Chapter 4

FILE SYSTEMS

Files
Directories
File system implementation
Example file systems

Introduction

- Three problems for accessing information
  - Must store large amounts of data
  - Information stored must survive the termination of the process using it
  - Multiple processes must be able to access the information concurrently
- Solution:
  - Make the information independent of any process
  - Using magnetic disks or solid-state drives

Introduction

- Think of a disk as
  - a linear sequence of fixed-size blocks
  - supporting two operations:
    - Read block k
    - Write block k
- These are very inconvenient operations
  - 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?

Introduction

- File: a new abstraction to solve this problem
  - logical units of information created by processes
  - a kind of address space
    - used to model the disk instead of modeling the RAM
  - Processes can read existing files and create new ones
  - They are Persistent
    - should disappear only when its owner explicitly removes it

Introduction

- File: a new abstraction to solve this problem
  - managed by the file system
    - part of the operating system dealing with files
  - Important aspects from user's standpoint
    - what constitutes a file
    - how files are named and protected
    - what operations are allowed on files
    - and so on
  - Important details for designers:
    - whether linked lists or bitmaps are used to manage free storage
    - how many sectors there are in a logical disk block
    - and so on

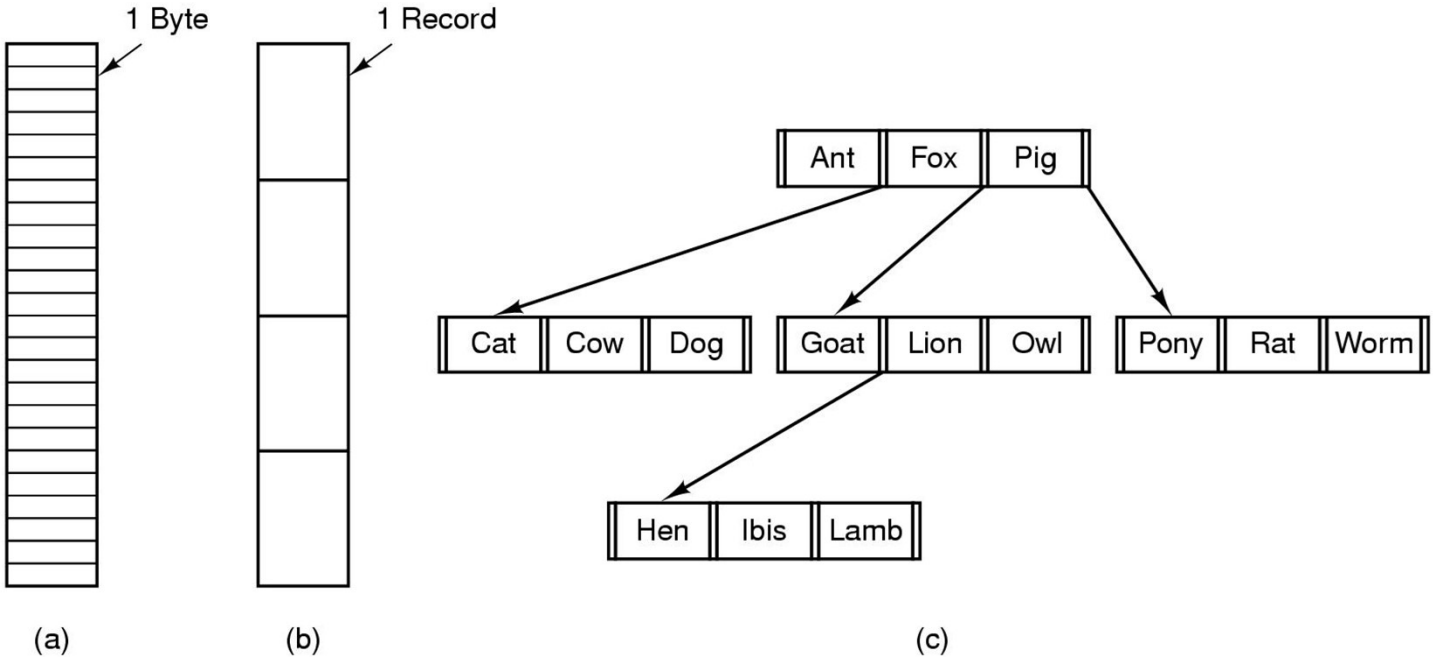
Files

- File naming
  - Many operating systems support two-part names

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

Files

- File Structure
  - byte sequence
  - record sequence
  - tree

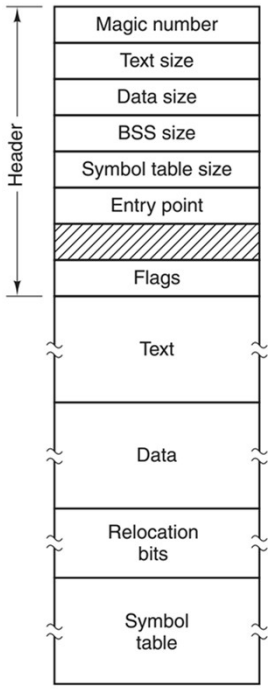


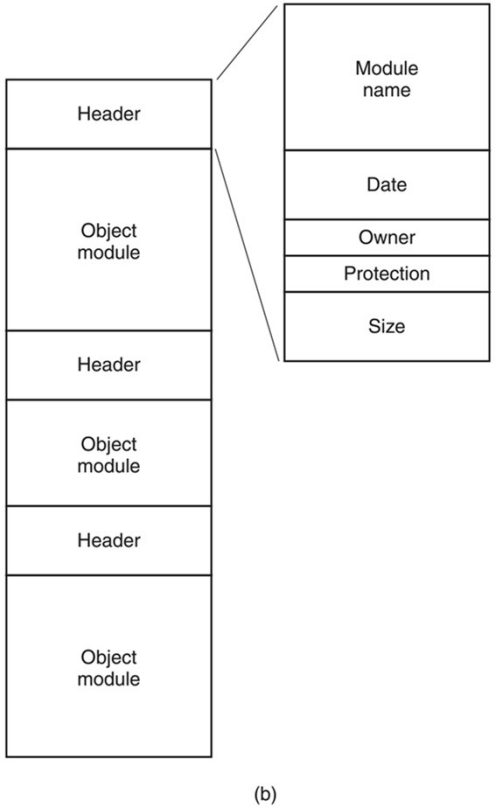
Files

- File Types
  - Regular files
    - contain user information
  - Directories
    - system files for maintaining the structure of the file system
  - Character special files
    - related to input/output
    - used to model serial I/O devices, such as terminals, printers, and networks
  - Block special files
    - used to model disks

Files

- File Types
  - We are interested in Regular files
    - ASCII files
      - consist of lines of text
    - binary files
      - they are not ASCII files





Files

- 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 …
      - every read operation gives the position in the file to start reading at
      - A seek to set the current position; read sequentially from the now-current position

File Attributes

- Possible file attributes

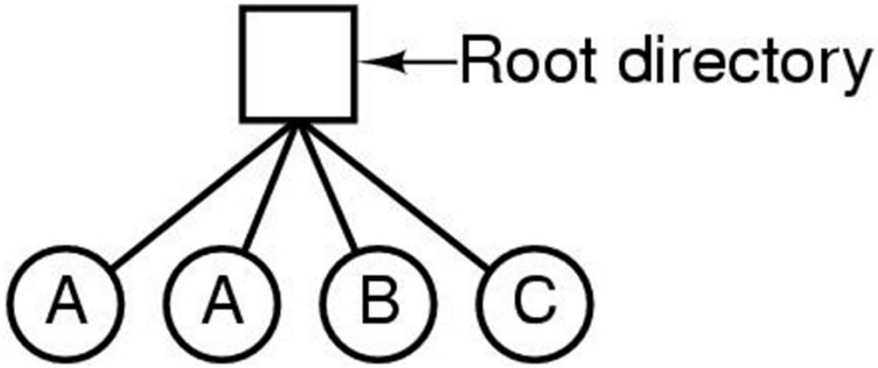
Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 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

File Operations

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get attributes
- Set Attributes
- Rename

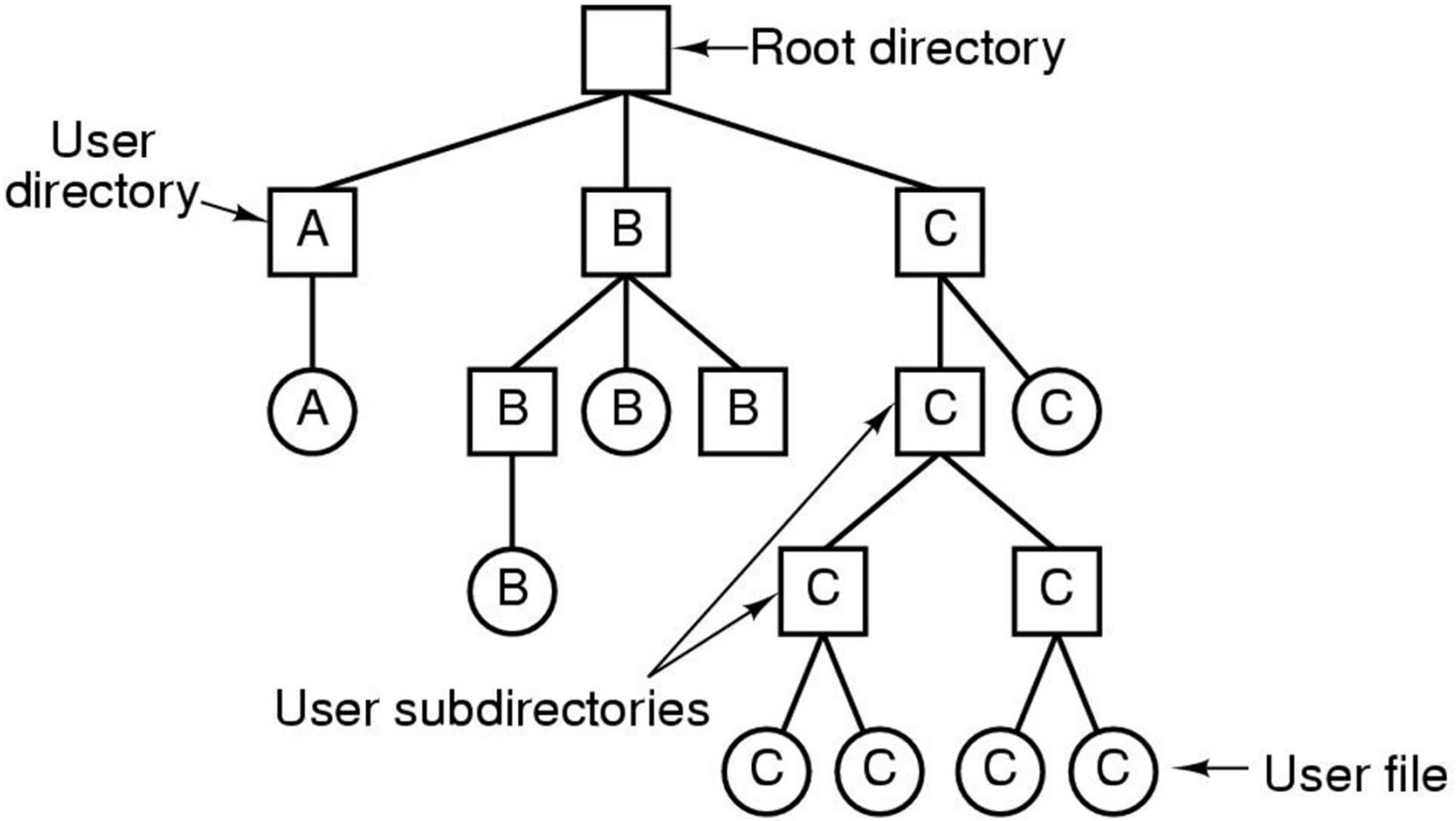
Directories

- single level directory systems
  - Was common on early personal computers
  - Advantage: simplicity
  - still used on simple embedded devices
    - digital cameras
    - some portable music players



