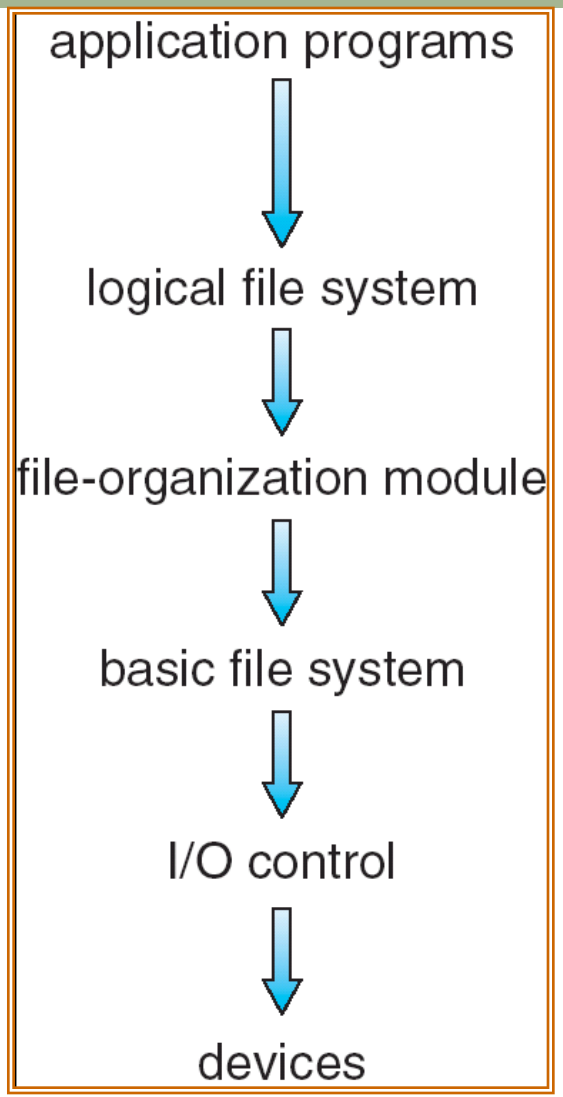
by I.-C. Lin, Dept. CS, NCTU. Textbook: Operating System Concepts 8ed

CHAPTER 11: IMPLEMENTING FILE SYSTEMS

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



Layered File System (cont.)

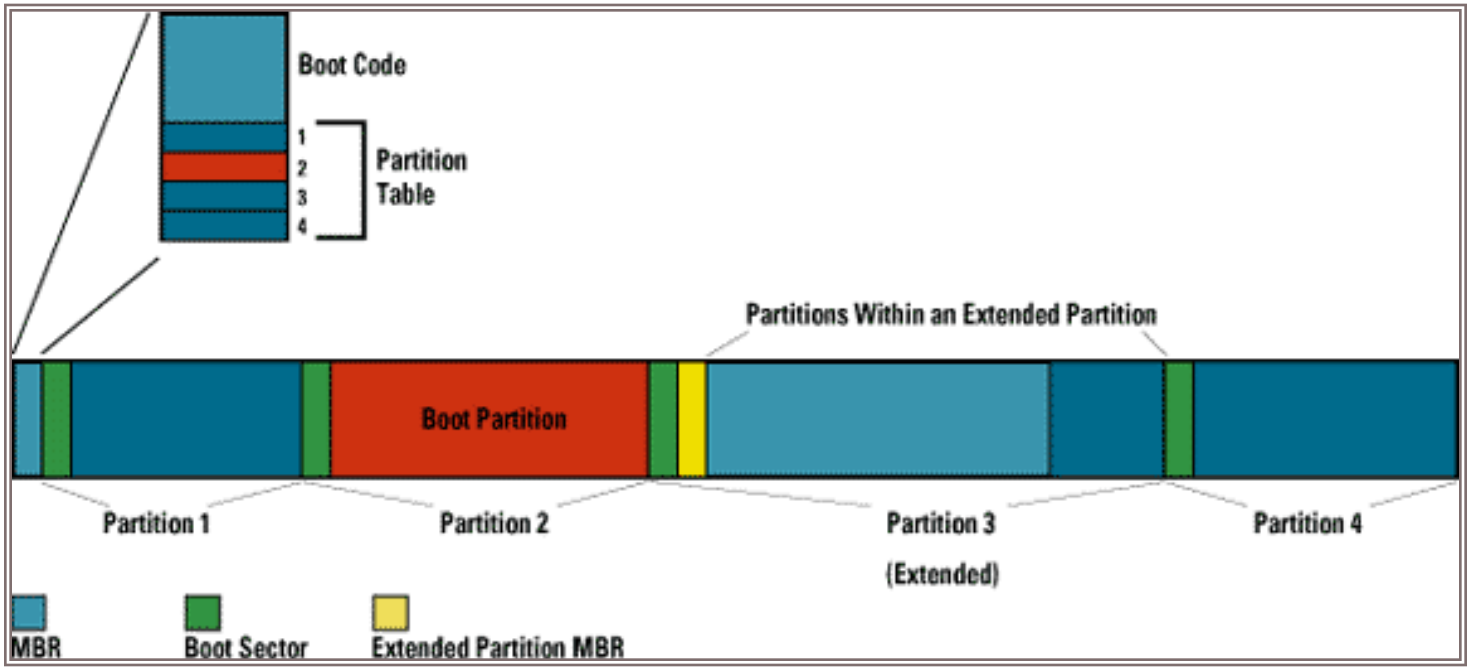
- I/O control
  - Device drivers and interrupt handlers.
  - Translate high-level commands to hardware-specific instructions.
- Basic file system
  - Issue generic commands to device drivers.
  - with physical blocks identified by its address (e.g. drive 1, cylinder 73, track2, sector 10)
- File-organization module
  - Logical blocks -> physical blocks
  - Free-space management
- Logical file system
  - Manage metadata. (all except the contents)
  - File-control block (FCB)

Layered File System (cont.)

