Applied Operating System
Chapter 5: File Systems

Prepared By:
Amit K. Shrivastava
Asst. Professor
Nepal College Of Information Technology

Files

- File: a named collection of data that may be manipulated as a unit by operations such as:
  - Open, Close, Create, Destroy, Copy, Rename, List
- Individual data items within a file may be manipulated by operations like:
  - Read
  - Write
  - Update
  - Insert
  - Delete
- File characteristics include:
  - Location
  - Accessibility
  - Type
  - Volatility
- Files can consist of one or more records

File Naming

- Probably the most important characteristic of any abstraction mechanism is the way the objects being managed are named. When a process creates a file, it gives the file a name. When the process terminates, the file continues to exist and can be accessed by other processes using its name.
- The exact rules for file naming vary somewhat from system to system, but all current operating systems allow strings of one to eight letters as legal file names. Frequently digits and special characters are also permitted. *Many file systems support names as long as 255* characters.
- Some file systems distinguish between upper- and lower-case letters, whereas others do not. UNIX (including all its variants) falls in the first category; MS-DOS falls in the second.

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 appropriatecontrol characters
- Who decides:
  - -Operating system
  - -Program

File Attributes

- **Name**. The symbolic file name is the only information kept in human readable form.
- **Identifier**. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type.** This information is needed for systems that support different types of files.
- **Location**. This information is a pointer to a device and to the location of the file on that device.
- **Size.** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection**. Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification.** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk

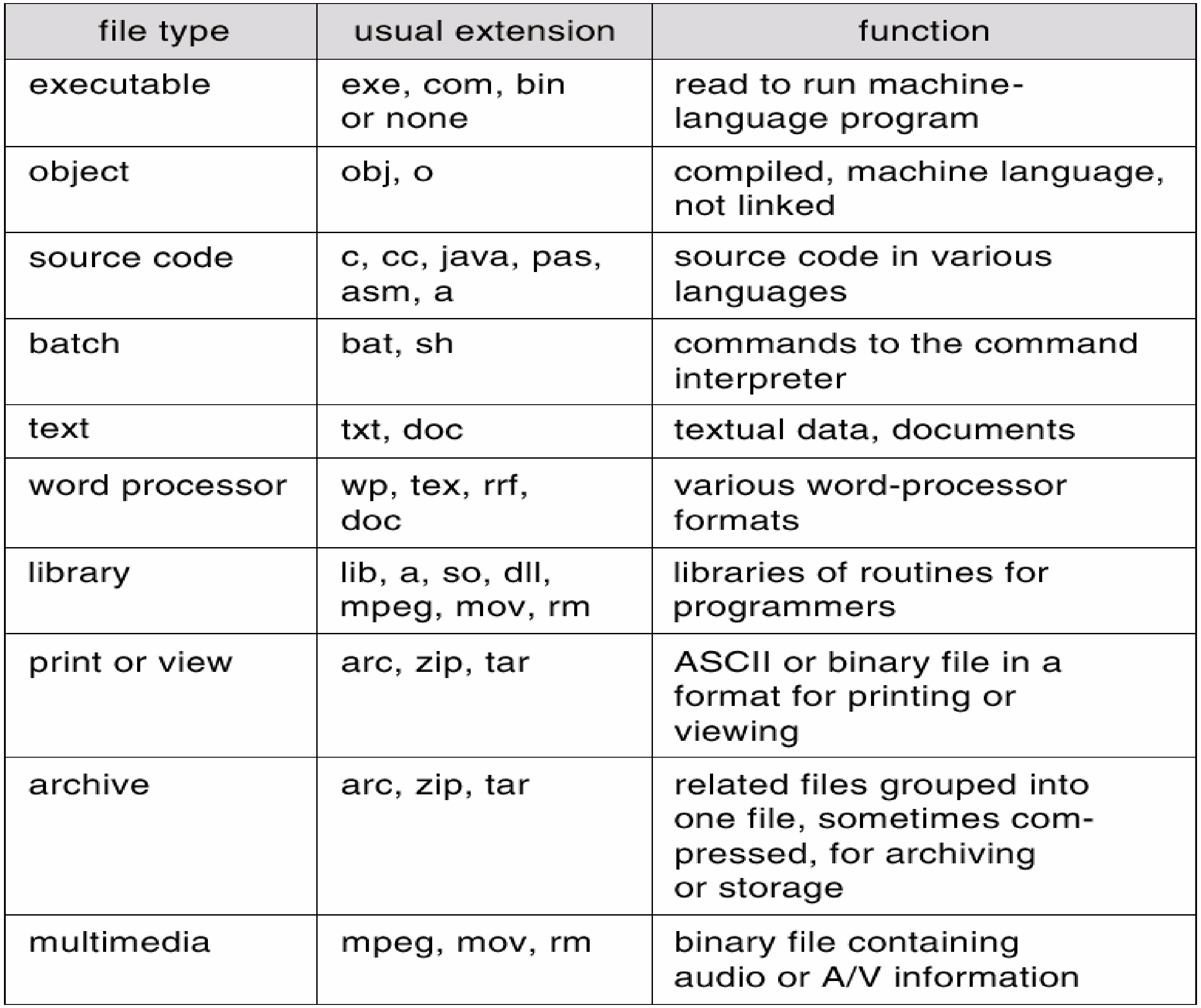
File Operations

- **Creating a file:** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** To write a file, we make a system call specifying both the name of the file and the information to be written to the file.
- **Reading a file**: To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.
- **Repositioning within a file:** The directory is searched for the appropriate entry, and the current-file-position is set to a given value. Repositioning within a file does not need to involve any actual I/O. This file operation is also known as a file *seek.*

File Operations(contd..)

- **Deleting a file:** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file**: The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged-except for file length-but lets the file be reset to length zero and its file space released.

