File Systems

Chapter 4

(Humour me)

What are the differences between e.g. RAM and a hard disk?

File Systems (1)

Essential requirements for long-term information storage:

1. It must be possible to store a very large amount of information.
2. Information must survive termination of process using it.
3. Multiple processes must be able to access information concurrently.

Interjection: thought experiment

Consider how a single program might store data on a device without any externally-managed organization.

How could we accomplish this?
What would be the use case?

(Not really a thought experiment, per se)

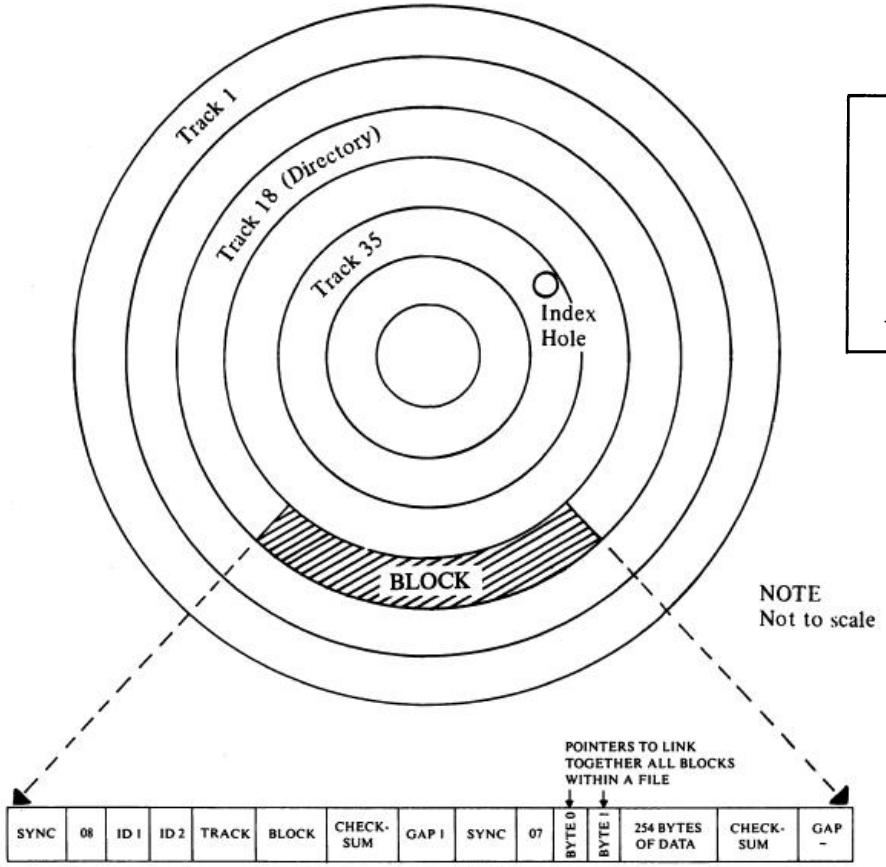
https://commons.wikimedia.org/wiki/File:Commodore-Datasette-C2N-Mk1-Front.jpg

(Not really an experiment, per se)

https://commons.wikimedia.org/wiki/File:Commodore-64-1541-Floppy-Drive-01.jpg

(more in Sakai)

APPENDIX D: DISK FORMATS



1540/1541 Format: Expanded View of a Single Sector

BLOCK DISTRIBUTION BY TRACK

Track number	Block Range	Total
1 to 17	0 to 20	21
18 to 24	0 to 18	19
25 to 30	0 to 17	18
31 to 35	0 to 16	17

1540/1541 BAM FORMAT

Track 18, Sector 0.

BYTE	CONTENTS	DEFINITION
0,1	18,01	Track and block of first directory block.
2	65	ASCII character A indicating 4040 format.
3	0	Null flag for future DOS use.
4—143		Bit map of available blocks for tracks 1—35.

*1 = available block
0 = block not available
(each bit represents one block)

File Systems (2)

Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:

1. Read block *k*.
2. Write block *k*

File Systems (3)

Questions that quickly arise:

1. How do you find information?
2. How do you keep one user from reading another user's data?
3. How do you know which blocks are free?

Files

What *is* a file?

File Naming

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.

(yes, another interjection)

- Why are extensions usually **three** characters?
  - (And how do we handle naming conventions across legacy systems?)
- But, wait, are extensions even *necessary*?

File Structure

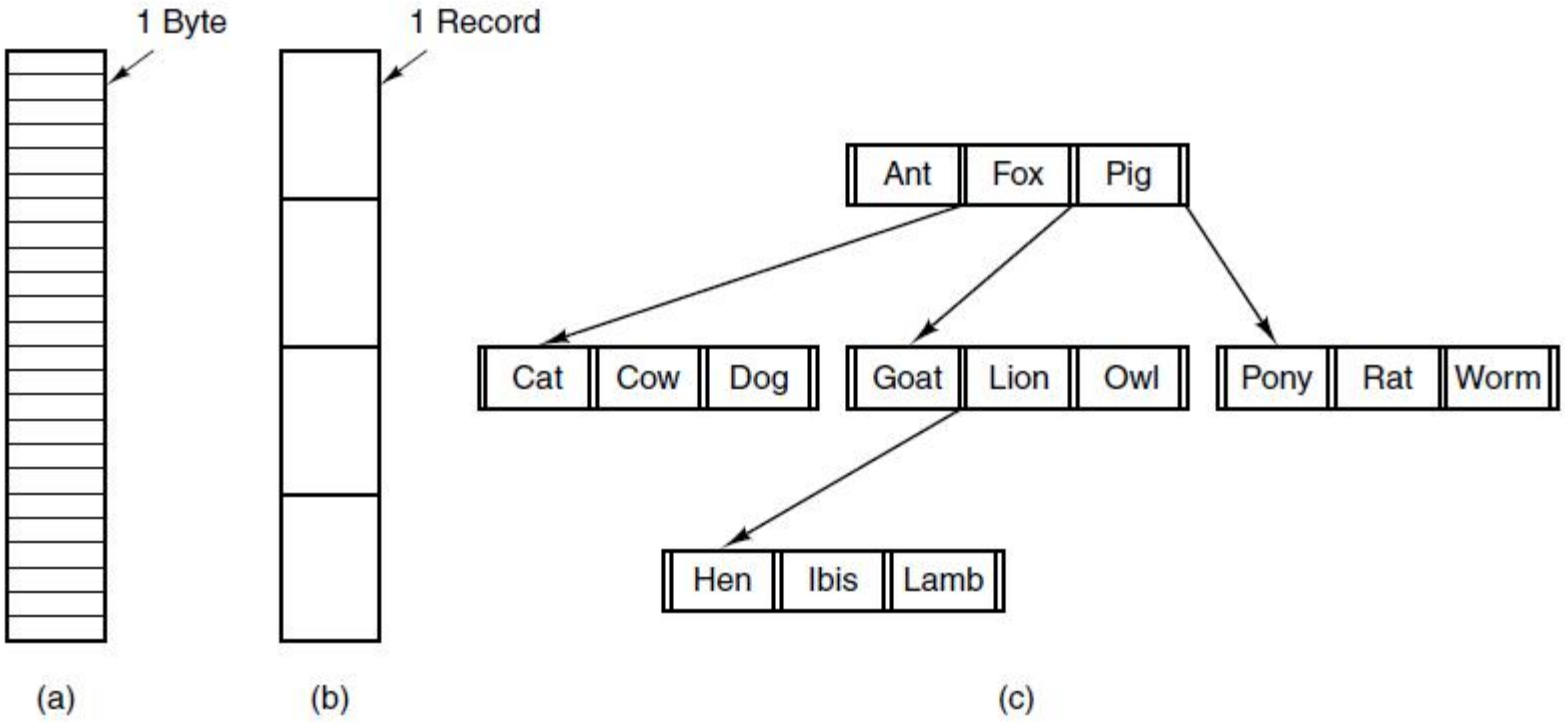


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.
General-purpose OSes use **(a)**

File Types

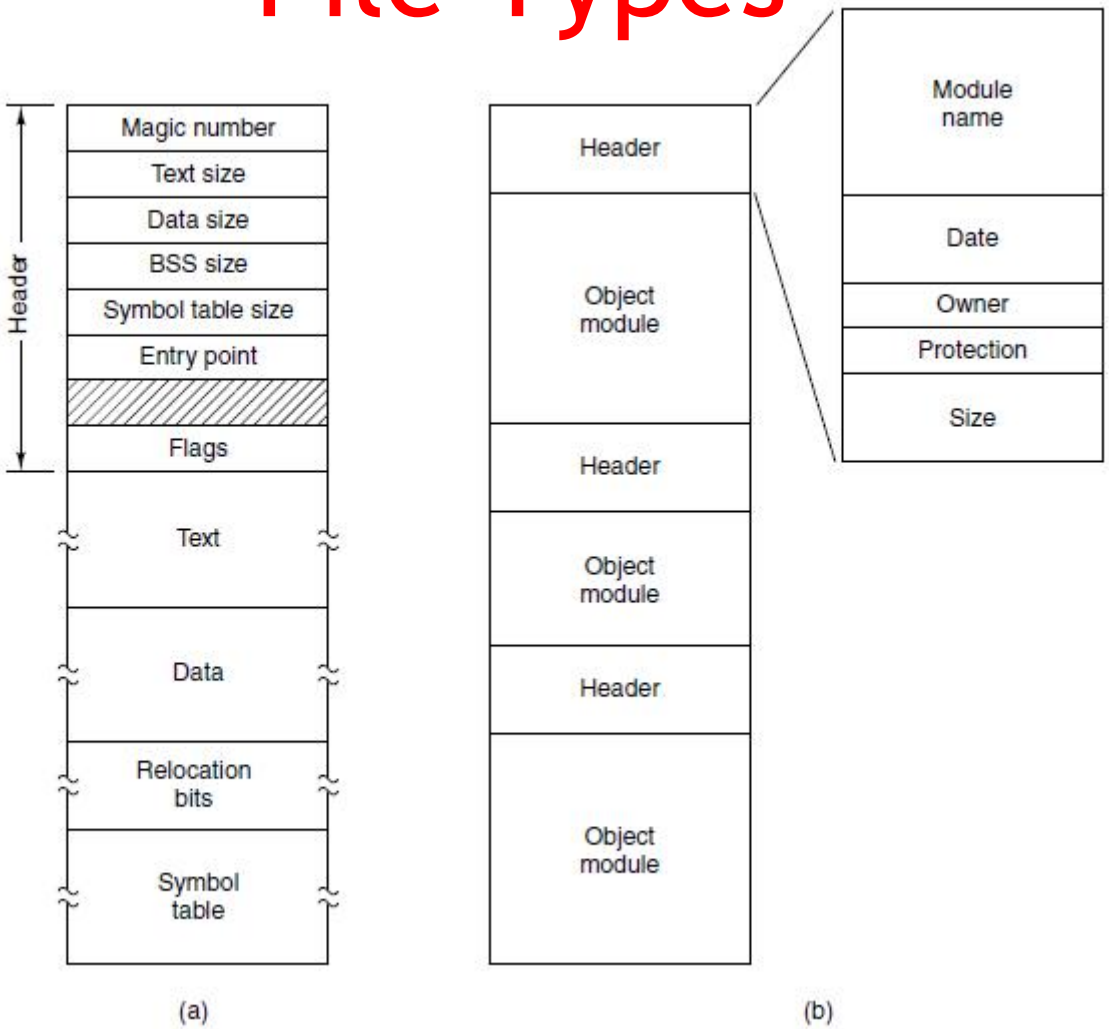


Figure 4-3. (a) An executable file. (b) An archive
https://en.wikipedia.org/wiki/List_of_file_signatures

File Attributes

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.

