OPERATING SYSTEMS

Storage Management - 6

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

OPERATING SYSTEMS

Storage Management - 6:
File Concept, Structure, Access Method, Directory and Directory Structure

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

OPERATING SYSTEMS

Course Syllabus - Unit 3

Unit 4: Storage Management

Mass-Storage Structur - 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

OPERATING SYSTEMS

Course Outline

37	Mass-Storage Structure: Mass-Storage overview	12.1	82.1
38	Disk Scheduling – FCFS, SSTF, SCAN, C-SCAN, LOOK	12.4	
39	Swap-Space Management, RAID Structure	12.6,12.7	
40	File Concept, File Structure, Access Methods	10.1-10.2	
41	Directory and Disk Structure	10.3	
42	File-System Mounting, File Sharing, Protecting	10.4-10.6	
43	Implementing File-Systems: File control Block (inode), partitions & mounting	11.1,11.2	
44	Disk Space Allocation methods: Contiguous, Linked, Indexed	11.4	
45	Case Study: Unix/Linux File systems	11.8	
46	NFS	16.7	

Topic Outline

- File Concept
- Access Methods
- Directory and Directory 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

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 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 Operations

- 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

File Operations

- **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.

File Operations

- **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

Open Files

- 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

Open File Locking

- 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

File Structure

- 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

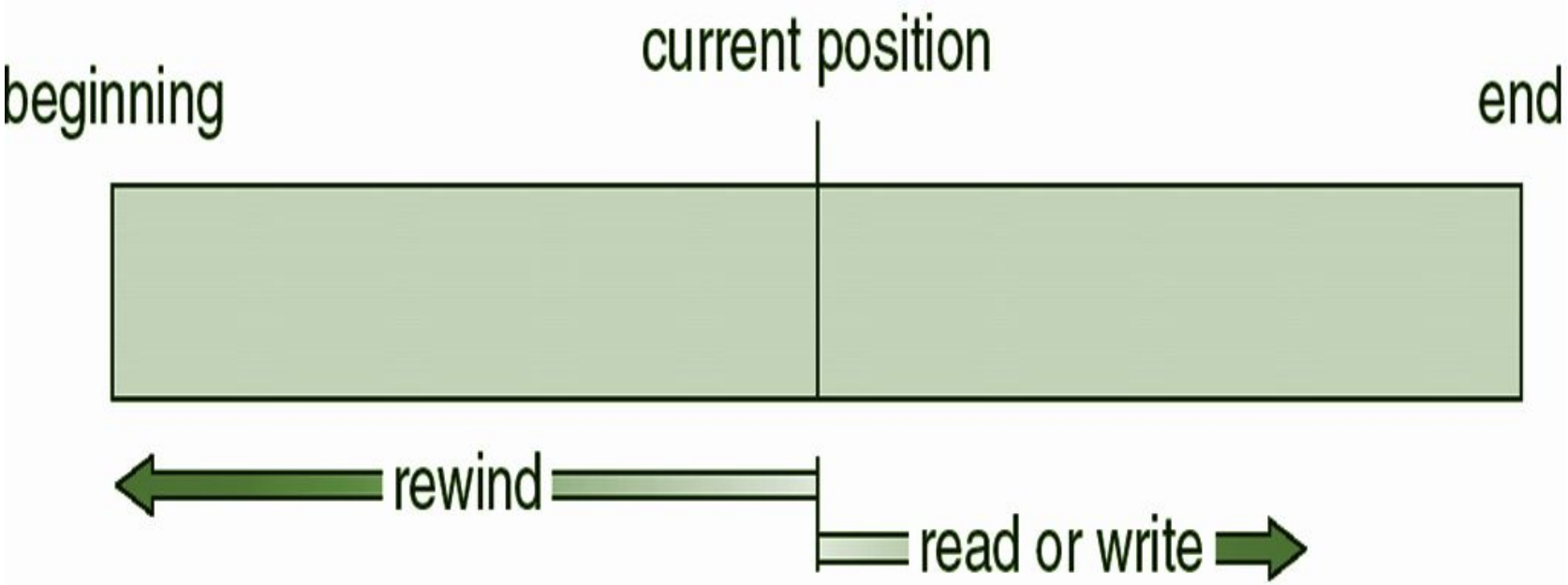
Access Methods

- **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

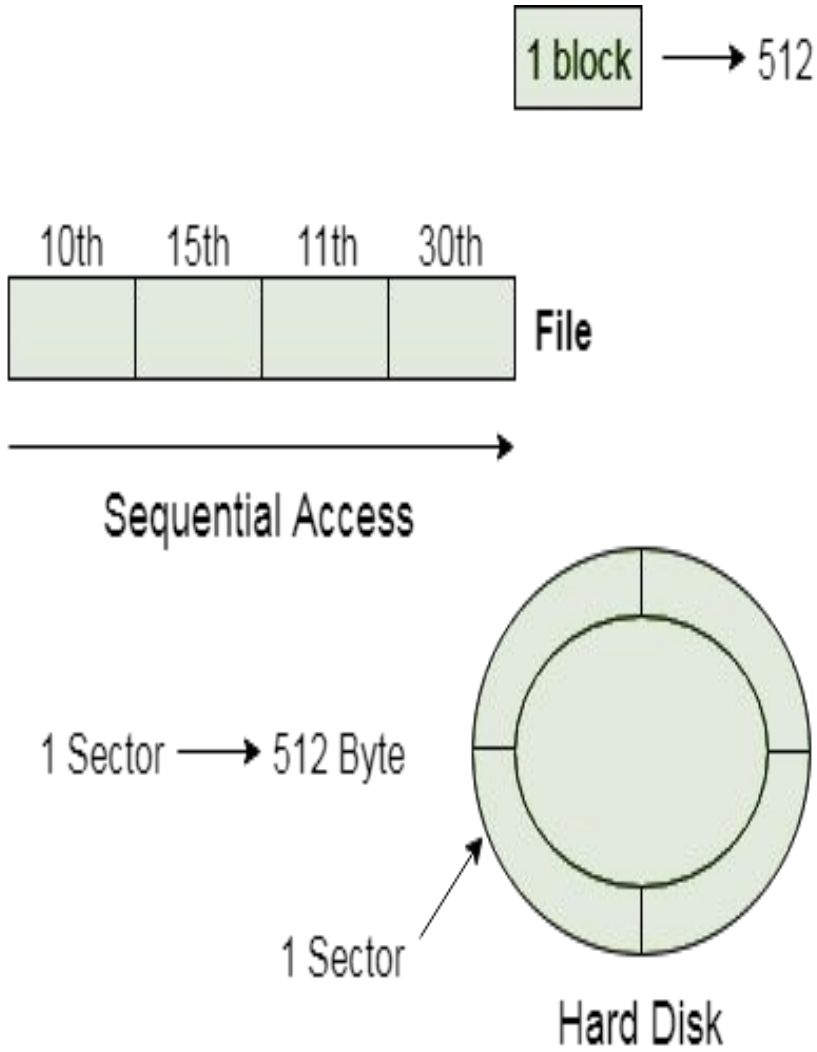
Sequential Access - File



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;

