File Systems

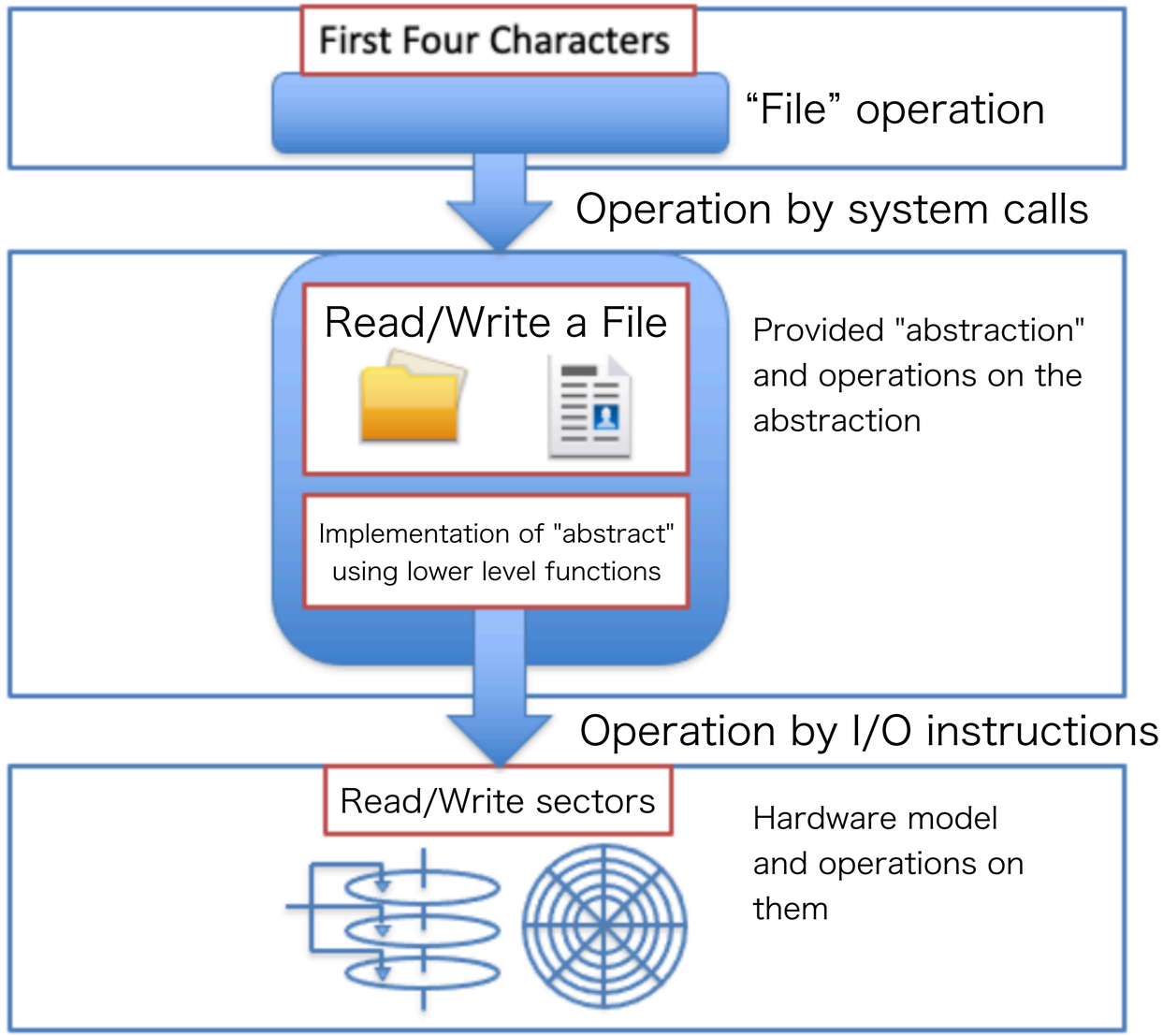
I233 Operating Systems

Abstraction — Layering of Abstraction (shown before)

User program

OS
(Filesystem,
Device Driver)

Hardware
(HDD device)



Chap. 4 File Systems (1)

- 4.1 FILES
  - 4.1.1 File Naming
  - 4.1.2 File Structure
  - 4.1.3 File Types
  - 4.1.4 File Access
  - 4.1.5 File Attributes
  - 4.1.6 File Operations
  - 4.1.7 An Example Program Using File-System Calls
- 4.2 DIRECTORIES
  - 4.2.1 Single-Level Directory Systems
  - 4.2.2 Hierarchical Directory Systems
  - 4.2.3 Path Names
  - 4.2.4 Directory Operations

Understanding the abstractions
“File” and “Directory”

Chap. 4 File Systems (2)

- 4.3 FILE-SYSTEM IMPLEMENTATION
  - 4.3.1 File-System Layout
  - 4.3.2 Implementing Files
  - 4.3.3 Implementing Directories
  - 4.3.4 Shared Files
  - 4.3.5 Log-Structured File Systems
  - 4.3.6 Journaling File Systems
  - 4.3.7 Virtual File Systems
- 4.4 FILE-SYSTEM MANAGEMENT AND OPTIMIZATION
  - 4.4.1 Disk-Space Management
  - 4.4.2 File-System Backups
  - 4.4.3 File-System Consistency
  - 4.4.4 File-System Performance
  - 4.4.5 Defragmenting Disks

Implementation of the abstractions
“File” and “Directory”

Chap. 4 File Systems (3)

- 4.5 EXAMPLE FILE SYSTEMS
  - 4.5.1 The MS-DOS File System
  - 4.5.2 The UNIX V7 File System
  - 4.5.3 CD-ROM File Systems

4.1.1 File Naming

- What kind of identifier to give to the object (file) ?
- String
  - character set
  - numbers? special characters?
- File extension
  - Make meaningful or not at the OS level

Example of File extensions

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

Figure 4-1. Some typical file extensions.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.266.

4.1.2 File Structure

- Internal structure of the “File”
  - Whether or not to have a structure that defines the rules and methods of access.
  - Bytes v.s. Records
  - Array v.s. Tree

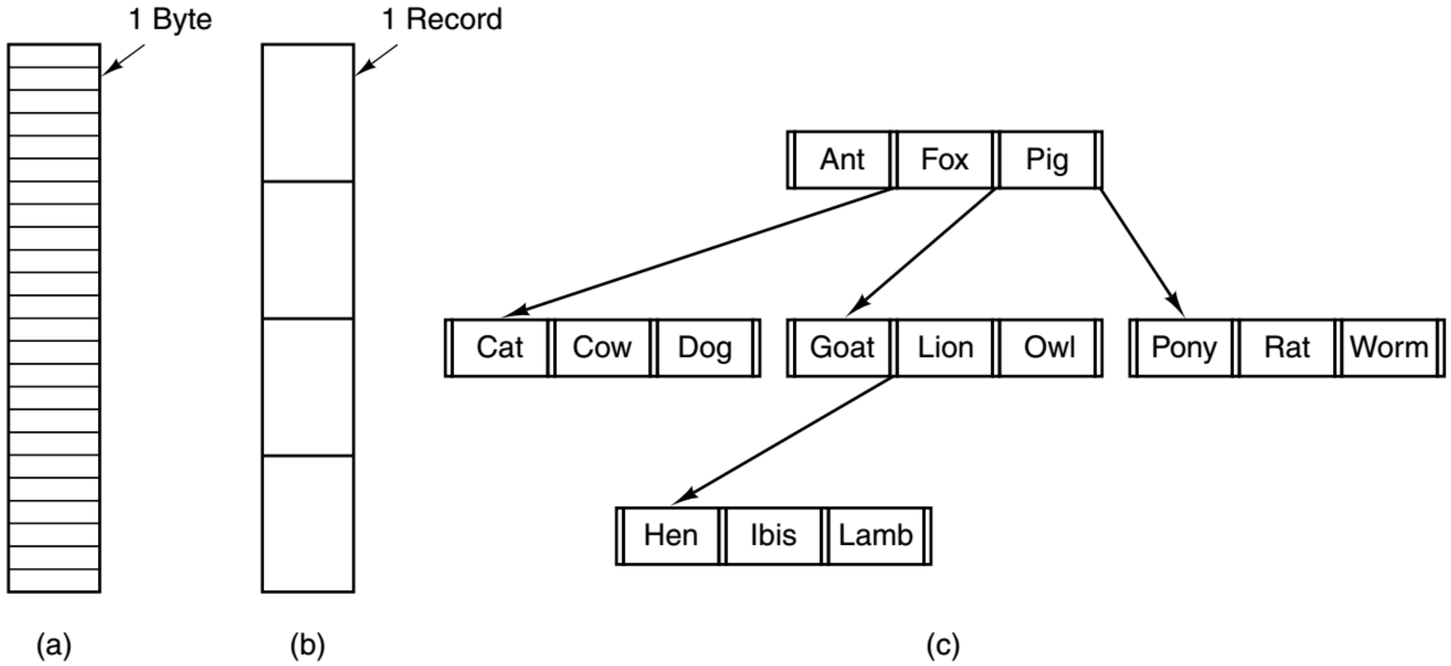


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.267.

4.1.3 File Types

- Regular files
- Special files
  - System files for storing system information
    - Directory information
  - A “name” for naming the resource on the file system.
    - Device files and IPC mechanism on UNIX
    - Network services, etc.
- Involvement of the OS in the contents of the file
  - Text (ASCII) files
  - Binary files
    - Executable binary files

Examples of binary files involving the OS

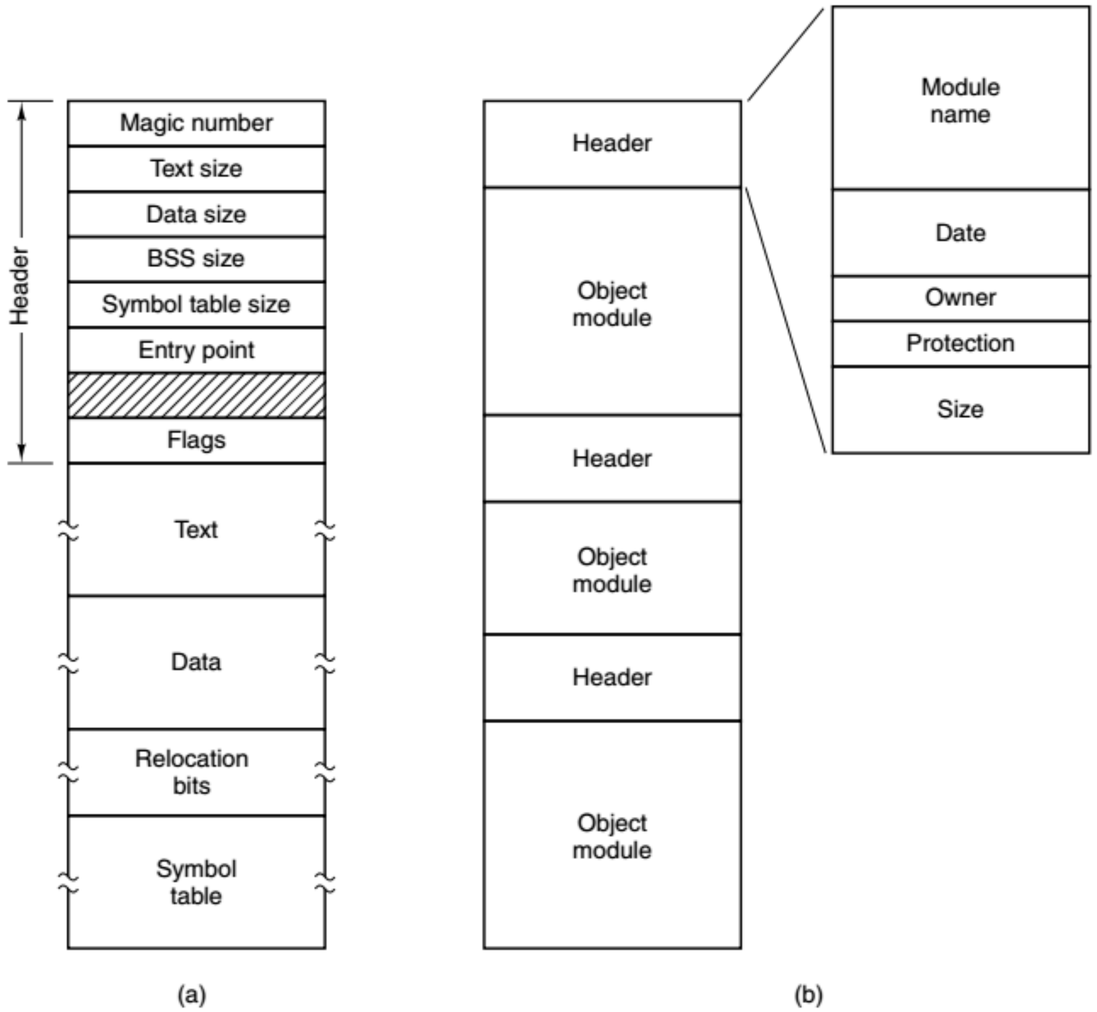


Figure 4-3. (a) An executable file. (b) An archive.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.270.

4.1.4 File Access

- Sequential access
  - A process can only access content in the order recorded in the file. It can only write in the order.
  - Concept of “rewind”
  - Convenient for magnetic tapes
- Random access
  - A process can access data at any location in a file.
  - Method of specifying the read/write position
    - Specifying explicitly in reading and writing
    - Concept of “current position” and “seek”

4.1.5 File Attributes

- Information about a file (other than its data)
  - Name (the name may also be regarded as an attribute)
  - Protection, file structure, size, type, …
- a.k.a. “metadata”

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

Figure 4-4. Some possible file attributes.

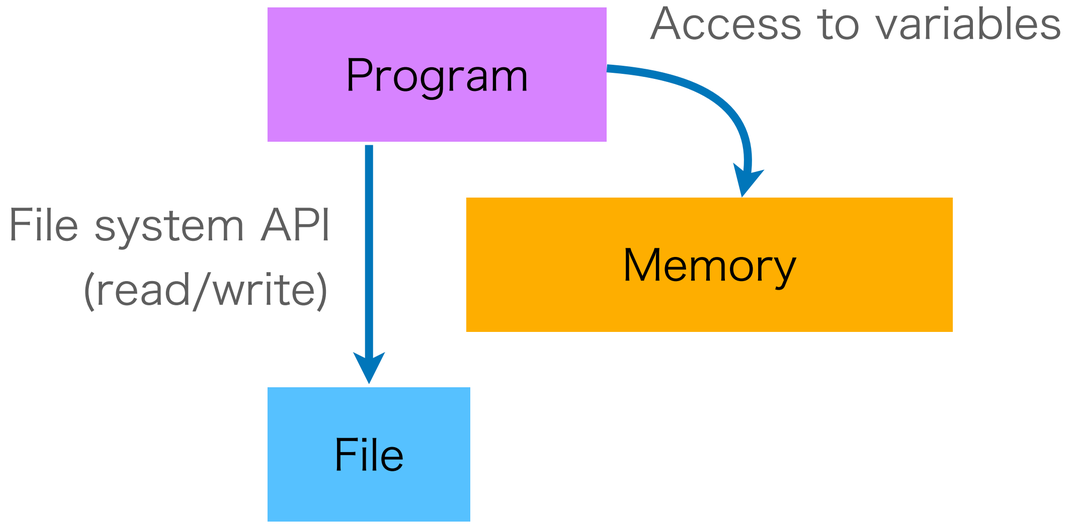
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.272.

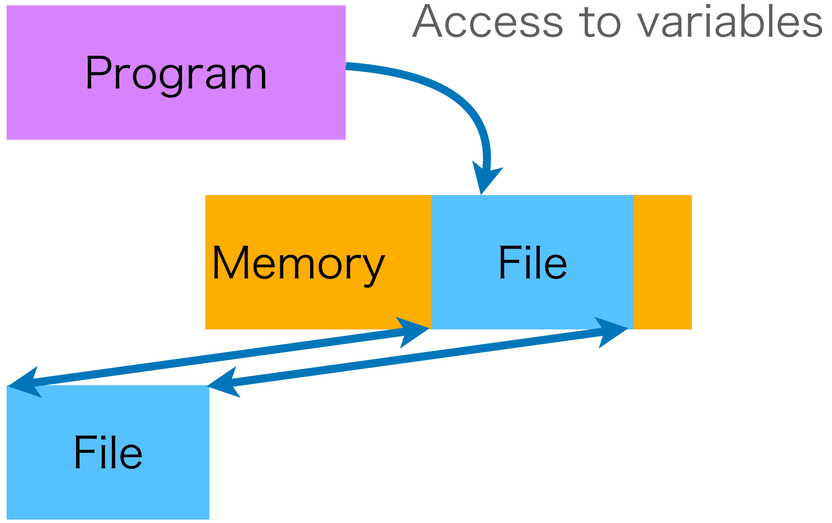
4.1.6 File Operations

- A typical example of a file handling API
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get attributes
  - Set attributes
  - Rename

Memory-Mapped Files

- Mapping file data to memory space
- Access to data in a file can be performed as access to memory, rather than through the file system API (read, write, seek, ⋯).
- Compatibility with virtual memory
- Compatibility with segmentation





4.2 Directories

- Flat v.s. Hierarchical

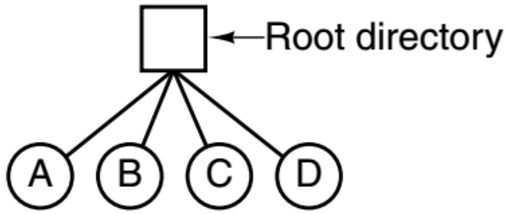


Figure 4-6. A single-level directory system containing four files.

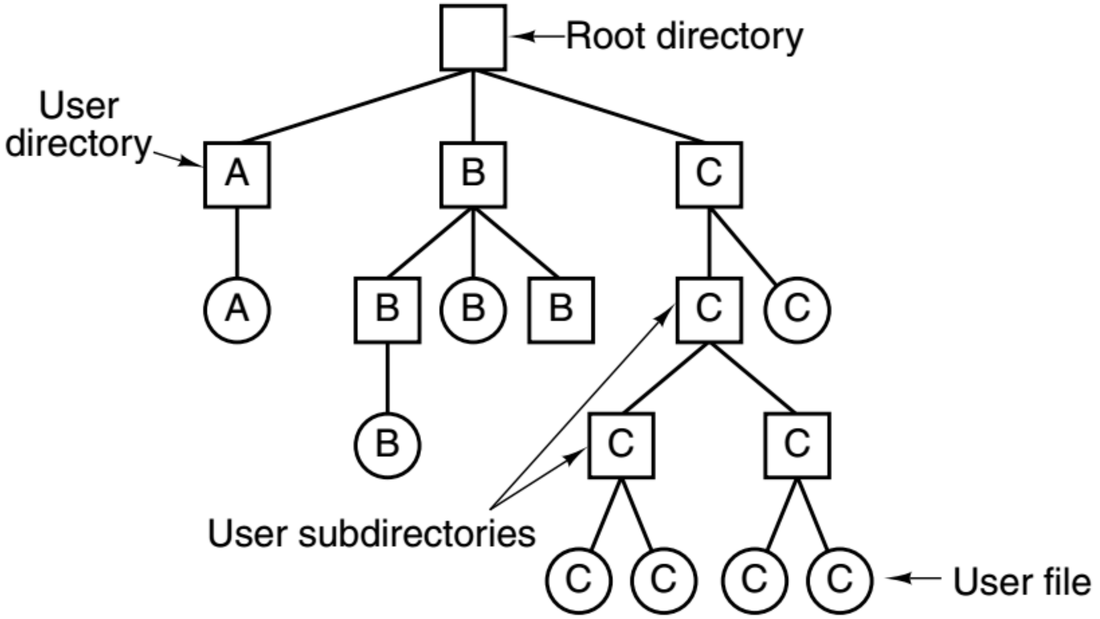


Figure 4-7. A hierarchical directory system.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.277.

4.2.3 Path Names

- The concept of identifying a specific location in a hierarchical file system.
  - If the hierarchy is not clearly identifiable, the concept of path names cannot exist either.
- Absolute path name
  - The path from the root directory to the file
- Relative path name
  - The path from (another) specific position in directory tree to the file
- Concept of “working directory” a.k.a. “current directory”
- Concept of “parent directory”, and notation

A UNIX directory tree

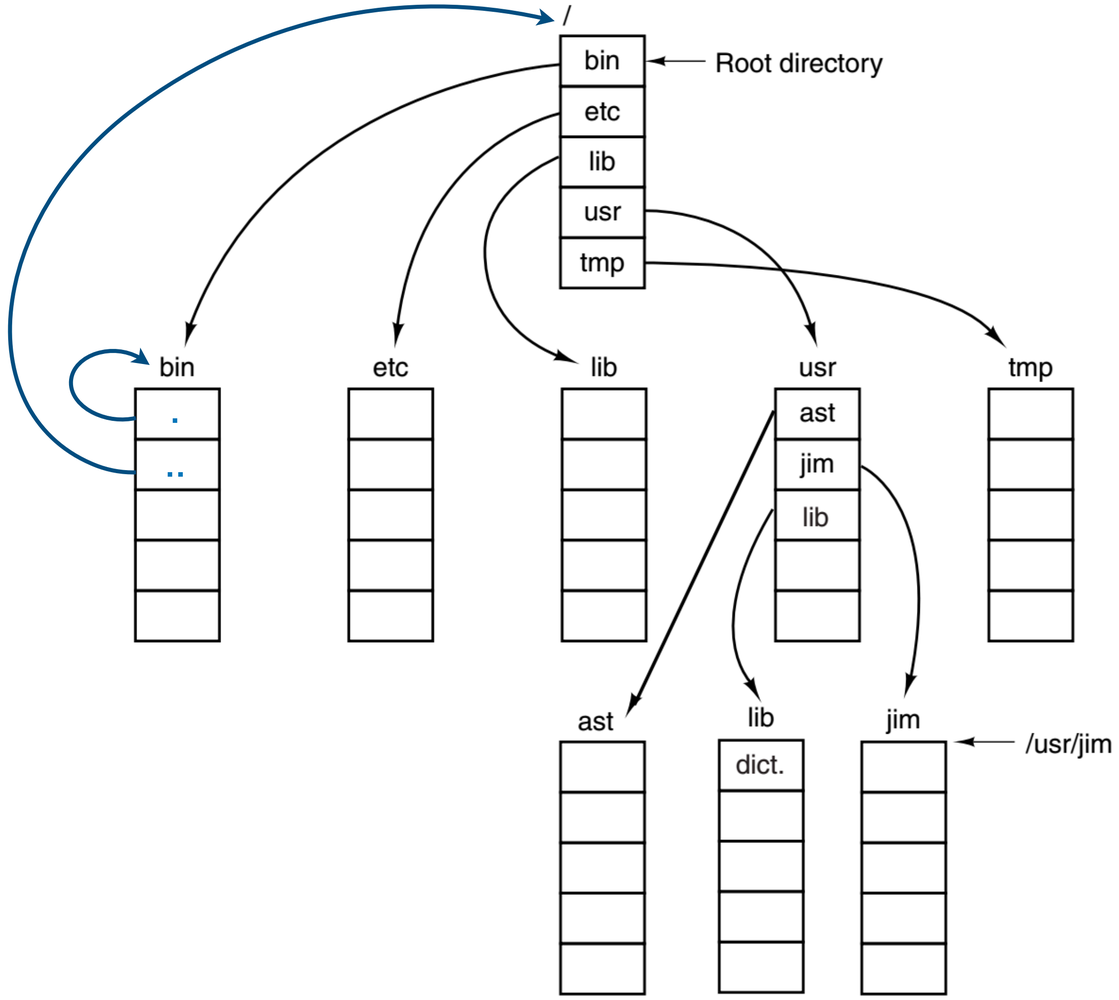


Figure 4-8. A UNIX directory tree.

