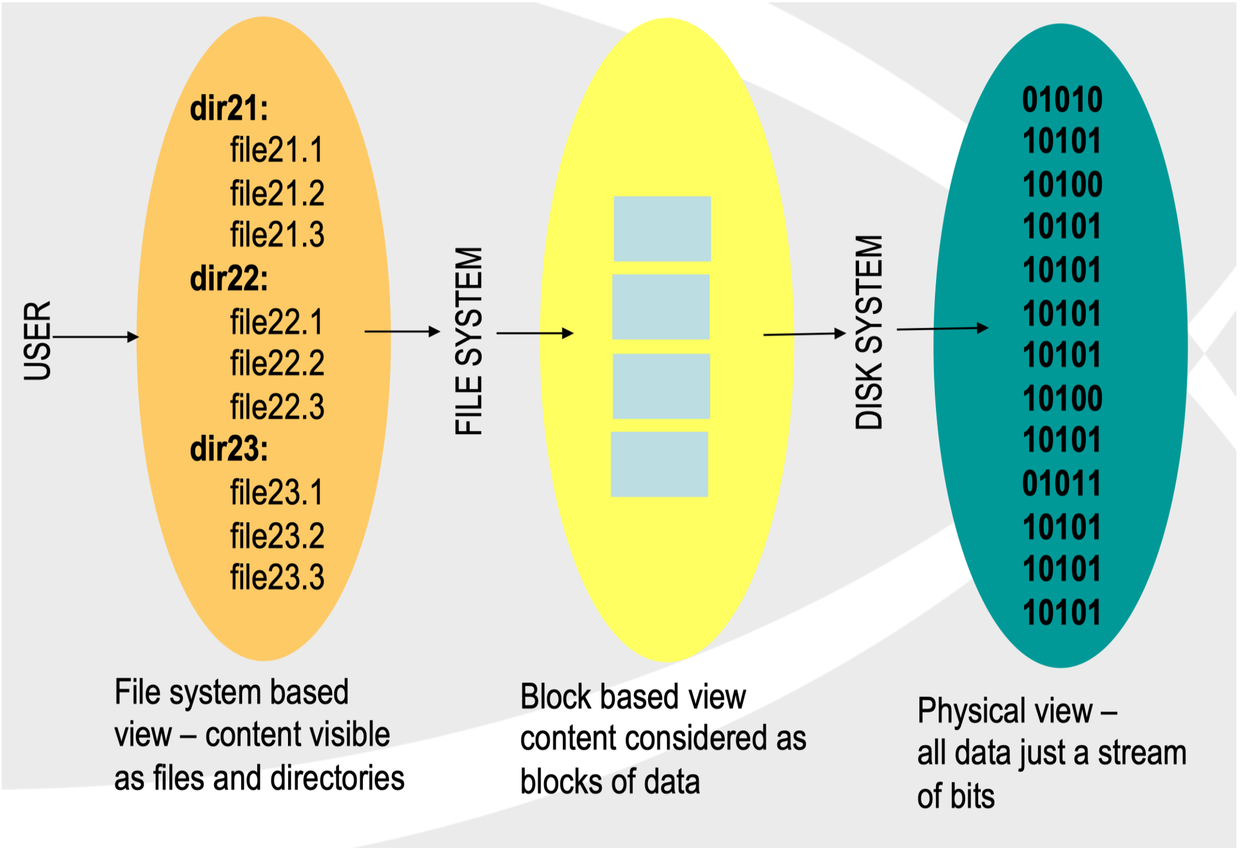
File Systems

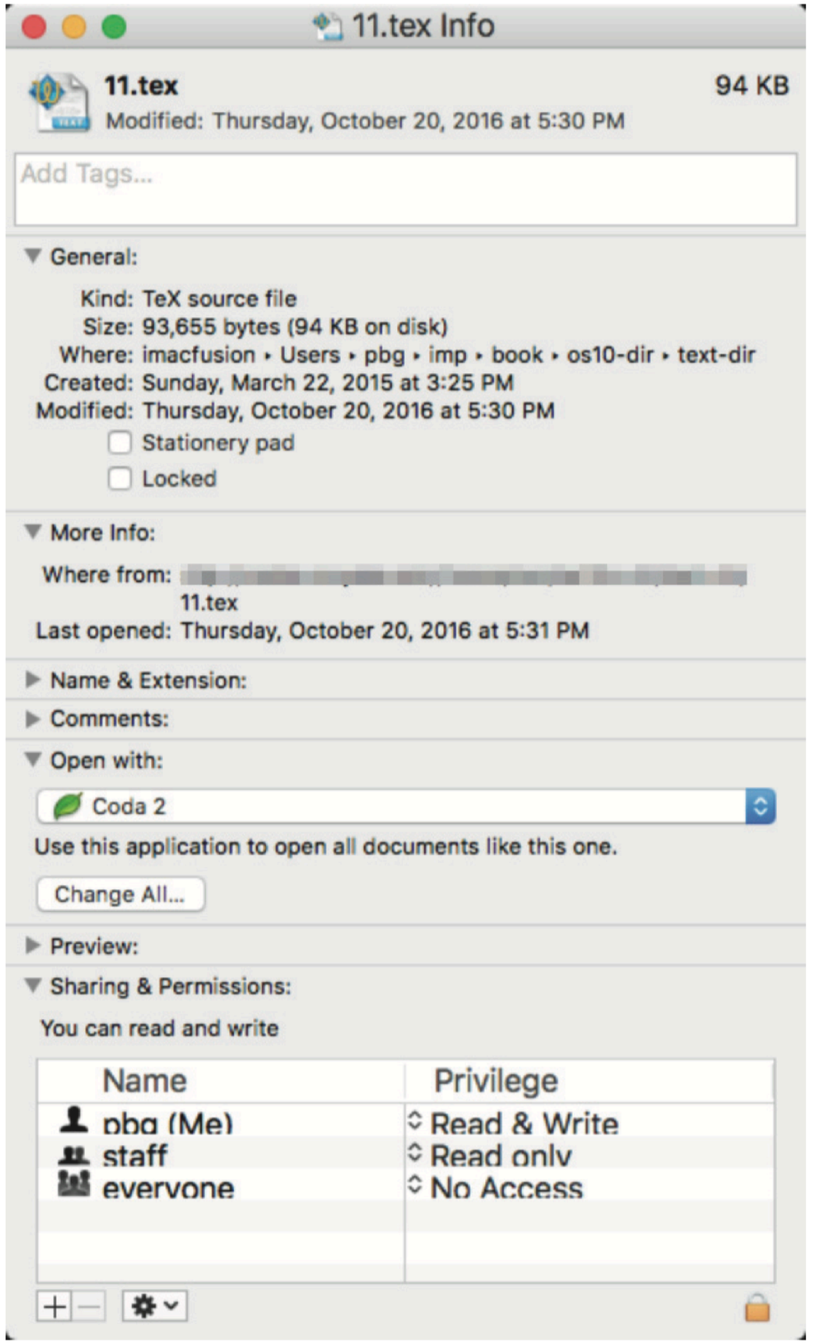
File Concept

- File is a named collection of related information that can be accessed by addressing
- File can have heterogeneous types of data
- Creator is responsible for defining the embedded data types, convention for access, etc
- File can be text, executable, media, data, etc
- File has two different views:
  - Logical view - view as users, applications, see it
  - Physical view - as it actually resides in storage



