Chapter 6

File Systems

6.1 Files
6.2 Directories
6.3 File system implementation
6.4 Example file systems

Files:
Long-term Information Storage

1. Must store large amounts of data

2. Information stored must survive the termination of the process using it

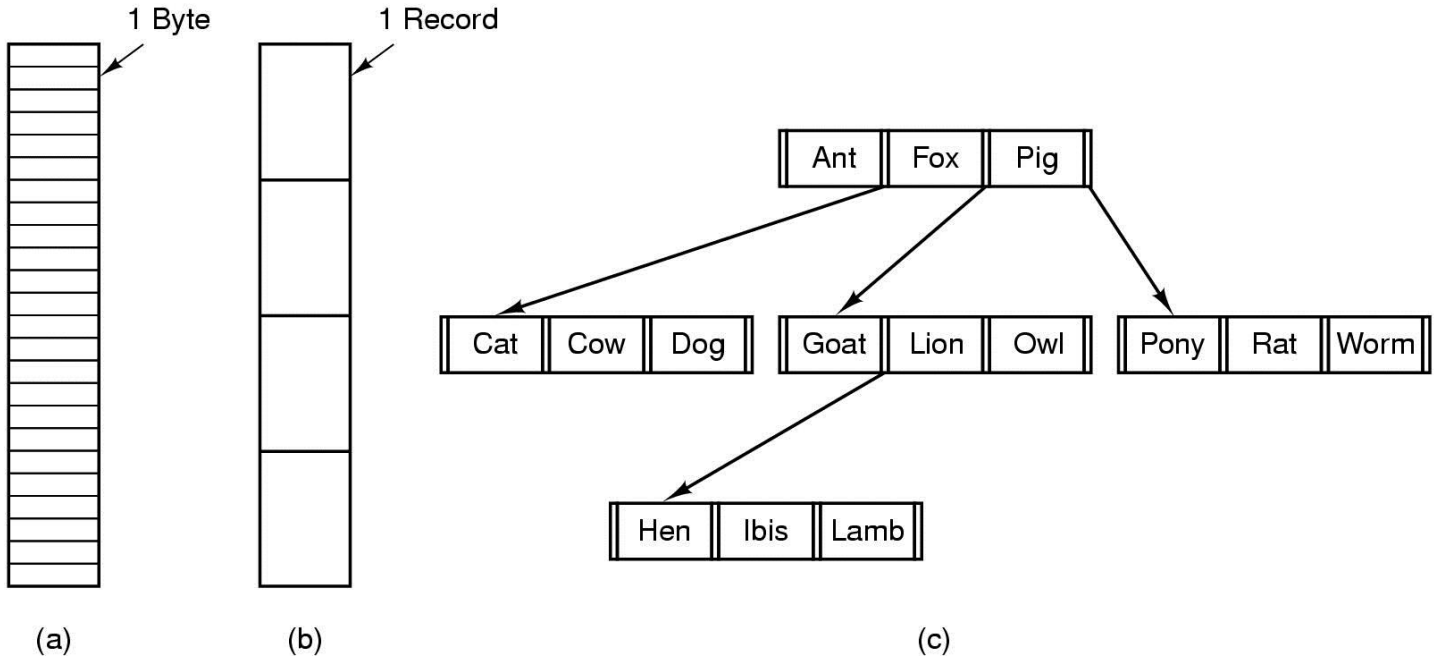
3. Multiple processes must be able to access the information concurrently

File Naming

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

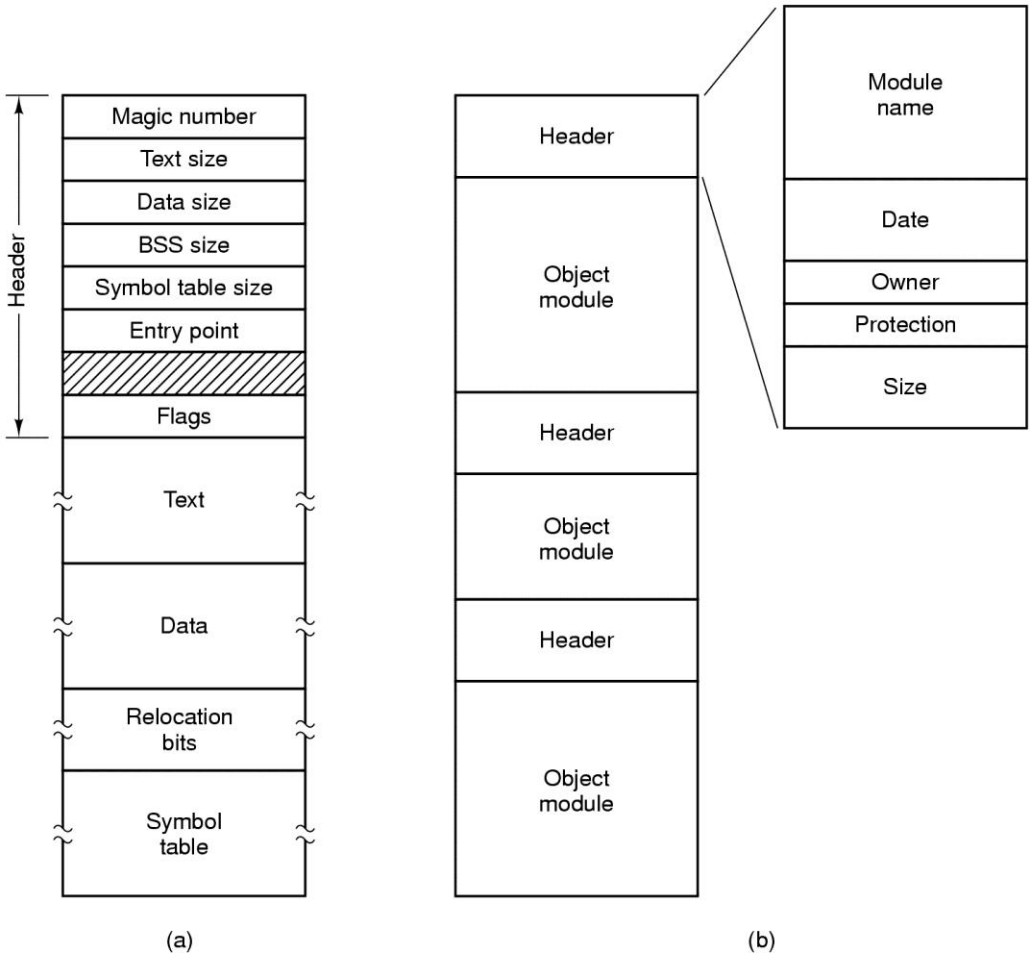
Typical file extensions.

File Structure



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

File Types



(a) An executable file (b) An archive

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

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

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

An Example Program Using File System Calls (1/2)

```
/* 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 */
```

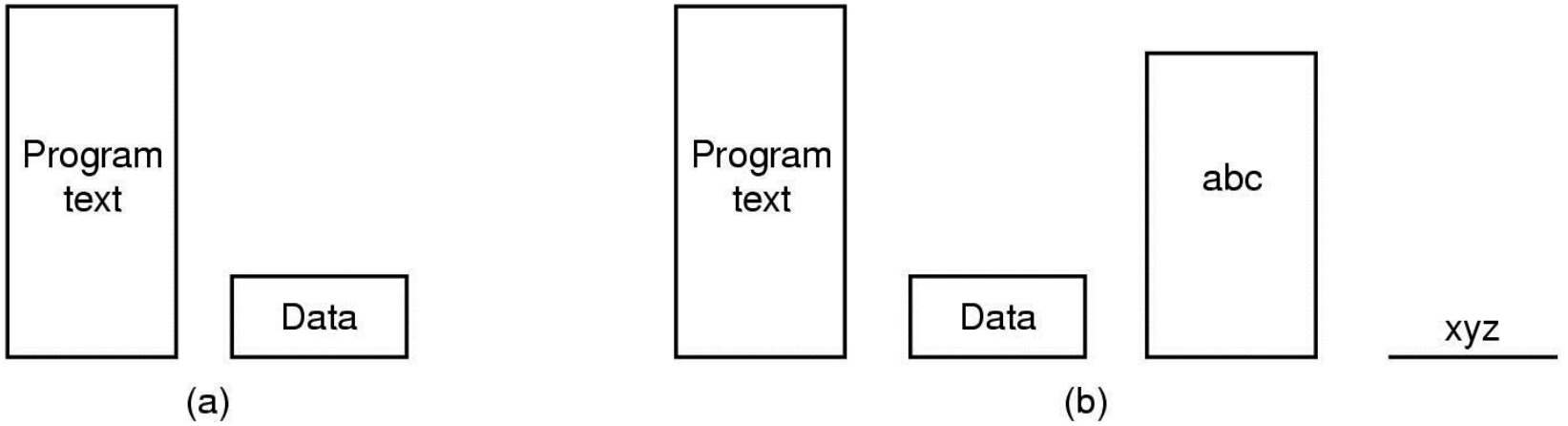
An Example Program Using File System Calls (2/2)

```
        /* 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 */
            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 */
}
```

