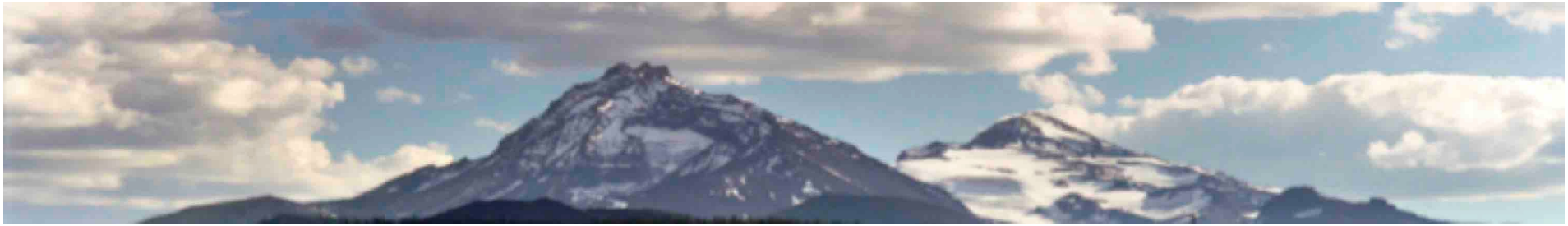


CS 415
Operating Systems
File Systems – Theory

Prof. Jieyang Chen
Department of Computer and Information Science
Fall 2025

UNIVERSITY OF OREGON

Logistics

- Project 3 due extended to 12/9
- Final exam next Thursday (12/11)
- Study guide will be released this week

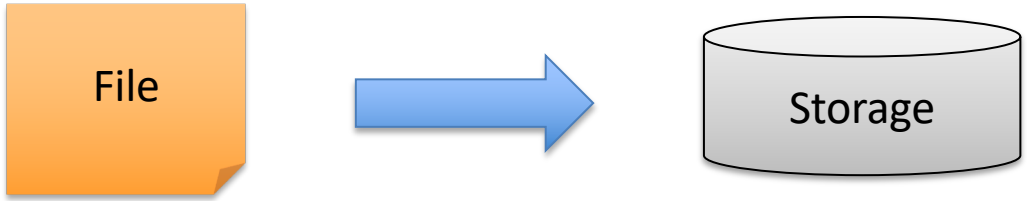
Outline

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

Why are Files an OS Problem?



- How to talk to the disk?
- Where to store your data in disk?
- How to operate different storage devices?
- How to coordinate with other processes/users?

Need to system operations for both supporting the file abstractions and perform necessary I/O operations

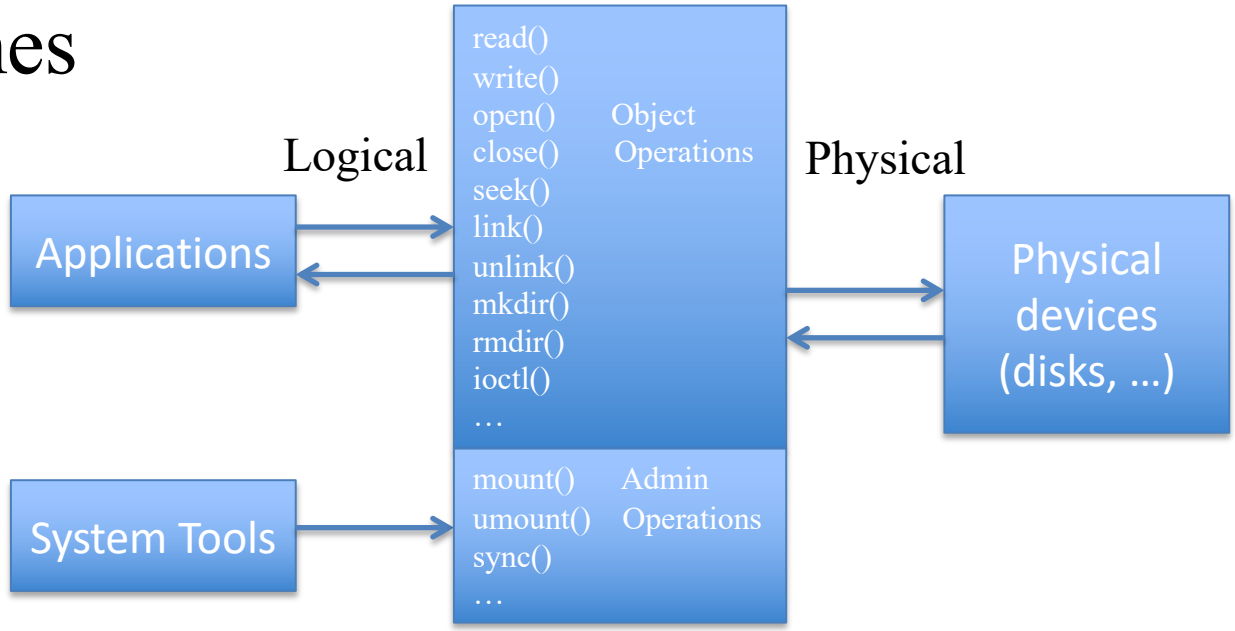
Interface versus Implementation

- Interface: high-level file system operations
- Implementation: engineering of a file system in OS
- Separation of concerns in the OS
  - Providing standardize functional interfaces and operational semantics to support programming portability
  - Allowing flexibility in developing mechanisms to realize different file system operations

File System Interface

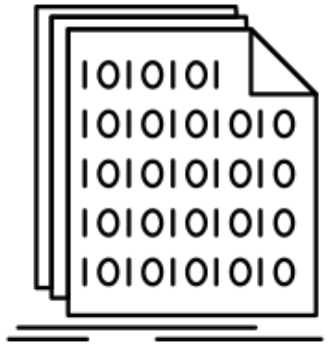
- Most visible part of the OS
- Consists of
  - Files
  - Directories
- And sometimes
  - Partitions

Think of the file system interface abstractly



What is a file?

- A repository for data
  - a sequence of bits, bytes, lines, or records…
  - data cannot be written to storage unless they are within a file
- Persistent (until explicitly deleted)
  - Stays around after program execution
- Meaning of file content defined by creator and user
  - OS perspective: concept of a file is extremely general



Note: file is the most basic way to share information.

Two Aspects To Consider

- User's view
  - Naming the file
  - What is the type and how is it structured/accessed?
  - What attributes does the file have?
  - What operations are support on the file?
  - What permissions are associated with the file?
- System implementation
  - Where is the file kept?
  - How are files from multiple users managed?
  - How is the storage system allocated?

Naming and Structure

- Naming
  - Typically use a tuple: *x.y*
  - *x* could give some clue about contents
  - *y* could relate to the nature of the file
- Structure
  - How is the file contents interpreted?
  - Byte stream
  - Sequence of records (more structured)
  - Indexed records (allows more positioning)

File Structure

- None (raw)
  - Sequence of bytes
- Simple record structure
  - Also called “lines”
  - Fixed length (know how to move to next record)
  - Variable length (include length in each record)
- Complex structures (examples)
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program (user)

File Types – Name, Extension

- These file types will have structure specific to their purpose
- Programs that use these files will need to know how to work with them (i.e., read/write)

