Chapter 10: File-System Interface

Chapter 10: File-System Interface

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- Data can NOT be written to secondary storage unless they are within a file.

File Structure

- A file has a certain defined **structure** which depends on its types:
  - A text file is a sequence of characters organized into lines.
  - A source file is a sequence of subroutines and function.
  - An object file is a sequence of bytes organized into blocks understandable by the system's linker.
  - An executable file is a series of code sections that the loader can bring into memory and execute.

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

File Operations

- File is an **abstract data type**
- **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

Open Files

- Several pieces of data are needed to manage open files:
  - File pointer: pointer to last read/write location, per process that has the file open
  - File-open count: the counter tracks the number of opens and closes, and reaches zero on the last close. The system can then remove the entry.
  - Disk location of the file: the info needed to locate the file on disk.
  - Access rights: per-process access mode information so OS can allow or deny subsequent I/O request

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

- **Sequential Access**

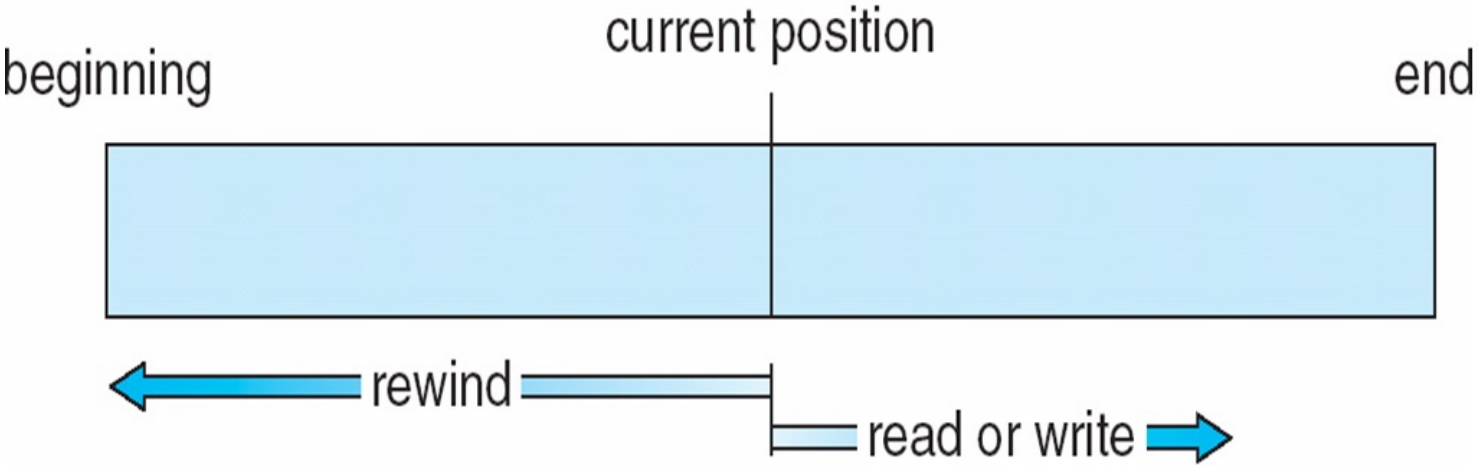
  read next
  write next
  reset
  no read after last write
  (rewrite)

- **Direct Access**

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

n = relative block number

Sequential-access File



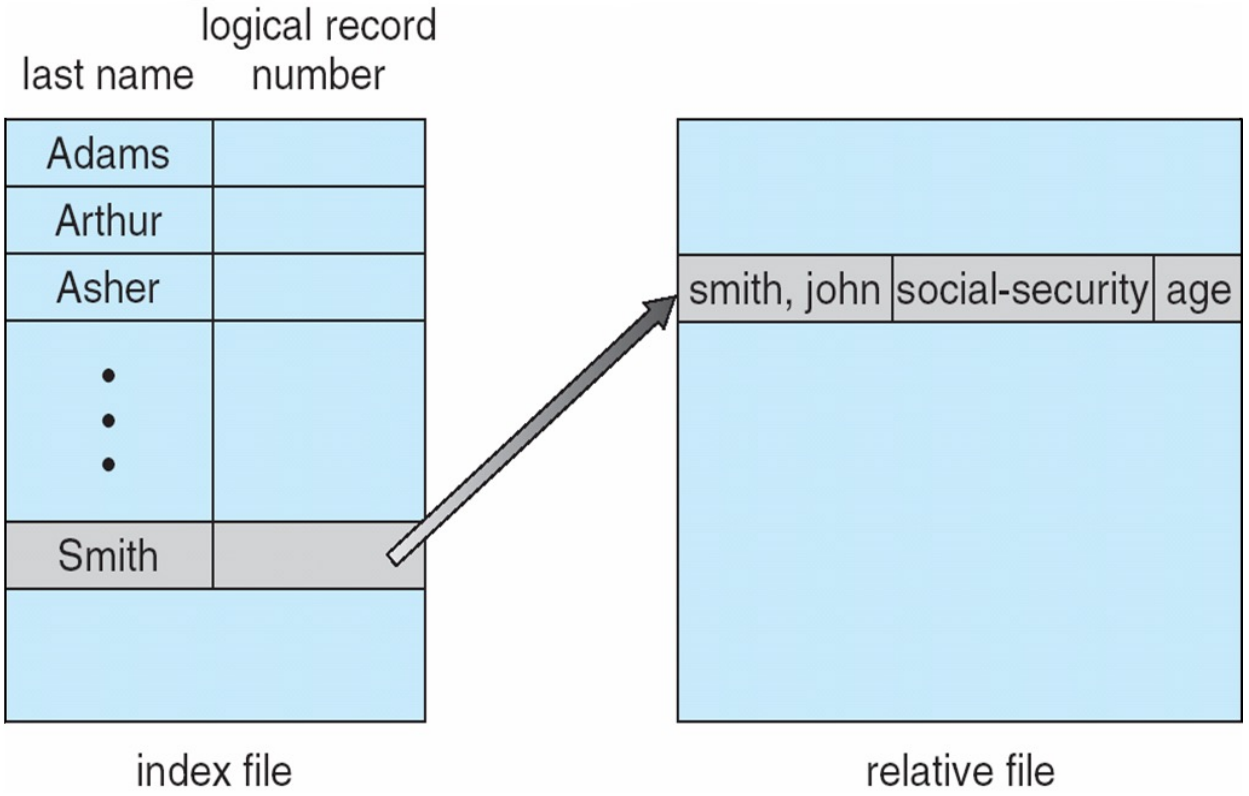
Information in the file is processed in order, one after the other.

Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

cp: current position

Example of Index and Relative Files

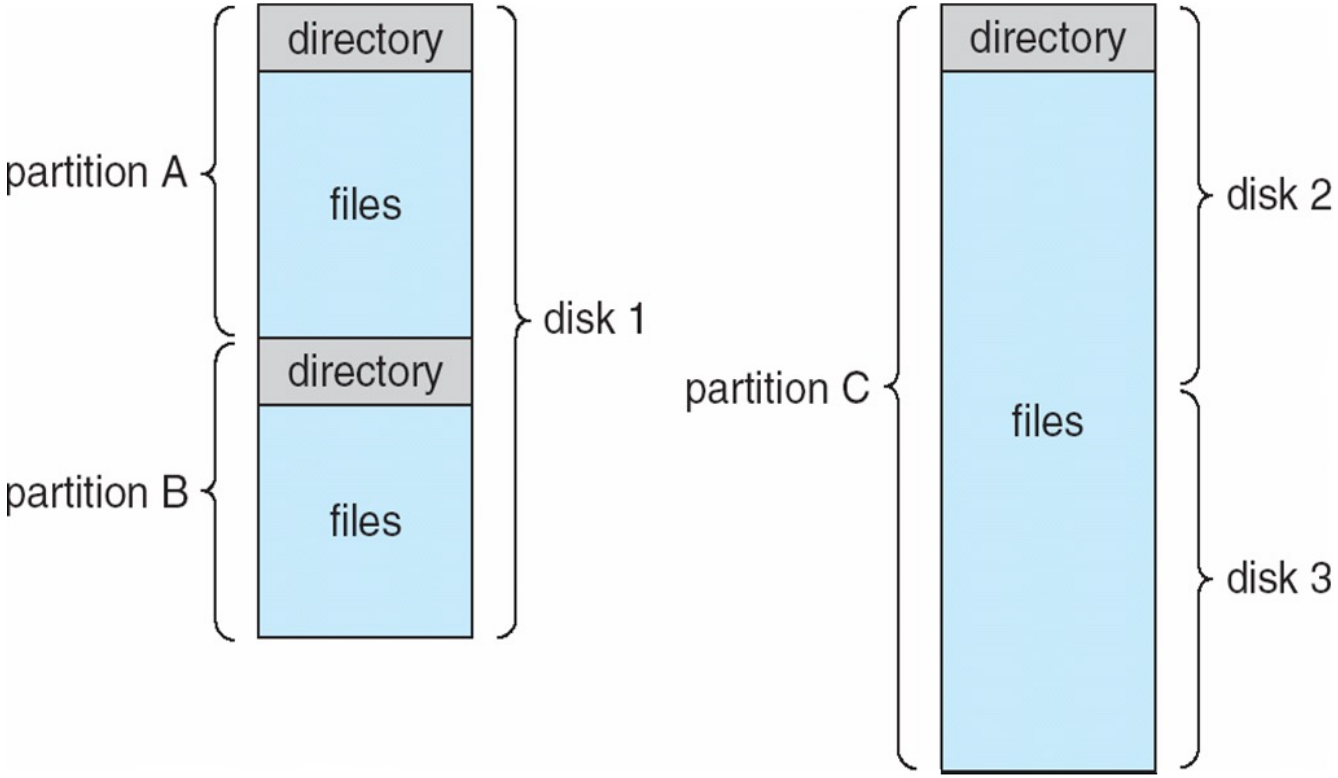


The **index** contains pointers to the various blocks. To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

Disk Structure

- Disk can be subdivided into partitions
- Disks or partitions can be redundant arrays of independent disks (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 known as a volume
- 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

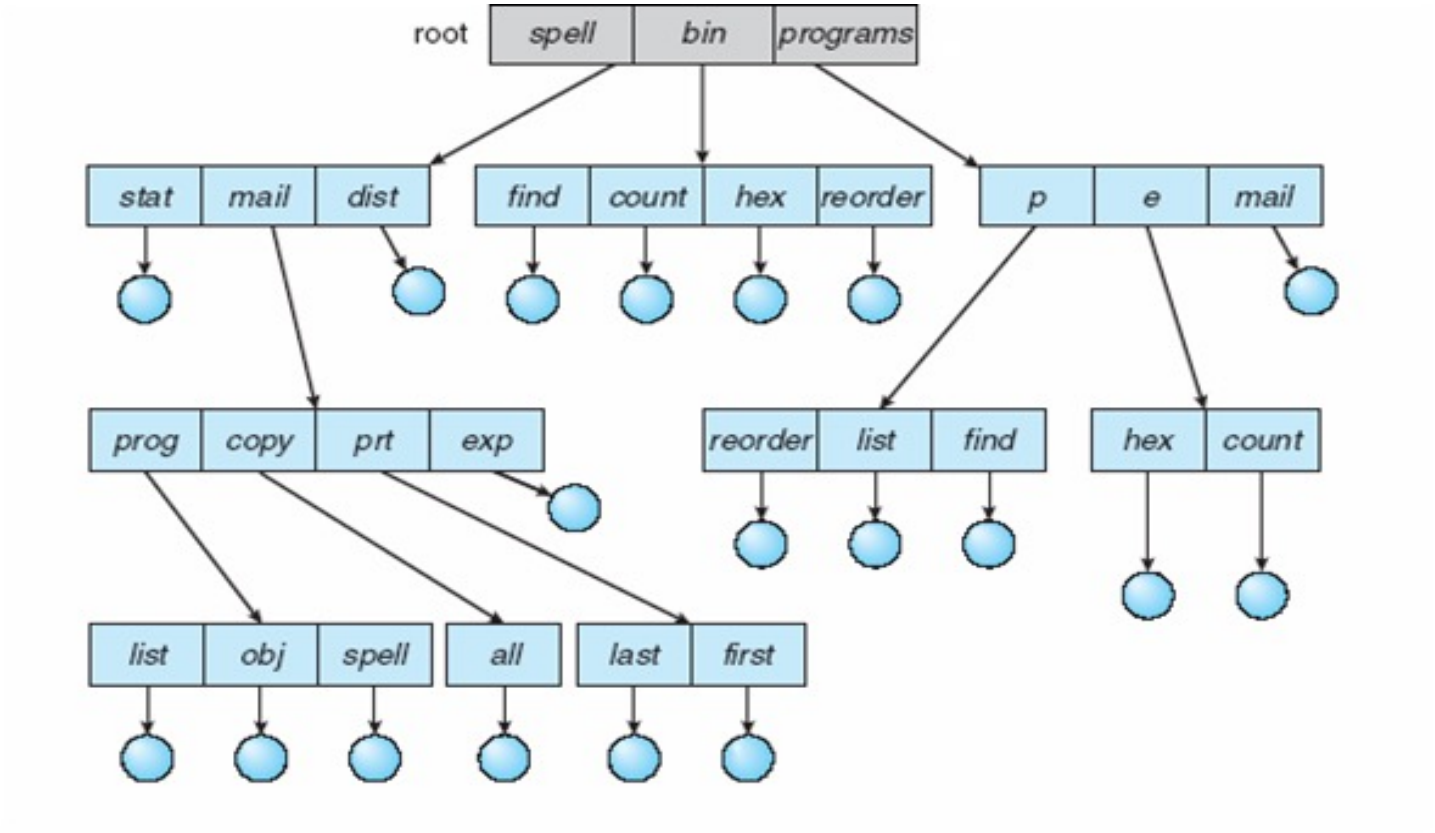


Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a device directory which records name, location, size and type.

Operations Performed on Directory

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

Tree-Structured Directories



- Absolute path: begins at the root and follows a path down to the specified file. root/spell/mail/prt/first
- Relative path: defines a path from the current directory. prt/first given root/spell/mail as current path.

