Overview

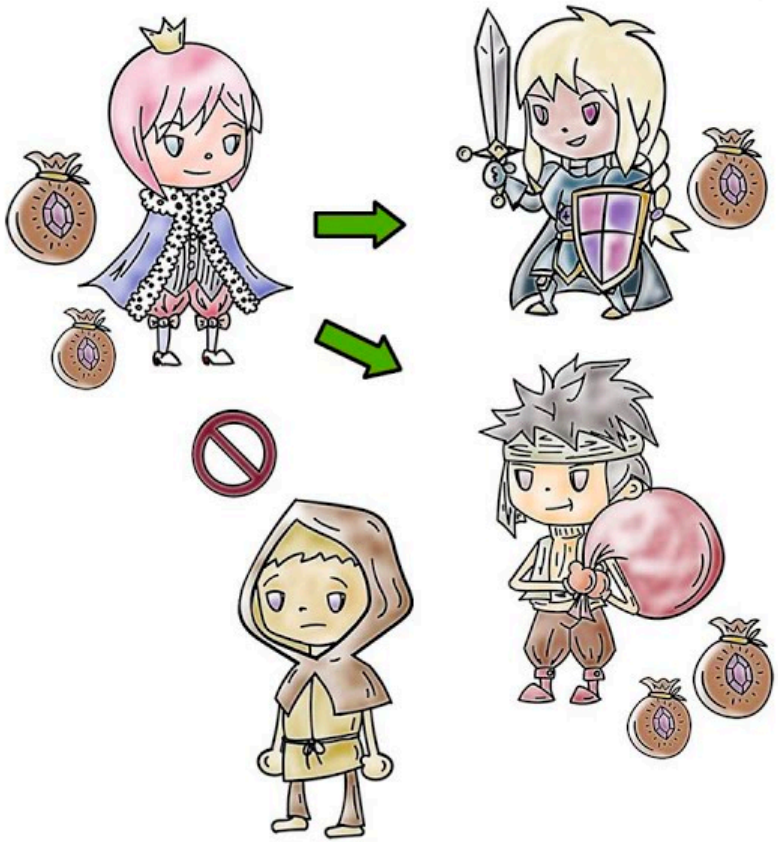
- Last Lecture
  - Scripting
- This Lecture
  - Linux/Unix file system
- Next Lecture
  - Basic System Administration

Access control models

- Discretionary access control
  - a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).
  - Subject centered
- Mandatory access control
  - a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity
  - Object centered

DAC

Discretionary Access Control



In discretionary access control (DAC), owner of a resource decides how it can be shared

- Owner can choose to give read or write access to other users

MAC

Mandatory Access Control (MAC) Models

- User works in a company and the company decides how data should be shared
  - Hospital owns patient records and limits their sharing
    - Regulatory requirements may limit sharing

ACL

- ACL: Access Control List
  - more secure and convenient than traditional access control
- ACL is a list of permissions attached to an object (file)
  - The list specifies who or what is allowed to access the object and what operations are allowed to be performed on the object
  - ACL consists of entries like [user, operations]
  - The operations can be R, W, X, D, and etc.

Role-based Access Control (RBAC)

