File Management

Overview

- File system – From user's point of view one of the most important part of the OS.
- File system allows the user to create a collection of data called files with the following properties –

1. Long-term existence
   - Stored on disk or secondary storage
2. Sharable between processes
   - Access can be controlled, with permissions
3. Structure
   - Depending on the file structure, a file can have internal structure convenient for a particular application.
   - Files can be organized in hierarchy or more complex structure – to reflect relationships among them.

A file system also provides a collection of functions that can be performed on files.

- **Create** – define new file and position it within file structure.
- **Delete** – remove from the file structure and destroyed.
- **Open** – to allow a process to perform functions on it.
- **Close** – close with respect to a process.
- **Read** – read all or a portion of a file.
- **Write (update)** – add new data, or change values.

File Management System

- A set of system software.
- The way a user of application may access files is through the FMS
- Programmer does not need to develop file management software

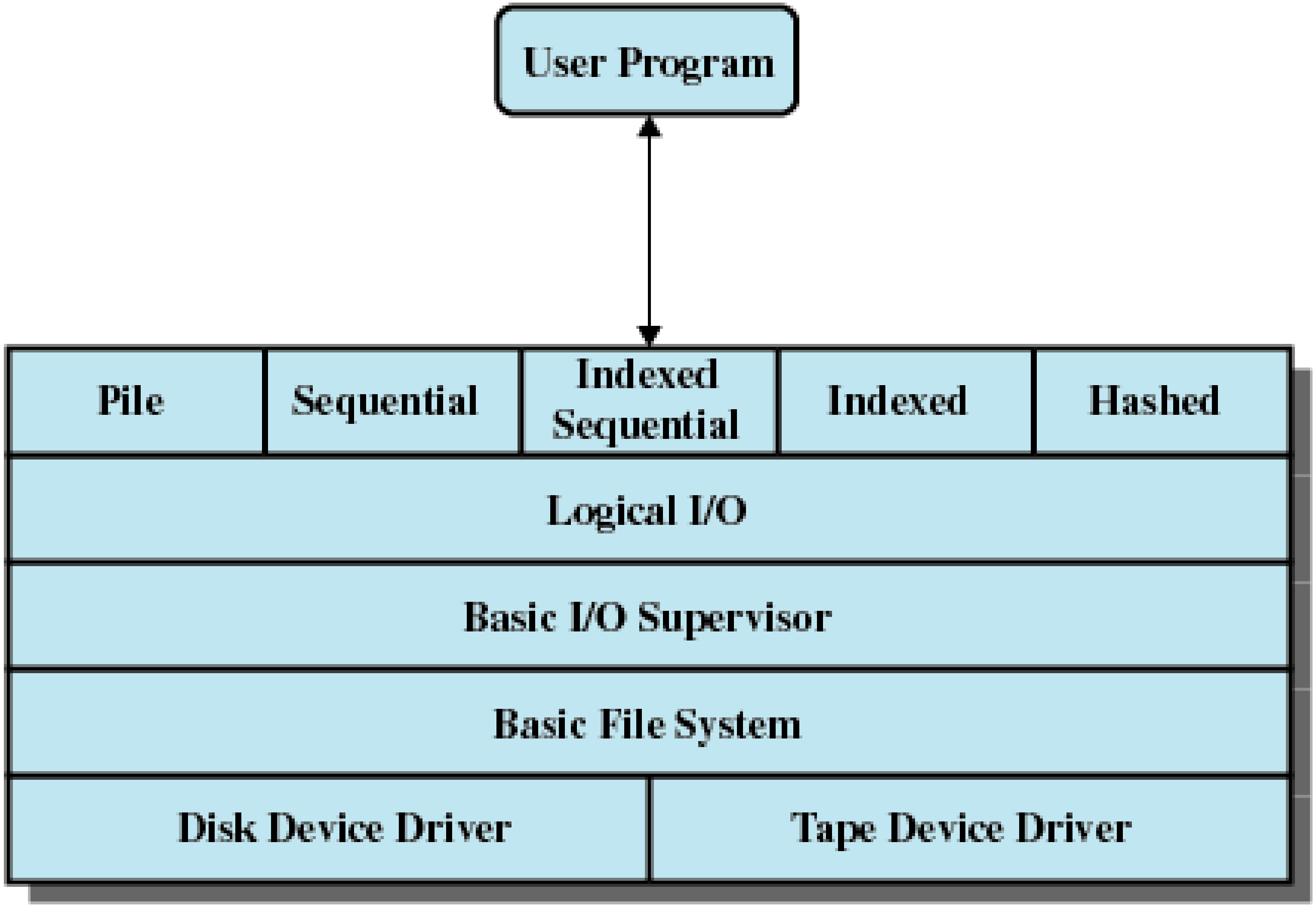


Figure 12.1 File System Software Architecture

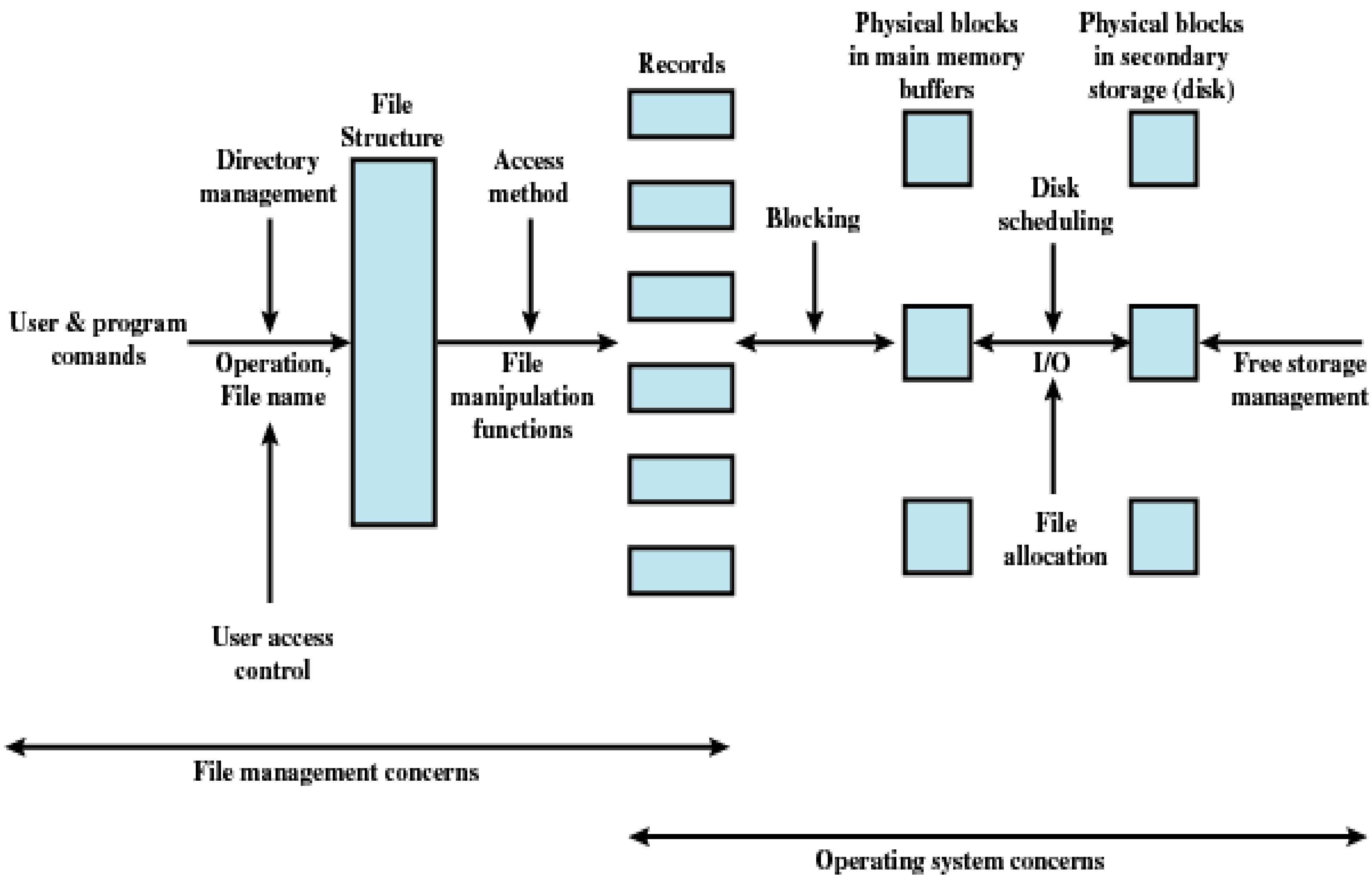
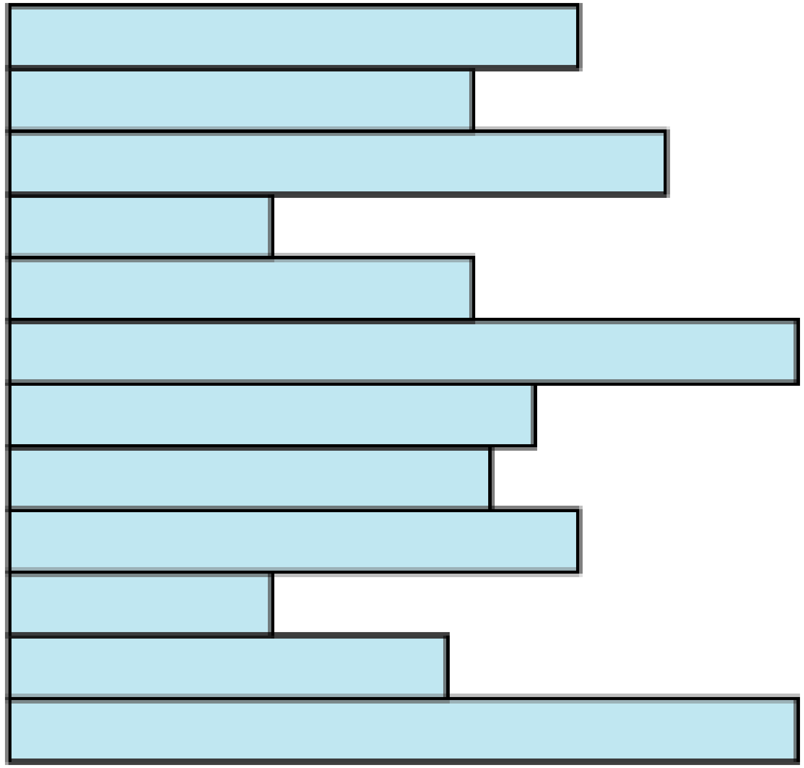


