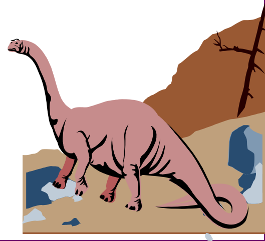
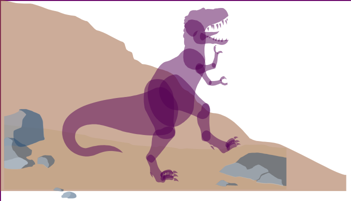
Chapter 10: File System Interface

肖 卿 俊
办公室：九龙湖校区计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页： https://csqjxiao.github.io/PersonalPage
电话：025-52091022

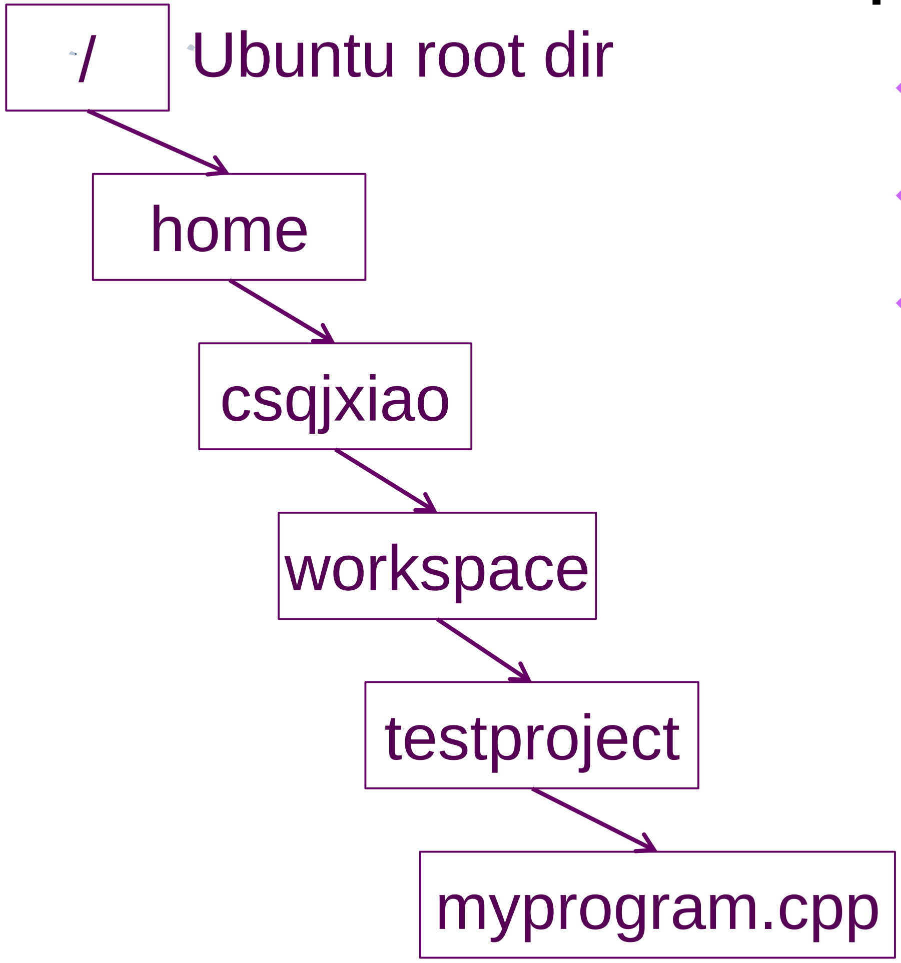
Chapter 10: File-System Interface

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

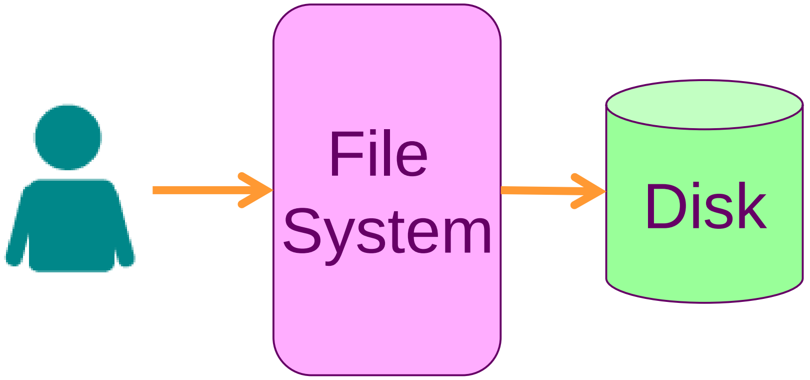




File System Concept

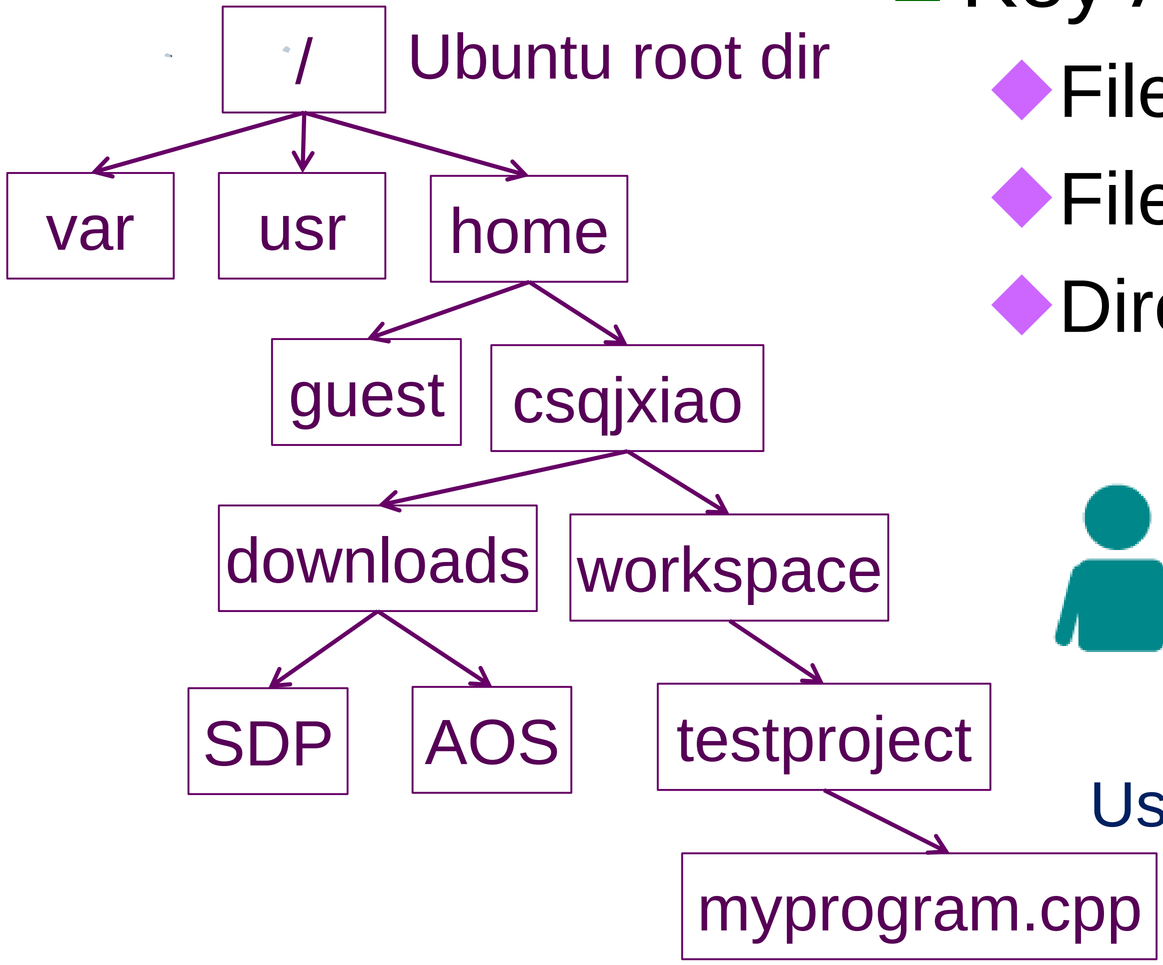


- Key Abstraction
  - File
  - Filename
  - Directory tree (folders)

