File Systems

Week 09-Lecture 2

File Systems Structure

Team

- **Instructors**
- Giancarlo Succi
- Joseph Brown

- **Teaching Assistants**
- Vladimir Ivanov
- Stanislav Litvinov
- Alexey Reznik
- Munir Makhmutov
- Hamna Aslam

Sources

- These slides have been adapted from the original slides of the adopted book:
- Tanenbaum & Bo, 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

The UNIX V7 File System (1)

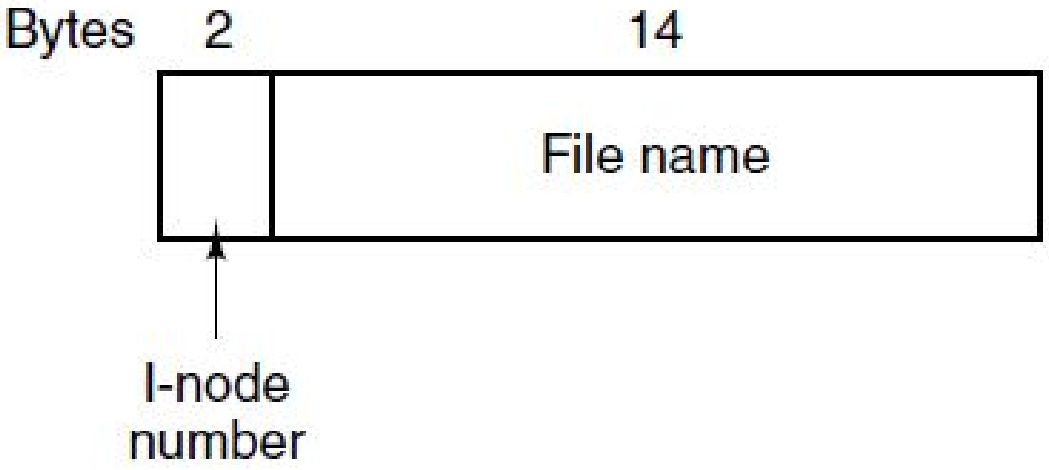


Figure 4-32. A UNIX V7 directory entry.

The UNIX V7 File System (2)

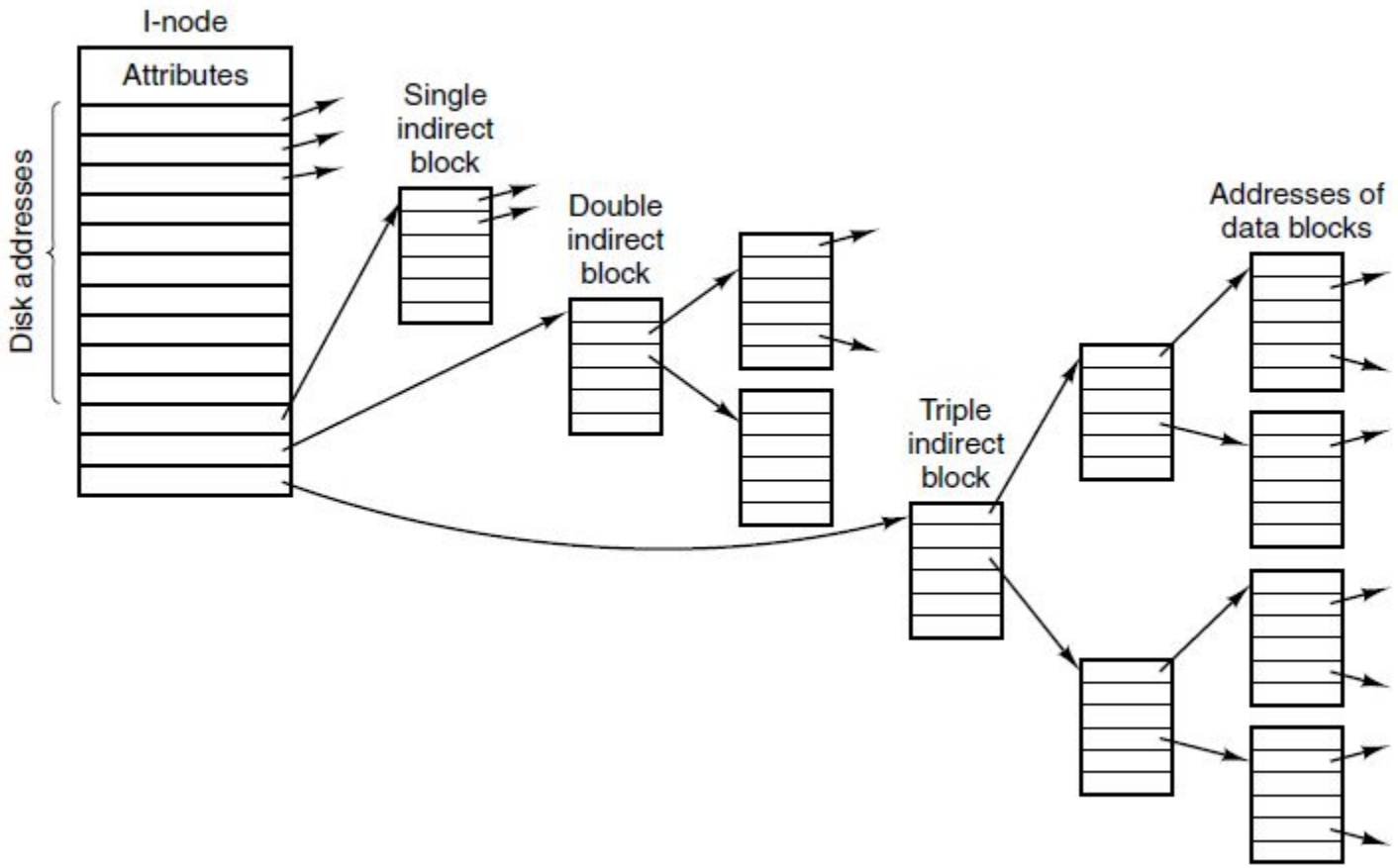


Figure 4-33. A UNIX i-node

The UNIX V7 File System (3)

