Module 6:
File management

File system structure, allocation methods

Dr. Rishikeshan C A
Assistant Professor (Sr.)
SCOPE, VIT Chennai

Objectives

File Systems :

- File system structure
- Allocation methods (contiguous, linked, indexed)
- Free-space management (bit vector, linked list, grouping)
- Directory implementation (linear list, hash table)

File-System Structure

- File system
  - Provide efficient and convenient access to disk
  - Easy access to the data (store, locate and retrieve)
- Two aspects
  - User's view
    - Define files/attributes, operations, directory
  - Implementing file system
    - Data structures and algorithms to map logical view to physical one

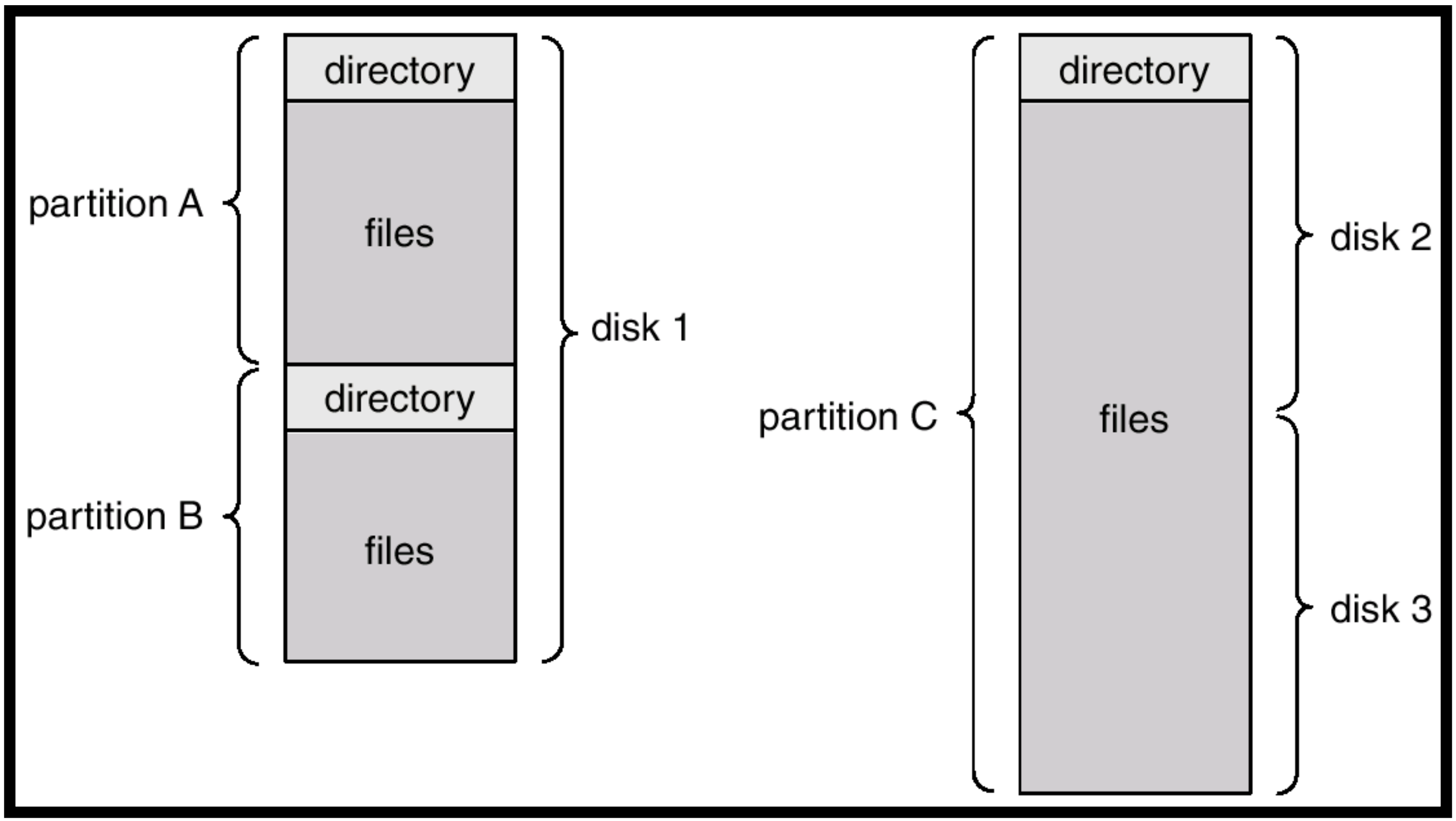
Many File Systems

- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660;
  - Unix has **UFS**, FFS;
  - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray,
  - Linux has more than 40 types, with **extended file system** ext2, ext3 and ext4 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

File Types – Name, Extension

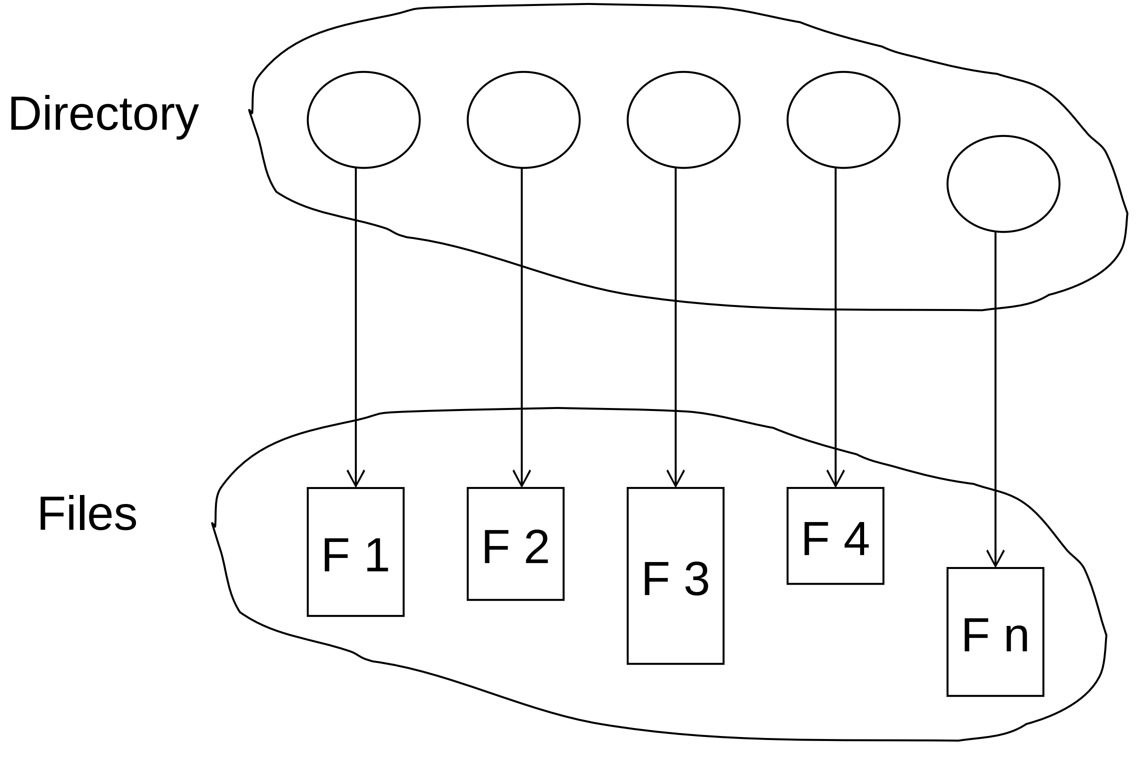
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data,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

A Typical File-system Organization



Directory Structure

- A collection of nodes containing information about all files



Operations Performed on Directory

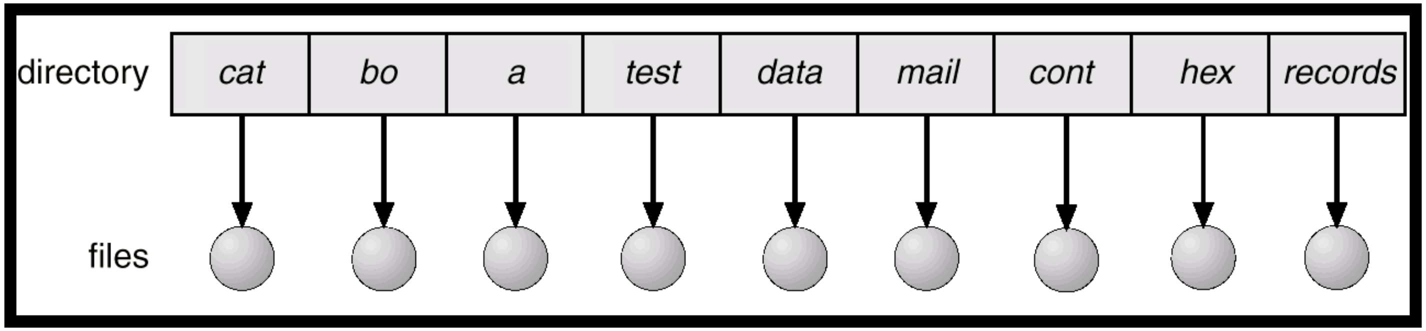
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (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, …)

Single-Level Directory

- A single directory for all users.