Figure 12.2 Elements of File Management

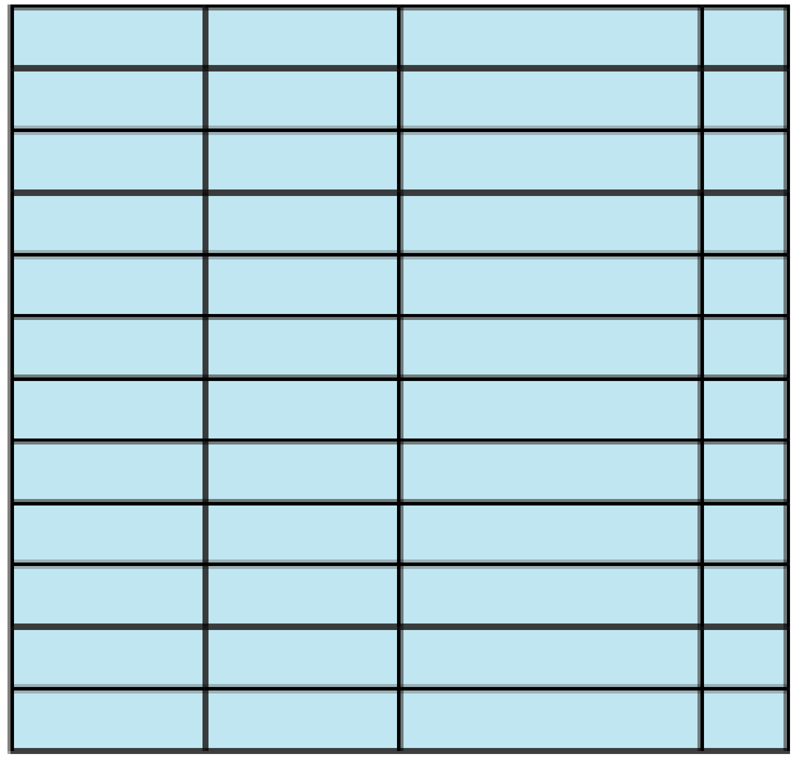
Pile



Variable-length records
Variable set of fields
Chronological order

(a) Pile File

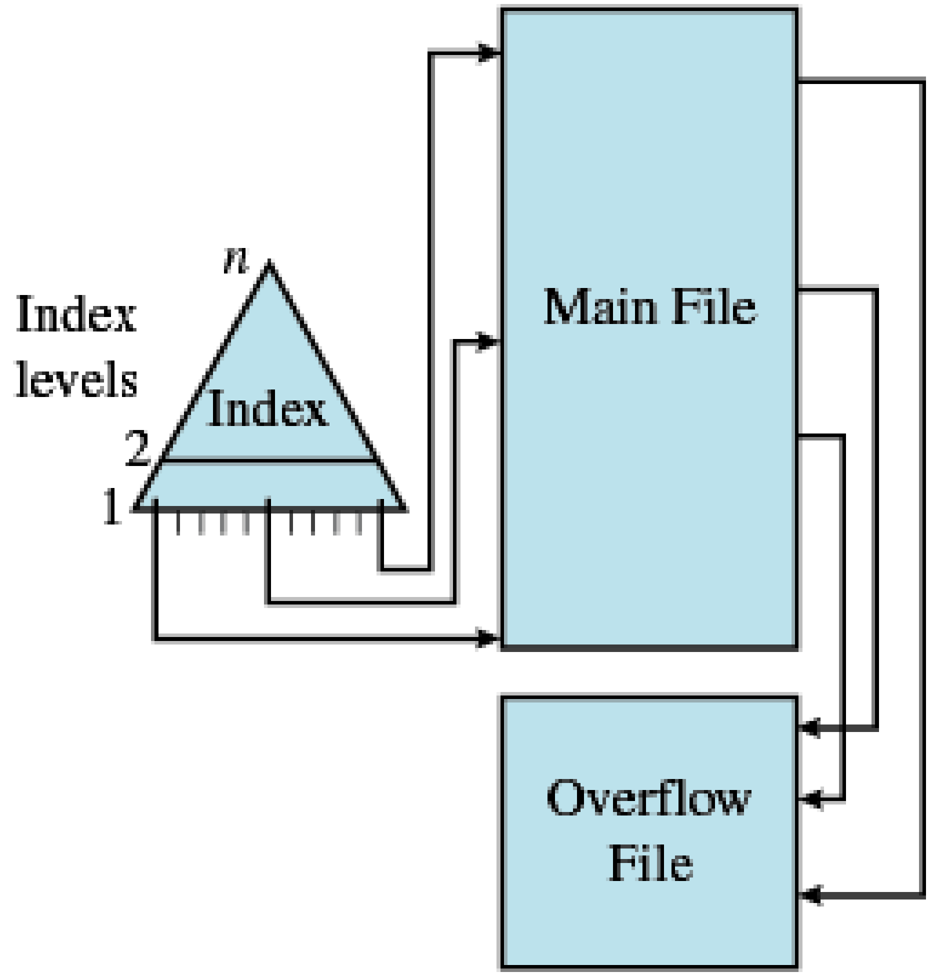
Sequential File



Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

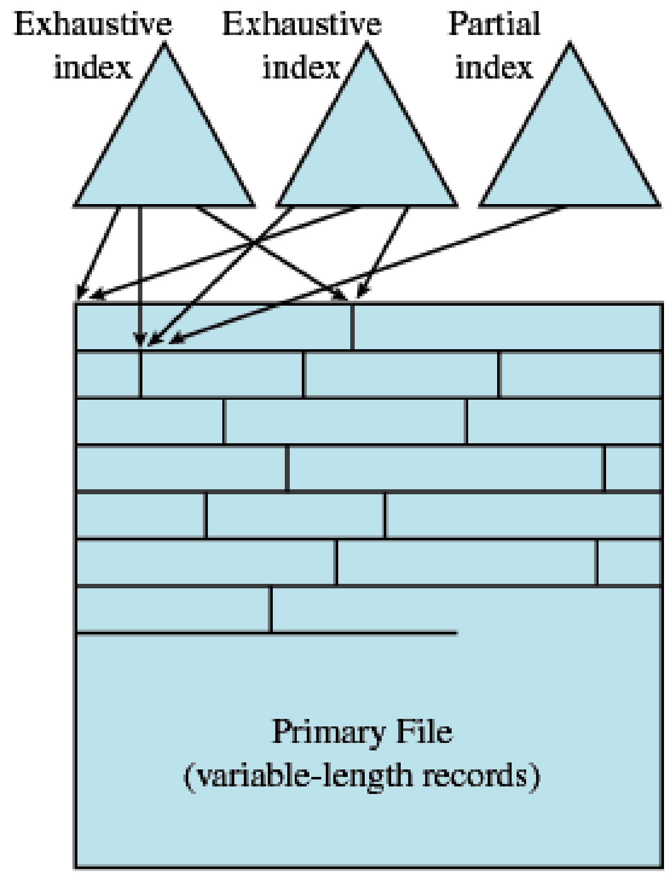
(b) Sequential File

Indexed Sequential File



(c) Indexed Sequential File

Indexed File



(d) Indexed File

Direct or Hashed File

- Directly access a block at a known address
- Key field required for each record

