File System Implementation

Chapter 11

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

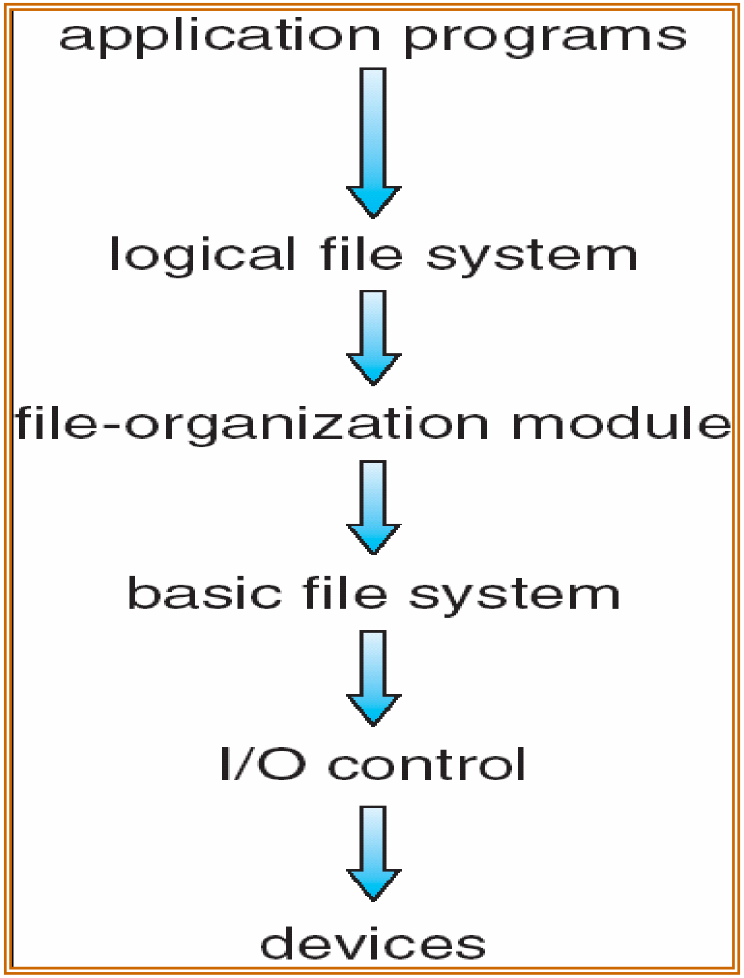
Chapter Outline

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

- 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



-- File-System Layers ...

- **Application Programs**
  - Interface that issues system calls to the logical file system.
- **Logical file system**
  - Manages the metadata information of the files. Ex. FCB, directories
  - Uses a symbolic file name (usually from the application program) and searches the directory to provide the file organization module with the information it needs.
  - Also provides protection and security
- **File-organization module**
  - Takes care of free-space management.
  - Translate logical to physical addresses |

... -- File-System Layers ...

- **Basic file system**
  - Issues generic commands to the appropriate device driver to read and write physical blocks on the disk.
    - Physical blocks (drive #, Cylinder #, Track #, Sector #)
  - Physical block is translated into block # and presented to I/O control layer.

... -- File-System Layers

- **I/O control**
  - Controls devices - Consists of device drivers, interrupt handlers:
  - Input to a device driver is generally a high-level command (e.g., retrieve block 150)
  - Output of a device driver is low-level, hardware specific instructions used by hardware controller, which interfaces the I/O device to the rest of the system.
  - Usually, device driver writes specific bit patterns to special locations in the I/O controller's memory (Control register) – These bits tells the controller the device location and what to do.

- File-System Implementation

- In Disk file-system structures
- In memory file-system structures
- Virtual file-systems

- In Disk file-system structures

- Boot control block
- Volume control block
- Directory structure per file system
- File control block

-- A Typical File Control Block (FCB)

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 ...

