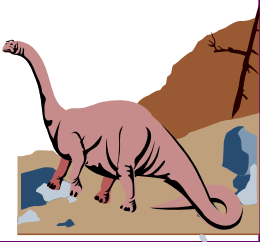
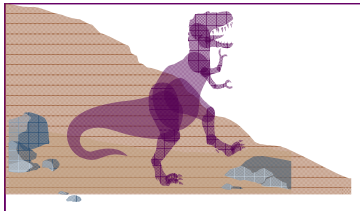
# Chapter 10: File System Interface

肖 卿 俊
办公室：计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页：http://cse.seu.edu.cn/PersonalPage/csqjxiao
电话：025-52091022

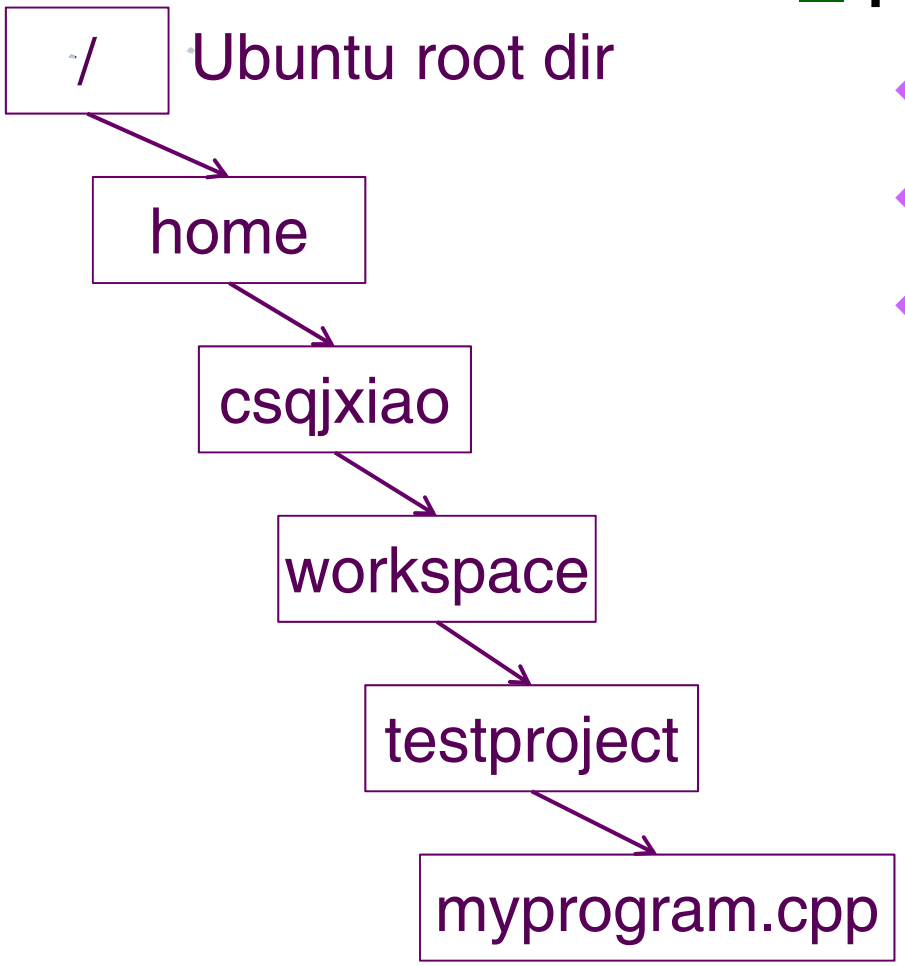
# Chapter 10: File-System Interface

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

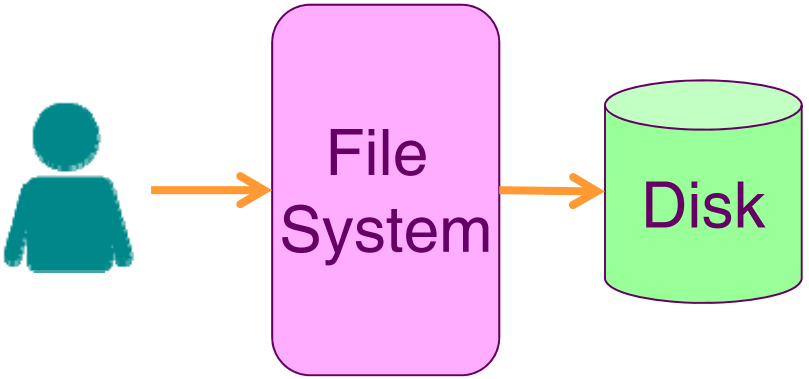




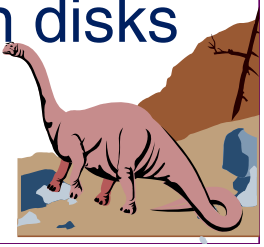
# File System Concept

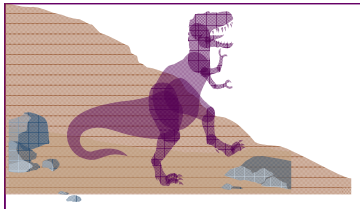



- Key Abstraction
  - File
  - Filename
  - Directory tree (folders)



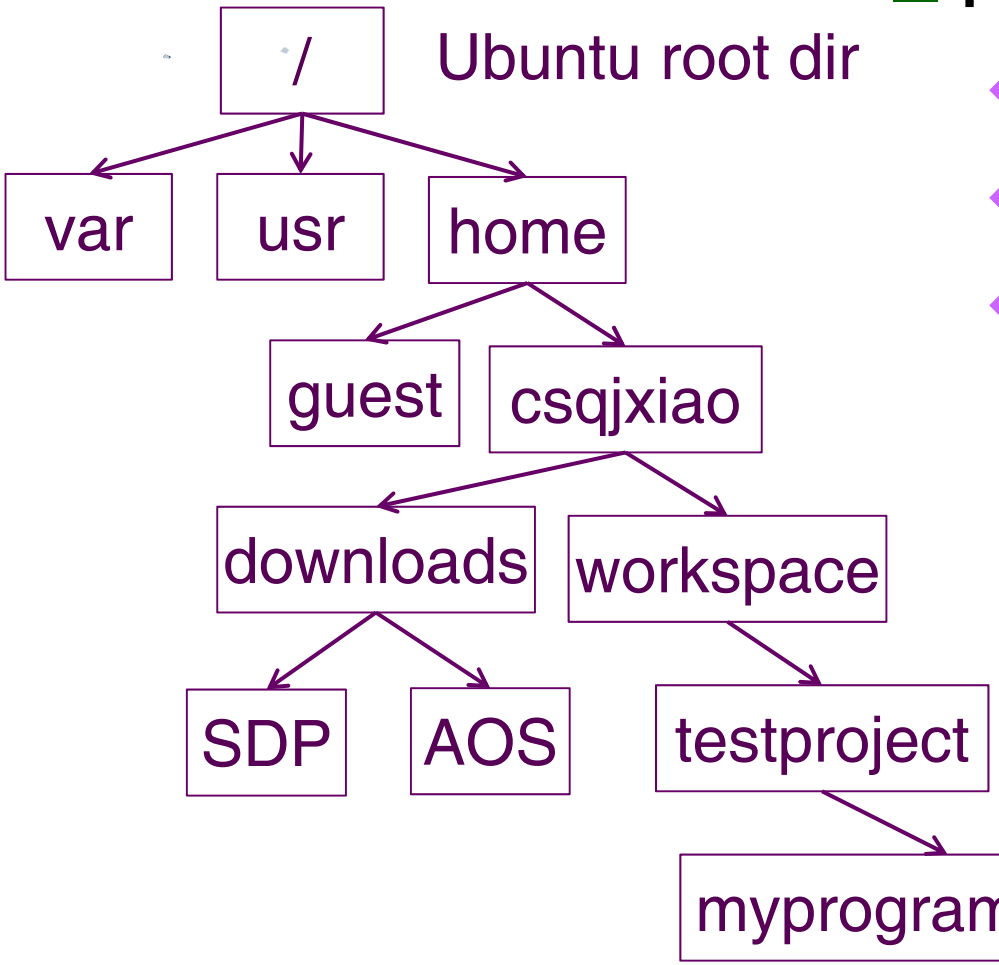

Users do not access directly the file blocks on disks

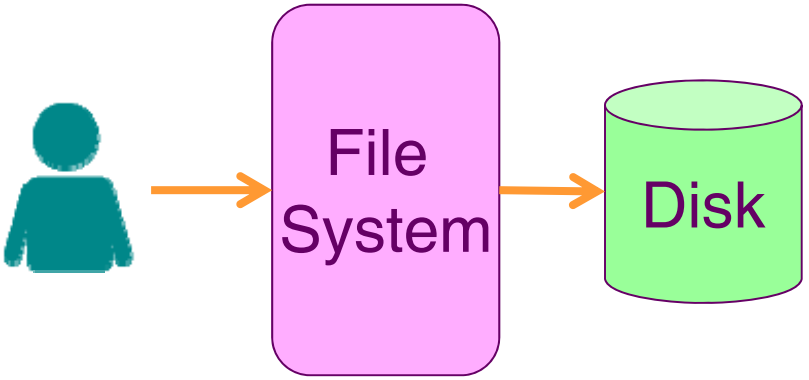




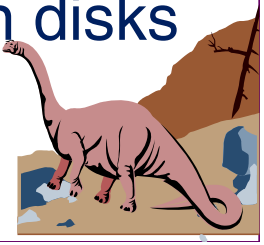
# File Path and Directory Tree

- Key Abstraction
  - File
  - Filename
  - Directory tree (folders)

