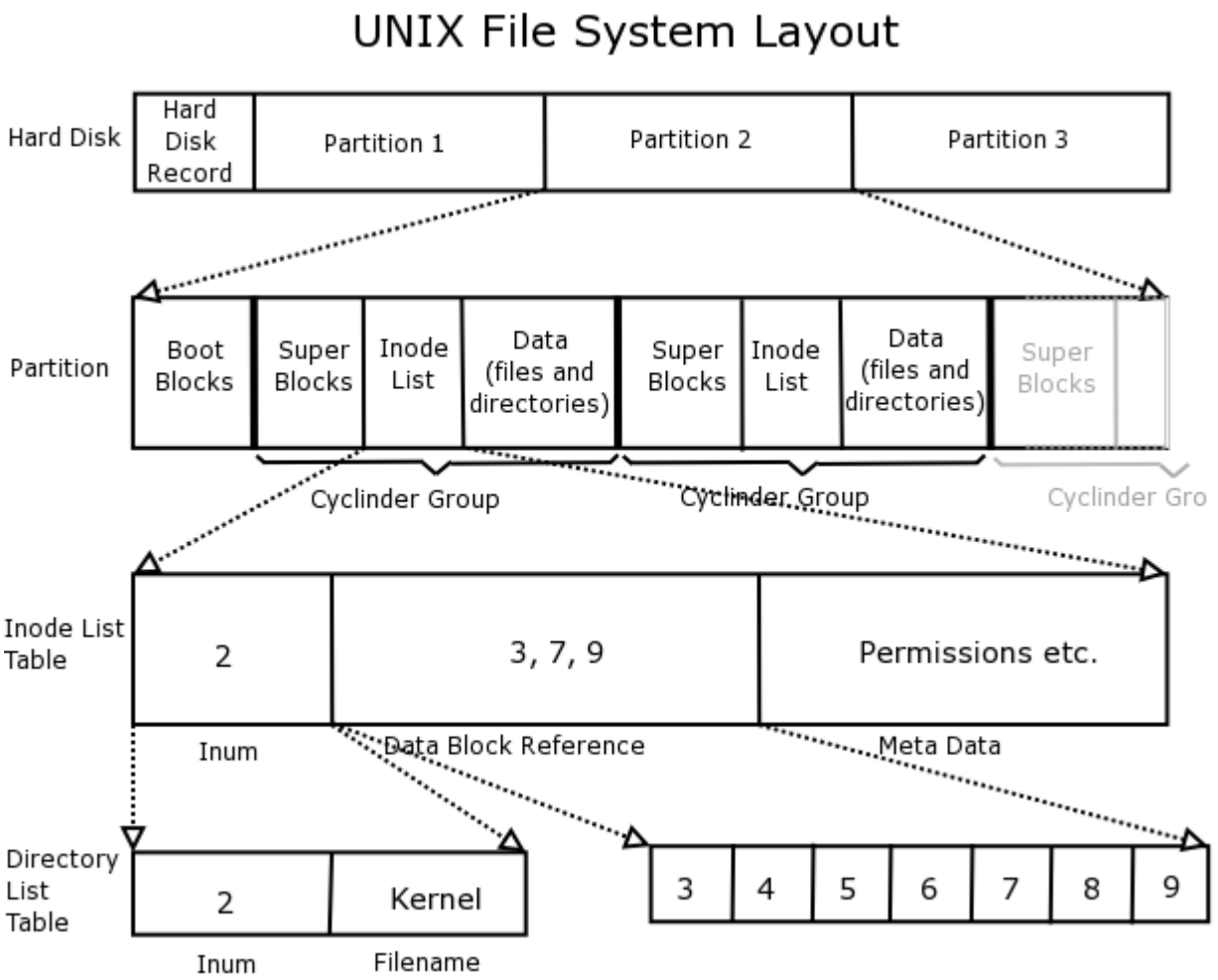
File Systems. Part 1



http://www.learnlinux.org.za/courses/build/internals/ch08s04.html

"… the abstractions of processes (and threads), address spaces, and files are the most important concepts relating to operating systems.

… Files are logical units of information created by processes. A disk will usually contain thousands or even millions of them, each one independent of the others.

… Processes can read existing files and create new ones if need be. Information stored in files must be persistent, that is, not be affected by process creation and termination. A file should disappear only when its owner explicitly removes it."

MOS4E

“… Files are managed by the operating system.

How they are structured, named, accessed, used, protected, implemented, and managed are major topics in operating system design.

As a whole, that part of the operating system dealing with files is known as the file system.”

MOS4E

File Naming

"… A file is an abstraction mechanism. It provides a way to store information on the disk and read it back later. This must be done in such a way as to shield the user from the details of how and where the information is stored, and how the disks actually work.

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."

MOS4E

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

Figure 4-1. Some typical file extensions.

MOS4E

File Structure

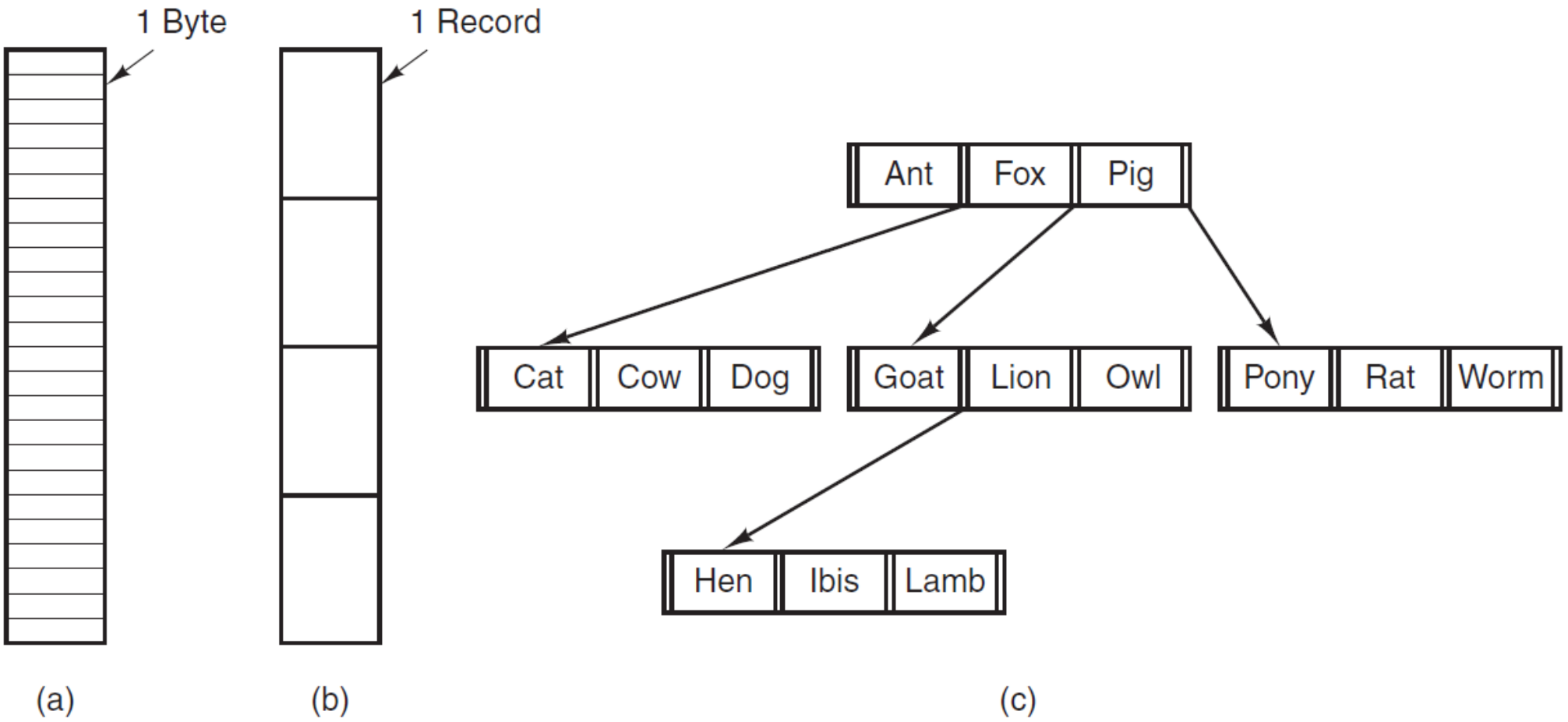


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Types

Regular files

Directories

Character special files
Block special files

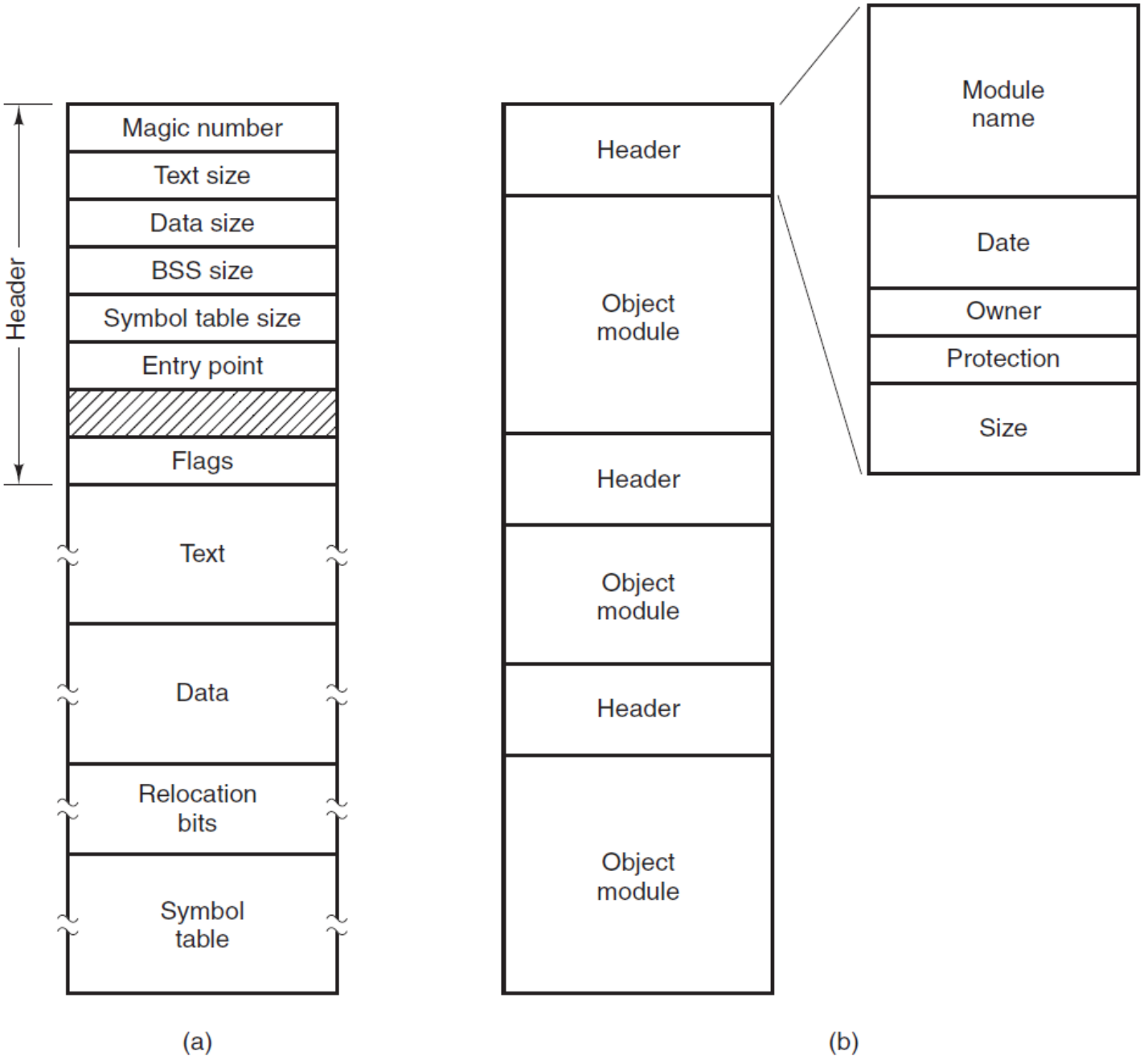


Figure 4-3. (a) An executable file. (b) An archive.

File Access

Sequential access

Random access

operation seek

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 was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4. Some possible file attributes.

MOS4E

File Operations (system calls)

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set attributes
11. Rename

MOS4E

```
/* 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 */
```

MOS4E

```
    /* 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 */
}
```

Figure 4-5. A simple program to copy a file.

Directories:
Single-level Directory Systems

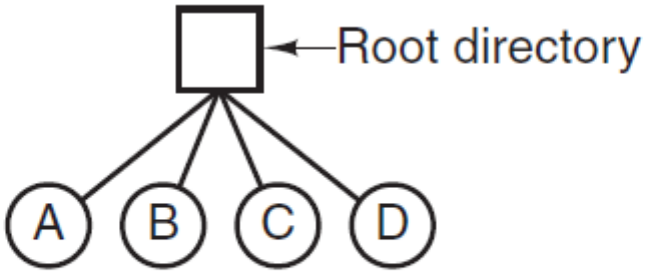


Figure 4-6. A single-level directory system containing four files.

MOS4E

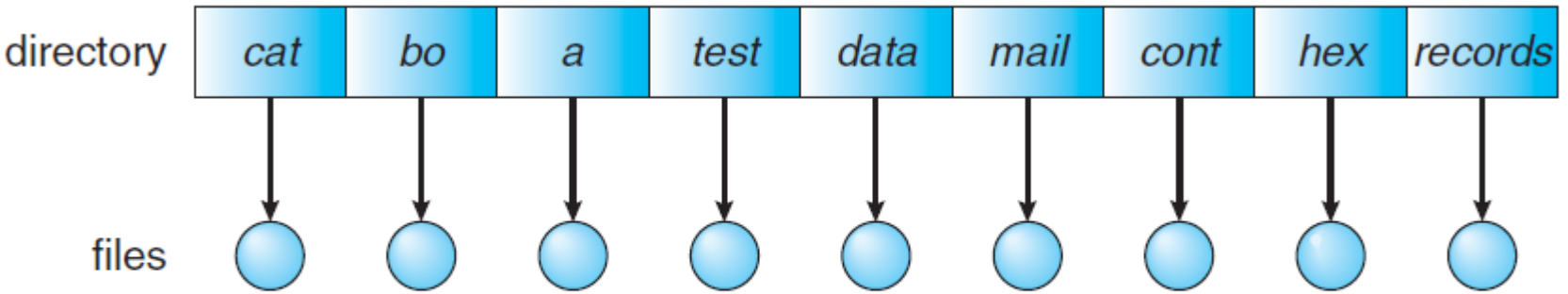


Figure 11.9 Single-level directory.

OSC9E

Directories:
Two-level Directory Systems

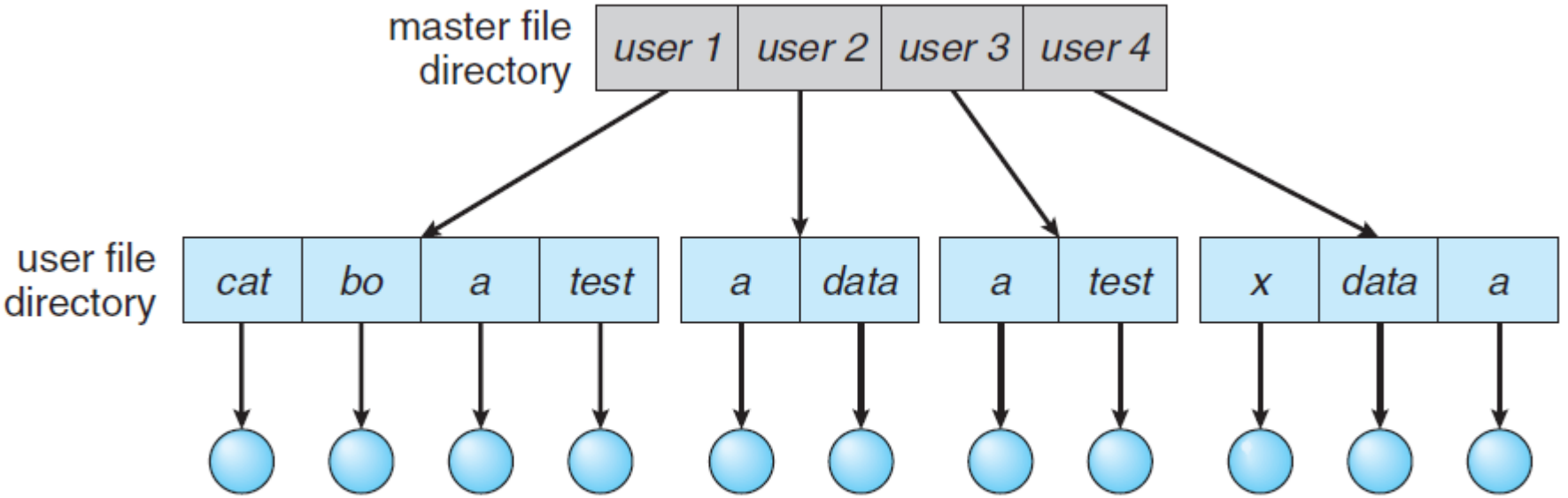


Figure 11.10 Two-level directory structure.

