Module 11: File-System Interface

- File Concept
- Access :Methods
- Directory Structure
- Protection
- Consistency Semantics

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides:
  - Operating system
  - Program

File Attributes

- **Name** – only information kept in human-readable form.
- **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

- create
- write
- read
- reposition within file – file seek
- 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 Types – name, extension

File Type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	complied, machine language, not linked
Source code	c, p, pas, 177, 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, etc.	various word-processor formats
Library	lib, a	libraries of routines
Print or view	ps, dvi, gif	ASCII or binary file
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed.

Access Methods

- Sequential Access

linked list

music
movie

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

- Direct Access

page#/frame#

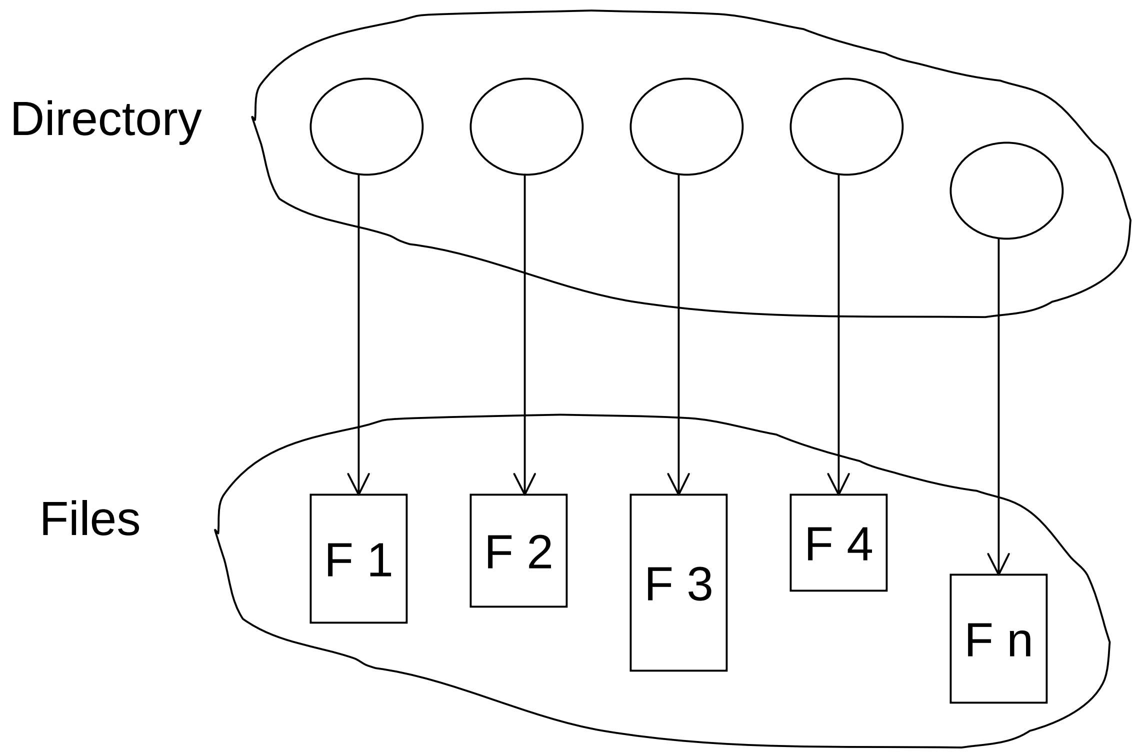
database

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

n = relative block number

Directory Structure

- A collection of nodes containing information about all files.



- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.

Information in a Device Directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

Operations Performed on Directory

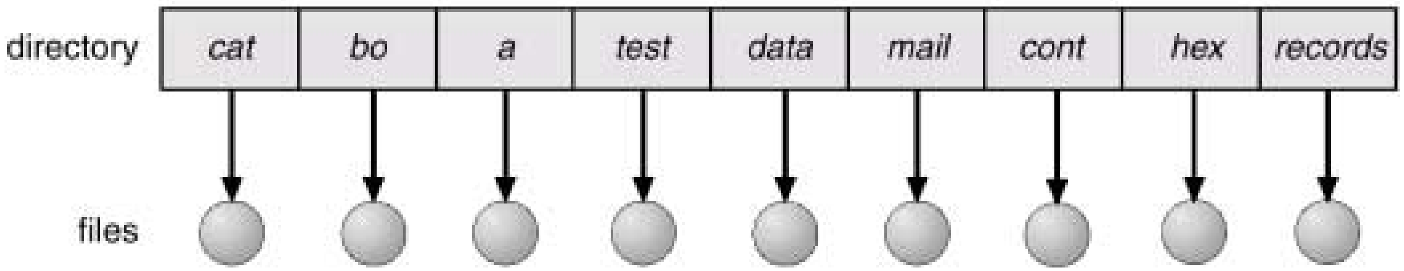
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

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 Pascal programs, all games, …)

Single-Level Directory

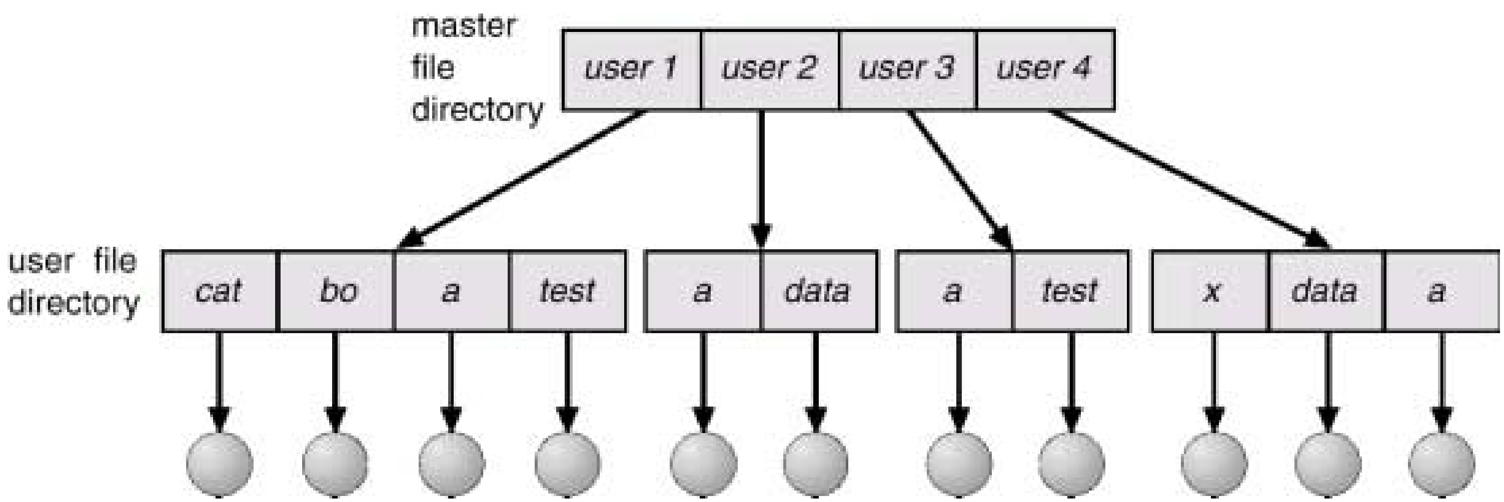
- A single directory for all users.



