File System Implementation

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

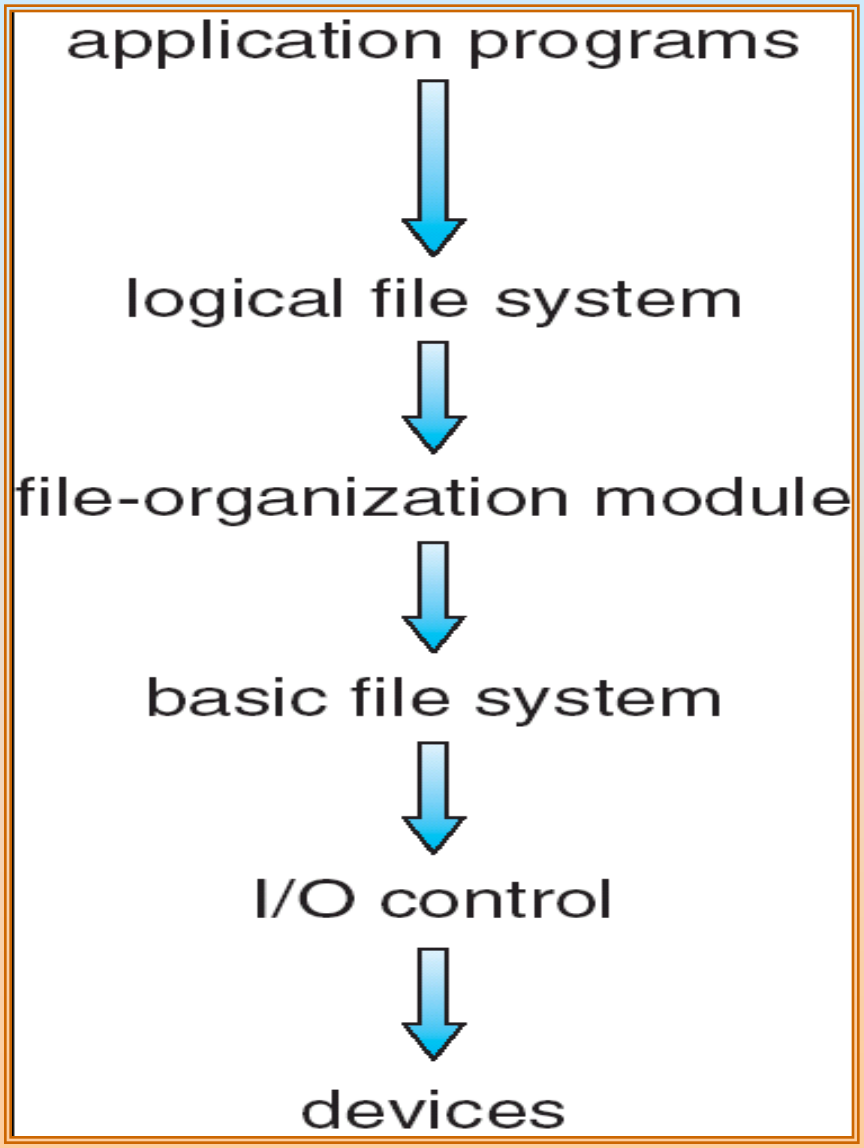
File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- I/O transfers between memory and disks are performed in units of *blocks*
- File control block (FCB) – storage structure consisting of information about a file
- Device driver - controls the physical device
- Disks have 2 important characteristics:
  - They can be rewritten in place
  - We can access directly any given block of information on the disks

File system organization

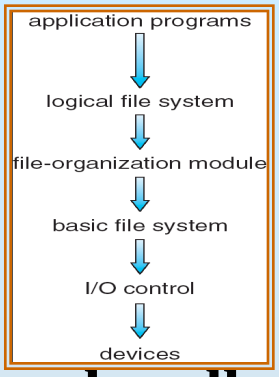
- To provide an efficient and convenient access to the disks, the operating system imposes a file system to allow the data to be stored, located, and retrieved easily.
- File system poses 2 different design problems :
  - Defining how the file system should look to the user
    - This involves defining of a file and its attributes, operations, the directory structure etc
  - Algorithms and data structures must be created to map the logical file system onto the physical secondary- storage devices.
- File system is itself composed of many different levels
  - Each level in the design uses the features of lower levels to create new features for use by higher levels

File system organization



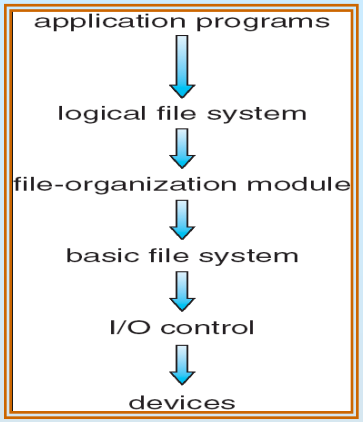
Layered file system

File system organization



- **I/O control**
  - **lowest level**
  - **consists of device drivers and interrupt handlers to transfer information between the memory and the disk system**
- **Basic file system**
  - **issues generic commands to the appropriate device driver to read and write physical blocks on the disks**
  - **Also manages memory buffers and caches (allocation, freeing, replacement)**
    - **Buffers hold data in transit**
    - **Caches hold frequently used data**

File system organization



- **File organization module**
  - **knows about files and their logical blocks, as well as physical blocks, which helps them to translate logical block addresses to physical block addresses for basic file system to transfer**
- **Logical file system**
  - **Uses the directory structure to provide the file organization module with the information that it needs, given the symbolic file name**
  - **Directory management**
  - **Also responsible for protection and security**

File Systems are many

- Many file systems, sometimes many within an operating system
  - Each with its own format
    - (CD-ROM is ISO 9660;
    - Unix has **UFS**, FFS;
    - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray
    - Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

File system mounting

- A file must be opened before it is used, a file system must be *mounted* before it can be available to processes on the system
- Procedure for mounting:
  - Operating system is given the name of the device, and the location within the file structure at which to attach the file system (called mount point)
    - Example: on UNIX -- /home
      - /home/ jaikishan
  - Operating system verifies that device contains a valid file system
    - It does so by asking the device driver to read the device directory and verifying that the directory has the expected format
  - Operating system notes in its directory structure that a file system is mounted at the specified mount point