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 compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

File Descriptor

- A file descriptor or file control block is a control block containing information the system needs to manage a file. It is a highly system-dependent structure. A typical file descriptor might include
  - Symbolic file name, location of file in secondary storage, file organization, access control data, type, disposition(permanent vs. temporary), creation date and time, destroy date, date and time last modified, access activity counts(number of reads, for example)
  - Ordinarily, file descriptor are maintained on secondary storage. They are brought to primary storage when a file is opened and is controlled by the operating system

File Access Methods

- **Sequential Access:** The simplest access method is **sequential access.** Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

  read next

  write next

  Reset

  no read after last write

  (rewrite)

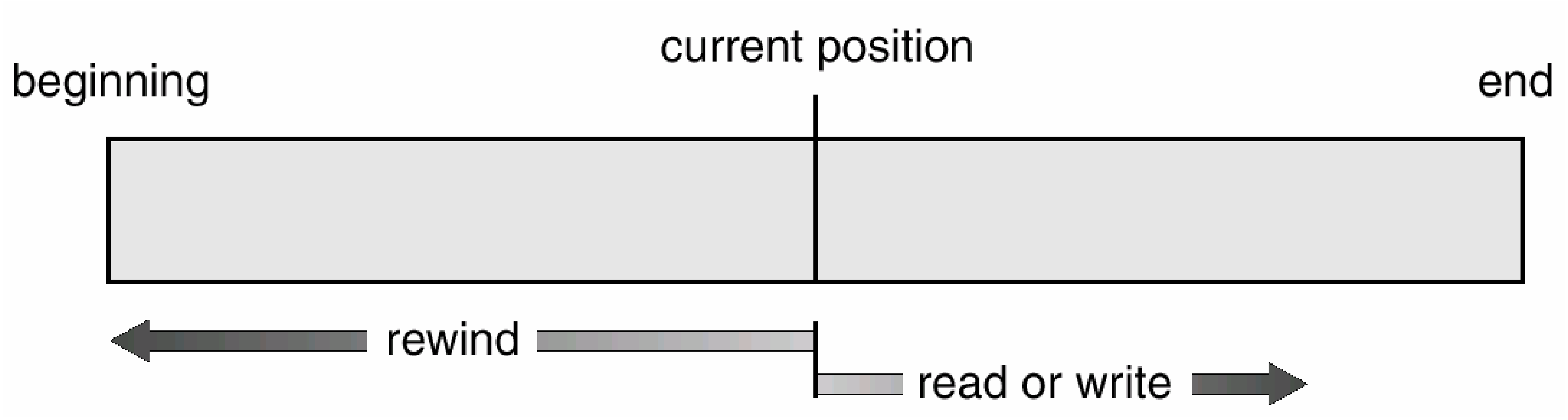
File Access Methods(contd..)

- **Direct Access:** Another method is direct access (or relative access). A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. A direct-access file allows arbitrary blocks to be read or written.

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

n= relative block number

Sequential-access File

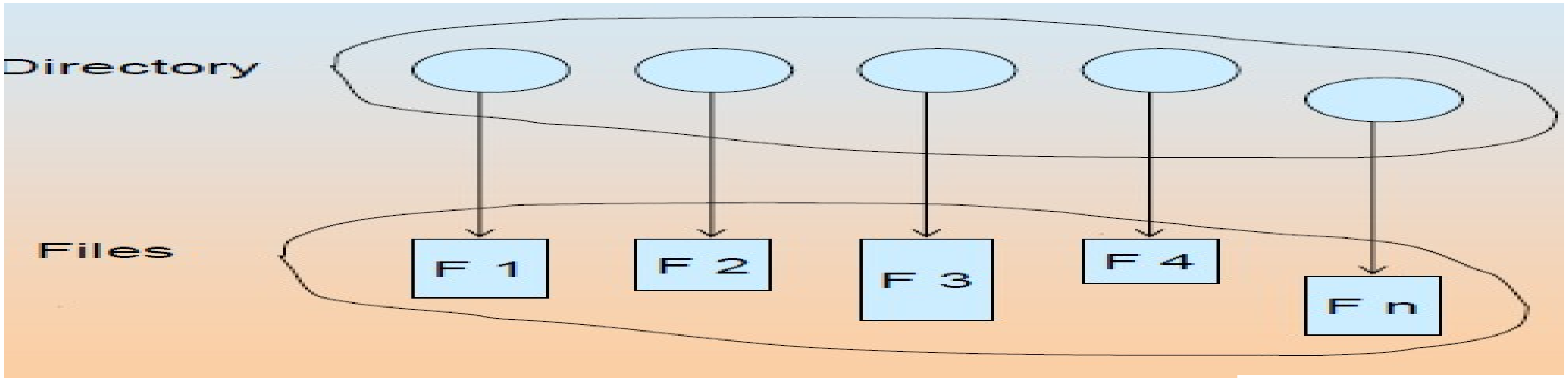


Simulation of Sequential Access on a Direct access File

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

Directory Structure

- Directories:
  - Files containing the names and locations of other files in the file system, to organize and quickly locate files
- Directory entry stores information such as:
  - File name
  - Location
  - Size
  - Type
  - Accessed
  - Modified and creation times