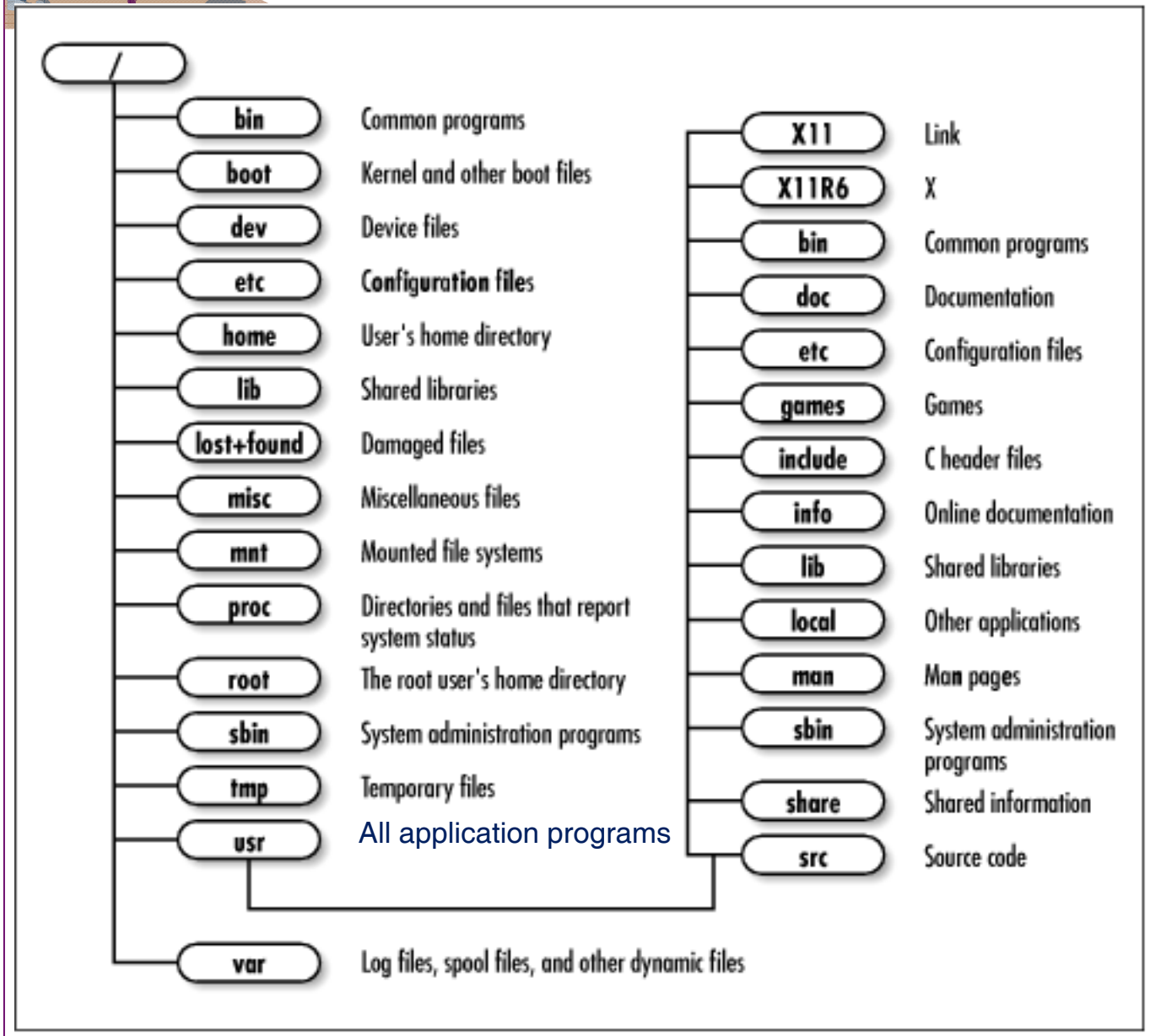



Users do not access directly the file blocks on disks

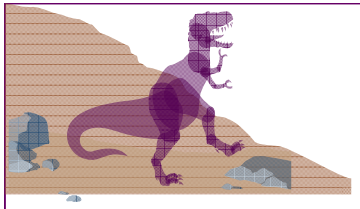


# Debian GNU/Linux Directory Tree



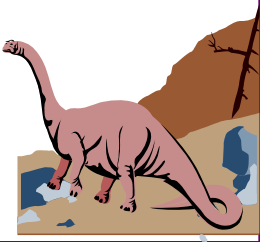

- /home (private): directories of users
- /dev: device files that represent hardware components
- /etc: important files for system configuration
- /bin: programs needed early in the boot process
- /usr: all application programs
- /var: configuration files
- /lib: shared libraries (for dynamically linked programs)

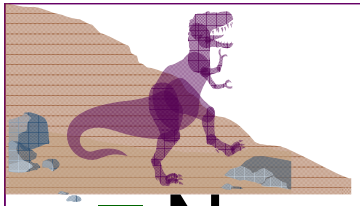
http://www.oreilly.com/openbook/debian/book/appa_01.html



# File Concept

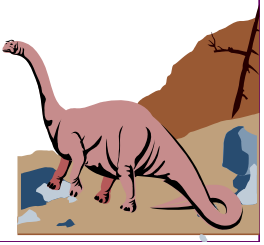
- Contiguous logical address space
- Types:
  - Data
    - ✓numeric
    - ✓character
    - ✓binary
  - Program

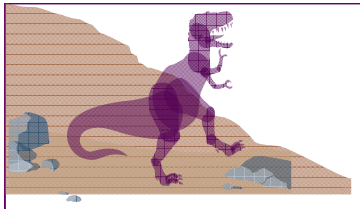




# File Structure

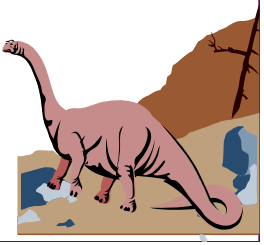
- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides?

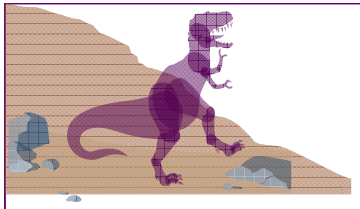




# File Attributes

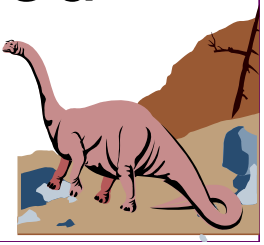
- **Name** – only information kept in human-readable form.
- **Type** – needed for systems that support different types.
- **Location** – pointer to file location on device.
- **Size** – current file size.

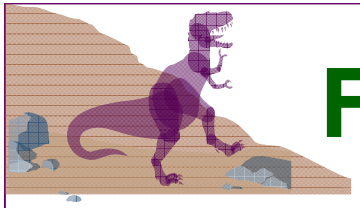




# File Attributes (Cont.)

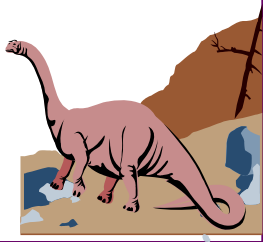
- **Protection** – controls who can do reading, writing, executing.
- **Time**, **date**, **and user identification** – data for protection, security, and usage monitoring.
- All these information about files are kept in the directory structure, which is maintained on the disk.





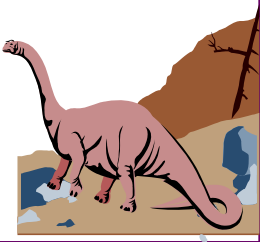
# File Types – Name, Extension

| 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 |



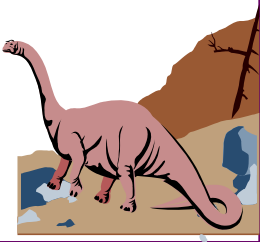
# File Operations from Developer's Perspective

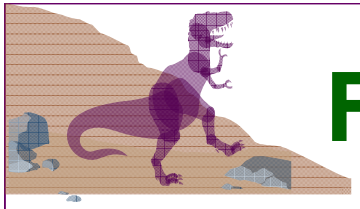
- Create
- Write
- Read
- Reposition within file – file seek
- Delete
- Truncate



# File Operations from Developer's Perspective (cont.)

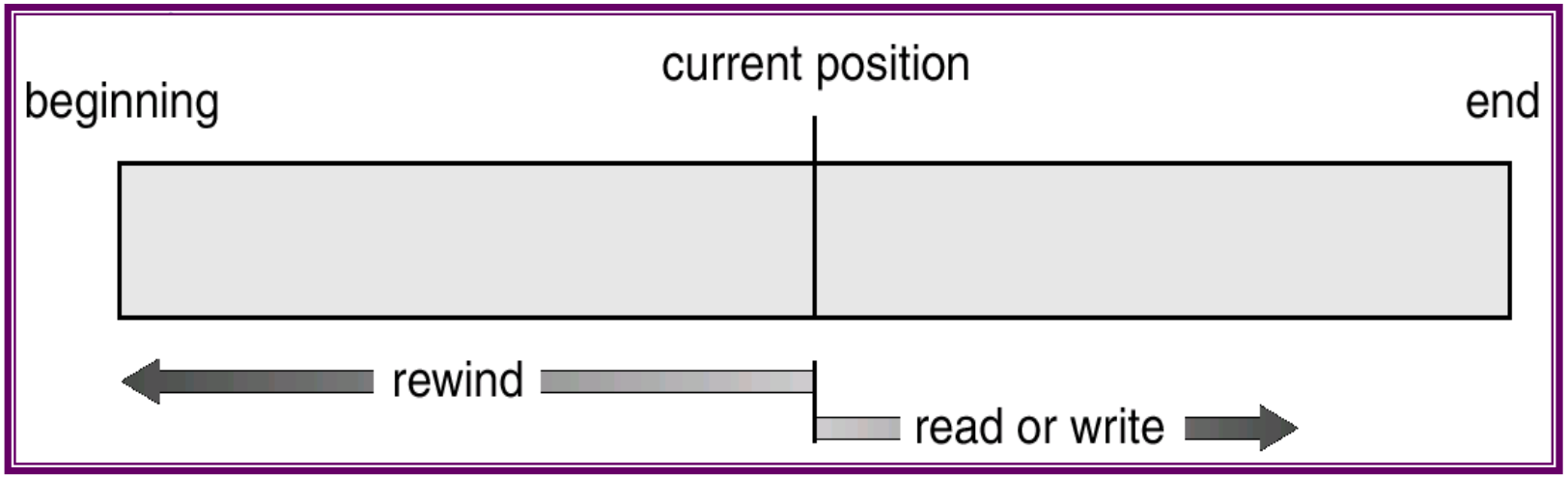
- open($F_i$) – search the directory structure on disk for entry $F_i$, and move the content of entry to memory.
- close($F_i$) – move the content of entry $F_i$ in memory to directory structure on disk.
- read($F_i$) – read the file content
- write($F_i$) – write to the file
- fseek($F_i$) – reposition the file cursor



