Operating Systems: Internals and Design Principles

Chapter 12
File Management

Seventh Edition

By William Stallings

Operating Systems: Internals and Design Principles

If there is one singular characteristic that makes squirrels unique among small mammals it is their natural instinct to hoard food. Squirrels have developed sophisticated capabilities in their hoarding. Different types of food are stored in different ways to maintain quality. Mushrooms, for instance, are usually dried before storing. This is done by impaling them on branches or leaving them in the forks of trees for later retrieval. Pine cones, on the other hand, are often harvested while green and cached in damp conditions that keep seeds from ripening. Gray squirrels usually strip outer husks from walnuts before storing.

— SQUIRRELS: A WILDLIFE HANDBOOK, Kim Long

Files

- 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

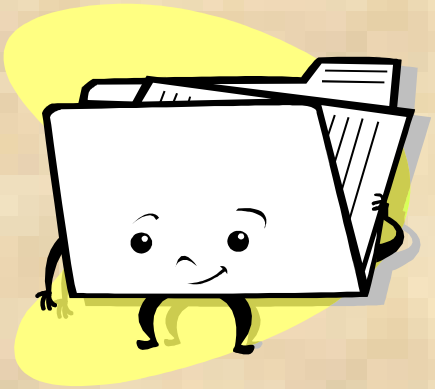
- files have names and can have associated access permissions that permit controlled sharing

Structure

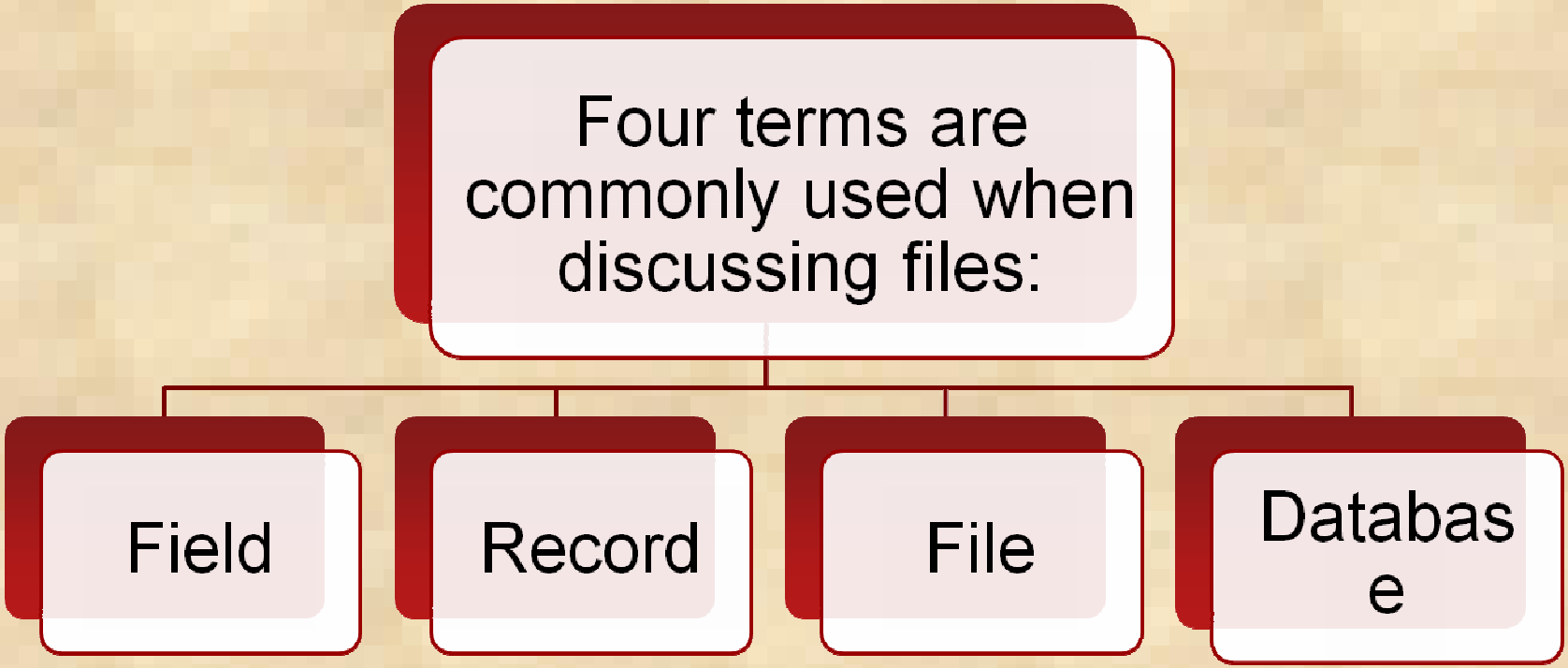
- files can be organized into hierarchical or more complex structure to reflect the relationships among files

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 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

Database

- collection of related data
- 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

- 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

File Management System 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 restructure the files in a form appropriate to the problem
5. should be able to move data between files
6. should be able to back up and recover files in case of damage
7. should be able to access his or her files by name rather than by numeric identifier

Typical Software Organization

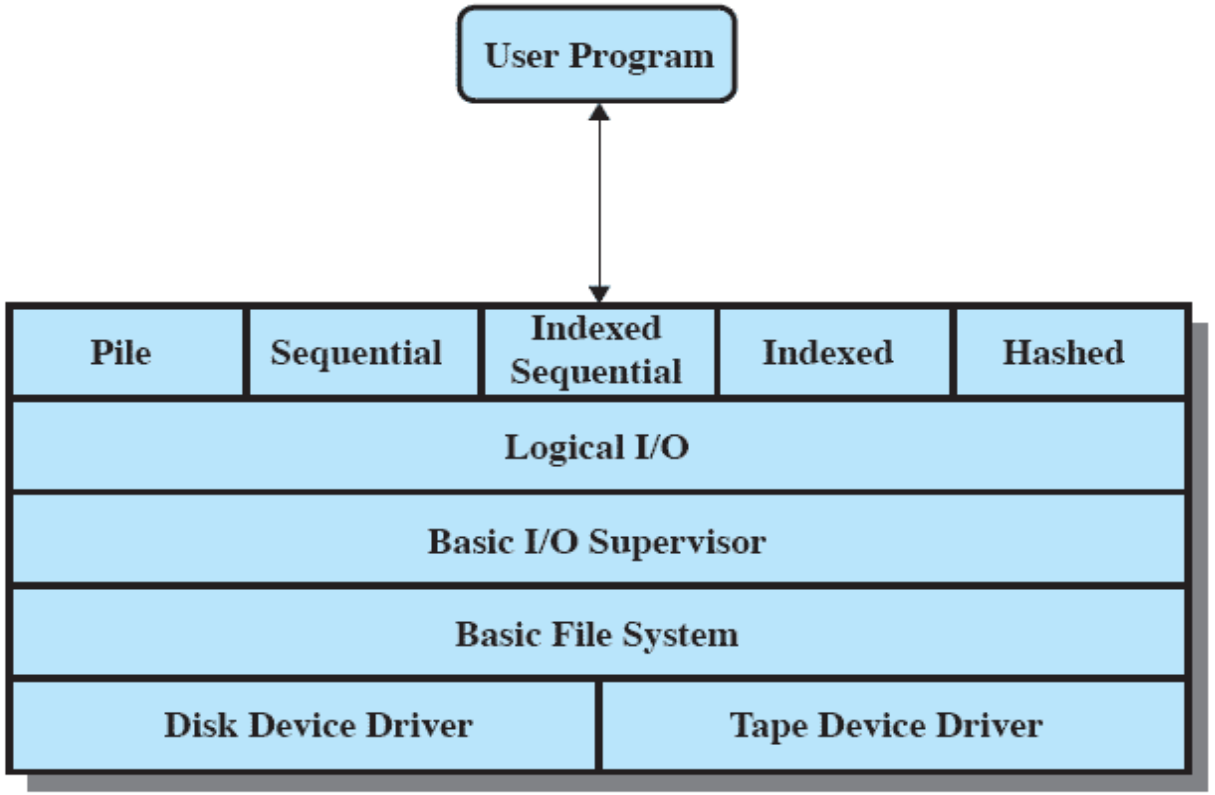
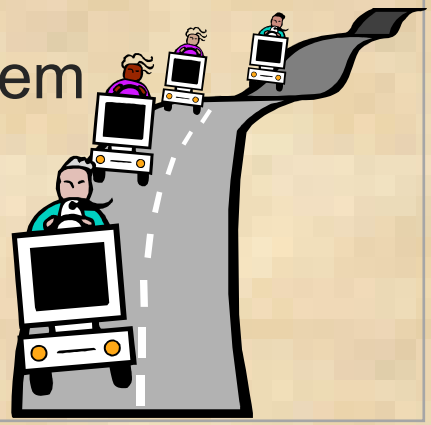


Figure 12.1 File System Software Architecture

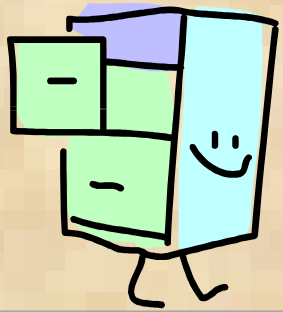
Device Drivers

- 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

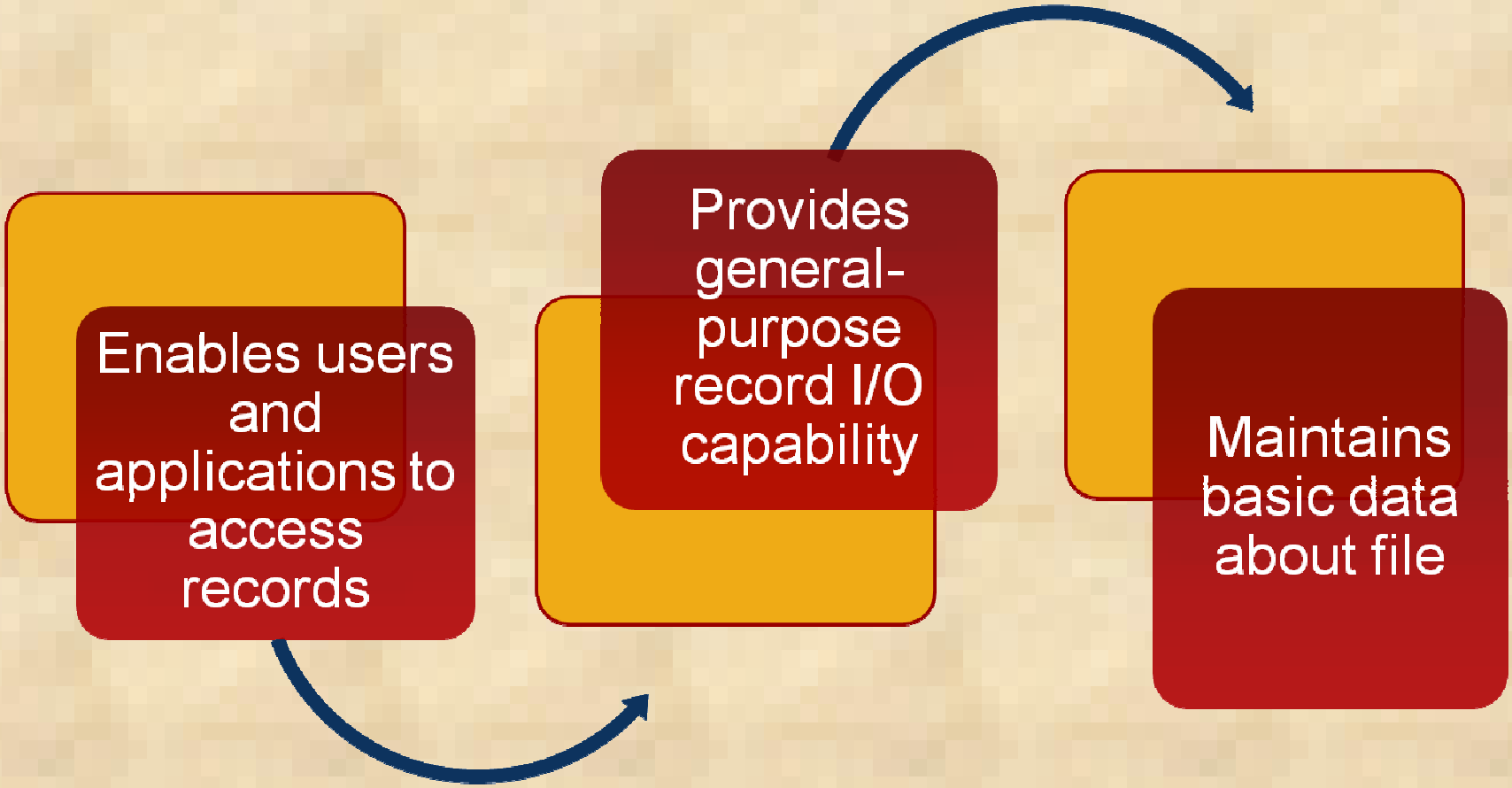
- Also referred to as the physical I/O level
- Primary interface with the environment outside the computer system
- Deals with blocks of data that are exchanged with disk or tape systems
- Concerned with the placement of blocks on the secondary storage device
- Concerned with buffering blocks in main 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