Tree-Structured Directories (Cont)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - pwd
  - cd /spell/mail/prog

Tree-Structured Directories (Cont)

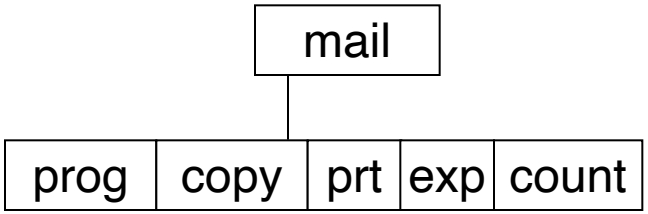
- Creating a new file is done in current directory
- Delete a file

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

  mkdir <dir-name>

  Example: if in current directory /mail

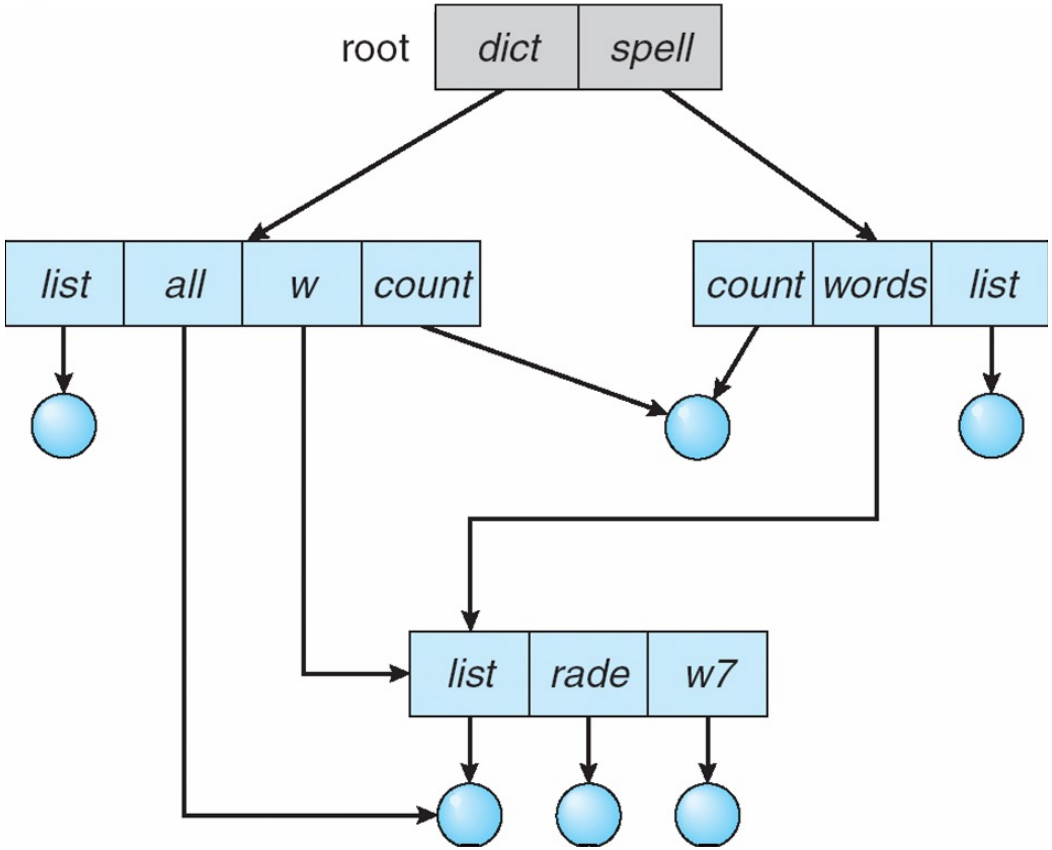
  mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"
Option1: do not delete a directory unless it is empty, such as MS-DOS
Option2: delete all files in that directory, such as UNIX rm command with r option

Acyclic-Graph Directories

- Have shared subdirectories and files
- Only one file exists. Any changes made by one person are immediately visible to the other.



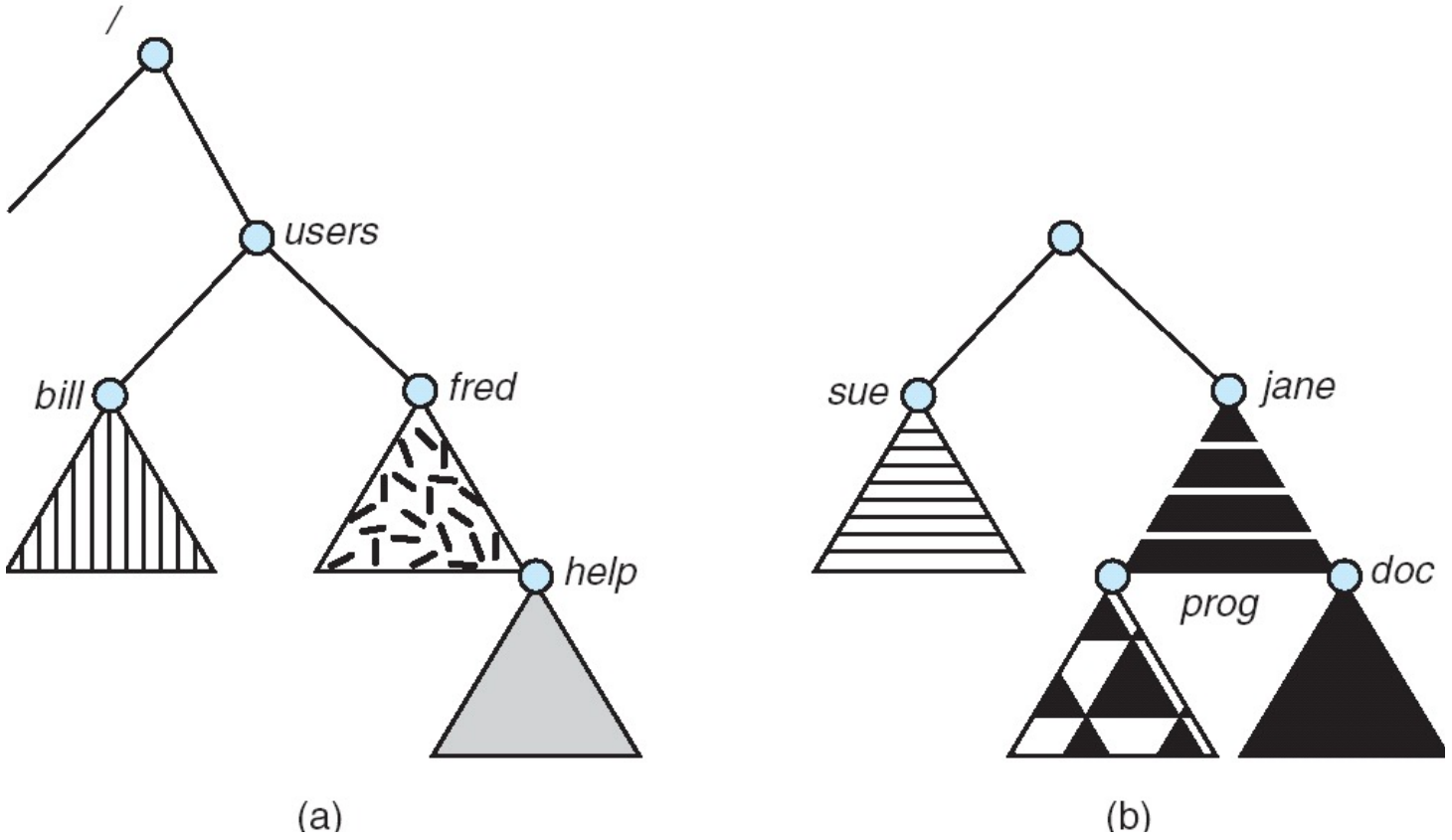
Acyclic-Graph Directories (Cont.)

- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

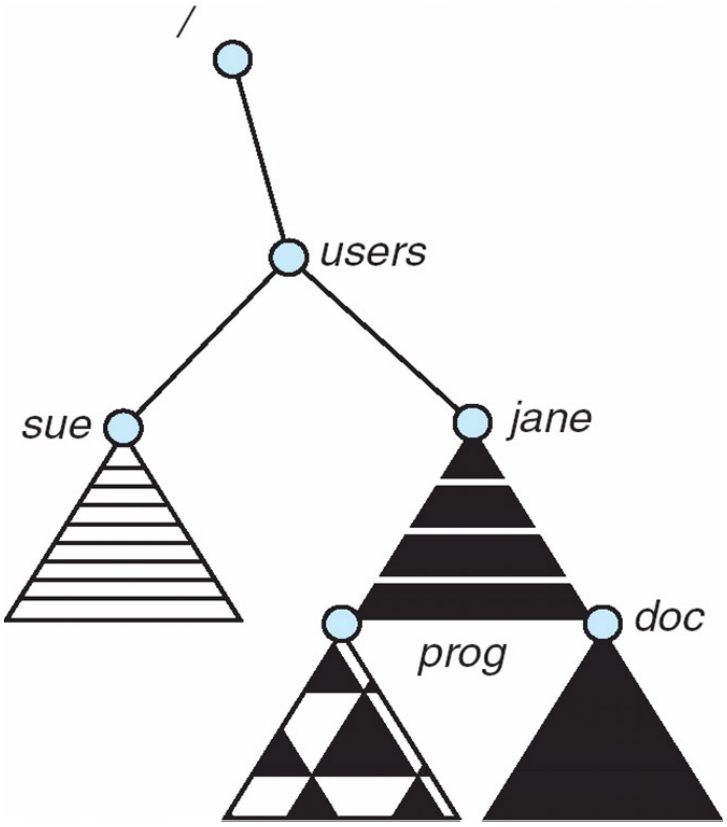
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file systemis mounted at a **mount point,** the location within the file structure where the file system is to be attached. An empty directory.

(a) Existing. (b) Unmounted Partition



Mount Point



Chapter 11: File System Implementation

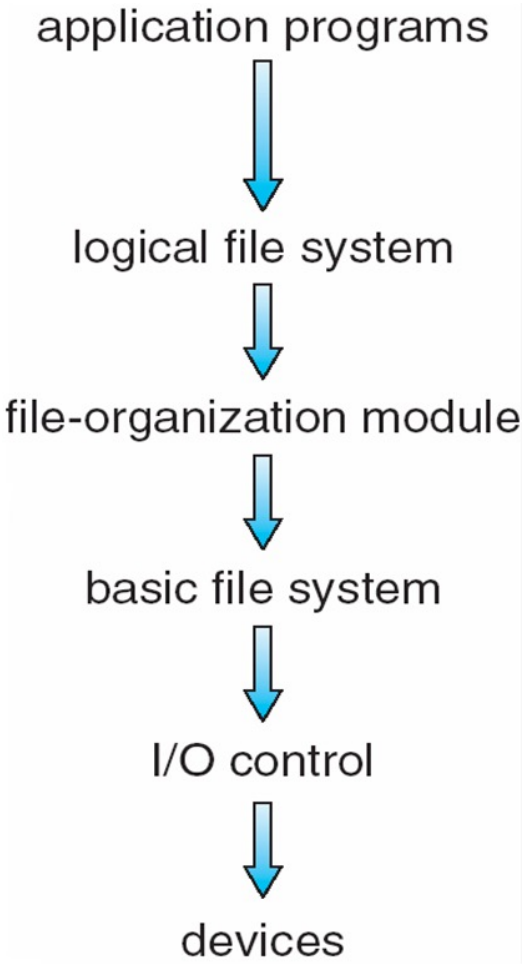
Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system organized into layers
- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- File control block – storage structure consisting of information about a file
- Device driver controls the physical device

Layered File System



File-System Implementation

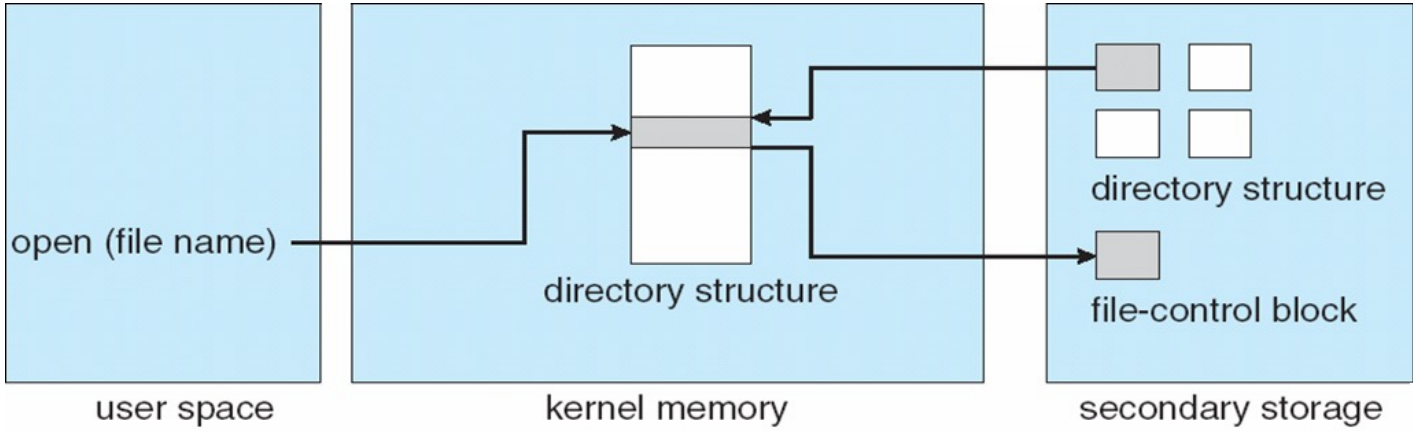
- Boot control block contains info needed by system to boot OS from that volume
- Volume control block contains volume details
- Directory structure organizes the files
- Per-file File Control Block (FCB) contains many details about the file

