JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

Lecture-32 By Vivek Dubey

COMPUTER SCIENCE AND ENGINEERING

CS403PC

COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

CS403PC

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Scheme of Operating System

S. No.	Course Code	Course Title	L	T	P	Credits
3	CS403PC	Operating Systems	3	0	0	3
6	CS406PC	Operating Systems Lab	0	0	3	1.5

Lecture-32 By Vivek Dubey

UNIT-5: File System Interface and Operations

- Access methods,
- Directory Structure,
- Protection,
- File System Structure,
  - Allocation methods,
  - Free-space Management.
- Usage of open, create, read, write, close, lseek, stat, ioctl system calls.

Lecture-32 By Vivek Dubey

File System Structure

File Allocation Methods

- **The allocation methods** define *how the files are stored in the disk blocks.*
- **There are** *three main disk space or file allocation methods*:
  A. Contiguous Allocation
  B. Linked Allocation
  C. Indexed Allocation
- **The main idea** behind these methods *is to provide*:
  - Efficient disk space utilization.
  - Fast access to the file blocks.

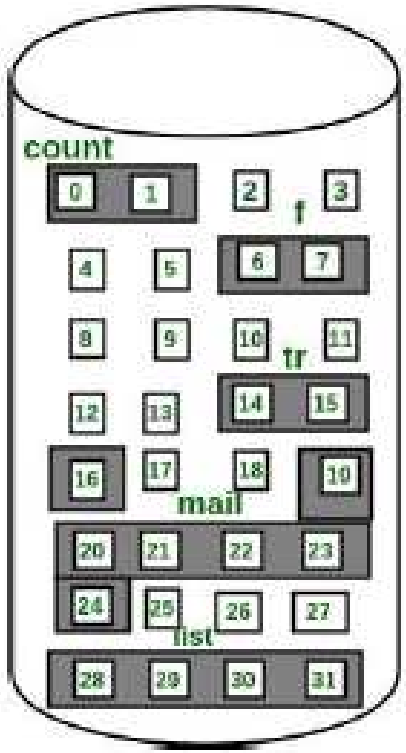
Lecture-32 By Vivek Dubey

A. Contiguous Allocation

Lecture-32 By Vivek Dubey

- **In this scheme**, *each file occupies* a contiguous set of blocks *on the disk.*
  - For example, **if a file requires n blocks** and *is given a block b as the starting location*, **then the blocks assigned to the file will be:** *b, b+1, b+2,……b+n-1.*
  - *This means that* **given the starting block address** and **the length of the file** (in terms of blocks required), *we can determine the blocks occupied by the file.*
  - **The directory entry for a file** *with contiguous allocation contains.*
    - Address of starting block
    - Length of the allocated portion.

The *file* *'mail'* in the following figure starts from the block 19 *with length = 6 blocks.* Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Directory

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

Lecture-32 By Vivek Dubey

Advantages of Contiguous Allocation

- **Both the Sequential and Direct Accesses** *are supported by this.*
- **For direct access,** the address of the kth block *of the file* which starts at block b *can easily be obtained as (b+k).*
- **This is extremely fast** since the number of seeks *are minimal* because of contiguous allocation of file blocks.