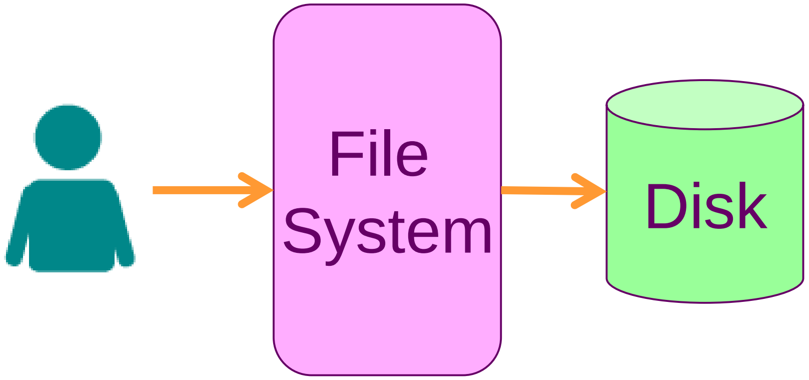


Users do not access directly the file blocks on disks

File Path and Directory Tree

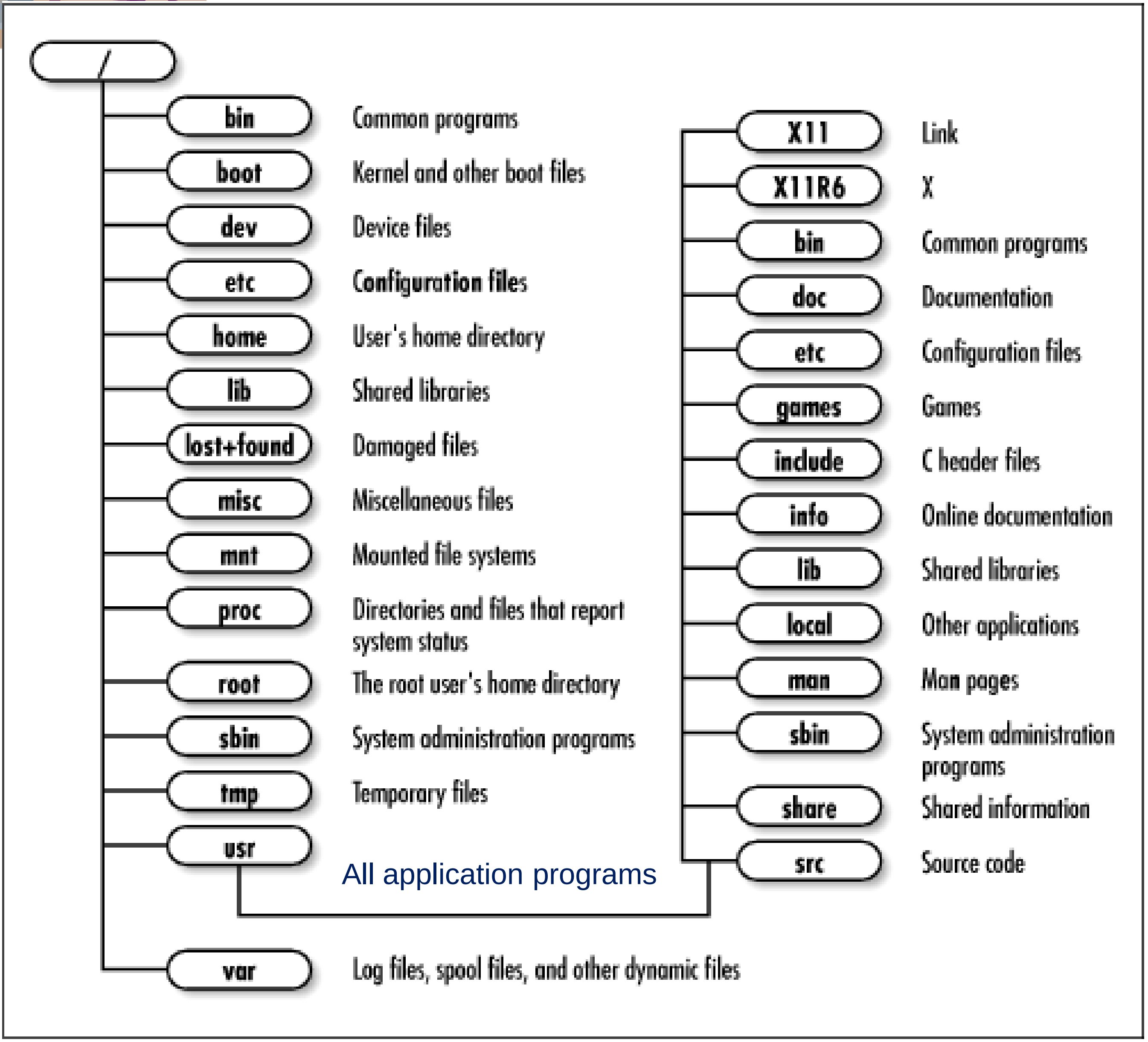


- Key Abstraction
  - File
  - Filename
  - Directory tree (folders)



Users do not access directly the file blocks on disks

Debian GNU/Linux Directory Tree

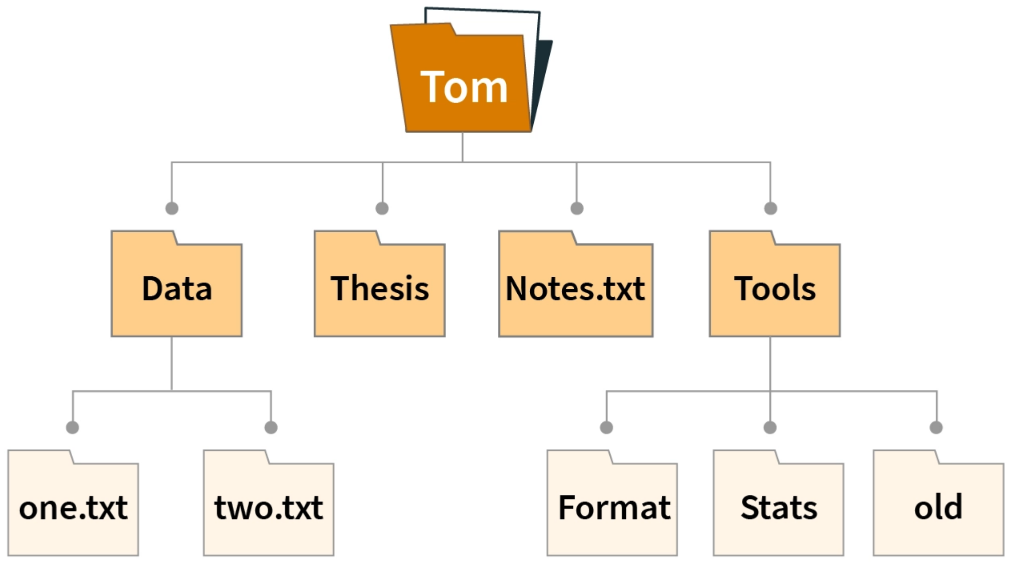


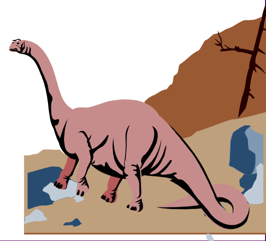
- /home (private): directories of users
- /dev: device files that represent hardware components
- /etc: important files for system configuration
- /bin: programs needed early in the boot process
- /usr: all application programs
- /var: log files, and other dynamic files
- /lib: shared libraries (for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

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

File Concept

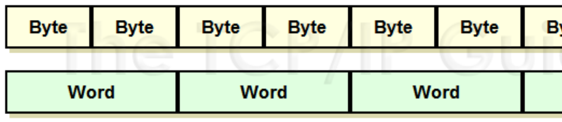
- Contiguous logical address space
- Types:
  - Data
    - ✓numeric
    - ✓character
    - ✓binary
  - Program



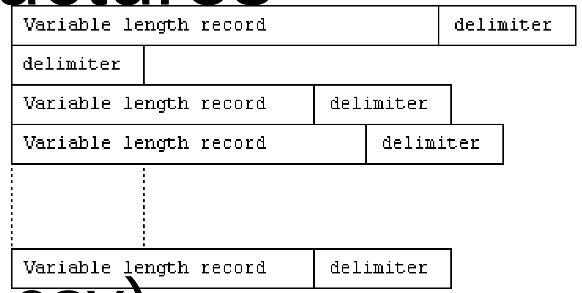


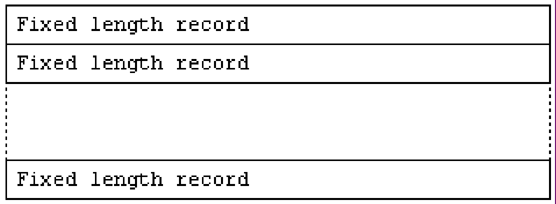
File Structure

- None - sequence of words, bytes

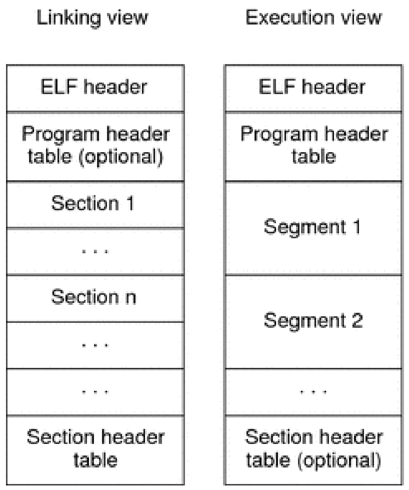


- Simple record structures
  - Lines (.txt)
  - Fixed length
  - Variable length (.csv)





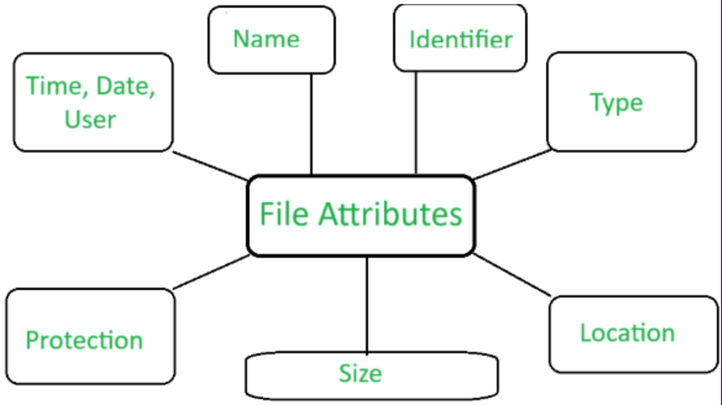
- Complex structures
  - Formatted document (.docx, .tex)
  - Relocatable load file (.obj, .so)



- Can simulate last two categories of structures with the first method, by inserting appropriate control characters

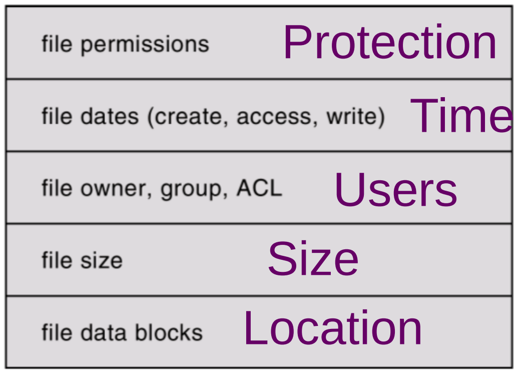
File Attributes (File Control Block, inode)

- **Name** – only information kept in human-readable form.
- **Type** – needed for systems that support different types.