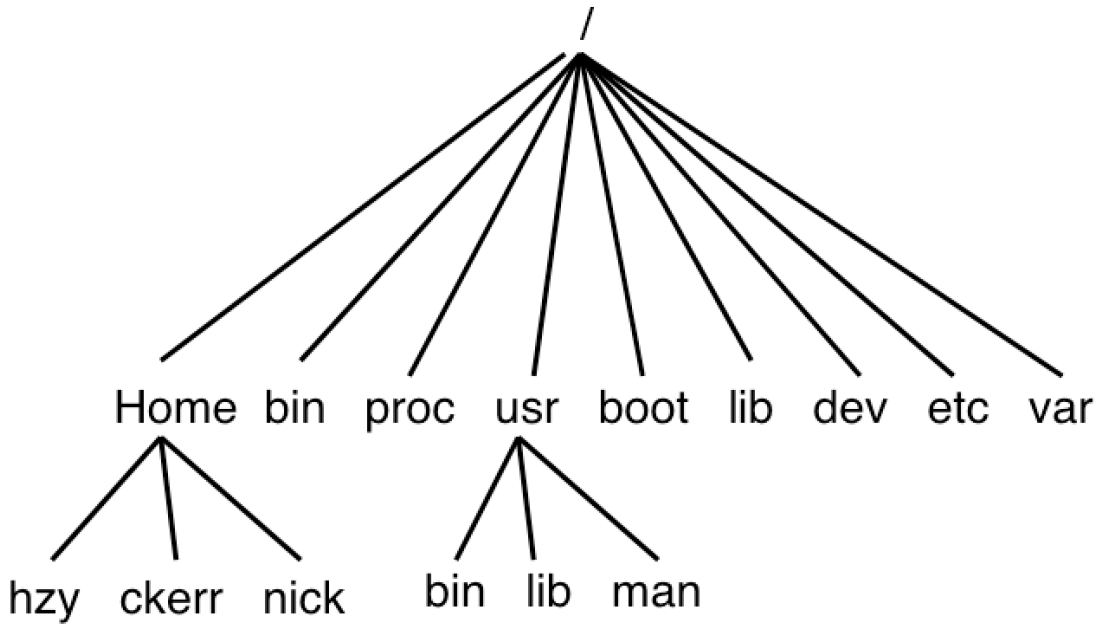
- An approach to limiting user access to systems
- It is policy neutral and defined around roles and privileges.
- It can implement both DAC and MAC.
- RBAC is similar to ACL except RBAC aggregates a group of users with the same privileges as roles.

Devices for filesystems

- USB devices: flash memory, hard drive, etc.
- Floppy disk
  - Density: 1.2M or 1.44M
  - Similar to hard disk except small and slow
- CD-ROM
  - ISO 9660 file system, UDF for DVD-ROM
- Tapes
  - Important backup media but will not be covered in this course
  - Check related sites for more info.
- Device naming convention under */dev*
  - *sda1,sda2,…, sdb1, sdb2, …,hda1, hda2,…, hdb1, hdb2, …, fd0,fd1,…*
  - Disks are labelled as a,b,…, and partitions are recognised by *1*, *2*, 3, ...

Linux/Unix directory tree

- Refer to http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/index.html

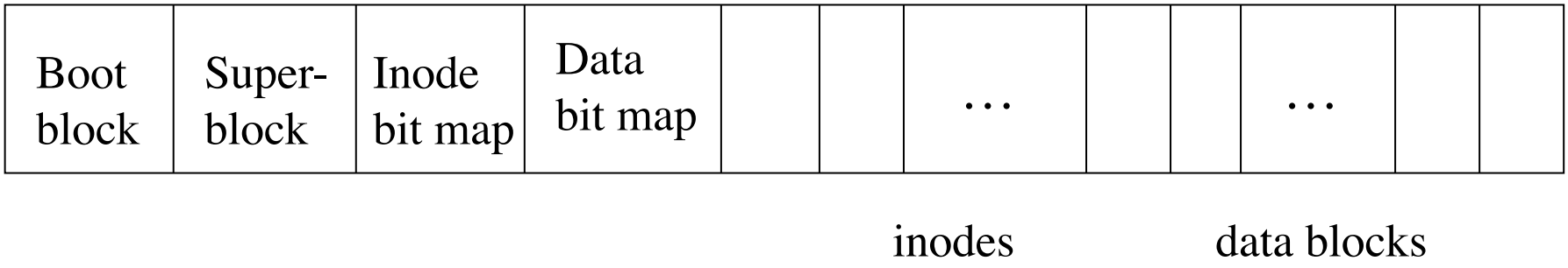


Unix file system

- A hierarchical file system
  - A convenient way of organising data in directories, sub-directories, and so on
- File system protocols
  - Open, read/write, …, close a stream of bytes
- File types
  - Different OSs use different ways to mark the file types
  - But Unix only has two types: executable and non-executable
- Permissions and access in UNIX
  - Each file may be readable, writable, and executable
  - Each file has an owner and belongs to one group
  - The owner can set access rights for himself, group members, and others
  - 9 bits for access rights, 3 special bits (setuid, setgid, sticky), 4 bits to tell the type of file (directory, soft link, etc.)

