Secondary Storage Management

File System Implementation

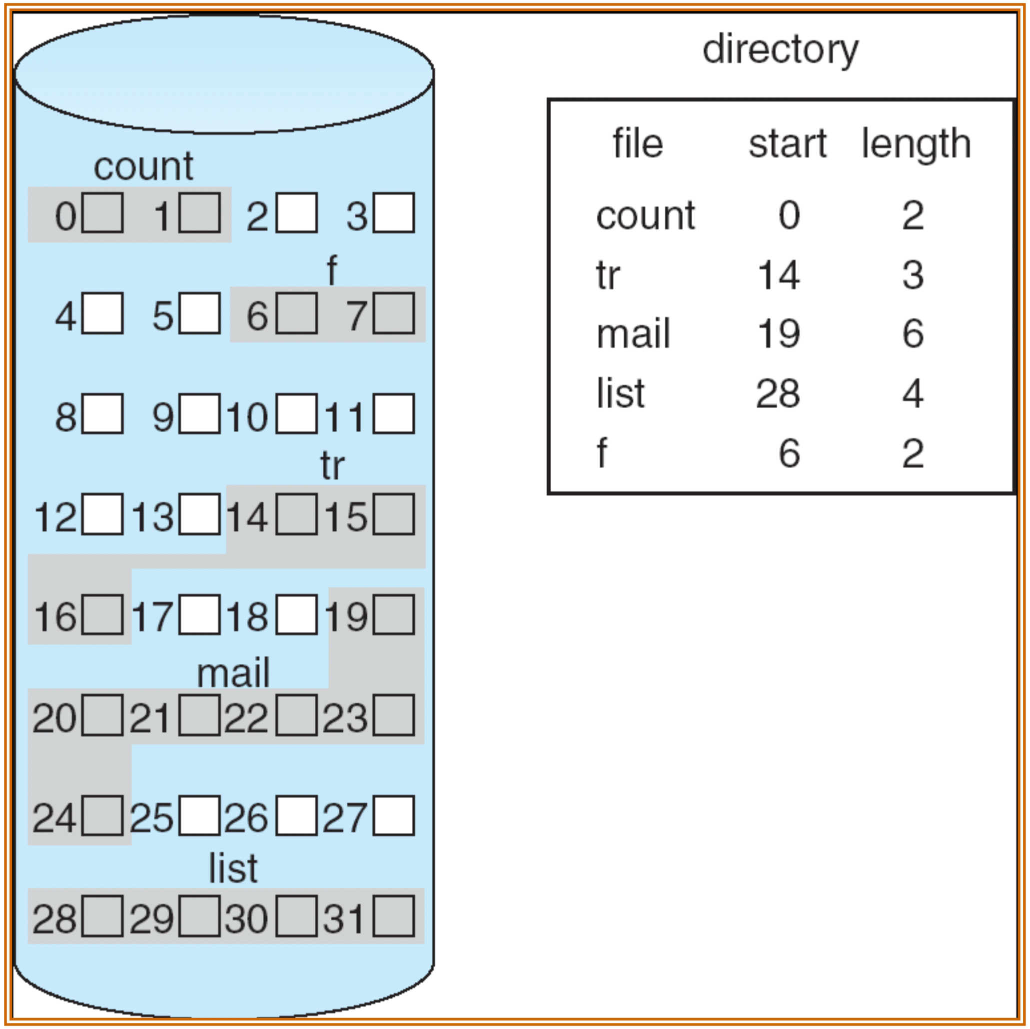
Dr. Manmath N. Sahoo
Dept. of CSE, NIT Rourkela

Allocation Methods

- Allocation methods address the problem of allocating space to files so that disk space is utilized effectively and files can be accessed quickly
- Three methods exist for allocating disk space
  - **Contiguous allocation**
  - **Linked allocation**
  - **Indexed allocation**

Contiguous Allocation

- Requires that each file occupy a set of contiguous blocks on the disk
- Accessing a file is easy

