



The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

# Chapter 11: File System Implementation





# Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS
- Example: WAFL File System





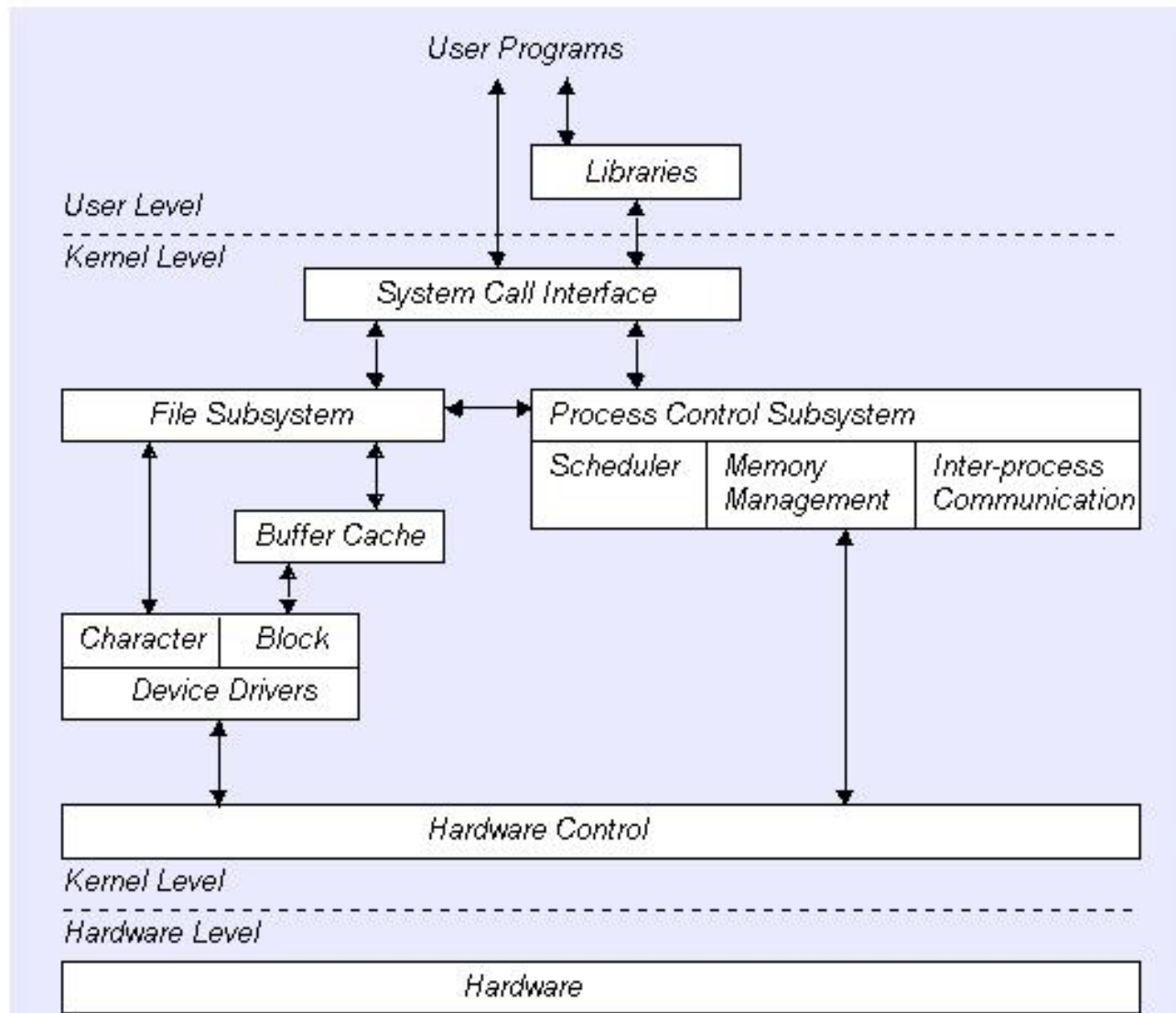
# Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs





# User program & Kernel interface in Linux



Note: This picture is excerpted from Write a Linux Hardware Device Driver, Andrew O' Shaughnessy, Unix world



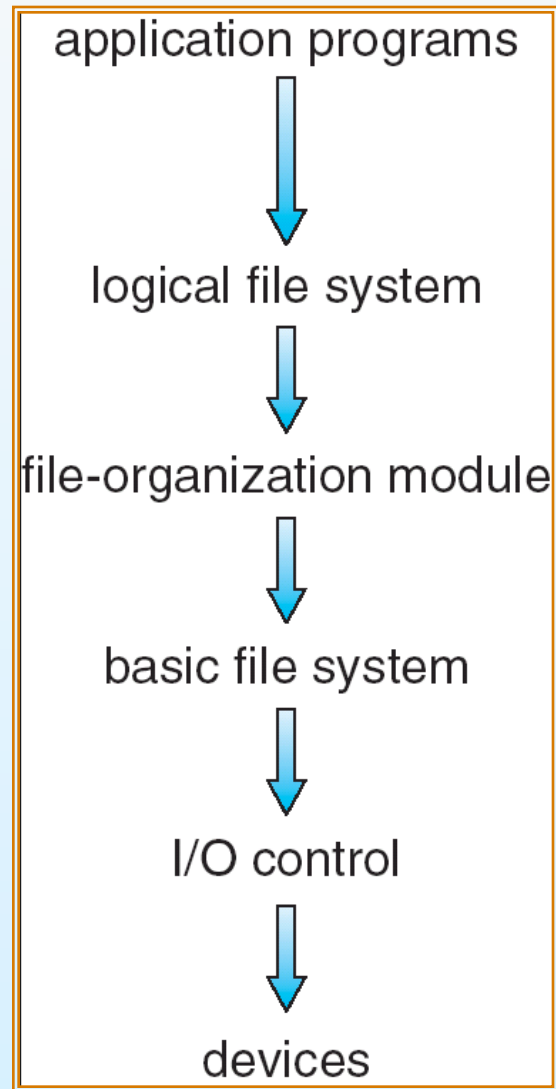
# File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks or SSD)
- File system organized into layers
- **File control block** – storage structure consisting of information about a file