Memory-Mapped Files



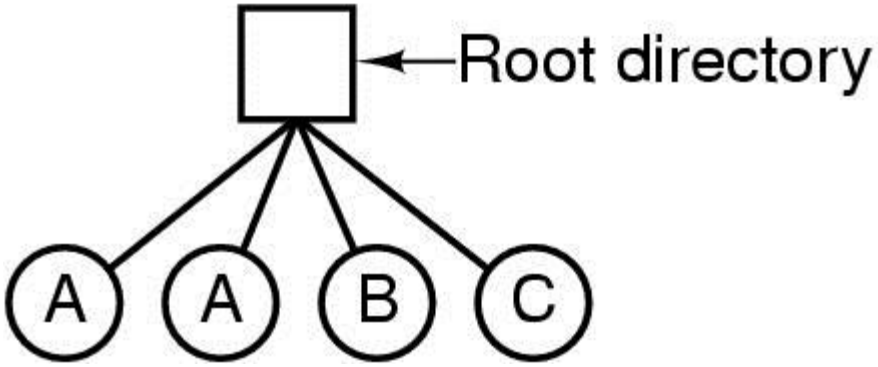
(a) Segmented process before mapping files into its address space

(b) Process after mapping

existing file *abc* into one segment

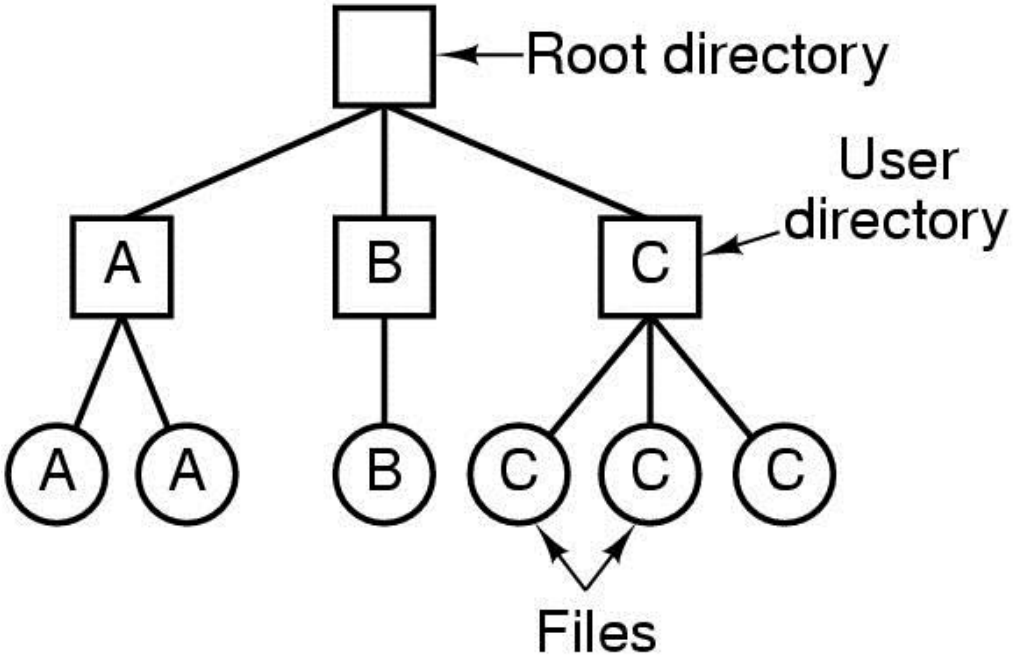
creating new segment for *xyz*

Directories
Single-Level Directory Systems



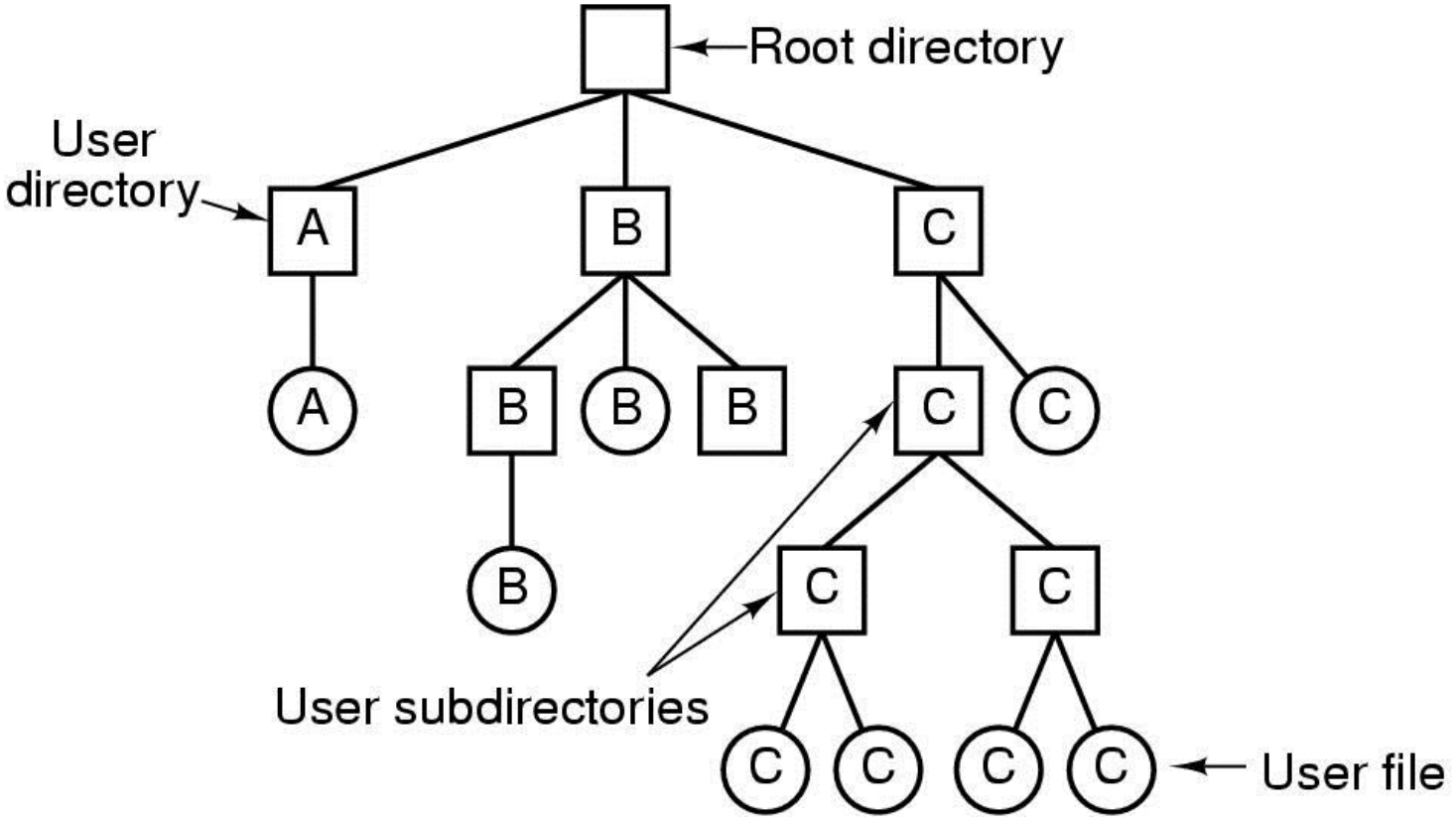
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two-level Directory Systems