OSC9E

Directories:
Hierarchical Directory Systems

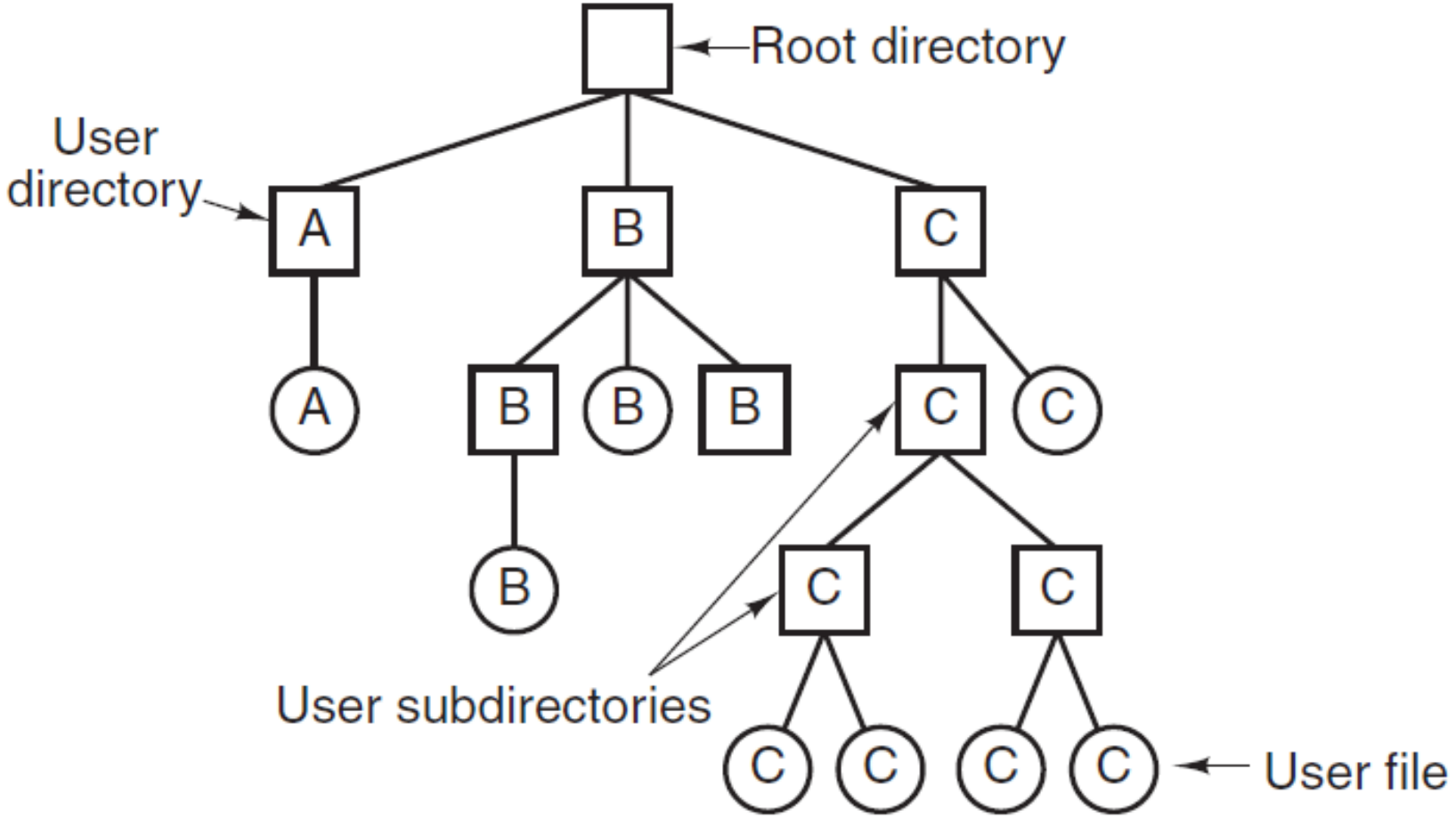


Figure 4-7. A hierarchical directory system.

MOS4E

Directories:
Hierarchical Directory Systems

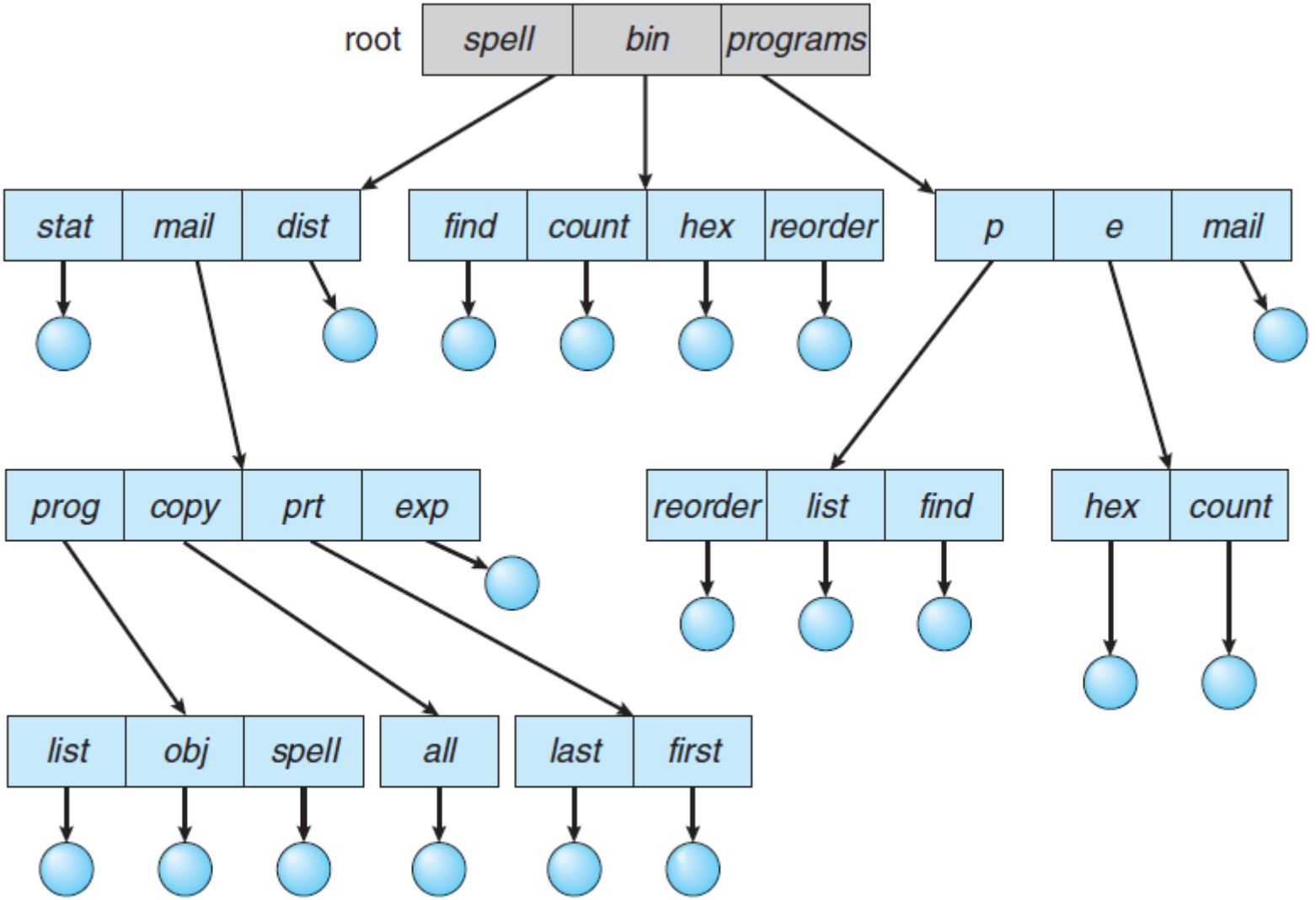


Figure 11.11 Tree-structured directory structure.

Path Names

Absolute path name

Relative path name

Working (current directory)

Special entries in every directory:

“.” (“dot”)	**current directory**
“..” (“dotdot”)	**parent directory**

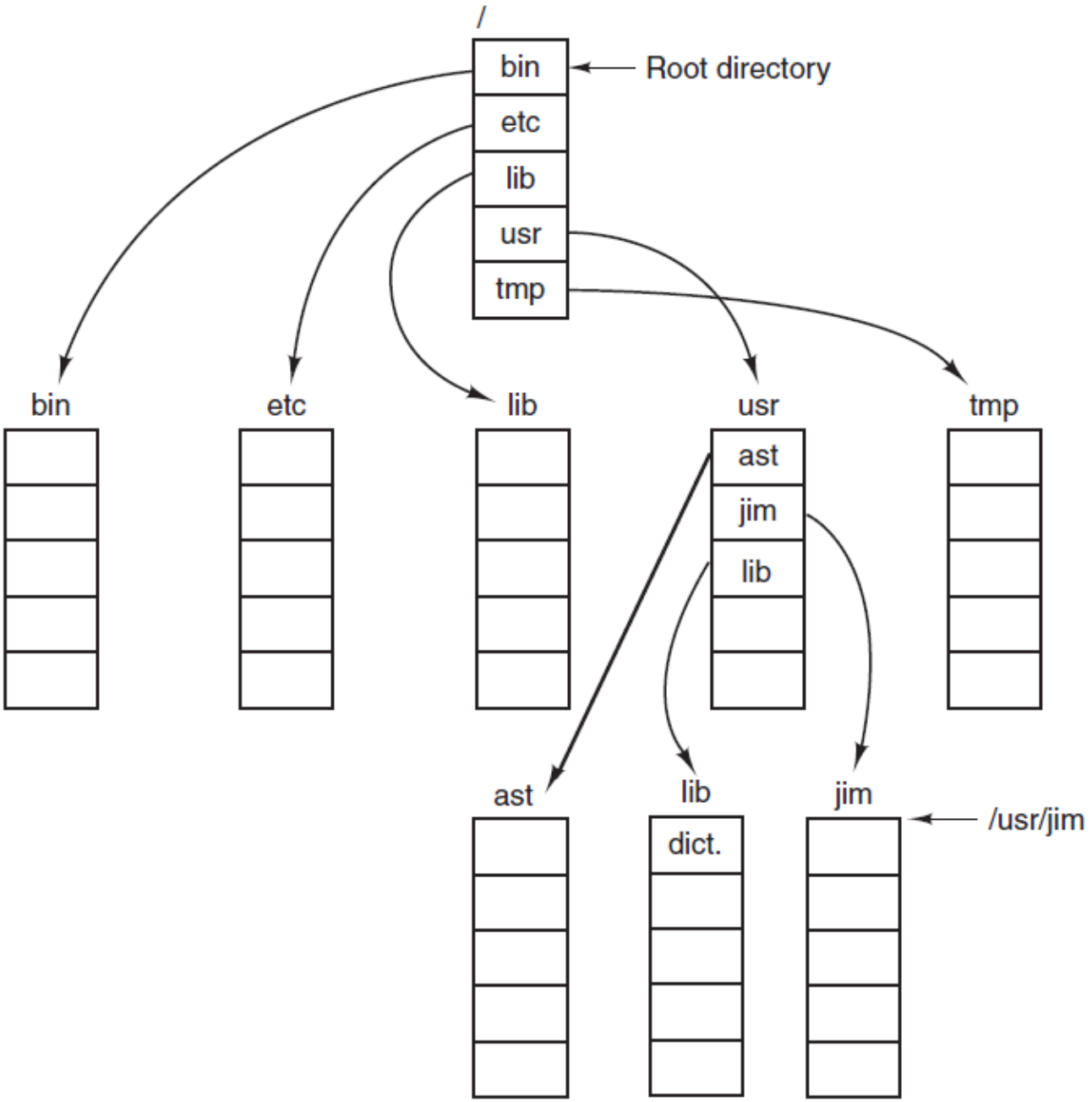


Figure 4-8. A UNIX directory tree.

MOS4E

Directories:
Acyclic-Graph Directories

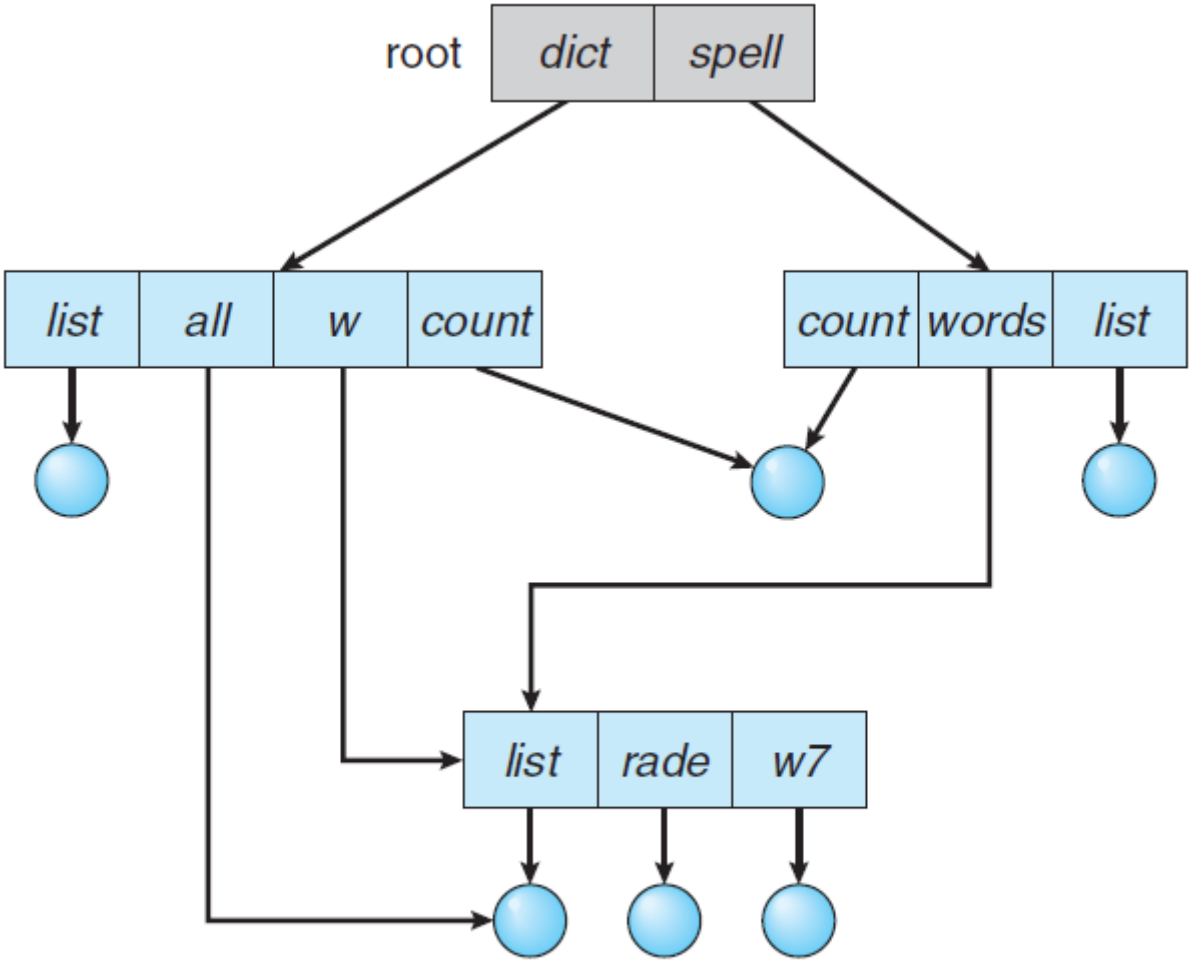


Figure 11.12 Acyclic-graph directory structure.

OSC9E

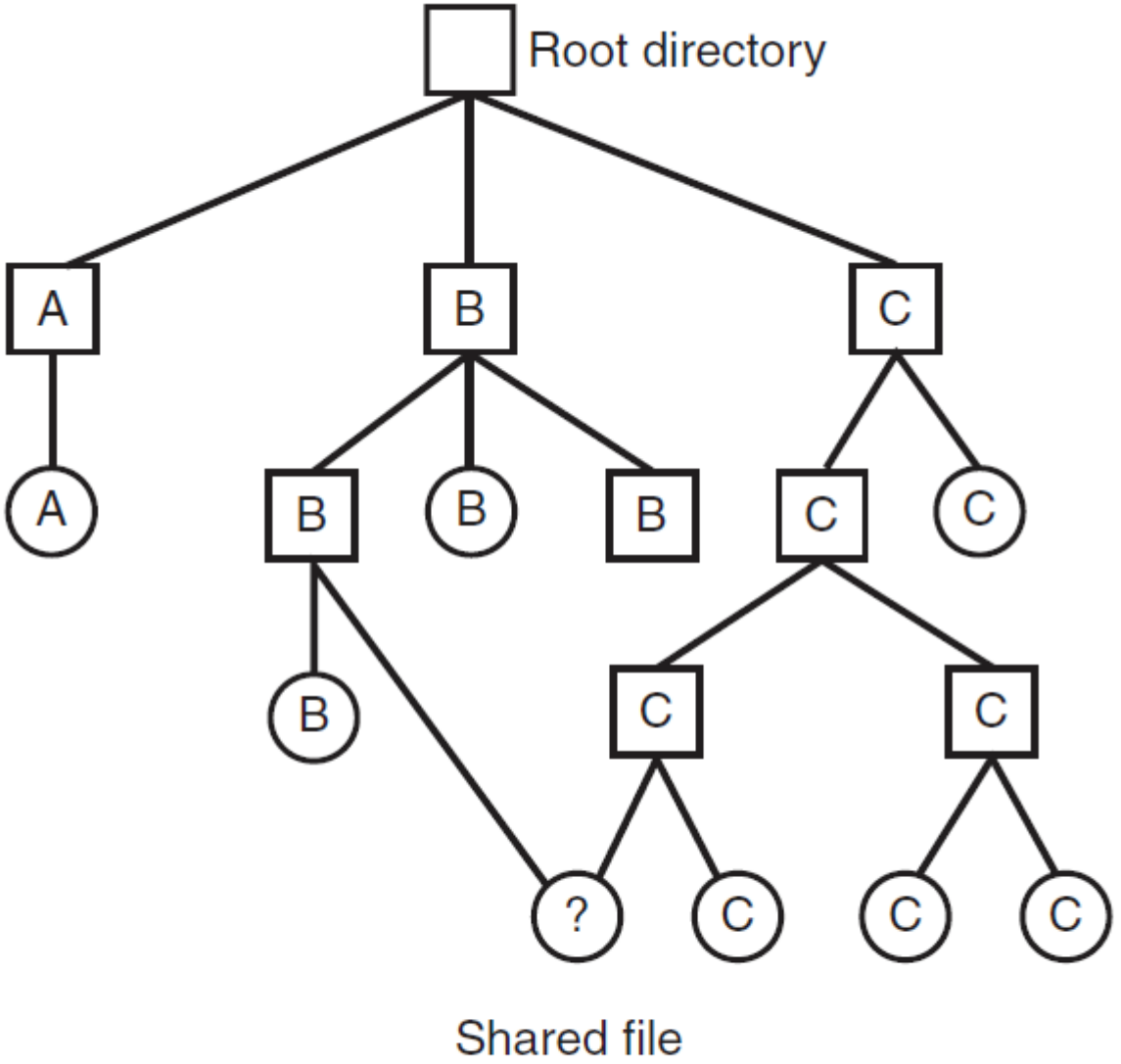


Figure 4-16. File system containing a shared file.