- Layered structure
  - Flexibility
  - Minimize duplicate codes to support multiple file systems.
  - E.g.
    - CD-ROM: ISO 9660
    - Unix: UNIX file system (UFS)
    - Windows: FAT, FAT32, NTFS (Windows NT File System)
    - Linux: Extended file system (ext2, ext3…)

On-disk Structure

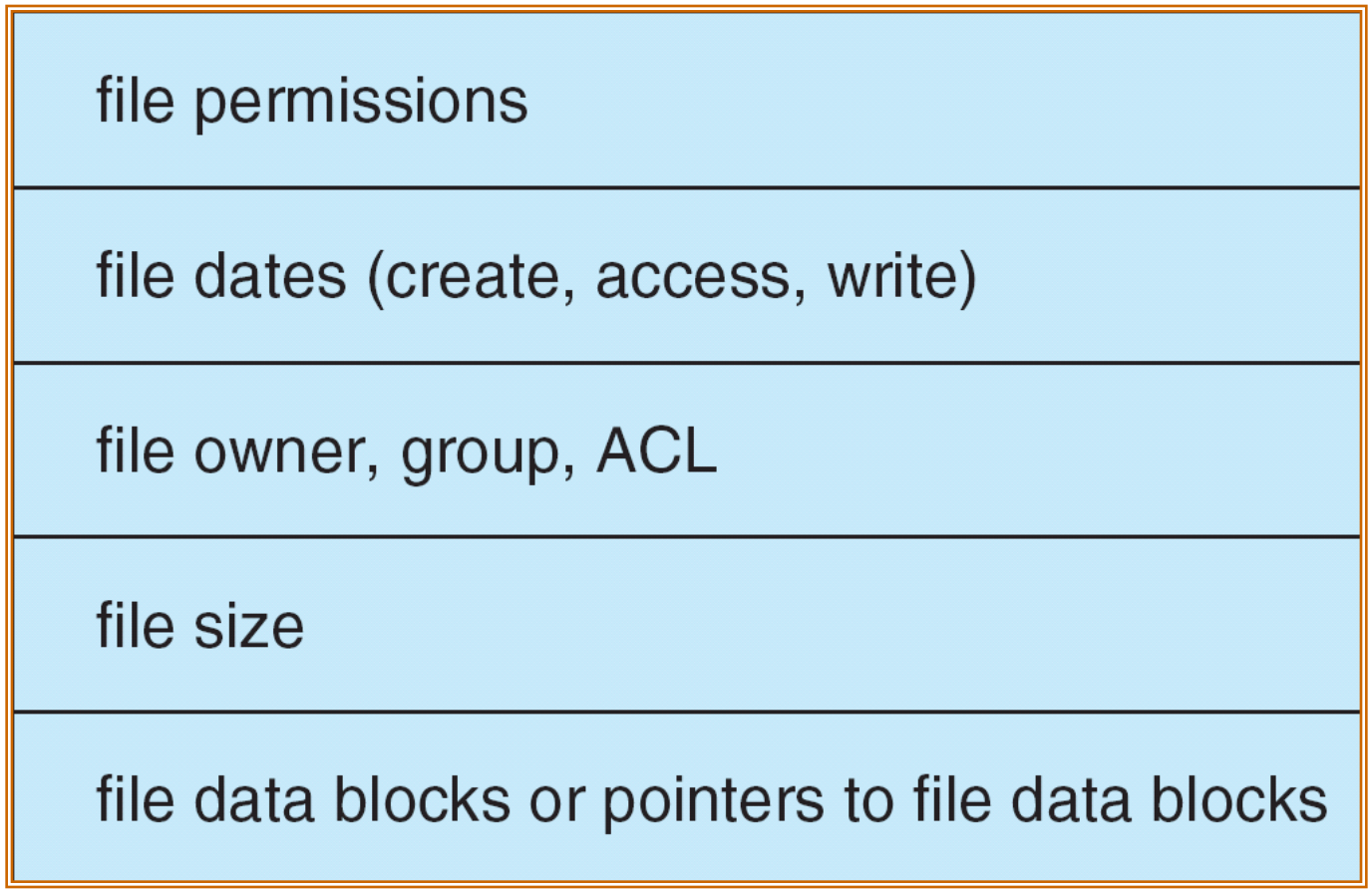
- On-disk structure
- Boot control block
  - UFS: boot block
  - NTFS: partition boot sector



On-disk Structure (cont.)

- Volume control block
  - About volume (“logical drive”) details
  - E.g. number of blocks, free block count…
  - UFS: superblock
  - NTFS: in master file table
- Directory structure
  - NTFS: in the master file table
- Per-file FCB
  - UFS: inode
  - NTFS: in the master file table

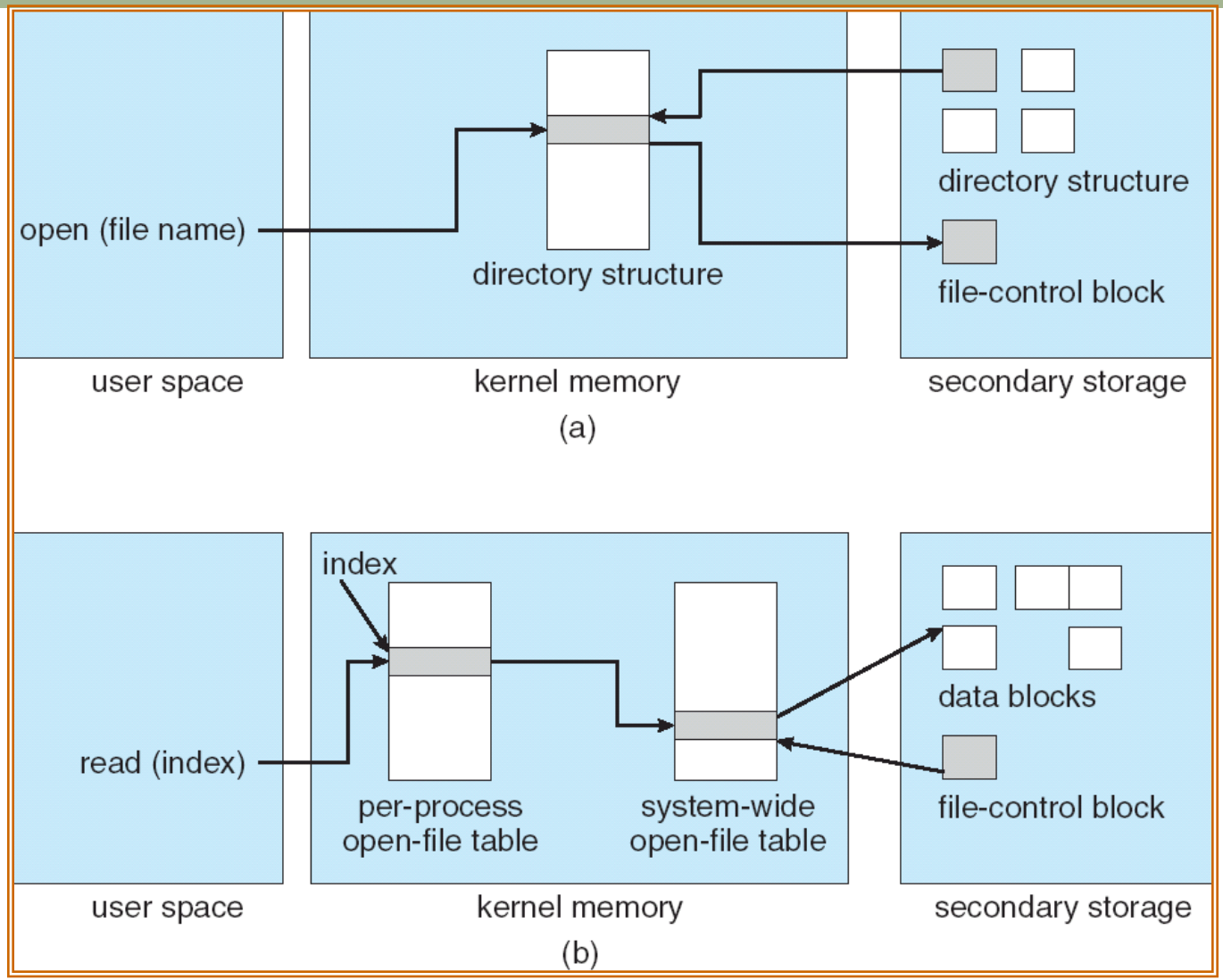
A Typical File Control Block



In-Memory File System Structures

- In-memory mounted table
- In-memory directory structure
- System-wide open-file table
  - Copy of the FCB of each open file, etc.
- Per-process open-file table
  - Per-process info. and a pointer to the system-wide open-file table