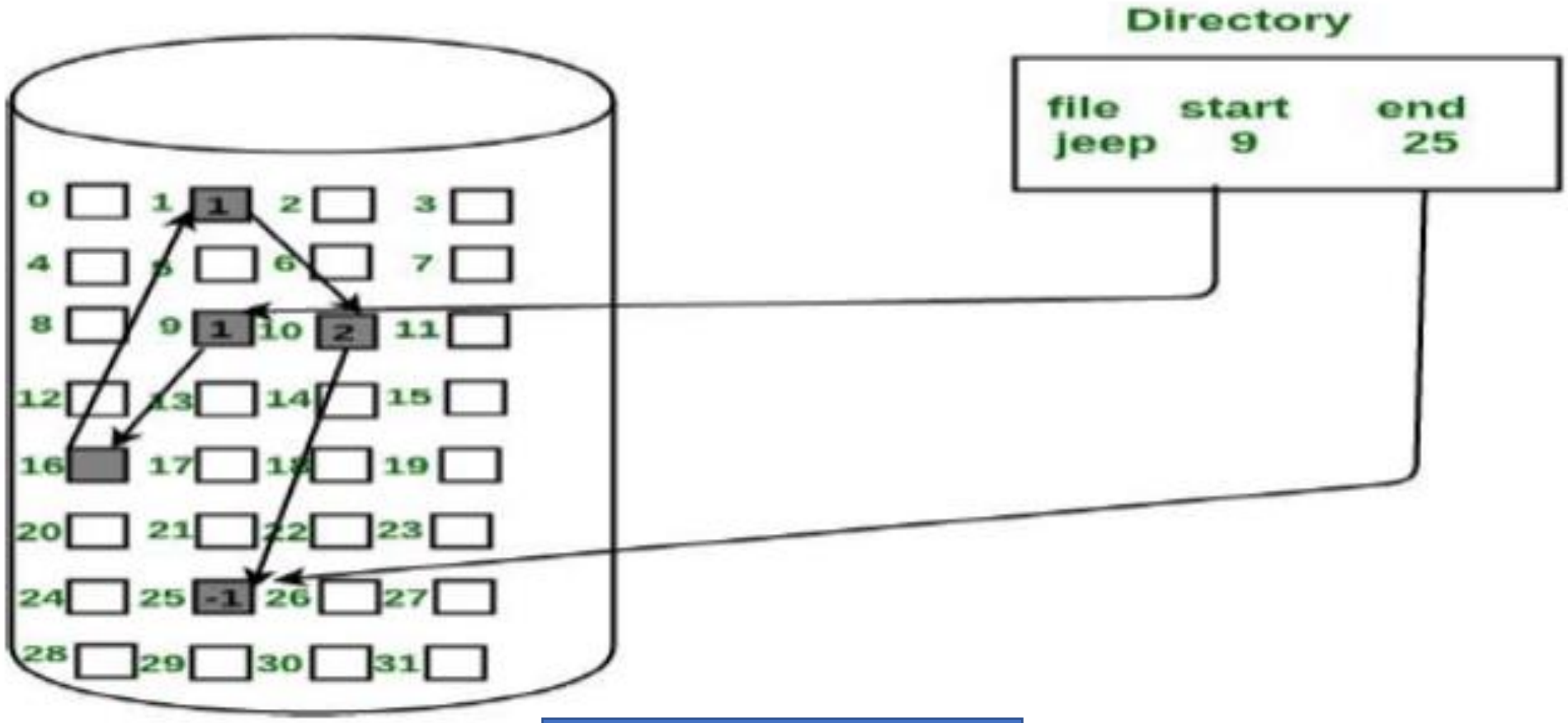
Disadvantages of Contiguous Allocation

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

B. Linked List Allocation

- In this scheme, **each file is a linked list** of disk blocks which **need not be** contiguous.
- **The disk blocks** *can be scattered* anywhere on the disk.
- **The directory entry contains** *a pointer* to the starting and the ending file block.
- **Each block contains** *a pointer* to the next block occupied by the file.

The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages of Linked List Allocation

