6

FILE SYSTEMS

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Fig. 6-1. Some typical file extensions.

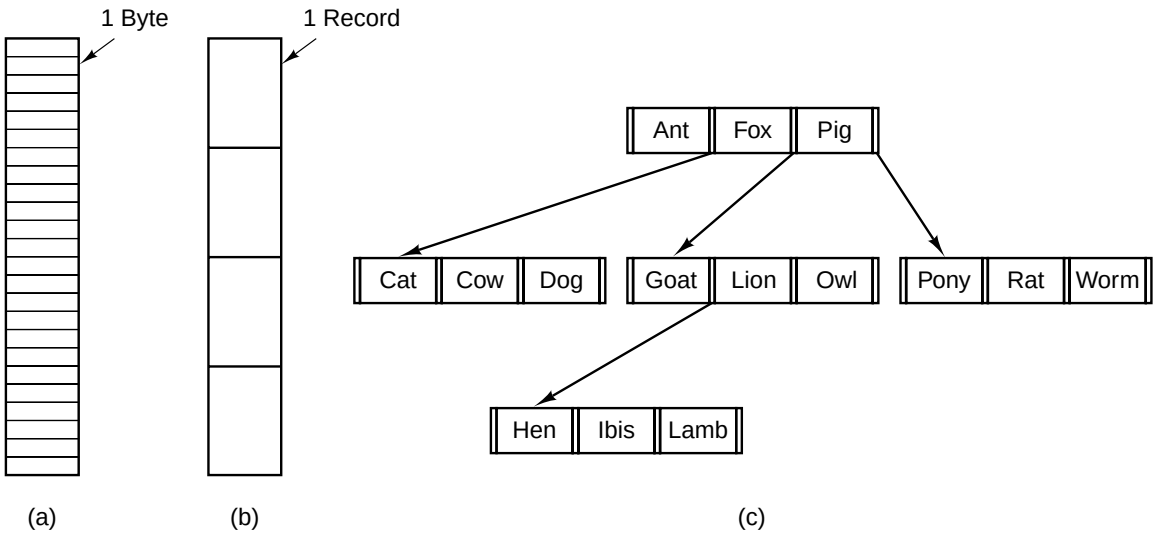

Fig. 6-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

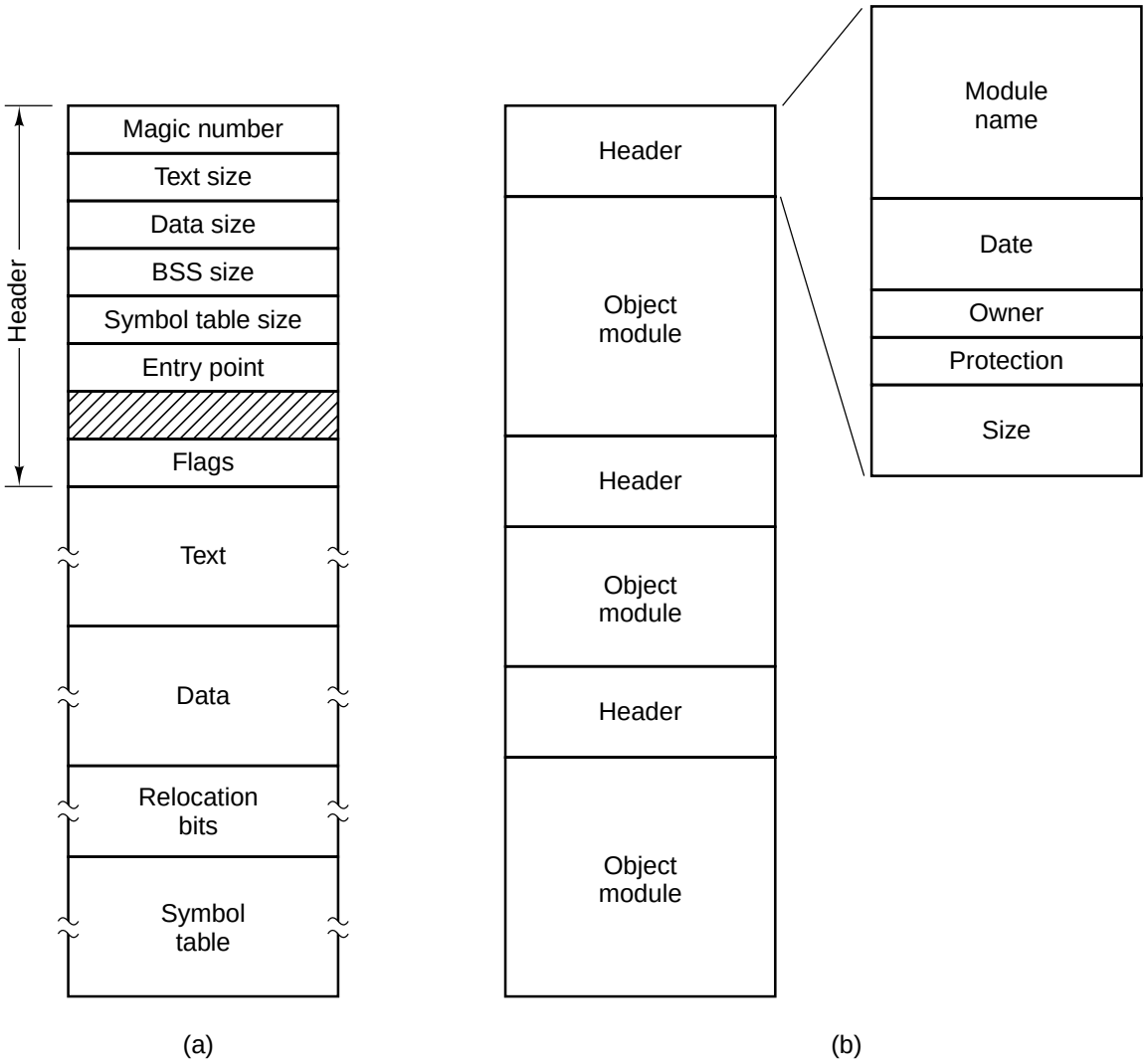

Fig. 6-3. (a) An executable file. (b) An archive.

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Fig. 6-4. Some possible file attributes.

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);    /* open the source file */
    if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE);   /* create the destination file */
    if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE);  /* read a block of data */
    if (rd_count <= 0) break;           /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count);  /* write data */
        if (wt_count <= 0) exit(4);     /* wt_count <= 0 is an error */
    }

    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                  /* no error on last read */
        exit(0);
    else
        exit(5);                        /* error on last read */
}
```

Fig. 6-5. A simple program to copy a file.

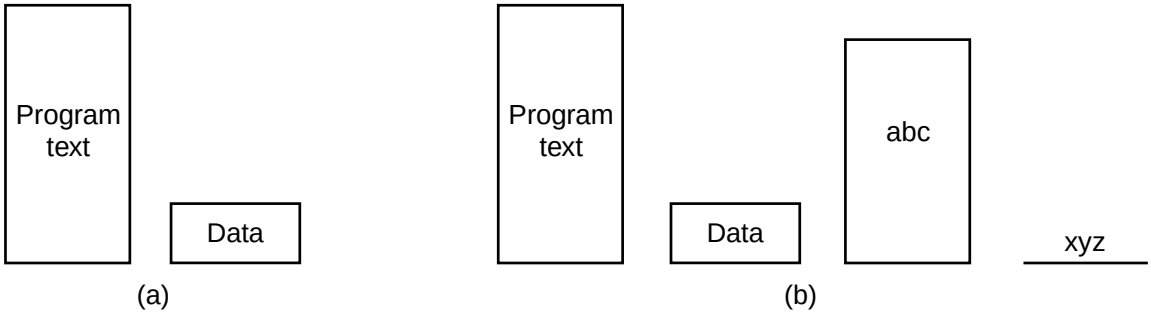

Fig. 6-6. (a) A segmented process before mapping files into its address space. (b) The process after mapping an existing file *abc* into one segment and creating a new segment for file *xyz*.

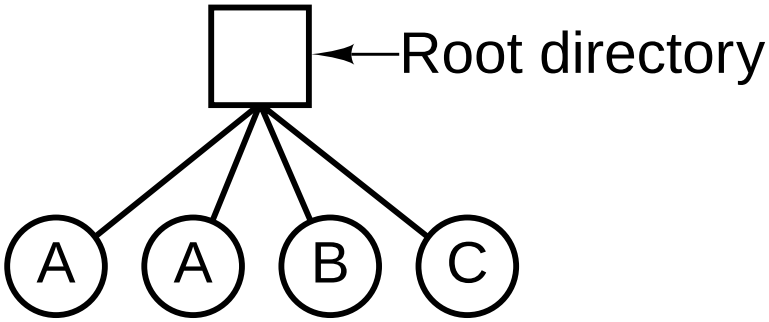

Fig. 6-7. A single-level directory system containing four files, owned by three different people, *A*, *B*, and *C*.

