Operating System Practice

Che-Wei Chang

chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

Advanced Operating System Concepts

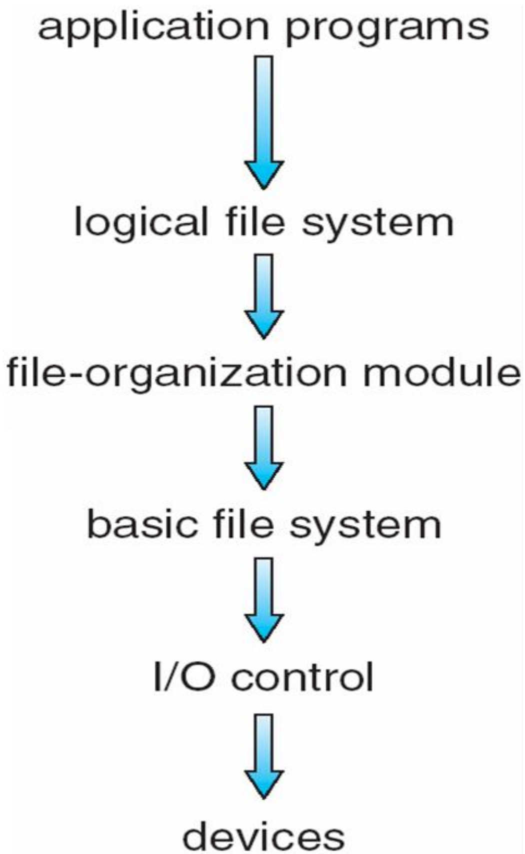
- Chapter 10: File System
- Chapter 11: Implementing File-Systems
- Chapter 12: Mass-Storage Structure
- Chapter 13: I/O Systems
- Chapter 14: System Protection
- Chapter 15: System Security

Study Items

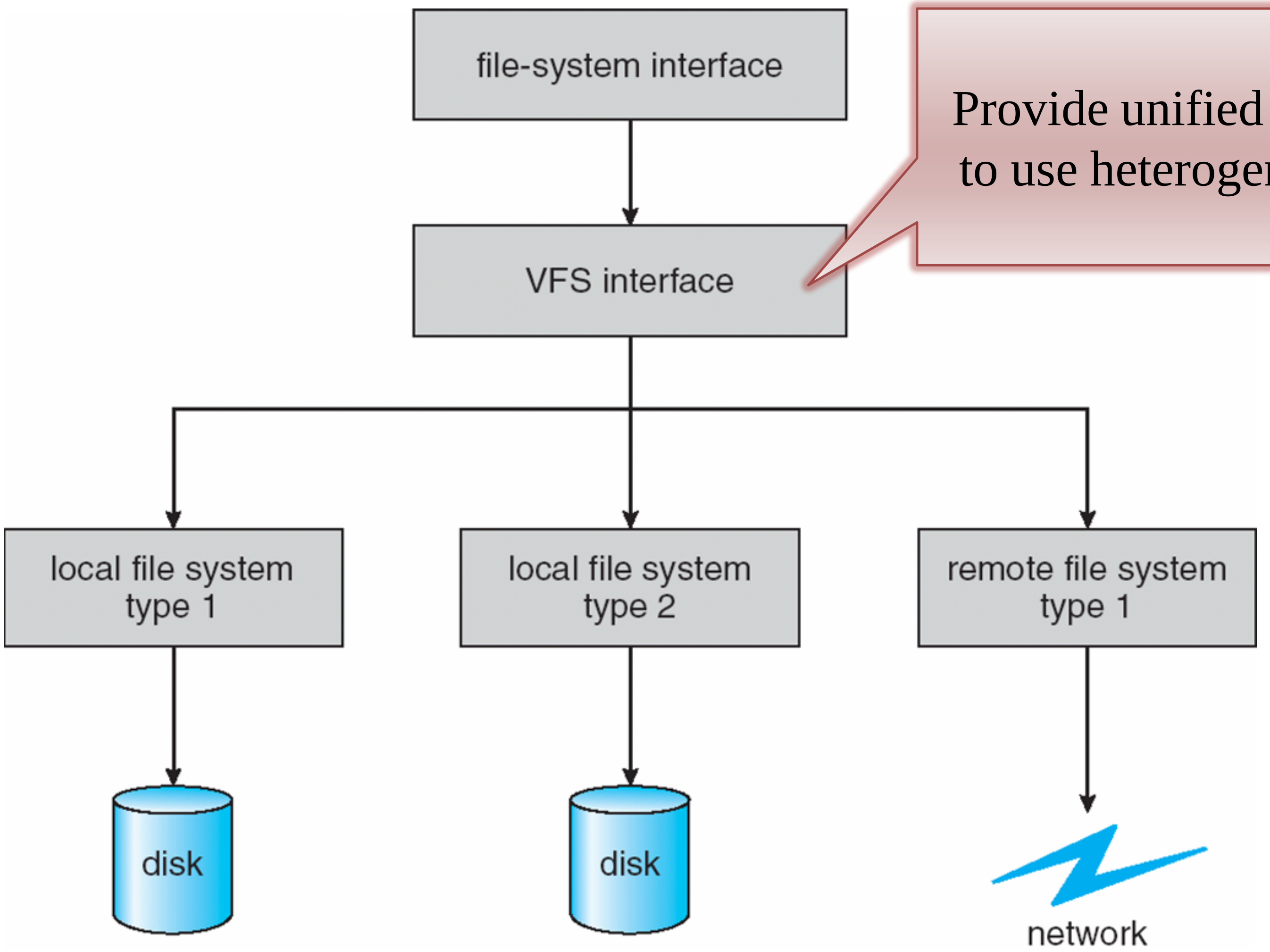
- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- NFS