Where is this metadata stored?

(speaking of file attributes…)

- Uh… was the file *name* included there?

File Operations

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

Example Program Using File System Calls (1)

```
/* 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 */
```

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

Example Program Using File System Calls (2)

```
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 */
```

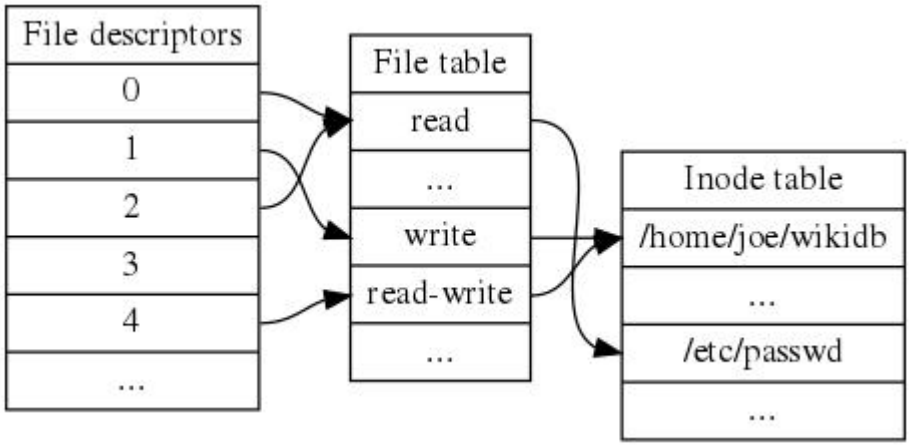
Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (3)

```
        /* 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.



https://commons.wikimedia.org/wiki/File:File_table_and_inode_table.svg

Single-Level Directory Systems

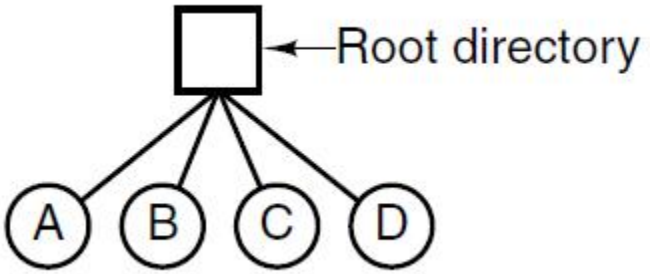


Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems

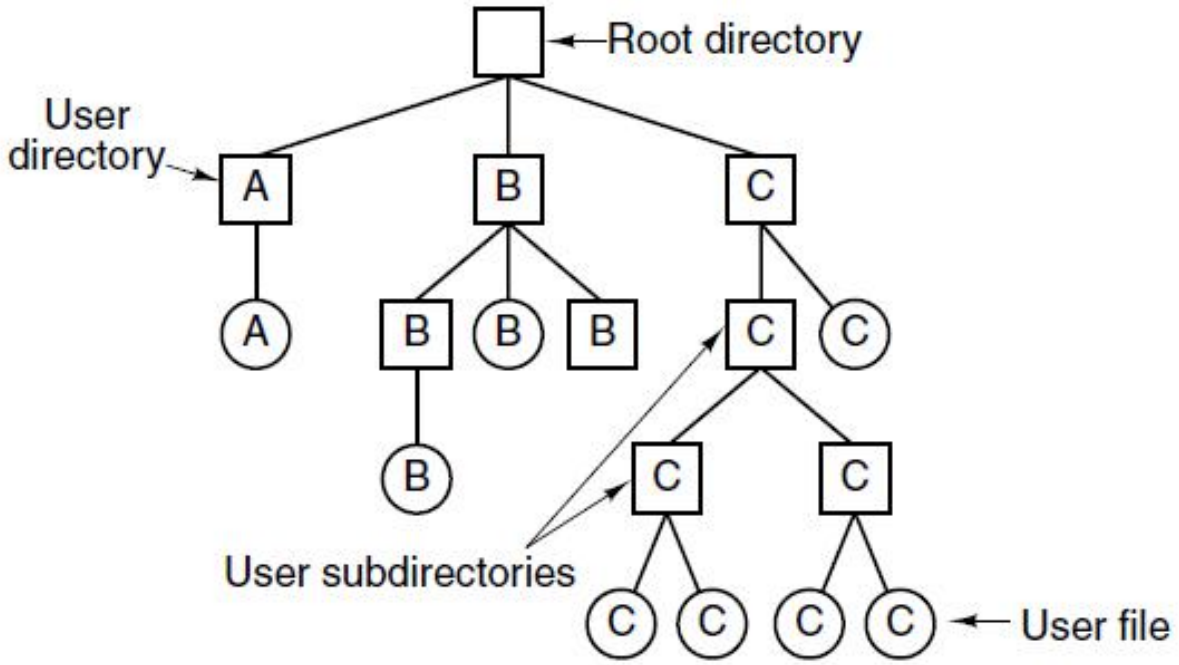


