File System Implementation

- File-System Structure
- Directory Implementation
- Allocation Methods
- Free-Space Management

File-System Structure

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

A Typical File Control Block

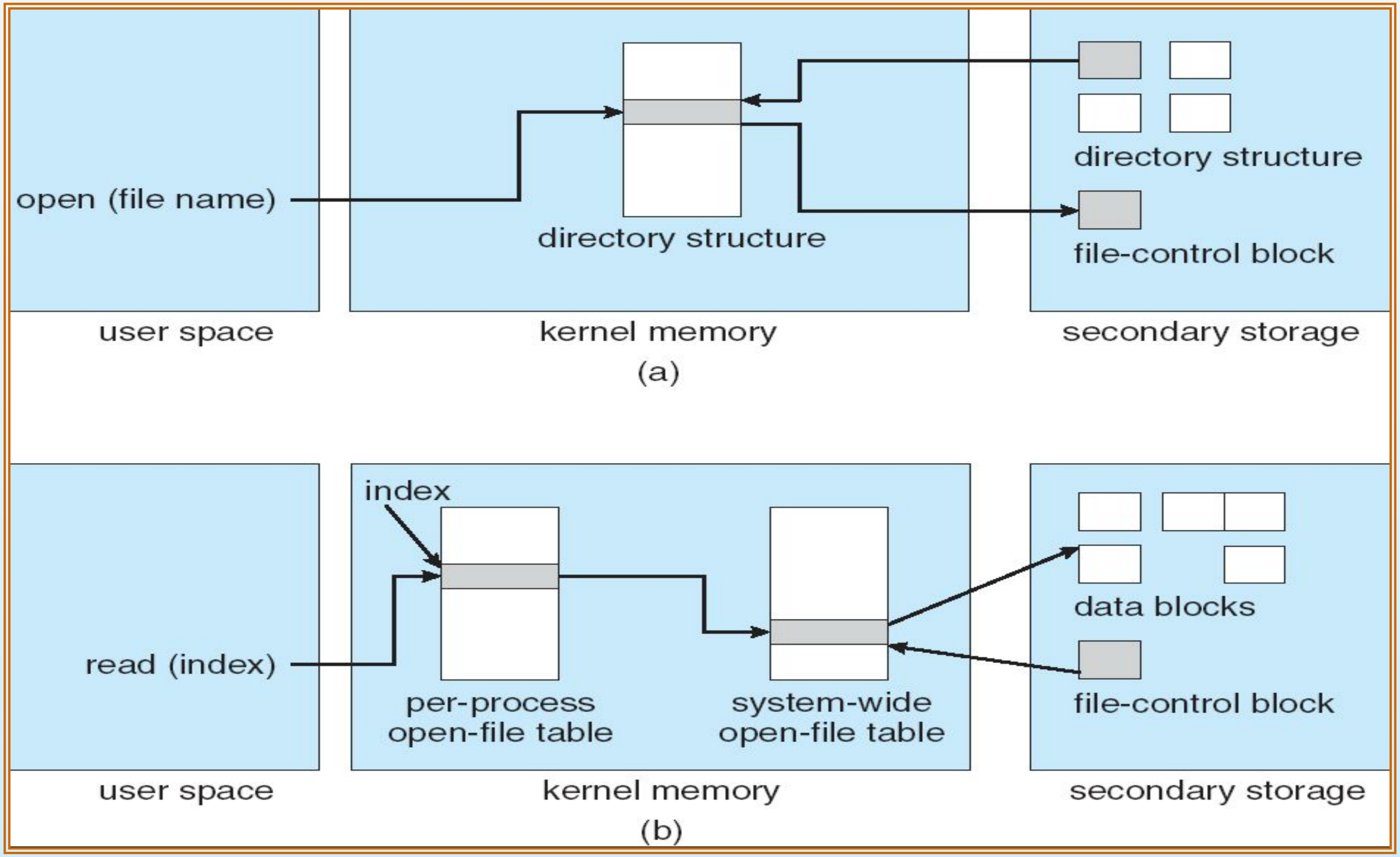
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