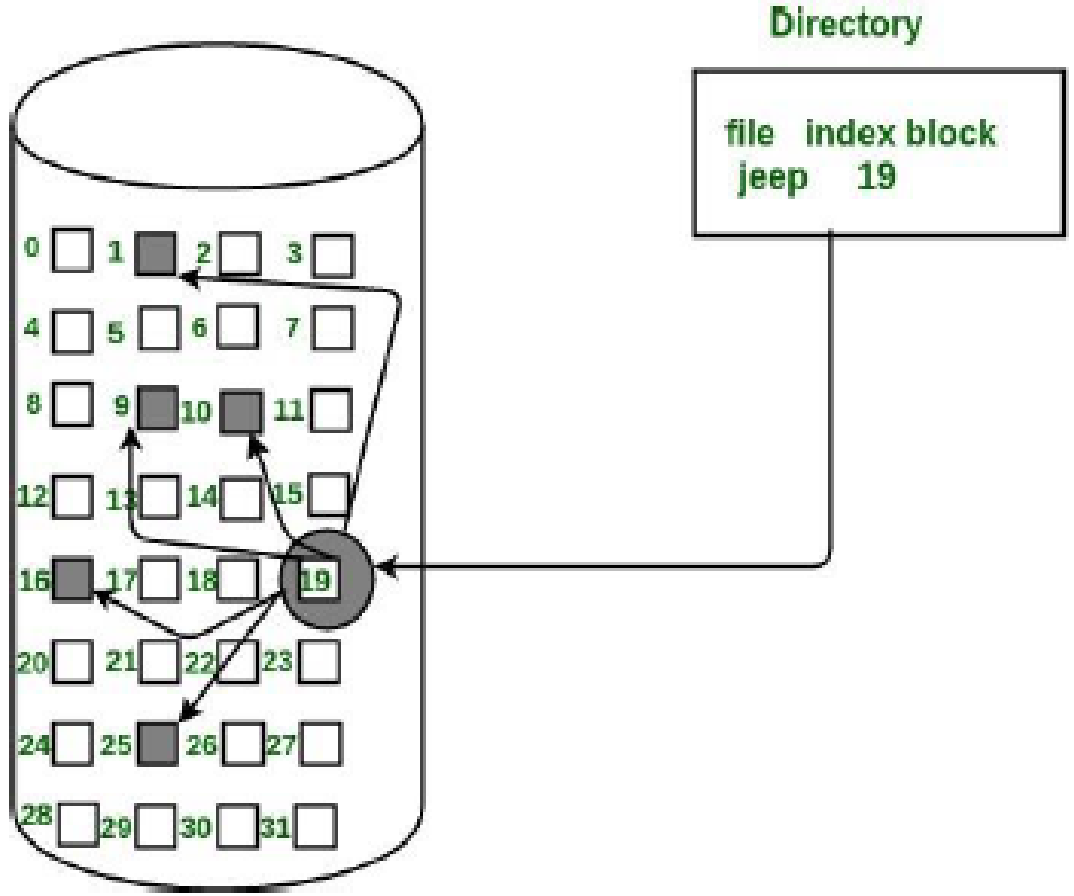
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages of Linked List Allocation

- **Because the file blocks are distributed** *randomly on the disk*, a large number of seeks **are needed to access** every block individually. This makes linked allocation slower.
- **It does not support** *random or direct access*. We can not directly access the blocks of a file. **A block k of a file** can be accessed *by traversing k blocks sequentially* (sequential access) from the starting block of the file via block pointers.
- **Pointers required** *in the linked allocation* suffer *some extra overhead.*

C. Indexed Allocation

- *In this scheme,* **a special block** known as the **Index block** *contains the pointers* to all the blocks occupied by a file.
- **Each file has** *its own index block*. **The i^{th} entry in the index block** contains *the disk address of the i^{th} file block.*
- **The directory entry** contains *the address of the index block* as shown in the image:



Advantages of Indexed Allocation

- **Indexed Allocation** supports direct access *to the blocks occupied* by the file and **therefore provides** fast access *to the file blocks*.
- It overcomes **the problem** *of external fragmentation.*

Disadvantages of Indexed Allocation

- **The pointer overhead for indexed allocation** is greater than *linked allocation*.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Contd..

- For files that are very large, single index block may not be able to hold all the pointers.
- Following mechanisms can be used to resolve this:
  - **Linked scheme**
  - **Multilevel index**
  - **Combined Scheme**

Linked scheme

- **This scheme** links two or more index blocks *together for holding the pointers.*
- **Every index block** would then contain *a pointer or the address* to the next index block.

