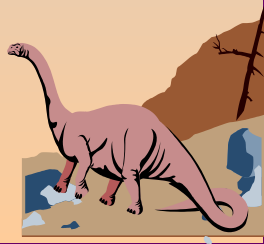
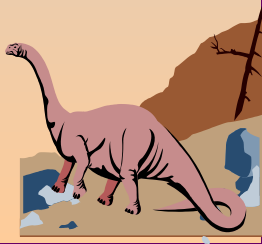
Chapter 10: File-System Interface

- File Concept
- Access Methods
- Directory Structure
- File System Mounting
- File Sharing
- Protection



File Concept

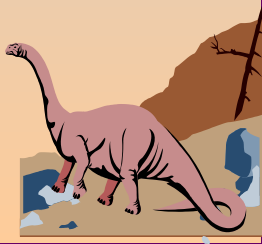
- A file is a named collection of related information that is recorded on secondary storage.
- From user's perspective, a file is a smallest allotment of logical secondary memory.
- Data cannot be written to secondary memory unless they are written in a file.
- In general, file is a sequence of bytes,bits, lines or records, the meaning of which is defined by the file's creator and user.
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program



File Structure

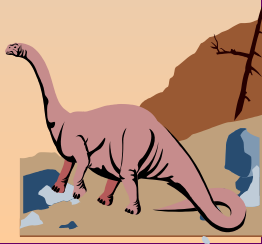
The structure of the file is defined by the type of the file.

- **Text File** – A sequence of characters organized into lines.
- **Source File** – Sequence of sub-routine and functions, each of which is organized as declarations followed by executable statements.
- **Object File** – Sequence of bytes organized into blocks understandable by system's linker.
- **Executable File** – Series of code sections that the loader can bring into memory and execute.



File Attributes

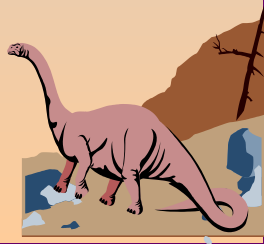
- Once a file is created, it becomes independent of the process, user and even the system that created it.
- File attributes:
  - **Name** – Only information kept in human-readable form.
  - **Identifier** – Unique tag,usually number,uniquely identifies a file.
  - **Type** – needed for systems that 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, usage monitoring, creation, last modification, last use.
- This information about files are kept in the directory structure, which again resides on secondary storage.
- Directory entry contains the file name, and unique identifier.This identifier in turn locates the other file attributes.



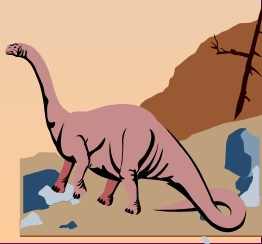
File Operations

Operating System provides system calls for the following file operations:

- Create File
- Write File
- Read File
- Reposition within File
- Delete File
- Truncate File
- Append File
- Rename File
- Open File
- Close File



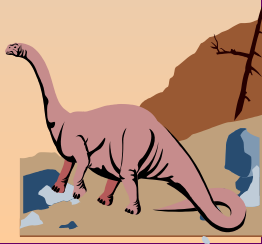
- **Creating a File**
  - Space in the file system must be found for the file.
  - An entry for the file must be made in the directory.
- **Writing a file**
  - Make a system call specifying both the name of the file and the information to be written to the file.
  - System must maintain a write pointer, to signify where next write must be performed.
  - Write pointer must be updated every time a write occurs.
- **Reading a File**
  - Make a system call specifying both the name of the file and the memory location where the read data should be put.
  - System must maintain a read pointer, to signify where next read must be performed from.
  - Read pointer must be updated every time a read occurs.



- Some systems keep **per-process current-file-position-pointer,** instead of a separate read and write pointers.

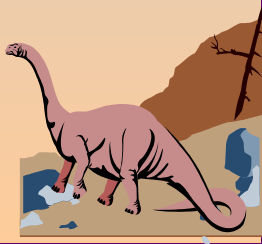
  Both read and write operations use the same pointer.
- **Repositioning within a file**
  - Directory entry is searched and file pointer is set to the given value.
  - No I/O is performed.
  - Also known as **file seek**.
- **Deleting a file**
  - Search the directory entry for the required file and de-allocate all the space allocated.
  - Erase the directory entry.
- **Truncating a File**
  - All attributes of the file remain unchanged, except size of the file, which is set to zero. And space is de-allocated.

- These primitive system calls are combined to perform other file operations. For eg, if user wants to copy file f1 to another,f2
  - Create f2
  - Read f1
  - Write f2
- We may want to have operations using which a user is allowed to determine size of the file, set file attributes.
- **Open File Table**
  - Instead of searching the directory structure every time a file operation is made, we use open system call.
  - When open system call is made, the file's entry is made in open-file table, which contains information about all open files(directory entry)
  - When close call is made, the entry is removed from the table.



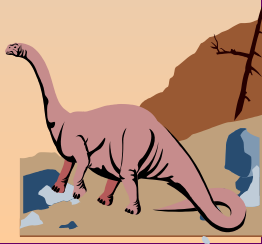
Open call in multi-user system(Unix)

- Multiple users may try to access the same file at the same time. Maintaining one file pointer is not enough.
- OS maintains two internal tables:
  - Per-process open file table
  - System wide open file table
- Per-process table tracks all files that a process has open. This table stores info required for use of the file, file pointer, access rights.
- Each entry in the per-process table in turn points to a location in the system wide table. System wide table keeps process independent info, file location, file size.



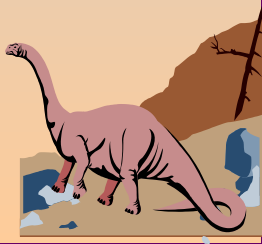
Open call in multi-user system(Unix) (contd.)

- System wide table also keeps **open count** with each file, indicating the number of processes accessing that file.
- Each open call increments this value and close call decrements this value.
- When the counter value becomes zero, the file space is de-allocated.
- Information associated with open file:
  - File pointer
  - File open count
  - Disk location of the file
  - Access rights



File Types

- If OS recognizes type of the file, then it can operate in reasonable way, eg- OS wont allow executing a word file.
- Common technique to implement file types is to have file extensions indicating the type of the file.
- Because these file types are supported by the operating system, they are considered as “hints” to applications that operate on them.
- Another possibility is to have a creator attribute with each file. This attribute is set by the OS when the file is created.
- This attribute helps to open the application in the correct application when we double click on it.

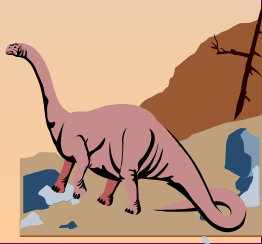


File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	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	binary file containing audio or A/V information

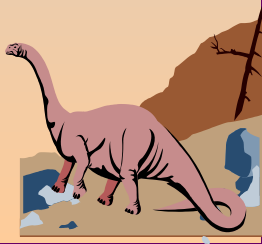
File Types (contd.)

