File System Interface

- What is a file?
- How can a file be accessed?
- How can a file be found?
- How are files protected?
- What is file consistency?
  - » Handling multiple users for a single file
  - » Spreading updates around

What Is a File?

- A file is a group of bits that share some logical relationship to one another
  - » Relationship defined by user or OS
  - » People can disagree on what goes into a single file
- A file usually has a contiguous logical address space
- Files can contain
  - » Data
    - – Numbers (binary, character, other)
    - – Text (documents, program source, e-mail, etc.)
    - – Structured information (database)
  - » Program (executable code, script)
  - » Some of each...

Internal File Structure

- Files may have internal structure, decided on by
  - » Operating system
  - » User program
- Structure can include
  - » Flat file / no structure (simple string of bits or bytes)
  - » Records
    - – Fixed length (e.g., *n* bytes per record)
    - – Variable length (e.g., one line of text per record)
  - » Complex structure
    - – Formatted word processing or spreadsheet document
    - – Executable file with relocation info, symbol table, etc.
- Simulate complex structure by using flat file with special data structures in file

File Attributes

- OS maintains information for each individual file
- Information maintained includes
  - » Name: human-readable pointer to the file
  - » Type: needed for systems with different file types (Mac, etc.)
  - » Location: pointers to file data on disk
  - » Size: number of bytes in the file
  - » Protection: information about who can use the file
  - » Owner: information about the file’s owner (for accounting)
  - » Timestamps: time of creation, last modification, last usage (perhaps others…) for accounting & security
- Information stored in the directory structure
  - » Structure maintained on disk
  - » Structure updated whenever information changes

Operations on Files

- General file operations
  - » Create: make a new file
  - » Delete: delete an existing file
  - » Open: find the appropriate directory entry on disk, and copy the entry to memory
  - » Close: write the directory entry for the file to disk, updating it
  - » Stat: get information about a particular file
  - » Other functions to query & update file information
- Access
  - » Read: read data from a file
  - » Write: write data to file
  - » Truncate: remove all data from the file

File Types

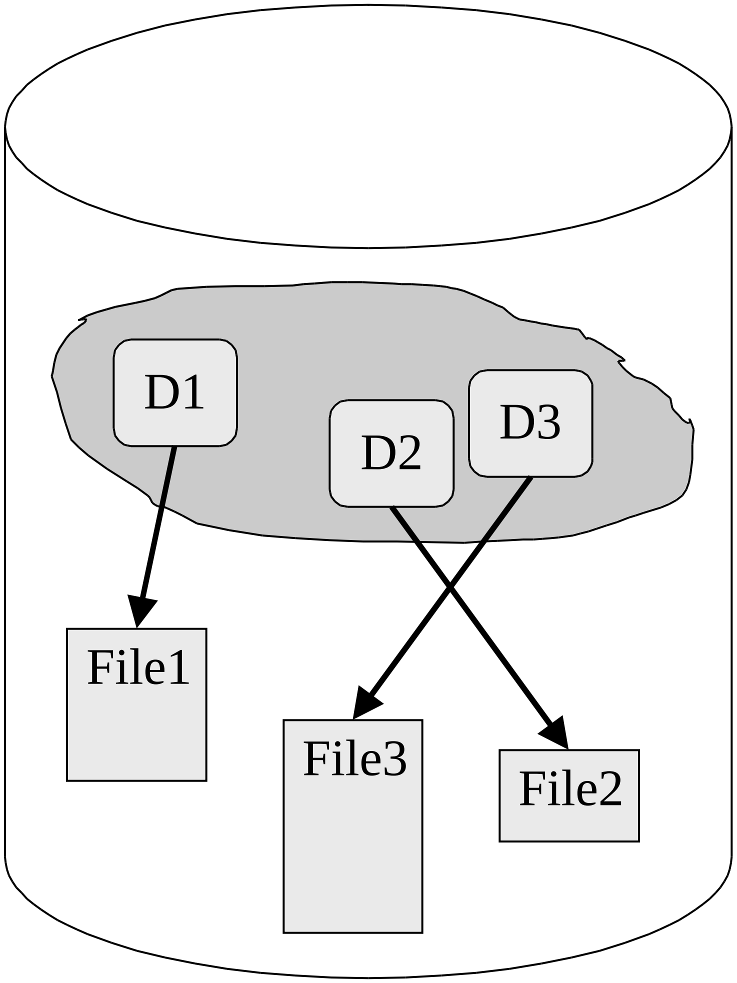
- File types are used to identify the kind of data in a file
- Types can be indicated by
  - » File name / required: MS-DOS
  - » File name / optional: Unix
  - » Type & creator info: Mac
- Some systems require type information (MS-DOS, Mac)
  - » Type included in name (.exe means executable on MS-DOS)
  - » Type included in file information, but not in name: Mac has creator & type information that shows up as different icons
- Other systems have optional type information (Unix)
  - » Text files on Unix need not end in .text or .txt
  - » Program suffixes done by convention
  - » C compilers will often accept files not ending in .c