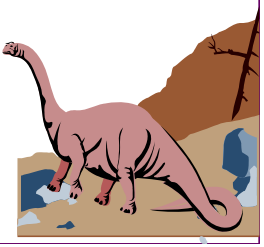


# File Content Access Methods

- **Sequential Access**

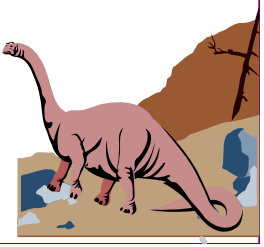



- **Direct Access**



# Simulation of Sequential Access on a Direct-Access File

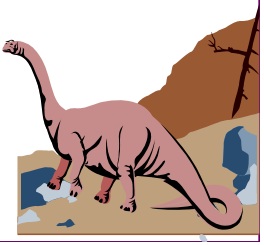
| sequential access | implementation for direct access |
|---|---|
| *reset* | *cp* = 0; |
| *read next* | *read cp*;<br>*cp* = *cp*+1; |
| *write next* | *write cp*;<br>*cp* = *cp*+1; |

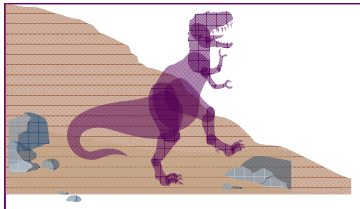


# Code Modifying a Key-Value Pair

| KEY | VALUE |
|---|---|
| integer | integer |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

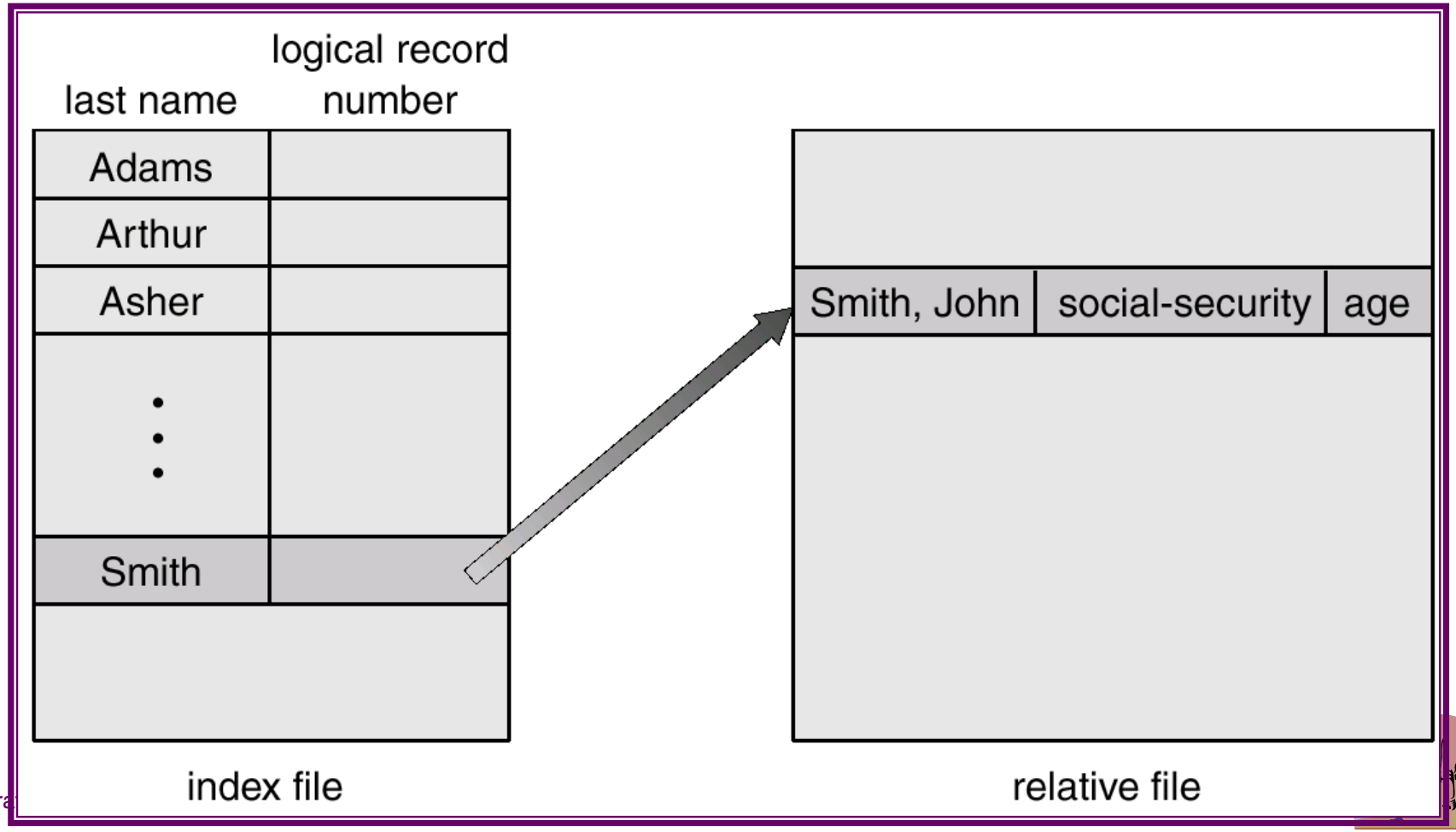
```
ssize_t len;
char * filename;
int key, srch_key, new_value;
filename = argv[1];
srch_key = strtol(argv[2], NULL, 10);
new_value = strtol(argv[3], NULL, 10);
int fd = open(filename, O_RDWR);
while(sizeof(int) == read(fd, key, sizeof(int))) {
        if(key != srch_key)
                lseek(fd, sizeof(int), SEEK_CUR);
        else {
                write(fd, &new_value, sizeof(int));
                close(fd);
                return EXIT_SUCCESS;
        }
}
fprintf(stderr, “key not found!”);
return EXIT_FAILURE;
```



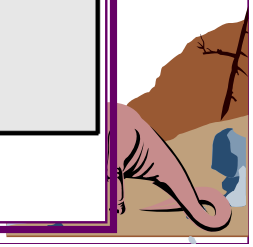


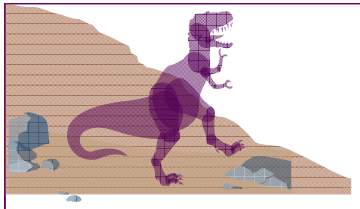
# Example of Index and Relative Files

- Store keys in the index file
- Store values in the relative file

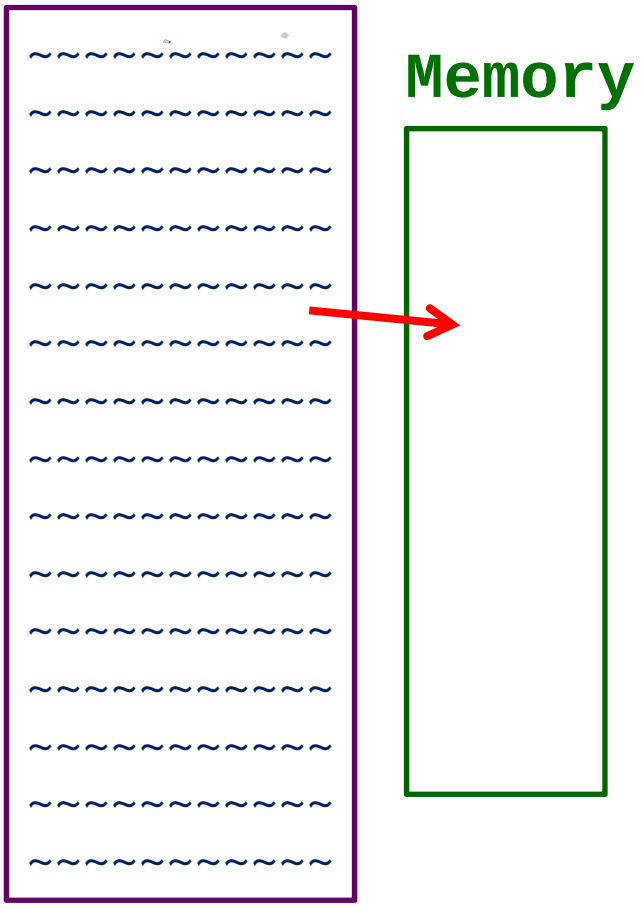






# File Content Direct Access by Memory Mapped File



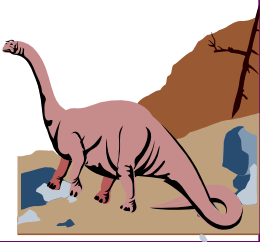

- **mmap**() creates a new mapping in the virtual address space of the calling process
- **munmap**() system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references

