OPERATING SYSTEMS (THEORY)

LECTURE - 11

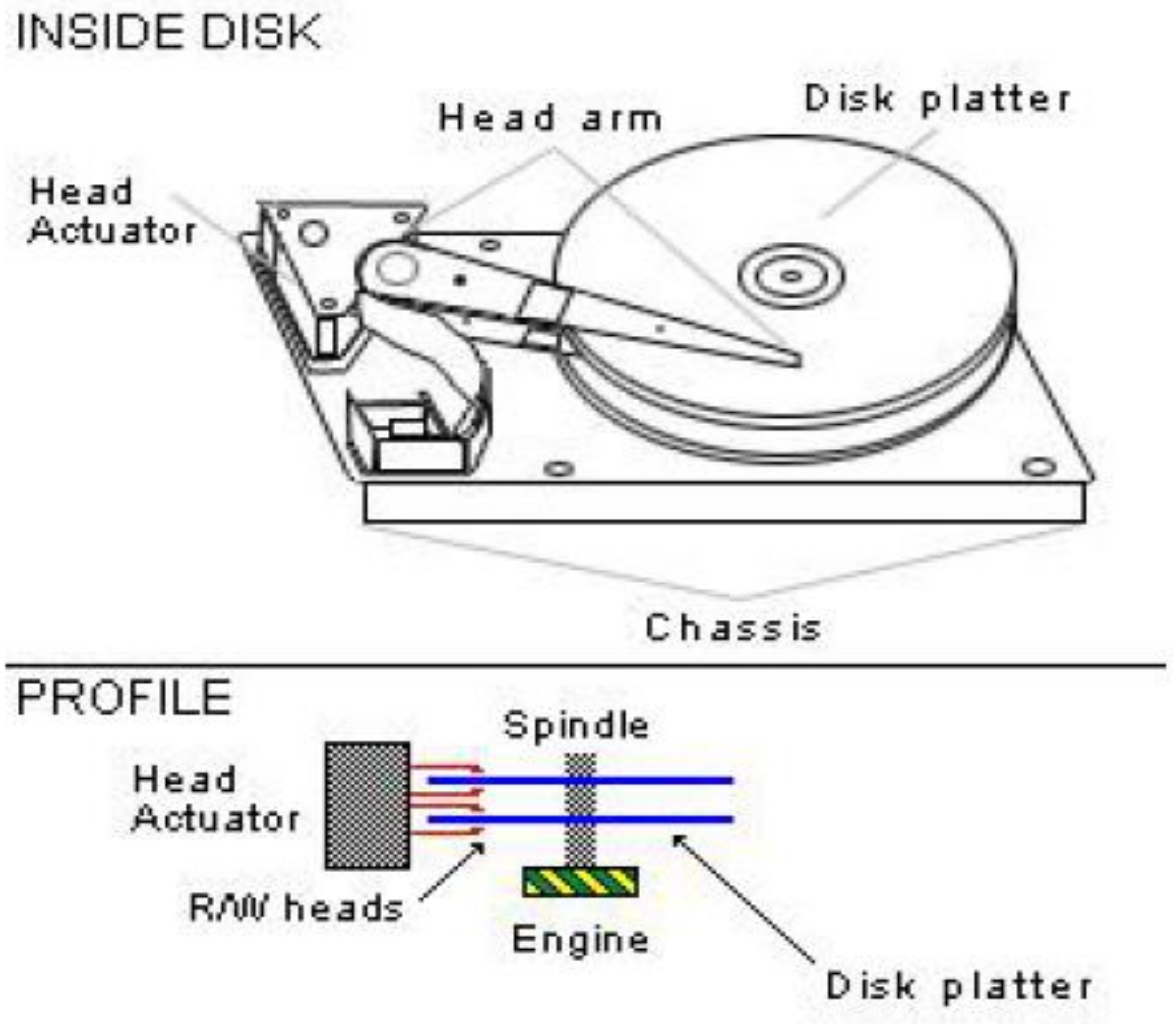
K.ARIVUSELVAN

Assistant Professor (Senior) – (SITE)

