Operating Systems
COMP2006

File Systems
Lecture 8

File Systems

References:

Silberschatz, Galvin, and Gagne, *Operating System Concepts*, Chapters 11 and 12.

Topics:

- File concept and access methods.
- Directory structures.
- File protection.
- File allocation methods.
- Free-space management.
- Directory implementation.
- Efficiency and performance.

File Management

- A file consists of sequence of bits, bytes lines, or records
  - Their meanings are defined by their creators.
- Files may be free-form (such as text files) or may be formatted rigidly.
  - Commonly, files represent programs (both source and object forms) and data.

- The OS is responsible for the following activities:
  - Creation and deletion of files.
  - Creation and deletion of directories.
  - Support of primitives for handling files and directories.
  - Mapping files onto secondary storage (i.e., disk structure).
  - Backup of files onto stable (non-volatile) media.

Note:

1 byte = 8 bits

1 word = a storage unit defined by the computer architecture. It can be 1 byte, 2 bytes, …, 8 bytes.

1 KB=2^{10}=1024 bytes. 1 MB=2^{20} bytes. 1 GB=2^{30} bytes. 1 TB=2^{40} bytes

File concept

- A file is a named collection of related info stored in secondary storage.
- User's perspective: a file is the smallest unit of logical secondary storage
  - data is written to secondary storage in the form of files.

- OS maps files (logical-storage units) to (non-volatile) physical devices.

- Many different types of information may be stored in a file:
  - Data files can be: Numeric, alphabetic, alphanumeric, or binary.
  - Program files can be:
    - Source: a sequence of routines, functions, declarations, and executable statements.
    - Object: a sequence of bytes organized into blocks understandable by the system's linker.
    - executable: a series of code sections that the loader can bring into memory and execute.

File concept (cont.)

- **File system** consists of:
  - A collection of files – each storing related data.
  - A directory structure – organizes and provides information about all the files in the system.
  - Partitions – used to separate physically or logically large collections of directories.

- **File Attributes:** Information about files are kept in the directory structure, which is maintained on the disk, and typically consist of:
  - **Name** – A file is referred to by its 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 – in bytes, words, or blocks.
  - **Protection** – controls who can do reading, writing, executing.
  - **Time**, **date**, and **user identification** – data for protection, security, and usage monitoring.

File operations

- The OS provides (through system calls) the following basic file operations:
  - Creating a file.
  - Writing a file.
  - Reading a file.
  - Reposition within file – file seek.
  - Deleting a file.
  - Truncating a file.
- Other common operations:
  - Appending a file.
  - Renaming a file.
  - Copying file.

- Most operations involve searching the directory for entry to the named file.
  - To avoid constant searching, OS keeps **open file table** that contains information about all open files.

File operations (cont.)

- Most system requires a file to be opened explicitly
  - **open(F_i)** – Search the directory on disk for entry F_i, copy directory entry (typically a pointer) into the **open-file-table**.
  - **close(F_i)** – Remove directory entry for F_i from open-file-table.
- In general, an open file needs the following information:
  - File pointer – unique for each process operating on the file.
  - File open count – tracks the number of processes calling **open**/**close** to certain file.
    - If count = 0, entry is removed from open-file-table.
  - Disk location of the file: keep in memory, to avoid reading it from disk for each operation.

- Multiuser OS, such as UNIX, keeps two levels of open-file-table:
  - **Per-process table**: keeps information of all open files for each process
    - e.g., file pointer for each file for use in the read and write calls, and pointers to the system-wide table.
  - **System-wide open-file-table**: contains information which is process-independent
    - e.g., file location on disk, access dates, file size

File types

- OS that recognizes and support file types avoids mistake
  - e.g., printing a binary-object of a program,
- Implement file type as part of file name
  - File name is split into – a *name* and an *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, perl, asm	source code in various languages
Batch	bat, sh	commands to the command interpreter
markup	txt, html, xml	textual data, documents
Word processor	docx, rtf, etc	various word-processor formats
Library	lib, a, dll	libraries of routines
Print or view	gif, pdf, jpg	ASCII or binary file for printing/viewing
Archive	zip, tar	Related files grouped into one file, sometimes compressed
multimedia	Mpeg, mp3	Binary file containing audio or A/V information

File Structure

- Files must conform to a required structure that is understood by the OS.
- Some of the file structures are:
  - Free forms, such as text files – sequence of characters organized into lines/pages.
  - Simple record structure.
    - Lines.
    - Fixed length.
    - Variable length.
  - Complex structures.
    - Formatted document.
    - Relocatable load file.
- OS must support at least one structure – i.e., for executable files – so that the system is able to load and run programs.

File Structure (cont.)

Who decides?

- Operating system: the more file structures, the more complex the OS to support them.
- Program/its creator: the problem if the used structure is not supported by OS.

Example: UNIX

- A file is a sequence of 8-bit byte; no interpretation from OS for these bits.
  - The scheme provides maximum flexibility, but little support
    - Each application program must include its own code to interpret an input file into the appropriate structure.

Internal file structure

- All disk I/O is performed in units of one block (physical record) and all blocks are the same size (e.g., 512 bytes).
- Logical records may vary in size.
  - Solution: packing a number of logical records into physical blocks.
    - This may create internal fragmentation.

Access Methods

Information in a file can be accessed using **sequential access** and **direct access**.

Sequential access

- Information in the file is processed in order, one record after the other.
  - Based on the *tape* model.
  - Editors and compilers use this model.

- Common operations:
  - *read*: reads *next* portion of the file, and advances file pointer.
  - *write*: appends to the end of file, advances file pointer.
  - *reset*: file pointer to the beginning of file.

Access Methods (cont.)

Direct access (relative access).

- A file is made up of fixed-length logical records that allow programs to read and write records in no particular order.
  - Based on disk model → allow random access to any file block, i.e., reads block 14, 32, and then writes to block 7.
- User provides only *relative block number*: an index relative to the beginning of file.
  - Databases are often of this type.

- Common file operations include:
  - *read n* → read block number *n*.
  - *write n* → write block number *n*.
  - OR: *position to n* followed by *read next*.
  - OR: *position to n* followed by *write next*.

Directory structure

- To manage large number of files, we need to organize them
  - The file system is broken into **partitions/volumes**.
    - One disk may contain several partitions.
    - One partition may also consists of several disks.
  - Each partition has device directory or volume table of contents (called ***directory***).
- Advantage of organizing directory (logically):
  - Efficiency – locate 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 C programs, all games, …).

Directory structure (cont.)

- A directory can be viewed as a symbol table that translates file names into their directory entries.
  - or a collection of nodes containing information about all files.
- Both the directory structure and the files reside on disk.

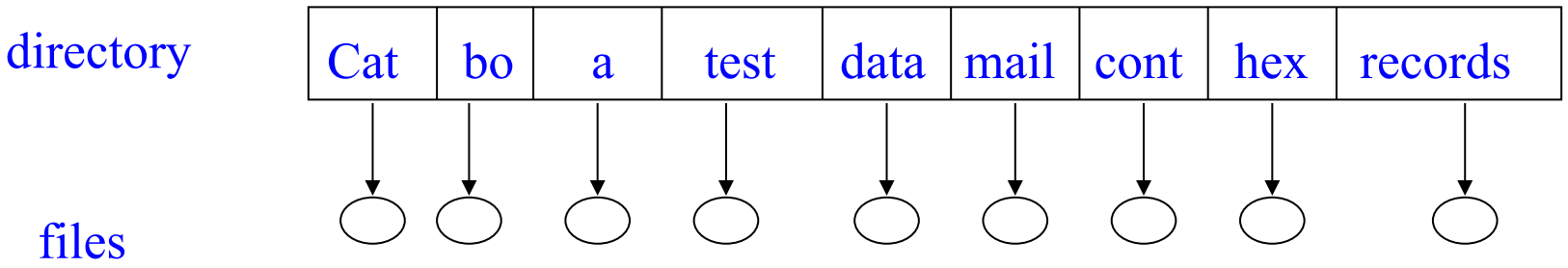
Operations performed on a directory are:

- Search/create/delete/rename a file.
- List a directory and traverse the file system.