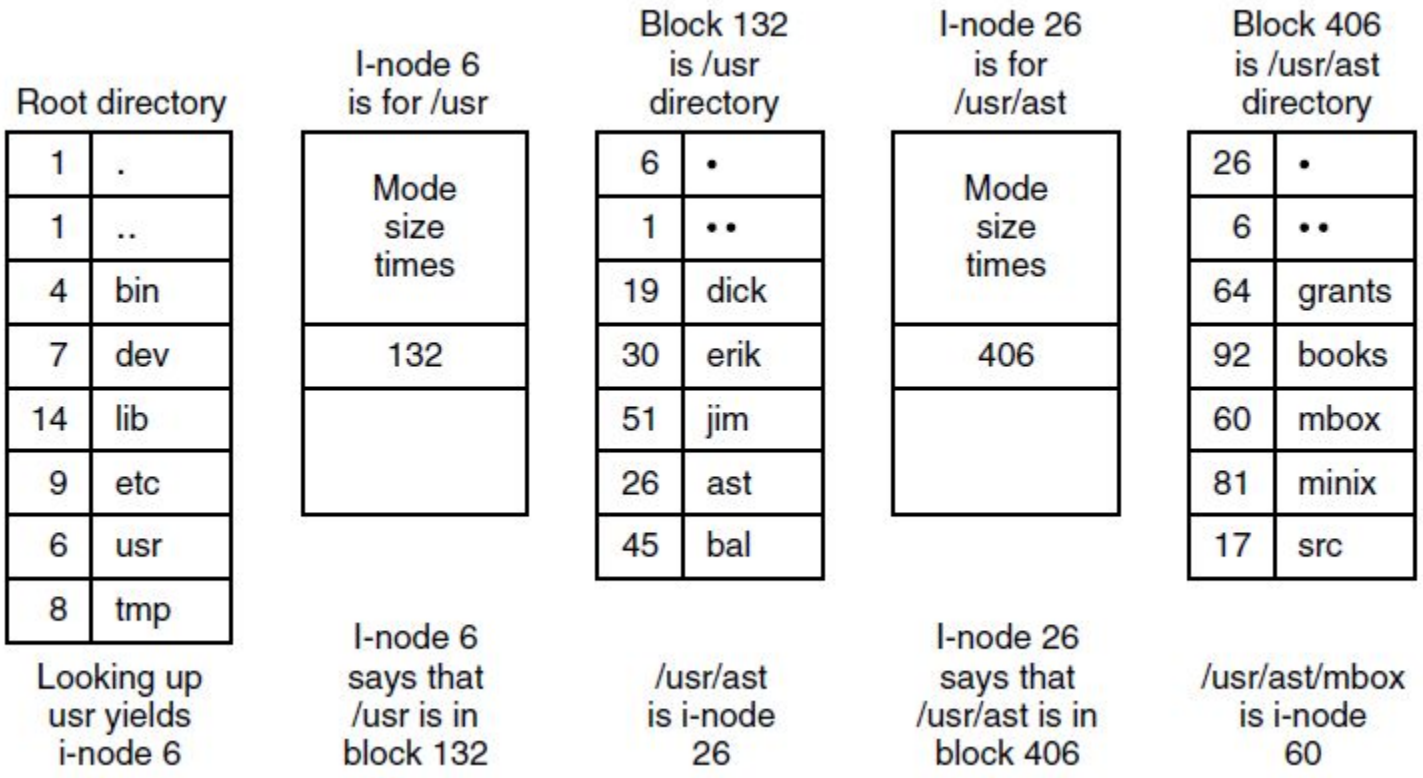


Figure 4-34. The steps in looking up */usr/ast/mbox*.

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:

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.

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)

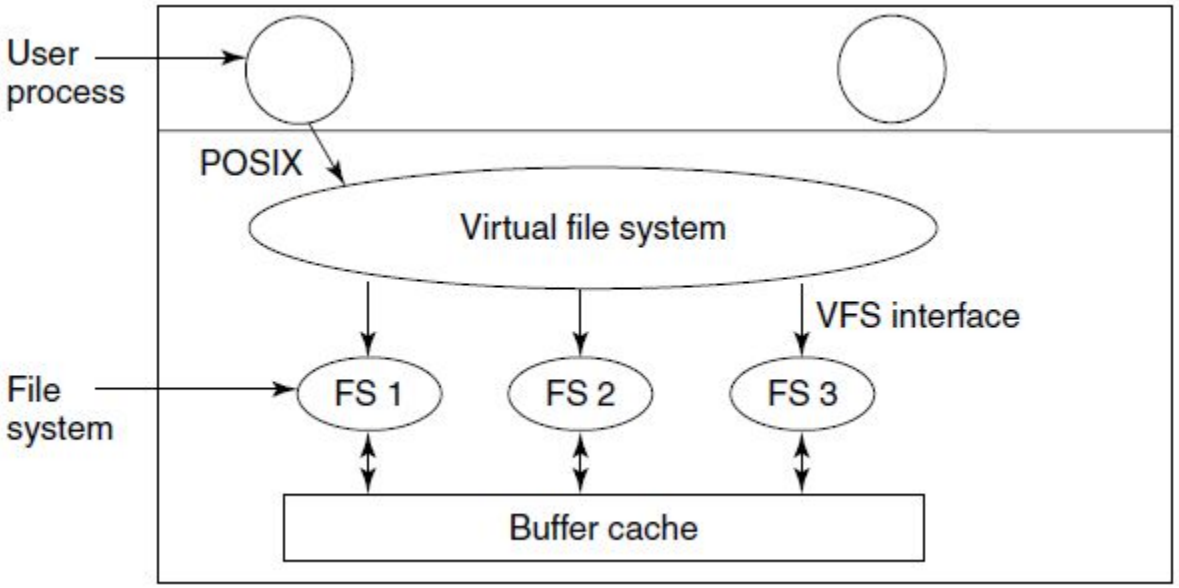


Figure 4-18. Position of the virtual file system.