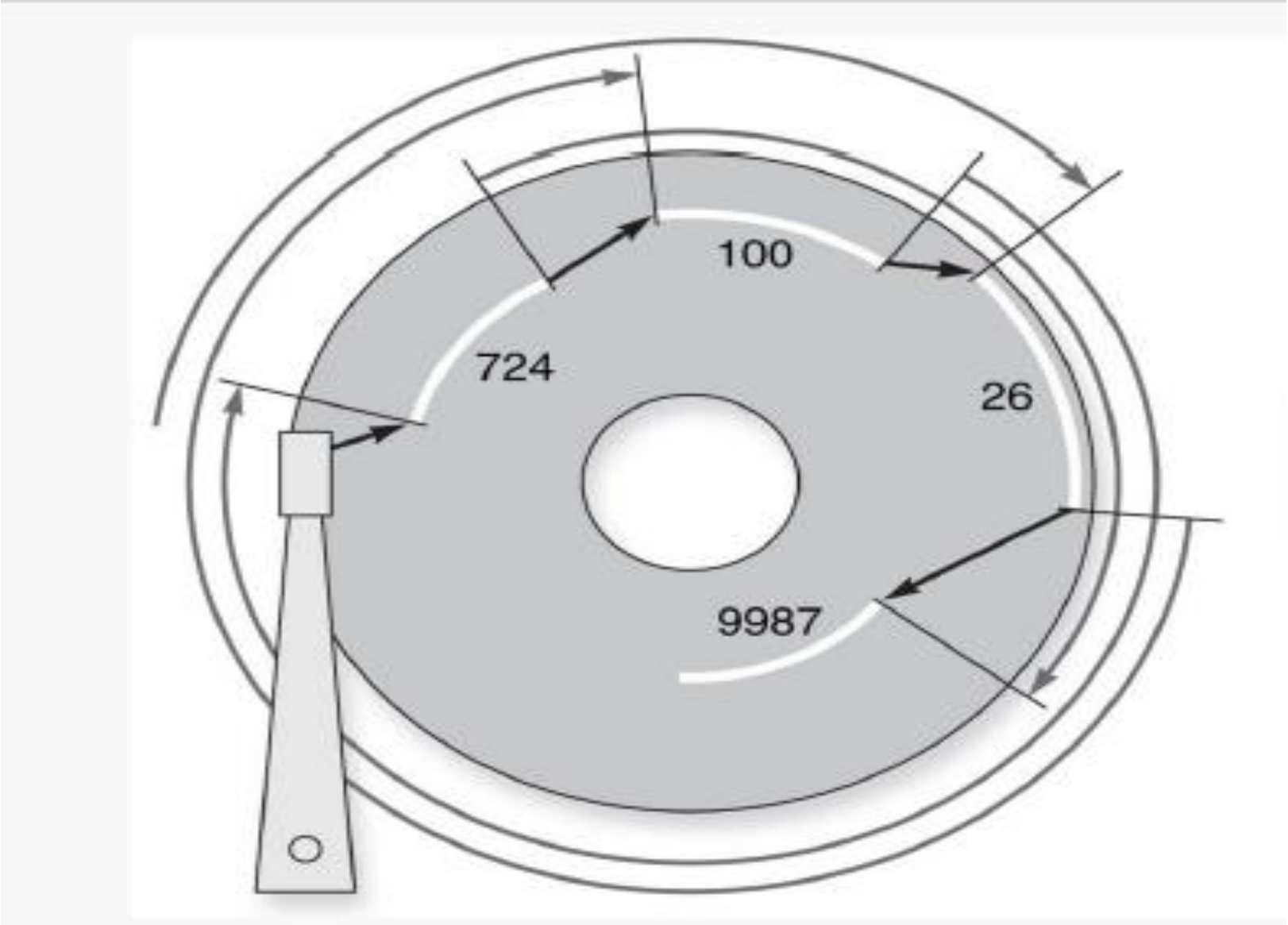
VIT University

FILE SYSTEMS

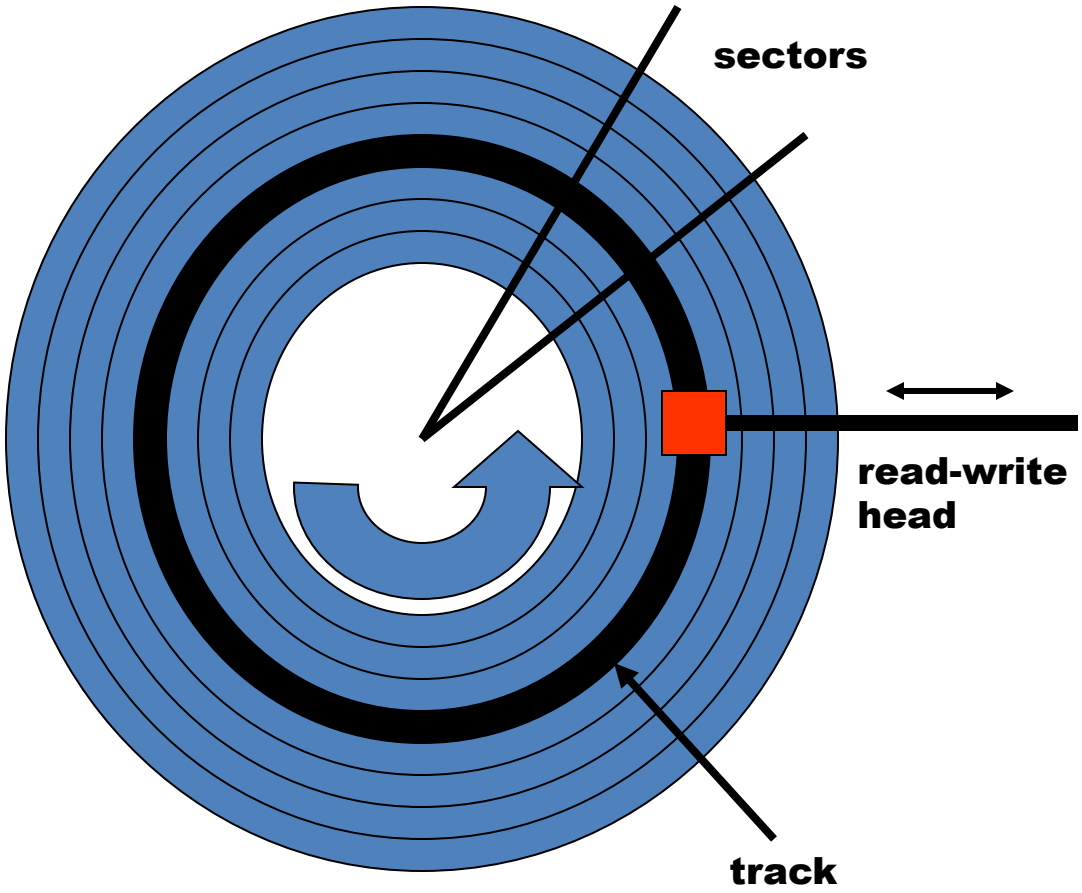
Physical Disk Organization



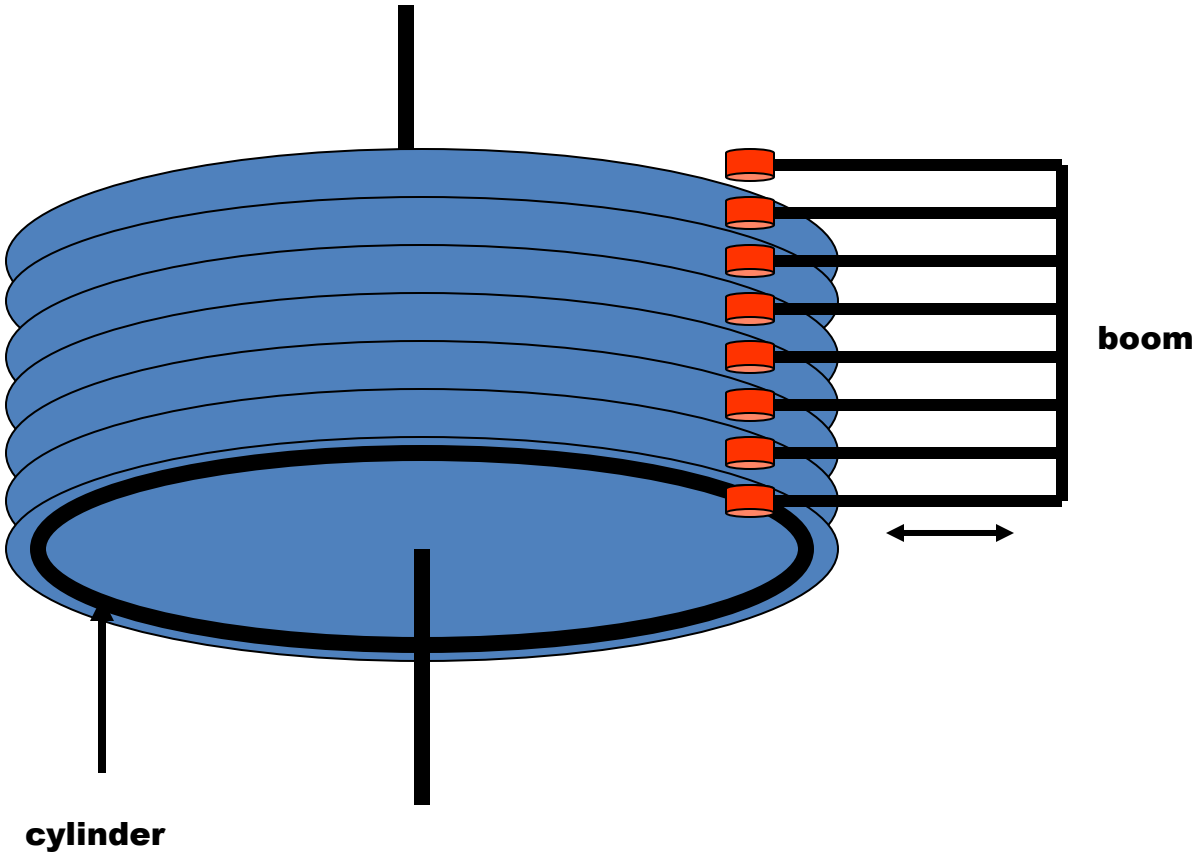
Physical Disk Organization



Physical Disk Organization



Physical Disk Organization



I/O management

(1) Programmed I/O:

- Data transfers initiated by a CPU under driver software control to access registers or memory on a device.
- The CPU issues a command then waits for I/O operations to be complete.
- As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time

I/O management

(2) Interrupt-driven I/O

- The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

(3) Direct memory access (DMA)

- CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.

I/O buffering

- Can be **block-oriented** or **stream-oriented**

(1) Block-oriented buffering:

- Information is stored in fixed-size blocks
- Transfers are made a block at a time
- Used for disks and tapes

I/O buffering

(2) Stream-oriented:

- Transfer information as a **stream of bytes**
- Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage

Important Terms

(1) Seek time is the time it takes to position the head on the desired track

(2) Rotational delay or **rotational latency** is the additional time it takes for the beginning of the sector to reach the head once the head is in position

(3) Transfer time is the time for the sector to pass under the head

File-System

File-System Structure

- **Two design problems:**
  - How should the **file system look to the user?**
  - How are **logical files mapped onto** the **physical storage devices**?

File Systems—Examples

	Year	Max File Size	Max Volume Size	Primary OS or use
FAT32	1996	4 GB	2 – 16 TB	Windows
NTFS 3.1	2001	16 TB	16 EB	Windows
ext3	1999	16 GB – 2 TB	2 – 16 TB	UNIX
ext4	2006	16 TB	1 EB	UNIX
HFS Plus	1998	8 EB	8 EB	Mac OS
ZFS	2004	16 EB	4 GB – 8 TB	Sun

Layers of File System Abstraction

Application Programs

Logical File System: manages metadata (all of the file-system structure except the actual contents of files) and directory structure

File-Organization Module: translates logical addresses to physical addresses for the basic file system

Basic File System: able to issue generic commands to device drivers, manages memory buffers and caches

I/O Control: device drivers and interrupt handlers to transfer information between main memory and disk system

Storage Devices

File-System Implementation

- **Implementing a file system requires various memory structures.**
- **On disk, a file system contains:**
  - **Boot control block** contains **info** needed by system to **boot OS** from that volume
  - **Volume control block** contains **volume details**, such as **number of blocks**, **size of blocks**, and **count of free blocks**
  - **Directory structure** organizes the files
  - **Per-file File Control Block (FCB)** contains many details about the file; known as an **inode** on most UNIX systems

File-System Implementation

- **In - memory** data structures

  – **Directory cache** holds information about **recently-accessed** directories

  – **System-wide open file table** contains a **copy of the FCB** for each **open file**

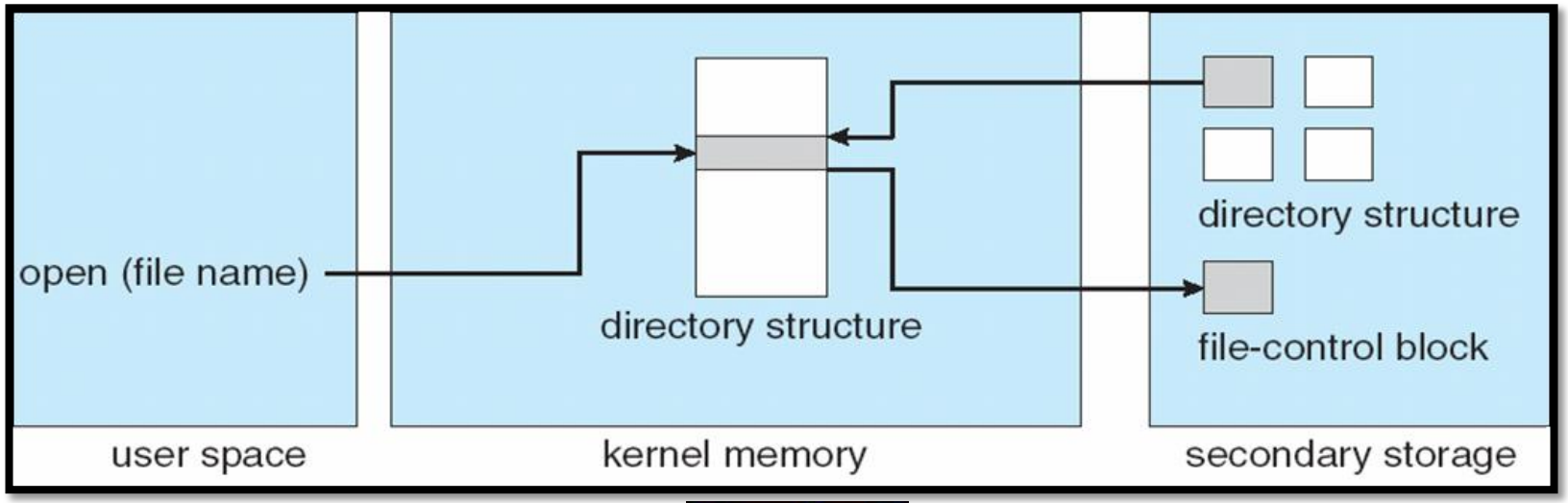
  – **Per-process open-file table** contains a **pointer to** the **entry in** the **system-wide open file table** for each file the process has **open**

File Open

- A file must be opened via the **`open()`** system call.