Accessing Files

- Sequential access
  - » File data accessed linearly: from start to finish
  - » OS keeps track of current position in file
  - » Next read or write starts at current position
  - » Current position updated to end of previous operation
  - » Current position may be reset to start of file
- Direct (random) access
  - » Repositioning can be done to any point in file
  - » Position supplied either
    - – With read or write request
    - – As a separate call to OS (lseek)
  - » Often, OS supports both sequential & random access
- Generally, sequential access is more common and faster

Directory Structures

- The directory is a structure on disk that contains information about all the files on the disk
  - » Structure may be complex
  - » Information about a single file may be saved in several locations
    - – Human-readable name in one place
    - – Other information (size, etc.) elsewhere
- Directory & files reside on disk
- Backups are kept on tape



Information Stored in a Directory

- Directory stores information *about* files
  - » Information is also called metadata
  - » Includes information about files discussed earlier
- In Unix, information is split into two pieces
  - » File name stored in “directory”
  - » Other information stored in inode
    - – File type
    - – Data location
    - – Current & maximum length
    - – Date last accessed (for archival purposes)
    - – Date last updated (for backup purposes)
    - – File owner (accounting & quota)
    - – Protection bits

Directory Operations

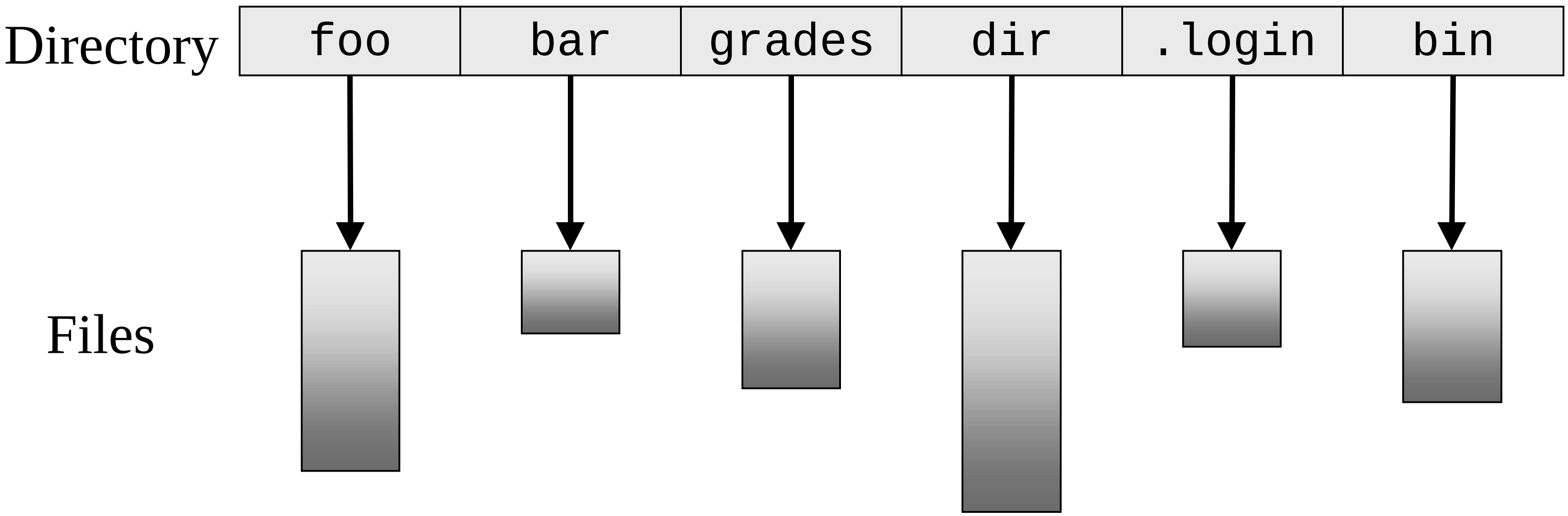
- Locate a file
  - Specified by name
  - Other parameters (size, etc.)
- Create a file: make an entry in the directory
- Delete a file: remove the directory entry
- List a directory
- Rename a file
- Traverse all (or a subset) of the file system