File Directories

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system
- Provides mapping between file names and the files themselves

Simple Structure for a Directory

- List of entries, one for each file
- Sequential file with the name of the file serving as the key
- Provides no help in organizing the files
- Forces user to be careful not to use the same name for two different files

Two-level Scheme for a Directory

- One directory for each user and a master directory
- Master directory contains entry for each user
    - Provides address and access control information
- Each user directory is a simple list of files for that user
- Still provides no help in structuring collections of files

Hierarchical, or Tree-Structured Directory

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries

Hierarchical, or Tree-Structured Directory

- Files can be located by following a path from the root, or master, directory down various branches
  - This is the pathname for the file
- Can have several files with the same file name as long as they have unique path names

Hierarchical, or Tree-Structured Directory

- Current directory is the working directory
- Files are referenced relative to the working directory

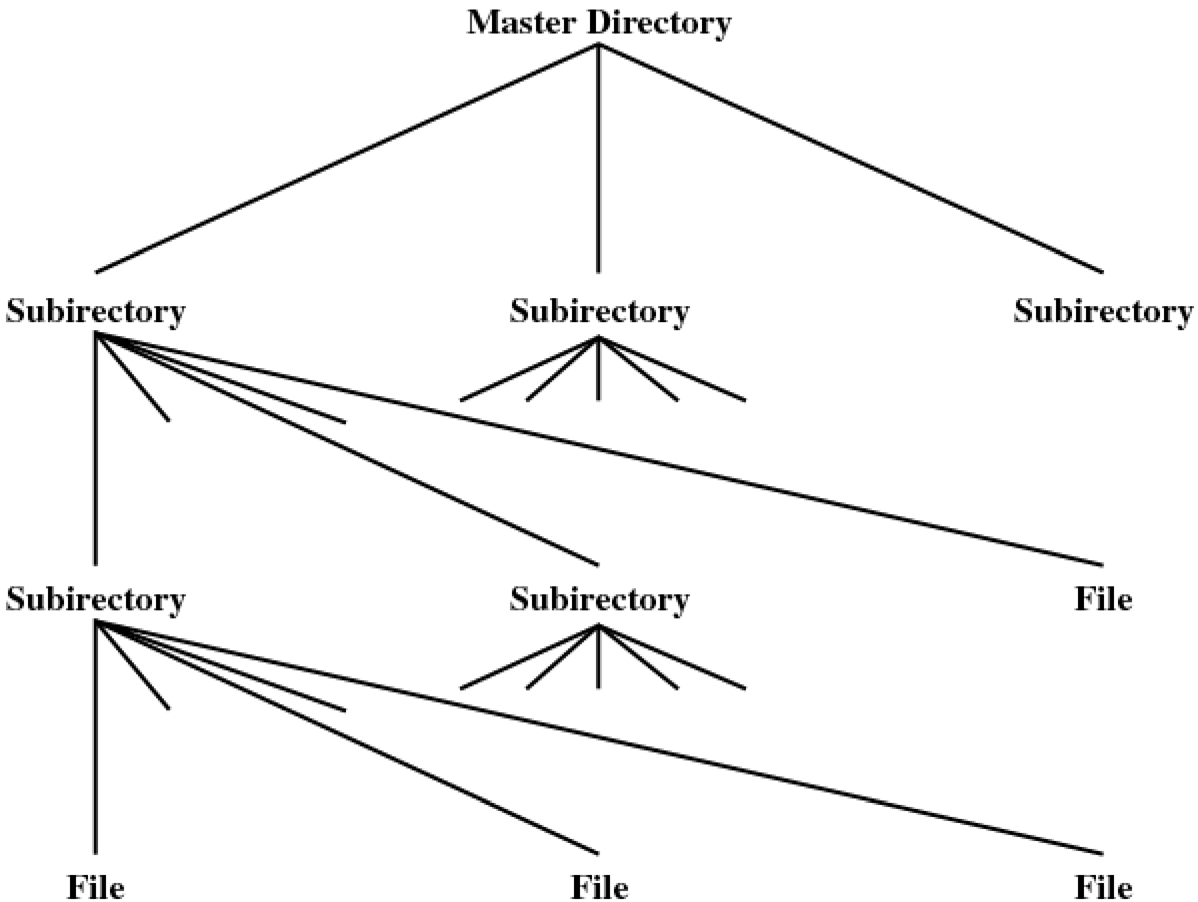


Figure 12.4 Tree-Structured Directory

Secondary storage management

- On secondary storage a file is a collection of blocks; the operating system or file management system is responsible for allocating blocks to files
- **Two management issues**
  - Space on secondary storage must be allocated to files
  - Keep track of the space available for allocation
- The approach taken for file allocation may influence the approach taken for available space management

Secondary storage management

- Three common **file allocation** methods are :
  - **Contiguous allocation**
    - Single contiguous set of blocks is allocated to a file at the time of file creation
  - **Chained allocation**
    - Each block contains a pointer to the next block in the chain
  - **Indexed allocation**
    - The file allocation table contains a separate one level index for each file; the index has one entry for each allocated portion (unit) to the file.

Secondary storage management

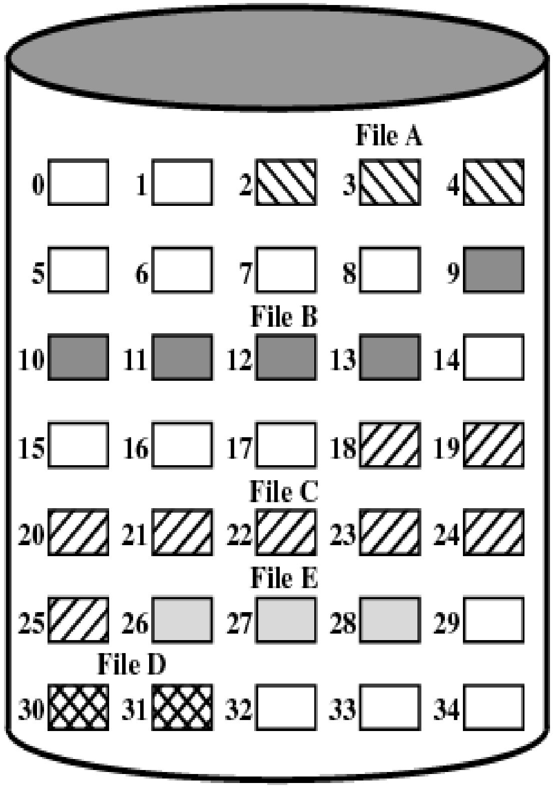
1. Contiguous allocation:

- A single contiguous set of blocks assigned to a file when it is created
- Only a single entry in the file allocation table showing the starting block and length of the file.
- Multiple blocks can be read at a time in case of sequential processing.
- Assuming contiguous blocks are stored on one track , disk seek time is minimum.
- How to satisfy a request of size n from a list of free holes? Strategies used are first fit, best fit and next fit.
- Disadv

Suffers from external fragmentation.