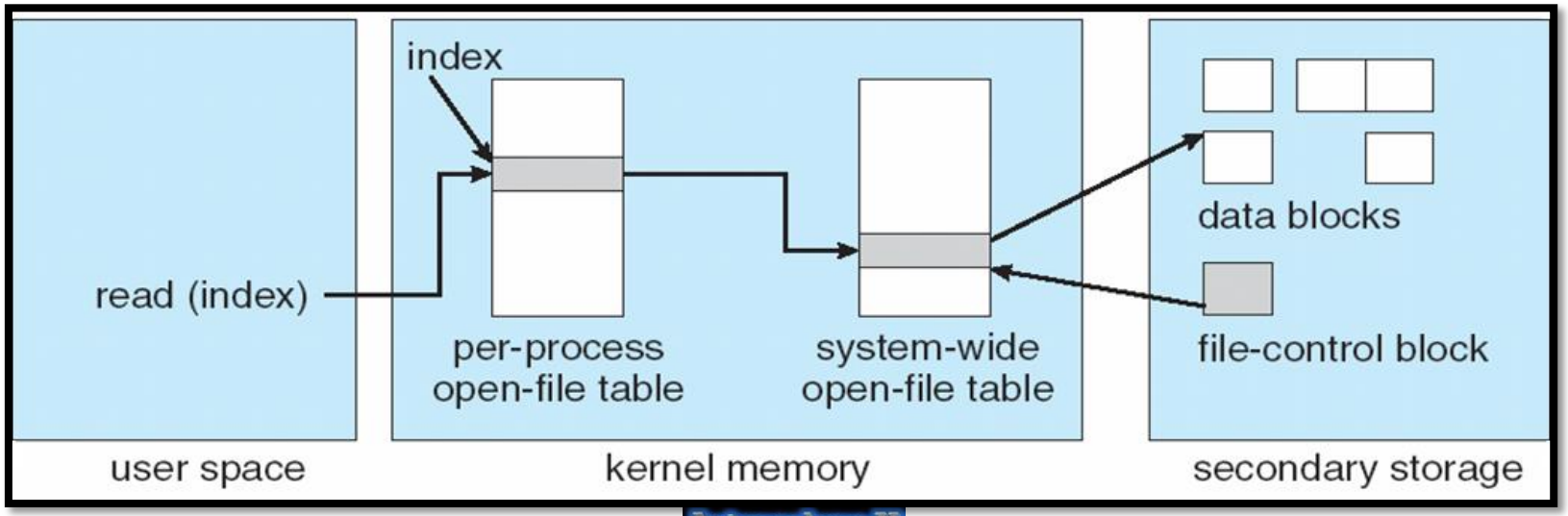
– If the file is **already in** the **system-wide open file table**, an entry is created in the **per-process open file table**

-- Otherwise, the directory is **searched** for the given file, and the **FCB is copied** into the **system-wide open file table** and an entry is created in the **per-process open file table**



File Close

- A file is **closed** via the `close()` system call.
  - The entry in the **per-process open file table** is **removed.**
  - The **open count** in the **system-wide entry** is decremented.
  - If all users have **closed** the file, then the entry in the **system-wide open file table** is removed.



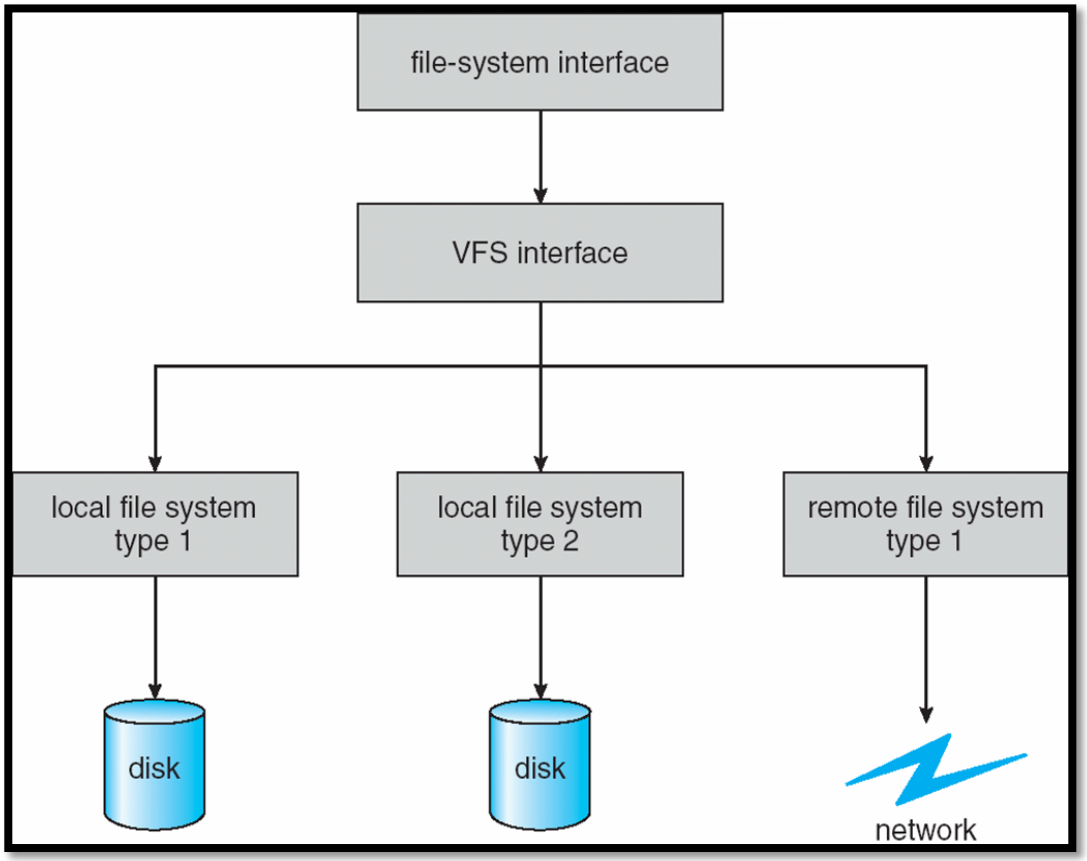
Virtual File Systems

- How does an **operating system** allow **multiple types of file systems** to be **integrated** into a **directory structure?**

- **Virtual file system (VFS)**

  - VFS allows the same system call interface (the API) to be used for **different types of file systems**.

  - VFS allows files to be **uniquely represented** throughout a network.



File Structure

- **Simple record structure:**
  - **Lines**
  - **Fixed length**
  - **Variable length**

- **Complex Structures:**
  - **Formatted document**
  - **Re-locatable load file**

•Can simulate last two with first method by inserting appropriate control characters.

Basic 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.

More file attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Common File Operations

- **Create**
- **Read/Write**
- **Seek: Reposition within file.**
- **Delete**

- **Append/Truncate**
- **Set/Get Attributes**

- **Open(*Fi*):** search the directory structure on disk for entry *Fi*, and move the content of entry to memory.

- **Close (*Fi*):** move the content of entry *Fi* in memory to directory structure on disk.

