# CS343 - Operating Systems

## Module-6A

## Introduction to Files & Directories


**Dr. John Jose**

**Assistant Professor**

**Department of Computer Science & Engineering**

**Indian Institute of Technology Guwahati, Assam.**

http://www.iitg.ac.in/johnjose/

# File-System Interface

- 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

# 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 Attributes

- **Name** – users identify a file with name.
- **Identifier** – unique number identifies file within file system
- **Type** – Format of data inside, application that can access it.
- **Location** – pointer to file location on device
- **Size** – amount of storage the file consumes
- **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

# File Types & 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 compressed, for archiving or storage |
| multimedia | mpeg, mov, rm, mp3, avi | binary file containing audio or A/V information |

# File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- **Open(F)** – search the directory structure on disk for entry **F**, and move the content of entry to memory
- **Close (F)** – move the content of entry **F** in memory to directory structure on disk

# File Open Operation

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
  - File pointer:  pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

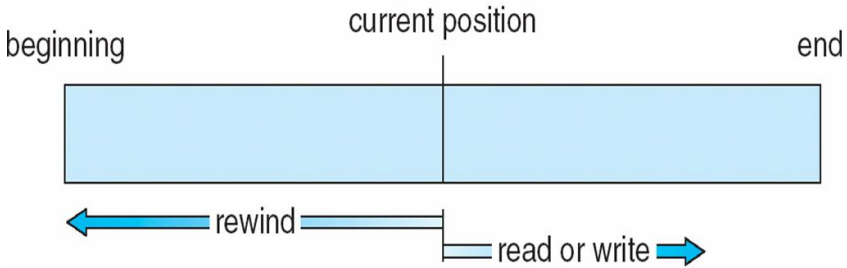
# Open File Locking

- **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

# Access Methods

- **Sequential Access**

```
read next
write next
reset
```



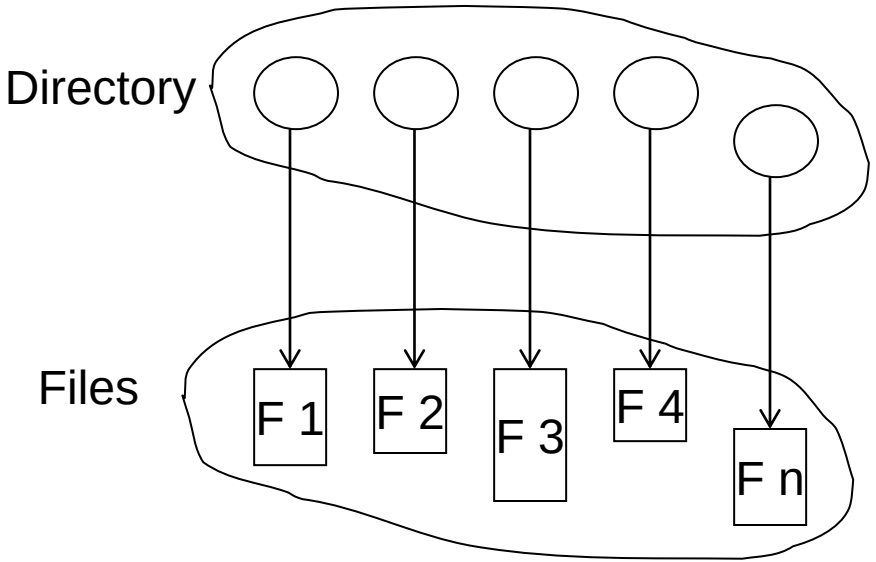

- **Direct Access**

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

*n* = relative block number

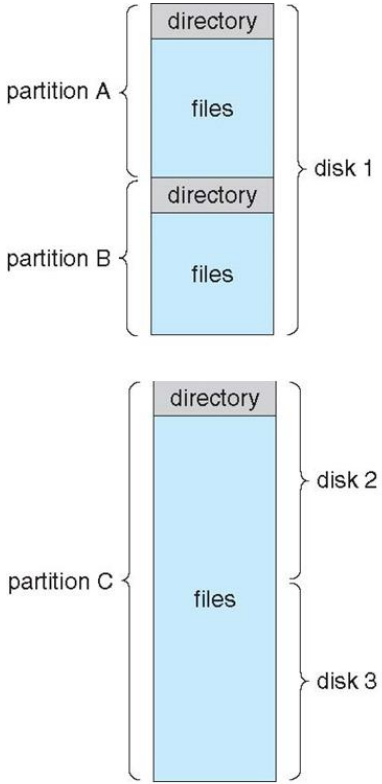
# Directory Structure

- A collection of nodes containing information about all files
- Both the directory structure and the files reside on disk



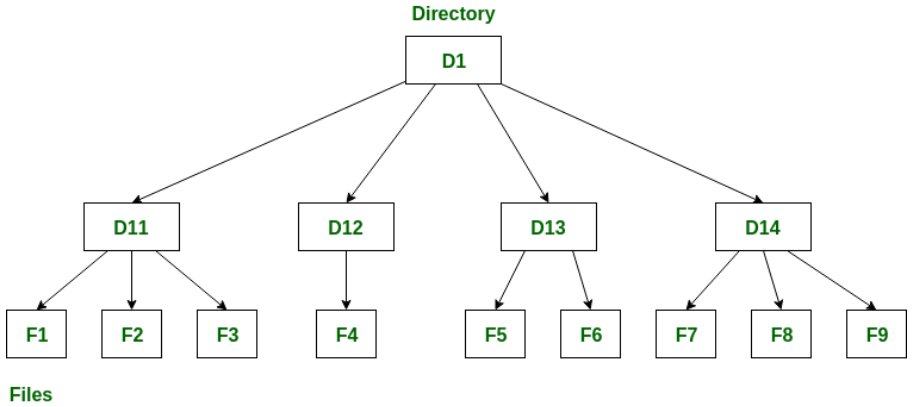
# Disk Structure

- 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
- Partitions also known as minidisks, slices
- Each partition contains a file system known as a **volume** that tracks that file system's info in **device directory** or **volume table of contents**



