File Systems

Instructor: Dr. Liting Hu

Motivation

- Many important applications need to store more information than have in virtual address space of a process
- The information must survive the termination of the process using it.
- Multiple processes must be able to access the information concurrently.

Motivation

- Disks are used to store files
- Information is stored in blocks on the disks
- Can read and write blocks

Motivation

- Use file system as an abstraction to deal with accessing the information kept in blocks on a disk
- Files are created by a process
- Thousands of them on a disk
- OS structures them, names them, protects them
- Two ways of looking at file system
  - User - how do we name a file, protect it, organize the files
  - Implementation - how are they organized on a disk

Start with user view, then go to implementer view

File Systems

- The user point of view
  - Naming
  - Structure
  - Directories

Naming

- One to 8 letters in all current OS's
- Unix, MS-DOS (Fat16) file systems discussed
- Fat (16 and 32) were used in first Windows systems
- Latest Window systems use Native File System
- All OS's use suffix as part of name
- **Unix does not always enforce a meaning for the suffixes**
- **DOS does enforce a meaning**

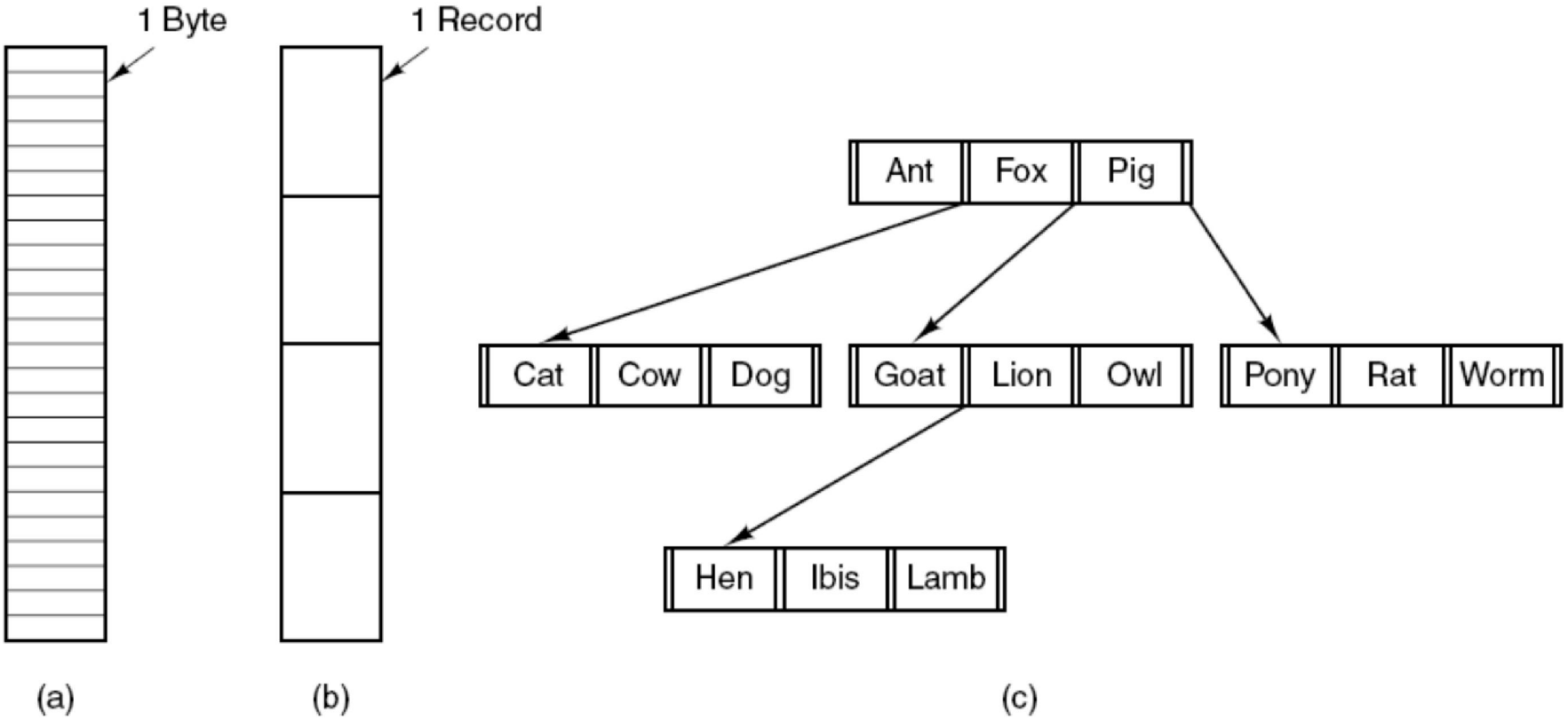
Suffix Examples

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

File Structure

- Byte sequences
  - Maximum flexibility - can put anything in
  - Unix and Windows use this approach
- Fixed length records (card images in the old days)
- Tree of records - uses key field to find records in the tree

File Structure



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

File Types

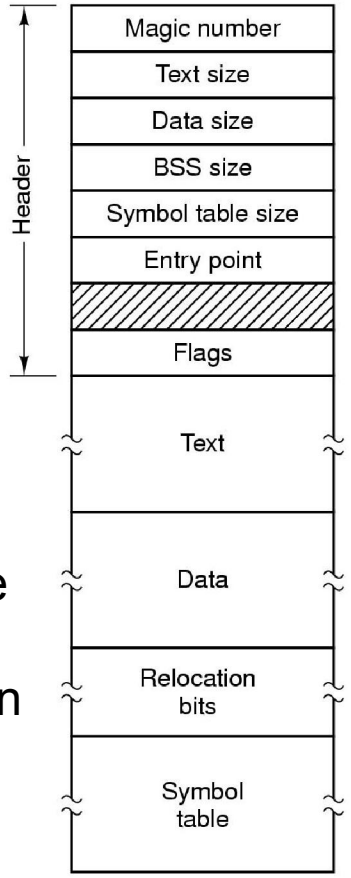
- Regular- contains user information
- Directories
- Character special files- model serial (e.g. printers) I/O devices
- Block special files - model disks

Regular Files

- ASCII or binary
- ASCII
  - Printable
  - Can use pipes to connect programs if they produce/consume ASCII
- Binary - Two Unix examples
  - Executable (magic field identifies file as being executable)
  - Archive-compiled, not linked library procedures
- Every OS must recognize its own executable

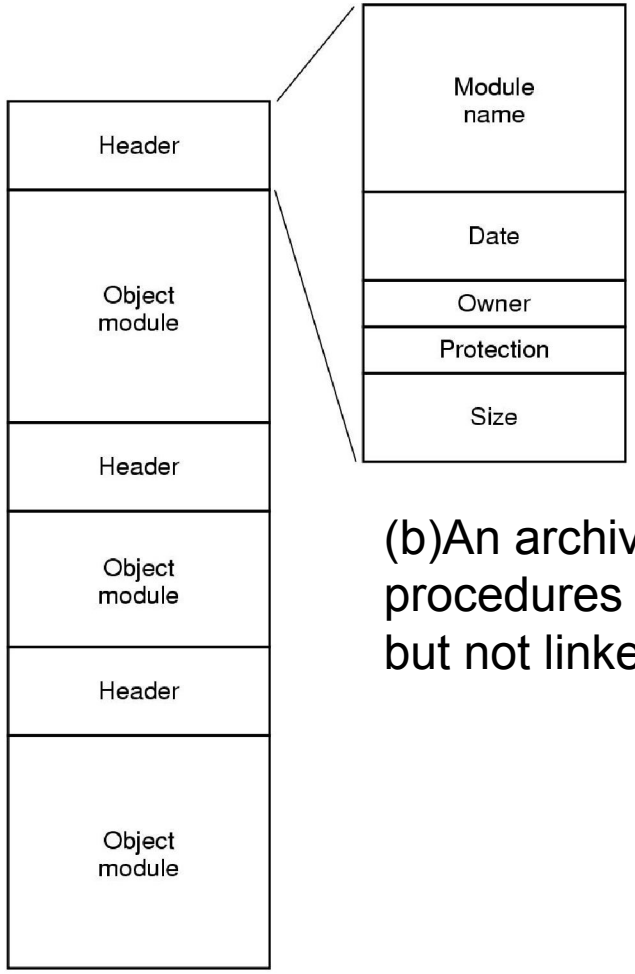
ASCII	'c'	'a'	't'
Hex	63	61	74
Binary	0110 0011	0110 0001	0111 1000

Binary File Types (Early Unix)



(a)

(a)An executable file (magic # identifiies it as an executable file)



(b)

(b)An archive-library procedures compiled but not linked

File Access

- Sequential access- read from the beginning, can't skip around
  - Corresponds to magnetic tape
- Random access- start where you want to start
  - Came into play with disks
  - Necessary for many applications, e.g. airline reservation system

File Attributes (hypothetical OS)

