

*Reference
Book:-
Operating
Systems:
Internals
and Design
Principles*
**By William
Stallings**

Chapter 12 File Management

Files

- Data collections created by users
- The File System is one of the most important parts of the OS to a user

Files

- 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

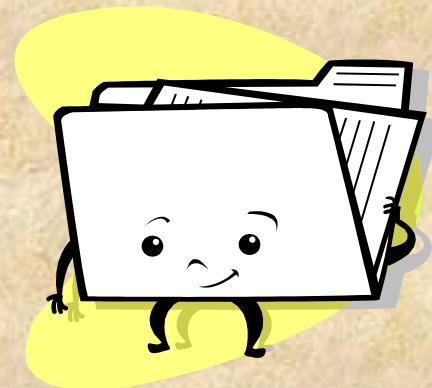
- files have names and can have associated access permissions that permit controlled sharing

Structure

- Depending upon the File System, A file can have an internal structure
- Files can be organized into hierarchical or more complex structure to reflect the relationships among files

Operation of Files

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



File Structure

Four terms are commonly used when discussing files:

Field

Record

File

Database

File Structure

- Files can be structured as a collection of records or as a sequence of bytes
- UNIX, Linux, Windows, Mac OS's consider files as a sequence of bytes
- Other OS's, notably many IBM mainframes, adopt the collection-of-records approach; useful for DB

Structure Terms

Field

- basic element of data
- contains a single value
- Eg- Employee's Last name, date of Birth
- characterized by its length and data type (e.g., ASCII string, decimal).
- fixed or variable length

Structure Terms

Record

- Collection of related fields that can be treated as a unit by some application program
- One field of the record is the **key** – a unique identifier
- For example,
- An employee record would contain such fields-as name, social security number, job classification, date of hire, and so on.
- Depending on design, records may be of fixed length or variable length.

Structure Terms

File

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

Structure Terms

Database

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

File Management System Objectives

- A file management system is that set of system software that provides services to users and applications in the use of files.
- Typically, the only way that a user or application may access files is through the file management system.

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

Typical File System Software Organization

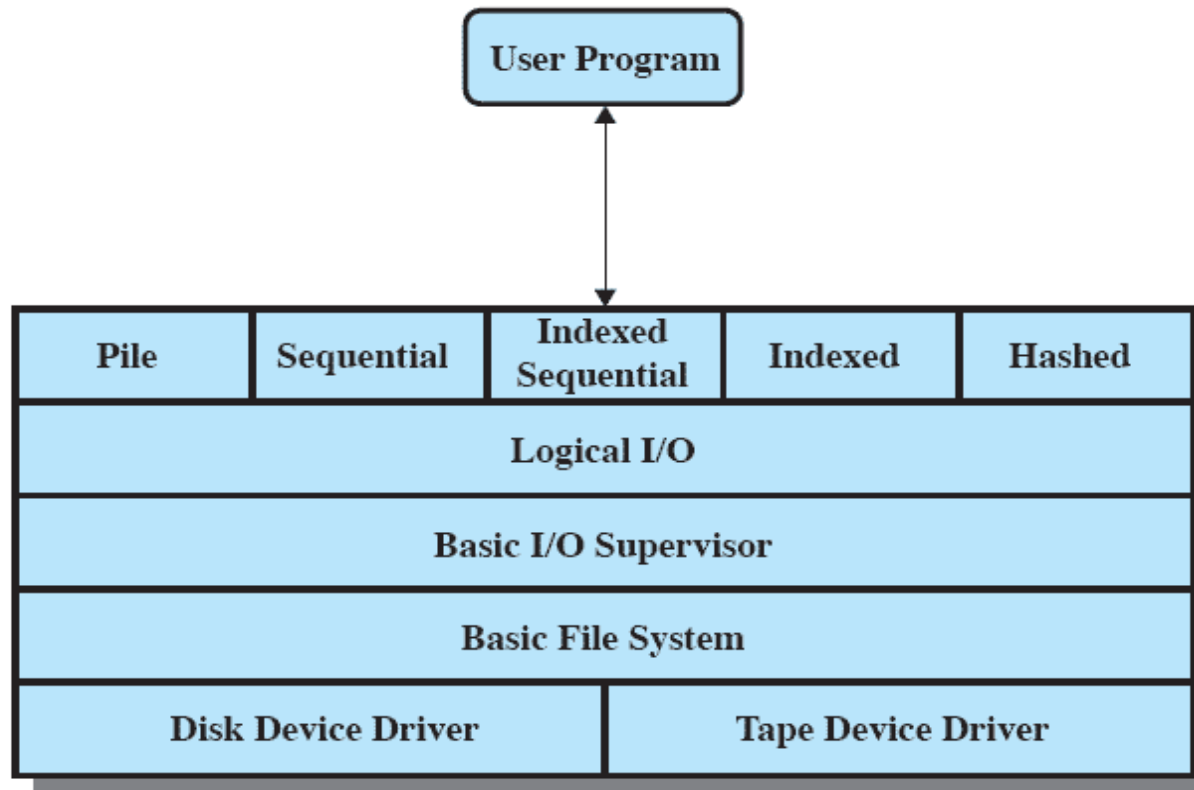


Figure 12.1 File System Software Architecture

File System Architecture

- File Formats – Access methods provide the interface to users
- Logical I/O
- Basic I/O
- Basic file system
- Device drivers

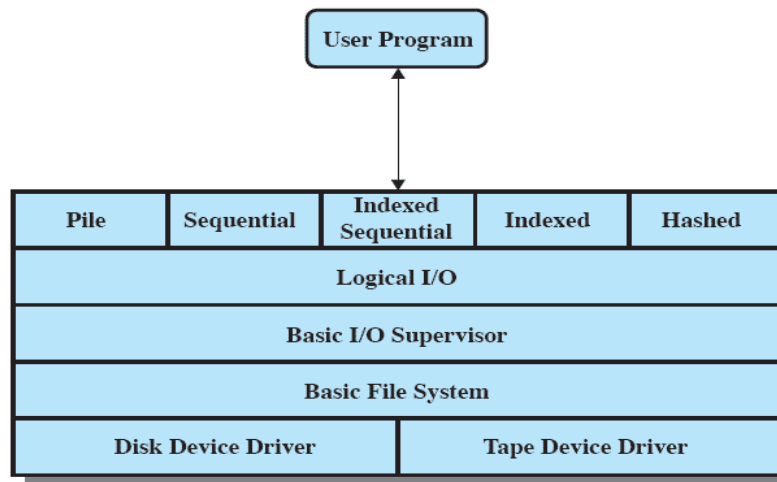
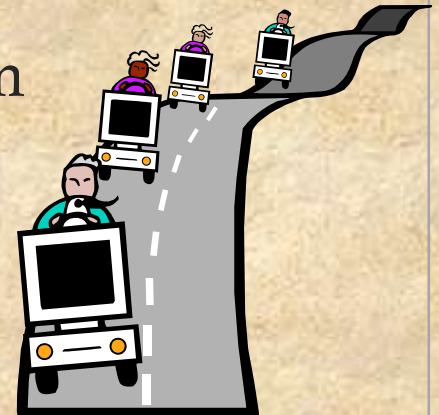


Figure 12.1 File System Software Architecture

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 other mass storage devices.
 - placement of blocks on the secondary storage device
 - **buffering blocks in main memory**
- Considered part of the operating system

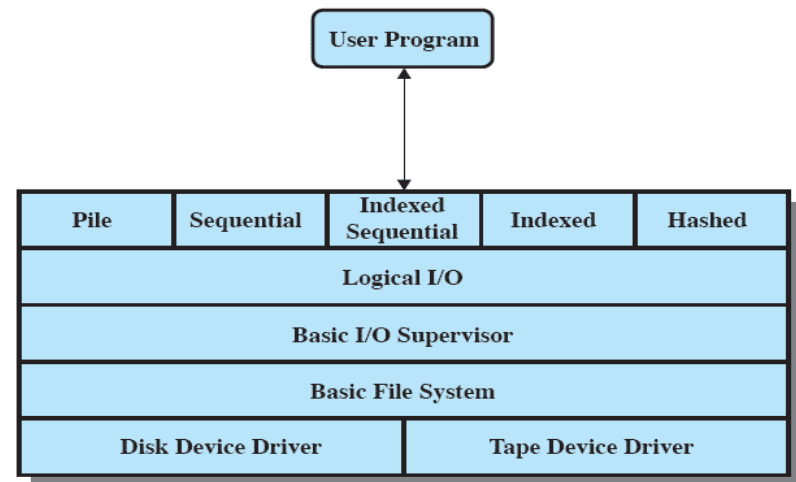


Figure 12.1 File System Software Architecture

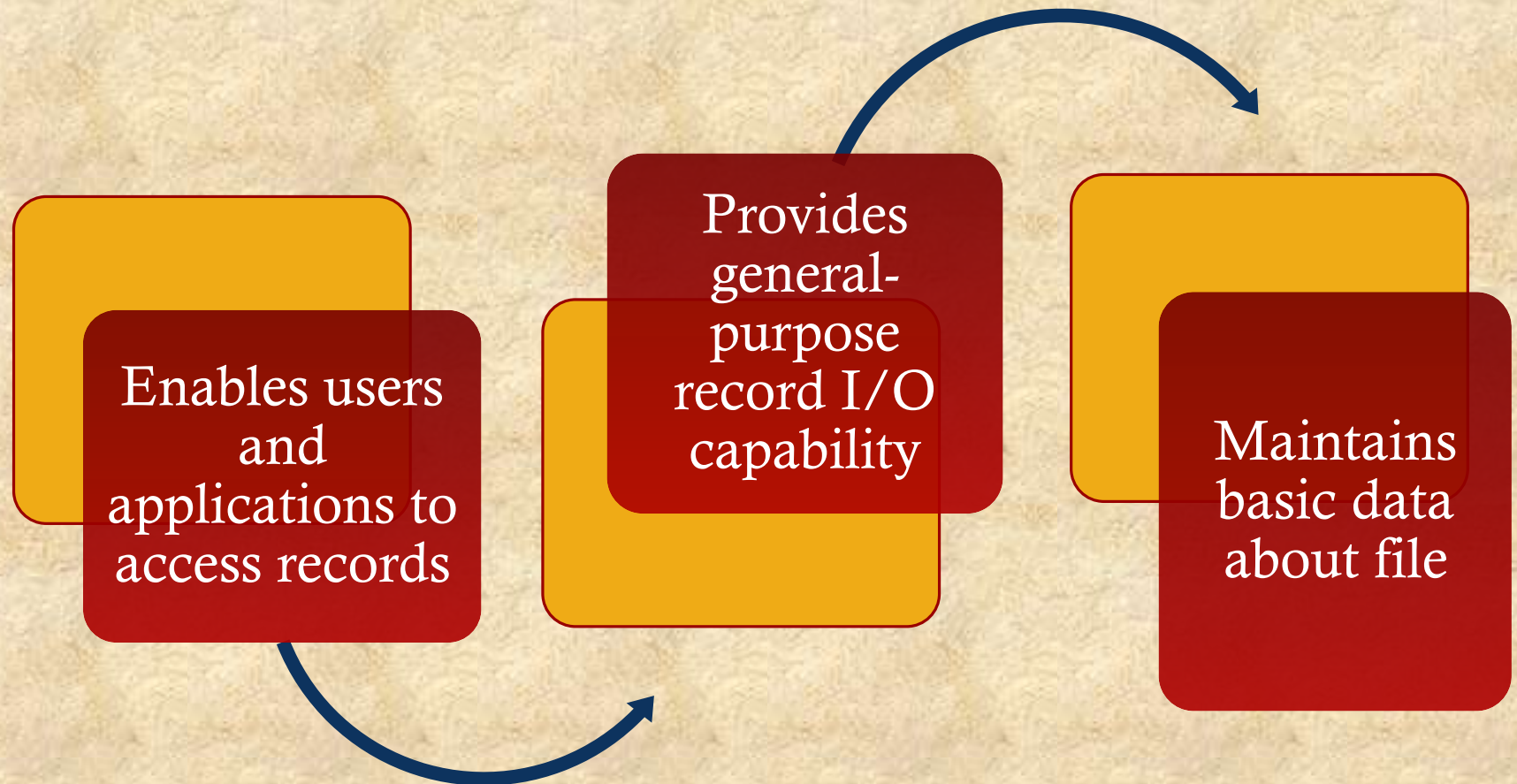




Basic I/O Supervisor

- 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



Logical I/O

This level is the interface between the logical commands issued by a program and the physical details required by the disk.

Logical units of data versus physical blocks of data to match disk requirements.

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**

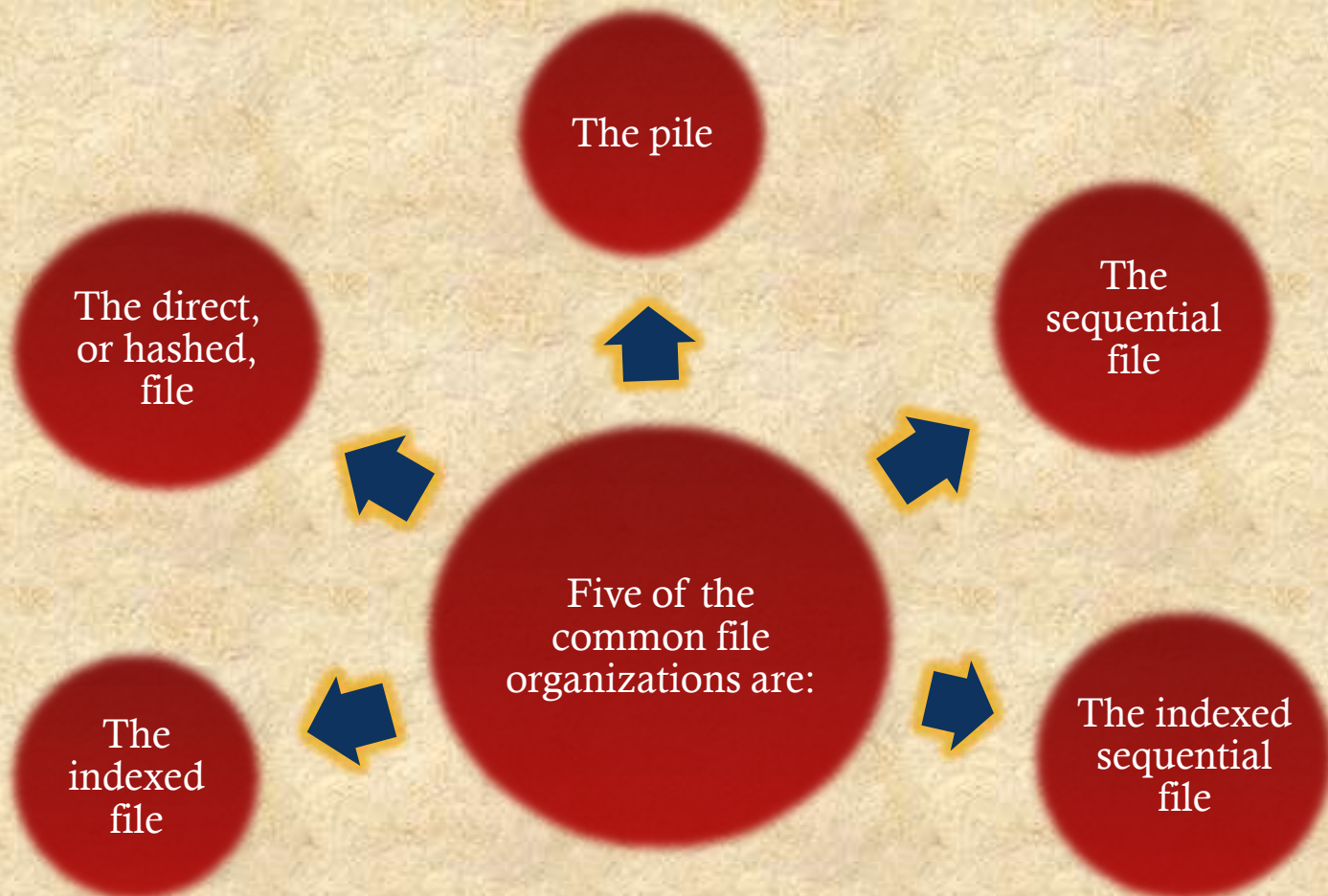


File Organization and Access

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

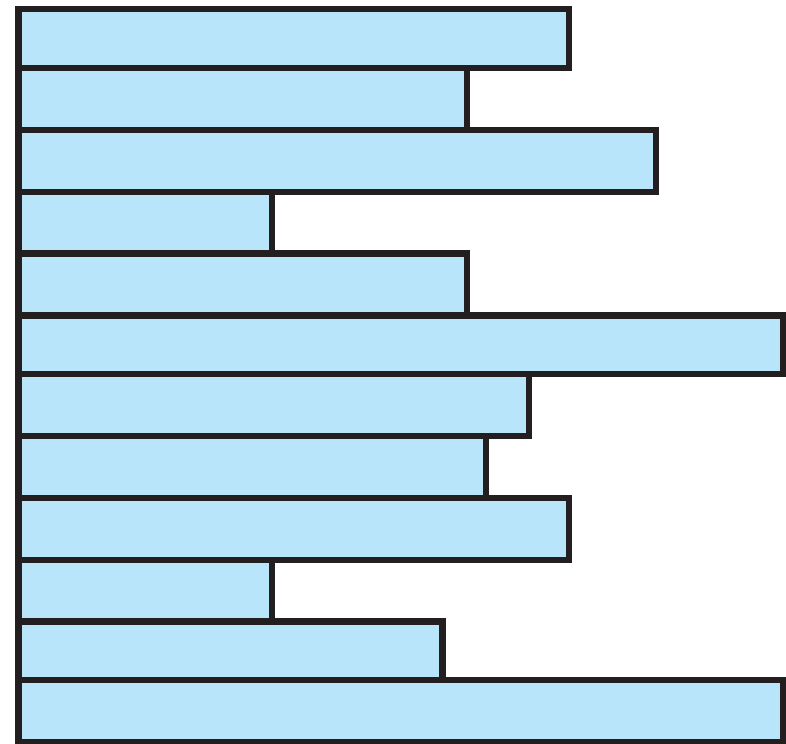


File Organization Types



The Pile

- Least complicated form of file organization
- Data are collected in the order they arrive
- Purpose is simply to accumulate the mass of data and save it

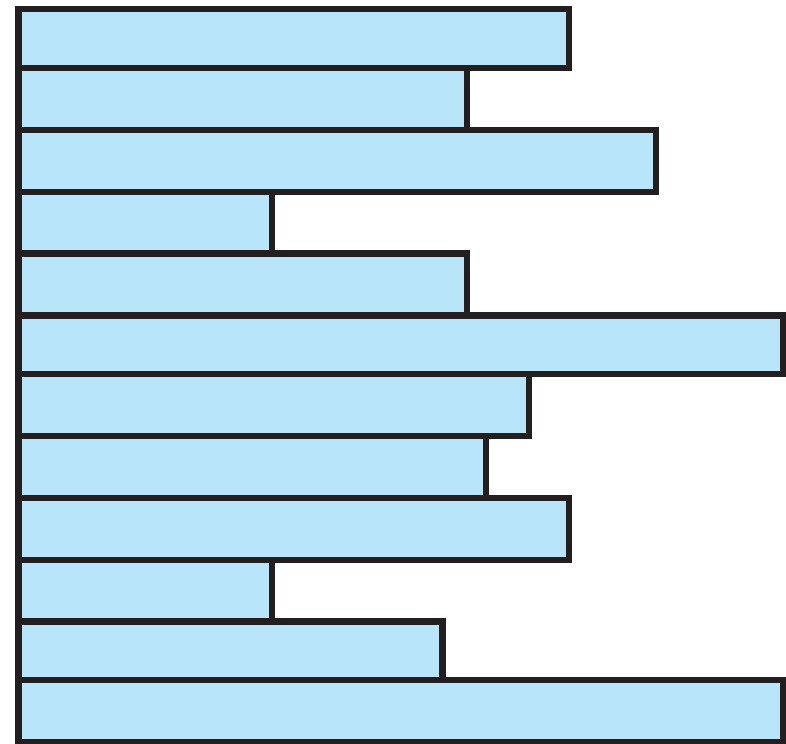


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

The Pile

- Each record consists of one burst of data
- Records may have different fields, or similar fields in different orders.

