CS 444 Operating Systems

Chapter 4 File Systems

J. Holly DeBlois

September 18, 2024

Long-Term Information Storage

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

Abstraction of a Disk

- Details of disk I/O are in Chapter 5
- Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:
  - Read block k
  - Write block k

Questions about Long-Term Information Storage

- 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

- Created by processes
- A kind of address space to model the disk
- Files must be persistent

Windows File Systems

- FAT-16
- FAT-32
- exFAT, extensible FAT, for flash memory
- NTFS
- ReFS, resilient file system, B+ tree

Capacities of Windows File Systems

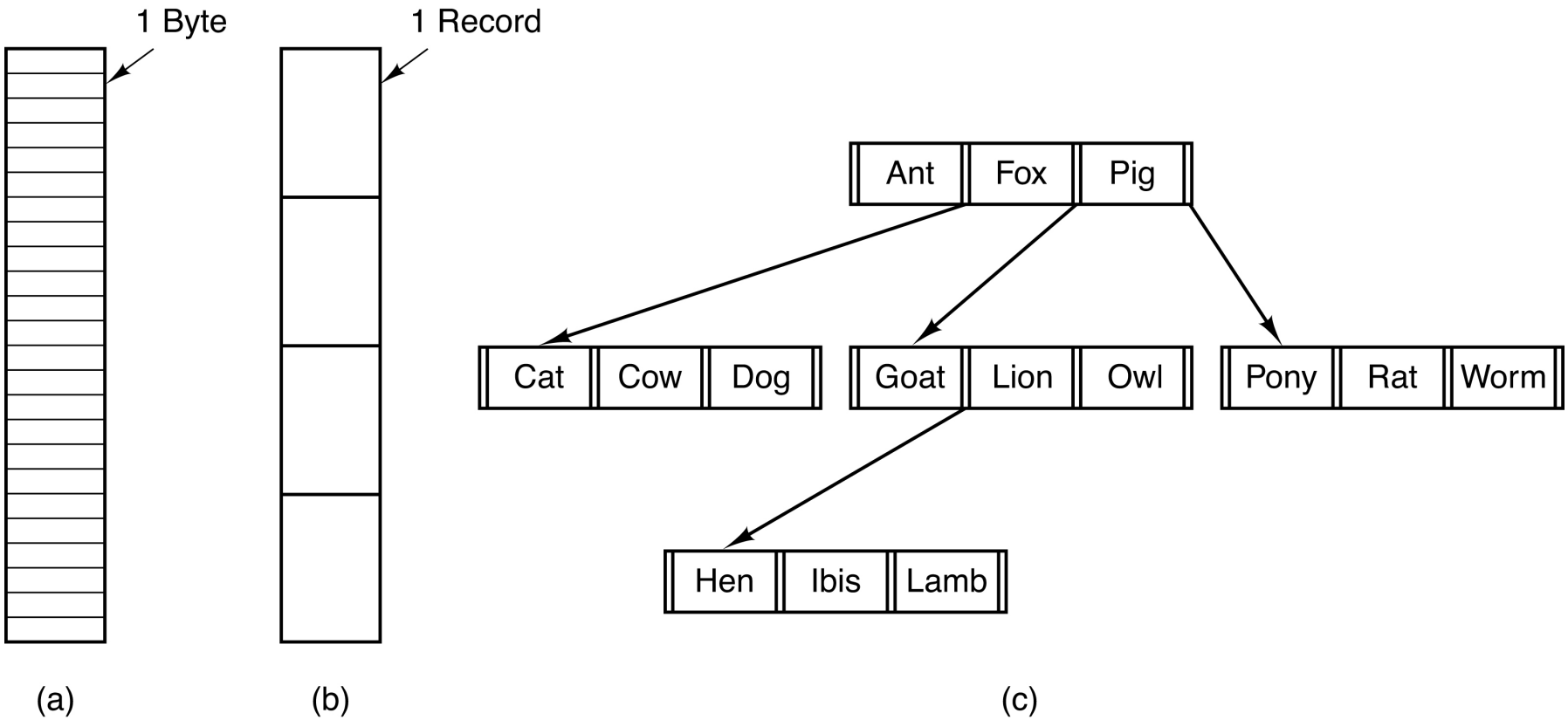
Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

File Extension

- `.c`, `.o`, etc
- In Unix, file extensions have no meanings to the OS
- The command `file` can determine file type without file extension
- Some types of files have a "magic number" stored near the beginning of the file

File Structure

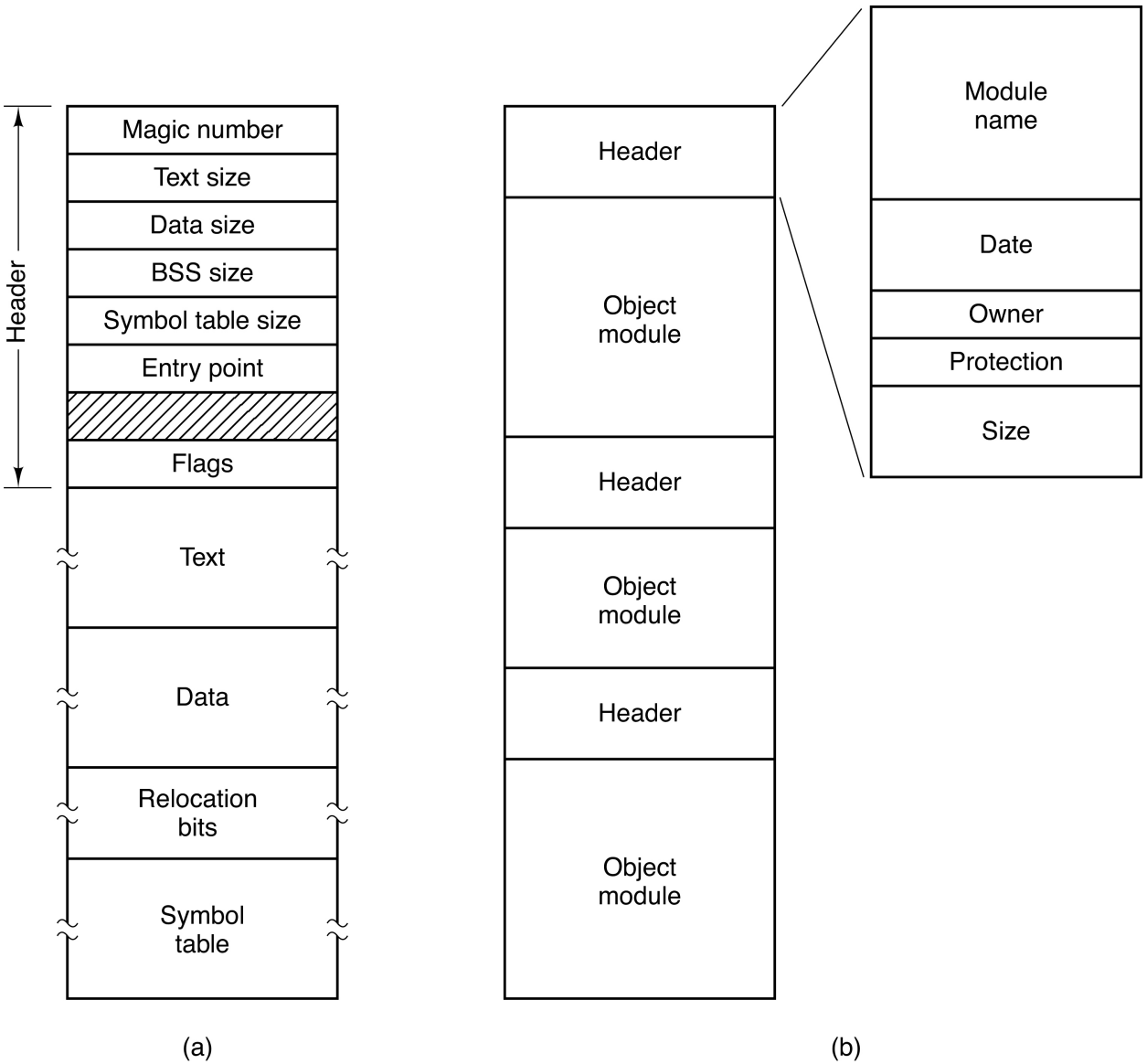
- Byte sequence: Unix, Windows
- Record sequence
- Tree



File Types

- Regular files
  - ASCII and binary files
  - ASCII files end a line by carriage return, line feed, or both
- Directories (folders)
- Character special files: terminals, printers, networks
- Block special files (disks)

