Operating Systems: Internals and Design Principles

Chapter 12
File Management

Eighth Edition

By William Stallings

12.1 Overview

- Data collections created by users
- The file system is one of the most important parts of the OS to a user
- Desirable properties of files:

Long-term existence

- files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

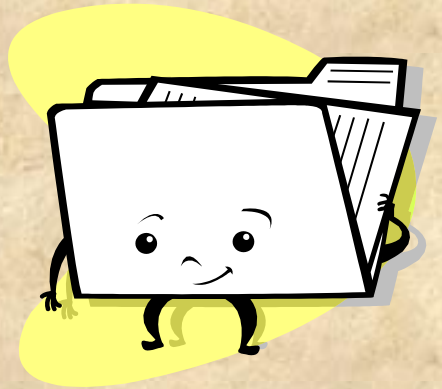
- files have names and can have associated access permissions that permit controlled sharing

Structure

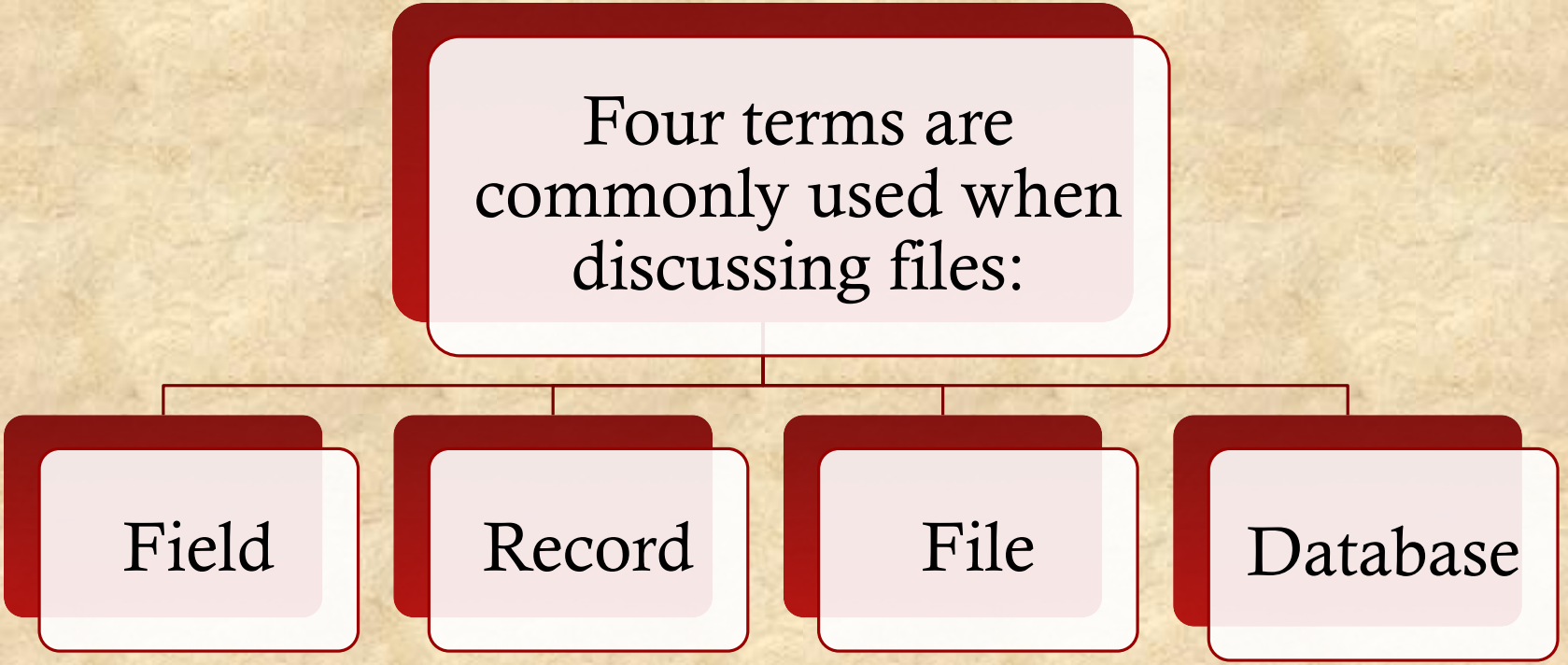
- files can be organized into hierarchical or more complex structure to reflect the relationships among files

File Systems

- Provide a means to store data organized as files as well as a collection of functions that can be performed on files
- Maintain a set of attributes (e.g., owner, creation time, last modified, and access privileges) associated with the file
- Typical operations include:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write



File Structure



Structure Terms

Field

- basic element of data
- contains a single value
- fixed or variable length

File

- collection of similar records
- treated as a single entity
- may be referenced by name
- access control restrictions usually apply at the file level

Record

- collection of related fields that can be treated as a unit by some application program
- fixed or variable length

Database

- a collection of related data
- the relationships among elements of data are explicit
- designed for use by a number of different applications
- consists of one or more types of files

File Management System

- A set of system software that provides services to users and applications in the use of files
- Objectives
  - Meet the data management needs of the user
  - Guarantee that the data in the file are valid
  - Optimize performance
  - Provide I/O support for a variety of storage device types
  - Minimize the potential for lost or destroyed data
  - Provide a standardized set of I/O interface routines to user processes
  - Provide I/O support for multiple users, in the case of multiple-user systems

Minimal User Requirements

- Each user:

1. should be able to create, delete, read, write and modify files
2. may have controlled access to other users' files
3. may control what type of accesses are allowed to the files
4. should be able to move data between files
5. should be able to back up and recover files in case of damage
6. should be able to access his or her files by name rather than by numeric identifier

File System Architecture

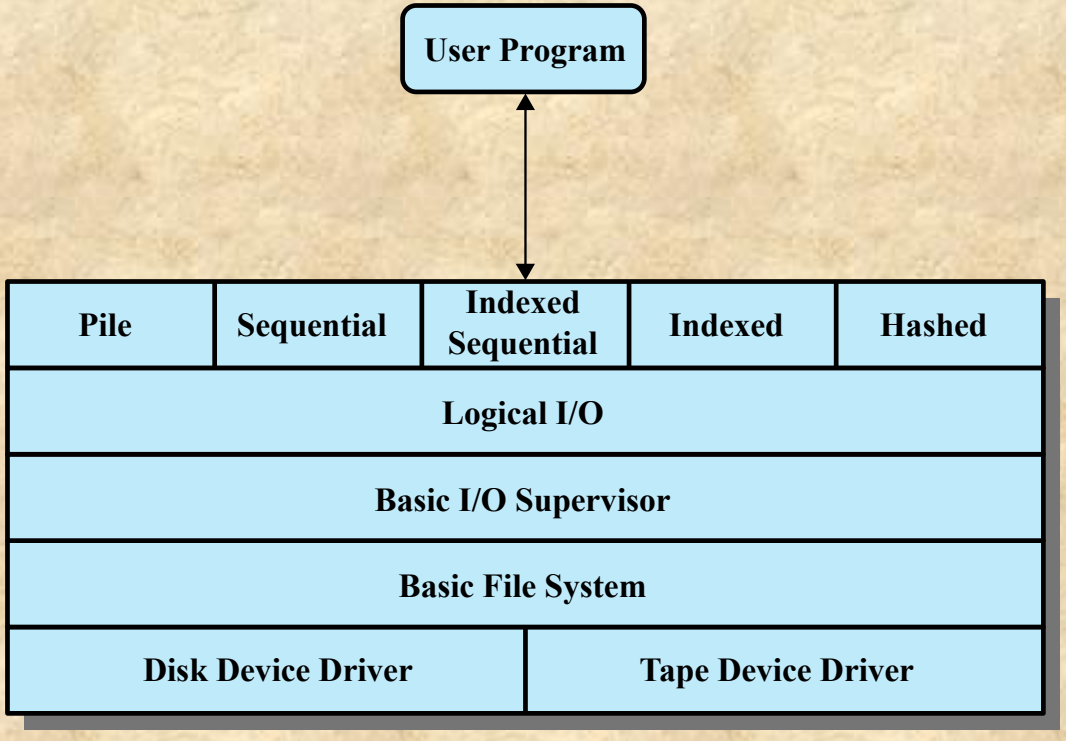
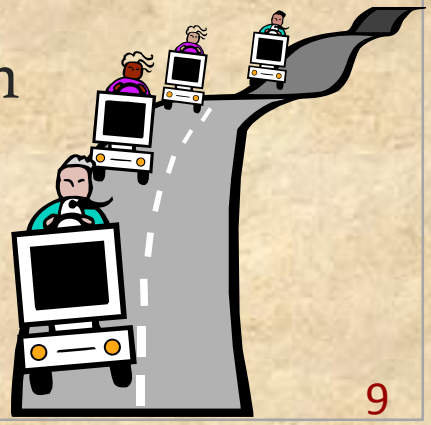


Figure 12.1 File System Software Architecture

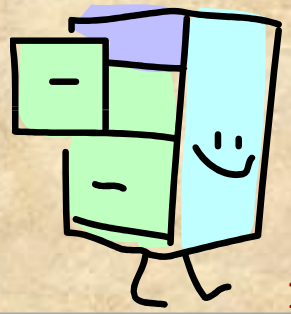
Device Drivers

- Located at the lowest level
- Communicates directly with peripheral devices
- Responsible for starting I/O operations on a device
- Processes the completion of an I/O request
- Considered to be part of the operating system



Basic File System

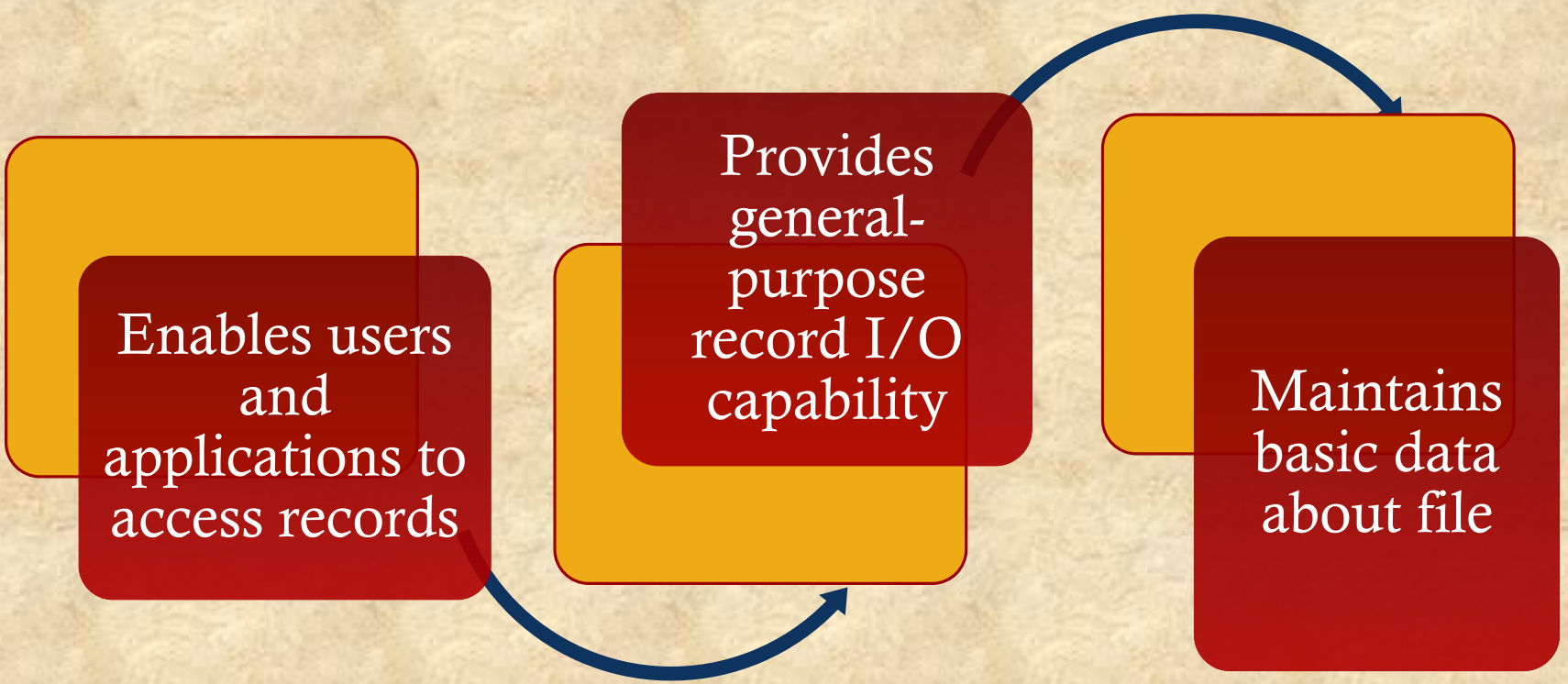
- Also referred to as the physical I/O level
- Primary interface with the environment outside the computer system
- Deals with blocks of data that are exchanged with disk or tape systems
- Concerned with the placement of blocks on the secondary storage device
- Does not understand the content of the data or the structure of the files involved
- Concerned with on the buffering of blocks in main memory
- Considered part of the operating system