Sequential Access - File - Additional Input

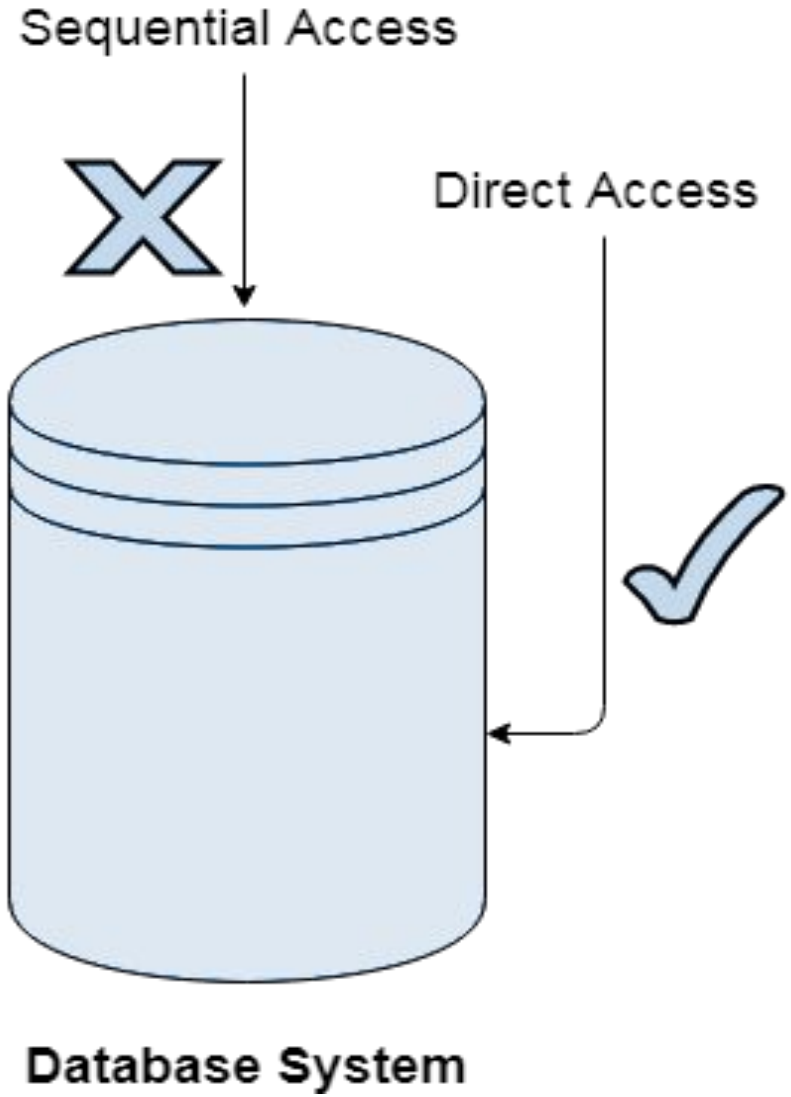


- Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.
- In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the 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.

Other Access Methods

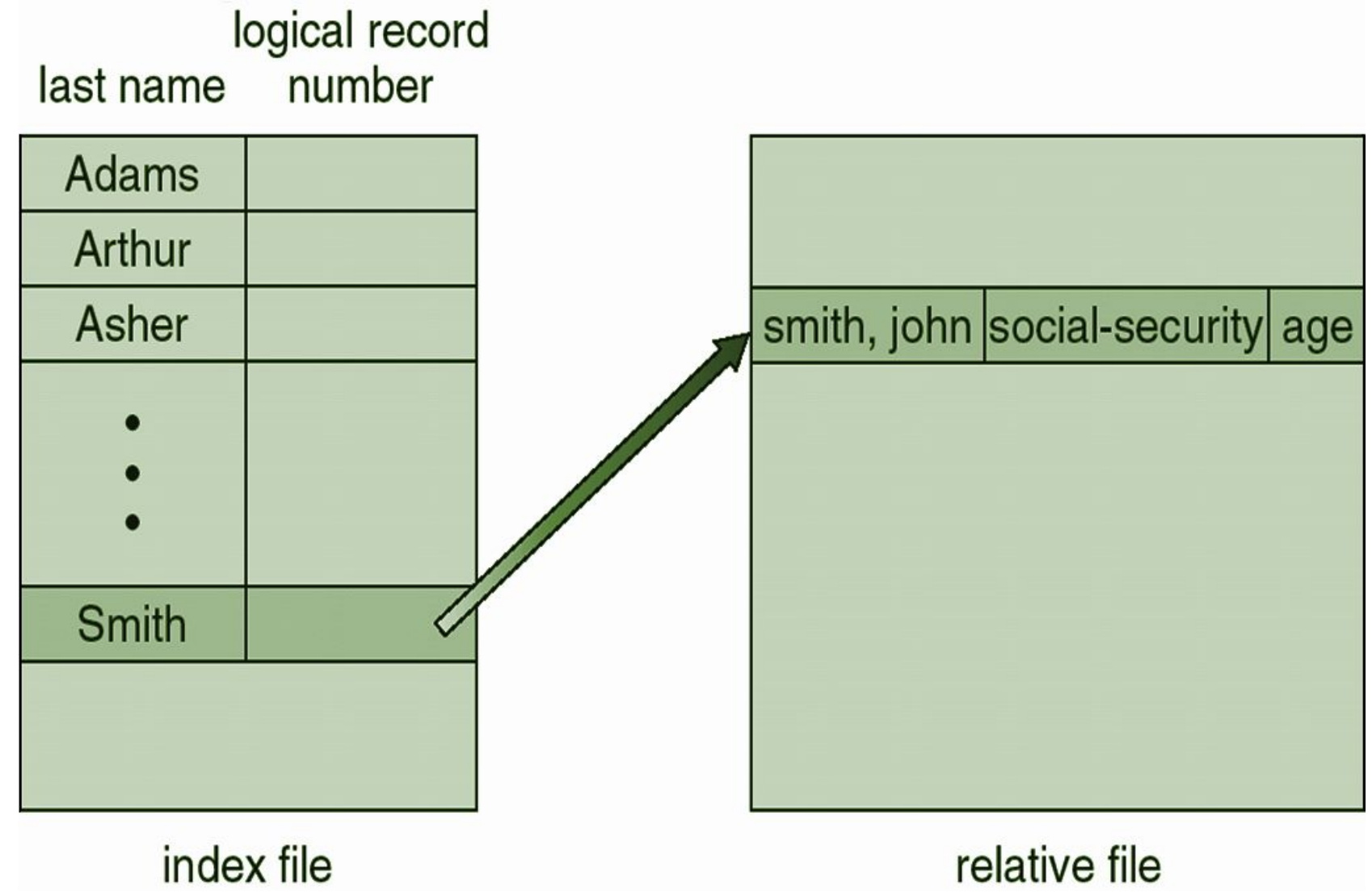
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed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