Access Methods

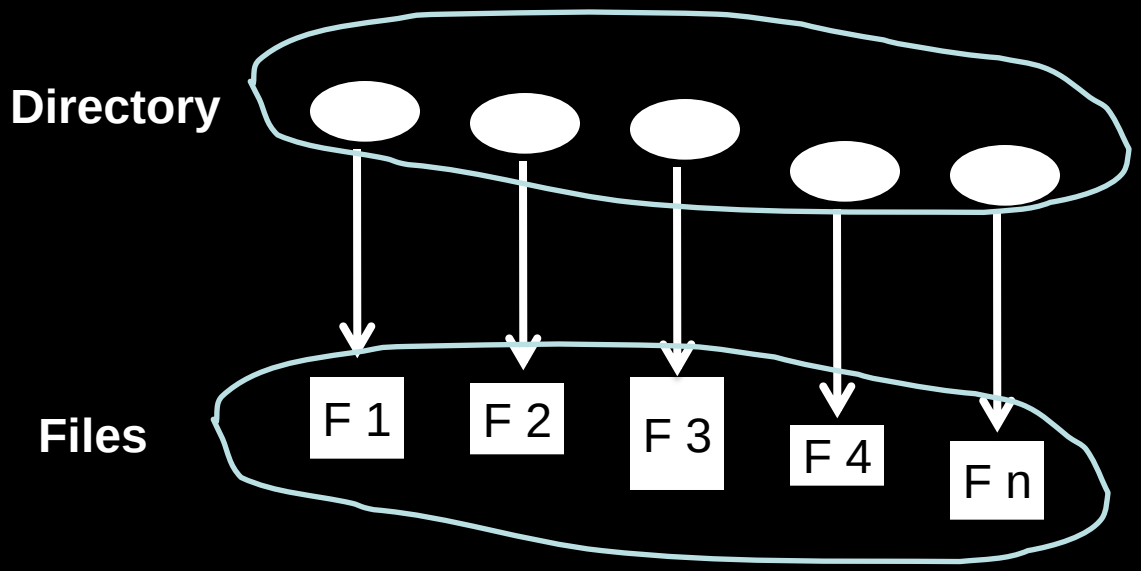
- **Sequential Access**
  - *read next*
  - *write next*
  - *reset*
  - no *read after last write*
    - *(rewrite)*

- **Direct Access**
  - *read n*
  - *write n*
  - *position to n*
    - *read next*
    - *write next*
  - *rewrite n*

n = relative block number

Directory Structures

- Collection of nodes containing information about all files.



- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.

Information in a Directory Entry

- Name
- Type
- Address
- Current length
- Maximum length

- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information

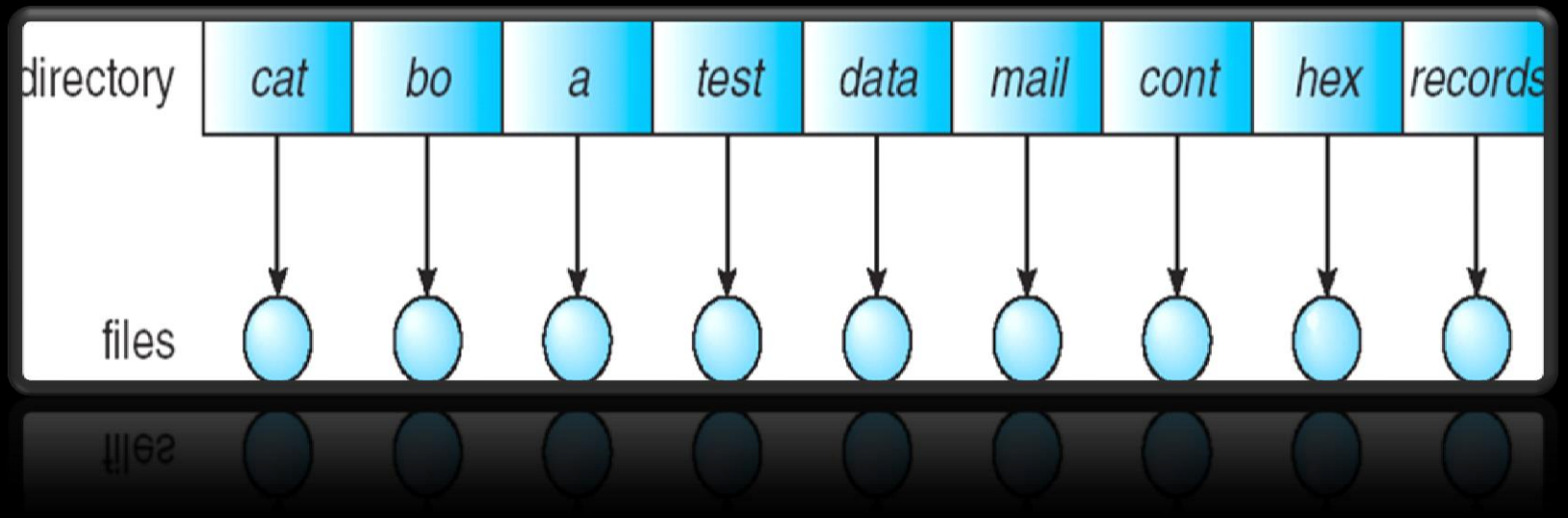
Directory Operations

- Search for a file
- Create a file
- Delete a file

- List a directory
- Rename a file
- Traverse the file system

Single-Level Directory

- A single directory for all users.



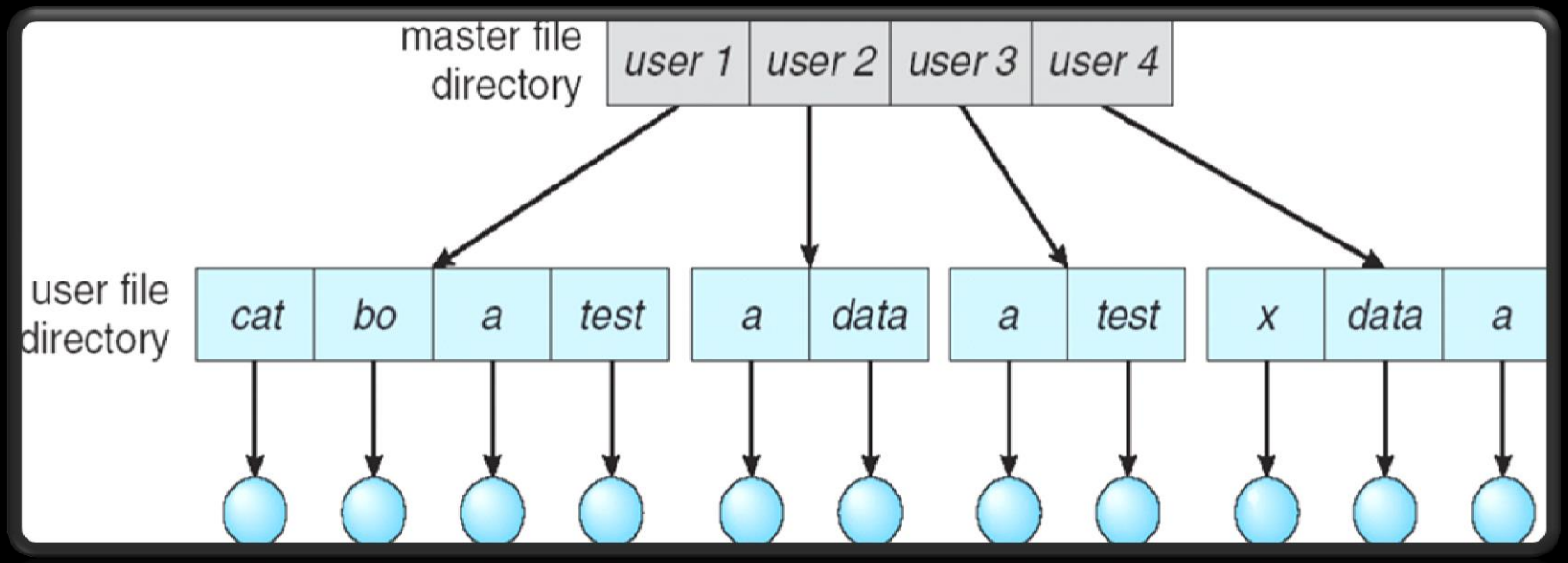
- **Common problems:**

  - Eventual length of directory.
  - Giving unique names to files.
  - Remembering names of files.

