



OPERATING SYSTEMS

Storage Management - 4,5,6

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 4



Unit 4 : Storage Management

Mass-Storage Structure - Mass-Storage overview, Disk Scheduling, Swap-Space Management, RAID structure. File System Interface - file organization/structure and access methods, directories, sharing. File System Implementation/Internals: File control Block (inode), partitions & mounting, Allocation methods. Case Study: Linux/Windows File Systems

- **Swap-Space Management, RAID structure**
- **File Concept, Access Methods**
- **Directory and Disk Structure**
- **File-System Mounting, File Sharing, Protection**
- **File-System Structure, File-System Implementation, Directory Implementation**
- **Allocation methods**
- **Case Study: Linux/Windows File Systems**

- Swap-Space Management
- RAID Structure and Levels

Swap-Space Management

- Swapping is that setting which occurs when the amount of **physical memory** reaches a critically low point and processes are moved from memory to swap space to free available memory.
- In practice, very few modern operating systems implement swapping in this fashion. Rather, systems now combine swapping with virtual memory techniques and swap pages, not necessarily entire processes.
- In fact, some systems now use the terms “swapping” and “paging” interchangeably, reflecting the merging of these two concepts.
- Swap-space management is another low-level task of the operating system. Virtual memory **uses disk space as an extension of main memory**. Since **disk access** is much slower than **memory access**, using swap space significantly decreases system performance.
- The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system. In this section, we discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

Swap-Space Management



- Swap-space — Virtual memory uses disk space as an extension of main memory
- Less common now due to increase in memory capacity
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
 - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
 - Kernel uses swap maps to track swap-space use

Swap-Space Use

- Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use.
- For instance, systems that implement swapping may use swap space to hold an entire process image, including the code and data segments
- Paging systems may simply store pages that have been pushed out of main memory.
- The amount of swap space needed on a system can therefore vary from a few megabytes of disk space to gigabytes, depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used.

- Note that it may be safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort processes or may crash entirely.
- Overestimation wastes disk space that could otherwise be used for files, but it does no other harm.

Swap-Space Location

- A swap space can reside in one of two places: it can be carved out of the normal file system, or it can be in a separate disk partition.
- If the swap space is simply a large file within the file system, normal file-system routines can be used to create it, name it, and allocate its space. This approach, though easy to implement, is inefficient.
- Navigating the directory structure and the disk allocation data structures takes time and (possibly) extra disk accesses.
- External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image.
- We can improve performance by caching the block location information in physical memory and by using special tools to allocate physically contiguous blocks for the swap file, but the cost of traversing the file-system data structures remains

Swap-Space Location

- Alternatively, swap space can be created in a separate raw partition.
- No file system or directory structure is placed in this space. Rather, a separate swap-space storage manager is used to allocate and deallocate the blocks from the raw partition.
- This manager uses algorithms optimized for speed rather than for storage efficiency, because swap space is accessed much more frequently than file systems (when it is used).
- Internal fragmentation may increase, but this trade-off is acceptable because the life of data in the swap space generally is much shorter than that of files in the file system. Since swap space is reinitialized at boot time, any fragmentation is short-lived.
- The raw-partition approach creates a fixed amount of swap space during disk partitioning. Adding more swap space requires either re-partitioning the disk (which involves moving the other file-system partitions or destroying them and restoring them from backup) or adding another swap space elsewhere.

Swap-Space Location

- Some operating systems are flexible and can swap both in raw partitions and in file-system space.
- Linux is an example: the policy and implementation are separate, allowing the machine's administrator to decide which type of swapping to use.
- The trade-off is between the convenience of allocation and management in the file system and the performance of swapping in raw partitions

Swap-Space Management



- Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
 - File data written to swap space until write to file system requested
 - Other dirty pages go to swap space due to no other home
 - Text segment pages thrown out and reread from the file system as needed
- What if a system runs out of swap space ?
- Some systems allow multiple swap spaces

Redundant Array of Inexpensive Disks (RAID) - About



- It can also be called appropriately as Redundant Array of Independent Disks
- RAID – Redundant Array of Inexpensive Disks
 - Multiple Disk Drives provides Reliability via Redundancy
- Increases the mean time to failure
- Mean time to repair – exposure time when another failure could cause data loss
- Mean time to data loss based on above factors

Redundant Array of Inexpensive Disks (RAID) - About



To create an optimal cost-effective RAID configuration, we need to simultaneously achieve the following goals:

- Maximize the number of disks being accessed in parallel.
- Minimize the amount of disk space being used for redundant data.
- Minimize the overhead required to achieve the above goals.

Redundant Array of Inexpensive Disks (RAID) - About

- Redundancy – Redundancy refers to having numerous components which offer the same function, so that the system functioning can continue in the event of partial system failure.
- Fault Tolerance – Fault Tolerance means, in the event of a component failure, the system is designed in such a way that a backup component will be available, in order to prevent loss of service.
- The distribution of data is done in two ways, which are two generalized concepts.
- **Mirroring** is one of the ways, which offers replication of data on another disk and the second concept is **striping**, where splitting of reproduced data takes place across the available disk drives.
- **Parity** is a concept, which is also streamlined into RAID technology as another way of storage method. It involves saving of information across the disk arrays, so that, the same information can be used to recreate or reconstruct the affected data, which is otherwise filled with errors or data loss, when disk drive fails.

Redundant Array of Inexpensive Disks (RAID) - About



- Redundant Array of Independent/Inexpensive Disks (RAID) is a technology that allows storing data across multiple hard drives.
- The purpose of RAID is to achieve data redundancy to reduce data loss and, in a lot of cases, improve performance. The best way to get in on the RAID action is with a Network Attached Storage - NAS
- RAID was created in 1988 in order to deal with costs of high-performing disk drives.
- The inventors argued that an array of inexpensive disks could outperform a single expensive disk, which was no doubt a huge problem when 10MB of storage cost upwards of \$100.
- RAID allows for data to be stored on multiple disk drives at once, though a RAID setup will appear to, say, Windows 10, as a single disk.

Redundant Array of Inexpensive Disks (RAID) - About



- A RAID controller is the thing that directs data in and out of storage drives that can be based on either software or hardware.
- In the software case, if you have a few drives set up in a RAID array inside your home PC, it's likely handled on a software level with your PC's processor (CPU) handling the work.
- In the hardware case, say with an external Network Attached Storage (NAS) enclosure, there will be a dedicated controller card to deal with input and output (I/O) operations of the RAID array.

Redundant Array of Inexpensive Disks (RAID) - About



- RAID 0 was the first version created, and it provides users with fast read and write speeds for improved performance.
- Despite its name, there is no data redundancy.
- Data is striped across drives, meaning each drive holds a piece of the overall information.
- Striping means quicker data access, but if one drive fails they will all fail, resulting in a loss of data.
- Speeds are measured by the number of drives in the RAID 0 array, so consider a RAID 0 array with four drives to be four times faster than a single drive.
- Instead of having a one-lane highway for all data to travel, there are now multiple lanes, all shipping data back and forth.

Redundant Array of Inexpensive Disks (RAID) - About



- The performance gains are offset by the lack of data redundancy and the risk of losing all drives in the event of one failing.
- Other than RAID 0, you'll also often hear of RAID 1, RAID 5, RAID 6, and RAID 10 setups.

Redundant Array of Inexpensive Disks (RAID) - About



- A RAID 1 setup consists of at least two drives that are mirrored to contain the exact same information.
- This RAID 1 setup includes fault tolerance, as one drive failing will not result in the other drives failing as well.
- There's no striping involved, so as long as one drive works, the array will continue to operate, making this a favorite of those who require high reliability.
- In most cases, read performance should be about the same as with a single disk, though there can be a detrimental effect to write speed and storage capacity
- When data is written to the array, it must be written to each drive independently.
- Write speeds will therefore only be as fast as the slowest drive in the array.
- Likewise, storage capacity is dependent on the size of the smallest disk, so having a 256GB and a 512GB drive means you'll lose half the storage space of the latter hardware.
- RAID 1 is generally the most expensive choice since its efficiency can basically be measured as the number of drives divided by its own number.

OPERATING SYSTEMS

Redundant Array of Inexpensive Disks (RAID) - About

- If you have three drives, RAID 5 is likely your best choice.
- RAID 5 uses a combination of striping and parity that is distributed across drives. In the event of a drive failure, an Exclusive Or (XOR) logic gate is used to piece together the lost drive using parity information from the other drives.
- This can be accomplished even while the other drives continue their regular function (though at a slower speed), meaning there will be minimal downtime if you lose a drive.



Redundant Array of Inexpensive Disks (RAID) - About

- RAID 6 is similar to RAID 5, though it uses at least four drives due to a dual parity setup.
- This means that in a four-drive setup you lose half of your storage space for parity, though your data will still be intact in the event of simultaneously losing two drives. Like RAID 5, read speeds are excellent due to striping, though write speeds are generally slowed down due to parity.
- Though it requires four drives to function, a RAID 6 setup is likely best suited for arrays with five or more drives.
- RAID 10 (also known as RAID 1+0) is best suited for four drives.
- Instead of relying on either mirroring or striping, it actually involves both, hence the 1+0.
- Data is essentially striped between two sets of mirrored disks, creating a system that can handle a single drive failure in either set of drives or a simultaneous drive failure in both sets. What it cannot deal with is a simultaneous drive failure of one set.

Redundant Array of Inexpensive Disks (RAID) - About

- RAID was originally invented to cut costs of '80s storage prices while simultaneously accounting for high fail-rates of some early technology.
- RAID setups are still primarily used in large servers; think of corporations that require 24/7 access to their critical data.
- Now though, in an age of cheap bulk storage, RAID setups are often used privately in conjunction with specialized tasks and to avoid monthly fees and privacy concerns surrounding cloud storage services.

Redundant Array of Inexpensive Disks (RAID) - Structure



- Frequently combined with NVRAM to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk striping uses a group of disks as one storage unit
- RAID is arranged into six different levels
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
 - Mirroring or shadowing (RAID 1) keeps duplicate of each disk
 - Striped mirrors (RAID 1+0) or mirrored stripes (RAID 0+1) provides high performance and high reliability
 - Block interleaved parity (RAID 4, 5, 6) uses much less redundancy



(a) RAID 0: non-redundant striping.



(b) RAID 1: mirrored disks.



(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



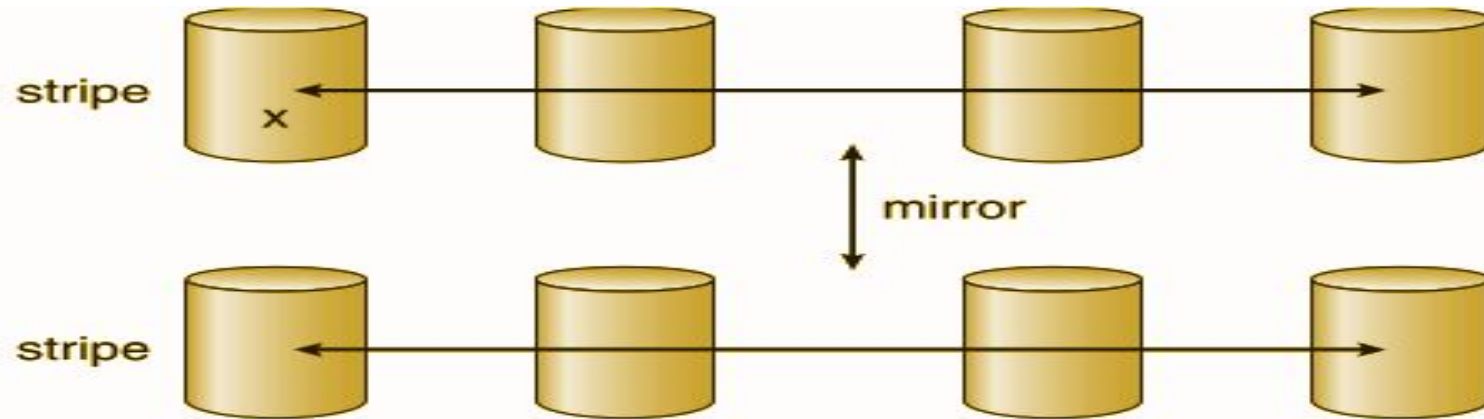
(f) RAID 5: block-interleaved distributed parity.



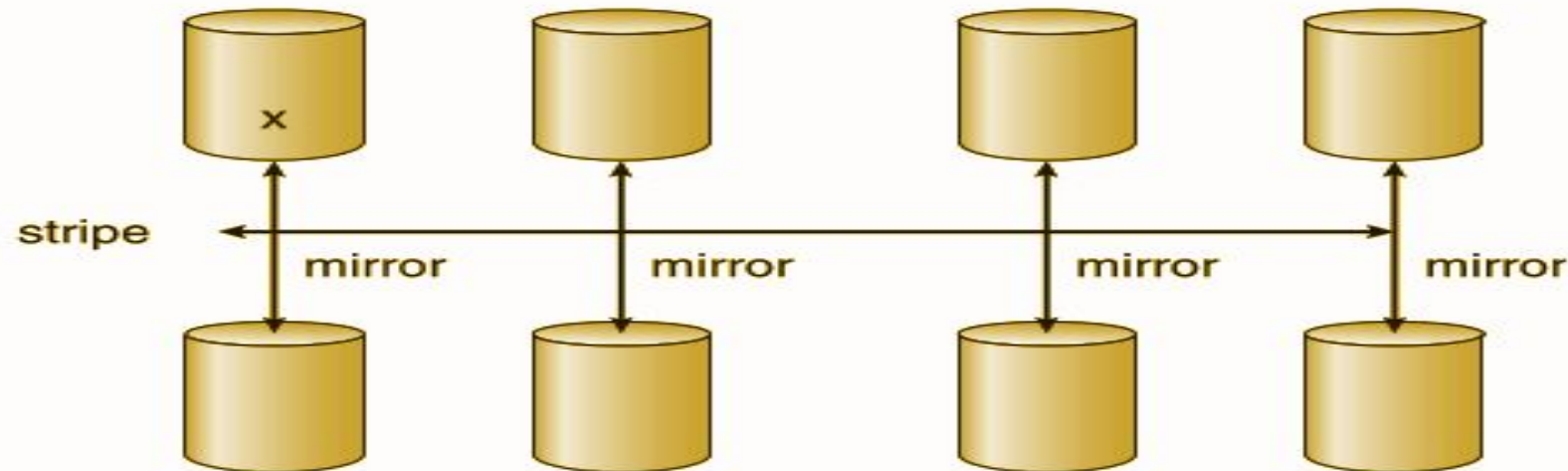
(g) RAID 6: P + Q redundancy.

OPERATING SYSTEMS

RAID (0 + 1) (1 + 0)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

- Regardless of where RAID implemented, other useful features can be added
- Snapshot is a view of file system before a set of changes take place (i.e. at a point in time)
- Replication is automatic duplication of writes between separate sites
 - For redundancy and disaster recovery
 - Can be synchronous or asynchronous
- Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible
 - Decreases mean time to repair

- **Swap-Space Management**
- **RAID Structure**
- **RAID Levels**

- File Concept
- Access Methods
- Directory and Disc Structure

File Concept

- Contiguous logical address space
 - Types:
 - Data
 - numeric
 - character
 - binary
 - Program
- Contents defined by file's creator
 - Many types
 - Consider
 - text file
 - source file
 - executable file

- **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 Systems - Additional Input

- In a computer, a file system sometimes written filesystem is the way in which files are named and where they are placed logically for storage and retrieval.
- Without a file system, stored information wouldn't be isolated into individual files and would be difficult to identify and retrieve.
- As data capacities increase, the organization and accessibility of individual files are becoming even more important in data storage.
- Digital file systems and files are named for and modeled after paper-based filing systems using the same logic-based method of storing and retrieving document
- File systems can differ between operating systems (OS), such as Microsoft Windows, macOS and Linux-based systems.
- Some file systems are designed for specific applications. Major types of file systems include distributed file systems, disk-based file systems and special purpose file systems.

File Concept - Additional Input

- File systems use its attributes and / or metadata to store and retrieve files
- Metadata is stored separately from the contents of the file, with many file systems storing the file names in separate directory entries.
- Some metadata may be kept in the directory, whereas other metadata may be kept in a structure called an inode

File Systems - Additional Input



- A file system stores and organizes data and can be thought of as a type of index for all the data contained in a storage device.
- Storage devices typically include hard drives, optical drives and flash drives
- File systems specify conventions for naming files, including the maximum number of characters in a name, which characters can be used and, in some systems, how long the file name suffix can be.
- In many file systems, file names are not case sensitive

File Systems - Additional Input

- Major file systems include the following:
 - File allocation table (FAT) is supported by the Microsoft Windows OS. FAT is considered simple and reliable, and it is modeled after legacy file systems. FAT was designed in 1977 for floppy disks, but was later adapted for hard disks. While efficient and compatible with most current OSes, FAT cannot match the performance and scalability of more modern file systems.
 - Global file system (GFS) is a file system for the Linux OS, and it is a shared disk file system. GFS offers direct access to shared block storage and can be used as a local file system.
 - GFS2 is an updated version with features not included in the original GFS, such as an updated metadata system. Under the terms of the GNU General Public License, both the GFS and GFS2 file systems are available as free software.

File Systems - Additional Input

- Major file systems include the following:
 - Hierarchical file system (HFS) was developed for use with Mac operating systems. HFS can also be referred to as Mac OS Standard, and it was succeeded by Mac OS Extended. Originally introduced in 1985 for floppy and hard disks, HFS replaced the original Macintosh file system. It can also be used on CD-ROMs.
 - The NT file system -- also known as the New Technology File System (NTFS) -- is the default file system for Windows products from Windows NT 3.1 OS onward. Improvements from the previous FAT file system include better metadata support, performance and use of disk space. NTFS is also supported in the Linux OS through a free, open-source NTFS driver. Mac OSes have read-only support for NTFS.
 - Universal Disk Format (UDF) is a vendor-neutral file system used on optical media and DVDs. UDF replaces the ISO 9660 file system and is the official file system for DVD video and audio as chosen by the DVD Forum.