Elements of File Management

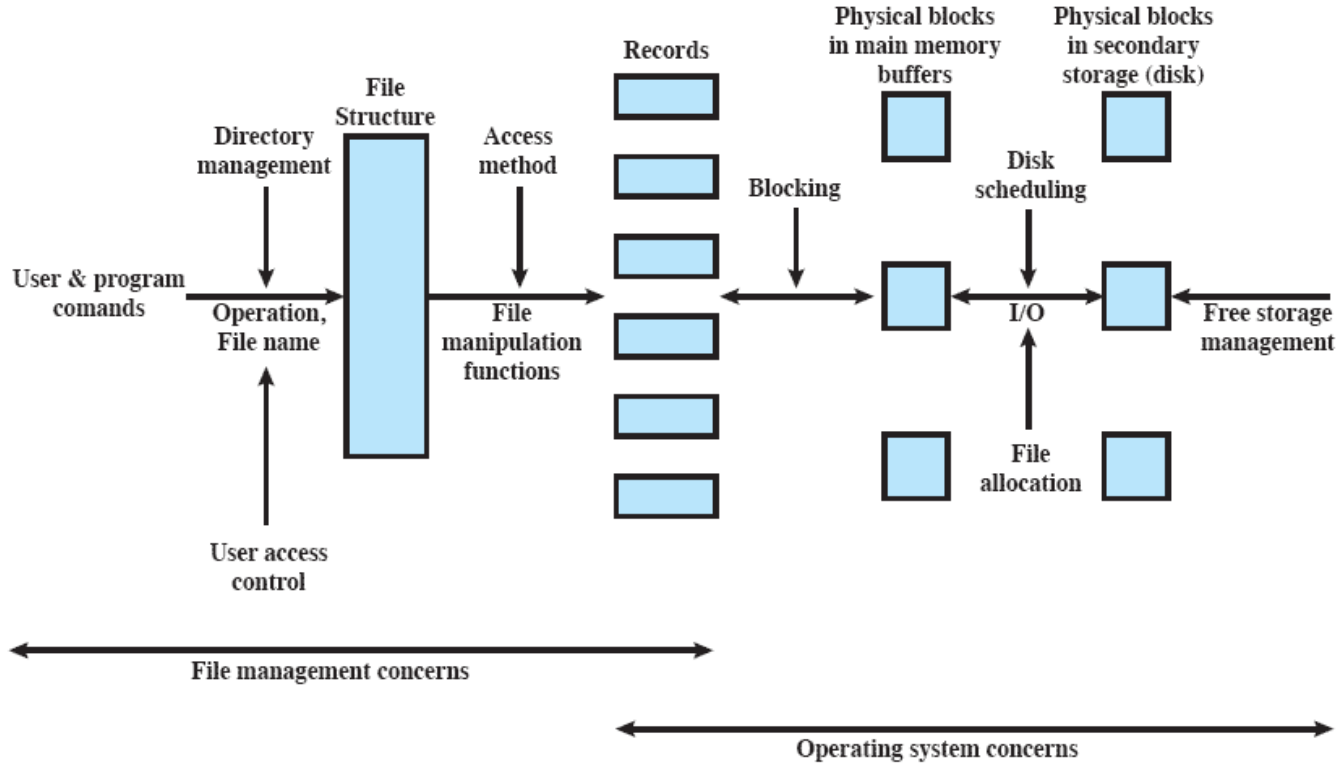


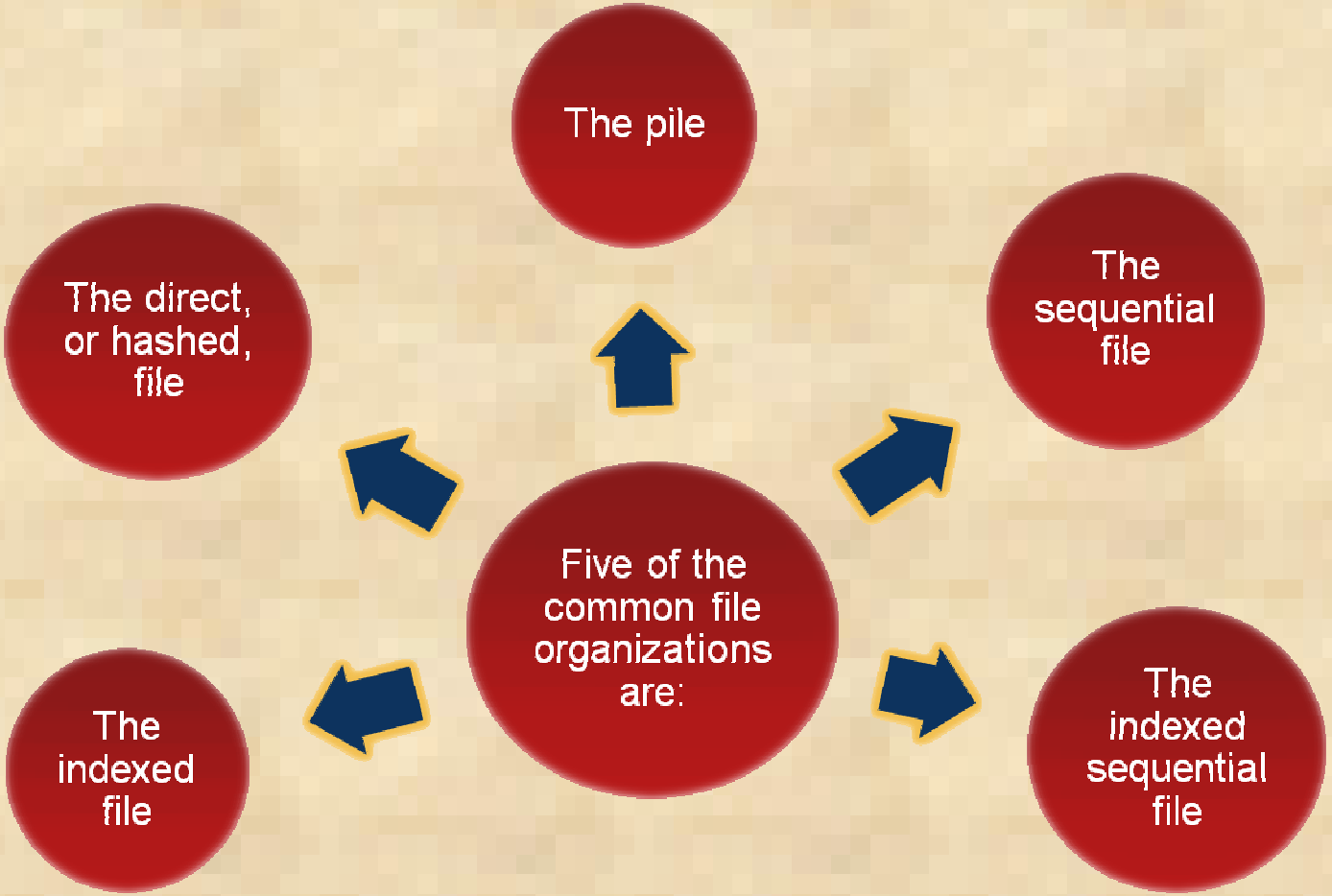
Figure 12.2 Elements of File Management

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

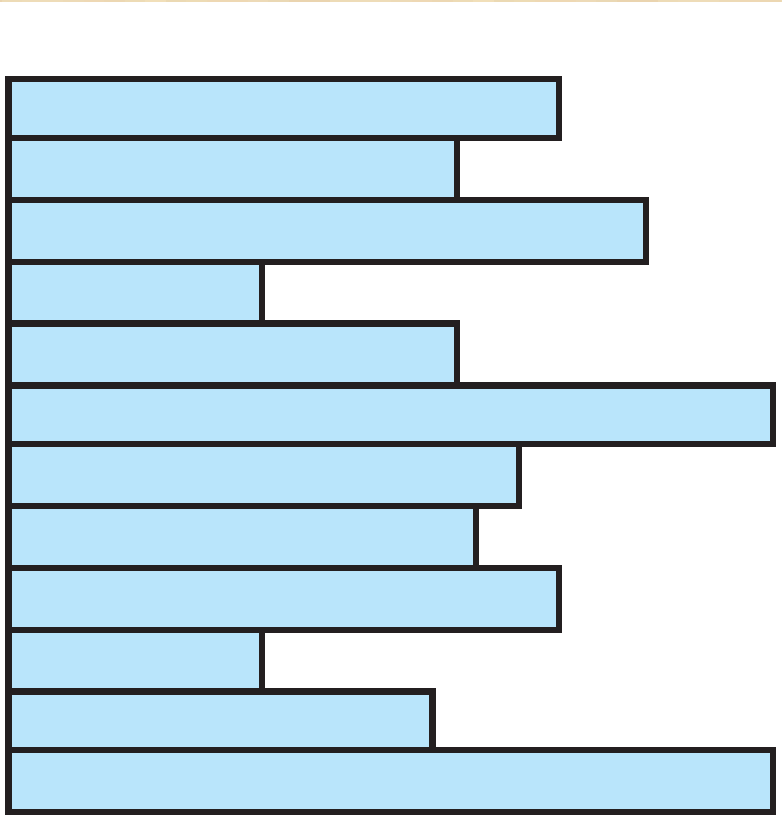
- Priority of criteria depends on the application that will use the file

File Organization Types



The Pile

- Least complicated form of file organization
- Data are collected in the order they arrive
- Each record consists of one burst of data
- Purpose is simply to accumulate the mass of data and save it
- Record access is by exhaustive search

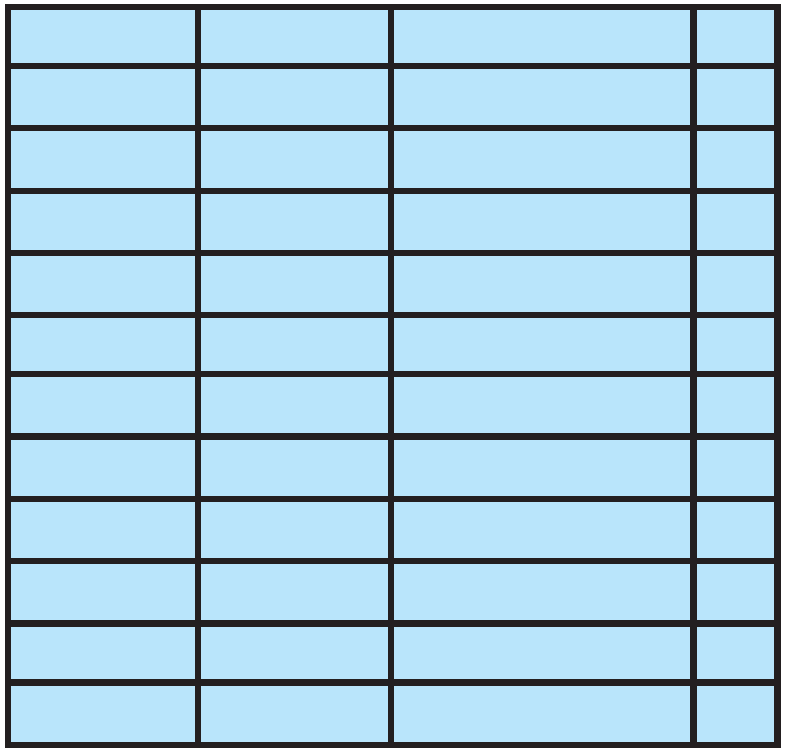


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

The Sequential File

- Most common form of file structure
- A fixed format is used for records
- Key field uniquely identifies the record
- Typically used in batch applications
- Only organization that is easily stored on tape as well as disk

