CSC 369

Week 8: File System Intro

University of Toronto, Department of Computer Science

File Systems

- Last few lectures talked about page replacement algorithms
  - Use disk for temporary storage of paged-out data
- Today we'll talk about file systems – persistent storage of data
  - Files
  - Directories
  - Sharing
  - Protection
  - File System Layouts
  - File Buffer Cache
  - Read Ahead

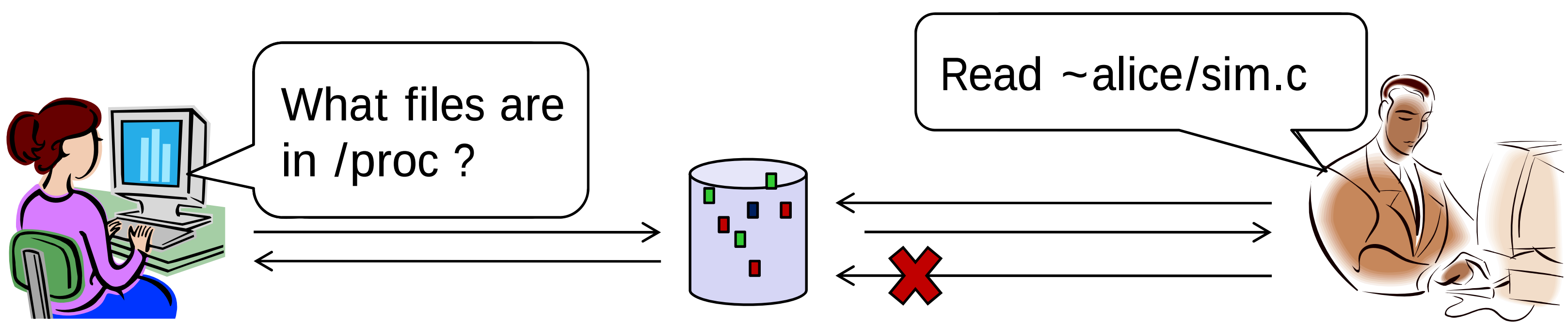
File (Management) Systems

- Provide long-term information storage
- Requirements:
    1. *Store very large amounts of information*
    2. *Information must survive the termination of process using it*
    3. *Multiple processes must be able to access info concurrently*

- Two views of file systems:
    - User view – convenient logical organization of information
    - OS view – managing physical storage media, enforcing access restrictions

File (Management) Systems

- Implement an abstraction (files) for secondary storage
- Organize files logically (directories)
- Permit sharing of data between processes, people, and machines
- Protect data from unwanted access (security)



Conceptual File Operations

- Creating a file
  - Find space in file system, add entry in *directory* mapping file name to location (and attributes)
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file
  - May erase the contents (or part of the contents) of a file while keeping attributes

Handling operations on files

- Involves searching the directory for the entry associated with the named file
  - when the file is first used actively, store its attribute info in a system-wide open-file table; the index into this table is used on subsequent operations $\Longrightarrow$ no searching

Unix example (open, read, write are syscalls):

```
main() {
    char onebyte;
    int fd = open("sample.txt", "r");
    read(fd, &onebyte, 1);
    write(STDOUT, &onebyte, 1);
    close(fd);
}
```