Basic I/O Supervisor

- Responsible for all file I/O initiation and termination
- Control structures that deal with device I/O, scheduling, and file status are maintained
- Selects the device on which I/O is to be performed
- Concerned with scheduling disk and tape accesses to optimize performance
- I/O buffers are assigned and secondary memory is allocated at this level
- Part of the operating system

Logical I/O


Enables users and applications to access records
Provides general-purpose record I/O capability
Maintains basic data about file

Access Method

- Level of the file system closest to the user
- Provides a standard interface between applications and the file systems and devices that hold the data
- Different access methods reflect different file structures and different ways of accessing and processing the data

File Management Functions

- Users and application programs interact with the file system by means of commands
- Directory serves to describe the location of all files, plus their attributes
- Only authorized users are allowed to access particular files in particular ways
- To translate user commands into specific file manipulation commands, the access method appropriate to this file structure must be employed
- The records or fields of a file must be organized as a sequence of blocks for output and unblocked after input
- Both disk scheduling and file allocation are concerned with optimizing performance

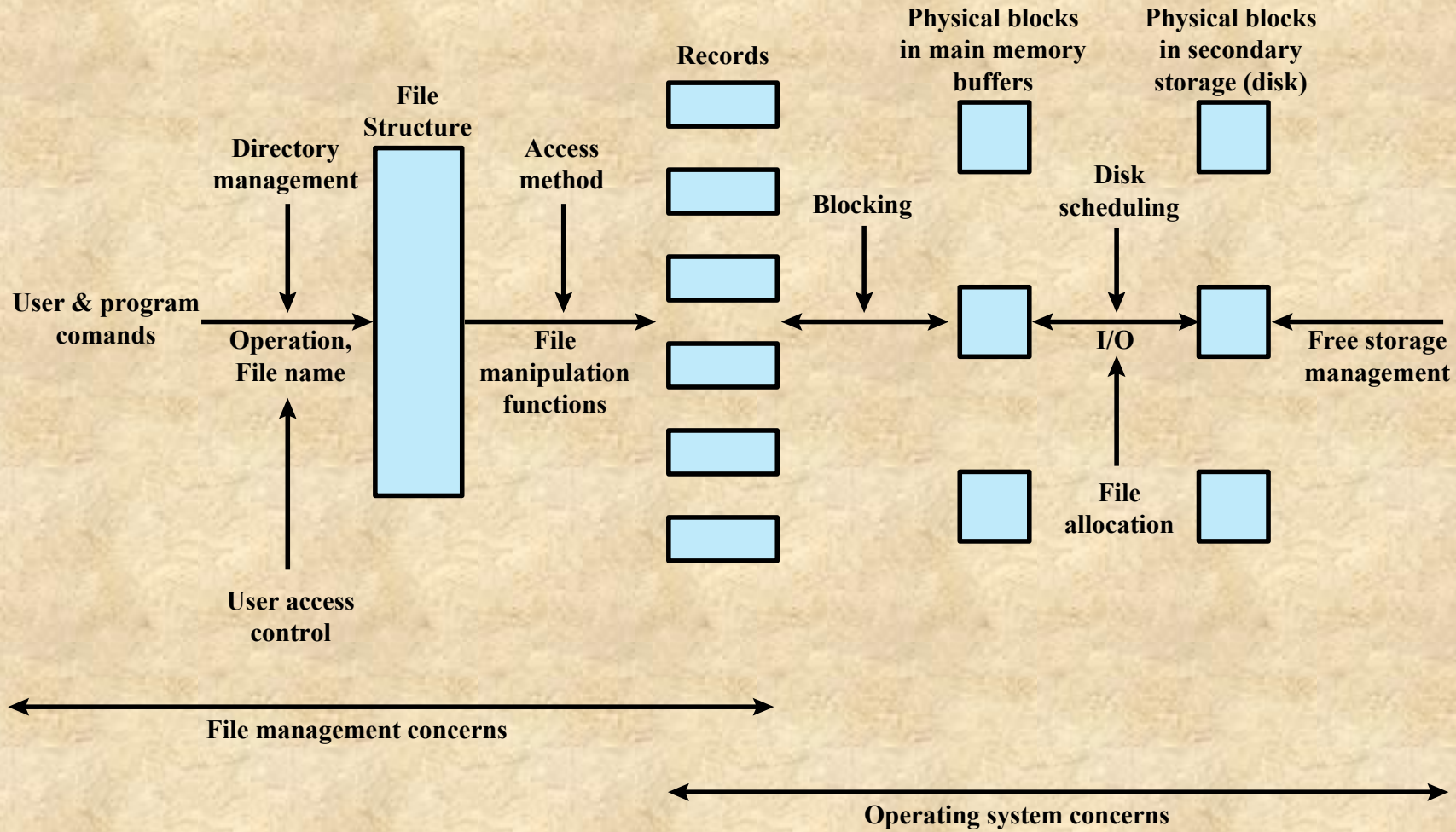


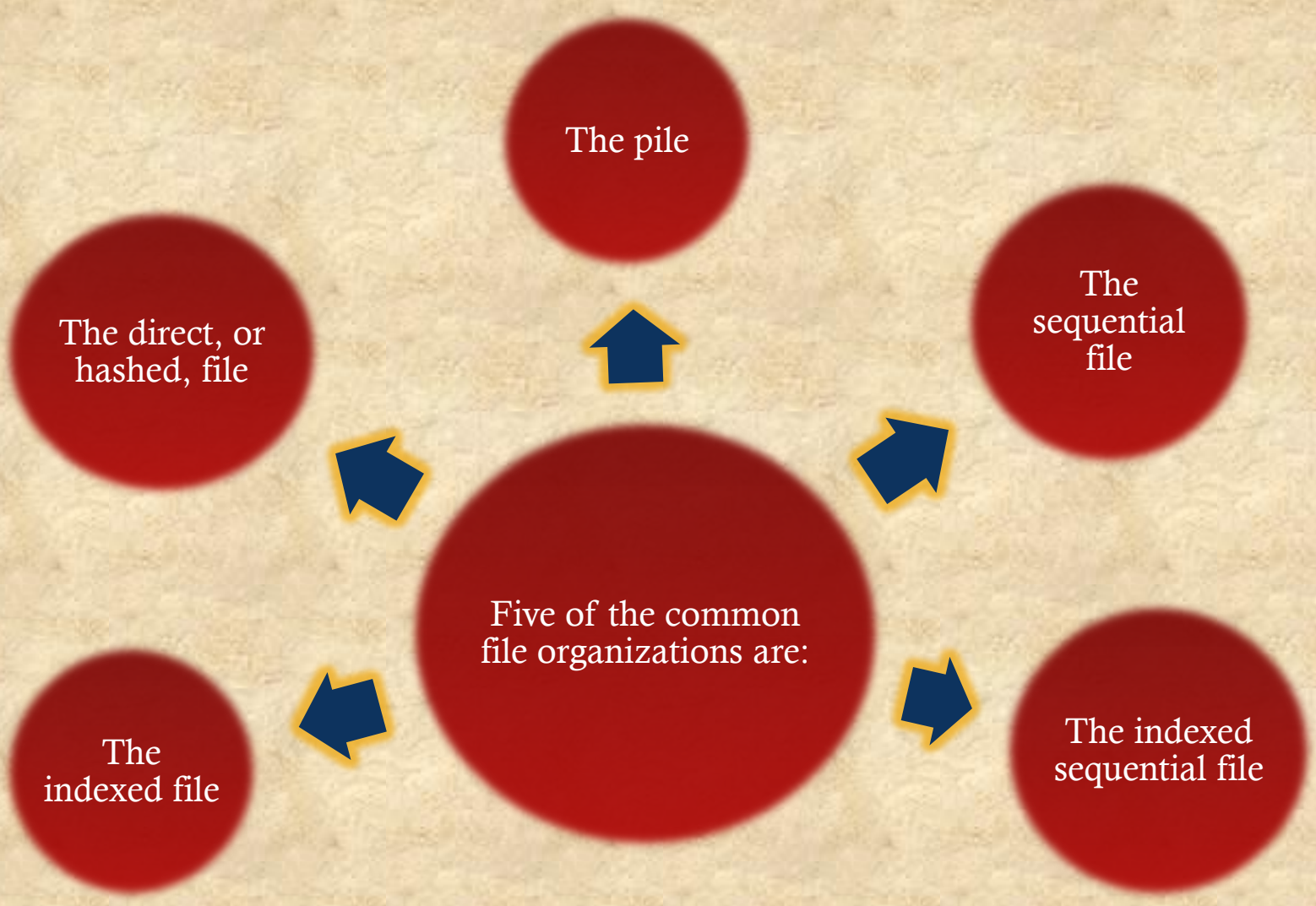
Figure 12.2 Elements of File Management

12.2 File Organization and Access

- ***File organization*** is the logical structuring of the records as determined by the way in which they are accessed
- In choosing a file organization, several criteria are important:
  - short access time
  - ease of update
  - economy of storage
  - simple maintenance
  - reliability

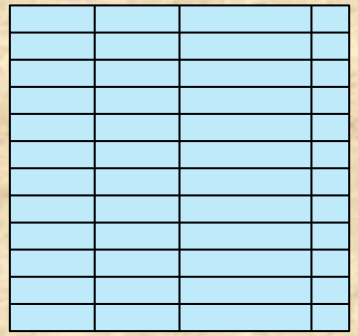
- The relative priority of criteria depends on the application that will use the file

File Organization Types



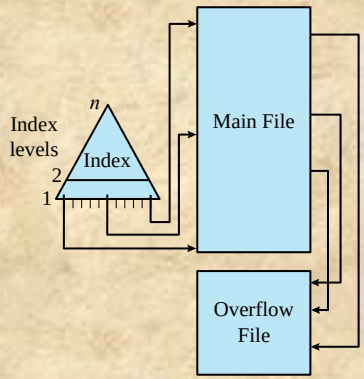
Variable-length records
Variable set of fields
Chronological order

(a) Pile File

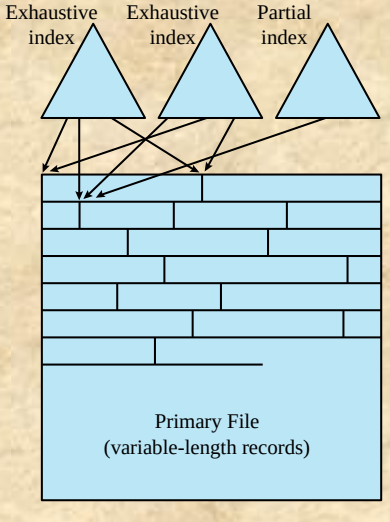


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File



(c) Indexed Sequential File



(d) Indexed File

Figure 12.3 Common File Organizations

The Pile

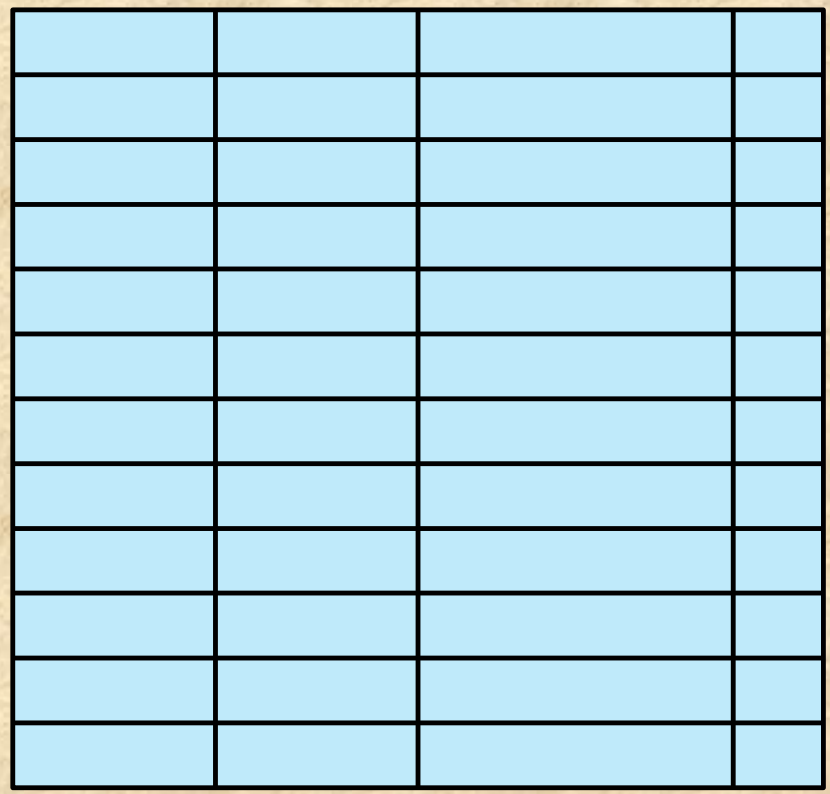
- The least-complicated form of file organization
- Data are collected in the order they arrive
- Each record consists of one burst of data
- The purpose is simply to accumulate the mass of data and save it
- Each field should be self-describing, including a field name as well as a value
- Record access is by exhaustive search