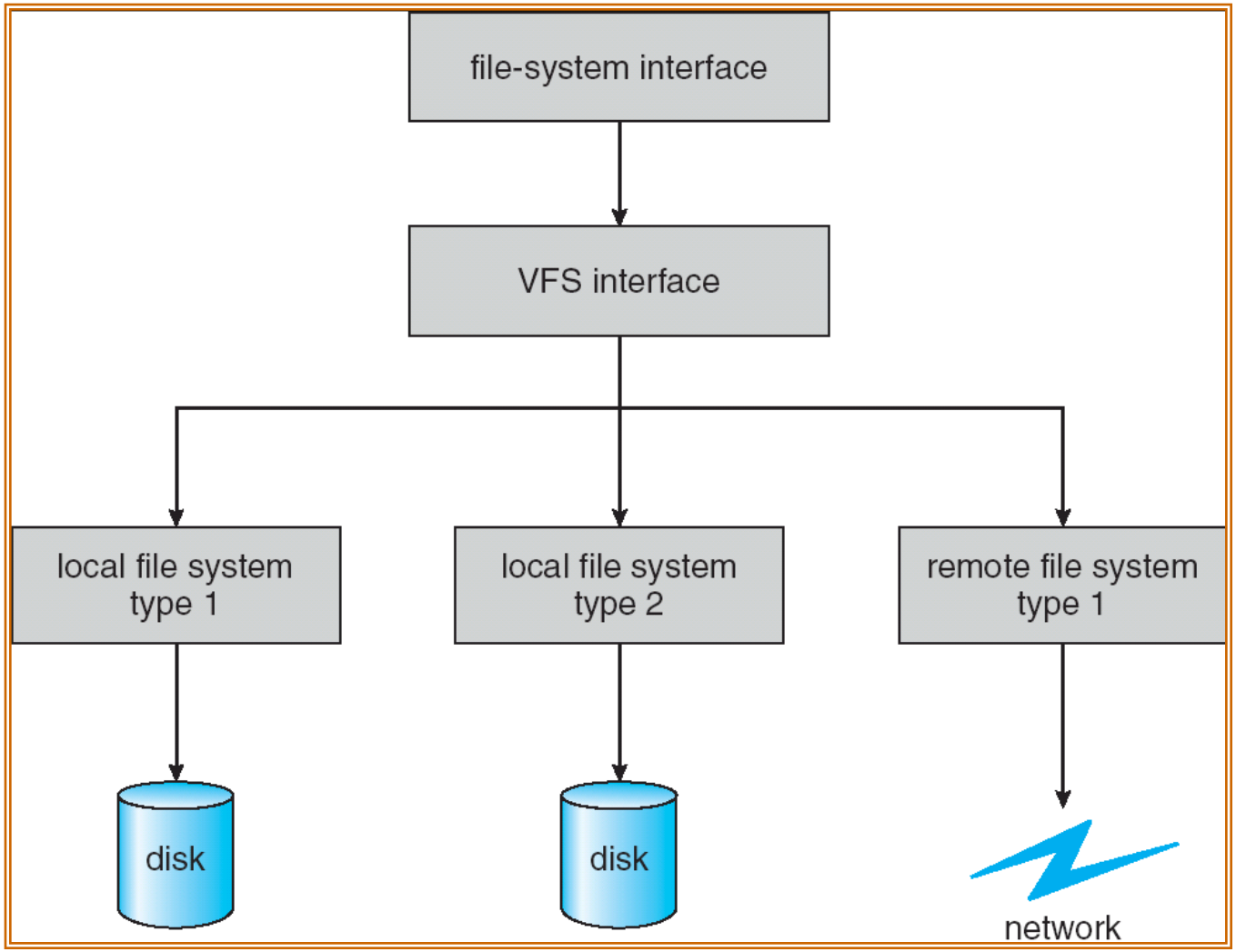
In-Memory File System Structures



Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS concurrently supports multiple types of 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



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

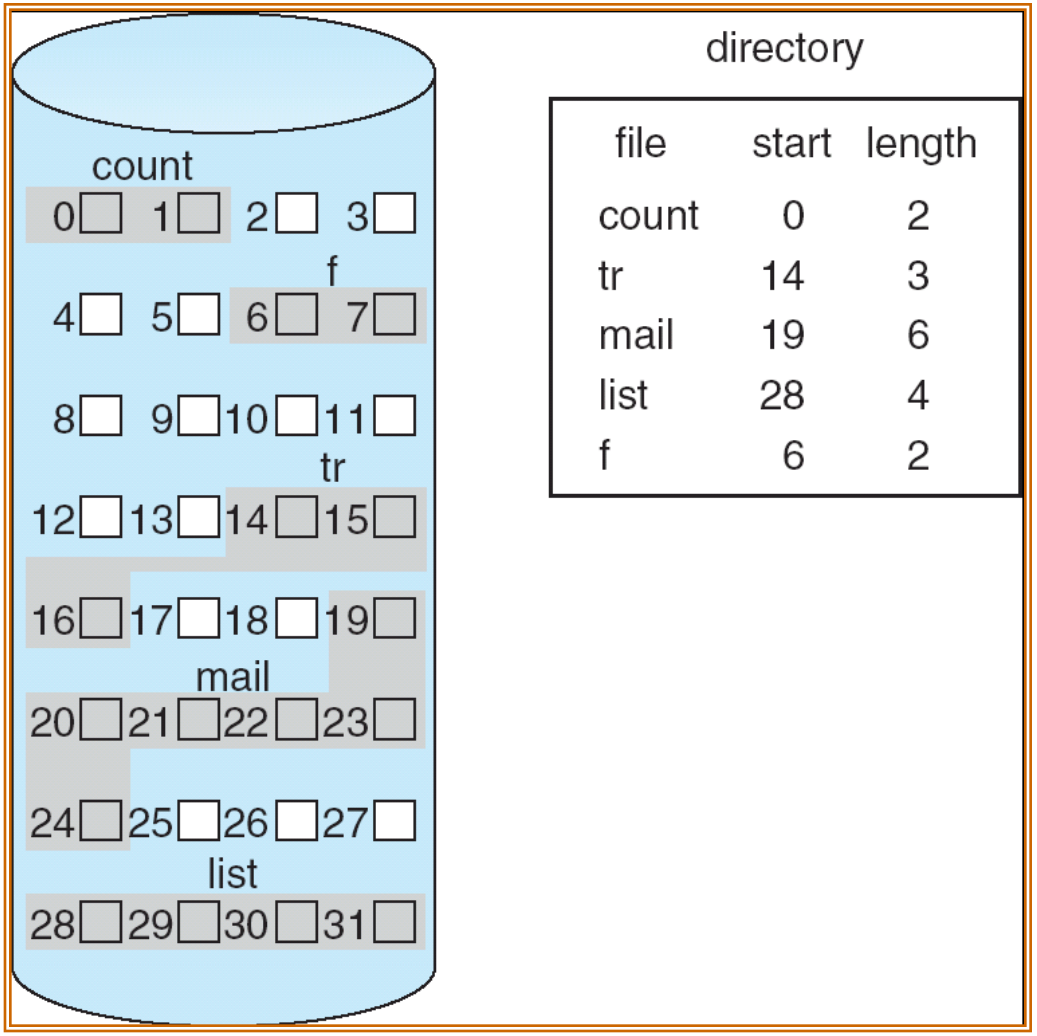
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