Information in a device directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner ID
- Protection information

Single-level directory

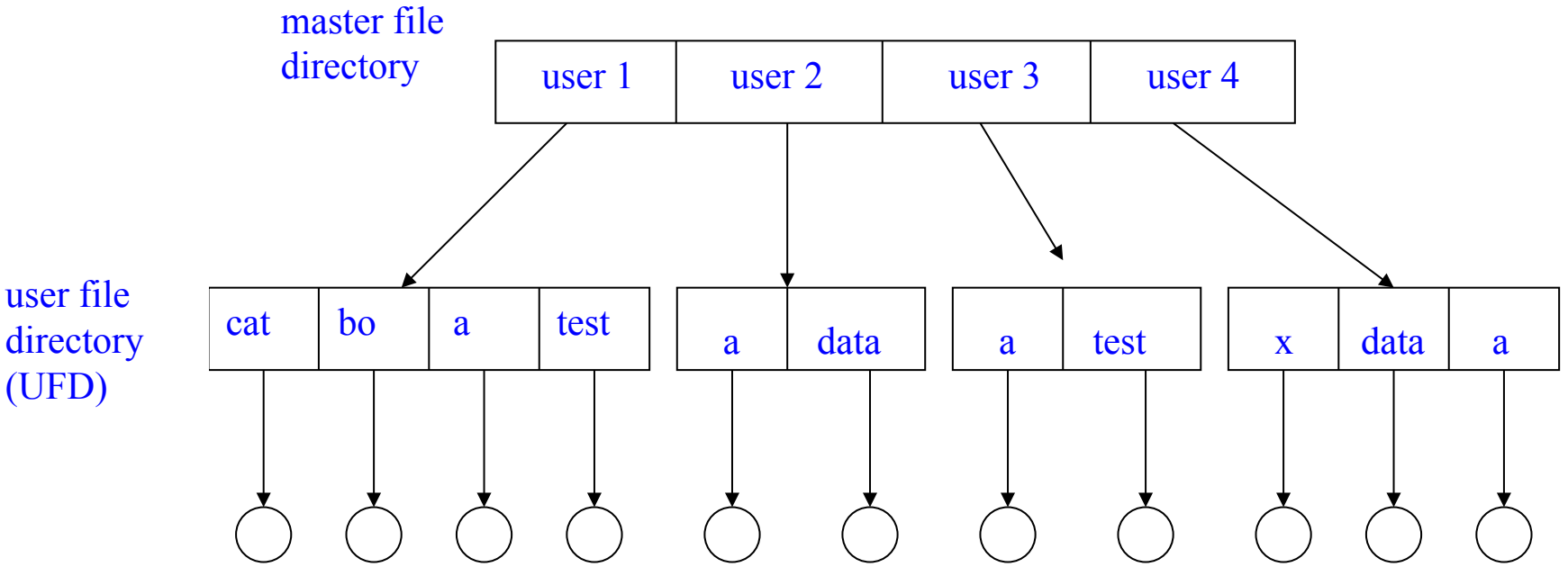


- A single directory for all users.
- The simplest directory structure.

Problems:

- Naming problem: two users may have same file names.
- For hundreds/thousands of files, it is difficult to remember all file names.

Two-level directory - two-level tree



- Create a separate directory for each user.
- Each user needs to search only his/her own user file directory (UFD) to find file.
- OS searches only UFD for creating/deleting/accessing certain file.
- A user name and a file name define a path name
  - every file has a path name.

Two-level directory- two-level tree (cont.)

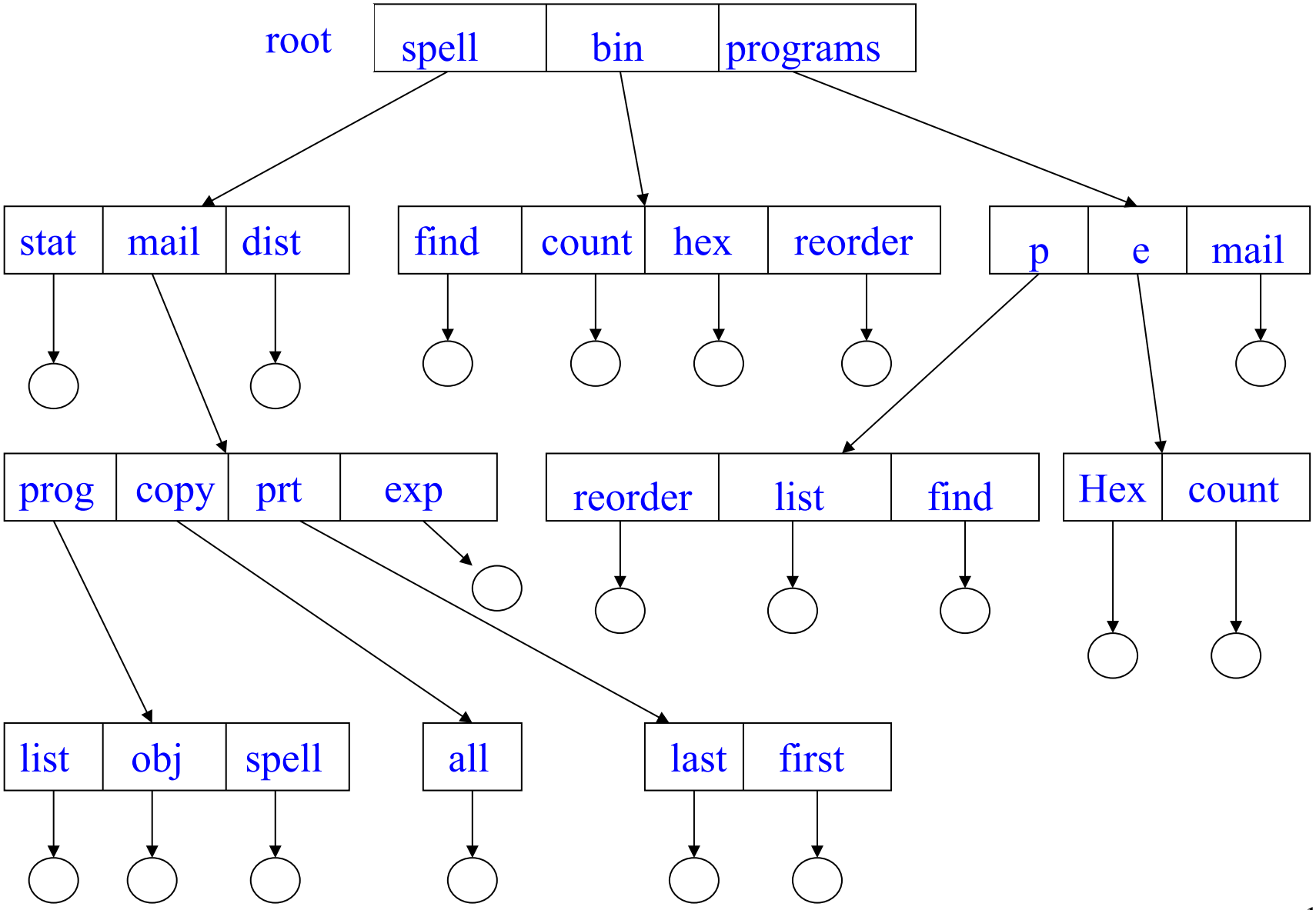
Advantage:

- Solve the name collision problem
  - can have the same file name for different user.
- Efficient searching.

Disadvantage:

- Isolate one user from another
  - users cooperating on same task cannot access one another's files.
- How do users access system files?

Tree-structure directories

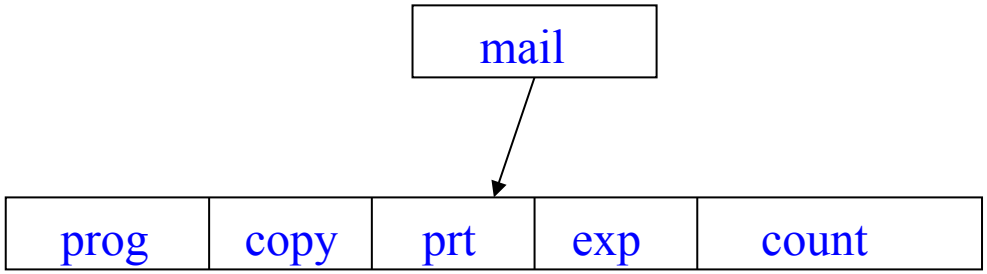


Tree-structure directories (cont.)

- A directory contains a set of files or **subdirectories**.
  - One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
  - The most common directory structure.
- Two types of path name:
  - *absolute* (begins at the root), or
  - *relative* (begins from the current directory) path name.