File Examples



- An executable
- An archive

File Access

- Sequential access
- Random access

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

File Operations

- Create
- Delete
- Open
- Close
- Read
- Write

- Append
- Seek
- Get attributes
- Set attributes
- Rename

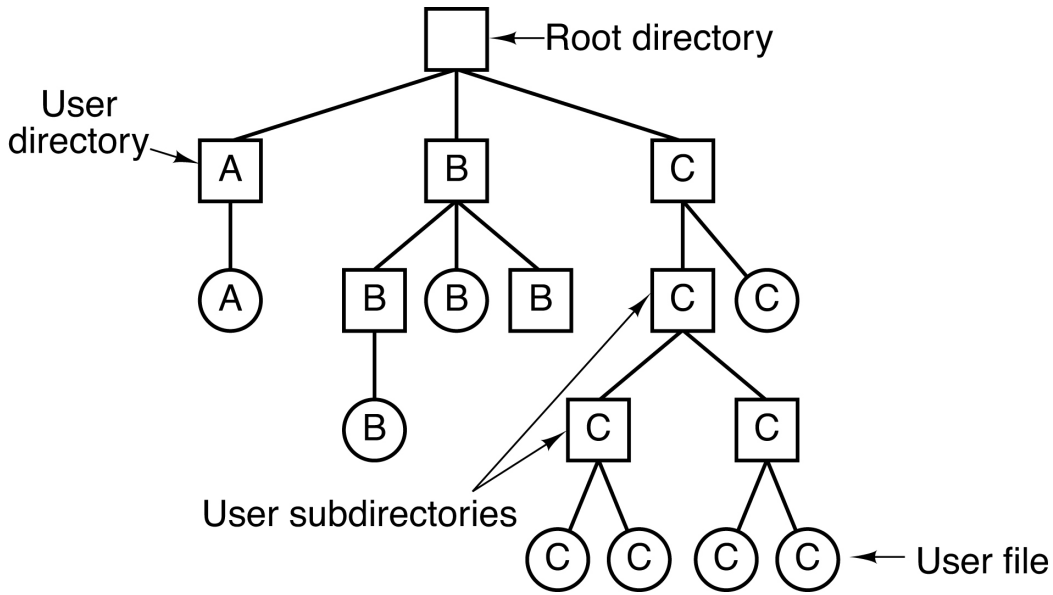
Example of File System Calls

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

- `open()`
- `creat()`
- `read()`
- `write()`
- `close()`

Directory Structure

- Single-level directory systems, flat
- Hierarchical directory systems



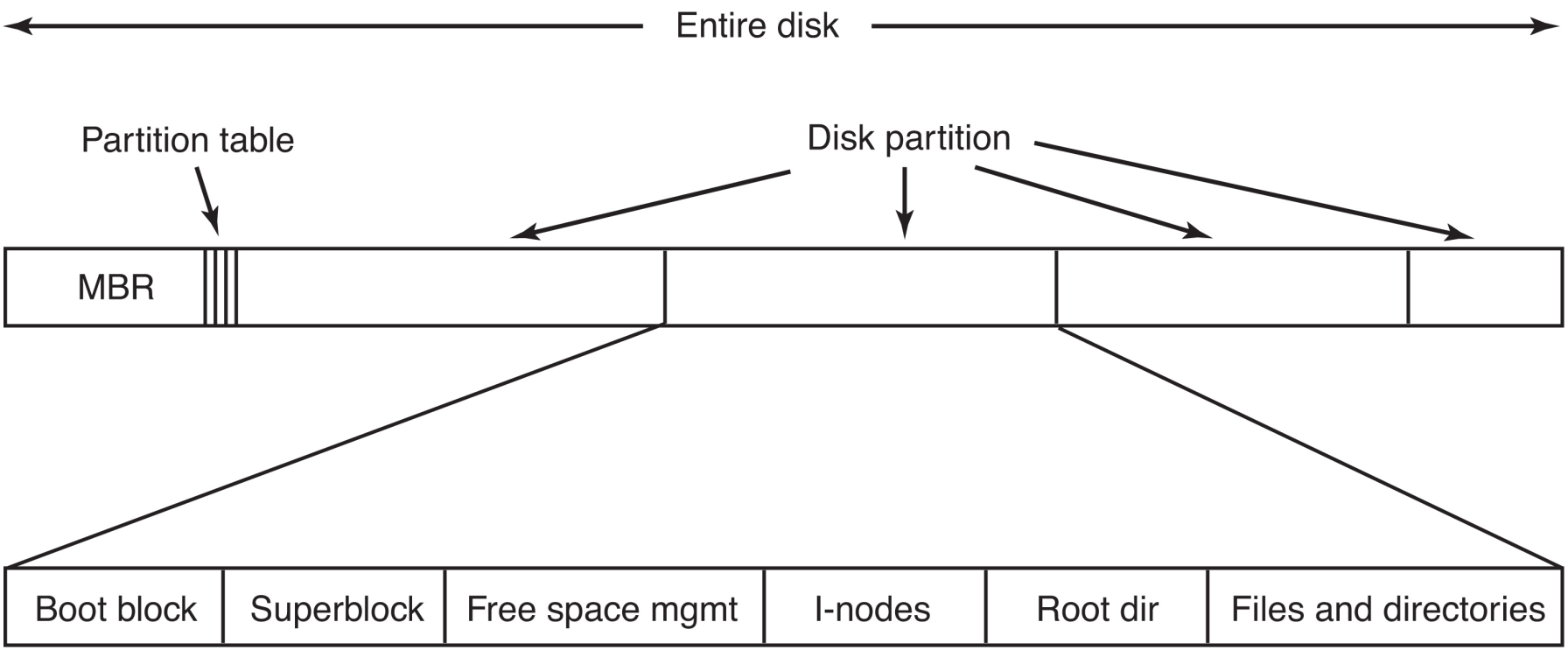
Path Names

- Absolute path names
  - Delimiters: Multics >, Unix /, Windows \
  - Original DOS is flat — no directories — and uses slashes for command options
  - Unix uses dashes for command options
- Relative path names
  - Working directory, current directory
  - Dot .
  - Parent directory, dot dot ..

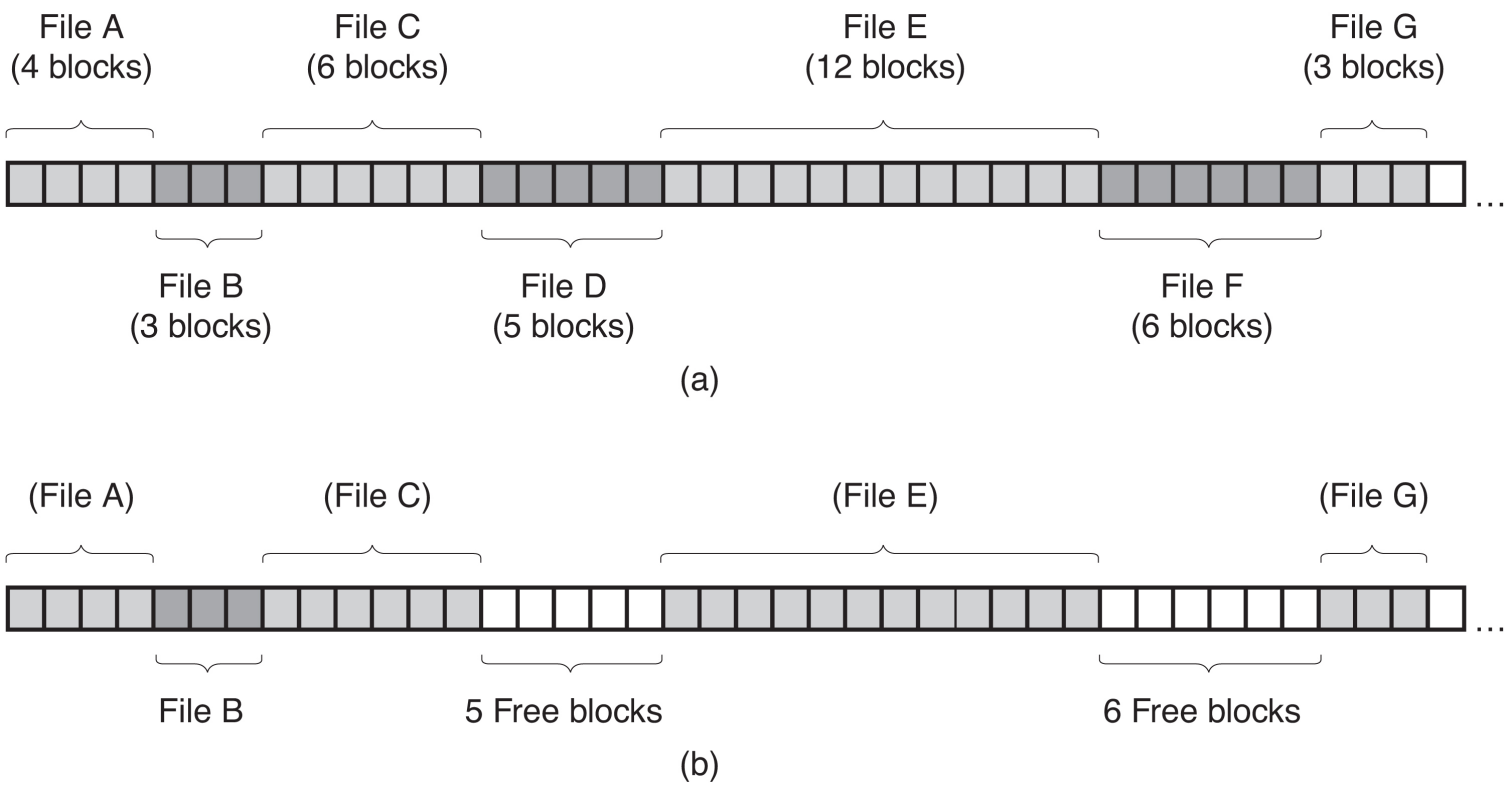
Directory Operations

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
  - Hard link: faster
  - Symbolic link may cross disk and partition boundaries
- Unlink