Note: extension gives hint to user/OS what the file type might be

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

File Attributes

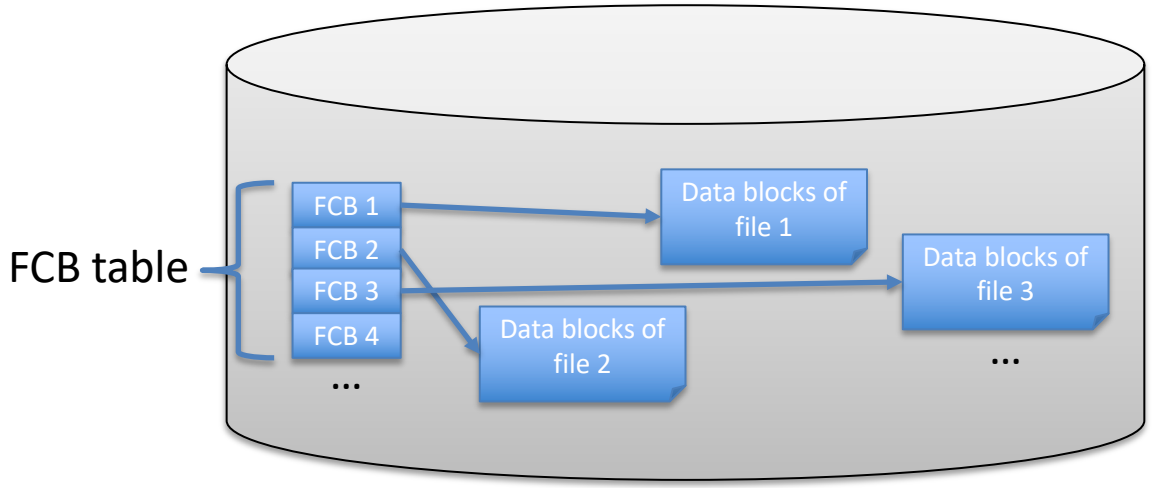
- *Name*
  - Only information kept in human-readable form
- *Identifier*
  - *U*nique tag (number) identifies file within file system
- *Type*
  - Needed for systems that support different types
- *Location*
  - *P*ointer to file location on device
- *Size*
  - Current file size
- *Protection*
  - Controls who can do reading, writing, executing
- *Time, date, user identification*
  - *D*ata for protection, security, and usage monitoring
- Information about files (metadata) are kept in the file system structure, which is maintained on the disk

Note: most of the attributes are stored in File Control Block (FCB).

File Control Block (FCB)

- "Real" file descriptor in storage
  - Contains all information about a file
  - Must read FCB first to know where the data blocks are located
  - File descriptor in program is a pointer to FCB (more detail comes soon)

Note: in Linux/Unix Control Block (FCB) is also called "inode".

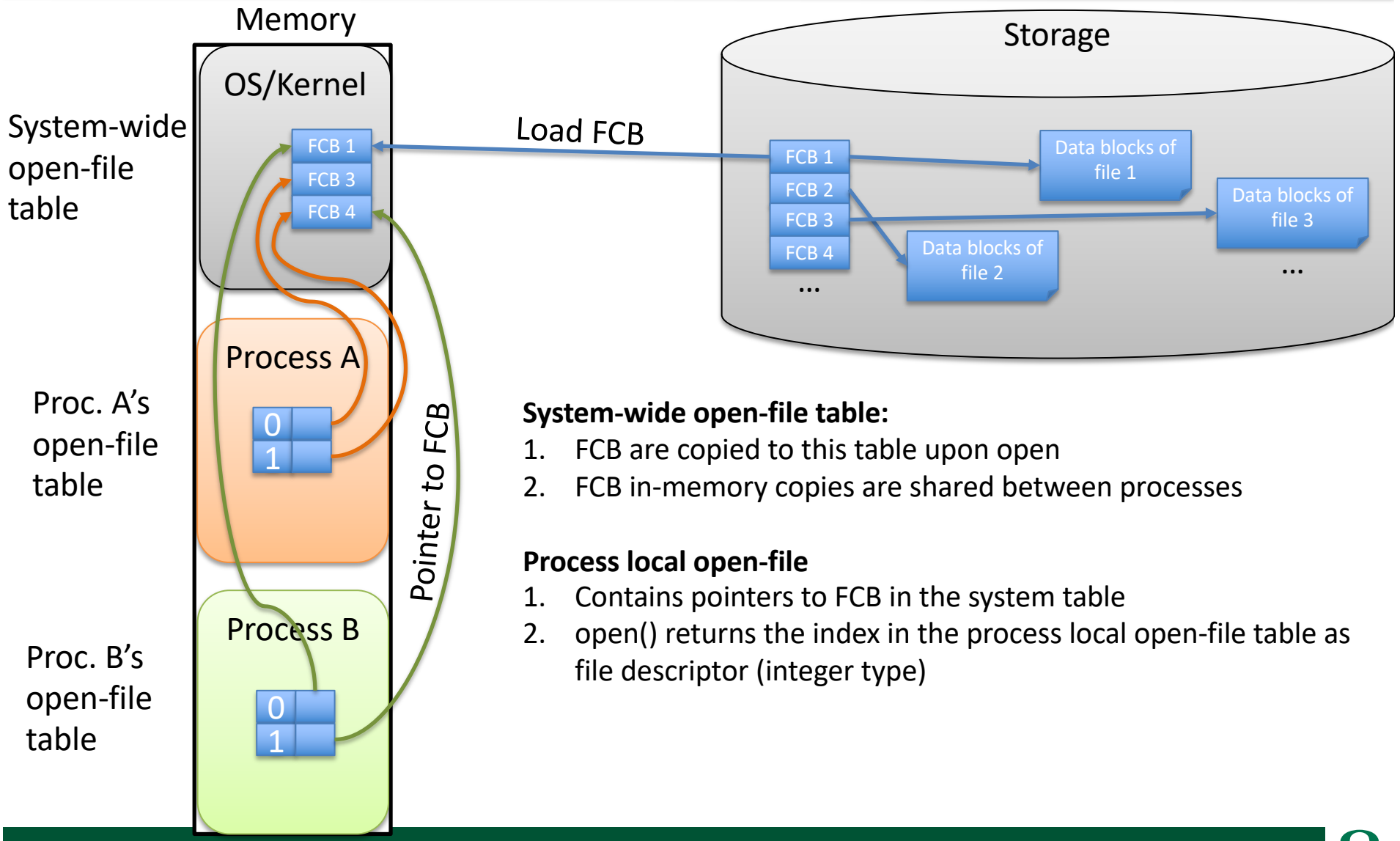


We will learn different ways to store these data blocks

File Operations

- File is an abstract data type
- *Create*
- *Write* – at write pointer location
- *Read* – at read pointer location
- *Seek* – reposition within file
- *Delete*
- *Truncate*
- *Open(F)* – search the directory structure on disk for entry F, and move its FCB to memory
- *Close(F)* – move the content of F's FCB in memory to directory structure on disk

Open Files



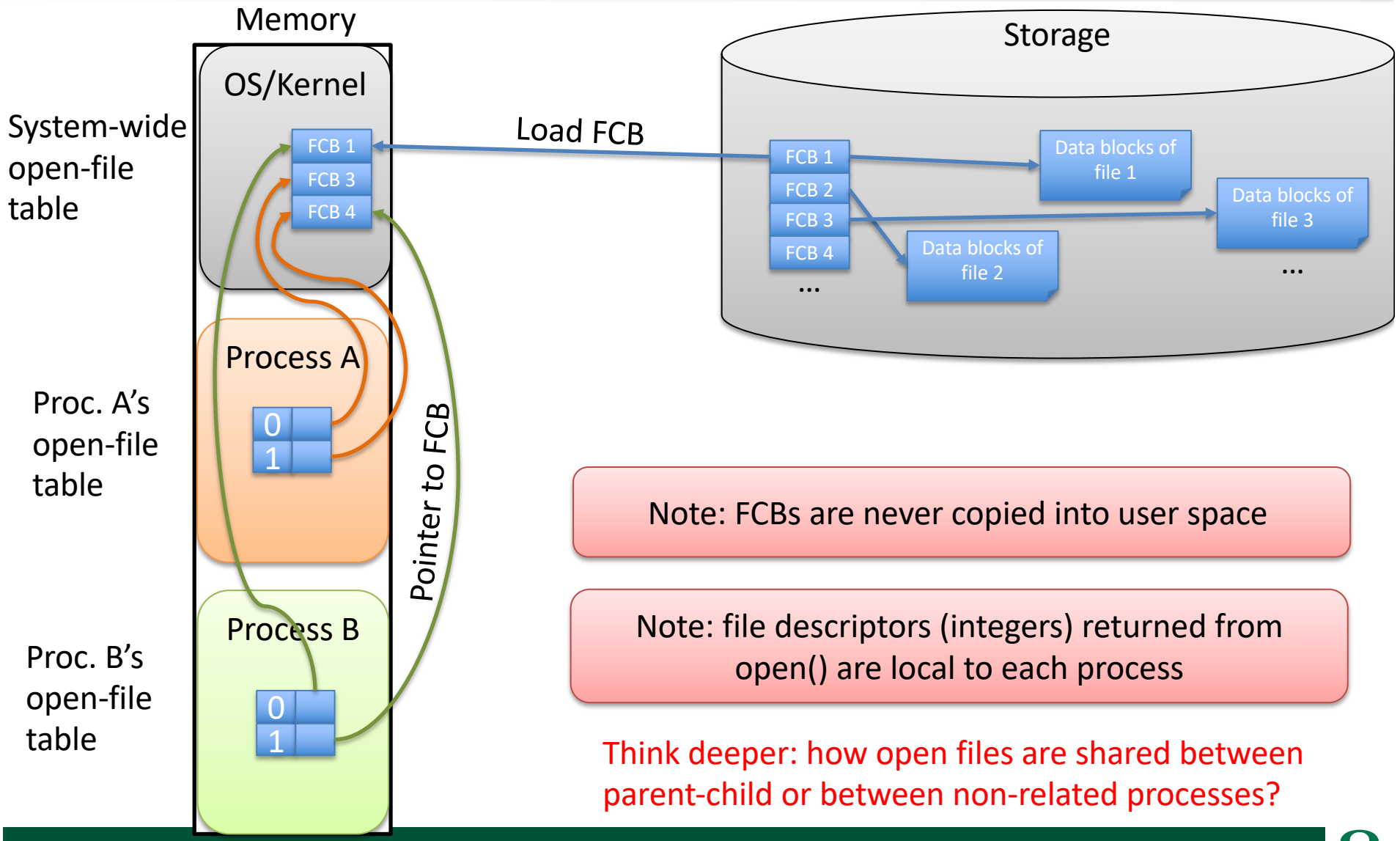
System-wide open-file table:

