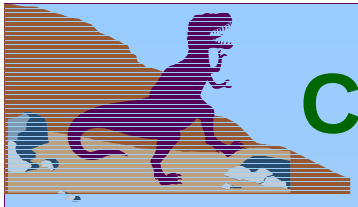
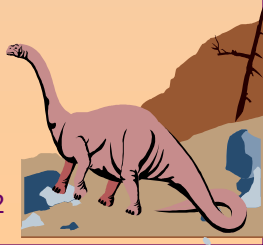
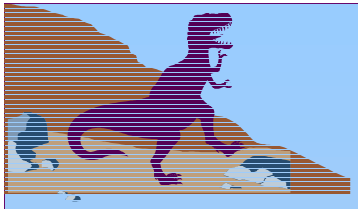


Chapter 11: File-System Interface

Chapter Outline

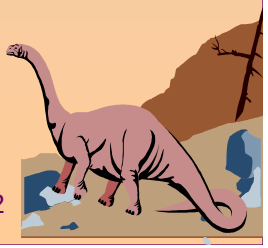
- File Concept
- Access Methods
- Directory Structure
- File System Mounting
- File Sharing
- Protection

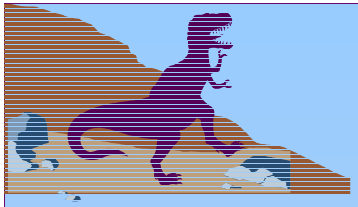




File Systems

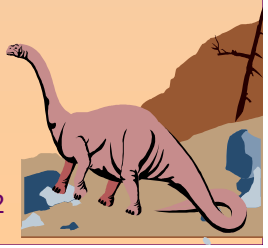
- *File System consists of*
  - *A collection of files*
  - *A directory structure*
  - *(possibly) partitions*
- *Important Issues*
  - *File protection*
  - *The semantics of file sharing*
- ***Note:*** *Historically, operating systems and file systems have been viewed as distinct entities.*
- *From the perspective of the modern user, this distinction is often blurred.*

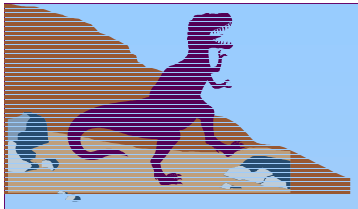




File Concept

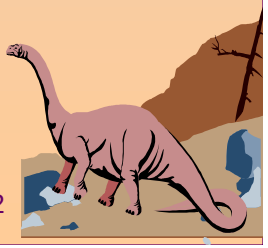
- *The operating system provides a uniform logical abstraction for the physical storage of information.*
- *Storage devices are nonvolatile.*
- *A file is a named collection of related information that is recorded on secondary storage.*
- Contiguous logical address space
- Types:
  - Data
    - ✔ numeric
    - ✔ character
    - ✔ binary
  - Program
    - ✔ *Source, object and executable file formats*

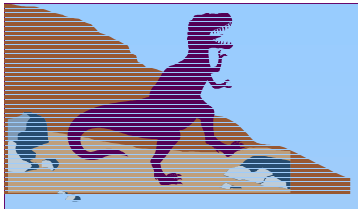




File Attributes

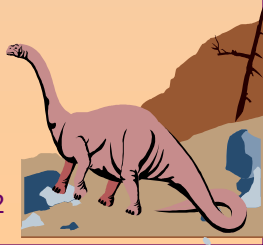
- **Name** – only information kept in human-readable form.
- *Identifier – a unique tag (i.e., an internal number) that identifies the file within the file system.*
- **Type** – needed for systems that support different types.
- **Location** – a pointer to file location on device.
- **Size** – current 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.

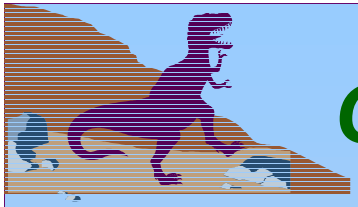




File Operations

- Create
- Write
- Read
- Reposition within file – file seek
- Delete
- Truncate
- 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.

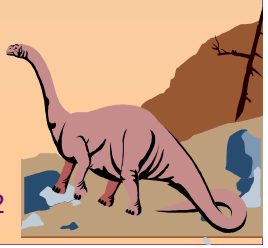


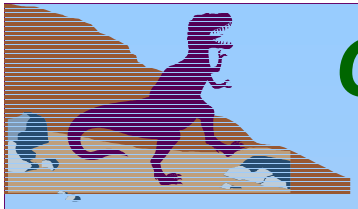


Claypool Example: Unix open( )

```
int open(char *path, int flags [, int mode])
```

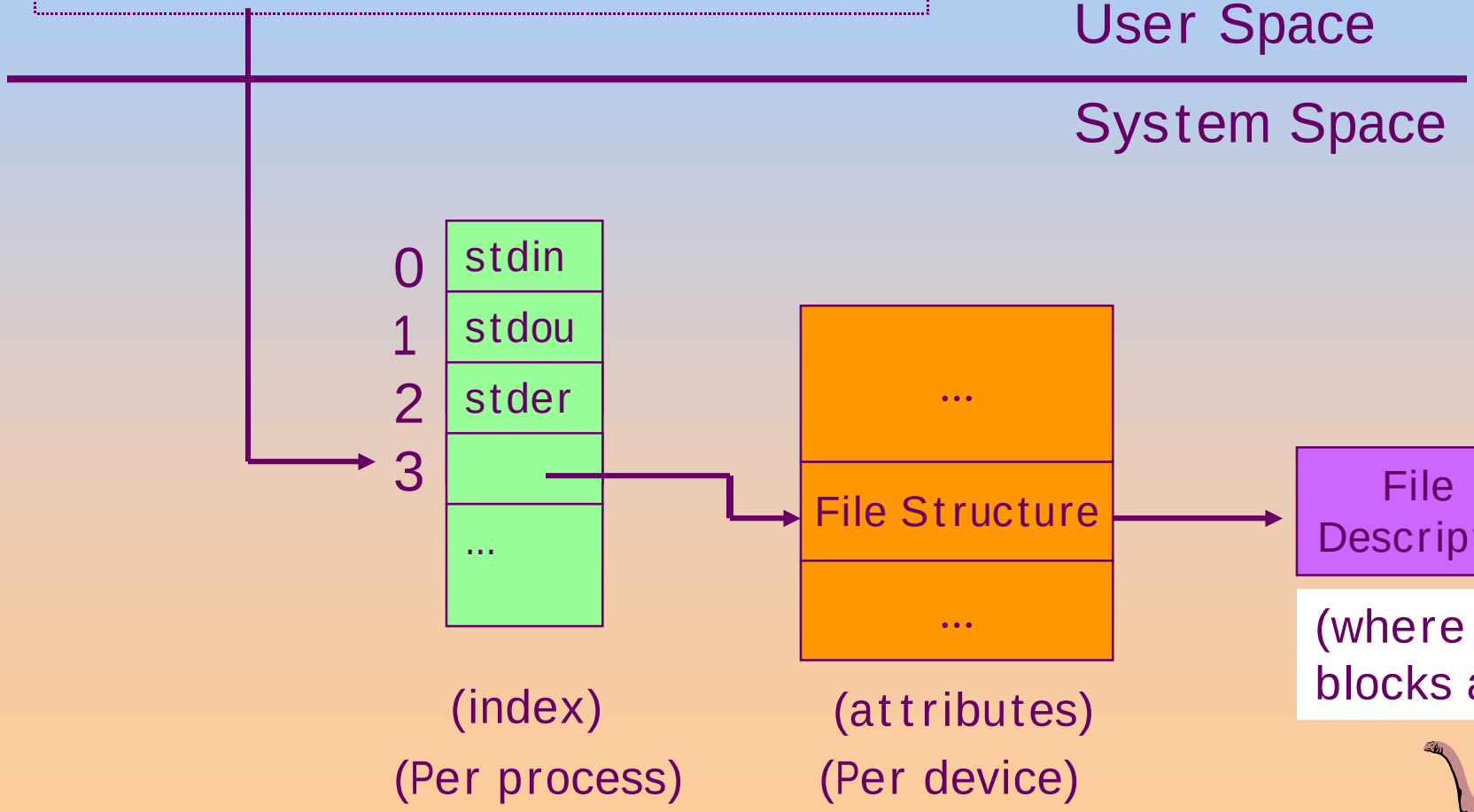
- ***path*** is name of file
- ***flags*** is bitmap to set switch
  - O_RDONLY, O_WRONLY…
  - O_CREATE then use `mode` for perms
- On success, returns index