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

How to deal with these problems?

Contiguous Allocation of Disk Space

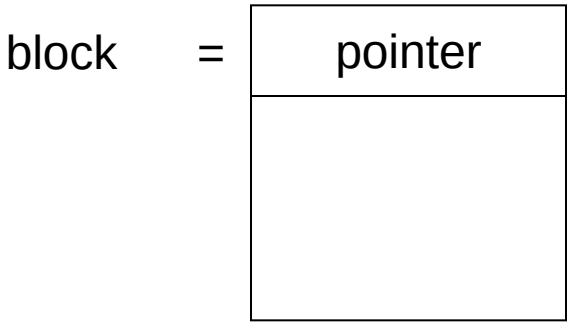


Extent-Based Systems

- 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.
  - A link is assigned to the next extend.
  - Internal fragmentation.

Linked Allocation

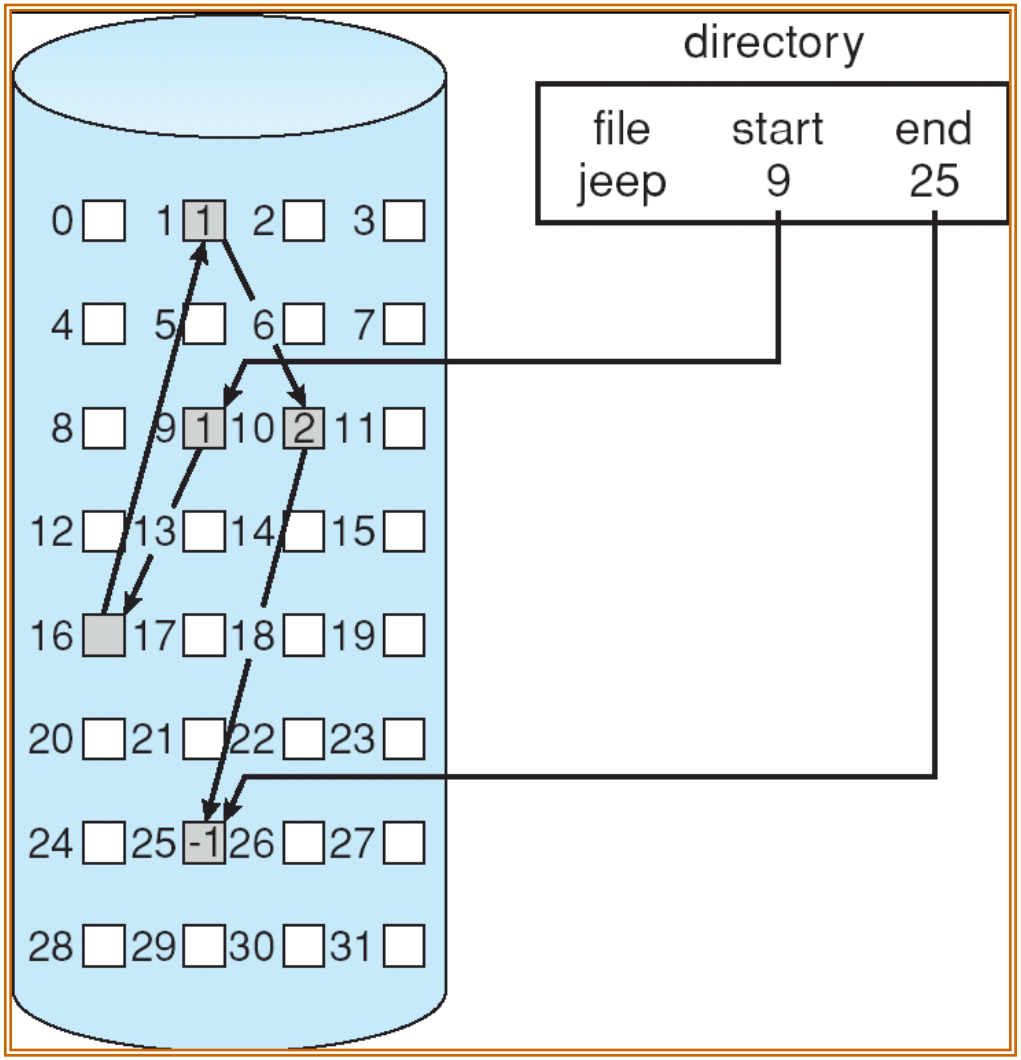
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