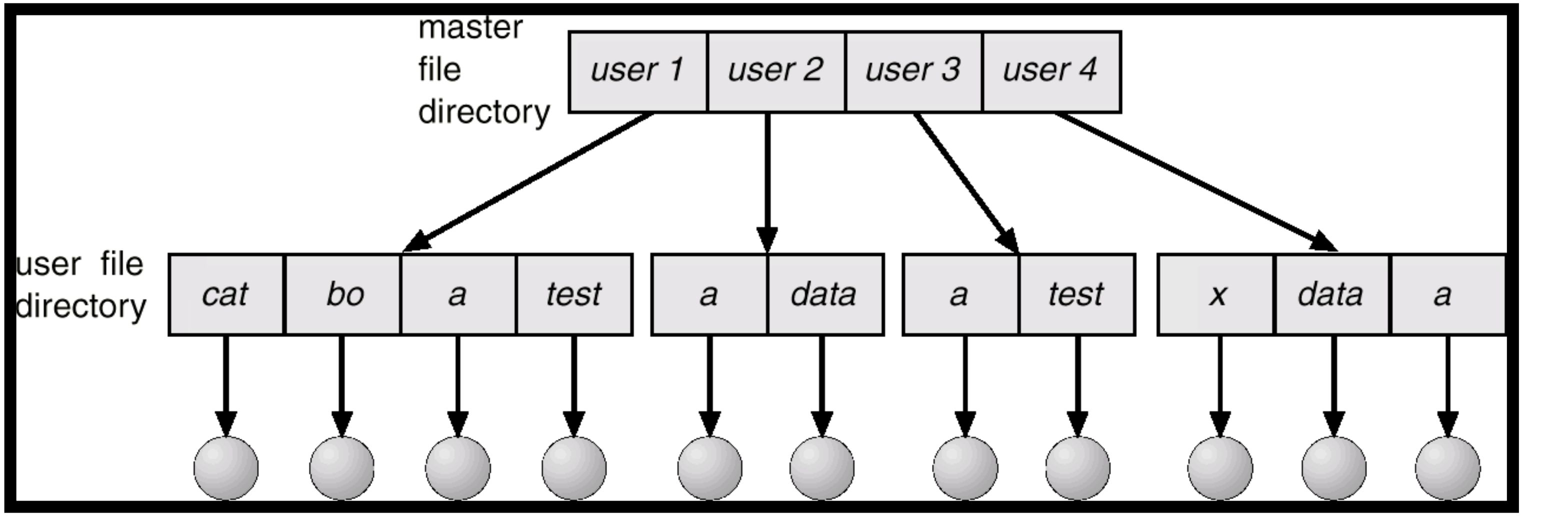


Naming problem

Grouping problem

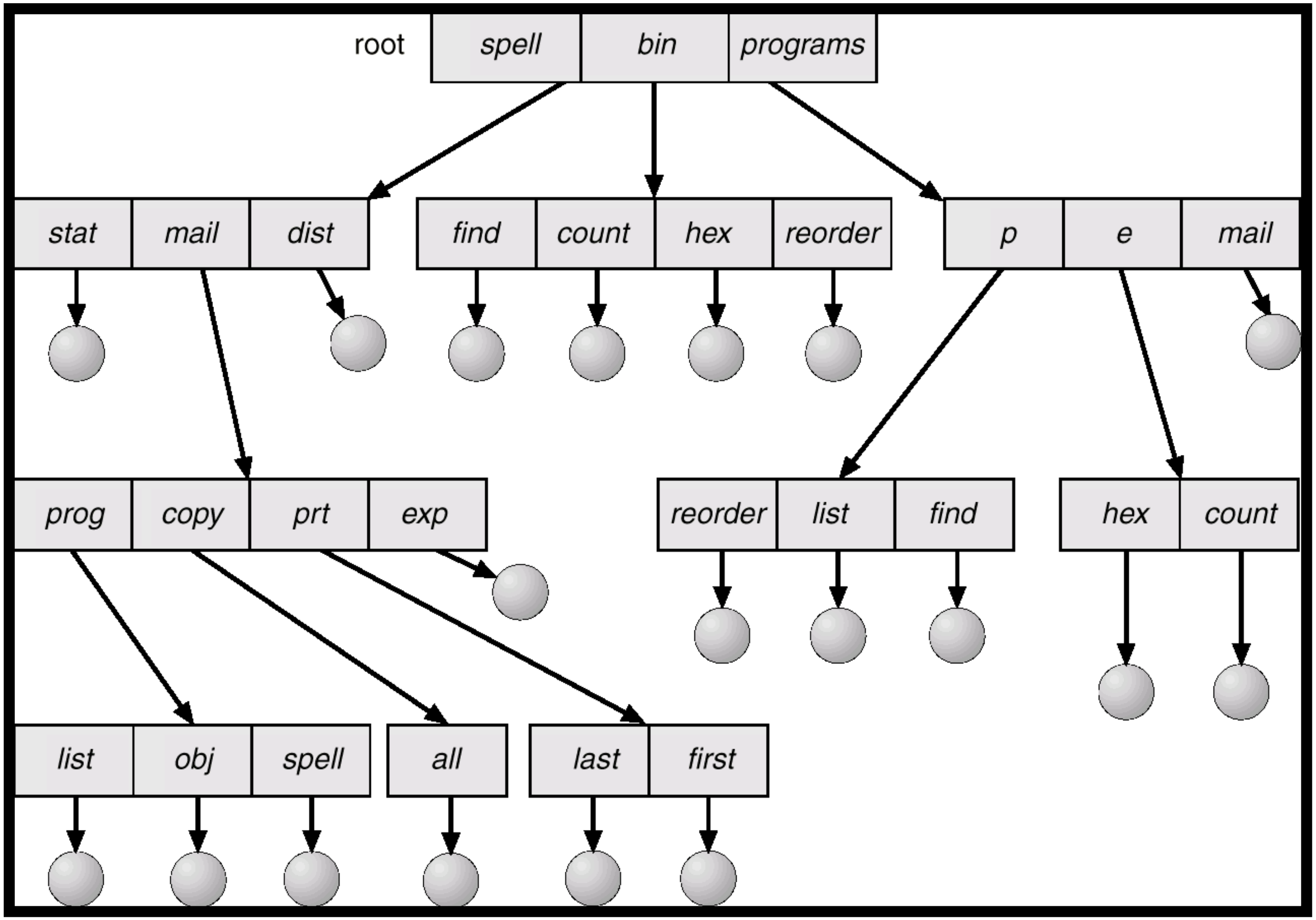
Two-Level Directory

- Separate directory for each user.



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list

Tree-Structured Directories (Cont.)

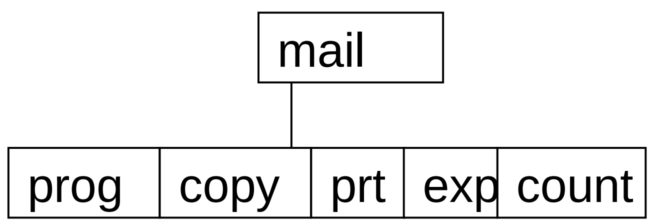
- **Absolute** or **relative** path name
- Creating a new file is done in current directory.
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory.

  mkdir <dir-name>

  Example: if in current directory **/mail**

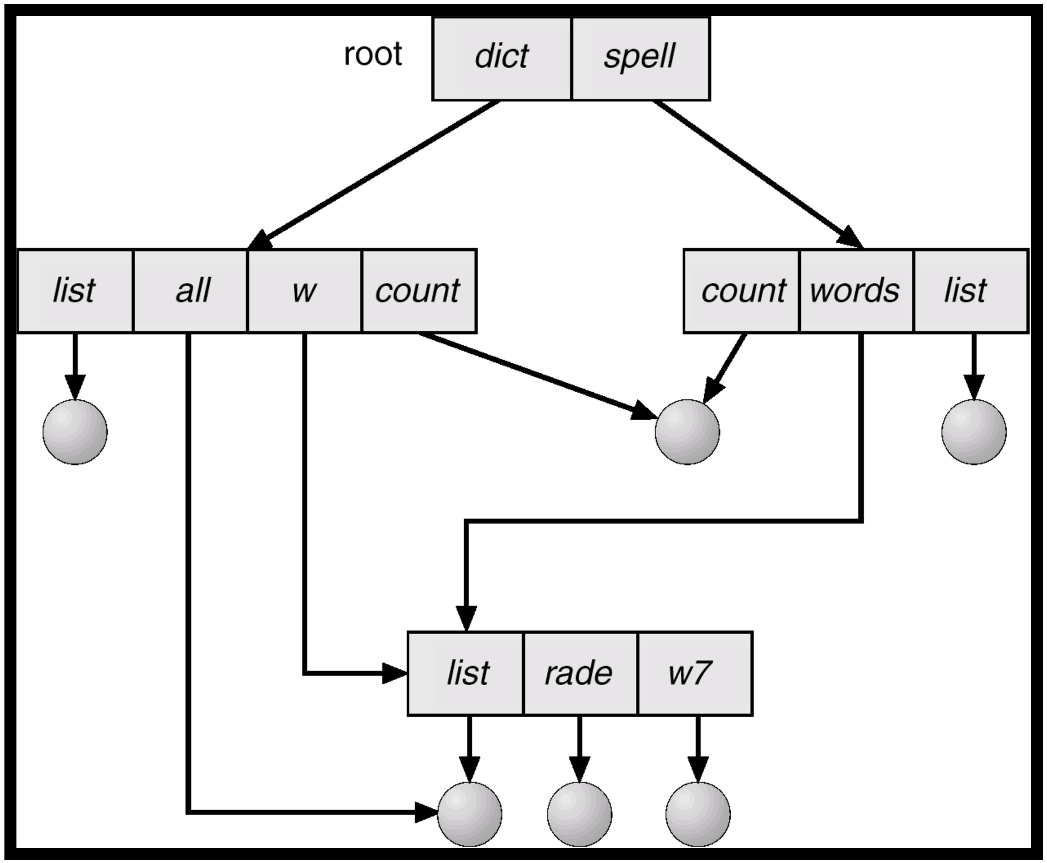
  mkdir count



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

Acyclic-Graph Directories

- Have shared subdirectories and files.