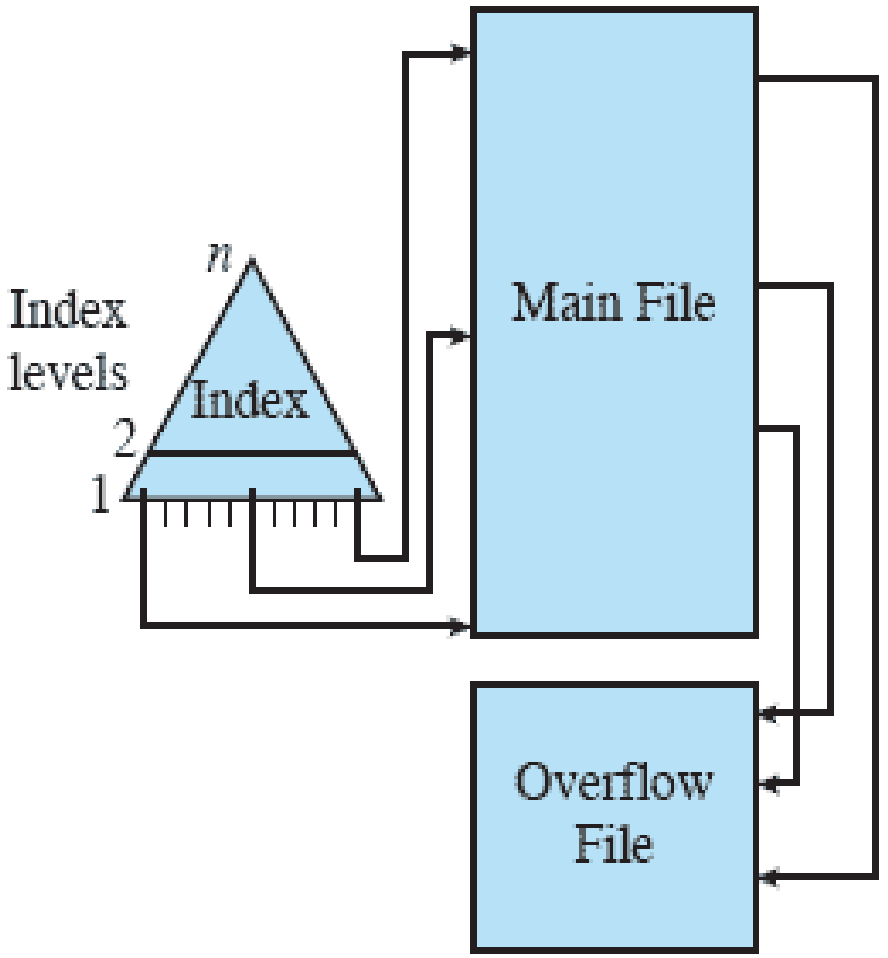


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Indexed Sequential File

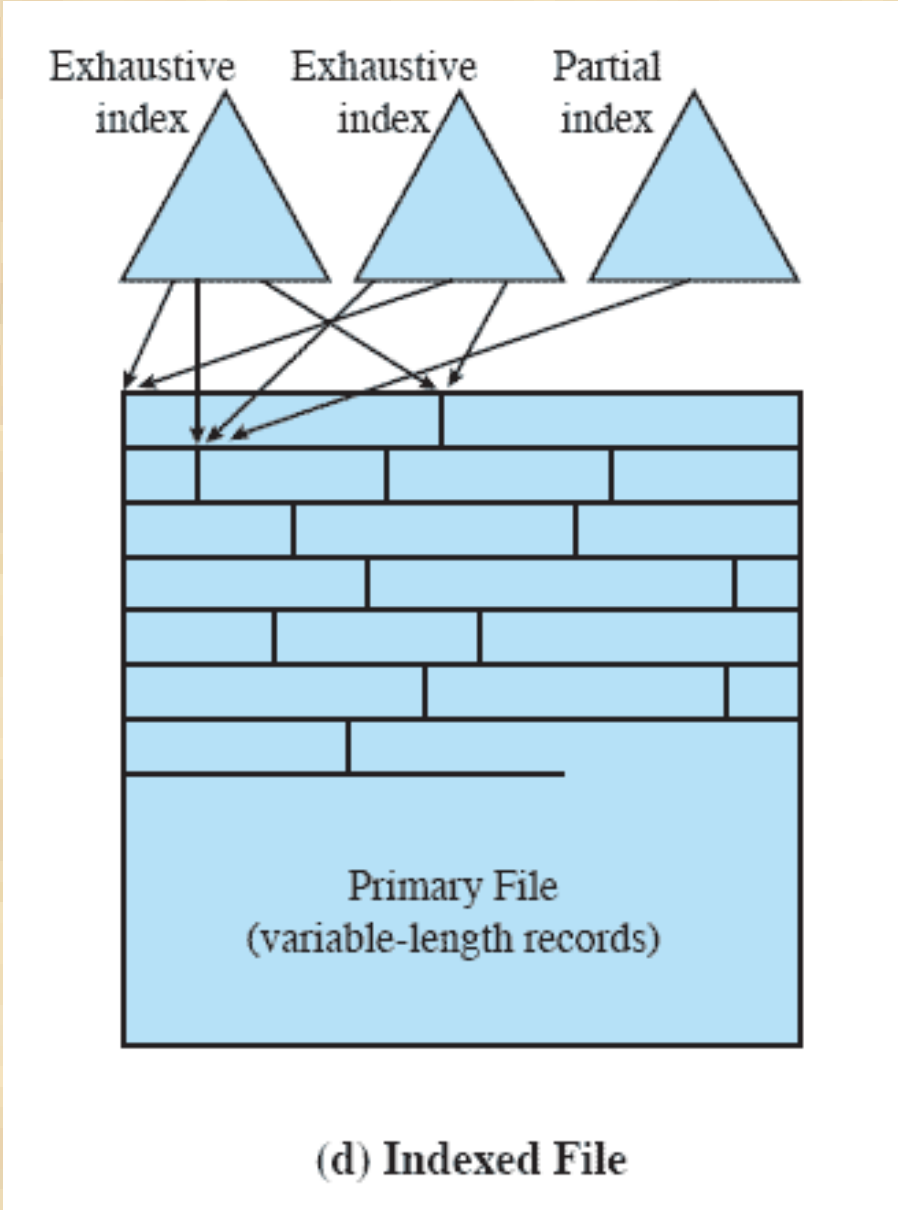
- Adds an index to the file to support random access
- Adds an overflow file
- Greatly reduces the time required to access a single record
- Multiple levels of indexing can be used to provide greater efficiency in access



(c) Indexed Sequential File

Indexed File

- Records are accessed only through their indexes
- Variable-length records can be employed
- Exhaustive index contains one entry for every record in the main file
- Partial index contains entries to records where the field of interest exists
- Used mostly in applications where timeliness of information is critical
- Examples would be airline reservation systems and inventory control systems



(d) Indexed File

Grades of Performance

Table 12.1 Grades of Performance for Five Basic File Organizations [WIED87]

	Space		Update		Retrieval		
	Attributes		Record Size				
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A	=	Excellent, well suited to this purpose	$\approx O(r)$
B	=	Good	$\approx O(o \times r)$
C	=	Adequate	$\approx O(r \log n)$
D	=	Requires some extra effort	$\approx O(n)$
E	=	Possible with extreme effort	$\approx O(r \times n)$
F	=	Not reasonable for this purpose	$\approx O(n^{>1})$

where

r = size of the result
o = number of records that overflow
n = number of records in 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

B-Trees

- A balanced tree structure with all branches of equal length
- Standard method of organizing indexes for databases
- Commonly used in OS file systems
- Provides for efficient searching, adding, and deleting of items

B-Tree Characteristics

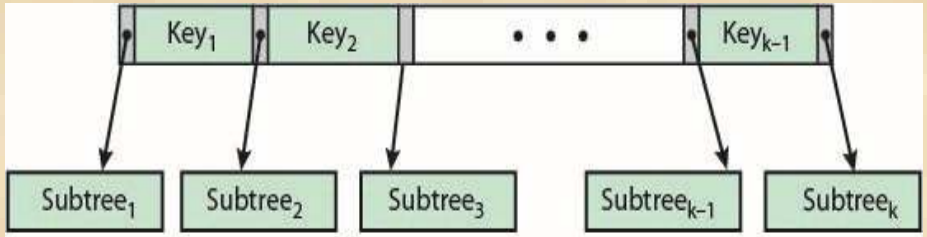


Figure 12.4 A B-tree Node with k Children

A tree structure (no closed loops) with the following characteristics:

- - the tree consists of a number of nodes and leaves
- - each node contains at least one key which uniquely identifies a file record, and more than one pointer to child nodes or leaves
- - each node is limited to the same number of maximum keys
- - the keys in a node are stored in non-decreasing order; each node has one more pointer than keys

B-Tree Characteristics

A B-tree is characterized by its minimum degree *d* and satisfies the following properties:

- every node has at most $2d - 1$ keys and $2d$ children or, equivalently, $2d$ pointers
- every node, except for the root, has at least $d - 1$ keys and d pointers, as a result, each internal node, except the root, is at least half full and has at least d children
- the root has at least 1 key and 2 children
- all leaves appear on the same level and contain no information. This is a logical construct to terminate the tree; the actual implementation may differ.
- a nonleaf node with k pointers contains $k - 1$ keys

Inserting Nodes Into a B-Tree

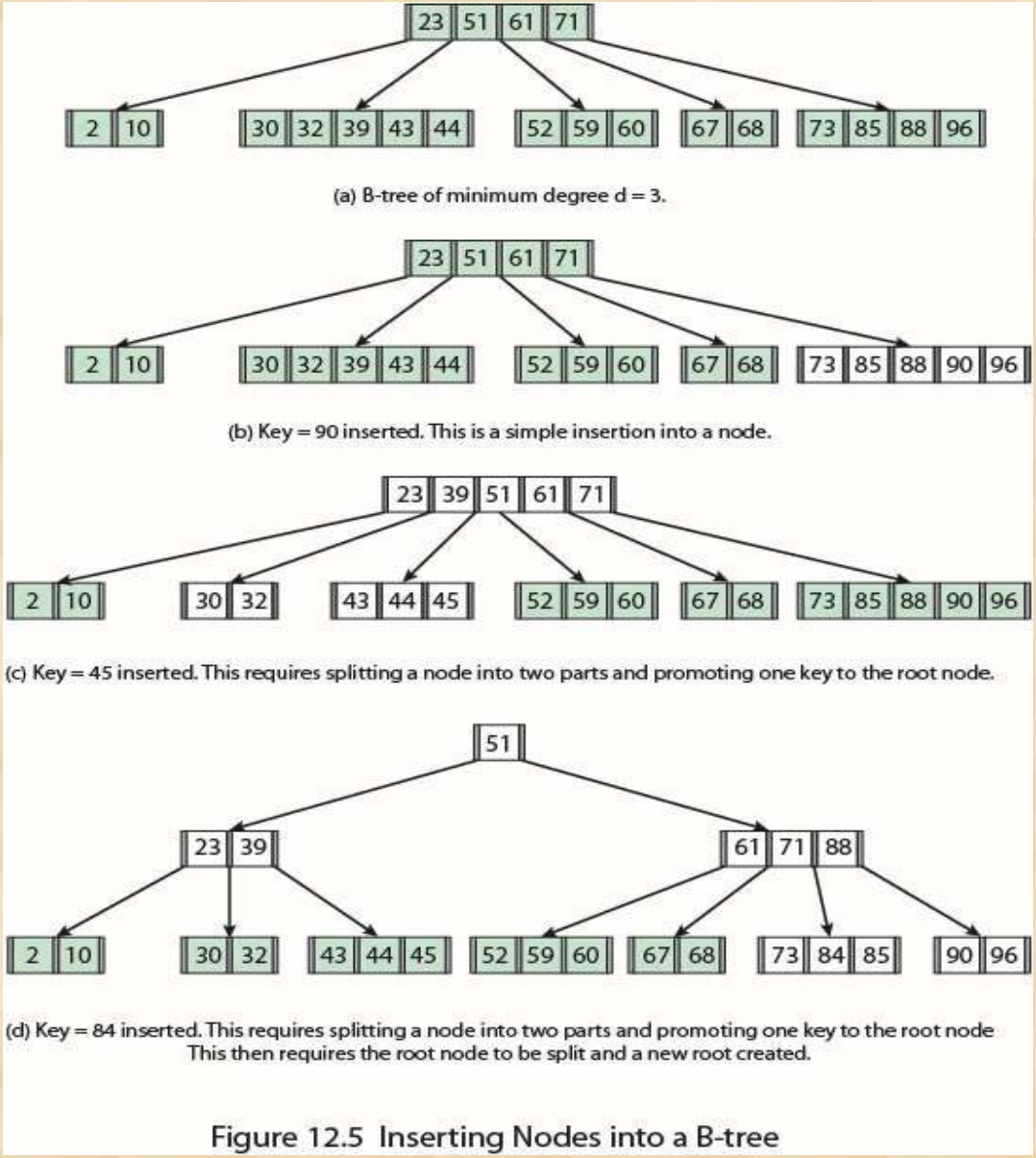
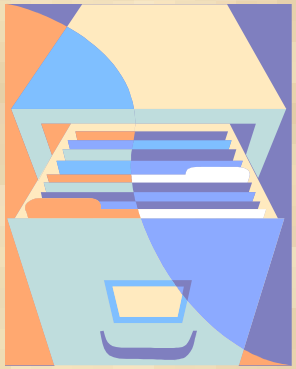


