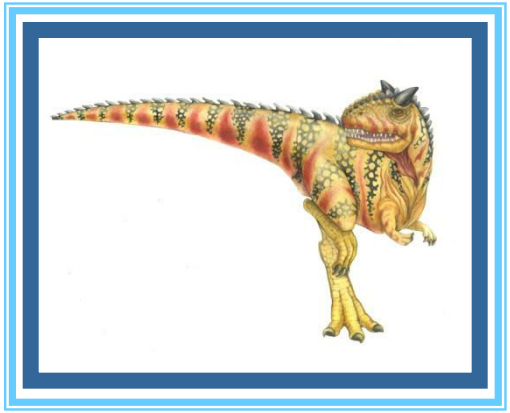
Chapter 11
File-System Interface



Chapter 11: File-System Interface

- *File Concept*
- *Access Methods*
- *Disk and Directory Structure*
- *File-System Mounting*
- *File Sharing*
- *Protection*

Objectives

- *To explain the function of file systems*
- *To describe the interfaces to file systems*
- *To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures*
- *To explore file-system protection*

File Concept

- *A file is a **named collection** of related information that is recorded on **secondary storage**.*
- *From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.*

File Concept

- *Commonly, files represent Types:*
  - ***Data***
    - *numeric*
    - *Alphabetic*
    - *Alphanumeric*
    - *binary*
  - ***Program (both source and object forms)***
- *The information in a file is defined by its creator.*
- *Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on.*

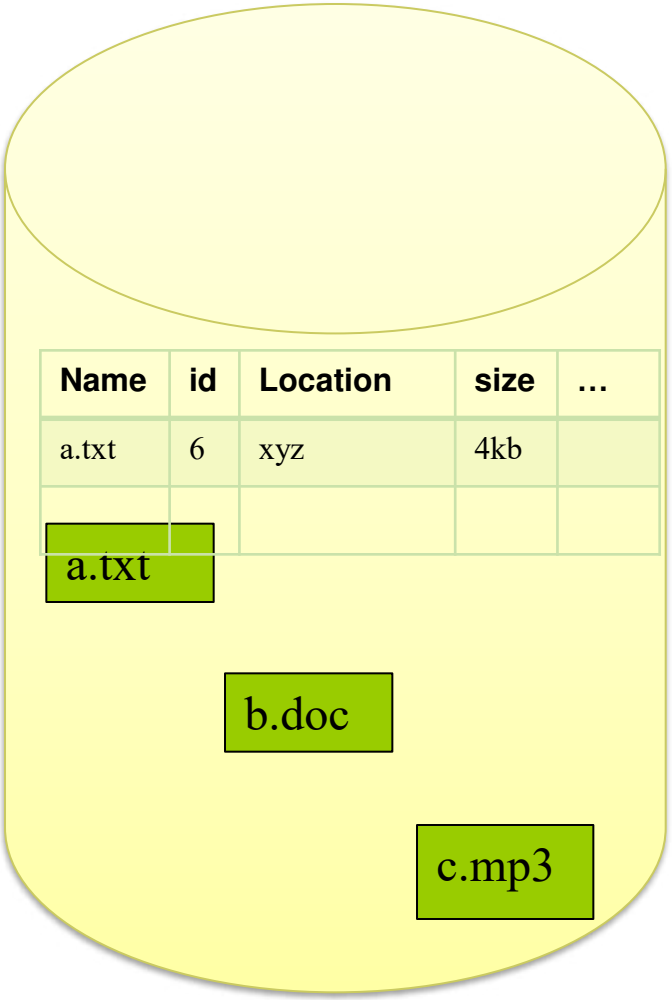
File Concept

- *A file has a **certain defined structure**, which depends on its type.*
- *A **text file** is a sequence of characters organized into lines (and possibly pages).*
- *A **source file** is a sequence of functions, each of which is further organized as declarations followed by executable statements.*
- *An **executable file** is a series of code sections that the loader can bring into memory and execute.*

File Attributes

- ***Name*** *– only information kept in human-readable form*
- ***Identifier*** *– unique tag (number) identifies file within file system*
- ***Type*** *– needed for systems that support different types*
- ***Location*** *– pointer to file location on device*
- ***Size*** *– current file size*
- ***Protection*** *– controls who can do reading, writing, executing*
- ***Time, date, and user identification*** *– data for protection, security, and usage monitoring.*
- *Information about files are kept in the directory structure, which is maintained on the disk Many variations, including extended file attributes such as file checksum.*
- *Information kept in the directory structure.*

File Attributes



File info Window on Mac OS X

File Attributes

- *The information about all files is kept in the **directory structure**, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier.*
- *The identifier in turn locates the other file attributes.*

File Operations

*File is an **abstract data type.** The operating system can provide system calls to*

- ***Create***
- ***Write** – at **write pointer** location*
- ***Read** – at **read pointer** location*
- ***Reposition within file** - **seek***
- ***Delete***
- ***Truncate***
- ***Other Operations :** appending, renaming, create copy*
- ***Open(Fi)***
- ***Close (Fi)** .*

File Operations

- *A file is an **abstract data type**. To define a file properly, we need to consider the operation performed on the file.*
- *Operating System can provide system calls to create, write, read, reposition, delete and truncate files.*

1. ***Creating a File:** Creation of a file is the first operation. For creating a file two steps are necessary.*
    - *Required space must be allocated.*
    - *An entry for new file must be made in the directory.*
    - *The directory entry records the name of the file and the location in the file system.*

File Operations

2. Writing a File:

- *For file writing operation, we make a system call specifying both the name of a file and the information to be written into the file.*
- *System searches the entire directory structure to find the location of the specified file.*
- *System keeps a write pointer that keeps track of writing at the location in the file.*
- *The system updates write pointer each time whenever a file write operation occurs.*

File Operations

3. **Reading a File:**
   - *To perform file read operation, we use a system call that specifies name of the file and the block of the file to read from it.*
   - *Again the directory is searched for the specified file.*
   - *System keeps a read pointer that keeps track of reading location in the file.*
   - *The system updates read pointer each time whenever a file read operation occurs*
4. **Repositioning Within a File**
   - *Repositioning within a file operation does not involve any actual input output.*
   - *The directory is searched for the appropriate entry and the current file position is set to a given value.*
   - *It is also known as files seek operation.*

File Operations

5. ***Deleting a File:***
   - *File deletion operation also requires searching of a specified file entry within the directory structure.*
   - *As soon as the file is deleted, space allocated to that file becomes available for further use.*
6. ***Truncating a File***
   - *In some cases the user may want to erase the file contents but keep its attributes as it is. This operation is called truncating a file.*
   - *Instead of delete a file and recreating it with same attributes, this function allows all attributes to remain unchanged except the file content.*
   - *File length attribute is reset to a length zero and its file space is released.*

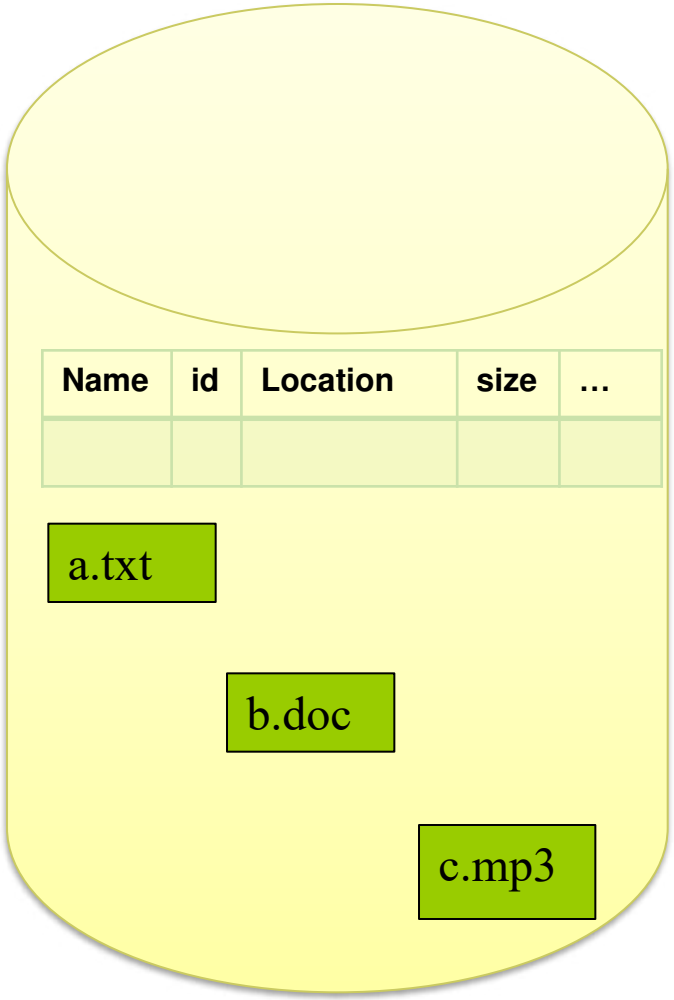
Open Files

- *The OS keeps a small table, called the **open-file table**, containing information about all **open files**.*
  - *When a file operation is requested, the file is specified via an **index** into this table, so no searching is required.*
  - *When the file is no longer being actively used, it is **closed** by the process, and the OS removes its entry from the open-file table.*
- *Most systems require that the programmer open a file explicitly with the **open()** system call before that file can be used.*
- *The open() operation takes a file name and searches the directory, copying the directory entry into the open-file table.*

File Attributes

a.txt		

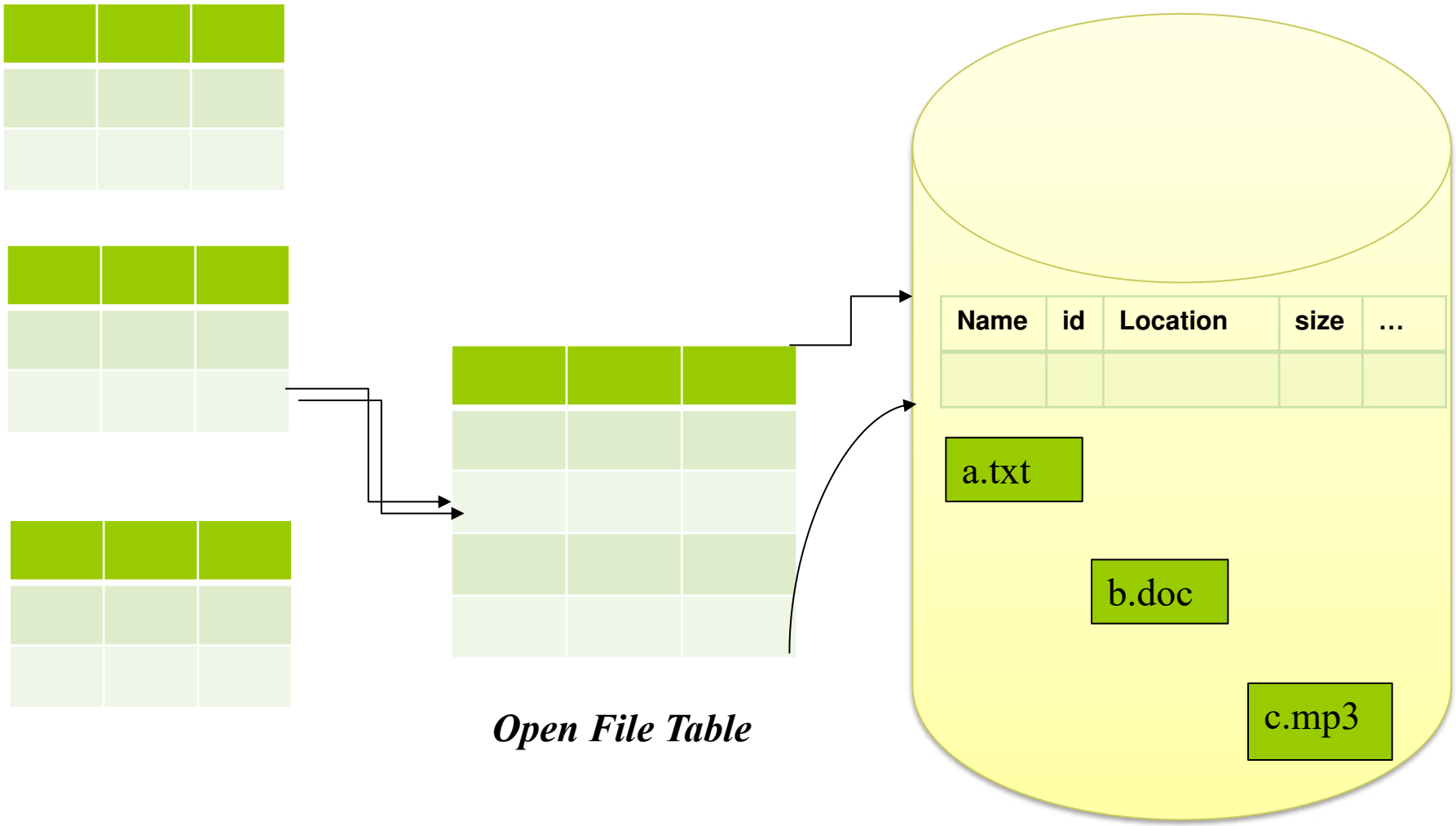
Open File Table



Open Files

- *Several processes may open the file where several processes may open the file simultaneously.*
- *Typically, the operating system uses two levels of internal tables:* ***a per-process table*** *and* ***a system-wide table****.*
  - ***System-wide table:*** *stores process-independent information, such as the location of the file on disk, access dates, and file size.*
  - ***Per-process table:*** *tracks all files that a process has open, Store information regarding the process's use of the file Example: the current file pointer for each file is found here, Access rights to the file and accounting, information.*
    - *Each entry in the per-process table in turn points to a system-wide open-file table*

File Attributes



Open Files

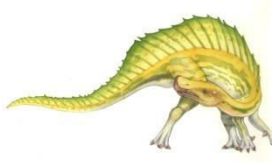
- *Once a file has been opened by one process, the system-wide table includes an entry for the file. When another process executes an* ***open()*** *call, a new entry is simply added to the process's open-file table pointing to the appropriate entry in the system-wide table.*
- ***Open count:*** *Typically, the open-file table also has an open count associated with each file to indicate how many processes have the file open. Each close() decreases this open count, and when the open count reaches zero, the file is no longer in use, and the file's entry is removed from the open-file table.*

Open Files

- *In summary, Several pieces of data are needed to manage open files:*
  - ***File Pointer:*** *Track the last read–write location per process that has the file open*
  - ***Open-file table****: tracks open files*
  - ***File-open count****: counter of number of times a file is open.*
  - ***Disk location of the file:*** *cache of data access information*
  - ***Access rights:*** *per-process access mode information*

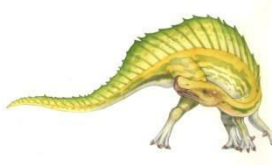
Open File Locking

- *File locks are provided by some operating systems and file systems allow one process to lock a file and prevent other processes from gaining access to it.*
  - *Similar to reader-writer locks*
  - ***Shared lock** similar to reader lock – several processes can acquire concurrently*
  - ***Exclusive lock** similar to writer lock*
- *Mandatory or advisory file locking mechanisms:*
  - ***Mandatory** – access is denied depending on locks held and requested, the operating system ensures locking integrity.*
  - ***Advisory** – it is up to software developers to ensure that locks are appropriately acquired and released. Processes can find status of locks and decide what to do*



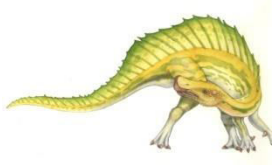
File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
      public static final boolean EXCLUSIVE = false;
      public static final boolean SHARED = true;
      public static void main(String arsg[]) throws IOException {
            FileLock sharedLock = null;
            FileLock exclusiveLock = null;
            try {
                  RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
                  // get the channel for the file
                  FileChannel ch = raf.getChannel();
                  // this locks the first half of the file - exclusive
                  exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
                  /** Now modify the data . . . */
                  // release the lock
                  exclusiveLock.release();
```



File Locking Example – Java API (Cont.)