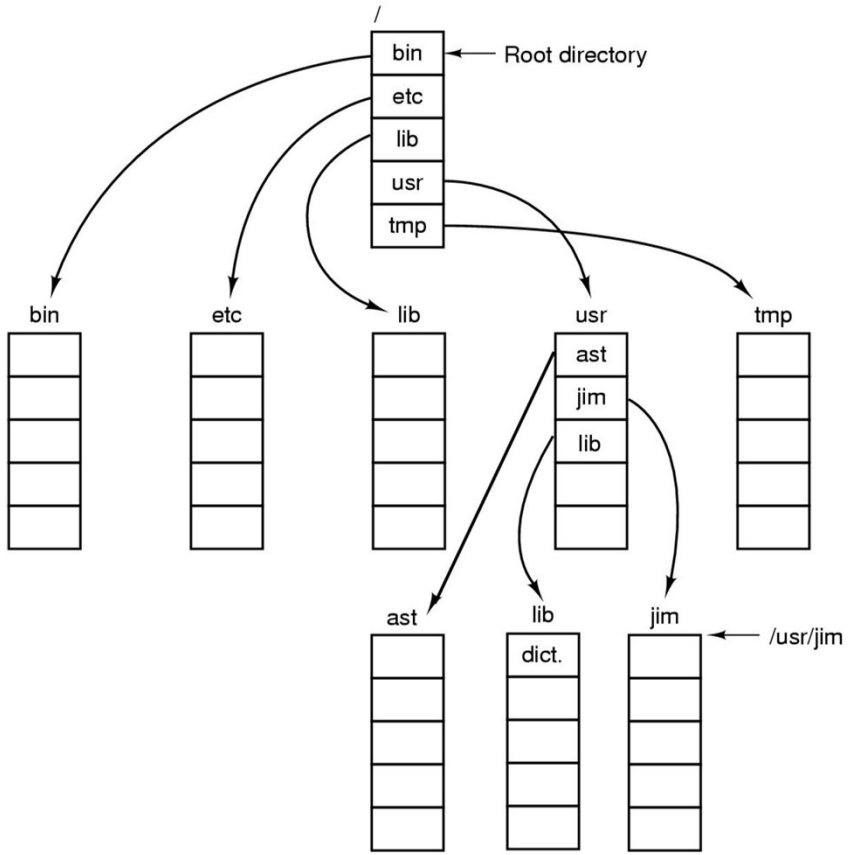
Directories

- hierarchical directory system



Directories

- Path Names
  - Absolute path names
  - Relative path names

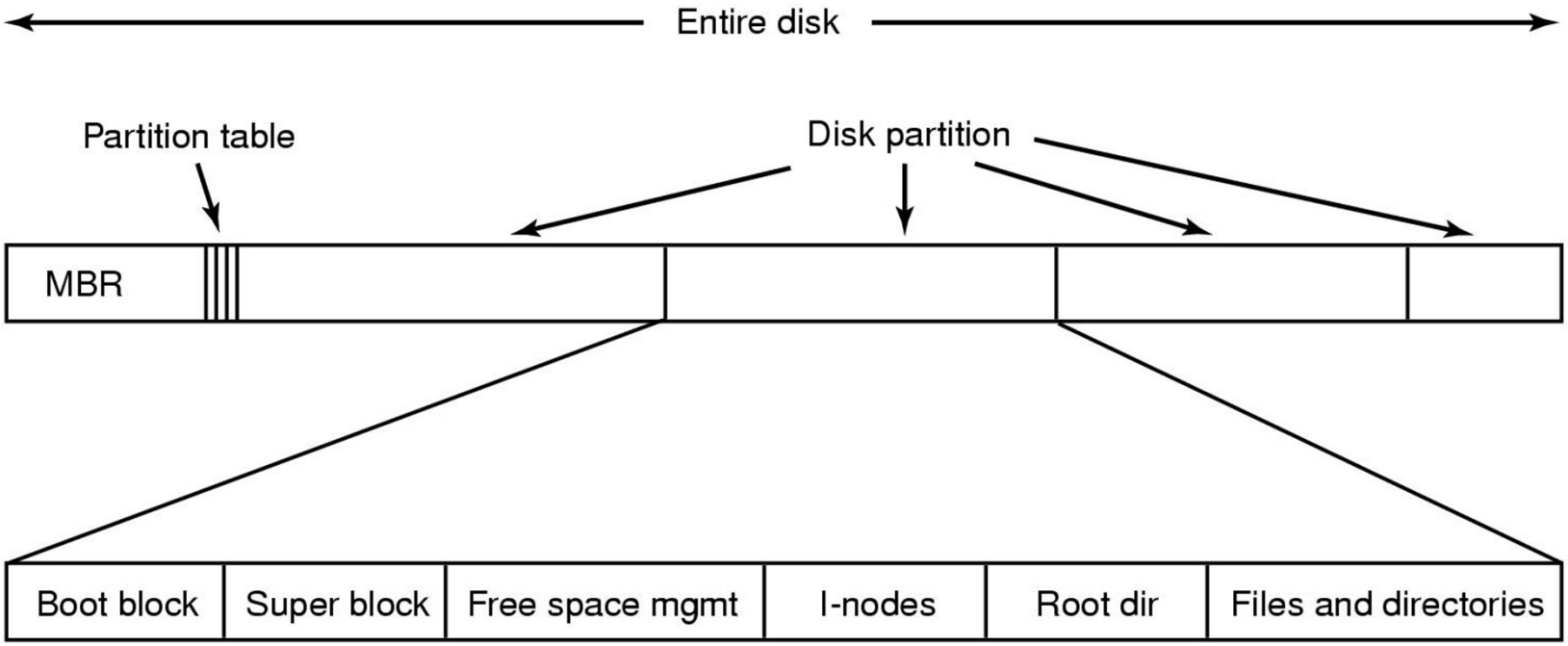


Directory Operations

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

File System Implementation

- File system layout

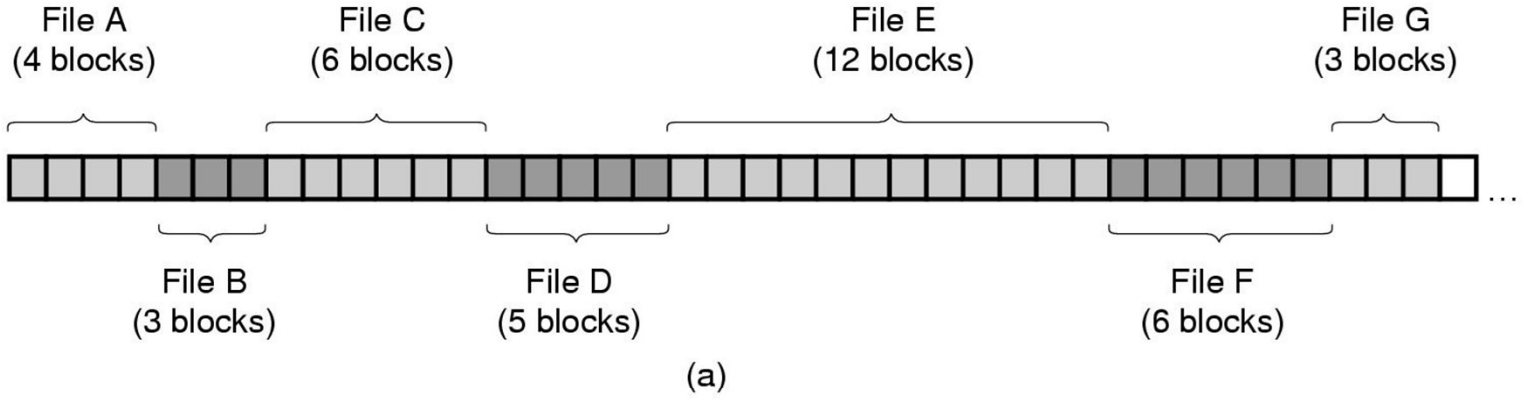


Implementing Files

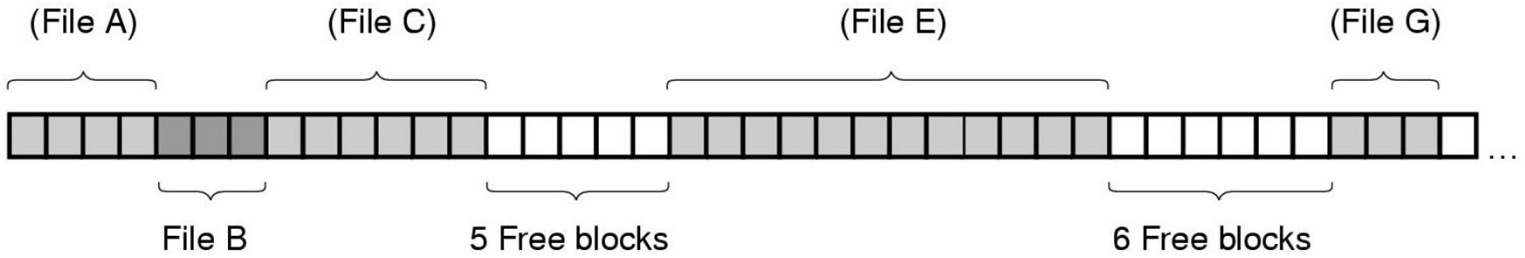
- Contiguous allocation
  - Two significant Advantages
    - Simple to implement
    - Excellent read performance

Implementing Files

- Contiguous allocation
  - Serious drawback
    - Disk becomes fragmented
  - Was used on magnetic-disk file systems years ago



(a)



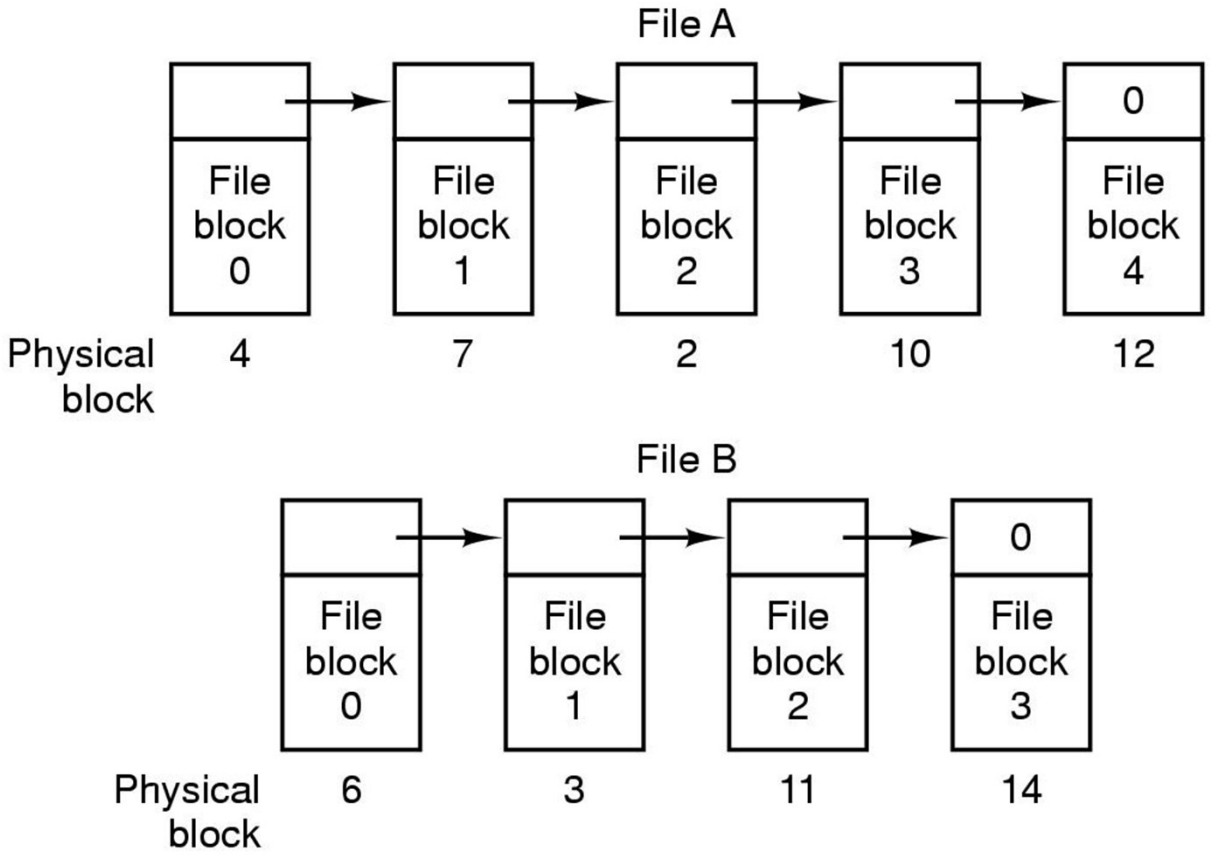
(b)

Implementing Files

- Contiguous allocation
  - Still used on CD-ROMs
  - DVD is a bit more complicated
    - File system: UDF (Universal Disk Format)
    - 30-bit number to represent file length
    - limits files to 1 GB
    - three or four 1-GB contiguous files

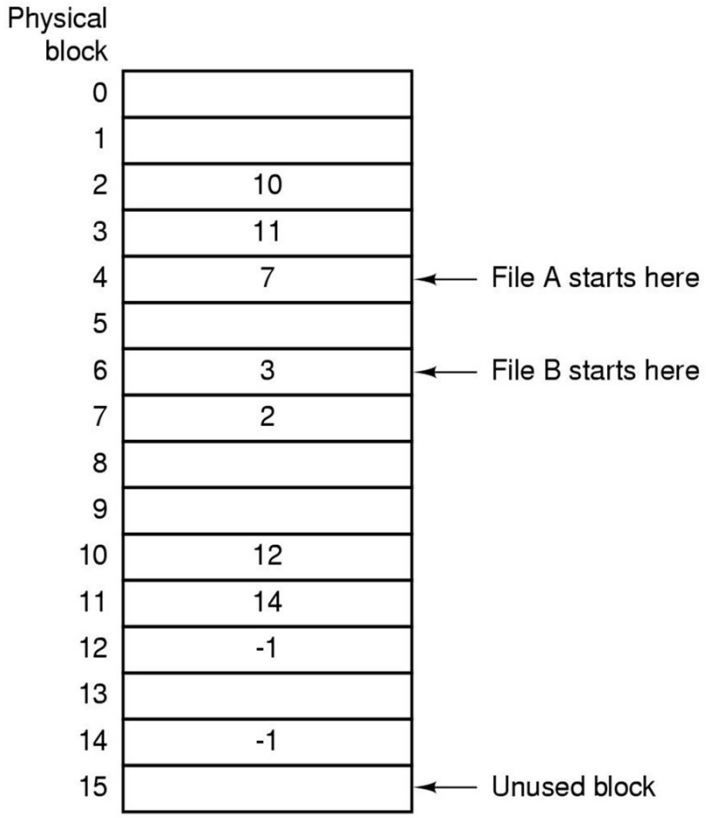
Implementing Files

- Linked-List Allocation



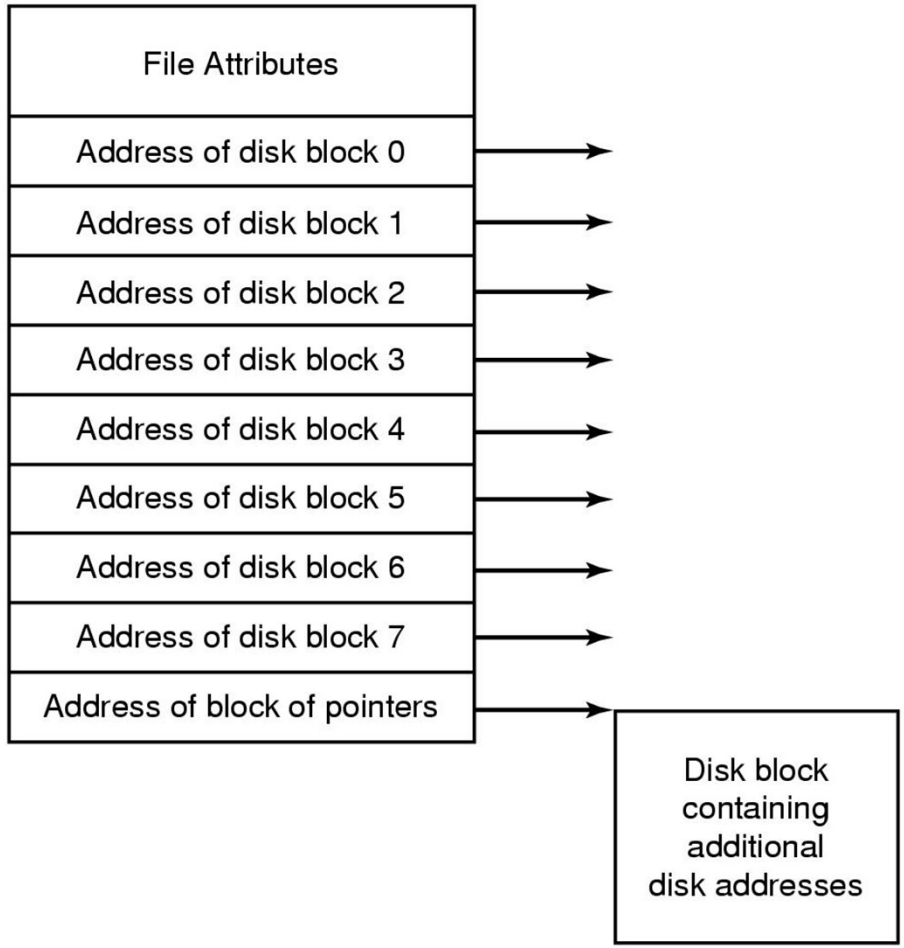
Implementing Files

- Linked list allocation using a file allocation table in RAM (FAT)



