Chapter 11
File System Implementation

Contents

- 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

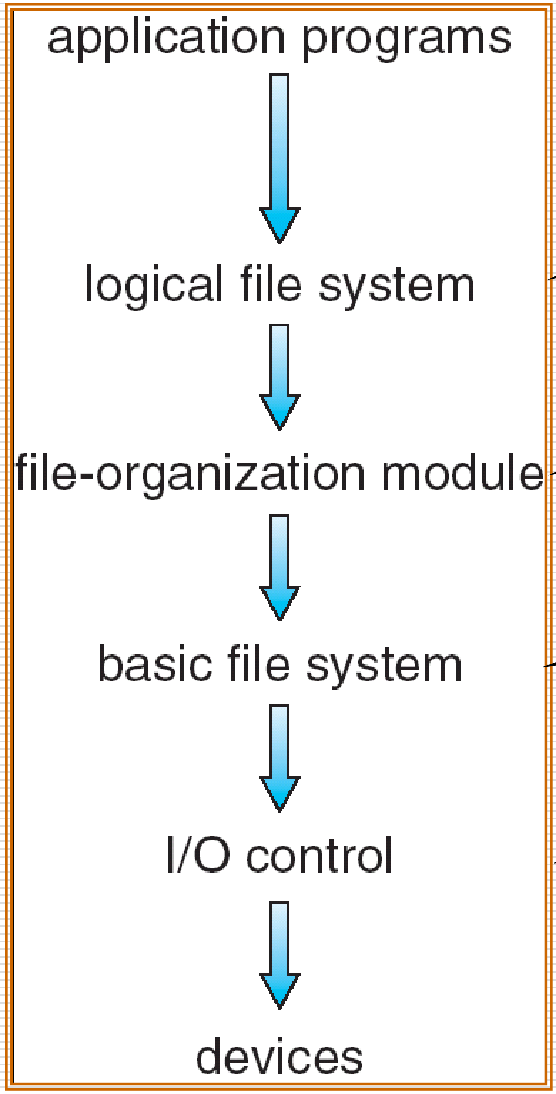
11.1 File-System Structure

- Characteristics that make disks convenient medium
  - A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same place.
  - From a disk, we can access directly any given block of information it contains.
- OS imposes one or more file systems to allow the data to be stored, located, and retrieved easily
- A file system poses two design problems:
  - How should the file system look to the user
  - How to create algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

File-System Structure

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

Layered File System



Manages metadata information, FCB will be needed for each file.

Knows about files and their logical blocks, as well as physical blocks.

Issues generic commands to the appropriate device driver to read and write blocks on the disk

Consists of device drivers and interrupt handlers to transfer information between memory and disk.

File system implementation

- Several on-disk and in-memory structures are used to implement a file system.
- On disk, the file system contains information about how to boot an OS, total number of blocks, the free blocks, directory structure, and so on.
  - A boot control block
  - Volume control block contains partition information—number of blocks, size of the blocks, free block count and pointers. UFS: superblock; NTFS: master file table.
  - A directory structure
  - A per-file FCB contains many details about the file.

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

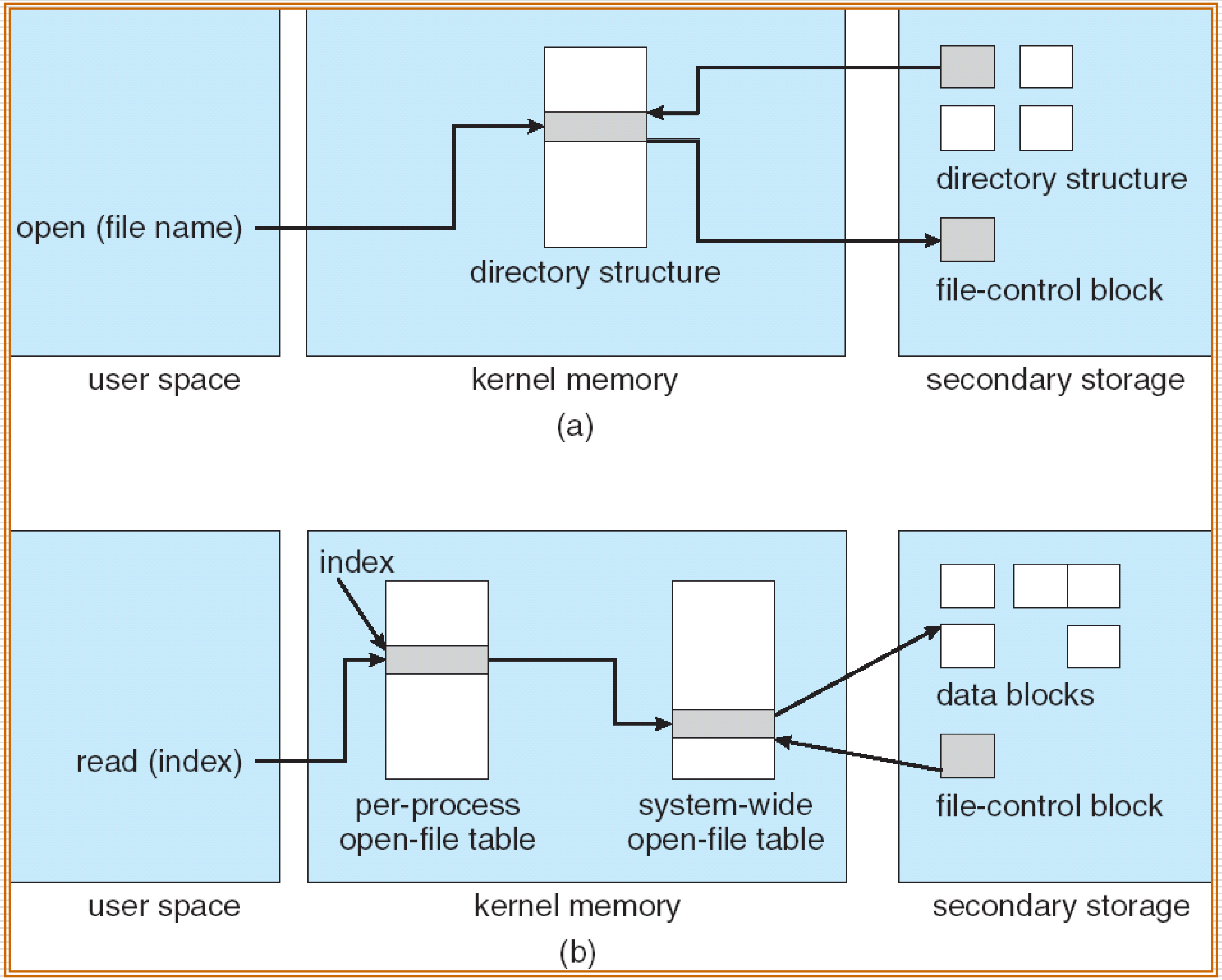
File system implementation

- In memory
  - An in-memory mount table contains information about each mounted volume.
  - An in-memory directory-structure cache holds the directory information of recently accessed directories.
  - System-wide open-file table contains a pointer of the FCB of each open file, as well as other information.
  - Per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information.

In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.

In-Memory File System Structures



Create, open, use and close a file

- Create a new file:
  - Locate a new FCB
  - Read appropriate directory into memory
  - Update it with the new file name and FCB
  - Write it back to the disk
- Open a file
  - Find the file
  - Copy the FCB to a system-wide open-file table in memory
  - Made an entry in the per-process open-file table
  - Return a pointer to the entry in the per-process open-file table
    - File descriptor in Unix
    - File handler in Windows

Create, open, use and close a file

- **Use**
  - **All operations are performed via the file pointer**
- **Close a file**
  - **The entry in per-process open-file table is removed**
  - **Open count in system-wide open-file table is decremented. If the open count is 0, then the entry is removed**