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

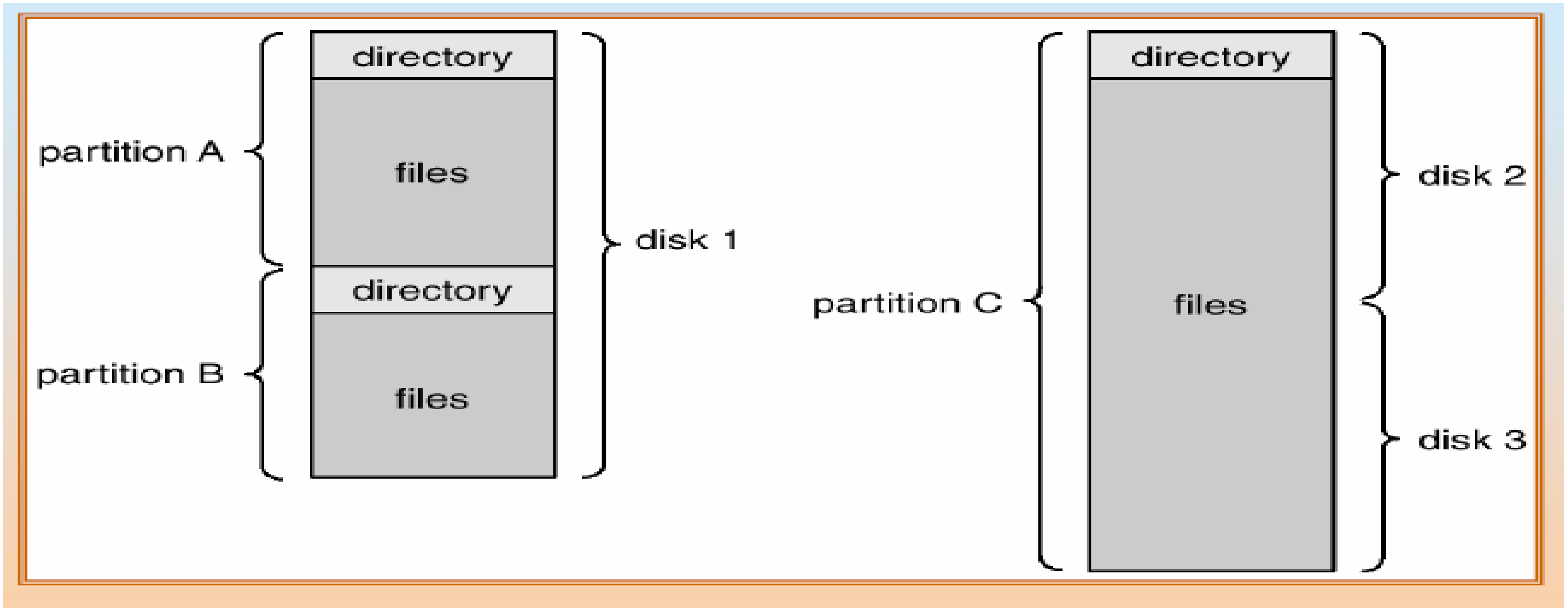


Fig: A typical file- system organization

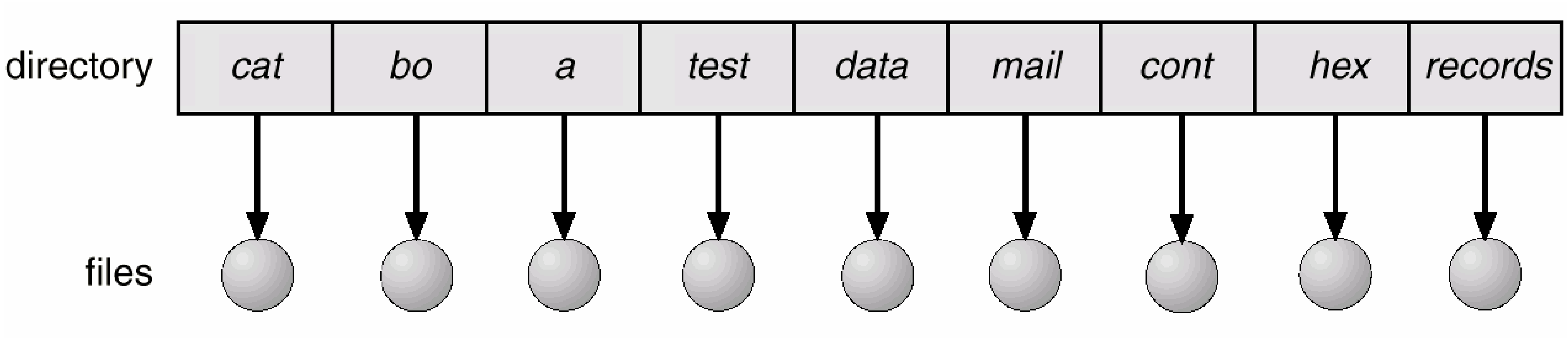
Directory Operations

- **Create a file**: New files need to be created and added to the directory.
- **Delete a file:** When a file is no longer needed, we want to remove it from the directory.
- **List a directory:** We need to be able to list the files in a directory, and the contents of the directory entry for each file in the list.
- **Rename a file:** Because the name of a file represents its contents to its users, the name must be changeable when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
- **Traverse the file system:** We may wish to access every directory, and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals.

Most common schemes for defining the logical structure of a directory are:

Single-level (or flat) directory:

- Simplest file system organization
- Stores all of its files using one directory
- No two files can have the same name
- File system must perform a linear search of the directory contents to locate each file, which can lead to poor performance



Hierarchical Directory

- A root indicates where on the storage device the root directory begins
- The root directory points to the various directories, each of which contains an entry for each of its files
- File names need be unique only within a given user directory
- The name of a file is usually formed as the pathname from the root directory to the file

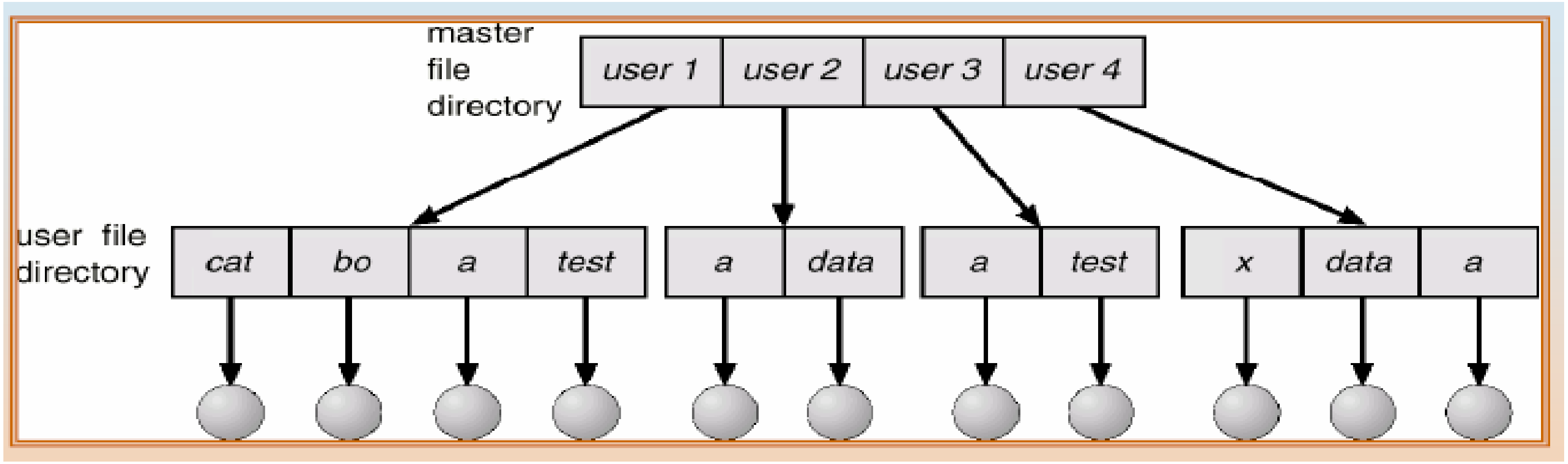


Figure : Two-level directory structure.