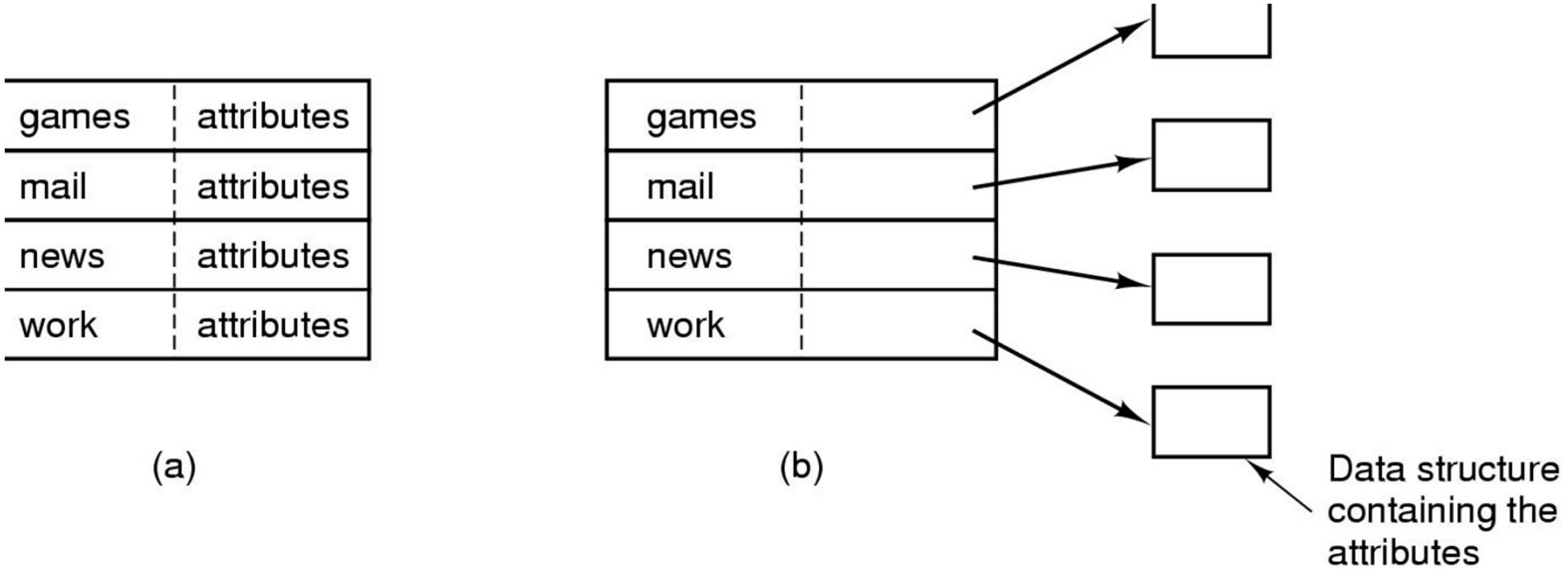
Implementing Files

- I-node



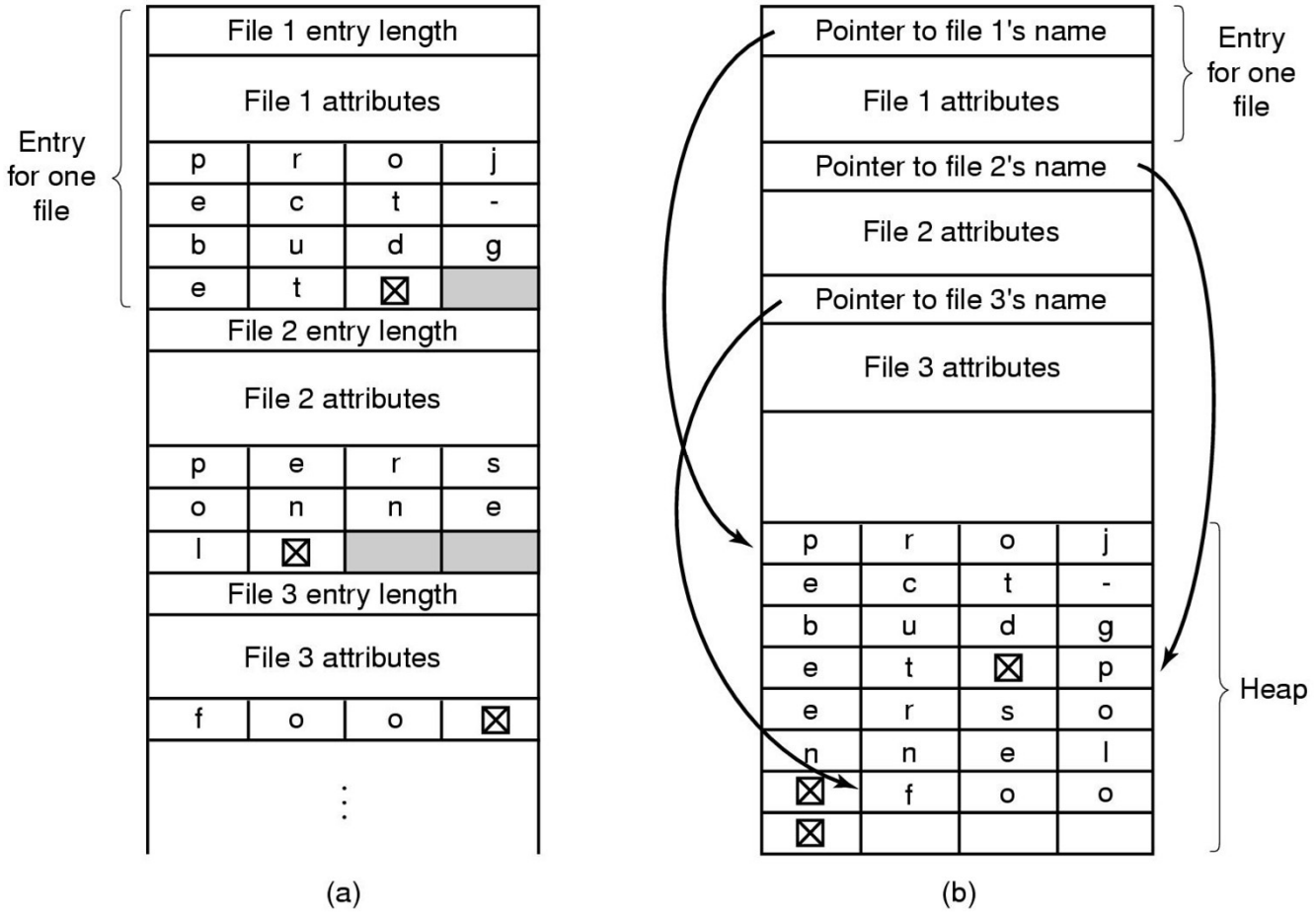
Implementing Directories

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



Implementing Directories

- Handling long file names in directory

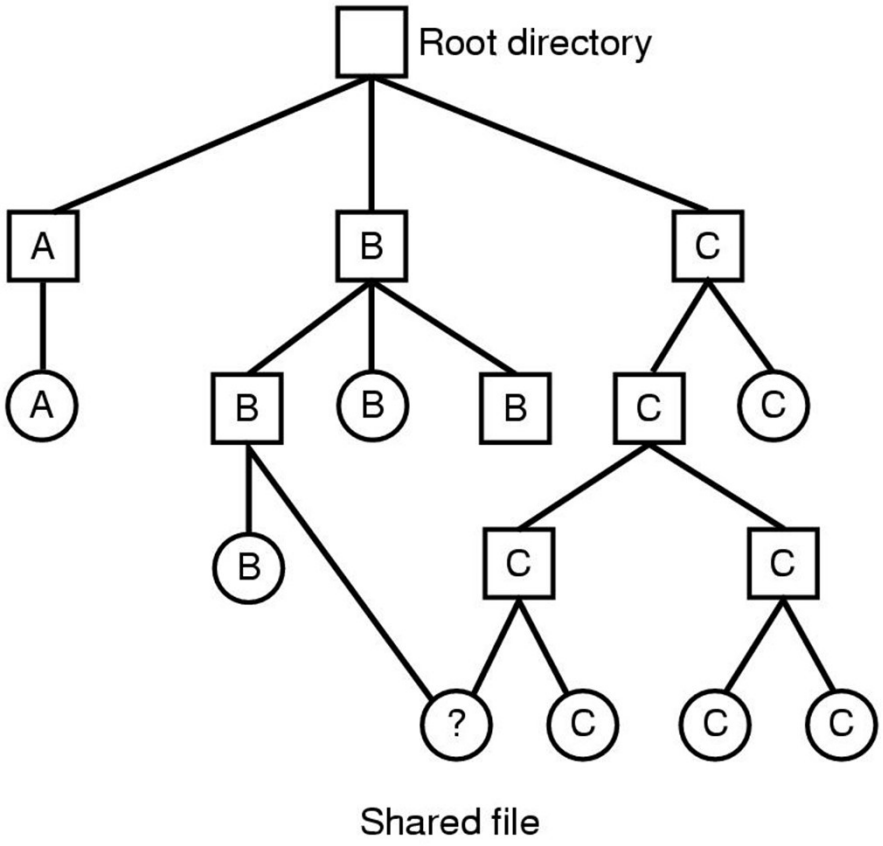


Implementing Directories

- Searching in directory
  - Linear search
    - can be slow for extremely long directories
  - use a hash table in each directory
    - much faster lookup
    - more complex administration

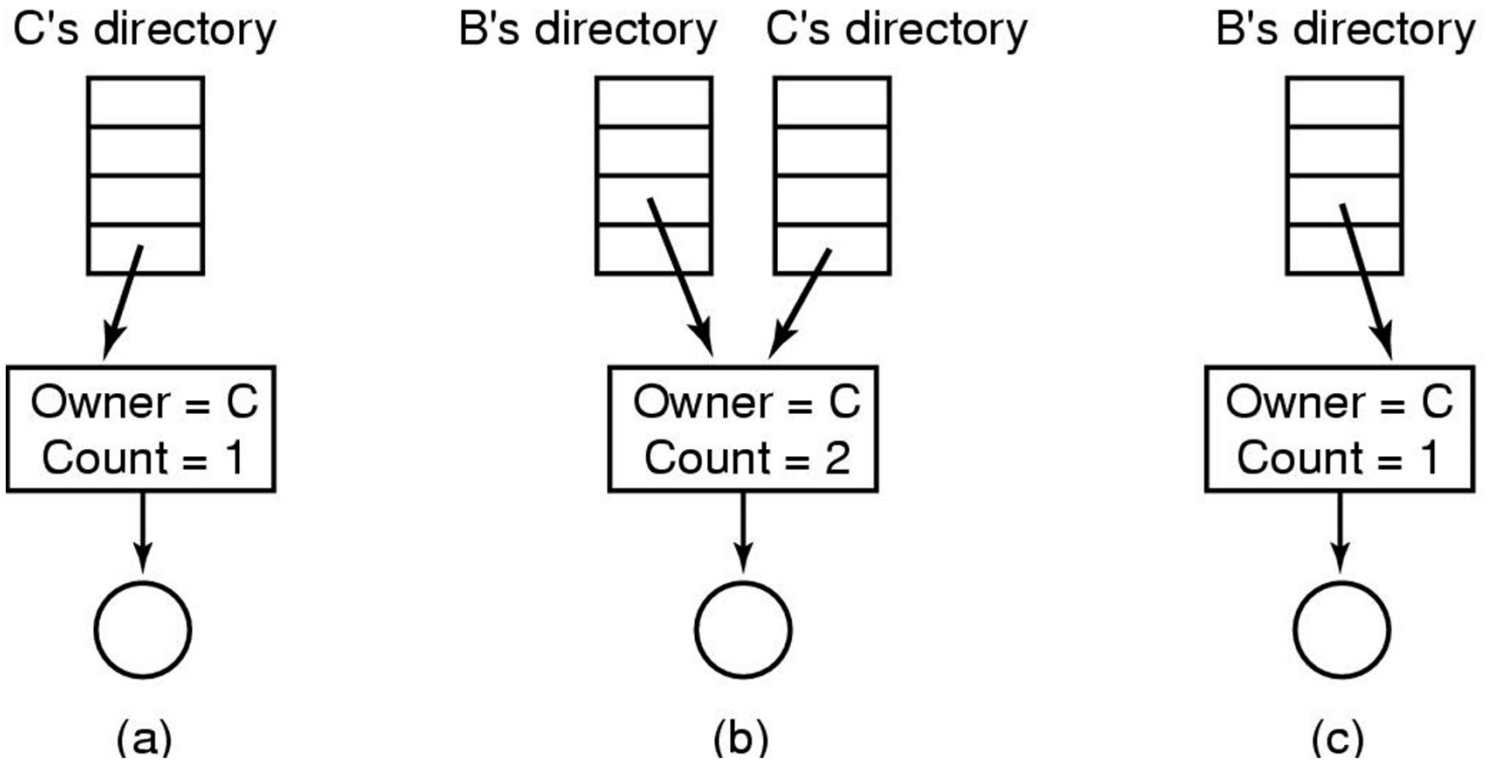
Shared Files

- File system containing a shared file
  - Hard link
  - Symbolic link



Shared Files

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



Log-Structured File Systems

- With CPUs faster, memory larger
  - disk caches can also be larger
  - increasing number of read requests can come from cache
  - thus, most disk accesses will be writes
  - Most writes are done in very small chunks.
    - highly inefficient,
    - 50-µsec disk write need 10ms seek and 4ms rotational delay.
    - Disk efficiency drops to a fraction of 1%