File System Layout

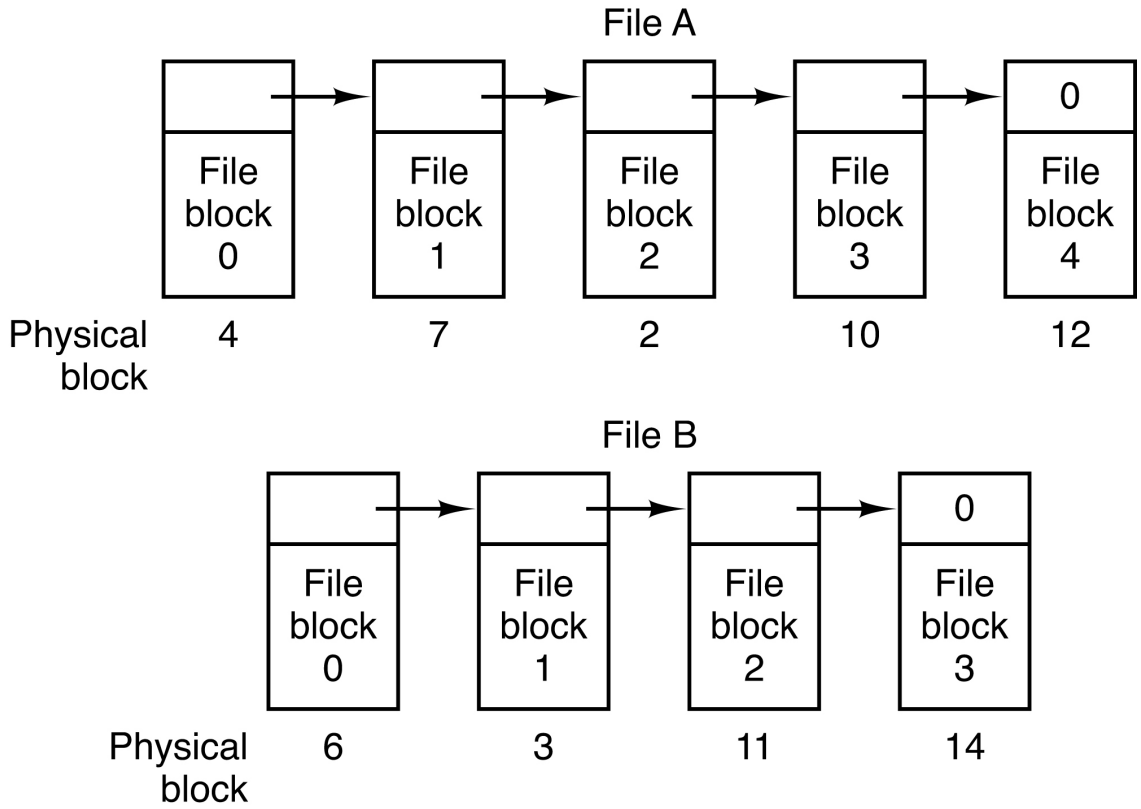


Contiguous Allocation



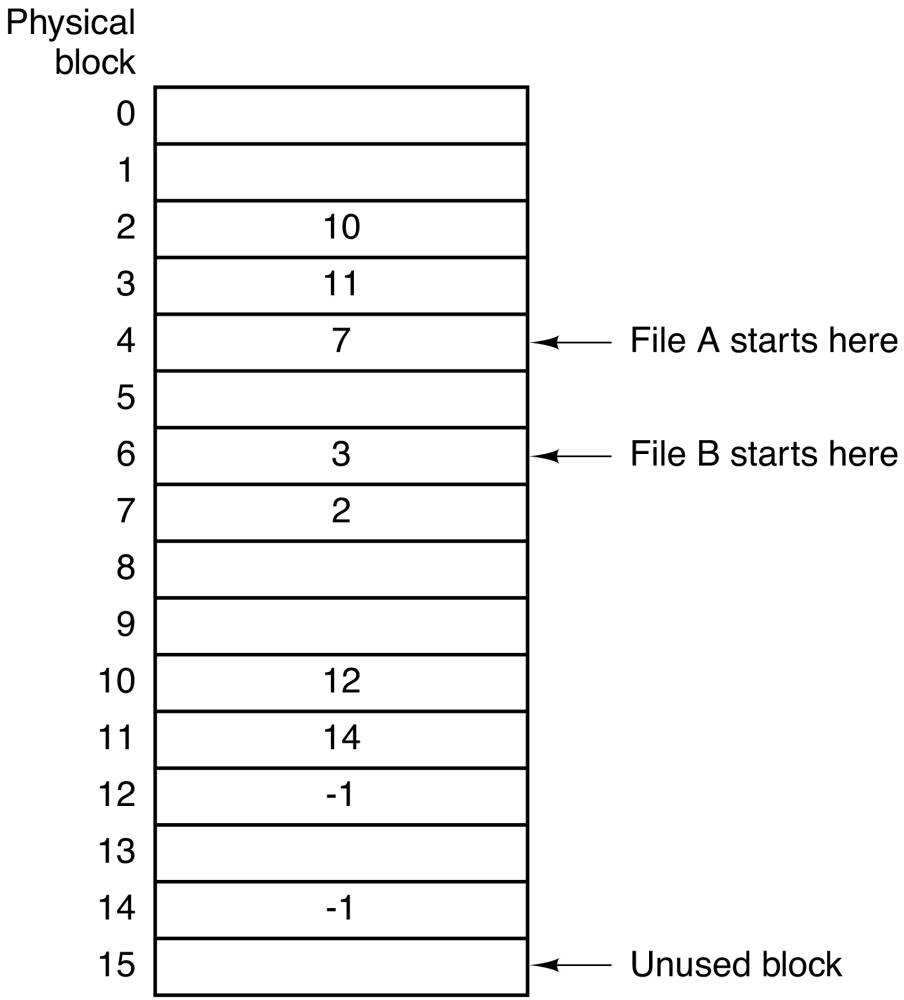
- Disadvantage: external fragmentation
- Advantages
  - Simplicity
  - Fast read

Linked-List Allocation



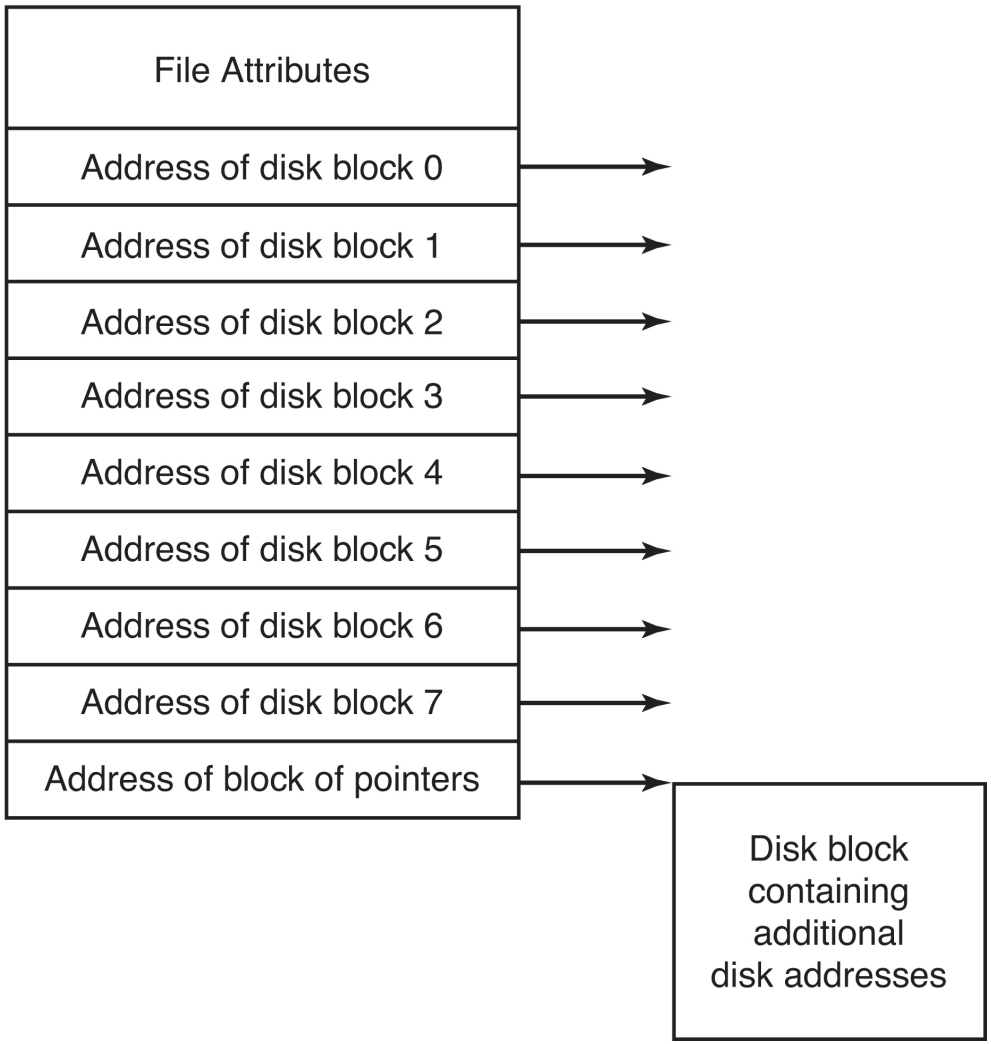
- No external fragmentation
- Slow read

Linked List by FAT



- File allocation table in RAM
- No external fragmentation
- Fast read
- Impractical for large disks