Figure 4-7. A hierarchical directory system.

Path Names

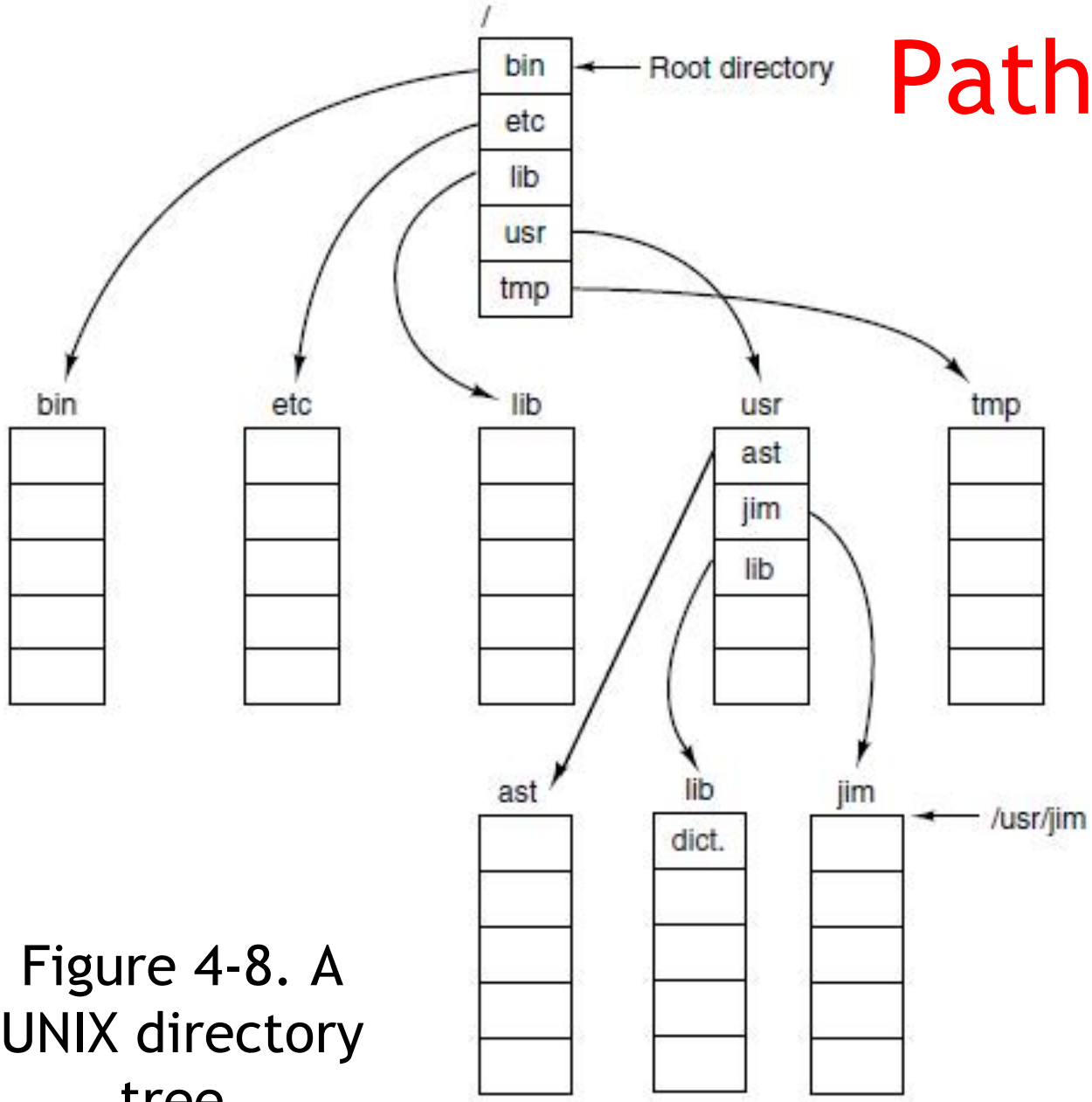


Figure 4-8. A UNIX directory tree.

Directory Operations

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

File System Layout

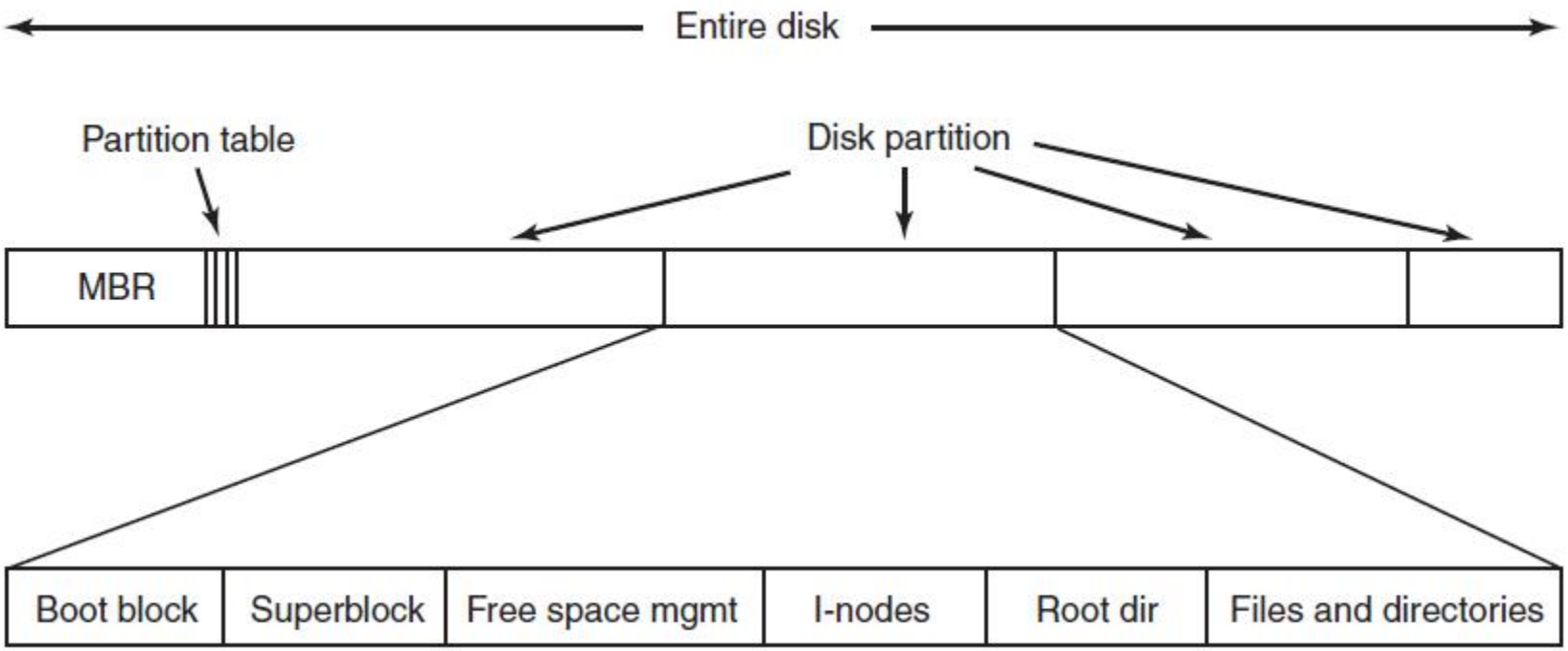


Figure 4-9. A possible file system layout.

Implementing Files
Contiguous Layout

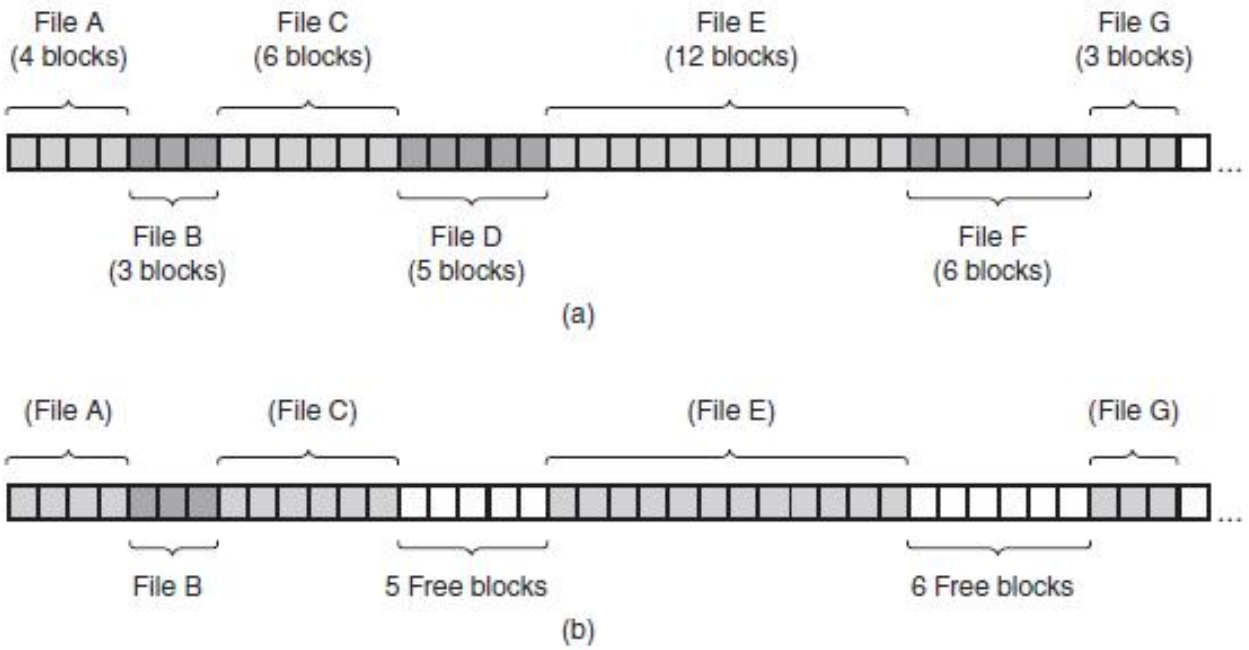


Figure 4-10. (a) Contiguous allocation of disk space for seven fil
(The simplicity and performance benefits should be obvious?)
(b) The state of the disk after files *D* and *F* have been removed

Implementing Files
Linked List Allocation

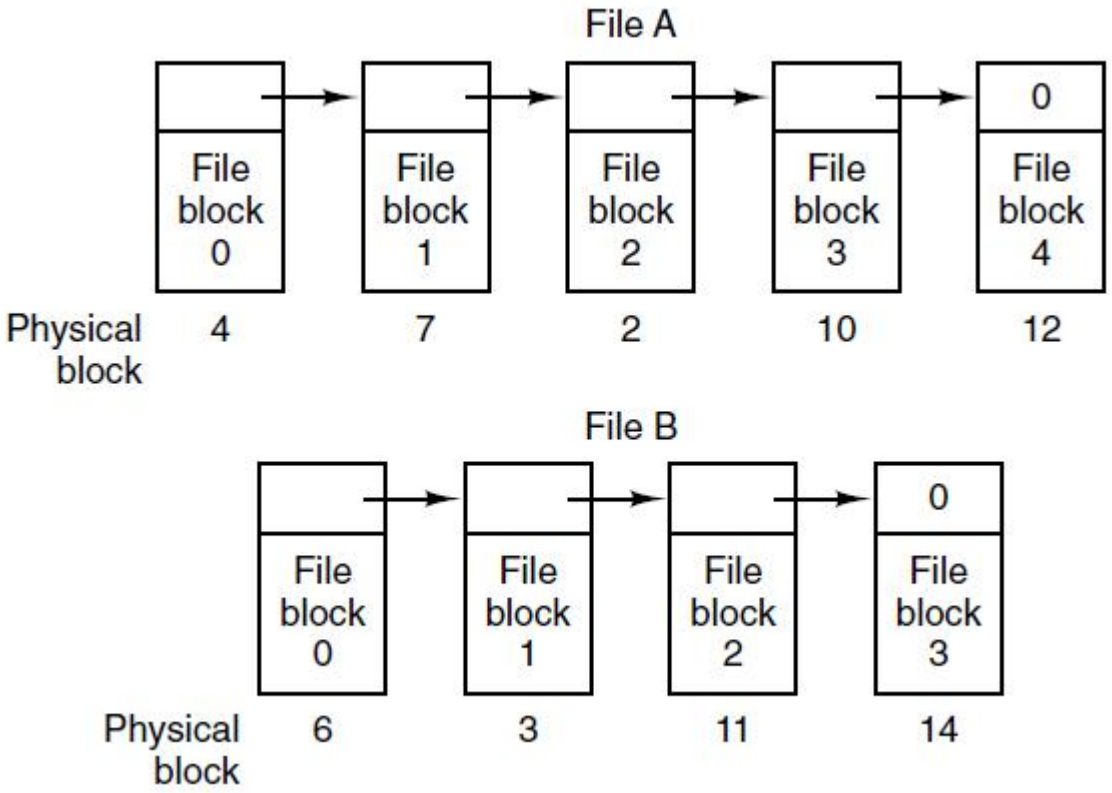


Figure 4-11. Storing a file as a linked list of disk blocks.

Implementing Files
Linked List – Table in Memory

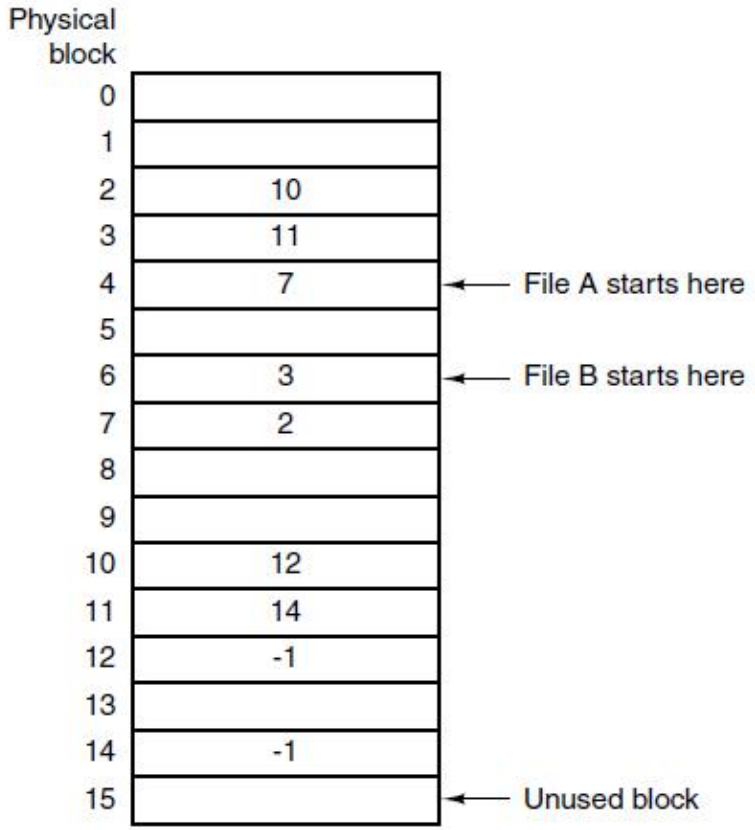


Figure 4-12. Linked list allocation using a file allocation table in main memory.