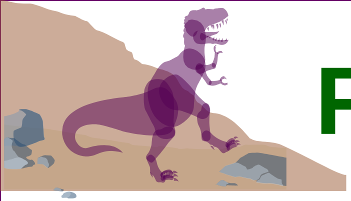
- **Location** – pointer to file location on device.
- **Size** – current file size.

File Control Block (inode)

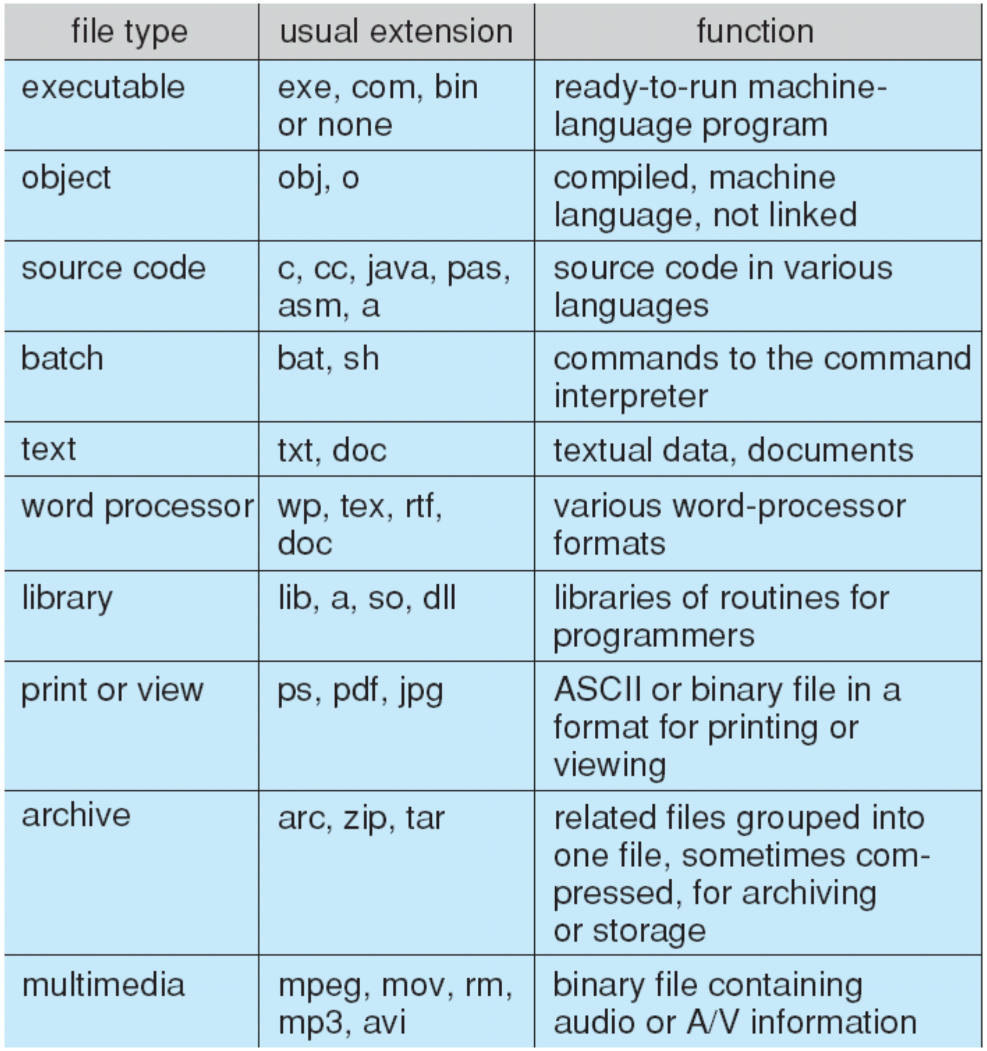


file permissions	Protection
file dates (create, access, write)	Time
file owner, group, ACL	Users
file size	Size
file data blocks	Location

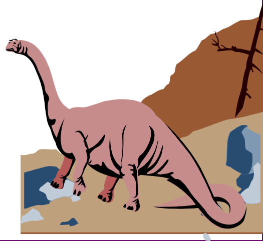
- **Protection** – controls who can do reading, writing, executing.
- **Time**, **date**, **and user identification** – data for protection, security, and usage monitoring.
- All these information of files are kept in directory structure, which is maintained on the disk.



File Types – Name, Extension

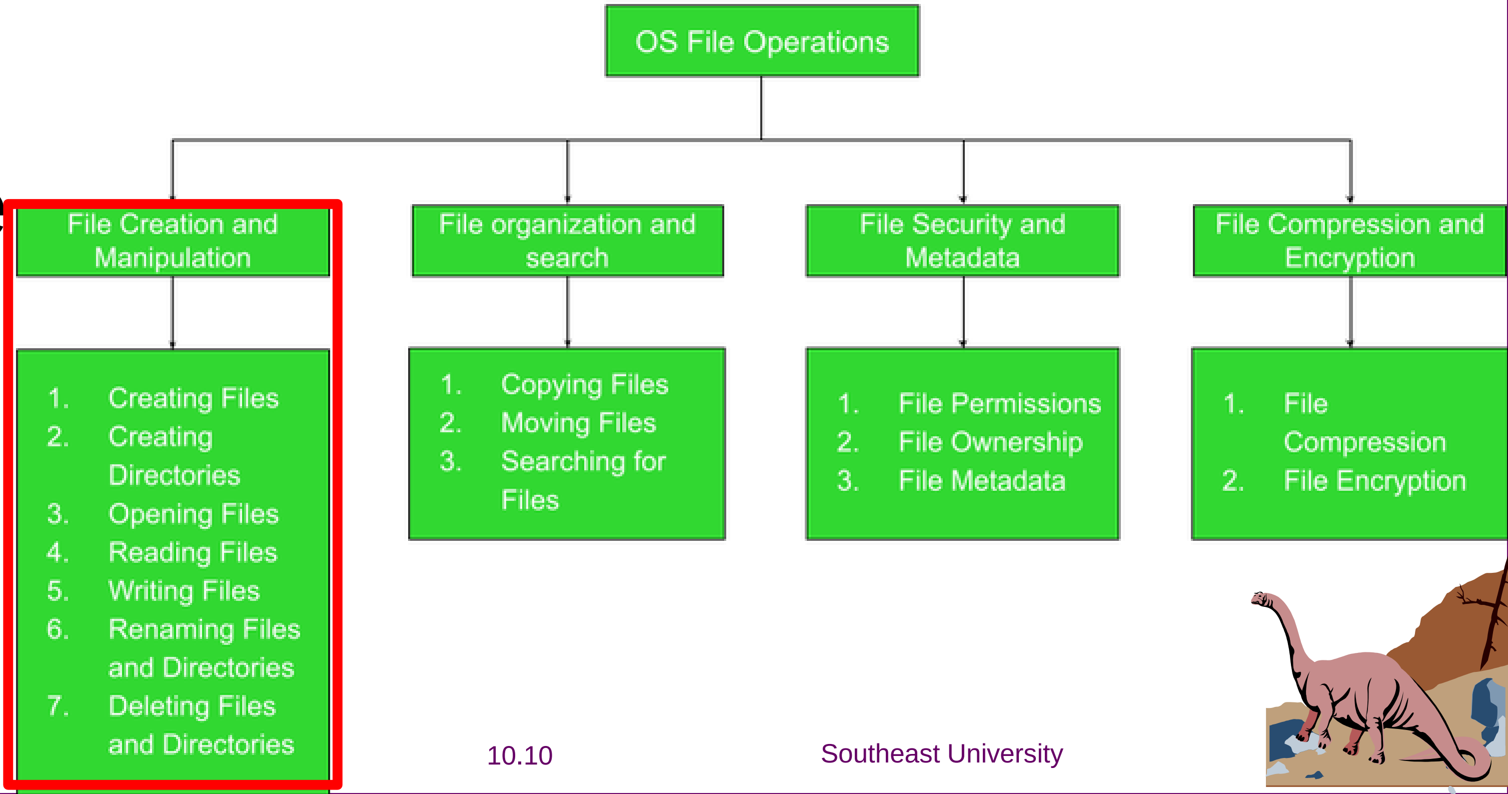


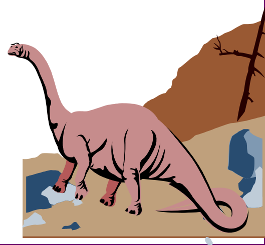
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information



File Operations from Developer's Perspective

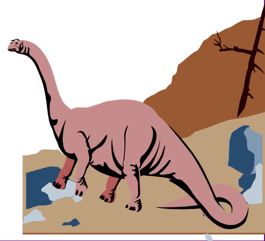
- Create
- Write
- Read
- Reposition within file – file seek
- Delete
- Truncate





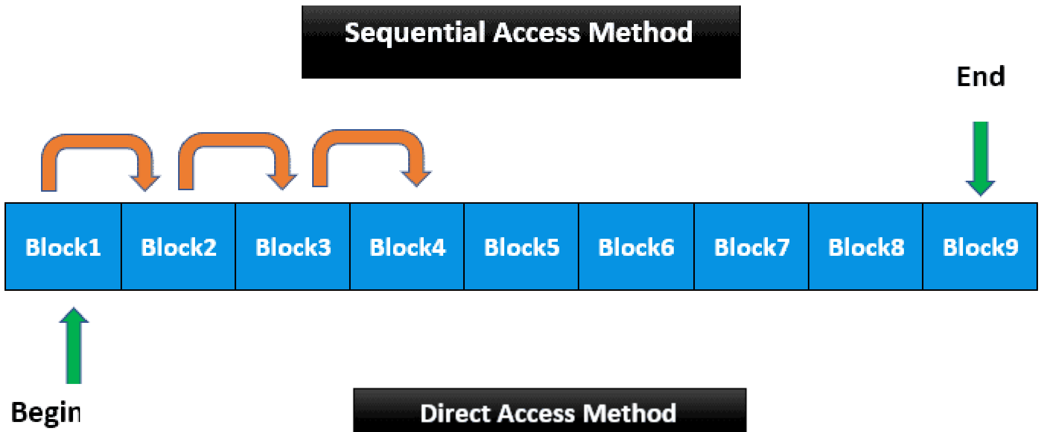
File Operations from Developer’s Perspective (cont.)