- Naming problem
- Grouping problem

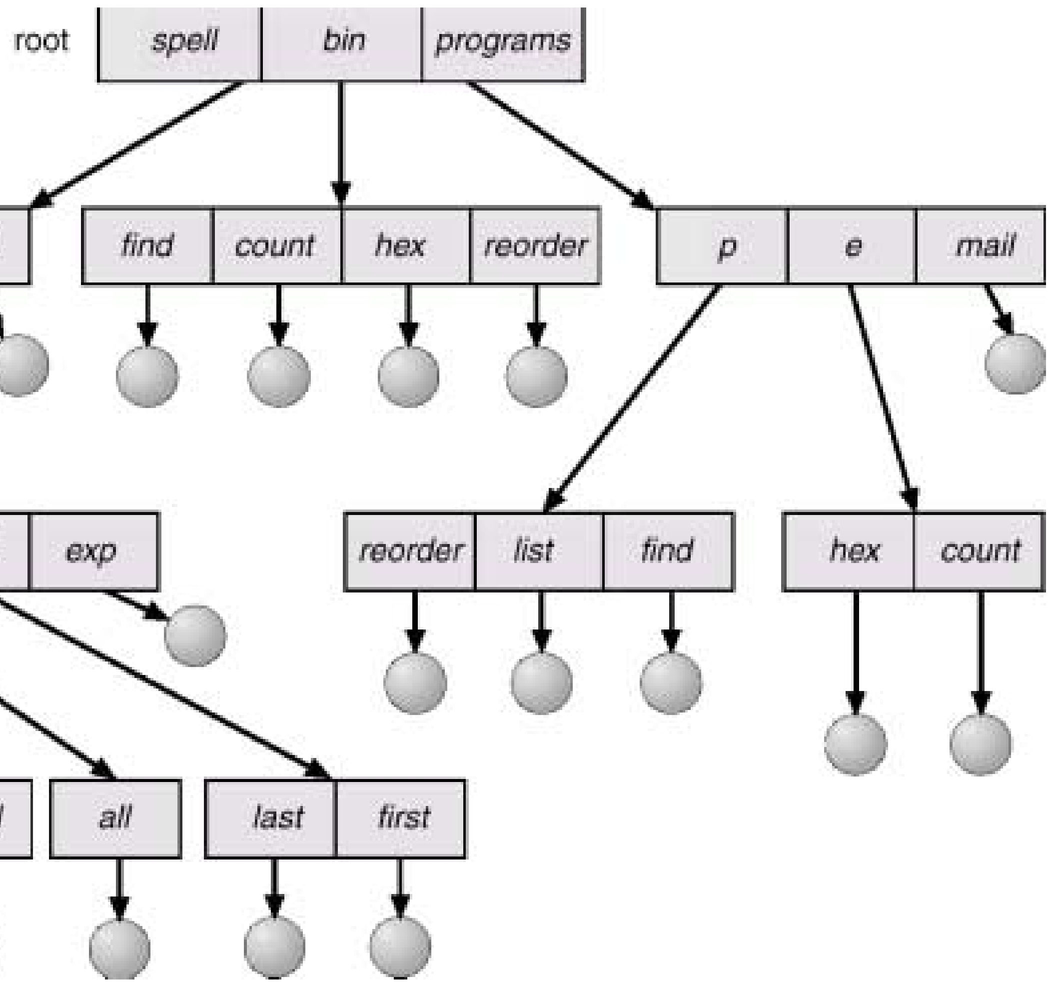
Two-Level Directory

- Separate directory for each user.



- Path name
- Can have the saem 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

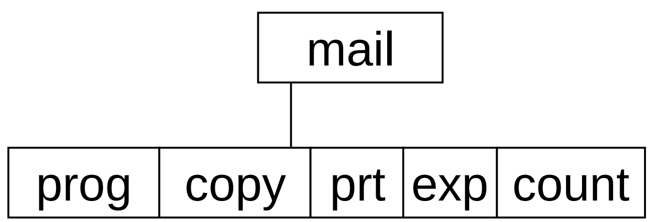
  rm <file-name>

- Creating a new subdirectory is done in current directory.

  mkdir <dir-name>

  Example: if in current directory /spell/mail

  mkdir count

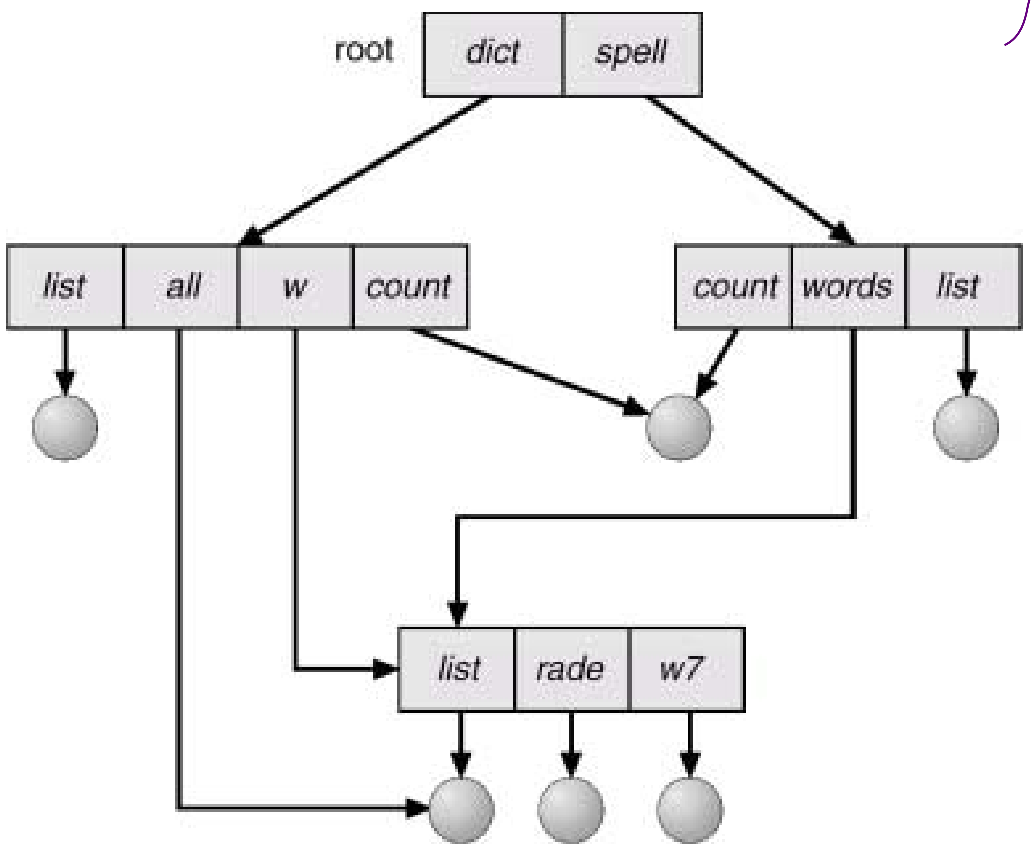


- Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.

Acyclic-Graph Directories

- Have shared subdirectories and files.

No cycles



Acyclic-Graph Directories (Cont.)