Directory Organization

- Efficiency
  - » Allow fast file location & directory entry updating
  - » Consume as little space as possible
- User convenience
  - » Naming allows multiple names for a single file
  - » Naming allows multiple files to have the same name, albeit for different users
- File grouping
  - » Allow files to be grouped together by common properties
  - » Groups determined by users (directories in Unix)
  - » Groups determined by system (all files modified since yesterday)

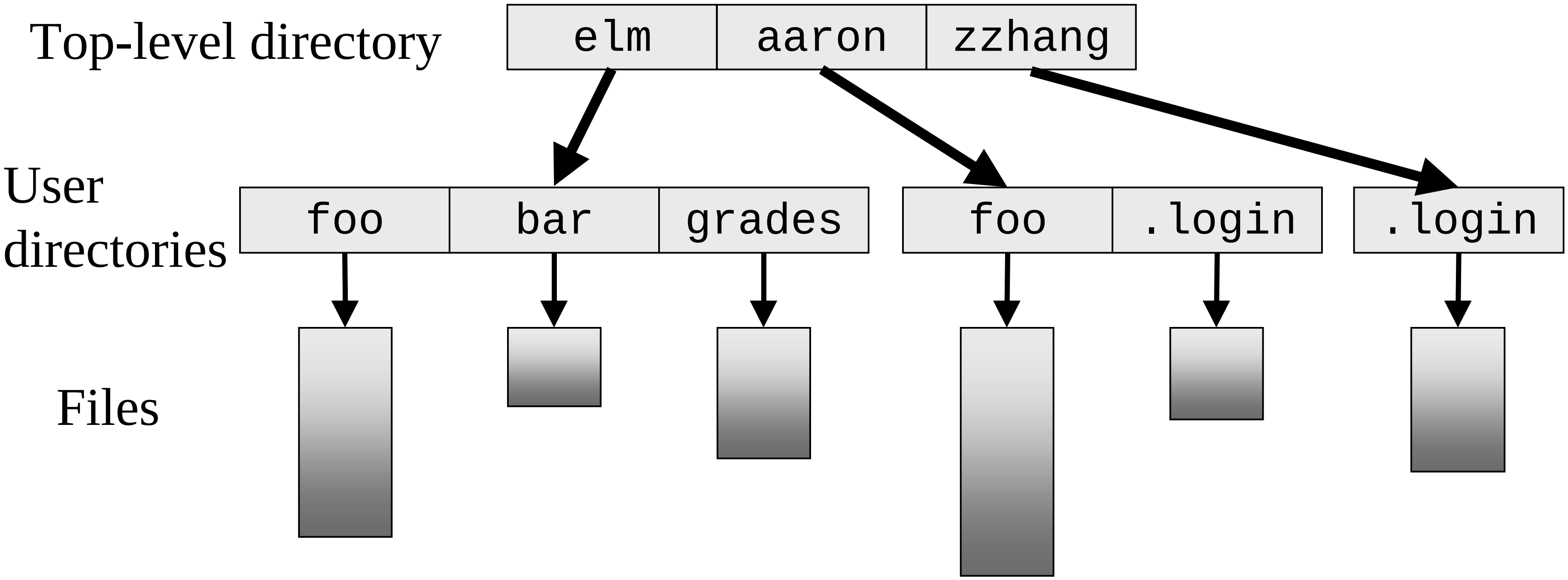
Single-Level (Flat) Directory

- Single directory for all users
- Problems
  - » File names are global: users can’t reuse the same name
  - » Grouping: no way of associating related files