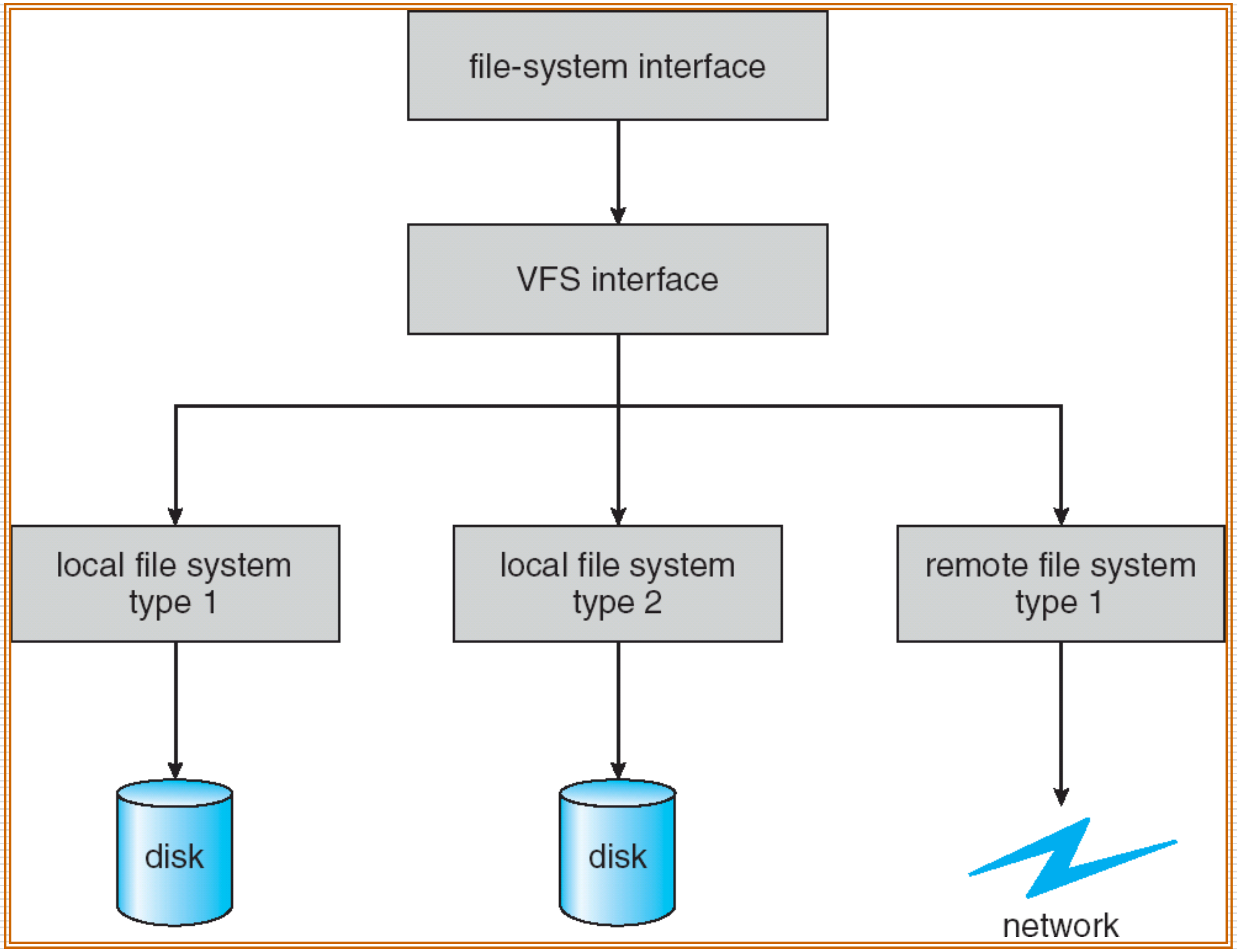
Partitions and Mounting

- Each partition can be raw or cooked
  - Swap
- Boot information can be stored in a separate partition.
- System can be dual-booted
  - A boot loader is needed.
    - BootManager bootstar 8.3，
    - Linux GRUB, GRUB - GRand Unified Bootloader
- Root partition contains the OS kernel and sometimes other system files.
- mounting

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.

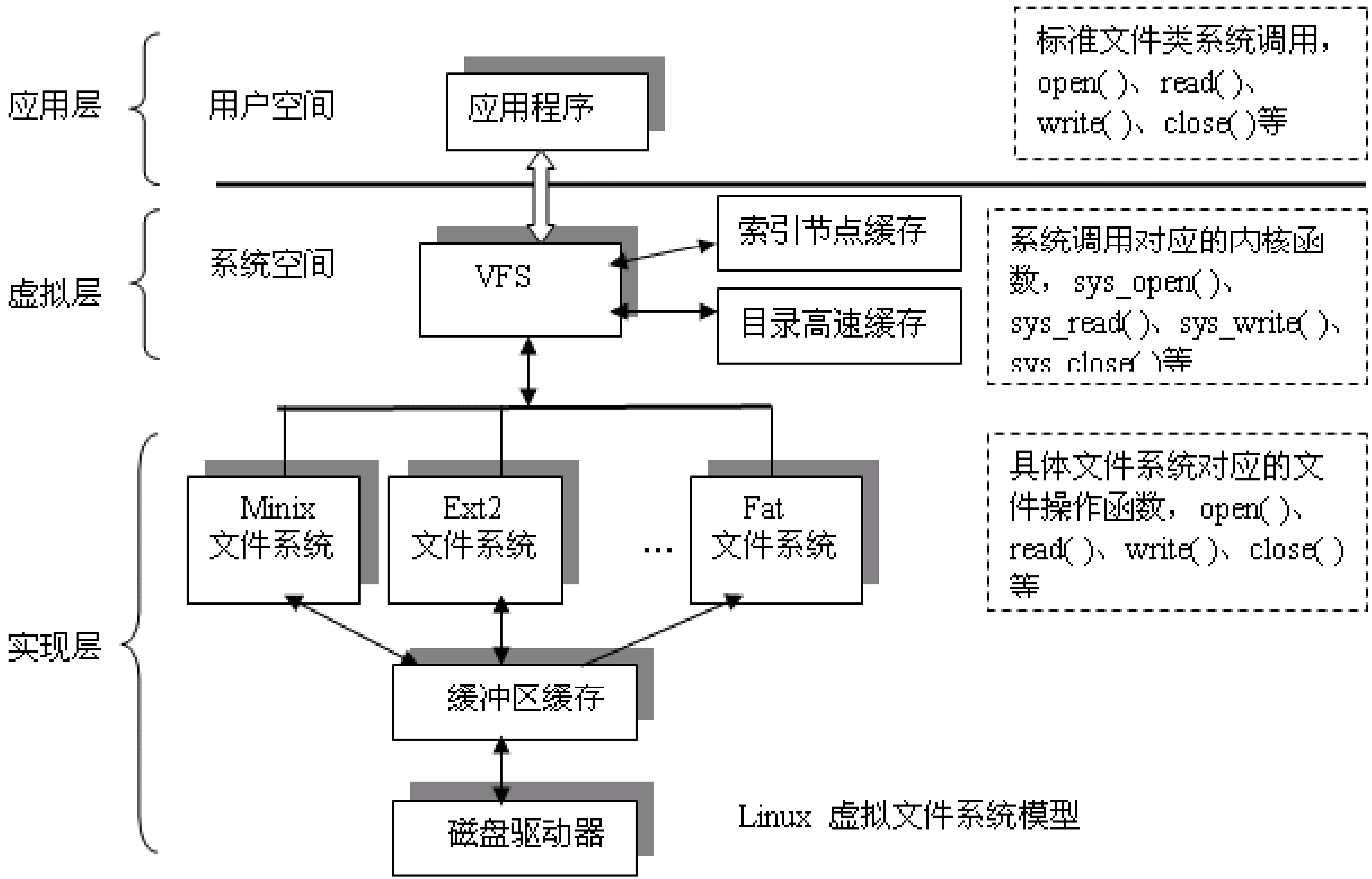
Schematic View of Virtual File System



Virtual File System

- VFS layer serves two functions:
  - It separates file-system-generic operations from their implementation by defining a clean VFS interface.
  - The VFS provides a mechanism for uniquely representing a file throughout a network, based on VNODE.
- 具体来说，VFS提供以下功能
  - •记录可用的文件系统类型；
  - •把文件系统与对应的存储设备联系起来；
  - •处理面向文件的通用操作；
  - •涉及具体文件系统的操作时，把它们映射到相关的具体文件系统。