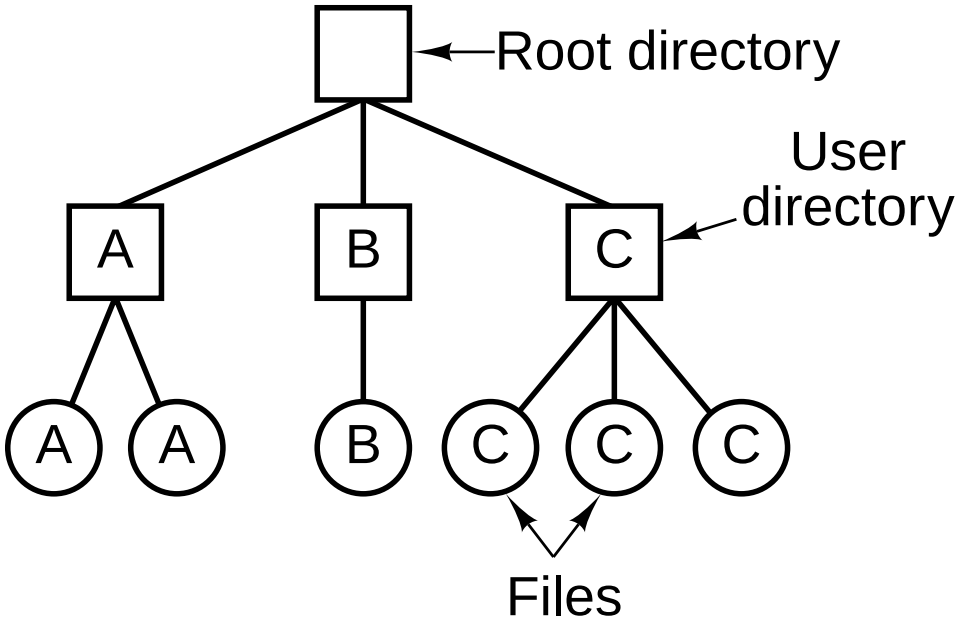

Fig. 6-8. A two-level directory system. The letters indicate the owners of the directories and files.

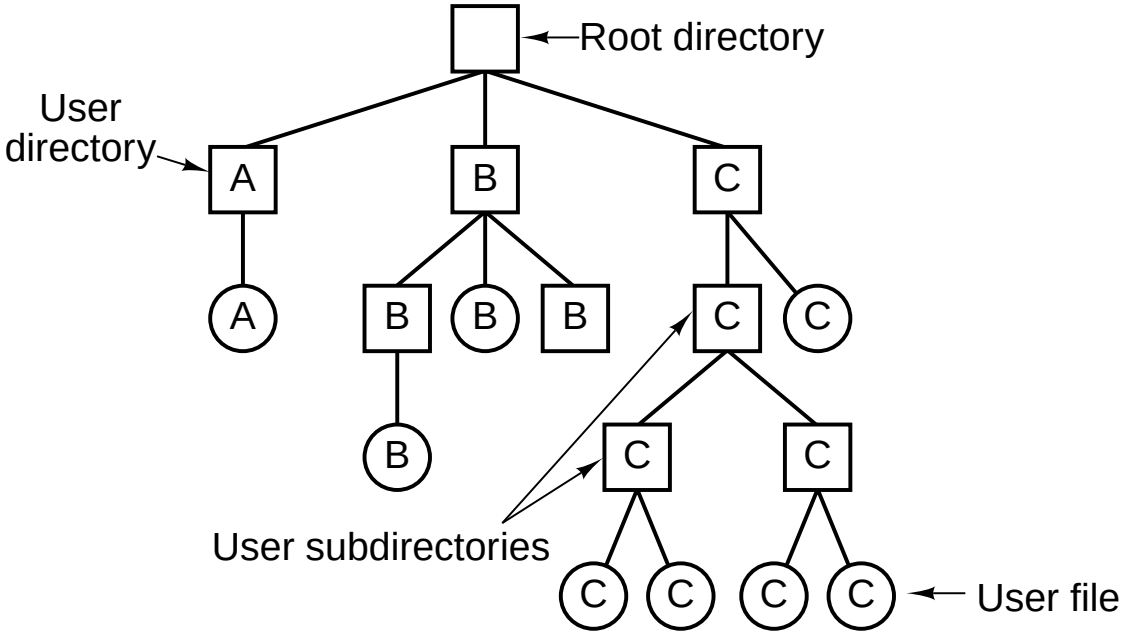

Fig. 6-9. A hierarchical directory system.

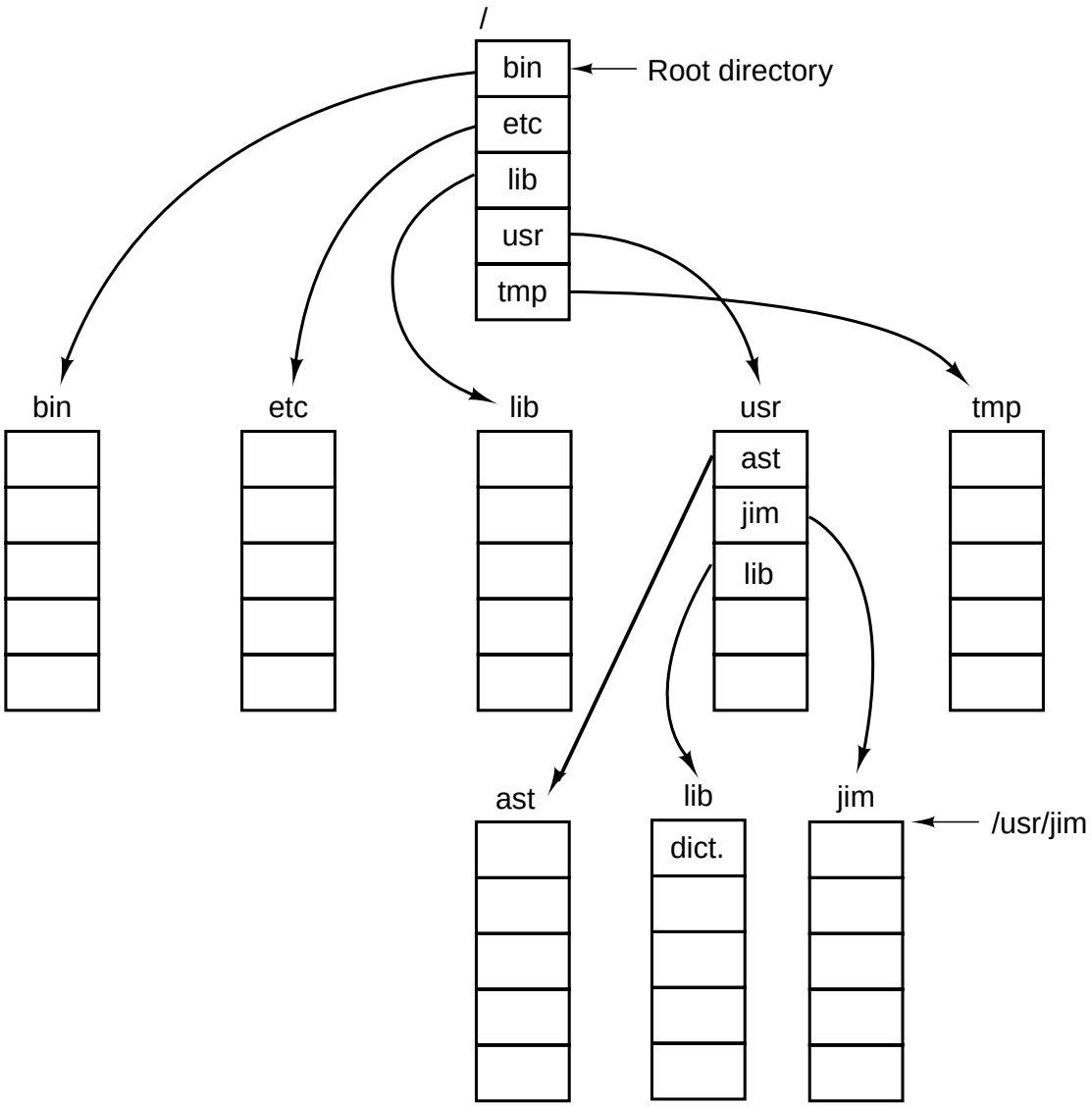

Fig. 6-10. A UNIX directory tree.

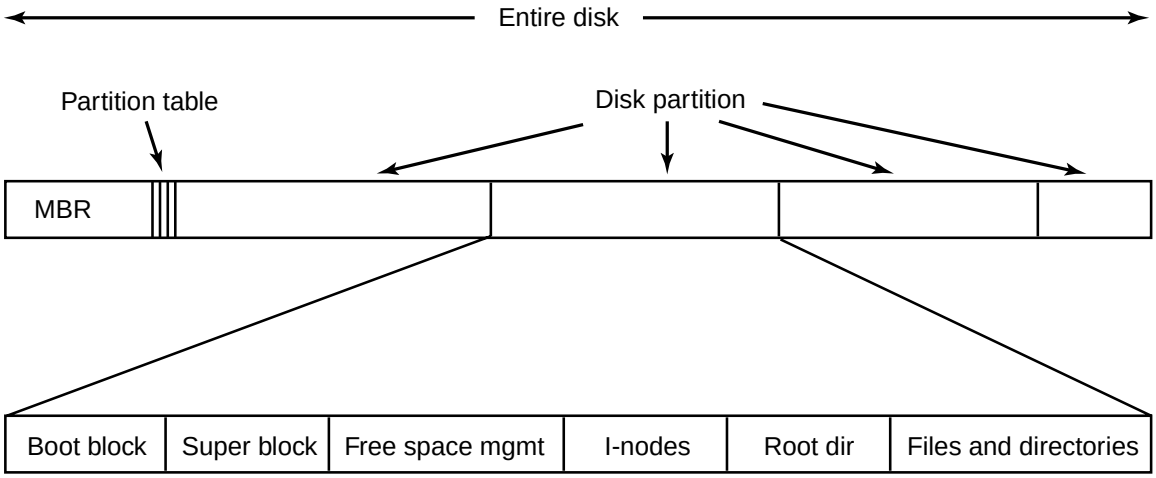

Fig. 6-11. A possible file system layout.