- open(F_i) – search the directory structure on disk for entry F_i, and move the content of the entry from disk to memory.
- close(F_i) – persist 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

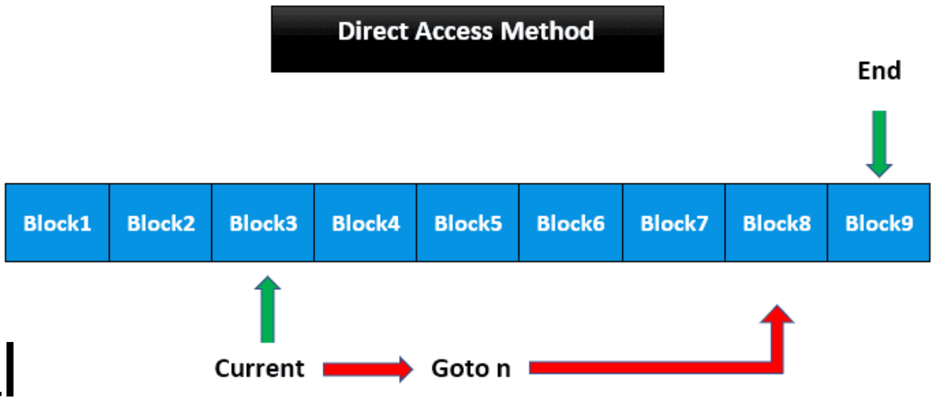


File Content Access Methods

- **Sequential Access Method (read/write)**

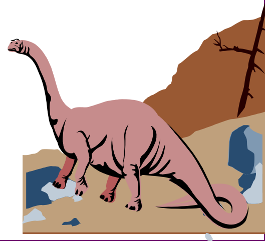


- **Direct Access Method (fseek)**



- Simulation of Sequential Access on a Direct-Access File

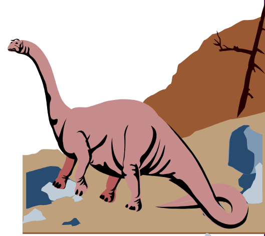
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;



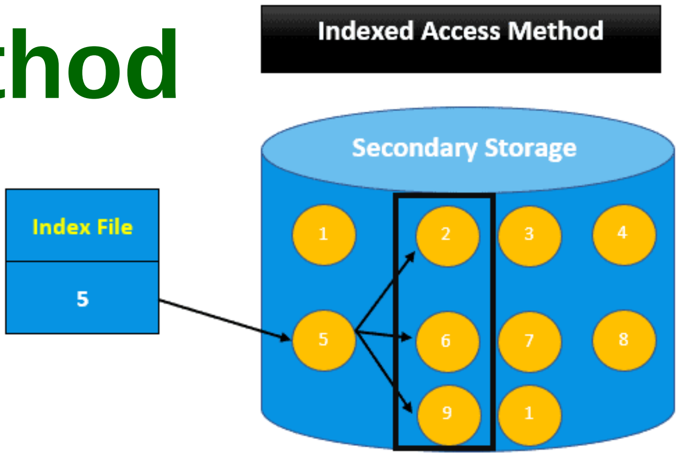
Example Code Modifying a Key-Value Pair in Fixed-Length Record Structure

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;
```

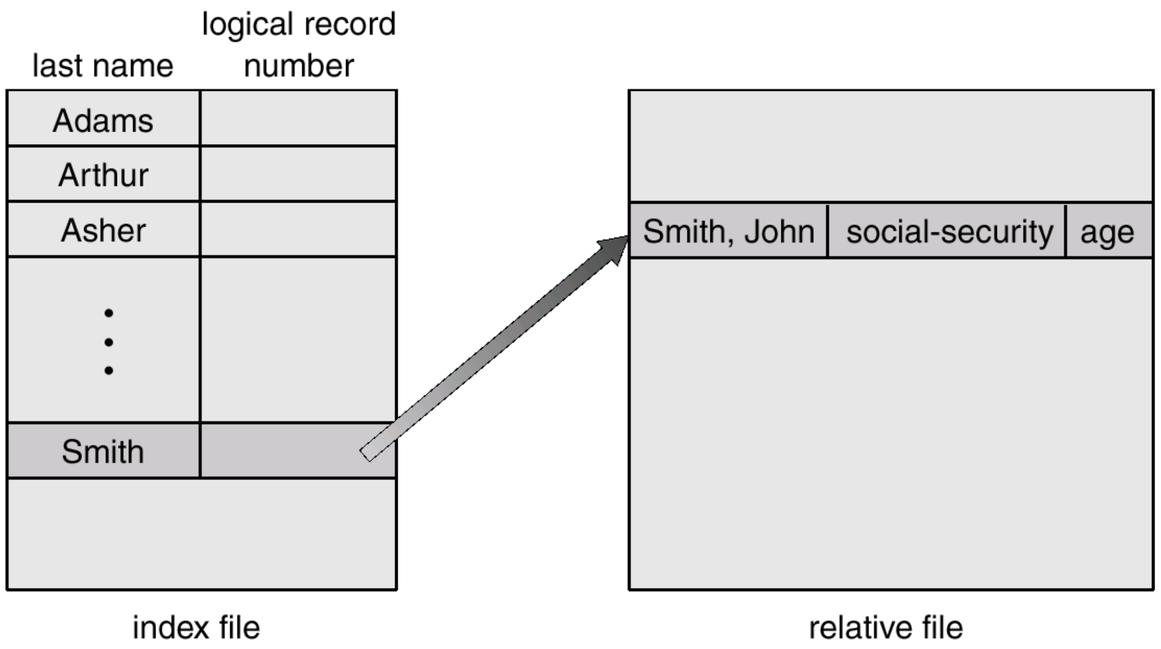


Index Access Method

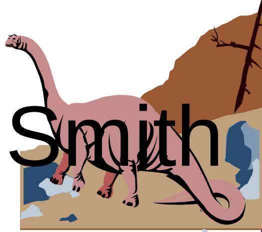


- Index Access Method:
  Store keys in the index file

- Store values (or records) in the relative file

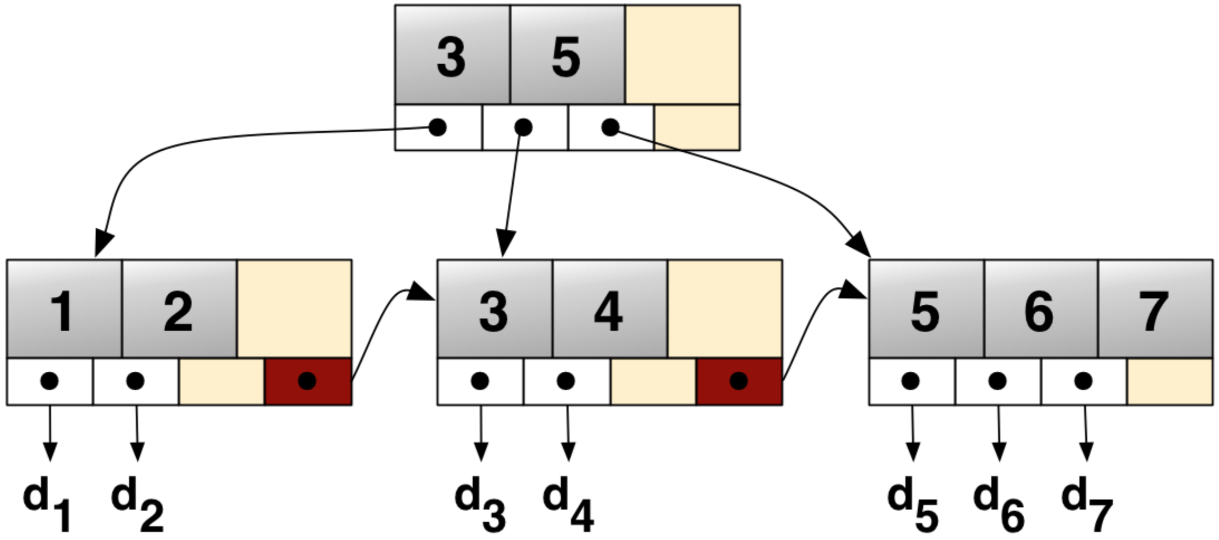


- How to quickly locate the record of John Smith



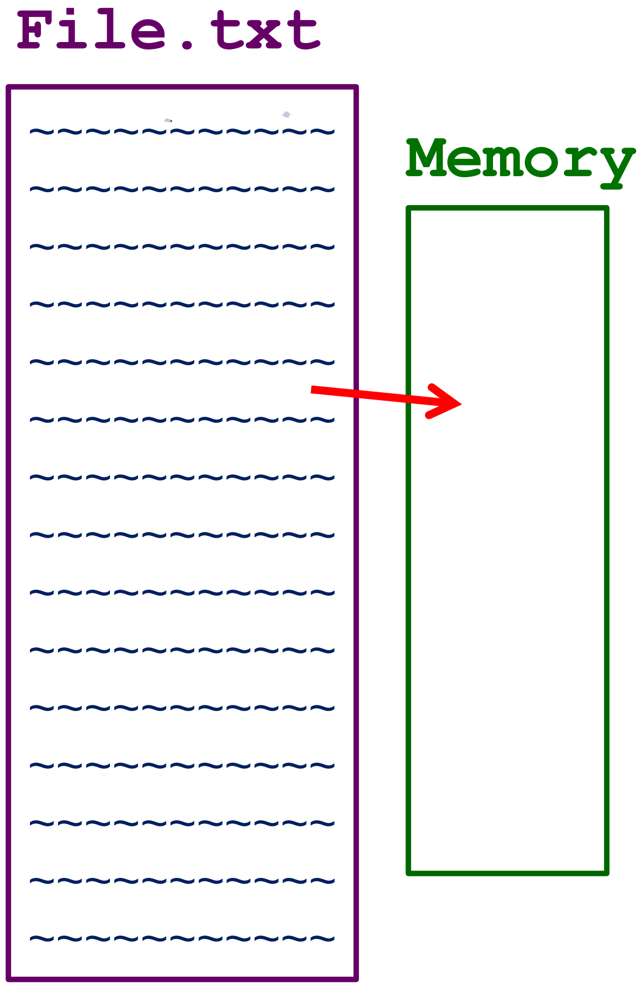
Index File is Organized as B+ Tree

- The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context — in particular, filesystems. Unlike binary search trees, B+ trees have very high fanout (number of pointers to child nodes in a node, typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.



https://en.wikipedia.org/wiki/B%2B_tree

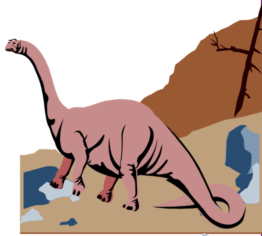
File Content Direct Access by Memory Mapped File



- **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

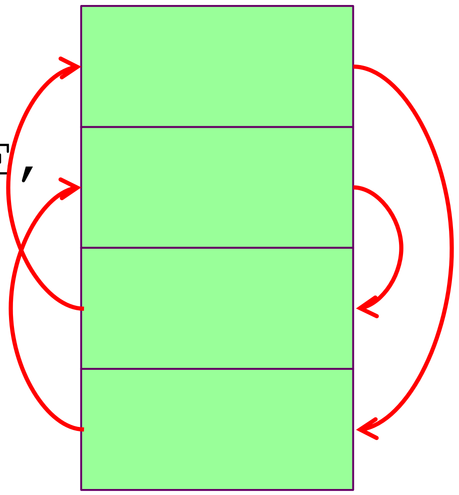
```
fd = open("file.txt", ….);
buffer = mmap(…, fd, …);

