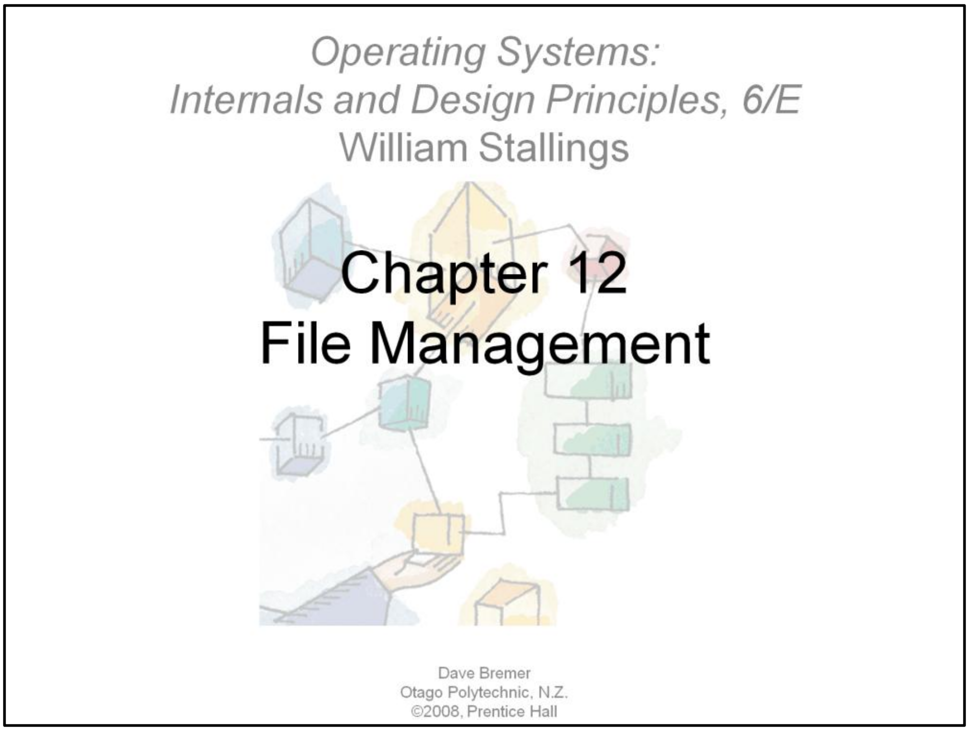
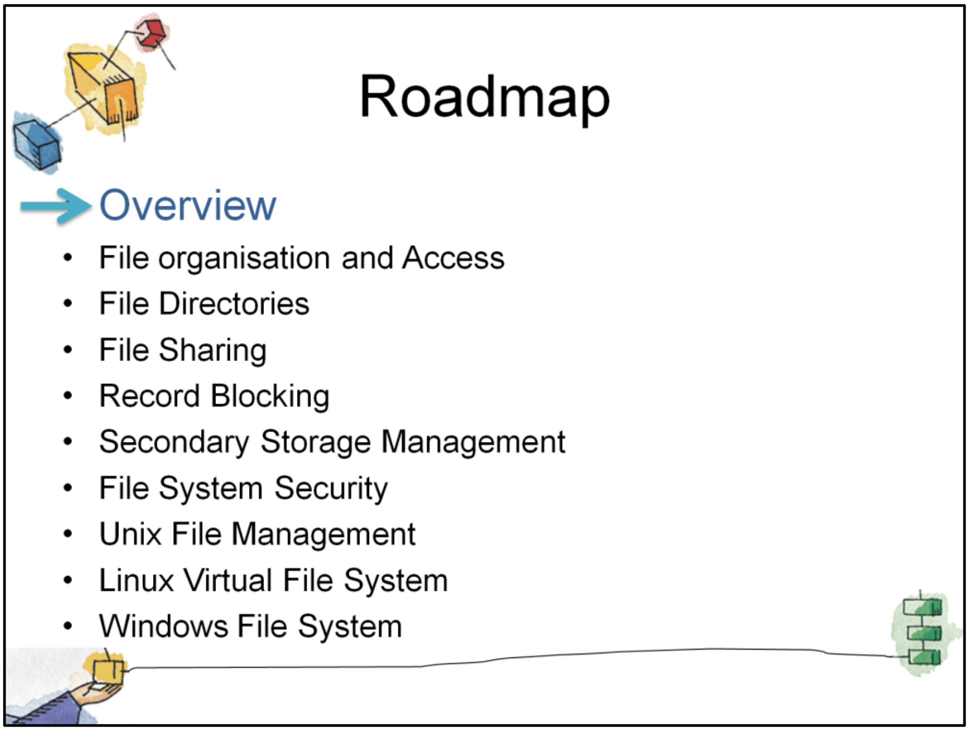


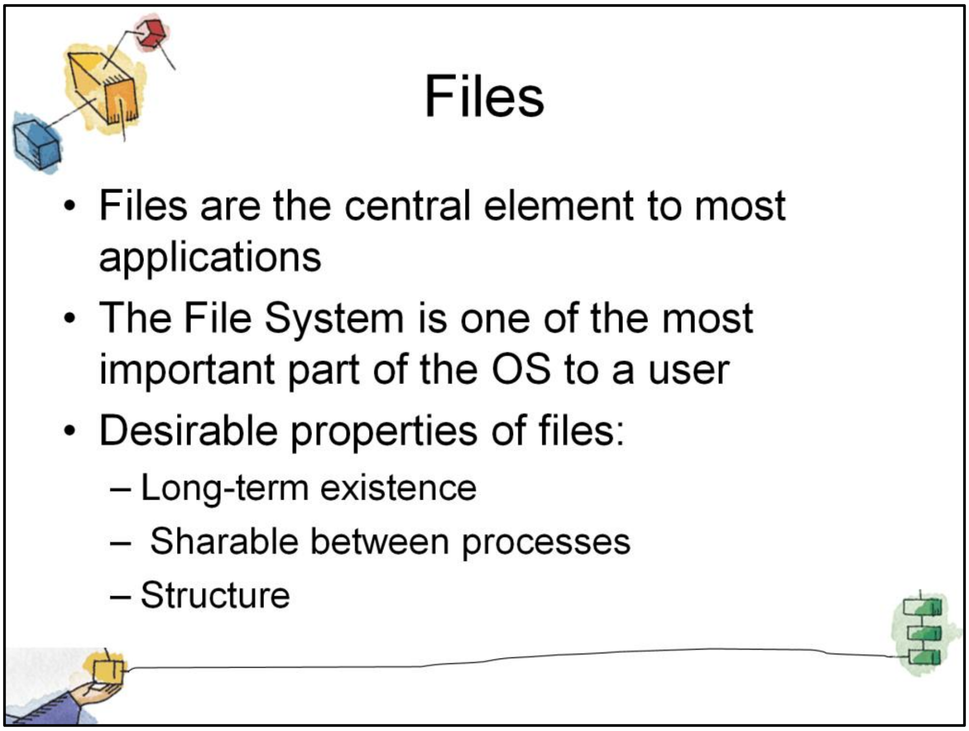
These slides are intended to help a teacher develop a presentation. This PowerPoint covers the entire chapter and includes too many slides for a single delivery. Professors are encouraged to adapt this presentation in ways which are best suited for their students and environment.



We begin with an overview, followed by a look at various file organization schemes.

Although file organization is generally beyond the scope of the operating system, it is essential to have a general understanding of the common alternatives to appreciate some of the design tradeoffs involved in file management.

The remainder of this chapter looks at other topics in file management.



In most applications, the file is the central element.

From the user’s point of view, one of the most important parts of an operating system is the file system.

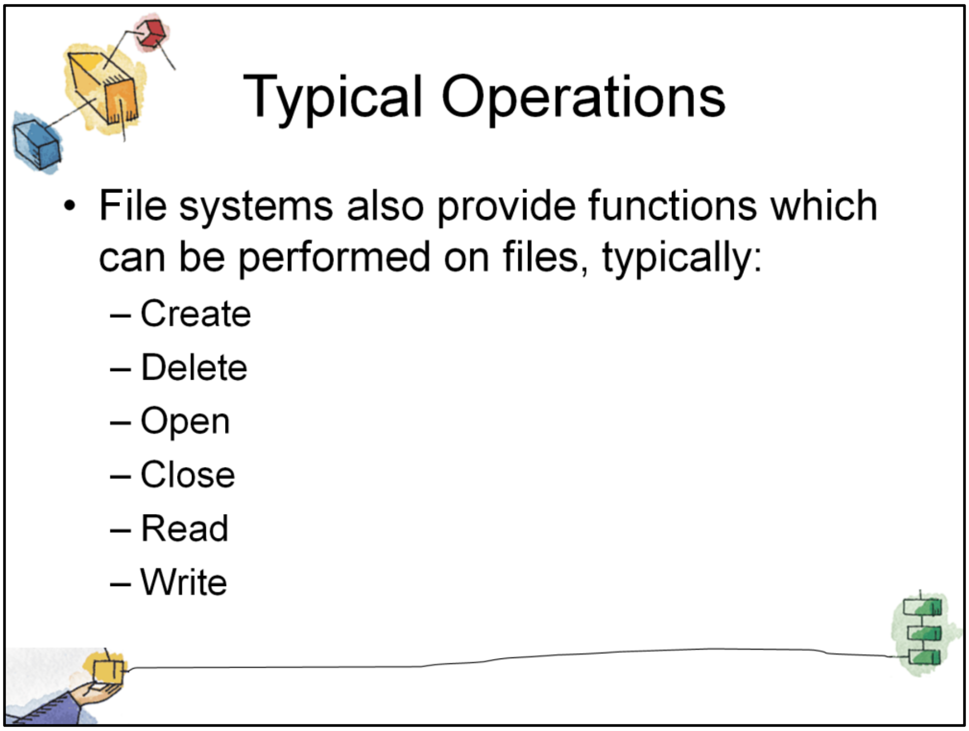
- The file system provides the resource abstractions typically associated with secondary storage.

Desirable properties include

- **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 on the file system, a file can have an internal structure that is convenient for particular applications. In addition, files can be organized into hierarchical or more complex structure to reflect the relationships among files.

File Management

- File management system consists of system utility programs that run as privileged applications
- Concerned with secondary storage



Any file system provides not only a means to store data organized as files, but a collection of functions that can be performed on files.

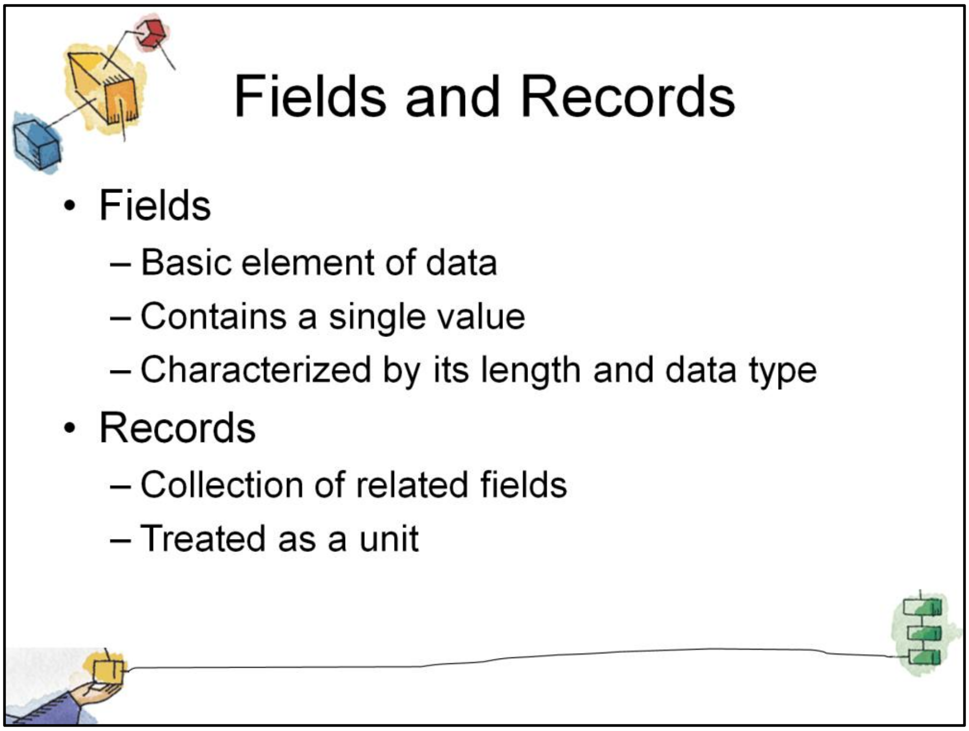
Typical operations include the following:

- **Create:** A new file is defined and positioned within the structure of files.
- **Delete:** A file is removed from the file structure and destroyed.
- **Open:** An existing file is declared to be “opened” by a process, allowing the process to perform functions on the file.
- **Close:** The file is closed with respect to a process, so that the process no longer may perform functions on the file, until the process opens the file again.
- **Read:** A process reads all or a portion of the data in a file.
- **Write:** A process updates a file, either by adding new data that expands the size of the file or by changing the values of existing data items in the file.

Terms

- Four terms are in common use when discussing files:
  - Field
  - Record
  - File
  - Database

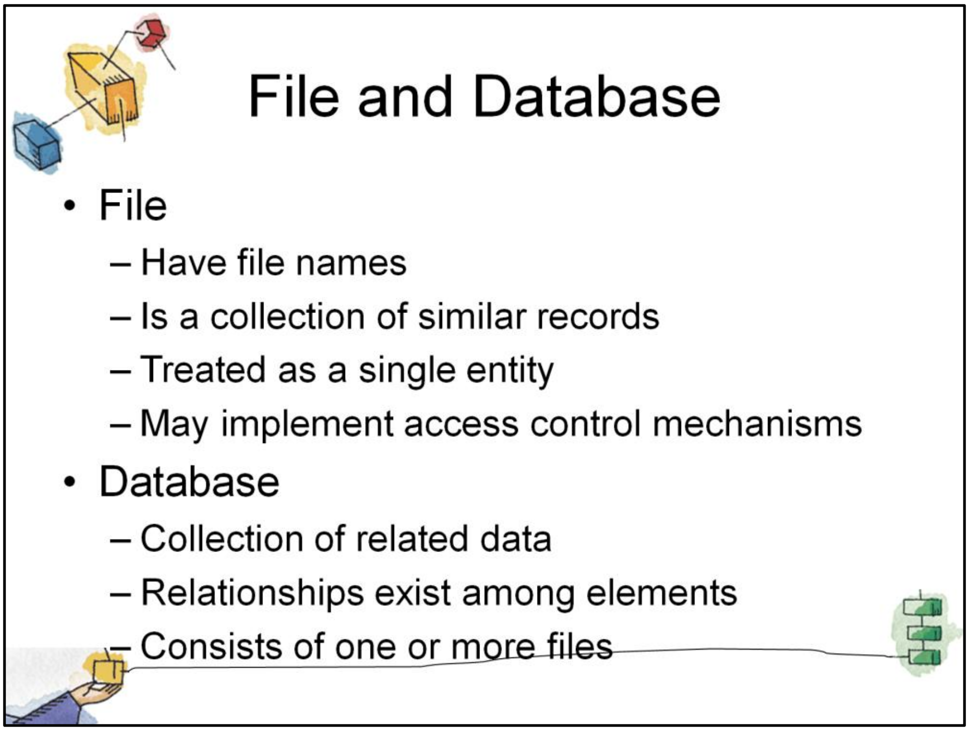
Discussed on following slides



A field is the basic element of data.

- It is characterized by its length and data type (e.g. ASCII string, decimal).
- Depending on the file design, fields may be fixed length or variable length.

A record is a collection of related fields that can be treated as a unit by some application program.

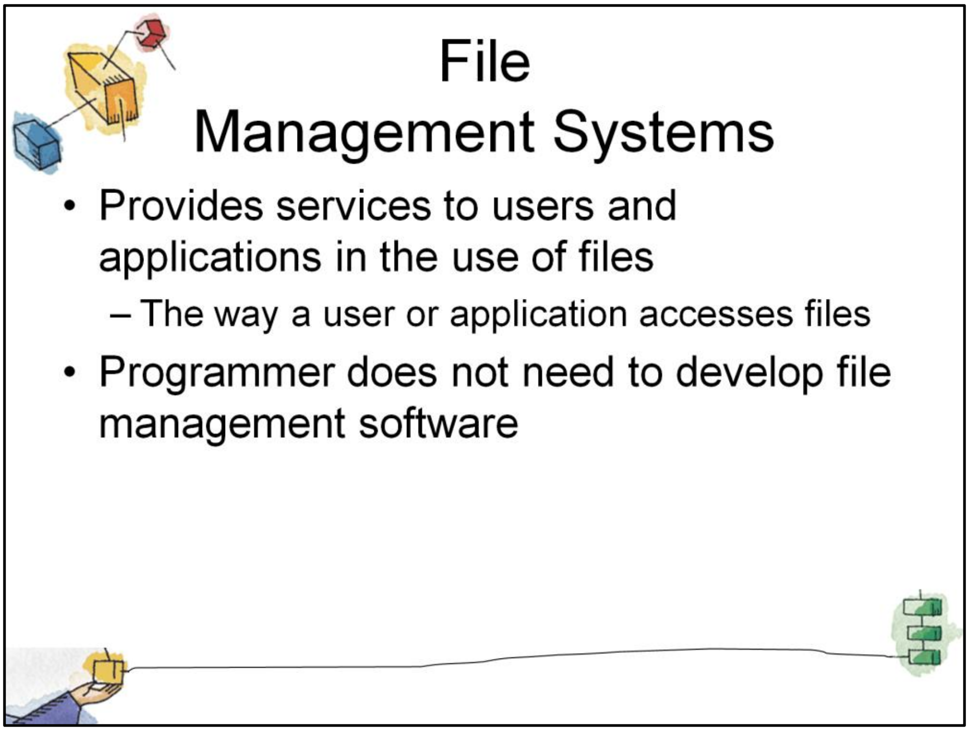


A file is a collection of similar records.

- The file is treated as a single entity by users and applications and may be referenced by name.
- Files have file names and may be created and deleted.
- Access control restrictions usually apply at the file level.

A database is a collection of related data.

- Explicit relationships exist among elements of data
- The database itself consists of one or more types of files.
- Usually, there is a separate database management system that is independent of the operating system, although that system may make use of some file management programs.

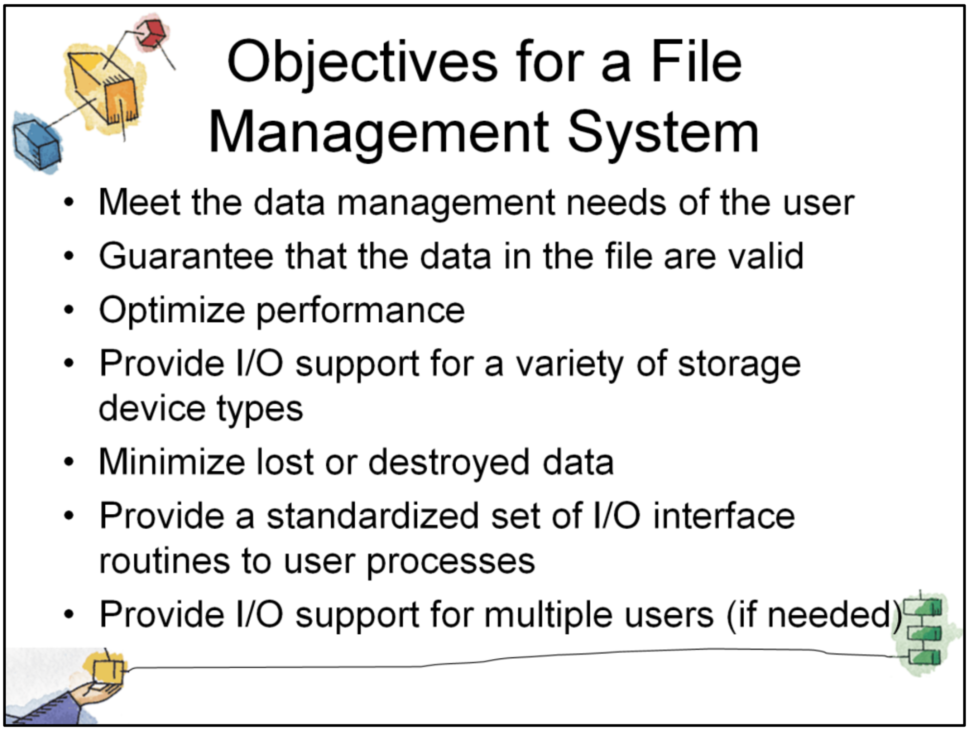


A file management system is the set of system software that provides services to users and applications in the use of files.

- Typically, the only way that a user or application may access files is through the file management system.

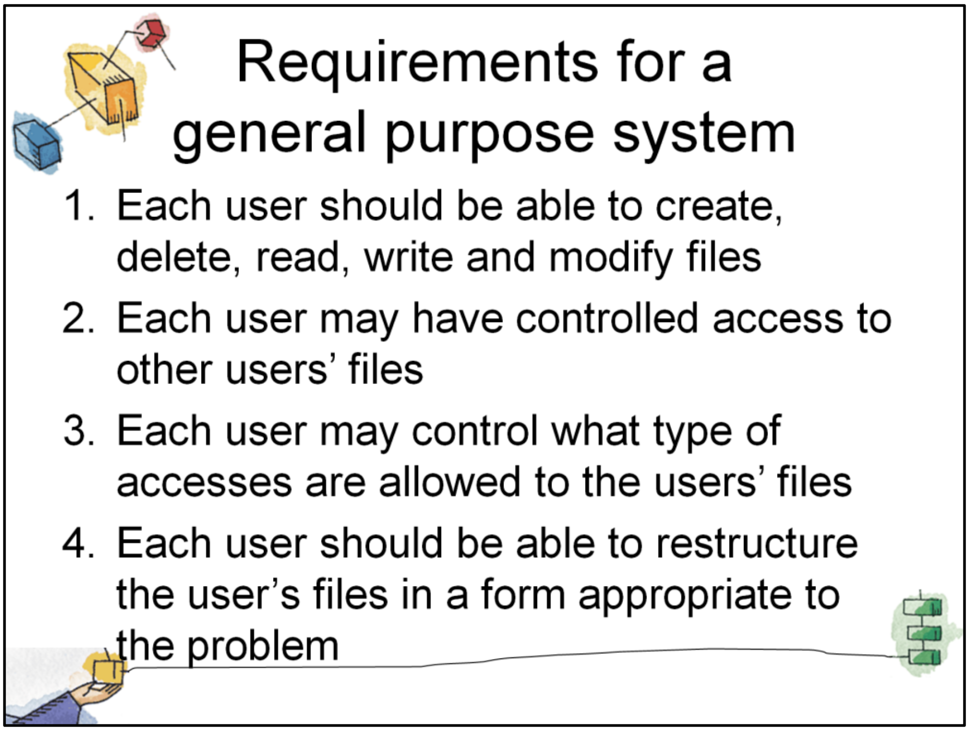
This relieves the user or programmer of the necessity of developing special-purpose software for each application

- and provides the system with a consistent, well-defined means of controlling its most important asset.