- What is the policy to delete a non-empty directory?
  - Do not delete a non-empty directory.
  - Allow deleting the directory including its contents (UNIX)
    - Using **rm -*rf***

Example (UNIX)

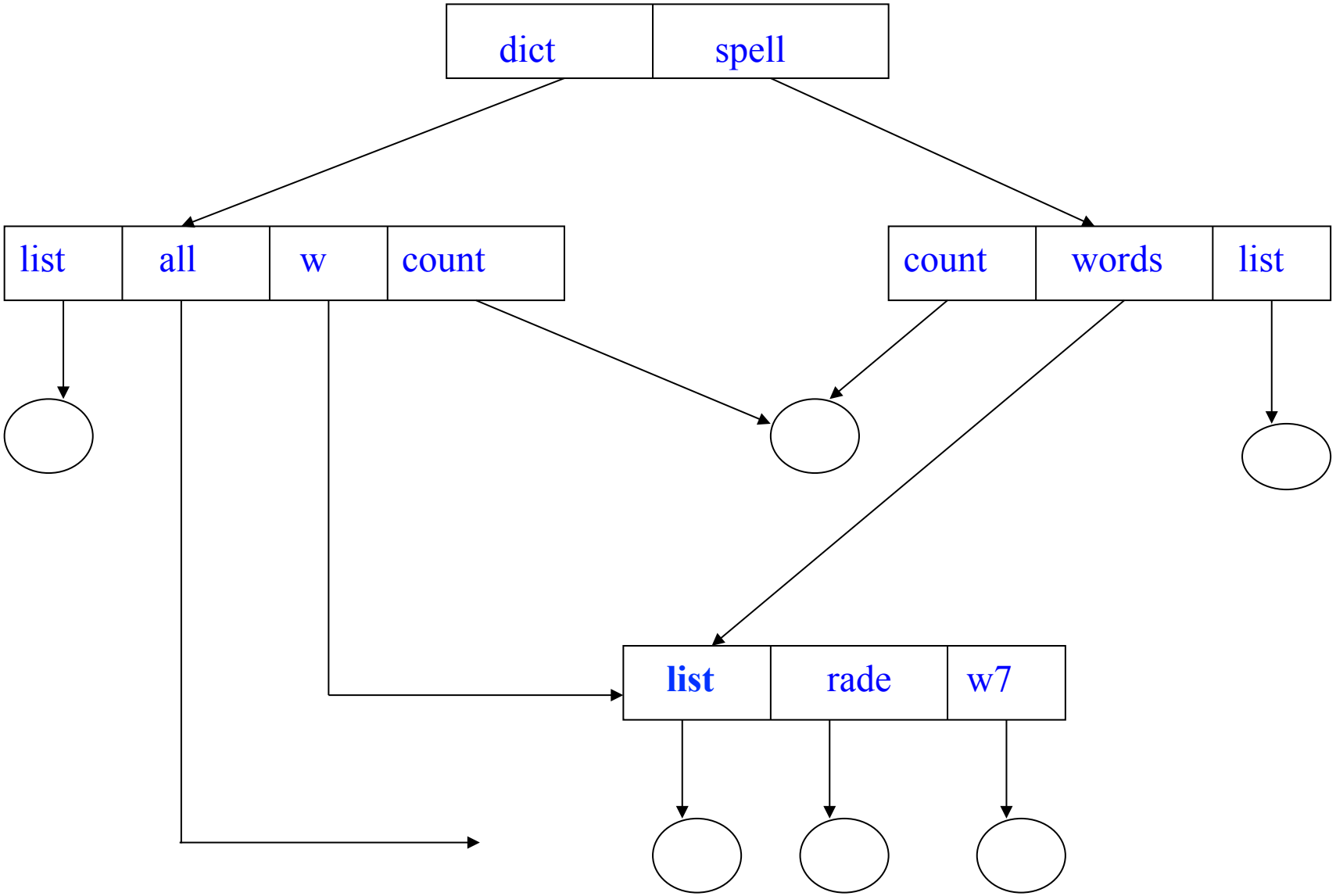


- Current directory is /spell/mail
  - **mkdir** count: to create a new directory
  - deleting 'mail' → deleting the entire subtree rooted by 'mail'.

Advantages of tree-structure:

- Efficient searching.
- Grouping capability – users can access files of other users.

Acyclic-graph directories



Acyclic-graph directories

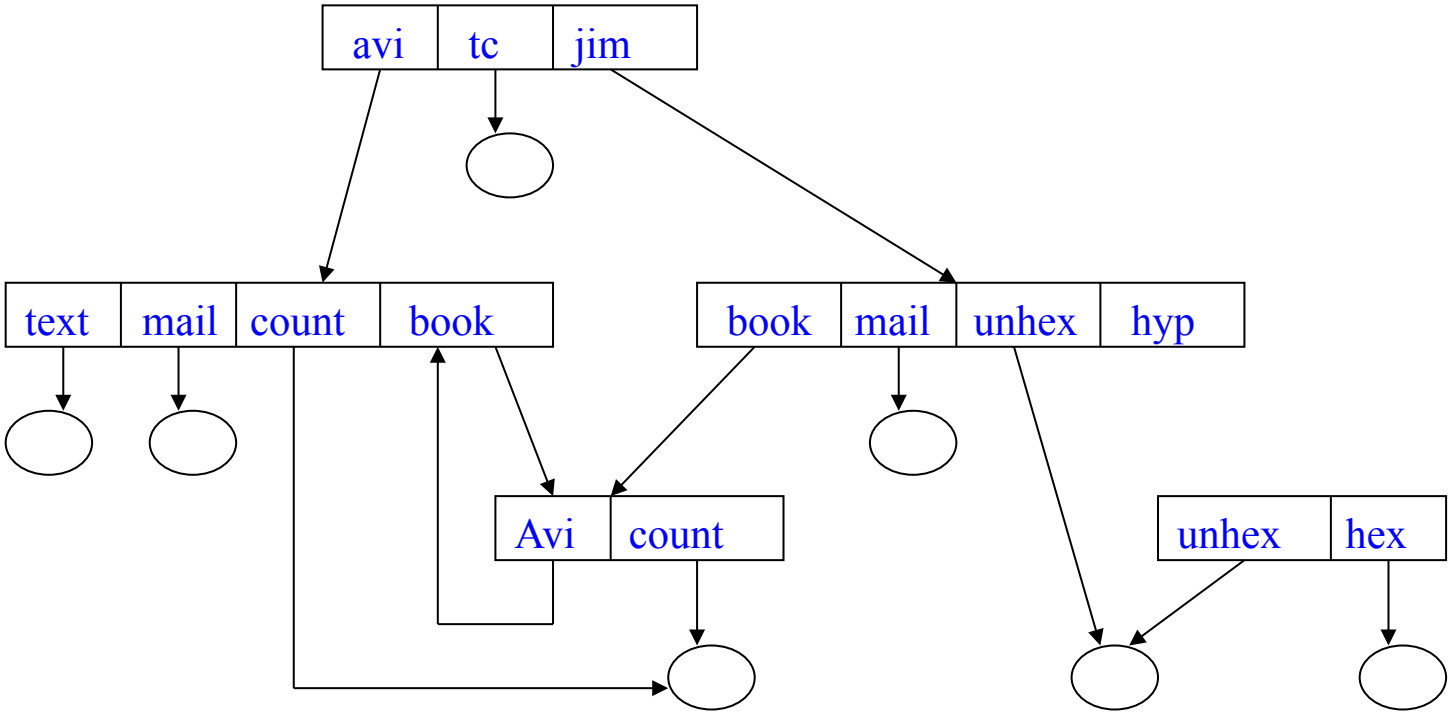
- An acyclic graph allows directories to have **shared subdirectories** and files.
  - The **same file or subdirectory** may be in two different directories.
  - **Tree structure prohibits** the sharing of files or directories.
  - An acyclic graph – a graph with no cycles – is a **generalization of a tree-structured directory**.
- Shared files or directories can be implemented:
  - Unix creates a new directory entry called *link* – a pointer to another file or subdirectory;

    OS ignores these links when traversing directory trees to preserve the acyclic structure of the system.
  - Duplicate all information about them in both sharing directories – problem on how to keep the two consistent if the file is modified.

Problems in Acyclic graph:

- A file may have several absolute path names.
  - Distinct file names may refer to the same file.
- Problem in deletion, e.g., if *dict* deletes *list* → dangling pointer.
  - Solutions: entry-hold-count solution

General Graph Directory



How do we guarantee no cycles?

- Allow only links to files (not subdirectories).
- Garbage collection → time consuming.
- Every time a new link is added use a cycle detection algorithm to determine if it is ok.

Protection

- File owner/creator should be able to control:
  - Type of accesses to the file and by whom.
- Access types:
  - read.
  - write.
  - execute.
  - append.
  - delete.
  - list.