Virtual File Systems (3)

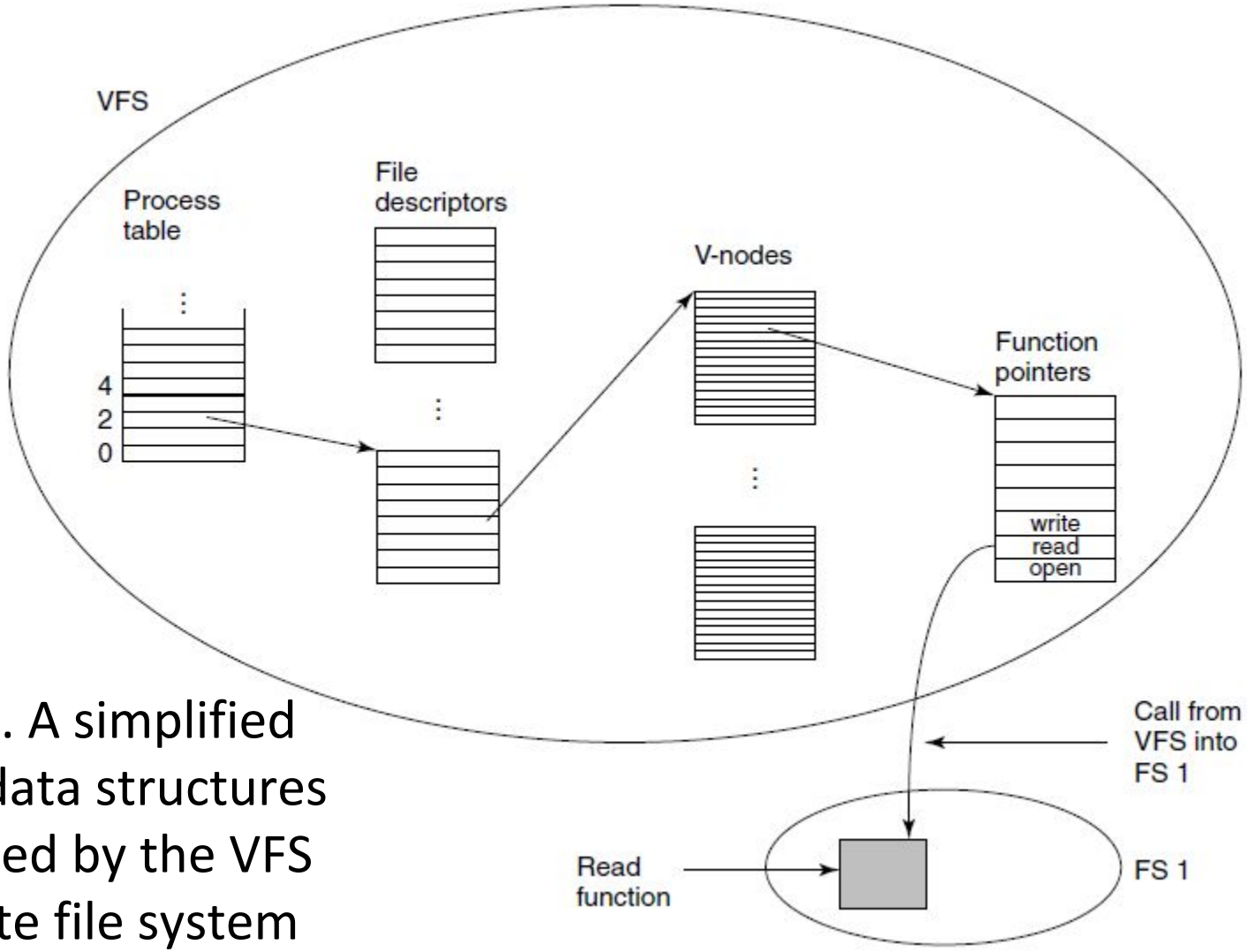


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a *read*.

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:

1) n consecutive bytes of disk space are allocated.
2) 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** (next slide)**,** 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)

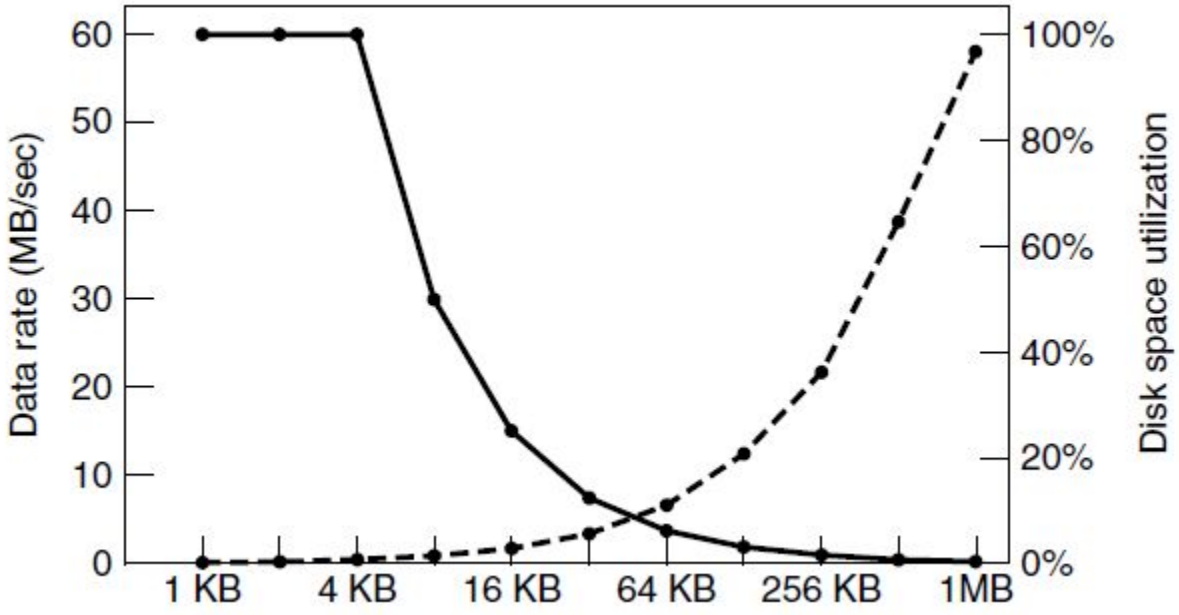


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

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**(next slide).