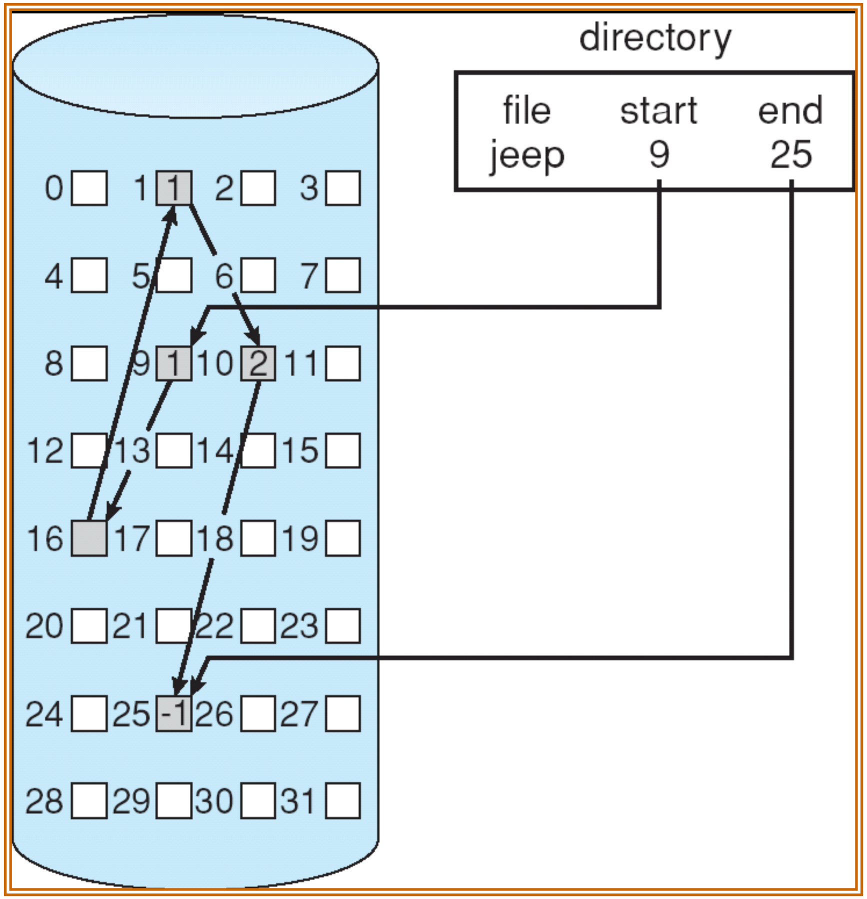


Contiguous Allocation: Issues

- Internal fragmentation
- External fragmentation
  - Need for compaction, which is time consuming
- Finding space for a new file (first fit, best fit, etc.)
- Determining space for a file, especially if it needs to grow
  - Allocating too little space – if file grows then copy it to a larger free hole – time?
  - Allocating too much space – if the file doesn't grow – large number of smaller holes
  - Even if we know the size of the file prior to the allocation, pre-allocation may be inefficient – file may grow very slowly over a long period of time

Linked Allocation

- Solves the problems of contiguous 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 blocks of a file



Linked Allocation

- **Creating** a new file requires only creation of a new entry in the directory
- **Writing** to a file causes the free-space management system to find a free block
  - This new block is written to and is linked to the end of the file
- **Reading** from a file requires only reading blocks by following the pointers from block to block

Linked Allocation

Advantages

- There is no external fragmentation
- Any free blocks on the free list can be used to satisfy a request for disk space
- The size of a file need not be declared when the file is created
- A file can continue to grow as long as free blocks are available
    - It is never necessary to go for compaction