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

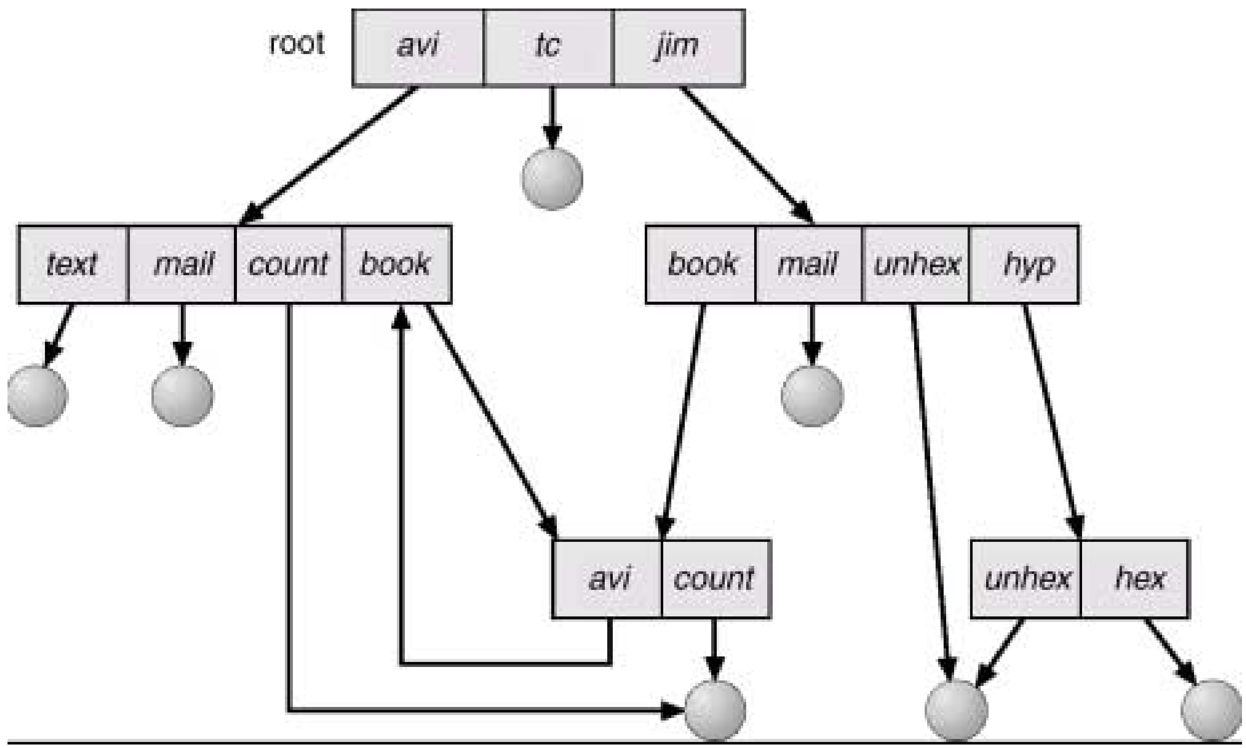
  Solutions:

  – 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.

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) groups 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*

- File-System Structure
- Allocation Methods
- Free-Space Management
- Directory Implementation
- Efficiency and Performance
- Recovery

File-System Structure

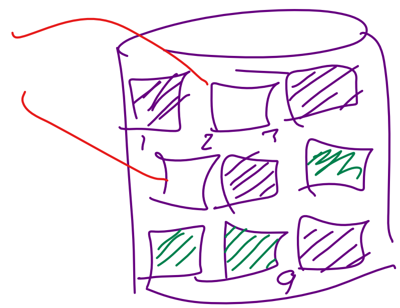
- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- *File control block* – storage structure consisting of information about a file.

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.
- Mapping from logical to physical.

file 3 blocks

Wasteful



start	len
5	3

Q

LA/512

R

Supports direct access

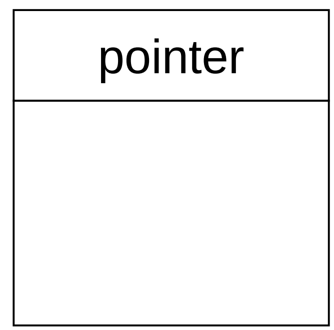
– Block to be accessed = ! + starting address

– Displacement into block = R

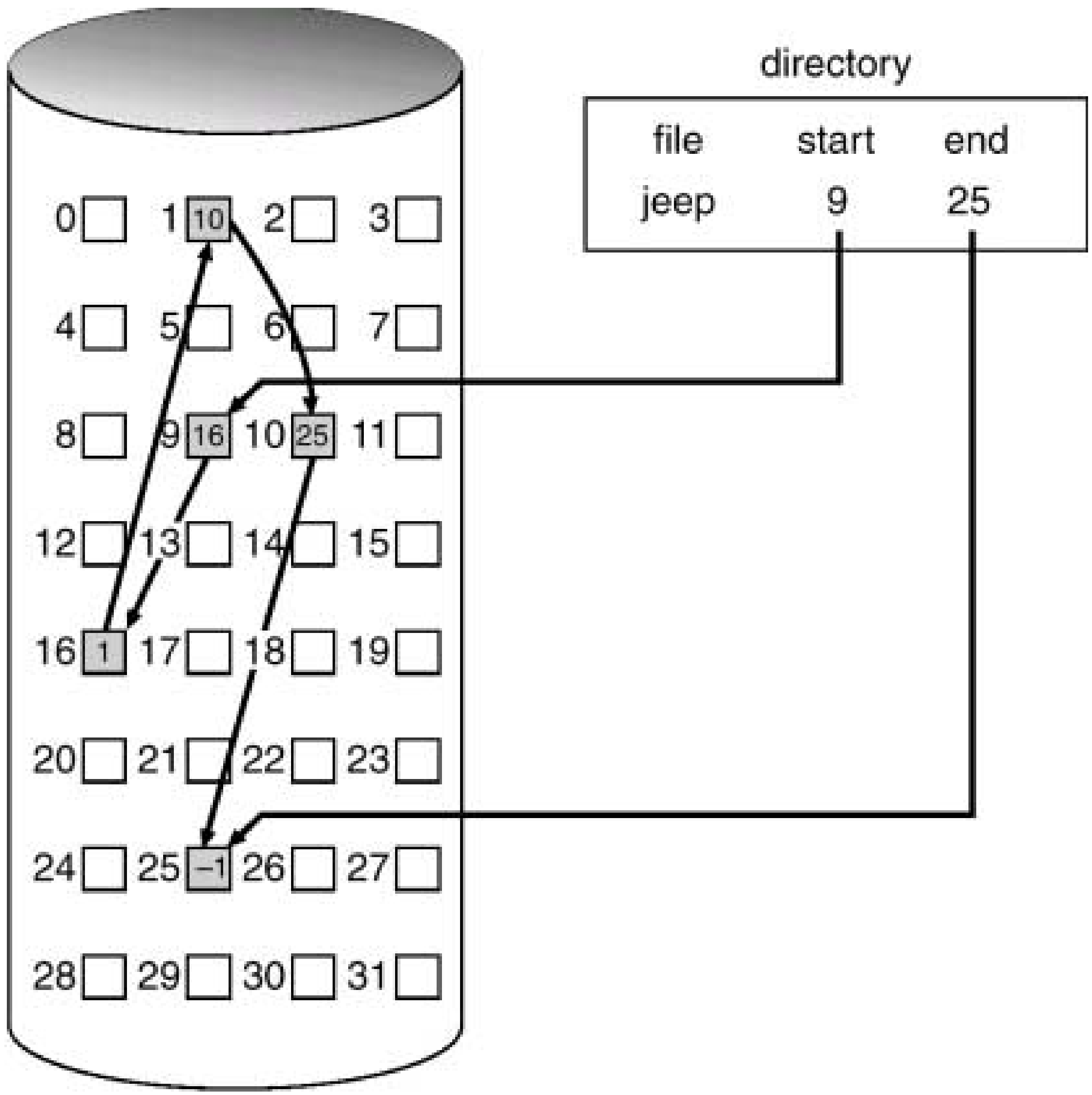
Linked Allocation

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

block =



- Allocate as needed, link together; e.g., file starts at block 9



Linked Allocation (Cont.)