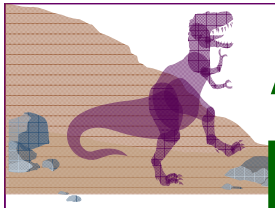
```
fd = open("file.txt", ….);
buffer = mmap(…, fd, …);

// manipulate the buffer

munmap(buf, …);
close(fd);
```

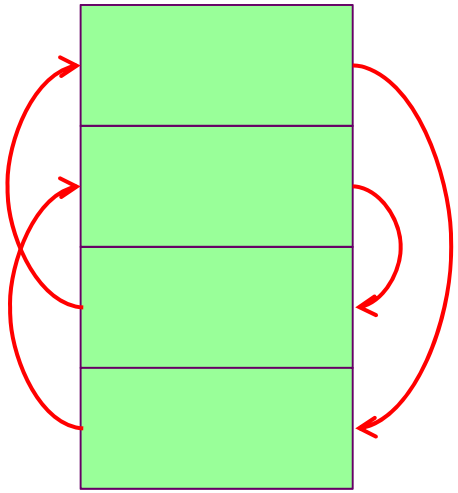
http://linux.die.net/man/2/mmap

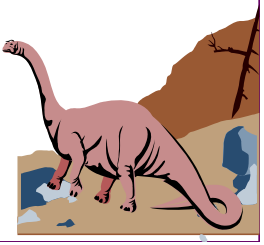


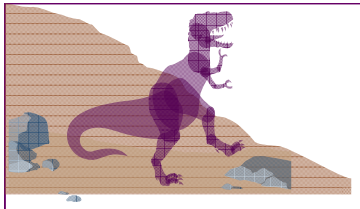


# An Example of Memory Mapped File: Shuffle Blocks within a File

```
filename = argv[1];
card_size = strtol(argv[2], NULL, 10);
fd = open(filename, O_RDWR);
len = lseek(fd, 0, SEEK_END);
lseek(fd, 0, SEEK_SET);
buf = mmap(NULL, len, PROT_READ |
PROT_WRITE, MAP_FILE | MAP_SHARED, fd, 0);
if( buf == (void*) -1) {
      fprintf(stderr, “mmap failed.\n”);
      exit(EXIT_FAILURE);
}
memshuffle(buf, len, card_size);
munmap(buf, len);
close(fd);
return EXIT_SUCCESS;
```

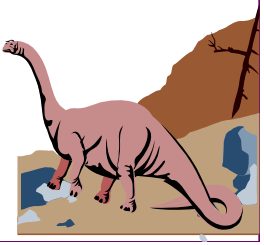


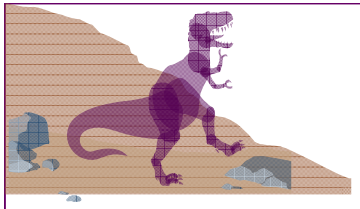




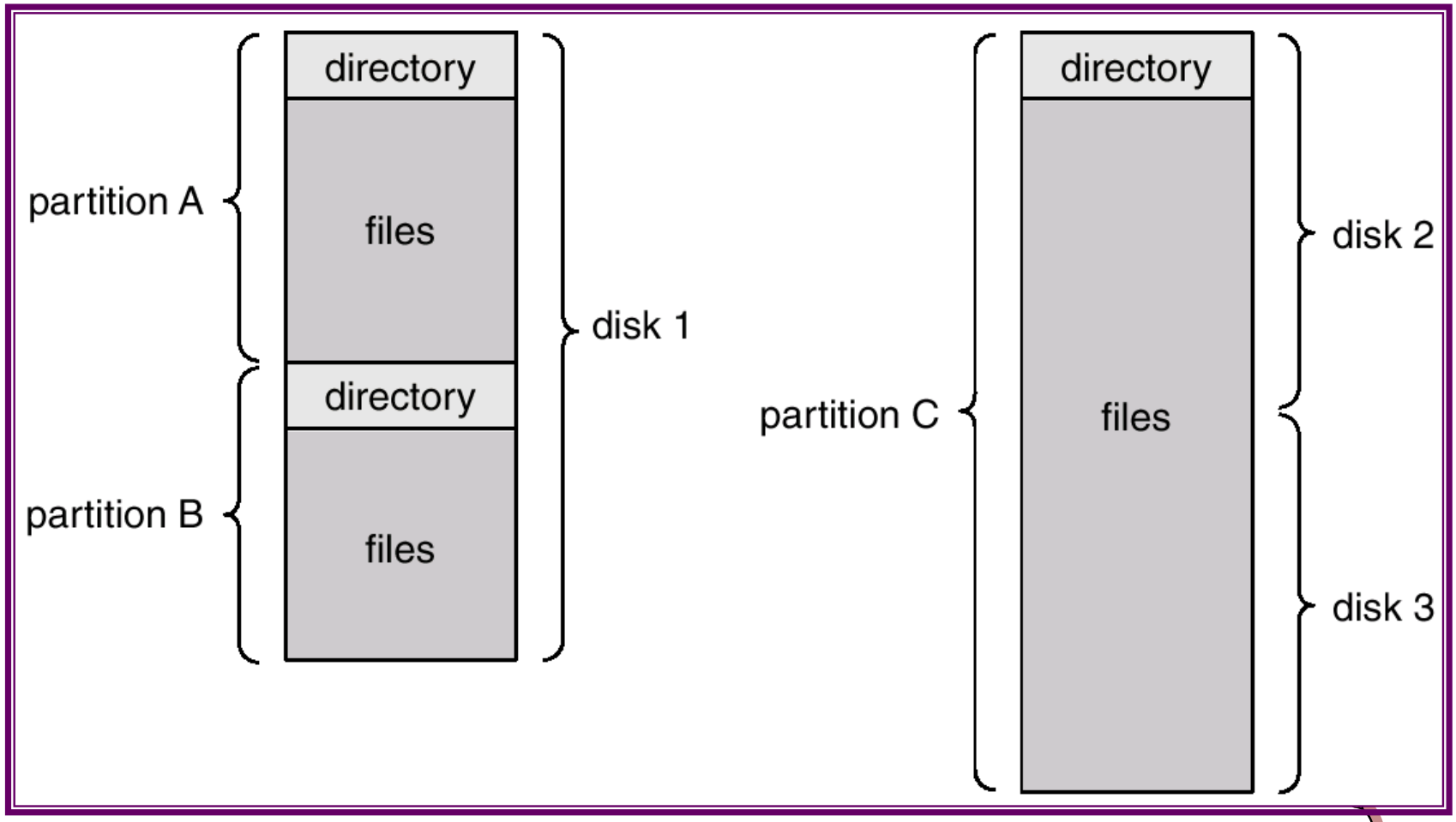
# Directory Structure

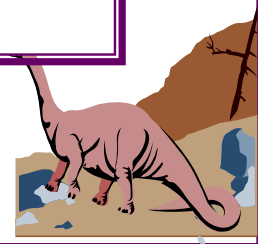
- disks are split into one or more partitions.
- each partition contains information about files within it
- The information is kept in entries in a device directory or volume table of contents

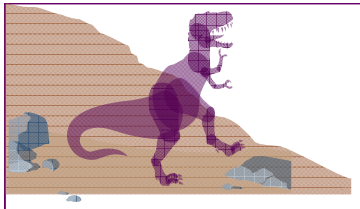




# A Typical File-system Organization

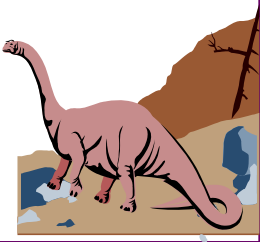




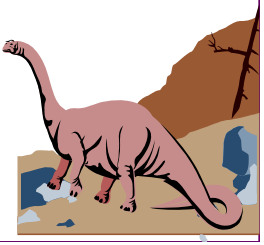
# Operations Performed on Directory

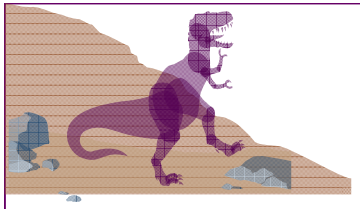
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system



# Organize the Directory (Logically) to Obtain

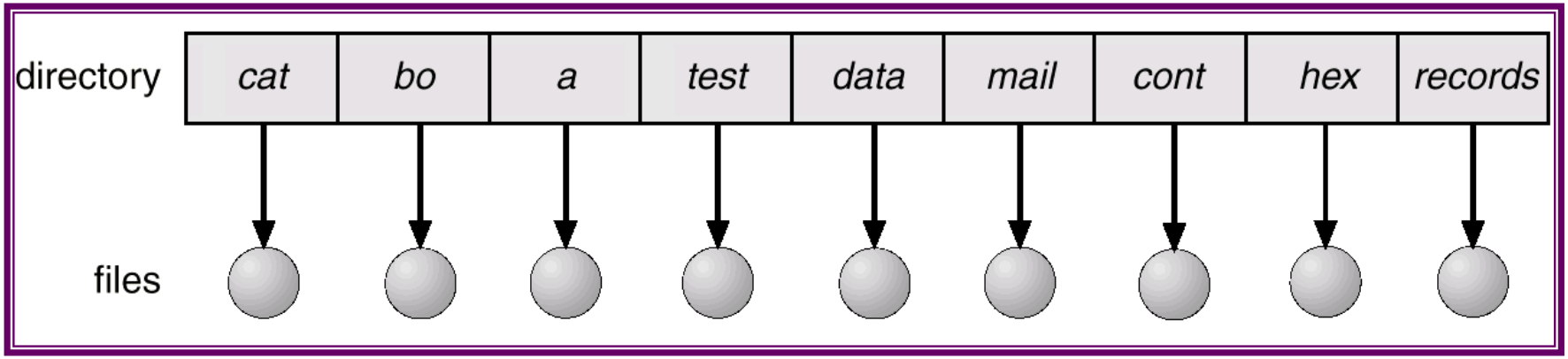
- **Efficiency** – locating a file quickly.
- **Naming** – convenient to users.
  - Two users can have same name for different files.
  - The same file can have several different names.
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)





