Operating Systems
CS 415

Lecture 13:

File System Interface

Suyash Gupta

Assistant Professor

Distopia Labs and ONRG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Announcements

- **Suyash Gupta**
  - Office Hours: Thursday, 10-11am, Deschutes 334

- **Nihal Balivada (TA)**
  - Office Hours: Wednesday/ Friday, 11-12pm, Deschutes 335

- **Ranjitha Rani (TA)**
  - Office Hours: Monday /Tuesday, 1-2pm, Deschutes 229

Assignment 3 is out!

- **Deadline → June 3, 2026 at 11:59pm PST**
- Please start working and talk to us if you are stuck.

- **Final → June 10, 2026 at 12:30pm PST, STB 145**
  - Closed book, no cheat sheets, no discussions.

Last Class

- Virtual Memory (Chapter 10)
- Next, we move to Chapter 13!

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.

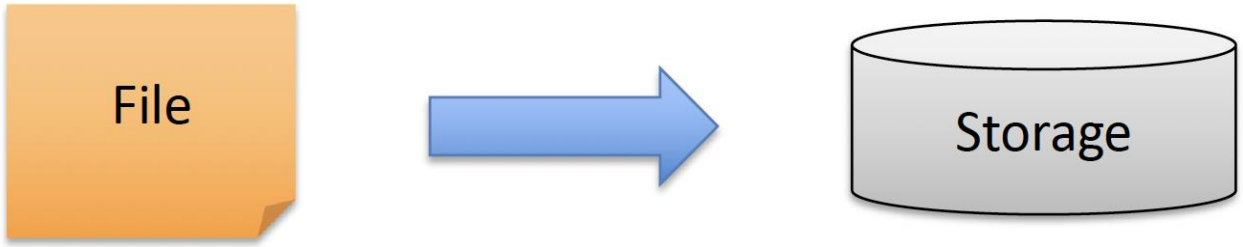
- Persistent (until explicitly deleted)
  - Stays around after program execution.

- Meaning of file content defined by creator and user.

Why are Files an OS Problem?

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?



Two Perspectives

- 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?

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

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 Types

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** → Unique tag (number) identifies file within file system.
- **Type** → Needed for systems that support different types.
- **Location** → Pointer to file location on device.
- **Size** → Current file size.
- **Protection** → Controls who can do reading, writing, executing.
- **Time, date, user identification** → For protection, security, and usage monitoring.

- Information about all files is kept in the directory structure, which resides on the same device as the files.
  - Each directory entry consists of the file's name and its unique identifier.

File Operations

- A file is an abstract data type.

- To define a file properly, we need operations that can be performed on files.

File Operations

- **Creating a file.**
  - First, find space in the file system for the file.
  - Second, make an entry for the new file in a directory.

- **Opening a file.**
  - Rather than have all file operations specify a file name, causing the OS to evaluate the name and check access permissions, open a file first.
  - If successful, returns a file handle → used as an argument in the other calls.

File Operations

- **Writing a file.**
  - Perform a system call specifying both the open file handle and the information to be written to the file.
  - The system must keep a write pointer to the location in the file where the next write is to take place.
  - The write pointer must be updated whenever a write occurs.

- **Reading a file.**
  - Perform a system call that specifies the file handle and where (in memory) the next block of the file should be put.
  - The system needs to keep a read pointer to the location in the file where the next read is to take place.
  - Once the read has taken place, the read pointer is updated.

File Operations

- **Repositioning within a file.**
  - The current-file-position pointer of the open file is repositioned to a given value.
  - Repositioning within a file need not involve any actual I/O.

- **Deleting a file.**
  - Search the directory for the named file.
  - Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase or mark as free the directory entry.

- **Truncating a file.**
  - The user may want to erase the contents of a file but keep its attributes.
  - This function allows all attributes to remain unchanged, except for file length.

File Control Block (FCB)

File Control Block (FCB)

- Also called an inode in Unix/Linux.
- Contains all information about a file.
- Stores metadata such as file size, ownership, permissions, timestamps, and crucially, pointers to the data blocks on disk.
- OS reads the FCB to locate the actual data blocks before any read/write.
- FCB is the persistent, on-disk structure that definitively describes a file.
  - When a file is opened, the OS reads the FCB from disk and keeps an in-memory copy in the open file table for fast access.

Opening a File

Opening a File

- **Step 1: File Descriptor**
  - When your program calls ***open("foo.txt"),*** the OS returns a small integer, example 4.
  - That number is all your program ever sees or uses.
  - It is just an index number.