```
                    // this locks the second half of the file - shared
                    sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                                         SHARED);
                    /** Now read the data . . . */
                    // release the lock
                    sharedLock.release();
            } catch (java.io.IOException ioe) {
                    System.err.println(ioe);
            }finally {
                    if (exclusiveLock != null)
                    exclusiveLock.release();
                    if (sharedLock != null)
                    sharedLock.release();
            }
        }
    }
```



File Types – Name, Extension

- *The name is split into two parts—a name and an extension. Examples include resume.docx, server.c, and ReaderThread.cpp.*

- *The system uses the extension to indicate the type of the file and the type of operations that can be done on that file. Example: a .com, .exe, or .sh extension can be executed.*
- *Extensions are not supported by the operating system, they can be considered "hints" to the applications that operate on them.*

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- *File types also can be used to indicate the internal structure of the file. Files have structures that **match the expectations** of the programs that read them.*
- *Further, certain files must conform to a required structure that is understood by the **operating system**.*
- *Some operating systems extend this idea **into a set of system-supported file structures**, with sets of special operations for manipulating files with those structures.*
- *File structure is the organization of the data in secondary storage in such a way that it can minimize the access time and storage space.*
- ***File structure = file data + accessing operations***

File Structure

- ***Disadvantages*** *of having the operating system support multiple file structures:*
- *The resulting size of the operating system is* ***cumbersome****.*
- *In addition, it may be necessary to define every file as one of the file types supported by the operating system. When new applications require information structured in ways not supported by the operating system, severe problems may result.*
- *Ex: For example, assume that a system supports two types of files: text files and executable binary files. Now, if we (as users) want to define an encrypted file we may find neither file type to be appropriate.*

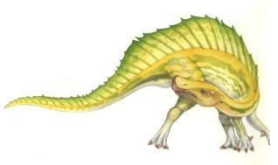
File Structure

- *A file have different kind of structure*
  - *None - sequence of words, bytes*
  - *Simple record structure*
    - *Lines of Fixed length*
    - *Lines of Variable length*
  - *Complex Structures*
    - *Formatted document*
    - *Relocatable load file*

File Access Methods

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

1. *Sequential access*
2. *Direct/Random access*
3. *Indexed sequential access*

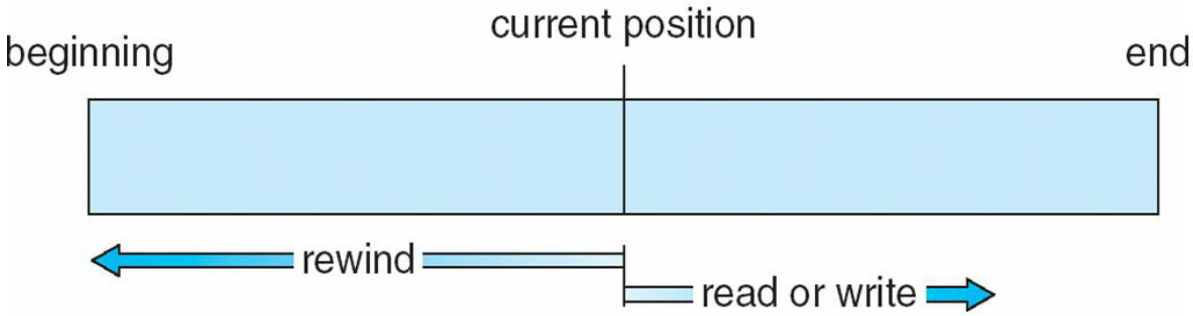


File Access Methods

1. ***Sequential Access***: *With sequential access, the device must move through all information up to the location where it is attempting to read or write. Based on Tape model.*

- *Data is accessed one record right after another in an order.*
- *Reads and writes make up the bulk of the operations on a file.*
- *Read command cause a pointer to be moved ahead by one or a read operation—read next()—reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.*
- *Similarly, the write operation—write next()—appends to the end of the file and advances to the end of the newly written material (the new end of file)..*
- *This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.*
- *A file can be reset to the beginning, and on some systems, a program may be able to skip forward or backward n records for some integer n—perhaps only for n = 1. Sequential access.*

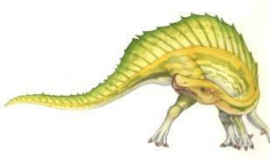
Sequential-access File



File Access Methods

2. Direct Access Methods: *Another method is* ***direct access*** *(or* ***relative access****).*

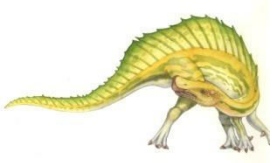
- *Allow programs to read and write records rapidly in no particular order.*
- *The direct-access method is based on a disk model of a file, since disks allow random access to any file block.*