Allocation Methods

- **An allocation method refers to how disk blocks are allocated for files:**

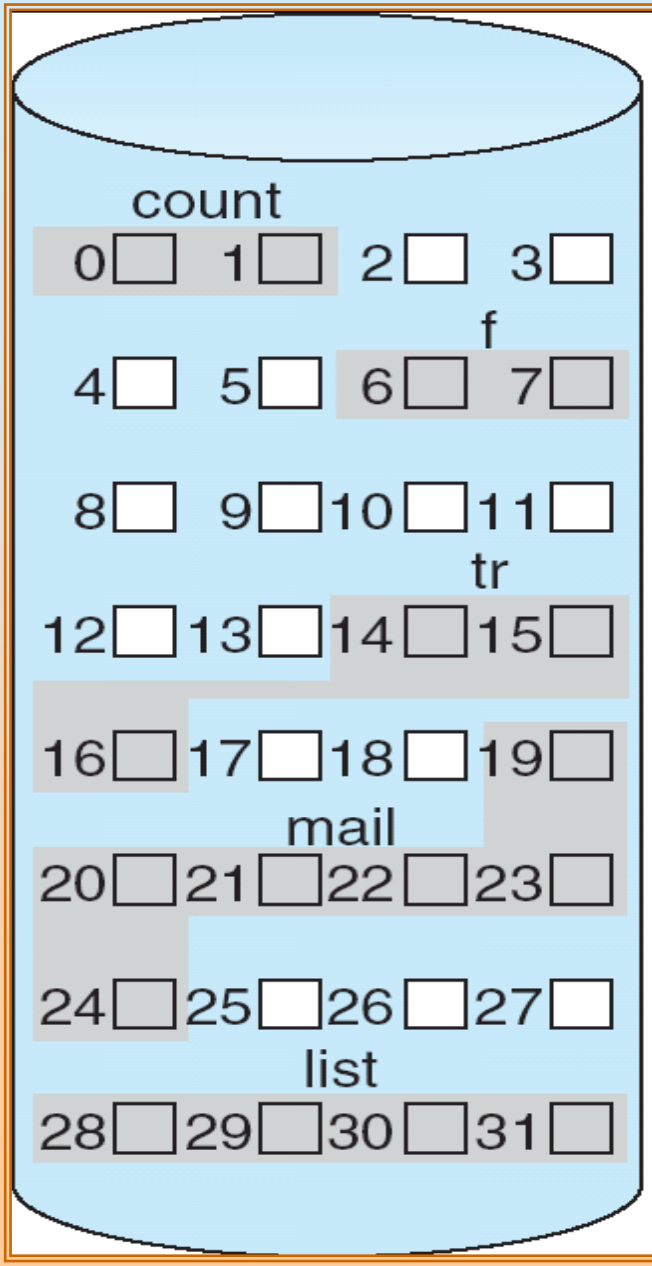
- **Types of allocation**

1. **Contiguous allocation**
2. **Linked allocation**
3. **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
- Accessing block b + 1 after block b normally requires no head movement
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file
- Accessing a file that has been allocated contiguously is easy:
  - for sequential access, the file system remembers the disk address of the last block referenced and whenever necessary reads the next block
  - for direct access to block i of a file that starts at block b, we can immediately access block b + i

Contiguous Allocation of Disk Space



directory

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

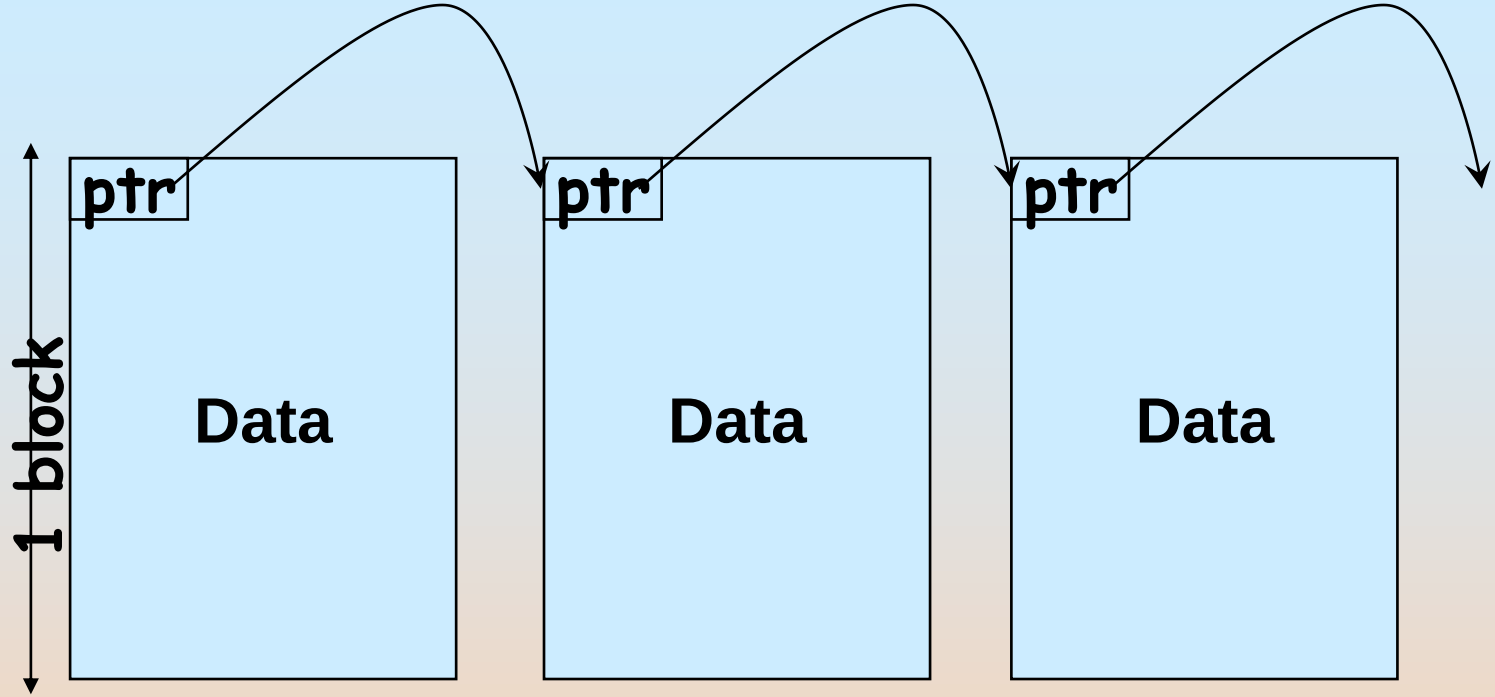
Contiguous allocation - Disadvantages:

- **Finding space for a new file**
  - **It can be seen as general dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes**
    - **First fit and best fit are the most common strategies**
- **External fragmentation**
  - **Exists whenever free space is broken into chunks**
- **Some older systems used contiguous allocation on floppy disks.**
  - **This effectively compacts all the free space into one contiguous space, solving fragmentation problem**
  - **Cost of compaction is time --- it may take hours and may be weeks**
  - **Normal system operation cannot continue during this down time, so compaction is avoided**
- **Major problem is determining how much space is needed for a file**

Contiguous allocation

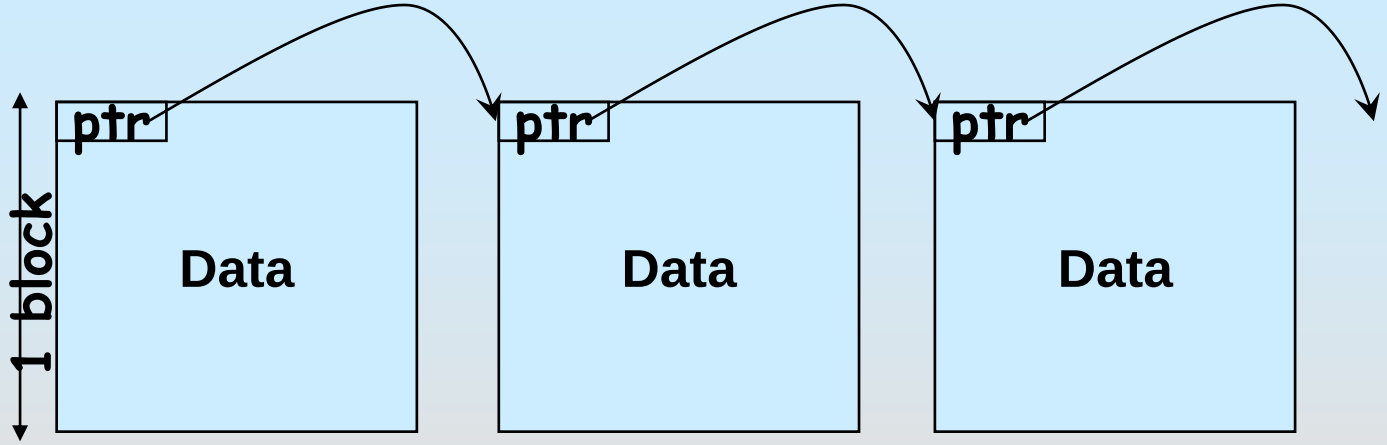
- To avoid all the drawbacks some newer file systems (i.e. VERITAS File System) use a modified contiguous allocation scheme
- Contiguous chunk of space is allocated initially, and then, when that amount is not large enough, another chunk of contiguous space, an extent, is added to the initial chunk of contiguous space
- The location of a files blocks is then recorded as a location and a block count, plus a link to the first block of next extent

Linked Allocation



- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Directory contains a pointer to the first and last blocks of the file.
- Pointers are not made available to the user

