CSCI 360
Introduction to Operating Systems

File Systems

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Outline

- File Abstraction
- File Storage
- File Concepts
- Directory Concepts
- Disk Partitions and File System Layout
- Disk Block Allocation
- Free Disk Block Management
- File System Performance

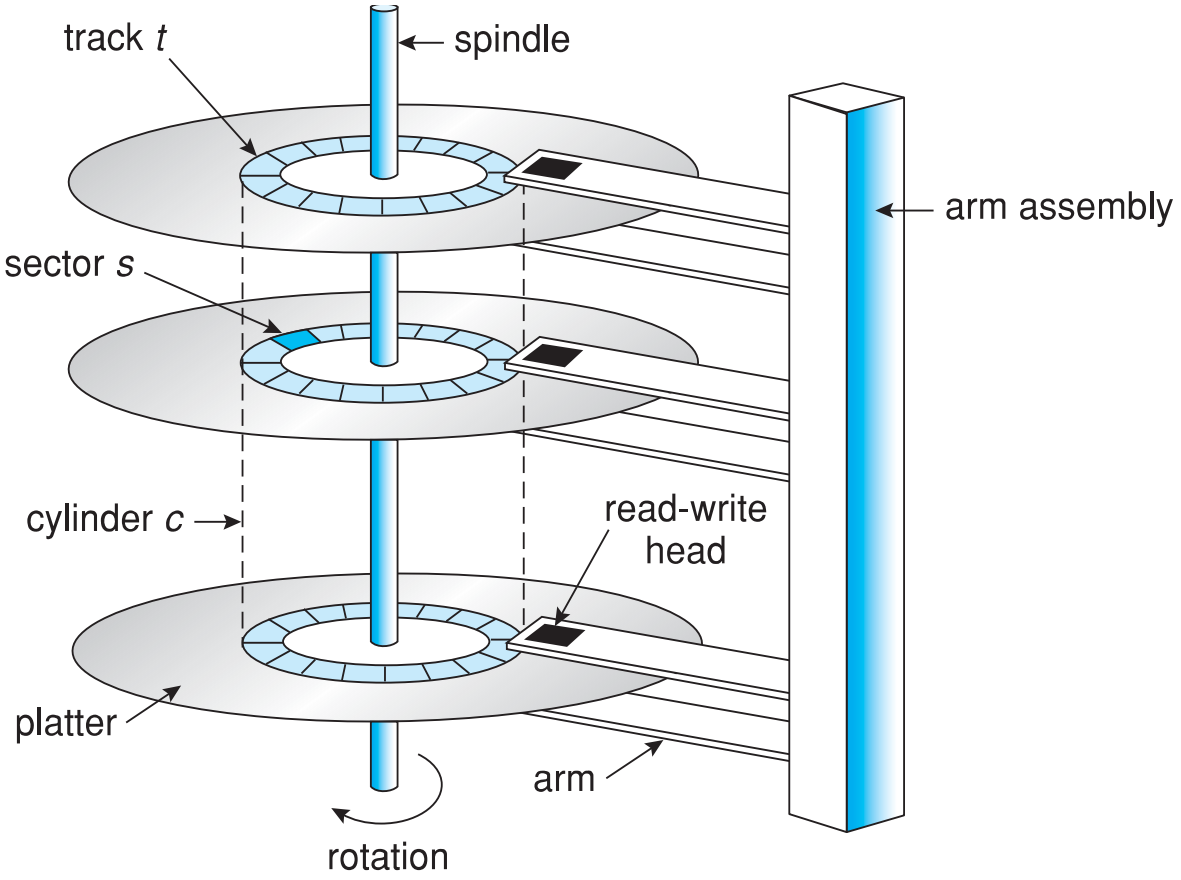
File Abstraction

File abstraction is resulted from the essential requirements for long-term information storage:

1. System needs to store a very large amount of information.
2. Stored information must survive termination of process that was using it.
3. If approved, multiple processes must be able to access the stored information concurrently.

File Storage: Hard Disk

System stores a large amount of information as a file on a hard disk

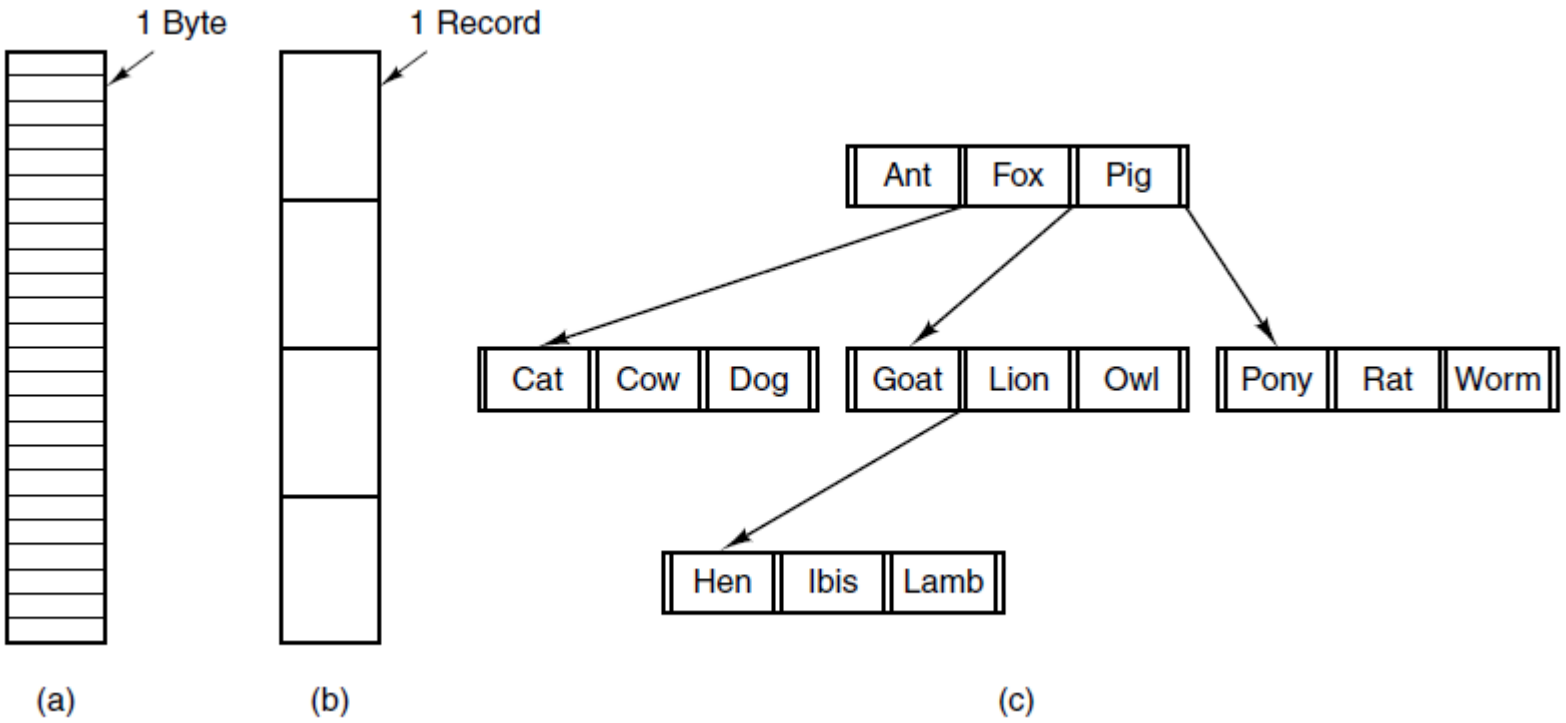


File Abstraction

- A disk is a linear sequence of fixed-size sectors or blocks and supporting two operations:
    1. Read block *k*.
    2. Write block *k*
- Information of a file may need one or more disk blocks.
- System allocates free disk blocks to a file and keeps track of this allocation.

File Structure

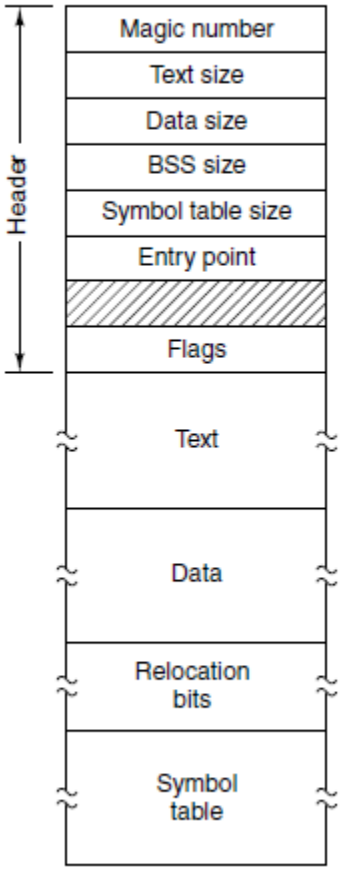
File information may be structured in different ways inside a file



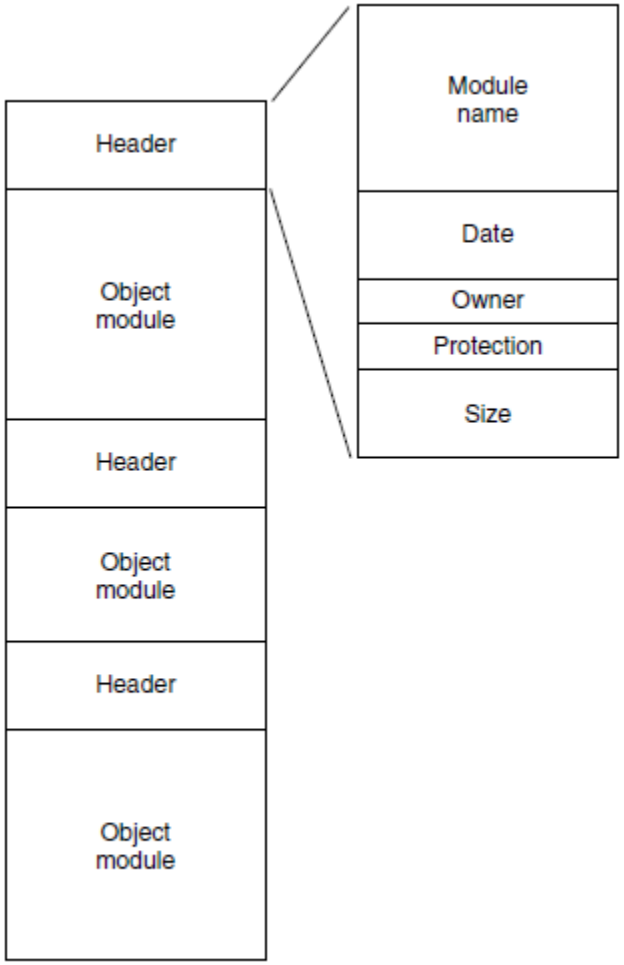
(a) Byte sequence. (b) Record sequence. (c) Tree.