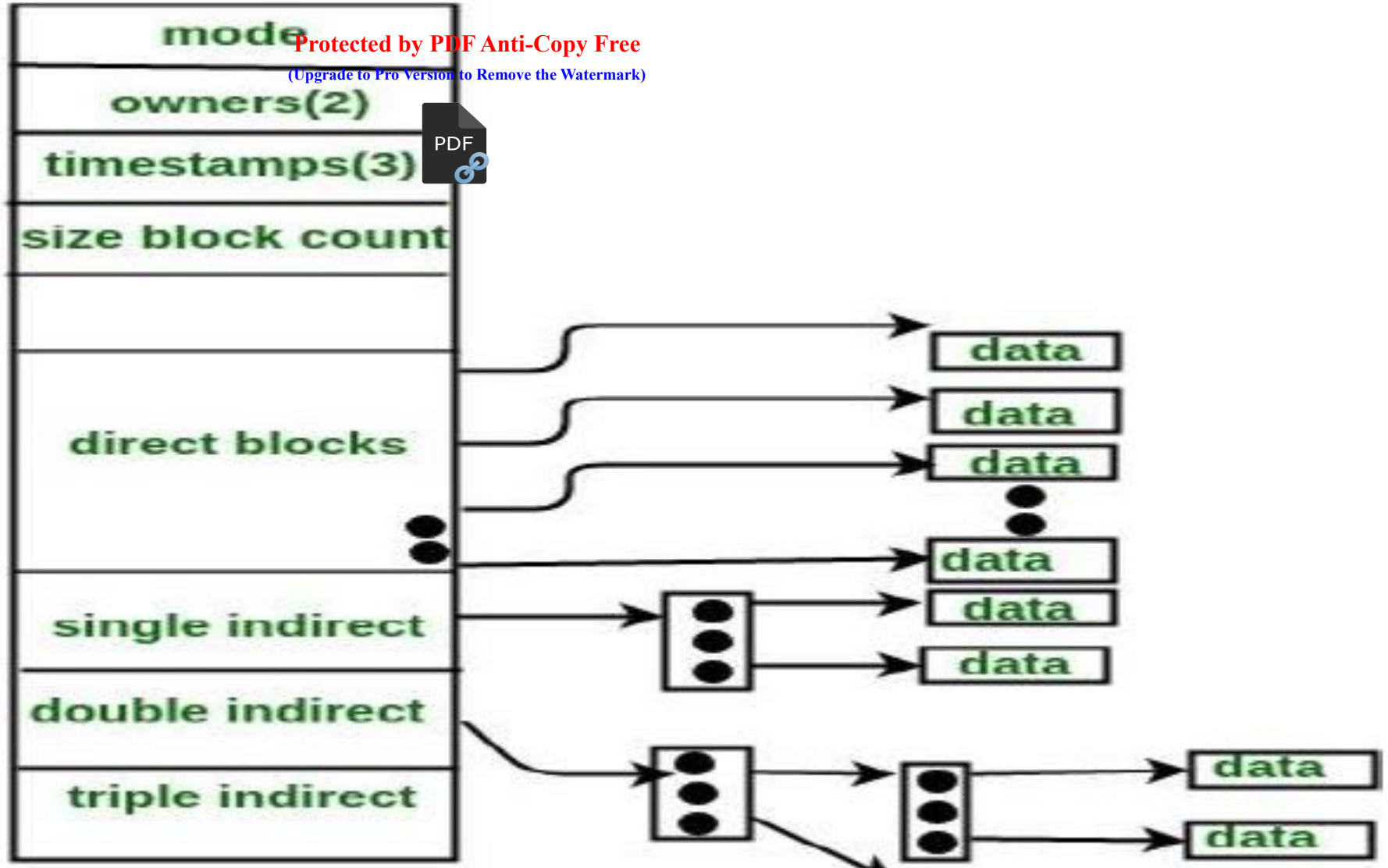
Multilevel Index

- In this policy, a first level index block is used
  - to point to the second level index blocks
  - which in turn points to the disk blocks occupied by the file.
- This can be extended
  - to 3 or more levels depending
  - on the maximum file size.

Combined Scheme

- *In this scheme*, **a special block** called the **Inode (information Node)** *contains all the information about the file*
  - such as the name, size, authority, etc and
  - the remaining space of Inode is used to store the Disk Block addresses
  - which contain the actual file *as shown in the image below.*
- **The first few of these pointers** *in Inode point* to the **direct blocks** i.e **the pointers contain** *the addresses of the disk blocks* **that contain** *data of the file*.
- **The next few pointers point** *to indirect blocks*. Indirect blocks may be
  A. Single Indirect, B. Double Indirect or C. Triple Indirect.
- **Single Indirect block** *is the disk block that* **does not contain the file data** *but the disk address of the blocks that contain the file data*.
- **Double Indirect blocks** *do not contain the file data but* **the disk address of the blocks that contain** *the address of the blocks containing the file data*.