Linked Allocation

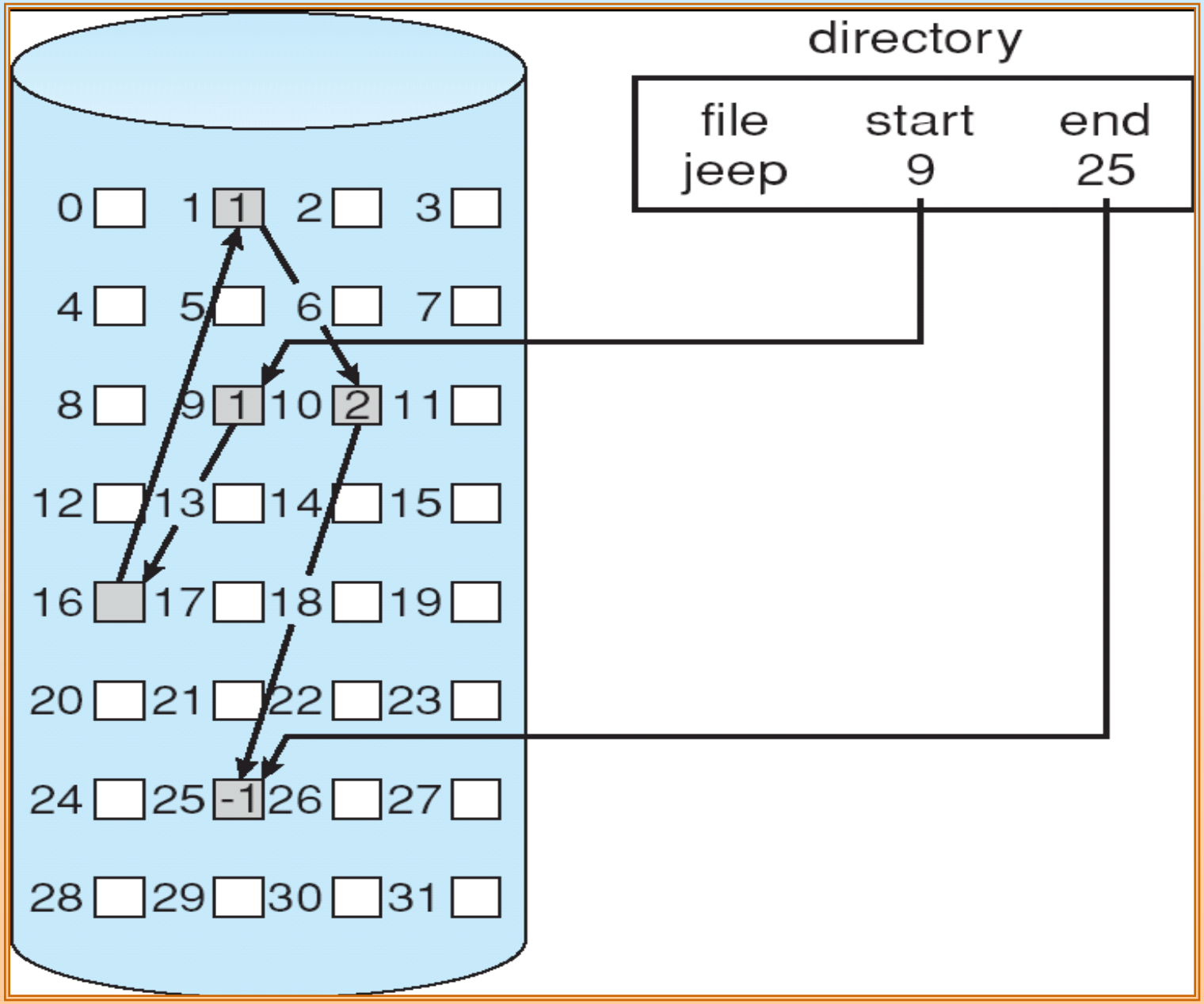


- To create a new file we simply create a new entry in the directory
  - With linked allocation each directory has a pointer to the first disk block of the file ---initialized to 0, the size field is also set to 0
  - Write to the file causes a free block to be found, and this block is written to, and linked to the end of the file
  - To read a file, we simply read blocks by following the pointers from block to block

Linked Allocation

- **Advantages:**
  - **No external fragmentation. Any free block on the free-space list can be used to satisfy the request**
  - **No need to declare the size of a file when it is created**
  - **A file can continue to grow as long as there are free blocks**
  - **There is no necessary to compact disk space**
- **Disadvantages:**
  - **Can be used effectively only for sequential-access files. To find i^{th} block of a file, start from the beginning and follow the pointers until i^{th} block**
  - **Space required for the pointers**
    - **User solution to this problem is to collect blocks into multiples, called clusters**
  - **reliability**

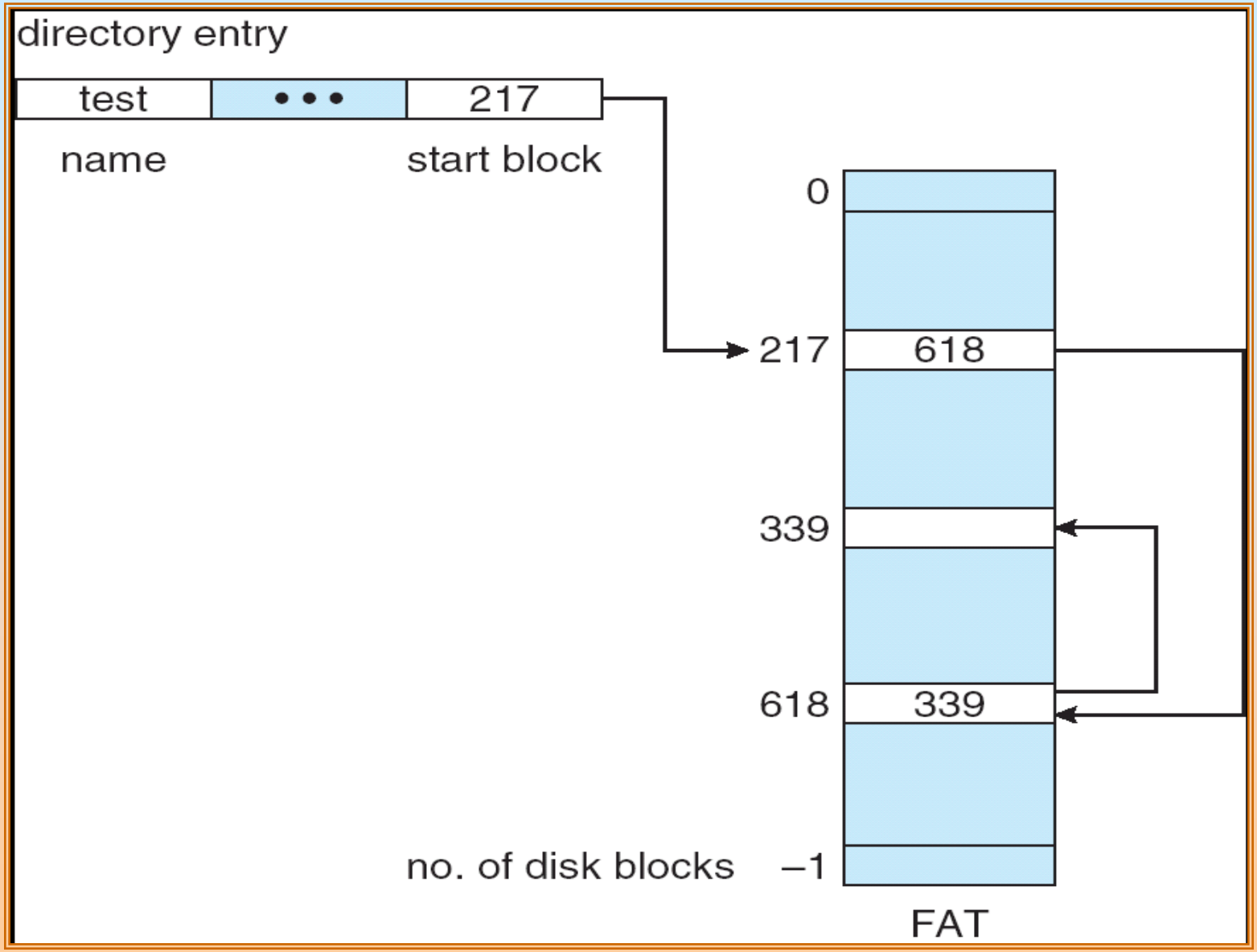
Linked Allocation of disk space



File-Allocation Table(FAT)