The In-memory information is used for both file-system management and performance improvement via caching. It Includes

- An in-memory partition table containing information about each mounted partition
- An in-memory directory structure that holds the directory information of recently accessed directories
- The **system wide open file** contains a copy of the FCB of each opened file as well as other information.
- The **per process open-file table** contains a pointer to the appropriate entry in the system-wide open-file table

In-Memory File System Structures



(a) File open (b) File read

Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program but time-consuming to execute
  - It requires a linear search to find a particular entry.
  - To create a new file, first search the directory to be sure that no existing file has the same name.

- **Hash Table** – linear list with hash data structure.
  - The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location

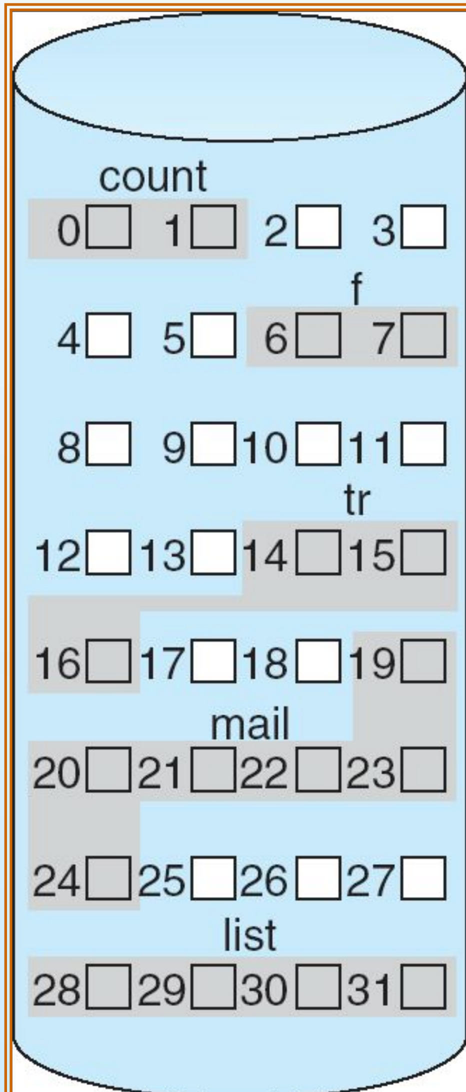
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files so that disk space is utilized effectively and files can be accessed quickly.:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple to access – only starting location (block #) and length (number of blocks) are required
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file
- Both sequential and Random access supported
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow
- The algorithms suffer from external fragmentation.

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 use a modified contiguous allocation scheme

- Extent-based file systems a contiguous chunk of space is allocated initially, and then, when that amount is not enough, another chunk of continuous space is added to the initial allocation.

