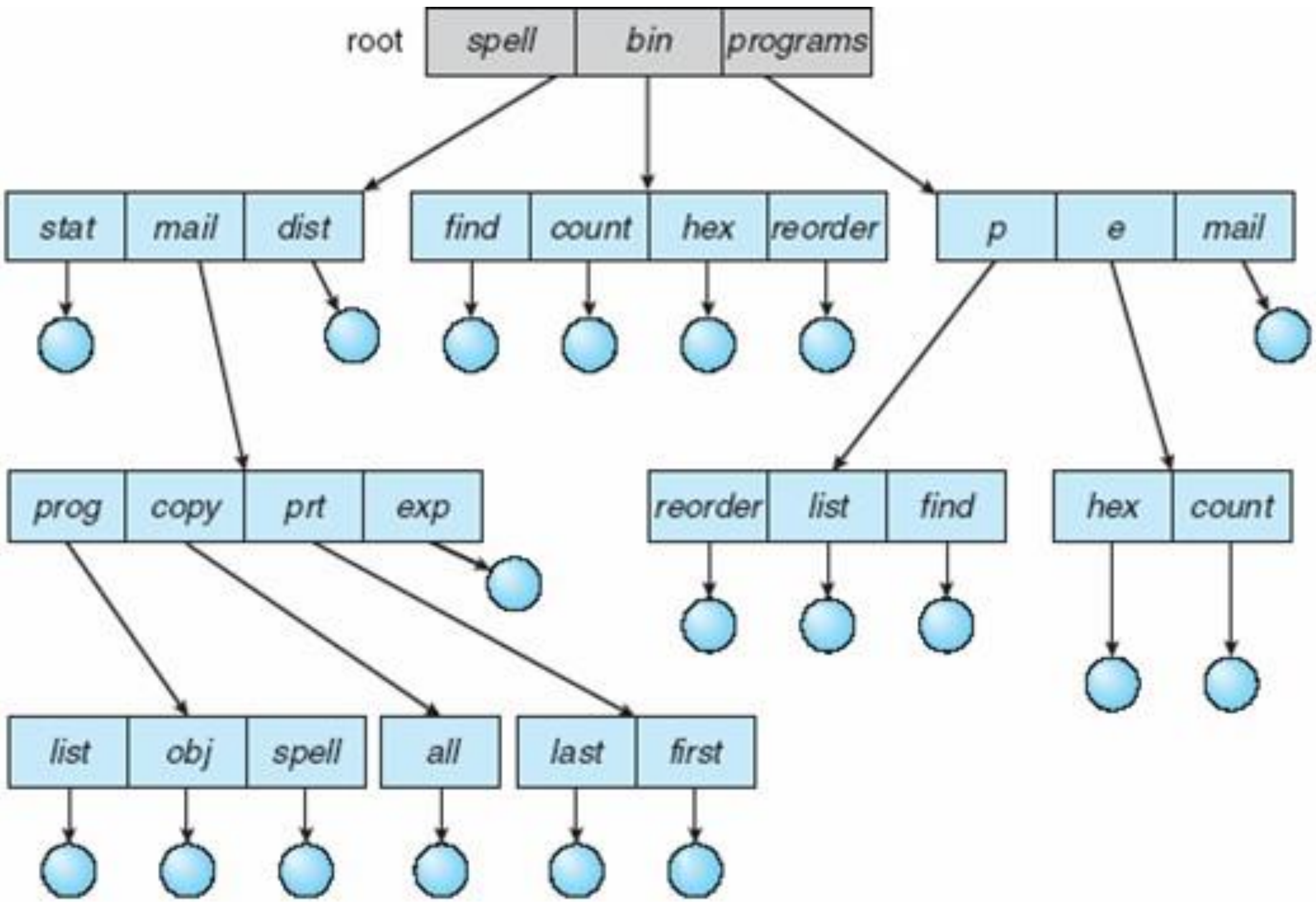
Two-Level Directory

- In two level directory, each user has his own user file directory(UFD). UFDs have the similar structure, but each lists only the files of a single user.



