File Systems

Week 10

Team

- Instructor
  - Giancarlo Succi
- Teaching Assistants
  - Nikita Lozhnikov (also Tutorial Instructor)
  - Manuel Rodriguez
  - Shokhista Ergasheva

Sources

- These slides have been adapted from the original slides of the adopted book:
  - Tanenbaum & Bos, Modern Operating Systems: 4th edition, 2013
    Prentice-Hall, Inc.

  and customised for the needs of this course.
- Additional input for the slides are detailed later

File Systems (1)

- Essential requirements for long-term information storage:
  - It must be possible to store a very large amount of information
  - Information must survive termination of a process using it
  - Multiple processes must be able to access information concurrently

File Systems (2)

- Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:
  - Read block *k*
  - Write block *k*

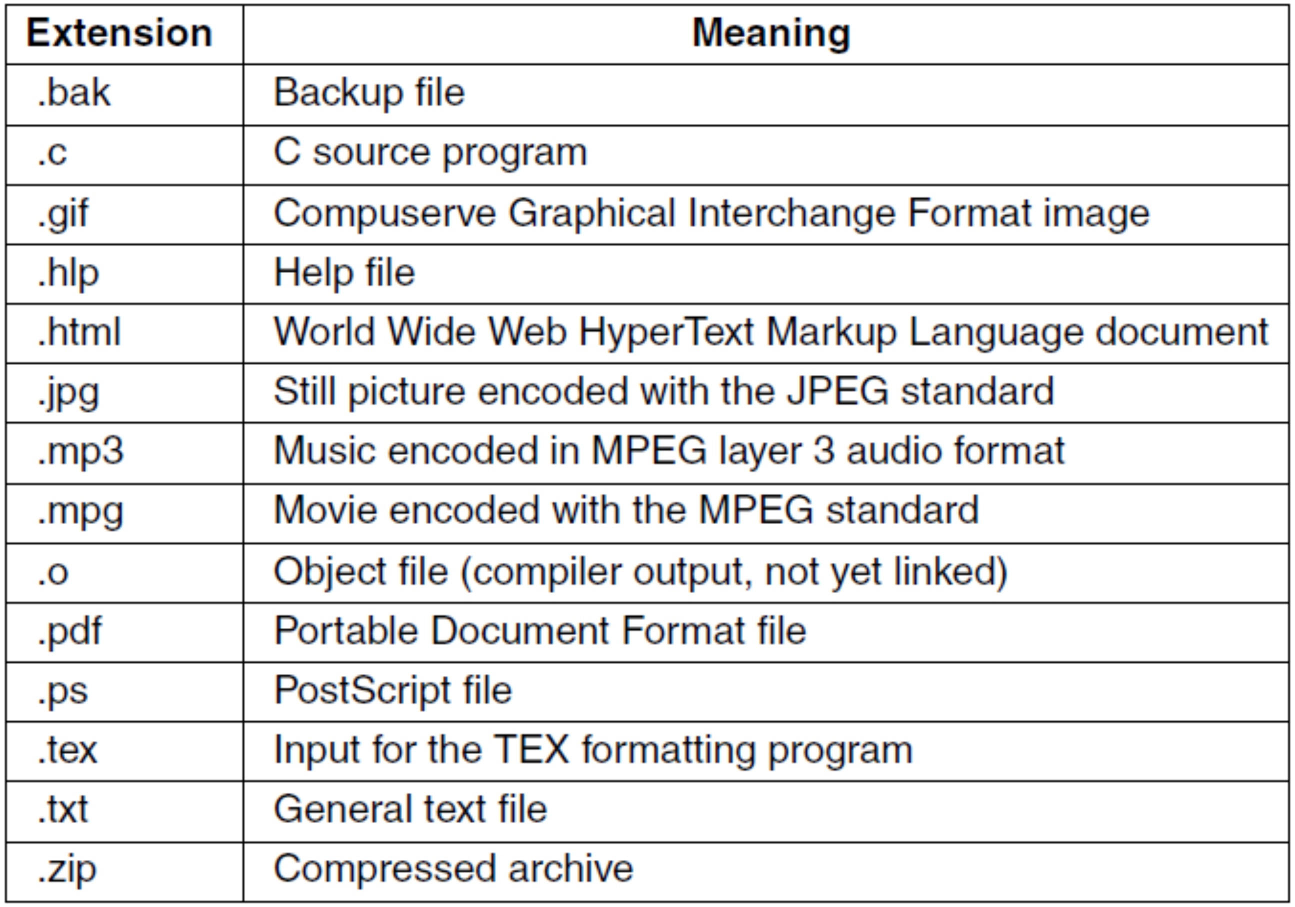
File Systems (3)

- Questions that quickly arise:
  - How do you find information?
  - How do you keep one user from reading another user’s data?
  - How do you know which blocks are free?

Files

- 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
- Typically, files have a name and an extension

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.

File Structure (1)

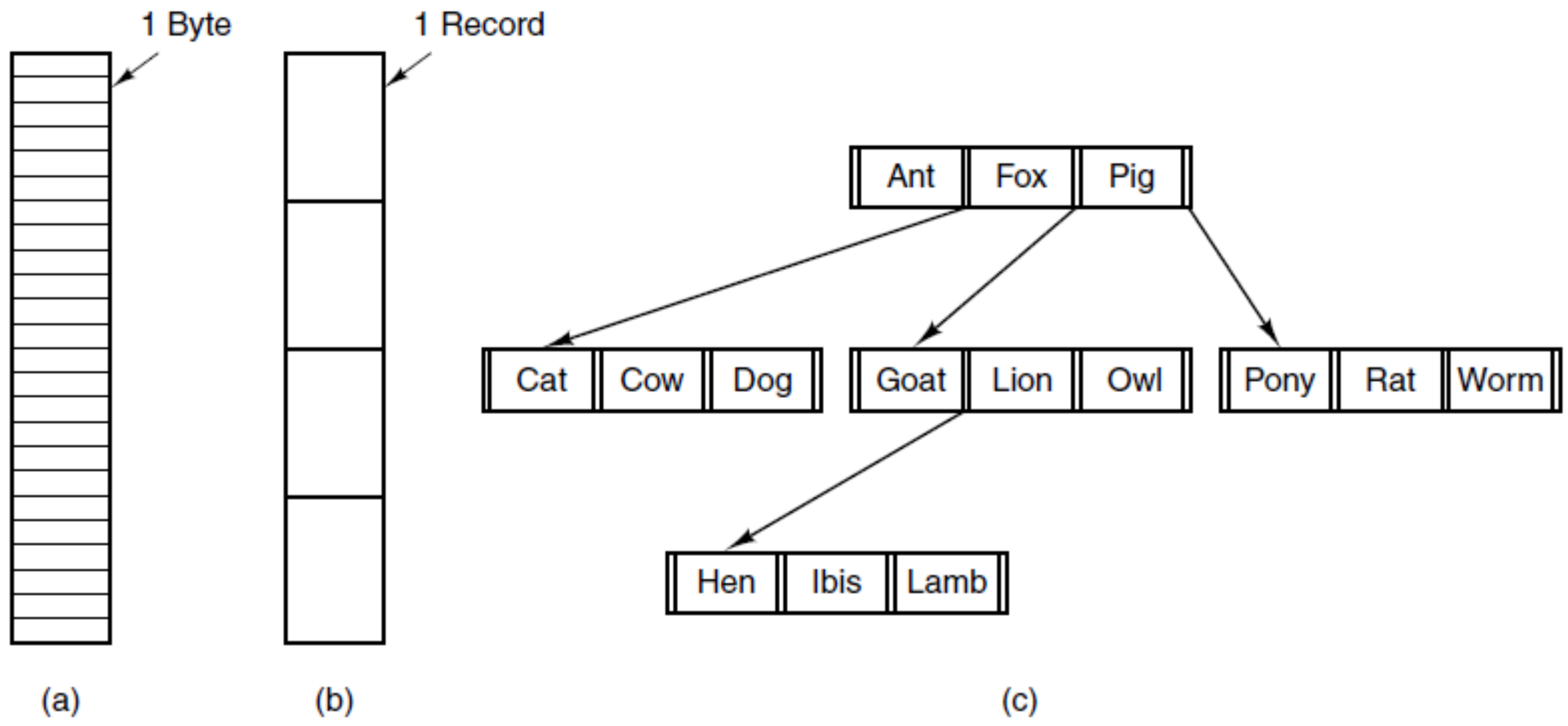


Figure 4.2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Structure (2)

- 4.2a: an unstructured sequence of bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows use this approach
- 4.2b: a sequence of fixed-length records, each with some internal structure
- 4.2c: consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record
  - The tree is sorted on the key field, to allow rapid searching for a particular key

File Types (1)

- Regular files are the ones that contain user information
- Directories are system files for maintaining the structure of the file system
- Character special files are related to I/O and used to model serial I/O devices, such as terminals, printers and networks
- Block special files are used to model disks

File Types (2)

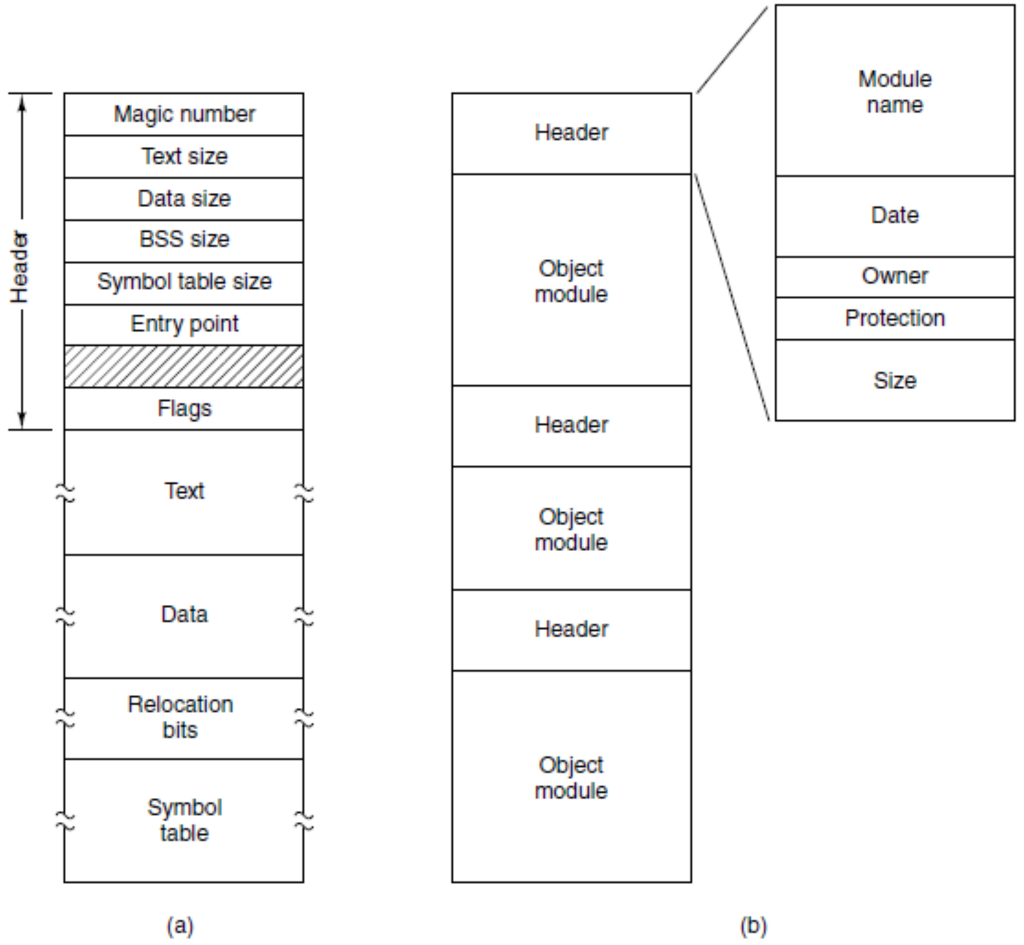


Figure 4-3. (a) An executable file. (b) An archive

File Types (3)

- **Regular files** are generally either ASCII files or binary files. ASCII Files consist of lines of text
- In some systems each line is terminated by a carriage return character; in others, the line feed character is used
- Some systems (e.g., Windows) use both.

File Types (4)

- **Binary Files** have internal structure known to programs that use them
- Example: a simple executable binary file taken from an early version of UNIX has five sections: header, text, data, relocation bits, and symbol table (Fig 4.3a)
- Second example of a binary file is an archive, also from UNIX (Fig 4.3b)

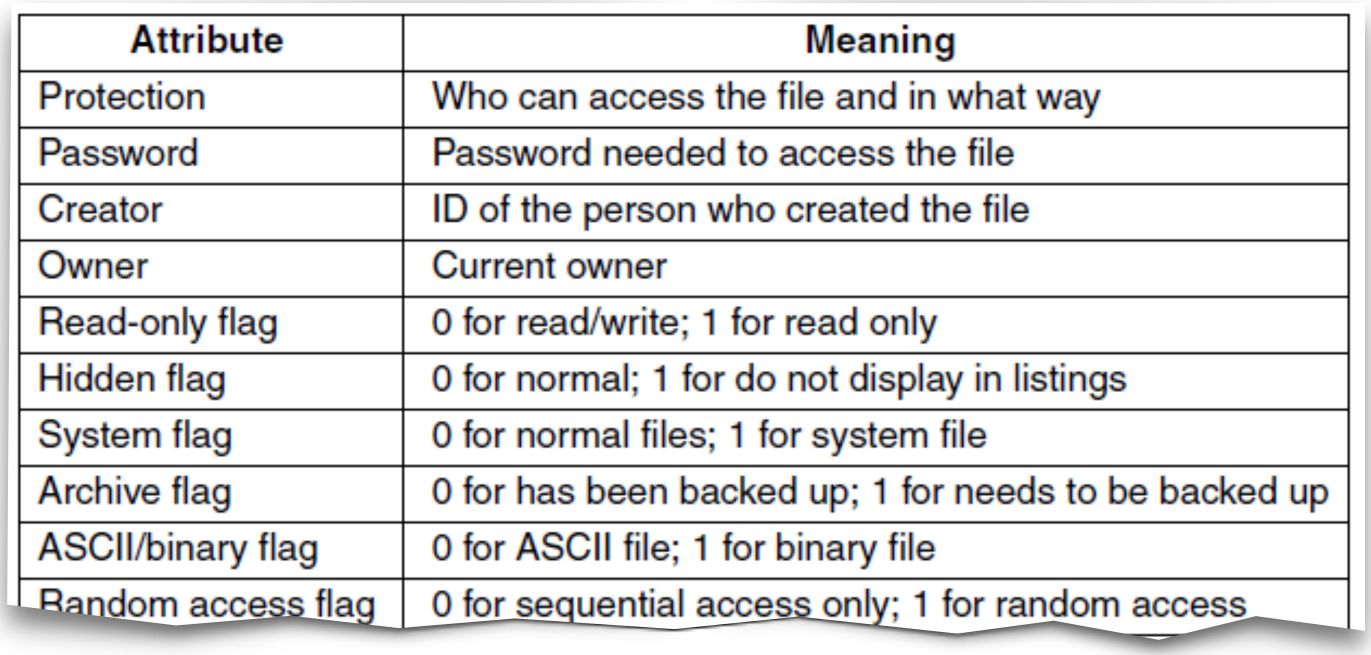
File Access (1)

- **Sequential Access**
  - Early operating systems provided only sequential access
  - In these systems, a process could read all the bytes or records in a file in order, starting at the beginning, but could not skip around and read them out of order

File Access (2)

- **Random Access**
  - When disks came into use for storing files, it became possible to read the bytes or records of a file out of order, or to access records by key rather than by position

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

Figure 4-4. Some possible file attributes.

