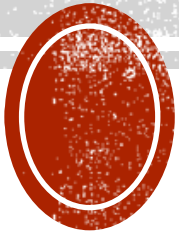
FILE SYSTEM

Chapter 5



SYLLABUS

- Files
  - Naming
  - Structure
  - Types
  - Access
  - Attributes
  - Operations
  - Memory mapped files
- Directories
  - Hierarchical directory system
  - Pathnames
  - Directory operations
- FileSystem Implementation
  - Implementing files
  - Implementing directories
  - Shared files
  - Disk space management
  - File system performance

FILES

- How to store the large amount of data into the computer?
- What happens, when process terminates or killed using some data?
- How to assign the same data to the multiple processes?

- *The solution to all these problems is to store information on disks or on other external media called **files**.*

FILE

- *A file is named **collection of related information** normally resides on a secondary storage device such as disk or tape.*
- Commonly, files represent **programs** (both source and object forms) and **data**;
  - data files may be numeric, alphanumeric, or binary.
- Information stored in files must be **persistent**,
  - - not be effected by power failures and system reboot.
- The files are managed by OS.
  - *The part of OS that is responsible to manage files is known as the **file system.***

FILES SYSTEM ISSUES

- How to *create* files?
  How they *named*?
  How they are *structured*?
  What *operation* are allowed on files?
  How to *protect* them?
  How are they *accessed* or *used*?
  How to *implement*?

FILE NAMING

- When a process creates a file, it gives the file name; while process terminates, the file continue to exist and can be accesses by other processes.
- *A file is named, for the convenience of its human users, and it is referred to by its name. A name is string of characters.*
- The string may be of digits or special characters (eg. 2, !,% etc).
  Some system differentiate between **the upper and lower case** character, whereas other system consider the equivalent (like Unix and MS-DOS)

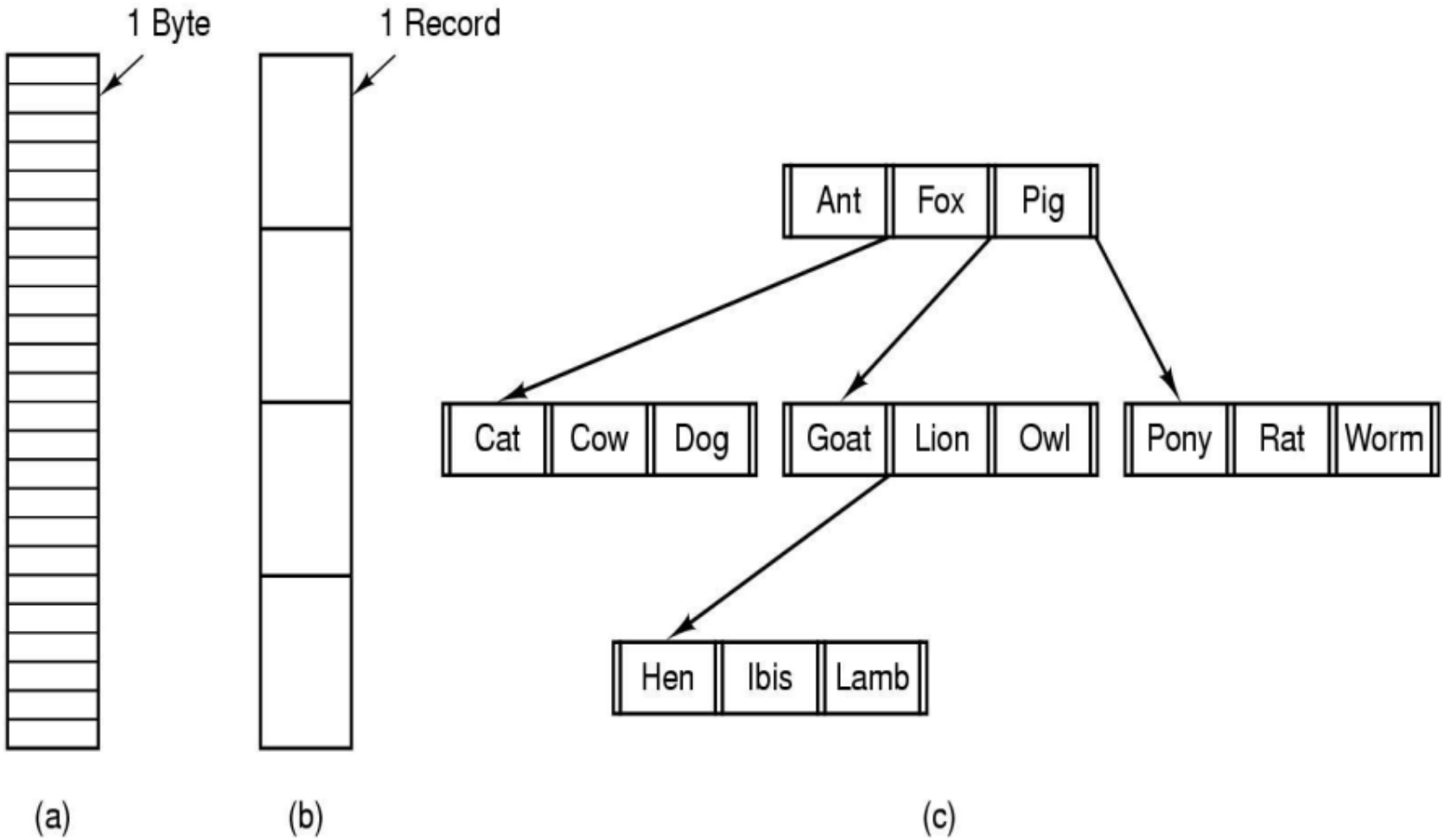
FILE NAMING

- Normally the string of **max 8 characters** are legal file name (e.g., in DOS),
- but many recent system support as long as **255 characters** (eg. Windows 2000).
- Many OSs support **two-part file names**;
  - separated by period; the part following the period is called the file extension
- **File Extension:**
  - usually indicates something about the file (e.g., file.c – C source file).
  - In some system it may have **two or more extension** such as in Unix proc.c.Z - C source file compressed using Ziv-Lampel algorithm.
  - In some system (e.g., Unix), file extension are **just conventions**;
  - In other system it requires (e.g., C compiler must requires .c source file).