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

$$LA/511 \begin{cases} Q \\ R \end{cases}$$

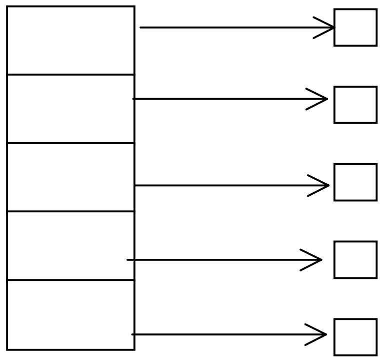
  – Block to be accessed is the Qth block in the linked chain of blocks representing the file.

  – Displacement into block = R + 1

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

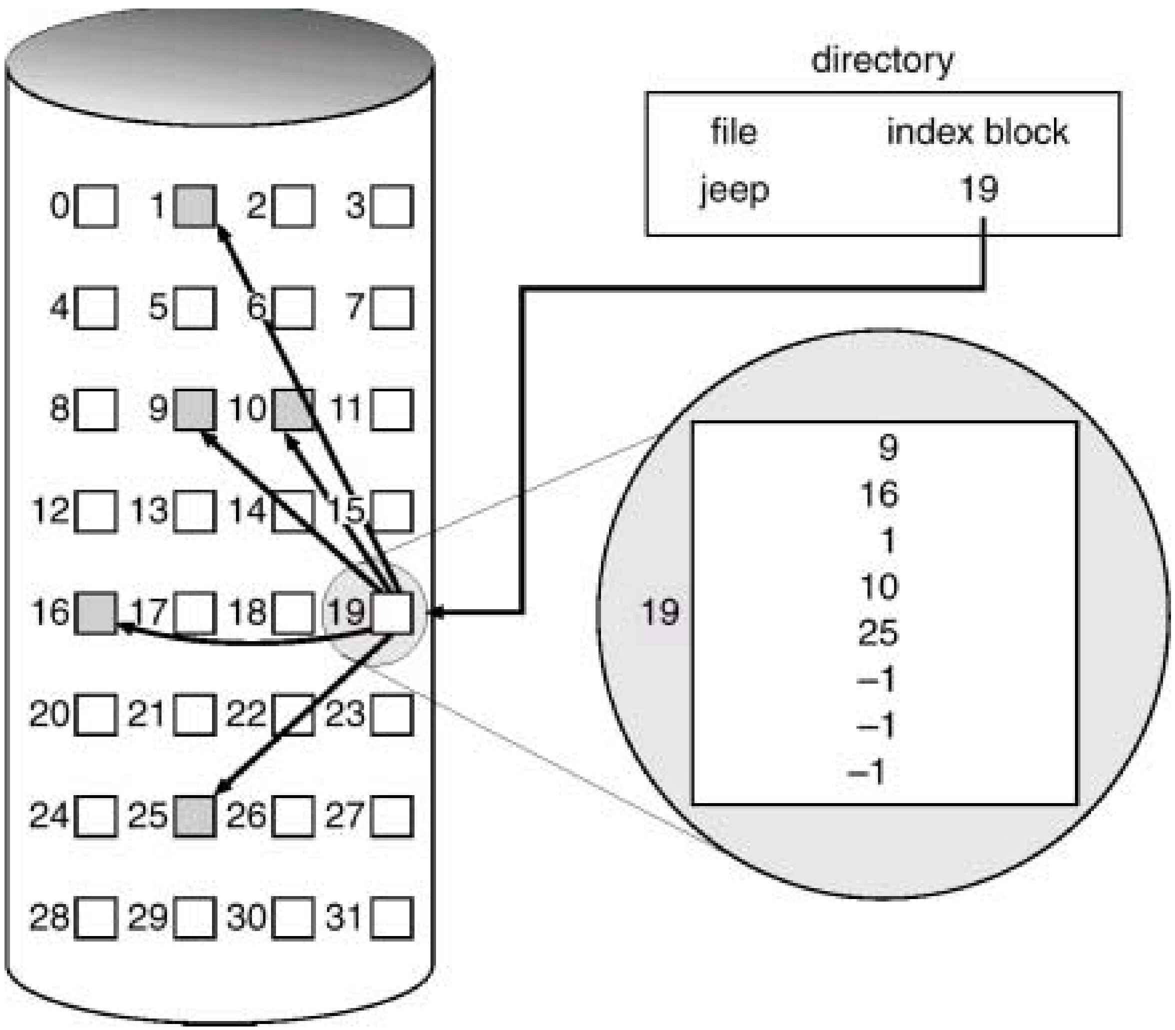
Indexed Allocation

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



index table

Example of 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.

$$\text{LA}/512 \begin{matrix} \nearrow Q \\ \searrow R \end{matrix}$$

  – *Q* = displacement into index table
  – *R* = displacement into block

Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).

$$\begin{tikzcd}[row sep=small] & Q_1 \\ \text{LA} / (512 \text{ x } 511) \arrow[ur, dash] \arrow[dr, dash] & \\ & R_1 \end{tikzcd}$$

  – Q_1 = block of index table

  – R_1 is used as follows:

$$\begin{tikzcd}[row sep=small] & Q_2 \\ R_1 / 512 \arrow[ur, dash] \arrow[dr, dash] & \\ & R_2 \end{tikzcd}$$

  – Q_2 = displacement into block of index table

  – R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)

- Two-level index (maximum file size is 512^3)

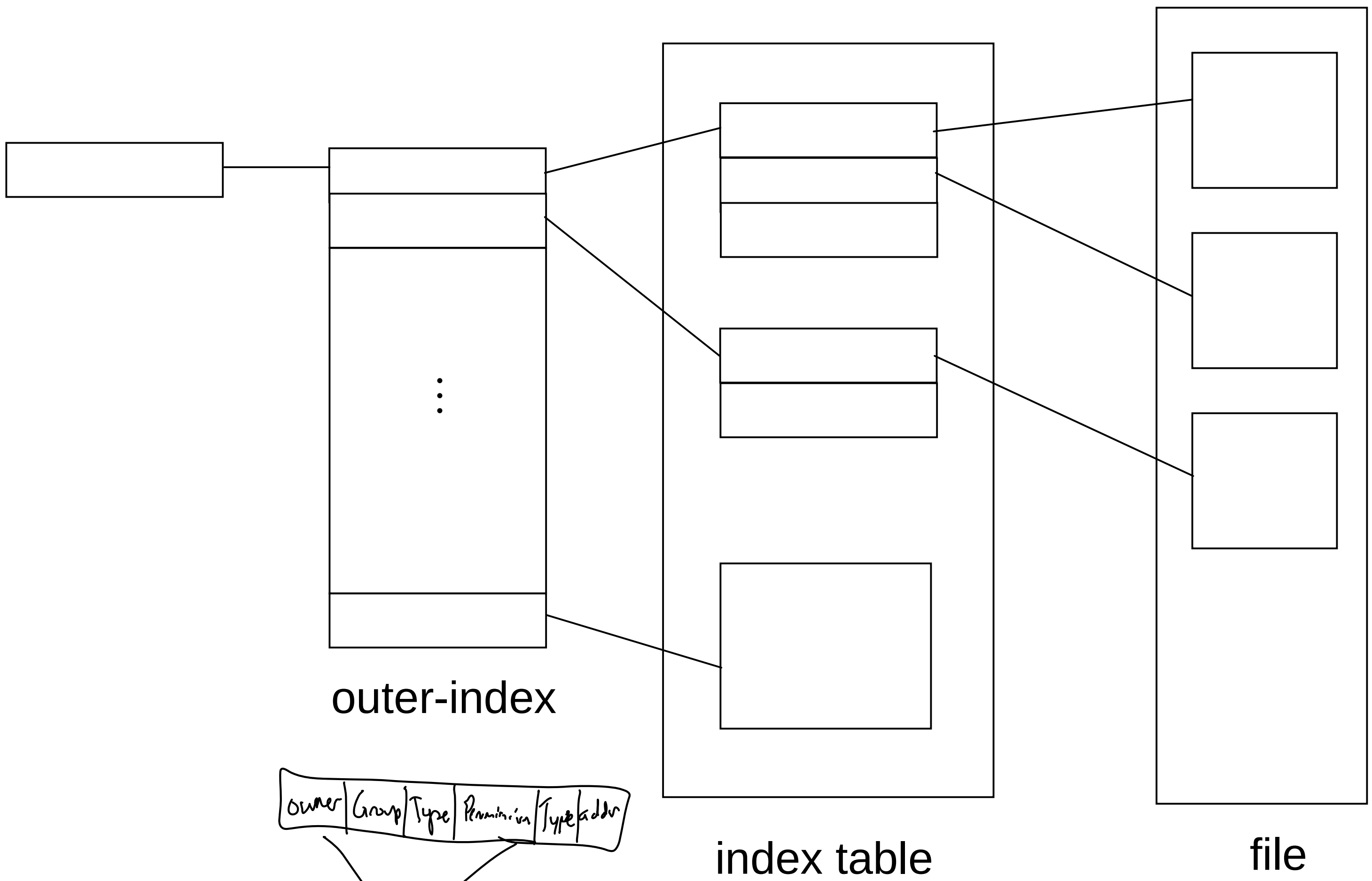
$$LA\ /\ (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