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

Linked Allocation

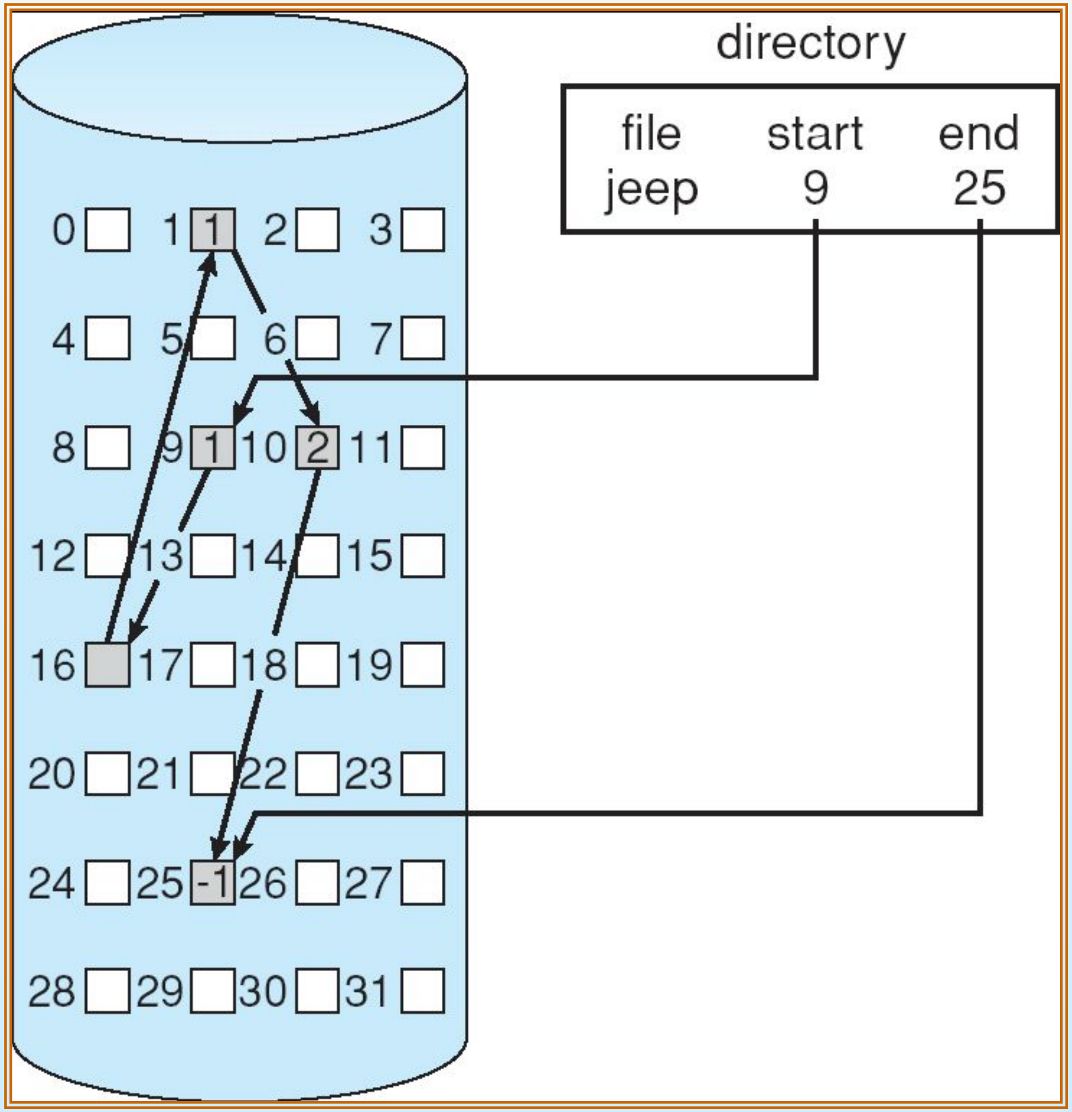
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.

- The directory contains a pointer to the first and last block of the file.

Linked Allocation (Cont.)