Opening a File

- **Step 2: Per-Process File Descriptor Table**
  - Every process has its own private table, essentially an array.
  - The integer 4 is used as the array index.
  - Slot 4 in your process's table holds a pointer to an entry in the next table.
  - This is why two different processes can both have a file descriptor 4 and refer to completely different files.

Opening a File

- **Step 3: System-Wide Open File Table**
  - One table shared by the whole OS.
  - Each entry is created when someone calls open() and it stores:
    - The current read/write offset (where in the file you are)
    - The access mode (read-only, write-only, etc.)
    - A reference count (how many descriptors point here)
    - A pointer to the inode/FCB

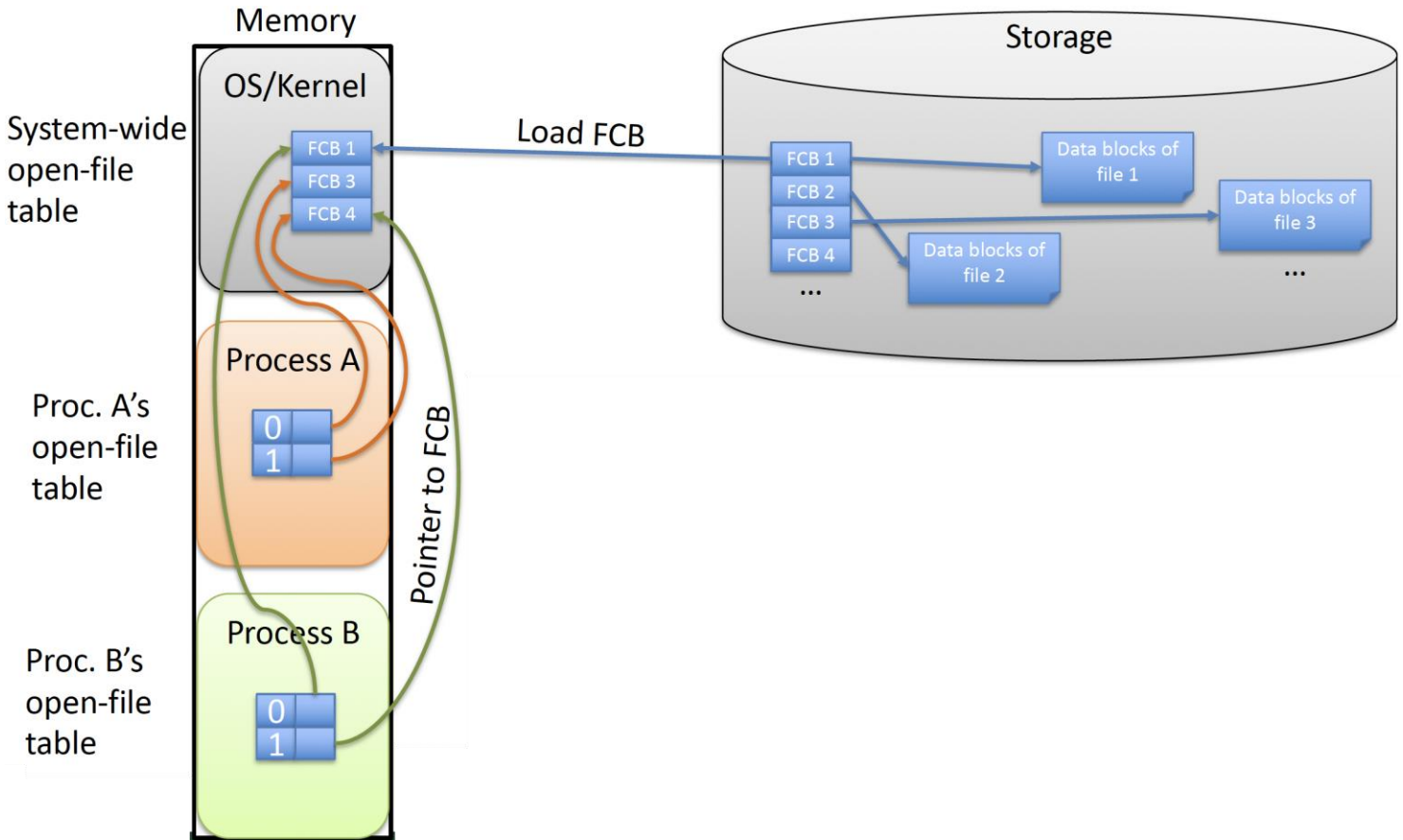
- This is why two processes can read the same file independently, each open() call gets its own entry with its own offset, but they share the same underlying inode.