Objectives include:

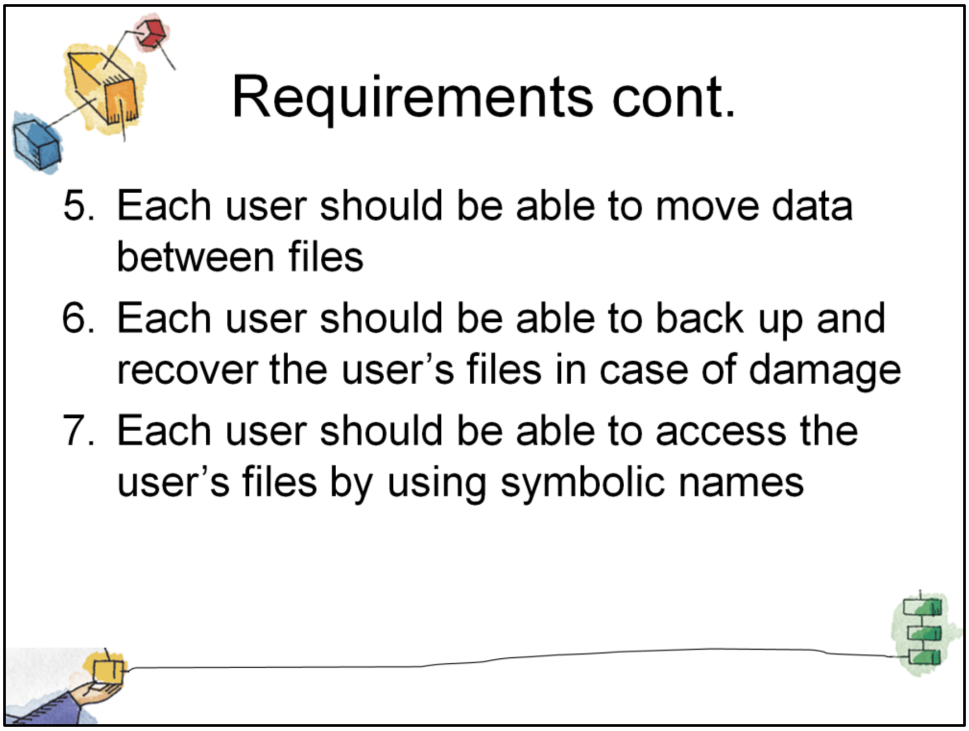
- To meet the data management needs and requirements of the user,
- To guarantee, to the extent possible, that the data in the file are valid
- To optimize performance, both from the system point of view in terms of over-all throughput and from the user's point of view in terms of response time
- To provide I/O support for a variety of storage device types
- To minimize or eliminate the potential for lost or destroyed data
- To provide a standardized set of I/O interface routines to user processes
- To provide I/O support for multiple users, in the case of multiple-user systems



For an interactive, general-purpose system, the following constitute a minimal set of requirements:

1. Each user should be able to create, delete, read, write, and modify files.
2. Each user may have controlled access to other users' files.
3. Each user may control what types of accesses are allowed to the user's files.
4. Each user should be able to restructure the user's files in a form appropriate to the problem.

5,6, and 7 on next slide



5. Each user should be able to move data between files.

6. Each user should be able to back up and recover the user's files in case of damage.

7. Each user should be able to access his or her files by name rather than by numeric identifier.

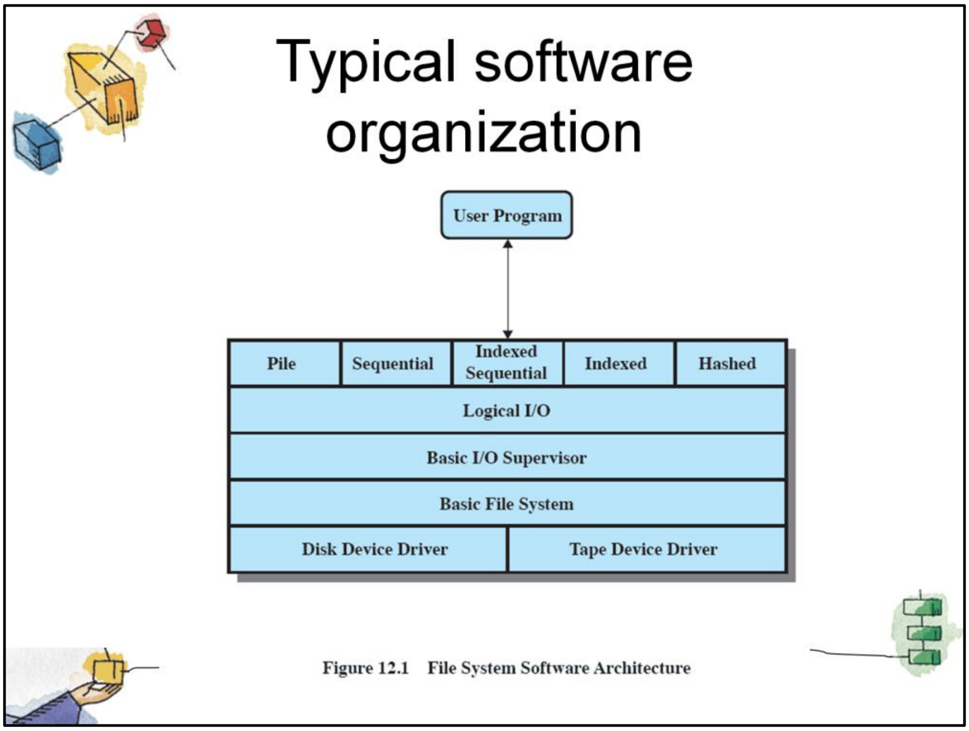
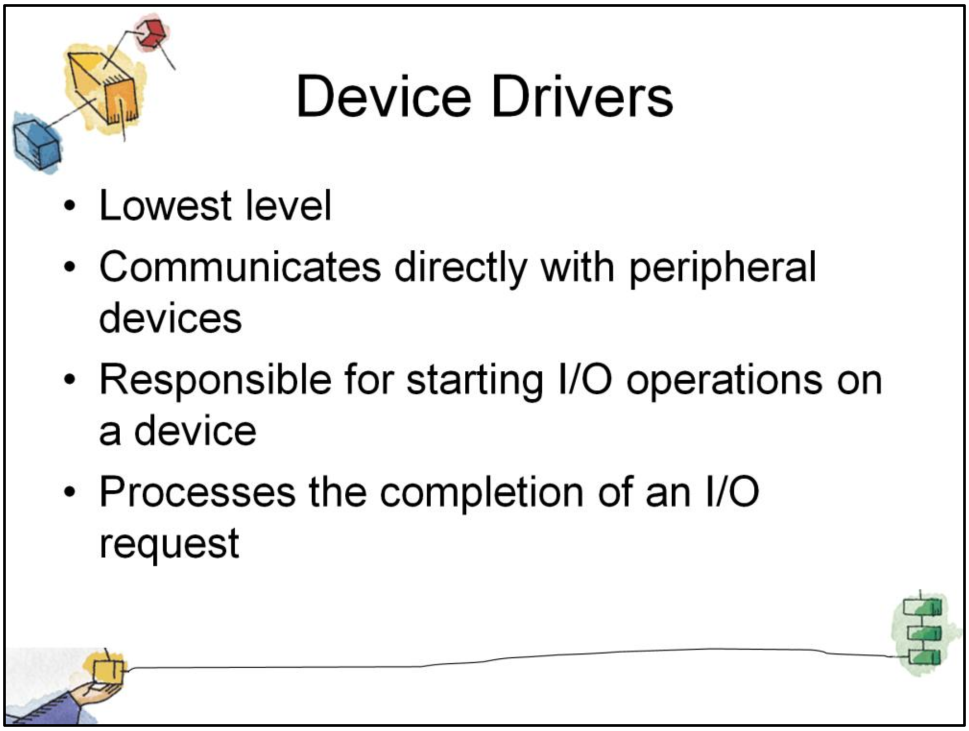


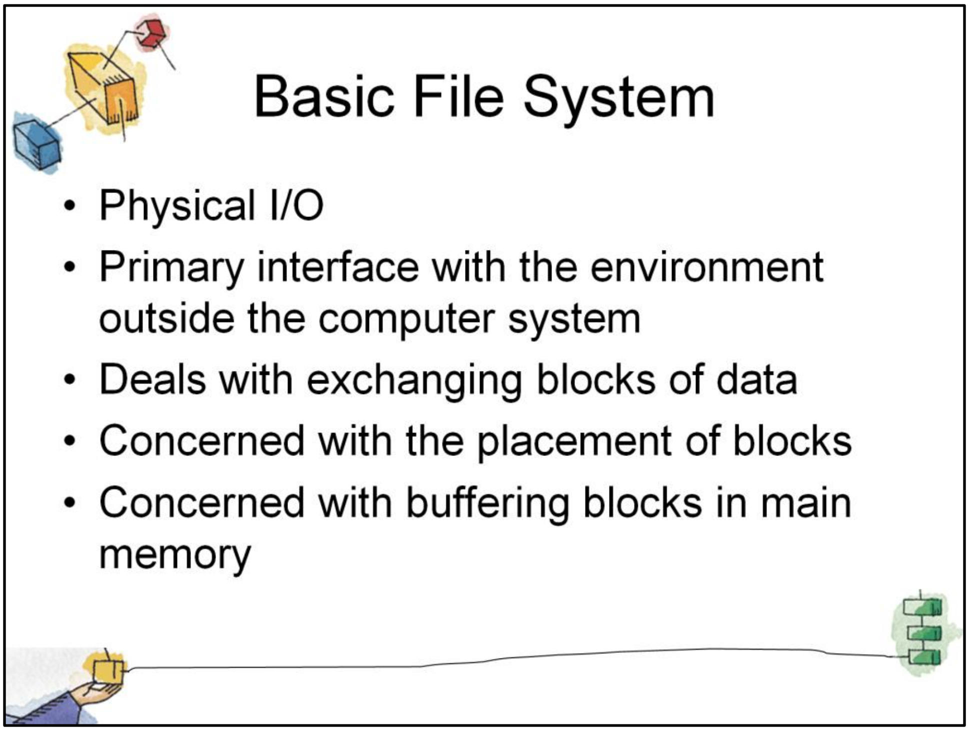
Figure 12.1 File System Software Architecture

Variations will exist between systems but typically have the aspects described above and in the following slides



device drivers communicate at the lowest level directly with peripheral devices or their controllers or channels.

- A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request.
- For file operations, the typical devices controlled are disk and tape drives.
- Device drivers are usually considered to be part of the operating system.

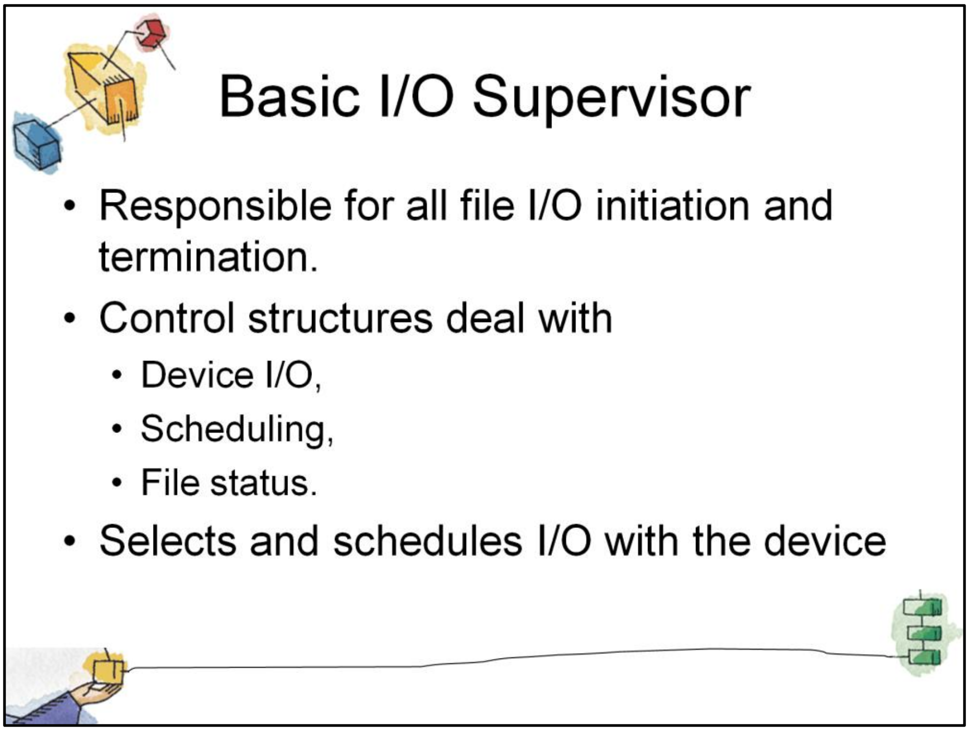


basic file system, or **the physical I/O level.**

This is the primary interface with the environment outside of the computer system.

It deals with blocks of data that are exchanged with disk or tape systems.

- It is concerned with the placement of those blocks on the secondary storage device and on the buffering of those blocks in main memory.
- It does not understand the content of the data or the structure of the files involved.
- The basic file system is often considered part of the operating system.



Basic I/O supervisor is responsible for all file I/O initiation and termination.

Control structures are maintained that deal with

- device I/O,
- scheduling, and
- file status.

The basic I/O supervisor selects the device on which file I/O is to be performed, based on the particular file selected.

It is also concerned with scheduling disk and tape accesses to optimize performance.

- I/O buffers are assigned and secondary memory is allocated at this level.

The basic I/O supervisor is 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

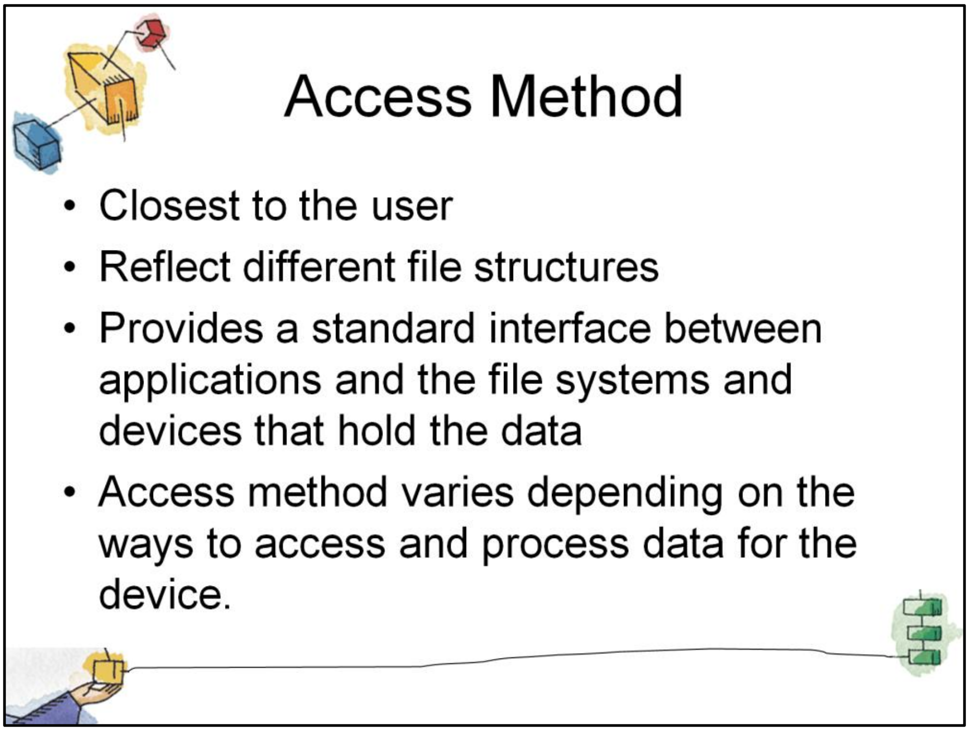
Logical I/O enables users and applications to access records.

Whereas the basic file system deals with blocks of data,

- the logical I/O module deals with file records.

Logical I/O provides a general-purpose record I/O capability

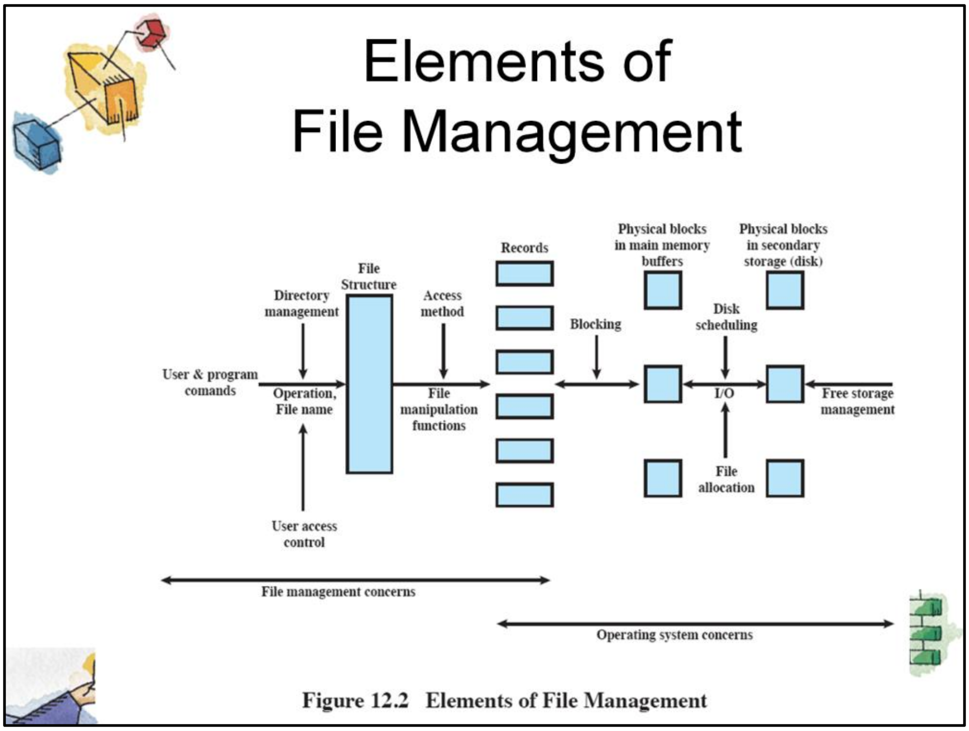
- and maintains basic data about files.



the access method is the level of the file system closest to the user.

It 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.



Users and application programs interact with the file system by means of commands for creating and deleting files and for performing operations on files.

Before performing any operation, the file system must identify and locate the selected file.

- This requires the use of some sort of directory that serves to describe the location of all files, plus their attributes.
- In addition, most shared systems enforce user access control

The basic operations that a user or application may perform on a file are performed at the record level.

- The user or application views the file as having some structure that organizes the records, such as a sequential structure

The secondary storage must be managed.

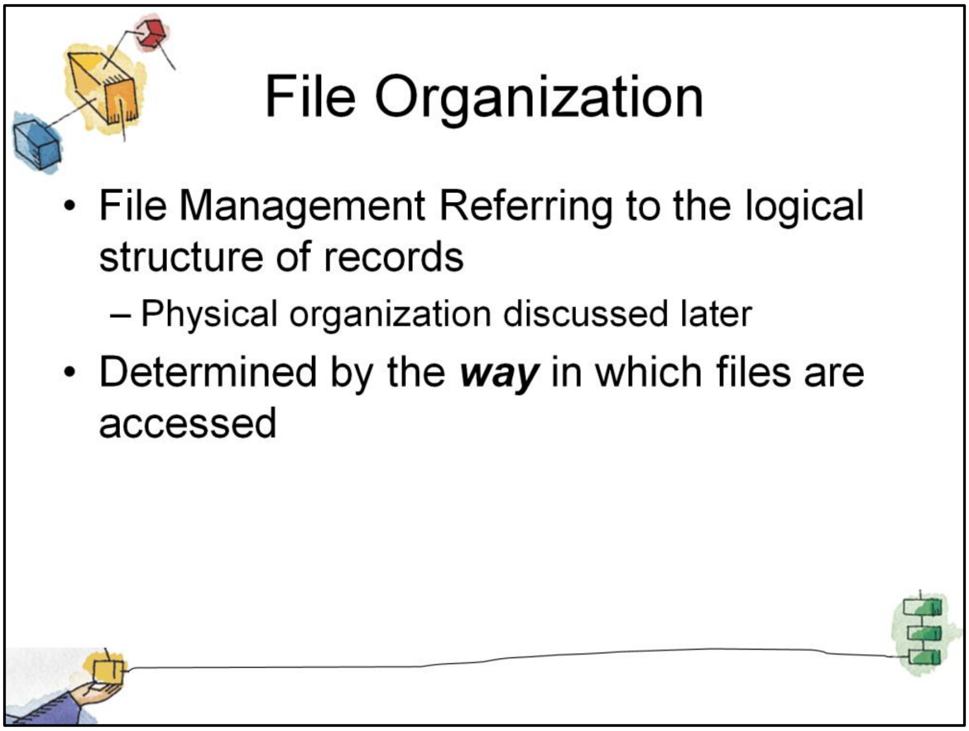
- This involves allocating files to free blocks on secondary storage and managing free storage so as to know what blocks are available for new files and growth in existing files.
- In addition, individual block I/O requests must be scheduled

Disk scheduling and file allocation are both concerned with optimizing performance.

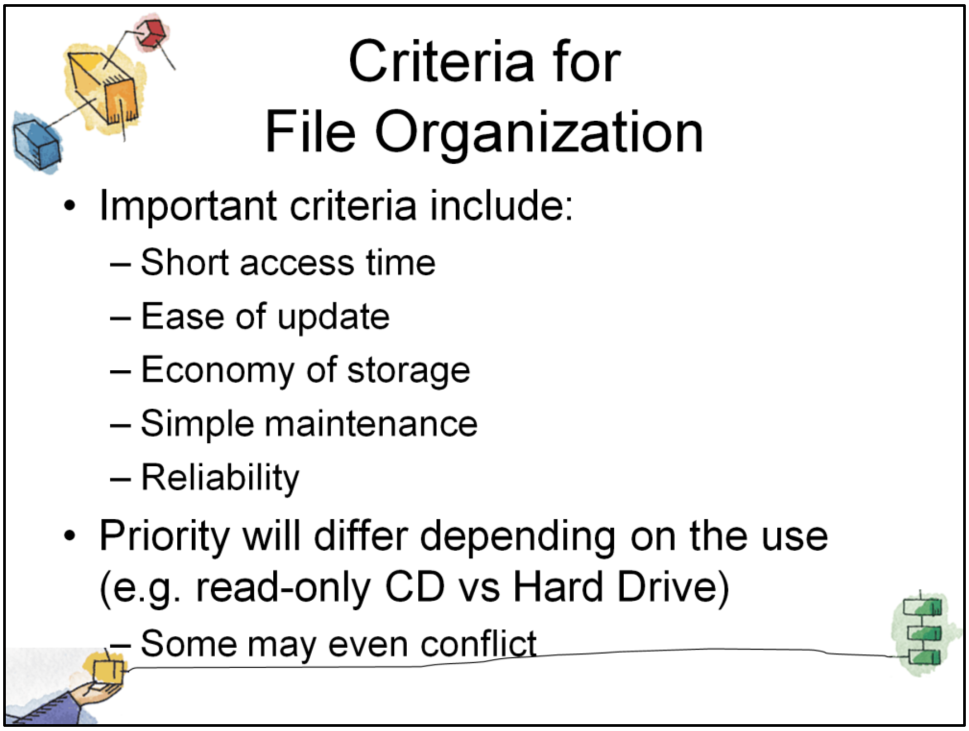
The optimization will depend on the structure of the files and the access patterns.