File Types

ASCII File



(a)



(b)

BINARY Files: (a) an executable file. (b) an archive

File Abstraction: Name

- Files are identified by their names.
- Each file has a unique **file name**.
- File names also have **file extensions** that indicate their types.

File Abstraction: Name Extension

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

Some typical file extensions.

File Abstraction: Attributes

- System also maintains many metadata or **attributes** about each file that are helpful to manage all the files in the system.

File Abstraction: 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

Some possible file attributes.

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
12. Link
13. Unlink

Example Program Using File System Calls

```
/* 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 */
```

A simple program to copy a file.

Example Program Using File System Calls

```
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 */
```

A simple program to copy a file.

Example Program Using File System Calls

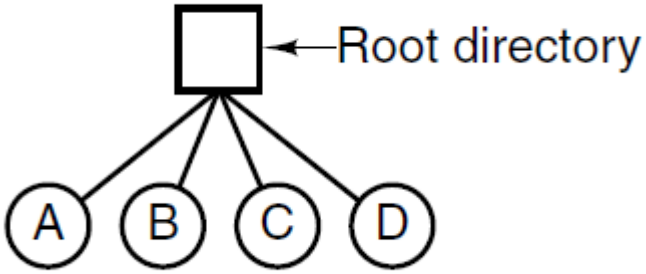
```
        /* Copy loop */
        while (TRUE) {
                rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                         /* if end of file or error, exit loop */