Access Lists and Groups

- Access right depends on user identity
  - Access control list (ACL) gives user names and access rights for each user
- Three classes of users:
  - Owner.
  - Group.
  - Public.
- Type of accesses: read, write, execute.
- Can 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.

  chmod 761 *game*
- Attach a group to a file.

  chgrp *G* *game*

File System Implementation

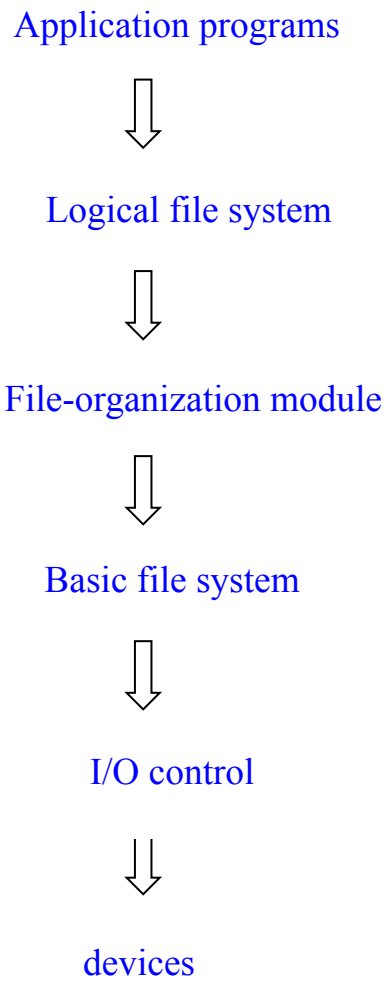
Design problems

- How the file system should appear to users.
- How to design data structure and algorithms to map logical addresses to physical addresses.

File System Structure

- File structure
  - Logical storage unit.
  - Collection of related information.
- File system resides on secondary storage (disks).
  - I/O transfers between memory and disk are performed in units of blocks.
  - A block is stored in one or more sectors.
  - Sector size varies from 32 bytes to 4096 bytes; usually 512 bytes.

File system is organized into layers



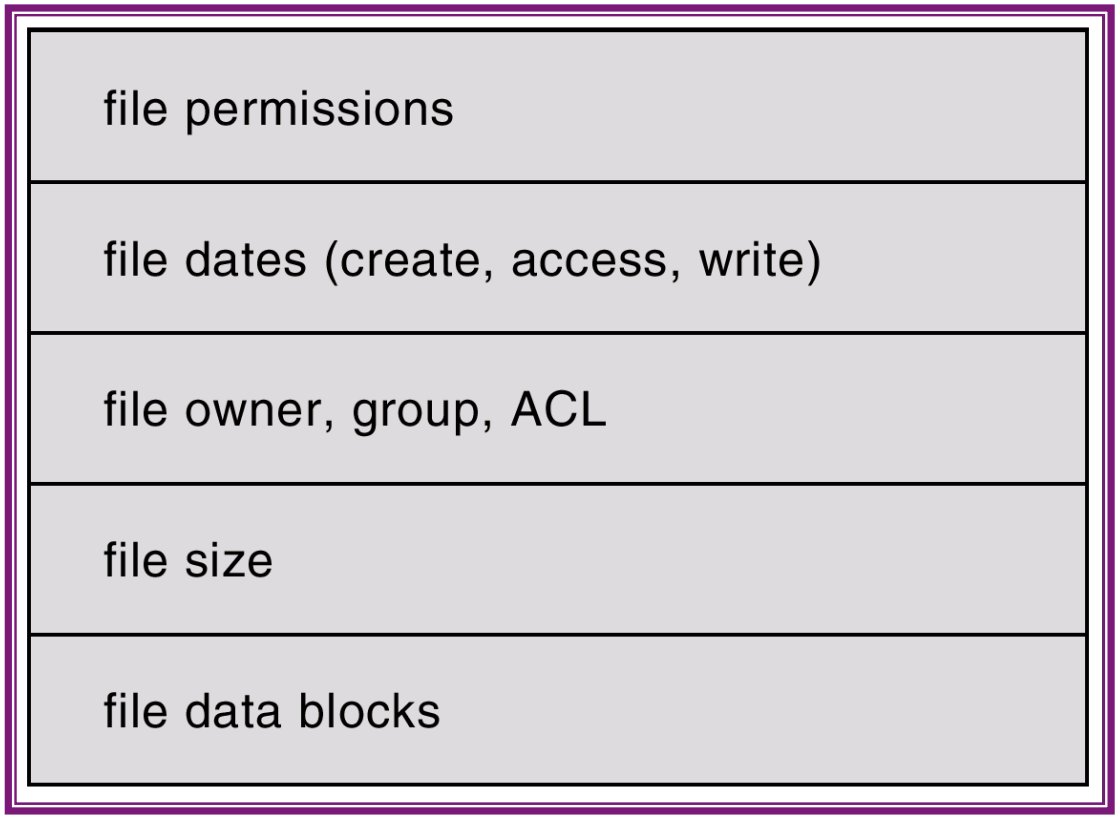
- **I/O control**: consists of device drivers and interrupt handlers to transfer information between memory and disk system.
- **Basic file system**: uses generic commands to the device driver to read/write physical blocks on the disk.
- **File organization module**: translates logical block addresses into physical block addresses.
- **Logical file system**: uses the directory structure to provide the file organization module with information given a symbolic file name.
- **Application programs**: creates a file, calls the logical file system.

Open file table

index	File name	permissions	Access dates	Pointer to disk block
0	test.c	rw rw rw	…	→
1	mail.txt	rw		→
2				
…	…	…	…	…
n				

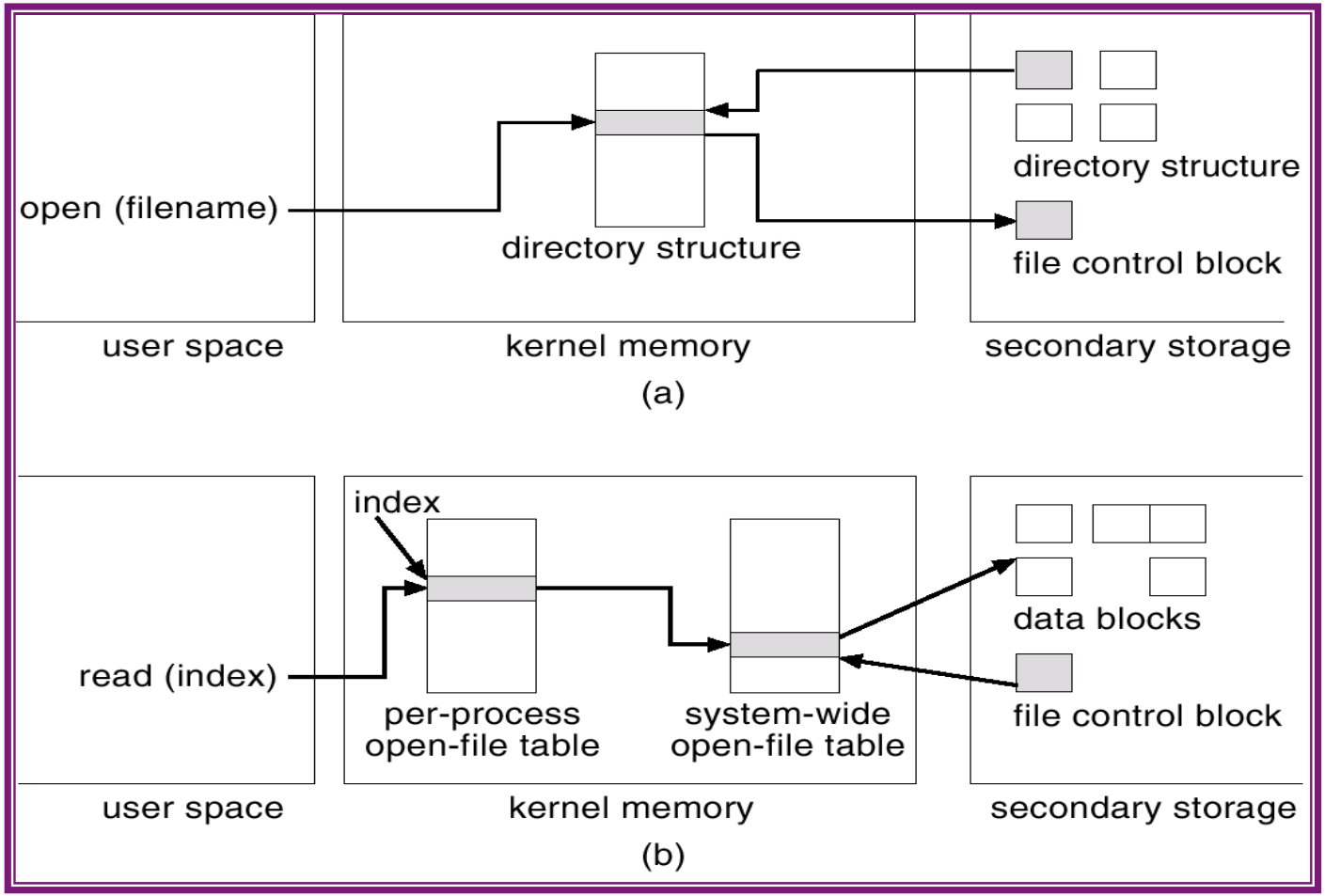
- Before a file can be accessed, it must be opened.
- The directory structure is searched for the desired file entry.
  - Parts of the directory structure are usually cached in memory (open file table) to speed up directory operations.
- *Open* will return an *index*
  - called *file descriptor* (UNIX), *file handle (*Windows), or *file control block* (FCB).
  - Use the index for further reference to the file
- FCB contains information about the file.
  - E.g., ownership, permissions, location of the file contents, etc.

