Chapter 11: File System Implementation

- Overview
  - File system structure – layered, block based
  - FS Implementation: FCB, mounting, VFS
  - Directory implementation: Linear, hash table, B-tree
  - Allocation methods: Contiguous, Linked, Indexed, FAT
  - Free-space management: Bit vector, Linked list

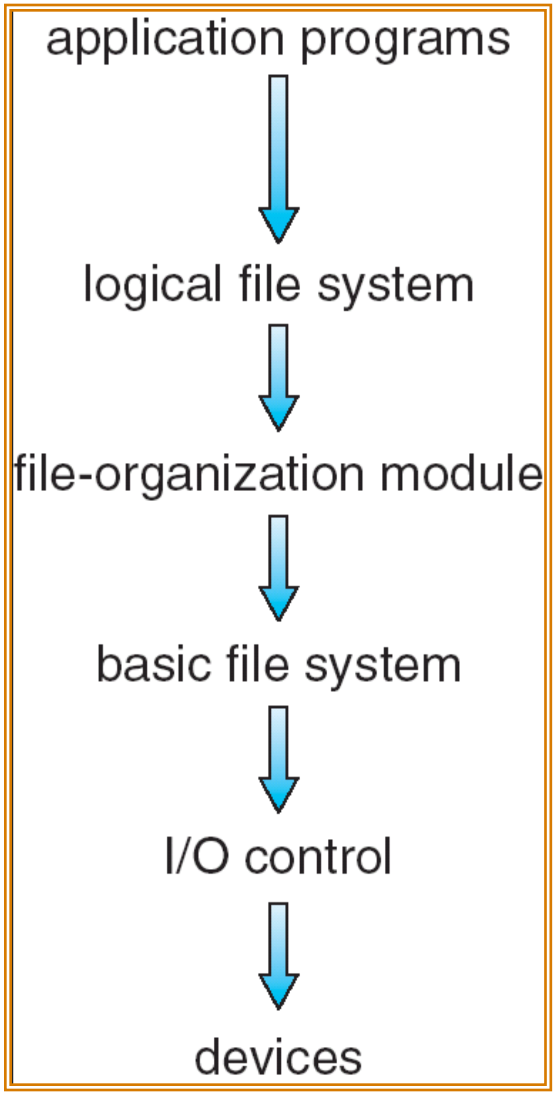
Chapter 11: File System Implementation

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (such as disks)

1. Boot control block - information needed to boot
2. Volume control block - information about volume/ partitions (# blocks, size of blocks, free block count, free block pointers)
3. Directory structure (inode)
4. Per file control blocks