# Operations Performed on Directory

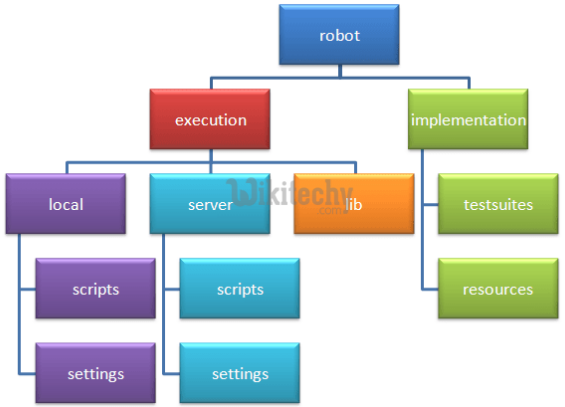
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system



# Directory Organization

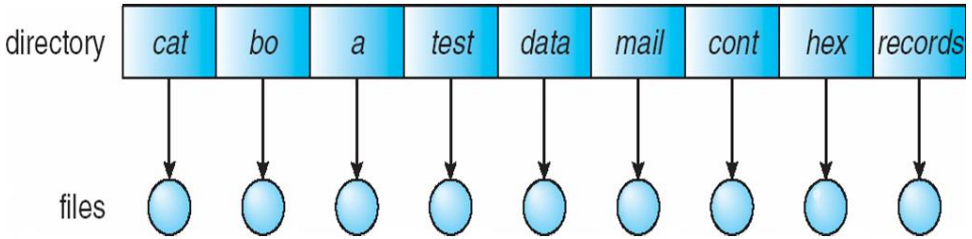
- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties,

  (e.g., all programs, all games, …)



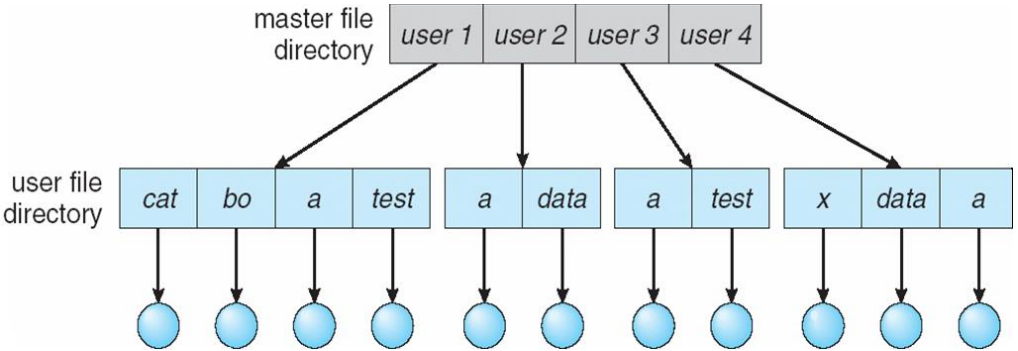
# Single-Level Directory

- A single directory for all users
- Naming problem
- Grouping problem

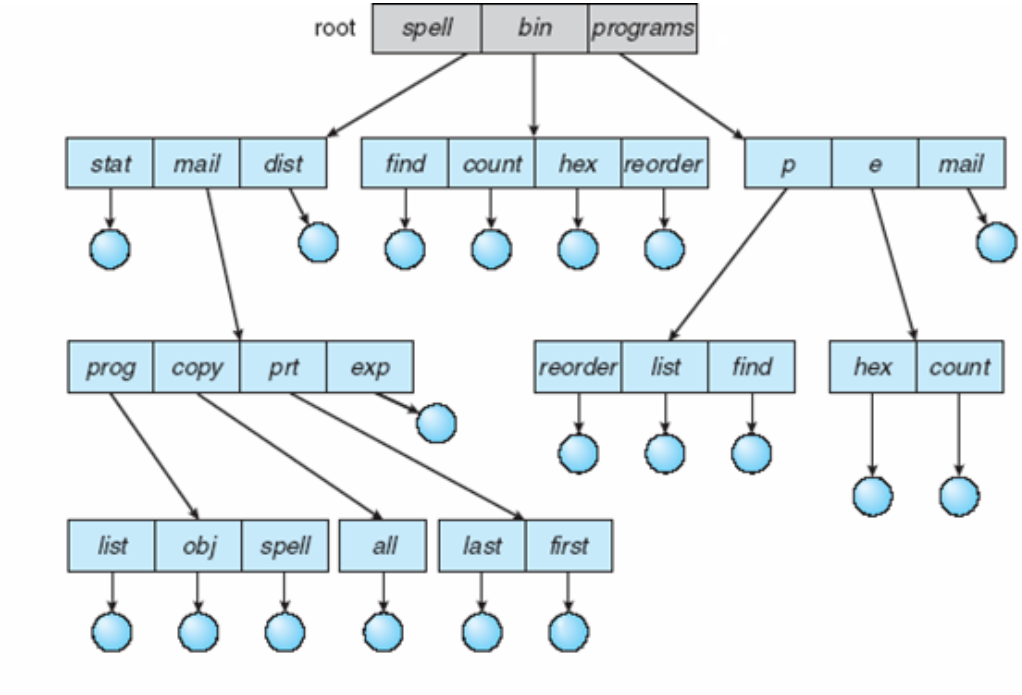


# Two-Level Directory

- Separate directory for each user
- Path name
- Can have the same file name for different user