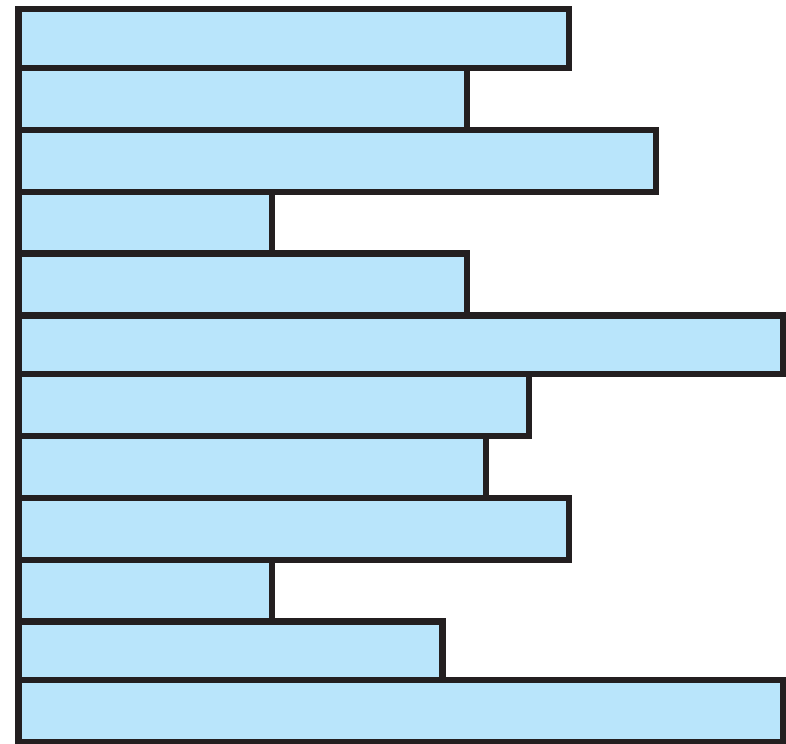


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

The Pile

- Because there is no structure to the pile file, record access is by exhaustive search.
- That is, if we wish to find a record that contains a particular field with a particular value,
- it is necessary to examine each record in the pile until the desired record is found or the entire file has been searched



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
- All records are of the same length,
- Records consist of the same number of fixed-length fields in a particular order.

Fixed-length records

Fixed set of fields in fixed order

Sequential order based on key field

(b) Sequential File

The Sequential File

- One particular field, usually the first field in each record, is referred to as the **key field**.
- The key field uniquely identifies the record;
- Thus key values for different records are always different.

Fixed-length records

Fixed set of fields in fixed order

Sequential order based on key field

(b) Sequential File

The Sequential File

- The records are stored in key sequence:
- alphabetical order for a text key, and
- numerical order for a numerical key.

Fixed-length records

Fixed set of fields in fixed order

Sequential order based on key field

(b) Sequential File

The Sequential File

- Typically used in batch applications if they involve the processing of all the records (e.g., a billing or payroll application).
- Easily stored on tape as well as disk

Fixed-length records

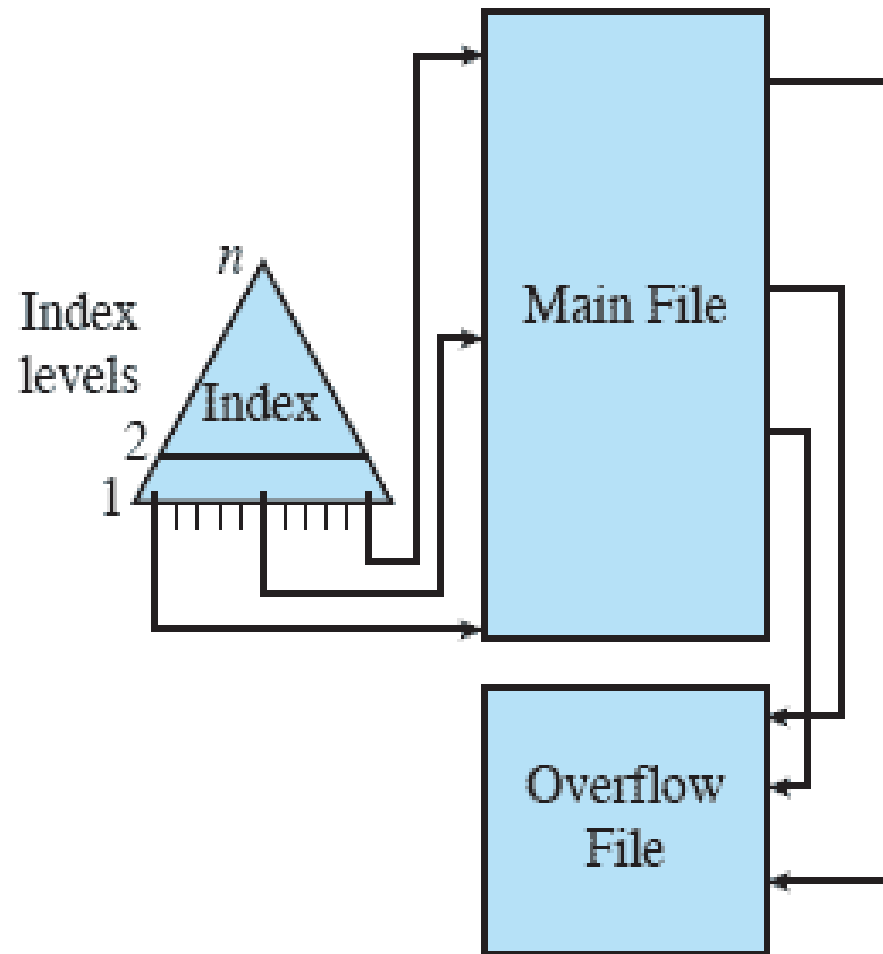
Fixed set of fields in fixed order

Sequential order based on key field

(b) Sequential File

Indexed Sequential File

- The indexed sequential file maintains the key characteristic of the sequential file:
- Records are organized in sequence based on a key field

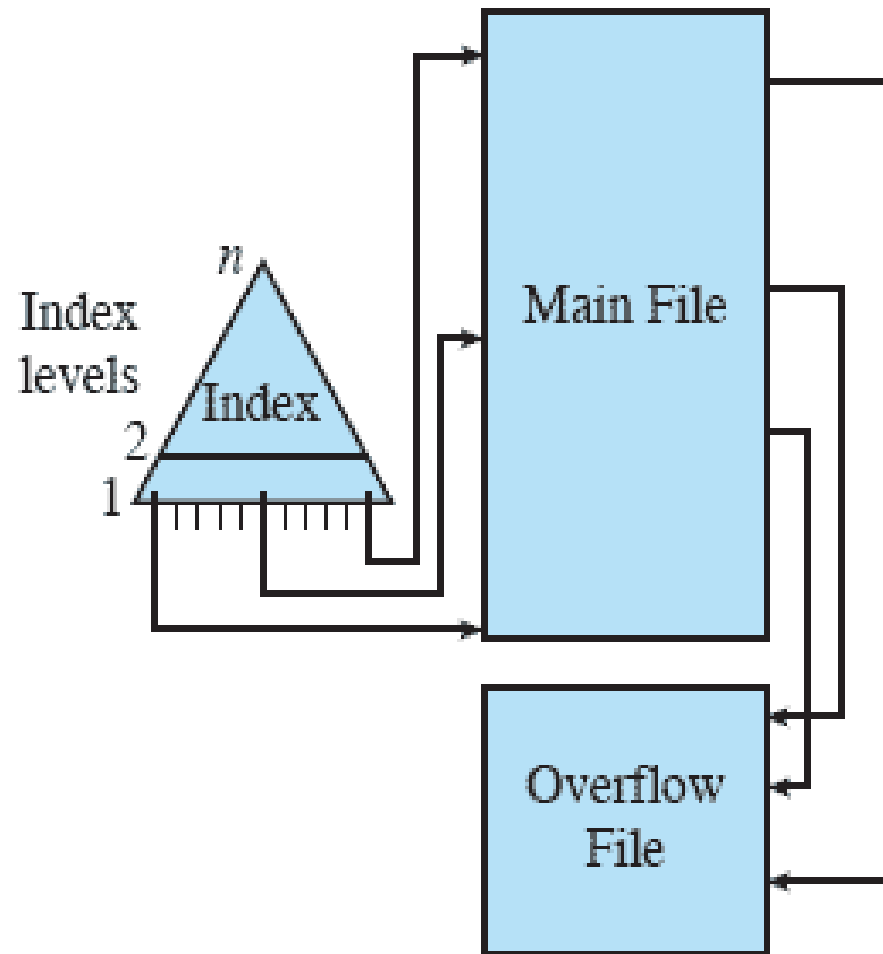


(c) Indexed Sequential File

Indexed Sequential File

■ Two features are added:-

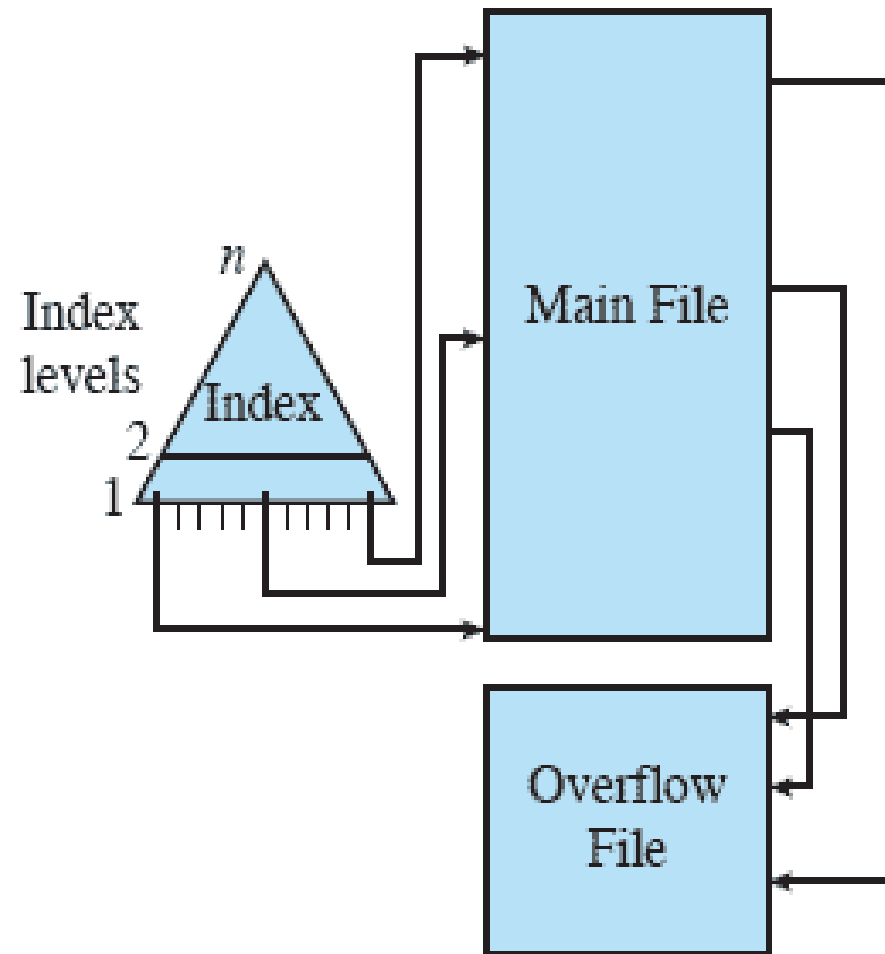
- 1) Adds an index to the file to support random access
- 2) An overflow file.



(c) Indexed Sequential File

Indexed Sequential File

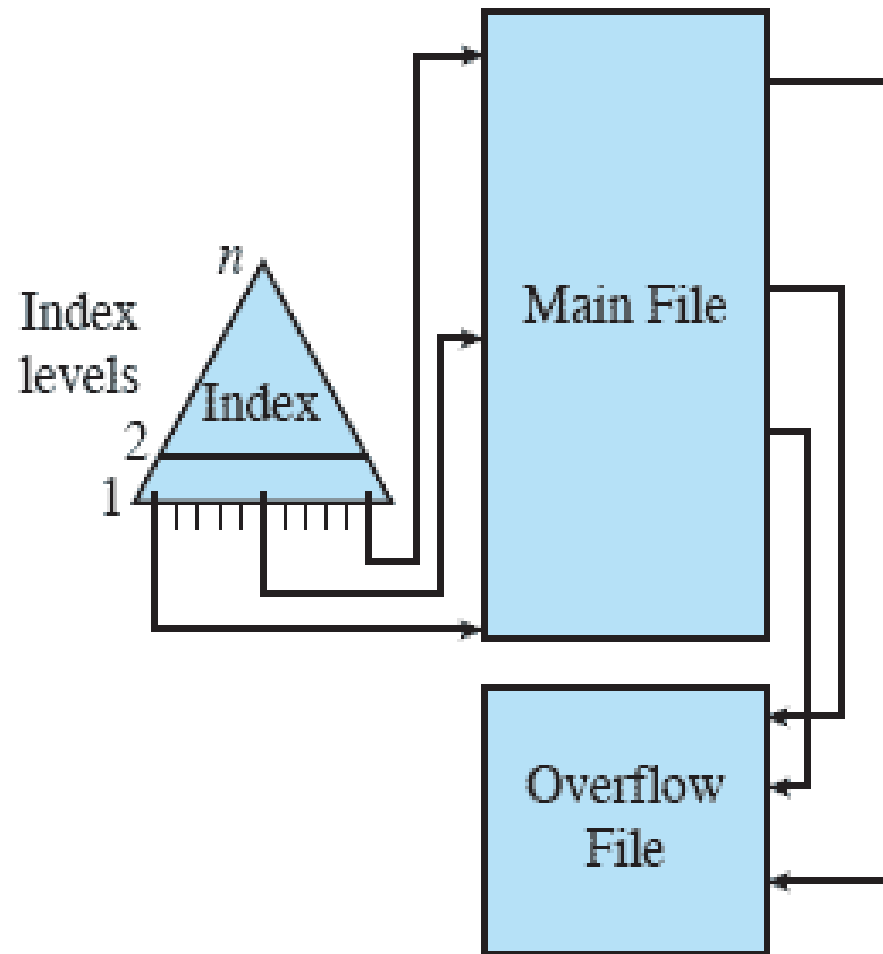
- Each record in the index file consists of two fields:
 - 1) a key field, which is the same as the key field in the main file,
 - 2) and a pointer into the main file.



(c) Indexed Sequential File

Indexed Sequential File

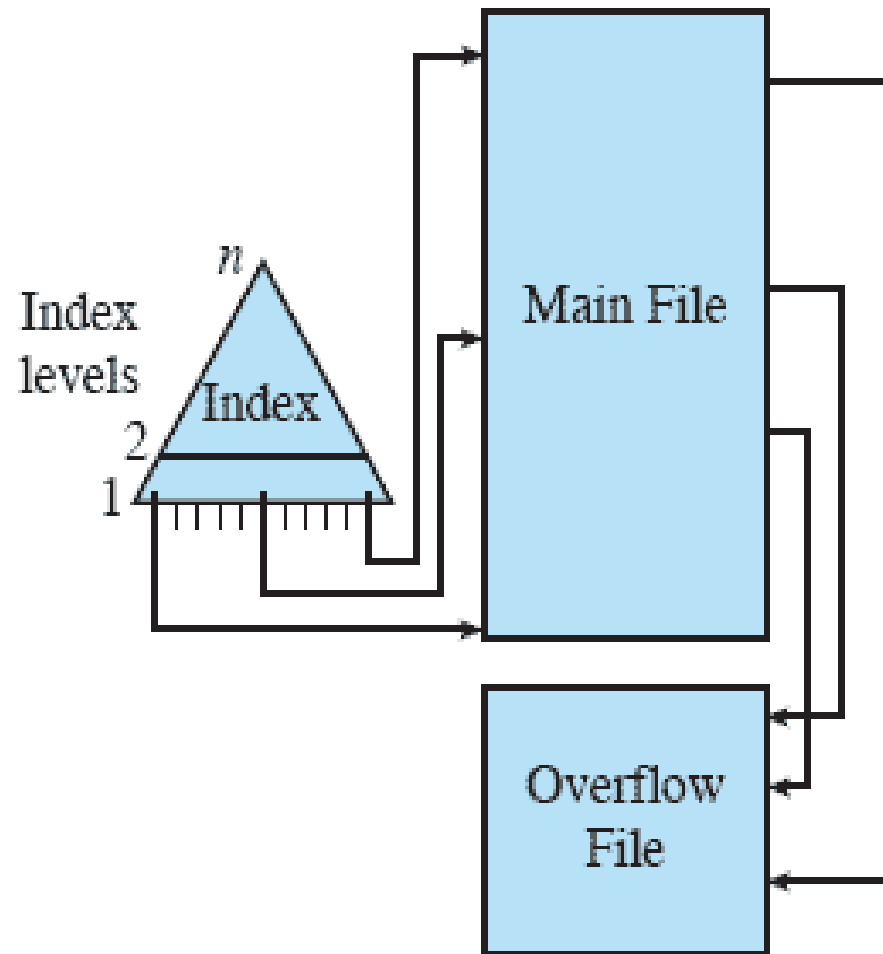
- To find a specific field, the index is searched to find the highest key value that is equal to or precedes the desired key value.
- The search continues in the main file at the location indicated by the pointer.



(c) Indexed Sequential File

Indexed Sequential File

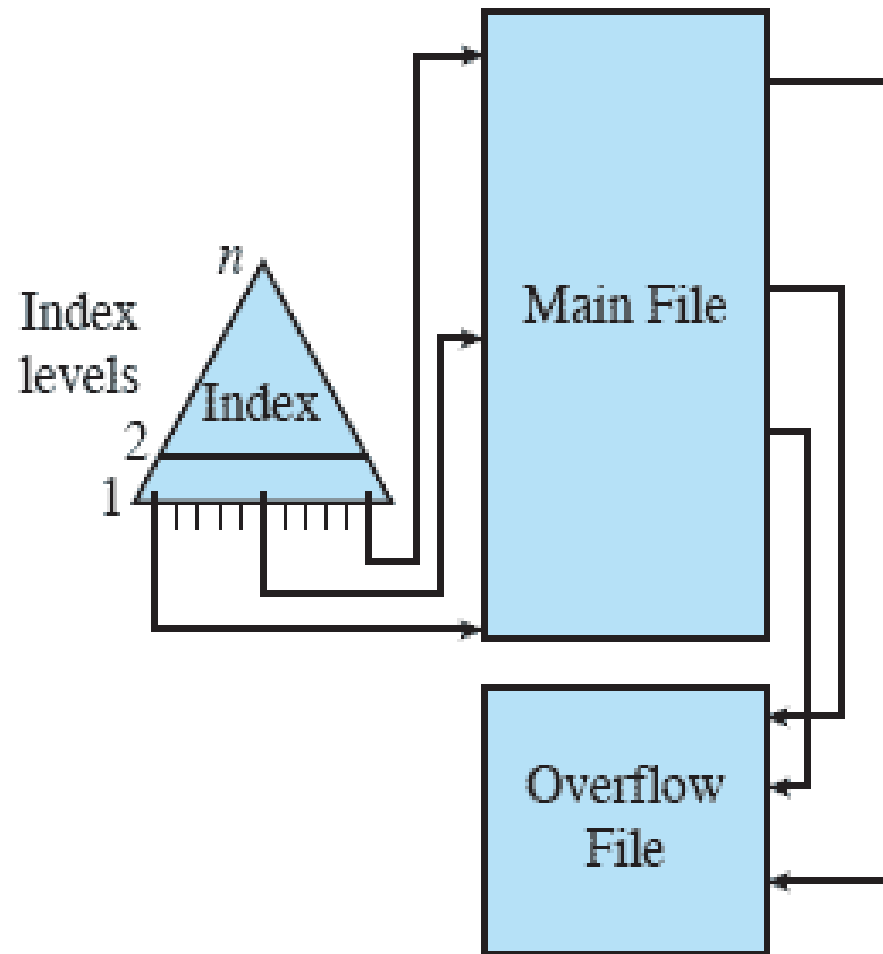
- Consider a sequential file with 1 million records.
- To search for a particular key value will require on average one half million record accesses.
- Now suppose that an index containing 1,000 entries is constructed, with the keys in the index more or less evenly distributed over the main file.
- The average search length is reduced.



