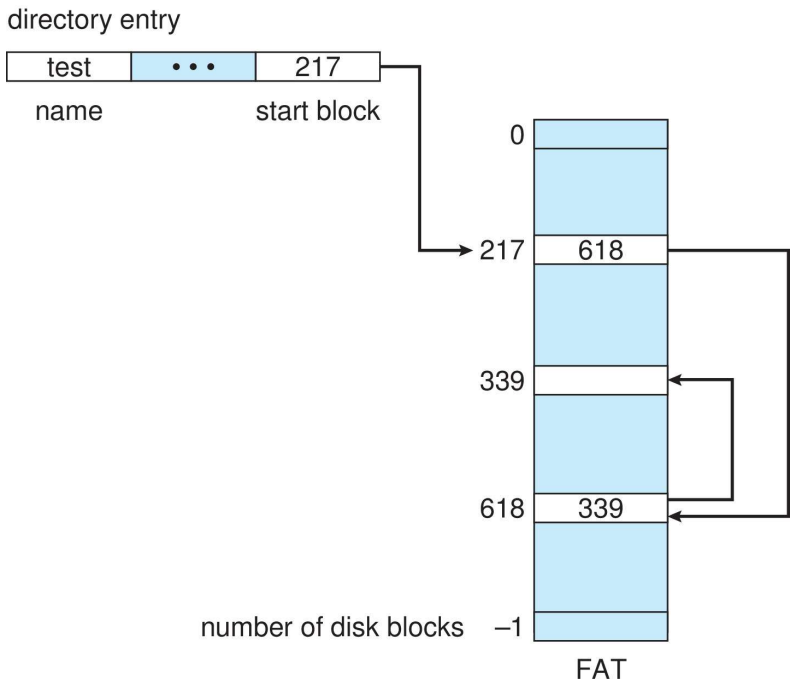
Allocation Methods - Linked

- FAT (File Allocation Table) variation
- Simple but efficient method for disk space allocation – used by MS-DOS operating system.
- A section of storage at the beginning of each volume is set aside to contain the table.
- Table has one entry for each block and index by block number.
- Directory entry contains the block number of the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- Chain continues until it reaches the last block(table value of 0).
- Allocating a new block to a file is a matter of finding the first 0-valued table entry and replacing the previous end of file value with the address of the new block.

FAT allocation scheme

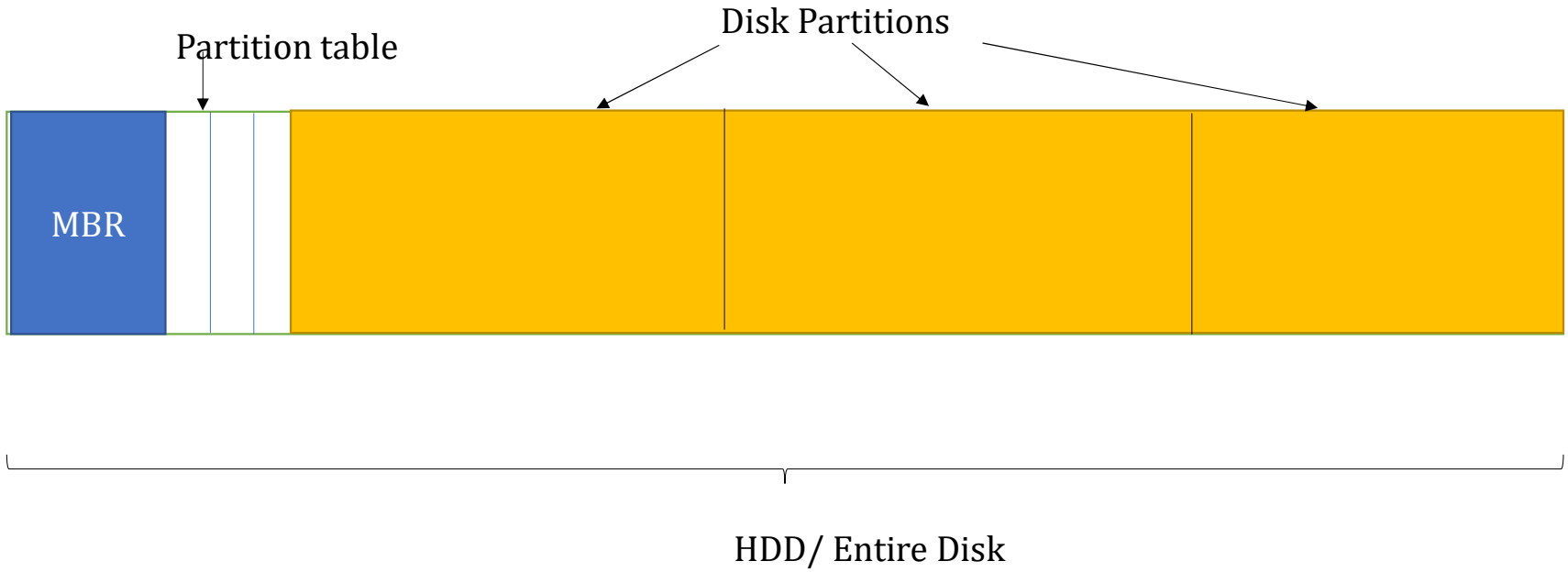
- FAT was developed by MS to support small disks and simple folder structures.
- Uses table to track the clusters on storage volume, clusters are linked together through their associated files and directories.
- Can result in a significant number of disk head seeks, unless FAT is cached.
  - Disk head must move to the start of the volume to read the FAT and find the block location. Then move to the location of the block.