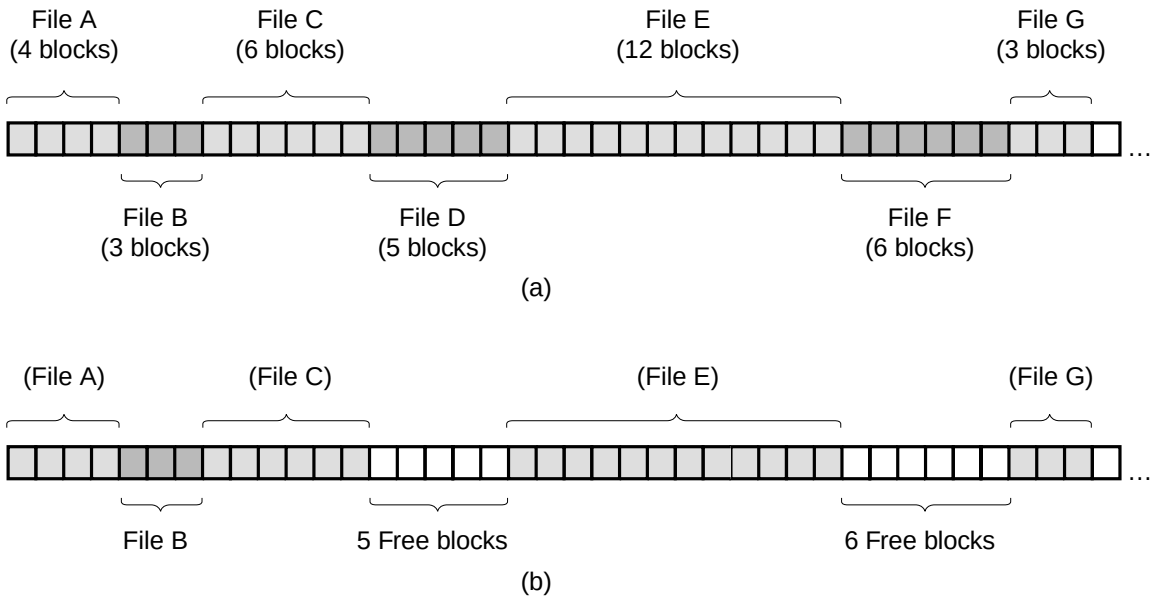

Fig. 6-12. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

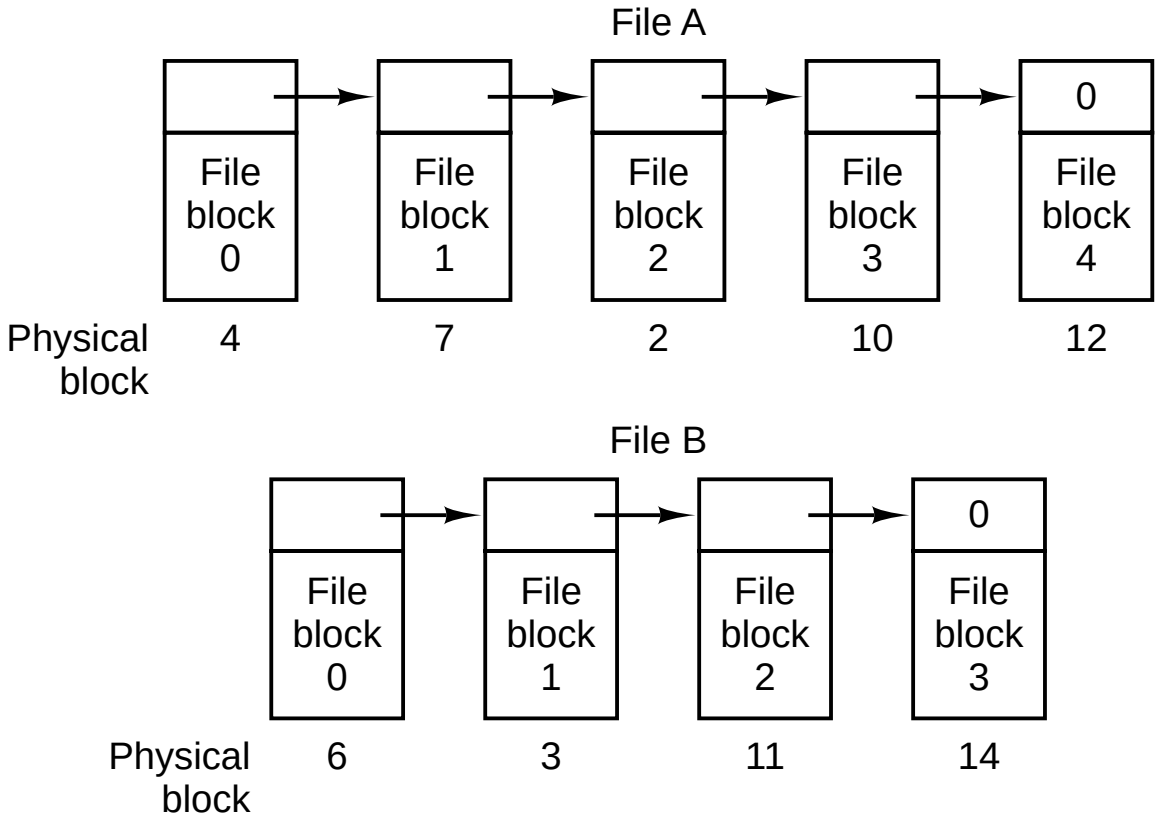

Fig. 6-13. Storing a file as a linked list of disk blocks.

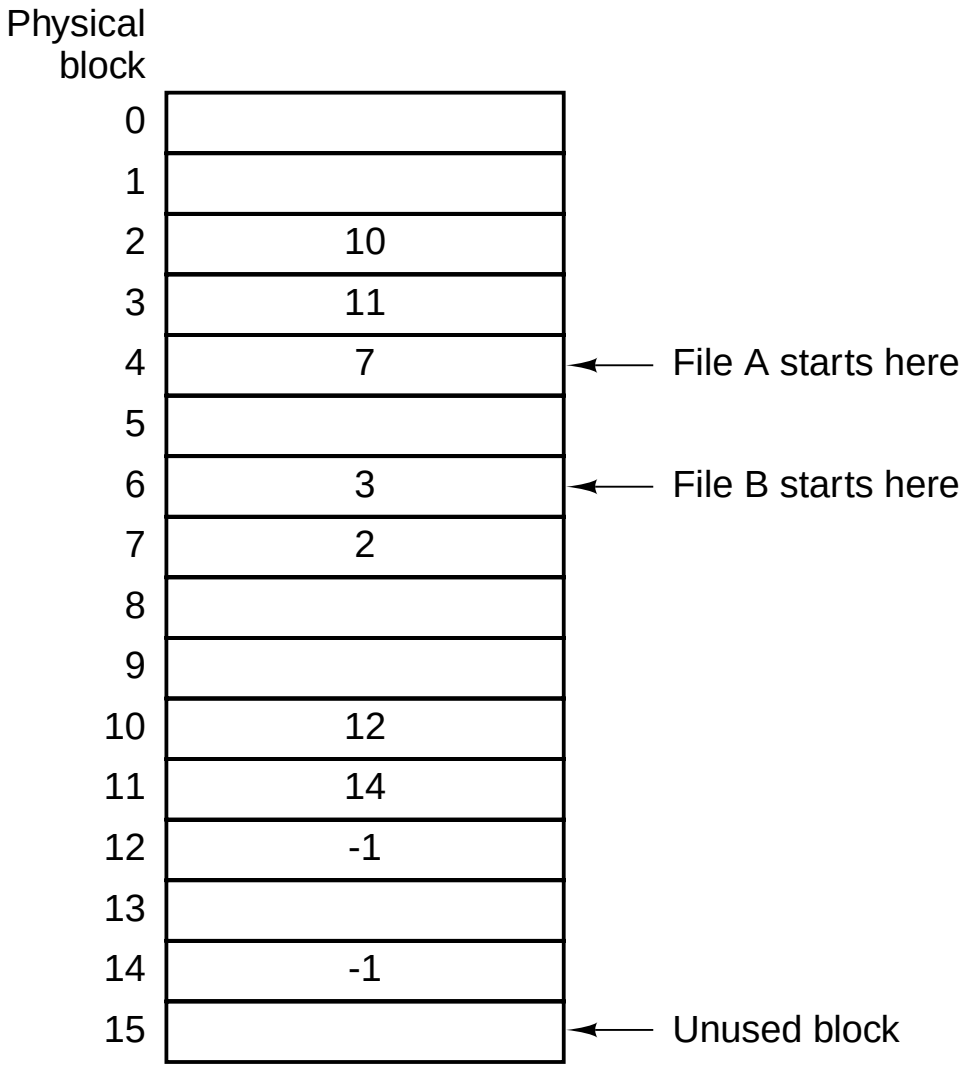

Fig. 6-14. Linked list allocation using a file allocation table in main memory.