- *For direct access the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.*
- *For the direct-access method, the file operations must be modified to include the block number as a parameter.*
- *Thus, we have read(n), where n is the block number, rather than read next(), and write(n) rather than write next().*



File Access Methods

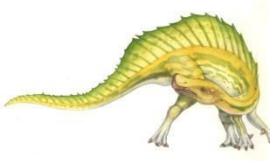
3. Other Access Methods

- *These methods generally involve the construction of an index for the file. The index, like an index in the back of a book, contains pointers to the various blocks.*
- *To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.*
- *For example, IBM's* ***indexed sequential-access method (ISAM)*** *uses a small master index that points to disk blocks of a secondary index*



File Access Methods

To find a particular item, we first make a binary search of the master index, which provides the block number of the secondary index. This block is read in, and again a binary search is used to find the block containing the desired record. Finally, this block is searched sequentially. In this way, any record can be located from its key by at most two direct-access reads.



File Access Methods

- ***Sequential Access***

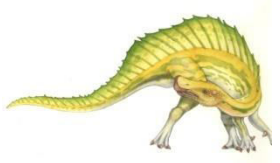
  read next
  write next
  reset
  no read after last write
  (rewrite)

- ***Direct Access*** – *file is fixed length logical records*

  read n
  write n
  position to n
  read next
  write next
  rewrite n

  n = relative block number

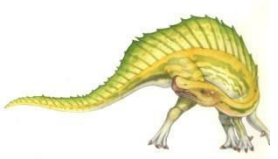
- *Relative block numbers allow OS to decide where file should be placed*
  - *See allocation problem*



Simulation of Sequential Access on Direct-access File

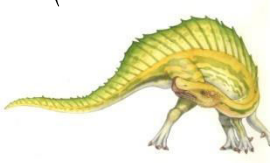
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

We can easily simulate sequential access on a direct-access file by simply keeping a variable *cp* that defines our current position

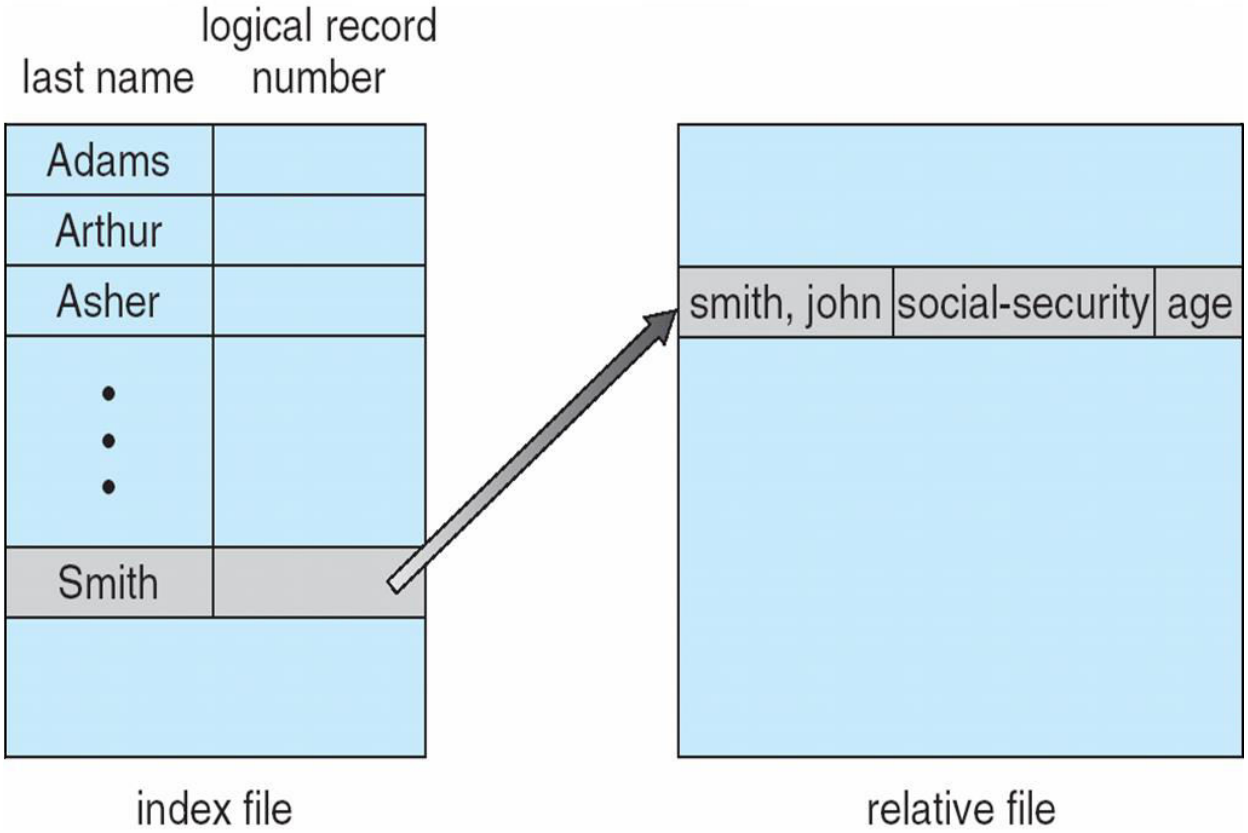


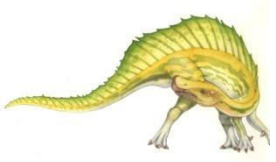
Other Access Methods

- *Can be built on top of base methods*
- *General involve creation of an index for the file*
- *Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)*
- *If too large, index (in memory) of the index (on disk)*
- *IBM indexed sequential-access method (ISAM)*
  - *Small master index, points to disk blocks of secondary index*
  - *File kept sorted on a defined key*
  - *All done by the OS*
- *VMS operating system provides index and relative files as another example (see next slide)*



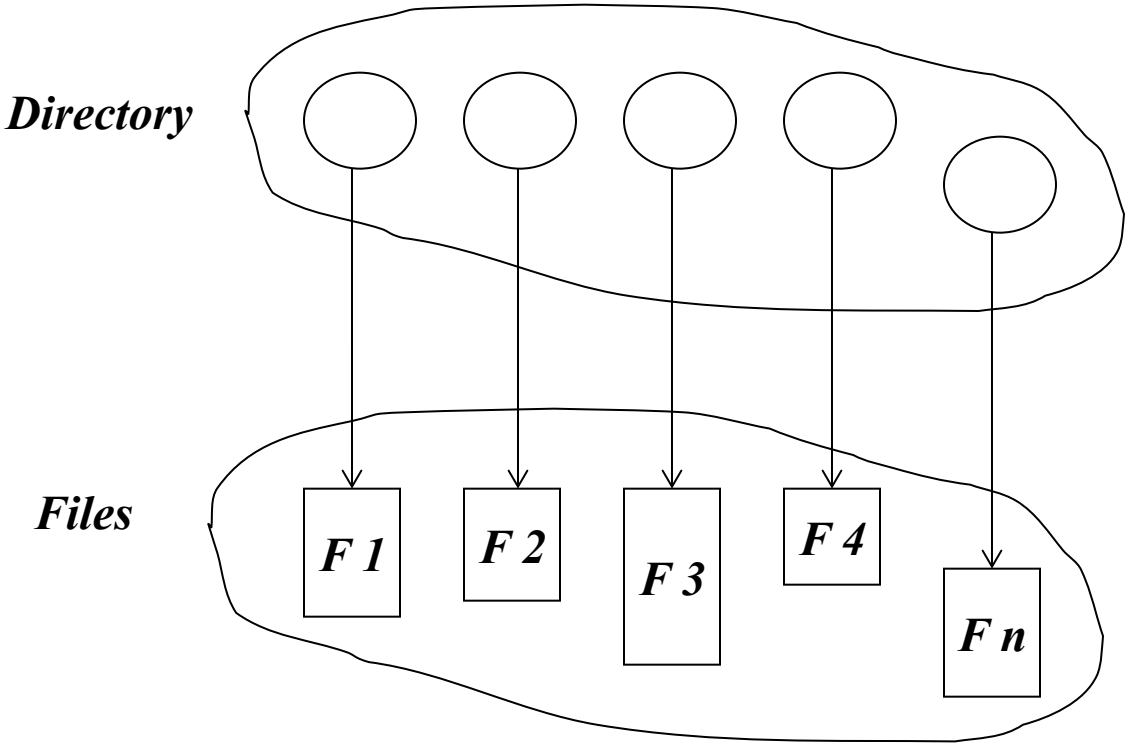
Example of Index and Relative Files



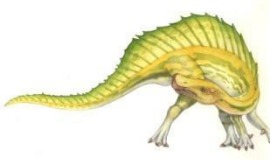


Directory Structure

- *A collection of nodes containing information about all files*

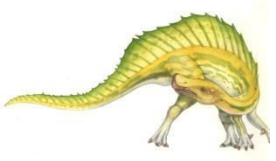


- *Both the directory structure and the files reside on disk*

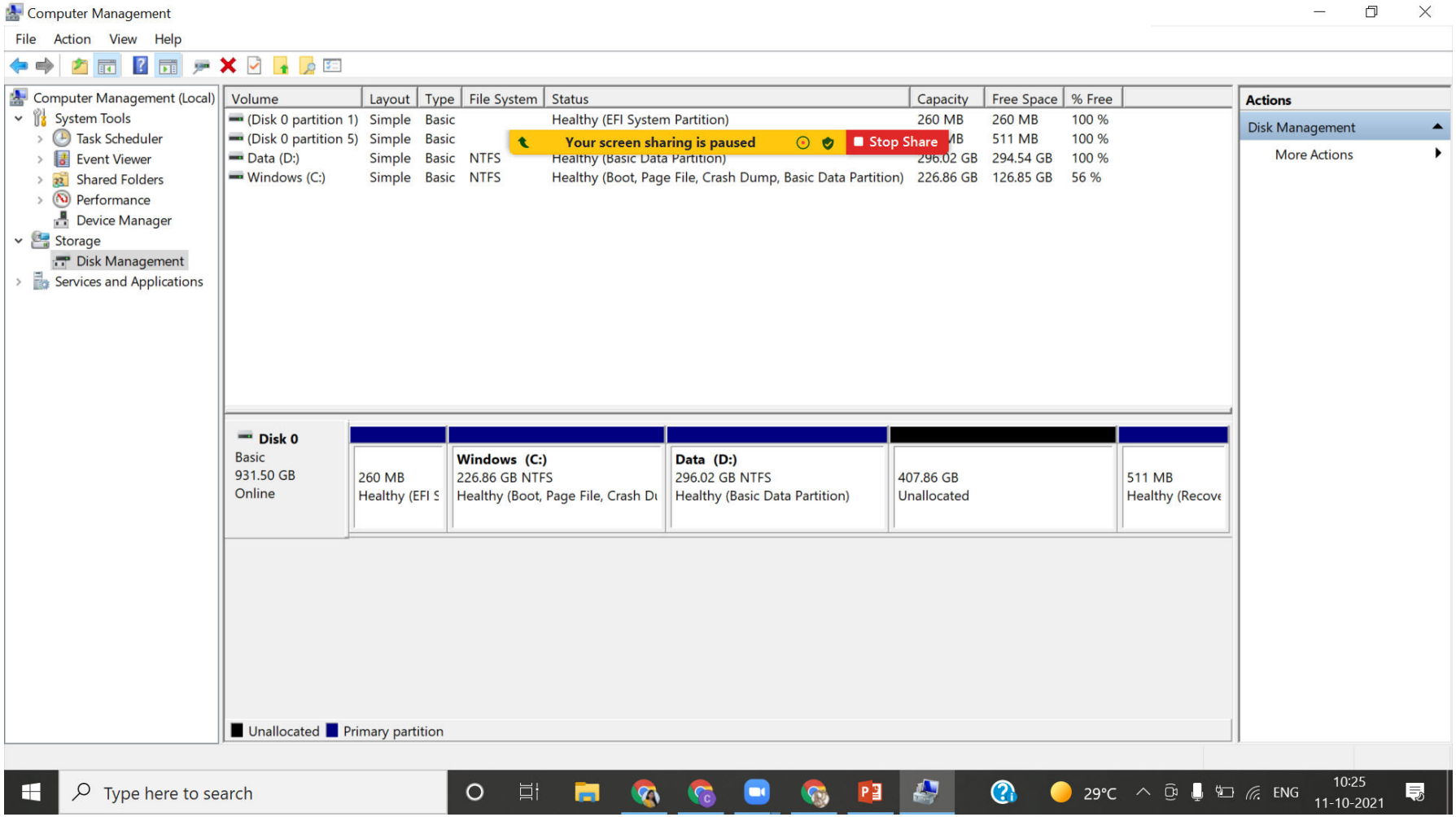


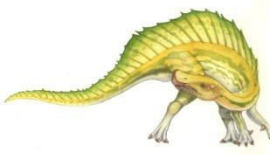
Disk Structure

- *Files are stored on **random-access storage devices**, including **hard disks, optical disks, and solid-state (memory-based) disks.***
- *A storage device can also be subdivided for **finer-grained control**. For example, a disk can be **partitioned** and each partition can hold a **separate file system.***
- *Storage devices can also be collected together into **RAID (Redundant Arrays of Independent Disks)** sets that provide protection from the failure of a **single disk**.*
- ***Partitioning** is useful:*
  - *Limiting the sizes of individual file systems*
  - *Putting multiple file-system types on the same device*
  - *Leaving part of the device available for other uses, such as swap space or unformatted (raw) disk space.*

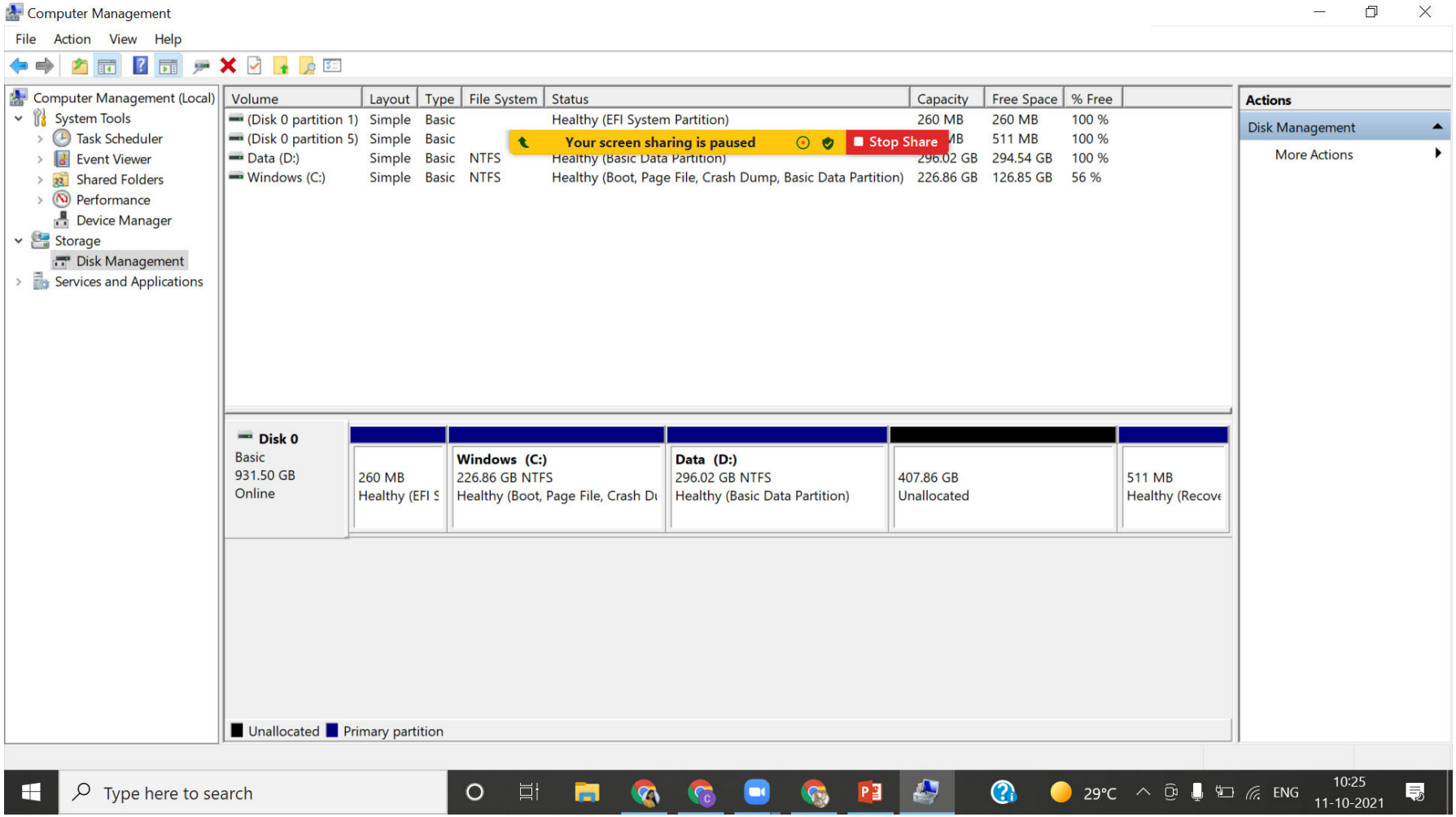


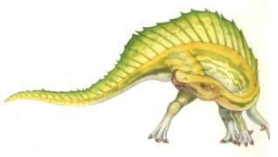
Disk Structure



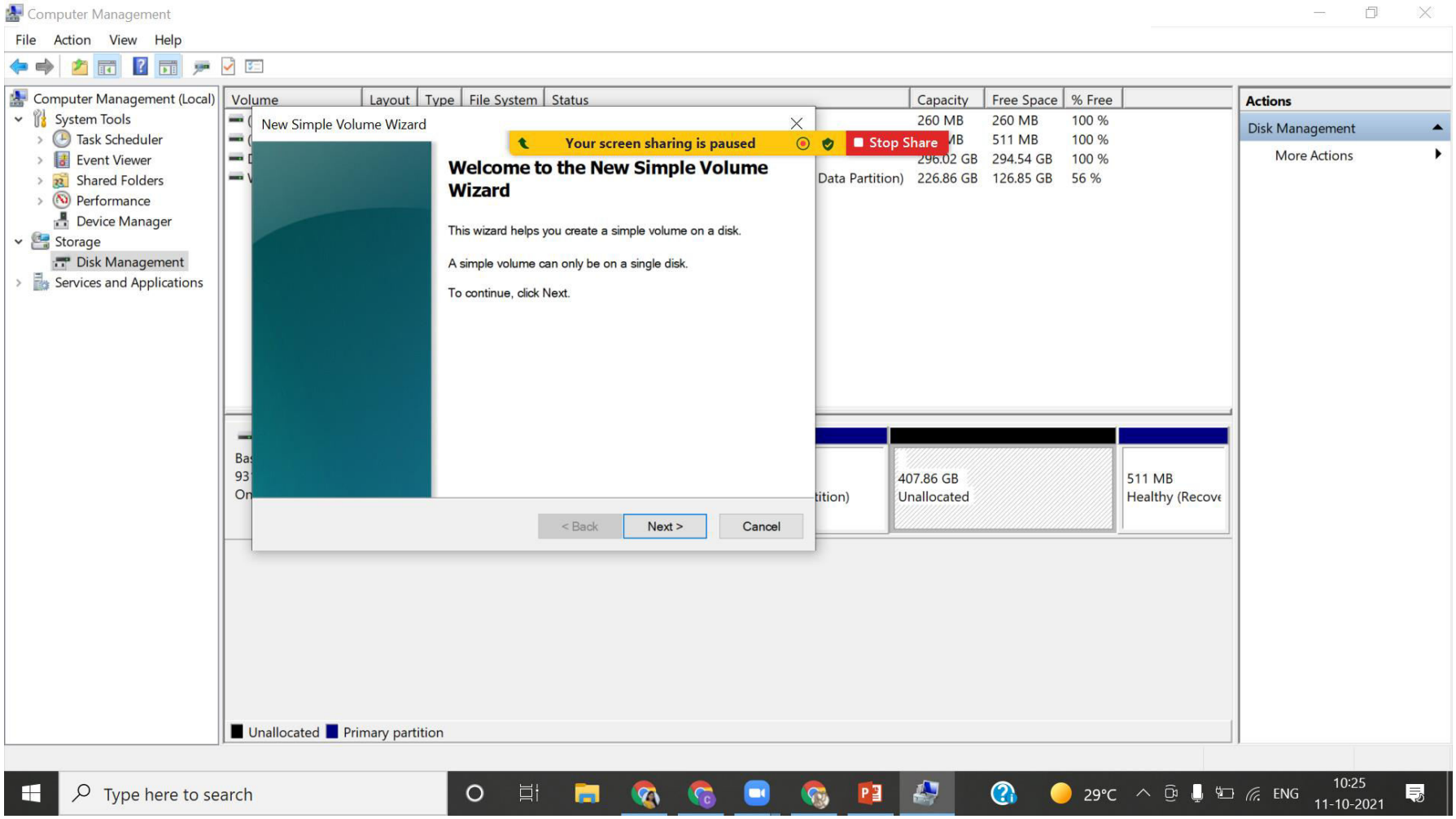


Disk Structure

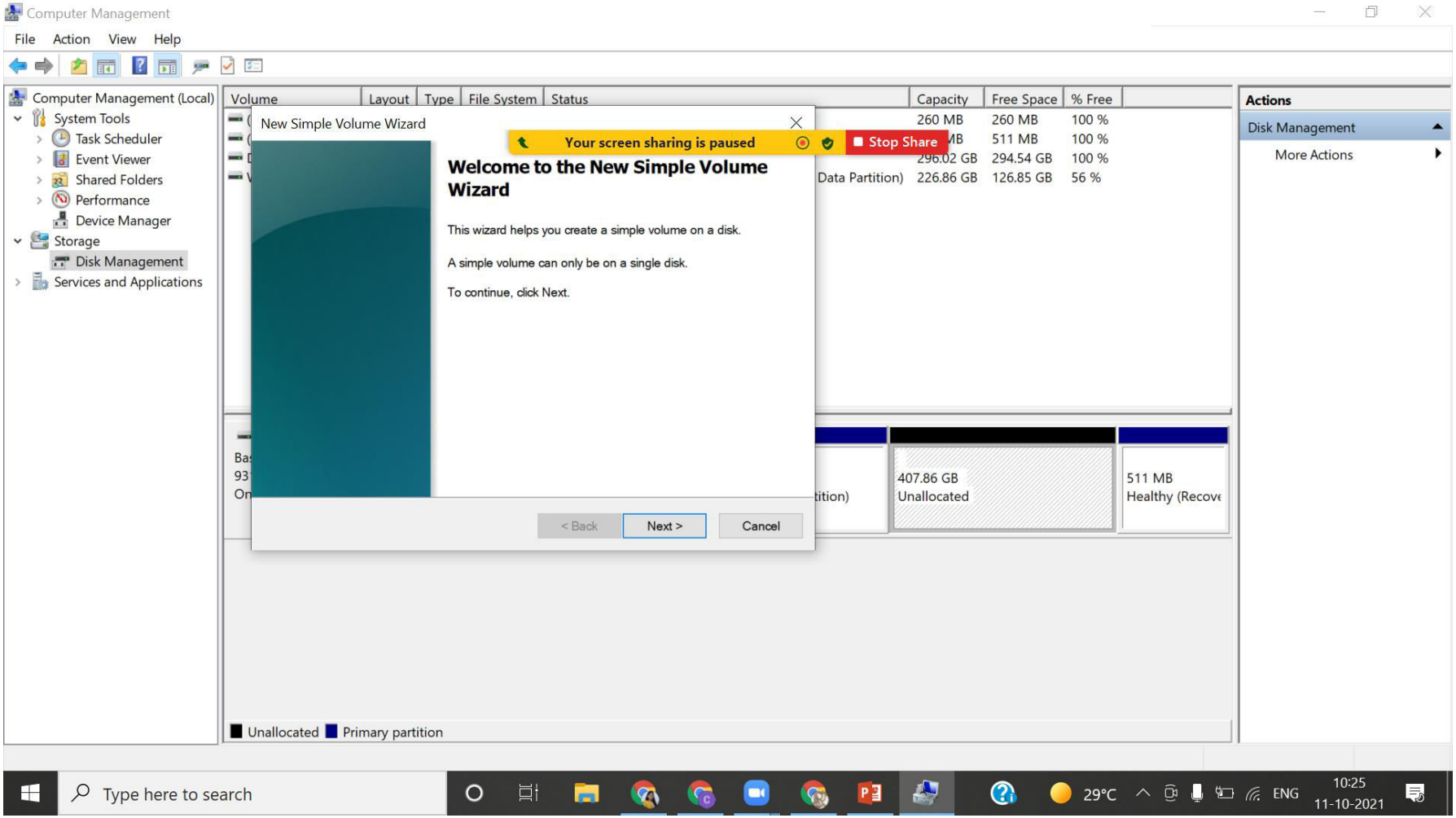




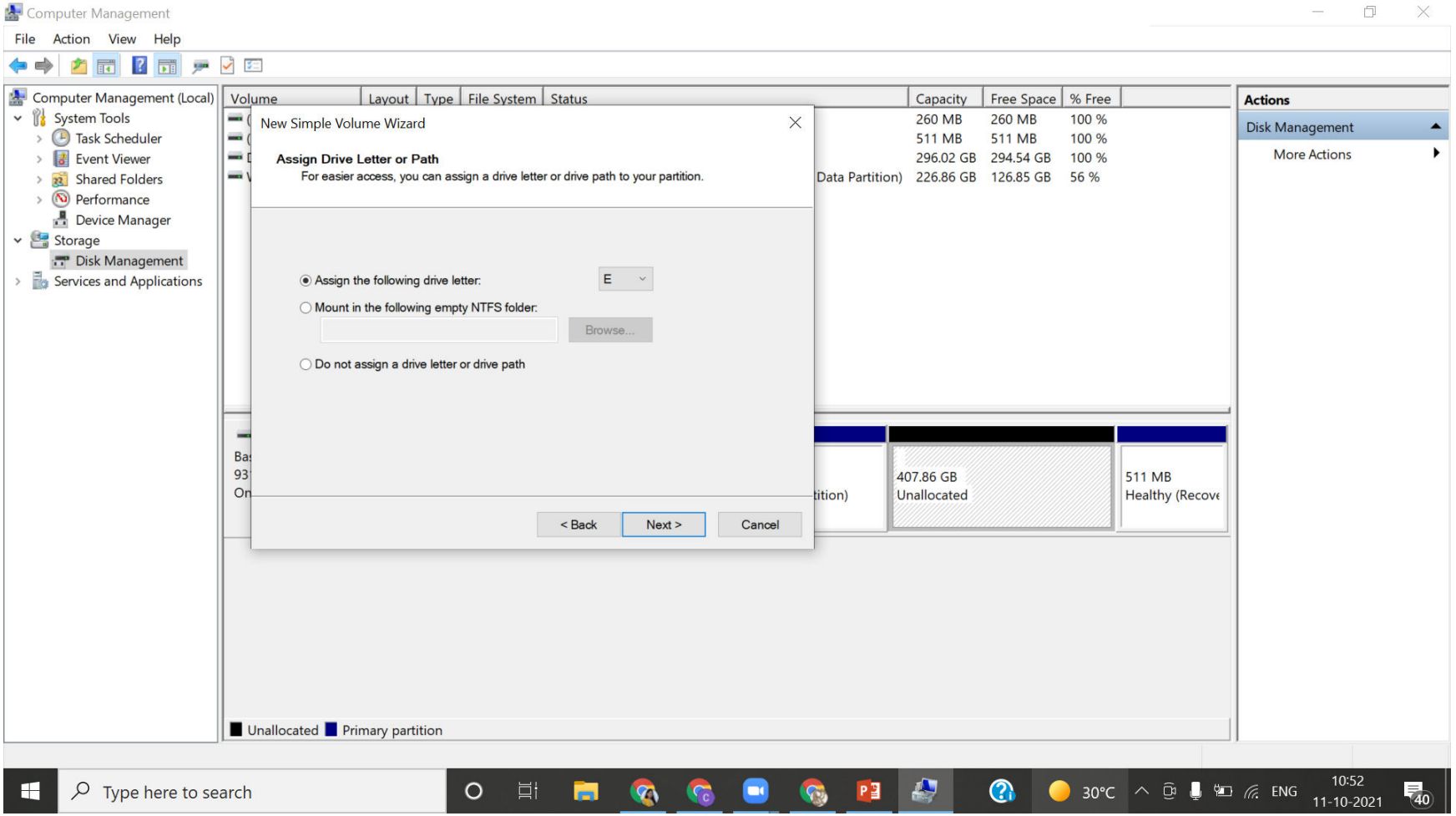
Disk Structure



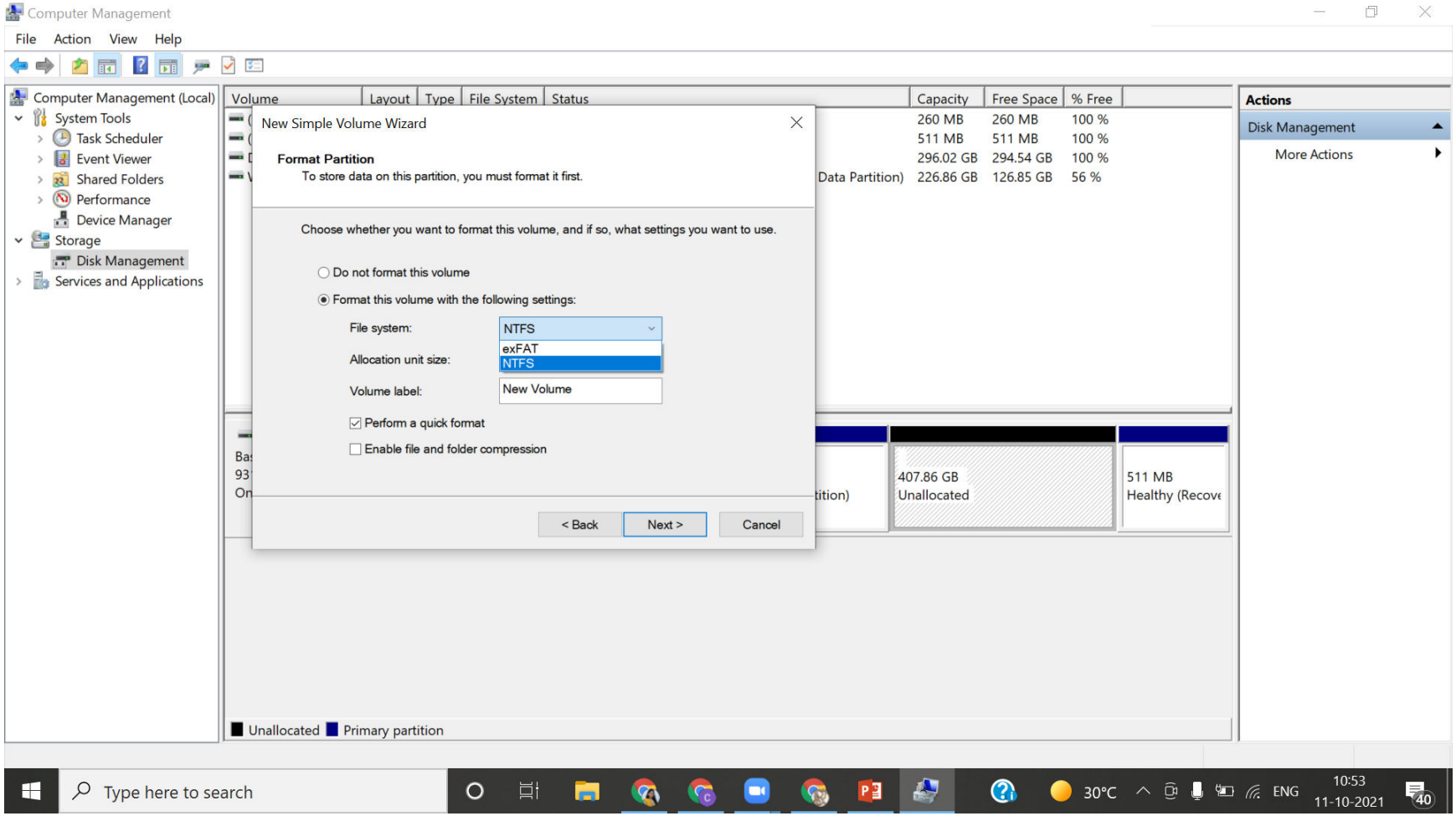
Disk Structure



Disk Structure



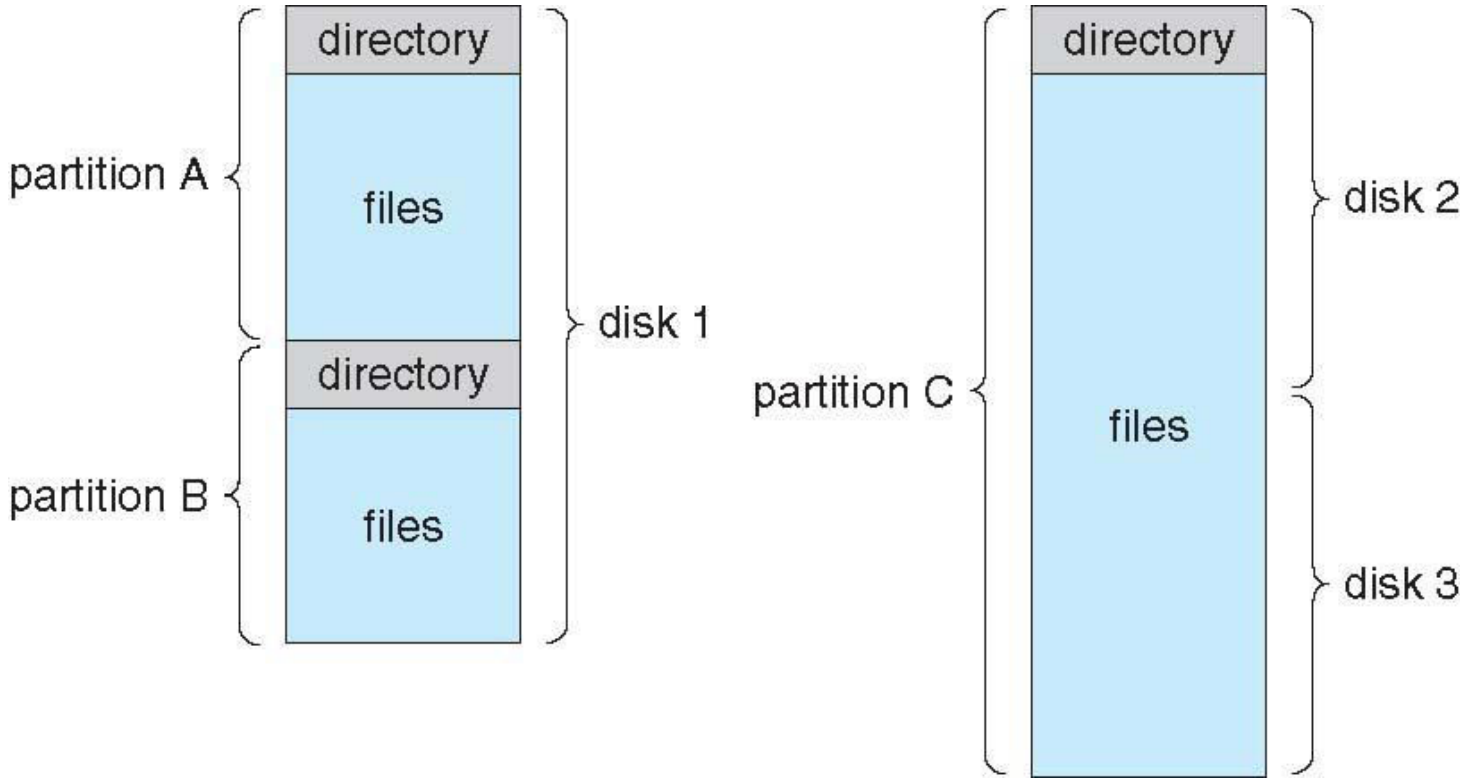
Disk Structure



Directory and Disk Structure

- *A file system can be created on each of these **parts** of the disk.*
- *Any entity containing a file system is generally known as a **volume**.*