File Operations (1)

- **Create:** empty file is created, some attributes are set
- **Delete:** file is deleted
- **Open:** fetches the attributes and list of disk addresses into main memory for rapid access later
- **Close:** file is closed to free up internal table space. Last block is forcefully written on the disk
- **Read:** data are read from file from current position
- **Write:** data are written at the current position

File Operations (2)

- **Append:** adds the data only to the end of the file
- **Seek:** repositions a file pointer to a specific position inside the file
- **Get attributes:** returns a set of file attributes
- **Set attributes:** changes some file attributes
- **Rename:** renames a file without making a copy of it

Directories (1)

- To keep track of files, file systems normally have directories or folders, which are themselves files

Directories (2)

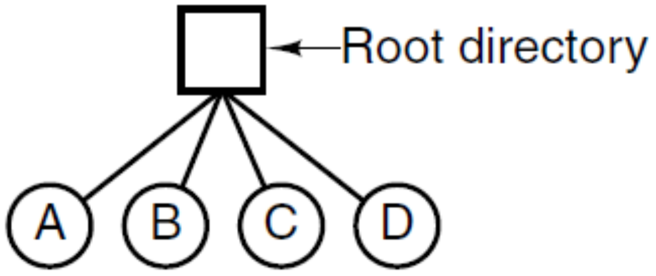


Figure 4-6. A single-level directory system containing four files

Directories (3)

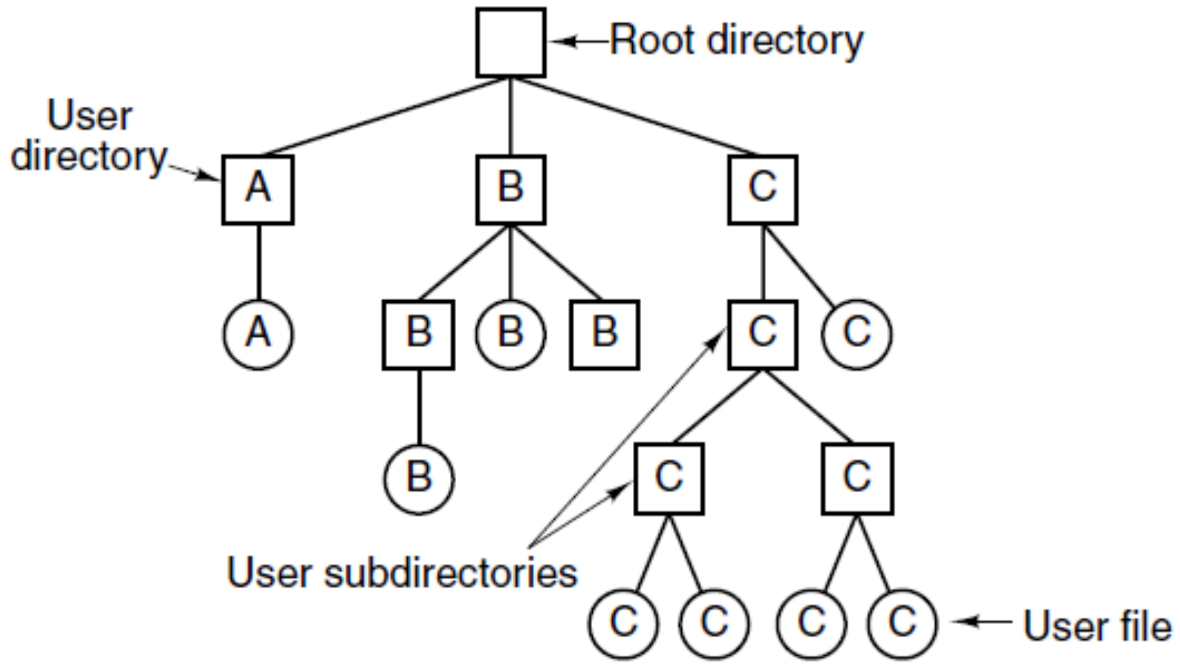


Figure 4-7. A hierarchical directory system

Path Names

- Two different methods are commonly used:
  - Absolute path name
  - Relative path name

Absolute Path Name

- Absolute path name consists of the path from the root directory to the file
- Example:
  The path */usr/ast/mailbox* means that the root directory contains a subdirectory *usr*, which in turn contains a subdirectory *ast*, which contains the file *mailbox*

Relative Path Name

- Relative path name is used in conjunction with the concept of the working directory (also called the current directory)
- Example:
  If the current working directory is */usr/ast*, then the file which absolute path is */usr/ast/mailbox* can be referenced simply as *mailbox*

Path Names

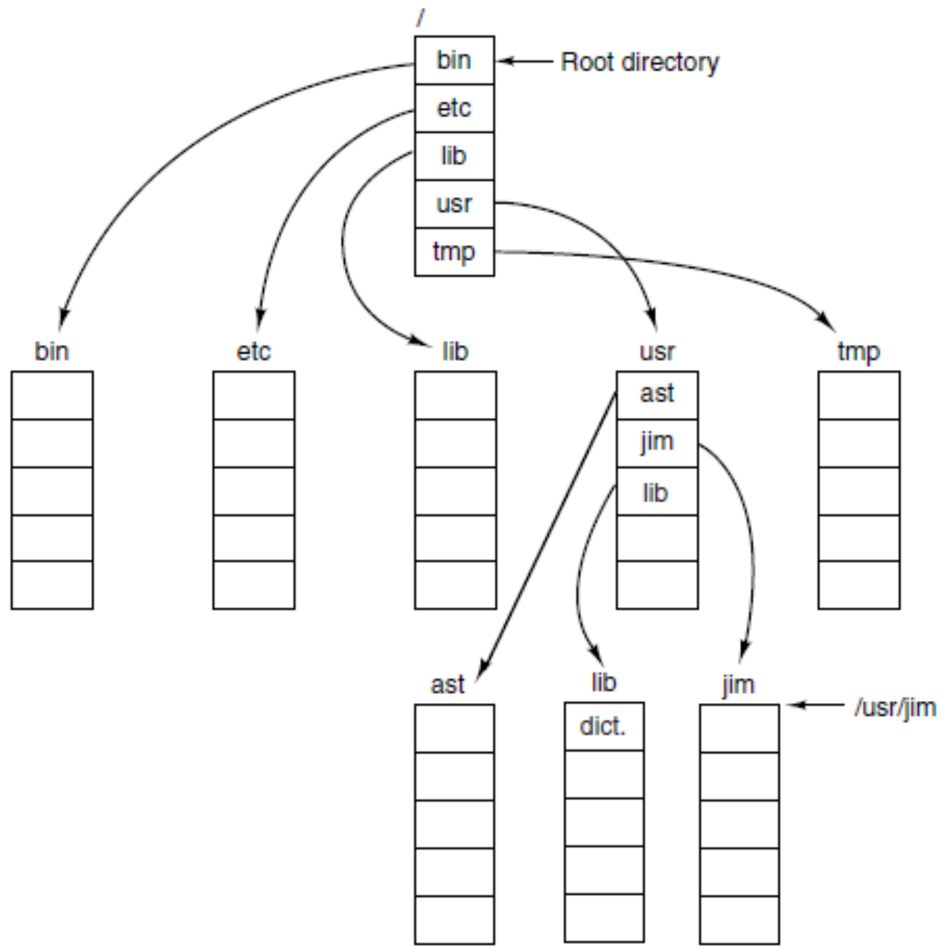


Figure 4-8. A UNIX directory tree

Directory Operations (1)

- **Create:** an empty directory is created
- **Delete:** a directory is deleted. Only empty directories can be deleted
- **Opendir:** to read from a directory, it must be opened like a file
- **Closedir:** closes directory, again, like a file

Directory Operations (2)

- **Readdir:** returns the next entry in an open directory in a standard format
- **Rename:** directory is renamed the same way as files are renamed
- **Link:** when a link to a file is created, it appears more than in one directory. Counter in file's i-node is incremented
- **Unlink:** a directory entry is removed

File System Layout (1)

- File systems are stored on disks
- Most disks can be divided up into one or more partitions, with independent file systems on each partition

File System Layout (2)

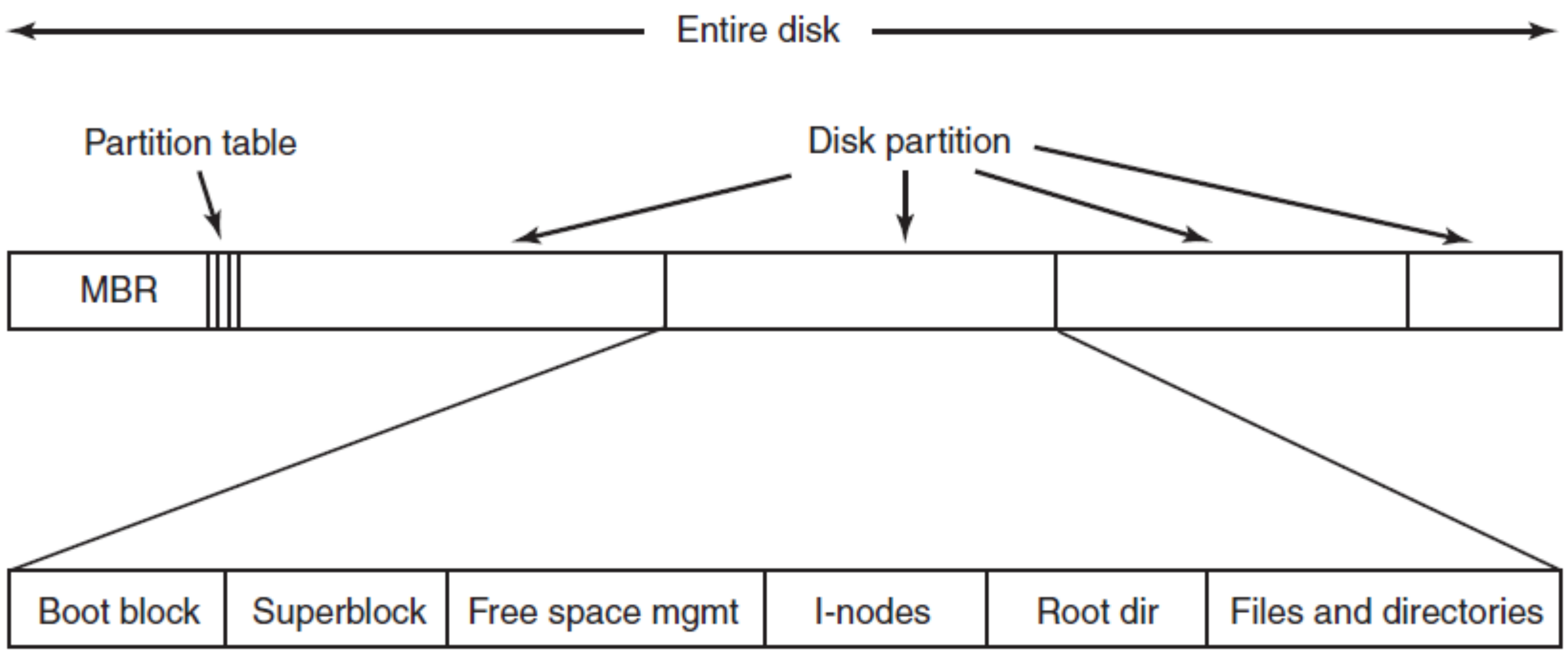


Figure 4-9. A possible file system layout.