MOS4E

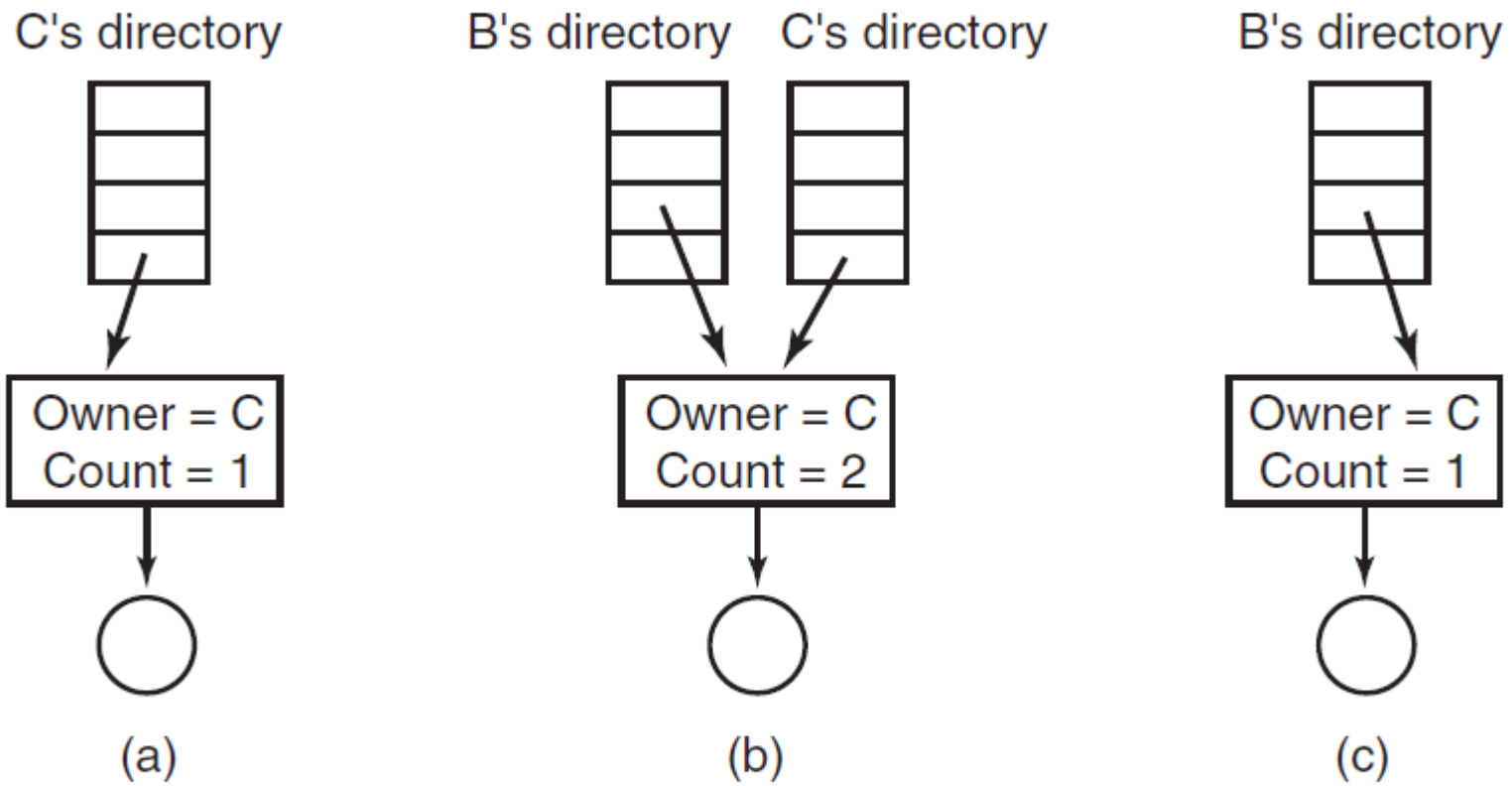


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

MOS4E

Directories:
General Graph Directories

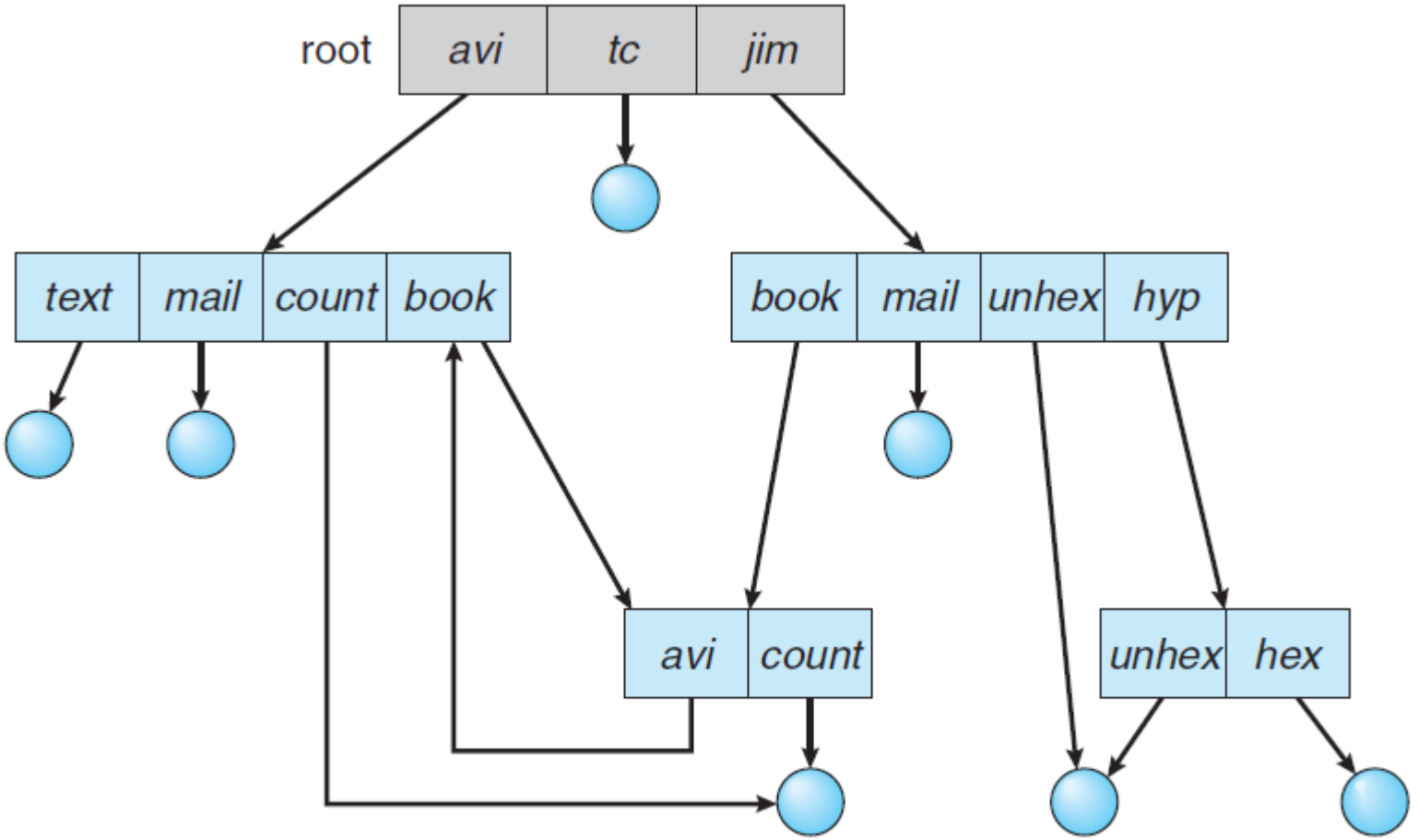


Figure 11.13 General graph directory.

OSC9E

Directory Operations (system calls)

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

MOS4E

File-System Implementation

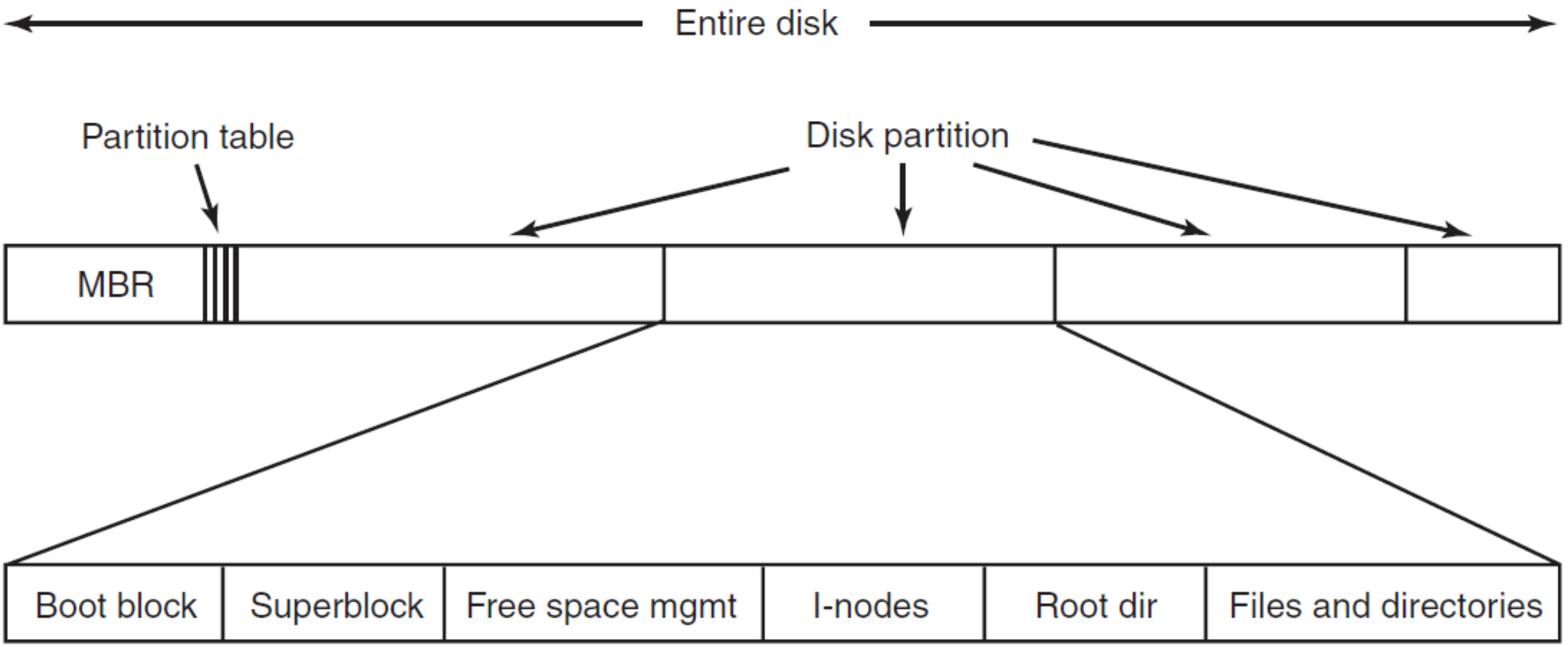


Figure 4-9. A possible file-system layout.

MOS4E

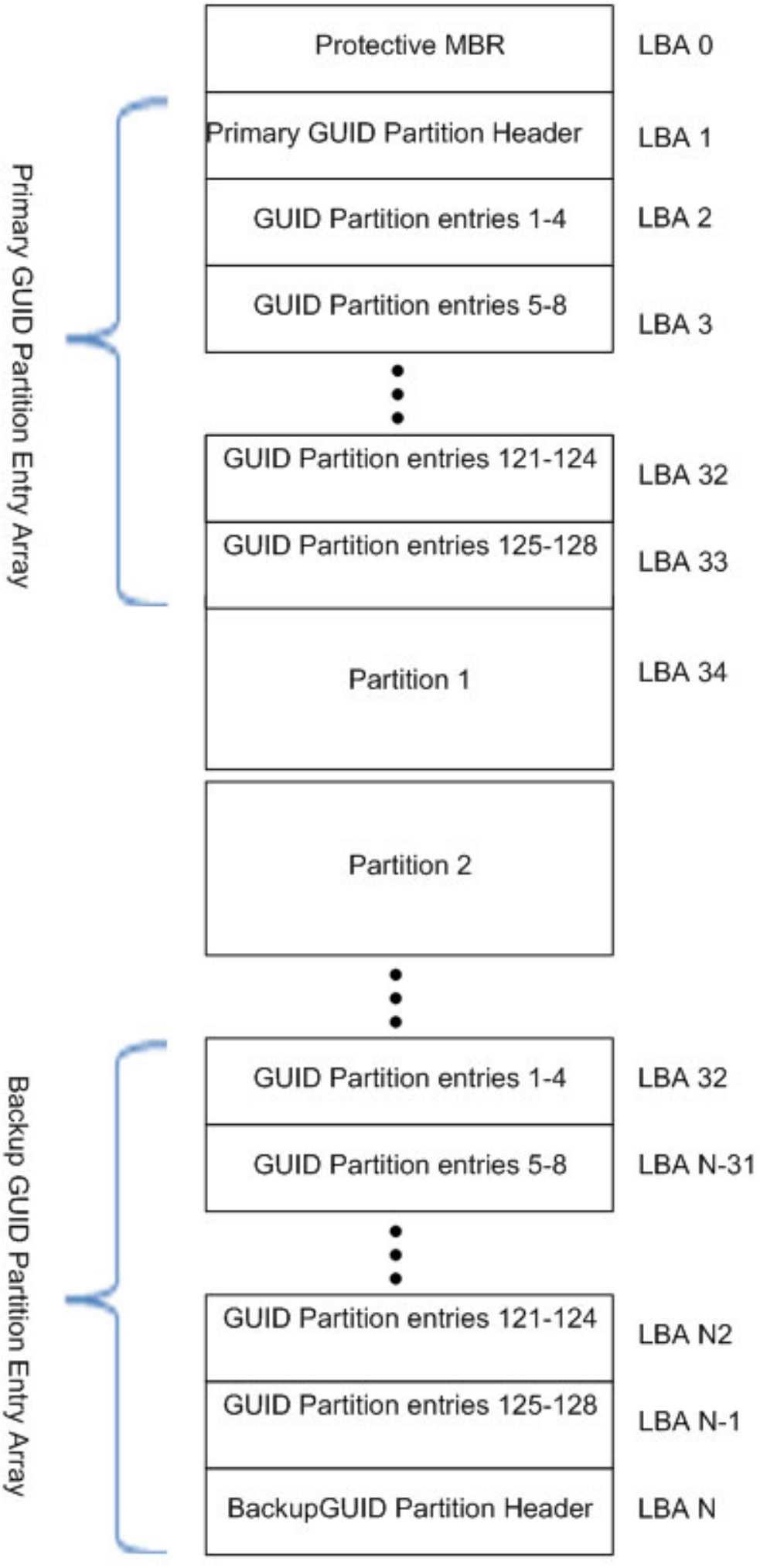
```
# fdisk -l /dev/sda
Disk /dev/sda: 931,5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: dos
Disk identifier: 0xd2cb19de

Disposit.  Inicio      Start      Final  Sectores   Size Id Tipo
/dev/sda1  *            2048    3074047   3072000   1,5G  7 HPFS/NTFS/exFAT
/dev/sda2            3074048  984686591 981612544 468,1G  7 HPFS/NTFS/exFAT
/dev/sda3          984688638 1927555071 942866434 449,6G  f W95 Ext'd (LBA)
/dev/sda4         1927555072 1953521663  25966592  12,4G  7 HPFS/NTFS/exFAT
/dev/sda5          984688640  986786296   2097657     1G 83 Linux
/dev/sda6          986787840 1053896703  67108864    32G 82 Linux swap / Solaris
/dev/sda7         1053898752 1927555071 873656320 416,6G 83 Linux
```

```
# parted /dev/sda
GNU Parted 3.2
Usando /dev/sda
¡Bienvenido/a a GNU Parted! Teclee «help» para ver la lista de órdenes.
(parted) print
Modelo: ATA ST1000DM003-1ER1 (scsi)
Disco /dev/sda: 1000GB
Tamaño de sector (lógico/físico): 512B/4096B
Tabla de particiones: msdos
Disk Flags:

Numero  Inicio  Fin     Tamaño  Tipo      Sistema de archivos  Banderas
 1      1049kB  1574MB  1573MB  primary   ntfs                 arranque
 2      1574MB  504GB   503GB   primary   ntfs
 3      504GB   987GB   483GB   extended                       lba
 5      504GB   505GB   1074MB  logical   ext4
 6      505GB   540GB   34,4GB  logical   linux-swap(v1)
 7      540GB   987GB   447GB   logical   ext4
 4      987GB   1000GB  13,3GB  primary   ntfs

(parted) quit
```



http://macrorit.com/help/partition-gpt-disk.html

```
# parted /dev/sda
GNU Parted 3.2
Usando /dev/sda
¡Bienvenido/a a GNU Parted! Teclee «help» para ver la lista de órdenes.
(parted) print
Modelo: HGST HTS 725050A7E630 (scsi)
Disco /dev/sde: 500GB
Tamaño de sector (lógico/físico): 512B/512B
Tabla de particiones: gpt
Disk Flags:

Numero  Inicio  Fin     Tamaño  Sistema de archivos  Nombre                          Banderas
 1      1049kB  1050MB  1049MB  ntfs                                                 oculta, diag
 2      1050MB  1322MB  273MB   fat32                EFI system partition            arranque, es
 3      1322MB  1456MB  134MB                        Microsoft reserved partition    msftres
 4      1456MB  247GB   245GB   ntfs                 Basic data partition            msftdata
 6      247GB   264GB   17,2GB  linux-swap(v1)
 7      264GB   486GB   222GB
 5      486GB   500GB   13,9GB  ntfs                                                 oculta, diag

(parted) quit
```

```
$ diskutil list
/dev/disk0 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *251.0 GB   disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:          Apple_CoreStorage Macintosh HD            250.1 GB   disk0s2
   3:                 Apple_Boot Recovery HD             650.1 MB   disk0s3

/dev/disk1 (internal, virtual):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:                 Apple_HFS Macintosh HD           +249.8 GB   disk1
                                Logical Volume on disk0s2
                                7464223A-BED2-4281-90B2-444BA62069D6
                                Unencrypted
```