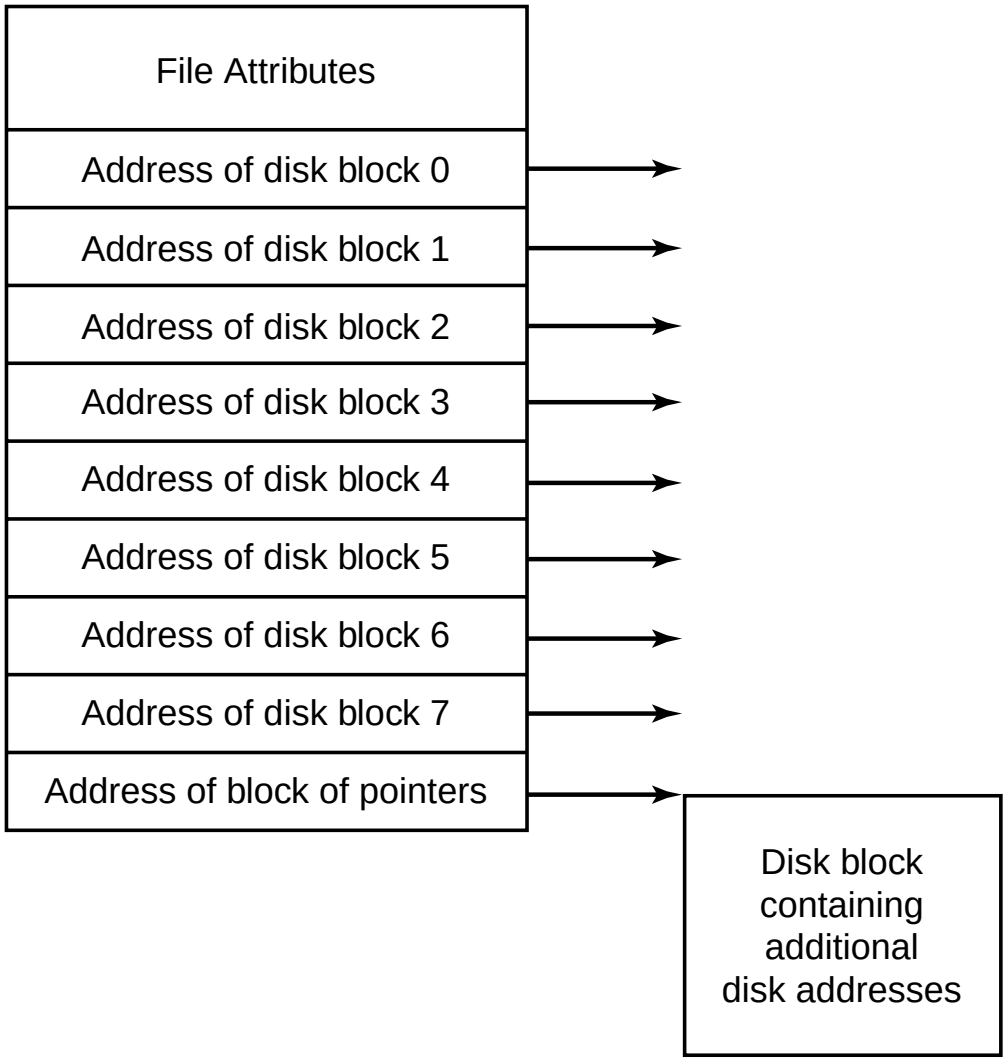

Fig. 6-15. An example i-node.

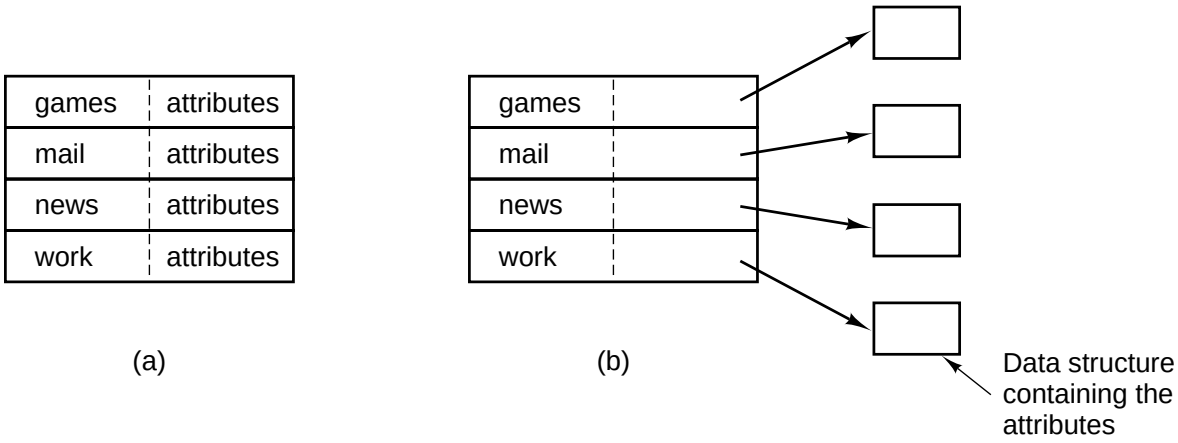

Fig. 6-16. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

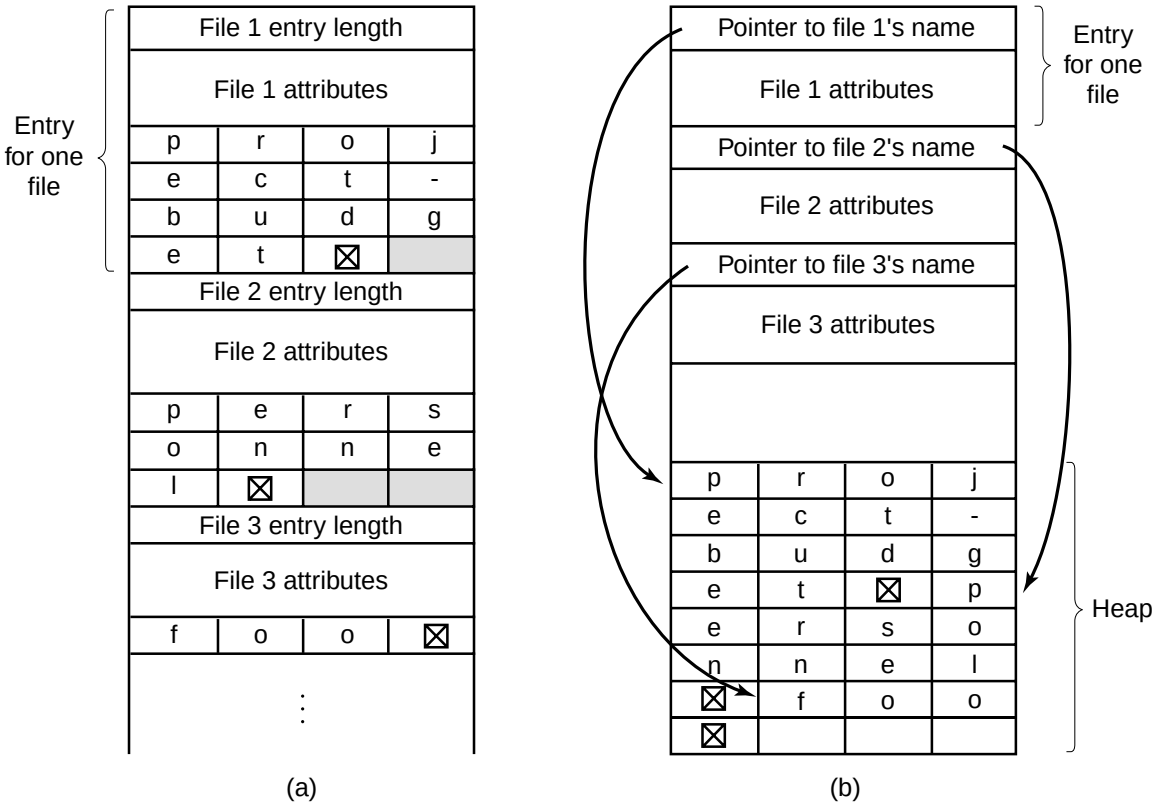

Fig. 6-17. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

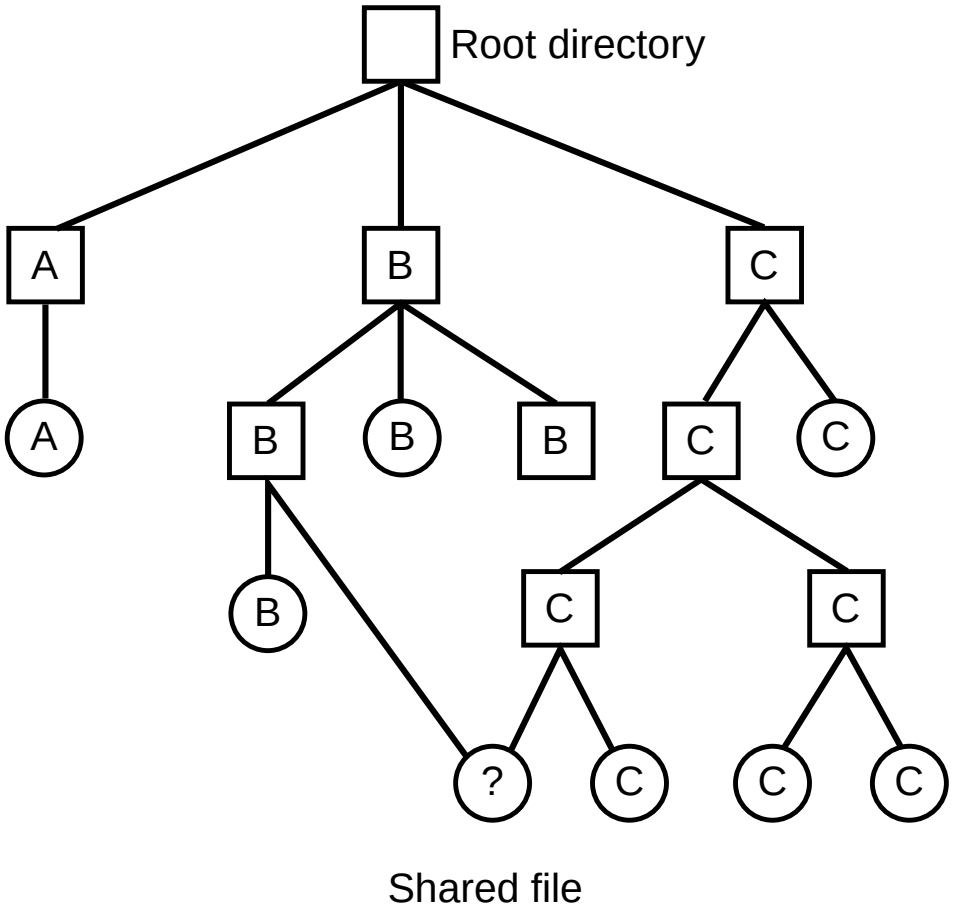

Fig. 6-18. File system containing a shared file.