1. FCB are copied to this table upon open
2. FCB in-memory copies are shared between processes

Process local open-file

1. Contains pointers to FCB in the system table
2. open() returns the index in the process local open-file table as file descriptor (integer type)

Open Files

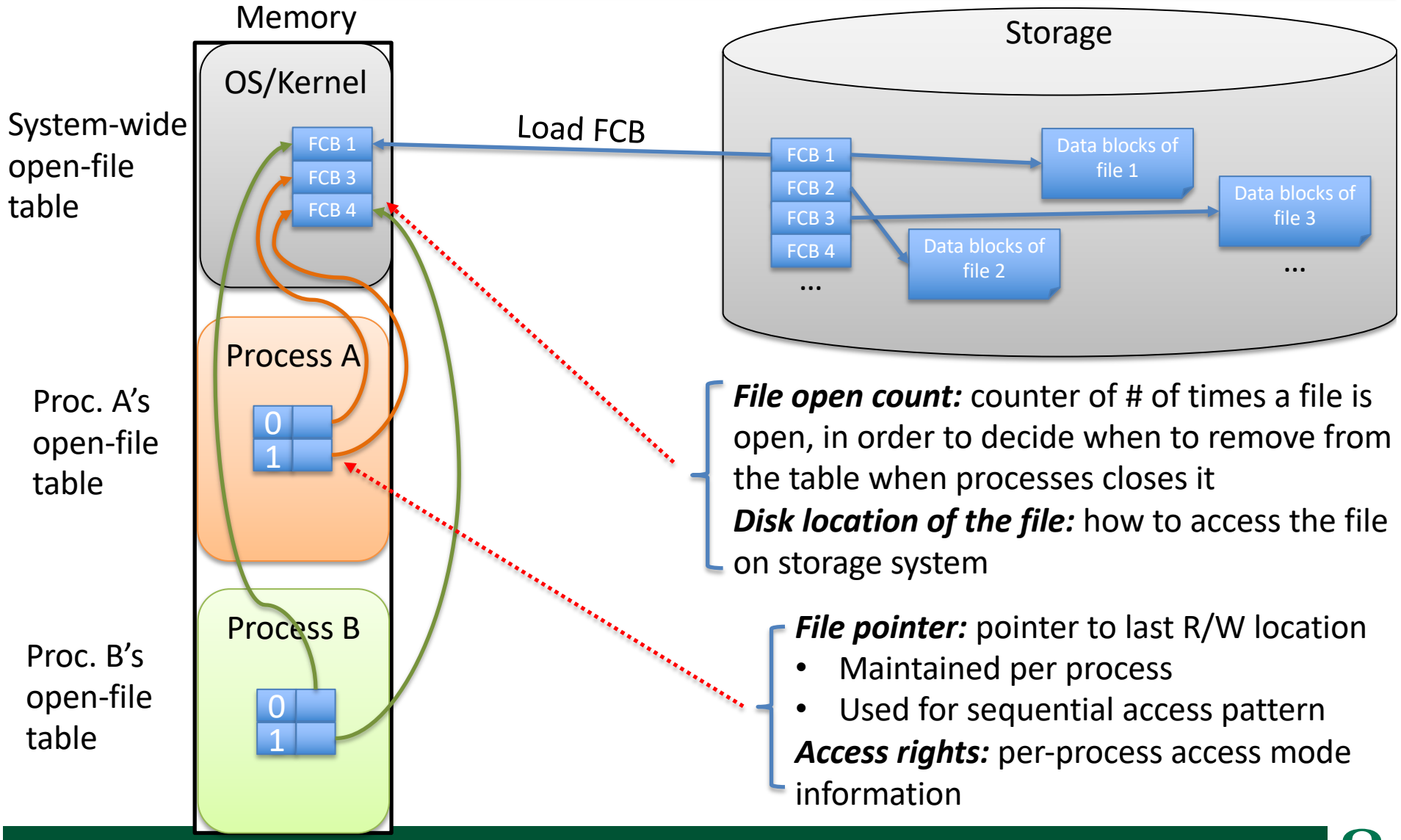


Note: FCBs are never copied into user space

Note: file descriptors (integers) returned from open() are local to each process

Think deeper: how open files are shared between parent-child or between non-related processes?

More information are stored in those tables



File open count: counter of # of times a file is open, in order to decide when to remove from the table when processes closes it

Disk location of the file: how to access the file on storage system

File pointer: pointer to last R/W location

- Maintained per process
- Used for sequential access pattern

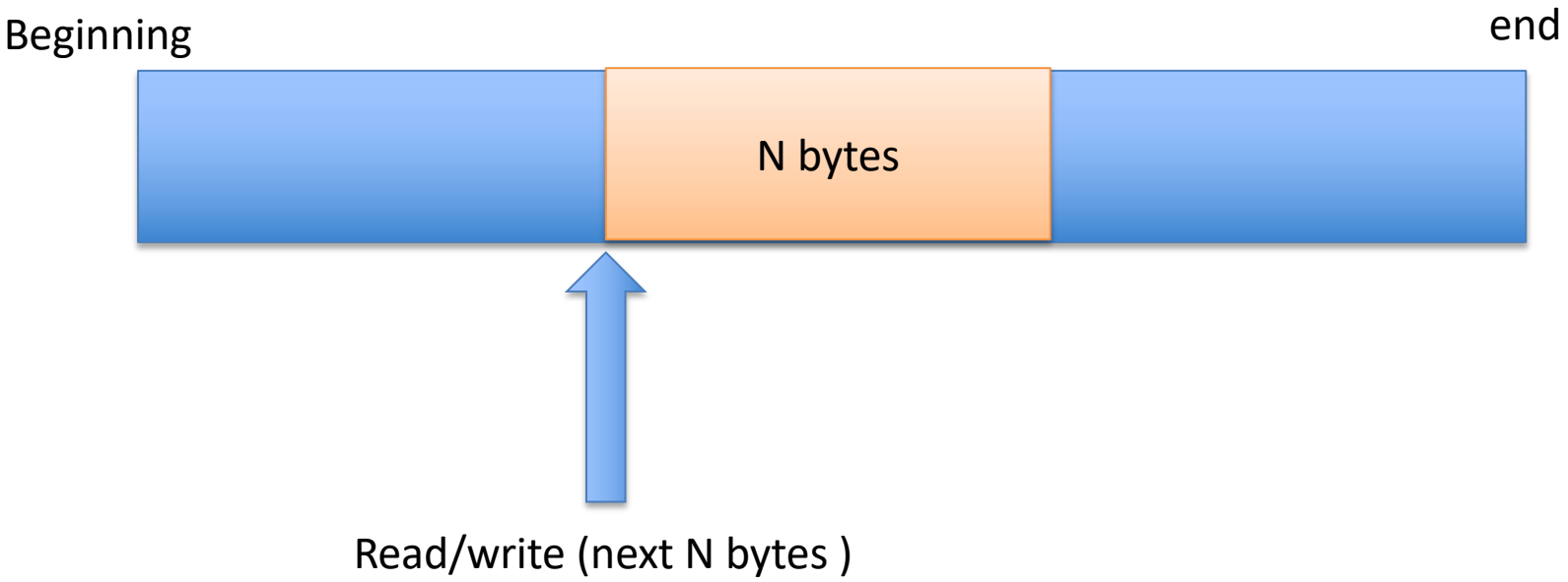
Access rights: per-process access mode information