Direct Access - File - Additional Input



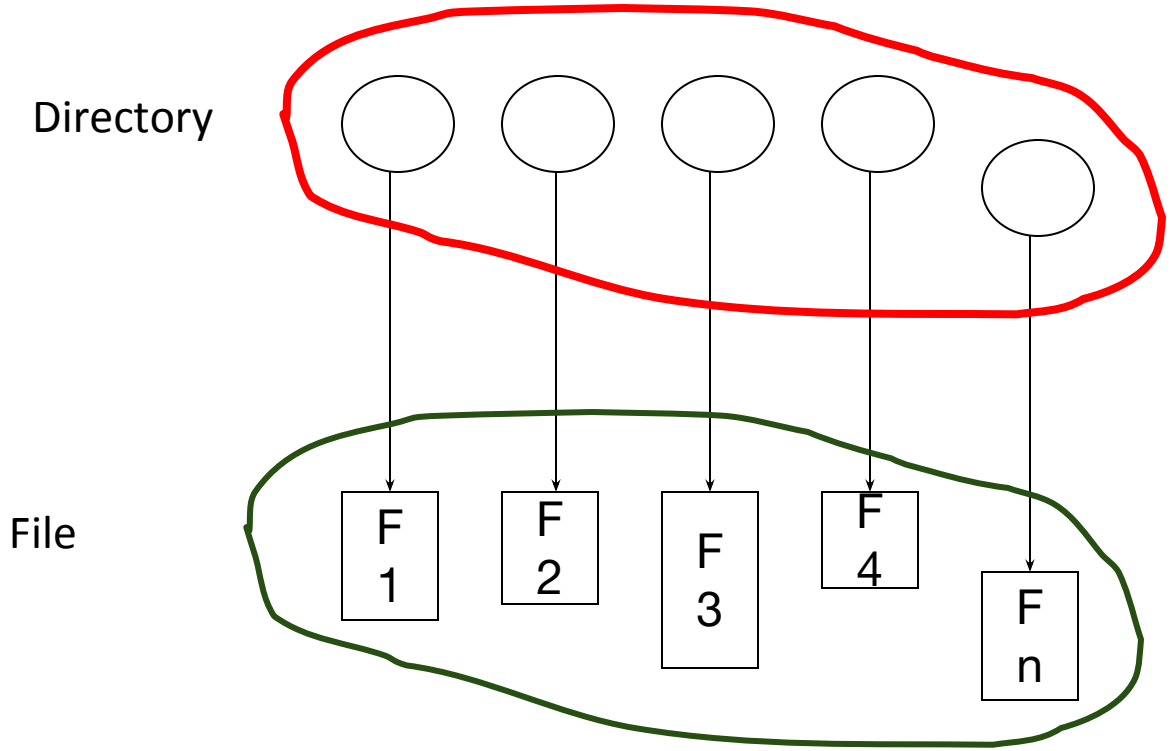
- The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.
- Suppose every block of the storage stores 4 records and we know that the record we needed 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