- *Each volume can be thought of as a **virtual disk**.*
- *Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.*
- *Each volume that contains a file system must also contain information about the files in the system in a **device directory** or **volume table of contents.***
- *The **device directory** (more commonly known simply as the directory) records information—such as name, location, size, and type—for all files on that volume.*

A Typical File-system Organization



Typical File-system ON Windows

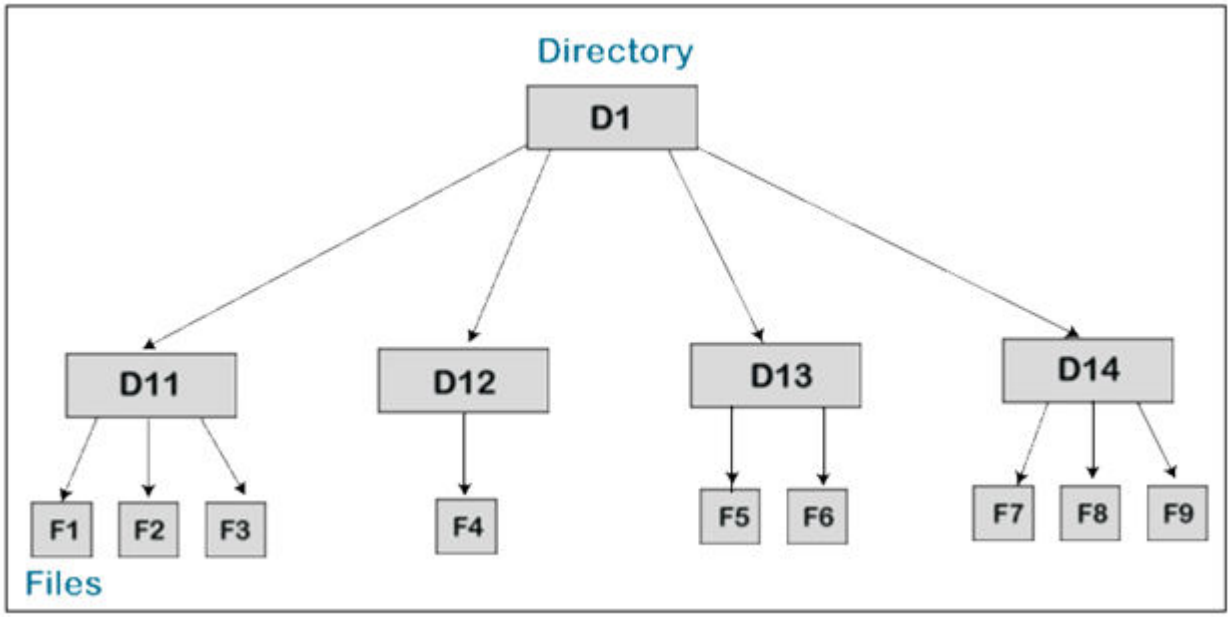
- *File Allocation Table* ***(FAT and FAT32)***
- *Microsoft New Technology File System* ***(NTFS)***
- *Microsoft Resilient File System (ReFS).* ***ReFS*** *was introduced on Windows Server 2012 systems.*

Types of File Systems

- *We mostly talk of **general-purpose** file systems, But systems frequently have many file systems, some general- and some **special- purpose***
- *Solaris has following file systems*
  - ***tmpfs –** Memory-based **volatile FS** for fast, temporary I/O.*
  - ***objfs –** Interface into kernel memory to get kernel symbols for debugging.*
  - ***ctfs** – Contract file system for managing **daemons.***
  - ***lofs –** Loopback file system allows one FS to be accessed in place of another.*
  - ***procfs –** Kernel interface to process structures.*
  - ***ufs, zfs –** General purpose file systems.*

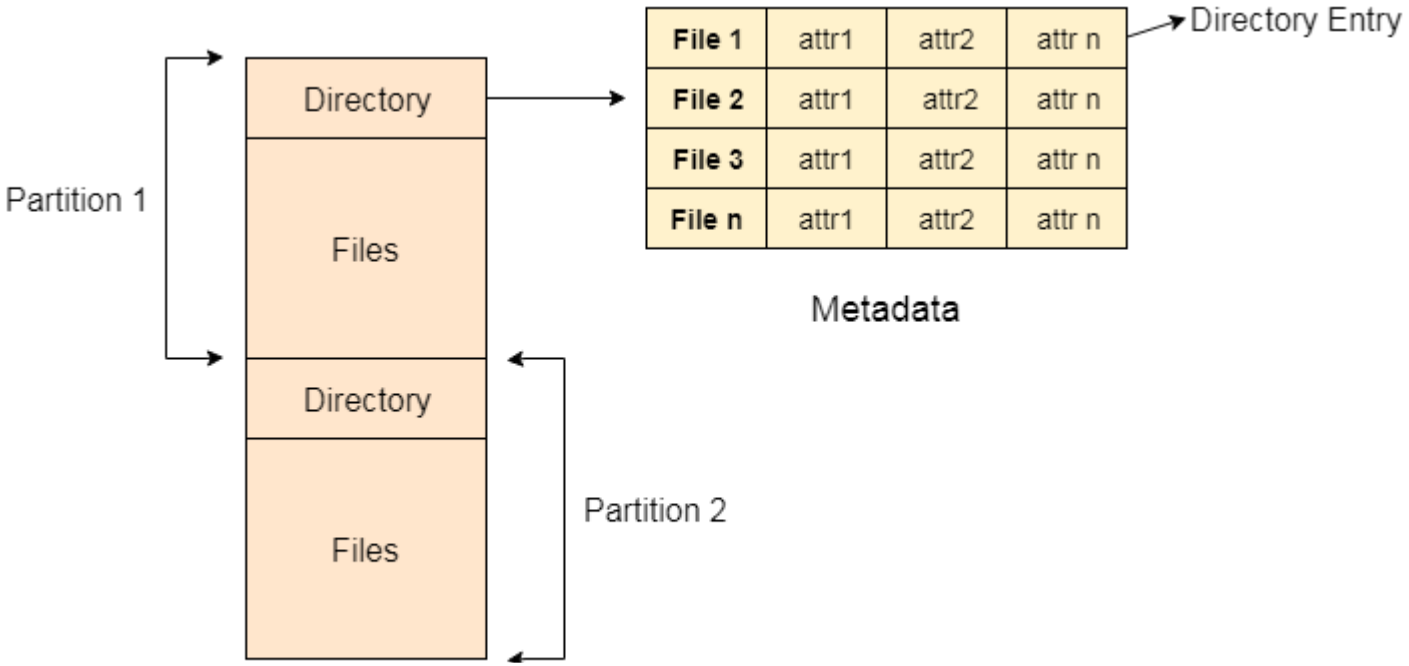
Directory Structure

- *A directory is a container that is used to contain **folders and files**. It organizes files and folders in a hierarchical manner.*



Directory Structure

- *In a directory structure, we can store the complete file attributes and with the help of the directory, we can maintain the information related to the files.*



Operations Performed on Directory

The operations that are to be performed on a directory

- *Search for a file*
- *Create a file*
- *Delete a file*
- *List a directory*
- *Rename a file*
- *Traverse the file system*

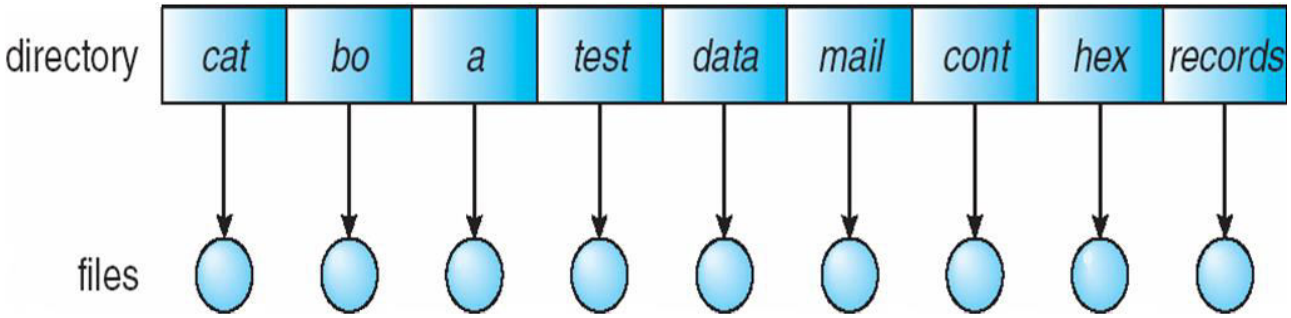
Directory Organization

The directory is organized logically to obtain.

- ***Efficiency*** *– Locating a file quickly.*
- ***Naming*** *– Convenient to users.*
  - *Two users can have same name for different files.*
  - *The same file can have several different names.*
- ***Grouping*** *– Logical grouping of files by properties, (e.g. all Java programs, all games, ...).*

Single-Level Directory

- *This is the **simplest directory** structure to implement. All files are contained in the same directory.*
- *Limitation of single level directory structure is that when number of files increase or when the system has more than one user, it becomes **unmanageable**.*
- *Since all files are in this same directory, the **name must be unique**.*
- *Can't create subdirectories, Ex: the directory called **data** can have no of files but **can not create more subdirectories.***



Single-Level Directory

Advantages:

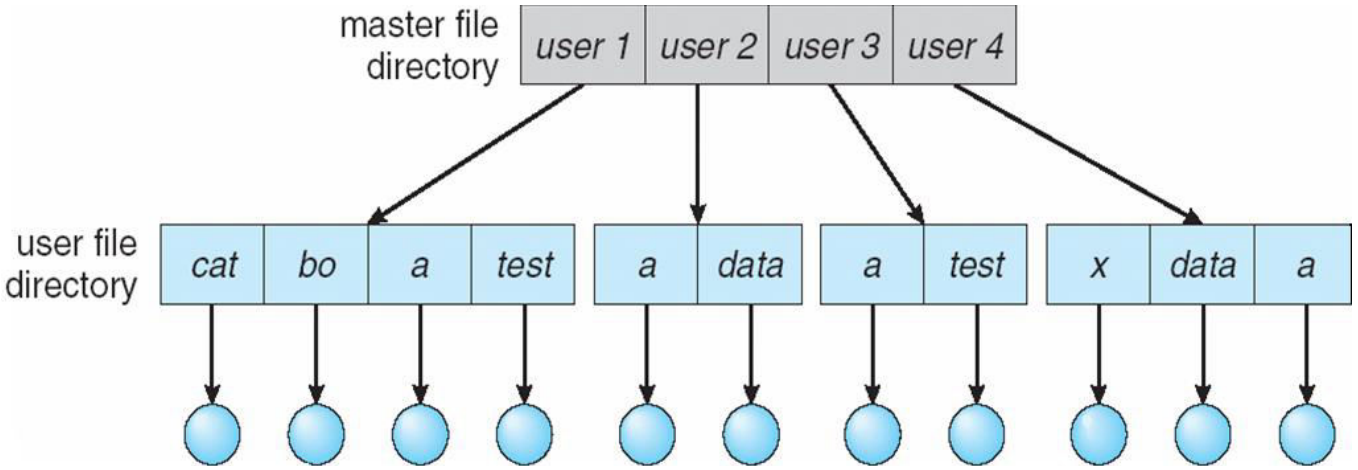
- *Since it is a **single directory**, so it is easy to implement, understand and maintain.*
- *If the files are smaller in size, **searching will become faster**.*
- *The operations like file **creation, searching, deletion, updating** are very easy in such a directory structure.*

Disadvantages:

- *There may chance of name **collision** because two files can not have the same name.*
- *Searching will become time taking if the directory is **large** (High Overhead).*
- *This can not group the same type of files together **(Grouping Problem).***
- *Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name **(Scalability).***

Two-Level Directory

- *In two level directory systems, There is one master directory which contains a **separate directory** for each user.*
- *The system doesn't let a user to enter in the other user's directory without permission.*



Two-Level Directory

Advantages:

- *We can give full path like **/User-name/directory-name/.***
- *Different users can have the same directory as well as the file name.*
- *Searching of files becomes easier due to pathname and user-grouping.*

Disadvantages:

- *A user is not allowed to share files with other users.*