                wt_count = write(out_fd, buffer, rd_count); /* write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

A simple program to copy a file.

Directory Abstraction

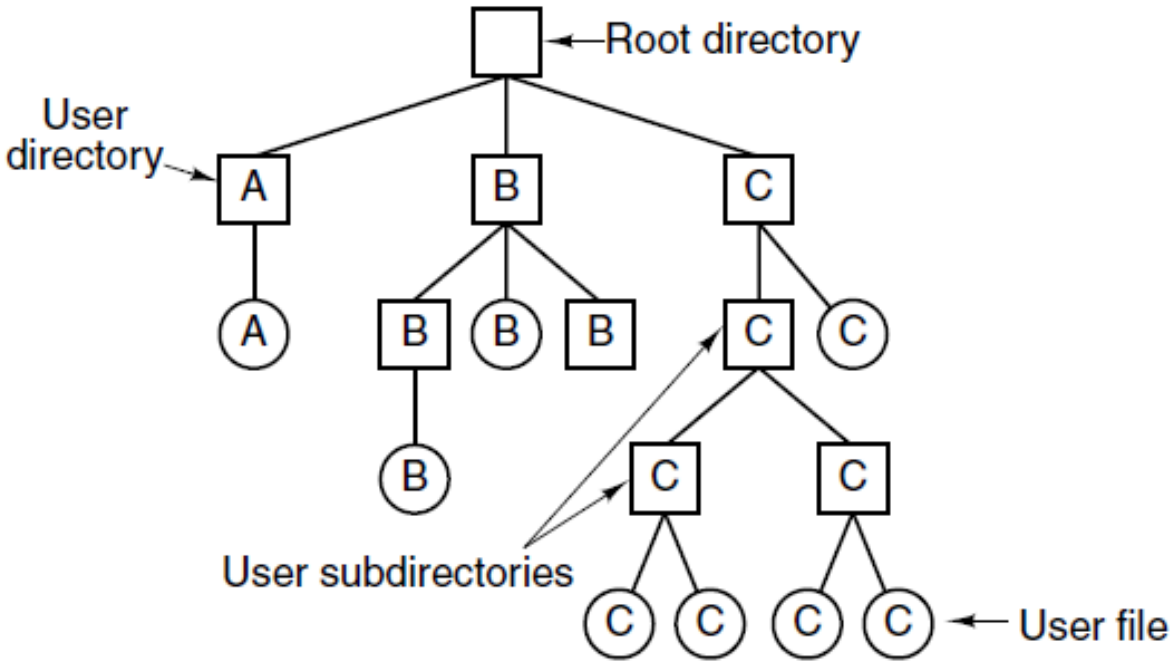
- System uses **directories** or **folders** to keep track of hundreds of files.
- Files are stored inside a directory.
- A directory may contain one or more **subdirectories**.
- Each **directory name** uniquely identifies a directory or subdirectory.

Single-Level Directory Systems



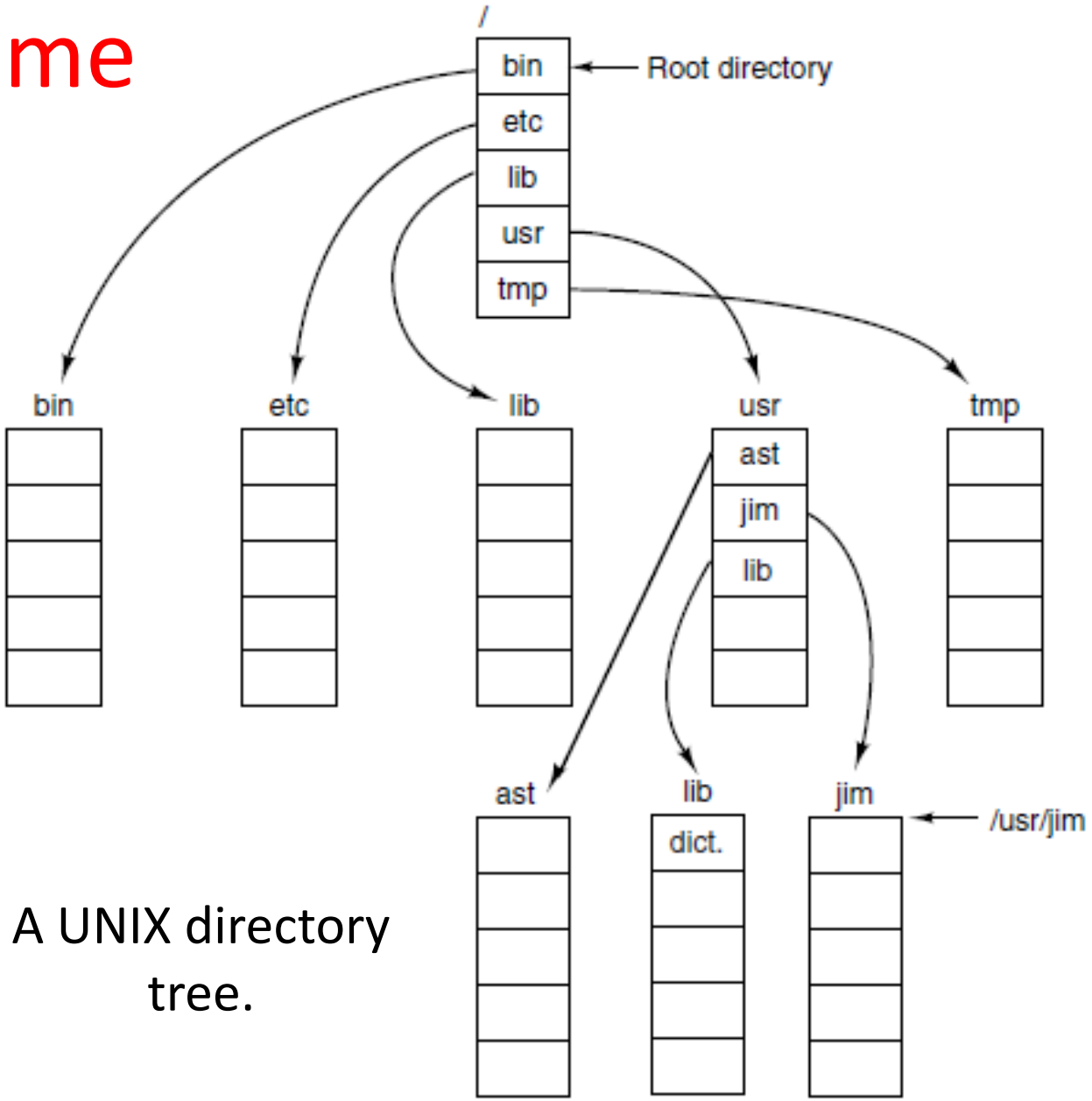
A single-level directory system containing four files.

Hierarchical Directory Systems



A hierarchical directory system.

Path Name

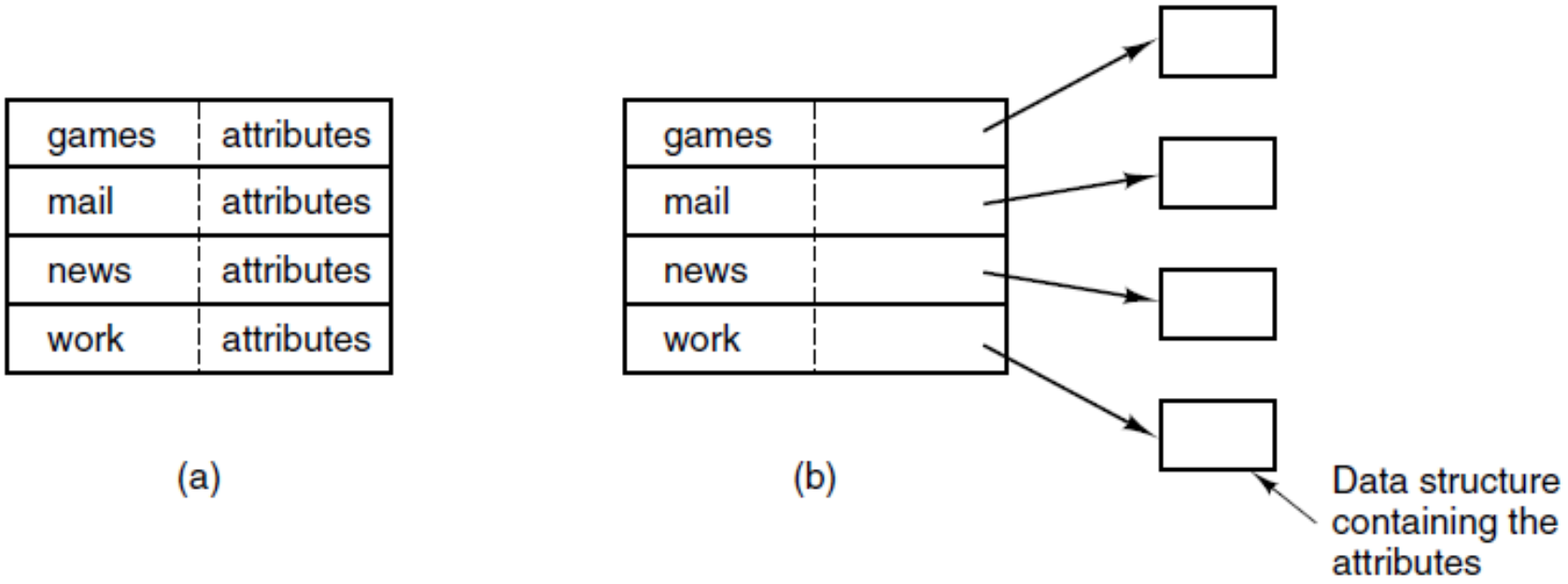


A UNIX directory tree.

Directory Operations

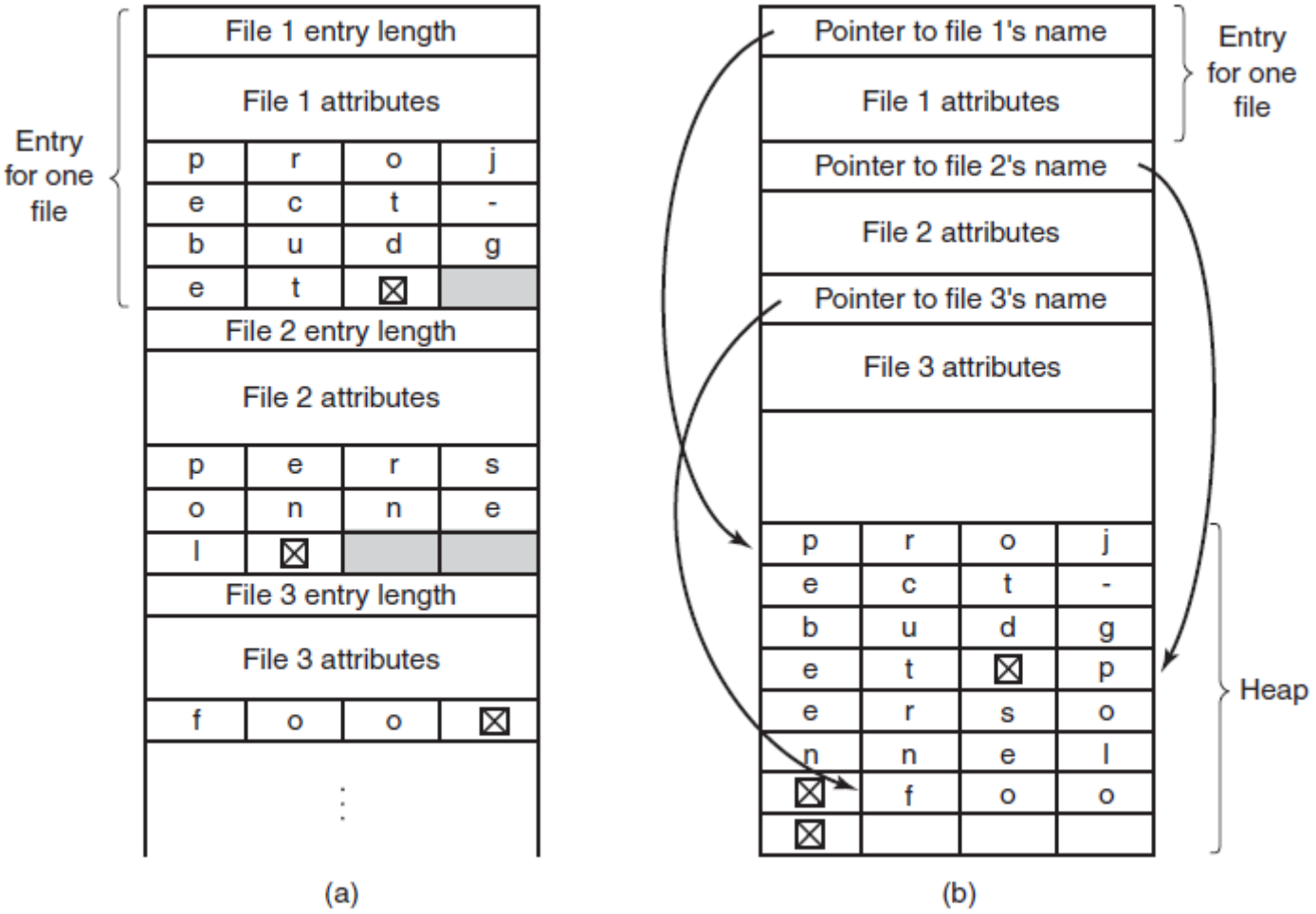
1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename

Directory Entries



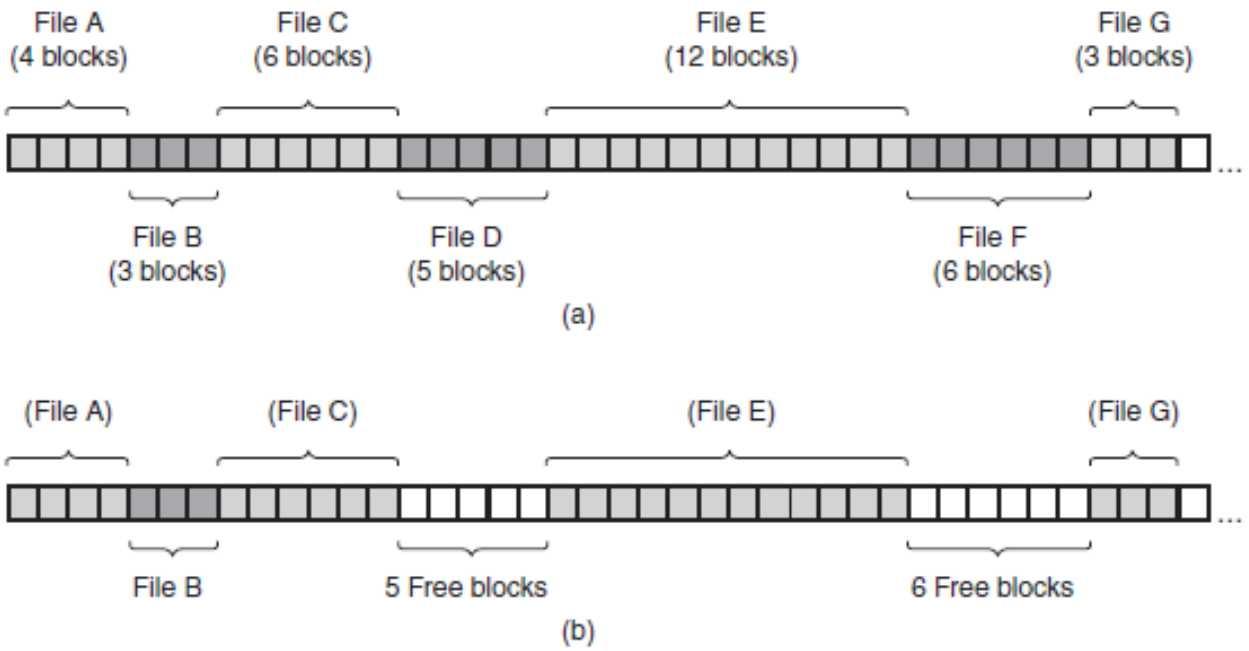
(a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each
entry just refers to an i-node.

Directory Entries (2)



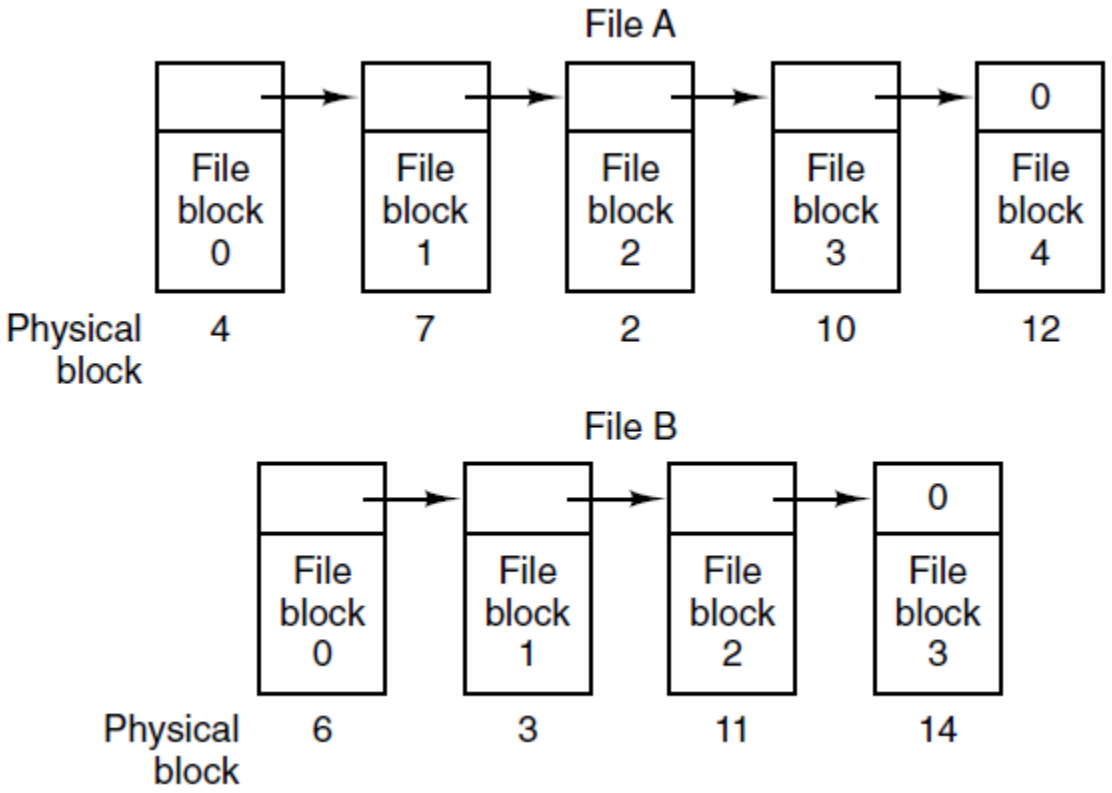
Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Disk Block Allocation: Contiguous



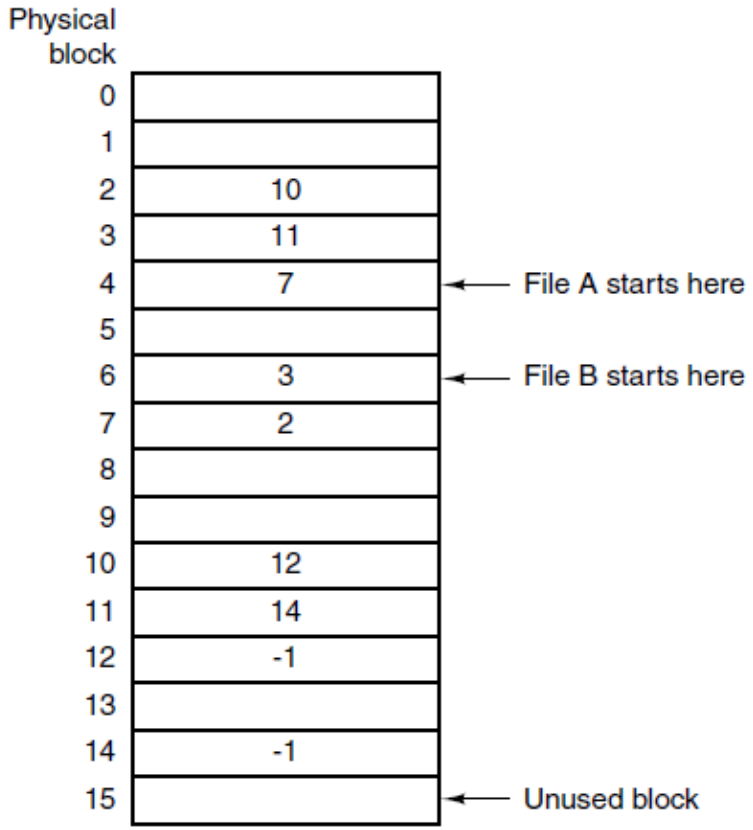
(a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Disk Block Allocation: Noncontiguous (Linked List)



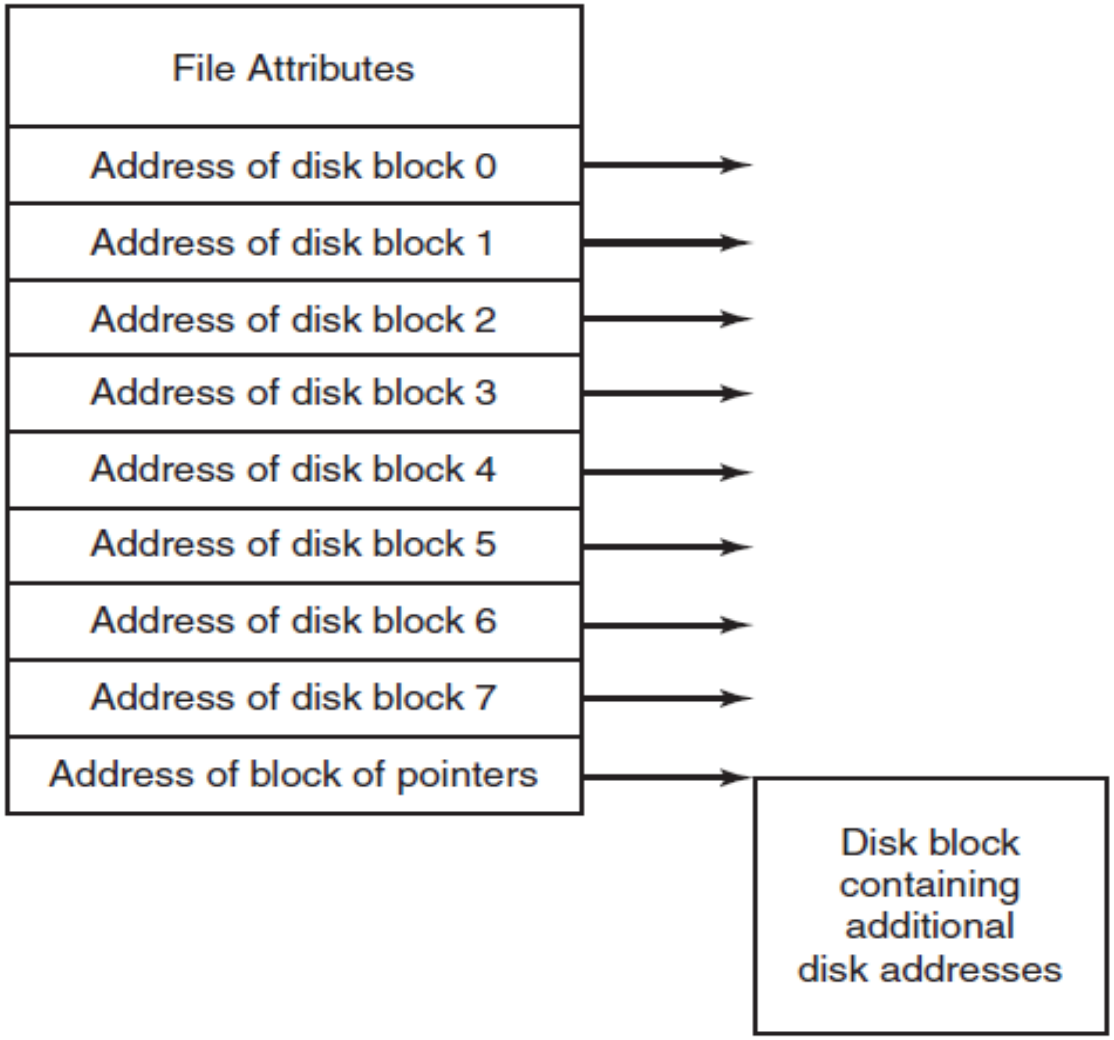