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

System Calls for files

- Create - with no data, sets some attributes
- Delete - to free disk space
- Open - after create, gets attributes and disk addresses into main memory
- Close - frees table space used by attributes and addresses
- Read - usually from current pointer position. Need to specify buffer into which data is placed
- Write - usually to current position

System Calls for files

- Append - at the end of the file
- Seek - puts file pointer at specific place in file. Read or write from that position on
- Get Attributes - e.g. make needs most recent modification times to arrange for group compilation
- Set Attributes - e.g. protection attributes
- Rename

How can system calls be used? An example-copyfile abc xyz

copyfile abc xyz

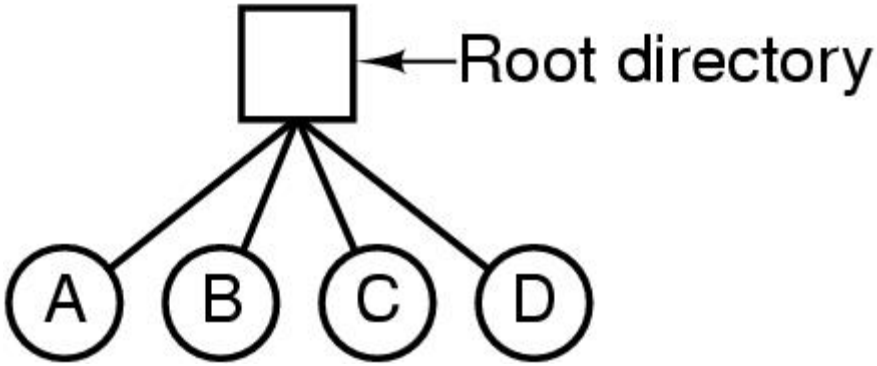
- Copies file abc to xyz
- If xyz exists it is over-written
- If it does not exist, it is created
- Uses system calls (read, write)
- Reads and writes in 4K chunks
- Read (system call) into a buffer
- Write (system call) from buffer to output file

```
/* 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 */
}
```

Directories

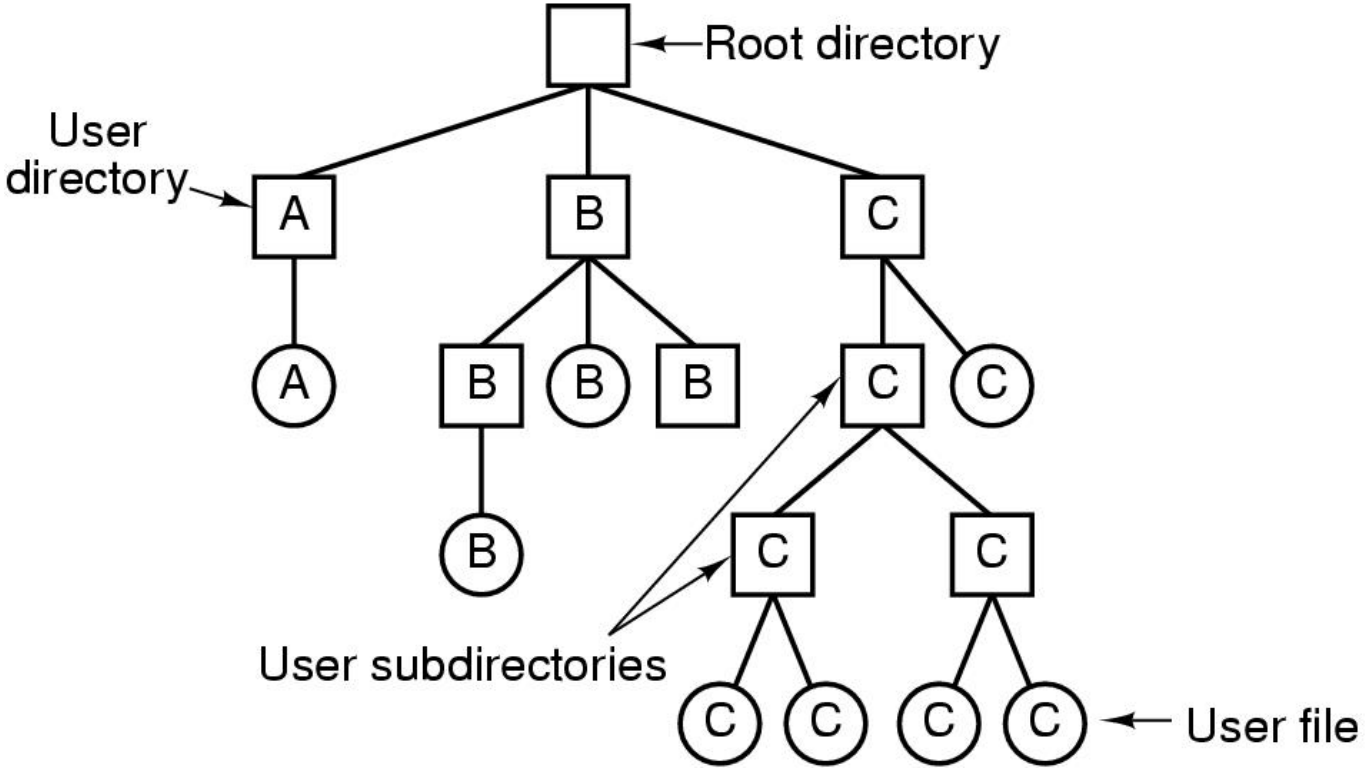
- **Files** which are used to organize a collection of files
- Also called folders in weird OS's

Single Level Directory Systems



A single-level directory system containing four files.

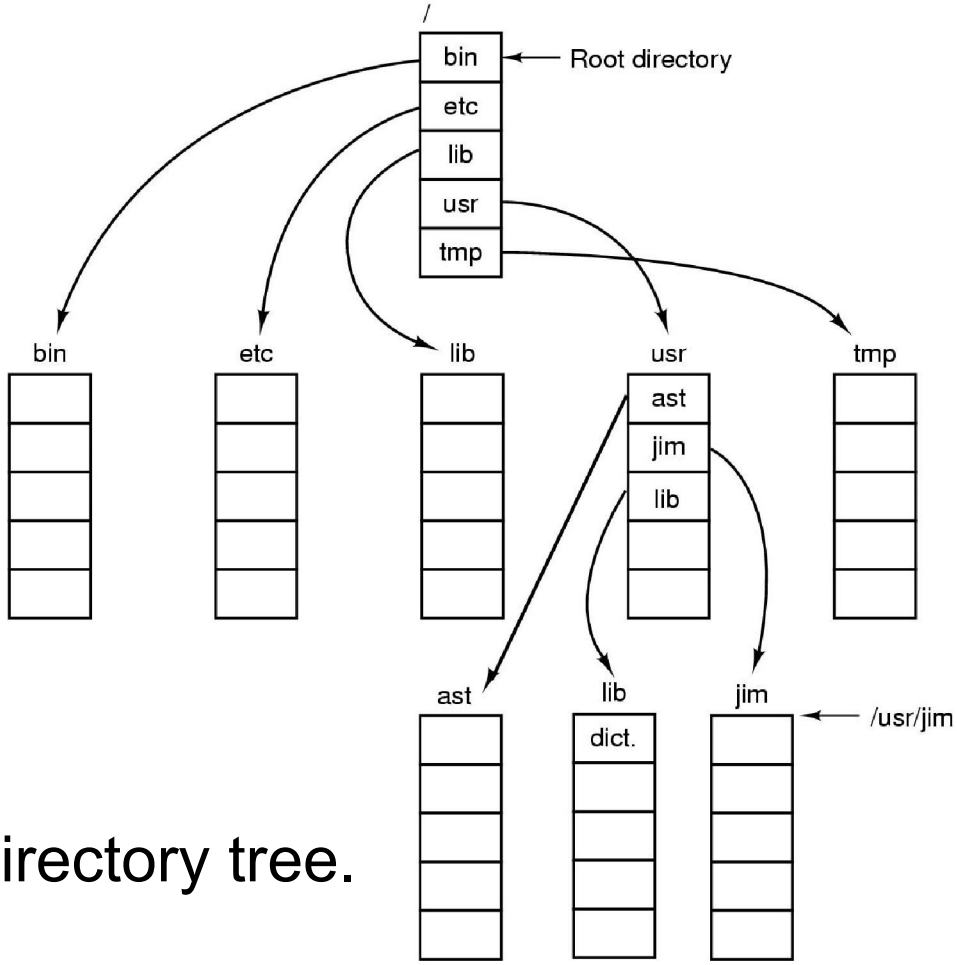
Hierarchical Directory Systems



Path names

- Absolute /usr/carl/cs310/miderm/answers
- Relative cs310/midterm/answers
- . Refers to current (working) directory
- .. Refers to parent of current directory

Path Names



A UNIX directory tree.