- File system organized into layers

Layered File System


application programs

logical file system

file-organization module

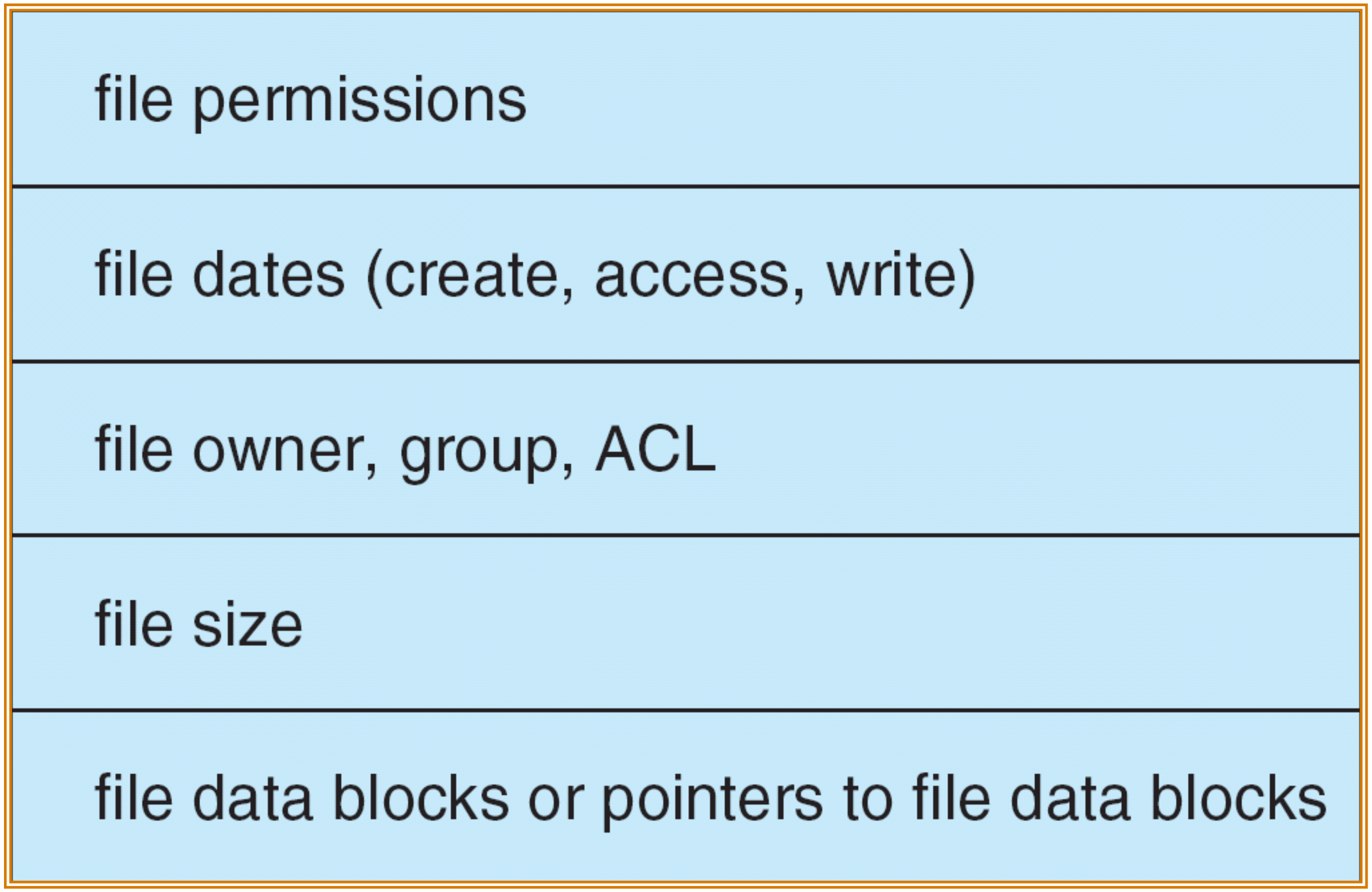
basic file system

I/O control

devices

A Typical File Control Block

- **File control block** – storage structure consisting of information about a file



file permissions

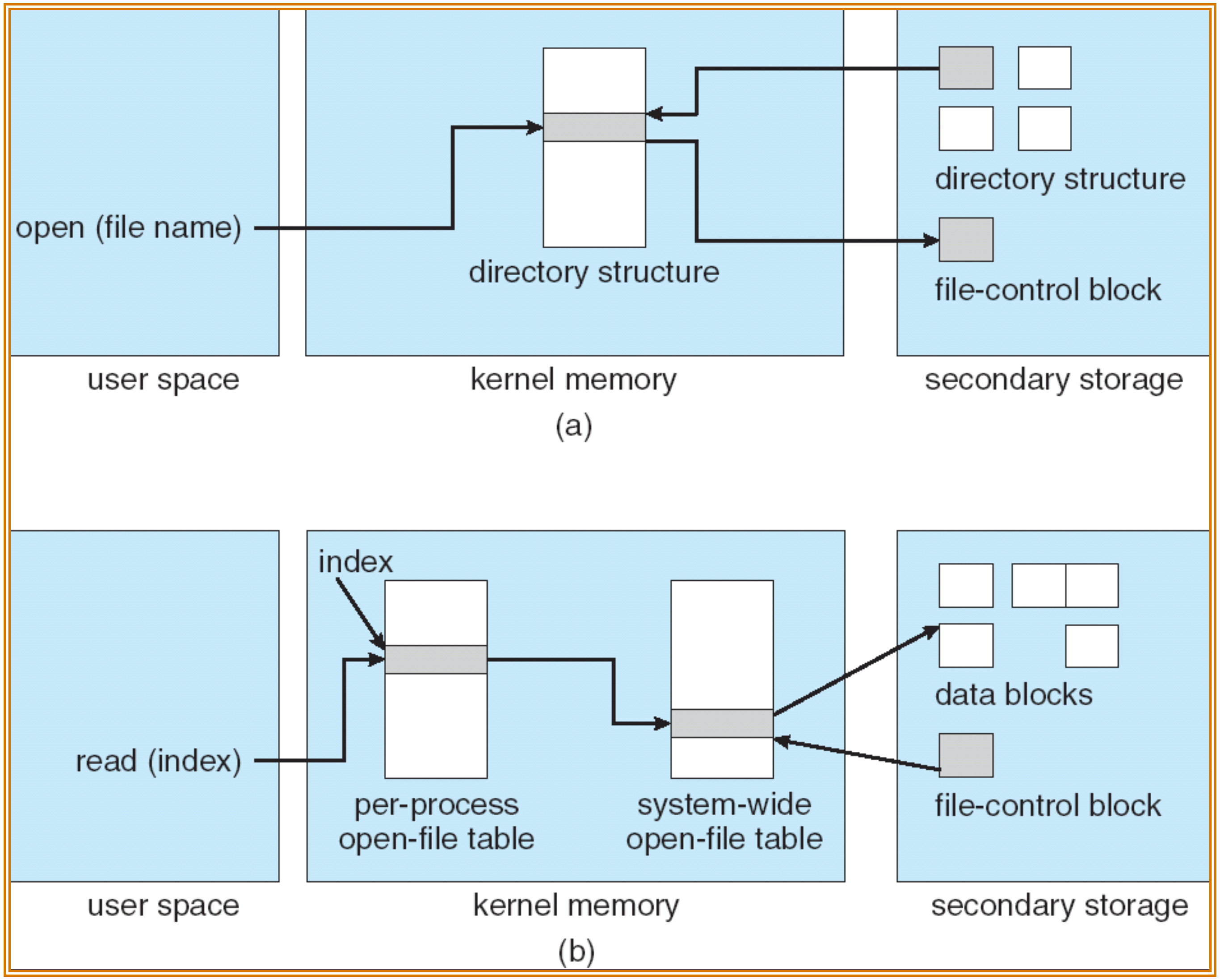
file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks

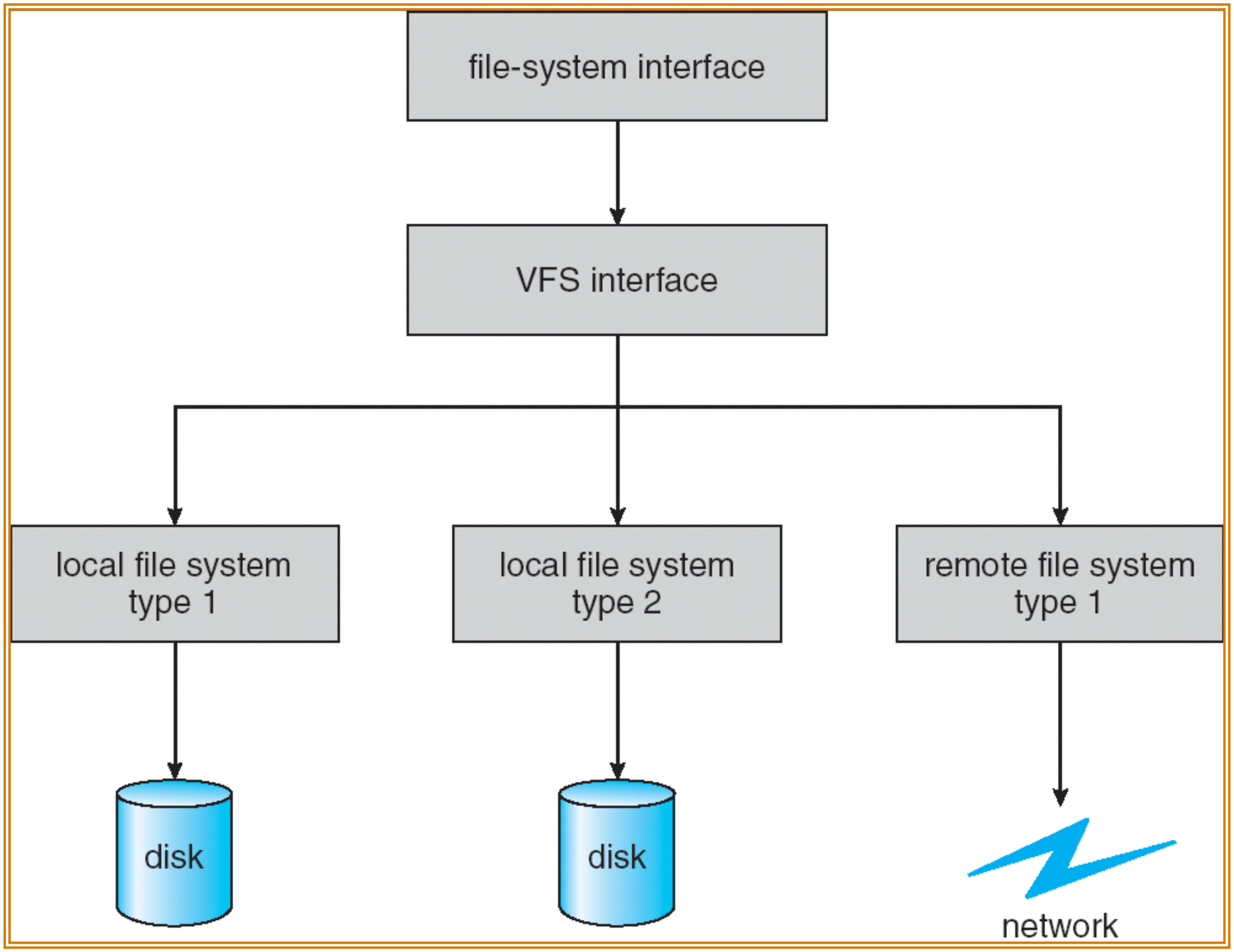
In-Memory File System Structures



Virtual File Systems

- There are many different file systems available on any operating systems
  - Windows: NTFS, FAT, FAT32
  - Linux: ext2/ext3, ufs, vfat, ramfs, tmpfs, reiserfs, xfs ...
- 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



Directory Implementation

- **Directories hold information about files**
- **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