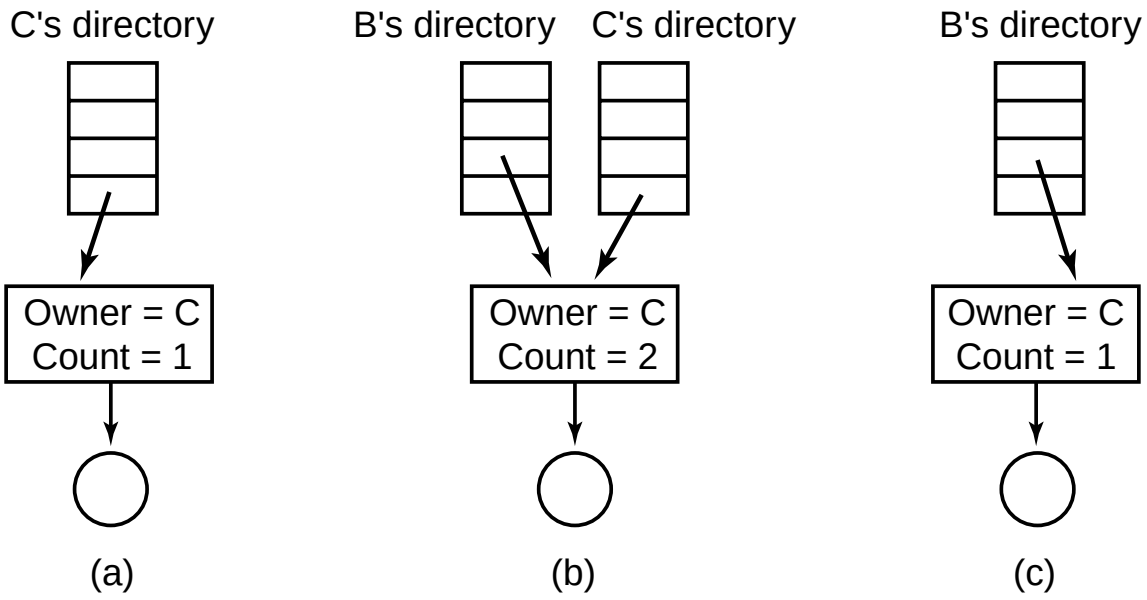

Fig. 6-19. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

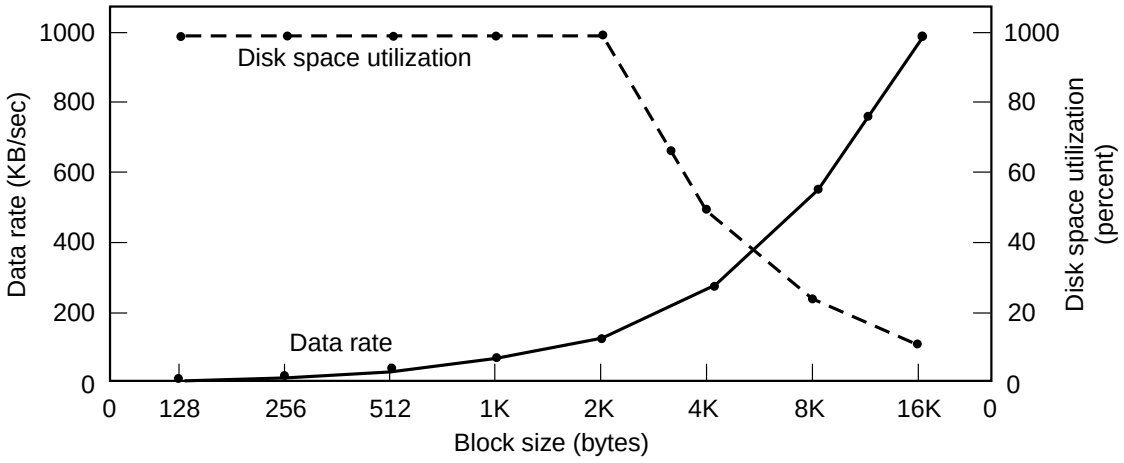

Fig. 6-20. The solid curve (left-hand scale) gives the data rate of a disk. The dashed curve (right-hand scale) gives the disk space efficiency. All files are 2 KB.

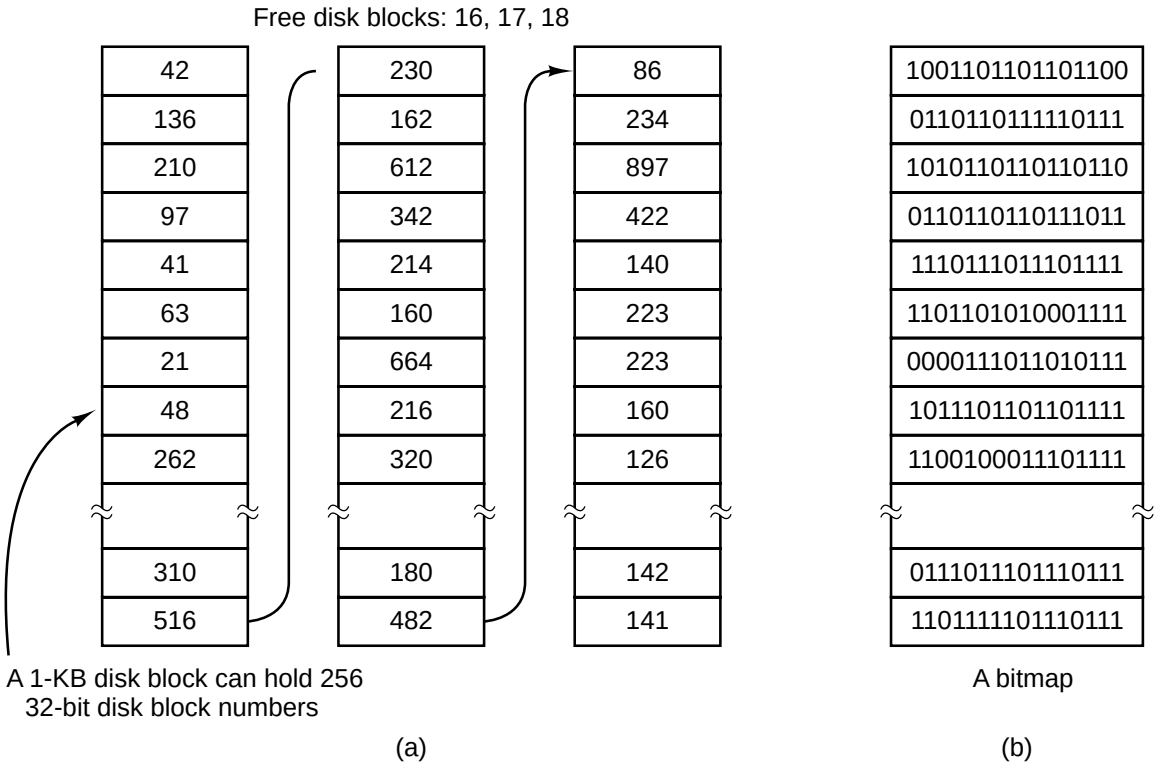

Fig. 6-21. (a) Storing the free list on a linked list. (b) A bitmap.