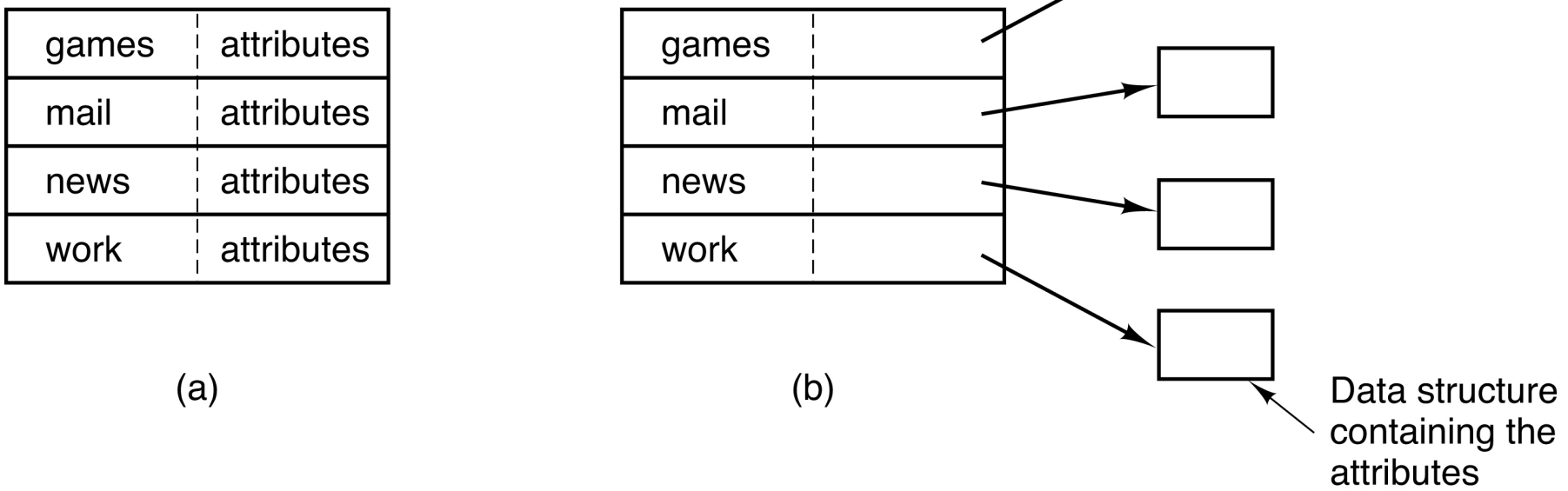
Linked List by I-Nodes



- Unix file systems use i-nodes

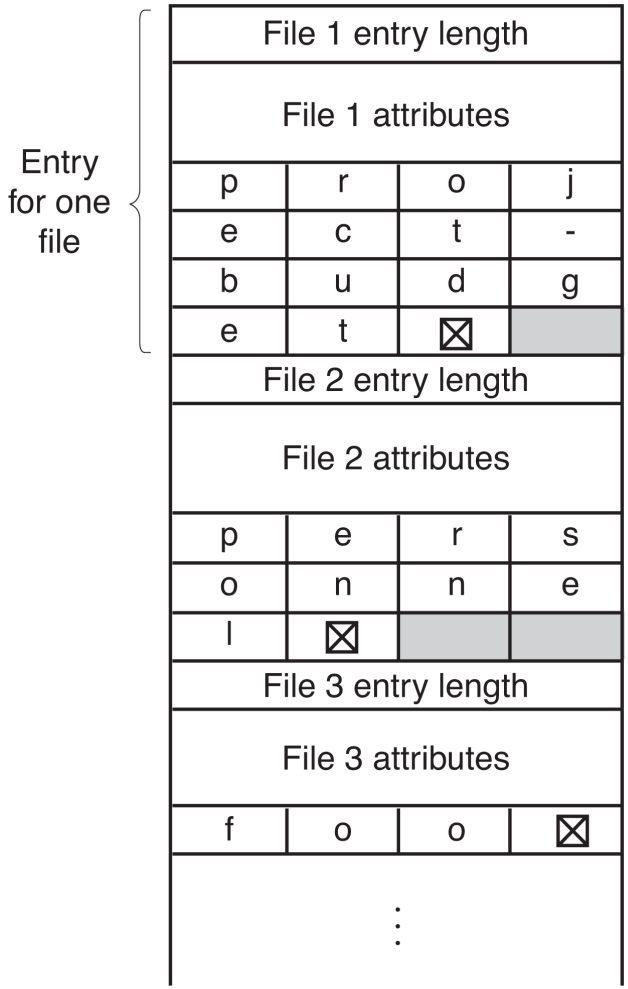
Implementing Directories

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



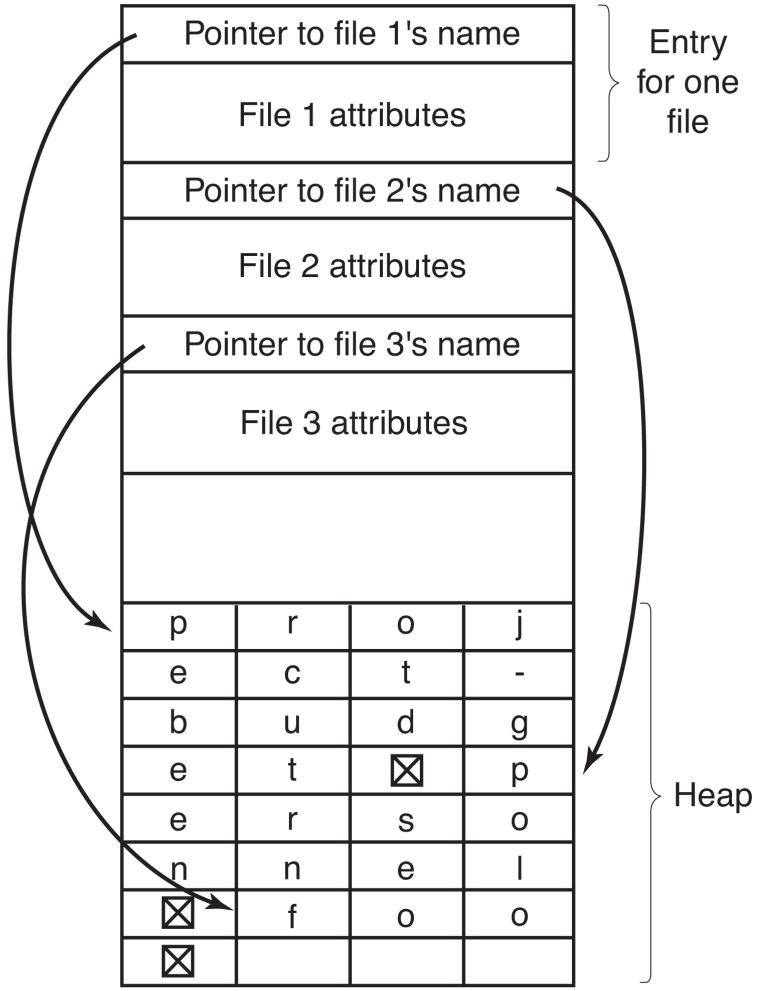
Accommodating Long File Names

- File names in the file entries, which have varying sizes



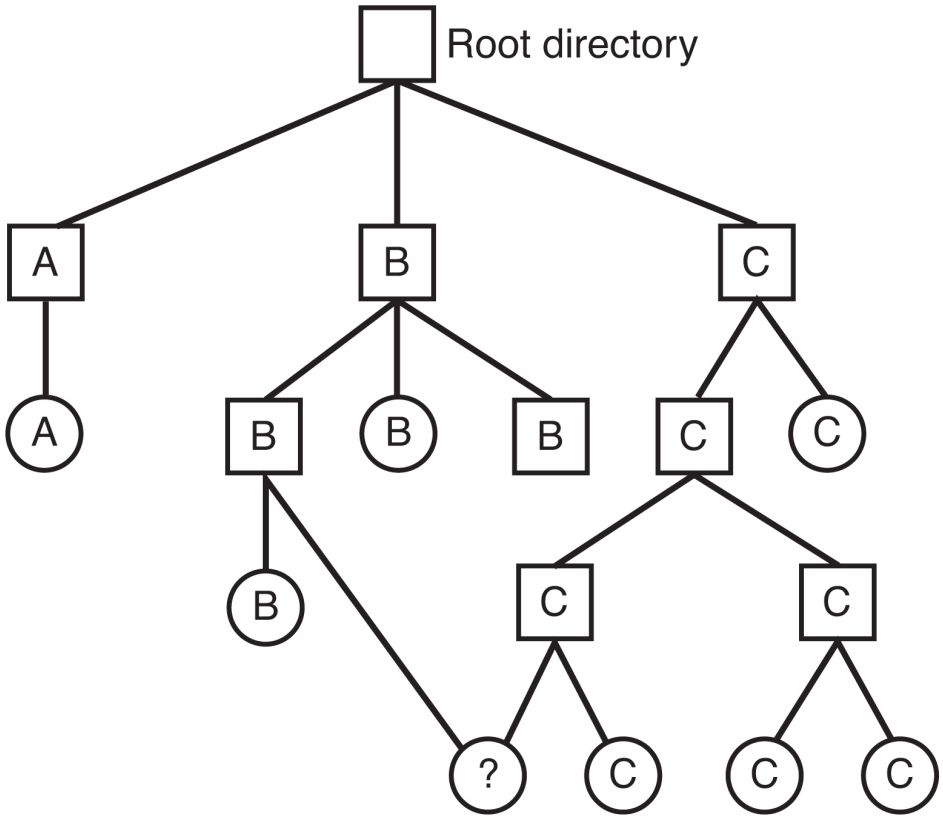
(a)

- Fixed-size entries, file names stored separately



(b)