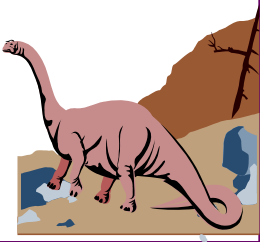
# Single-Level Directory

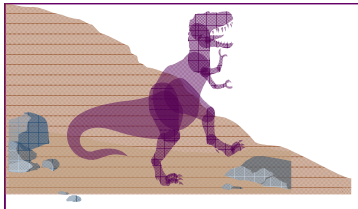
- A single directory for all users.




Naming problem

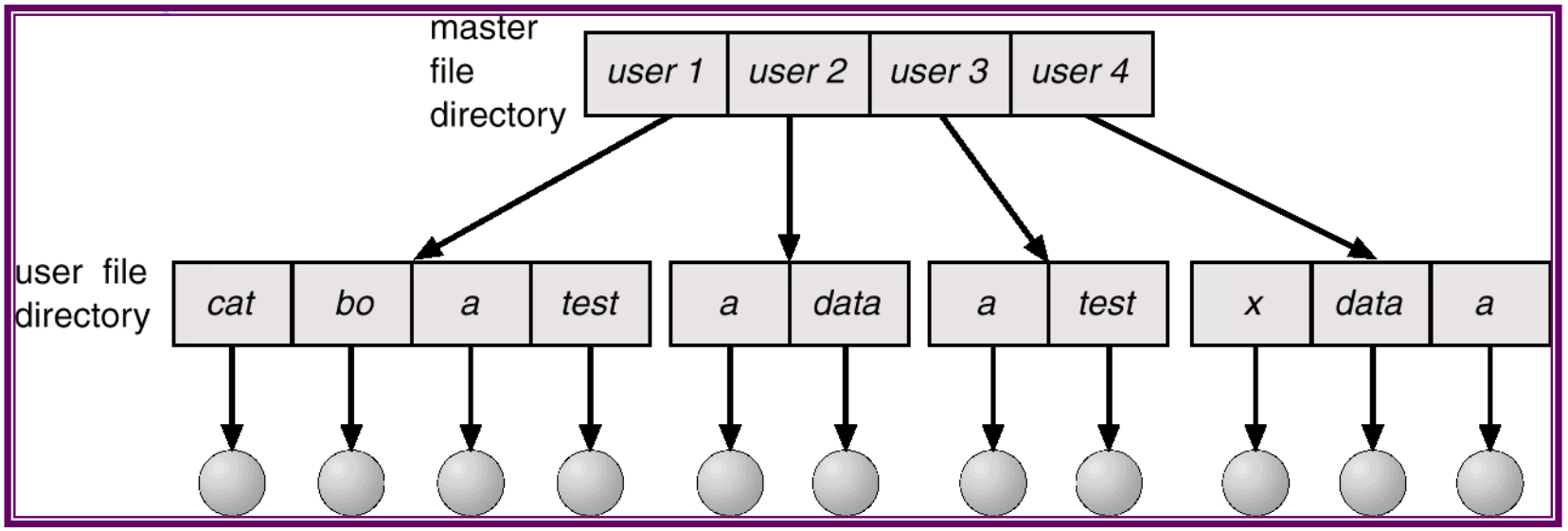
Grouping problem



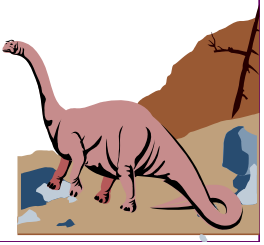


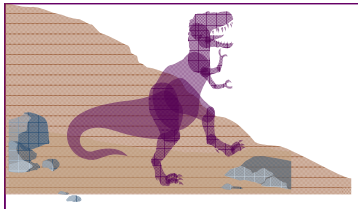
# Two-Level Directory

- Separate directory for each user.

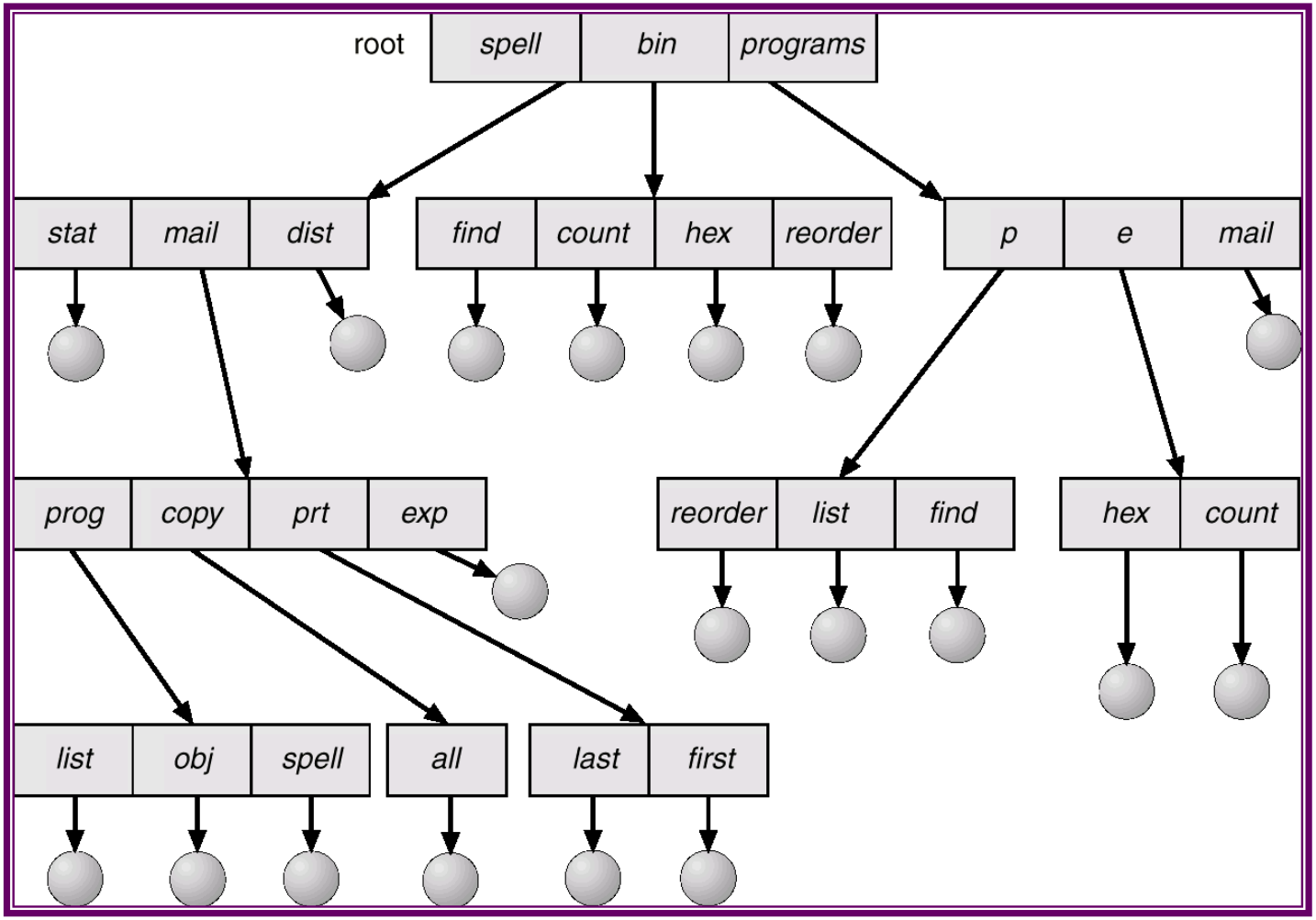


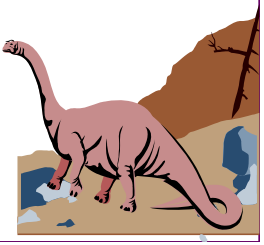

- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

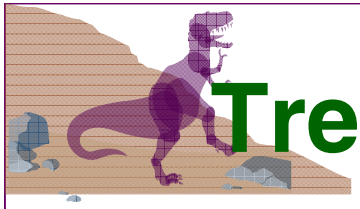




# Tree-Structured Directories

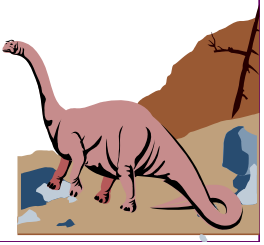


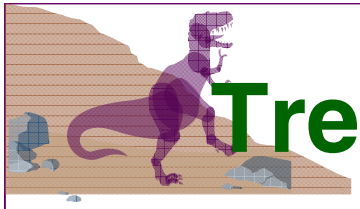





# Tree-Structured Directories (cont.)

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

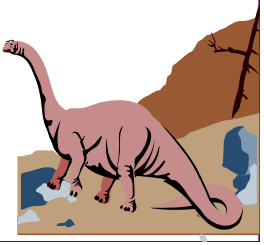


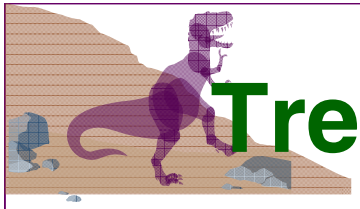


# Tree-Structured Directories (cont.)

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

  **rm** <file-name>





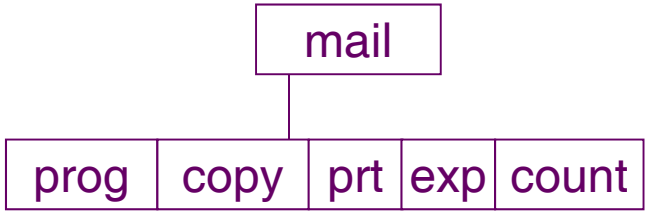
# Tree-Structured Directories (cont.)