# Layered File System





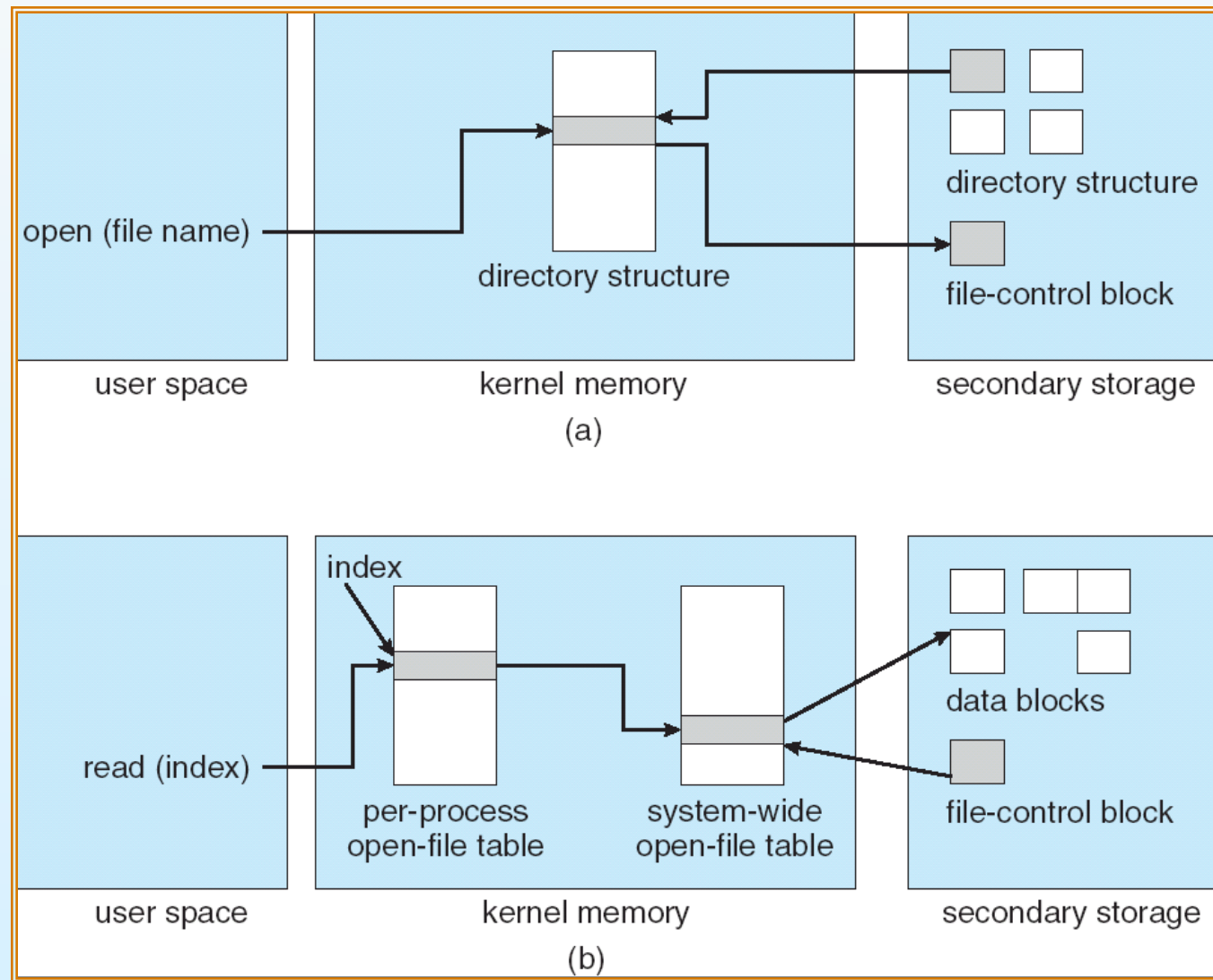
# A Typical File Control Block

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





# In-Memory File System Structures







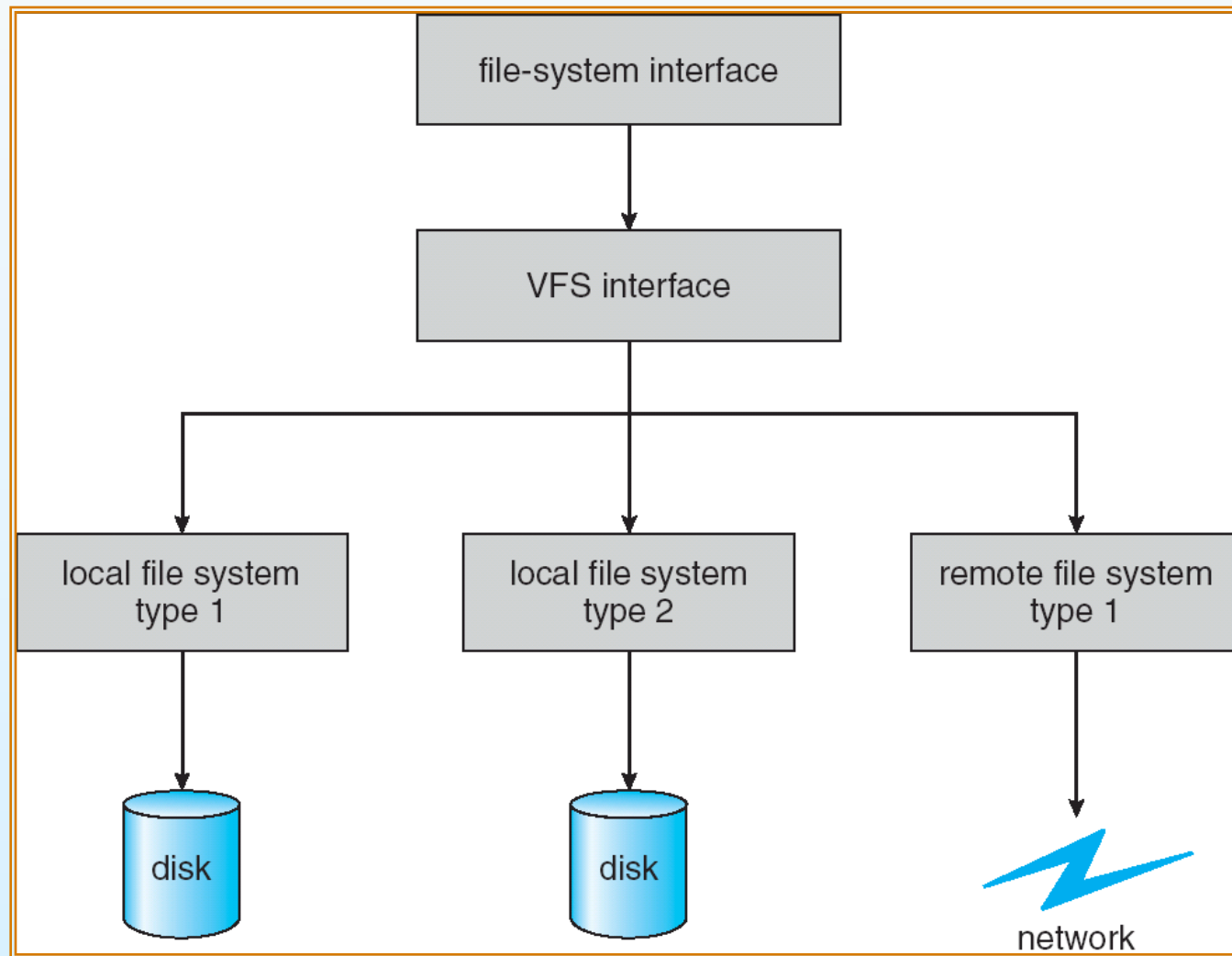
# Virtual File Systems

- Virtual File Systems (VFS) 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.
- The API is to the VFS interface, rather than any specific type of file system.
  - Basic operations are open, close, read & write