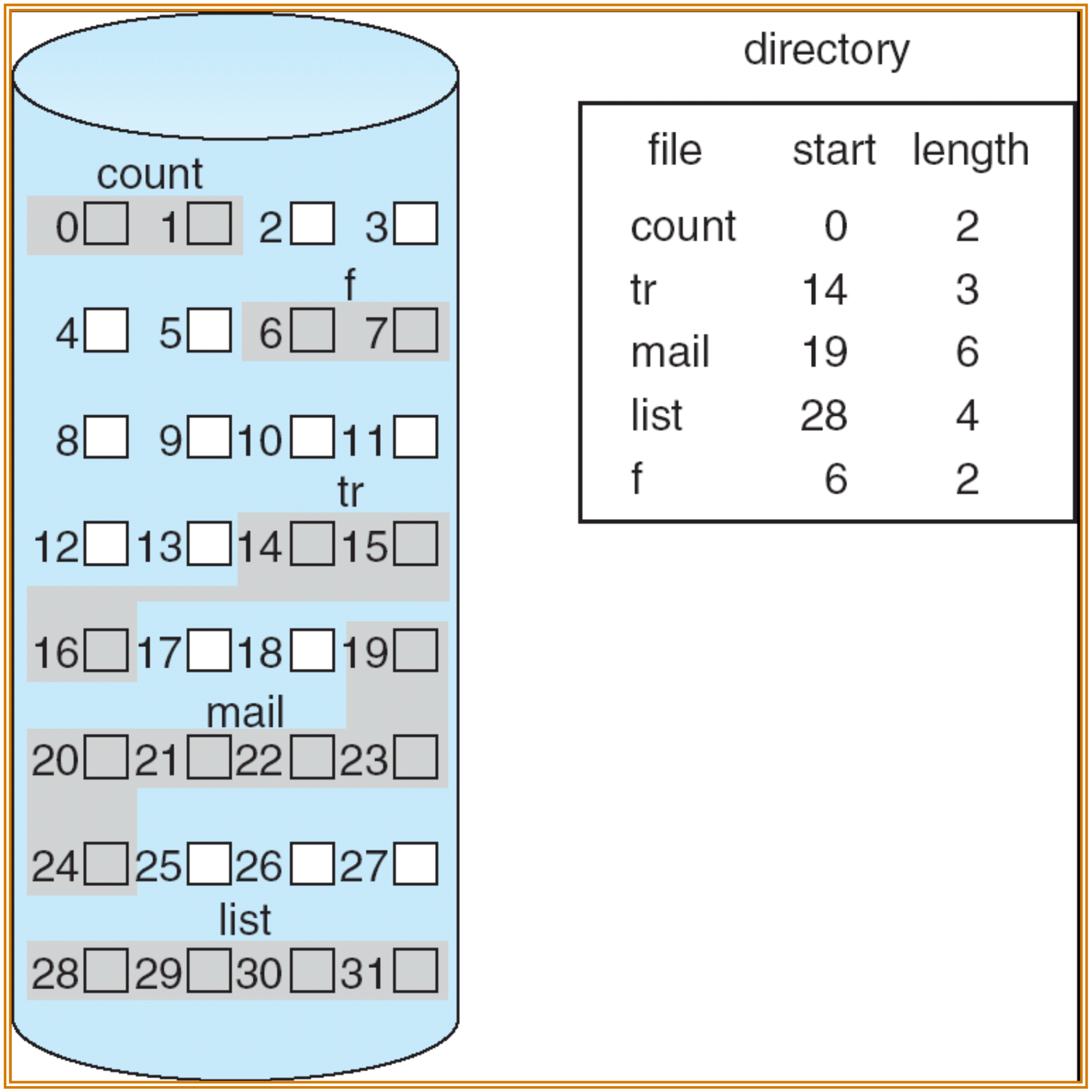
Allocation Methods

- 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 of Disk Space



directory

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

Extent-Based Systems

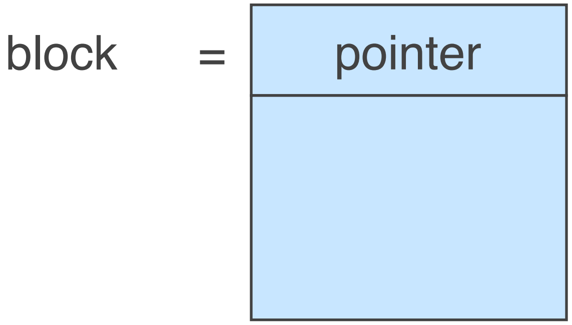
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme

- 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.