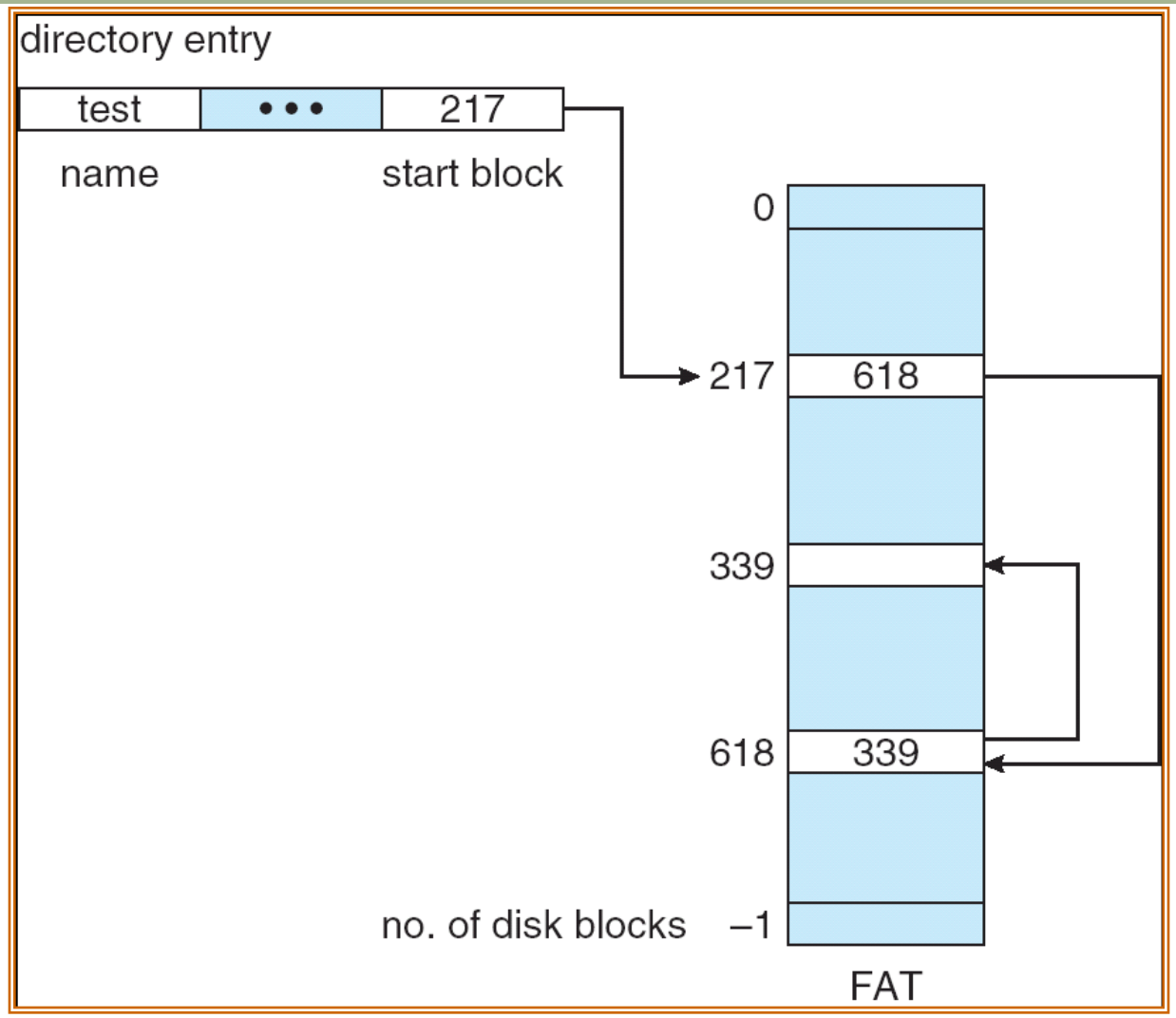
Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system
  - no waste of space
- No random access
- File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation

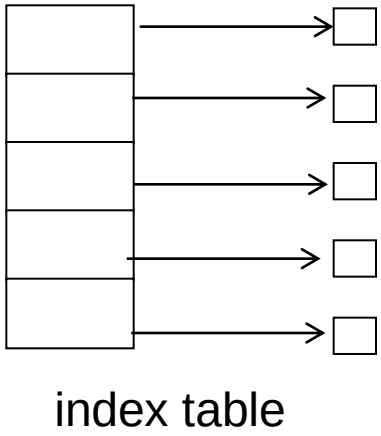


File-Allocation Table

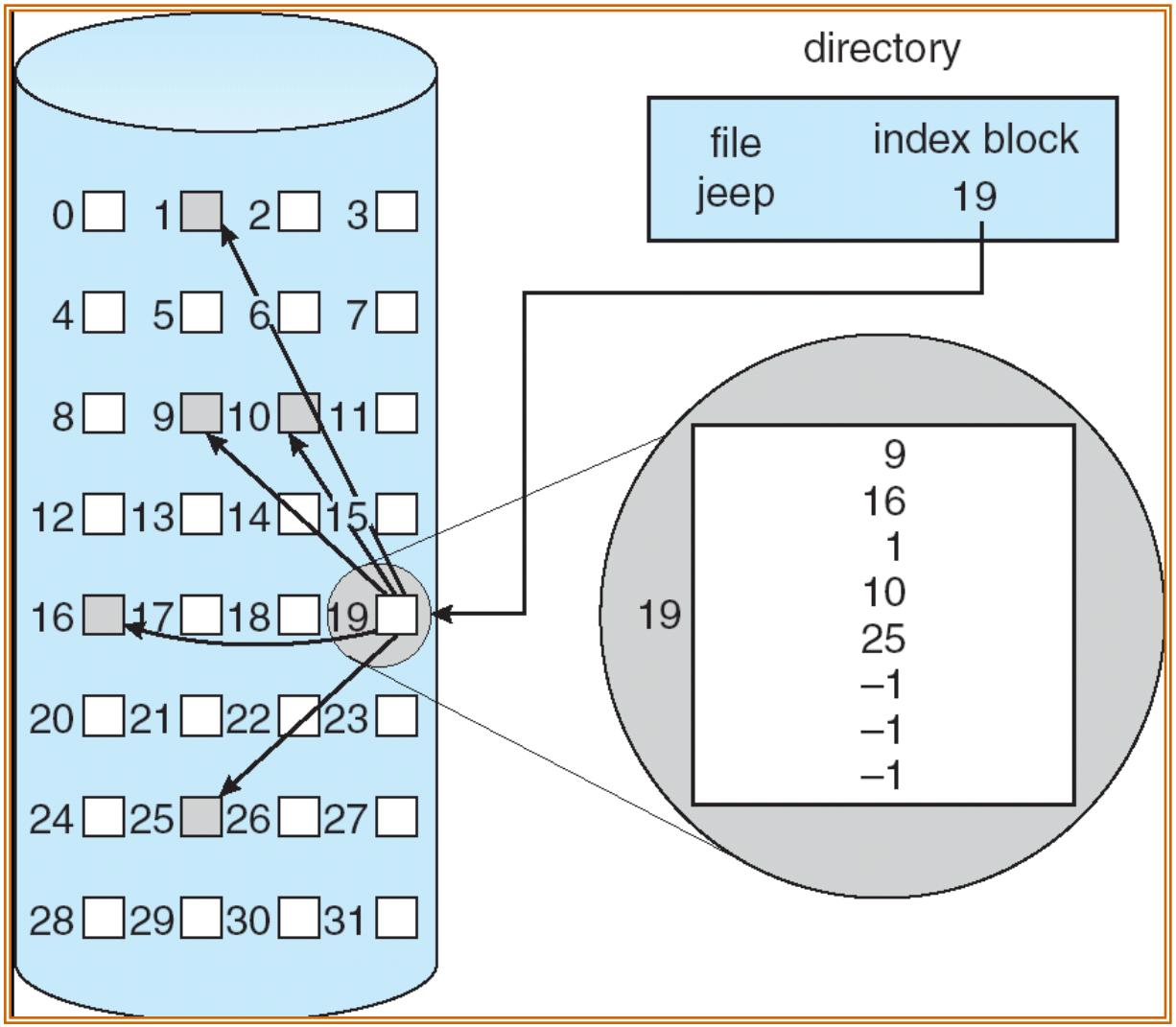


Indexed Allocation

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



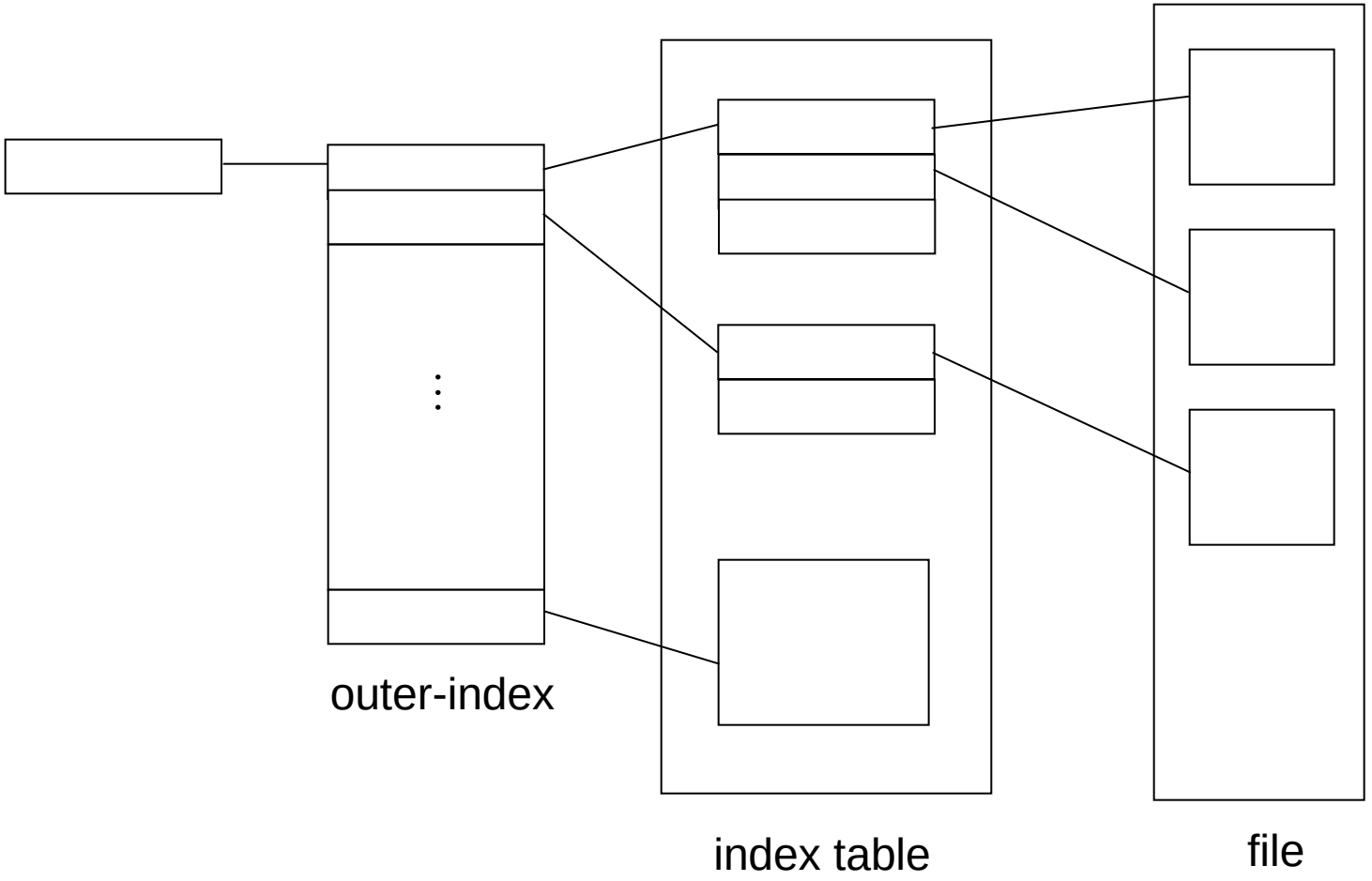
Example of Indexed Allocation



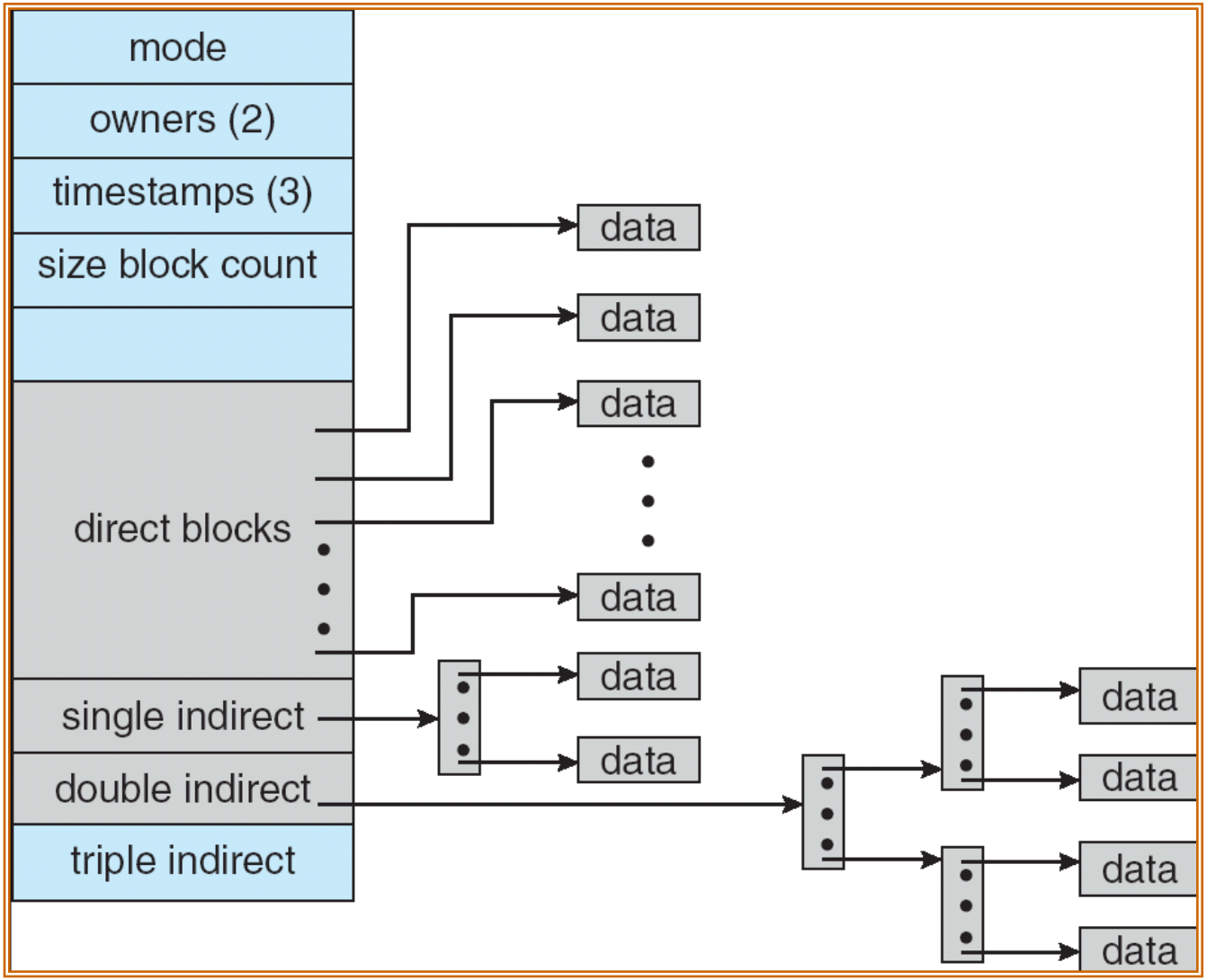
Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- How large should the index block be?
  - Linked scheme – Link blocks of index table (no limit on size).
  - Multilevel index