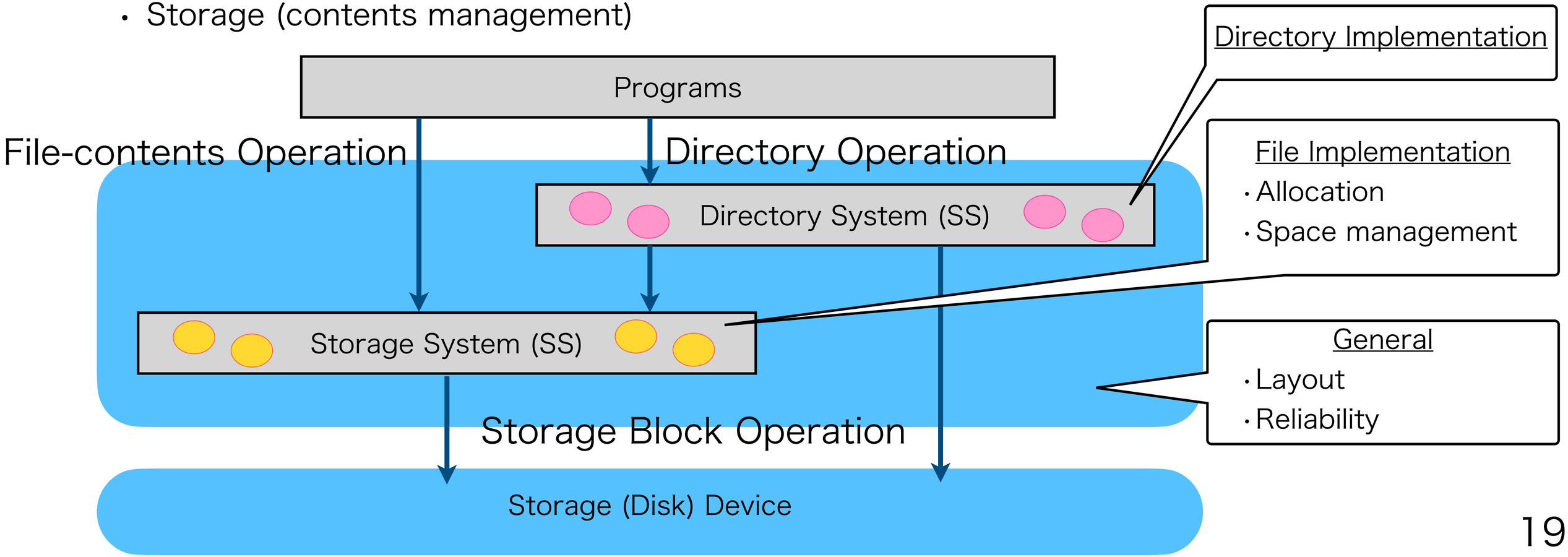
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.279.

4.2.4 Directory Operations

- A typical example of a directory handling API
  - Create
  - Delete
  - Opendir
  - Closedir
  - Readdir
  - Rename
- UNIX specific API
  - Link
  - Unlink

4.3 File-System Implementation

- The implementation of the file-system can be divided into two parts.
  - Directory (Name, Hierarchical structure, Attribute management) or Metadata
  - Storage (contents management)



4.3.1 File-System Layout

- Layout / Structure of “Basic area” on disk
  - Area for region management
  - Boot block
  - File-system

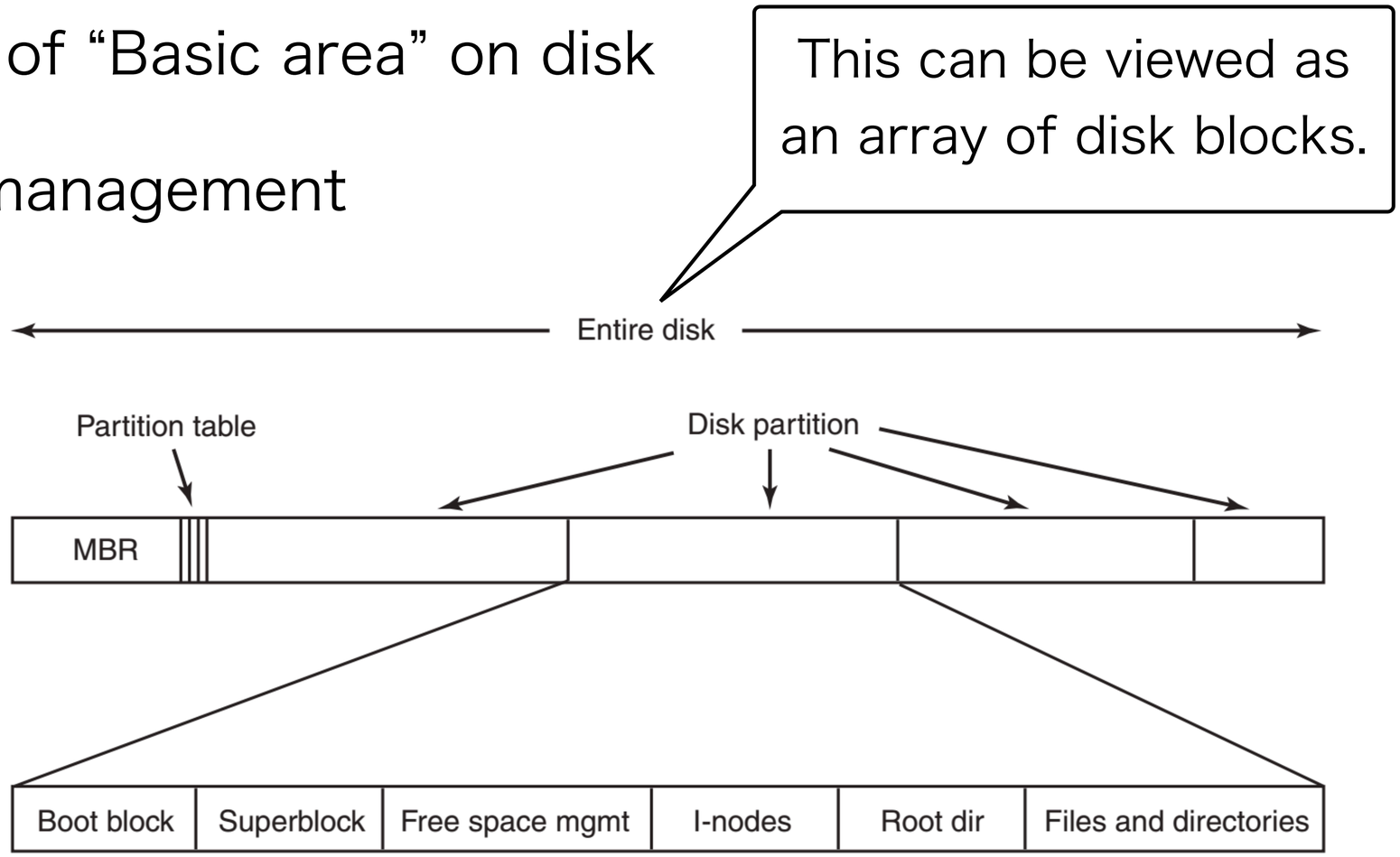


Figure 4-9. A possible file-system layout.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.282.

4.3.2 Implementing Files

- How to store each file-contents?
  - Contiguous Allocation
  - Linked-List Allocation
  - Linked-List Allocation using a Table in Memory
  - I-nodes

Contiguous Allocation

- Store each file as a contiguous run of disk blocks.
- Excellent sequential read performance
- Fragmentation of free blocks
- Block management by bitmap

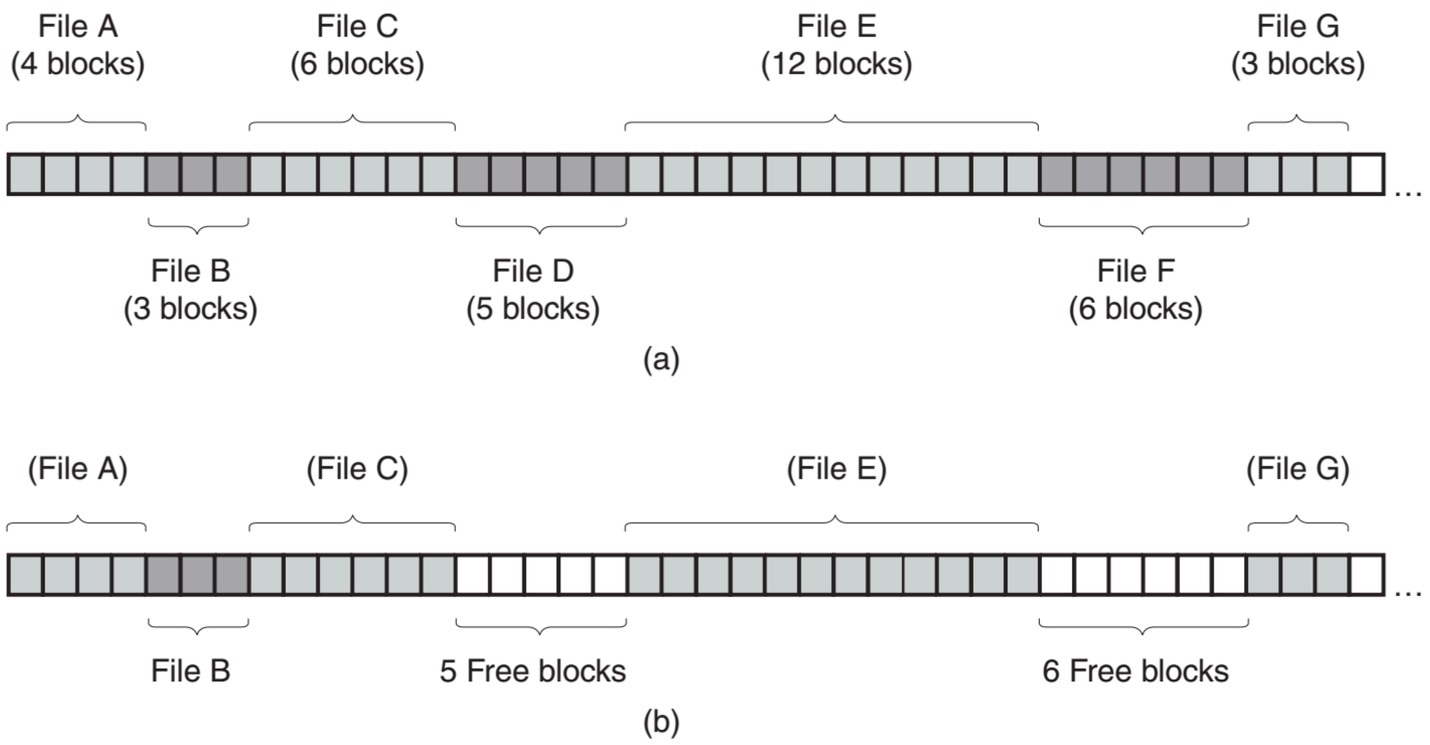


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.283.

Linked-List Allocation

- Keep each file as a linked list of disk blocks
- Also manage free blocks by linked list
- Good performance on alloc and free
- Random access is extremely slow
- Data storage in a block is no longer a power of two

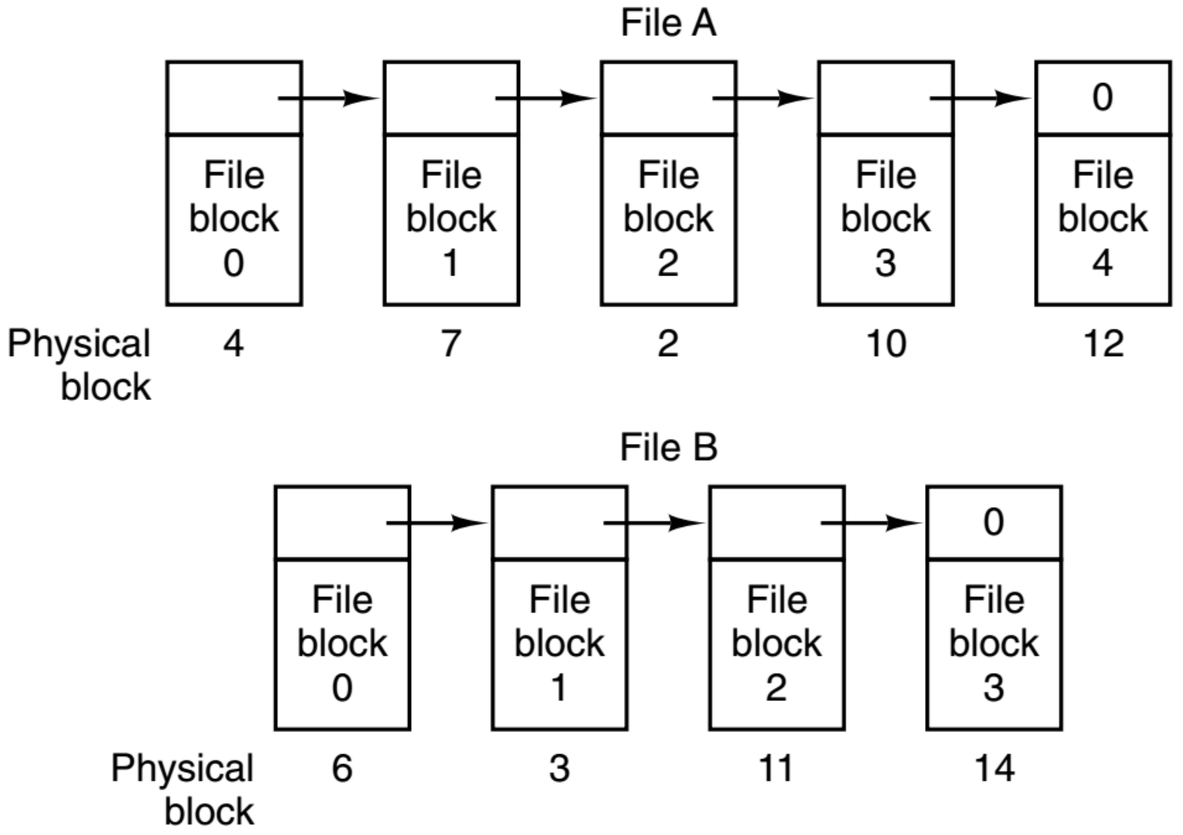


Figure 4-11. Storing a file as a linked list of disk blocks.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.285.

Linked-List Allocation using a Table in Memory

- Eliminate the disadvantages of the linked-list allocation with a pointer table in memory
- Need applying changes on table to disk

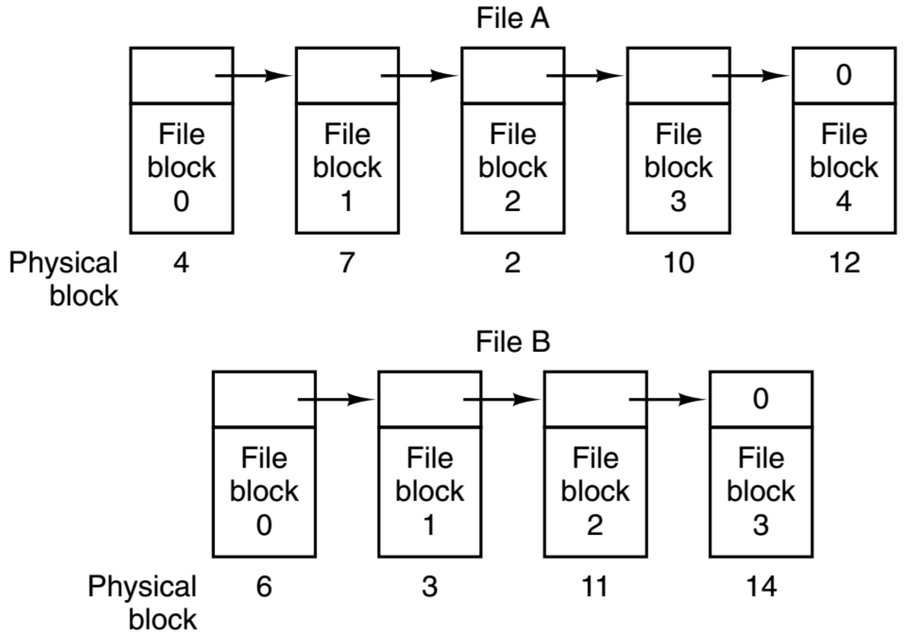


Figure 4-11. Storing a file as a linked list of disk blocks.

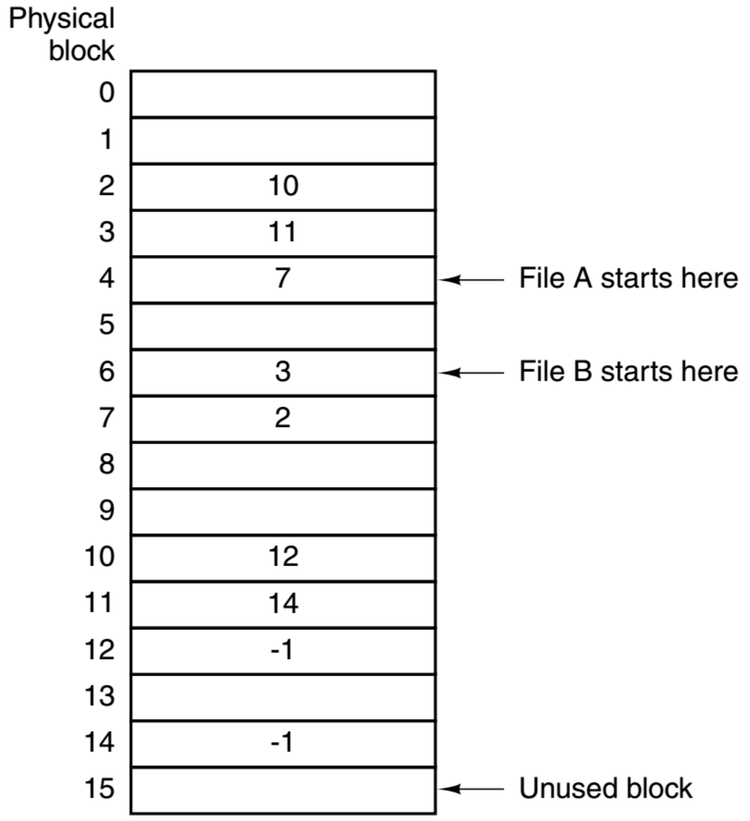


Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.285, 286.

I-nodes

- A table that is present in each file lists the blocks that make up that file.

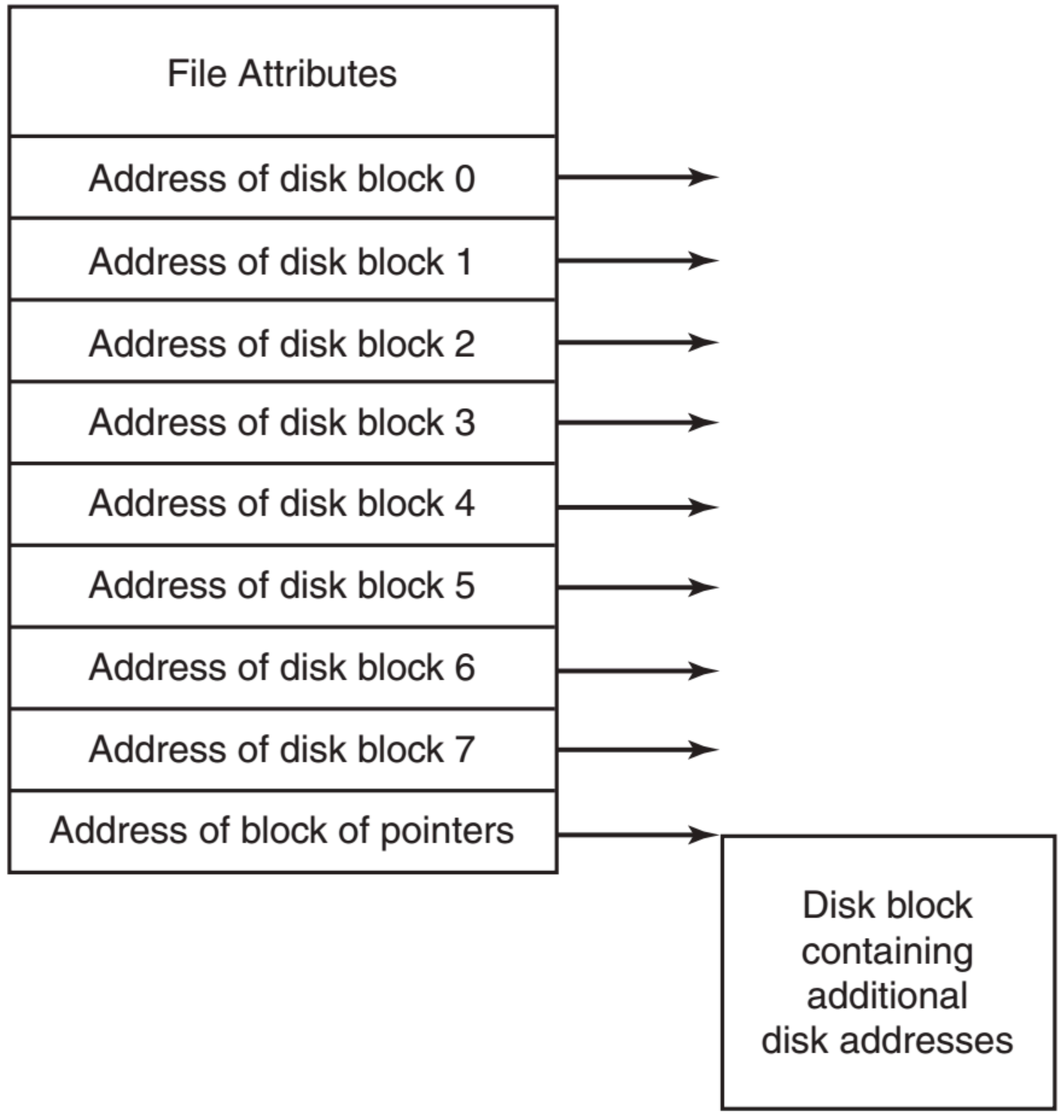


Figure 4-13. An example i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.287.