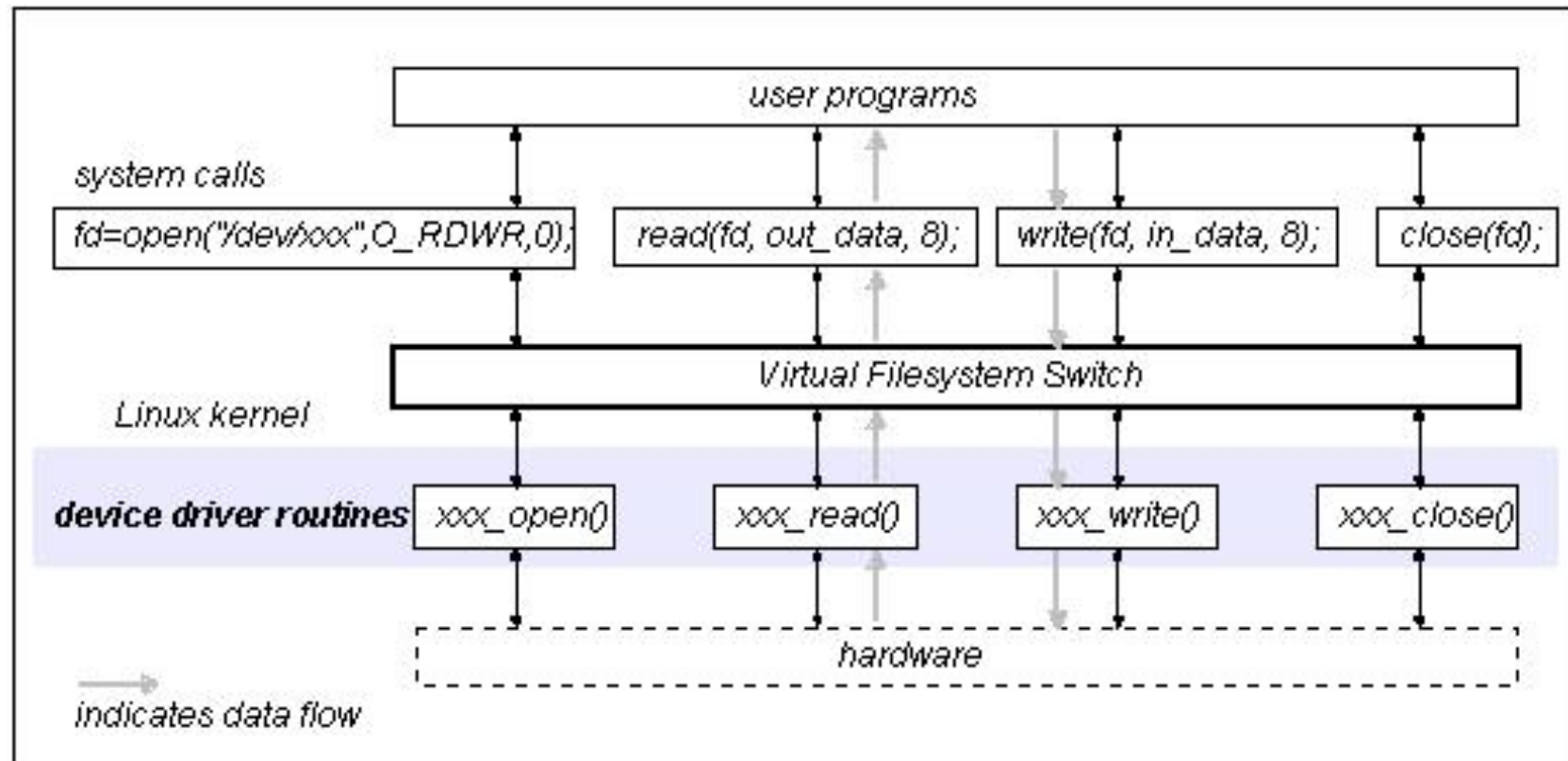


# Schematic View of Virtual File System





# VFS & Device Driver interface



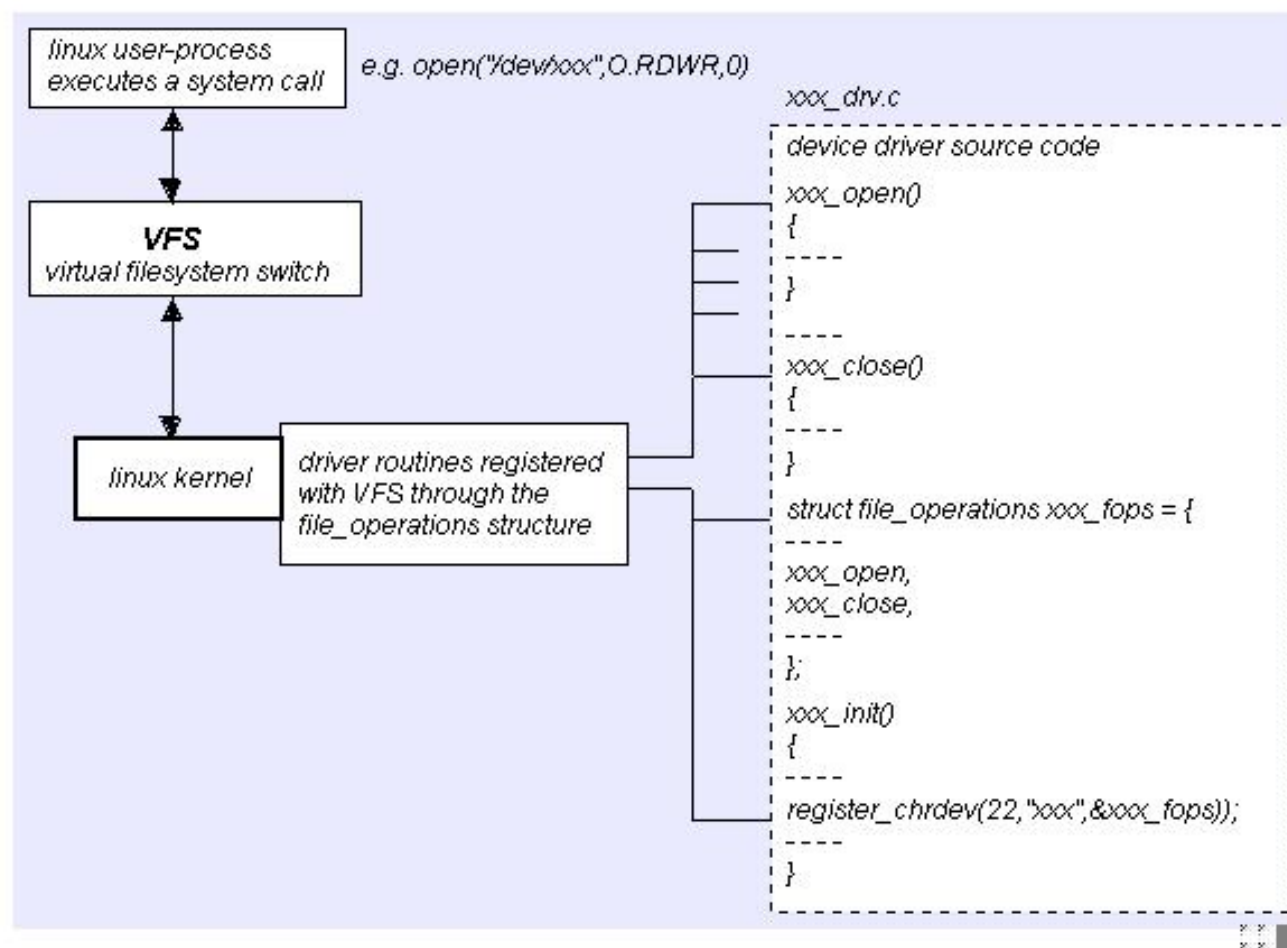
Note: This picture is excerpted from Write a Linux Hardware Device Driver, Andrew O' Shaughnessy, Unix world





# VFS & Major number

- principal interface between a device driver and Linux kernel





# Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
  
- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size





# Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**





# Contiguous Allocation

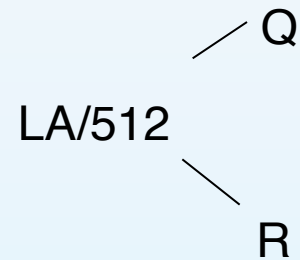
- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow





# Contiguous Allocation

- Mapping from logical to physical



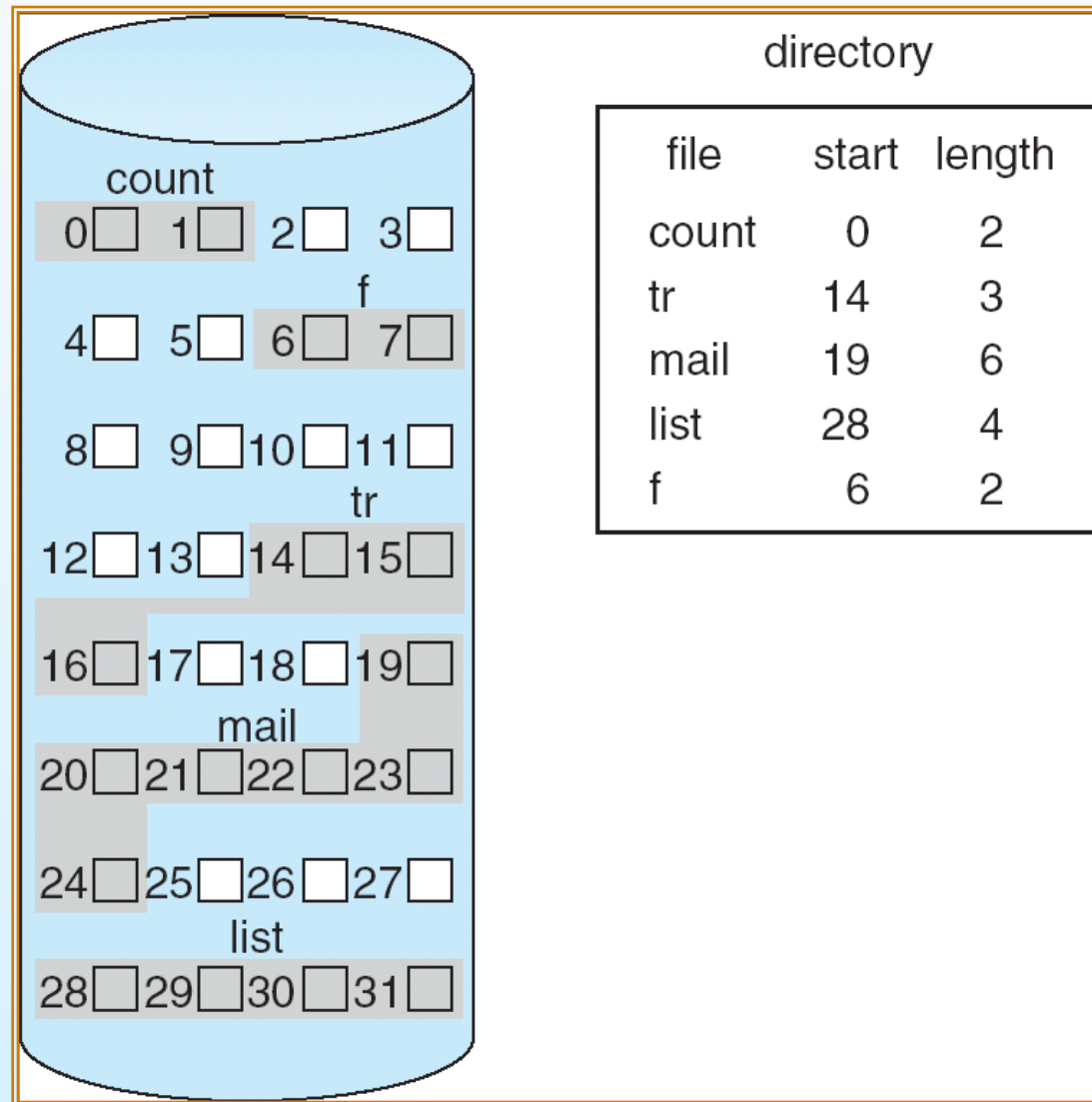
Block to be accessed =  $Q + \text{starting address}$   
Displacement into block =  $R$







# Contiguous Allocation of Disk Space





# Extent-Based Systems

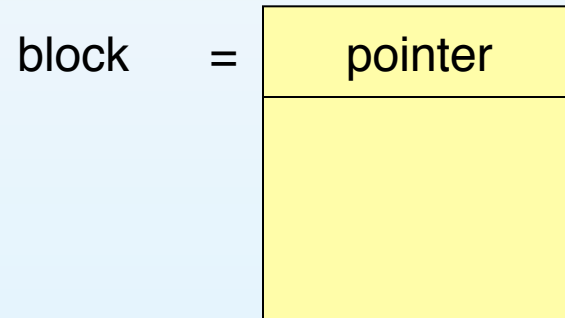
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents.





# Linked Allocation

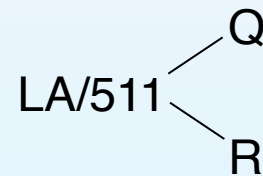
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.





## Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping



Block to be accessed is the Qth block in the linked chain of blocks representing the file.