(c) Indexed Sequential File

Indexed Sequential File

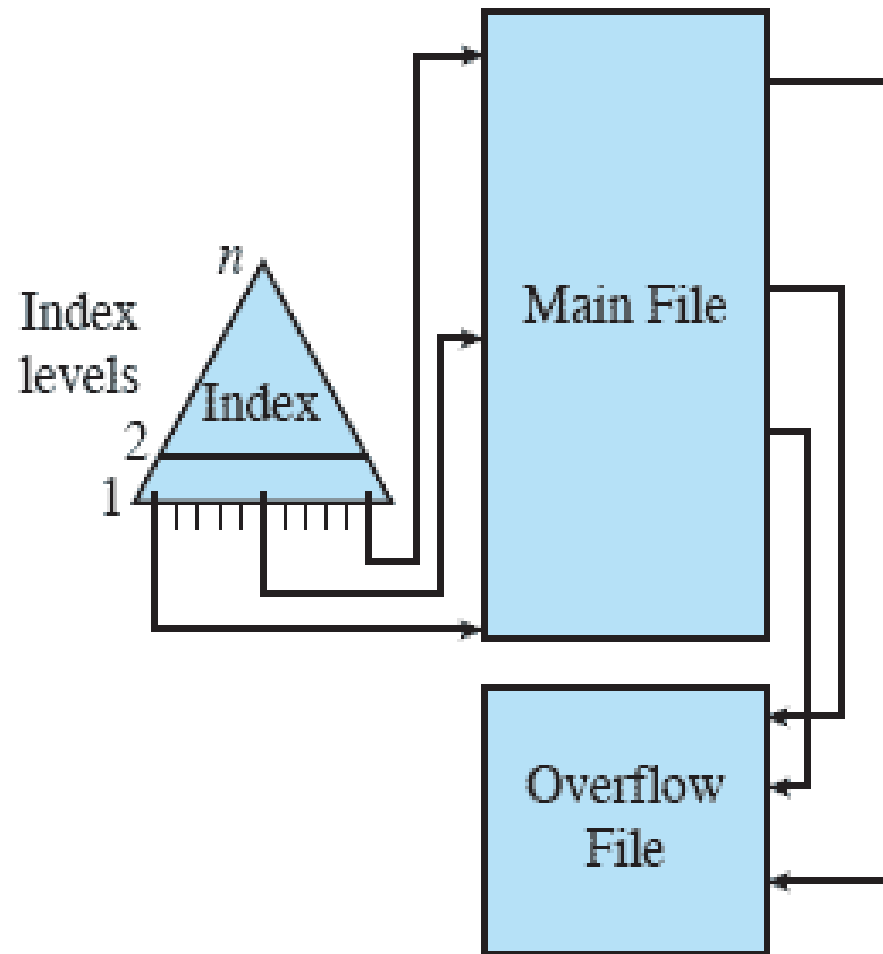
- Additions to the file are handled in the following manner:
- Each record in the main file contains an additional field not visible to the application, which is a pointer to the overflow file.



(c) Indexed Sequential File

Indexed Sequential File

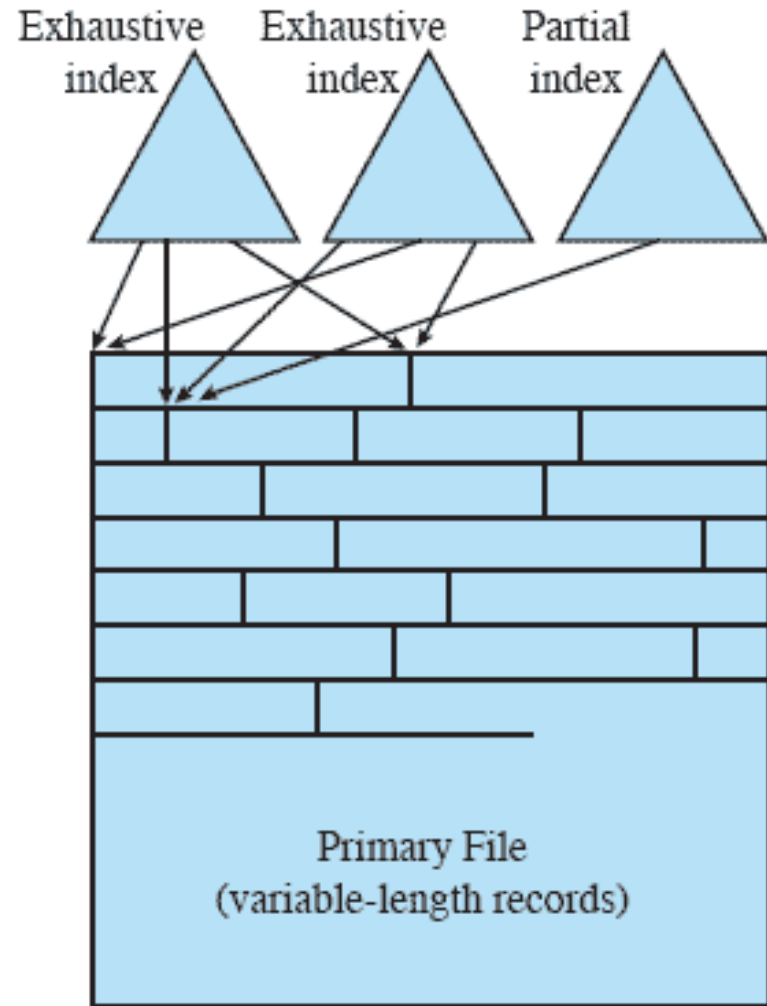
- When a new record is to be inserted into the file,
- it is added to the overflow file.
- The record in the main file that immediately precedes the new record in logical sequence is updated to contain a pointer to the new record in the overflow file.



(c) Indexed Sequential File

Indexed File

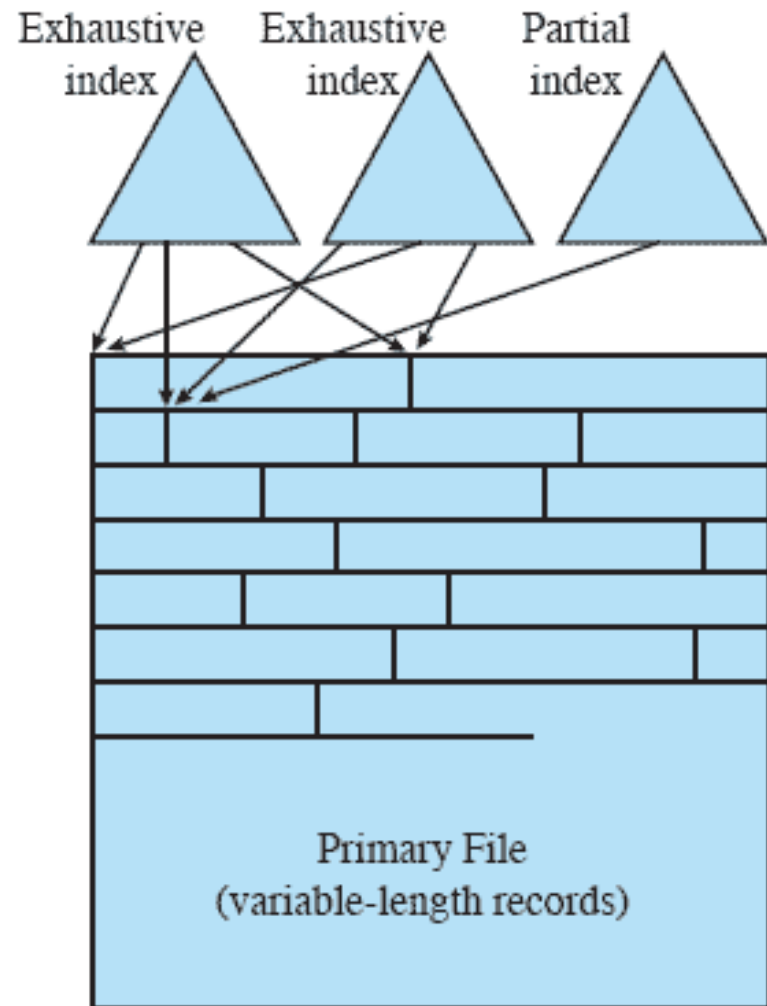
- A structure is needed that employs multiple indexes, one for each type of field that may be the subject of a search.
- Records are accessed only through their indexes
- The result is that there is now no restriction on the placement of records as long as a pointer in at least one index refers to that record.
- Variable-length records can be employed



(d) Indexed File

Indexed File

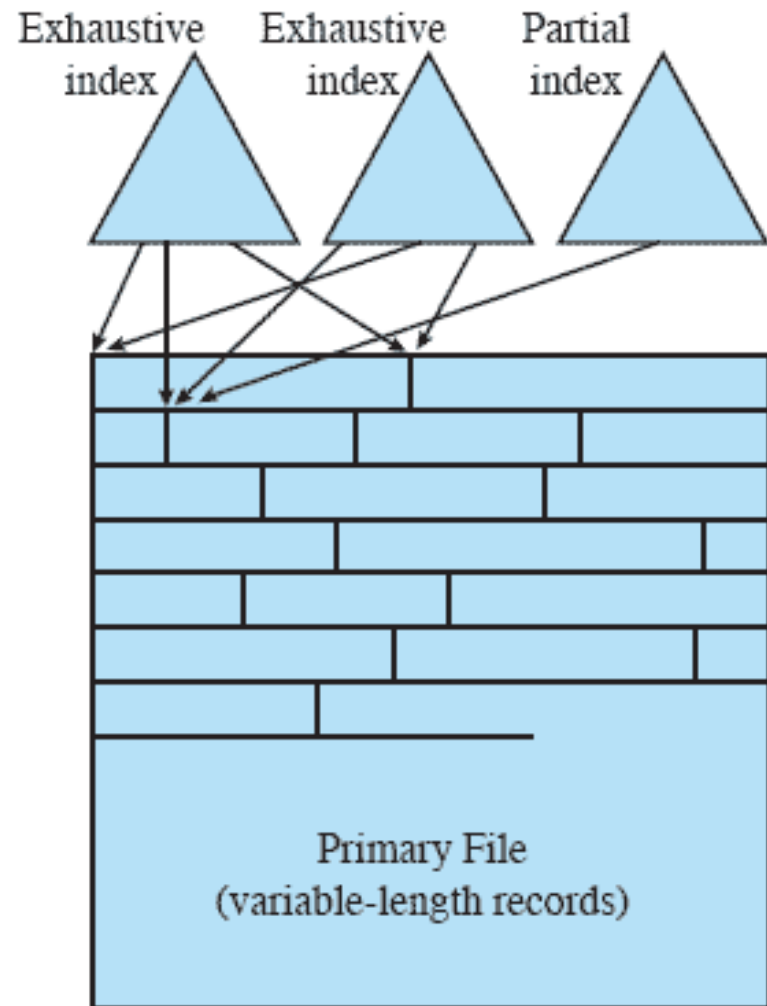
- Two types of indexes are used.
- **An exhaustive index contains one entry for every record in the main file.**
- **A partial index contains entries to records where the field of interest exists.**
- With variable-length records, **some records will not contain all fields.**



(d) **Indexed File**

Indexed File

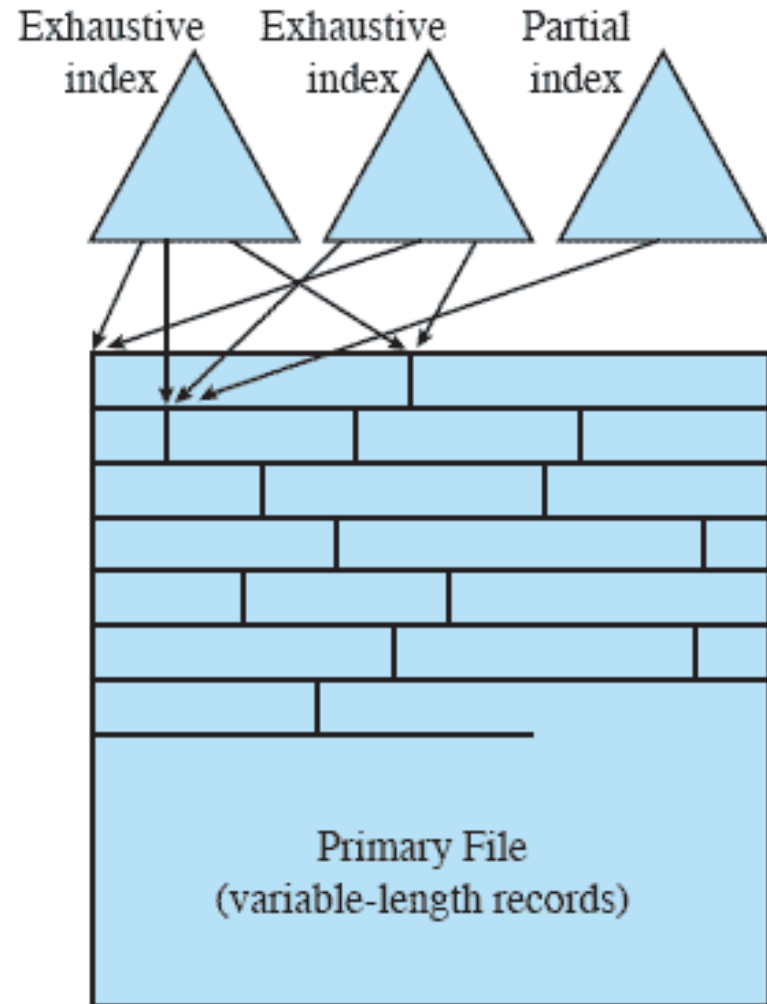
- Indexed files are used mostly in **applications where timeliness is critical and where data are rarely processed exhaustively.**
- Examples are airline reservation systems and inventory control systems.



(d) **Indexed File**

Indexed File

- When a new record is added to the main file, all of the index files must be updated.
- Used mostly in applications where timeliness of information is critical
- Examples would be airline reservation systems and inventory control systems



(d) Indexed File

Direct or Hashed File

- The direct file makes use of hashing on the key value.
- No concept of sequential ordering here.

Examples are:

- directories
- pricing tables
- schedules
- name lists

Direct or Hashed File

- 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, and name lists.

Examples are:

- directories
- pricing tables
- schedules
- name lists

Table 12.2 Information Elements of a File Directory

File Directory Information

- Associated with any file management system and collection of files is a file directory.
- The directory contains information about the files, including attributes, location, and ownership.
- The directory is itself a file, accessible by various file management routines.



File Directory Information

Table 12.2 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-the information typically stored in the directory for each file in the system.

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

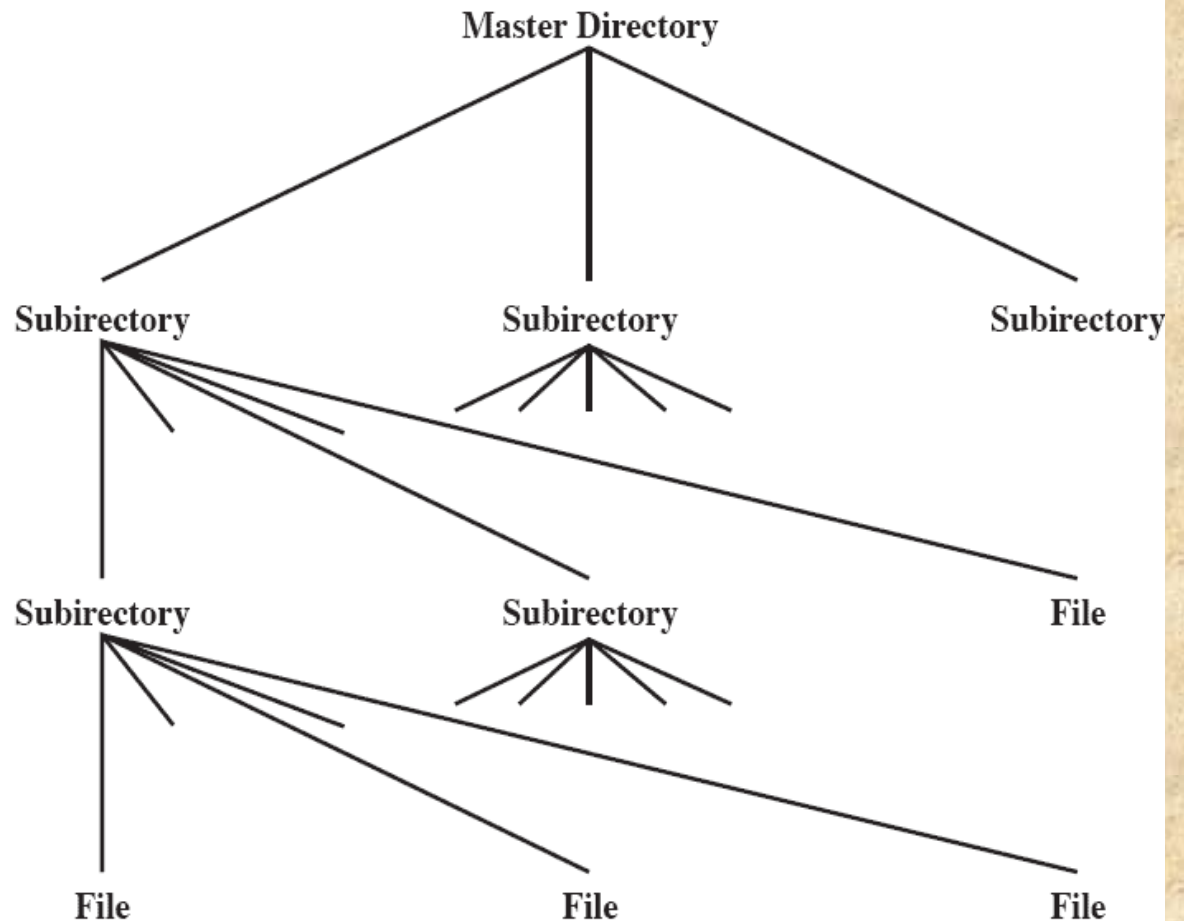
Names must be unique only within the collection of files of a single user