FILE STRUCTURE

- Files must have structure that is understood by OS.
- Files can be structured in several ways.

- The most common structures are:
- **Unstructured**
- **Record Structured**
- **Tree Structured**



a) Unstructured b) Record structured c) Tree structured

UNSTRUCTURED

- Consist of **unstructured sequence of bytes or words.**
- OS does not know or care what is in the file.
- Any meaning must be imposed by user level programs.
- Provides maximum flexibility;
- user can put anything they want and name they, anyway that is convenient.
- Both Unix and Windows use these approach.

RECORD STRUCTURE

- A file is a sequence of fixed-length records, each with some internal structure.
- Each read operation returns one records, and write operation overwrites or append one record.
- Many old mainframe systems use this structure.

- As a historical note, in decades gone by, when the 80-column punched card was king, many (mainframe) operating systems based their file systems on files consisting of 80-character records.

TREE STRUCTURED

- File consists of tree of records, not necessarily all the same length.
- Each containing a key field in a fixed position in the record, sorted on the key to allow the rapid searching.
- The operation is to get the record with the specific key.
- Used in large mainframe for commercial data processing.

FILE TYPES

- UNIX and Windows, for example, have **regular files** and **directories**,
  - UNIX also has **character** and **block special files**
- ***Regular files*****:** contains user information, are generally *ASCII* or *binary*.
- ***Directories*****:** system files for maintaining the structure of file system.
- ***Character Special files*****:** related to I/O and used to model serial I/O devices such as terminals, printers, and networks.
- ***Block special files*****:** used to model disks.

FILE TYPES

- **ASCII files:**
  Consists of line of text
  Each line is **terminated either by carriage return** character or by line feed character or both.
  They can be displayed and printed as is and can be edited with ordinary text editor.
- **Binary files:**
  Consists of sequence of byte only.
  They have some internal structure known to programs that use them (e.g., executable or archive files).

- *Many OS use extension to identify the file types; but Unix like OS use a **magic number** to identify file types*.

FILE ACCESS METHOD

- Read Information from the file.
- **1. Sequential Access:**
  - a process could **read all the bytes or records in a file in order**,
  - could not skip around and read them out of order.

- **2. Direct Access:**
  - Files whose bytes or records can be **read in any order**
  - **Based on disk model** of file, since disks allow random access to any block.
  - Operations: *read n, write n* (n is block number) or *seek* – to set current position.
  - File can be access sequentially from the current position

FILE ATTRIBUTES

- In addition to name and data, all other information about file is termed as **file attributes.**
- The file attributes may very from system to system.

- Some common attributes are listed in the table:

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

FILE OPERATIONS

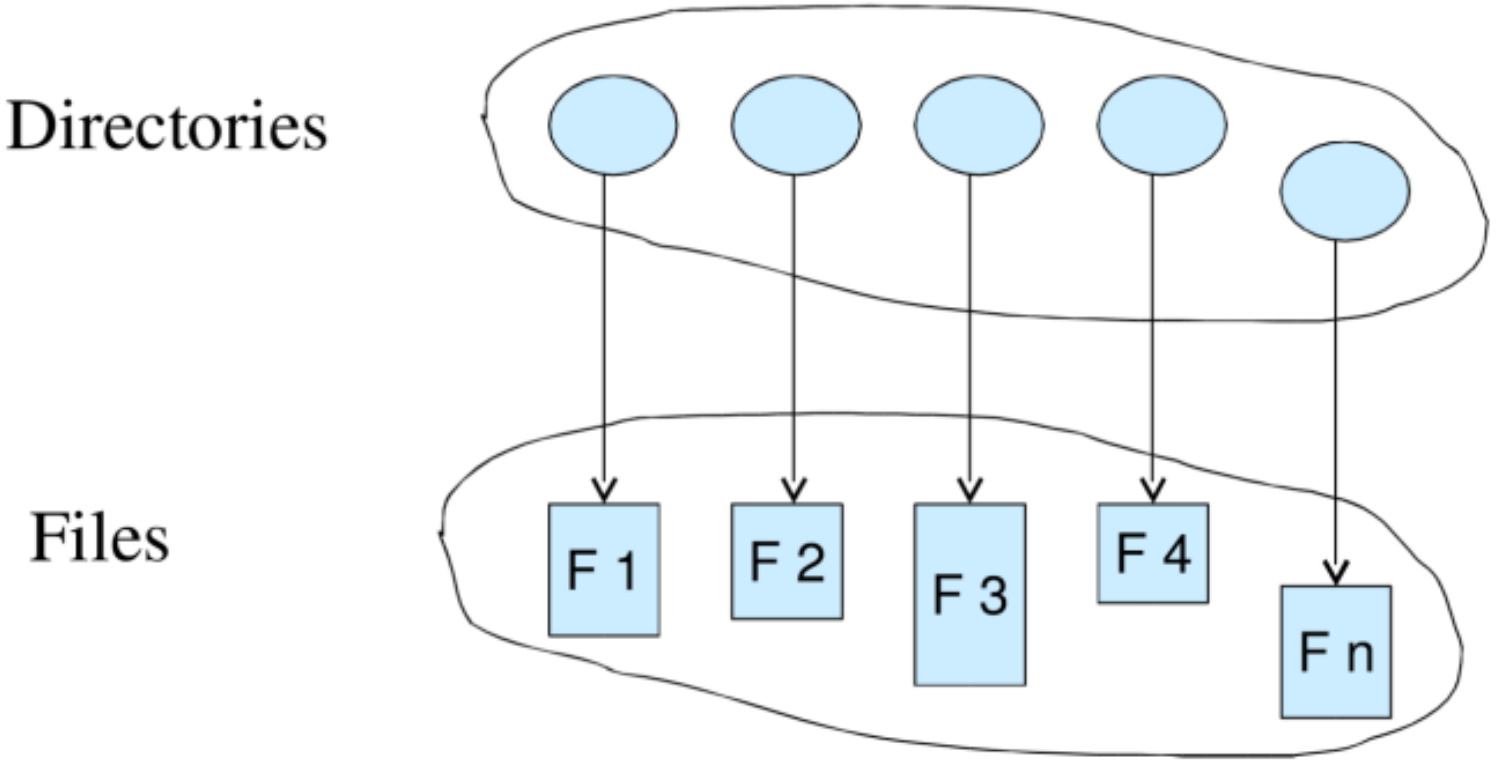
- **OS provides system calls** to perform operations on files.
- Some common calls are:
- ***Create:*** If disk space is available it create new file without data.
- ***Delete:*** Deletes files to free up disk space.
- ***Open:*** Before using a file, a process must open it.
- ***Close:*** When all access are finished, the file should be closed to free up the internal table space.
- ***Read:*** Reads data from file.