File Systems and DBMS Additional Input

- Like a file system, a database management system (DBMS) efficiently stores data that can be updated and retrieved. The two are not interchangeable, however. While a file system stores unstructured, often unrelated files, a DBMS is used to store and manage structured, related data.
- A DBMS creates and defines the restraints for a database.
- A file system allows access to single files at a time and addresses each file individually.
- Because of this, functions such as redundancy are performed on an individual level, not by the file system itself.
- This makes a file system a much less consistent form of data storage than a DBMS, which maintains one repository of data that is defined once.
- The centralized structure of a DBMS allows for easier file sharing than a file system and prevents anomalies that can occur when separate changes are made to files in a file system.
- A DBMS keeps security constraints high, relying on password protection, encryption and limited authorization. More security does result in more obstacles when retrieving data, so in terms of general, simple-to-use file storage and retrieval, a file system may be preferred.

File systems definition evolves - Additional Input



- While previously referring to physical, paper files, the term file system was used to refer to digital files as early as 1961. By 1964, it had entered general use to refer to computerized file systems.
- The term file system can also refer to the part of an OS or an add-on program that supports a file system. Examples of such add-on file systems include the Network File System (NFS) and the Andrew File System (AFS).
- In addition, the term has evolved to refer to the hardware used for nonvolatile storage, the software application that controls the hardware and architecture of both hardware and software.

- File is an **abstract data type**
- **Create**
 - Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.
- **Write – at write pointer location**
 - Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written

- **Read** – at **read pointer** location
 - Every file is opened in three different modes : Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.
- **Reposition within file - seek**
 - Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.
- **Delete**
 - Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.

- **Truncate**
 - Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file get replaced.
- **Open(F_i)** – search the directory structure on disk for entry F_i , and move the content of entry to memory
- **Close (F_i)** – move the content of entry F_i in memory to directory structure on disk

- Several pieces of data are needed to manage open files:
 - **Open-file table:** tracks open files
 - File pointer: pointer to last read/write location, per process that has the file open
 - **File-open count:** counting of number of times a file is open – to allow removal of data from open-file table when last processes closes it
 - Disk location of the file: cache of data access information
 - Access rights: per-process access mode information

- Provided by some operating systems and file systems
 - Similar to reader-writer locks
 - **Shared lock** similar to reader lock – several processes can acquire concurrently
 - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
 - **Mandatory** – access is denied depending on locks held and requested
 - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

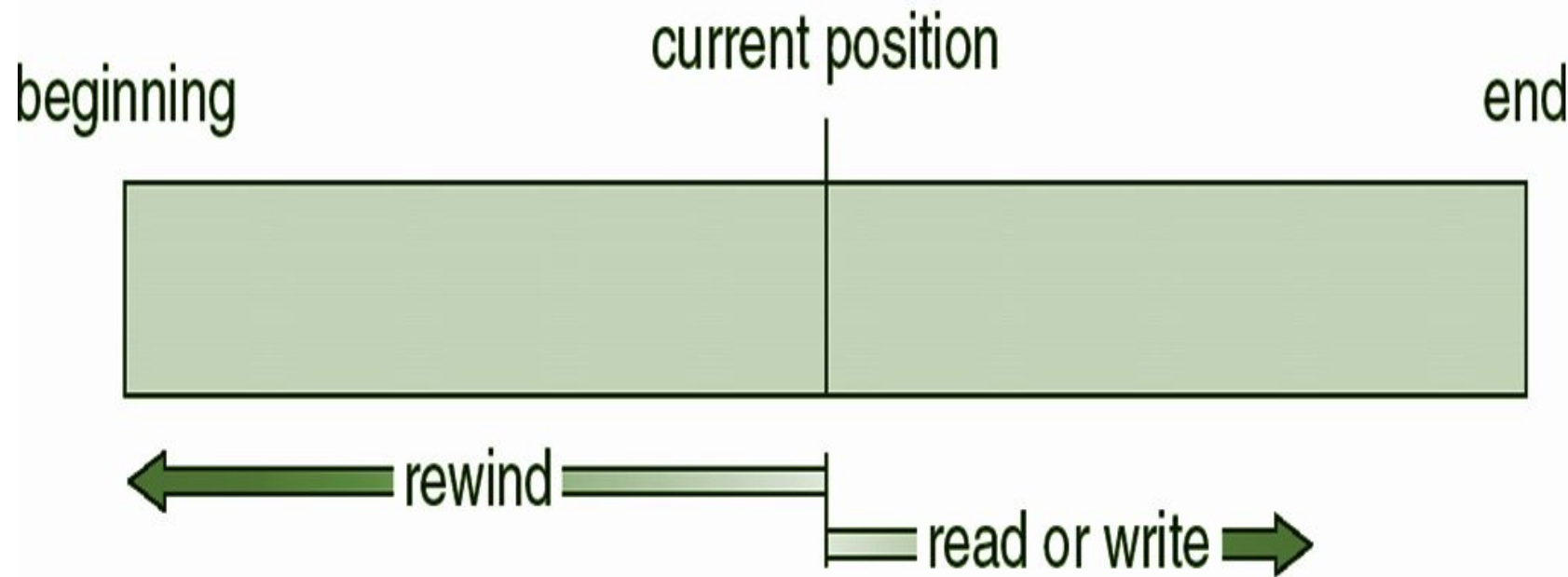
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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

- None - sequence of words, bytes
- Simple record structure
 - Lines
 - Fixed length
 - Variable length
- Complex Structures
 - Formatted document
 - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
 - Operating system
 - Program

- **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

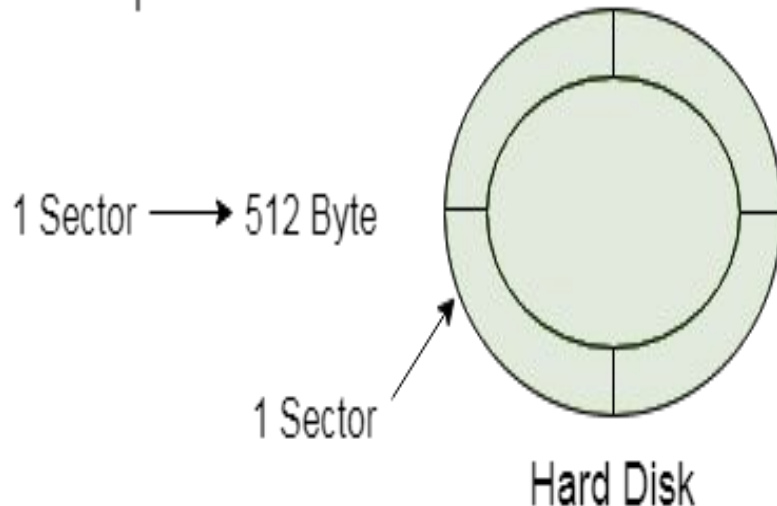
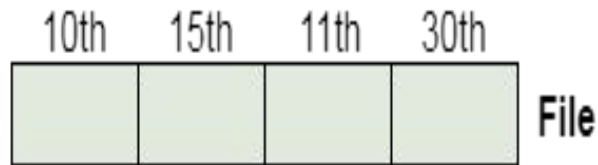
Sequential Access - File



sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp + 1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp + 1;</i>

Sequential Access - File - Additional Input

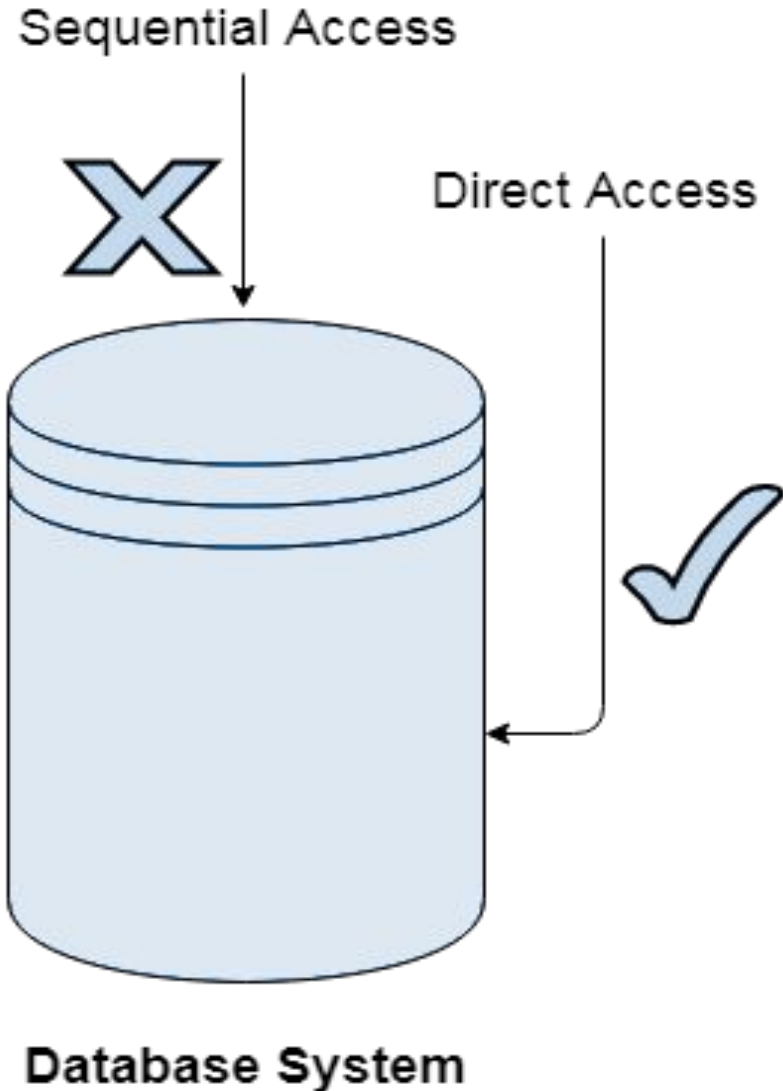
1 block → 512 Byte



- Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.
- In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file.
- If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.
- Modern word systems do provide the concept of direct access and indexed access but the most used method is sequential access due to the fact that most of the files such as text files, audio files, video files, etc need to be sequentially accessed.

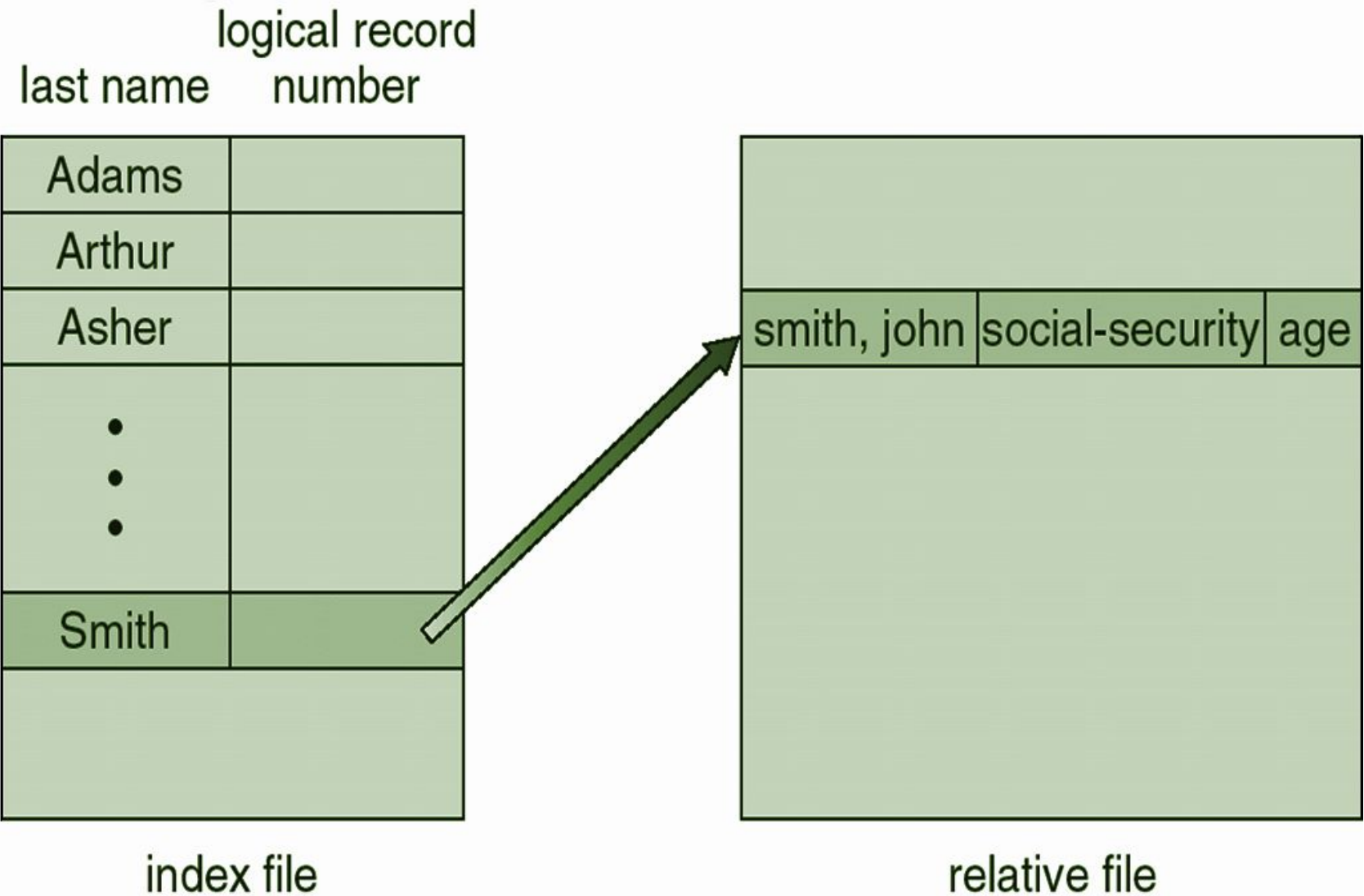
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed 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

Direct Access - File - Additional Input



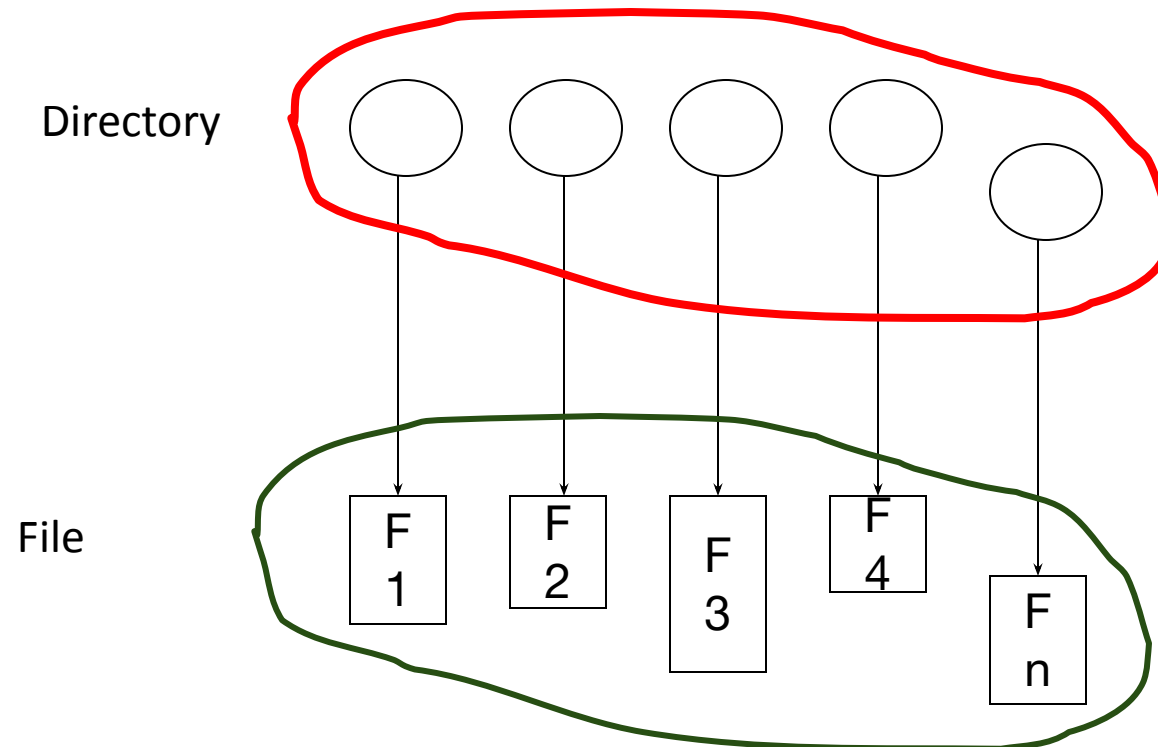
- The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.
- Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.
- Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.