File system can easily enforce access restriction on directories

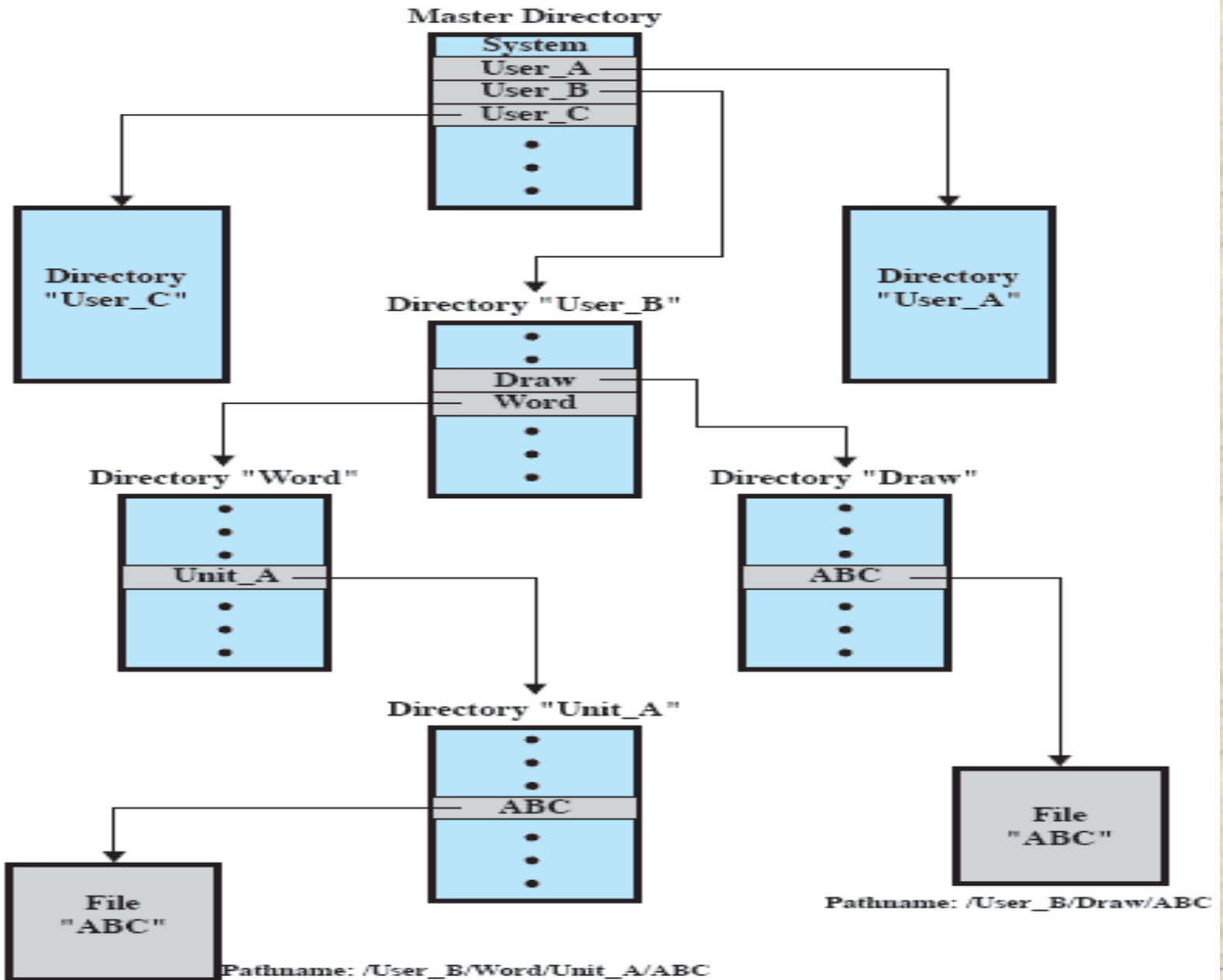
Figure 12.4

Tree-Structured Directory

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



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

Owner

usually the
initial creator
of the file

has full rights

may grant
rights to
others

Specific Users

individual
users who are
designated by
user ID

User Groups

a set of users
who are not
individually
defined

All

all users who
have access to
this system

these are
public files



Secondary Storage Management

- 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.
- This raises two management issues.
 - First, space on secondary storage must be allocated to files,
 - Second, it is necessary to keep track of the space available for allocation.



File Allocation

- Disks are divided into physical blocks (sectors on a track)
- Files are divided into logical blocks (subdivisions of the file)
- Logical block size = some multiple of a physical block size

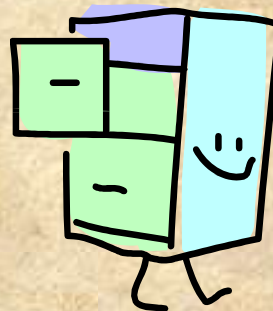


File Allocation

- The operating system or file management system is responsible for allocating blocks to files
- Space is allocated to a file as one or more *portions* (contiguous set of allocated disk blocks). A portion is the logical block size
- *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

Summarizing the 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

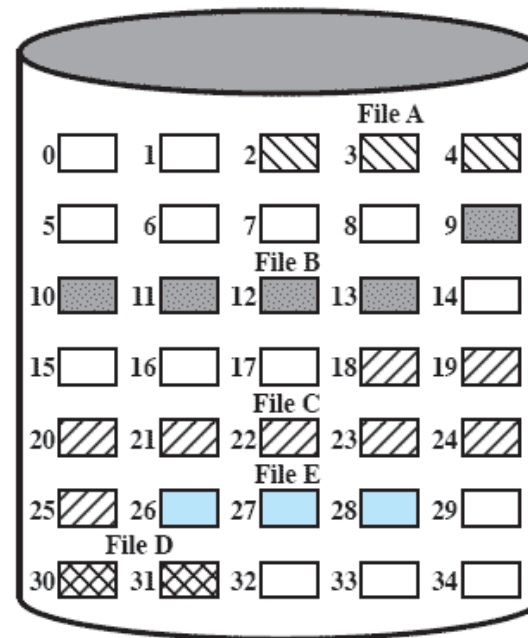
	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size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

Contiguous File Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation

- Thus, this is a preallocation strategy,

- Preallocation strategy using variable-size portions



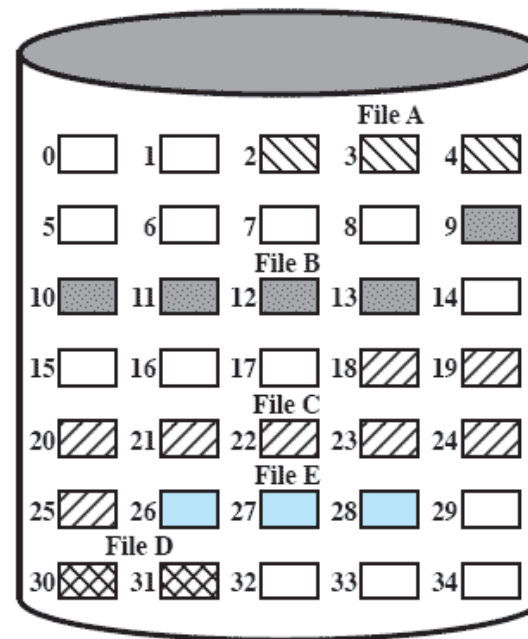
File Allocation Table		
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.9 Contiguous File Allocation

Contiguous File Allocation

- The file allocation table needs **just a single entry for each file**,

- showing the **starting block and the length of the file**.



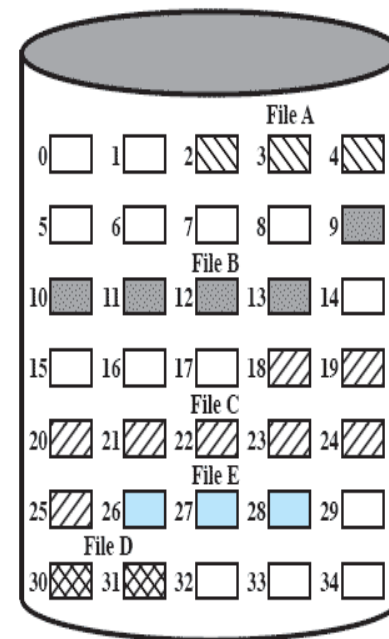
File Allocation Table

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

Contiguous File Allocation

- Contiguous allocation is the **best from the point of view of the individual sequential file**.
- Multiple blocks can be read in at a time to improve I/O performance for sequential processing.
- It is also easy to retrieve a single block.
- For example, if a file starts at block **b**, and the **i**th block of the file is wanted, its location on secondary storage is simply $b + i - 1$.

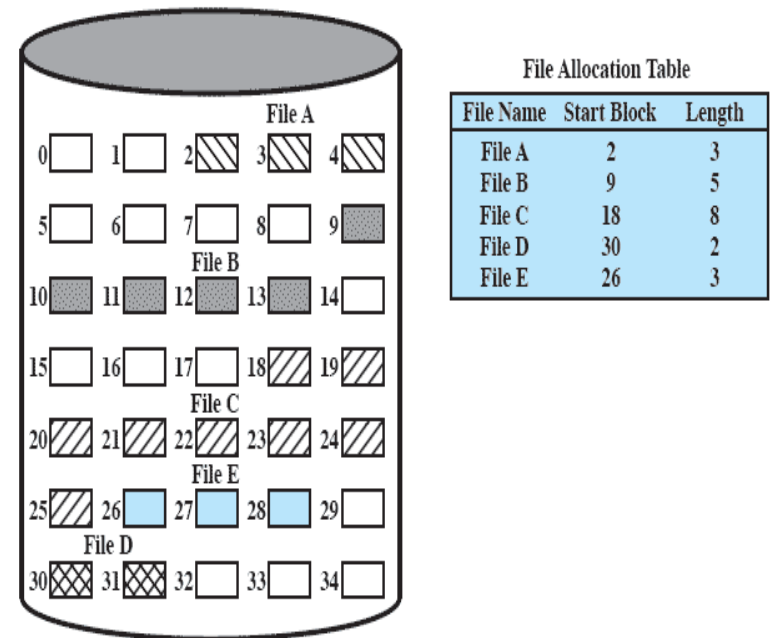


File Allocation Table		
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

12.9 Figure 12.7 Contiguous File Allocation

Contiguous File Allocation

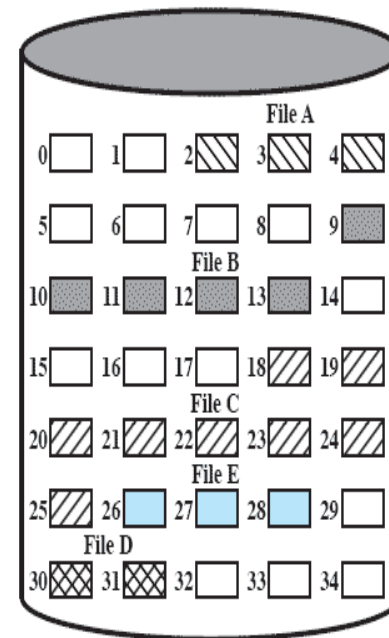
- Contiguous allocation presents some problems?



12.9 Figure 12.7 Contiguous File Allocation

Contiguous File Allocation

- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length.
- Soln?

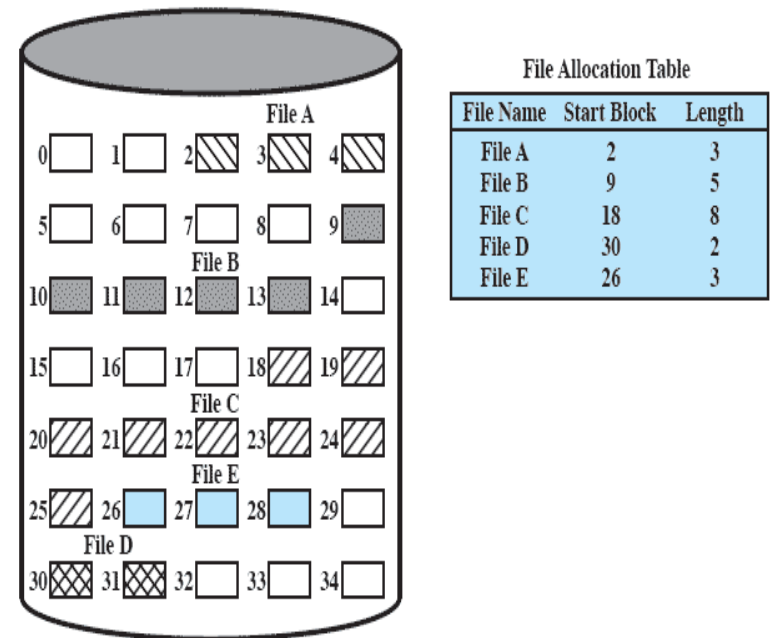


File Allocation Table		
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	4
File D	30	2
File E	26	3

12.9 Figure 12.7 Contiguous File Allocation

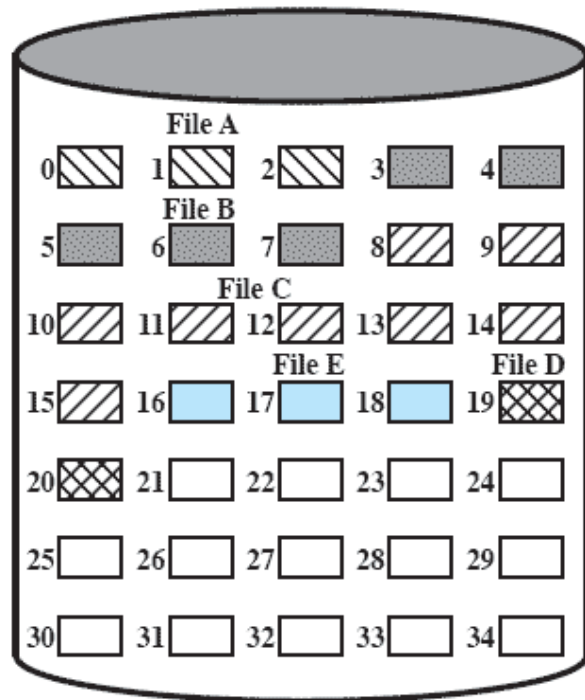
Contiguous File Allocation

- Soln?
- From time to time, it will be necessary to perform a compaction algorithm to free up additional space on the disk



12.9 Figure 12.7 Contiguous File Allocation

After Compaction



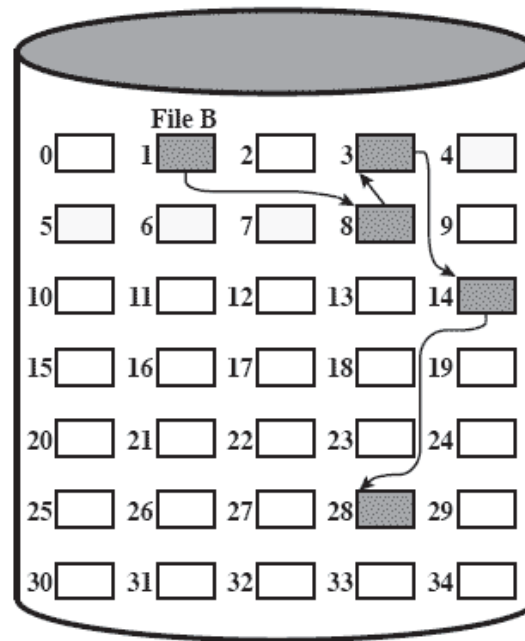
File Allocation Table

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.
- showing the starting block and the length of the file.



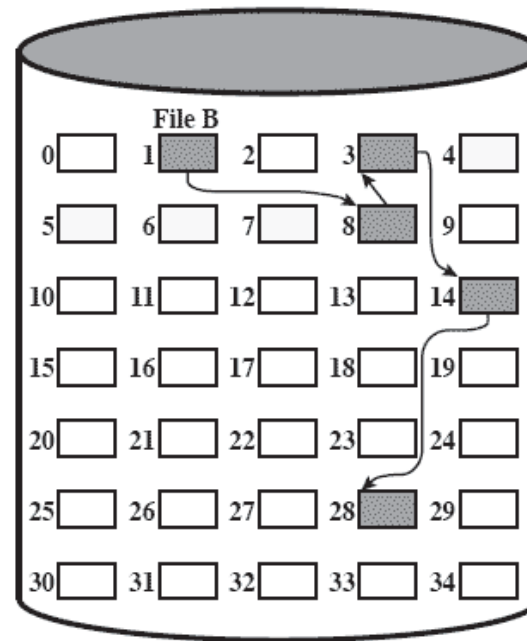
File Allocation Table

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

Figure 12.11 Chained Allocation

Chained Allocation

- Although preallocation is possible,
- **it is more common simply to allocate blocks as needed.**
- The selection of blocks is now a simple matter:
- **Any free block can be added to a chain.**



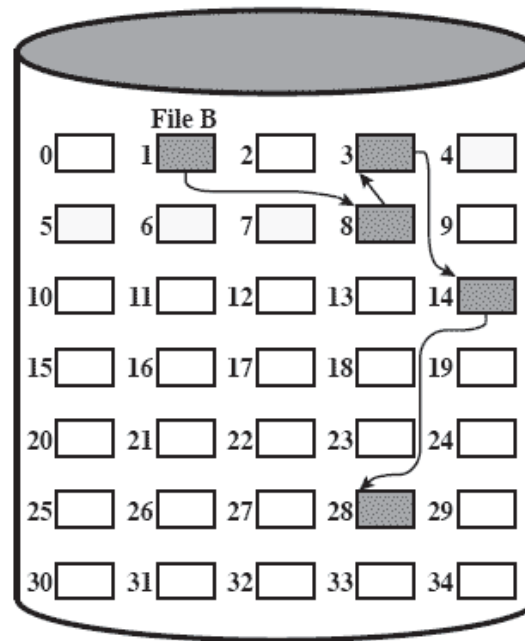
File Allocation Table

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

Figure 12.11 Chained Allocation

Chained Allocation

- **There is no external fragmentation** to worry about because only one block at a time is needed
- Better for sequential files



File Allocation Table

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

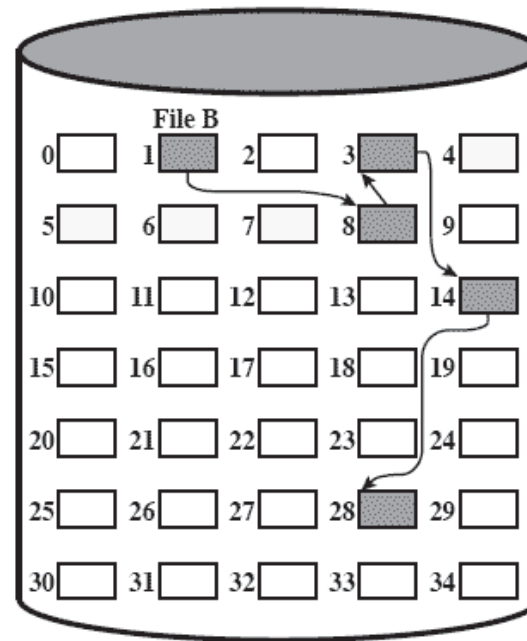
Figure 12.11 Chained Allocation

Chained Allocation

- One consequence of chaining, as described so far, is that there is **no accommodation of the principle of locality**.