A Typical File Control Block

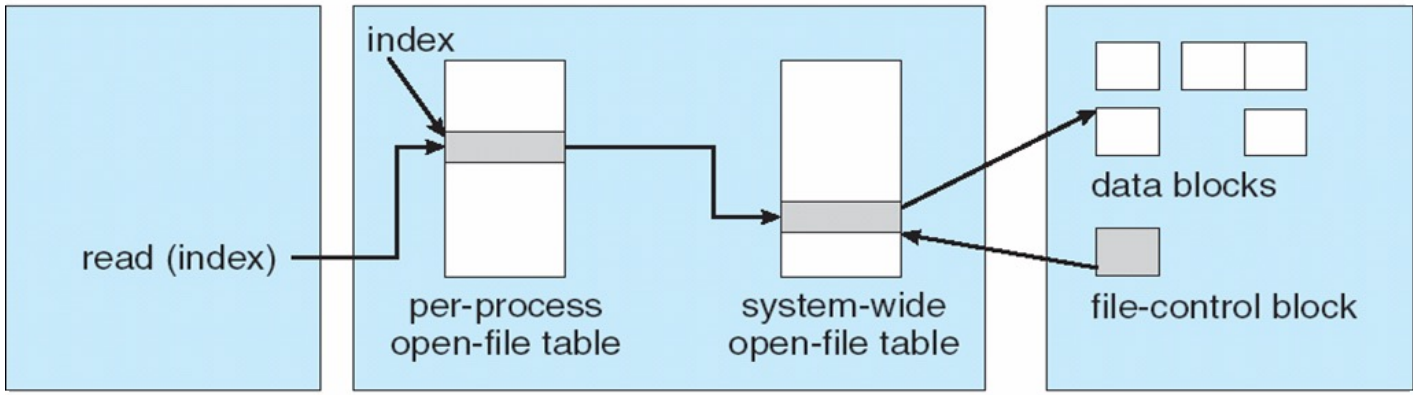
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

the necessary file system structures provided by the OS



(a) refers to opening a file



(b) refers to reading a file

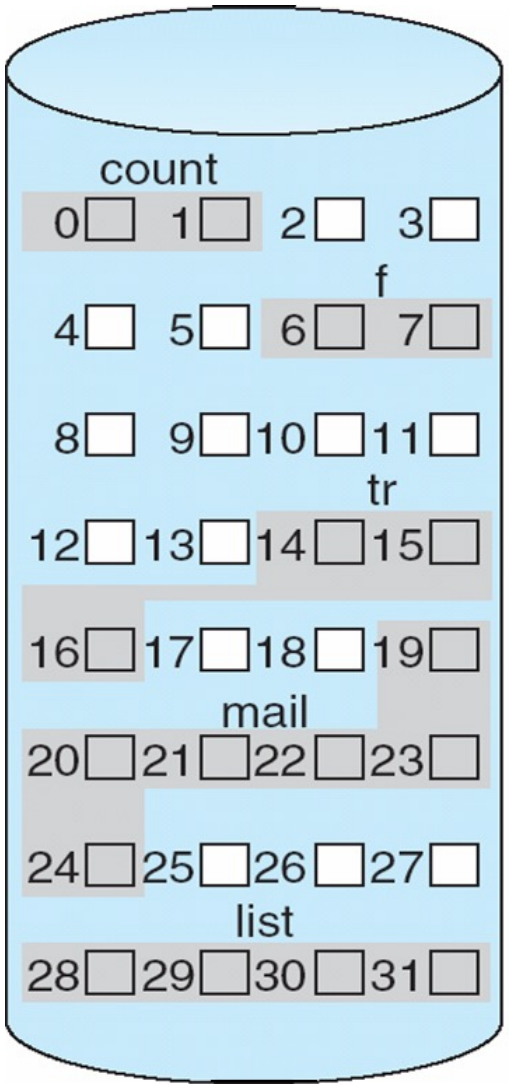
Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation of Disk Space



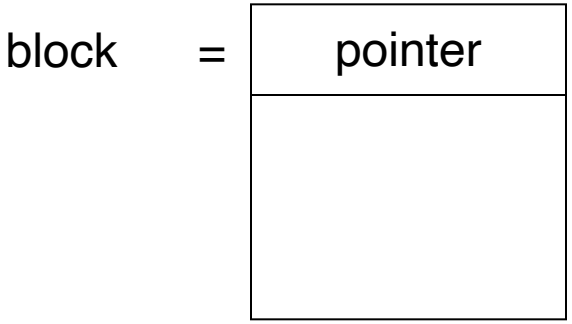
directory

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

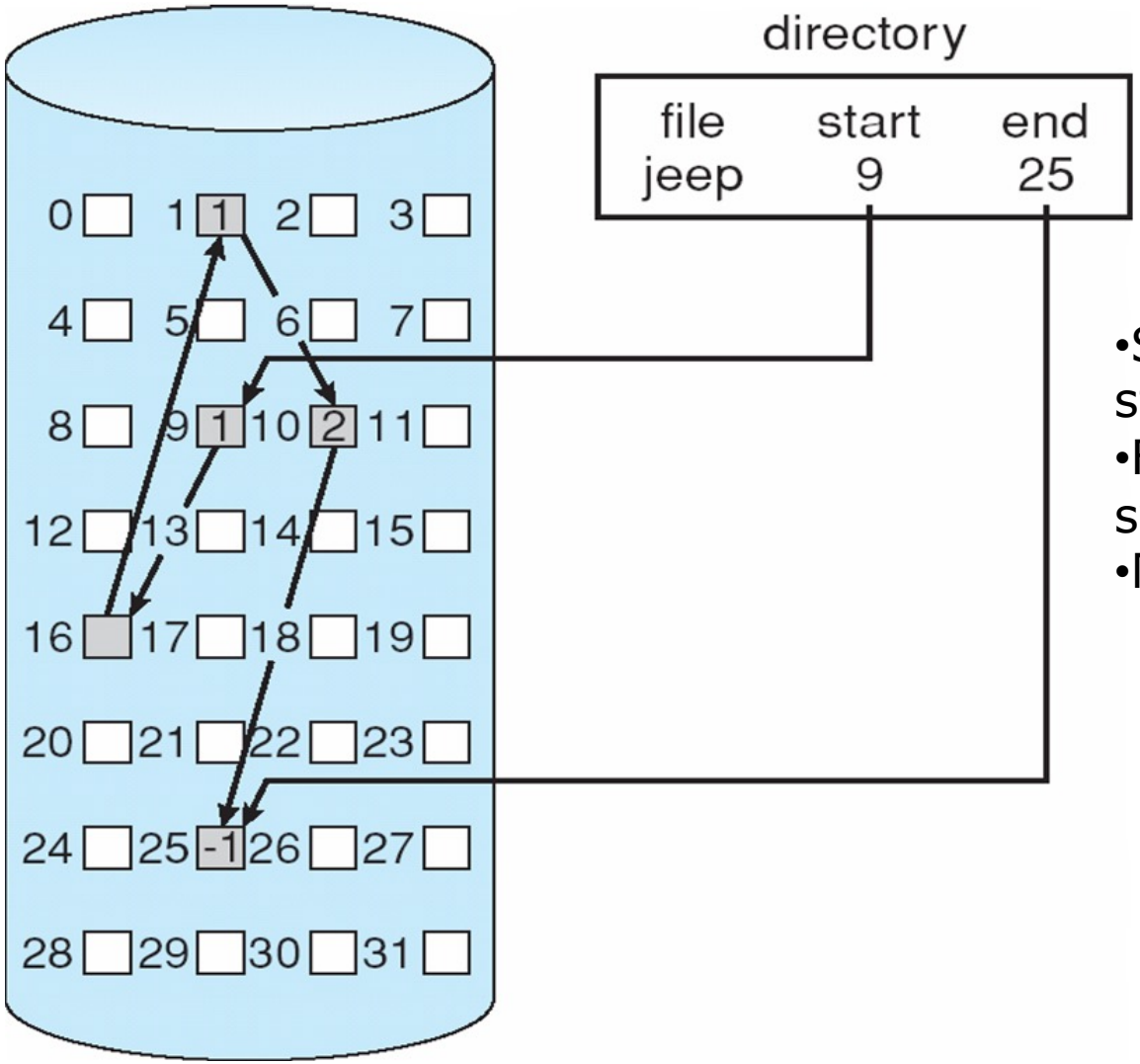
- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Linked Allocation

