NATIONAL ECONOMICS UNIVERSITY

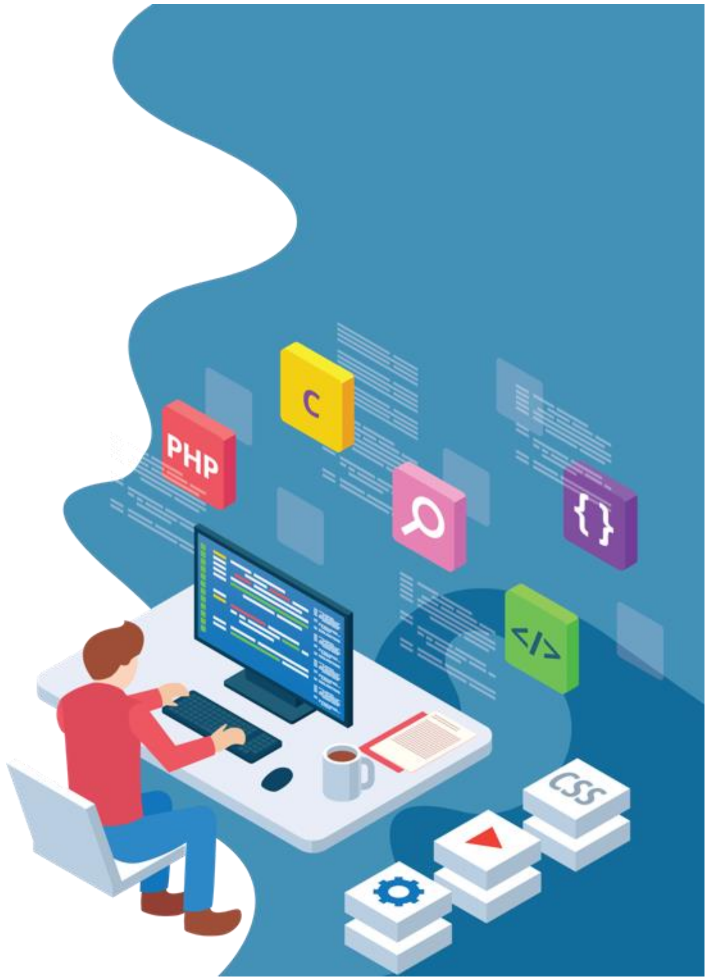
SCHOOL OF INFORMATION TECHNOLOGY AND DIGITAL ECONOMICS

CHAPTER 5

FILE-SYSTEM

OUTLINE

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

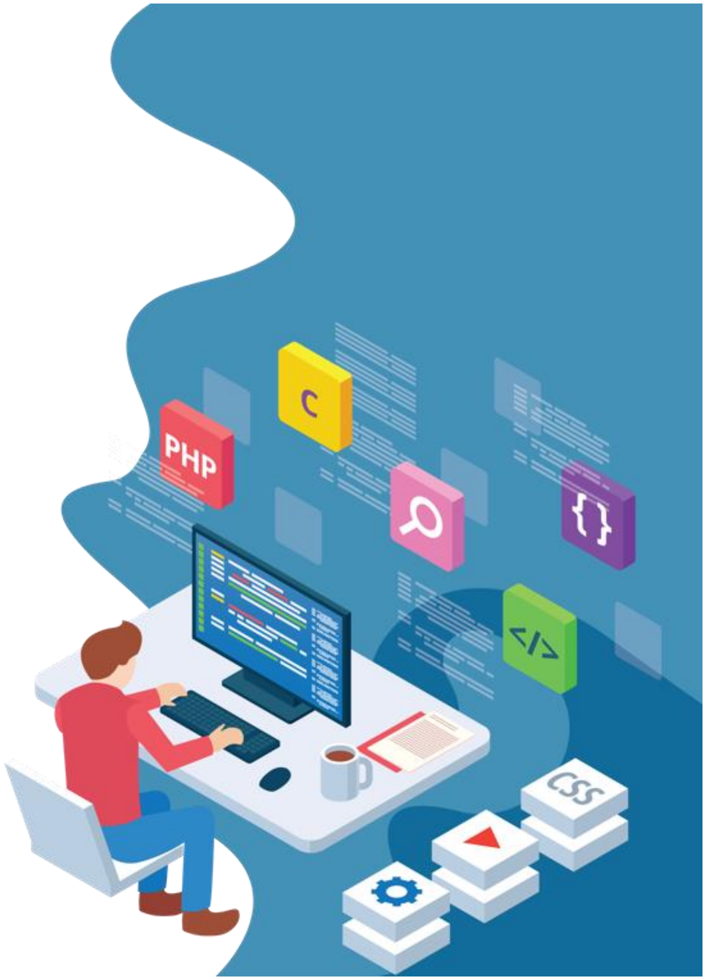


OBJECTIVES

- 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
- Describe the details of implementing local file systems and directory structures
- Discuss block allocation and free-block algorithms and trade-offs

OUTLINE

- File Interface
  - ➡ File Concept
  - Access Methods
  - Disk and Directory Structure
  - File Sharing
  - Protection
- File System Implementation



FILE CONCEPT

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**

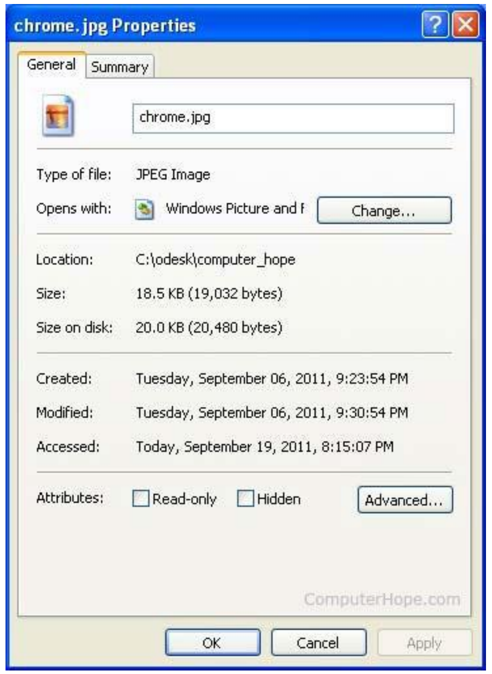
FILE STRUCTURE

- File types also can be used to indicate the internal structure of the file.
- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program

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, and user identification** – data for protection, security, and usage monitoring
- **Information** about files are kept in the directory structure, which is maintained on the disk
- **Many variations**, including extended file attributes such as file checksum

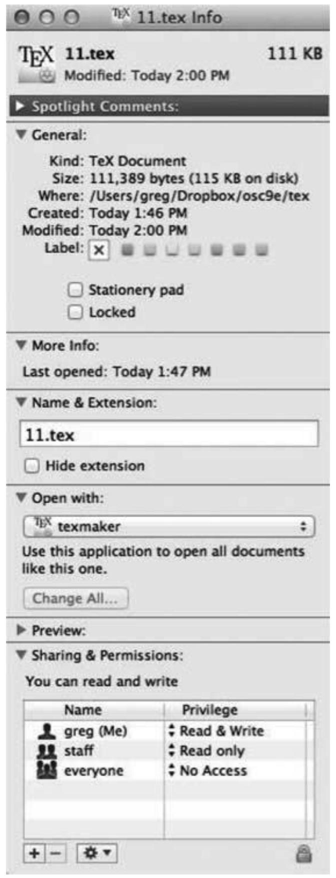
FILE INFO ON WINDOWS AND MAC OS X


chrome.jpg Properties

General | Summary

chrome.jpg

Type of file:	JPEG Image
Opens with:	Windows Picture and F
Location:	C:\odesk\computer_hope
Size:	18.5 KB (19,032 bytes)
Size on disk:	20.0 KB (20,480 bytes)
Created:	Tuesday, September 06, 2011, 9:23:54 PM
Modified:	Tuesday, September 06, 2011, 9:30:54 PM
Accessed:	Today, September 19, 2011, 8:15:07 PM
Attributes:	☐ Read-only ☐ Hidden


11.tex Info

11.tex 111 KB
Modified: Today 2:00 PM

Spotlight Comments:

General:

Kind:	TeX Document
Size:	111,389 bytes (115 KB on disk)
Where:	/Users/greg/Dropbox/osc9e/tex
Created:	Today 1:46 PM
Modified:	Today 2:00 PM
Label:	

☐ Stationery pad
☐ Locked

More Info:

Last opened: Today 1:47 PM

Name & Extension:

11.tex

☐ Hide extension

Open with:

texmaker

Use this application to open all documents like this one.

Preview:

Sharing & Permissions:

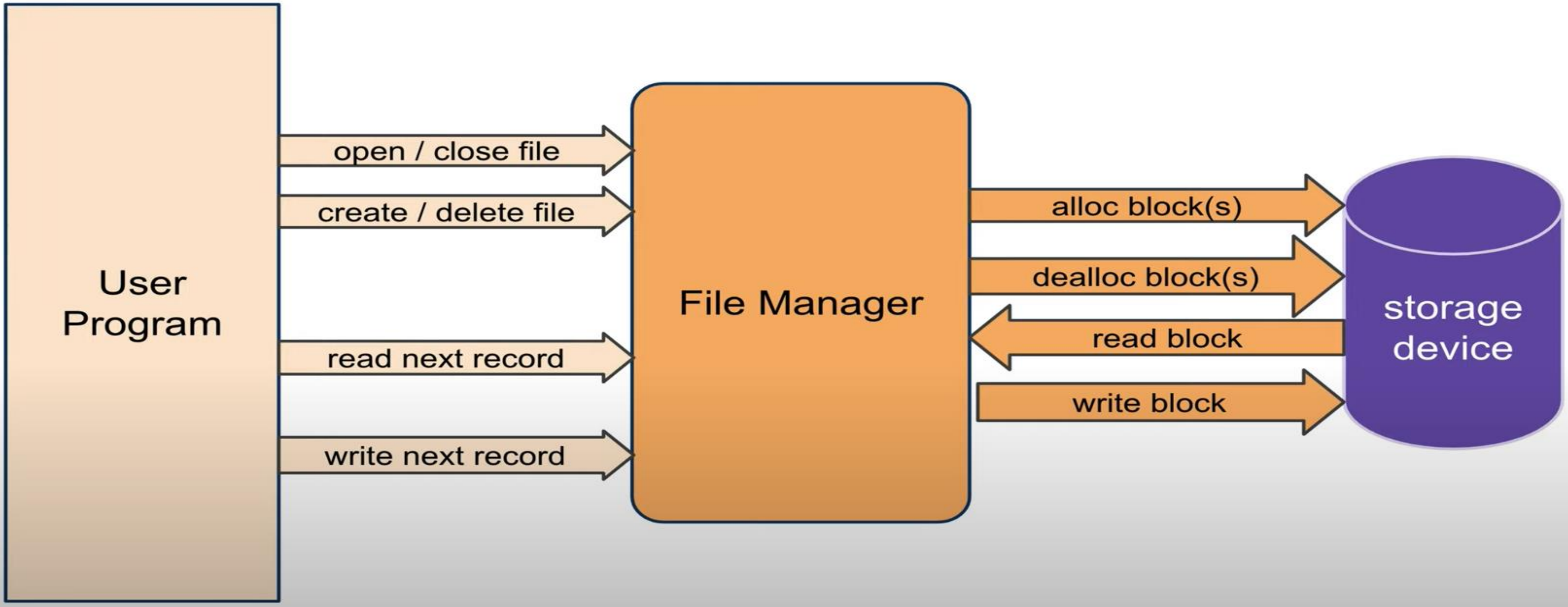
You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

FILE OPERATIONS

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry ***F_i*** in memory to directory structure on disk
- **Process:** ***open-file table***
- **OS:** ***system-wide table***

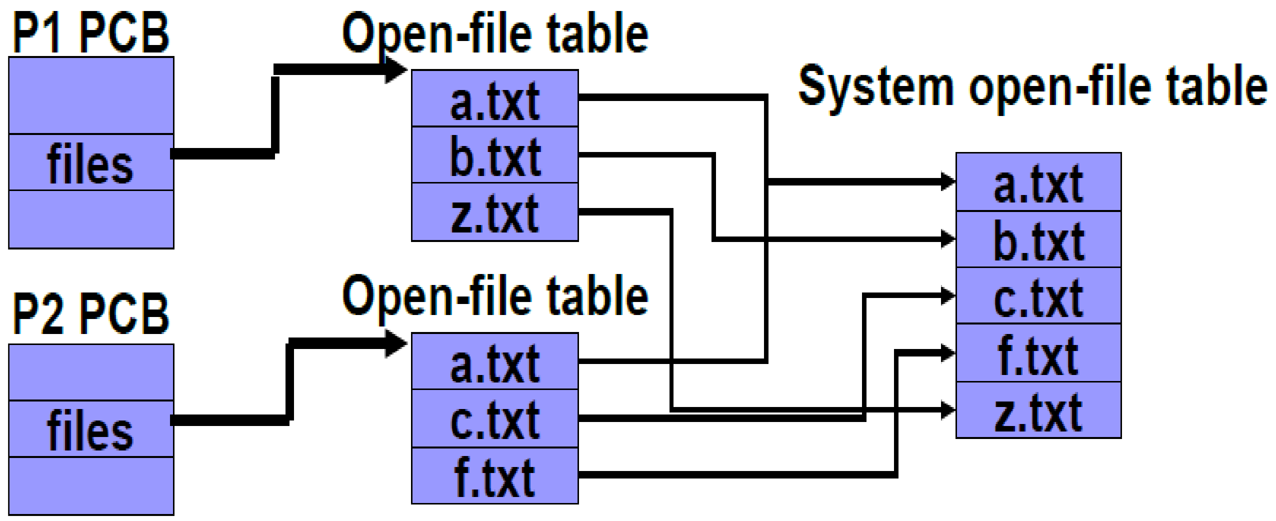
FILE OPERATIONS



block = one or more "disc" sectors/blocks

OPEN-FILE TABLES

- Per-process table
  - Tracking all files *opened by this process*
  - Current file pointer for *each opened file*
  - *Access rights* and accounting information
- System-wide table
  - Each entry in the ***per-process table*** points to this table
  - Process-independent information such as disk location, access dates, file size
  - Open count



OPEN FILE LOCKING

- Provided by some operating systems and file systems
  - Similar to **reader-writer locks**
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

FILE LOCKING EXAMPLE – JAVA API

- In the Java API, acquiring a lock requires first obtaining the FileChannel for the file to be locked.
- The lock() method of the FileChannel is used to acquire the lock.
- The API of the lock() method is