File System Layers

- Logical File System: manage metadata information
  - Translate file name into file number and file handle by maintaining file control blocks (**i-nodes** in Unix)
  - Directory management
  - Protection
- File-Organization Module: understand files, logical address, and physical blocks
  - Translate logical block number to physical block number
  - Manage free space, disk allocation
- Basic File System: translate generic commands for device drivers
- I/O Control: translate commands into hardware instructions



Virtual File System



Provide unified interface for users to use heterogeneous file systems

File-System Implementation

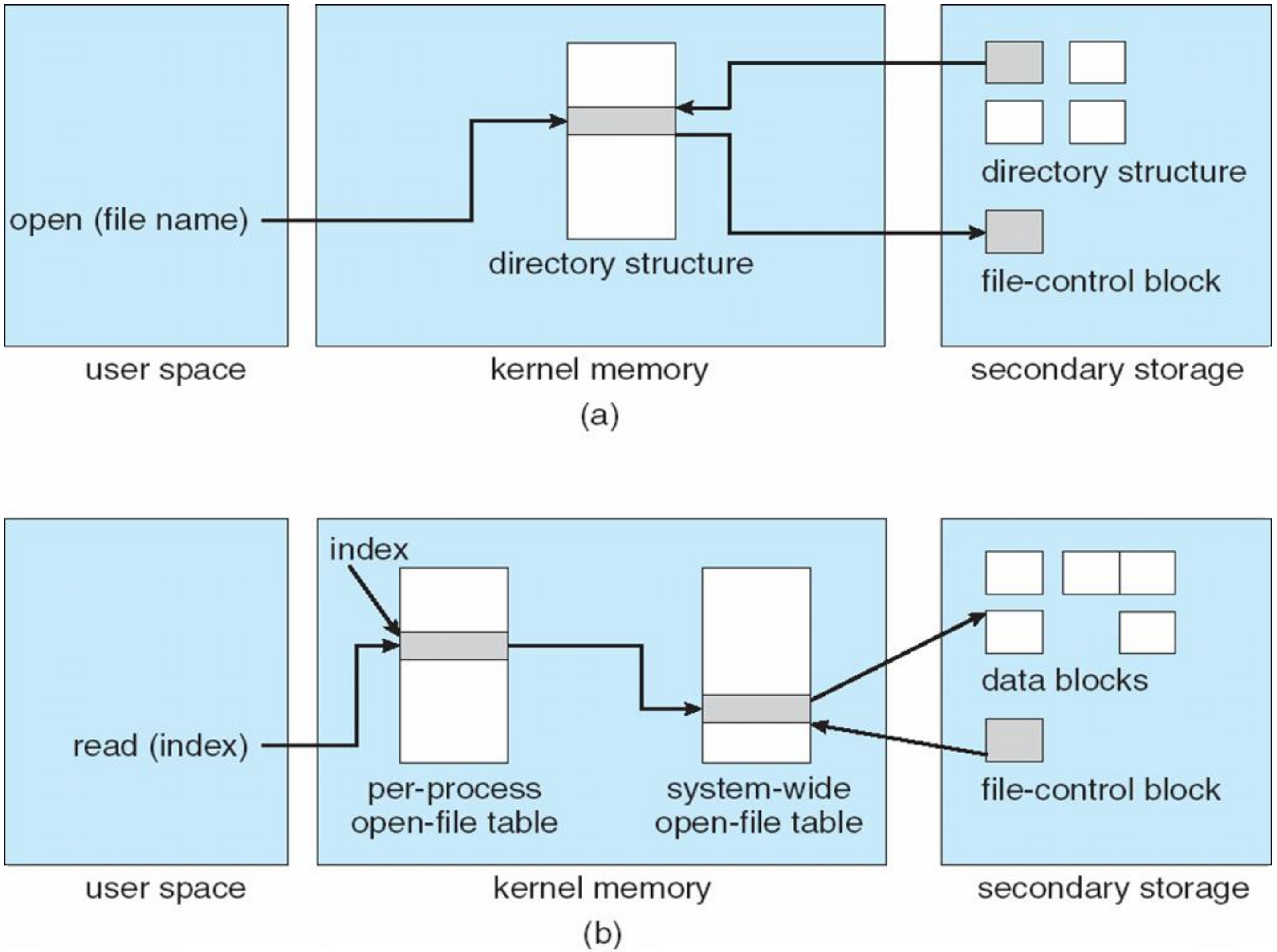
- Boot control block contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- Volume control block (superblock, master file table) contains volume details
  - Total number of blocks, number of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table
- Per-file File Control Block (FCB) contains many details about the file
  - i-node number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