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

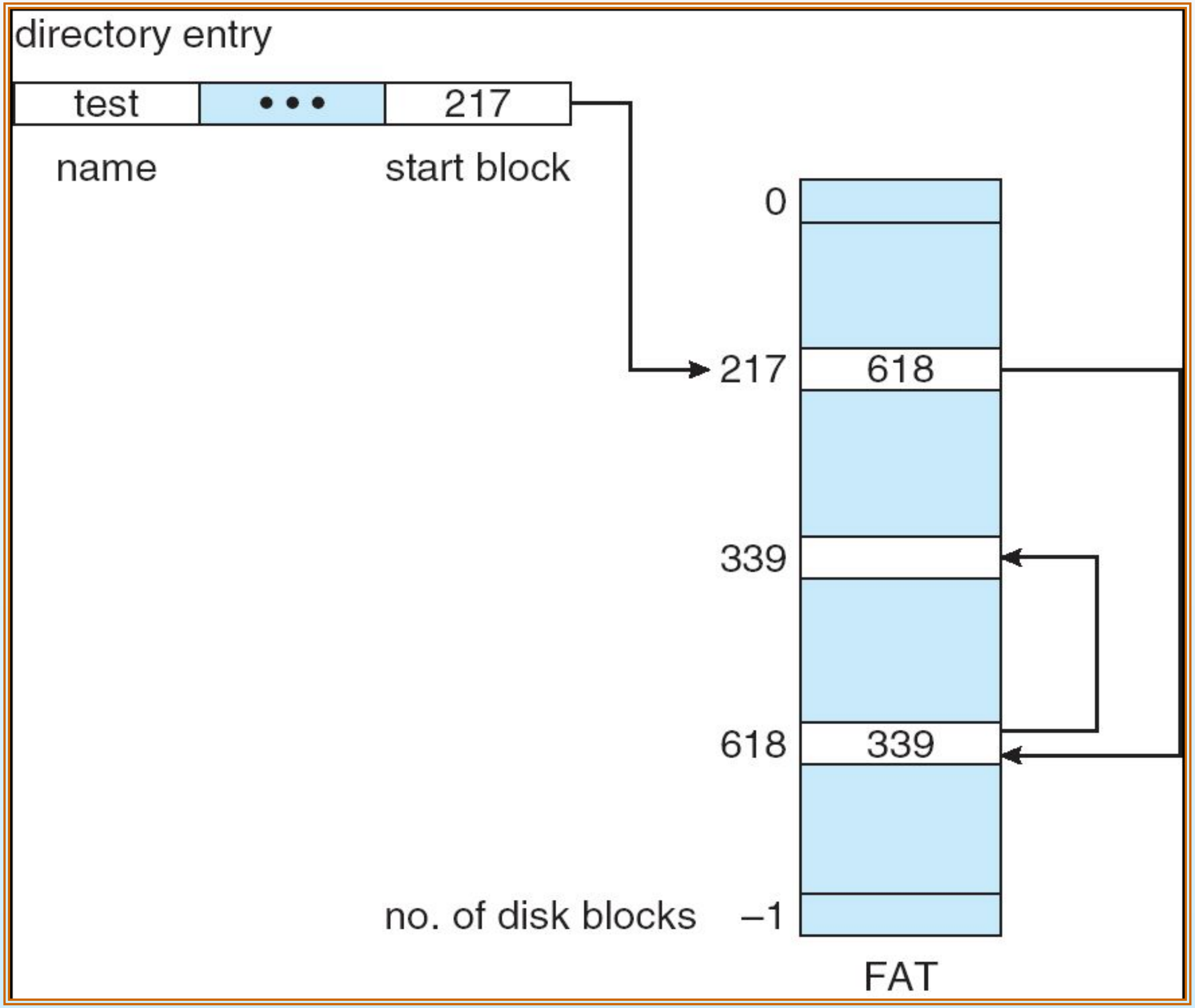
Linked Allocation



File-Allocation Table

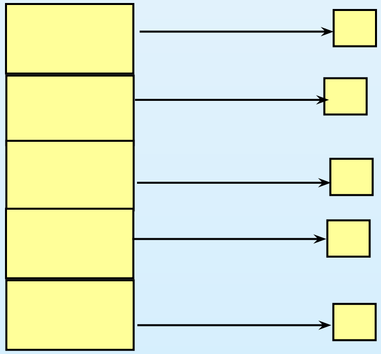
- An variation on the linked allocation method is to use of a file-allocation table (FAT)
- A section of disk at the beginning of each partition is set aside to contain the table.
- The table has one entry for each disk block and is indexed by block number
- The directory entry contains the block number of the first block of the file
- The table entry indexed by that block number is then contains the block number of the next block in the file
- This chain continue till the last block which has a special end-of-file value as the table entry.