Linux VFS



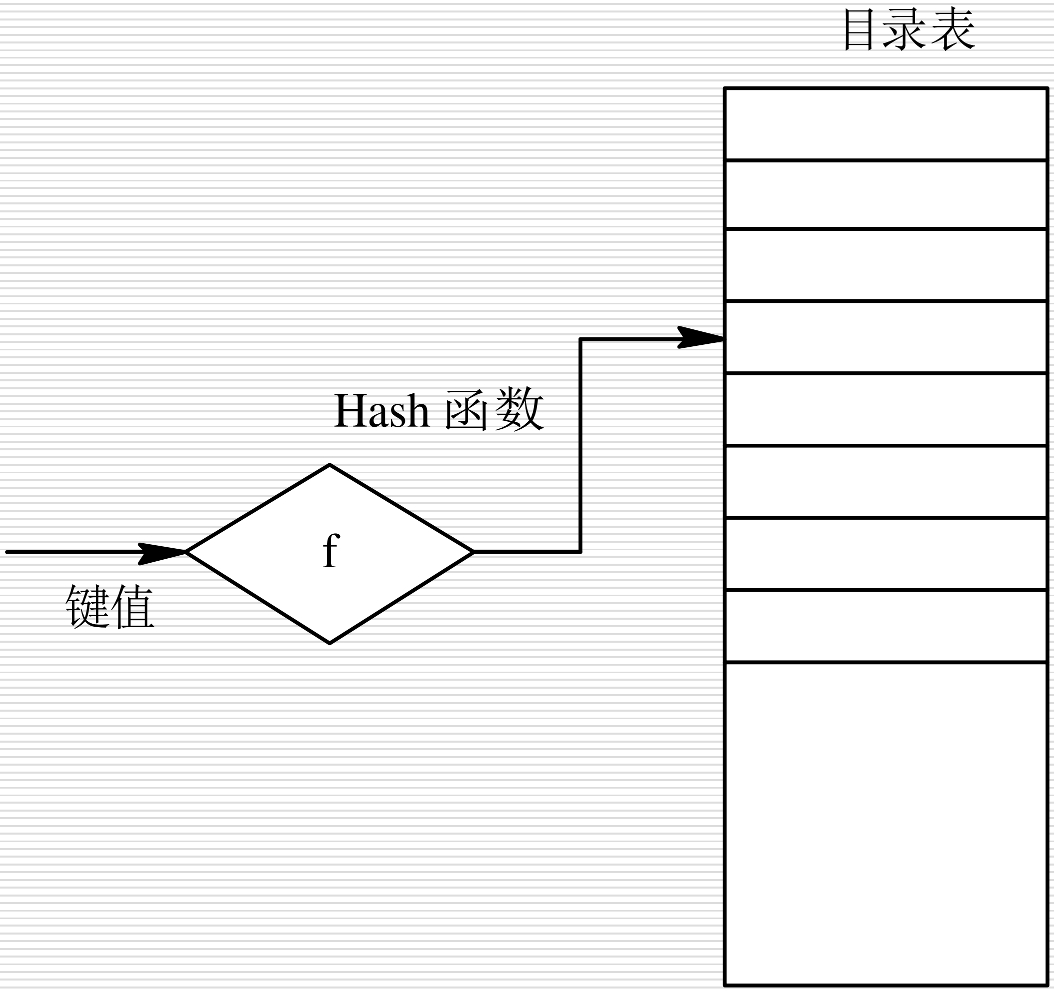
Linux 虚拟文件系统模型

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

Hash table

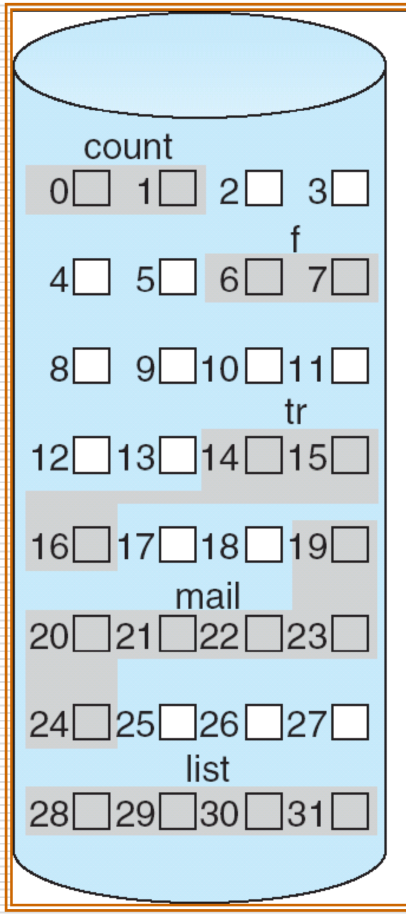


11.4 Allocation Methods

- 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



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

Contiguous Allocation

- Mapping from logical to physical

Block to be accessed = i + starting address

Contiguous Allocation of Disk Space

- Characteristic
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Random access
  - Wasteful of space (dynamic storage-allocation problem)
  - Files cannot grow

Extent-Based Systems

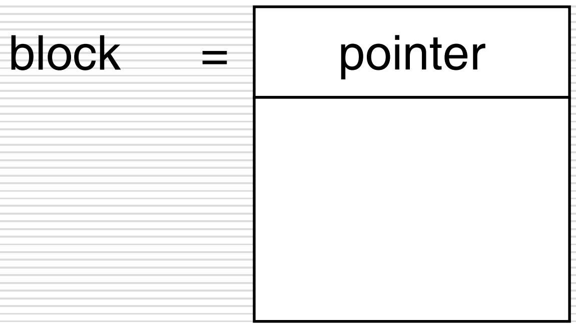
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme

- Extent-based file systems allocate disk blocks in **extents**

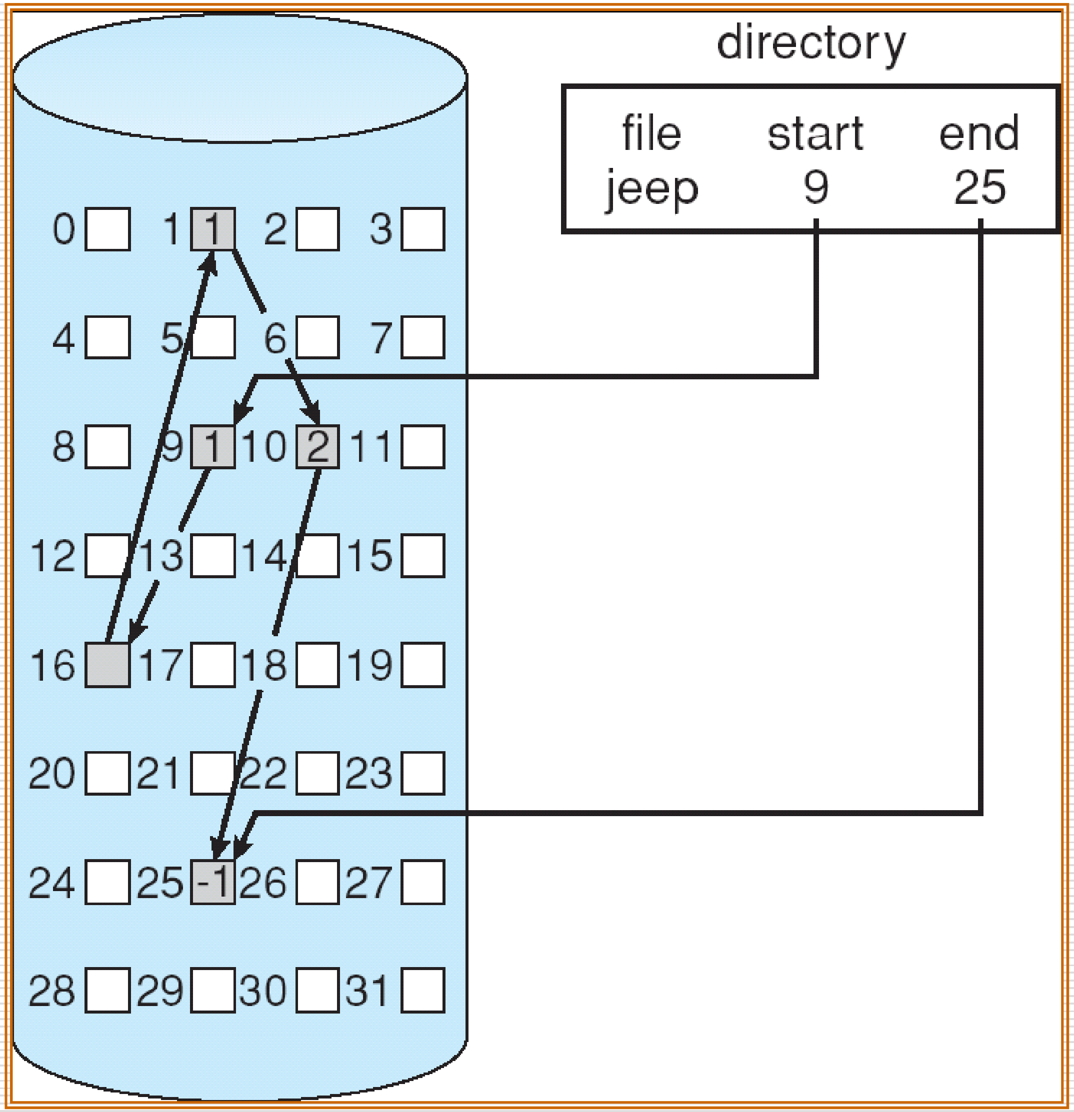
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents.

Linked Allocation

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



Linked Allocation



Linked Allocation (Cont.)