Variable-length records
Variable set of fields
Chronological order

(a) Pile File

The Sequential File

- Most common form of file structure
- A fixed format is used for records
- The key field uniquely identifies the record
- Typically used in batch applications that process all the records(e.g., a billing or payroll application)
- Only organization that is easily stored on tape as well as disk

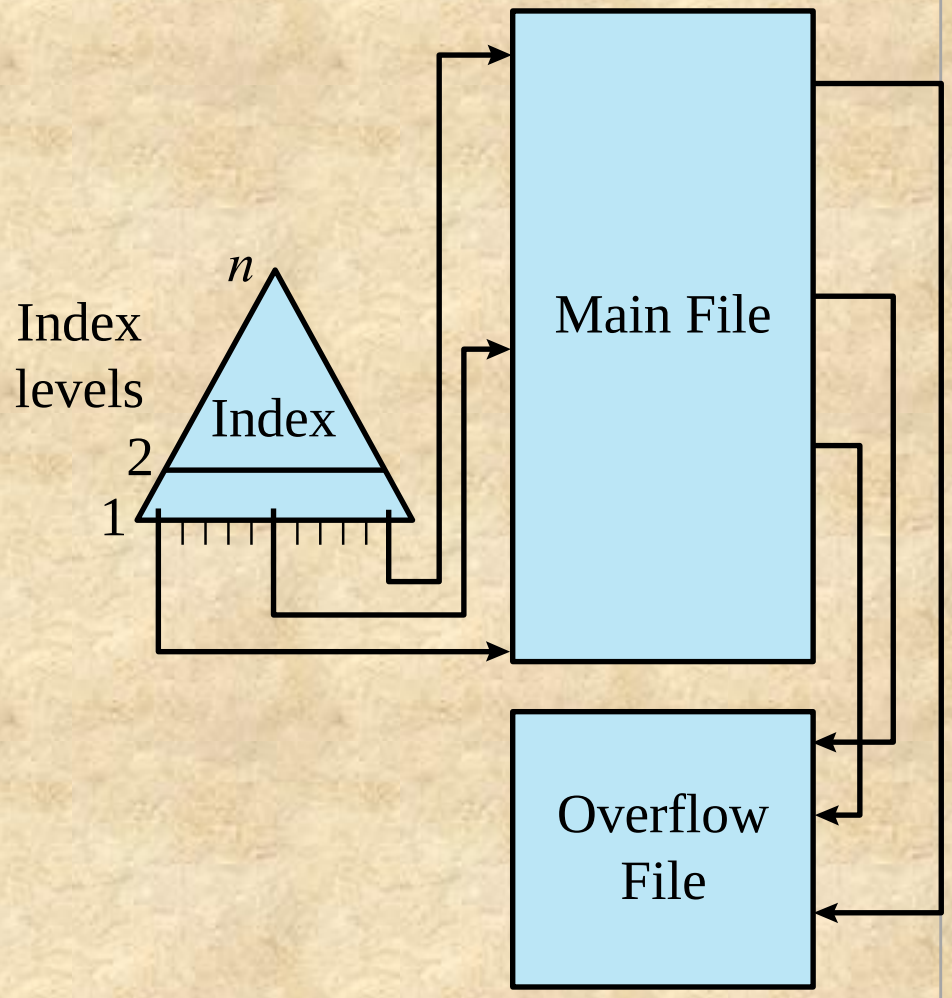


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Indexed Sequential File

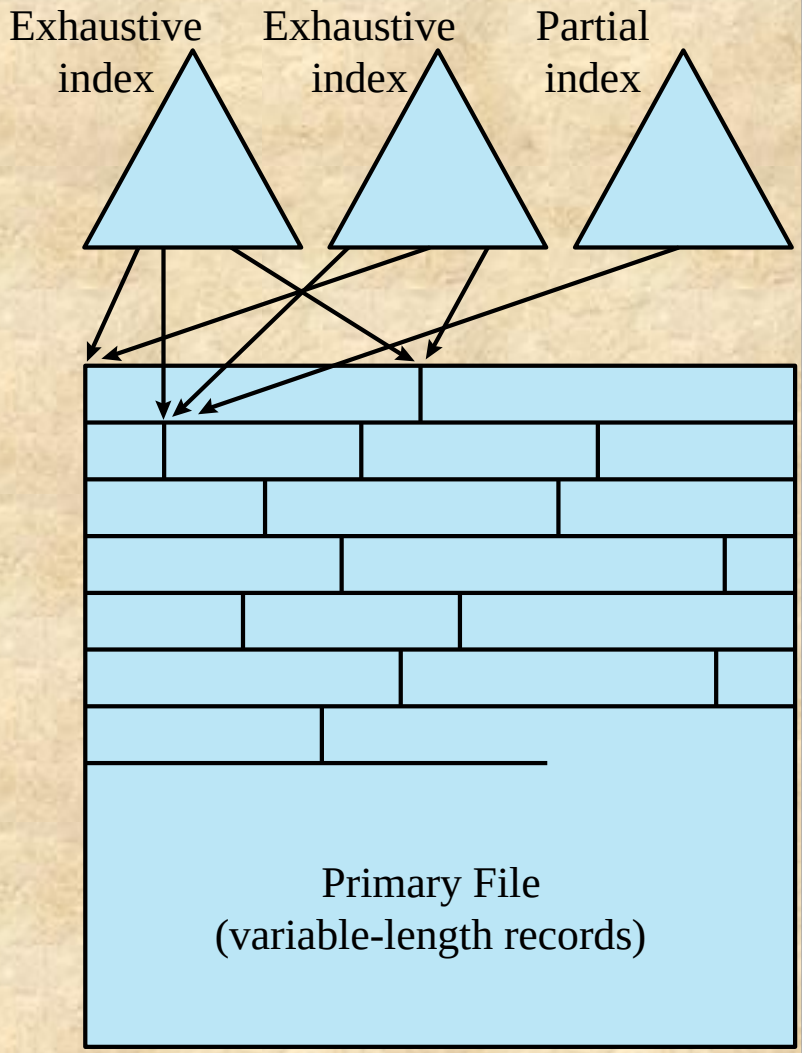
- Used to overcome the disadvantages of the sequential file
- Records are organized in sequence based on a key field
- Adds an index to the file to support random access and an overflow file
- Each record in the index file contains:
  - A key field
  - A pointer into the main file
- Search the index to find the highest key value that is equal to or precedes the desired key value
  - greatly reduces the time required to access a single record
- To process the entire file sequentially, the records of the main file are processed in sequence until a pointer to the overflow file is found, then accessing continues in the overflow file until a null pointer is encountered, at which time accessing of the main file is resumed where it left off



(c) Indexed Sequential File

Indexed File

- Support flexibility of efficiently searching by various attributes through multiple indexes
- Records are accessed only through their indexes
  - there is at least one index for each record
  - variable-length records can be employed
- Two types of indexes
  - exhaustive index contains one entry for every record in the main file
  - partial index contains entries to records where the field of interest exists
  - the index itself is organized as a sequential file for ease of searching
- Used mostly in applications where timeliness of information is critical
  - airline reservation systems
  - inventory control systems



(d) Indexed File

Direct or Hashed File

- Access directly any block of a known address
- Makes use of hashing on the key value
- Often used where:
  - very rapid access is required
  - fixed-length records are used
  - records are always accessed one at a time

Examples are:

- directories
- pricing tables
- schedules
- name lists

12.4 File Directories

Table 12.1 Information Elements of a File Directory

Basic Information	
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
Address Information	
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

(Table can be found on page 567 in textbook)

Structure

- To understand the requirements for a file structure, it is helpful to consider the types of operations that may be performed on the directory:

Search → Create files → Delete files → List directory → Update directory

Two-Level Scheme

There is one directory for each user and a master directory

The master directory has an entry for each user directory providing address and access control information

Each user directory is a simple list of the files of that user

Names must be unique only within the collection of files of a single user

The file system can easily enforce access restriction on directories

- A more powerful and flexible approach is the hierarchical, or tree-structure directory
- At any level, a directory may consist of entries for subdirectories and/or entries for files
- Directory organization
  - a sequential file
  - a hashed structure

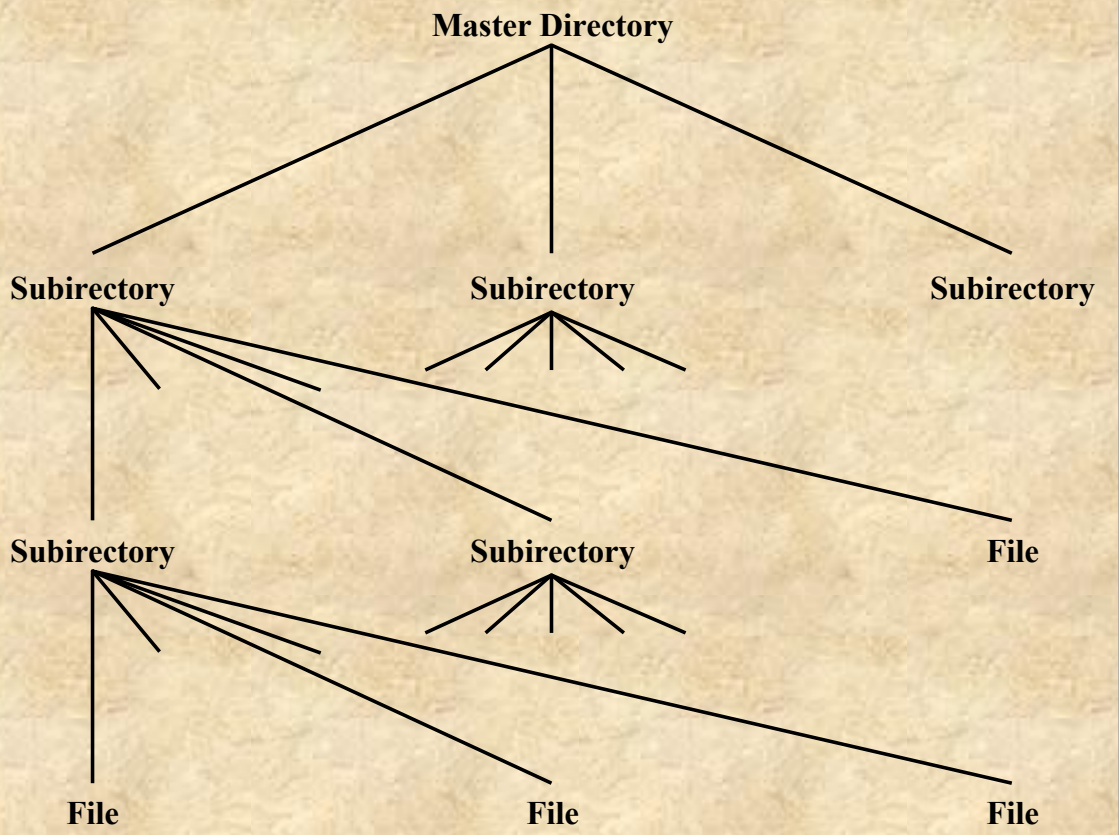


Figure 12.6 Tree-Structured Directory

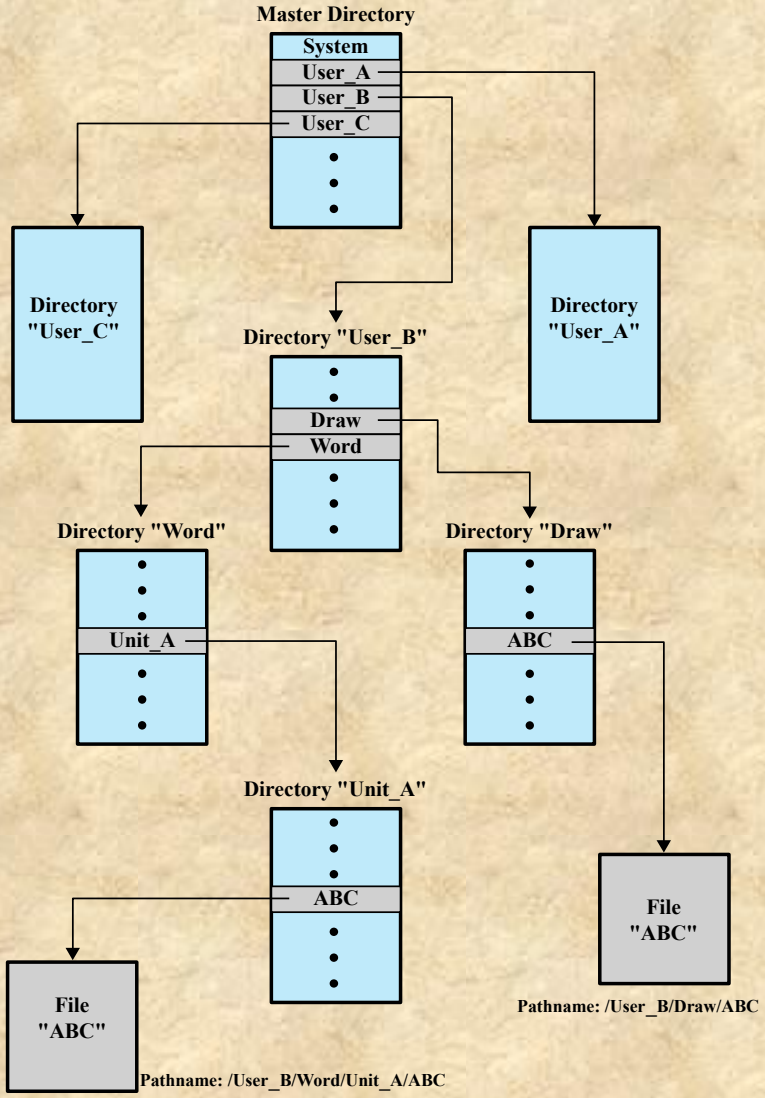


Figure 12.7 Example of Tree-Structured Directory

12.5 File Sharing

Two issues arise when allowing files to be shared among a number of users:

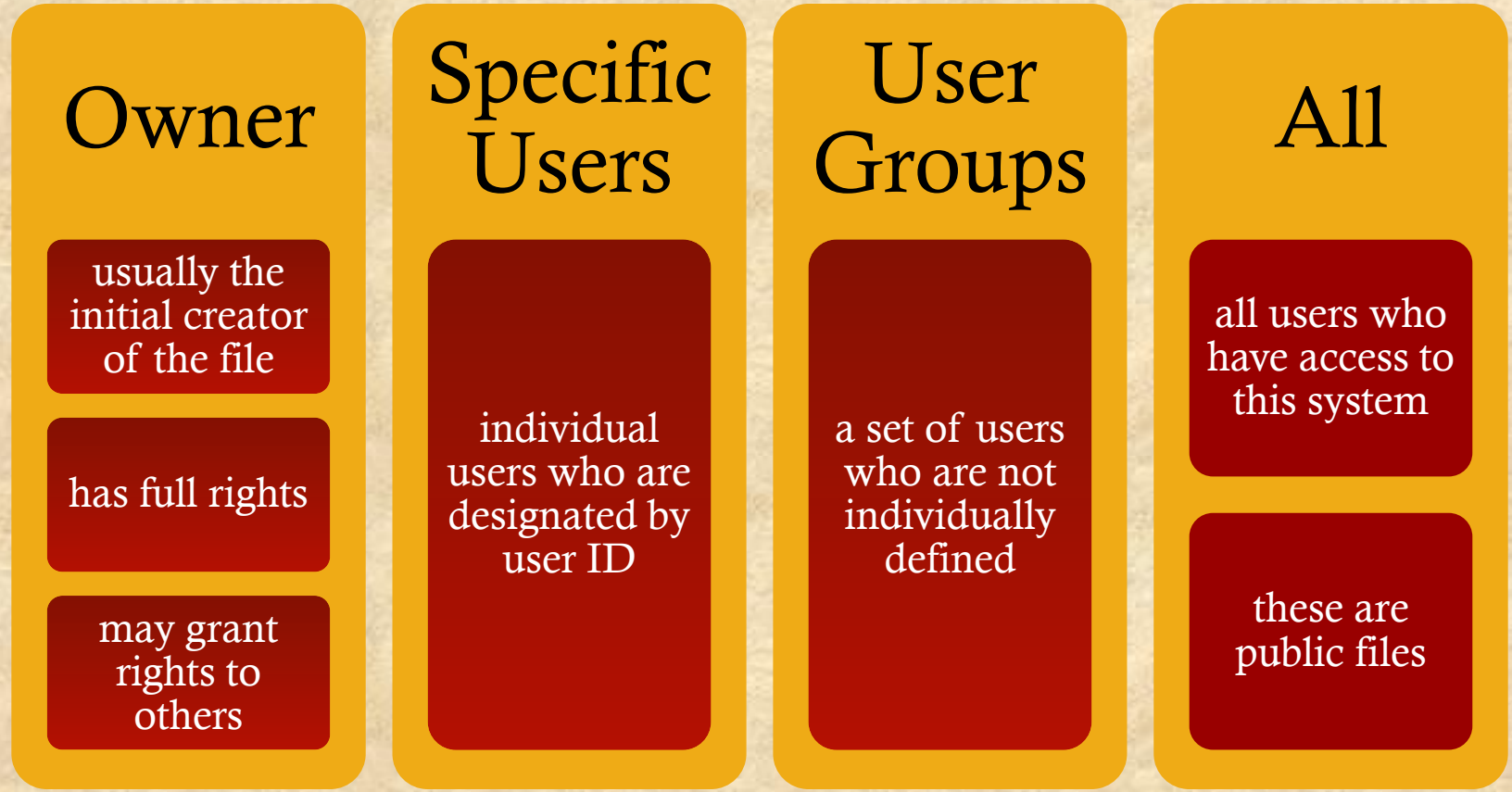
- access rights
- management of simultaneous access

Access Rights

- Users or groups of users are granted certain access rights to a file
- Representative access rights
- ***None*:** the user would not be allowed to read the user directory that includes the file
- ***Knowledge*:** the user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
- ***Execution*:** the user can load and execute a program but cannot copy it
- ***Reading*:** the user can read the file for any purpose, including copying and execution
- ***Appending*:** the user can add data to the file but cannot modify or delete any of the file's contents
- ***Updating*:** the user can modify, delete, and add to the file's data
- ***Changing protection*:** the user can change the access rights granted to other users
- ***Deletion*:** the user can delete the file from the file system

User Access Rights

- Access rights provided to different classes of users



12.6 Record Blocking

- Blocks are the unit of I/O with secondary storage
  - for I/O to be performed, records must be organized as blocks

- Given the size of a block, three methods of blocking can be used:

- Fixed-Length Blocking
  - fixed-length records are used, and an integral number of records are stored in a block
  - unused space at the end of each block (***Internal fragmentation***)
  - common mode for sequential files
- **Variable-Length Spanned Blocking**
  - fix variable-length records are used and are packed into one or more blocks with no unused space
  - efficient of storage and no limit on record size
  - Difficult to implement
  - Spanned blocks require more than one I/O operation
  - files are difficult to update, regardless of the organization
- **Variable-Length Unspanned Blocking**
  - variable-length records are used, but spanning is not employed
  - wasted space in most blocks
  - limits record size to the size of a block

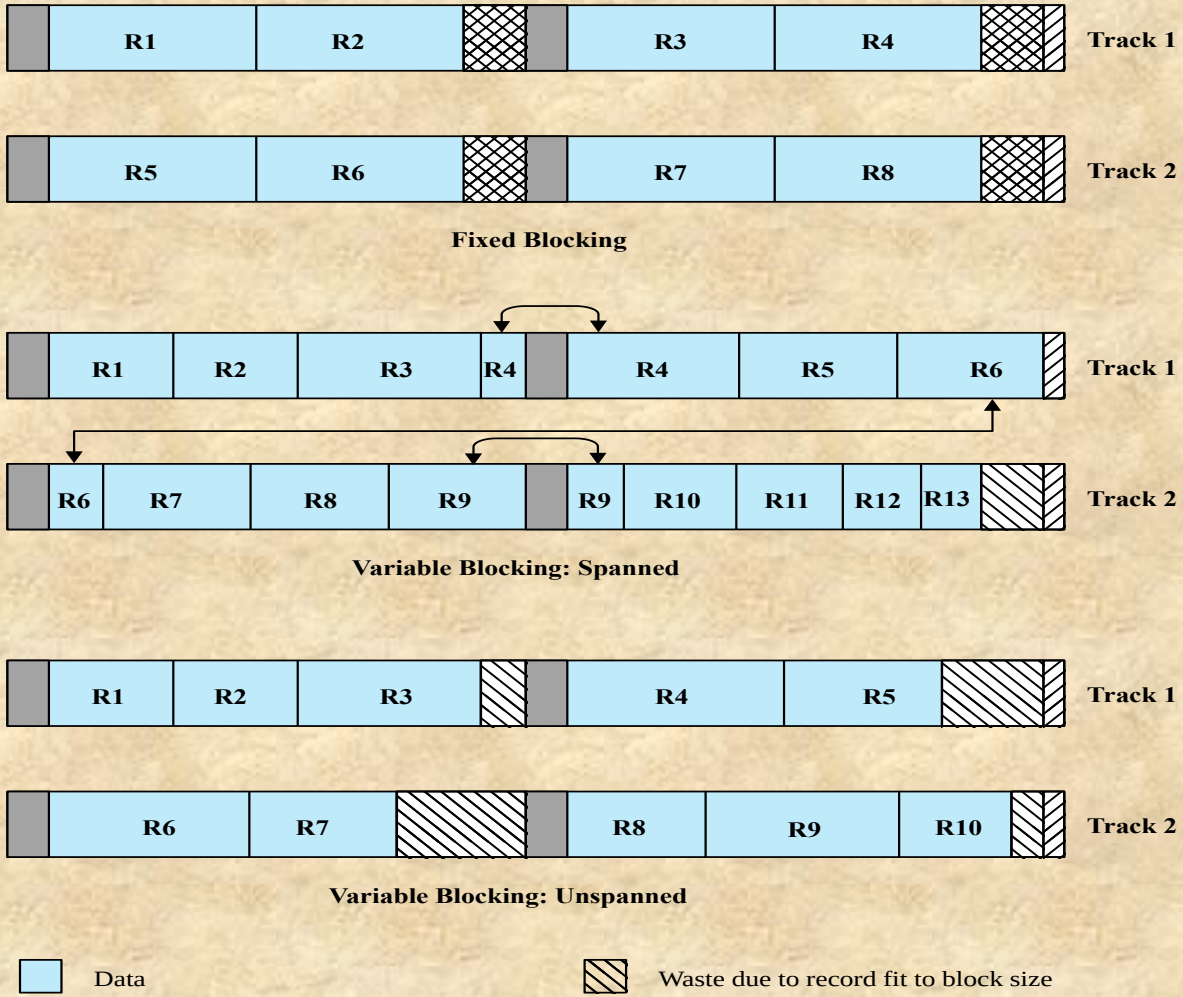


Figure 12.8 Record Blocking Methods [WIED87]

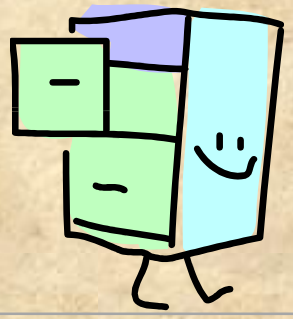
12.7 Secondary Storage Management

File Allocation

- On secondary storage, a file consists of a collection of blocks
- The operating system or file management system is responsible for allocating blocks to files
- The approach taken for file allocation may influence the approach taken for free space management
- Space is allocated to a file as one or more ***portions*** (a contiguous set of allocated blocks)
- ***File allocation table (FAT)***
  - data structure used to keep track of the portions assigned to a file

Preallocation vs Dynamic Allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
- For many applications, it is difficult to estimate reliably the maximum potential size of the file
  - tends to be wasteful because users and application programmers tend to overestimate file size
- Dynamic allocation allocates space to a file in portions as needed



Portion Size

- In choosing a portion size, there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency
- Items to be considered:
  1) contiguity of space increases performance
     - especially for Retrieve_Next operations
     - greatly for transactions running in a transaction-oriented operating system
  2) having a large number of small portions increases the size of tables needed to manage the allocation information
  3) having fixed-size portions simplifies the reallocation of space
  4) having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation

Alternatives

Two major alternatives:

Variable, large contiguous portions

- provides better performance
- the variable size avoids waste
- the file allocation tables are small
- space is hard to reuse

Blocks

- small fixed portions provide greater flexibility
- they may require large tables or complex structures for their allocation
- contiguity has been abandoned as a primary goal
- blocks are allocated as needed

