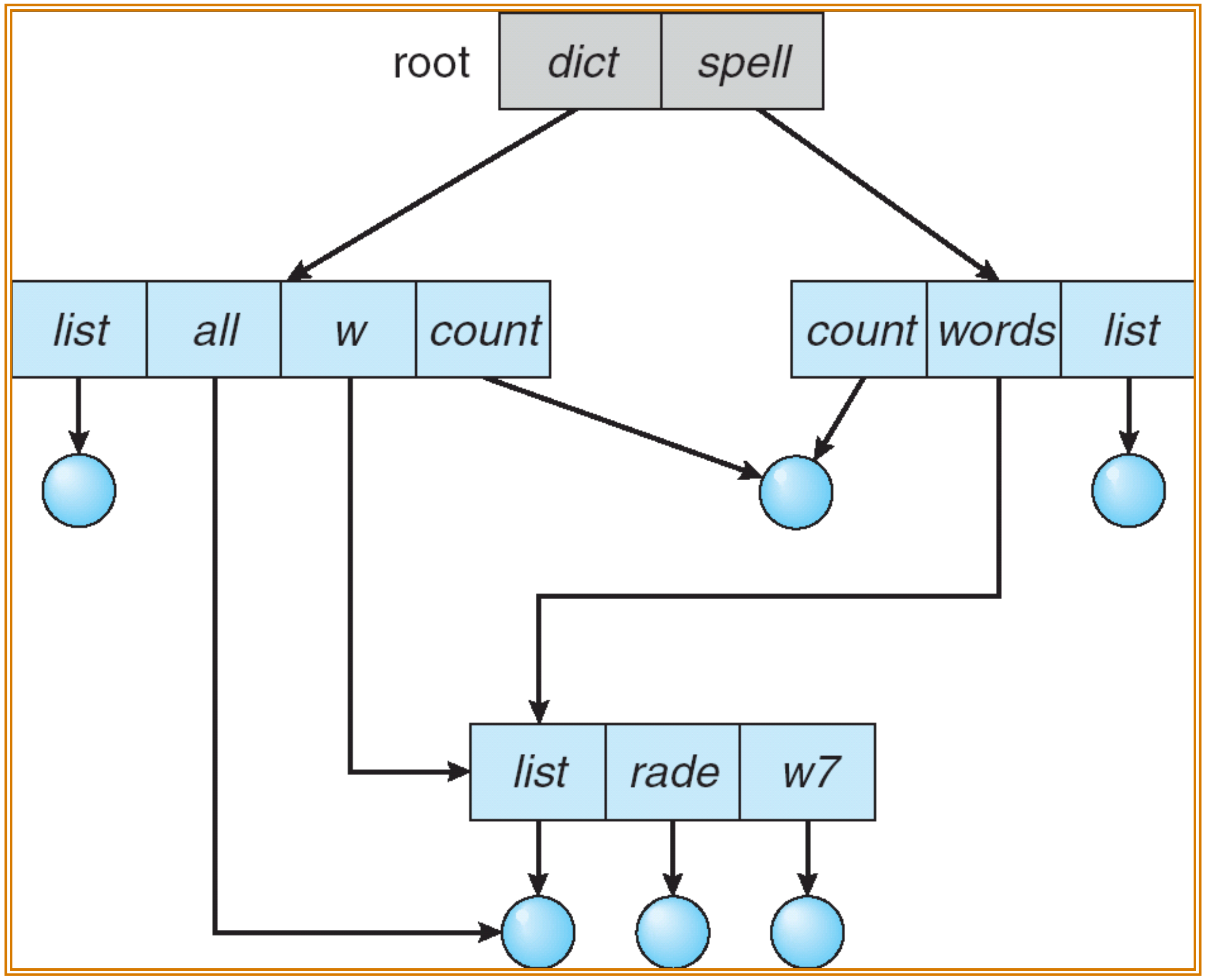
# Acyclic-Graph Directories

- Have shared subdirectories and files