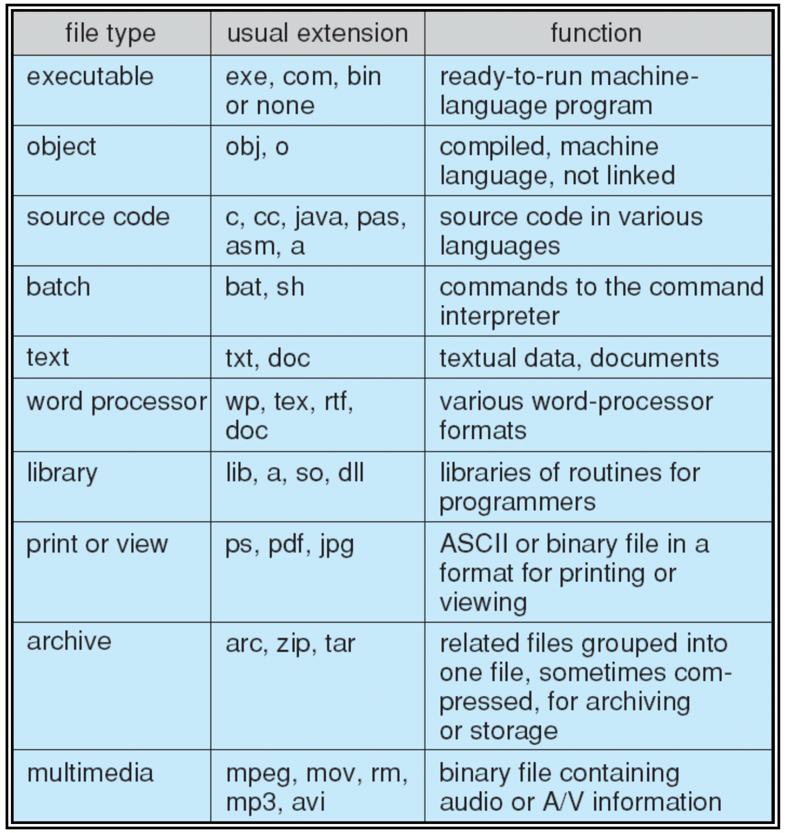
  FileLock lock(long begin, long end, boolean shared)

- Setting shared to **true is for shared locks**; setting shared to **false acquires the lock exclusively**.
- The lock is released by invoking the release() of the FileLock returned by the lock() operation

```java
FileLock sharedLock = null;
FileLock exclusiveLock = null;
RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
// get the channel for the file
FileChannel ch = raf.getChannel();
// this locks the first half of the file - exclusive
exclusiveLock = ch.lock(0, raf.length() / 2, EXCLUSIVE);
/** Now modify the data . . . */
// release the lock
exclusiveLock.release();
// this locks the second half of the file - shared
sharedLock = ch.lock(raf.length() / 2 + 1, raf.length(), SHARED);
/** Now read the data . . . */
// release the lock
sharedLock.release();
if (exclusiveLock != null)
exclusiveLock.release();
if (sharedLock != null)
sharedLock.release();
```

FILE LOCKING EXAMPLE – JAVA API

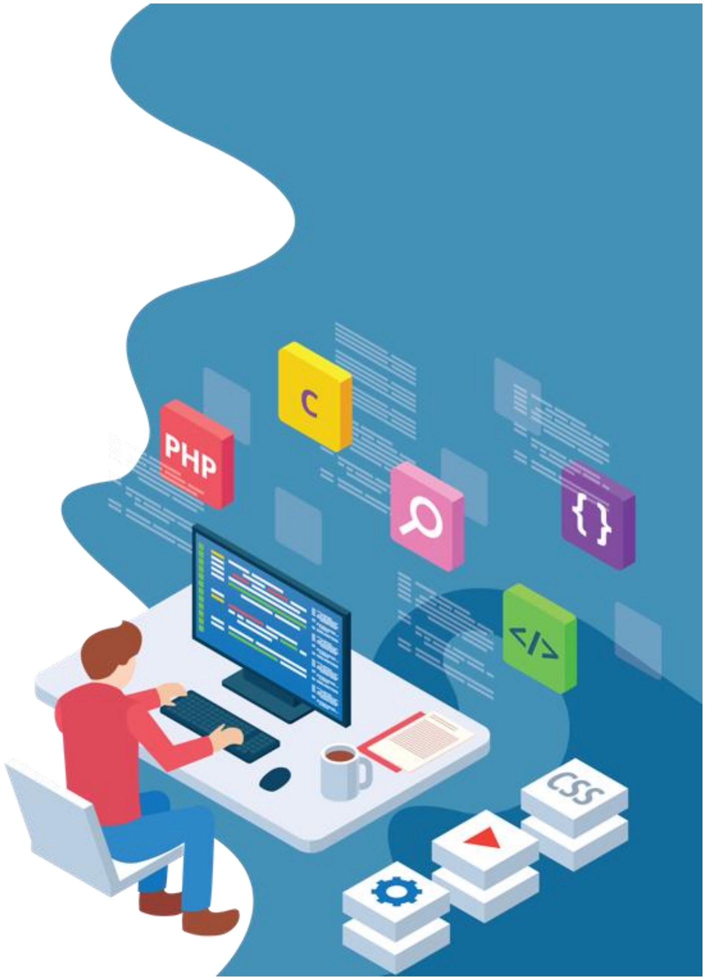
FILE TYPES – NAME, EXTENSION



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 com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

OUTLINE

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

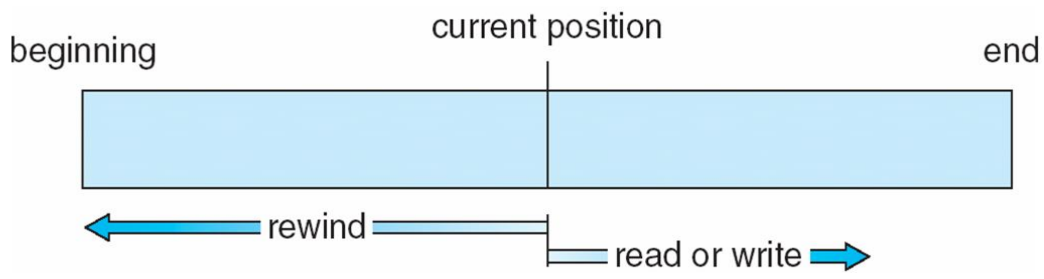


ACCESS METHODS

- Files store information.
- When it is used, this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways.
  - Sequential Access
  - Direct Access
  - Other Access Methods

SEQUENTIAL-ACCESS FILE

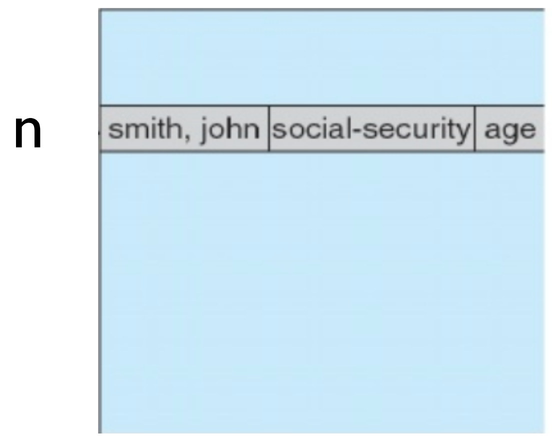
- The simplest access method
- Information in the file is processed in order



- A read operation - read next() - reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- Similarly, the write operation - write next() - appends to the end of the file and advances to the end of the newly written material (the new end of file).

DIRECT ACCESS (OR RELATIVE ACCESS)

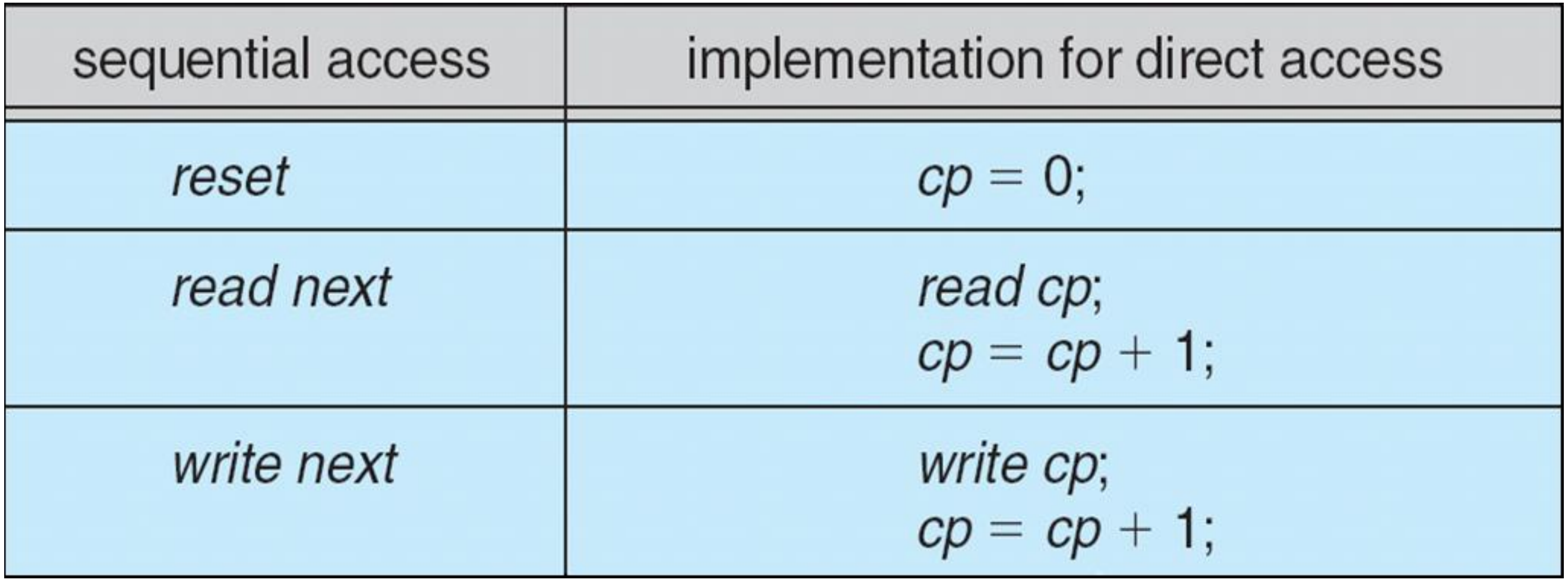
- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order
- Databases are often of this type



```
read n
write n
position to n
    read next
    write next
rewrite n
```