File-Allocation Table

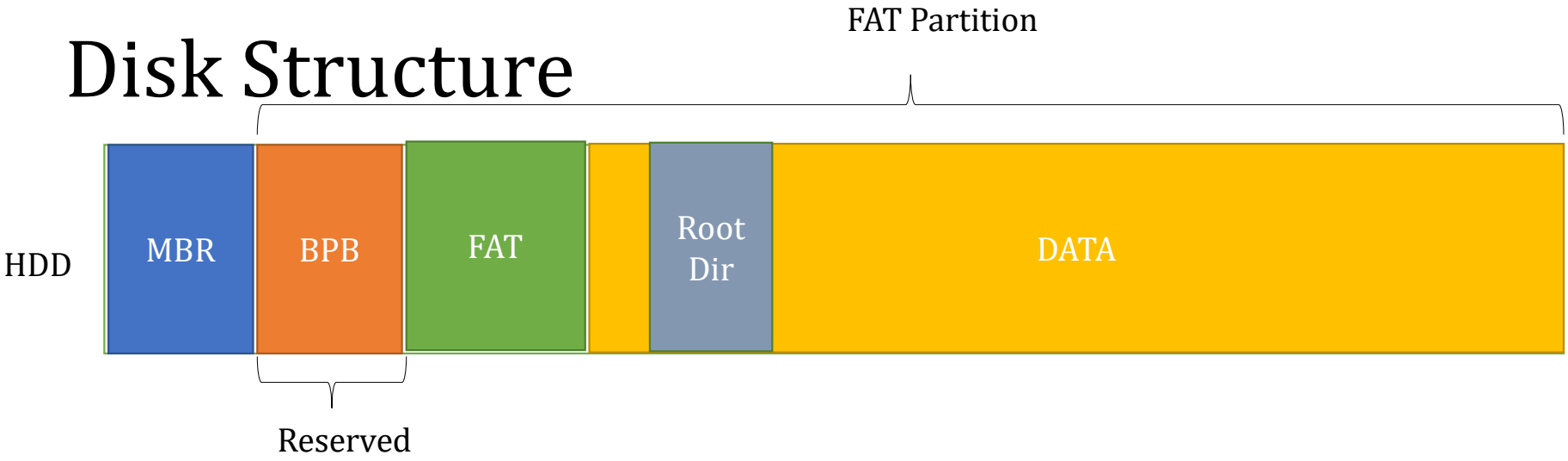


Master Boot Record (MBR)

- Information present in the first sector of any hard disk.
- Contains information to boot the OS in the RAM.



Disk Structure



- Master Boot Record – 1st sector of the device, consist of,
  - Boot Loader Program
  - Offset values to access each partition
- FAT Partition
  - Reserved section
    - Contains BIOS parameter block (BPB) – first sector of the partition

Allocation Methods – Linked (Cont.)