Open File Table
<console device>
…
sample.txt
…
…

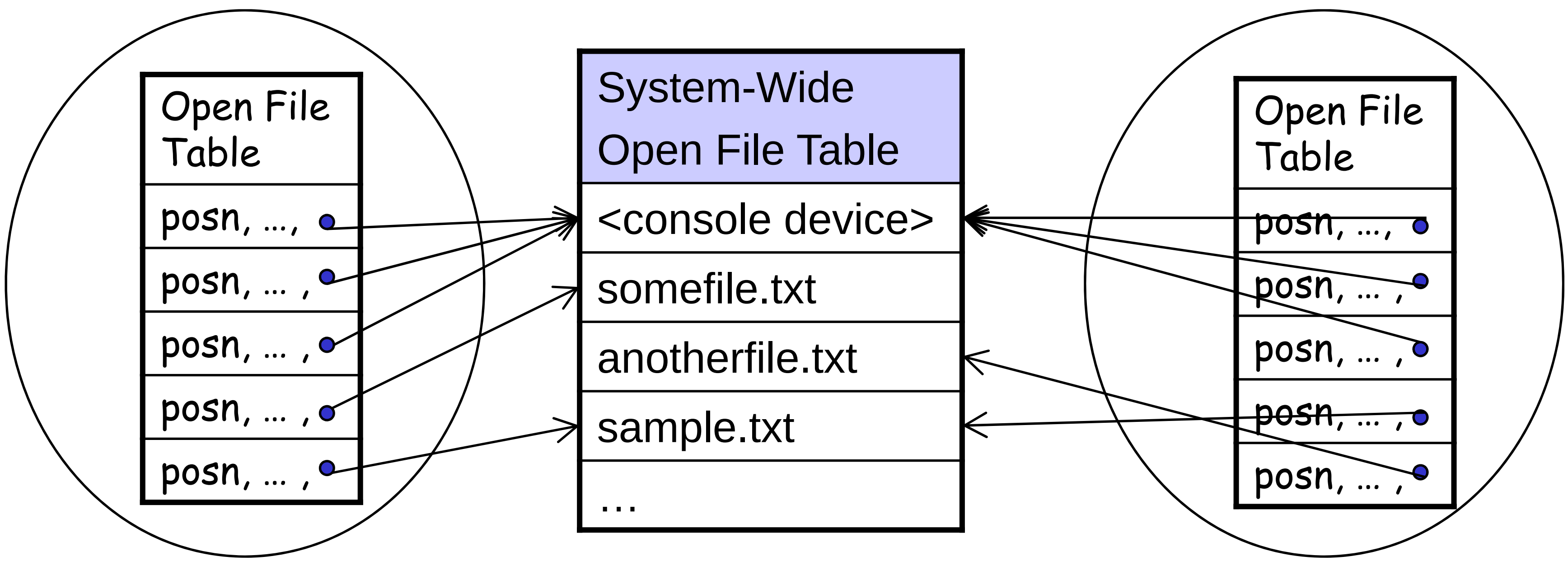
File Sharing

- File sharing is incredibly important for getting work done
  - Basis for communication and synchronization

- Two key issues when sharing files
  - Semantics of concurrent access
    - What happens when one process reads while another writes?
    - What happens when two processes open a file for writing?
  - Protection

Shared open files

- There are actually 2 levels of internal tables
  - a per-process table of all files that each process has open (this holds the current file positions for the process)
  - each entry in the per-process table points to an entry in the system-wide open-file table (for process independent info)

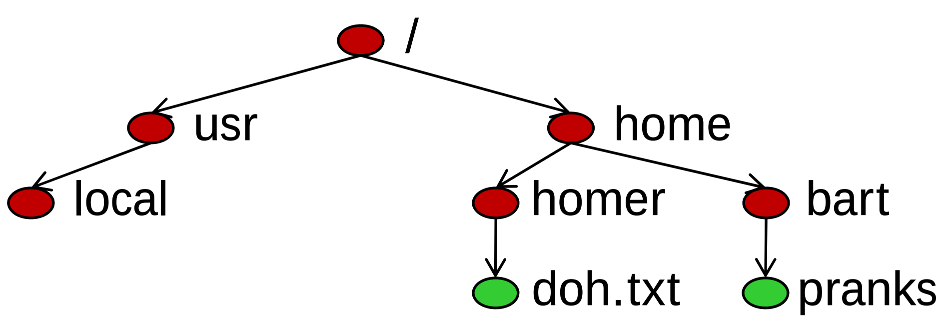


File Access Methods

- General-purpose file systems support simple methods
  - Sequential access – read bytes one at a time, in order
  - Direct access – random access given block/byte number
- Database systems support more sophisticated methods
  - Record access – fixed or variable length
  - Indexed access
- What file access method does Unix, NT provide?
- Older systems provide more complicated methods
  - Modern systems typically only support simple access

Directories

- Directories serve multiple purposes
  - For users, they provide a structured way to organize files
  - For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk
  - Also store information about files (owner, permission, etc.)
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, /home, …)



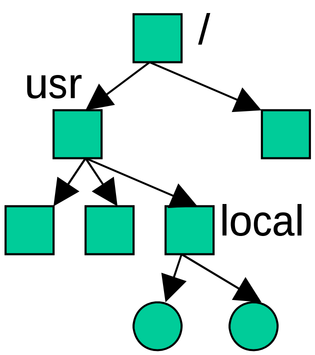
Directory Structure

- A directory is a list of entries – names and associated *metadata*
  - Metadata is not the data itself, but information that describes properties of the data (size, protection, location, etc.)
- List is usually unordered (effectively random)
  - Entries usually sorted by program that reads directory
- Directories typically stored in files
  - Only need to manage one kind of secondary storage unit