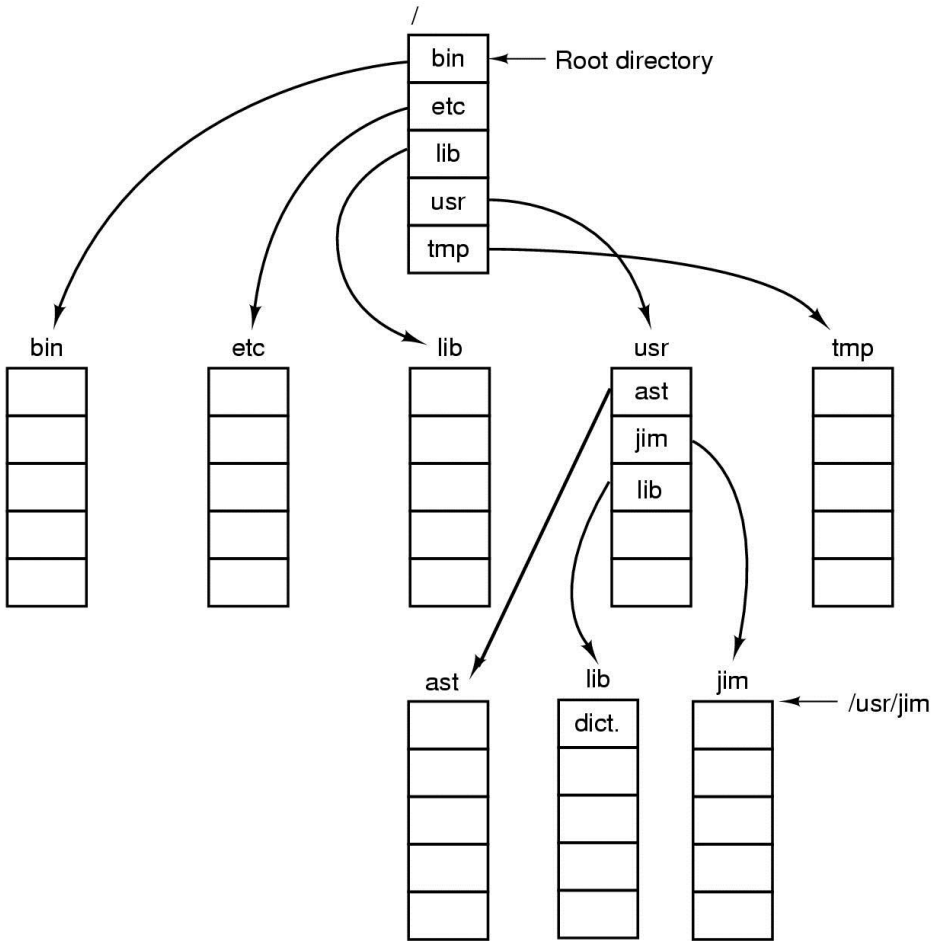
Letters indicate *owners* of the directories and files

Hierarchical Directory Systems



A hierarchical directory system

Path Names

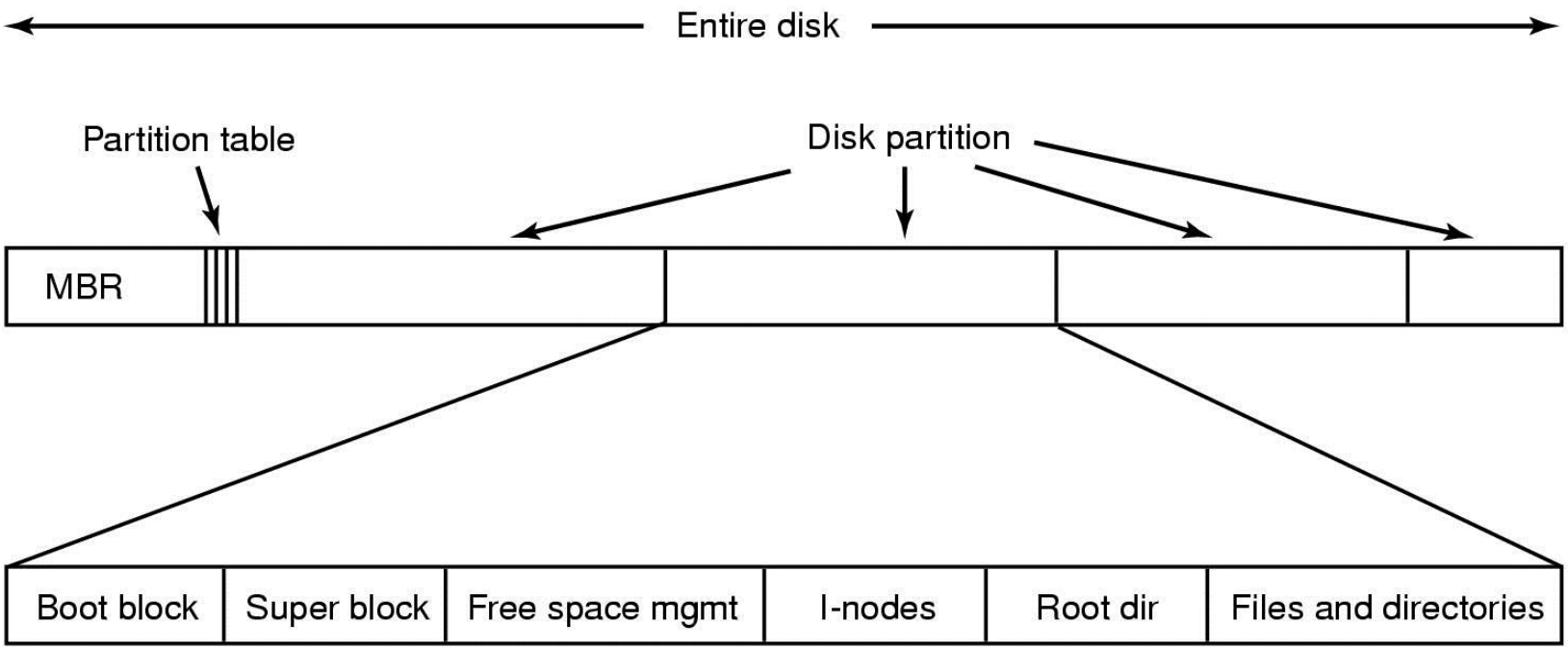


A UNIX directory tree

Directory Operations

- Create
  - Dot(.) and dotdot(..) are put automatically
- Delete
- Opendir
  - Before a dir can be read, it must be opened.
- Closedir
  - Frees up internal table space
- Readdir
  - Formerly *read* that deals w/ intern. str. of dir.
- Rename
- Link
  - Creates a link from a existing file to the path
- Unlink
  - A directory entry is removed.