Example of Index and Relative Files



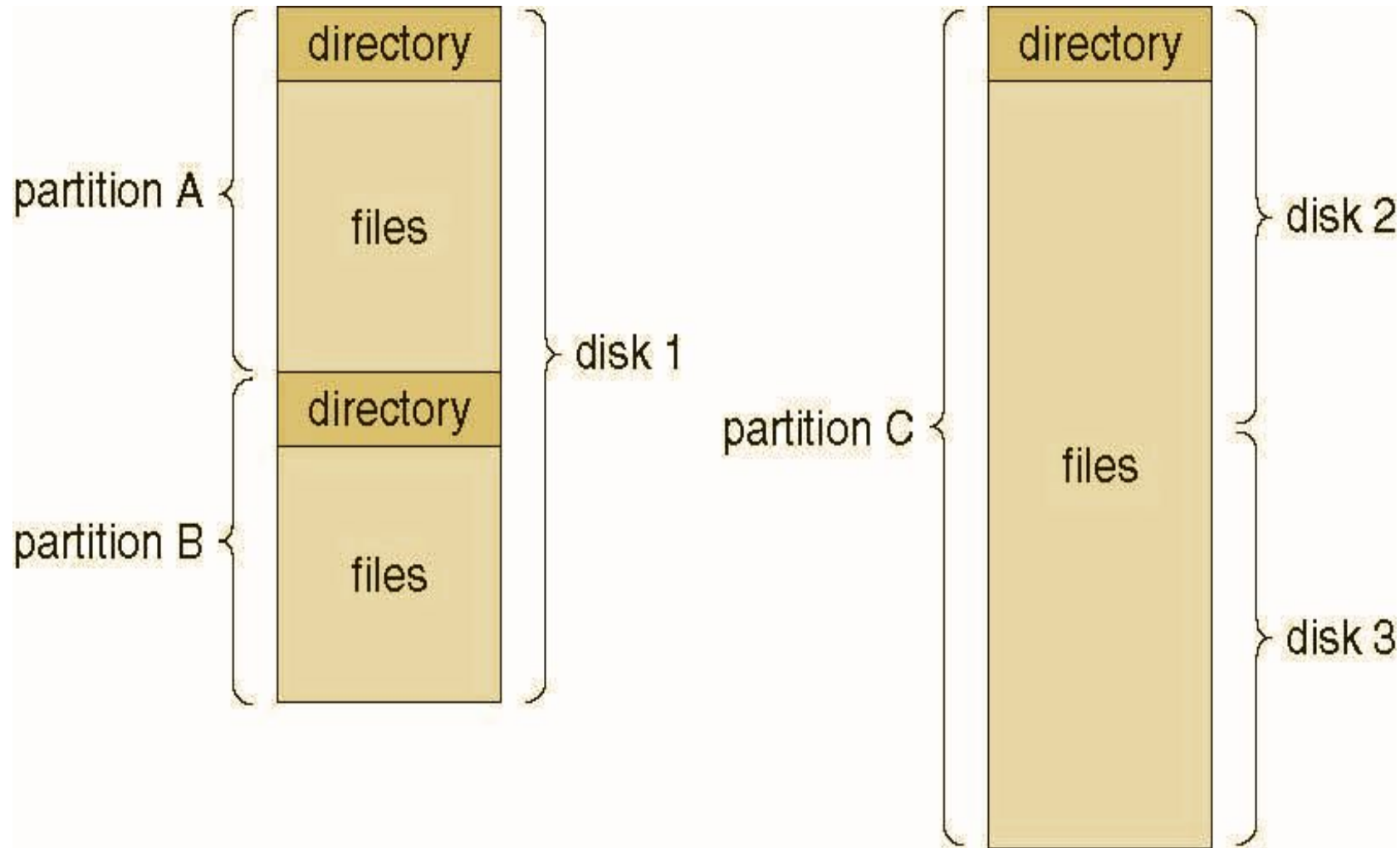
Directory Structure

- A collection of nodes containing information about all files
- Both the directory structure and the files reside on disk



- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

Typical File-system Organization



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
- Consider Solaris has
 - 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

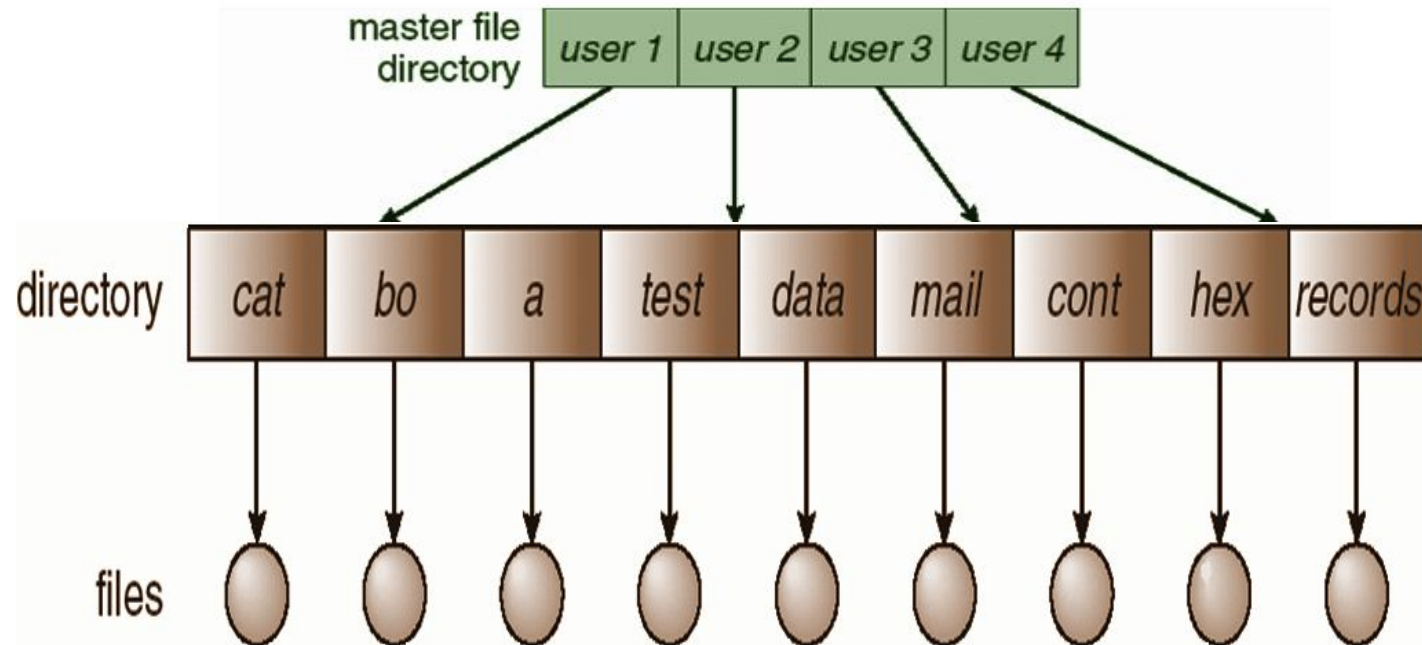
Operations Performed on Directory

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

- 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

- A single directory for all users



- Naming problem
- Grouping problem

Single-Level Directory

- Single level directory is simplest directory structure. In it all files are contained in same directory which make it easy to support and understand.
- A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user.
- Since all the files are in the same directory, they must have the unique name . if two users call their dataset test, then the unique name rule violated.

Advantages:

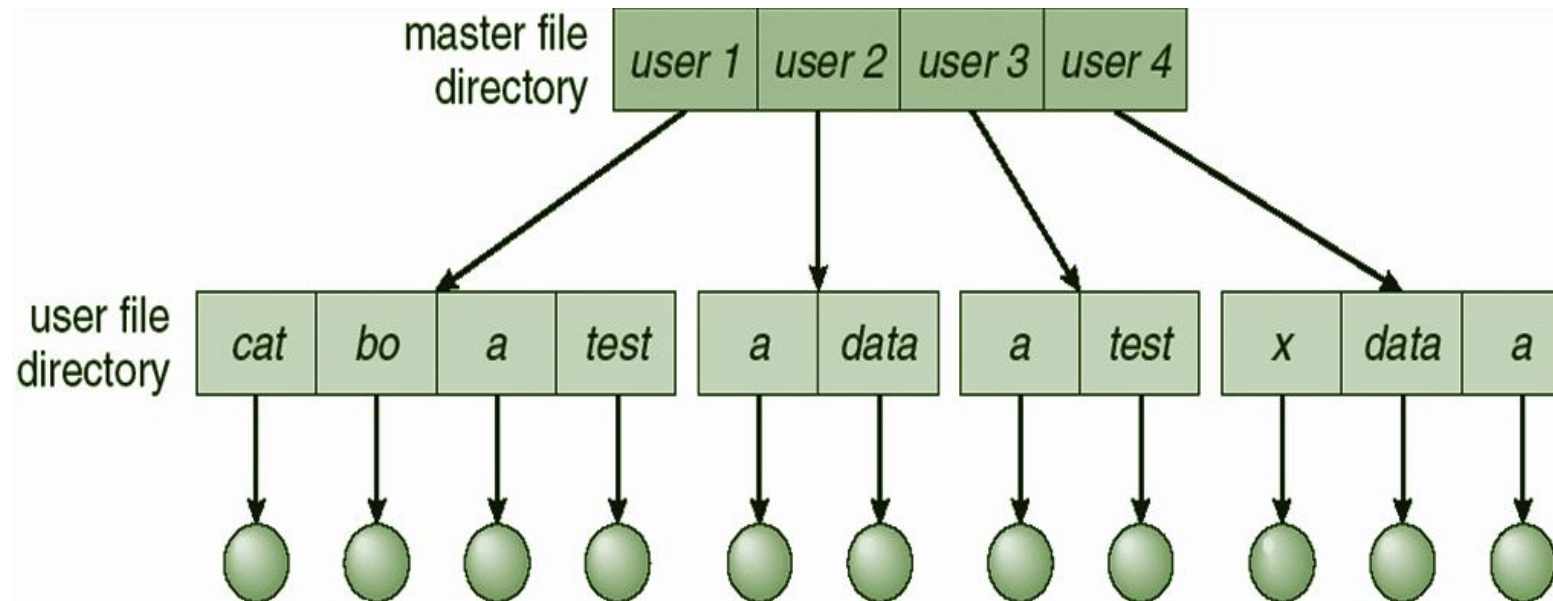
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

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.
- In this cannot group the same type of files together.

Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Two-Level Directory

- In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user. System's *master file directory (MFD)* is searched whenever a new user ID is logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

Advantages:

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

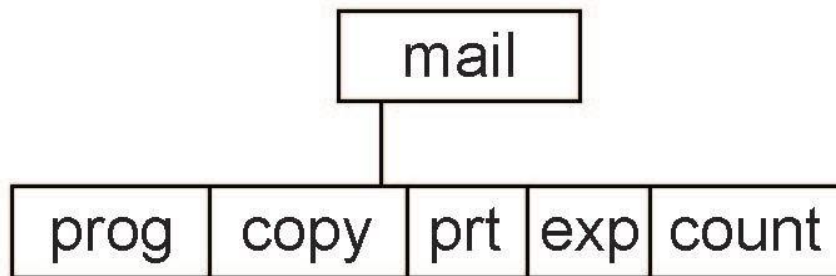
Disadvantages:

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

- Efficient searching
- Grouping Capability
- Current directory (working directory)
 - **cd /spell/mail/prog**
 - **type list**

Tree Structured Directory

- **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”

Advantages:

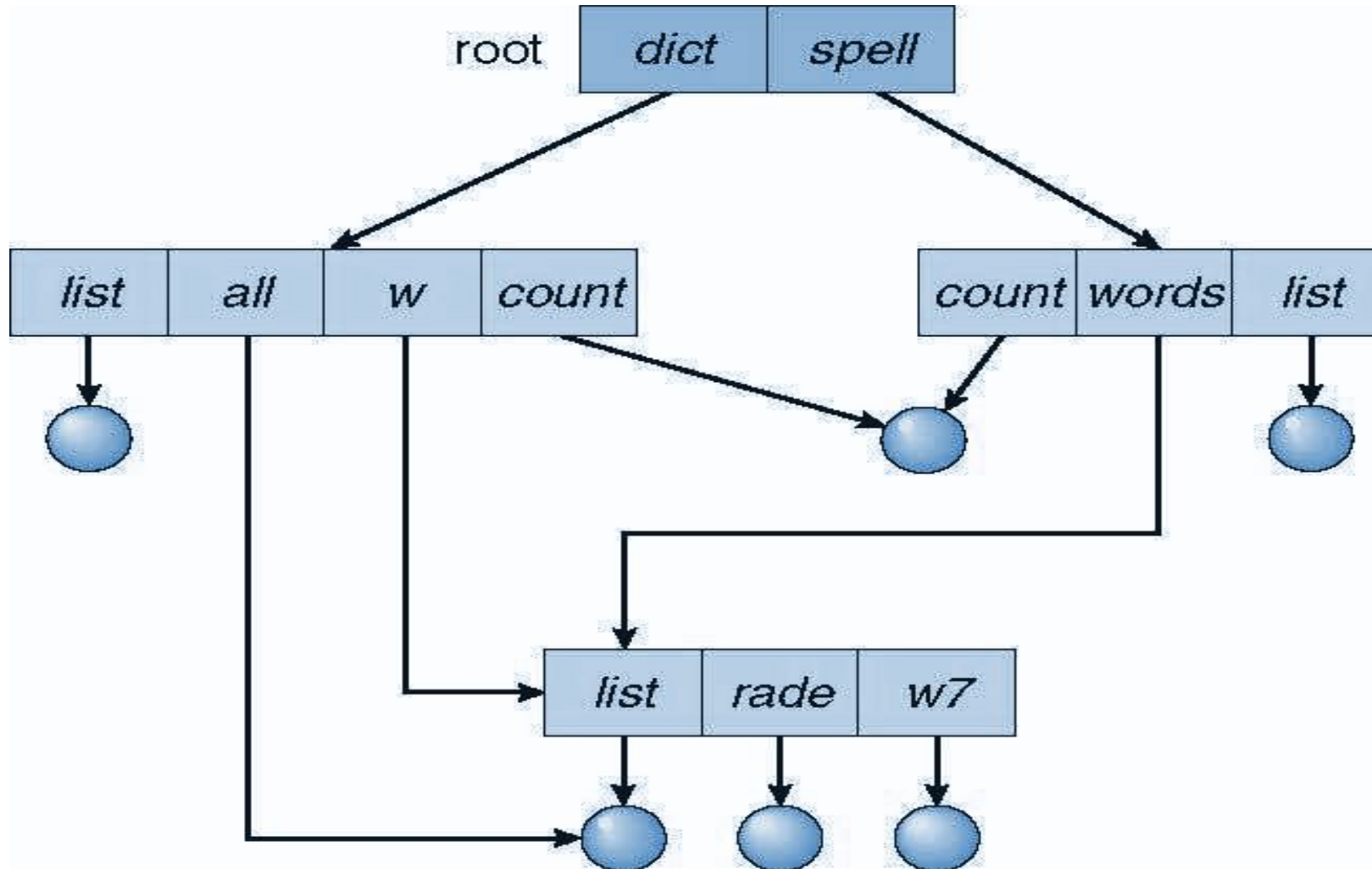
- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

Acyclic Graph Directory

- Have shared subdirectories and files



- Two different names (aliasing)
- If **dict** deletes **list** \Rightarrow 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

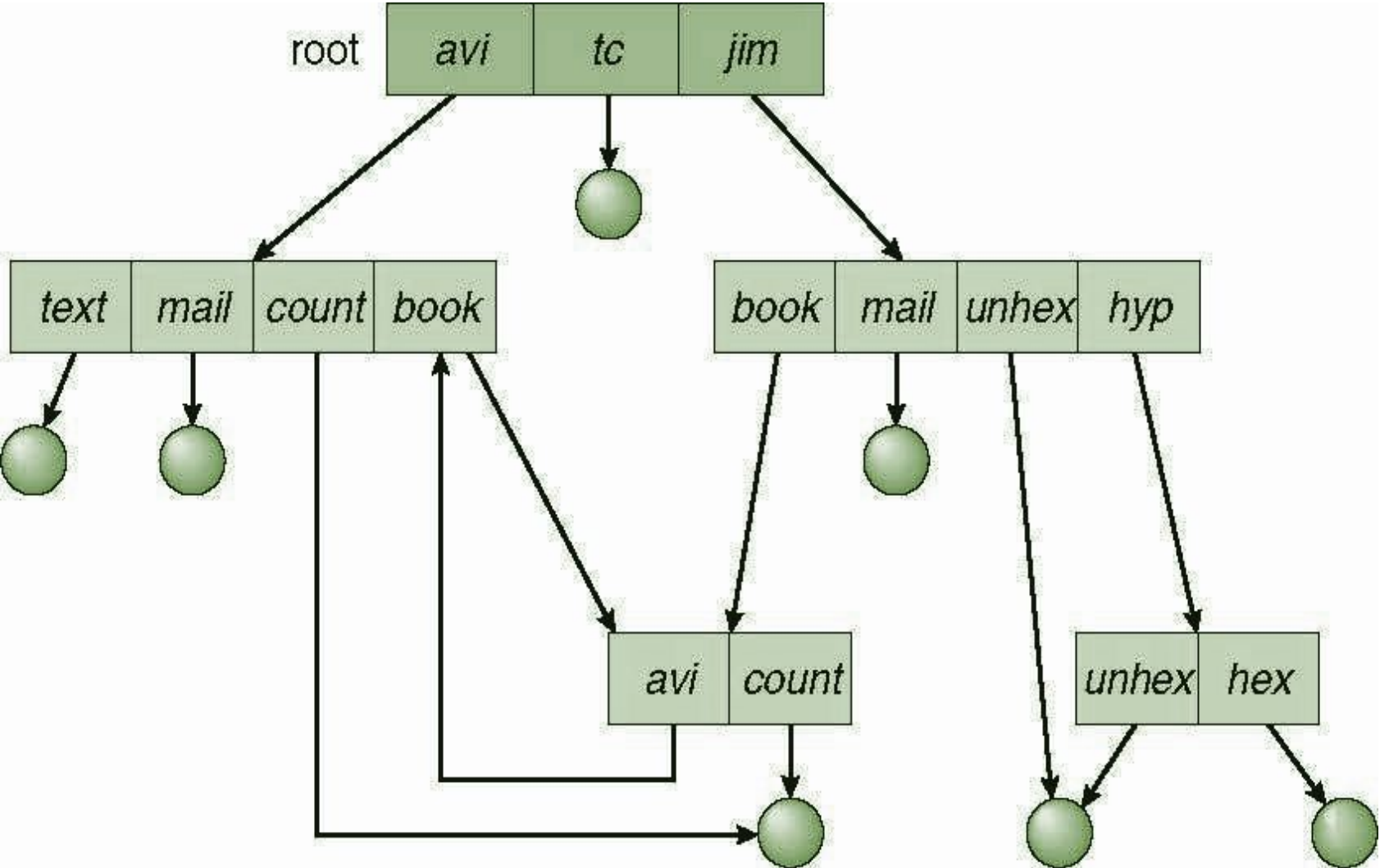
Advantages:

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

Disadvantages:

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

General Graph Directory



- 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

Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

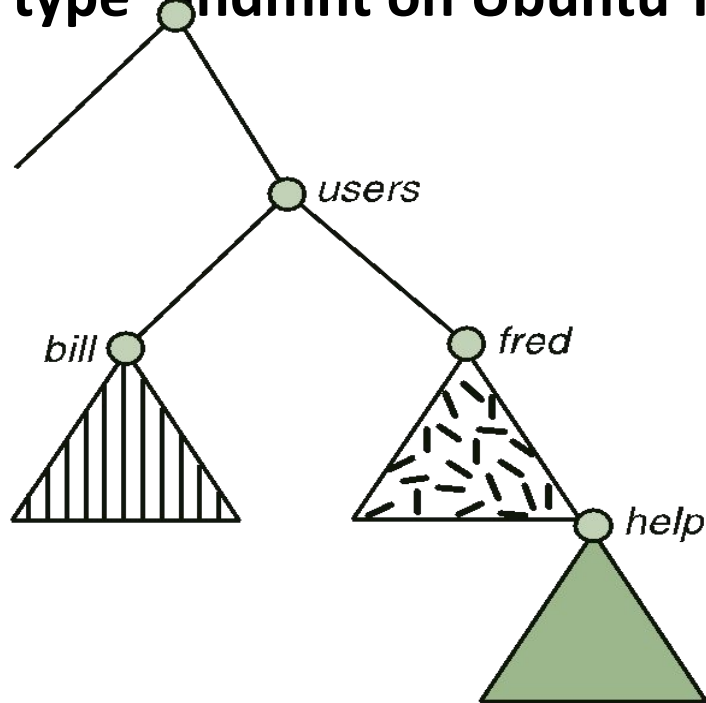
- It is more costly than others.
- It needs garbage collection.