Figure 12.5 Inserting Nodes into a B-tree

Table 12.2 Information Elements of a File Directory

File Directory Information



Operations Performed on a Directory

- 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

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

File system can easily enforce access restriction on directories

Figure 12.4

Tree-Structured Directory

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries

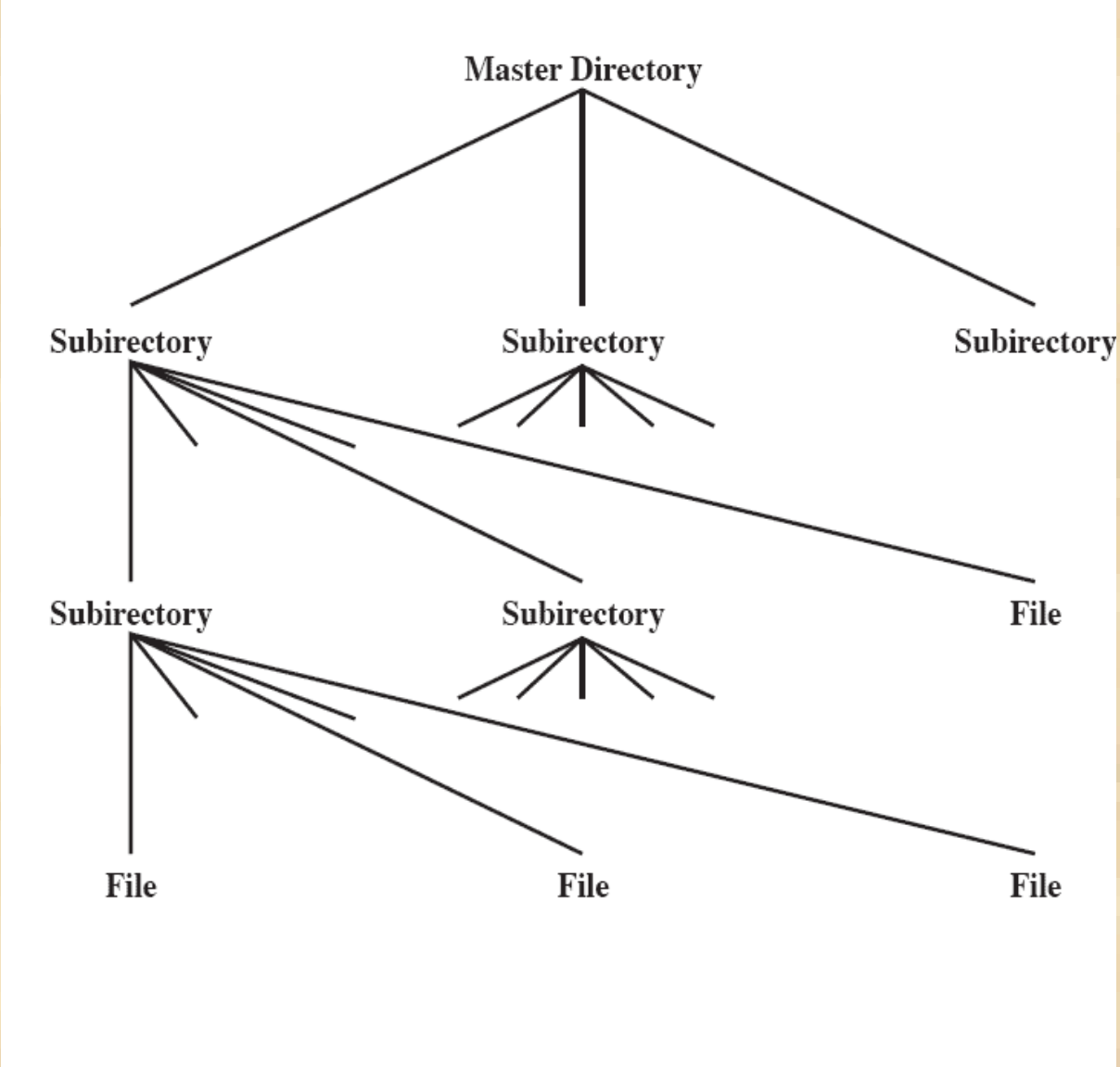
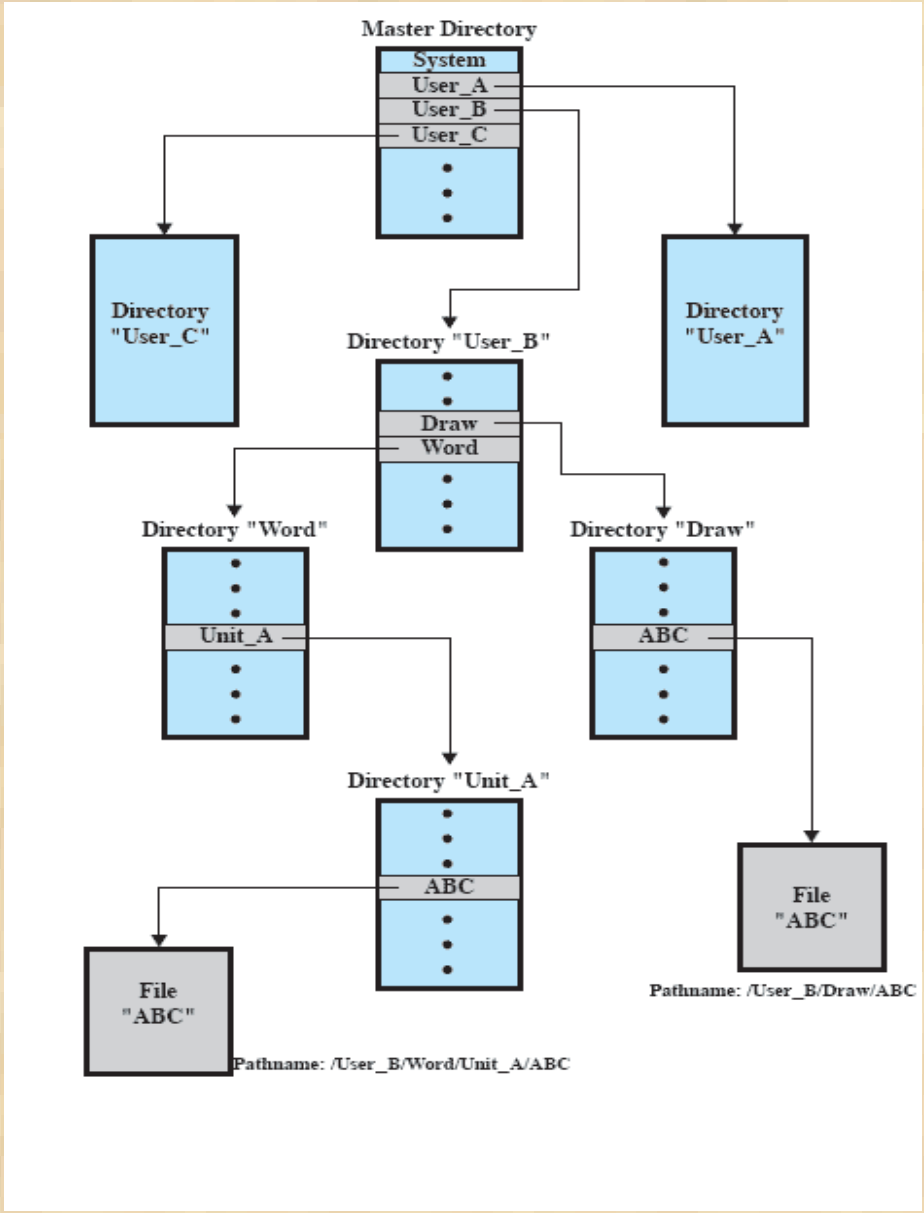


Figure 12.7 Example of Tree-Structured Directory



File Sharing

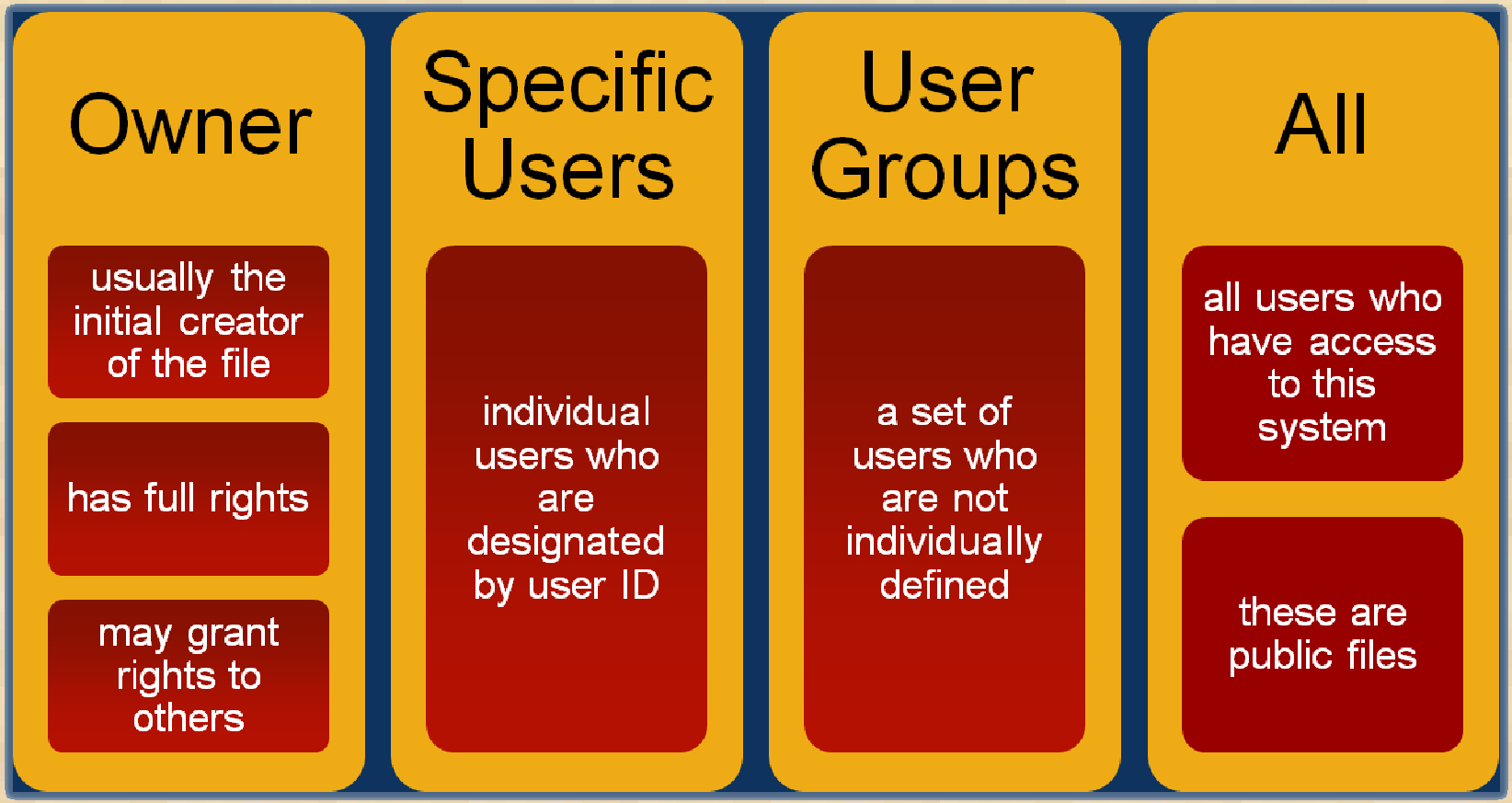
Two issues arise when allowing files to be shared among a number of users:

- access rights
- management of simultaneous access

Access Rights

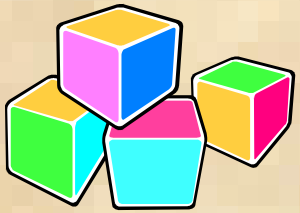
- ***None***
  - the user would not be allowed to read the user directory that includes the file
- ***Knowledge***
  - the user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
- ***Execution***
  - the user can load and execute a program 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



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:

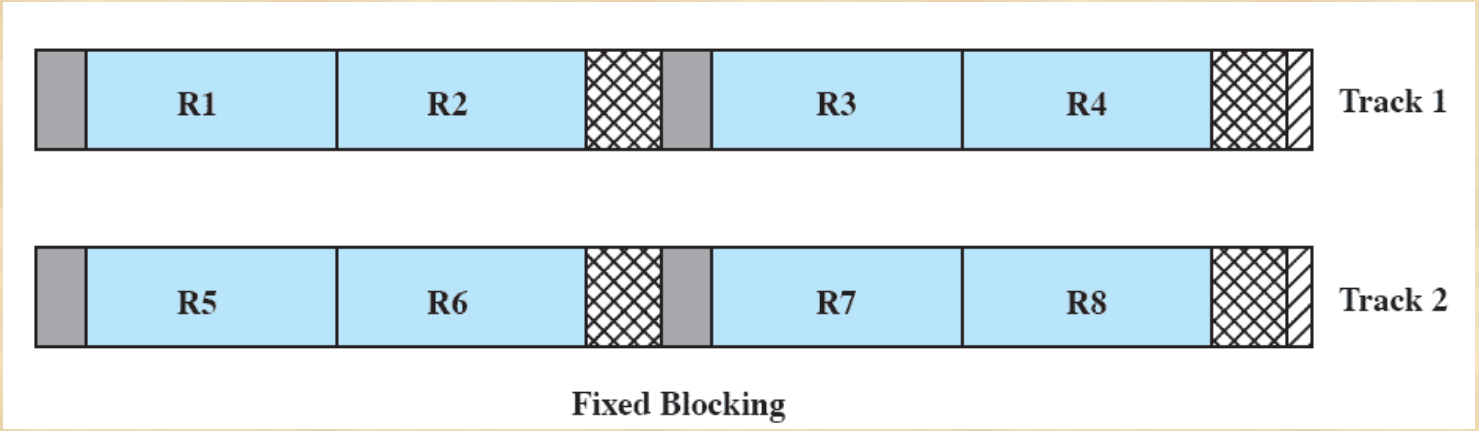
1) **Fixed-Length Blocking** – fixed-length records are used, and an integral number of records are stored in a block

   Internal fragmentation – unused space at the end of each block

2) **Variable-Length Spanned Blocking –** variable-length records are used and are packed into blocks with no unused space

3) **Variable-Length Unspanned Blocking** – variable-length records are used, but spanning is not employed