- Mount table
- Directory structure cache
- System-wide open-file table (copy of FCB)
- Per-process open-file table

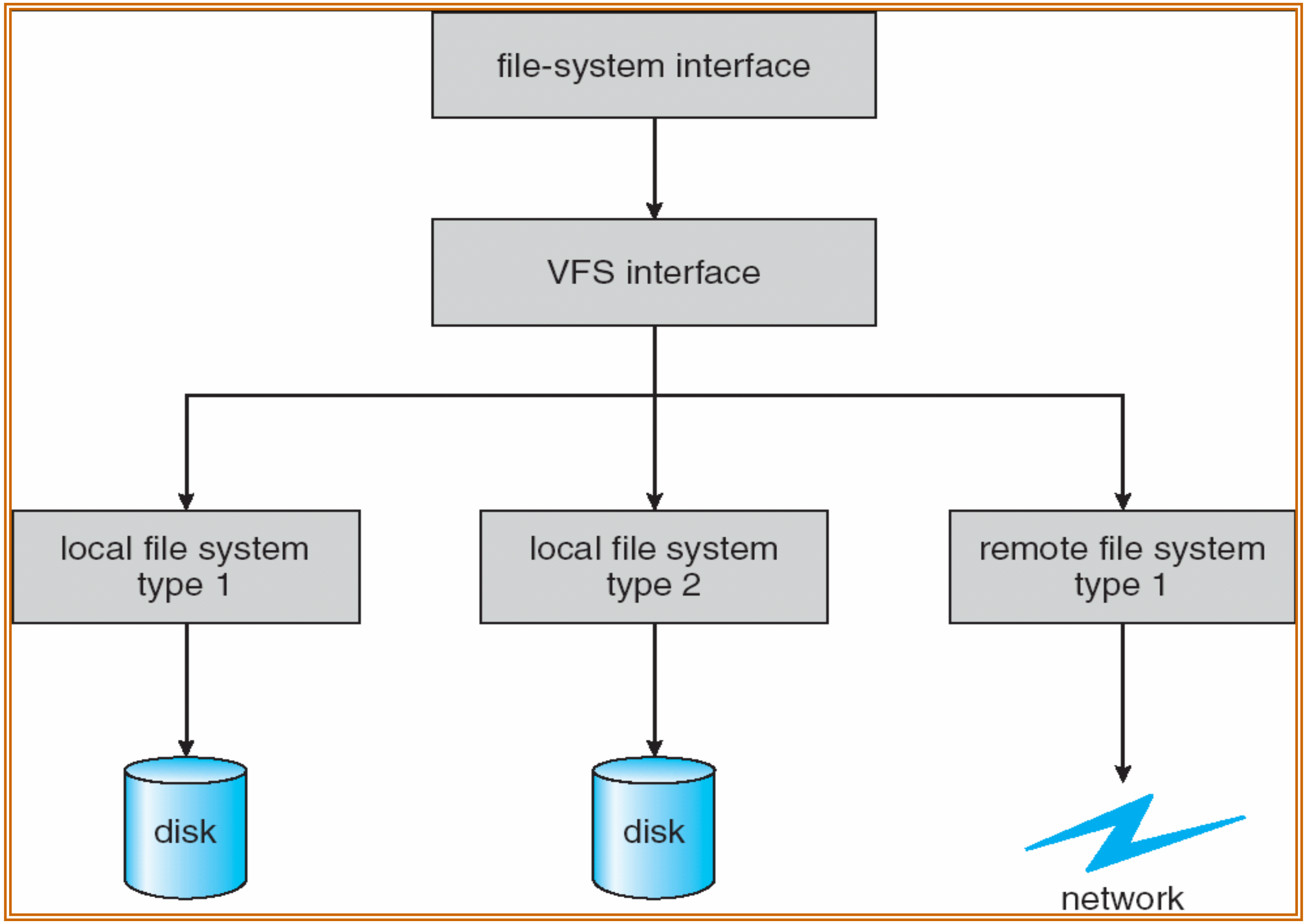
-- Virtual File Systems

- 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



- 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