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

ACL: Access Control List

In-Memory File System Structures



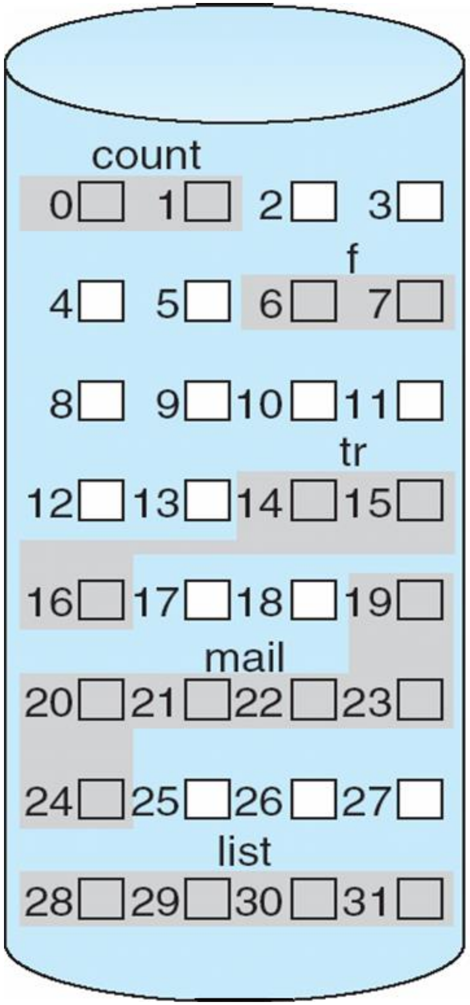
Directory Implementation

- Linear List: file names with pointers to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use tree structure
- Hash Table: linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location

Allocation Methods

- Contiguous Allocation – each file occupies set of contiguous blocks
- Linked Allocation – each file a linked list of blocks
- Indexed Allocation – each file has its own index block(s) of pointers to its data blocks

Contiguous Allocation Scheme



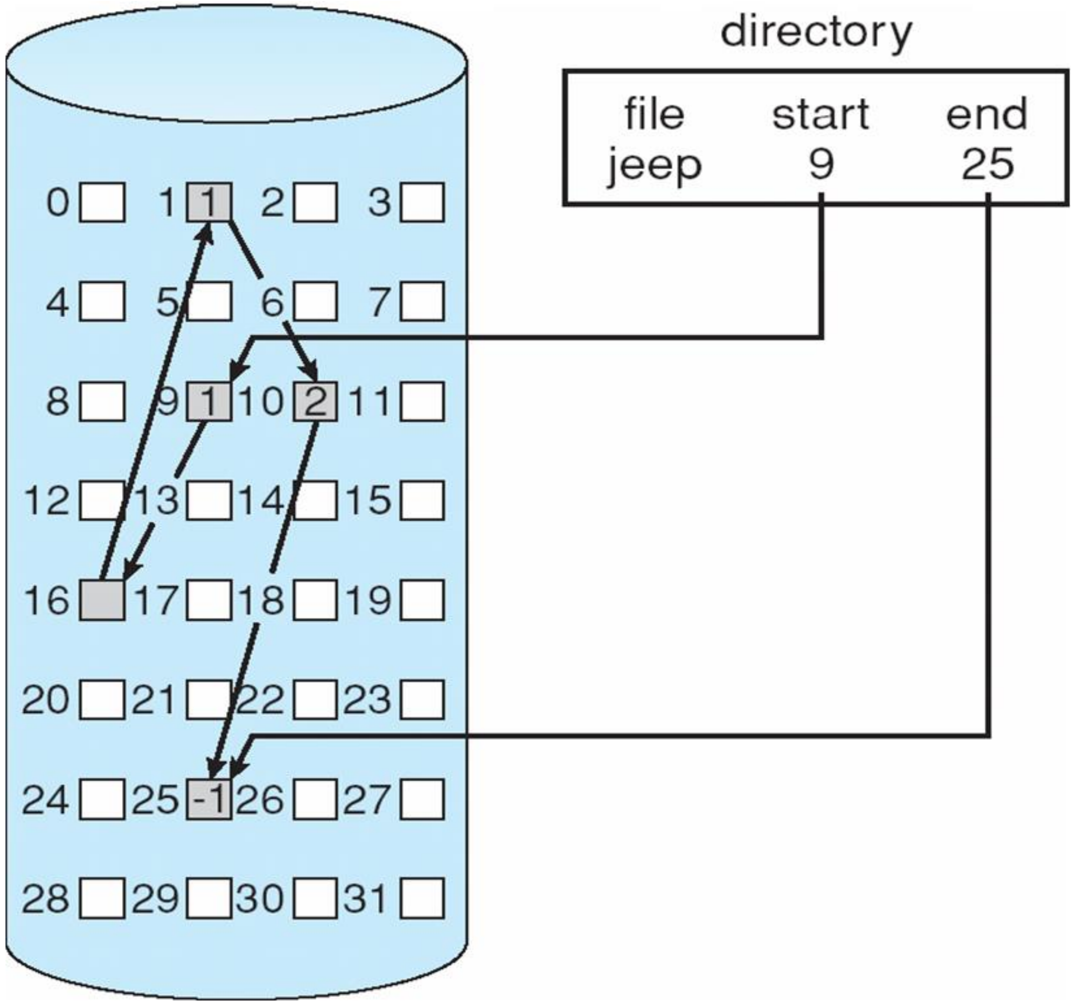
directory

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

Contiguous Allocation

- Best performance in most cases
- Simple – only starting location (block number) and length (number of blocks) are required
- Problems include finding space for file, knowing file size, external fragmentation, need for compaction
- Extent-based file systems allocate disk blocks in extents
  - Extents are allocated for extra space of file allocation
  - A file consists of one or more extents