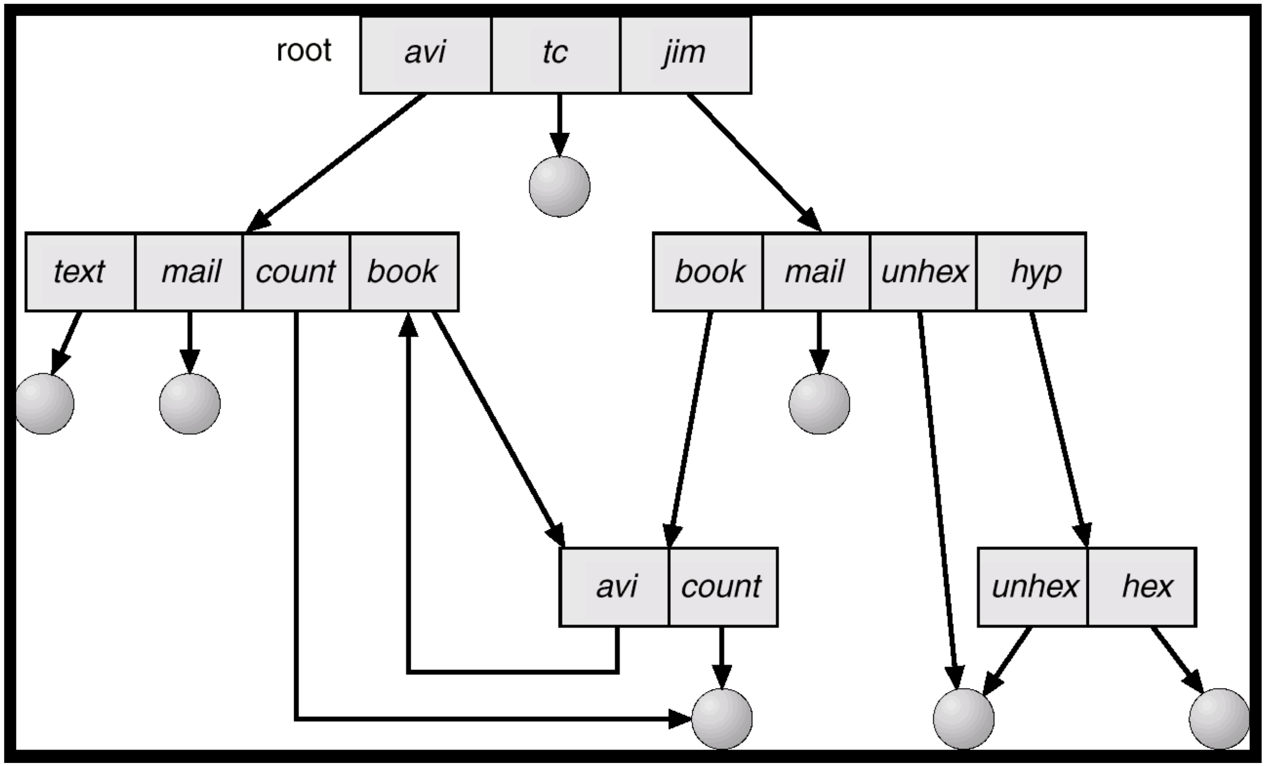


Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *list* $\Rightarrow$ dangling pointer.

  Solutions:
  - Backpointers, so we can delete all pointers.
    Variable size records a problem.
  - Backpointers using a daisy chain organization.
  - Entry-hold-count solution.

General Graph Directory



General Graph Directory (Cont.)

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

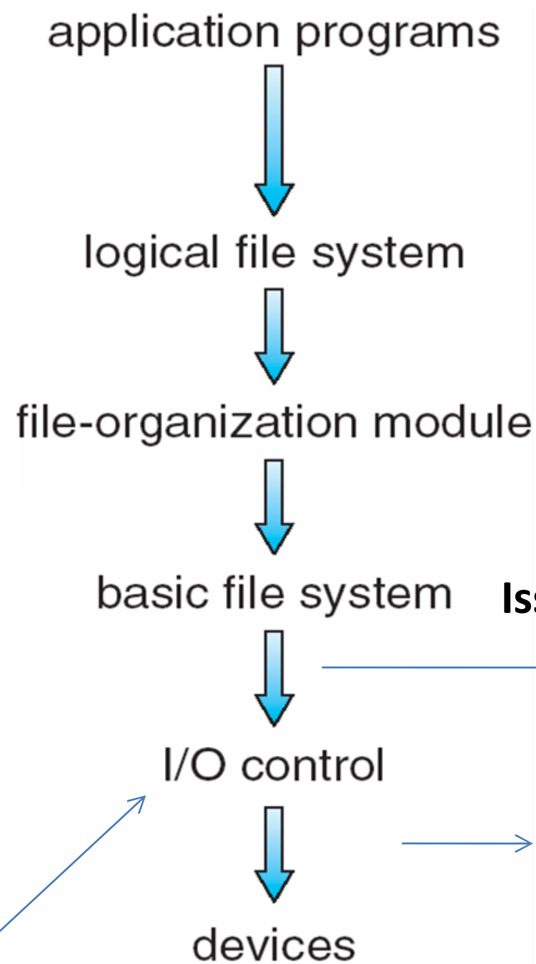
File System Structure - Layered

FILE SYSTEM STRUCTURE:

When talking about "the file system", you are making a statement about both the rules used for file access, and about the algorithms used to implement those rules. Here's a breakdown of those algorithmic pieces.

Application Programs	The code that's making a file request.
Logical File System	This is the highest level in the OS; it does protection, and security. Uses the directory structure to do name resolution.
File-organization Module	Here we read the file control block maintained in the directory so we know about files and the logical blocks where information about that file is located.
Basic File System	Knowing specific blocks to access, we can now make generic requests to the appropriate device driver.
IO Control	These are device drivers and interrupt handlers. They cause the device to transfer information between that device and CPU memory.
Devices	The disks / tapes / etc.

Layered File System



Manages **FCB**

Translates logical to physical blocks

Issue commands

R/W Physical block (cylinder, track, sector)

Hardware specific instruction

- Each level uses the feature of low level
- Create new features for higher level

Device driver, transfer information between memory/disk

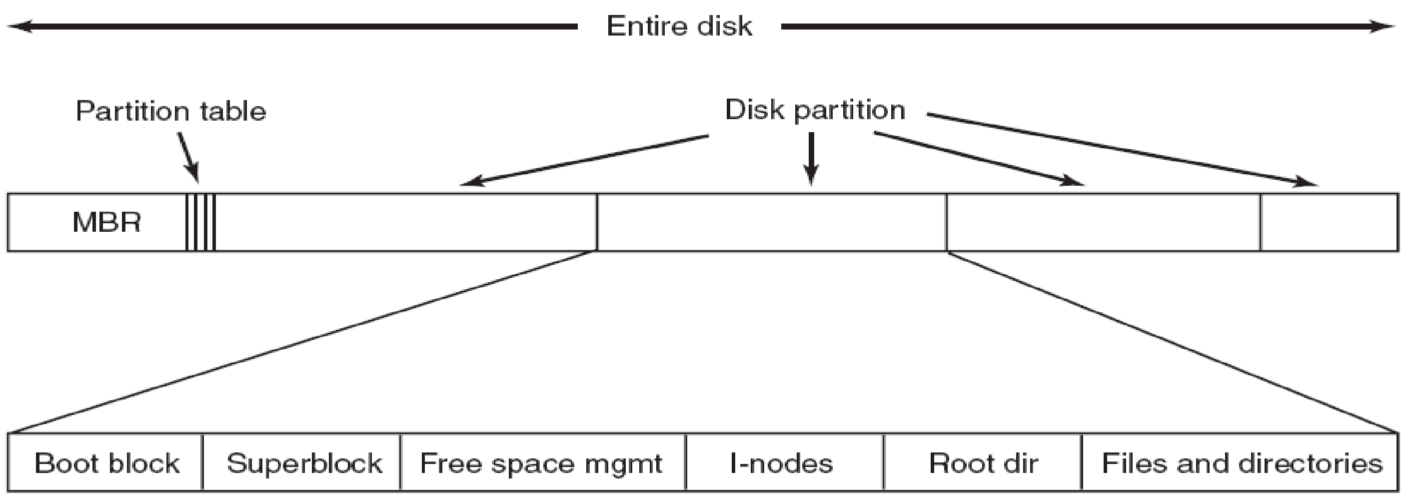
File system data structures

- On disk
- In memory

Disk Layout

- **Boot control block** contains info needed by system to boot OS from that partition
  - Needed if partition contains OS, usually first block of partition
- **Partition control block** (**superblock, master file table**) contains partition 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
- Per-file **File Control Block (FCB)** contains many details about the file
  - Inode number, permissions, size, dates
  - NTFS stores into in master file table using relational DB structures