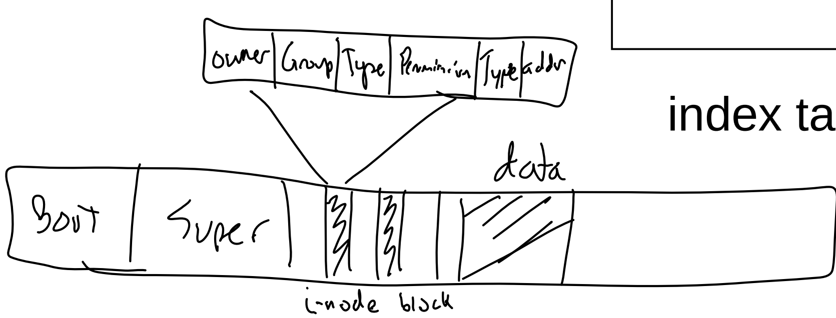
  - Q_1 = displacement into outer-index
  - R_1 is used as follows:

$$R_1\ /\ 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

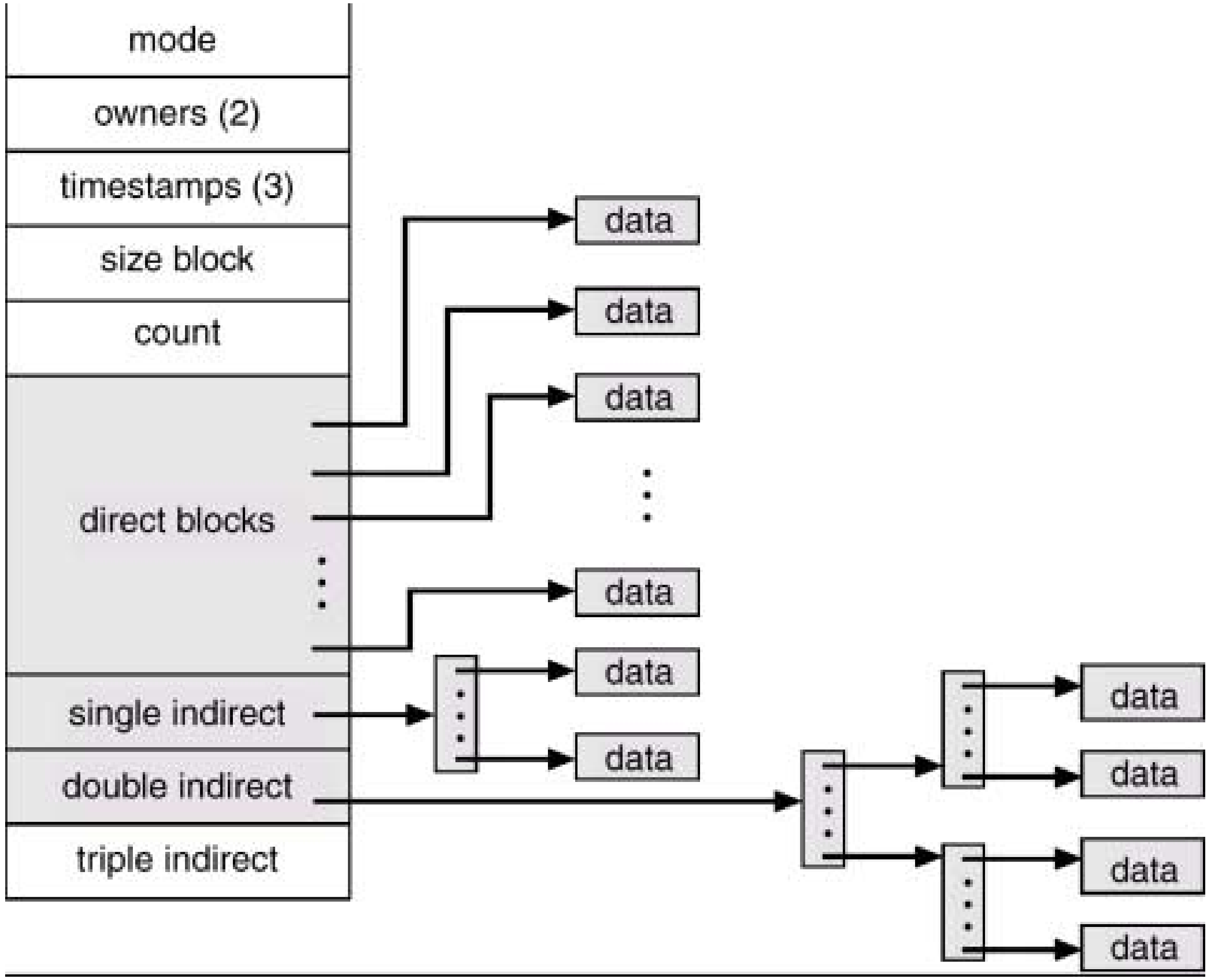
  - Q_2 = displacement into block of index table
  - R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)





Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- Bit vector (*n* blocks)