- Advantages：
  - Simple – need only starting address
  - Free-space management system – no waste of space
- Disadvantages
  - No random access
  - Pointers takes space
  - reliability
- Mapping

LA/511

R

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

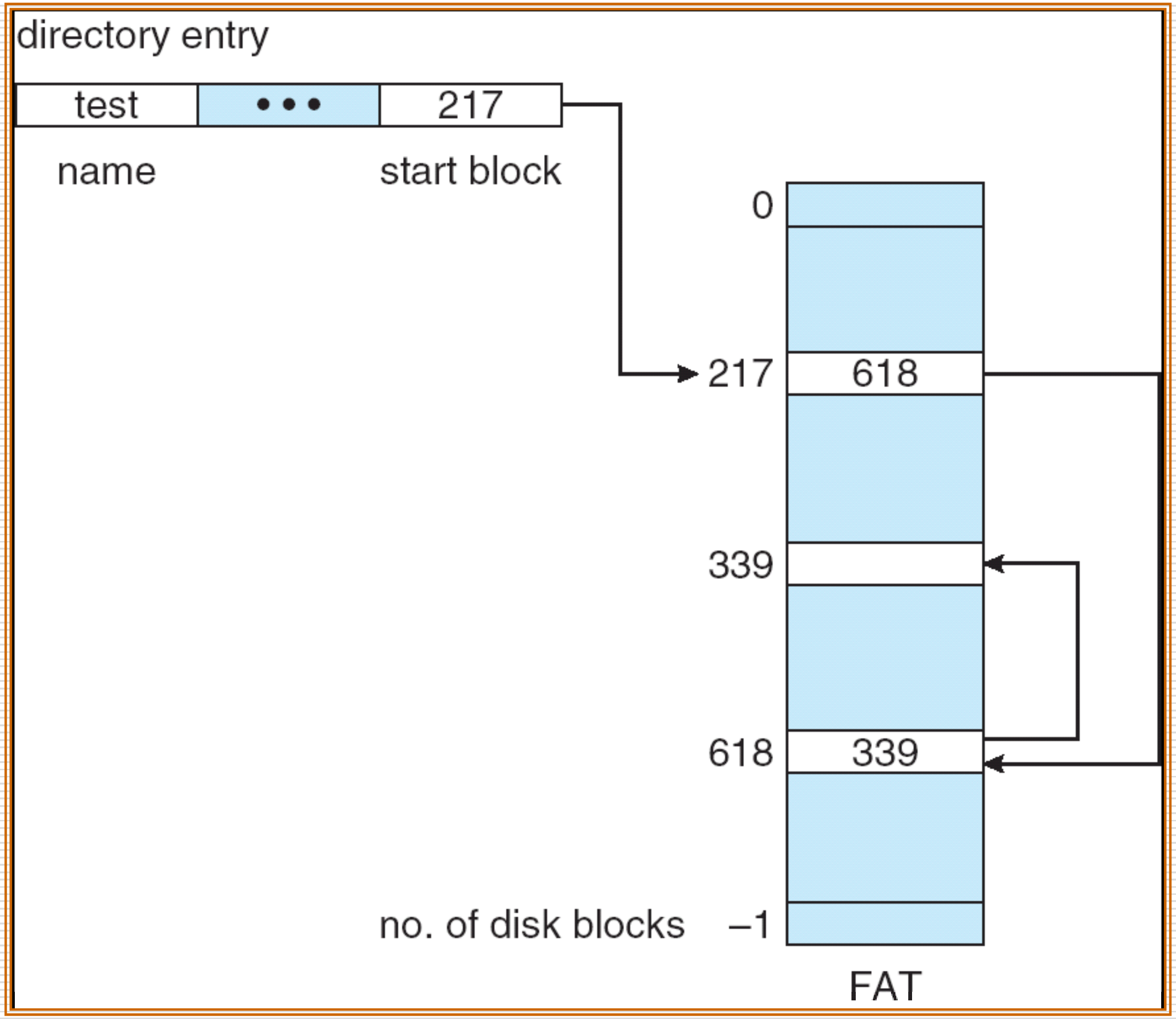
Displacement into block = R + 1

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

File-Allocation Table

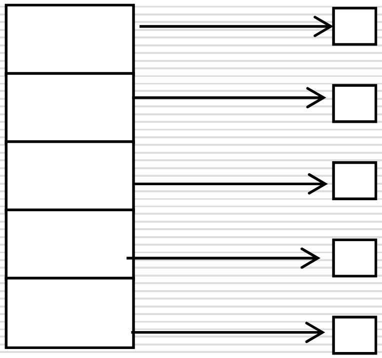
- 每个分区的开始部分用于存储该分区FAT表。
- 每个磁盘块在该表中有一项，该表可以通过块号来索引。
- 目录条目中含有文件首块的块号码。根据块号码索引的FAT条目包含文件下一块的块号码。这种链会一直继续到最后一块，该块对应FAT条目的值为文件结束值。未使用的块用0值来表示。
- 为文件分配一个新的块只要简单地找到第一个值为0的FAT条目，用新块的地址替换前面文件结束值，用文件结束值替代0。
- 如果不对FAT采用缓存，FAT分配方案可能导致大量的磁头寻道时间。但通过读入FAT信息，磁盘能找到任何块的位置，从而实现随机访问。