Opening a File

- **Step 4: In-Memory Inode/FCB Copy**
  - The inode lives permanently on disk, but when a file is opened, the OS loads a copy into RAM for fast access.
  - This in-memory copy holds the file's metadata: size, permissions, owner, and pointers to where the actual data blocks are on disk.

Opening a File

- **Step 5: Disk Data Blocks**
  - The inode's block pointers are followed to find and read the actual file content on the physical disk.



Internal File Structure

- Disk systems have a well-defined block size determined by the size of a sector.
- All disk I/O is performed in units of one block and all blocks are the same size.
- May need to packing a number of logical records into physical blocks.
  - The logical record size, physical block size, and packing technique determine how many logical records are in each physical block.

Access Methods

- Files store information.

- We need mechanisms to access this information and read into computer memory.

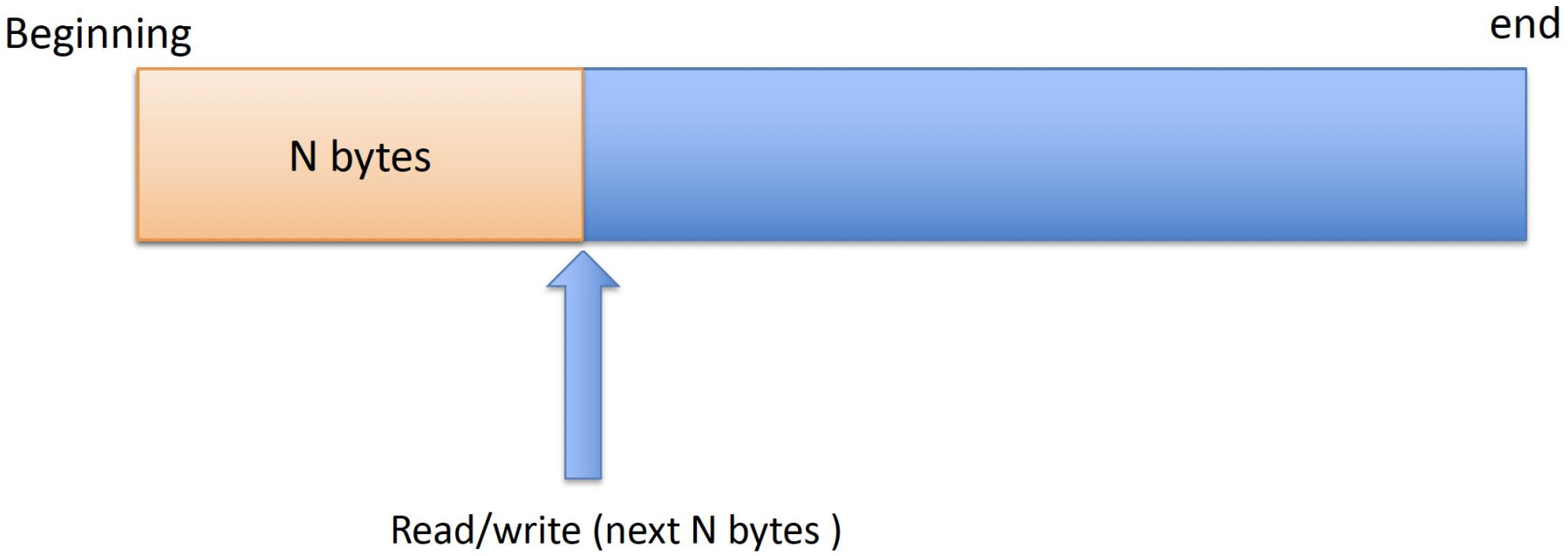
Access Methods

- Files store information.
- We need mechanisms to access this information and read into computer memory.
- Two key methods:
  - Sequential Access
  - Direct Access