0	1	2				n-1
					…	

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

- 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
- Grouping
- Counting

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

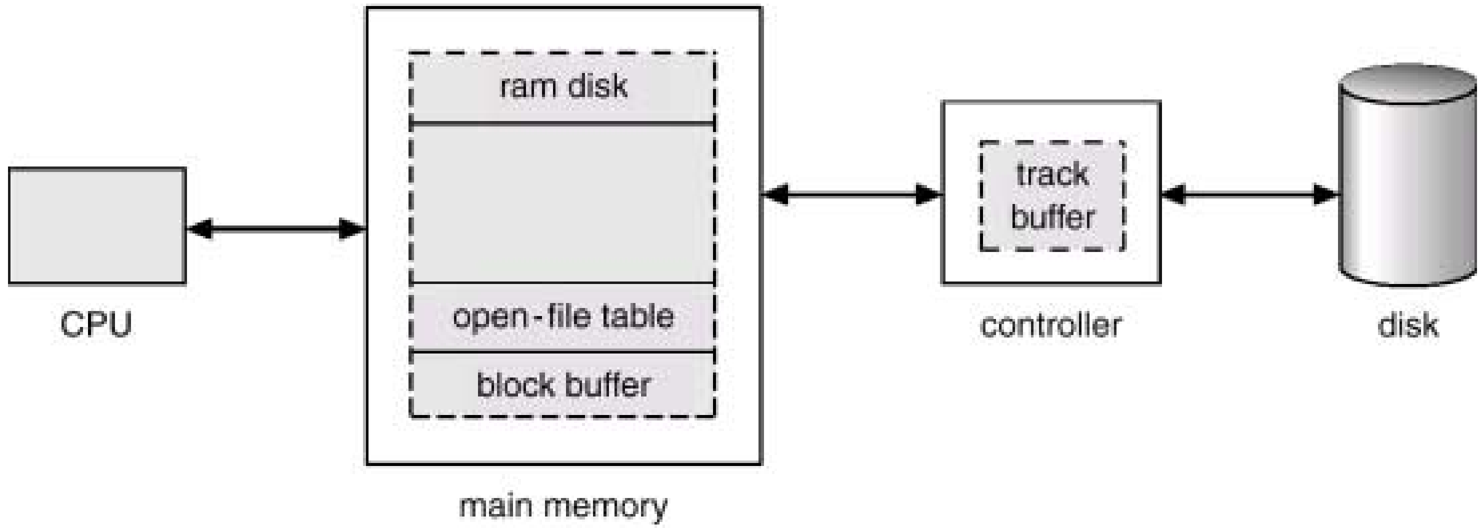
Directory Implementation

- 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

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 sued blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memroy as virtual disk, or RAM disk.

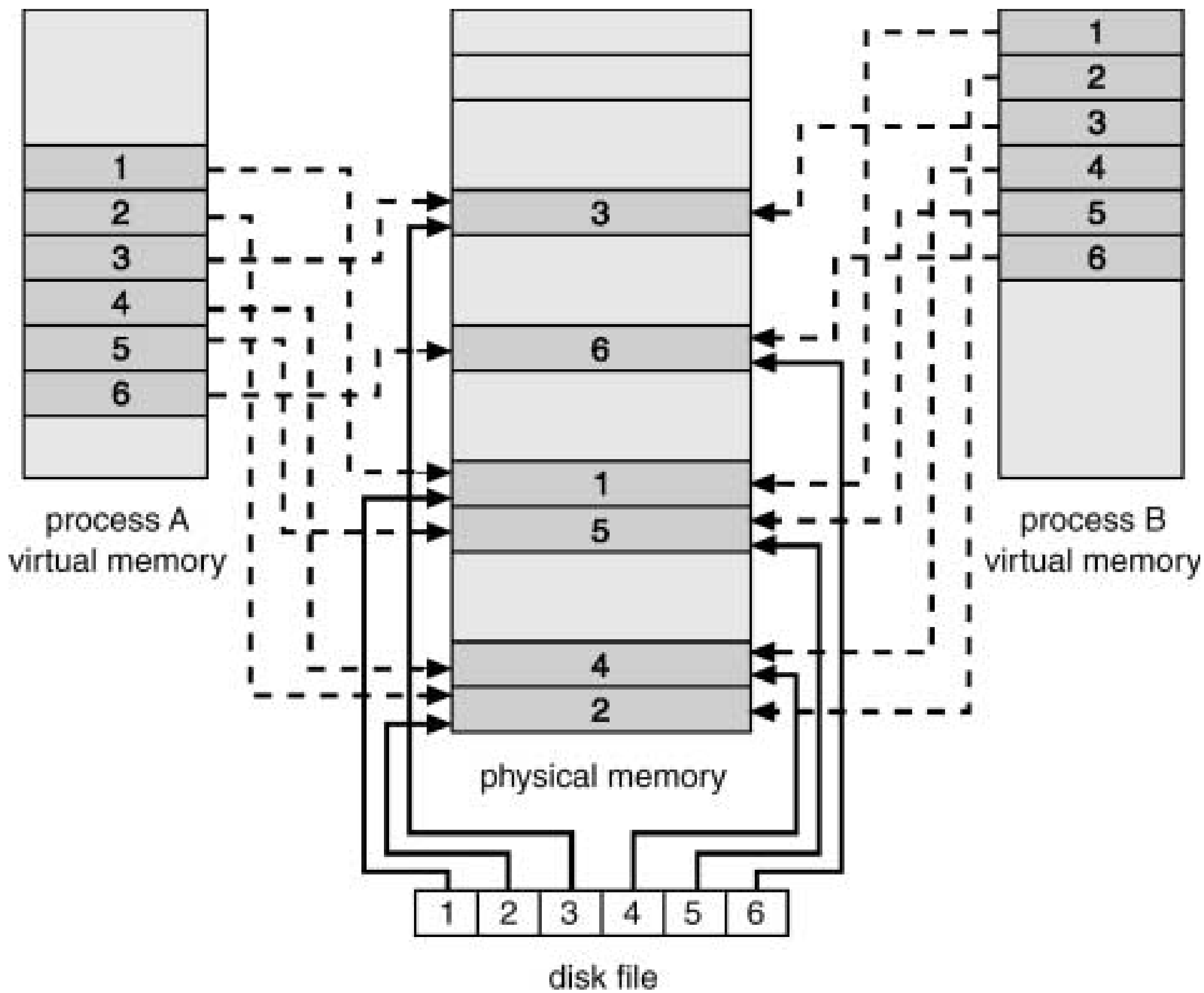
Various Disk-Caching Locations



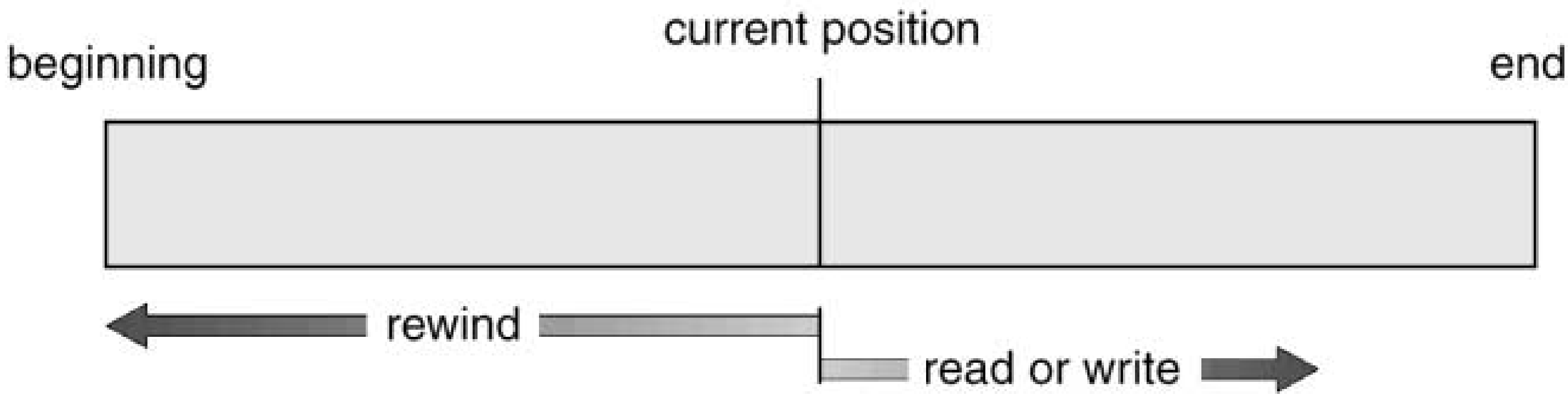
Recovery

- Consistency checker – 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.