- FAT (File Allocation Table) variation
  - Much like a linked list, but faster on disk and cacheable
  - FAT version refers to #bits user to each entry in allocation table.
  - A volume formatted with FAT is allocated in clusters.
    - Cluster size is determined by the volume size
    - FAT16 contains 2 bytes per cluster within the file allocation table
    - FAT32 contains 4bytes per cluster within the file allocation table

Attribute	FAT12	FAT16	FAT32
Used For	Floppies; small hard drives	Small to large hard drives	Large to very large hard drives
Size of Each FAT Entry	12 bits	16 bits	28 bits
Maximum Number of Clusters	~4,096 2^{12}	~65,536 2^{16}	~268,435,456 2^{32}
Supported Cluster Sizes	512 B to 4 KB	2 KB to 32 KB	4 KB to 32 KB
Maximum Volume Size	16,736,256 B (16 MB)	2,147,123,200 B (2 GB)	~2^{41} B (2 TB)

Allocation Methods – Linked (Cont.)

- How many clusters are in FAT32 ?
  - 2^{32} = 4294967296 clusters
- Suppose you are given 2 GB file partition, calculate the cluster size of each of the file system (FAT16 and FAT32)

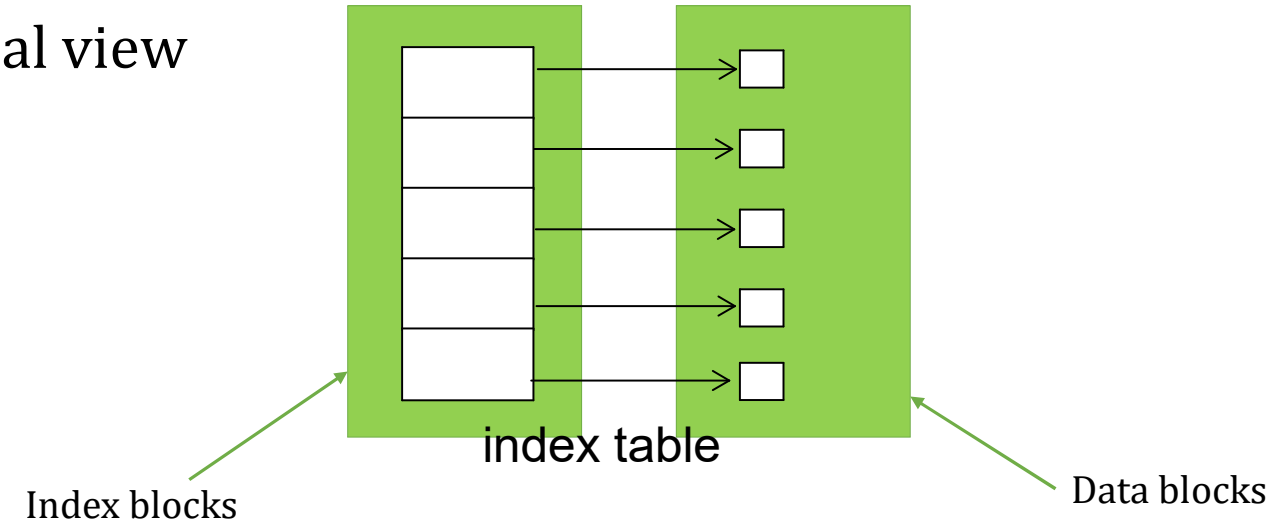
2GB = 2*1024*1024*1024 bytes = $2x2^{10}x2^{10}x2^{10} = 2x2^{30} = 2^{31}$

  - FAT16 (means 2^{16} clusters) = $2^{31}/ 2^{16} = 2^{5}$B =32 KB
  - FAT32 (means 2^{32} clusters) = $2^{31} / 2^{32} = 2^{-1}$B = = 0.5 B