Linked Allocation

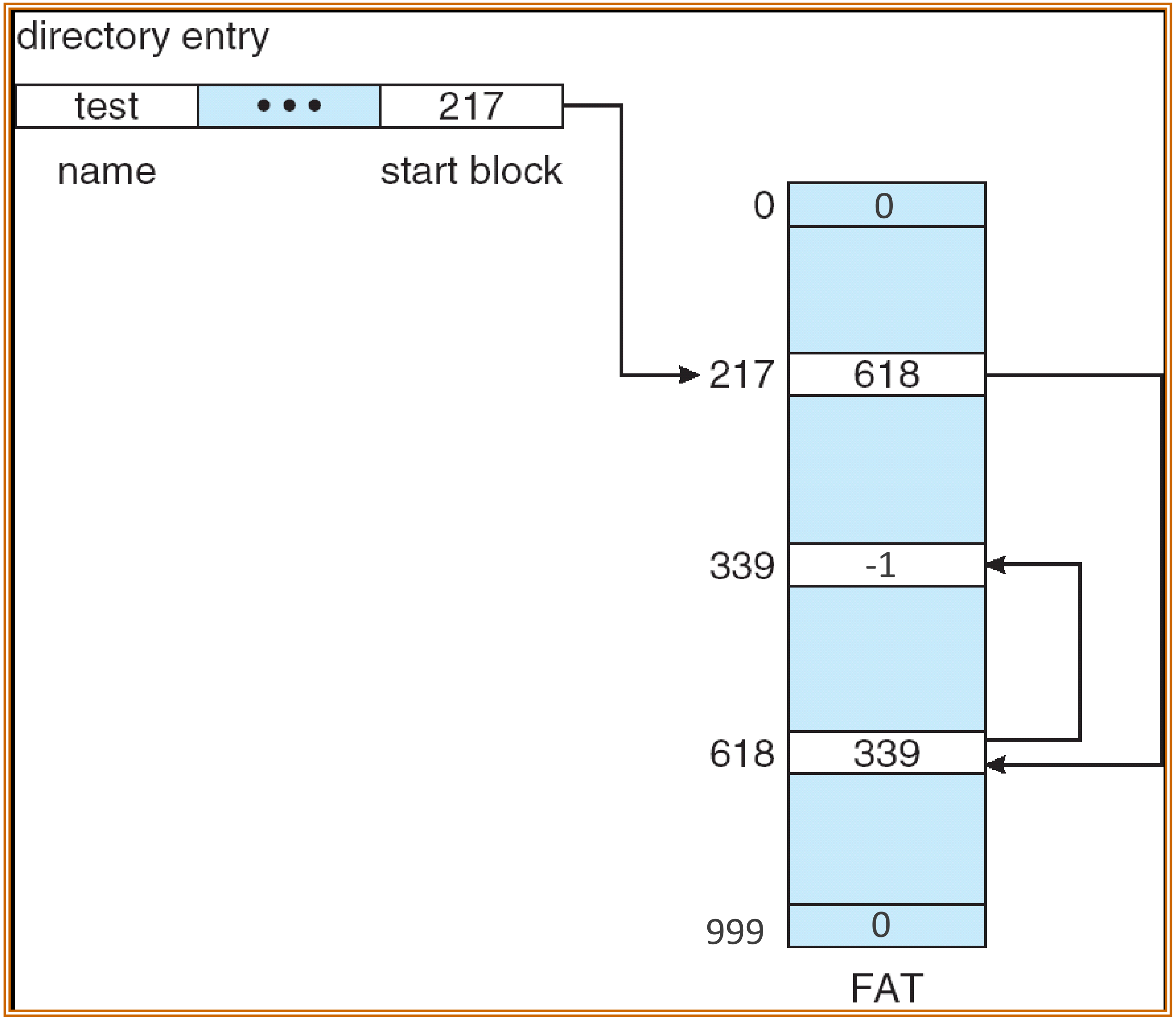
Disadvantages

- Can only be used effectively for <u>sequential access</u> of files
  - It is inefficient to support <u>direct access</u> capability
- Disk space is required to store the block pointers
  - One solution is the clustering of a certain constant number of blocks (e.g., 4) – internal fragmentation may be high
- Relies on the integrity of the links – an error might result in a pointer value becoming corrupt and then pointing into the free-space list or to the blocks of another file

Variant of Linked Allocation

- File Allocation Table (FAT)
  - The FAT 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 the block number contains the block number of the next block in the file
  - The last block contains a special end-of-file value as the table entry
  - Unused blocks are indicated by a zero table value
  - To allocate a new block to a file
    - Find the first zero-valued table entry
    - Replace the previous end-of-file value with the address of the new block