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

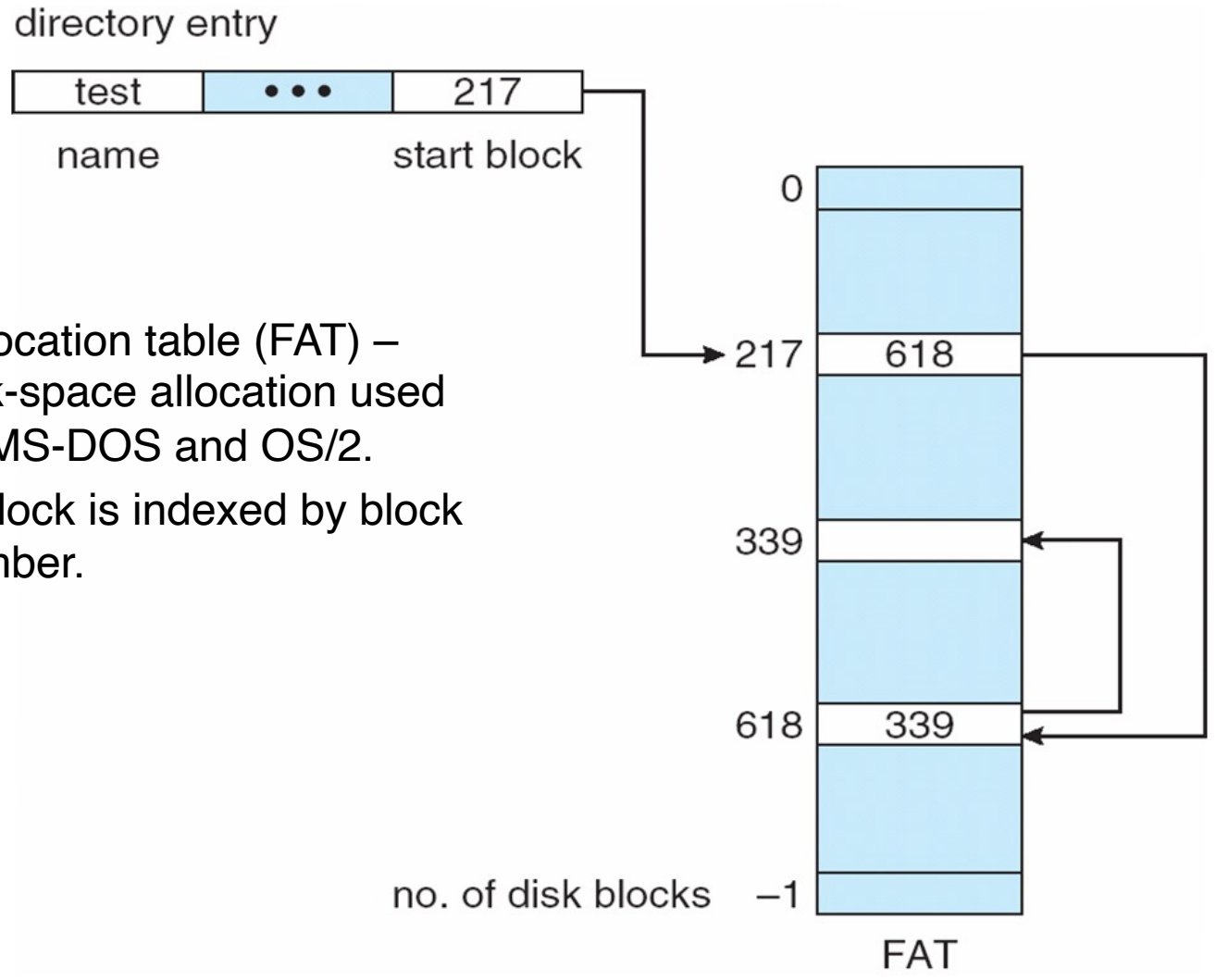


Linked Allocation



- Simple – need only starting address
- Free-space management system – no waste of space
- No random access

File-Allocation Table

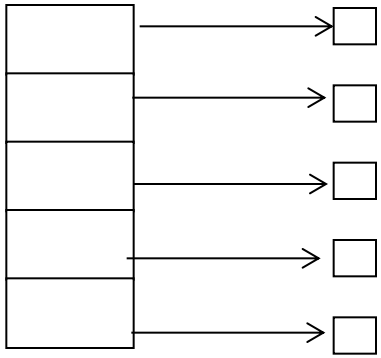


File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Each block is indexed by block number.