File-Allocation Table



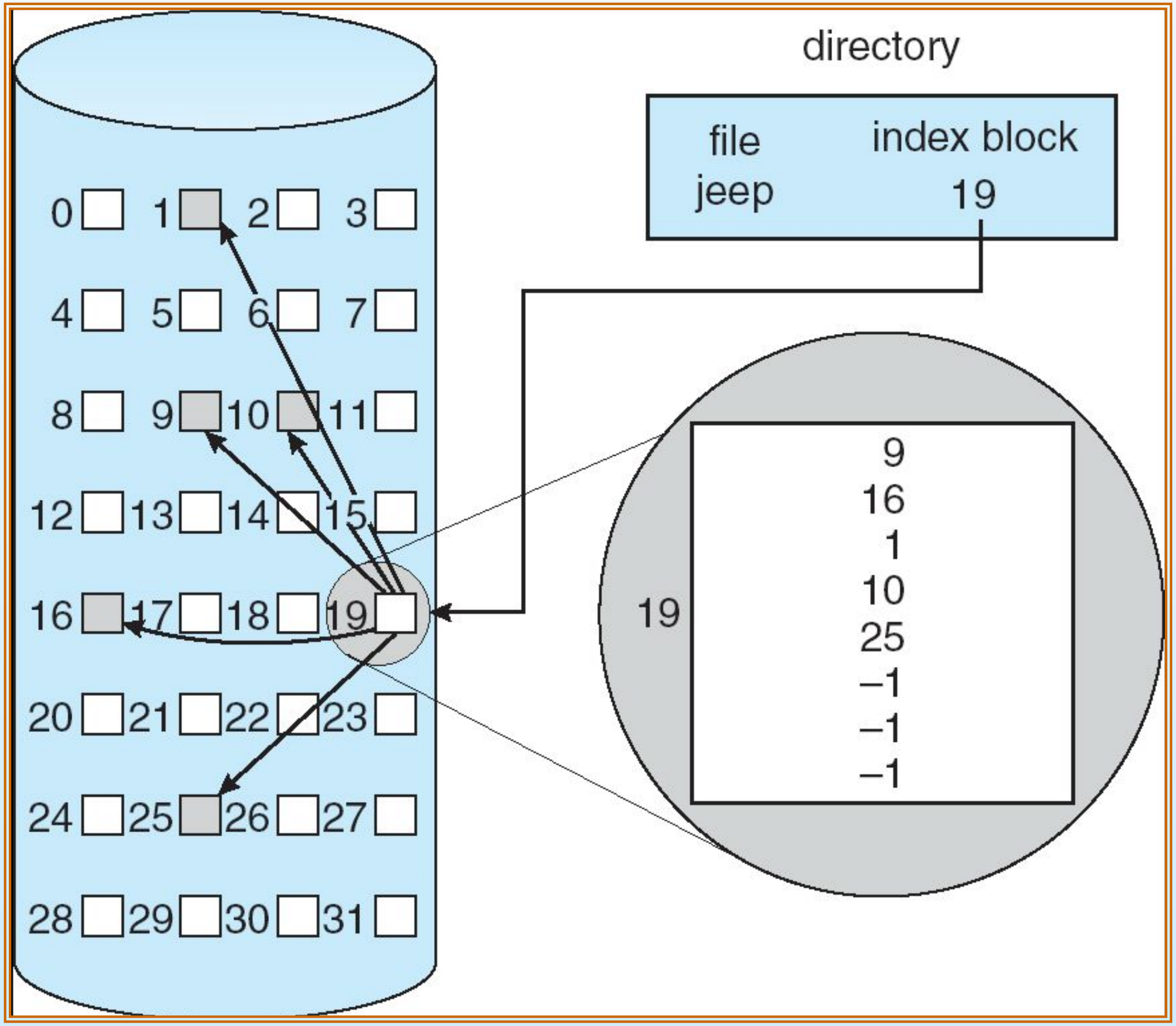
Indexed Allocation

- *Each file has its own index block.*
- Brings all pointers together into the *index block*.
- *The directory contains the address of the index block.*