- Path name
- Can have the same file name for different user
- Efficient searching

Tree-Structured Directories



Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

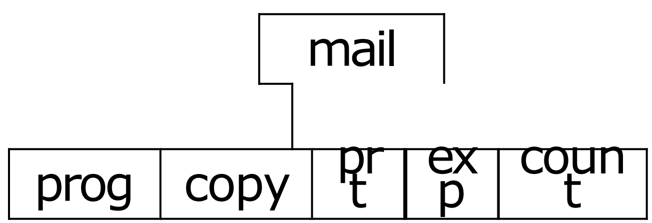
    rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /mail

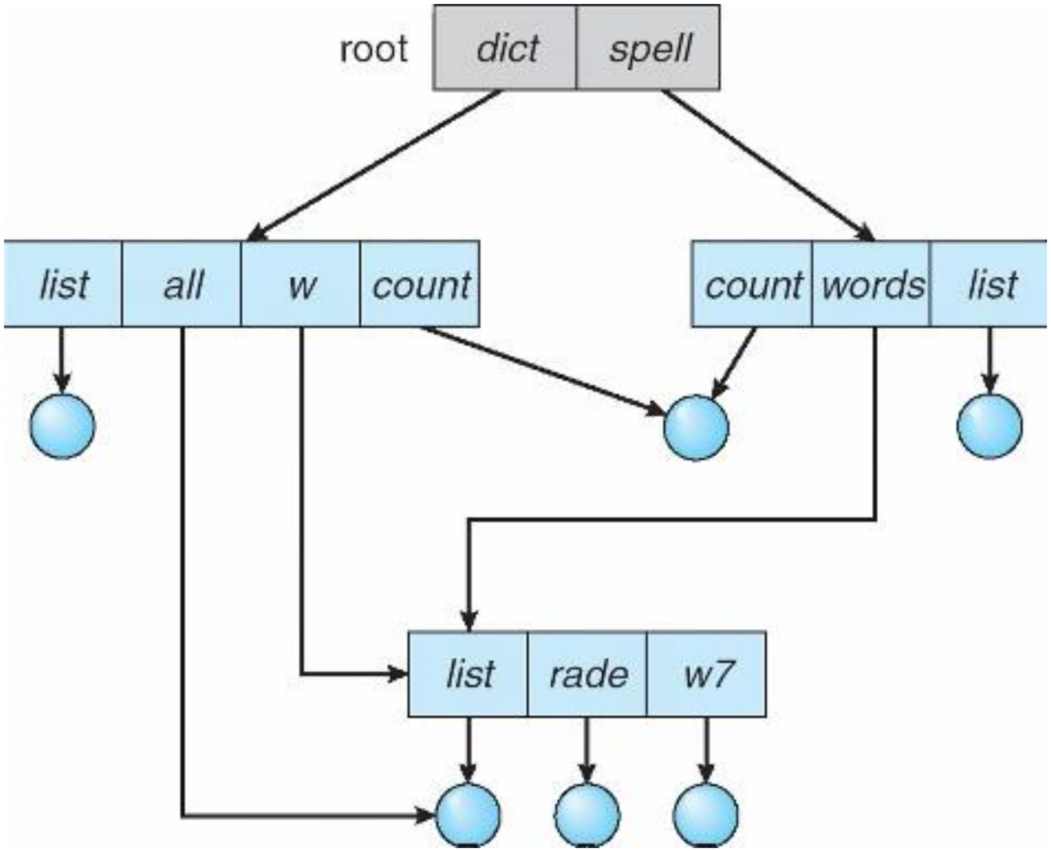
  mkdir count



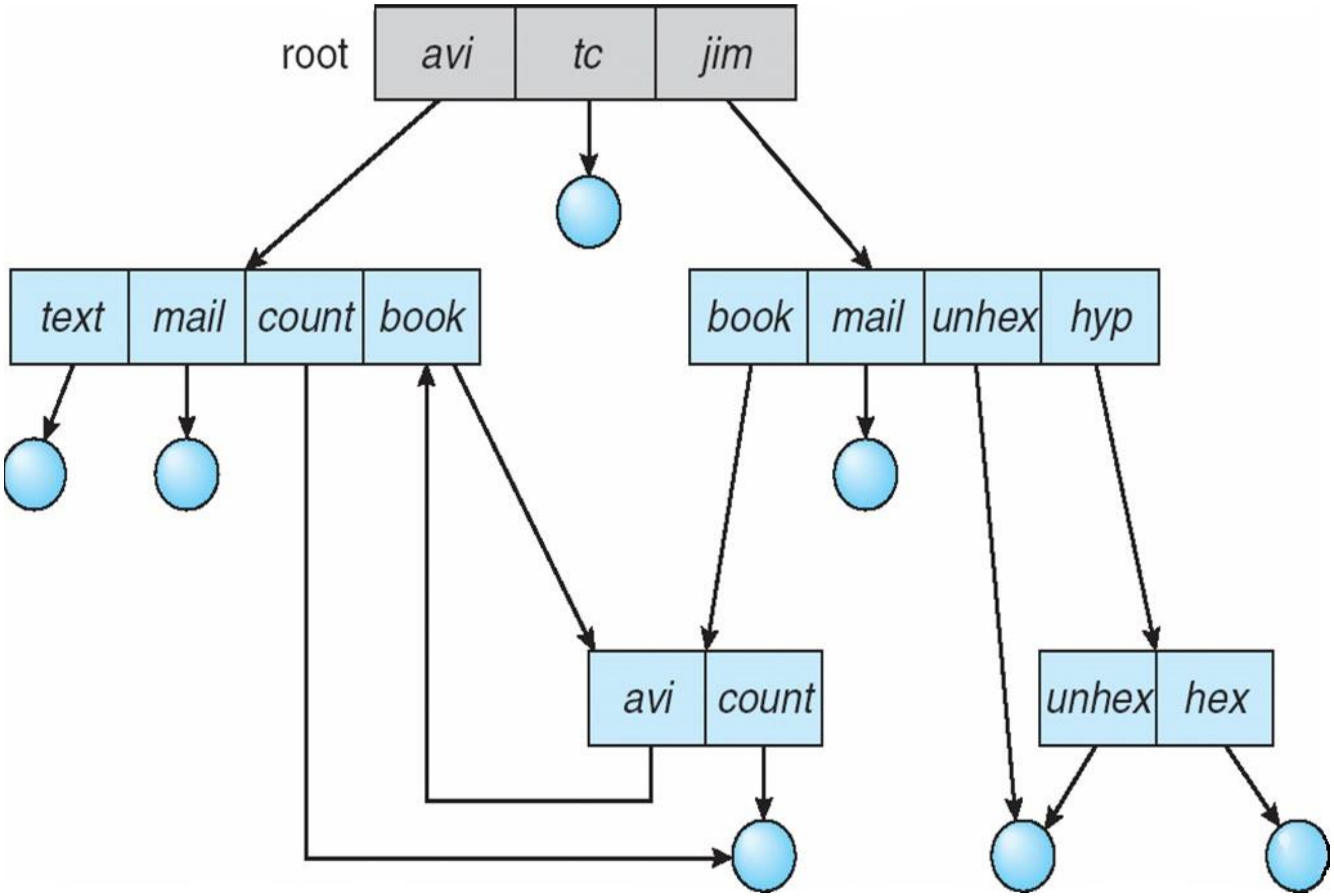
Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Acyclic-Graph Directories

- Have shared subdirectories and files



General Graph Directory



Directory Implementation

1. **Linear List**
   - Simple to program
   - Time consuming to execute
   - Create new file
     - Make sure it's a unique file name
     - Add entry at end of directory
   - Delete a file
     - Search for the file
     - Release the space allocated to it
     - Mark the space as unused or attach it to list of free directory entries or copy the last directory entry here.