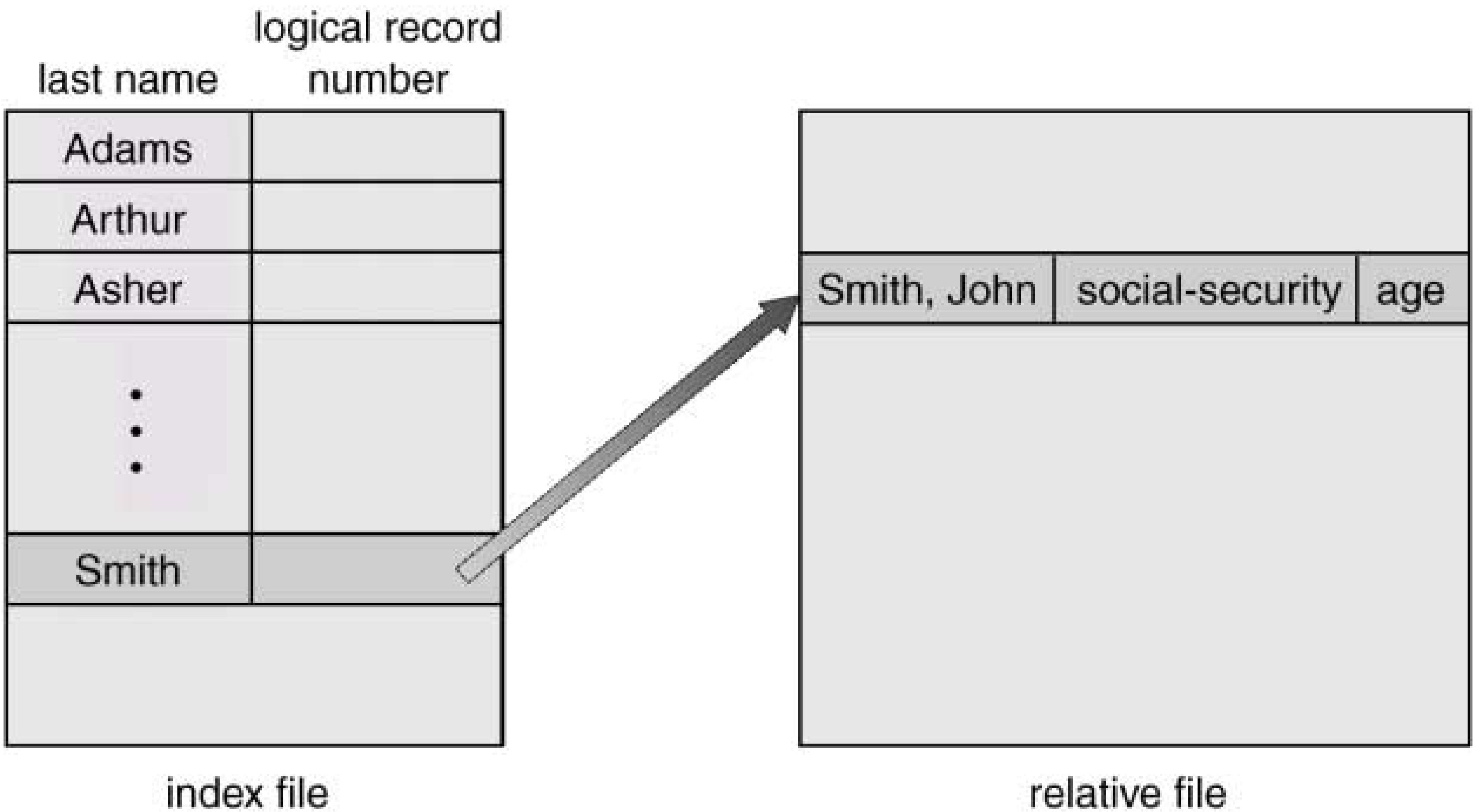
Memory-mapped Files



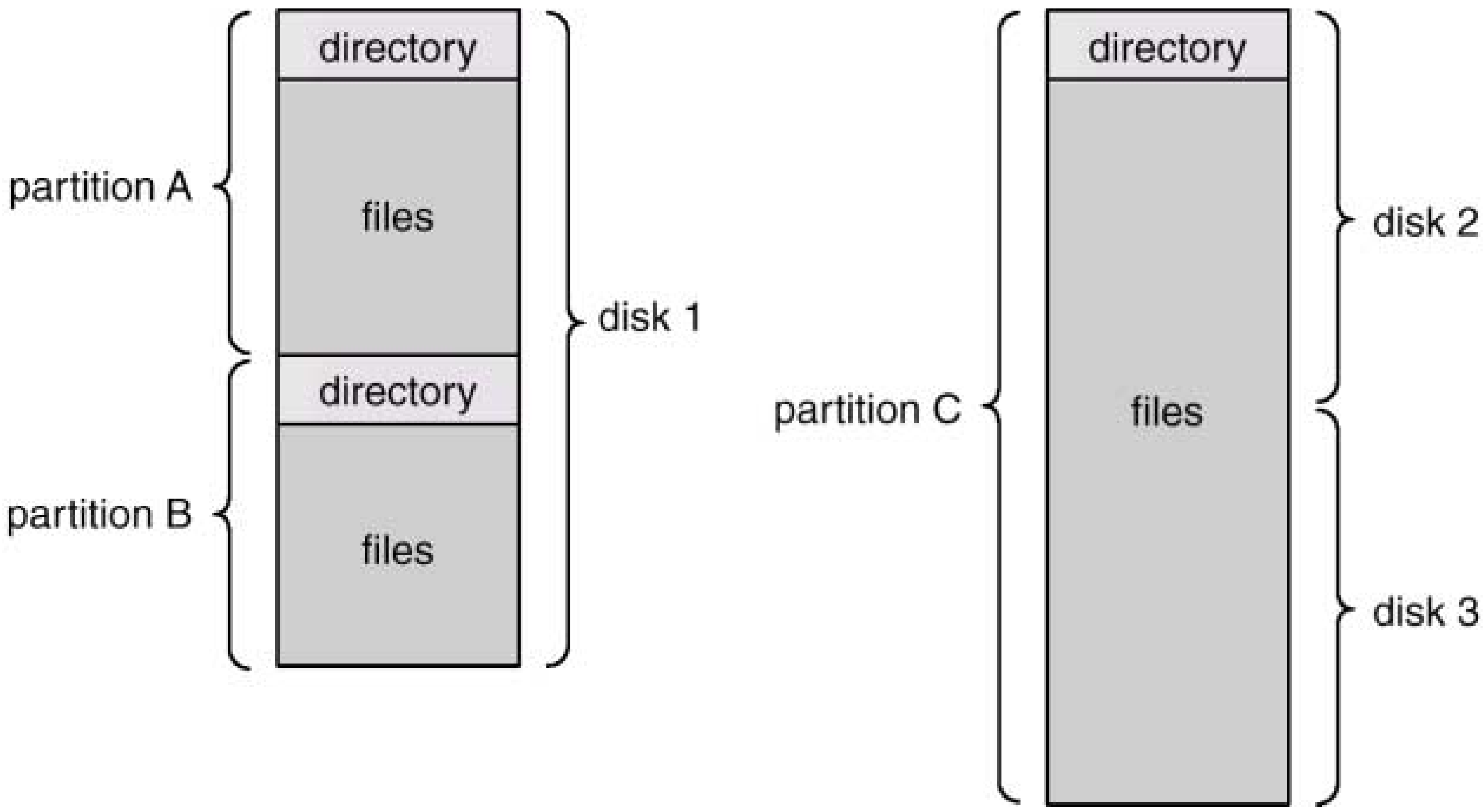
Sequential-access File



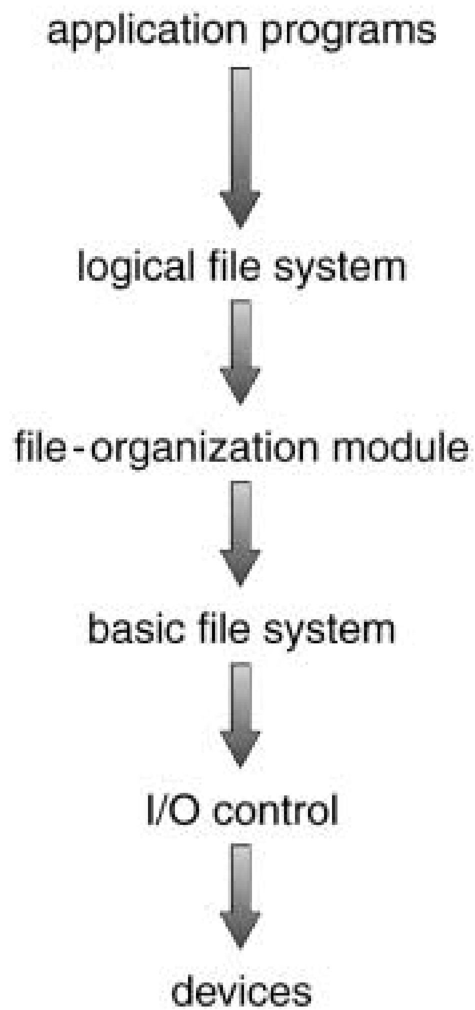
Example of Index and Relative Files



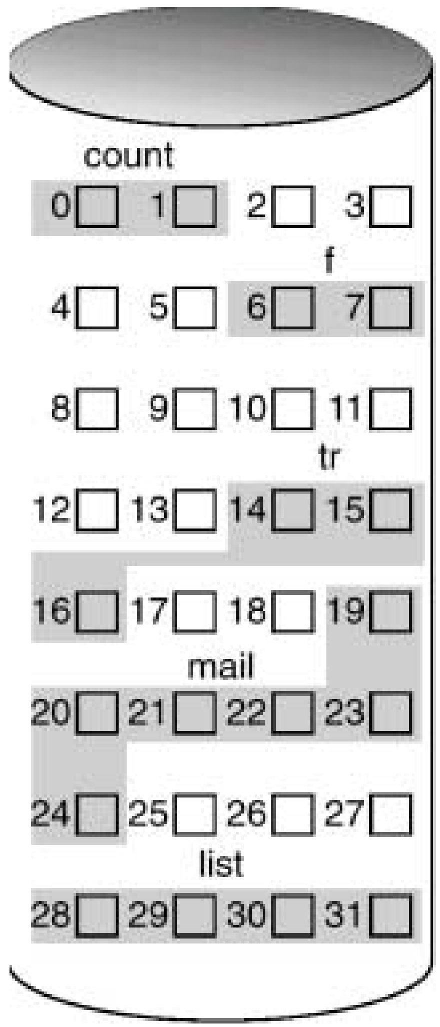