- Creating a new subdirectory is done in current directory.

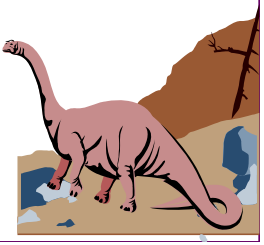
  **mkdir** <dir-name>

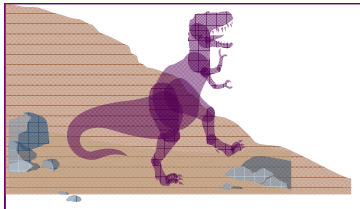
  Example: if in current directory **/mail**

  **mkdir** count



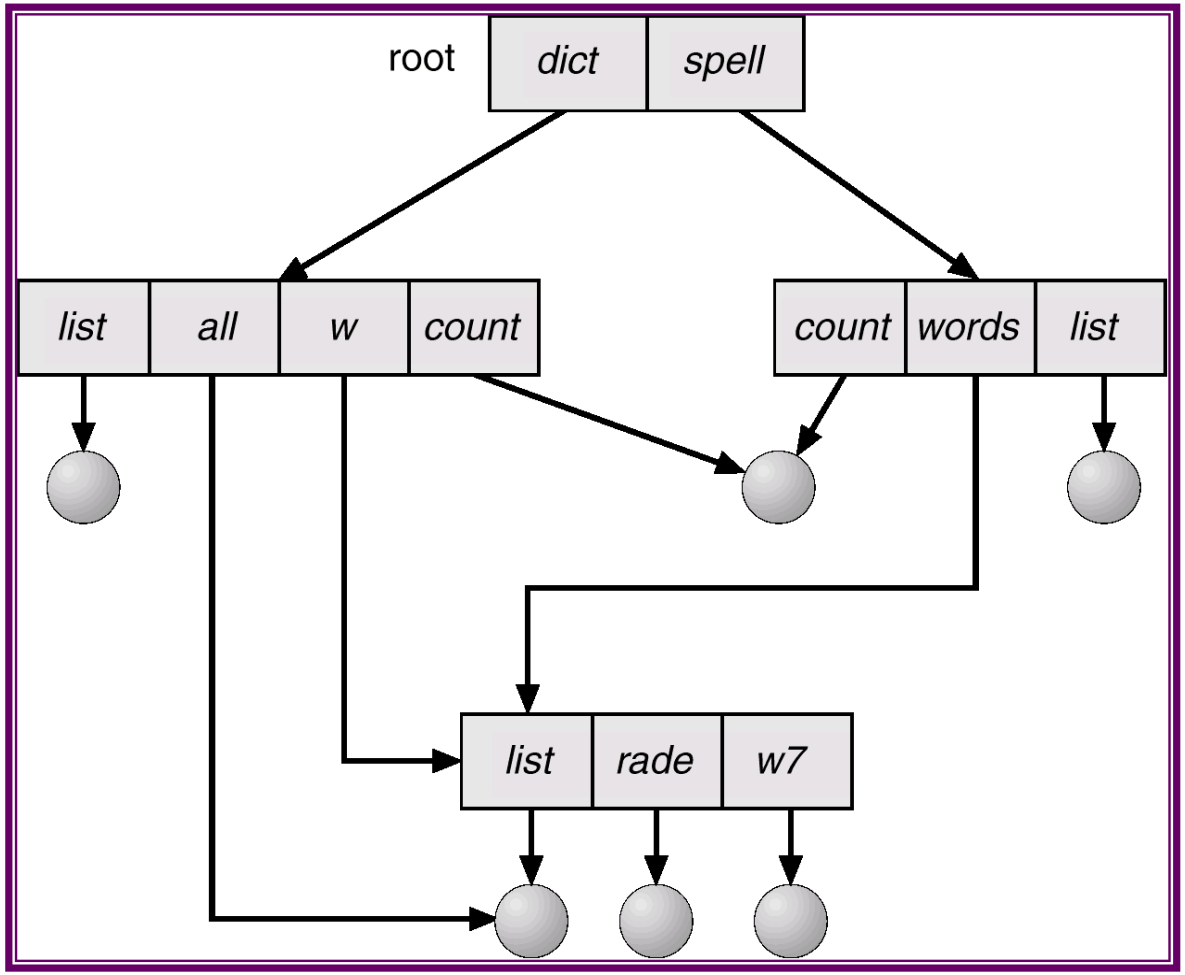

Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.

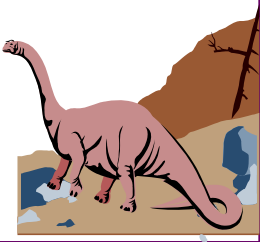


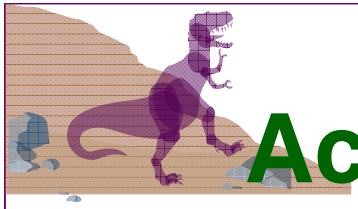


# Acyclic-Graph Directories

- Have shared subdirectories and files.



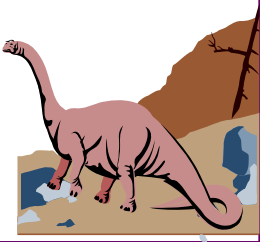



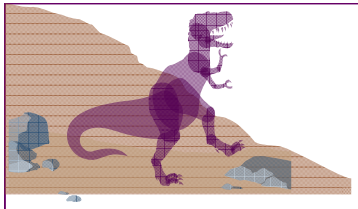


# Acyclic-Graph Directories (cont.)

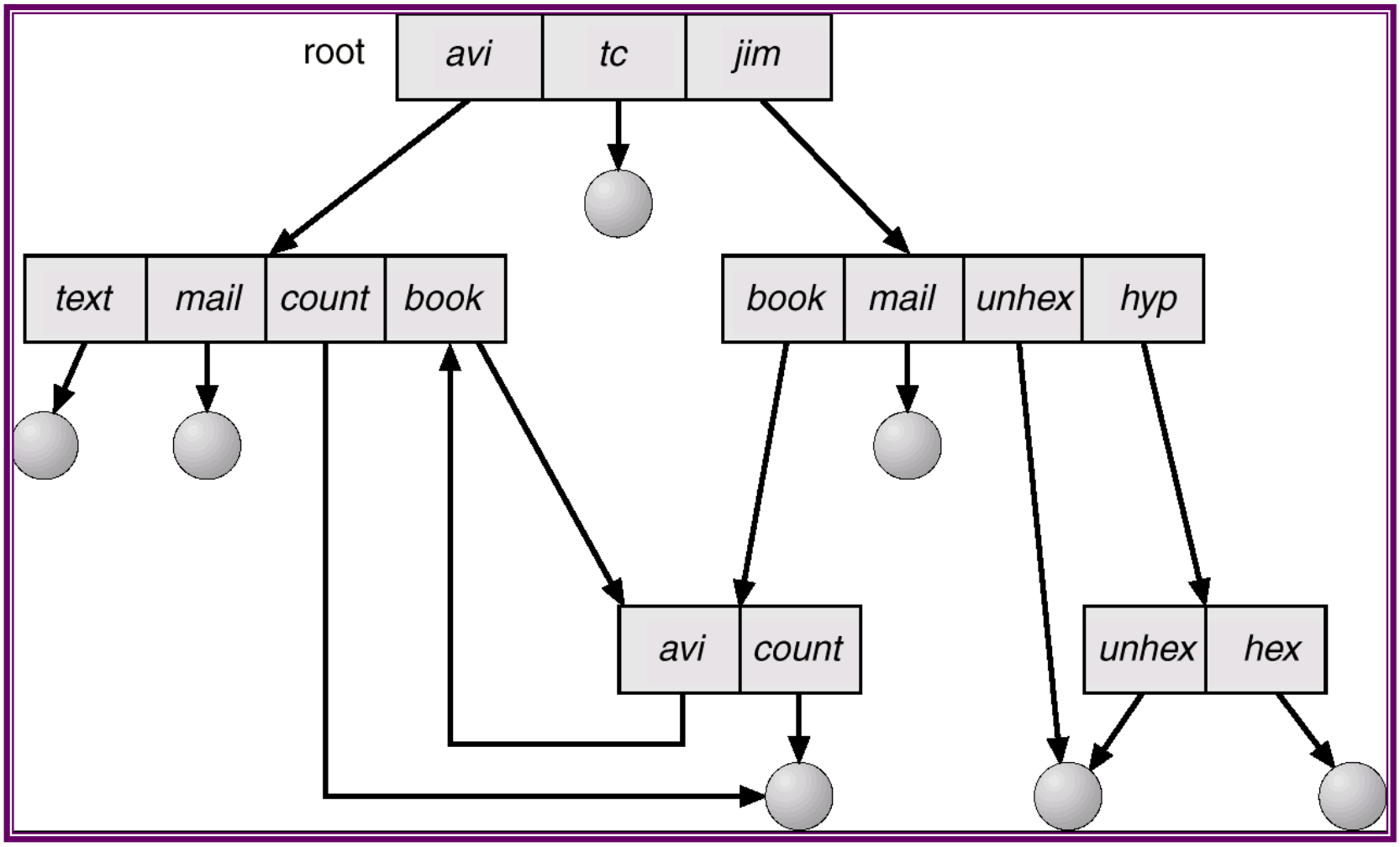
- Two different names (aliasing)
- If *dict* deletes *count* $\Rightarrow$ dangling pointer.