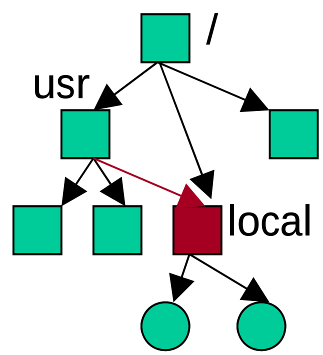
Possible Organizations

- single-level, two-level, tree-structured
- acyclic-graph directories: allows for shared directories
  - the *same* file or subdirectory may be in 2 different directories

Tree-structured:



Acyclic graph:



The directory "local" has two paths, /local and /usr/local

Directory Implementation

- Option 1: Linear List
  - Simple list of file names and pointers to data blocks
  - Requires linear search to find entries
  - Easy to implement, slow to execute
    - And directory operations are frequent!
- Option 2: Hash Table
  - Add hash data structure to linear list
  - Hash file name to get pointer to the entry in the linear list

File Links

- Sharing can be implemented by creating a new directory entry called a *link* : a pointer to another file or subdirectory
  - Hard links
    - Second directory entry identical to the first
  - Symbolic, or soft, link
    - Directory entry refers to file that holds “true” path to the linked file

Issues with Acyclic Graphs

- With links, a file may have multiple absolute path names
  - traversing a file system should avoid traversing shared structures more than once
- Sharing can occur with duplication of information, but maintaining consistency is a problem
  - E.g. updating permissions in directory entry with hard link
- Deletion: when can the space allocated to a shared file be deallocated and reused?
  - somewhat easier to handle with symbolic links
    - deletion of a link is OK; deletion of the file entry itself deallocates space and leaves the link pointers dangling
  - keep a reference count for hard links

File System Implementation

How do file systems use the disk to store files?

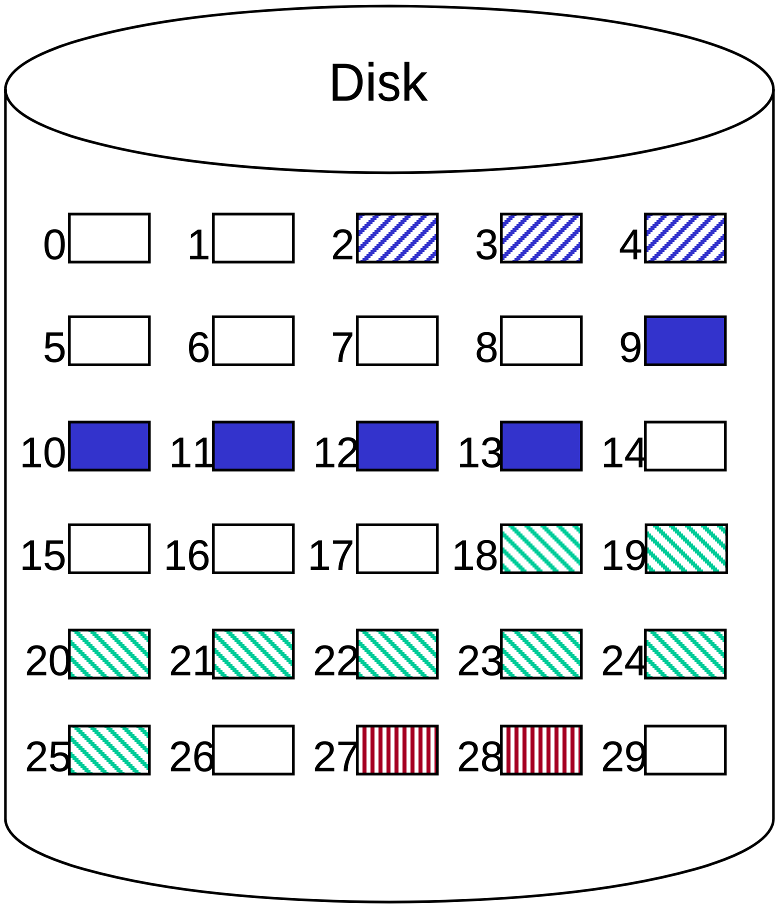
- File systems define a block size (e.g., 4KB)
  - Disk space is allocated in granularity of blocks
- A “Master Block” determines location of root directory (aka *partition control block, superblock)*
  - Always at a well-known disk location
  - Often replicated across disk for reliability
- A free map determines which blocks are free
  - Usually a bitmap, one bit per block on the disk
  - Also stored on disk, cached in memory for performance
- Remaining disk blocks used to store files (and dirs)
  - There are many ways to do this

Disk Layout Strategies

- Files span multiple disk blocks
- How do you find all of the blocks for a file?