Directory Operations

- Create - creates directory
- Delete - directory has to be empty to delete it
- Opendir - Must be done before any operations on directory
- Closedir
- Rename
- Link - links file to another directory
- Unlink - Gets rid of directory entry

Let's go to implementer view

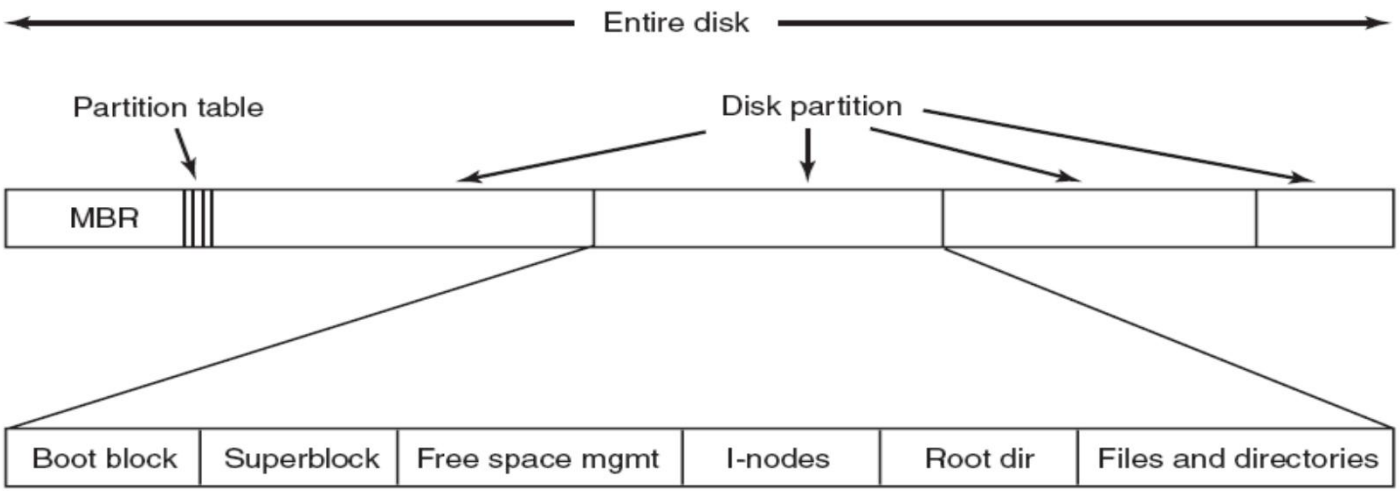
File Implementation

- Files stored on disks. Disks broken up into one or more partitions, with separate fs on each partition
- Sector 0 of disk is the Master Boot Record
- Used to boot the computer
- End of MBR has partition table. Has starting and ending addresses of each partition.
- One of the partitions is marked active in the master boot table

File Implementation

- Boot computer => BIOS reads/executes MBR
- MBR finds active partition and reads in first block (boot block)
- Program in boot block locates the OS for that partition and reads it in
- All partitions start with a boot block

A Possible File System Layout



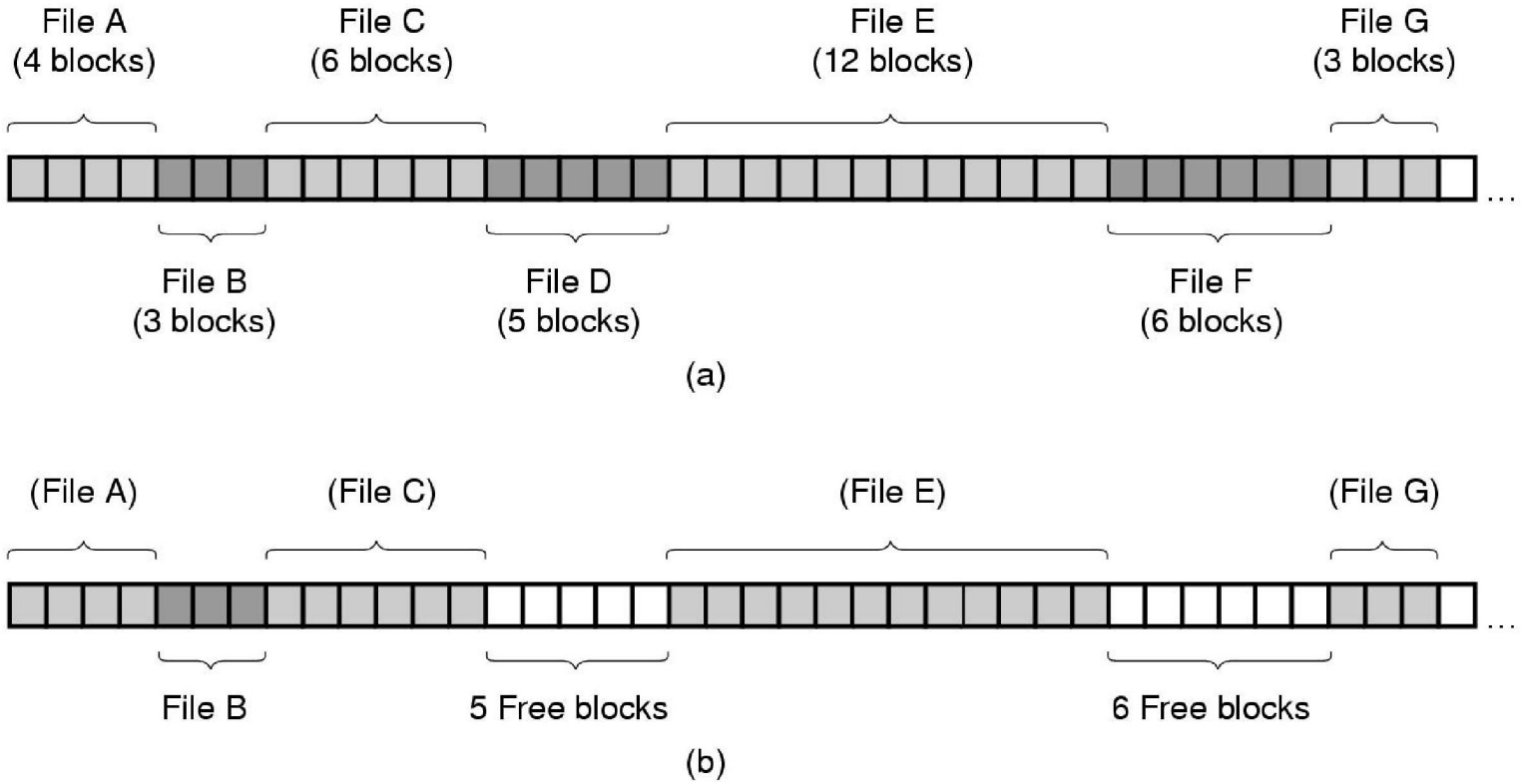
File System Layout

- Superblock contains info about the fs (e.g. type of fs, number of blocks, …)
- i-nodes contain info about files

Allocating Blocks to files

- Most important implementation issue
- Methods
  - Contiguous allocation
  - Linked list allocation
  - Linked list using table
  - I-nodes

Contiguous Allocation



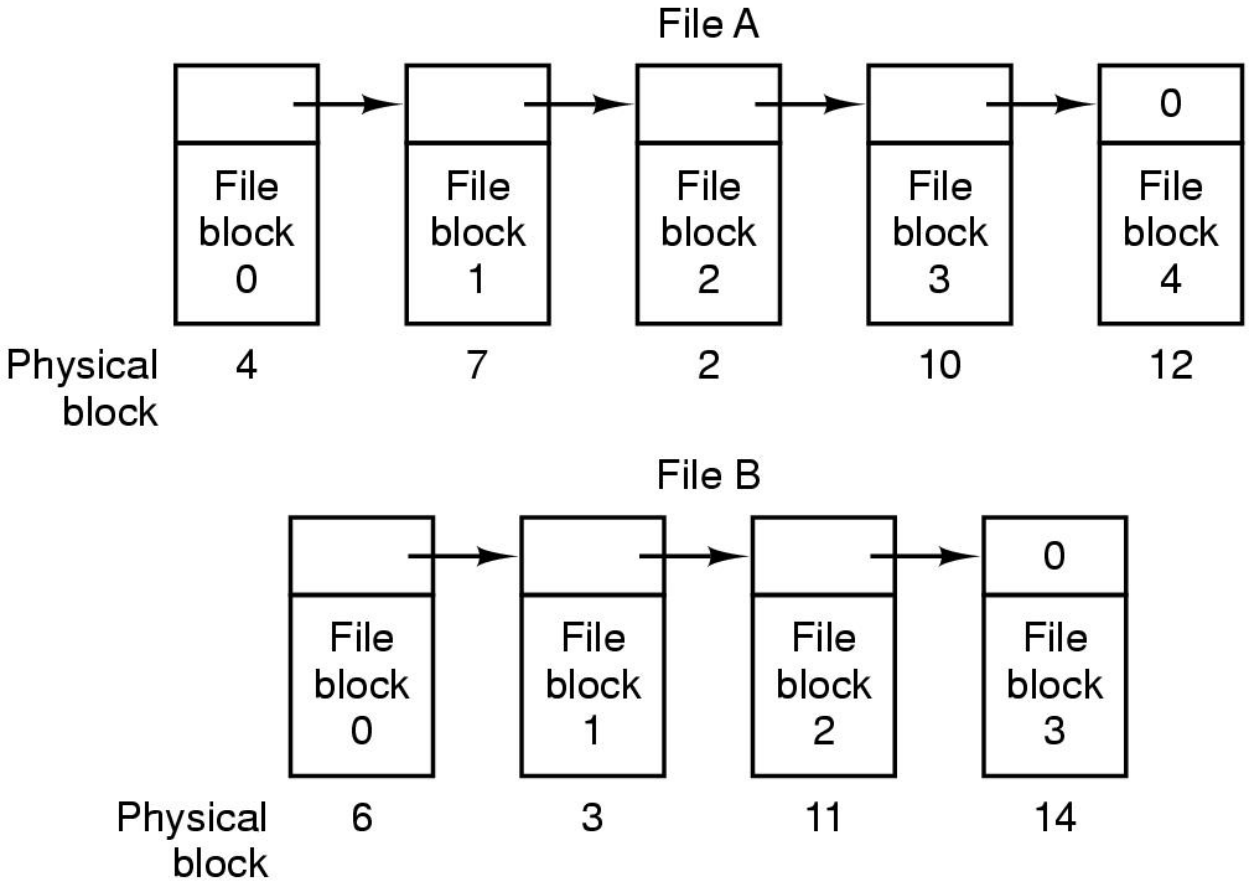
(a) Contiguous allocation of disk space for 7 files.