Implementing Files: Contiguous Allocation

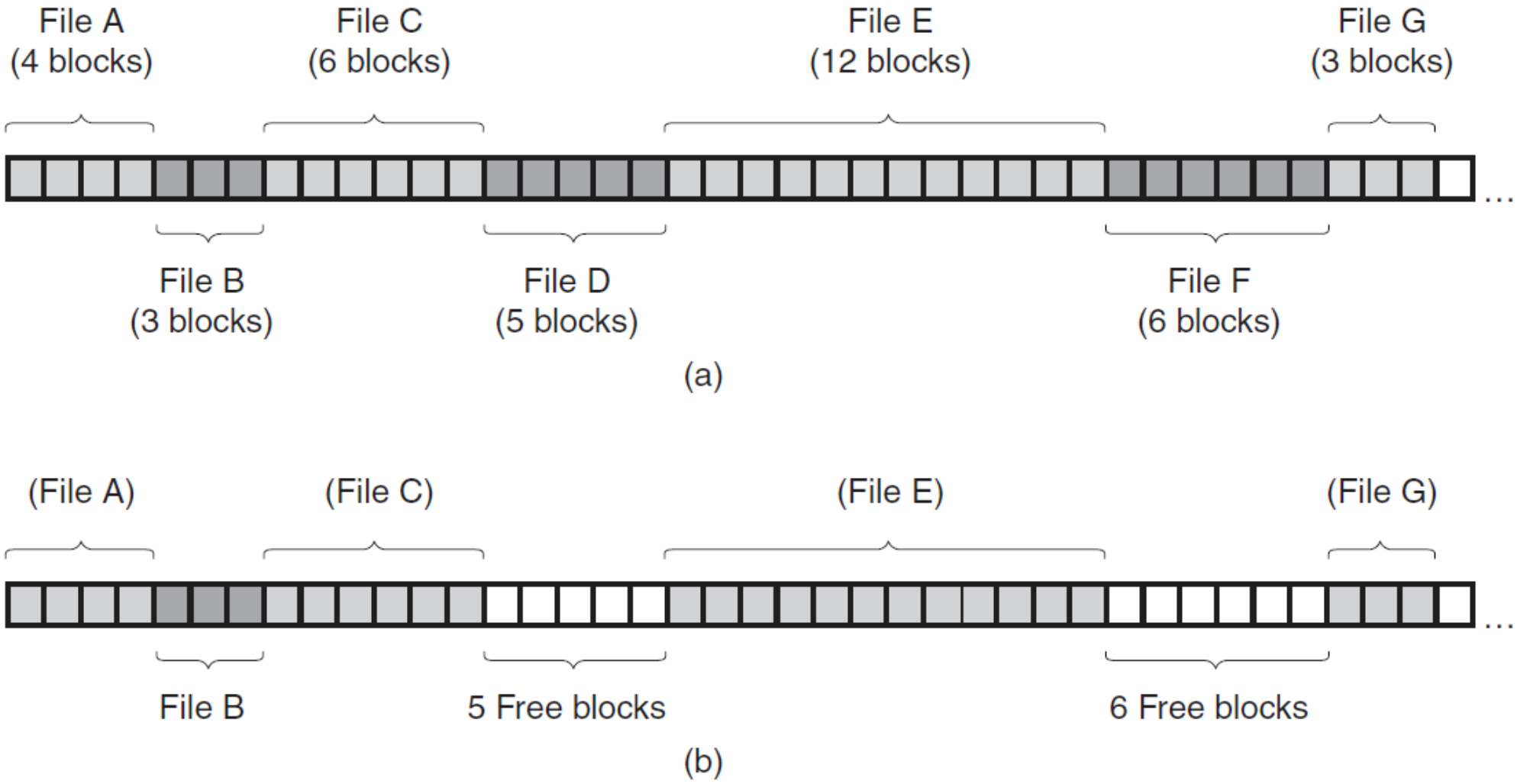
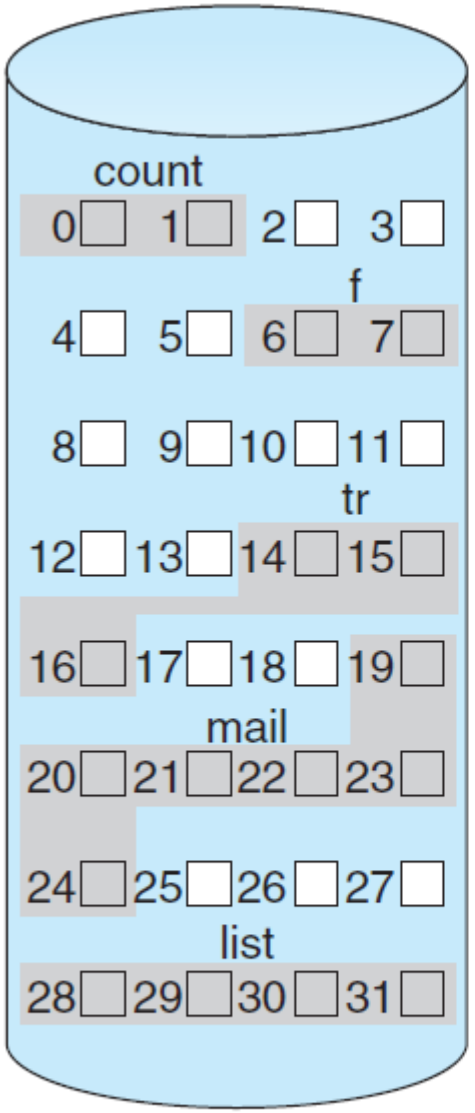


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

MOS4E

Implementing Files: Contiguous Allocation

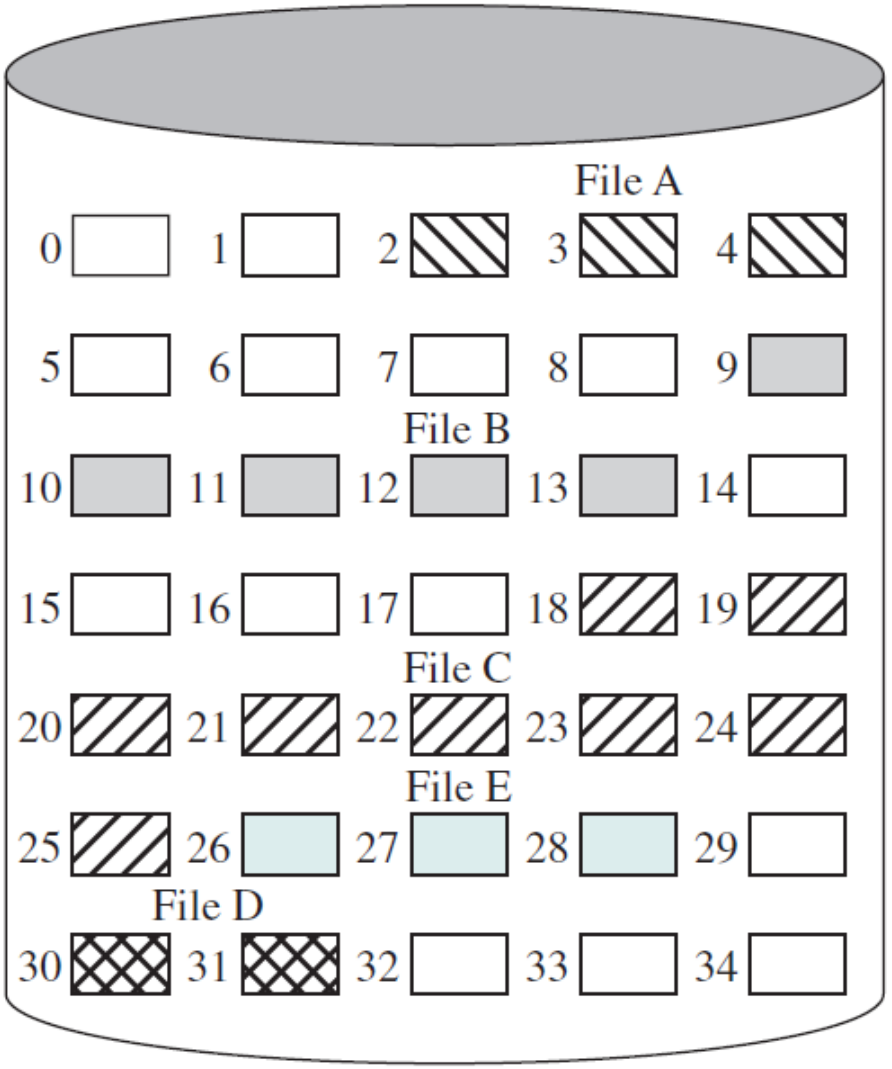


directory

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

Figure 12.5 Contiguous allocation of disk space.

Implementing Files: Contiguous Allocation



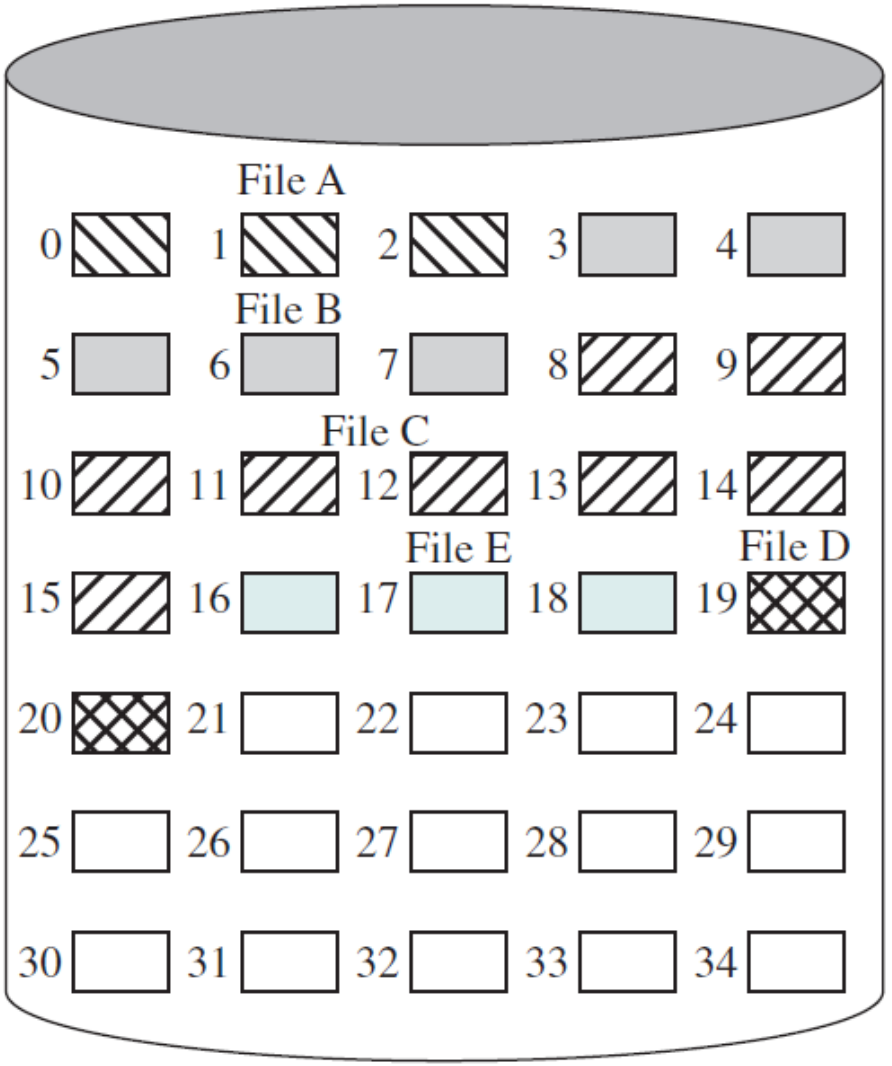
File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.9 Contiguous File Allocation

OSIDP9E

Implementing Files: Contiguous Allocation



File allocation table

File name	Start block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.10 Contiguous File Allocation (After Compaction)

OSIDP9E

Implementing Files: Linked-List Allocation

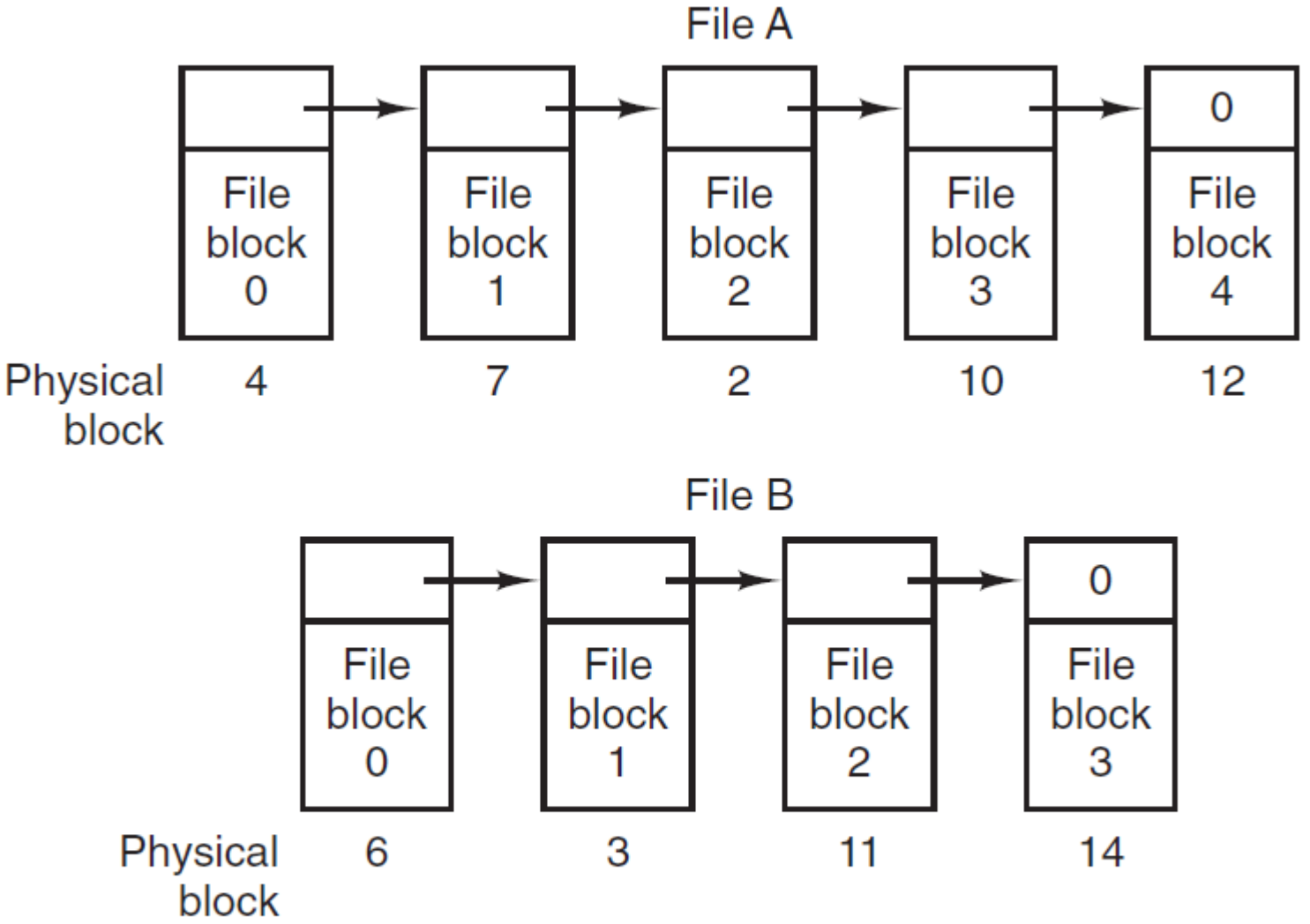


Figure 4-11. Storing a file as a linked list of disk blocks.

MOS4E

Implementing Files: Linked-List Allocation

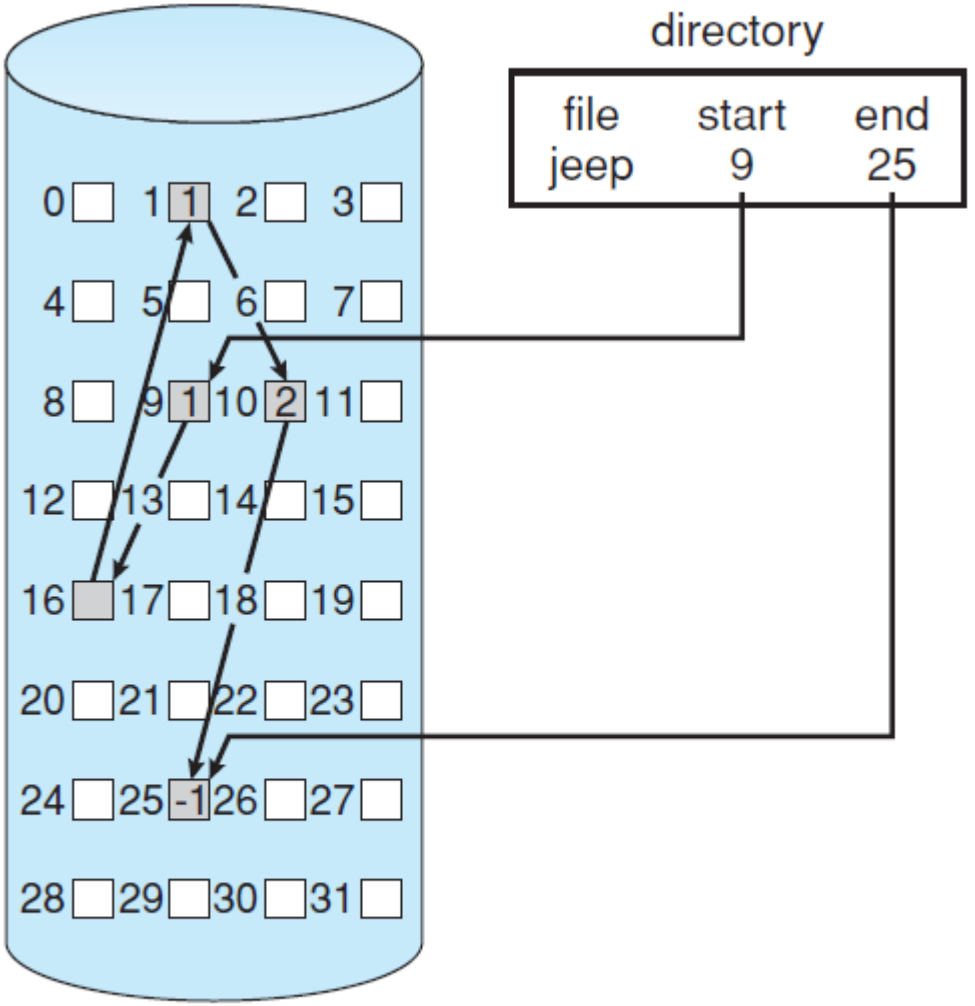
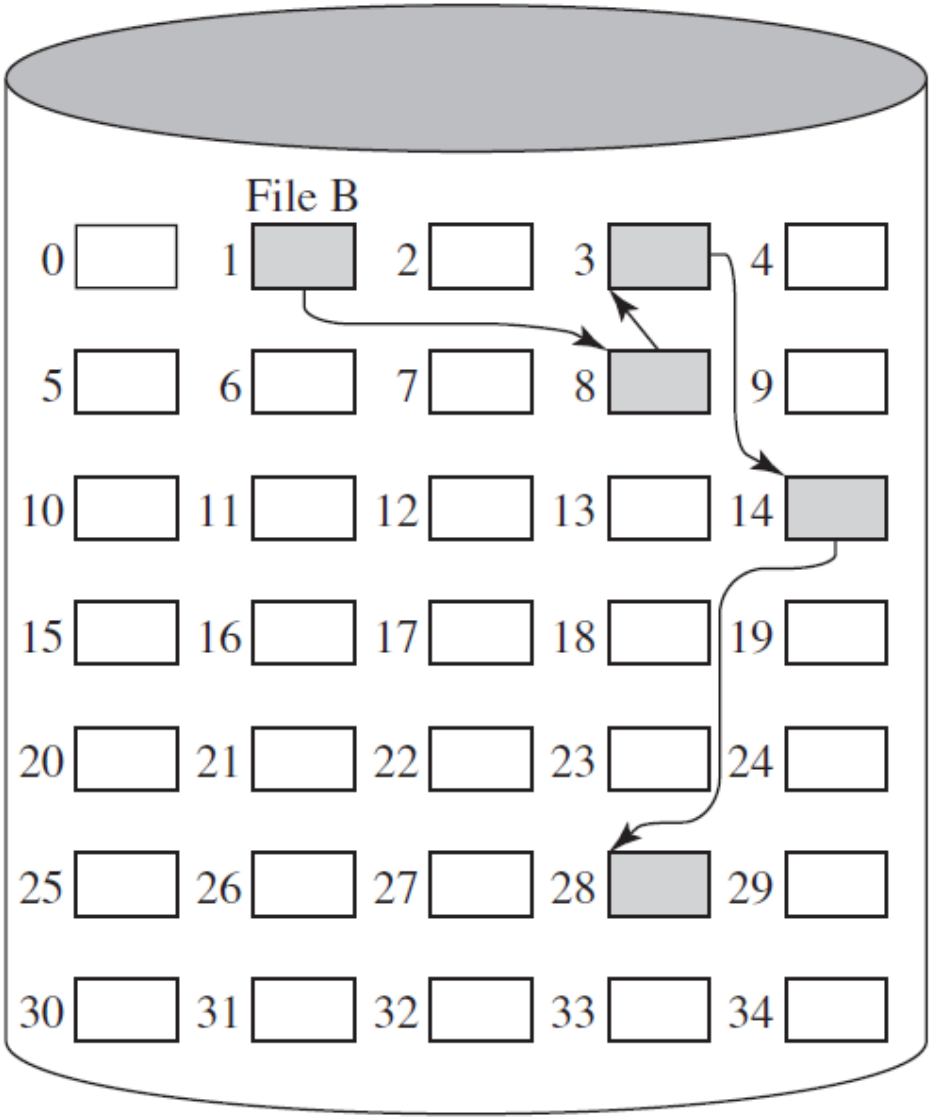


Figure 12.6 Linked allocation of disk space.