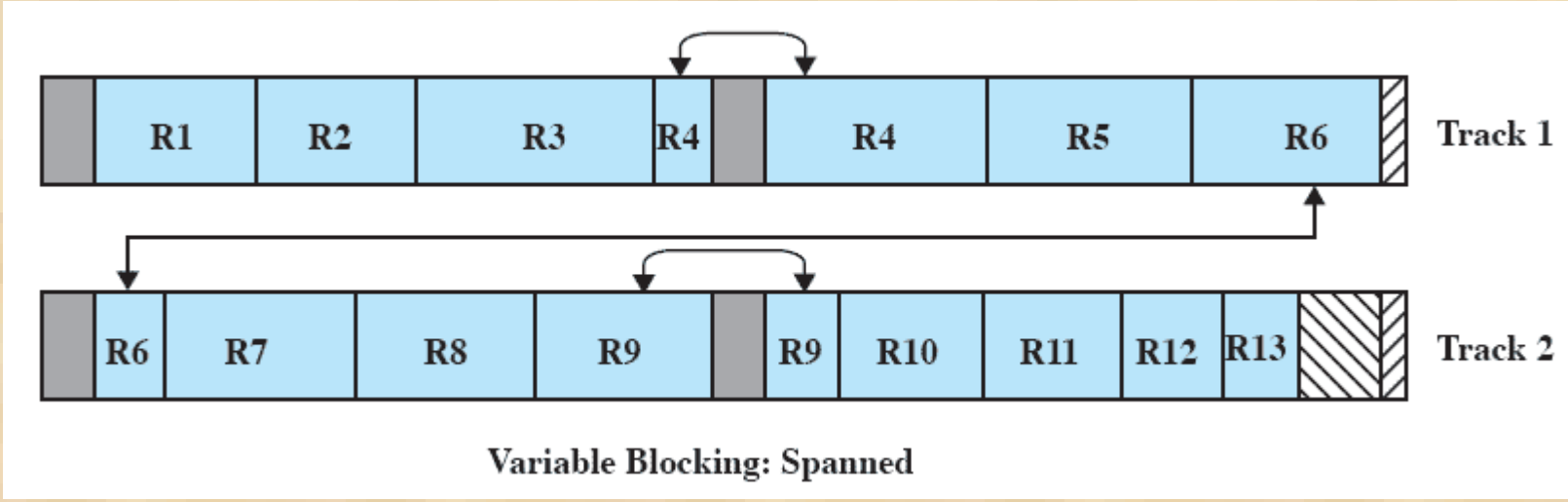
Fixed Blocking



Fixed Blocking

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

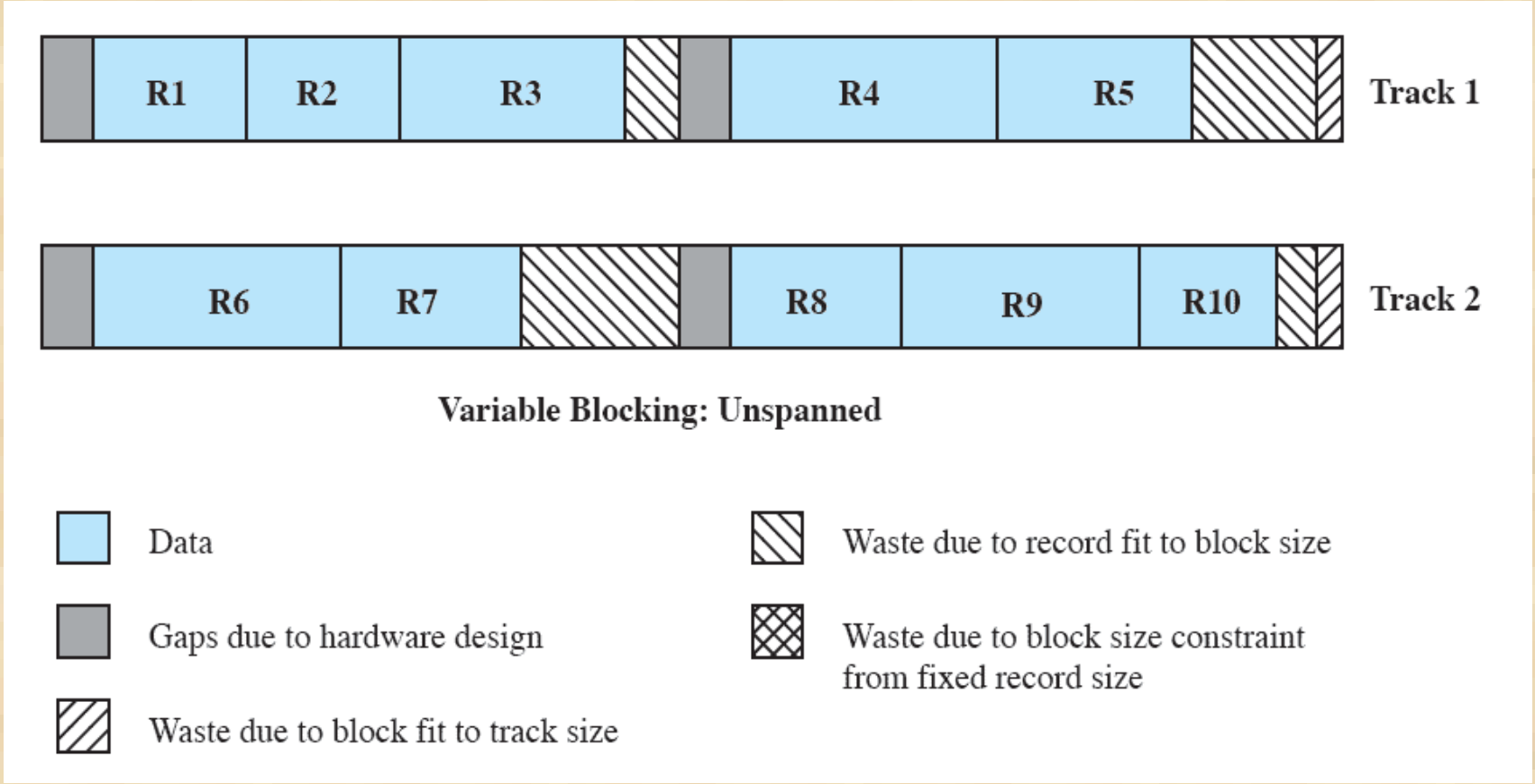
Variable Blocking: Spanned



Variable Blocking: Spanned

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

Variable Blocking: Unspanned

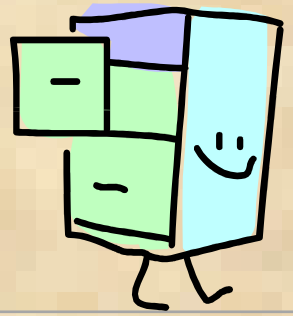


File Allocation

- 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*** (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 size
  - Dynamic allocation allocates space to a file in portions as needed



Portion Size

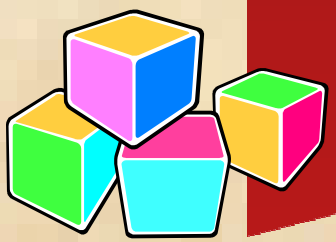
- In choosing a portion size there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency
- Items to be considered:
  1) contiguity of space increases performance, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented 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



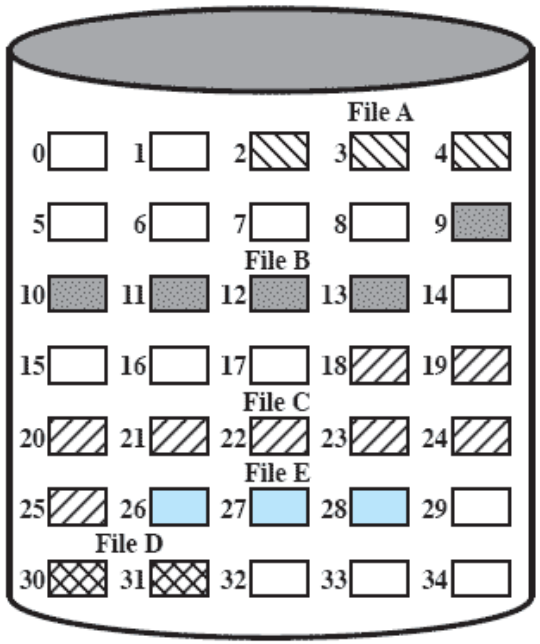
Blocks

- 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

Table 12.3
File Allocation Methods

Contiguous File Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation
- Preallocation strategy using variable-size portions
- Is the best from the point of view of the individual sequential file

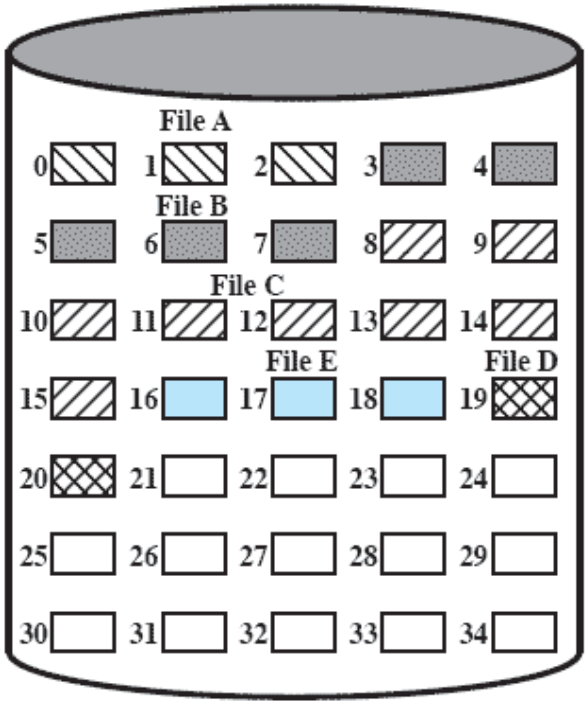


File Allocation Table

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

After Compaction



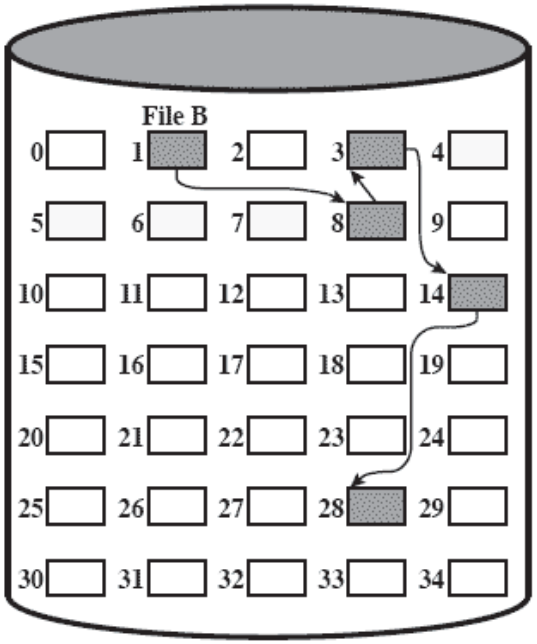
File Allocation Table

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)

Chained Allocation

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- The file allocation table needs just a single entry for each file
- No external fragmentation to worry about
- Best for sequential files

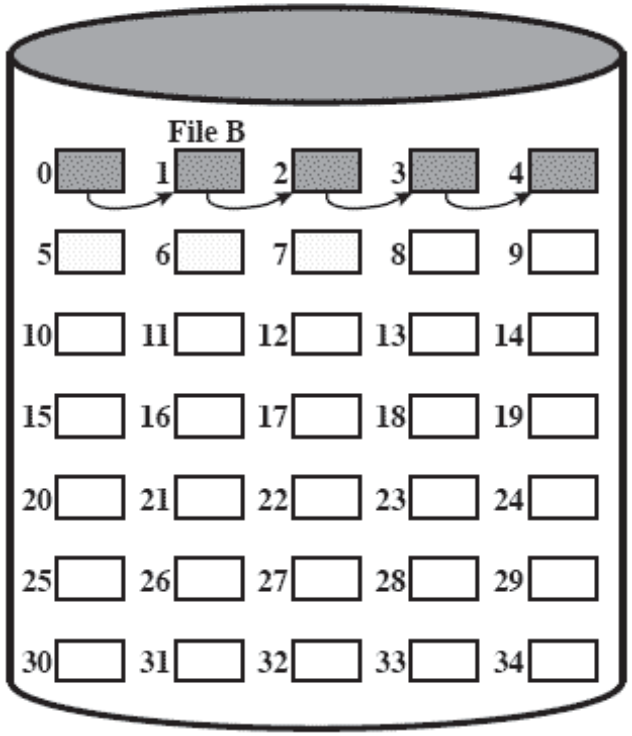


File Allocation Table

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

Figure 12.11 Chained Allocation

Chained Allocation After Consolidation



File Allocation Table

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

Figure 12.12 Chained Allocation (After Consolidation)