Implementation of file systems

- Implementation of UNIX file systems
  - Basic units: blocks, default size 4096, can be adjusted when a file system is created.
  - Two structures are created for a file system
    - Superblock (with backups) and inodes
    - Use **dumpe2fs –h /dev/sda**



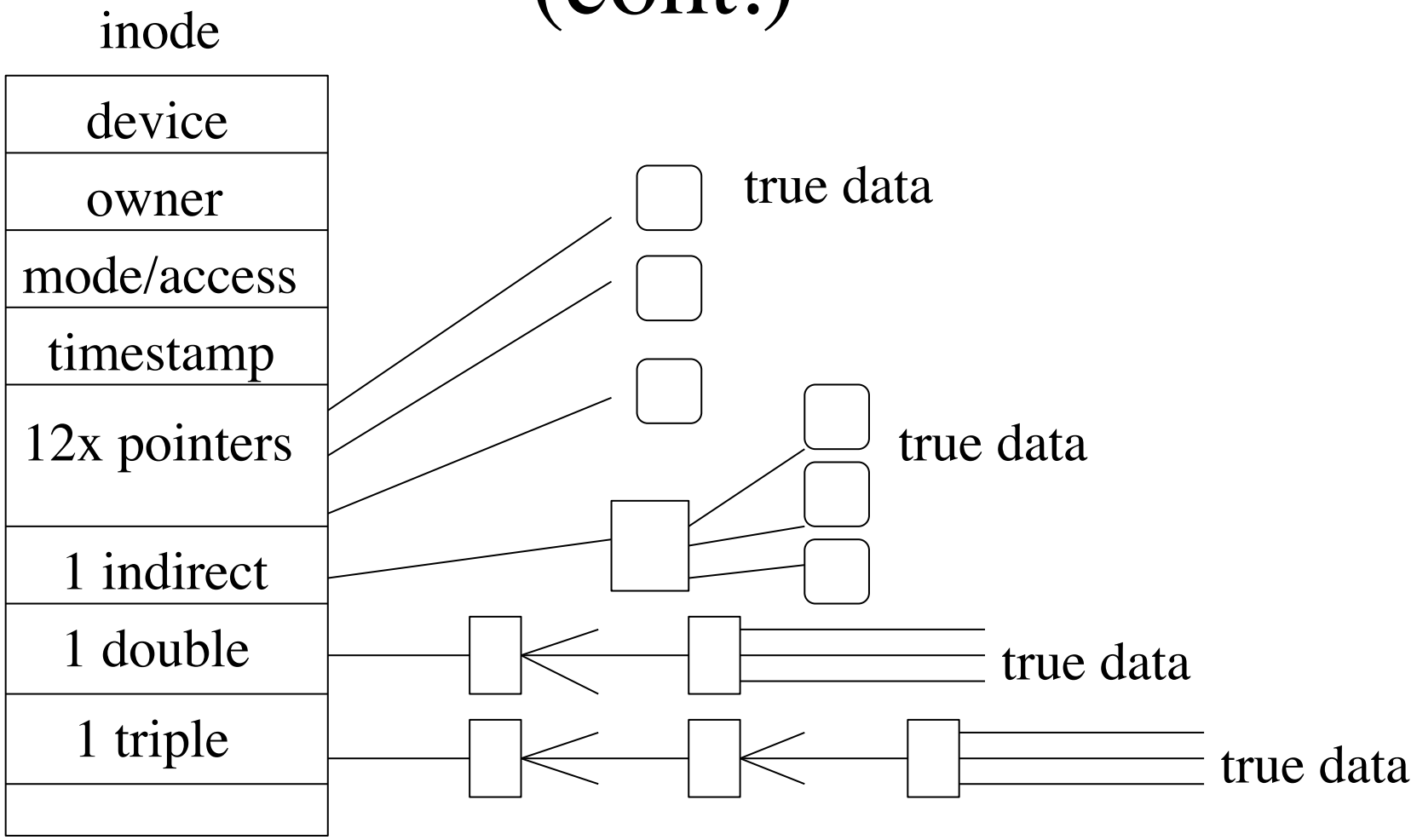
Implementation of file systems
(cont.)