Log-Structured File Systems

- LFS Strategy structures entire disk as a log
  - have all writes initially buffered in memory
  - periodically
    - collect buffered data into a single segment
    - write as a contiguous segment at the end of the log
  - At the start of each segment
    - is a segment summary
    - telling what can be found in the segment

Log-Structured File Systems

- LFS Strategy structures entire disk as a log
  - i-nodes still exist and scattered all over the log
    - An i-node map is required to find i-nodes
    - Entry i in this map points to i-node i on the disk
    - The map is kept on disk
    - it is also cached in memory
  - Opening a file now consists of
    - using the map to locate the i-node for the file.
    - the addresses of the blocks can be found from i-node

Log-Structured File Systems

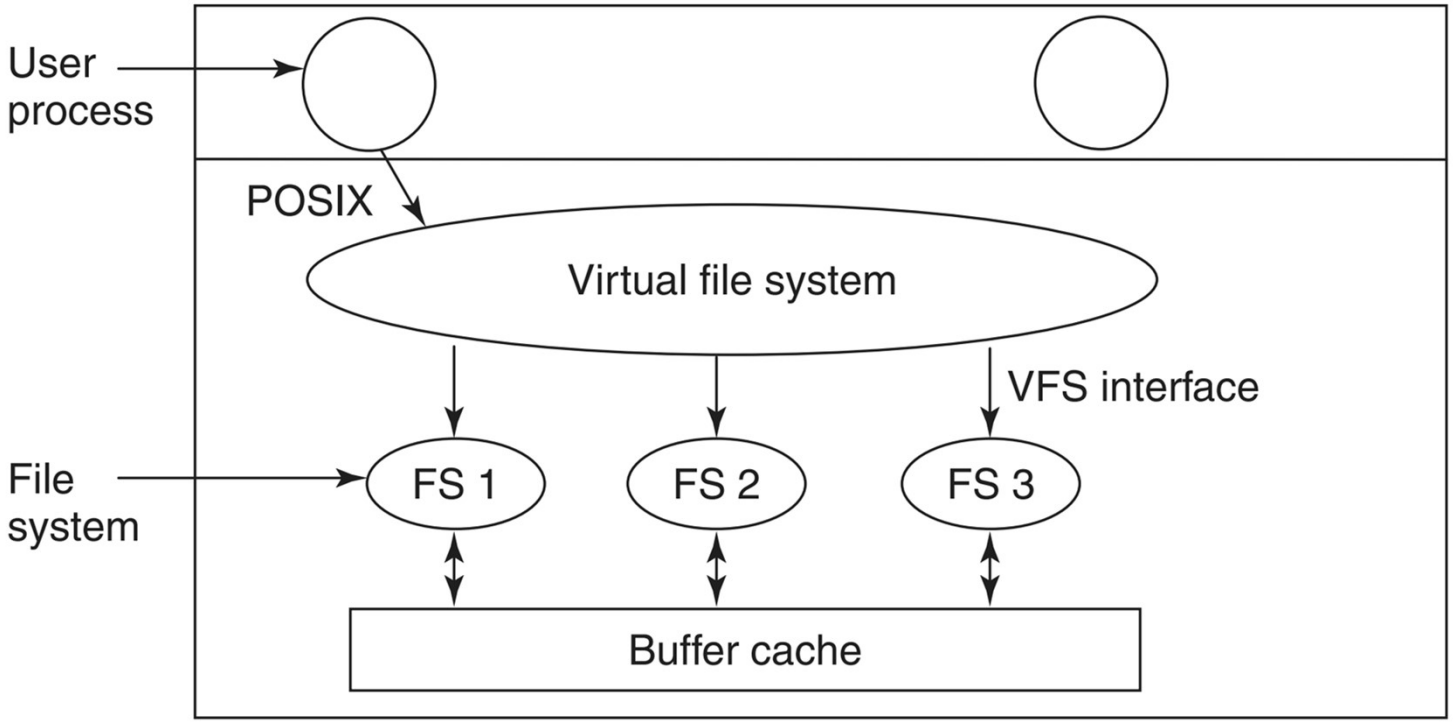
- LFS Strategy structures entire disk as a log
  - Eventually the log will occupy the entire disk
    - no new segments can be written to the log
    - many existing segments may have un-needed blocks
    - For example
      - if a file is overwritten
      - its i-node will now point to the new blocks
      - but the old ones will still be occupying space in previously written segments

Log-Structured File Systems

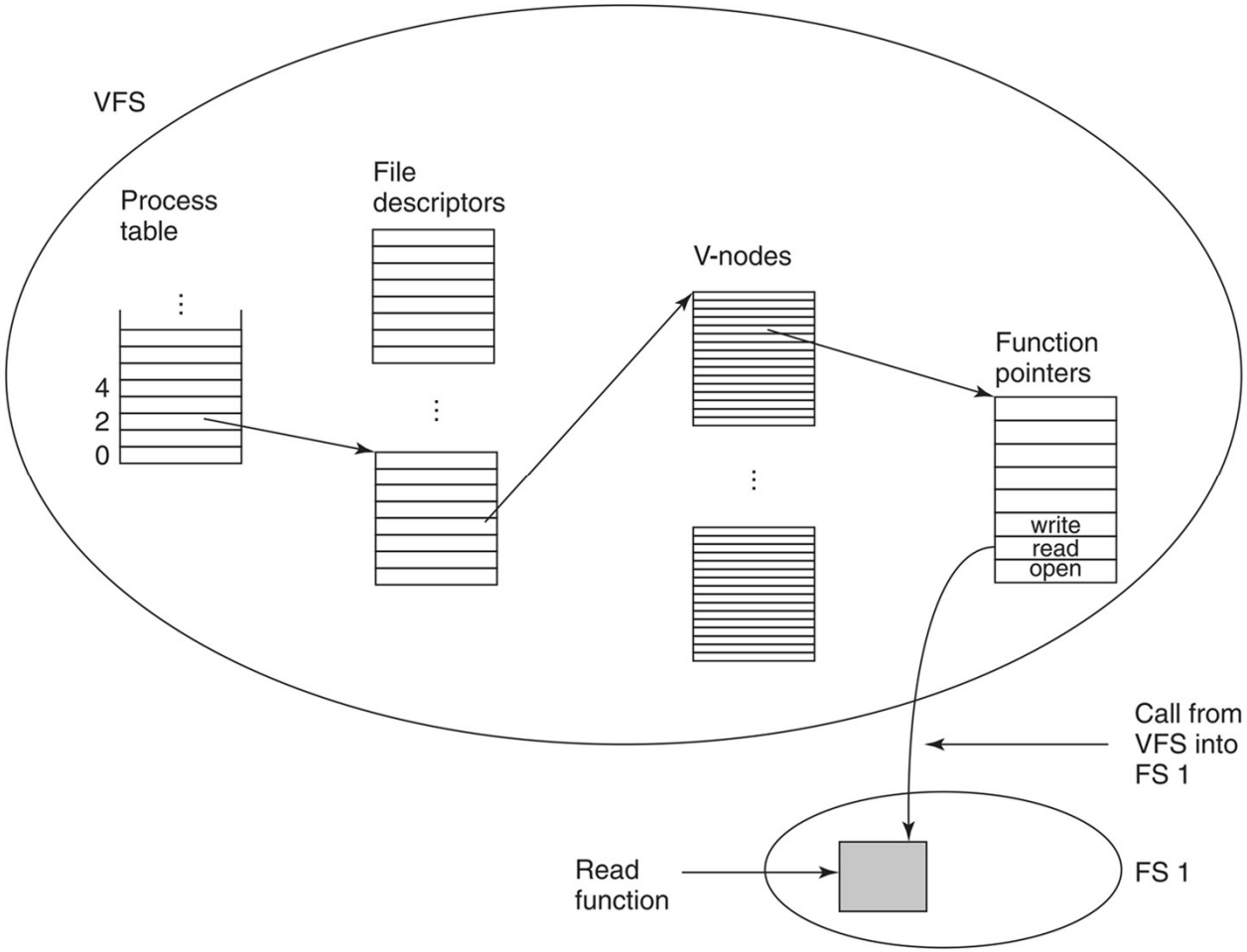
- LFS Strategy structures entire disk as a log
  - Eventually the log will occupy the entire disk
    - a cleaner thread scans the log circularly
      - reads the summary to see which i-nodes and files are there.
      - then checks with the current i-node map
      - old i-nodes and file blocks are discarded.
      - The i-nodes and blocks that are still in use go into memory to be written out in the next segment.
      - The original segment is then marked as free

Virtual File Systems

- To integrate multiple file systems together
  - abstract common part of all file systems
  - put that code in a separate layer