// manipulate the buffer

munmap(buffer, …);
close(fd);
```

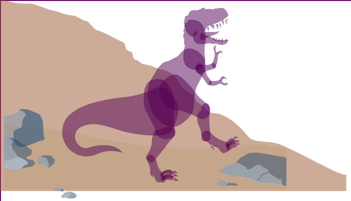
http://linux.die.net/man/2/mmap



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

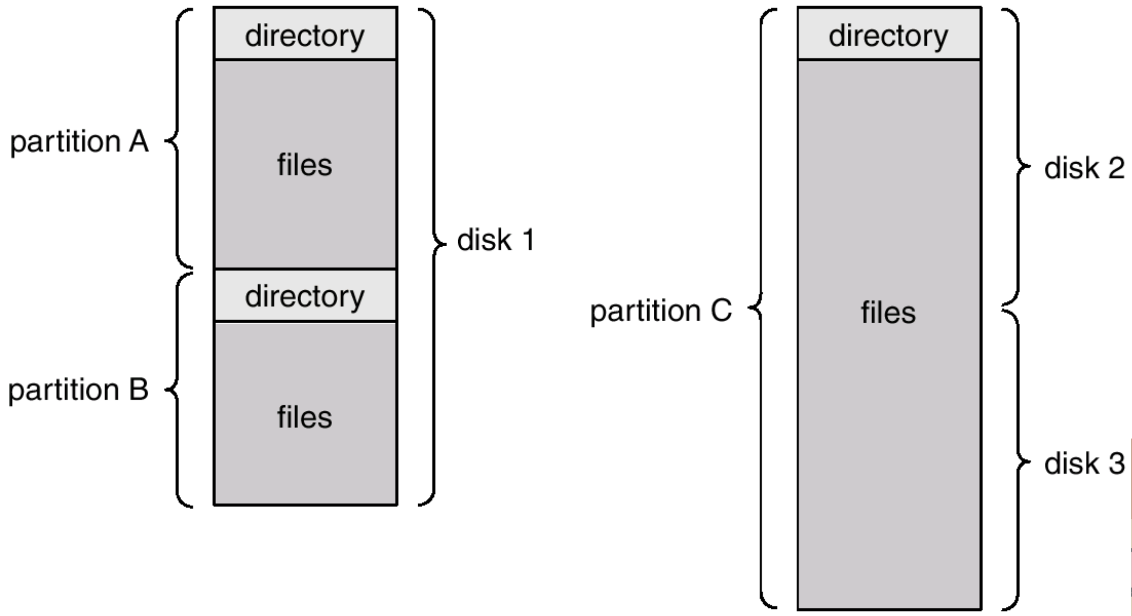
```
filename = argv[1];
card_size = strtol(argv[2], NULL, 10);
fd = open(filename, O_RDWR);
len = lseek(fd, 0, SEEK_END);
lseek(fd, 0, SEEK_SET);
buf = mmap(NULL, len, PROT_READ | PROT_WRITE,
           MAP_FILE | MAP_SHARED, fd, 0);
if( buf == (void*) -1) {
      fprintf(stderr, “mmap failed.\n”);
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





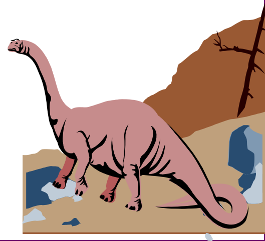
Directory Structure

- 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



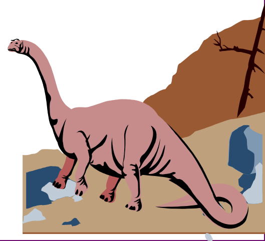
Operations Performed on Directory

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



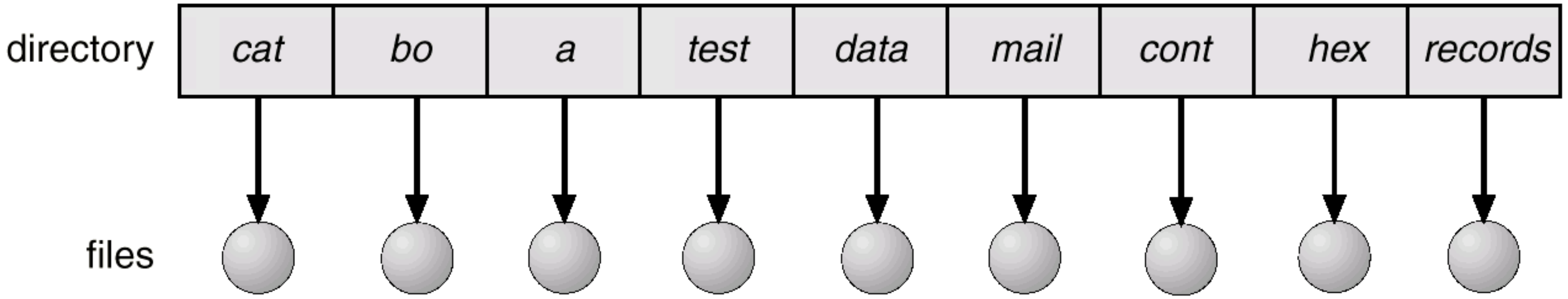
Organize the Directory (Logically) to Obtain

- **Efficiency** – locating a file quickly.
- **Naming** – convenient to users.
  - Two users can have the 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

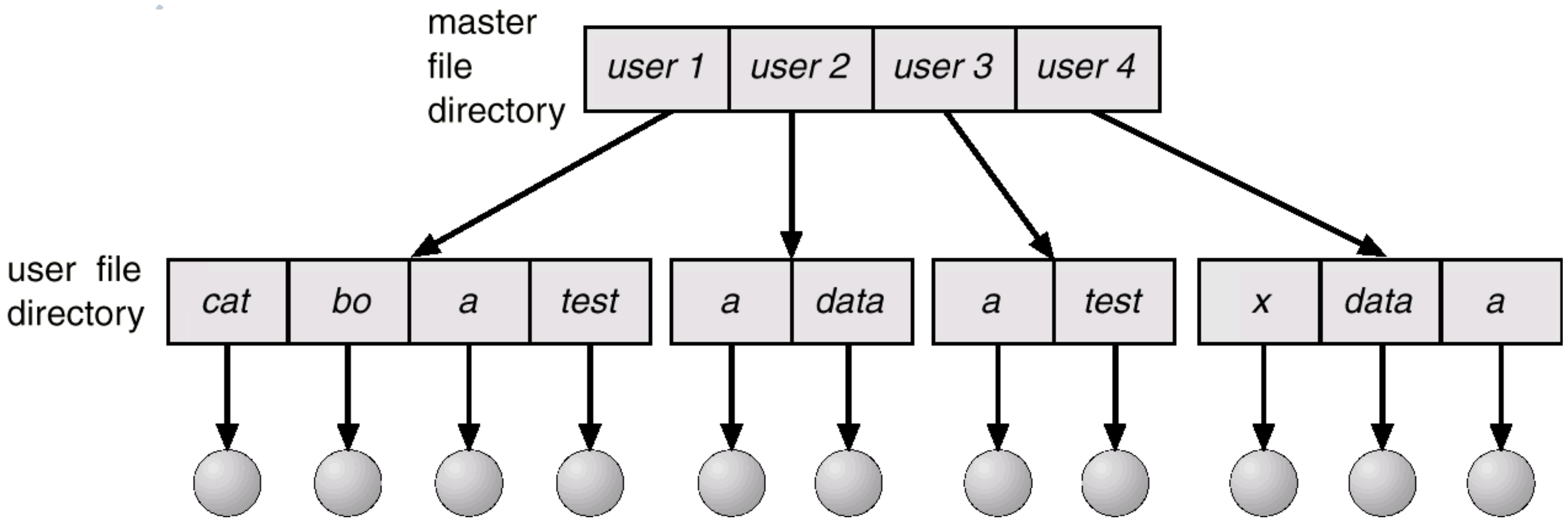
- A single directory for all users.