Implementing Files I-nodes

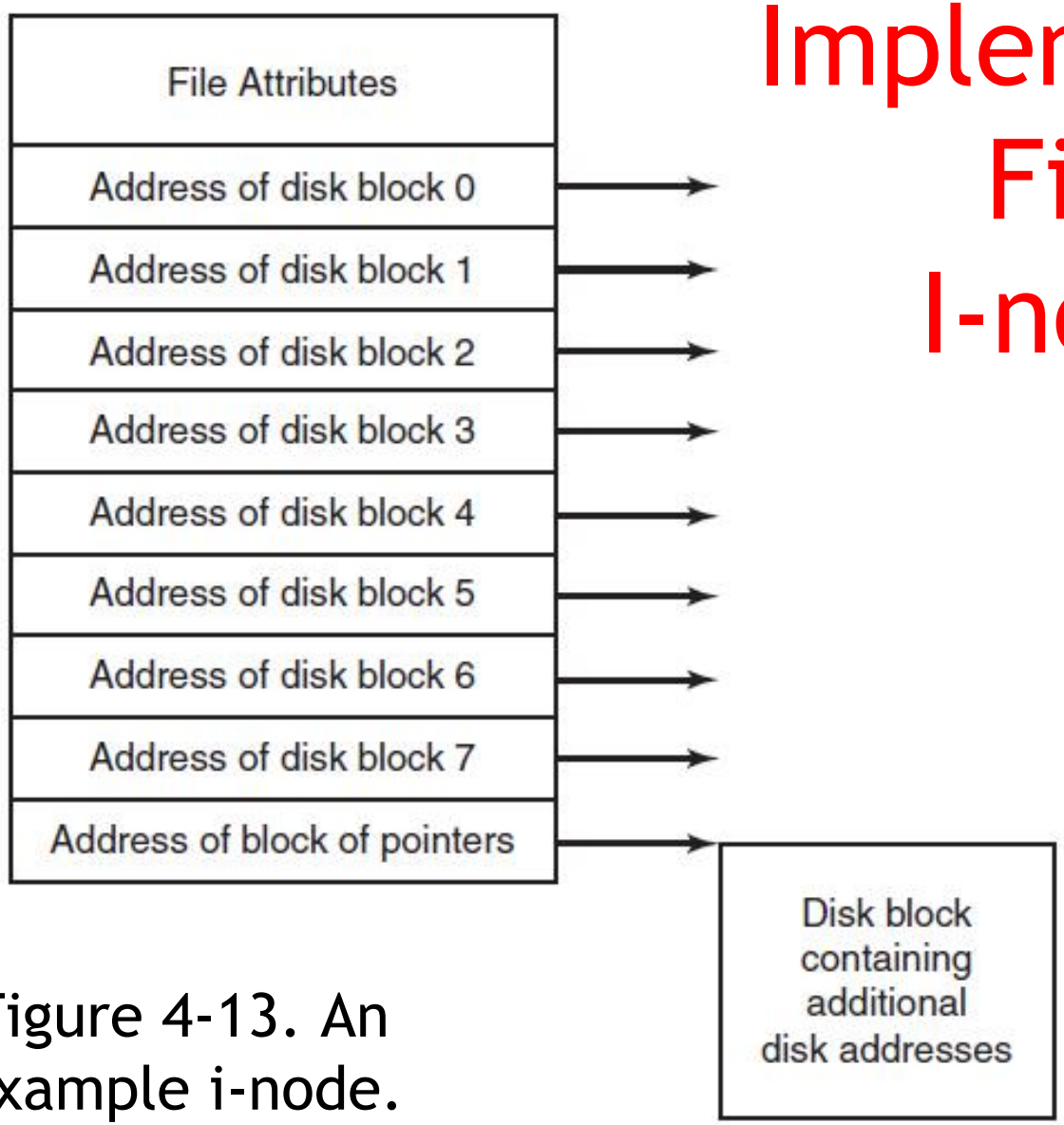


Figure 4-13. An example i-node.

Implementing Directories (1)

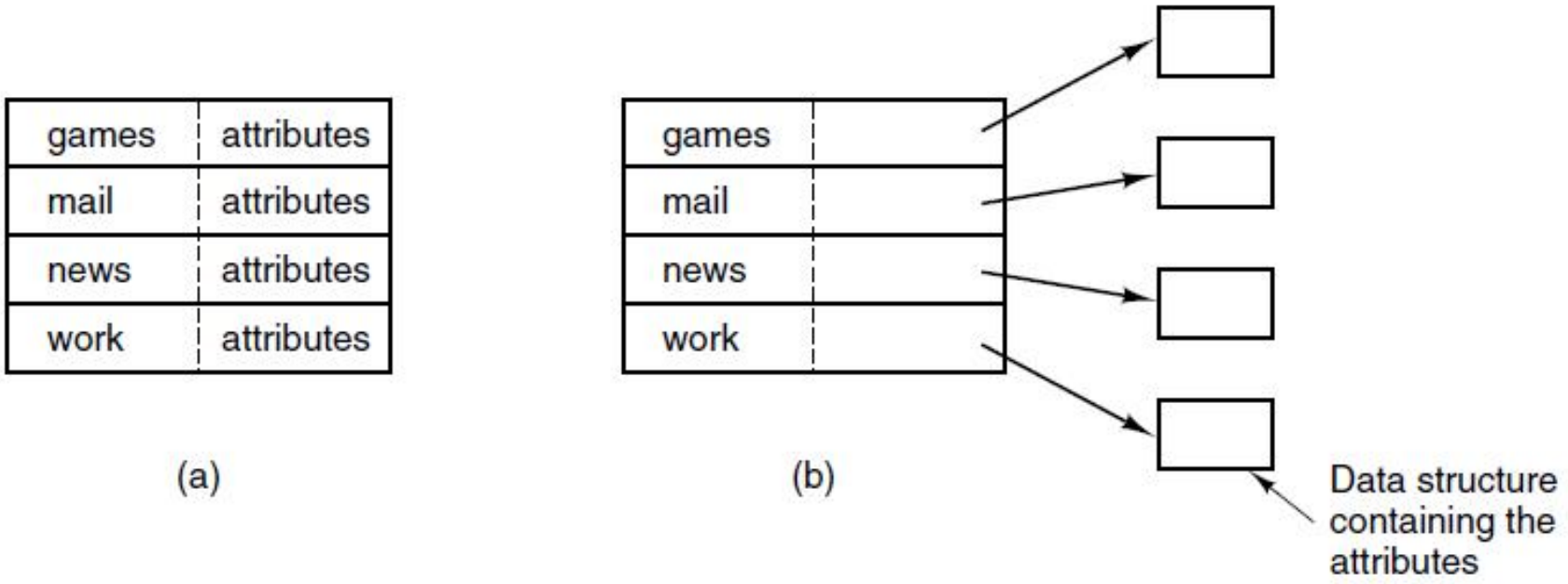


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Implementing Directories (2)

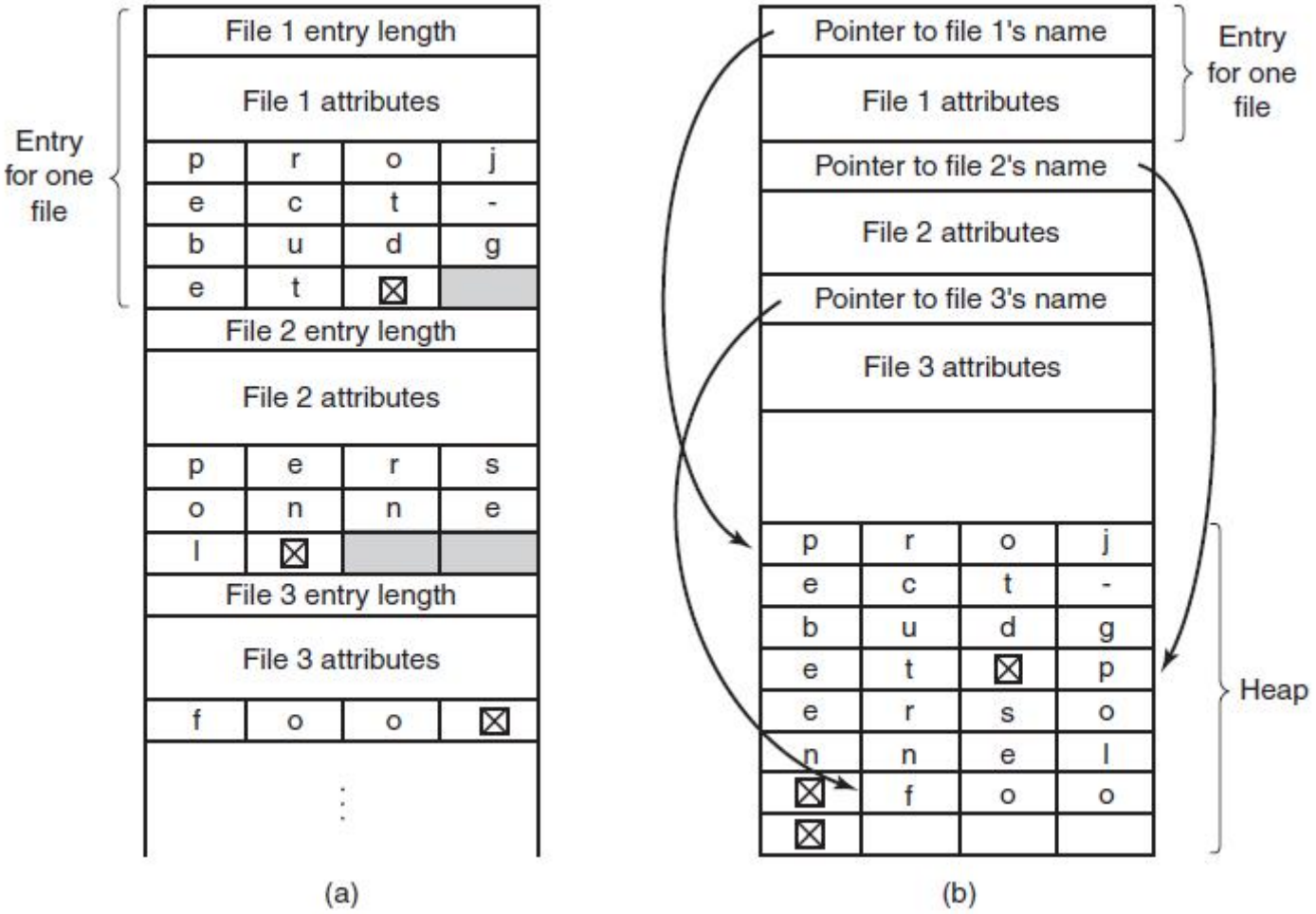


Figure 4-15. Tw o ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files (1)

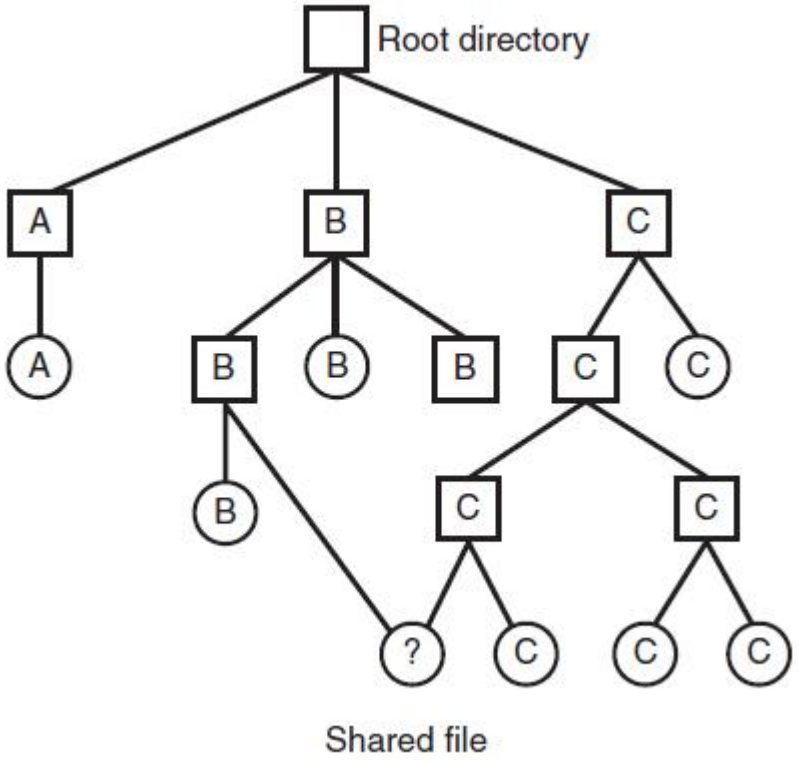


Figure 4-16. File system containing a shared file. (btw, if the file is expanded/appended, does that affect B/C?)

Shared Files (2)

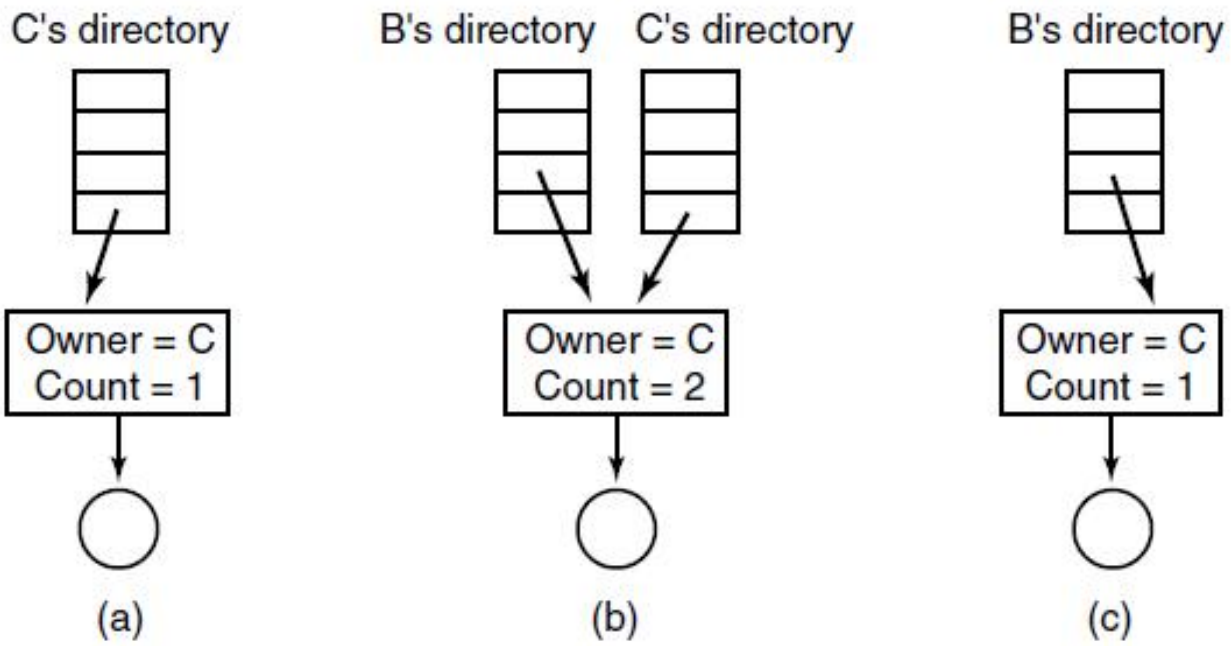


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

(Make sure I remember to demonstrate something neat

Interjection

Have I remembered to show you both a demonstration of ‘block sizes’ and of a file system yet?

Journaling File Systems

Steps to remove a file in UNIX:

1. Remove file from its directory.