OSC9E

Implementing Files: Linked-List Allocation



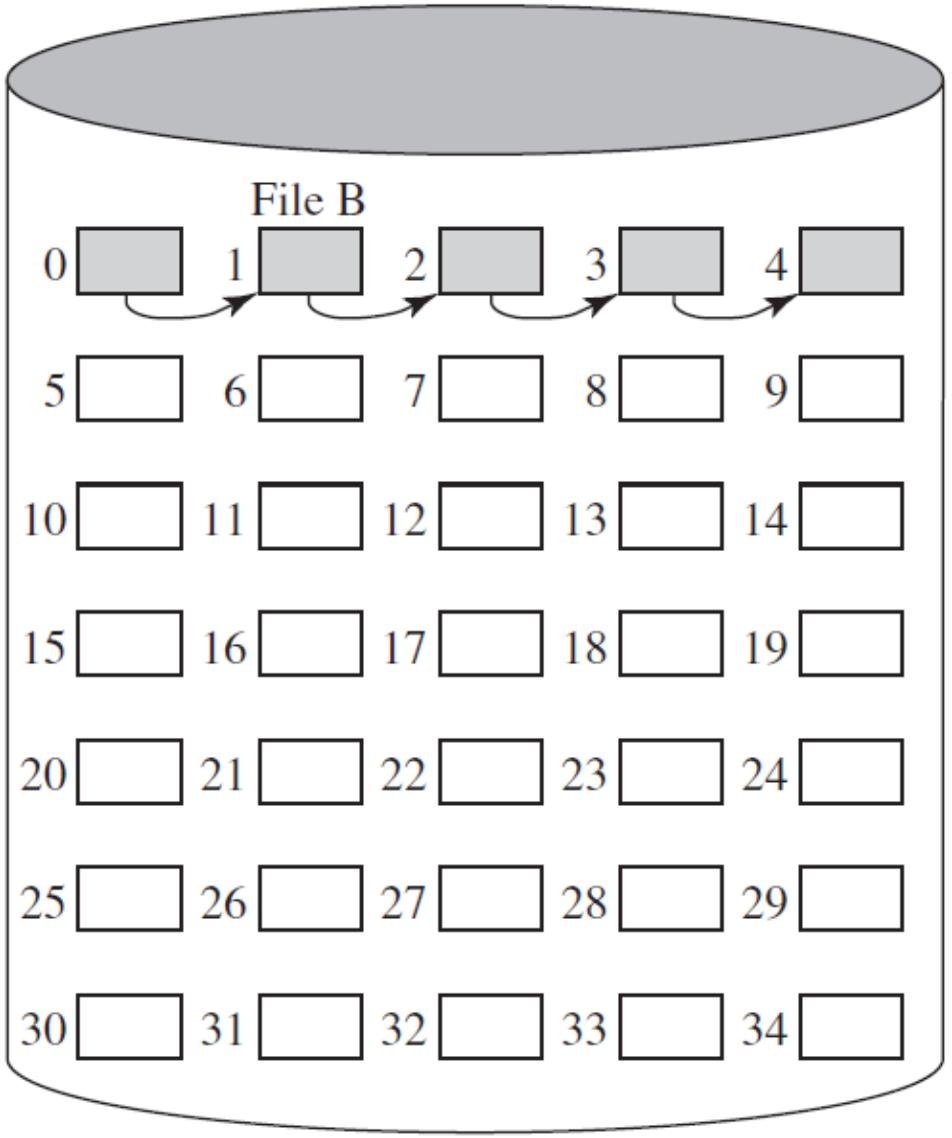
File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.11 Chained Allocation

OSIDP9E

Implementing Files: Linked-List Allocation



File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.12 Chained Allocation (After Consolidation)

OSIDP9E

Implementing Files:
Linked-List Allocation Using a Table in Memory

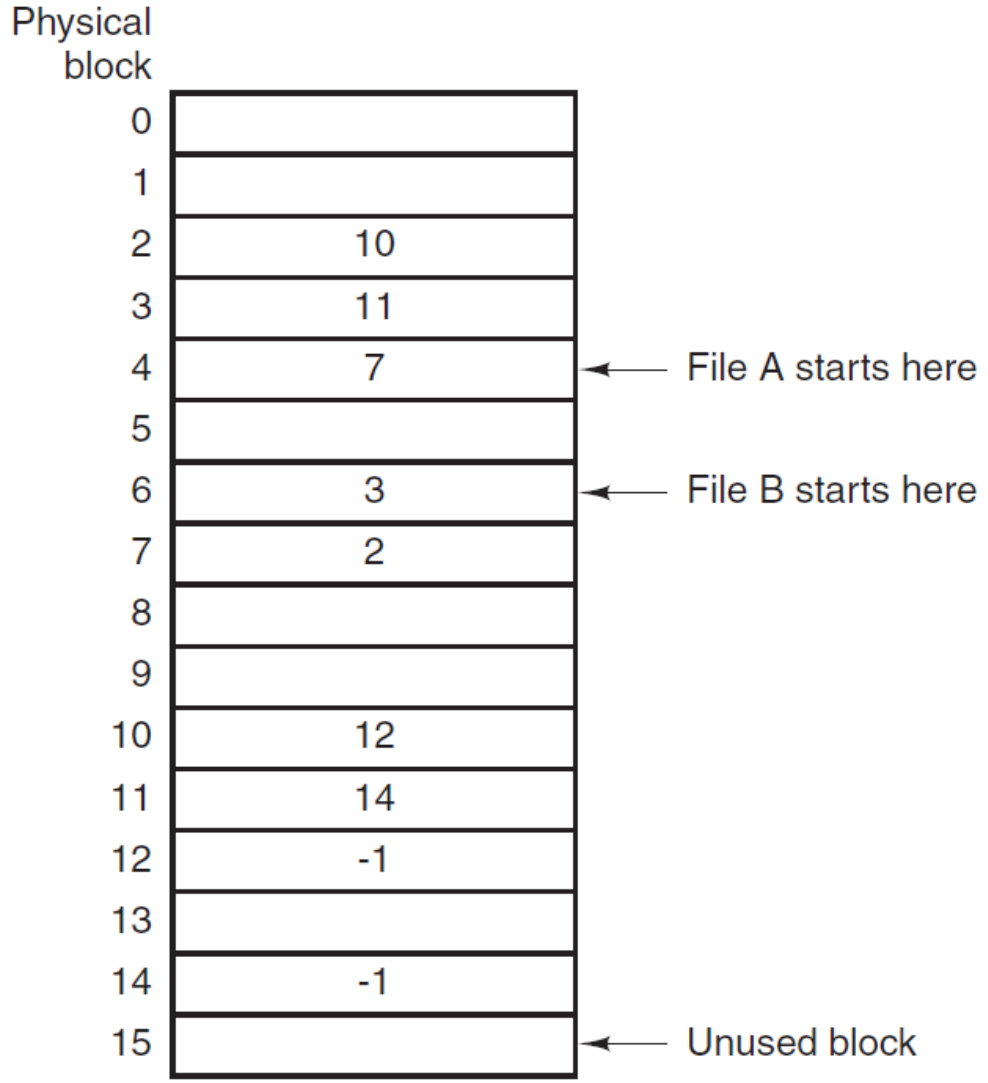


Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

MOS4E

Implementing Files:
Linked-List Allocation Using a Table in Memory

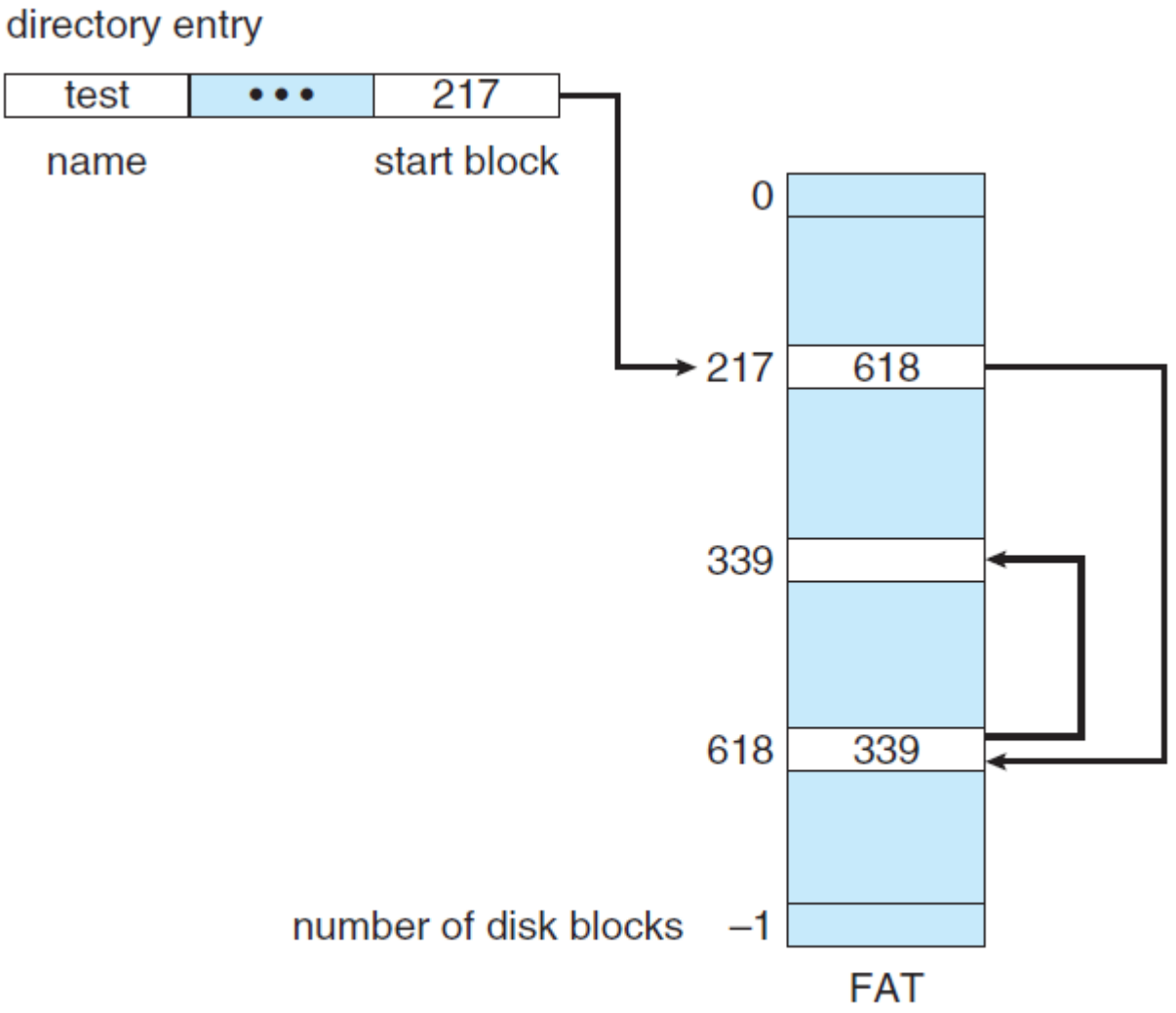


Figure 12.7 File-allocation table.