File-Allocation Table



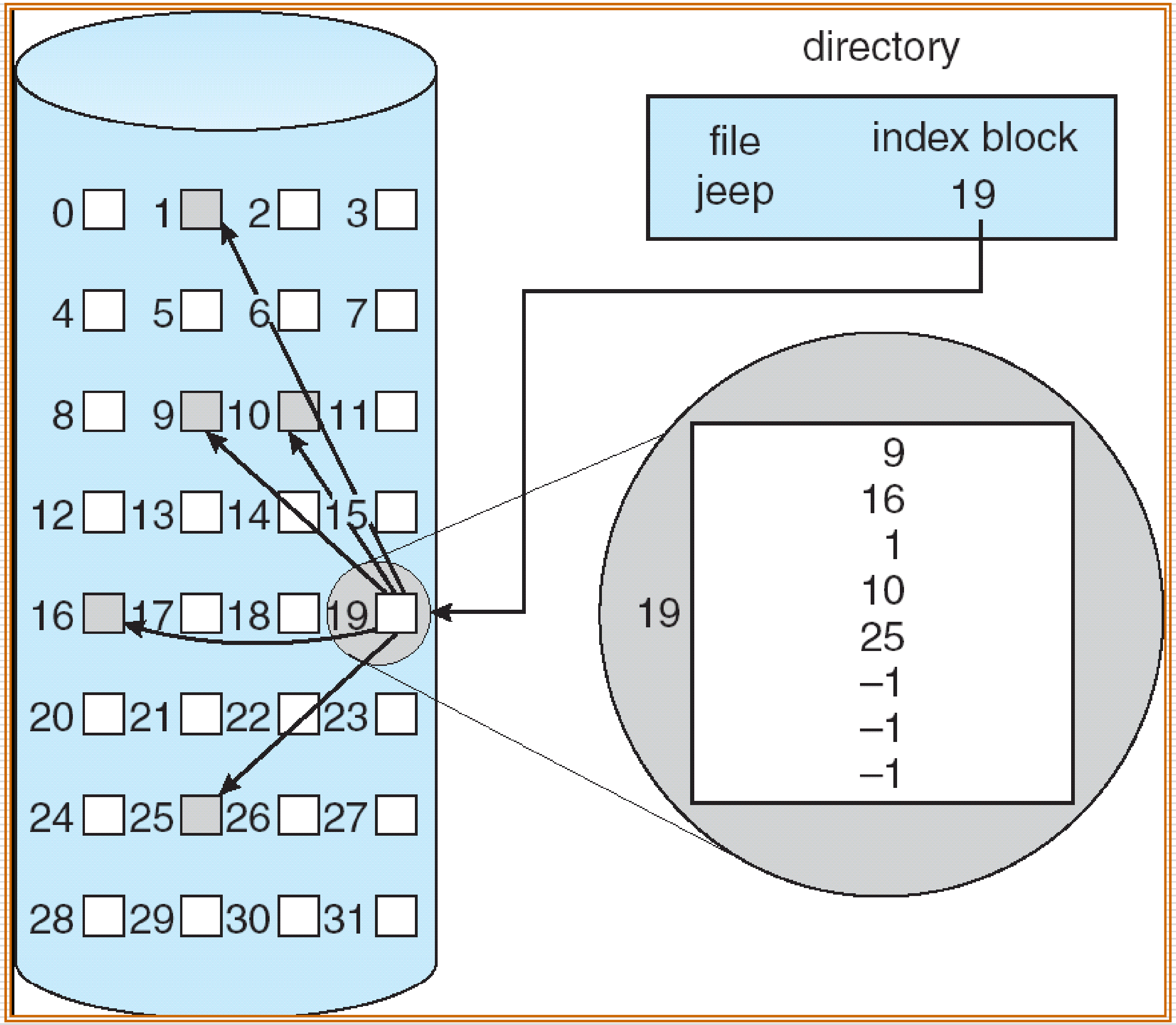
Indexed Allocation

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



index table

Example of Indexed Allocation



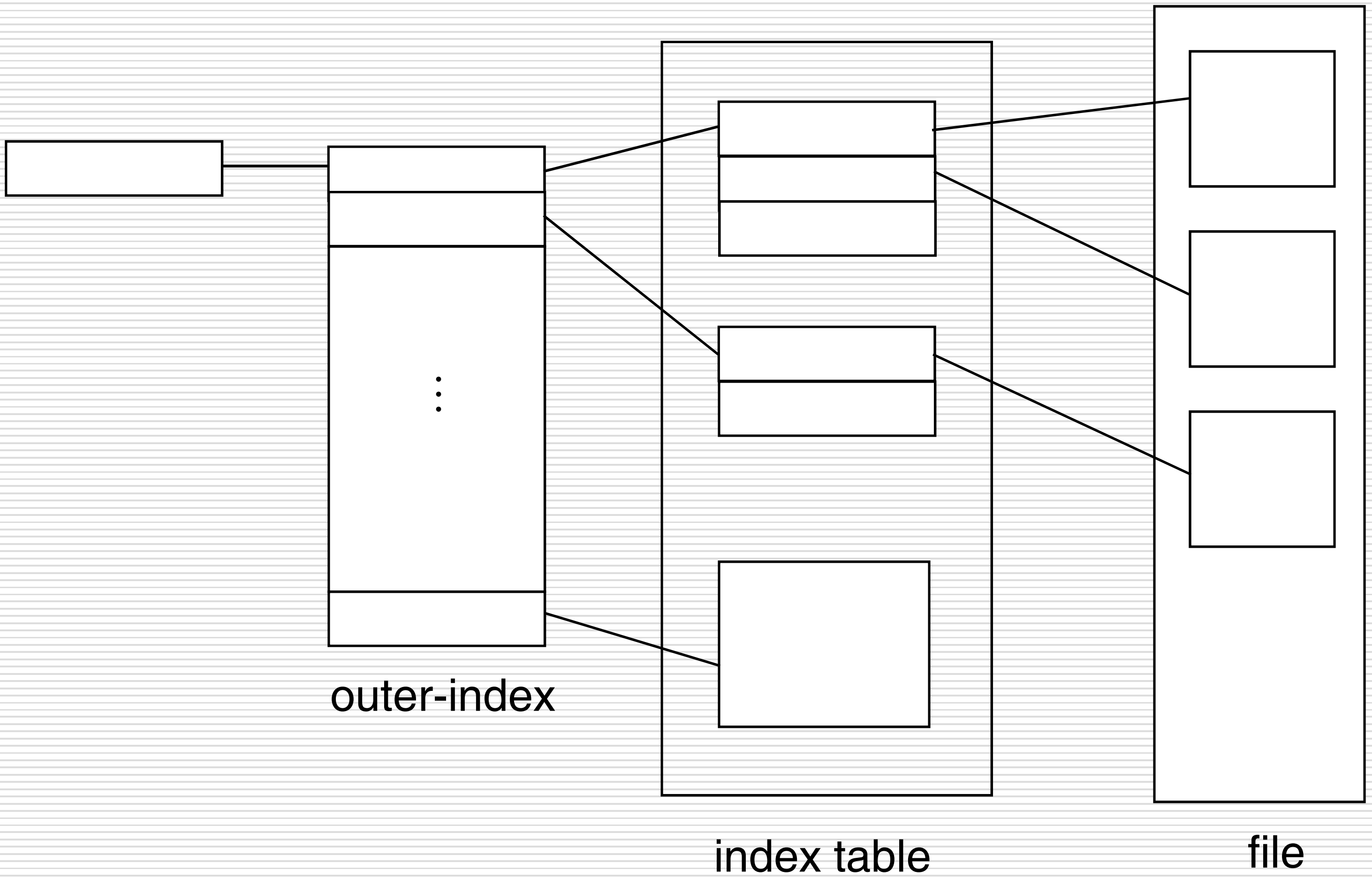
Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

Indexed Allocation – Mapping (Cont.)

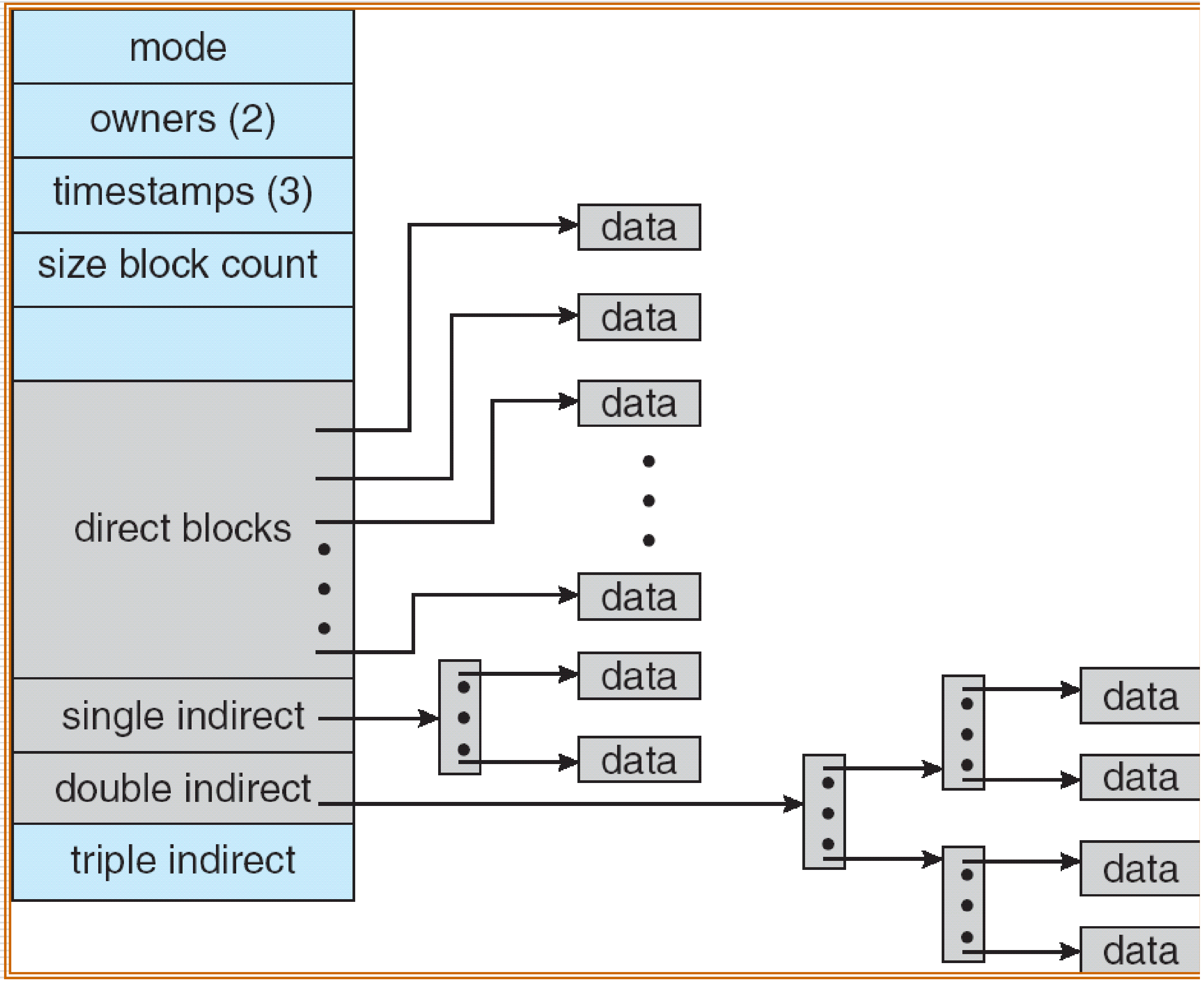
- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).
- Two-level index (maximum file size ？)

Indexed Allocation – Mapping (Cont.)



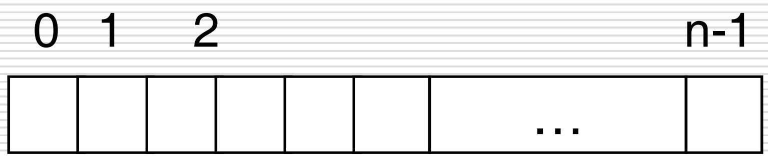
Combined Scheme: UNIX (4K bytes per block)

- 15 pointers
- 12 direct pointers
- 1 single indirect pointer
- 1 double indirect pointer
- 1 triple indirect pointer



11.5 Free-Space Management

- Bit vector (*n* blocks)