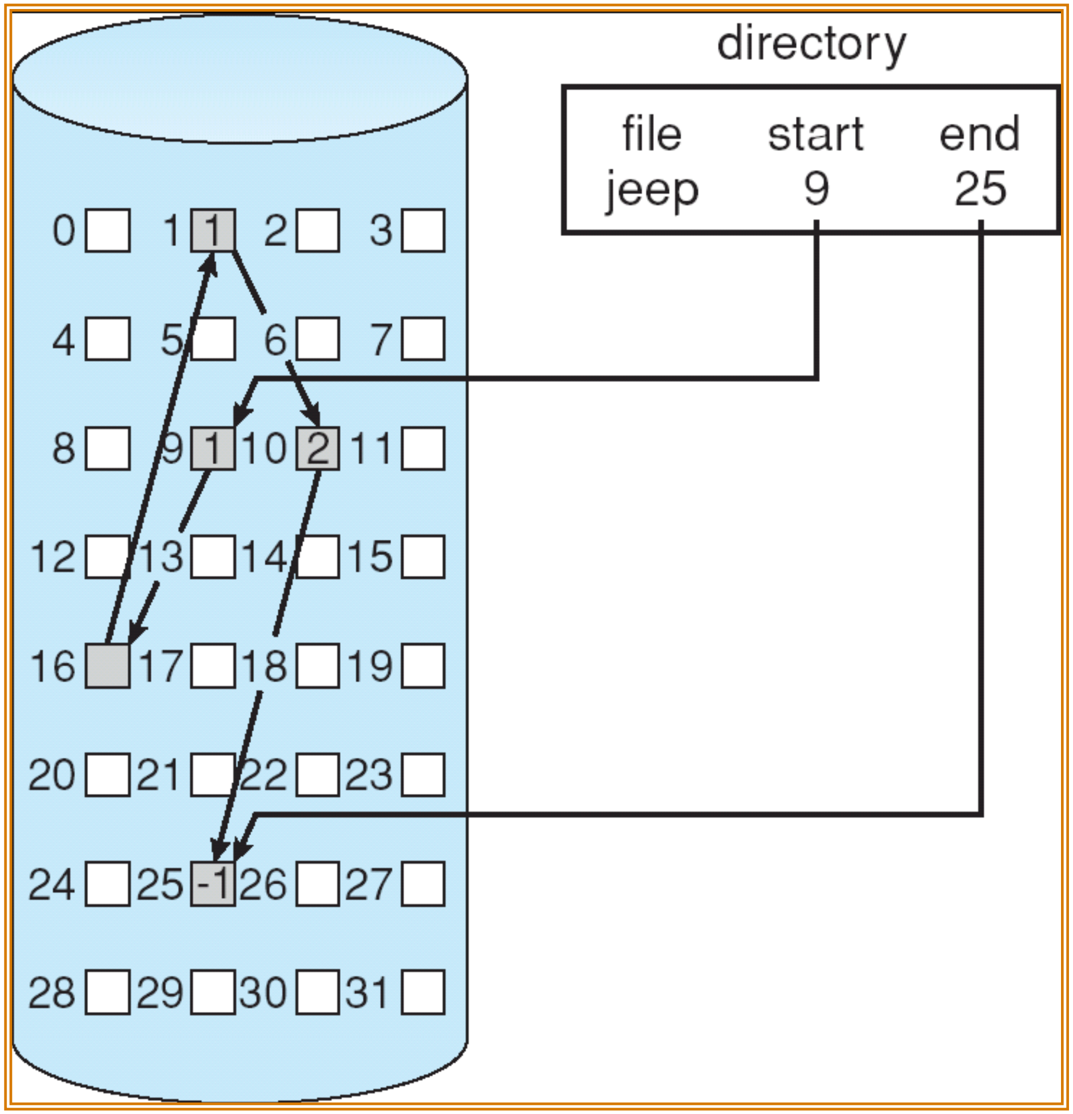
Linked Allocation

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

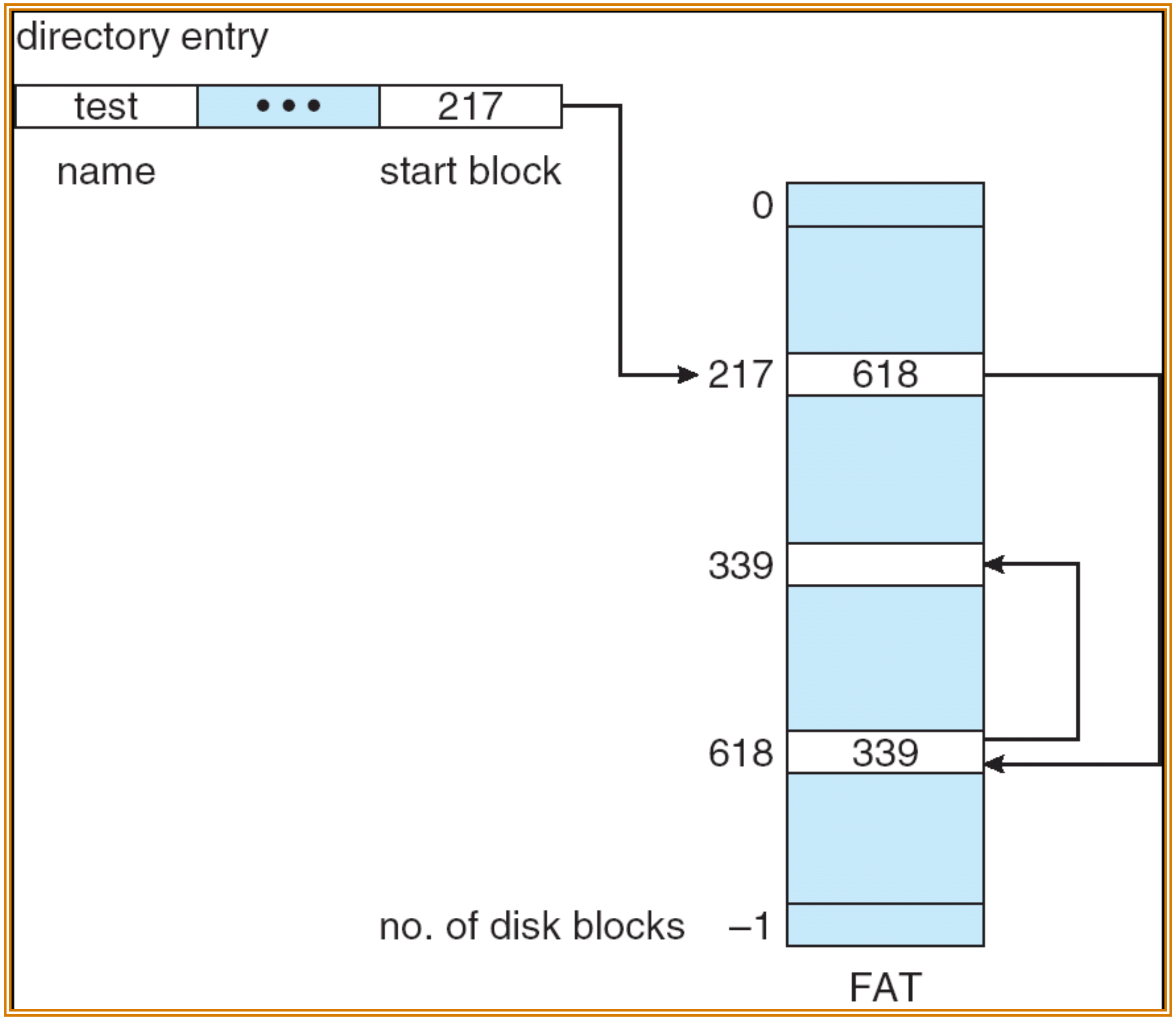


- Simple – need only starting address
- Free-space management system – no waste of space
- No random access