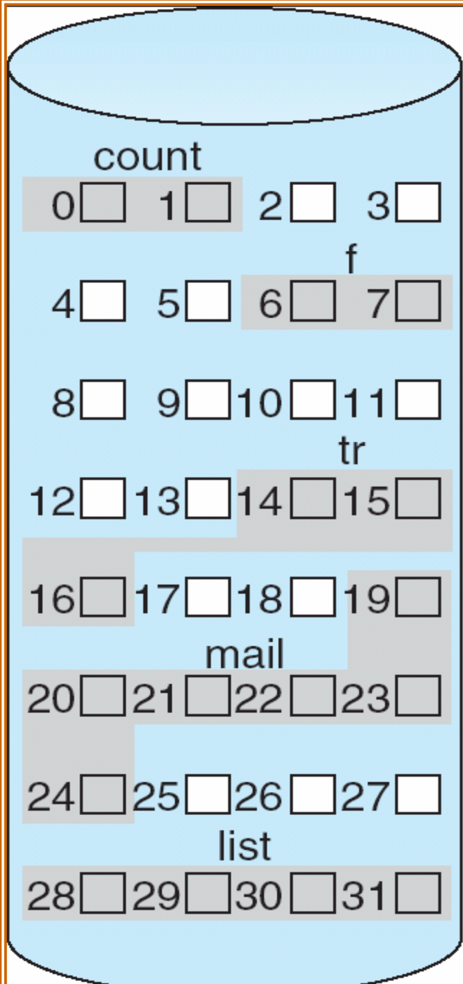
- Allocation Methods

- 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 of Disk Space



directory

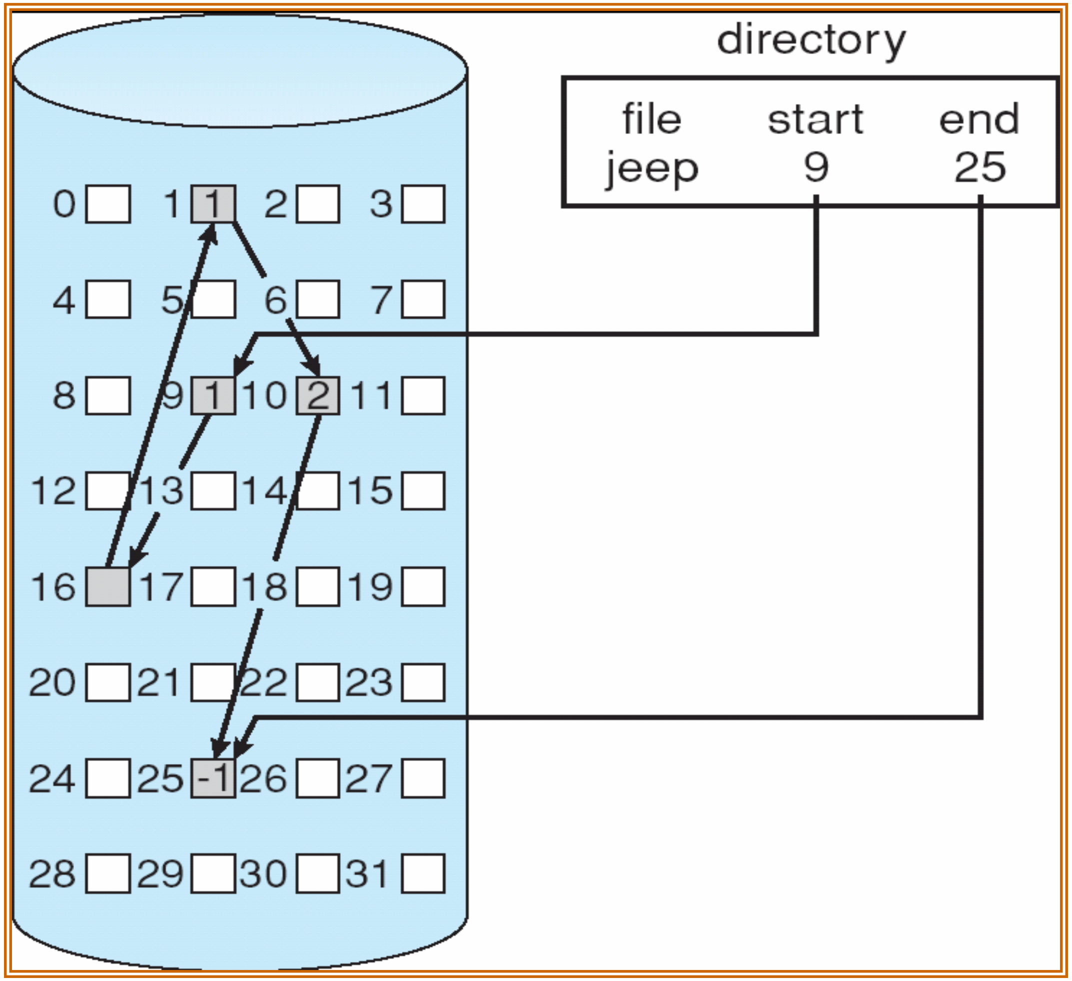
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