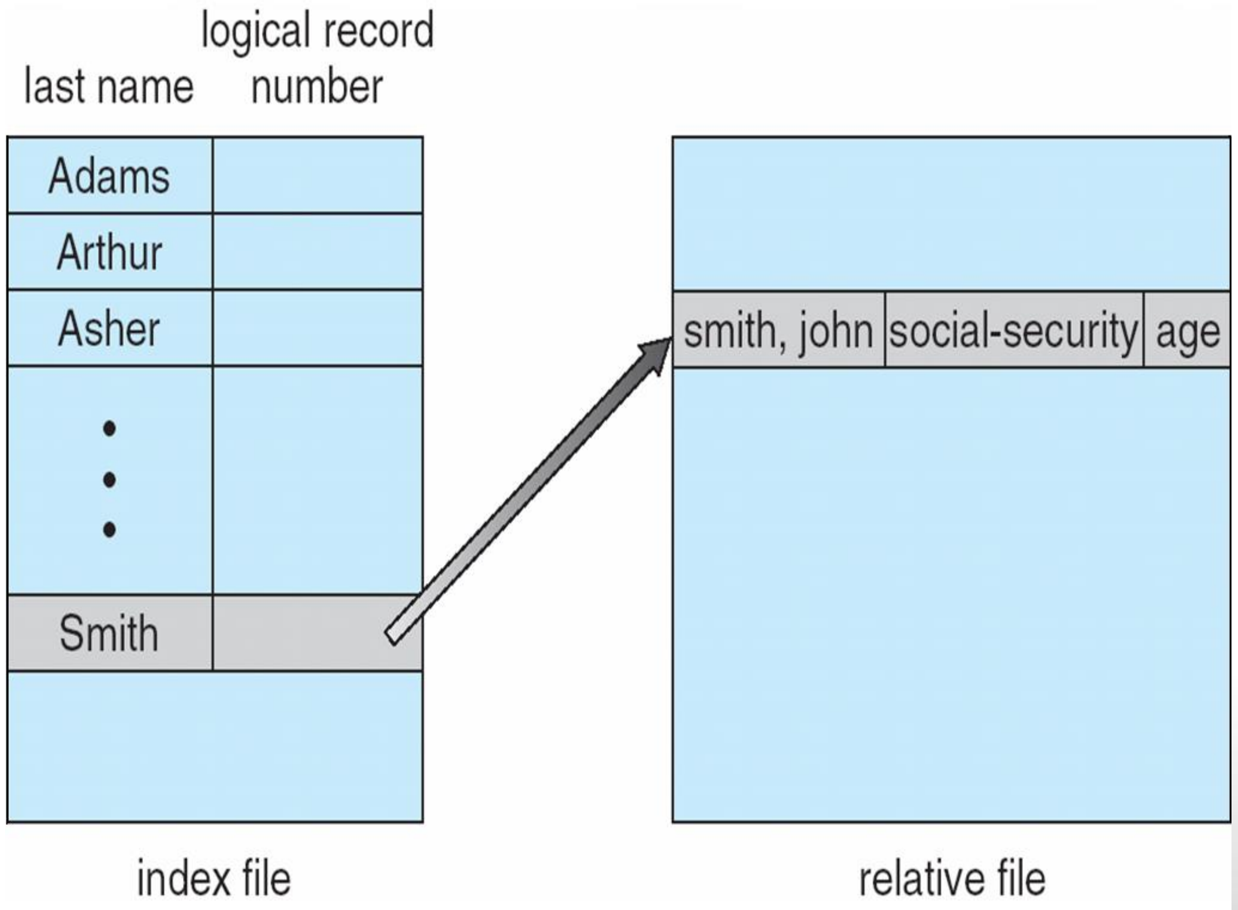
- read(n), where n is the block number

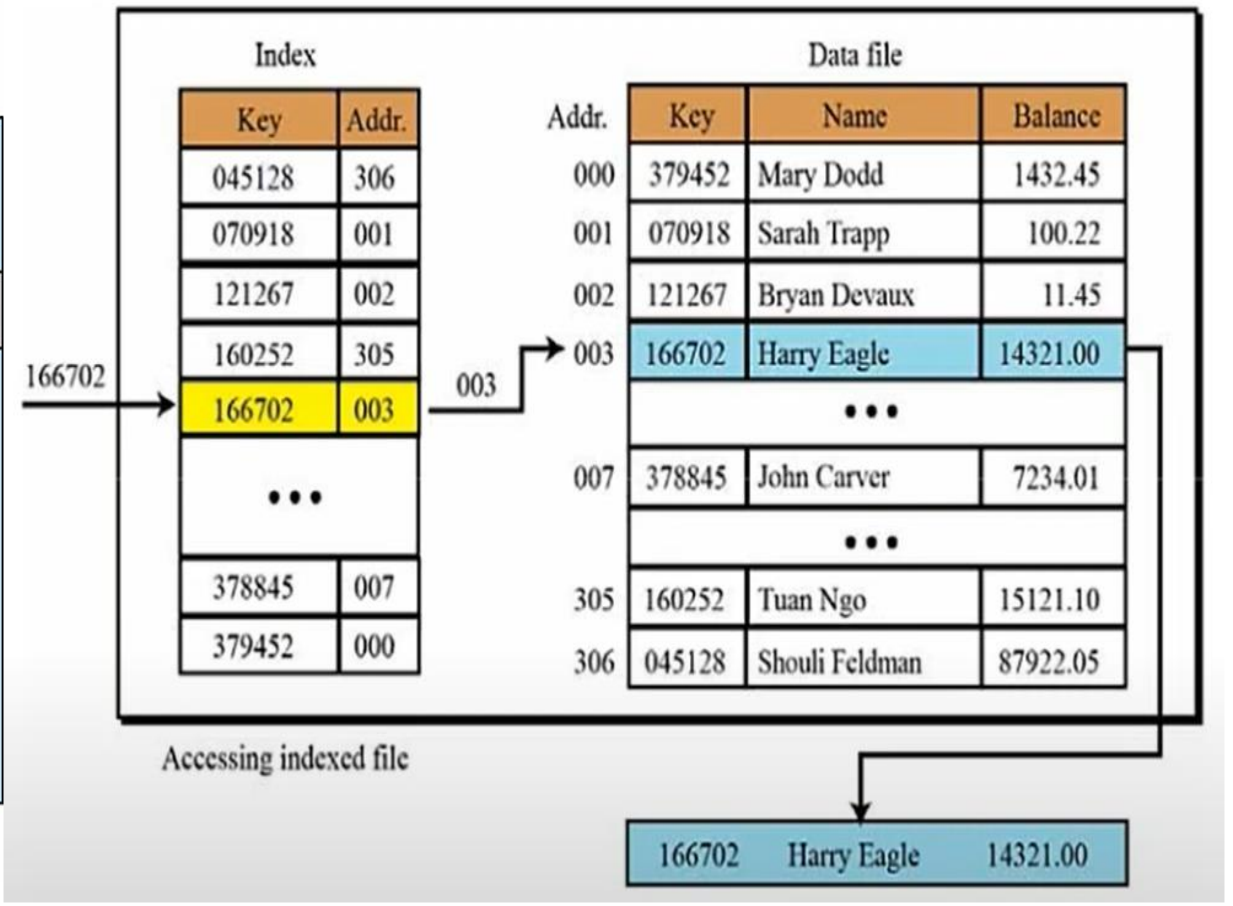
SIMULATION OF SEQUENTIAL ACCESS ON DIRECT-ACCESS FILE



sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

INDEX AND RELATIVE FILES





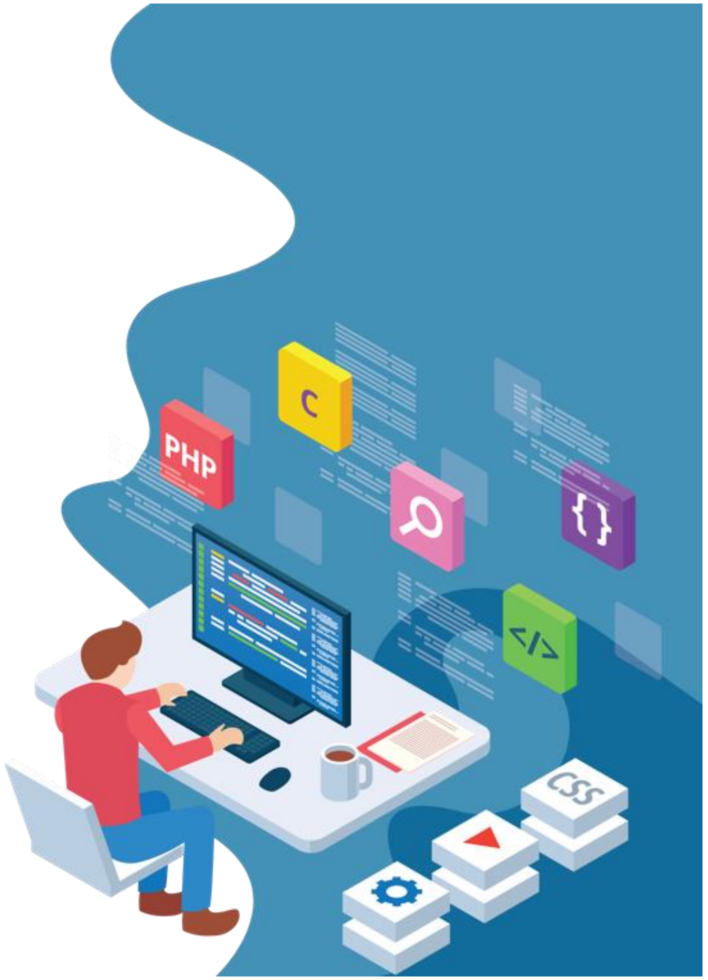
Key	Addr.
045128	306
070918	001
121267	002
160252	305
166702	003
...	
378845	007
379452	000

Addr.	Key	Name	Balance
000	379452	Mary Dodd	1432.45
001	070918	Sarah Trapp	100.22
002	121267	Bryan Devaux	11.45
003	166702	Harry Eagle	14321.00
	...		
007	378845	John Carver	7234.01
	...		
305	160252	Tuan Ngo	15121.10
306	045128	Shouli Feldman	87922.05

166702 Harry Eagle 14321.00

OUTLINE

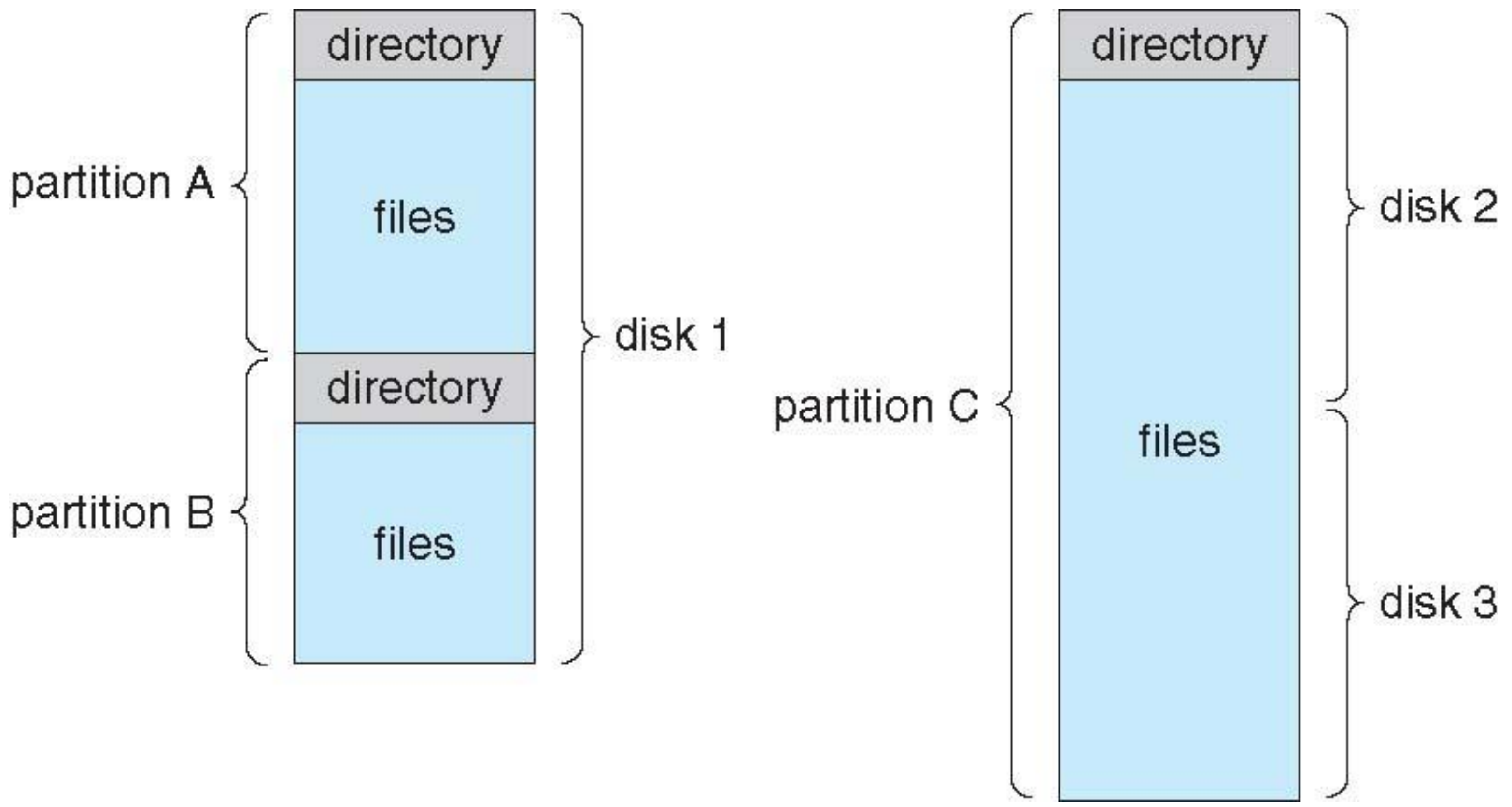
- File Interface
  - File Concept
  - Access Methods
  - ➡ Disk and Directory Structure
  - File Sharing
  - Protection
- File System Implementation



DISK STRUCTURE

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory**

A TYPICAL FILE-SYSTEM ORGANIZATION



OPERATIONS PERFORMED ON DIRECTORY

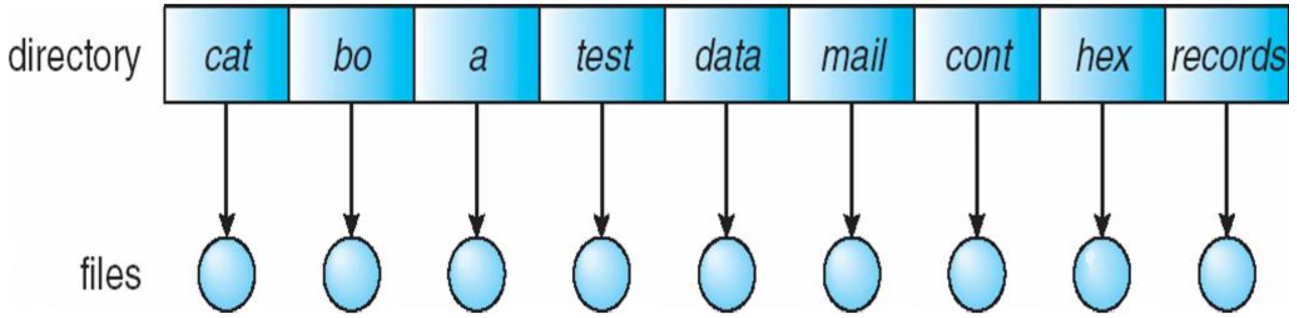
- The directory can be viewed as a symbol table that translates file names into their directory entries.
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

LOGICAL STRUCTURE OF A DIRECTORY

- The most common schemes for defining the logical structure of a directory:
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - General Graph Directory

SINGLE-LEVEL DIRECTORY

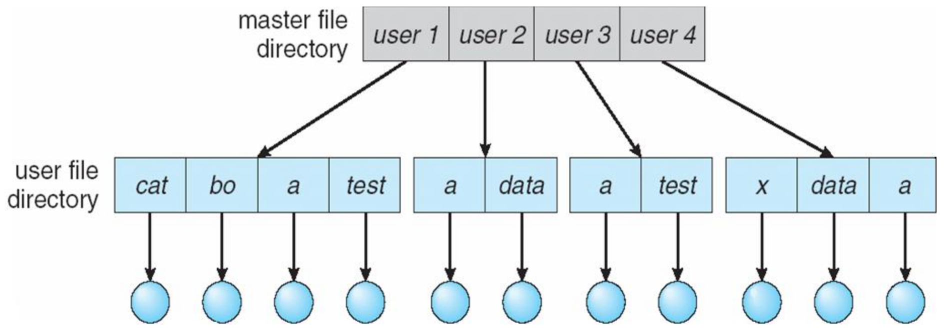
- A single directory for all users



- Naming problem
- Grouping problem

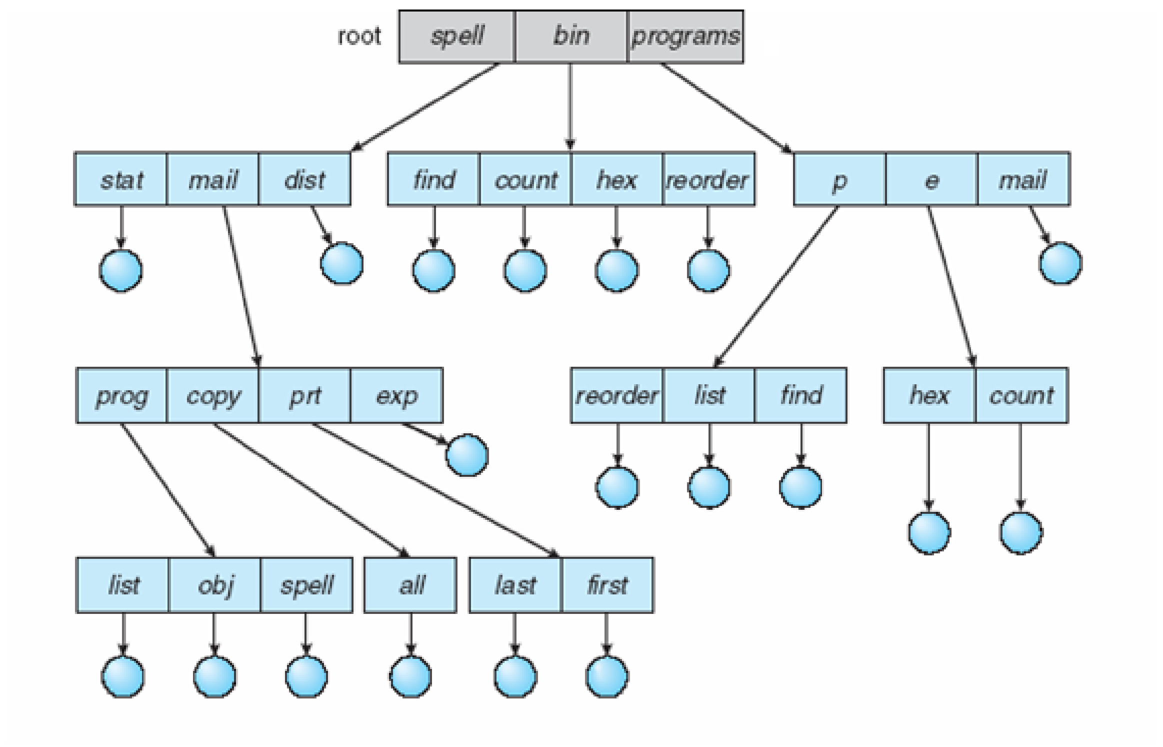
TWO-LEVEL DIRECTORY

- Each user has his own user file directory (UFD)



  - Solves the name-collision problem
  - Can have the same file name for different user
  - Efficient searching
  - No grouping capability

TREE-STRUCTURED DIRECTORIES



TREE-STRUCTURED DIRECTORIES (CONT.)