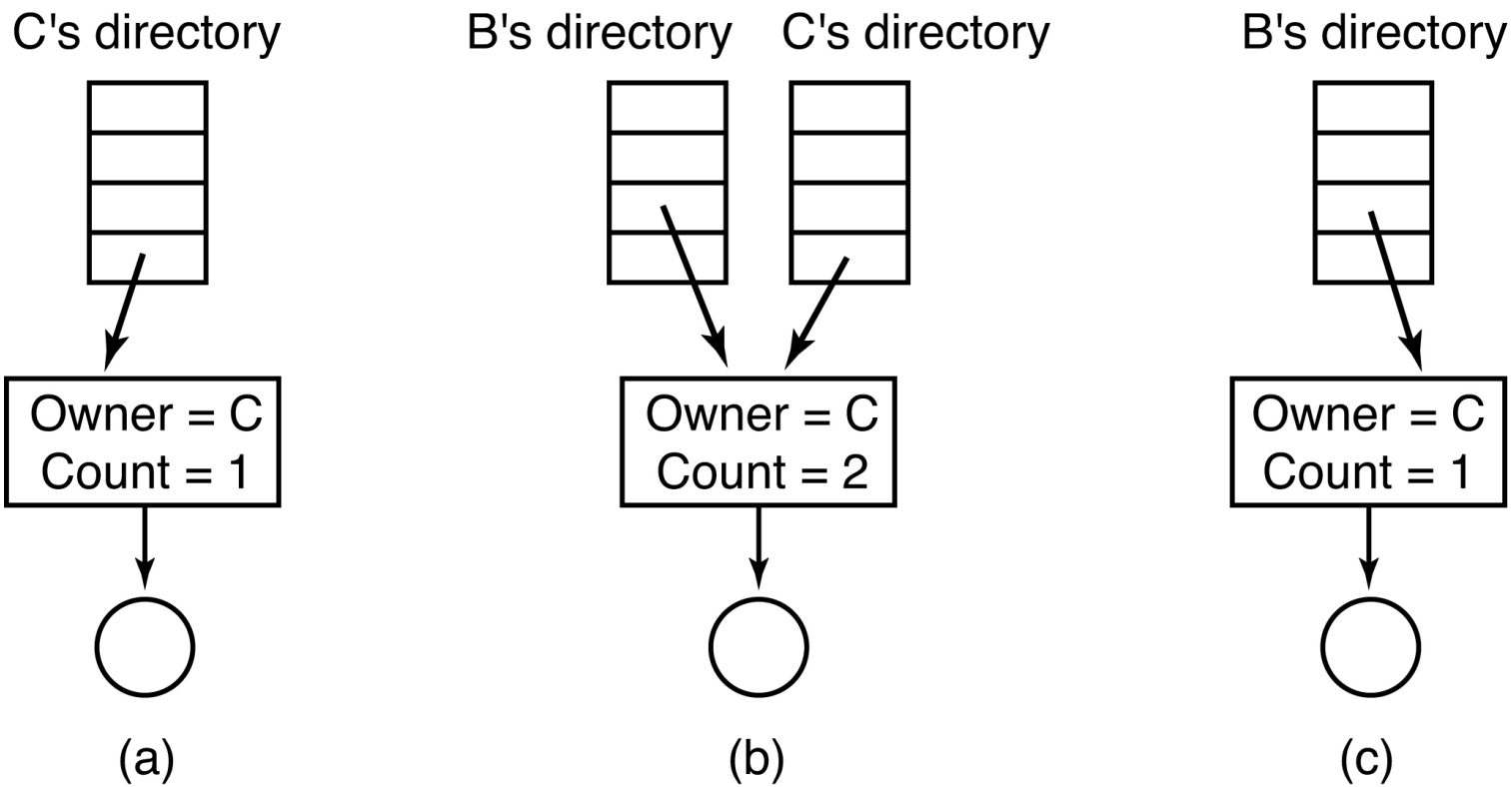
Shared File



Shared file

Ownership of Hard Linked Files

- Prior to linking
- After linking
- After the original owner removes the file

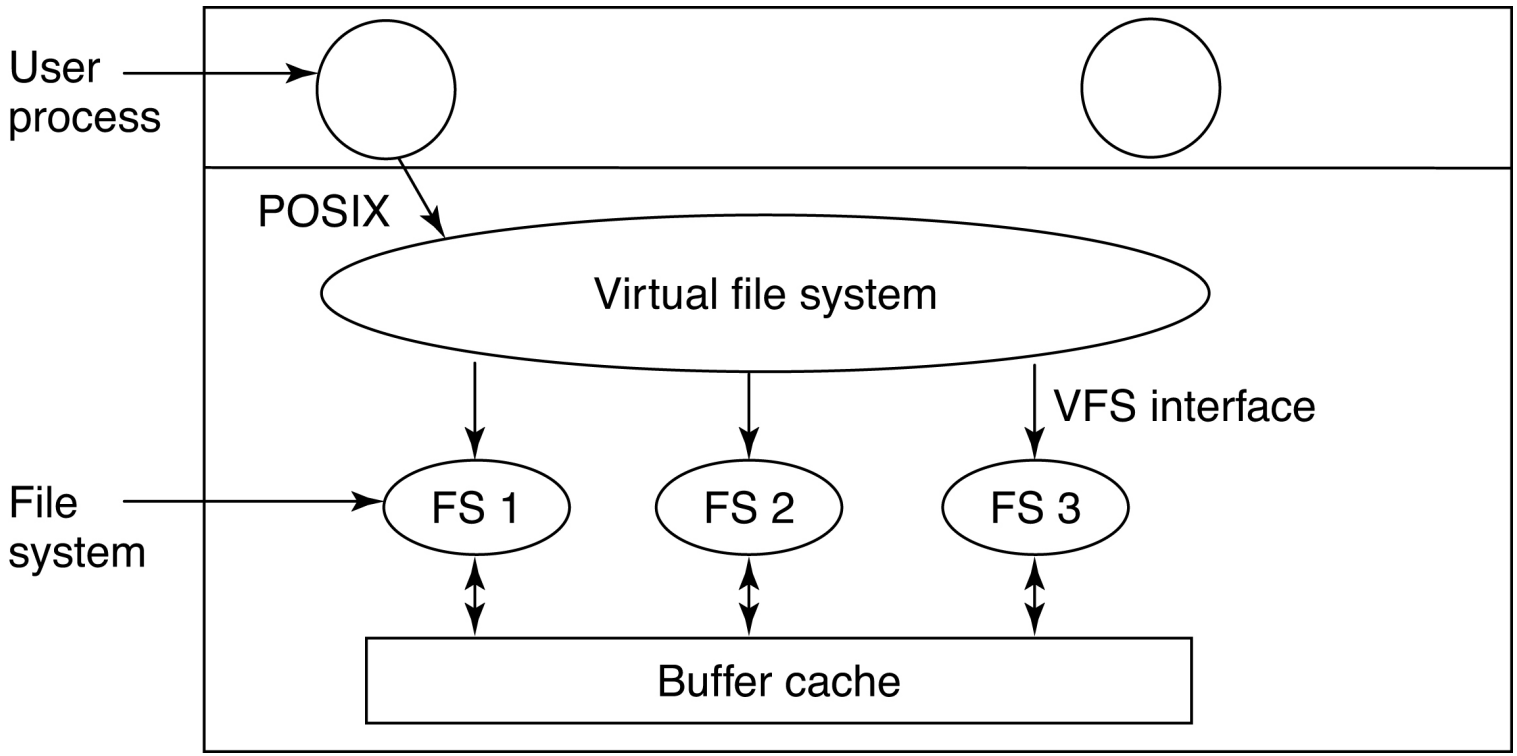


Journaling File Systems

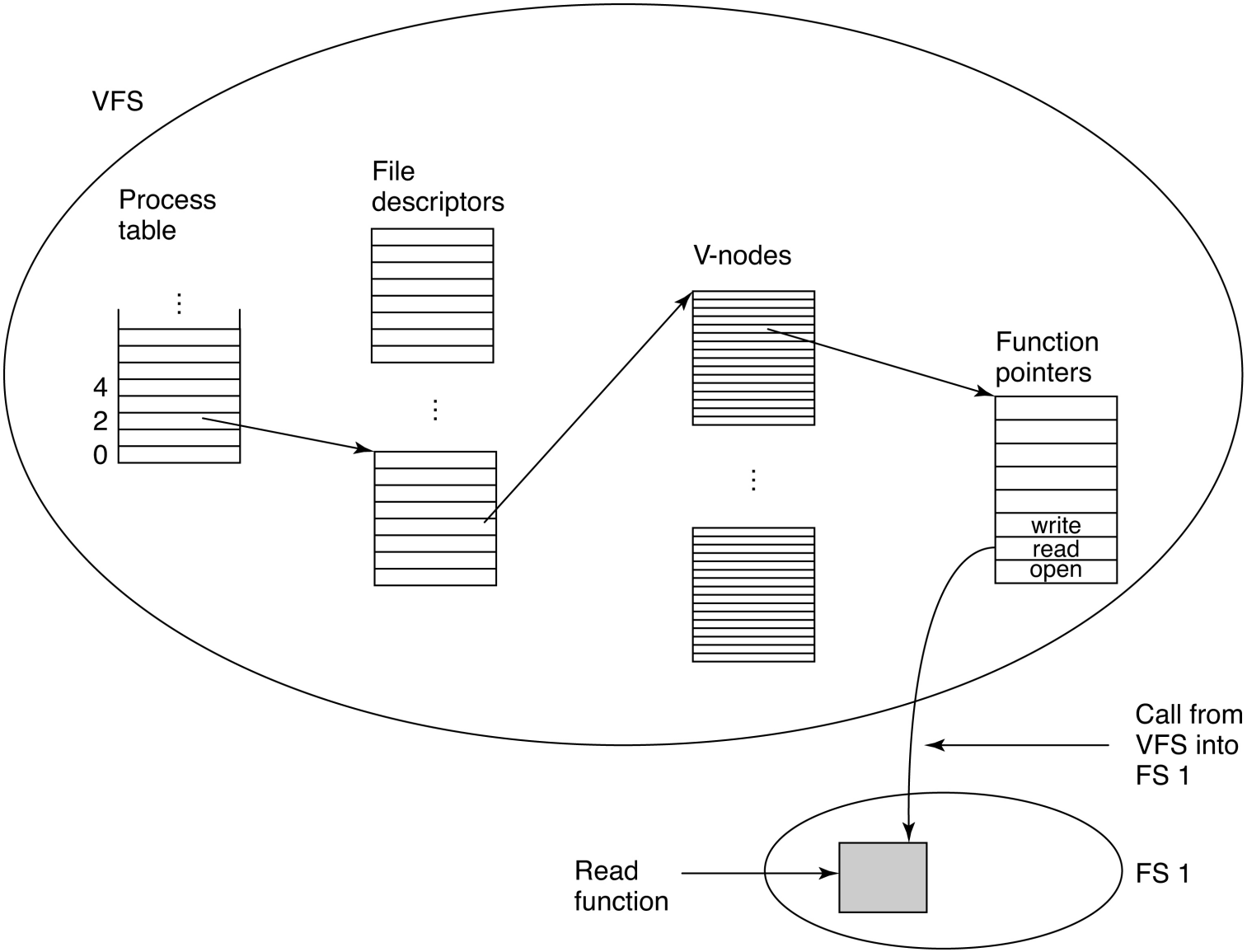
- Steps to remove a file in UNIX:
  1. Remove file from its directory
  2. Release i-node to the pool of free i-nodes
  3. Return all disk blocks to pool of free disk blocks
- Write these steps in a journal before performing them
- All steps must be idempotent

Virtual File Systems

- Unix integrates all file systems into one VFS
  - POSIX interface
  - VFS interface
- Windows uses drive letters



Data Structures for VFS



File System Management

- Disk space management
- File system backups
- File system consistency
- Performance
- Defragmentation

File Size Distributions

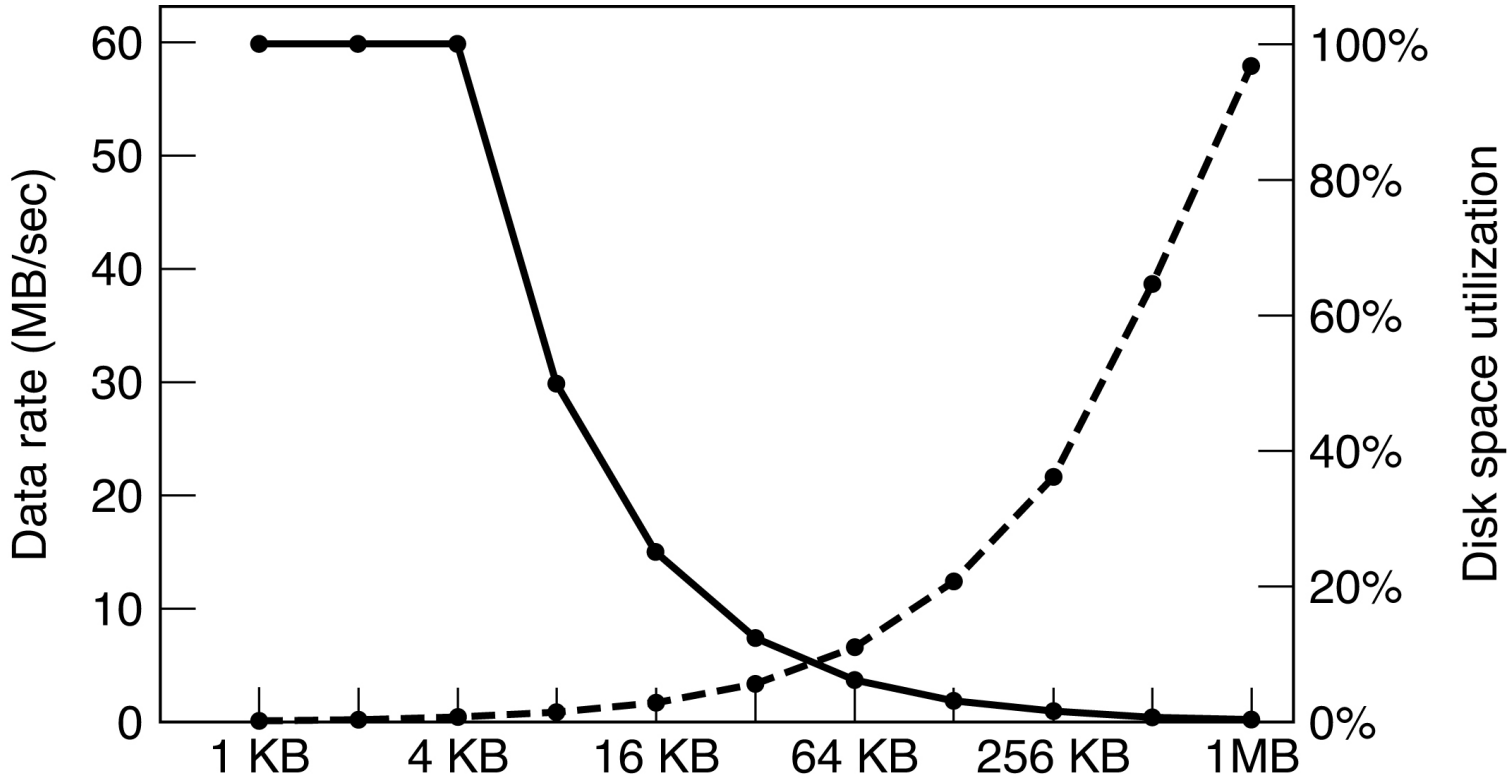
- Percentage of files smaller than a given size in bytes