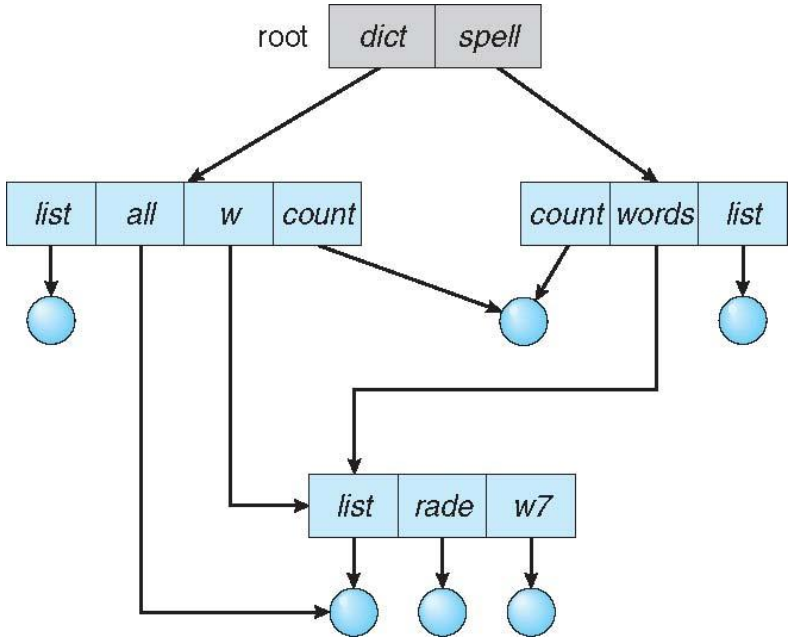


# Tree-Structured Directories

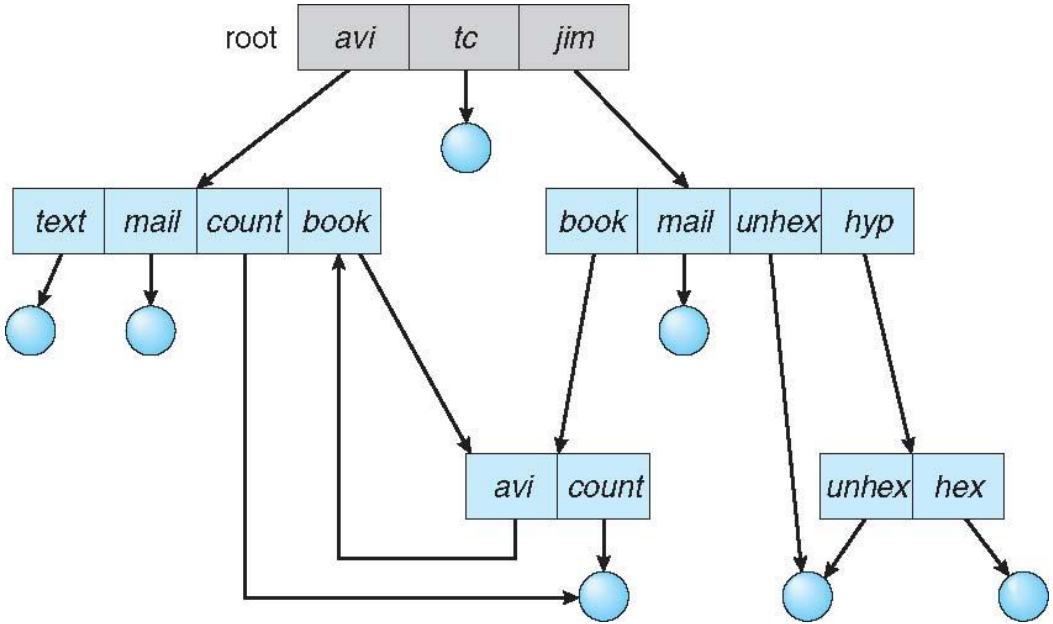


# Acyclic-Graph Directories

- Have shared subdirectories and files



# General Graph Directory




**johnjose@iitg.ac.in**
**http://www.iitg.ac.in/johnjose/**

# CS343 - Operating Systems

## Module-6B

## File System Implementation


**Dr. John Jose**

**Assistant Professor**

**Department of Computer Science & Engineering**

**Indian Institute of Technology Guwahati, Assam.**

http://www.iitg.ac.in/johnjose/

# File-System Implementation

- File-System Mounting
- File Sharing & Protection
- File-System Structure
- File-System Implementation
- Directory Implementation

# Objectives

- To explore file-system protection and security features
- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems

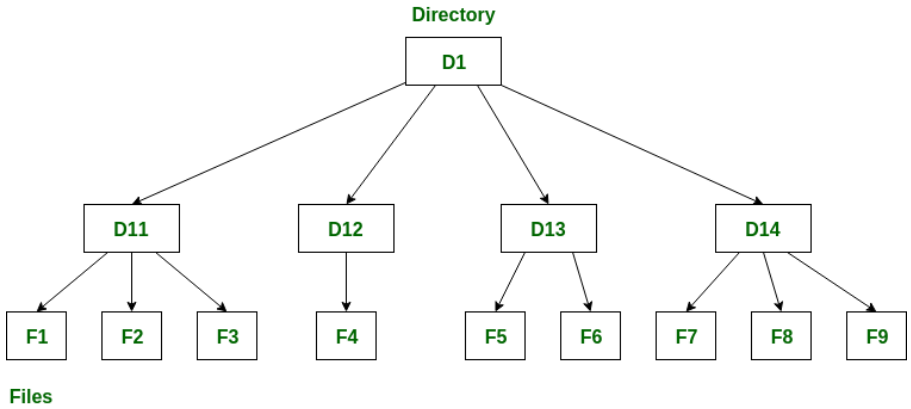
# File Concept

- File – Logical Storage Unit
- Types: Data files (numeric, character, binary) & Program files
- Operations: Create, Write, Read, Seek, Delete, Truncate, Open, Close

| 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 |

# Directory Structure

- A collection of nodes containing information about all files
- **Basic operations on directory**
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