- To overcome this problem

- Some systems **periodically consolidate files**

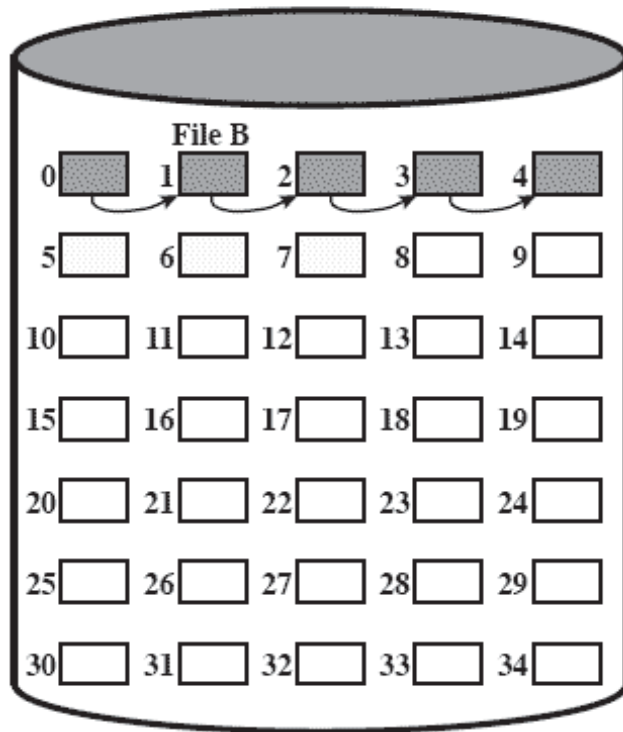


File Allocation Table

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

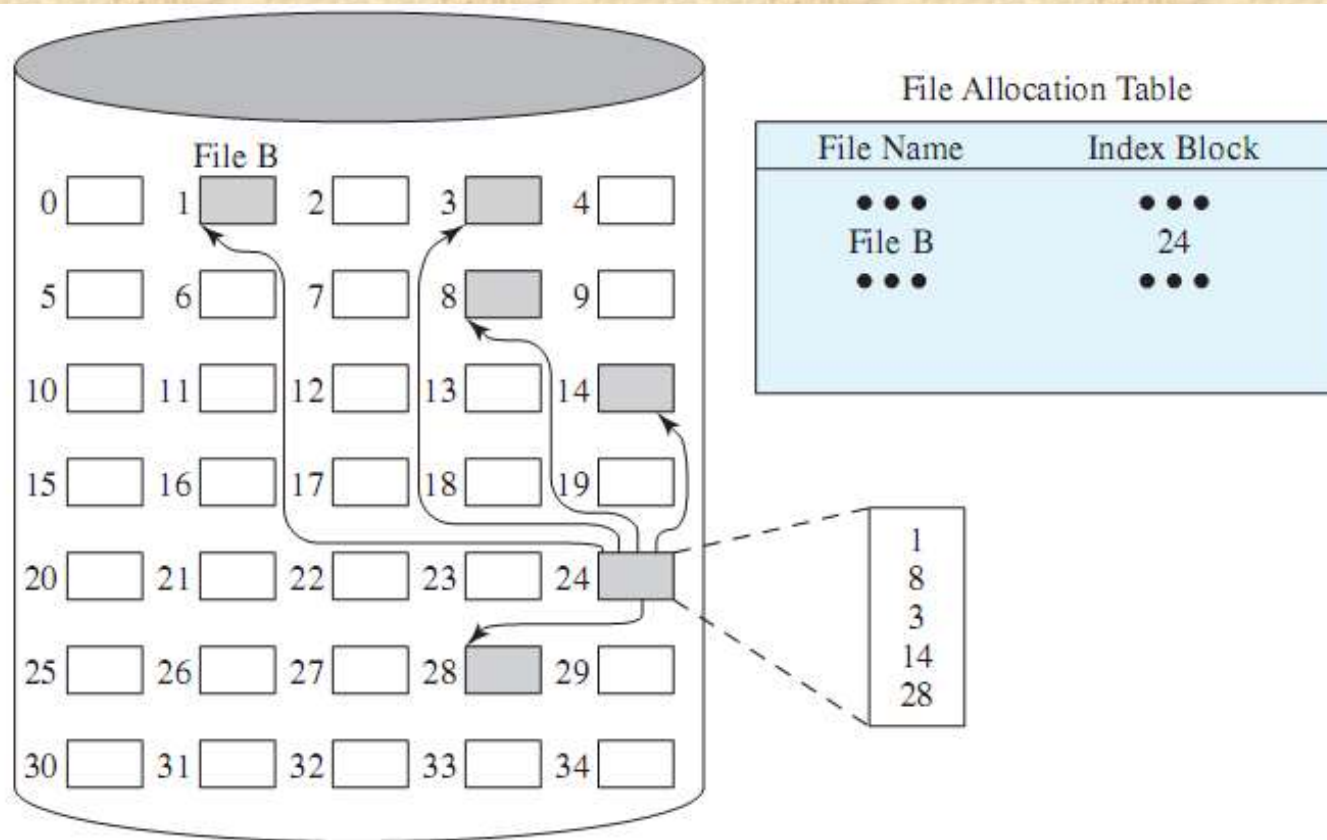
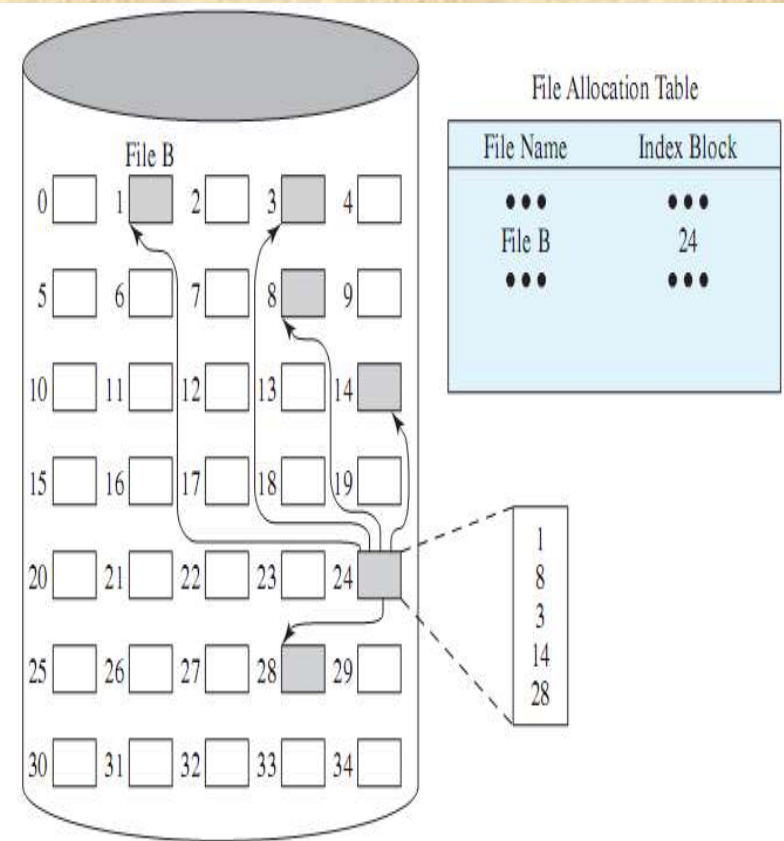


Figure 12.13 Indexed Allocation with Block Portions

Indexed Allocation with Block Portions

- In this case, the file allocation table contains a separate one-level index for each file
- The index has one entry for each portion allocated to the file.
- File index for a file is kept in a separate block, and the entry for the file in the file allocation table points to that block



Indexed Allocation with Variable Length Portions

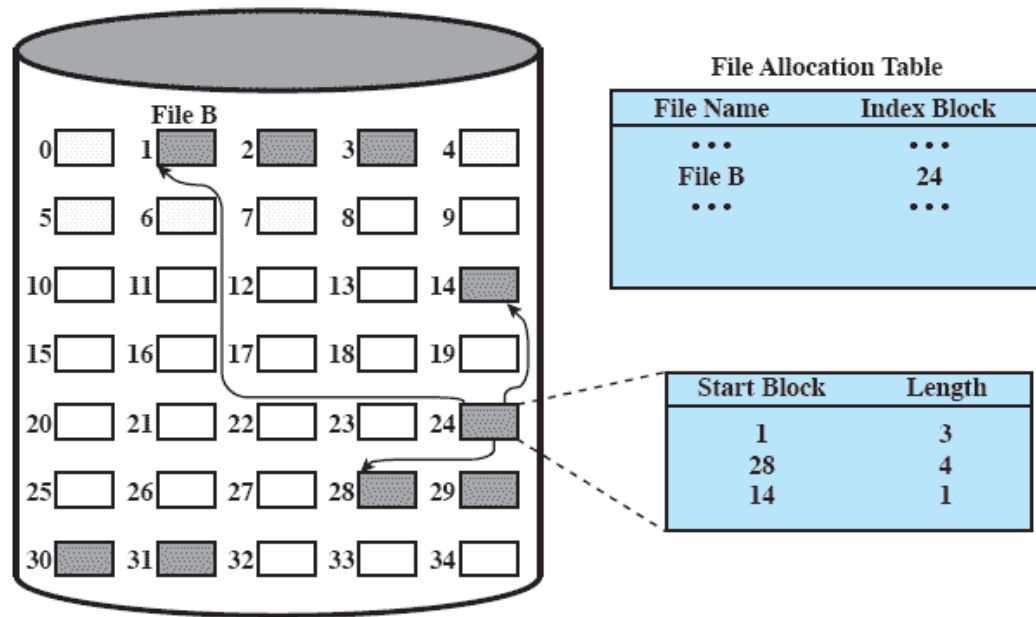
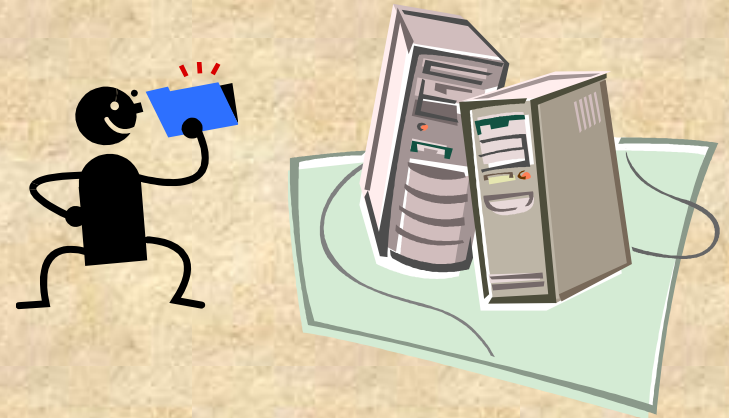


Figure 12.14 Indexed Allocation with Variable-Length Portions

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 (Bit Vectors)

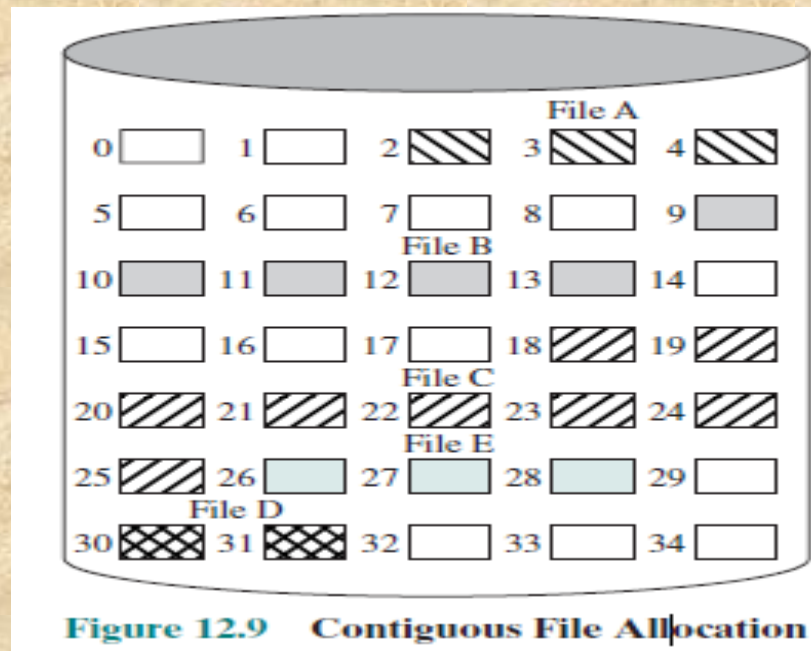
- This method uses a vector containing one bit for each block on the disk
- Each entry of a 0 corresponds to a free block,
- Each 1 corresponds to a block in use

Advantages:

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

Bit Tables (Bit Vectors)

- For the disk layout of Figure 12.9 , A vector of length 35 is needed and would have the following value:
- 0011100001111100001111111111011000



Bit Tables (Bit Vectors)

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

Chained Free Portions

- If allocation is a block at a time, simply choose the free block at the head of the chain and adjust the first pointer or length value.
- If allocation is by variable-length portion, a first-fit algorithm may be used:
 - The headers from the portions are fetched one at a time to determine the next suitable free portion in the chain.
 - Again, pointer and length values are adjusted.

Chained Free Portions

- 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
- 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

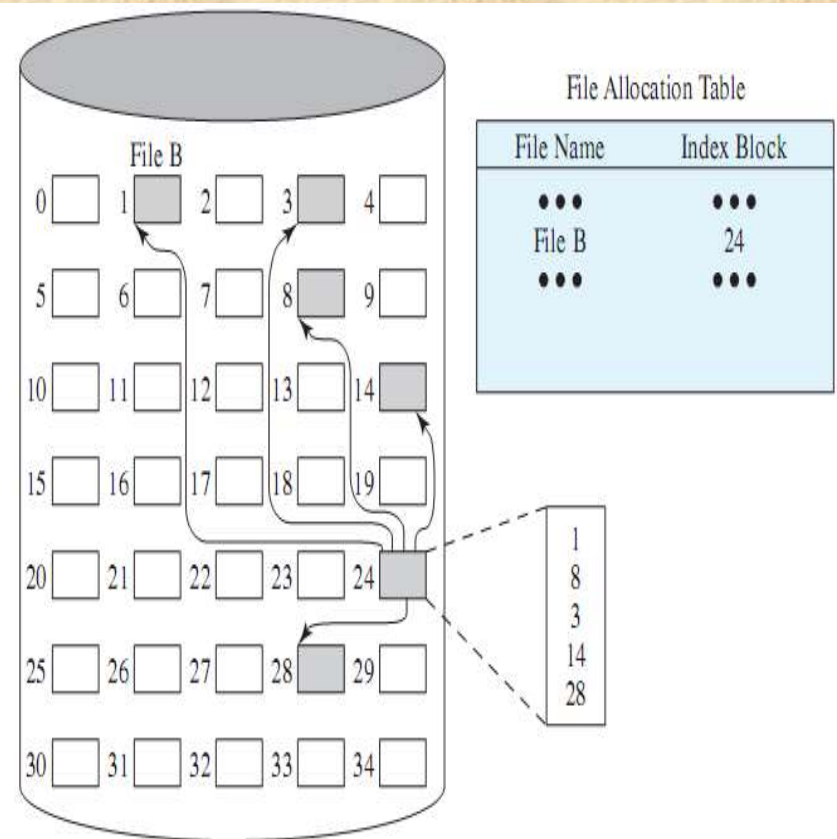


Figure 12.11 Indexed Allocation with Block Portions

Indexing

- 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

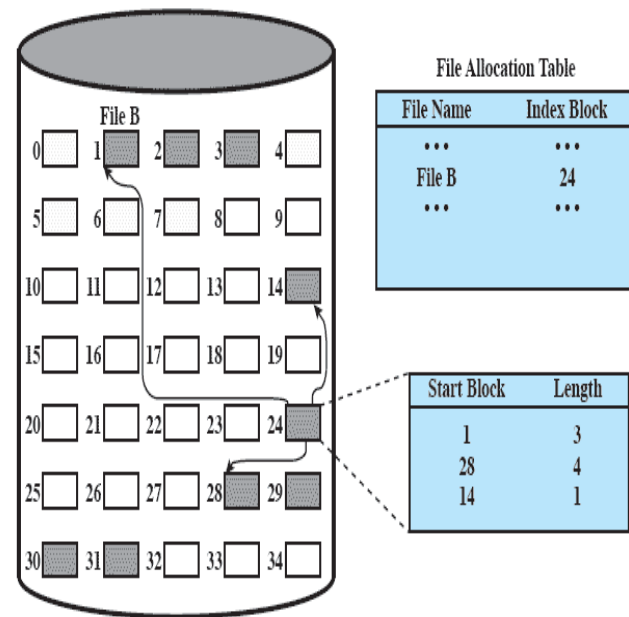


Figure 12.12 Indexed Allocation with Variable-Length Portions

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

Free Block List

- As the free block list is too large to store in main memory,
- Free Block List must be stored on disk rather than in main memory.
- There are two effective techniques for storing a small part of the list in main memory.

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

Free Block List

- Stack-
- When a new block is allocated, it is popped from the top of the stack, which is in main memory.
- Similarly, when a block is deallocated, it is pushed onto the stack.

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

Free Block List

- Stack-
- There only has to be a transfer between disk and main memory when
- the in-memory portion of the stack becomes either full or empty.

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

Free Block List

- Queue-
- A block is allocated by taking the first entry from the head of the queue and
- deallocated by adding it to the end of the tail of the queue.

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

Free Block List

- Queue-
- There only has to be a transfer between disk and main memory
- when either the in-memory portion of the head of the queue becomes empty or
- the in-memory portion of the tail of the queue becomes full.

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

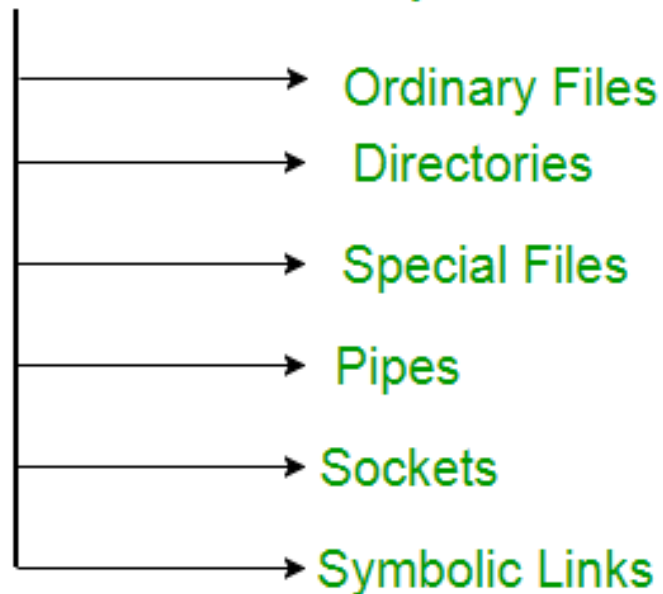
the list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory

Review

- File systems can support files organized as a sequence of bytes or as a sequence of records
- Access methods depend on file organization
- Disk storage of files can be contiguous, linked or indexed
- **Logical blocks of a file are mapped to one or more disk sectors to create physical blocks.**
- File Allocation Tables map files to disk locations

UNIX File Management

Classification of Unix File System :



UNIX File Management

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

Regular, or Ordinary

- **Regular files contain information entered in them by a user, an application program, or a system utility program.**
- **An ordinary file is a file on the system that contains data, text, or program instructions.**

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
- The file system does not impose any internal structure to a regular file but treats it as a stream of bytes.