- Efficient searching
- Grouping Capability
- In normal use, each process has a **current directory** that contains most of the files that are of current interest to the process

TREE-STRUCTURED DIRECTORIES (CONT)

- **Absolute** or **relative** path name:
- An **absolute path** name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A **relative path** name defines a path from the current directory.
- For example:
  - The current directory: **root/spell/mail, then**
  - the relative path: **prt/first**
  - the absolute path name: **root/spell/mail/prt/first.**

TREE-STRUCTURED DIRECTORIES (CONT)

- Creating a new file is done in current directory
- Delete a file:

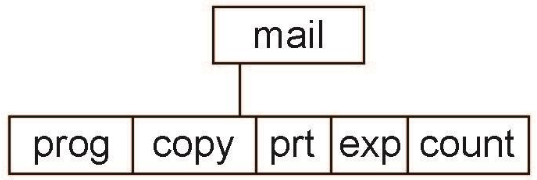
```
rm <file-name>
```

- Creating a new subdirectory:

```
mkdir <dir-name>
```

Example: if in current directory `/mail`

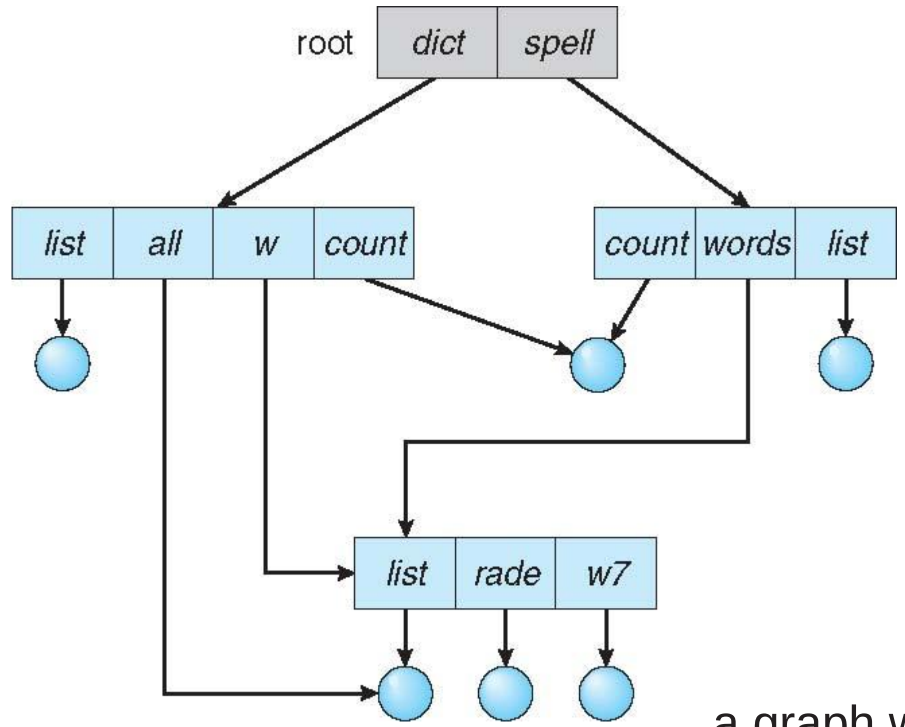
```
mkdir count
```



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

ACYCLIC-GRAPH DIRECTORIES

- Have shared subdirectories and files:

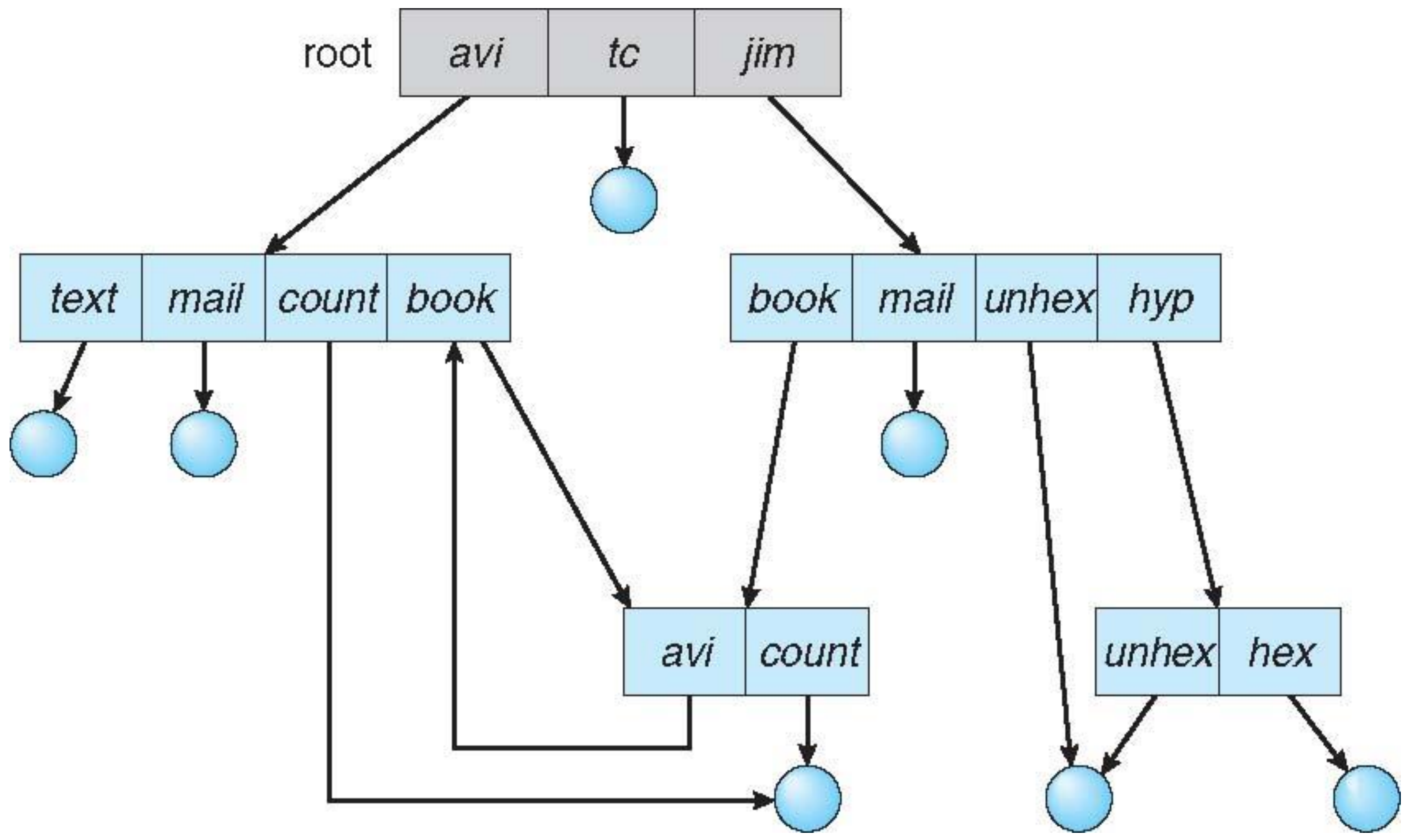


a graph with no cycles

ACYCLIC-GRAPH DIRECTORIES

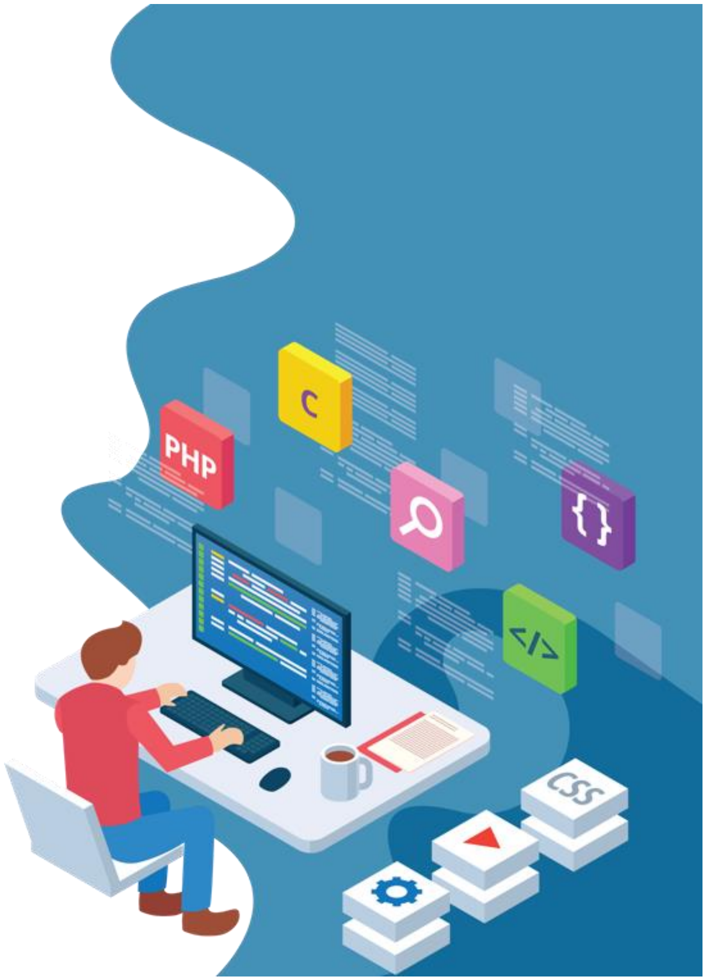
- Shared files and subdirectories can be implemented in several ways:
  - A common way, exemplified by many of the UNIX systems, is to create a new directory entry called a **link**.:
    - ln -s /mnt/my_drive/movies ~/my_movies
    - ln -s lab1.c. lab_link.c
  - Another common approach to implementing shared files is simply to **duplicate** all information about them in both sharing directories

GENERAL GRAPH DIRECTORY



OUTLINE

- File Interface
  - File Concept
  - Access Methods
  - Disk and Directory Structure
  - ➡ File Sharing
  - Protection
- File System Implementation



FILE SHARING

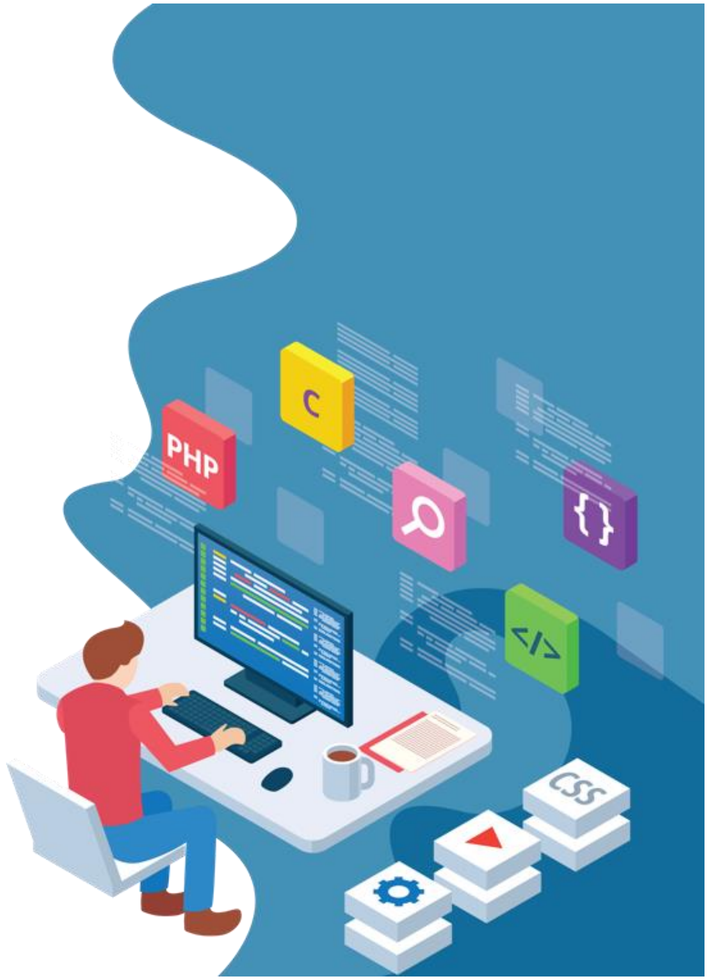
- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

FILE SHARING – REMOTE FILE SYSTEMS

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS (Network File System)** is standard UNIX client-server file sharing protocol
  - **CIFS (Common Internet File System)** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls

OUTLINE

- File Interface
  - File Concept
  - Access Methods
  - Disk and Directory Structure
  - File Sharing
  - ➡ Protection
- File System Implementation

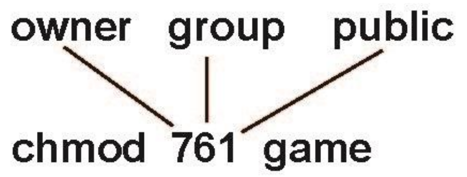


PROTECTION

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

ACCESS LISTS AND GROUPS

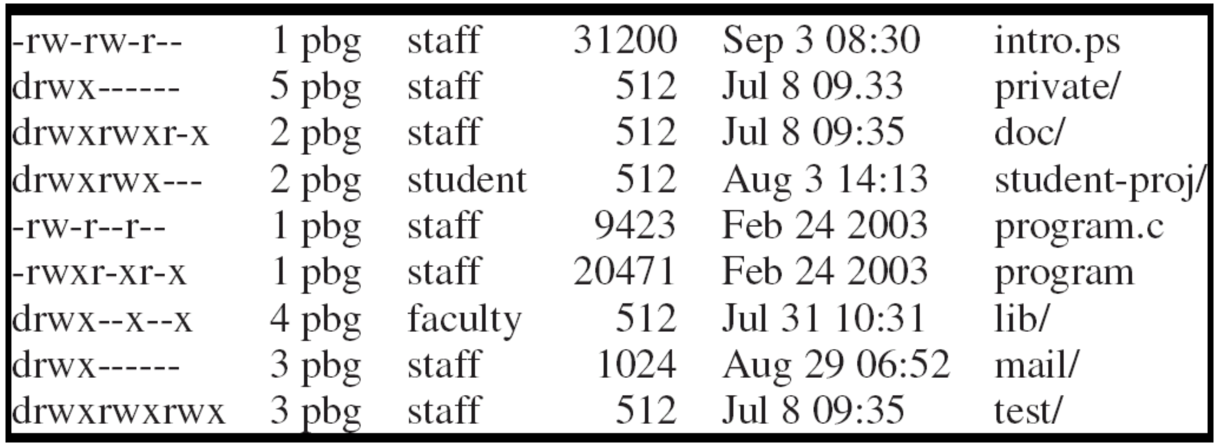
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux



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

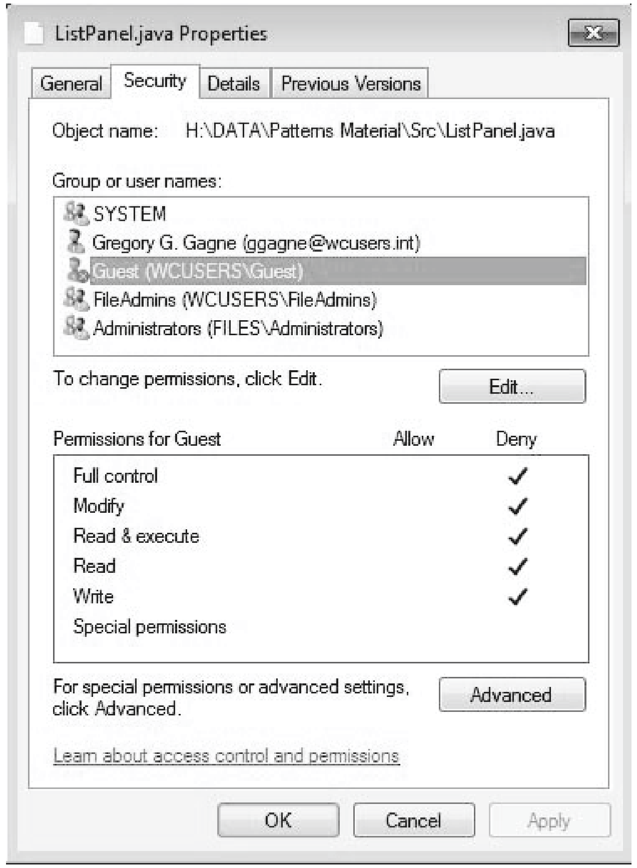
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