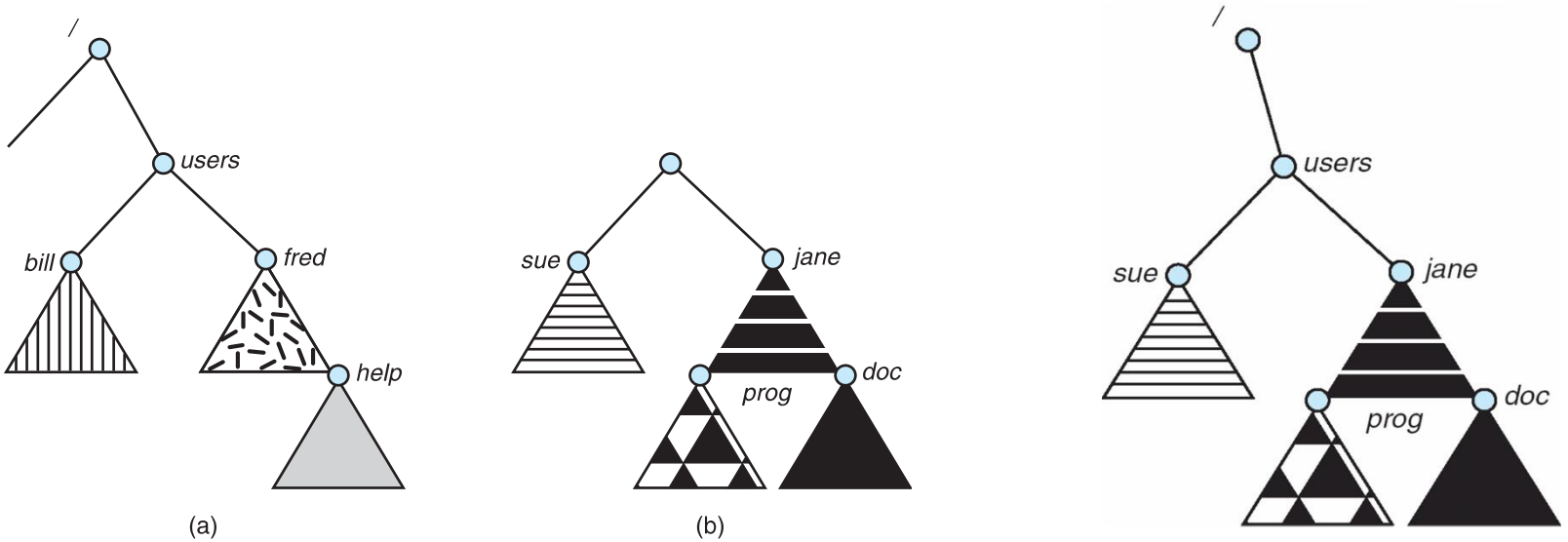


# File System Mounting

- **Mounting** is a process by which the OS makes files and directories on a storage device available for users to access via the file system.
- OS acquires access to the storage medium; recognize, read and process file system structure and metadata on it.
- The location in file system that the newly-mounted medium was registered is called **mount point**.
- When the mounting process is completed, the user can access files and directories on the medium from the mount point.

# File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**



# 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

# File Sharing – Remote File Systems

- Standard operating system file calls are translated into remote calls
- **NFS** is standard UNIX client-server file sharing protocol
- **CIFS** is standard Windows protocol
- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

# File Sharing – Failure Modes

- All file systems have failure modes
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols include all information in each request, allowing easy recovery, but less security

# 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

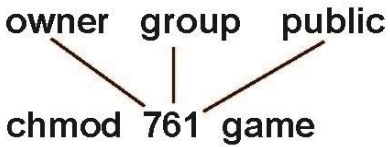
- Mode of access:  read, write, execute
- Three classes of users on Unix / Linux  [owner, group, public]
- Manager creates a group (G) and add some users to the group.

RWX

a) **owner access** 7 $\Rightarrow$ 1 1 1

b) **group access** 6 $\Rightarrow$ 1 1 0

c) **public access** 1 $\Rightarrow$ 0 0 1

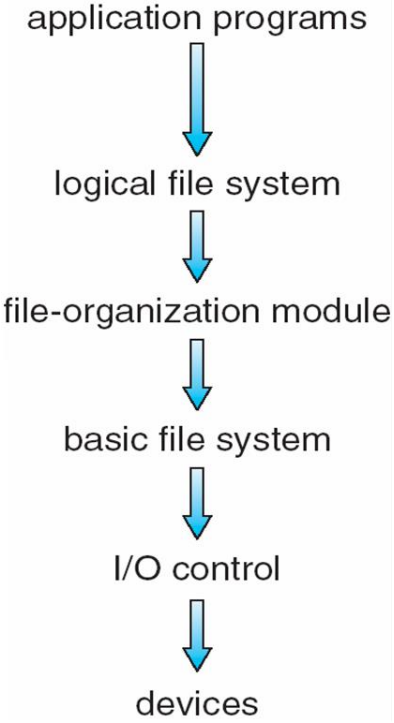


# File-System Structure

- File is the Logical storage unit
- **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 physical space for files.
- I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure that has information about a file

# Layered File System

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks
  - Directory management & Protection
- **File organization module** understands files, logical address, physical blocks - Translates logical block # to physical block #