- **File Concept**
- **Access Methods**
- **Directory and Directory Structure**

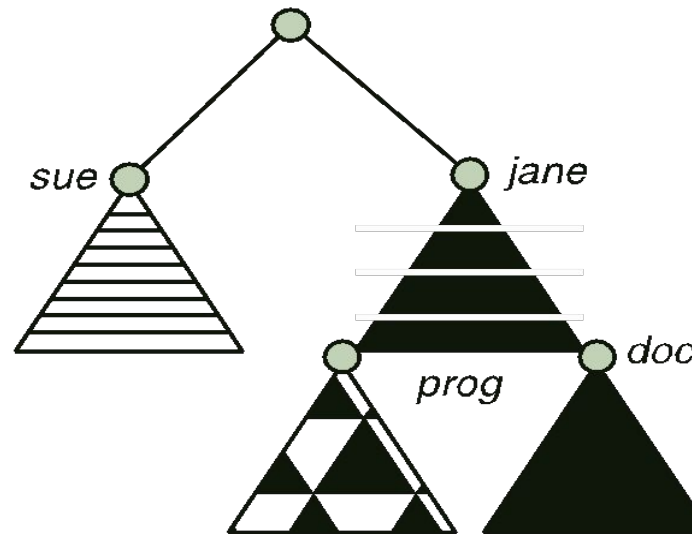
- File System Mounting
- File Sharing
- Protection

File System Mounting

- A file system must be **mounted** before it can be accessed
- An unmounted file system Figure b is mounted at a **mount point**
- type `findmnt` on Ubuntu Terminal



(a)



(b)

File System Mounting

- Before you can access the files on a file system, you need to mount the file system. Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system. The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.
- The mount point is the directory (usually an empty one) in the currently accessible filesystem to which a additional filesystem is mounted.
- It becomes the **root directory** of the added directory tree, and that tree becomes accessible from the directory to which it is mounted (i.e., its mount point).
- Any original contents of a directory that is used as a mount point become invisible and inaccessible while the filesystem is still mounted.

File System Mounting

- The /mnt directory exists by default on all Unix-like systems. It, or usually its subdirectories (such as /mnt/floppy and /mnt/usb), are intended specifically for use as mount points for removable media such as CDROMs, USB key drives and floppy disks.
- On some operating systems, everything is mounted automatically by default so that users are never even aware that there is any such thing as mounting. Linux and other Unix-like systems can likewise be configured so that everything is mounted by default, as a major feature of such systems is that they are highly configurable.

File System Mounting

- However, they are not usually set up this way, for both safety and security reasons. Moreover, only the root user (i.e., administrative user) is generally permitted by default to mount devices and filesystems on such systems, likewise as safety and security measures.
- In the simplest case, such as on some personal computers, the entire filesystem on a computer running a Unix-like operating system resides on just a single partition, as is typical for Microsoft Windows systems. More commonly, it is spread across several partitions, possibly on different physical disks or even across a network. Thus, for example, the system may have one partition for the root directory, a second for the /usr directory, a third for the /home directory and a fourth for use as swap space. (Swap space is a part of HDD that is used for virtual memory, which is the simulation of additional main memory).

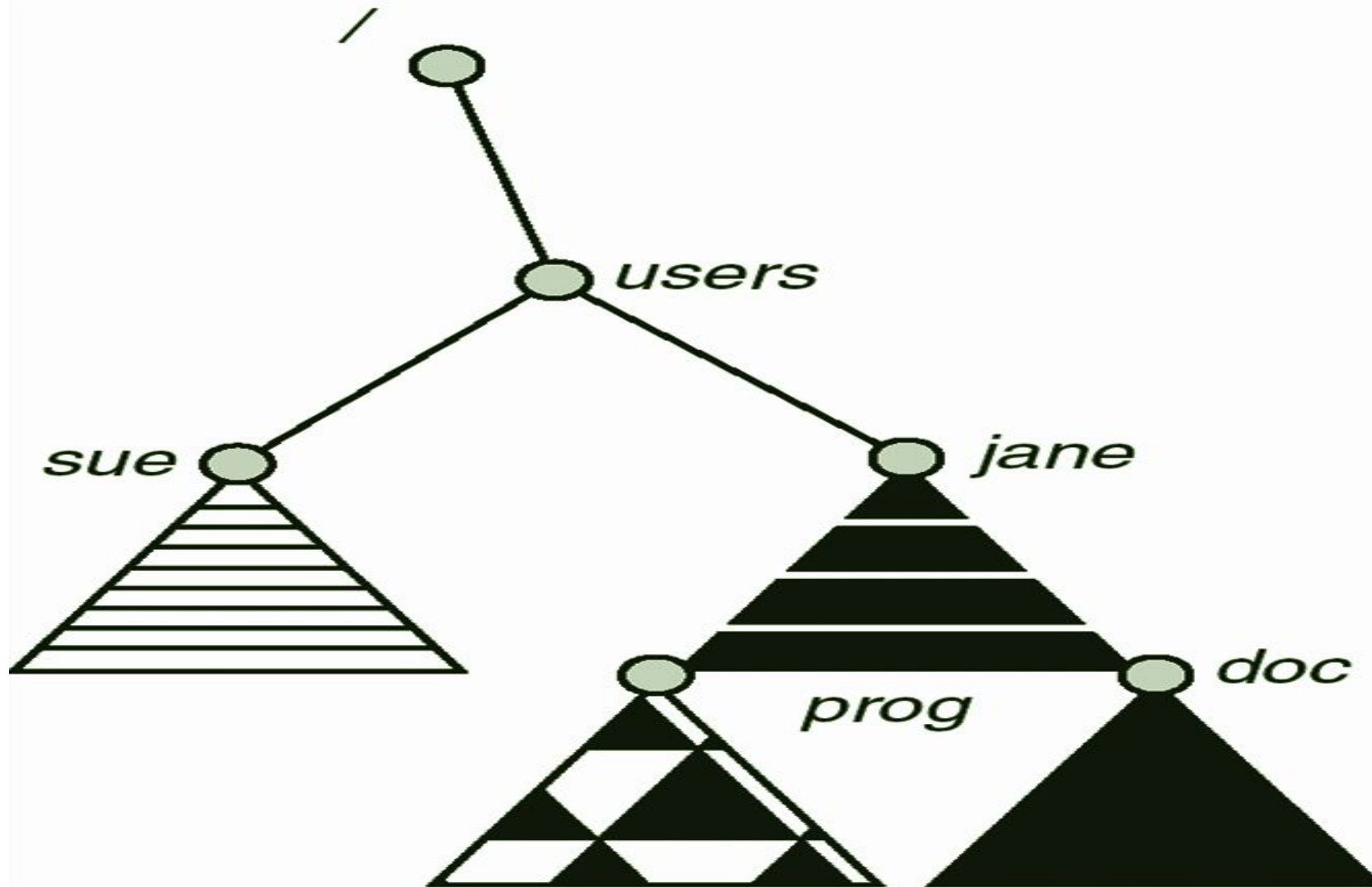
File System Mounting

- The only partition that can be accessed immediately after a computer boots (i.e., starts up) is the root partition, which contains the root directory, and usually at least a few other directories as well. The other partitions must be attached to this root filesystem in order for an entire, multiple-partition filesystem to be accessible. Thus, about midway through the boot process, the operating system makes these non-root partitions accessible by mounting them on to specified directories in the root partition.
- Systems can be set up so that external storage devices can be mounted automatically upon insertion. This is convenient and is usually satisfactory for home computers. However, it can cause security problems, and thus it is usually not (or, at least, should not be) permitted for networked computers in businesses and other organizations. Rather, such devices must be mounted manually after insertion, and such manual mounting can only be performed by the root account.

File System Mounting

- Mounting can often be performed manually by the root user by merely using the mount command followed by the name of the device to be mounted and its mounting destination (but in some cases it is also necessary to specify the type of filesystem). For example, to mount the eighth partition on the first HDD, which is designated by /dev/hda8, using a directory named /dir8 as the mount point, the following could be used: `mount /dev/hda8 /dir8`
- Removing the connection between the mounted device and the rest of the filesystem is referred to as unmounting. It is performed by running the umount (with no letter n after the first u) command, likewise followed by the name of the device to be unmounted and its mount point. For example, to unmount the eighth partition from the root filesystem, the following would be used:
`umount /dev/hda8 /dir8`

Mount Point



- Sharing of files on 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 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
 - Group of a file / directory

File Sharing - Remote File Systems



- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** - Common Internet File System is standard Windows protocol
 - Standard operating system file calls are translated into remote calls

- Distributed Information Systems (**distributed naming services**) such as
 - LDAP(Lightweight Directory Access Protocol),
 - DNS (Domain Name System)
 - NIS (Network Information Service)
 - Active Directory implement unified access to information needed for remote computing

- 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

- Specify how multiple users are to access a shared file simultaneously
 - Similar to process synchronization algorithms
 - Tend to be less complex due to disk I/O and network latency (for remote file systems)
 - Andrew File System (AFS) implemented complex remote file sharing semantics
 - Unix file system (UFS) implements:
 - Writes to an open file visible immediately to other users of the same open file
 - Sharing file pointer to allow multiple users to read and write concurrently
 - AFS has session semantics
 - Writes only visible to sessions starting after the file is closed

- 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

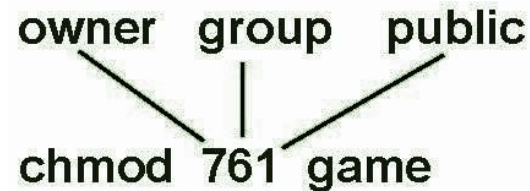
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

 RWX
a) **owner access** 7 \Rightarrow 1 1 1

 RWX
b) **group access** 6 \Rightarrow 1 1 0

 RWX
c) **public access** 1 \Rightarrow 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.

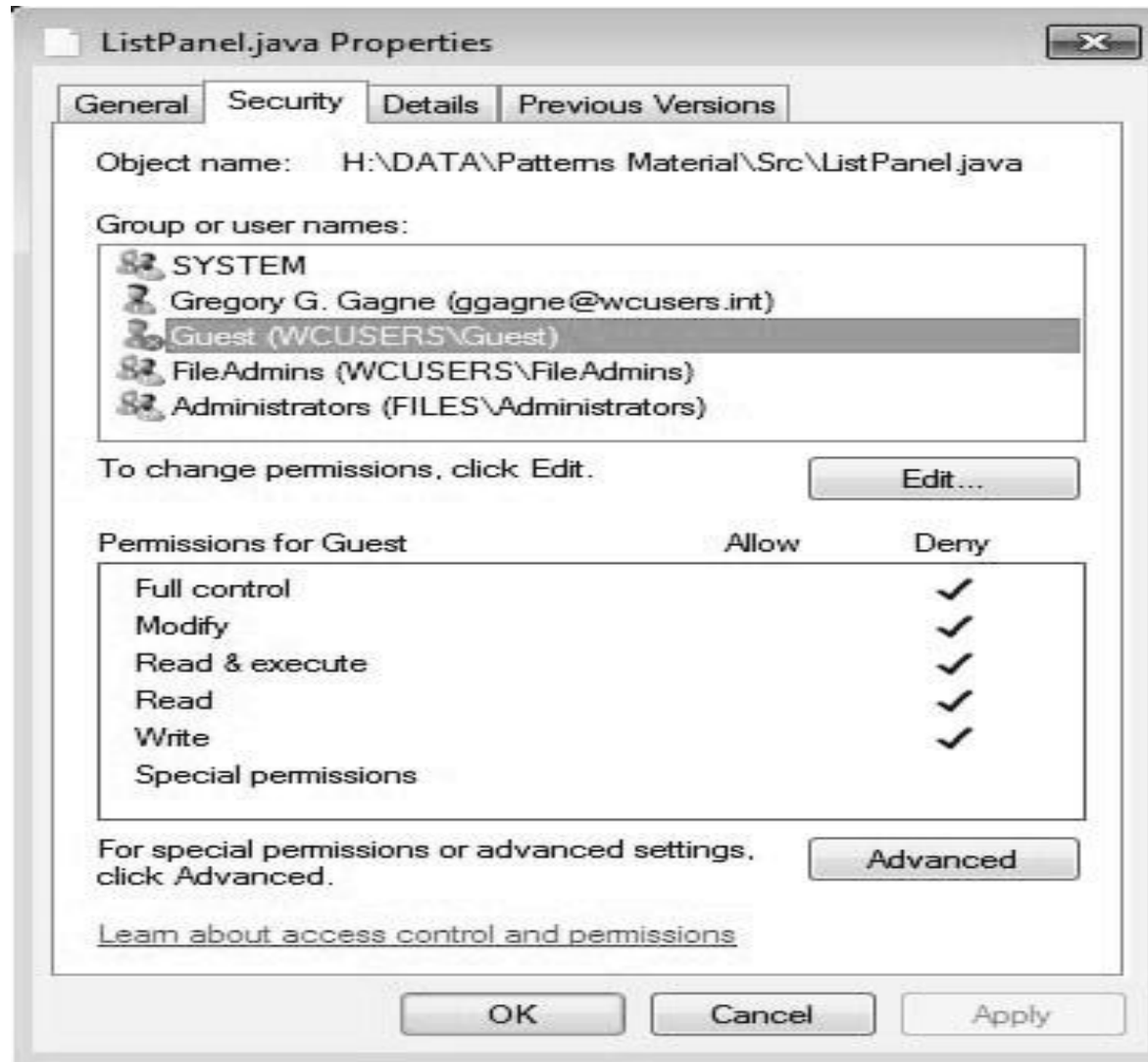


```
graph TD
    owner --> 7
    group --> 6
    public --> 1
    7 --- chmod
    6 --- chmod
    1 --- chmod
    chmod --- game
```

Attach a group to a file

chgrp G game

Access Lists and Groups - Windows Example



Access Lists and Groups - Linux Example

```
sridatta@sridatta:~$ ls -l
total 155248
drwxr-xr-x 26 sridatta sridatta      4096 Aug 23 19:18 anaconda3
drwxrwxr-x  6 sridatta sridatta      4096 Oct  9 05:25 AQRS_2020_Final_Sep6_2020
drwxrwxr-x  7 sridatta sridatta      4096 Sep 19 19:15 Aspirations_Python
drwxr-xr-x  6 sridatta sridatta      4096 Oct 24 05:35 Desktop
drwxr-xr-x  2 sridatta sridatta      4096 Sep 24 09:50 Documents
drwxr-xr-x  3 sridatta sridatta      4096 Oct 24 05:19 Downloads
drwxrwxr-x  2 sridatta sridatta      4096 Aug 11 08:05 DTP
-rw-rw-r--  1 sridatta sridatta 36920918 Oct 21 20:36 'From the COEs Desk.m4v'
drwxrwxr-x  2 sridatta sridatta      4096 Sep 25 06:28 Live_Different_Video_Format
drwxr-xr-x  2 sridatta sridatta      4096 Aug 11 04:51 Music
drwxrwxr-x  2 sridatta sridatta      4096 Aug 11 08:04 OBS_Output
drwxrwxr-x  4 sridatta sridatta      4096 Oct 18 19:34 PESU_DD_Status
drwxr-xr-x  3 sridatta sridatta      4096 Aug 20 11:08 Pictures
drwxr-xr-x  2 sridatta sridatta      4096 Aug 11 04:51 Public
drwxrwxr-x  8 sridatta sridatta      4096 Aug 11 08:37 rtl8188fu
drwxrwxr-x  9 sridatta sridatta      4096 Sep  6 05:30 rtl8821ce
drwxr-xr-x  9 sridatta sridatta      4096 Sep 24 06:24 snap
drwxrwxr-x  2 sridatta sridatta      4096 Sep 20 06:54 StaticLinkingDemo
drwxr-xr-x  2 sridatta sridatta      4096 Aug 11 04:51 Templates
drwxrwxr-x  2 sridatta sridatta      4096 Sep 19 22:01 thumbnail
-rw-rw-r--  1 sridatta sridatta 121964532 Oct  7 20:42 UE18CS302_071020_RC7_NVP_Compressed.mp4
drwxr-xr-x  3 sridatta sridatta      4096 Sep 23 21:28 Videos
sridatta@sridatta:~$
```

Access Lists and Groups - Linux Example



Live_Different_Video_Format Properties

Basic

Permissions

Local Network Share

Owner:

Me

Access:

Create and delete files

Group:

sridatta

Access:

Create and delete files

Others

Access:

Access files

Security context:

unknown

Change Permissions for Enclosed Files...