- UNIX does not provide all these features.
- UNIX supports **magic number**, stored at the beginning of the file to indicate roughly the type of the file.
- Not all files have magic number.
- Unix allow file name extension hints, but these extensions are not enforced of implemented by the OS.



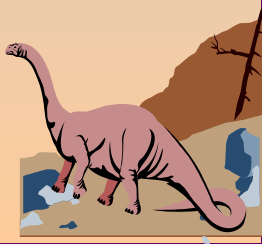
File Structure

- Disadvantage of OS supporting multiple file structures- The size of the OS is very large coz it should contain handling code for all file structures.
- UNIX supports no file structure, it considers every file to be a sequence of 8-bit byte. No interpretation of these bits is made by the OS.
- This scheme provides maximum flexibility.
- Each application should have the code to interpret the input file in the required structure.



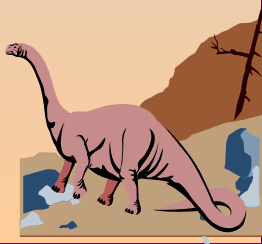
Internal File Structure

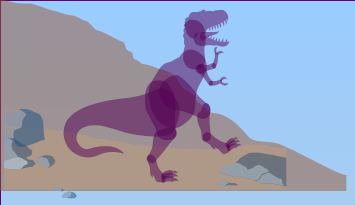
- Logical Record – Record size of the user file.
- Physical Block Size – Disk I/O is performed in the units of disk block size.
- In most cases, logical record size is not the same as physical block size. We then have to pack multiple logical records into one physical block. This is known as **packing.**
- Packing technique – How many logical records are packed into one physical block.
- Packing is done either by user's application program or by OS.
- Because disk I/O is allocated always in blocks, last block of most files suffer from **internal fragmentation ;** Larger the block size,greater the internal fragmentation.



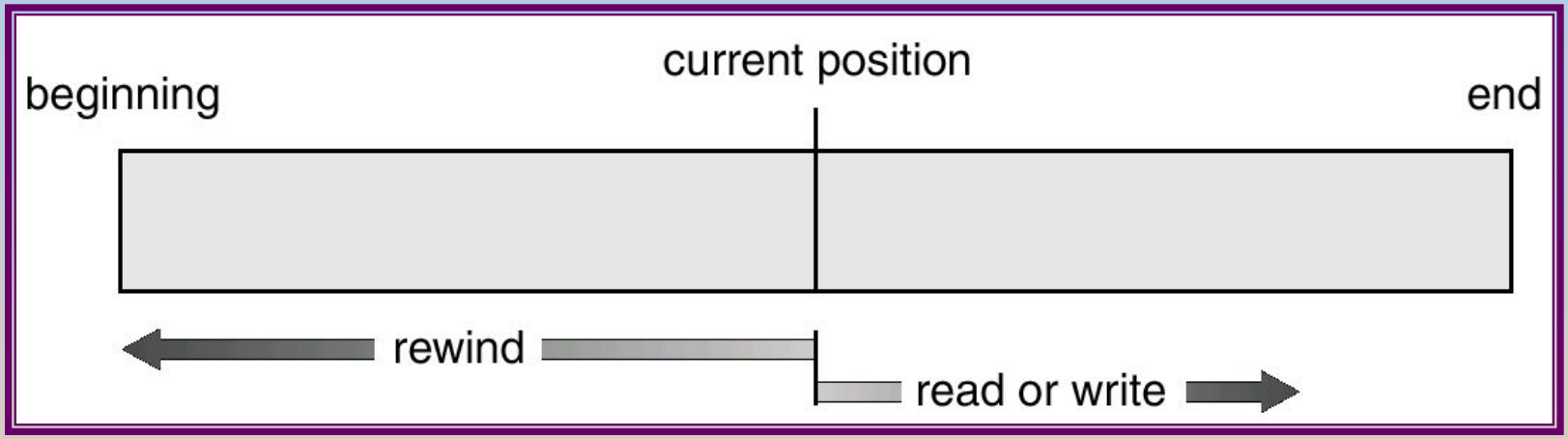
Access Methods

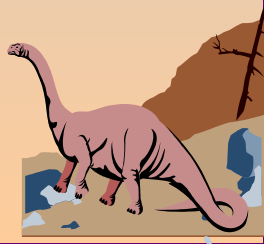
- Access methods:
  - Sequential access
  - Direct access
- **Sequential Access** – Information in the file is processed in order, one record after the other.
- Read operation reads a records and automatically increments the pointer.
- Write operation appends to the end of the file.
- Reset to the beginning and, skip forward or backward *n* records.
- Sequential access method is based on tape model of file.





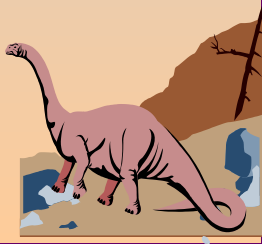
Sequential-access File





Access Methods(contd.)

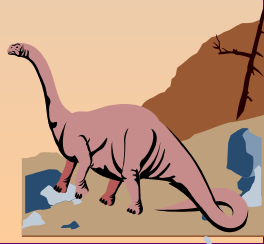
- **Direct method** – File is made up of fixed length logical records hat allow programs to read and write records rapidly in no particular order.
- Direct access method is based on disk model of file.
- File is viewed as numbered sequence of records or blocks.
- Allows arbitrary records to be accessed,eg,7,23,14,5.
- Allows immediate access to large amount of information.
- Block number provided by the user is **relative block number,** relative to the starting address of the logical file.
- Given a logical record length L, a request for record N is turned into I/O for L bytes starting from L*(N-1) (first record is N=1)
- Sequential access can be simulated on a direct access file.
- It is very difficult and cumbersome to simulate direct access on sequential access file.