OSC9E

Implementing Files: Indexed Allocation

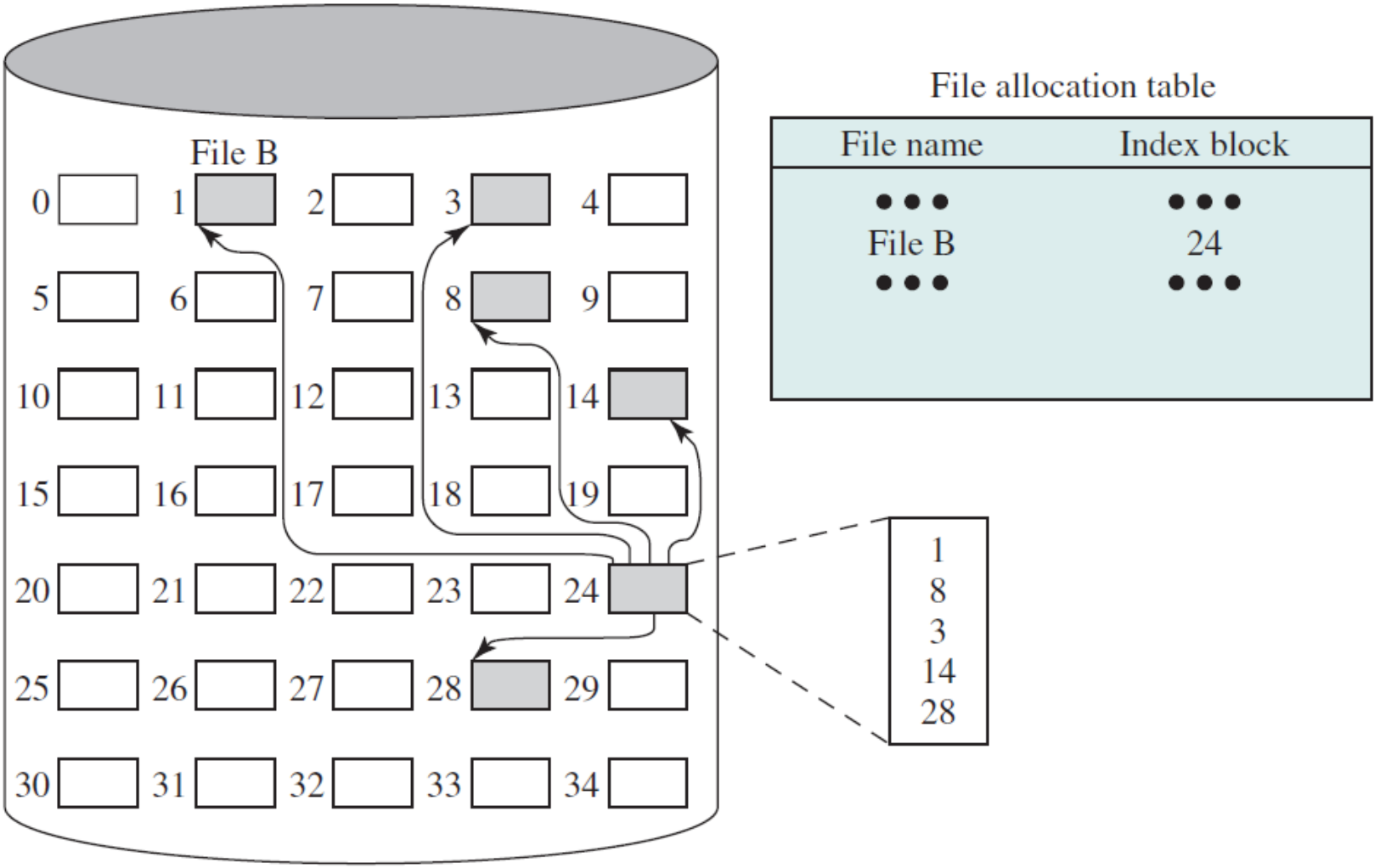


Figure 12.13 Indexed Allocation with Block Portions

OSIDP9E

Implementing Files: Indexed Allocation

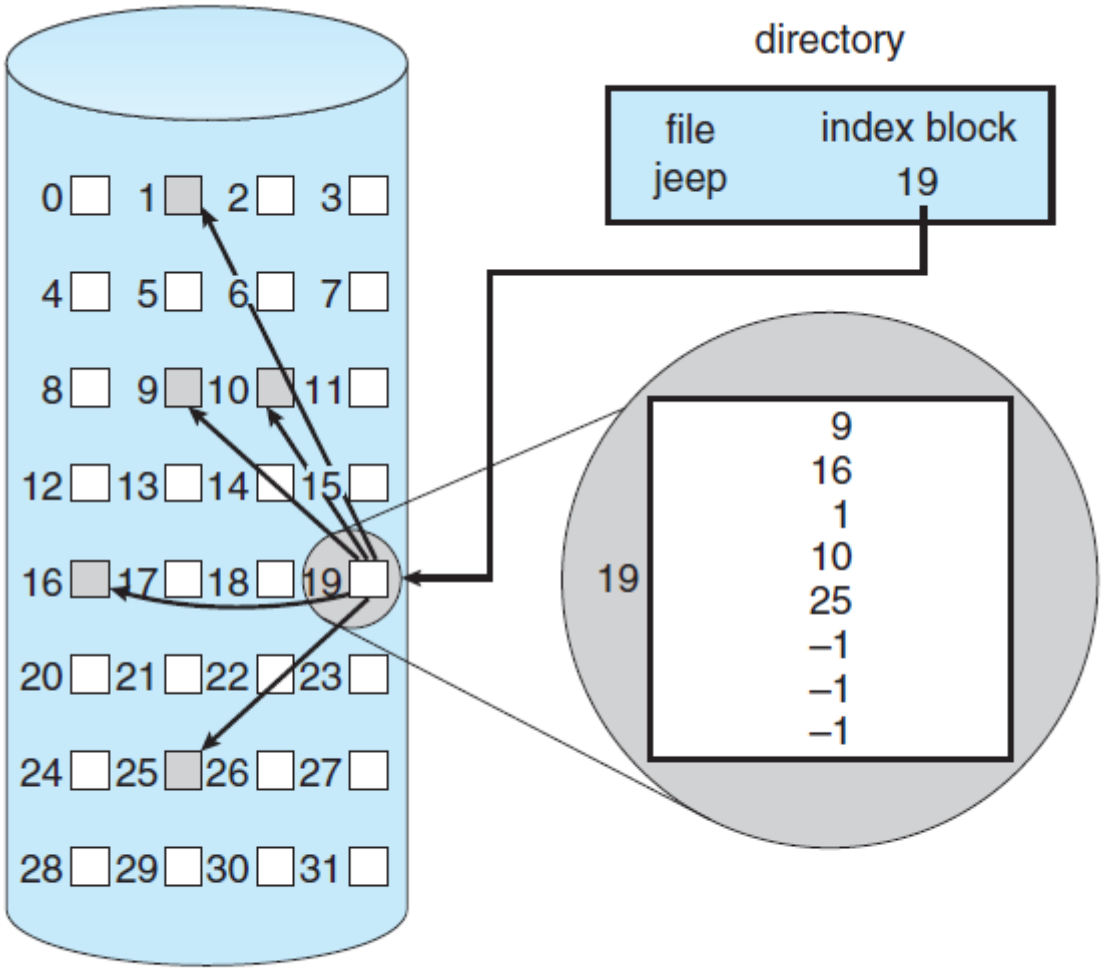
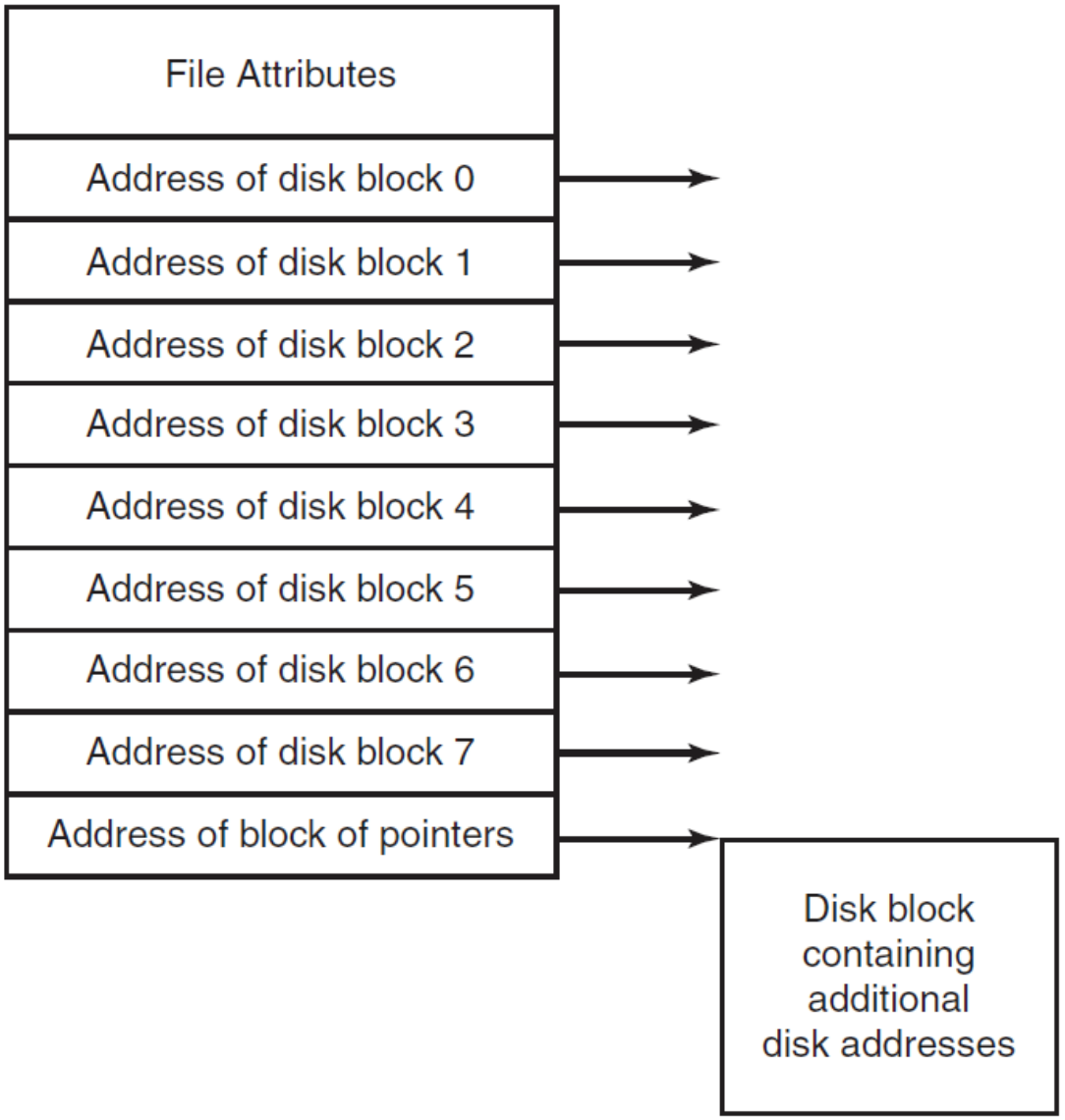


Figure 12.8 Indexed allocation of disk space.

OSC9E

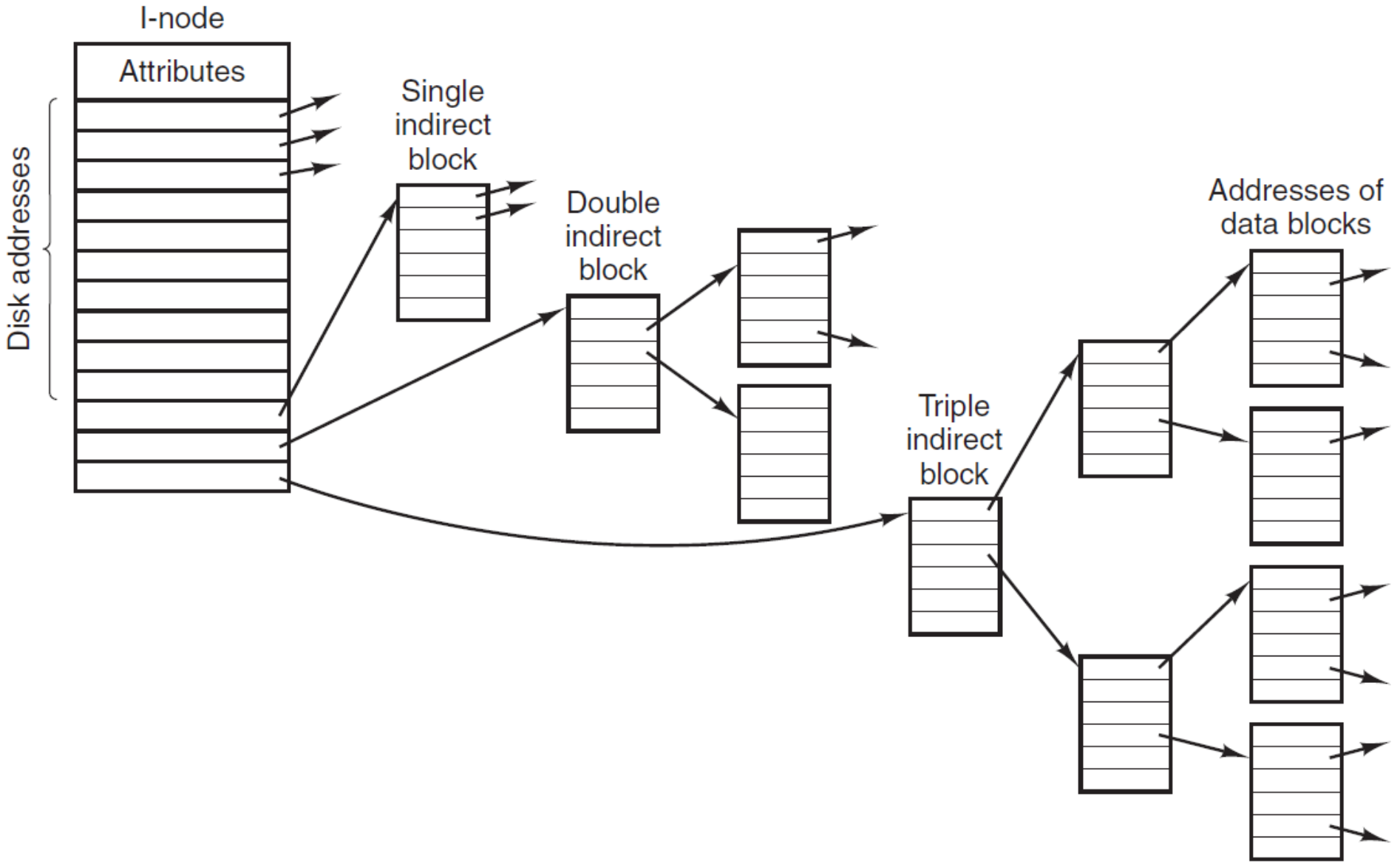
Implementing Files: Indexed Allocation



MOS4E

Figure 4-13. An example i-node.

Implementing Files: Indexed Allocation



MOS4E

Figure 4-33. A UNIX i-node.

Implementing Files: Indexed Allocation

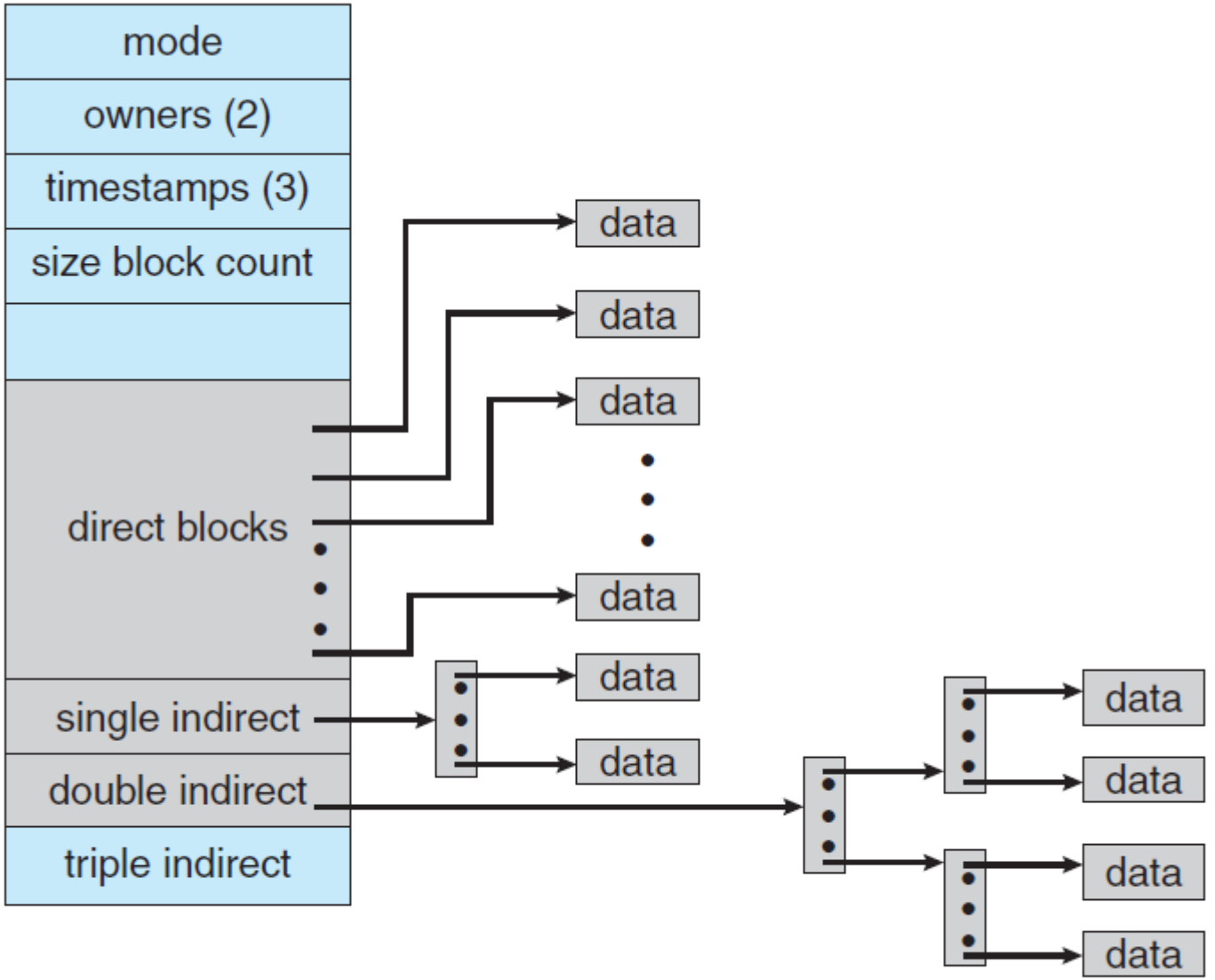
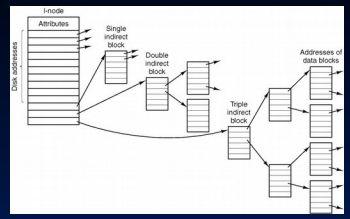


Figure 12.9 The UNIX inode.

OSC9E

“This is the house that Jack built …”
from Mother Goose

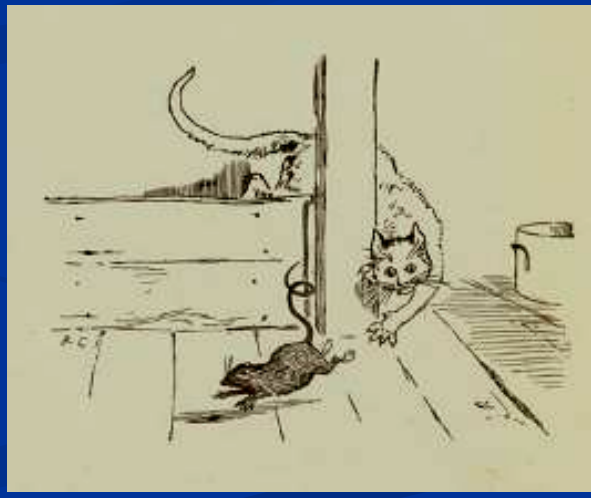


This is the house that Jack built.

This is the malt
That lay in the house that Jack built.

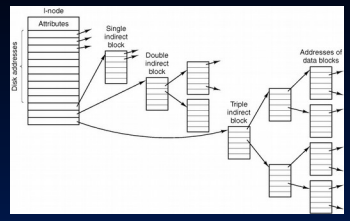
This is the rat,
That ate the malt
That lay in the house that Jack built.

This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.



Randolf Caldecott

“This is the house that Jack built …”
from Mother Goose

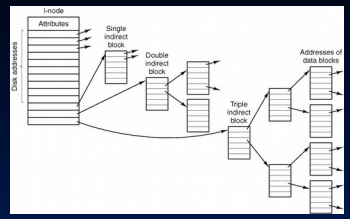


This is the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

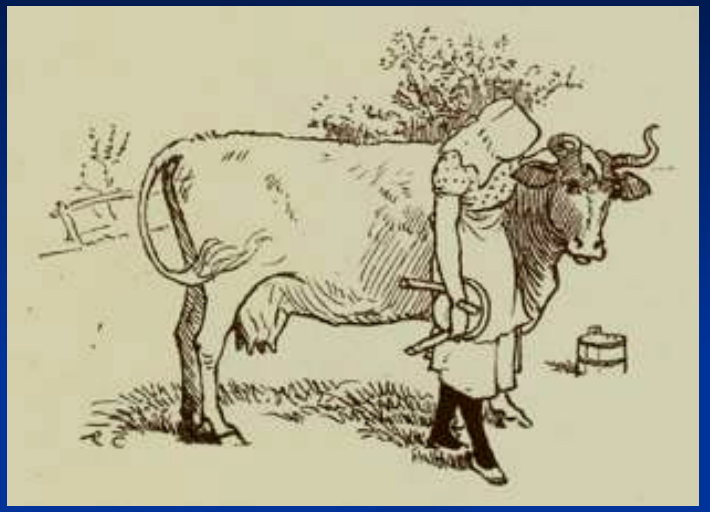
This is the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

Randolf Caldecott

“This is the house that Jack built …”
from Mother Goose



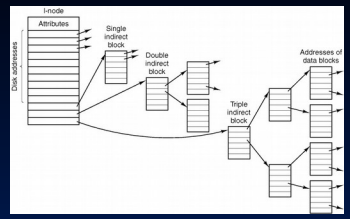
This is the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.



This is the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

Randolf Caldecott

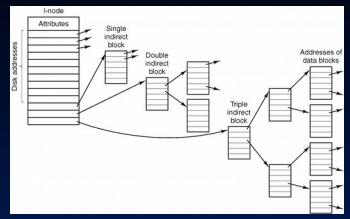
“This is the house that Jack built …”
from Mother Goose



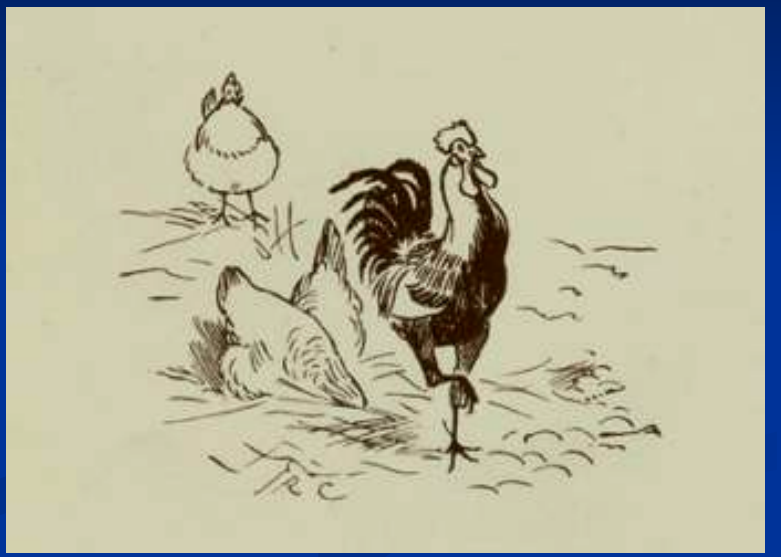
This is the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

Randolf Caldecott

“This is the house that Jack built …”
from Mother Goose

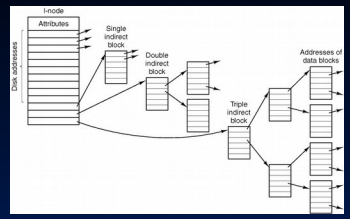


This is the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.