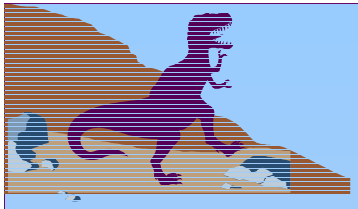




Claypool Example: Unix open( ) Under the Hood

```
int fid = open("blah", flags);
read(fid, …);
```



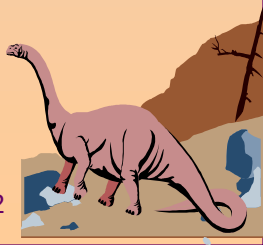


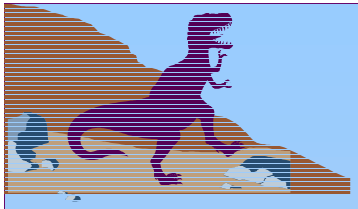
Claypool Example: WinNT/2000 `CreateFile()`

- Returns file object handle:

```
HANDLE CreateFile (
  lpFileName, // name of file
  dwDesiredAccess, // read-write
  dwShareMode,  // shared or not
  lpSecurity, // permissions
  ...
)
```

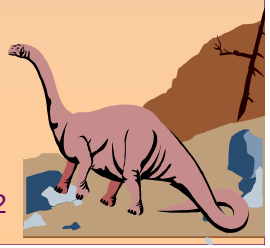
- File objects used for all: files, directories, disk drives, ports, pipes, sockets and console

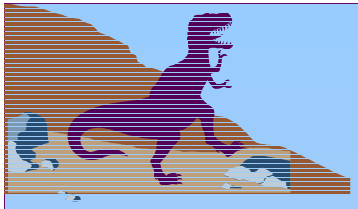




File Types – Name, Extension

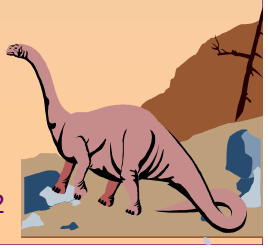
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data,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

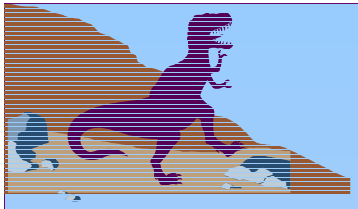




File Structure

- *File types may be used to indicate the internal structure of a file.*
- *An OS may **require** a file to have a specific structure so that the OS will provide special operations for those files conforming to the set of system-supported file structures.*
  - *e.g., VMS supported three defined file structures.*
  - *Others (UNIX, MS-DOS) support a **minimal** number of file structures.*
- *This is an obvious tradeoff between flexibility and system support!*

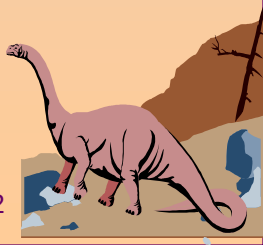


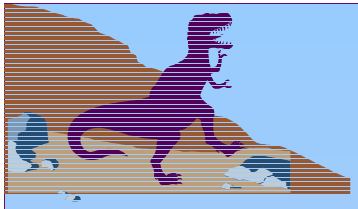


Access Methods

- *Access methods determine the way that files are accessed and read into memory.*
- *Some systems **only** support one access method while other OS's support many access methods.*
- **Sequential Access**
  - *The most common method used by editors and compilers.*
  - *Information is processed in order.*

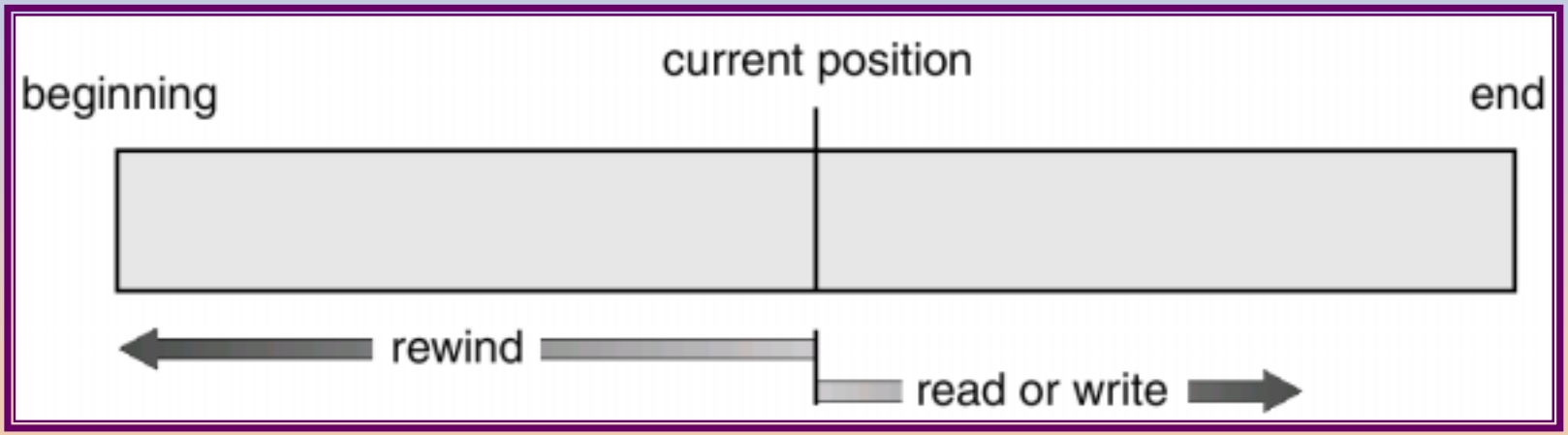
read next
write next
reset
no *read after last write*
(rewrite)

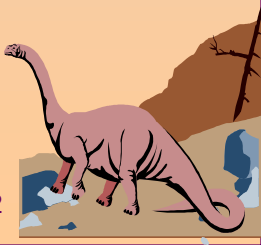


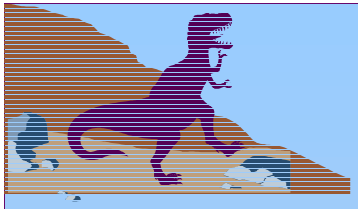


Sequential Access File

- *Based on a **tape model** of a file.*
- *May be able to skip forward **n** records.*



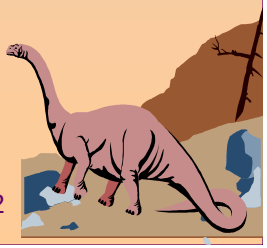


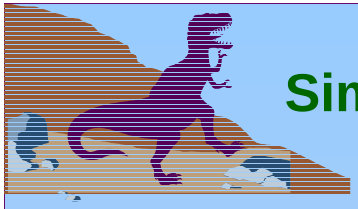


Direct Access File

- *File is made up of fixed-length logical records that allow programs to read and write records in no particular order.*
- *The files is viewed as a numbered sequence of blocks or records.*
- *Very useful in databases.*
- **Direct Access** {*n* = relative block number}

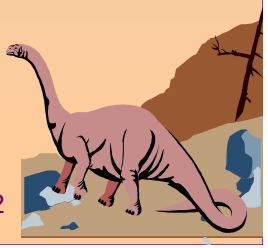
  read n
  write n
  position to n
  read next
  write next
  rewrite n