File System Implementation



A possible file system layout

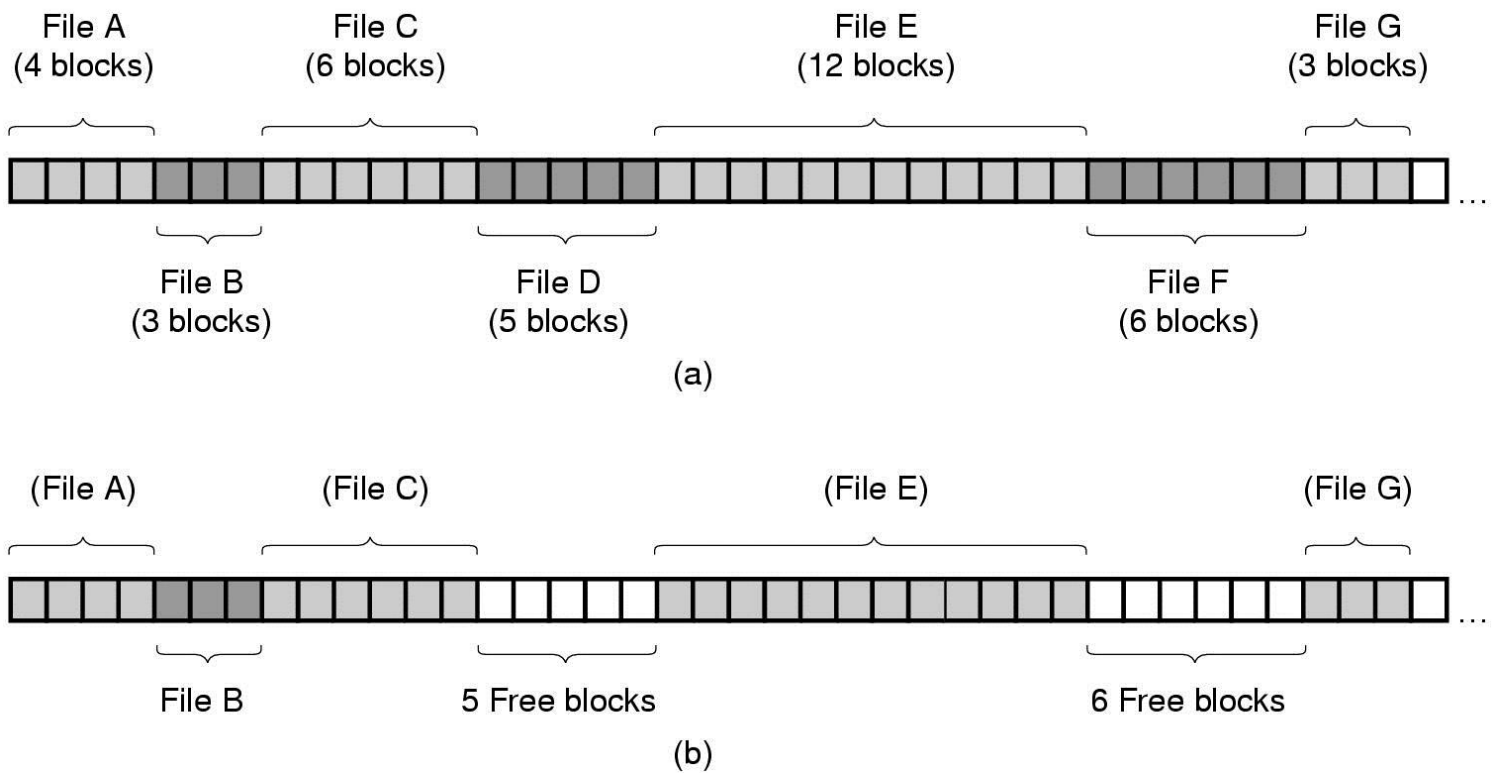
File System Layout

- Each partition has an independent file system.
- MBR(Master Boot Record)
  - Used to boot the computer
  - Partition table
    - Starting and ending addresses of each partition
    - One of the partitions is marked as active
- Boot sequence
  1. BIOS reads in and executes the MBR.
  2. MBR locates the active partition, reads in its first block, called the boot block, and executes it.
  3. The program in the boot block loads the operating system contained in that partition.

Layout of a Disk Partition

- Boot block
- Superblock
  - Key parameters of the file system
    - Magic number to identify the file system type
    - Number of blocks etc.
- Free space mgmt
  - A bitmap or a list of pointers
- I-nodes
  - Attributes and disk addresses of the file's blocks
- Root dir
- Files and directories

Contiguous Allocation



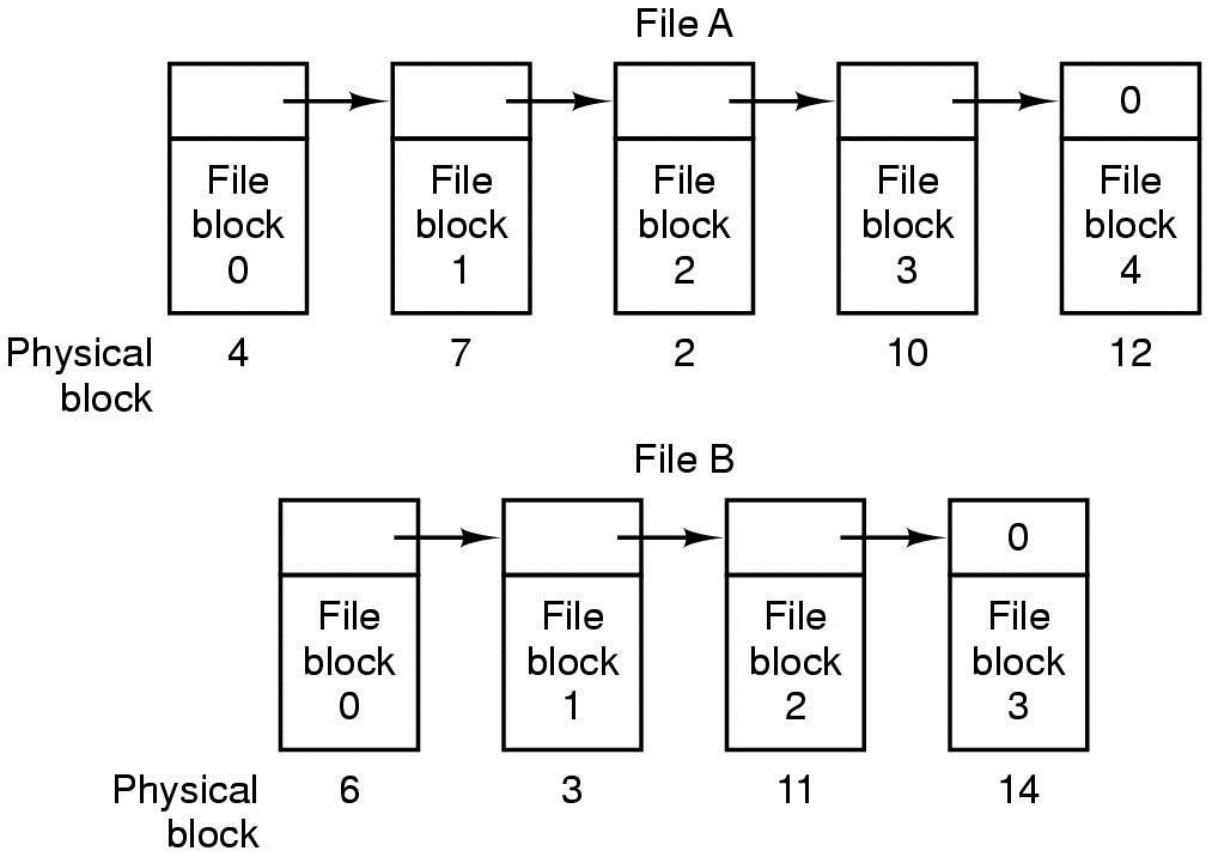
(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

Contiguous Allocation

- Advantages
  - Simple to implement
    - Disk address of the first block
    - Number of blocks in the file
  - High read performace
    - The entire file can be read from the disk in a single operation
- Disadvantages
  - In time, the disk becomes fragmented.
- Feasible and widely used on CD-ROMs

Linked List Allocation



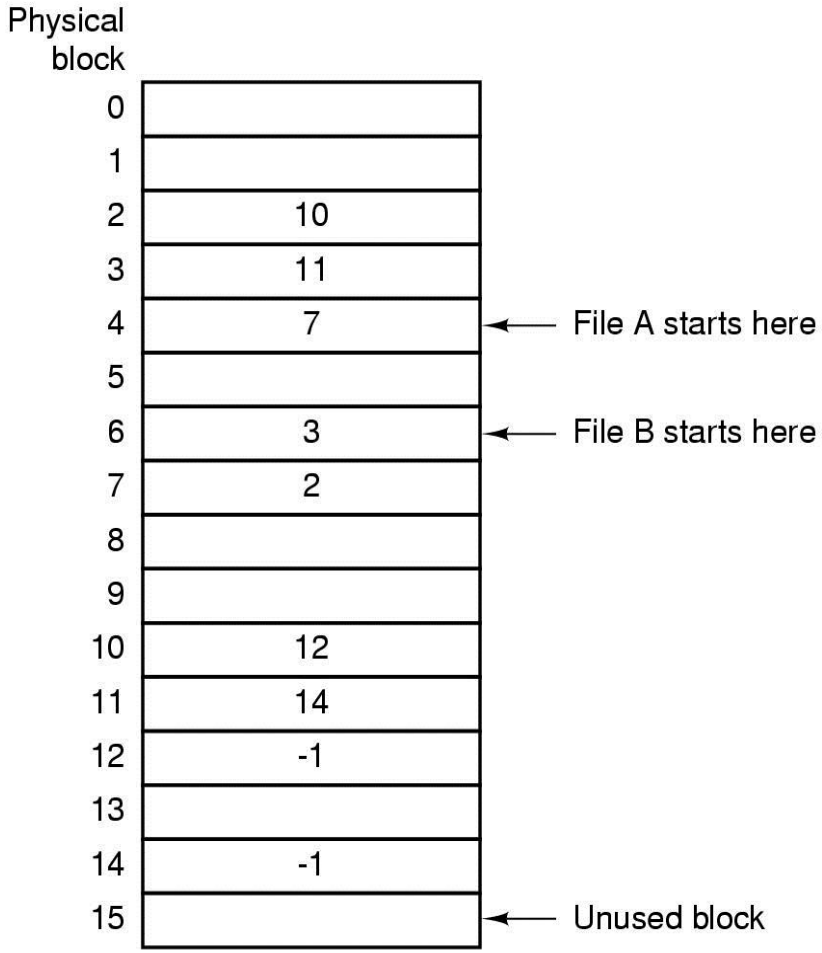
Storing a file as a linked list of disk blocks

Linked List Allocation

- Advantages
  - No space is lost to disk fragmentation
  - The dir entry keeps the disk address of the first block.