Example of Index and Relative Files



Directory Structure

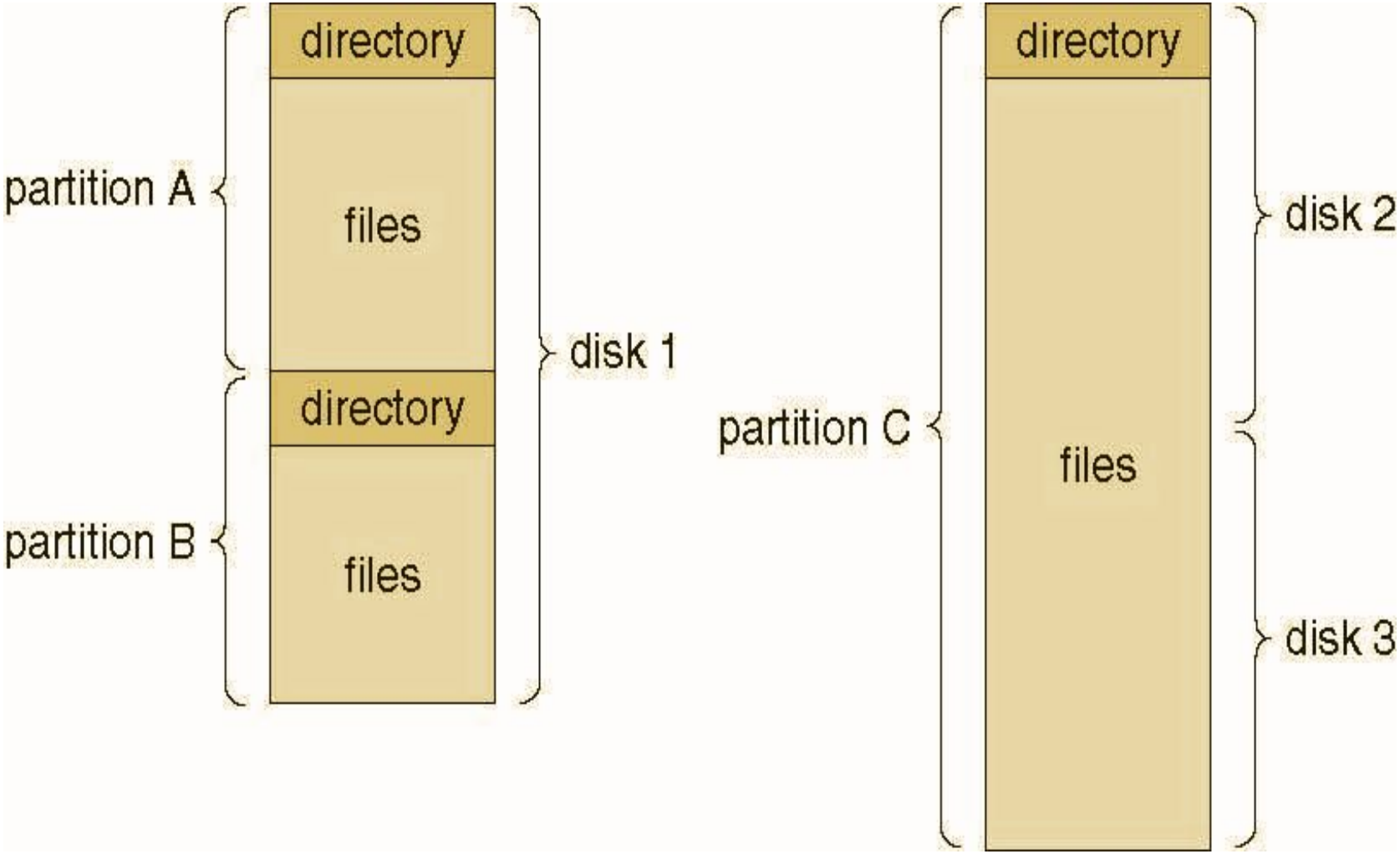
- A collection of nodes containing information about all files
- Both the directory structure and the files reside on disk



Disk Structure

- 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 may 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

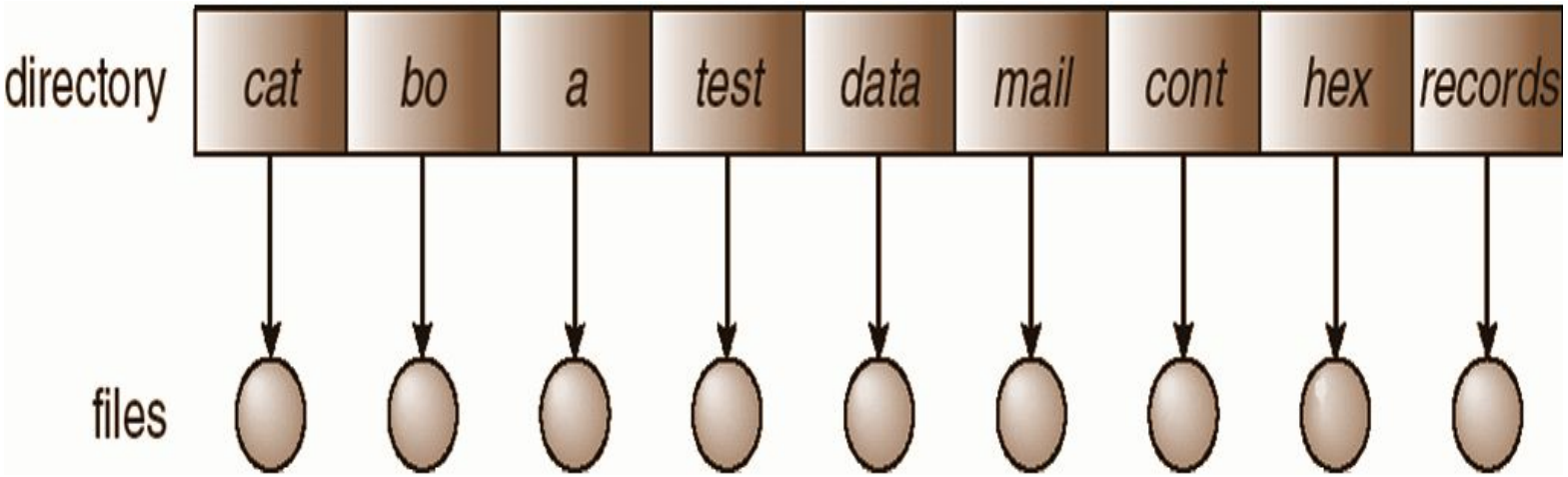
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