-- Extent-Based Systems

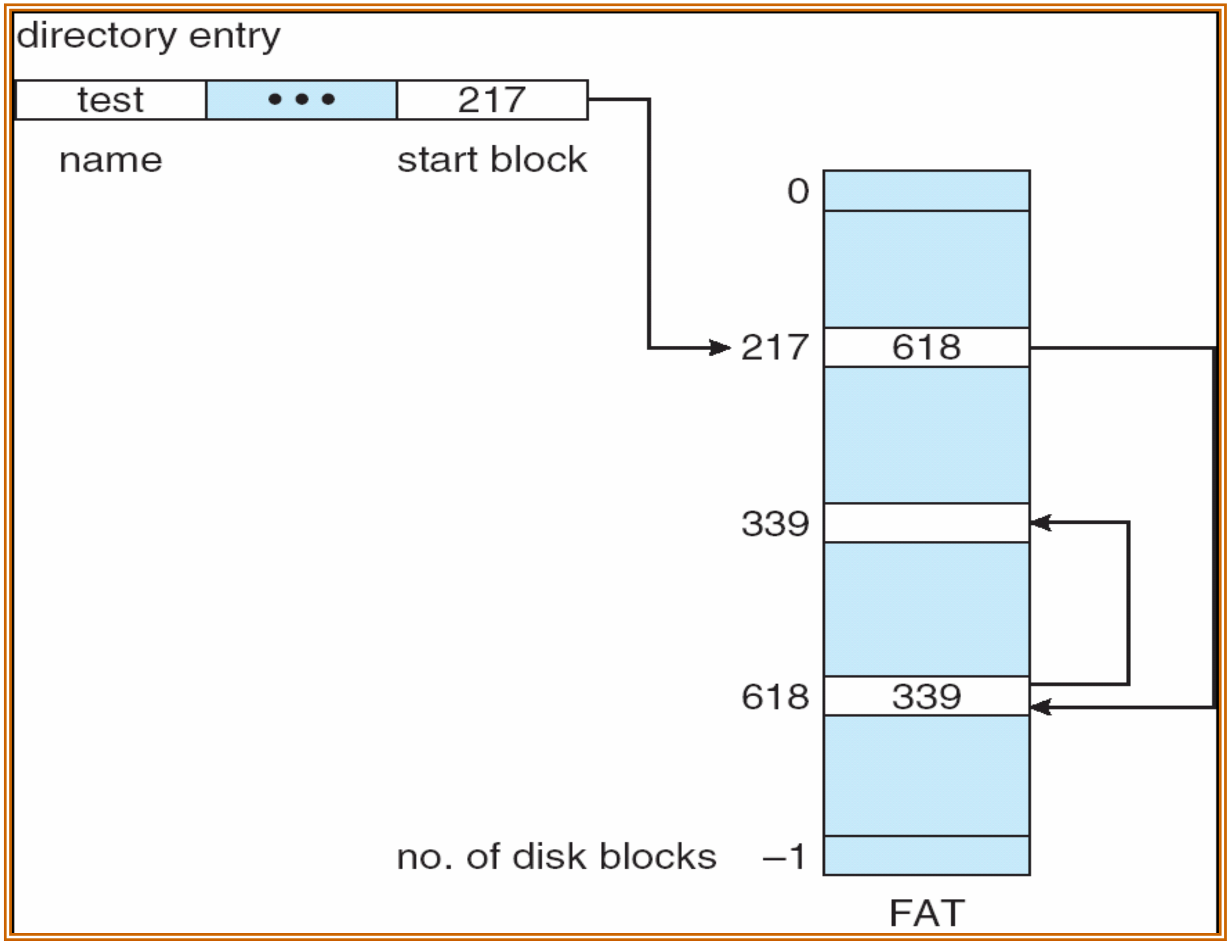
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents.