File Attributes

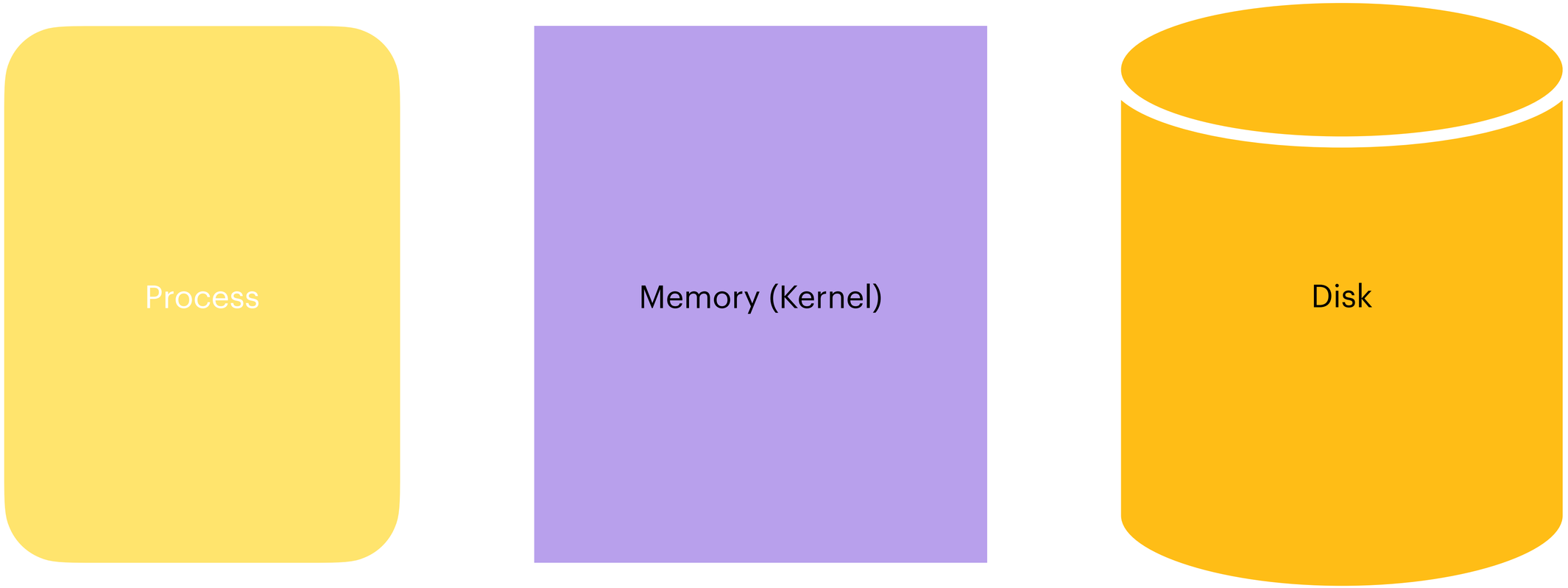
- File attributes vary from OS to OS, but following are common:
- Name: Symbolic name in human readable form
- Identifier: A unique tag for identification within the file system
- Location: Information about the device holding the file, and location inside the device
- Size: Current size of the file, number of blocks allocated
- Protection: Access control information, reading, writing permissions
- Timestamps: last accessed, created, modified, etc



File Operations

- OS provides many operations on a file, here are the common ones (implemented using system calls)
- **Creating a file**: Allocate space in the storage, make an entry in the directory for the newly created file, check permission whether file can be created at the current directory location
- **Opening a file**: This operation returns a file handle that can be used by other file operations. This makes the file operation stateful. You open the file, then you know the file exists, you have the permission to read or write, etc. OS will also setup a read and write pointer that points to the location in the file for reading and writing, respectively.
- **Writing a file**: You are given a file handle returned by the open operation along with the data. OS will write the data and move and internal write pointer (offset) so the next write will go to the location after the data that just got written.