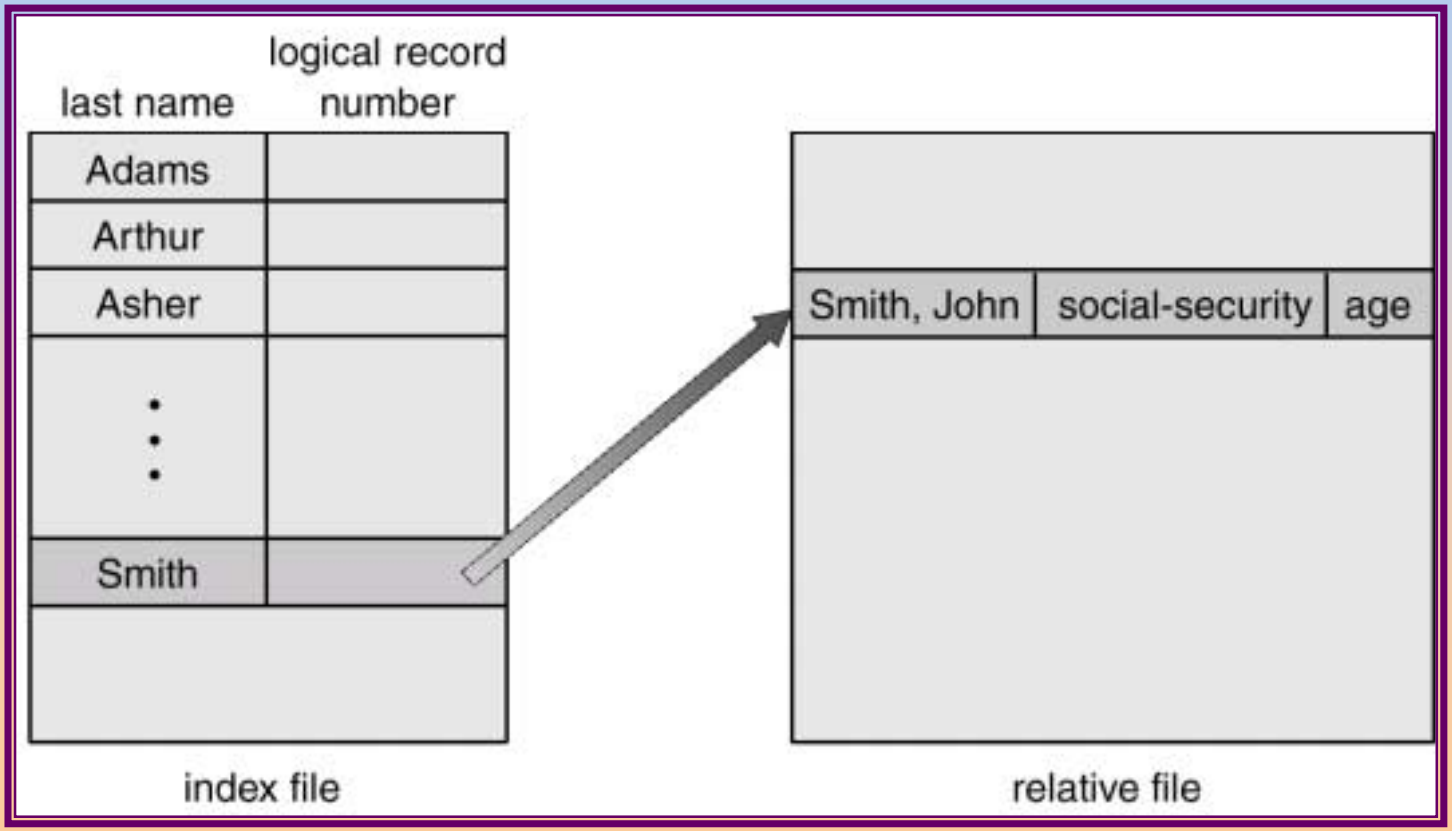
Simulation of Sequential Access on a Direct-access File

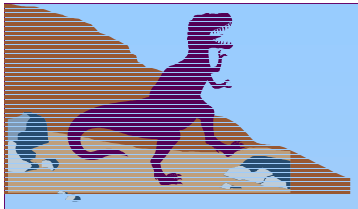
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;



Example of Index and Relative Files

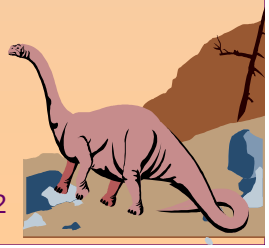
- *Index Sequential Access Method (ISAM) – uses indexes in a hierarchy to point to records in a file.*

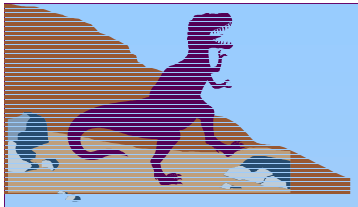




Directory Structure

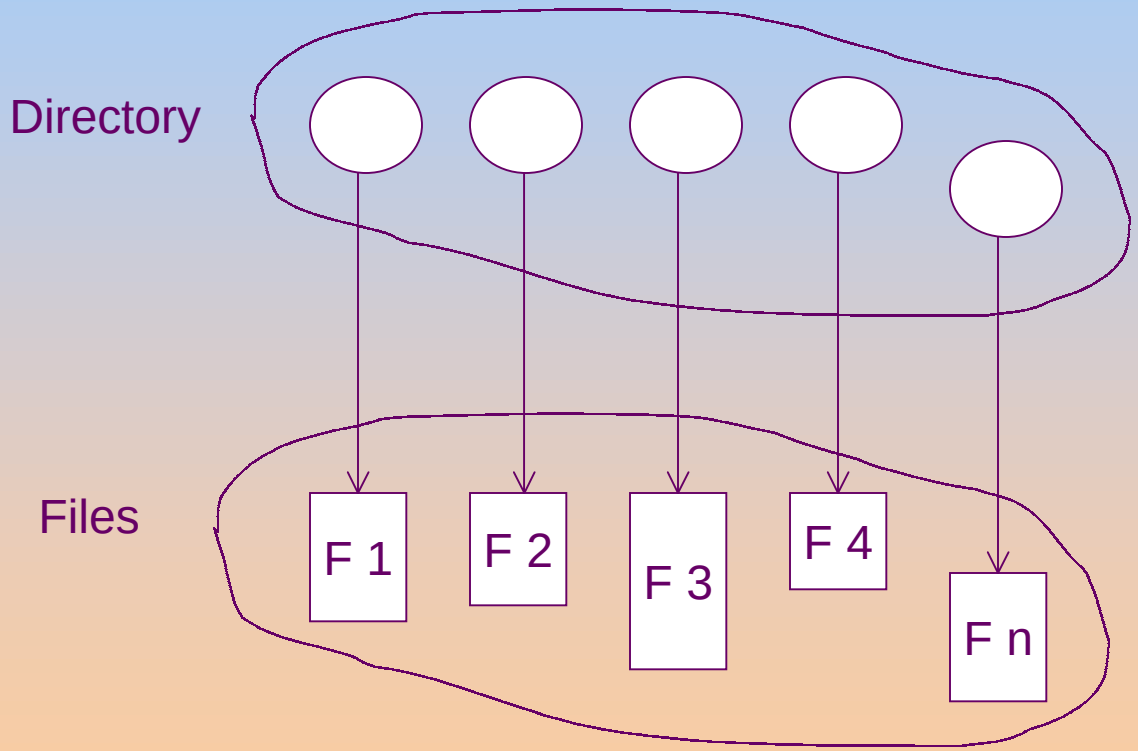
- *Partitions (or Volumes) – can be viewed as the abstraction of **virtual disks.***
- *Disks can be partitioned into separate areas such that each partition is treated as a separate storage device.*
- *The other way -- a partition may be defined to be **more than one** disk device.*
- *Partitions can store **multiple** operating systems such that a system can boot more than one OS.*
- *Each partition contains information about files in a **device directory** (or a **VTOC** – Volume Table of Contents).*
- *Each directory records file attribute information.*



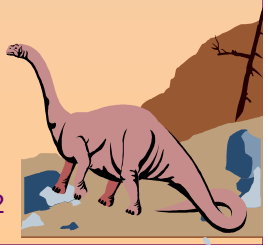


Directory Structure

- A collection of nodes containing information about all files.