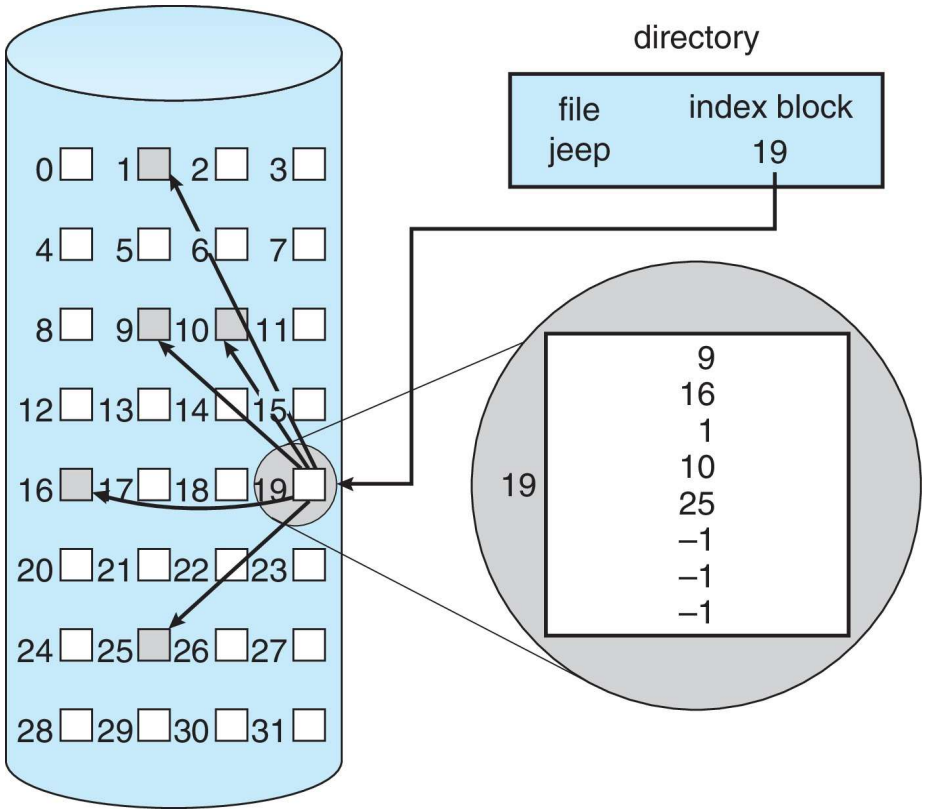
Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

- Logical view



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table for every file
- Random access
- Dynamic access without external fragmentation but have overhead of index block. Each file needs one index table.
- Number of entries in the index block depends on the size of blocks, and size of block address (pointer). Assume 512 byte blocks and block address is 4 bytes, index block could hold references to 512/4 = 128 blocks.
- Maximum size of a file would be 128 x 512 B = 64KB. We need only 1 block for index table.
- If all files are 64KB in a file system, we spend $1/128^{th}$ (4/512) of disk space for index blocks. (limitation)
- Limitation: maximum file size is limited by the block size

Indexed Allocation (Cont.)

- What would be the maximum file size that you can store indexed scheme for FAT32
  - What if block size is 32KB and pointer address size is 4B.

    #Index block references = 32 x 1024 / 4 = 8x1024

    Max. file size = 8x1024 x 32KB = 256 MB
  - What if block size is 64KB and pointer address size is 4B.

    #Index block references = 64 x 1024 / 4 = 16x1024

    Max. file size = 16x1024 x 64KB = 1GB

Indexed Allocation (Cont.) – performance

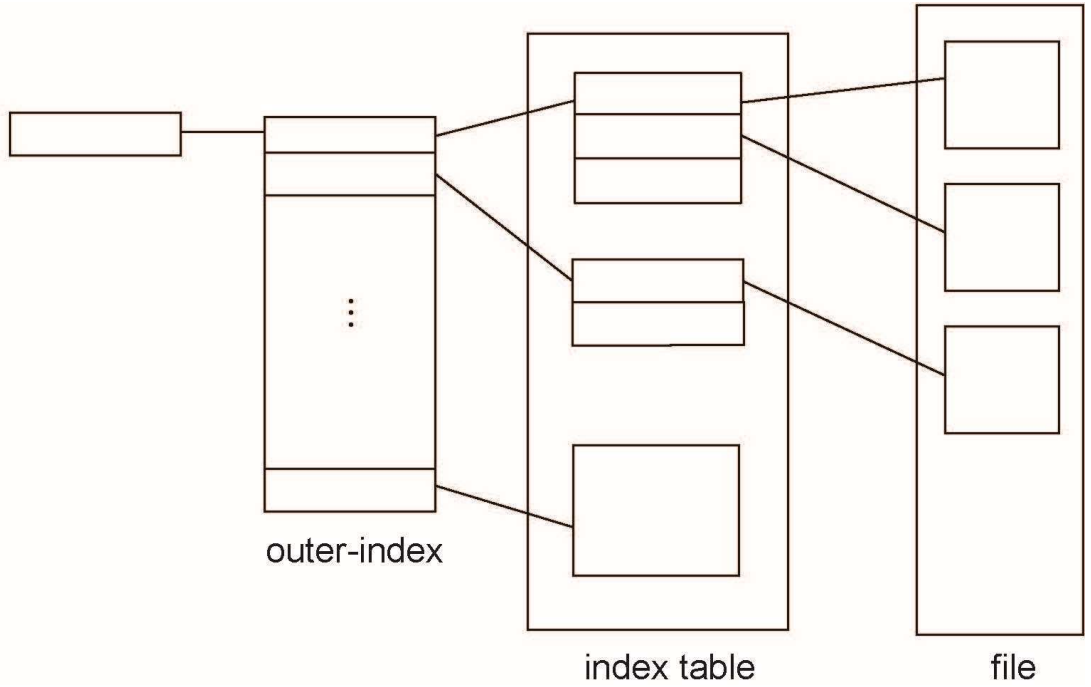
- Index allocation supports direct access
  - Without suffering from external fragmentation, - any free block on the storage device can satisfy a request for more space
  - Suffers from wasted space – pointer overhead is greater than pointer overhead of linked allocation.
  - Eg: consider we have a file of only 1-2 blocks
    - With linked allocation – we lose space only one pointer per block.
    - With indexed allocation – an entire index block must be allocated even only 1-2 pointers will be non-null.
- Index blocks can be cached in memory, but the data blocks may be spread all over a volume.