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)

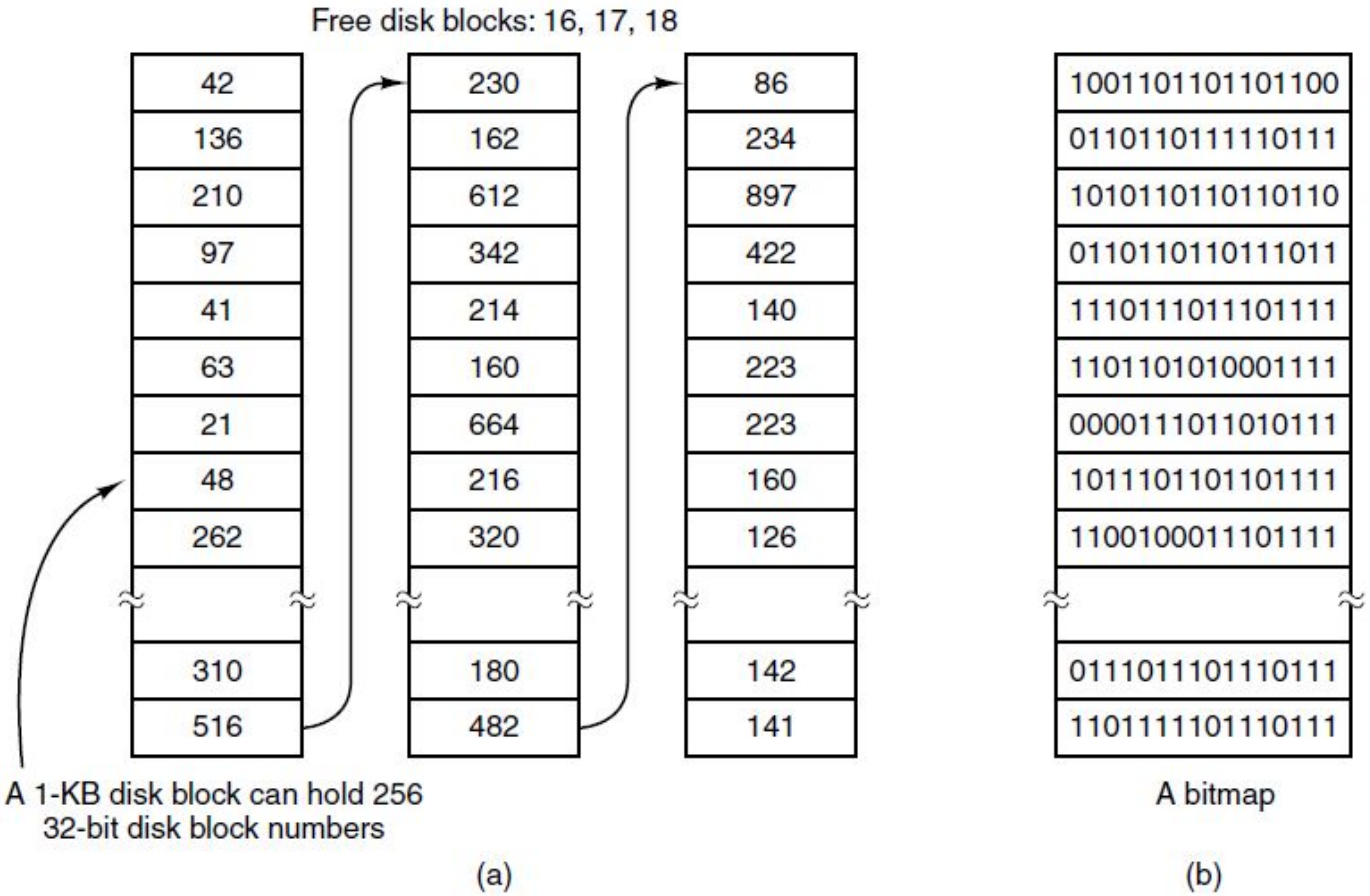


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Keeping Track of Free Blocks (3)

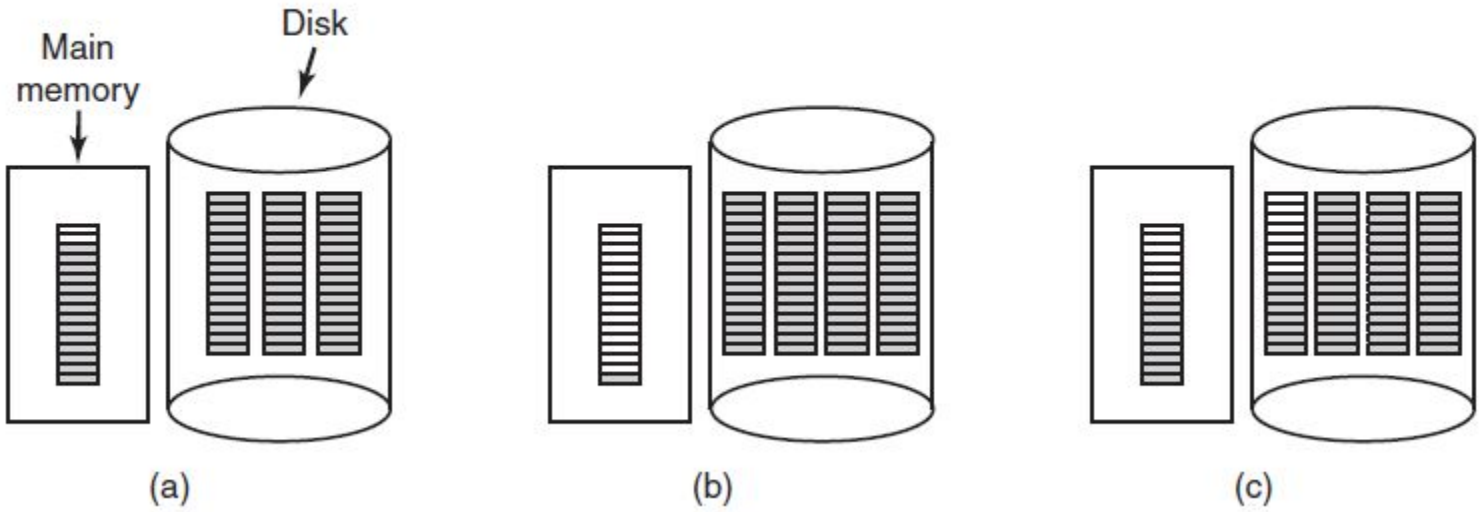


Figure 4-23. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

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)