File System Layout (3)

- Sector 0 of the disk is called the MBR (**Master Boot Record**) and is used to boot the computer
- The end of the MBR contains the partition table which table gives the starting and ending addresses of each partition

File System Layout (4)

- Boot sequence:
  - BIOS reads in and executes the MBR
  - MBR locates the active partition, reads in its first block, which is called the boot block, and executes it
  - The program in the boot block loads the operating system contained in that partition

File System Layout (5)

- The layout of a disk partition also contains **superblock** which includes:
  - magic number to identify the file-system type
  - the number of blocks in the file system
  - other key administrative information

Implementing Files (1)

- Various methods are used in implementing file storage to keep track of which disk blocks go with which file:
  - Contiguous Allocation
  - Linked-List Allocation
  - i-nodes

Implementing Files (2)

- Contiguous allocation: stores each file as a contiguous run of disk blocks
- Advantages:
  - Simplicity. We need two numbers only: the disk address of the first block and the number of blocks in the file
  - Read performance: the entire file can be read from the disk in a single operation
- Disadvantages:
  - Disk fragmentation

Implementing Files (3)

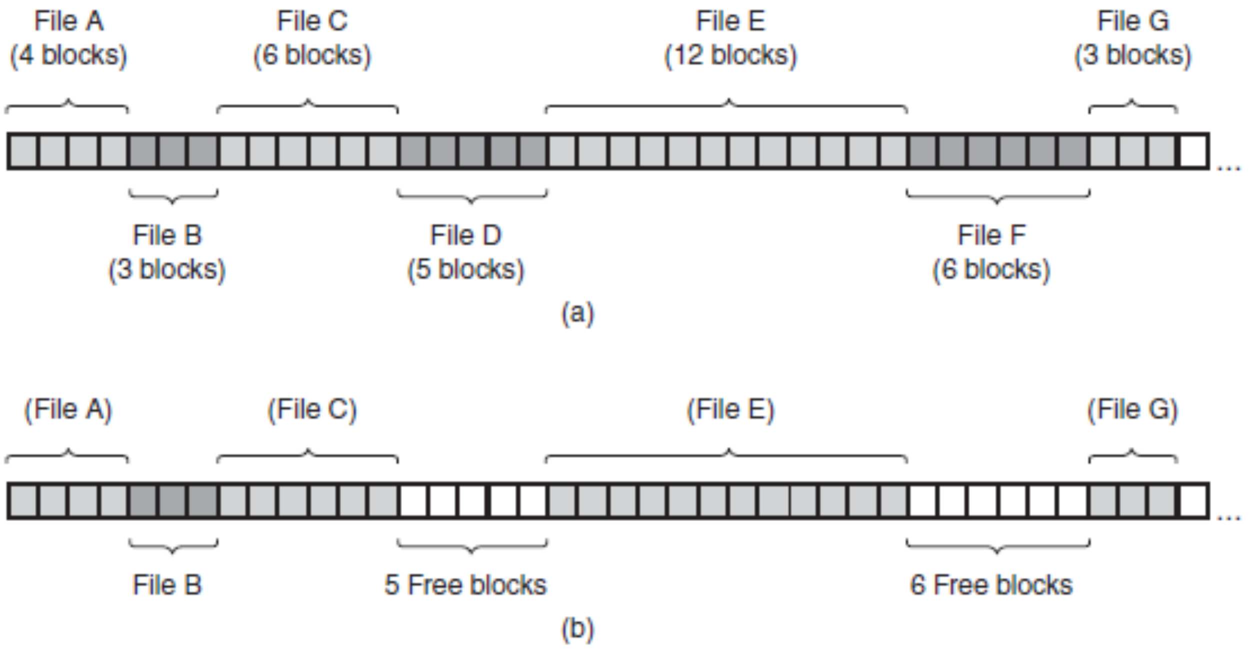


Figure 4-10. (a) Contiguous allocation of disk space for seven files (b) The state of the disk after files D and F have been removed

Implementing Files (4)

- Linked List Allocation: the first word of each block is used as a pointer to the next one. The rest of the block is for data
- Advantages:
  - No fragmentation
- Disadvantages:
  - Random access is extremely slow
  - Only a part of a block is used for storage

Implementing Files (5)

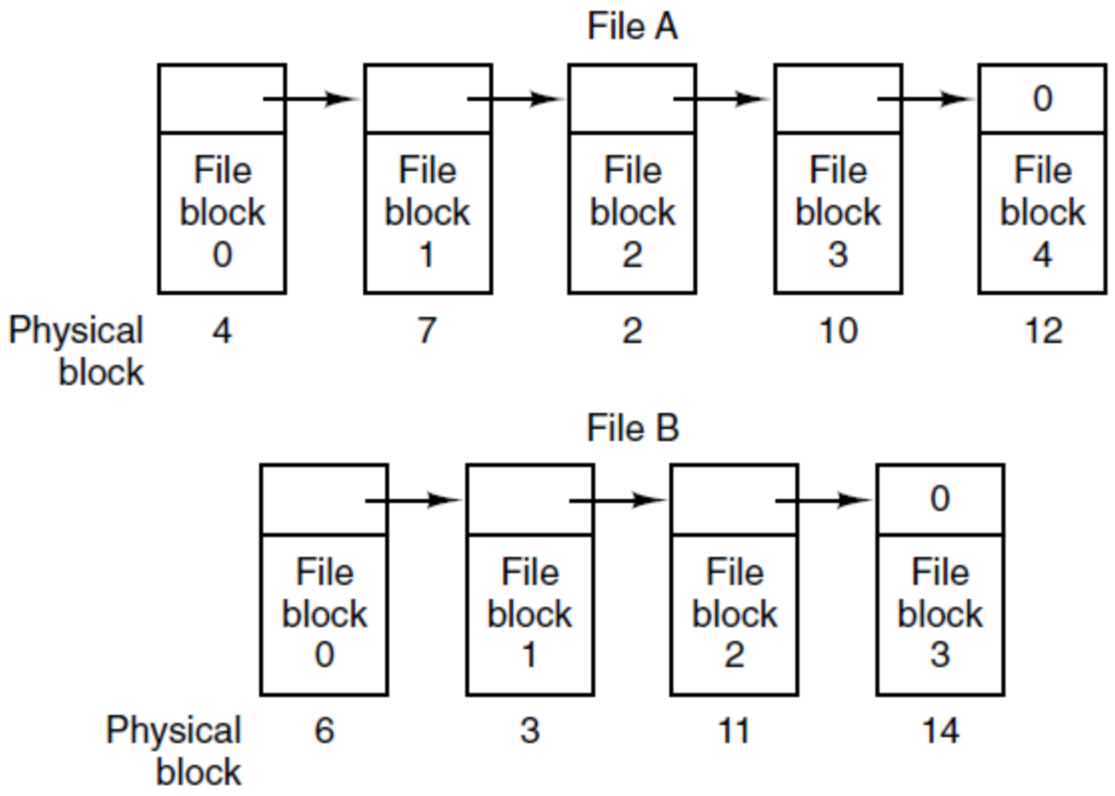


Figure 4-11. Storing a file as a linked list of disk blocks.

Implementing Files (6)

- Linked-list allocation using a table in memory takes the pointer word from each disk block and putting it in a table in memory
- Advantages:
  - Since the table is in memory, lookup is fast
  - Only data are stored in blocks
- Disadvantages:
  - The whole table must be in memory (for 1-TB disk and 1-KB block size, table size is approx. 3 GB)

Implementing Files (7)

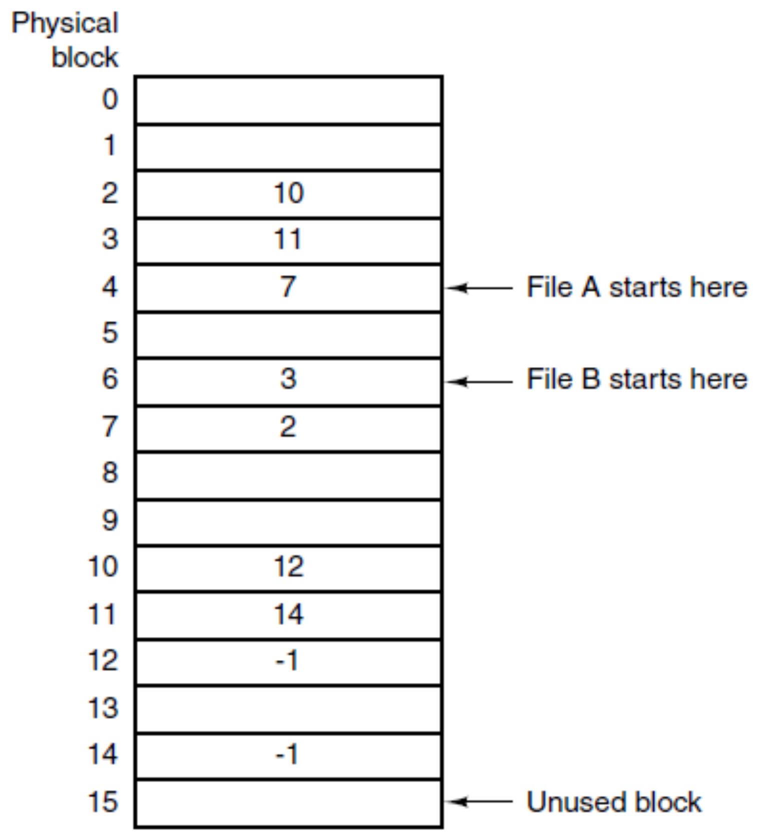


Figure 4-12. Linked list allocation using a file allocation table in main memory.

Implementing Files (8)

- A data structure called an i-node (index-node) is associated with each file, which lists the attributes and disk addresses of the file’s blocks
- Advantages:
  - Only i-node for a file which is open must be in memory
  - Requires an array in memory whose size is proportional to the maximum number of files that may be open at once

Implementing Files (9)

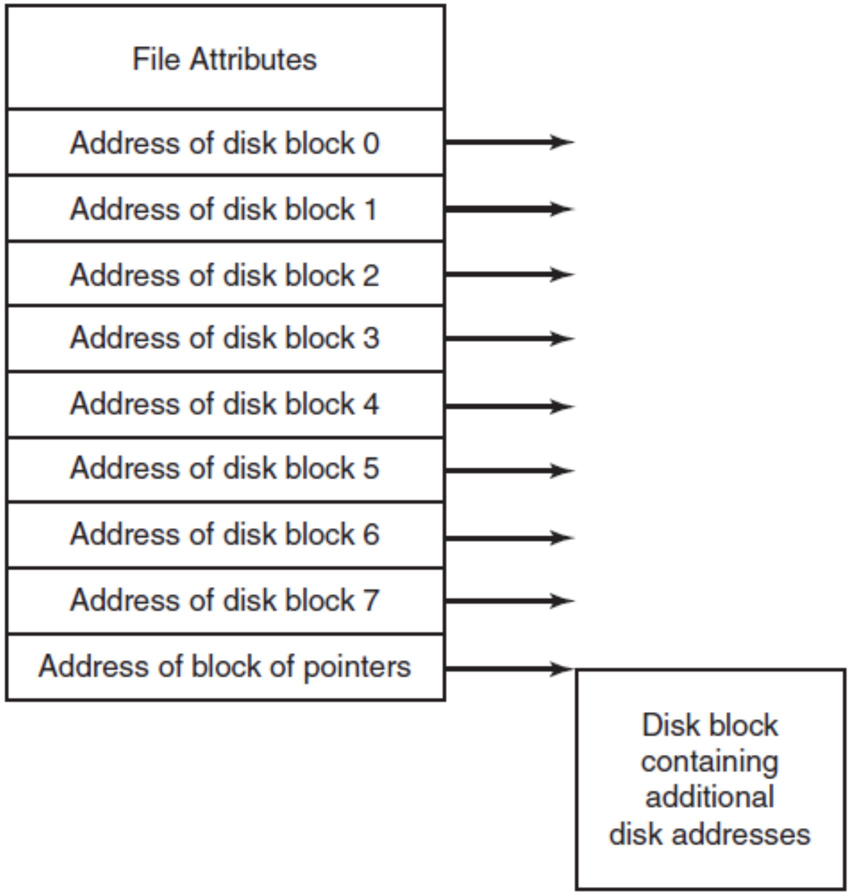


Figure 4-13. An example i-node

Implementing Directories (1)

- When a file is opened, the operating system uses the path name supplied by the user to locate the directory entry on the disk
- The directory entry provides the information needed to find the disk blocks
- Depending on the system, this information may be the disk address of the entire file (with contiguous allocation), the number of the first block (both linked-list schemes), or the number of the i-node

Implementing Directories (2)

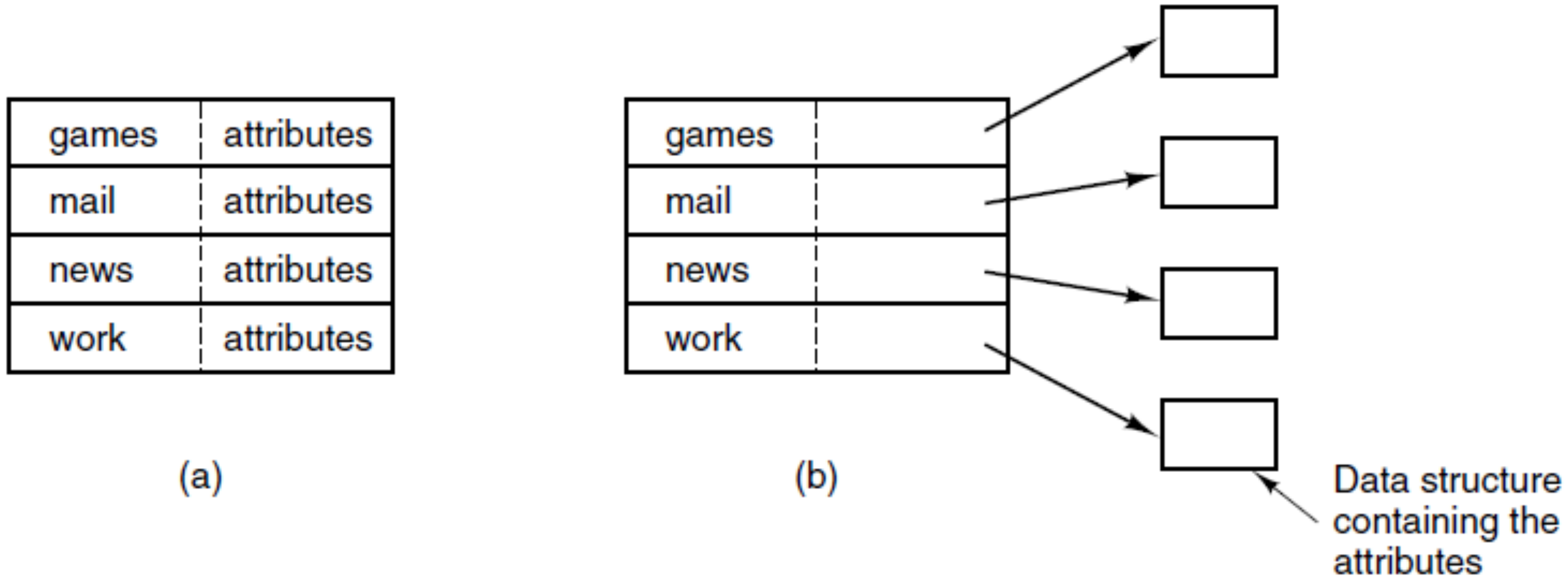


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry
(b) A directory in which each entry just refers to an i-node

Implementing Directories (3)