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/
```

WINDOWS 7 ACCESS-CONTROL LIST MANAGEMENT



Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:

- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

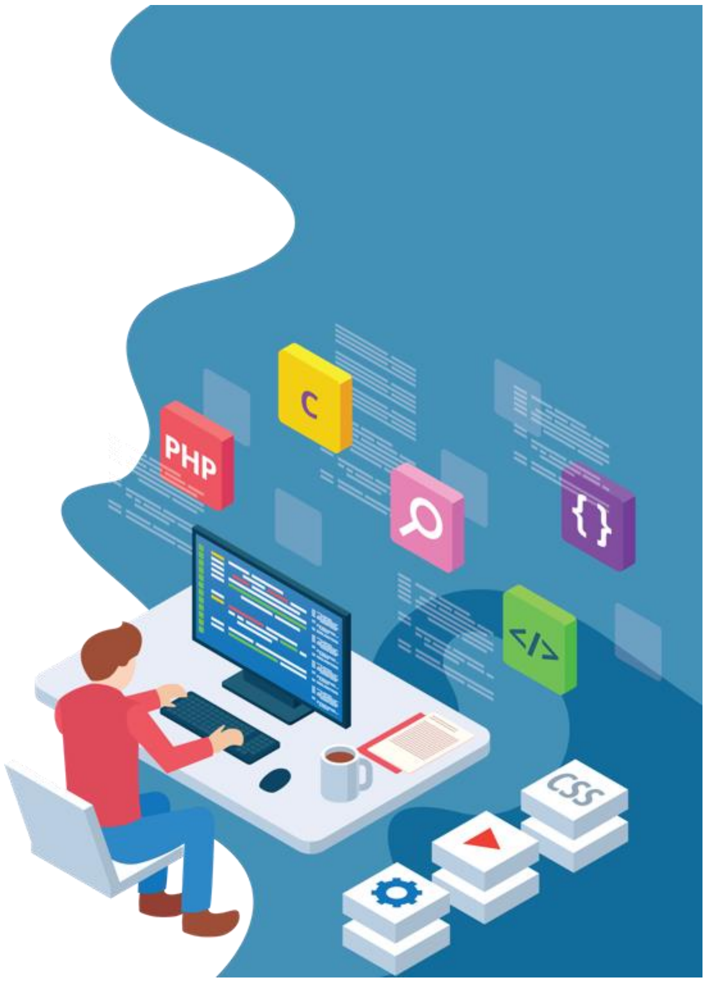
To change permissions, click Edit.

Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced.

OUTLINE

- File Interface
- File System Implementation
  - ➡ File-System Structure
  - File-System Implementation
  - Directory Implementation
  - Allocation Methods
  - Free-Space Management
  - Efficiency and Performance
  - Recovery



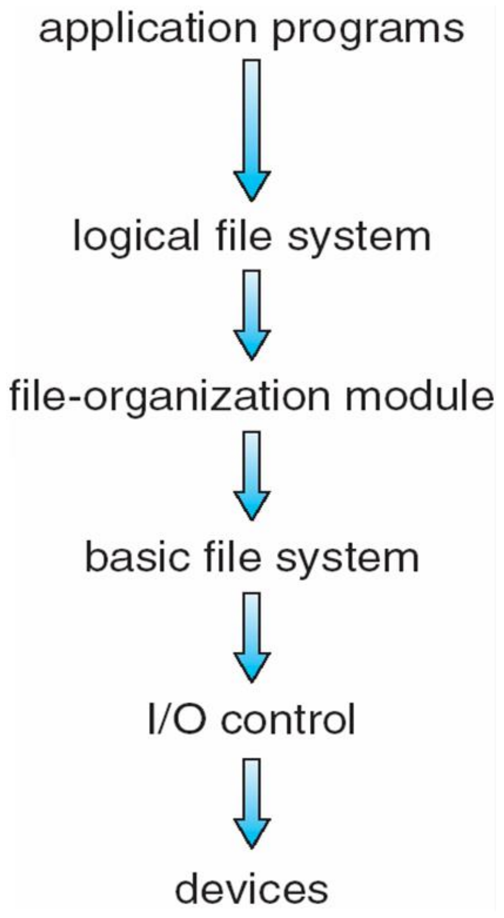
BACKGROUND

- To improve I/O efficiency, I/O transfers between memory and disk are performed in units of **blocks**. (1-n sectors) = n*512 bytes
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- **File control block** (inode in UNIX) – storage structure consisting of information about a file
- **Device driver** controls the physical device

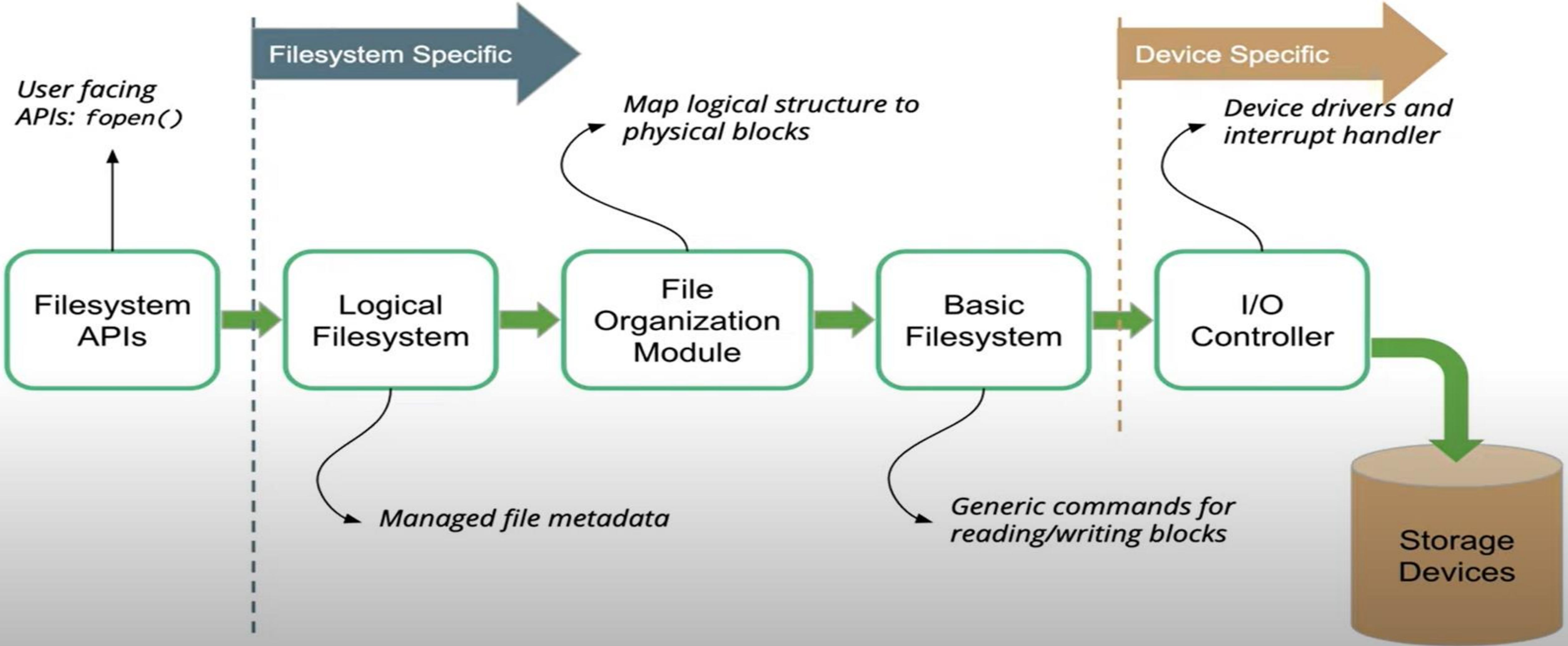
FILE-SYSTEM STRUCTURE

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

LAYERED FILE SYSTEM



LAYERDISIGN



FILE SYSTEM LAYERS

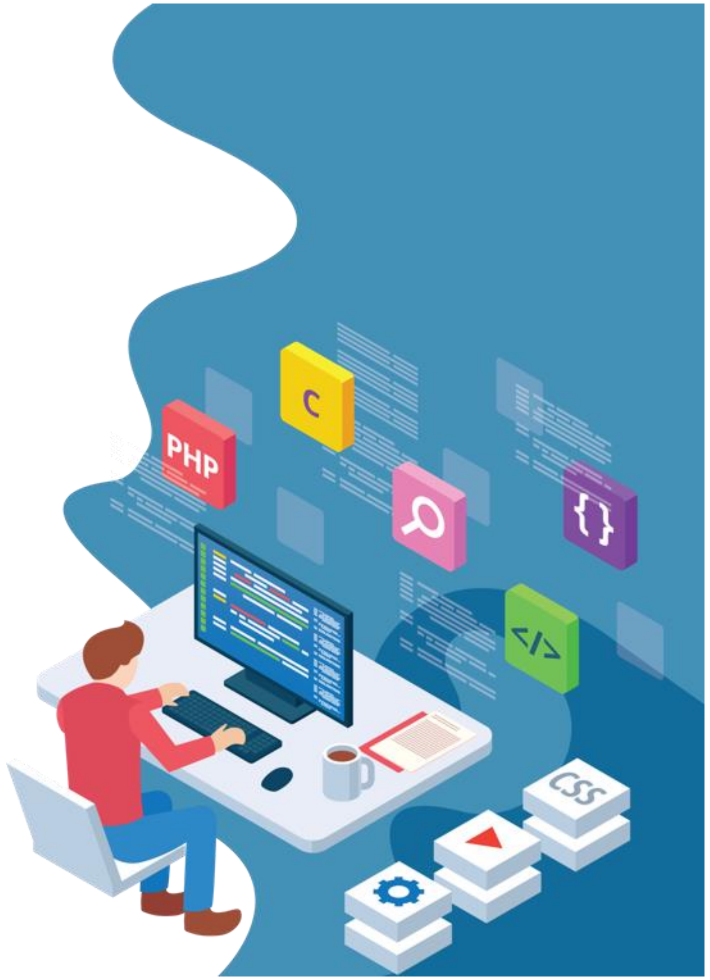
- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like : Read drive1, cylinder 72, track 2, sector 10, into memory location 1060
  - Outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
- Translates logical block # (the unit used by the **'kernel'** for read/writ operations) to physical block # (the unit used by the **'disk controller'** for read/writ operations)
- Manages free space, disk allocation

FILE SYSTEM LAYERS (CONT.)

- Many file systems, sometimes many within an operating system, Each with its own format:
  - CD-ROM is ISO 9660
  - Unix has UFS, FFS;
  - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray
  - Linux has more than 40 types, the standard Linux file system is known as the extended file system, with the most common versions being ext3 and ext4
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

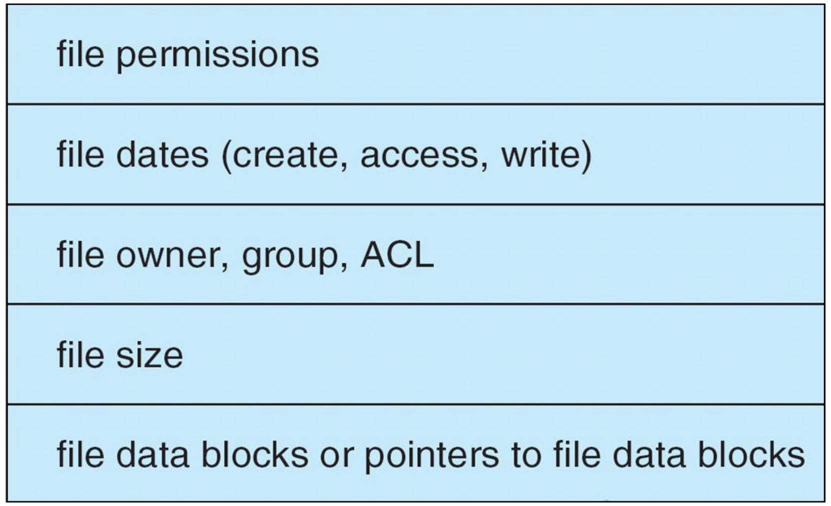
OUTLINE

- File Interface
- File System Implementation
  - File-System Structure
  - ➡ File-System Implementation
  - Directory Implementation
  - Allocation Methods
  - Free-Space Management
  - Efficiency and Performance
  - Recovery



FILE-SYSTEM IMPLEMENTATION

- To **create** a new file, an application program calls the logical file system.
- It allocates a new **File Control Block** **(FCB)** which contains many details about the file:

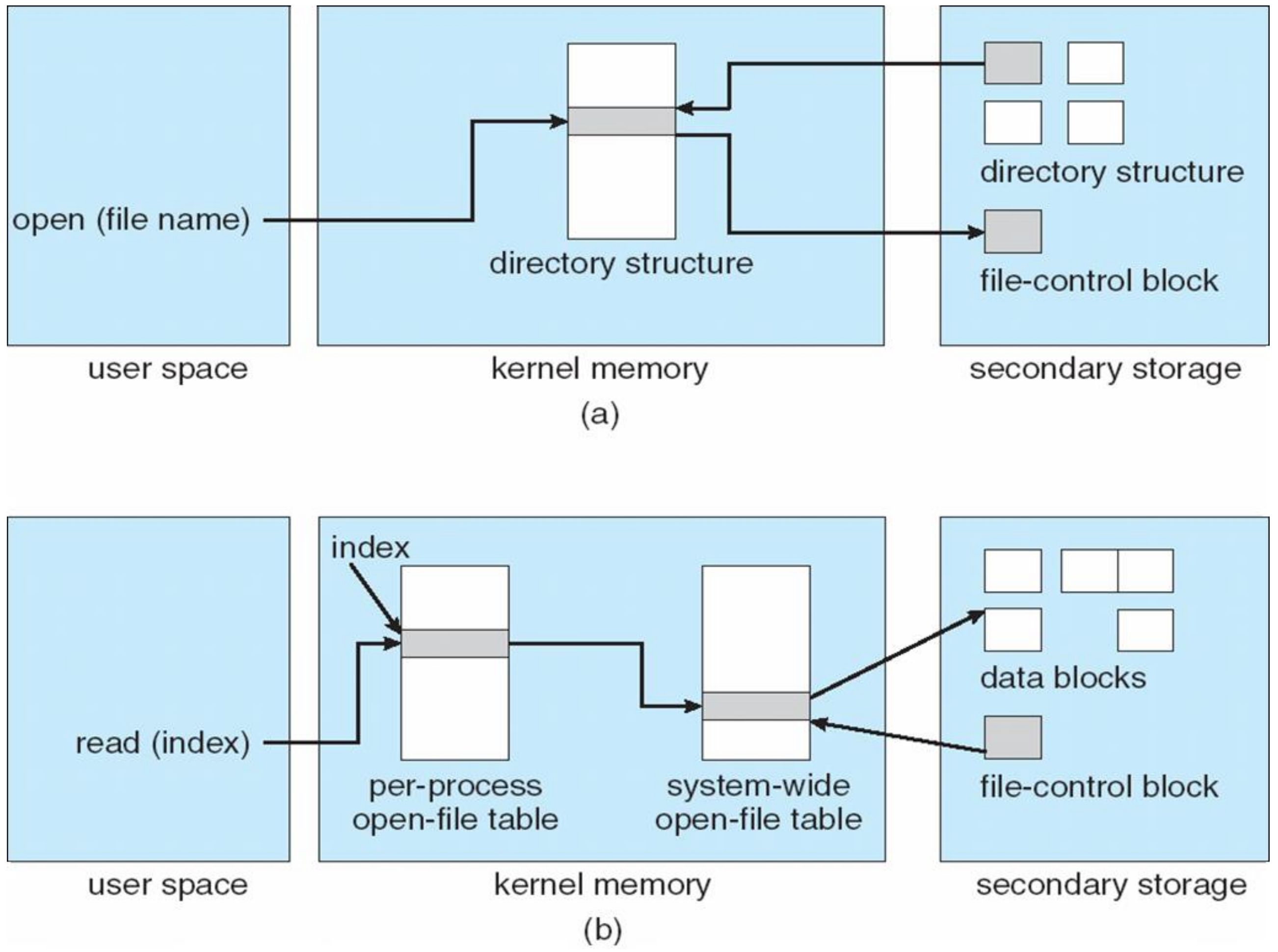


file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

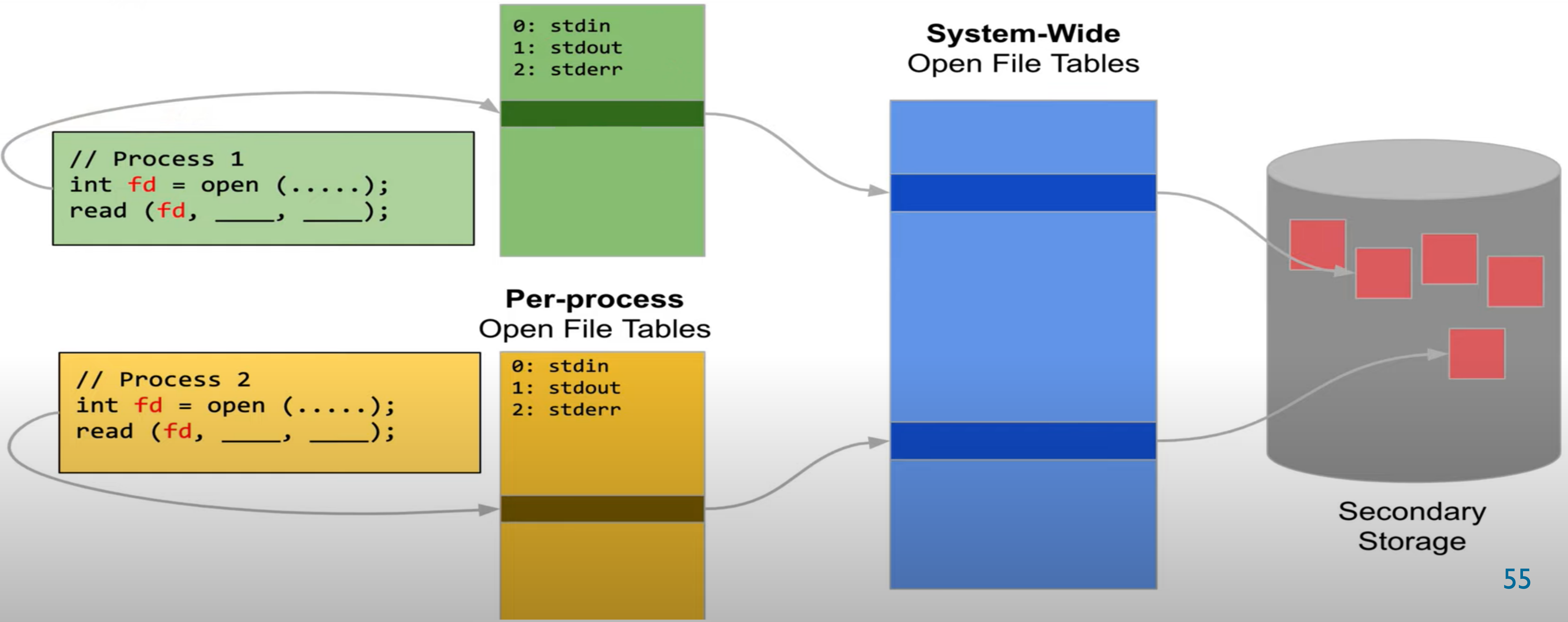