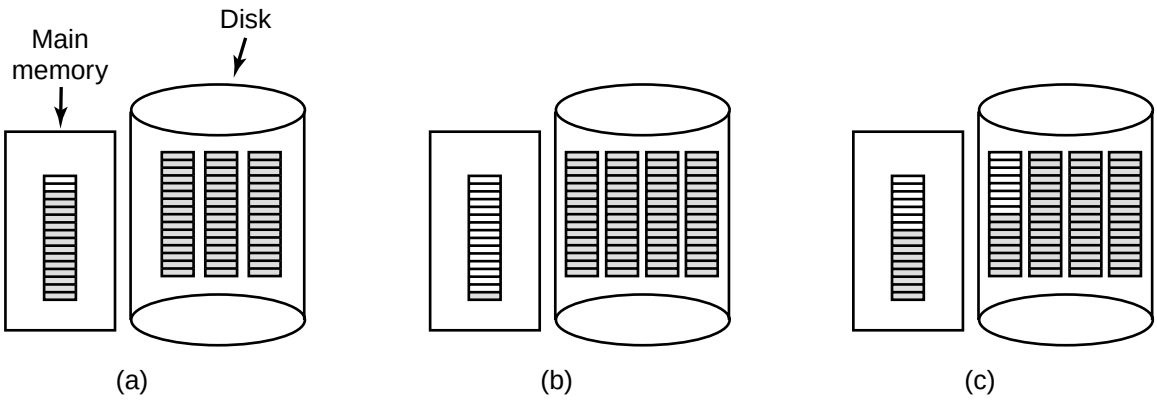

Fig. 6-22. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

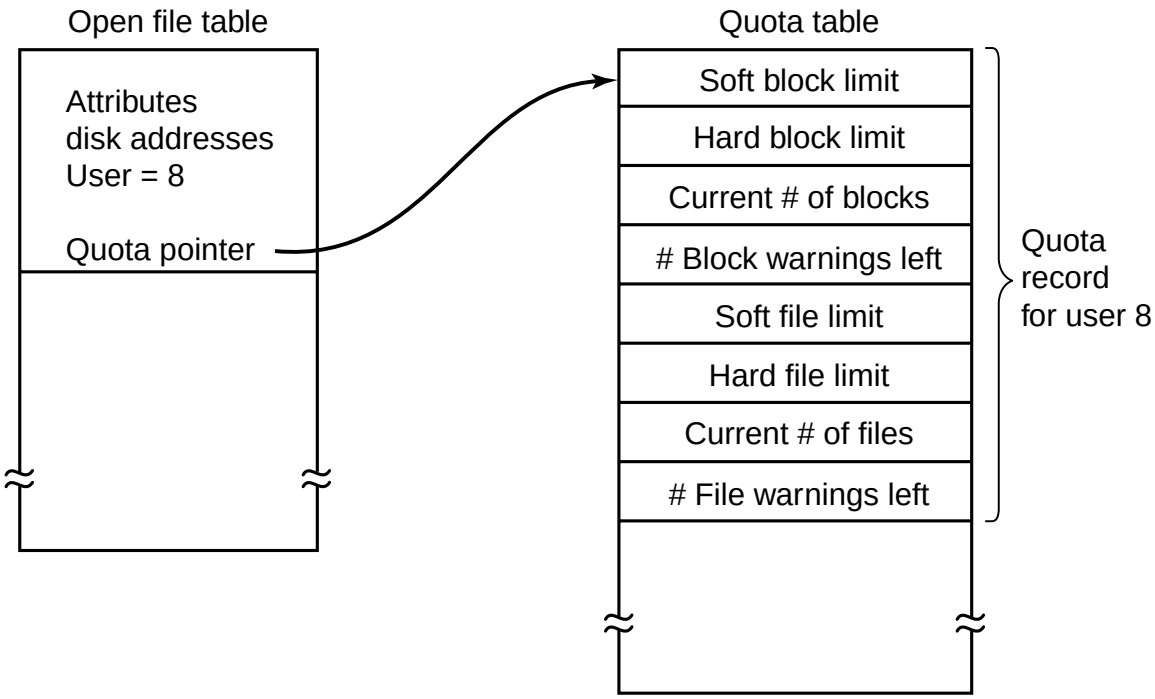

Fig. 6-23. Quotas are kept track of on a per-user basis in a quota table.

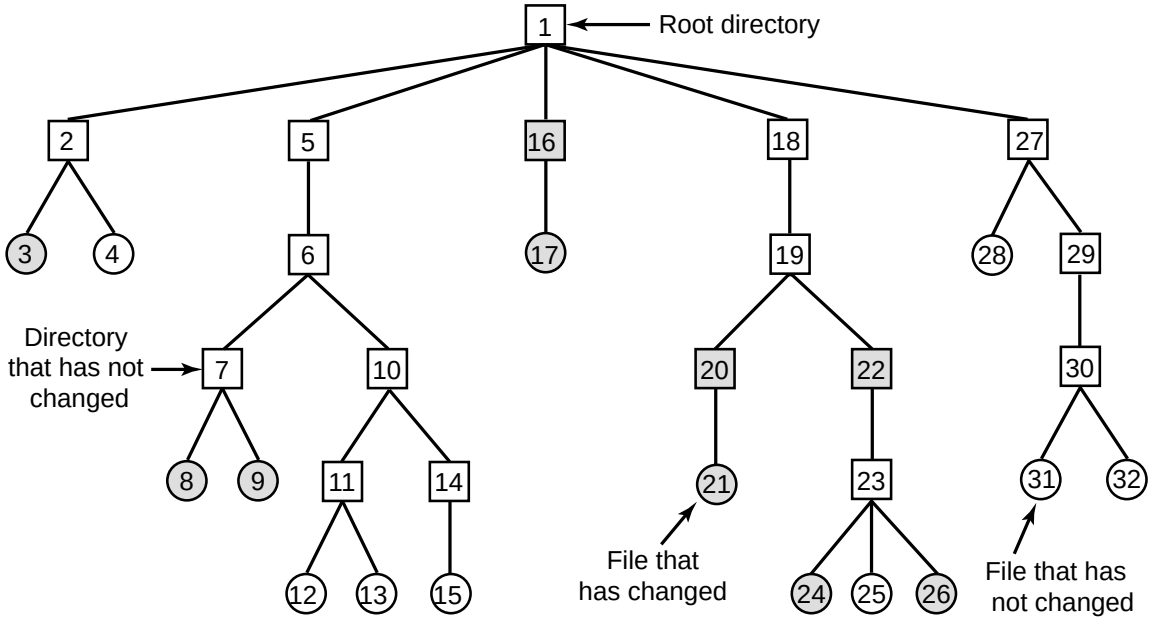

Fig. 6-24. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

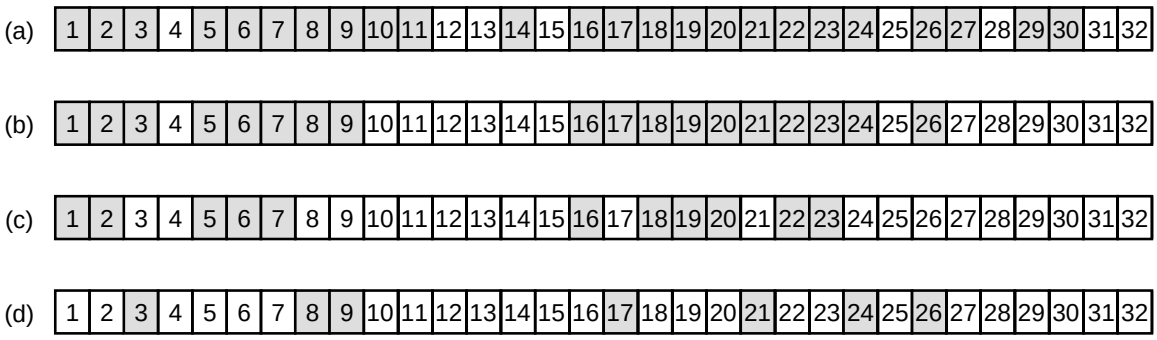

Fig. 6-25. Bit maps used by the logical dumping algorithm.