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

- 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.

Single-Level Directory

Advantages:

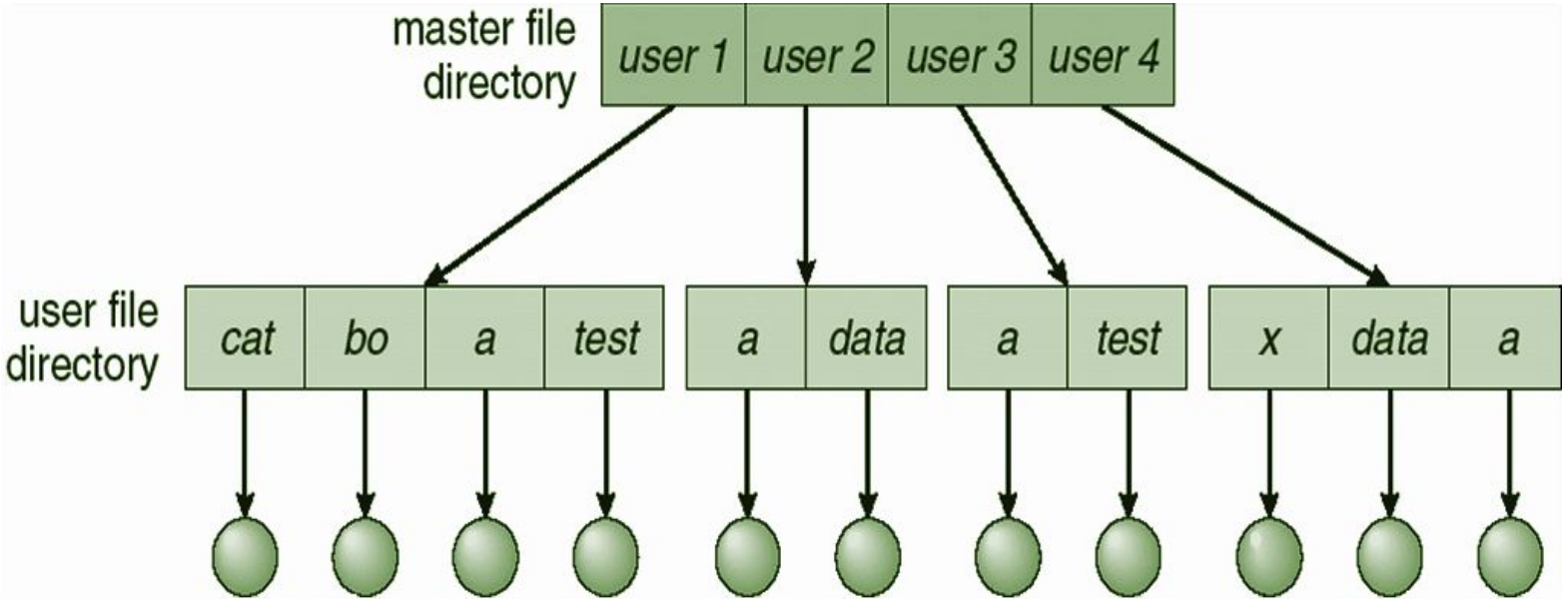
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

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 there own *user files directory (UFD)*. The UFDs has similar structures, but each lists only the files of a single user. system's *master file directory (MFD)* is searches whenever a new user id=s 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 become more easy due to path name and user-grouping.