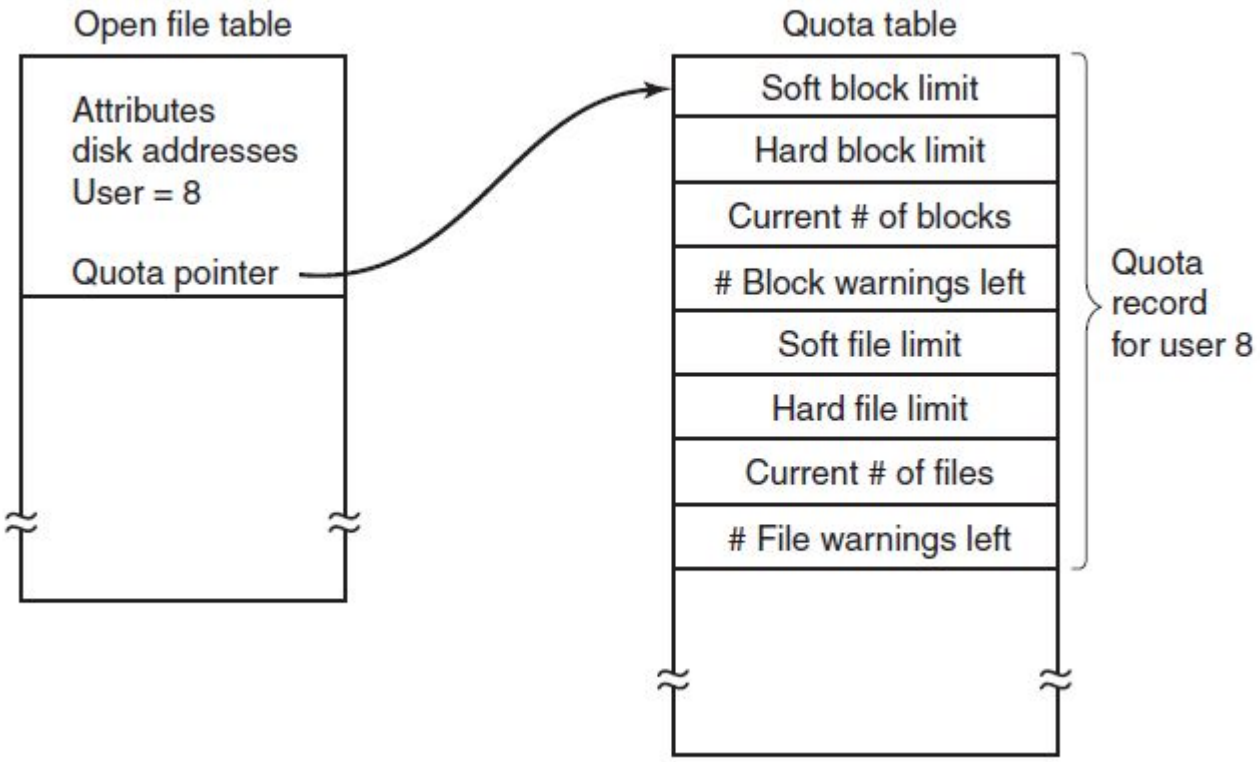


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

File System Backups (1)

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

1. Recover from disaster.
2. 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, next slide).
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)

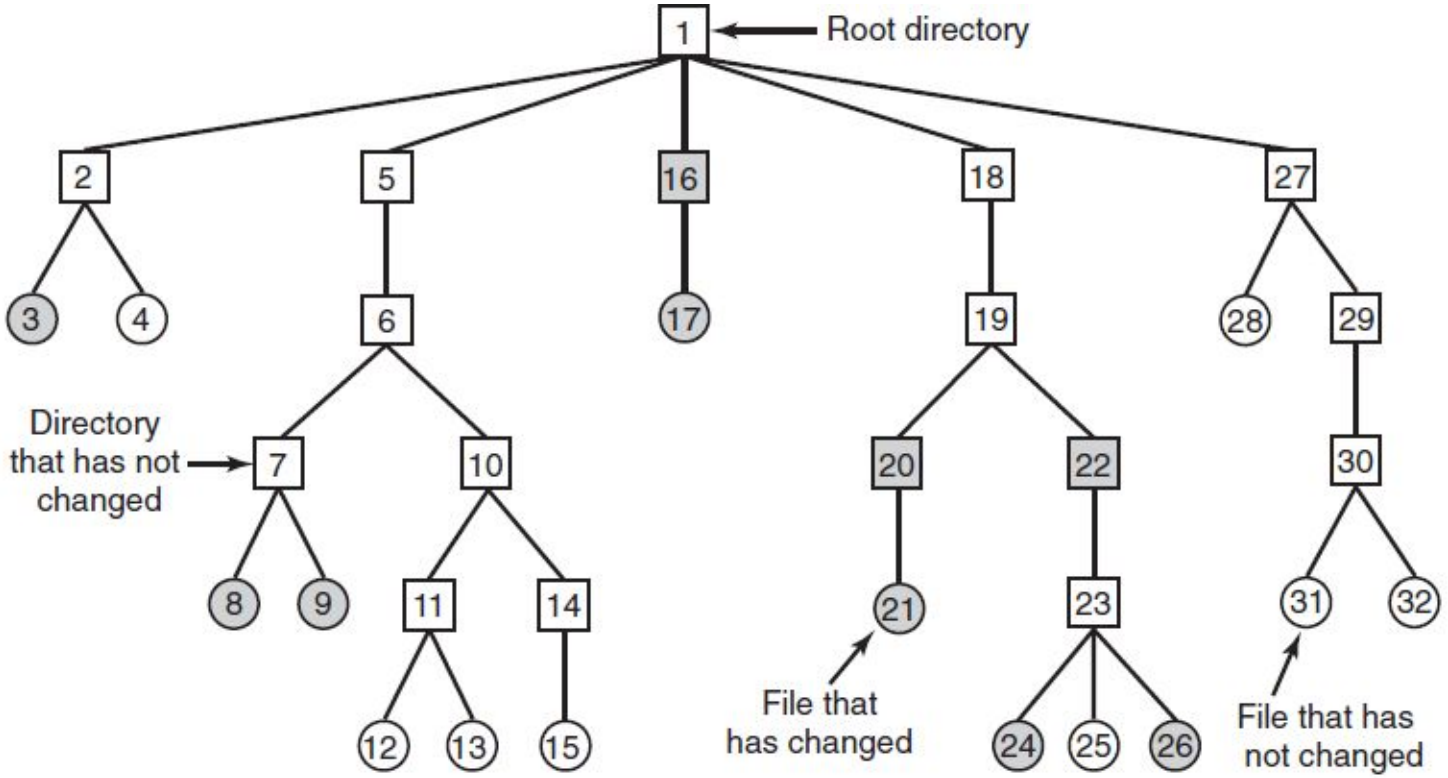


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

File System Backups (8)