Disk Layout


Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories

A Typical File Control Block

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

In-Memory File System Structures

- In memory directory structure holds the directory information of recently accessed directories
- **System-wide-open file** contains a copy of FCB for each file
- **Per-process open file table**: contains pointer to appropriate entry in the **system wide open file table**

File Allocation Methods

- In all allocation methods, disk is formatted into blocks.
  - As a result, it is possible to have both internal and external fragmentation in schedules that use contiguous blocks.
- Each allocation method has a different way of calculating the **Physical Address** from the **Logical Address** (**LA** in slides)

  Calculate the PA from the LA as a function of blocksize.
- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous
  - Linked List
  - Indexed
- You may refer online source for this topic
  - https://www.geeksforgeeks.org/file-allocation-methods/

Allocating Blocks to files

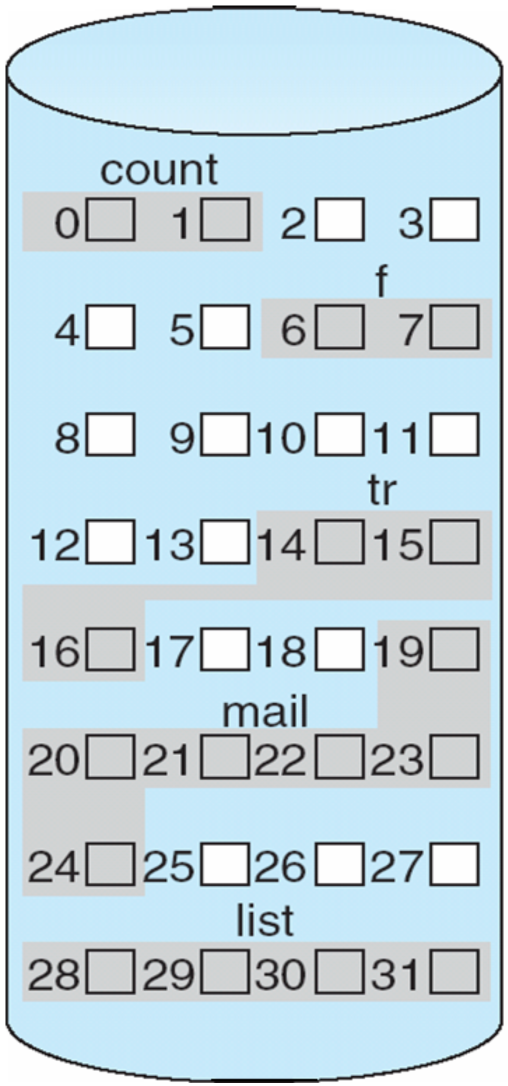
An allocation method refers to how disk blocks are allocated for files

- Most important implementation issue
- Methods
  - Contiguous allocation
  - Linked list allocation
  - Linked list using table
  - I-nodes

Allocation Methods - Contiguous

- **Contiguous allocation** – each file occupies set of contiguous blocks
- Blocks are allocated b, b+1, b+2,…….
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required (directory)
  - Easy to implement
  - Read performance is great. Only need one seek to locate the first block in the file. The rest is easy.
- Accessing file is easy
  - Minimum disk head movement
  - Sequential and direct access

Contiguous Allocation of Disk Space



directory

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

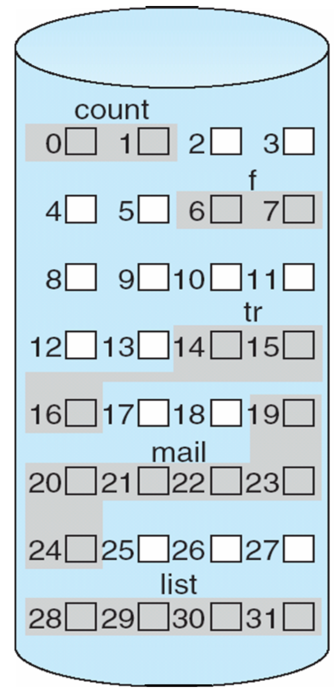
Contiguous Allocation

- Mapping from logical to physical

a) Accessing the file requires a minimum of head movement.

b) Easy to calculate block location: block i of a file, starting at disk address **b**, is **b + i.**

c) Difficulty is in finding the contiguous space, especially for a large file. Problem is one of dynamic allocation (**first fit, best fit, etc.)** which has external fragmentation. If many files are created/deleted, compaction will be necessary.



directory

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

LA/512 → Q, R

- It's hard to estimate at create time what the size of the file will ultimately be. What happens when we want to extend the file --- we must either terminate the owner of the file, or try to find a bigger hole.
- Notice the greyed areas, 0-1, 6-7, 14-16, 19-24, 28-31. Each group is a file.

Block to be accessed = Q + starting address
Displacement into block = R

Contiguous Allocation cont..

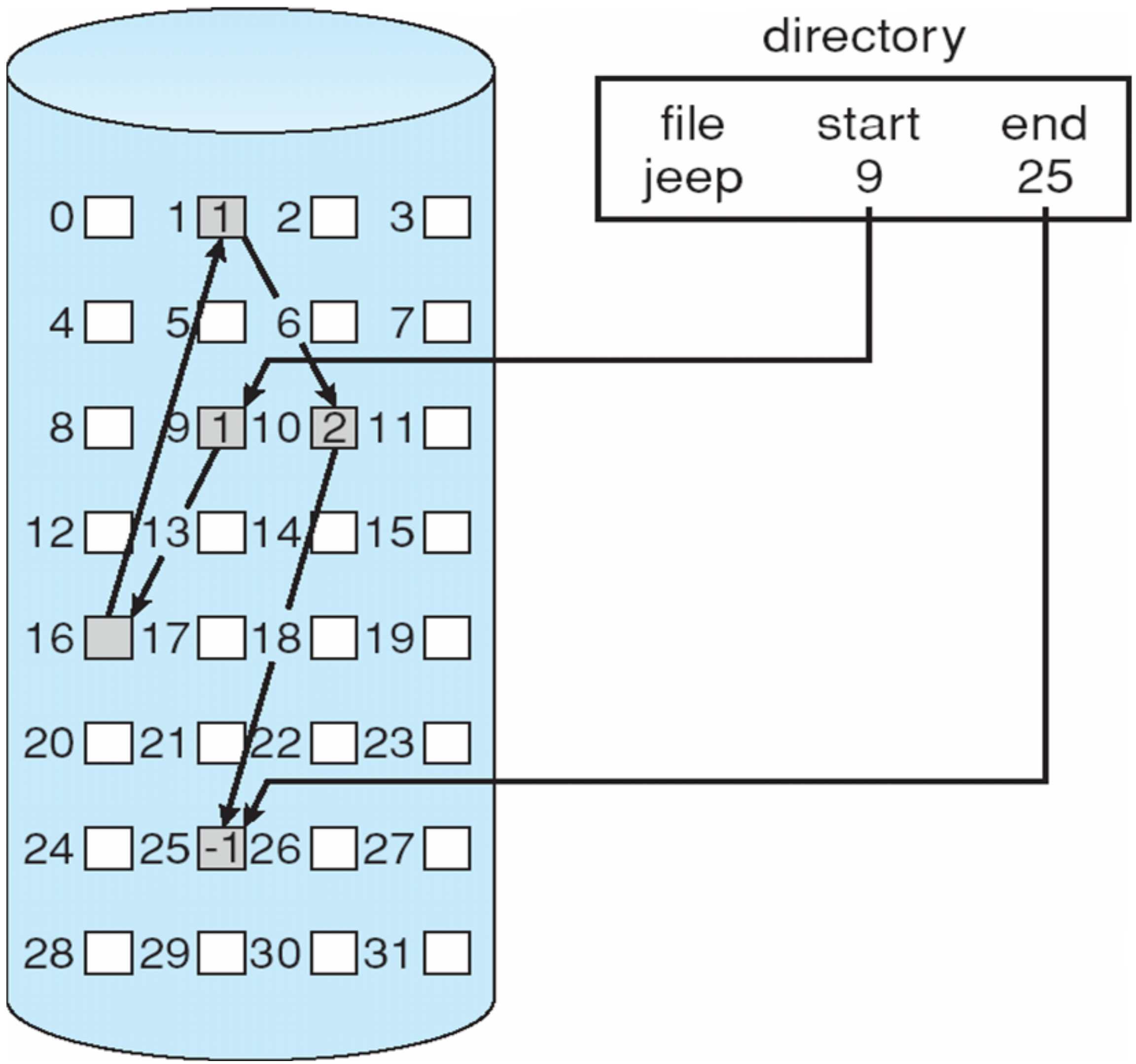
- Problems
- – Finding space for file
  - Satisfy the request of size n from the list of holes
  - External fragmentation
    - – Need for **compaction routine**
    - – **off-line** (**downtime**) or **on-line**
- – Do not know the file size a priori
  - Terminate and restart
  - Overestimate
  - Copy it in a larger hole
  - Allocate new contiguous space (Extent)