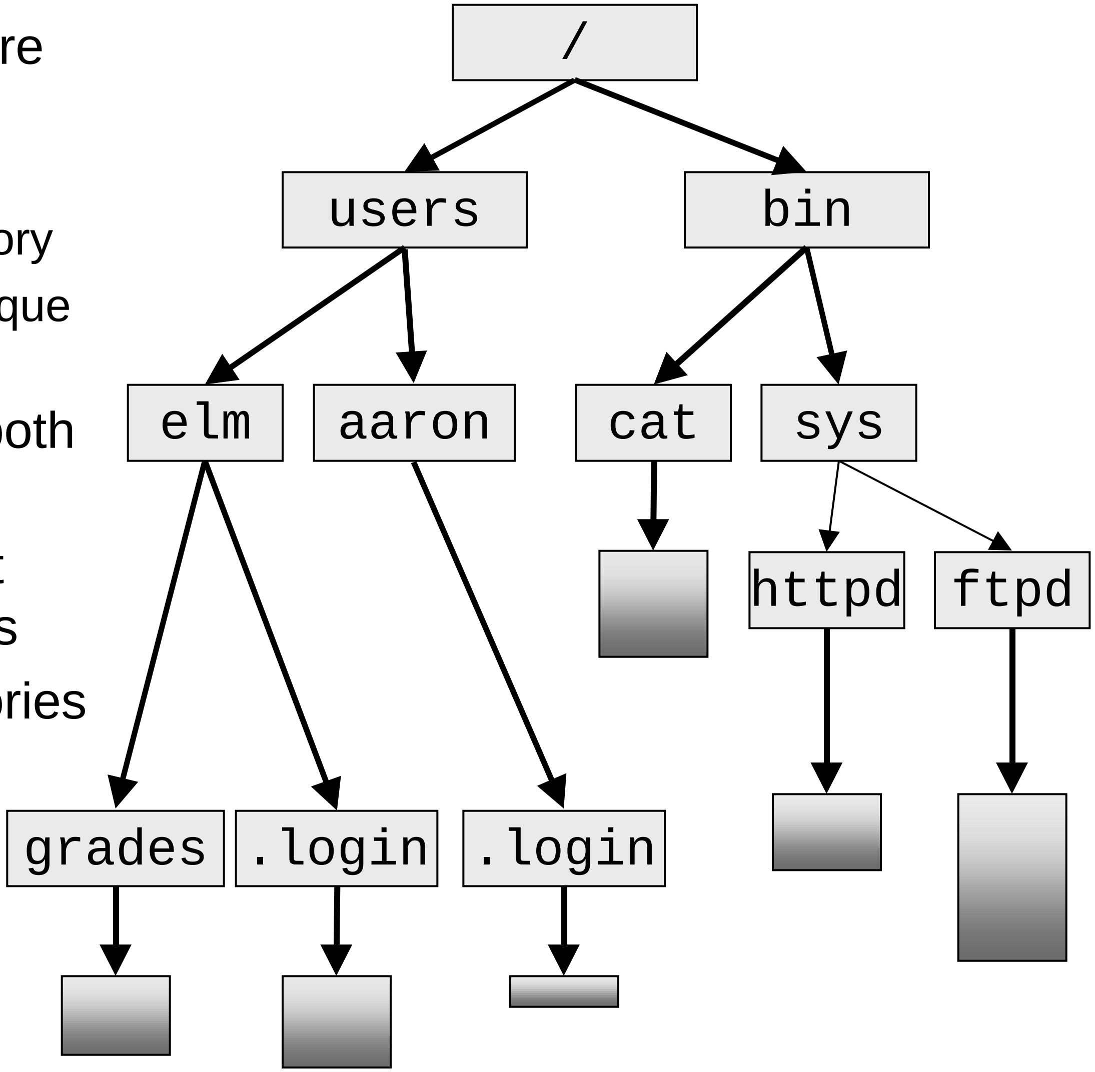
One Directory Per User

- Separate directory for each user
  - » Single directory for any user
  - » Naming problem resolved: different users can reuse a name
  - » Still no way of grouping files
- Introduces the concept of “path name”