- Disadvantages
  - Random access is extremely slow.
  - The amount of data storage in a block is no longer a power of two because the pointer takes up a few bytes. (Many programs read and write in blocks whose size is a power of two.)

Linked List Allocation Using a Table(FAT) in Memory



Linked list allocation using a file allocation table in RAM

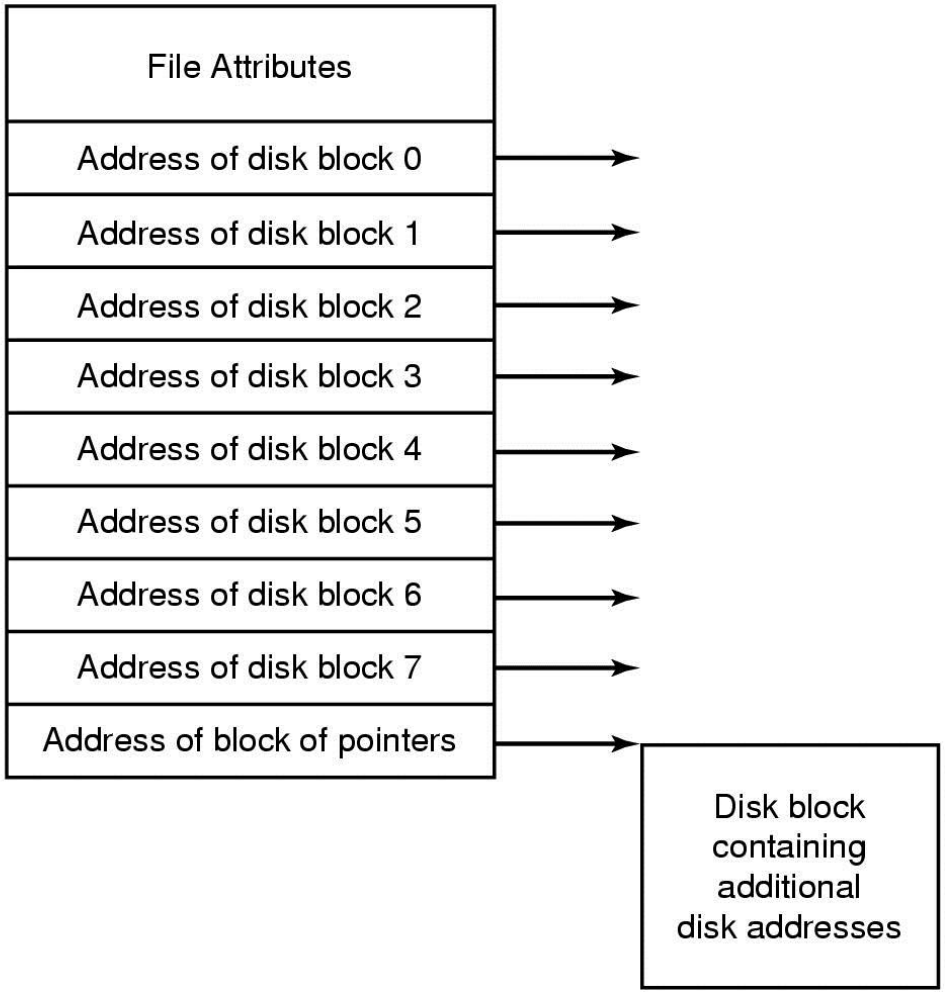
Linked List Allocation Using a Table in Memory

- FAT(File Allocation Table)

- Advantages
  - Random access is much easier.
  - The chain can be followed without making any disk references.
  - The dir entry keeps the starting block number.

- Disadvantages
  - The entire table must be in memory all the time : the table is proportional in size to the disk.

I-nodes

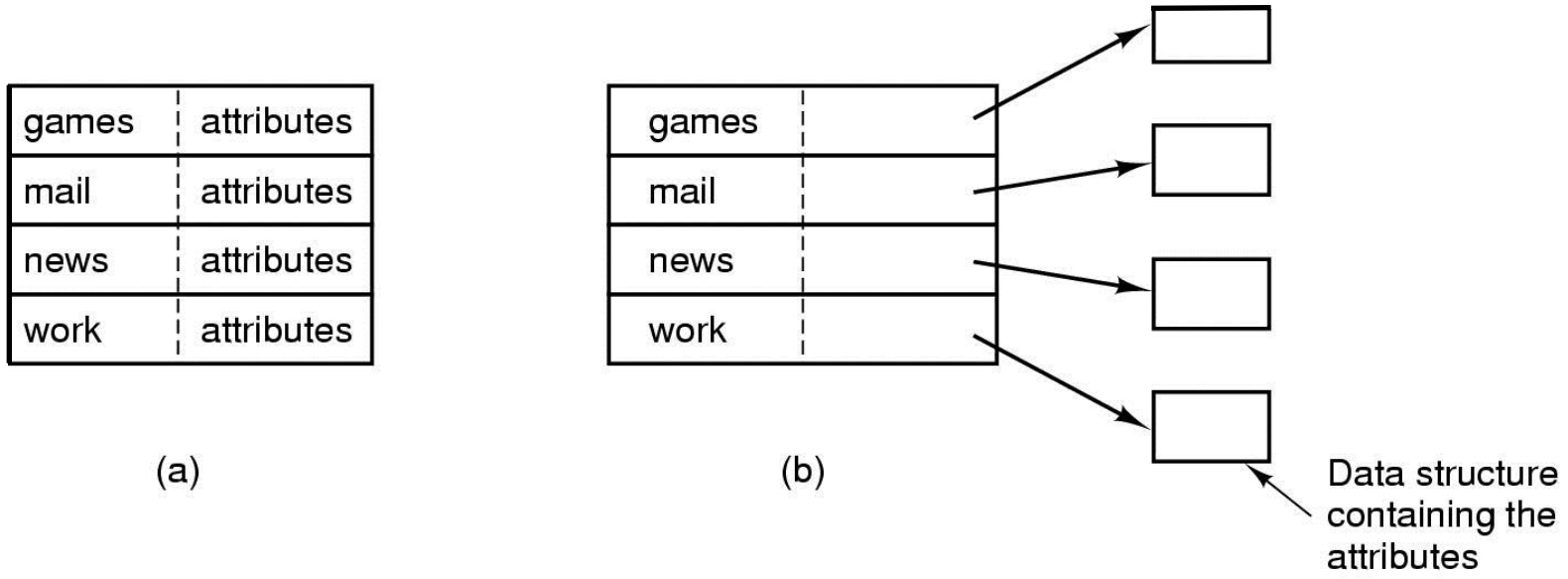


An example i-node

I-nodes

- Attributes and disk addresses of the file's blocks
- Advantages
  - Requires an array in memory whose size is proportional to the maximum number of files that can be open at once
- Last entry
  - Address of block of pointers

Implementing Directories (1)



(a) A simple directory (MS-DOS/Windows)