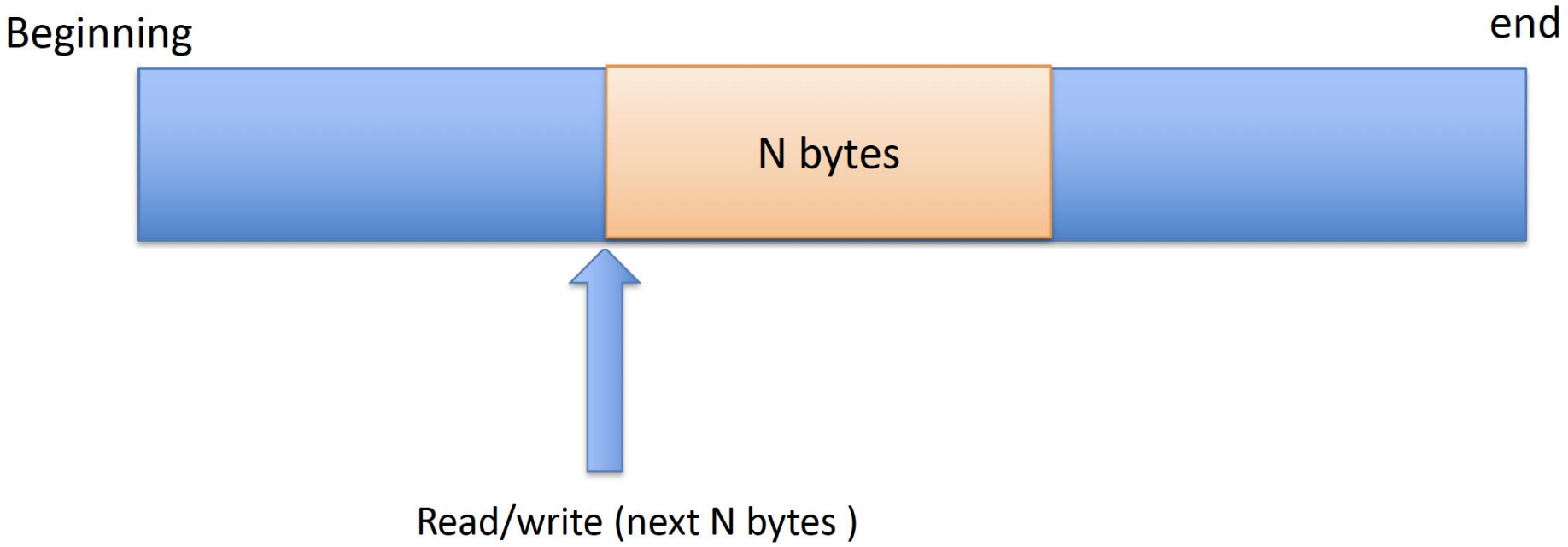
Sequential Access

- Three key methods:
  - reset → set file pointer to beginning of file.
  - read next → advance file pointer to “next” location.
  - write next → advance file pointer to “next” location.