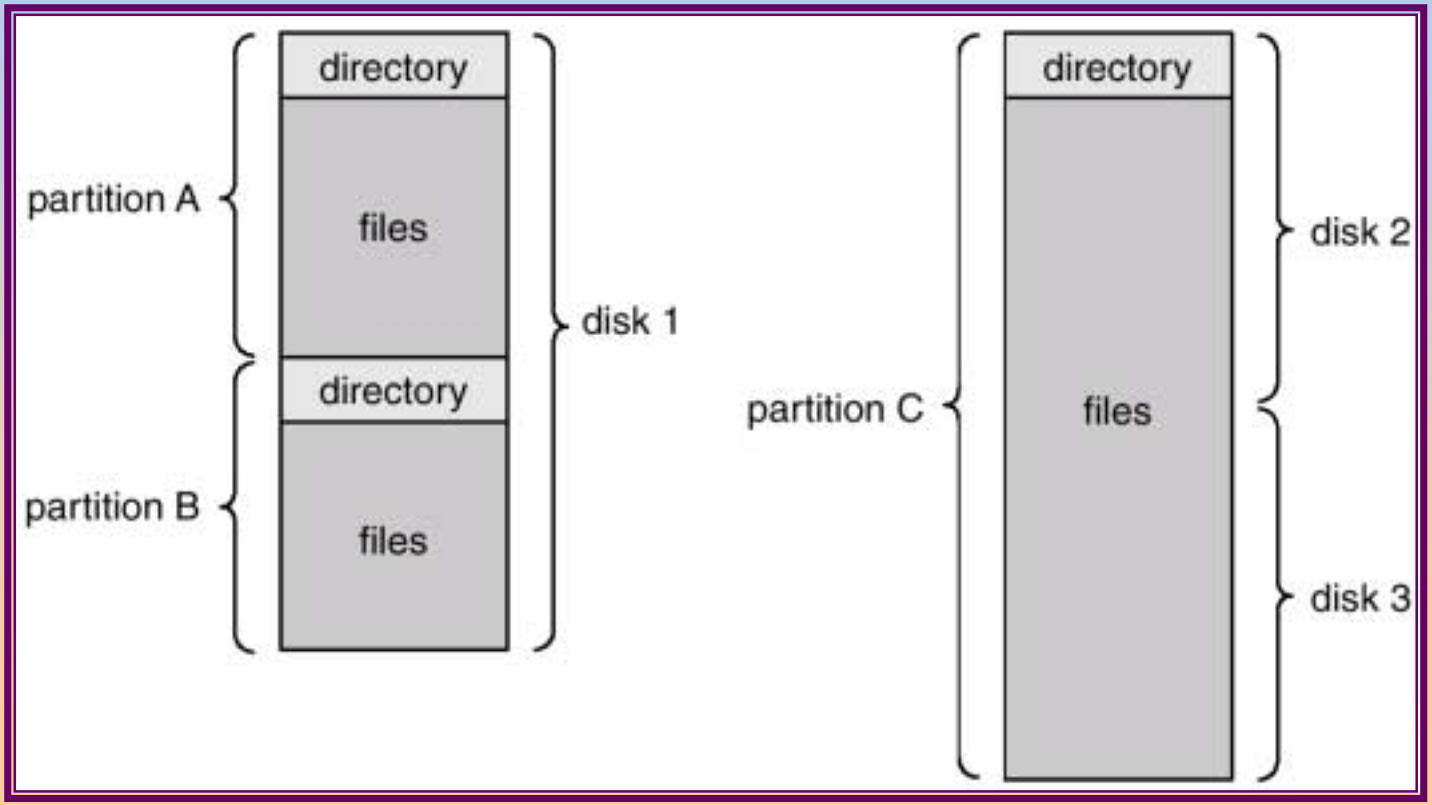


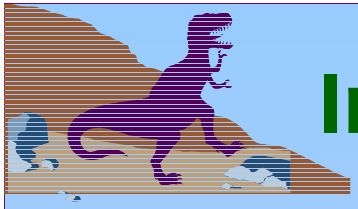
Both the directory structure and the files reside on disk.
Backups of these two structures are kept on tapes.



A Typical File System Organization

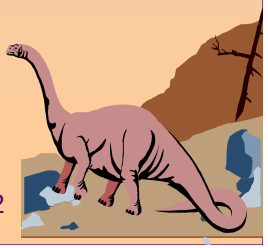
- *A directory can viewed as a "symbol table" that translates file names into their directory entries.*

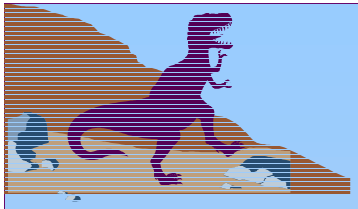




Information in a Device Directory

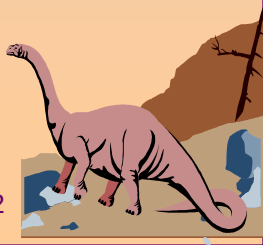
- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

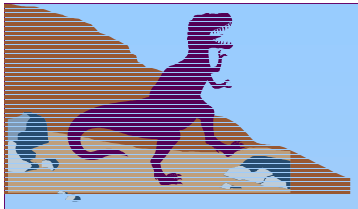




Directory Operations

- Search for a file – *need to find a particular entry or be able to find file names based on a pattern match.*
- Create a file - *and **add** its entry to the directory.*
- Delete a file – *and **remove** it from the directory.*
- List a directory – *list both the files in the directory and the directory contents for each file.*
- Rename a file – *renaming may imply changing the position of the file entry in the directory structure.*
- Traverse the file system – *the directory needs a logical structure such that every directory and every file within each directory can be accessing **efficiently.***

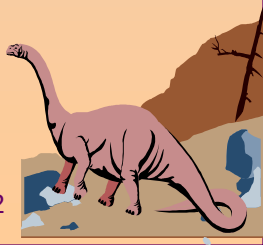


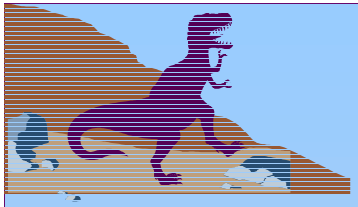


Directory Design Goal

To organize the logical structure to obtain:

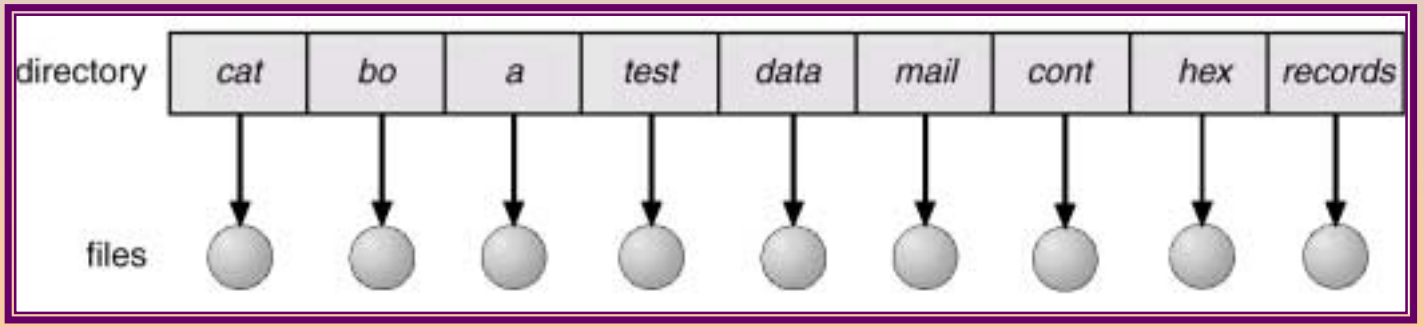
- **Efficiency** – locating a file quickly.
- **Naming** – convenient to users.
  - Two users can have same name for different files.
  - The same file can have several different names.
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)

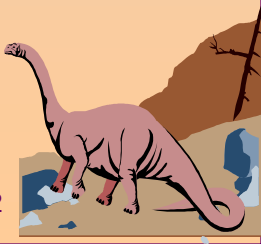


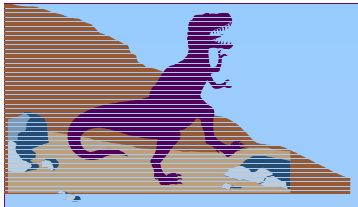


Single-Level Directory

- *The simplest solution::* A single-level directory with file entries for all users contained in the same directory.
- *Advantages:*
  - *Easy to support and understand.*
- *Disadvantages::*
  - *Requires **unique file names** {the naming problem}.*
  - *No natural system for keeping track of file names {the grouping problem}.*