IN-MEMORY STRUCTURES

- Mount table storing file system mounts, mount points, file system types
- System-Wide Open-File Table
  - Contains a copy of the FCB, i.e., inode, of each open file
- Per-Process Open-File Table
  - Contains a pointer to the appropriate entry in the System-Wide Open-File Table

IN-MEMORY STRUCTURES

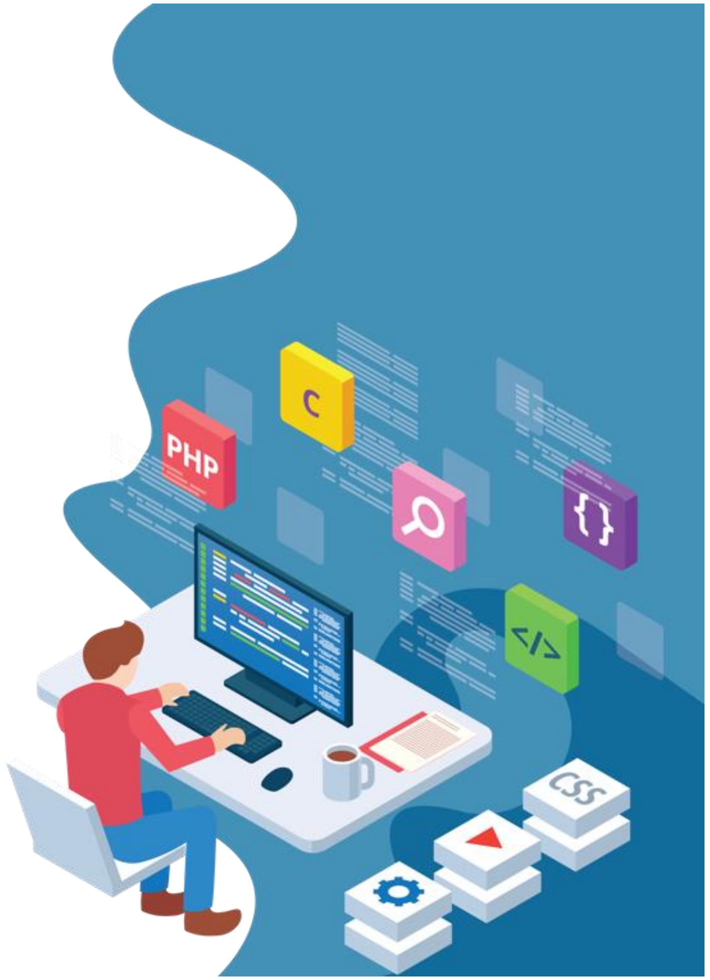


FILE HANDLES



OUTLINE

- File Interface
- File System Implementation
  - File-System Structure
  - File-System Implementation
  - ➡ Directory Implementation
  - Allocation Methods
  - Free-Space Management
  - Efficiency and Performance
  - Recovery

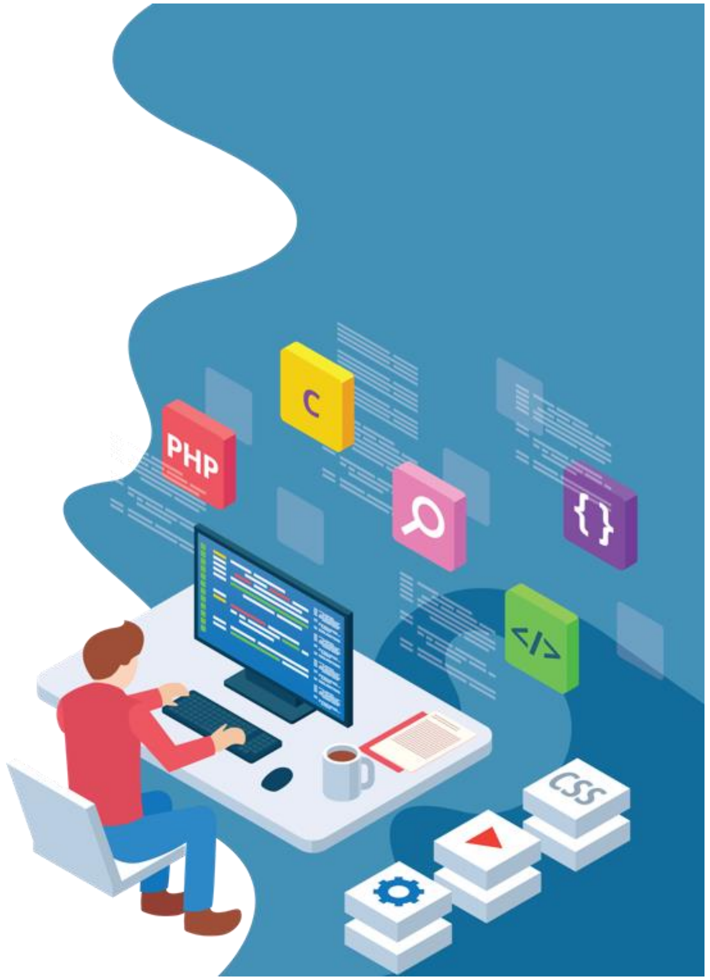


DIRECTORY IMPLEMENTATION

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

OUTLINE

- File Interface
- File System Implementation
  - File-System Structure
  - File-System Implementation
  - Directory Implementation
  - ➡ Allocation Methods
  - Free-Space Management
  - Efficiency and Performance
  - Recovery



ALLOCATION METHODS

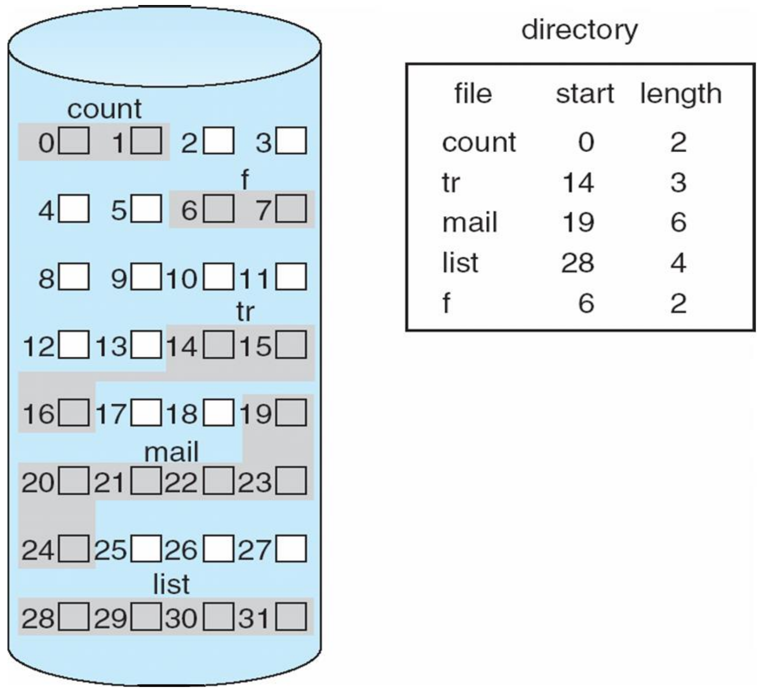
- Three major methods of allocating disk space are in wide use:
  1. Contiguous
  2. Linked
  3. Indexed.
- Although some systems support all three, it is more common for a system to use one method for all files within a file-system type.

CONTIGUOUS ALLOCATION

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

CONTIGUOUS ALLOCATION (CONT)

- Mapping from logical to physical

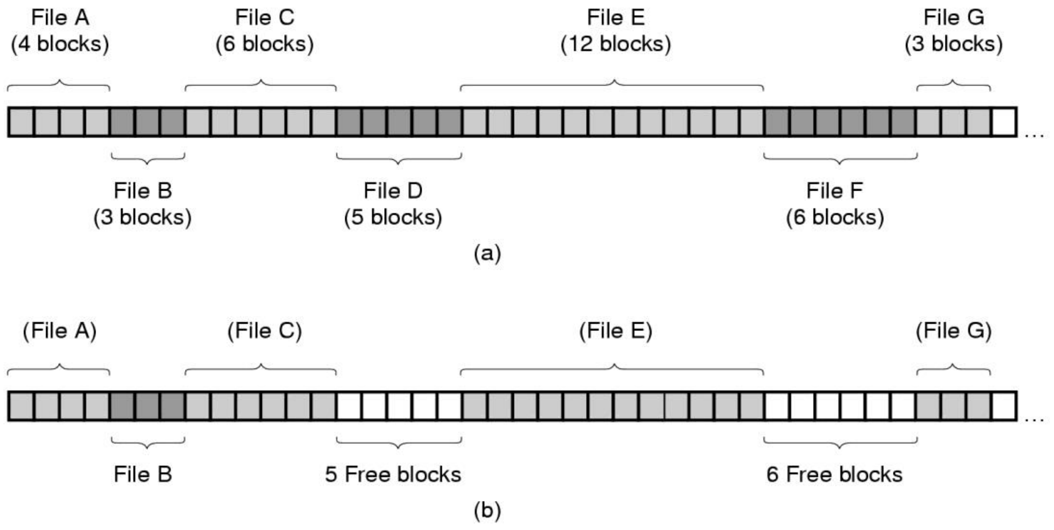


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

CONTIGUOUS ALLOCATION (CONT)

- Problems:
  - finding space for file (determining how much space is needed for a file?
  - external fragmentation



EXTENT-BASED SYSTEMS

- Many newer file systems (i.e.,Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

LINKED ALLOCATION

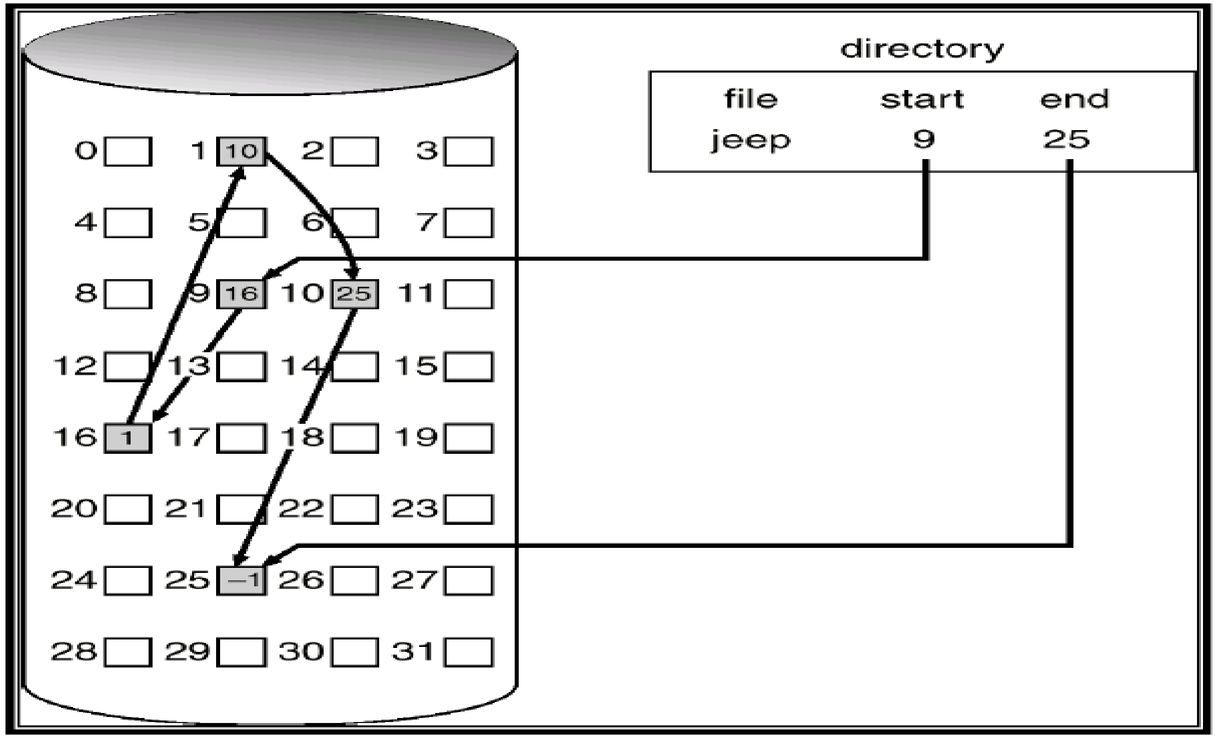
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - Each block contains pointer to next block
  - No external fragmentation, compaction
  - Free space management system called when new block needed

LINKED ALLOCATION

- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk
- The directory contains a pointer to the first and last blocks of the file.
- For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25.
- Each block contains a pointer to the next block.
- If each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.

LINKED ALLOCATION

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- Scheme



LINKED ALLOCATION

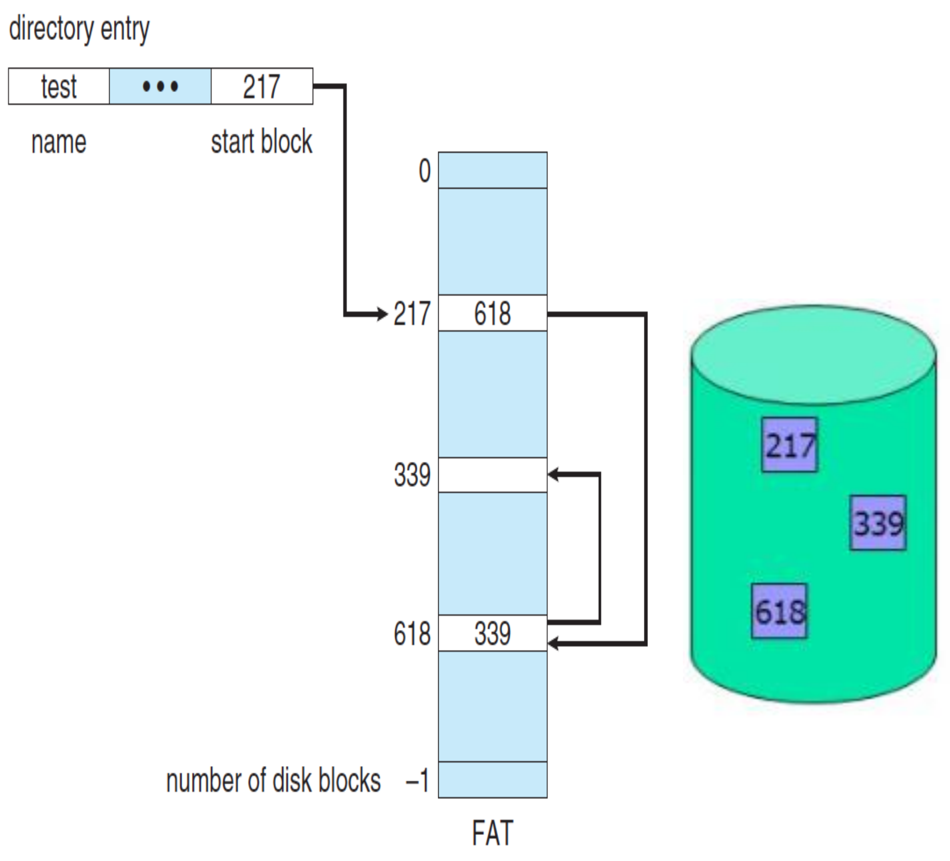
- Create a new file -> create a new entry in the directory. Each directory entry has a pointer to the first disk block of the file.