Randolf Caldecott

“This is the house that Jack built …”
from Mother Goose



This is the farmer sowing his corn,
That kept the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

Randolf Caldecott

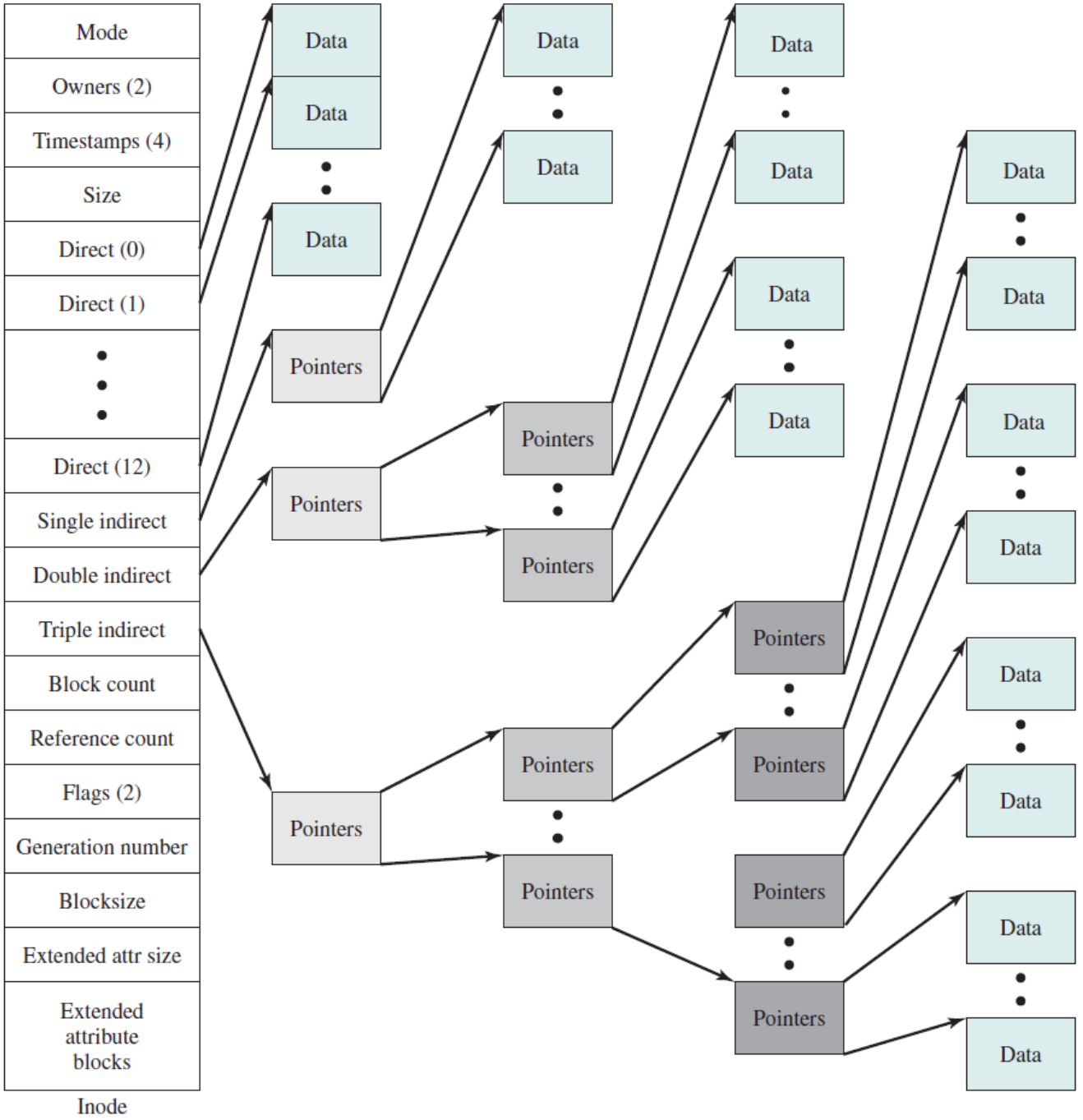


Figure 12.15 **Structure of FreeBSD Inode and File**

OSIDP9E

Implementing Directories

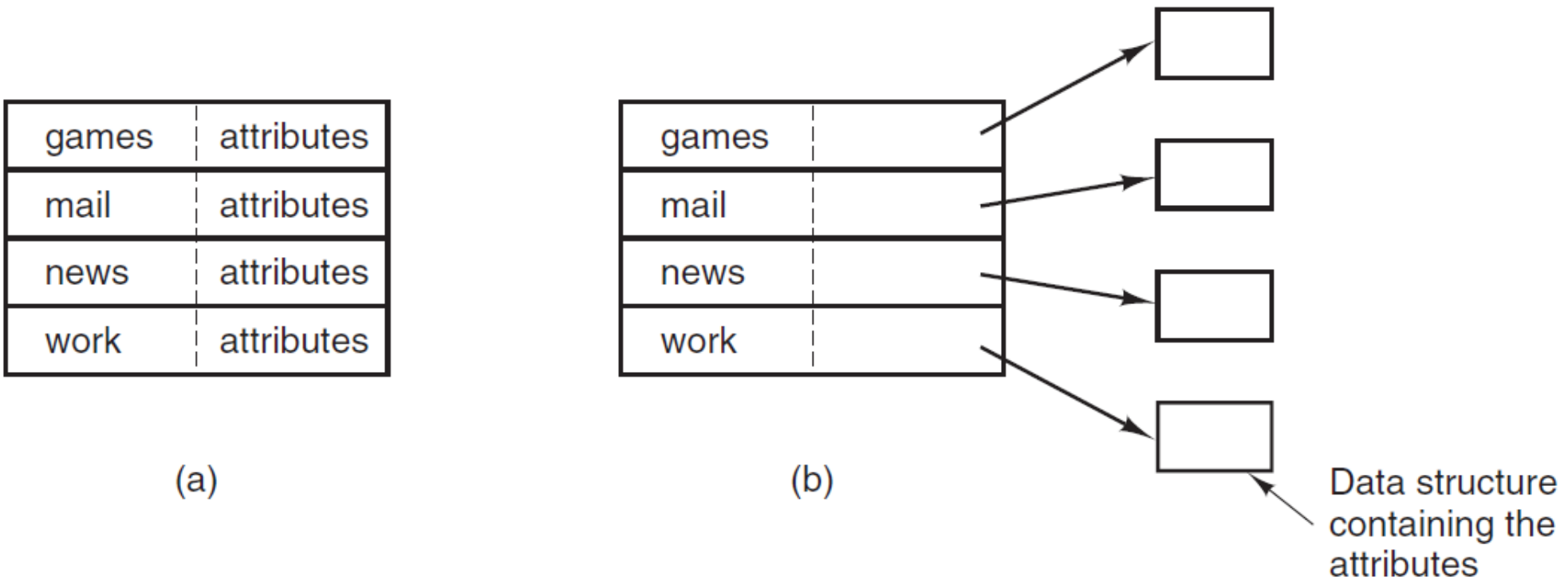


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

MOS4E

Implementing Directories

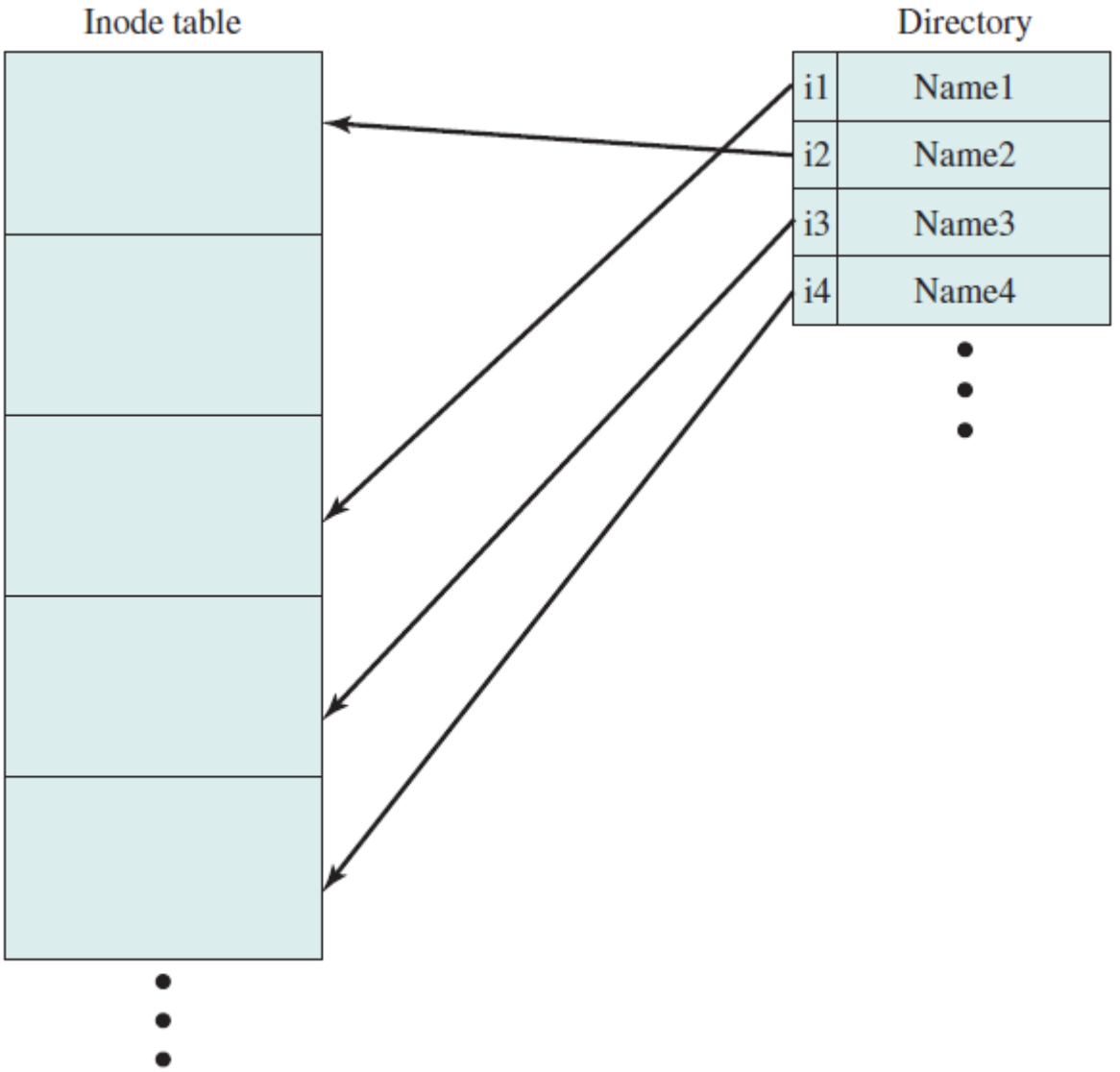


Figure 12.16 UNIX Directories and Inodes

OSIDP9E

Implementing Directories

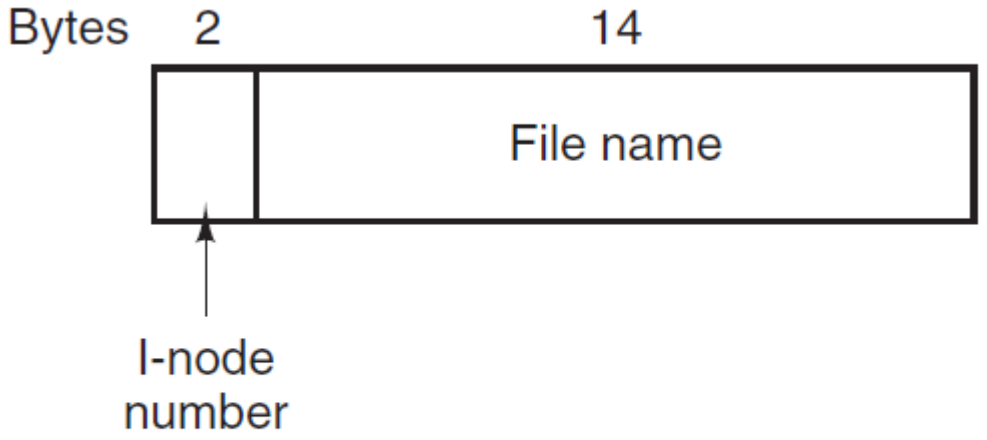


Figure 4-32. A UNIX V7 directory entry.

MOS4E

Virtual File Systems

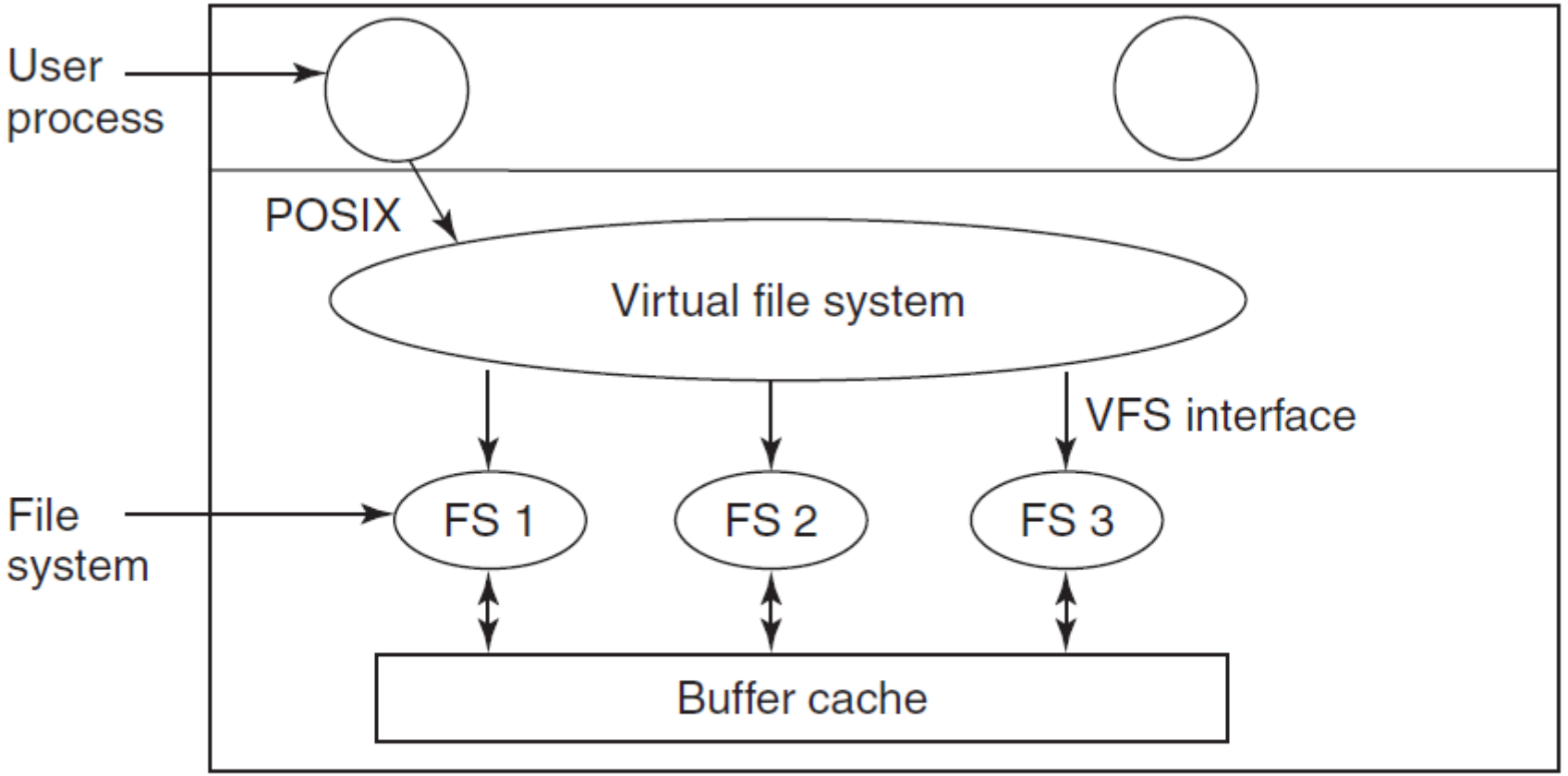


Figure 4-18. Position of the virtual file system.