4.3.3 Implementing Directories

- Information that may be maintained in the directory
  - File Name
  - File Attribute
  - Information on the SS

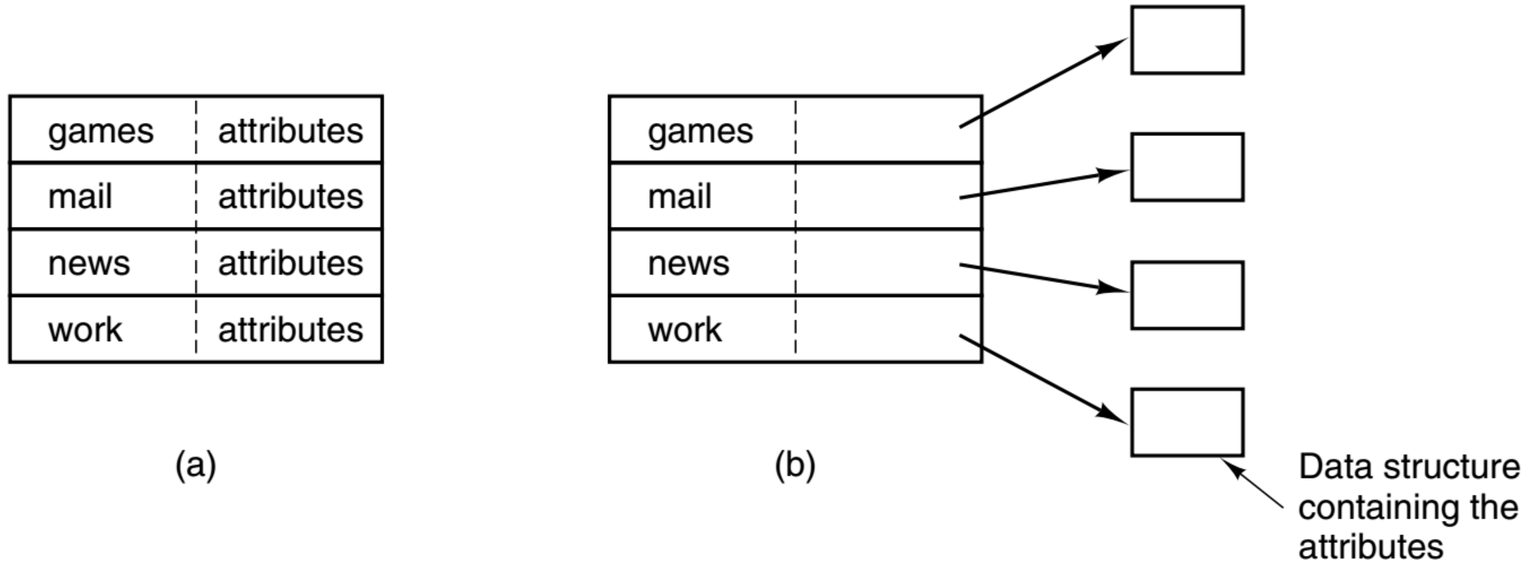


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.288.

4.3.4 Shared Files

- Referencing the same file from multiple locations in the tree
  - i-node sharing
  - Use special “LINK-type” files

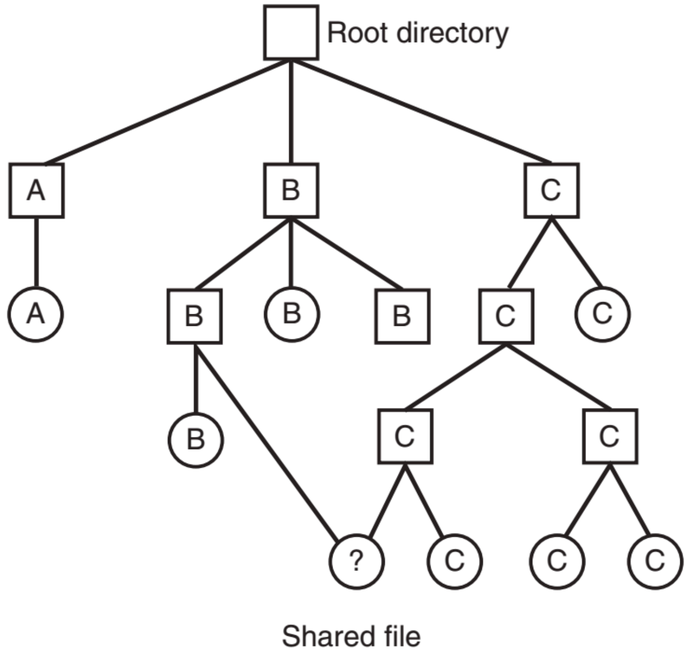


Figure 4-16. File system containing a shared file.

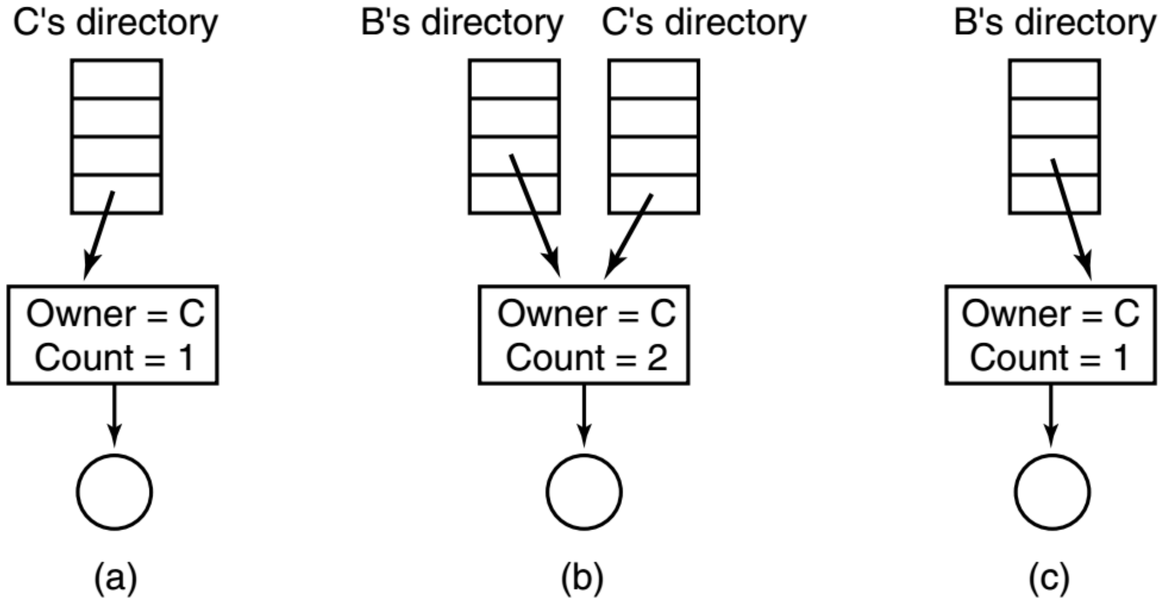


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.291, 292.

4.4.1 Disk-Space Management

- File storing strategies
  - Contiguous Storing
  - Split up into a number of contiguous blocks: Block size?
- Keeping Track of Free Blocks
  - Bitmap v.s. Linked-list
- Per-user capacity management

Determine the Block Size

- Various candidates
  - Specification of physical device (sector, truck, cylinder size)
  - Page size in paging system
- Factors
  - Disk space utilization
    - Using a block size of 32KB on a system with a median file size of 1KB wastes 96% of the total disk space.
  - Data rate
    - A small block size means that most files will span multiple blocks and thus need multiple seeks and rotational delays to read them.

Disk space utilization v.s. Data rate

- Disk space utilization
  ≒ 1 if Block size < Median file size,
  else Median file size / Block size

- Required transfer time
  ≒ No. of blocks * (Ave. seek time
  + Ave. wait time + transfer time)

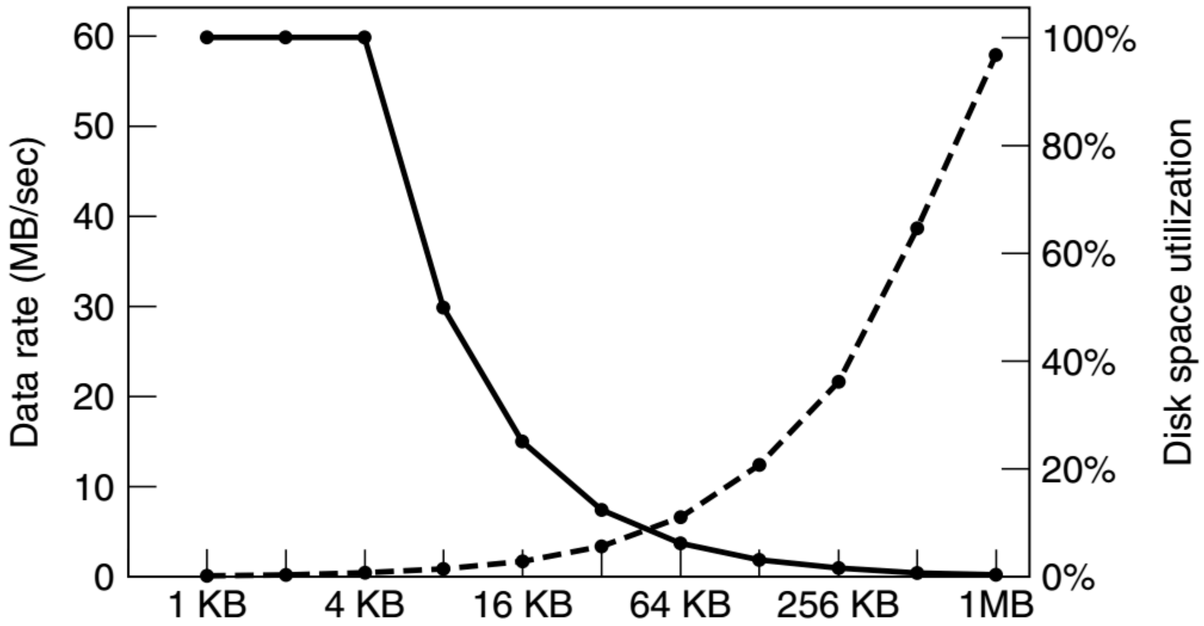


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk-space efficiency. All files are 4 KB.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.302.

Keeping Track of Free Blocks: Bitmap v.s. Linked-list

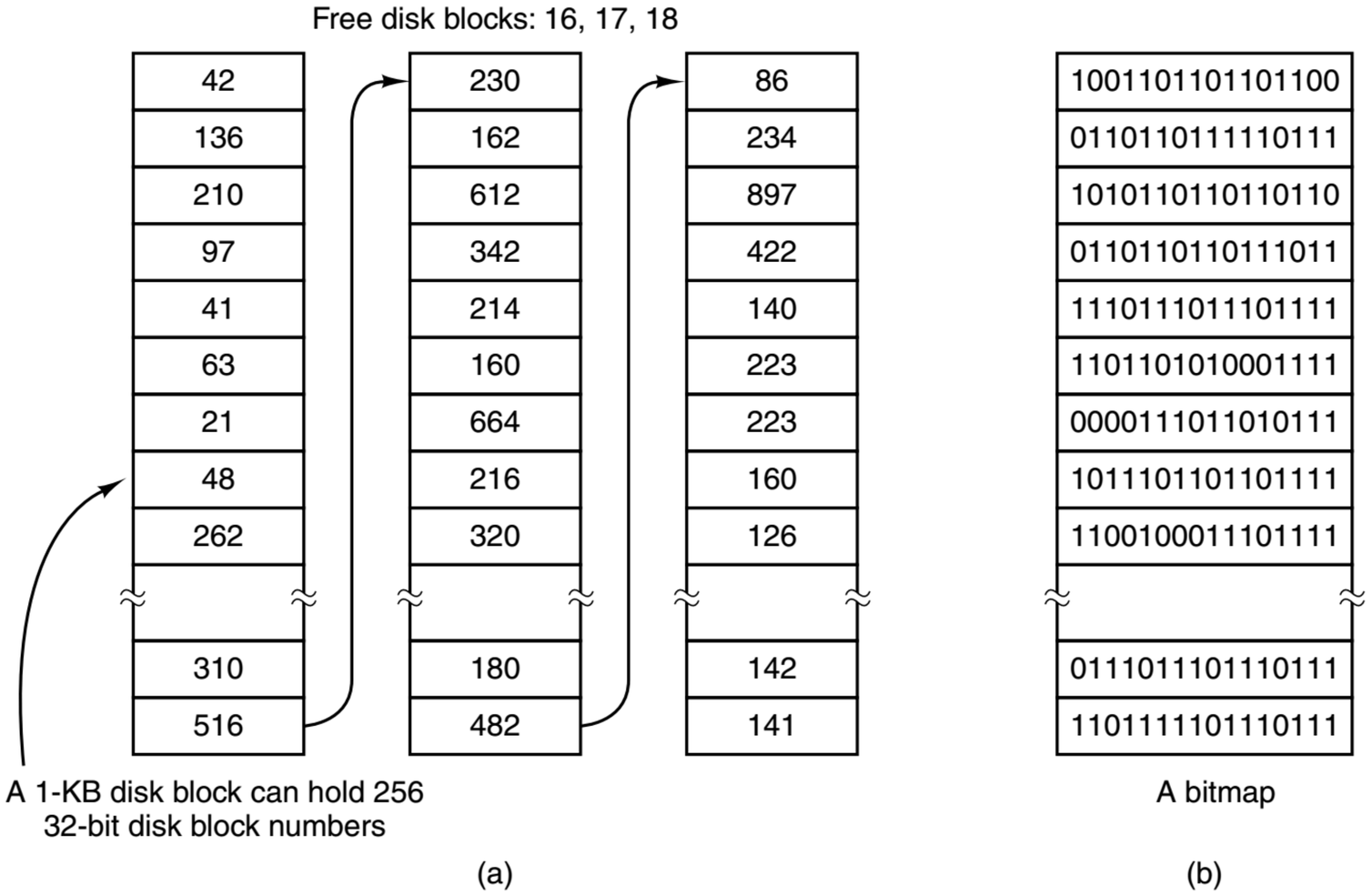


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.303.

Disk Quota

- A mechanism to manage the amount of file system usage for each user

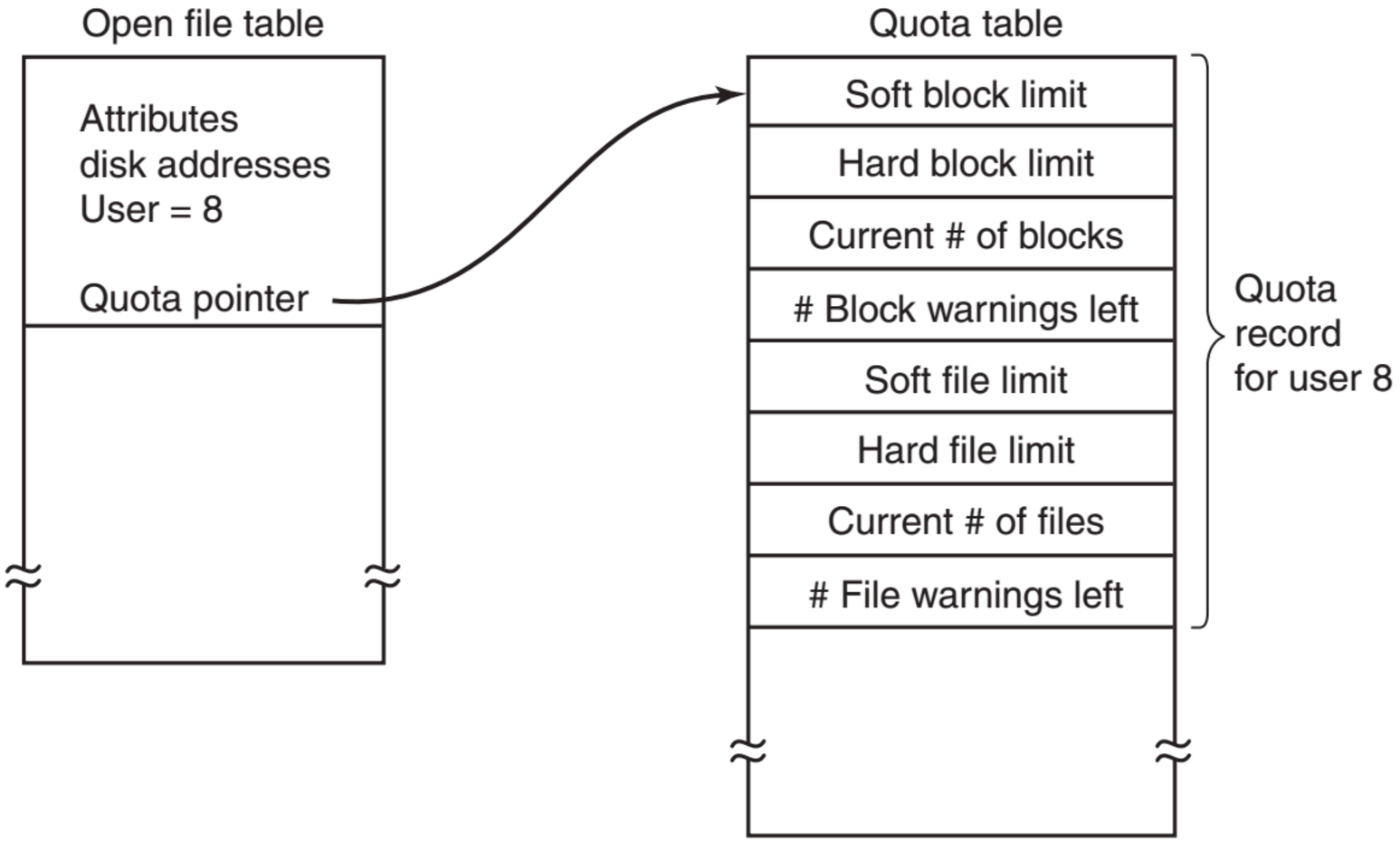


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.306.

File-System Reliability

- File system failure or corruption is much more serious than computer failure or corruption.
- The file system is capable of masking small failures (disk block failures) that occur daily.
- Need backups in case of major failures, or inspection and recovery mechanisms in case of management information inconsistencies.

4.4.2 File-System Backups

- Facts
  - Some files in the file system do not need to be backed up.
  - Files that have not been modified since the last backup do not need to be backed up. (The concept of “incremental dump”)
  - The pros and cons of compressing dump data.
  - Whether dumping is possible in online state.
  - Security issues with "copies of information" created by dumps
- Two strategies for dumping a disk to a backup disk:
  - Physical dump: Copy the entire block of a disk device as is.
  - Logical dump: Copy only the information necessary to restore the structure of the file system. (e.g. incremental dump)

Incremental dump algorithm

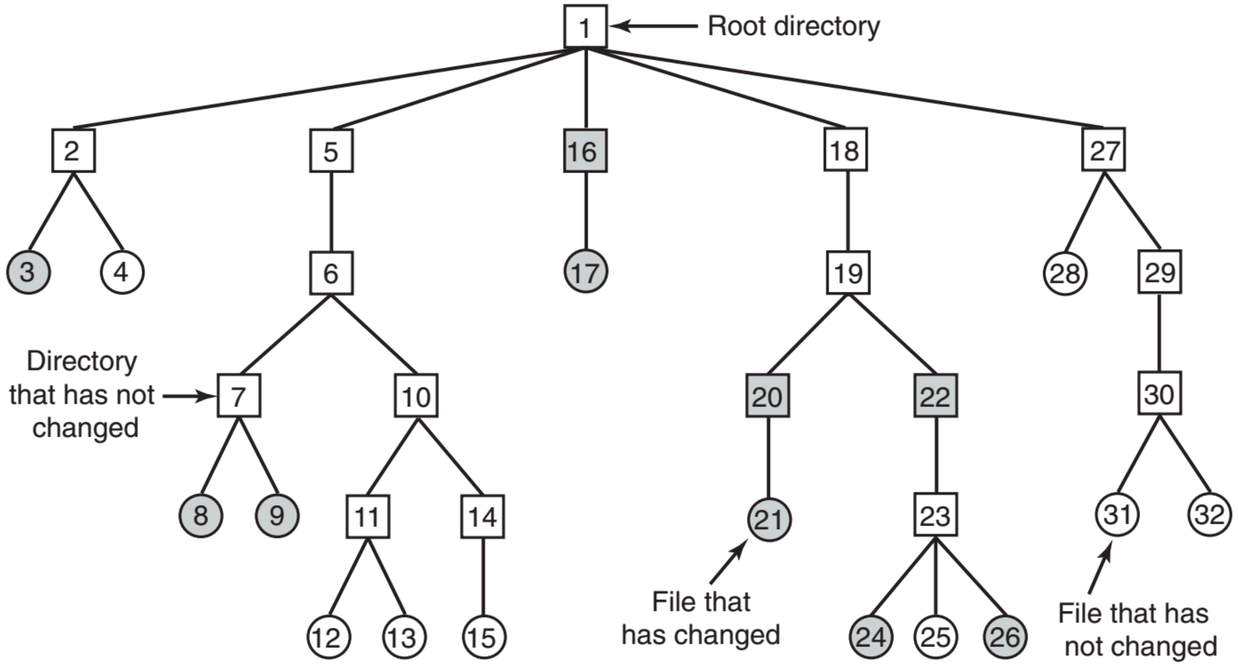


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

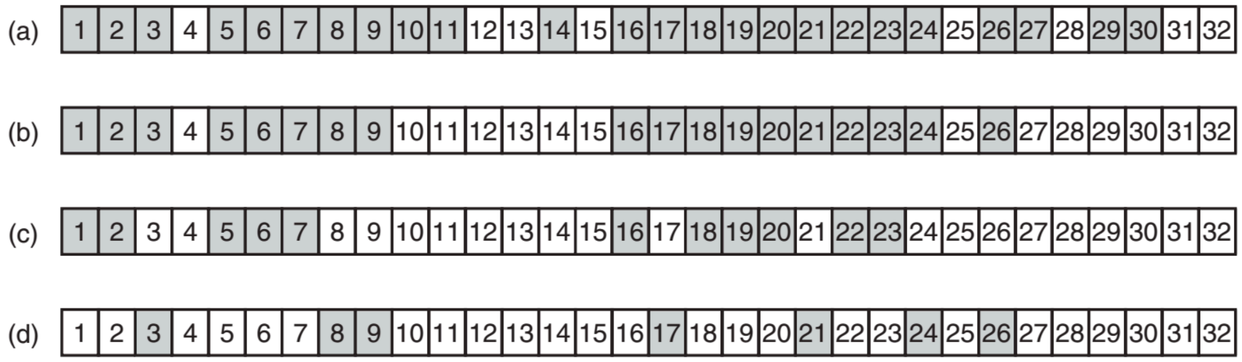


Figure 4-26. Bitmaps used by the logical dumping algorithm.