FILE OPERATIONS

- ***Append:*** Adds data at the end of the file.
- ***Seek:*** Repositions the file pointer to a specific place in the file.
- ***Get attributes:*** Returns file attributes for processing.
- ***Set attributes:*** To set the user settable attributes when file changed.
- ***Rename:*** Rename file.

DIRECTORY STRUCTURE

- A directory is a **node containing information about files.**



SINGLE-LEVEL-DIRECTORY:

- All files are contained in the same directory.
- Easy to support and understand;
- but difficult to manage
  large amount of files and to manage different users.

TWO-LEVEL-DIRECTORY:

- Separate directory for each user.
- Used on a multiuser computer and on a simple network computers.
- It has problem when
  - users want to cooperate on some task and to access one another's files.
  - a single user has large number of files.

HIERARCHICAL-DIRECTORY:

- Generalization of two-level-structure to a tree of arbitrary height.
- This allow the user to create their own subdirectories and to organize their files accordingly.
- To allow to share the directory for different user
- Acyclic-graph-structure is used.
- Nearly all modern file systems are organized in this manner.

PATH NAMES

- ***Absolute***
- Path name starting from root directory to the file.
  - **e.g. In Unix:** /usr/user1/bin/lab2.
  - **E.g. In Windows:** C:\Users\Ananta\Teaching_Materials\3rd Sem\OS\Lecture_Slides
- path separated by **/ in Unix** and **\ in windows**.

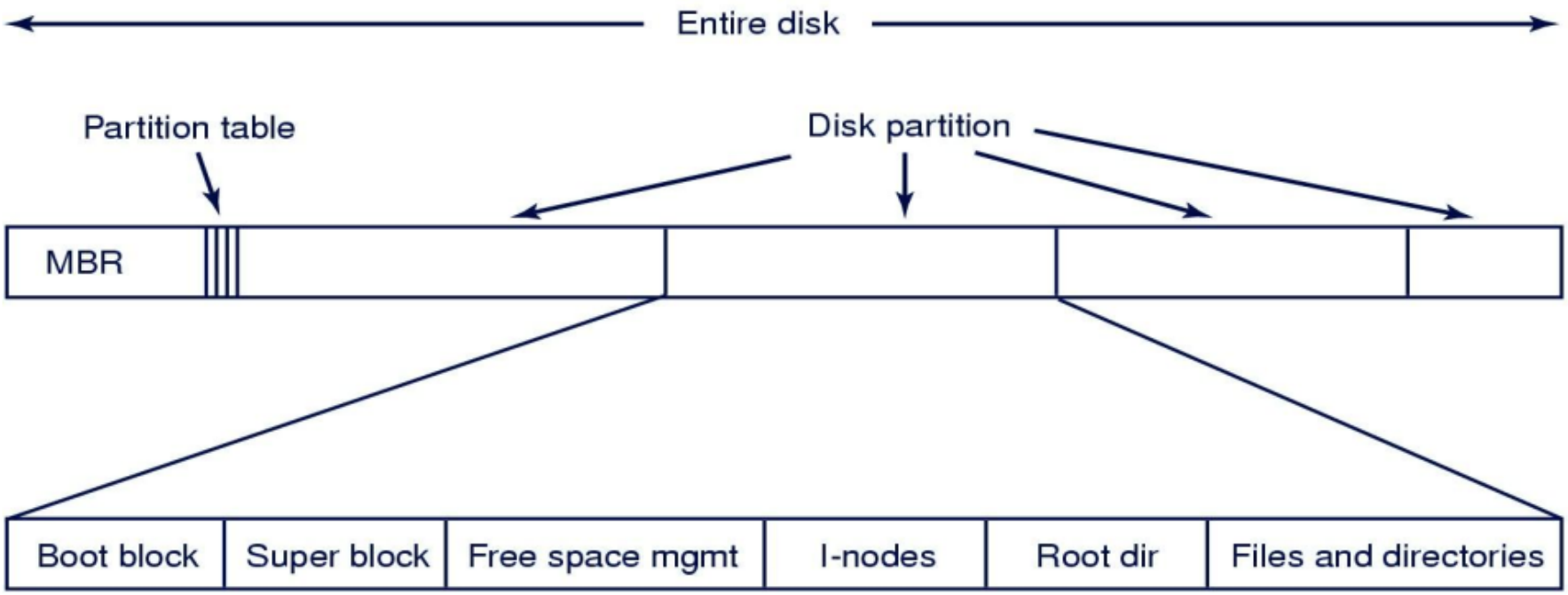
- ***Relative***
- Concept of **working directory**.
- A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory. E.g., bin/lab2 is enough to locate same file if current working directory is /usr/user1

FILE-SYSTEM IMPLEMENTATION

- How files and directories are stored?
- How disk space is managed?
- How to make every thing work efficiently and reliably?