2. Release i-node to the pool of free i-nodes.
3. Return all disk blocks to pool of free disk blocks.

Uh… wait…

Virtual File Systems (1)

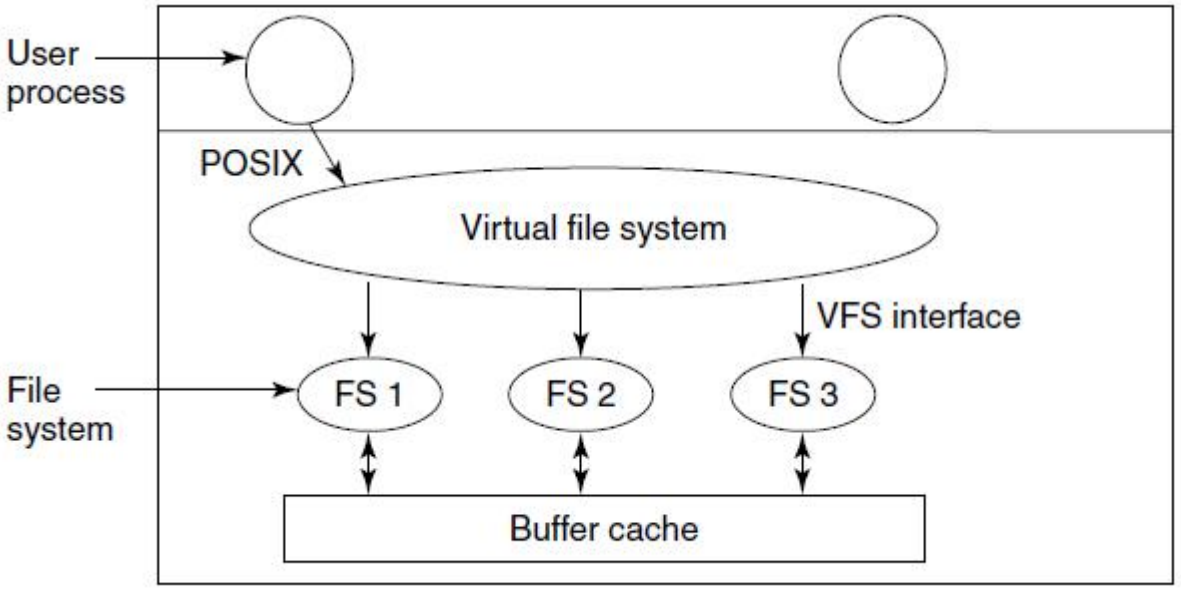


Figure 4-18. Position of the virtual file system.

Virtual File Systems (2)

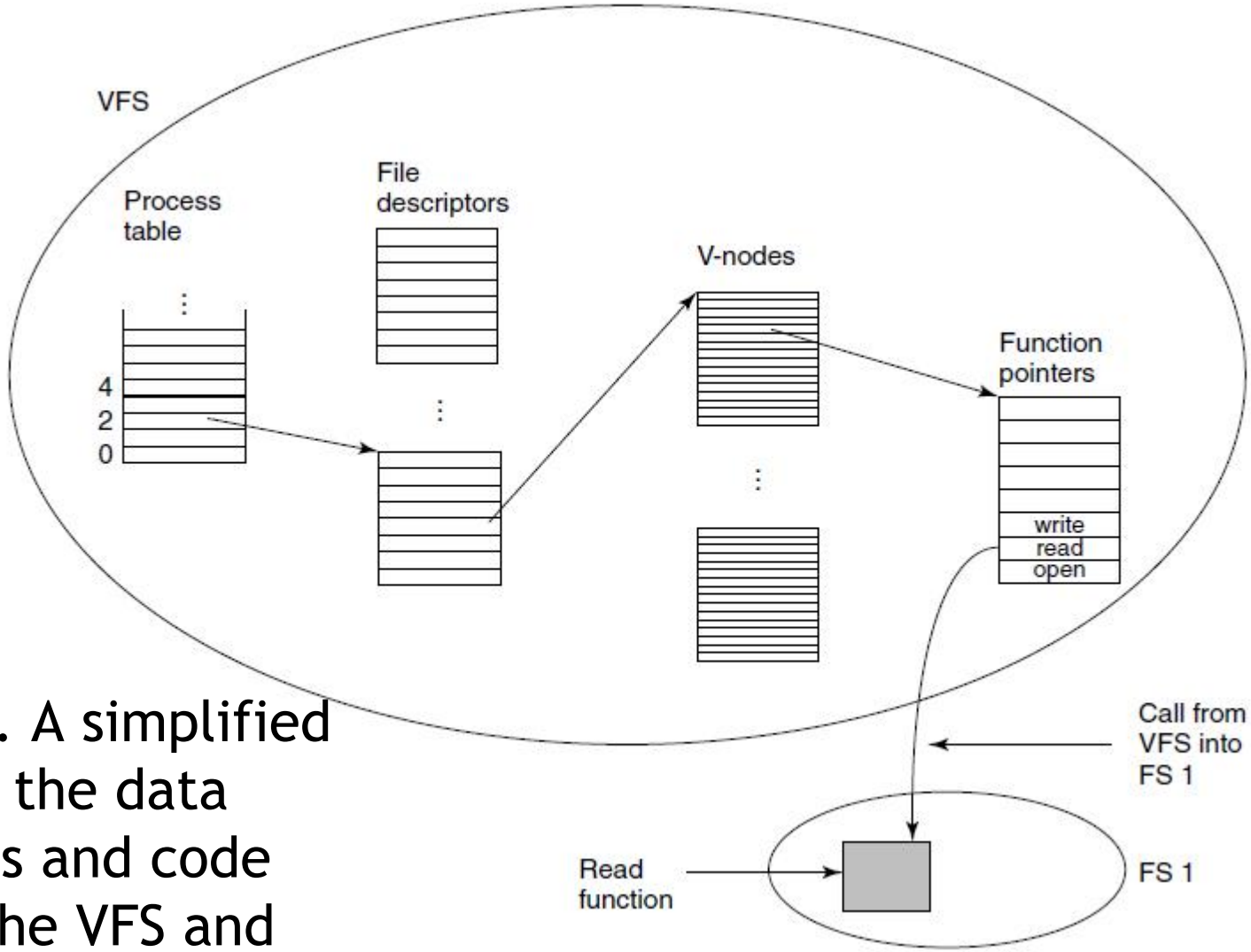


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a *read*.

Disk Space Management (1)

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Figure 4-20. Percentage of files smaller than a given size (in bytes).

Disk Space Management (2)

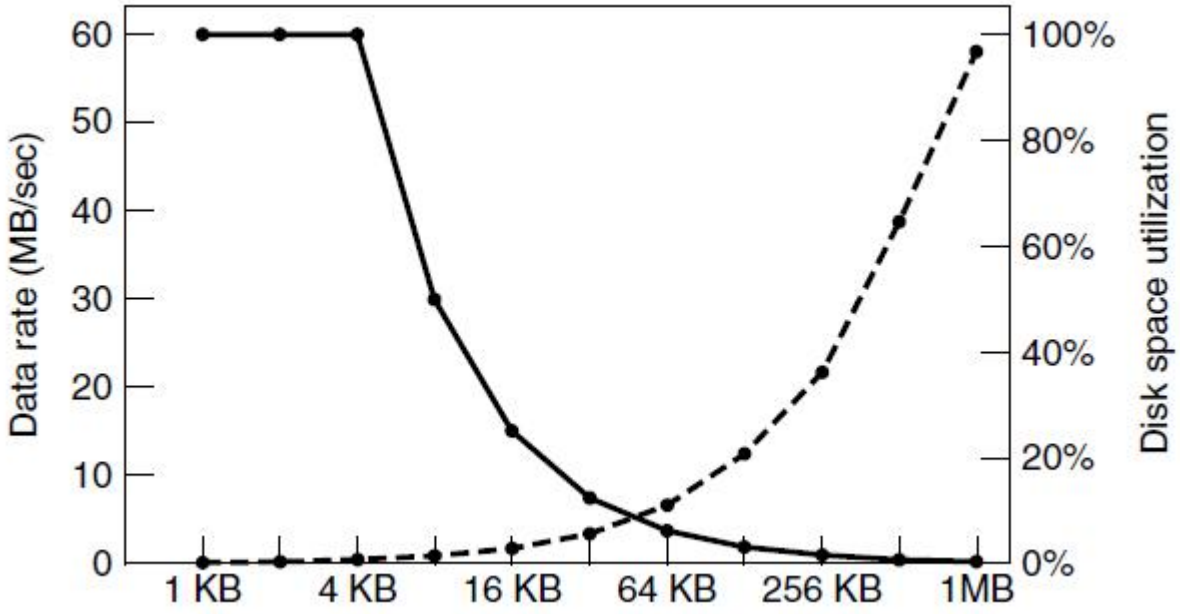


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Keeping Track of Free Blocks (1)

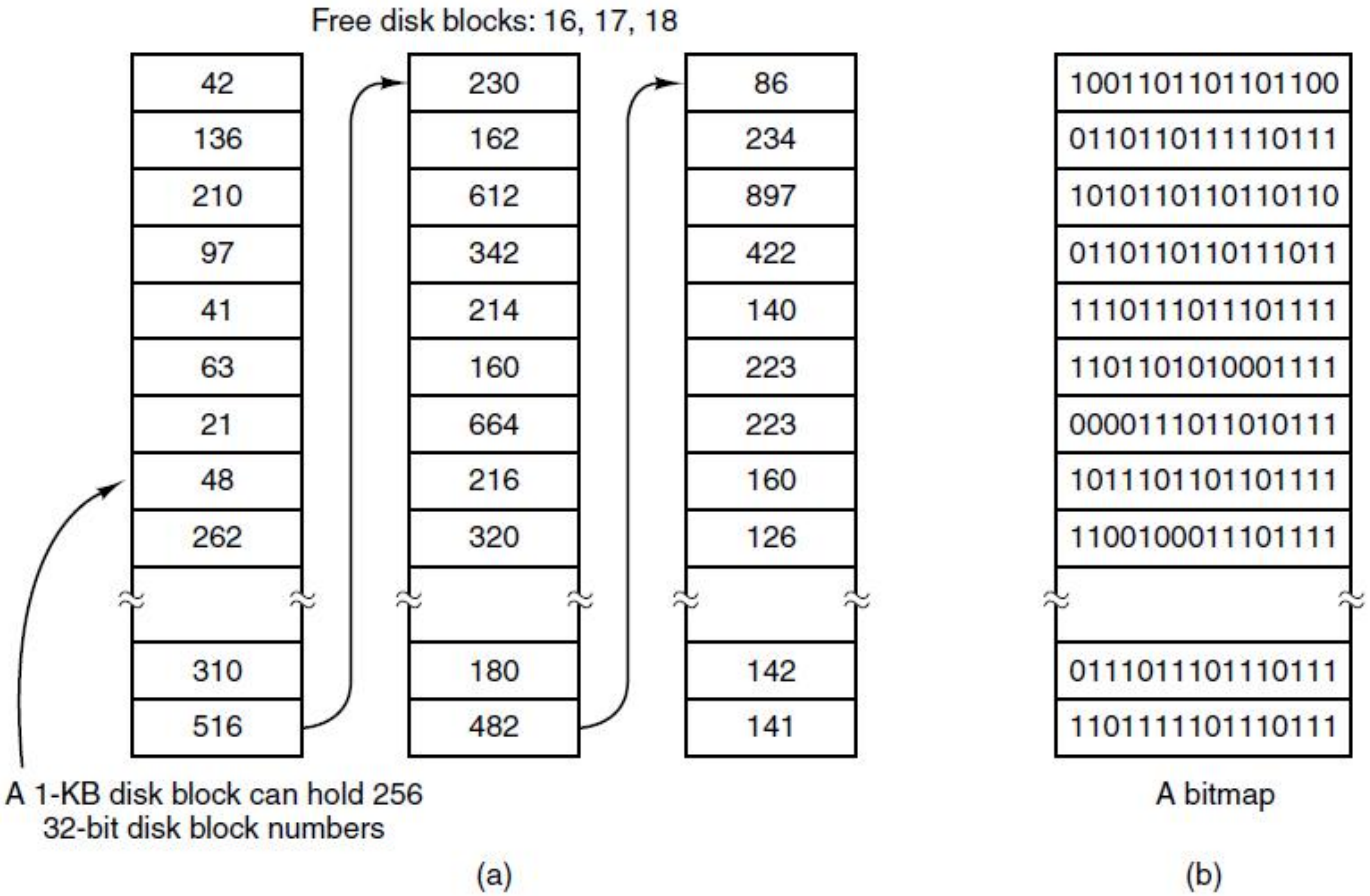


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Keeping Track of Free Blocks (2)