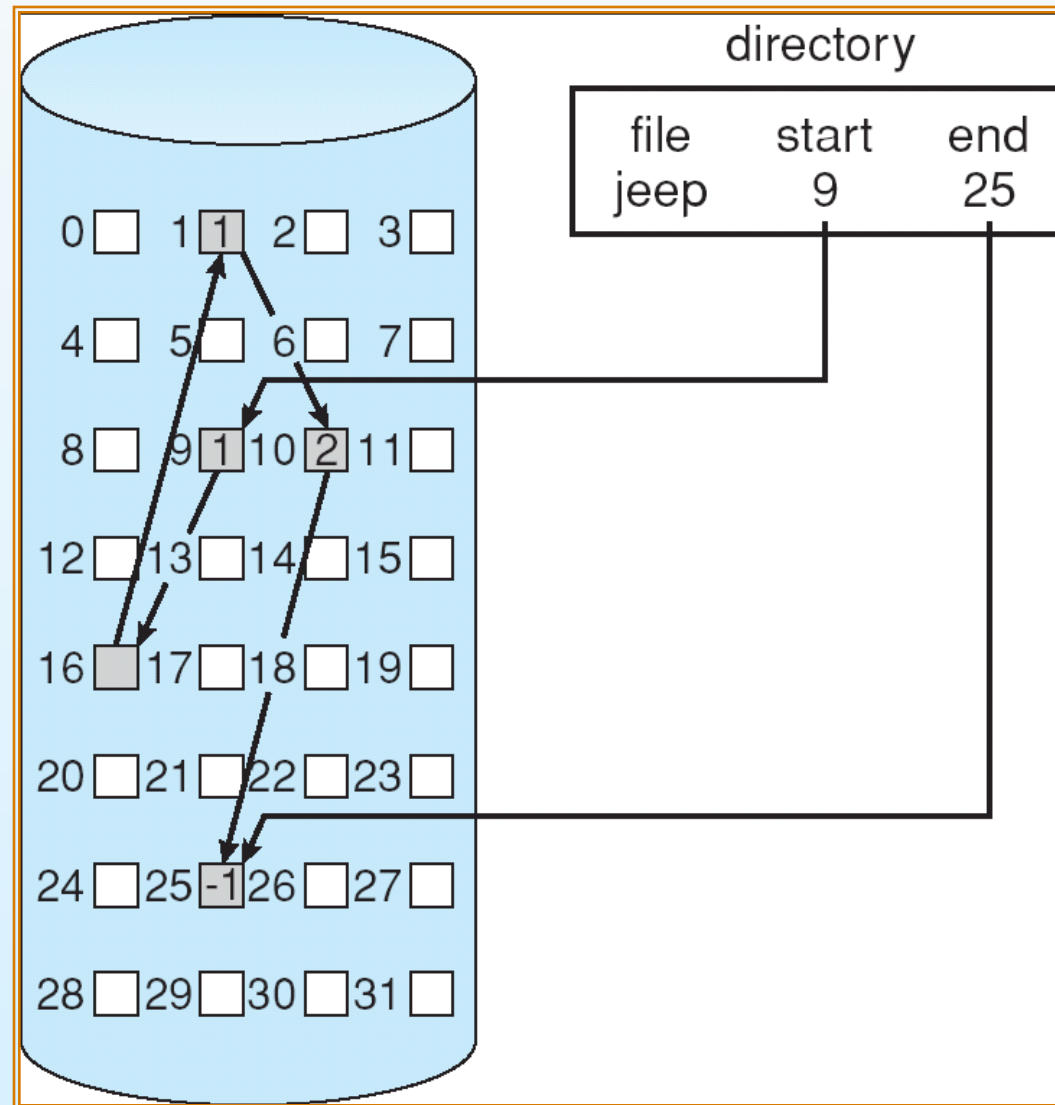
Displacement into block =  $R + 1$

File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.



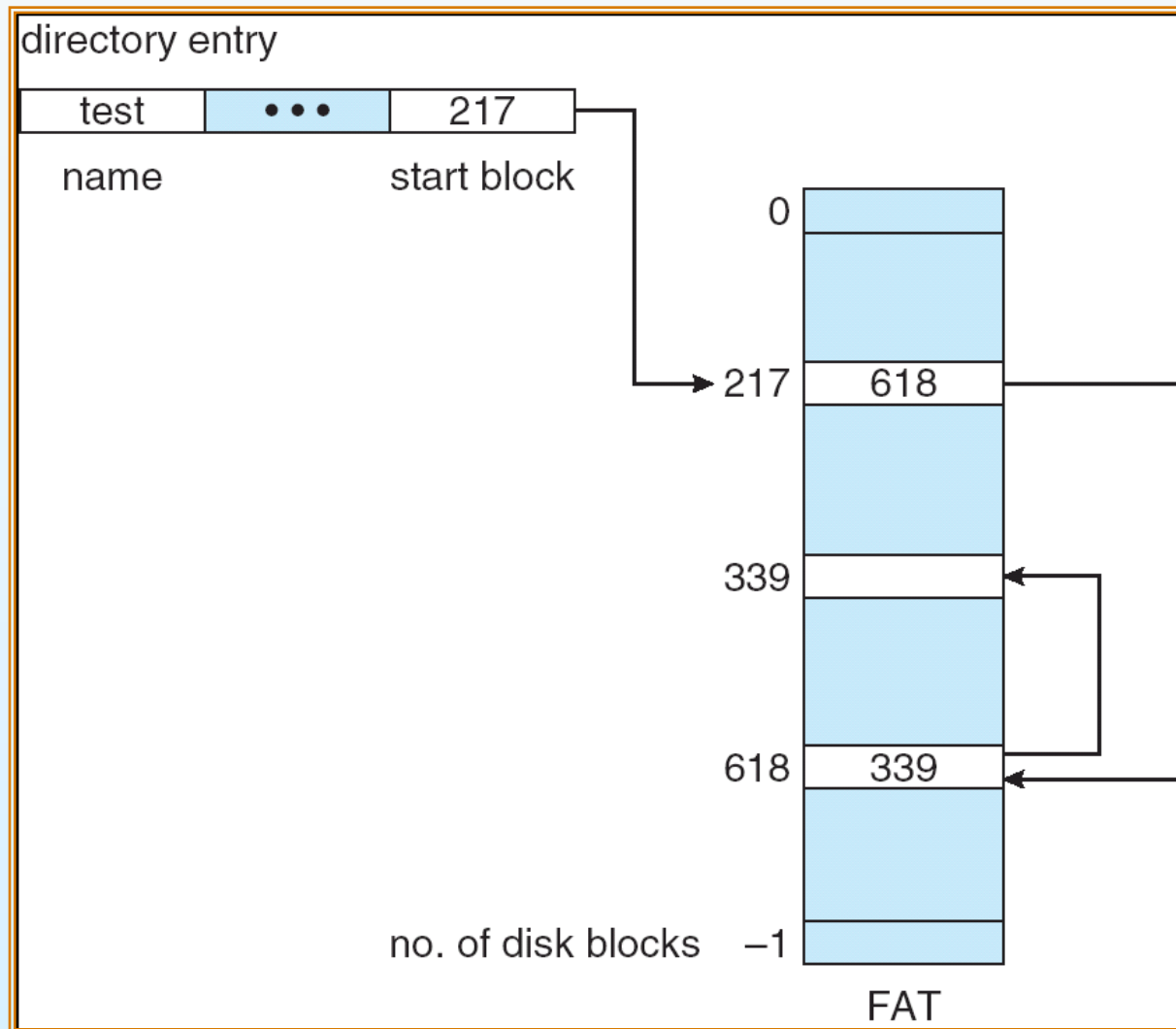


# Linked Allocation





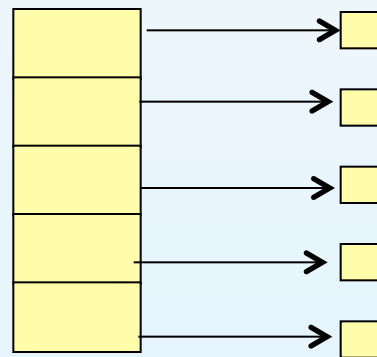
# File-Allocation Table





# Indexed Allocation

- Brings all pointers together into the *index block*.
- Logical view.

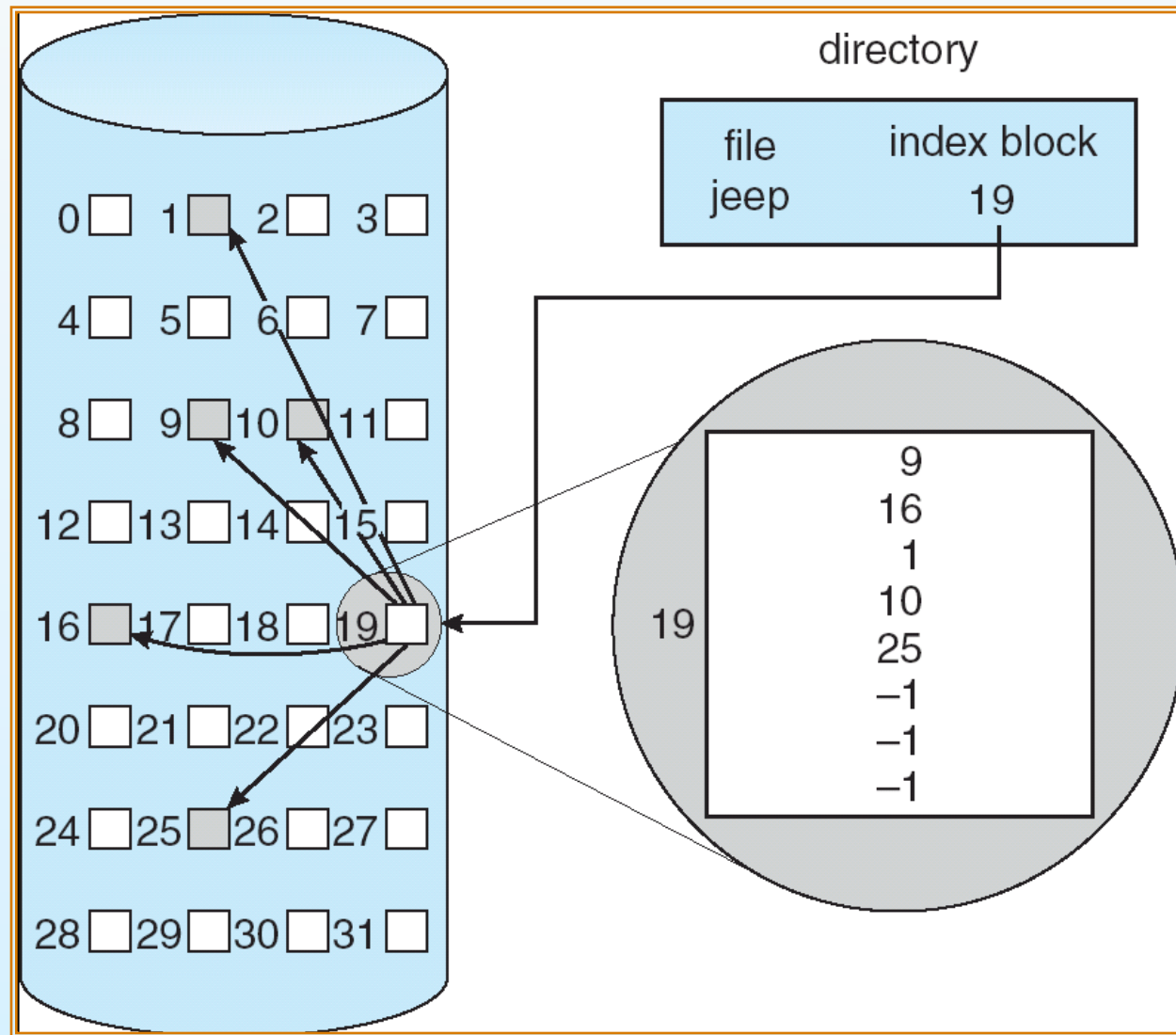


index table





# Example of Indexed Allocation

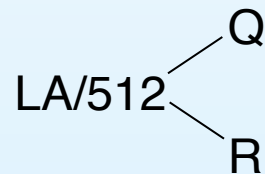






# Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file and block size of 512 words. We need only 1 block for index table.