-- Linked Allocation

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



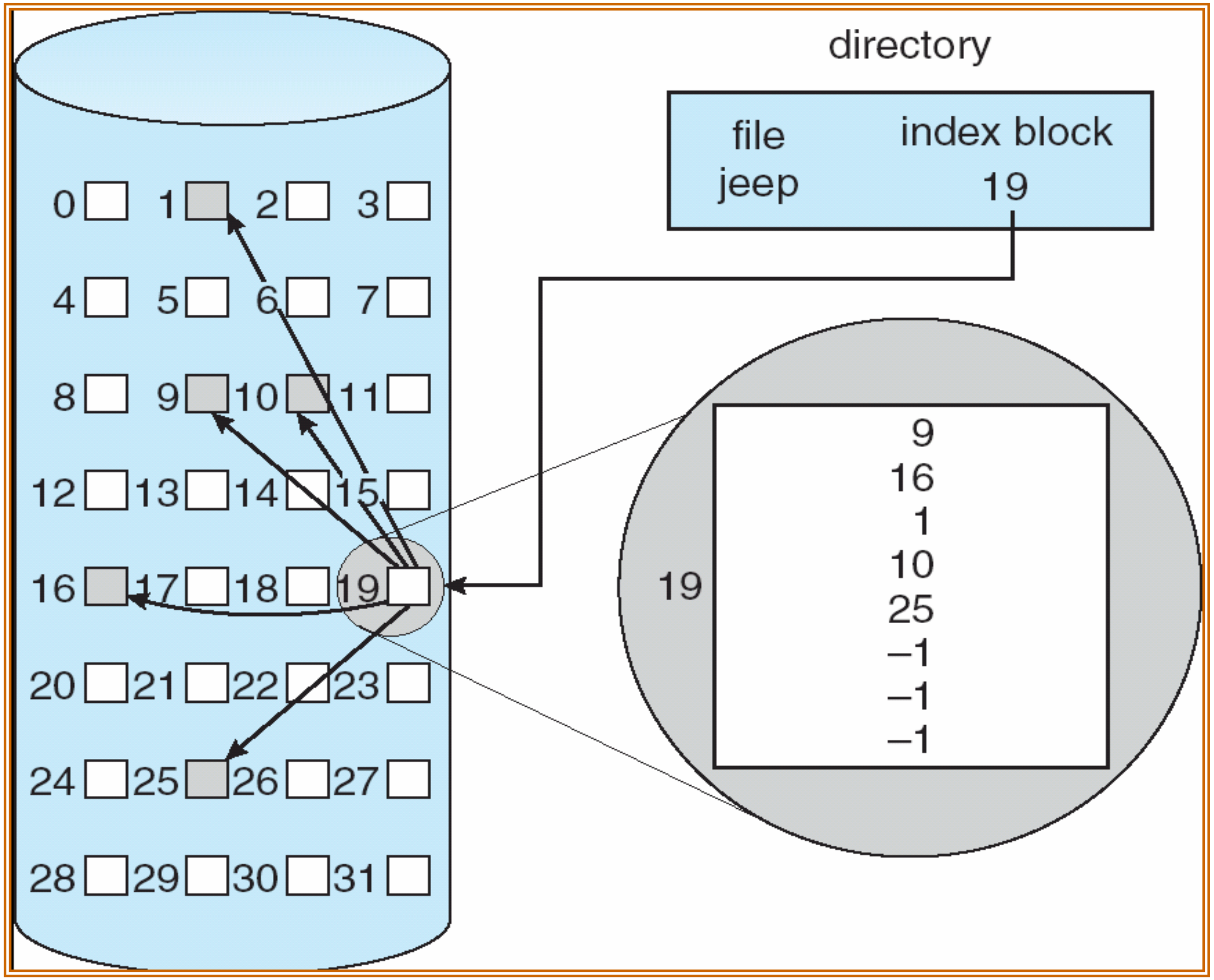
--- File-Allocation Table



-- Indexed Allocation

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.