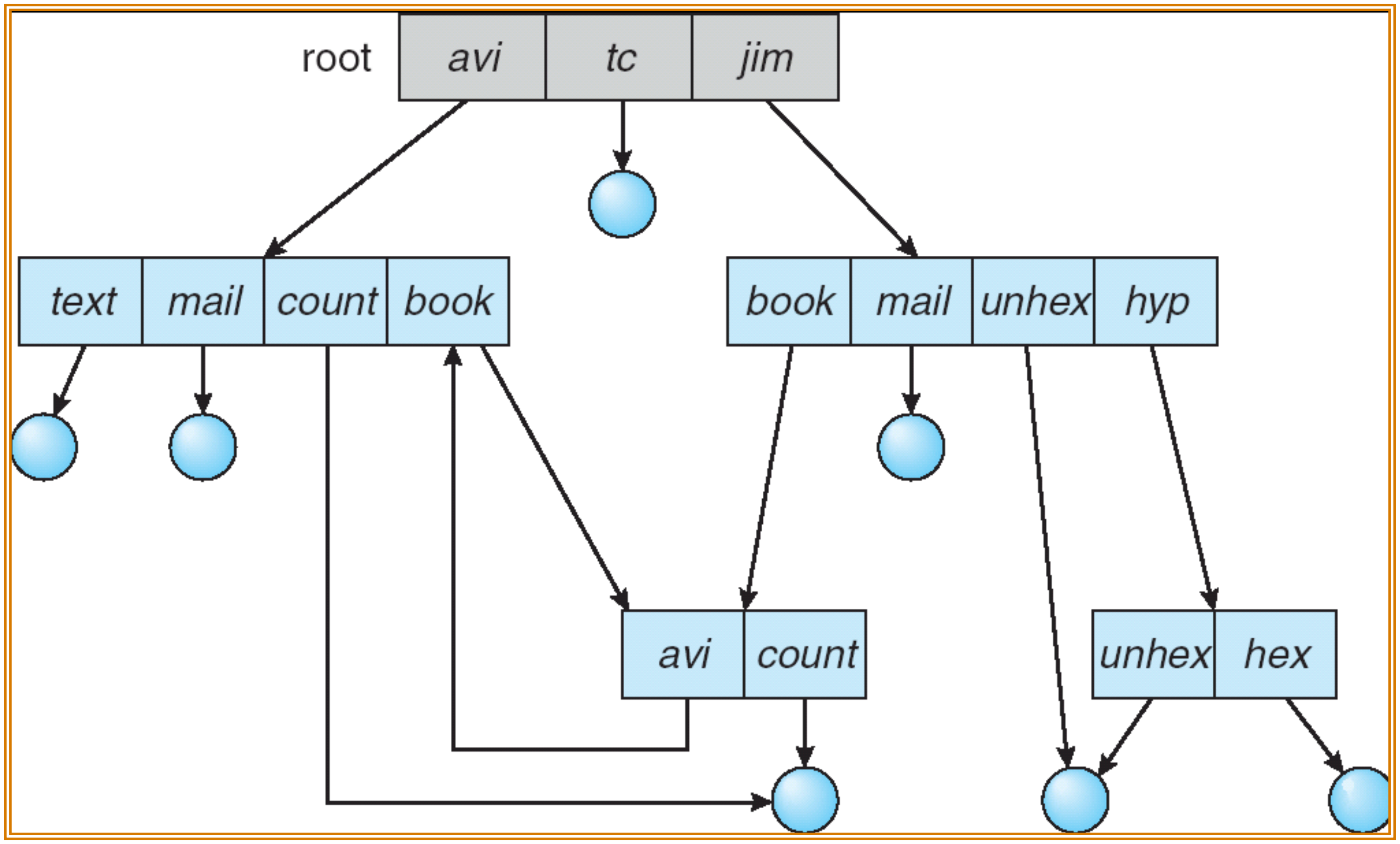
## Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)