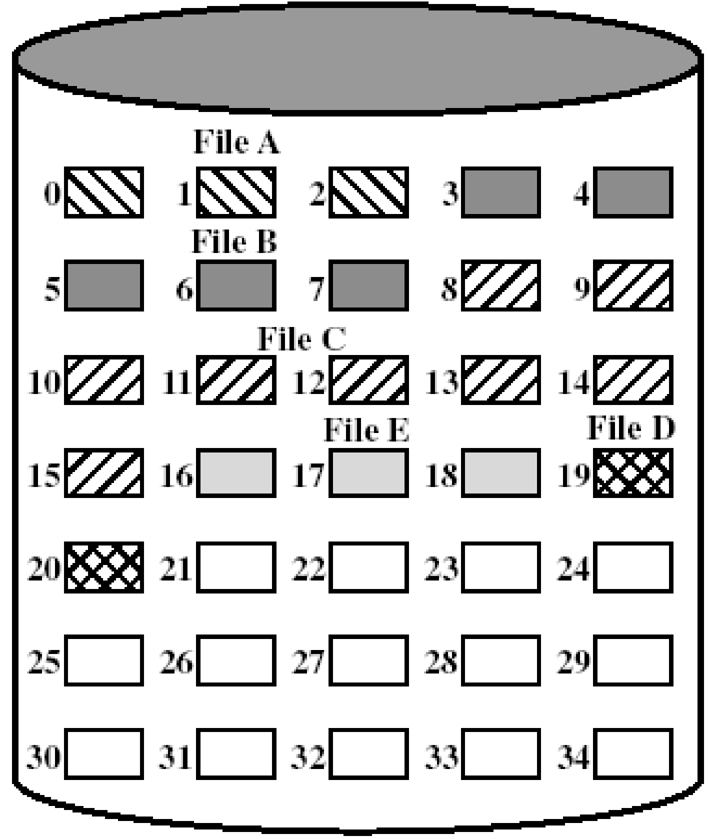
As files are allocated and deleted, free space on the disk is broken down into smaller pieces.

Soln : USE COMPACTION



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.7 Contiguous File Allocation

Before Compaction

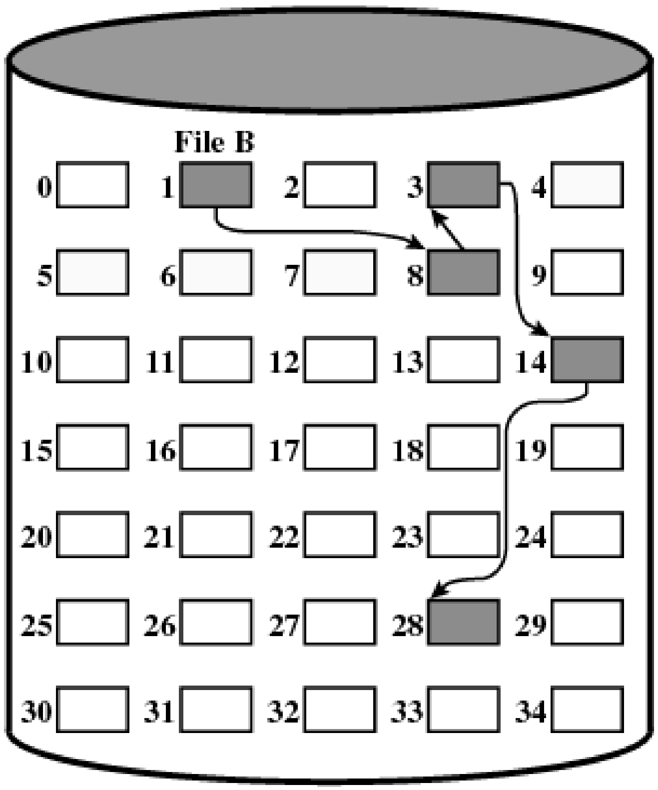
After Compaction

Secondary storage management

2. Chained Allocation

- Each file is a linked list of disk blocks; the blocks can be scattered anywhere on the disk
- Each block contains a pointer to the next block in the chain
- File allocation table consists of
  - Starting block and length of file
- No external fragmentation
  - Any free block can be added to a chain
- Best for sequential files
- No accommodation of the principle of locality

Secondary storage management



File Allocation Table

File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...

Note : Some FATs may contain **start block** and **end**(instead of length)

Figure 12.9 Chained Allocation

Secondary storage management

Disadv :

- No accommodation of the principle of locality.

  If it is required to bring in several blocks of file at a time, then a series of accesses to different parts of the disk are required (Increase in head movement).

Secondary storage management

3. Indexed Allocation

- File allocation table contains one level index for each file.
- The file index for a file is kept in a separate block and the entry for the file allocation table points to that block
- Eliminates external fragmentation

Secondary storage management

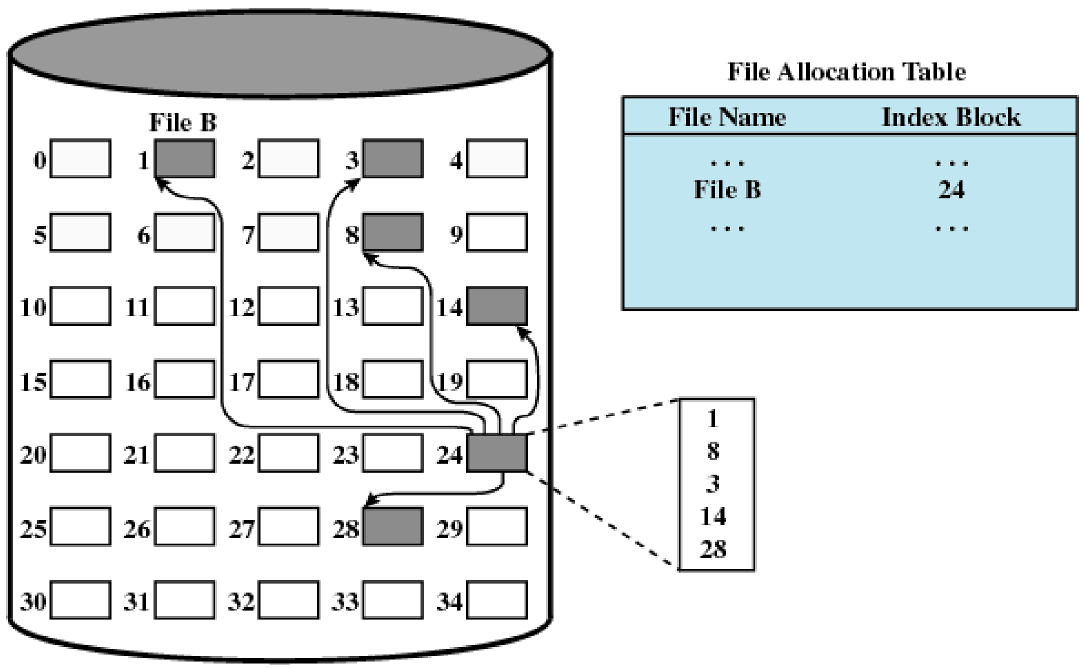


Figure 12.11 Indexed Allocation with Block Portions

Free Space Management

- Just as allocated space must be managed, so must the unallocated space.
- To perform file allocation, it is necessary to know which blocks are available
- A *disk allocation table* is needed in addition to a file allocation table

Free Space Management

1. Bit Tables(Bit vector)

- This method uses a vector containing one bit for each block on the disk
- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use.
- For example, for the disk layout of 35 blocks

00111000011111000011111111111011000

Means 0 1 5 6 7 8 …… are free

Free Space Management

Adv:

Easy to find one or more contiguous group of free blocks.

Disadv:

- The amount of memory (in bytes) required for a block bitmap is

  disk size in bytes/(8 * file system block size)