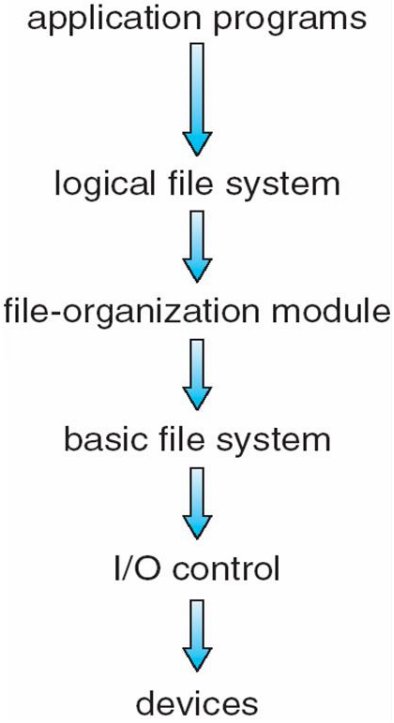


# Layered File System

- **Basic file system** issue generic commands to appropriate device driver

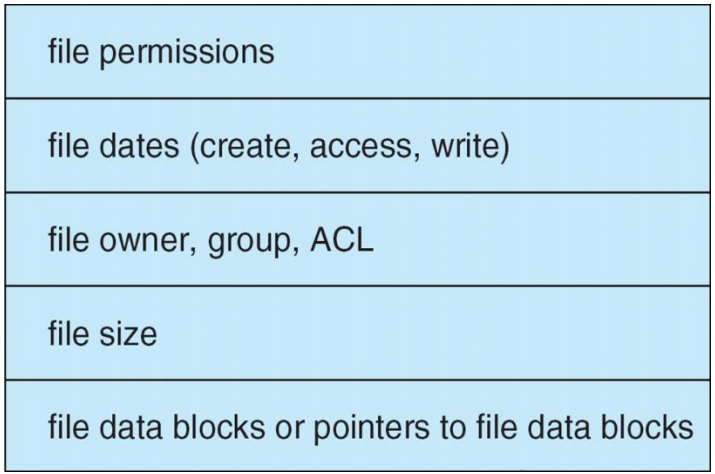
  Eg: *read drive1, cylinder 72, track 2, sector 10, into memory location 1060*

- **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



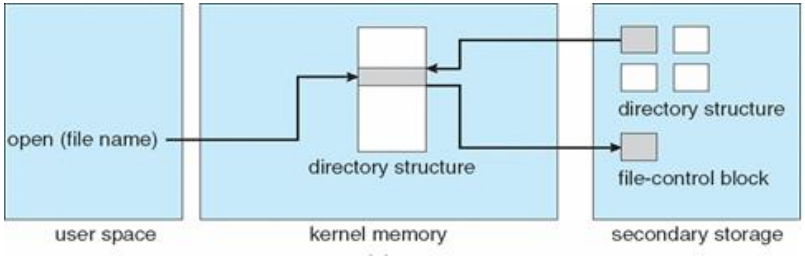
# File-System Implementation

- **File Control Block** contains many details about the file
  - inode number, permissions, size, dates

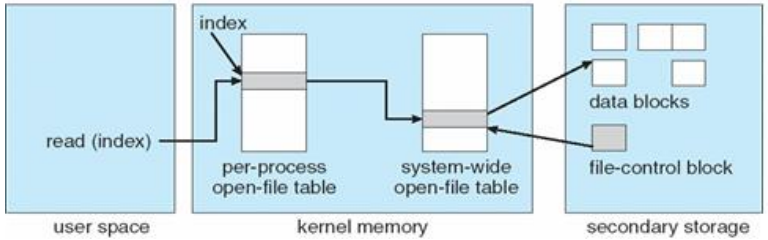


# In-Memory File System Structures

- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address




opening a file




reading a file

# File-System Implementation

- **Boot control block** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

# Partitions and Mounting

- Partition can be a volume containing a file system or just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-os booting

# Partitions and Mounting

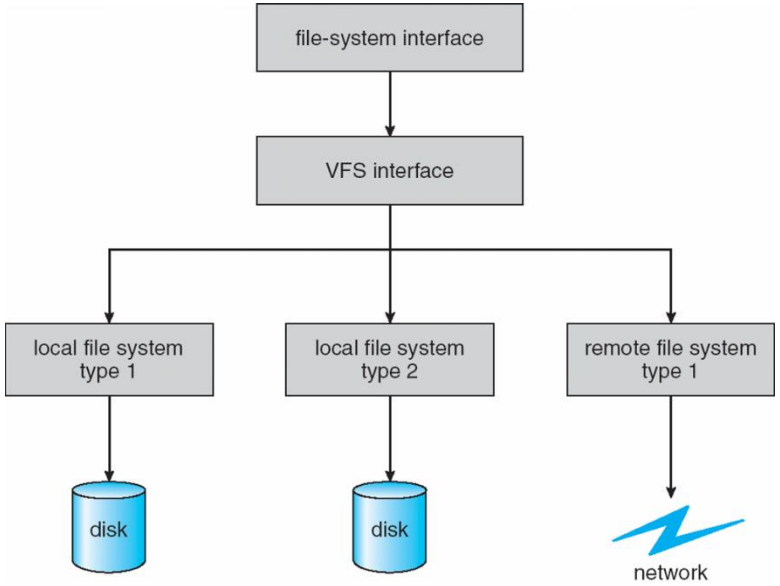
- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access

# Virtual File Systems

- **Virtual File Systems** (**VFS**) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
  - Then dispatches operation to appropriate file system implementation routines