- *Still, it not very **scalable**, two files of the same type cannot be grouped together in the same user.*

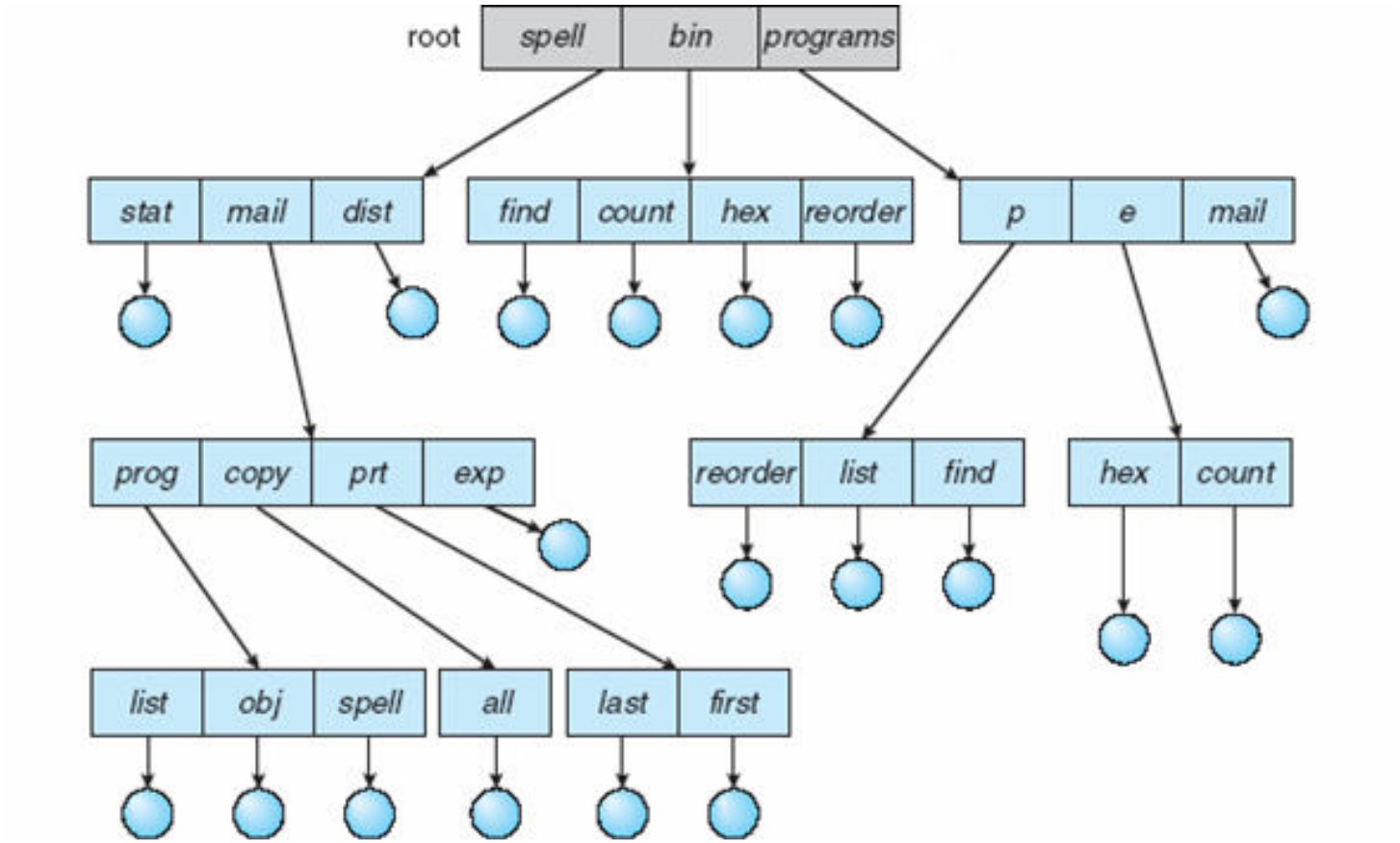
Tree-Structured Directories

- *A two-level directory is a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.*
- *This generalization allows the user to create their* ***own subdirectories*** *and to organize their files accordingly.*
- *In Tree structured directory system, any directory entry can either be a* ***file or sub directory.***
- *The similar kind of files can now be* ***grouped*** *in one directory.*

Tree-Structured Directories

- *Most common directory structure.*
- *This structure has root directory and every file in system has a* ***unique path name.***
- *This structure provide* ***efficient searching.***
- ***Naming*** *problem is totally solved.*

Tree-Structured Directories



Tree-Structured Directories (Cont.)

Advantages:

- *Very **scalable**, the probability of name collision is less.*
- *Searching becomes very easy, we can use both absolute paths as well as relative path.*

Disadvantages:

- *Every file does not fit into the hierarchical model, files may be saved into multiple directories.*
- *We can not share files.*
- *Tree-structure directory is not efficient because, in this, if we want to access a file, then it may go under multiple directories.*

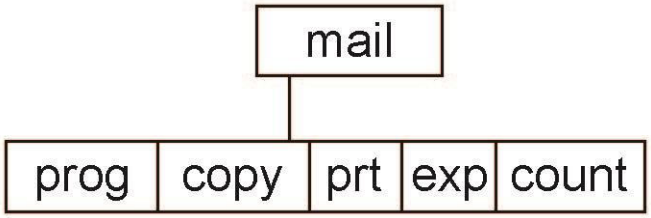
Tree-Structured Directories (Cont)

- ***Absolute*** *or* ***relative*** *path name*
- *Creating a new file is done in current directory*
- *Delete a file* ***rm <file-name>***
- *Creating a new subdirectory is done in current directory*

mkdir <dir-name>

Example: if in current directory ***/mail***

mkdir count



Deleting ***"mail"*** $\Rightarrow$ *deleting the entire subtree rooted by "mail"*

Acyclic-Graph Directories (Cont.)

- *Two different names (aliasing)*
- *If **dict** deletes **list** ⇒ dangling pointer*
- *Solutions:*
  - *Backpointers, so we can delete all pointers*

    Variable size records a problem
  - *Backpointers using a daisy chain organization*
  - *Entry-hold-count solution*
- *New directory entry type*
  - ***Link** – another name (pointer) to an existing file*
  - ***Resolve the link** – follow pointer to locate the file*

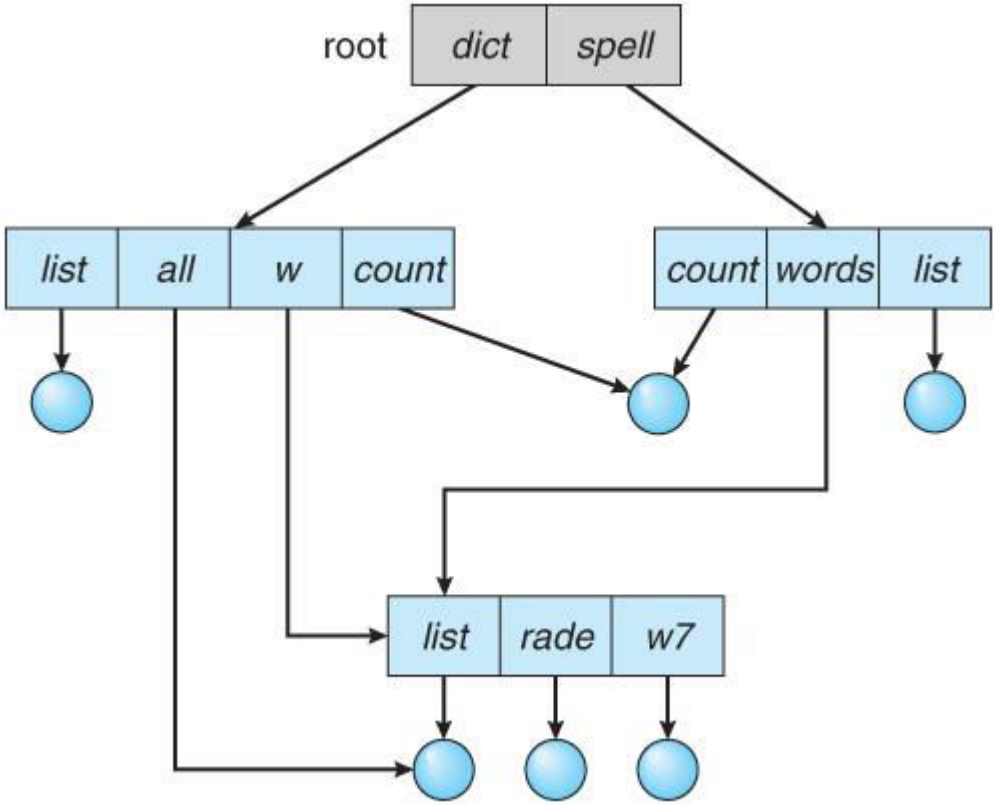
Acyclic-Graph Directories (Cont.)

- *In the tree-structure directory, the same files cannot exist in the multiple directories, so sharing the files is the **main problem** in the tree-structure directory.*
- *With the help of the acyclic-graph directory, we can provide the sharing of files.*
- *In the acyclic-graph directory, more than one directory can point to a similar file or subdirectory.*
- *An acyclic graph is a graph with no cycle and **allows us to share subdirectories and files***

Acyclic-Graph Directories (Cont.)

- *The **same file or subdirectories** may be in two different directories. It is a natural generalization of the tree-structured directory.*
- *It is used in the situation like when two programmers are working on a joint project and they need to access files.*
- *These kinds of directory graphs can be made using links or aliases. We can have multiple paths for a same file.*
- *Links can either be symbolic (logical) or hard link (physical).*

Acyclic-Graph Directories (Cont.)



Acyclic-Graph Directories (Cont.)

Advantages:

- *We can share files.*
- *Searching is easy due to different-different paths.*

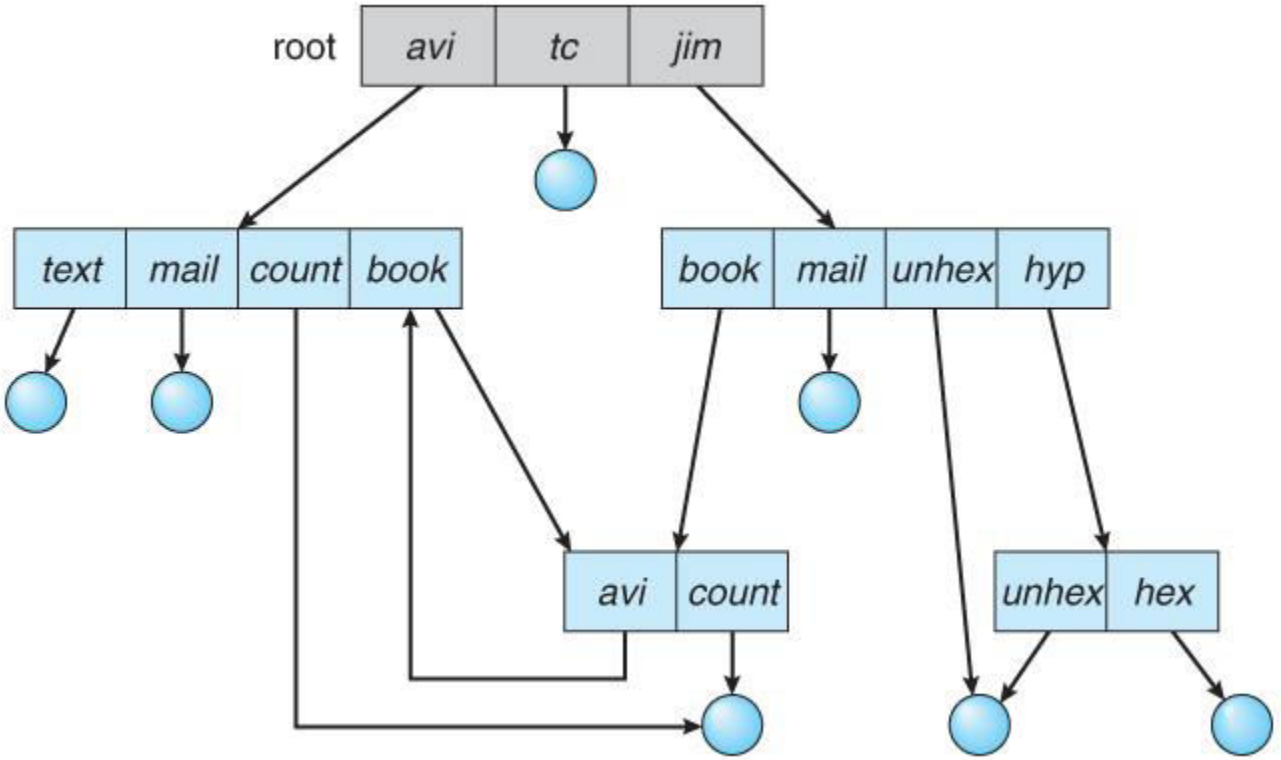
Disadvantages:

- *We share the files via linking, in case deleting it may create the problem,*
- *If the link is a **soft link** then after deleting the file we left with a dangling pointer.*
- *In the case of a **hard link**, to delete a file we have to delete all the references associated with it.*

General Graph Directory (Cont.)

- *The General-Graph directory is another vital type of directory structure.*
- *In this type of directory structure, within a directory we can create* ***cycle of the directory*** *where we can derive the various directory with the help of more than one parent directory.*
- *The main issue in the general-graph directory is to calculate the total space or size, taken by the directories and the files*

General Graph Directory (Cont.)



General Graph Directory (Cont.)

Advantages