Access Methods

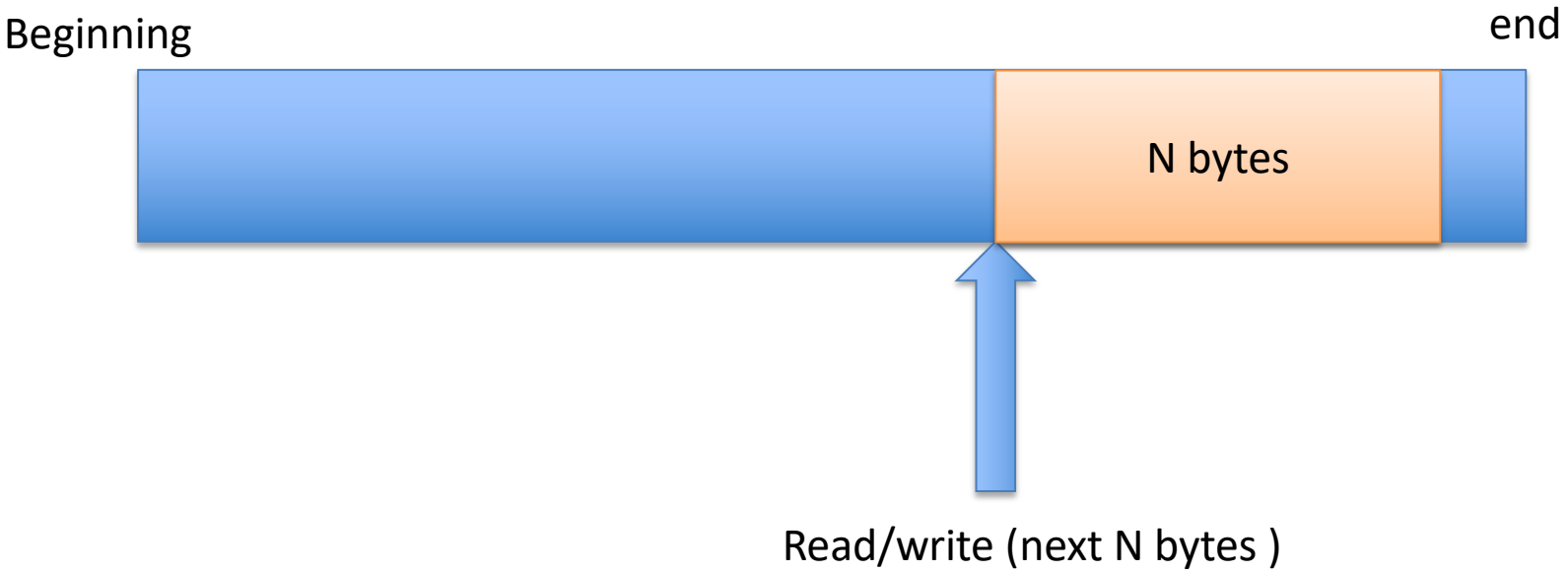
- Sequential Access
  - *reset* (set file pointer to beginning of file)
  - *read* next (advance file pointer to ”next” location)
  - *write* next (advance file pointer to ”next” location)
- Direct Access
  - *read n* (set file pointer to *n* location, then read)
  - *write n* (set file pointer to *n* location, the write)
  - *position* to *n*
    - *read* next
    - *write* next
- *n* = relative block number