Contd..



Free space management in Operating System

- **The system keeps** *tracks of the free disk blocks* for allocating space *to files when they are created*.
- **Also, to reuse the space** *released from deleting the files*, *however* free space management becomes crucial.
- **The system maintains** *a free space list* **which keeps track of the disk blocks that** are not allocated to some file or directory.
- **The free space list** *can be implemented mainly as*:
  - ✓Bitmap or Bit Vector
  - ✓Linked List
  - ✓Grouping
  - ✓Counting

Lecture-32 By Vivek Dubey

Bitmap or Bit Vector

- A Bitmap or Bit Vector is *series or collection of bits where* each bit corresponds to a disk block.
- The bit can take *two values: 0 and 1*: ***0 indicates*** *that the block is allocated* and **1 indicates** a free block.
- The given instance of disk blocks *on the disk in Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits **as: 0000111000000110**.

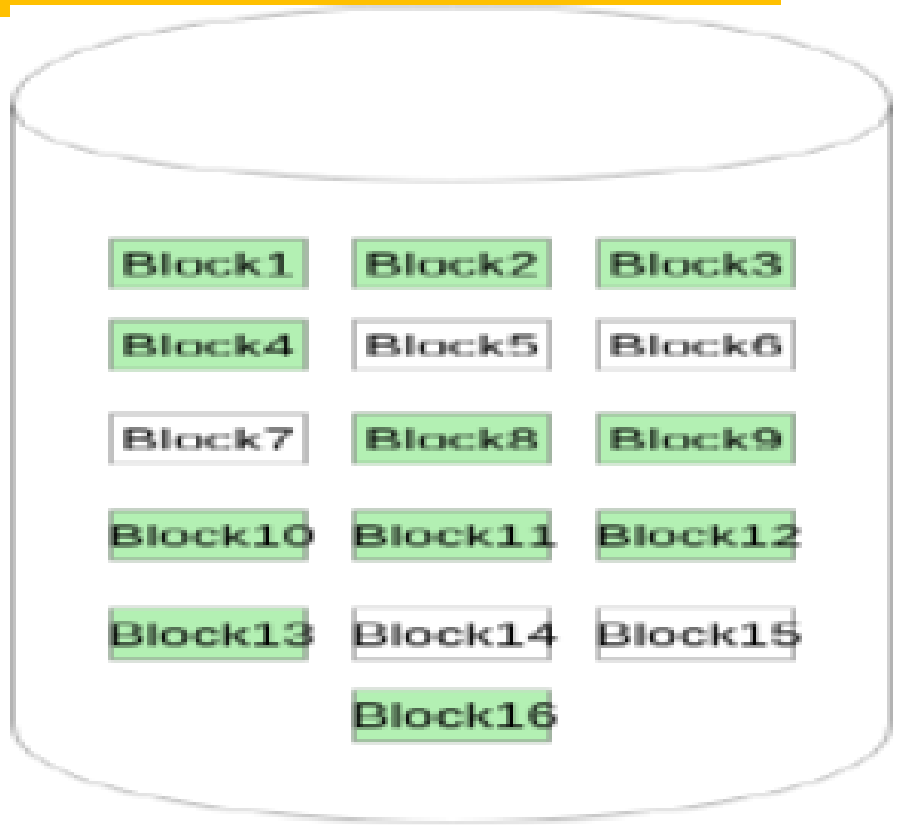


Figure - 1

Lecture-32 By Vivek Dubey

Advantages of Bitmap or Bit Vector

- **Simple** *to understand*.
- **Finding the first free block is** *efficient*.
  - It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0).
  - **The first free block is** *then found by scanning* for the first 1 bit *in the non-zero word*.

Lecture-32 By Vivek Dubey

Contd..

- **The block number can be** *calculated as*:
  - *(number of bits per word) *(number of 0-values words) + offset of bit first bit 1 in the non-zero word .*

- *For the Figure-1,* **we scan the bitmap sequentially** *for the first non-zero word.*
  - The first group of 8 bits (00001110) constitute a non-zero word since all bits are not 0. After the non-0 word is found, we look for the first 1 bit. This is the 5th bit of the non-zero word. So, offset = 5. Therefore, the first free block number = 8*0+5 = 5.