File Allocation Methods

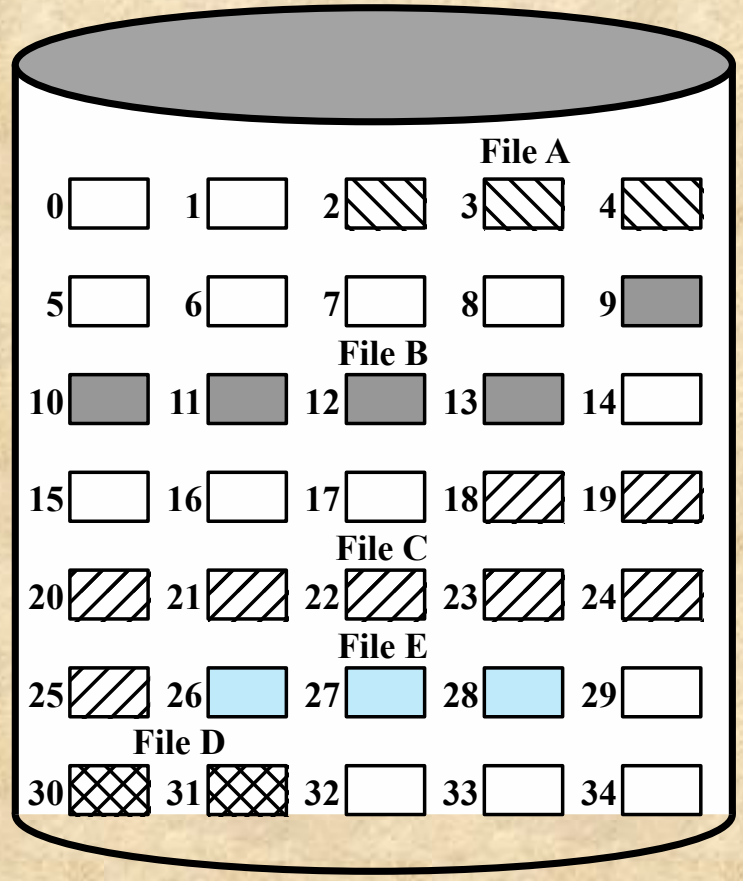
- Contiguous allocation
  - a preallocation strategy, using variable-size portions
  - FAT needs just a single entry for each file with the starting block and the length of the file
  - the best from the point of view of the individual sequential file
  - Advantages
    - Multiple blocks can be read in at a time to improve I/O performance for sequential processing
    - easy to retrieve a single i^{th} block (Ex: $b + i - 1$ if a file starts at block b)
  - Problems
    - External fragmentation
    - Need to perform compaction algorithm from time to time
    - periodic consolidation of files may be necessary

- Chained allocation
  - the opposite extreme from contiguous allocation
  - allocation is on an individual block basis with a pointer to the next block in the chain
  - FAT needs just a single entry for each file with the starting block and the length of the file
  - although preallocation is possible, it is more common to allocate blocks dynamically
  - Advantages
    - no external fragmentation
    - best suited to sequential files that are to be processed sequentially
  - Problems
    - no accommodation of the principle of locality
    - if it is necessary to bring in several blocks of a file at a time, as in sequential processing, then a series of accesses to different parts of the disk are required

- Indexed allocation
  - FAT contains a separate one-level index for each file
    - the index has one entry for each portion allocated to the file
    - the file indexes are not physically stored as part of the FAT, but are kept in a separate block
  - allocation may be on the basis of either fixed-size blocks (Figure 12.13) or variable-size portions (Figure 12.14)
  - supports both sequential and direct access to the file
    - the most popular form of file allocation
  - advantages
    - allocation by blocks eliminates external fragmentation, whereas allocation by variable-size portions improves locality
  - problems
    - file consolidation may be done from time to time
      - reduces the size of the index in the case of variable-size portions, but not in the case of block allocation

Table 12.2
File Allocation Methods

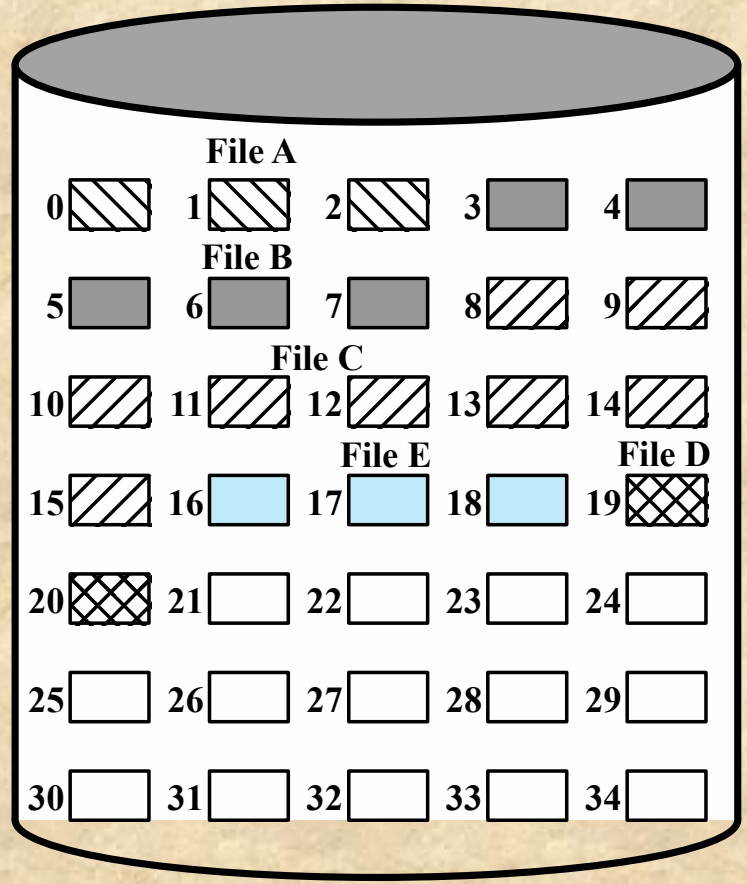
	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

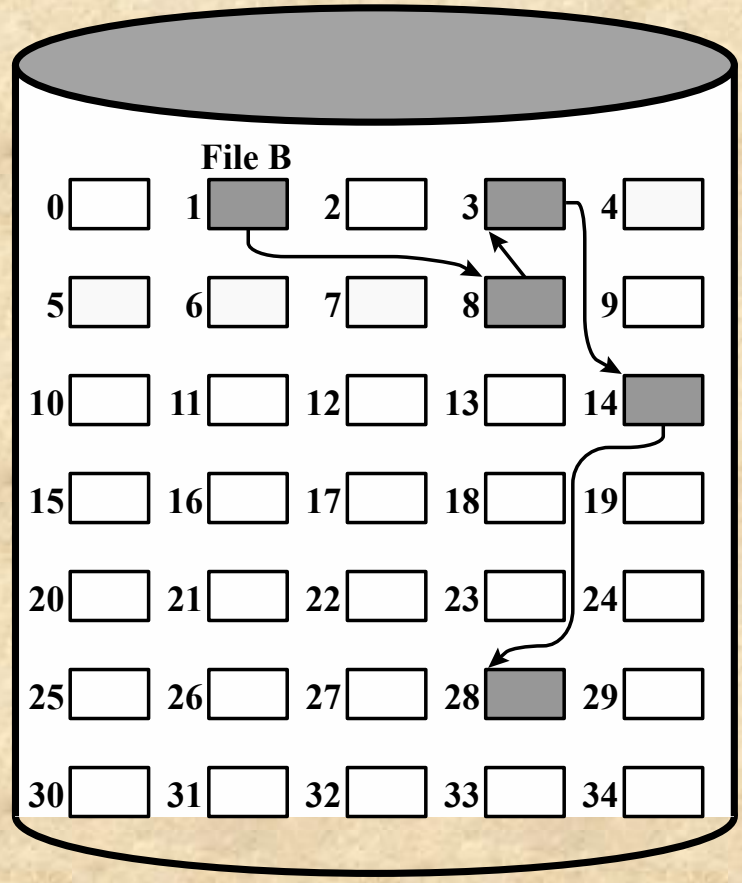
Figure 12.9 Contiguous File Allocation



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

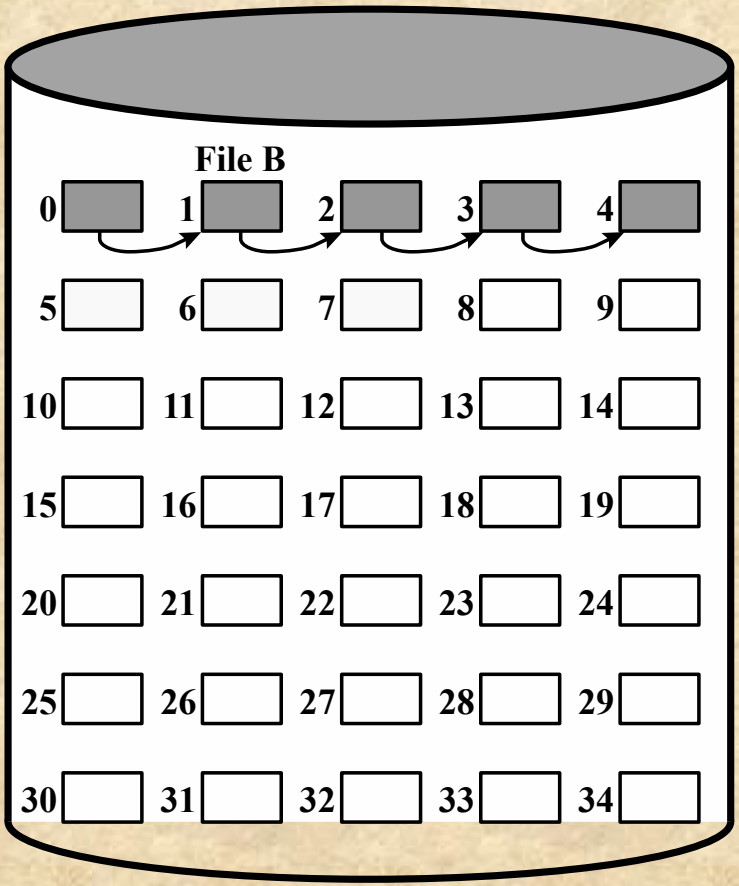
Figure 12.10 Contiguous File Allocation (After Compaction)



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.11 Chained Allocation



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.12 Chained Allocation (After Consolidation)