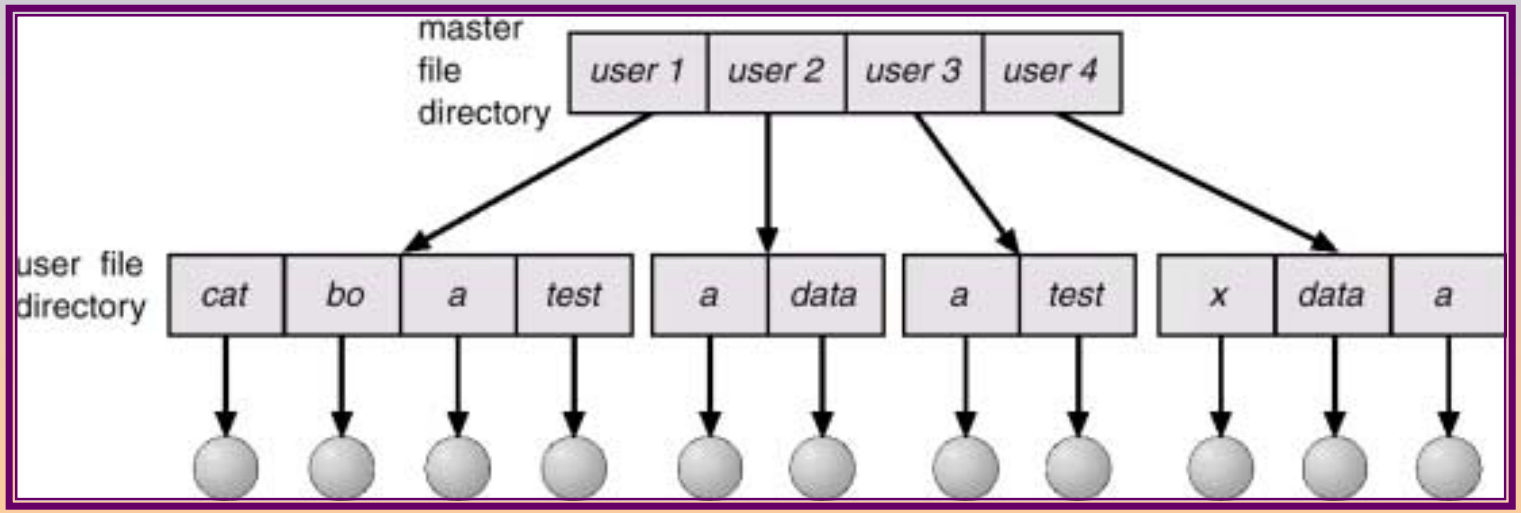


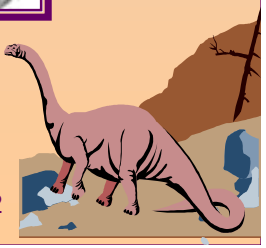


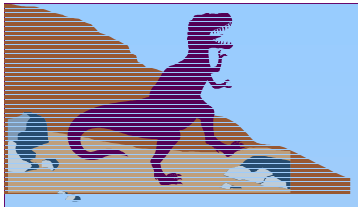


Two-Level Directory

- *Standard solution:* a separate directory for each user.
- *The system's* ***Master File Directory (MFD)*** *has pointers to individual* ***User File Directories (UFD's).***
- File names default to localized UFD for all operations.

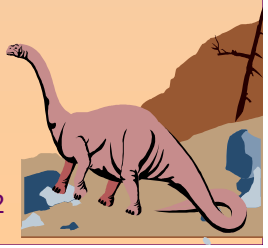


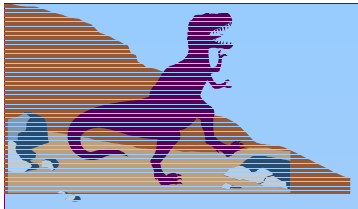




Two-Level Directory

- *Advantages*
  - *Solves the name-collision problem.*
  - *Isolates users from one another ➔ a form of protection.*
  - *Efficient searching.*
- *Disadvantages*
  - *Restricts user cooperation.*
  - *No logical grouping capability (other than by user).*





Path Name

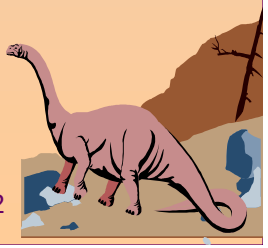
- *If a user can access another user's files, the concept of* ***path name*** *is needed.*
- *In two-level directory, this tree structure has MFD as root of path through UFD to user file name at leaf.*
- ***Path name :: username + filename***
- *Standard syntax -- /user/file.ext*

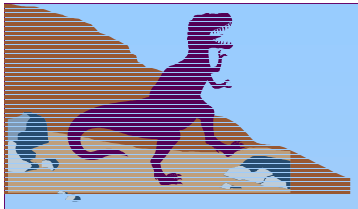
Add Partitions

- *Additional syntax needed to specify partition*
  - *e.g. in MS-DOS* ***C:\user\file.ext***

System files

- *Dotted files in Unix*

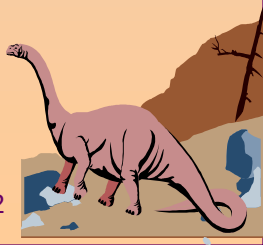


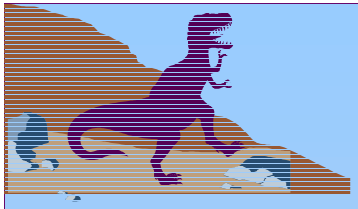


Path Name

System File Issues

- *Those programs provided as part of the system (e.g. loaders, compilers, utility routines)*
- *e.g., Dotted files in Unix*
- *Another tradeoff issue*
  - *Copy all system files into each UFD* **OR**
  - *Create **special** user file directory that contains the system files.*
    - ✔ *Note: This complicates the file search procedure.*
    - ✔ *Default is to search local UFD, and then special UFD.*
- *To override this default search scheme, the user specifies a specific sequence of directories to be searched when a files is named – the search path.*