Sequential File Access

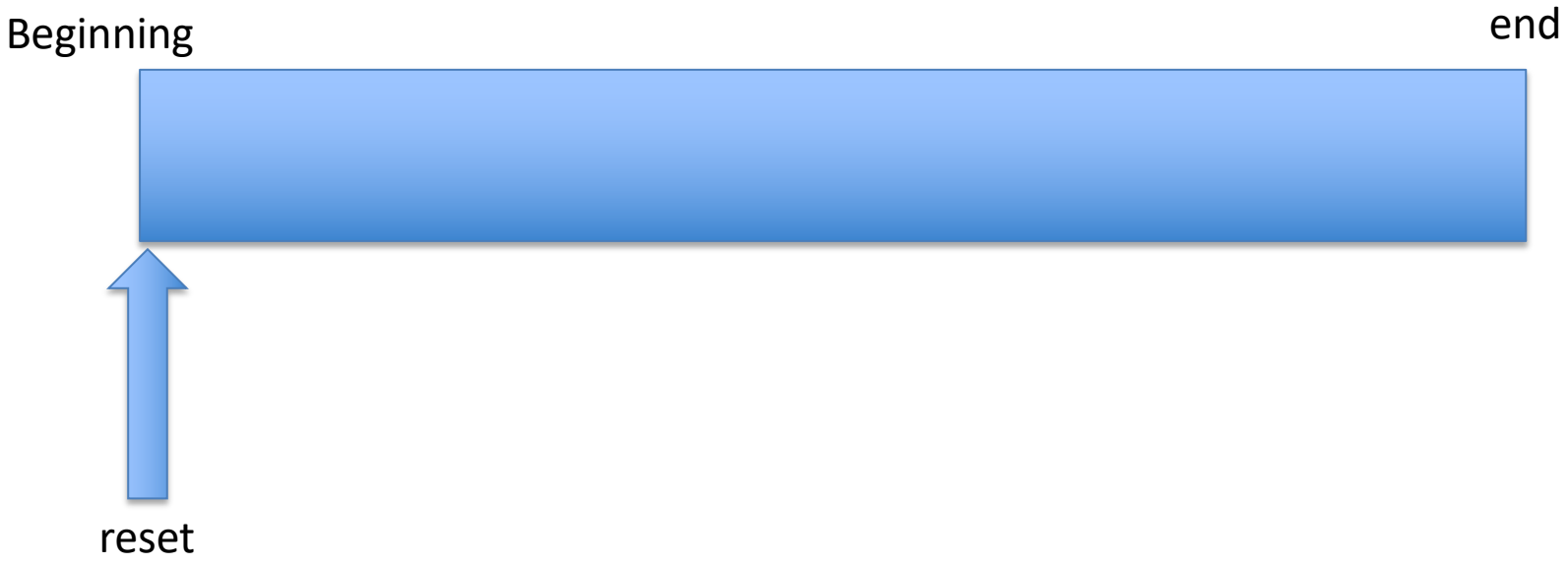
Sequential File Access



Sequential File Access

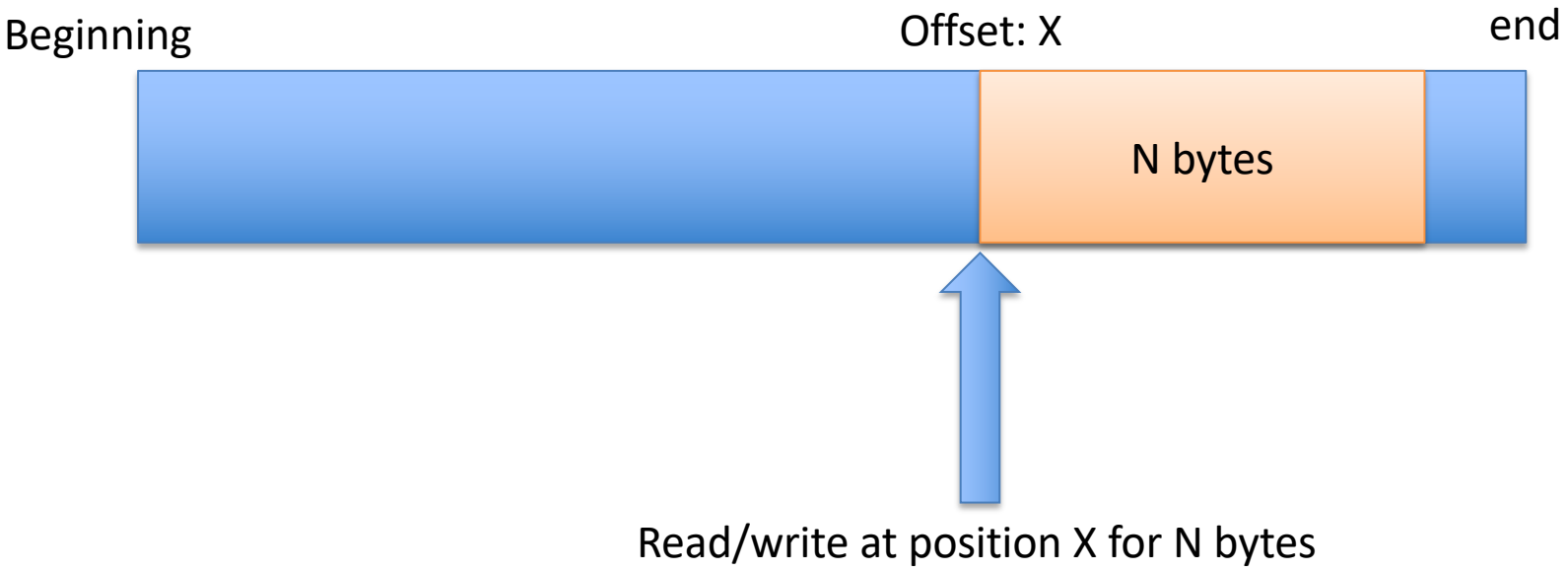


Sequential File Access



Code example:
posix-sequantial-file.c

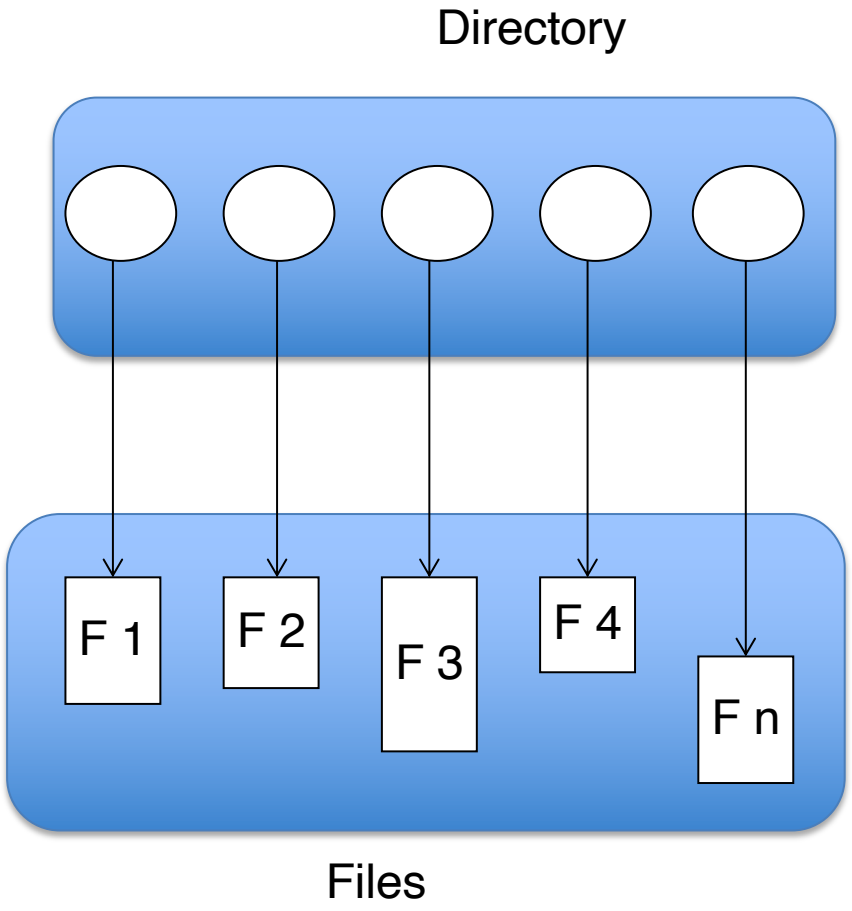
Direct File Access



Code example:
posix-direct-access-file.c

Directory

- A way of organizing files
- Each directory entry has:
  - File/directory name
  - A way to get to the data blocks of that file/directory (file pointer)
- Both the directory structure and the files reside on disk



Everything is file in Linux/Unix

- Regular files (containing data)
- Directories (containing information about files)

```
drwx------@   4 Jieyang  staff       128 Sep  4 15:03 Applications
drwxr-xr-x    6 Jieyang  staff       192 Sep 24 22:56 CS-415
drwx------@   6 Jieyang  staff       192 Nov 16 19:45 Desktop
drwx------+  13 Jieyang  staff       416 Sep 22 16:15 Documents
-rw-r--r--    1 Jieyang  staff        61 Jun  1  2025 double.c
-rw-r--r--    1 Jieyang  staff       536 Jun  1  2025 double.o
```