Linked Allocation Scheme

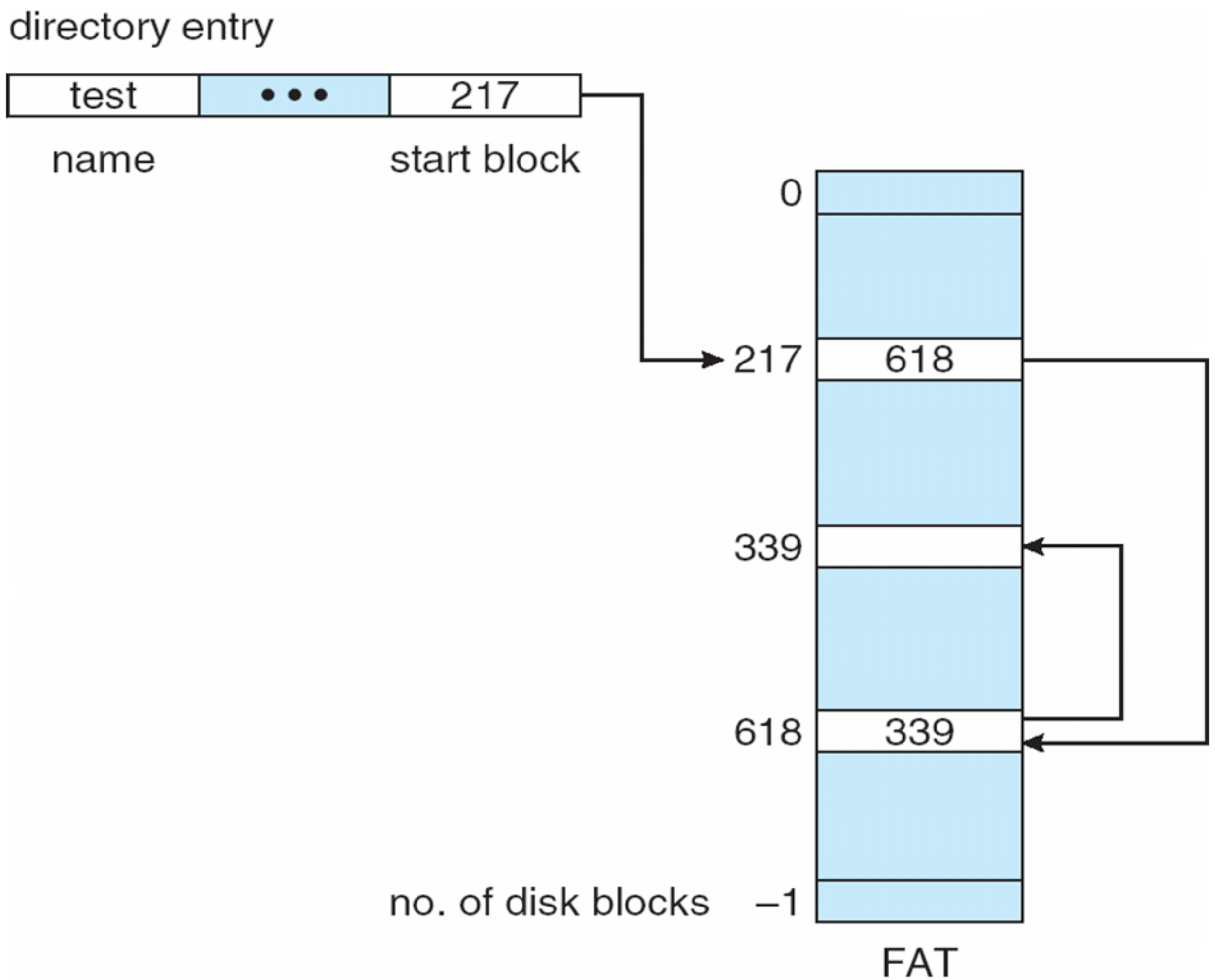


實樸勞勤

Linked Allocation

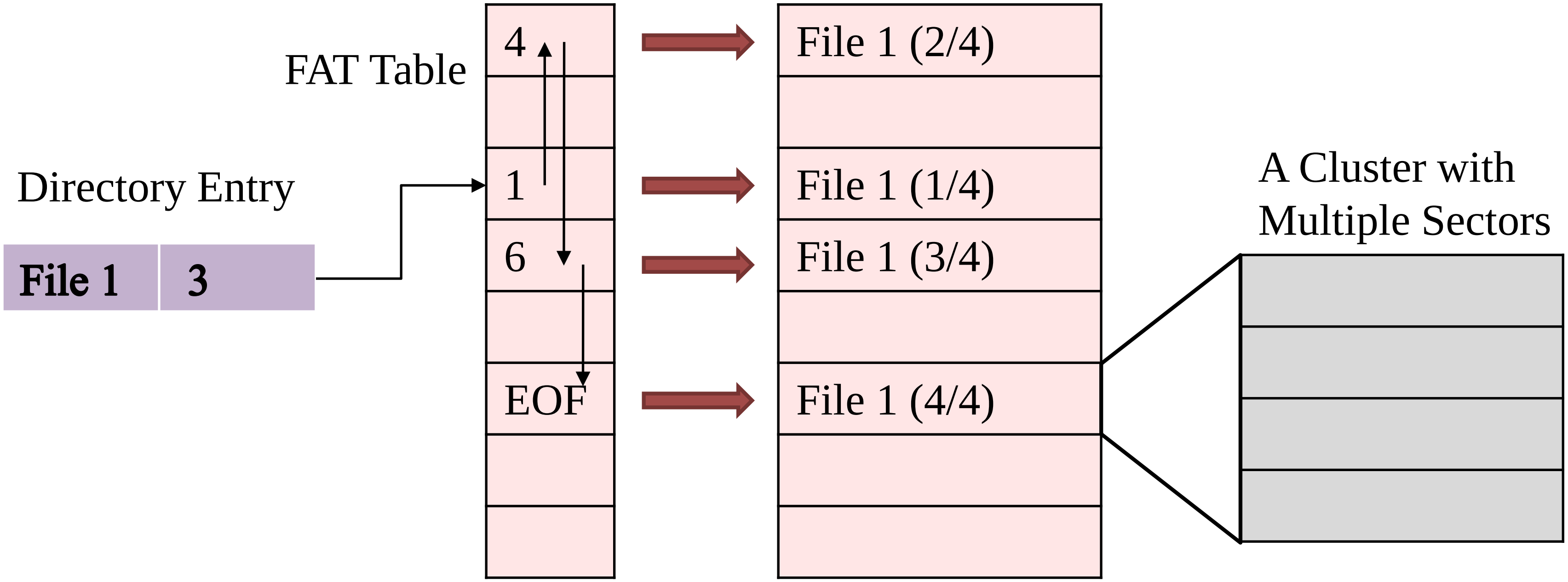
- Allocation Method
  - Each block contains pointer to next block
  - File ends at nil pointer
  - No external fragmentation
  - No compaction
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/O operations and disk seeks
- FAT (File Allocation Table) Variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