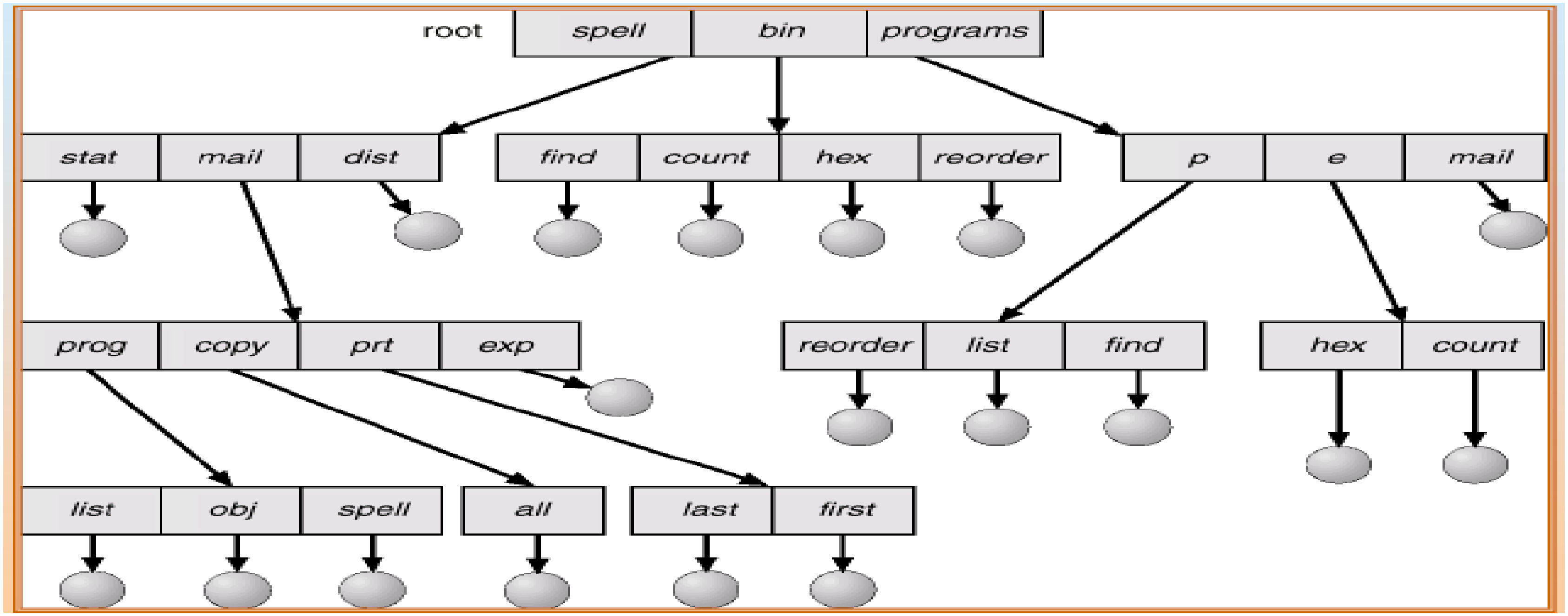


Fig: Tree structure directory

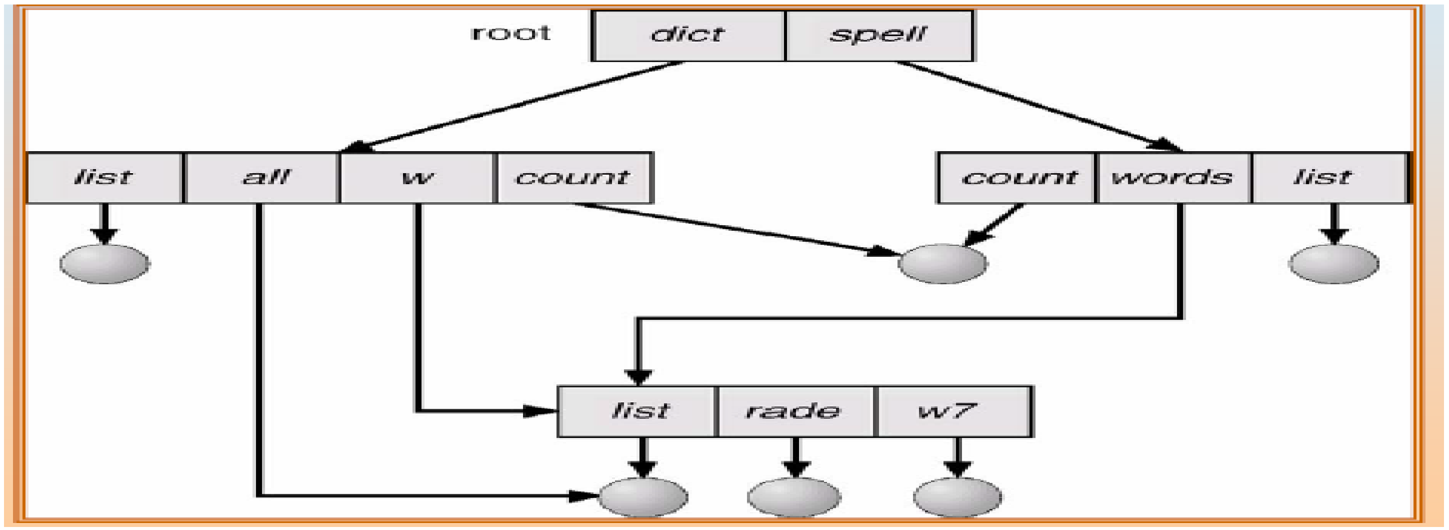


Fig: Acyclic-graph directory structure

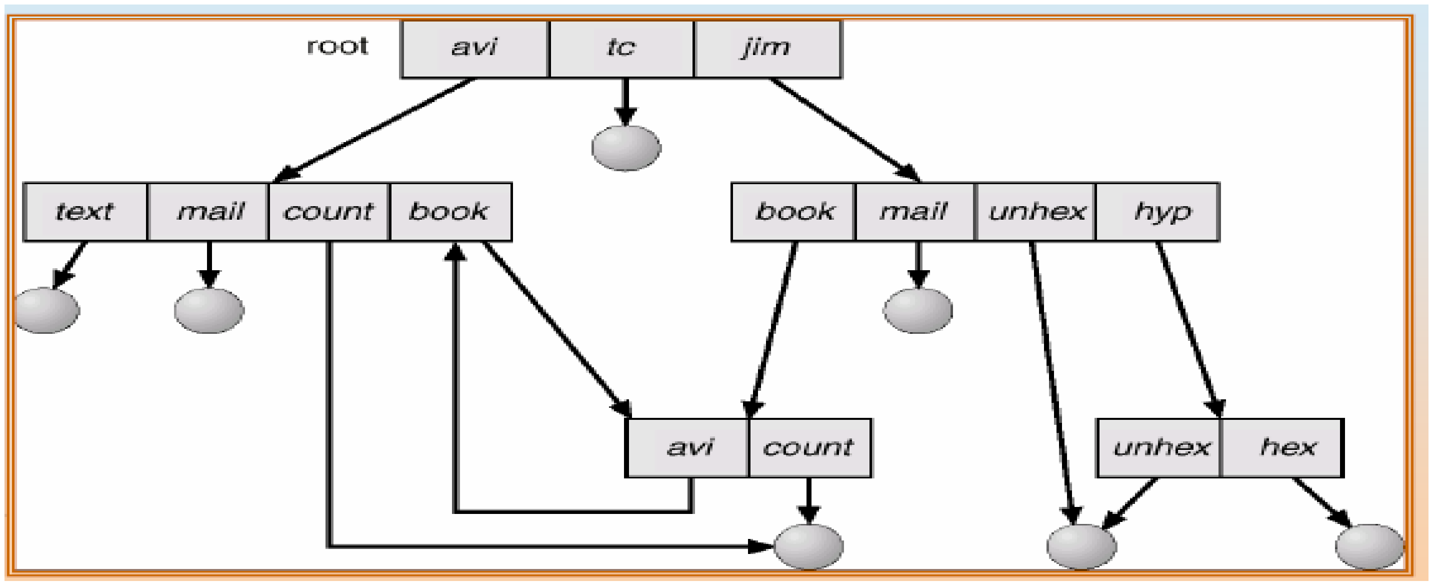


Fig: General graph directory

Protection

- When information is kept in a computer system, one major concern is **protection**, or guarding against improper access.
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read: Read from the file.
  - Write: Write or rewrite the file.
  - Execute: Load the file into memory and execute it.
  - Append: Write new information at the end of the file.
  - Delete: Delete the file
  - List: List the name and attributes of the file.

Access Lists and Groups

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

RWX

a) **owner access7⇒1 1 1**

RWX

b) **group access6⇒1 1 0**

RWX