Indexed Allocation – Mapping (Cont.)

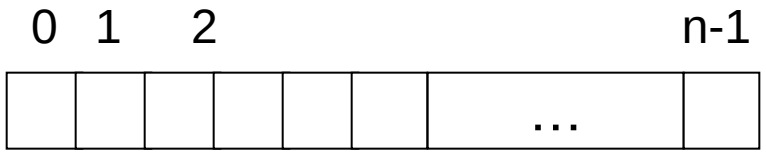


Combined Scheme: UNIX (4K bytes per block)



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.)

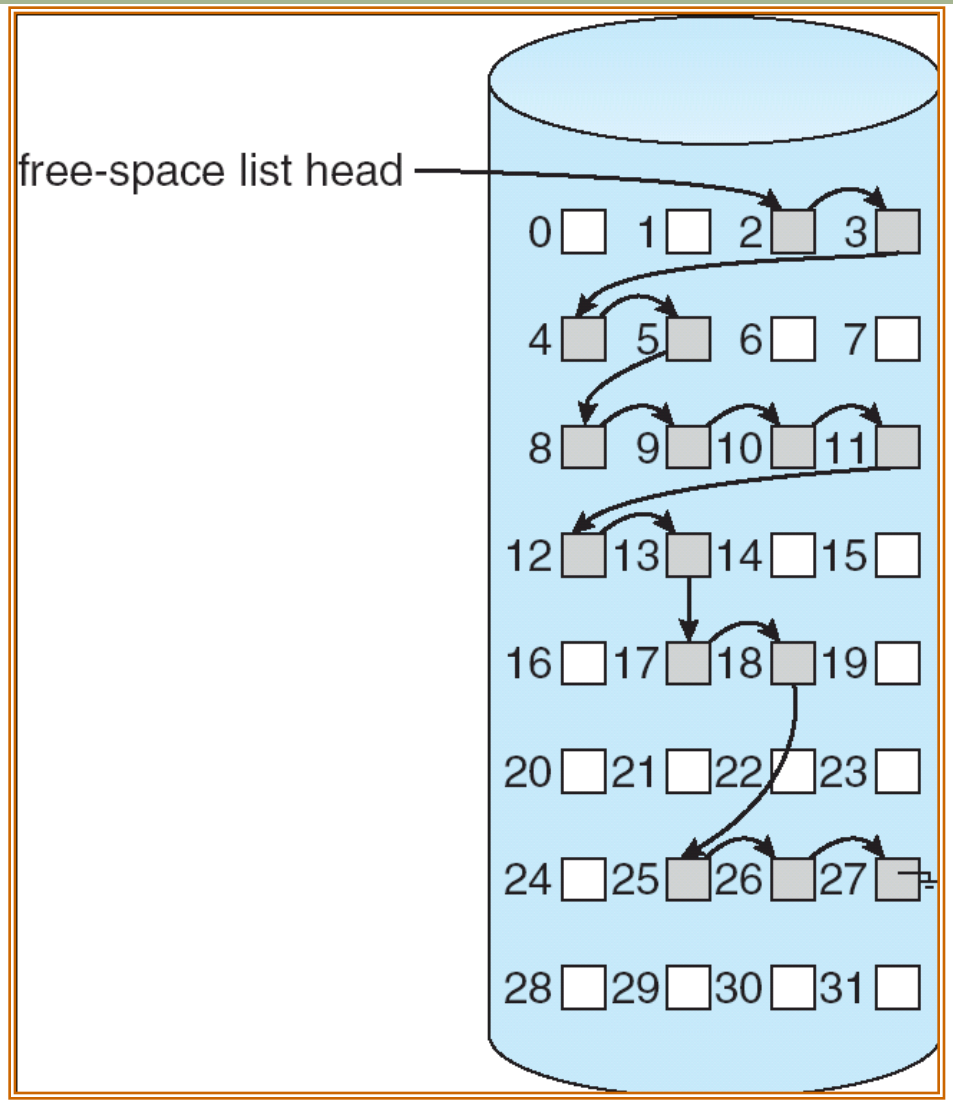
- Bit map requires extra space
  - Example:

    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
  - Modified free-list (n free block a group)
- Counting
  - With a variable free-contiguous-block length