2. Hash Table

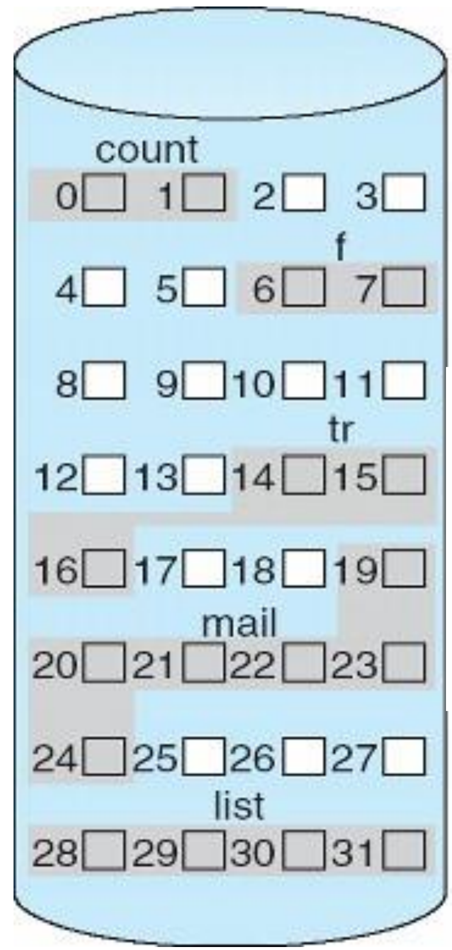
- Linear list + hash structure
- Compute hash value of file names and returns pointer to file in the linear list
- Decreases search time
- Collisions are to be avoided in case two file names hash to same location
- Disadvantage – hash functions are of fixed size (0-64 or 0-128 etc)

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files so that disk space is utilized effectively and files can be accessed quickly.
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Allocation Methods

- Contiguous Allocation
  - File occupy contiguous blocks
  - Directory entry contains
    - File name
    - Starting block
    - Length (total blocks)
  - Access possible
    - Sequential
    - Direct
  - Problems
    - Finding space for new file
    - External fragmentation
    - Determining space requirement of a file