- This pointer is initialized to null (the end-of-list pointer value) to signify an empty file.
- A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file.
- To read a file, we simply read blocks by following the pointers from block to block.
- The size of a file need not be declared when the file is created ➔ File can continue to grow as long as free blocks are available.
- The usual solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks ➔increase in internal fragmentation

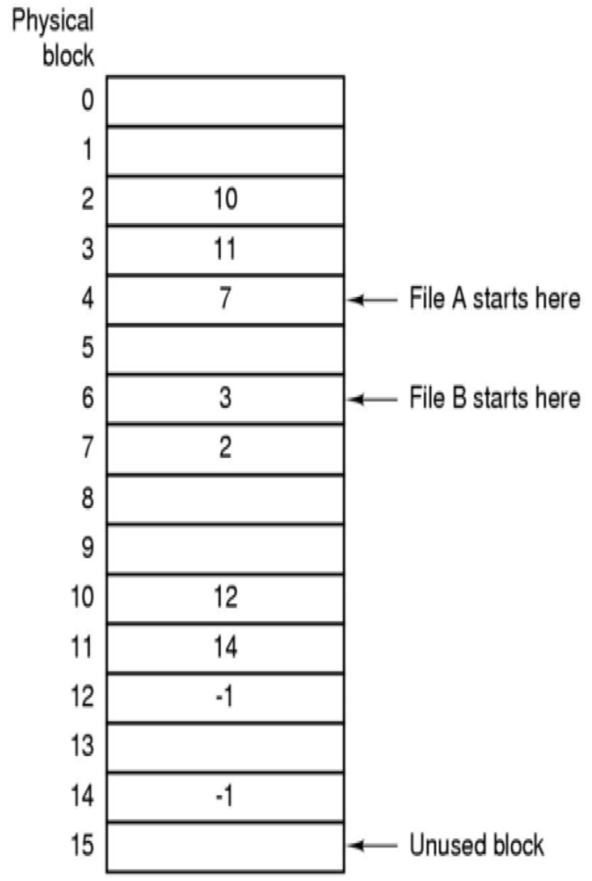
LINKED ALLOCATION- RELIABILITY

- Recall that the files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged.
- A bug in the operating-system software or a disk hardware failure might result in picking up the wrong pointer.
- This error could in turn result in linking into the free-space list or into another file.

FAT ALLOCATION METHOD (FAT)

- An important variation on linked allocation is the use of a file-allocation table (FAT)
- Beginning of volume has table, indexed by block number
- Much like a linked list, but faster on disk and cacheable
- New block allocation simple

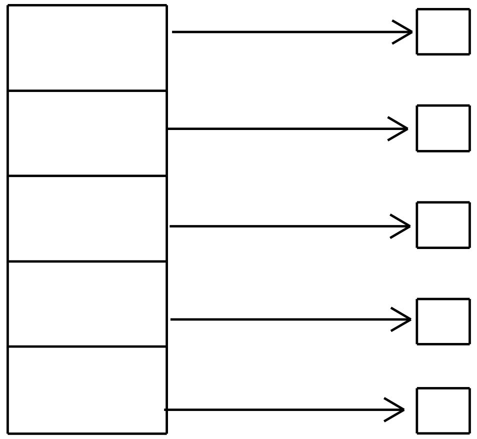




FAT example

INDEXED ALLOCATION

- Each file has its own **index block**(s) of pointers to its data blocks
- Logical view:

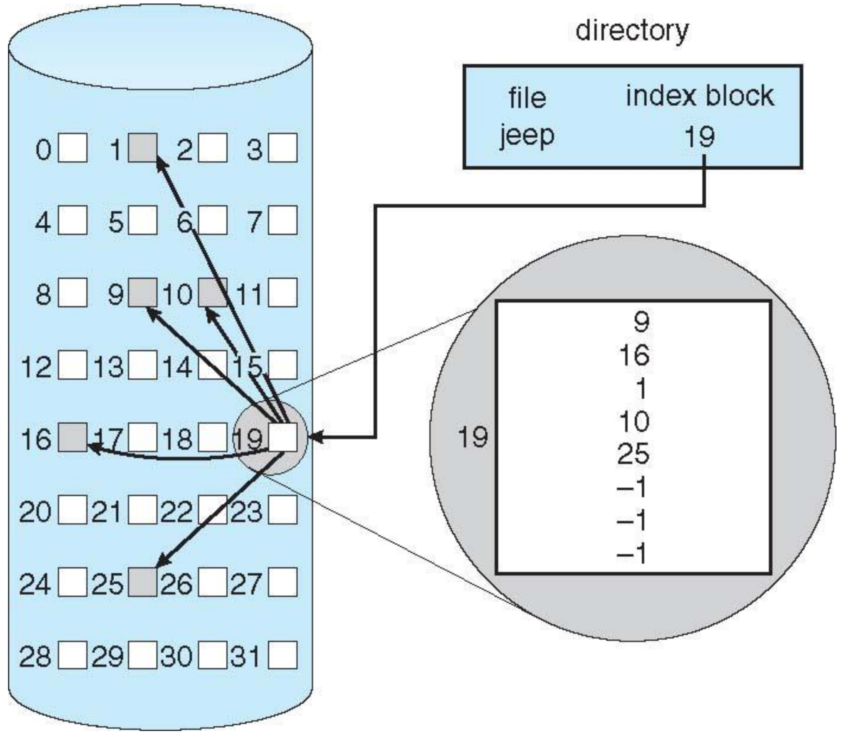


index table

INDEXED ALLOCATION

- The directory contains the address of the index block.
- To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.
- This scheme is similar to the paging scheme
- When the file is created, all pointers in the index block are set to null.
  - When the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} index-block entry.

EXAMPLE OF INDEXED ALLOCATION



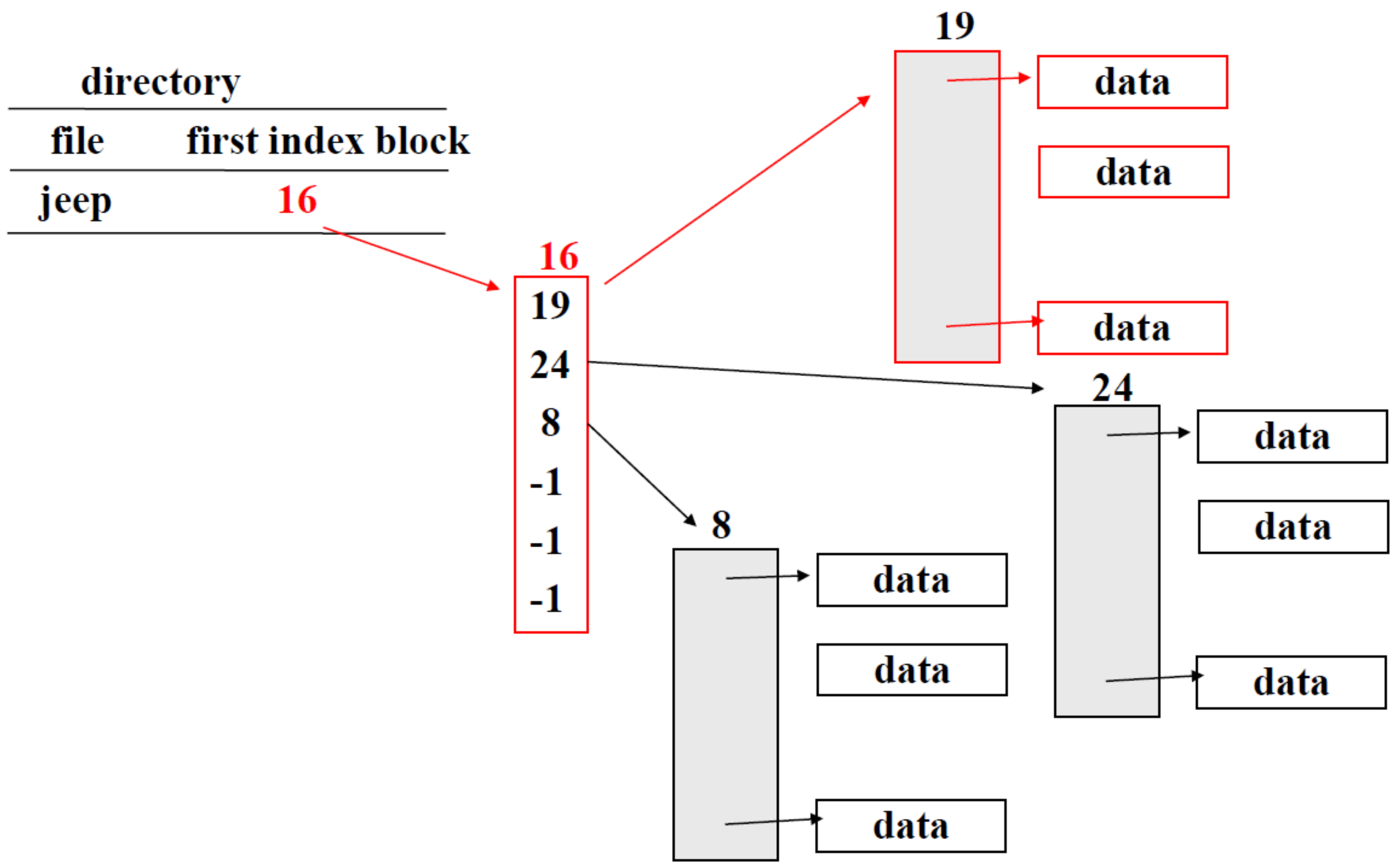
INDEXED ALLOCATION – SMALL FILES

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Consider a common case in which we have a file of only one or two blocks:
  - With linked allocation, we lose the space of only one pointer per block
  - With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-null.

(Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table)

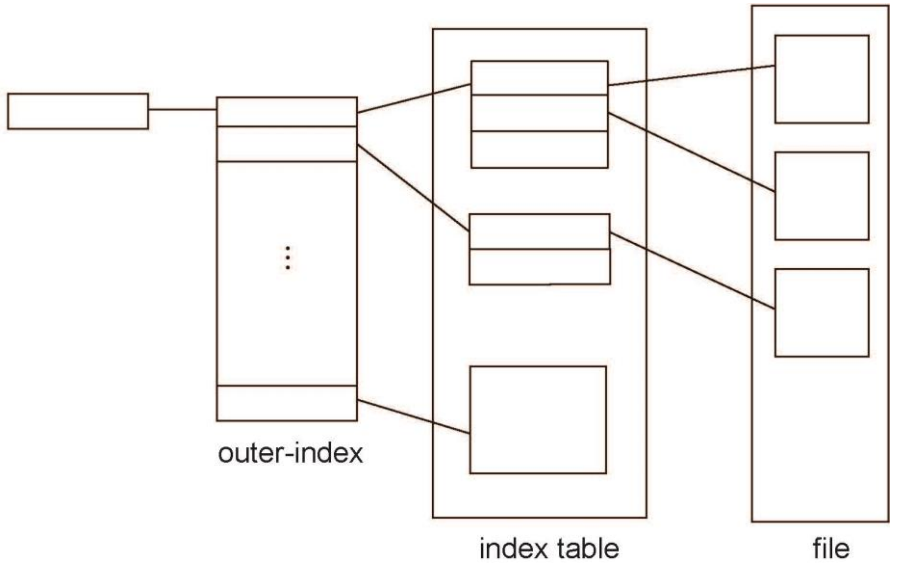
INDEXED ALLOCATION – LARGE FILES

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
  - Linked scheme – Link blocks of index table (no limit on size)
  - Multi-level indexing

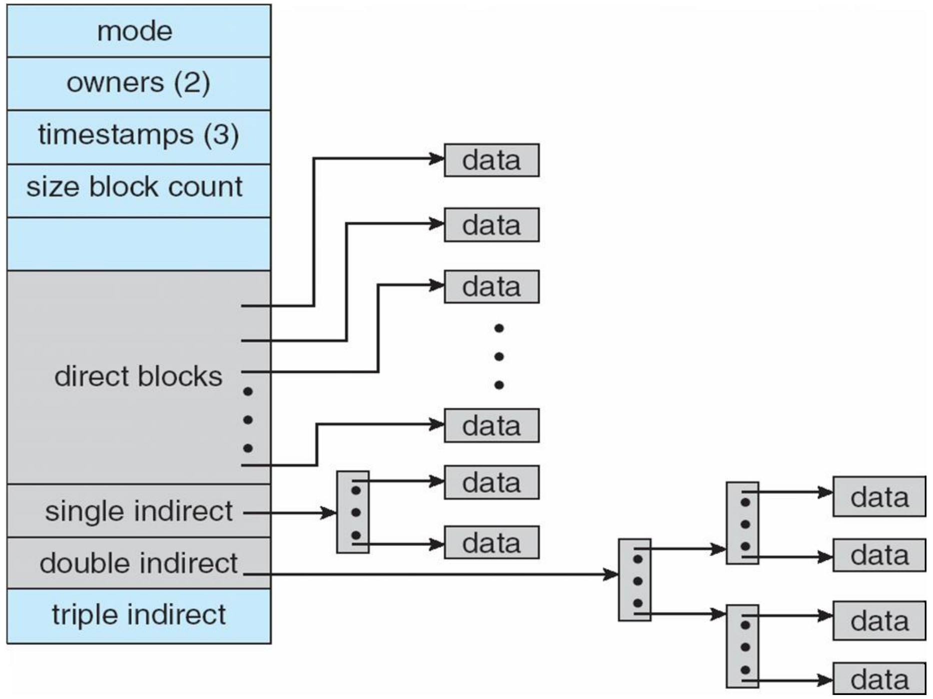


INDEXED ALLOCATION – MAPPING (CONT.)

- With 4,096-byte blocks, we could store 1,024 four-byte pointers in an index block. Two levels of indexes allow 1,048,576 data blocks and a file size of up to 4 GB.



COMBINED SCHEME: UNIX UFS

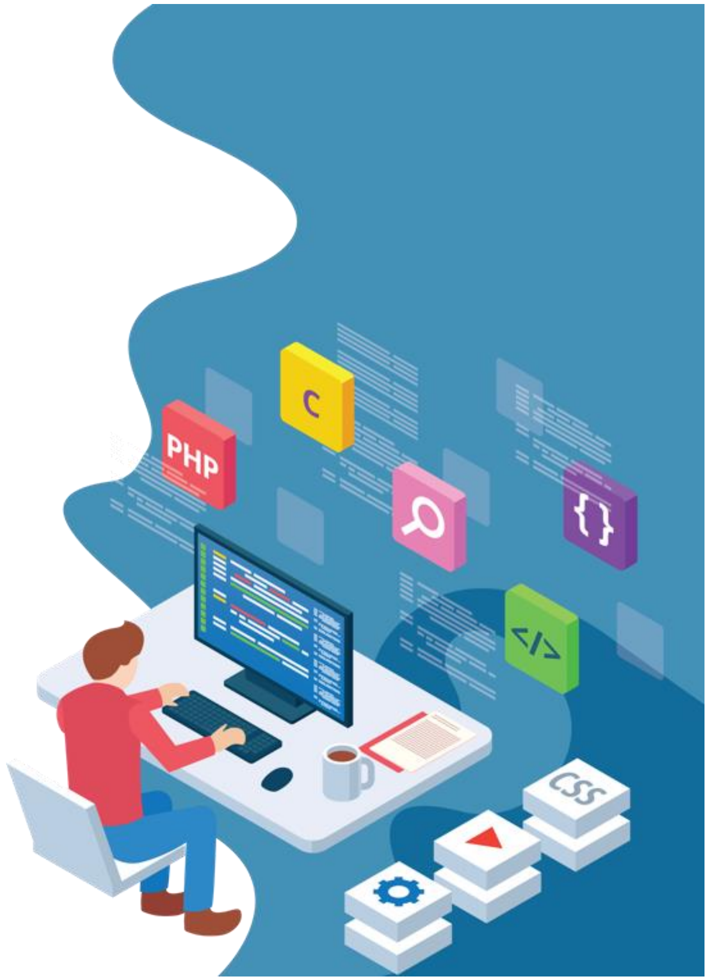