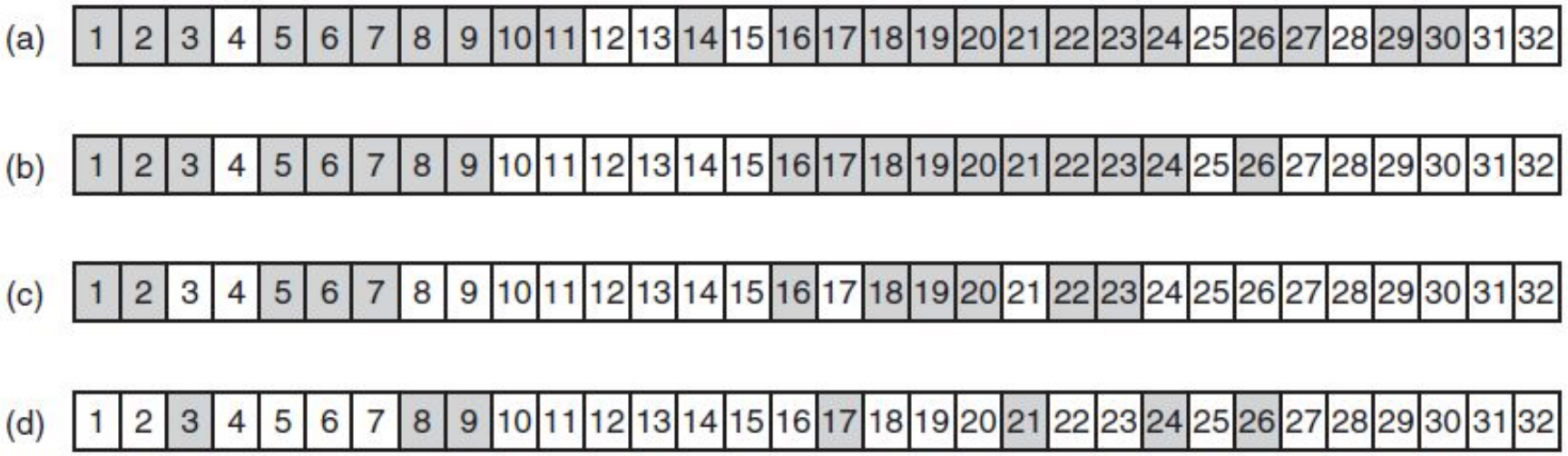


Figure 4-26. Bitmaps used by the logical dumping algorithm.

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. 4-27(a)**. However, as a result of a crash, the tables might look like **Fig. 4-27(b)**, in which block 2 does not occur in either table. It will be reported as being a **missing block**.

File System Consistency (4)

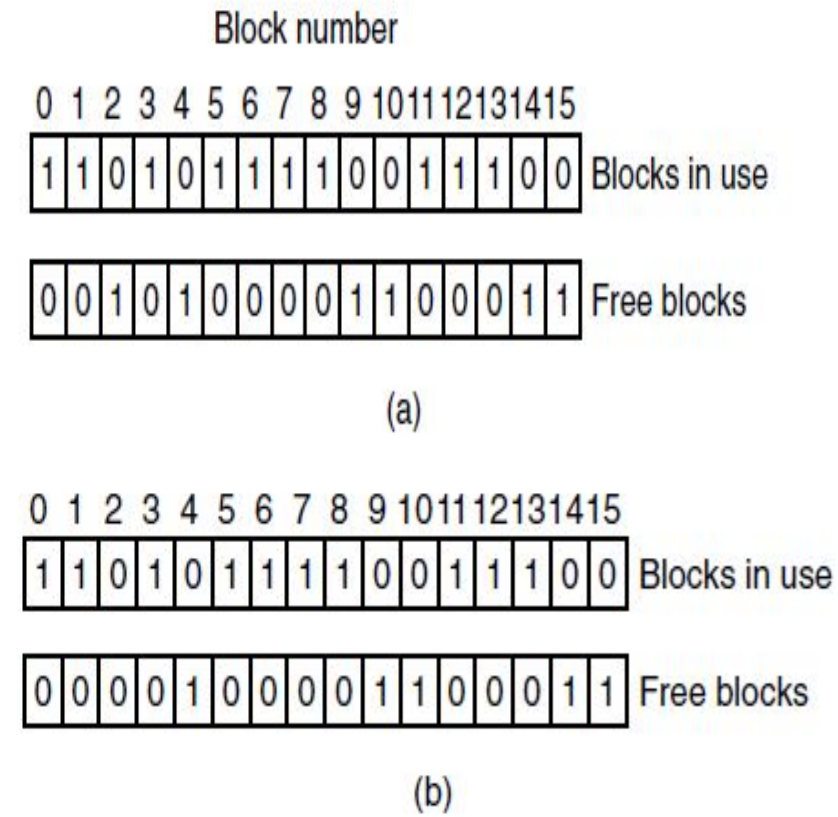
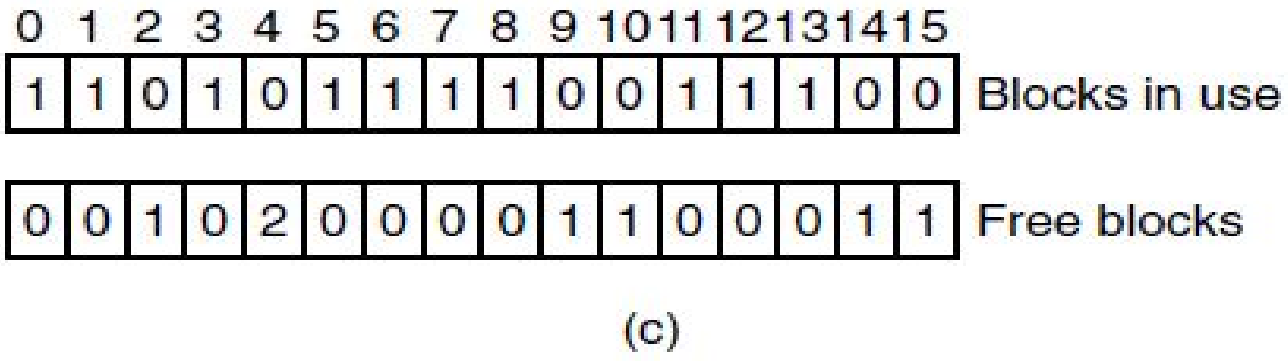


Figure 4-27. File system states. (a) Consistent. (b) Missing block.

File System Consistency (5)

Another situation that might occur is that of **Fig. 4-27(c)**. 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.