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

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

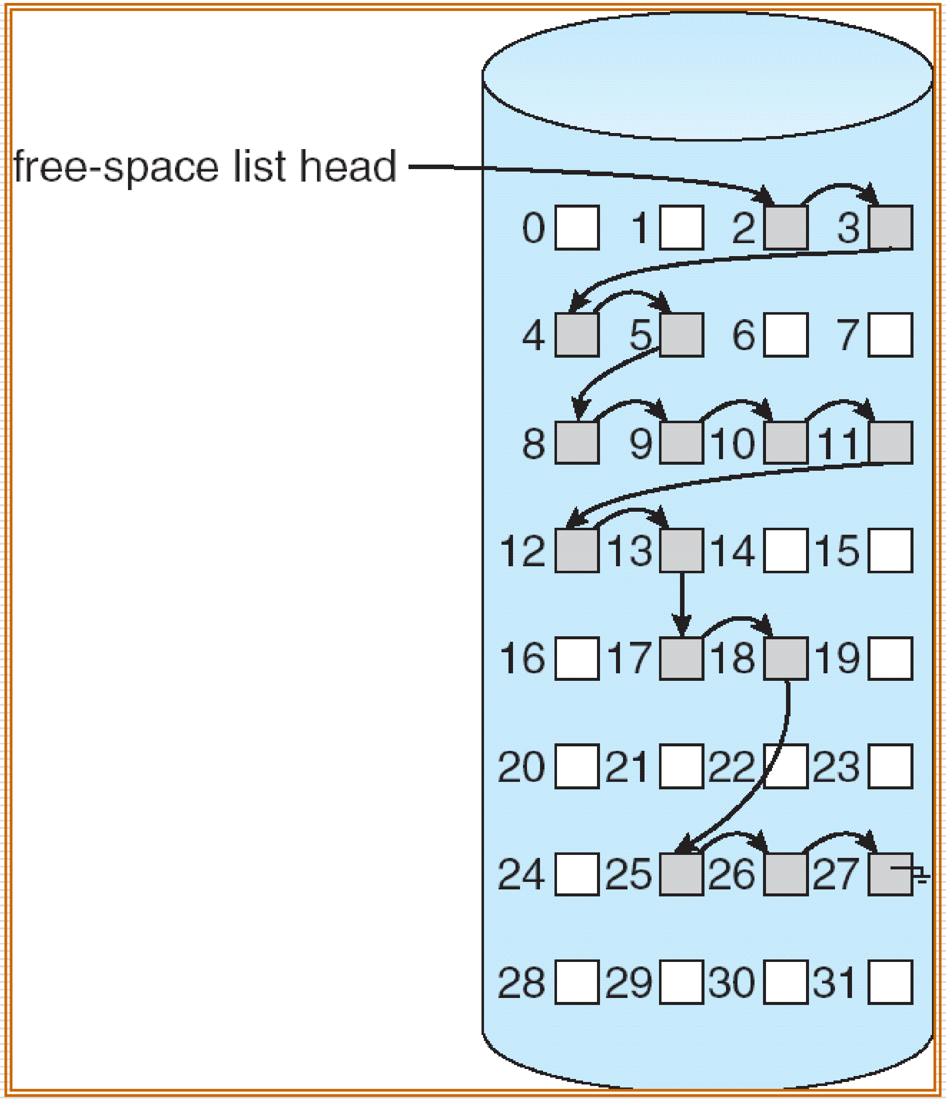
Free-Space Management (Cont.)

- Linked list (free list)
  - No waste of space
  - Cannot get contiguous space easily
  - Long time to find large number of free blocks
- Grouping
  - To store the addresses of n free blocks in the first block.
  - The last block contains the addresses of another n free blocks.
  - The addresses of large number of free blocks can be found quickly.
- Counting
  - To keep the address of the first block, and the number n of free contiguous blocks that follow the first one.

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 bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

Linked Free Space List on Disk



11.6 Efficiency and Performance

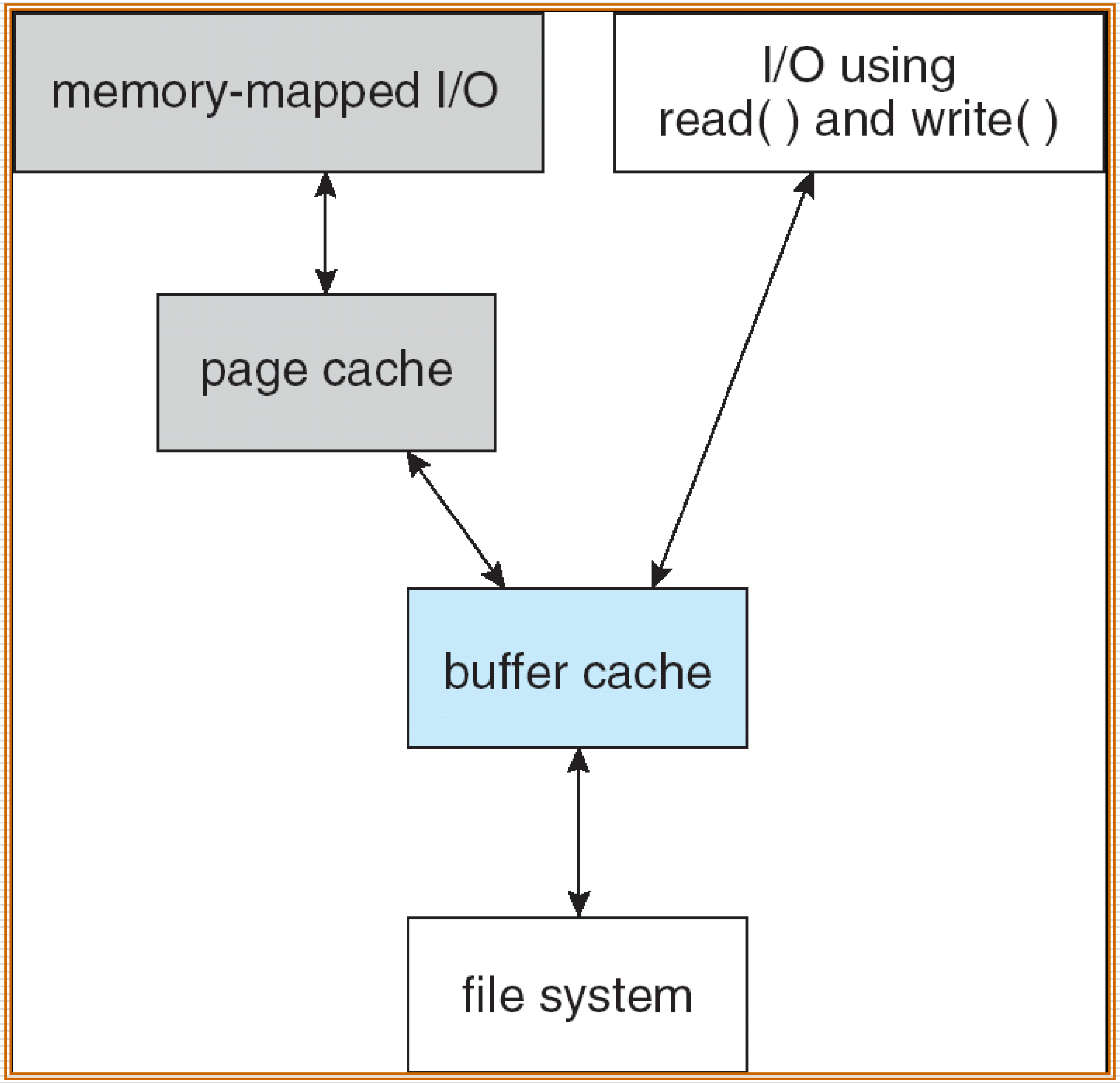
- Efficiency dependent on:
  - disk allocation and directory algorithms
    - i-node
  - types of data kept in file's directory entry
    - pointers

- 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

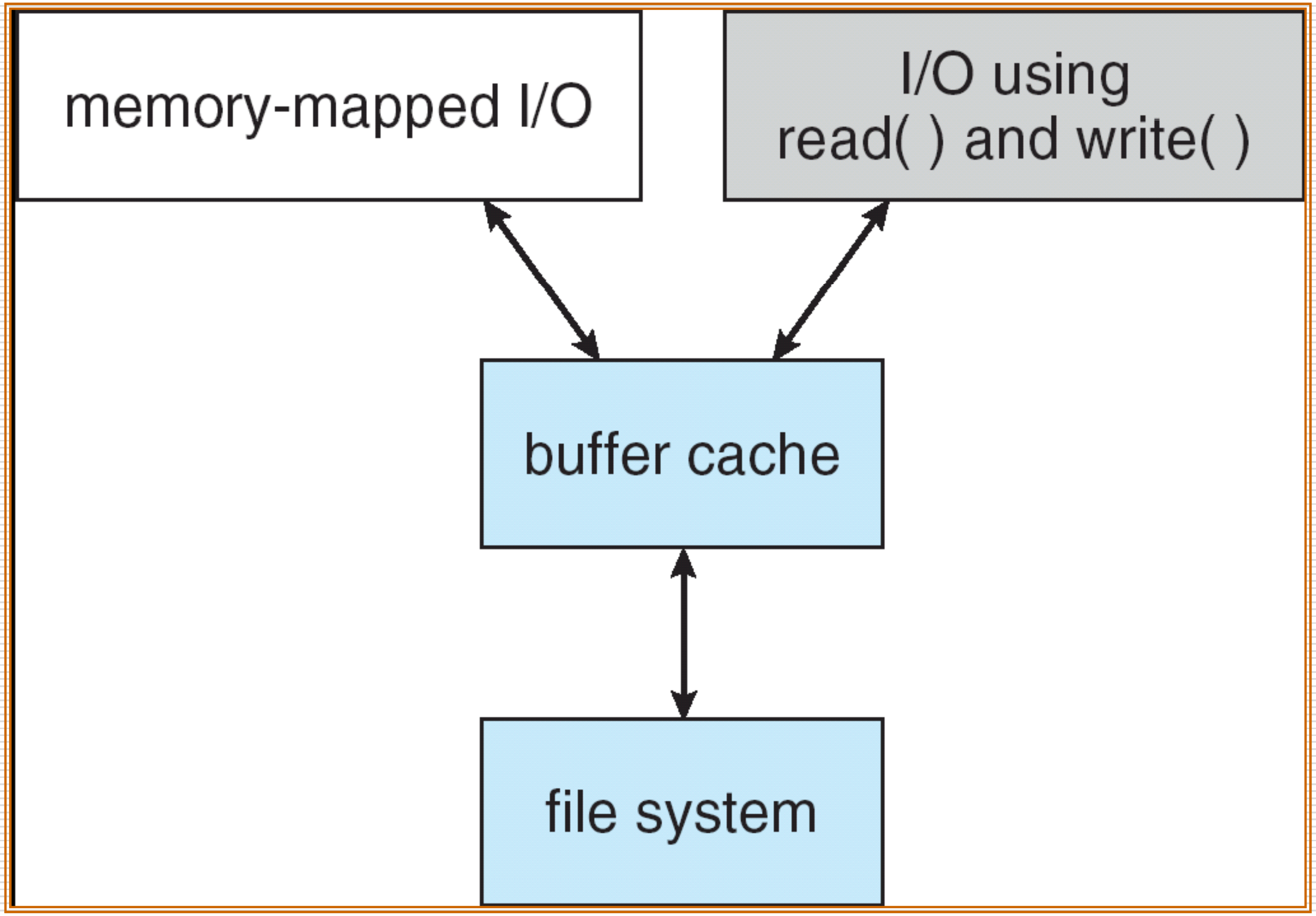
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- In Unix and Linux, 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