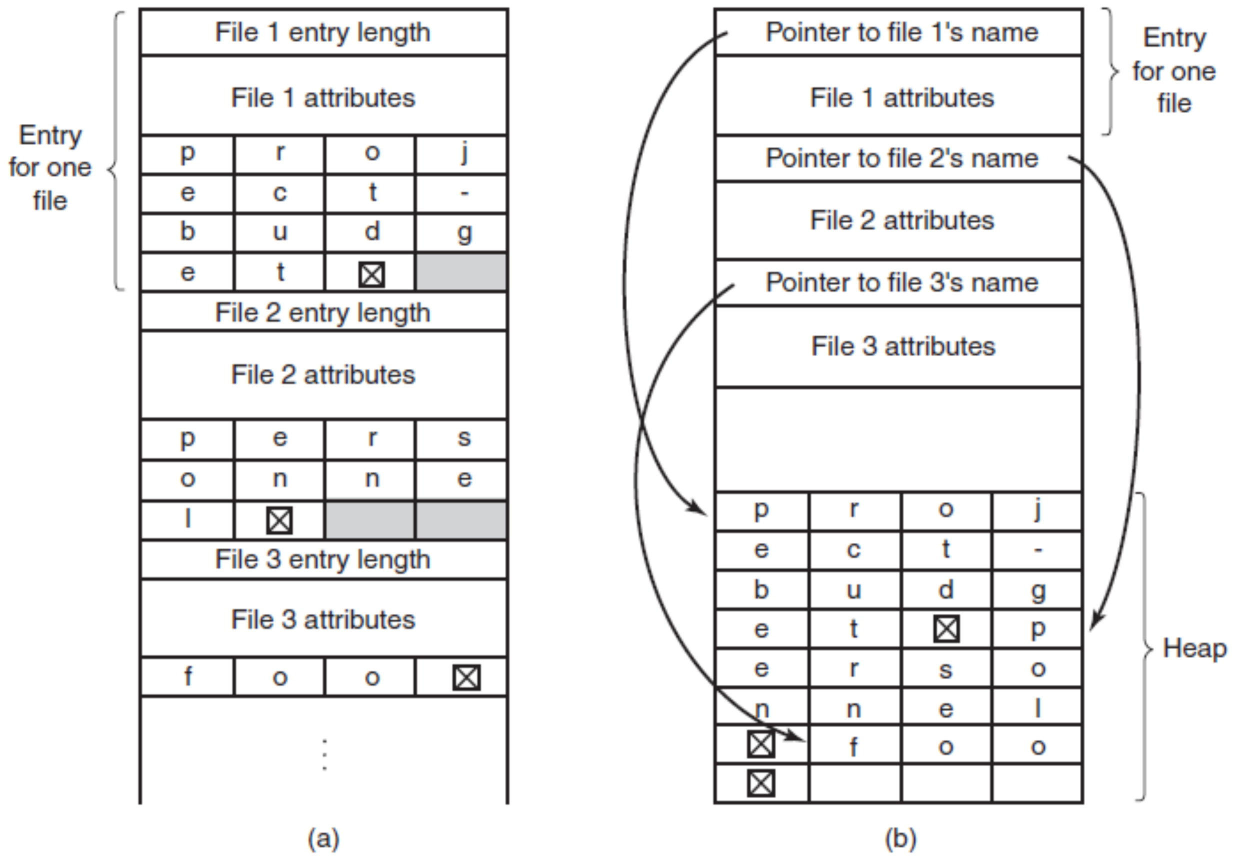


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files (1)

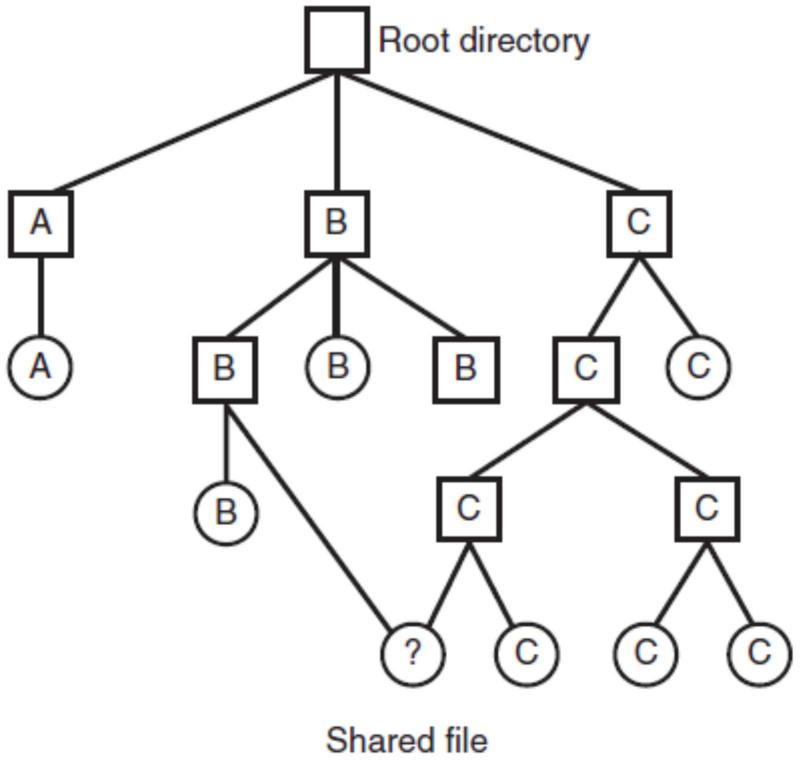


Figure 4-16. File system containing a shared file.

Shared Files (2)

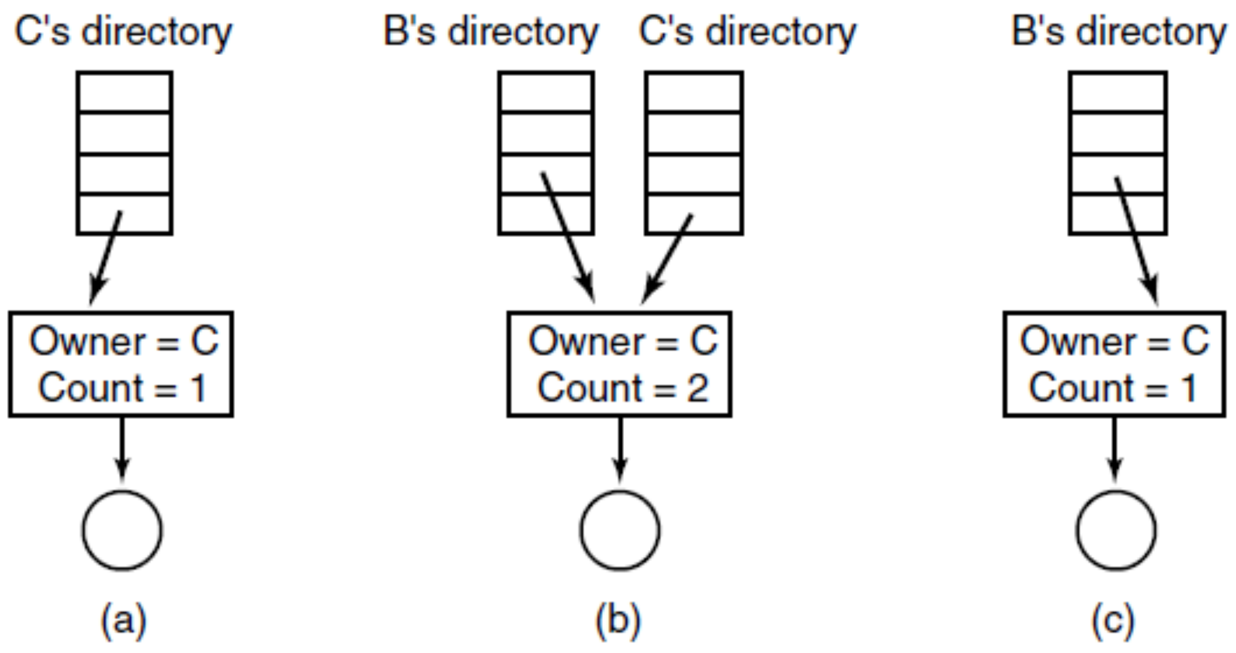


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file

The UNIX V7 File System (1)

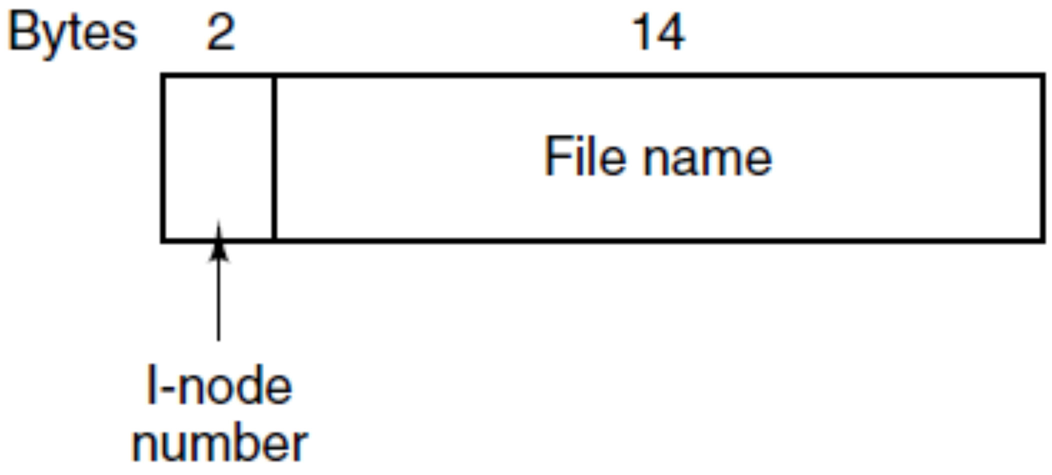


Figure 4-32. A UNIX V7 directory entry

The UNIX V7 File System (2)

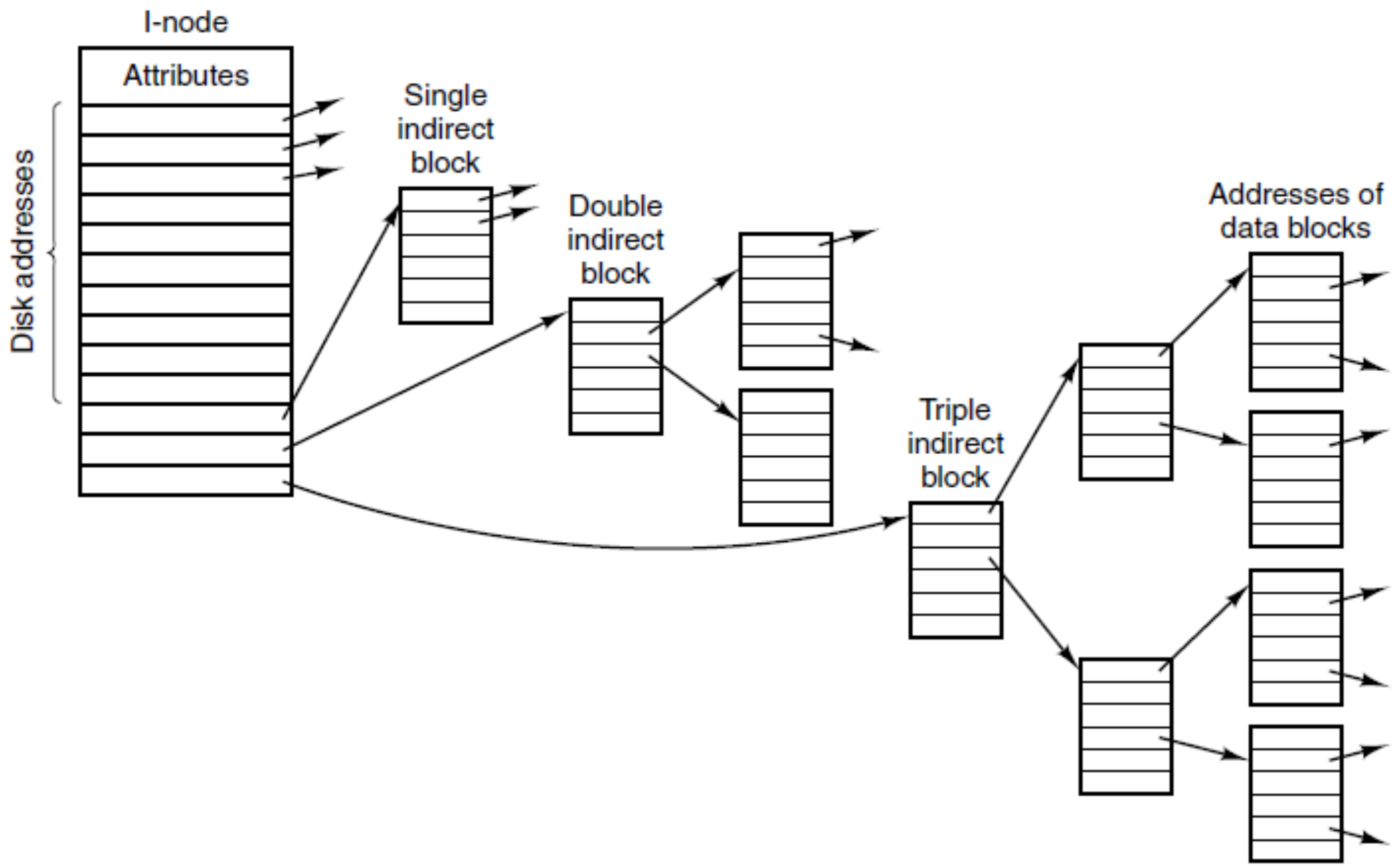
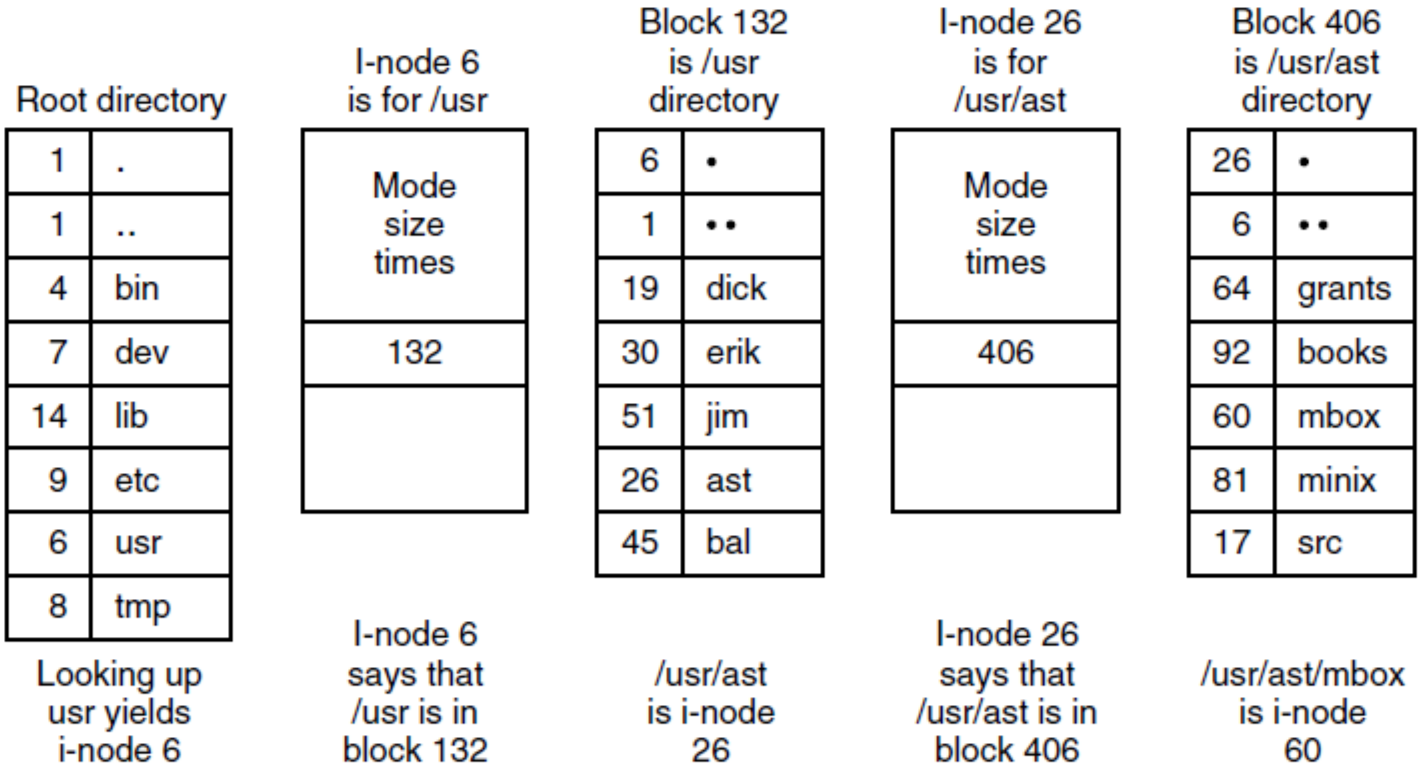