FILE-SYSTEM LAYOUT

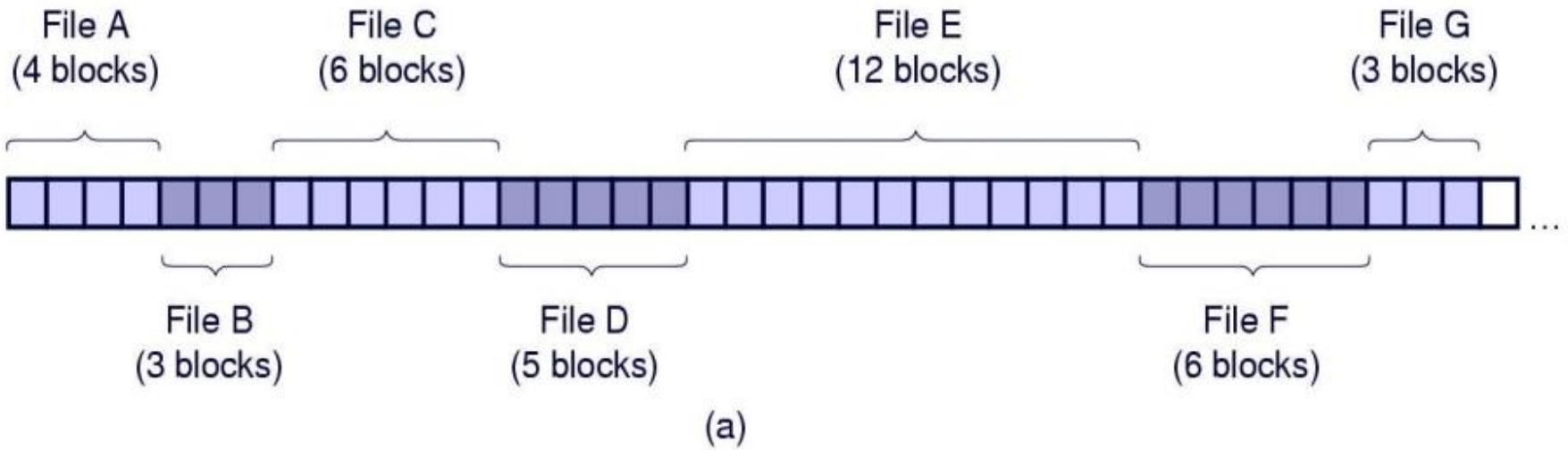
- Disks are divided up into one or more partitions, with independent file system on each partition.
- **BootBlock** has **bootstrap program** that initializes the operating system (OS) during startup.
- a ***super block*** describes the state of the file system: the total size of the partition, type (ext2, ext3…) the block size etc.
- In linux every file is recognized with integer number known as **inode number.**

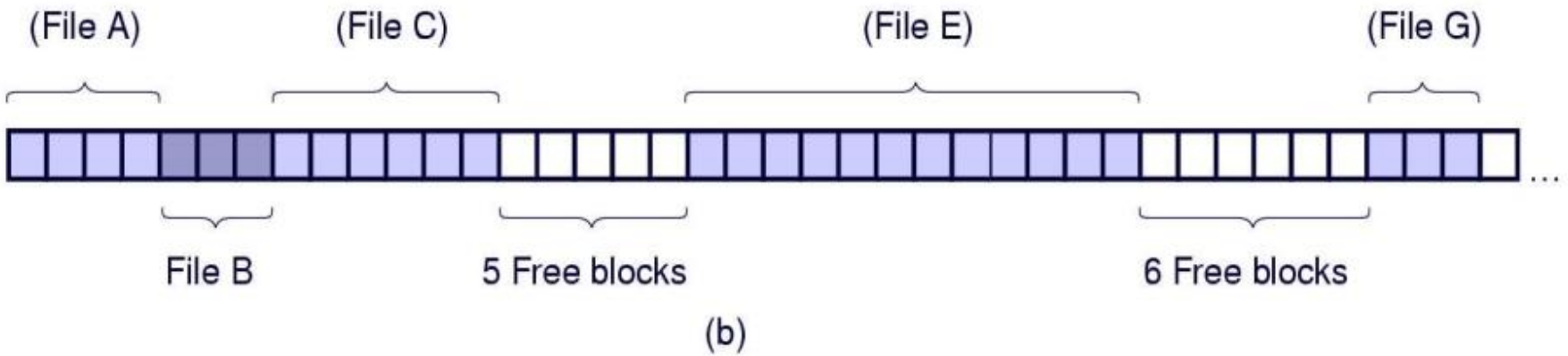


ALLOCATION METHODS: CONTIGUOUS ALLOCATION

- *Each file occupy a set of contiguous block on the disk*
- Disk addresses define a linear ordering on the disk.
- File is defined by the disk address and length in block units.
- With 2-KB blocks, a 50-KB file would be allocated 25 consecutive blocks.
- *Both sequential and direct access can be supported by contiguous allocation*

CONTIGUOUS ALLOCATION





CONTIGUOUS ALLOCATION

- **Advantages:**
  Simple to implement:
  - accessing a file that has been allocated contiguously is easy.