File-Allocation Table (1/2)

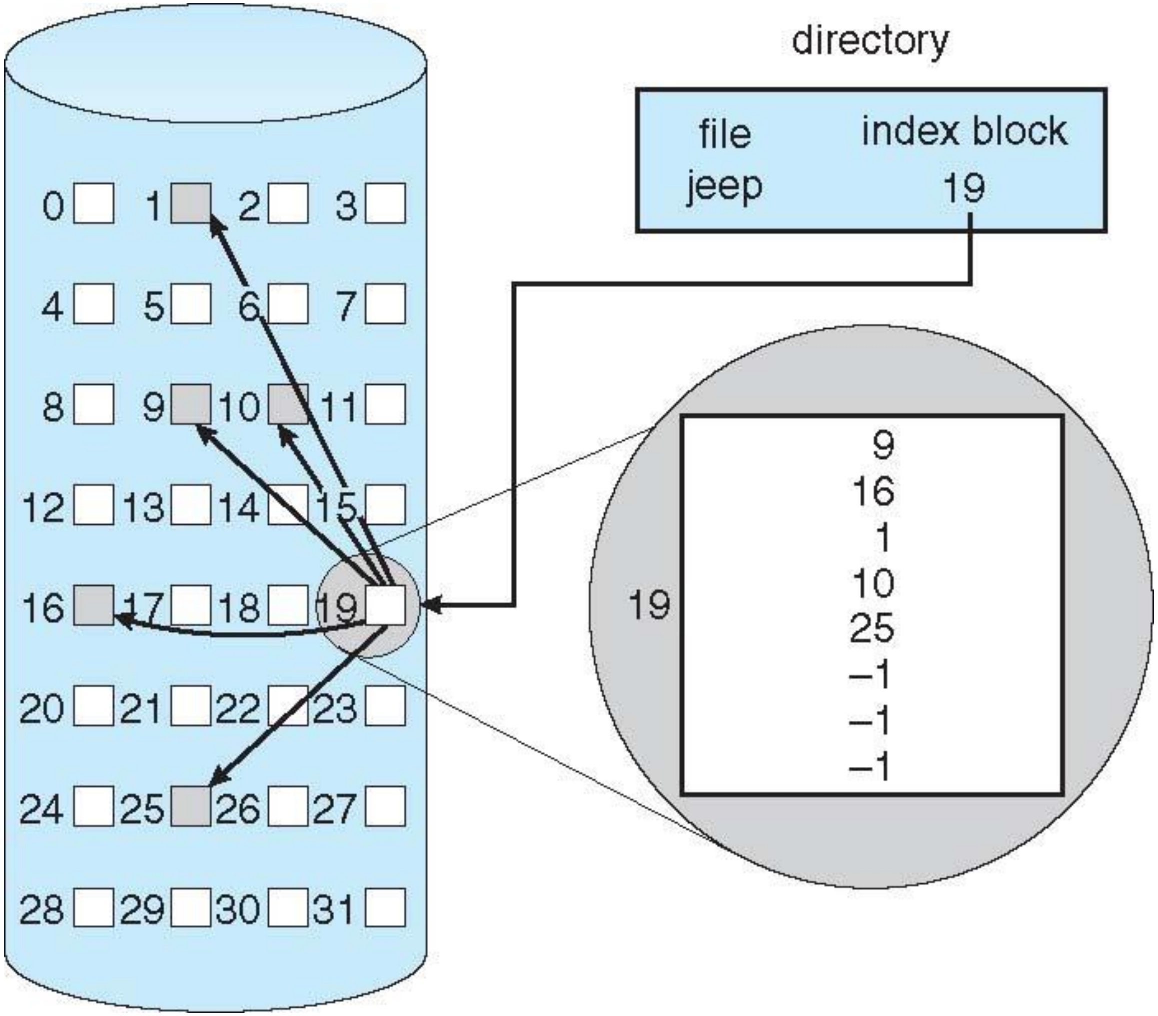


File-Allocation Table (2/2)

- File Allocation Table (FAT) is a computer file system architecture
- There are FAT12, FAT16, FAT32, and EXFAT