MOS4E

Virtual File Systems

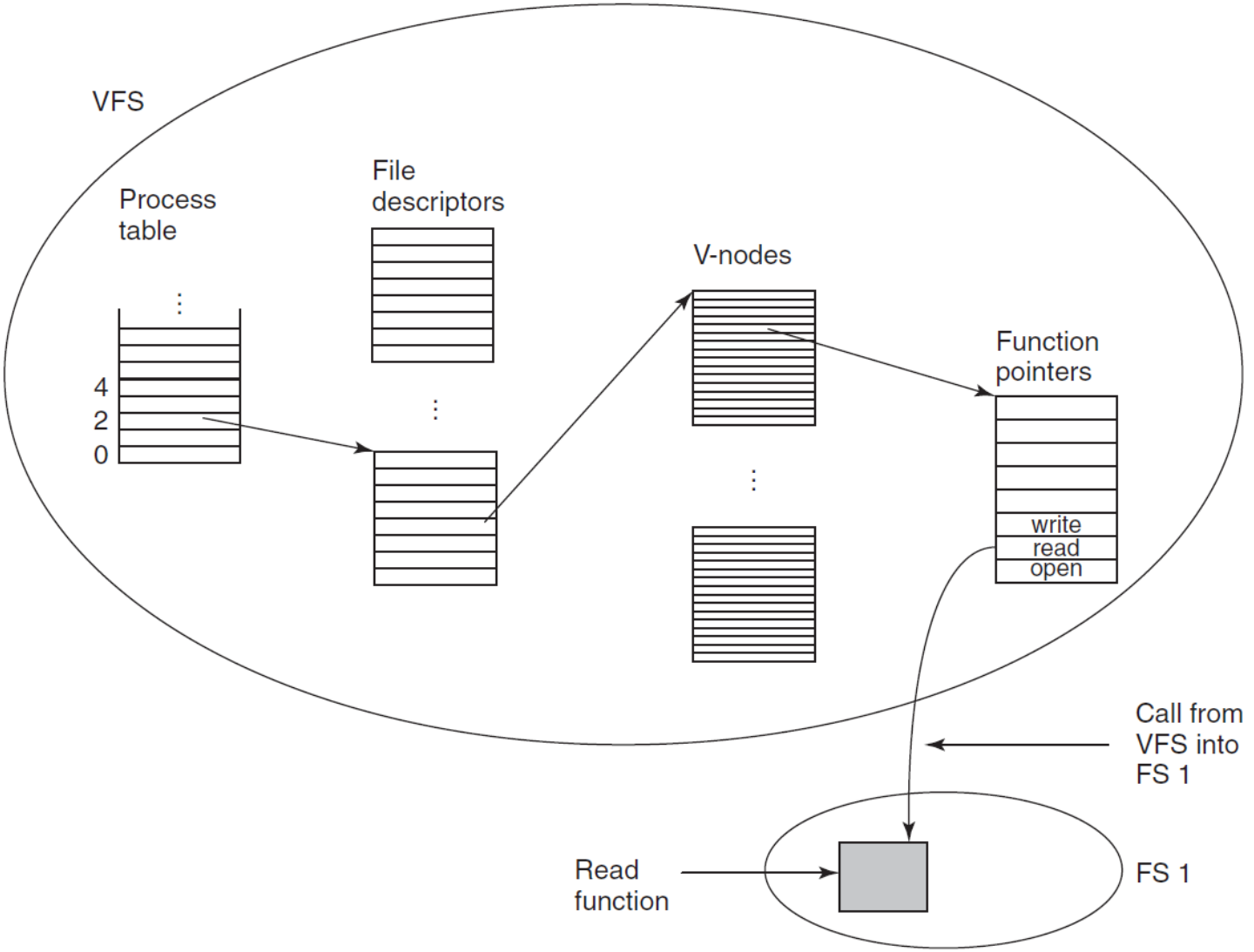


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

MOS4E

Virtual File Systems

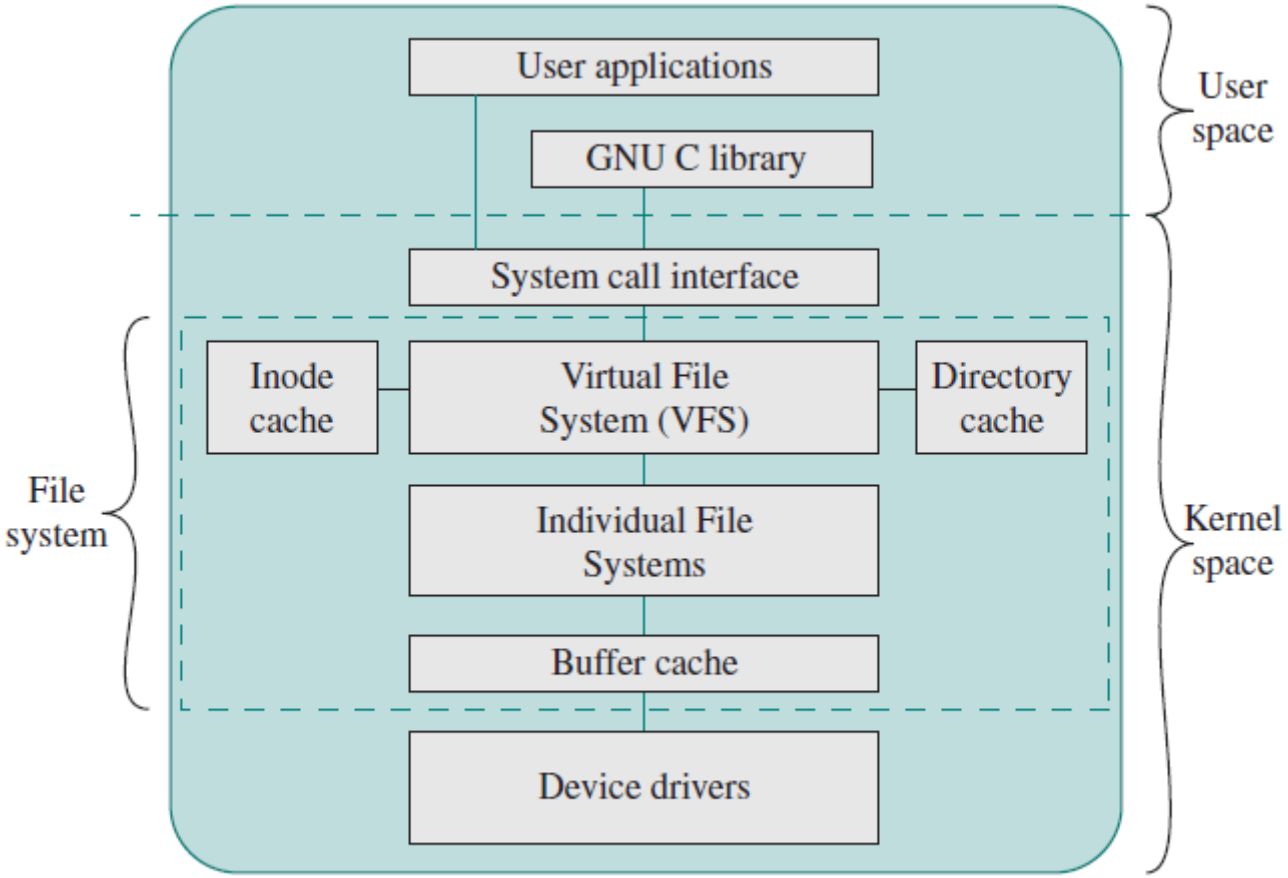


Figure 12.17 Linux Virtual File System Context

OSIDP9E

Virtual File Systems

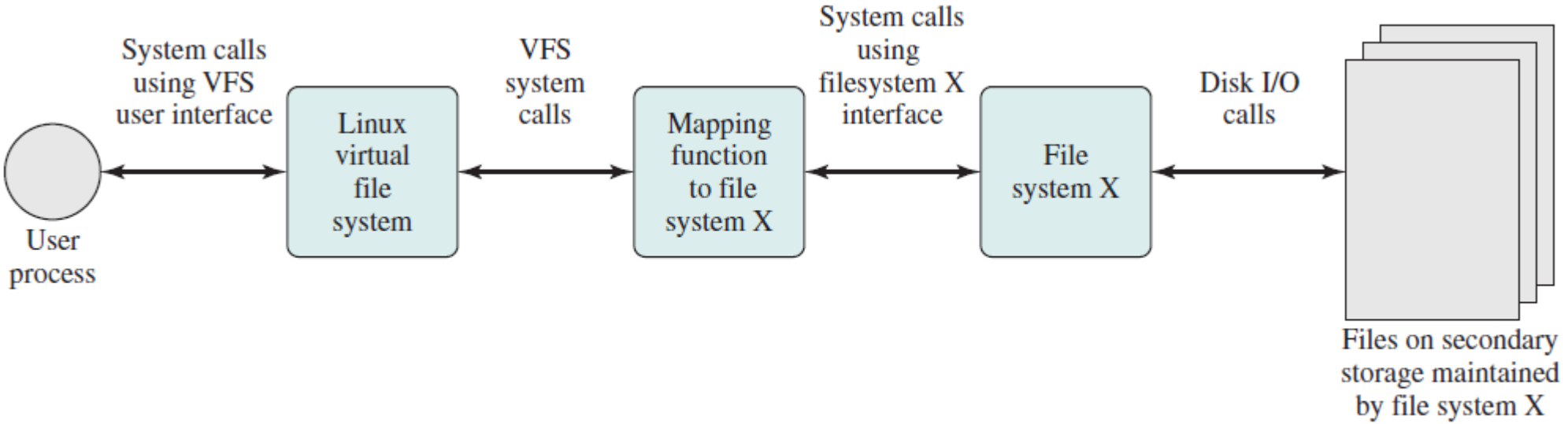


Figure 12.18 **Linux Virtual File System Concept**

OSIDP9E

Keeping Track of Free Blocks

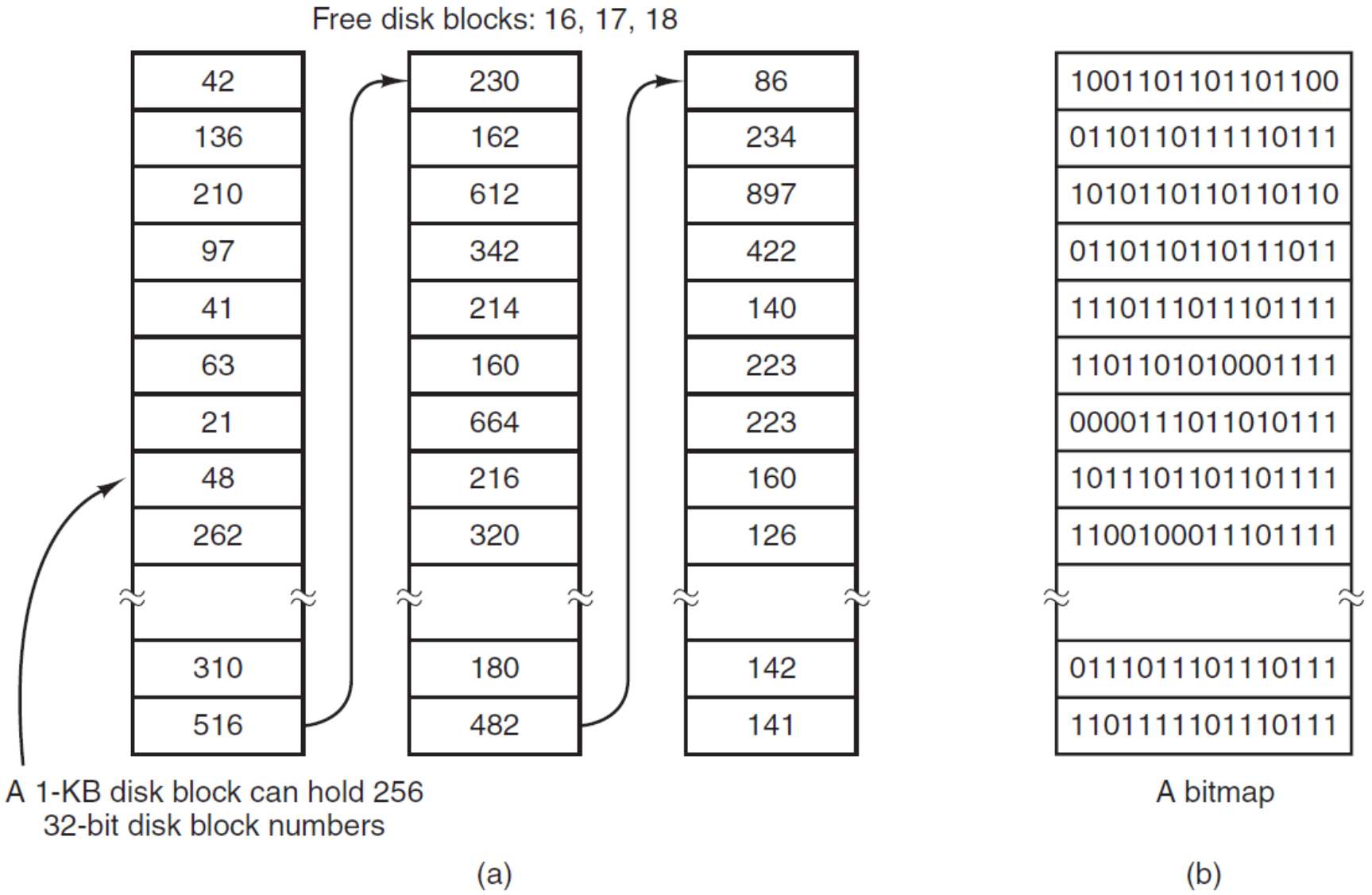


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

MOS4E

Fast File System

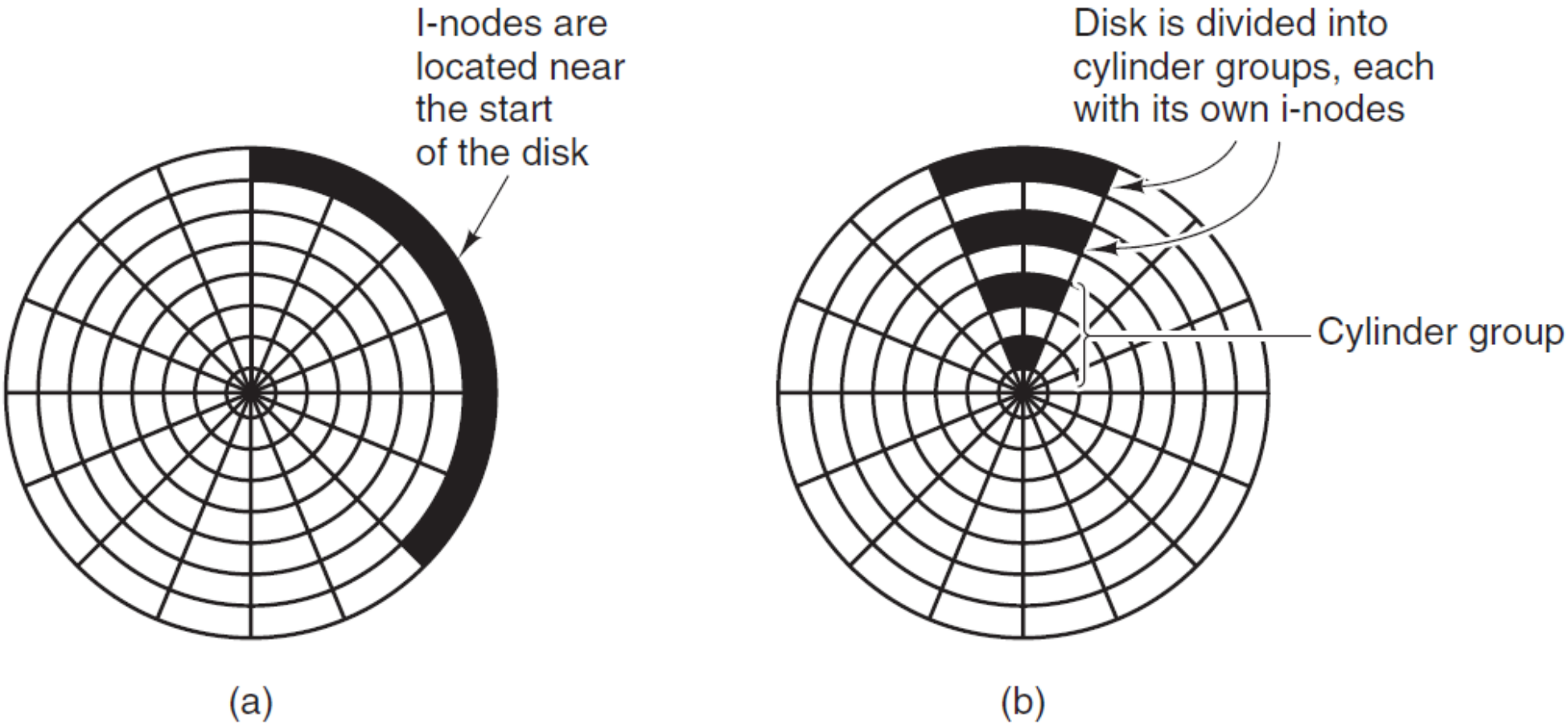


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

MOS4E

The Linux Ext2 File System

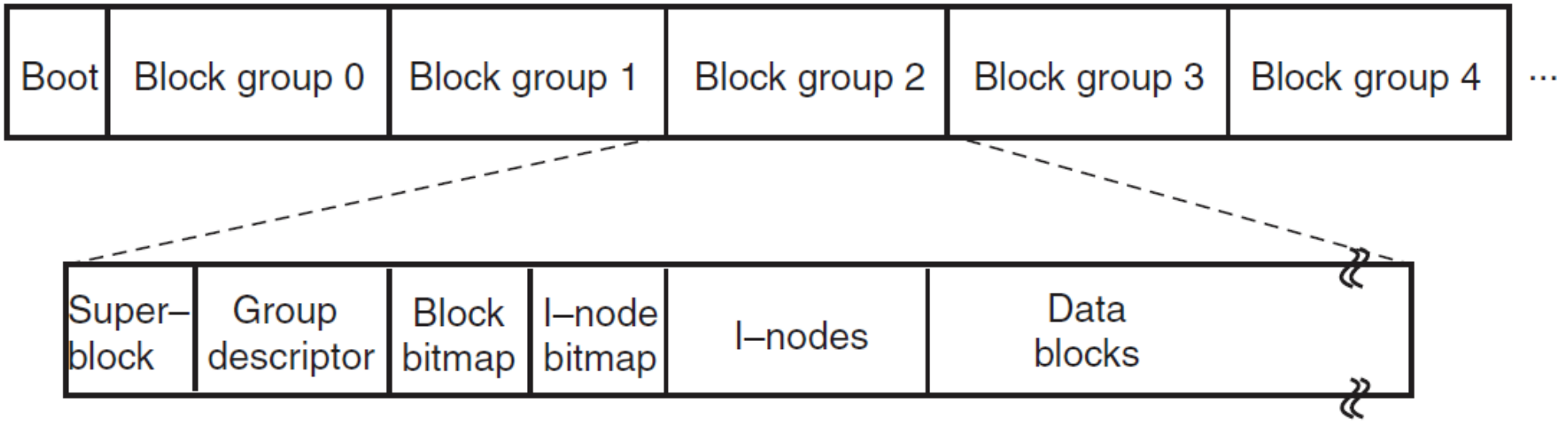


Figure 10-31. Disk layout of the Linux ext2 file system.

MOS4E

The Linux Ext2 File System

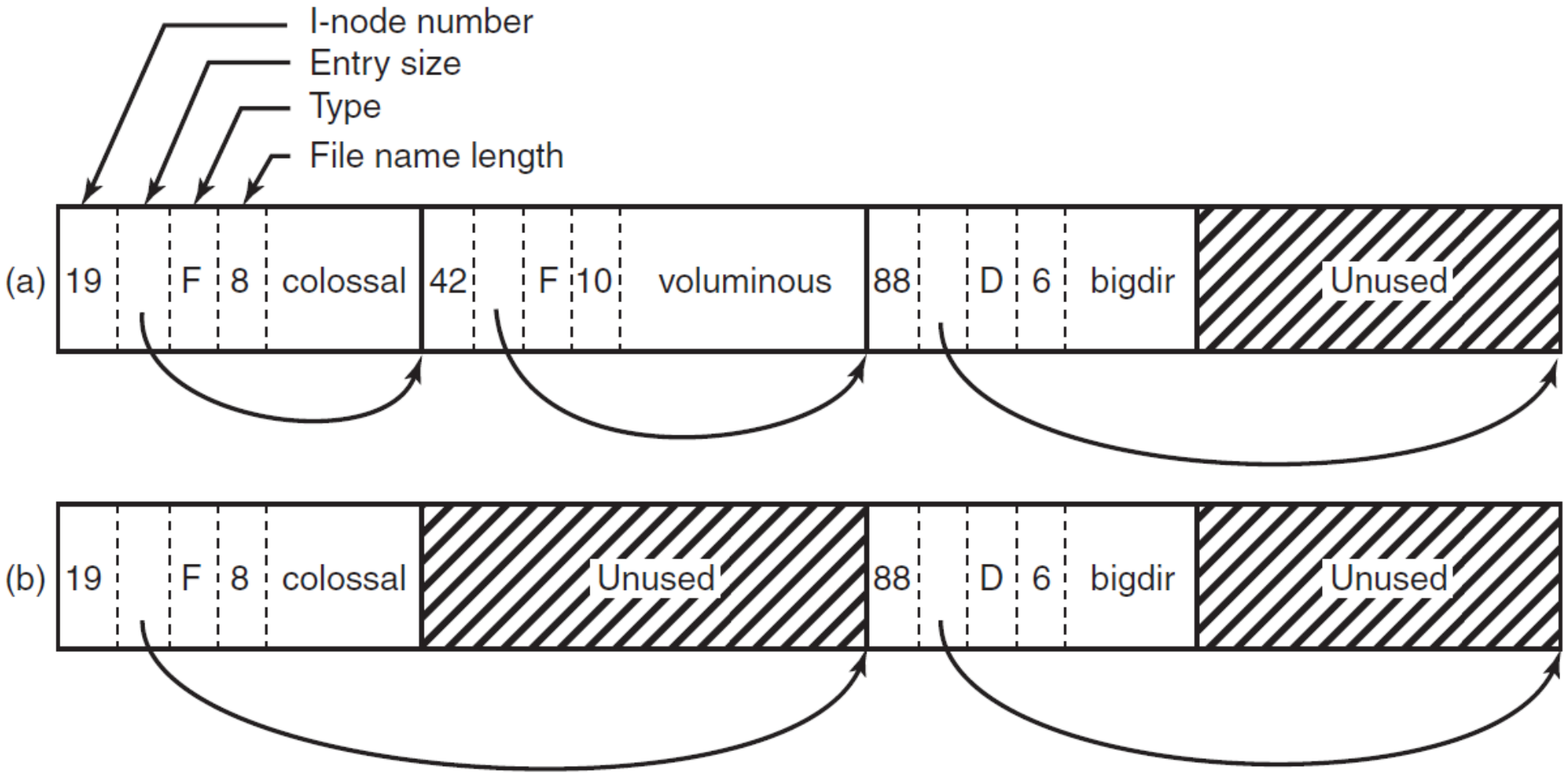


Figure 10-32. (a) A Linux directory with three files. (b) The same directory after the file *voluminous* has been removed.

MOS4E

The Linux Ext2 File System

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	60	Address of first 12 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Figure 10-33. Some fields in the i-node structure in Linux.

MOS4E