- An important variation on linked allocation is FAT
- Simple and efficient method of disk-space allocation used by the MS-DOS and OS/2 OS
- A section of disk at the beginning of each partition is set aside to contain the FAT
- 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
- FAT entry is indexed by that block number that contains the block number of the next block in the file
- Chain continues until last block
- Last block has a special entry
- Unused blocks are indicated by a 0 table value

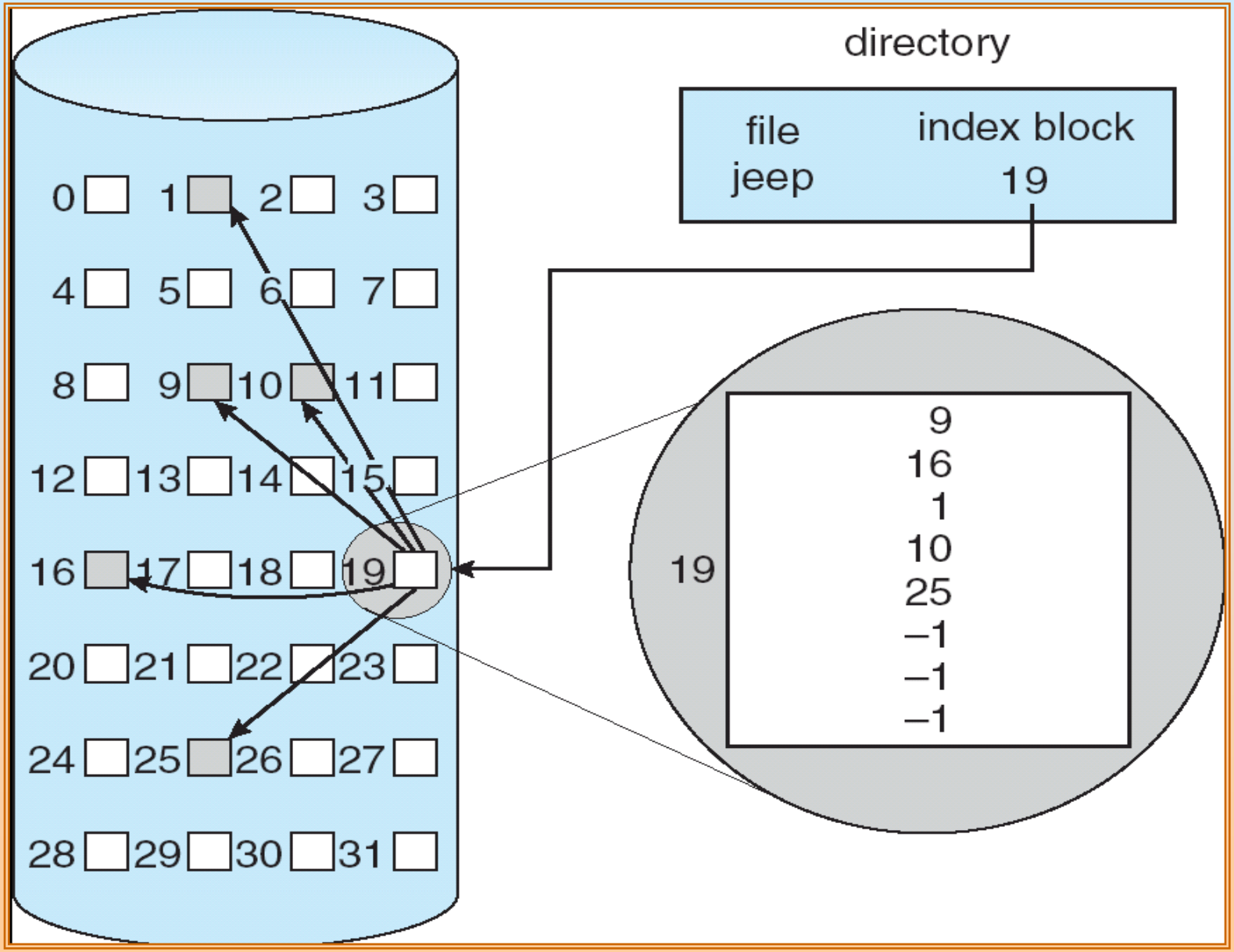
File-Allocation Table



Indexed Allocation

- Linked allocation solves external fragmentation and size-declaration problems of contiguous allocation
- It cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order
- Indexed allocation solves this problem by bringing all pointers together into one location : the **index block**
- Each file has its own index block, which is an array of disk-block addresses
- The i^{th} entry in the index block points to i^{th} block of the file
- Directory contains the address of the index block
- When file is created, all pointers in the index block are set to nil
- When i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} index block entry

Example of Indexed Allocation



Indexed Allocation

- **Advantages:**
  - **Supports direct access**
  - **No external fragmentation**
  - **Any free block on the disk can satisfy a request for more space**
- **Disadvantages:**
  - **Suffers from wasted space**
  - **The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation**
  - **An entire index block must be allocated, even if only one or two pointers will be non-nil**