UNIX File Management

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

Directory

- contains a list of file names plus pointers to associated inodes (index nodes)
- Directories are hierarchically organized

UNIX File Management

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

Special

- Contains no data but provides a mechanism to map physical devices to file names.
- **These file names are used to access peripheral devices, such as terminals and printers.**
- **Each I/O device is associated with a special file,**

UNIX File Management

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

Special

- **Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations.**
- **Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems.**

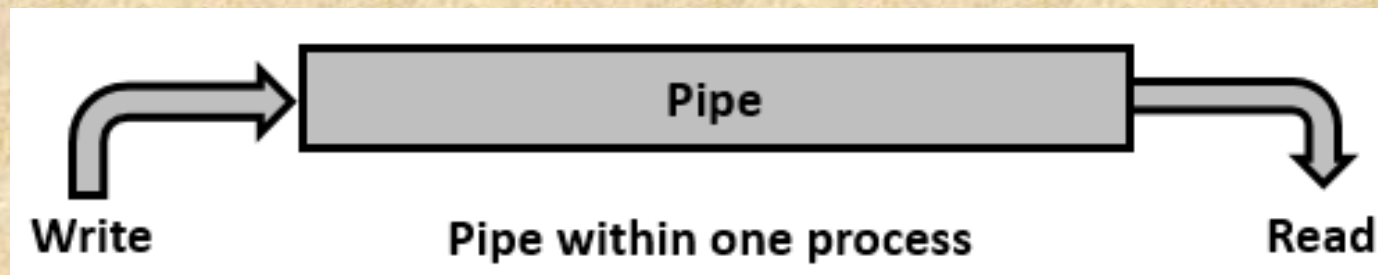
UNIX File Management

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

Named pipes

- **An interprocess communications facility**
- A pipe file buffers data received in its input
- so that a process that reads from the pipe's output receives the data on a first-in-first-out basis.

- A *pipe* is a section of shared memory that processes use for communication.
- One process writes information to the pipe, then the other process reads the information from the pipe.



UNIX File Management

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

Sockets

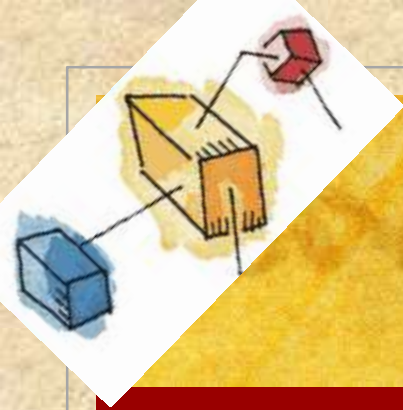
- **Sockets** – A Unix socket (or Inter-process communication socket) **is a special file which allows for advanced inter-process communication.**
- A Unix Socket is used in a client-server application framework.

UNIX File Management

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

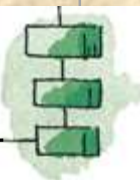
Symbolic links

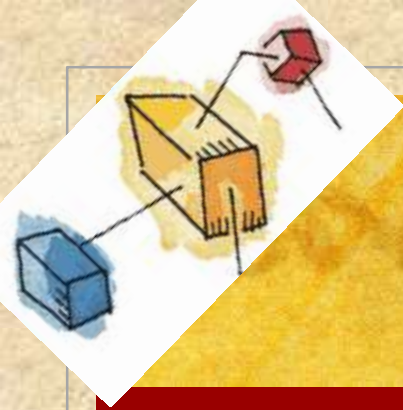
- A data file that contains the name of the file it is linked to



Symbolic link

- Symbolic link is used for referencing some other file of the file system.
- **Symbolic link is also known as Soft link. It contains a text form of the path to the file it references.**
- **To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to.**





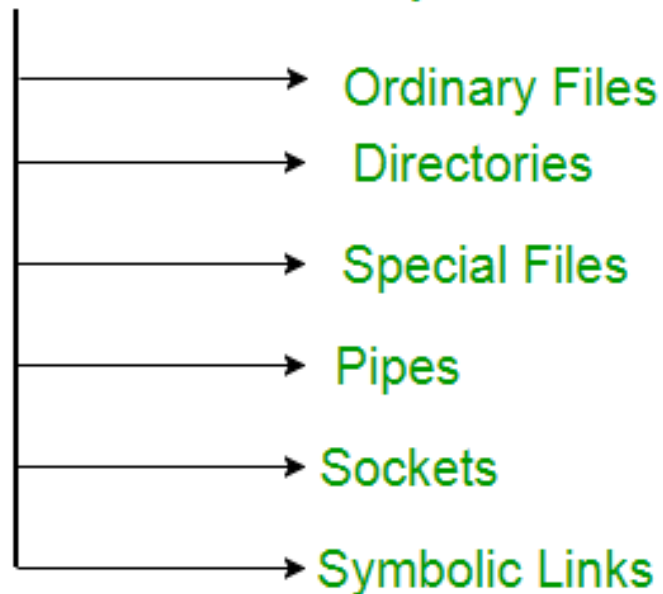
Symbolic link

- If we delete the soft link itself , the data file would still be there.If we delete the source file or move it to a different location, symbolic file will not function properly.
- In long-format output of `ls -l` ,
- Symbolic link are marked by the “l” symbol (that’s a lower case L).



UNIX File Management

Classification of Unix File System :

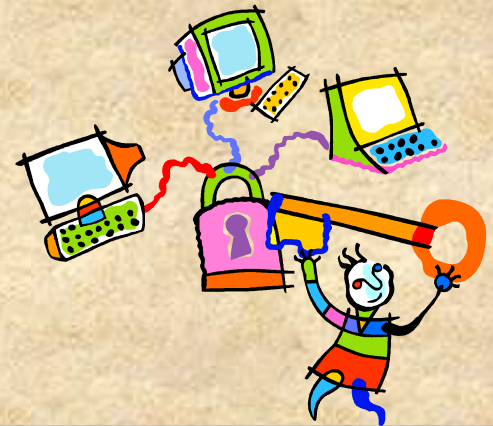


How to Find Out these File Types in Linux

- <https://www.geeksforgeeks.org/how-to-find-out-file-types-in-linux/>

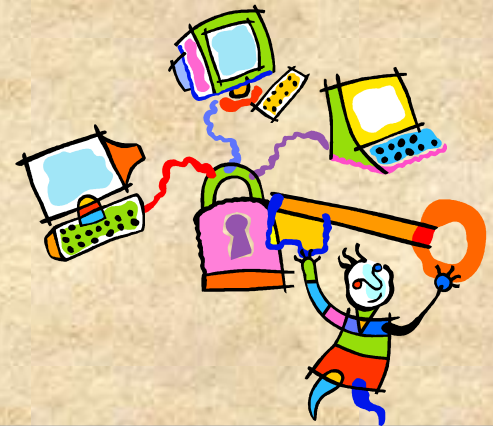
Inodes

- Modern UNIX operating systems support multiple file systems
- but map all of these into a uniform underlying system for supporting file systems and allocating disk space to files.
- All types of UNIX files are administered by the OS by means of inodes.



Inodes

- The inode (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory.
- Each inode stores the attributes and disk block locations of the object's data

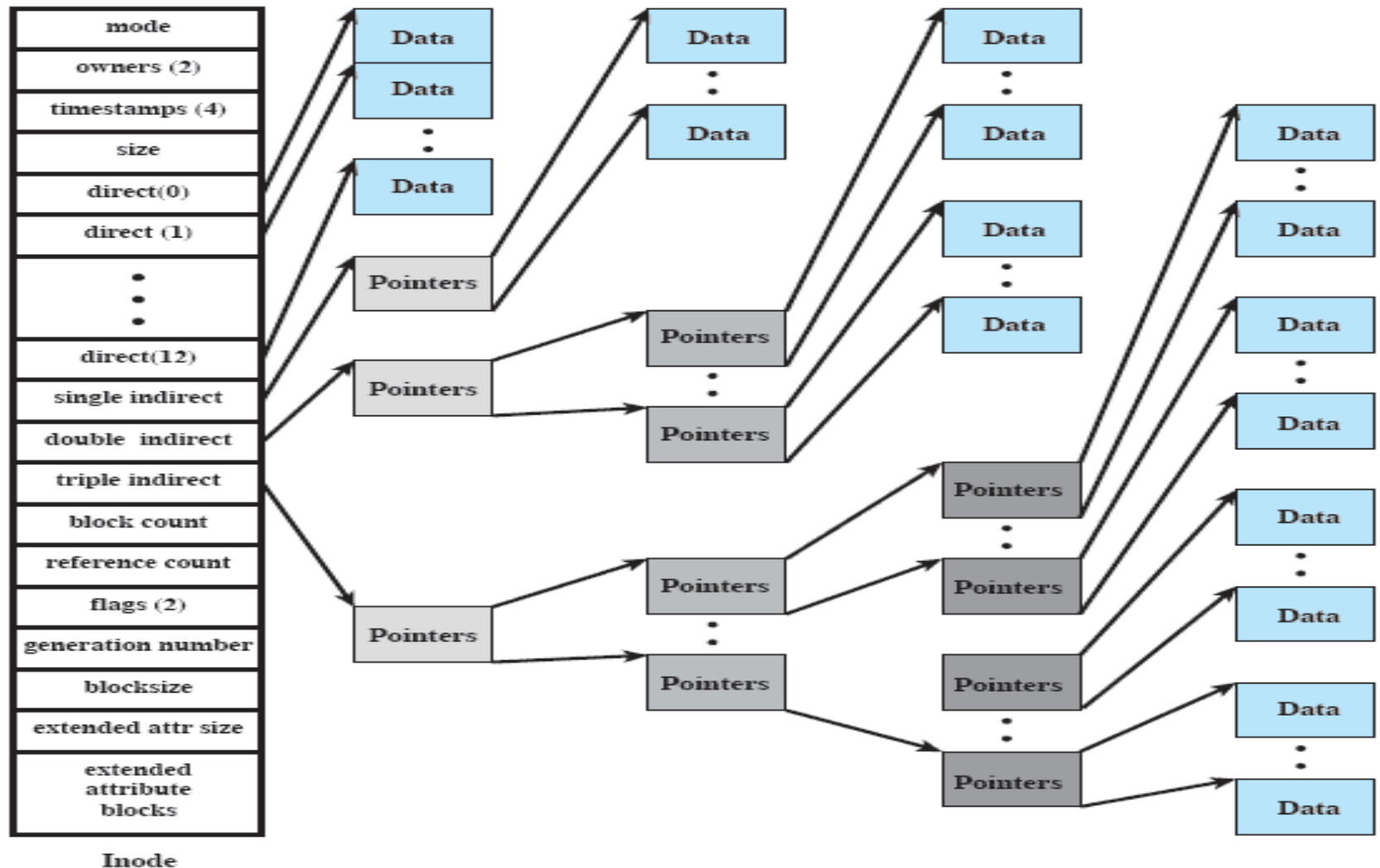


Inodes

- An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file



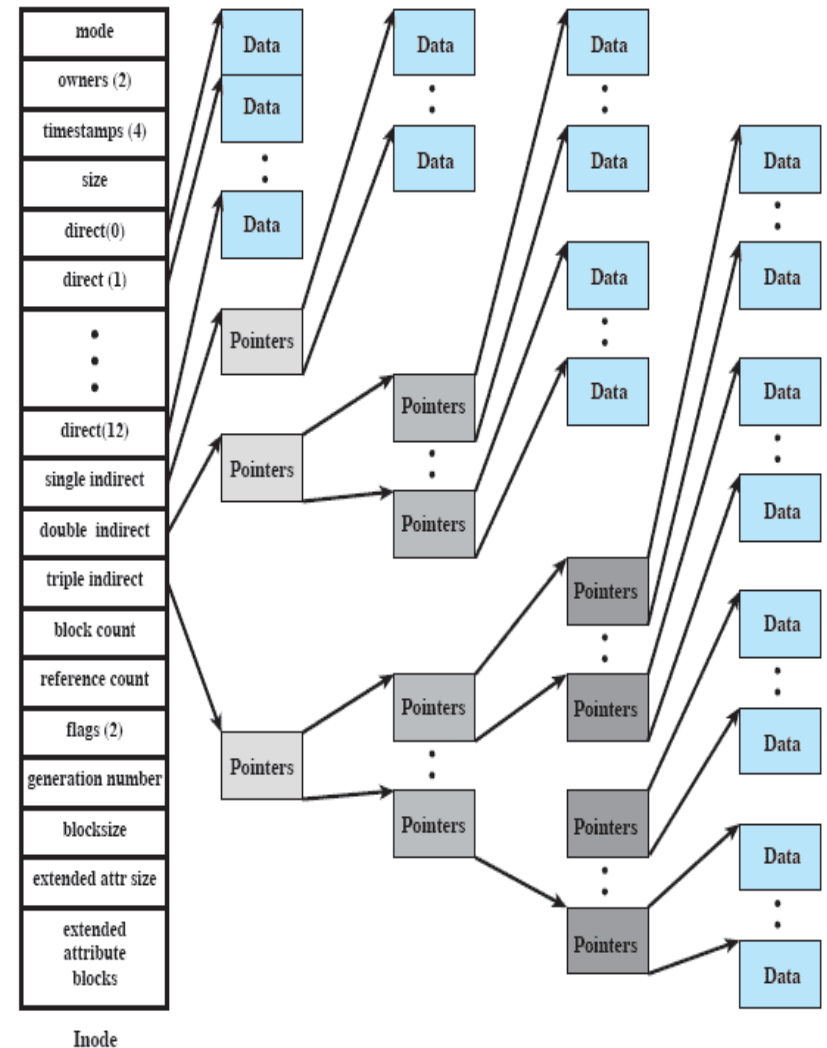
FreeBSD Inode and File Structure



FreeBSD Inode and File Structure

- The exact inode structure varies from one UNIX implementation to another.

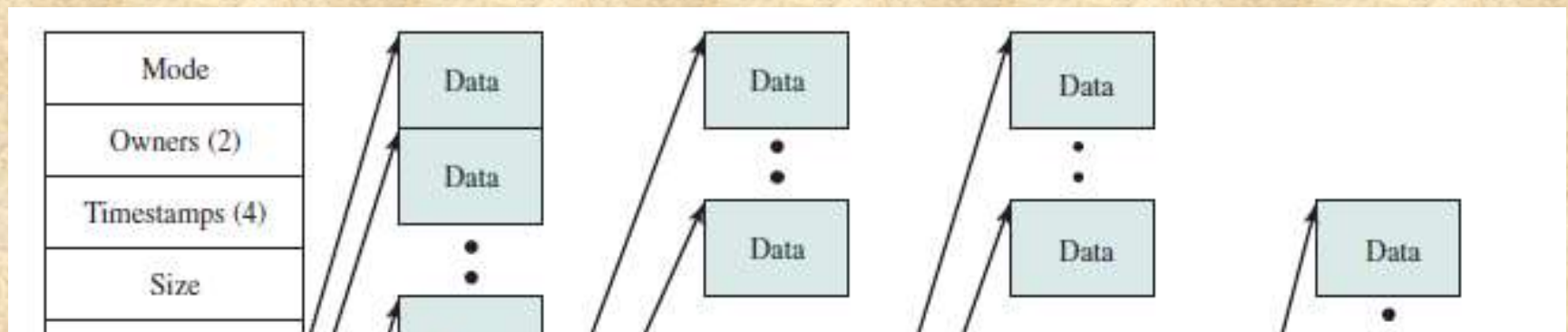
FreeBSD is a free and open-source Unix operating system



FreeBSD Inode and File Structure

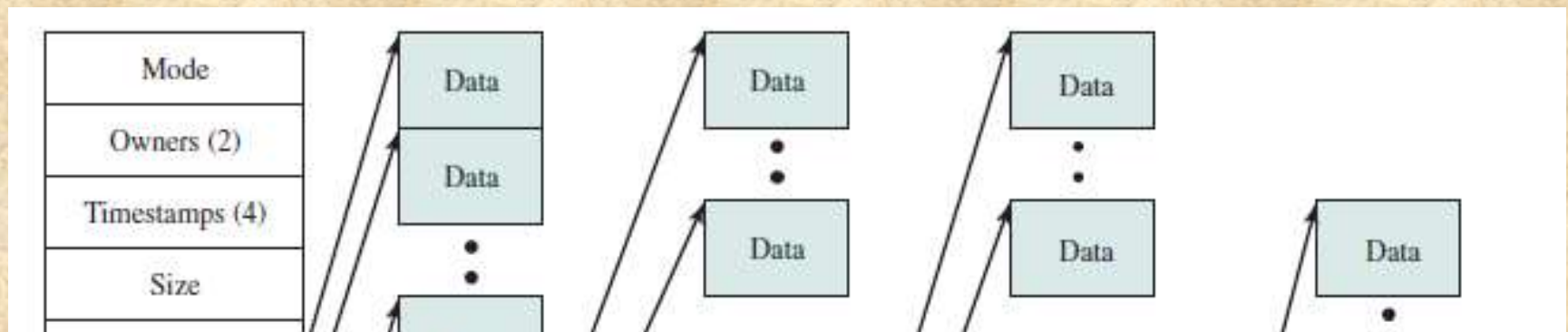
The FreeBSD inode structure, includes the following data elements:

- The type and access mode of the file
- The file's owner and group-access identifiers



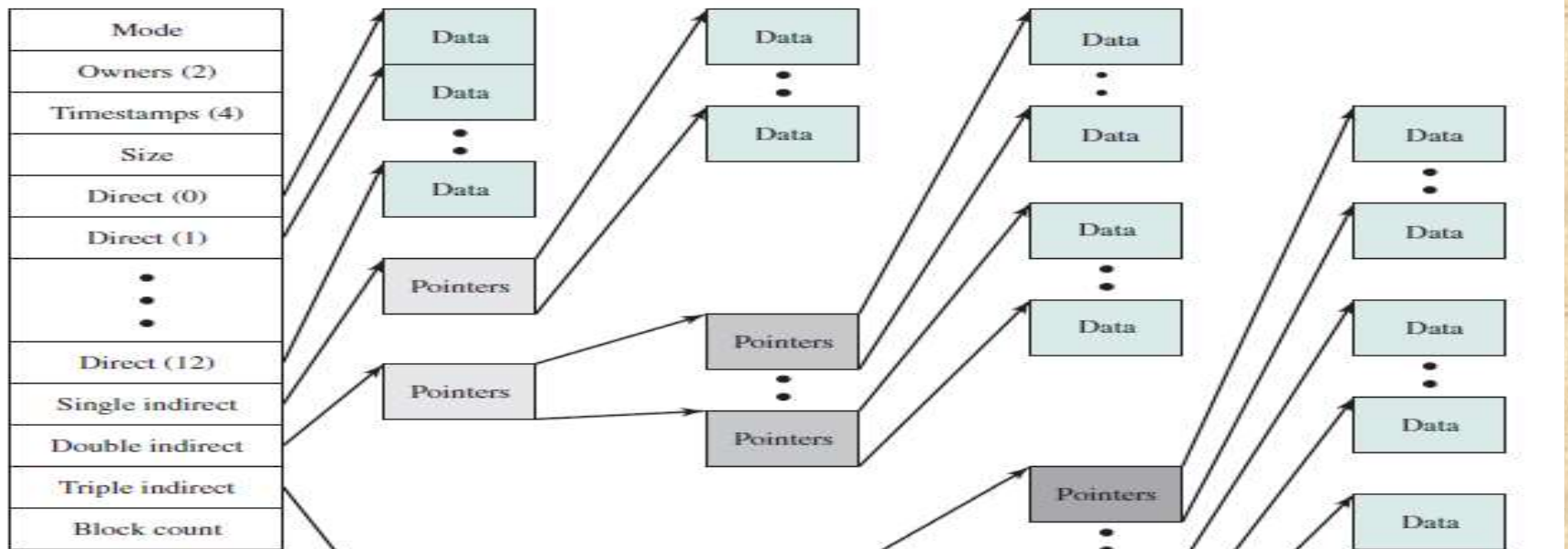
FreeBSD Inode and File Structure

- 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



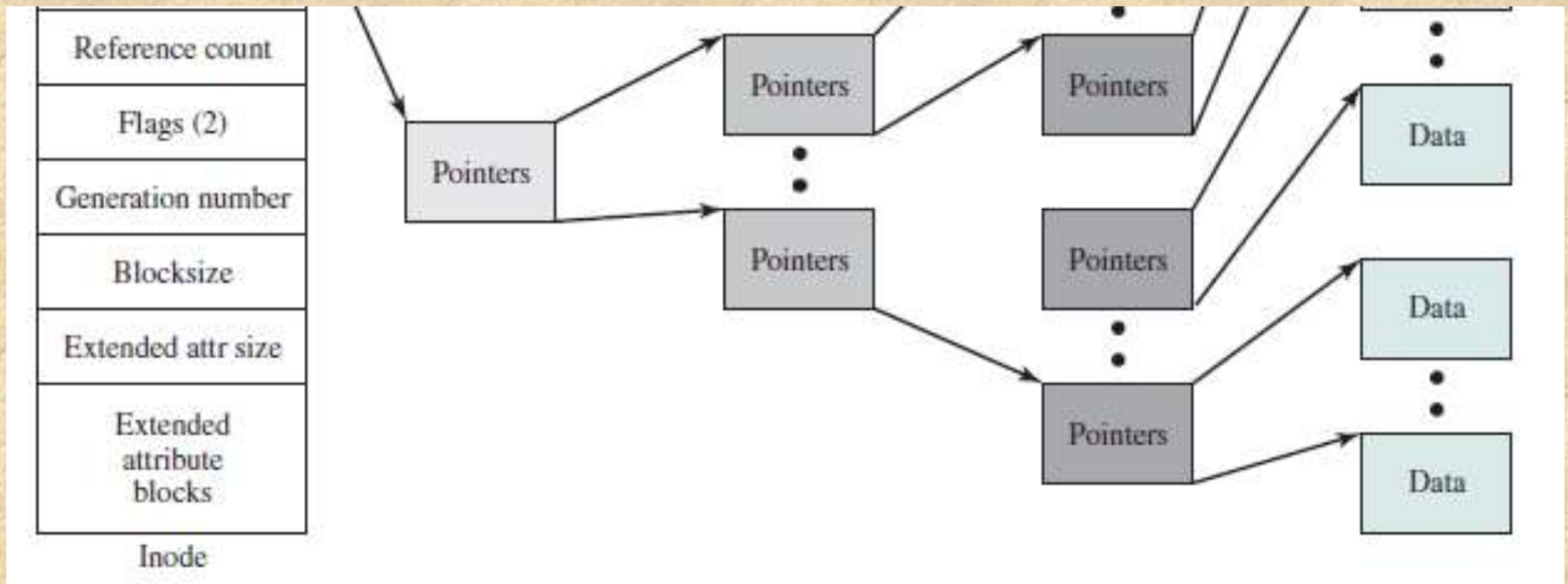
FreeBSD Inode and File Structure

- A sequence of block pointers
- The number of physical blocks used by the file, including blocks used to hold indirect pointers and attributes