Access Methods

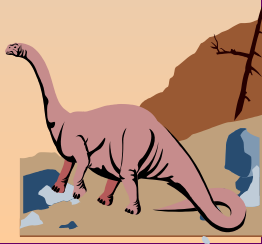
- **Sequential Access operations**
  - read next
  - write next
  - reset
  - no read after last write (rewrite)
- **Direct Access operations**
  - read n (nth record)
  - write n
  - position to n
    - read next
    - write next
  - rewrite n

  n = relative block number/record

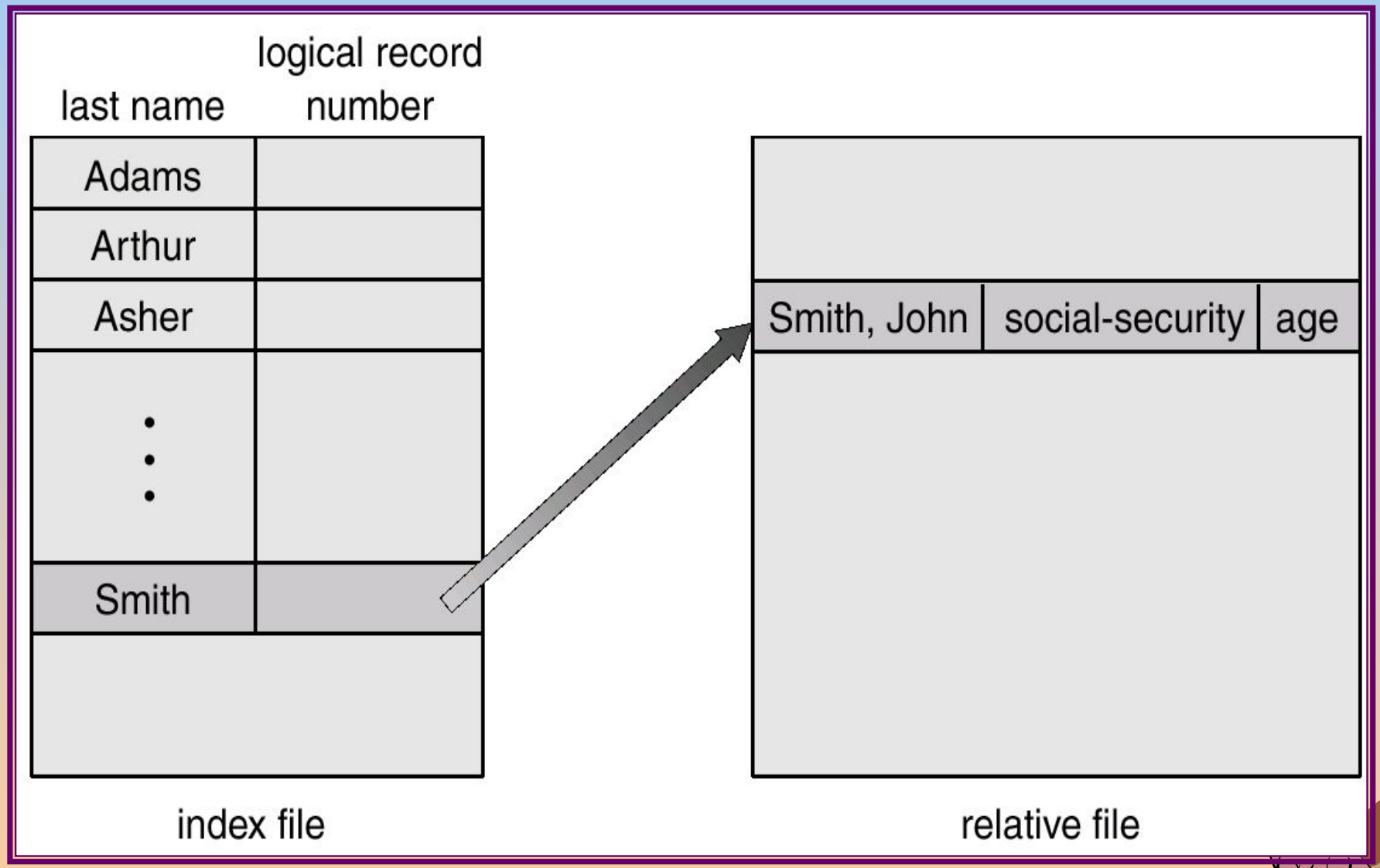


Other Access Methods

- Other access methods are build on direct access method.
- Index for a file – Index to a file contains pointers to blocks of file.
- To find a record in the file, we first search the index and then use the pointer in the index table to access the required record.
- Advantage of index – Binary search is possible on index file(might not be possible for original file if the size is large, coz it has to be brought in main memory)
- If the index file also becomes too large to handle, then we create an index to the index file.