# Virtual File Systems

- The API is to the VFS interface, rather than any specific type of file system



# Virtual File Systems

- For example, Linux has four object types:
  - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
  - **`int open(. . .)`**—Open a file
  - **`int close(. . .)`**—Close an already-open file
  - **`ssize t read(. . .)`**—Read from a file
  - **`ssize t write(. . .)`**—Write to a file
  - **`int mmap(. . .)`**—Memory-map a file

# 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


**johnjose@iitg.ac.in**
**http://www.iitg.ac.in/johnjose/**

# CS343 - Operating Systems

## Module-6C

## File Allocation and Free Space Management


**Dr. John Jose**

**Assistant Professor**

**Department of Computer Science & Engineering**

**Indian Institute of Technology Guwahati, Assam.**

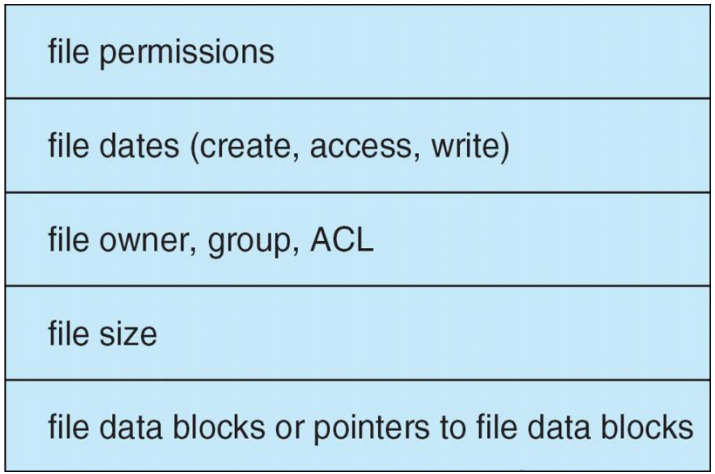
http://www.iitg.ac.in/johnjose/

# File-System Implementation

- File Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

# File-System Implementation

- **File Control Block** contains many details about the file
  - inode number, permissions, size, dates

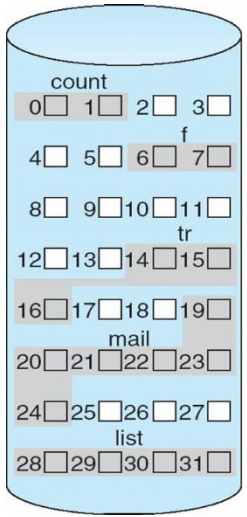


# Contiguous Allocation

- 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

- Mapping from logical to physical
  - Block to be accessed = starting address
  - Displacement into block = length




directory

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

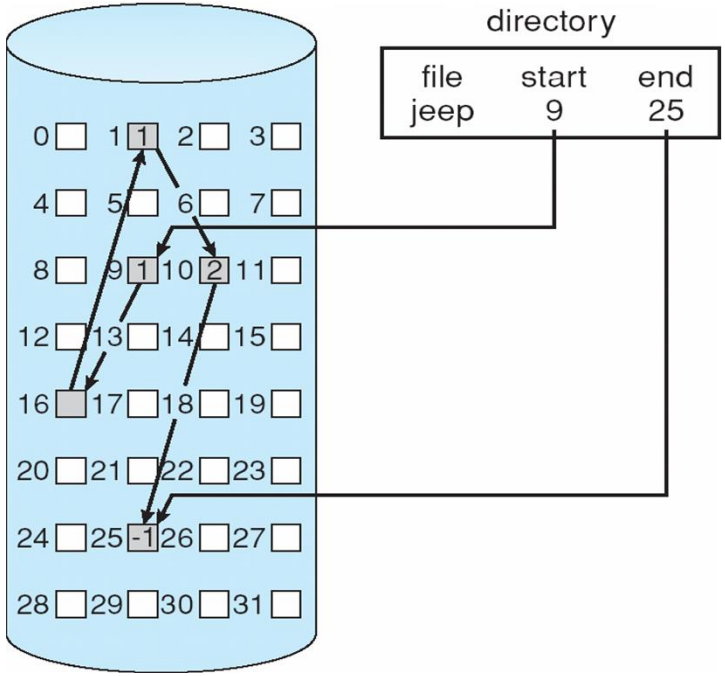
# Extent-Based Systems

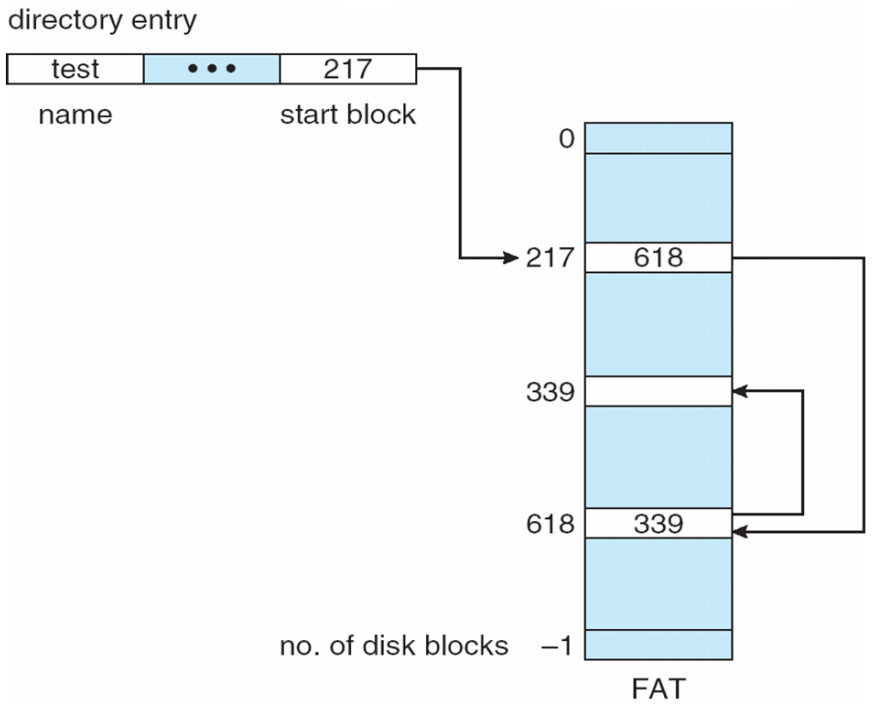
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