Indexed Allocation with Block Portions

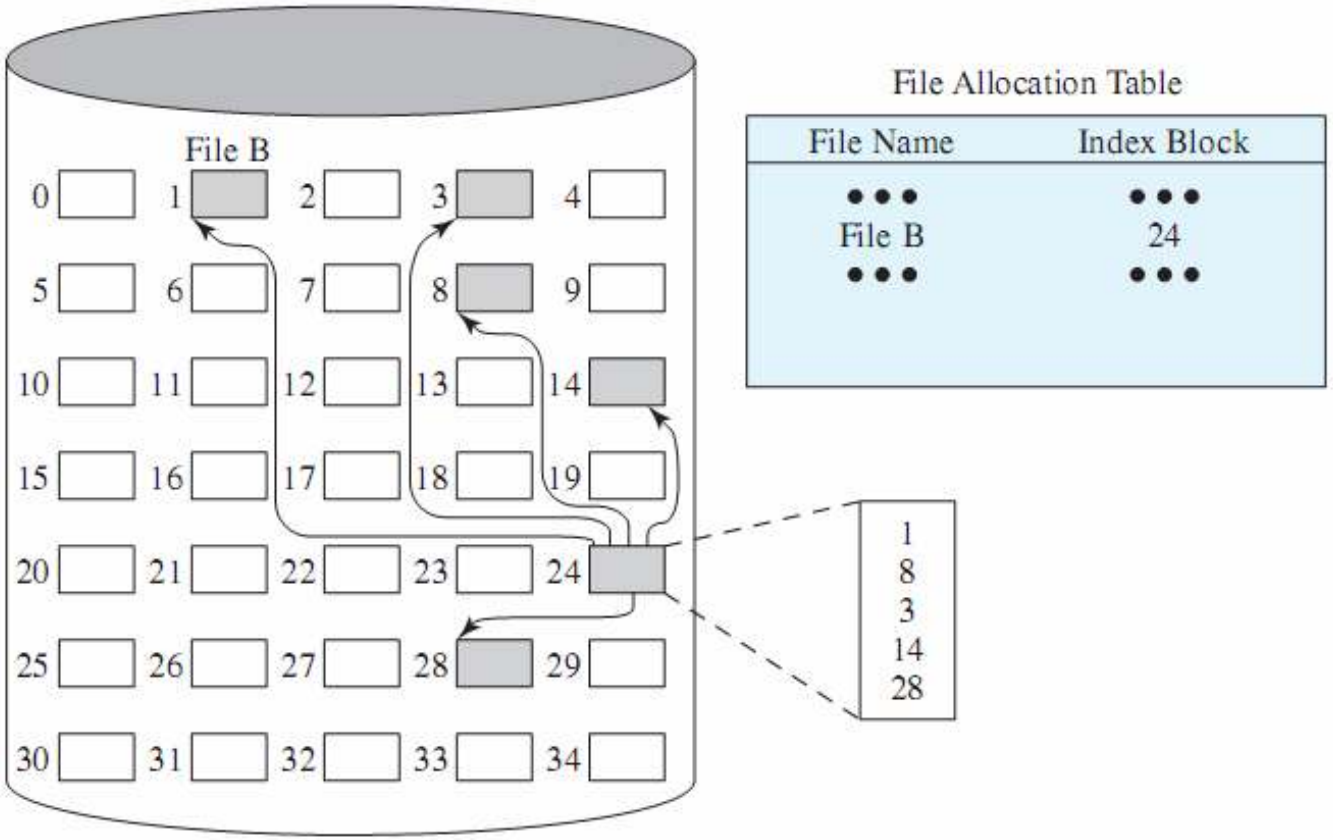


Figure 12.13 **Indexed Allocation with Block Portions**

Indexed Allocation with Variable Length Portions

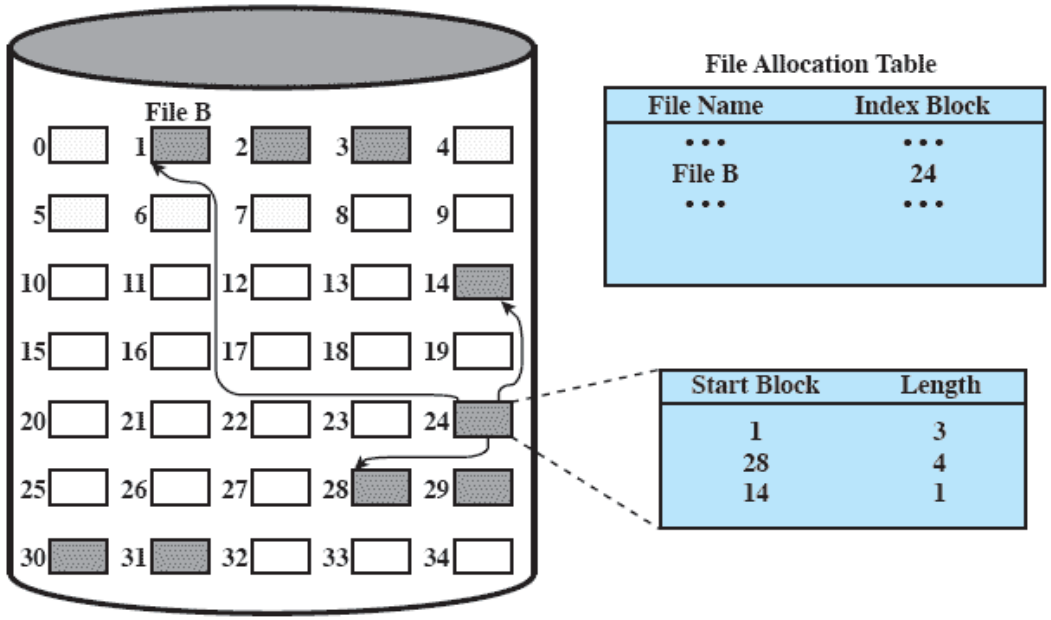
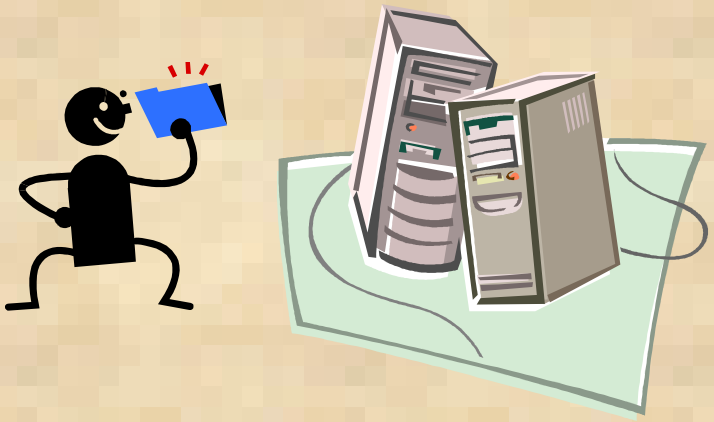


Figure 12.14 Indexed Allocation with Variable-Length Portions

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

- This method 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

Advantages:

- works well with any file allocation method
- it is as small as possible

Chained Free Portions

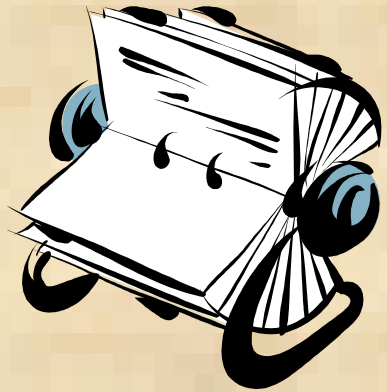
- The free portions may be chained together by using a pointer and length value in each free portion
- Negligible space overhead because there is no need for a disk allocation table
- Suited to all file allocation methods

Disadvantages:

- leads to fragmentation
- 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

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
- 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

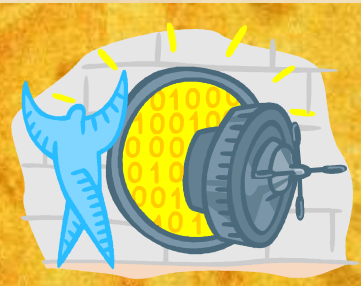
Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number

- the size of the free block list is 24 or 32 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

Access Matrix

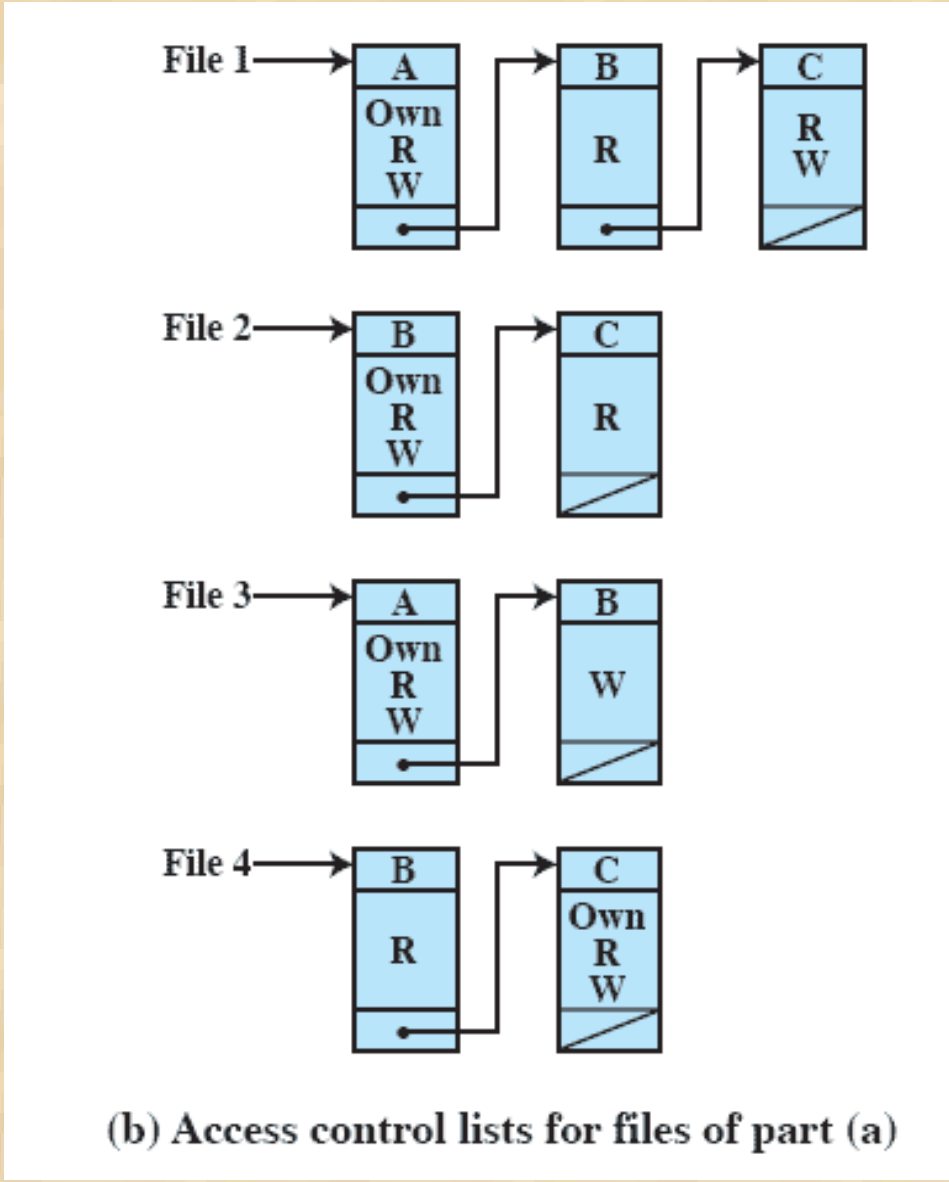
- The basic elements are:
  - **subject** – an entity capable of accessing objects
  - **object** – anything to which access is controlled
  - **access right** – the way in which an object is accessed by a subject

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

(a) Access matrix

Access Control Lists

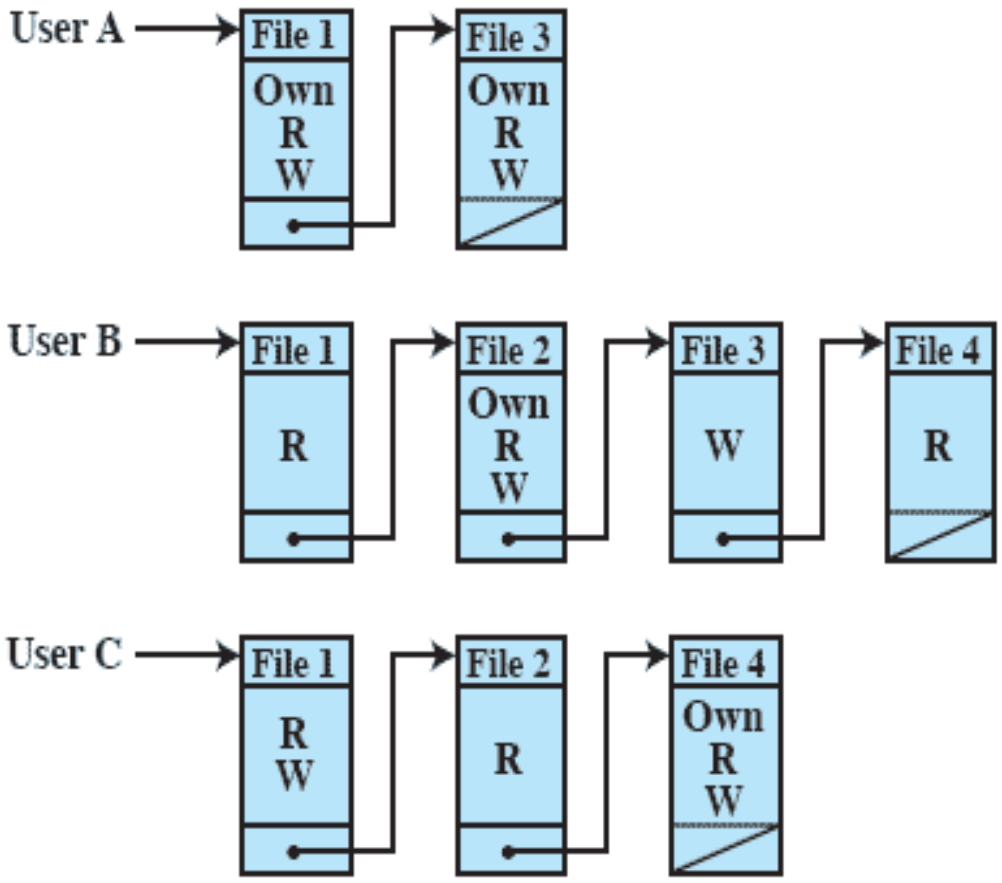
- A matrix may be decomposed by columns, yielding **access control lists**
- The access control list lists users and their permitted access rights



(b) Access control lists for files of part (a)

Capability Lists

- Decomposition by rows yields **capability tickets**
- A **capability ticket** specifies authorized objects and operations for a user



(c) Capability lists for files of part (a)