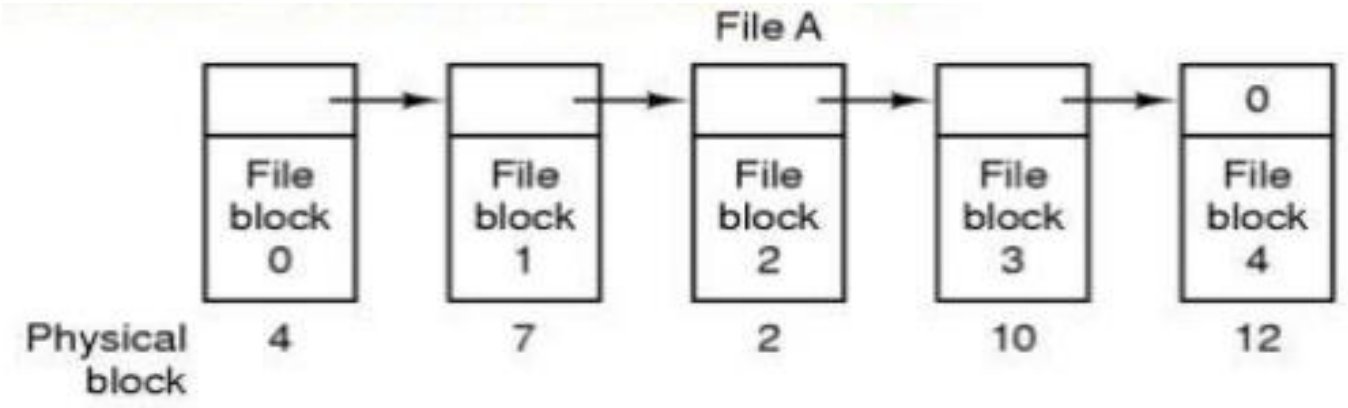
  High performance:
  - the entire file can be read in single operation i.e. decrease the seek time.
- **Problems:**
  Fragmentation:
  - when files are allocated and deleted the free disk space is broken into holes.

  Dynamic-storage-allocation problem:
  - searching of right holes.
    Required pre-information of file size.

- *Due to its good performance it used in many system; it is widely used in* ***CD-ROM file system.***

LINKED ALLOCATION

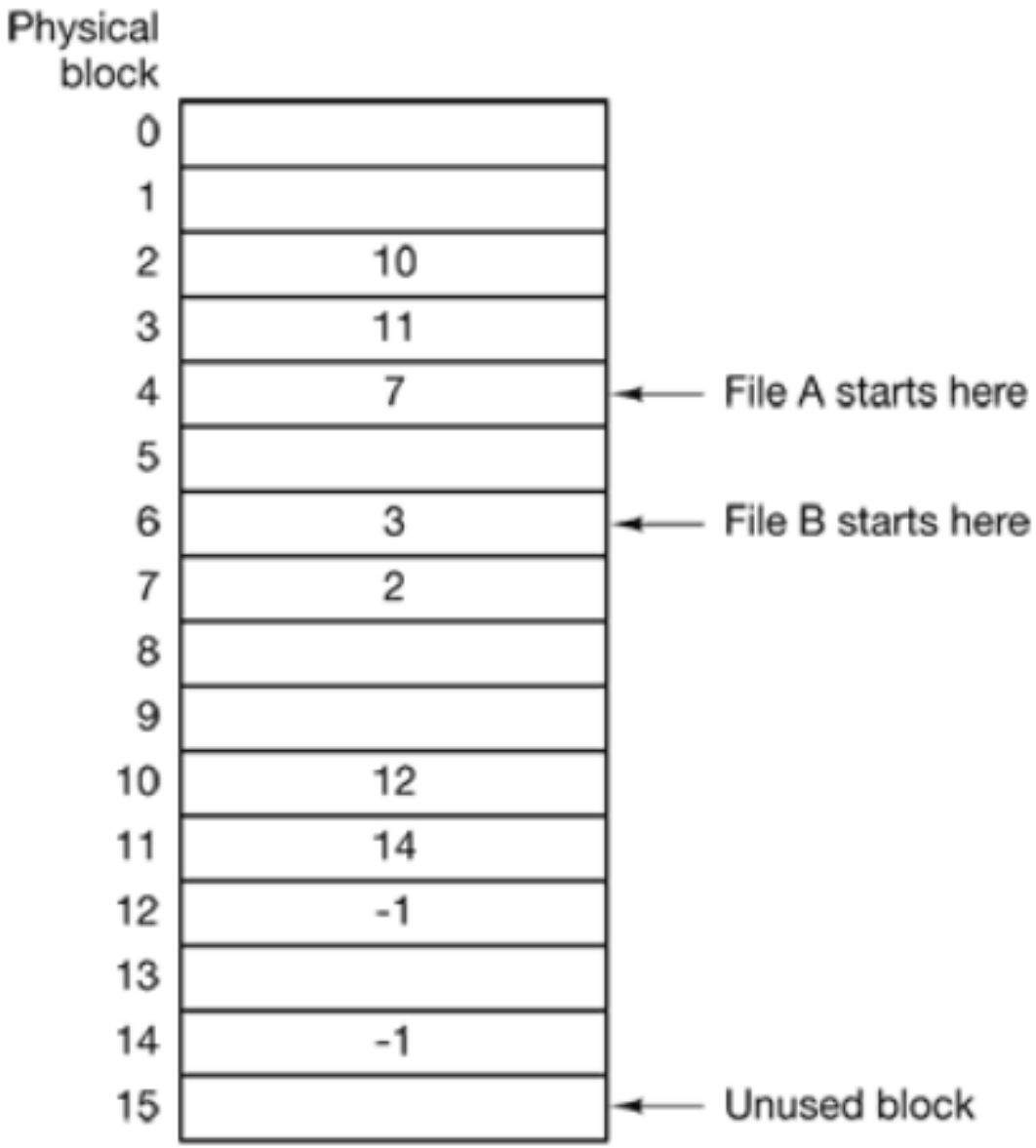
- *Each file is a linked list of disk blocks;*
- Each block contain the pointer to the next block of same file.
- Unlike contiguous allocation, every disk block can be used in this method.
- To create the new file, we simply create a new entry in the directory; with linked allocation, each directory entry has a pointer to the first block of the file.