Figure 4-33. A UNIX i-node

The UNIX V7 File System (3)



Root directory
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
Looking up usr yields i-node 6
I-node 6 is for /usr
Mode size times
132
I-node 6 says that /usr is in block 132
Block 132 is /usr directory
6 •
1 ••
19 dick
30 erik
51 jim
26 ast
45 bal
/usr/ast is i-node 26
I-node 26 is for /usr/ast
Mode size times
406
I-node 26 says that /usr/ast is in block 406
Block 406 is /usr/ast directory
26 •
6 ••
64 grants
92 books
60 mbox
81 minix
17 src
/usr/ast/mbox is i-node 60

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

Log-Structured File Systems

- The entire disk is structured as a great big log. In this design, i-nodes still exist and even have the same structure as in UNIX, but they are now scattered all over the log, instead of being at a fixed position on the disk

Journaling File Systems

- The basic idea is to keep a log of what the file system is going to do before it does it, so that if the system crashes before it can do its planned work, upon rebooting the system can look in the log to see what was going on at the time of the crash and finish the job

Journaling File Systems

- Steps to remove a file in UNIX:
  - Remove file from its directory
  - Release i-node to the pool of free i-nodes
  - Return all disk blocks to pool of free disk blocks

Journaling File Systems

- For the 3 step process described in previous slide, the journaling file system first write a log entry listing the three actions to be completed
- The log entry is then written to disk

Journaling File Systems

- Only after the log entry has been written, the various operations begin
- After the operations complete successfully, the log entry is erased
- If the system now crashes, upon recovery the file system can check the log to see if any operations were pending

Virtual File Systems (1)

- The key idea is to abstract out that part of the file system that is common to all file systems and put that code in a separate layer that calls the underlying concrete file systems to actually manage the data

Virtual File Systems (2)

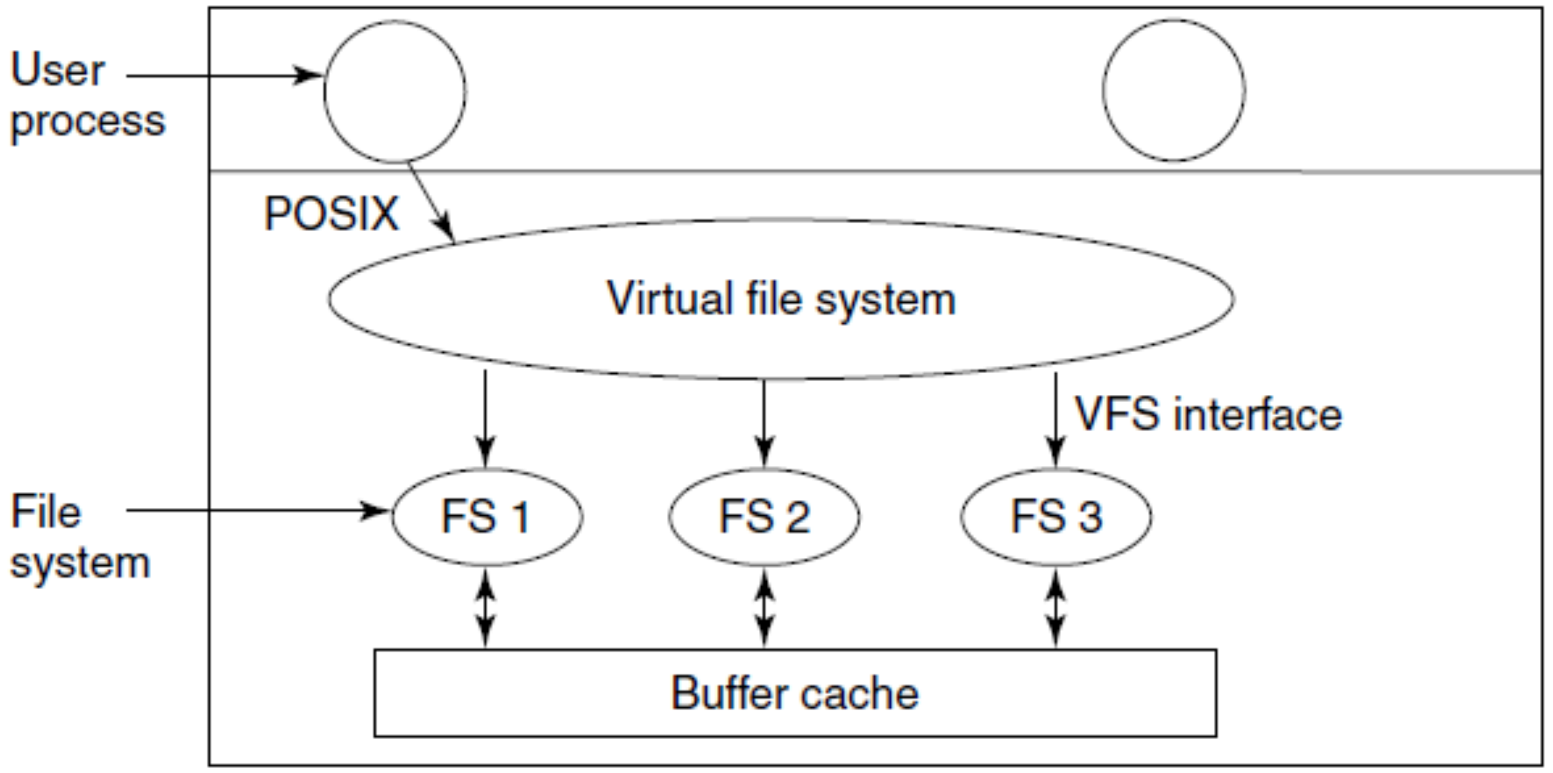


Figure 4-18. Position of the virtual file system.

Virtual File Systems (3)

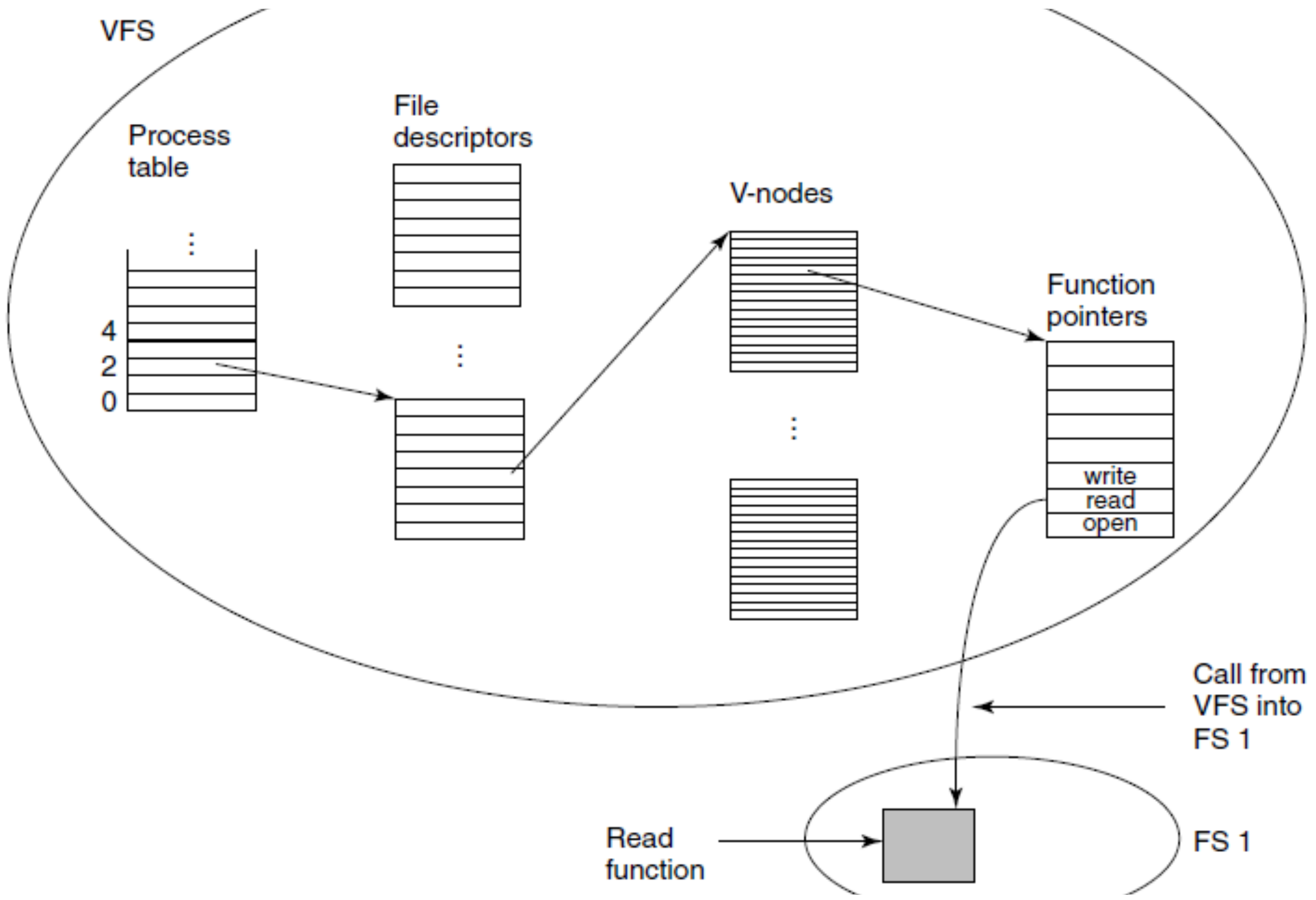


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a read

Disk Space Management (1)

- Files are normally stored on disk, so management of disk space is a major concern to file-system designers
- Two general strategies are possible for storing an *n* byte file:
  - *n* consecutive bytes of disk space are allocated
  - Or the file is split up into a number of (not necessarily) contiguous blocks

Disk Space Management (2)

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 (3)

- Considering Fig. 4-21, assume that all files are 4 KB
- The dashed curve of Fig. 4-21 shows the data rate for such a disk as a function of block size
- The solid curve of Fig. 4-21 shows the space efficiency as a function of block size

Disk Space Management (4)