Roadmap

- Overview
- → **File organisation and Access**
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



"file organization" refers to the logical structuring of the records as determined by the way in which they are accessed.

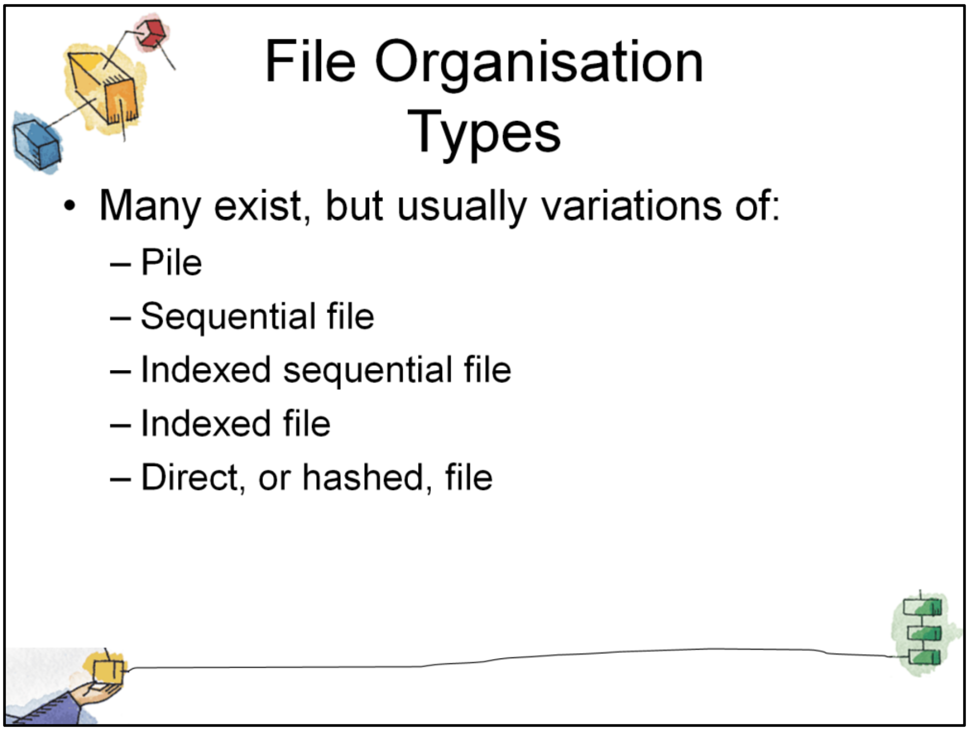


The relative priority of these criteria will depend on the applications that will use the file.

- e.g. if a file is only to be processed in batch mode, with all of the records accessed every time, then rapid access for retrieval of a single record is of minimal concern.
- A file stored on CD-ROM will never be updated, and so ease of update is not an issue.

These criteria may conflict.

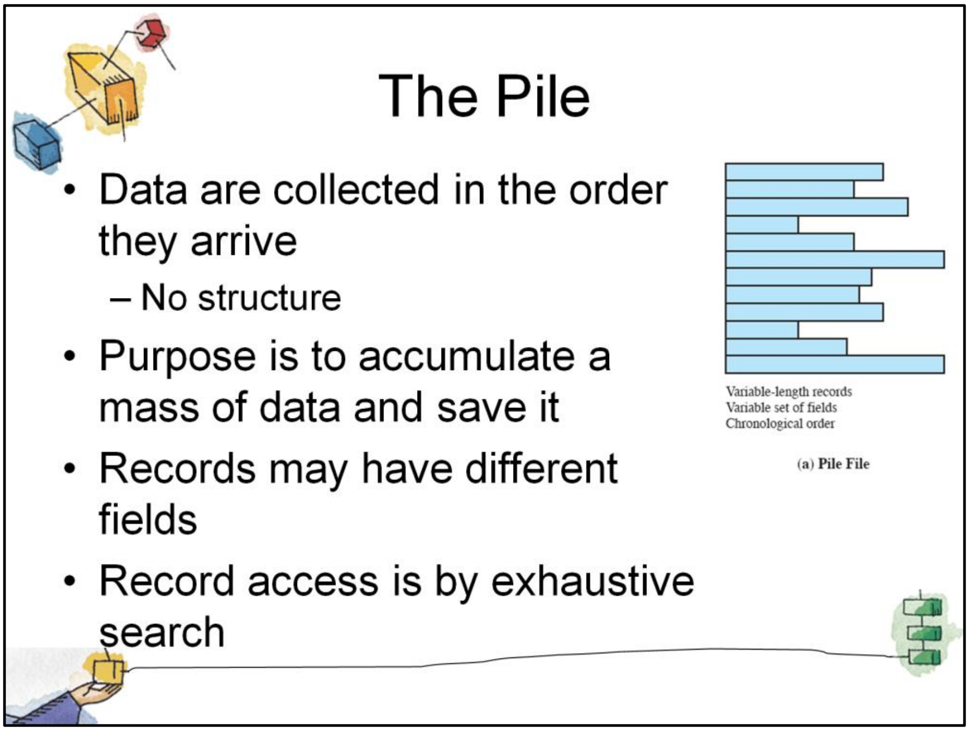
- E.g. for economy of storage, there should be minimum redundancy in the data, but redundancy is a primary means of increasing the speed of access to data (such as indexes.)



Most structures used in actual systems either fall into one of these categories or can be implemented with a combination of these organizations.

The five organizations are:

- The pile
- The sequential file
- The indexed sequential file
- The indexed file
- The direct, or hashed, file



The least-complicated form of file organization may be termed the pile.

Data are collected in the order in which they arrive.

- Each record consists of one burst of data.

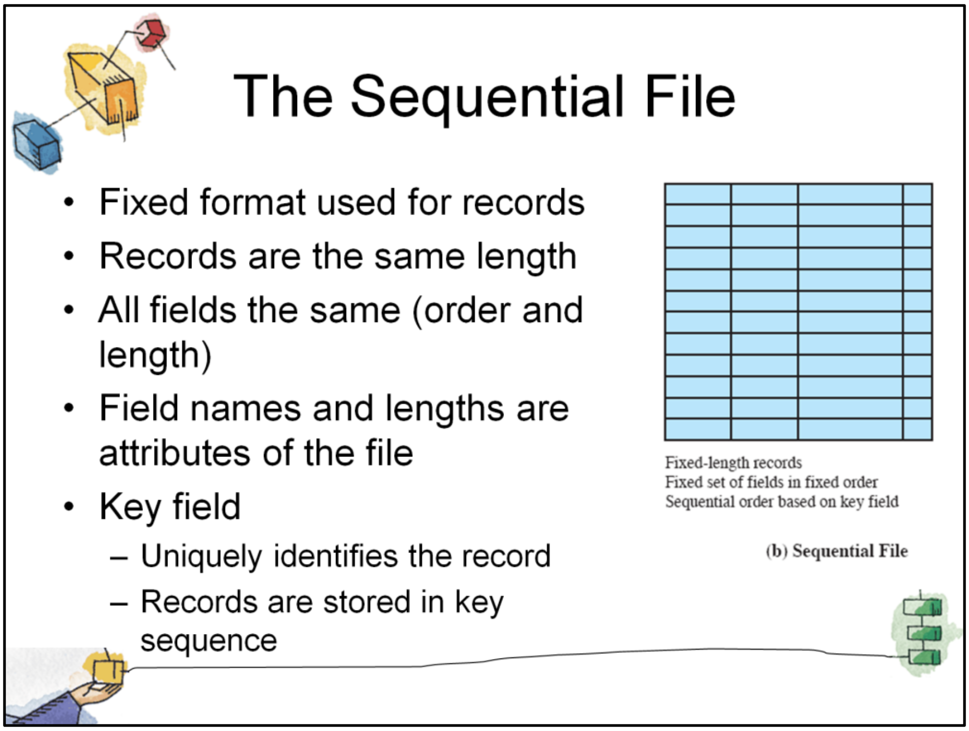
The purpose of the pile is simply to accumulate the mass of data and save it.

Records may have different fields, or similar fields in different orders.

- Thus, each field should be self-describing, including a field name as well as a value.
- The length of each field must be implicitly indicated by delimiters.

Because there is no structure to the pile file, record access is by exhaustive search.

- ie , 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.
- to find all records that contain a particular field or contain that field with a particular value, then the entire file must be searched.



The most common form of file structure.

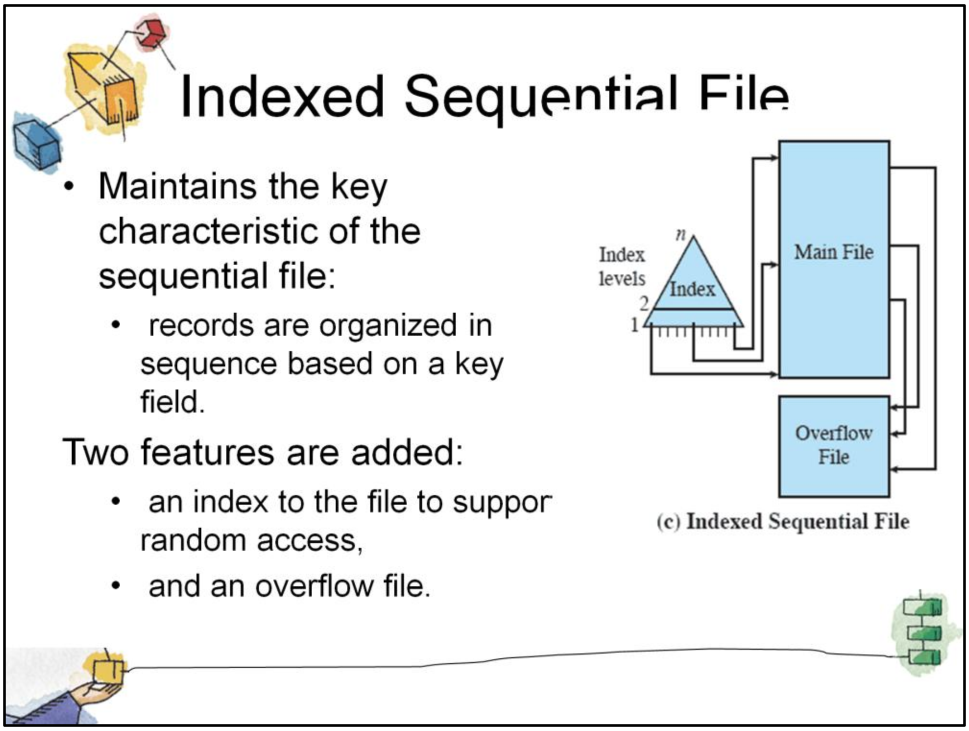
A fixed format is used for records.

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

- Because the length and position of each field are known, only the values of fields need to be stored;
- the field name and length for each field are attributes of the file structure.

The key field uniquely identifies the record;

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



The indexed sequential file maintains the key characteristic of the sequential file:

- records are organized in sequence based on a key field.

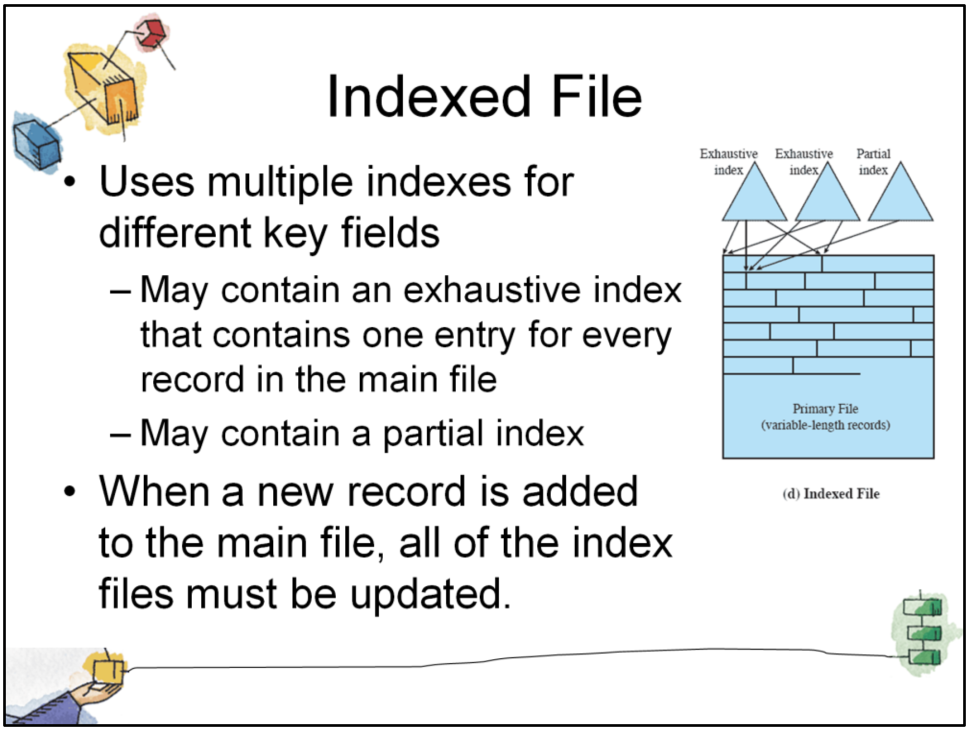
Two features are added:

- an index to the file to support random access,
- and an overflow file.

The index provides a lookup capability to reach quickly the vicinity of a desired record.

The overflow file is similar to the log file used with a sequential file

- but is integrated so that a record in the overflow file is located by following a pointer from its predecessor record.



In the general indexed file, the concept of sequentiality and a single key are abandoned.

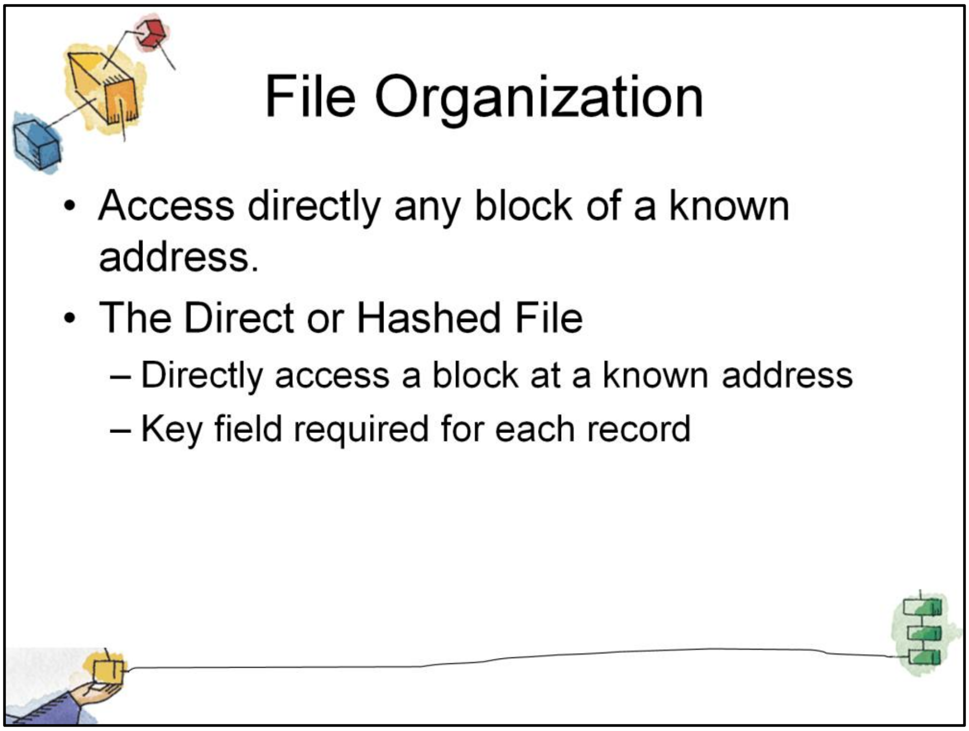
Records are accessed only through their indexes.

- 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.

Two types of indexes are used.

- An exhaustive index contains one entry for every record in the main file. The index itself is organized as a sequential file for ease of searching.
- A partial index contains entries to records where the field of interest exists.

When a new record is added to the main file, all of the index files must be updated.



Exploits the capability found on disks to access directly any block of a known address.

A key field is required in each record.

- But there is no concept of sequential ordering.

The direct file makes use of hashing on the key value.

Direct files are often used where very rapid access is required, where fixed length records are used, and where records are always accessed one at a time.

Performance

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

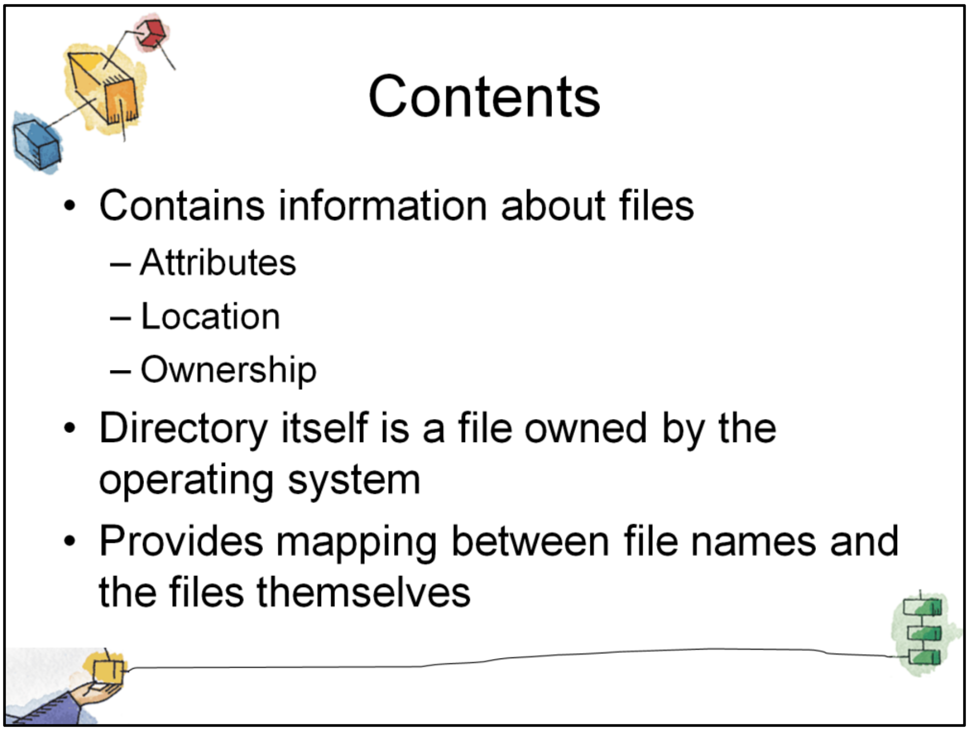
	Space Attributes		**Update** Record Size		**Retrieval**		
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

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

Roadmap

- Overview
- File organisation and Access
- → File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



The directory contains information about the files, including attributes, location, and ownership.

- Much of this information is managed by the operating system.

The directory is itself a file, accessible by various file management routines.

- Although some of the information in directories is available to users and applications, this is generally provided indirectly by system routines.

Directory Elements: Basic Information

- File Name
  - Name as chosen by creator (user or program).
  - Must be unique within a specific directory.
- File type
- File Organisation
  - For systems that support different organizations

From the user's point of view, the directory provides a mapping between file names, known to users and applications, and the files themselves.

This, and the following slides, summarises table 12.2

Directory Elements: Address Information

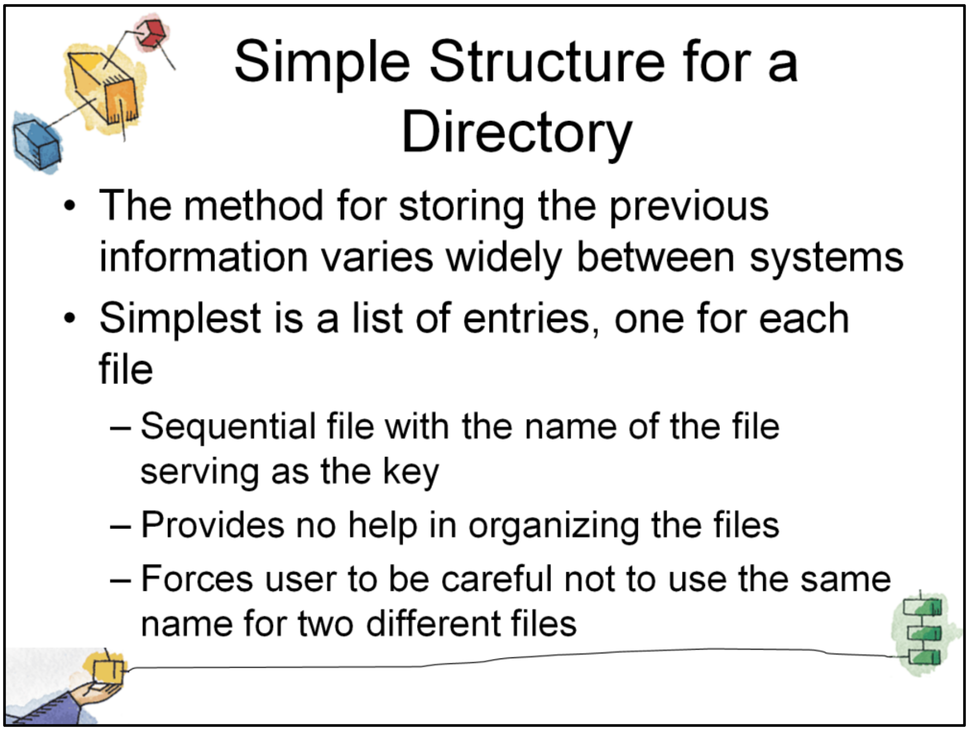
- Volume
  - Indicates device on which file is stored
- Starting Address
- Size Used
  - Current size of the file in bytes, words, or blocks
- Size Allocated
  - The maximum size of the file

Directory Elements: Access Control Information

- Owner
  - The owner may be able to grant/deny access to other users and to change these privileges.
- Access Information
  - May include the user's name and password for each authorized user.
- Permitted Actions
  - Controls reading, writing, executing, transmitting over a network

Directory Elements: Usage Information

- Date Created
- Identity of Creator
- Date Last Read Access
- Identity of Last Reader
- Date Last Modified
- Identity of Last Modifier
- Date of Last Backup
- Current Usage
  - Current activity, locks, etc

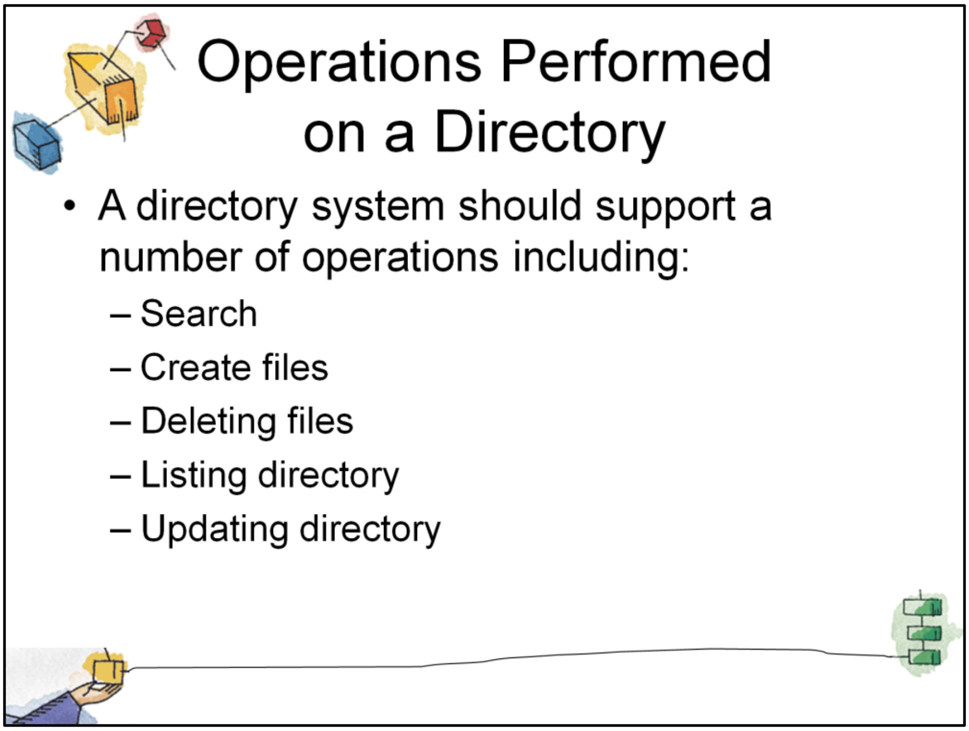


The way in which the information of Table 12.2 is stored differs widely among various systems.