1. Contiguous allocation
    - Like memory
    - Fast, simplifies directory access
    - Inflexible, causes fragmentation, needs compaction
2. Linked, or chained, structure
    - Each block points to the next, directory points to the first
    - Good for sequential access, bad for all others
3. Indexed structure (indirection, hierarchy)
    - An "index block" contains pointers to many other blocks
    - Handles random better, still good for sequential
    - May need multiple index blocks (linked together)

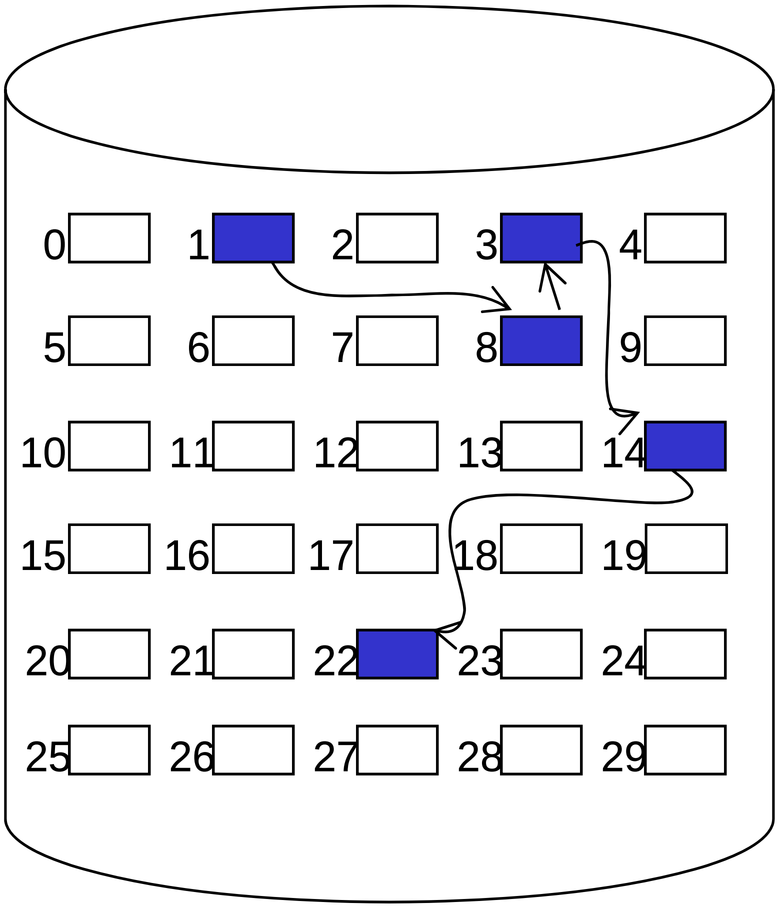
Contiguous Allocation



directory

File Name	Start Blk	Length
File A	2	3
File B	9	5
File C	18	8
File D	27	2

Linked Allocation



directory

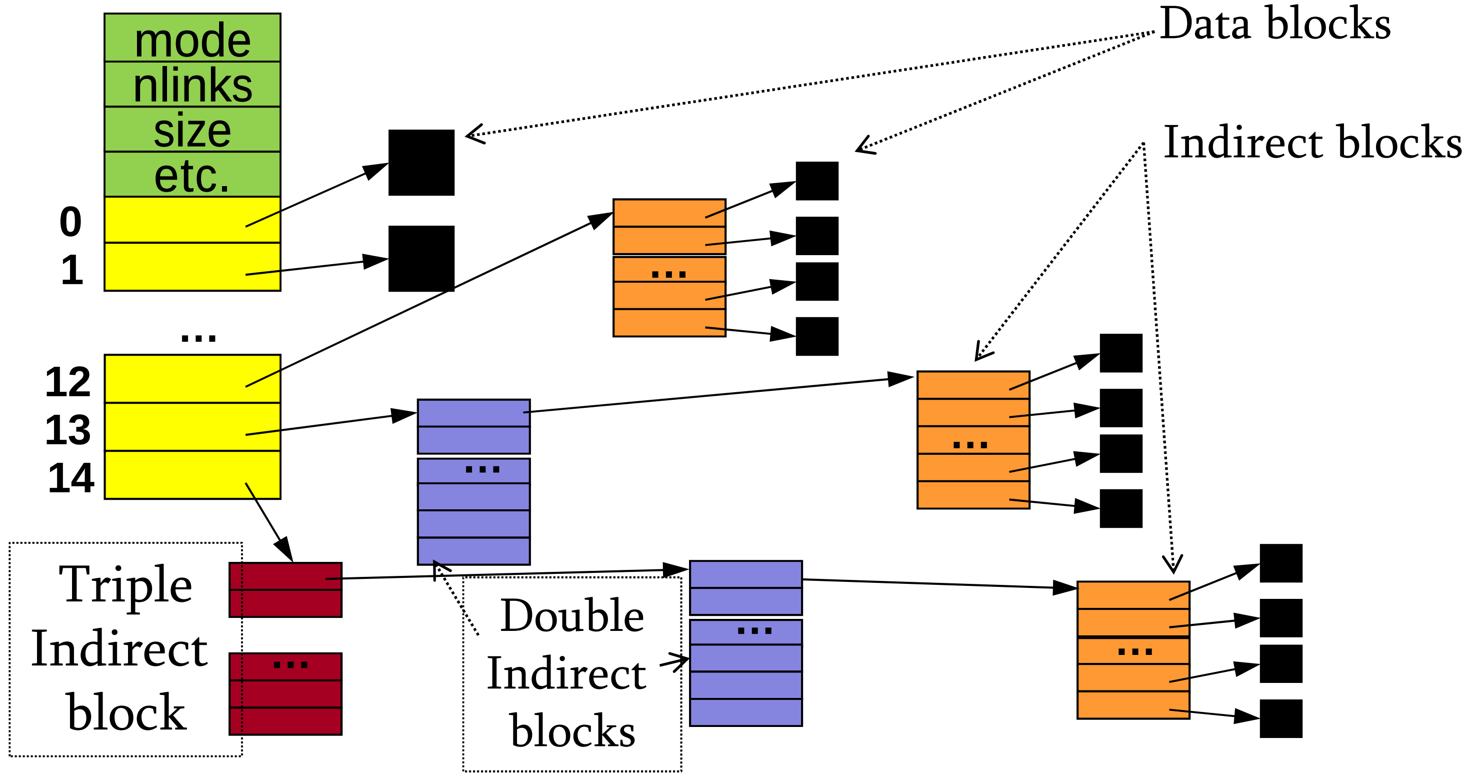
File Name	Start Blk	Last Blk
…	…	…
File B	1	22
…	…	…

Indexed Allocation: Unix Inodes

- Unix inodes implement an indexed structure for files
- All file metadata is stored in inode
  - Unix directory entries map file names to inodes
- Each inode contains 15 block pointers
  - First 12 are *direct* block pointers
    - Disk addresses of first 12 data blocks in file
  - Then single indirect block pointer
    - Address of block containing addresses of data blocks
  - Then double indirect block pointer
    - Address of block containing addresses of single indirect blocks
  - Then triple indirect block pointer

Example UNIX Inode

- Inodes are smaller than disk blocks
  - Unix System V – 64 bytes
  - Ext2 Linux file system – 72 bytes



Path Name Translation

- Let’s say you want to open “/user/homer/doh.txt”
- What does the file system do?
  - Open directory “/” (the root, well known, can always find)
  - Search for the entry “user”, get location of “user” (in directory entry)
  - Open directory “user”, search for “homer”, get location of “homer”
  - Open directory “homer”, search for “doh.txt”, get location of “doh.txt”
  - Open file “doh.txt”
- Systems spend a lot of time walking directory paths
  - This is why open is separate from read/write
  - OS will cache prefix lookups for performance
    - /a/b, /a/bb, /a/bbb, etc., all share “/a” prefix