(b) The state of the disk after files D and F have been removed.

Contiguous Allocation

- The good
  - Easy to implement
  - Read performance is great. Only need one seek to locate the first block in the file. The rest is easy.
- The bad
  - **Disk becomes fragmented over time**
  - CD-ROM's use contiguous allocation because the fs size is known in advance
  - DVD's are stored in a few consecutive 1 GB files because standard for DVD only allows a 1 GB file max

Linked List Allocation



Storing a file as a linked list of disk blocks.

Linked Lists

The good

- Gets rid of fragmentation

The bad

- Random access is slow. Need to chase pointers to get to a block

Linked lists using a table in memory

- Put pointers in table in memory
- File Allocation Table (FAT)
- Windows

The Solution-Linked List Allocation Using a Table in Memory

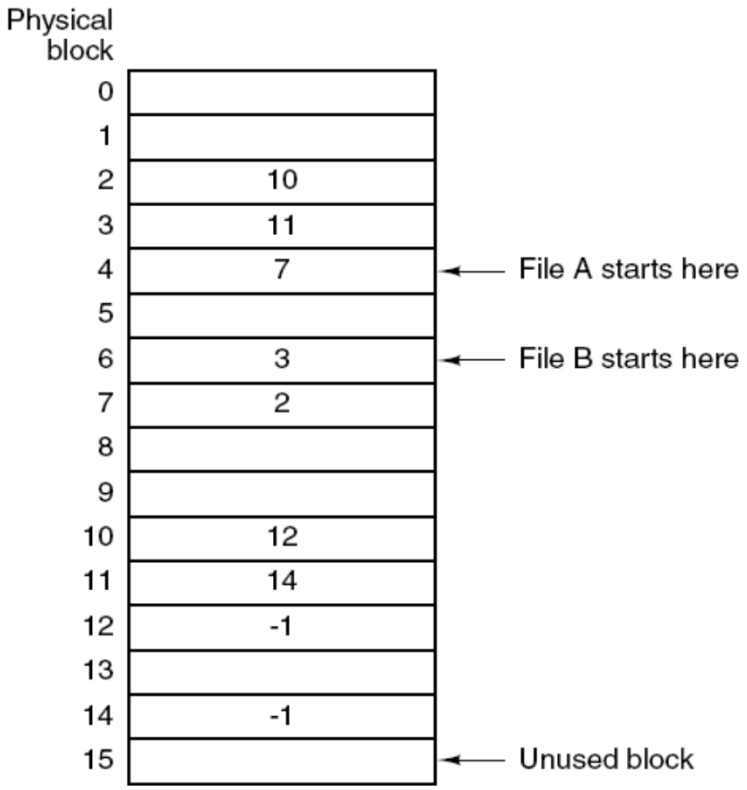


Figure 4-12. Linked list allocation using a file allocation table in main memory.

Linked lists using a table in memory

- The bad – table becomes really big
  - E.g 200 GB disk with 1 KB blocks needs a 600 MB table
  - Growth of the table size is linear with the growth of the disk size

I-node

- Keep data structure in memory only for active files
  - Data structure lists disk addresses of the blocks and attributes of the files
- K active files, N blocks per file => k*n blocks max!!
- Solves the growth problem
  - Very big N: Last entry in table points to disk block which contains pointers to other disk blocks

I-node

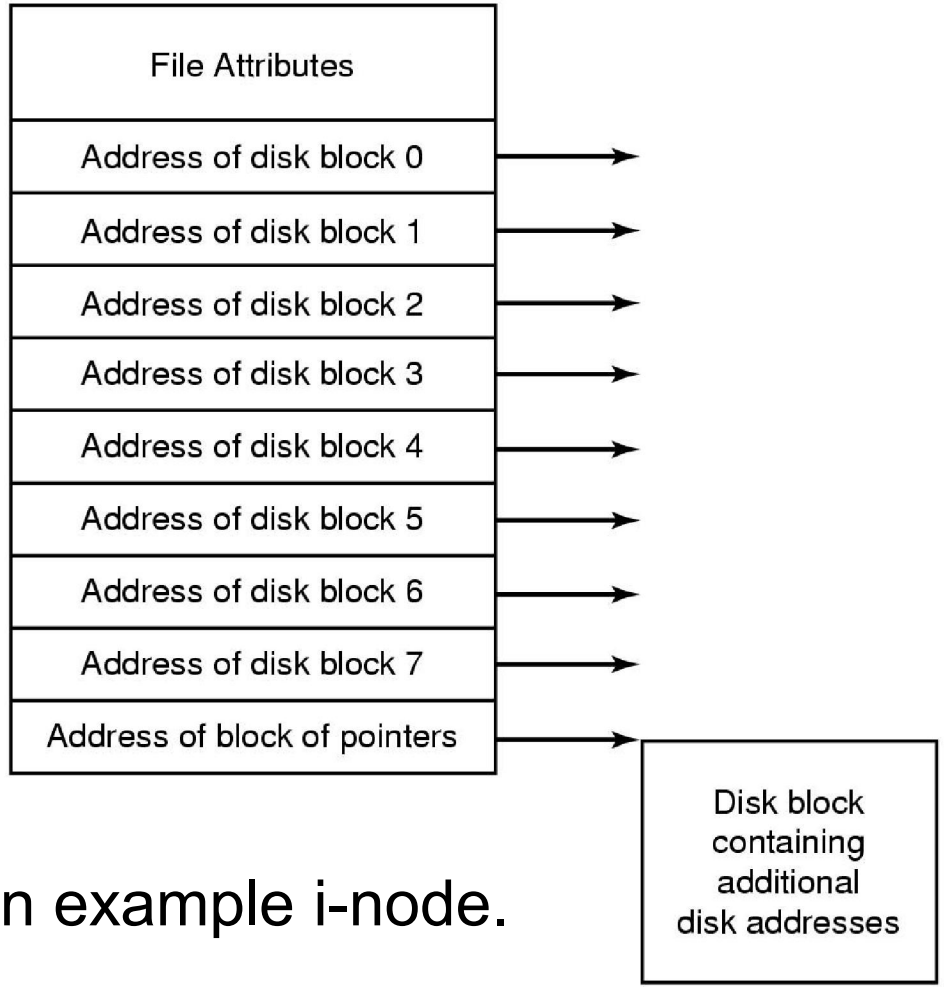


Figure 4-13. An example i-node.