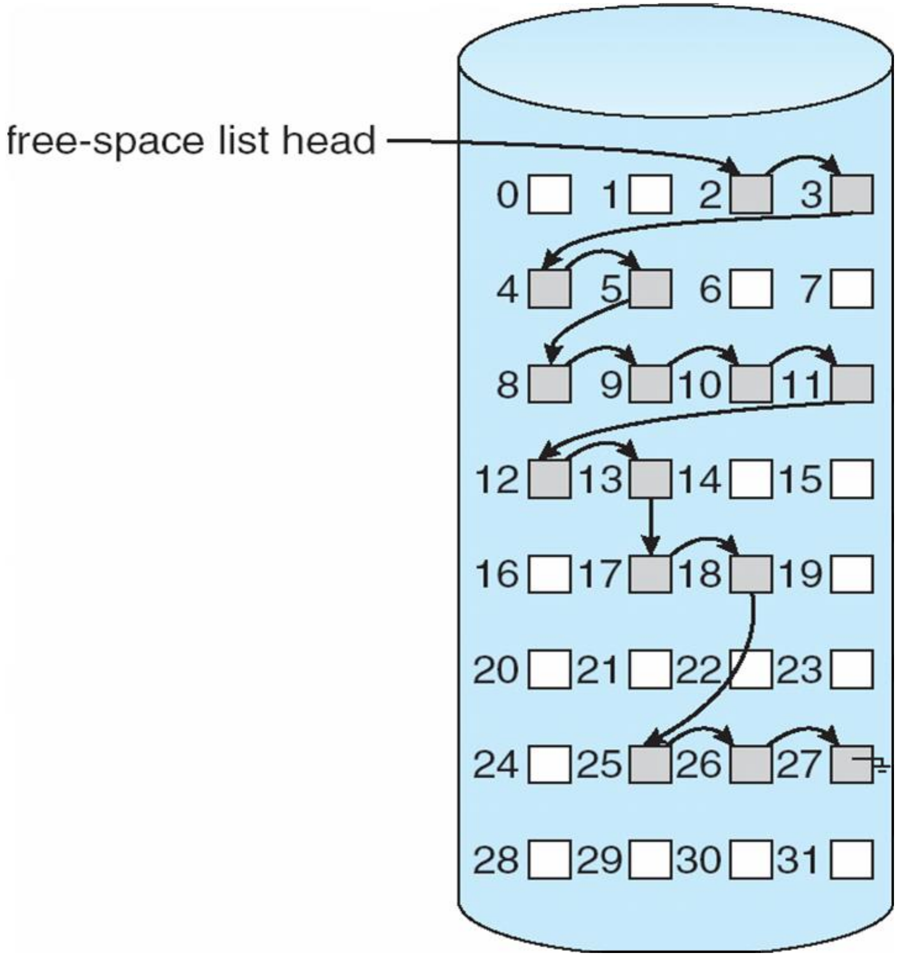
- Thus, for a 16-Gbyte disk with 512-byte blocks, the bit table occupies about

  4 Mbytes.
- Can we spare 4 Mbytes of main memory for the bit table?

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.
- This first block contains a pointer to the next free disk block and so on.
- Not efficient since to traverse the list we must read each block, which requires a substantial I/O.

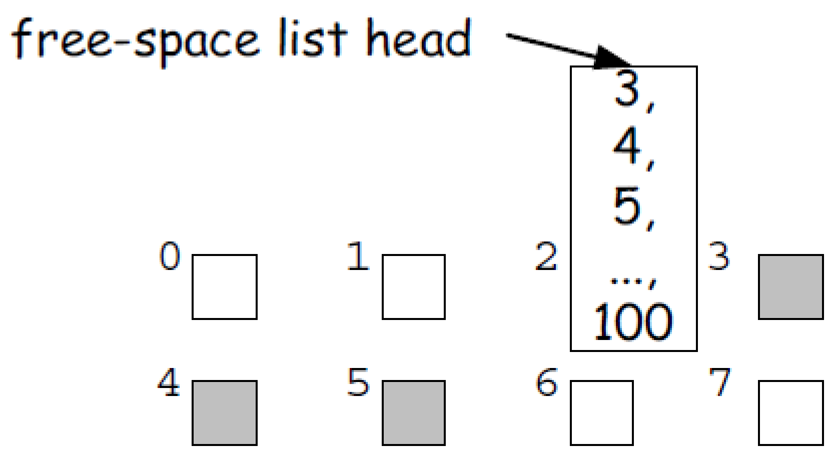

free-space list head
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31

Free Space Management

3. Grouping

- A modification of the linked-list approach is to store the addresses of n free blocks in the first free block(if the block can hold upto n addresses).
- The first n-1 of these are actually free.
- The last one is the disk address of another block containing addresses of another n free blocks.
- The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

Free Space Management



Free Space Management

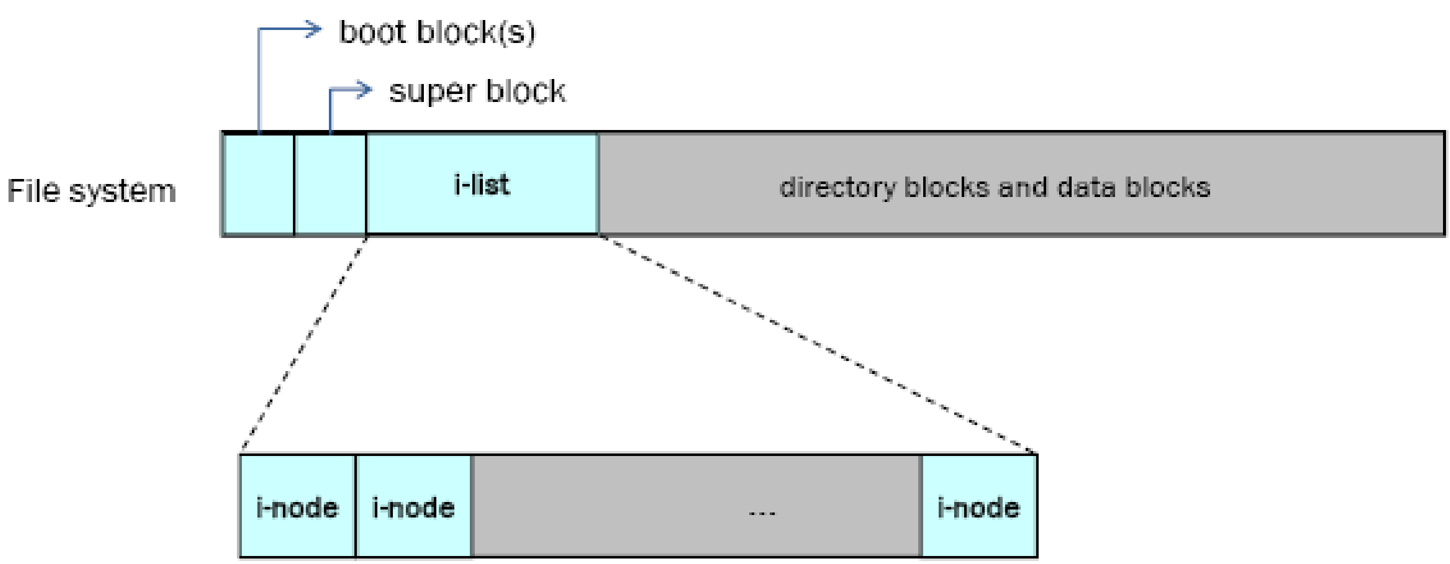
4. Counting

- Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when contiguous allocation is used.
- Thus, rather than keeping a list of free disk addresses, the address of the first free block is kept 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.

Unix file system

- In the original Unix file system, Unix divided physical disks into logical disks called *partitions*.

1. Boot block
2. Super Block
3. Array of inodes
4. Directory and data blocks



Unix file system overview

1. **Boot block** is located in the first few sectors of a file system. The boot block contains the initial bootstrap program used to load the operating system.

2. **Super block** consists of

- the size of the file system
- the number of free blocks in the file system
- a list of free blocks available on the file system
- the size of the inode list
- the number of free inodes in the file system
- a list of free inodes in the file system