Live_Different_Video_Format Properties

Basic

Permissions

Local Network Share

Owner:

Me

Cancel

Change Permissions for Enclosed Files

Change

Files

Owner:

Read and write

Folders

Create and delete files

Group:

Read and write

Create and delete files

Others:

Read-only

Access files

Security context:

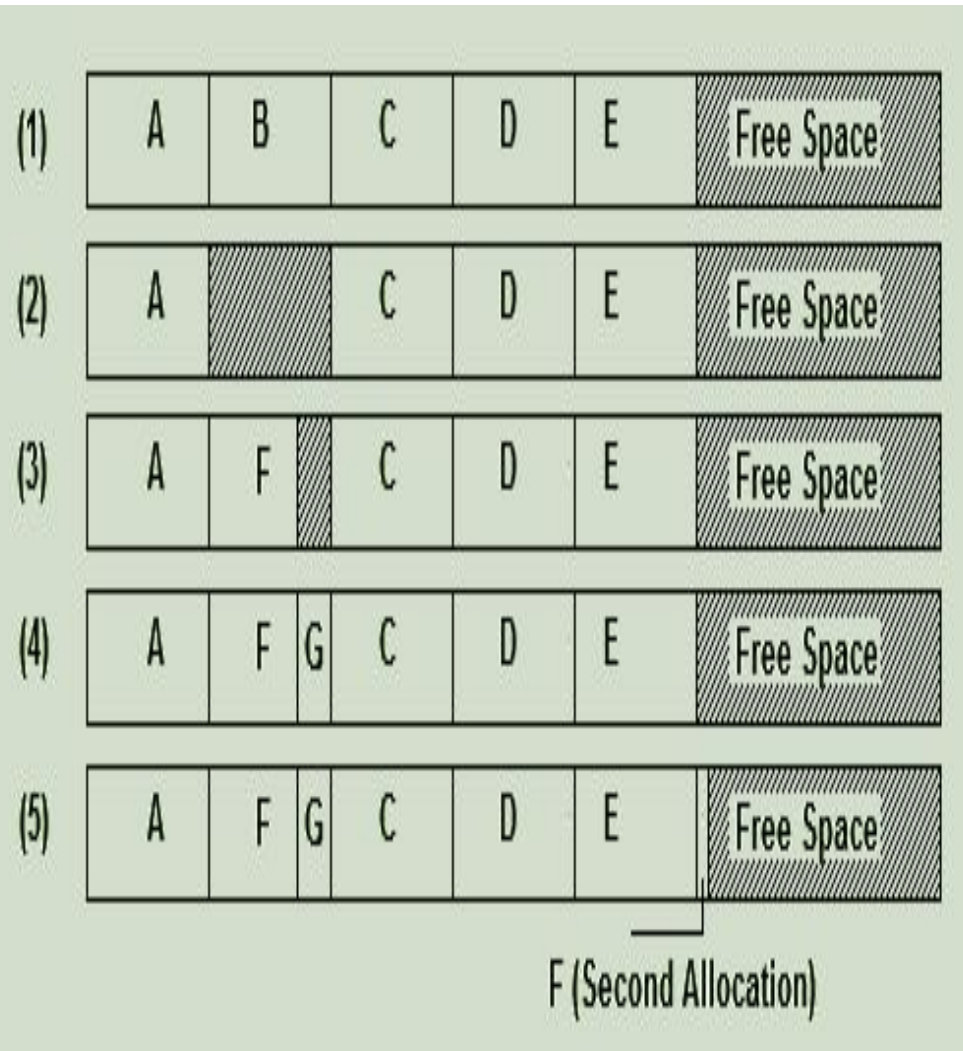
unknown

Change Permissions for Enclosed Files...

- File System Mounting
- File Sharing
- Protection

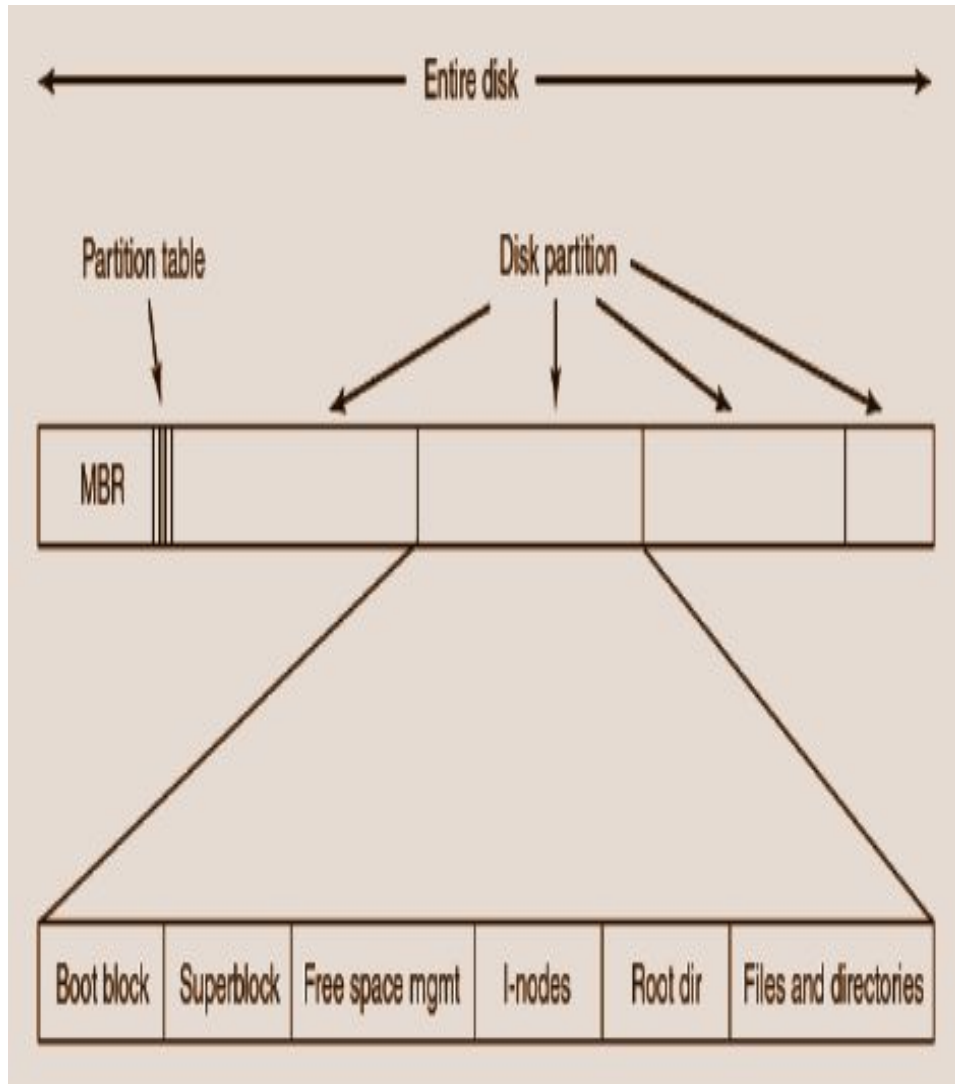
- **File-System Structure**
- **File-System Implementation**
- **Virtual File Systems**
- **Virtual File Systems Implementation**

- File structure
 - Logical storage unit
 - Collection of related information
- **File system** resides on secondary storage (disks)
 - Provides user interface to storage, mapping logical to physical
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily



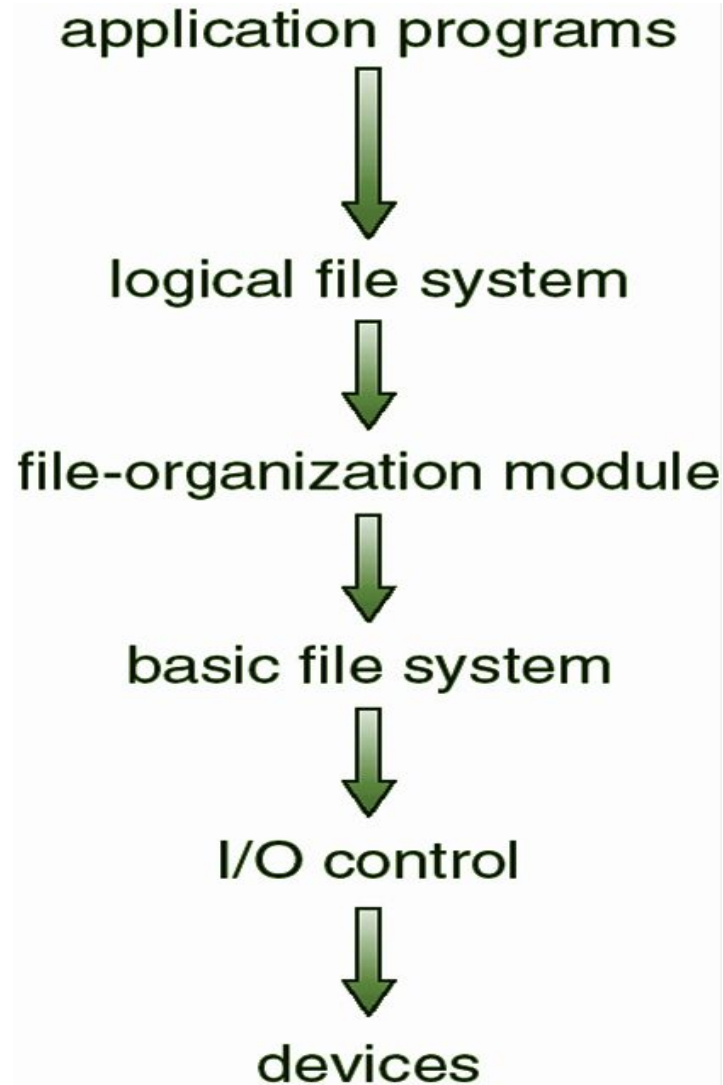
- File system implementation defines how files and directories are stored, how disk space is managed, and how to make everything work efficiently and reliably.

File System Structure



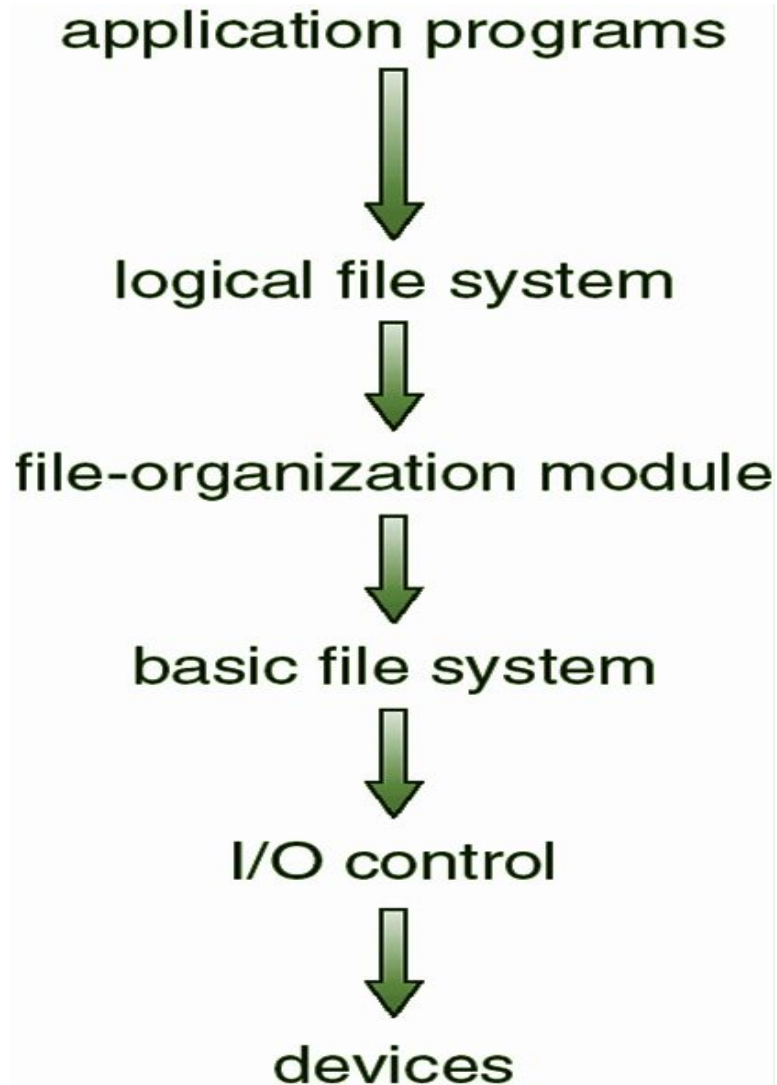
- File Systems are stored on disks. The figure depicts a possible File-System Layout.
- **MBR**: Master Boot Record is used to boot the computer
- **Partition Table**: Partition table is present at the end of MBR. This table gives the starting and ending addresses of each partition.
- **Boot Block**: When the computer is booted, the BIOS reads in and executes the MBR. The first thing the MBR program does is locate the active partition, read in its first block, which is called the boot block, and execute it. The program in the boot block loads the operating system contained in that partition. Every partition contains a boot block at the beginning though it does not contain a bootable operating system.
- **Super Block**: It contains all the key parameters about the file system and is read into memory when the computer is booted or the file system is first touched.

- Disk provides in-place rewrite and random access
 - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers



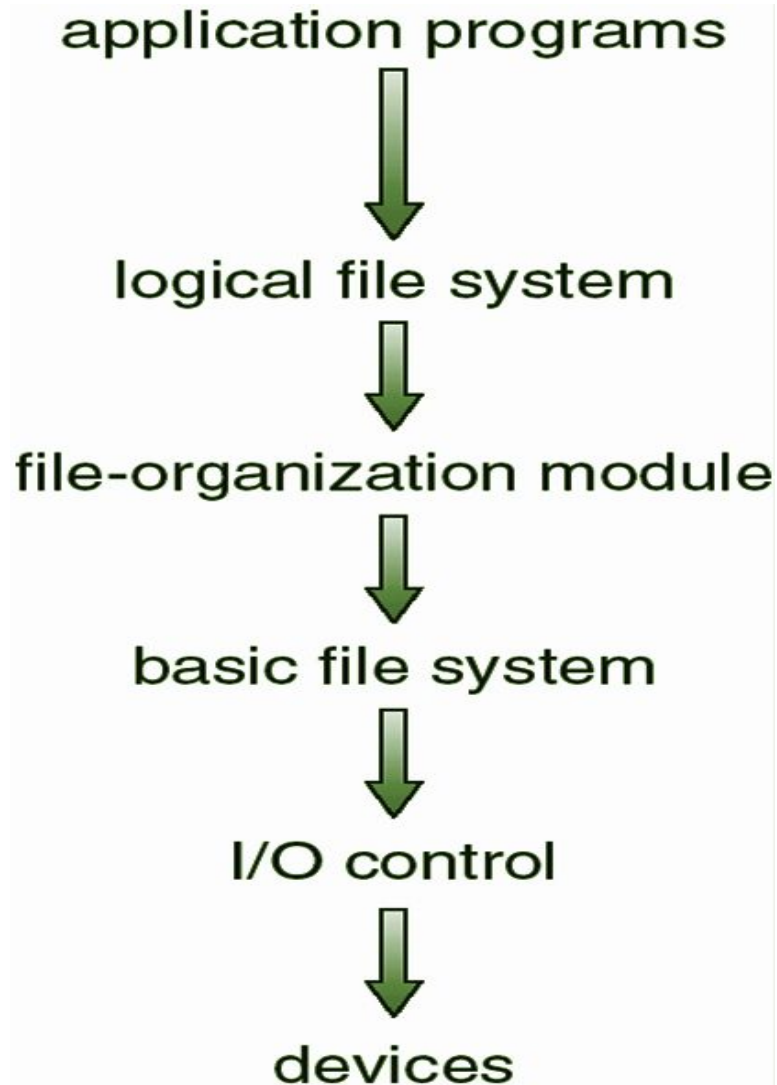
- **Device drivers** manage I/O devices at the I/O control layer
 - Given commands like “read drive1, cylinder 72, track 2, sector 10, into memory location 1060” outputs low-level hardware specific commands to hardware controller
- Basic file system given command like “retrieve block 123” translates to device driver

Layered File System



- Also manages memory buffers and caches (allocation, freeing, replacement)
 - Buffers hold data in transit
 - Caches hold frequently used data
- File organization module understands files, logical address, and physical blocks
 - Translates logical block # to physical block #
 - Manages free space, disk allocation

Layered File System



- Many file systems, sometimes many within an operating system
 - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with **extended file system** ext2 and ext3 leading; plus distributed file systems, etc.)
 - New ones still arriving – ZFS, GoogleFS, Oracle Automatic Storage Management - ASM, Filesystem in Uersspace - FUSE

File System Implementation

- We have system calls at the API level, but how do we implement their functions ?
 - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
 - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
 - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
 - Names and inode numbers, master file table

File System Implementation

- A storage volume is an identifiable unit of data storage. It can be a removable hard disk, but it does not have to be a unit that can be physically removed from a computer or storage system.
- Each storage volume has a system-unique name or number that allows a user to identify it. In some systems, the physical unit may be divided into several identifiable volumes.
- A partition is a logical division of a hard disk. It is often created to have different operating systems (OSes) on the same hard disk and is generated when a user formats a disk.
- The main difference between a storage volume and partition is the type of disk used. A volume is created on a dynamic disk -- a logical structure that can span multiple physical disks -- while a partition is created on a basic disk
- A volume is also more flexible than a partition. Storage volumes can expand or contract, and they use mirroring and striping. On a physical server, the OS is typically installed on a partition, while everything else is installed on a volume. In addition, volumes support multiple disks that can be organized into various RAID structures that protect stored data in the event of a hardware failure.