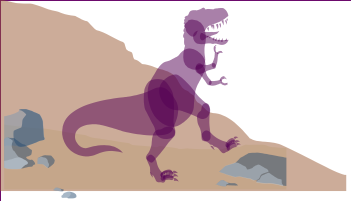
  - Naming problem
  - Grouping problem

Two-Level Directory

- Separate directory for each user.

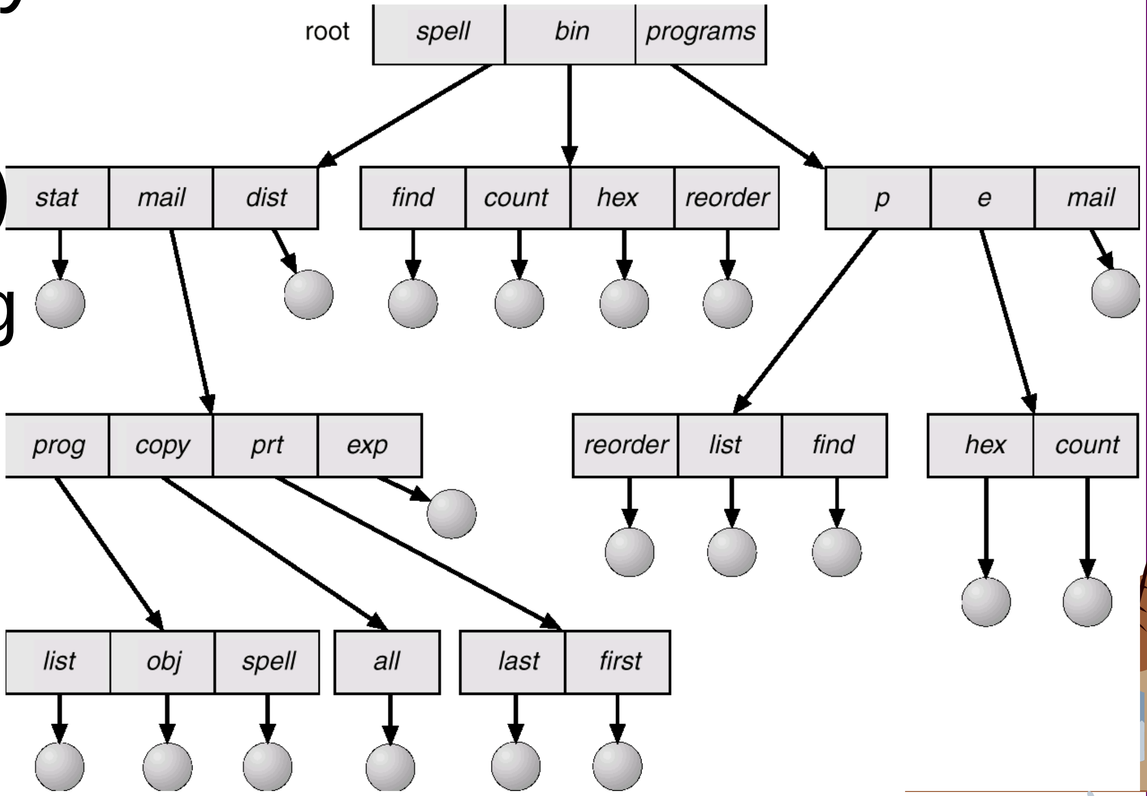


- Efficient searching
- Support path name, so can have the same file name for different users
- No grouping capability



Tree-Structured Directories

- Efficient searching
- Convenient naming
  - Two users can have the same name for different files
- Grouping capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list



Tree-Structured Directories (cont.)

- **Absolute** or **relative** path name
- Can create a new file in current directory (**pwd**)
- Example: if in current directory **/mail**

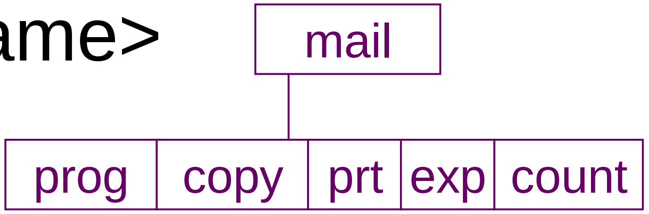
  mkdir count

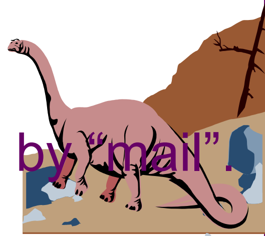
- Delete a file

  rm <file-name>

- Creating a new subdirectory in current directory.

  mkdir <dir-name>



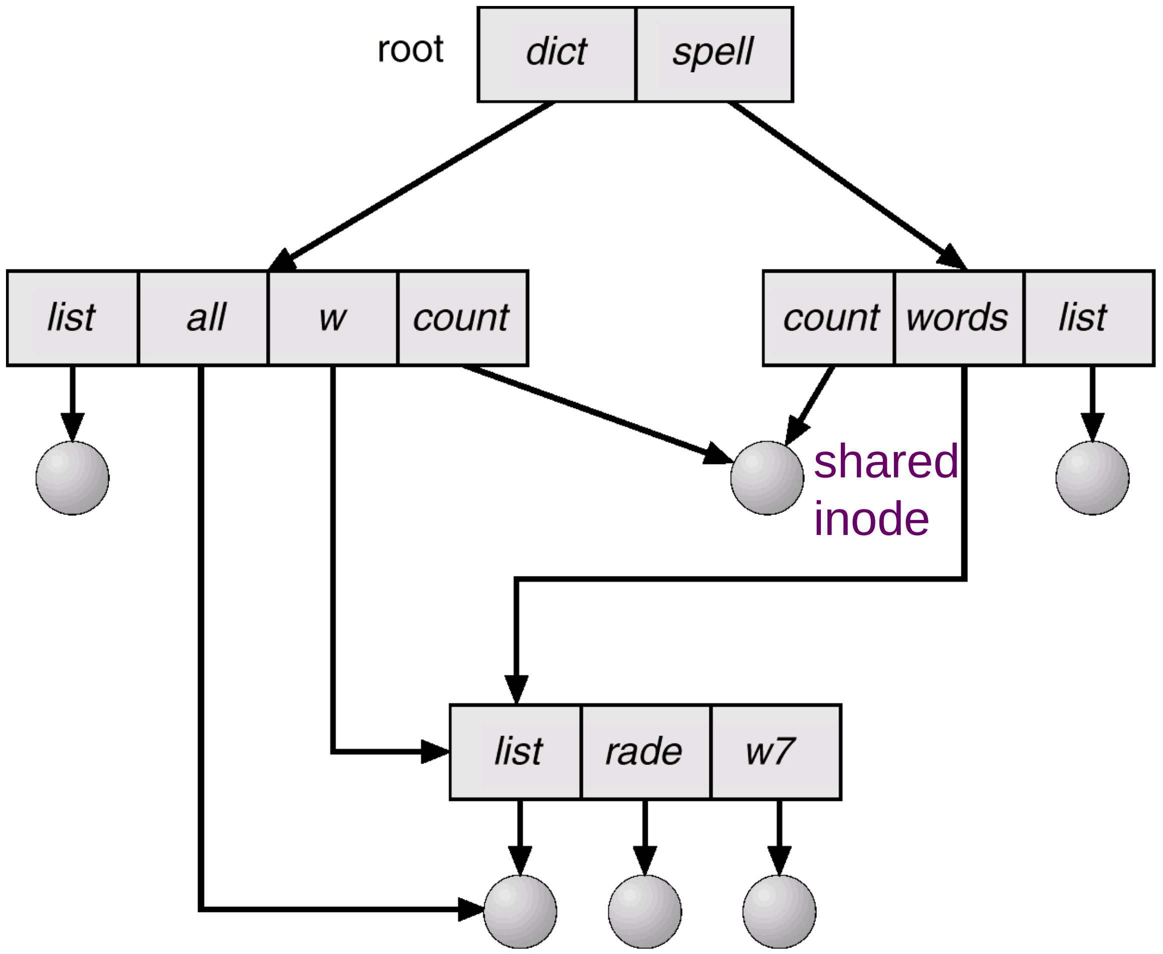


Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.

Acyclic-Graph Directories

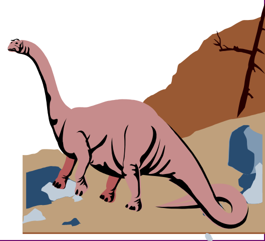
- Have shared subdirectories and files.

The same file can have several different paths.



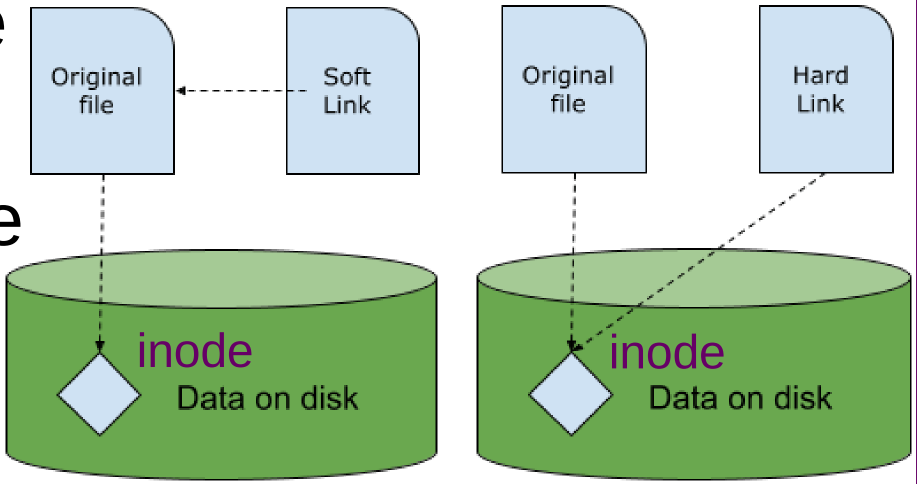
Acyclic-Graph Directories (cont.)