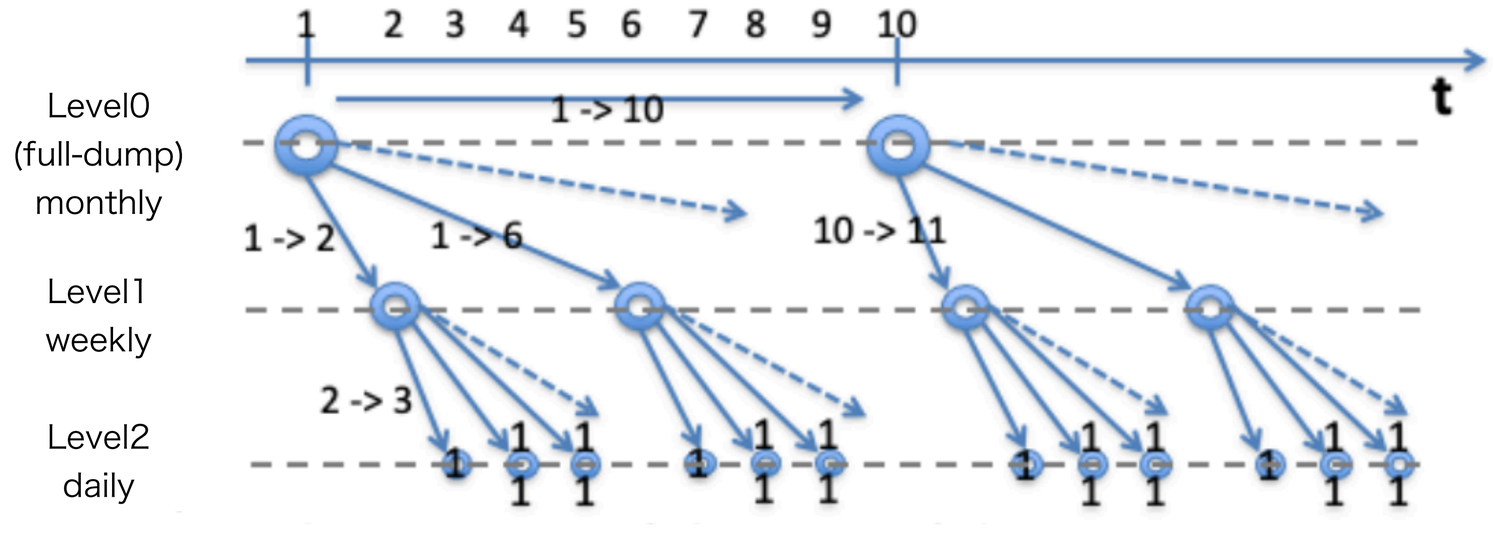
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.310, 311.

Arguments in Incremental dump

- Deleted file handling
  - In a incremental dump, can deleted files be "restored" as if they had been deleted?
- Special file handling
  - Special files representing the devices should not be dumped.
  - What about other special files (symbolic links, etc.) ?
- What to do with files that have large holes?
  - Core files often have a hole of hundreds of megabytes between the data segment and the stack. If not handled properly, each restored core file will fill this area with zeros and thus be the same size as the virtual address space (e.g., 2^32 bytes, or worse yet, 2^64 bytes).

Scheduling for incremental dump

- A series of dumps with incremental dumps can be used to restore the state of the file system to the arbitrary moment when the dump was done.
- However, it requires restoring the incremental dumps in order.
  e.g. If you are backing up every day, and you want to go back to the 700th day after you first started backing up, you will have to restore 699 incremental dump files in order.
- This problem can be solved by making careful use of the "starting point of the difference" in each incremantal dump.



For any given day, you can restore the state of the day by restoring the last level 0, the last level 1, and the level 2 of that day in this order.

4.4.3 File-System Consistency

- If the information that makes up the file system is not successfully written to disk due to some reason (such as an OS crash), the file system will be in a state of inconsistency.
- Kinds of consistency
  - Block consistency: A block is either a free block or a block that constitutes a file, and appears only once in the management structure of either.
  - File (directory) consistency: The sum of the number of references to a particular i-node from all directory entries is equal to the link-count stored in that i-node.
- Examples of inconsistencies
  - There is an i-node that is not referenced from anywhere.
  - I-node's link count (number of references) is larger than it actually is.
  - There is a free block that are not in the free bitmap.
  - A free block in the free bitmap is also used as part of the file.
  - Incorrect value in superblock.

Example of inconsistency

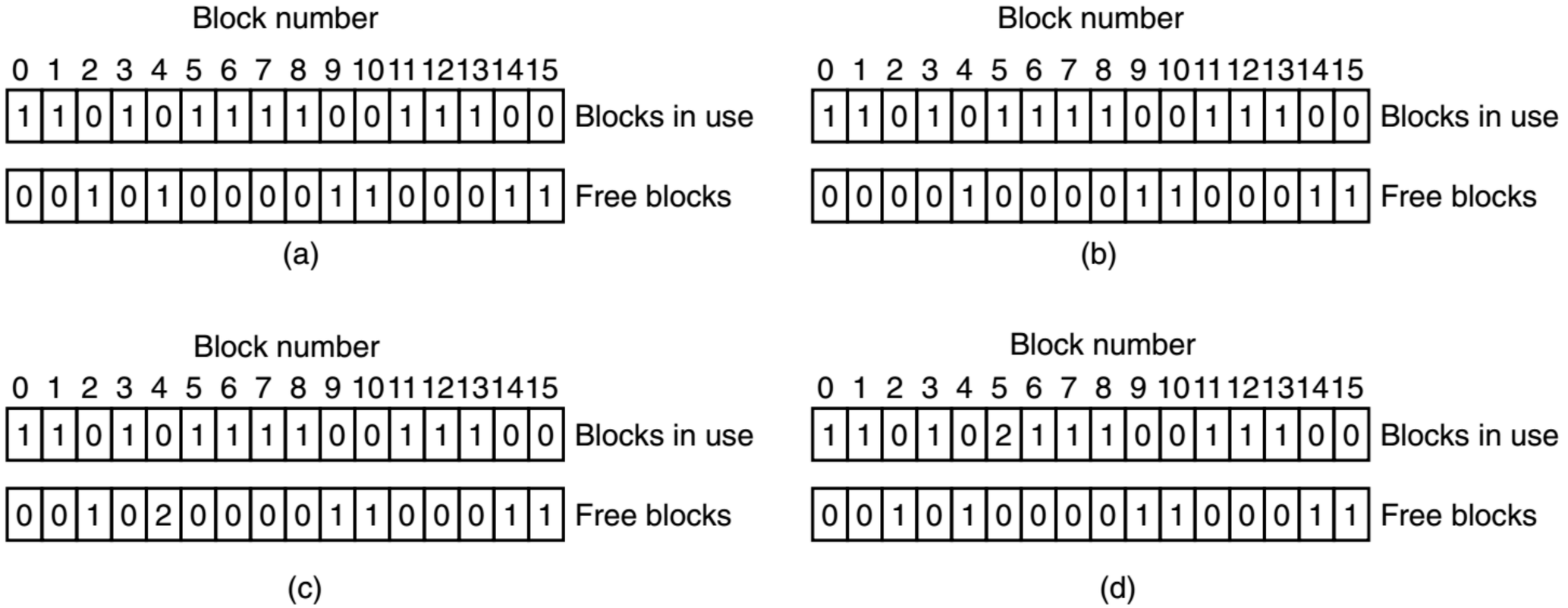


Figure 4-27. File-system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.313.

Examples of Consistency Checking and Repairing

- Block consistency checking
  - Two tables, each one containing a counter for each block, initially set to 0. The counters in the first table keep track of how many times each block is present in a file; the counters in the second table record how often each block is present in the free list.
  - Scan all i-nodes and increase by one the corresponding element in the "Blocks in Use" table for all blocks traced from that i-node.
  - Scan the free space management structure and increment the corresponding element of the "Free Blocks" table by one for each block that is managed as free.
  - When the above operation is completed, all blocks must have a 1 in either “Blocks in use" or “Free Blocks”.
- Repairing example on duplicate data blocks
  - Allocate a free block, copy the contents of duplicated block into it, and insert the copy into one of the files.
  - This will fix the duplicate block condition.
  - The file-system structure is at least made consistent, however whether the contents (semantics) of the file are safe or not needs to be inspected separately.

4.4.4 File-System Performance

- Block cache (buffer cache)
- Block Read Ahead
  - Sequential-access
- Reducing Disk-Arm Motion
  - Putting blocks that are likely to be accessed in sequence close to each other

Miscellaneous File-systems

- 4.3.5 Log-Structured File Systems
  - Reimplement the UNIX file system in such a way as to achieve the full bandwidth of the disk, even in the face of a workload consisting in large part of small random writes.
  - The basic idea is to structure the entire disk as a great big log.
  - But, not widely used, in part due to their being highly incompatible with existing file systems.
- 4.3.6 Journaling File Systems
  - One of the ideas inherent in them, robustness in the face of failure, can be easily applied to more conventional file systems.
  - Keep a log of what the file system is going to do before it does it, so that if the system crashes before it can do its planned work, upon rebooting the system can look in the log to see what was going on at the time of the crash and finish the job.

4.5.1 The MS-DOS File System

- a.k.a. “FAT” file-system
- Adopting a hierarchical directory
- Isolate block management from directory entrie
- The "next block" table indexed by block number

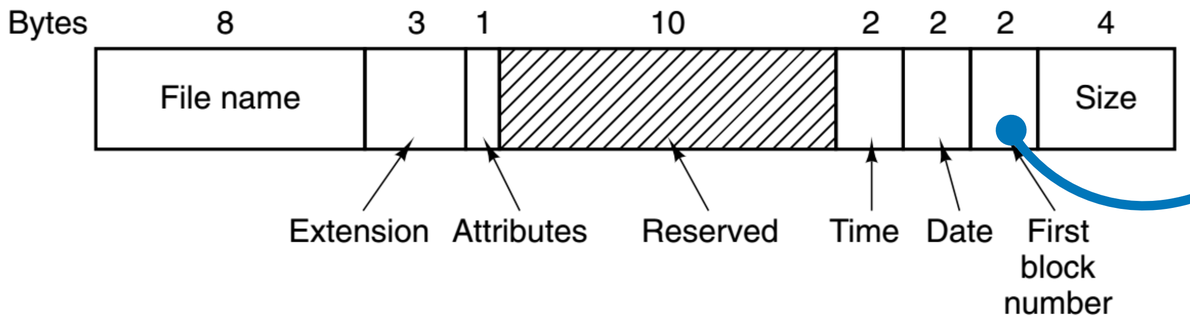


Figure 4-30. The MS-DOS directory entry.

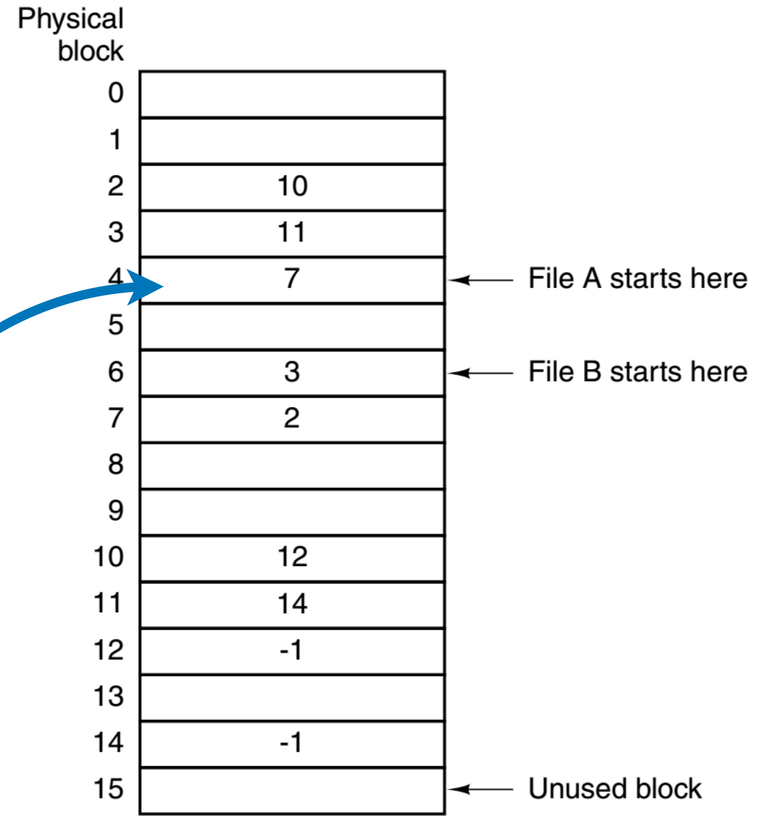


Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.321, 286.

4.5.2 The UNIX V7 File System

- Separation of directory entries (managing only names) and i-nodes (managing file entities and their attributes)

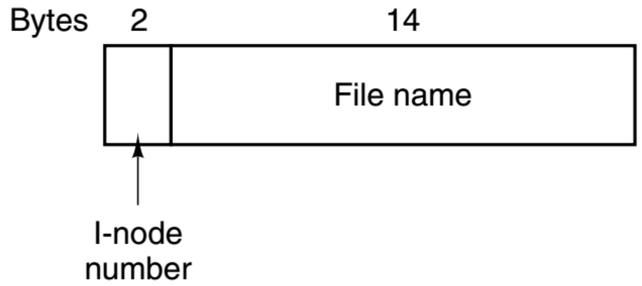


Figure 4-32. A UNIX V7 directory entry.

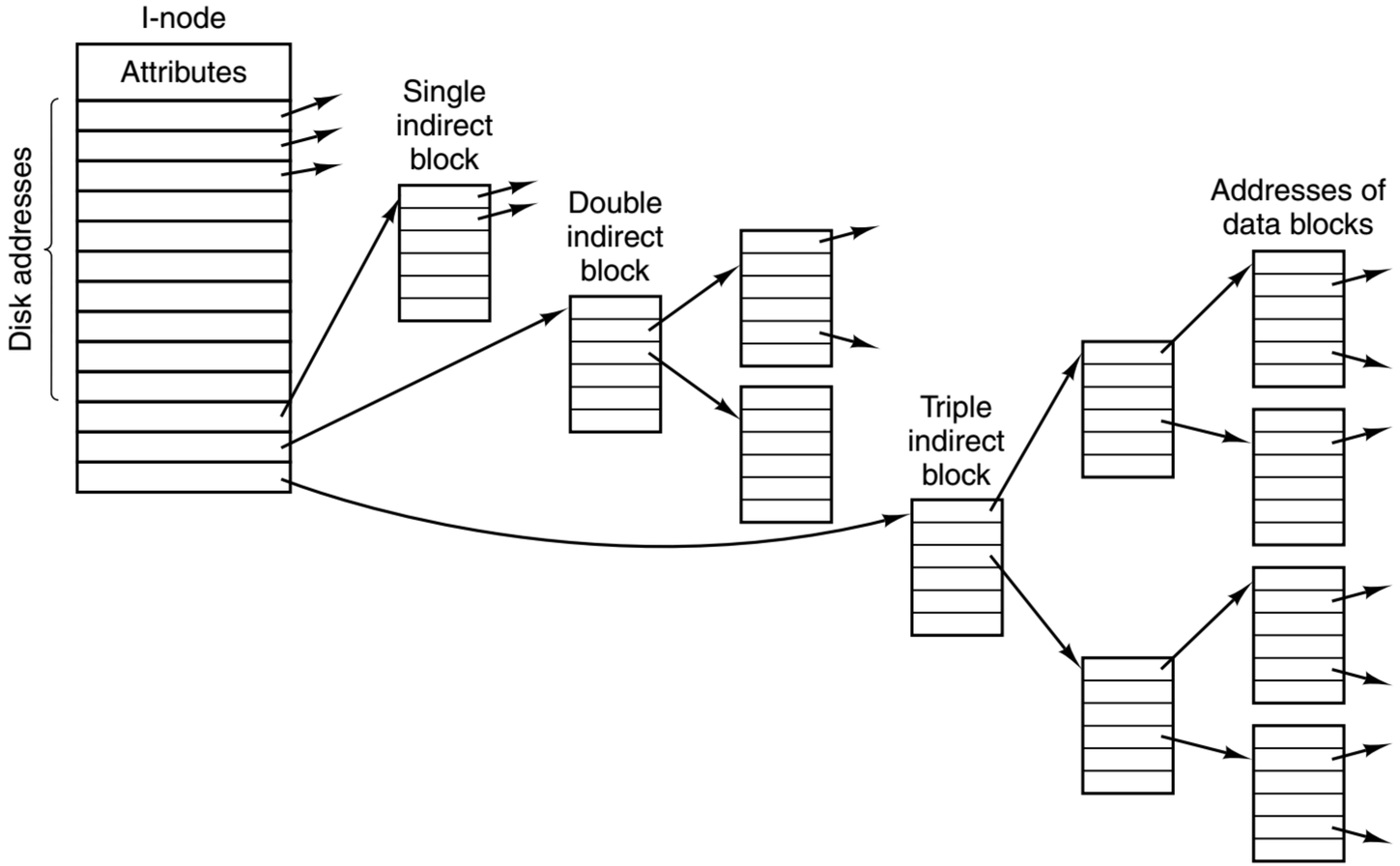


Figure 4-33. A UNIX i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.324, 325.

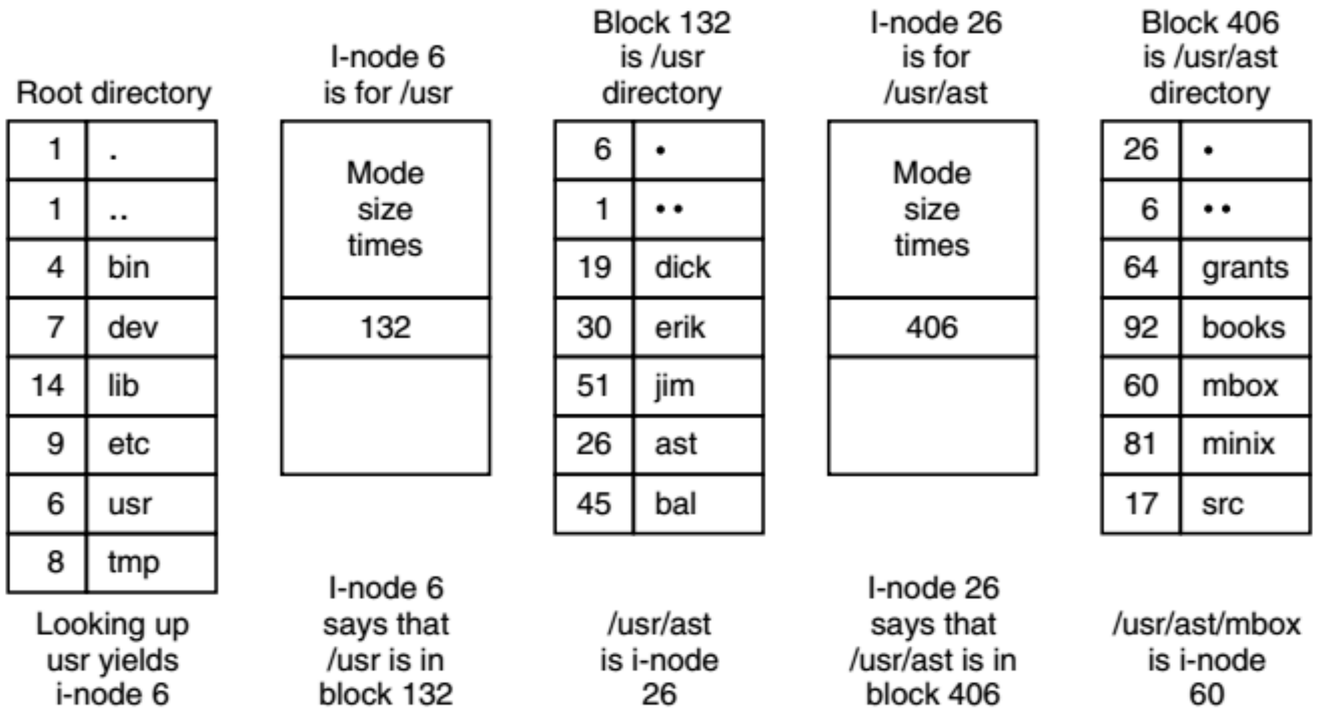


Figure 4-34. The steps in looking up */usr/ast/mbox*.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.326.