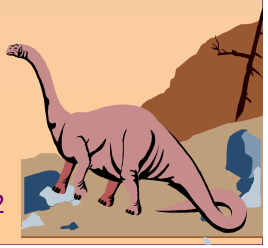


Tree-Structured Directories

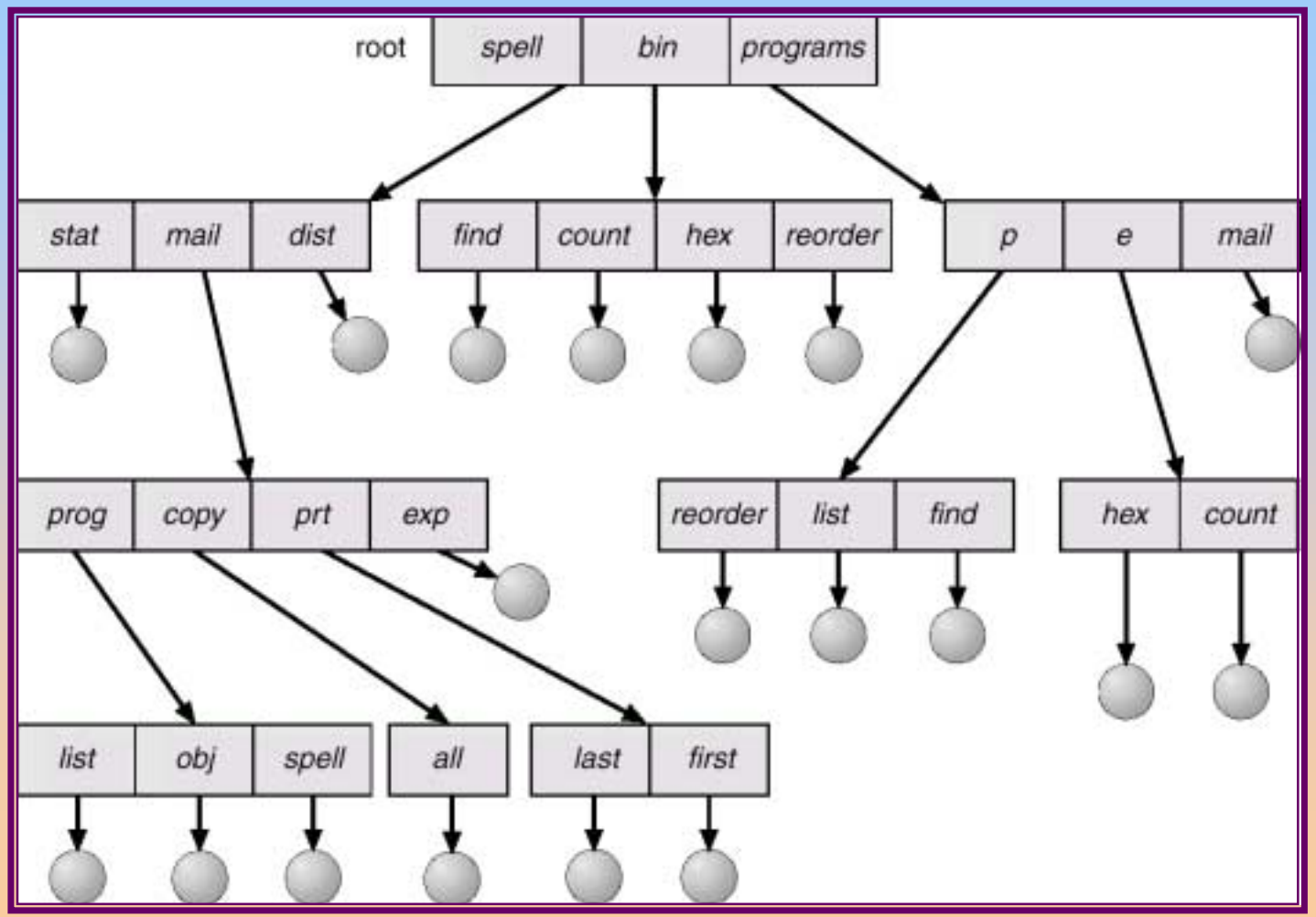
- *This generalization to a directory tree structure of arbitrary height allows users to create their own subdirectories and organize their files accordingly.*

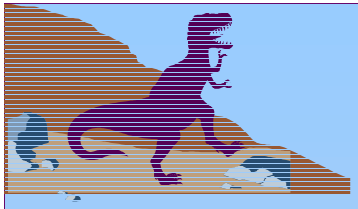
Directory

- *Becomes simply another file.*
- *Contains a set of files or subdirectories.*
- *All directories have the same internal format.*
- *One bit in directory entry defines entry as file or directory.*
- *Special commands are used to create and delete directories.*



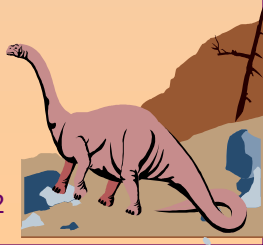
Tree-Structured Directories

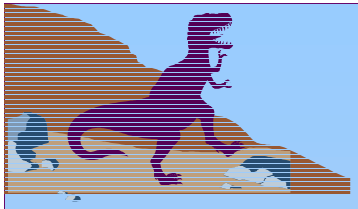




Tree-Structured Directories

- *Advantages*
  - Efficient searching
  - Grouping Capability
- Each user has a ***current directory*** (working directory)
  - **cd** /spell/mail/prog
  - **type** list





Tree-Structured Directories

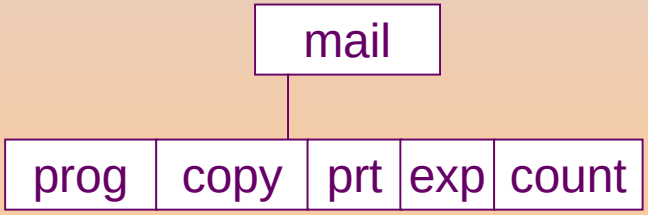
- **Absolute** or **relative** path name
- Creating a new file is done in current directory.
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory.

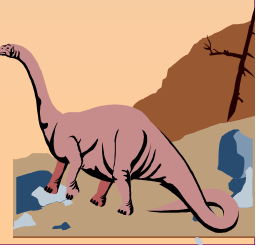
  mkdir <dir-name>

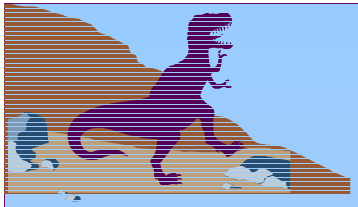
  Example: if in current directory **/mail**

  mkdir count



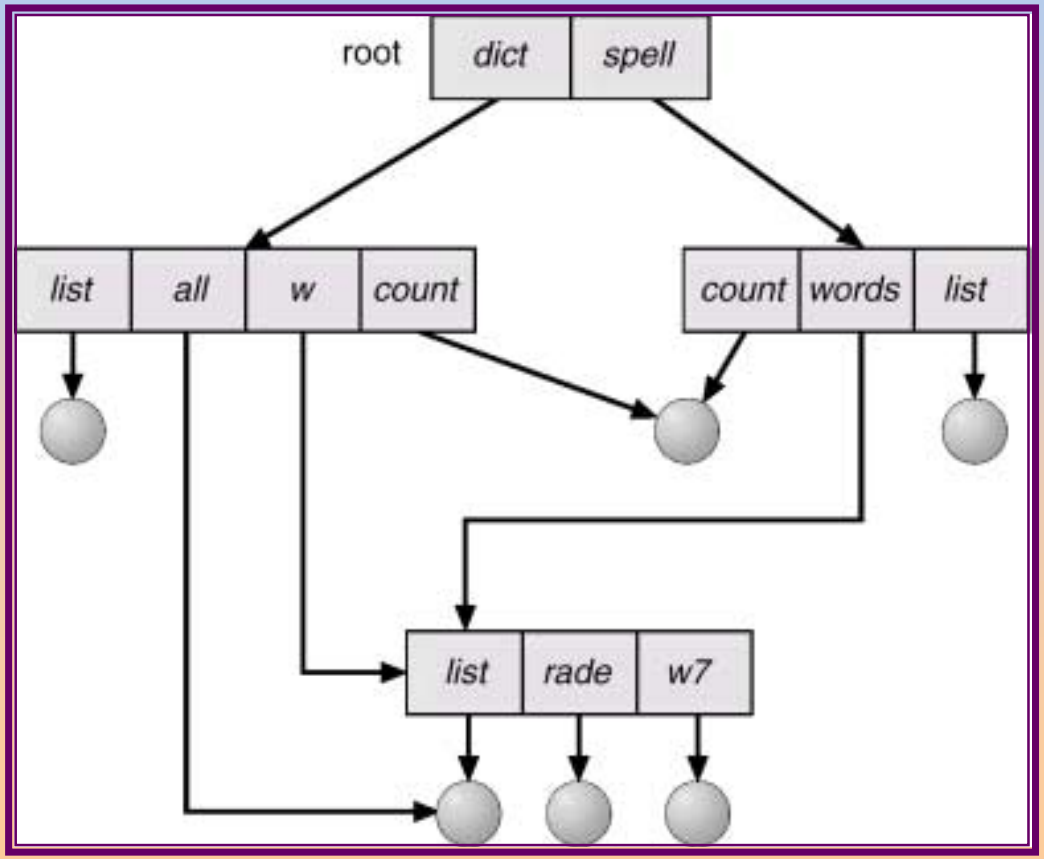
Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.

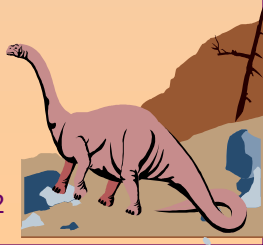


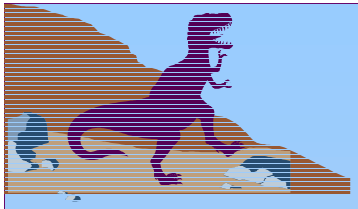


Acyclic-Graph Directories

- *A tree structure prohibits the sharing of files or directories.*
- Acyclic graphs allow directories to have shared subdirectories and files.