- Two different names (aliasing)
- If *dict* deletes *count* $\Rightarrow$ dangling pointer.
- Solutions:
  - Backpointers, so we can delete all pointers.
  - Entry-hold-count solution.
- These links we talked about are hard links in UNIX/Linus



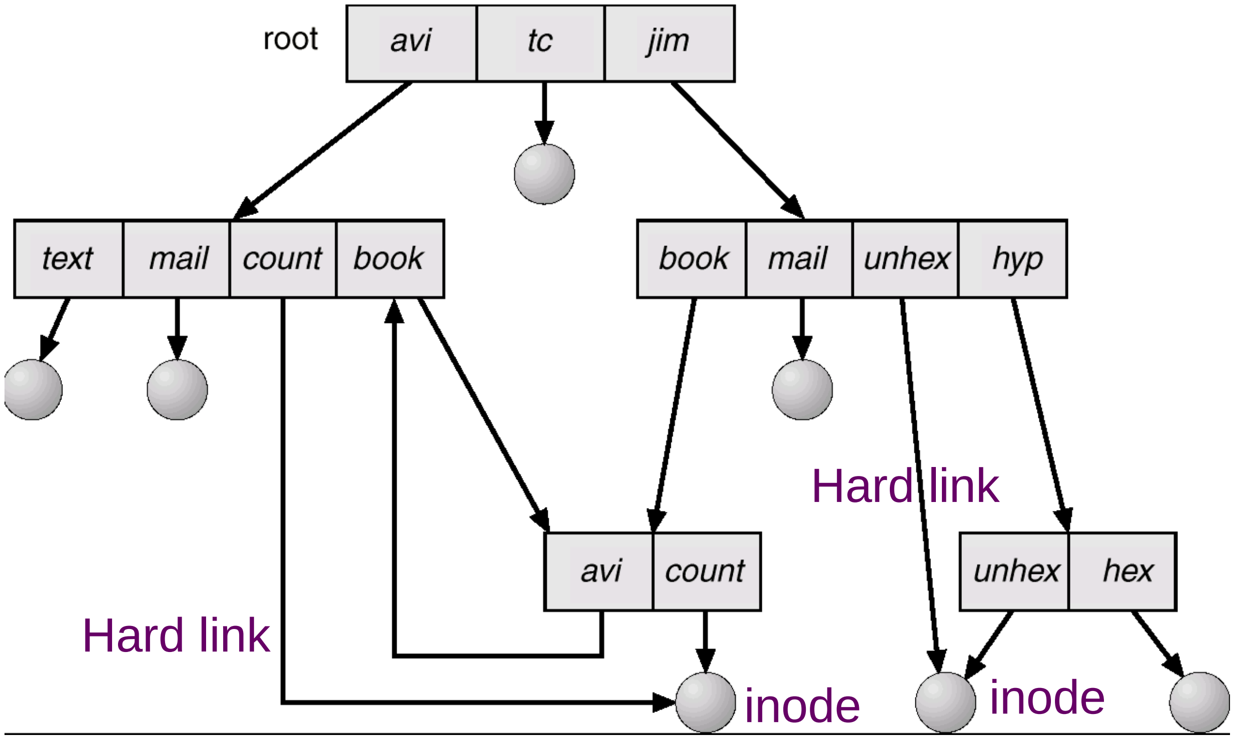
In Linux, Shortcuts are known as Links

- Soft Links (symbolic links)
  - You can make a link for either a file or a folder
  - You can create link (shortcut) on different partition
  - You got a different inode number from original.
  - If real copy is deleted the link will not work.
- Hard Links
  - For files only, and you cannot create a hard link on different partition (it should be on same partition)
  - You got the same inode number as original
  - If the real copy is deleted the link will work (because it act as original file)



General Graph Directory

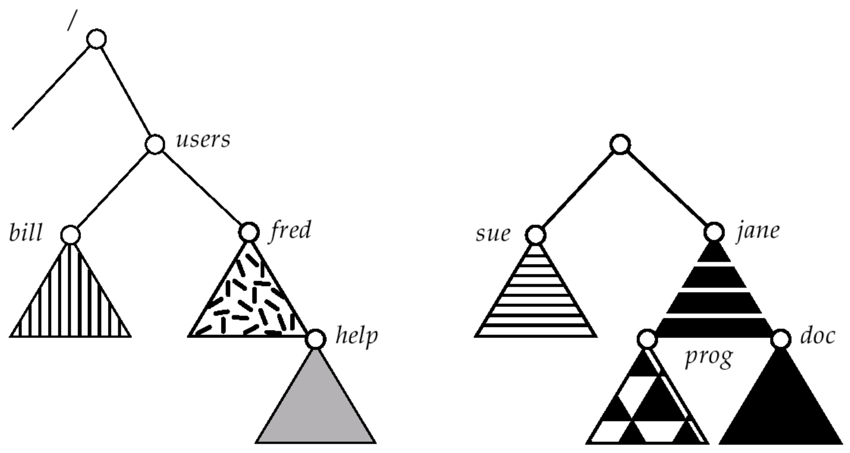
- The directory graph can have cycles



- 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

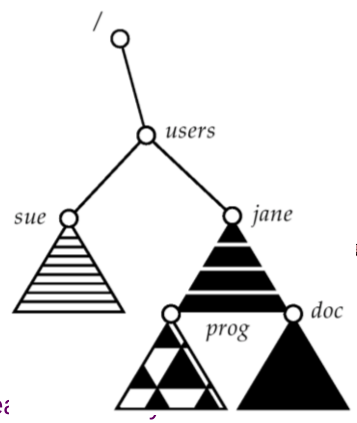
- A file system must be **mounted** before it can be accessed.
- An unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**.
- (a) Existing (b) Unmounted Partition

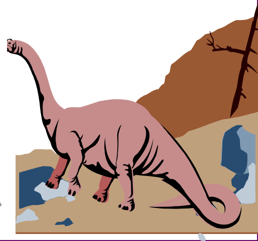


(a)

(b)

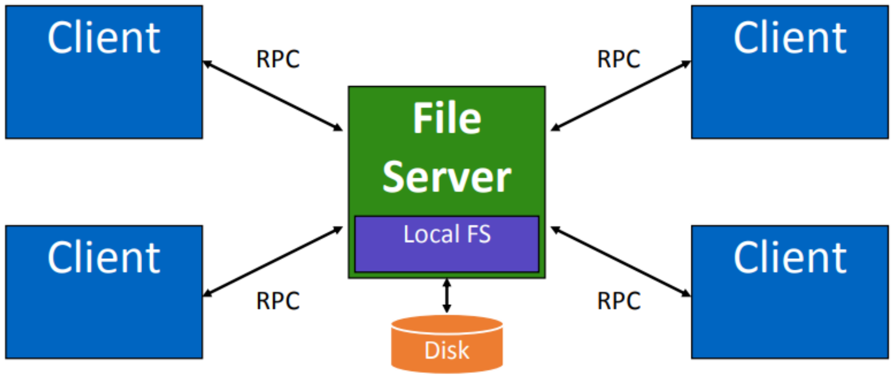
Mount Point

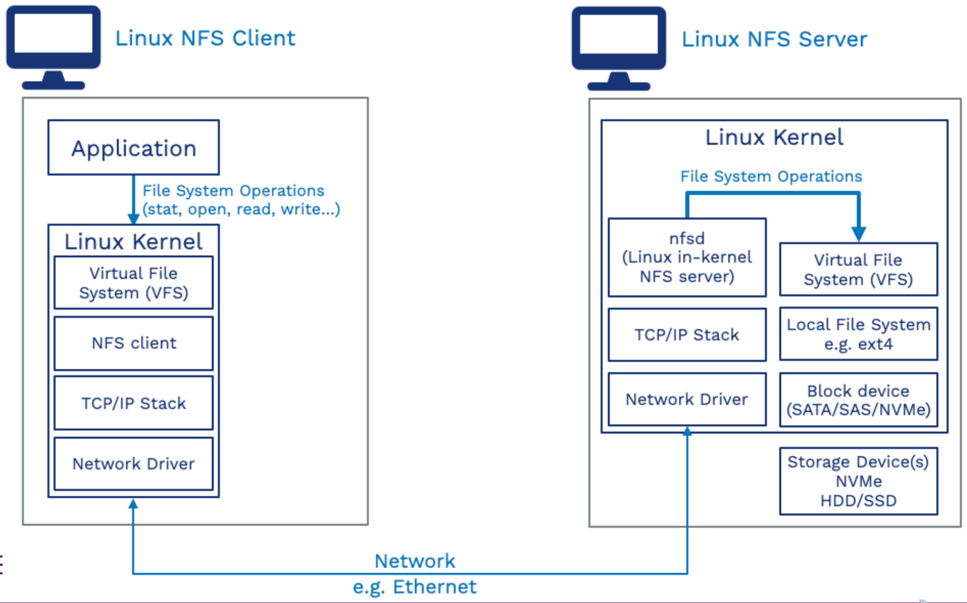


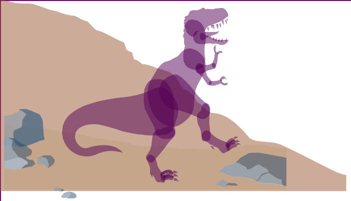


File Sharing

- 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.