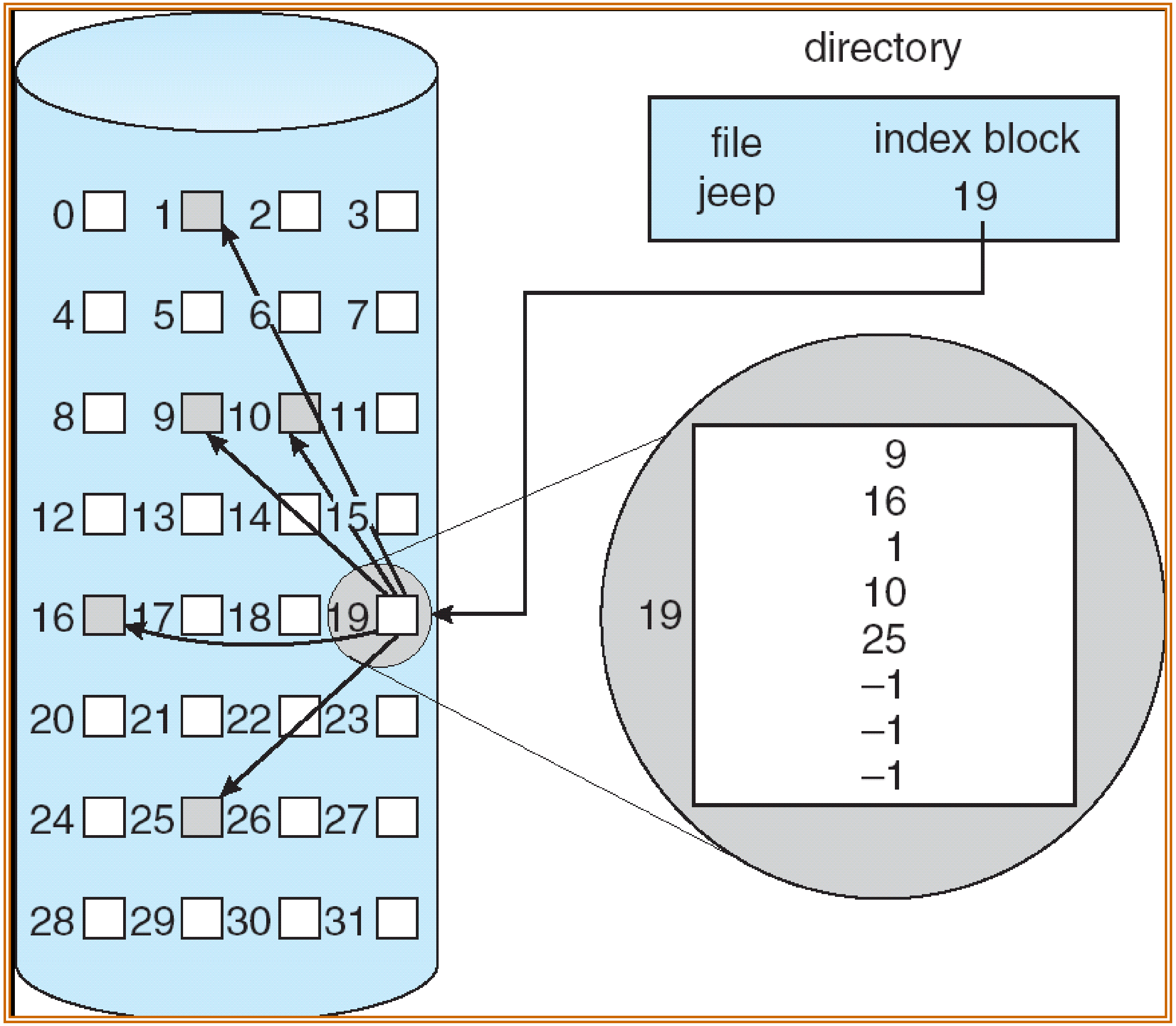
File Allocation Table (FAT)



Indexed Allocation

- Solves the problems of linked allocation by bringing all the pointers (for a file's blocks) together into one location called the *index block*
- Each file has its own index block, which is an array of pointers to disk blocks
- Each entry in the index block points to the corresponding block of the file
- The directory contains the address of the index block
- **Advantage**: Supports direct access without suffering from external fragmentation
- **Disadvantage**: Pointer overhead is more