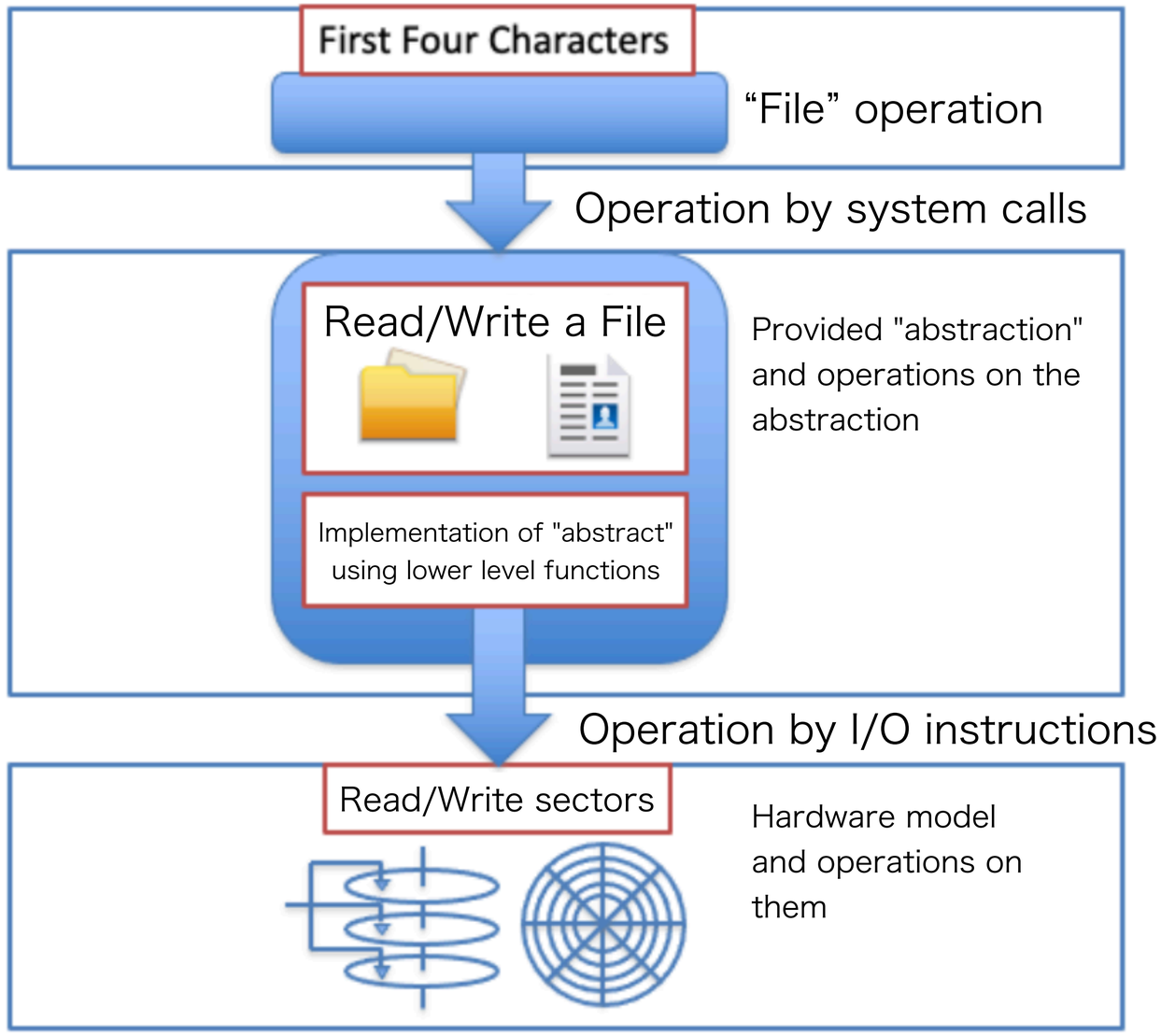
日本語資料

抽象化とは — 抽象化のレイヤリング (再々掲)

User program

OS
(Filesystem,
Device Driver)

Hardware
(HDD device)



Chap. 4 ファイルシステム (1)

- 4.1 ファイル
  - 4.1.1 ファイルの名前付け
  - 4.1.2 ファイル構成
  - 4.1.3 ファイルの種類
  - 4.1.4 ファイルアクセス
  - 4.1.5 ファイル属性
  - 4.1.6 ファイル操作
  - 4.1.7 例題プログラム
- 4.2 ディレクトリ
  - 4.2.1 1 階層ディレクトリシステム
  - 4.2.2 階層型ディレクトリシステム
  - 4.2.3 パス名
  - 4.2.4 ディレクトリ操作

抽象「ファイル」「ディレクトリ」の理解

Chap. 4 ファイルシステム (2)

- 4.3 ファイルシステムの実装
  - 4.3.1 ファイルシステムの配置
  - 4.3.2 ファイルの実装
  - 4.3.3 ディレクトリの実装
  - 4.3.4 共有ファイル
  - 4.3.5 ログ階層化ファイルシステム
  - 4.3.6 ジャーナリングファイルシステム
  - 4.3.7 仮想ファイルシステム
- 4.4 ファイルシステムの管理と最適化
  - 4.4.1 ディスク空間管理
  - 4.4.2 ファイルシステムのバックアップ
  - 4.4.3 ファイルシステムの一貫性
  - 4.4.4 ファイルシステムの性能
  - 4.4.5 分断化の解消
  -

抽象「ファイル」「ディレクトリ」の実装

Chap. 4 ファイルシステム (3)

- 4.5 ファイルシステムの例
  - 4.5.1 MS-DOS ファイルシステム
  - 4.5.2 UNIX V7 ファイルシステム
  - 4.5.3 CD-ROM ファイルシステム

4.1.1 ファイルの名前付け

- 対象物(ファイル)にどのような識別子を与えるか。
- 文字列
  - 文字セット
  - 数字・特殊文字
- 拡張子とその意味
  - OS レベルで意味を持たせる場合・持たせない場合

ファイル拡張子の例

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

Figure 4-1. Some typical file extensions.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.266.

4.1.2 ファイル構成 (ファイル編成)

- ファイルの内部構造
  - アクセスの規則・方法を定めた構造を持つか持たないか
  - バイト v.s. レコード
  - 列 v.s. 木構造

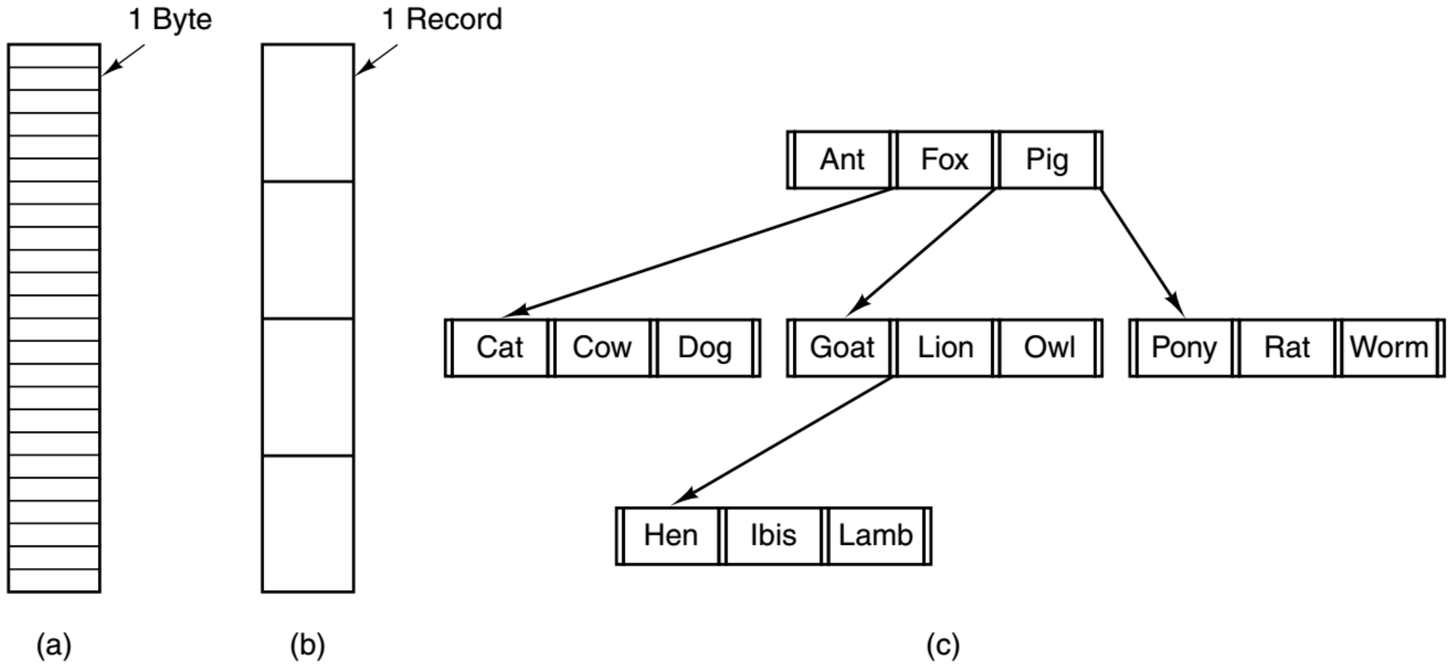


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.267.

4.1.3 ファイルの種類

- 通常ファイル (Regular files)
- 特殊ファイル
  - システム管理用の情報
    - ディレクトリ情報
  - 資源への名前付けをファイルシステム上で行うための「名前」
    - *nix システムにおけるデバイスファイルやプロセス間通信機構
    - 実験的なシステム (e.g. Plan 9) におけるネットワークサービス
- ファイル内部の情報に対する OS の関与
  - テキストファイル
  - バイナリファイル
    - 実行可能形式

OS が関与するバイナリファイルの例

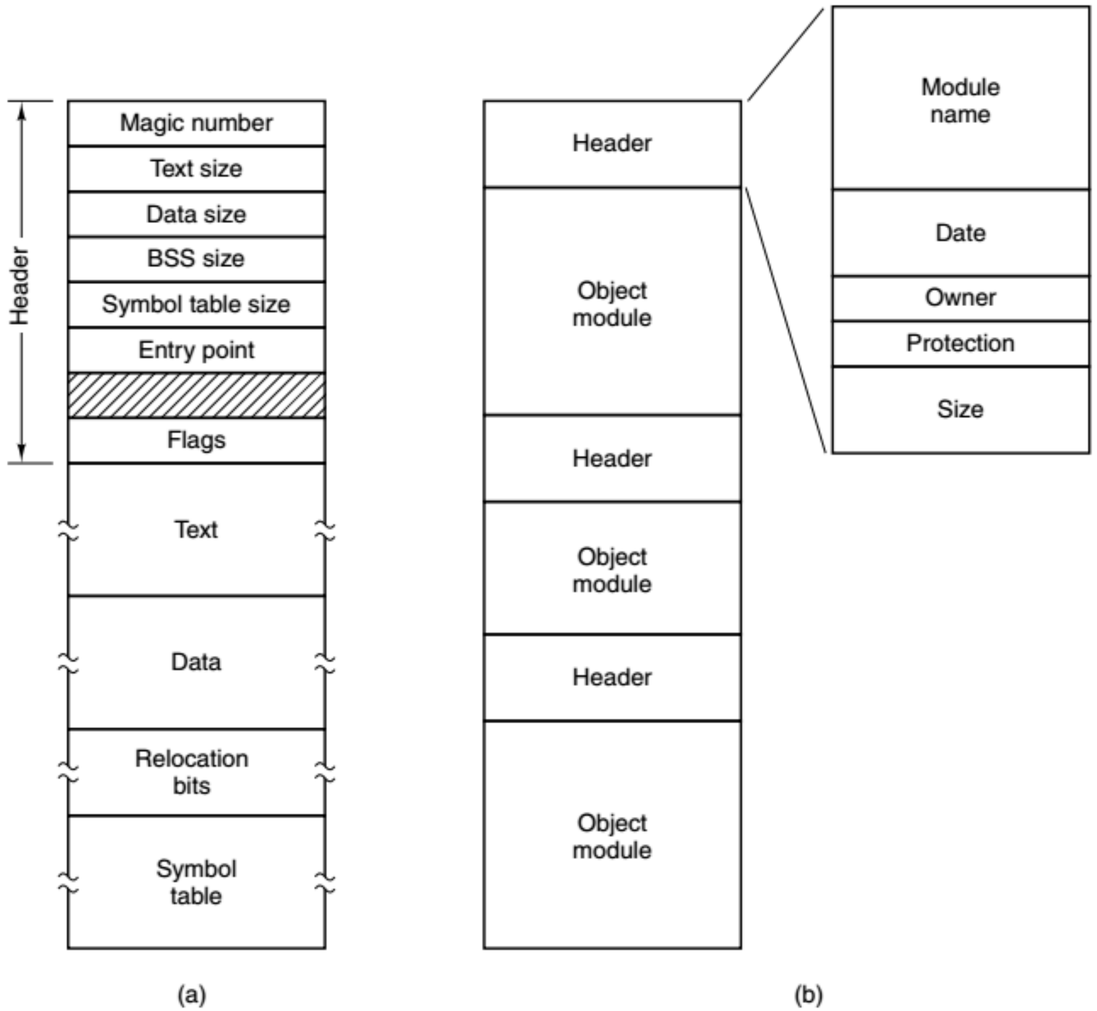


Figure 4-3. (a) An executable file. (b) An archive.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.270.

4.1.4 ファイルアクセス

- 順アクセス
  - ファイルに記録されている順番でしか内部情報にアクセスすることができない・順番にしか書き込むことができない。
  - 巻き戻しの概念
  - 磁気テープ装置が示唆するアクセス方法
- ランダムアクセス
  - ファイル内の任意の位置にあるデータを読み書きできる。
  - 読み書き位置の指定方法。
    - 読み書きにおいて明示的に指定する方法
    - 「読み書き位置」の概念と「シーク」の概念

4.1.5 ファイルの属性

- 特定のファイルに関する情報 (データ以外)
  - 名前 (名前を属性と見なすこともある)
  - 保護情報、ファイル構成に関する情報、サイズ、種類、…
- 「メタデータ」とも呼ばれる

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

Figure 4-4. Some possible file attributes.

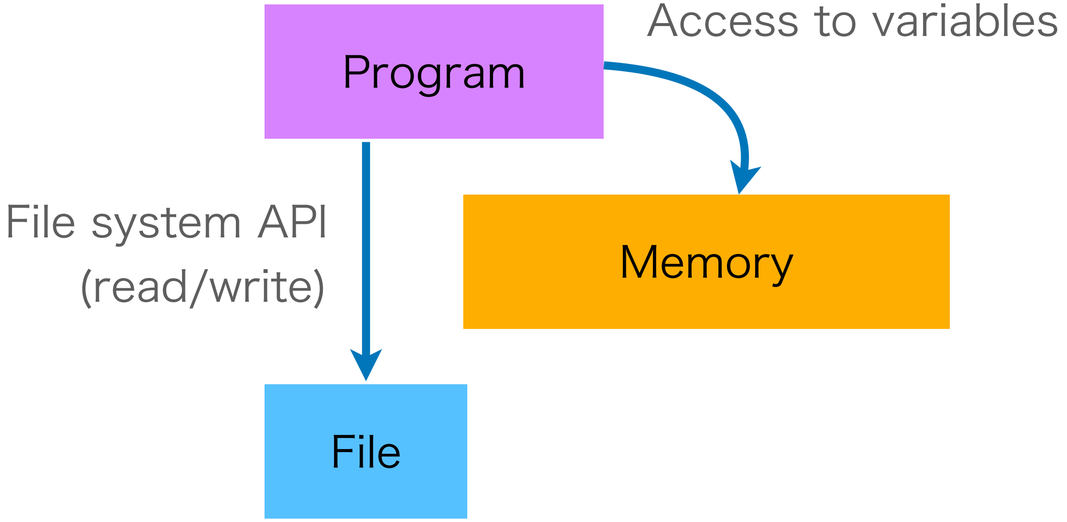
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.272.

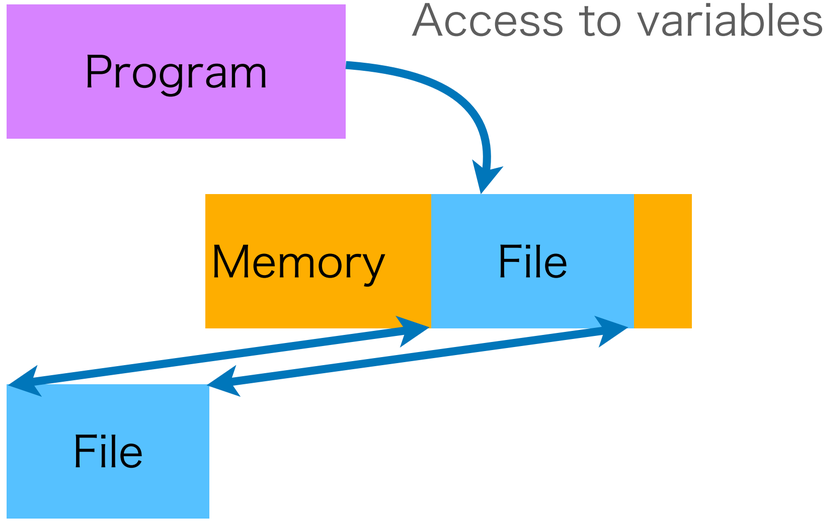
4.1.6 ファイル操作

- ファイル操作 API の典型例
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get attributes
  - Set attributes
  - Rename

ファイルのメモリへのマップ (メモリマップドファイルシステム)

- ファイルの内容をメモリ空間に割り当ててしまう操作。
- ファイル内の情報へのアクセスが、ファイルシステム API (read, write, seek, …) ではなく、メモリへのアクセスとして実行できる。
- 仮想メモリとの相性
- セグメンテーション方式との相性





4.2 ディレクトリ

- フラット v.s. ユーザ数だけのフラット v.s.階層型

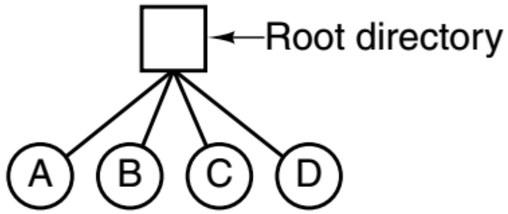


Figure 4-6. A single-level directory system containing four files.

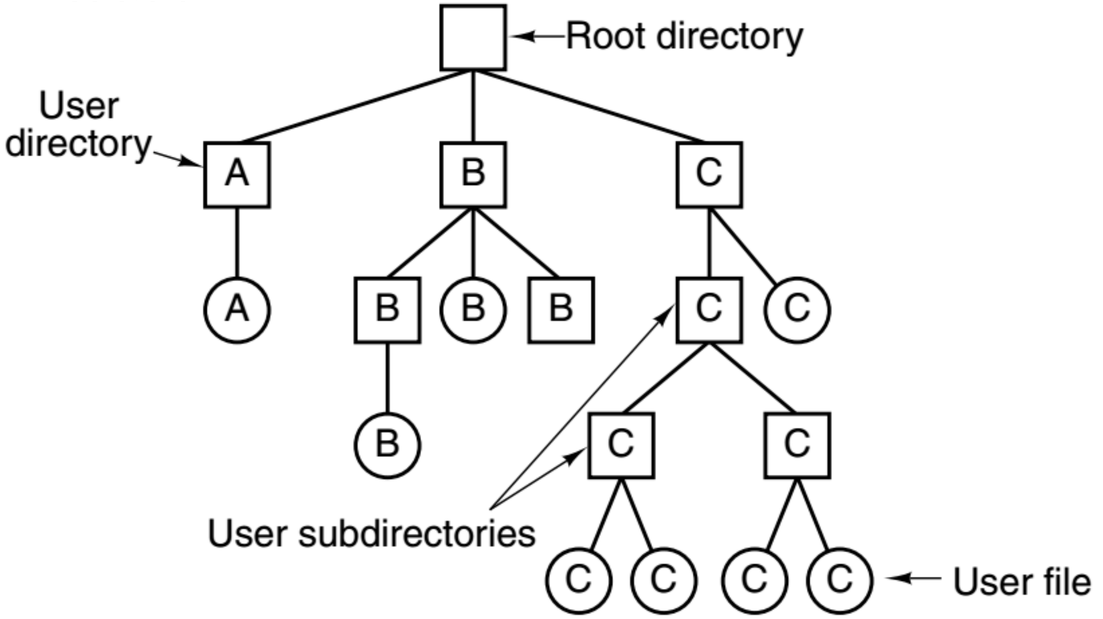


Figure 4-7. A hierarchical directory system.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.277.

4.2.3 パス名

- 階層型ディレクトリ内で特定の位置を識別する概念
  - 階層がはっきりと識別できない場合は、パス名の概念も存在しない。
- 絶対パス名
  - ルートからの経路
- 相対パス名
  - 階層内の(別の)特定の位置からの経路
- カレントディレクトリの概念
- 親ディレクトリの概念と表示

UNIX のディレクトリ木

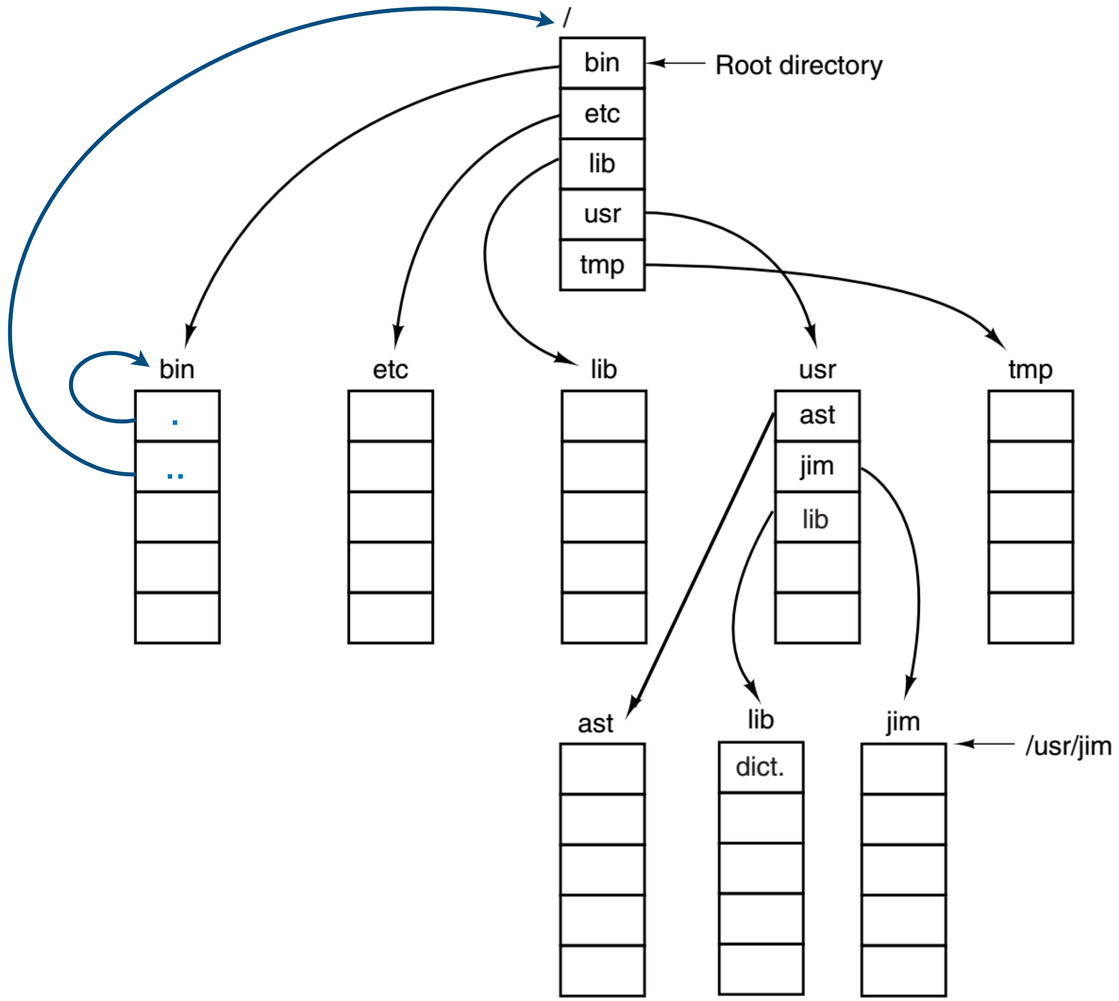


Figure 4-8. A UNIX directory tree.