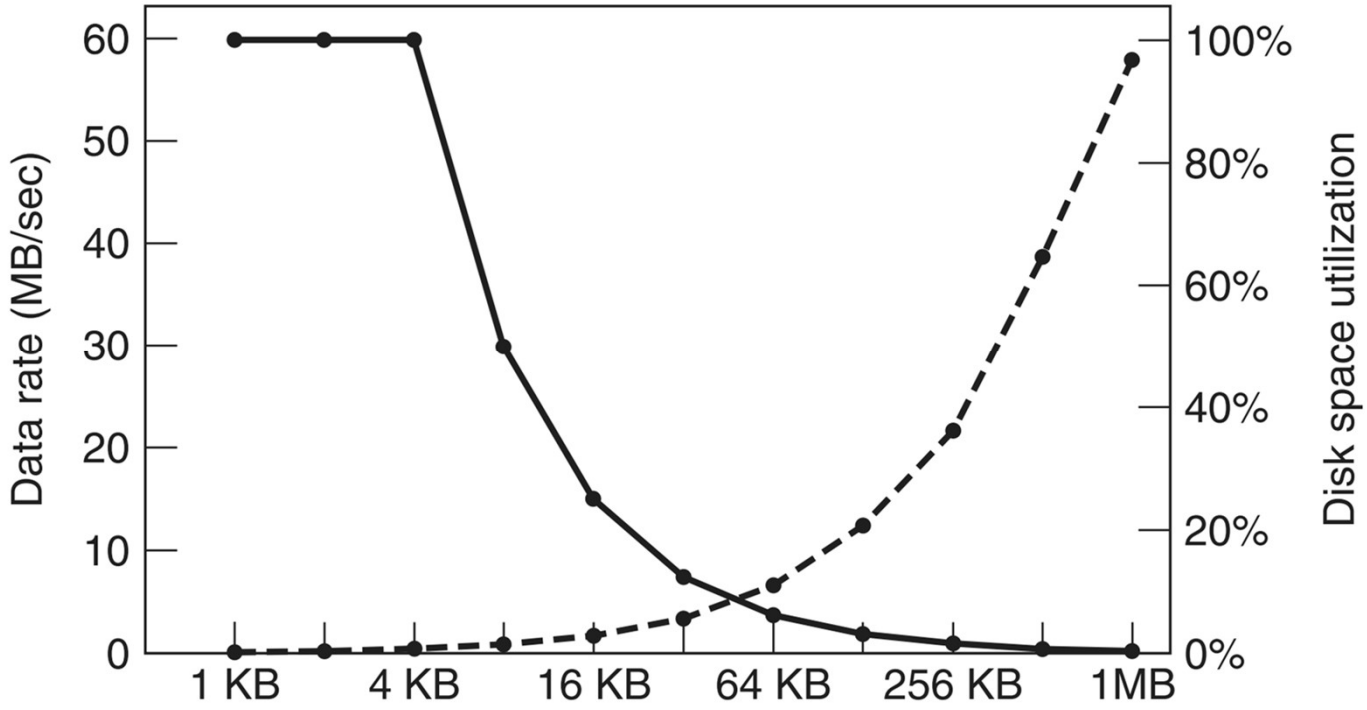


Virtual File Systems



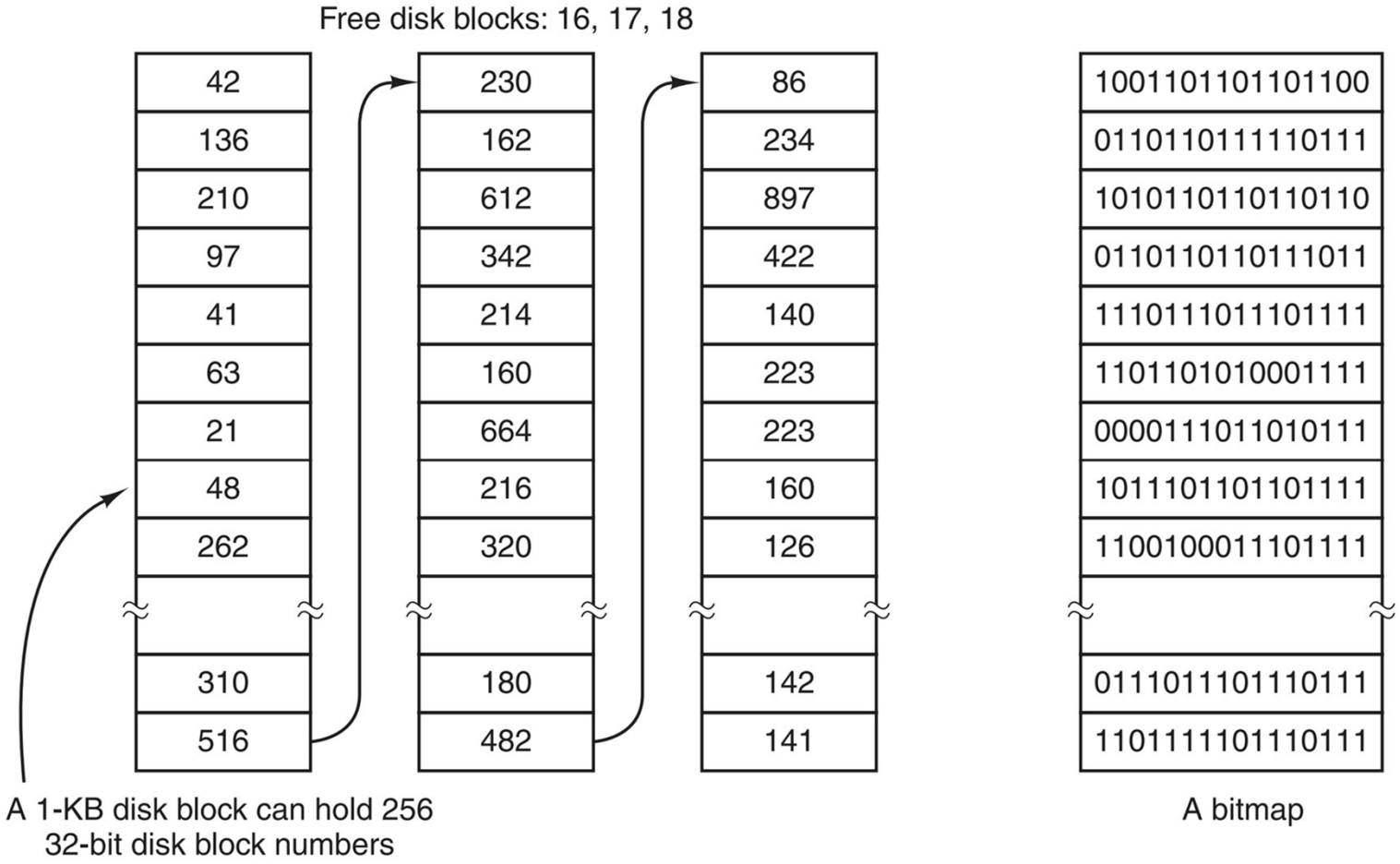
Disk Space Management

- Block size
  - too large: we waste space
  - too small: we waste time



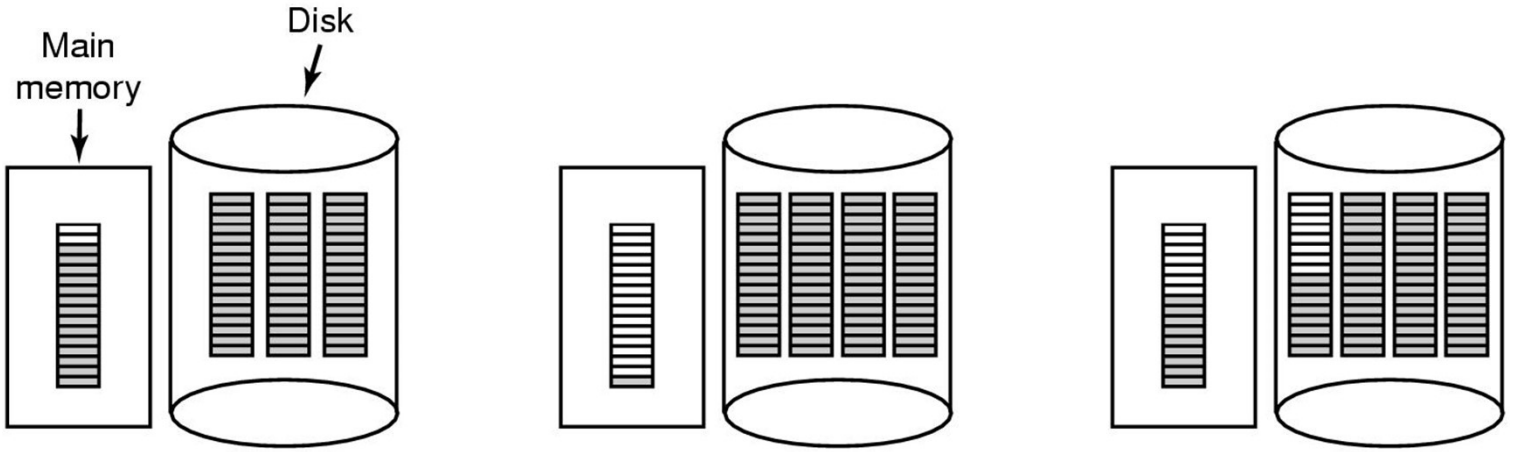
Disk Space Management

- Keeping track of free blocks



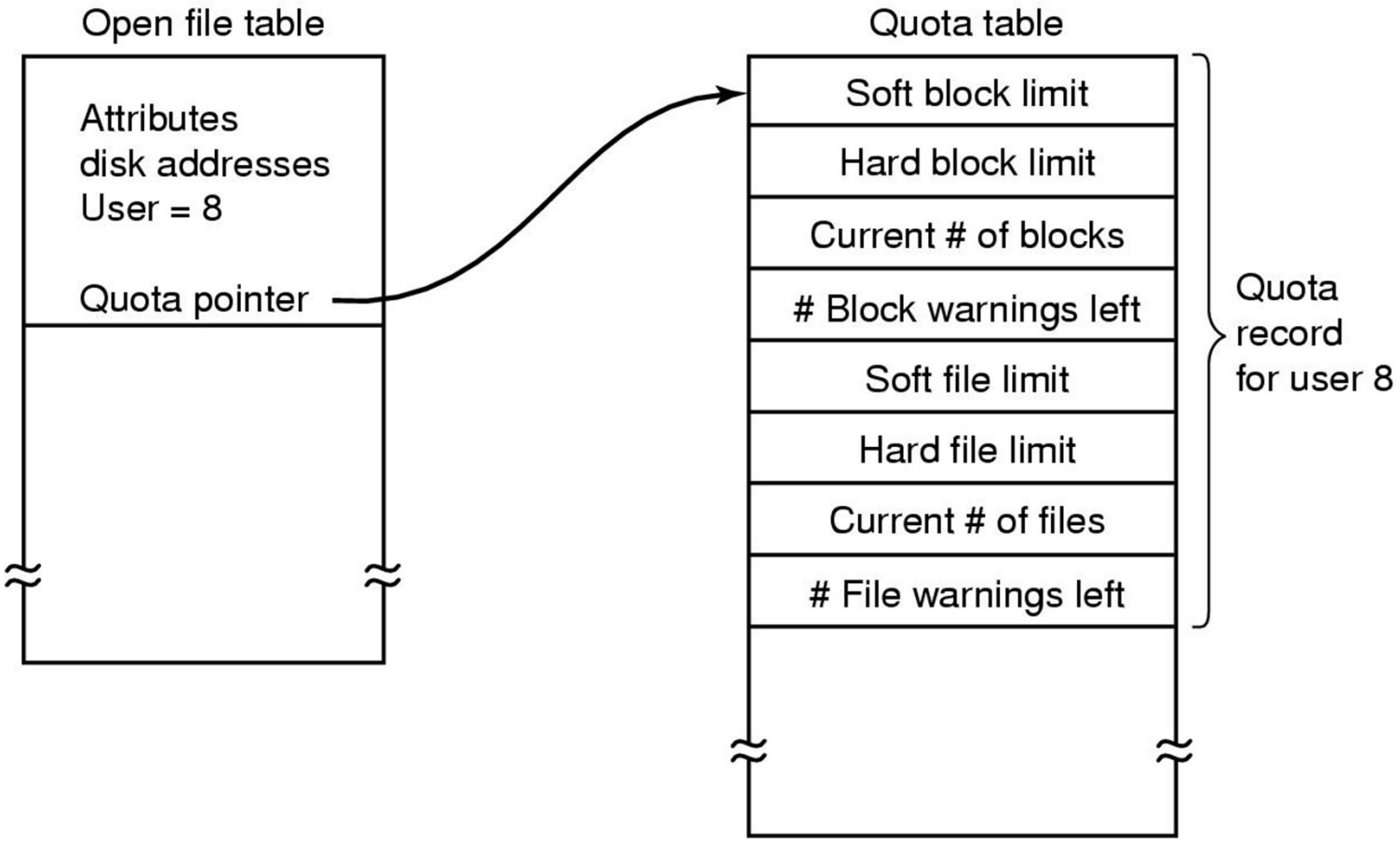
Disk Space Management

- (a) Almost-full block of pointers to free disk blocks in RAM
  - - three blocks of pointers on disk
- (b) Result of freeing a 3-block file
- (c) Alternative strategy for handling 3 free blocks



Disk Space Management

- Quotas for keeping track of each user's disk use



File System Backups

- Making a backup
  - takes a long time
  - occupies a large amount of space
  - doing it efficiently and conveniently is important
- Issues
  - should the entire file system be backed up or only part of it?
  - wasteful to back up files that have not changed
    - idea of incremental dumps
  - Desire to compress
  - Difficult to perform a backup on an active file system
    - rapid snapshots
  - introduces many nontechnical problems into an organization
    - Security

File System Backups

- Two strategies for dumping a disk
  - Physical dump
    - Advantage
      - Simplicity and great speed
    - issues
      - unused blocks
      - bad blocks
      - paging, hibernation and other internal files
    - Disadvantages
      - inability to
        - skip selected directories
        - make incremental dumps
        - restore individual files upon request

File System Backups

- Two strategies for dumping a disk
  - Logical dump
    - starts at specified directories
    - recursively dumps all changed files and directories
    - most common form
    - easy to restore a specific file or directory

File System Backups

- Logical dump algorithm
  - Dumps all directories (even unmodified ones) on the path to a modified file or directory
    - to make it possible to restore to a fresh file system
    - to make it possible to incrementally restore a single file

File System Backups

- Logical dump algorithm
  - Sample file system to be dumped

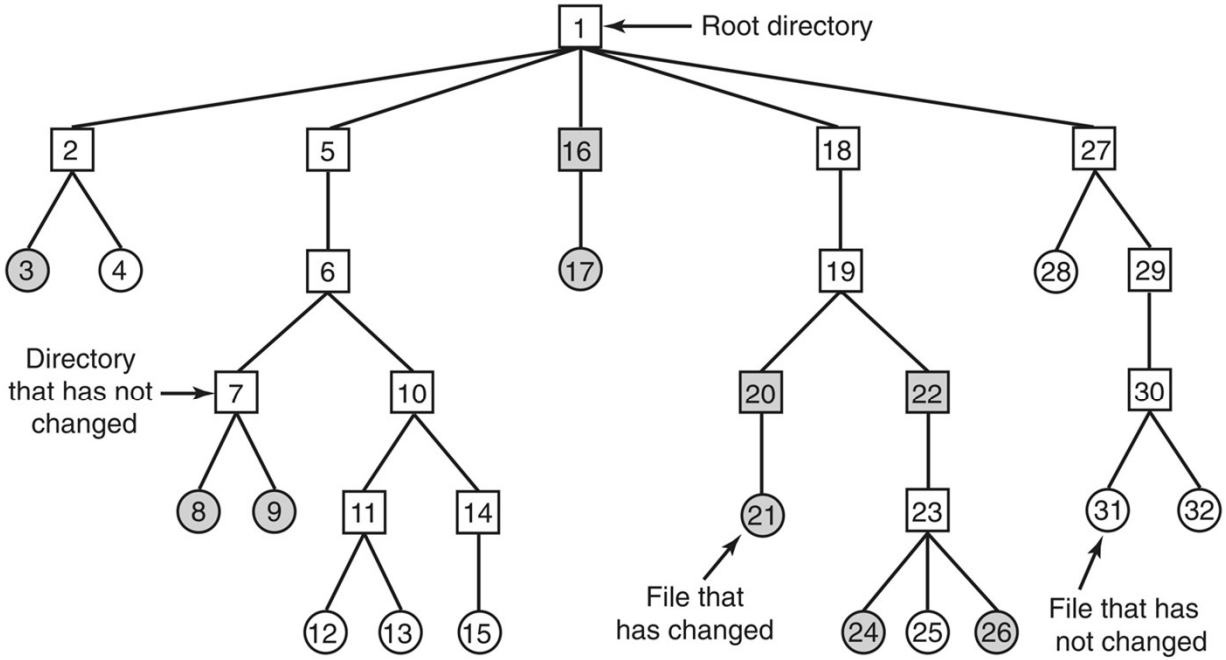
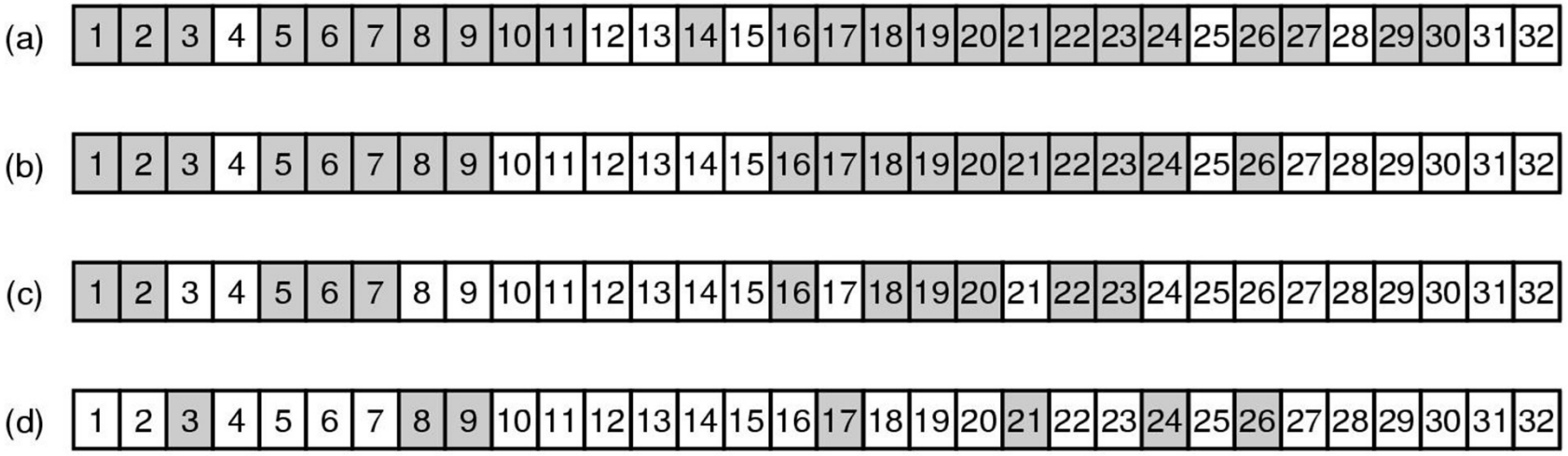