- If *dict* deletes *list* ⇒ dangling pointer

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

# General Graph Directory



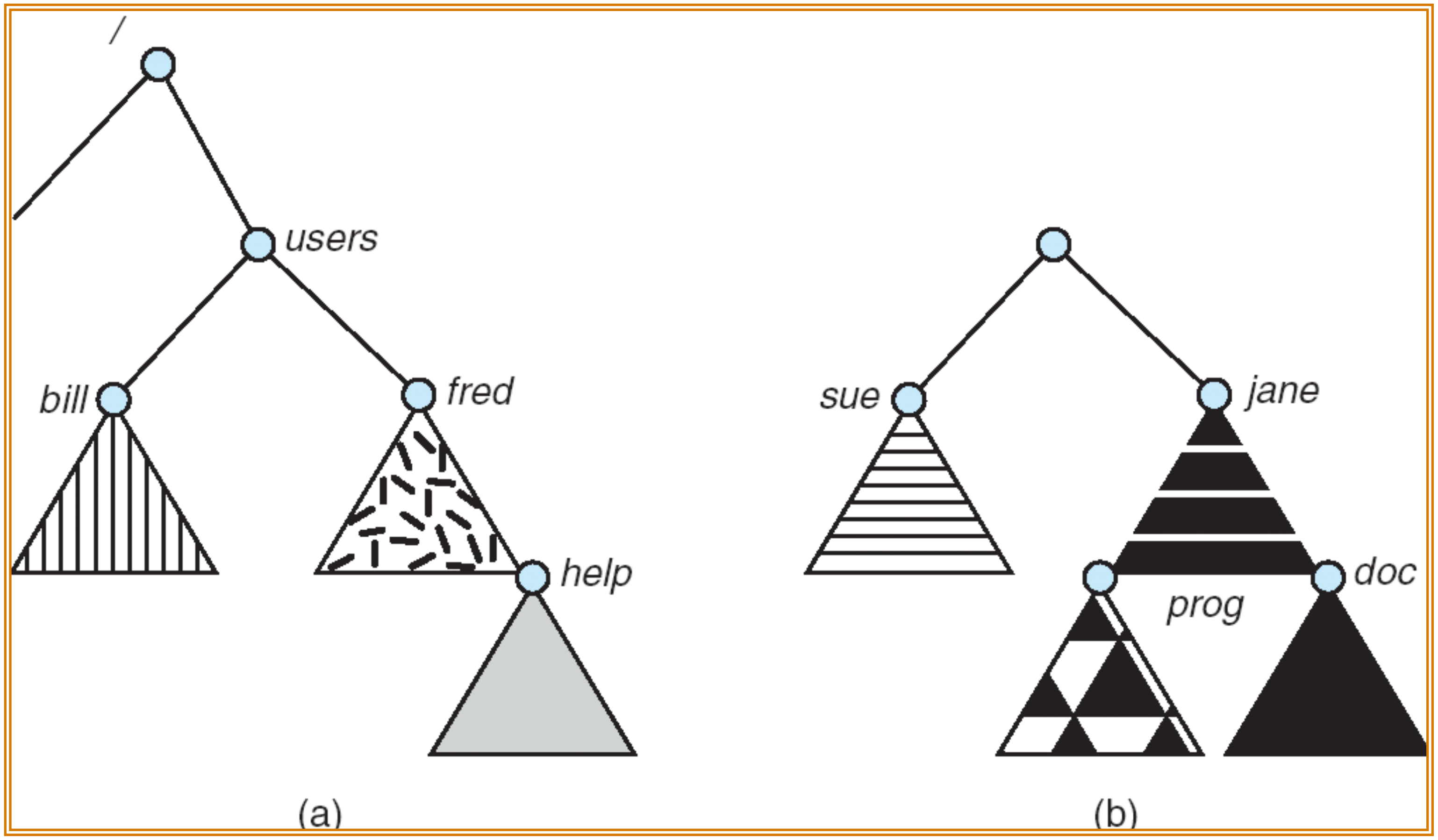
## General Graph Directory (Cont.)