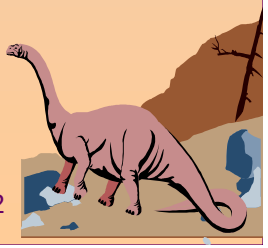


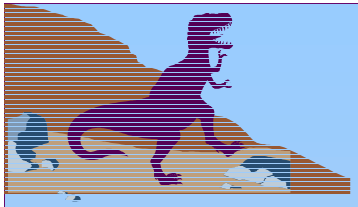


Acyclic-Graph Directories

Implementations of shared files or directories

- *Links*
  - *A new type of directory entry*
  - *Effectively a **pointer** to another file or subdirectory*
    - ✔ *Implemented as an absolute or relative path name.*
  - *A link entry is resolved by using the path name to locate the real file.* {Note the inefficiency !}
  - *Problems are similar to **aliasing** because distinct file names can refer to the same file.*
- *Duplicate all information in sharing directories*
  - *Big problem is maintaining consistency when the file is modified.*

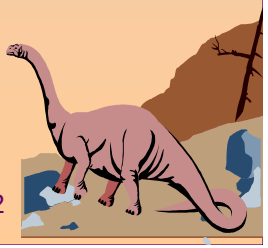


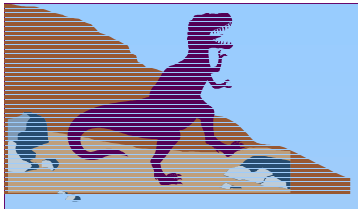


Acyclic-Graph Directories

Problems to consider with link implementation:

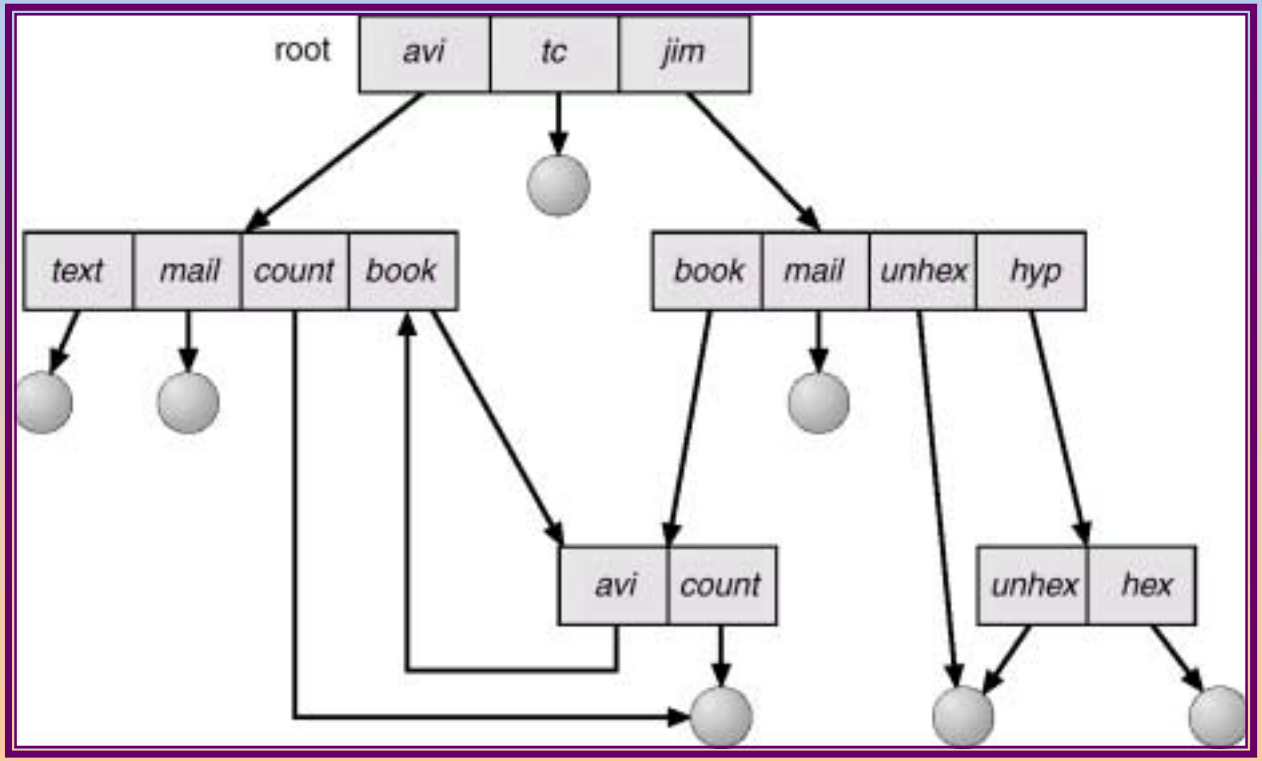
- *Upon **traversal** of file system, do not want to traverse shared structures more than once (e.g., doing backups or accumulating file statistics).*
- *On **deletion**, which action to take?*
  - *Option1: remove file when anyone issues **delete** ➔ possible **dangling pointer** to non-existent file.*
  - *Option2: **[UNIX]** use symbolic links ➔links are left when file is deleted and user has to "realize" that original file is gone.*
  - *Option3: maintain a **file reference list** containing one entry for each reference to the file* {disadvantages – variable and large list}.
  - *Option4: keep a **count** of the number of references. When count=0, file is deleted.*

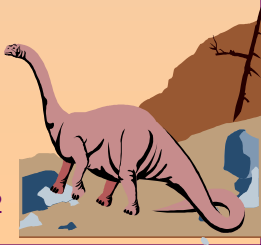


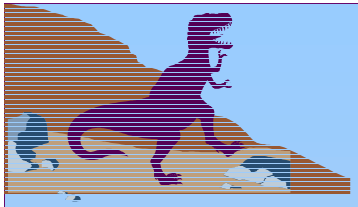


General Graph Directory

- *When **links** are added to an existing tree-structured directory, a **general graph structure** can be created.*

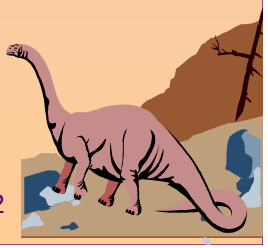


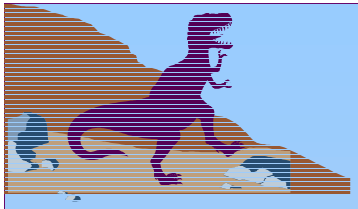




General Graph Directory

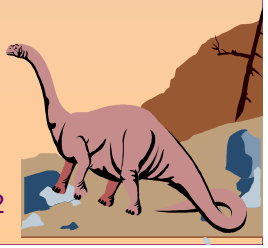
- *A general graph can have cycles and cycles cause problems when searching or traversing file system.*
- How do we guarantee no cycles?
  - Allow only links to files not subdirectories.
  - Use Garbage collection. {computationally expensive}
  - Every time a new link is added, use a cycle detection algorithm to determine whether a cycle now exists. {computationally expensive}
- *An alternative approach – to bypass links during directory traversal.*