- *Superblock*
  - contains the info on the boundaries of the partition, info about where the inode table (and number of inodes), where data blocks start (and their size), the first inode (root /), etc. If the superblock is lost or damaged, the whole file system would be unreadable. It is important to make superblock backups when a file system is created, e.g ext2. The Linux file system check program **fsck** can do this.

Implementation of file systems (cont.)

- *Inode* (index node) is the data structure which holds the specific data about a particular file. Regardless of how large a file is, there is exactly one inode per file.
- Three ways of addressing data are used in inodes
  - Direct addressing: for files < 48KB
  - Indirect addressing: for files <4 MB
  - Double-indirect: for files < 4GB
  - Triple-indirect: for files < 4TB
- When a file system is created it creates a fixed number (which can be decided by SA) of inodes.
- How to decide the number of inodes?
- Find the information of inode of a file:
  - **ls −li <file>**, **stat <file>**

Implementation of file systems (cont.)

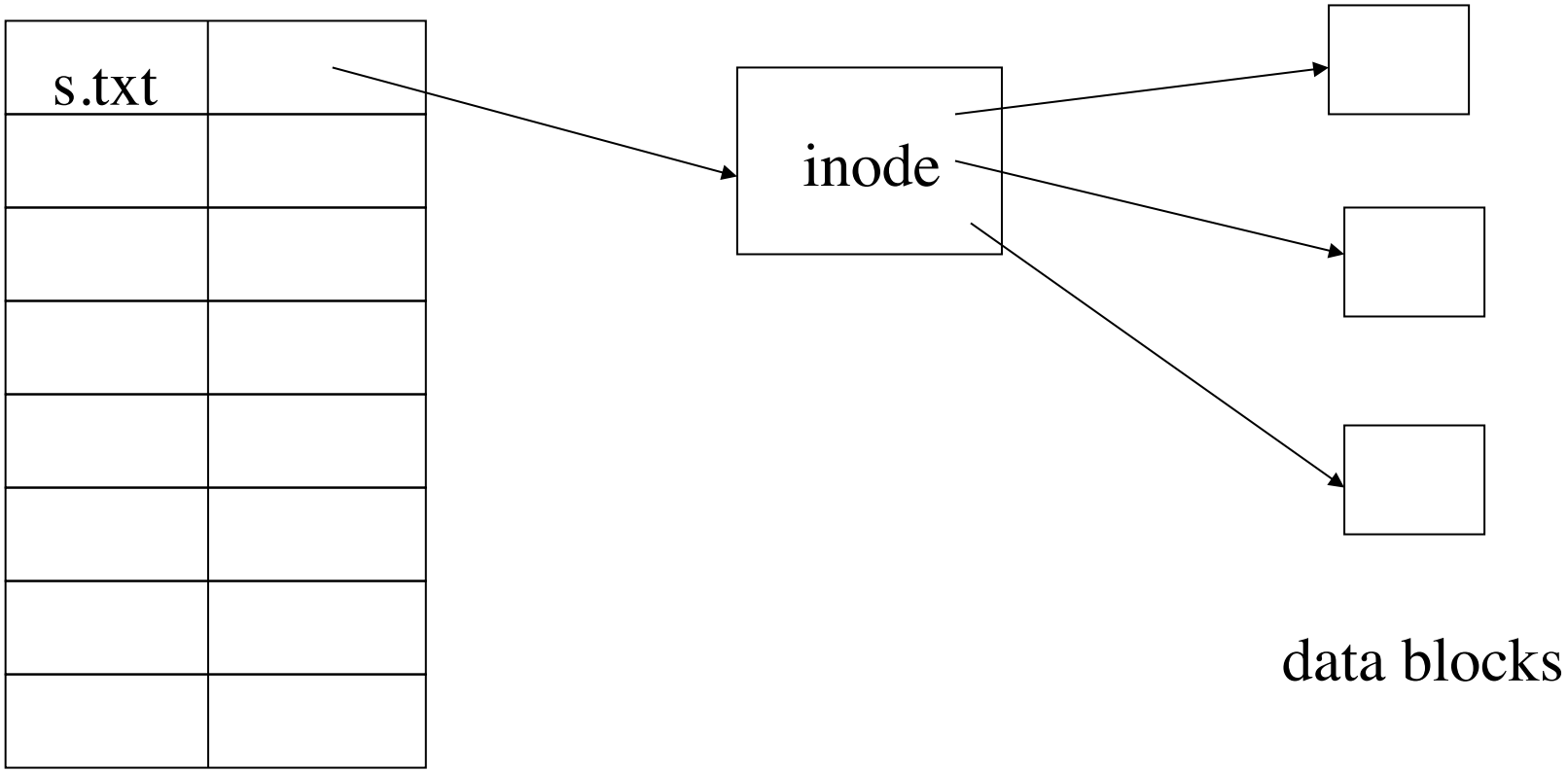


Implementation of file systems (cont.)

- Inodes elements: the device on which the file resides, the user id and the group id of the owner, the type of file and its protection bits, timestamps indicating the last time the file was written to, size of the file, and etc
  - 16 bits are used for protection/access control and other attributes, only 12 bits are changeable by users
  - rwxrwxrwx, sticky bit, setuid bit, setgid bit
  - rwxrwxrwx can be expressed by numbers such as 666 or 777
  - The leading 4 bits are used to represent plain file, directory, soft link, or other purposes (modern OS use 32 bits now!)
- Inodes that are not in use are kept in a doubly linked list called the free list.
- File names are stored in a directory structure (which again is a file), not in the inodes themselves, with pointers in the directory to the appropriate inodes.

Structure of a directory

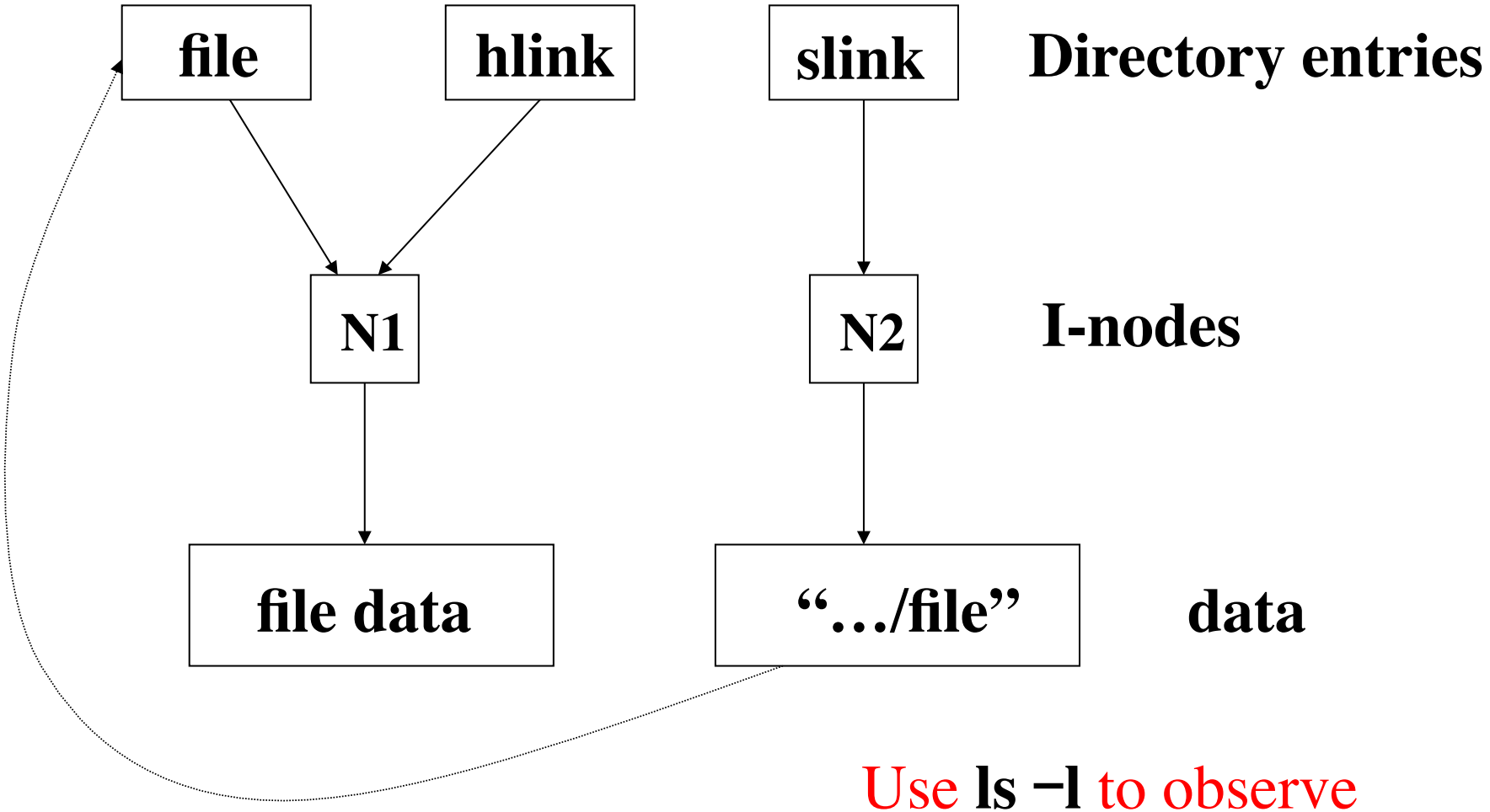
A directory is a file!