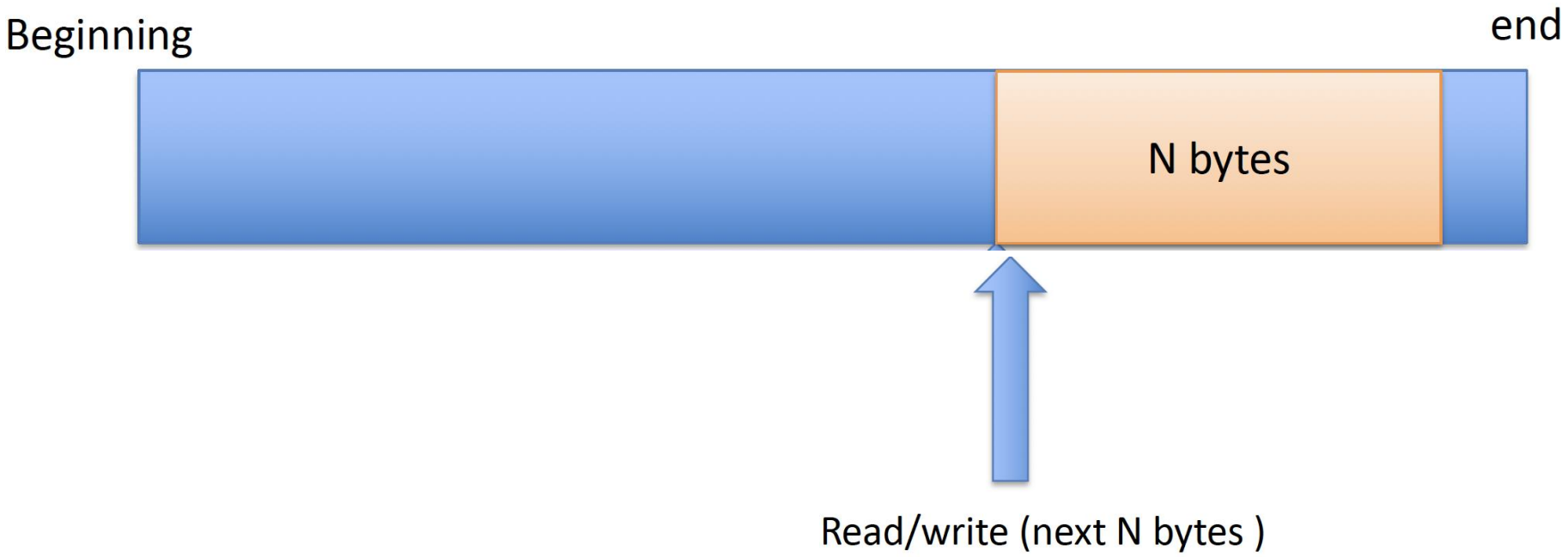
Sequential Access



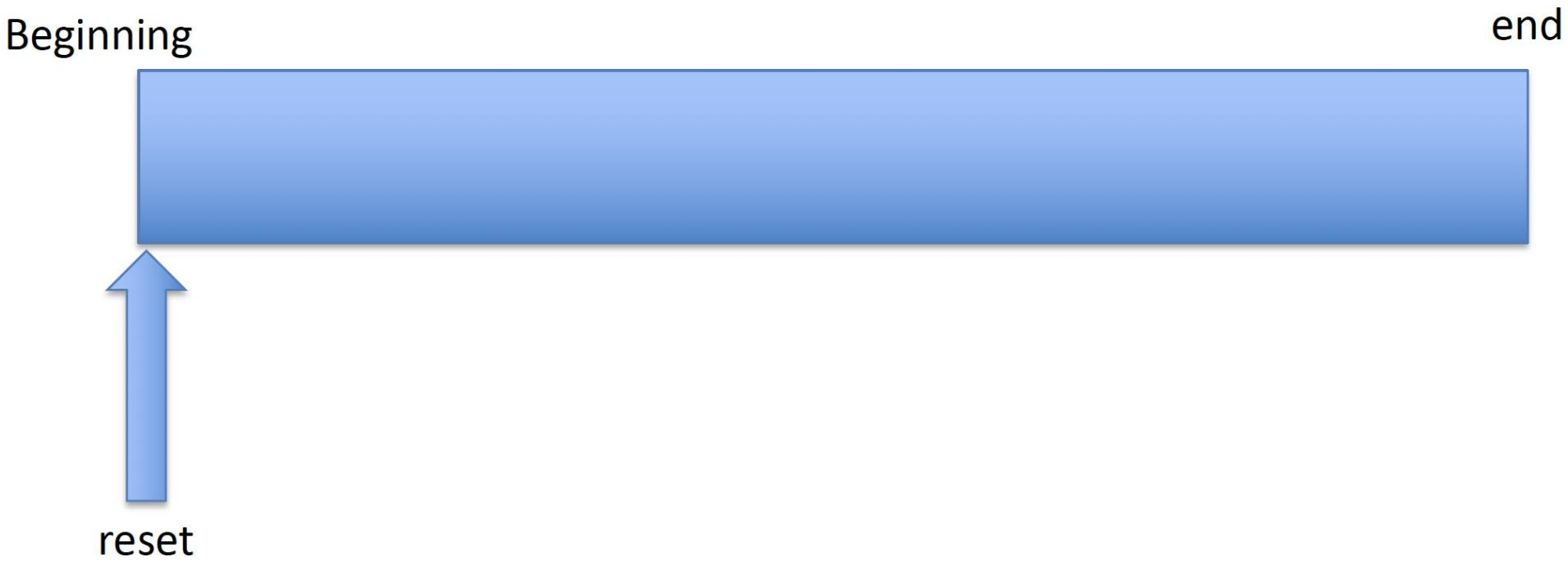
Sequential Access



Sequential Access



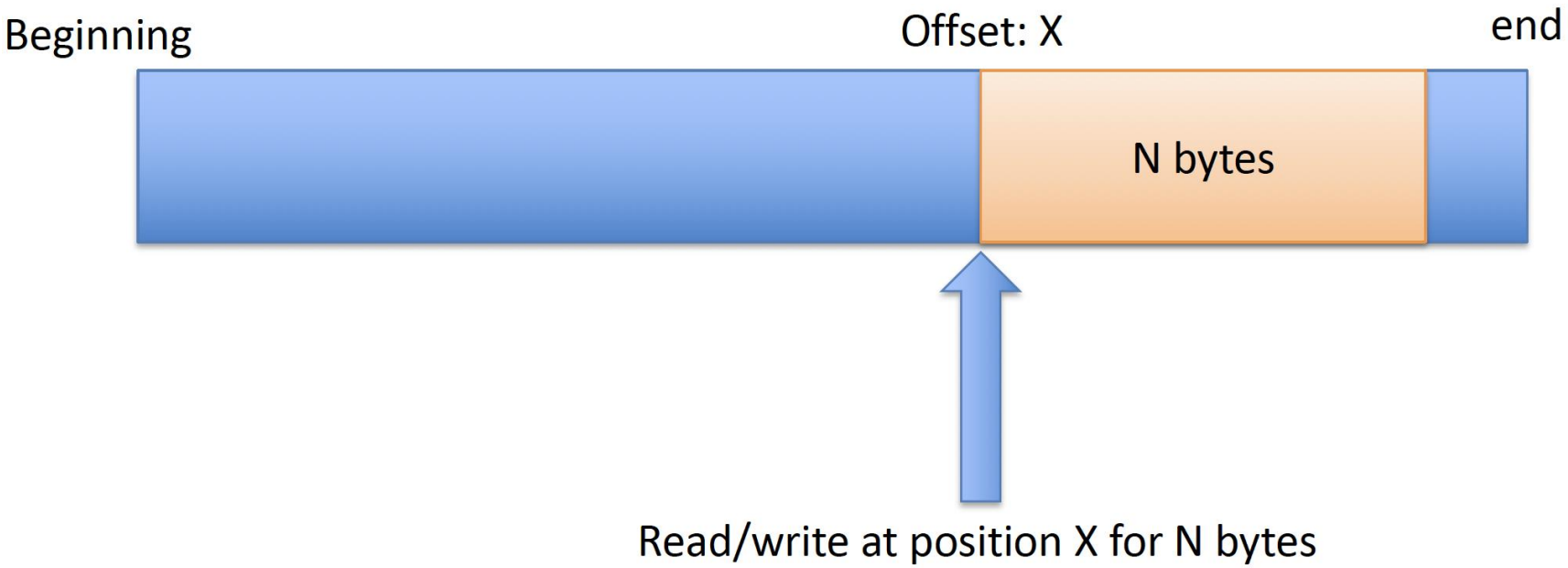
Sequential Access



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

Direct Access



Directories

Directories

- Directory is a symbol table that translates file names into their file control blocks.

- Directory itself can be organized in many ways.