Unix file system overview

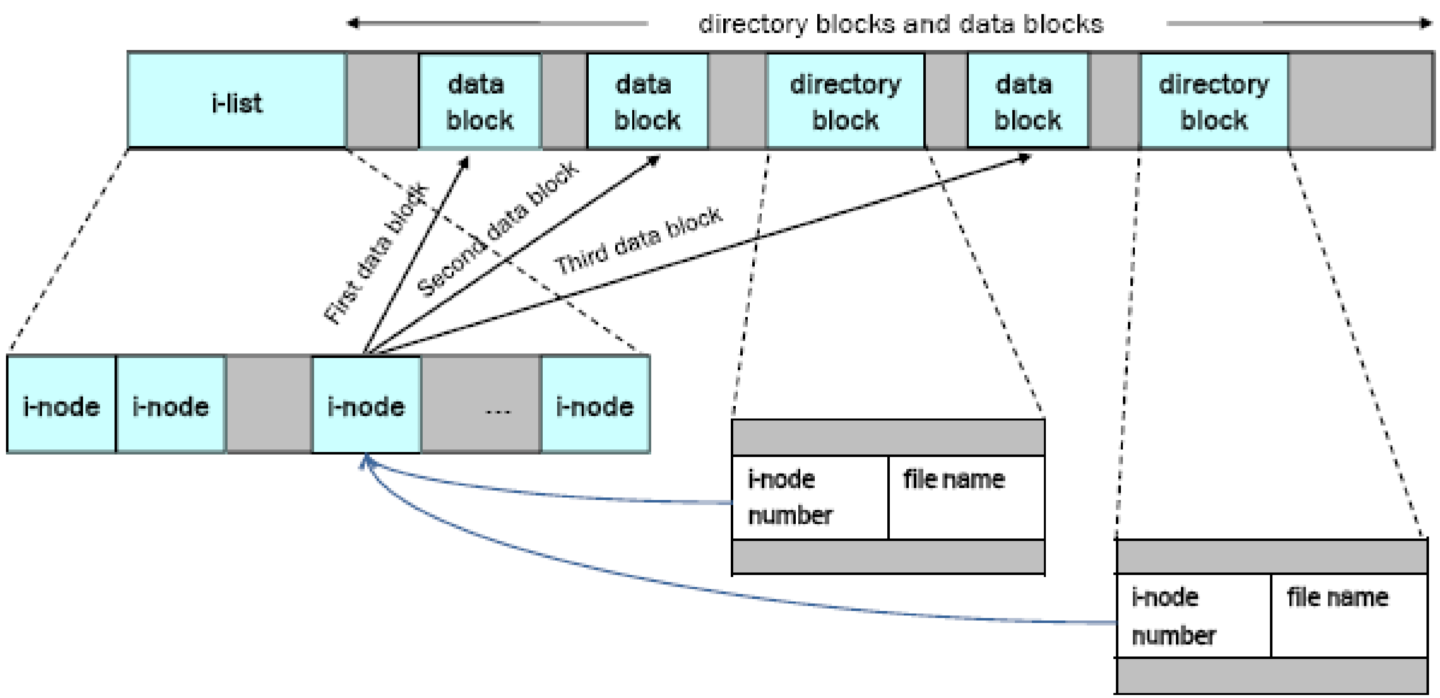
3. Inode is the data structure that describes the attributes of a file, including the layout of its data on disk.

- Every file has a **unique inode**
- Contain the **information** necessary for a process to access a file
- Inode consists of consists of

- file owner identifier
- file type
- file access permissions
- file access times
- number of links to the file
- table of contents for the disk address of data in a file
- file size

Unix file system overview

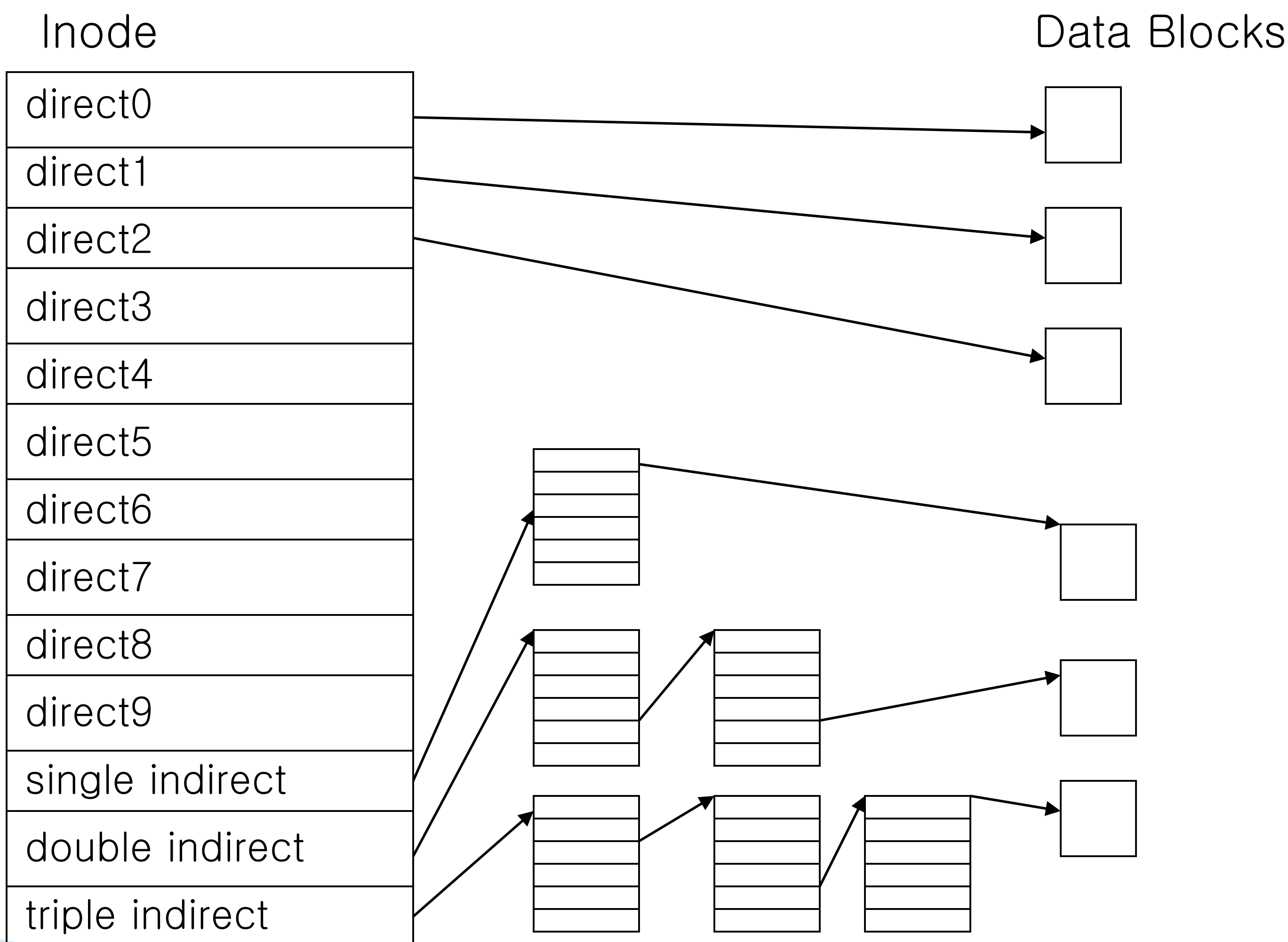
4. *Data* blocks containing the actual contents of files.
There is a one to one mapping of files to inodes and vice versa.
Thus, while users think of files in terms of file names, Unix thinks of files in terms of inodes.