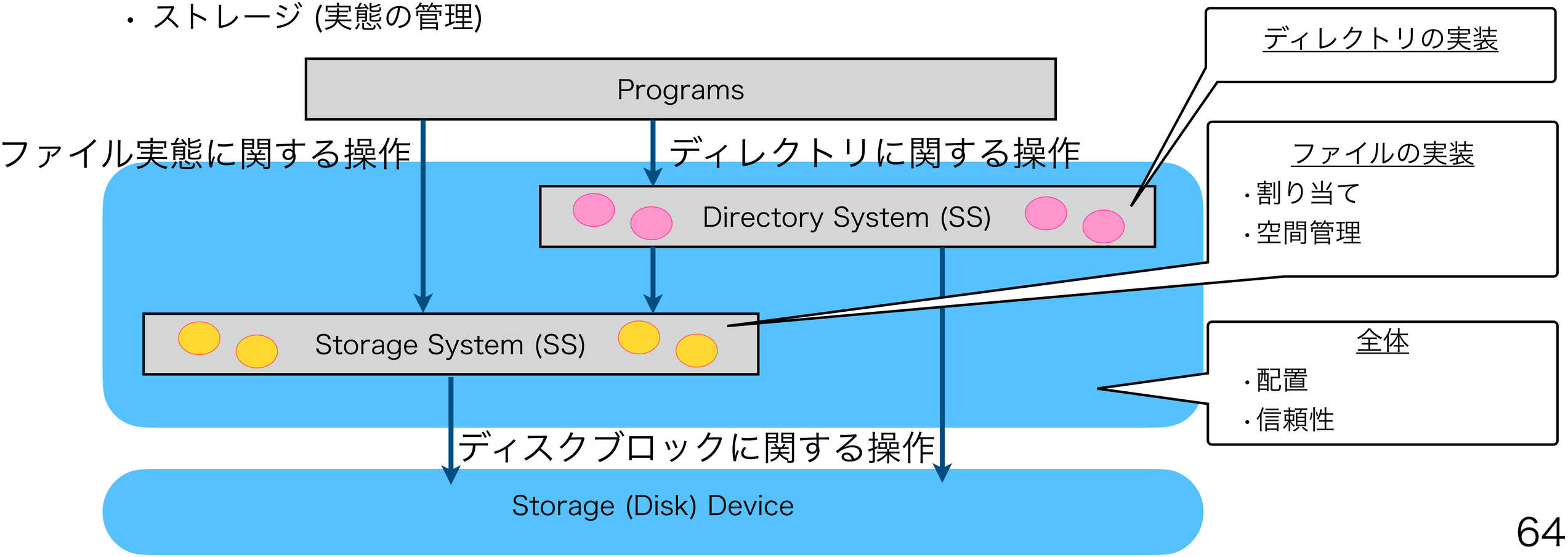
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.279.

4.2.4 ディレクトリ操作

- 典型的なディレクトリ操作 API
  - Create
  - Delete
  - Opendir
  - Closedir
  - Readdir
  - Rename
- UNIX 特有の API
  - Link
  - Unlink

4.3 ファイルシステムの実装

- ファイルシステムの実装は、2 つに分けて考えることができる
  - ディレクトリ (名前、階層構造、属性の管理) または メタデータ
  - ストレージ (実態の管理)



4.3.1 ファイルシステムの配置

- ディスク上の「基本的な領域」の配置・構成
  - 領域管理のための領域
  - ブート領域
  - ファイルシステム

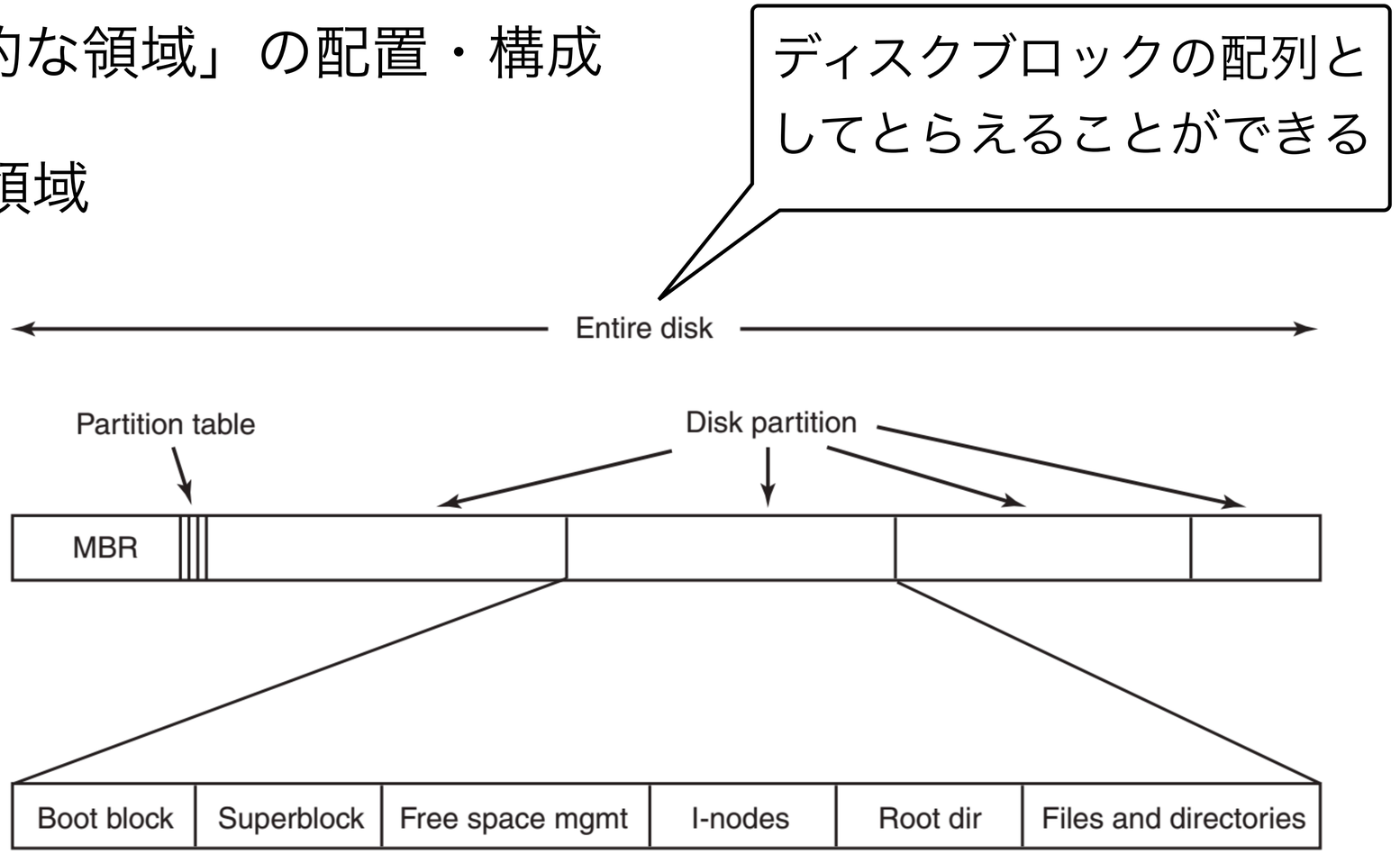


Figure 4-9. A possible file-system layout.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.282.

4.3.2 ファイルの実装

- ファイルの実体を、ディスク上でどのように格納するか。
  - 連続割り当て
  - 連結リスト割り当て
  - テーブル割り当て
  - I-node 方式

連続割り当て

- ファイルの実体を連続したディスクブロックに格納する。
- 高速な順アクセス。
- 空き領域の断片化。
- 管理はビットマップで。

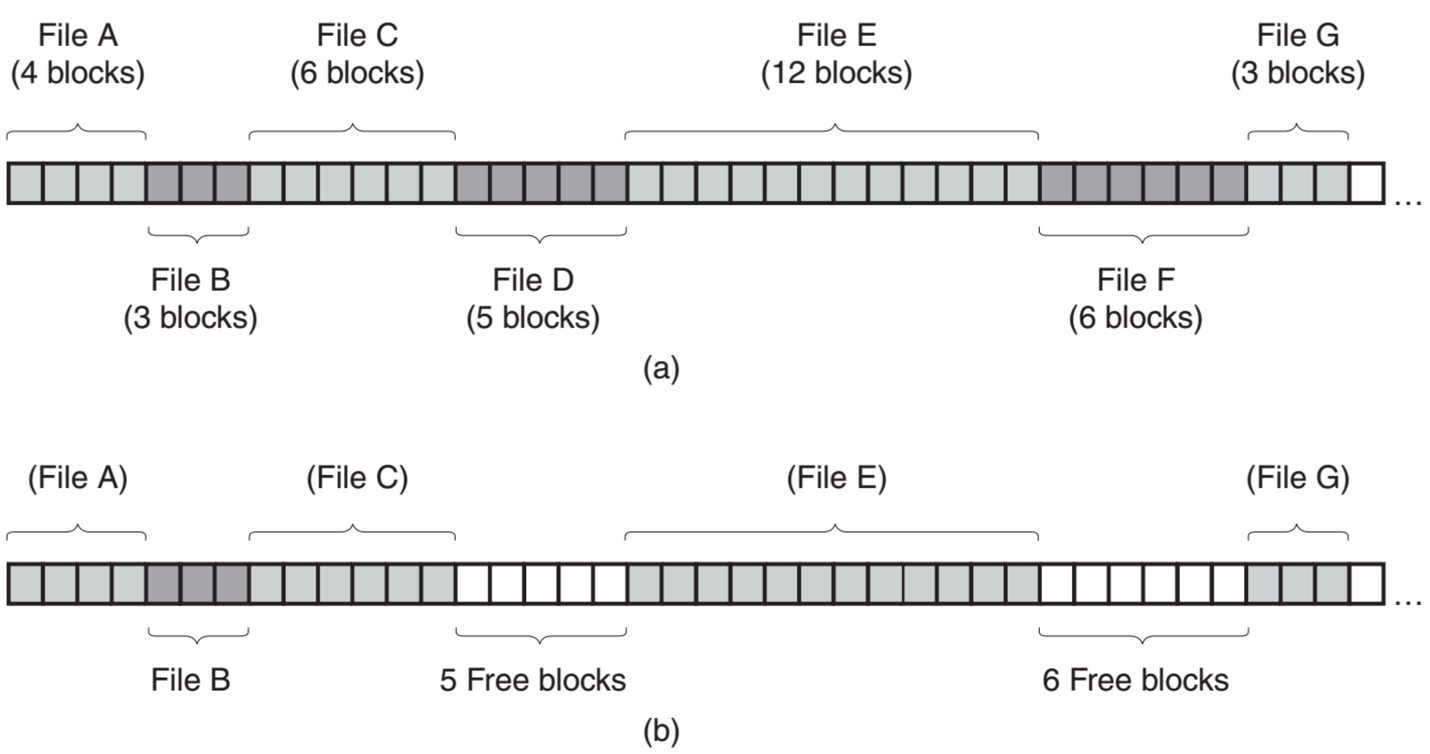


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.283.

連結リスト割り当て

- ファイルを構成するブロックをリストとして管理する。
- 高速な割り当てと解放。
- ランダムアクセスが不可能。
- ブロックサイズが 2 のべき乗でなくなる。

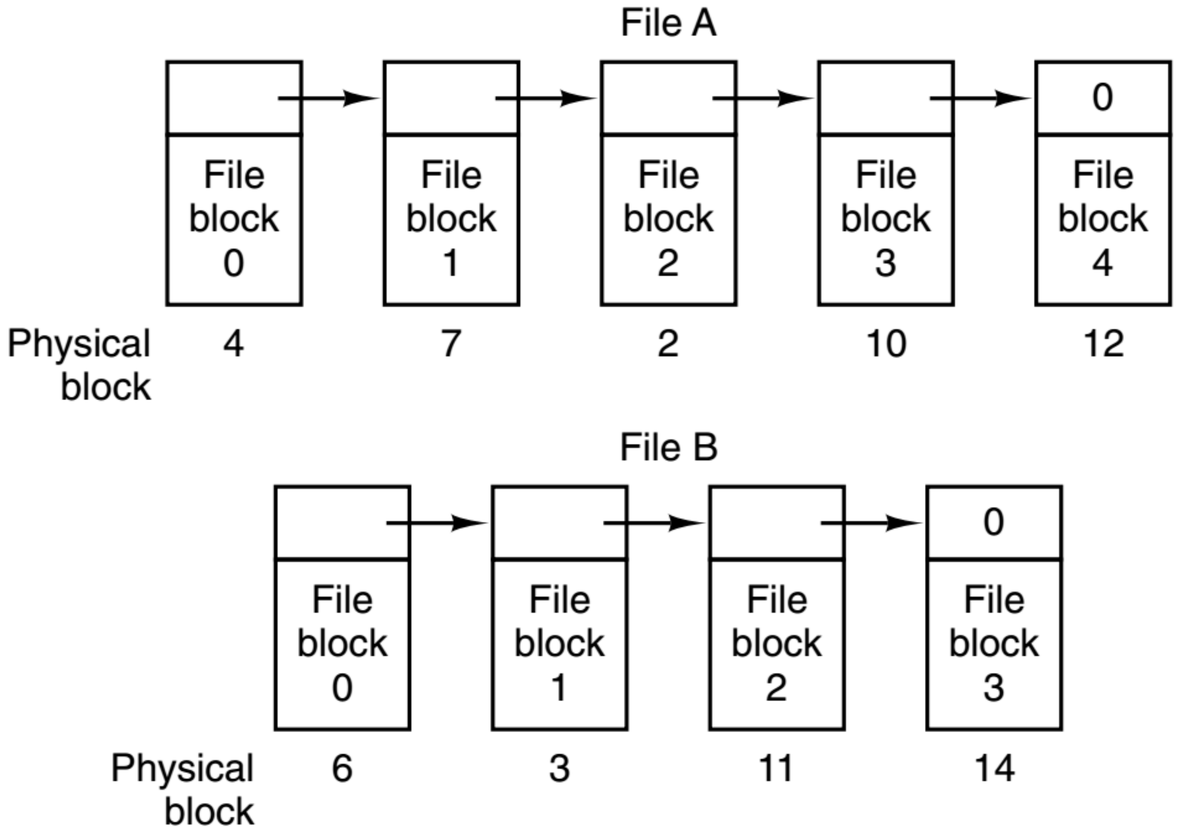


Figure 4-11. Storing a file as a linked list of disk blocks.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.285.

メモリ上のテーブルによる連結リスト方式

- 連結リスト方式の欠点を、メモリ上のポインタテーブルで解消
- テーブルの書き戻し

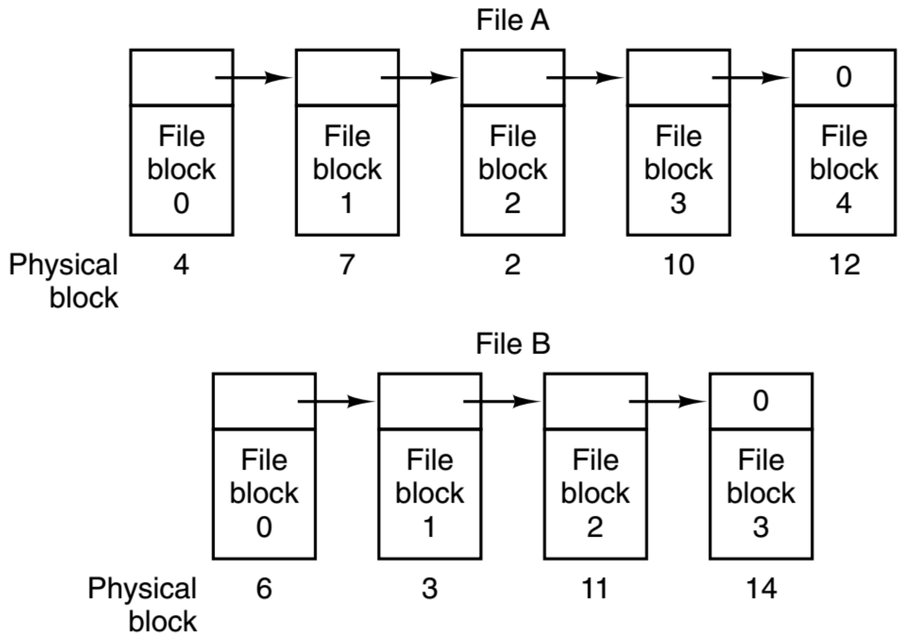


Figure 4-11. Storing a file as a linked list of disk blocks.

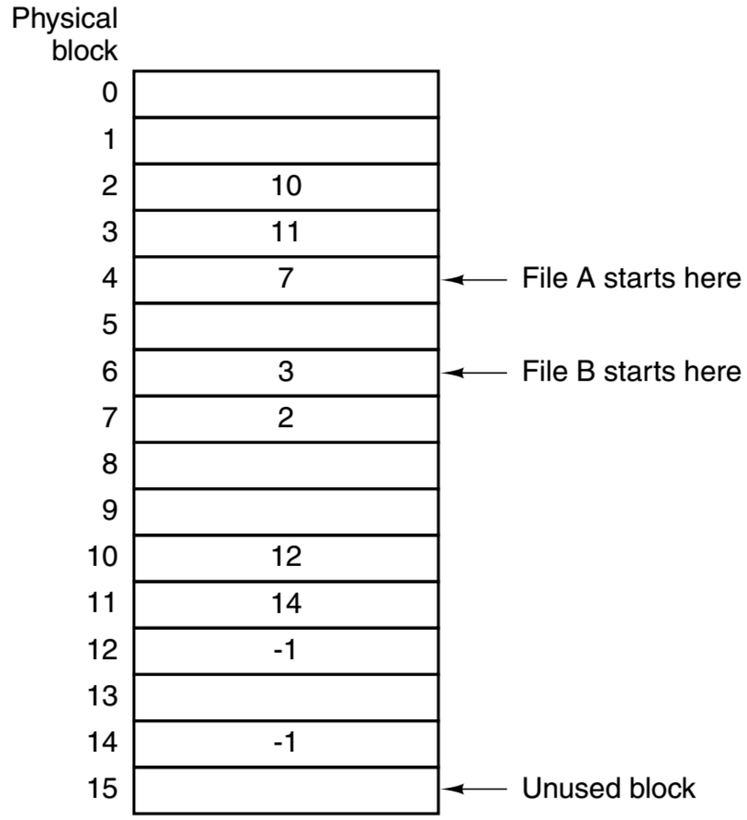


Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.285, 286.

I-nodes 方式

- ファイルごとに、そのファイルを保持するブロックのテーブル(配列)を持つ方式。

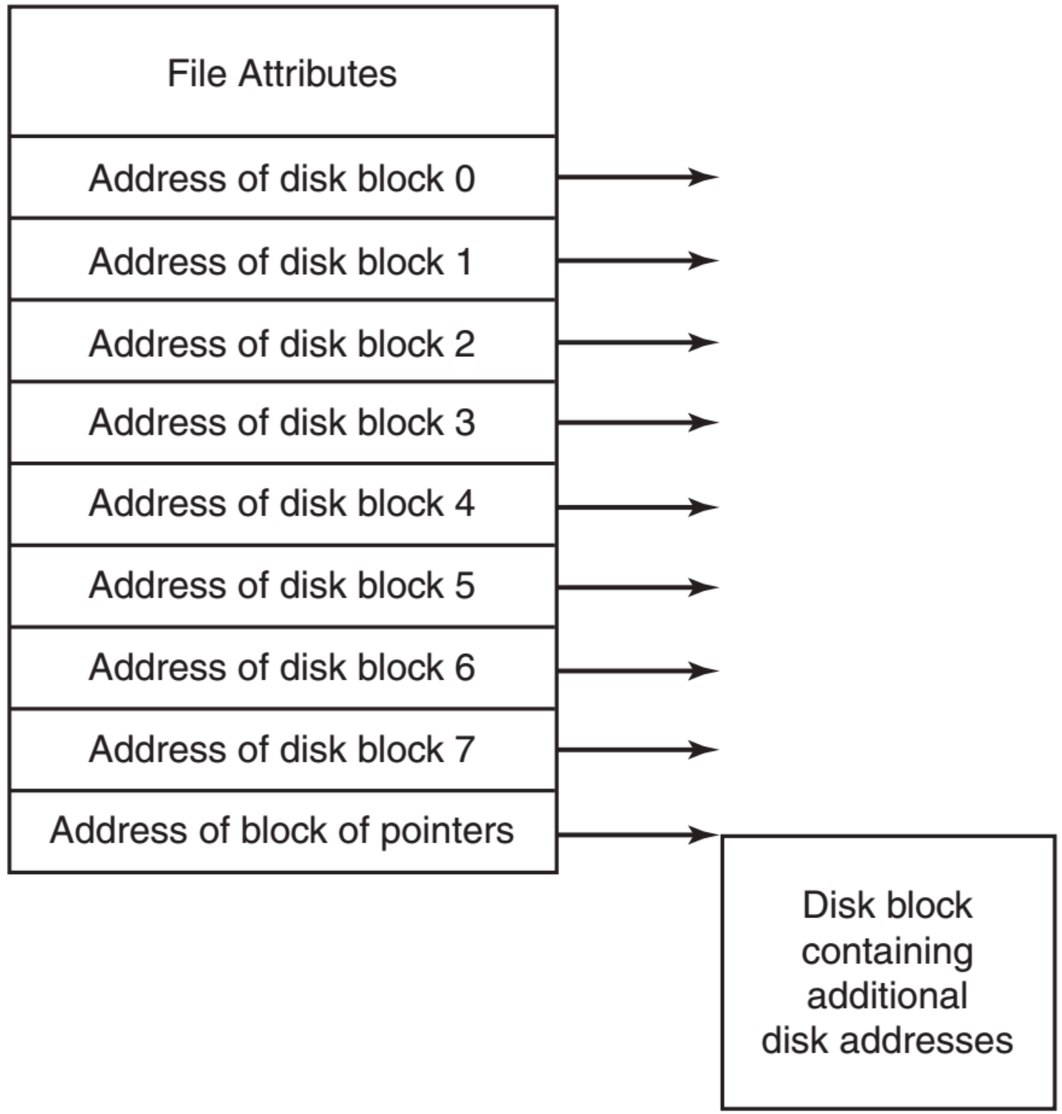


Figure 4-13. An example i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.287.