--- Example of Indexed Allocation

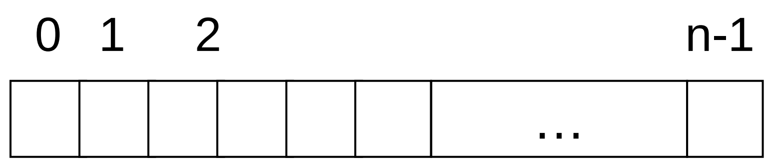


- Free-Space Management

- Bit vector
- Linked free space list
- Grouping
- Counting

-- Bit Vector ...

- 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

... -- Bit vector ...

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

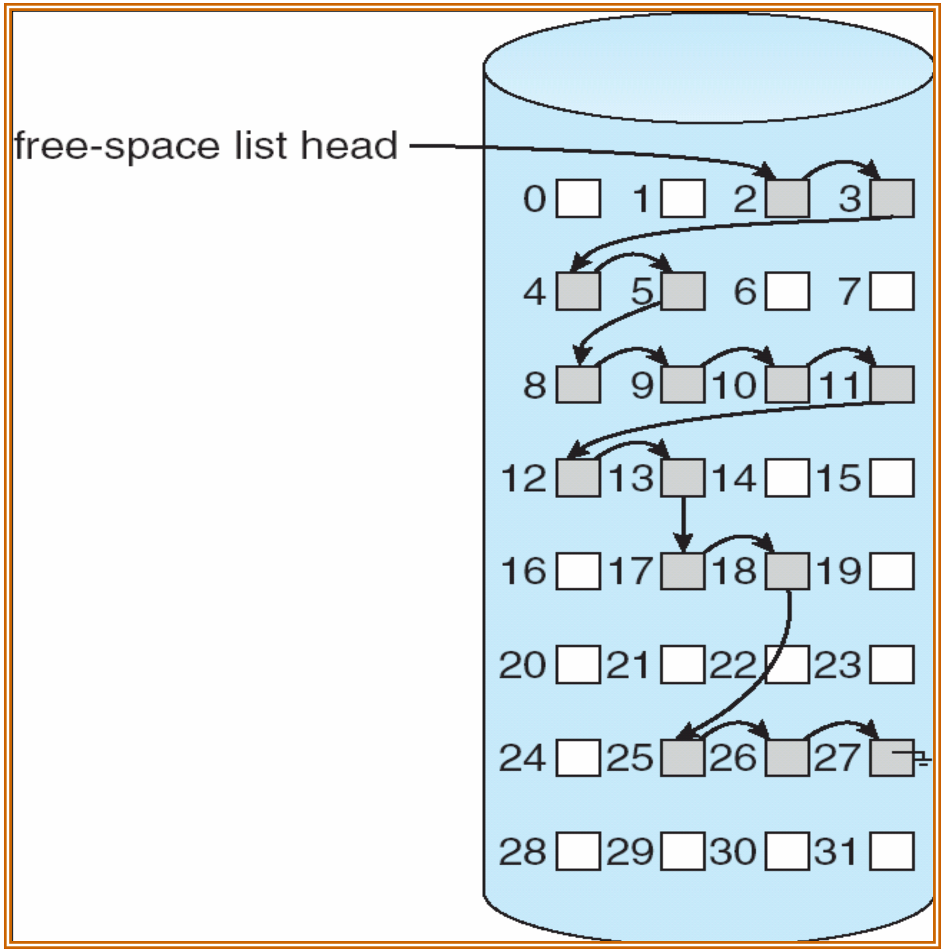
    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

... -- Bit vector

- 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



-- Grouping

- Modification of the linked free-list
- Stores the address of the n free blocks in the first block.
  - The first n-1 of these blocks are actually free
  - The last block contains the address of another n free blocks

-- Counting

- This approach takes advantage of the fact that contiguous blocks are allocated and freed simultaneously.
- Keeps the address of the first free blocks and the number n of free contiguous blocks
- Particularly good if space is allocated using contiguous allocation.

End of Chapter 11