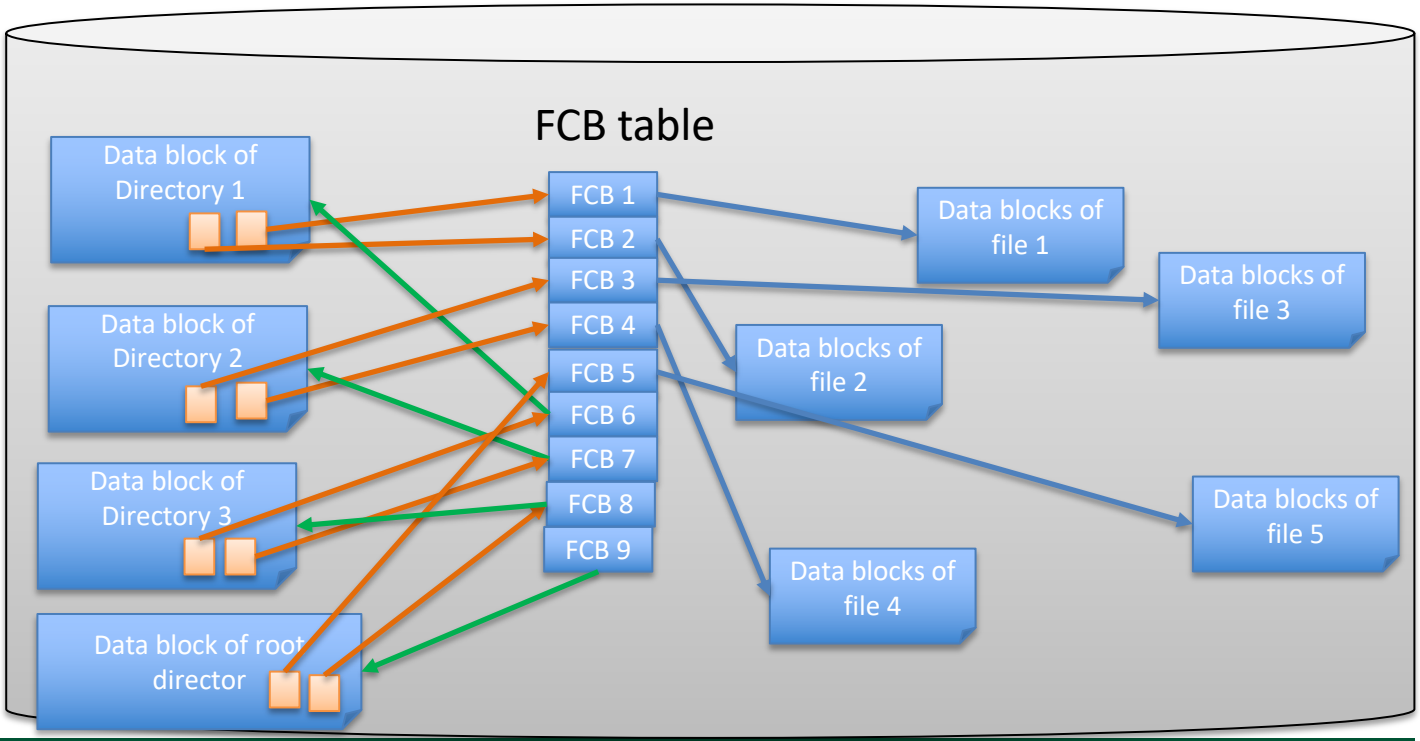
d = special file type: directory

Directories are files too!

Directories are files too!

- Root dir
  - Dir3
    - Dir1
      - File1
      - File2
    - Dir2
      - File3
      - File4
  - File5

Observation 1: directory data block stores pointer to FCB of file in it
Observation 2: directory itself also has FCB
Observation 3: root is also a directory (a special one)

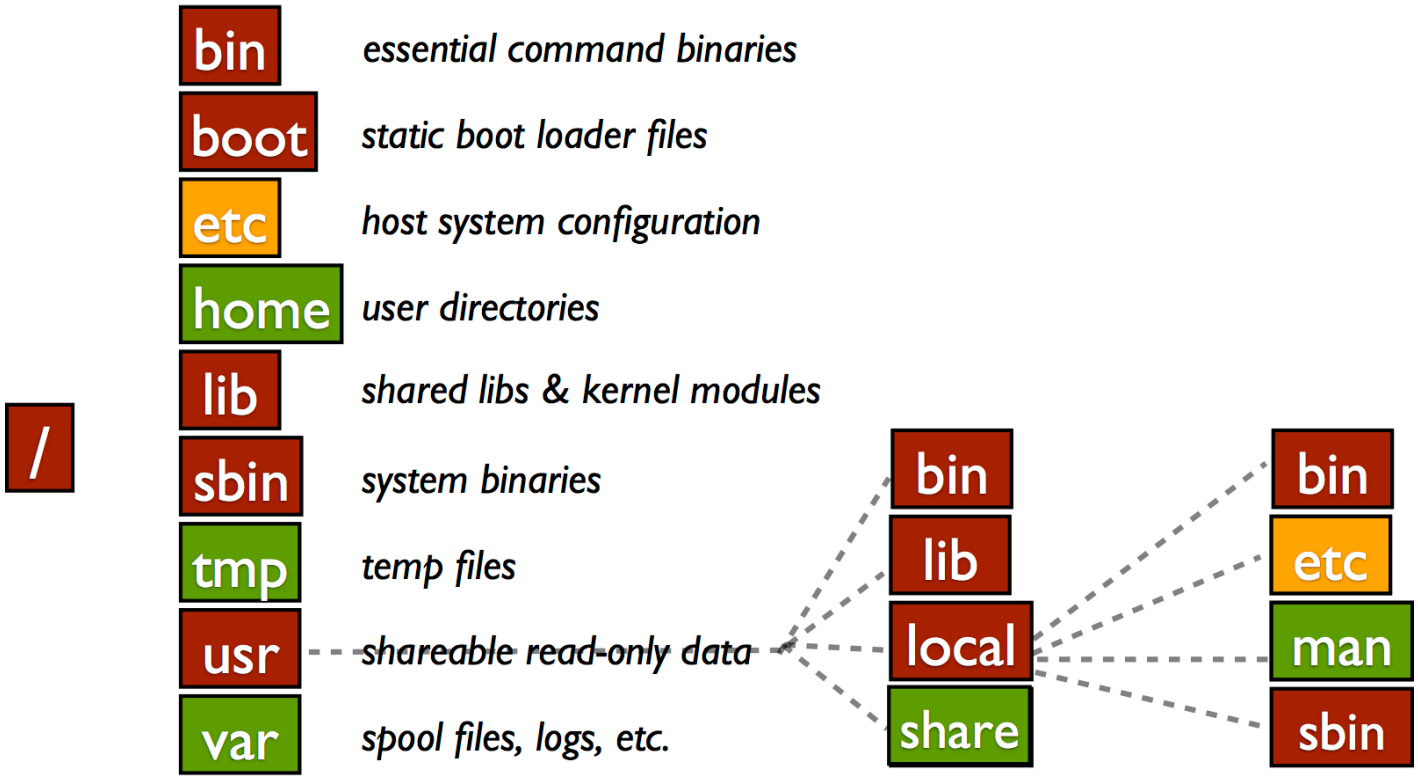


File System Details for Directories

- Organization
  - Flat (only 1 directory)
  - Hierarchical file system
- File names
  - Absolute (name is fully specified)
  - Relative (name is with respect to a "current" location
- Directory Operations:
  - *Create*
  - *Delete* <directory>
  - *Open* <directory>
  - *Close* <directory>
  - *Read* <directory>
  - *Rename* <directory>
  - *Link* (allows a file to appear in more than 1 directory)
  - *Unlink*

File System Hierarchy Standard

- UNIX *filesystem hierarchy standard* (FHS)
  - Convention to enable software and users to find the location of installed files and directories



Types of File Systems

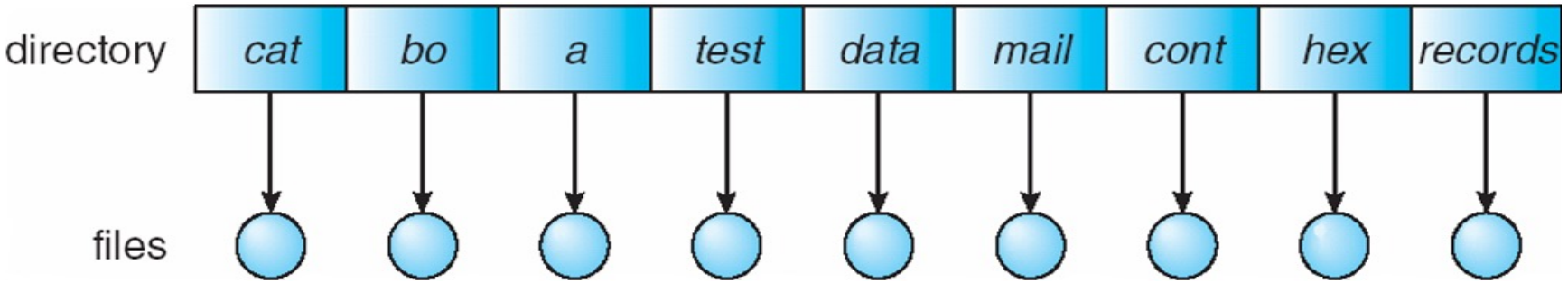
- We mostly talk of *general-purpose file systems*
  - Systems frequently have many file systems
    - some general purpose and some special purpose
- Consider Solaris has
  - *tmpfs* – memory-based volatile FS for fast, temporary I/O
  - *objfs* – interface into kernel memory to get kernel symbols for debugging
  - *ctfs* – contract file system for managing daemons
  - *lofs* – loopback file system allows one FS to be accessed in place of another
  - *procfs* – kernel interface to process structures
  - *ufs*, *zfs* – general purpose file systems

Directory Organization

- The directory is organized logically to obtain:
  - Efficiency – locating a file quickly
  - Convenient to users
    - two users can have same name for different files
    - the same file can have several different names
  - Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)
- Also concerned about efficiency of directory