4.3.3 ディレクトリの実装

- ディレクトリで管理される可能性のある情報
  - 名前
  - 属性
  - SS に関する情報

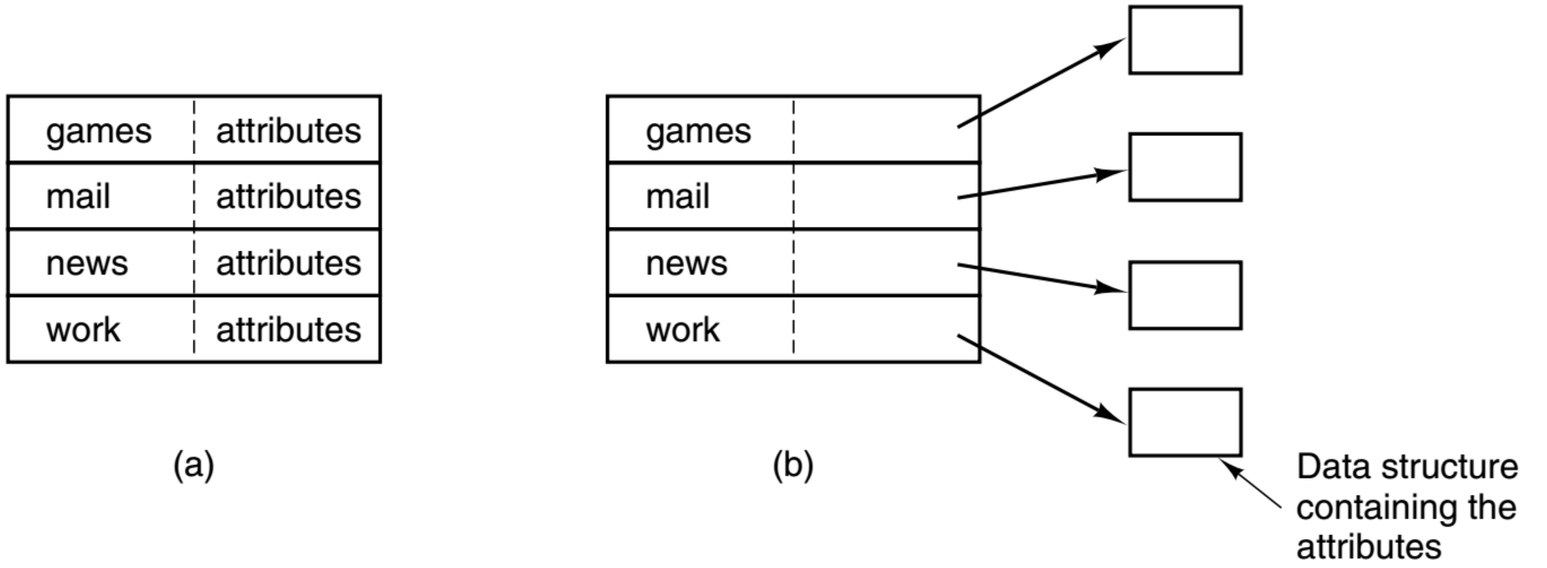


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.288.

4.3.4 共有ファイル

- 同じファイルが木の複数の場所から参照できるようにする。
  - i-node 共有を行う
  - “リンク型” の特殊なファイルを用いる

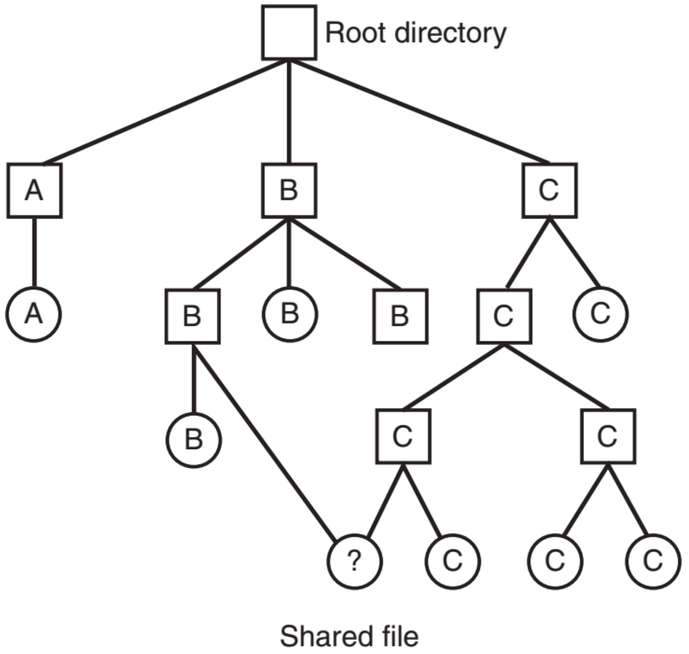


Figure 4-16. File system containing a shared file.

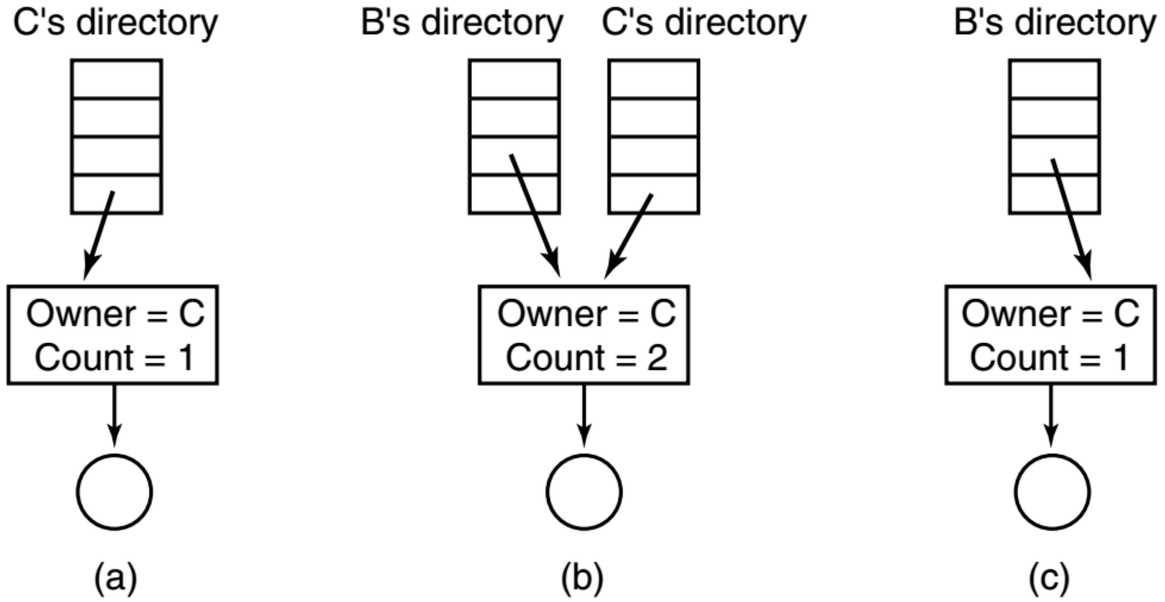


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.291, 292.

4.4.1 ディスク空間管理

- ファイルの格納方式
  - 連続領域での保存
  - 「ブロック」に分割しての保存: ブロックサイズは?
- 空きブロック管理
  - ビットマップ v.s. リスト
- ユーザごとの利用量管理

ブロックサイズの決定

- さまざまなサイズの候補
  - 物理デバイスの特性 (セクタ・トラック・シリンダのサイズ)
  - ページングシステムにおけるページサイズ
- いくつかの要因
  - サイズの大小
    - 大きくすれば内部フラグメンテーションが大きくなる。 e.g. ファイルサイズの中央値が 1KB のシステムで 32KB のブロックサイズを用いると、ディスク全体の 96% 強の領域が無駄になる。
  - 転送速度
    - ブロック数が小さいと、同じ大きさのファイルを転送する場合でもブロックの転送回数が増える = 各ブロックへのシーク動作・開店待ち動作が増加する。

空間利用率 v.s. データ転送速度

- 空間利用率
  ≒ 1 if ブロックサイズ < ファイルサイズの中央値, else ファイルサイズの中央値 / ブロックサイズ

- 転送所要時間
  ≒ ブロック数 * (平均シーク時間 + 平均待ち時間 + 転送時間)

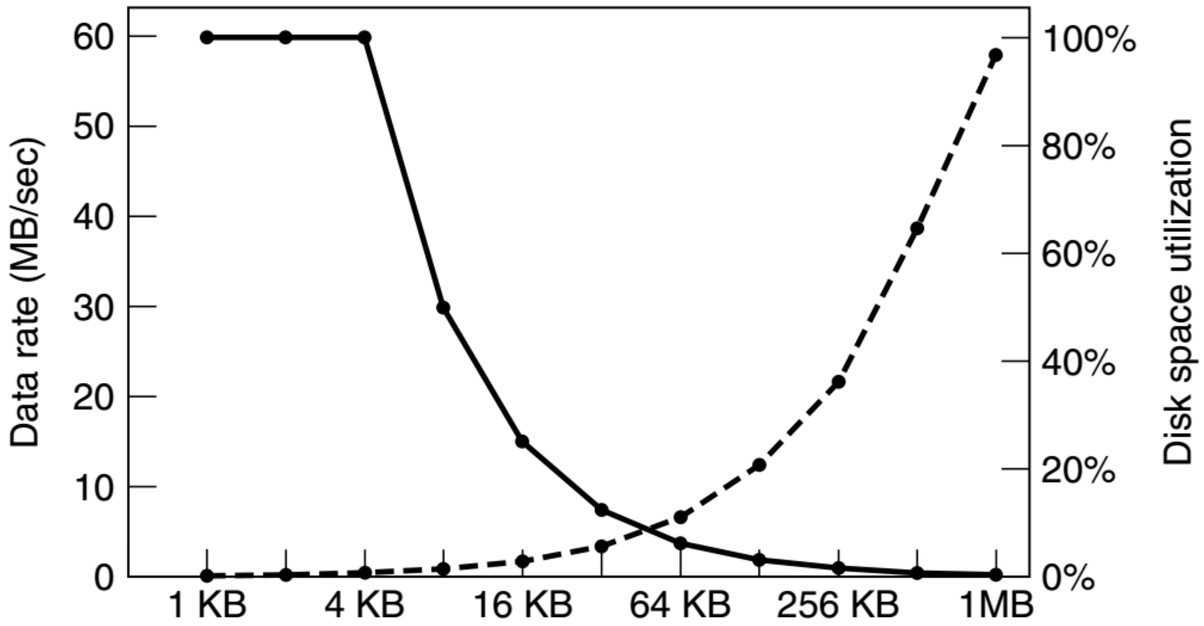


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk-space efficiency. All files are 4 KB.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.302.

空きブロック管理: ビットマップ v.s. リスト

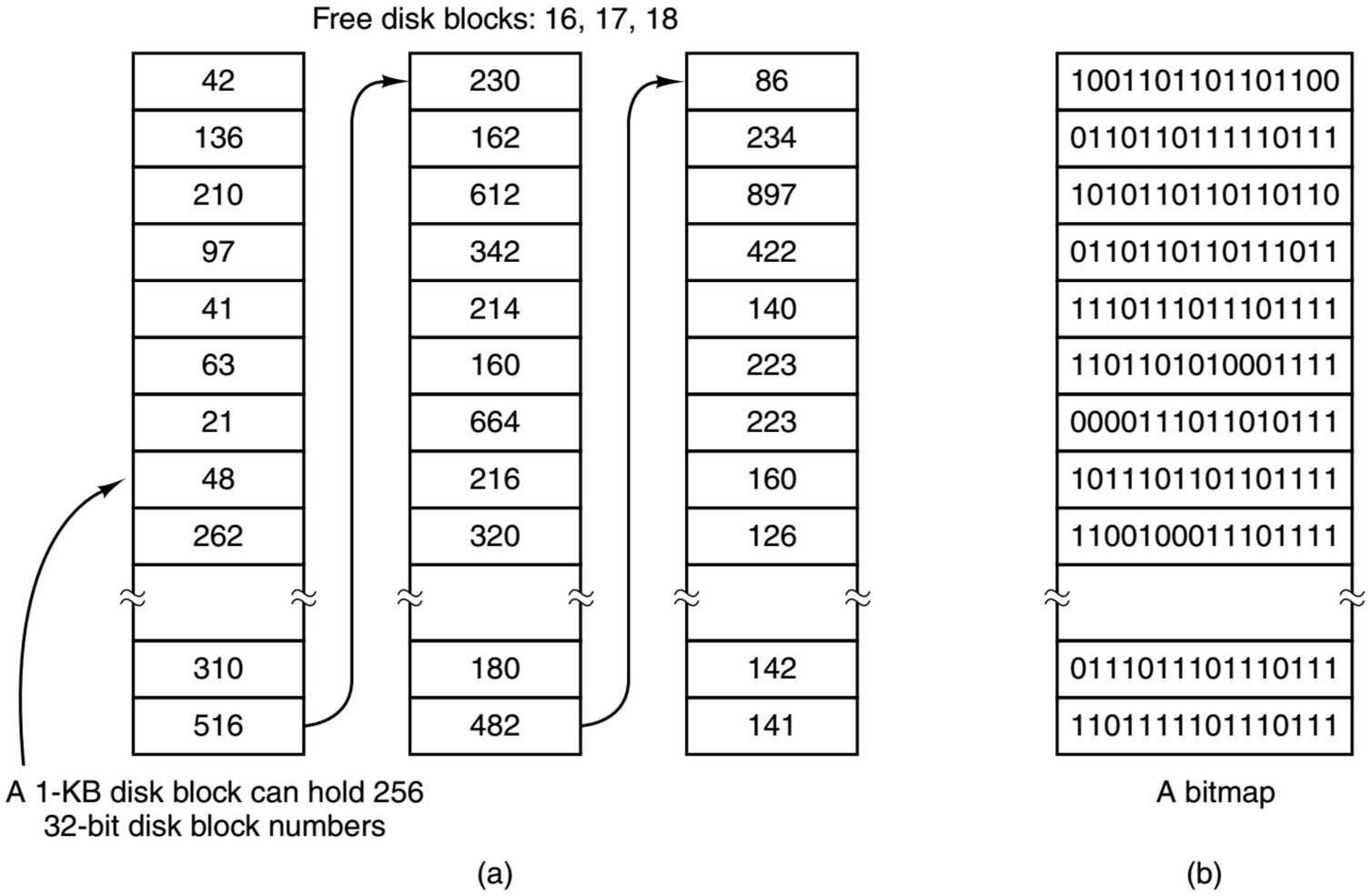


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.303.

クォータ

- ユーザごとにファイルシステムの使用料を管理する仕組み

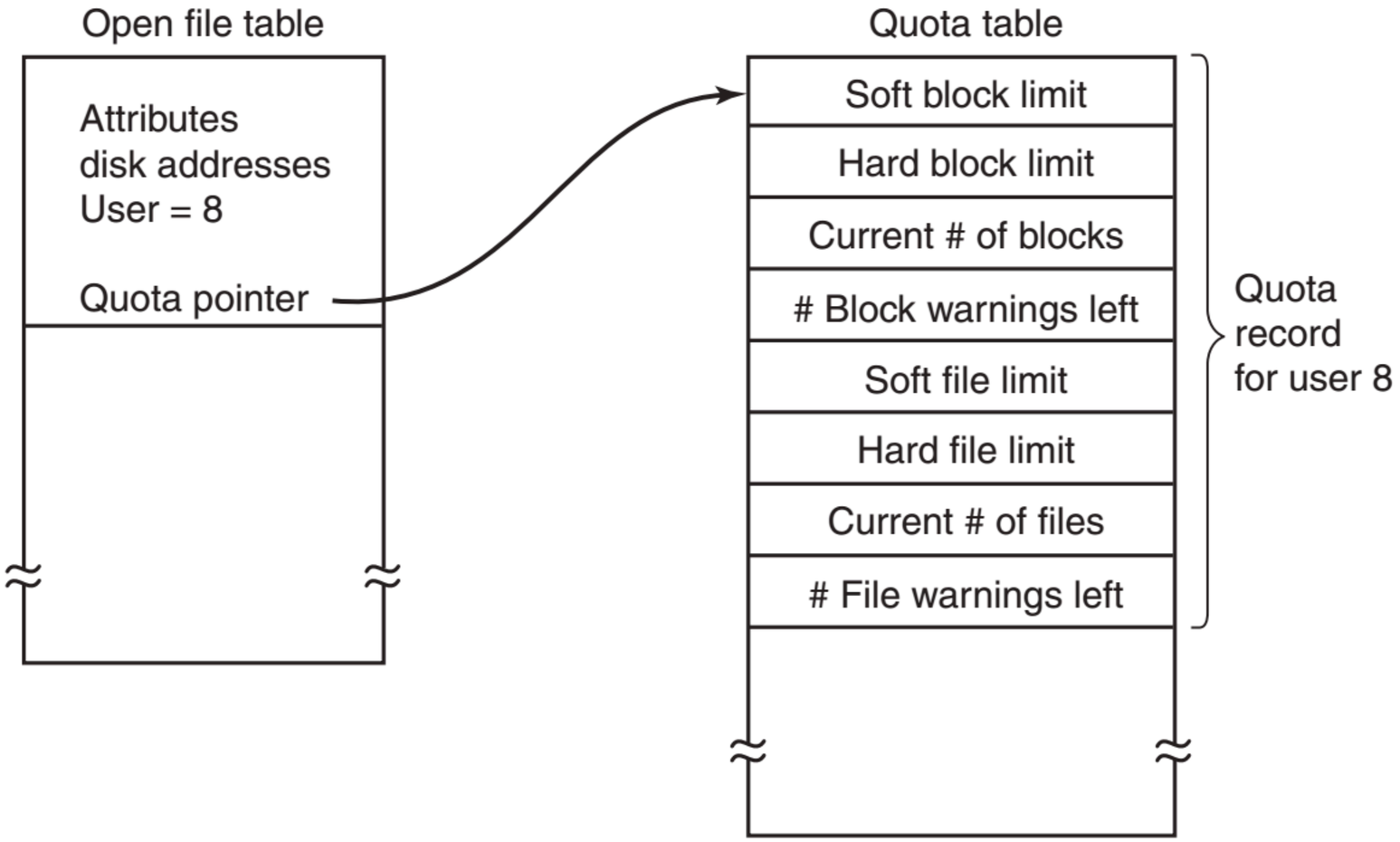


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.306.

ファイルシステムの信頼性

- ファイルシステムの障害・破損はコンピュータの障害・破損よりもずっと深刻である。
- ファイルシステムには、日々生じる小さな障害 (ディスクブロック不良) を隠蔽 (マスク) する機能が備わっている。
- 大規模な障害 (ディスク装置の障害) に備えたバックアップ、あるいは管理情報の一貫性喪失に備えた検査と回復の機構が必要である。

4.4.2 バックアップ: ファイルシステムのダンプ

- 考慮すべき要因
  - ファイルシステム内には、ダンプを必要としないファイルがある。
  - 前回のバックアップ以降に変更されていないファイルは、バックアップの必要がない(差分ダンプの考え方)
  - ダンプデータを圧縮することの是非
  - オンライン状態でのダンプの可否
  - ダンプによって作成された「情報のコピー」によるセキュリティ上の問題
- 2 種類のダンプ方法
  - 物理ダンプ: ディスク装置の全ブロックをそのままコピーする方法。
  - 論理ダンプ: ファイルシステムの構造が復元できるように、必要な情報だけをコピーする方法。差分ダンプはこちら。

差分ダンプのアルゴリズム

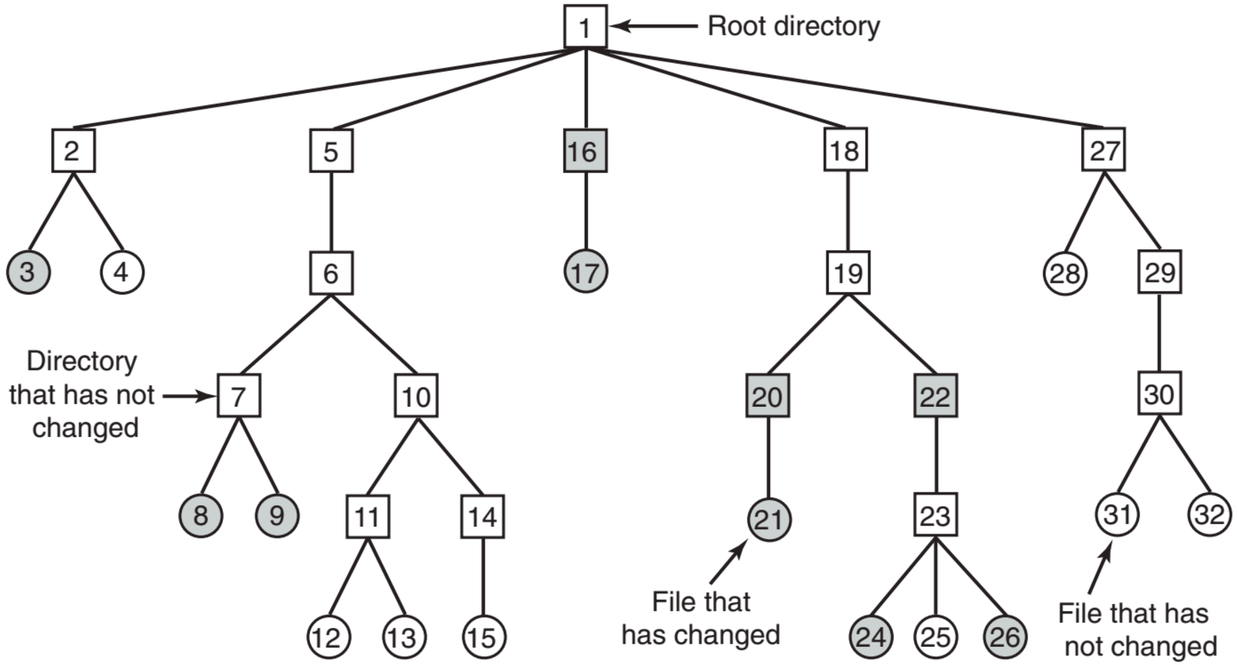


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

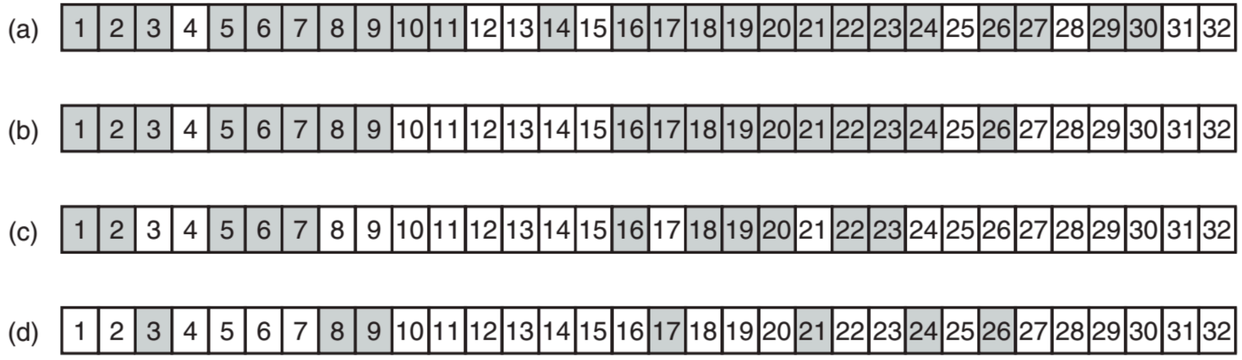


Figure 4-26. Bitmaps used by the logical dumping algorithm.