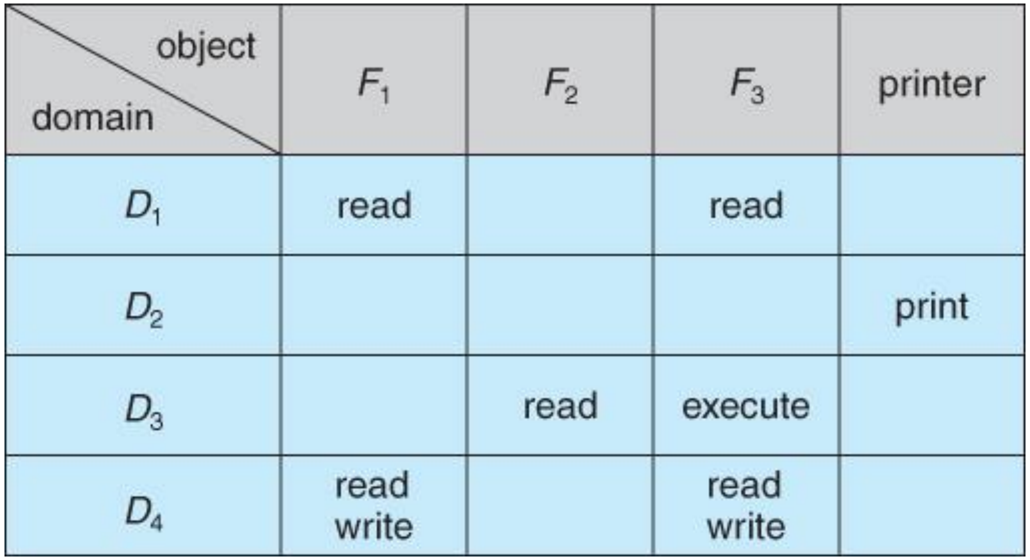

Linux NFS Client
Application
File System Operations (stat, open, read, write...)
Linux Kernel
Virtual File System (VFS)
NFS client
TCP/IP Stack
Network Driver
Linux NFS Server
Linux Kernel
File System Operations
nfsd (Linux in-kernel NFS server)
Virtual File System (VFS)
TCP/IP Stack
Local File System e.g. ext4
Network Driver
Block device (SATA/SAS/NVMe)
Storage Device(s) NVMe HDD/SSD
Network e.g. Ethernet



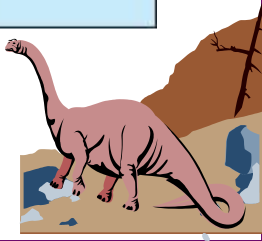
File Access Protection

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

Role-based Access Control



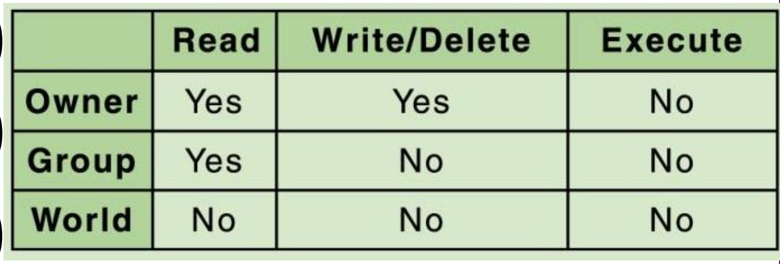
object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	



Access Lists and Groups

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

  a) **owner access** 6 ⇒ 1 1 0
  b) **group access** 4 ⇒ 1 0 0
  c) **public access** 0 ⇒ 0 0 0



	Read	Write/Delete	Execute
Owner	Yes	Yes	No
Group	Yes	No	No
World	No	No	No

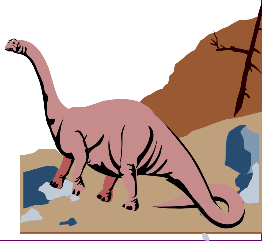
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

chmod 761 game

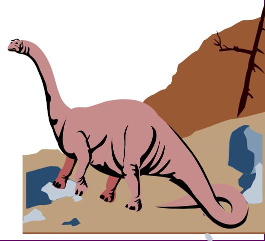
Attach a group to a file

chgrp G game



A Sample UNIX Directory Listing

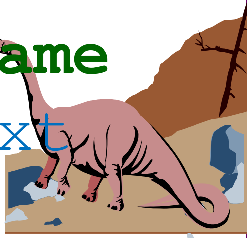
```
-rw-rw-r--    1 pbg   staff     31200   Sep 3 08:30    intro.ps
drwx------    5 pbg   staff       512   Jul 8 09.33    private/
drwxrwxr-x    2 pbg   staff       512   Jul 8 09:35    doc/
drwxrwx---    2 pbg   student     512   Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg   staff      9423   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/
```

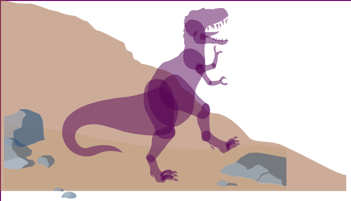


Question about File Access-Control

- 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
```

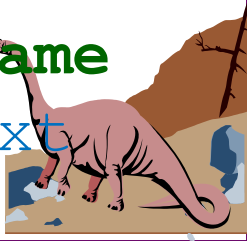


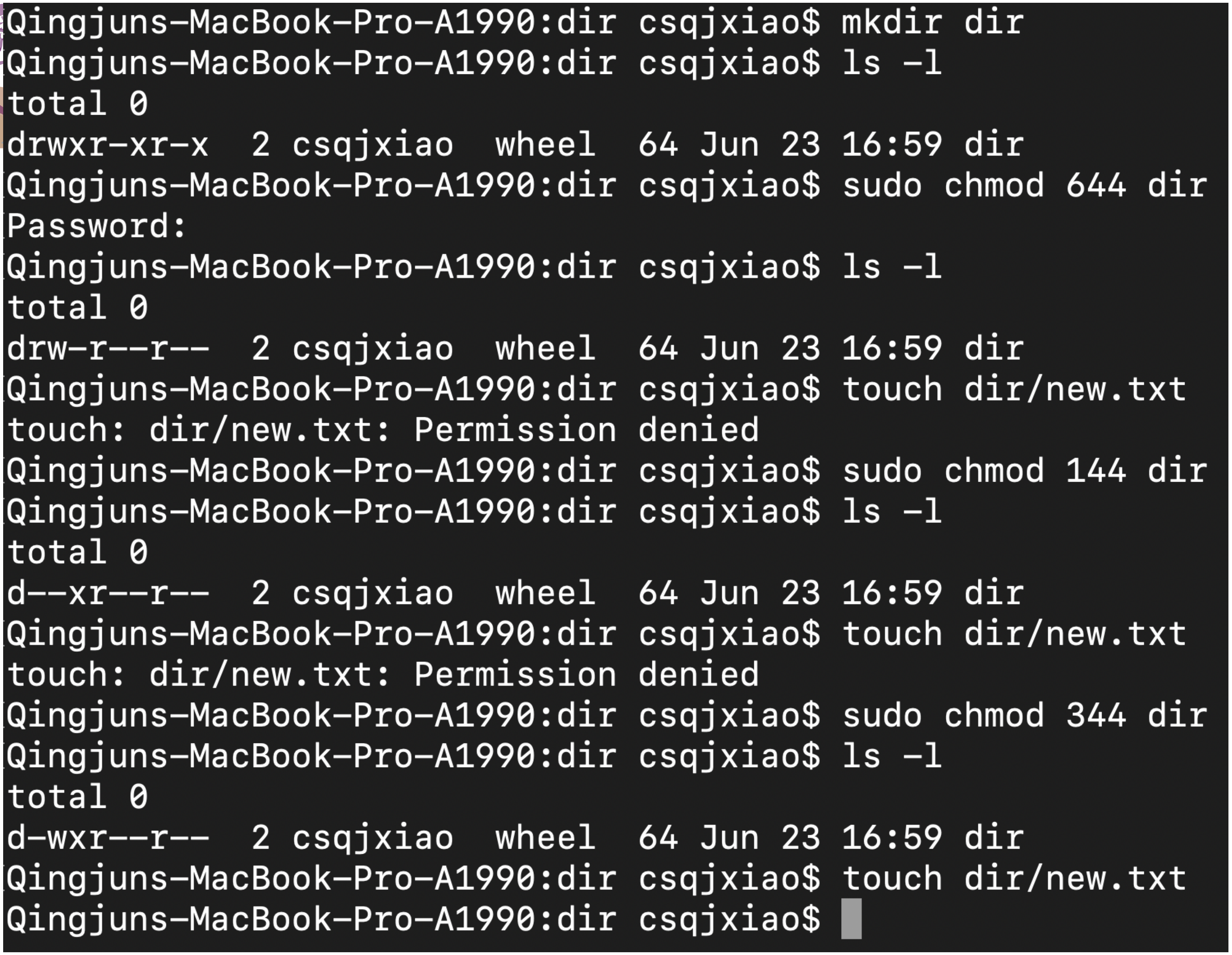


Another Question

- 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
```





```
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ mkdir dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
drwxr-xr-x  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 644 dir
Password:
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
drw-r--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
touch: dir/new.txt: Permission denied
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 144 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
d--xr--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
touch: dir/new.txt: Permission denied
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ sudo chmod 344 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ ls -l
total 0
d-wxr--r--  2 csqjxiao  wheel  64 Jun 23 16:59 dir
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ touch dir/new.txt
Qingjuns-MacBook-Pro-A1990:dir csqjxiao$ 
```

MacOS的执行结果：目录dir在创建的开始，rwx权力都属于owner。后面不管是644、144权力，都会touch报错Permission denied。改成344，有x和w权力，就没问题了。 cd dir 进入dir目录的操作，一定需要dir目录的执行权。