File Operations



File Operations

- **Reading a file**: A file handle, memory buffer, and size of data to be read from the file. Read is going to copy data from the file to the memory buffer up to the size of data specified in the call. The data copying (in the file) starts at the current location of the read pointer. There are no data to be read (pointer at the end) we return an End-of-file. On successful read, the file pointer is moved.
- **Repositioning within a file**: Current file pointer (read, write, or common) is repositioned to the given value.
- **Deleting a file**: Search the directory and remove the entry if the permission permits the process to do so. If the file entry has other links, the file allocation remains, otherwise the storage is recovered.
- **Truncating a file**: file length is reset to zero, all blocks holding user data are recovered

File Operations: Required Information

- File pointer: We need to keep track of where in the file the read/write operations are taking place. This information is unique to each process.
- File open count: We need to track the processes that are opening and closing the files. This way we can reallocate the entry in the Open-File Table when it is not needed anymore.
- Location of the file: we need to keep track of disk blocks that are holding the data we need. This way we can avoid reading the directory all the time
- Access rights: A file is opened for a specific type of access. OS would deny any other type of access on an open file

File Types

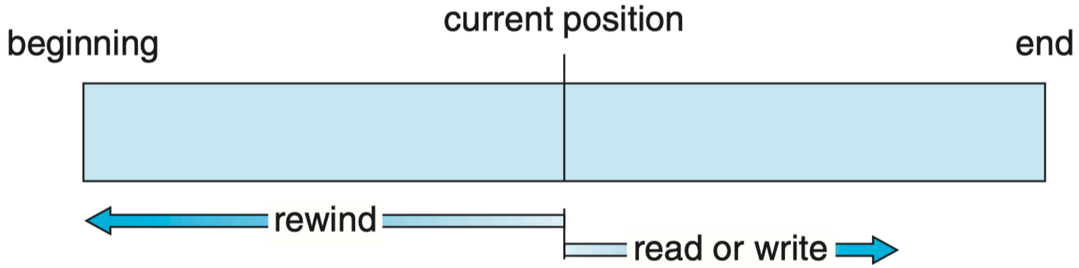
- OS recognizes some file types: executable files, others are not of much interest to OS
- Many file types exist for use by different application — compilers, linkers, etc
- File types are denoted using file name extensions
- In UNIX, file types are recognized using a finger printing technique called magic numbers

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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Access Methods

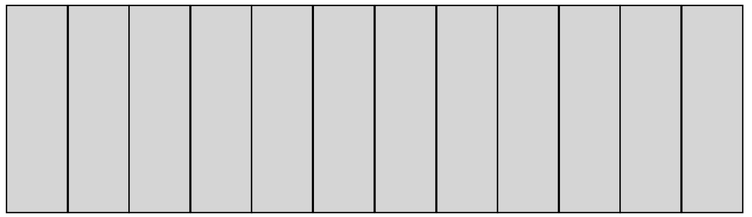
Sequential Access Method

- Information in the file is accessed in order, one record (byte stream) at a time
- Editors and compiler access files this way
- Provided two methods: read_next(), and write_next()
- Read_next() — reads the next record in the file and advances the pointer automatically so the portion after the read one will be next
- Write_next() — appends to the end of the file and advances to the end of the newly written portion



Direct Access

- File is made up of fixed length logical records
- Allows the program to read and write any record in the file
- We can emulate sequential access using direct access files
- Direct access files can be used to build other types — indexed files, etc



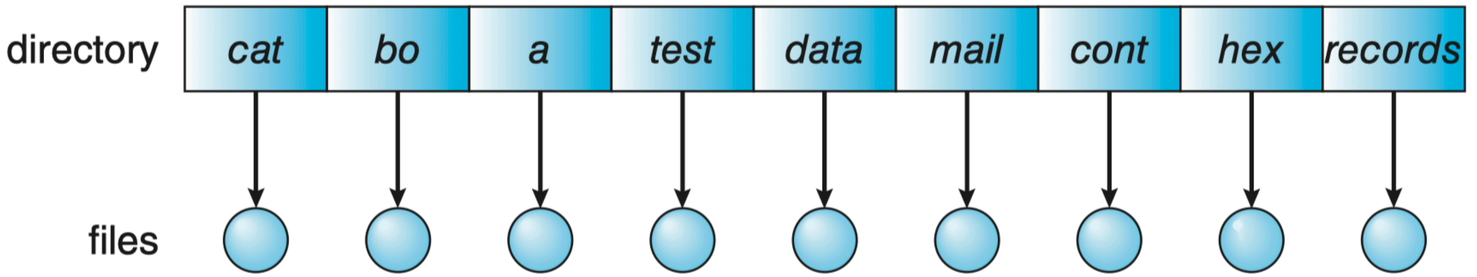
sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Directory Structure