Allocation Methods - Linked

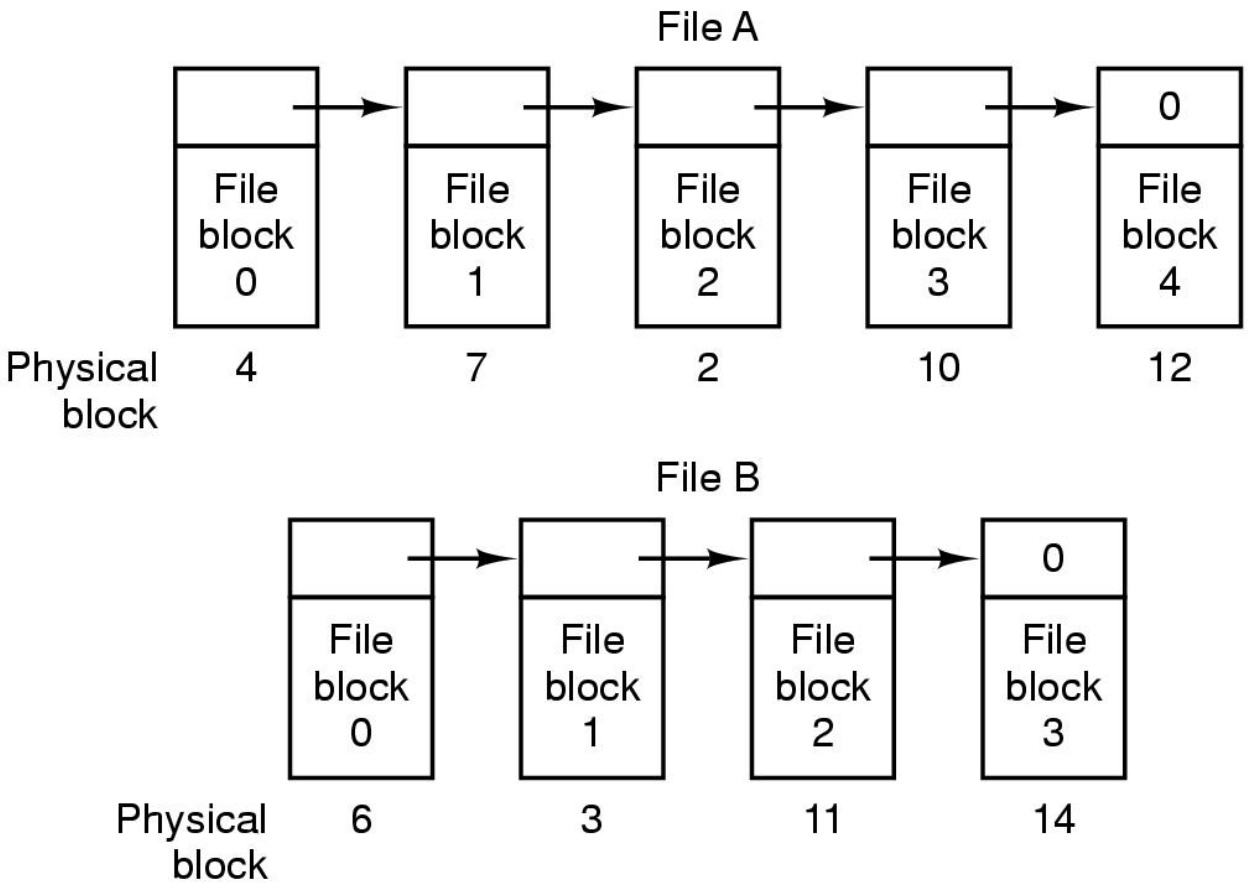
- **Linked allocation** – each file a linked list of blocks
- Advantages
  - No external fragmentation, No compaction needed.
  - Free space management system called to allocate new block
    - Internal fragmentation still a problem.
  - Improve efficiency by clustering blocks into groups. As disk fills, some files will be allocated on distant blocks.
- Disadvantages
  - Each block contains pointer to next block, ends in nil pointer.
  - Reliability can be a problem because pointers can become corrupted.
    - lose pointer a in the middle = a large portion of the file lost.
  - Locating a block can take many I/Os and disk seeks

    Supports only sequential files well

Linked Allocation



Linked List Allocation



Storing a file as a linked list of disk blocks.

Linked Allocation

- Free blocks are arranged from the free space management
- No external fragmentation
- Files can continue to grow

Disadvantage

1. Effective only for sequential access
   Random/direct access (i-th block) is difficult

2. Space wastage
If block size 512 B
Disk address 4B
Effective size 508B

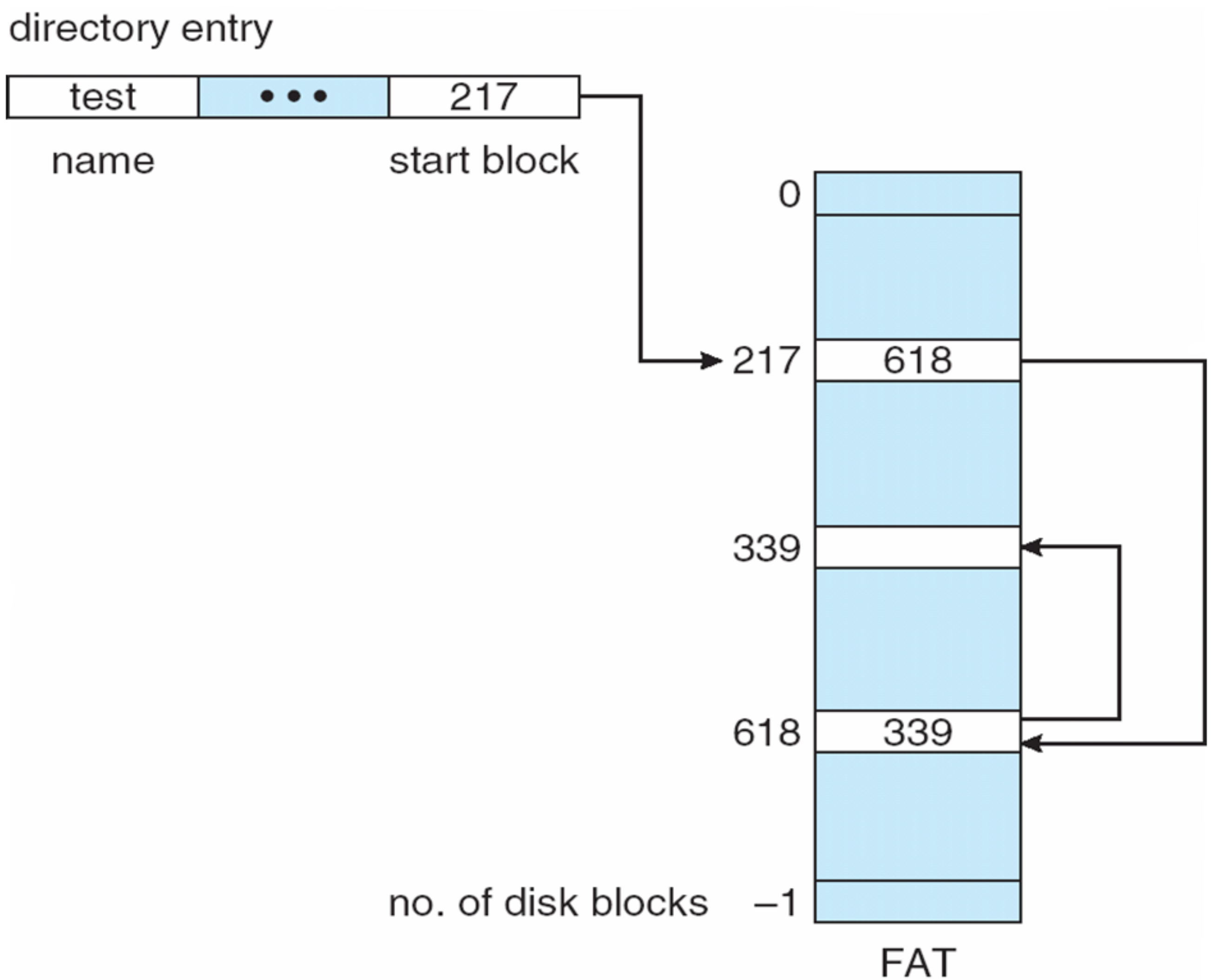
3. Reliability
Lost/damaged pointer
Bug in the OS software and disk hardware failure

4. Poor performance

Solution: Clusters

- **Improves disk access time (head movement)**
- **Decreases the link space needed for block**
- **Internal fragmentation**

File-Allocation Table



Linked Allocation (con’t)

LINKED ALLOCATION

Pointers use up space in each block. Reliability is not high because any loss of a pointer loses the rest of the file.

A File Allocation Table is a variation of this.

It uses a separate disk area to hold the links.