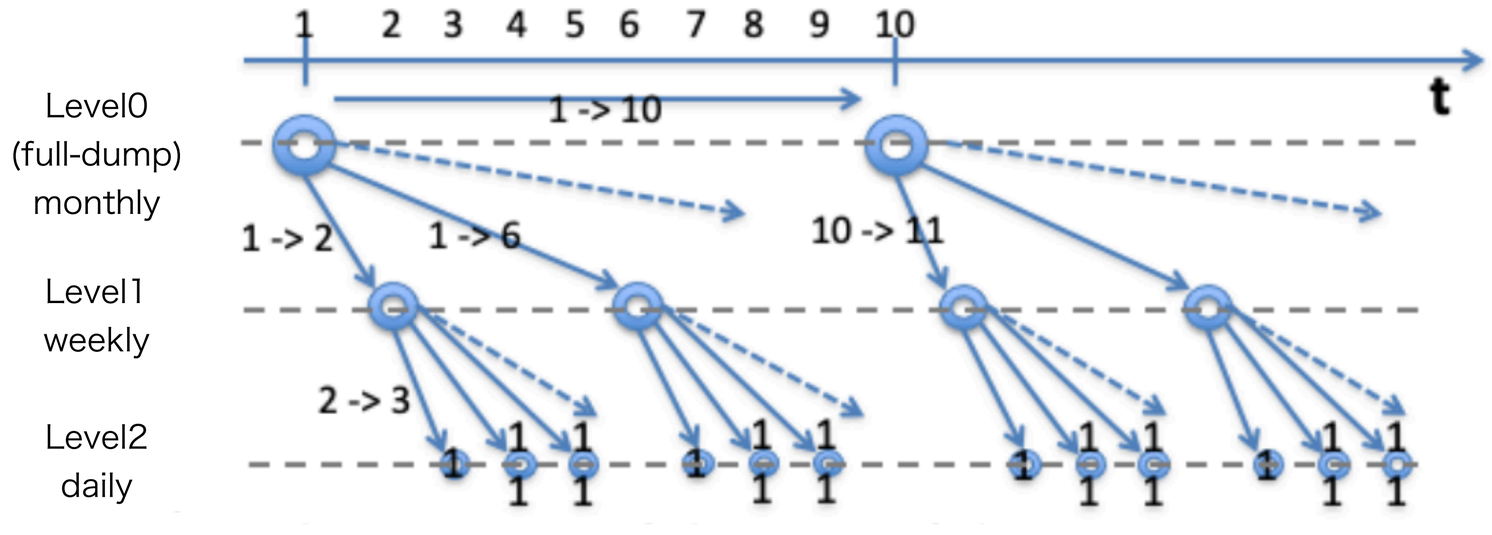
Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.310, 311.

差分ダンプにおける議論

- 削除されたファイルの取扱
  - 差分ダンプでは、削除されたファイルは、削除されたように「復元」できるのか?
- 特殊ファイルの取扱
  - デバイスを表す特殊ファイルはダンプすべきではない。
  - では、他の特殊ファイル(シンボリックリンクなど)はどうか。
- 大きな空孔が存在するファイルはどうするか。
  - たとえば、OS が異常終了した際に吐き出されるメモリイメージ(コアダンプ)は、ふつう物理メモリと同じサイズである。

差分ダンプのスケジューリング

- 差分ダンプによる一連のダンプを利用すると、ファイルシステムをダンプが実行された瞬間の任意の状態に戻すことができる。
- しかしながら、それには差分ダンプの復元を順番に行わなければならない:
  e.g. 毎日バックアップをしている場合、最初にバックアップを開始してから 700 日目の状態に戻すには、699 個の差分ファイルを順番に重ね書きしないといけない。
- 各差分ダンプで「差分の起点」を上手に利用することで、この問題を避けることができる。



任意の一日に対して、直前のレベル 0、直前のレベル 1、その日のレベル 2 を、この順序で復元することで、その日の状態に戻すことができる。

4.4.3 ファイルシステムの一貫性

- ファイルシステムを構成する情報の、ディスクへの書き込み(書き戻し)が、例えば OS クラッシュ等の理由で完了しないと、ファイルシステムの一貫性が失われた状態になる。
- 一貫性の種類
  - ブロック一貫性: あるブロックは、空きブロックであるか、ファイルを構成するブロックであり、それぞれの管理構造の中に唯一回だけ出現する。
  - ディレクトリー貫性: すべてのディレクトリエントリからの、特定の i-node への参照数の和は、その i-node の被参照数と等しい。
- 一貫性が失われている状態の例
  - どこからも参照されない i-node がある。
  - i-node のリンクカウント(被参照数)が実際よりも大きい。
  - フリービットマップに入っていない空きブロックがある。
  - フリービットマップに入っている空きブロックが、ファイルの一部としても用いられている。
  - スーパーブロック中の値の誤り。

一貫性の喪失例

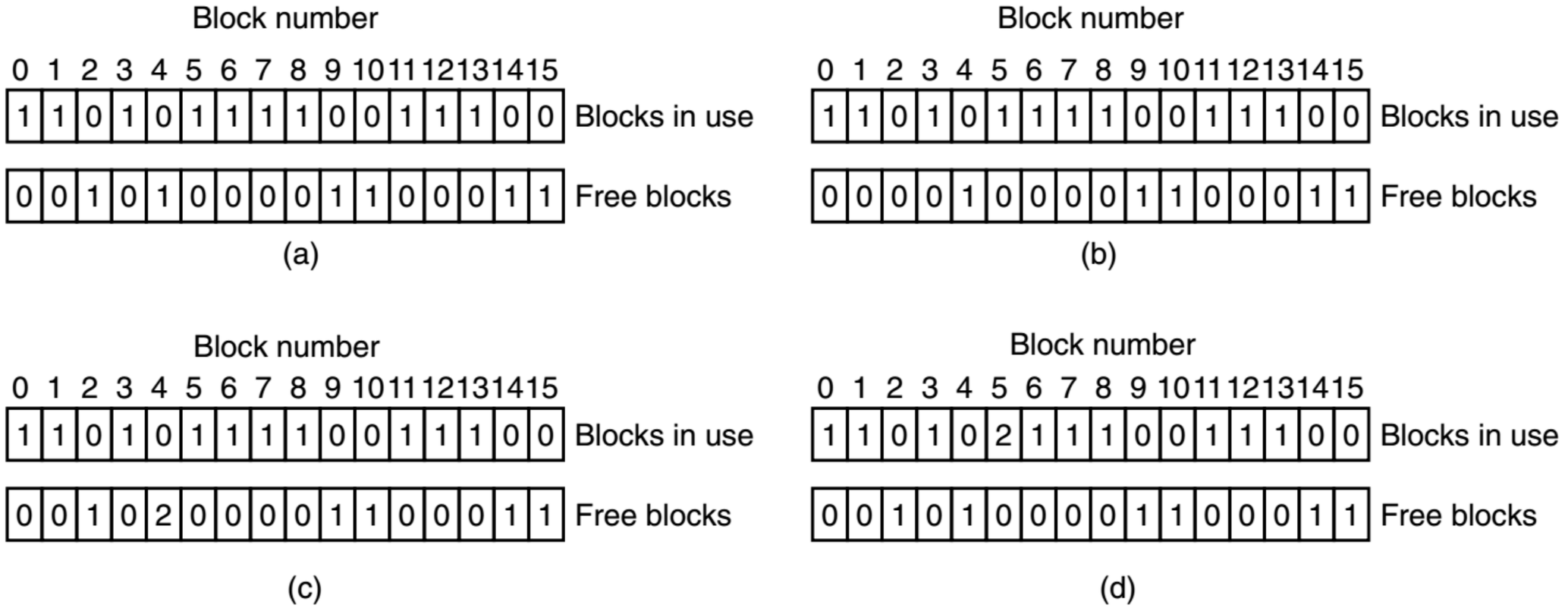


Figure 4-27. File-system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.313.

一貫性検査と回復の例

- ブロック一貫性
  - 全ブロックに対して、ブロックの出現回数をカウントする配列と、空き領域管理構造中に出現した回数をカウントする配列を用意。
  - 全ての i-node をスキャンし、その i-node から辿れる全てのブロックに対し、「使用中ブロック」配列の対応する要素を 1 つ増やす。
  - 空き領域管理構造をスキャンし、空きとして管理されているブロックに対し、「空きブロック」配列の対応する要素を 1 つ増やす。
  - 上記の操作が終了したとき、すべてのブロックは、「使用中」または「空き」のどちらかに 1 が入っていなければならない。
- ブロック一貫性喪失の回復例 (重複ブロックの場合)
  - 重複ブロックのコピーを生成し、それぞれを元のブロックを参照していた i-node に割り当てる。
  - これにより、重複ブロック状態は解消される。
  - ファイルシステム的には一貫性を取り戻すが、ファイルの内容(意味)が破壊されていないかどうかは別途検査する必要がある。

4.4.4 ファイルシステムの性能

- ブロックキャッシュ (バッファキャッシュ)
- ブロックの先読み
  - シーケンシャルアクセス
- ディスクアーム移動(シーク距離)の削減
  - 同じファイルに属するブロックを近づけて配置する。

色々なファイルシステム

- 4.3.5 ログ構造化ファイルシステム
  - 小さなランダム書き込みが大部分を占める作業負荷に対しても、ディスクの全帯域幅を確保できるようにすべく、UNIXのファイルシステムを根本的に再実装した。
  - 基本的な考え方は、ディスク全体を大きなログとして構成することである。
  - しかし、既存のファイルシステムとの互換性が低いこともあり、あまり普及していない。
- 4.3.6 ジャーナリングファイルシステム
  - ファイルシステムに内在するアイデアの1つである「障害に直面したときの堅牢性」は、従来のファイルシステムにも簡単に適用することができる。
  - ファイルシステムが何かを実行する前に、何をしようとしているのかのログを残しておくことで、予定した作業を完了する前にシステムがクラッシュした場合、再起動時にログを見て、クラッシュ時に何が行われていたかを確認し、作業を再開することができる。

4.5.1 MS-DOS ファイルシステム

- いわゆる “FAT” ファイルシステム
- 階層化ディレクトリの導入
- ブロック管理をディレクトリエントリから分離
- ブロック番号でインデックスされる「次ブロック」テーブル

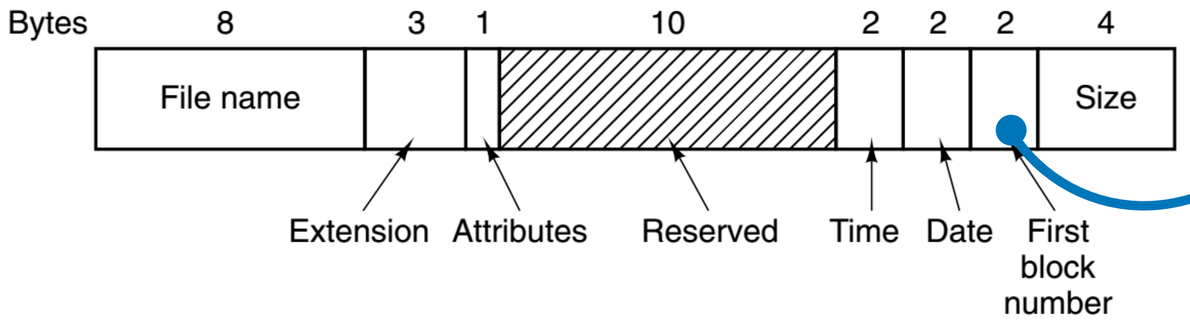


Figure 4-30. The MS-DOS directory entry.

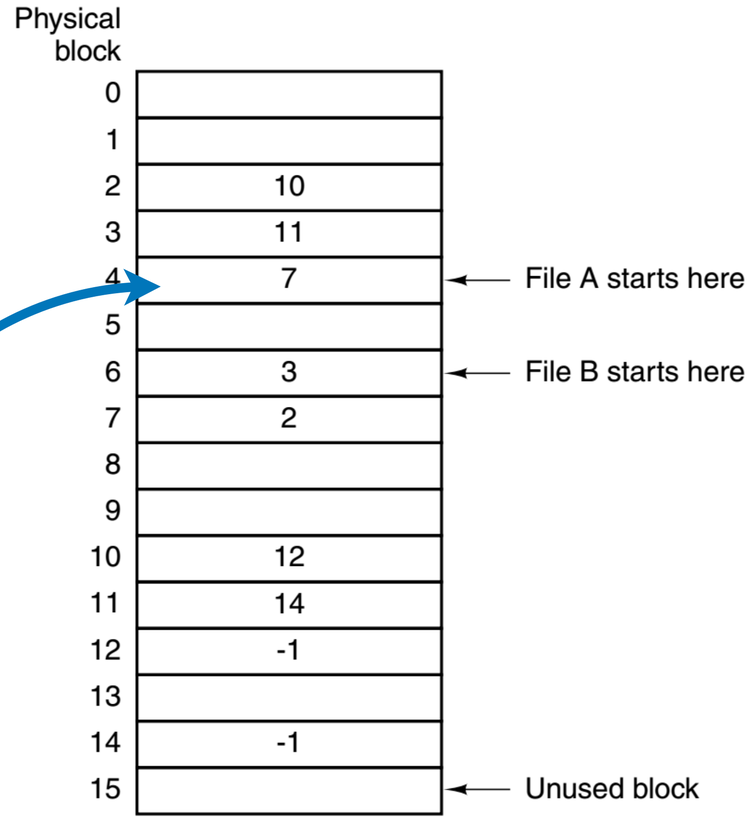


Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.321, 286.

4.5.2 UNIX V7 ファイルシステム

- ディレクトリエントリ(名前だけを管理)と i-node (ファイル実体とその属性を管理)の分離

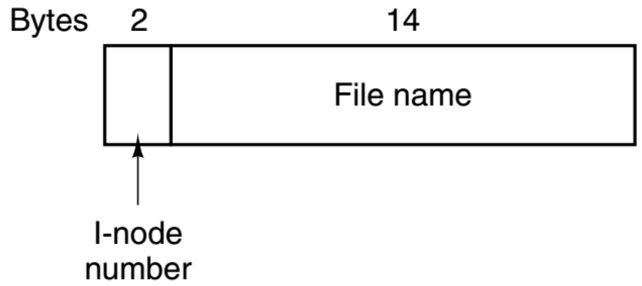


Figure 4-32. A UNIX V7 directory entry.

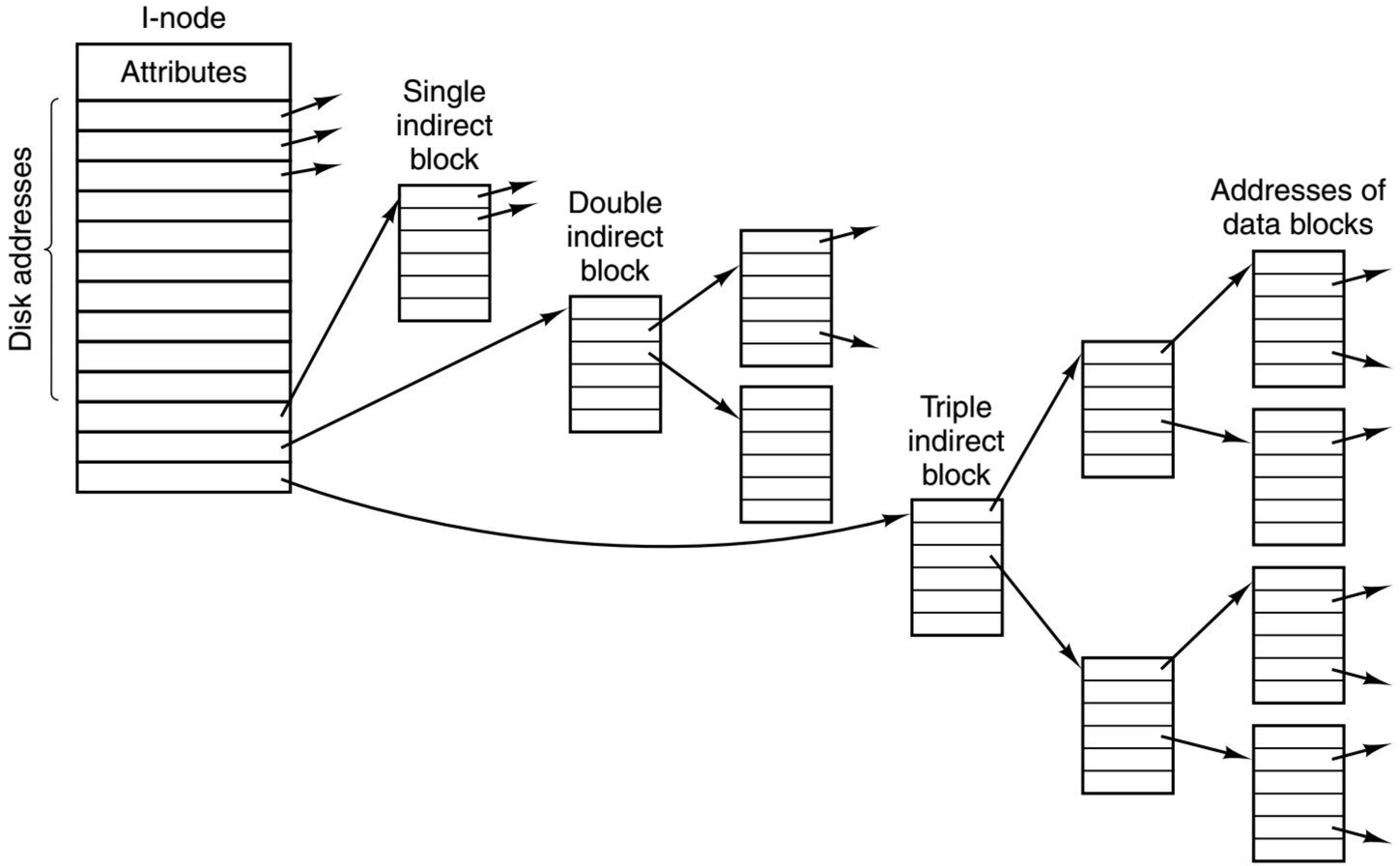


Figure 4-33. A UNIX i-node.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.324, 325.

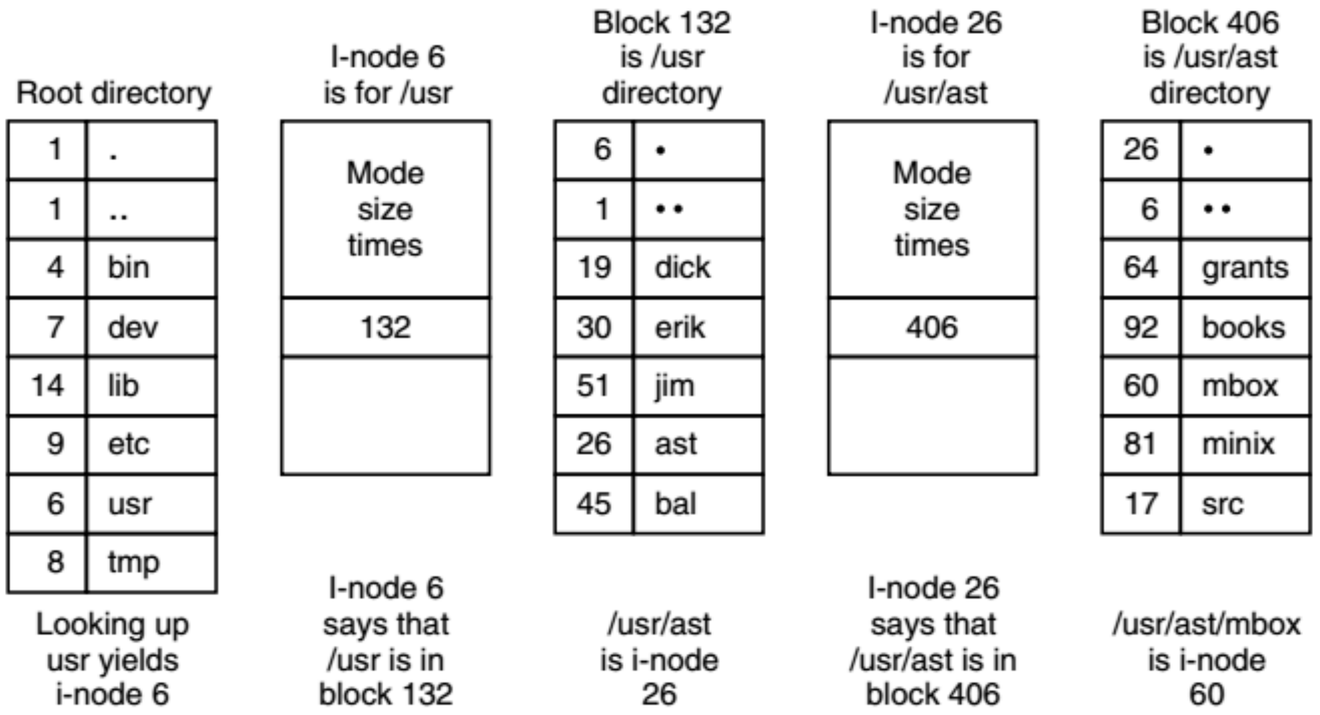


Figure 4-34. The steps in looking up */usr/ast/mbox*.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.326.