Linked Free Space List on Disk



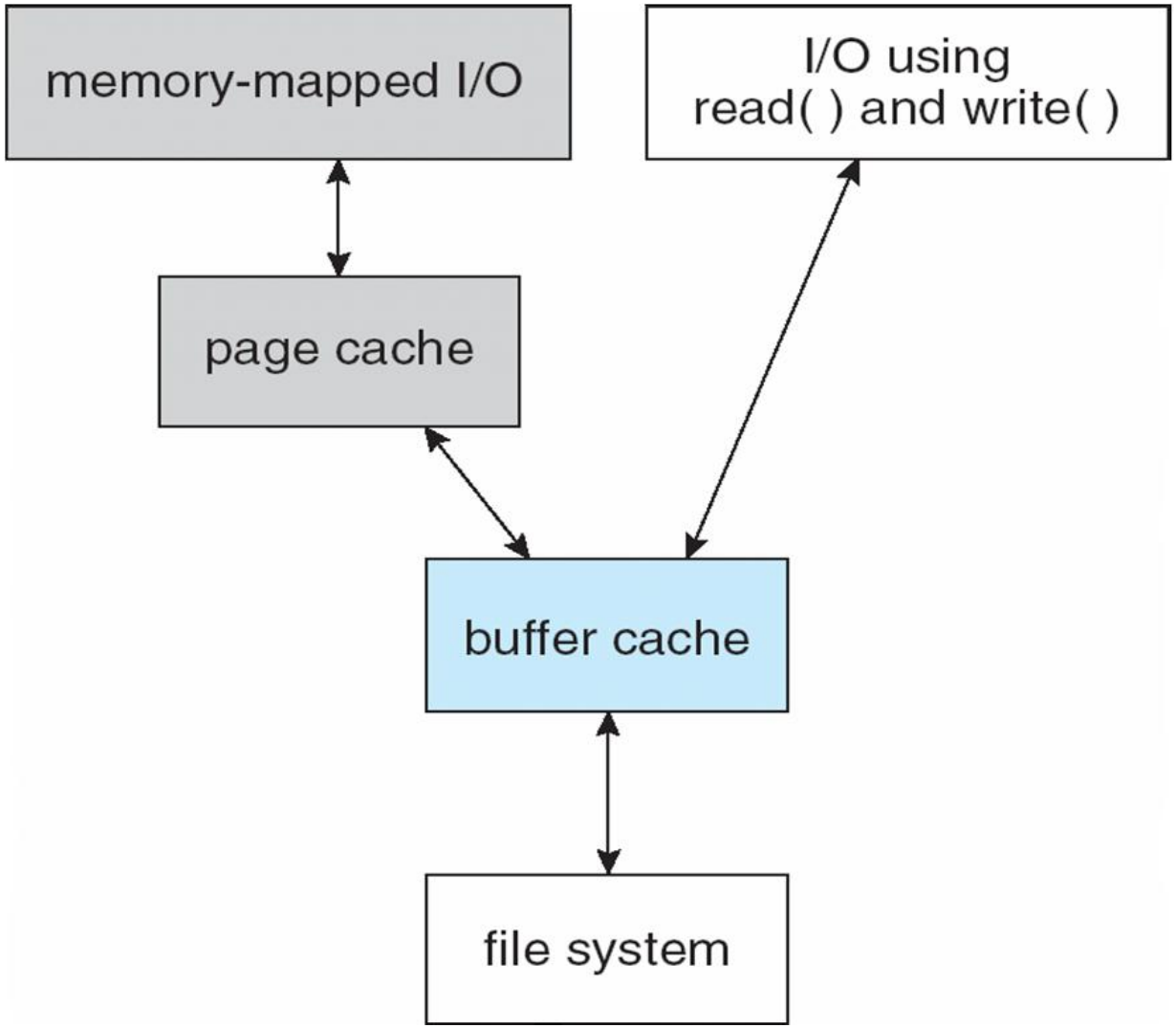
Efficiency and Performance

- 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

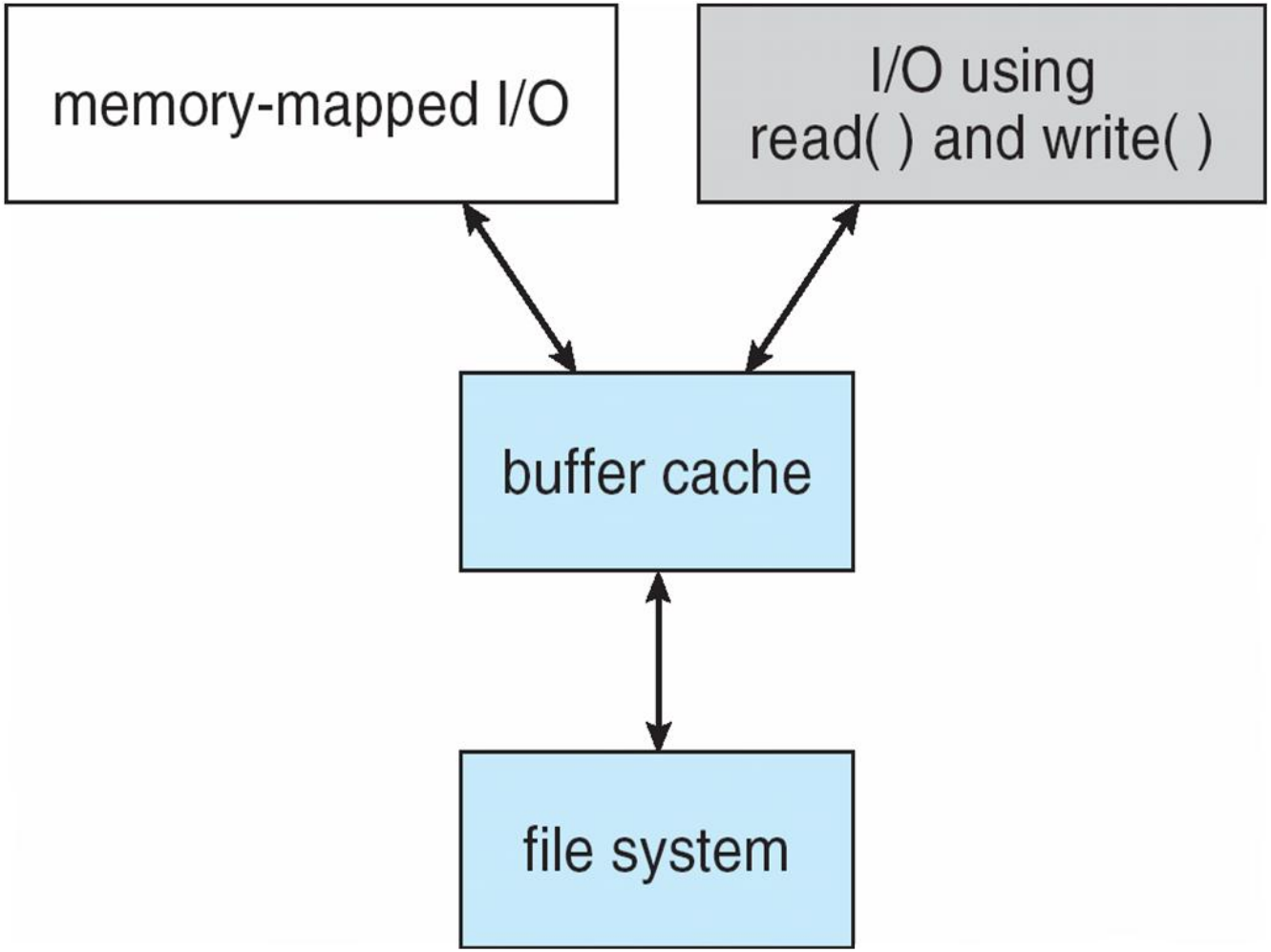
I/O Without a Unified Buffer Cache



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)
- 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