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

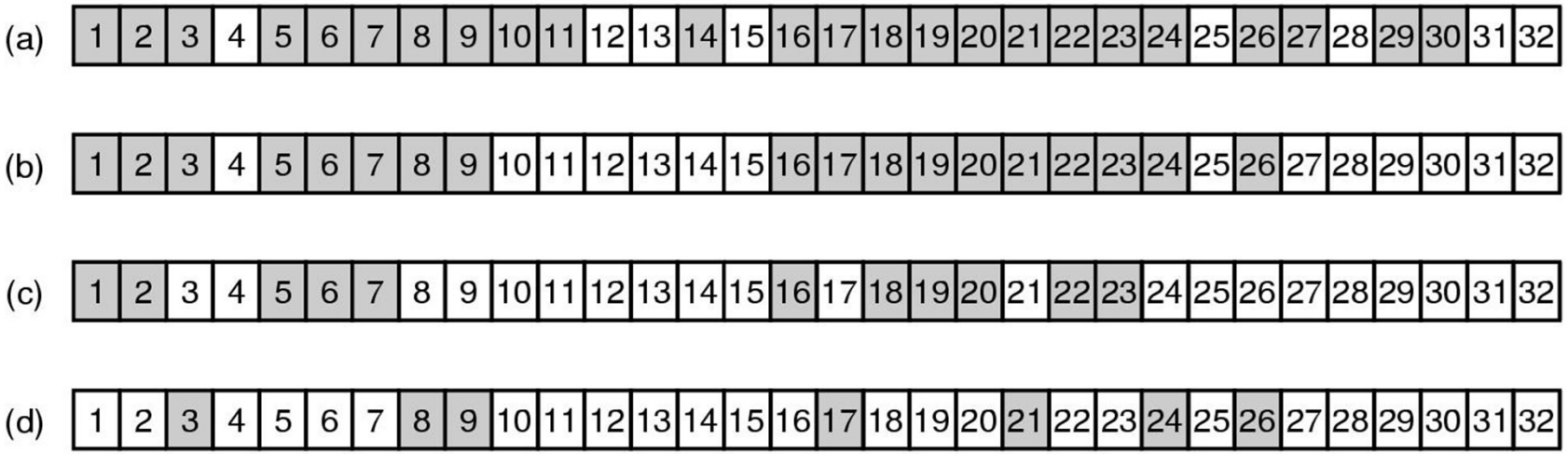
File System Backups

- Logical dump algorithm
  - maintains a bitmap indexed by i-node number
  - operates in four phases



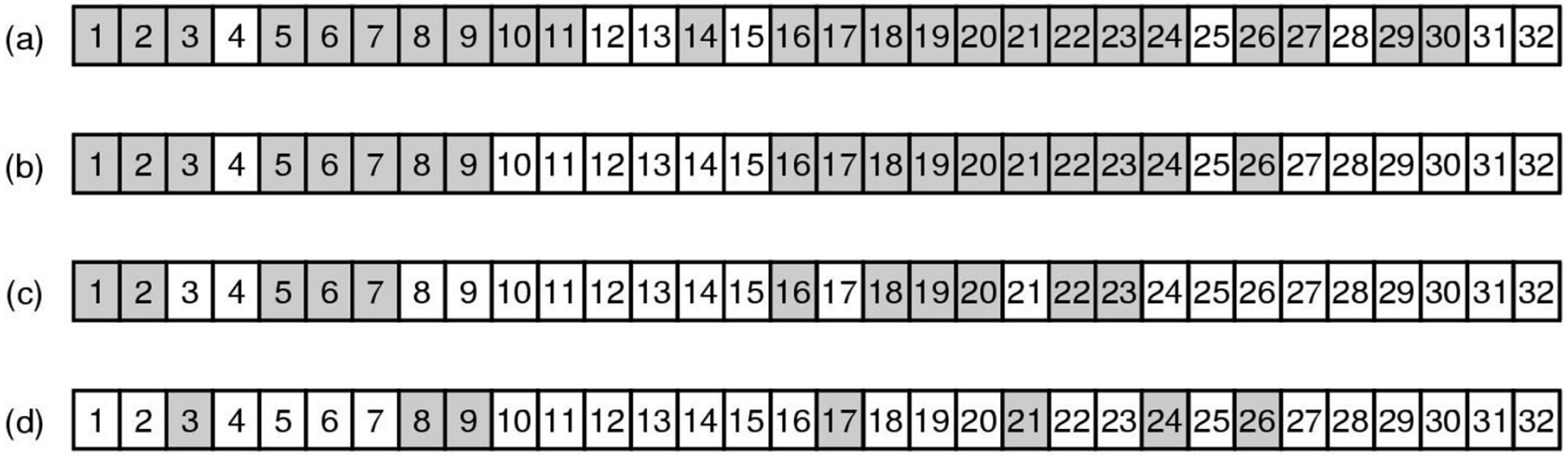
File System Backups

- Logical dump algorithm
  - Phases 1
    - Marks I-nodes for
      - Modified files
      - All directories



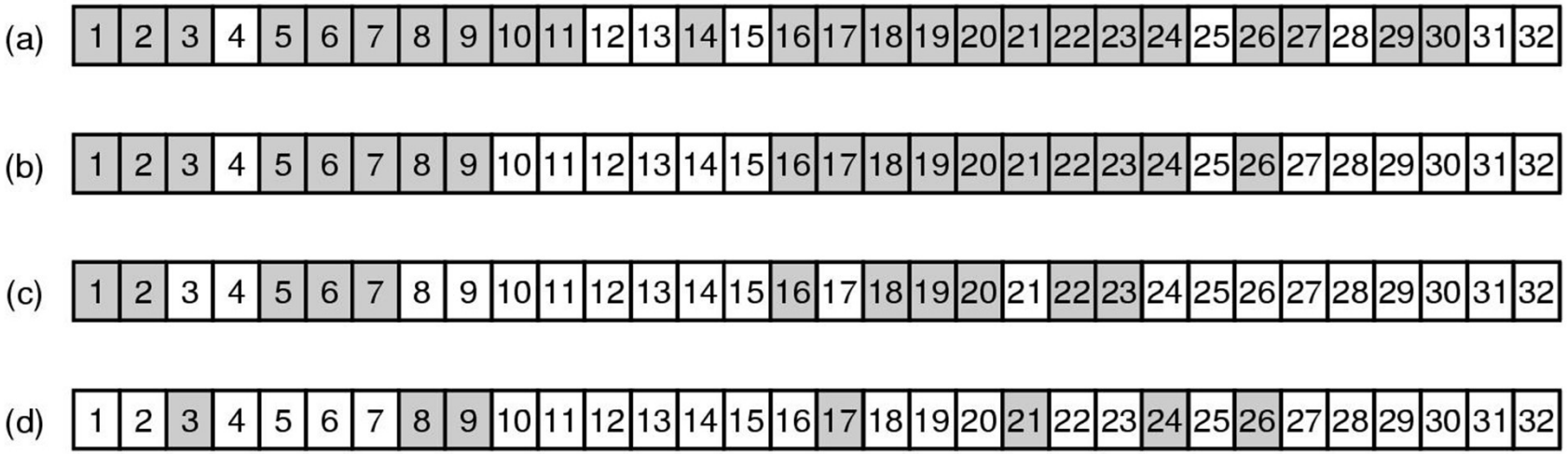
File System Backups

- Logical dump algorithm
  - Phases 2
    - Recursively walks the tree
      - unmarking any directories that have no modified files or directories



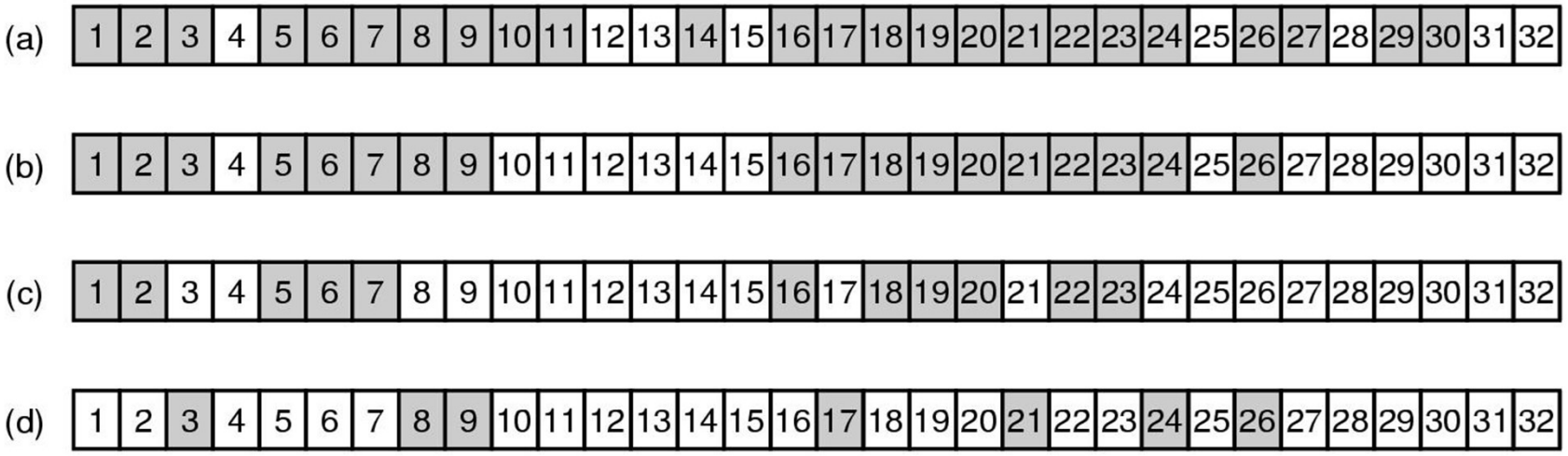
File System Backups

- Logical dump algorithm
  - Phases 3
    - Scanning the i-nodes in numerical order
      - Dumping all marked directories
      - Each directory is prefixed by the directory's attributes



File System Backups

- Logical dump algorithm
  - Phases 4
    - Dumping marked files
      - Prefixed by their attributes



File System Backups

- Logical dump algorithm
  - Restoring is straightforward
    - An empty file system is created on the disk
    - The most recent full dump is restored
      - Directories are all restored first
      - Then the files themselves are restored.
    - This process is then repeated with the first incremental dump
    - then the next one, and so on

File System Backups

- Logical dumping issues
  - free block list must be reconstructed after restoring all dumps
  - Links
  - UNIX files may contain holes
  - special files, named pipes, and anything that is not a real file should never be dumped