Tree-Structured Directory

- Expand two-level structure to “infinite” levels: tree structure
  - » Grouping by sub-directory
  - » Name must only be unique within sub-directory
- Directories can contain both directories and files
- Current directory: default directory for file accesses
- Path name: list of directories along path to file

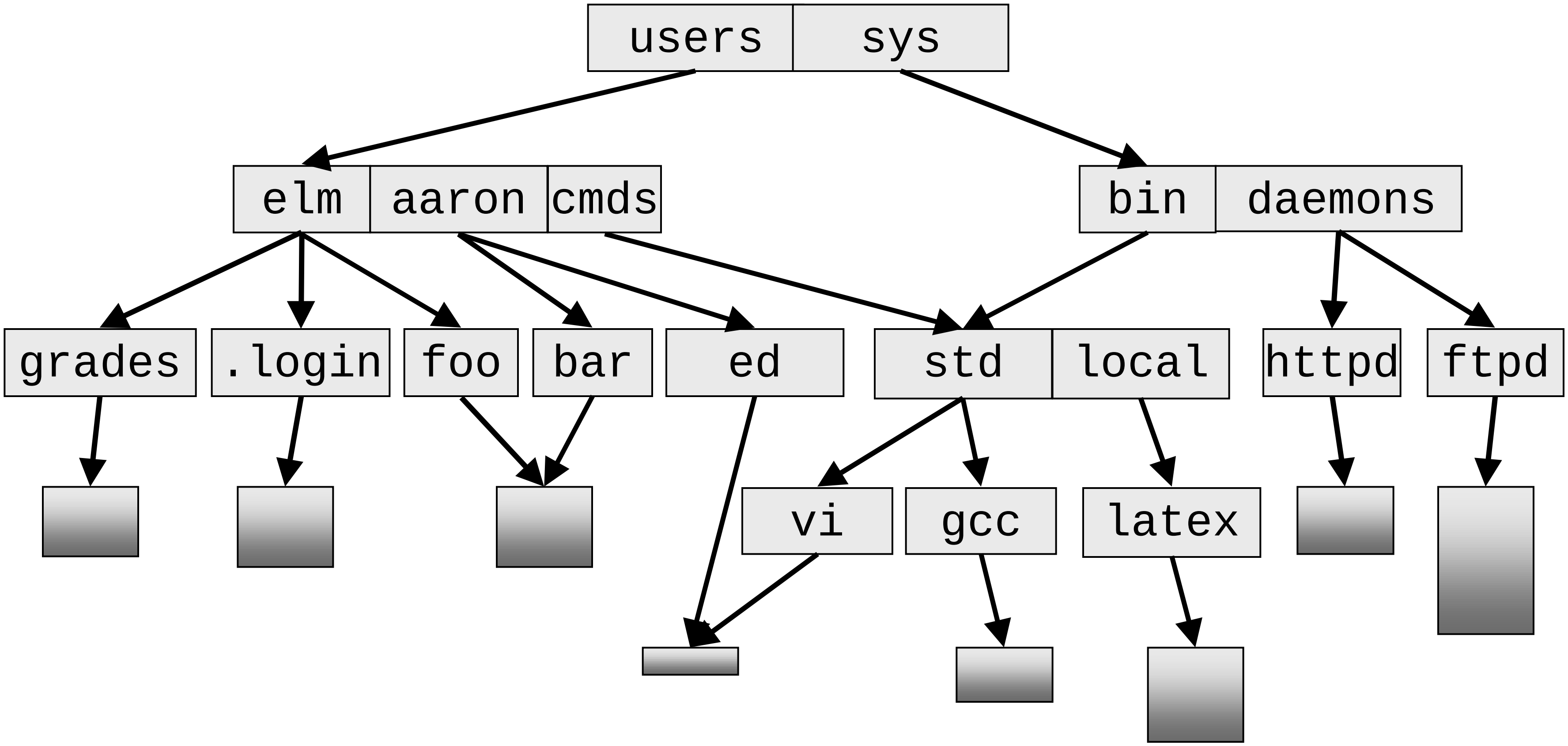


More on Tree-Structured Directories

- Path names may be specified as
  - » Relative: start in current directory
  - » Absolute: start at root (top of tree)
- Files can be created anywhere in the tree if path is fully specified
- Non-empty directories can’t be deleted directly
  - » First, delete all the files
  - » Next, delete the directory itself
  - » Follow this procedure recursively to delete an entire sub-tree
- Problem: files can still have only a single name: attached at a single place in the tree structure

Acyclic Graph Directories

- Want more than one name per file (or sub-directory!)
- Solution: allow multiple pointers to each file or sub-directory
- Cycles prevented: links must point lower in the tree