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

Disk Block Size Tradeoff

- The dashed curve, left-hand scale, gives the data rate of a disk
- The solid curve, right-hand scale, gives the disk space efficiency

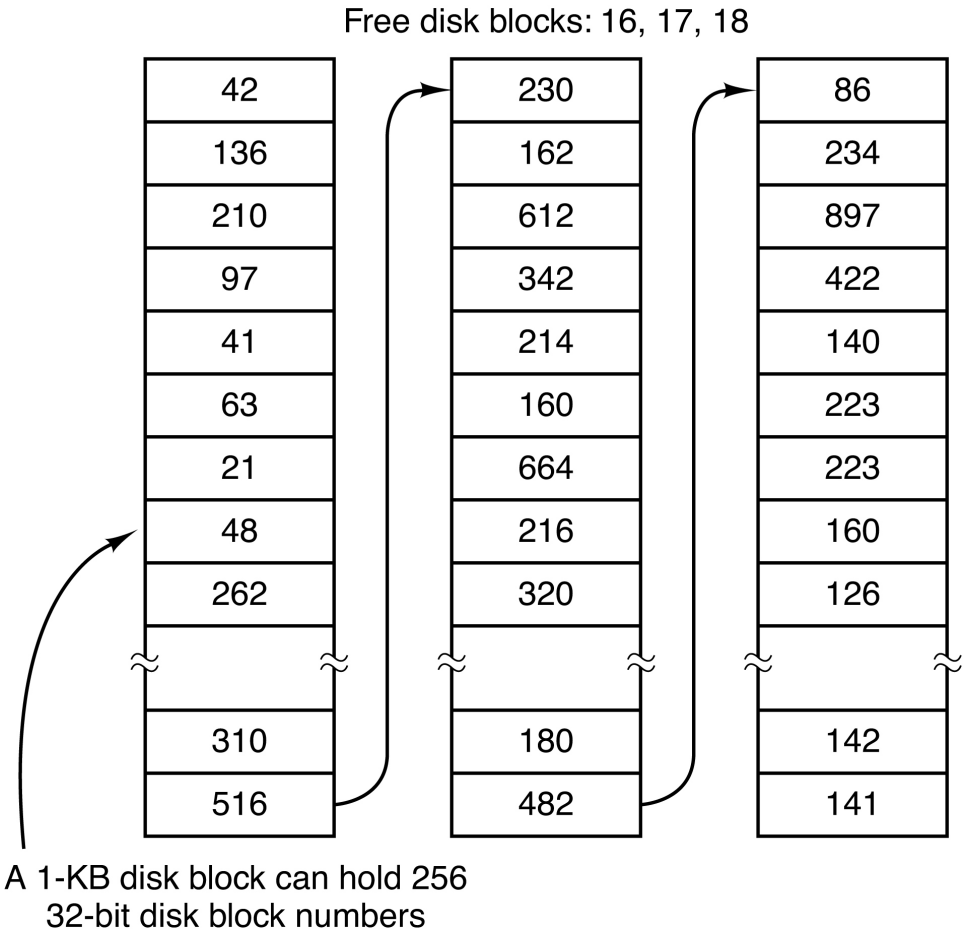


Two Ways to Keep Track of Free Blocks

- Linked list
  - Use the free blocks to maintain the list of free blocks
  - Take more space, 4B per free block
- Bitmap
  - Take less space, 1 bit per free block

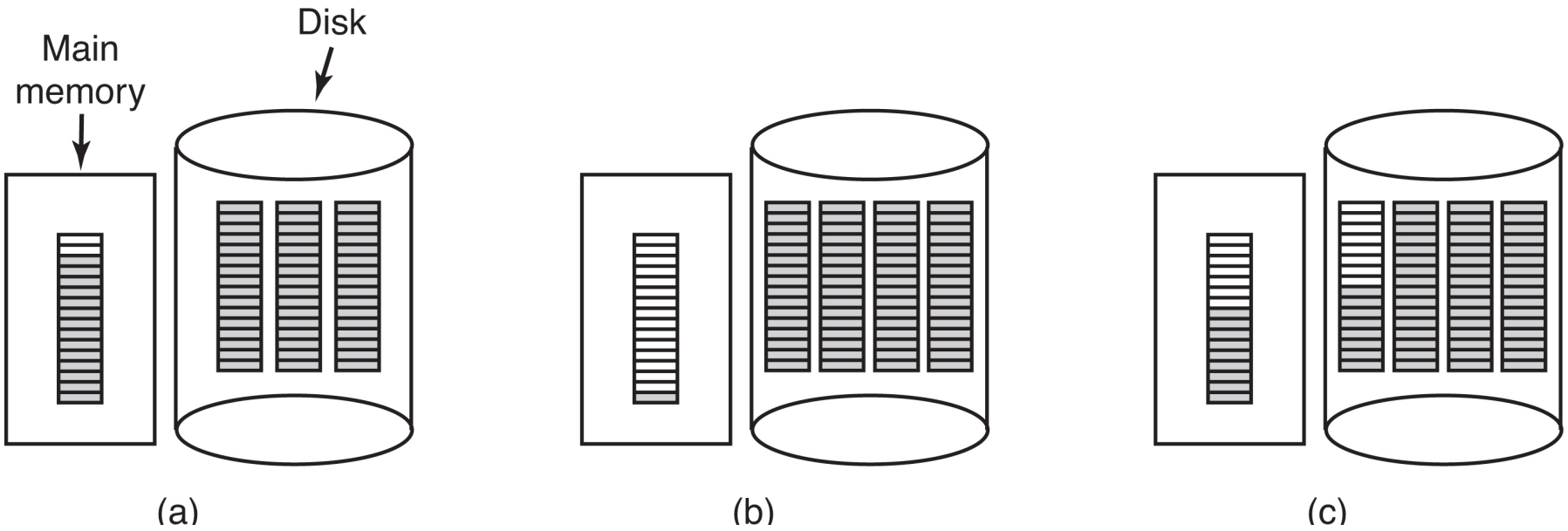
Keeping Track of Free Blocks

- A linked list
- A bitmap



Keep a Block of Free Disk Blocks in RAM

- An almost full block of pointers to free disk blocks
- After freeing 3 disk blocks
- Alternative: keep a half block of free disk blocks in RAM

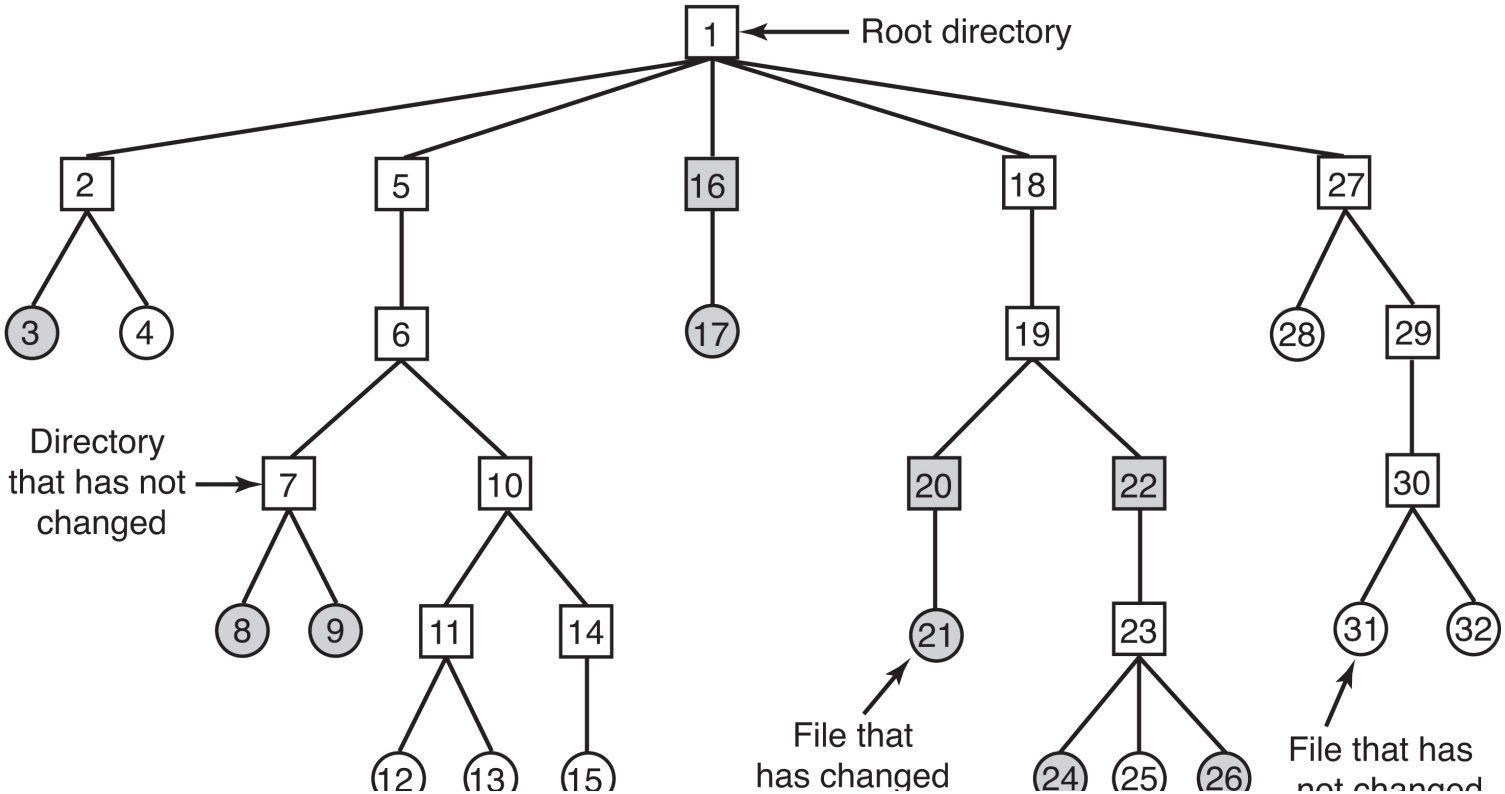


File System Backups

- Backups to tape are generally made to handle one of two potential problems:
  - Recover from disaster
  - Recover from stupidity
- Incremental dumps
  - Make a complete dump periodically
  - Make a daily dump of modified files since the last full dump