Linked Allocation

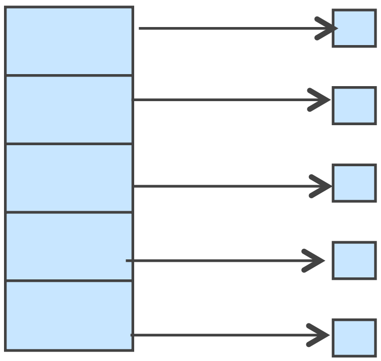


File-Allocation Table (DOS FAT)



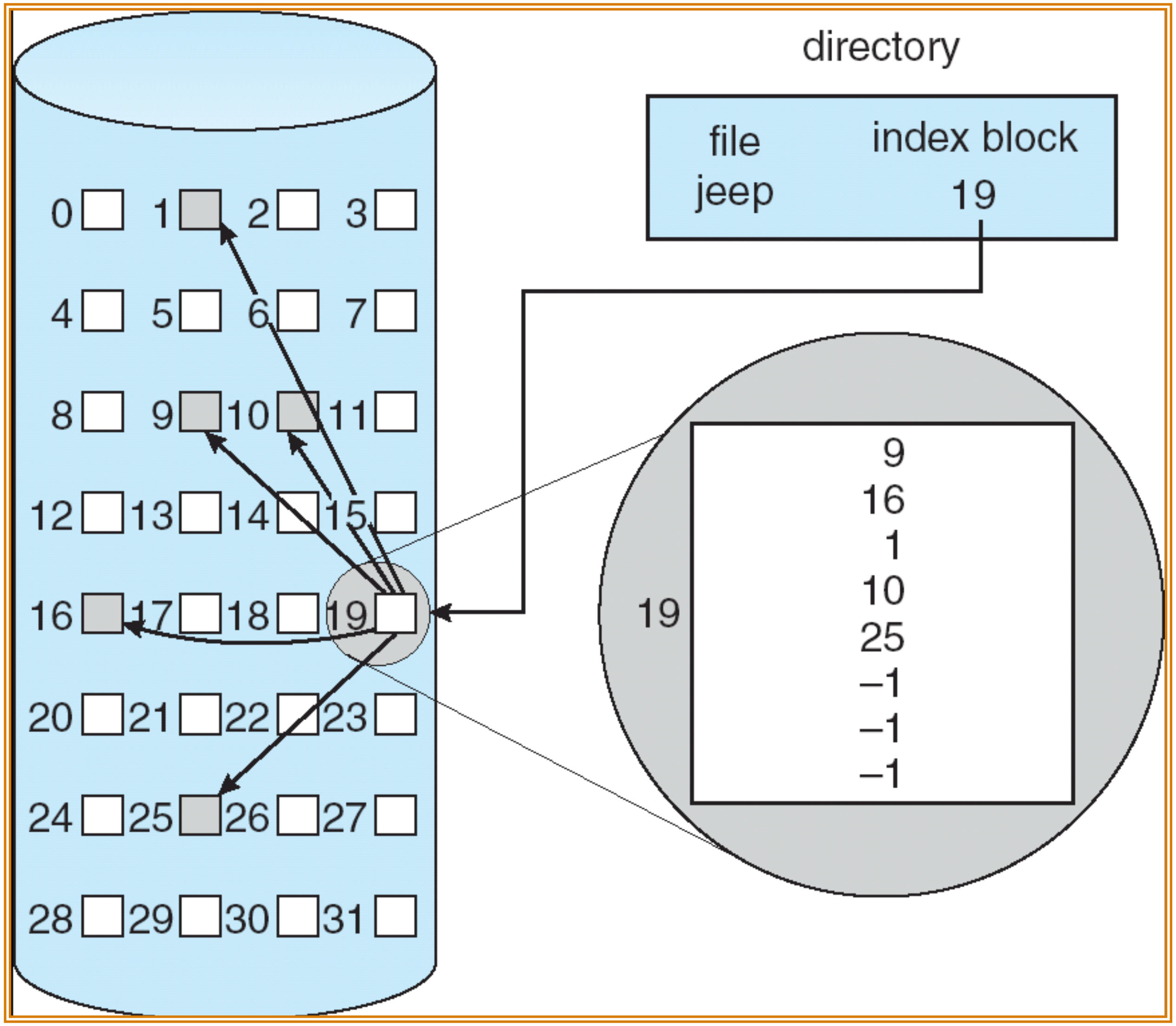
Indexed Allocation