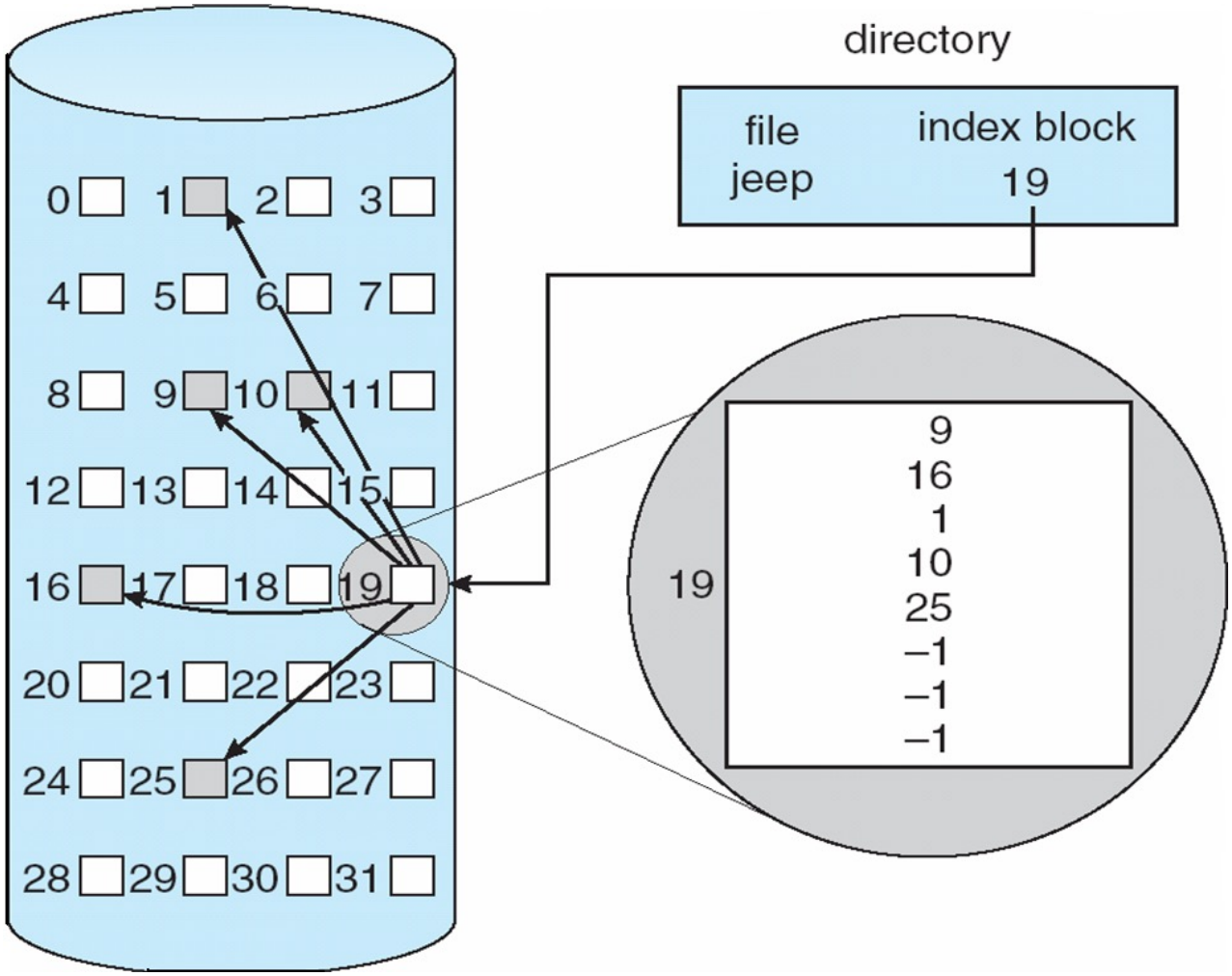
Indexed Allocation

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



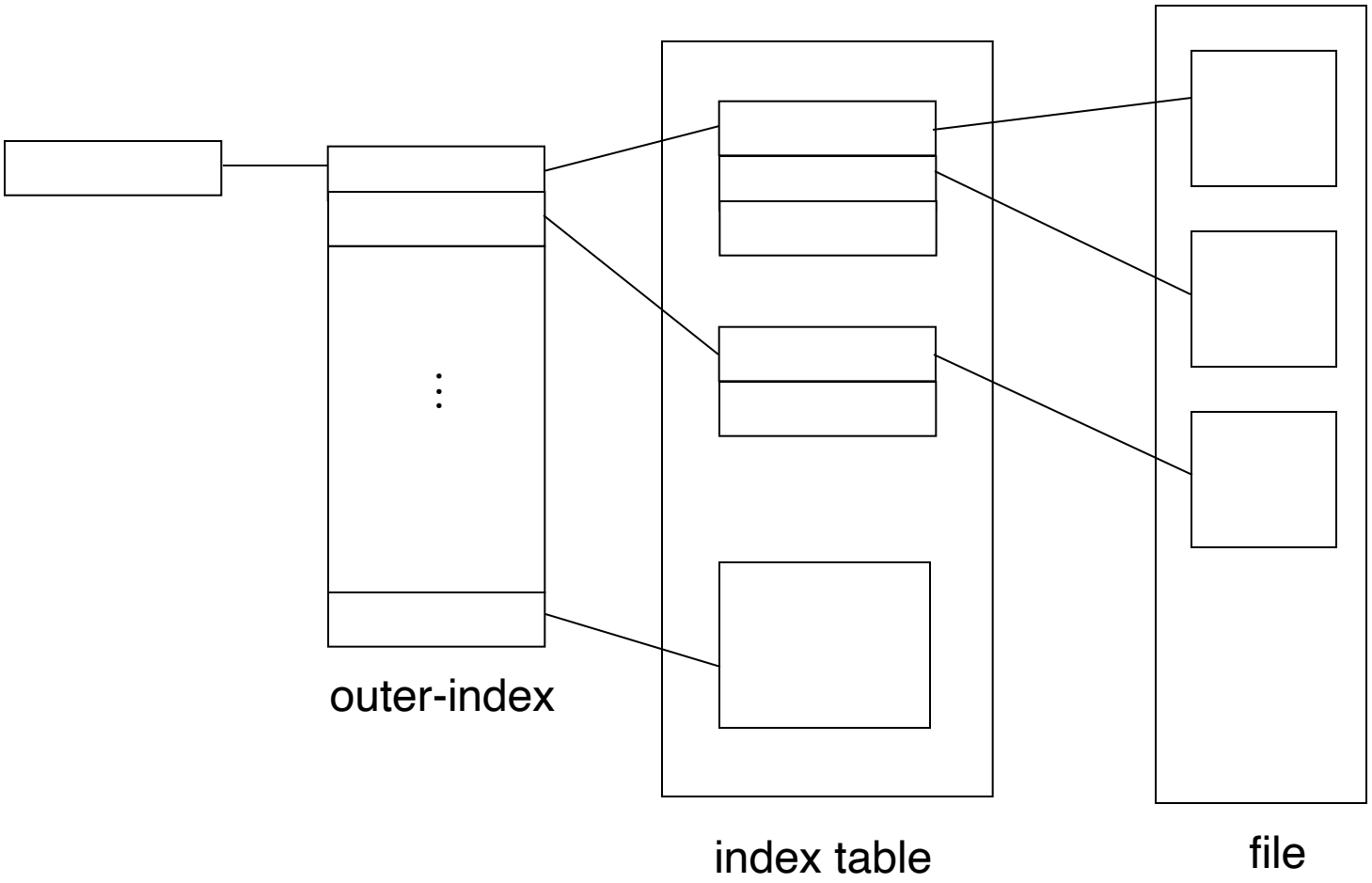
index table

Example of Indexed Allocation

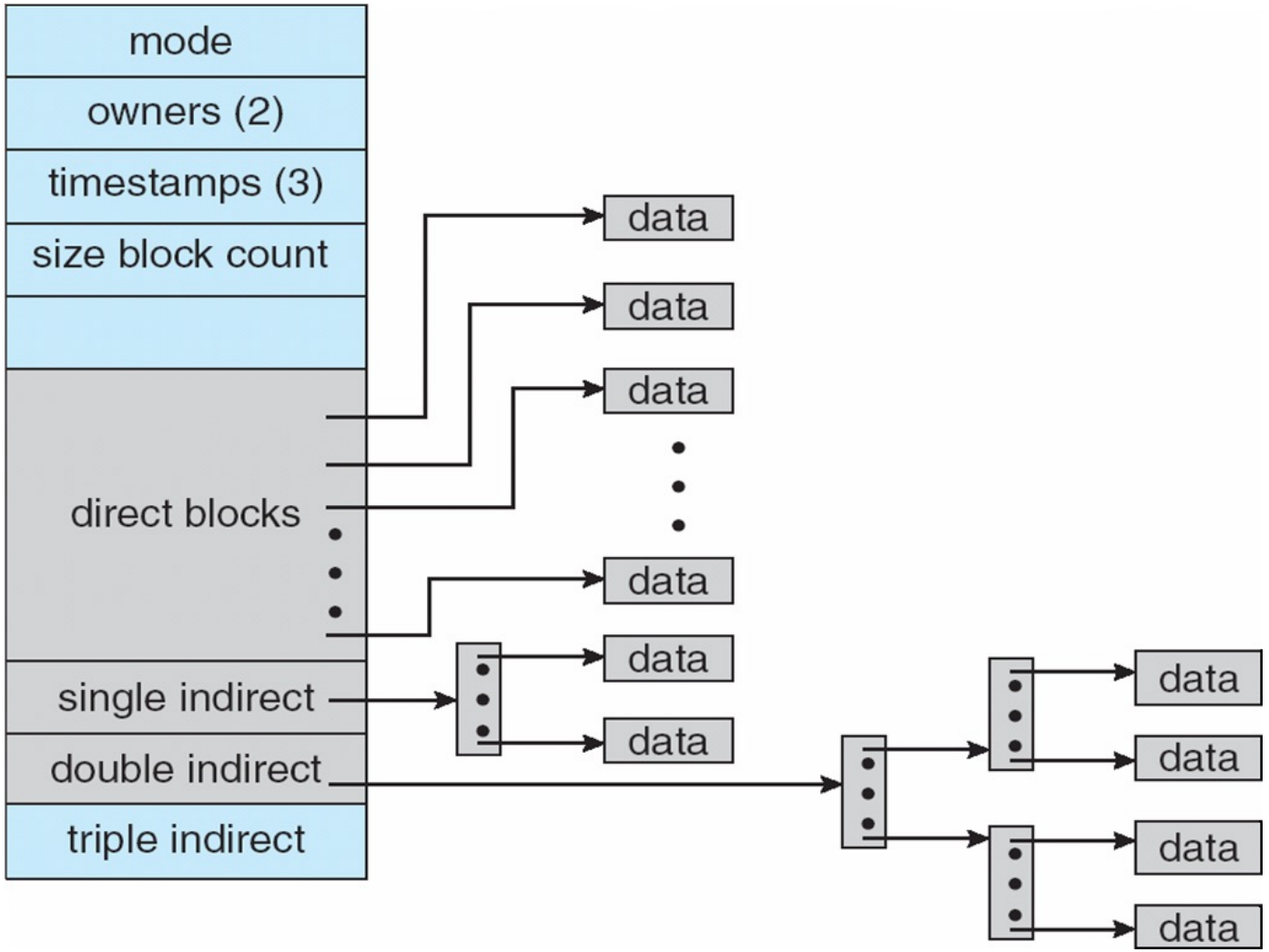


- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

Indexed Allocation – Mapping (Cont.)

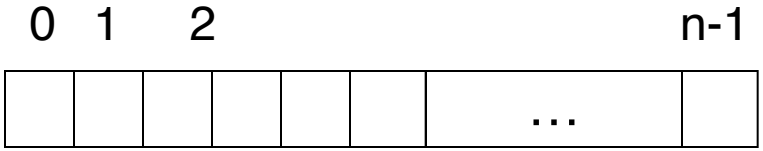


Combined Scheme: UNIX UFS (4K bytes per block)



Free-Space Management