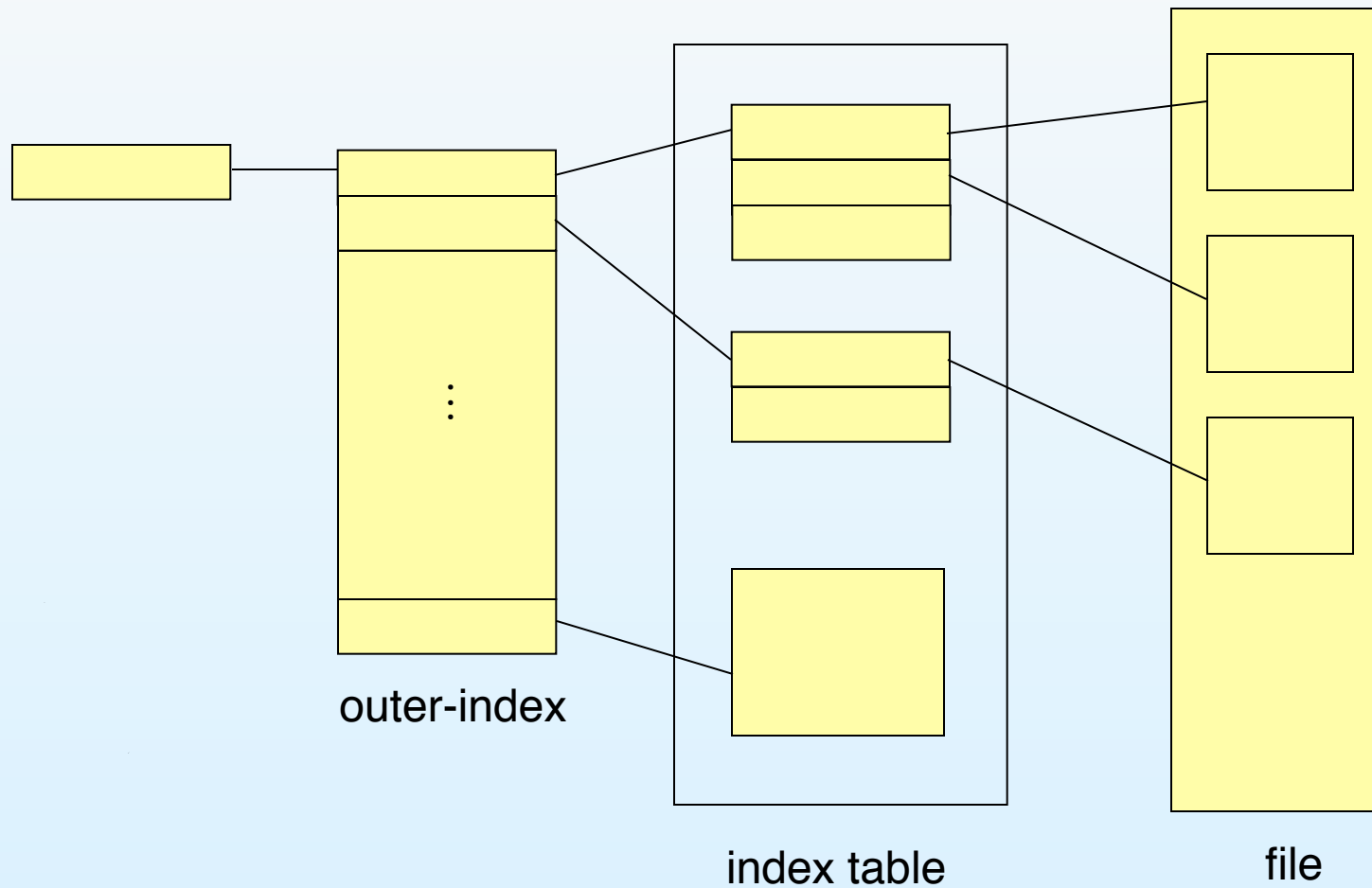


Q = displacement into index table  
R = displacement into block





# Indexed Allocation – Mapping (Cont.)





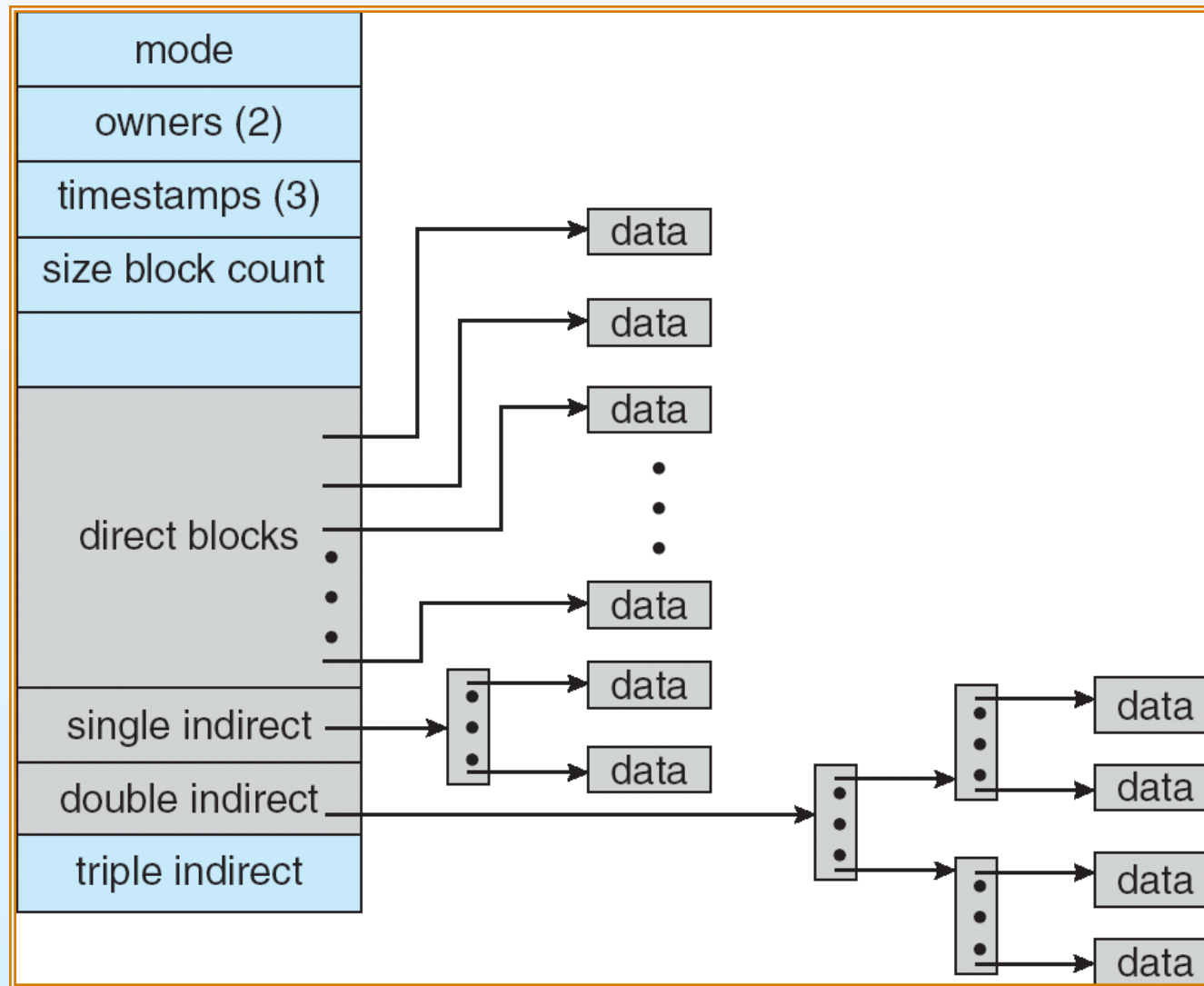
# Summary

- Contiguous Allocation
  - Each file occupies a set of contiguous blocks on the disk
  - Simple
  - Random access
  - Wasteful of space (dynamic storage-allocation problem)
  - Files cannot grow
- Linked Allocation
  - Simple – need only starting address
  - Free-space management system – no waste of space
  - No random access
- Indexed Allocation
  - Need index table
  - Random access
  - Dynamic access without external fragmentation, but have overhead of index block.