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

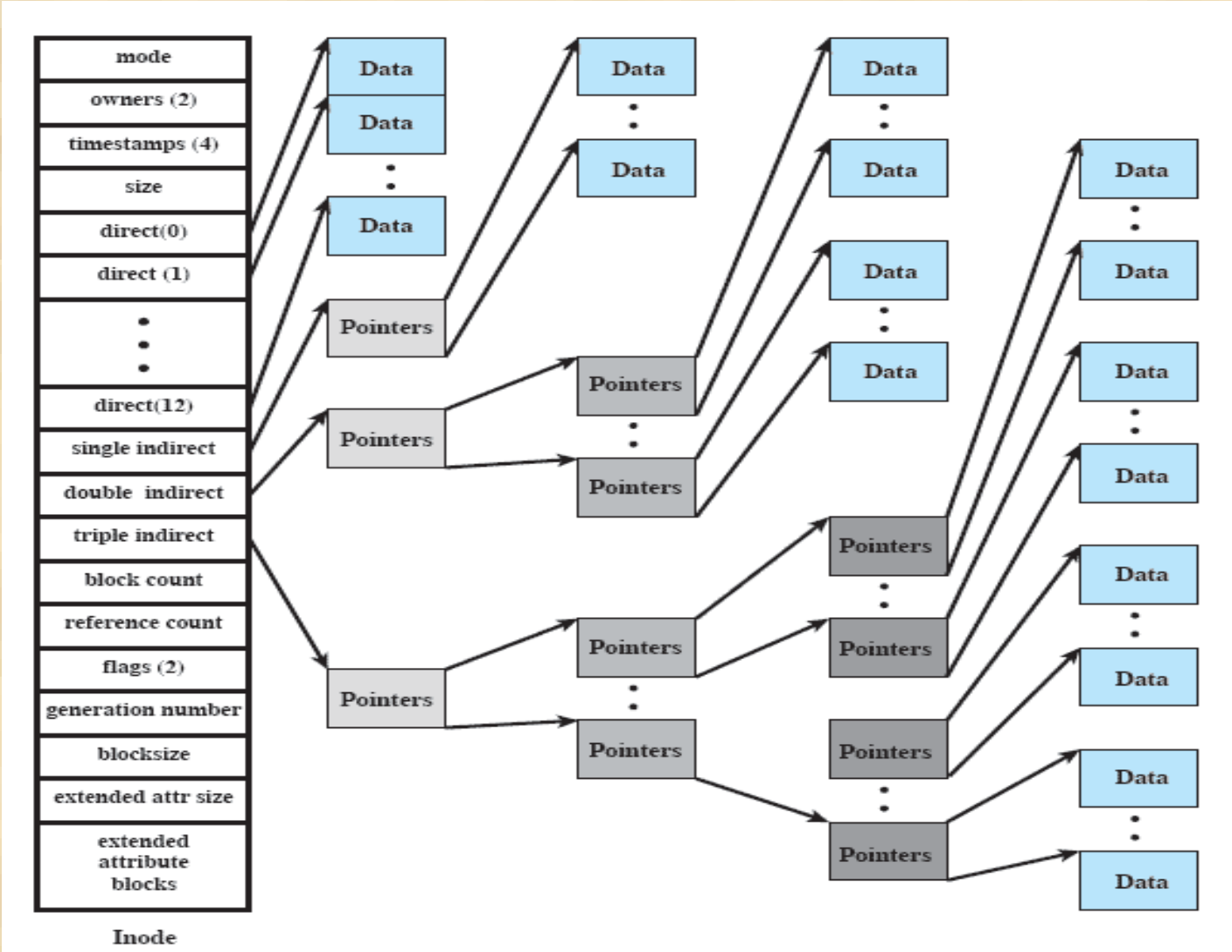
Symbolic links

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

Inodes

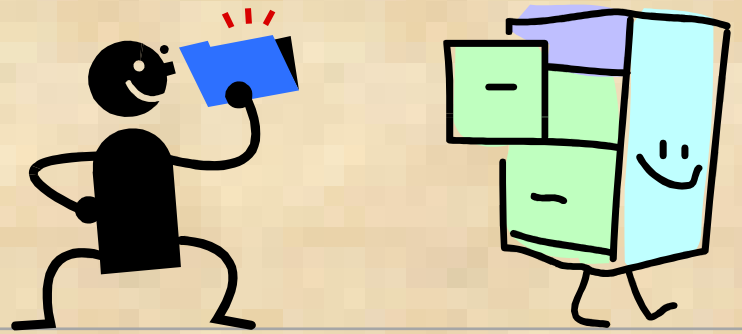
- All types of UNIX files are administered by the OS by means of inodes
- An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file
- Several file names may be associated with a single inode
  - an active inode is associated with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

FreeBSD Inode and File Structure



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)



Capacity of a FreeBSD File with 4 Kbyte Block Size

Table 12.4

UNIX Directories and Inodes

- Directories are structured in a hierarchical tree
- Each directory can contain files and/or other directories
- A directory that is inside another directory is referred to as a subdirectory

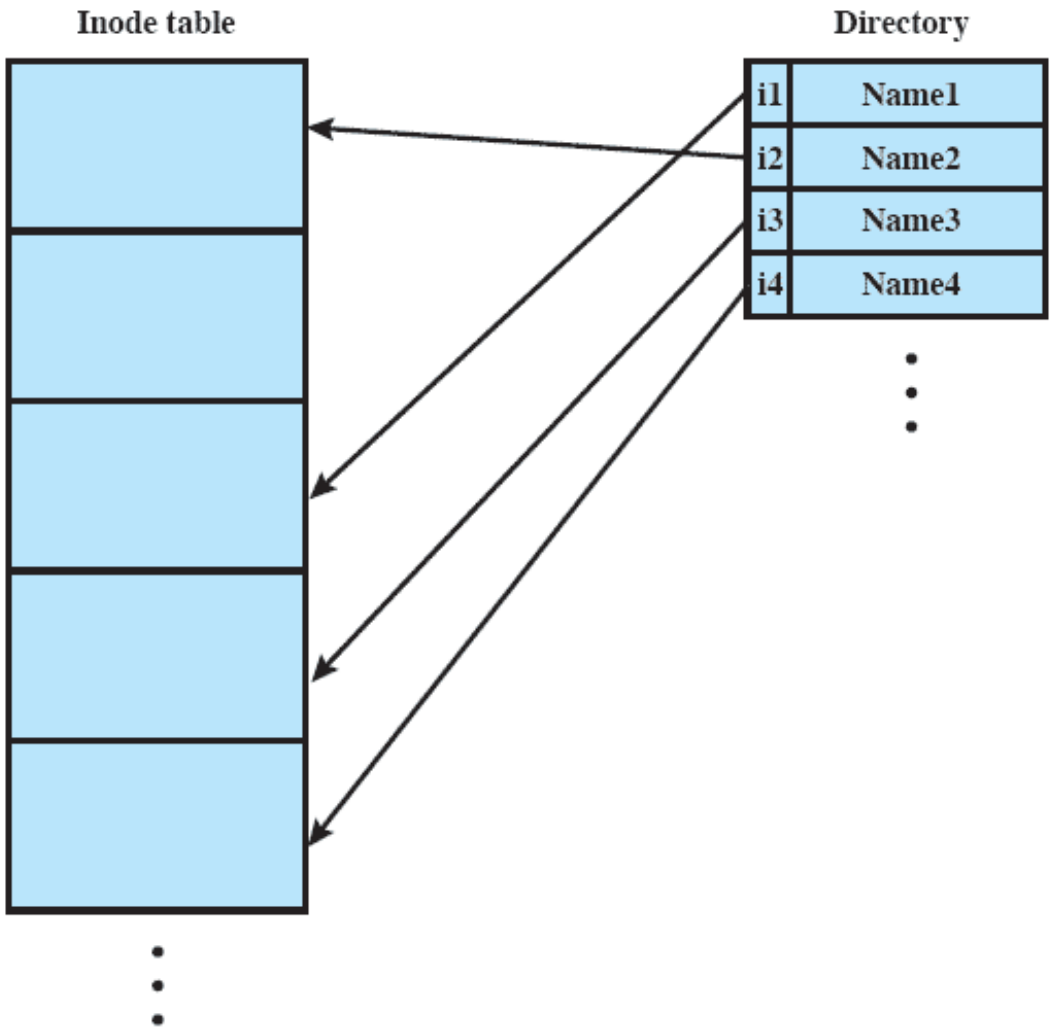
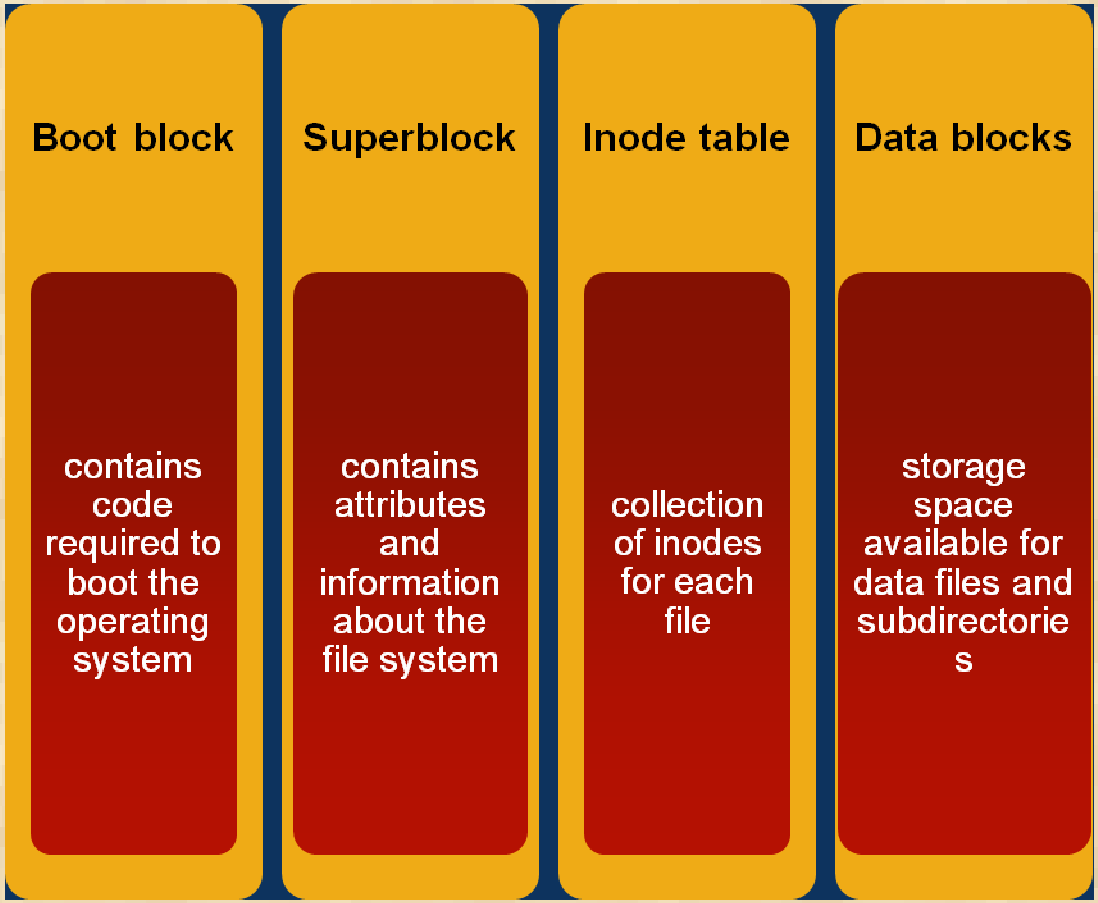


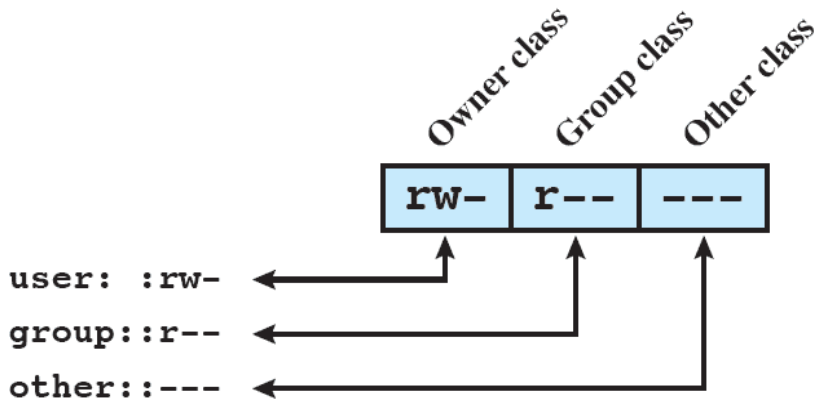
Figure 12.17 UNIX Directories and Inodes

Volume Structure

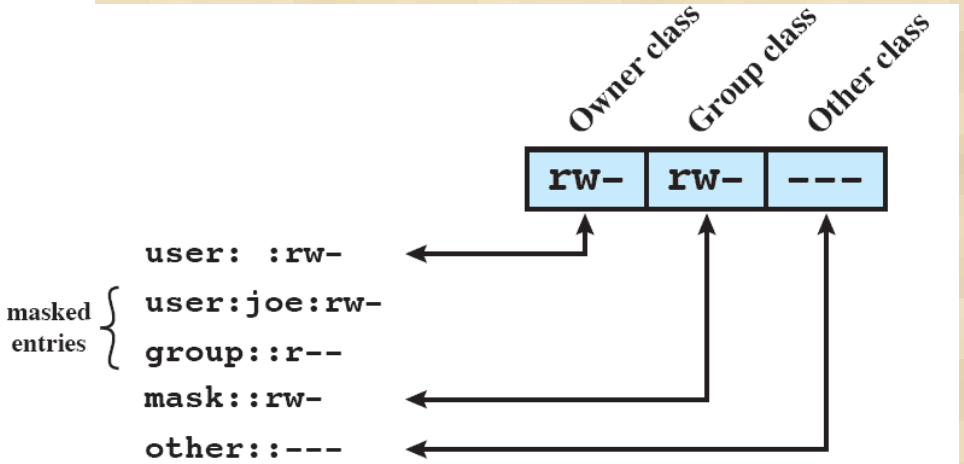
- A UNIX file system resides on a single logical disk or disk partition and is laid out with the following elements:



UNIX File Access Control



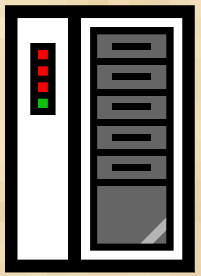
(a) Traditional UNIX approach (minimal access control list)