c) **public access1⇒0 0 1**

ACL(Access Control List)

- The technique which consists of associating with each object an (ordered) list containing all the domains that may access the object, and how. This list is called the **Access Control List or ACL** and is illustrated in Fig. 1.
- When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.
- Here we see three processes, each belonging to a different domain. *A, B, and C, and three files F1, F2, and F3. For simplicity, we will assume that each domain* corresponds to exactly one user, in this case, users *A, B, and C. Often in the security literature,* the users are called **subjects or principals,** to contrast them with the things owned, the **objects,** such as files.

ACL(contd…)

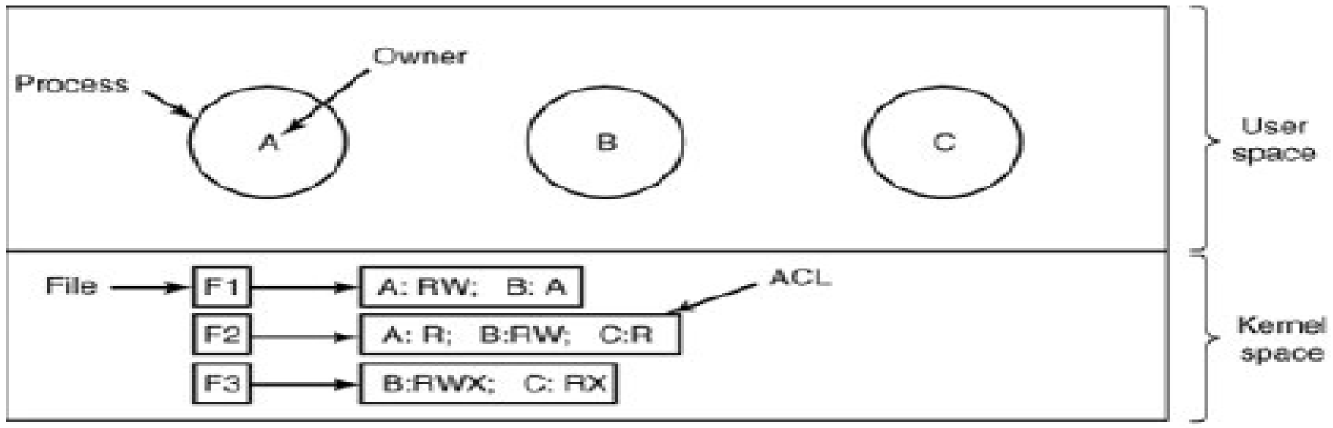


Figure 1. Use of access control lists to manage file access.