Storing a file as a linked list of disk blocks.

Allocation Table (Linked List) in Memory

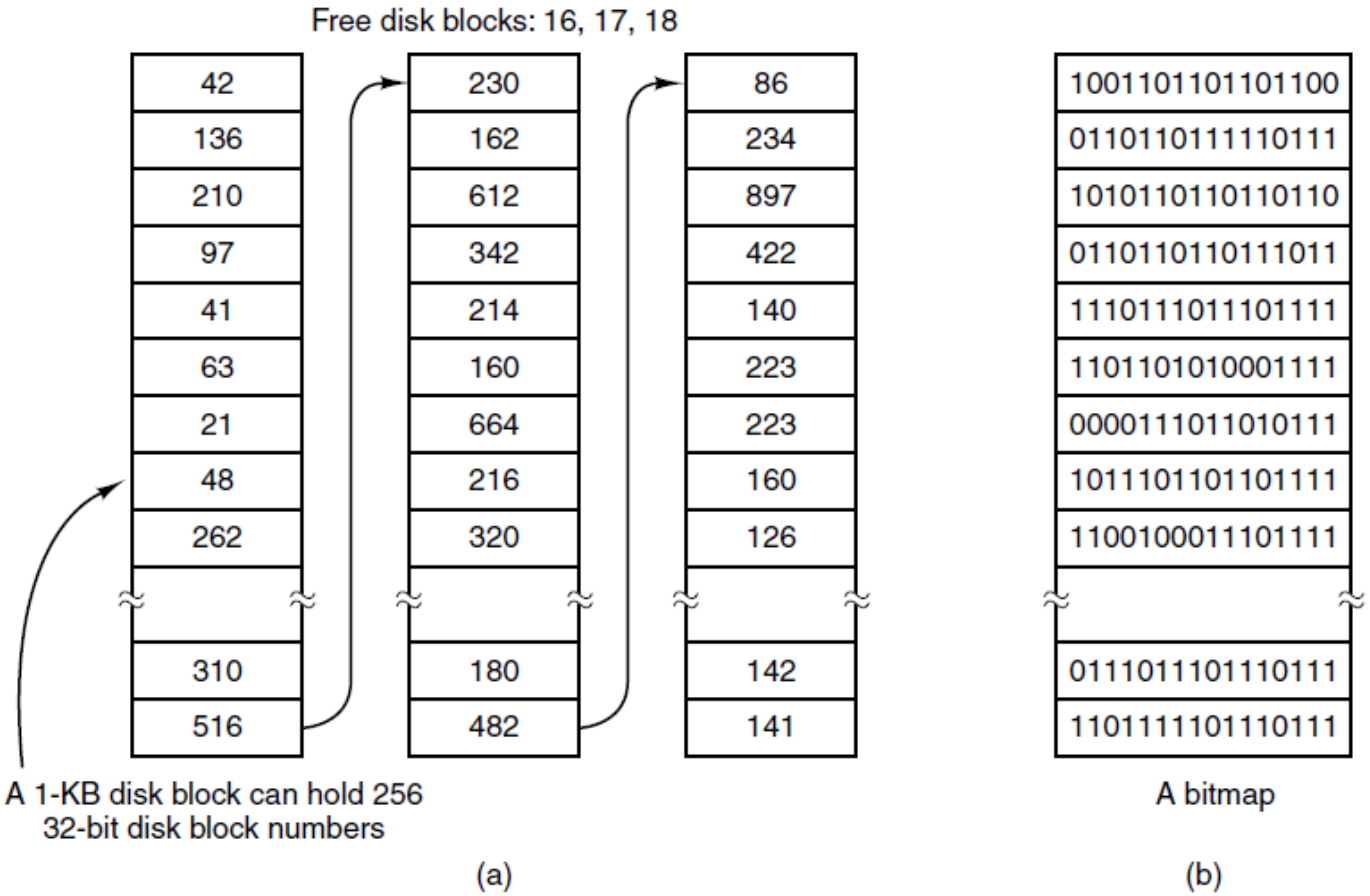


Linked list allocation using a file allocation table in main memory.

Allocation Table (List) in I-node



Keeping Track of Free Blocks



(a) Storing the free list on a linked list. (b) A bitmap.

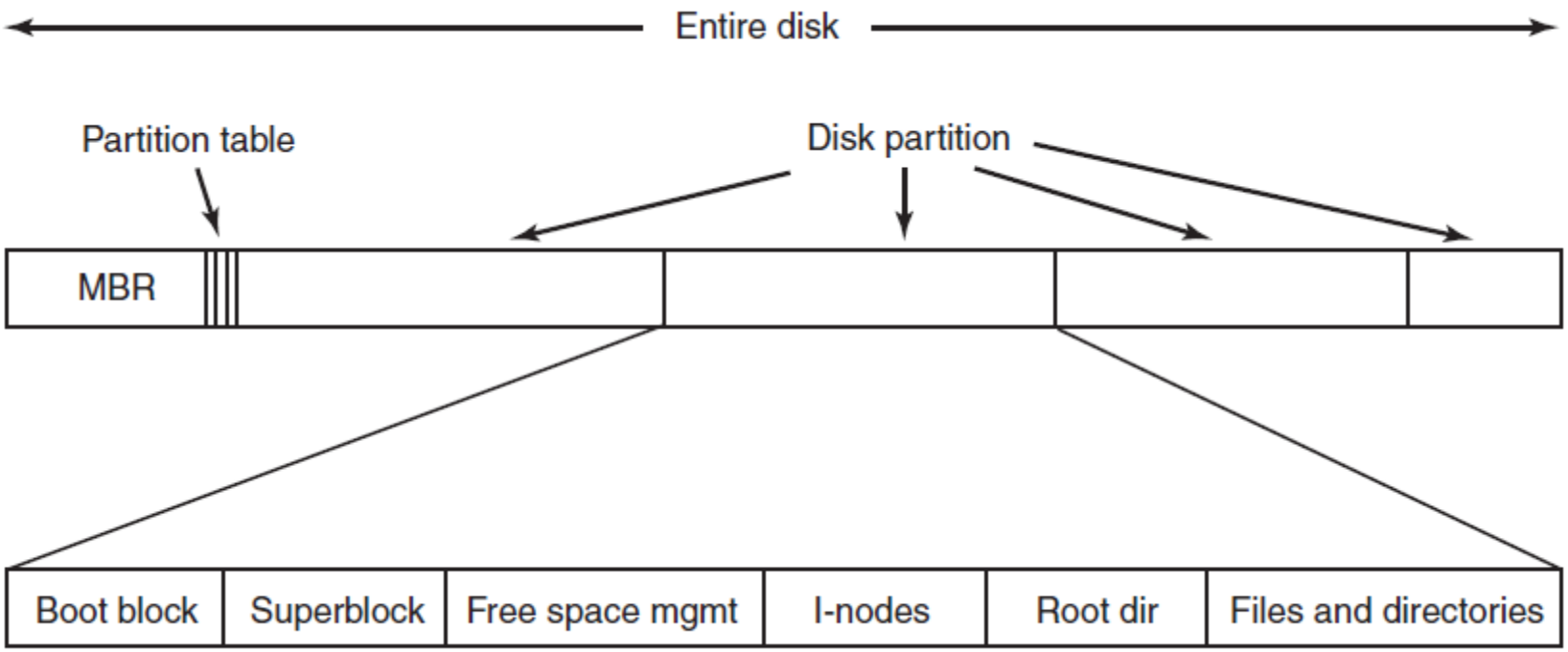
File System Disk Layout

- First disk block is the **Master Boot Block** (MBR), which contains the code that is executed when the computer starts.
- A **disk** is partitioned into several **partitions** to hold multiple operating systems on the same disk; one in each partition and only one partition is active.
- A **partition table** is kept after MBR on the disk that keeps track of the **active partition** and the starting addresses of all the partitions.

File System Disk Layout

- **MBR code** reads the partition table and locates the active partition, then reads the first block, called **boot block,** of the active partition and executes it.
- **Boot block** code loads the operating system, including filesystem, from the partition into the memory.
- **Filesystem** information consists of ***superblock*** (filesystem's metadata, e.g., filesystem type, number of blocks etc.), ***free-block*** information, ***allocated-block*** information, ***root directory*** information, and the ***other directories and files*** information.