Disadvantages:

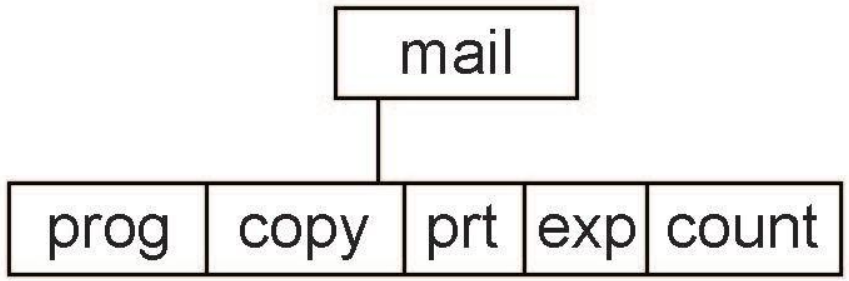
- A user is not allowed to share files with other users.
- Still it not very scalable, two files of the same type cannot be grouped together in the same user.

Tree Structured Directory

- 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” ⇒ deleting the entire subtree rooted by “mail”

Tree Structured Directory

Advantages:

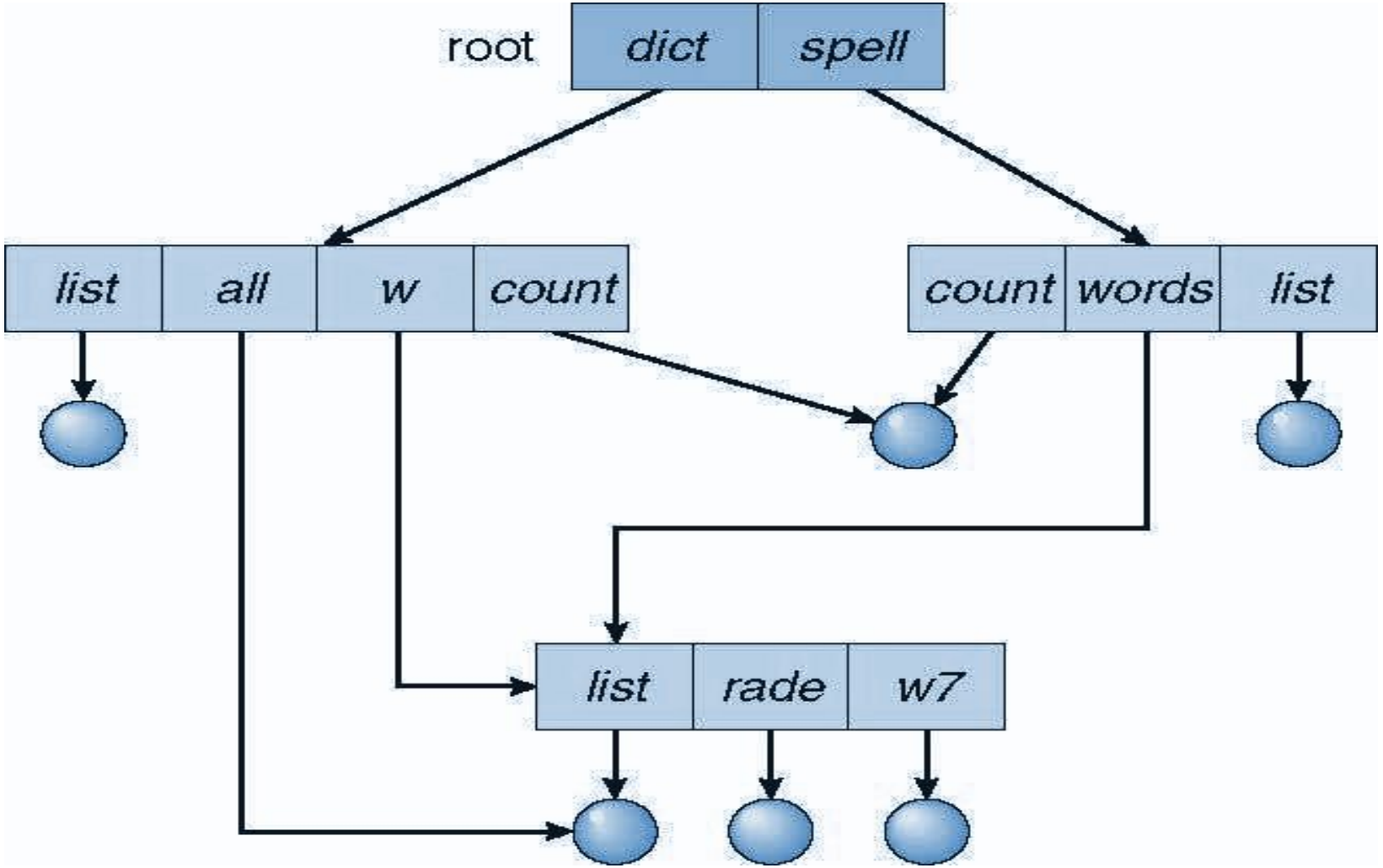
- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