- Some of the information may be stored in a header record associated with the file;
- This reduces the amount of storage required for the directory, making it easier to keep all or much of the directory in main memory to improve speed.

The simplest form of structure for a directory is that of a list of entries, one for each file.

- This structure could be represented by a simple sequential file, with the name of the file serving as the key.



NB the simple list will not easily support these operations.

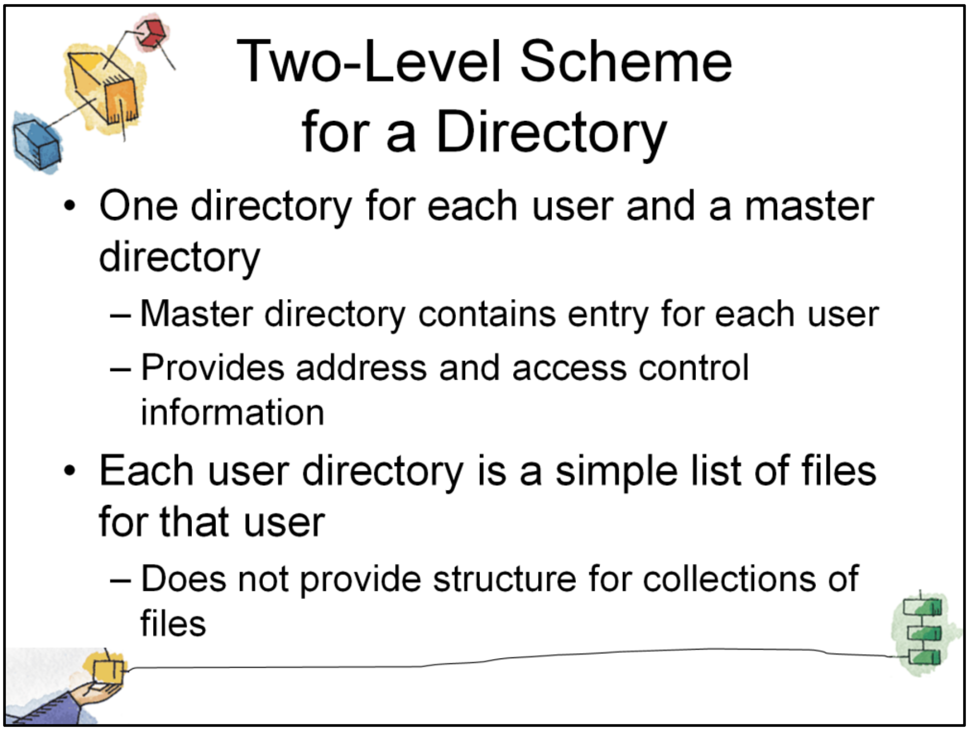
Search: When a user or application references a file, the directory must be searched to find the entry corresponding to that file.

Create file: When a new file is created, an entry must be added to the directory.

Delete file: When a file is deleted, an entry must be removed from the directory.

List directory: All or a portion of the directory may be requested. Generally, this request is made by a user and results in a listing of all files owned by that user, plus some of the attributes of each file

Update directory: Because some file attributes are stored in the directory, a change in one of these attributes requires a change in the corresponding directory entry.



In this case, 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.

This arrangement means that names must be unique only within the collection of files of a single user, and that the file system can easily enforce access restriction on directories.

However, it still provides users with no help in structuring collections of files.

Hierarchical, or Tree-Structured Directory

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

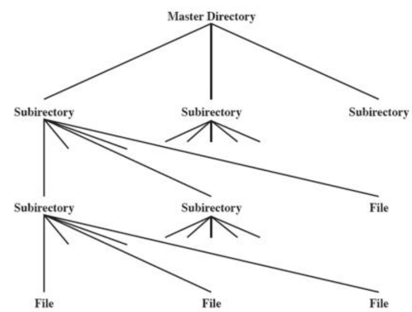


Figure 12.4 Tree-Structured Directory

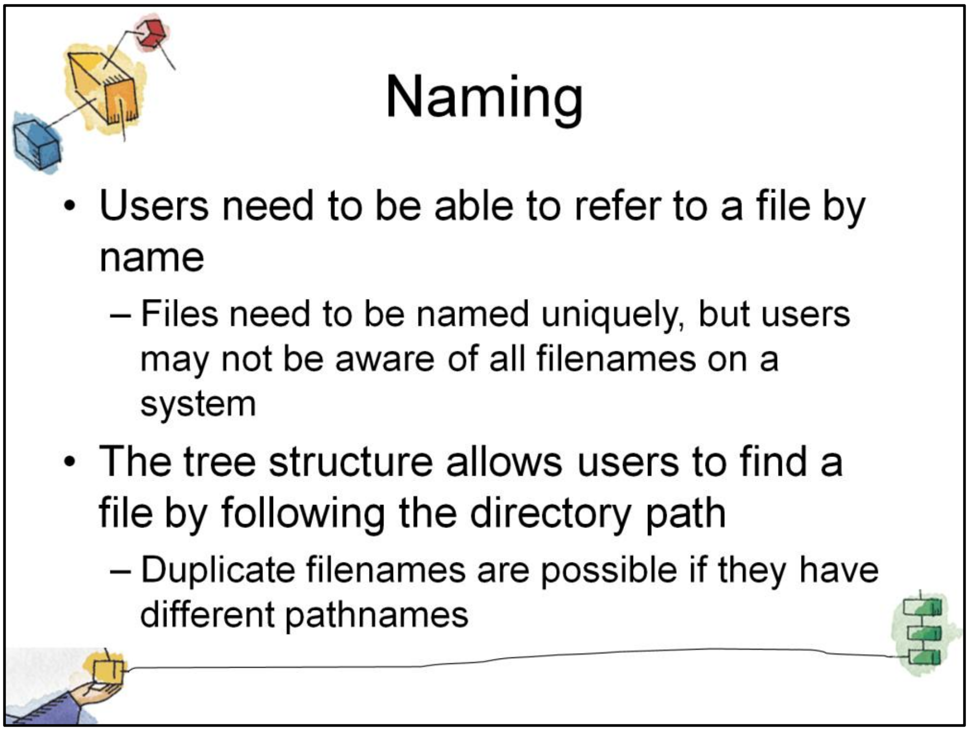
There is a master directory, which has under it a number of user directories.

Each of these user directories, in turn, may have subdirectories and files as entries.

The simplest approach is to store each directory as a sequential file.

When directories may contain a very large number of entries, such an organization may lead to unnecessarily long search times.

- If so, a hashed structure is preferred.



Users need to be able to refer to a file by a symbolic name.

- Each file in the system must have a unique name in order that file references be unambiguous.
- But it is an unacceptable burden on users to require that they provide unique names, especially in a shared system.

The use of a tree-structured directory minimizes the difficulty in assigning unique names.

- Any file in the system can be located by following a path from the root or master directory down various branches until the file is reached.
- The series of directory names, culminating in the file name itself, constitutes a pathname for the file.

Example of Tree-Structured Directory

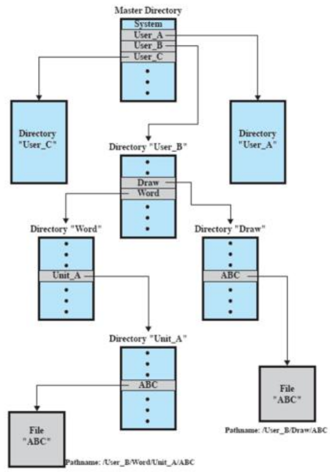
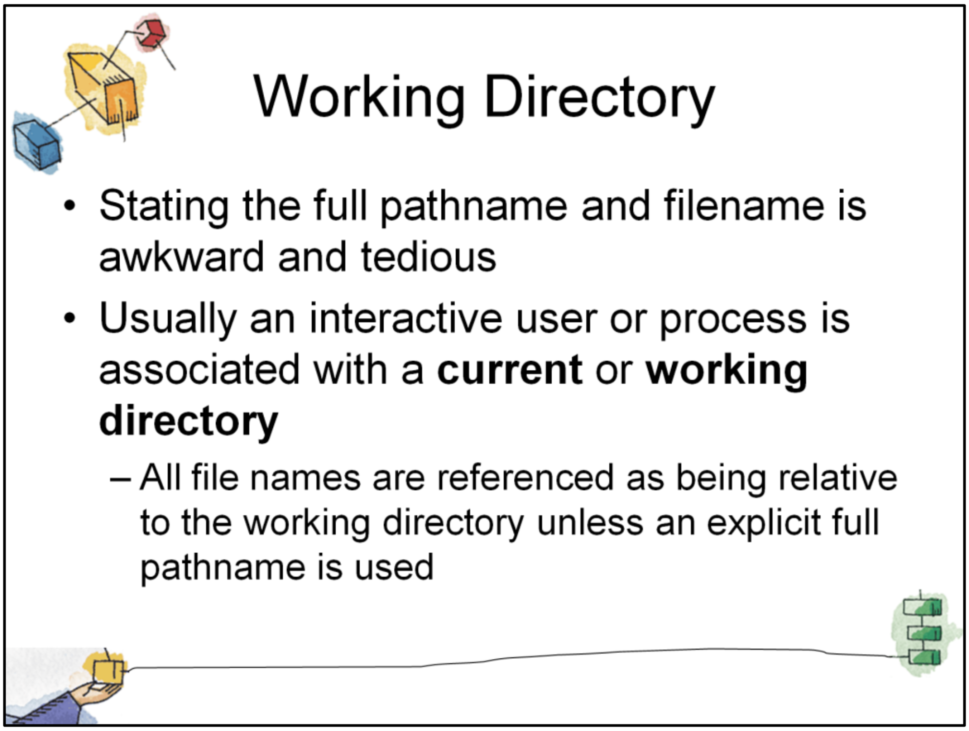


Figure 12.5 Example of Tree-Structured Directory



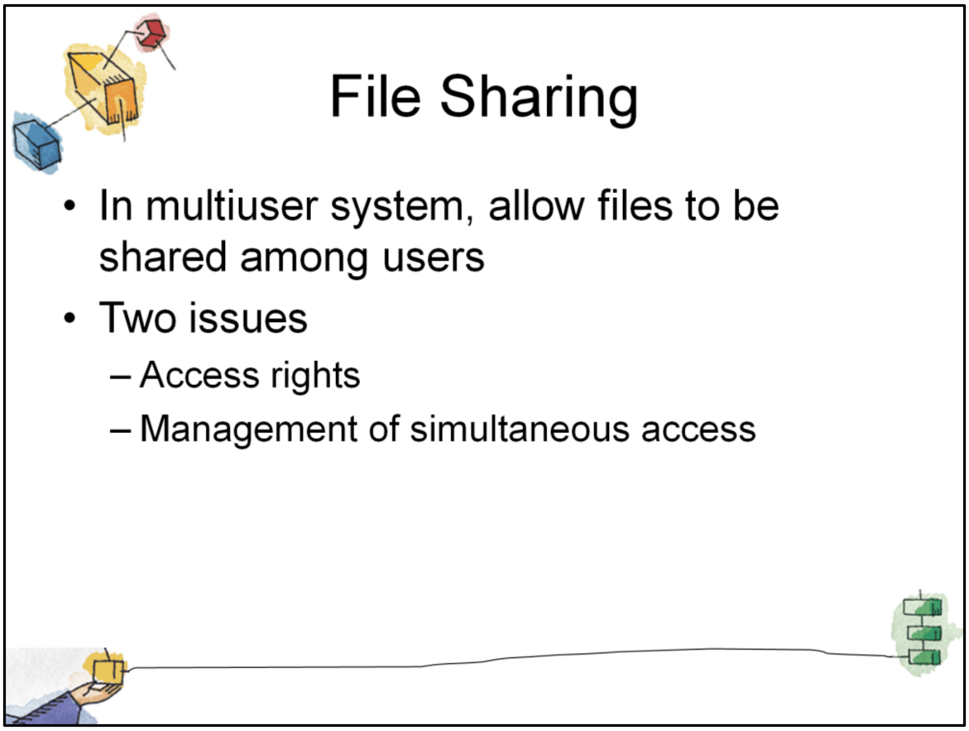
It would be awkward for a user to have to spell out the entire pathname every time a reference is made to a file.

Typically, an interactive user or a process has associated with it a current directory, often referred to as the working directory.

- Files are then referenced relative to the working directory.

Roadmap

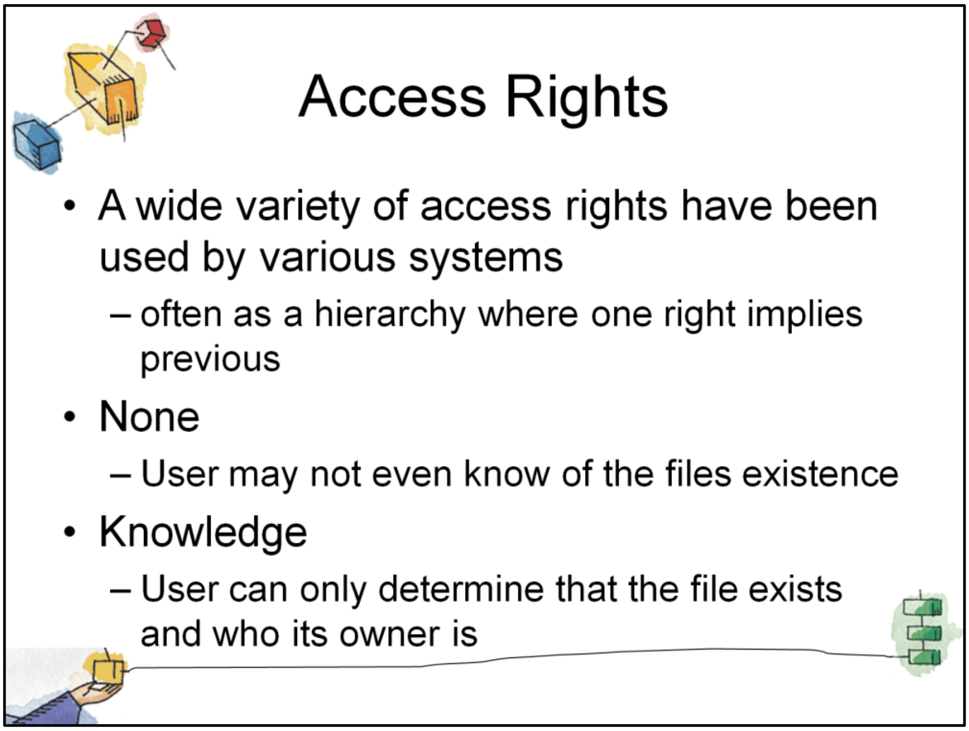
- Overview
- File organisation and Access
- File Directories
- **File Sharing**
- Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



In a multiuser system, there is almost always a requirement for allowing files to be shared among a number of users.

Two issues arise:

- access rights and
- the management `of simultaneous access.



The file system should provide a number of options so that the way in which a particular file is accessed can be controlled.

Typically, users or groups of users are granted certain access rights to a file.

A wide range of access rights has been used.

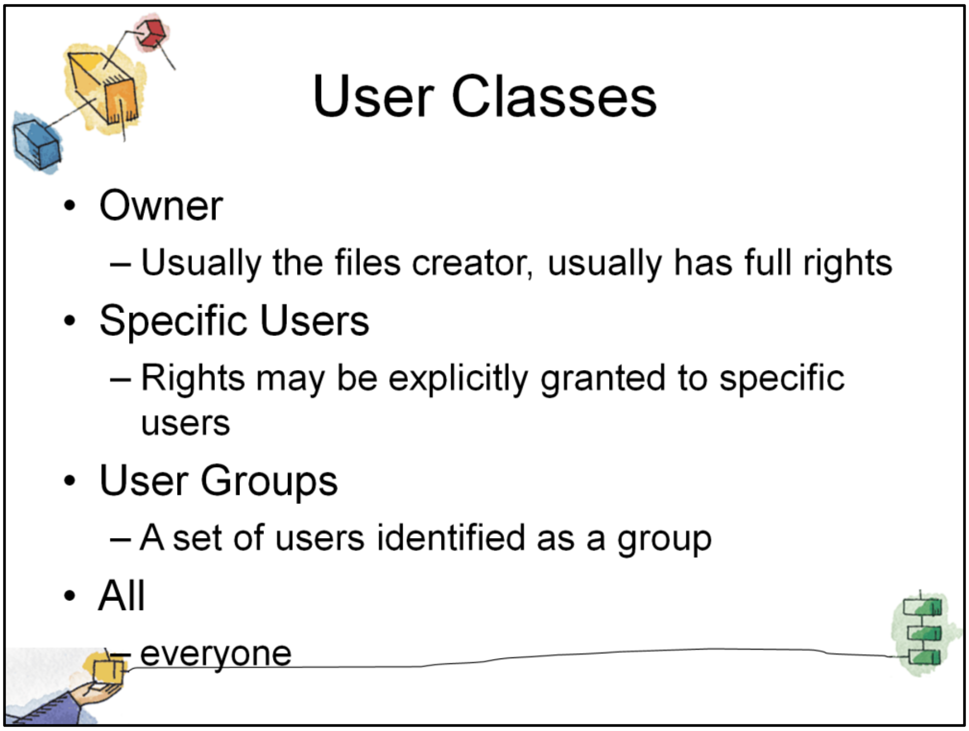
- These rights can be considered to constitute a hierarchy, with each right implying those that precede it.

Access Rights cont…

- 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

Access Rights cont…

- Updating
  - The user can modify, delete, and add to the file's data.
- Changing protection
  - User can change access rights granted to other users
- Deletion
  - User can delete the file



Owner of a given file, usually the person who initially created a file.

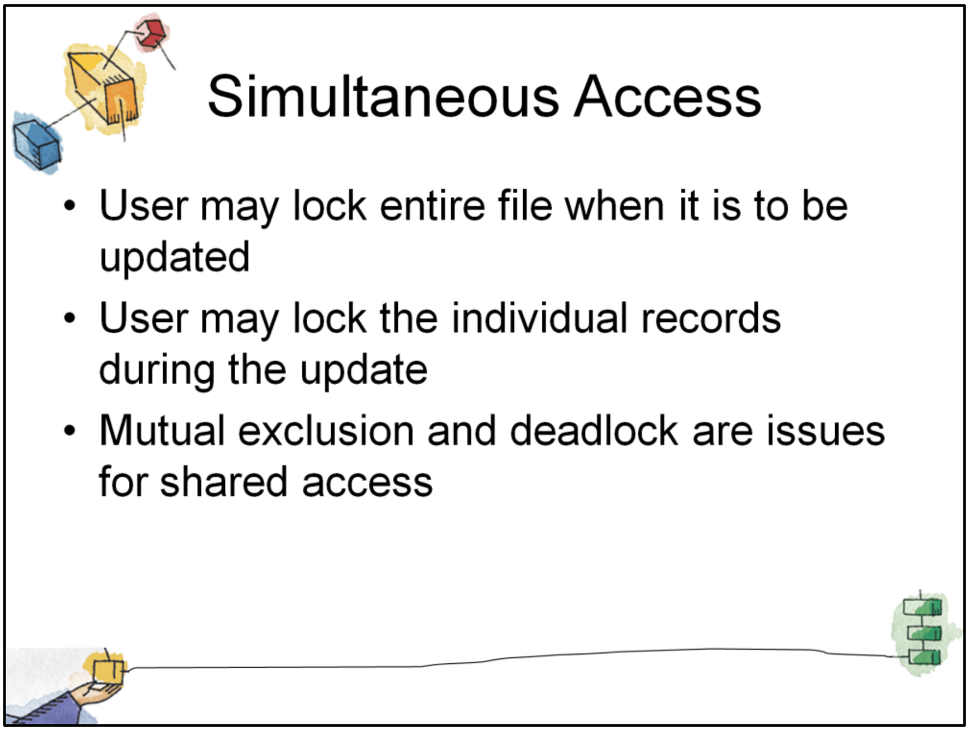
The owner has all of the access rights listed previously and may grant rights to others.

Specific user: Individual users who are designated by user ID.

User groups: A set of users who are not individually defined.

- The system must have some way of keeping track of the membership of user groups.

All: All users who have access to this system. These are public files.



When access is granted to append or update a file to more than one user, the operating system or file management system must enforce discipline.

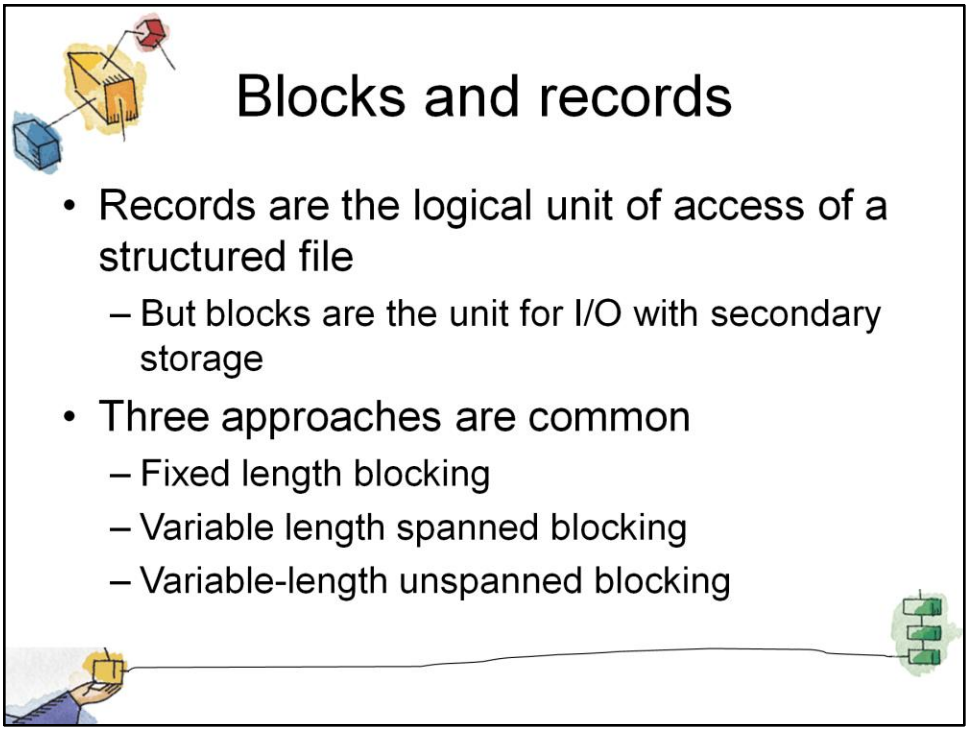
A brute-force approach is to allow a user to lock the entire file when it is to be updated.