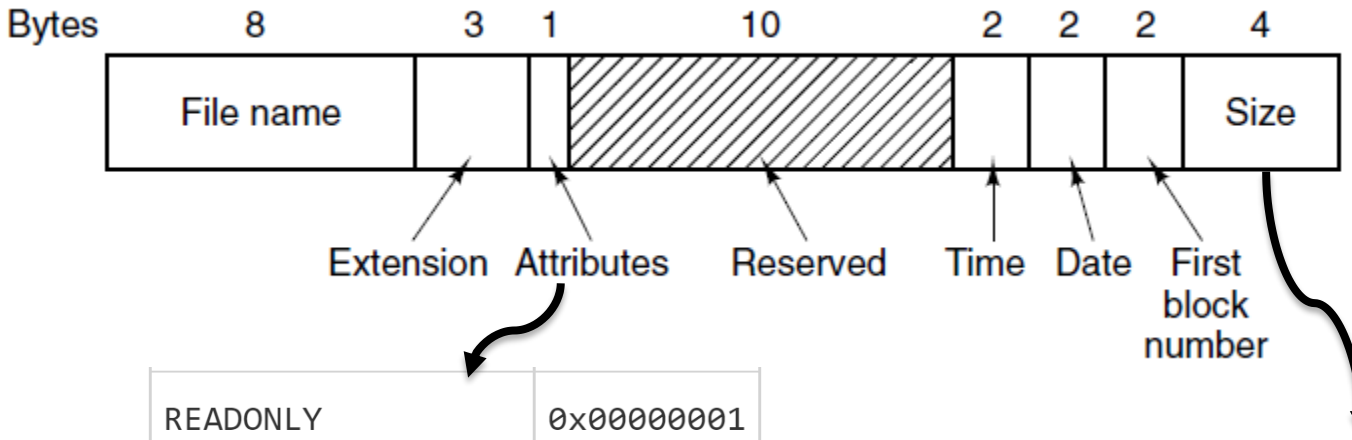
File System Disk Layout



A possible file system layout.

The MS-DOS File System

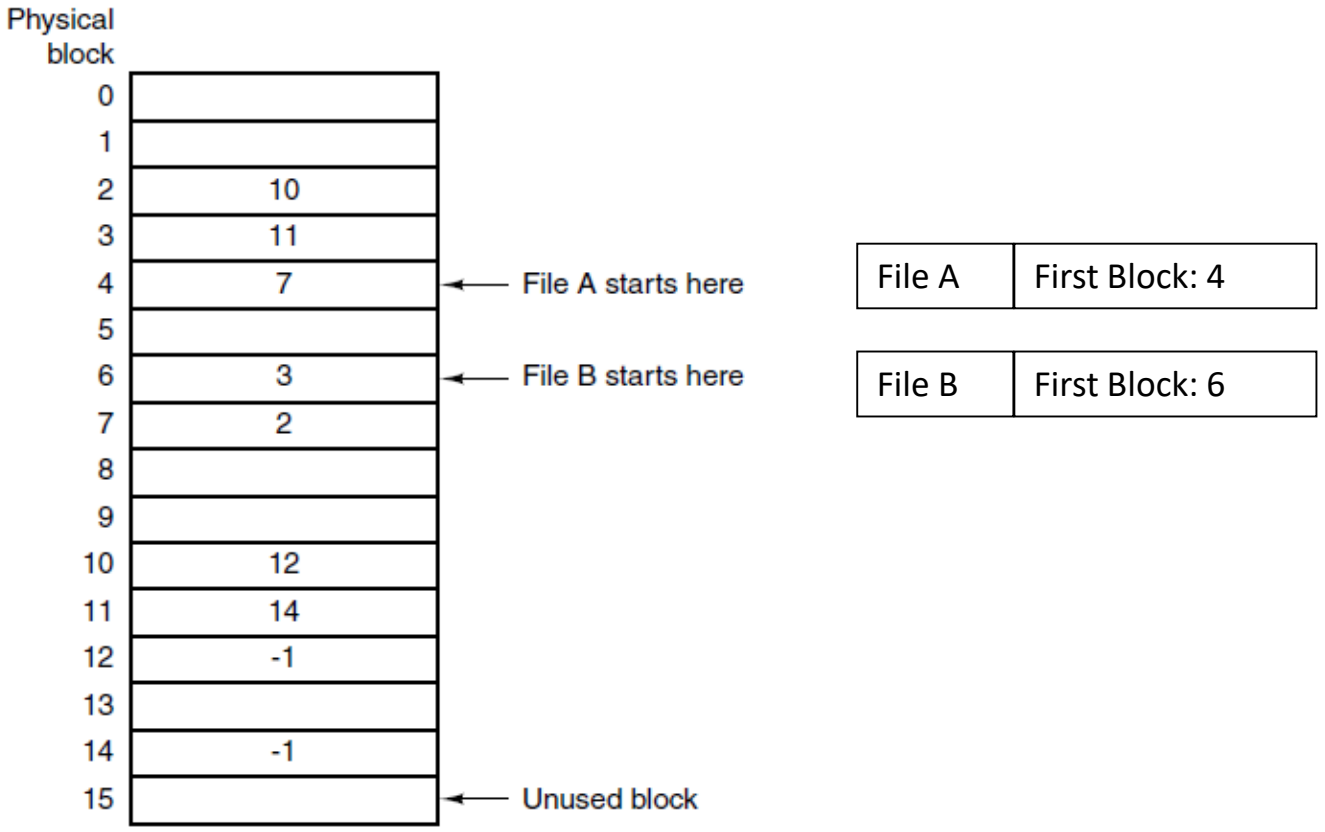
The MS-DOS directory entry.



READONLY	0x00000001
HIDDEN	0x00000002
SYSTEM	0x00000004
	0x00000008
DIRECTORY	0x00000010
ARCHIVE	0x00000020

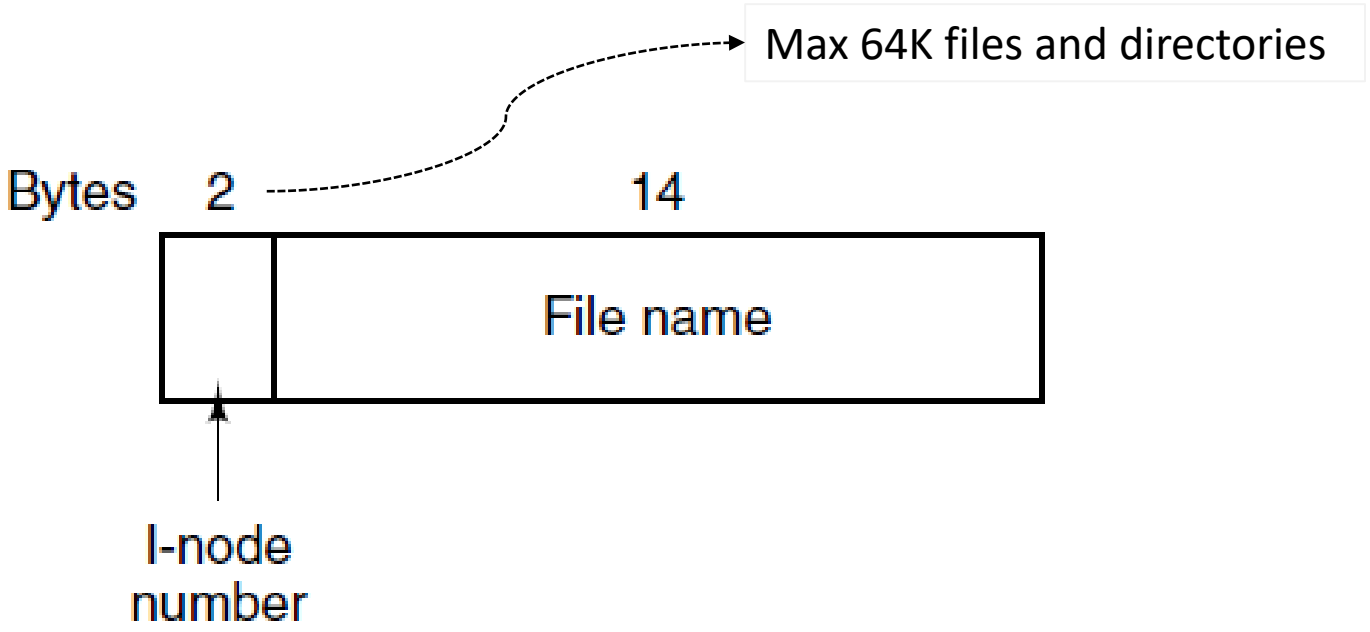
File Size
4GB in theory
2GB in practice

The MS-DOS File System



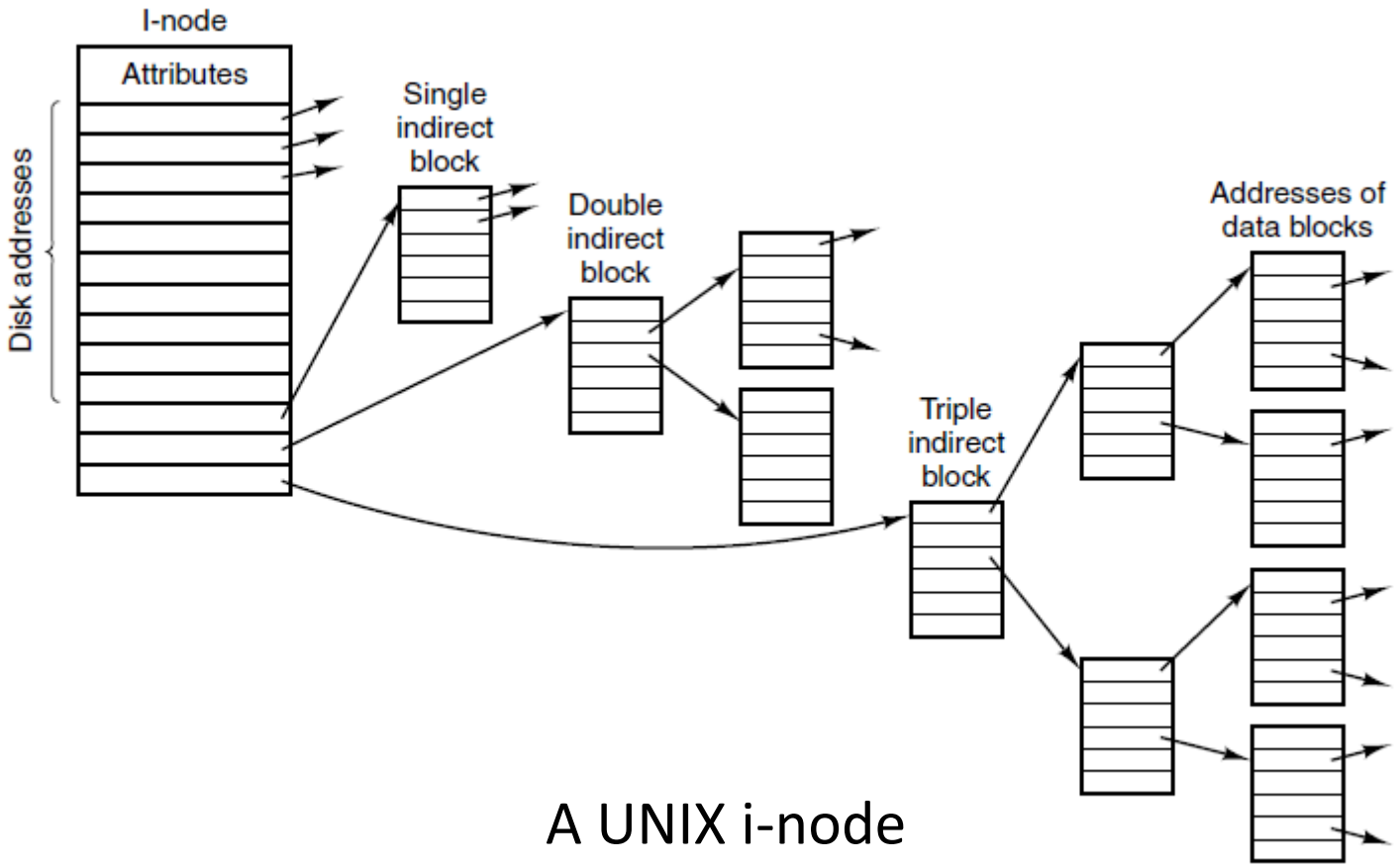
File Allocation Table (FAT)

The UNIX V7 File System



A UNIX V7 directory entry.

The UNIX V7 File System



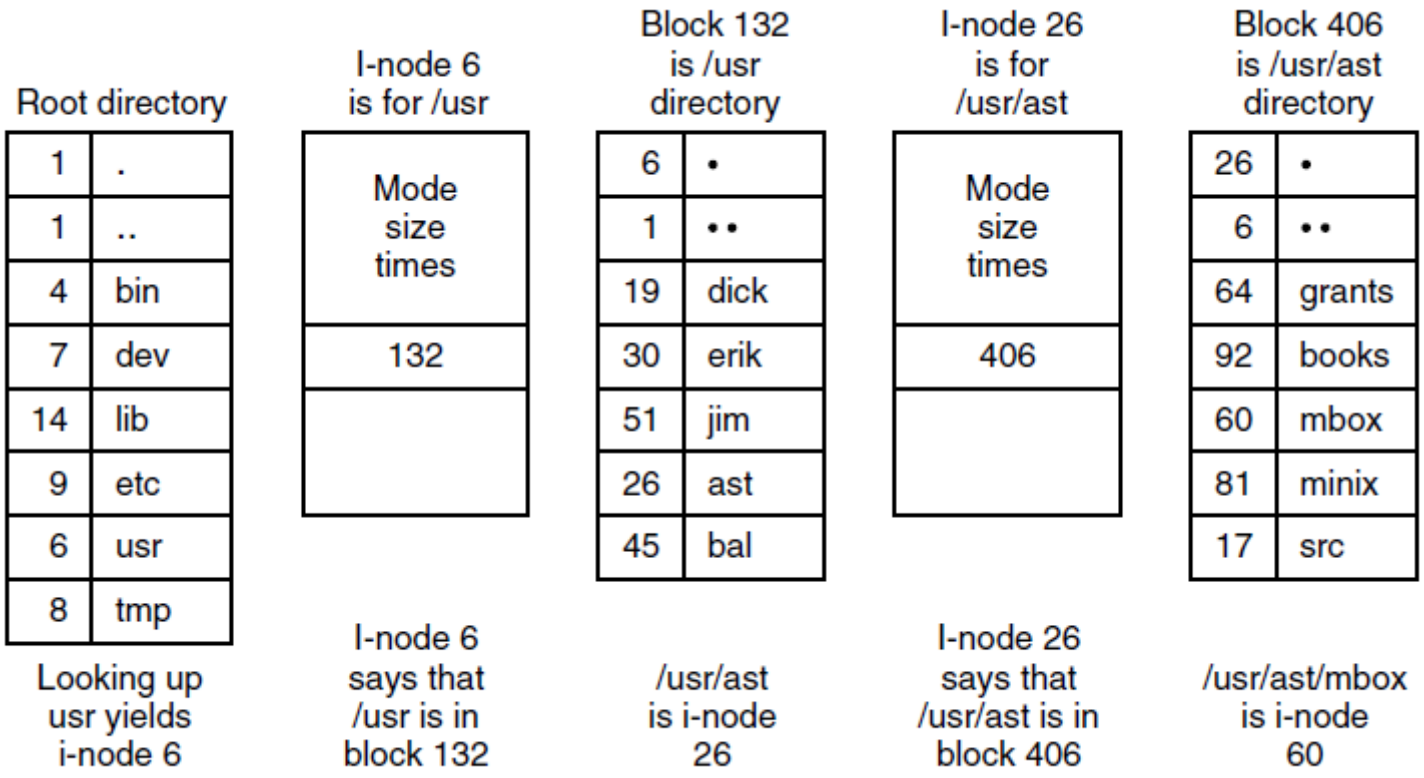
A UNIX i-node

The UNIX V7 File System

A UNIX i-node Attributes:

- File Size
- Creation Time
- Last Access Time
- Last Modified Time
- Owner
- Group
- Protection
- Link Count

The UNIX V7 File System



The steps in looking up */usr/ast/mbox*.

Unix File Systems

Steps to add a file in UNIX:

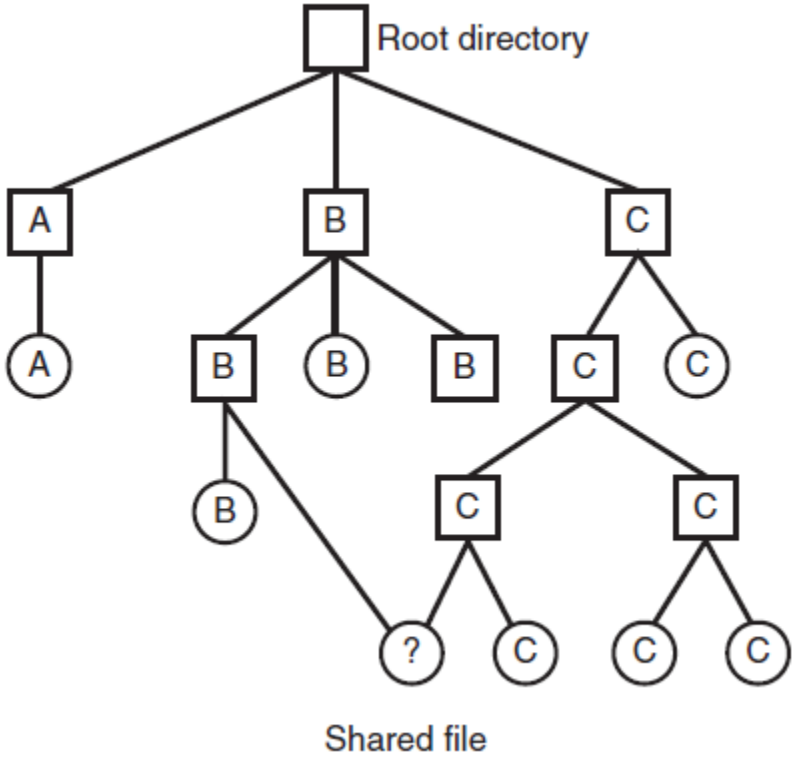
1. Get the disk blocks for the file from the pool of free disk blocks.
2. Get a free i-node for the file from the pool of free i-nodes.
3. Record the disk blocks into i-node and set the link count to 1.
4. Add a directory entry for the file in its directory.

Unix File Systems