Indexed Allocation

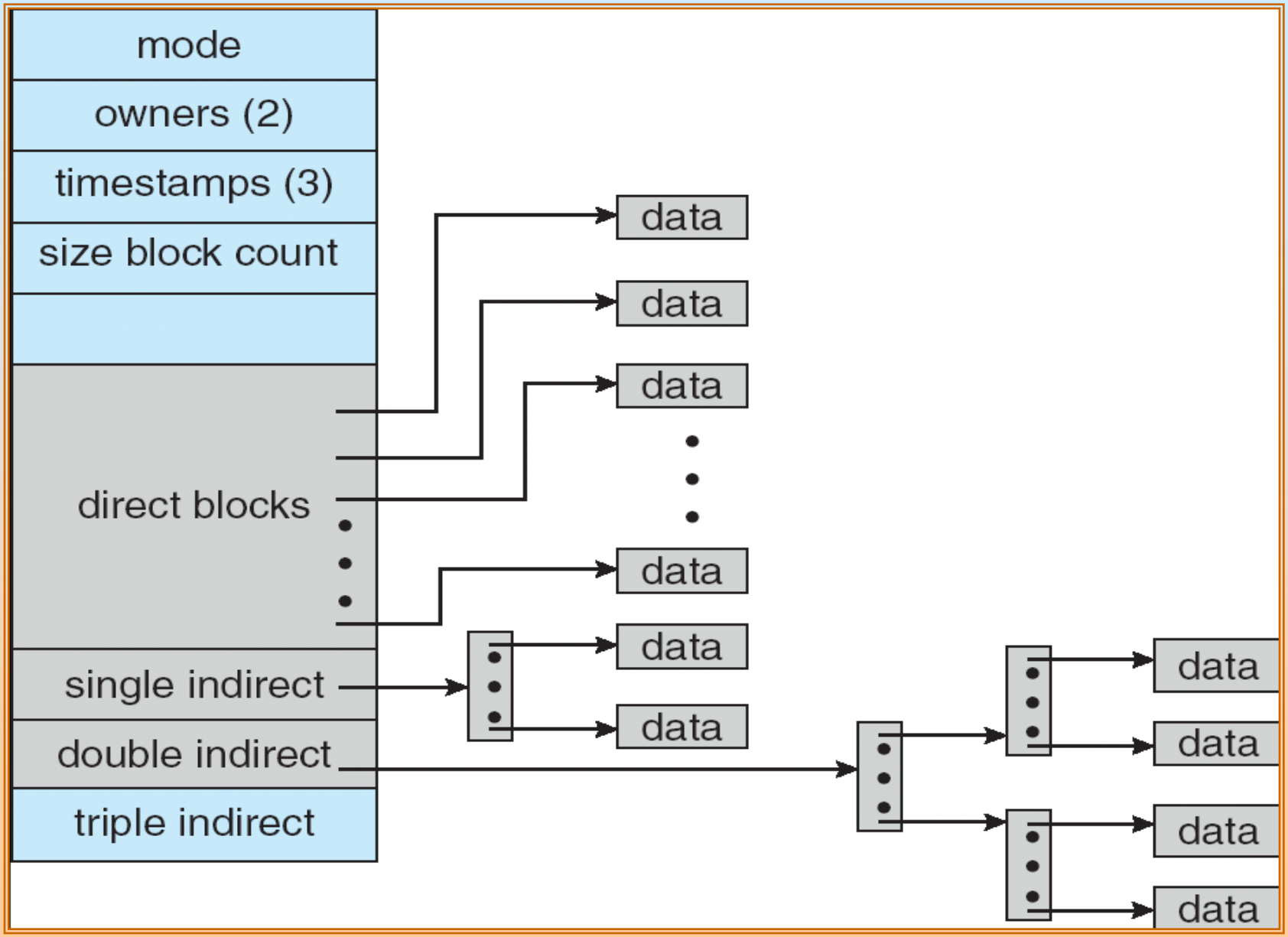
- How large the index block should be?
  - Small files —> as small as possible
  - Large files —> as large as needed

- Mechanisms to deal with size of index
  - Linked Scheme
  - Multilevel index
  - Combined Scheme

Indexed Allocation

- **Linked scheme:**
  - **An index block is normally one disk block**
  - **Can be read and written directly by itself**
  - **For large files, we may link together several index blocks**
- **Multilevel index:**
  - **Use separate index block to point to index blocks, which point to the file bocks themselves**
  - **To access a block, the OS uses the first level index to find second level index to find the desired data block.**
- **Combined scheme:**
  - **Keep the first n pointers of the index block in the files index block (or inode)**

Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

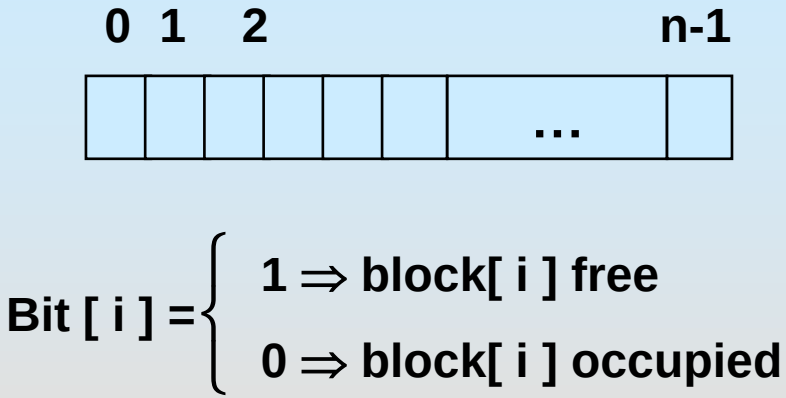
- To keep track of free disk space, the system maintains a free-space list
  - Free-space list records all disks blocks that are free – those are not allocated to some file or directory
- To create a file :
  - Search the free space list for the required amount of space, and allocate the space to the new file
  - This space is then removed from the free-space list
- Implementation of Free Space List

1. Bit vector 2. Linked List 3.Grouping 4. Counting

1. Bit vector

  - Free space list is implemented as a bit map or bit vector
  - Each block is represented by 1 bit.
  - If block is free, bit is 1; if allocated, bit is 0

Free-Space Management



- Main advantage is that it is relatively simple and efficient to find the first free block, or n consecutive free blocks on the disk
- Sequentially check each word in the bit map to see whether that value is non zero. First non zero word scanned for first 1 bit, which is the location of the first free block