11.7 Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - UNIX—fsck
  - MS-DOS--chkdsk
- Use system programs to **back up** data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

11.8 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

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

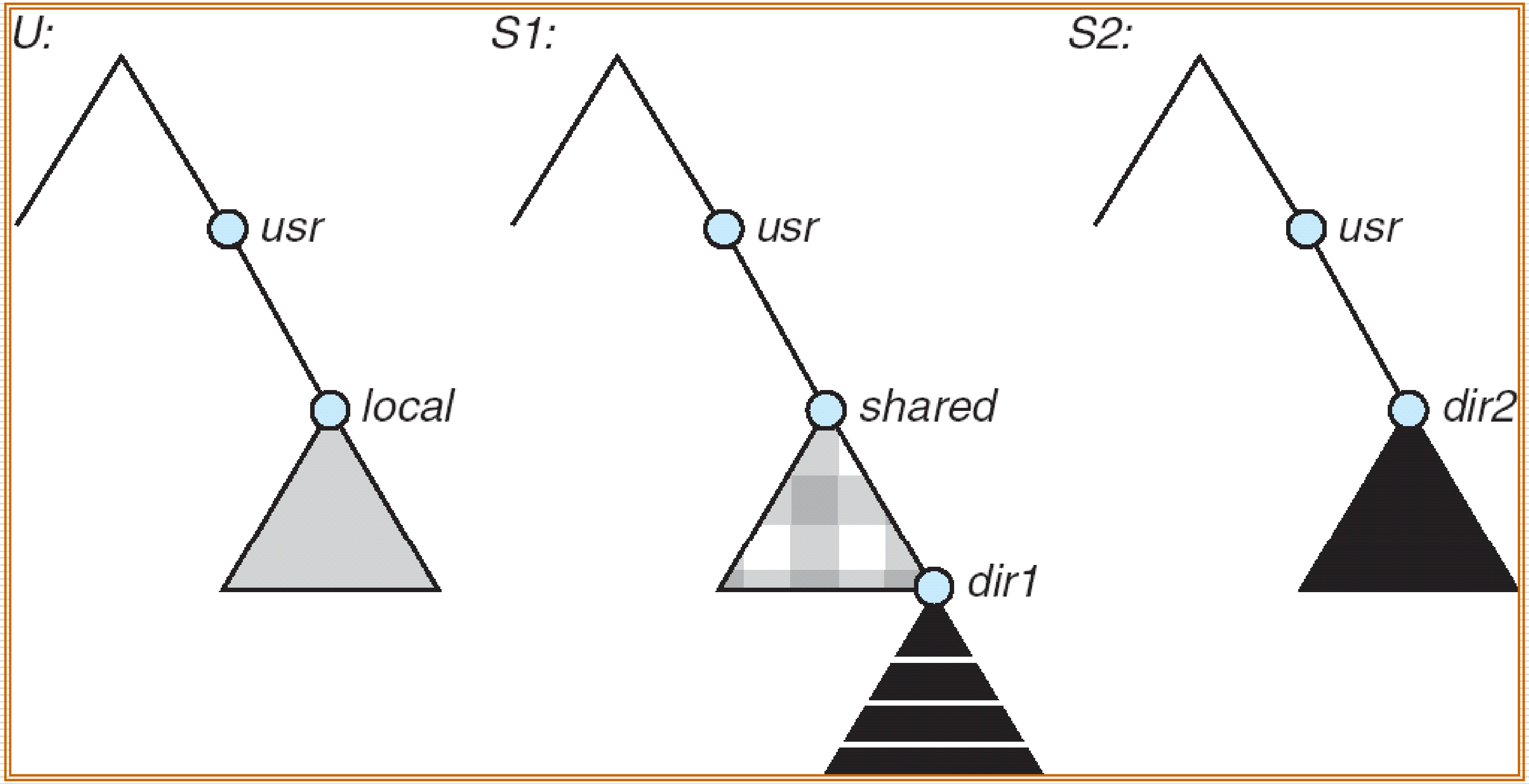
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media

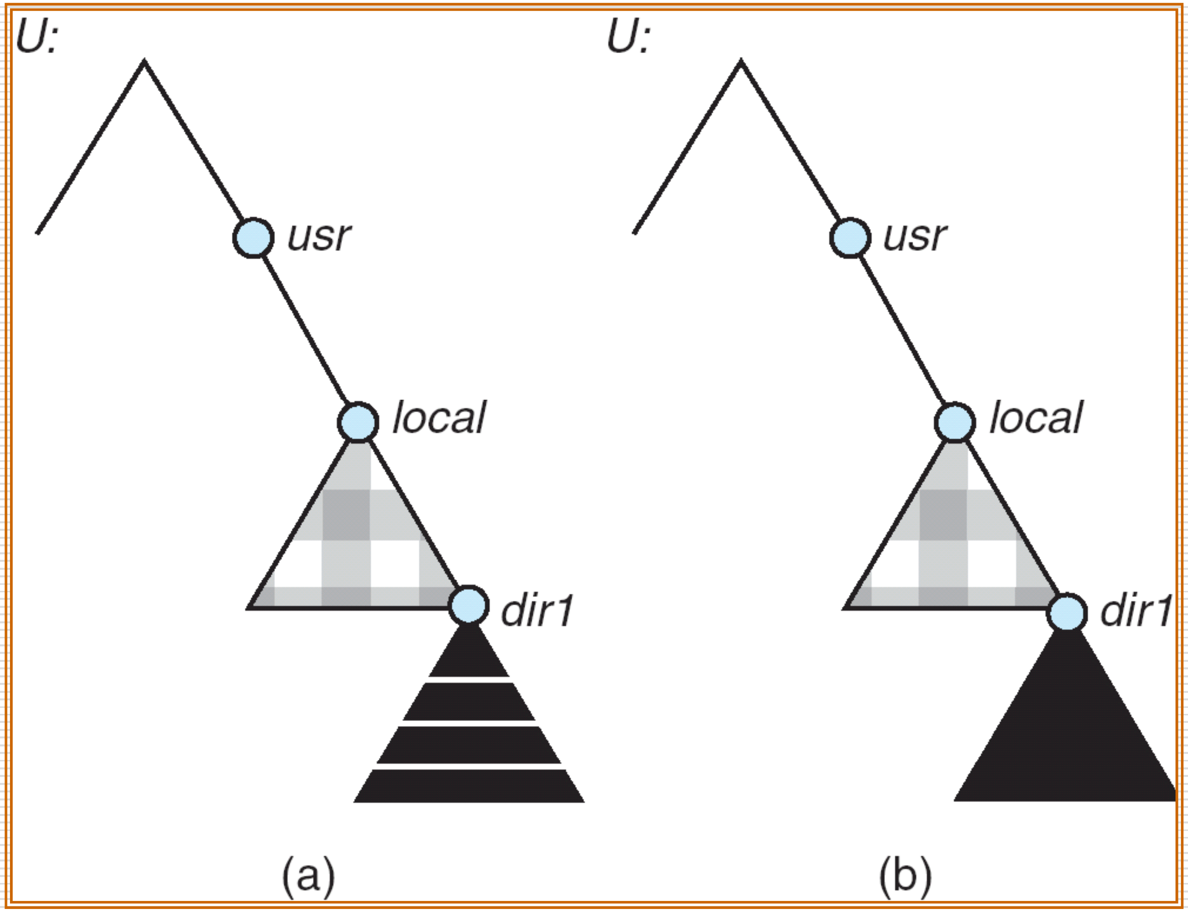
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

- 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



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 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