Indexed Allocation (Cont.) – performance

- How large the index block should be ?
  - Every file must have an index block, so index block should be small
  - If the index block is too small – it will not be able to hold enough pointers for a larger file
- Mechanisms,
  - Linked scheme – index block is normally one storage block. It can read/written directly by itself.
    - To allow larger files we can link several index blocks together. Index block might contain a small header giving the name of the file and a set of first 100 disk-block addresses. Next address is null or is a pointer to another index block (for a larger file)

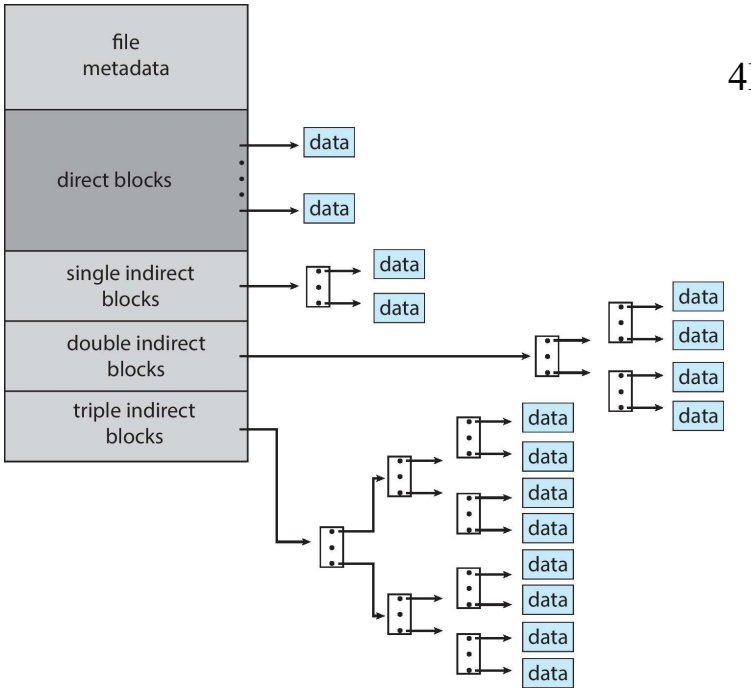
Indexed Allocation (Cont.) – performance

- Multilevel index – variant of linked representation
  - Uses first-level index block points to a set of second level index block, which in turn points to file blocks.
  - To access a block, OS uses first level index to find second level and then uses that block to find the desired data block. This level scheme can continue to a third or fourth level depending on the max. size of the block.
    - Eg: 4096 byte blocks, we could store 1024 - 4byte pointers, in an index block. Two levels of indexes allow 1048576 data blocks and a file size of up to 4GB



Indexed Allocation (Cont.) – performance

- Combined scheme – used in UNIX based file systems



4K bytes per block, 32-bit addresses

More index blocks than can be addressed with 32-bit file pointer

Question

- Assume you are given a system that consist of 32 bit addressing. Each data block is 1KB and pointer size is 4 byte in size. What is the maximum file size?
- Assume that there are 10 direct pointers to data blocks, 1 indirect pointer, 1 double indirect pointer, and 1 triple indirect pointer