The advantages of general-graph directory are:

1. *The General-Graph directory is more flexible than the other directory structure.*
2. *Cycles are allowed in the general-graph directory.*

Disadvantages

The disadvantages of general-graph directory are:

1.In general-graph directory, garbage collection is required.

2.General-graph directory is more costly, among other directory structures.

3.Search algorithms can go into infinite loops.

General Graph Directory (Cont.)

- *How do we guarantee no cycles?*
  - *Allow only links to file not subdirectories*
  - ***Garbage collection***
  - *Every time a new link is added use a cycle detection algorithm to determine whether it is OK*

File Sharing

- *Sharing of files on a multi-user systems is desirable*
- *Sharing may be done through a **protection** scheme*
- *On distributed systems, files may be shared across a network, Network File System (NFS) is a common distributed file-sharing method*
- *In a multi-user system*
  - ***User IDs** identify users, allowing permissions and protections to be per-user*
  - ***Group IDs** allow users to be in groups, permitting group access rights*
  - ***Owner of a file /directory:** grant access, has the most control over the file*
- *Group of a file/directory: subset of users who can share access to the file.*

File Sharing – Remote File Systems

- *Uses networking to allow communication among remote computers.*
- *Networking allows the sharing of resources spread across a campus or even around the world.*
- *One obvious resource to share is data in the form of files.*
- *The first implemented method is manually sending via programs like **FTP***
- *The second major method uses a **distributed file system (DFS)** in which remote directories are visible from a local machine.*
- *The third method is the **WorldWide Web**, is a reversion to the first. A browser is needed to gain access to the remote files, and separate operations are used to transfer files. Increasingly, cloud computing is being used for file sharing as well.*

File Sharing – Client Server Model

- *The machine containing the files is the **server**, and the machine seeking access to the files is the **client**.*
- *The server declares that a resource is available to clients and specifies exactly which resource and exactly which clients.*
- *Server can serve multiple clients, client can use multiple servers.*
- *Client identification is more difficult. A client can be specified by a network name or other identifier, such as an IP address, but these can be spoofed, or imitated.*
- *File operation requests are sent on behalf of the user with id) across the network to the server via the DFS protocol.*
- *The server then applies the standard access checks to determine access rights. The request is either allowed or denied. If it is allowed, a file handle is returned to the client application, and the application then can perform read, write, and other operations on the file.*

File Sharing – Distributed Information Systems

- *To make client–server systems easier to manage, distributed information systems, also known as distributed naming services, provide unified access to the information needed for remote computing.*
- *The domain name system (DNS) provides host-name-to-network-address translations for the entire Internet.*
- *Before DNS became widespread, files containing the same information were sent via e-mail or ftp between all networked hosts.*

File Sharing – Distributed Information Systems

- *The machine containing the files is the **server**, and the machine seeking access to the files is the **client**.*
- *The server declares that a resource is available to clients and specifies exactly which resource and exactly which clients.*
- *Server can serve multiple clients, client can use multiple servers.*
  - *Client and user-on-client identification is insecure or complicated*
  - ***NFS** is standard UNIX client-server file sharing protocol*
  - ***CIFS** is standard Windows protocol*
  - *Standard operating system file calls are translated into remote calls*
- *Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing*

File Sharing – Failure Modes

- *All file systems have failure modes*
  - *For example corruption of directory structures or other non-user data, called* ***metadata***