- A finer grain of control is to lock individual records during update.

Issues of mutual exclusion and deadlock must be addressed in designing the shared access capability.

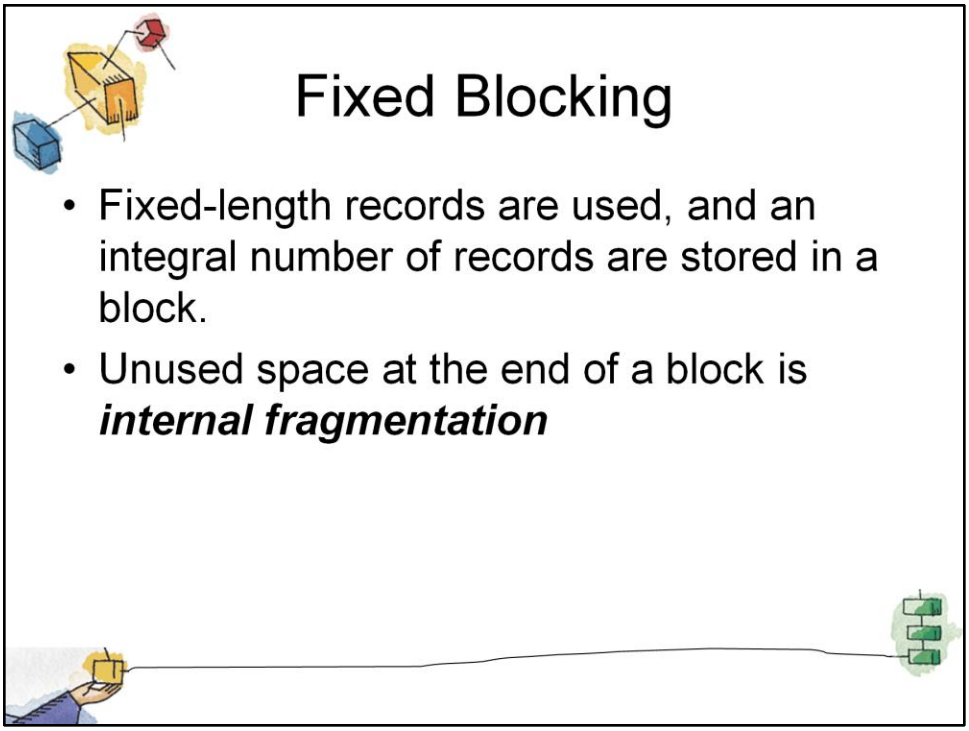
Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- → Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



Records are the logical unit of access of a structured file,

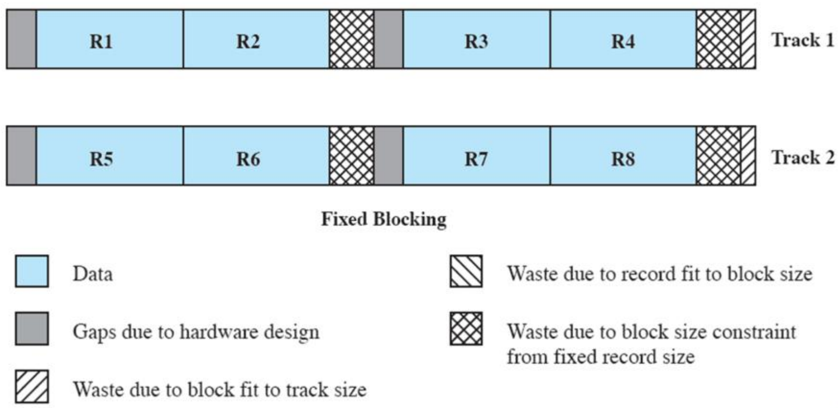
Whereas blocks are the unit of I/O with secondary storage. For I/O to be performed, records must be organized as blocks.

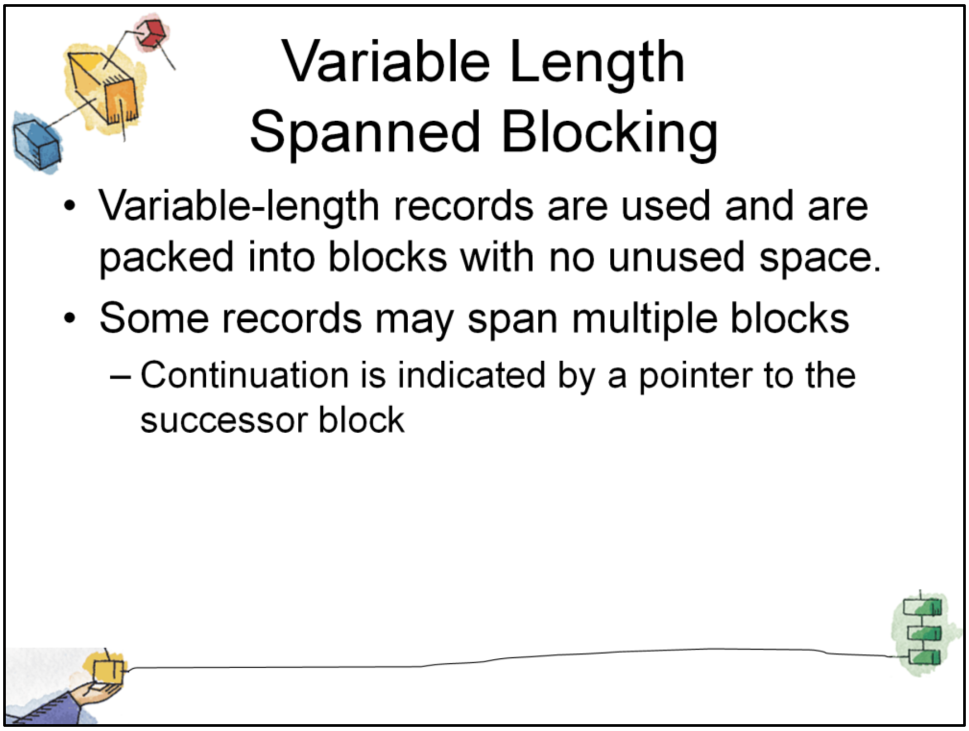


Fixed blocking: Fixed-length records are used, and an integral number of records are stored in a block.

- There may be unused space at the end of each block.
- This is referred to as **internal fragmentation.**

Fixed Blocking

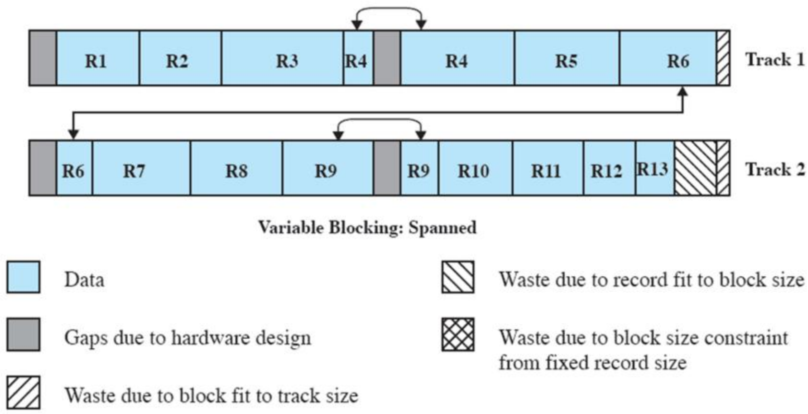




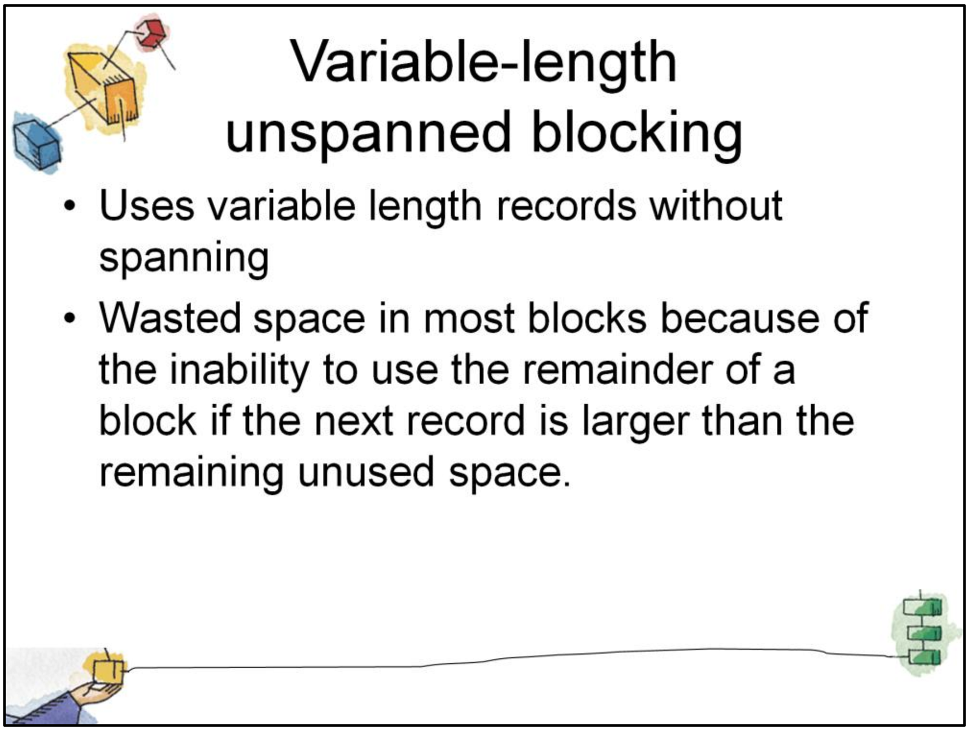
Variable-length records are used and are packed into blocks with no unused space.

Thus, some records must span two blocks, with the continuation indicated by a pointer to the successor block.

Variable Blocking: Spanned



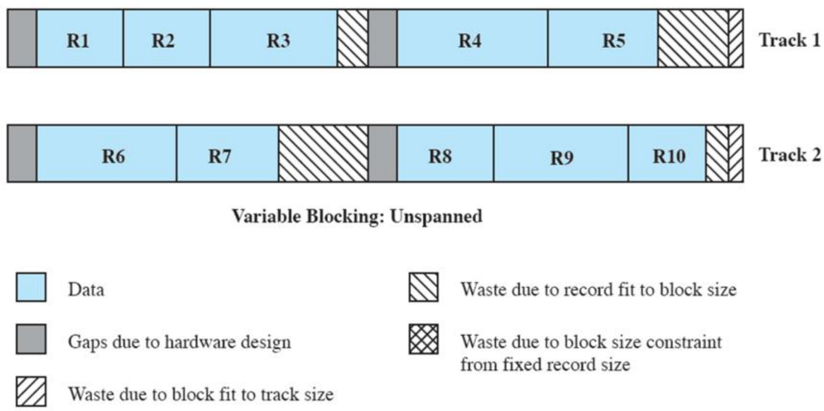
Variable Blocking: Spanned



Variable-length records are used, but spanning is not employed.

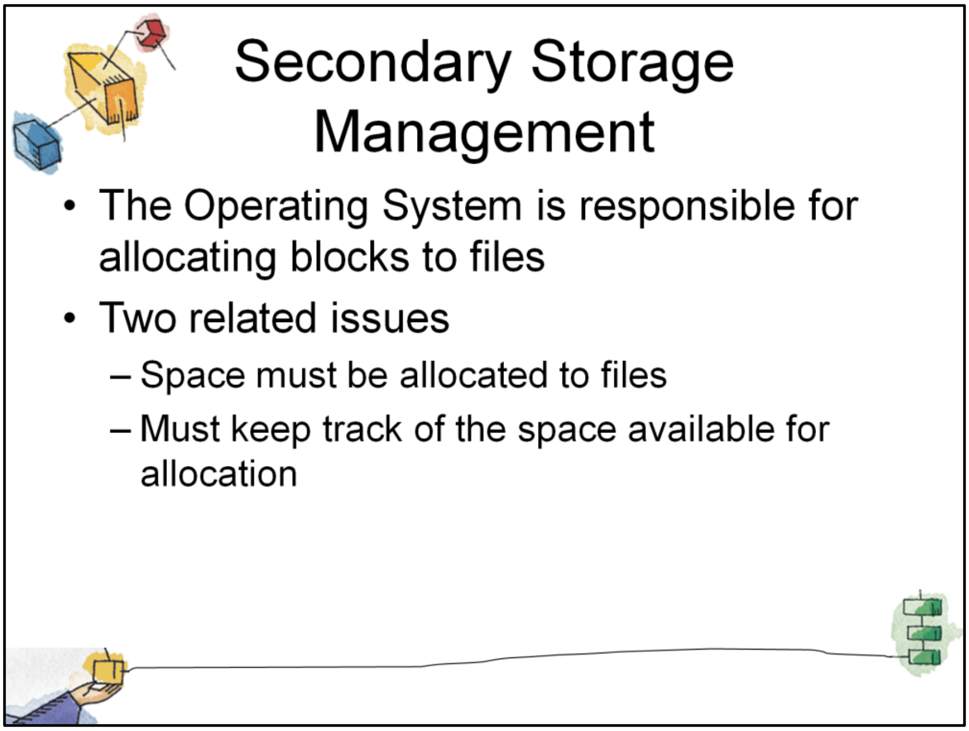
There is wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.

Variable Blocking: Unspanned



Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- → **Secondary Storage Management**
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System

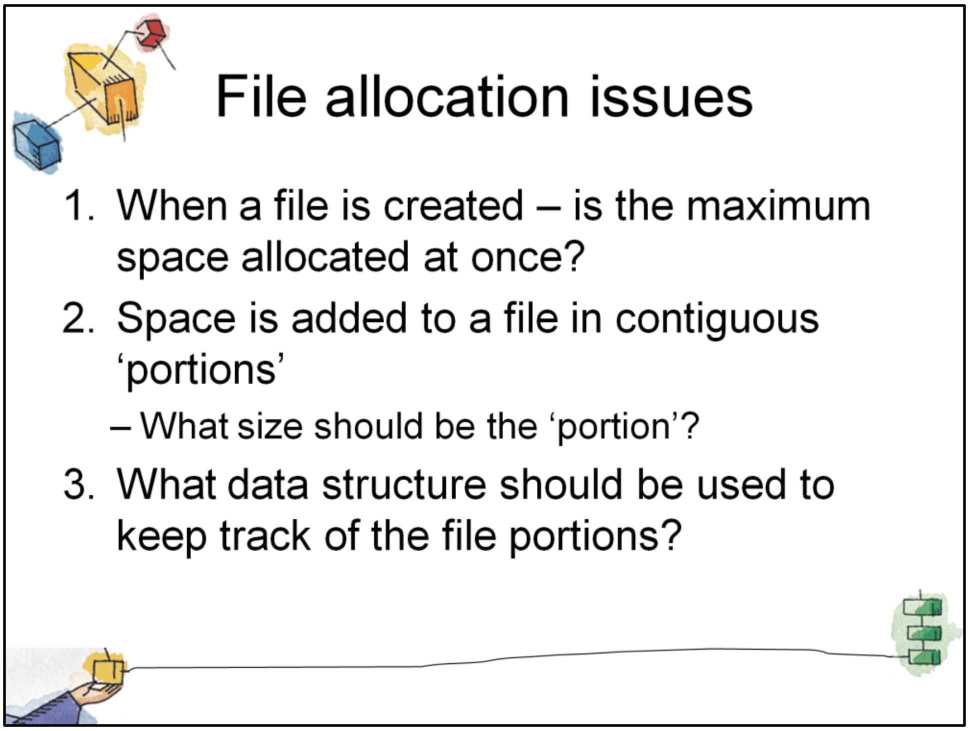


On secondary storage, a file consists of a collection of blocks.

- 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.



1. When a new file is created, is the maximum space required for the file allocated at once?

2. Space is allocated to a file as one or more contiguous units, which we shall refer to as portions.

- That is, a portion is a contiguous set of allocated blocks.
- The size of a portion can range from a single block to the entire file.
- What size of portion should be used for file allocation?

3. What sort of data structure or table is used to keep track of the portions assigned to a file?

- An example of such a structure is a file allocation table (FAT), found on DOS and some other systems.

Preallocation vs Dynamic Allocation

- Need the maximum size for the file at the time of creation
- Difficult to reliably estimate the maximum potential size of the file
- Tend to overestimated file size so as not to run out of space

A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request.

Sometimes such as program compilations, the production of summary data files, or the transfer of a file from another system over a communications network, this value can be reliably estimated.

- But usually it is difficult if not impossible to estimate reliably the maximum potential size of the file.
- In those cases, users and application programmers would tend to overestimate file size, leading to wasted space

Thus, there are advantages to the use of dynamic allocation, which allocates space to a file in portions as needed.

Portion size

- Two extremes:
  - Portion large enough to hold entire file is allocated
  - Allocate space one block at a time
- Trade-off between efficiency from the point of view of a single file, or the overall system efficiency

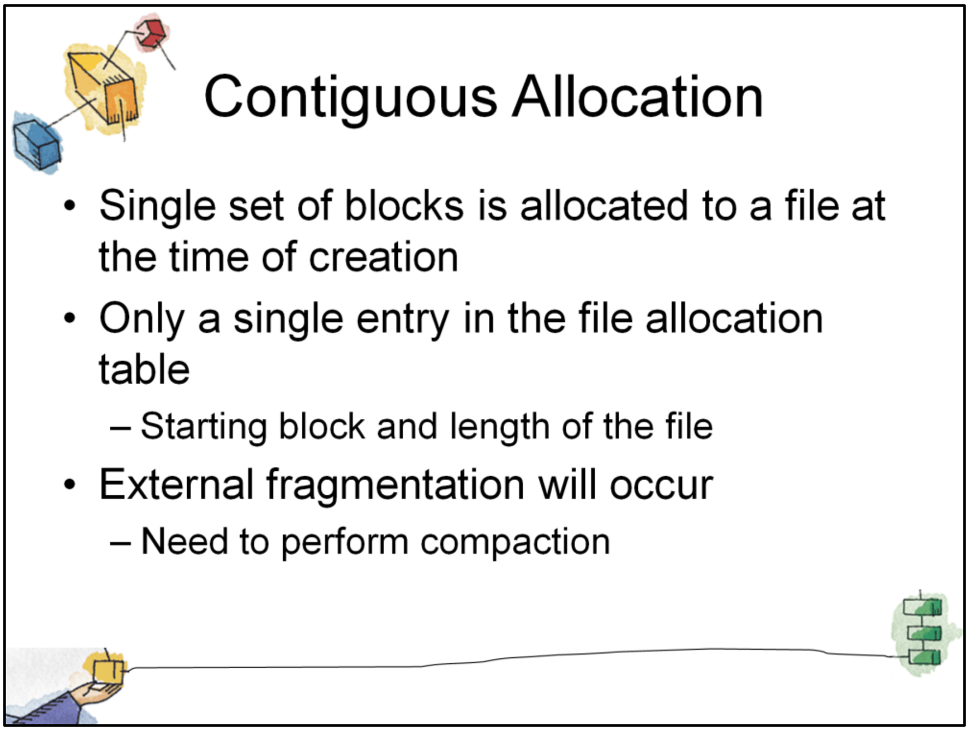
At one extreme, a portion large enough to hold the entire file is allocated.

Space on the disk is allocated one block at a time.

In choosing a portion size, there is a tradeoff between efficiency from the point of view of a single file versus overall system efficiency.

File Allocation Method

- Three methods are in common use:
  - contiguous,
  - chained, and
  - indexed.



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

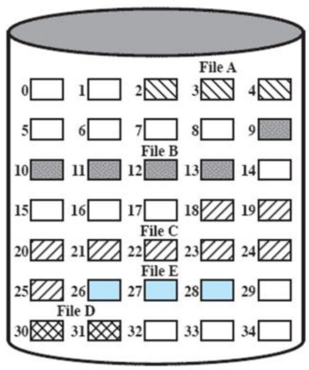
- This is a preallocation strategy, using variable-size portions.

The file allocation table needs just a single entry for each file, showing the starting block and the length of the file.

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.

Contiguous File Allocation

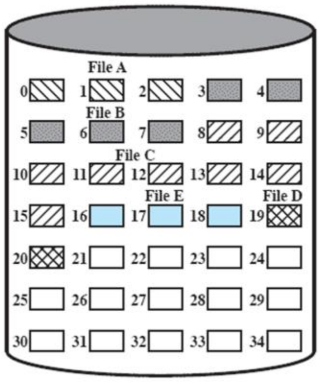


File Allocation Table

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 12.7 Contiguous File Allocation

External fragmentation



File Allocation Table

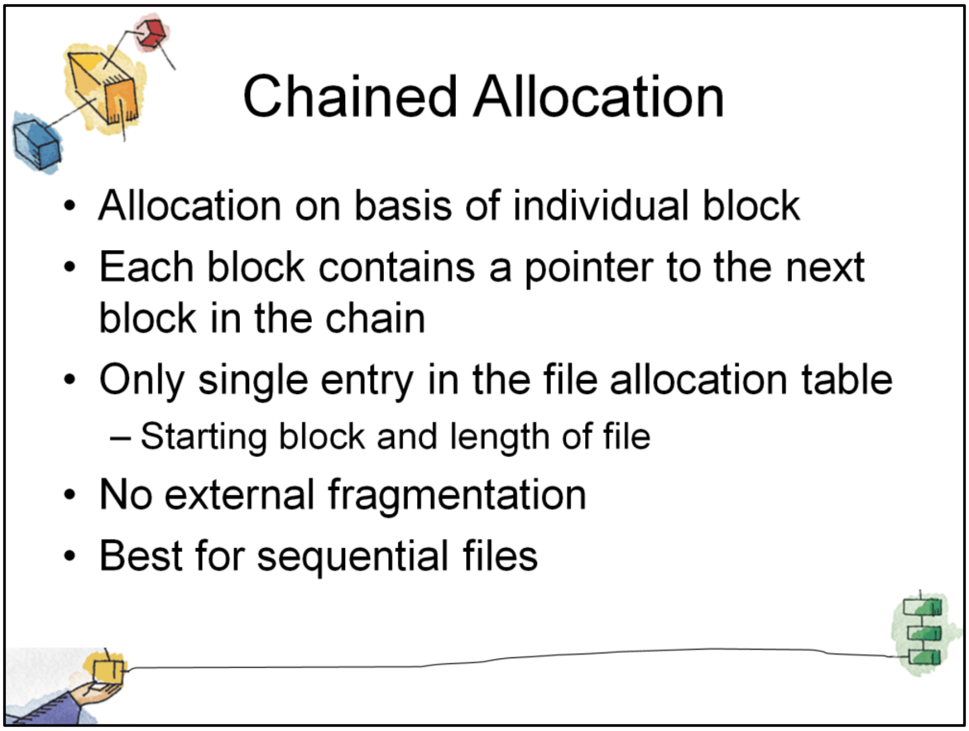
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

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

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

Also, with preallocation, it is necessary to declare the size of the file at the time of creation, with the problems mentioned earlier.



Typically, 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.