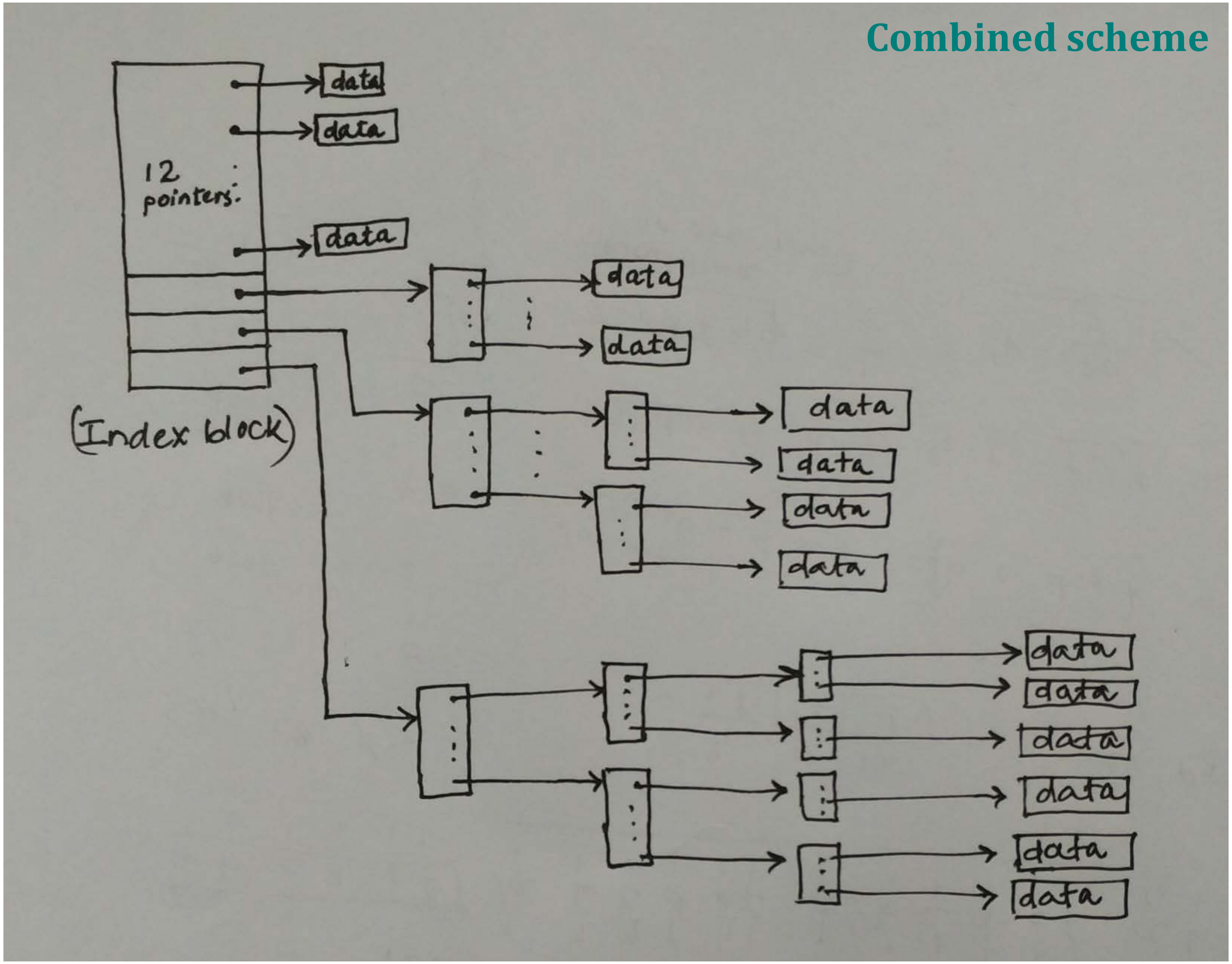
Indexed Allocation



Sizing of the Index Block

- Approach #1: Linked scheme
  - Several index blocks can be linked together
- Approach #2: Multilevel index scheme
  - A first-level index block points to a set of second-level index blocks
- Approach #3: Combined scheme
  - Used in the UNIX file system
  - A specific set of pointers points directly to file blocks
  - Three special pointers
    - 1st points to a <u>single</u> indirect block,
    - 2nd points to a <u>double</u> indirect block, and
    - 3rd points to a <u>triple</u> indirect block