- *Remote file systems add new failure modes, due to network failure, server failure*
- *Recovery from failure can involve* ***state information*** *about status of each remote request*
- ***Stateless*** *protocols such as NFS v3 include all information in each request, allowing easy recovery but less security*

File Sharing – Consistency Semantics

- *Consistency Semantics deals with the consistency between the **views of shared files** on a networked system. When one user changes the file, when do other users **see the changes**?*

UNIX Semantics

- *The UNIX file system uses the following semantics:*
  - *Writes to an open file are immediately visible to any other user who has the file open.*
  - *One implementation uses a shared location pointer, which is adjusted for all sharing users.*
- *The file is associated with a single exclusive physical resource, which may delay some accesses.*

File Sharing – Consistency Semantics

Session Semantics

- *The Andrew File System, AFS uses the following semantics:*
  - *Writes to an open file are not immediately visible to other users.*
  - *When a file is closed, any changes made become available only to users who open the file at a later time.*
- *According to these semantics, a file can be associated with multiple (possibly different) views. Almost no constraints are imposed on scheduling accesses. No user is delayed in reading or writing their personal copy of the file.*
- *AFS file systems may be accessible by systems around the world. Access control is maintained through (somewhat) complicated access control lists, which may grant access to the entire world (literally) or to specifically named users accessing the files from specifically named remote environments.*

File Sharing – Consistency Semantics

Immutable-Shared-Files Semantics

- *Under this system, when a file is declared as **shared** by its creator, it becomes immutable. Hence it becomes read-only, and shared access is simple.*

Protection

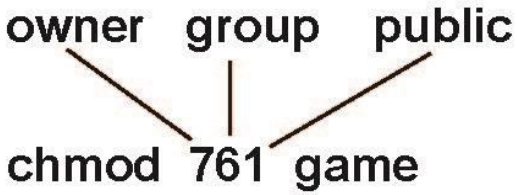
- *File owner/creator should be able to control:*
  - *what can be done*
  - *by whom*
- *Types of access*
  - ***Read***
  - ***Write***
  - ***Execute***
  - ***Append***
  - ***Delete***
  - ***List***

Access Lists and Groups

- *Mode of access: read, write, execute*
- *Three classes of users on Unix / Linux*

			RWX
a) ***owner access***	*7*	⇒	*1 1 1*
b) ***group access***	*6*	⇒	*1 1 0*
c) ***public access***	*1*	⇒	*0 0 1*

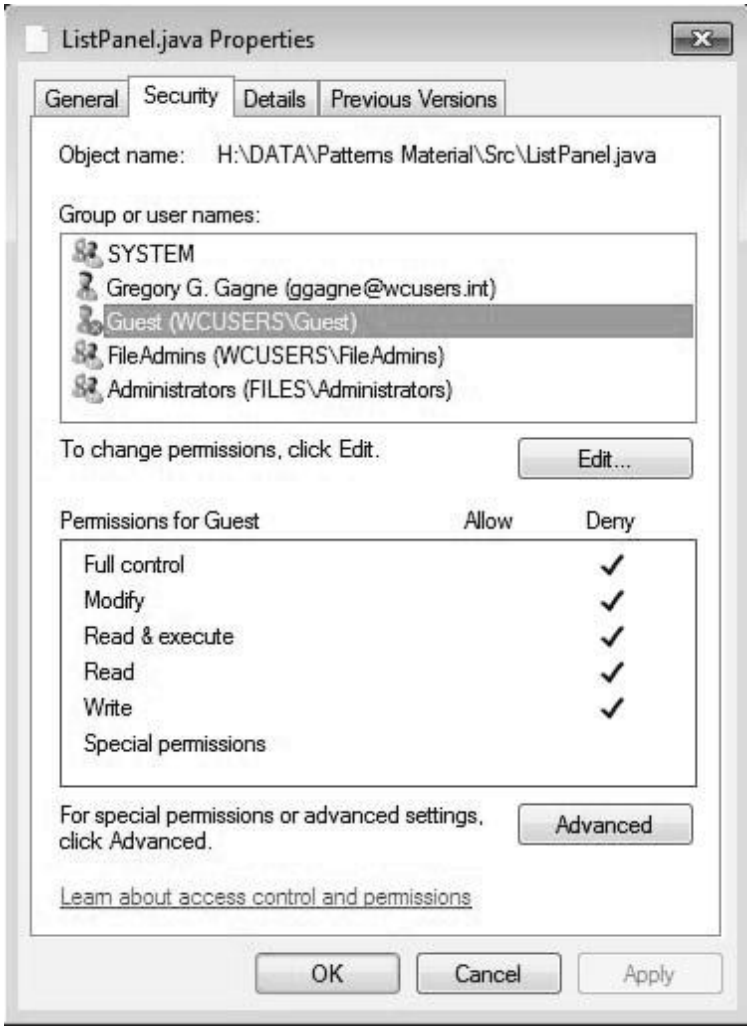
- *Ask manager to create a group (unique name), say G, and add some users to the group.*
- *For a particular file (say game) or subdirectory, define an appropriate access.*



- *Attach a group to a file*

chgrp G game

Windows 7 Access-Control List Management



A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Allocation Methods

Allocation method provides a way in which the disk will be utilized and the files will be accessed.

1 .Contiguous Allocation

2. Non Contiguous Allocation

i. Linked Allocation

ii. Indexed Allocation`

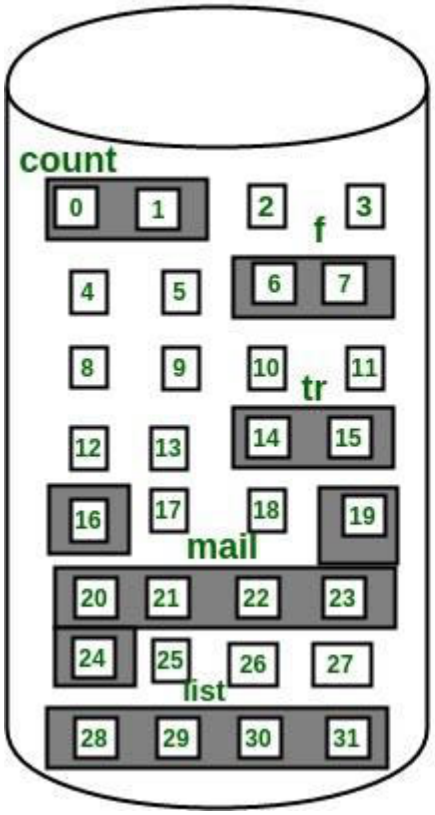
The main idea behind these methods is to provide:

- ***Efficient disk space utilization.***
- ***Fast access to the file blocks.***

Contiguous Allocation

- *If the blocks are allocated to the file in such a way that all the **logical blocks** of the file get the **contiguous physical block** in the hard disk then such allocation scheme is Known as **contiguous allocation**.*
- *In this scheme, **each file occupies a contiguous set of blocks on the disk**. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,…….b+n-1.*
- *This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.*
- *The directory entry for a file with contiguous allocation contains Address of starting block Length of the allocated portion.*

Contiguous Allocation



Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation

Advantages

1.Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).

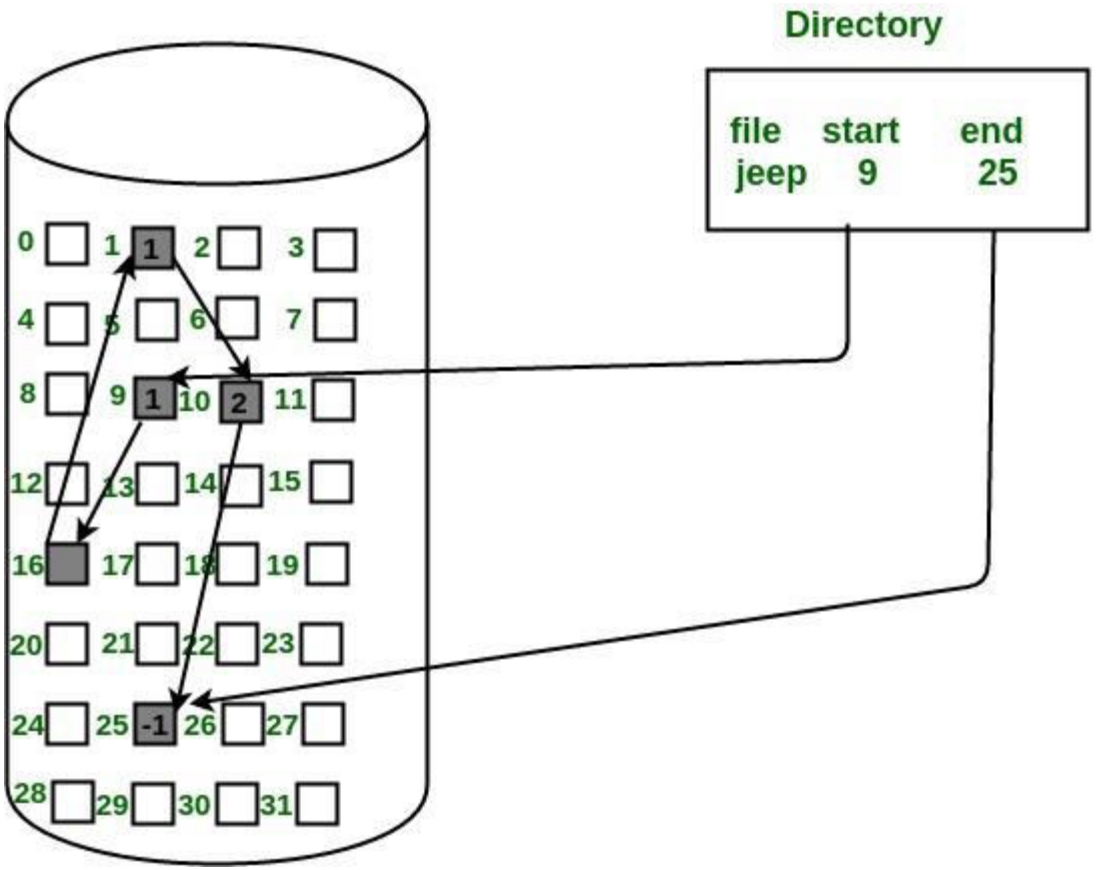
2.It is simple to implement.

3.We will get Excellent read performance.

Disadvantages

1. *This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.*
2. *Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.*

Linked Allocation