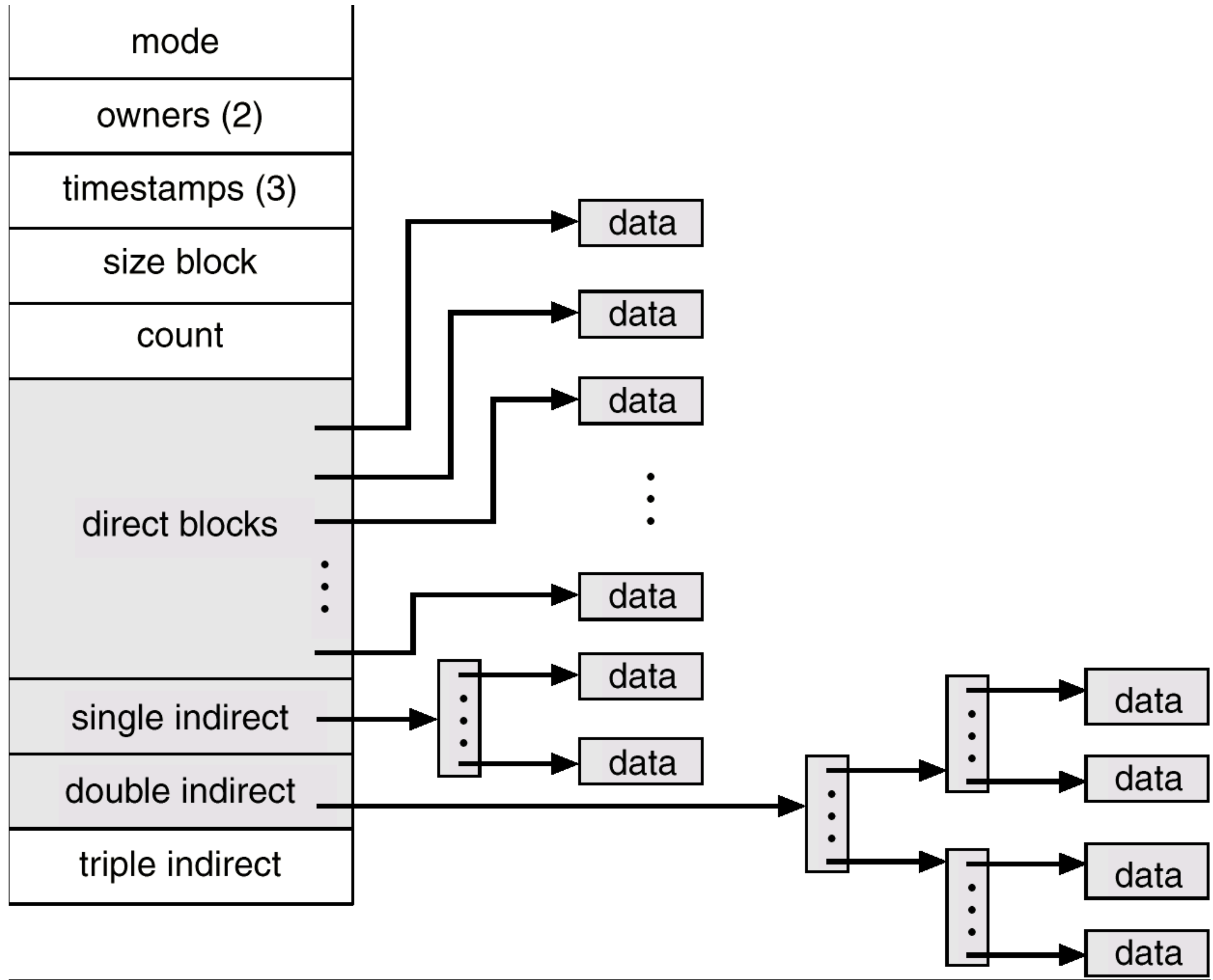


Structure of a regular file

- Each inode contains a table of contents for the disk address of data in a file.
- Some unix systems contain 10 directs block numbers, 1 indirect block number, 1 double indirect block number and 1 triples indirect block number.

Indexing inside the i-node





mode
owners (2)
timestamps (3)
size block
count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data

- **Direct block numbers**

  These contain block numbers that contain the file's data. Having these gives us direct access to the file's data.
- **Indirect block number**

  This is a block number of a block that contains a list of direct block numbers. Each block number is the number of a block that contains the file's data.
- **Double indirect block number**

  This refers to a block that contains a list of indirect block numbers. Each indirect block number is the number of a block that contains a list of direct block numbers.
- **Triple indirect block number**

  This refers to a block that contains a list of double indirect block numbers . Each double indirect block number is the number of a block that contains a list of indirect block numbers. Each of these contains a list of direct block number.

- Assume size of each block is 1K(1024) bytes and that a block number is addressable by a 32 bit integer(4-bytes) ,
- Then a block can hold up to 256 block numbers (1024bytes / 4bytes)
- Inode contains

  10 direct blocks, 1 indirect, 1 double-indirect, 1 triple indirect
- Capacity =

Direct blocks will address: 1K × 10 blocks = 10,240 bytes

1 Indirect block: additional 256 X 1K = 256K bytes

1 Double indirect block: additional 256 X 256 K =64M bytes

1 Triple indirect block: additional 256 X 64 MB= 16G

- Maximum file size = 10,240 + 256K + 64M + 16G =
- = 17247250432 bytes ≈ **16G bytes**

Example

Block size = 1024 bytes, 10 direct,
indirect/double/triple are 1

How does a process access byte 3073 of a file?

10 direct blocks addresses 10240 bytes of file data.

Since 3073<10240, hence byte 3073 lies in one of the direct block numbers.

3073/1024=3.0009

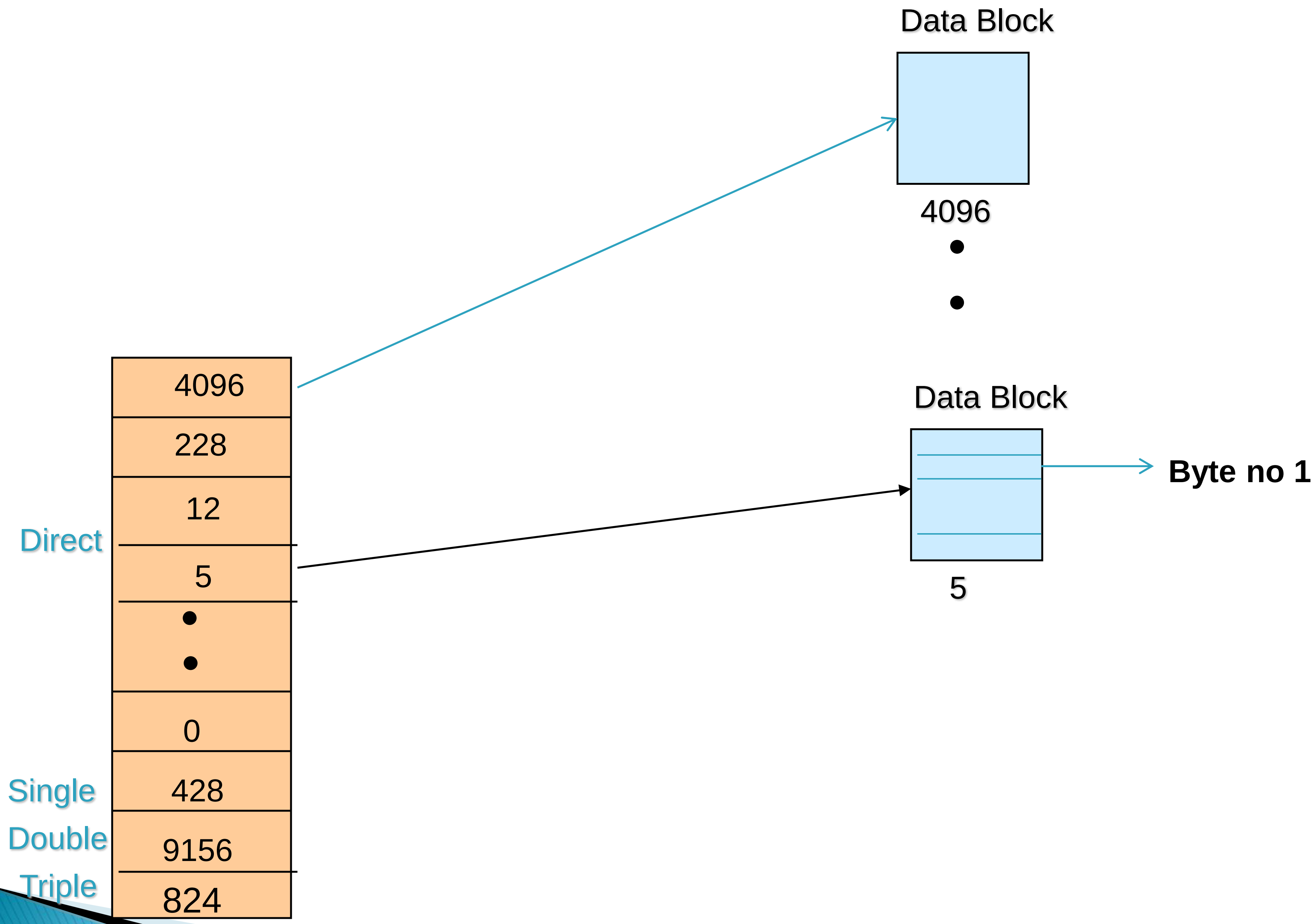
Hence 3073 lies in block pointed to by direct entry 3.

Where does the byte lie in that data block(at what offset)?

1024*3=3072

(3073-3072)=1

Hence the byte lies in block pointed to by direct entry 3 and in that data block at offset 1.



Data Block
4096
Data Block
Byte no 1
5
4096
228
12
Direct
5
0
Single
428
Double
9156
Triple
824

Example

Block size = 1024 bytes, 10 direct, indirect/double/triple are 1

How does a process access byte 11265 of a file?

10 direct blocks addresses 10240 bytes of file data.

Since 11265>10240, hence byte 11265 lies within single indirect block number.

11265-10240=1025(within single indirect it is byte no 1025)

Which entry within single indirect block no?

1025/1024=1.0009

Entry numbered 1(Assuming it starts from 0)

In this entry there is a data block no.