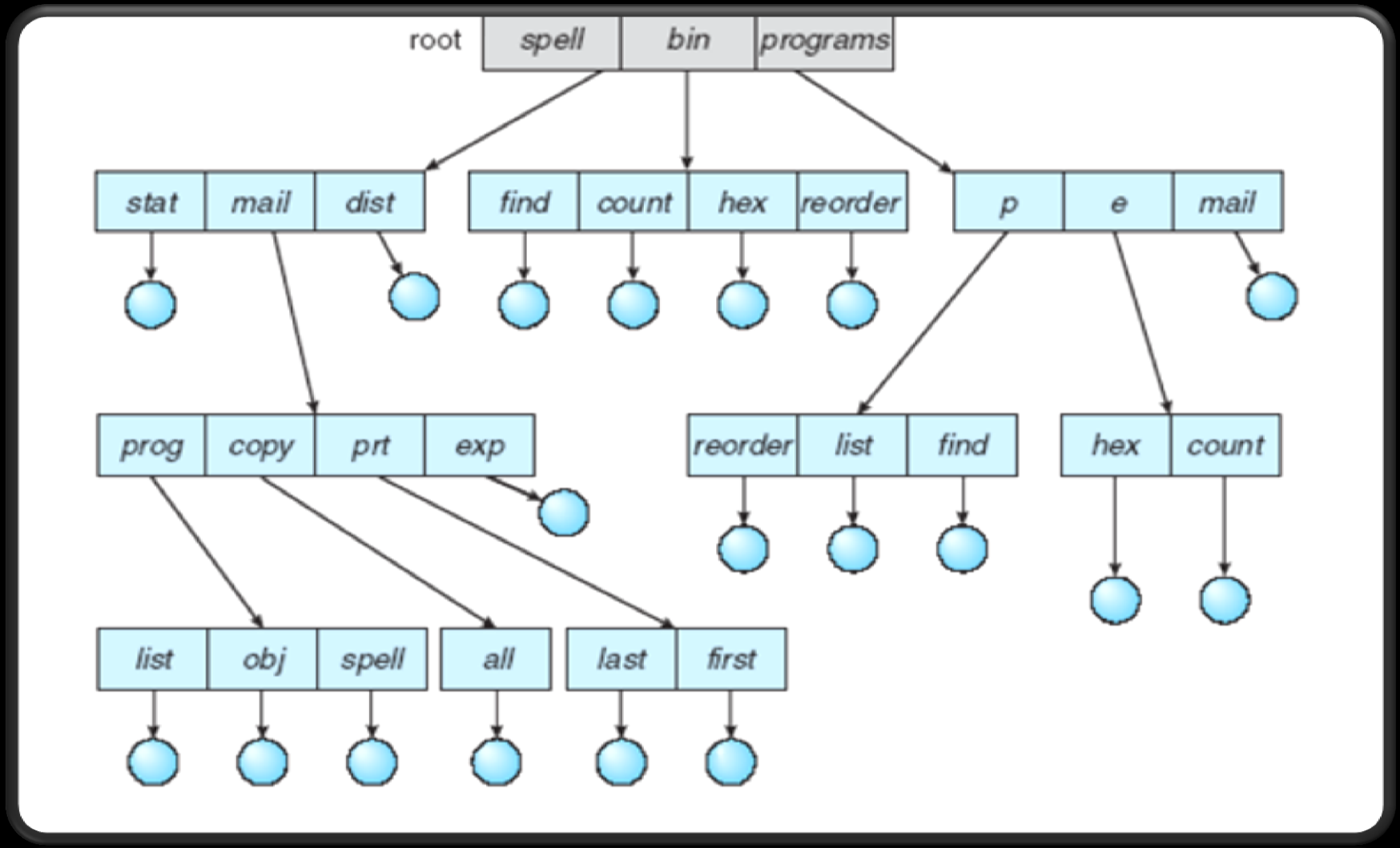
Two-Level Directory

- Separate directory for each user.
- Use of path name.

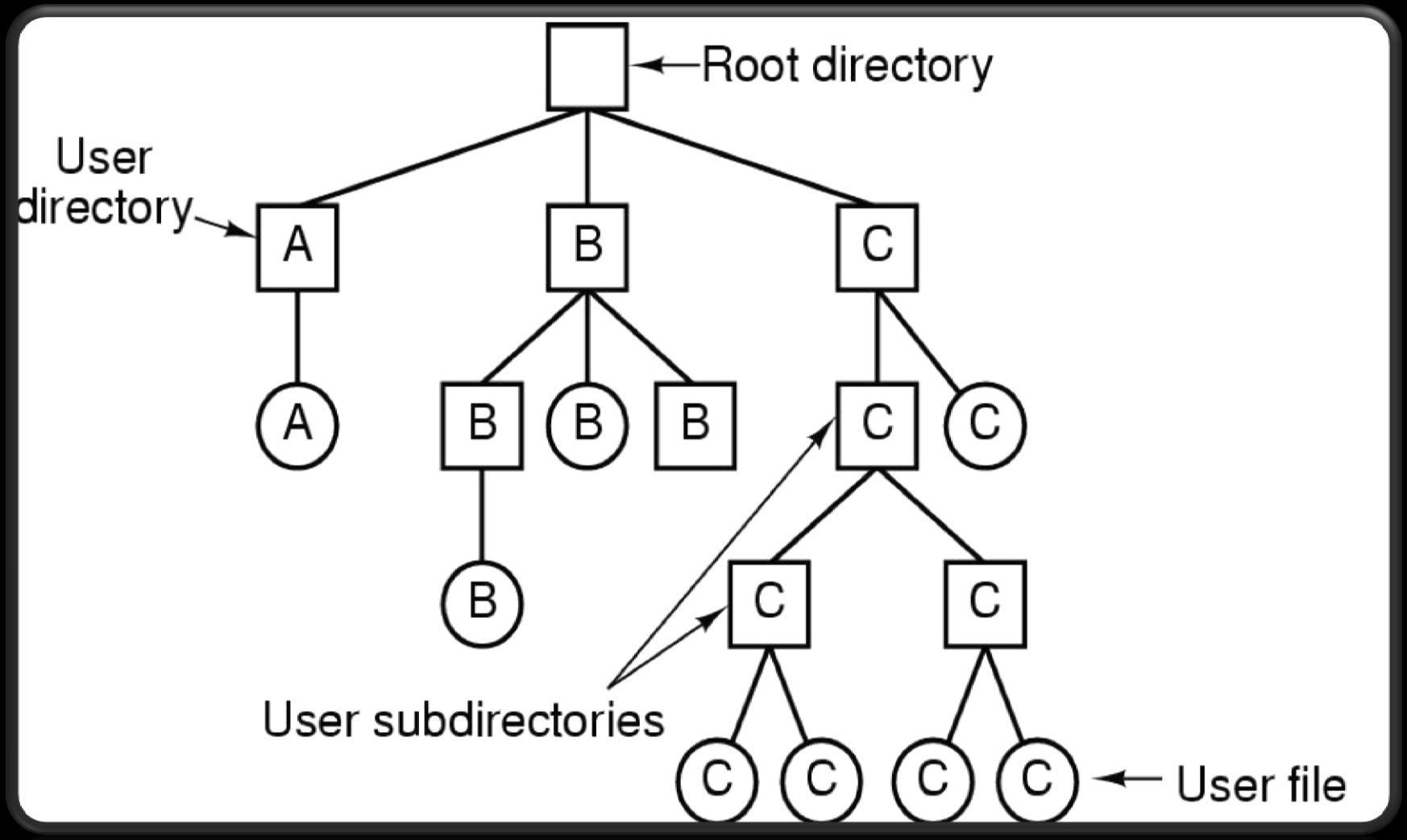
- Can have the same file name for different users.
- Provides efficient searching.



Tree-Structured Directory



Tree-Structured Directory Components

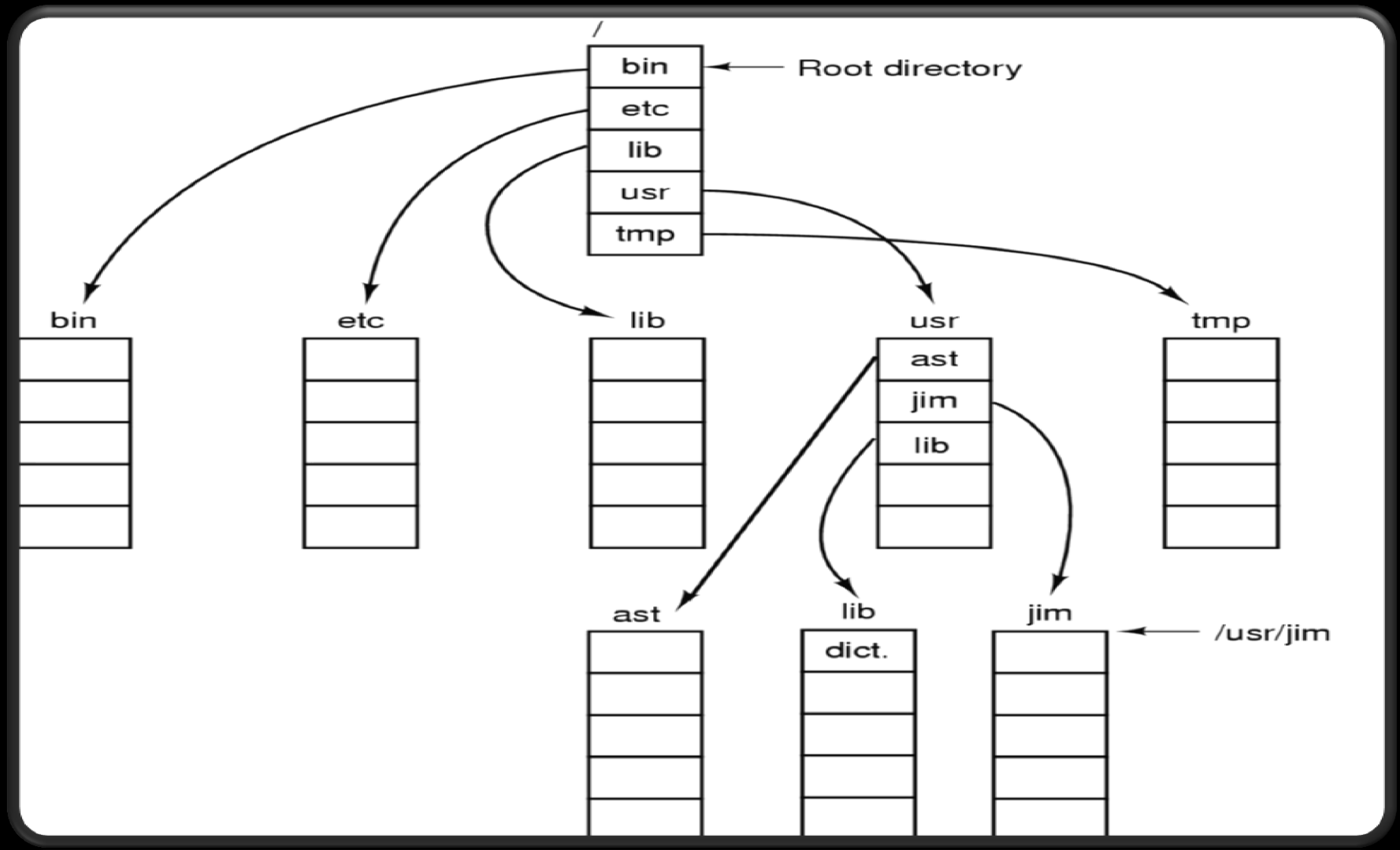


Tree-Structured Directories

- Provides efficient searching.
- Has grouping capability.