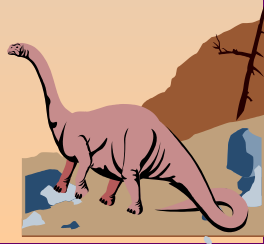


Example of Index and Relative Files

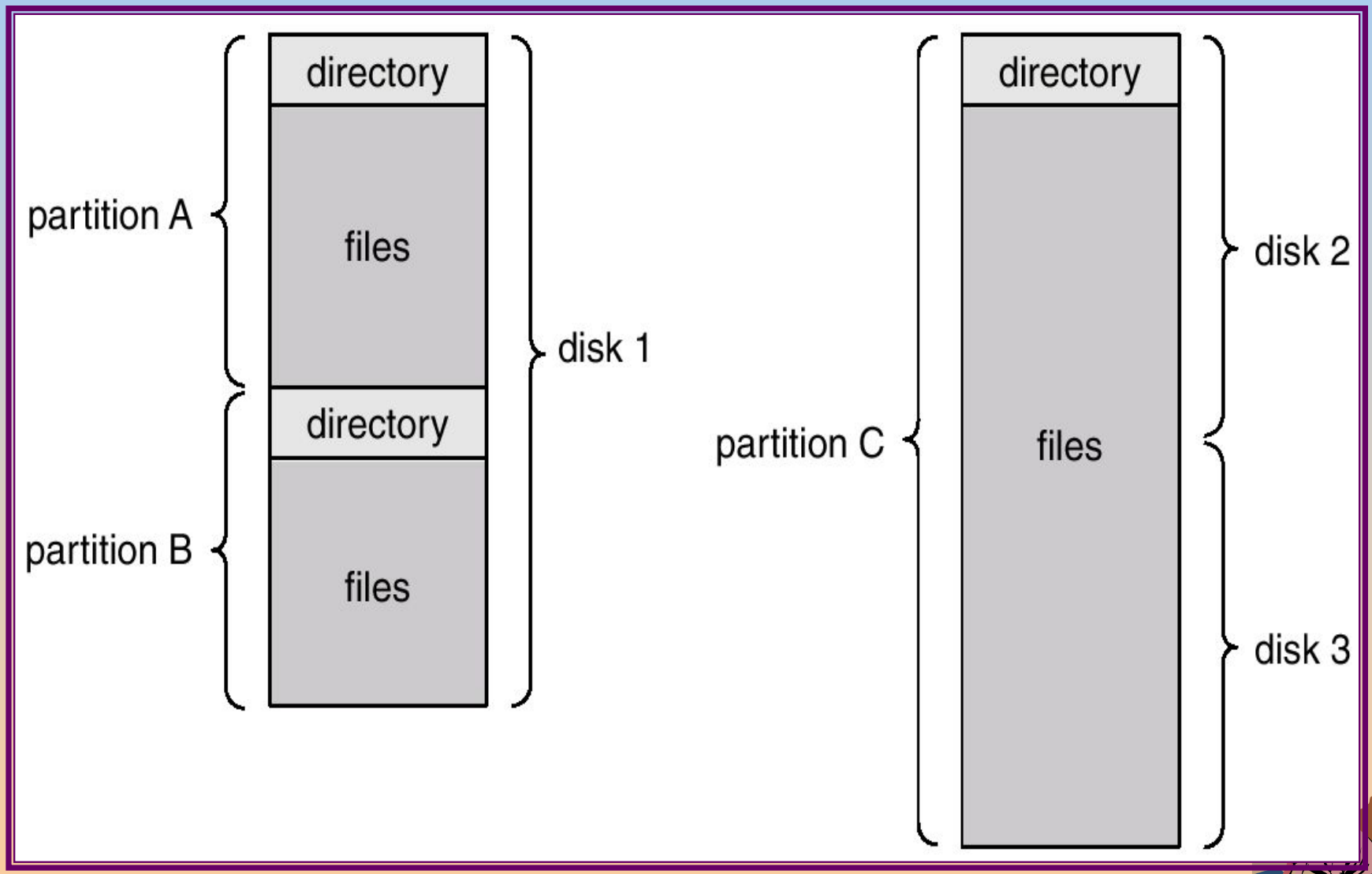


Directory Structure

- Organize data on the disk using **Partitions,** in which files and directories reside.
- A disk can have multiple partitions.
- Each partition is treated as a separate storage device.
- Each partition contains information about the files within that partition, kept in device directory.

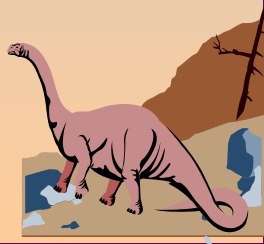


A Typical File-system Organization



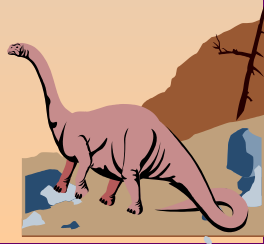
Information in a Device Directory

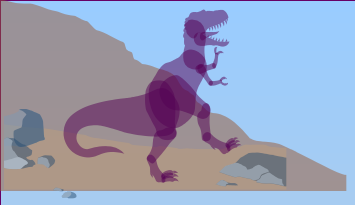
- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner ID
- Protection information



Operations Performed on Directory

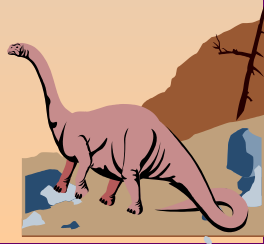
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

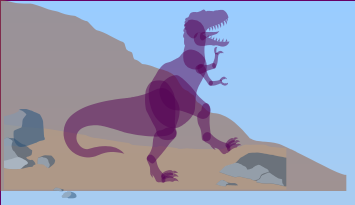




Logical Structure of a Directory

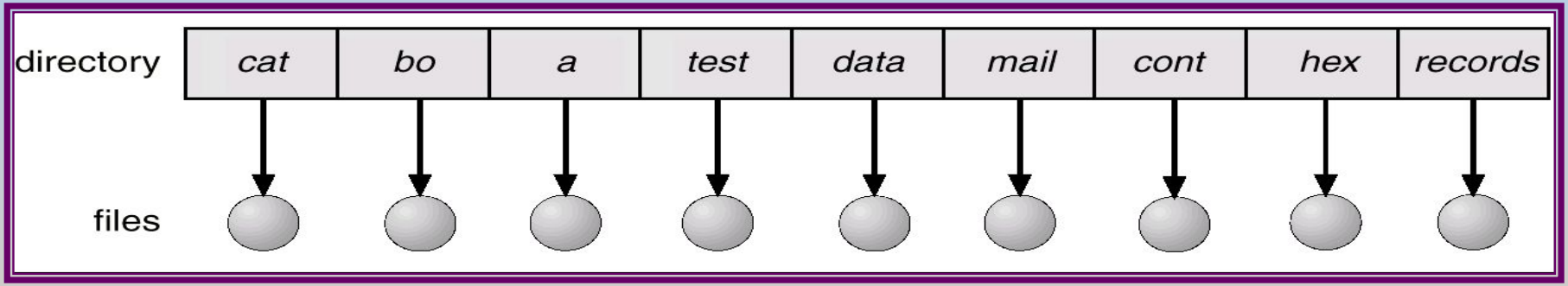
- Single-Level Directory
- Two-Level Directory
- Tree-Structured Directory
- Acyclic-Graph Directory
- General Graph Directory



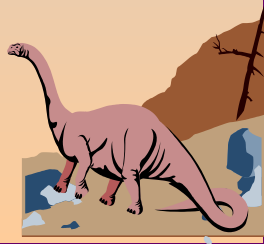


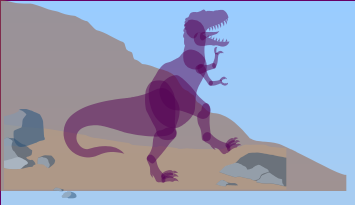
Single-Level Directory

- A single directory for all users.
- Easiest to support and understand.



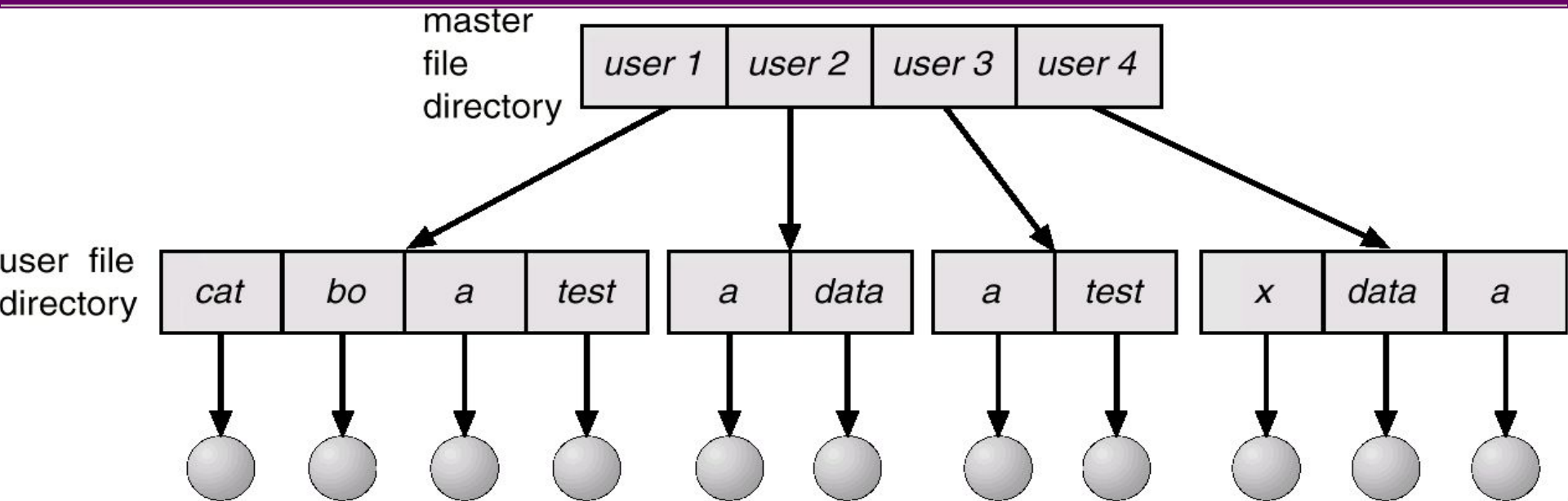
- Limitations of this scheme:
  - Naming problem – Since all the files are in the same directory, they must have unique names, even if they belong to different users.