- 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

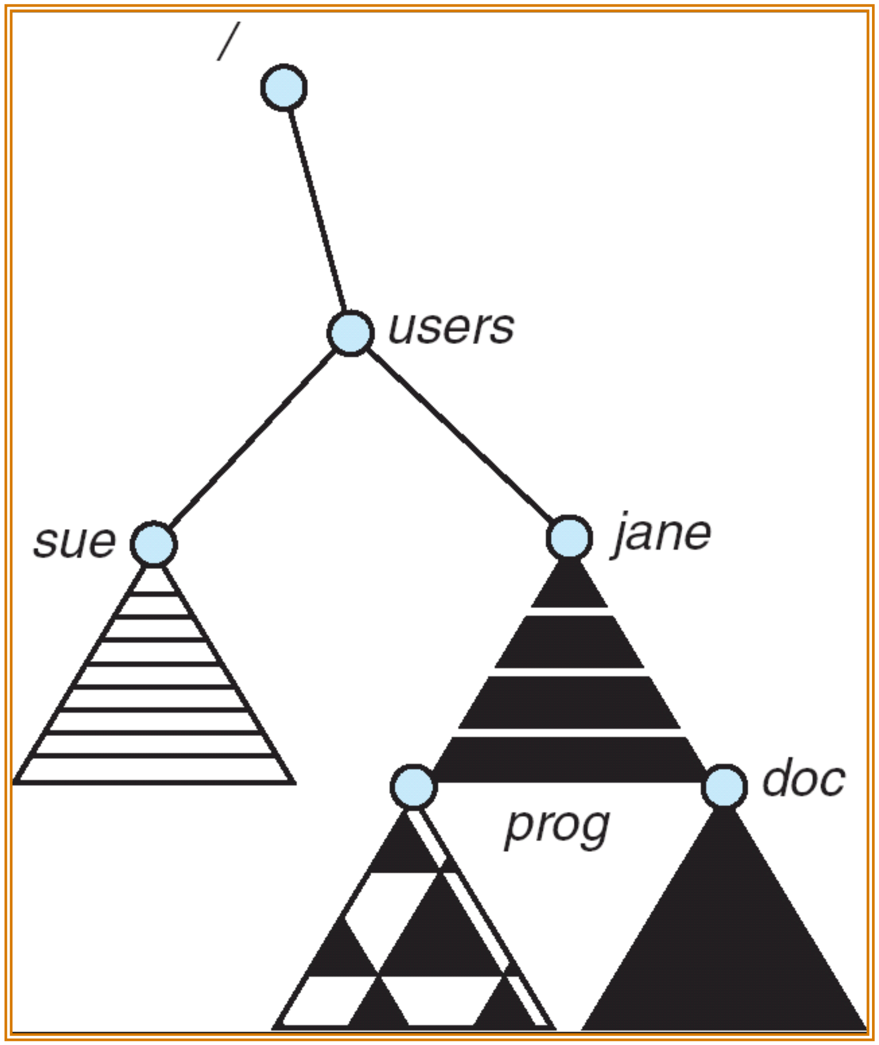
- A file system must be mounted before it can be accessed
- A unmounted file system is mounted at a mount point

# (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

## File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

## File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 7 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed

## Protection

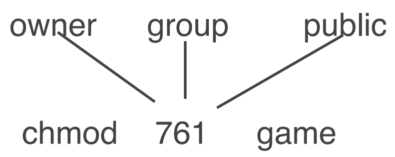
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

# Access Lists and Groups

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

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

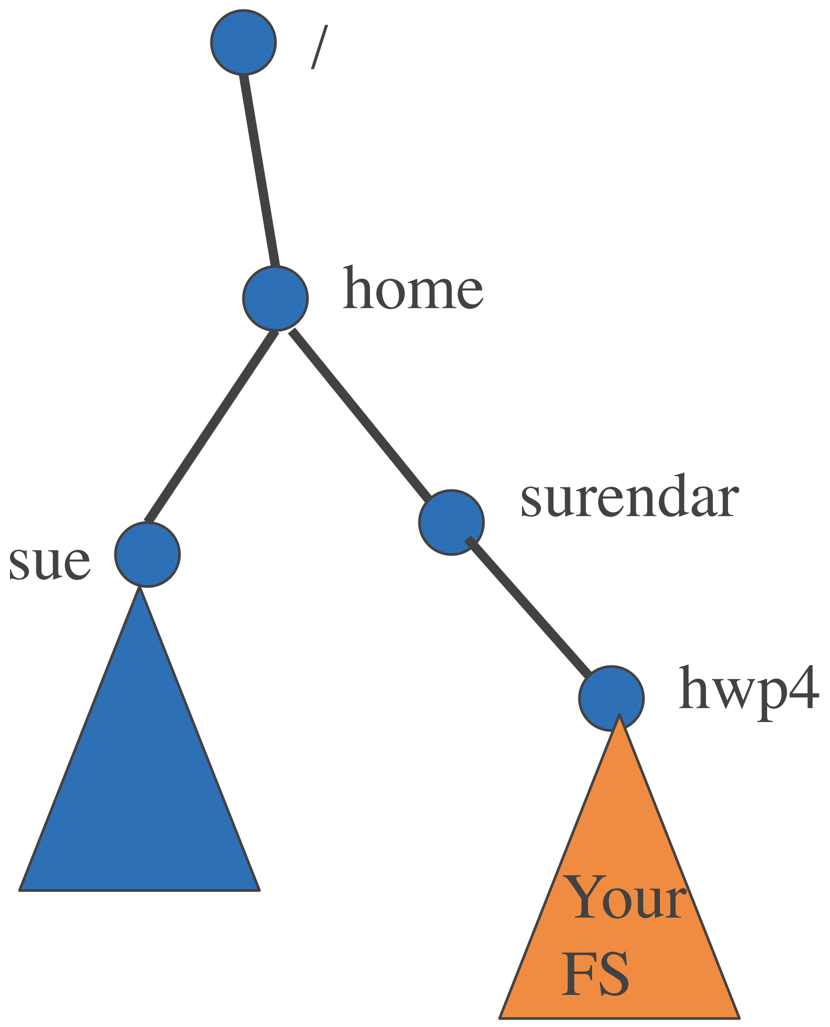
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