Soft/hard links

- Links (use **ln** to create links)
  - Objects which appear in the file system and look just like files
  - Pointers to other files which are elsewhere in the strict hierarchy
- Symbolic (soft) links
  - A small file containing the name of the true file
  - Can point to a non-existing file (broken link)
- Hard links
  - More permanent
  - To delete a file with hard links, all of the hard links must be removed

Soft/hard links (cont.)



Use **ls −l** to observe

Advanced file attributes

- s-bits and t-bit (sticky bit)
  - Apart from the read, write, and execute file attributes, UNIX has three other flags
- Setuid bit
  - **chmod u+s *file***
  - For executable files, this bit tells UNIX that regardless of who runs the program it should be executed with the privilege of the owner of the file.
  - System admins should be aware of this because some root programs may be run by users to allow them limited access to root privileges (e.g. ps). Warning: security holes!

Advanced file attributes

- Setgid bit
  - **chmod g+s *file***
  - Similar to setuid bit except the program is run with the rights of the owner's group
  - In BSD UNIX , System 5 and Linux, if this bit is set on a directory then any new files created in that directory assume the group ownership of the parent directory, not the login group of the user who created the file.
- Sticky bit (t-bit)
  - **chmod +t *dir***
  - A file with this bit set can be kept in memory once loaded into memory for performance reasons (obsolete in modern OS)
  - A directory with this bit set restricts the deletion of files within it (only the owner or superuser can delete her files though the directory is set mode 777)
  - Example - files under /tmp

Special files

- /proc in unix (also /sys in Linux 2.6)
  - Looks like a file system. It is a virtual file system. Contents are created on-demand.
  - An easy way to access kernel data structures such as the process list
- Files in /dev
  - Certain files in the UNIX OS are not really files but “handles” to devices. They are called device nodes.
  - A device node is a way into a device through a file system interface. It is convenient to be able to use normal file operations to access devices
  - ***mknod*** is used to make i-nodes for devices
  - Major and minor numbers are used for devices
    - Check **/usr/src/linux/Documentation/devices.txt** for details on device major and minor numbers
  - Example: /dev/fd0 - the first floppy drive.

Backups

- Commands for backups
  - **tar**, **dd**
  - **rsync** (for incremental backups; refer to the example below)
    - https://linuxconfig.org/how-to-create-incremental-backups-using-rsync-on-linux
- Full and incremental backup
  - Full backup is a copy of every file
  - An incremental backup is a copy of only those files which have changed since the last backup was taken
  - Incremental backup works on the concept of levels, 0,1,2,3,4,…9. At each incremental backup level, you back up everything that has changed since the previous backup at the same or lower level (higher number)
- Backup policies
  - Daily backup, Weekly backup, Monthly backup

Journaling

- Logs changes to a journal before actually writing to the main file system
  - Journaling can have a severe impact on performance because it requires that all data be written twice

Summary

- Access control models
- Unix/Linux access control bits
- Advanced file attributes (s-bits and t-bit)
- Difference of soft link and hard link
- What is incremental backup in comparison with full backup?