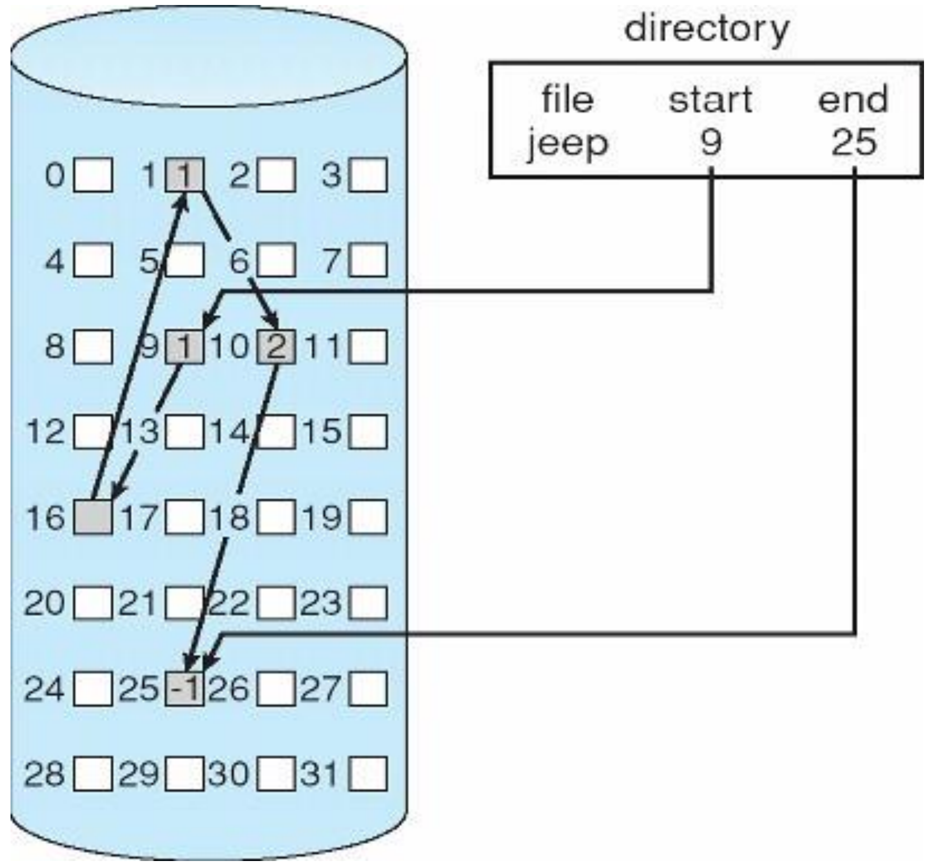


directory

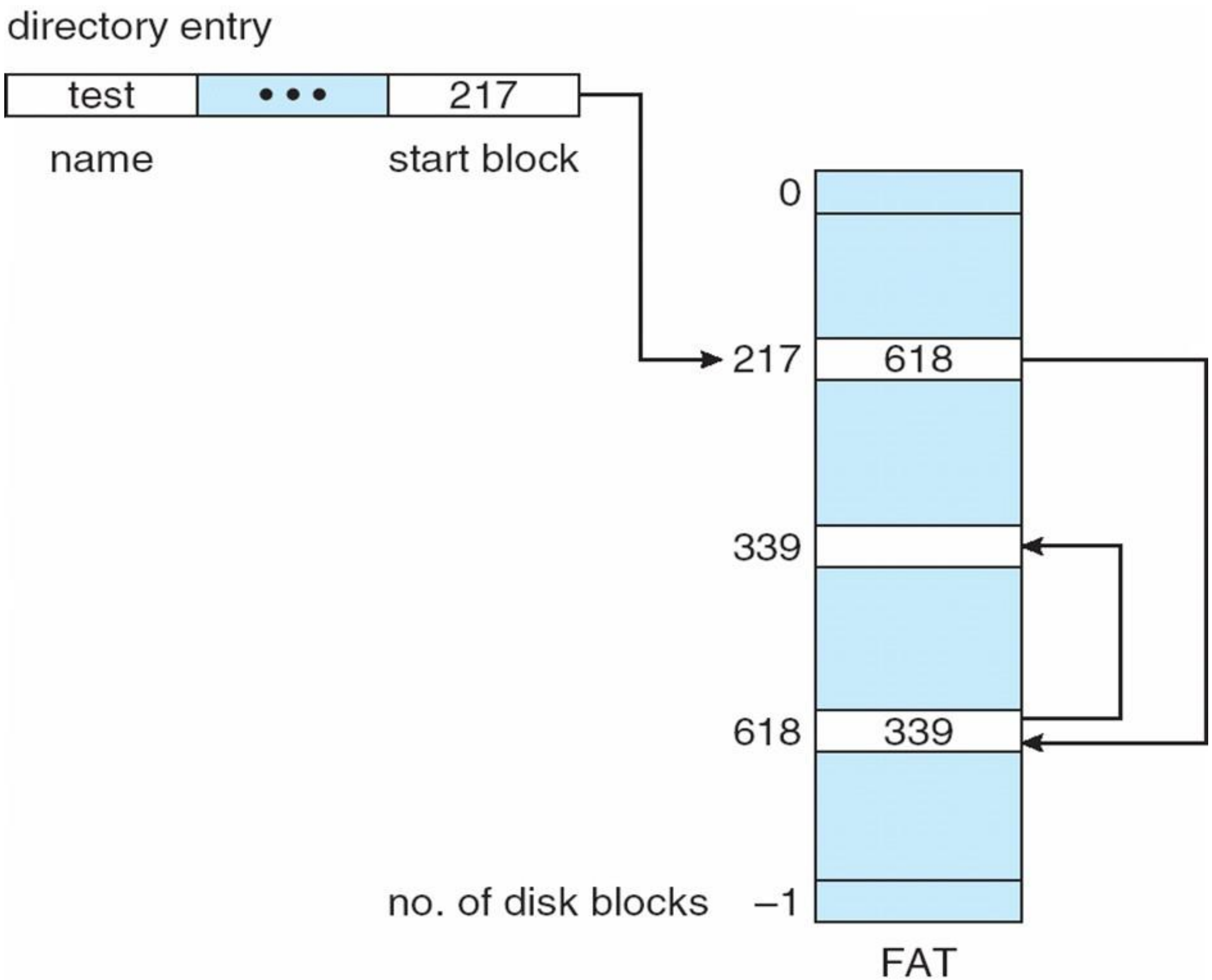
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation

- Directory structure contains
  - Pointer to first and last block of file
- Advantages
  - No external fragmentation
  - No issue with increase in file size
- Disadvantages
  - Only sequential access
  - Reliability – loss of a pointer
  - Space required for pointers
  - Solution: make *cluster* of blocks
    - Problem: internal fragmentation



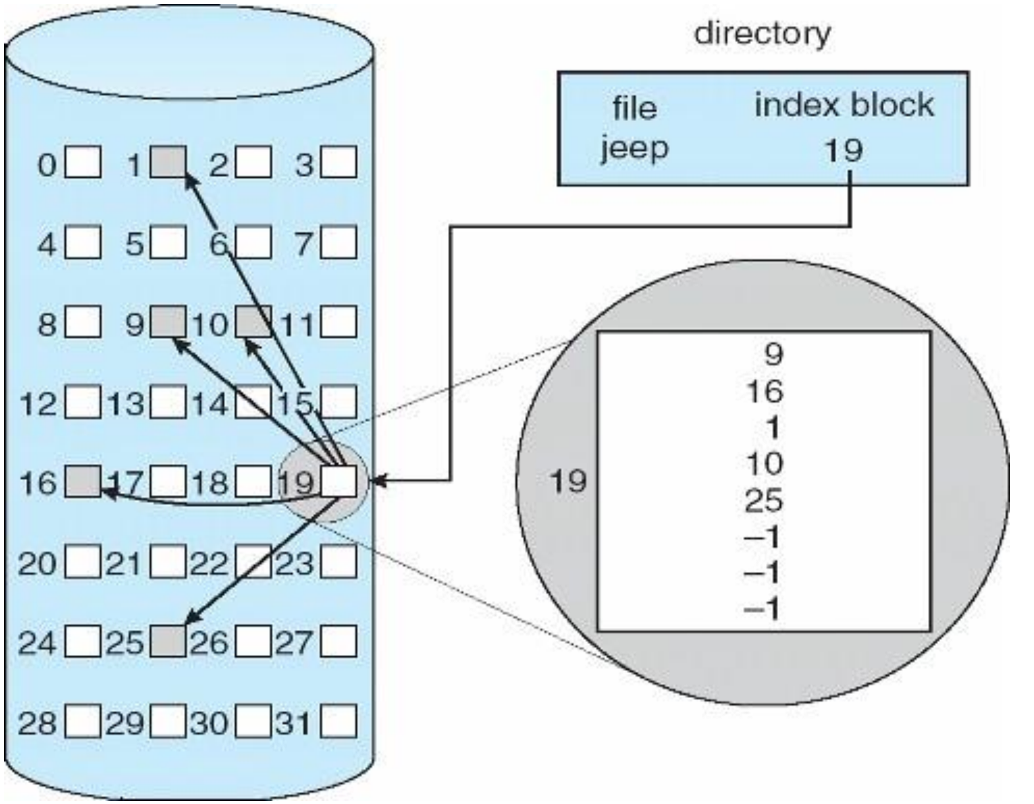
Example - FAT