- Current directory (working directory):
  cd /spell/mail/prog
- Use absolute or relative path name.

Directory Path Names



Allocation Methods

- How do we **allocate space** for files on a **disk?**

- We desire:
  - To use **disk space efficiently**
  - To be able to **access files quickly**

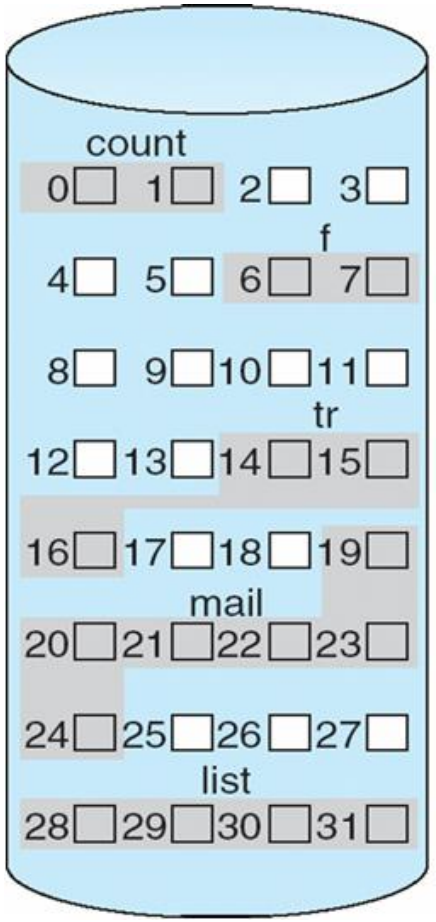
- **Three common methods:**
  1. **Contiguous allocation**
  2. **Linked allocation**
  3. **Indexed allocation**

Contiguous Allocation

- Each **file** occupies a set of **contiguous blocks** on the **disk**

- **Advantages**:
  - **Simple** to **read or write** a file
  - Provides **random access** within a file

- **Disadvantages**:
  - Must **find contiguous space** for each **new file**
  - **Dynamic storage-allocation problem:** use **best-fit** or **worst-fit** algorithms to fit files in blocks
  - **External fragmentation**
  - Files **cannot grow** easily



directory

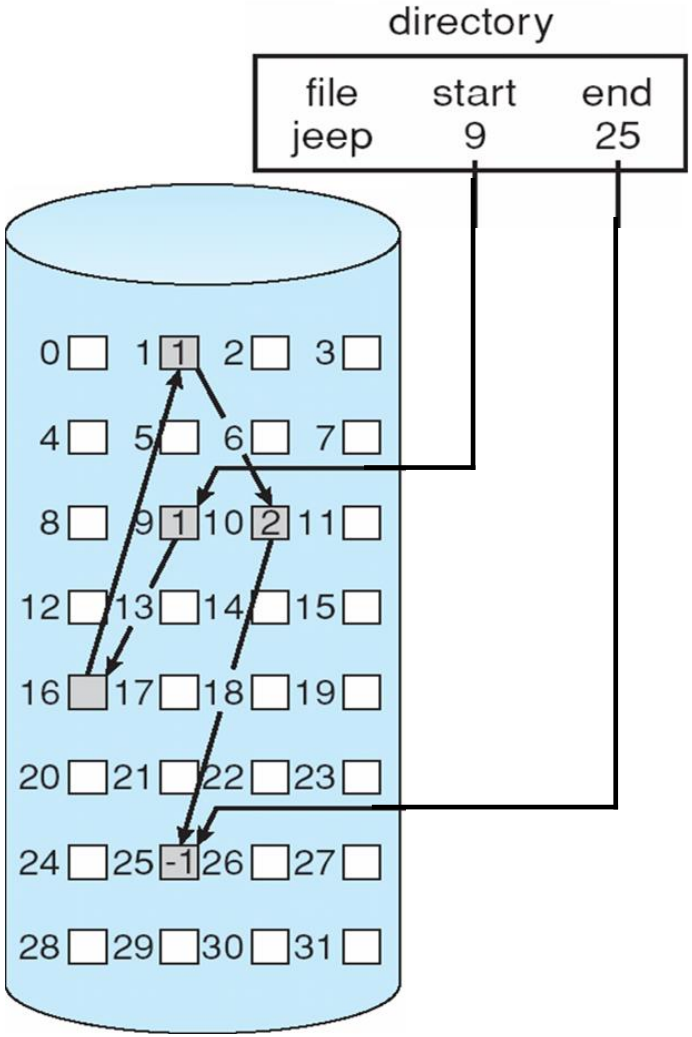
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation

- Each **file** is a **linked list** of **disk blocks**, which may be **scattered** on the **disk.**
- **Directory** contains a **pointer to** the **first** and **last** **blocks**, and
- **Each block** contains a **pointer to** the **next block.**

- **Advantages**:
  - **No external fragmentation**
  - **Easy** to **expand the size** of a **file**

- **Disadvantages**:
  - **Not suitable** for **random access** within a **file**
  - **Pointers take up** some **disk space**
  - **Difficult** to **recover a file** if a **pointer** is **lost or damaged**
  - -- **Greater internal fragmentation**



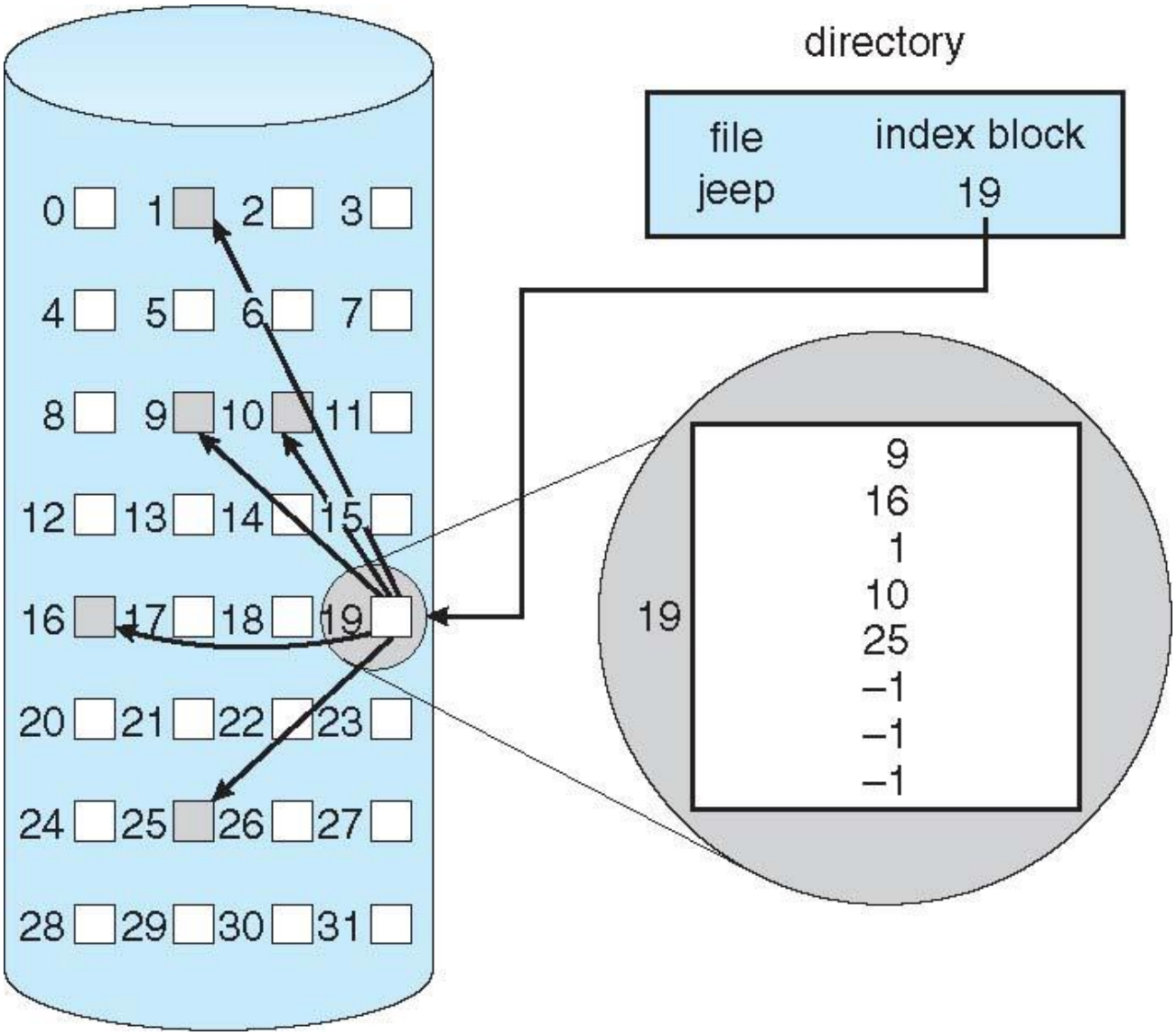
Indexed Allocation

- **Indexed allocation** brings **all pointers together** into **one location** called the **index block.**
- **Each file** has its **own index block**, which is an **array of disk-block addresses** (**address *i*** is the **address of** the ***i*th block** of the **file**).