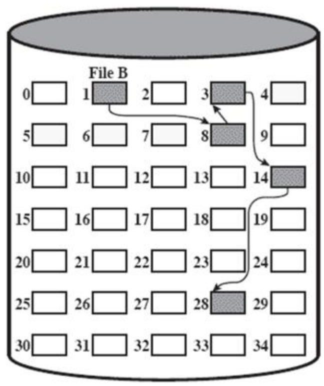
Although preallocation is possible, it is more common simply to allocate blocks as needed.

- The selection of blocks is now a simple matter: any free block can be added to a chain.
- There is no external fragmentation to worry about because only one block at a time is needed.

This type of physical organization is best suited to sequential files that are to be processed sequentially.

- To select an individual block of a file requires tracing through the chain to the desired block.

Chained Allocation



File Allocation Table

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

Figure 12.9 Chained Allocation

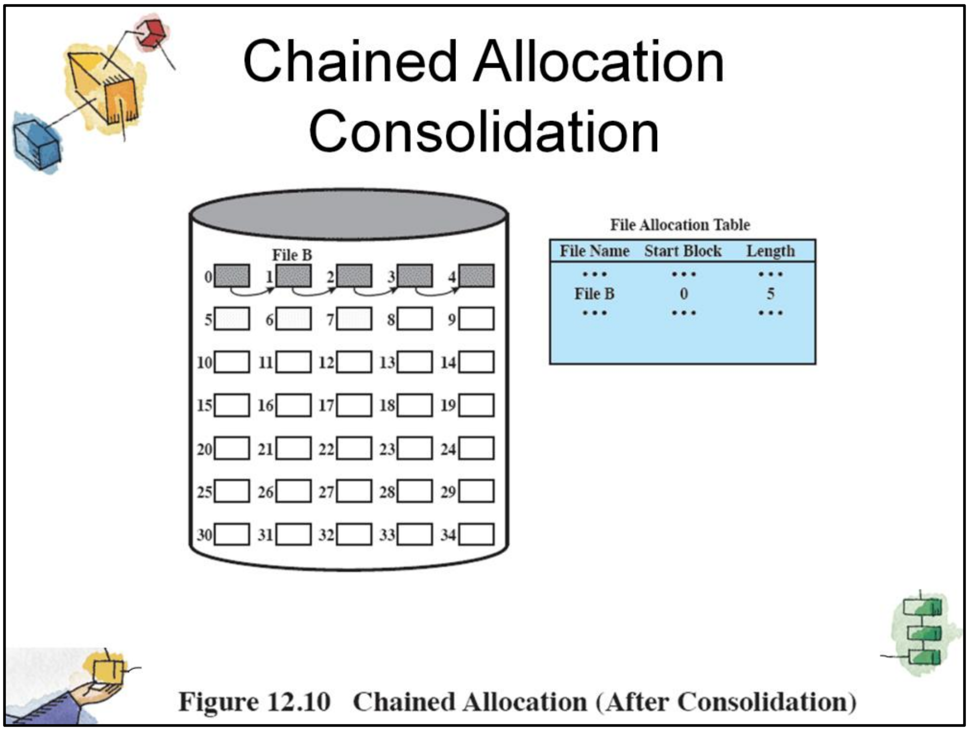
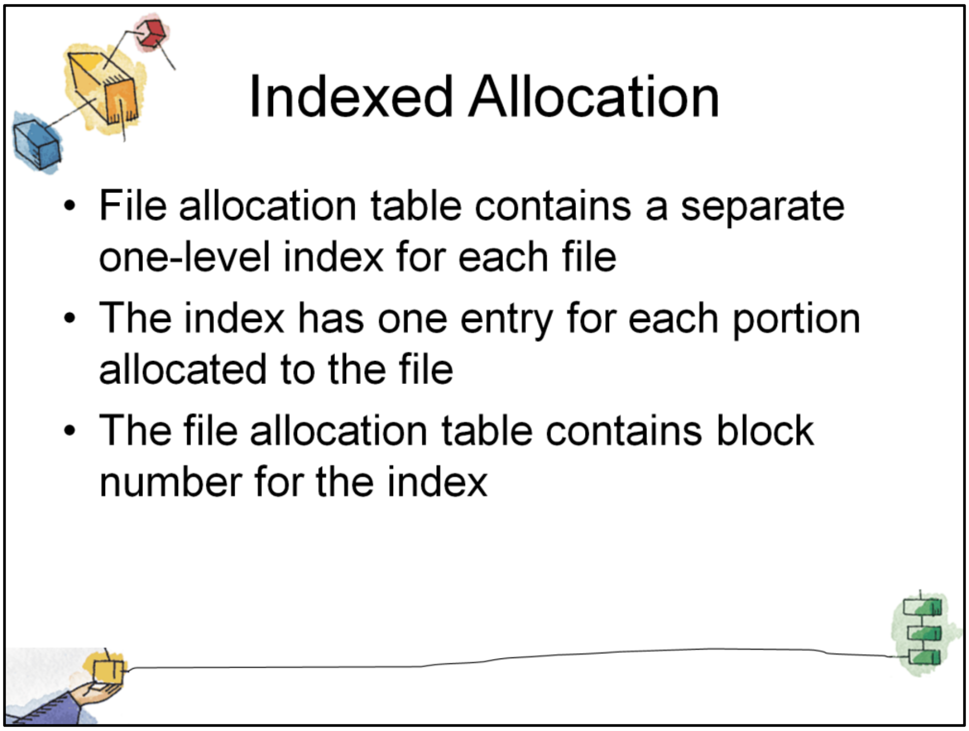


Figure 12.10 Chained Allocation (After Consolidation)

One consequence of chaining, is that there is 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.

- This is perhaps a more significant effect on a single-user system but may also be of concern on a shared system.
- To overcome this problem, some systems periodically consolidate files



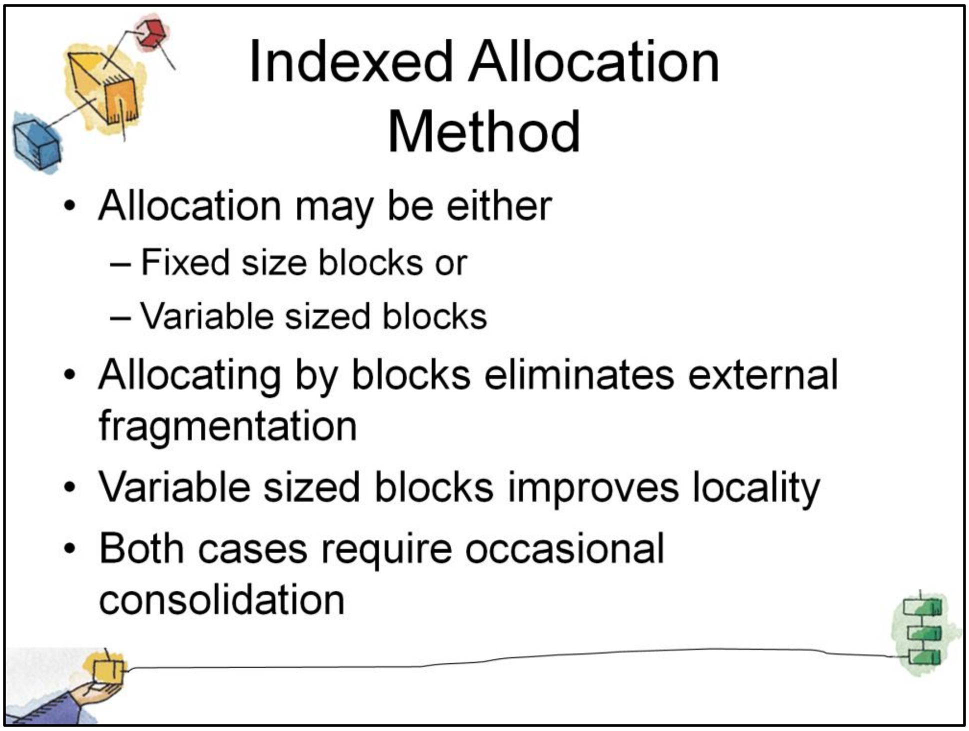
This addresses many of the problems of contiguous and chained allocation.

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.

Typically, the file indexes are not physically stored as part of the file allocation table.

- Rather, the 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.



Allocation may be on the basis of either

- fixed-size blocks or
- variable-size portions

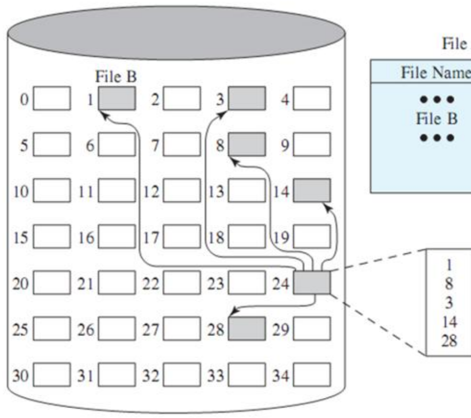
Allocation by blocks eliminates external fragmentation,

- whereas allocation by variable-size portions improves locality.

In either case, file consolidation may be done from time to time.

- File consolidation reduces the size of the index in the case of variable-size portions, but not in the case of block allocation.

Indexed allocation with Block Portions

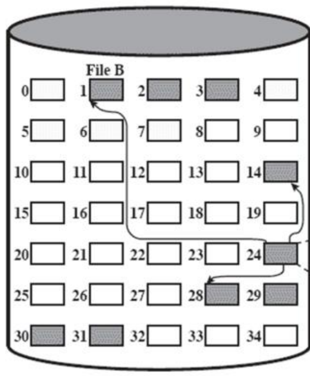


File Allocation Table

File Name	Index Block
•••	•••
File B	24
•••	•••

Figure 12.11 **Indexed Allocation with Block Portions**

Indexed Allocation with Variable Length Portions

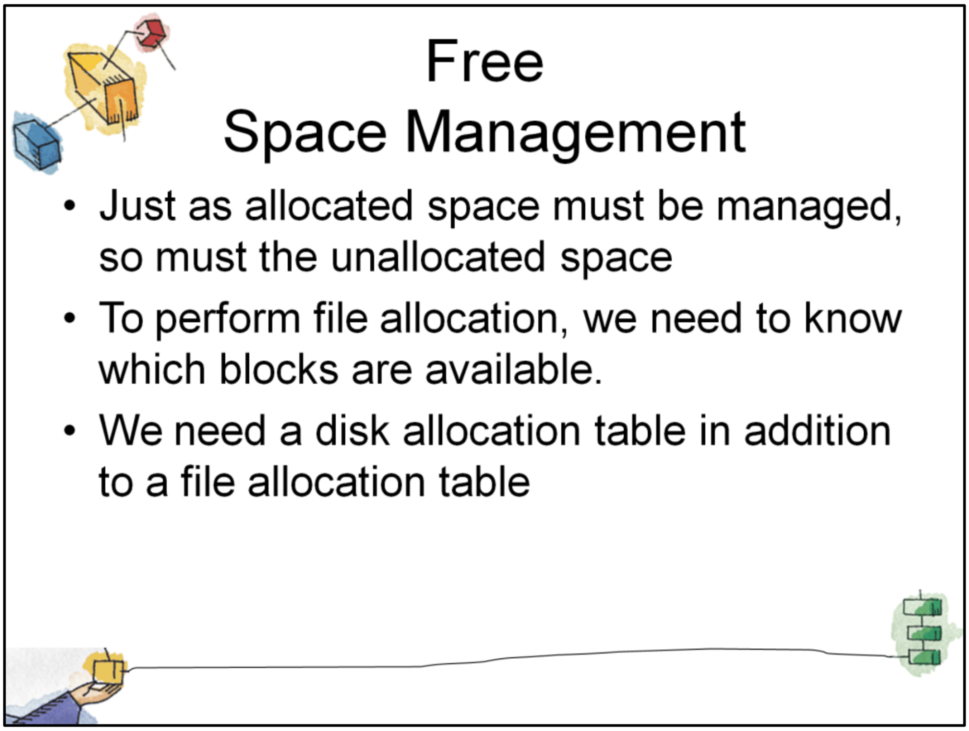


File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

Start Block	Length
1	3
28	4
14	1

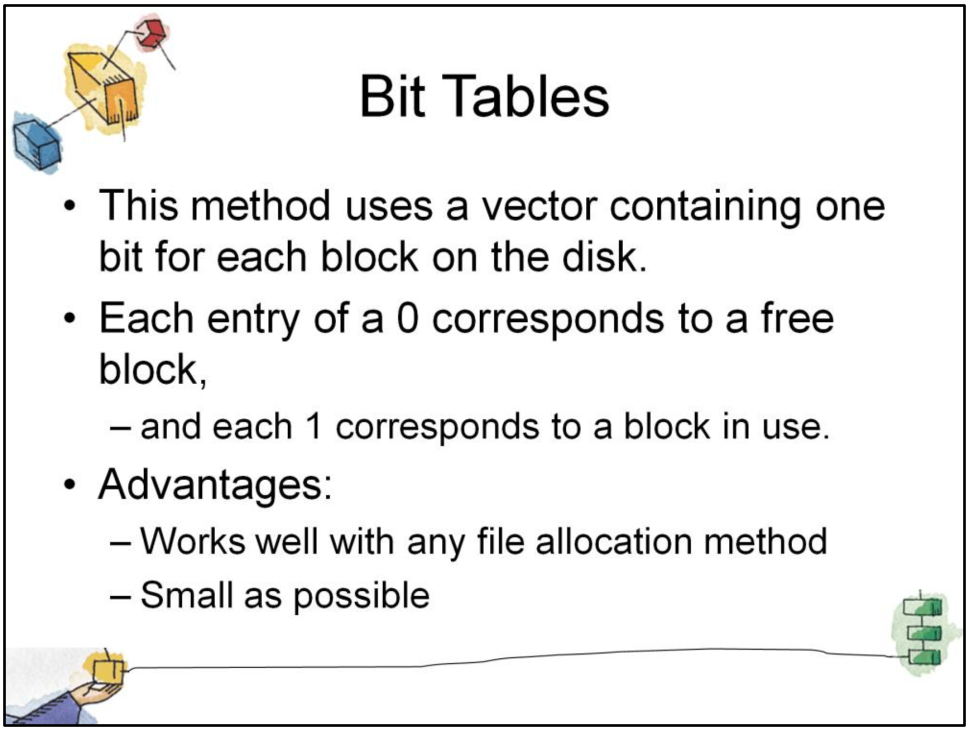
Figure 12.12 Indexed Allocation with Variable-Length Portions



Just as the space that is allocated to files must be managed, so the space that is not currently allocated to any file must be managed.

To perform any of the file allocation techniques described previously, it is necessary to know what blocks on the disk are available.

Thus we need a disk allocation table in addition to a file allocation table.

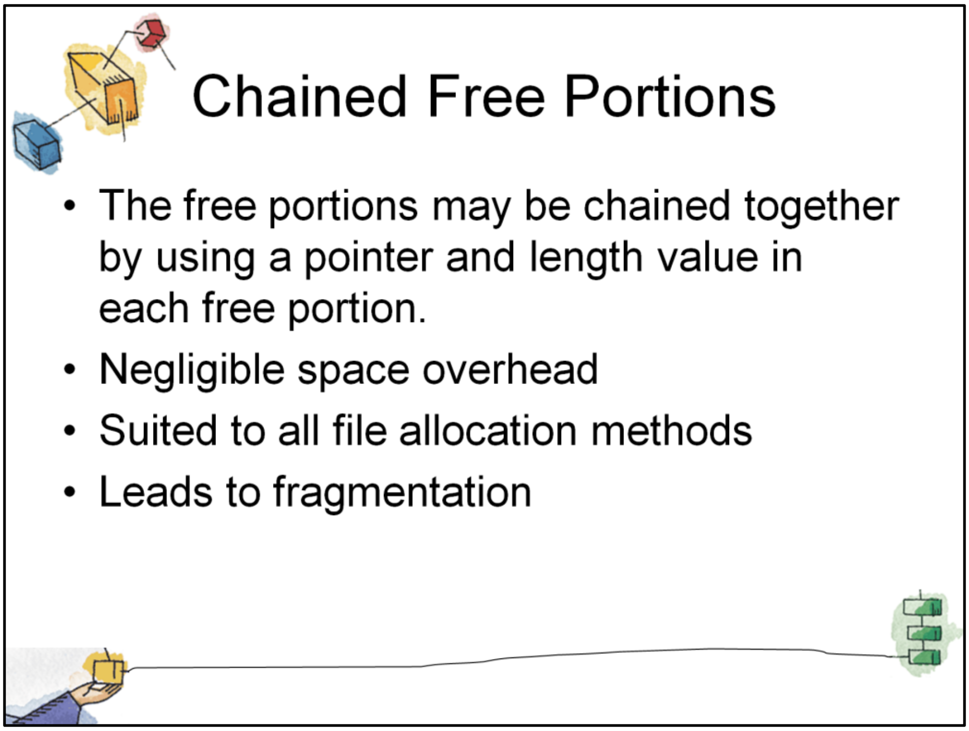


This method uses a vector containing one bit for each block on the disk.

- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use.

A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks.

- Thus, a bit table works well with any of the file allocation methods outlined.
- Another advantage is that it is as small as possible.



The free portions may be chained together by using a pointer and length value in each free portion.

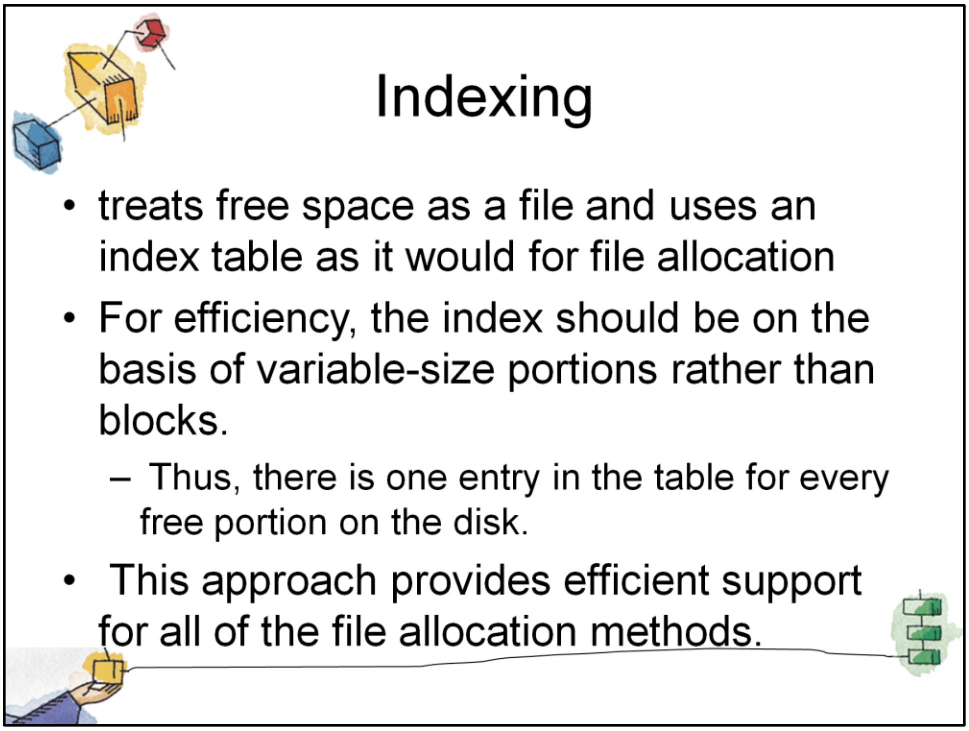
This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain and the length of the first portion.

This method is suited to all of the file allocation methods.

- 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.

This method has its own problems.

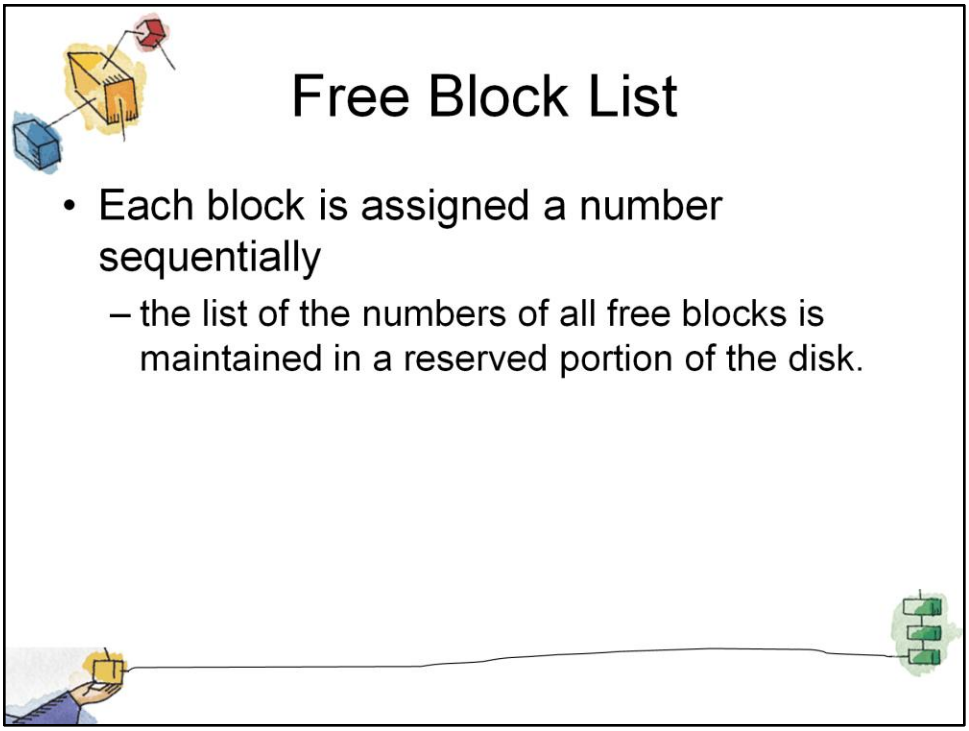
- After some use, the disk will become quite fragmented and many portions will be a single block long.
- Also note that every time you allocate a block, you need to read the block first to recover the pointer to the new first free block before writing data to that block. If many individual blocks need to be allocated at one time for a file operation, this greatly slows file creation
- Similarly, deleting highly fragmented files is very time consuming.



The indexing approach treats free space as a file and uses an index table as described under file allocation.

For efficiency, the index should be on the basis of variable-size portions rather than blocks.

- Thus, there is one entry in the table for every free portion on the disk.
- This approach provides efficient support for all of the file allocation methods.



In this method, each block is assigned a number sequentially and 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, so the size of the free block list is 24 or 32 times the size of the corresponding bit table and thus must be stored on disk rather than in main memory.