  - Any organization scheme must allow us to insert, delete and search entries, and to list all the entries in the directory.

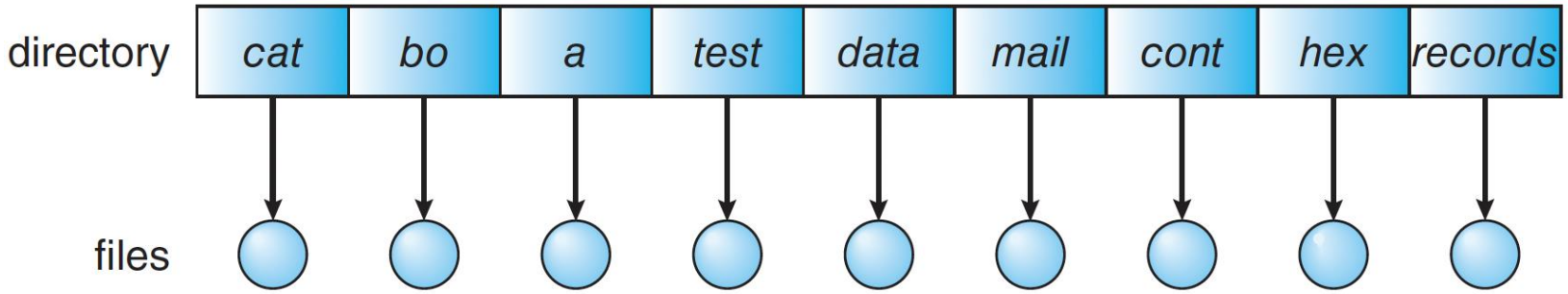
Operations on Directories

- Search for a file.
- Create a file.
- Delete a file → Leaves a hole in the directory structure and the file system may have a method to defragment the directory structure.
- List a directory.
- Rename a file.
- Traverse the file system →to access every directory and every file within a directory structure.