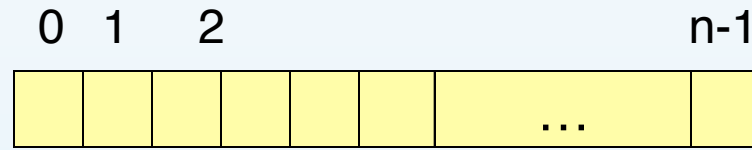
# Combined Scheme: UNIX/Linux (4K bytes per block)





# Free-Space Management

- Bit vector ( $n$  blocks)



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

Block number calculation

(number of bits per word) \*  
(number of 0-value words) +  
offset of first 1 bit





# Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:
    - block size =  $2^{12}$  bytes
    - disk size =  $2^{30}$  bytes (1 gigabyte)
    - $n = 2^{30}/2^{12} = 2^{18}$  bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting





# Free-Space Management (Cont.)

- Need to protect:
  - Pointer to free list
  - Bit map
    - ▶ Must be kept on disk
    - ▶ Copy in memory and disk may differ
    - ▶ Cannot allow for block[*i*] to have a situation where  $\text{bit}[i] = 1$  in memory and  $\text{bit}[i] = 0$  on disk
  - Solution:
    - ▶ Set  $\text{bit}[i] = 1$  in disk
    - ▶ Allocate block[*i*]
    - ▶ Set  $\text{bit}[i] = 1$  in memory





# Directory Implementation

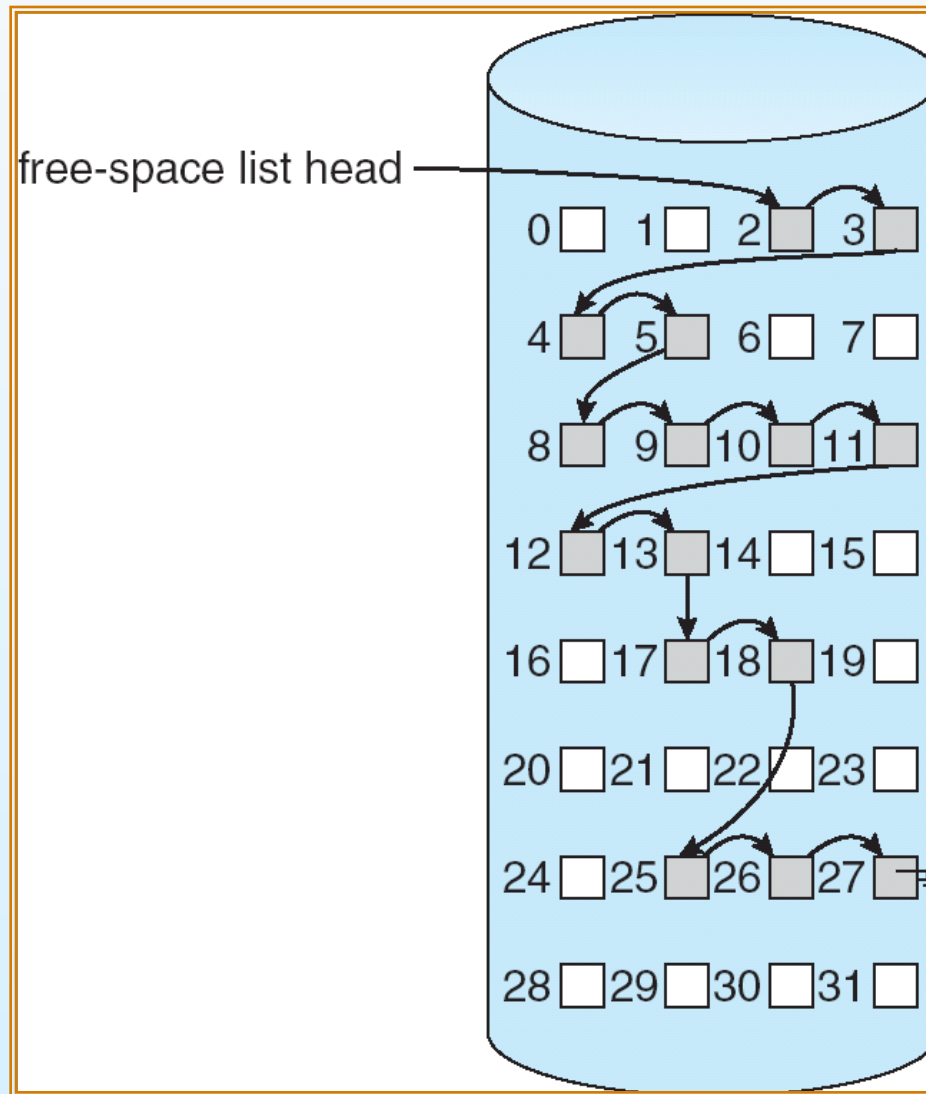
- Linear list of file names with pointer to the data blocks
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size