Free-Space Management

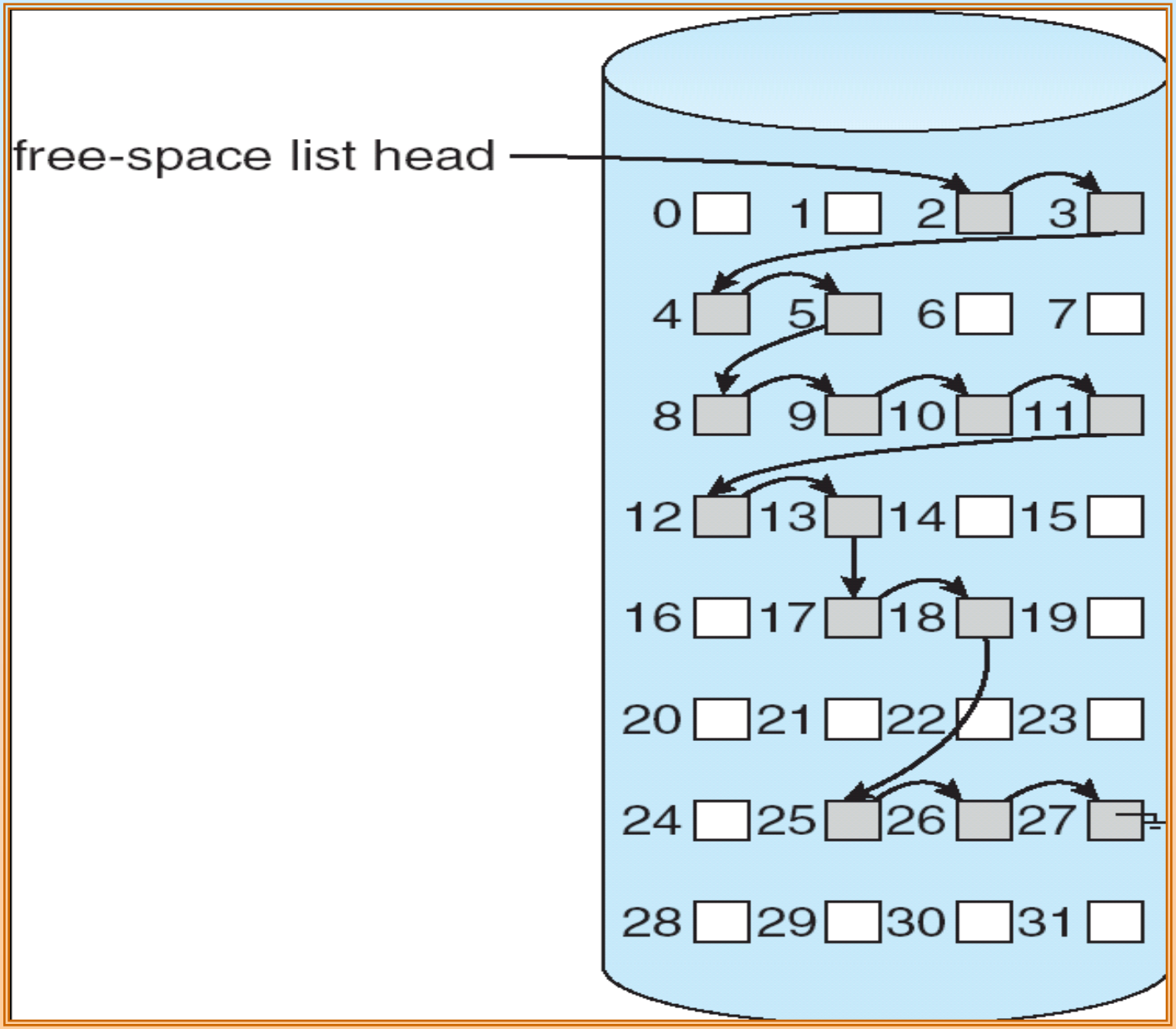
- Disadvantage is that – bit vectors are inefficient unless the entire vector is kept in main memory and written to disk occasionally for recovery needs
  - Keeping in main memory is possible for smaller disks, but not for larger ones
  - Bit map requires extra space
  - Example:
- block size = 1 KB = 1024 bytes
- disk size = 40GB = 41943040 KB = 41943040 blocks
- So, 41943040 bits are required to represent it in bitmap
- i.e., 41943040 / 8 = 5242880bytes =5120kb=5mb
- 5 MB extra space is required for keeping bit map

Free-Space Management

2. 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
- First block contains a pointer to the next free disk block, and soon
- Disadvantages :
  - Not efficient to traverse the list, we must read each block, which requires substantial I/O time

Linked Free Space List on Disk



Free-Space Management

3. Grouping:

- Store the addresses of ‘n’ free blocks in the first free block.
- The first ‘n – 1’ of these blocks are actually free
- Last block contains the address of another n free blocks, and so on
- Advantage:
  - Addresses of a large number of free blocks can be found quickly, unlike linked list approach

Free-Space Management (Cont.)

Generally several contiguous blocks are allocated or freed simultaneously.

4. Counting :

- **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 blocks that follow the first block**
- **Each entry in the free space list then consists of a disk address and a count**
- **Although each entry requires more space than a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1**