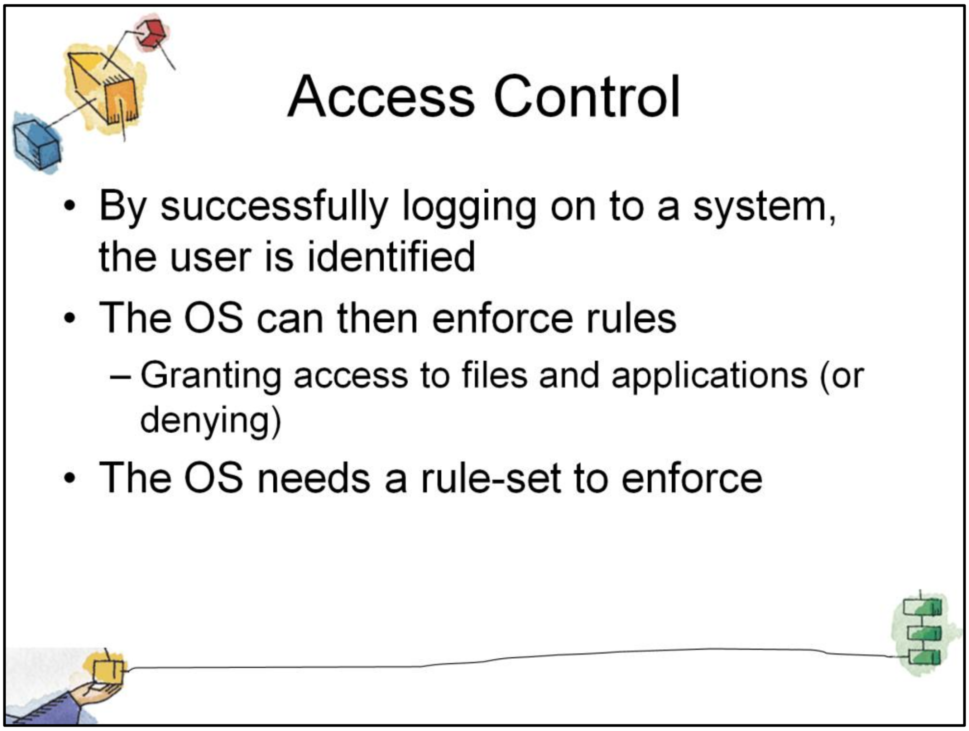
Volumes

- 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;
  - instead they need only appear that way to the OS or application.
- A volume may be the result of assembling and merging smaller volumes.

The term volume is used somewhat differently by different operating systems and file management systems, but in essence a volume is a logical disk.

Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- → File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



Following successful logon, the user has been granted access to one or a set of hosts and applications.

Through the user access control procedure, a user can be identified to the system.

- Associated with each user, there can be a profile that specifies permissible operations and file accesses.
- The operating system can then enforce rules based on the user profile.

Access Matrix

- One such rule set is an Access Matrix

	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

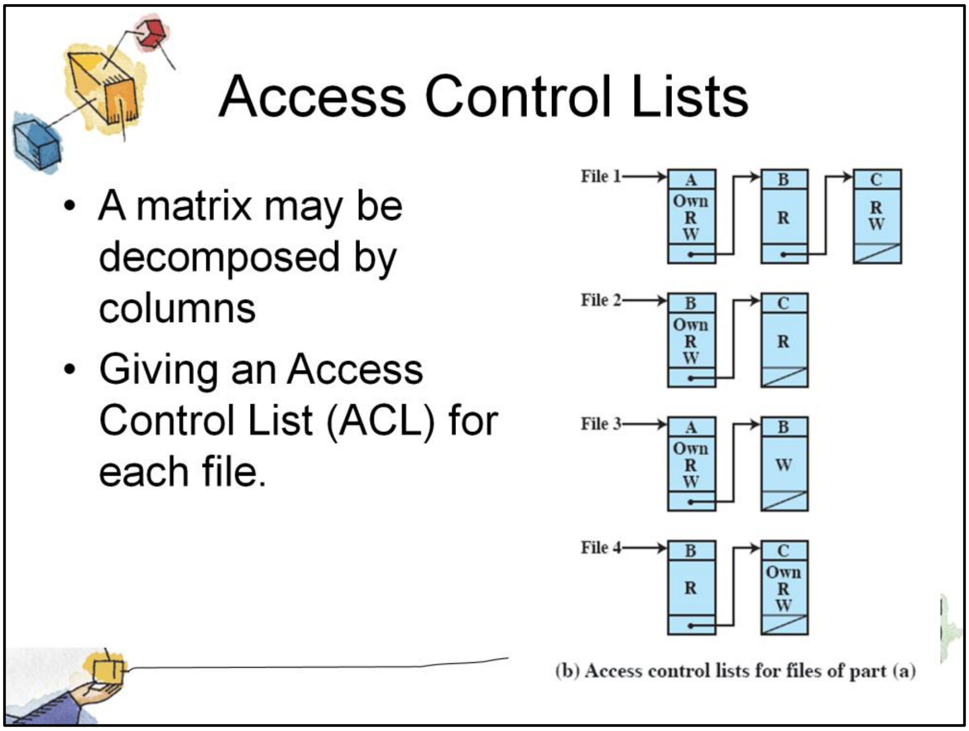
One dimension of the matrix consists of identified subjects that may attempt data access.

- Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, hosts, or applications instead of or in addition to users.

The other dimension lists the objects that may be accessed.

- At the greatest level of detail, objects may be individual data fields.
- More aggregate groupings, such as records, files, or even the entire database, may also be objects in the matrix.

Each entry in the matrix indicates the access rights of that subject for that object.

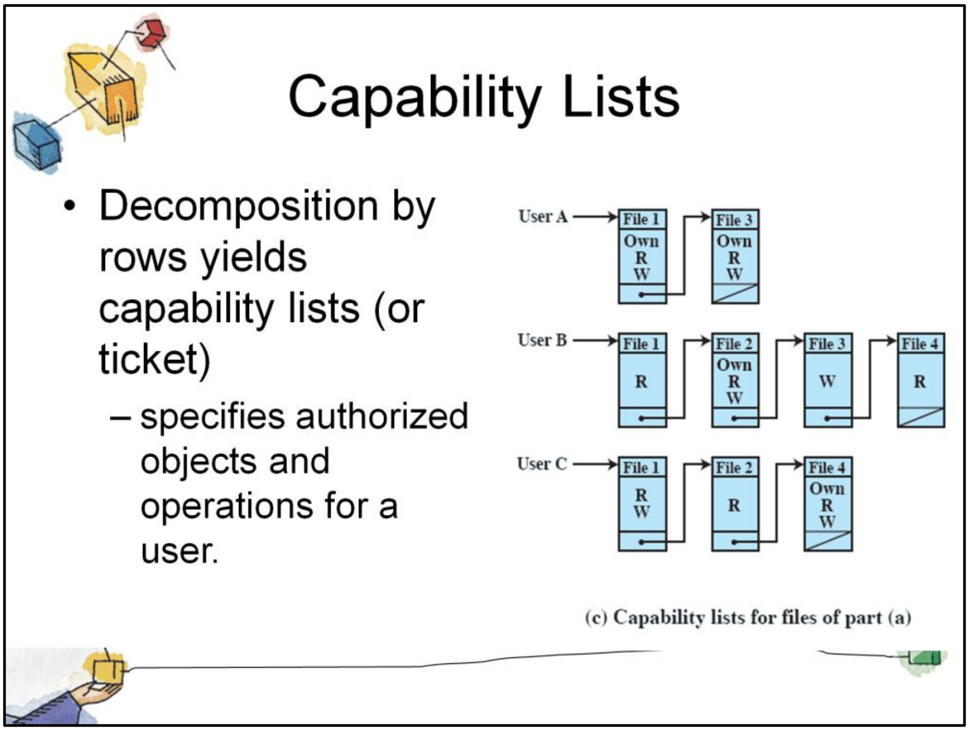


In practice, an access matrix is usually sparse and is implemented by decomposition in one of two ways. The matrix may be decomposed by columns, yielding access control lists

Thus for each object, an access control list lists users and their permitted access rights.

- The access control list may contain a default, or public, entry.
- This allows users that are not explicitly listed as having special rights to have a default set of rights.

Elements of the list may include individual users as well as groups of users.



Decomposition by rows yields capability tickets

A capability ticket specifies authorized objects and operations for a user.

- Each user has a number of tickets and may be authorized to loan or give them to others.

Because tickets may be dispersed around the system, they present a greater security problem than access control lists.

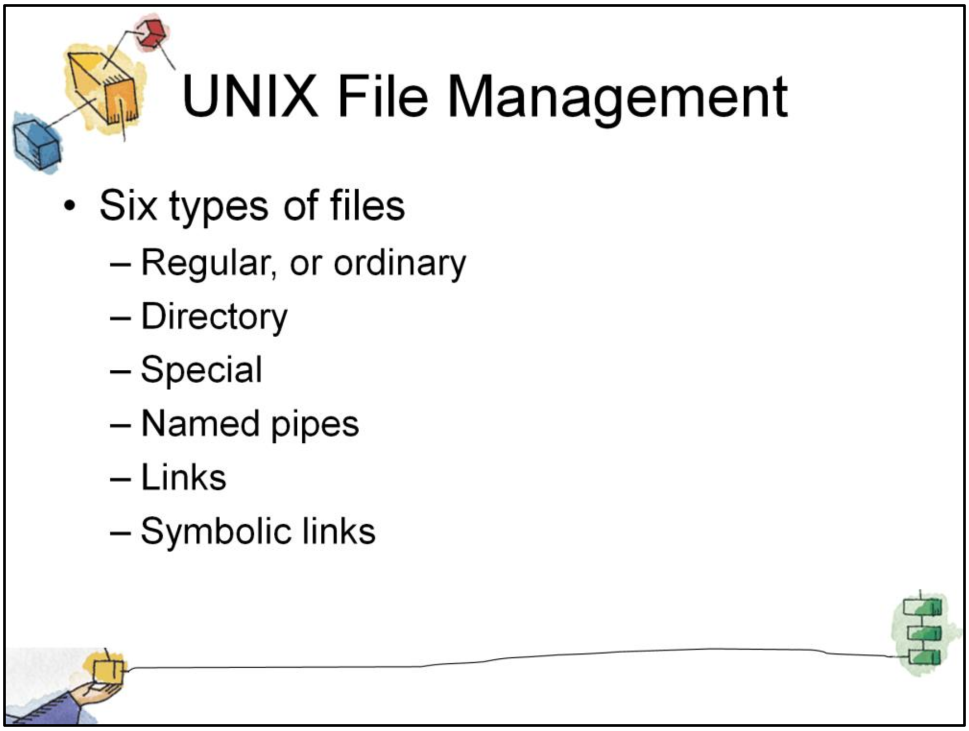
- In particular, the ticket must be unforgeable.

Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- File System Security

→ Unix File Management

- Linux Virtual File System
- Windows File System



Regular, or ordinary: Contains arbitrary data in zero or more data blocks.

- Regular files contain information entered in them by a user, an application program, or a system utility program.
- The file system does not impose any internal structure to a regular file but treats it as a stream of bytes.

Directory: Contains a list of file names plus pointers to associated inodes (index nodes)

- Directories are hierarchically organized
- Directory files are actually ordinary files with special write protection privileges so that only the file system can write into them, while read access is available to user programs.

Special: Contains no data, but provides a mechanism to map physical devices to file names.

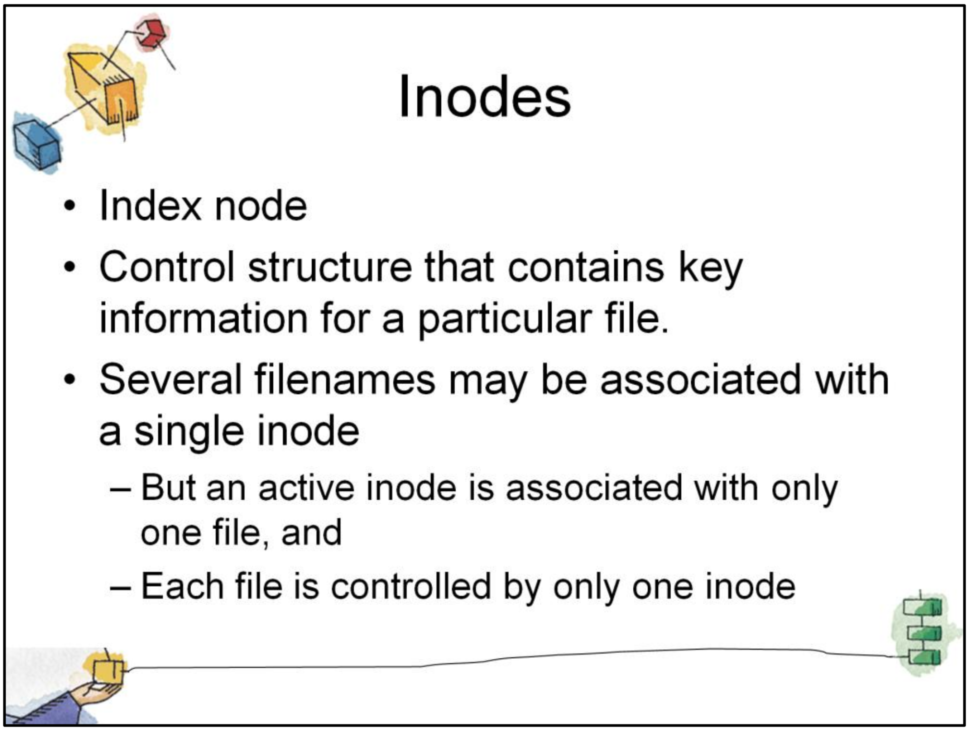
- The file names are used to access peripheral devices, such as terminals and printers.
- Each I/O device is associated with a special file.

Named pipes: A pipe is 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.

Links: A link is an alternative file name for an existing file.

Symbolic links: This is a data file that contains the name of the file it is linked to.



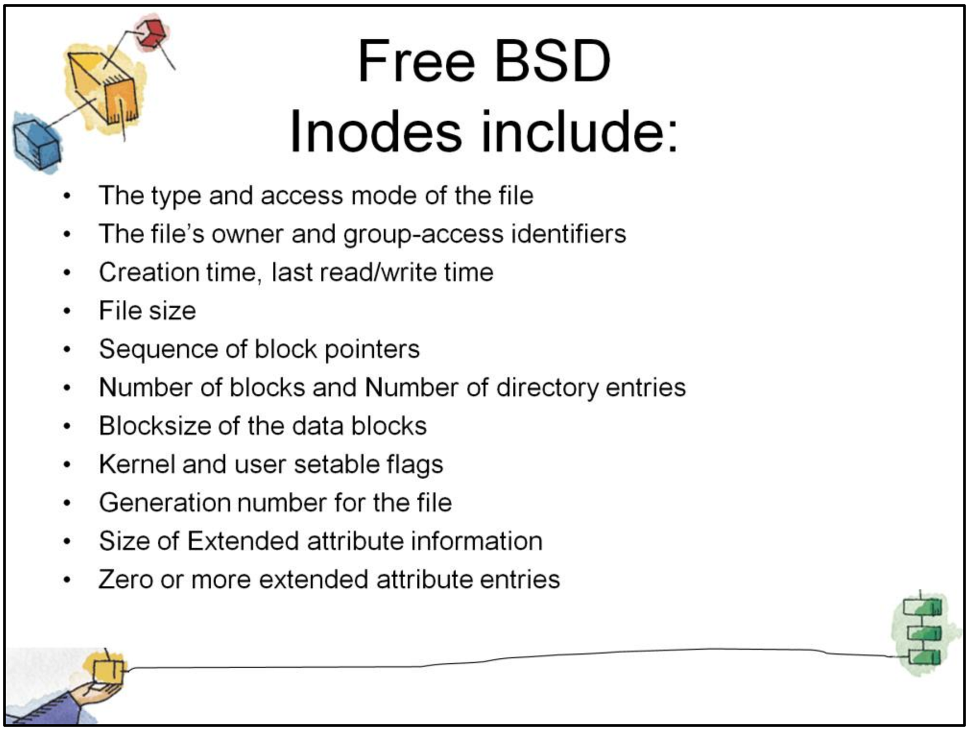
An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file.

Several file names may be associated with a single inode,

- but an active inode is associated with exactly one file, and each file is controlled by exactly one inode.

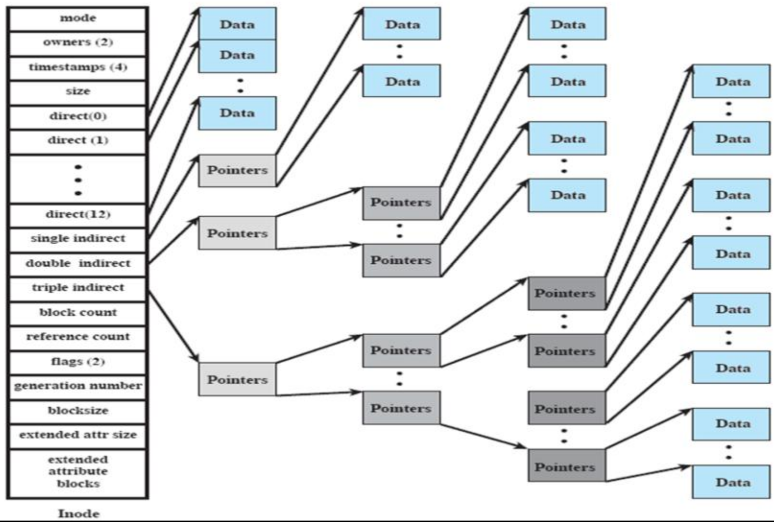
The attributes of the file as well as its permissions and other control information are stored in the inode.

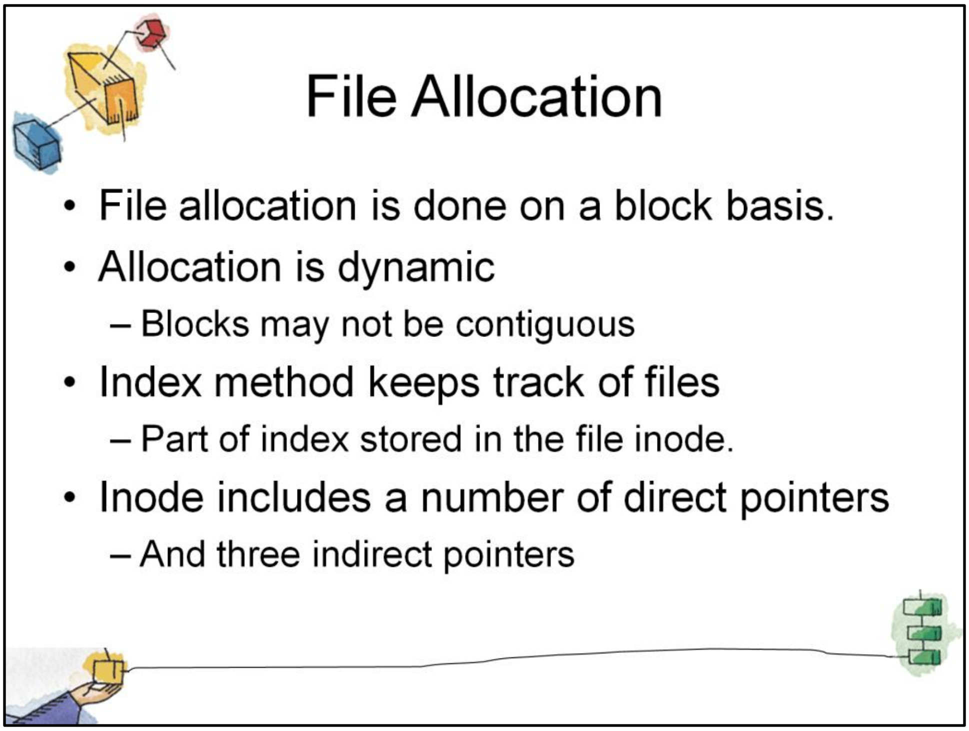
The exact inode structure varies from one UNIX implementation to another.



- 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 the reference the file
- The kernel and user setable 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

FreeBSD Inode and File Structure





File allocation is done on a block basis.

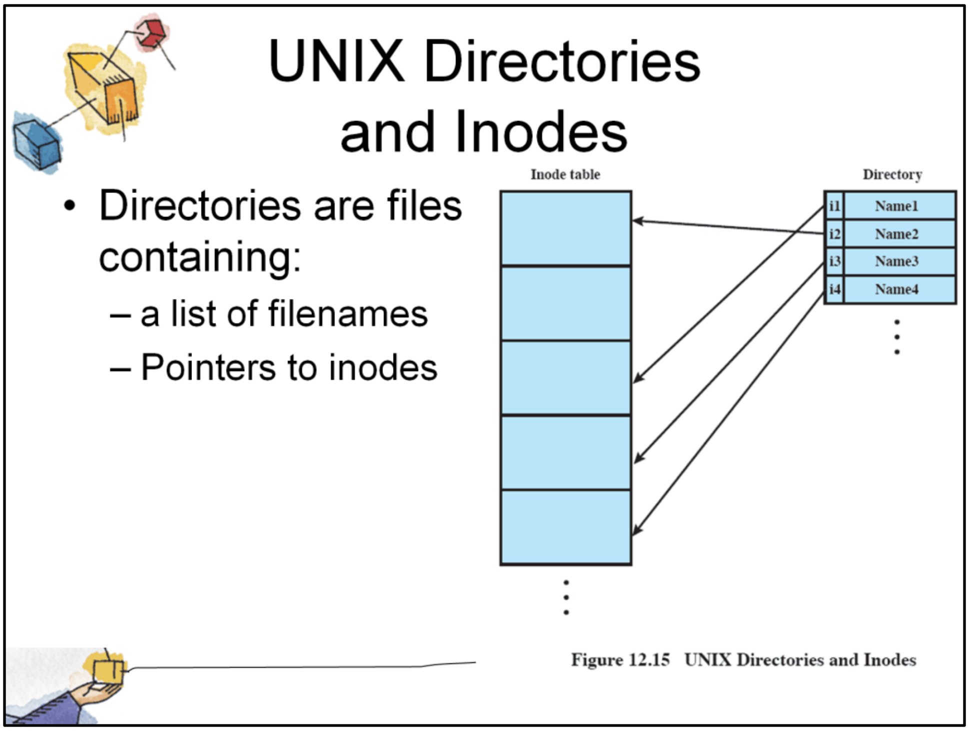
Allocation is dynamic, as needed, rather than using preallocation.

- the blocks of a file on disk are not necessarily contiguous.

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).



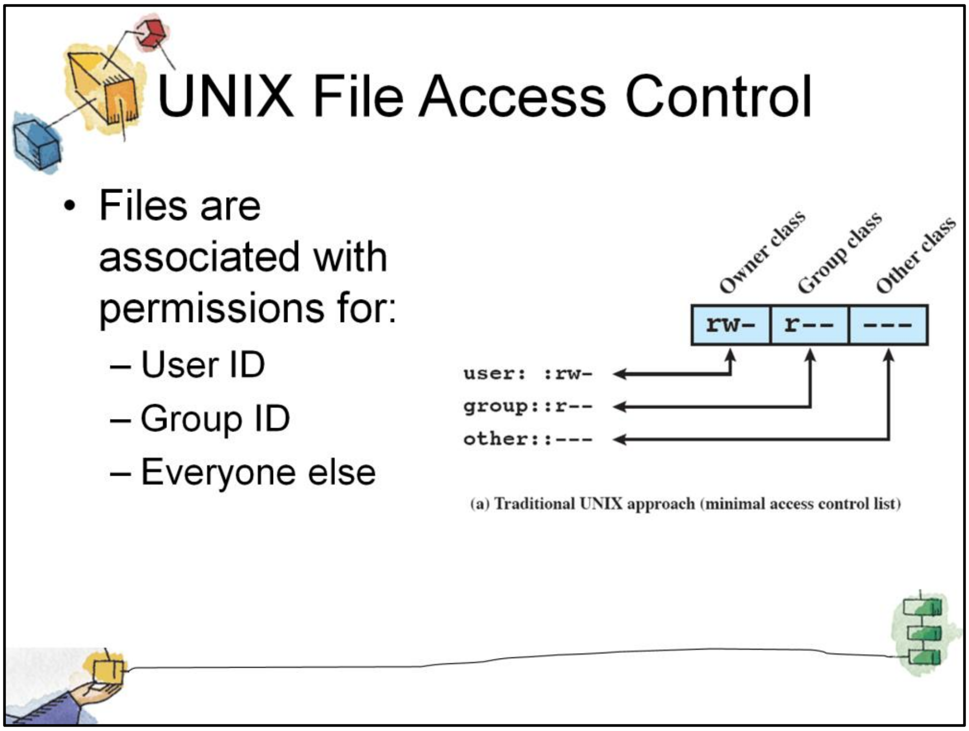
Directories are structured in a hierarchical tree.