A Typical File Control Block (FCB)



In-Memory File System Structures

- This figure shows the file system structures provided by OS.
  - Figure (a) refers to opening a file; Figure (b) refers to reading a file.



Directory Implementation

- Use *Linear list* of file names with pointers to the data blocks.
  - Require a linear search to find a particular entry.
  - Simple to program.
  - Disadvantage: Time-consuming to execute.

- Use *Hash table* – linear list with hash data structure.
  - Decrease directory search time.
  - *Collision* – situation where two file names hash to the same location.
  - Difficulties:
    - Fixed size of the hash table, and the dependence of the hash function on the size of the hash table.

Allocation Methods

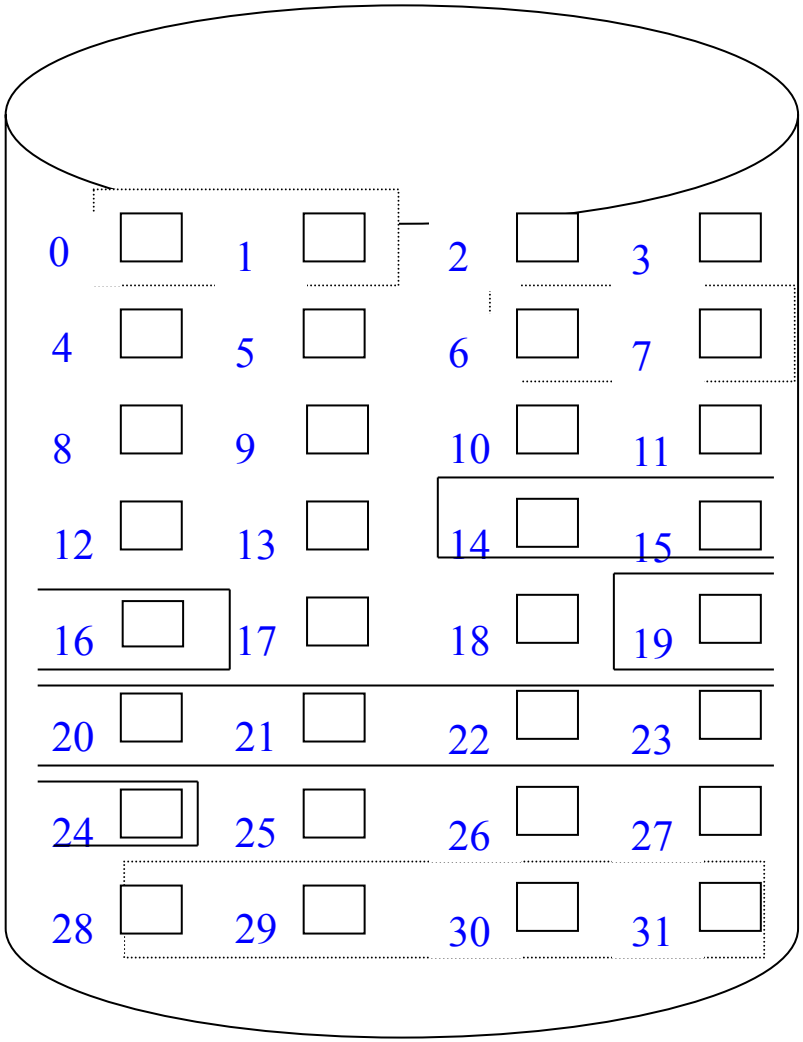
- How to allocate disk space (blocks) to files so that the space can be utilized effectively and files can be accessed quickly?

- Three major methods:
  - Contiguous.
  - Linked.
  - Indexed.

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 and kept in directory entry.
  - Support sequential and direct accesses.
- Disadvantages:
  - How to find the *best* space for a new file (dynamic storage-allocation problem).
    - How to satisfy a request of size *n* from a list of free blocks.
    - Use first-fit, best-fit, worst-fit algorithm → create external fragmentation.
    - Need disk compaction → expensive.
  - Files can not grow.
    - How much space is needed by a file?
- Mapping from logical address (LA) to physical:
  - Q = LA DIV 512, R = LA MOD 512.
    - Block to be accessed is the Q^{th} block; the first is block 0
    - Offset address: R
    - E.g., LA = 1000 is in block 1 (the second block), offset 488

Contiguous Allocation (cont.)



directory

file	start	length
A	0	2
B	14	3
C	19	6
D	28	4
E	6	2

Extent-based systems

- Extent-based systems allocate disk blocks in **extents**.
  - a modified contiguous allocation scheme.
- An **extent** is a contiguous block of disks.
  - Extents are allocated for file allocation.
  - A file consists of one or more extents.

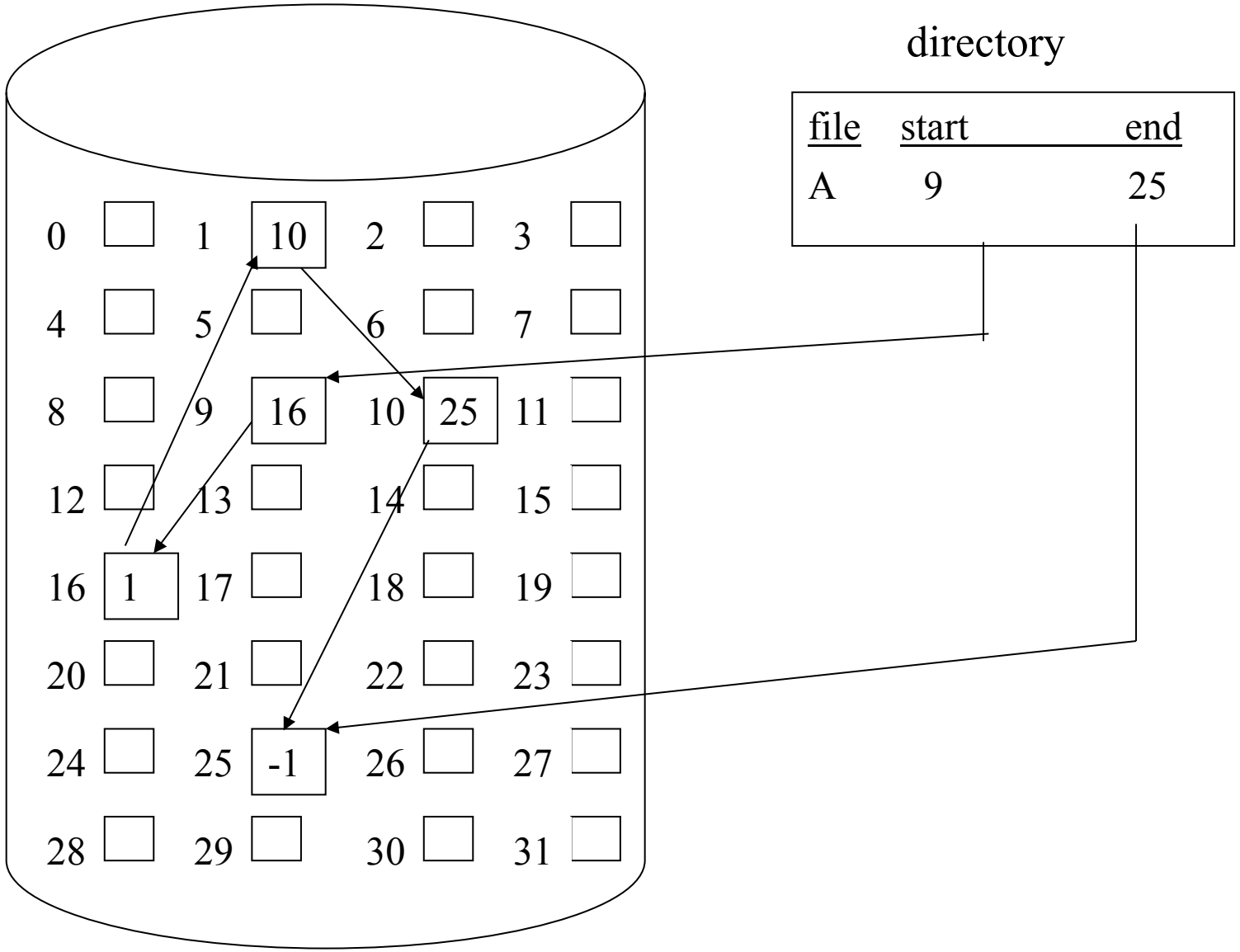
Link Allocation

- Each file is a linked list of disk blocks.
  - Allocate as needed, link together; e.g., file starts at block 9.
  - Blocks may be scattered anywhere on the disk.
  - Solve external fragmentation and size declaration problem of contiguous allocation.
- Each block contains a pointer to next block.
  - If block size = 512 bytes, and a pointer requires 4 bytes, users can use only 508 bytes.
- No external fragmentation, and files can grow as long as there are free blocks.
- Mapping: Q = LA DIV 508, R = LA MOD 508.
  - Block to be accessed is the Q^{th} block in the list.
  - Offset address: R