Single-Level Directory

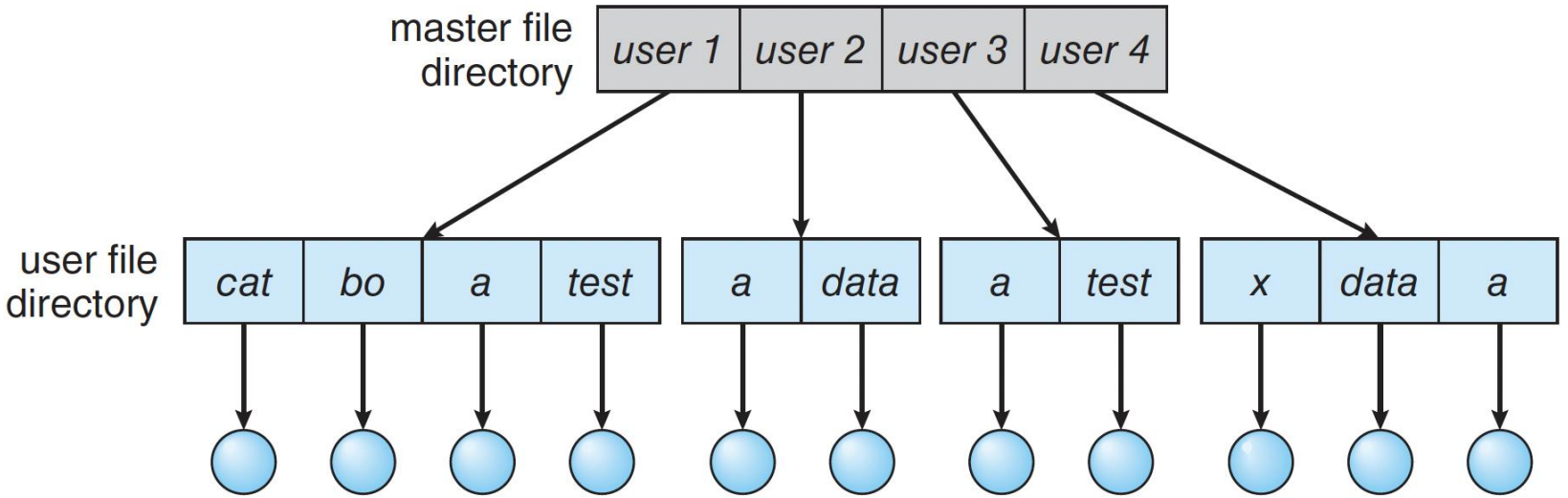
- All files are in the same directory.

- When the number of files increases or when the system has more than one user, all files must have unique names.



Two-Level Directory

- When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched.
  - The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- Each user has his own **user file directory (UFD)**.
  - The UFDs have similar structures, but each lists only the files of a single user.

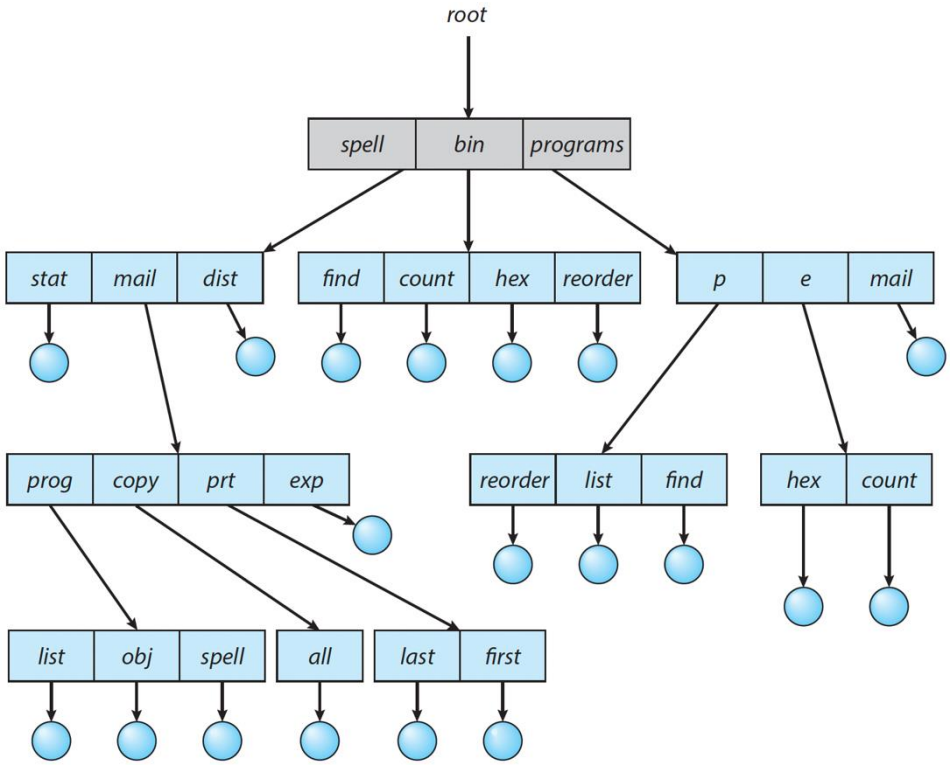


Two-Level Directory

- Effectively isolates one user from another.
- Not efficient when users want to cooperate on some task and to access one another's files.
  - Some systems do not allow local user files to be accessed by others.