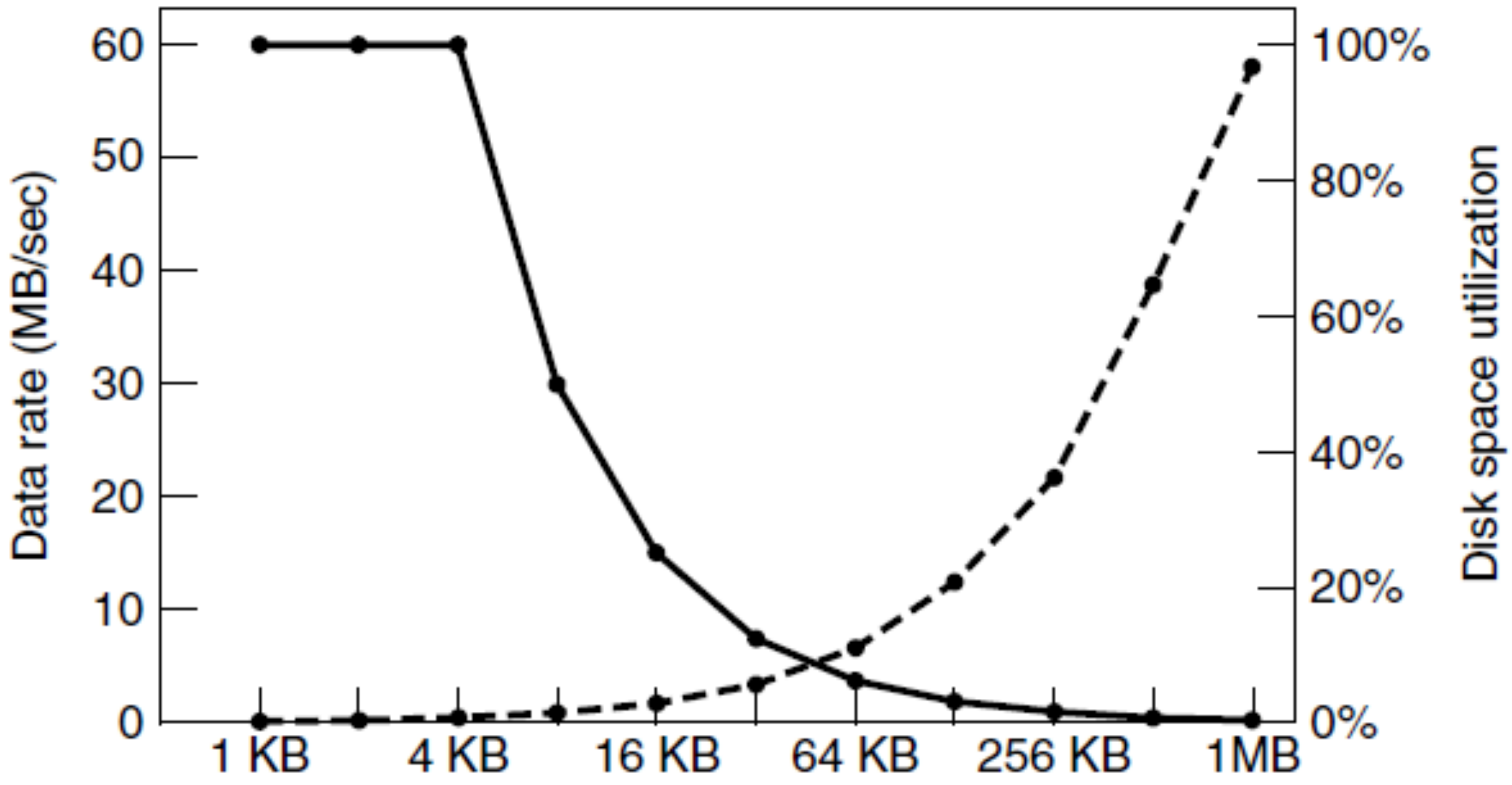


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Disk Space Management (5)

- The two curves of Fig 4-21 can be understood as follows:
  - The access time for a block is completely dominated by the seek time and rotational delay, so given that it is going to cost 9 msec to access a block, the more data that are fetched, the better
  - Hence the data rate goes up almost linearly with block size (until the transfers take so long that the transfer time begins to matter)

Disk Space Management (6)

- What the curves show of Fig. 4-21, however, is that performance and space utilization are inherently in conflict. Small blocks are bad for performance but good for disk space utilization
- For these data, no reasonable compromise is available. The size closest to where the two curves cross is 64 KB, but the data rate is only 6.6 MB/sec and the space efficiency is about 7%, neither of which is very good

Keeping Track of Free Blocks (1)

- To keep track of free blocks. Two methods are widely used, as shown in Fig. 4-22
- The first one consists of using a linked list of disk blocks, with each block holding as many free disk block numbers as will fit
- The other free-space management technique is the bitmap. A disk with *n* blocks requires a bitmap with *n* bits

Keeping Track of Free Blocks (2)

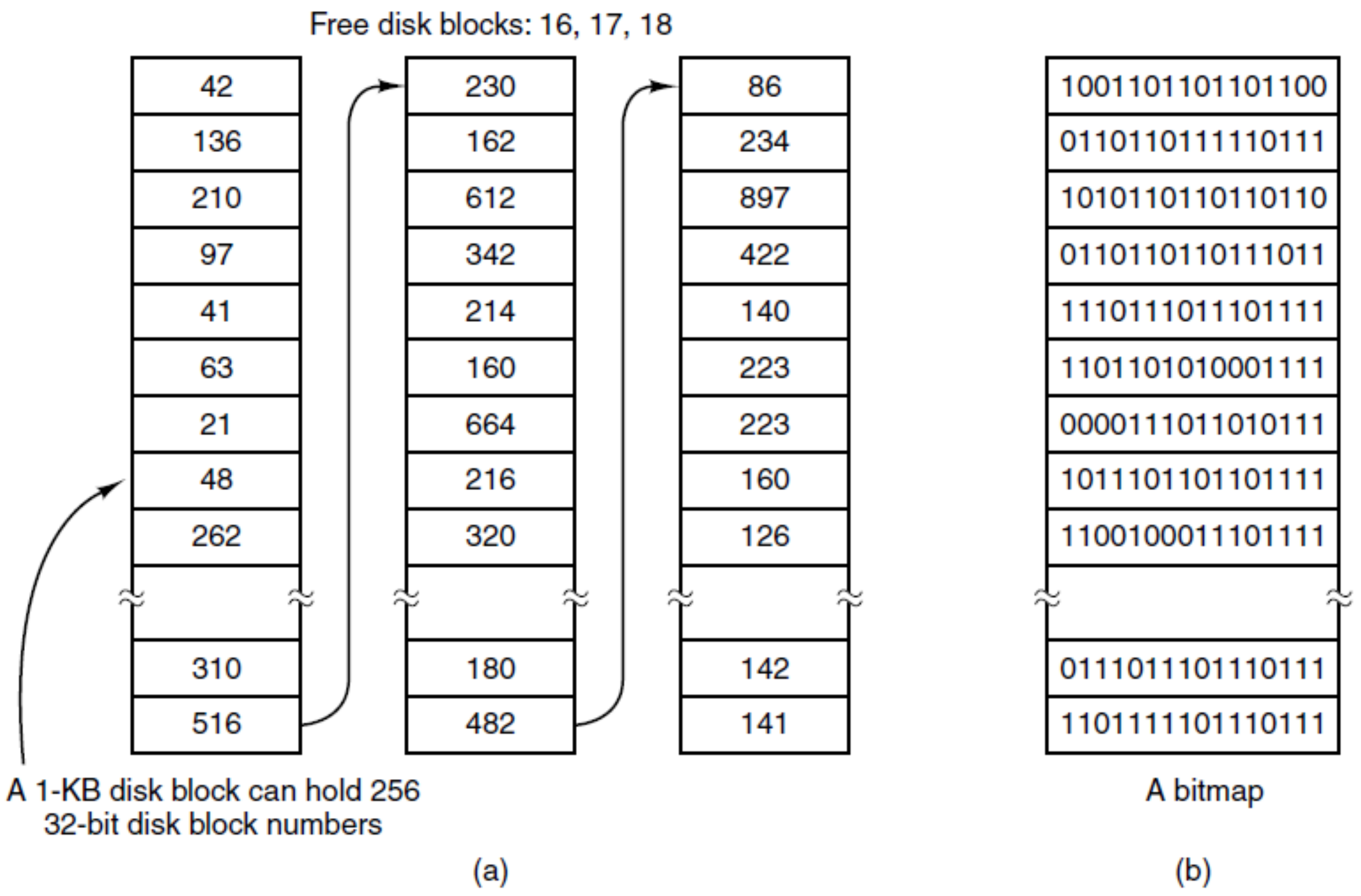


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Keeping Track of Free Blocks (3)

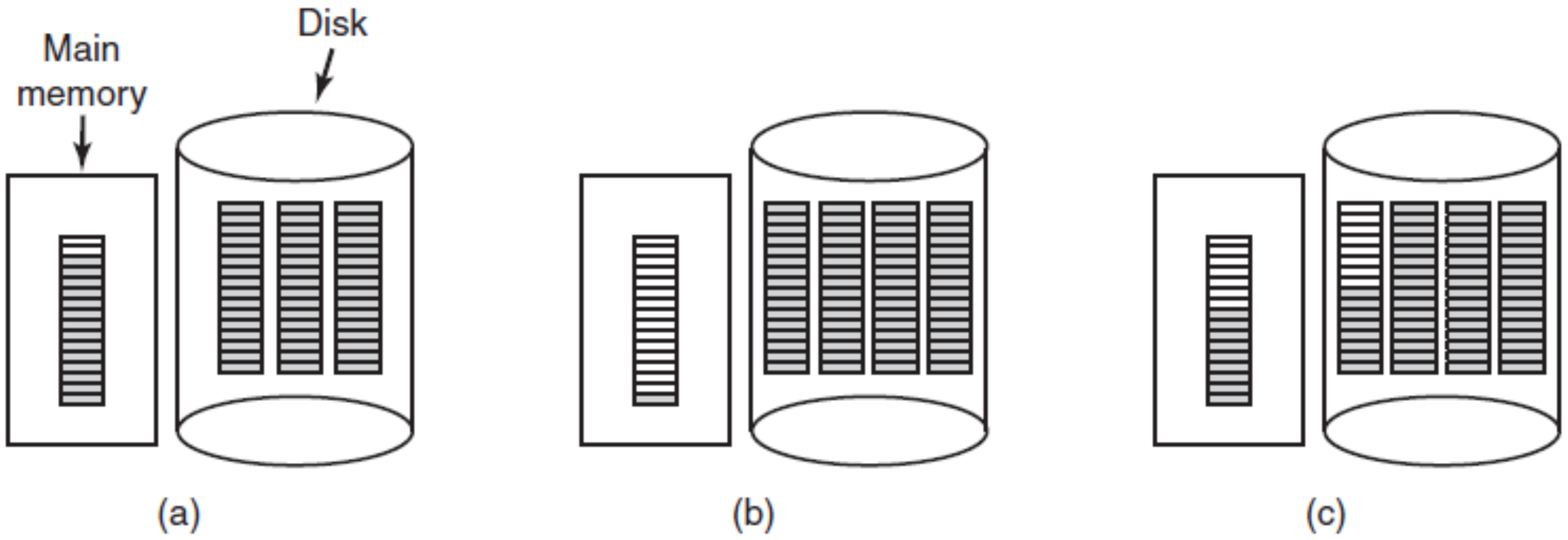


Figure 4-23. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

Disk Quotas (1)

- The system administrator assigns each user a maximum allotment of files and blocks, and the operating system makes sure that the users do not exceed their quotas

Disk Quotas (2)

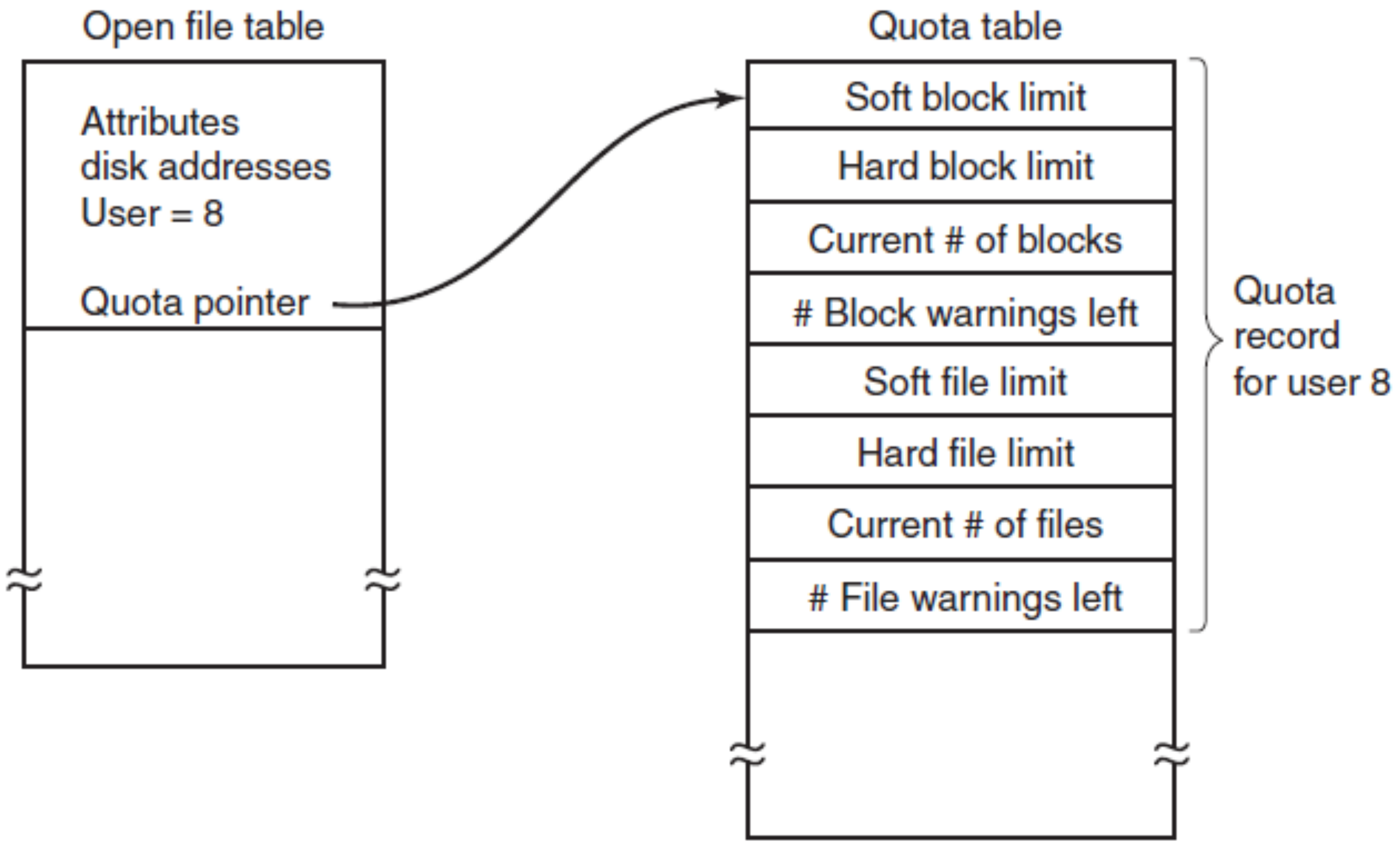


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

File System Backups (1)

- Backups to tape are generally made to handle one of two potential problems:
  - Recover from disaster
  - Recover from stupidity

File System Backups (2)

- Making a backup takes a long time and occupies a large amount of space, so doing it efficiently and conveniently is important
- First, should the entire file system be backed up or only part of it?
- It is not necessary to backup files if they can all be reinstalled from the manufacturer's Website or the installation DVD