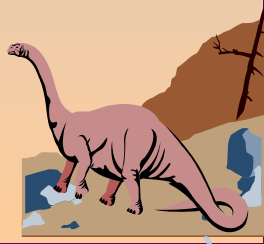
Two-Level Directory

- Separate directory for each **user**, called user file directory(UFD).
- Each UFD is placed in systems master file directory(MFD)
- MFD is indexed by user name.
- When a user refers to a particular file, only his UFD is searched.
- Disadvantage:
  - The structure isolates one user from another, but the users might want to access each other's files.



Two-Level Directory(contd.)

- To solve the disadvantage, the users are allowed to access each other's user space by specifying the path name of the file.
- **Path Name** – user name, partition name, directory name and file name.
- Path Name is used to name a file uniquely.
- Example – C:/try/help1.c , C is the partition name, try is the directory name, help1.c is the file name.



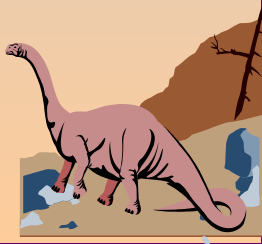
Two-Level Directory(contd.)

- **Special Case**
  - Special case occurs for system files.
  - Loaders,assemblers,compilers are defined as files.
  - When appropriate command is given, the file is loaded and executed.
  - As each user space is separate, these system files must be copied in every user space coz if each user does not specify the complete path name then only its own user directory is searched.
  - But copying these files in every space will take lot of memory space.

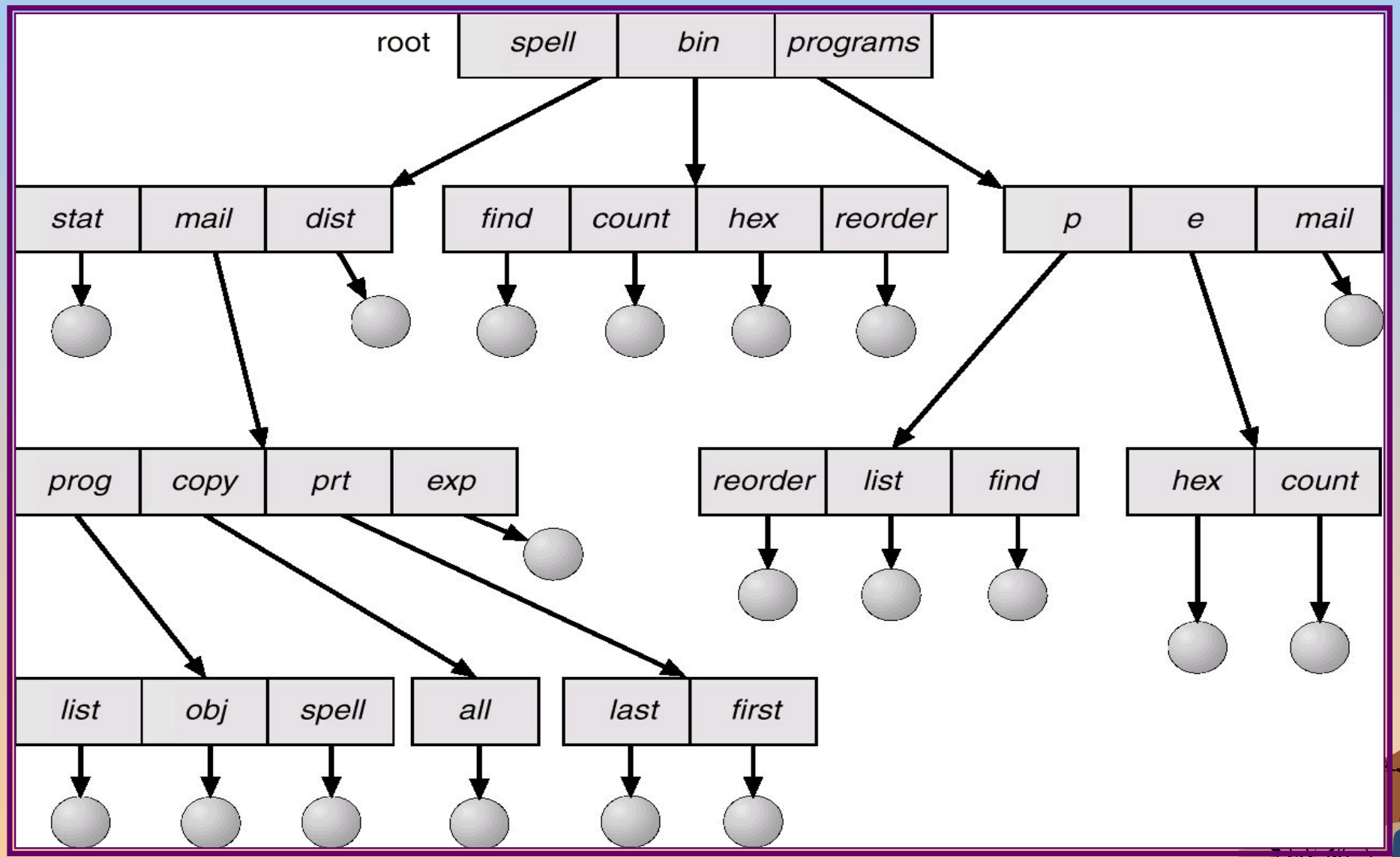
  Solution: **Search Path**

  Search path is specified for the system.

  When a system file execute command is made then all the sequence of directories specified in the search path are searched for the required file.

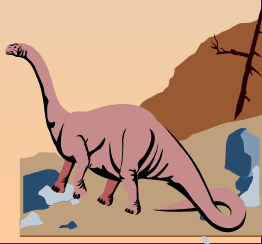


Tree-Structured Directories



Tree-Structured Directories (Contd.)