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



Acyclic Graph Directory

- 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 Directory

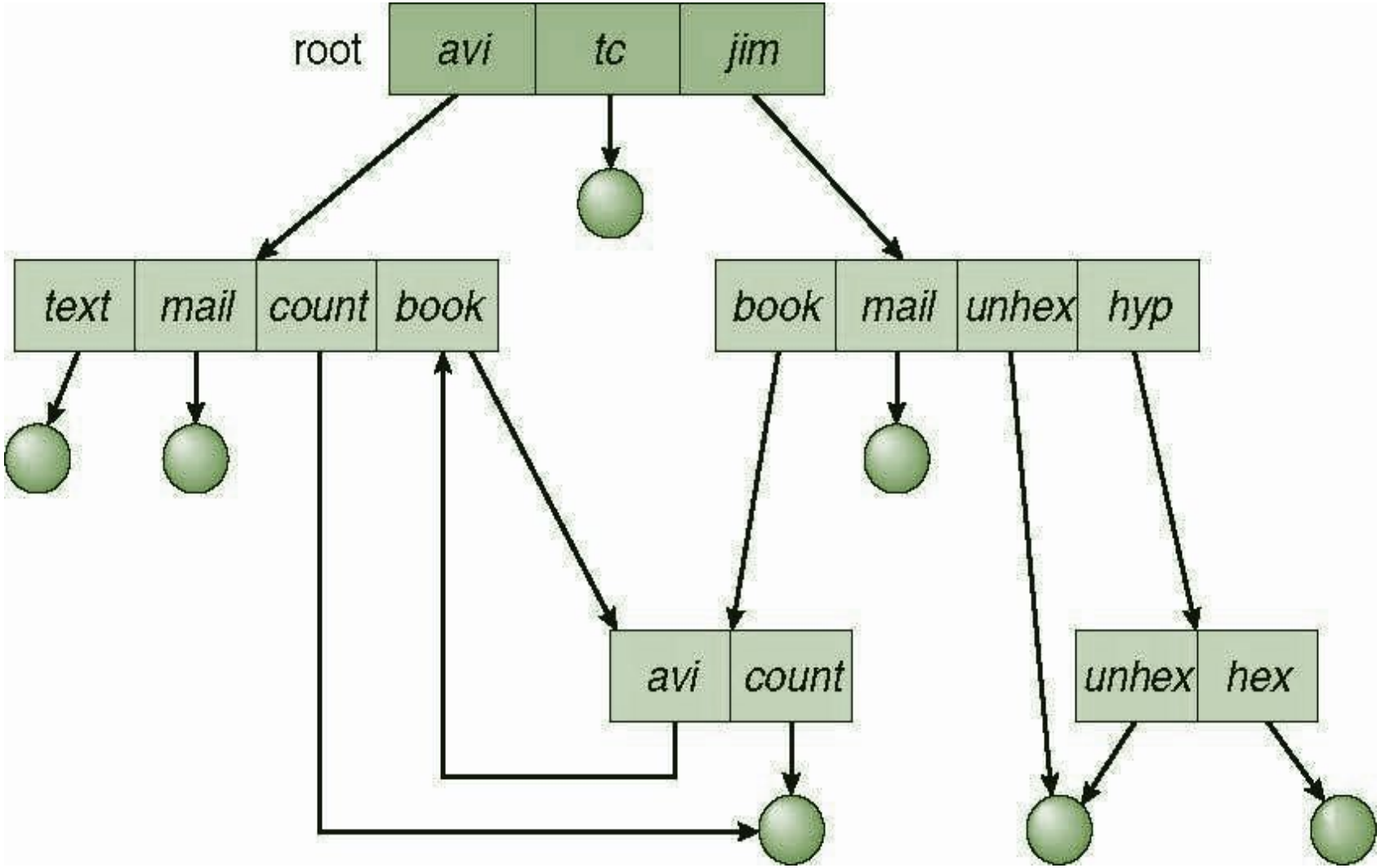
Advantages:

- 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 left with a dangling pointer.
- In case of hardlink, to delete a file we have to delete all the reference associated with i

General Graph Directory



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

General Graph Directory

Advantages:

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

Disadvantages:

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

Topics Uncovered in this Session

- File Concept
- Access Methods
- Directory and Directory Structure

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