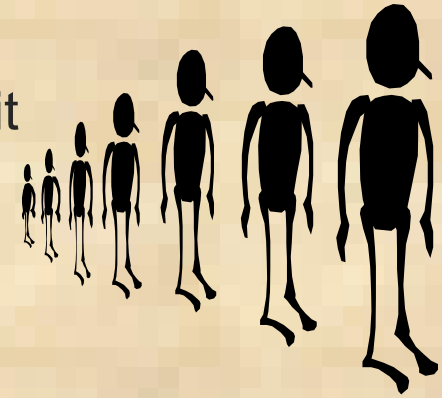


(b) Extended access control list

Access Control Lists in UNIX

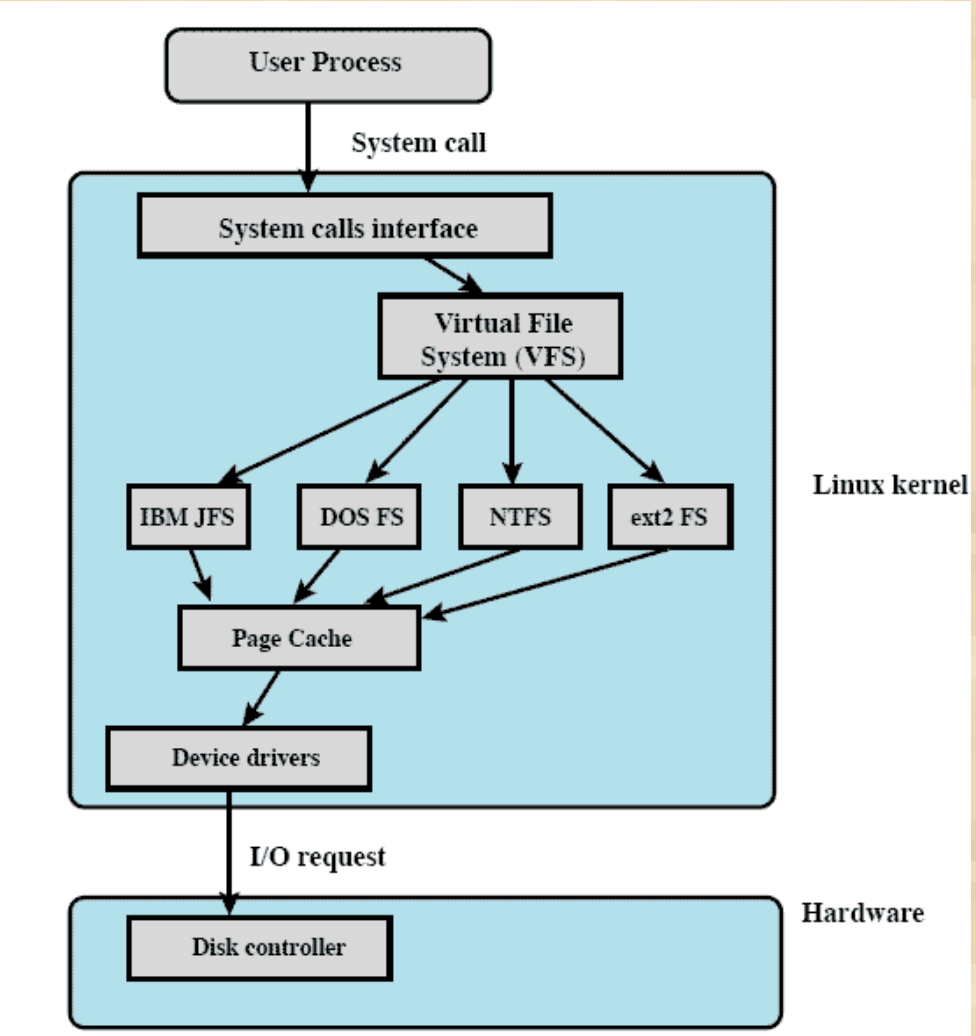
- FreeBSD allows the administrator to assign a list of UNIX user IDs and groups to a file
- Any number of users and groups can be associated with a file, each with three protection bits (read, write, execute)
- A file may be protected solely by the traditional UNIX file access mechanism
- FreeBSD files include an additional protection bit that indicates whether the file has an extended ACL





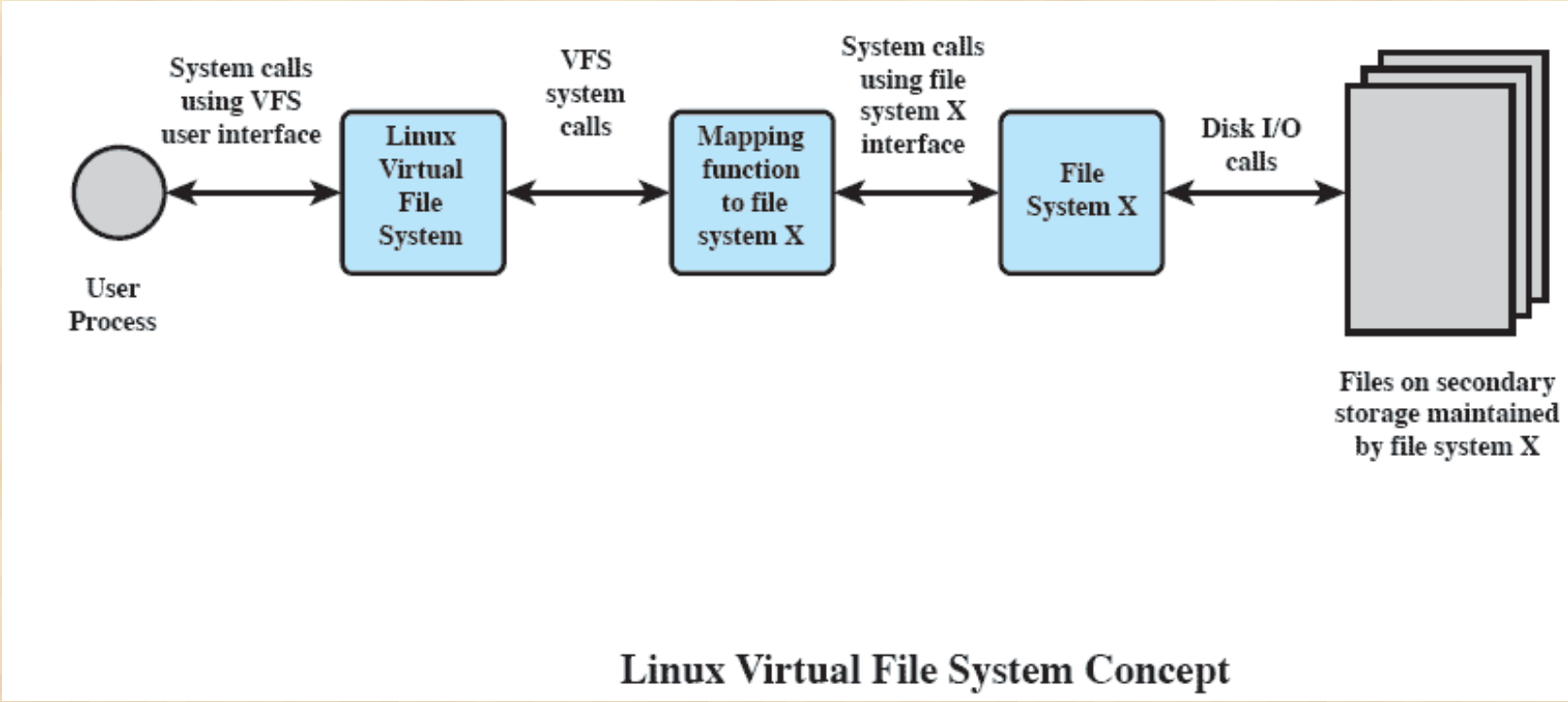
Linux Virtual File System (VFS)

- Presents a single, uniform file system interface to user processes
- Defines a common file model that is capable of representing any conceivable file system's general feature and behavior
- Assumes files are objects that share basic properties regardless of the target file system or the underlying processor hardware



Linux Virtual File System Context

The Role of VFS Within the Kernel



Linux Virtual File System Concept

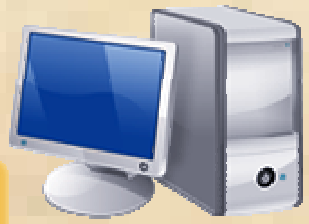
Primary Object Types in VFS

Superblock Object

- represents a specific mounted file system

Dentry Object

- represents a specific directory entry



Inode Object

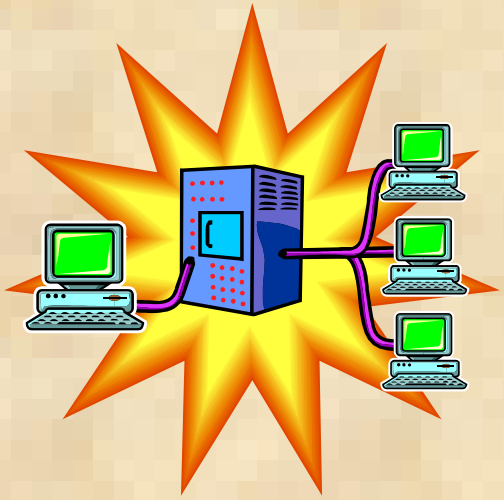
- represents a specific file

File Object

- represents an open file associated with a process

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
  - security
  - large disks and large files
  - multiple data streams
  - journaling
  - compression and encryption
  - hard and symbolic links



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

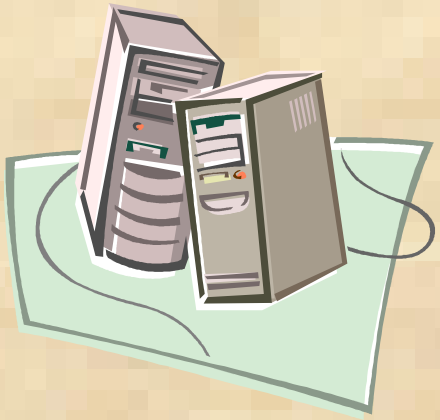
- one or more contiguous sectors
- the cluster size in sectors is a power of 2

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 264 bytes

Table 12.5
Windows NTFS Partition and Cluster Sizes

NTFS Volume Layout



- Every element on a volume is a file, and every file consists of a collection of attributes
  - even the data contents of a file is treated as an attribute

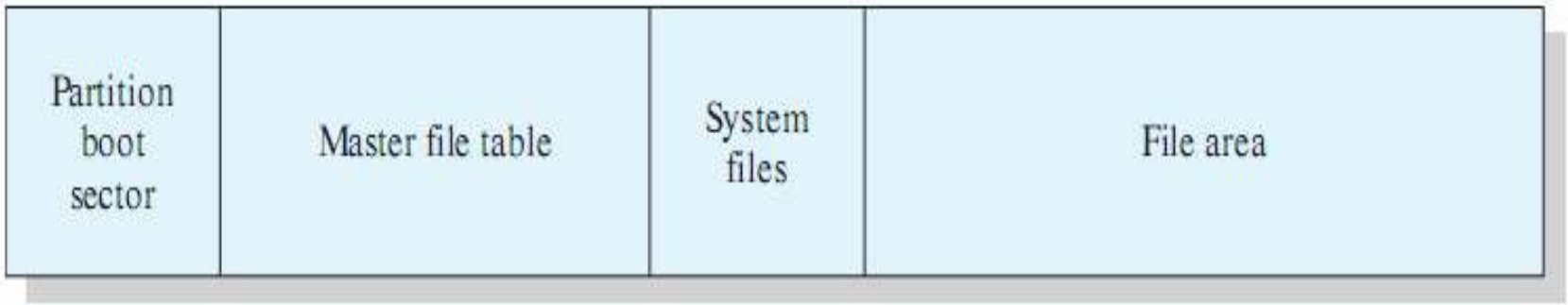


Figure 12.21 **NTFS Volume Layout**

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 12.6

Windows NTFS Components

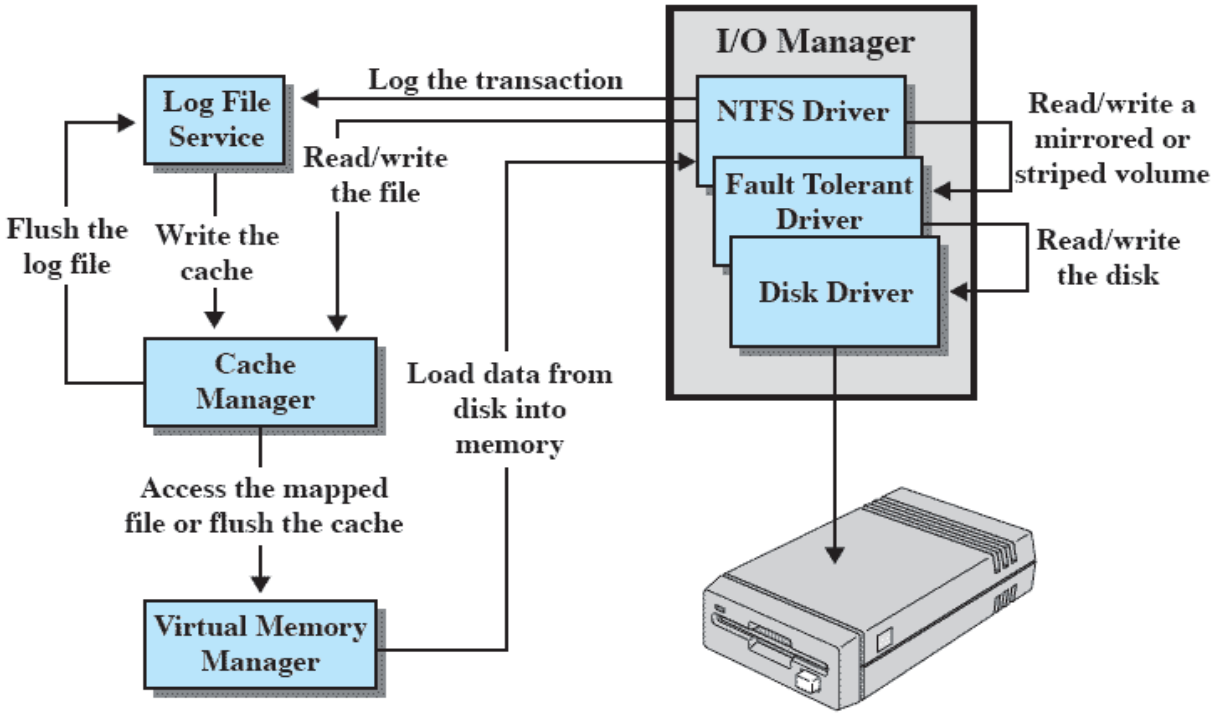
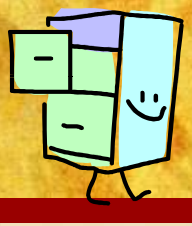


Figure 12.22 Windows NTFS Components



Summary

- **A file management system:**
  - is a set of system software that provides services to users and applications in the use of files
  - is typically viewed as a system service that is served by the operating system
- **Files:**
  - consist of a collection of records
  - if a file is primarily to be processed as a whole, a sequential file organization is the simplest and most appropriate
  - if sequential access is needed but random access to individual file is also desired, an indexed sequential file may give the best performance
  - if access to the file is principally at random, then an indexed file or hashed file may be the most appropriate
  - directory service allows files to be organized in a hierarchical fashion
- **Some sort of blocking strategy is needed**
- **Key function of file management scheme is the management of disk space**
  - strategy for allocating disk blocks to a file
  - maintaining a disk allocation table indicating which blocks are free