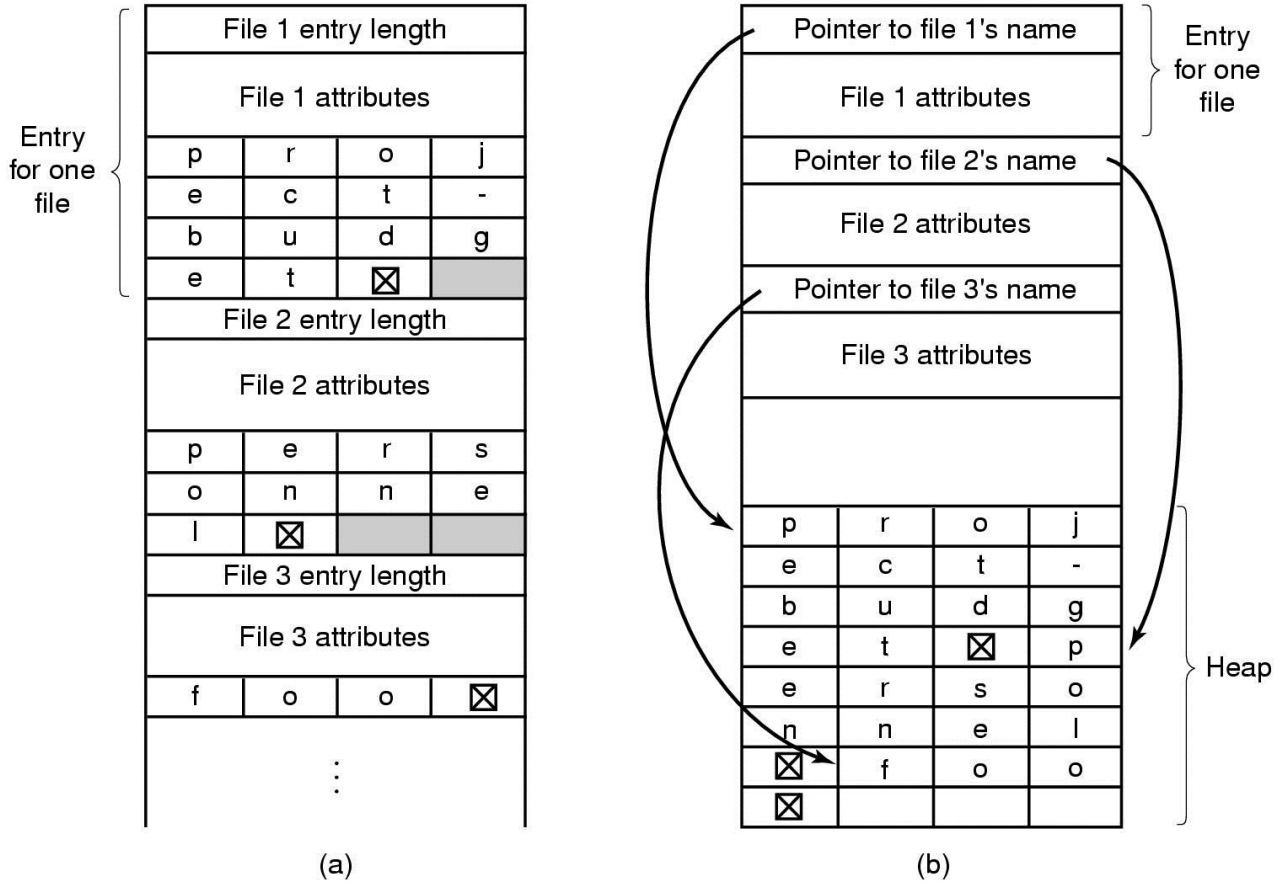
- fixed size entries of

  file name, attributes, and disk addresses

(b) Directory in which each entry just refers to an i-node(UNIX)

- file name and I-node number

Two ways of handling long file names in directory



(a) In-line

- When a file is removed, a hole can be introduced.

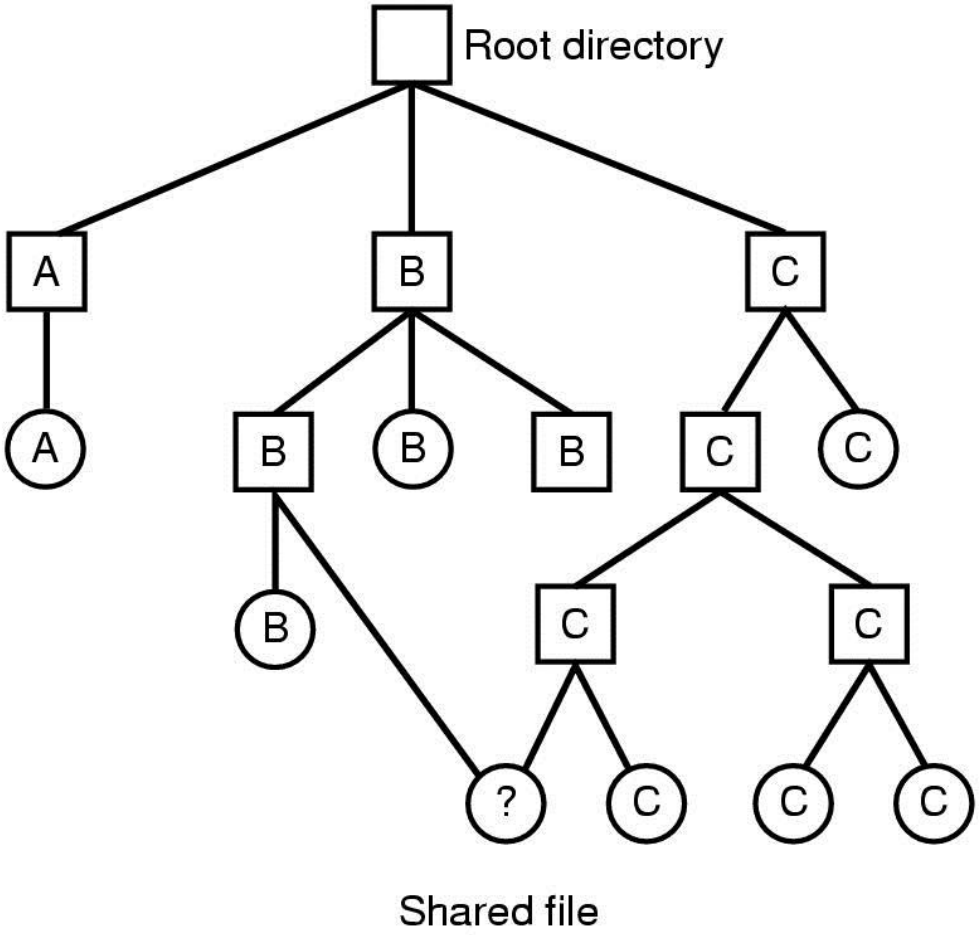
(b) In a heap

- No need for file names to begin at word boundaries

Speeding up the File Name Search

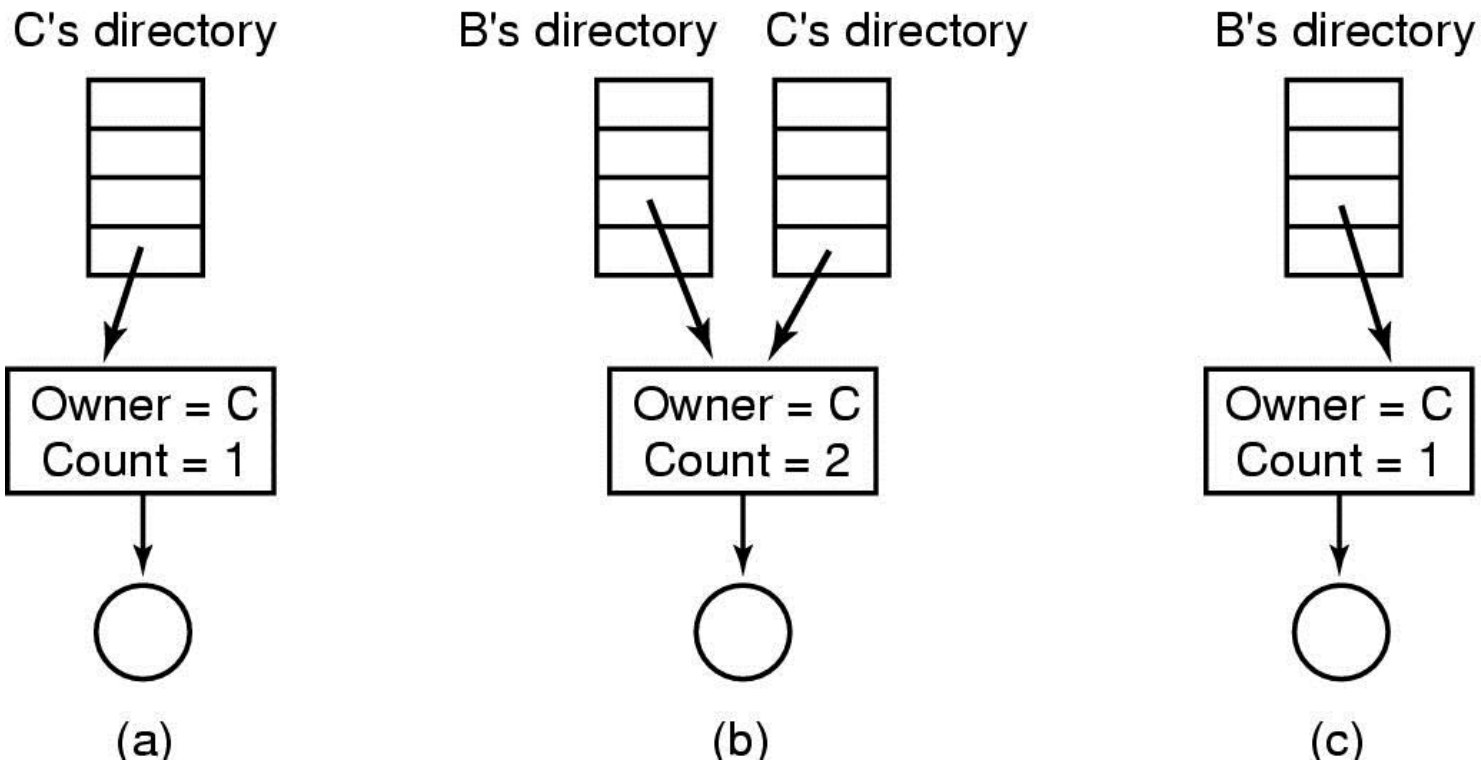
- Hash table
  - Hashes by the file name
  - More complex administration
  - Works if directories contain a large number of files
- Caching
  - Caches the results of searches
  - Works if a small number of files comprise the majority of the lookups