Sizing of the Index Block

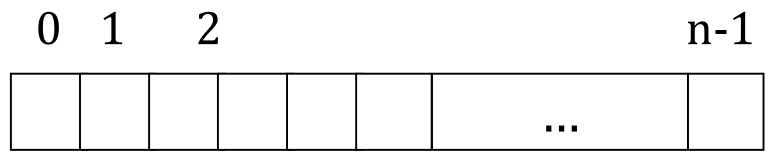


Free-Space Management

- Bit Vector Approach
- Linked List Approach
- Grouping
- Counting

Bit Vector Approach

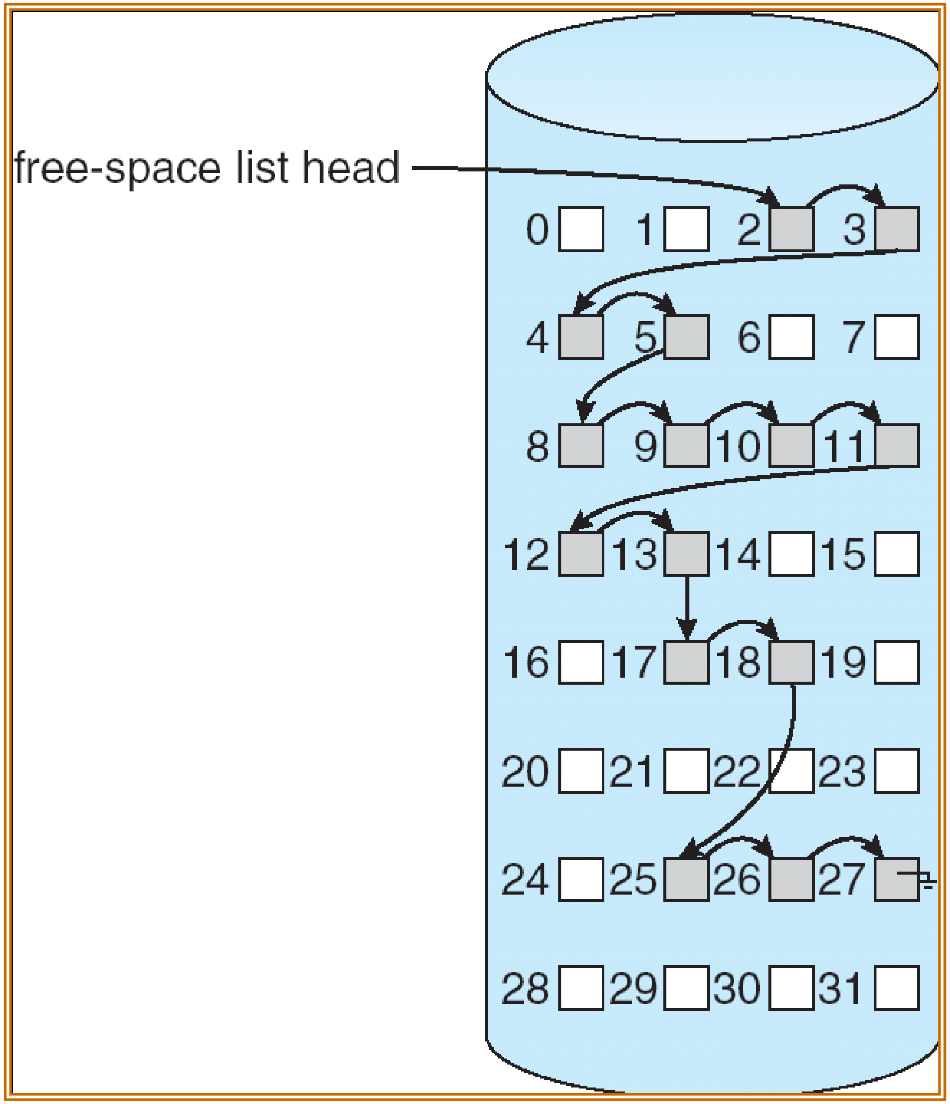
- Free-space blocks are tracked via a bit vector (where n = #blocks)



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

- \+ Simple to implement
- \+ Easy to find free blocks
- \- 40GB disk with 1KB block requires 5MB of bit-vector
- \- why to store the information about allocated blocks!!

Linked List (free list) Approach



Grouping (modification of free list)

- First free block stores the addresses of next ***n*** free
- The first ***n–1*** of these blocks are actually free
- The last block contains the addresses of another ***n*** free blocks

Counting

- Take advantage of the fact that several contiguous blocks may be allocated or freed simultaneously
- No. of entries in the free space table is the number of free holes
- Keep the address of the first free block and the number ***n*** of free contiguous blocks that follow the first block