•Each file has an ACL associated with it. File *F1 has two entries in its ACL (separated by a* semicolon). The first entry says that any process owned by user *A may read and write the file.* The second entry says that any process owned by user *B may read the file. All other accesses by* these users and all accesses by other users are forbidden. Note that the rights are granted by user, not by process. As far as the protection system goes, any process owned by user *A can read* and write file *F1. It does not matter if there is one such process or 100 of them. It is the owner,* not the process ID, that matters. File *F2 has three entries in its ACL: A, B, and C can all read the file, and in addition B can also* write it. No other accesses are allowed. File *F3 is apparently an executable program, since B and C can both read and execute it. B can also write it.*

File-System Structure

- The file system structure is the most basic level of organization in an operating system. The way an operating system interacts with its users, applications, and security model nearly always depends on how the operating system organizes files on storage devices. Providing a common file system structure ensures users and programs can access and write files.
- File systems break files down into two logical categories:
  - Shareable vs. unsharable files
  - Variable vs. static files
- Shareable files can be accessed locally and by remote hosts; unsharable files are only available locally.
- Variable files, such as documents, can be changed at any time; static files, such as binaries, do not change without an action from the system administrator.

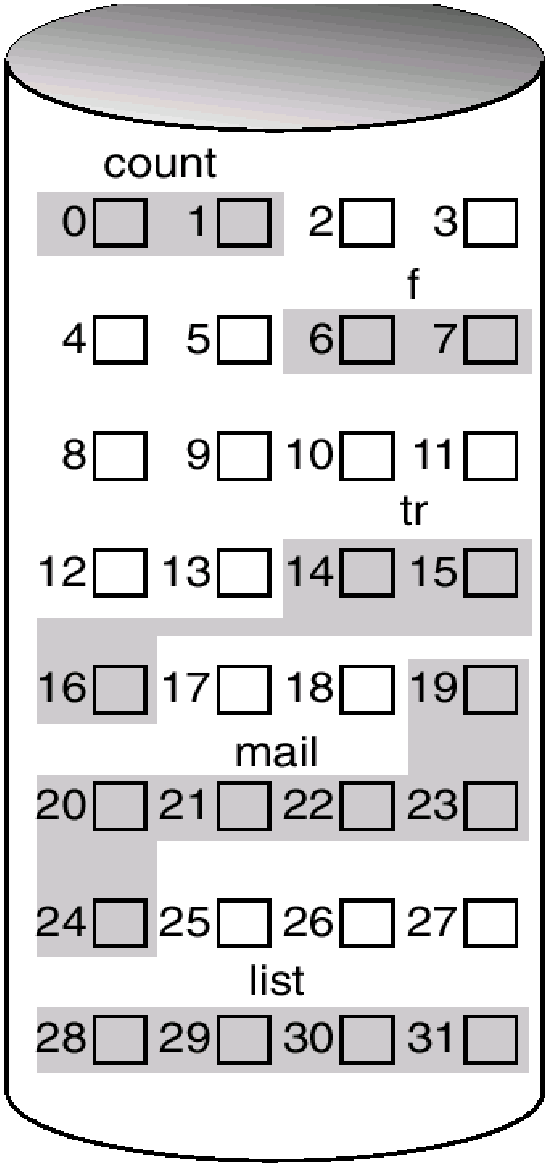
Methods of Allocation

- File allocation
  - Problem of allocating and freeing space on secondary storage is somewhat like that experienced in primary storage allocation under variable-partition multiprogramming
- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

- The **contiguous-allocation** method requires each file to occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block b + 1 after block b normally requires no head movement. When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), it is only one track. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed. It access randomly and is wasteful of sapce.