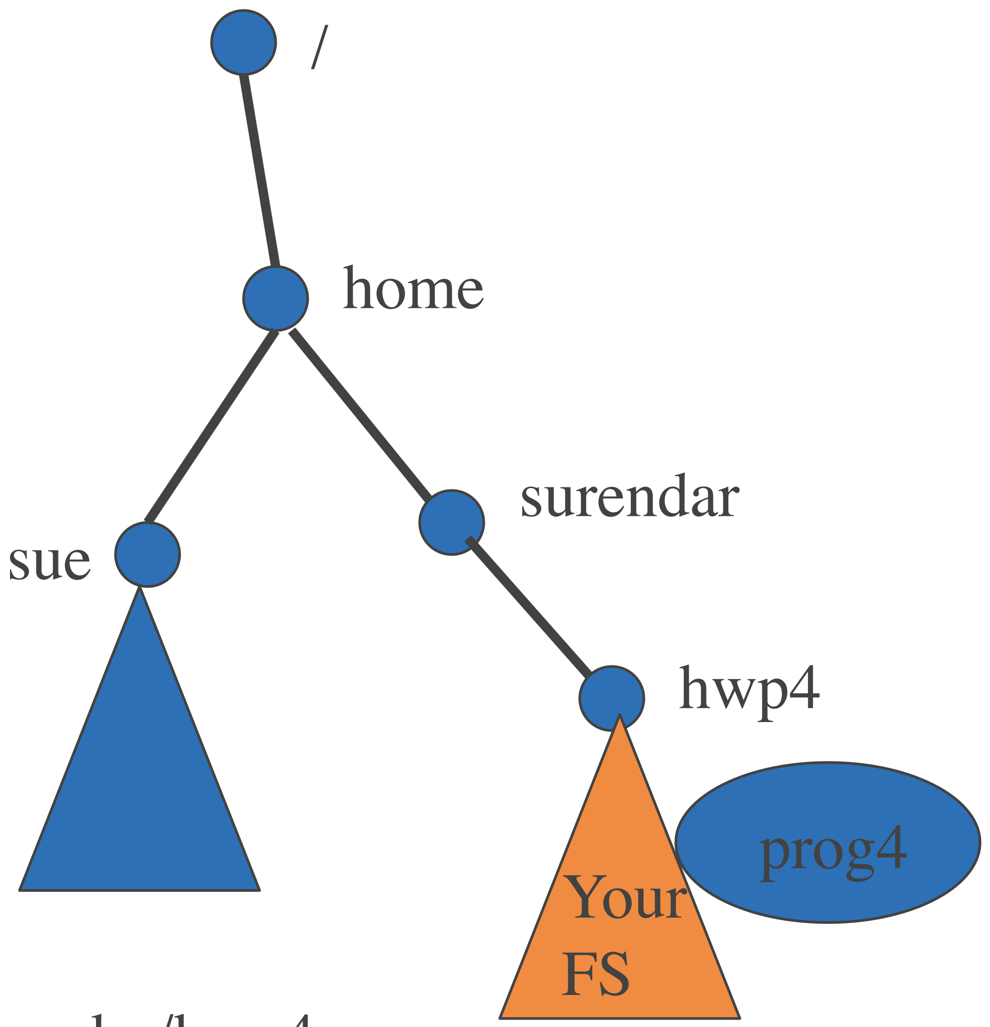
Attach a group to a file

chgrp    G    game

## Project – 4: FS using FUSE



# Project – 4: FS using FUSE



Run

```
./prog4 /home/surendar/hwp4
```