# Linked Allocation

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

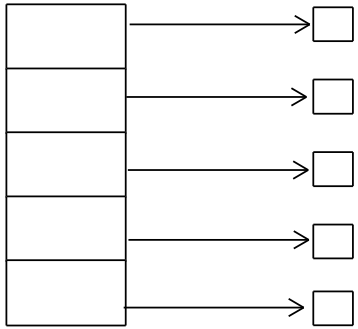
# Linked Allocation



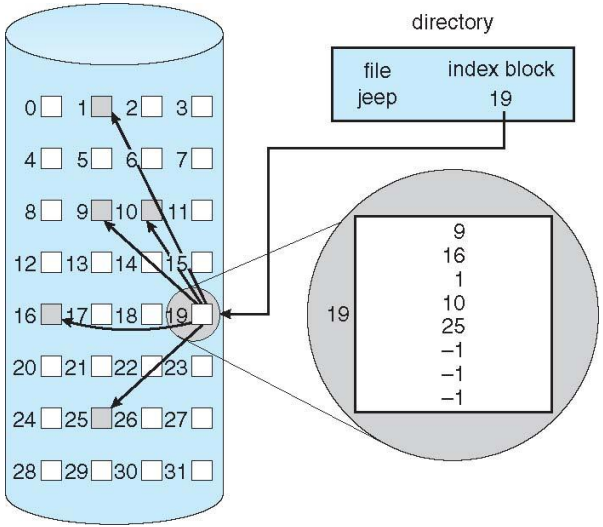



# Indexed Allocation

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



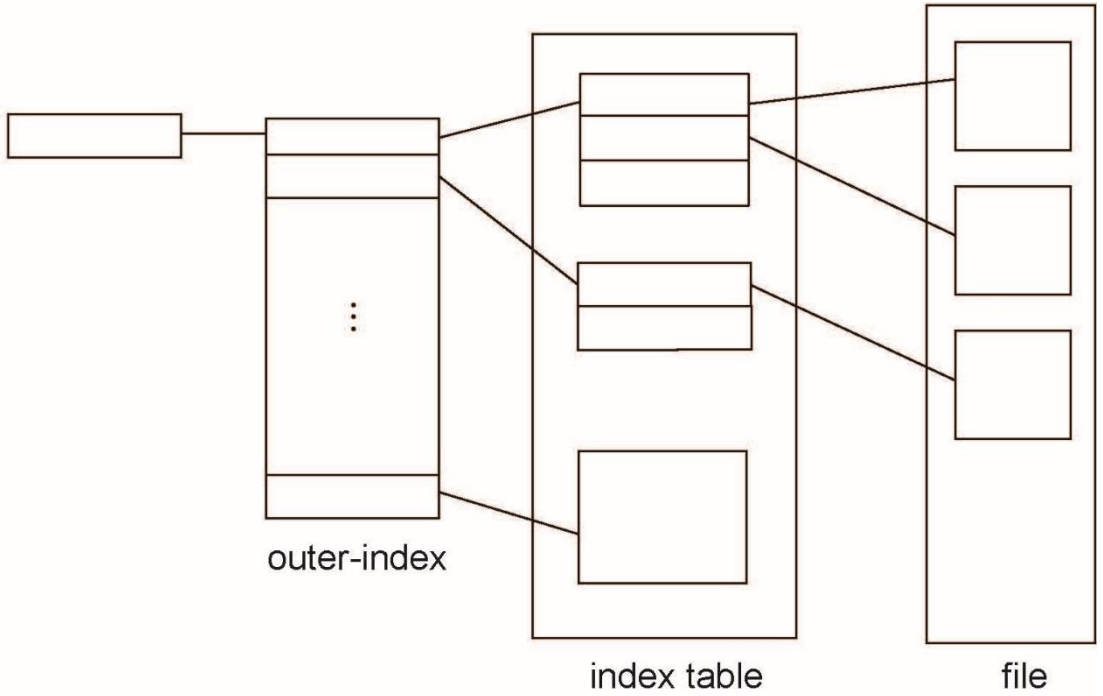
index table



# Indexed Allocation

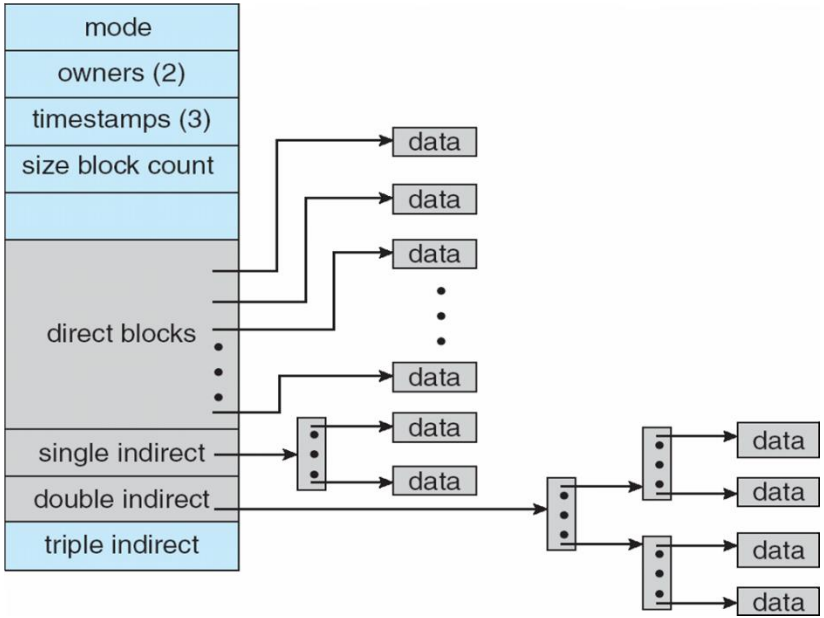
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Single Level and Multilevel Index for small and large files