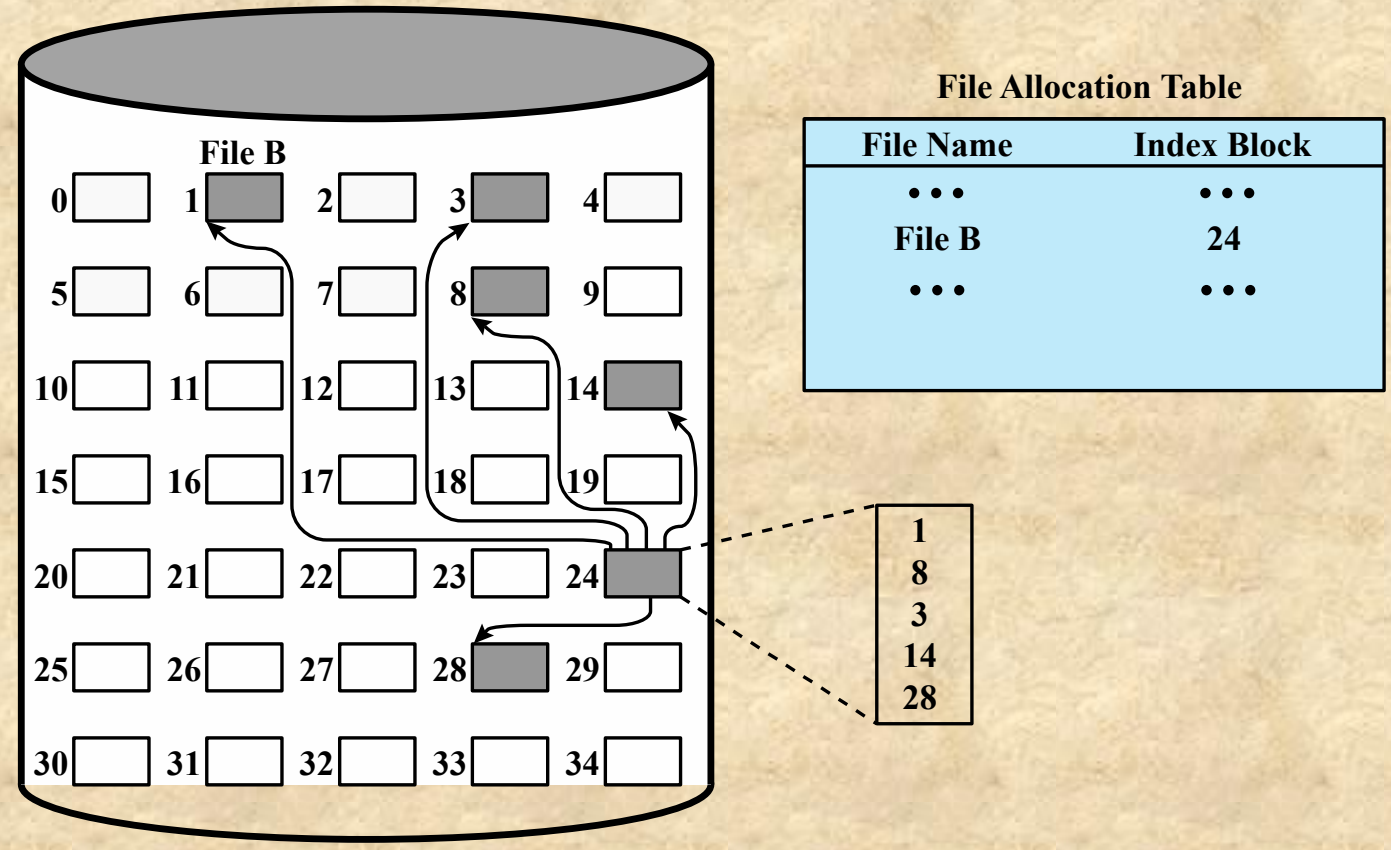


Figure 12.13 Indexed Allocation with Block Portions

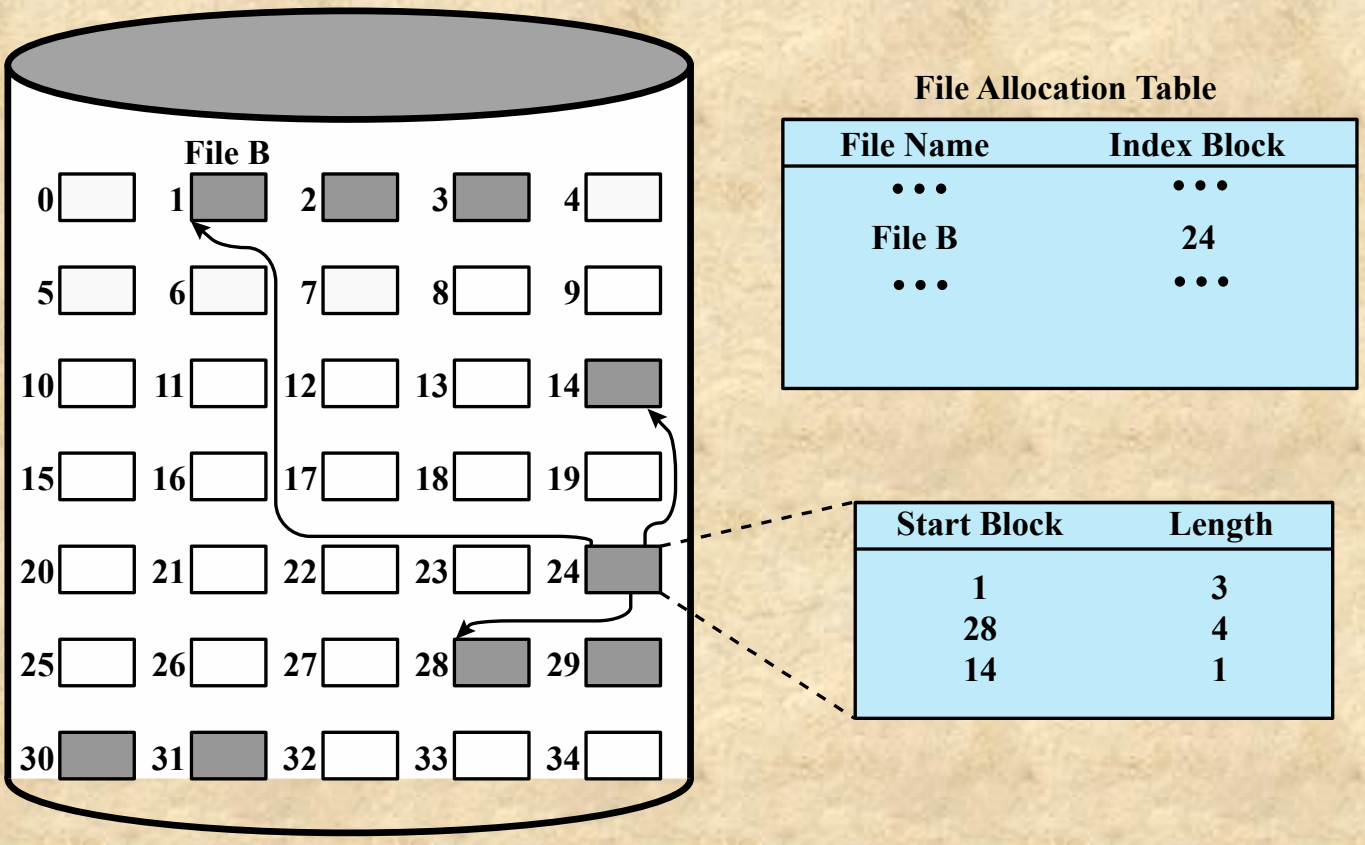
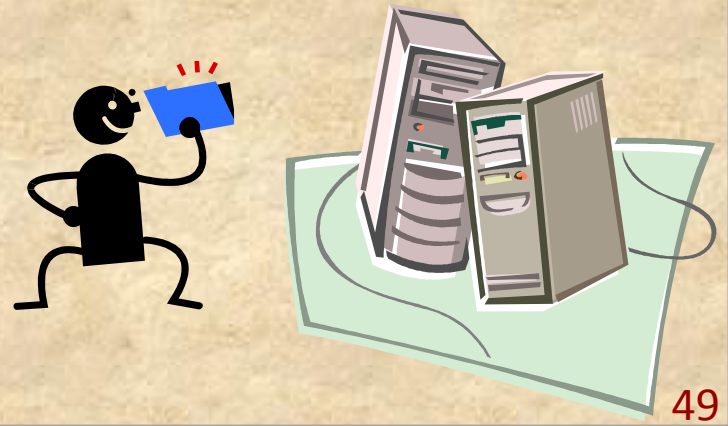


Figure 12.14 Indexed Allocation with Variable-Length Portions

Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- To perform file allocation, it is necessary to know which blocks are available
- A ***disk allocation table*** is needed in addition to a file allocation table



Bit Tables

- Uses a vector containing one bit for each block on the disk
- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use
- for the disk layout of TP43, a vector of length 35 is needed as below:
  00111000011111000011111111111011000

Advantages:

- relatively easy to find one or a contiguous group of free blocks
  - works well with any file allocation method
- it is as small as possible

Chained Free Portions

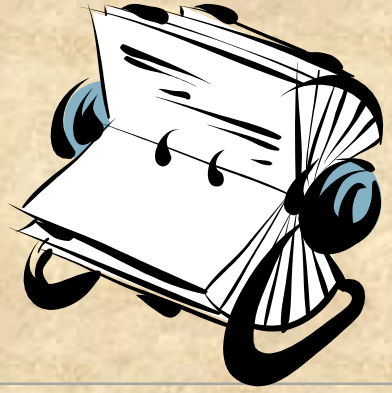
- The free portions may be chained together by using a pointer and length value in each free portion
- Negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first portion
- Suited to all file allocation methods

Disadvantages:

- Leads to fragmentation
  - many portions will be a single block long after some use
- Every time you allocate a block, you need to read the block first to recover the pointer to the new first free block before writing data to that block
  - greatly slows file creation and deleting highly fragmented files is very time consuming

Indexing

- Treats free space as a file and uses an index table as it would for file allocation
- For efficiency, the index should be on the basis of variable-size portions rather than blocks
  - one entry in the table for every free portion on the disk
- This approach provides efficient support for all of the file allocation methods



Free Block List

Each block is assigned a number sequentially

- the list of the numbers of all free blocks is maintained in a reserved portion of the disk

Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number

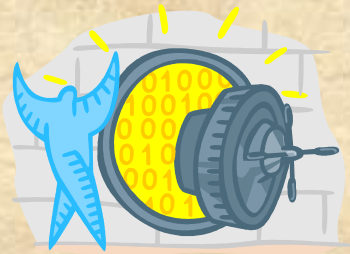
- the size of the free block list is the number of 512-byte block times the size of the corresponding bit table and must be stored on disk

There are two effective techniques for storing a small part of the free block list in main memory:

- the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory
- the list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory

Volumes

- A collection of addressable sectors in secondary memory that an OS or application can use for data storage
- The sectors in a volume need not be consecutive on a physical storage device
  - they need only appear that way to the OS or application
- A volume may be the result of assembling and merging smaller volumes



12.8 UNIX File Management

- In the UNIX file system, six types of files are distinguished:

Regular, or ordinary
- contains arbitrary data in zero or more data blocks

Directory
- contains a list of file names plus pointers to associated *inodes*

Special
- contains no data but provides a mechanism to map physical devices to file names

Named pipes
- an interprocess communications facility

Links
- an alternative file name for an existing file

Symbolic links
- a data file that contains the name of the file it is linked to

Inodes

- All types of UNIX files are administered by the OS by means of *inodes*
- An *inode* (index node) is a control structure that contains the key information needed by the operating system for a particular file
- Several file names may be associated with a single *inode*
  - an active *inode* is associated with exactly one file
  - each file is controlled by exactly one *inode*

Data elements of Inodes

- The type and access mode of the file
- The file's owner and group-access identifiers
- The time that the file was created, when it was most recently read and written, and when its *inode* was most recently updated by the system
- The size of the file in bytes
- A sequence of block pointers, explained in the next subsection
- The number of physical blocks used by the file, including blocks used to hold indirect pointers and attributes

- The number of directory entries that reference the file
- The kernel and user-settable flags that describe the characteristics of the file
- The generation number of the file (a randomly selected number assigned to the *inode* each time that the latter is allocated to a new file; the generation number is used to detect references to deleted files)
- The blocksize of the data blocks referenced by the *inode* (typically the same as, but sometimes larger than, the file system blocksize)
- The size of the extended attribute information
- Zero or more extended attribute entries
  - the author of a document, the character encoding of a plain-text document, or a checksum, digital certificate, etc.

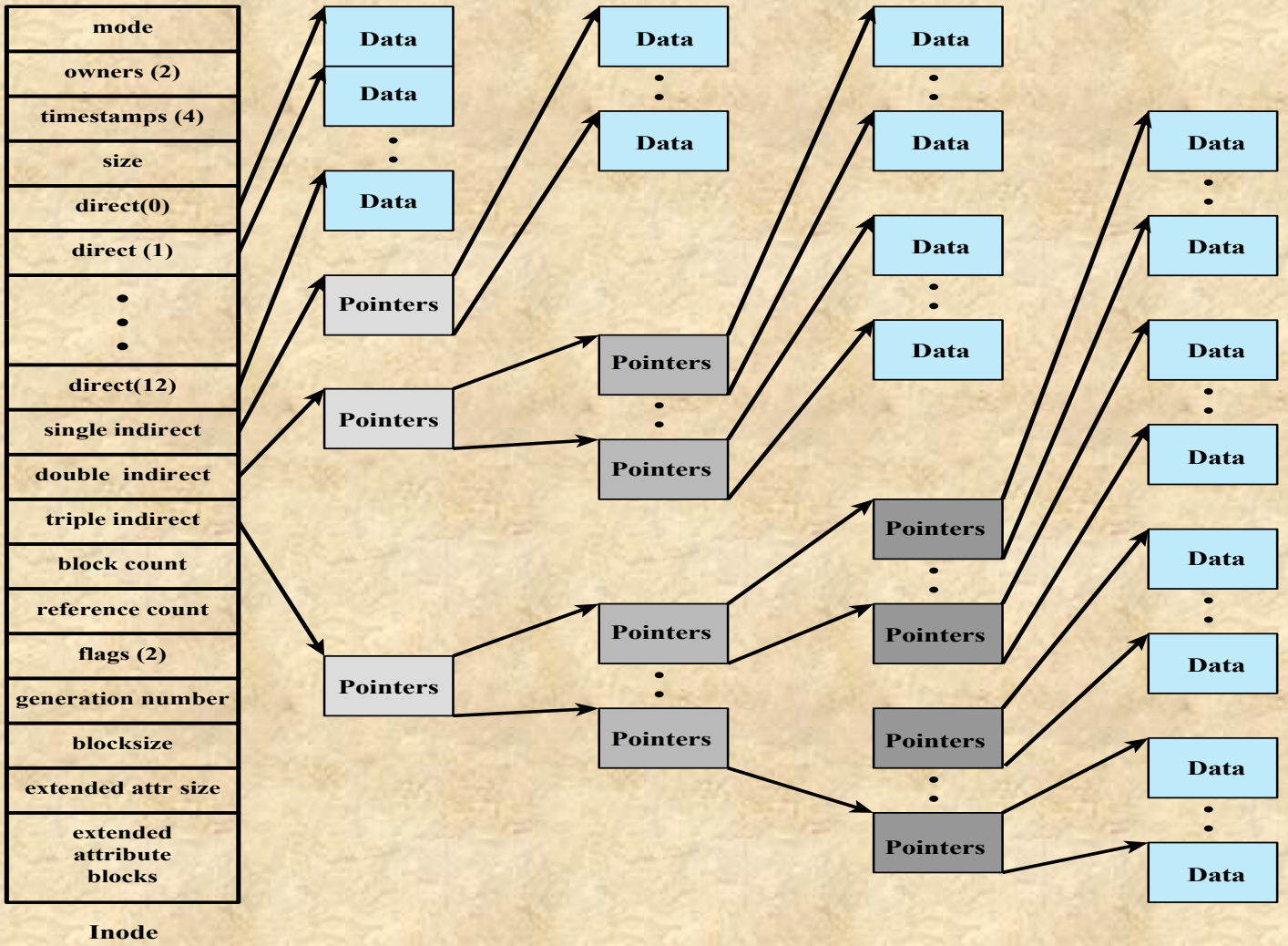


Figure 12.15 Structure of FreeBSD inode and File

File Allocation

- File allocation is done on a block basis
- Allocation is dynamic, as needed, rather than using preallocation
- An indexed method is used to keep track of each file, with part of the index stored in the inode for the file
- In all UNIX implementations, the *inode* includes a number of direct pointers and three indirect pointers (single, double, triple)