LINKED ALLOCATION

- It solves Disk Fragmentation problems of contiguous allocation.
- Problems,
- Random access is extremely slow.
- Read Size is not power of 2, **peculiar size**
  - first few bytes of each block occupied to a pointer to the next block, reads of the full block size require acquiring and concatenating information from two disk blocks, which generates extra overhead.
- ***Solution: Using File Allocation Table (FAT)*.**
  The table has one entry for each disk block containing next block number for the file.
- This resides at the beginning of each disk partition.

LINKED ALLOCATION USING FAT



LINKED ALLOCATION USING FAT

- Advantages:
  - The entire block is available for data.
  - random access time is improved

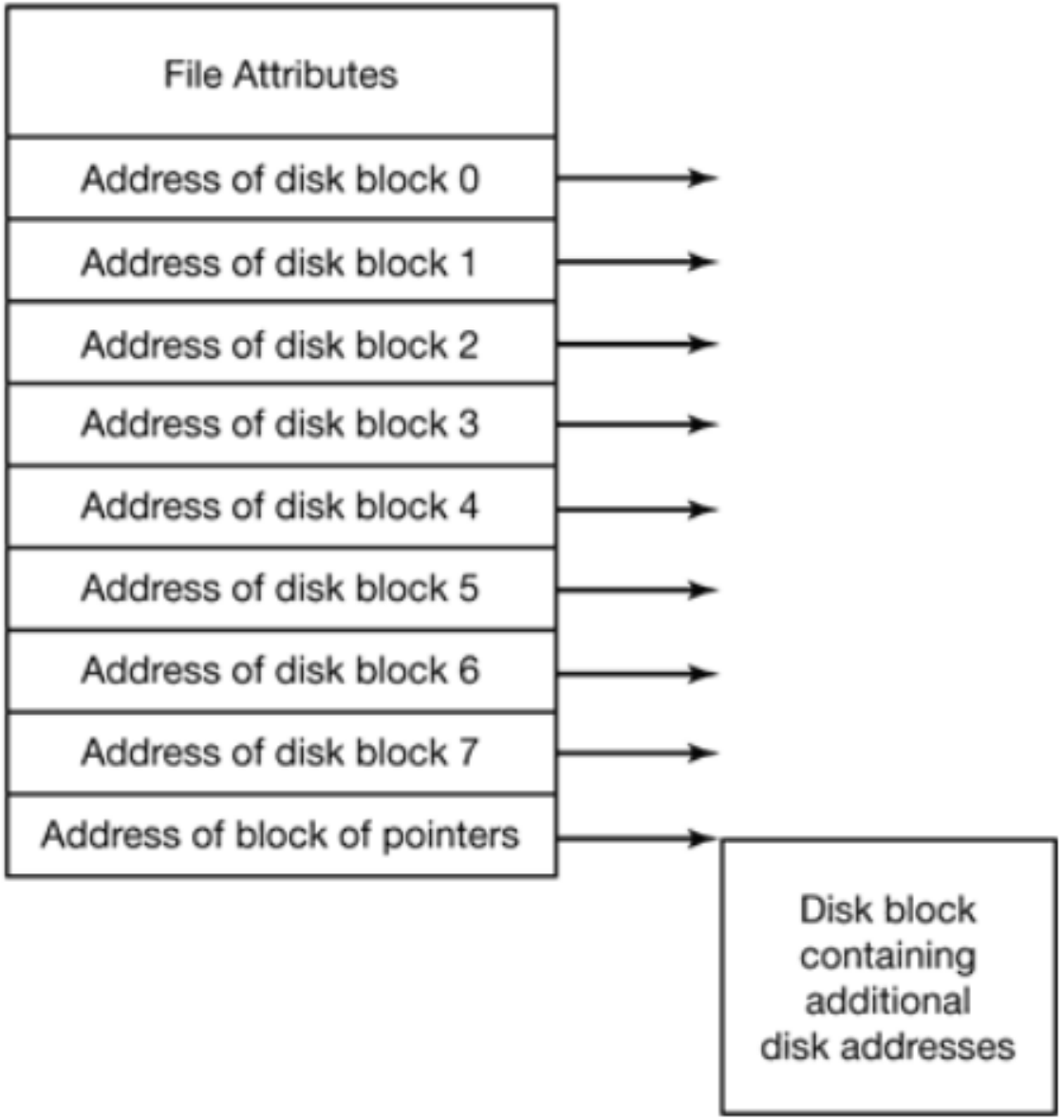
- Problems:
  - The entire table must be in memory all the time to make it work.
  - With 20GB disk and 1KB block size, the table needs 20 millions entries. Suppose each entry requires 3 bytes. Thus the table will take up 60MB of main memory all the time.