Linked List

- *In this approach,* **the free disk blocks** *are linked together i.e. a free block* contains a pointer to the next free block. *The block number* **of the very first disk block** *is stored at a separate location* on disk and is also cached in memory.
- *In Figure-2,* **the free space list head** points to Block 5 which points to Block 6, the next free block and so on. **The last free block would contain** *a null pointer indicating* the end of free list. **A drawback of this method** *is the I/O required for free space list traversal.*

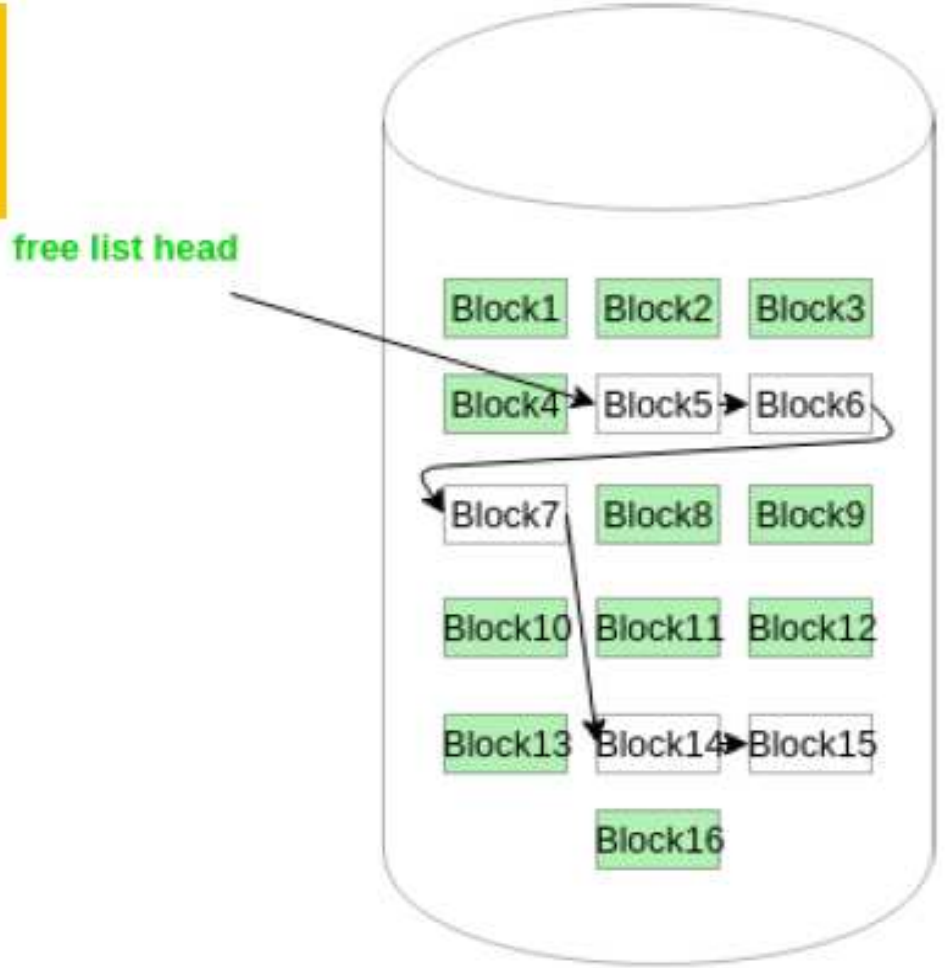


Figure - 2

Lecture-32 By Vivek Dubey

Grouping

- **This approach stores** *the address of the free blocks* in the first free block.
- **The first free block stores** *the address of some*, say n free blocks.
- *Out of these n blocks*, **the first n-1 blocks are** actually free and **the last block contains** the address of next free n blocks.
- An **advantage** *of this approach is that* the addresses of a group *of free disk blocks can be found easily.*

Counting

- **This approach stores** *the address of the* first free disk block and **a number n of** *free contiguous disk blocks* that follow the first block.
- **Every entry in the list would contain**:
  - *Address of first free disk block*
- **A number n**
  - For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

Lecture-32 By Vivek Dubey