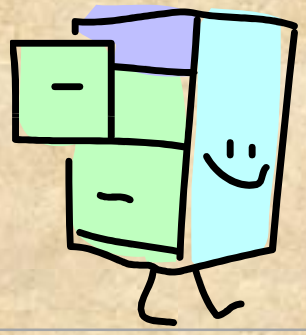


Table 12.3
Capacity of a FreeBSD File with 4-Kbyte Block Size

Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	512 ´ 512 = 256K	1G
Triple Indirect	512 ´ 256K = 128M	512G

Directories

- Structured in a hierarchical tree
- Each directory
  - contain files and/or other directories
  - simply a file that contains a list of file names plus pointers to associated *inodes*
- Each directory entry (*dentry*) contains a name for the associated file or subdirectory plus an integer called the *i-number* (index number)
  - each *i-number* is used as an index into the *inode* table

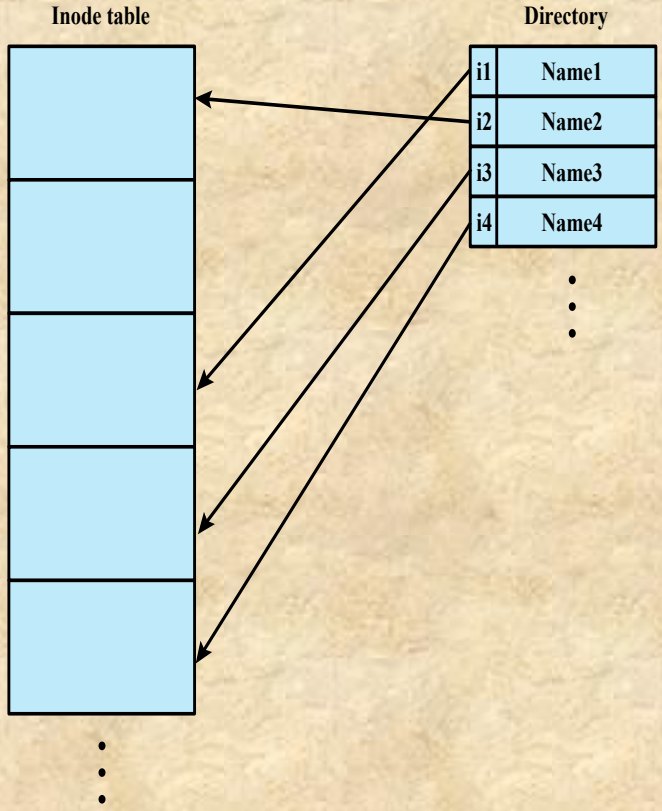
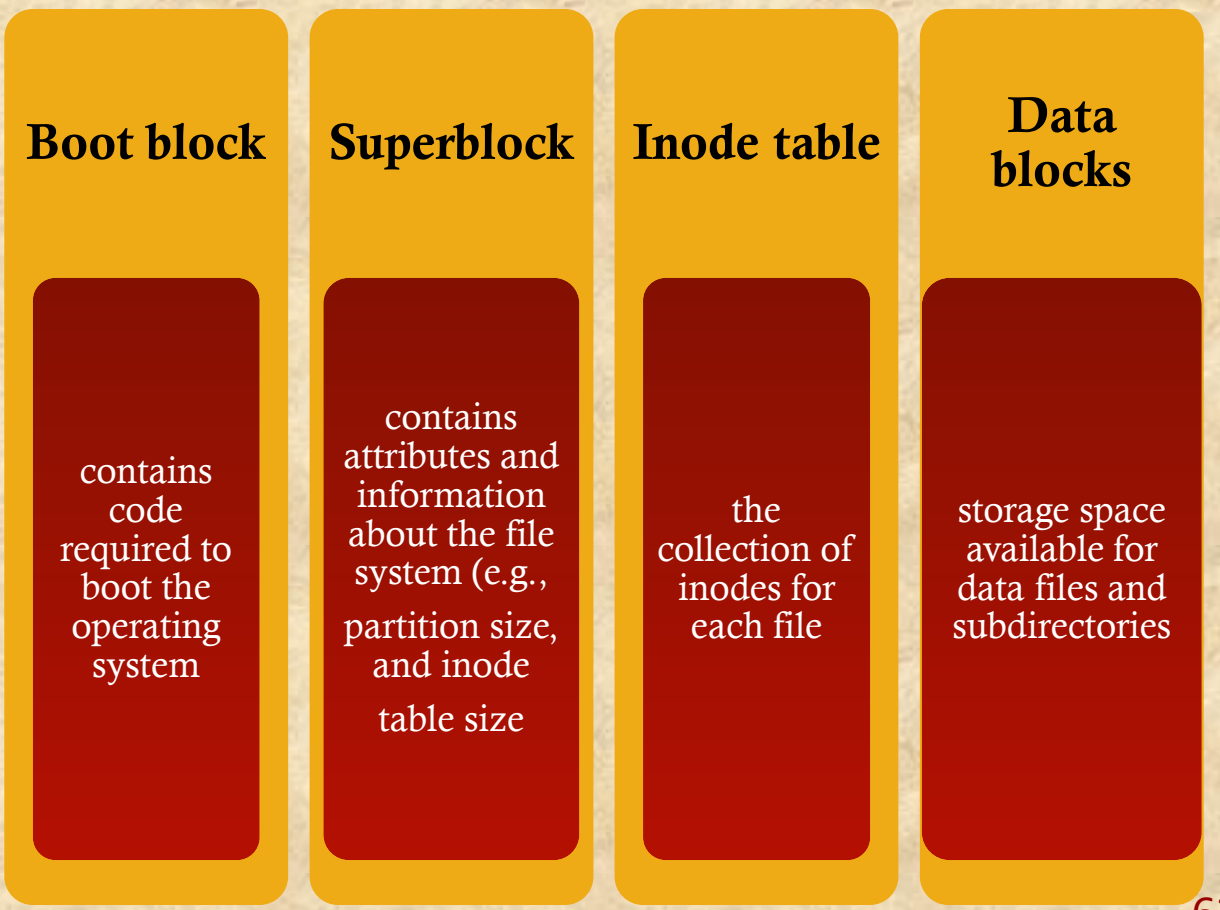


Figure 12.16 UNIX Directories and Inodes

Volume Structure

- A UNIX file system resides on a single logical disk or disk partition and is laid out with the following elements:



12.9 LINUX Virtual File System

- Presents a single, uniform file system interface to user processes
- For any specific file system, a mapping module is needed to transform the characteristics of the real file system to the characteristics expected by the virtual file system
- The key ingredients of the Linux file system strategy
  - A user process issues a file system call (e.g., read) using the VFS file scheme
  - The VFS converts this into an internal (to the kernel) file system call that is passed to a mapping function for a specific file system [e.g., ext2 FS]
    - the mapping function is simply a mapping of file system functional calls from one scheme to another
  - The original user file system call is translated into a call that is native to the target file system
  - The target file system software is then invoked to perform the requested function on a file or directory under its control and secondary storage

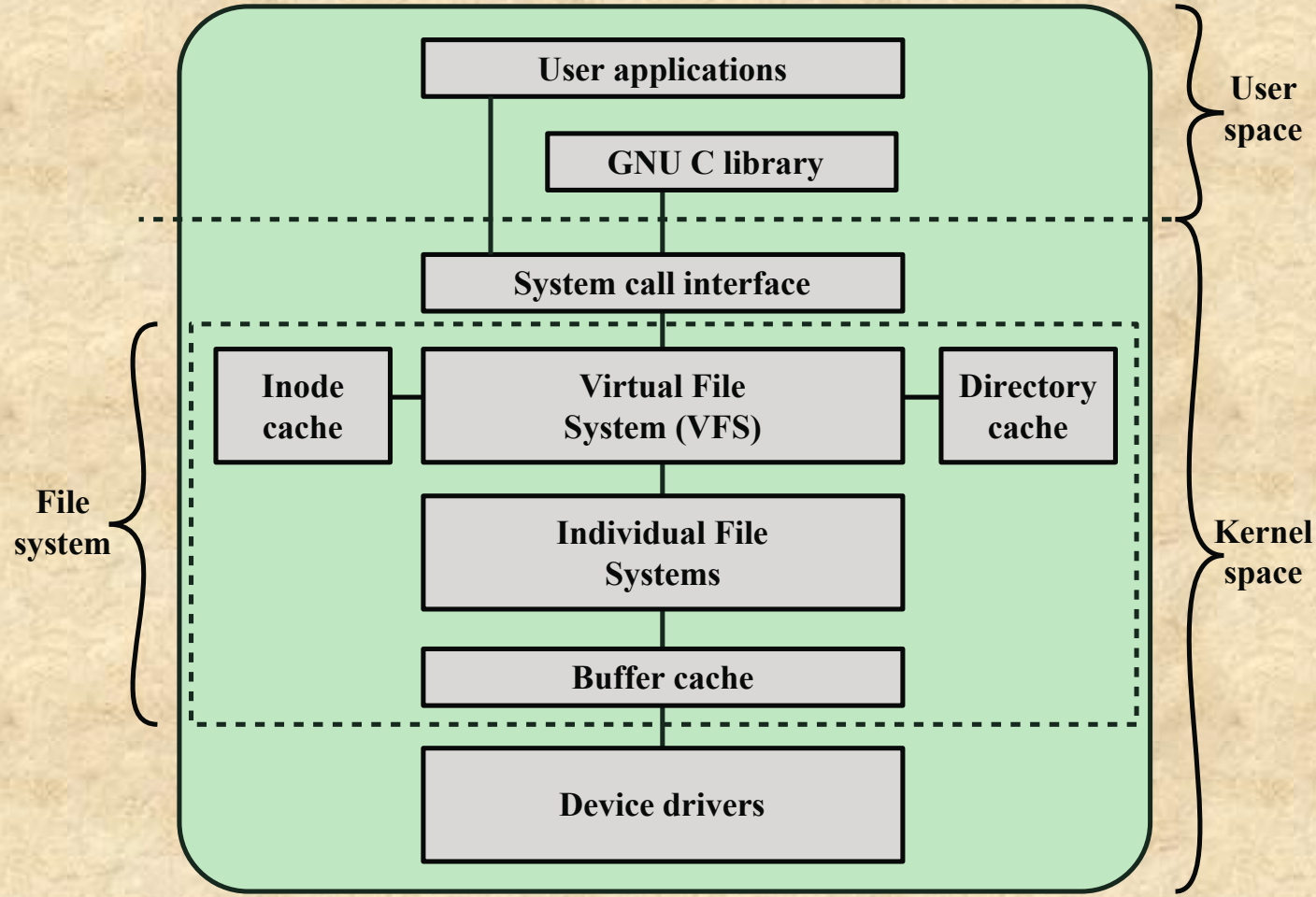


Figure 12.17 Linux Virtual File System Context

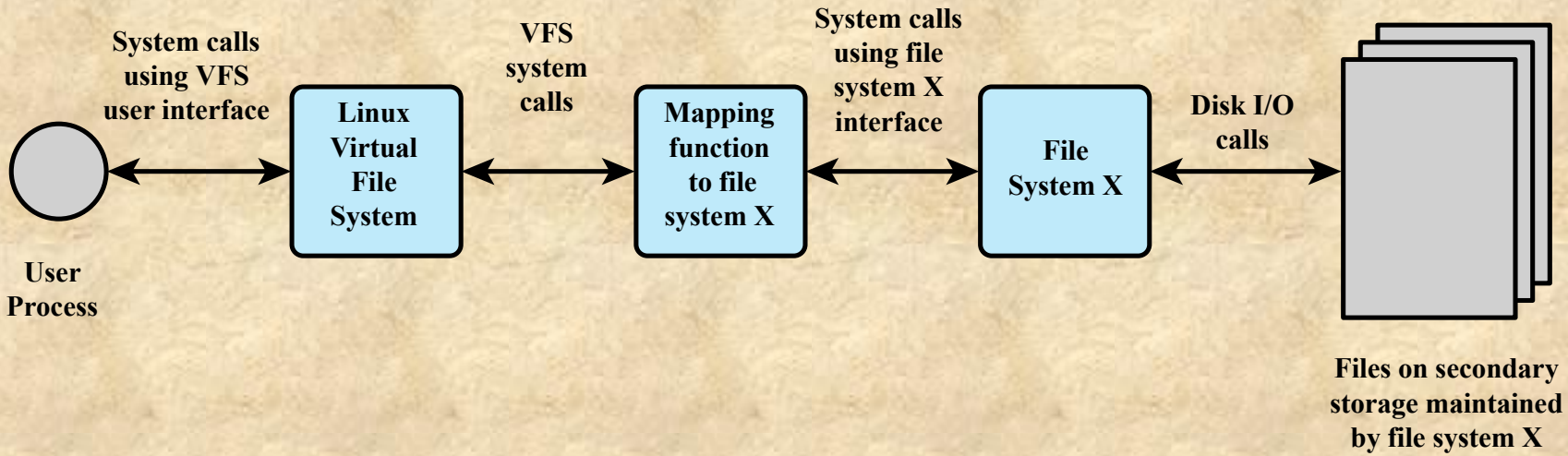


Figure 12.18 Linux Virtual File System Concept

Primary Object Types in VFS

Superblock Object

- represents a specific mounted file system

Dentry Object

- represents a specific directory entry

Inode Object

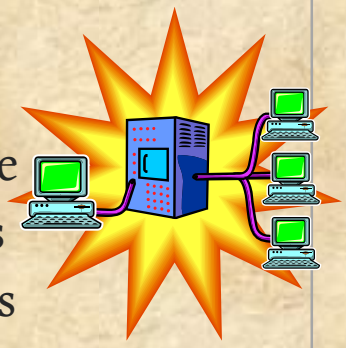
- represents a specific file

File Object

- represents an open file associated with a process

12.10 Windows File System

- The developers of Windows NT designed a new file system, the New Technology File System (NTFS) which is intended to meet high-end requirements for workstations and servers
- Key features of NTFS:
  - recoverability: the ability to recover from system crashes and disk failure
  - security
  - large disks and large files
  - multiple data streams: define multiple data streams for a single file
  - journaling: keep a log of all changes made to files on the volumes
  - compression and encryption: entire directories and individual files
  - hard and symbolic(or soft) links: provided to support POSIX
    - A hard link creates another file with a link to the same underlying inode
    - A symbolic link contains a text string that is automatically interpreted and followed by the operating system as a path to another file or directory