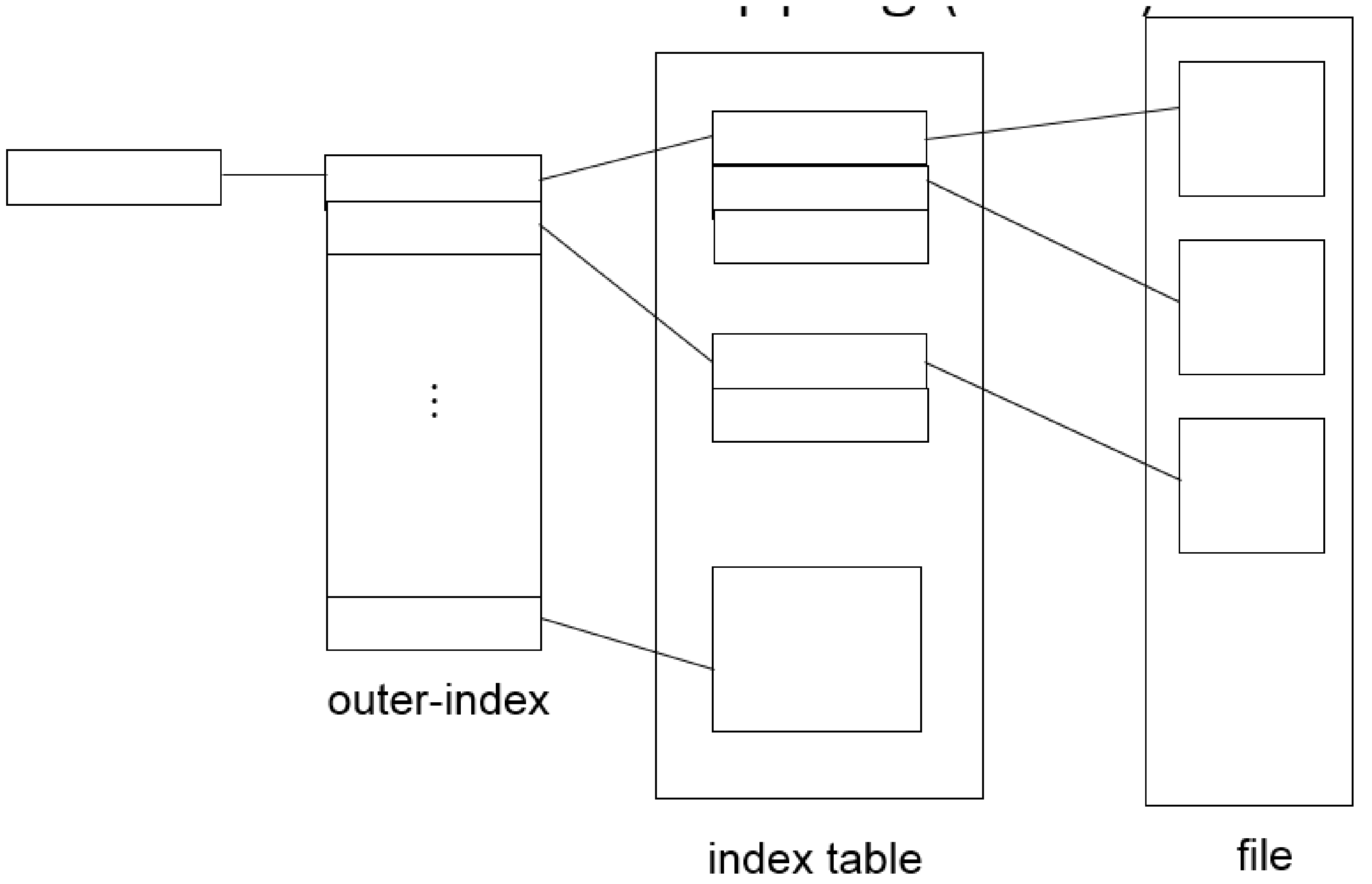
directory entry
test
• • •
217
name
start block
0
217
618
339
618
339
no. of disk blocks
–1
FAT

Indexed Allocation

- Clubs all the pointers into one block – index block
- Directory entry contains
  - File name
  - Index block number
- Access
  - Direct
- Issue
  - Size of index block
  - Sol: multilevel indexing



Indexed Allocation – Mapping (Cont.)