File System Consistency

- UNIX utility: fsck
  - Block check

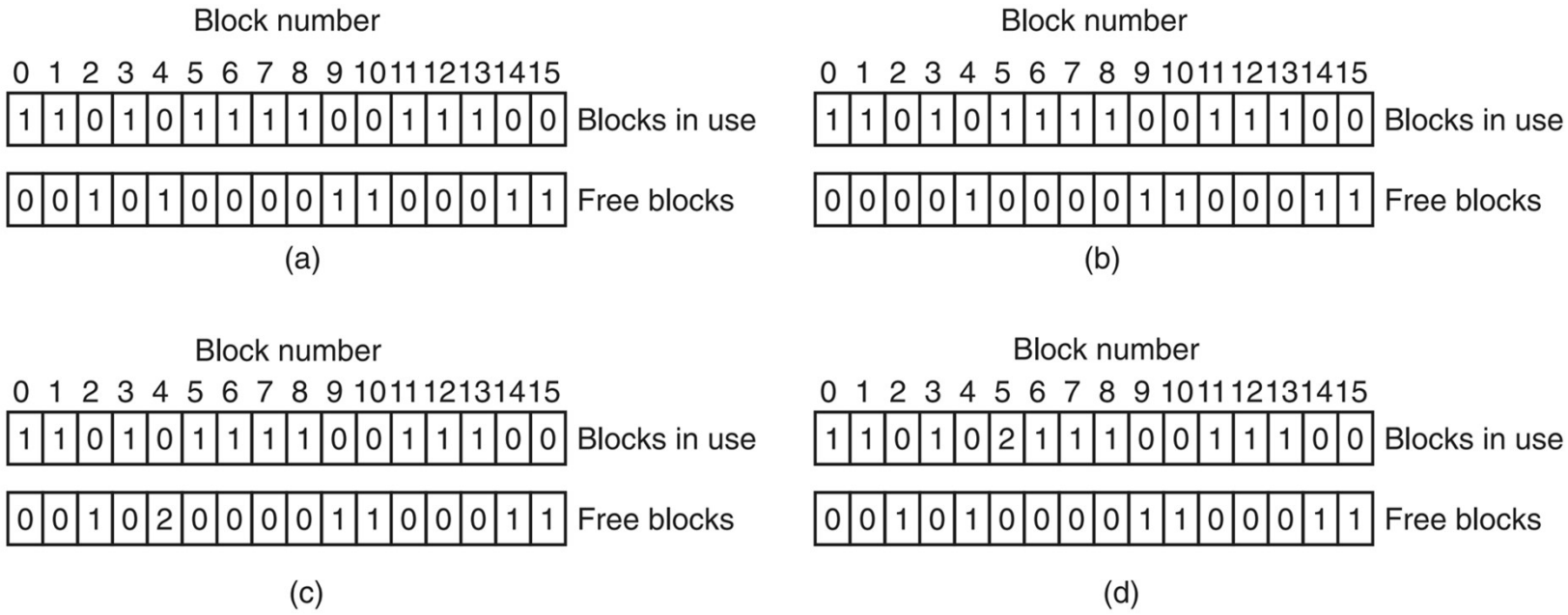
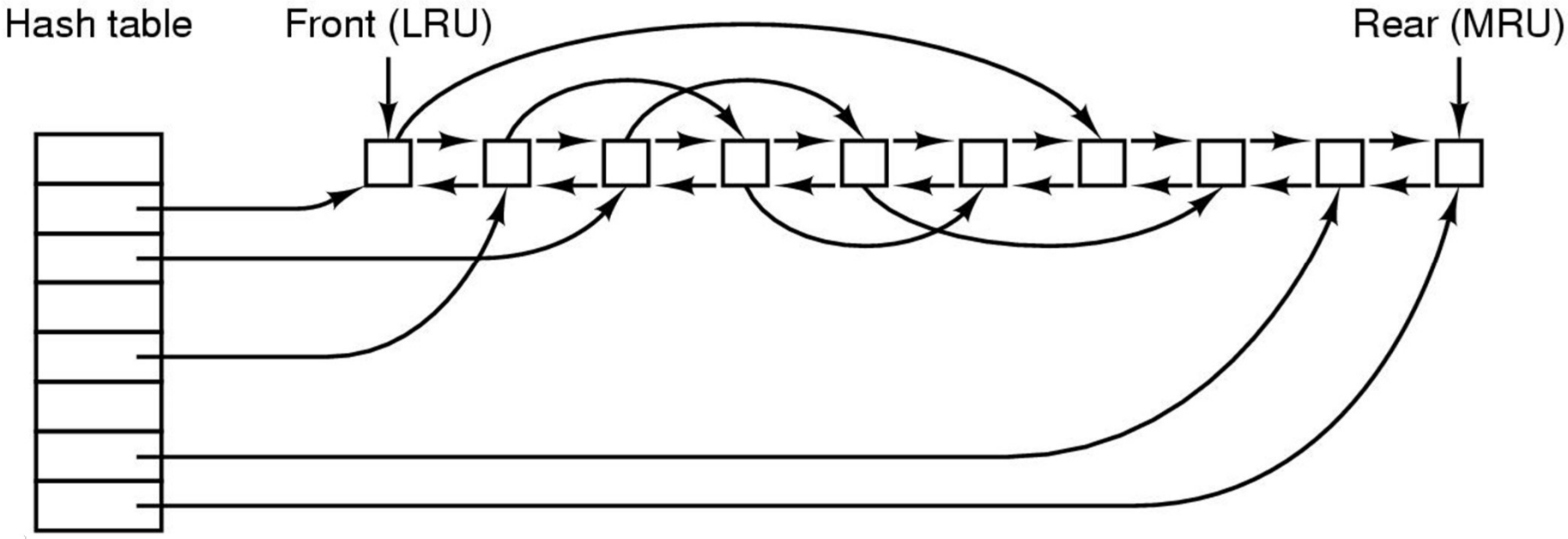


Figure 4-27. File-system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

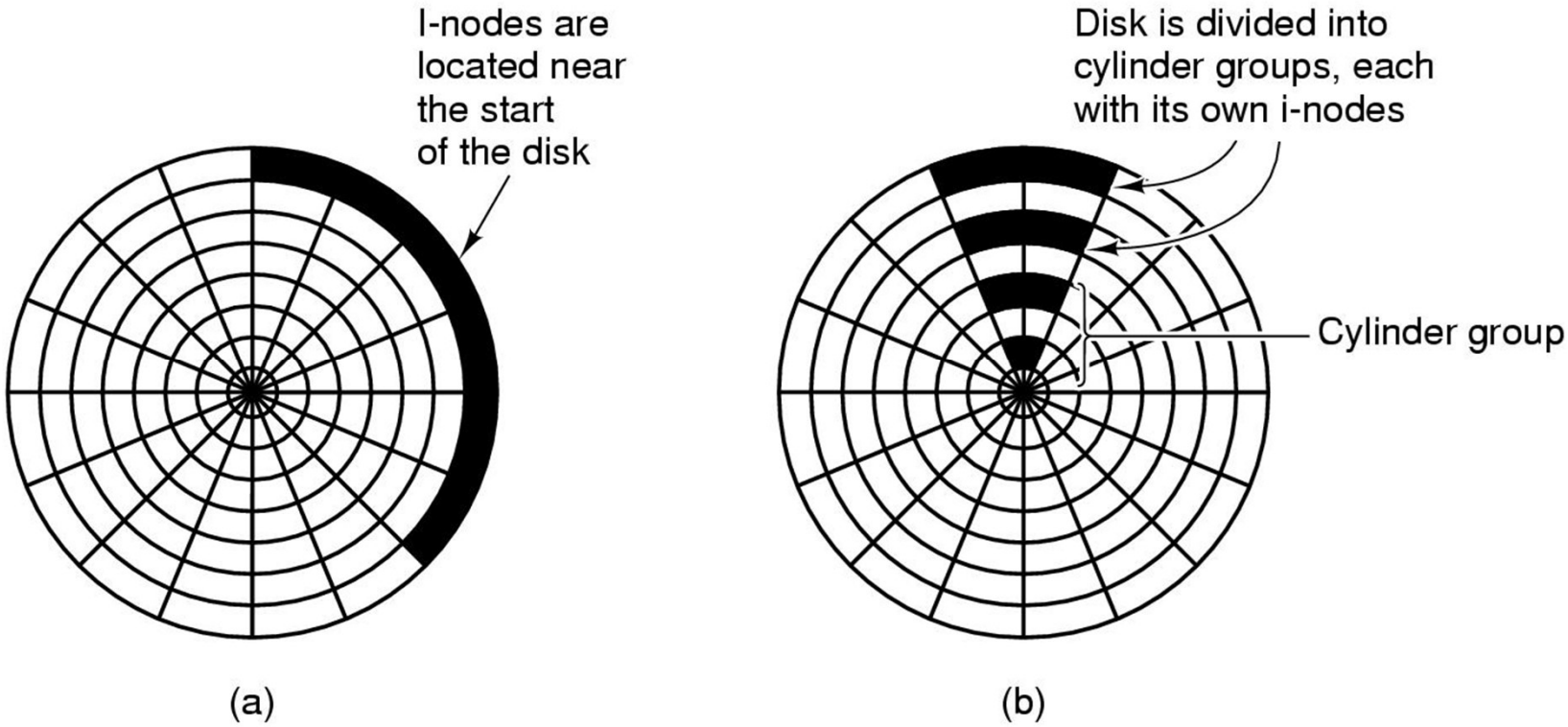
File System Consistency

- UNIX utility: fsck
  - File check
    - Inspect each directory in the file system recursively
      - Count the references for each i-node
      - Compare the counts with link counts in i-nodes
        - link count too high
          - Waste space
          - Should be fixed by setting the link count to the correct value
        - link count too low
          - potentially catastrophic
          - solution is just to force the link count to the actual number of directory entries

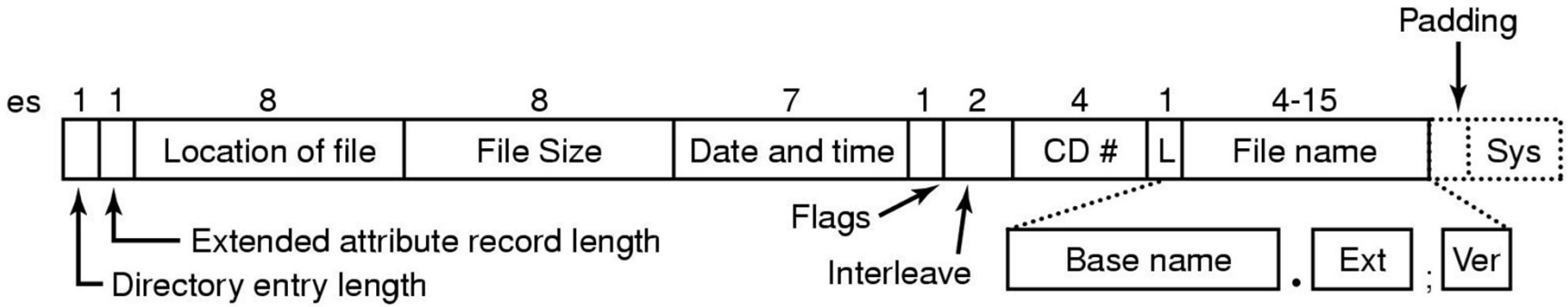
File System Performance (1)



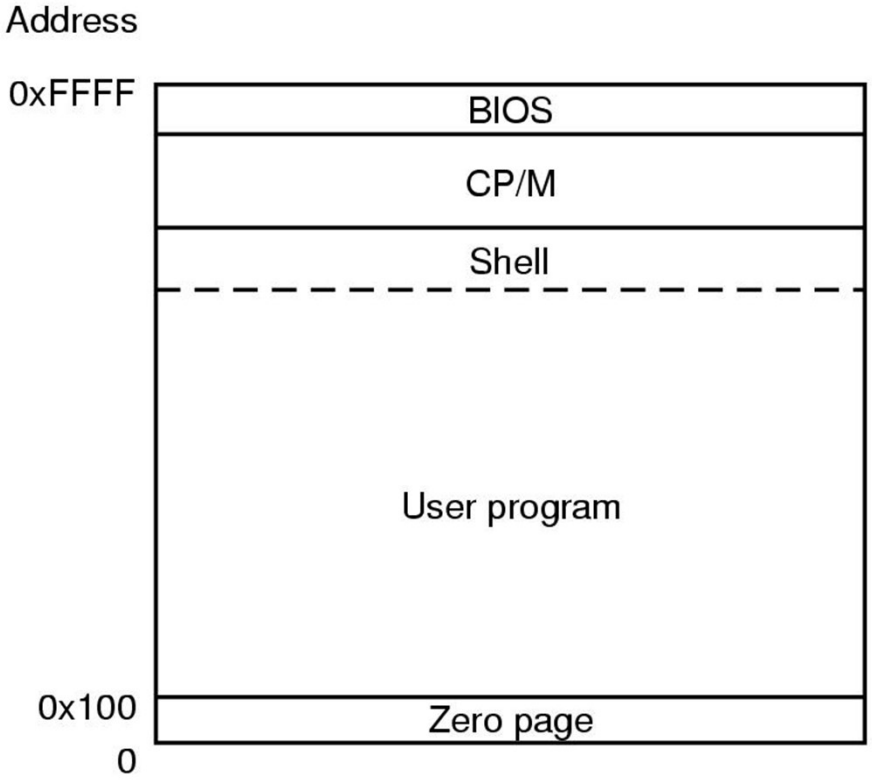
File System Performance (2)



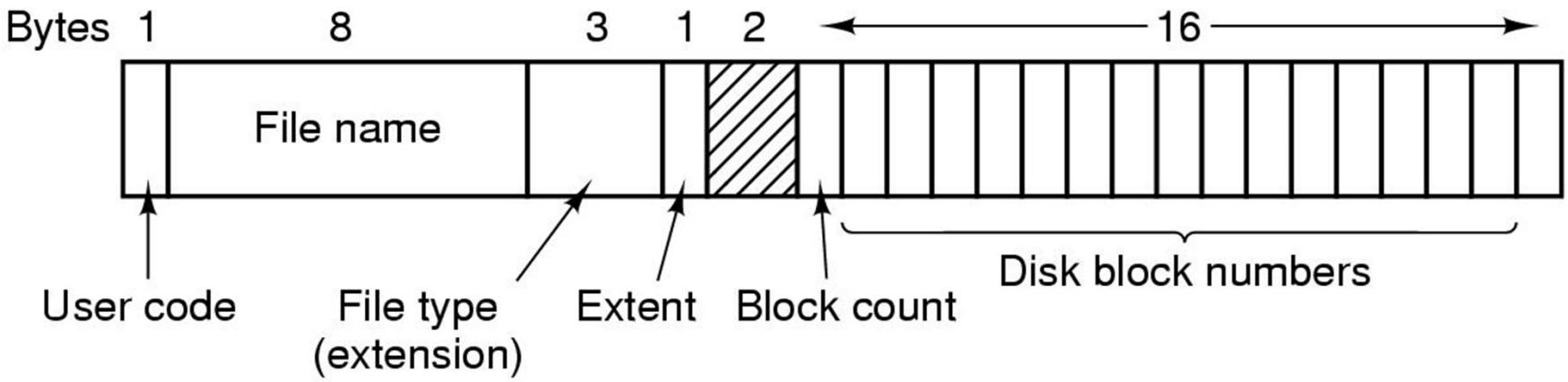
Example File Systems
CD-ROM File Systems