Steps to remove a file in UNIX:

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

Unix Shared Files



File system containing a shared file.

Assume User B shares User C's file

Unix Shared Files

Steps to add shared file in UNIX:

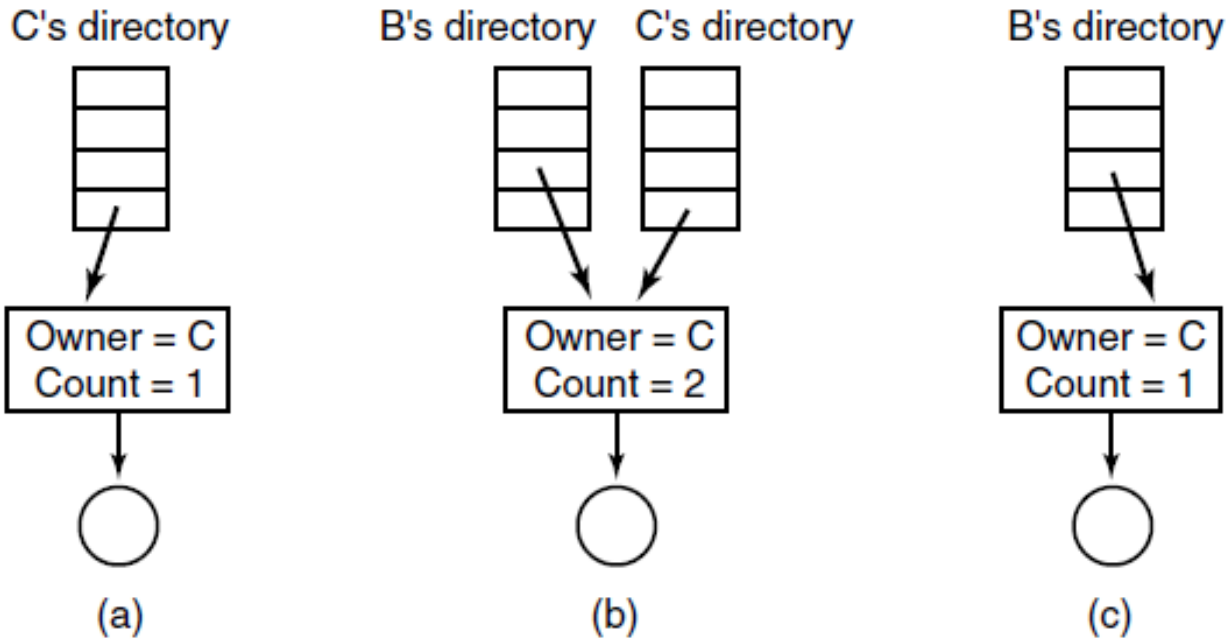
1. Get the i-node of the original file.
2. Increment the link count.
3. Add a directory entry for the shared file in its directory.

Unix Shared File

Steps to remove a shared file in UNIX:

1. Remove directory entry of the shared file from its directory.
2. Decrement the link count in i-node.
3. If link count becomes zero release i-node to the pool of free i-nodes and return all disk blocks to the pool of free disk blocks.

Unix Shared Files



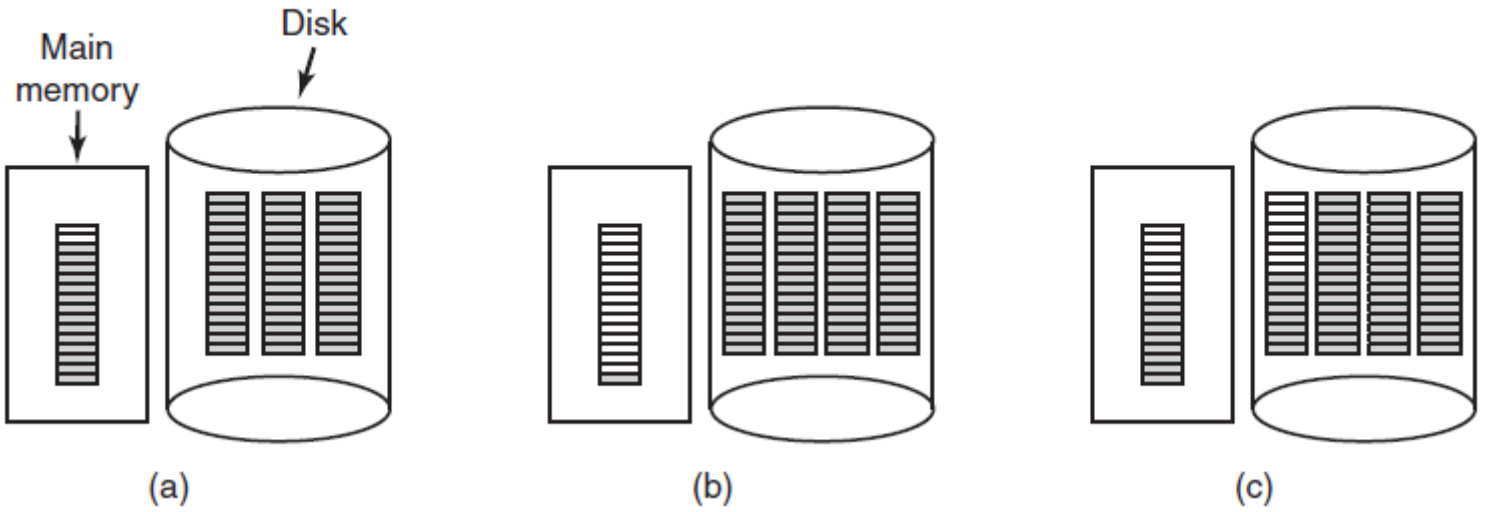
(a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Free Block List Enhancement

To enhance performance of free block list

- One block of free block list is kept in memory
- When disk blocks are needed that are taken from the in memory list and are removed from it.
- If in memory free block list becomes empty a new disk block of free block list is brought into memory.
- When disks blocks are freed that are added to the in memory list.
- If the in memory list becomes full it is written back to disk.

Free Block List Enhancement