This method doesn't use space in data blocks. Many pointers may remain in memory.

A FAT file system is used by MS-DOS.

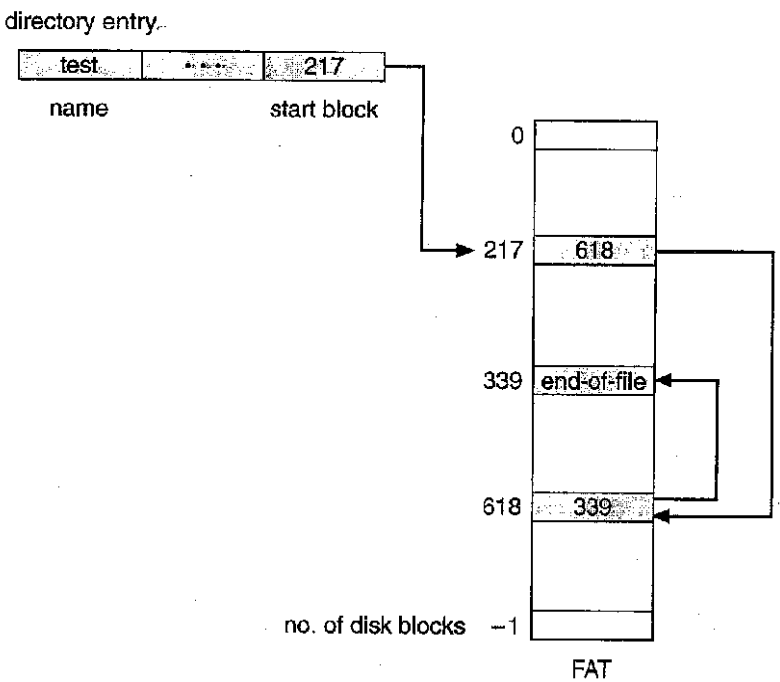
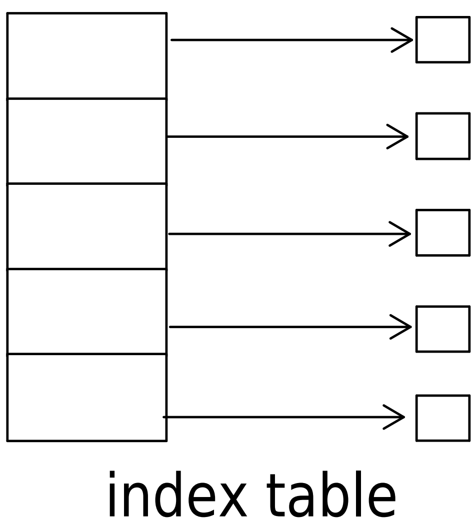


Figure 11.5 File-allocation table.

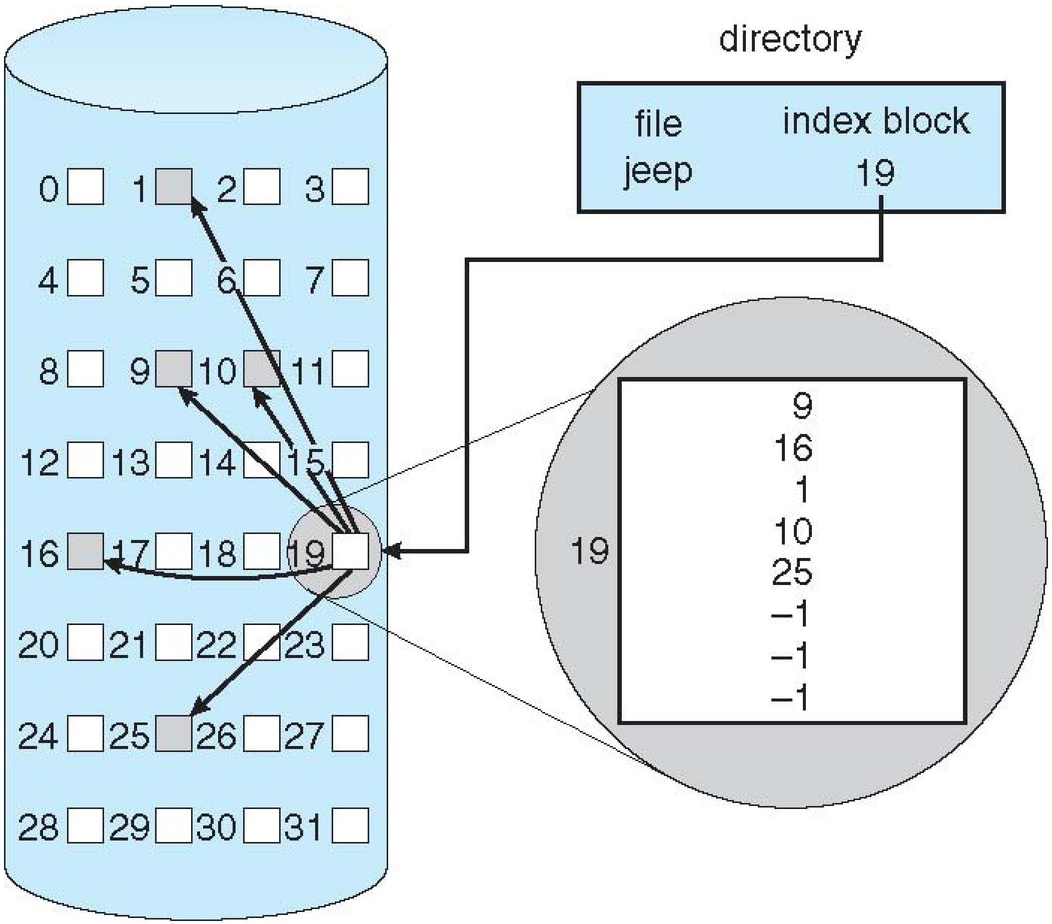
Allocation Methods - Indexed

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

- Logical view



Example of Indexed Allocation



Indexed Allocation contd..

- Efficient random access without external fragmentation,
- Size of index block
  - One data block
- Overhead of index block
  - Wastage of space
  - Small sized files

Read Performance

PERFORMANCE ISSUES FOR THESE STORAGE METHODS

It's difficult to compare mechanisms because usage is different. Let's calculate, for each method, the number of disk accesses to read block i from a file:

contiguous: **1** access from location **start + i.**

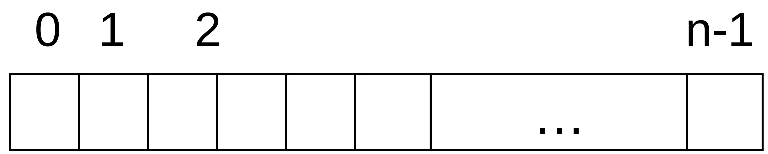
linked: **i + 1** accesses, reading each block in turn. (is this a fair example?)

index: **2** accesses, 1 for index, 1 for data.

multilevel: **3** accesses, 2 for index, 1 for data.

Free-Space Management

- File system maintains **free-space list** to track available blocks
- **Bit vector** or **bit map** (*n* blocks)
- Each block is represent by 1 bit



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

Simple and Efficient to find first free blocks or n consecutive free blocks

Free Space Management

We need a way to keep track of space currently free. This information is needed when we want to create or add (allocate) to a file. When a file is deleted, we need to show what space is freed up.

BIT VECTOR METHOD

- Each block is represented by a bit

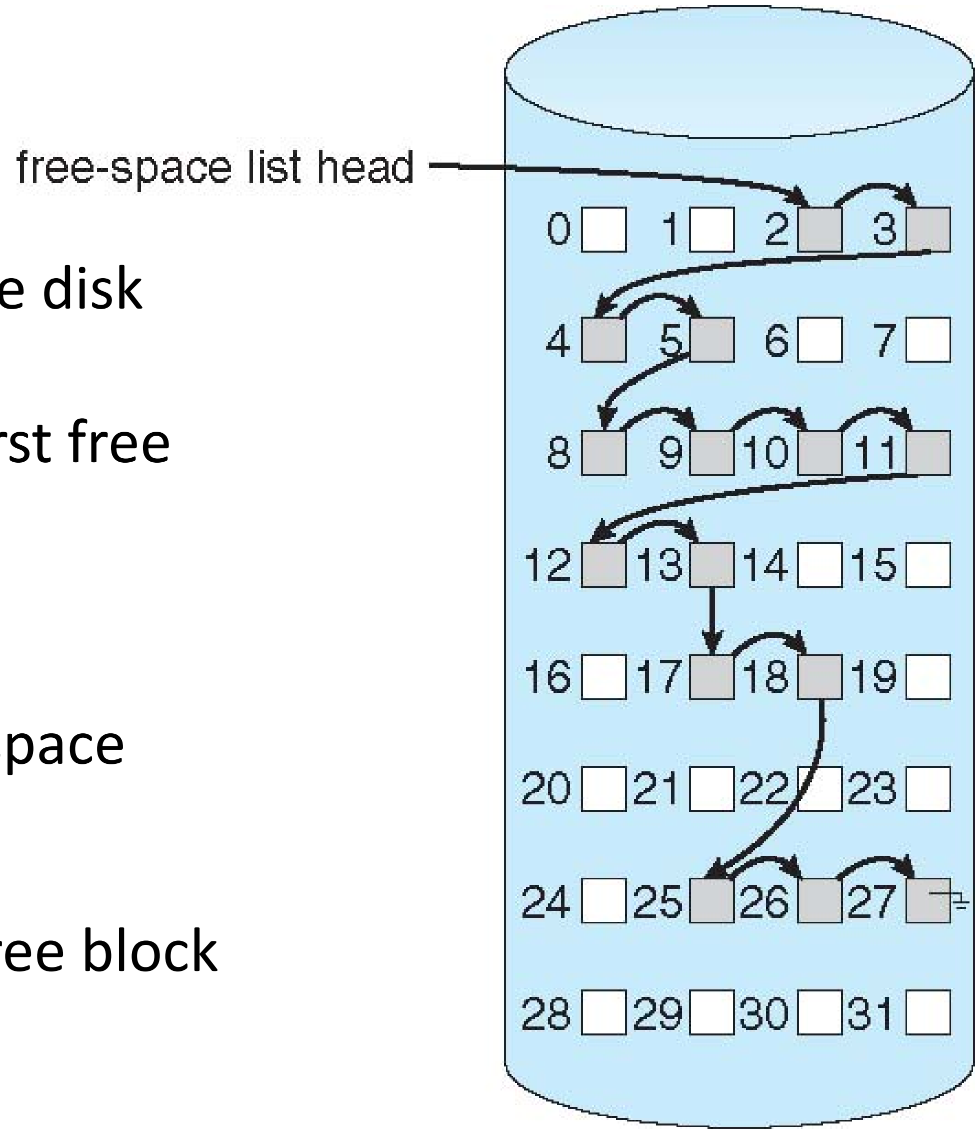
  1 1 0 0 1 1 0 means blocks 2, 3, 6 are free.