(c)

File System Consistency (6)

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, leading to a situation in which the same block is 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 (7)

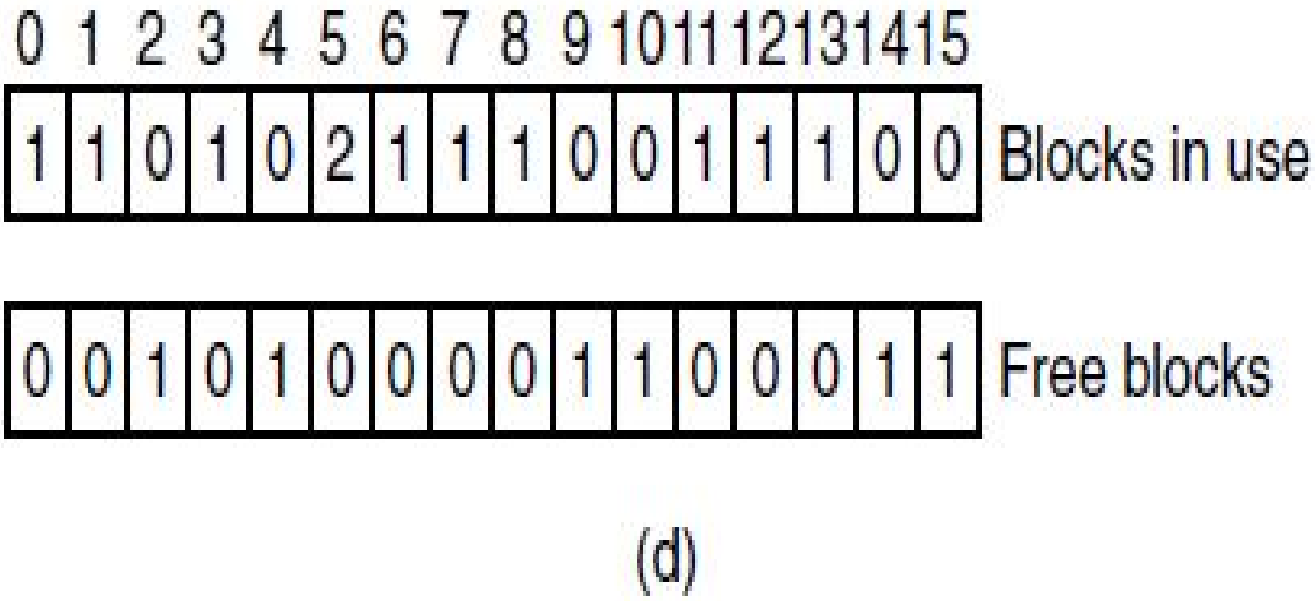


Figure 4-27. File system states.(d) Duplicate data block.

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, next slide).

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)

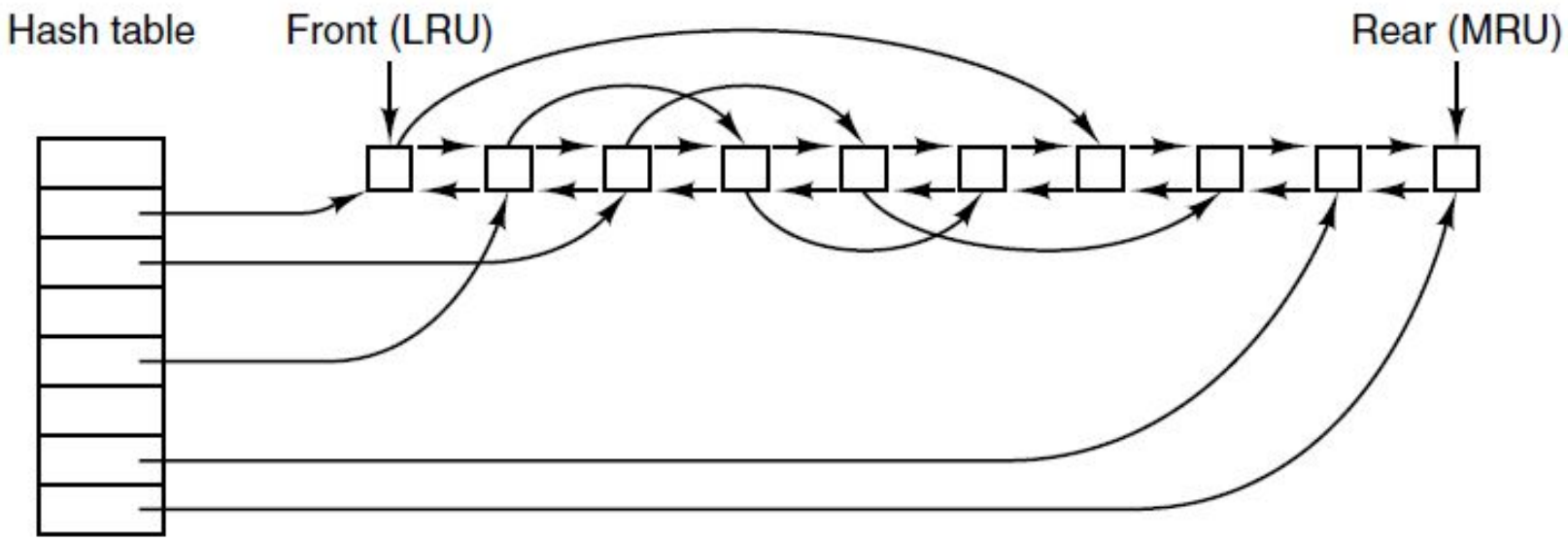


Figure 4-28. The buffer cache data structures.