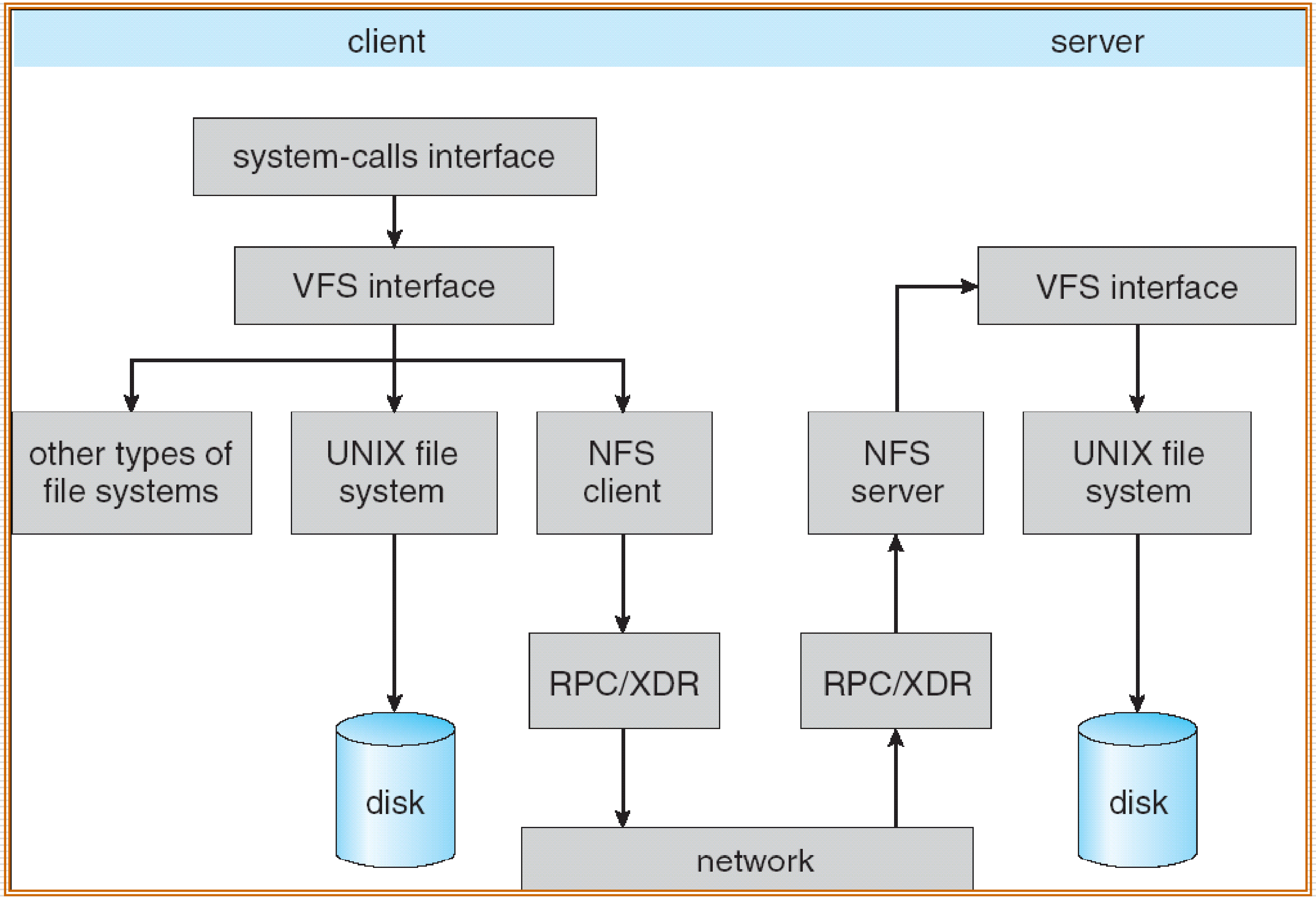
Three Major Layers of NFS Architecture

- 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

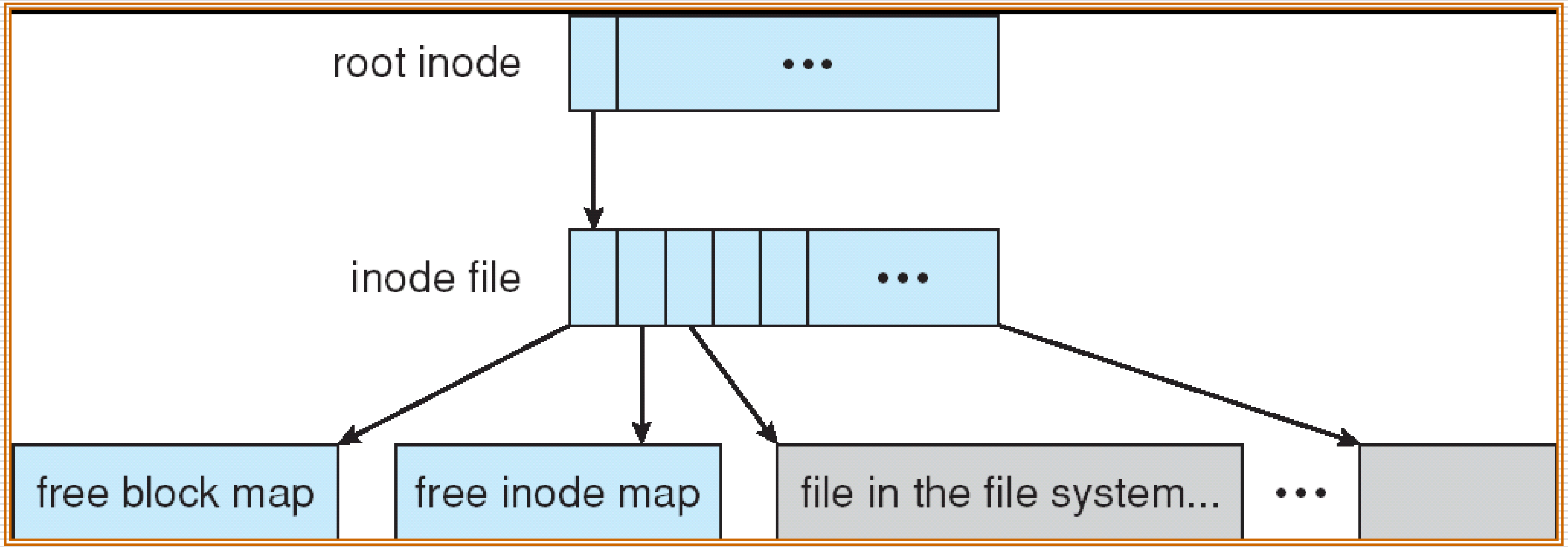
NFS Remote Operations

- 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

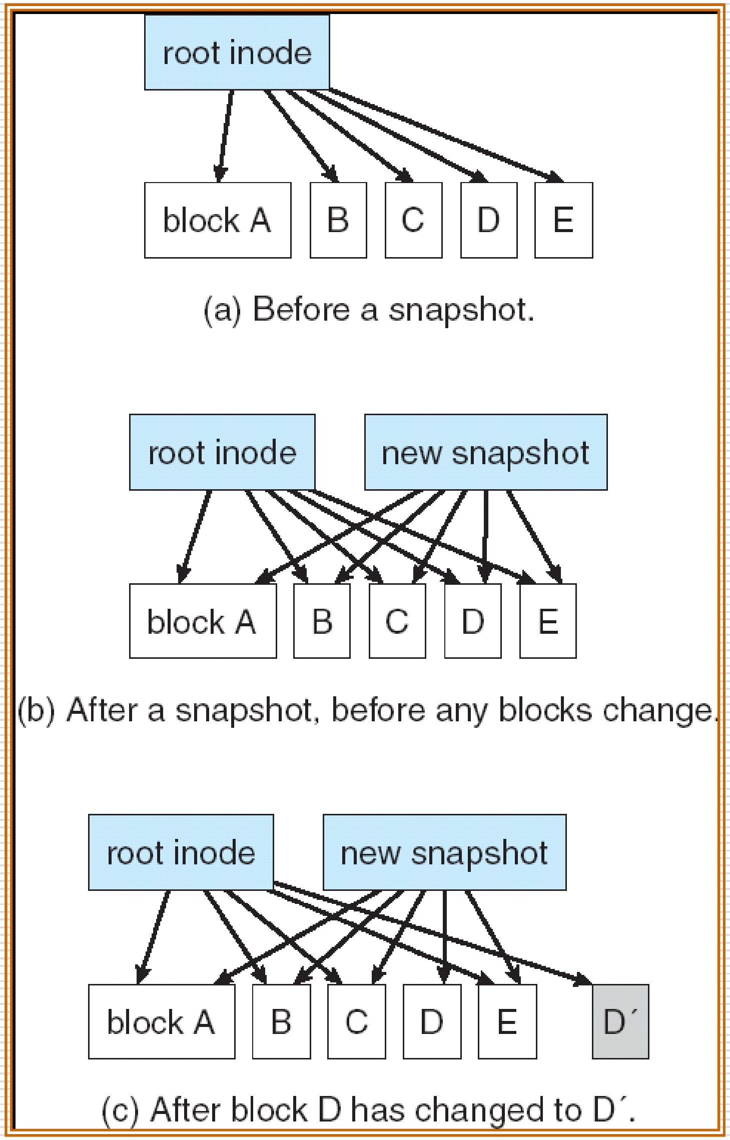
Example: WAFL File System

- 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



11.02

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

assignments

- 11.2 11.4 11.6

End of Chapter 11

Any Question?