Single-Level Directory

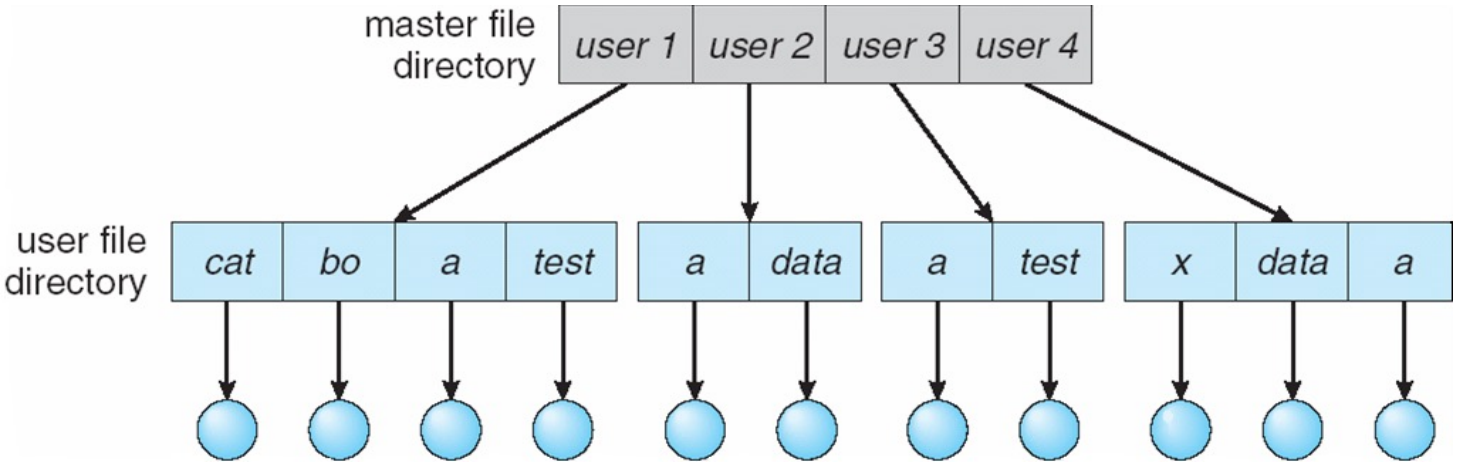
- A single directory for all users



- Naming problem
  - Each filename must be unique
- Grouping problem
  - Difficult to organize files into groups
  - Would have to incorporate into the name
- Locating a file requires searching the directory

Two-Level (Multi-Level) Directory

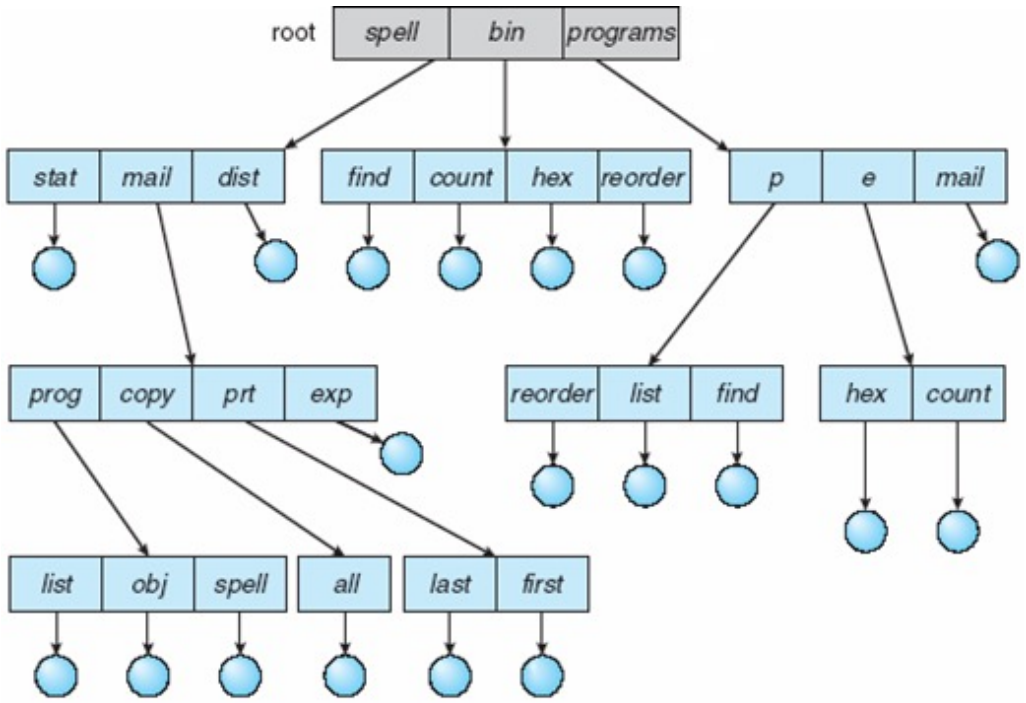
- Separate directory for each user



- Filenames are *path names*: directory + file
- User files can have the same name, but their pathnames will be different
- Allows for more efficient searching in the file system

Tree-Structured Directories

- Add additional levels to enable efficient searching
- Grouping capability
- Current directory (working directory)
  - cd /spell/mail/prog

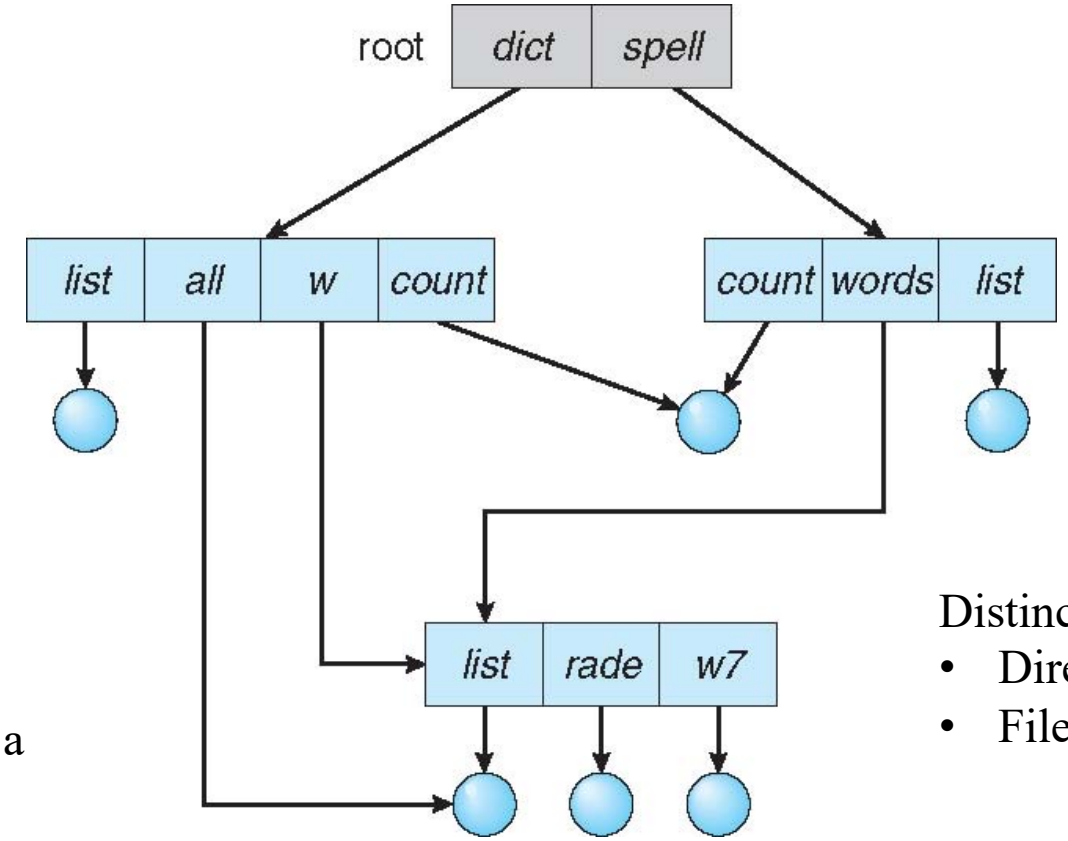


Path Names

- Absolute or relative path name
- Creating a new file is done in current directory
- Delete a file
  - *rm* <file-name>
- Creating a new subdirectory is done in current directory
  - *mkdir* <dir-name>