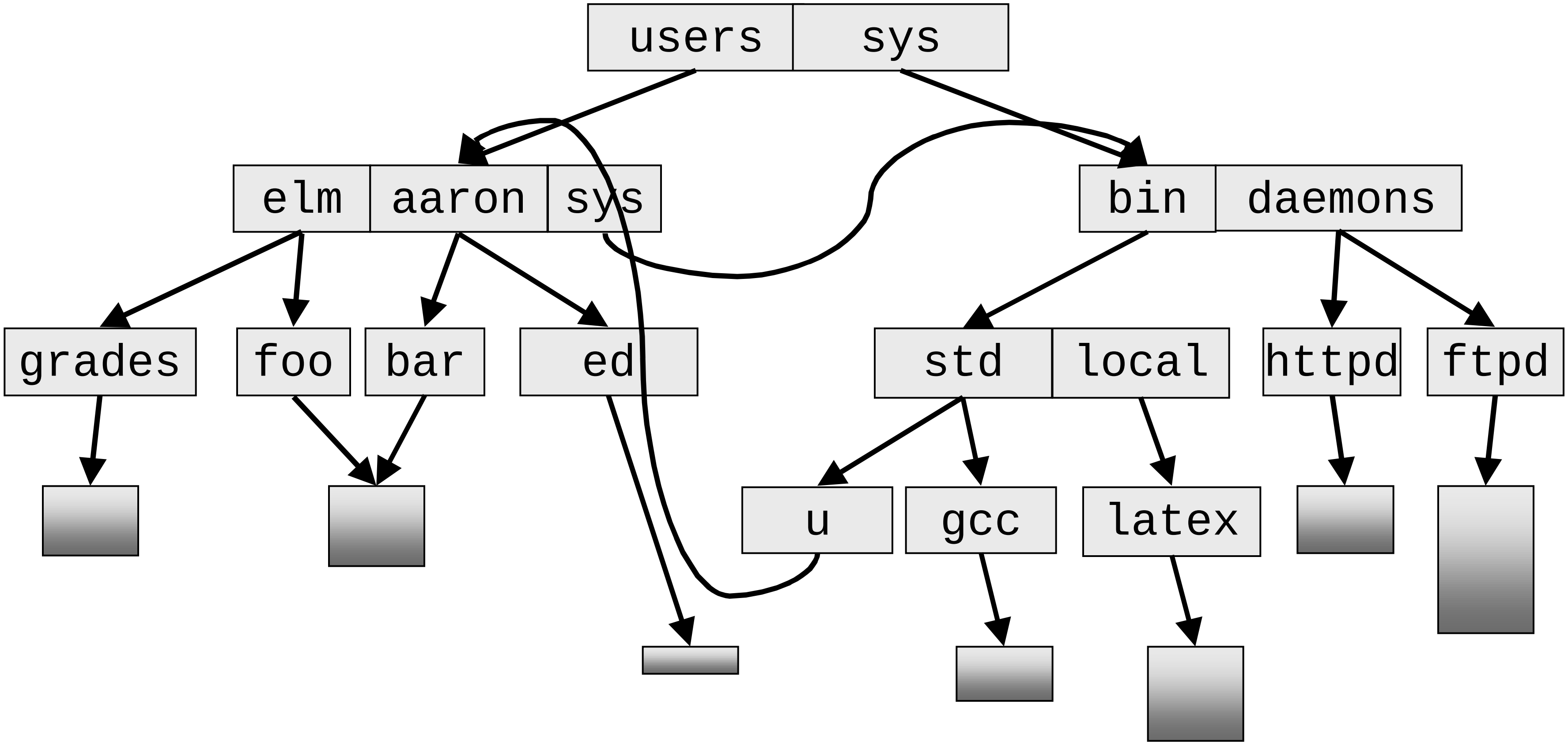


Issues with Acyclic Graph Directories

- Aliasing: files and directories can have multiple different names
  - » Pointers to a file or directory at several places in graph
  - » Keep track of number of pointers to a given object
- Removing a link: file system must decide whether to remove object pointed to by link
  - » Keep an array (or list) of all pointers to a given object
    - – Easy to figure out who points to the object
    - – May lead to inefficiency: variable-sized allocations
  - » Keep count of links to this object
    - – Single integer -> delete object when count reaches 0
    - – Must keep count consistent: increment count and allocate link atomically

General Graph Directories

- Links can go from anywhere to anywhere
- Cycles are possible and must be dealt with



Dealing With Cycles in Directories

- Problem: cycles make several things difficult
  - » Lookups can go into infinite loops
  - » Unlinking can have difficulty telling whether an object is actually unreferenced
- Solutions:
  - » Allow links to files but not to subdirectories
    - – Eliminates cycles
    - – Reduces flexibility
  - » Use standard garbage collection algorithms to delete unreferenced objects
  - » Try to detect cycles when a link is added - slow
  - » Use "soft links": link is just a name, not a pointer
    - – No need to garbage collect objects
    - – May result in "dangling" links...

Protection in File Systems

- File owner (initially, its creator) should be able to control
  - » What can be done to a file
  - » Who can do things to a file
- Access types for files are
  - » Read
  - » Write
  - » Execute (files only)
  - » Append (files only)
  - » Delete
- Directories also allow
  - » List
  - » Create
- May need “meta-permissions”: ability to grant permissions to others

Protection in Unix

- Directories are stored as files, so only one protection mechanism
- Each file has a user (owner) and group associated with it
  - » User ID and group ID stored in inode