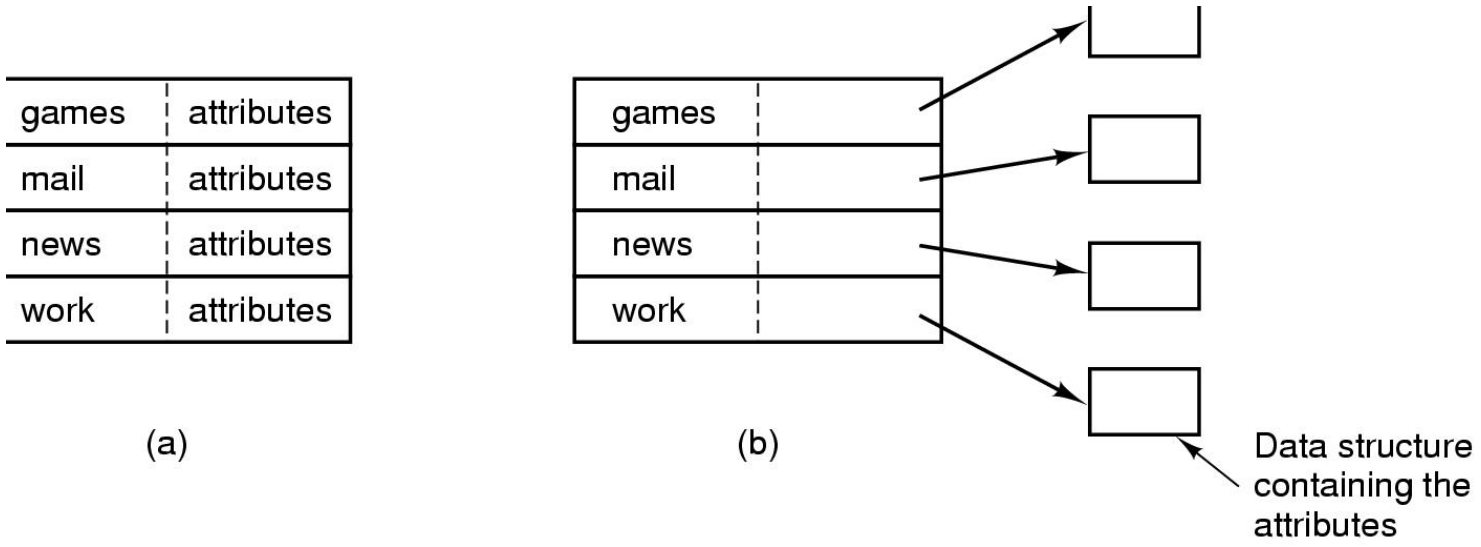
I-node Structure

- https://www.youtube.com/watch?v=tMVj22EWg6A

Implementing Directories

- Open file, path name used to locate directory
- Directory specifies block addresses by providing
  - Address of first block (contiguous)
  - Number of first block (linked)
  - Number of i-node

Implementing Directories

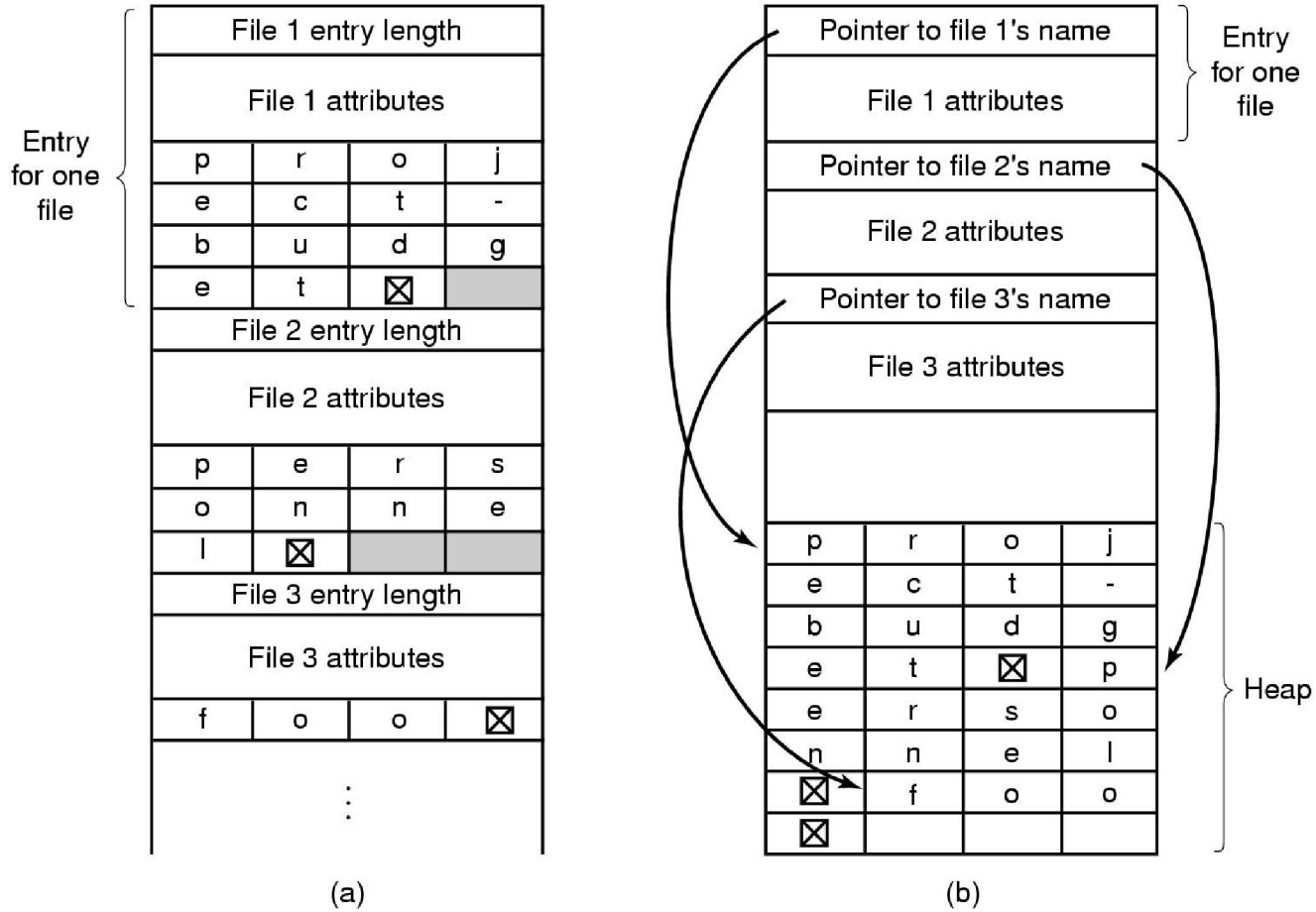


(a) fixed-size entries with the disk addresses and attributes (DOS)
(b) each entry refers to an i-node. Directory entry contains attributes. (Unix)

Implementing Directories

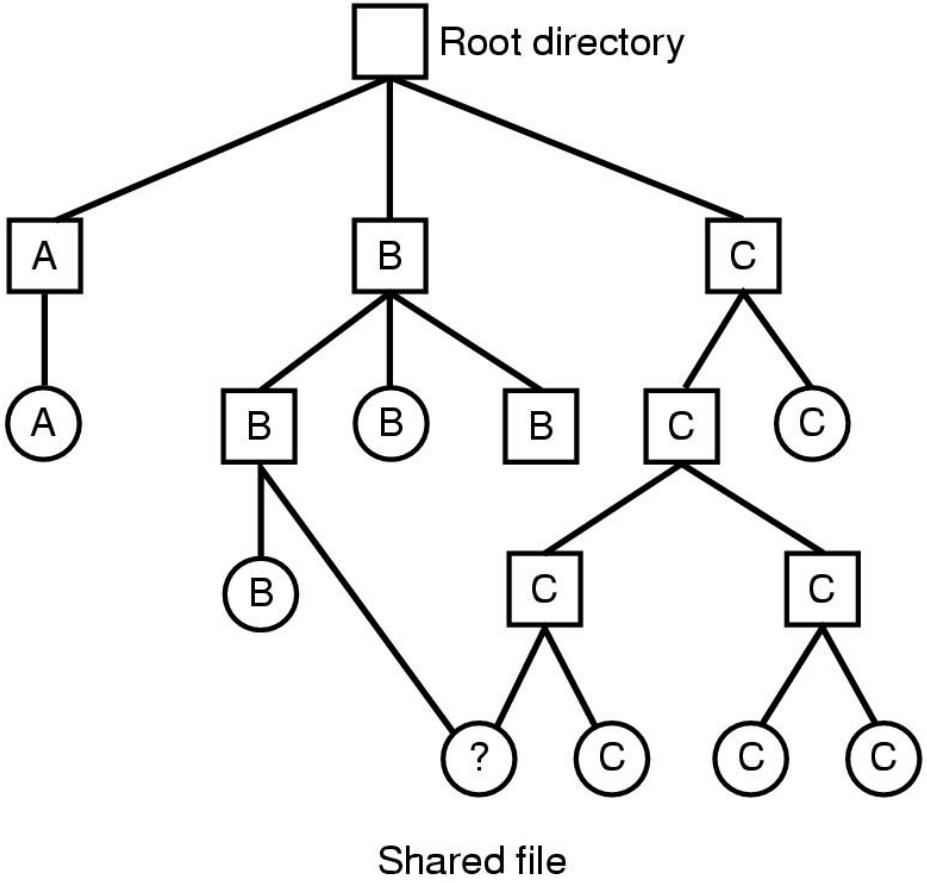
- How do we deal with variable length names?
- Problem is that names have gotten very long
- Two approaches
  - Fixed header followed by variable length names
  - Heap-pointer points to names