- **Advantages**:
  - **Supports direct access**
  - **No external fragmentation**

- **Disadvantages**:
  - **Wasted space** within **index blocks**
  - **Data blocks** may be **spread all over** the **volume**, resulting in **many read/write head movements.**

Indexed Allocation



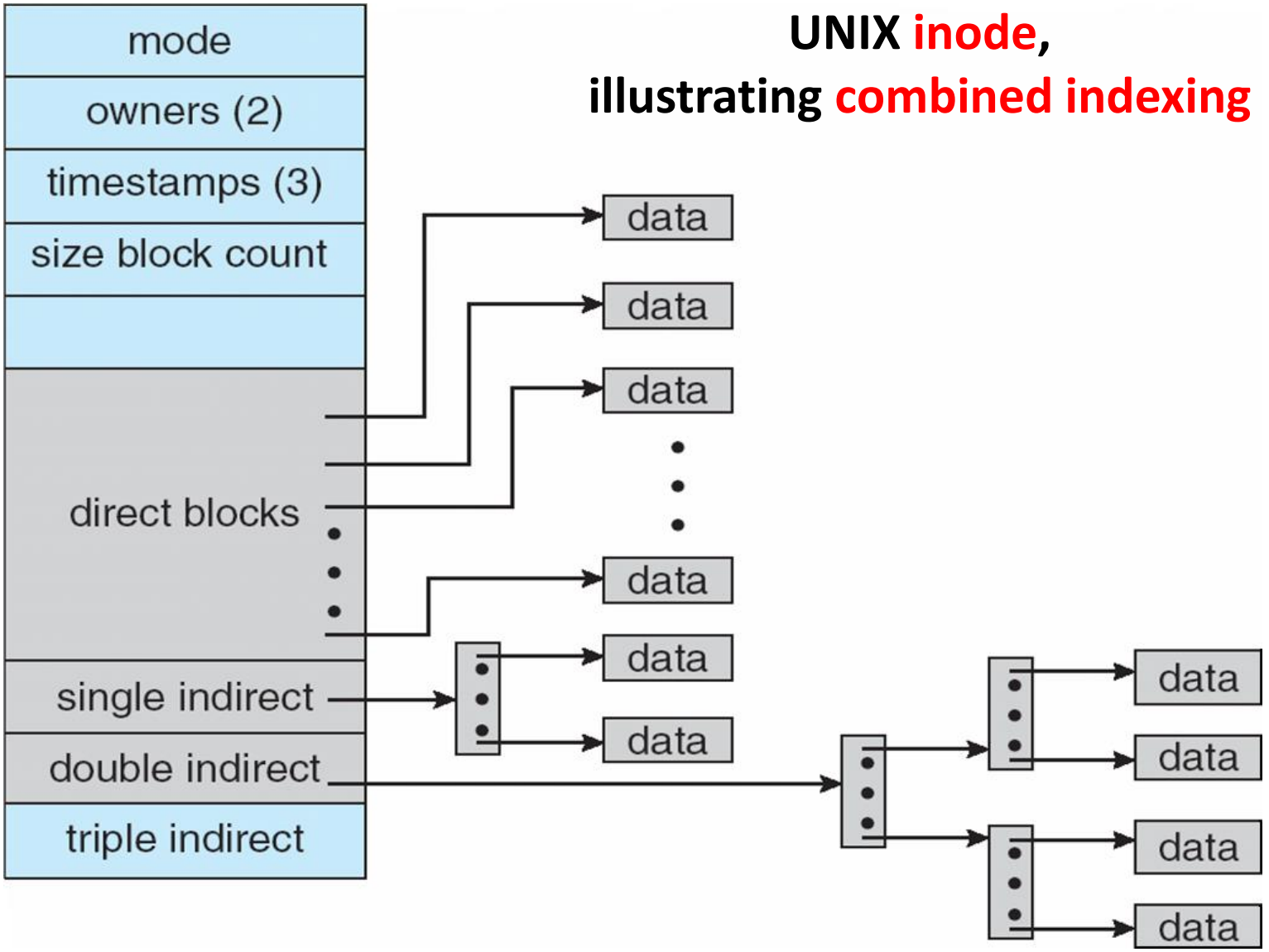
Indexed Allocation

- **How big should the index blocks be?**
  - A **small block** **cannot contain enough pointers** for a **large file**.
  - A **large block** **wastes space** with each **small file**.

- **Linked scheme:** for **large files**, link together several **index blocks**
- **Multilevel index:** a **first-level index block** points to a set of **second-level index blocks**, which contain pointers to file blocks.
- **Combined scheme:** Suppose we can store **15 pointers** of the **index block** are stored in the **FCB** (or **inode**, on **UNIX**).
  - **First 12** of these are pointers to **direct blocks** (that contain **file data**)
  - **Next 3** are pointers to **indirect blocks** (that contain **pointers**)
    - **First** points to a **single indirect block**
    - **Second** points to a **double indirect block**
    - **Third** points to a **triple indirect block**
  - This allows ***very*** **large file sizes** (UNIX implementations of this scheme support files that are terabytes in size).

Indexed Allocation

UNIX inode,
illustrating combined indexing



Disk Scheduling Algorithms

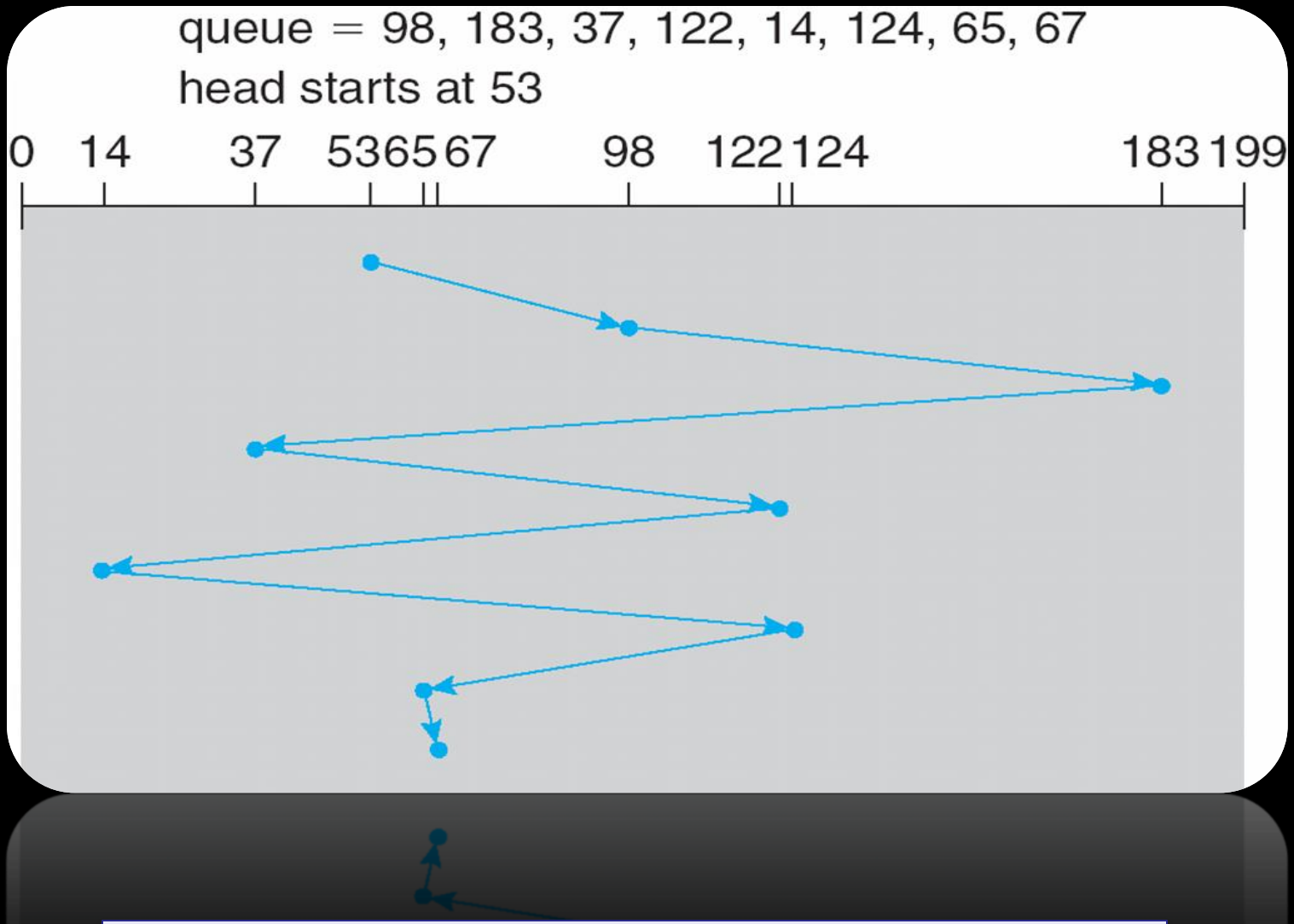
- Several algorithms exist to schedule the servicing of disk I/O requests.