# Linked Free Space List on Disk





# Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
  
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk





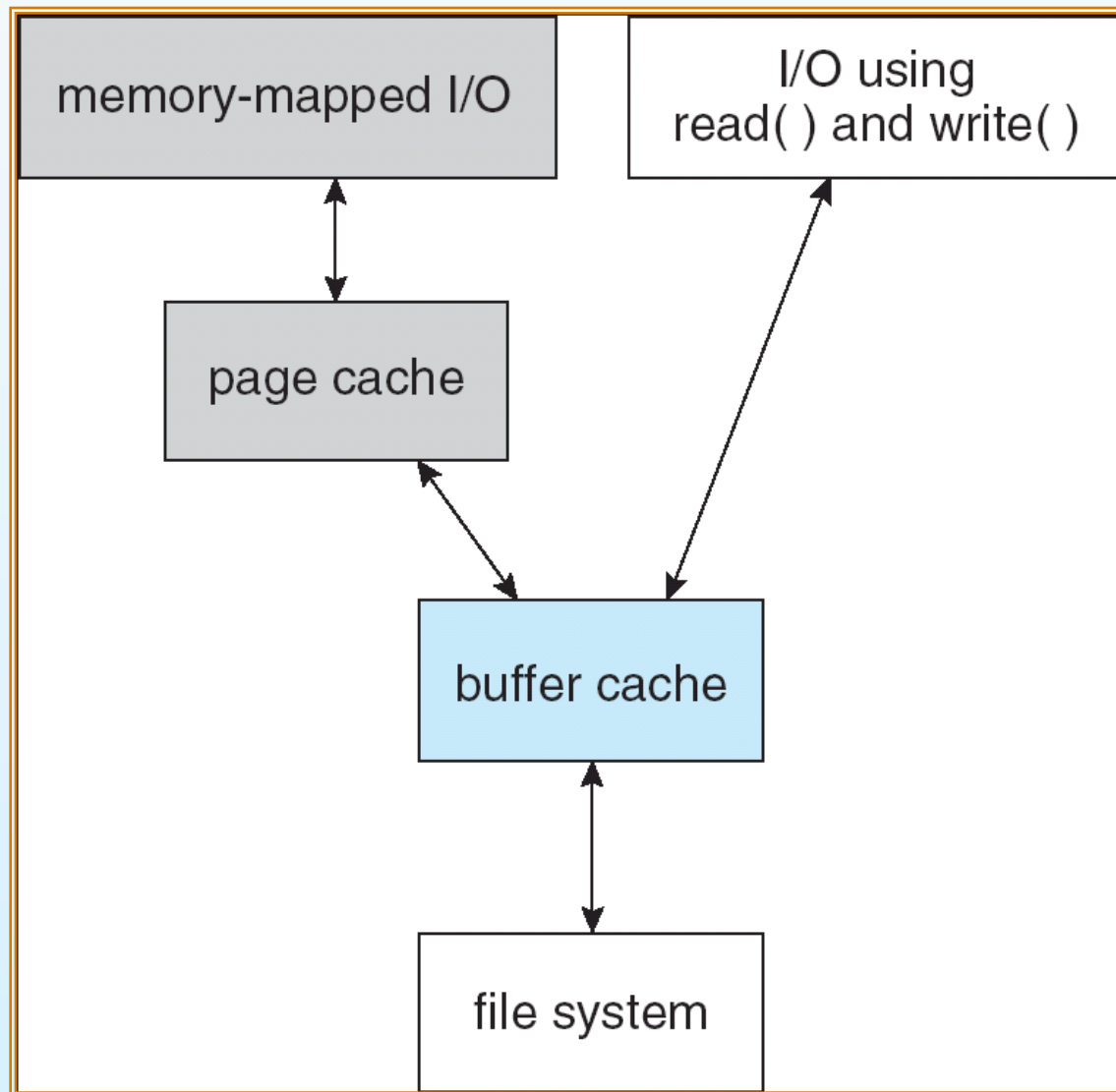
# Page Cache

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





# I/O Without a Unified Buffer Cache





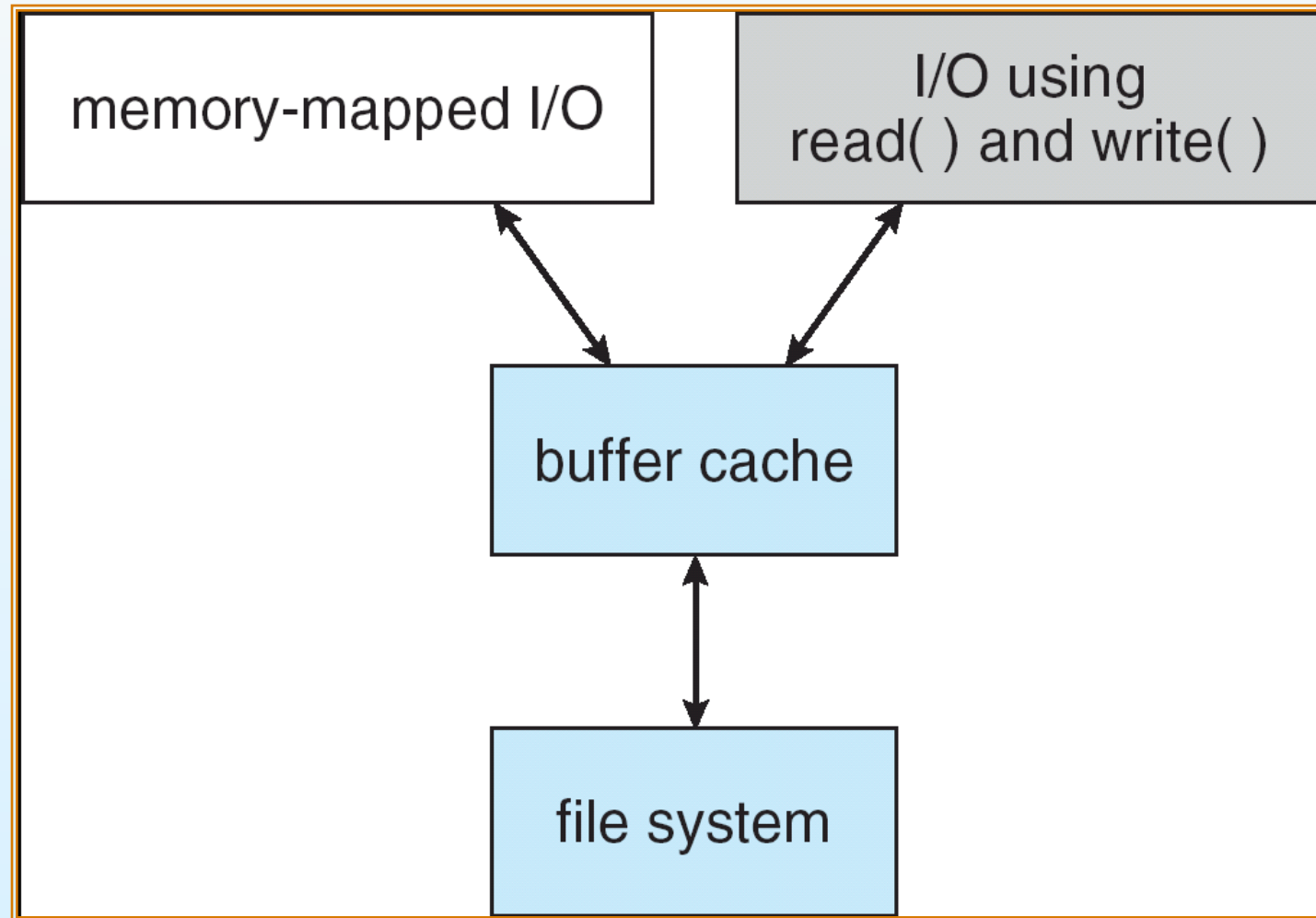
# Unified Buffer Cache

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





# I/O Using a Unified Buffer Cache





# Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up** data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Network storage & Disaster Recovery site
- Recover lost file or disk by **restoring** data from backup





# Log Structured File Systems

- **Log structured** (or journaling) file systems record each 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
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed







# Stop here Apr 25





# The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet)





## NFS (Cont.)

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory is mounted over a local file system directory
    - ▶ The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
  - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - ▶ Files in the remote directory can then be accessed in a transparent manner
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory





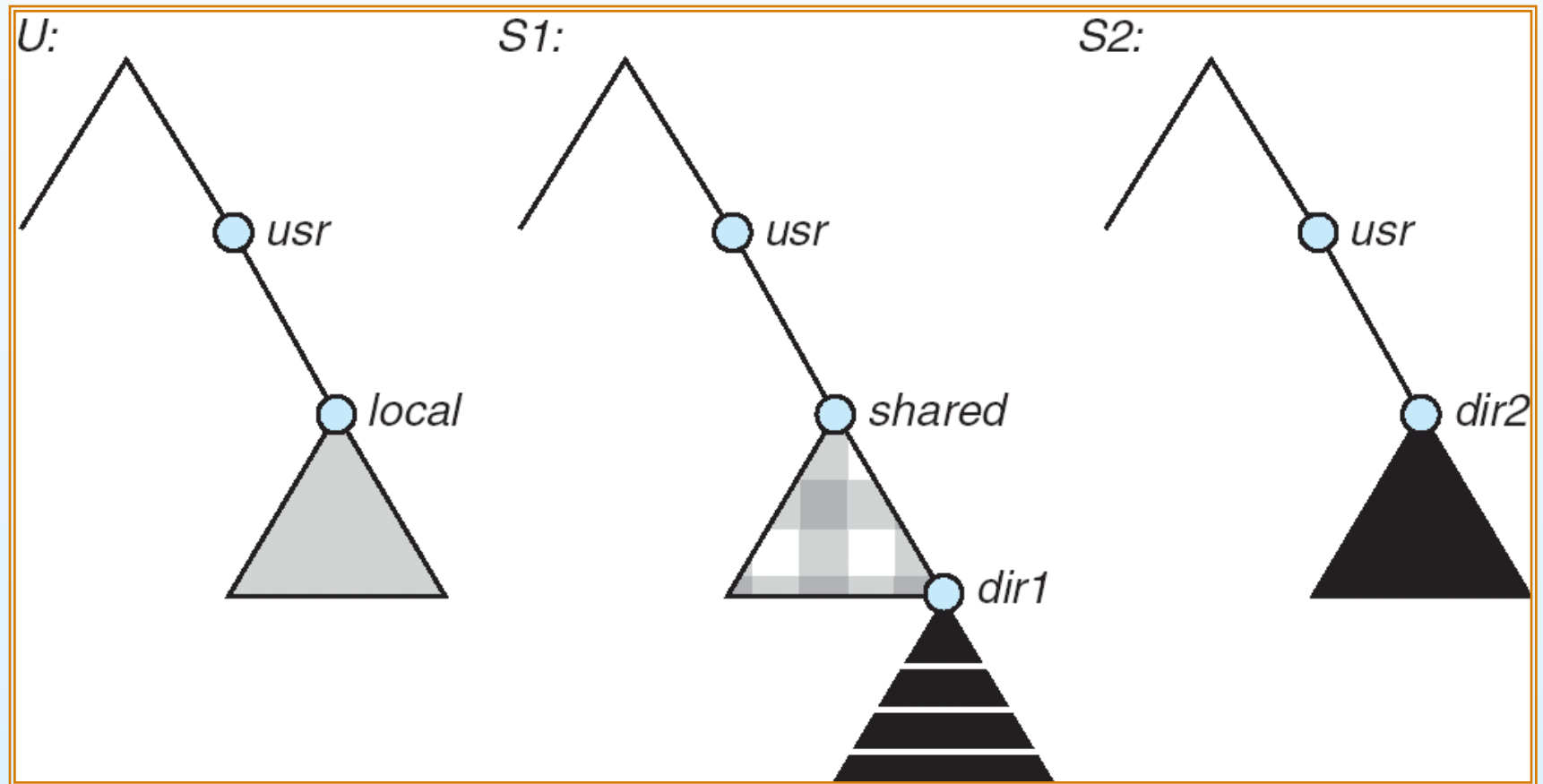
## NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services



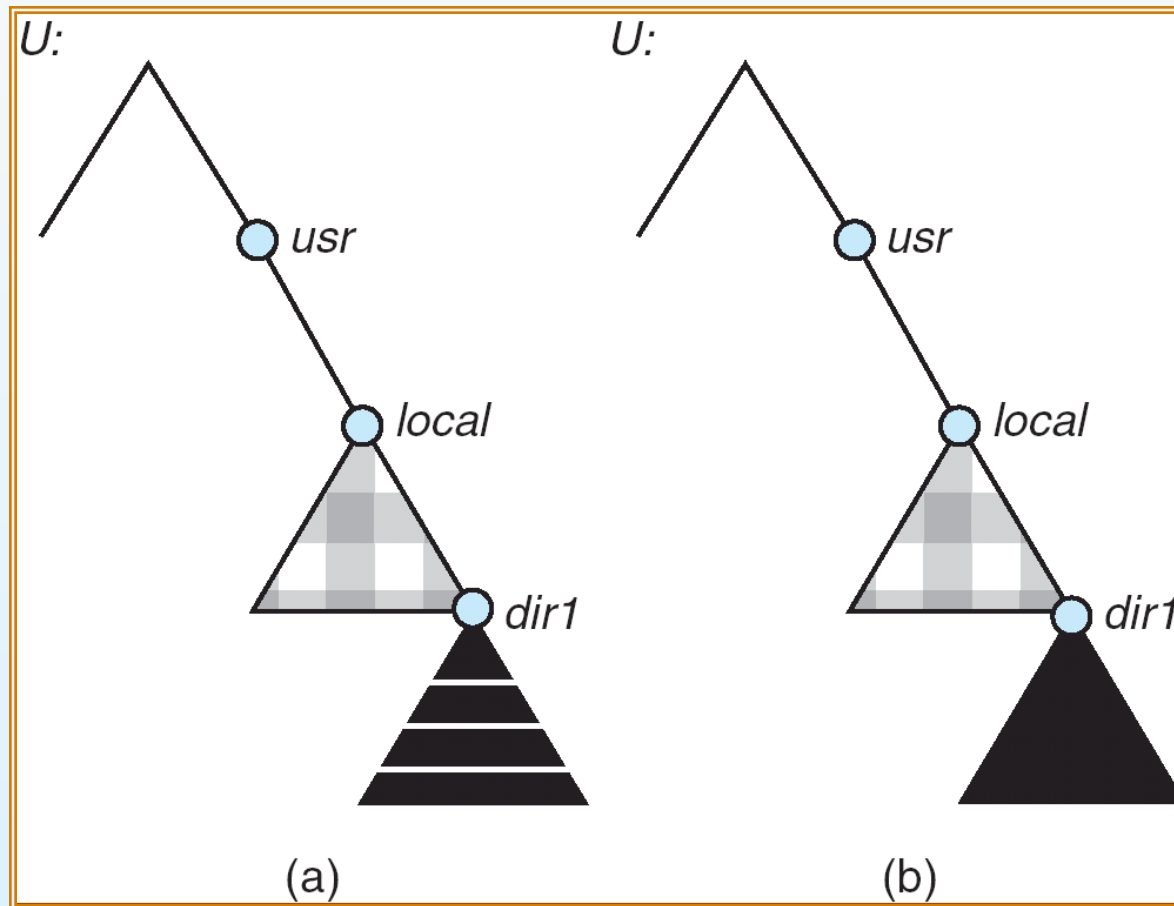


# Three Independent File Systems





# Mounting in NFS



Mounts

Cascading mounts





# NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side





# NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
  - searching for a file within a directory
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments  
(NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms







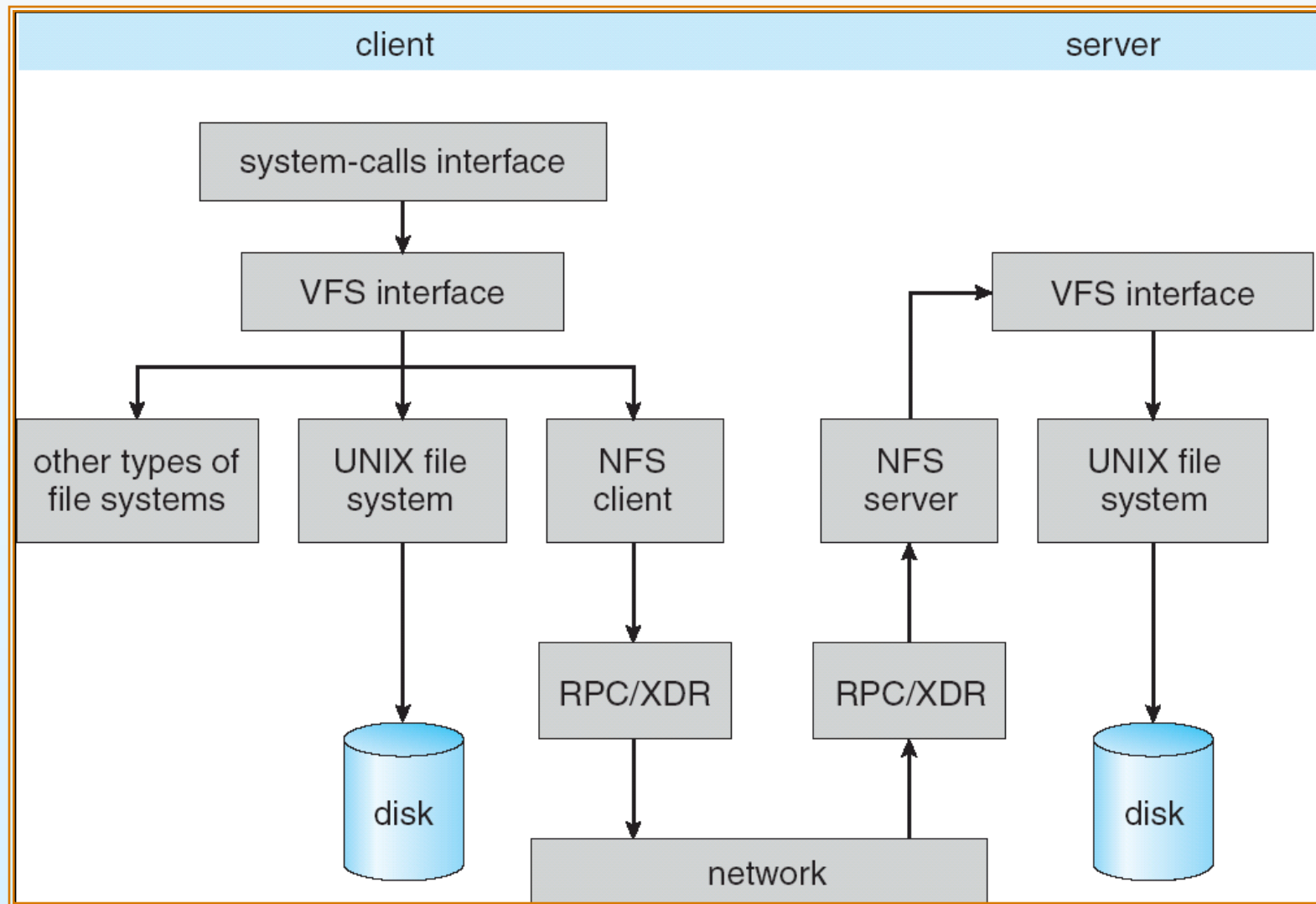
# Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open**, **read**, **write**, and **close** calls, and **file descriptors**)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
  - Implements the NFS protocol





# Schematic View of NFS Architecture





# NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names





# NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk





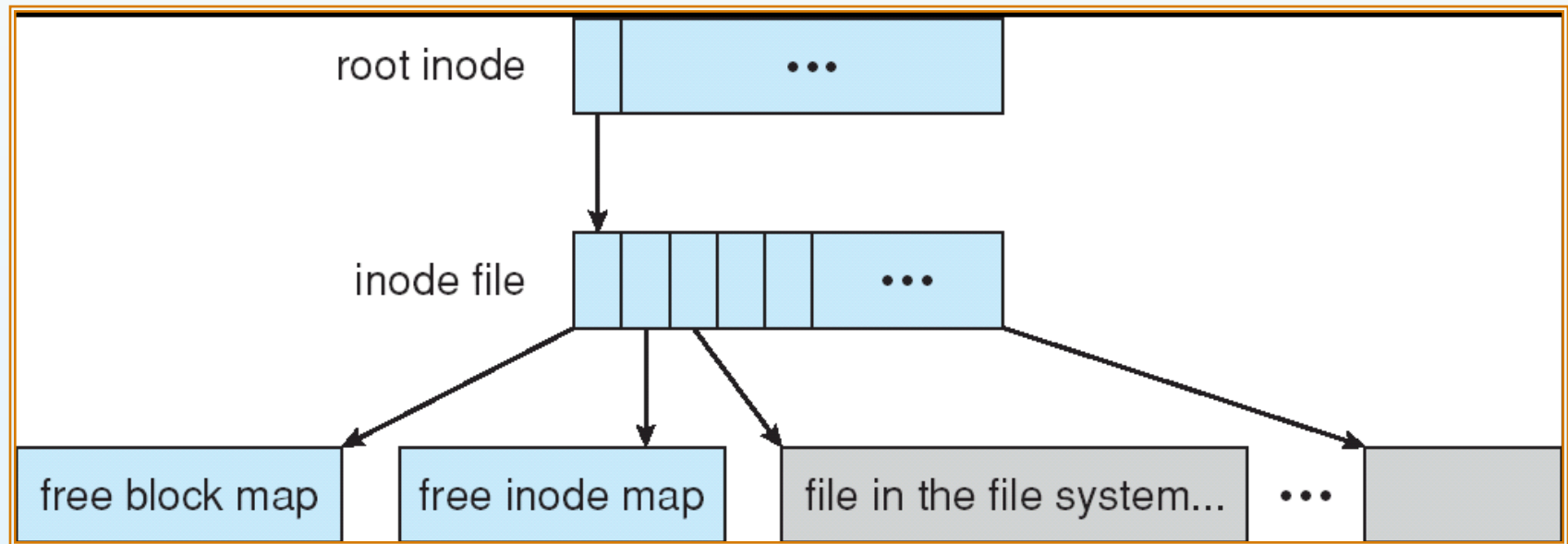
# Example: WAFL File System

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications



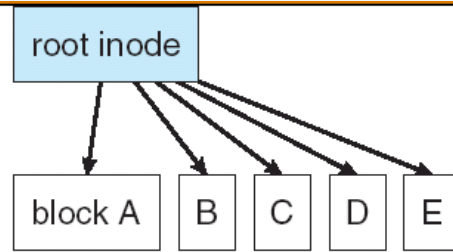


# The WAFL File Layout

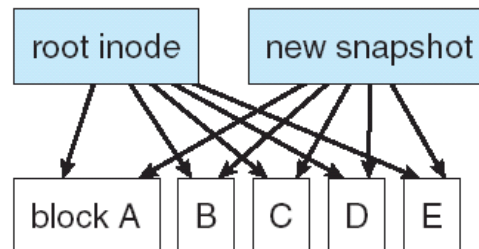




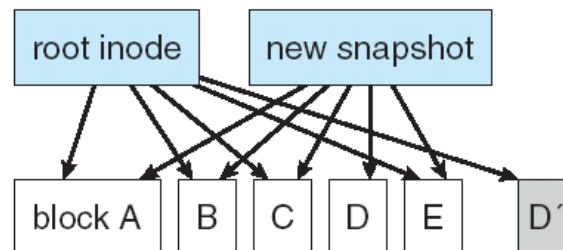
# Snapshots in WAFL



(a) Before a snapshot.



(b) After a snapshot, before any blocks change.



(c) After block D has changed to D'.





## 11.02

file permissions

file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks







The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

# End of Chapter 11