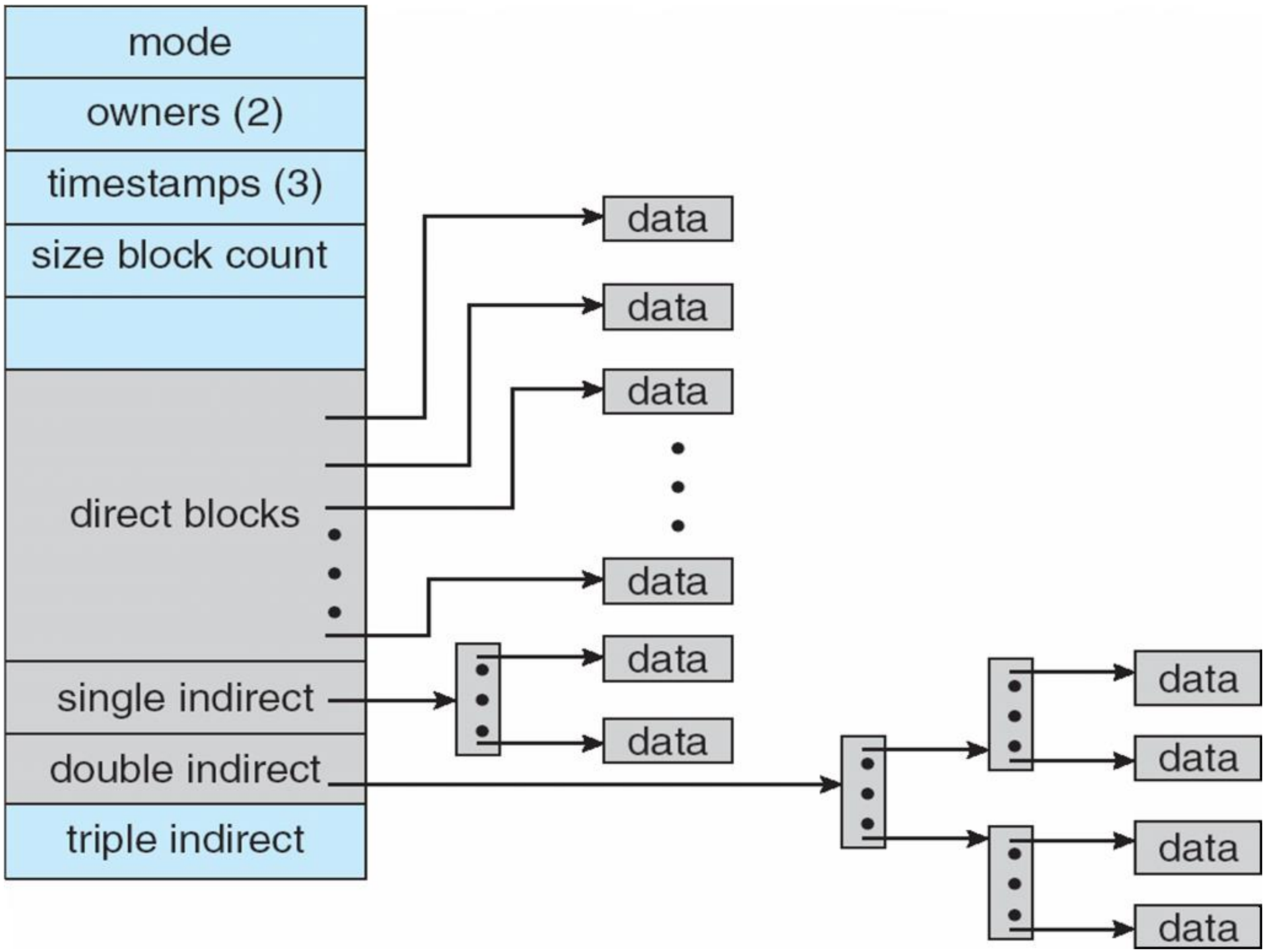
Indexed Allocation Scheme



Indexed Allocation

- Need index table
- Good at random access
- Dynamic access without external fragmentation, but have overhead of index block
- If more than one index block is required
  - Linked scheme
  - Multilevel index
  - Combined scheme

Combined Scheme: UNIX UFS



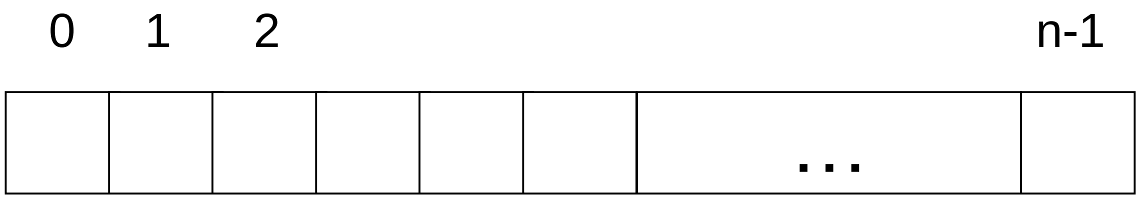
Department of Computer Science and Information Engineering, Chang Gung University

Free Space Management— Bit Map

- Bit map requires extra space
  - Example:

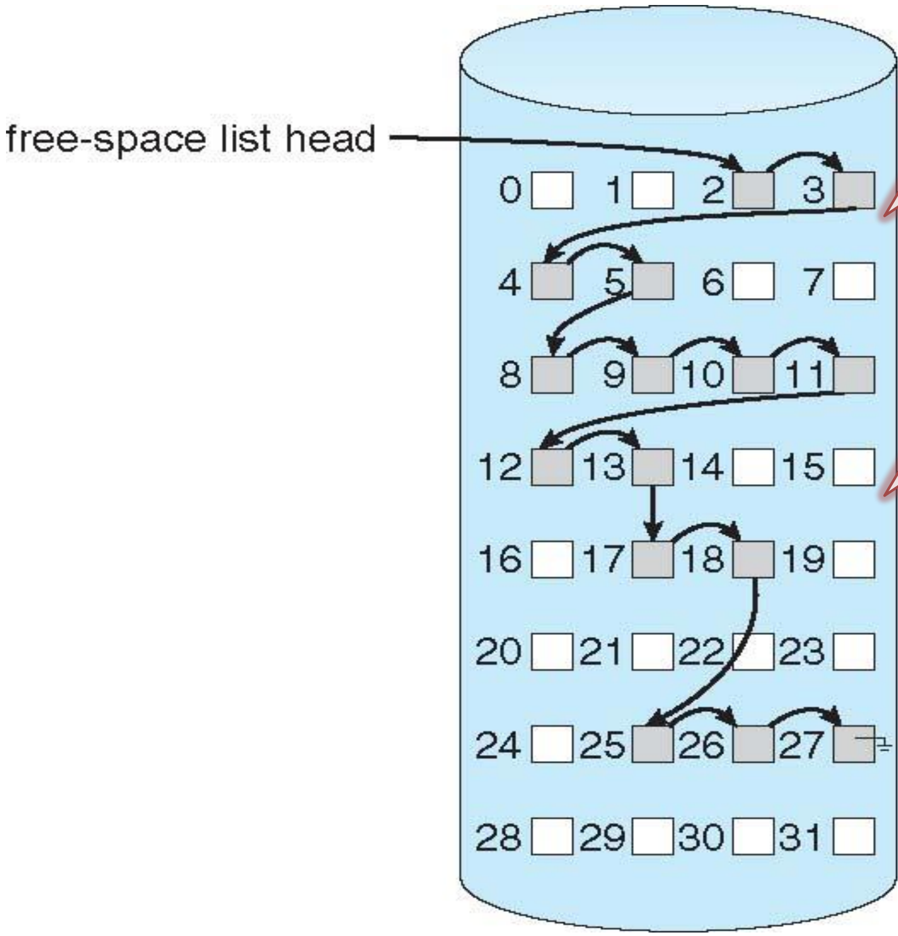
    block size = 4KB = 2^{12} bytes
    disk size = 2^{40} bytes (1 terabyte)
    $n = 2^{40}/2^{12} = 2^{28}$ bits (32 MB)
    if clusters of 4 blocks -> 8 MB of memory

- Easy to get contiguous files



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

Free Space Management—Linked List



Cannot get contiguous space easily

No waste of space

Free Space Management Methods

- 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

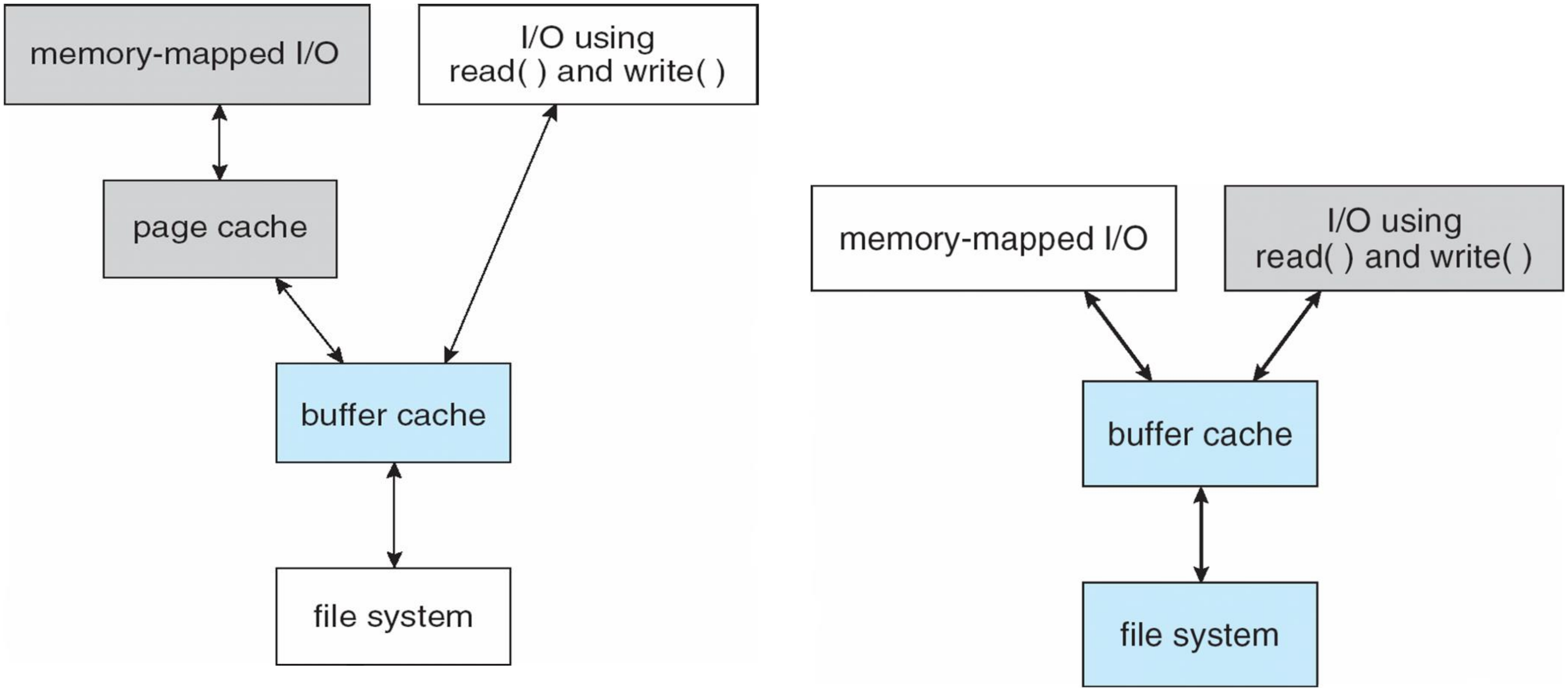
Efficiency and Performance

- Efficiency Considerations
  - 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
- Performance Considerations
  - 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 must hit disk before acknowledgement
    - Asynchronous writes more common, buffer-able, faster
  - Free-behind and read-ahead – techniques to optimize sequential access
  - Reads frequently slower than writes (for harddisk)