- This method allows an easy way of finding contiguous free blocks. Requires the overhead of disk space to hold the bitmap.
- A block is not REALLY allocated on the disk unless the bitmap is updated.
- What operations (disk requests) are required to create and allocate a file using this implementation?

Linked Free Space List on Disk

- Link together all the free disk blocks
- Keep a pointer to the first free block

- Cannot get contiguous space easily
  - Traverse the list
- Generally require first free block



Free Space Management

FREE LIST METHOD

- Free blocks are chained together, each holding a pointer to the next one free.
  - Link List
- This is very inefficient since a disk access is required to look at each sector.

GROUPING METHOD

- In one free block, put lots of pointers to other free blocks. Include a pointer to the next block of pointers. Linked indexed list.

COUNTING METHOD

- Since many free blocks are contiguous, keep a list of dyads holding the starting address of a "chunk", and the number of blocks in that chunk.

- Format < disk address, number of free blocks >

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

- Mode of access: read, write, execute
- Three classes of users

	RWX			
a) **owner access**		7	$\Rightarrow$	1 1 1
	RWX			
b) **group access**		6	$\Rightarrow$	1 1 0
	RWX			
c) **public access**		1	$\Rightarrow$	0 0 1

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

owner group public

chmod 761 game

Attach a group to a file

chgrp G game

Thank You!

Implementing Directories

- OS uses path name supplied by the user to locate the directory entry
- Stores attributes
- Directory entry specifies block addresses by providing
  - Number of first block (contiguous)
  - Number of first block (linked)
  - Number of i-node

Directory Implementation

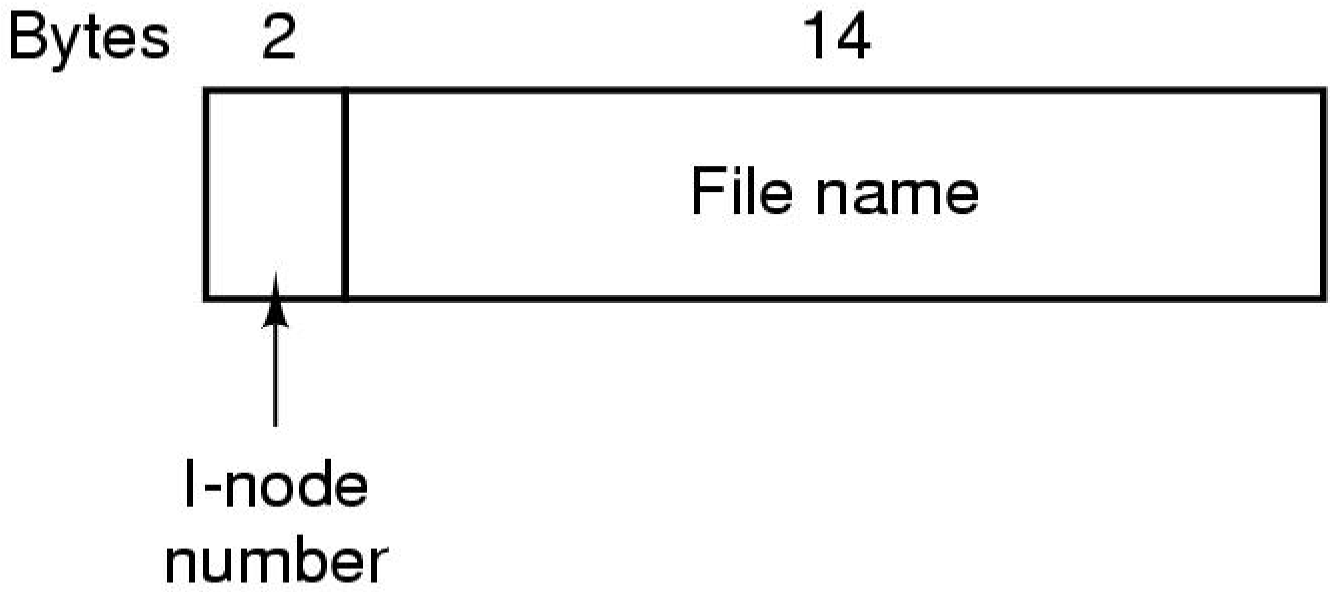
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
  - New file creation / deletion

- Cache the frequently accessed entry
- Binary search to speedup directory search
  - Could keep ordered alphabetically via linked list
  - or use B+ tree

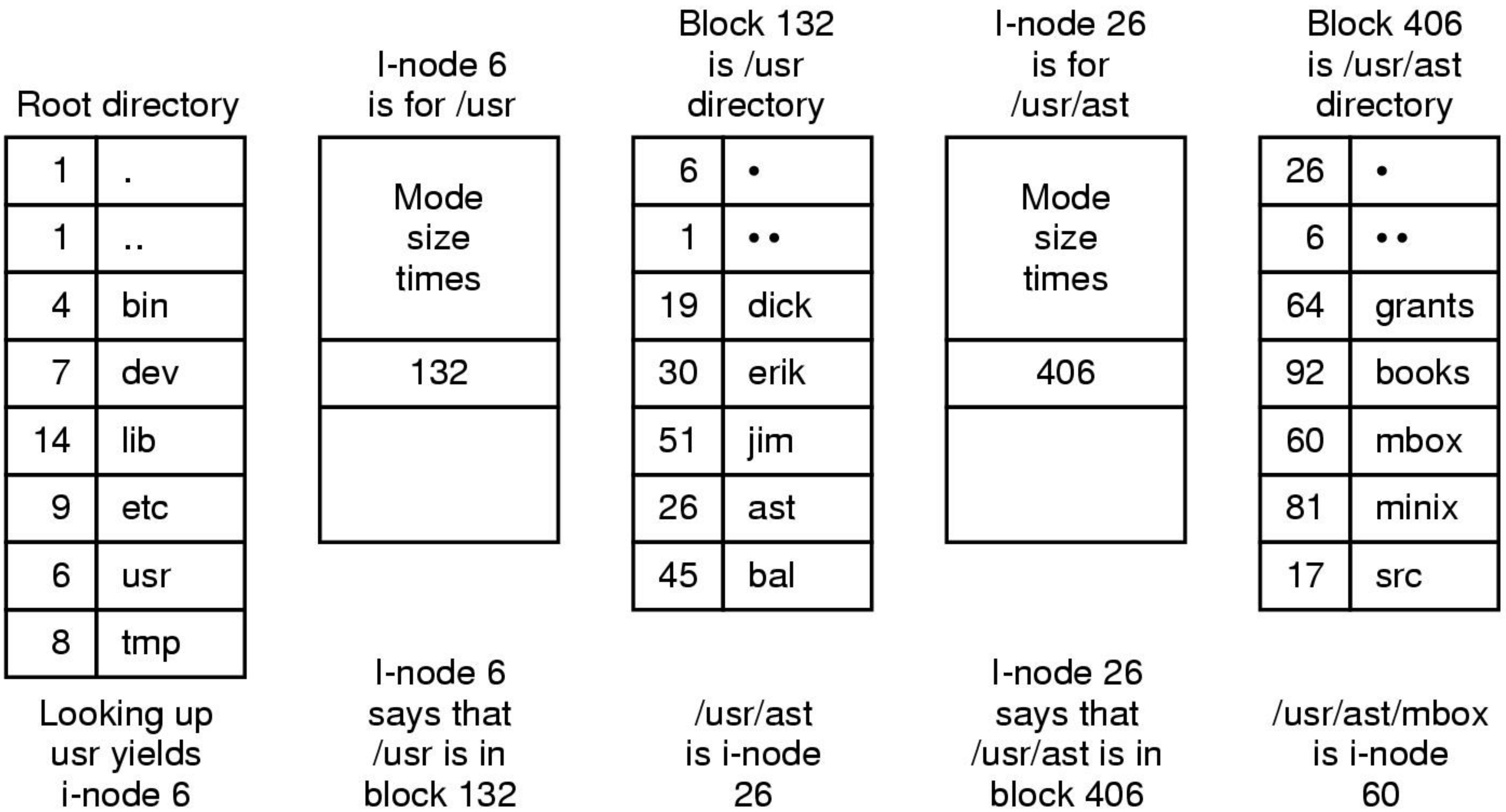
Directory Implementation

- **Hash Table** – hash data structure
  - Hash value computed from filename
  - Decreases directory search time
  - Insertion and deletion simple
  - **Collisions** – situations where two file names hash to the same location
    - Chaining
- Hash table of fixed size
  - Only good if entries are fixed size (CD-ROM)
- Performance depends on hash function