I-NODE (INDEX NODE)

- I-node is a data structure, Associated with each file
- Lists file attributes and disk address of each blocks in file

- Advantage
  - far smaller than the space occupied by the file table
  - need only in memory when the corresponding file is open

I-NODE


File Attributes
Address of disk block 0
Address of disk block 1
Address of disk block 2
Address of disk block 3
Address of disk block 4
Address of disk block 5
Address of disk block 6
Address of disk block 7
Address of block of pointers
Disk block containing additional disk addresses

DIRECTORY IMPLEMENTATION

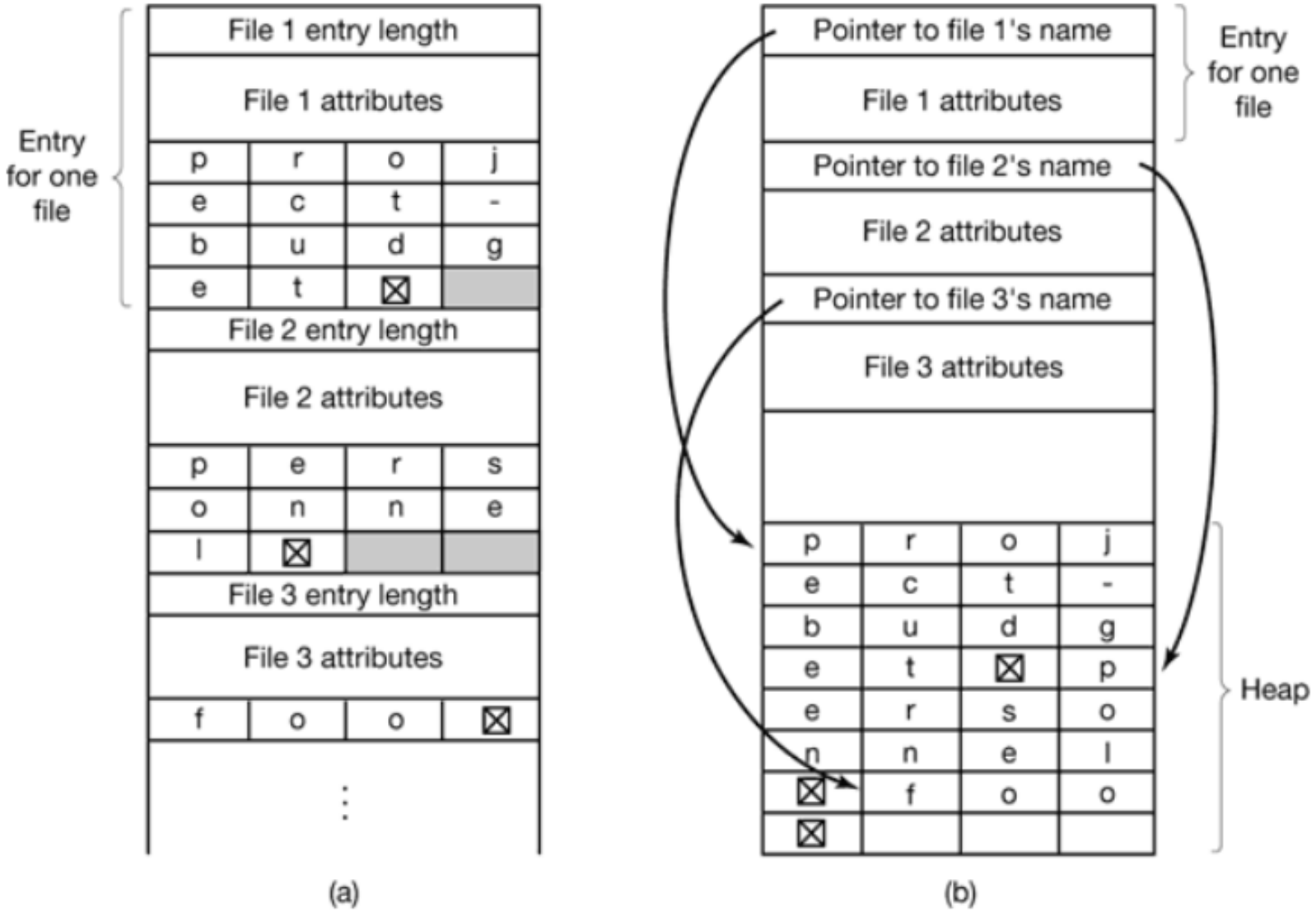
- *The directory entry provides the information needed to find the disk block of the file.*
- *The file attributes are stored in the directory*.

LINEAR LIST

- Use the linear list of the file names with pointer to the data blocks.
- It requires linear search to find a particular entry.
- Advantages: Simple to implement.
- Problem:
  - when a file is removed, a variable-sized gap is introduced
    - next file to be entered may not fit
    - But compacting the directory is feasible because it is entirely in memory.
  - single directory entry may span multiple pages, so a **page fault may occur** while reading a file name

IN A HEAP

- make the directory entries themselves all fixed length
- keep the file names together in a heap at the end of the directory
- Advantage:
  - when an entry is removed, the next file entered will always fit there
- Disadvantage:
- In all of the designs so far, directories are searched linearly from beginning to end when a file name has to be looked up. For extremely long directories, **linear searching can be slow.**


Entry for one file
File 1 entry length
File 1 attributes
p r o j
e c t -
b u d g
e t
File 2 entry length
File 2 attributes
p e r s
o n n e
l
File 3 entry length
File 3 attributes
f o o
(a)
Pointer to file 1's name
File 1 attributes
Pointer to file 2's name
File 2 attributes
Pointer to file 3's name
File 3 attributes
Entry for one file
p r o j
e c t -
b u d g
e t p
e r s o
n n e l
f o o
Heap
(b)