Shared Files (1)



File system containing a shared file

Shared Files (2)



(a) Situation prior to linking

(b) After the link is created

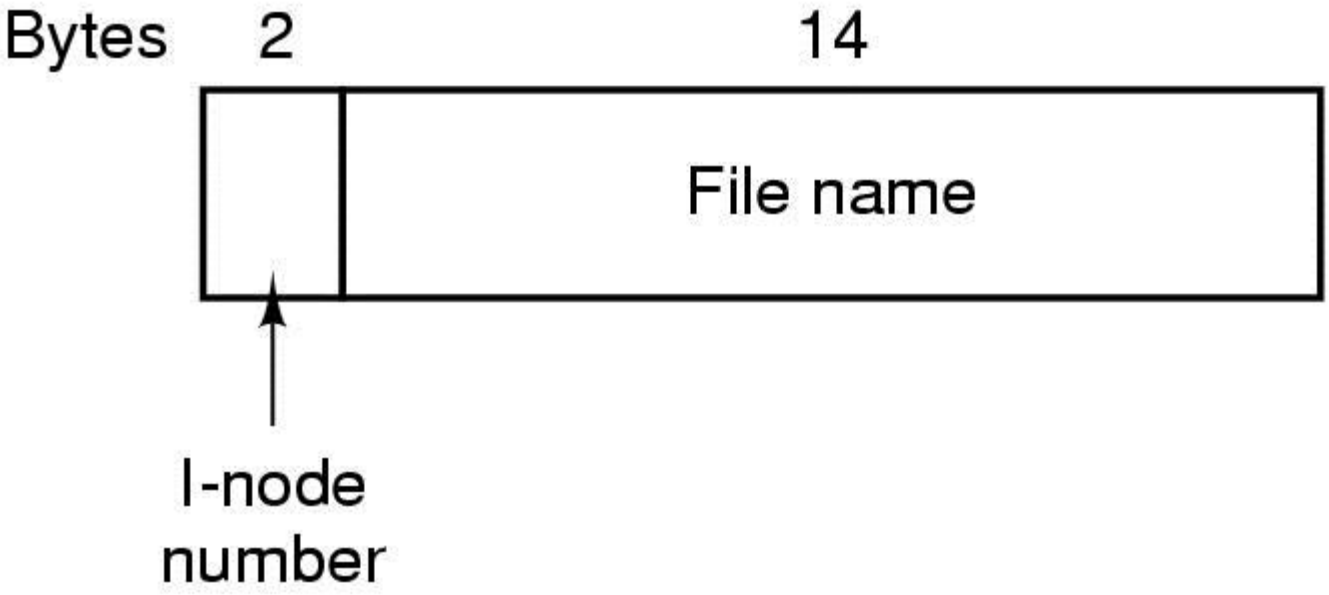
(c)After the original owner removes the file

Shared Files (3)

- Hard Link
  - A directory entry points to a data structure associated with the file.
  - Does not change the ownership, but increases the link count
  - Hard to find all the directory entries for a file to be deleted

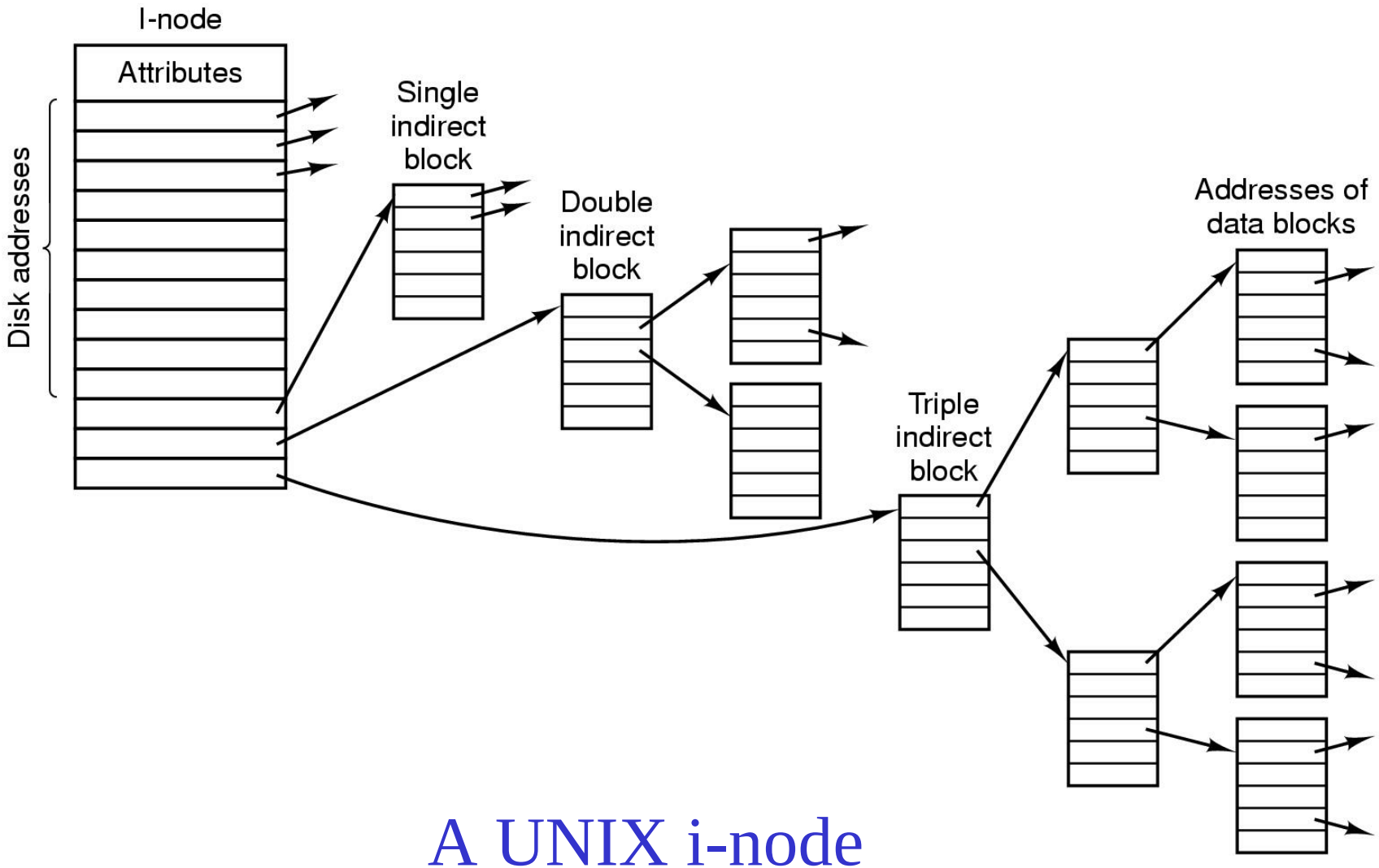
- Symbolic Link
  - A directory entry for a new file of type LINK contains the path name of the file to be linked.
  - Extra disk access to reach a file
  - Extra i-node (and disk block to store the path)
  - Can be used to link files on different machines

The UNIX V7 File System (1)