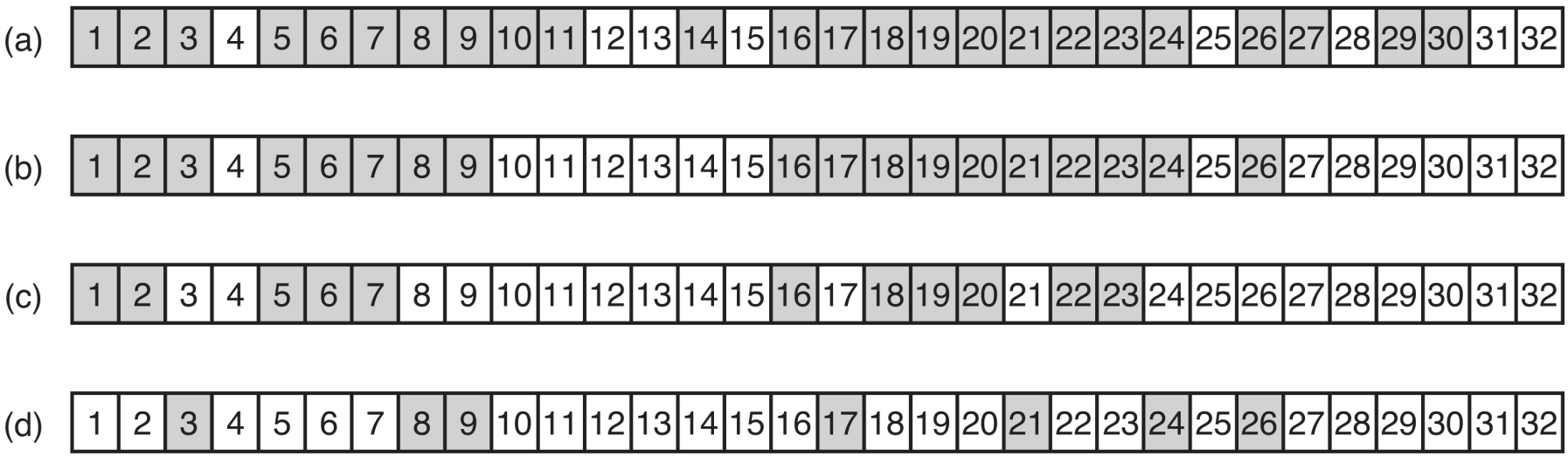
Logical Dump

- 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



Bitmaps for Logical Dump

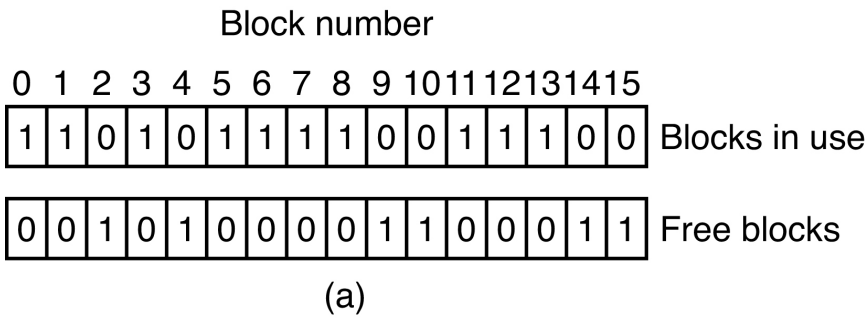
- Mark all directories and all modified files
- Unmark directories with no modified descendants
- Write all marked directories
- Write all marked files

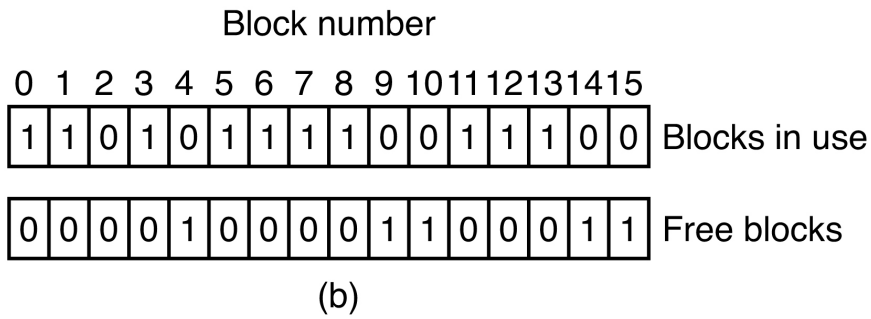


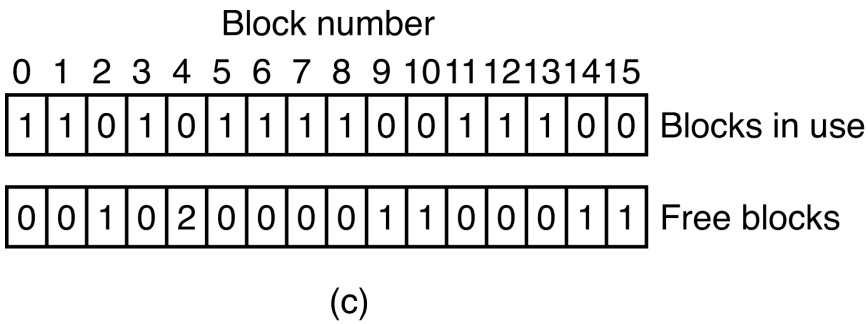
File System Consistency

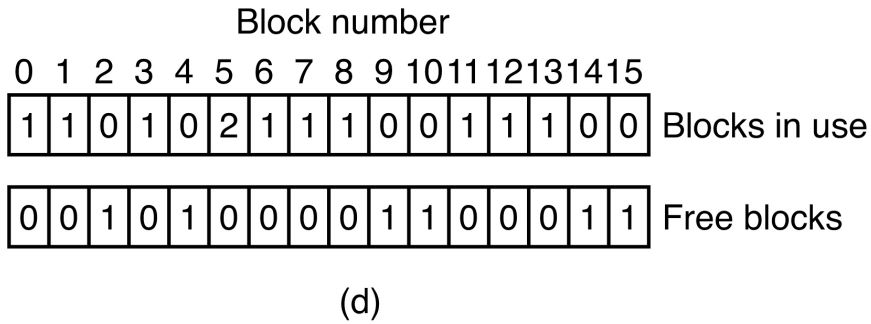
- Unix utility: `fsck`
- Get blocks in use from i-nodes
- Get free blocks from free block list/map

- (a) Consistent
- (b) Missing block
- (c) Duplicate block in free list
- (d) Duplicate data block









File System Performance

- Disk cache, modified LRU
- Read ahead for sequentially accessed files
- Reduce disk-arm motion
- Defragment
  - Windows: defrag HDD regularly
  - Linux: no need

Disk Cache with Modified LRU

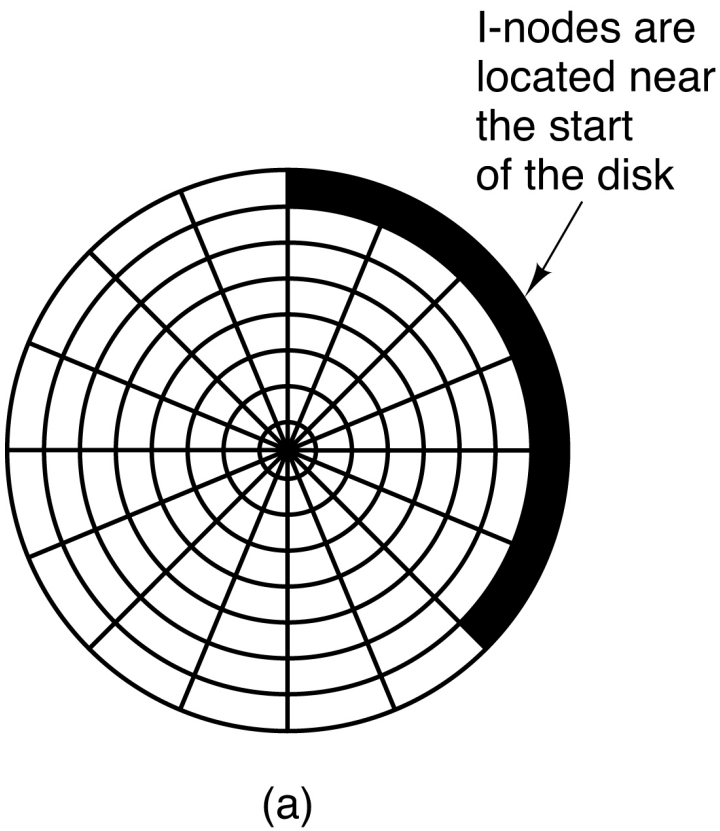
- Some blocks rarely referenced two times within a short interval
- Leads to a modified LRU scheme, taking two factors into account:
  - Is the block likely to be needed again soon?
  - Is the block essential to the consistency of the file system?
    i-node blocks, indirect blocks, directory blocks
- Newest and best: IBM's adaptive replacement cache, ARC