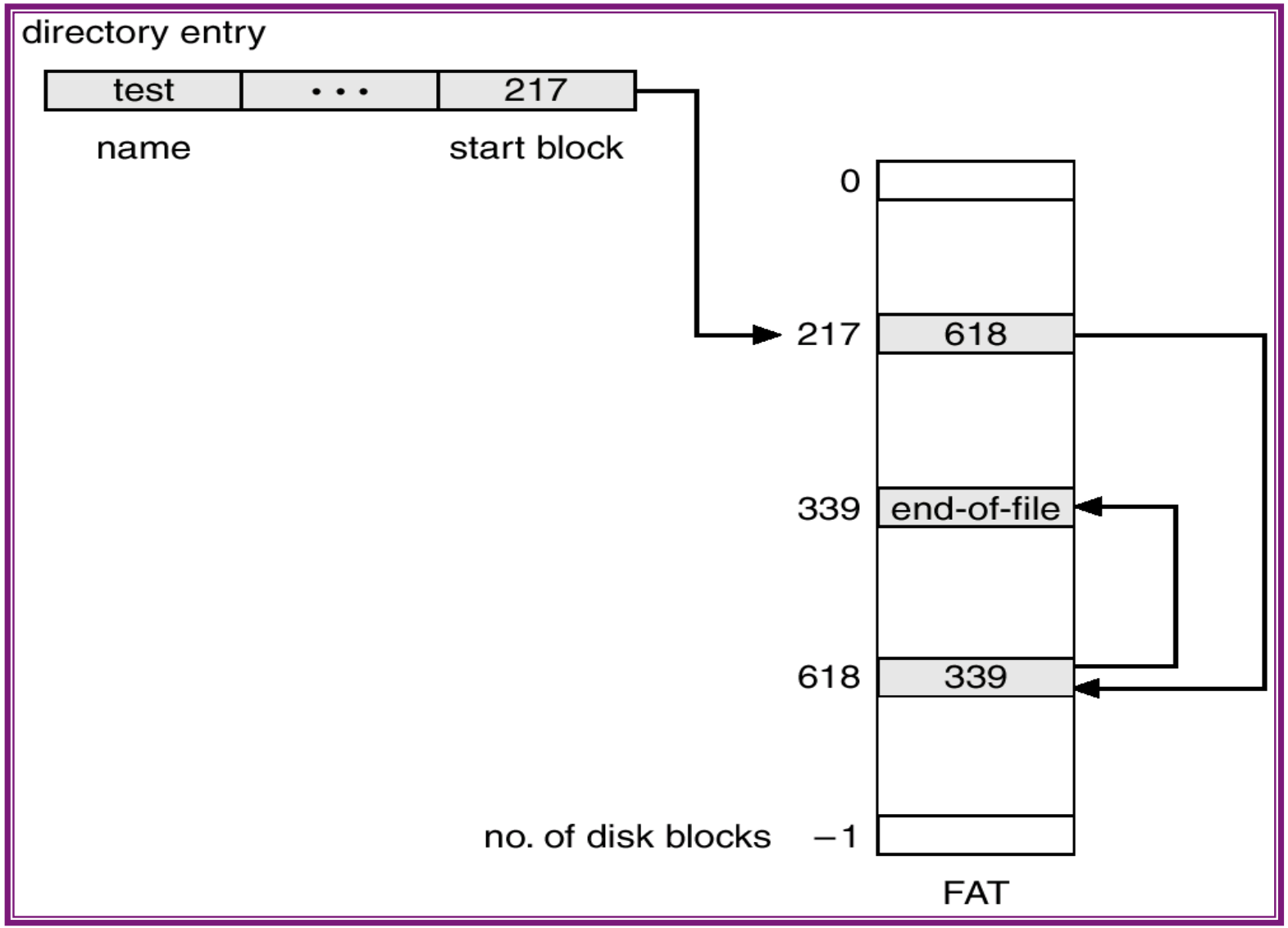
Link Allocation (cont.)

- Disadvantages:
  - Efficient only for sequential accesses.
    - For direct access it has to search from the beginning → not efficient.
  - Reliability issue: if a pointer is lost/damaged.
  - 4 of 512 bytes used for pointers.
    - Solution: use clusters of blocks; allocate clusters rather than blocks.
    - Problem: internal fragmentation.
- *File-allocation table* (FAT) - disk-space allocation used by MS-DOS and OS/2 is a variant of linked allocation.

Link Allocation (cont.)



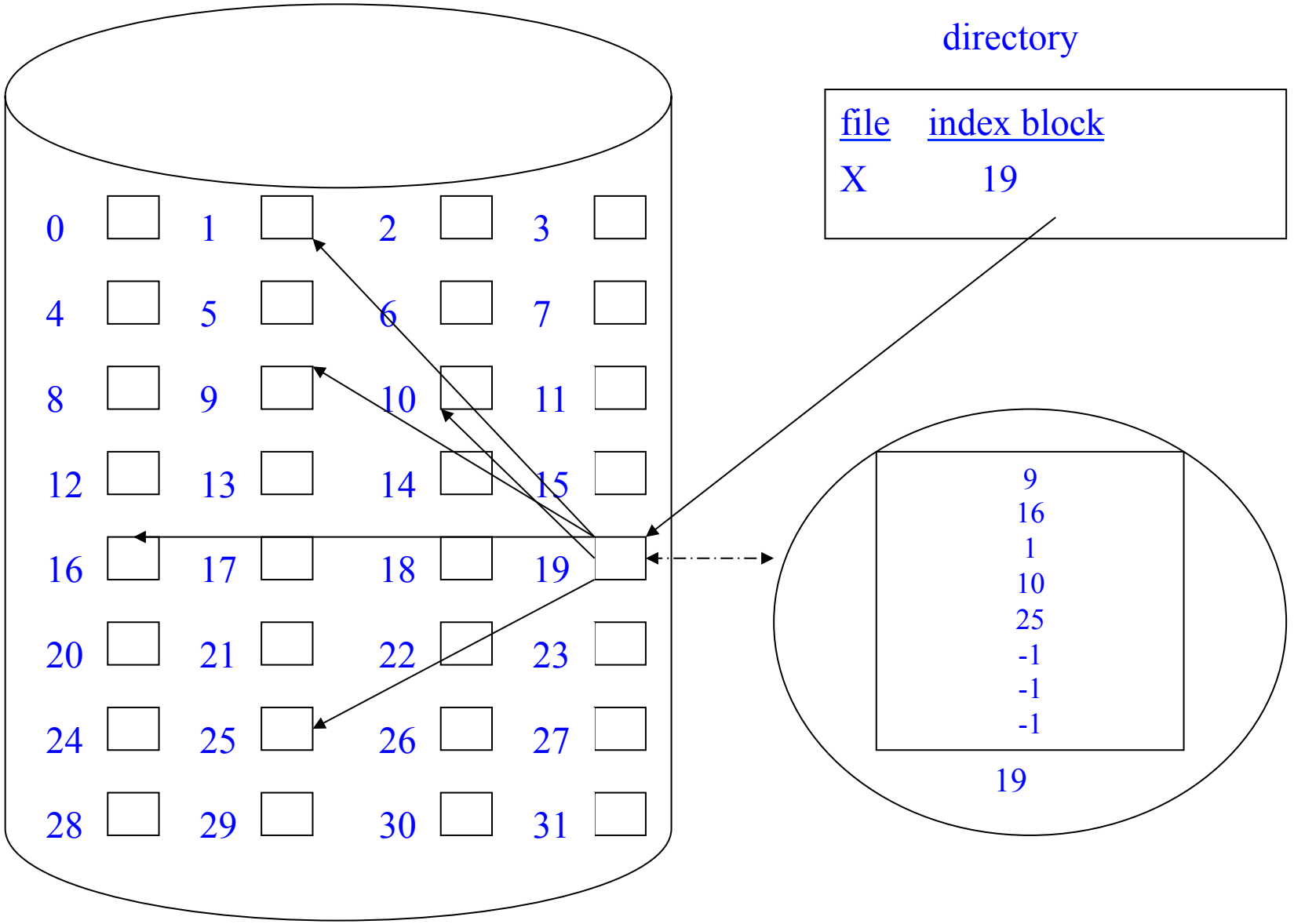
File-Allocation Table



Indexed Allocation

- Each file has its own index block – an array of disk block addresses.
  - It stores all pointers together in the *index block.*
  - *i*-th entry in the index block points to the *i*-th block of the file.
  - The directory contains the address of the index block.
  - An index block is typically one disk block.
  - Support direct accesses.
- It supports dynamic access without external fragmentation, but have overhead of the index block.
  - Must determine how large an index block is.
    - Too small, it cannot support large files
    - Too large, it will waste space.
    - For a large file, use a linked scheme, a multilevel index, or a combined scheme.