Acyclic-Graph Directories

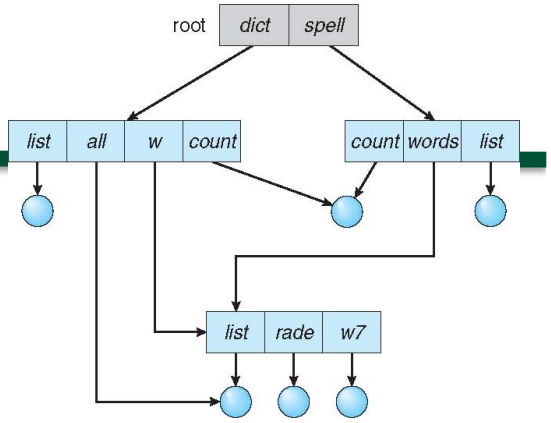
- Have shared subdirectories and files
- No cycles



Distinct pathnames
- Directory pathnames
- File pathnames

Different path to a
- Directory
- File

Links



- Two different names (*aliasing*)
- Link
  - Makes a file appear in more than one dirs
- When is it convenient to have this?
- 2 types of links:

Soft links

- create a file which contains the name of the other file
- use this path name to get to the actual file when it is accessed
- problem: extra overhead of parsing and following components till the file is reached.

Hard links

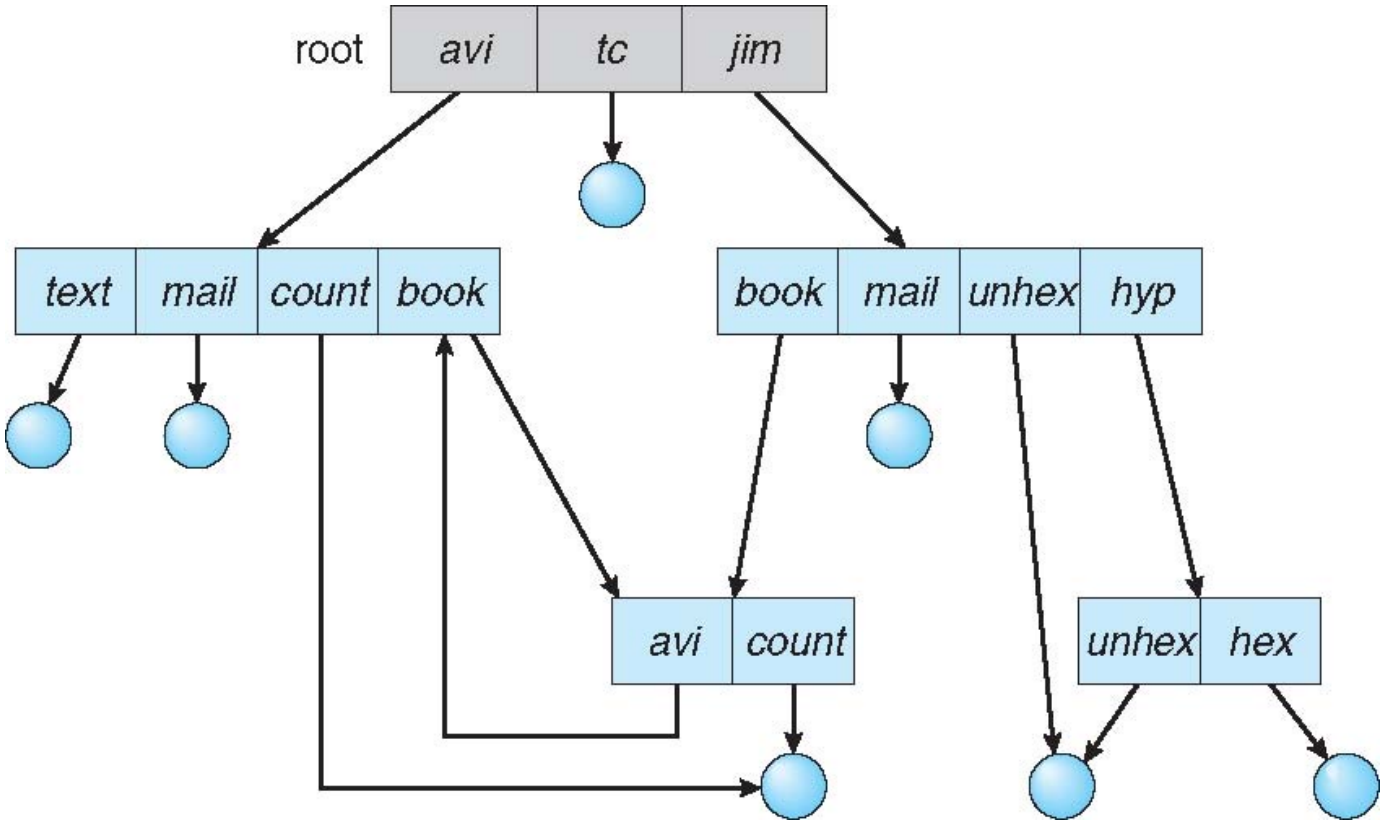
- create a directory entry and with a reference to a file
- others directory entries may reference the same file

Hard Link Problem

- What if the creator wants to delete the file?
  - There could be other references to the file
  - Can not free the file until all references are removed
- Solutions:
  - Store *backpointers* in file so we can delete all pointers

General Graph Directory

- Cycles allowed



General Graph Directory Problems

- Cycles can cause confusion about whether paths are valid and when to remove files
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection to track references
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

File Sharing

- In a multi-user system, there will be interest in sharing files
- What types of files are of interest to share
- System files
  - Shared by all
  - Examples: OS binaries, OS statistics, code development tools, websites, …
- Per user files
  - May want to work with others
  - Or with particular groups of users

How to Share

- Files and links
  - Short cut through the file system
    - hard and soft
- Directories
  - Must provide the other user or group access to your directory
- Remote file systems
  - Access files on another machine
  - Must provide the other user or group access to your machine and directory

Two permission control mechanisms
1. Basic Unix permissions
2. Access Control Policy

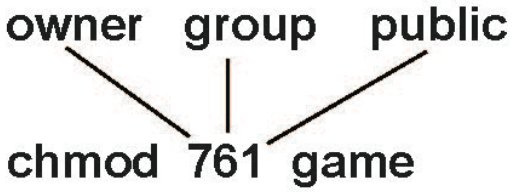
Basic Unix permissions

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

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

#	Permission	rwx
7	full	111
6	read and write	110
5	read and execute	101
4	read only	100
3	write and execute	011
2	write only	010
1	execute only	001
0	none	000

- 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
```

A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Access Control Policy

- An access control system determines what *rights* a particular *entity* has for a set of *objects*
- It answers the question
  - Do you have the right to read /etc/passwd?
  - Does Alice have the right to view the CS website?
  - Do students have the right to share project data?
- An *Access Control Policy* answers these questions

Simplified Access Control

- Subjects are the active entities that do things
  - You, Alice, students, Prof. Malony
- Objects are passive things that things are done to
  - /etc/passwd, CS website, project data, grades
- Rights are actions that are taken
  - read, view, share, change

More flexible than basic Unix permissions

The Access Matrix

- An access matrix is one way to represent policy.
  - Frequently used mechanism for describing policy
  - Columns are objects
  - Subjects are rows
  - Cells are rights
- To determine if S_i has right to access object O_j, find the appropriate entry.
- Succinct descriptor for O(|S|*|O|) entries
- There is a matrix for each right

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

Secrecy

- Does the following protection state ensure the secrecy of J's private key in O_1?
- Does the following access matrix protect the integrity of J's public key file O_2?

	Private key	Public key
J	R	RW
K	N	R
L	N	R

File Protection in Unix

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

Summary

- File System Interface
  - Files
  - Directories
  - Partitions
- Operations on the interface
  - Mounting (partitions)
  - Sharing (files)
  - Protection (files)

Next Class

- File system implementation