- Directory is a symbol table translating the file names to file control blocks - file control blocks will tell us where to find the data in the file
- Directory must allow the following operations:
- Create a file: new files added to the directory
- Delete a file: deleted file can leave hole in the directory
- List a directory: we need to list the contents of a directory
- Rename a file: change name and other attributes of a file
- Traverse the file system:

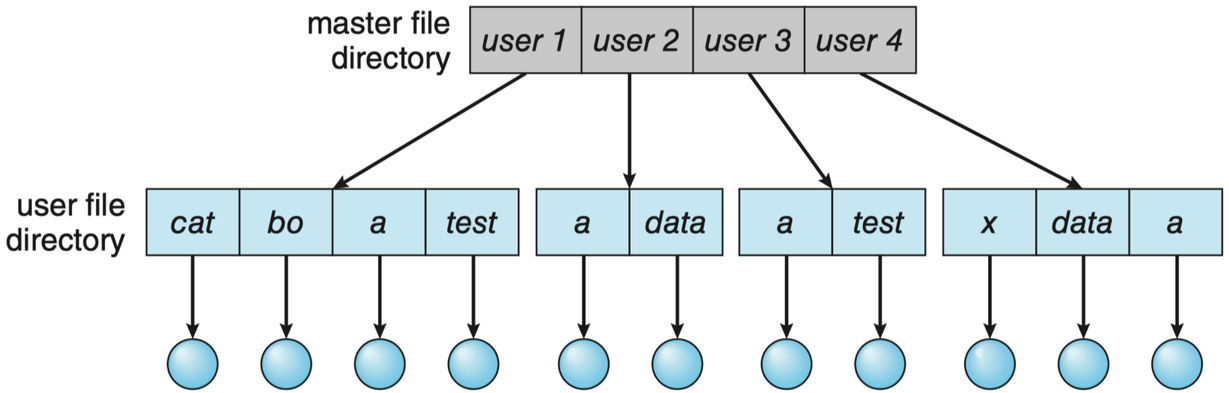
Single-Level Directory

- Simplest directory structure
- Files must have unique names
- Could be useful for embedded devices