Contiguous Allocation of Disk Space



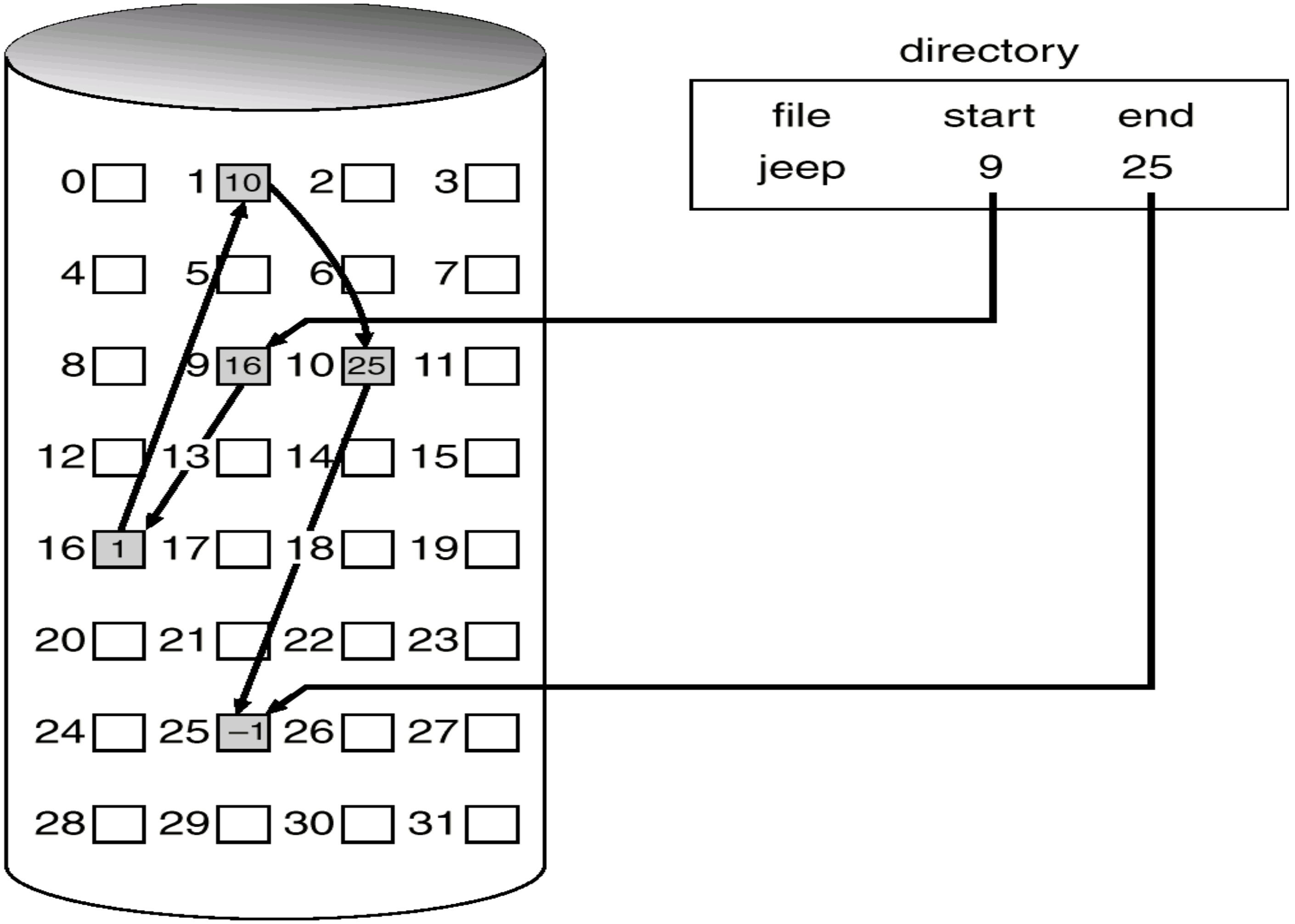
directory

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

Linked List Allocation

- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally block 25 (Figure 12.6). Each block contains a pointer to the next block. These pointers are not made available to the user. It does not access randomly and provide free space management system so there is no waste of space.

Linked Allocation



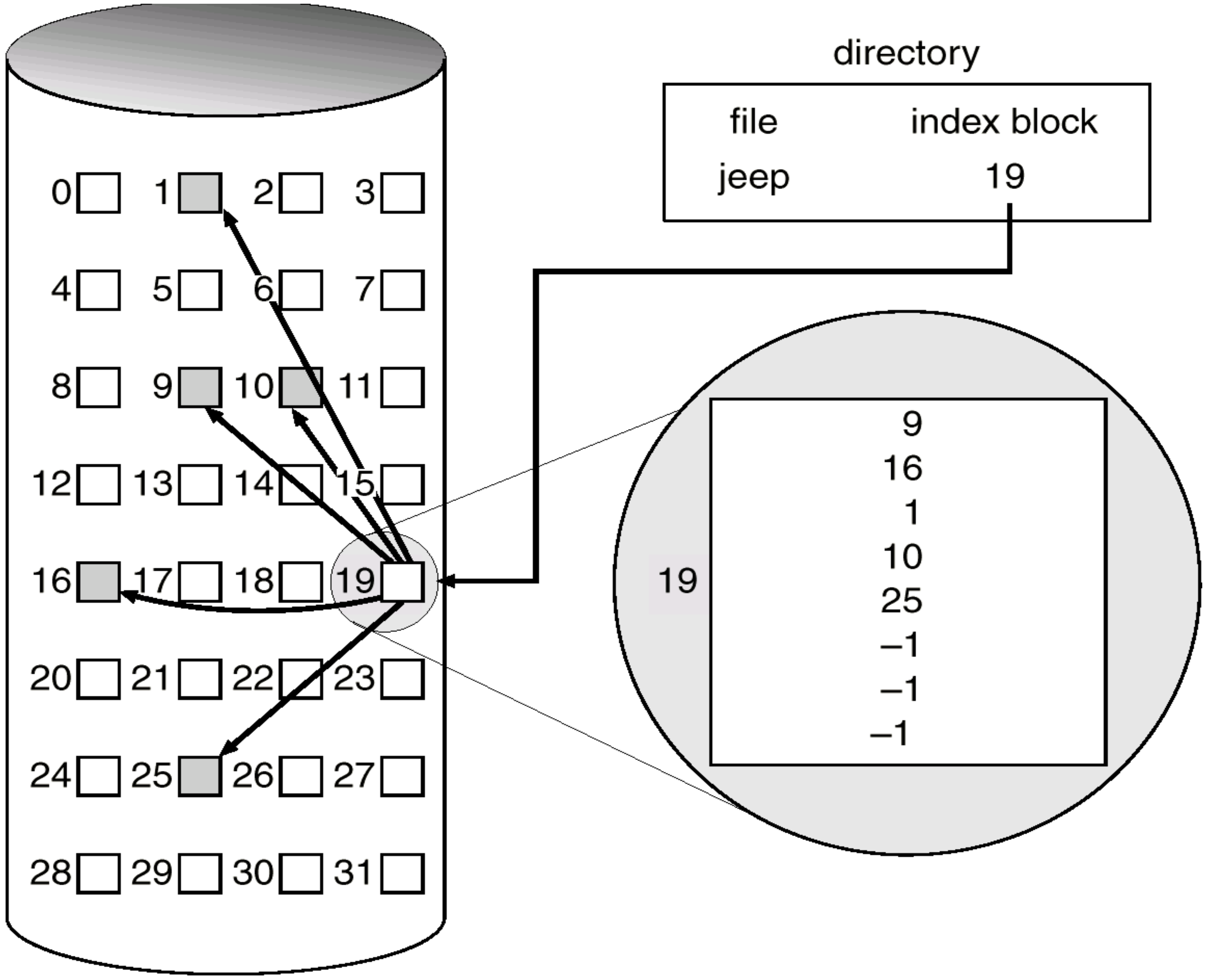
Index Allocation

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

$$\begin{tikzcd} & Q \\ LA/512 \arrow[ur, no head] \arrow[dr, no head] & \\ & R \end{tikzcd}$$

- Q = displacement into index table
- R = displacement into block

Example of Indexed Allocation



Free-Space Management

- Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list. The free-space list records all disk blocks that are free - those not allocated to some file or directory. To create a file, the free-space list has to be searched for the required amount of space, and allocate that space to a new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Bit-Vector

- The free-space list is implemented as a bit map or bit vector. Each block is represented by a 1 bit. If the block is free, the bit is 0; if the block is allocated, the bit is 1.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free-space bit map would be: 11000011000000111001111110001111…
- The main advantage of this approach is that it is relatively simple and efficient to find n consecutive free blocks on the disk.