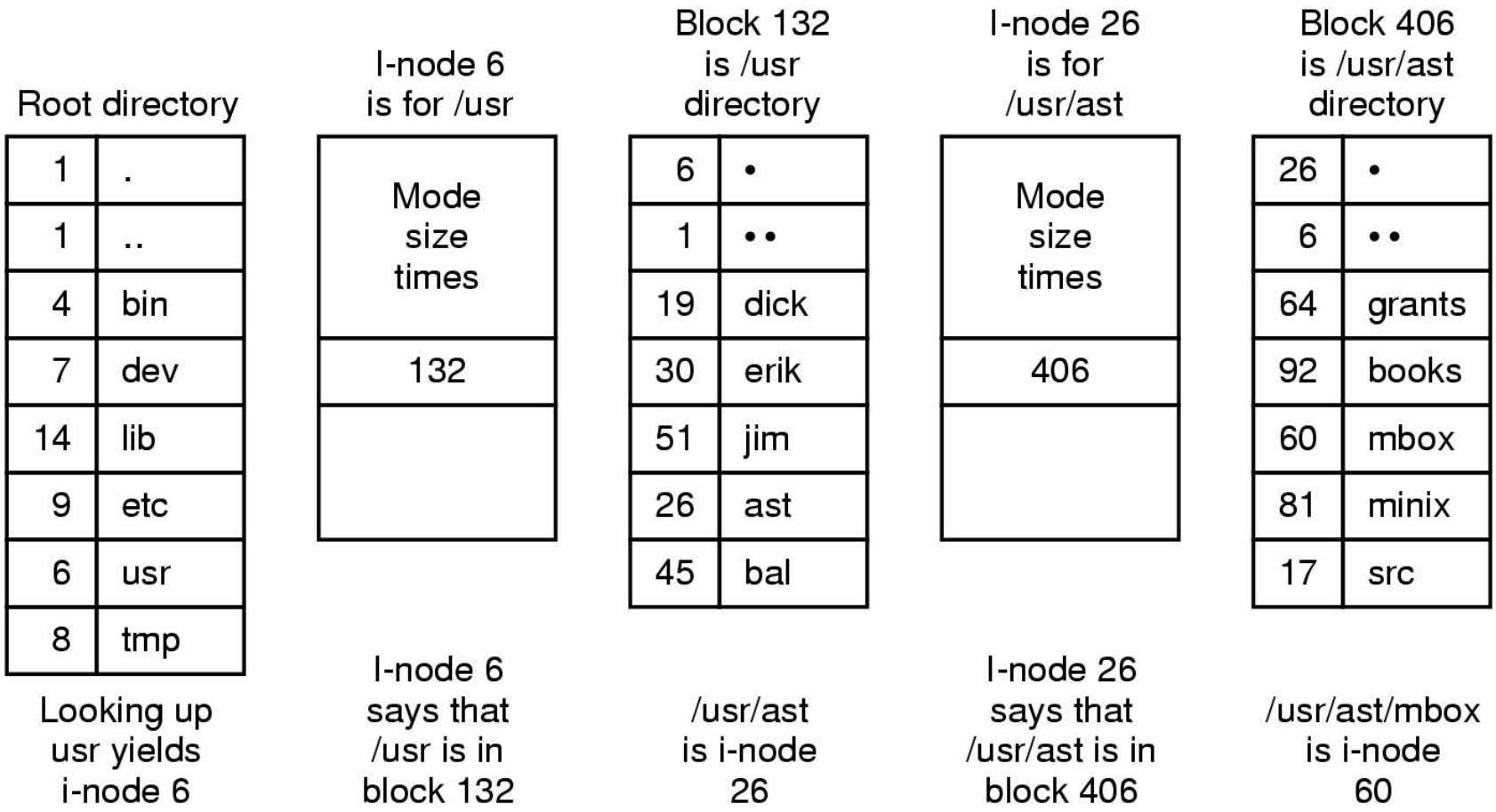
A UNIX V7 directory entry

The UNIX V7 File System (2)



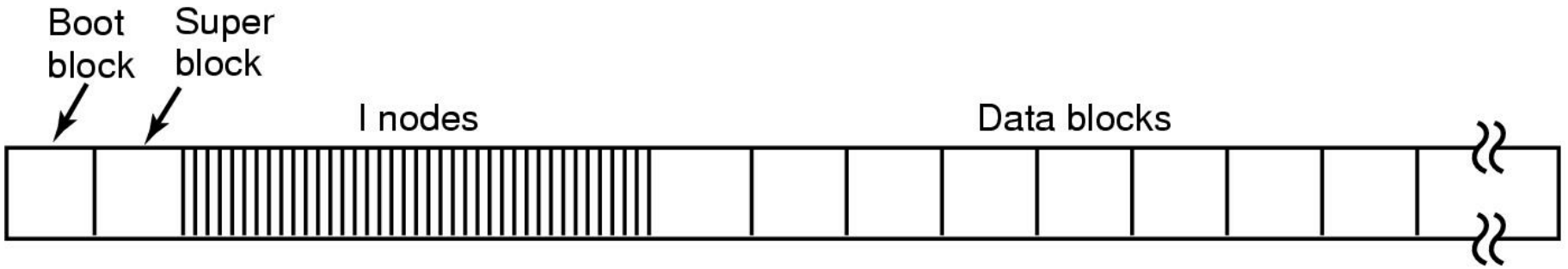
A UNIX i-node

The UNIX V7 File System (3)



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

UNIX File System (1)



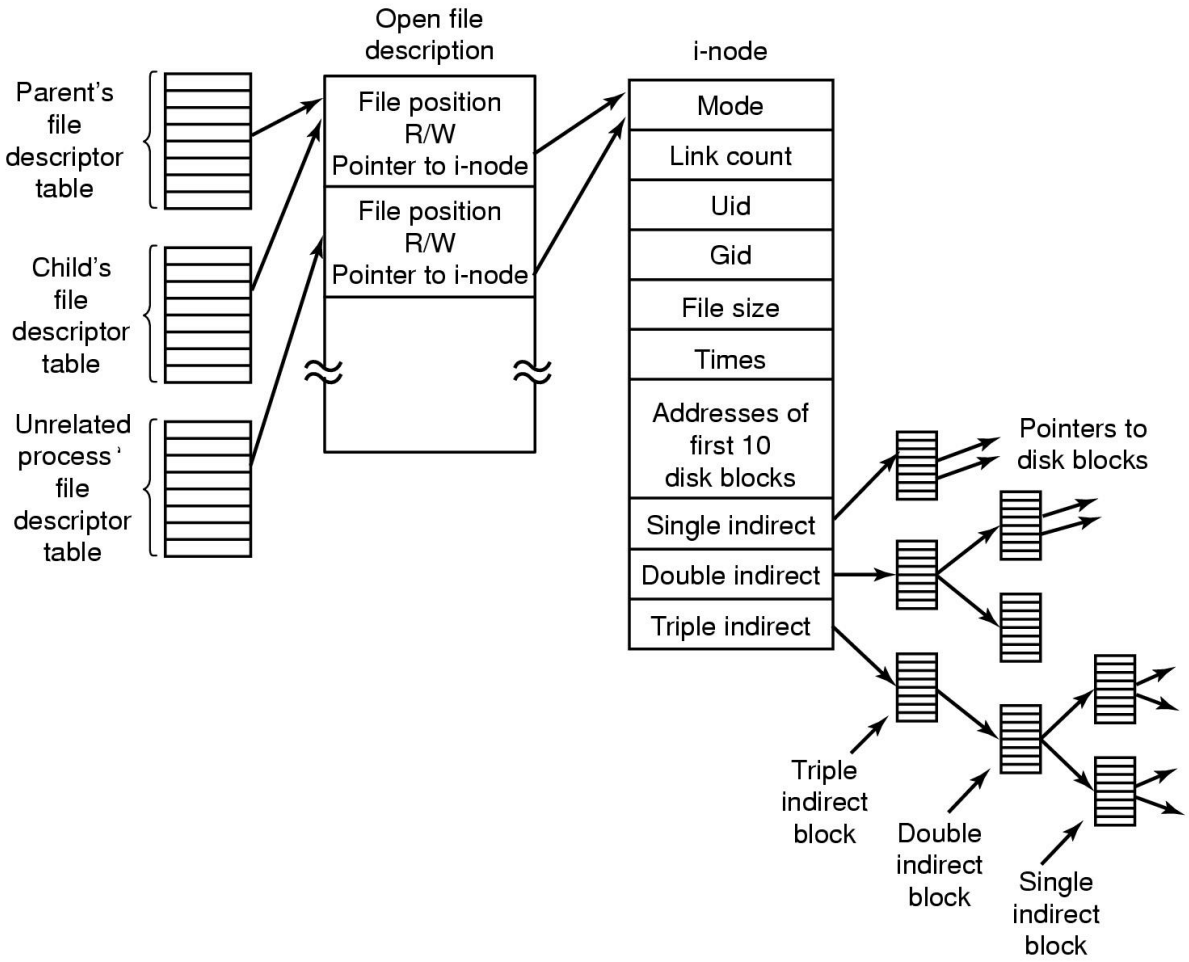
Disk layout in classical UNIX systems

UNIX File System (2)

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Structure of the i-node

UNIX File System (3)



The relation between the file descriptor table, the open file description