Linked Allocation

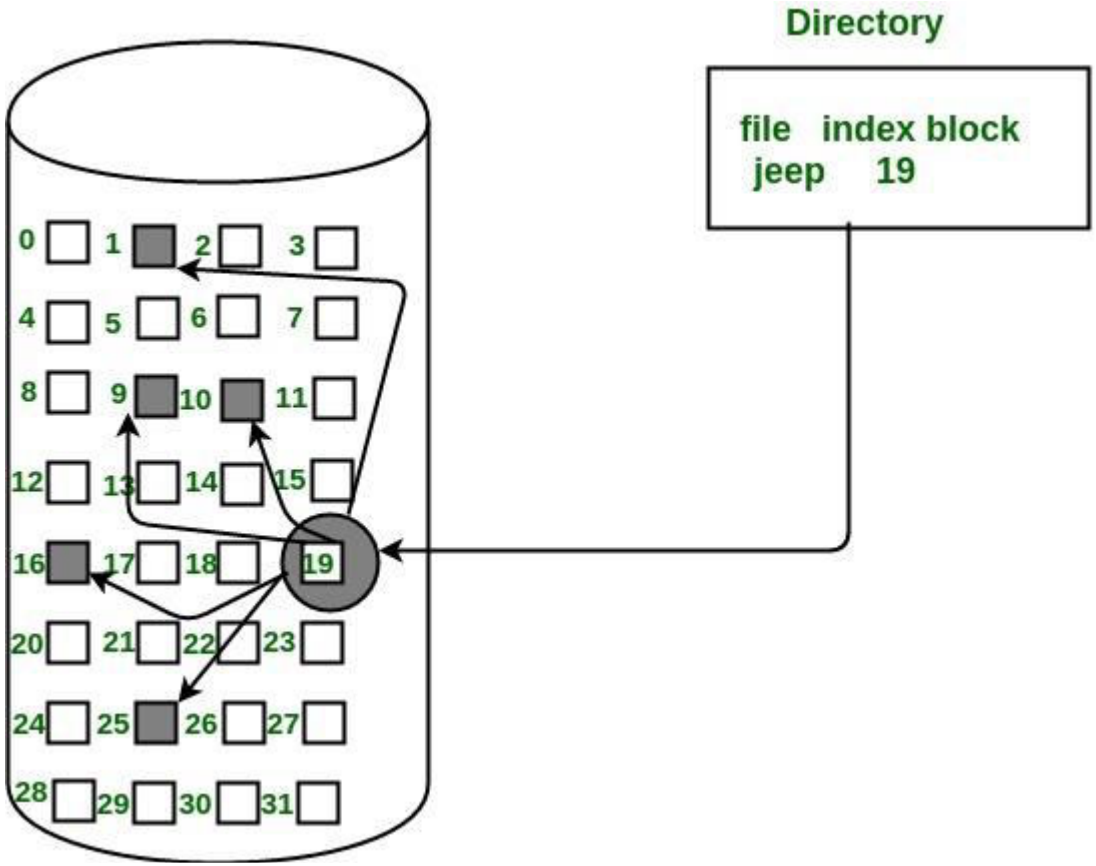
Advantages:

- *This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.*
- *This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.*

Disadvantages:

- *Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.*
- *It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.*
- *Pointers required in the linked allocation incur some extra overhead.*

Indexed Allocation



Indexed Allocation

Advantages:

- *This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.*
- *It overcomes the problem of external fragmentation.*

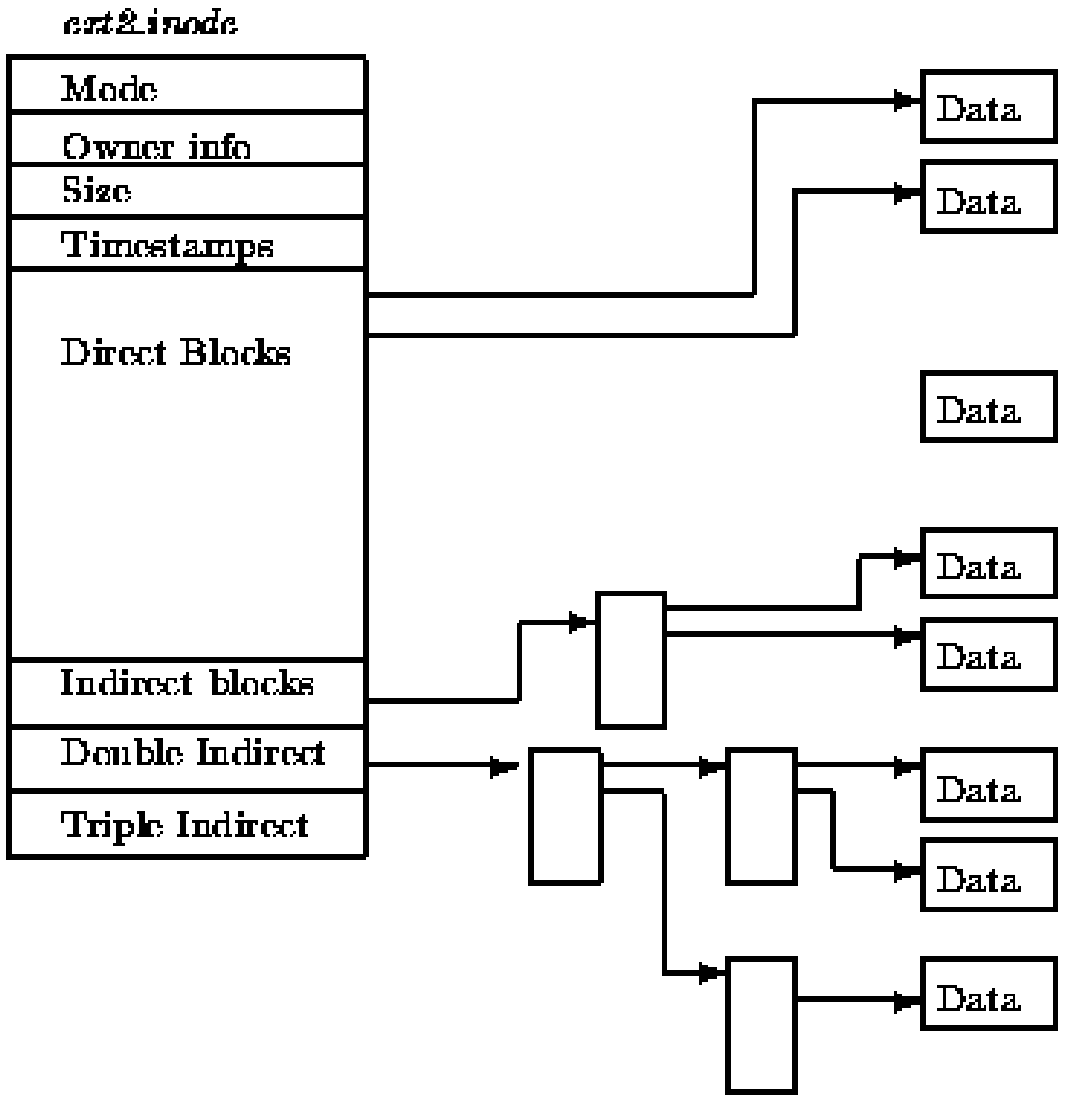
Disadvantages:

- *The pointer overhead for indexed allocation is greater than linked allocation.*

Inode (Index Node)

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.



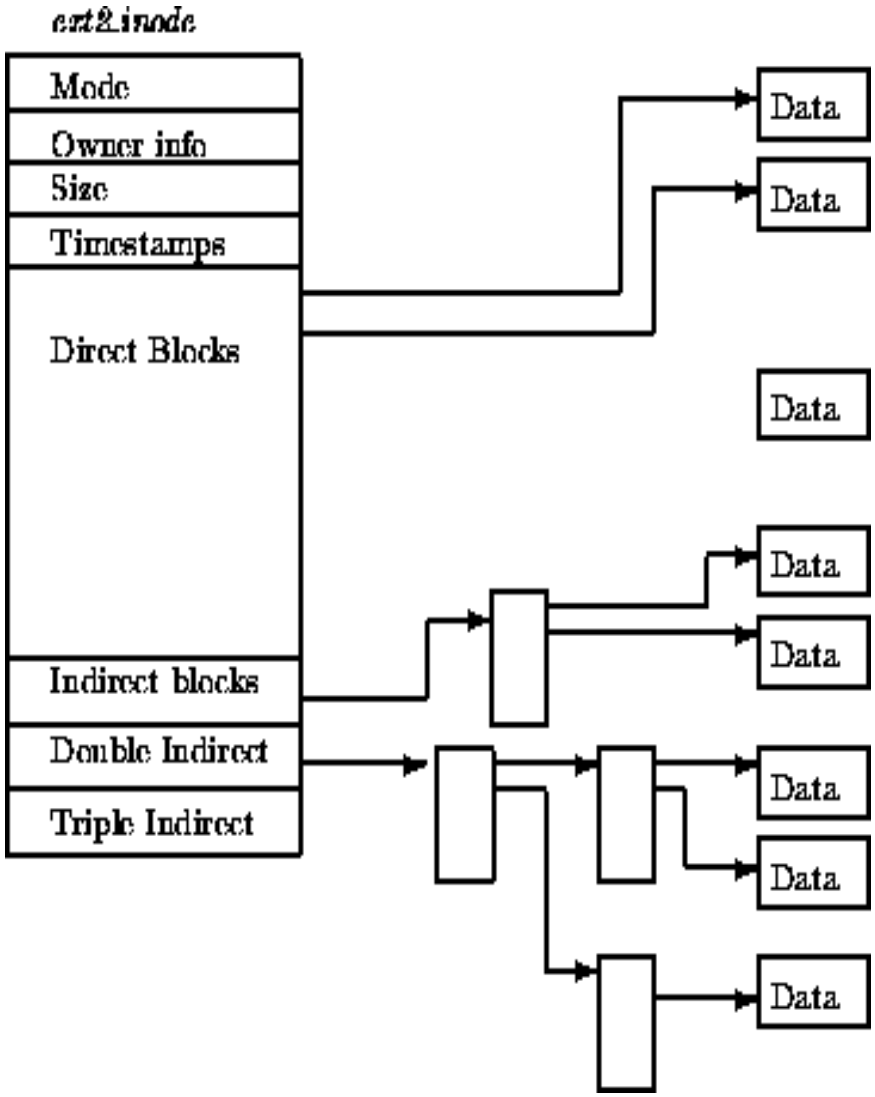
Inode (Index Node)

Key Components of an Inode

1. **File Metadata**:
    1. File type (regular file, directory, symbolic link, etc.)
    2. Permissions (read, write, execute)
    3. Owner (user ID and group ID)
    4. Size
    5. Timestamps (creation, last access, modification)
2. **Block Pointers**:
    1. Direct pointers: Point directly to the data blocks.
    2. Indirect pointers:
        1. Single indirect: Points to a block containing more block pointers.
        2. Double indirect: Points to a block containing single indirect block pointers.
        3. Triple indirect: Points to a block containing double indirect block pointers.

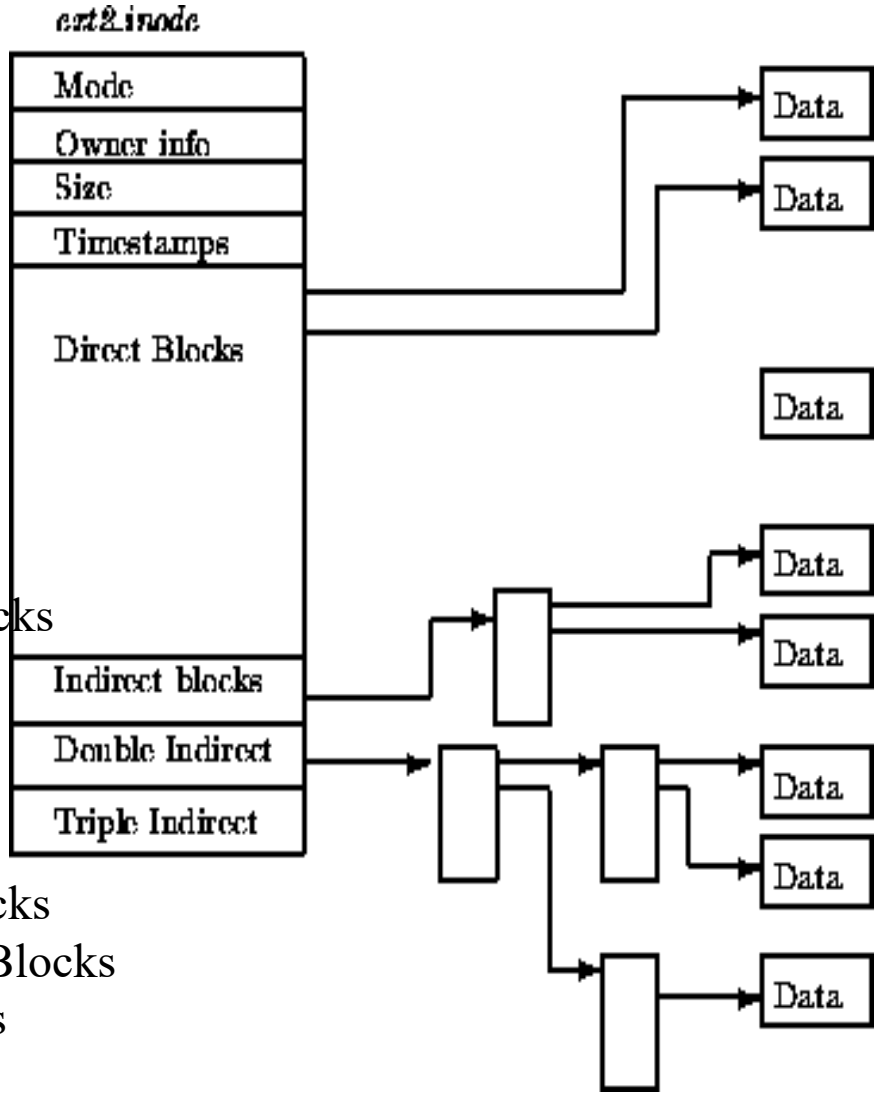
Inode (Index Node)

An indirect block is a block that contains an array of pointers. Instead of pointing directly to data blocks, these pointers in the indirect block point to other blocks that, in turn, contain pointers to data blocks.



Inode (Index Node)

- Inode
  - Direct Pointers (e.g., 0-11)
    - Data Blocks
  - Single Indirect Pointer
    - Block of Pointers to Data Blocks
      - Data Blocks
  - Double Indirect Pointer
    - Block of Pointers to Single Indirect Blocks
      - Block of Pointers to Data Blocks
        - Data Blocks
  - Triple Indirect Pointer
    - Block of Pointers to Double Indirect Blocks
      - Block of Pointers to Single Indirect Blocks
        - Block of Pointers to Data Blocks
          - Data Blocks



Inode (Index Node)

The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointer The disk block size is 4 kB and the disk block addresses 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____ GB.

Inode (Index Node)

Assumptions:

1. **Block Size (B)**: Usually 4 KB (4096 bytes).
2. **Pointer Size (P)**: Typically 4 bytes (32-bit pointer).

This gives the number of pointers per block as:

$$\text{Pointers per block} = \frac{\text{Block Size}}{\text{Pointer Size}} = \frac{4096}{4} = 1024 \text{ pointers/block}.$$

1. **Direct Pointers**:
   - There are 12 direct pointers. Each points directly to a block.
   - Maximum data addressed by direct pointers:
     $$\text{Direct Data} = 12 \times \text{Block Size} = 12 \times 4096 = 48 \text{ KB}.$$
2. **Single Indirect Pointer**:
   - Points to a block containing 1024 pointers.
   - Each pointer in that block points to a data block.
   - Maximum data addressed by the single indirect pointer:
     $$\text{Single Indirect Data} = 1024 \times \text{Block Size} = 1024 \times 4096 = 4 \text{ MB}.$$

Inode (Index Node)

3. **Double Indirect Pointer**:
   - Points to a block containing 1024 pointers.
   - Each pointer in that block points to another block containing 1024 pointers.
   - Each of those pointers finally points to a data block.
   - Maximum data addressed by the double indirect pointer:
     $$\text{Double Indirect Data} = 1024 \times 1024 \times \text{Block Size} = 1024^2 \times 4096 = 4\ \text{GB}.$$
4. **Triple Indirect Pointer**:
   - Points to a block containing 1024 pointers.
   - Each pointer in that block points to another block containing 1024 pointers.
   - Each of those pointers points to another block containing 1024 pointers.
   - Each of those final pointers points to a data block.
   - Maximum data addressed by the triple indirect pointer:
     $$\text{Triple Indirect Data} = 1024 \times 1024 \times 1024 \times \text{Block Size} = 1024^3 \times 4096 = 4\ \text{TB}.$$

End of Chapter 11