Implementing Directories



Handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files

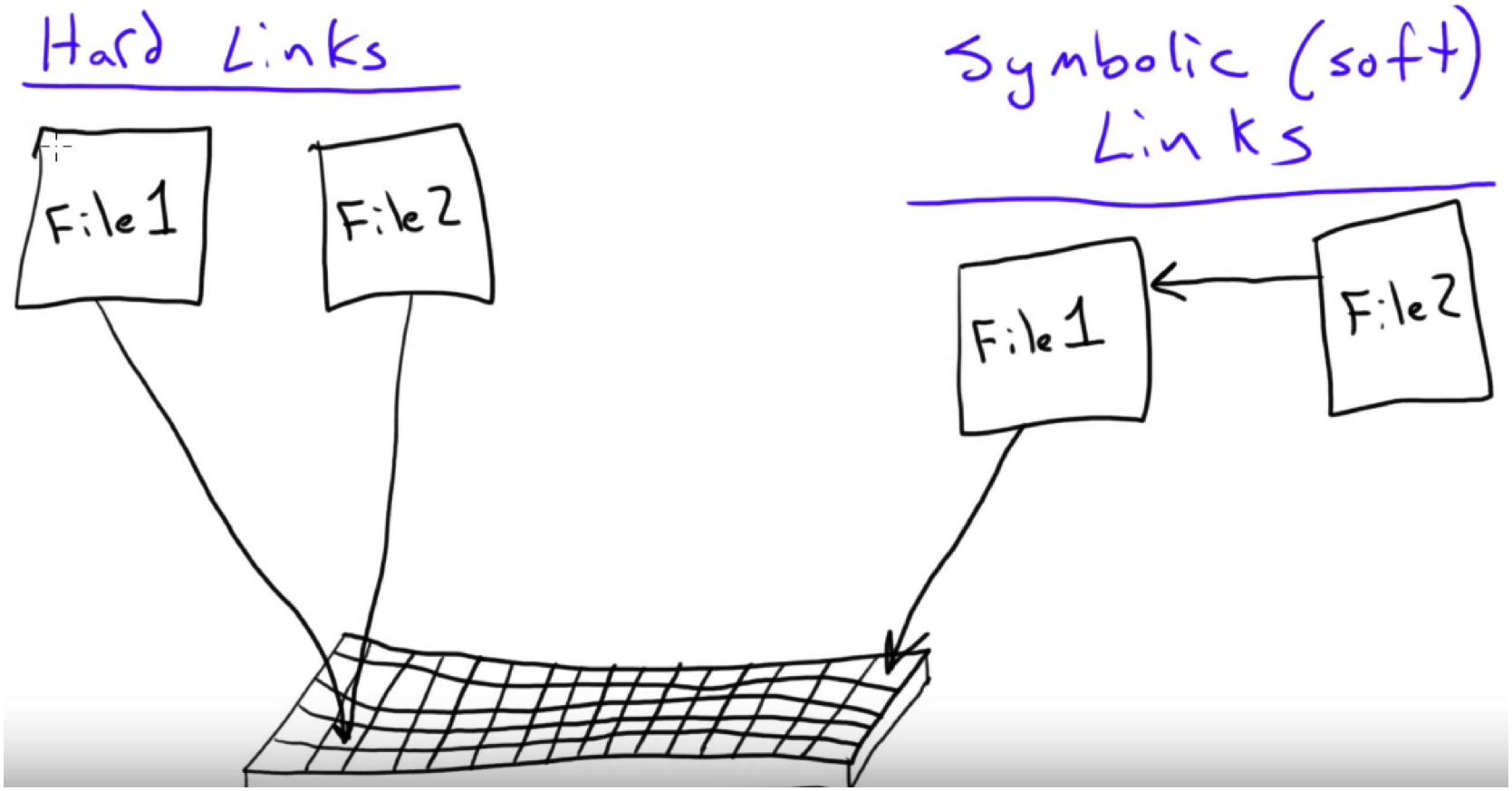


Shared file

File system containing a shared file. File systems is a directed acyclic tree (DAG)

Shared files

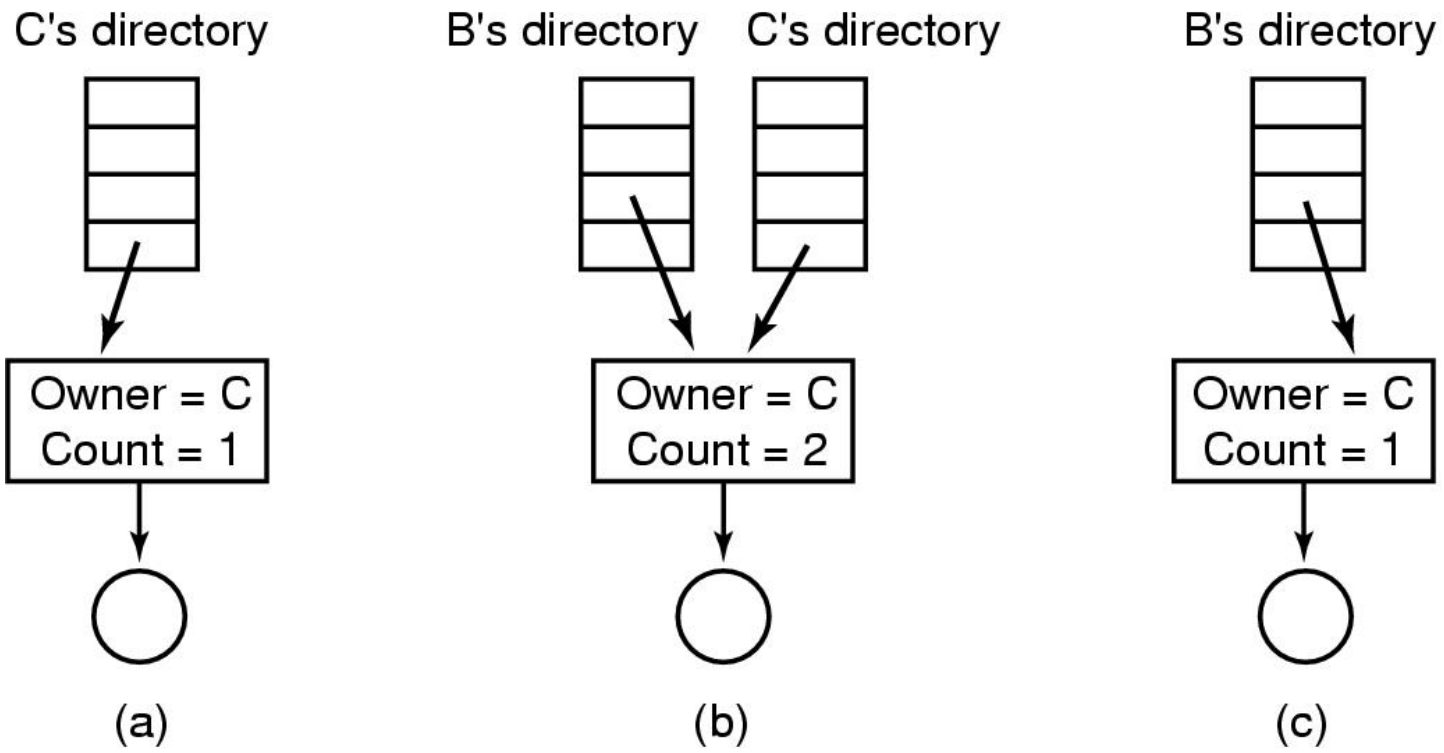
- If B or C adds new blocks, how does other owner find out?
  - Use special i-node for shared files-indicates that file is shared
  - Use symbolic link - a special file put in B's directory if C is the owner. Contains the path name of the file to which it is linked


Hard Links
File1
File2
Symbolic (soft) Links
File1
File2

I-node problem

- If C removes file, B's directory still points to i-node for shared file
- If i-node is re-used for another file, B's entry points to wrong i-node
- Solution is to leave i-node and reduce number of owners

I-node problem and solution



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

Journaling File Systems

Want to guard against lost files when there are crashes. Consider what happens when a file has to be removed.

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

If there is a crash somewhere in this process, have a mess.