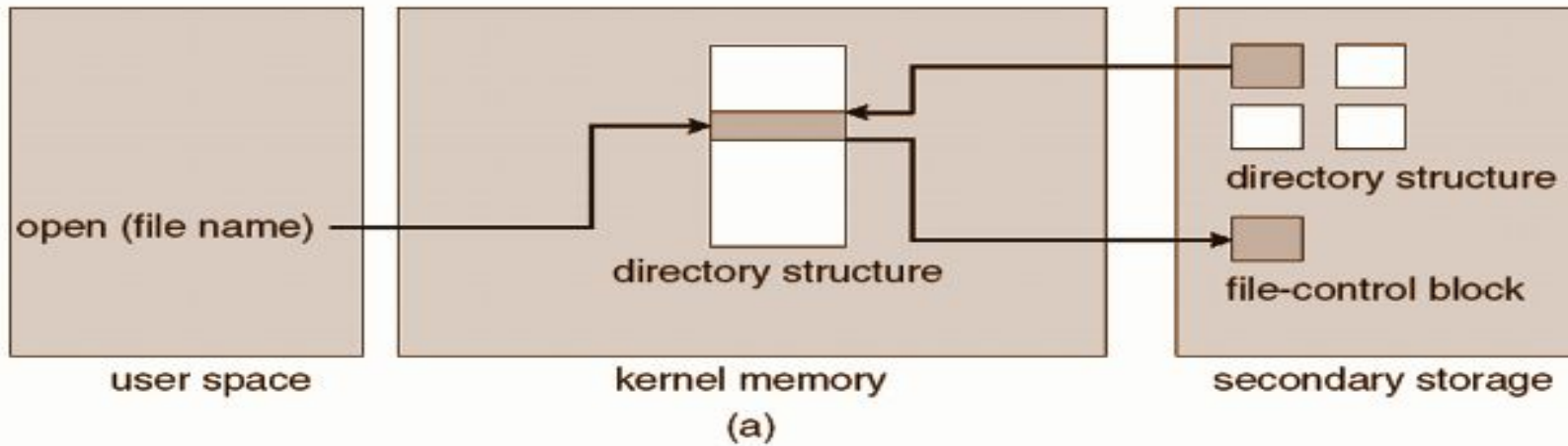
File System Implementation

- Volume is the label of the drive or the partition.
- Partition is a “carved out” segment of the drive memory.
- For example, your computer has a hard drive where Windows / Linux runs and all your programs and probably files are stored. Usually the label (volume) of this drive is “C”.
- A partition is a portion of a hard disk drive that functions as though it is a physically separate drive. After the partition is formatted and assigned a drive letter, the partition is called a drive.
- For example, if you have one hard disk drive and you create two partitions, you can install the operating system and applications on the first partition and store user data on the second partition. If you want to create three partitions, you can use one partition for the operating system, one partition for data folders that are used by applications, and one partition for user data.
- Once a Partition is formatted and given a drive letter it is called a Volume. This can be a physical volume (on one partition of a physical disk) or a logical volume (which sits on RAID Partition for instance that spans multiple physical hard drives).

- Per-file **File Control Block (FCB)** contains many details about the file
 - inode number, permissions, size, dates
 - NFTS stores into in master file table using relational DB structures

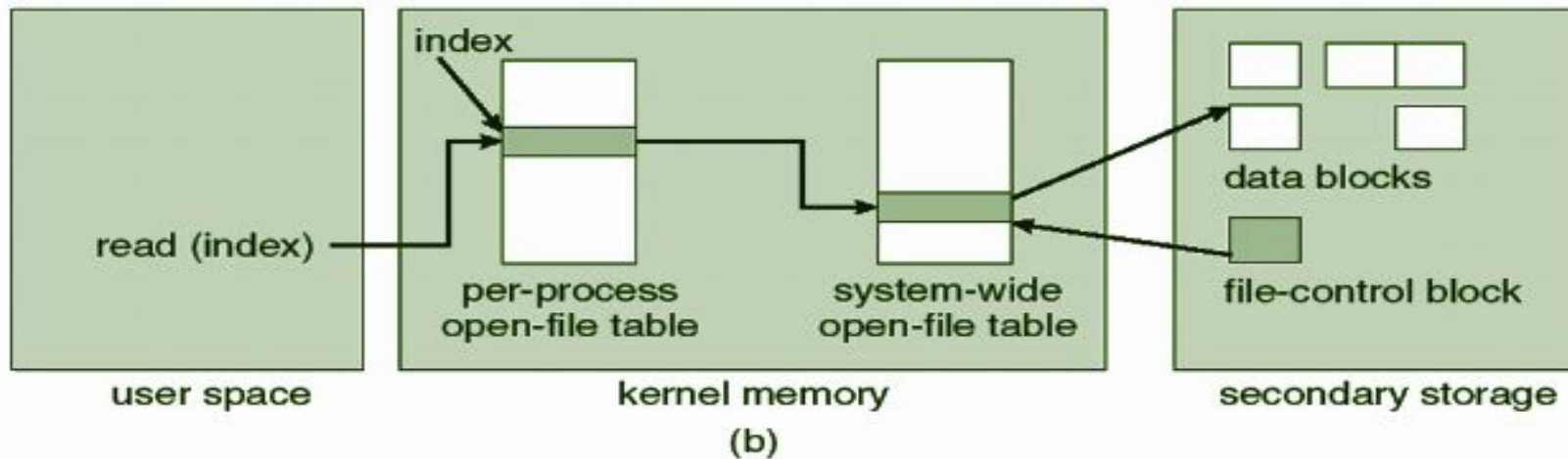
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures



- Mount table storing file system mounts, mount points, file system types
- The above figure illustrates the necessary file system structures provided by the operating systems
- Figure (a) refers to opening a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

In-Memory File System Structures

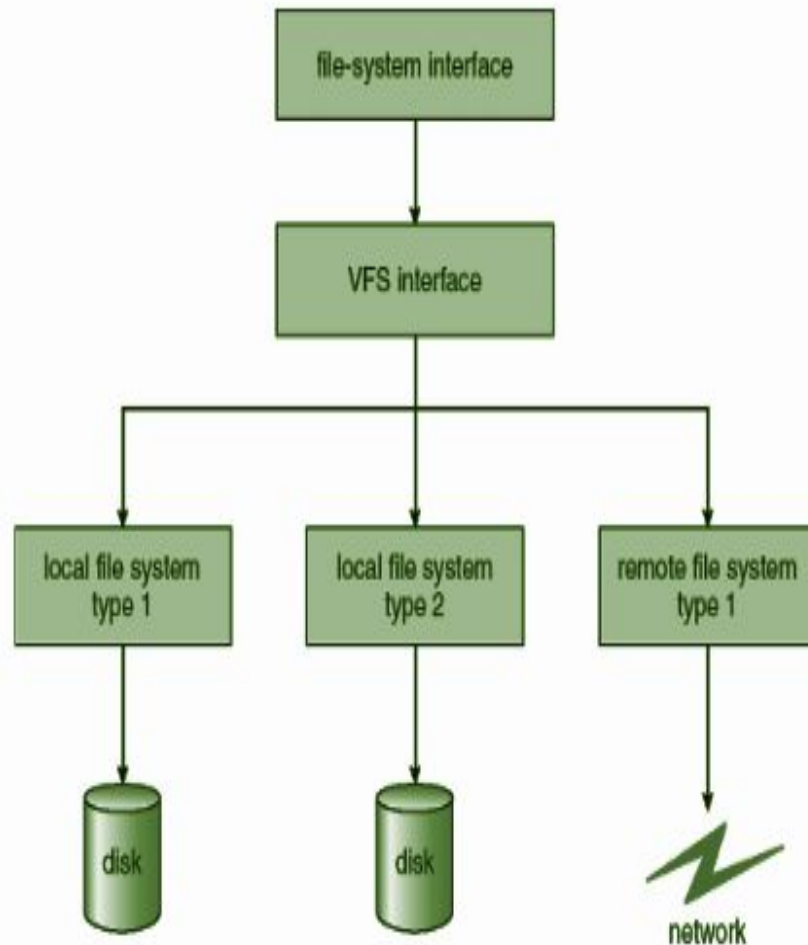


- Mount table storing file system mounts, mount points, file system types
- The above figure illustrates the necessary file system structures provided by the operating systems
- Figure (b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

Partitions and Mounting



- Partition can be a volume containing a file system (“cooked”) or raw – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system or a boot management program for multi-os booting
- Root partition contains the OS, other partitions can hold other Oses, other file systems, or be raw
 - Mounted at boot time
 - Other partitions can mount automatically or manually
- At mount time, file system consistency is checked
 - Is all metadata correct?
 - If not, fix it, try again
 - If yes, add to mount table, allow access



- **Virtual File Systems (VFS)** on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
 - Separates file-system generic operations from implementation details
 - Implementation can be one of many file systems types, or network file system
 - Implements **vnodes** which hold inodes or network file details
 - Then dispatches operation to appropriate file system implementation routines
 - The API is to the VFS interface, rather than any specific type of file system

Virtual File Systems Implementation

- For example, Linux has four object types:
 - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
 - Every object has a pointer to a function table
 - Function table has addresses of routines to implement that function on that object
 - For example:
 - `int open(. . .)` — Open a file
 - `int close(. . .)` — Close an already-open file
 - `ssize_t read(. . .)` — Read from a file
 - `ssize_t write(. . .)` — Write to a file
 - `int mmap(. . .)` — Memory-map a file

- **File-System Structure**
- **File-System Implementation**
- **Virtual File Systems**
- **Virtual File Systems Implementation**

- **Contiguous Allocation**
- **Linked allocation**
- **Indexed Allocation**

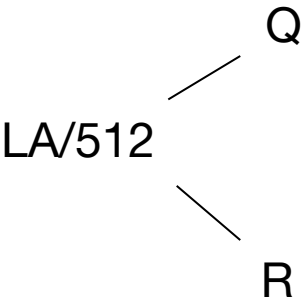
Contiguous Allocation

- An allocation method refers to how disk blocks are allocated for files
- Contiguous allocation – each file occupies set of contiguous blocks
 - Best performance in most cases
 - Simple – only starting location (block #) and length (number of blocks) are required
 - Problems include finding space for file, knowing file size, external fragmentation, need for compaction off-line (downtime) or on-line

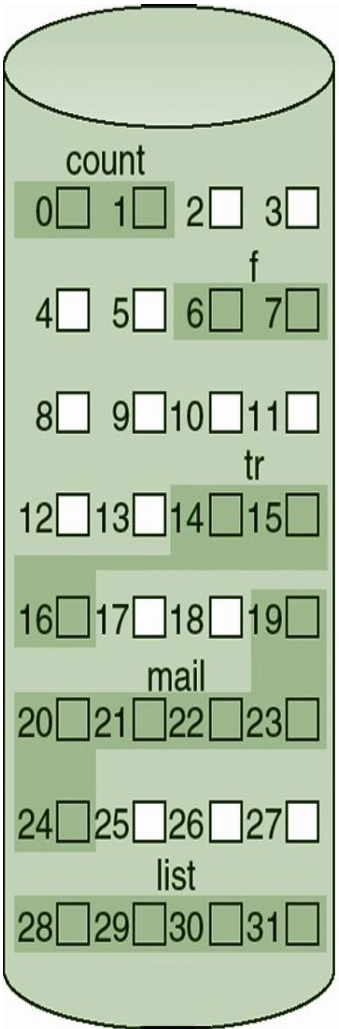
OPERATING SYSTEMS

Contiguous Allocation

- Mapping from logical to physical



- Block to be accessed = Q + starting address
Displacement into block = R

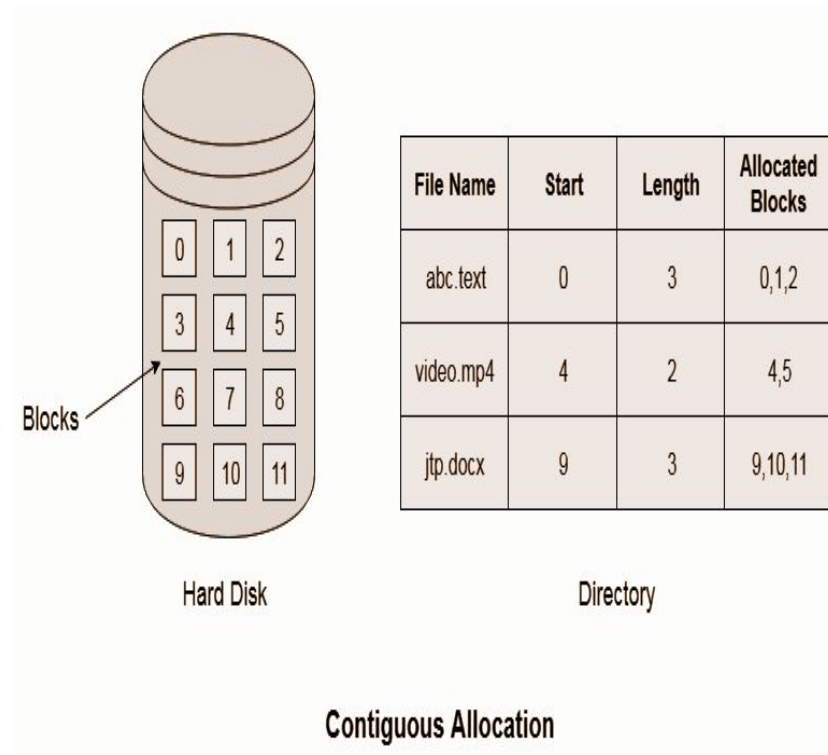


directory

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

Contiguous Allocation

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
 - Extents are allocated for file allocation
 - A file consists of one or more extents



Advantages:

1. In the contiguous allocation, sequential and direct access both are supported.
2. For the direct access, the starting address of the k th block is given and further blocks are obtained by $b+K$,
3. This is very fast and the number of seeks is minimal in the contiguous allocation method.

Disadvantages:

1. Contiguous allocation method suffers internal as well as external fragmentation.
2. In terms of memory utilization, this method is inefficient.
3. It is difficult to increase the file size because it depends on the availability of contiguous memory.

- Linked allocation – each file a linked list of blocks
 - File ends at nil pointer
 - No external fragmentation
 - Each block contains pointer to next block
 - No compaction, external fragmentation
 - Free space management system called when new block needed
 - Improve efficiency by clustering blocks into groups but increases internal fragmentation
 - Reliability can be a problem
 - Locating a block can take many I/Os and disk seeks

- In this scheme, the directory entry contains the pointer of the first block and pointer of the ending block.
- These pointers are not for the users.
- For example, a file of six blocks starts at block 10 and end at the block. Each pointer contains the address of the next block.
- When we create a new file we simply create a new entry with the linked allocation.
- Each directory contains the pointer to the first disk block of the file. when the pointer is nil then it defines the empty file.

Advantages:

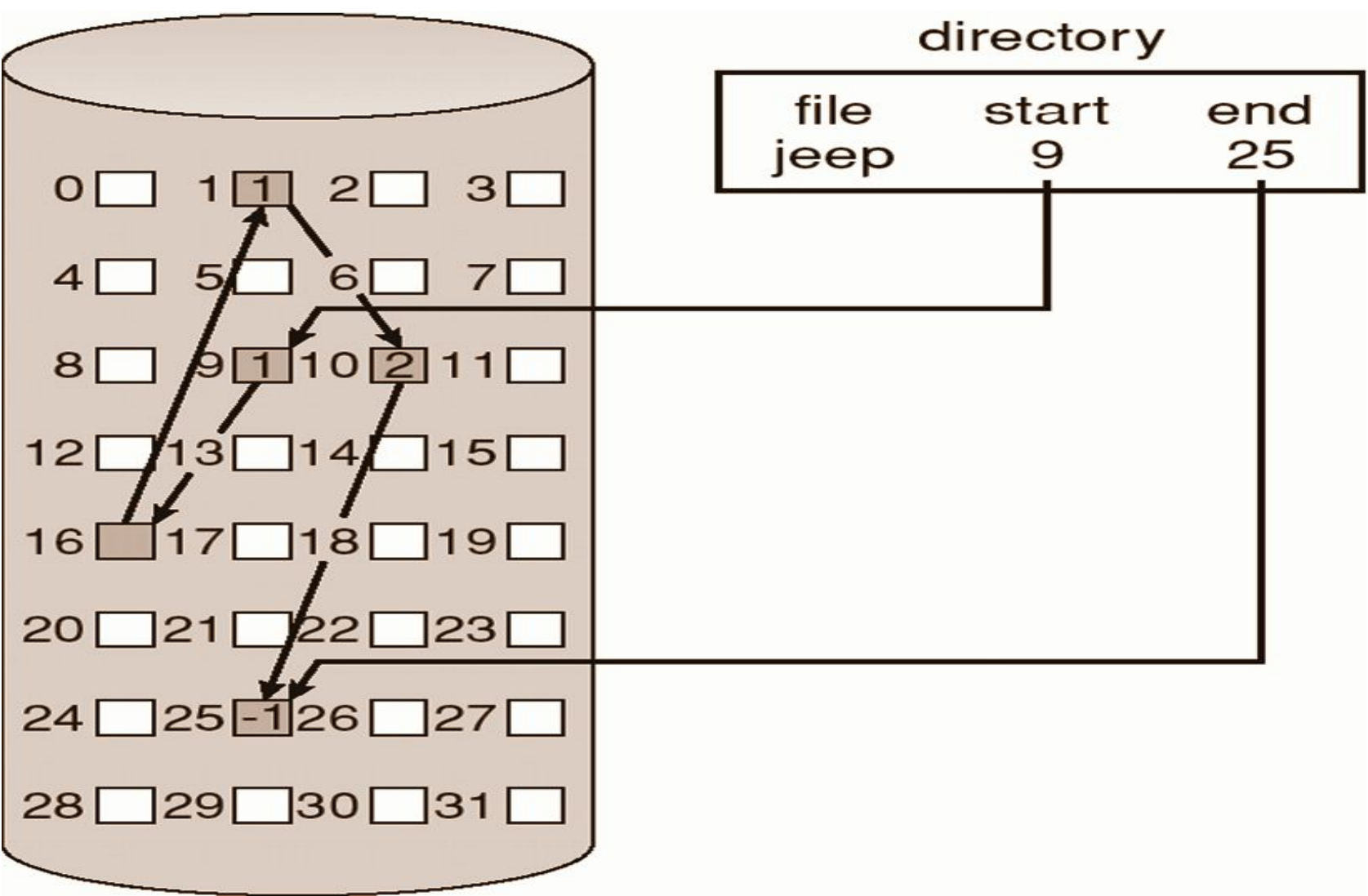
1. In terms of the file size, this scheme is very flexible.
2. We can easily increase or decrease the file size and system does not worry about the contiguous chunks of memory.
3. This method free from external fragmentation this makes it better in terms of memory utilization.