NTFS Volume and File Structure

- NTFS makes use of the following disk storage concepts:

Sector

- the smallest physical storage unit on the disk
- the data size in bytes is a power of 2 and is almost always 512 bytes

Cluster

- one or more contiguous sectors
- the cluster size in sectors is a power of 2

Volume

- a logical partition on a disk, consisting of one or more clusters and used by a file system to allocate space
- can be all or a portion of a single disk or it can extend across multiple disks
- the maximum volume size for NTFS is 2^{64} clusters

Figure 12.19 NTFS Volume Layout

- Partition boot sector contains information about the volume layout and the file system structures as well as boot startup information and code
- Master File Table contains information about all of the files and folders (directories)
- System Files
  - **MFT2:** A mirror of the first few rows of the MFT, used to guarantee access to the volume in the case of a single-sector failure in the sectors storing the MFT
  - **Log file:** A list of transaction steps used for NTFS recoverability
  - **Cluster bit map:** A representation of the space on the volume, showing which clusters are in use
  - **Attribute definition table:** Defines the attribute types supported on this volume and indicates whether they can be indexed and whether they can be recovered during a system recovery operation

Master File Table (MFT)

- The heart of the Windows file system is the MFT
- The MFT is organized as a table of 1,024-byte rows, called records
- Each row describes a file on this volume, including the MFT itself, which is treated as a file
- Each record in the MFT consists of a set of attributes that serve to define the file (or folder) characteristics and the file contents

Table 12.5
Windows NTFS File and Directory Attribute Types

Attribute Type	Description
Standard information	Includes access attributes (read-only, read/write, etc.); time stamps, including when the file was created or last modified; and how many directories point to the file (link count).
Attribute list	A list of attributes that make up the file and the file reference of the MFT file record in which each attribute is located. Used when all attributes do not fit into a single MFT file record.
File name	A file or directory must have one or more names.
Security descriptor	Specifies who owns the file and who can access it.
Data	The contents of the file. A file has one default unnamed data attribute and may have one or more named data attributes.
Index root	Used to implement folders.
Index allocation	Used to implement folders.
Volume information	Includes volume-related information, such as the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or folder.

Note: Colored rows refer to required file attributes; the other attributes are optional.

Recoverability

- The key elements that support recoverability
  - I/O manager
    - NTFS driver handles the basic open, close, read, and write functions
    - The software RAID module FTDISK
  - Log file service
    - Used to recover an NTFS-formatted volume from a system failure
  - Cache manager for enhancing performance
  - Virtual memory manager
    - the NTFS accesses cached files by mapping file references to virtual memory references and reading and writing virtual memory.

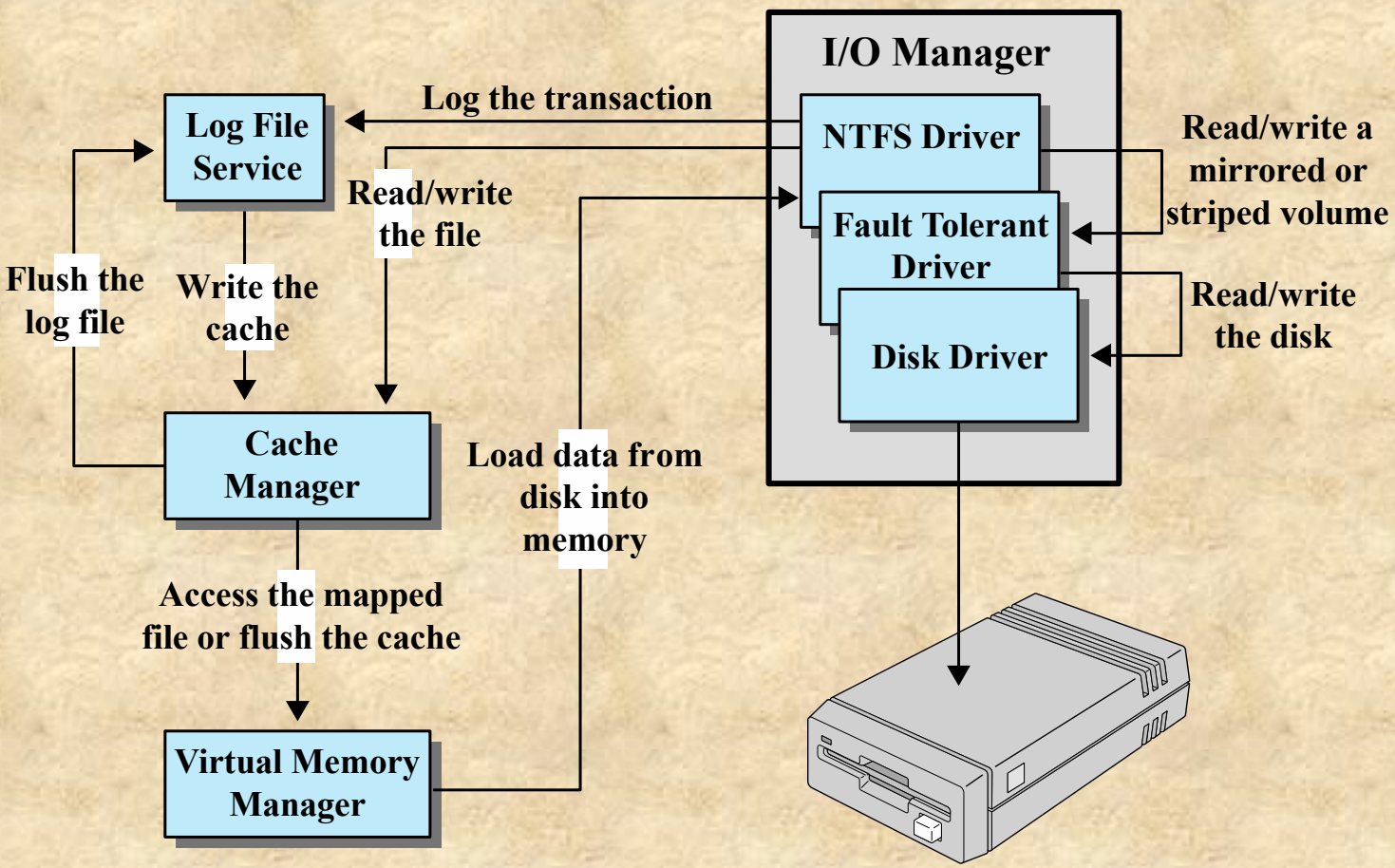
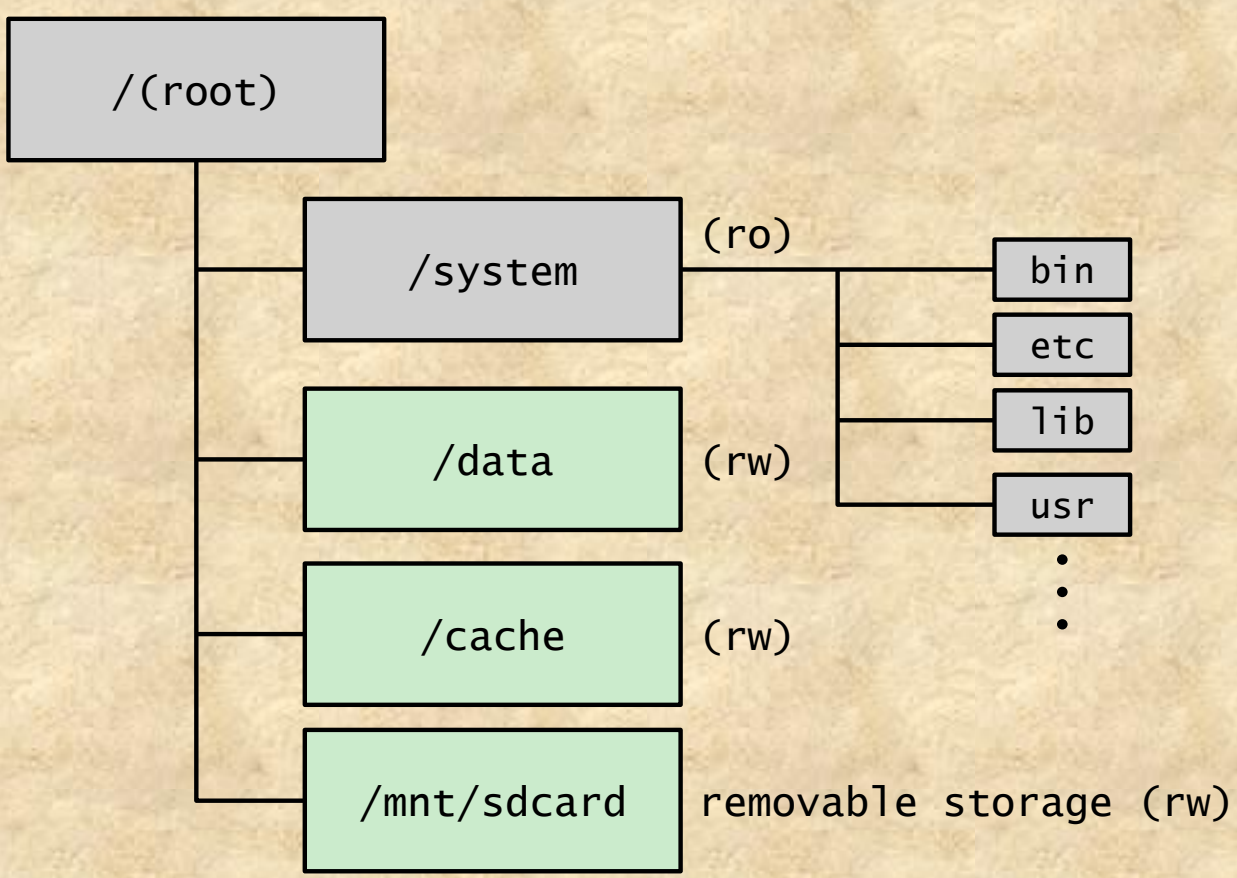


Figure 12.20 Windows NTFS Components

12.11 Android File Management

- System directory
  - contain the core parts of the operating system: system binaries, system libraries, and configuration files
  - include a basic set of Android application such as Alarmclock, Calculator, and Camera
- Data directory
  - provide the principal location used by applications to store files
- Cache directory
  - the partition where Android stores frequently accessed data and app components
- Mnt/sdcard directory
  - not a partition on the internal memory but rather on the SD(secure digital) card



ro: mounted as read only
rw: mounted as read and write

Figure 12.21 Typical Directory Tree of Android

SQLite

- Based on the Structured Query Language (SQL) developed by IBM
- Most widely deployed SQL database engine in the world
- Designed to provide a streamlined SQL-based database management system suitable for embedded systems and other limited memory systems
- The full SQLite library can be implemented in under 400 KB
- In contrast to other database management systems, SQLite is not a separate process that is accessed from the client application
  - the library is linked in and thus becomes an integral part of the application program

Summary

- File structure
- File management systems
- File organization and access
  - The pile
  - The sequential file
  - The indexed sequential file
  - The indexed file
  - The direct or hashed file
- B-Trees
- File directories
  - Contents
  - Structure
  - Naming
- File sharing
  - Access rights
  - Simultaneous access
- Record blocking
- Android file management
  - File system
  - SQLite
- Secondary storage management
  - File allocation
  - Free space management
  - Volumes
  - Reliability
- UNIX file management
  - Inodes
  - File allocation
  - Directories
  - Volume structure
- Linux virtual file system
  - Superblock object
  - Inode object
  - Dentry object
  - File object
  - Caches
- Windows file system
  - Key features of NTFS
  - NTFS volume and file structure
  - Recoverability