Performance

- Contiguous
  - Requires only 1 access to get a disk block
- Linked
  - Requires *i* disk reads to read i^{th} block
- Indexed
  - Depends on
    - level of indexing
    - Size of file
    - Position of desired block

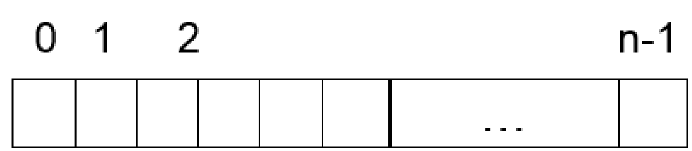
Free Space management

Disk space is limited, the space from deleted files is reused for new files, if possible. To keep track of free disk space, the system maintains a **free-space list.**

1. Bit vector
2. Linked list
3. Grouping
4. Counting

Bit Vector

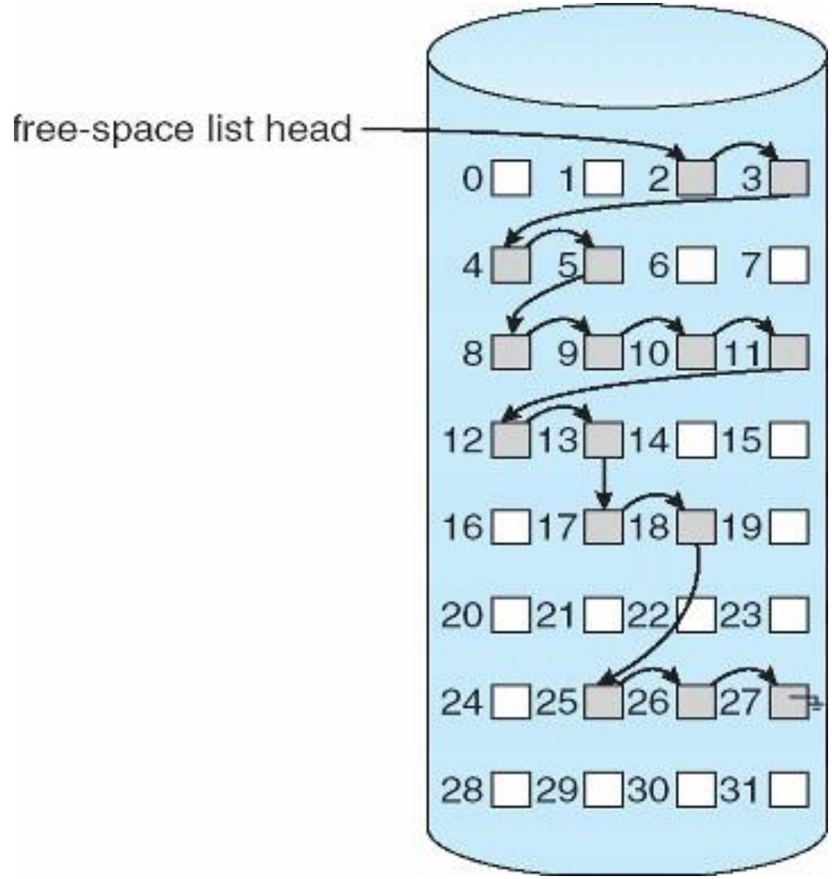
- The free-space list is implemented as a bit **map** or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.



Grouping

- Store the addresses of *n* free blocks in the first free block.
- The first n—1 of these blocks are actually free.
- The last block contains the addresses of another *n* free blocks, and so on.

Counting

- Keep the address of the first free block and the number *n* of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.
- Each entry requires more space than would a simple disk address, overall list will be shorter, as long as the count is generally greater than 1.

Summary

- Files
- File Structure
- Access Methods
- Directory Structure
- Directory Implementation
- Allocation Methods
- Free Space Management