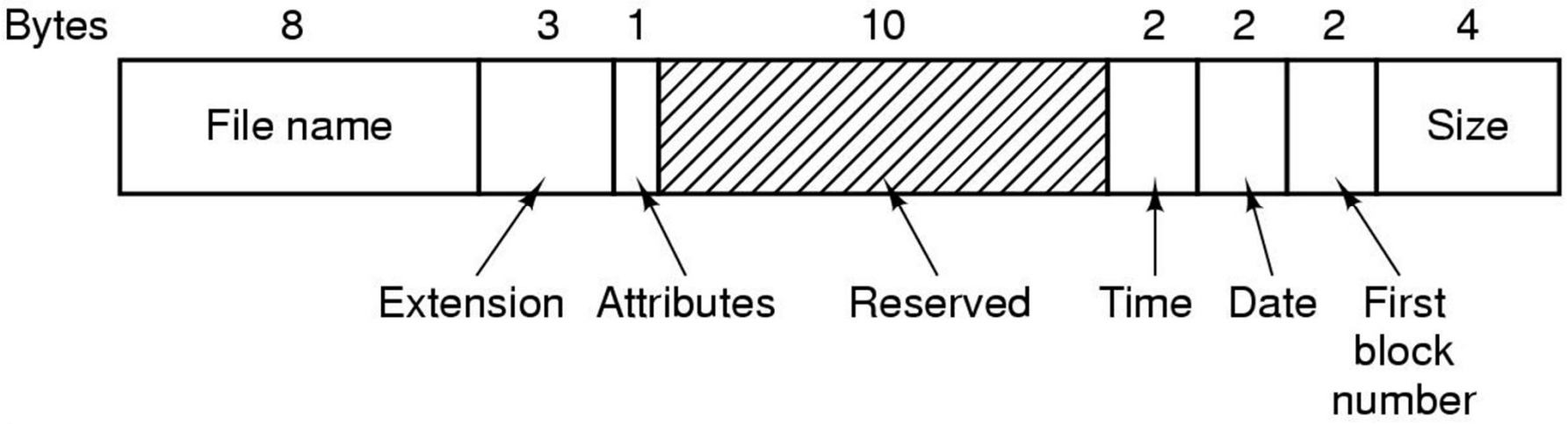
The CP/M File System (1)



The CP/M File System (2)



The MS-DOS File System (1)

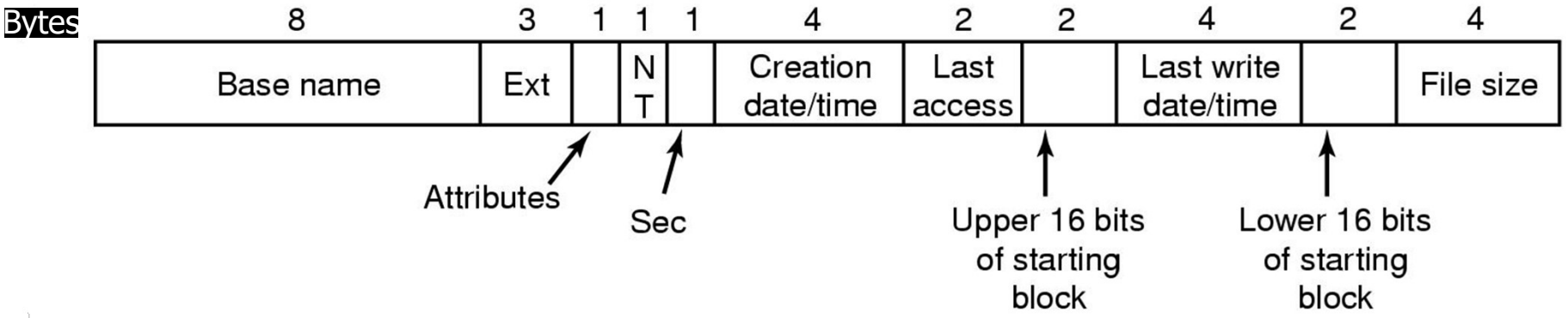


The MS-DOS File System (2)

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

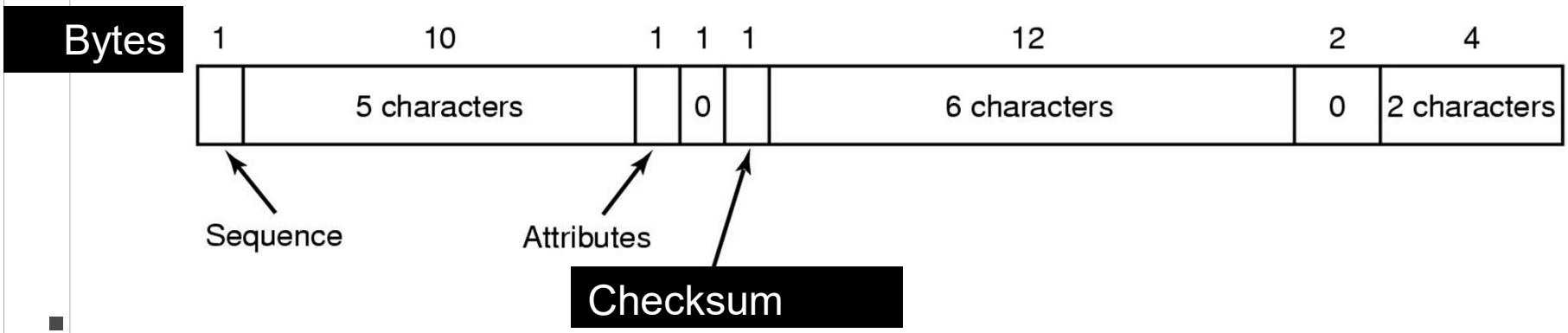
The Windows 98 File System (1)

- The extended MS-DOS directory entry

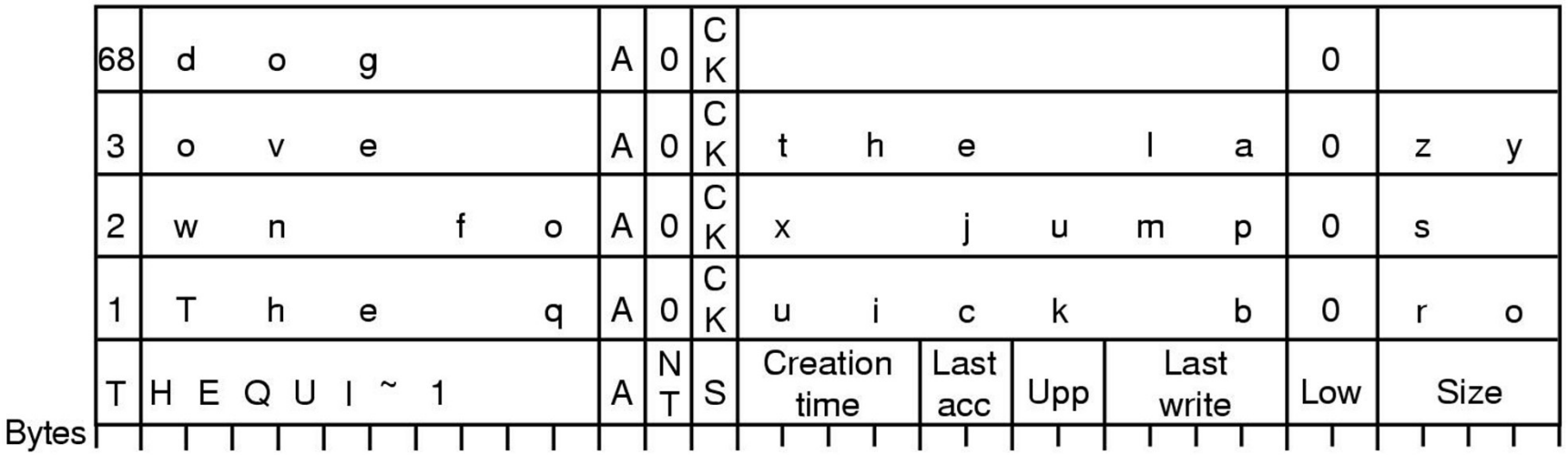


The Windows 98 File System (2)

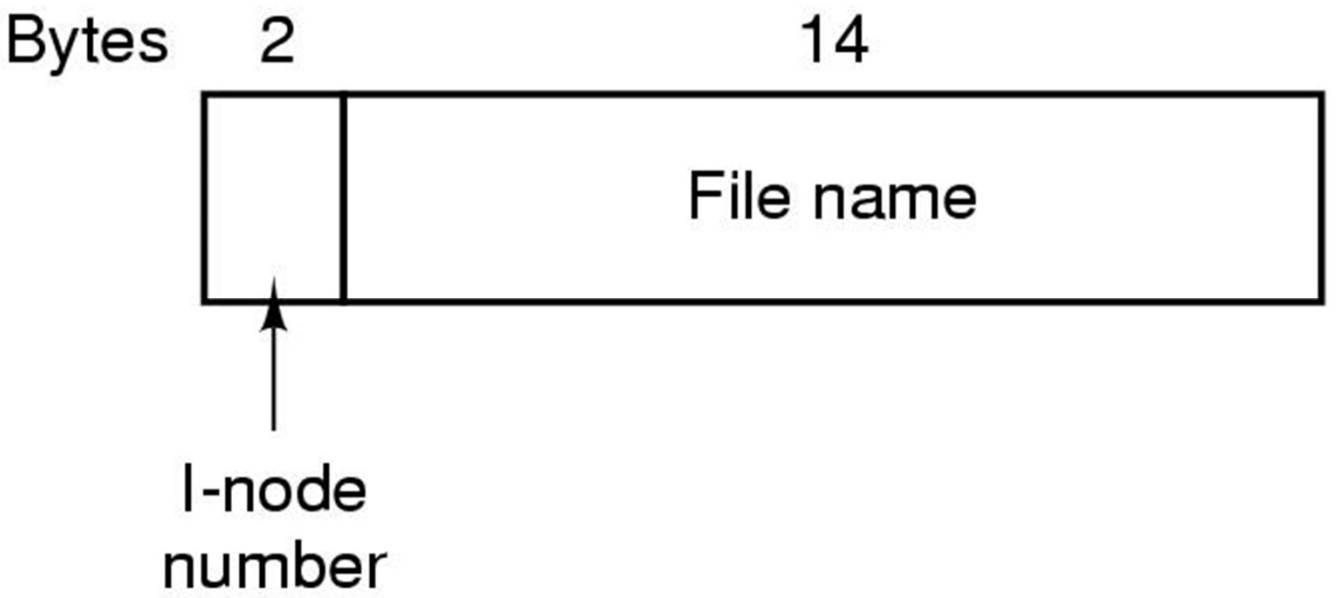
- An entry for (part of) a long file name in Windows 98



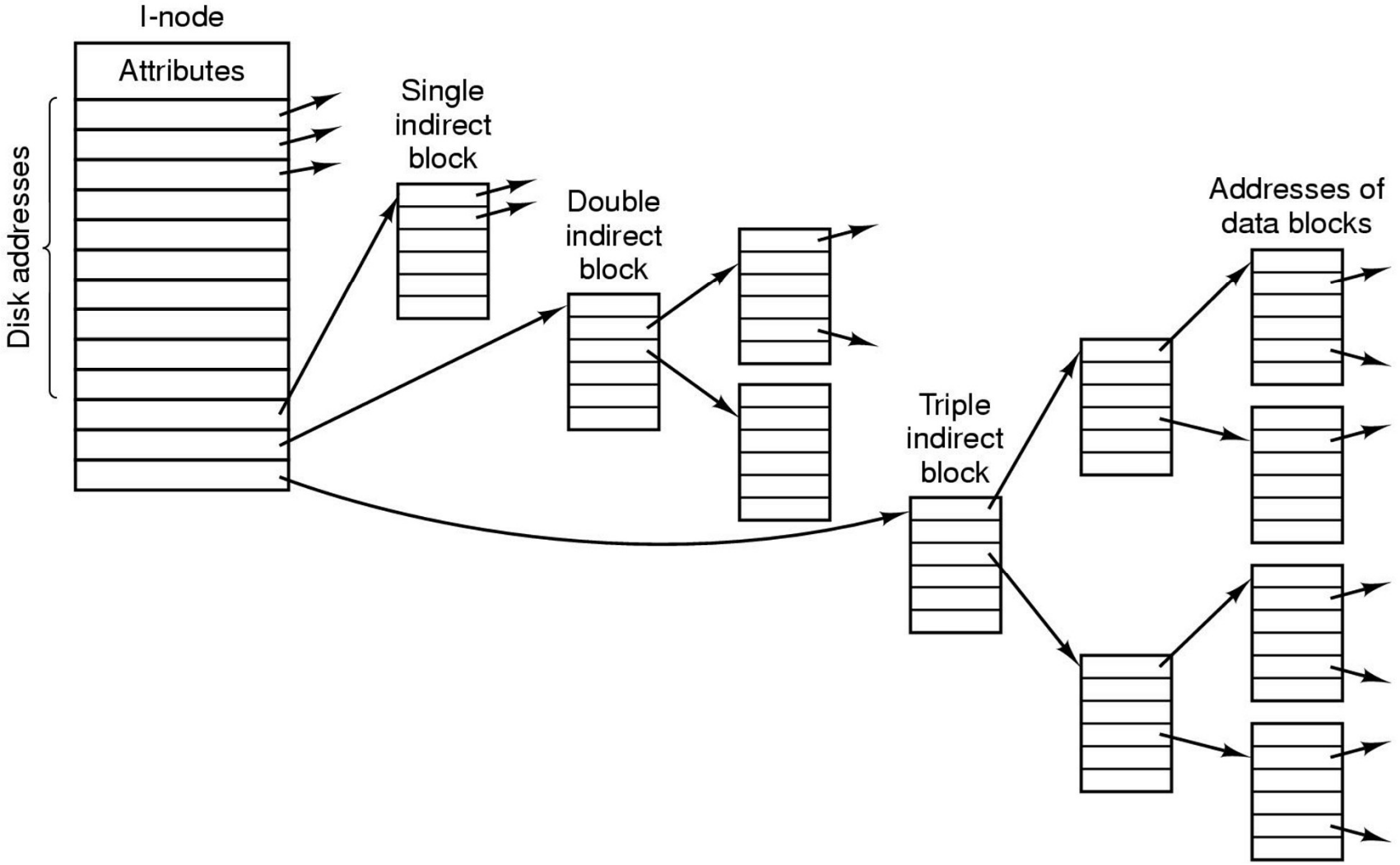
The Windows 98 File System (3)



The UNIX V7 File System (1)

- 

The UNIX V7 File System (2)



The UNIX V7 File System (3)