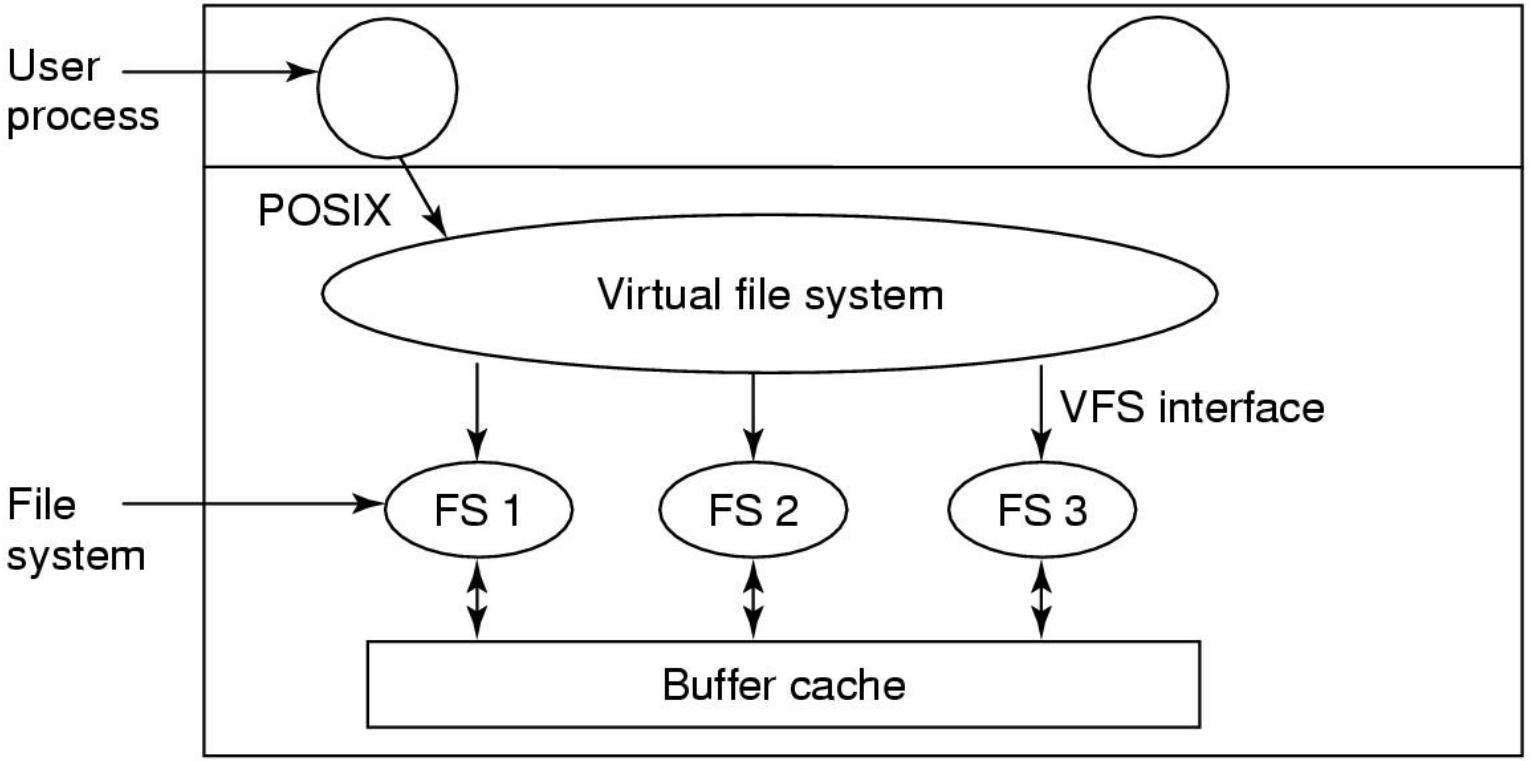
Journaling File Systems

- Keep a journal (i,.e. list) of actions before you take them, write journal to disk, then perform actions. Can recover from a crash!
- Need to make operations idempotent. Must arrange data structures to do so.
  - Mark block n as free is an idempotent operation.
  - Adding freed blocks to the end of a list is not idempotent
  - NTFS (Windows) and Linux use journaling

Virtual File Systems

- Have multiple fs on same machine
- Windows specifies fs (drives)
- Unix integrates into VFS
  - Vfs calls from user
  - Lower calls to actual fs
- Supports Network File System-file can be on a remote machine

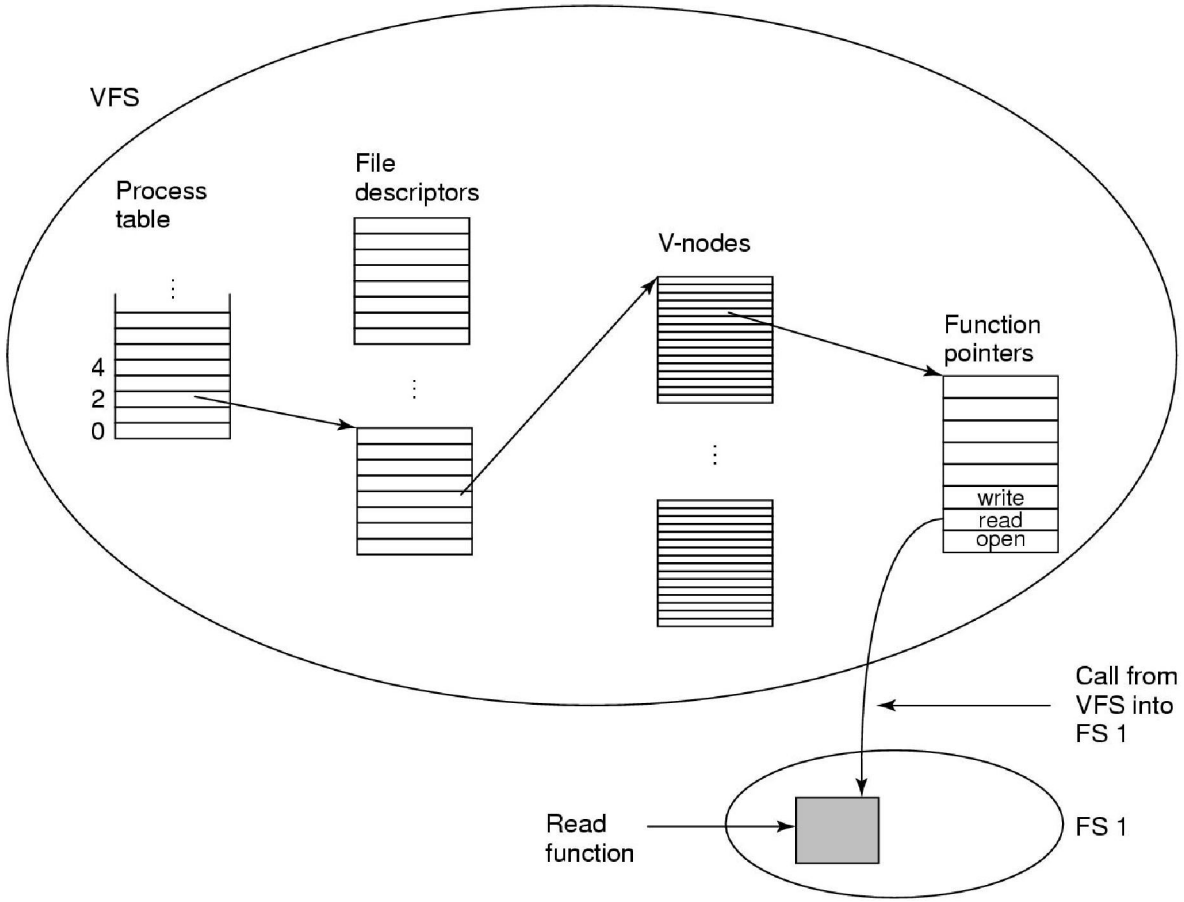
Virtual File Systems (1)



VFS-how it works

- File system registers with VFS (e.g. at boot time)
- At registration time, fs provides list of addresses of function calls the vfs wants
- Vfs gets info from the new fs i-node and puts it in a v-node
- Makes entry in fd table for process
- When process issues a call (e.g. read), function pointers point to concrete function calls

Virtual File Systems



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

The UNIX V7 File System (1)

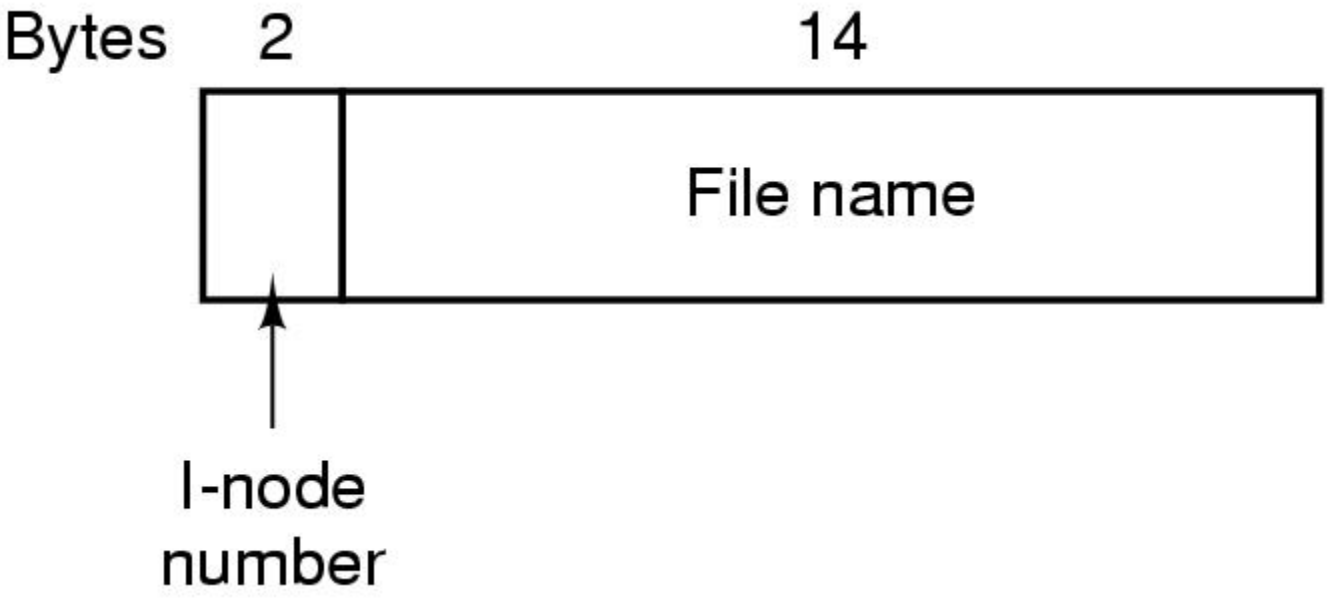


Figure 4-33. A UNIX V7 directory entry.