I/O request queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

First Come First Serve (FCFS)

- Handle I/O requests sequentially
- Suffers from global zigzag effect.

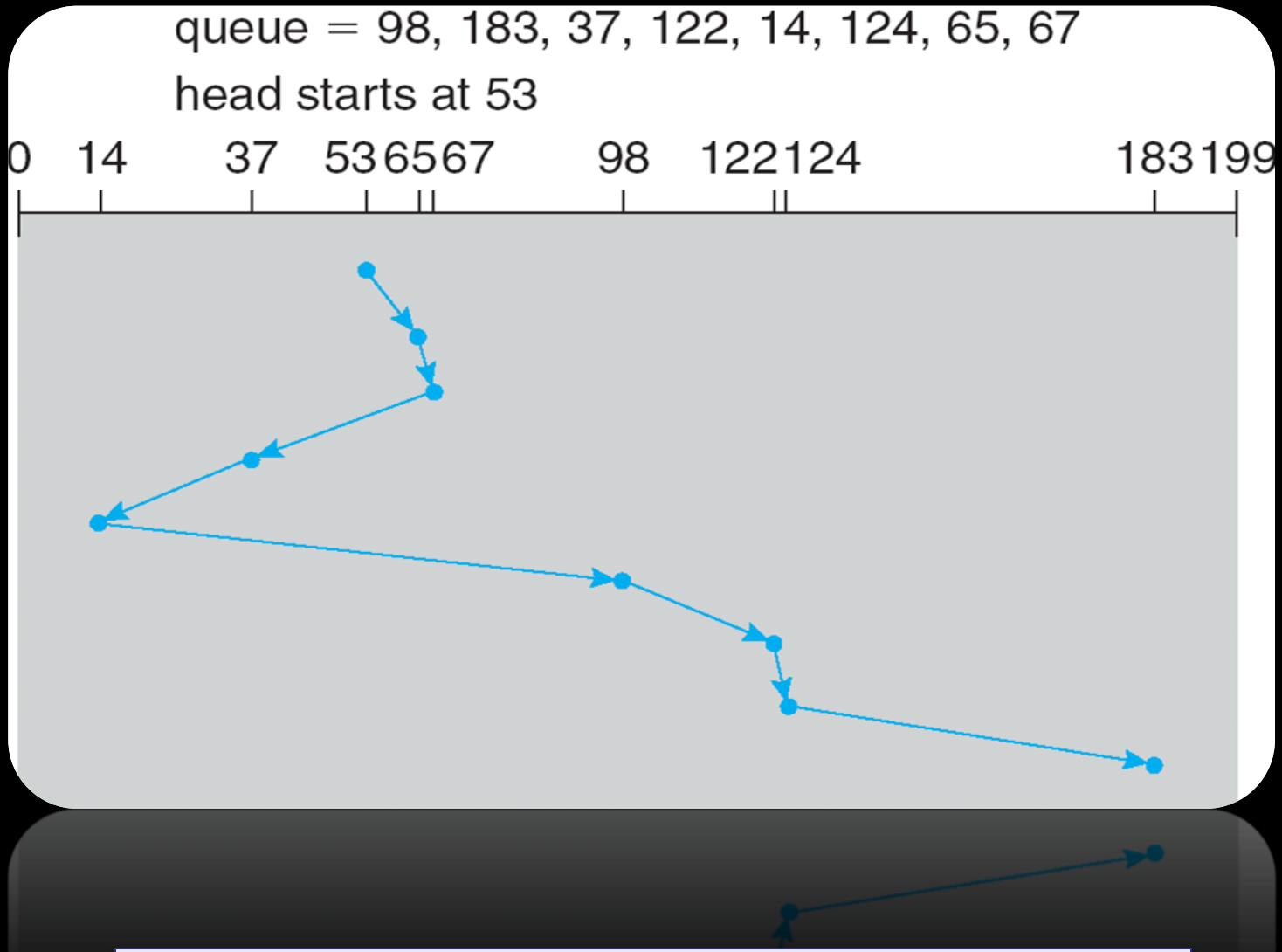


Total head movement = 640 cylinders.

Shortest Seek Time First (SSTF)

- Selects the request with the **minimum seek time** from the **current head position**.

- Also called **Shortest Seek Distance First** (SSDF)

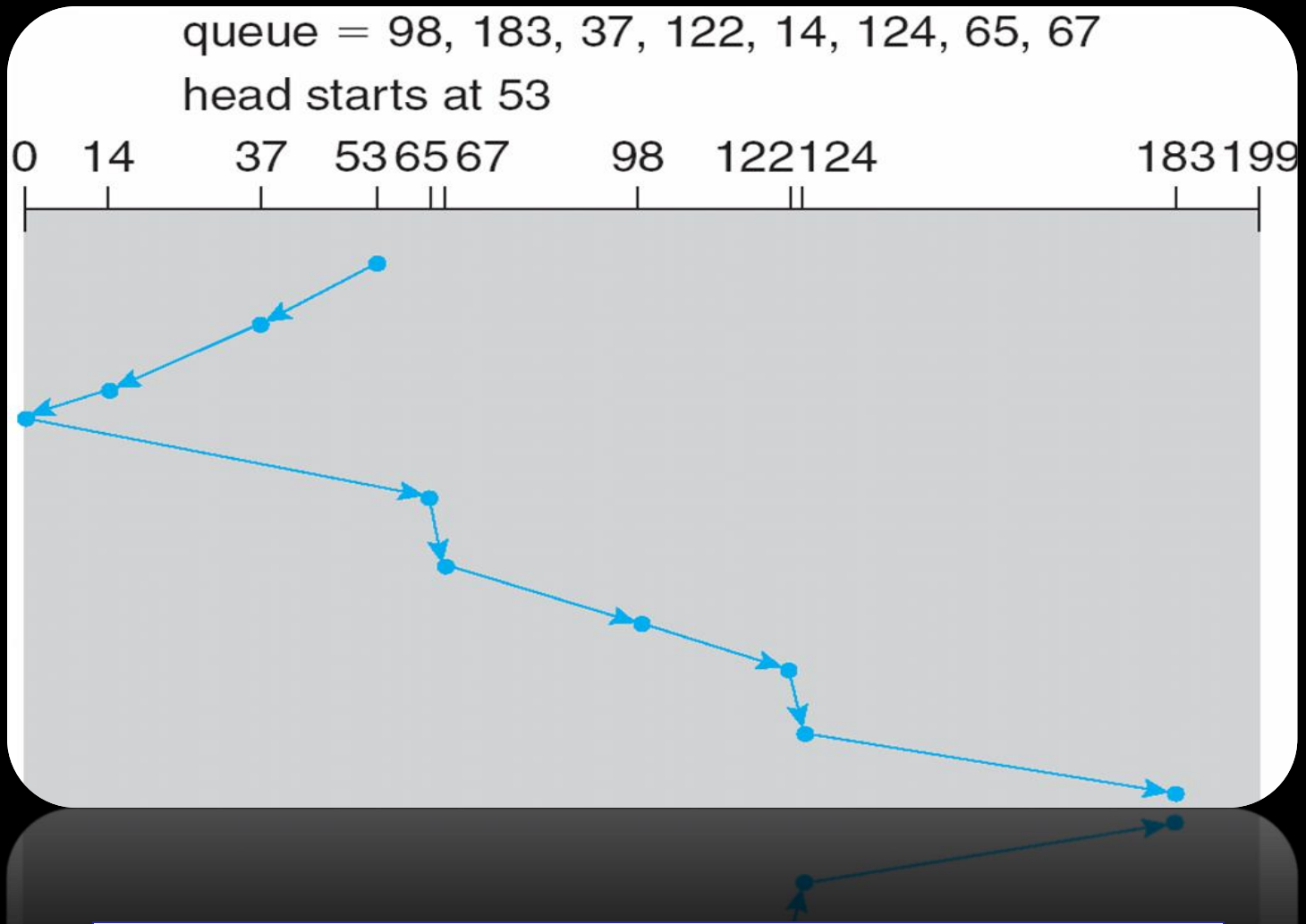


Total head movement = 236 cylinders

Scan

- The disk arm starts at one end of the disk, and moves toward the other end

- Servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.



Total head movement = 208 cylinders

C-Scan

- **The head moves from one end of the disk to the other, servicing requests as it goes.**

- When it reaches the other end, however, it immediately returns to the beginning of the disk, **without servicing any requests** on the return trip.