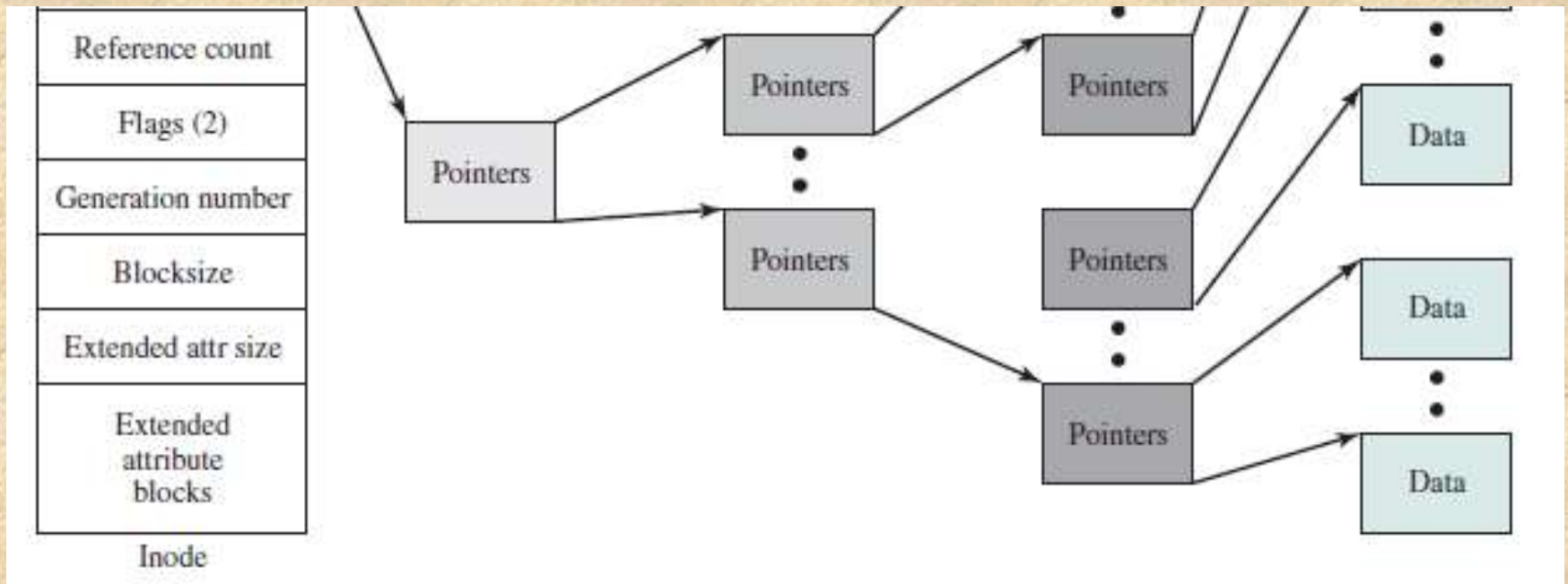
FreeBSD Inode and File Structure

- The number of directory entries that reference the file
- The kernel and user-settable flags that describe the characteristics of the file



FreeBSD Inode and File Structure

- 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 blocksize of the data blocks referenced by the inode

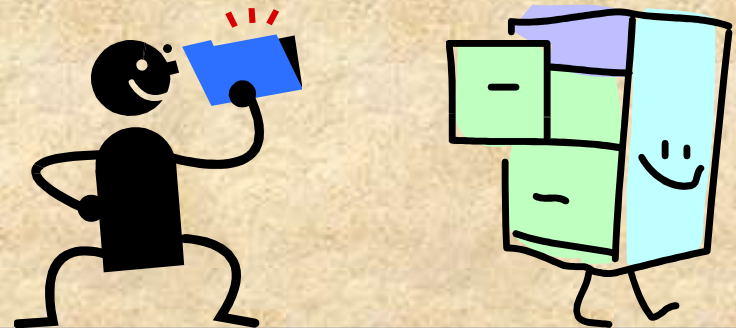


FreeBSD Inode and File Structure

- On the disk, there is an inode table, or inode list,
 - that contains the inodes of all the files in the file system.
- When a file is opened,
 - its inode is brought into main memory and stored in a memory-resident inode table.

File Allocation-Unix

- 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)



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

UNIX Directories

and Inodes

- A directory is simply a file that contains
- a list of file names plus pointers to associated inodes.

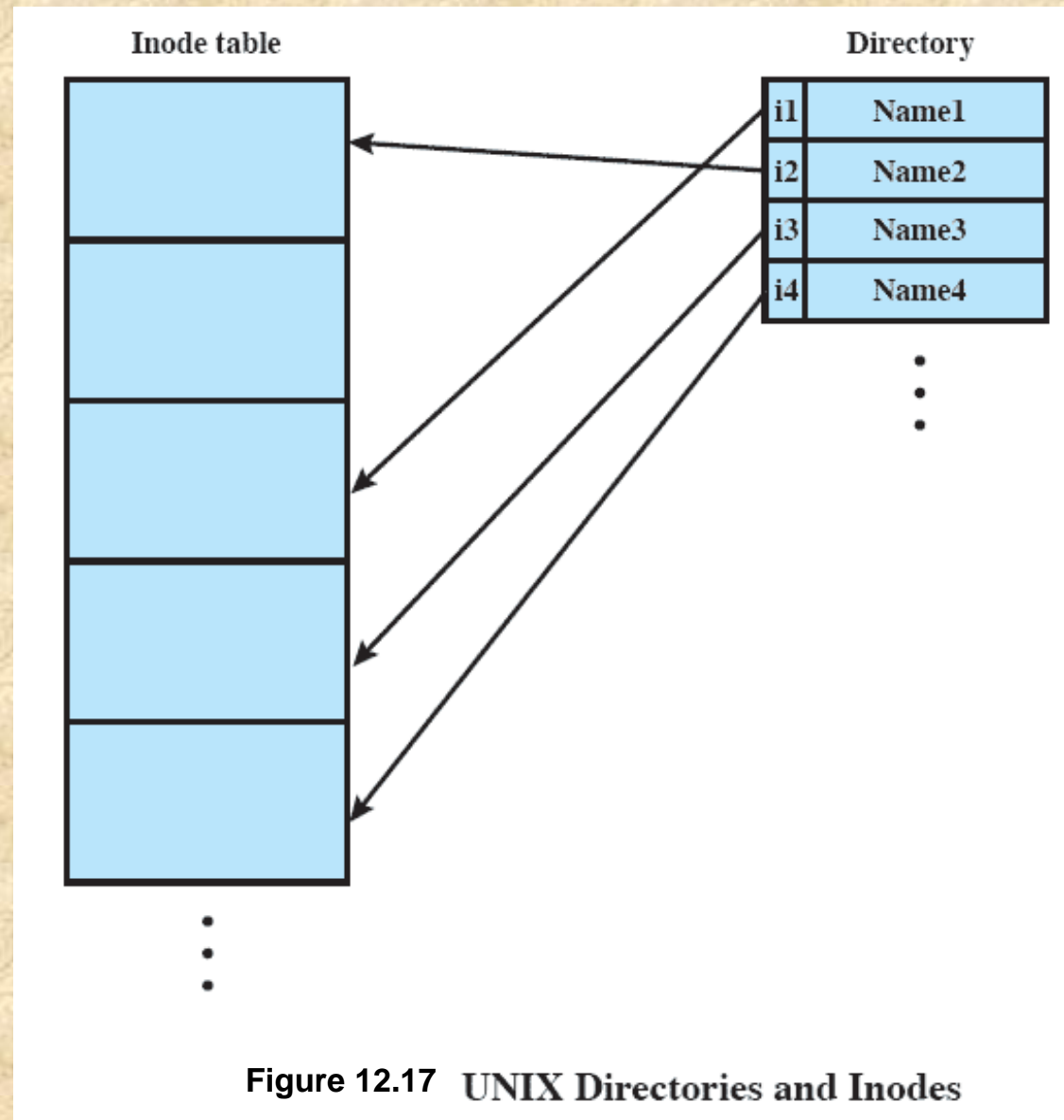


Figure 12.17 UNIX Directories and Inodes

UNIX Directories

and Inodes

- Each directory entry (dentry) contains a name for the associated file or subdirectory plus an integer called the i-number (index number).
- When the file or directory is accessed, its i-number is used as an index into the inode table.

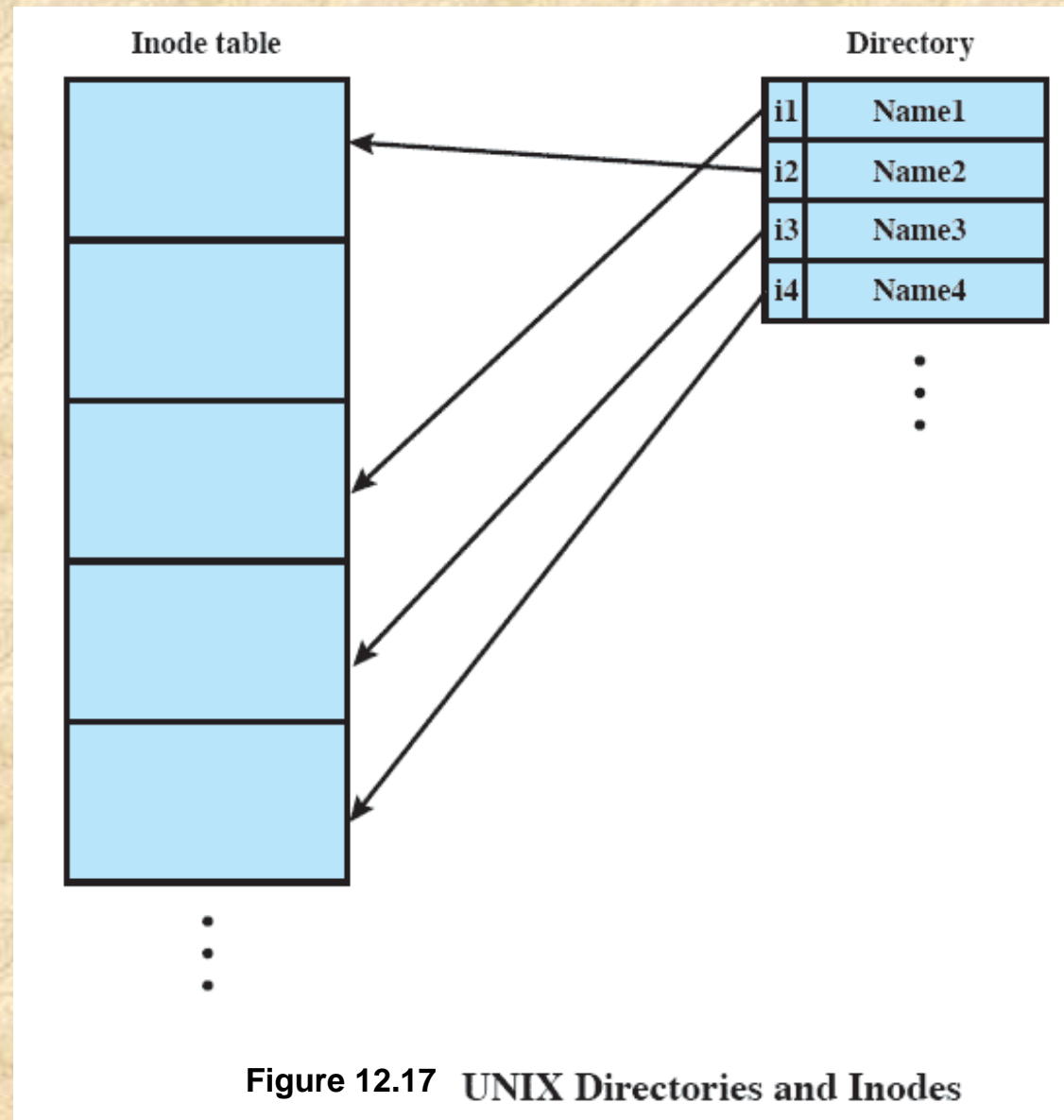
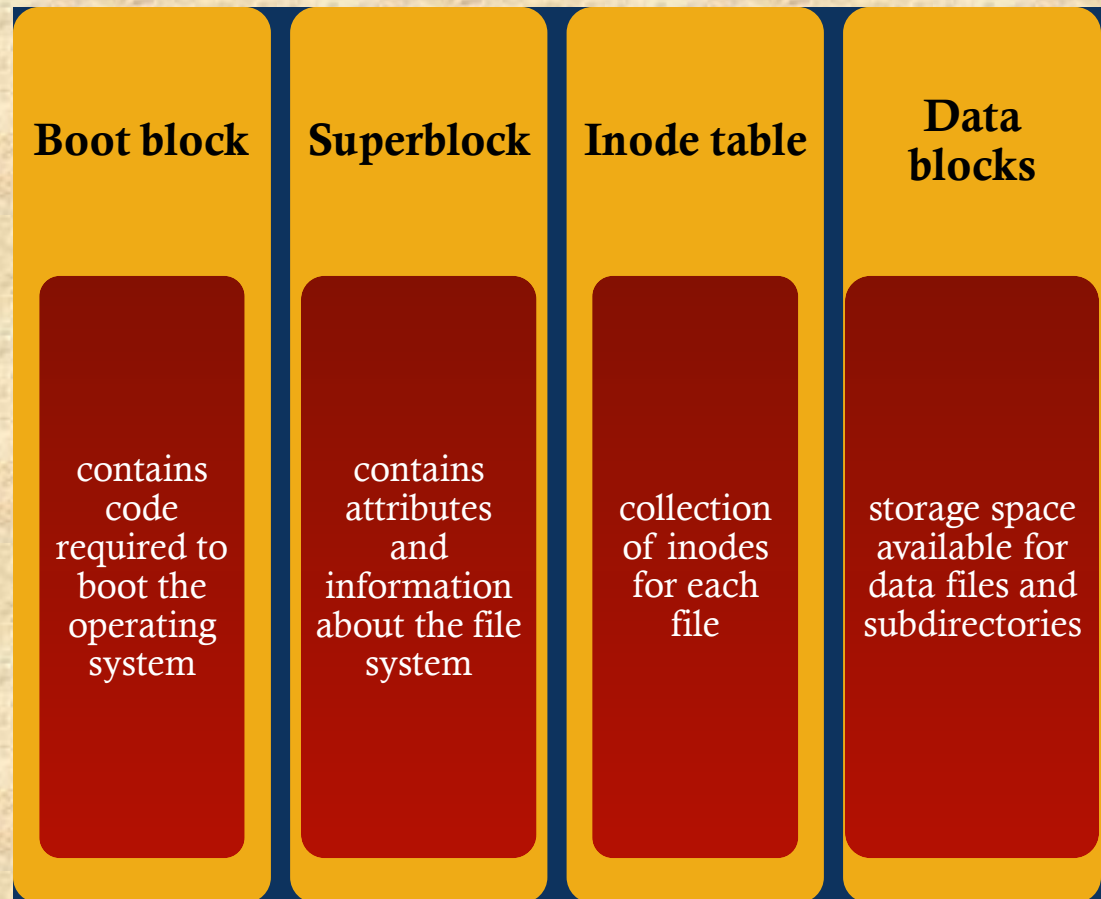


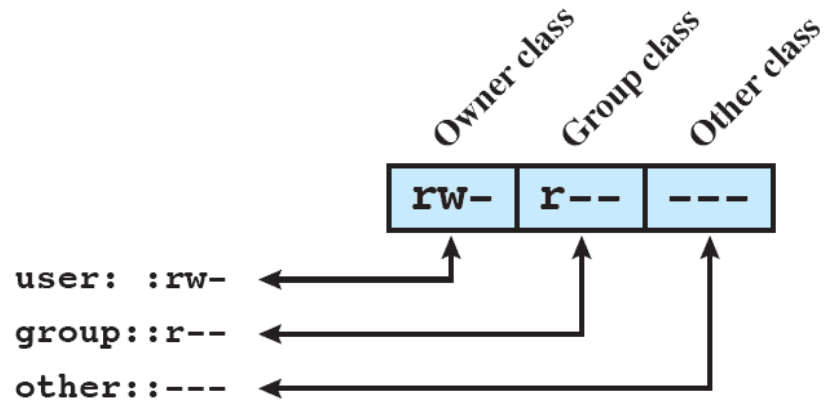
Figure 12.17 UNIX Directories and Inodes

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

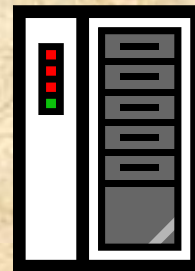


(a) Traditional UNIX approach (minimal access control list)