The UNIX V7 File System (2)

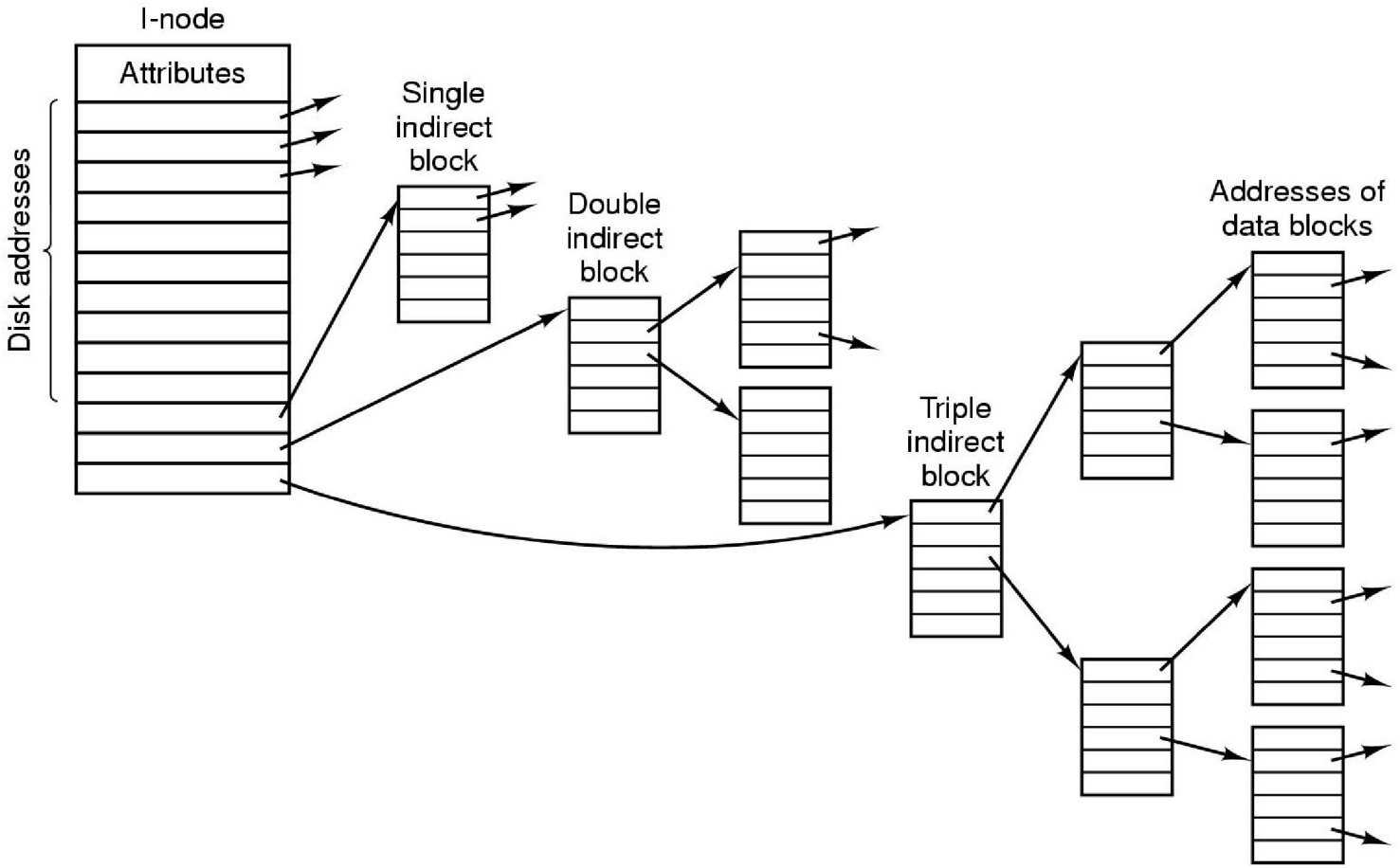


Figure 4-34. A UNIX i-node.

The UNIX V7 File System (3)

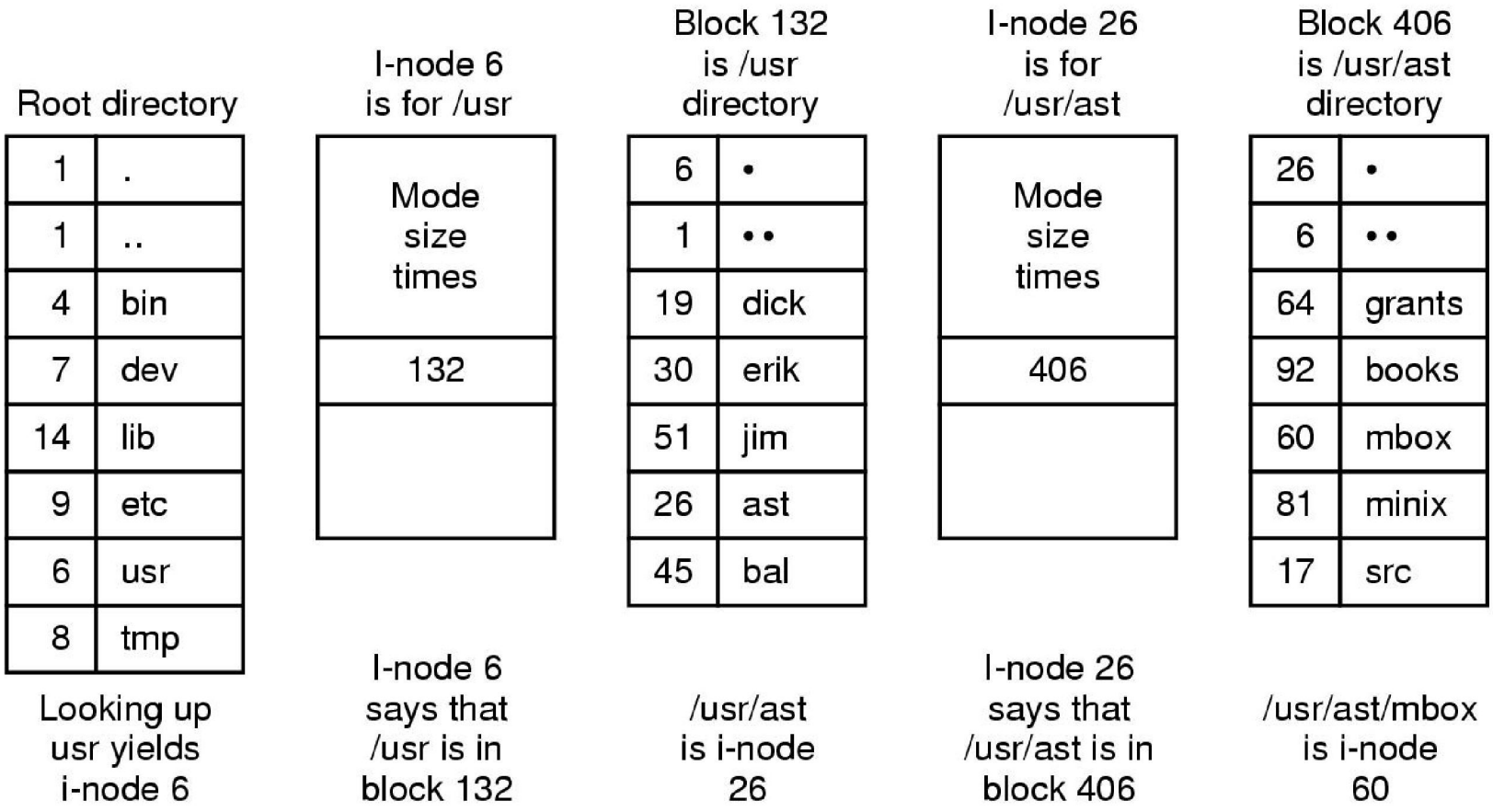


Figure 4-35. The steps in looking up ***/usr/ast/mbox***.

Takeaways