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



index table

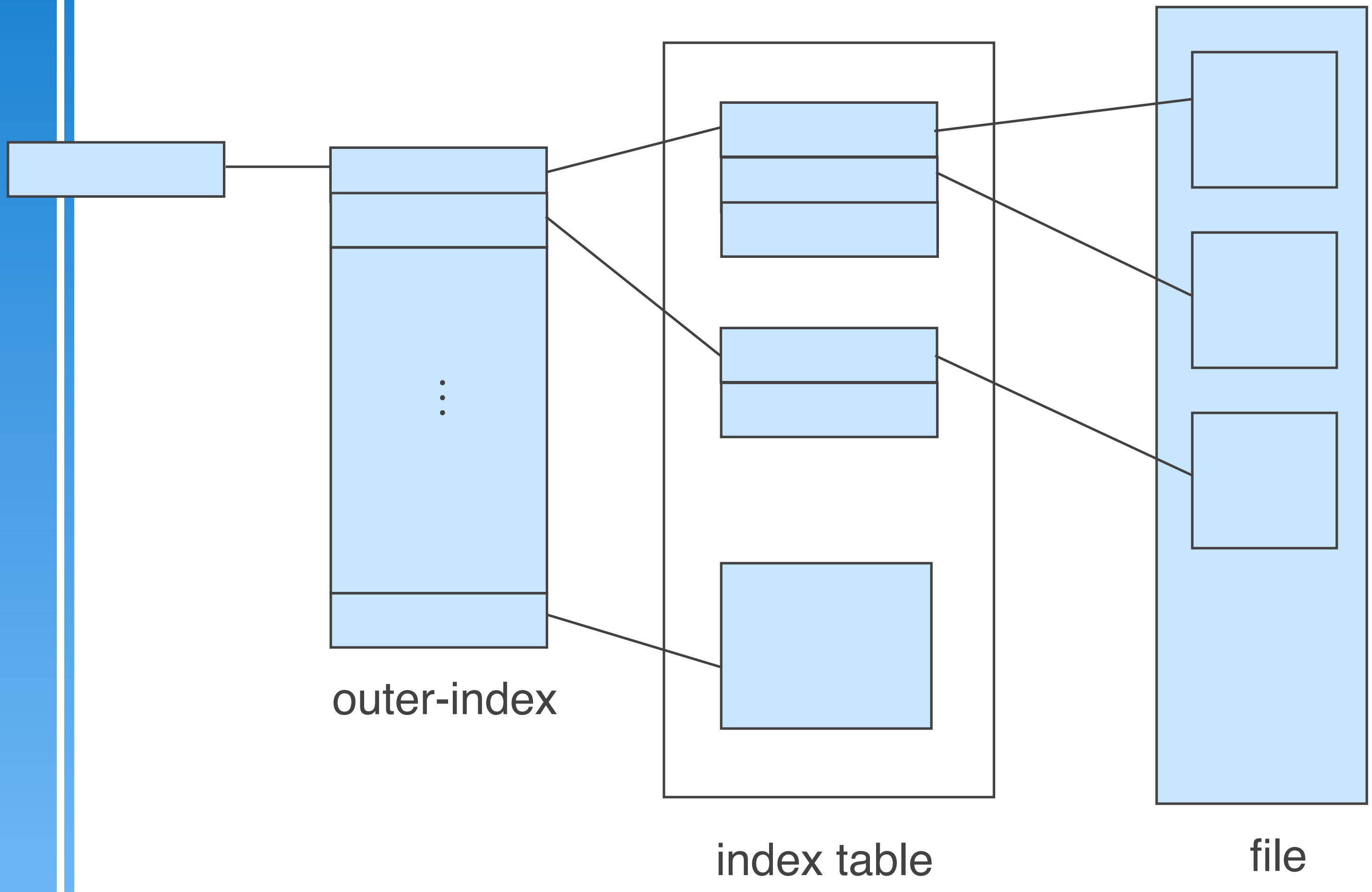
Example of Indexed Allocation



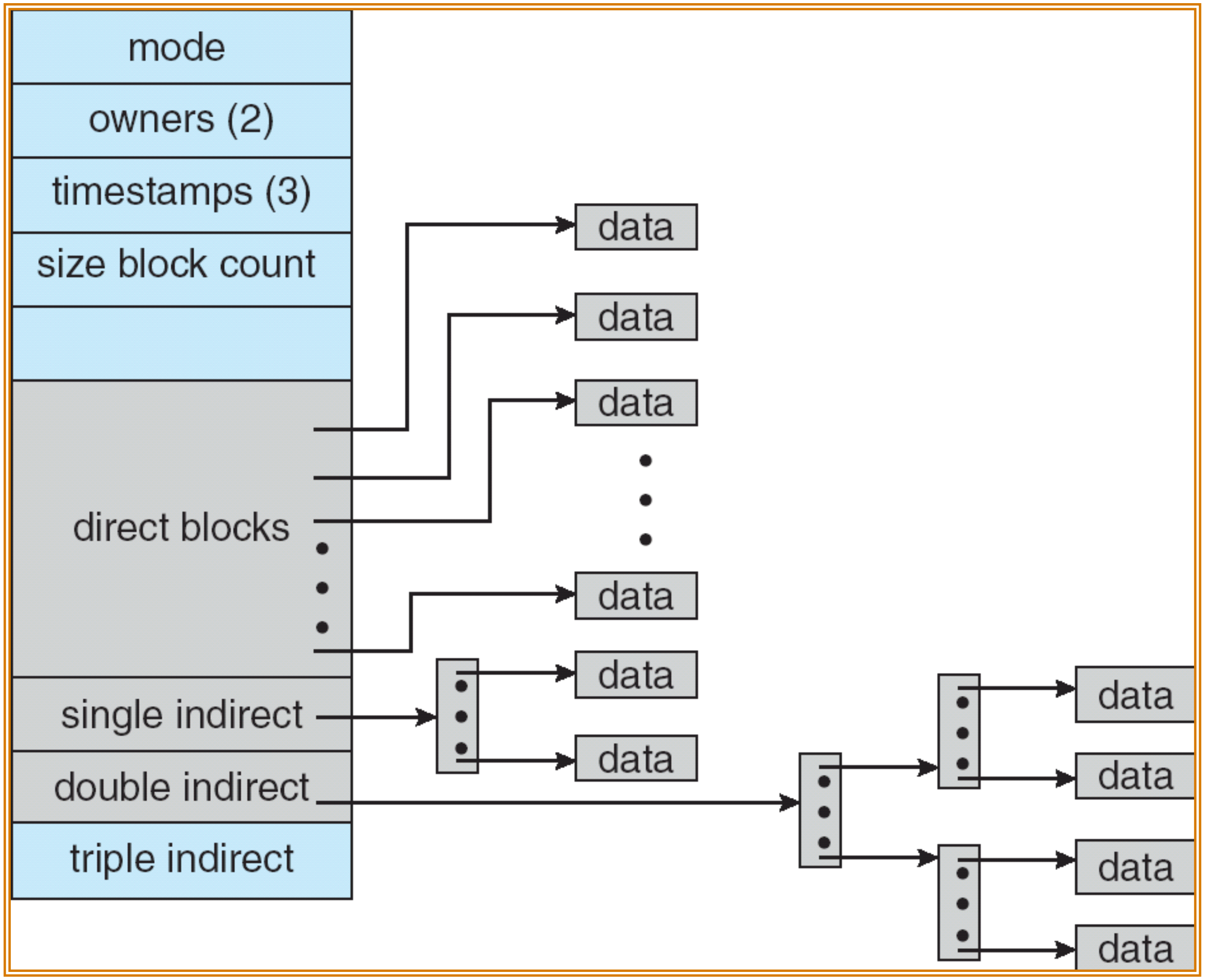
Indexed Allocation (Cont.)