Free-Space Management(contd..)

Linked List

- Another approach is to link all the free disk blocks together, keeping a pointer to the first free block. This block contains a pointer to the next free disk block, and so on. In the previous example, a pointer could be kept to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on. This scheme is not efficient; to traverse the list, each block must be read, which requires substantial I/O time.

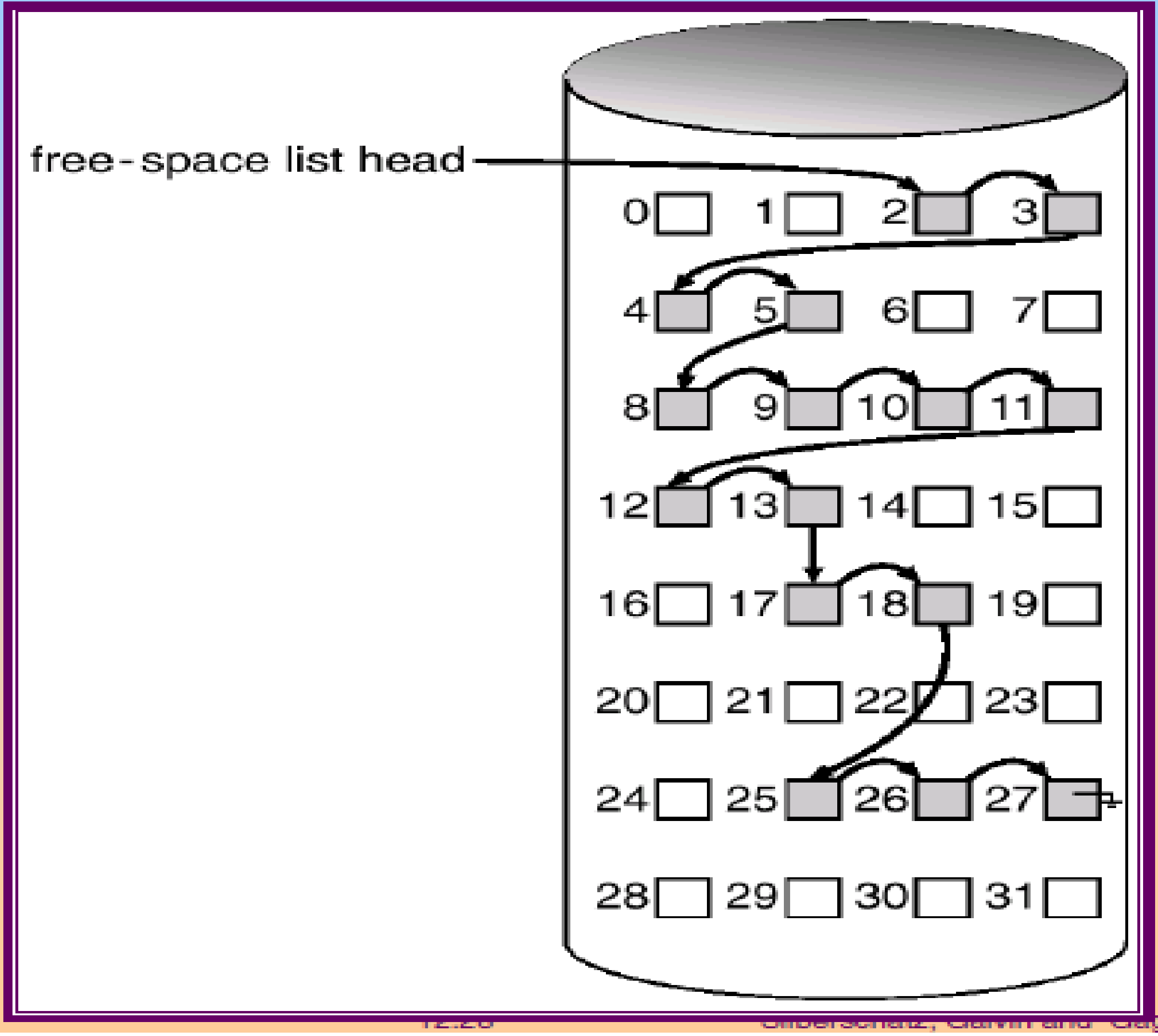
Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these are actually free. The last one is the disk address of another block containing addresses of another n free blocks. The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

Counting

- Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when contiguous allocation is used. Thus, rather than keeping a list of free disk addresses, the address of the first free block is kept 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. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

Linked Free Space List on Disk



Directory Implementation

- Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space.

Linear List

- A linear list is the simplest and easiest directory structure to set up, but it does have some drawbacks.
- Finding a file (or verifying one does not already exist upon creation) requires a linear search.
- Deletions can be done by moving all entries, flagging an entry as deleted, or by moving the last entry into the newly vacant position.
- Sorting the list makes searches faster, at the expense of more complex insertions and deletions.
- A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.
- More complex data structures, such as B-trees, could also be considered.

Hash Table

- A hash table can also be used to speed up searches.
- Hash tables are generally implemented ***in addition to*** a linear or other structure.
- Only good if entries are fixed size, or use chained-overflow method.

Recovery

Consistency Checking

- compares data in directory structure with data blocks on disk, and tries to fix inconsistencies . Can be slow and sometimes fails
- The allocation and free-space management algorithms dictate what types of problems the checker can find, and how successful it will be in fixing those problems. For instance, if linked allocation is used and there is a link from any block to its next block, then the entire file can be reconstructed from the data blocks, and the directory structure can be recreated.

Backup and Restore

- In order to recover lost data in the event of a disk crash, it is important to conduct backups regularly. Files should be copied to some removable medium, such as magnetic tapes, floppy disk or optical disk.
- Recovery from the loss of an individual file, or of an entire disk, may then be a matter of **restoring** the data from backup.
- To minimize the copying needed, we can use information from each file's directory entry.