File System Backups (3)

- In UNIX, all the special files (I/O devices) are kept in a directory /*dev*. Not only is backing up this directory not necessary, it is downright dangerous because the backup program would hang forever if it tried to read each of these to completion
- It is also wasteful to back up files that have not changed since the previous backup, which leads to the idea of incremental dumps

File System Backups (4)

- Two strategies can be used for dumping a disk to a backup disk: a physical dump or a logical dump
- A physical dump starts at block 0 of the disk, writes all the disk blocks onto the output disk in order, and stops when it has copied the last one

File System Backups (5)

- A logical dump starts at one or more specified directories and recursively dumps all files and directories found there that have changed since some given base date (e.g., the last backup for an incremental dump or system installation for a full dump)

File System Backups (6)

- Most UNIX systems use this algorithm in Fig 4.25
- In the figure we see a file tree with directories (squares) and files (circles). The shaded items have been modified since the base date and thus need to be dumped. The unshaded ones do not need to be dumped

File System Backups (7)

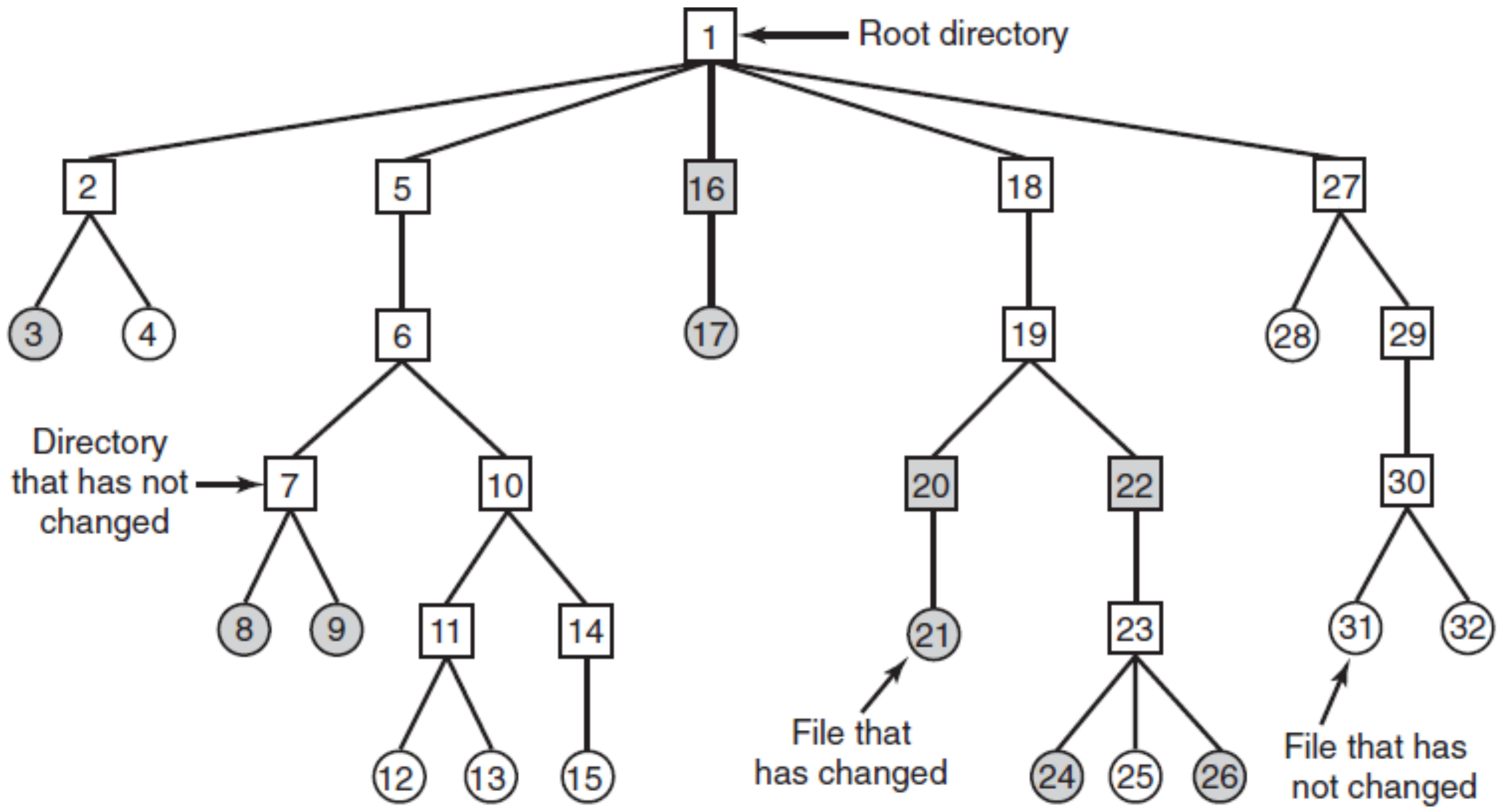


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

File System Backups (8)

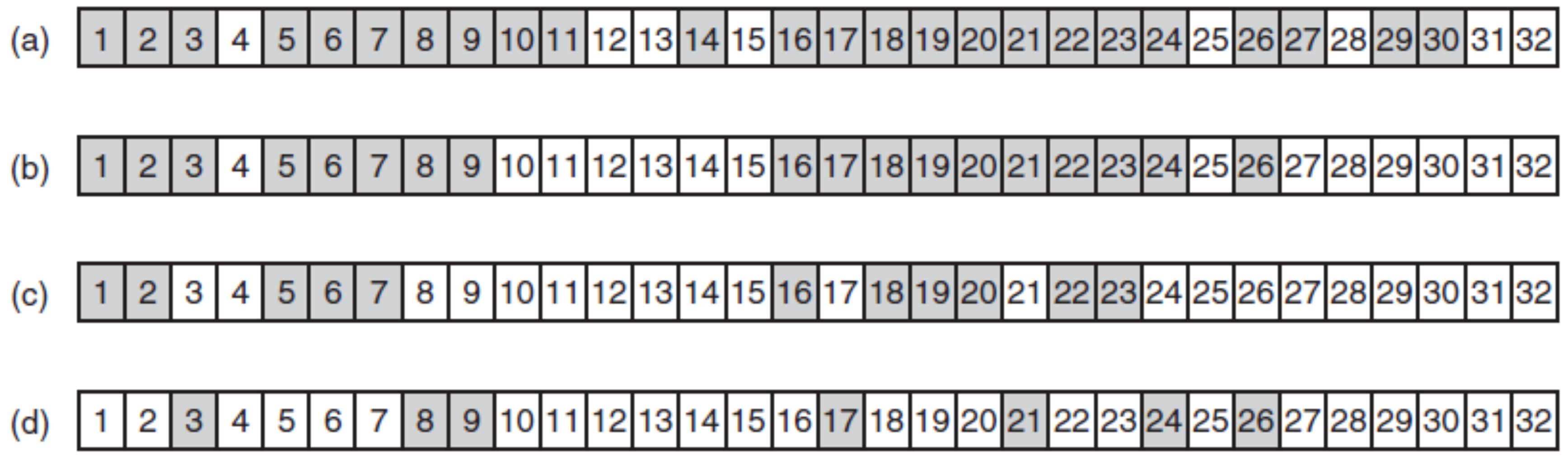


Figure 4-26. Bitmaps used by the logical dumping algorithm.

File System Consistency (1)

- Many file systems read blocks, modify them, and write them out later. If the system crashes before all the modified blocks have been written out, the file system can be left in an inconsistent state

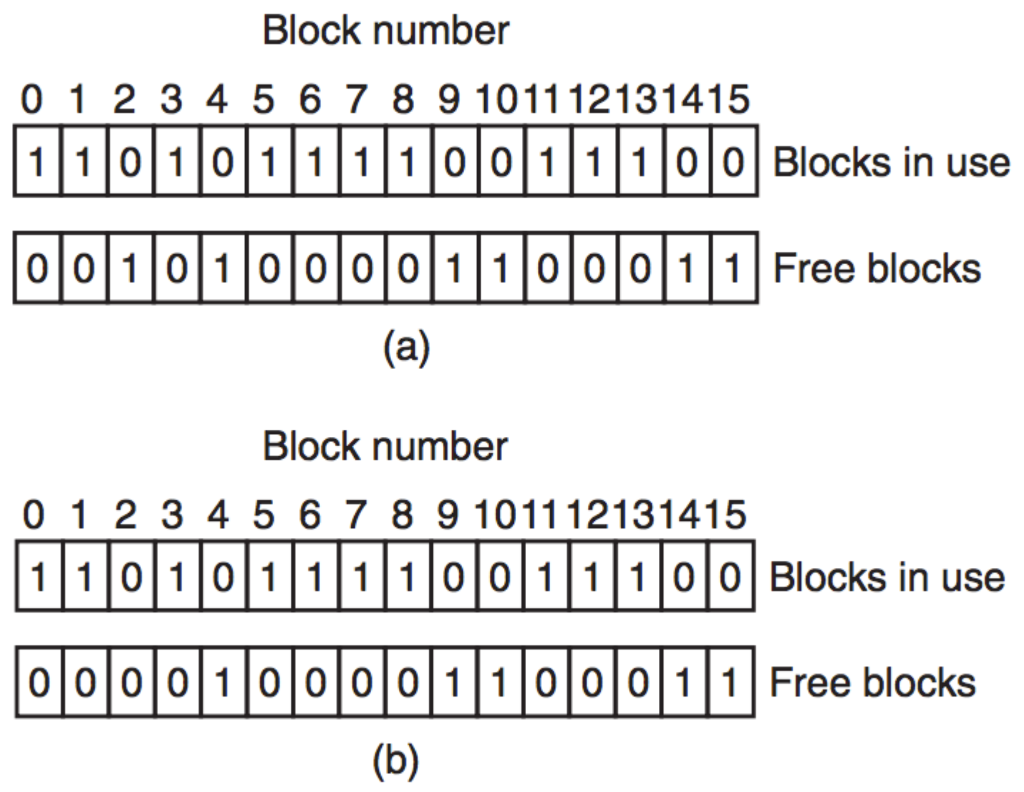
File System Consistency (2)

- Most computers have a utility program that checks file-system consistency
- For example, UNIX has *fsck*; Windows has *sfc* (and others). This utility can be run whenever the system is booted, especially after a crash
- Two kinds of consistency checks can be made: blocks and files

File System Consistency (3)

- If the file system is consistent, each block will have a 1 either in the first table or in the second table, as illustrated in Fig. 4-27a. However, as a result of a crash, the tables might look like Fig. 4-27b, in which block 2 does not occur in either table. It will be reported as being a missing block

File System Consistency (4)



(a)

Block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(b)

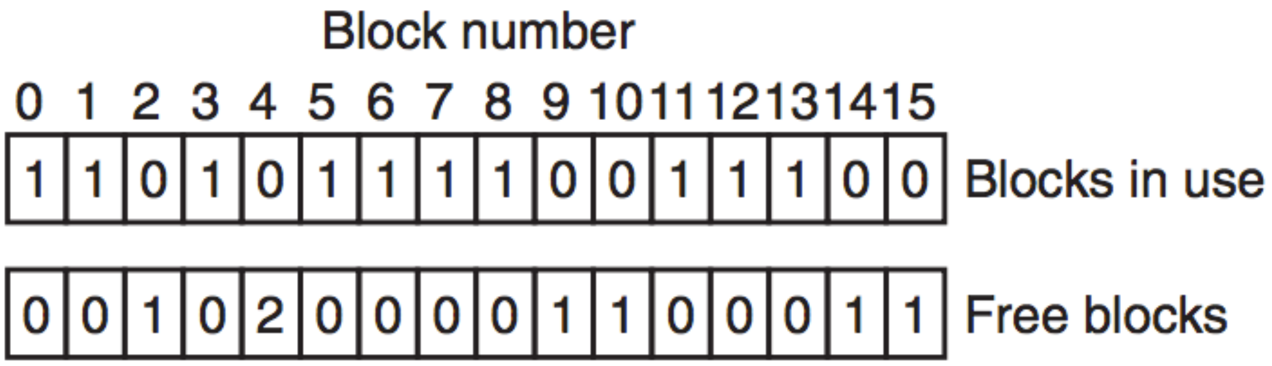
Block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1

Figure 4-27. File system states
(a) Consistent (b) Missing block

File System Consistency (5)

- Another situation that might occur is that of Fig. 4-27c. Here we see a block, number 4, that occurs twice in the free list (duplicates can occur only if the free list is really a list; with a bitmap it is impossible). The solution here is also simple: rebuild the free list

File System Consistency (6)



Block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1

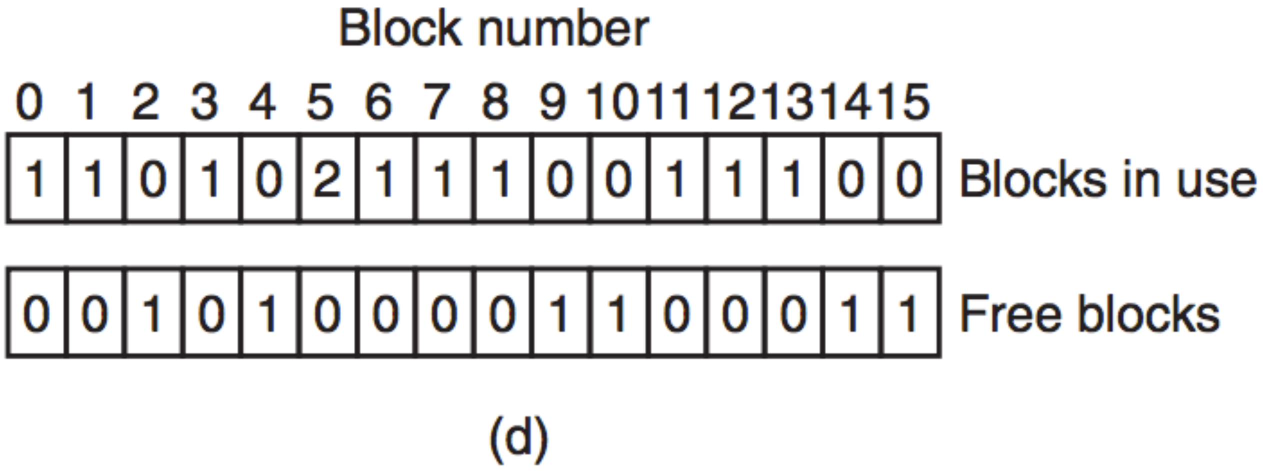
(c)

Figure 4-27. File-system states. (c) Duplicate

File System Consistency (7)

- The worst thing that can happen is that the same data block is present in two or more files, as shown in Fig. 4-27(d) with block 5
- If either of these files is removed, block 5 will be put on the free list. The same block is now both in use and free at the same time. If both files are removed, the block will be put onto the free list twice
- The appropriate action for the file-system checker to take is to allocate a free block, copy the contents of block 5 into it, and insert the copy into one of the files

File System Consistency (8)



Block number

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(d)

Figure 4-27. File system states.(d) Duplicate data block.