Indexed Allocation (cont.)



Indexed Allocation Mapping

- Mapping from logical to physical in a file of **maximum size** of 64K bytes and block size of 512 bytes.
  - Assume each pointer is 4 bytes → each index block contains 128 pointers

  Q = displacement into index table

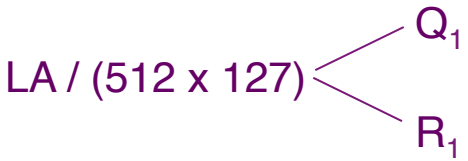
  R = displacement into block

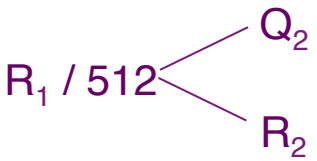
- Mapping from logical to physical in a file of **unbounded** length (block size of 512 bytes, 4 byte pointers).
  - **Linked scheme** – Link blocks of index table,
    - Each link block has 127 pointers, and thus can address up to 127 * 512 words.
    - e.g., 2 linked blocks: **max file size**: 2*127*512

    Q1 = block of index table

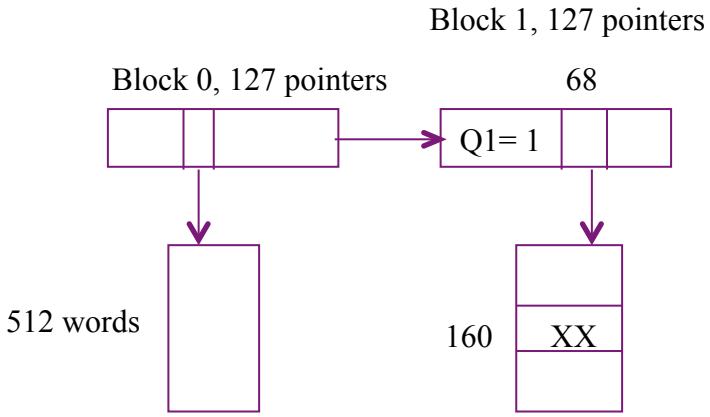
    Q2 = displacement into block of index table

    R2 = displacement into block of file





  - Example: LA = 100000
    - Q1 = 100000 DIV (512 * 127) = 1
    - R1 = 100000 MOD (512 * 127) = 34976
    - Q2 = 34976 DIV 512 = 68
    - R2 = 34976 MOD 512 = 160



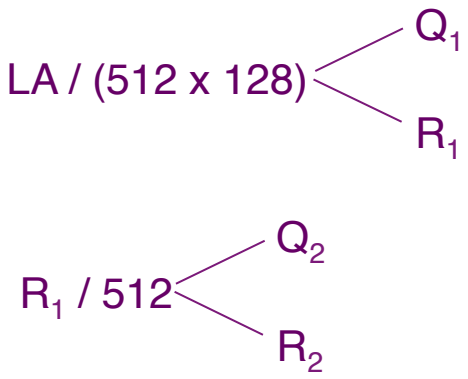
Indexed Allocation Mapping (cont.)

- Mapping from logical to physical in a file of **unbounded length** (block size of 512 bytes, 128 4-byte pointers).
  - **Two-level index** (**maximum file size** is $128^2 * 512$ bytes)

    Q1 = displacement into outer-index

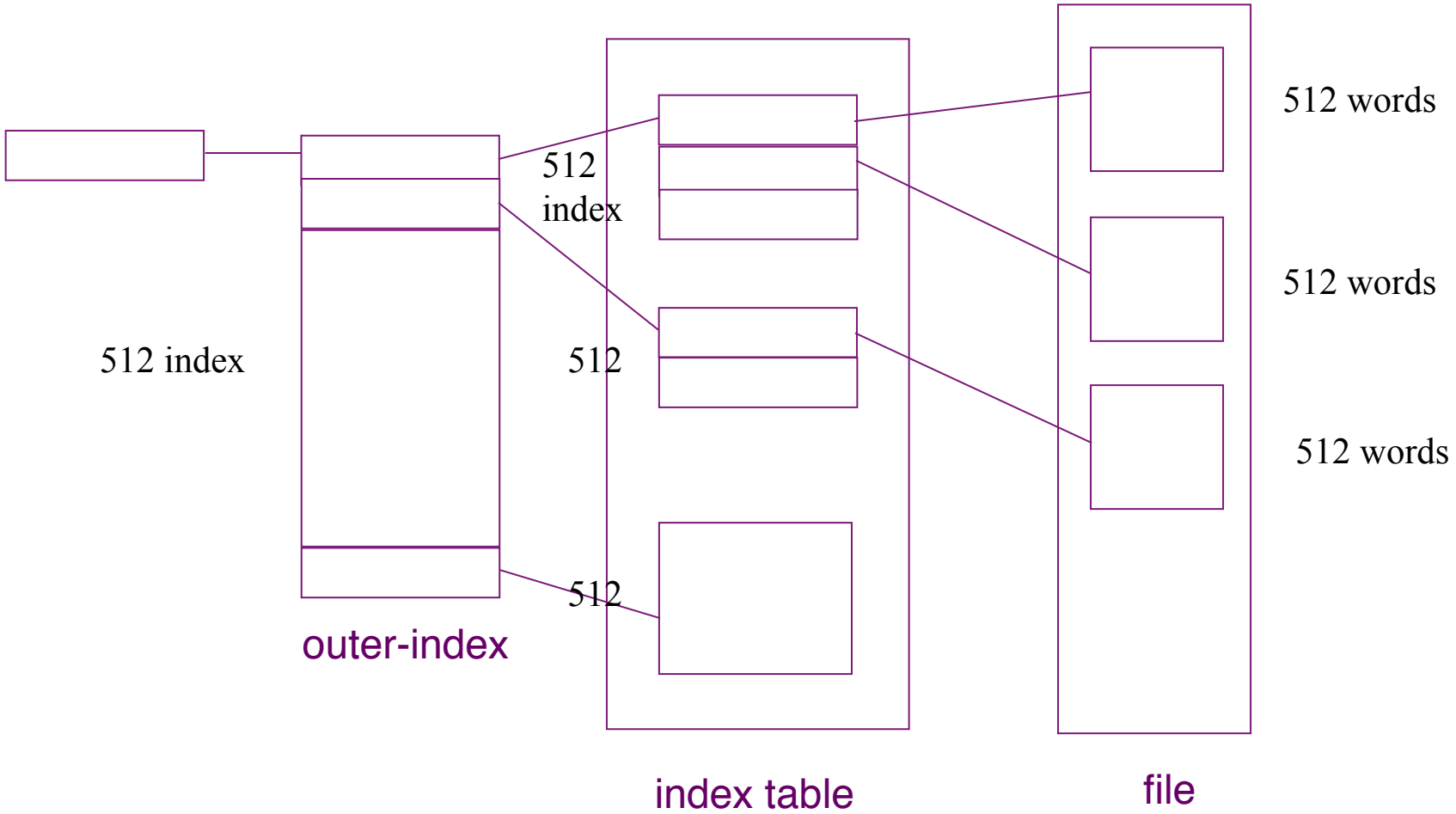
    Q2 = displacement into block of index table