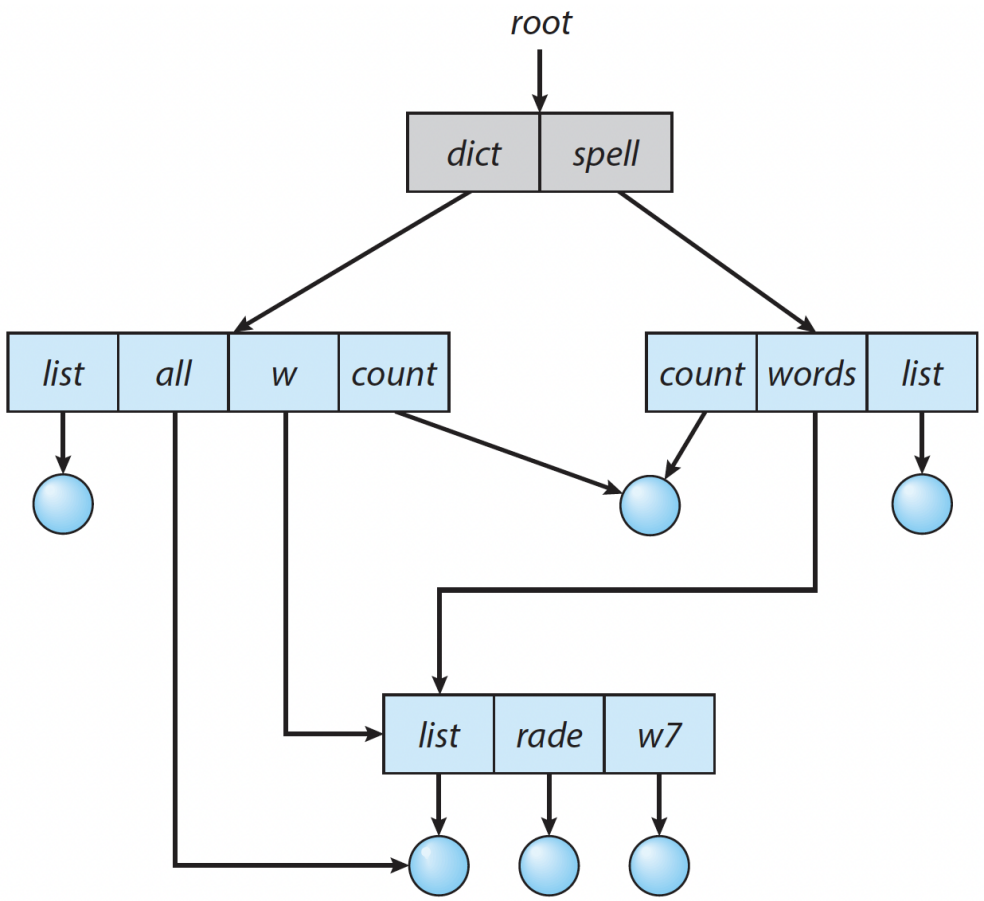
Tree Structured Directory

- Add additional levels to enable efficient searching.
  - One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Each process has a Current directory (working directory).



Acyclic-Graph Directories

- An acyclic graph → no cycles.
- Allows directories to share subdirectories and files.
  - The same file or subdirectory may be in two different directories.
  - Not the same as two copies of the file.
  - changes made by one person are immediately visible to the other.



Implementing Acyclic-Graph Directories

- Links!

- Two different names (aliasing)
  - Makes a file appear in more than one directories.

Links

- **Hard Links**
  - Just another directory entry pointing directly to the same inode.
  - Two filenames that are completely equal and resolve to the same inode number.
  - The inode has a link count that tracks how many directory entries point to it.
  - When you delete one hard link, the OS just decrements the count to 1.
  - Only when the count hits 0 does the OS free the inode and data blocks.

- **Soft Links**
  - A soft link (symbolic link) is a brand new inode whose data is just a path string.
  - It has its own inode number and is a separate file.
  - Every time you access the link, the OS does a two-step lookup: read the path string from the symlink's inode, then re-resolve that path from scratch.
  - If the original file is deleted, the link becomes a dangling link.

Links

- **Hard Links**

  ln original.txt hardlink.txt

- **Soft Links**

  ln -s original.txt softlink.txt

Path Names?

Path Names?

- User name + a file name.
  - Every file in the system has a path name.

- **Absolute Path name**:
  - In UNIX and Linux, an absolute path name begins at the root (which is designated by an initial "/") and follows a path down to the specified file, giving the directory names on the path.

- **Relative path name**:
  - If the current directory is ***spell/mail,*** then the relative path name ***prt/first*** refers to the same file as the absolute path name ***/spell/mail/prt/first***.

Access Control

Access Control

- Define who can access the file and what mode.
- 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

Sample Unix Directory Structure

-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/