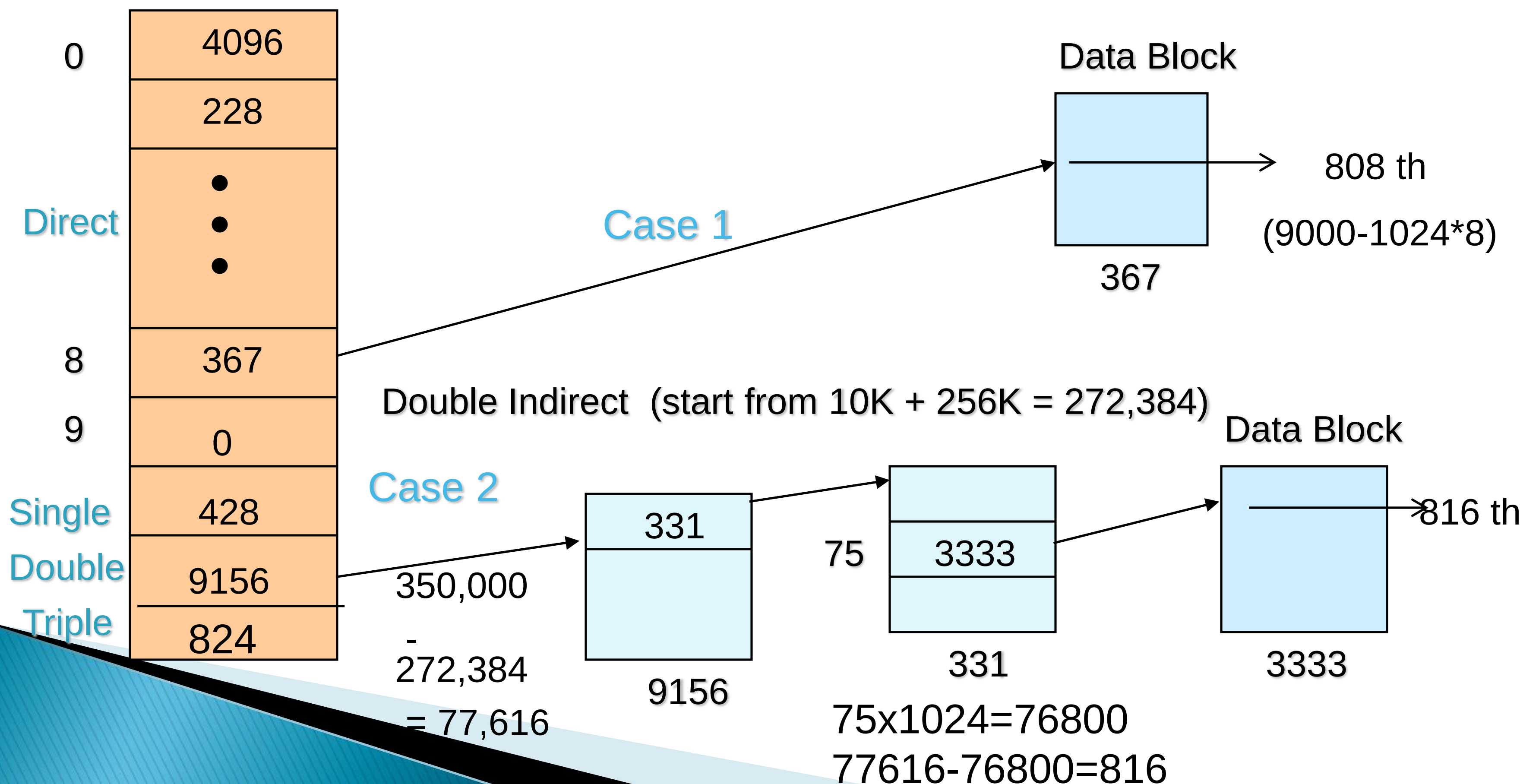
Which offset within this block?

1025-(1x1024)=1 {byte offset 1}

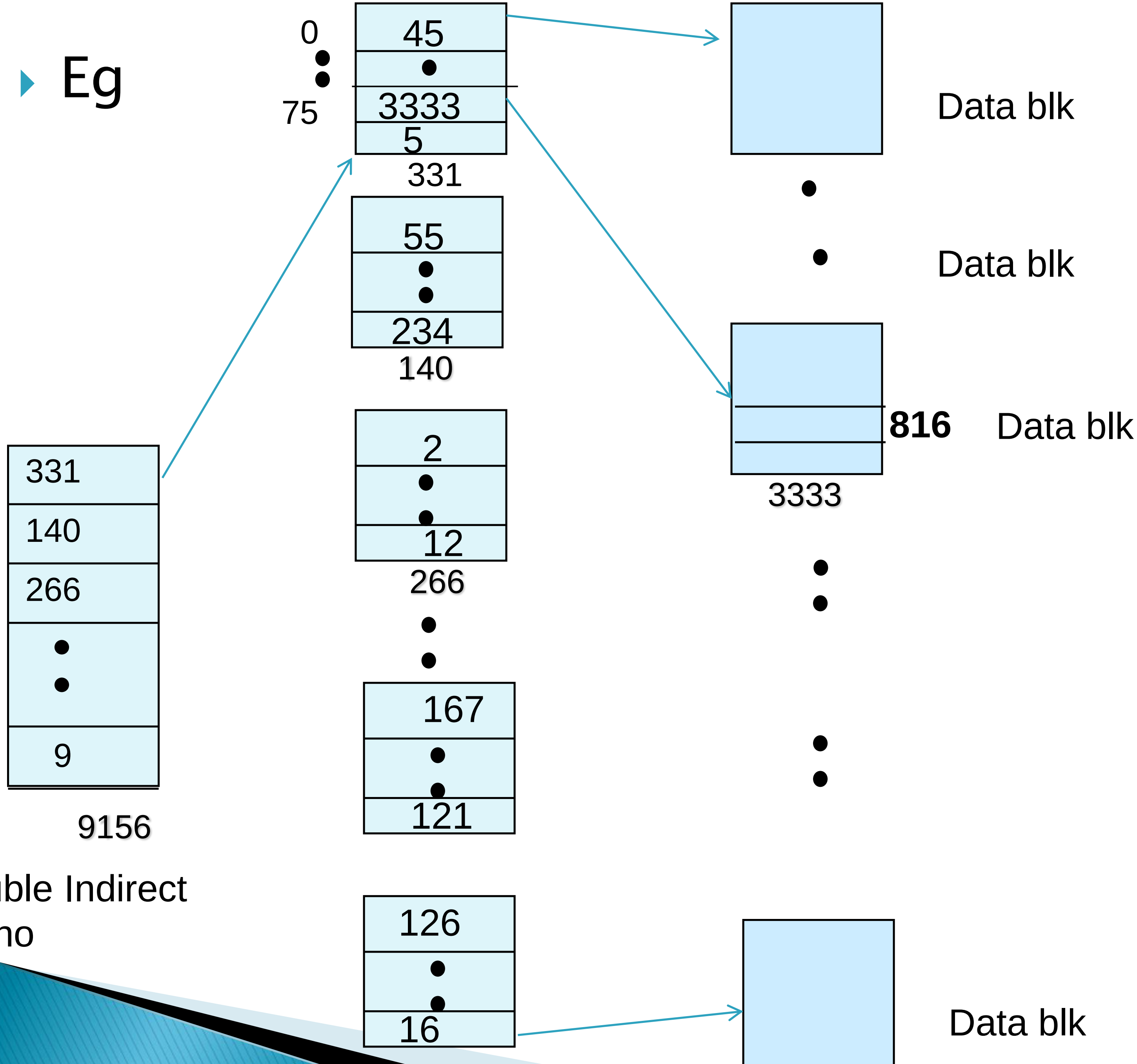
Example

- Block size = 1024 bytes, 10 direct, indirect/double/triple are 1
- Case 1 : Access byte offset 9000 of a file.
- Case 2 : Access byte offset 350,000 of a file.

Example – Accessing the Block



- Eg



Double Indirect blk no

Example – Accessing the Block

- If a process wants to access byte offset 350,000 in the file, it must access a double indirect block, number 9156 in the figure.
- Since an indirect block has room for 256 block numbers, the first byte accessed via the double indirect block is byte number 272,384 (256K + 10K)
- Byte number 350,000 in a file is therefore byte number 77,616 of the double indirect block.
- Since each single indirect block accesses 256K bytes, byte number 350,000 must be in the 0th single indirect block of the double indirect block — block number 331.

Example – Accessing the Block

- Since each direct block in a single indirect block contains IK bytes, byte number 77,616 of a single indirect block is in the 75th direct block in the single indirect block — block number 3333.
- Finally, byte number 350,000 in the file is at byte number 816 in block 3333.