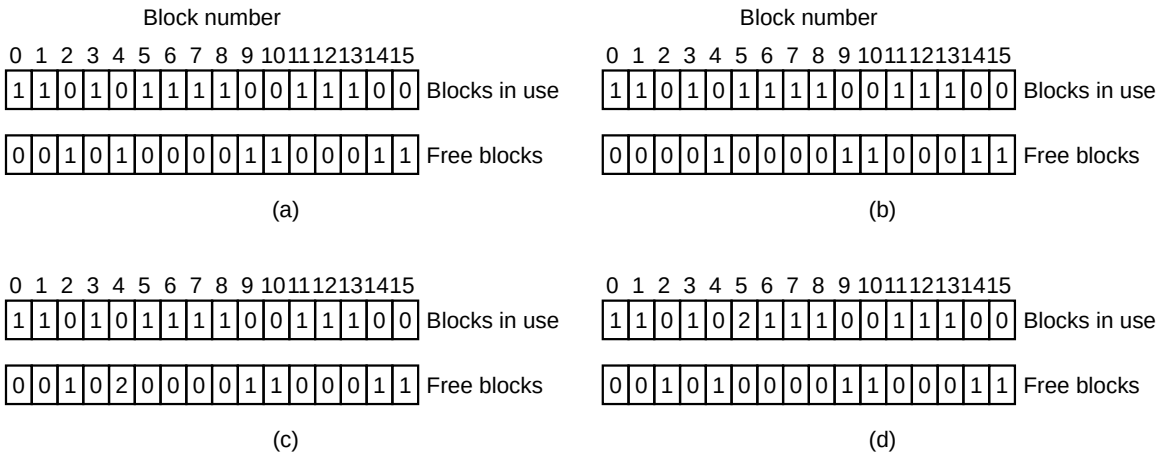

Fig. 6-26. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

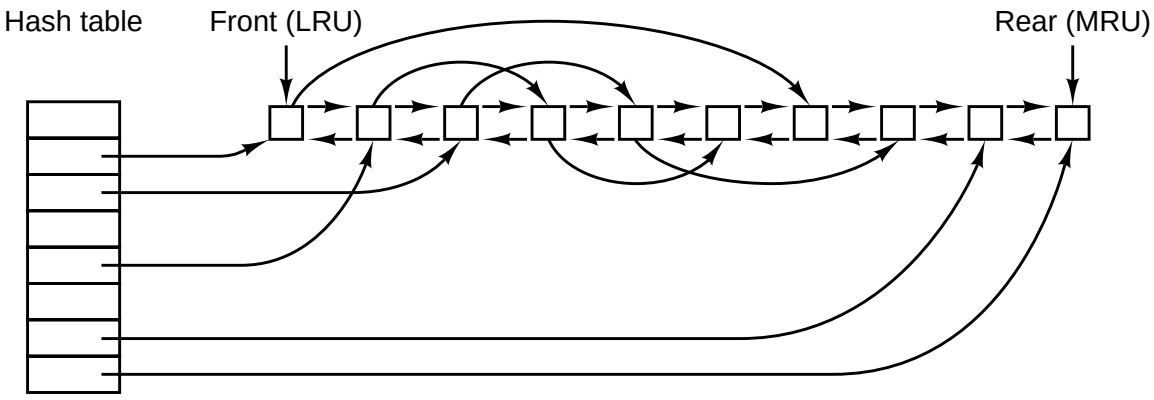

Fig. 6-27. The buffer cache data structures.

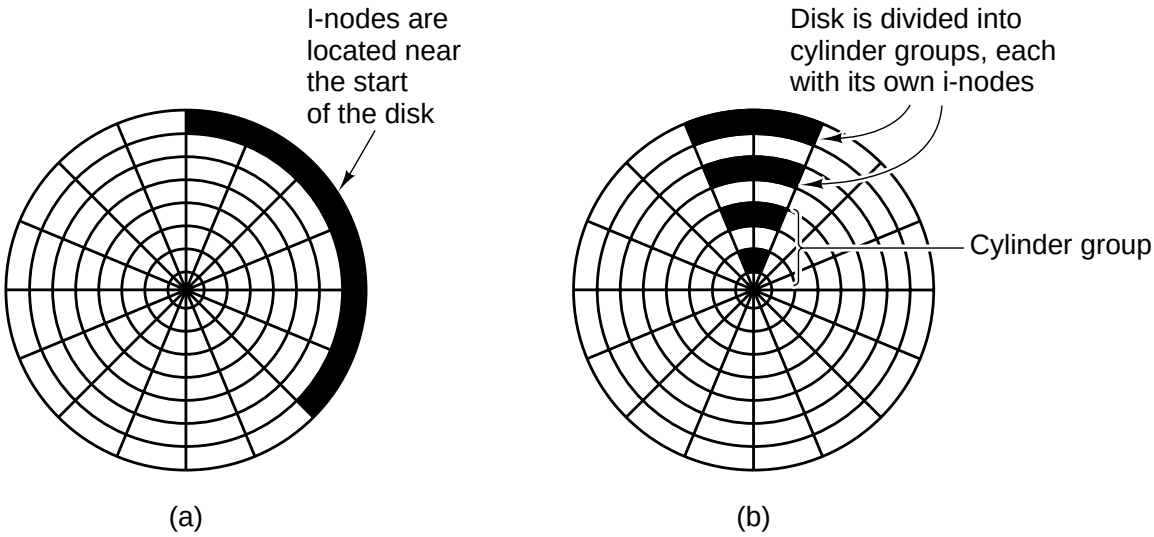

Fig. 6-28. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

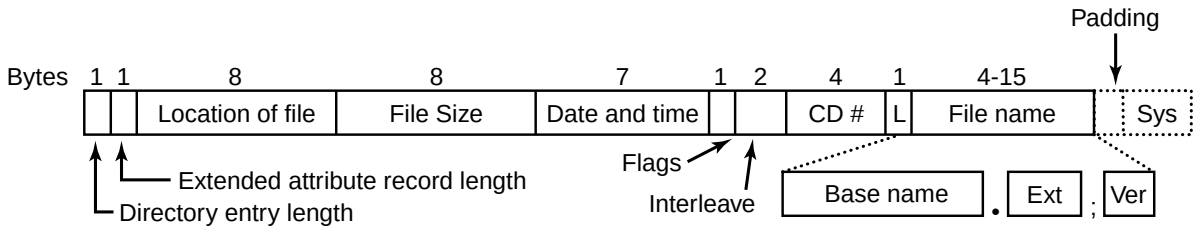

Fig. 6-29. The ISO 9660 directory enty.

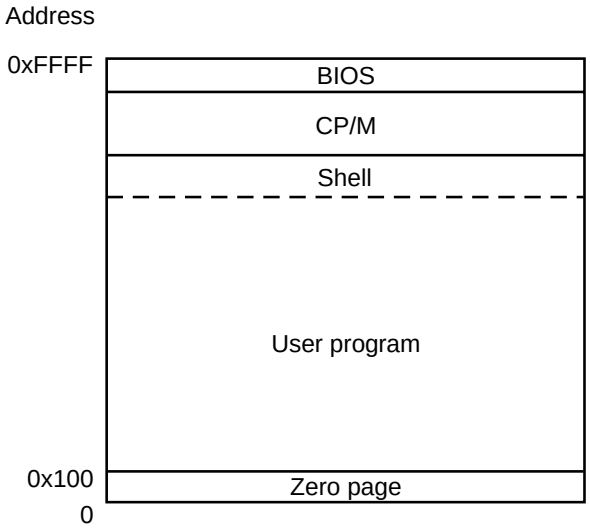

Fig. 6-30. Memory layout of CP/M.

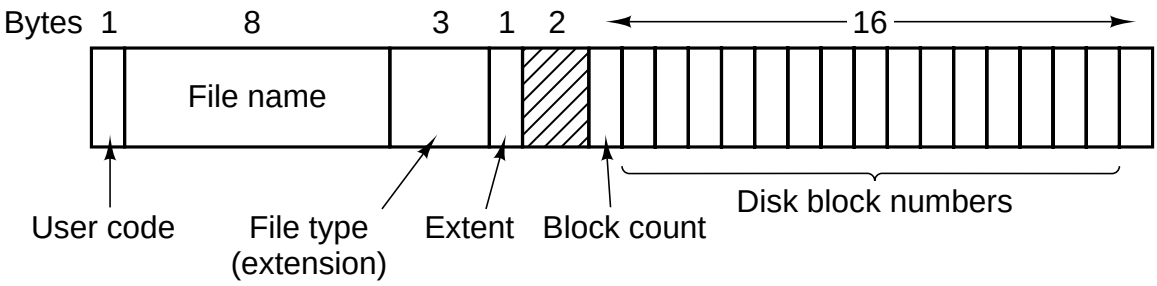

Fig. 6-31. The CP/M directory entry format.

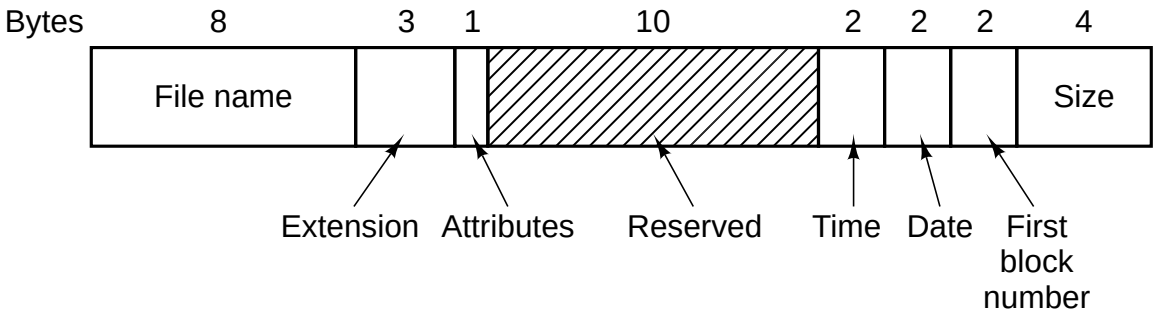

Fig. 6-32. The MS-DOS directory entry.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Fig. 6-33. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

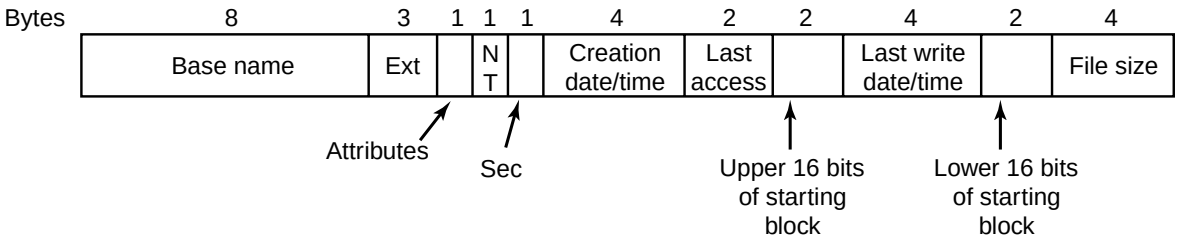

Fig. 6-34. The extended MOS-DOS directory entry used in Windows 98.