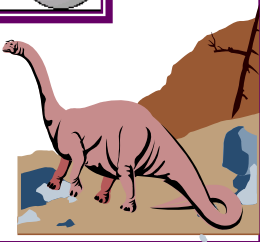
  Solutions:
  - Backpointers, so we can delete all pointers.
  - Entry-hold-count solution.

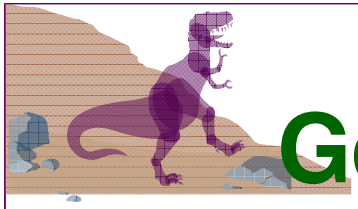




# General Graph Directory

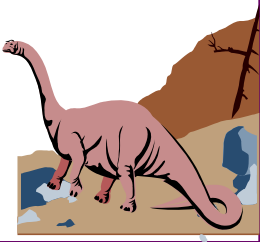


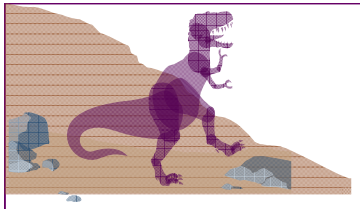





# General Graph Directory (cont.)

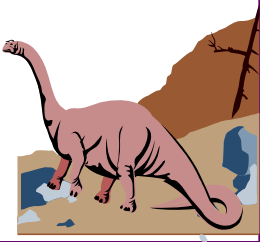
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories.
  - Garbage collection.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

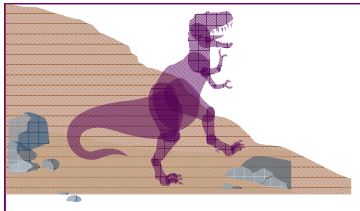




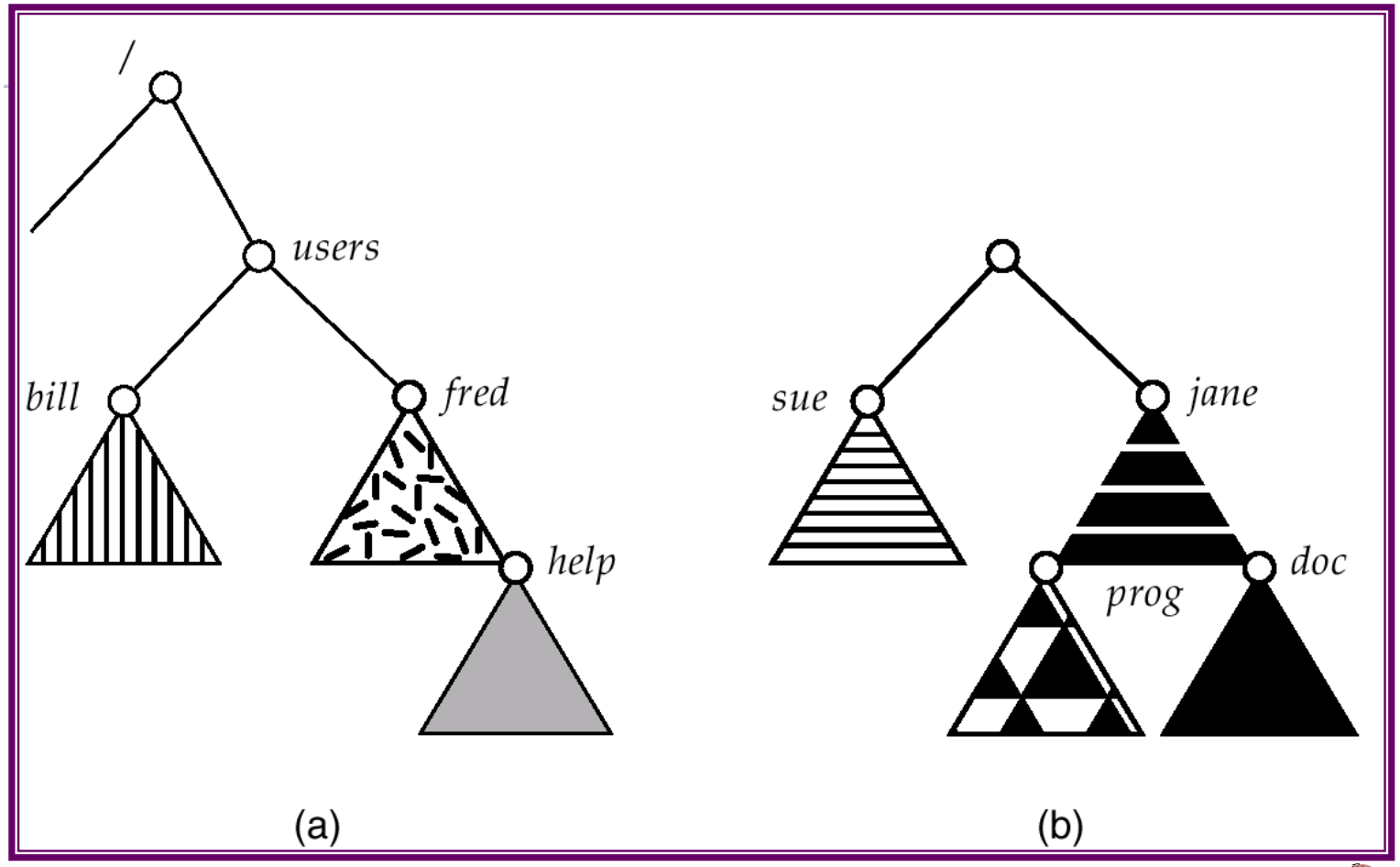
# File System Mounting

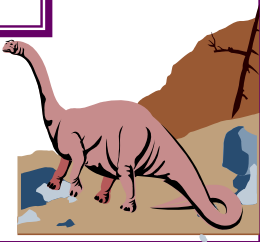
- A file system must be **mounted** before it can be accessed.
- An unmounted file system (I.e. Fig. 11-11(b)) is mounted at a **mount point**.

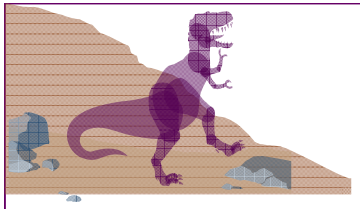




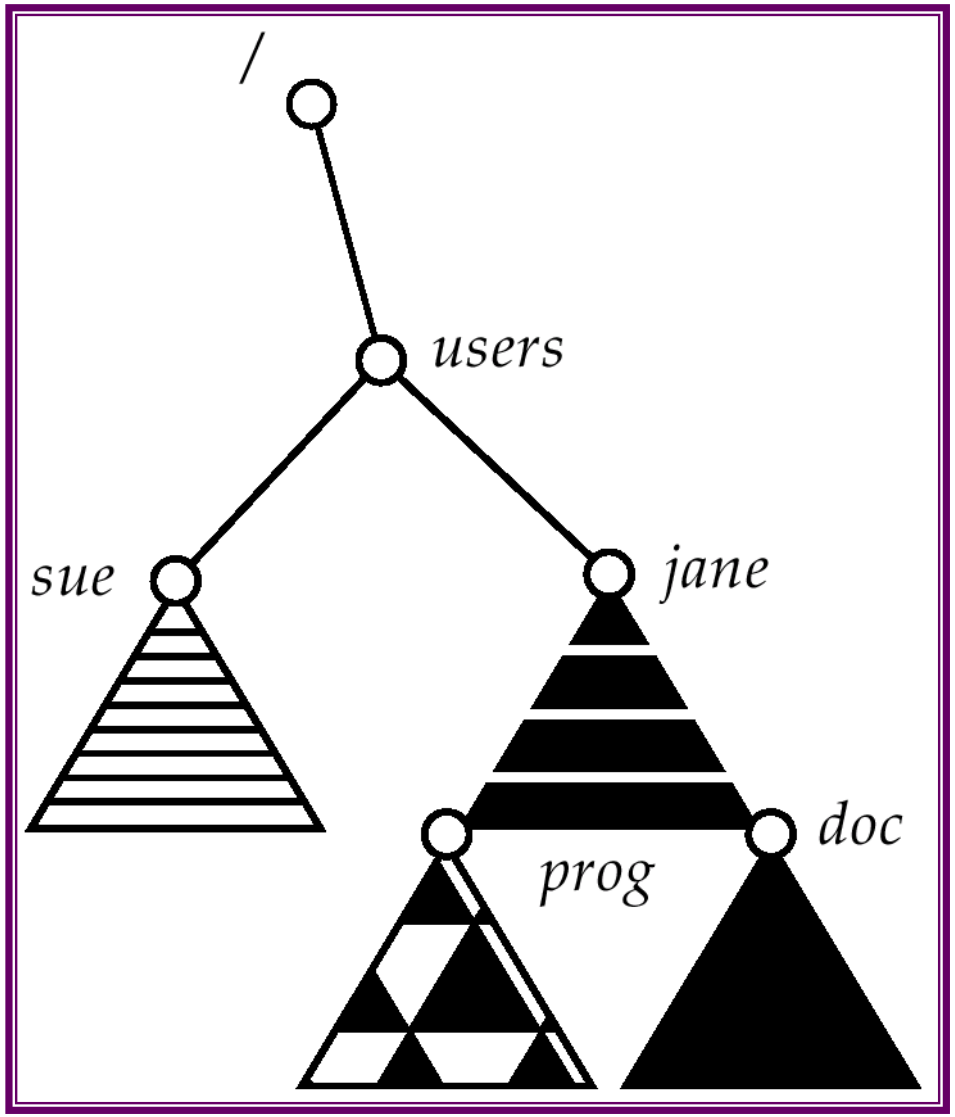
# (a) Existing
# (b) Unmounted Partition