Performance Issue

- Adding instructions to the execution path to save one disk I/O is reasonable
  - Intel Core i7 Extreme Edition 990x at 3.46Ghz = 159,000 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - Typical disk drive at 250 I/O operations per second
    - 159,000 MIPS / 250 = 630 million instructions during one disk I/O
  - Fast SSD drives provide 60,000 IOPS
    - 159,000 MIPS / 60,000 = 2.65 millions instructions during one SSD I/O

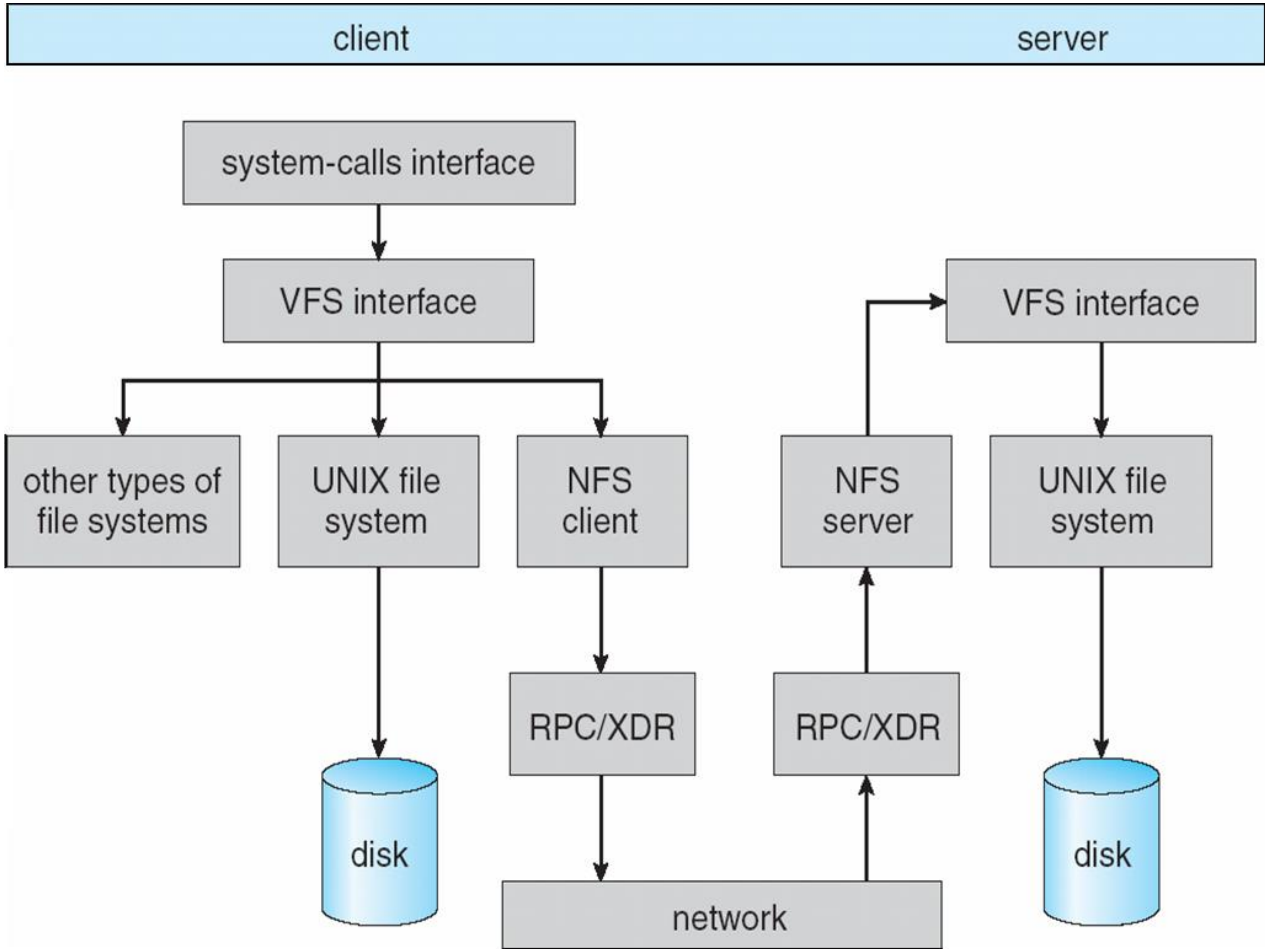
I/O With and Without a Unified Buffer Cache



Recovery Issue

- Consistency checking compares data in directory structure with data blocks on disk and tries to fix inconsistencies
- System programs have to back up data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Log structured (or journaling) file systems record each metadata update to the file system as a transaction
  - 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

Schematic View of NFS



NFS: Network File System

RPC: Remote Procedure Call

XDR: External Data Representation

Three 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