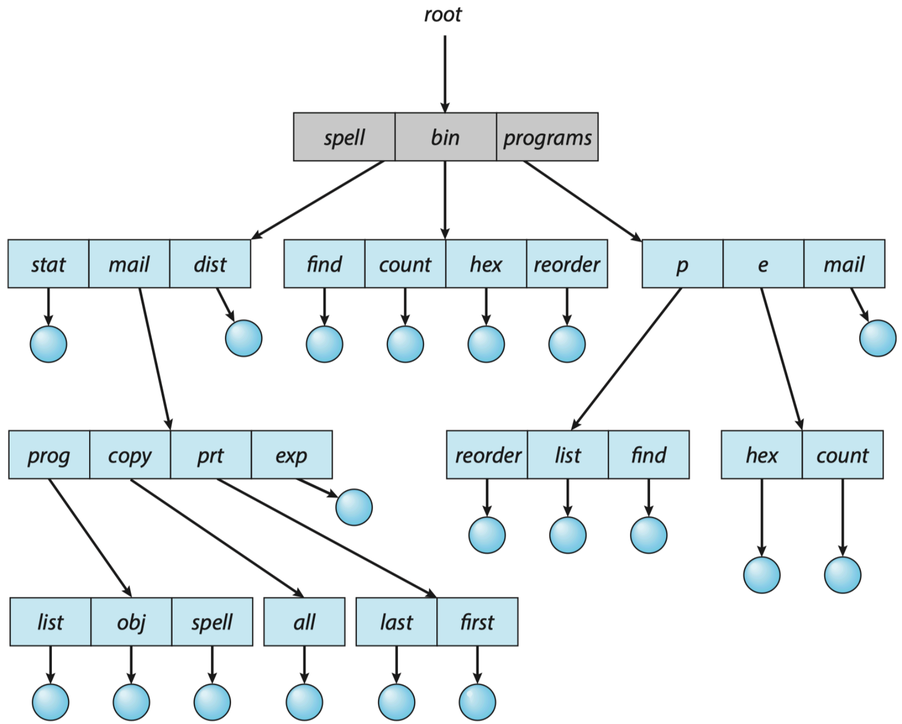
Two-Level Directory

- In a two-level directory, each user has a single level user file directory (UFD)
- UFD lists files for a particular user
- Master file directory (MFD) links all the users
- The MFD is indexed by user name and points to the different UFDs
- Access control for inter-user file access is not well developed
- Dealing with system files such as compilers, loaders, etc is also not well supported



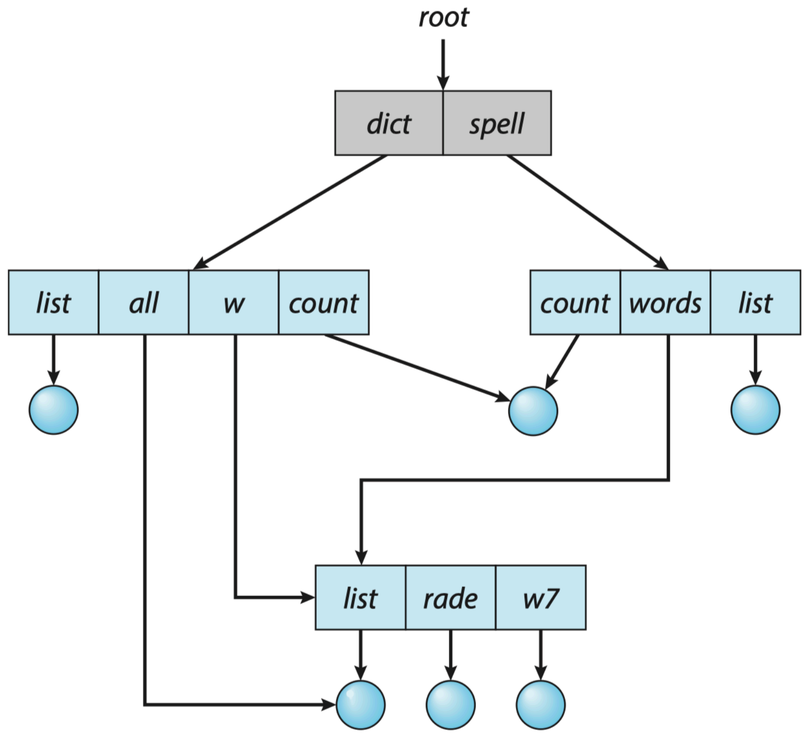
Tree-Structured Directories

- Directory contains a set of sub-directories and files
- Directories are implemented as files with a special (OS defined) structure
- Process has a current directory - location in the tree
- Files and directories can be references with relative paths with respect to current directory or absolute paths
- Directories can be deleted recursively in some OSes and others do not allow recursive delete



Acyclic Graph Directories

- Acyclic graph directories allows sharing of files and directories - not copies - links to the same instance
- Modification of a file or directory will be visible from all locations that link to the instance
- Challenge: to keep cycles out
- Simple idea: link to only files (used by hard links in UNIX/Linux)
- Soft links in UNIX/Linux allow linking to directories - can lead to cycles - and result in chaos!



Memory-Mapped Files

- Leverage virtual memory to map files into a process' memory space
- No need to use read() and write() system calls to access file - just reading and writing memory is sufficient
- Writes to a file may not be synchronously written to disk - in the memory - until the memory mapped file is closed or memory flushed to disk in page replacement