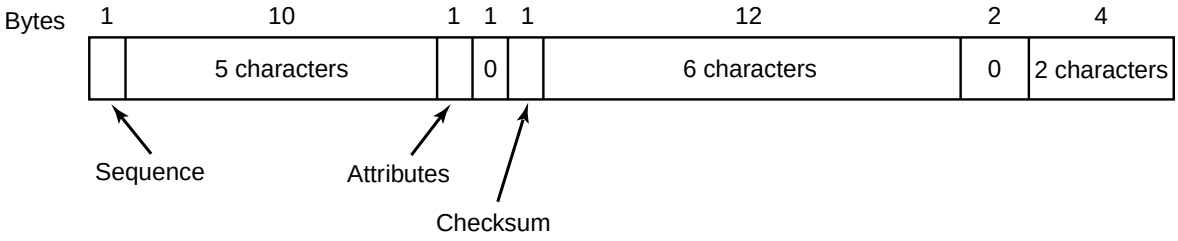

Fig. 6-35. An entry for (part of) a long file name in Windows 98.

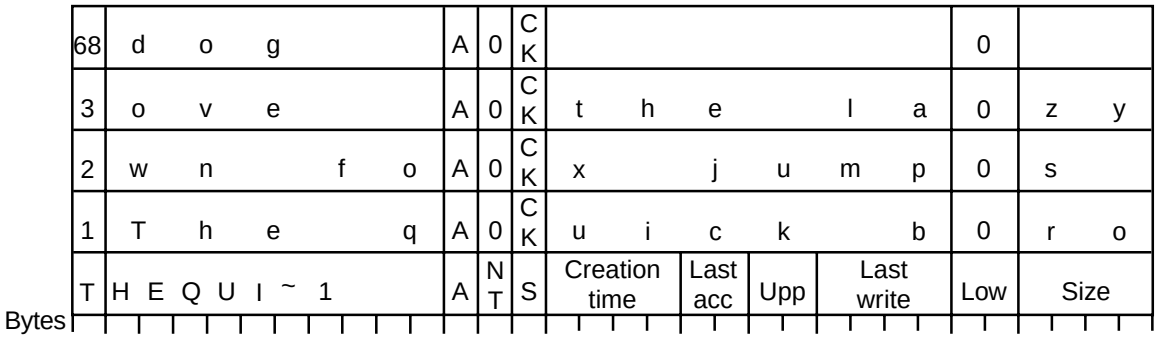

Fig. 6-36. An example of how a long name is stored in Windows 98.

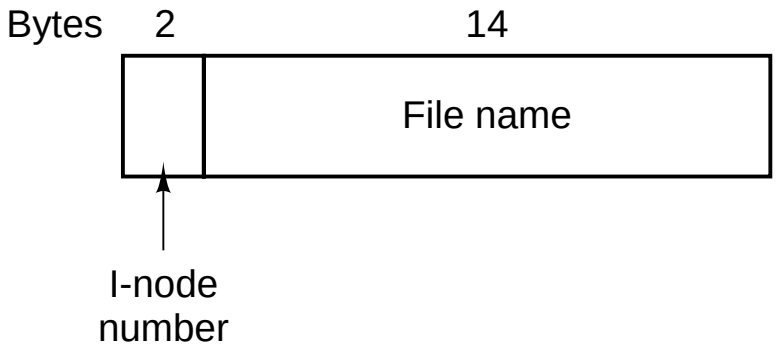

Fig. 6-37. A UNIX V7 directory entry.

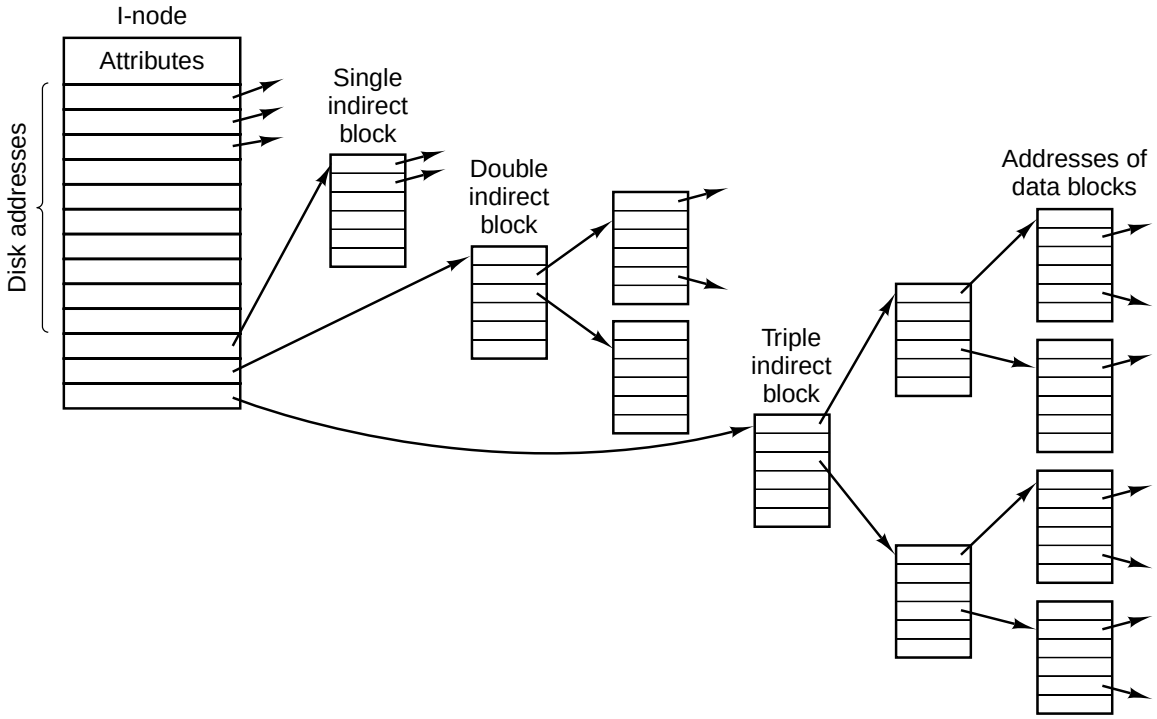

Fig. 6-38. A UNIX i-node.

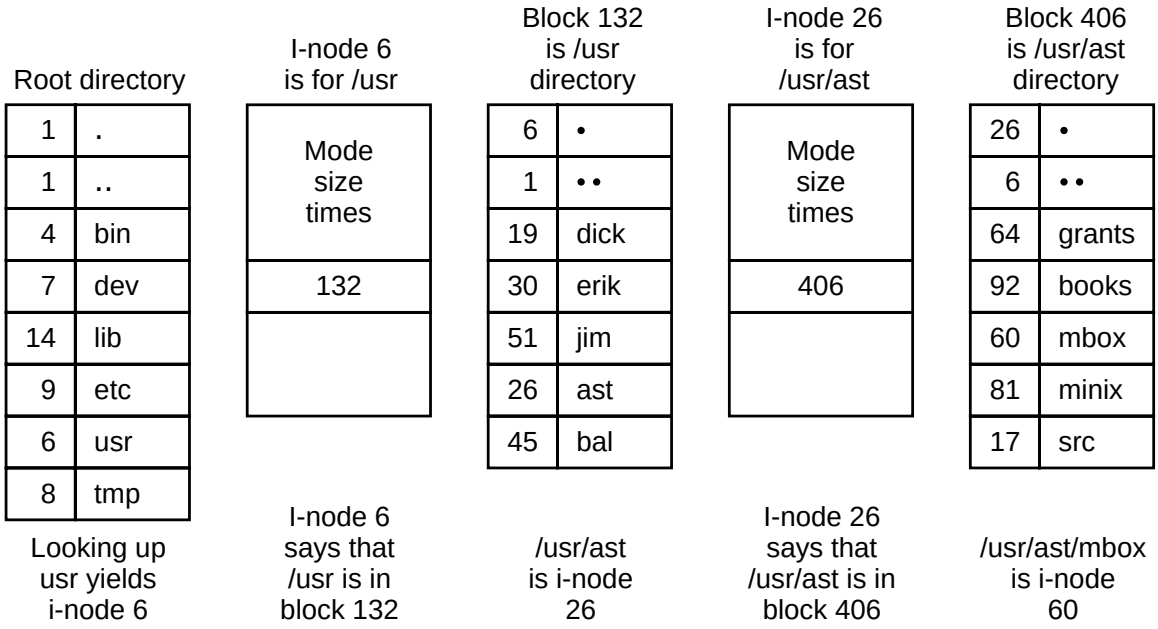

Fig. 6-39. The steps in looking up */usr/ast/mbox*.