Efficiency and Performance

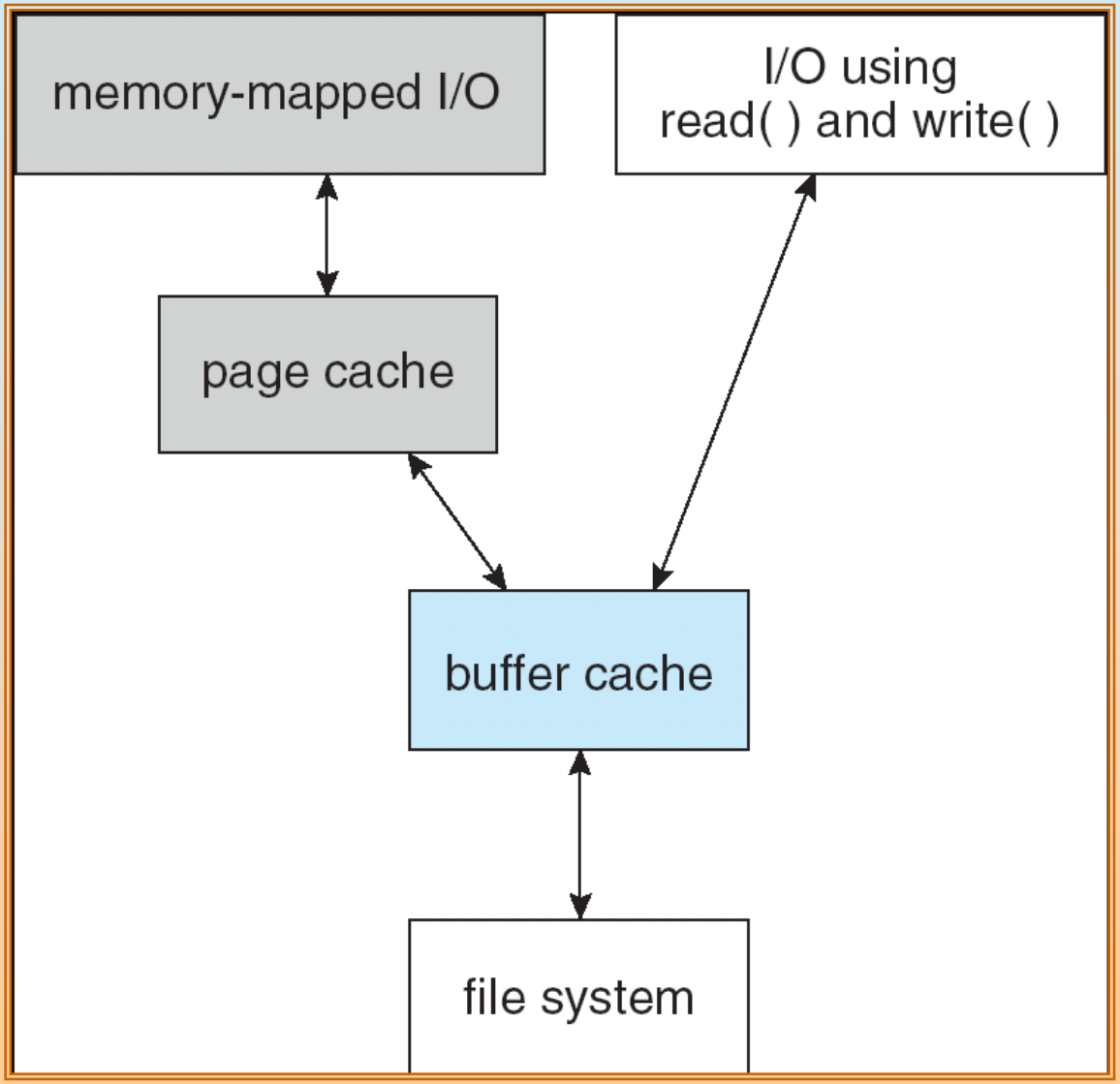
- **Efficiency dependent on:**
  - **disk allocation and directory algorithms**
  - **types of data kept in file's directory entry**

- **Performance**
  - **disk cache – separate section of main memory for frequently used blocks**
  - **free-behind and read-ahead – techniques to optimize sequential access**
  - **improve PC performance by dedicating section of memory as virtual disk, or RAM disk**

Page Cache

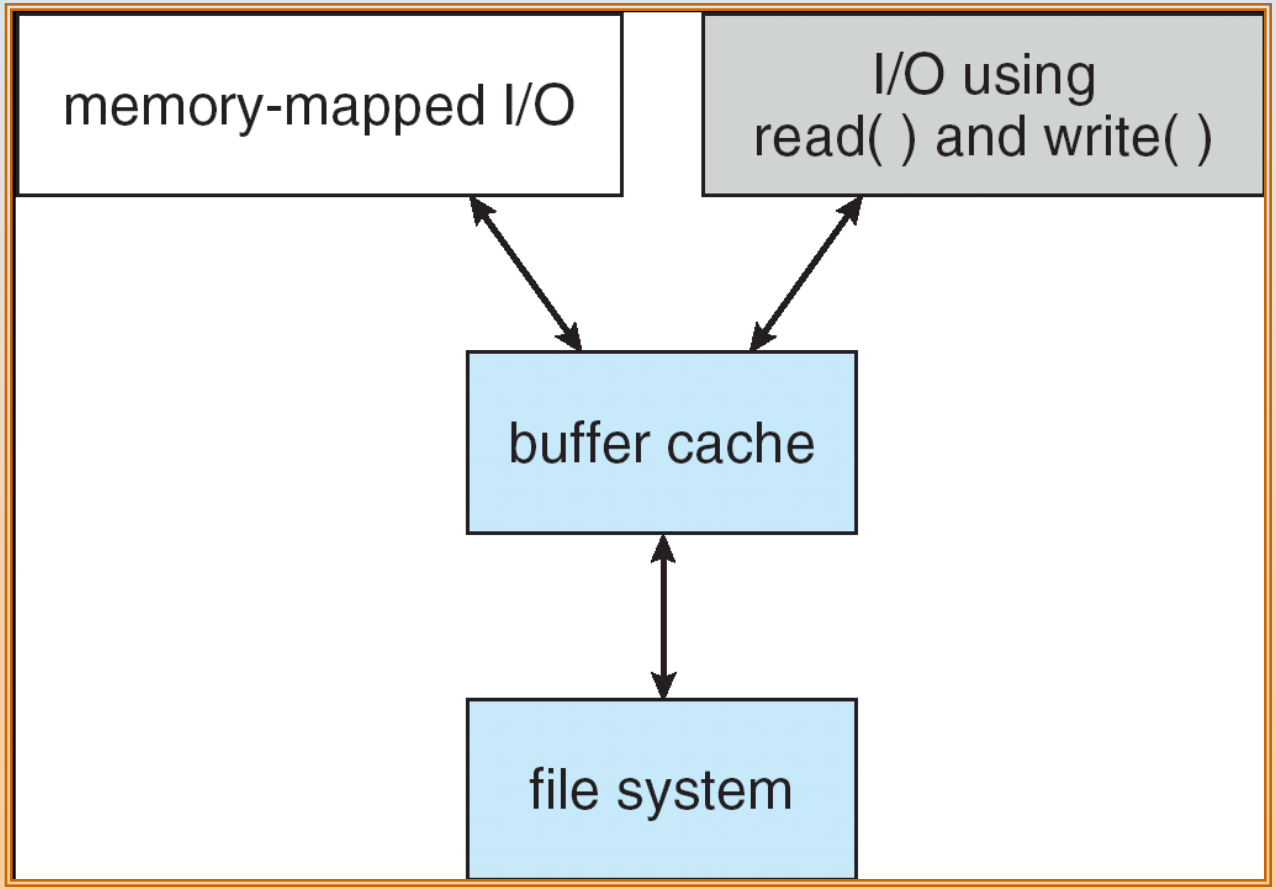
- A page cache caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache

I/O Without a Unified Buffer Cache



Unified Buffer Cache

- **A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O**



Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup

End of Chapter