- A directory contains a set of files or sub-directories.
- One bit is used to specify a file(0) or a sub-directory(1)
- Current directory (working directory)
  - Each user has its own current directory.
  - User can change the current directory.
- Path names can be of two types:
  - Absolute path name
  - Relative path name
- Absolute path name – begins at the root and follows a path down to the specified file, giving all the directory names on the path.
- Relative path name – defines a path from the current directory.
- Eg. – if current dir is root/spell/mail, then relative path is prt/first
  for the same file, absolute path is root/spell/mail/prt/first.



Tree-Structured Directories (Contd.)

- **Deletion of directory-**

1. Directory should not be allowed to be deleted until the directory is empty.

   User has to manually delete all the directories and files in the directory to be deleted.

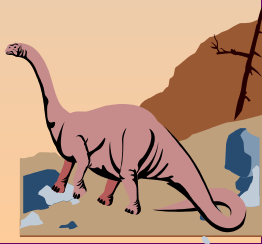
   Lot of work.

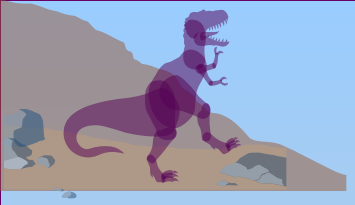
   Implemented by MS-DOS.

2. When a request is made to delete a file, all the files and directories inside it are also deleted.

   Implemented by UNIX.

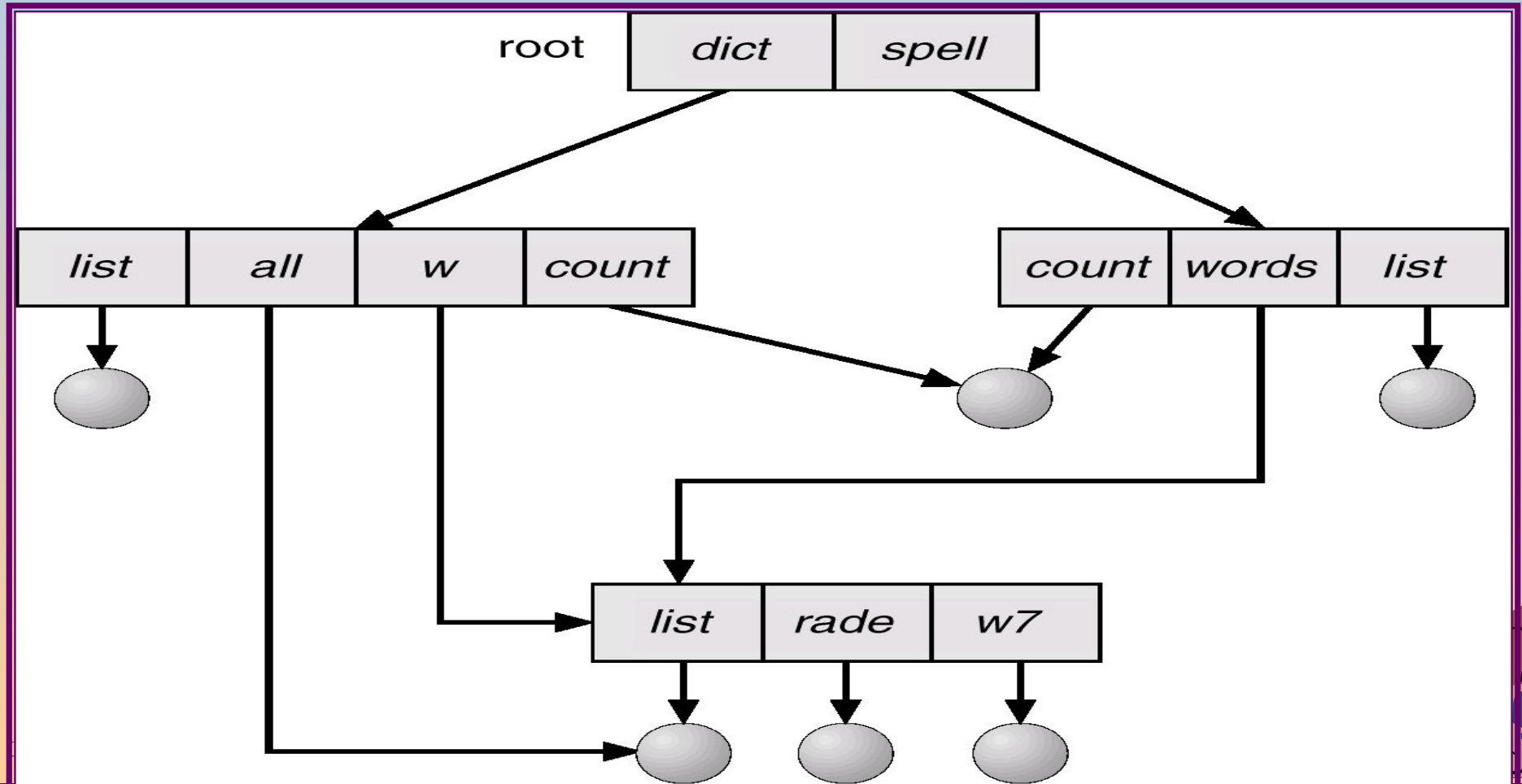
   Dangerous.





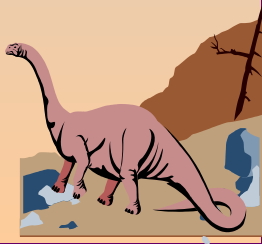
Acyclic-Graph Directories

- Have shared subdirectories and files.
- With shared files, only one actual file exist, so any changes made in one are visible to other.



Acyclic-Graph Directories (Contd.)

- One of the ways to implement shared subdirectories is using **links**.
- A link is a pointer to another file or sub-directory.
- UNIX uses this implementation.
- When a reference is made to a file,
  - directory is searched,
  - If the directory entry says it is a link.
  - The link is resolved and path to the file/sub-directory is found.
  - Real file is located using the link.



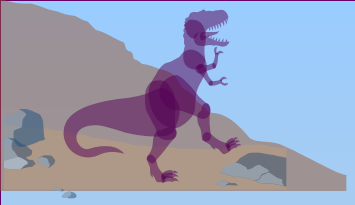
Acyclic-Graph Directories (Contd.)

- Another way to implement shared files is to simple duplicate all information about the file.
- Problem 1 – Maintaining consistency if file is modified.
- Problem 2 - In shared files, a file might have multiple absolute paths.
- If we are trying to access the whole file system and try to find data about all the files/sub-directories, then though same file can have multiple absolute paths, we might traverse the same file multiple times.
- Problem 3- Deletion of file. When can we de-allocate the space given to a shared file?
- Possibility 1 – To delete the shared file, whenever anyone deletes it.