Typical File-System Organization



Layered File System



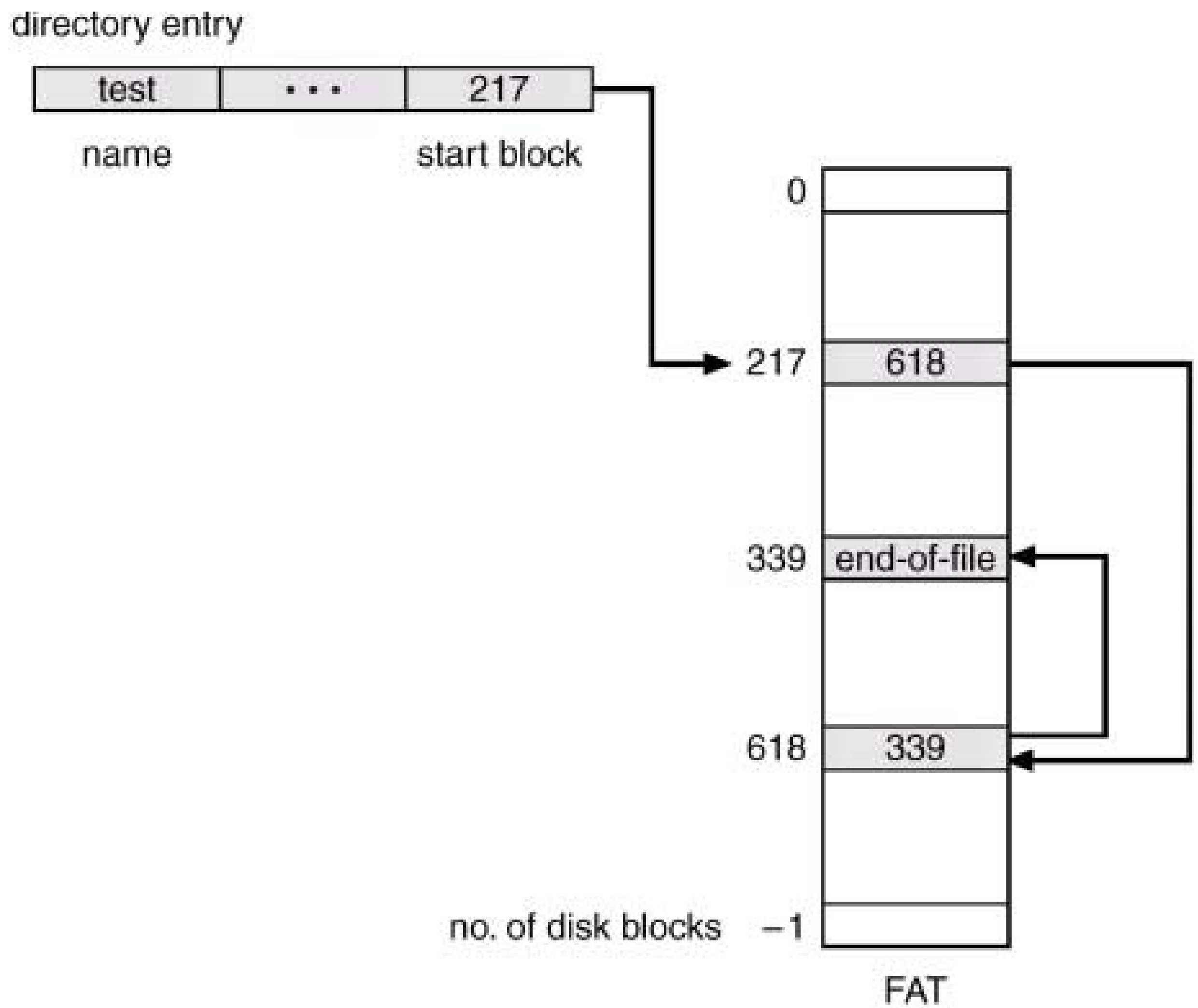
Contiguous Allocation of Disk Space



directory

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

File-Allocation Table



Linked Free-Space List on Disk