Disadvantages:

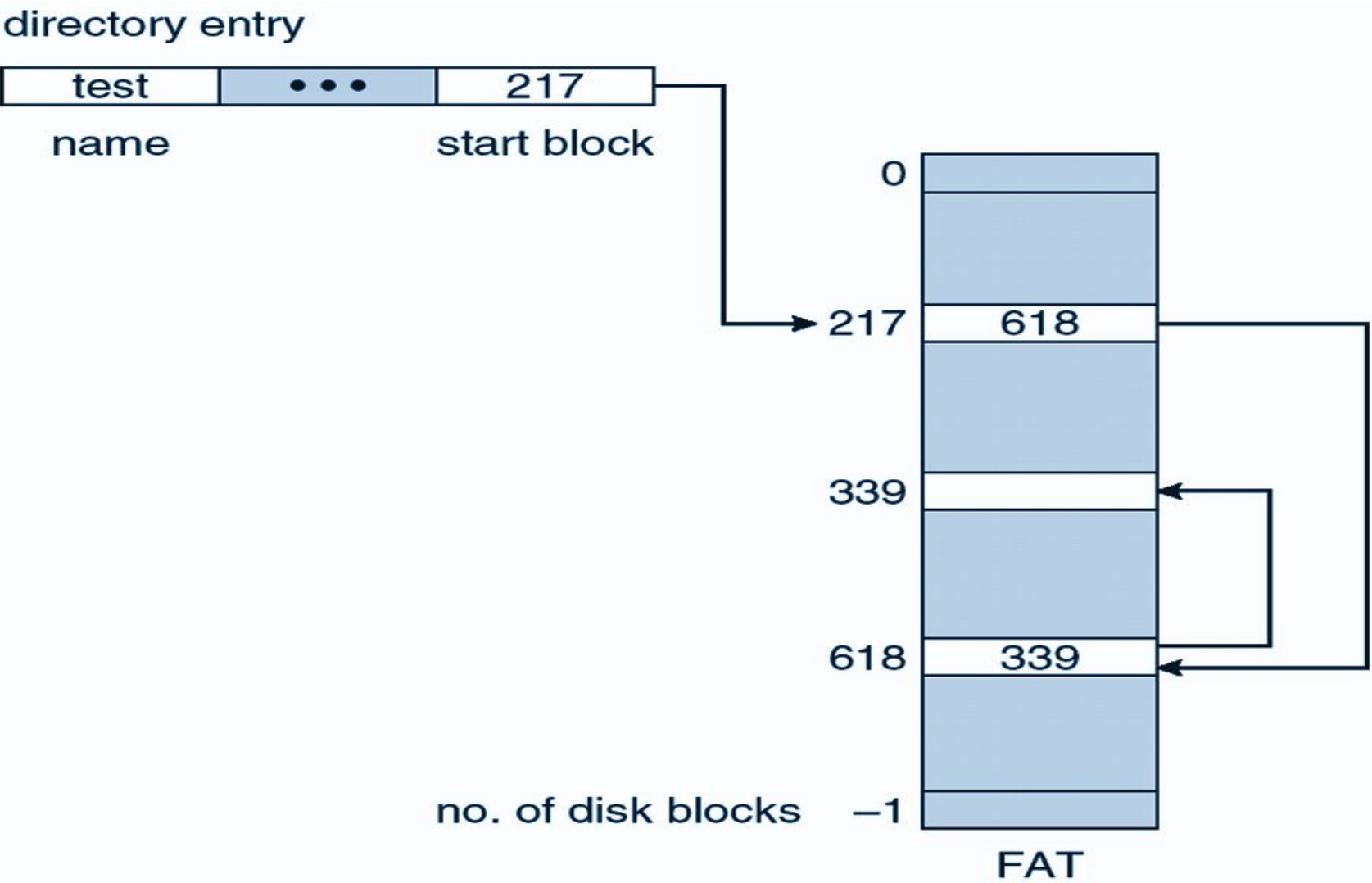
1. In this scheme, there is large no of seeks because the file blocks are randomly distributed on disk.
2. Linked allocation is comparatively slower than contiguous allocation.
3. Random or direct access is not supported by this scheme we cannot access the blocks directly.
4. The pointer is extra overhead on the system due to the linked list.

OPERATING SYSTEMS

Linked Allocation



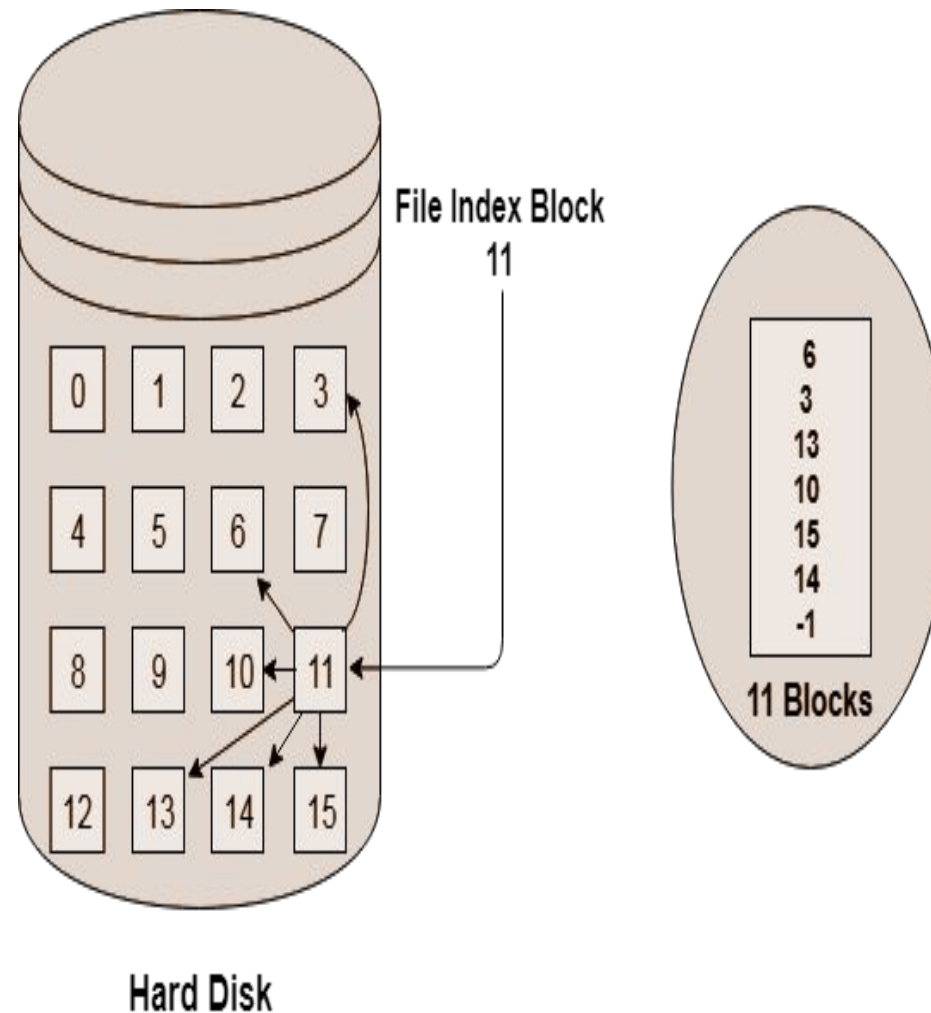
Linked Allocation File Allocation Table



OPERATING SYSTEMS

Indexed Allocation

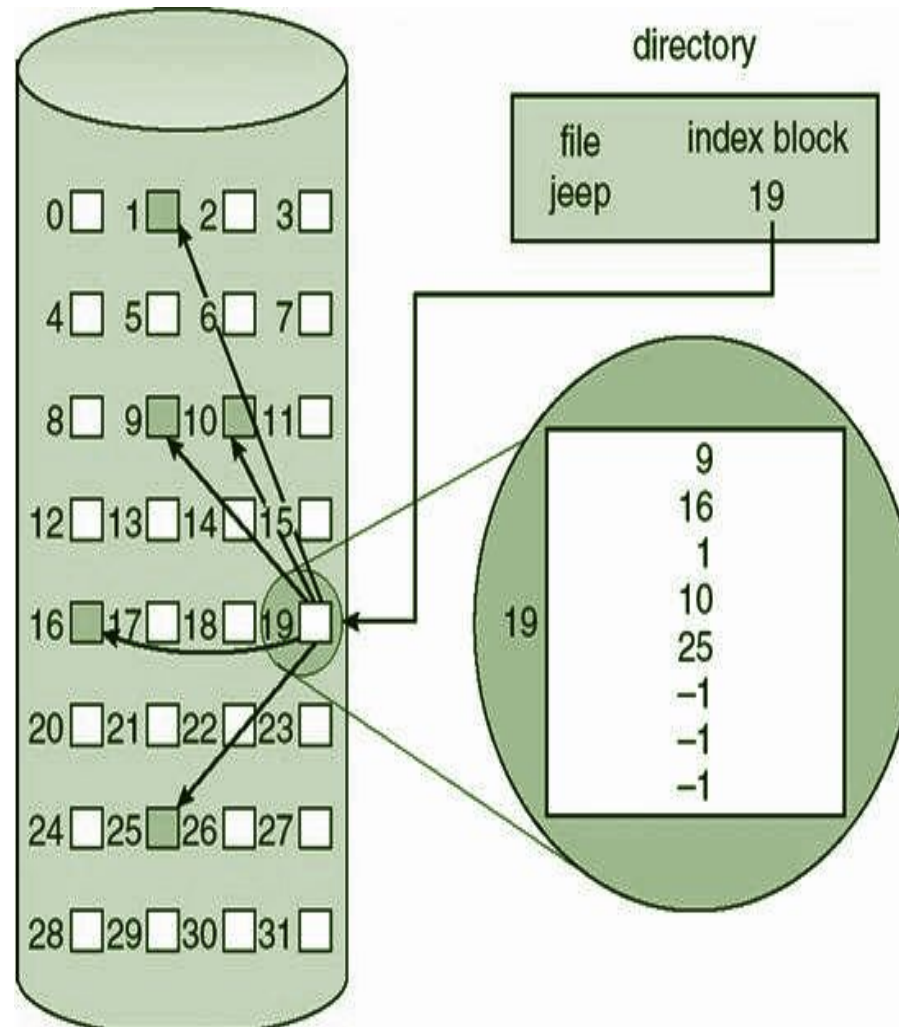
- Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block.
- Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file.
- Directory entry will only contain the index block address.



OPERATING SYSTEMS

Indexed Allocation

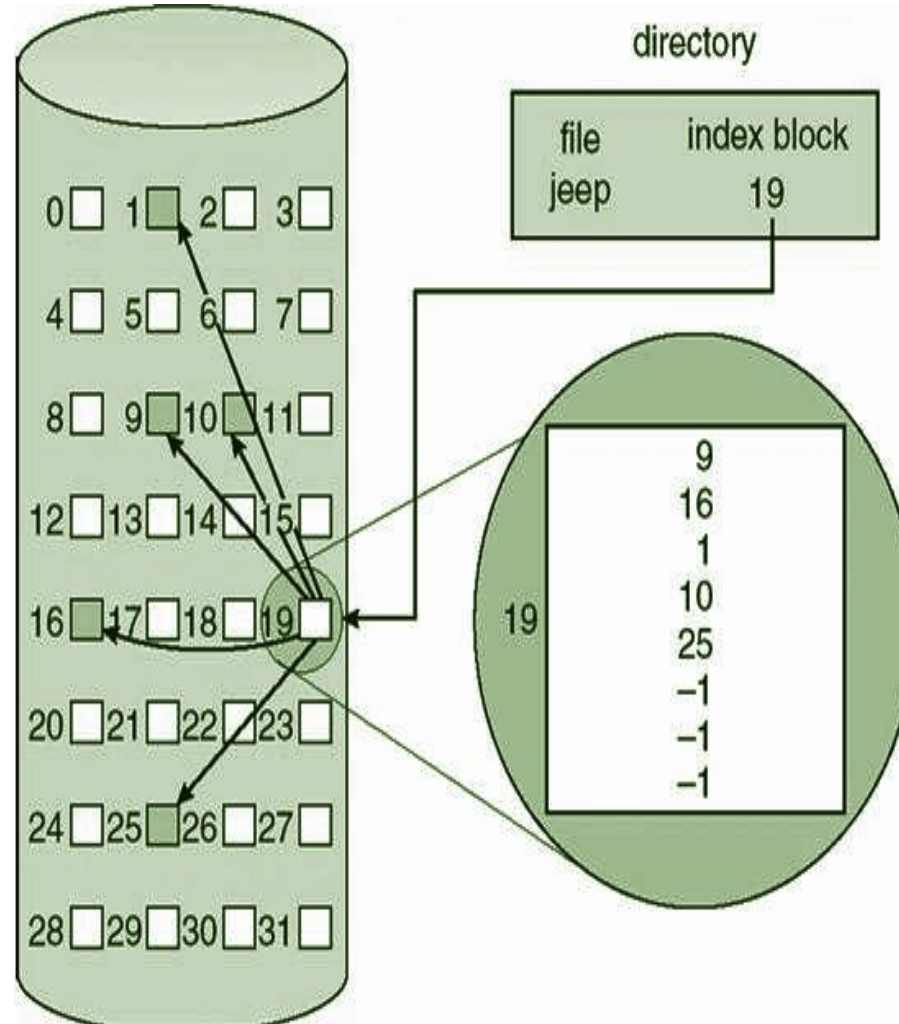
- In Linked Allocation the pointers along with the blocks were scattered across the disk and needed to be retrieved in order by visiting each block for accessing the file.
- In indexed allocation method, all the pointers are gathered together into one location known as Index Block.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.



OPERATING SYSTEMS

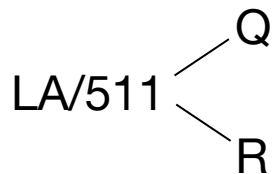
Indexed Allocation

- When a file is created initially, all pointers in the index block are set to null value.
- As new blocks are written, the pointers are modified accordingly.



Indexed Allocation

- Indexed allocation supports direct access and does not suffer from any external fragmentation.
- Indexed allocation suffers from the problem of wasted space.
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table



Q = displacement into index table

R = displacement into block

Advantages

1. Supports direct access
2. A bad data block causes the loss of only that block.

Disadvantages

1. A bad index block could cause the loss of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.
4. More pointer overhead

- Indexed allocation supports direct access and does not suffer from any external fragmentation
- Indexed allocation suffers from the problem of wasted space.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

- For files that are very large, single index block may not be able to hold all the pointers.
- Following mechanisms can be used to resolve this:
 - **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
 - **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.

OPERATING SYSTEMS

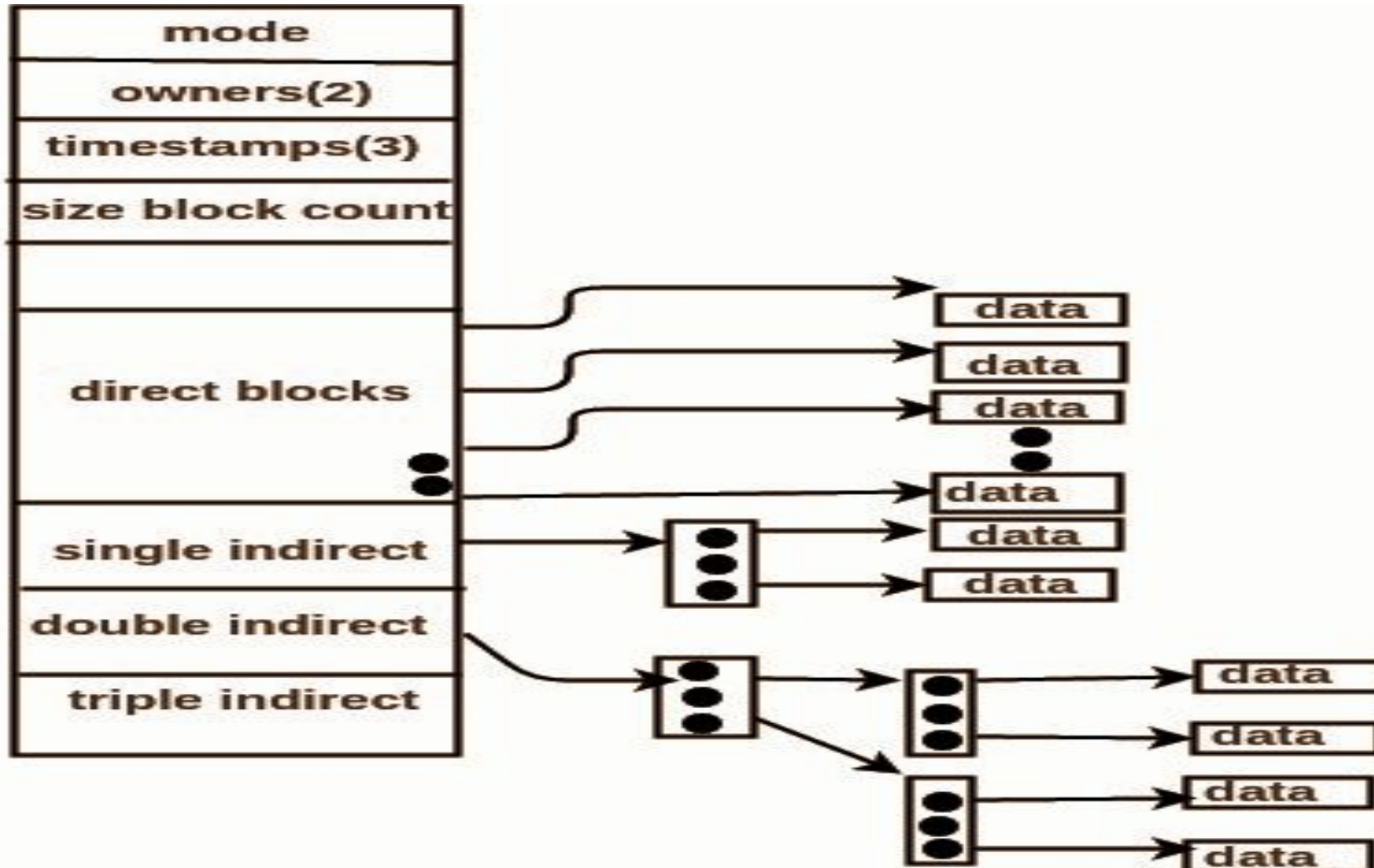
Indexed Allocation

- For files that are very large, single index block may not be able to hold all the pointers.
- Following mechanisms can be used to resolve this:
 - **Combined Scheme:** In this scheme, a special block called the inode (information Node) contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file as shown in the image below.
 - The first few of these pointers in Inode point to the direct blocks i.e the pointers contain the addresses of the disk blocks that contain data of the file.
 - The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect. Single Indirect block is the disk block that does not contain the file data but the disk address of the blocks that contain the file data.
 - Similarly, double indirect blocks do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