File System Performance (1)

- Access to disk is much slower than access to memory. Reading a 32-bit memory word might take 10 nsec. Reading from a hard disk might proceed at 100 MB/sec
- As a result of this difference in access time, many file systems have been designed with various optimizations to improve performance

File System Performance (2)

- The most common technique used to reduce disk accesses is the block cache/buffer cache
- Operation of the cache is illustrated in Fig. 4-28
- A second technique Block Read Ahead, for improving perceived file-system performance is to try to get blocks into the cache before they are needed to increase the hit rate
- Another important technique is to reduce the amount of disk-arm motion by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder

File System Performance (3)

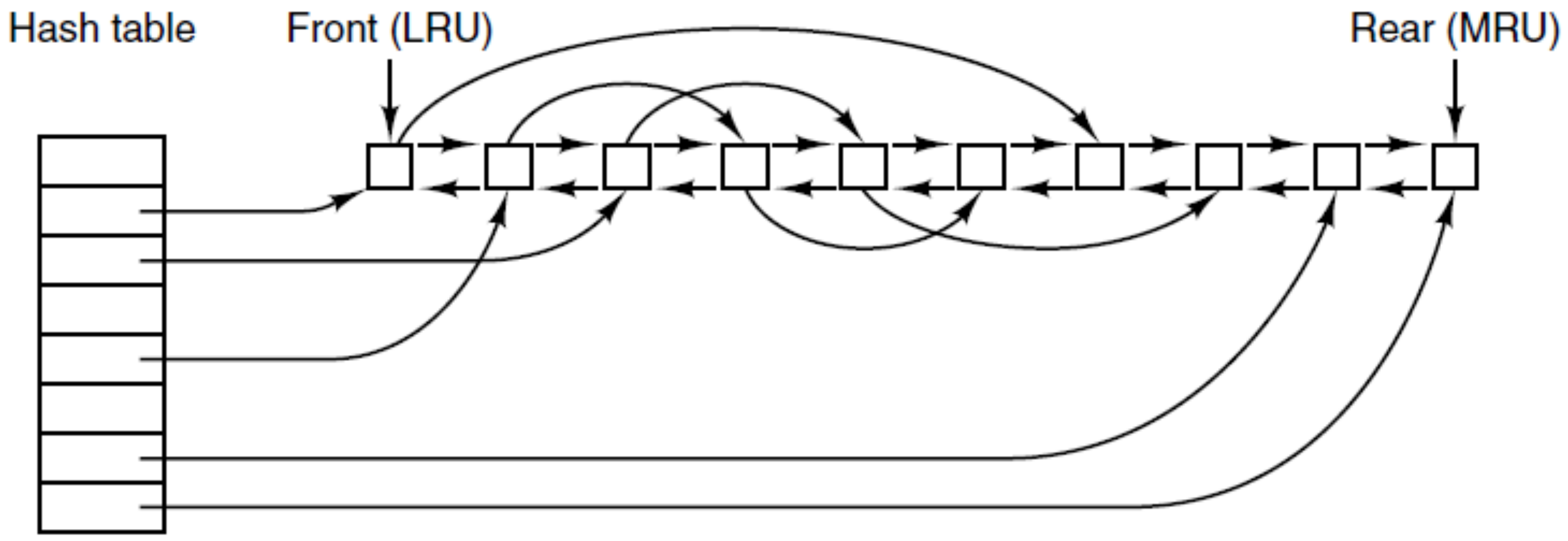


Figure 4-28. The buffer cache data structures.

Reducing Disk Arm Motion

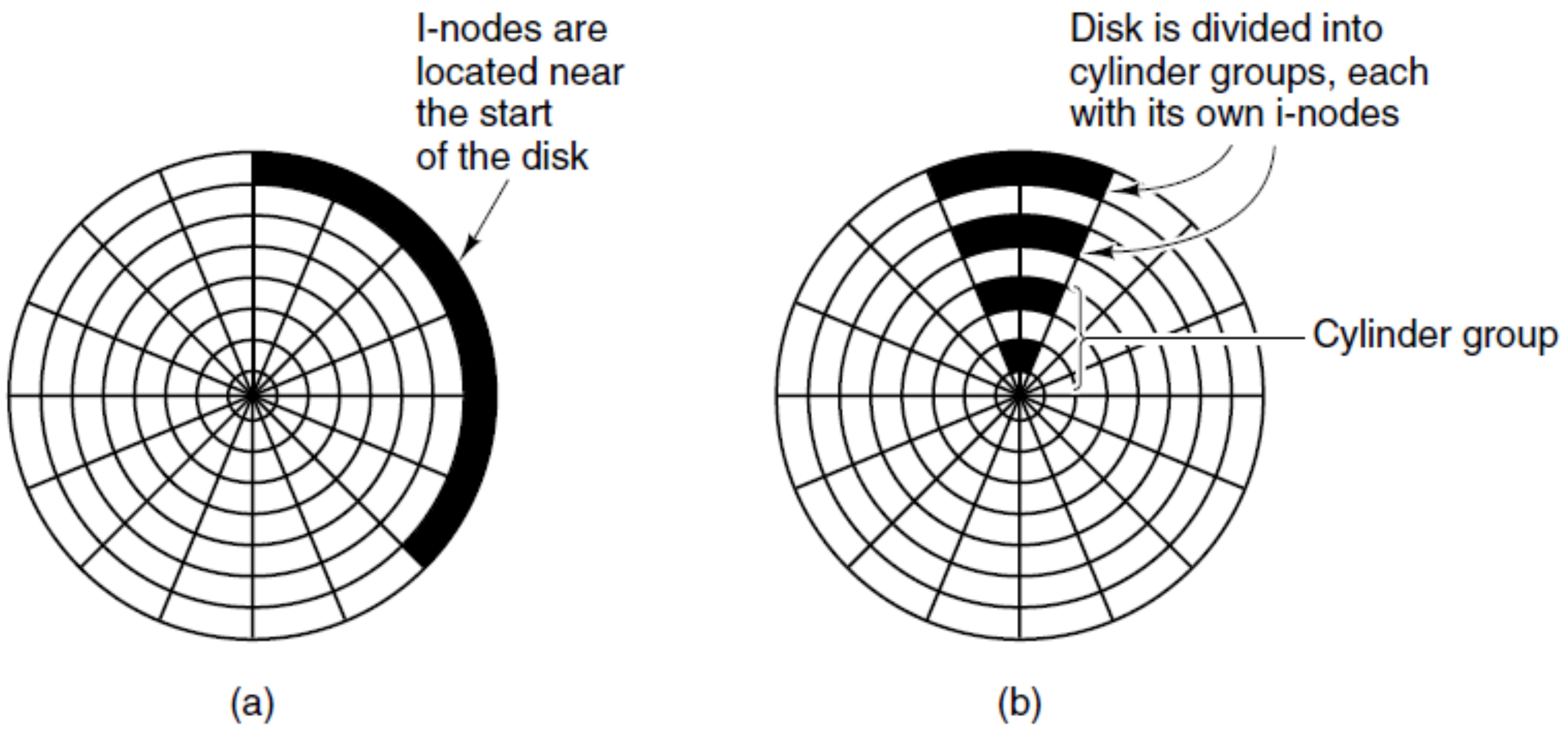

I-nodes are located near the start of the disk
(a)
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(b)

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

Defragmenting Disks (1)

- When the operating system is initially installed, the programs and files it needs are installed consecutively starting at the beginning of the disk, each one directly following the previous one
- All free disk space is in a single contiguous unit following the installed files

Defragmenting Disks (2)

- Files are created and removed and typically the disk becomes badly fragmented, with files and holes all over the place
- Windows has a program **defrag**, that moves files around to make them contiguous and to put all (or at least most) of the free space in one or more large contiguous regions on the disk

The MS-DOS File System (1)

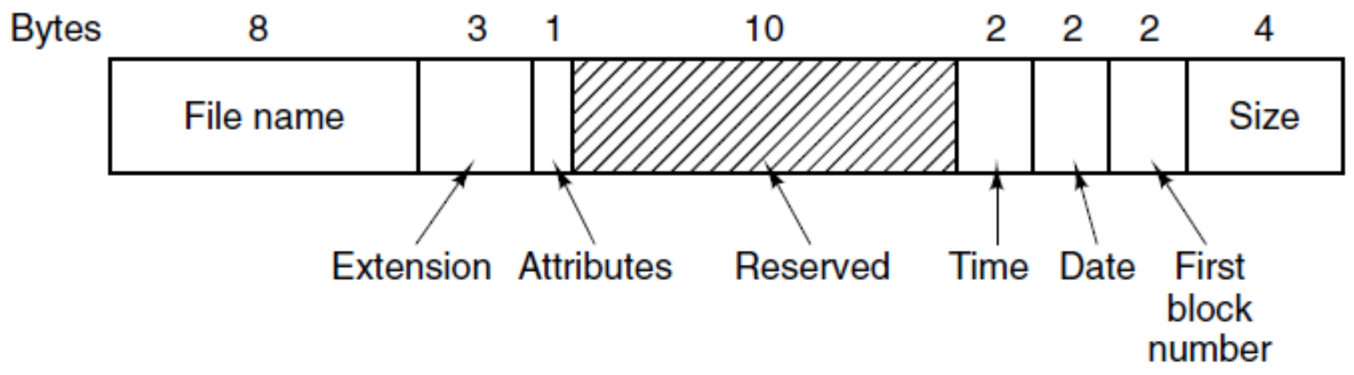
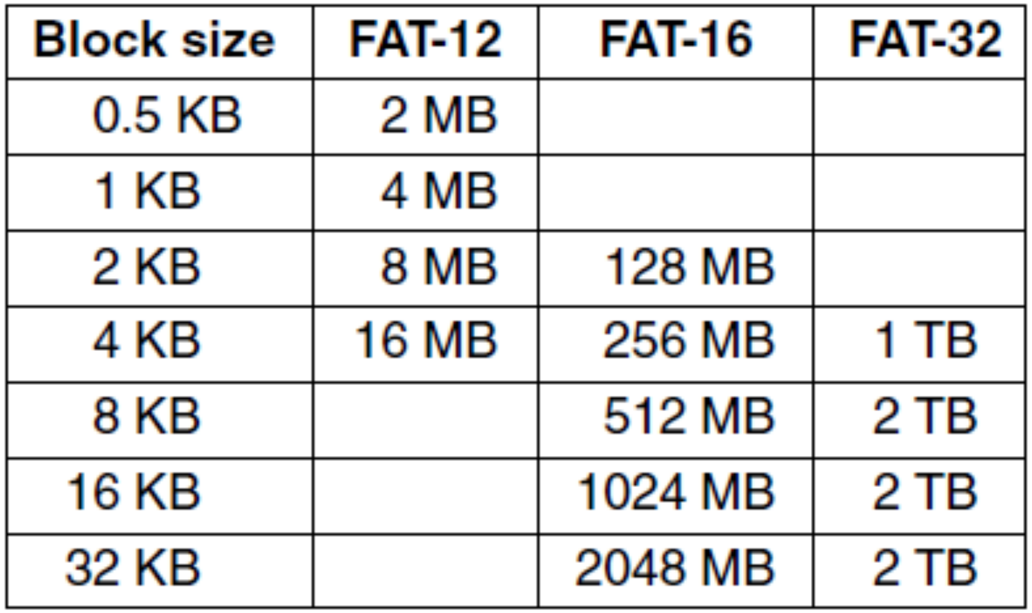


Figure 4-30. The MS-DOS directory entry.

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 ISO 9660 File System

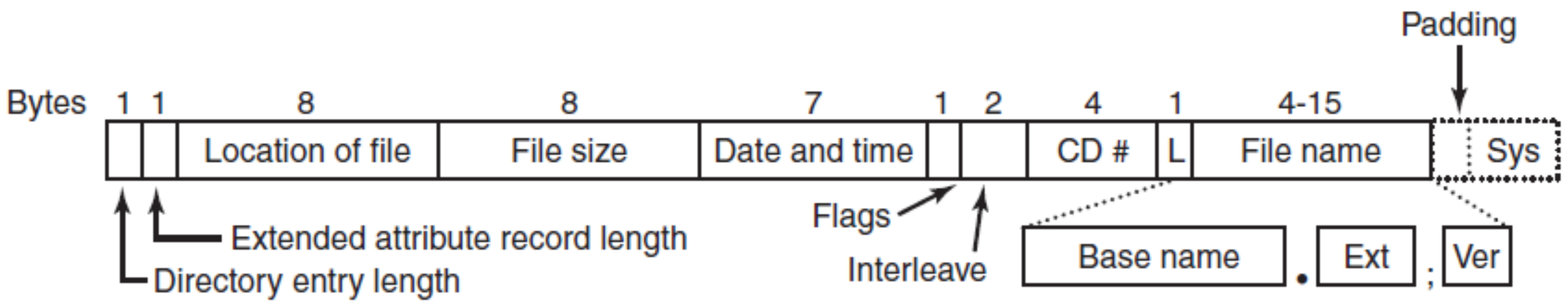


Figure 4-35. The ISO 9660 directory entry.

Rock Ridge Extensions

- PX - POSIX attributes.
- PN - Major and minor device numbers.
- SL - Symbolic link.

Joliet Extensions

- The major extensions provided by Joliet are:
  - Long file names
  - Unicode character set
  - Directory nesting deeper than eight levels
  - Directory names with extensions

END

Week 10 – Lecture

References

- Tanenbaum & Bos, Modern Operating Systems: 4th edition, 2013 Prentice-Hall, Inc.