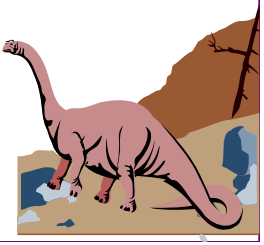


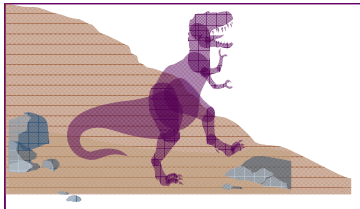





# Mount Point

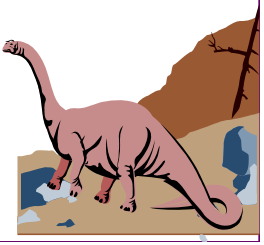


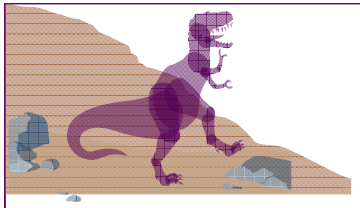





# File Sharing

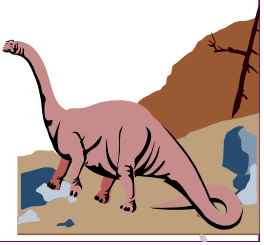
- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a *protection* scheme.
- On distributed systems, files may be shared across a network.
- Network File System (NFS) is a common distributed file-sharing method.

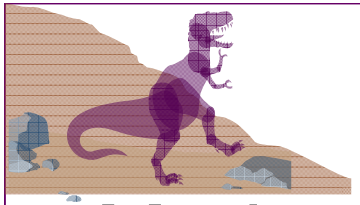




# Protection

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





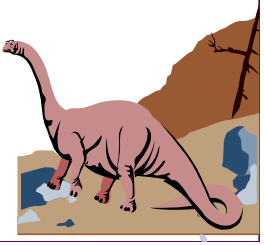
# Access Lists and Groups

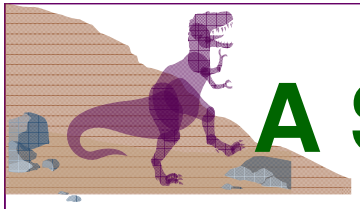
- Mode of access: read, write, execute
- Three classes of users RWX
  - a) **owner access** 7 ⇒ 1 1 1
  - b) **group access** 6 ⇒ 1 1 0
  - c) **public access** 1 ⇒ 0 0 1
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

Attach a group to a file chmod 761 game

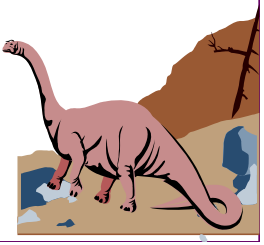
chgrp G game

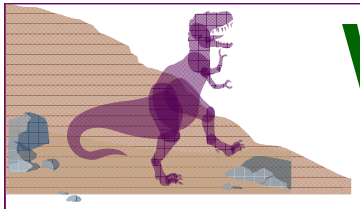




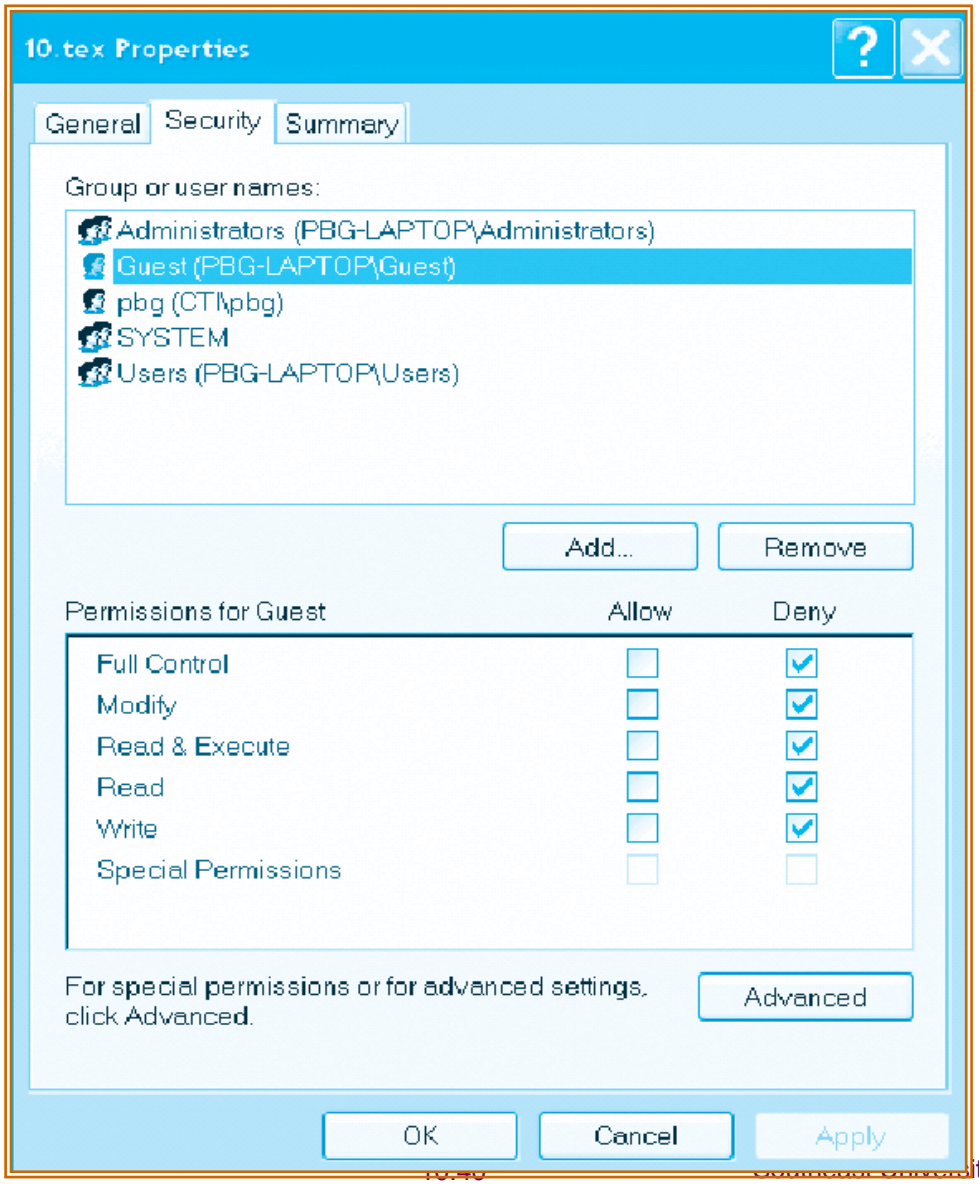
# A Sample UNIX Directory Listing

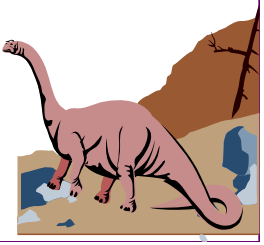
| | | | | | |
|---|---|---|---|---|---|
| -rw-rw-r-- | 1 pbg | staff | 31200 | Sep 3 08:30 | intro.ps |
| drwx------ | 5 pbg | staff | 512 | Jul 8 09.33 | private/ |
| drwxrwxr-x | 2 pbg | staff | 512 | Jul 8 09:35 | doc/ |
| drwxrwx--- | 2 pbg | student | 512 | Aug 3 14:13 | student-proj/ |
| -rw-r--r--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





# Windows XP Access-Control List Management



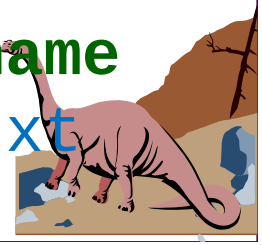



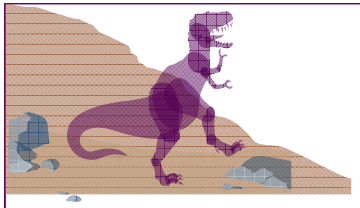
# Question about File Access-Control

- Which of the following will generate a permission error?
  - cat foo.txt
  - cat dir/bar.txt
  - touch dir/new.txt

```
$ ls -l ./
Permission      user  group     ….   Filename
drw-r--r--      me    me              dir
-rw-r--r--      other other           foo.txt

$ sudo ls -l dir
Permission      user  group     ….   Filename
-rw-r--r--      me    me              bar.txt
```





# Another Question

- Which of the following will generate a permission error?
  - cat foo.txt
  - cat dir/bar.txt
  - touch dir/new.txt

```
$ ls -l ./
Permission      user  group     ….   Filename
d--xr--r--      me    me              dir
-rw-r--r--      other other           foo.txt

$ sudo ls -l dir
Permission      user  group     ….   Filename
-rw-r--r--      me    me              bar.txt
```