Reducing Disk Arm Motion

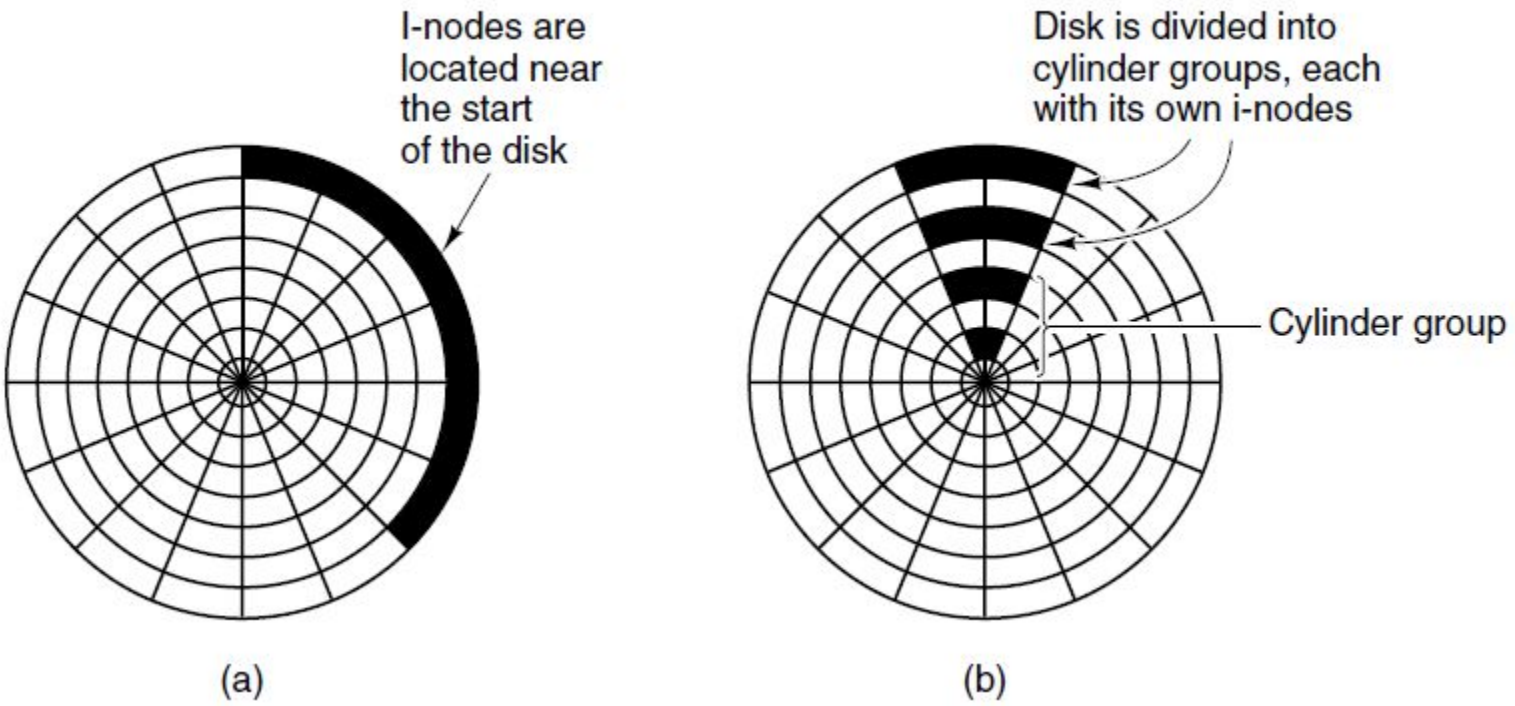


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

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)

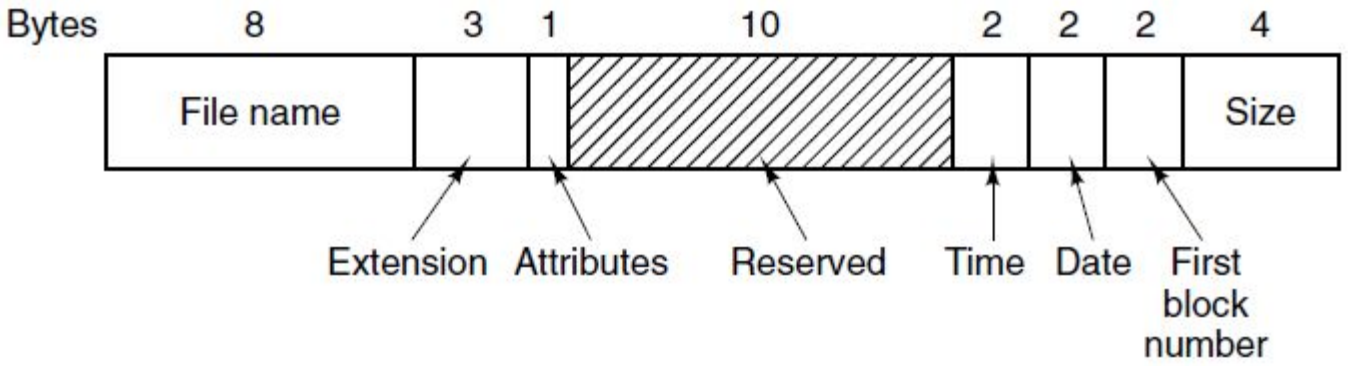


Figure 4-30. The MS-DOS directory entry.

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

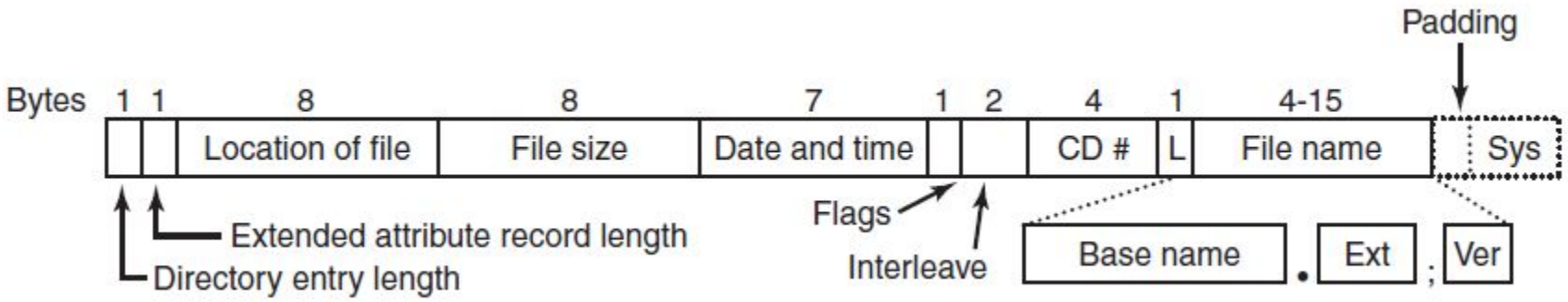


Figure 4-35. The ISO 9660 directory entry.

Rock Ridge Extensions

1. PX - POSIX attributes.
2. PN - Major and minor device numbers.
3. SL - Symbolic link.
5. NM - Alternative name.
6. CL - Child location.
7. PL - Parent location.
8. RE - Relocation.
9. TF - Time stamps.

Joliet Extensions

The major extensions provided by Joliet are:

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

End

Chapter 4