HASH TABLE

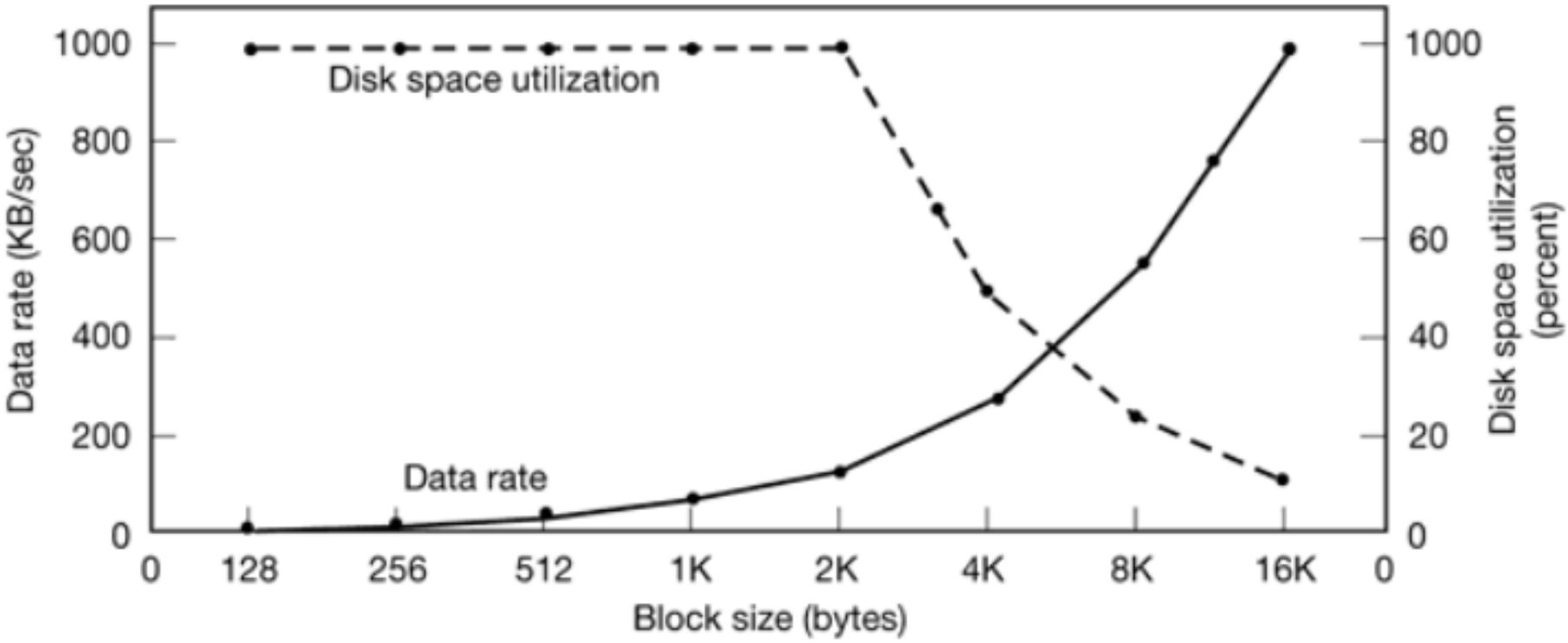
- *It consist linear list with hash table.*
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- If that key is already in use, a linked list is constructed.

- **Advantages:** greatly decrease the file search time.
- **Problem:** complex administration

DISK SPACE MANAGEMENT

- **Block Size**
- Nearly all file systems chop files up into fixed-size block.
- Using a large size result the wastage of disk space.
- Using a small size increase the seek and rotational delay- low data access rate

BLOCK SIZE


Disk space utilization
Data rate
Data rate (KB/sec)
Disk space utilization (percent)
Block size (bytes)
0
200
400
600
800
1000
0
20
40
60
80
1000
0
128
256
512
1K
2K
4K
8K
16K

KEEPING TRACK OF FREE BLOCKS

- *To keep track of free blocks, system maintains the freespace-list.*
- The free-space-list records all free blocks- those not allocated to some file or directory .
- To create a file, system search the free-space-list for required amount of space, and allocate that space to the new file and removed from free-space-list.
- When a file is deleted, its disk space is added to the freespacelist.
- *The free-space-list can be implemented in two ways:*
  Bitmap
  Linked list

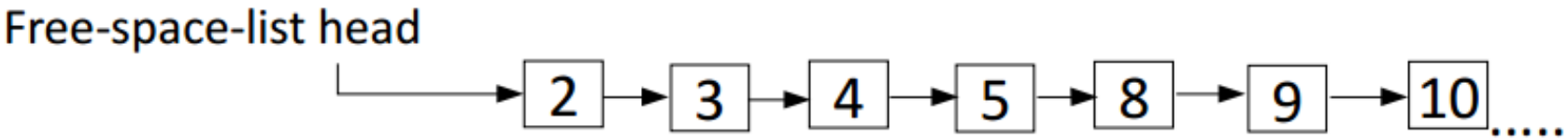
BITMAP

- *Each block is represented by a bit. If the block is free, the bit is 1; if the block is allocated the bit is 0.*
- A disk with n blocks requires a bitmap with n bits.
- ex: Consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18…are free, and rest are allocated.
- The free space bit map would be 00111100111111100010.....
- **Advantages:** Simple and efficient in finding first free block, or
  n consecutive free blocks.
- **Problems:** Inefficient unless the entire bitmap is kept in main memory.
- Keeping in main memory is possible only for small disk, when disk is large the bitmap would be large.

- *Many system use bitmap method.*

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.*
- The first block contains a pointer to the next free block.
- The previous example can be represented in linked as



- Advantages: Only one block is kept in memory.
- Problems: Not efficient; to traverse list, it must read each block.

FILE SYSTEM RELIABILITY

- **Destruction of a file system** is often a far greater **disaster** than destruction of a computer.
- **restoring** all the information will be **difficult, time consuming,** and in many cases, **impossible**
- **Backups:**
  - disk crash, fire, flood, or other natural catastrophe
  - often accidentally remove files that they later need again
- should the entire file system be backed up or only part of it?
- executable programs can be reinstalled
- Temp folder not necessary
- it is wasteful to back up files that have not changed since the last backup
- **Incremental Dumps:** Backup only those which have been changed
- Single bad spot on compressed back up file will be corrupted while decompression

HOME WORKS ??