OPERATING SYSTEMS

Indexed Allocation

4K bytes per block, 32-bit addresses

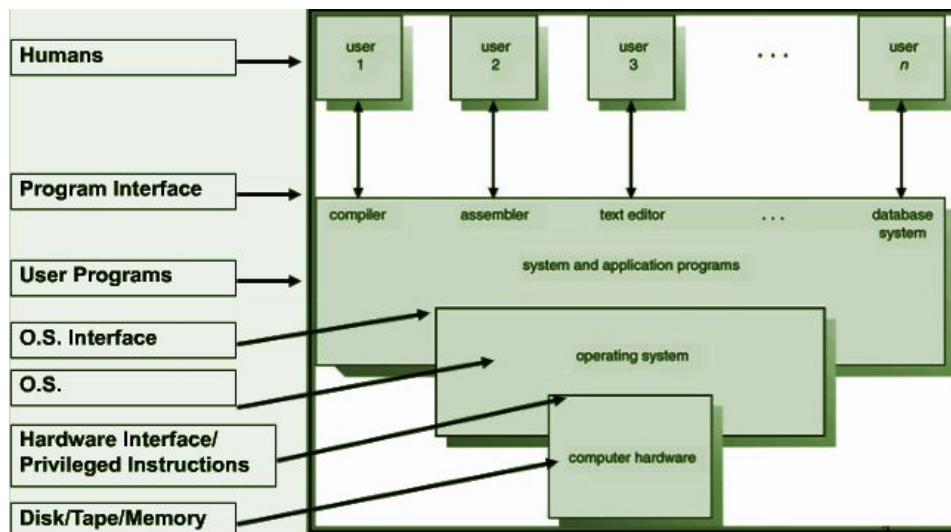
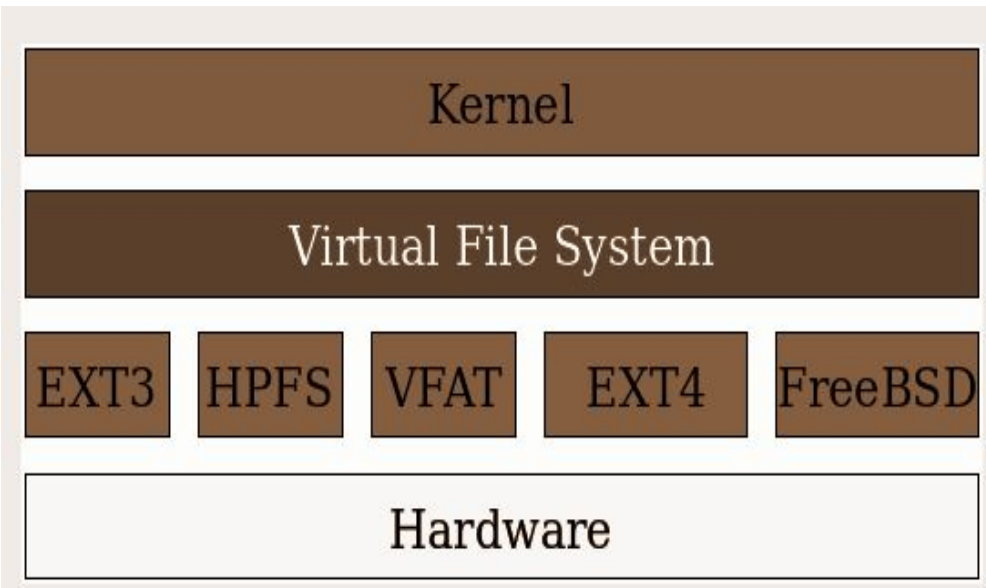


- **Contiguous Allocation**
- **Linked allocation**
- **Indexed Allocation**

- Case Study: Unix / Linux File System

- To the user, Linux file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)

- The entire Linux directory structure starting at the top (/) root directory.
- A specific type of data storage format, such as EXT3, EXT4, BTRFS ("butter F S", "b-tree F S"), XFS, and so on. Linux supports almost 100 types of filesystems, including some very old ones as well as some of the newest.
- Each of these filesystem types uses its own metadata structures to define how the data is stored and accessed.
- A partition or logical volume formatted with a specific type of filesystem that can be mounted on a specified mount point on a Linux filesystem.



- The first part of this two-part implementation is the Linux virtual filesystem.
- This virtual filesystem provides a single set of commands for the kernel, and developers, to access all types of filesystems.
- The virtual filesystem software calls the specific device driver required to interface to the various types of filesystems.
- The filesystem-specific device drivers are the second part of the implementation.
- The device driver interprets the standard set of filesystem commands to ones specific to the type of filesystem on the partition or logical volume.

- The Linux VFS is designed around object-oriented principles and is composed of four components:
 - A set of definitions that define what a file object is allowed to look like
 - The **inode object** structure represent an individual file
 - The **file object** represents an open file
 - The **superblock object** represents an entire file system
 - A **dentry object** represents an individual directory entry

- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects
 - For example for the file object operations include (from struct file_operations in /usr/include/linux/fs.h)
 - int open(. . .) — Open a file
 - ssize_t read(. . .) — Read from a file
 - ssize_t write(. . .) — Write to a file
 - int mmap(. . .) — Memory-map a file

The Linux ext3 File System

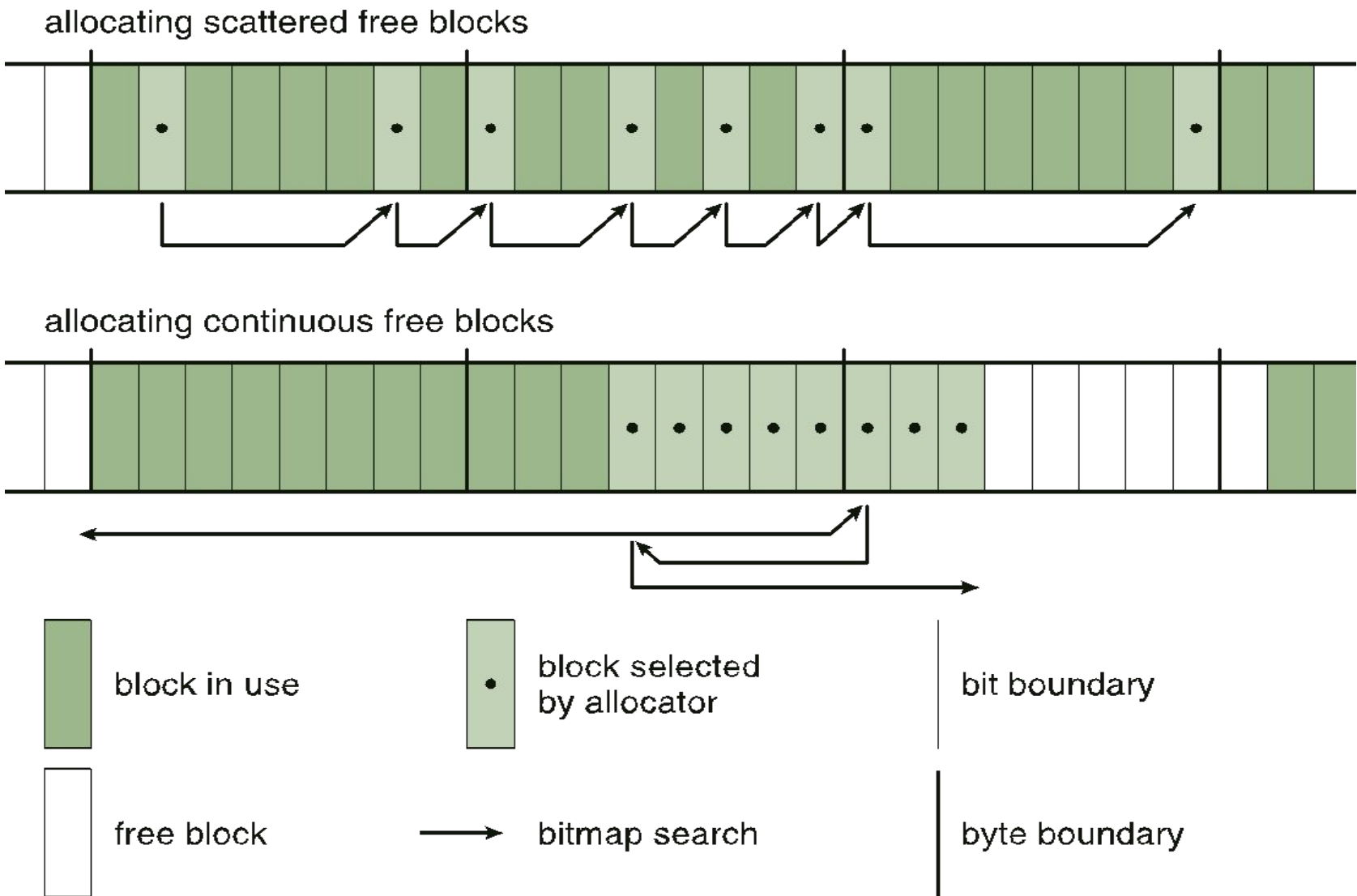


- **ext3** is standard on disk file system for Linux
 - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
 - Supersedes older **extfs**, **ext2** file systems
 - Work underway on ext4 adding features like extents
 - of course, many other file system choices with Linux distros

The Linux ext3 File System

- The main differences between ext2fs and FFS concern their disk allocation policies
 - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
 - ext3 does not use fragments; it performs its allocations in smaller units
 - The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
 - ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a **block group**
 - Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

Ext3 Block-Allocation Policies



The maximum number of blocks for ext3 is 2^{32} . The size of a block can vary, affecting the maximum number of files and the maximum size of the file system

Block size	Maximum file size	Maximum file-system size
1 KiB	16 GiB	4 TiB
2 KiB	256 GiB	8 TiB
4 KiB	2 TiB	16 TiB
8 KiB ^[limits 1]	2 TiB	32 TiB

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions**
 - Once in log file, considered committed
 - Over time, log file transactions replayed over file system to put changes in place
- On system crash, some transactions might be in journal but not yet placed into file system
 - Must be completed once system recovers
 - No other consistency checking is needed after a crash (much faster than older methods)
- Improves write performance on hard disks by turning random I/O into sequential I/O

Three levels of journaling are available in the Linux kernel implementation of ext3: journal, ordered, and writeback.

- **Journal** is the lowest risk mode, writing both data and metadata to the journal before committing it to the filesystem. This ensures consistency of the file being written to, as well as the filesystem as a whole, but can significantly decrease performance.
- **Ordered** is the default mode in most Linux distributions; ordered mode writes metadata to the journal but commits data directly to the filesystem. As the name implies, the order of operations here is rigid: First, metadata is committed to the journal; second, data is written to the filesystem, and only then is the associated metadata in the journal flushed to the filesystem itself. This ensures that, in the event of a crash, the metadata associated with incomplete writes is still in the journal, and the filesystem can sanitize those incomplete writes while rolling back the journal. In ordered mode, a crash may result in corruption of the file or files being actively written to during the crash, but the filesystem itself—and files not actively being written to—are guaranteed safe.
- **Writeback** is the third—and least safe—journaling mode. In writeback mode, like ordered mode, metadata is journaled, but data is not. Unlike ordered mode, metadata and data alike may be written in whatever order makes sense for best performance. This can offer significant increases in performance, but it's much less safe. Although writeback mode still offers a guarantee of safety to the filesystem itself, files that were written to during or before the crash are vulnerable to loss or corruption.

The Linux Proc File System

- The **proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
 - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
 - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer
- The /proc filesystem contains a illusionary filesystem. It does not exist on a disk. Instead, the kernel creates it in memory. It is used to provide information about the system (originally about processes, hence the name).

The Linux Proc File System

`/proc/1`

A directory with information about process number 1. Each process has a directory below `/proc` with the name being its process identification number.

`/proc/cpuinfo`

Information about the processor, such as its type, make, model, and performance.

`/proc/devices`

List of device drivers configured into the currently running kernel.

`/proc/dma`

Shows which DMA channels are being used at the moment.

`/proc/filesystems`

Filesystems configured into the kernel.

`/proc/interrupts`

Shows which interrupts are in use, and how many of each there have been.

The Linux Proc File System

`/proc/ioprots`

Which I/O ports are in use at the moment.

`/proc/kcore`

An image of the physical memory of the system. This is exactly the same size as your physical memory, but does not really take up that much memory; it is generated on the fly as programs access it. (Remember: unless you copy it elsewhere, nothing under `/proc` takes up any disk space at all.)

`/proc/kmsg`

Messages output by the kernel. These are also routed to **syslog**.

`/proc/ksyms`

Symbol table for the kernel.

`/proc/loadavg`

The 'load average' of the system; three meaningless indicators of how much work the system has to do at the moment.

`/proc/meminfo`

Information about memory usage, both physical and swap.

`/proc/modules`

Which kernel modules are loaded at the moment.

`/proc/net`

Status information about network protocols.

The Linux Proc File System

`/proc/self`

A symbolic link to the process directory of the program that is looking at `/proc`. When two processes look at `/proc`, they get different links. This is mainly a convenience to make it easier for programs to get at their process directory.

`/proc/stat`

Various statistics about the system, such as the number of page faults since the system was booted.

`/proc/uptime`

The time the system has been up.

`/proc/version`

The kernel version.

Unix / Linux File System - Additional Input

/ (root filesystem)	The root filesystem is the top-level directory of the filesystem. It must contain all of the files required to boot the Linux system before other filesystems are mounted. It must include all of the required executables and libraries required to boot the remaining filesystems. After the system is booted, all other filesystems are mounted on standard, well-defined mount points as subdirectories of the root filesystem.
/bin	The /bin directory contains user executable files.
/boot	Contains the static bootloader and kernel executable and configuration files required to boot a Linux computer.
/dev	This directory contains the device files for every hardware device attached to the system. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices.
/etc	Contains the local system configuration files for the host computer.
/home	Home directory storage for user files. Each user has a subdirectory in /home.
/lib	Contains shared library files that are required to boot the system.
/media	A place to mount external removable media devices such as USB thumb drives that may be connected to the host.

Unix / Linux File System - Additional Input

/mnt	A temporary mount point for regular file systems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
/opt	Optional files such as vendor supplied application programs should be located here.
/root	This is not the root (/) filesystem. It is the home directory for the root user.
/sbin	System binary files. These are executables used for system administration.
/tmp	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
/usr	These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.
/var	Variable data files are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more.

Snap is a software packaging and deployment system developed by Canonical for the operating systems that use the Linux kernel.

Unix / Linux File System - Additional Input

```
drwxr-xr-x  2 root root  4096 Oct  8 08:51 bin
drwxr-xr-x  3 root root  4096 Oct  9 08:23 boot
drwxrwxr-x  2 root root  4096 Mar  8  2018 cdrom
drwxr-xr-x 21 root root 4620 Oct 29 08:28 dev
drwxr-xr-x 148 root root 12288 Oct 28 12:17 etc
drwxr-xr-x  3 root root  4096 Mar  8  2018 home
lrwxrwxrwx  1 root root   34 Oct  8 08:52 initrd.img -> boot/initrd.img-4.15.0-119-generic
lrwxrwxrwx  1 root root   34 Oct  8 08:52 initrd.img.old -> boot/initrd.img-4.15.0-117-generic
drwxr-xr-x 23 root root  4096 Oct  8 08:50 lib
drwxr-xr-x  2 root root  4096 Oct  8 08:50 lib64
drwx----- 2 root root 16384 Mar  8  2018 lost+found
drwxr-xr-x  3 root root  4096 Mar 10  2018 media
drwxr-xr-x  2 root root  4096 Apr 21  2016 mnt
drwxr-xr-x  4 root root  4096 Apr 20  2018 opt
dr-xr-xr-x 294 root root    0 Oct 29 08:27 proc
drwx-----  7 root root  4096 Sep 16  2019 root
drwxr-xr-x 29 root root   860 Oct 29 08:33 run
drwxr-xr-x  2 root root 12288 Oct  8 08:50/sbin
drwxr-xr-x 18 root root  4096 Sep 21 13:31 snap
drwxr-xr-x  2 root root  4096 Apr 21  2016 srv
dr-xr-xr-x 13 root root    0 Oct 29 09:58 sys
drwxrwxrwt 23 root root 94208 Oct 29 10:01 tmp
drwxr-xr-x 12 root root  4096 Aug  3  2019 usr
drwxr-xr-x 14 root root  4096 Apr 21  2016 var
lrwxrwxrwx  1 root root   31 Oct  8 08:52 vmlinuz -> boot/vmlinuz-4.15.0-119-generic
lrwxrwxrwx  1 root root   31 Oct  8 08:52 vmlinuz.old -> boot/vmlinuz-4.15.0-117-generic
```

- - - Regular file
- b - Block special file
- c - Character special file
- d - Directory
- l - Symbolic link
- n - Network file
- p - FIFO
- s - Socket

- **Case Study: Unix /
Linux File System**



THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com