# Indexed Allocation



# UNIX UFS

4K bytes per block, 32-bit addresses



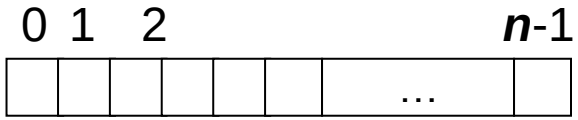

More index blocks than can be addressed with 32-bit file pointer

# 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 – multiple index block reads.

# Free-Space Management

- File system maintains **free-space list** to track available blocks
- **Bit vector** or **bit map** (***n*** blocks)




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

# Free-Space Management

- Bit map requires extra space
  - Example:

    block size = 4KB = $2^{12}$ bytes

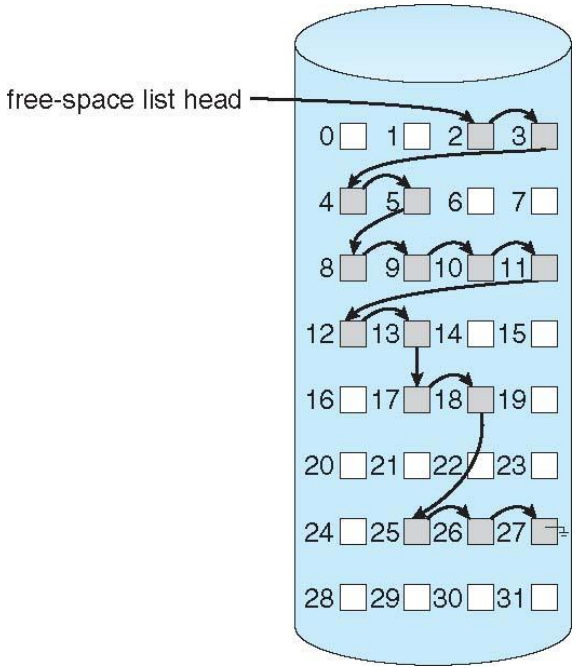
    disk size = $2^{40}$ bytes (1 terabyte)

    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

- Easy to get contiguous files

# Linked Free Space List on Disk

- **Linked list (free list)**
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



# Free-Space Management

- **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

# Free-Space Management

## Space Maps

- Divides device space into **metaslab** units and manages metaslabs
  - Given volume can contain hundreds of metaslabs
- Each metaslab has associated space map -uses counting algorithm

# Efficiency and Performance

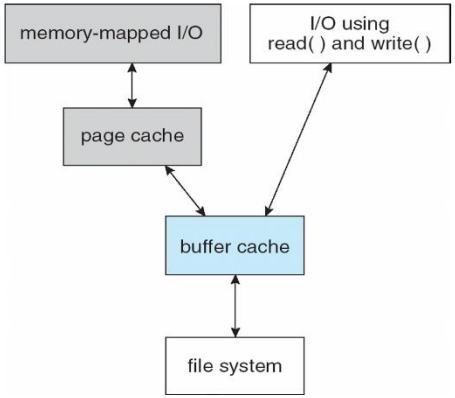
- Efficiency dependents on:
  - Disk allocation and directory algorithms
  - Types of data kept in file’s directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

# Efficiency and Performance

- Performance
  - Keeping data and metadata close together
  - **Buffer cache** – separate section of main memory for frequently used blocks
  - **Synchronous** writes sometimes requested by apps or needed by OS
    - No buffering / caching - writes done on disk directly
    - **Asynchronous** writes more common, buffered, faster

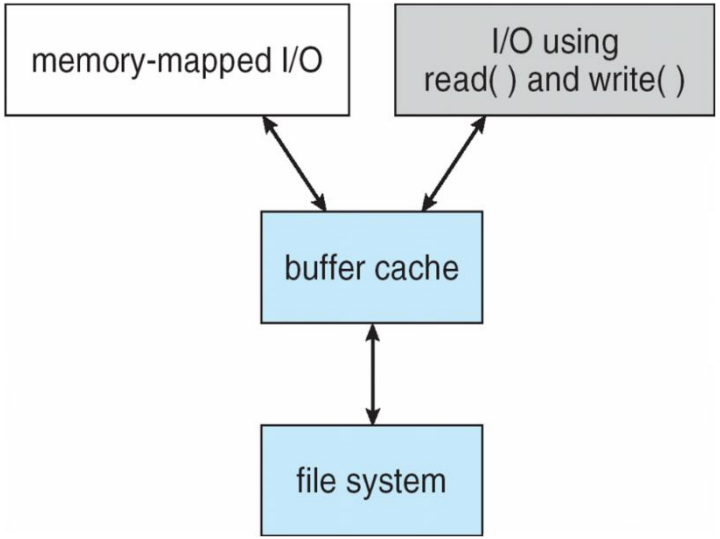
# Page Cache and Buffer Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache



# Unified Buffer Cache

- A **unified buffer cache** uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid **double caching**



# 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

# Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each metadata update to the file system as a **transaction**
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log
  - The transactions in the log are asynchronously written to the file system structures
- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata


**johnjose@iitg.ac.in**
**http://www.iitg.ac.in/johnjose/**