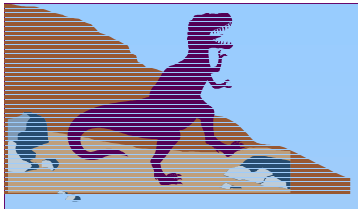




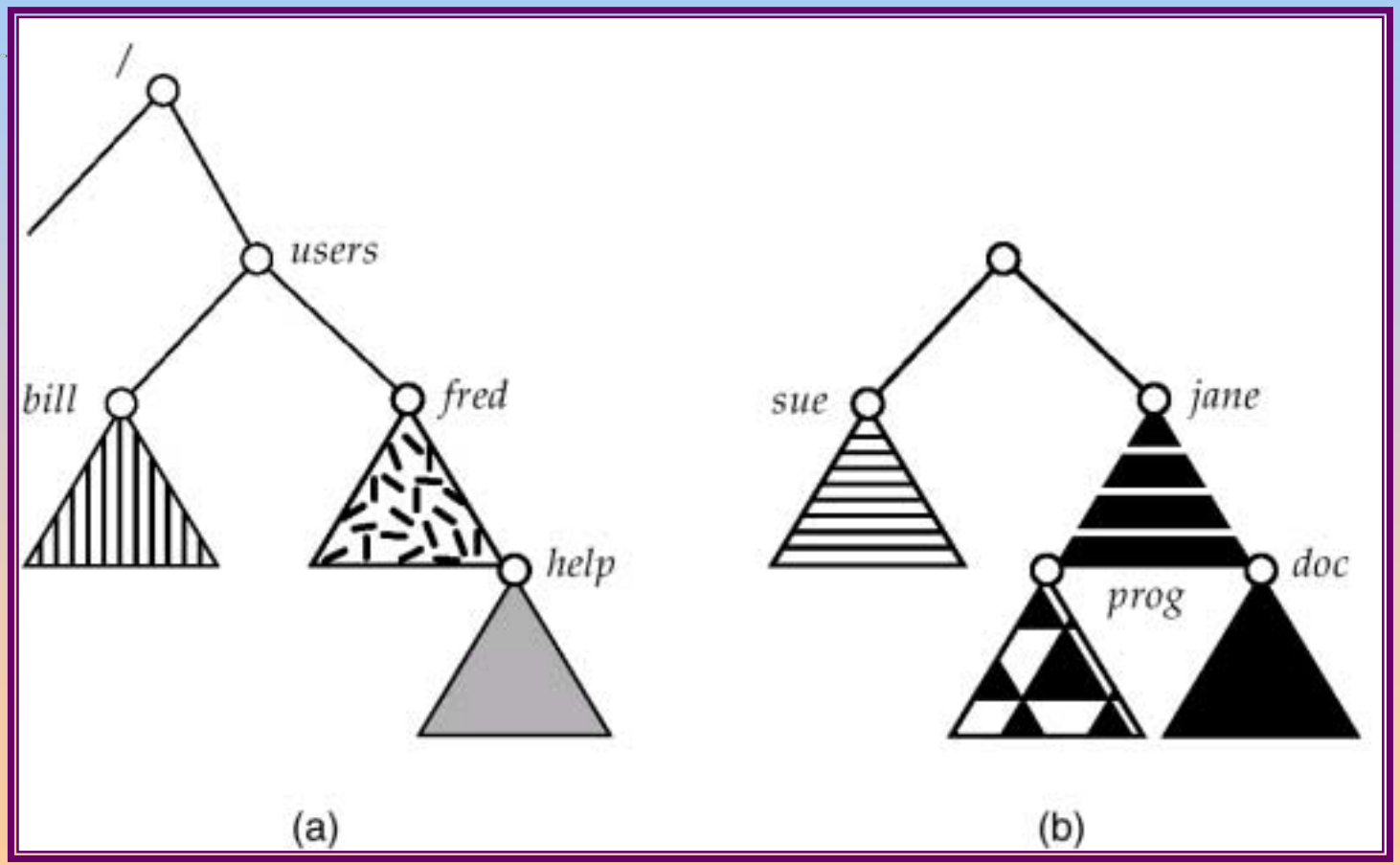
File System Mounting

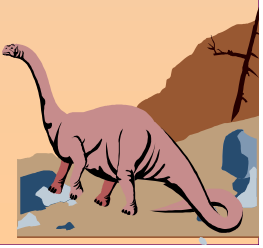
- A file system must be **mounted** before it can be *available to processes on the system*.
- *The mount procedure :: the OS is given the **device name** and the **location within the file structure** at which to attach the the file system. {the mount point}*
- *A mount point is typically an empty directory where the mounted file system will be attached.*
- *The OS verifies that device has valid file system by asking device driver to read the device directory and verify that directory has the proper format.*

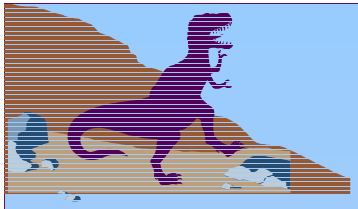




(a) Existing file system.
(b) Unmounted partition residing on /device/dsk

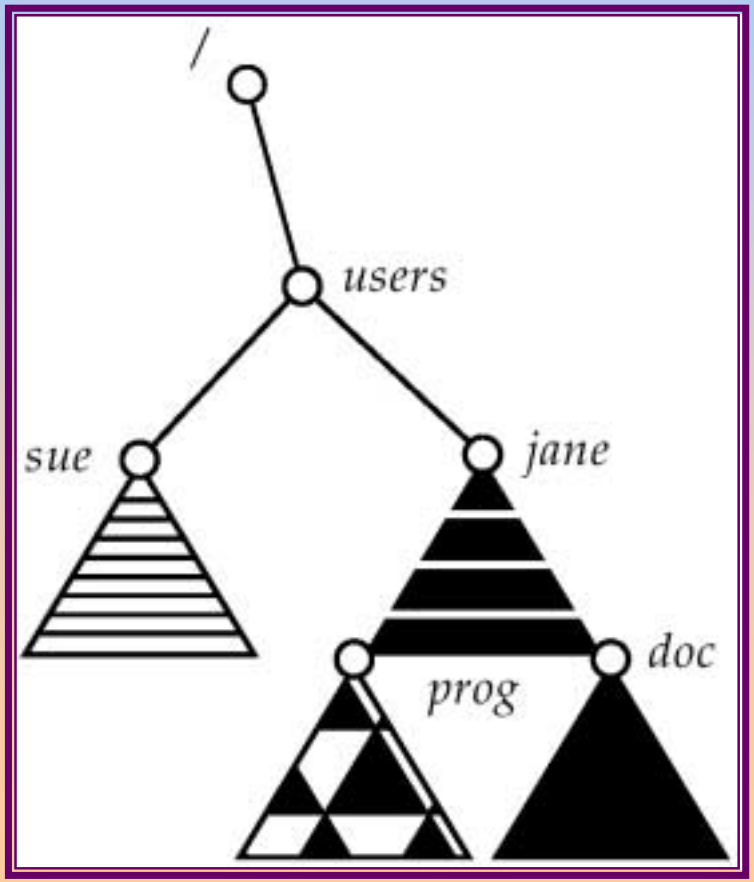


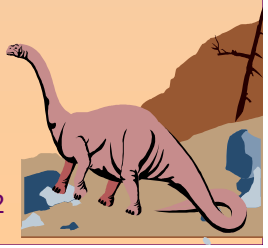


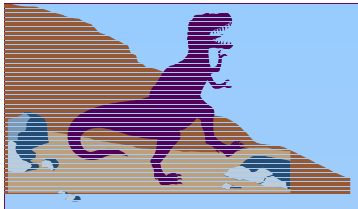


Mount Point

The effect of mounting partition over /users

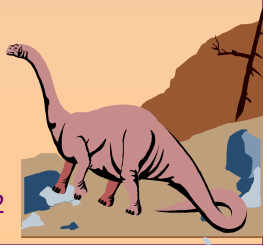


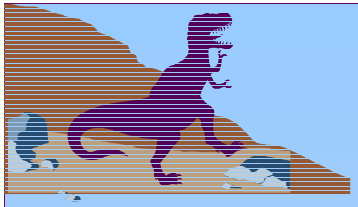




File Sharing

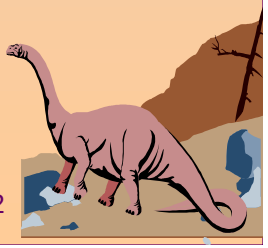
- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a *protection* scheme.
- On distributed systems, files may be shared across a network.
- Network File System (NFS) is a common distributed file-sharing method.

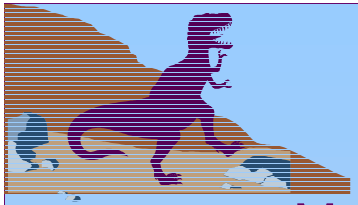




Protection

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



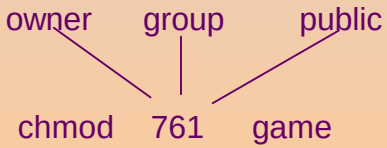


Access Lists and Groups

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

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