    R2 = displacement into block of file

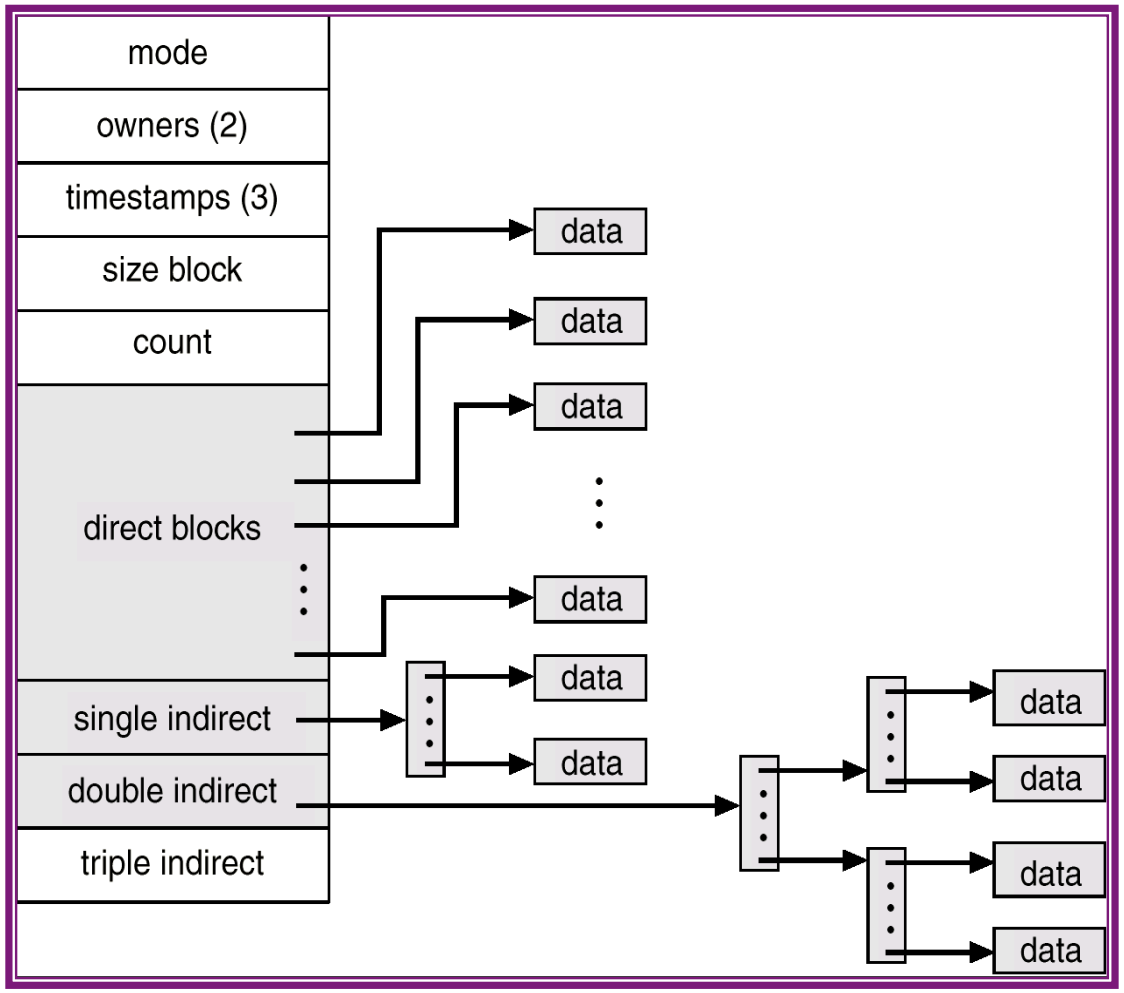


- Assume a system with block size = 4096 bytes = 4KB and 4 byte pointers
  - There are 4096/4 = 1024 pointers per index block
  - For two levels, there are 1024^2 pointers each of which can point to one data block of size 4KB
    - Thus, such configuration supports up to $1024^2 * 4KB = 4GB$ of file.

Two level index



Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- OS maintains a free space list to keep track of free disk space.
  - The space is limited → need to reuse the space from deleted files for new files, if possible.

- Several implementations:
  - Bit vector/ bit map.
  - Linked list.
  - Grouping.
  - Counting.

Bit vector (n blocks)

- Each block is represented by 1 bit: free (1), allocated(0)

Example:

- Free blocks: 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27. The free space bit map: 00111100111111000110000000111

Advantage:

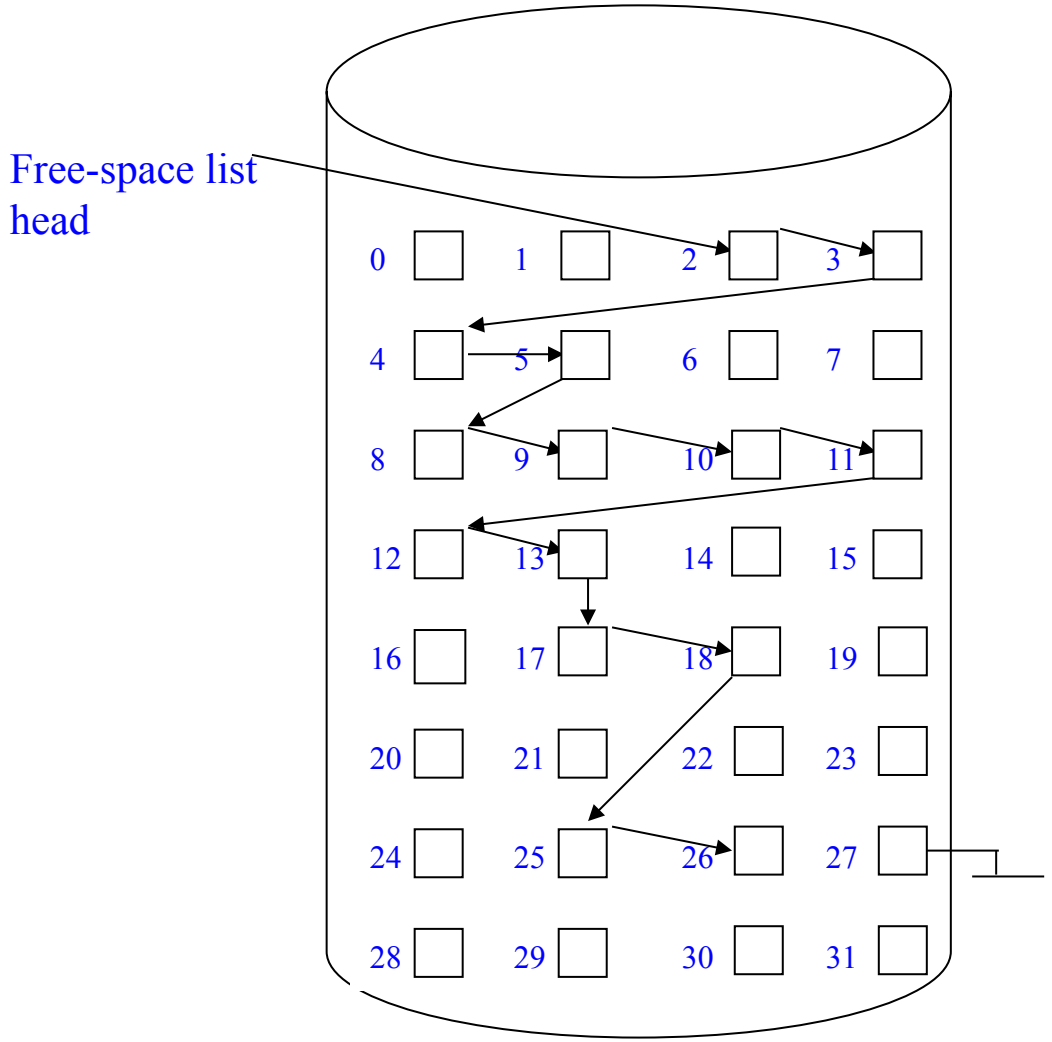
- Simple and efficient to find the first free block, or *n* consecutive free blocks on the disk.
- Many computer supply bit-manipulation instruction for this purpose.

Disadvantage:

- Bit map requires extra space.

Link List (free list)

- Link together all the free blocks, and keep a pointer to the first free block in a special location on the disk, and caching it in memory.
- The first block contains a pointer to next free disk block, and so on.
- Not efficient – to traverse the list, we must read each block – requires substantial I/O time.
- Cannot get contiguous space easily.



Grouping & Counting

Grouping

- A modified free list approach.
- Store the addresses of *n* free blocks in the first free block.
- The first *n*-1 of these blocks are actually free, and the last block contains the addresses of another *n* free blocks, etc.

Counting

- Generally several contiguous blocks may be allocated or freed simultaneously.
- 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 consists of a disk address and a count.

Efficiency and Performance

- Disks tend to be the major bottleneck in system performance – they are the slowest main computer component.
- Efficiency depends on:
  - Disk allocation and directory algorithms.
  - Types of data kept in file's directory entry.
- Needs techniques to improve the efficiency and performance of disks.
- Performance
  - *Disk cache* – separate section of main memory for frequently used block.
  - *Free- behind* and *read-ahead* – techniques to optimize sequential access.