The UNIX File System

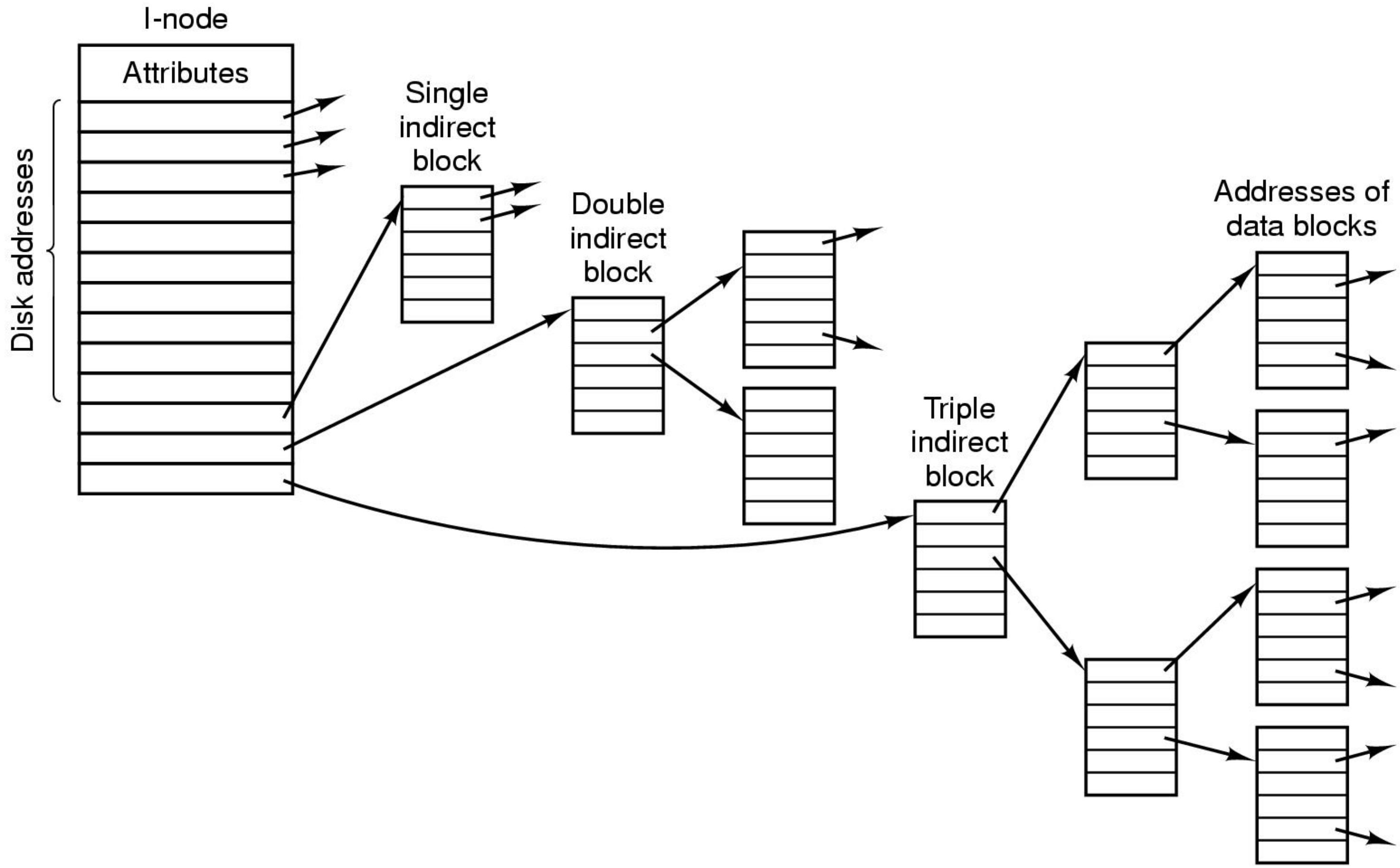


The UNIX File System



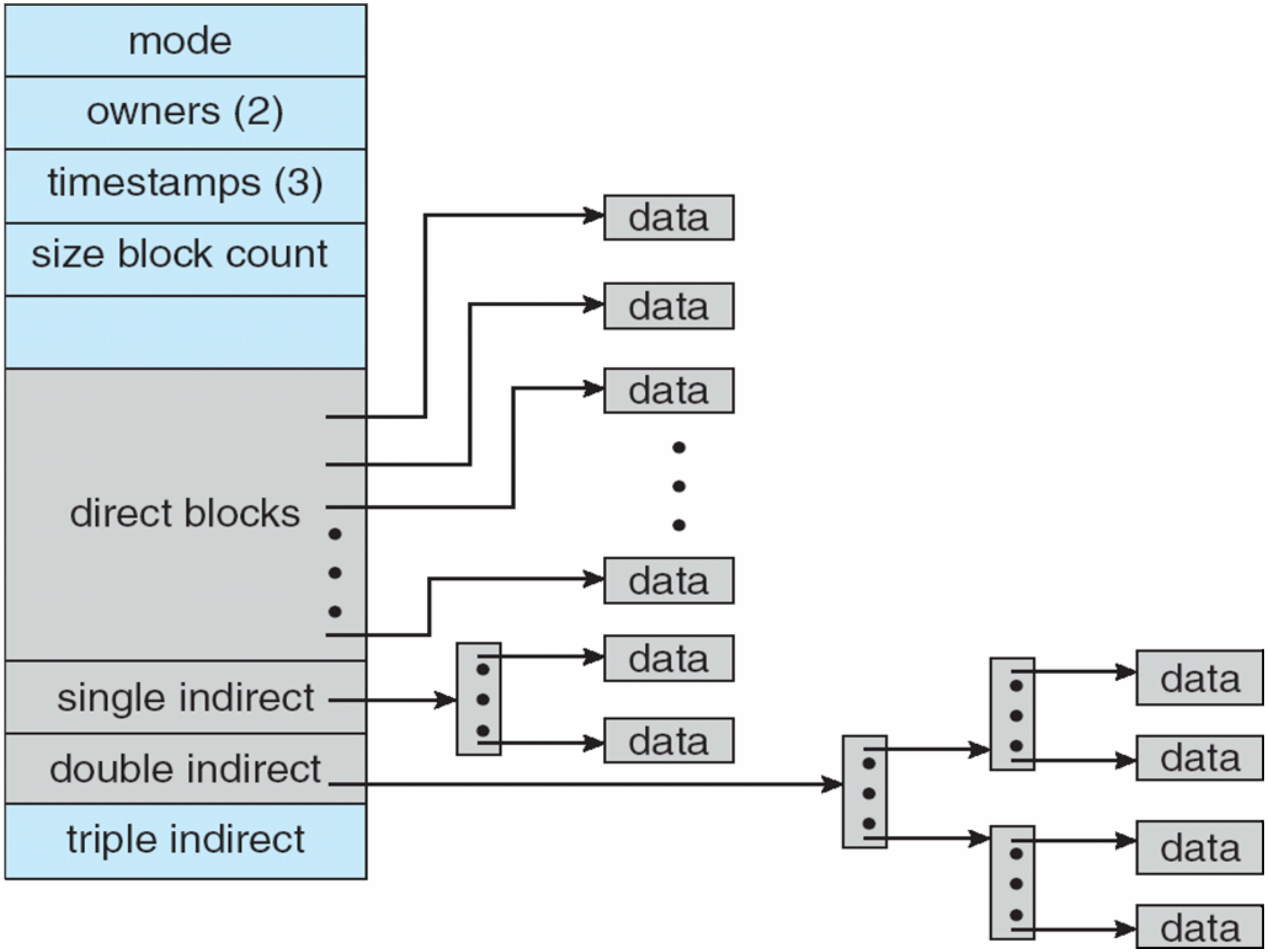
The steps in looking up */usr/ast/mbox*.

The UNIX File System



A UNIX i-node.

Combined Scheme: UNIX UFS
(4K bytes per block, 32-bit addresses)



Virtual File Systems

- **Virtual File Systems** (**VFS**) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    - Implements **vnodes** which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems (Cont.)

- The API is to the VFS interface, rather than any specific type of file system