  This might lead to dangling pointers(links)
- Possibility 2 – If system implements links using symbolic links/hard links

  Deletion of symbolic link leads to the deletion of the link **only**.

  Deletion of file leads to de-allocation of the file, and links are handled when they come in use(illegal file name error).



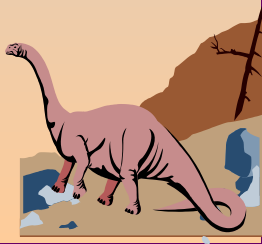
Acyclic-Graph Directories (Contd.)

- Possibility 3 – To preserve the file until one of the links is remaining.

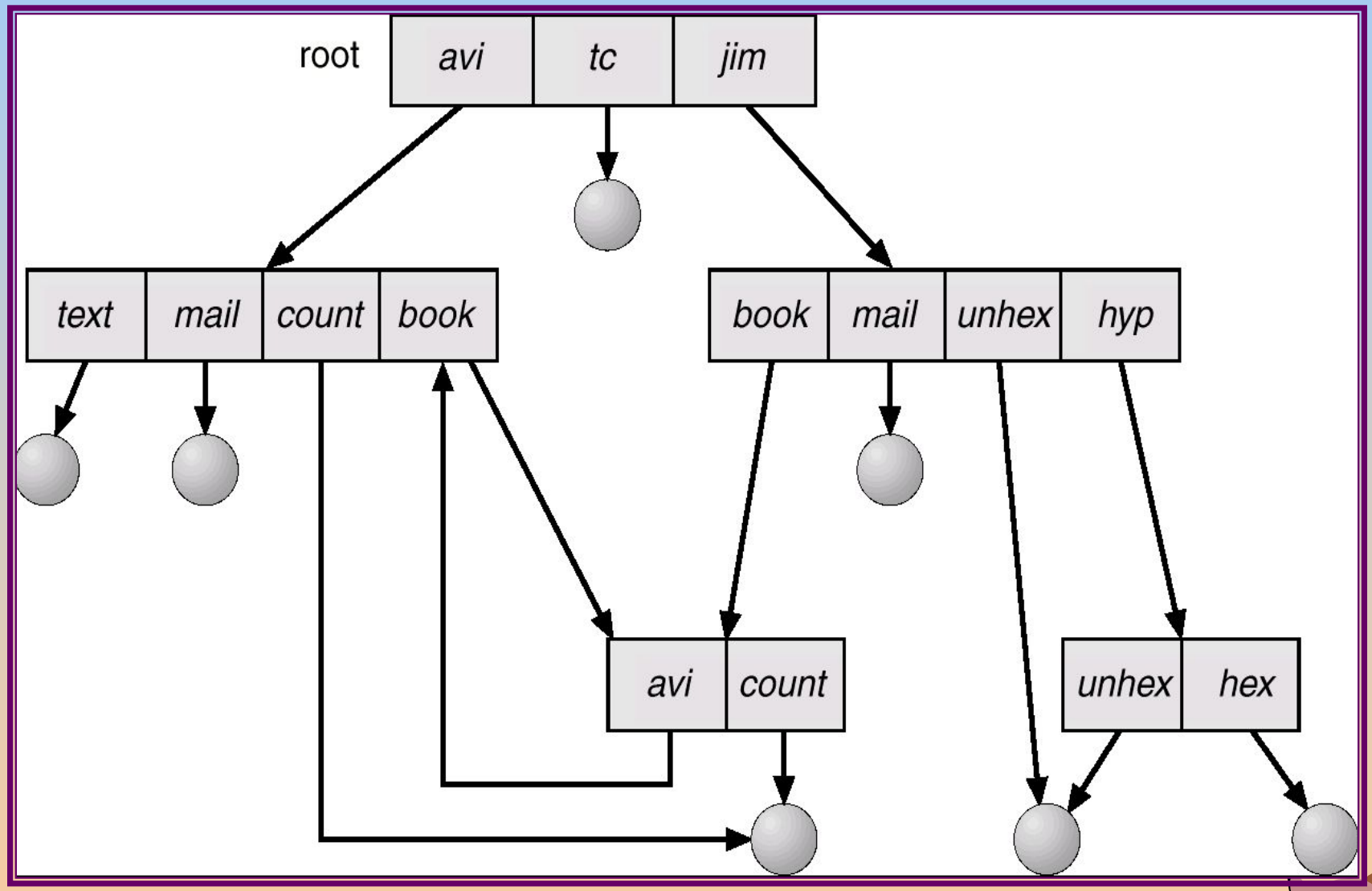
  Implemented by UNIX for hard links.

  A reference count is maintained for the number of links remaining.

- Problem with acyclic-graph directory implementation – Ensuring that there are no cycles.
- Advantage of acyclic-graph directory implementation – Not being stuck in an infinite loop of traversal(coz no cycle).



General Graph Directory



General Graph Directory (Contd.)

- How can we avoid looping continuously in the cycle during directory traversal?
  - Arbitrarily limit the number of files that can be traversed.
  - Garbage Collection
  - Bypass links
- Garbage Collection – involves traversing the entire file system, marking everything that can be accessed.

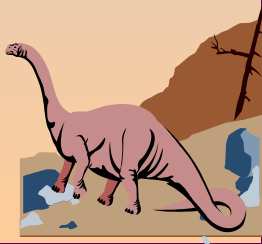
  Then, a second pass collects everything that is not marked onto a list of free space.

  Usually used in the case of occurrence of self references. The reference count may be non-zero, even when it is no longer possible to reach the file/sub-directory.