Can we think of a good way to keep track of data on a disk, that (e.g. on magnetic storage) would minimize the number of seek times we could ever have to wait?

And what might it store, precisely?

Disk Quotas

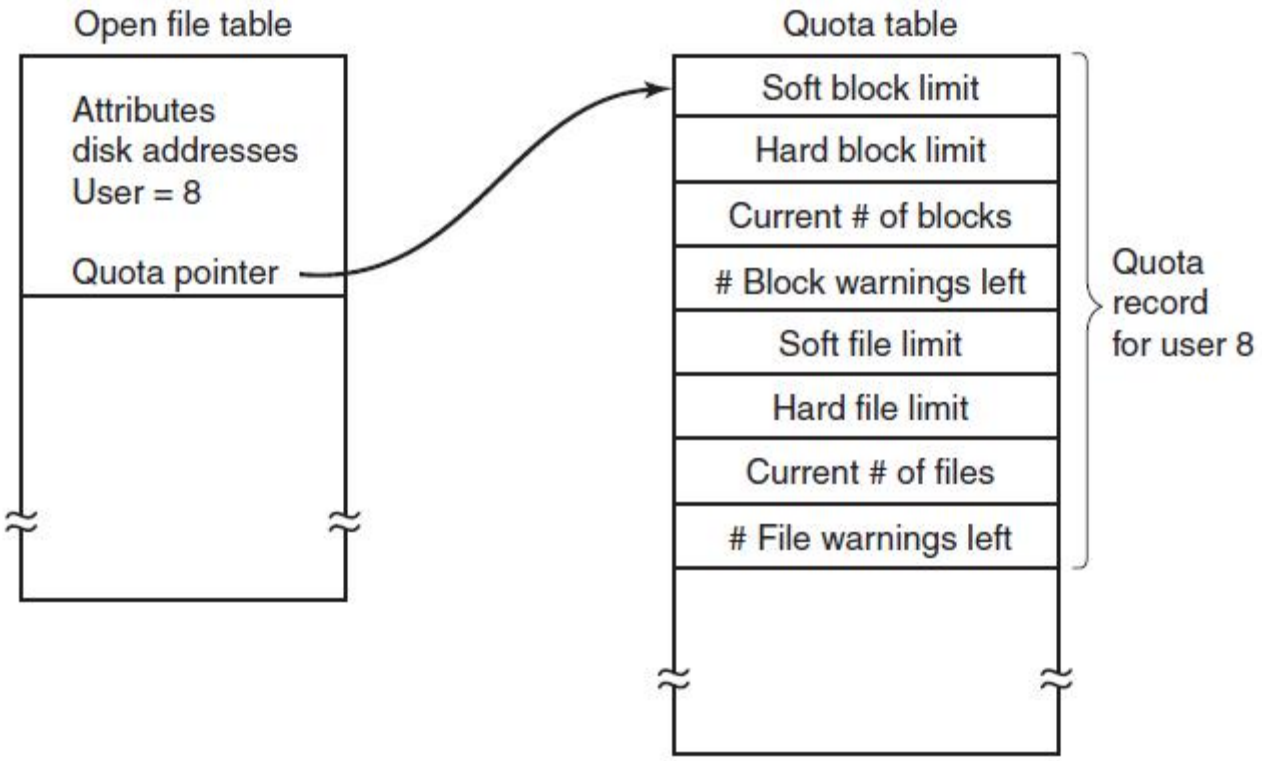


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

File System Backups (1)

Backups to tape are generally made to handle one of two potential problems:

1. Recover from disaster.
2. Recover from stupidity.

File System Backups (1.5)

What are our (possibly competing) goals?

1. Security of data
2. Reasonableness of speed

(Incremental) File System Backups

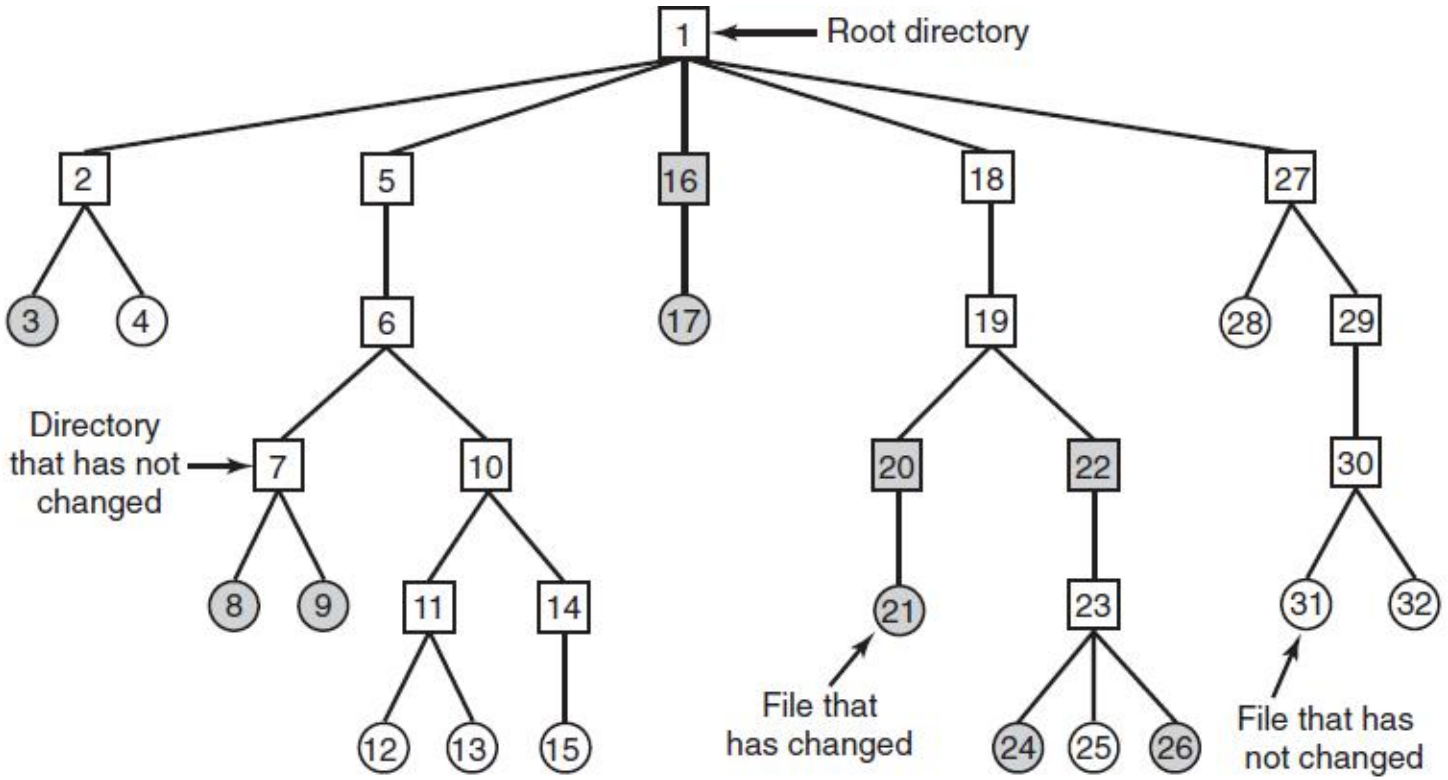


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

File System Backups (3)

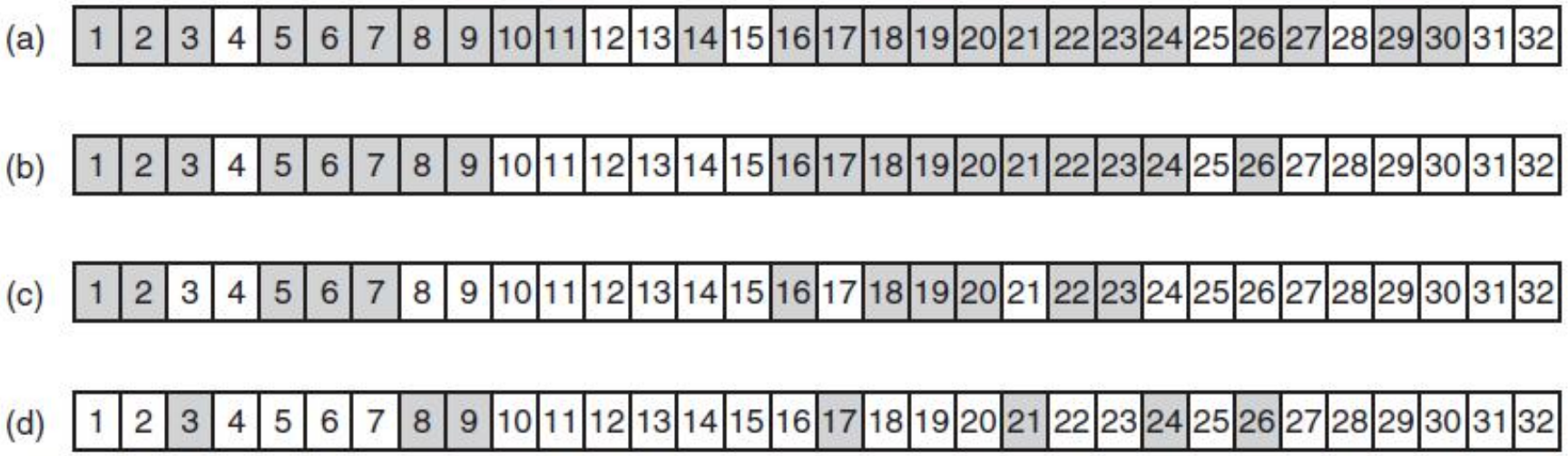


Figure 4-26. Bitmaps used by the logical dumping algorithm.

File System Consistency

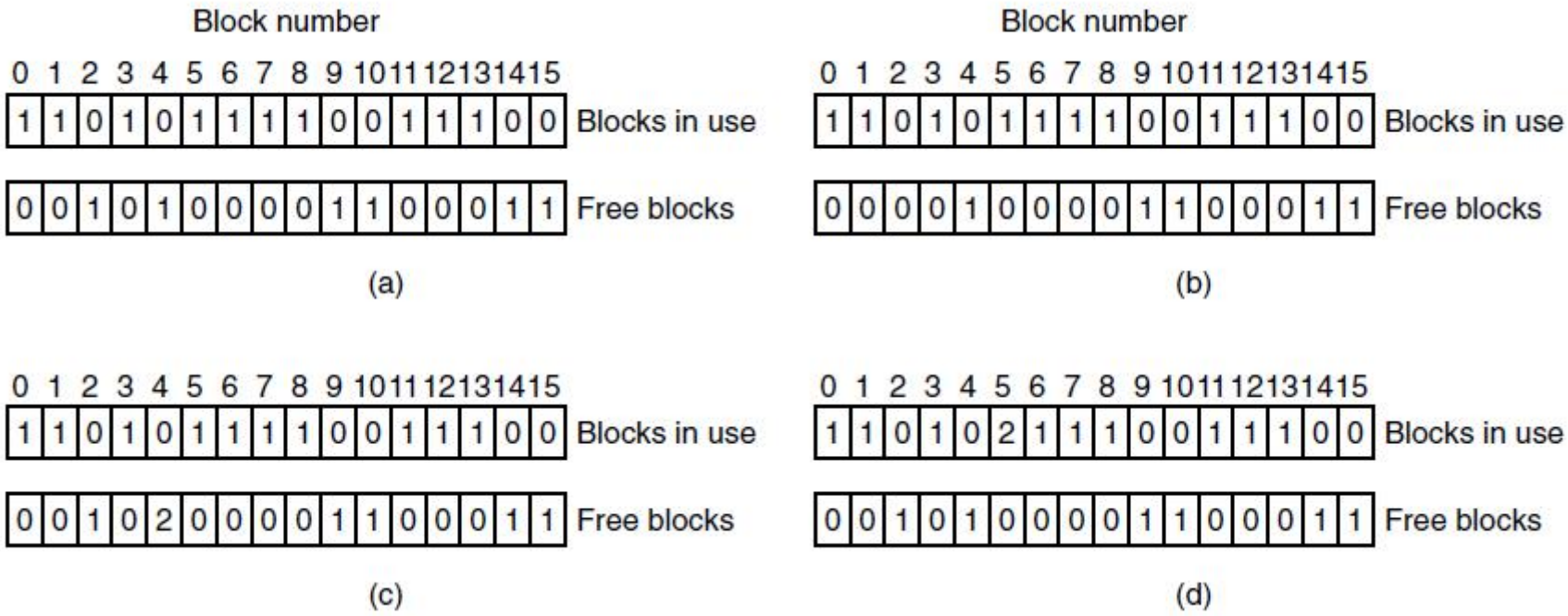


Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