Solution

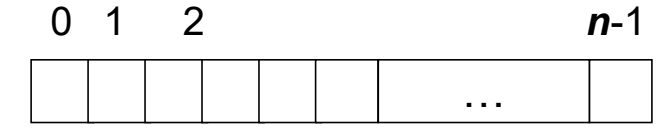
- #pointers in a block = 256
- #direct blocks (size) = 10x1KB = 10KB
- #single indirect blocks (size) = 256x1KBx1 = 256KB
- #double indirect blocks (size) = 256x256x1KBx1 = 64MB
- #triple indirect blocks (size) = 256x256x256x1KBx1= 16GB
- Total size = 16GB + 64MB+256KB+10KB = around 16GB

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

Free-Space Management

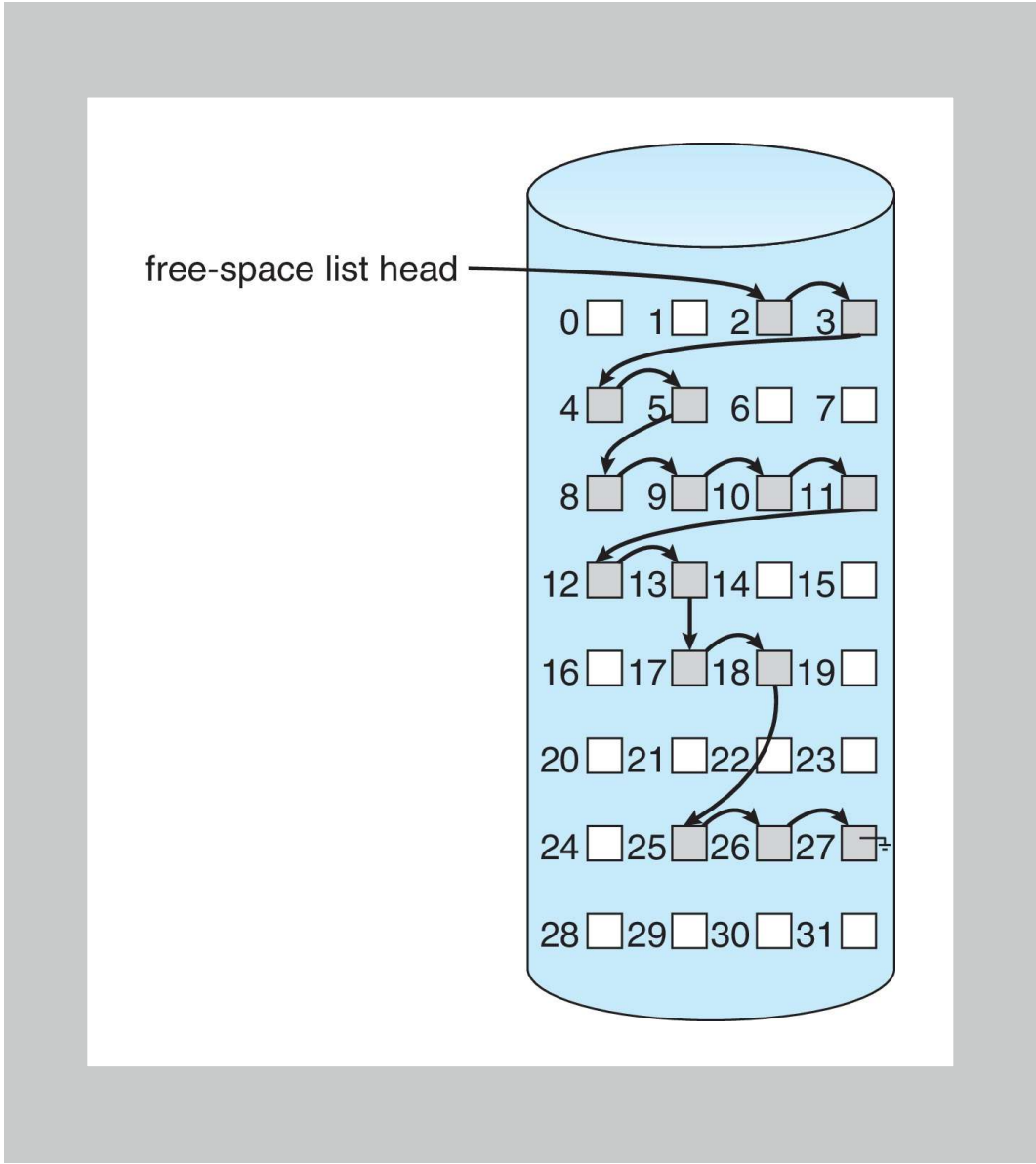
- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (***n*** blocks)



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

Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Thank you!