- Bit vector (*n* blocks)



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

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

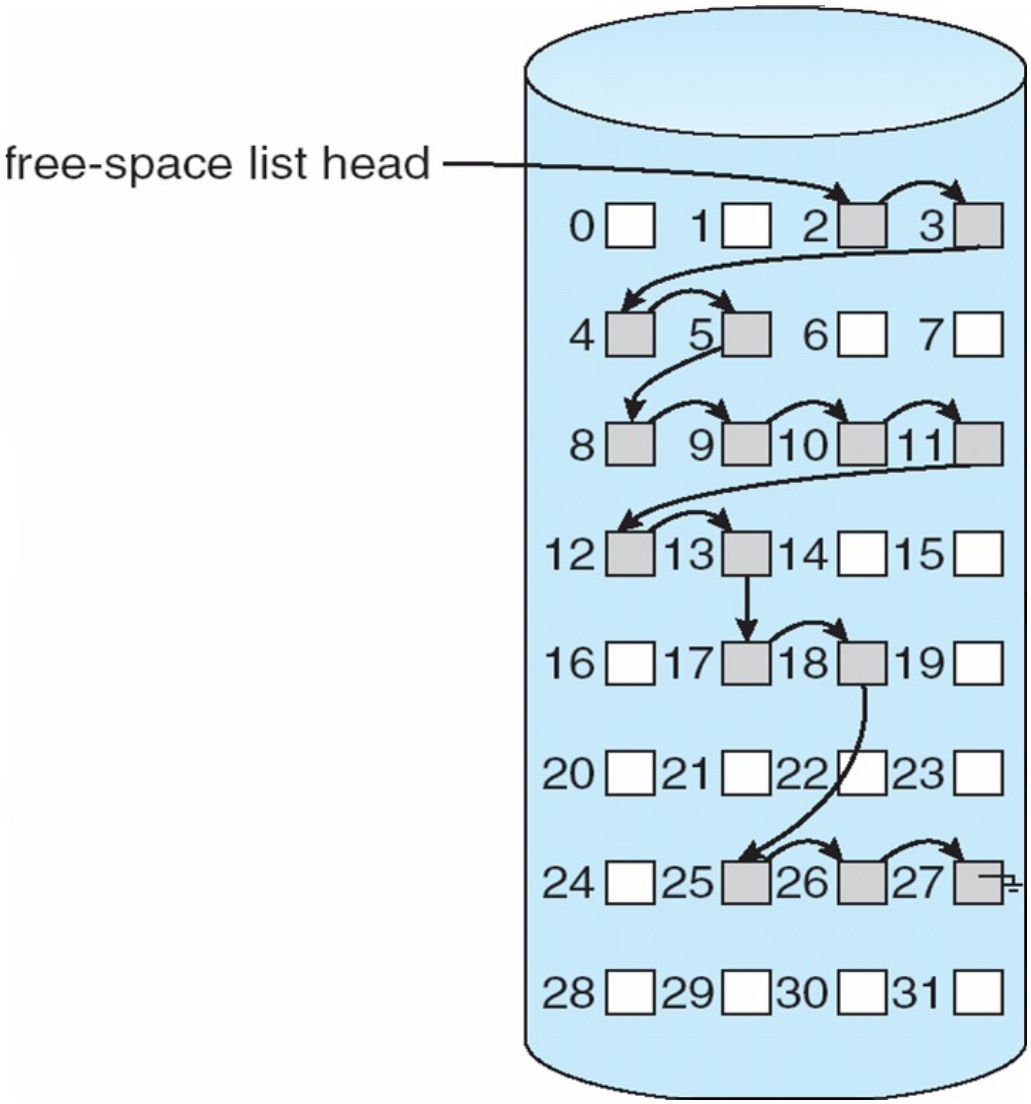
    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space

Free-Space Management (Cont.)

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk
    - Copy in memory and disk may differ
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

Linked Free Space List on Disk



End of Chapter 11