PERFORMANCE

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

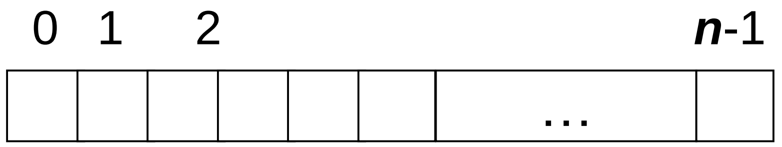
OUTLINE

- File Interface
- File System Implementation
  - File-System Structure
  - File-System Implementation
  - Directory Implementation
  - Allocation Methods
  - ➡ Free-Space Management
  - Efficiency and Performance
  - Recovery



FREE-SPACE MANAGEMENT

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (***n*** blocks)



$$\text{bit}[\boldsymbol{i}] = \begin{cases} 1 \Rightarrow \text{block}[\boldsymbol{i}] \text{ free} \\ 0 \Rightarrow \text{block}[\boldsymbol{i}] \text{ occupied} \end{cases}$$

- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be:

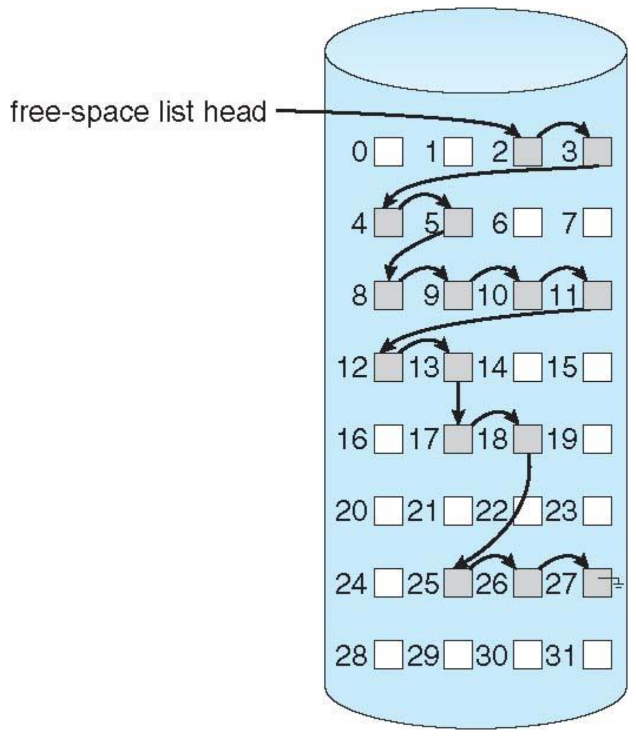
001111001111110001100000011100000 ...

FREE-SPACE MANAGEMENT (CONT.)

- Bit map requires extra space (in main memory)
  - A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks, although clustering the blocks in groups of four reduces this number to around 83 KB per disk.
  - A 1-TB disk with 4-KB blocks requires 256 MB to store its bit map.
- Given that disk size constantly increases, the problem with bit vectors will continue to escalate as well.

LINKED FREE SPACE LIST ON DISK

- Linked list (free list)
- Keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- Update last point when release

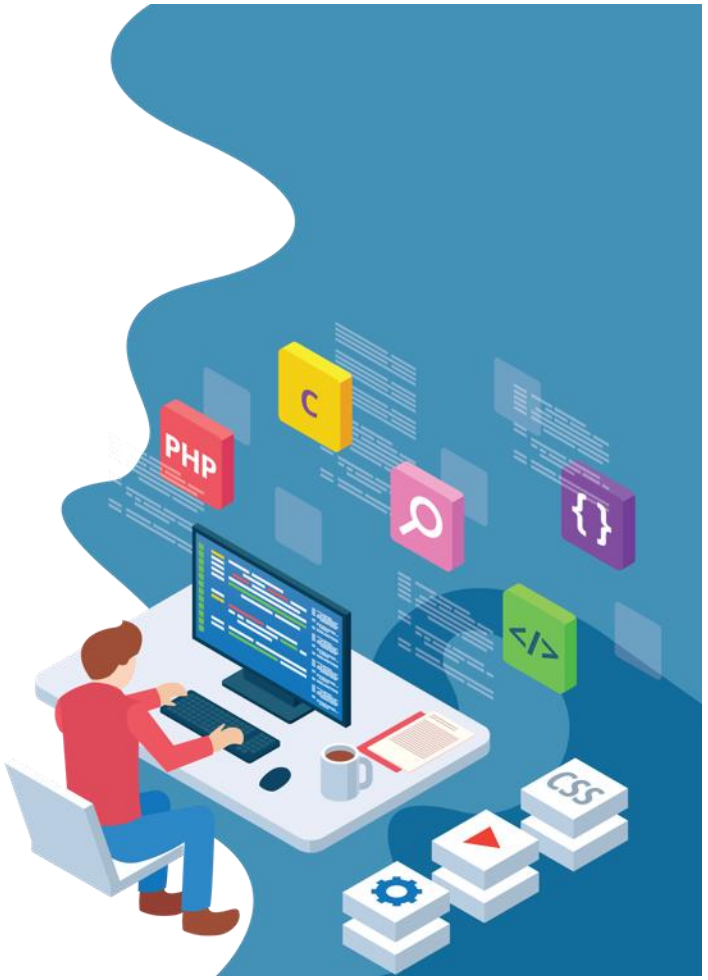


FREE-SPACE MANAGEMENT (CONT.)

- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

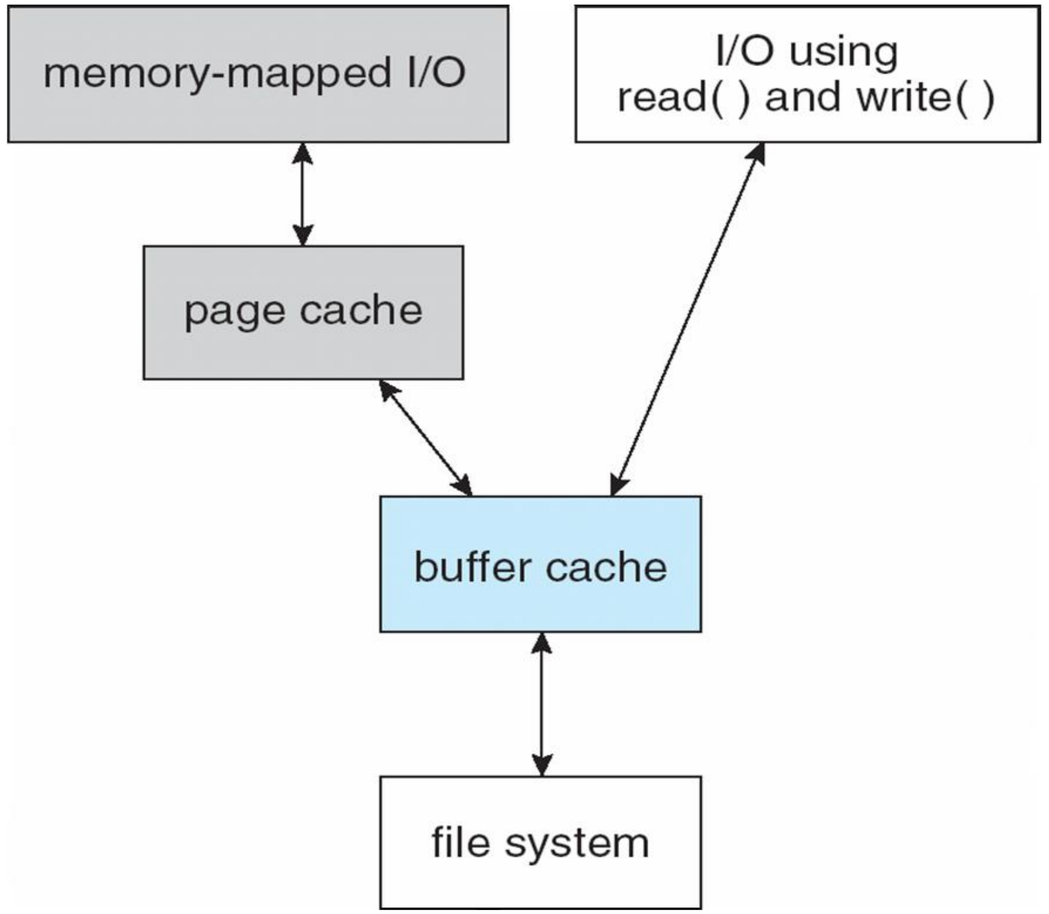
OUTLINE

- File Interface
- File System Implementation
  - File-System Structure
  - File-System Implementation
  - Directory Implementation
  - Allocation Methods
  - Free-Space Management
  - Efficiency and Performance
  - Recovery



PERFORMANCE

- Some systems maintain a separate section of main memory for a buffer cache, where blocks are kept under the assumption that they will be used again shortly.
- Other systems cache file data using a page cache which uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks



RECOVERY

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

RECOVERY (CONT)

- A typical backup schedule may then be as follows:
  - **Day 1**. Copy to a backup medium all files from the disk. This is called a full backup.
  - **Day 2**. Copy to another medium all files changed since day 1. This is an incremental backup.
  - **Day 3**. Copy to another medium all files changed since day 2.

…

  - **Day N**. Copy to another medium all files changed since day N−1. Then go back to day 1.
- Using this method, we can restore an entire disk by starting restores with the full backup and continuing through each of the incremental backups