Unix Inodes and Path Search

- Unix Inodes are not directories
- They describe where on the disk the blocks for a file are placed
  - Directories are files, so inodes also describe where the blocks for directories are placed on the disk
- Directory entries map file names to inodes
  - To open “/user”, use Master Block to find inode for “/” on disk and read inode into memory
  - inode allows us to find data block for directory “/”
  - Read “/”, look for entry for “user”
  - This entry gives/locates the inode for “user”
  - Read the inode for “user” into memory
  - The inode says where first data block is on disk
  - Read that block into memory to access the data in the file

Operations on Directories

- Search
  - find a particular file within directory
- Create file
  - add a new entry to the directory
- Delete file
  - remove an entry from the directory
- List directory
  - Return file names and requested attributes of entries
- Update directory
  - Record a change to some file's attributes

Example Directory Operations

Unix

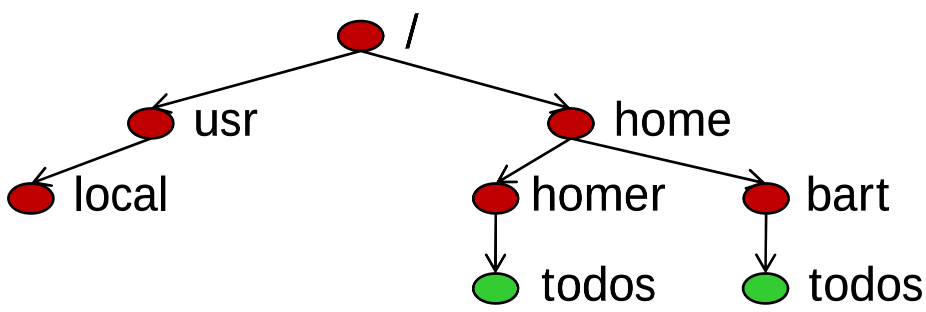
- Directories implemented in files
  - Use file ops to create dirs
- C runtime library provides a higher-level abstraction for reading directories
  - opendir(name)
  - readdir(DIR)
  - seekdir(DIR)
  - closedir(DIR)

Windows NT/XP

- Explicit dir operations
  - CreateDirectory(name)
  - RemoveDirectory(name)
- Very different method for reading directory entries
  - FindFirstFile(pattern)
  - FindNextFile()

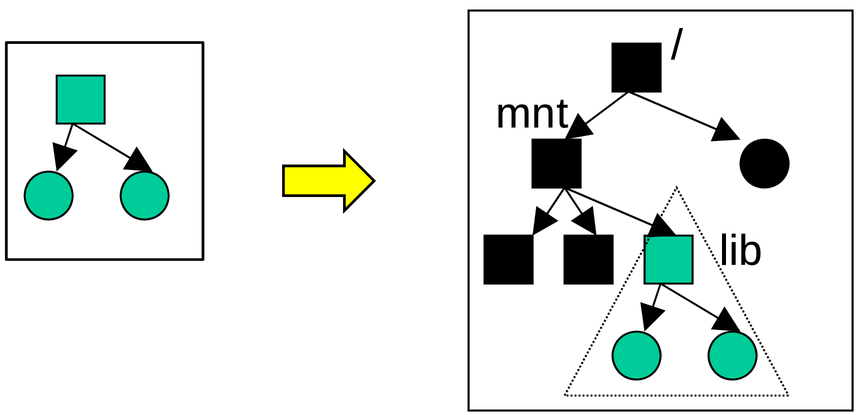
Current Working Directory

- Most file systems support the notion of a *current working directory*
  - Printed by "pwd" command on Unix
  - Relative path or file names specified with respect to current directory
  - Absolute names start from the root of directory tree
  - Special names: "." == current directory, ".." == parent
  - Examples: If current directory is bart then "todos" refers to /home/bart/todos. Homer's "todos" could be referred to as "/home/homer/todos", or "../homer/todos"



File System Mounting

- File system "namespace" may be built by gluing together subtrees from multiple physical partitions
  - Each device (or disk partition) stores a single file system
  - Mount point is an empty directory in the existing namespace
  - Parent directory notes that a file system is mounted at dir



Summary

- File systems – Interface, structure, basic implementation
- Files
  - Operations, access methods
- Directories
  - Operations, using directories to do path searches
- Sharing
- Protection
- File System Layouts
  - Unix inodes

Next (next) week…

- File systems details - “the good stuff” :)
  - Unix inode structure
  - More details on space management, implementations
  - Disk characteristics and file system optimizations
  - Disk scheduling