  - » Text translation of UID & GID looked up in system files
- Each file has three sets of protection bits associated with it
  - » One set of rights for each of user, group, and “others”
  - » Possible rights are:
    - – Read
    - – Write
    - – Execute (lookup for directories)
  - » Only owner (or root) can change the permissions
  - » Owner (or root) can change owner or group

Protection in AFS

- AFS has more flexible model than Unix
  - » Unix requires predefined groups (created by system admin)
  - » Unix allows only one group per file
- AFS uses access control lists (ACLs)
  - » List can be as long as desired
  - » List may include individual users and/or predefined groups
  - » Lists only attached to directories, and apply to files within the directory
- Permissions given for
  - » Read, write, insert (create), delete, lookup / list
  - » Lock (k): lock files within the directory
  - » Administer (a) directory permissions

File Consistency

- Multiple users may access a file at the same time
  - » If only reads, no problem: file doesn’t change
  - » If at least one writer, problems crop up
- Problems with writers
  - » One writer, many readers
    - – Readers see changes “eventually”
    - – Issue: how long is eventually?
    - – Either get old or new data
    - – After file is closed, changes “stick”
  - » Many writers (possibly with readers)
    - – Writers could make conflicting changes
    - – Order of changes is very important, but could be difficult to synchronize (remember synchronization?)
    - – This situation is very uncommon

Examples of File Consistency

- Unix semantics
  - » Writes to an open file are immediately visible to others who have the file currently open
  - » Option: single pointer advanced by all processes that have the file open (single file image)
- Session semantics
  - » Writes to a file are not seen by other processes that currently have the file open
  - » Writes to a file are sent to the file when it is closed, and are visible to any processes that open the file after that point