- Each directory can contain files and/or other directories.
- A directory that is inside another directory is referred to as a subdirectory.

A directory is 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).

- When the file or directory is accessed, its i-number is used as an index into the inode table.



Each UNIX user is assigned a unique user identification number (user ID).

A user is also a member of a primary group, and possibly a number of other groups, each identified by a group ID.

When a file is created, it is designated as owned by a particular user and marked with that user's ID.

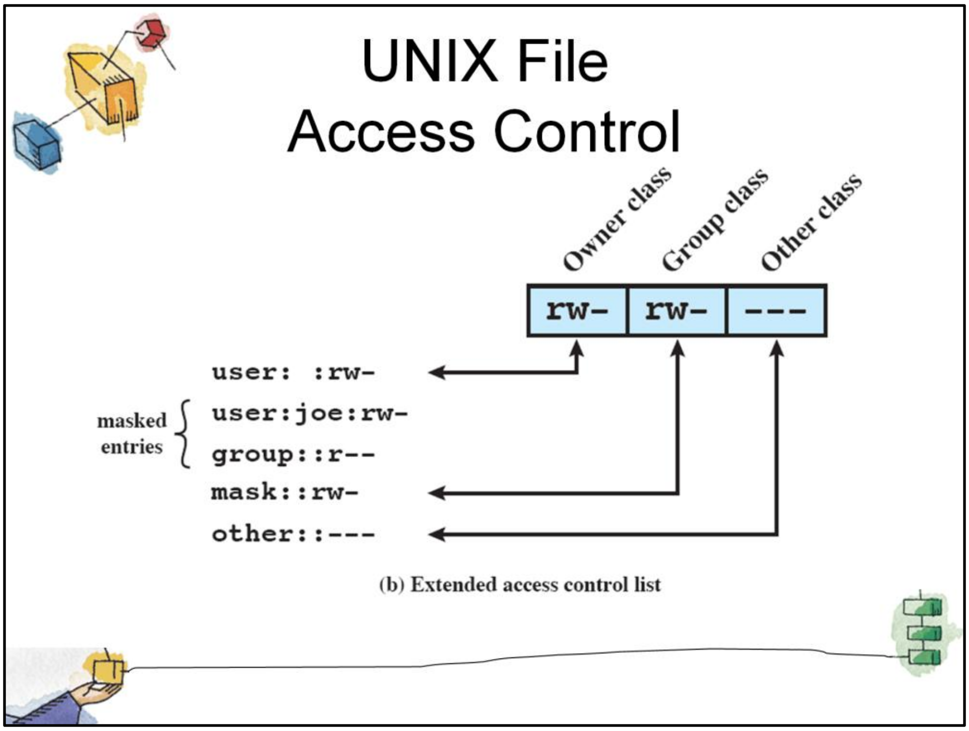
- It also belongs to a specific group, which initially is either its creator's primary group, or the group of its parent directory if that directory has SetGID permission set.

Associated with each file is a set of 12 protection bits.

- The owner ID, group ID, and protection bits are part of the file's inode.

Nine of the protection bits specify read, write, and execute permission for the owner of the file, other members of the group to which this file belongs, and all other users.

These form a hierarchy of owner, group, and all others, with the highest relevant set of permissions being used.



(b) Extended access control list

FreeBSD and most UNIX implementations that support extended ACLs use the following strategy:

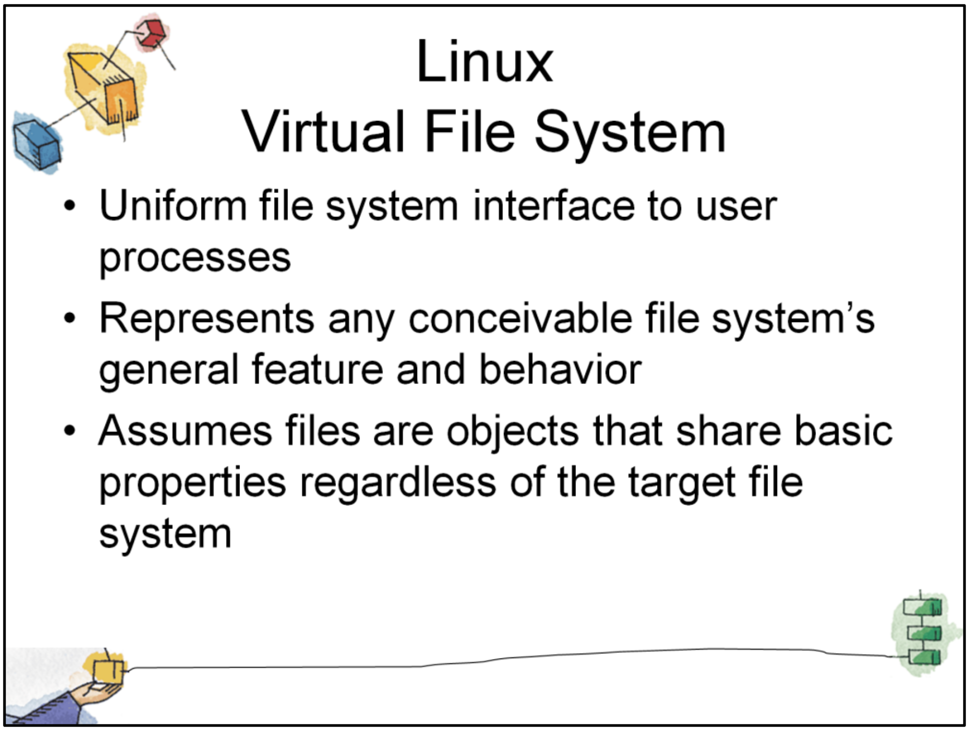
1. The owner class and other class entries in the 9-bit permission field have the same meaning as before

2. The group class entry specifies the permissions for the owner group for this file.
   - These permissions represent the maximum permissions that can be assigned to named users or named groups, other than the owning user.
   - In this latter role, the group class entry functions as a mask.

3. Additional named users and named groups may be associated with the file, each with a 3-bit permission field.
   - The permissions listed for a named user or named group are compared to the mask field.
   - Any permission for the named user or named group that is not present in the mask field is disallowed.

Roadmap

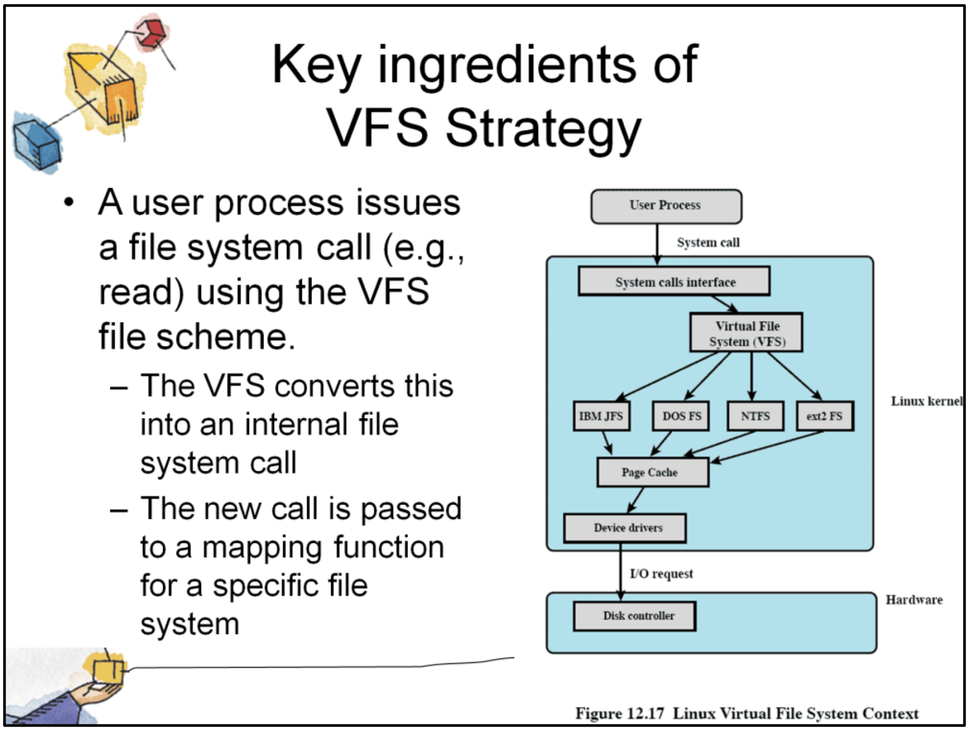
- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- **Linux Virtual File System**
- Windows File System



virtual file system (VFS), presents a single, uniform file system interface to user processes.

- The VFS defines a common file model that is capable of representing any conceivable file system's general feature and behavior.

The VFS assumes that files are objects in a computer's mass storage memory that share basic properties regardless of the target file system or the underlying processor



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)].

In most cases, the mapping function is simply a mapping of file system functional calls from one scheme to another.

The role of VFS within the Kernel

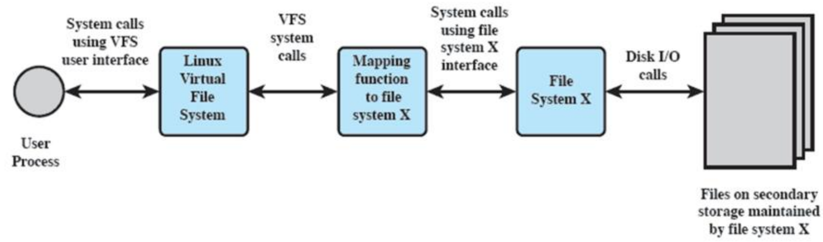


Figure 12.18 Linux Virtual File System Concept

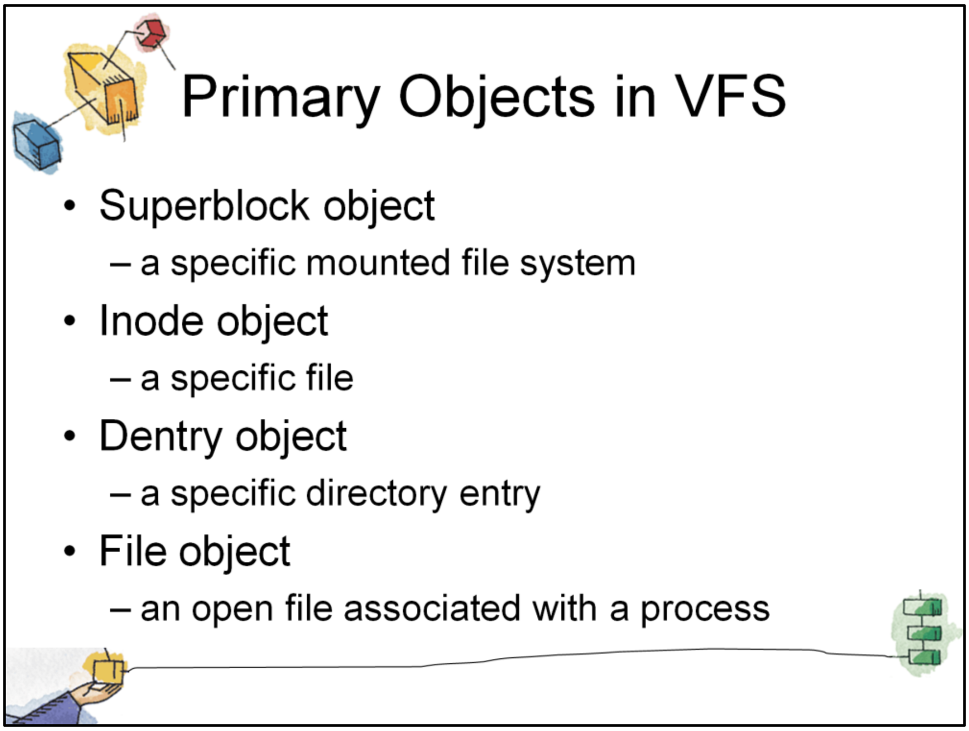
This figure indicates the role that VFS plays within the Linux kernel.

When a process initiates a file-oriented system call (e.g., read), the kernel calls a function in the VFS.

- This function handles the file-system-independent manipulations and initiates a call to a function in the target file system code.
- This call passes through a mapping function that converts the call from the VFS into a call to the target file system.

The VFS is independent of any file system, so the implementation of a mapping function must be part of the implementation of a file system on Linux.

The target file system converts the file system request into device-oriented instructions that are passed to a device driver by means of page cache functions.



VFS is an object-oriented scheme.

- VFS objects are implemented simply as C data structures.

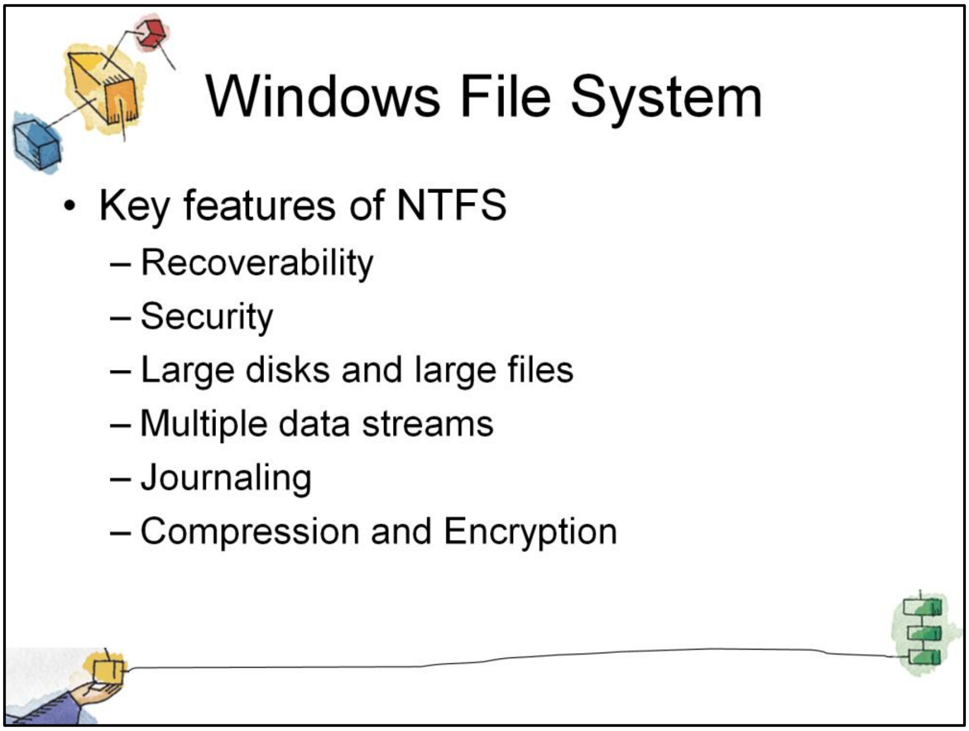
Each object contains both data and pointers to file-system-implemented functions that operate on data.

The four primary object types in VFS are as follows:

- **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

Roadmap

- Overview
- File organisation and Access
- File Directories
- File Sharing
- Record Blocking
- Secondary Storage Management
- File System Security
- Unix File Management
- Linux Virtual File System
- Windows File System



Recoverability: the ability to recover from system crashes and disk failures.

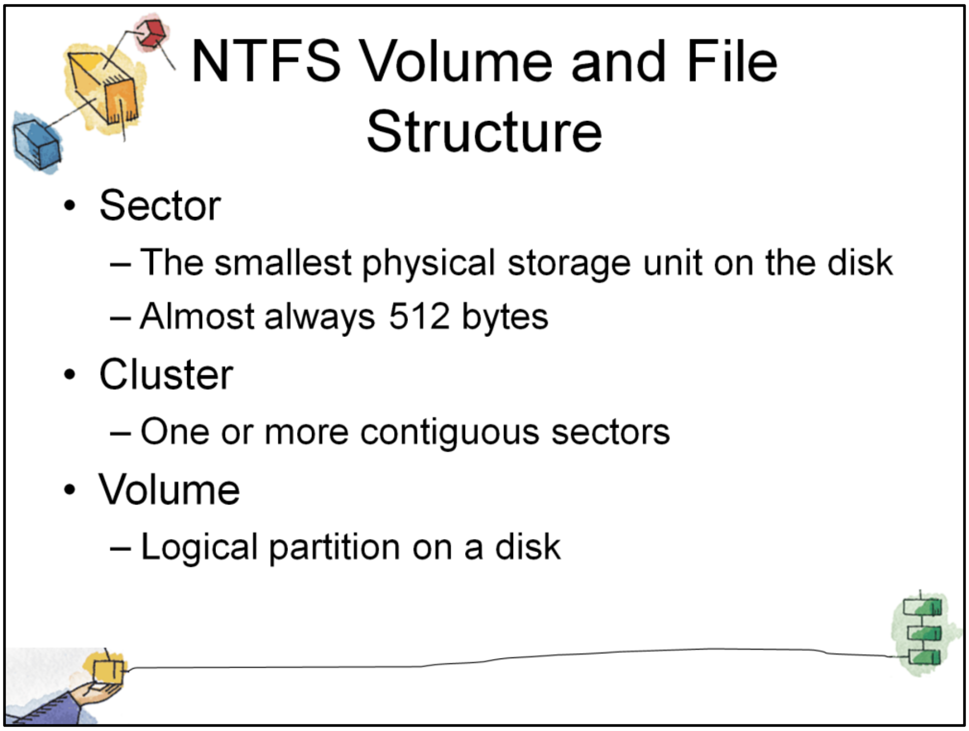
Security: NTFS uses the Windows object model to enforce security.

Large disks and large files: NTFS supports very large disks and very large files more efficiently than most other file systems, including FAT.

Multiple data streams: The actual contents of a file are treated as a stream of bytes.

Journaling: NTFS keeps a log of all changes made to files on the volumes.

Compression and Encryption: Entire directories and individual files can be transparently compressed and/or encrypted.



Sector: The smallest physical storage unit on the disk.

- almost always 512 bytes.

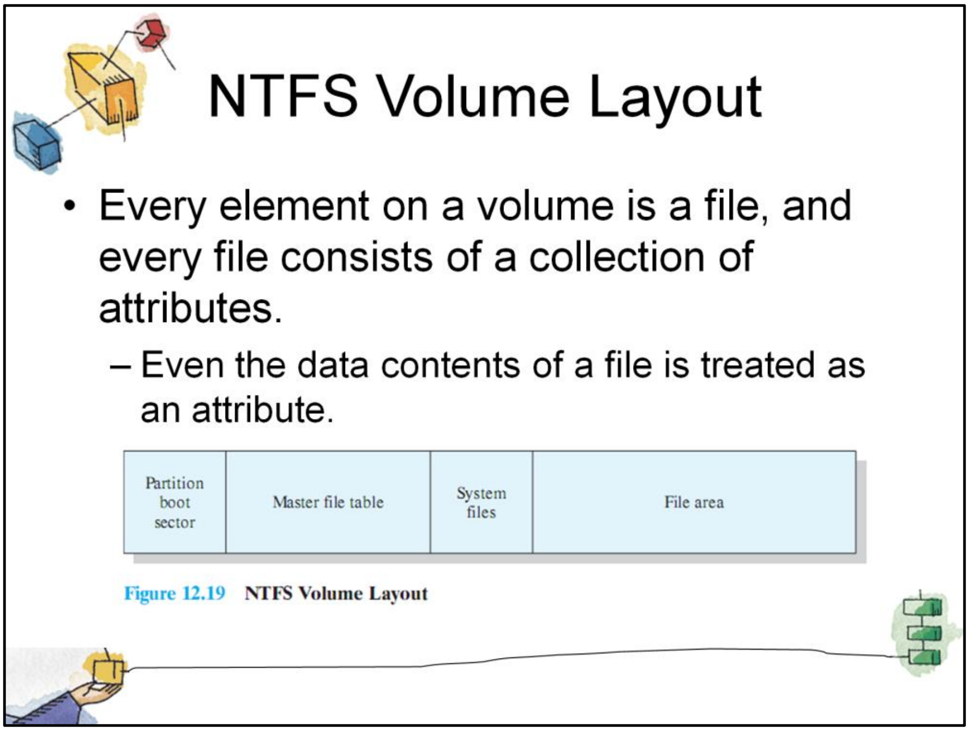
Cluster: One or more contiguous (next to each other on the same track) sectors.

Volume: A logical partition on a disk, consisting of one or more clusters and used by a file system to allocate space.

- At any time, a volume consists of a file system information, a collection of files, and any additional unallocated space remaining on the volume that can be allocated to files.
- A volume can be all or a portion of a single disk or it can extend across multiple disks.
- If hardware or software RAID 5 is employed, a volume consists of stripes spanning multiple disks.
- The maximum volume size for NTFS is 2^{64} bytes.

Efficient with Large Files

Volume Size	Sectors per Cluster	Cluster Size
512Mbyte	1	512bytes
512Mbyte – 1 Gbyte	2	1K
1–2 Gbyte	4	2K
2–4 Gbyte	8	4K
4–8 Gbyte	16	8K
8–16 Gbyte	32	16K
16–32 Gbyte	64	32K
>32Gbyte	128	64K



The first few sectors on any volume are occupied by the partition boot sector which contains information about the volume layout and the file system structures as well as boot startup information and code.

master file table (MFT) contains information about all of the files and folders (directories) on this NTFS volume.

- The MFT is a list of all files and their attributes on this NTFS volume, organized as a set of rows in a relational database structure.

system files. typically about 1 Mbyte in length includes:

- MFT2: A mirror of the first three rows of the MFT, used to guarantee access to the MFT in the case of a single-sector failure
- Log file: A list of transaction steps used for NTFS recoverability
- Cluster bit map: A representation of 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

Windows NTFS Components

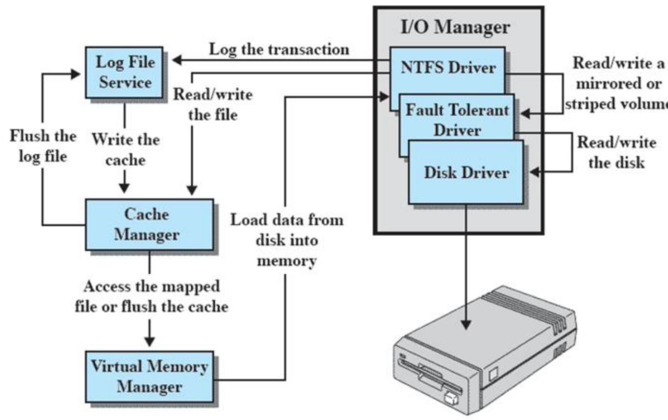


Figure 12.20 Windows NTFS Components

I/O manager: Includes the NTFS driver, which handles the basic open, close, read, write functions of NTFS.

Log file service: Maintains a log of file system metadata changes on disk.

- The log file is used to recover an NTFS-formatted volume in the case of a system failure (i.e., without having to run the file system check utility).

Cache manager: Responsible for caching file reads and writes to enhance performance.

- The cache manager optimizes disk I/O.

Virtual memory manager: The NTFS accesses cached files by mapping file references to virtual memory references and reading and writing virtual memory.