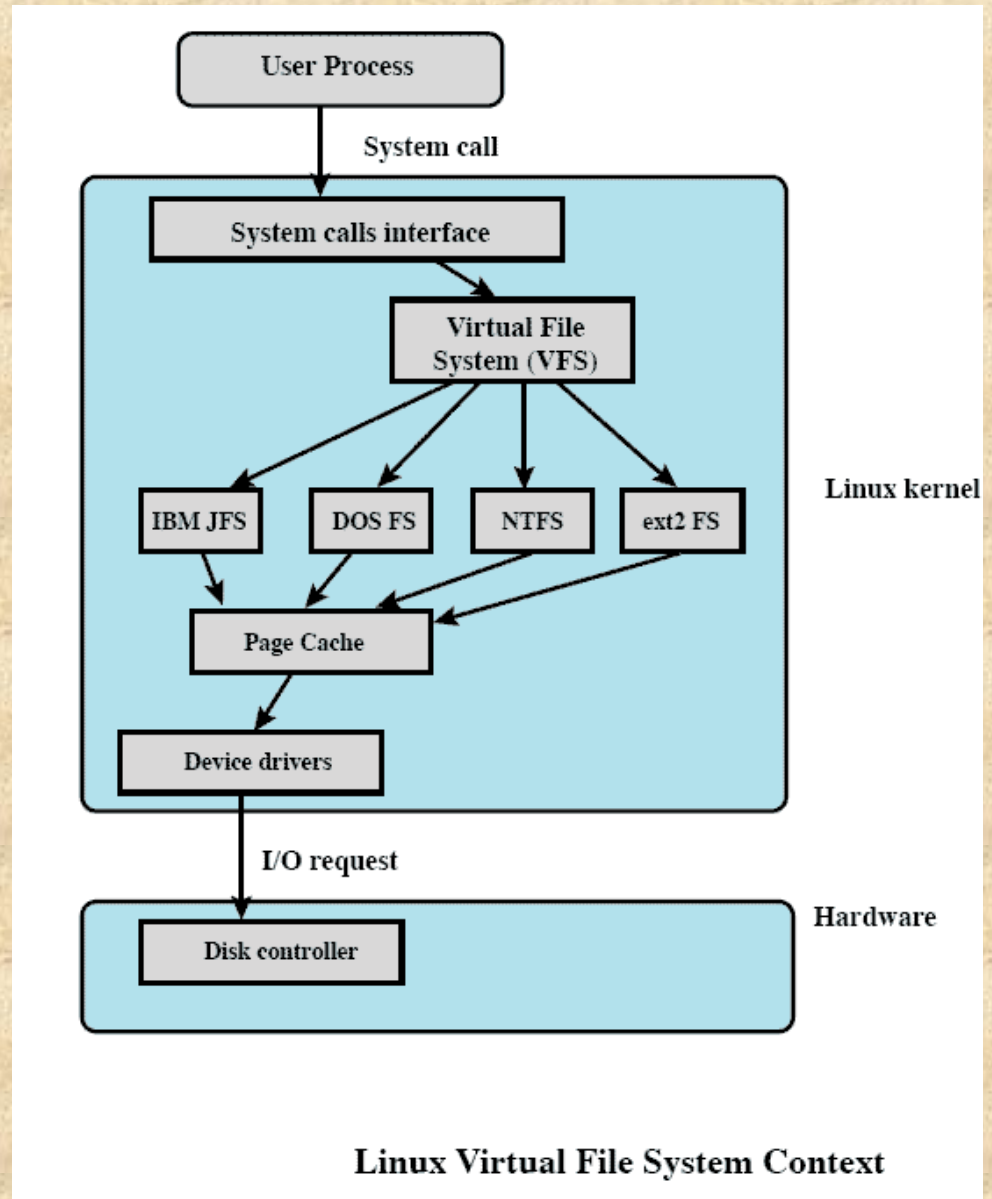
Access Control Lists in UNIX

- 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)

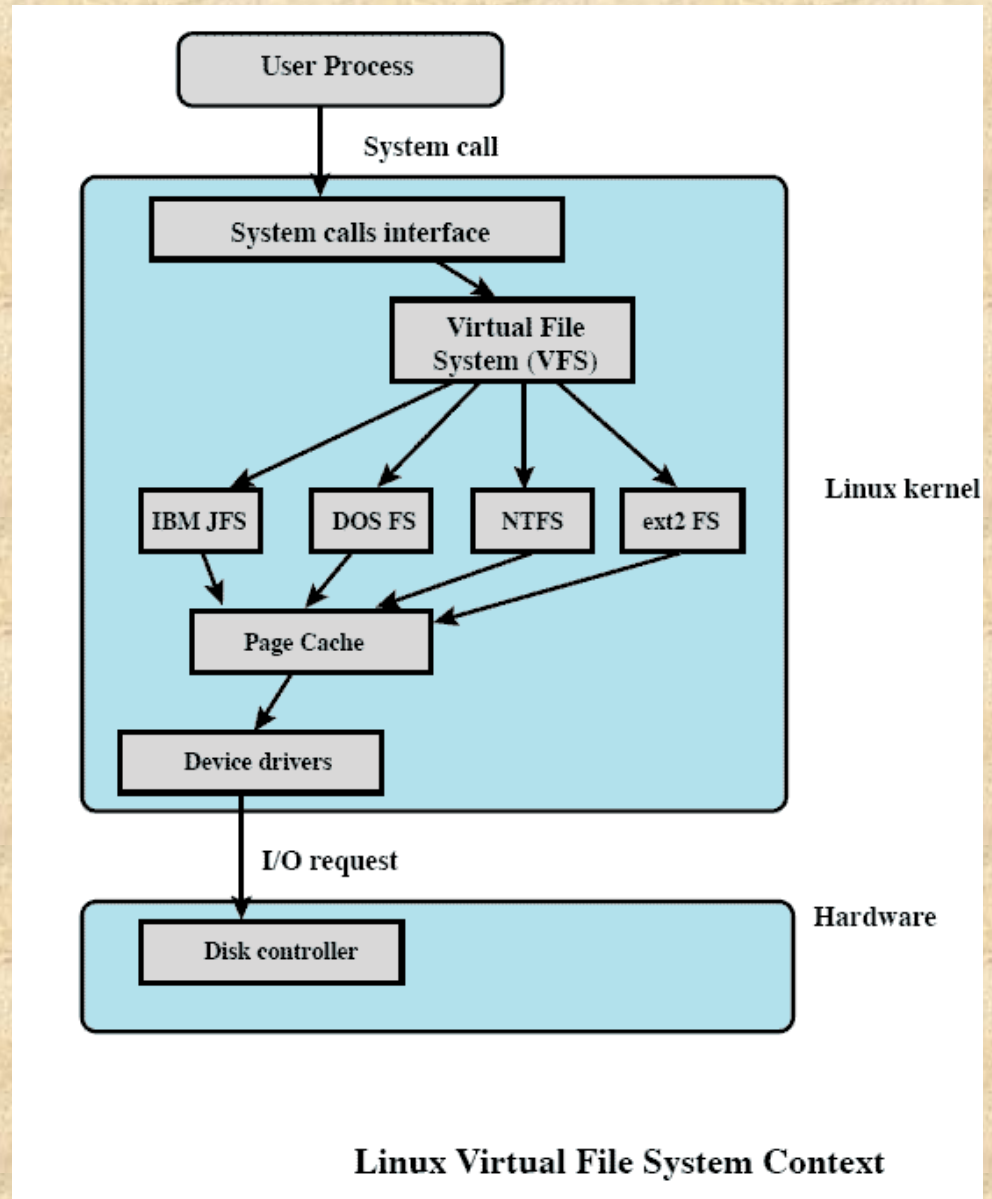
- Linux includes a versatile and powerful file-handling facility,
- Designed to support a wide variety of file management systems and file structures.
- Presents a single, uniform file system interface to user processes



Linux Virtual File

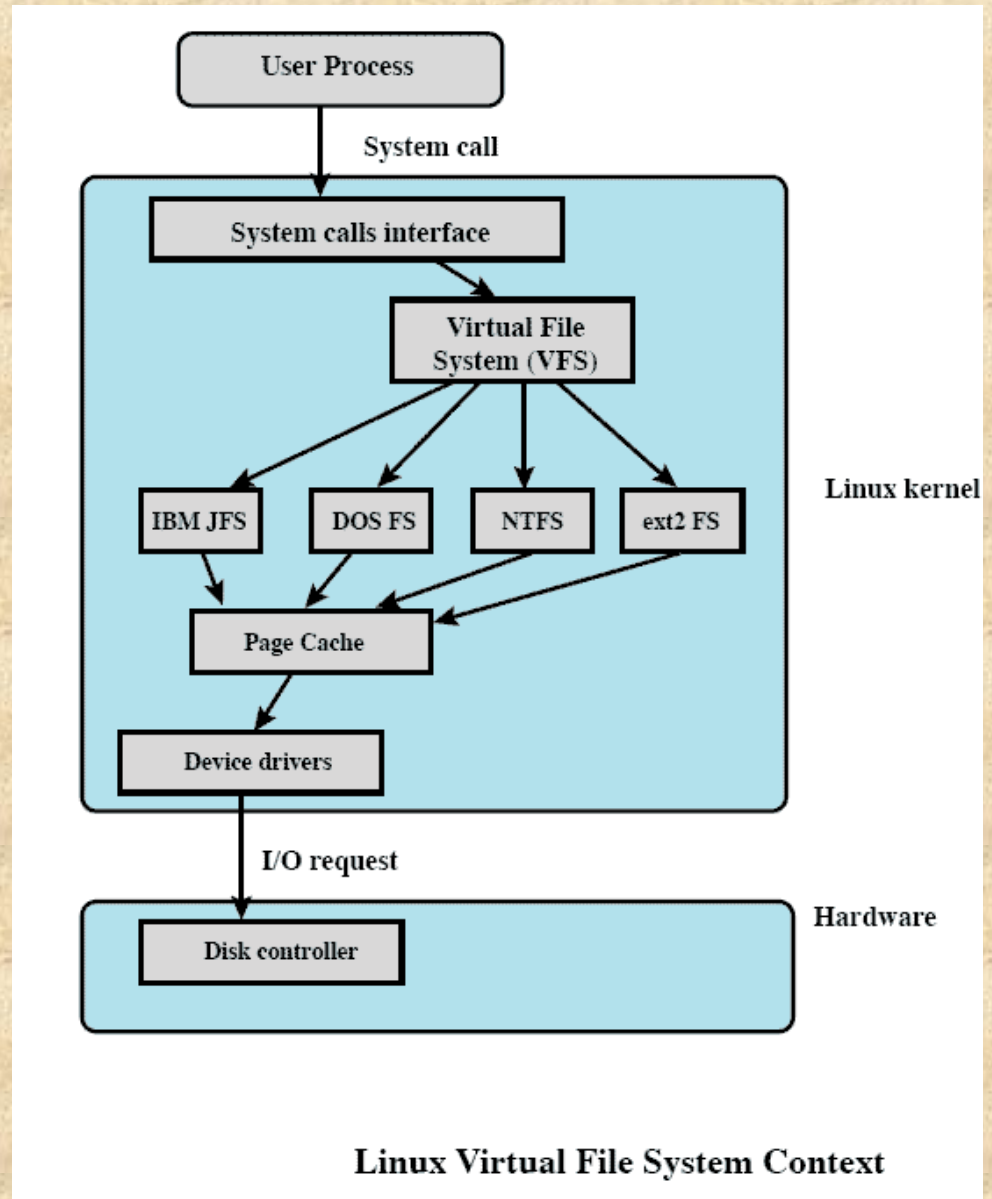
System (VFS)

- Linux includes a versatile and powerful file-handling facility,
- Designed to support a wide variety of file management systems and file structures.
- Presents a single, uniform file system interface to user processes



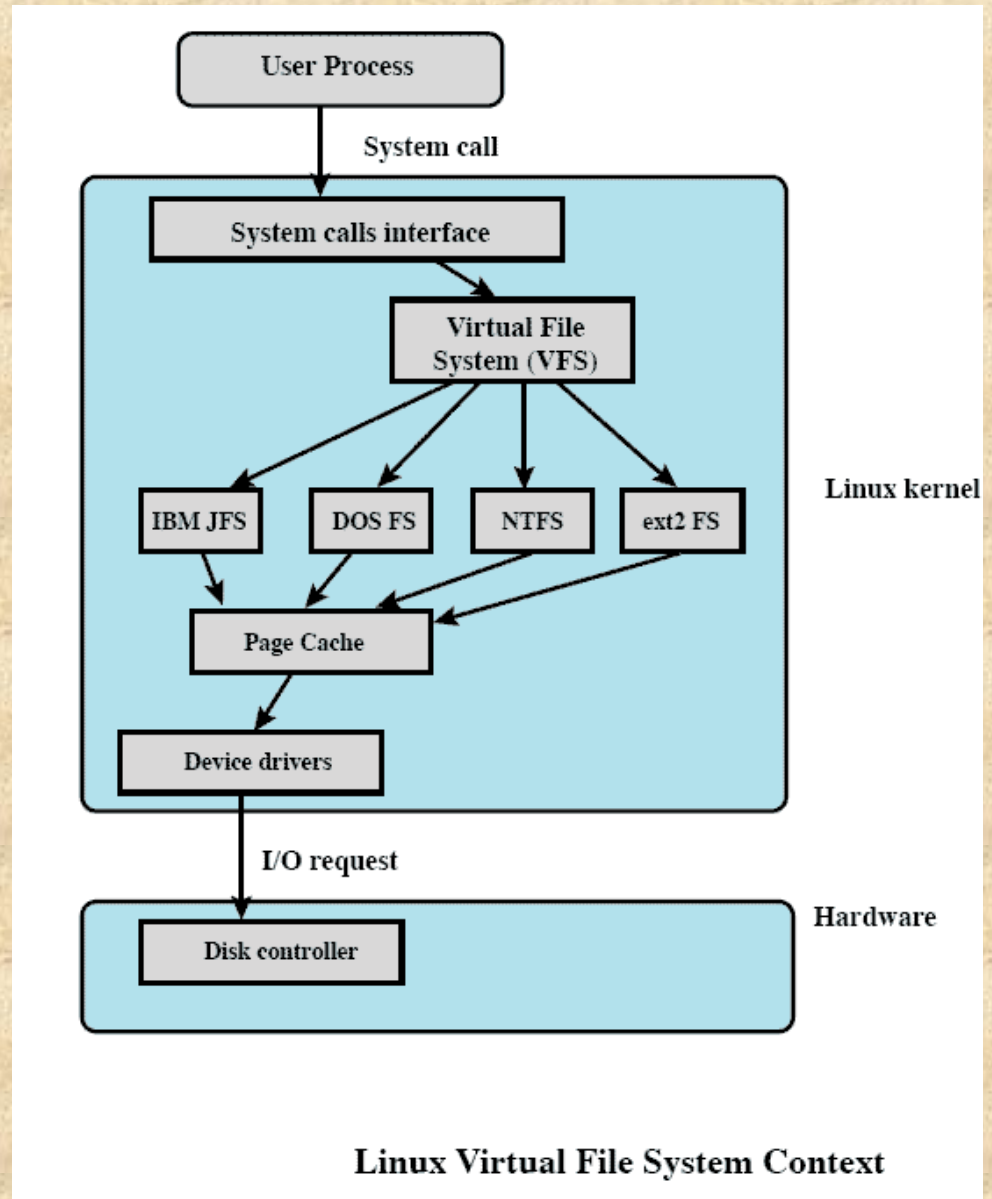
Linux Virtual File System (VFS)

- Defines a common file model that is
- capable of representing any conceivable file system's general feature and behavior



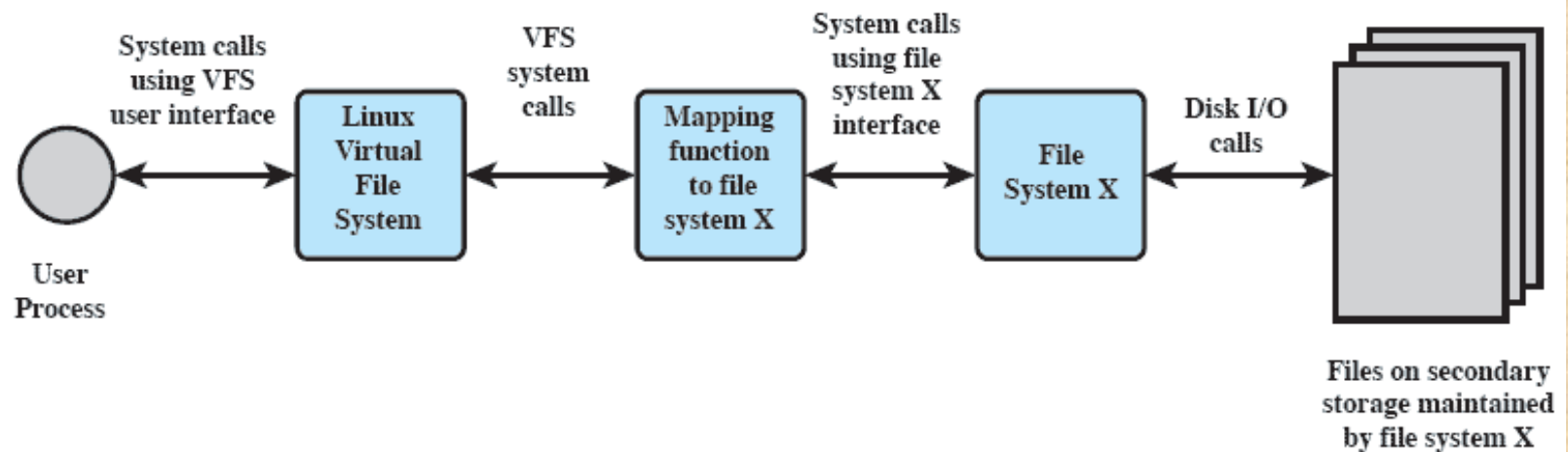
Linux Virtual File System (VFS)

- VFS Assumes files are objects that share basic properties regardless of the target file system or the underlying processor hardware



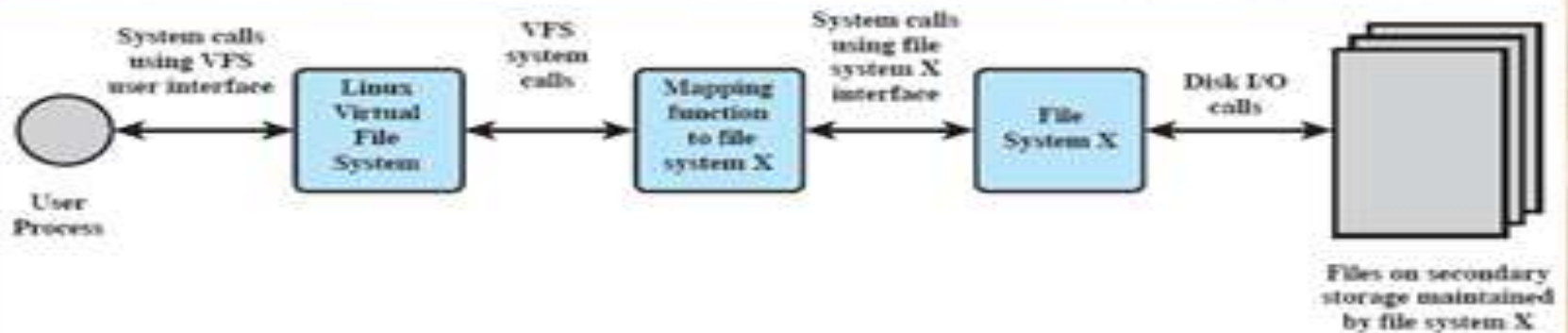
The Role of VFS

Within the Kernel



Linux Virtual File System Concept

The Role of VFS Within the Kernel



Linux Virtual File System Concept

- 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., IBM's Journaling File System (JFS)].
- The mapping function is simply a mapping of file system functional calls from one scheme to another.

Primary Object Types in VFS

- VFS is an object-oriented scheme.
- Because it is written in C, rather than a language that supports object programming (such as C++ or Java),
- VFS objects are implemented simply as C data structures.

Primary Object Types in VFS

Superblock Object

- represents a specific mounted file system

Inode Object

- represents a specific file



Dentry Object

- represents a specific directory entry

File Object

- represents an open file associated with a process