- 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.)

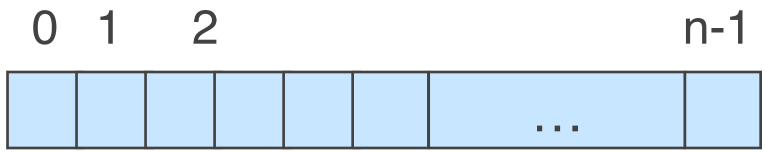


Combined Scheme: UNIX (4K bytes per block)



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.)

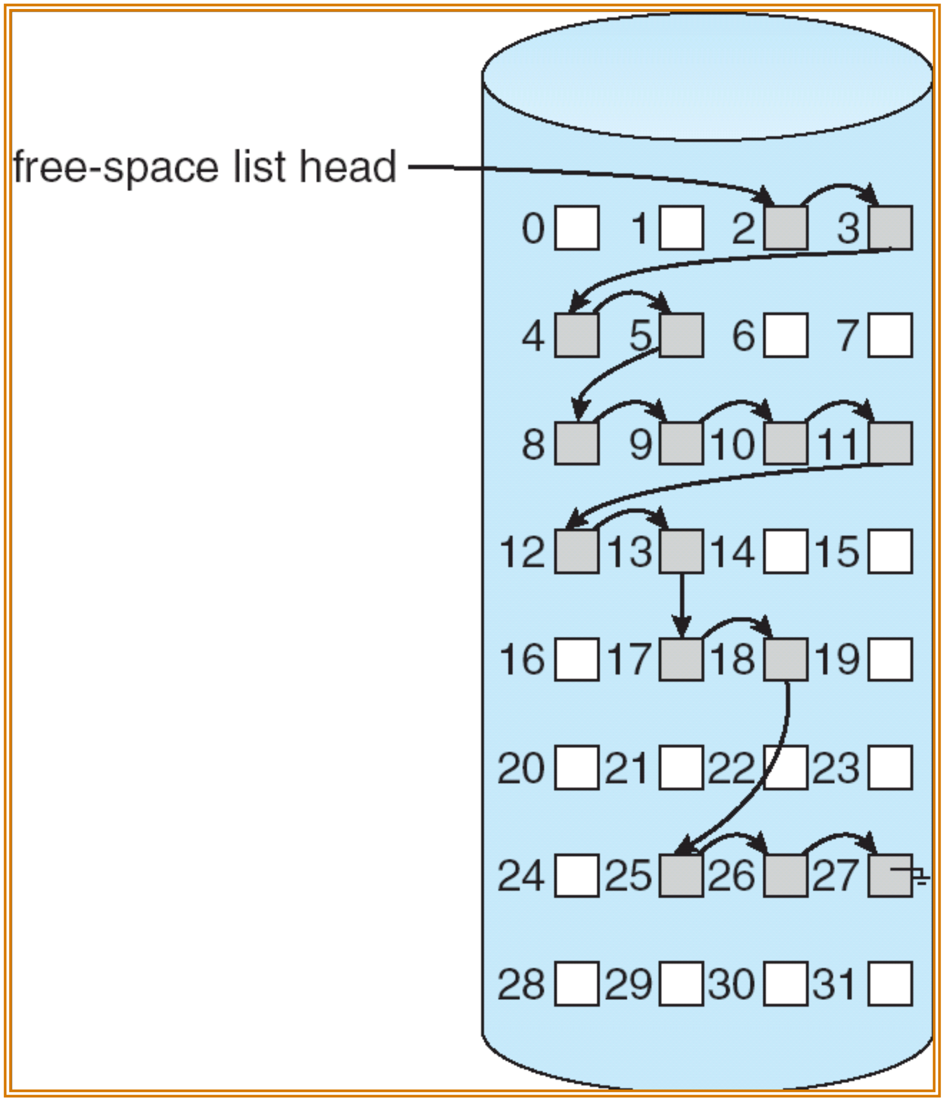
- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

    disk size = 2^{38} bytes (256 Gigabyte)

    $n = 2^{38}/2^{12} = 2^{26}$ bits (or 8 Mbytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

Linked Free Space List on Disk



Free-Space Management (Cont.)

- Need to protect against inconsistency:
  - 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