Reducing Disk Arm Motion

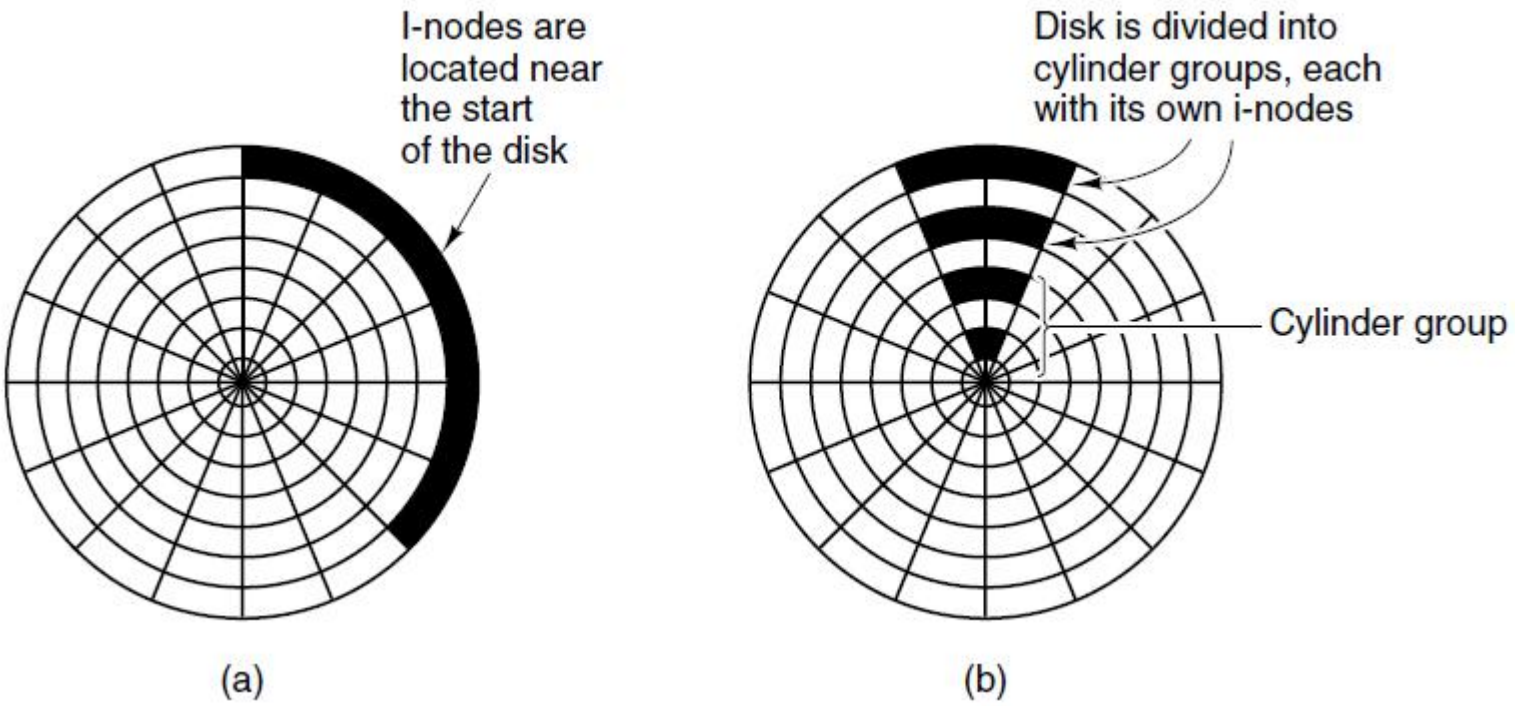


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

Other storage concerns

How familiar are we with:

- Fragmentation/defragmentation
- Write/wear levelling

The MS-DOS File System (1)

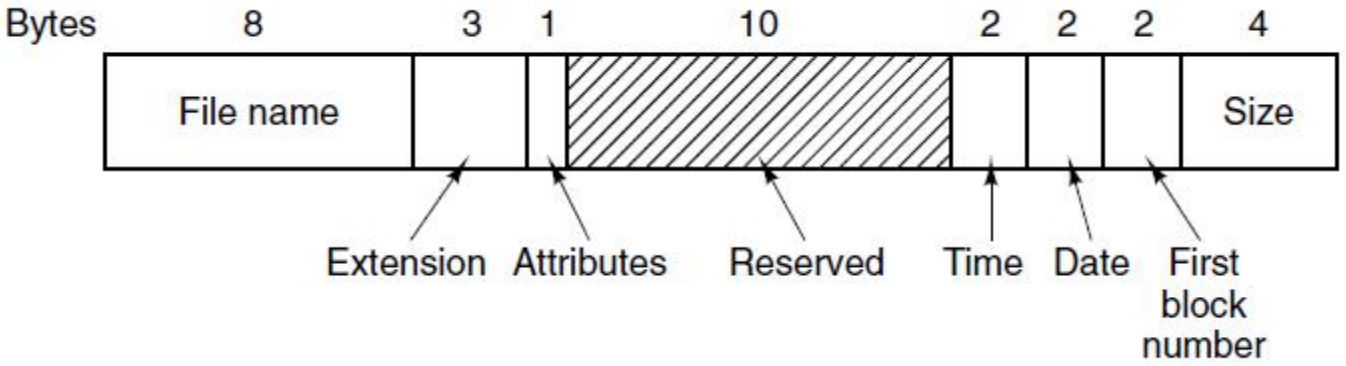


Figure 4-30. The MS-DOS directory entry.

The MS-DOS File System (2)

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

Figure 4-31. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

The UNIX V7 File System (1)

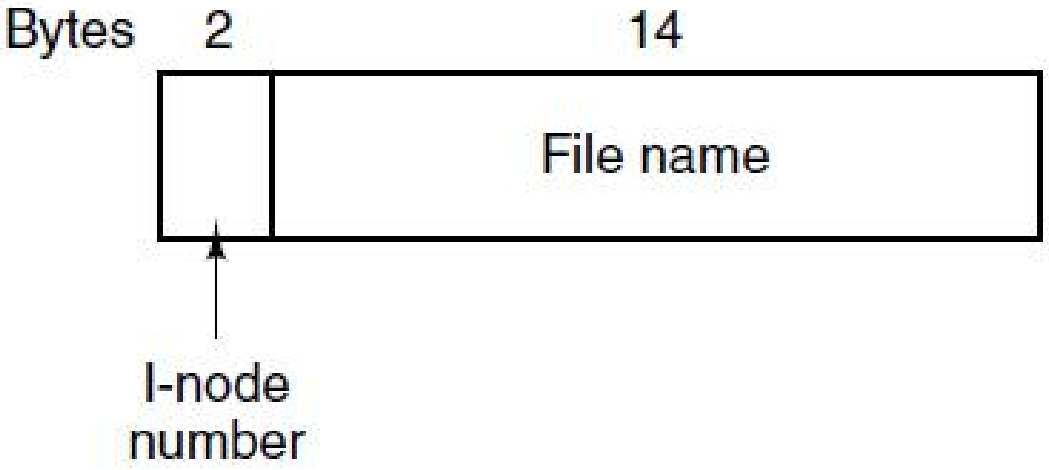


Figure 4-32. A UNIX V7 directory entry.

(bash time?)

((honestly, notes to myself))

- ls –li
  - (and/or ls -lhis)
  - touch
- stat
- df –i (and df -B)
- find / -inum (X) -xdev
- ln [-s] target name (check stat again!)
- ls -ld (and ./.. …wait, what?)
- tune2fs -l

- I can’t do this on sandcastle (because root), but consider the following:

```
$ stat a.out
  File: a.out
  Size: 14472           Blocks: 32         IO Block: 4096   regular file
Device: 810h/2064d      Inode: 11854       Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 1000/     moo)   Gid: ( 1000/     moo)
Access: 2022-05-02 05:39:10.660000000 -0400
Modify: 2022-05-02 05:39:10.660000000 -0400
Change: 2022-05-02 05:39:10.660000000 -0400
 Birth: -

# debugfs -R "stat <11854>" /dev/sdb
Inode: 11854   Type: regular    Mode:  0755   Flags: 0x80000
Generation: 3585818697    Version: 0x00000000:00000001
User:  1000   Group:  1000   Project:     0   Size: 14472
File ACL: 0
Links: 1   Blockcount: 32
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x626fa6be:9d5b3400 -- Mon May  2 05:39:10 2022
 atime: 0x626fa6be:9d5b3400 -- Mon May  2 05:39:10 2022
 mtime: 0x626fa6be:9d5b3400 -- Mon May  2 05:39:10 2022
crtime: 0x626fa6be:9af8da00 -- Mon May  2 05:39:10 2022
Size of extra inode fields: 32
Inode checksum: 0xde45b9ec
EXTENTS:
(0-3):1116528-1116531
```

The UNIX V7 File System (2)

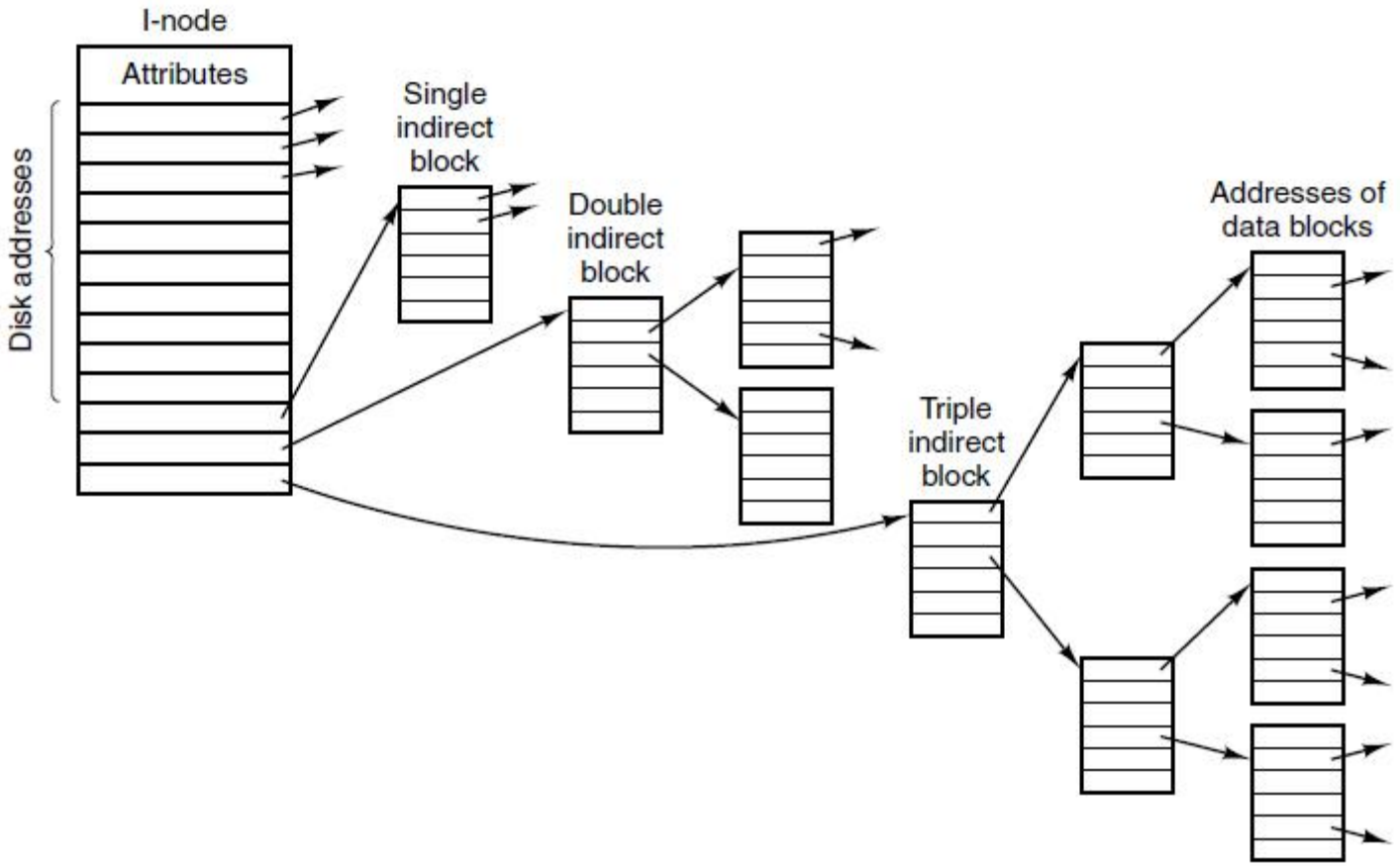


Figure 4-33. A UNIX i-node

The UNIX V7 File System (3)

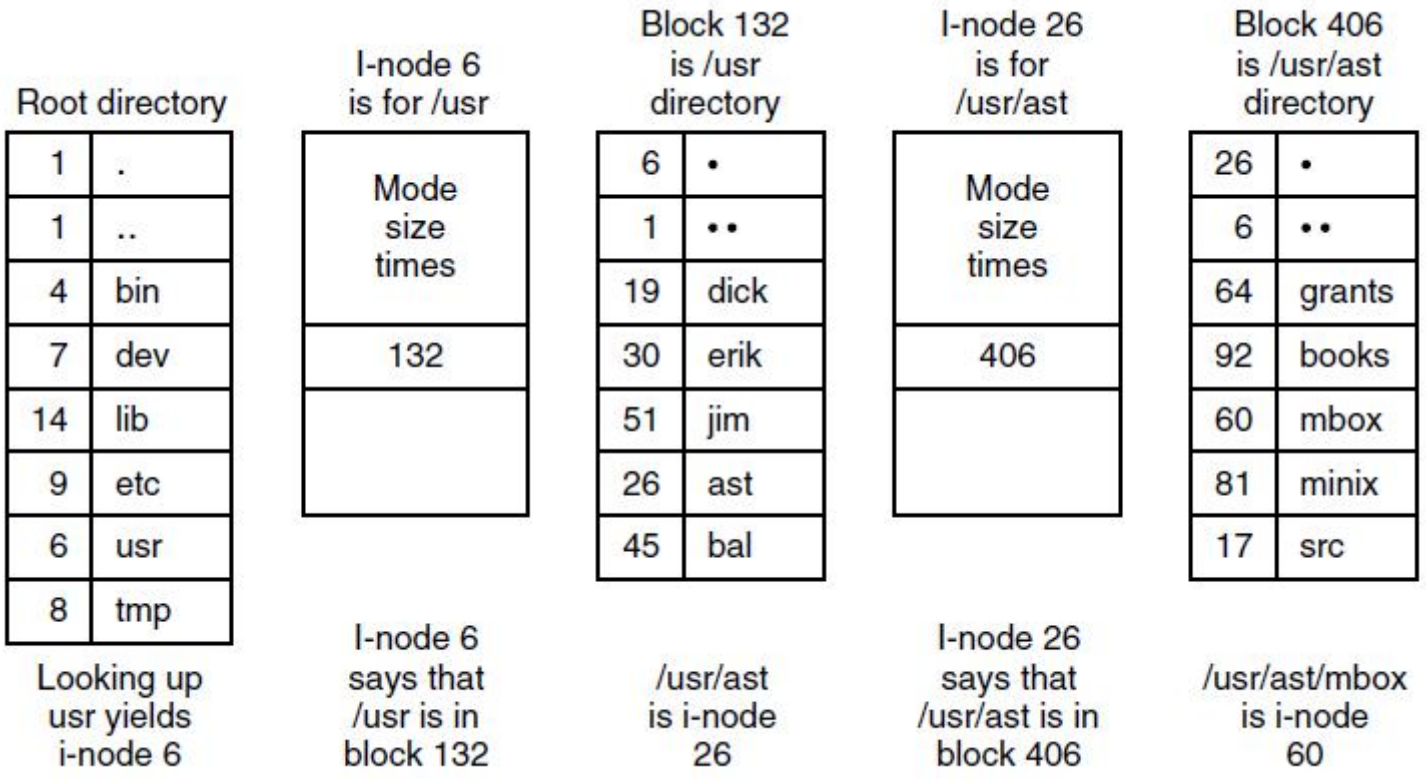


Figure 4-34. The steps in looking up */usr/ast/mbox*.

The ISO 9660 File System

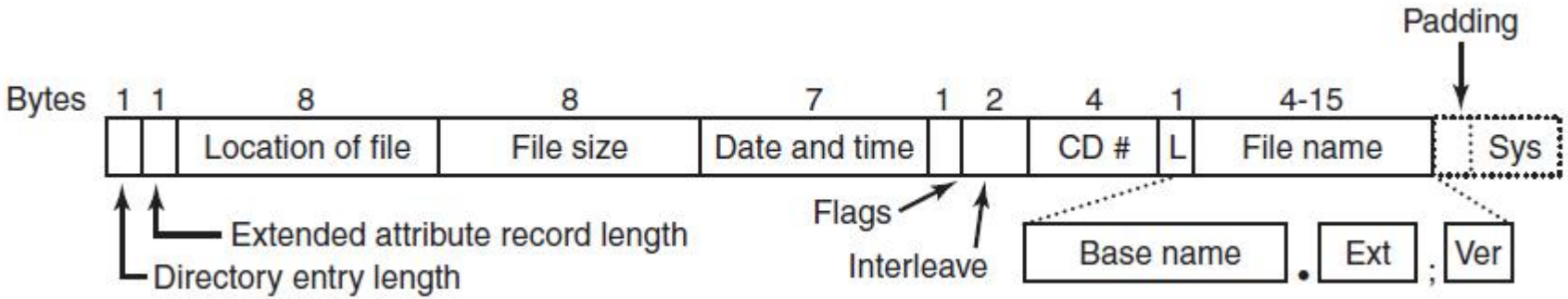


Figure 4-35. The ISO 9660 directory entry.

Joliet Extensions

1. Long file names.
2. Unicode character set.
3. Directory nesting deeper than eight levels.
4. Directory names with extensions

End

Chapter 4