index table

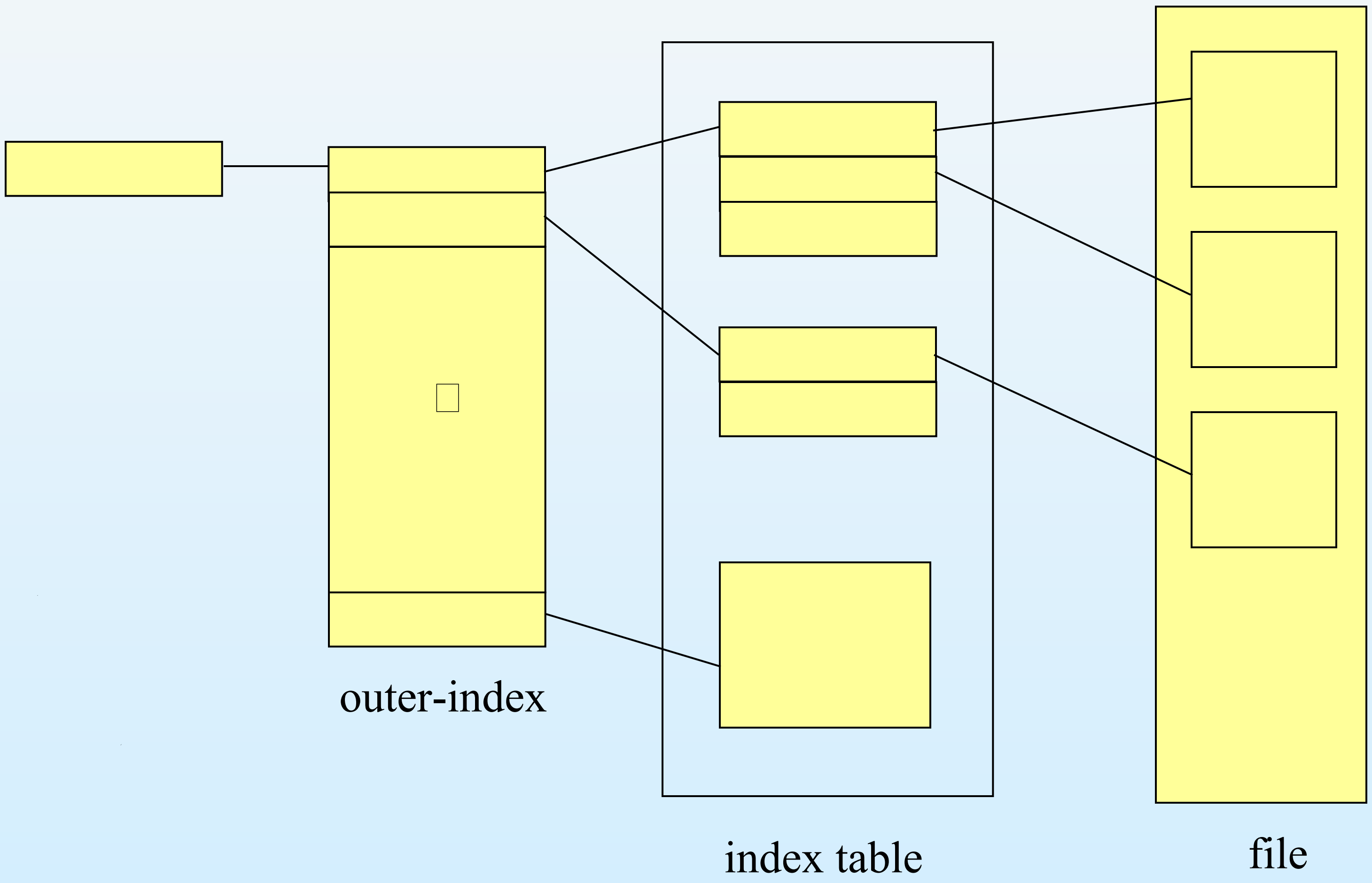
Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Indexed allocation suffer from wasted space. Pointer overhead of index block is generally greater than the pointer overhead of linked allocation
- With indexed allocation, an entire index block must be allocated, even if file size is of one or two blocks.
- For larger file two or more index block may be assigned, linked together.
- Even multilevel indexing may be allowed similar to multilevel paging.