Write Modified Disk Cache to Disk

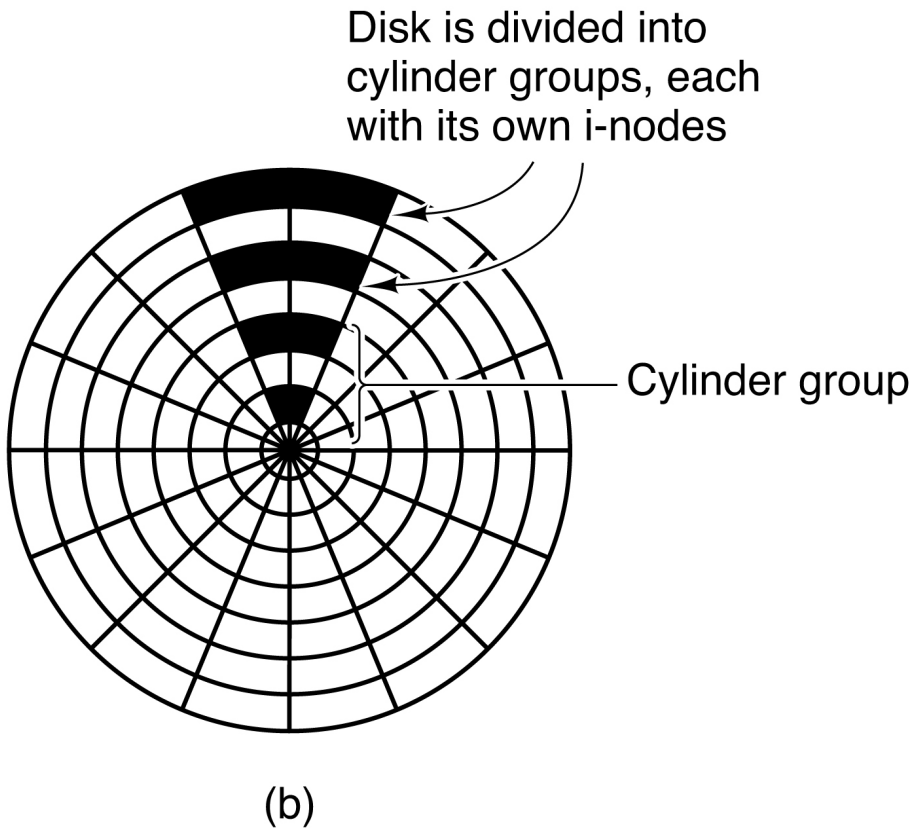
- Unix
  - A system call sync
  - A daemon process update calls sync every 30 sec
- Windows
  - In the past, write-through cache
  - Now, just like Unix

Reducing Disk Arm Motion

- I-nodes placed at the start of the disk

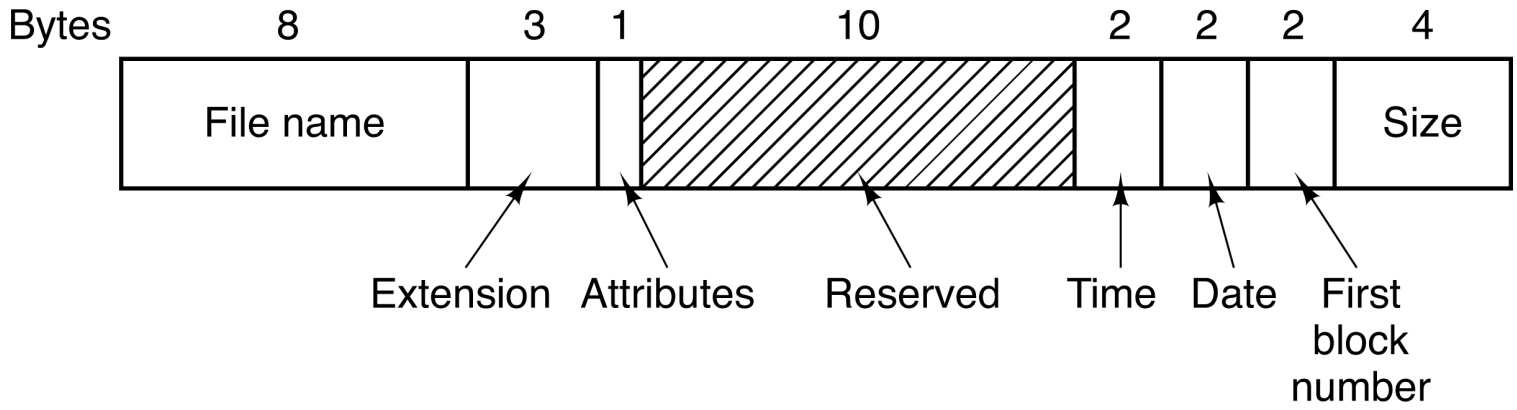


- Disk divided into cylinder groups, each with its own blocks and i-nodes

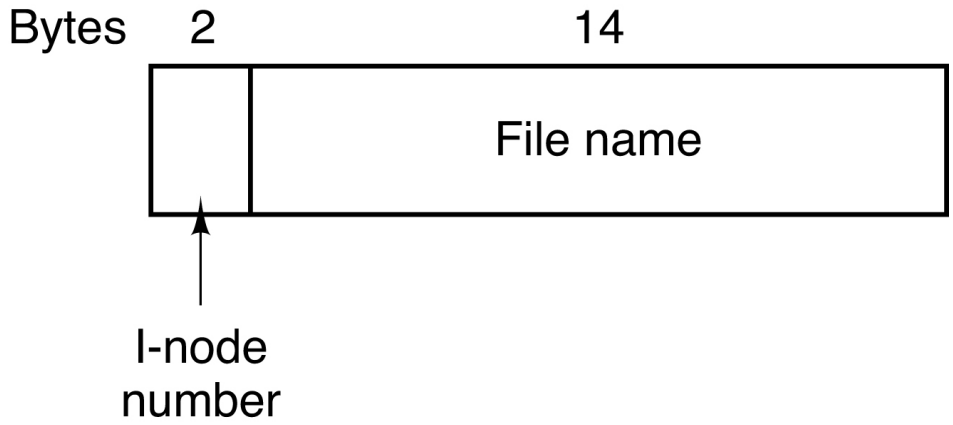


Directory Entries

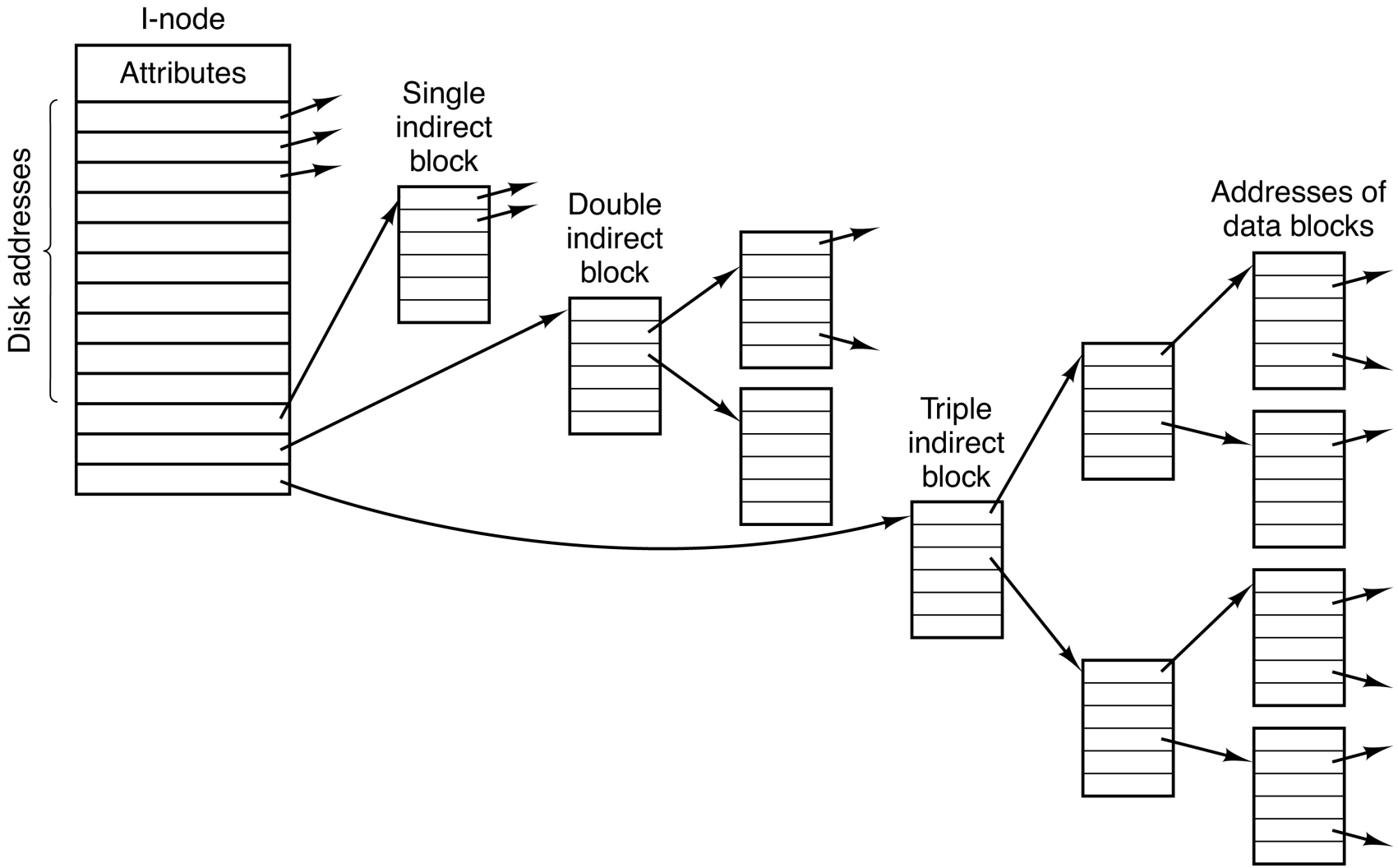
- An MS-DOS directory entry



- A Unix V7 directory entry



A Unix I-Node


I-node
Attributes
Disk addresses
Single indirect block
Double indirect block
Triple indirect block
Addresses of data blocks

Following an Absolute Path

- Looking up `/usr/ast/mbox`