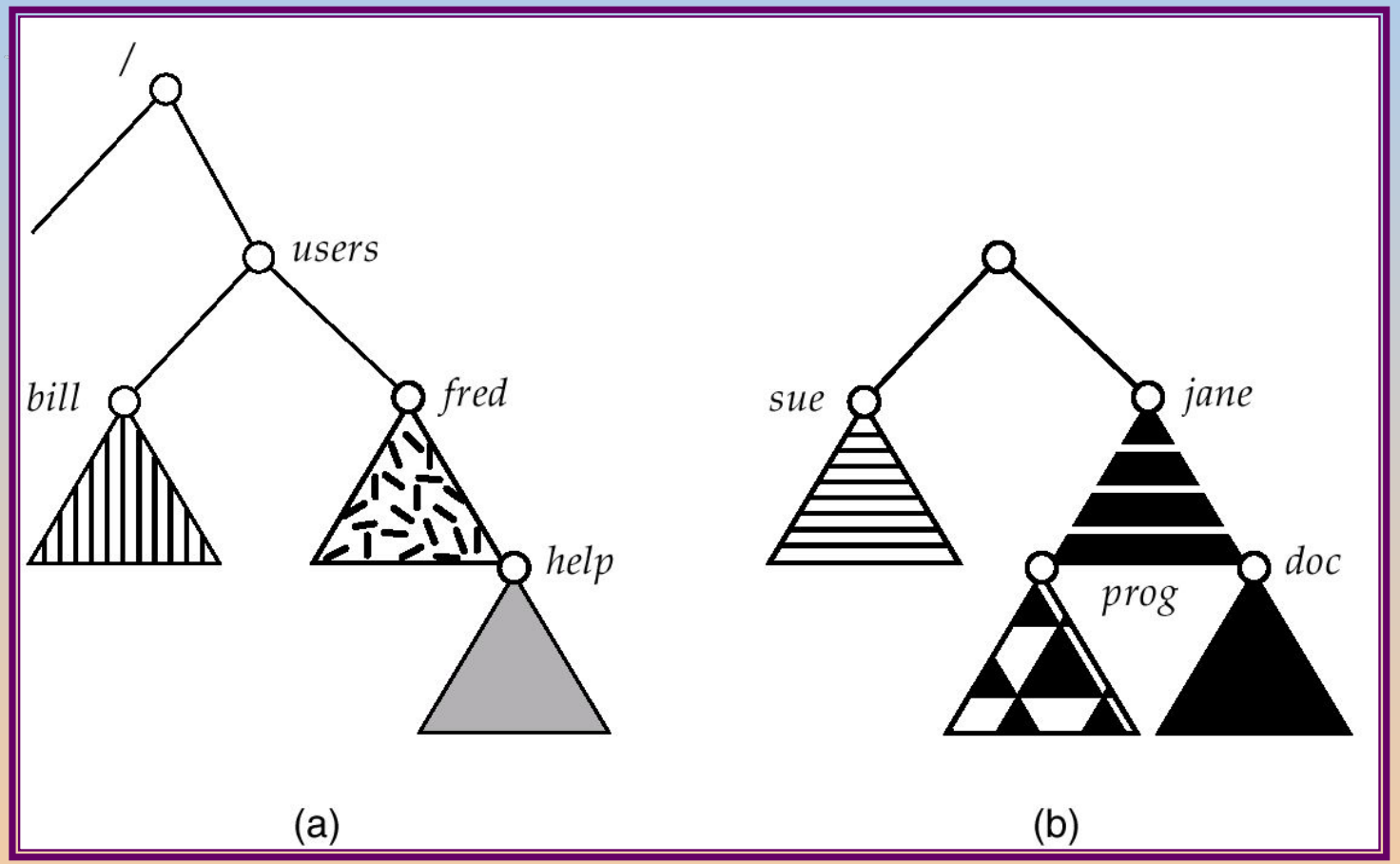
  Disadvantage – Extremely time consuming.
- Bypass Links – Cycles are avoided if links are avoided during directory traversals. Also no extra overhead is incurred.

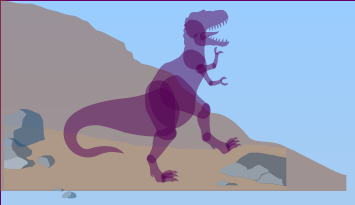
File System Mounting

- Just as a file must be opened before it is used, a file system must be **mounted** before it can be accessed.
- Mount Procedure
  - OS is given the name of the device/directory and the location within the file structure at which to attach the file system.
- A un-mounted file system is mounted at a **mount point**.

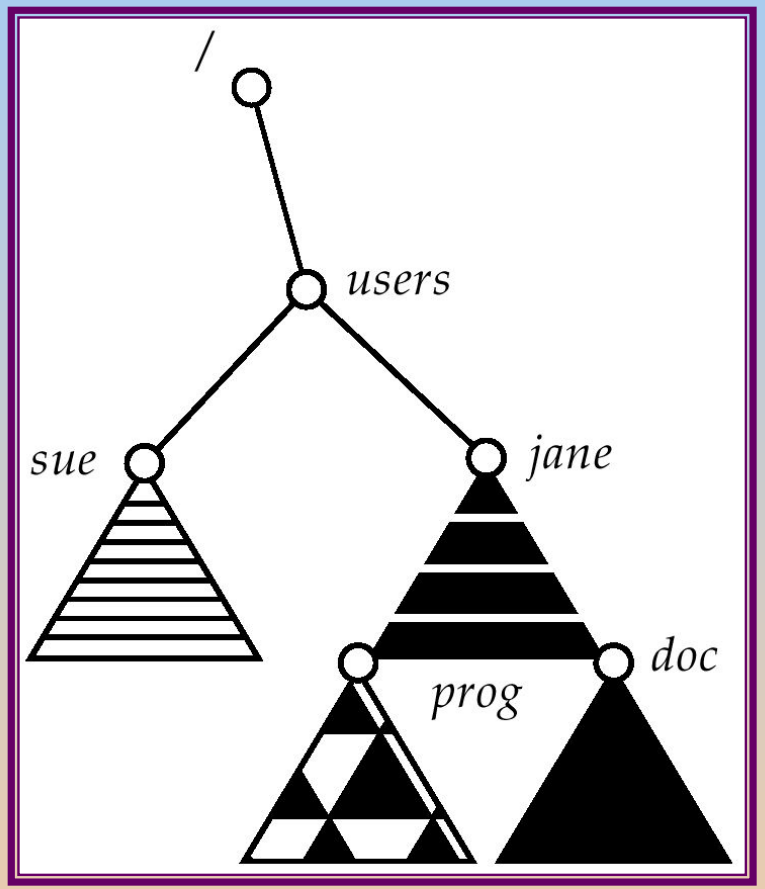


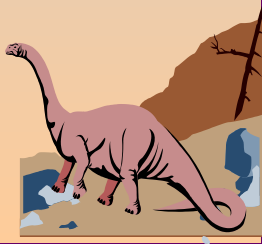
(a) Existing. (b) Unmounted Partition





Mount Point

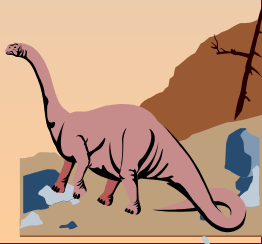




Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List
- Another way to implement protection is to associate password with every file.

  Problem with this approach – user has to remember lot of passwords.



Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

  RWX

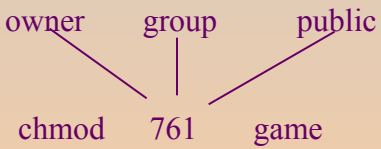
  a) **owner access** 7 ⇒ 1 1 1

  RWX

  b) **group access** 6 ⇒ 1 1 0

  RWX

  c) **public access** 1 ⇒ 0 0 1
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

chgrp G game