(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.

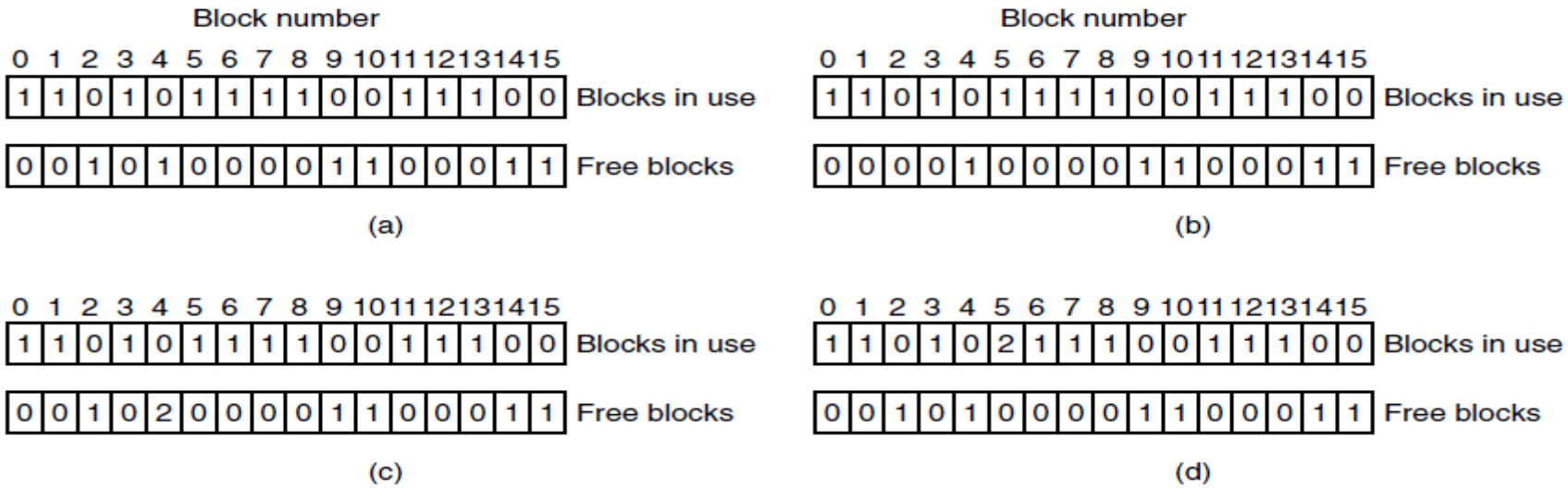
File System Consistency

Two kinds of consistency checks

- Block Consistency
- File Consistency

File System Consistency

In block consistency check, each block is accounted for how many times it is **in use by the files** and how many times it is **in free block list.**



File system states. (a) Consistent. (b) Missing block **2**. (c) Duplicate block **4** in free list. (d) Duplicate data block **5**.

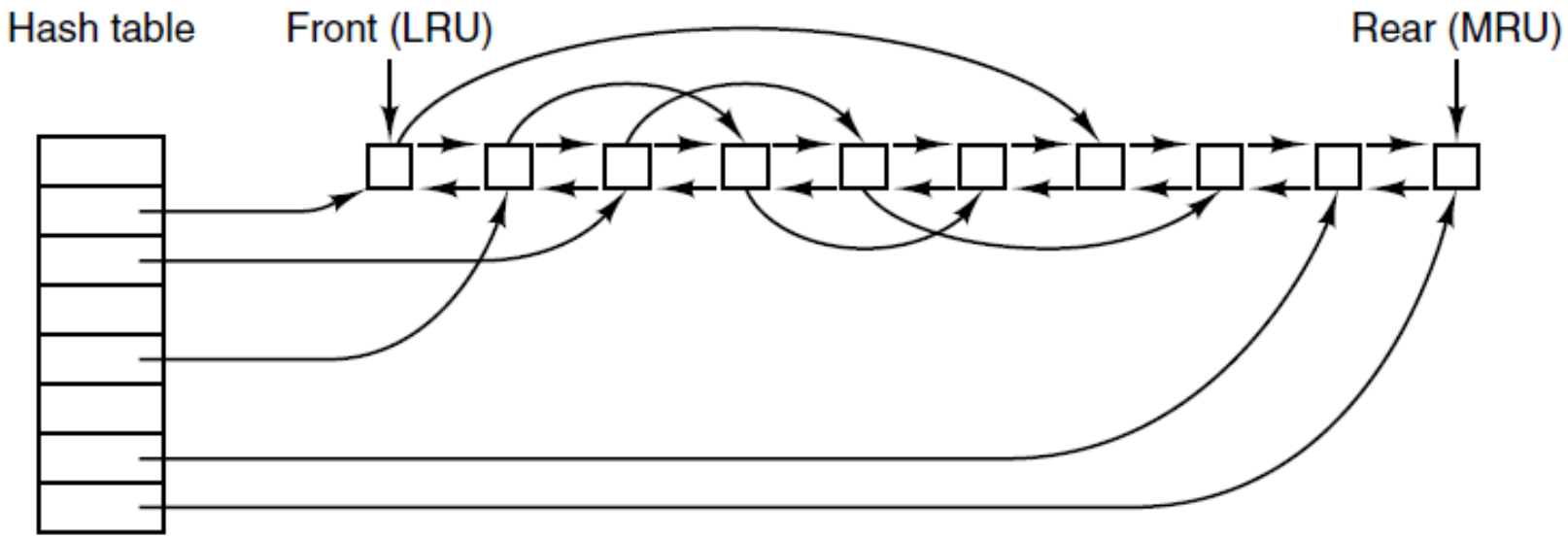
File System Consistency

- In file consistency, check all the directories starting from the root directory.
- Count the presence of each i-node in the directory entries of the whole file system.
- Compare the present count against the link count of each i-node.
- If the link count is higher or lower set it to the present count.

File System Caching

- A set of disk blocks are kept in memory in order to reduce the number of disk accesses.
- Against any read request, the cache blocks are checked first.
- If the block is in the cache, the request is handled without a disk access.
- If the block is not in the cache, it is brought into the cache first and then the request is handled from the cache.

File System Caching



The buffer cache data structures.

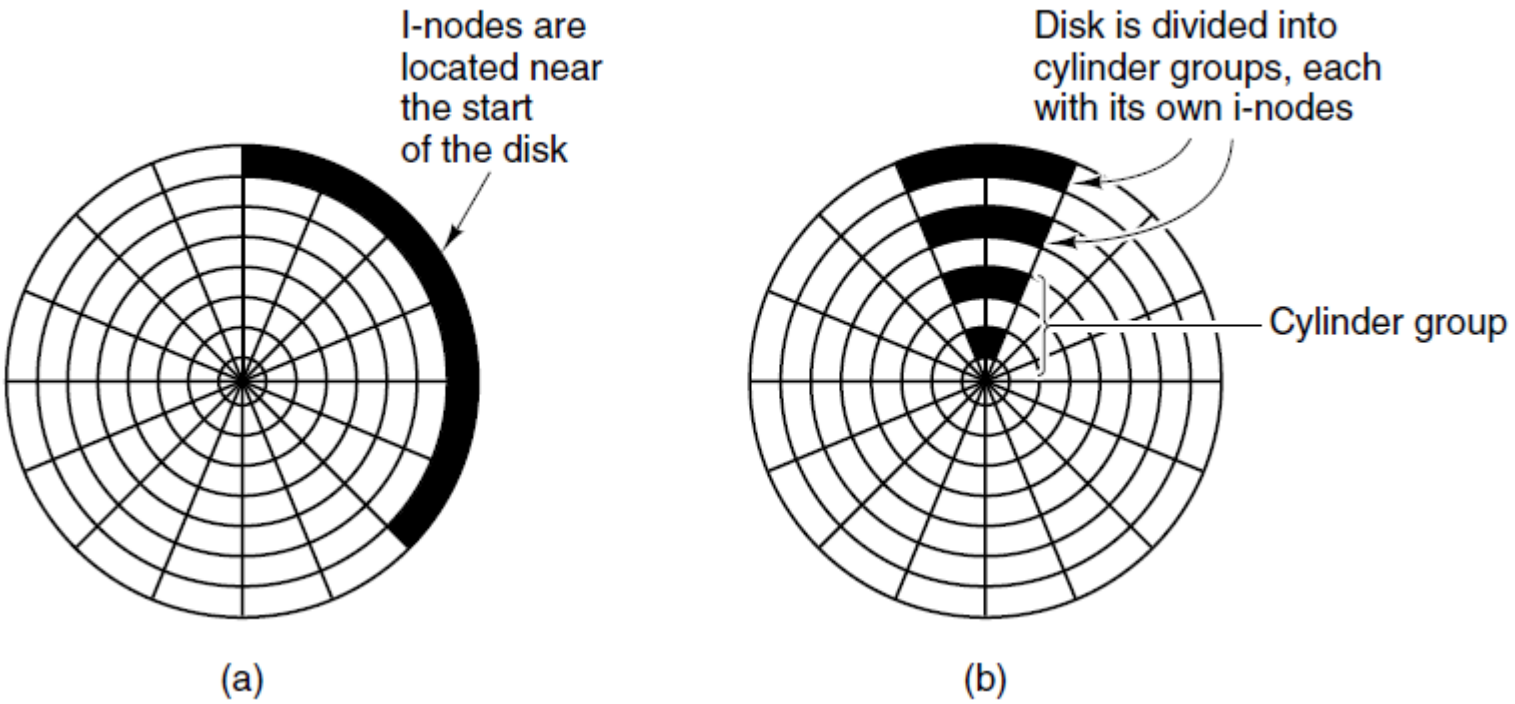
File System Caching

- Some blocks rarely referenced two times within a short interval.
- Leads to a modified LRU scheme, taking two factors into account:
  1. Is the block likely to be needed again soon?
  2. Is the block essential to the consistency of the file system?
- Blocks are classified into following classes
  - i-node
  - Indirect
  - Directory
  - Full data
  - Partially full data

File System Caching

- Block that will not be needed soon goes to the front of the list and its cache will be reclaimed soon whenever necessary.
- Block that will be needed soon goes to the rear of the list its cache will not be reclaimed soon.
- Block that is necessary for the file system consistency and have been modified should be written into the disk immediately irrespective of its position in the list.

Reducing Disk Arm Motion



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

Reducing Disk Arm Motion

When a new file is created it can choose any i-node from the free i-node list but choose the data blocks from the same cylinder group

Summary

- File Abstraction
- File Concepts
- Directory Concepts
- Disk Partitions and File System Layout
- Disk Block Allocation
- Free Disk Block Management
- File System Performance

Next

I/O Systems

- I/O Concepts
- I/O Software
- I/O Software Layers