Indexed Allocation – Mapping (Cont.)



Free-Space Management

- To keep track of disk space, the system maintains a free space list which keep records of all free blocks
- When a new file is created it is assigned free block from free apace list. Similarly when a new file/directory is deleted its block are added to free space list.

Free-Space Management

- **Bit Vector (*n* blocks)**
- Each block is represented by 1 bit. If the block is free, the bit is 1; if allocated then bit is 0

e.g. if disk blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17 are free and rest are allocated then the bit vector would be:

001111001111110001000000000-------------

- If bit vector is large and require multiple blocks then

- 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:

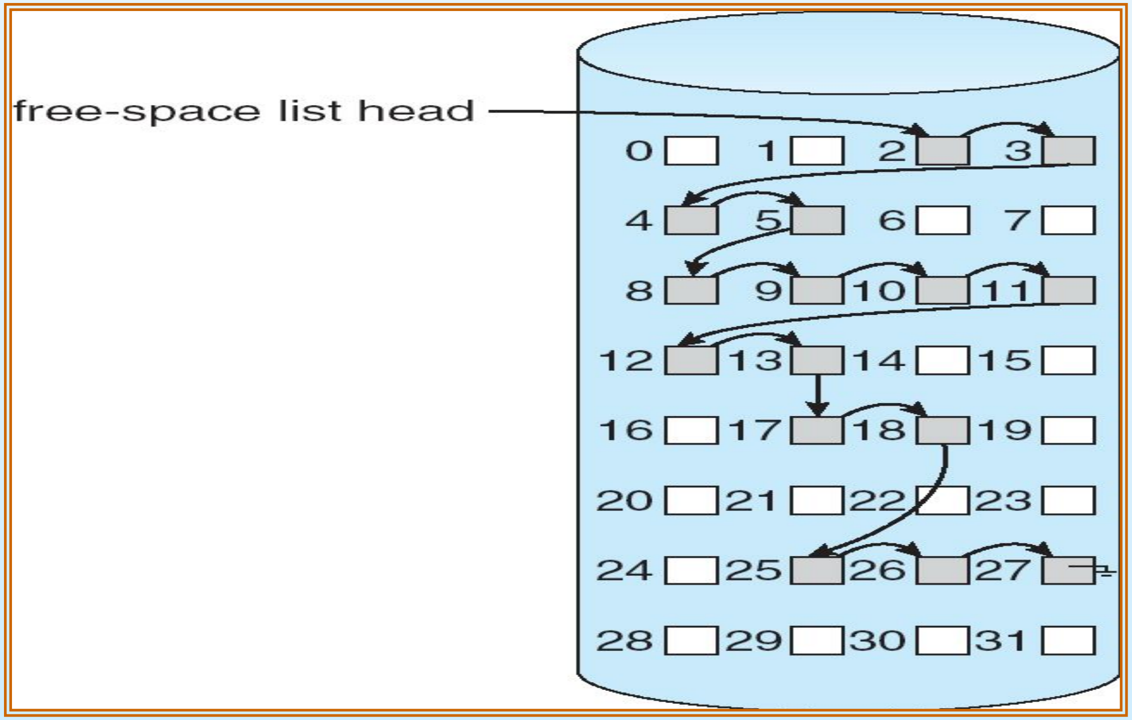
block size = 2^{12} bytes

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

$n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

Free-Space Management (Cont.)

- **Linked List**
  - Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.



Free-Space Management (Cont.)

- **Grouping**
  - Store the addresses of n free blocks in the first free block.
  - The first n-1 of these block are actually free.
  - The last block contains the addresses of another n free blocks and so on.
  - The importance of this implementation is that the addresses of a large number of free blocks can be found quickly, unlike in the standard linked list approach.

Free-Space Management (Cont.)

- **Counting**
  - By taking advantage of the fact that, generally several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous allocation algorithm or through clustering.

  - Thus rather than keeping a list of n free disk addresses we can keep the address of the first free block and the number n of free contiguous block that follow the first block.