innovate achieve lead

BITS Pilani
Pilani Campus

UNIX File Systems

Vaibhav Soni
Assistant Professor
Department of Computer Science and Information Systems

Points to be Covered

- Unix File Systems
- Types of Files
- Hierarchical Organization of Files
- Inodes
- Inode Table
- Incore Copy of Inode
- Unix File Storage and Access Methods
  - Direct Access Technique
  - Indirect Access Technique

File Systems

Unix File Systems	A logical technique of **organizing and storing** large amounts of information in a way that makes it easy to manage.
	File is a smallest unit in which the information is stored.
	Everything is represented as a file in Unix.

Classification of Files in Unix

Regular File	• Usually contains data, text, or program instructions.
Device Files	• Used to represent any real physical devices such as printer, CD-ROMs, etc.
Directory Files	• Contains both the regular and device files. • Each Entry consists of File Name and inode number.

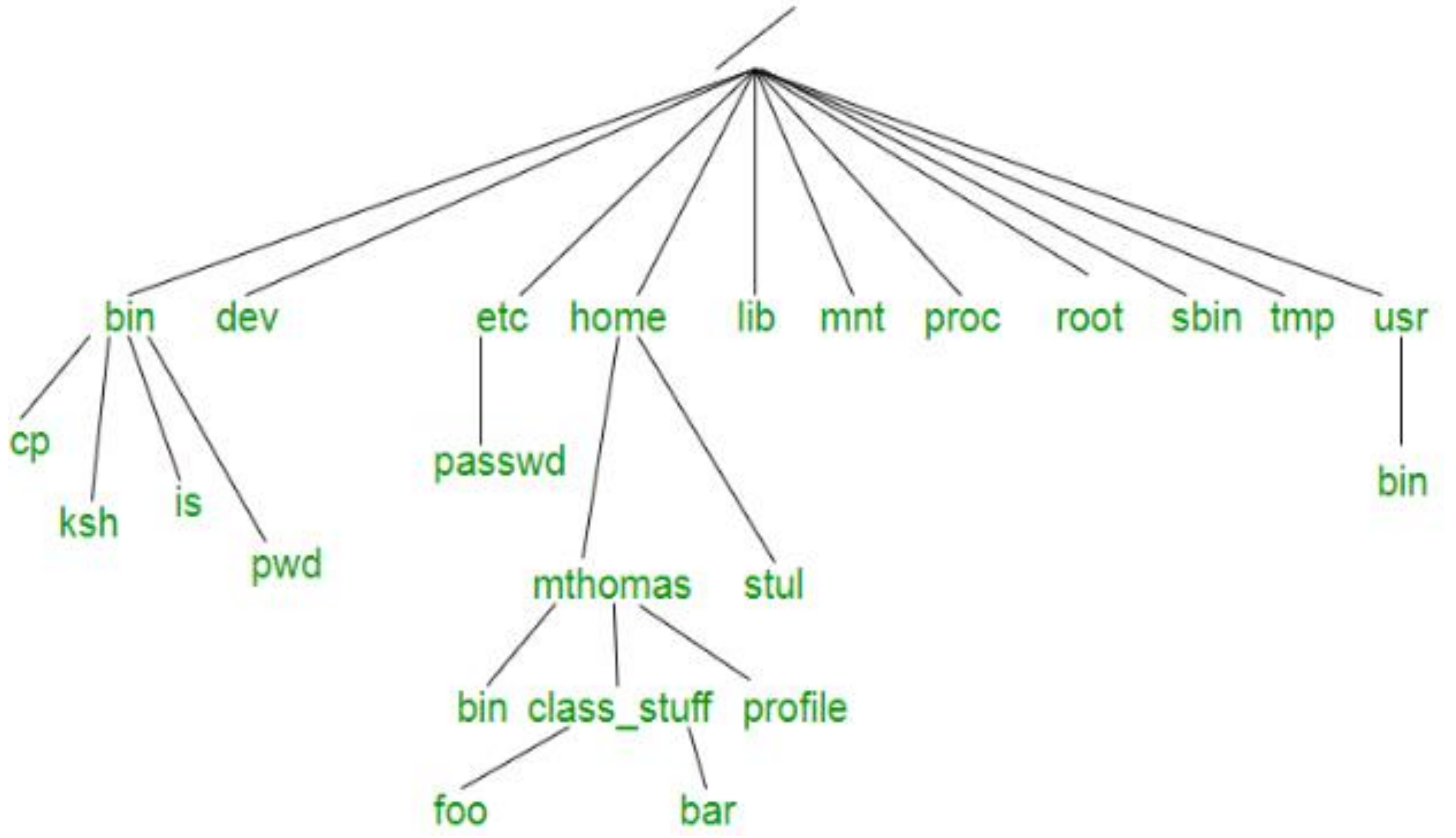
Hierarchical Organization of Files

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Hierarchical File Structure:

- All of the files in the UNIX file system are organized into a multi-leveled hierarchy called a directory tree.
- A family tree is an example of a hierarchical structure that represents how the UNIX file system is organized. The UNIX file system might also be envisioned as an inverted tree or the root system of plant.
- At the very top of the file system is single directory called "root" which is represented by a / (slash). All other files are "descendants" of root.
- The number of levels is largely arbitrary, although most UNIX systems share some organizational similarities. The "standard" UNIX file system is discussed later.

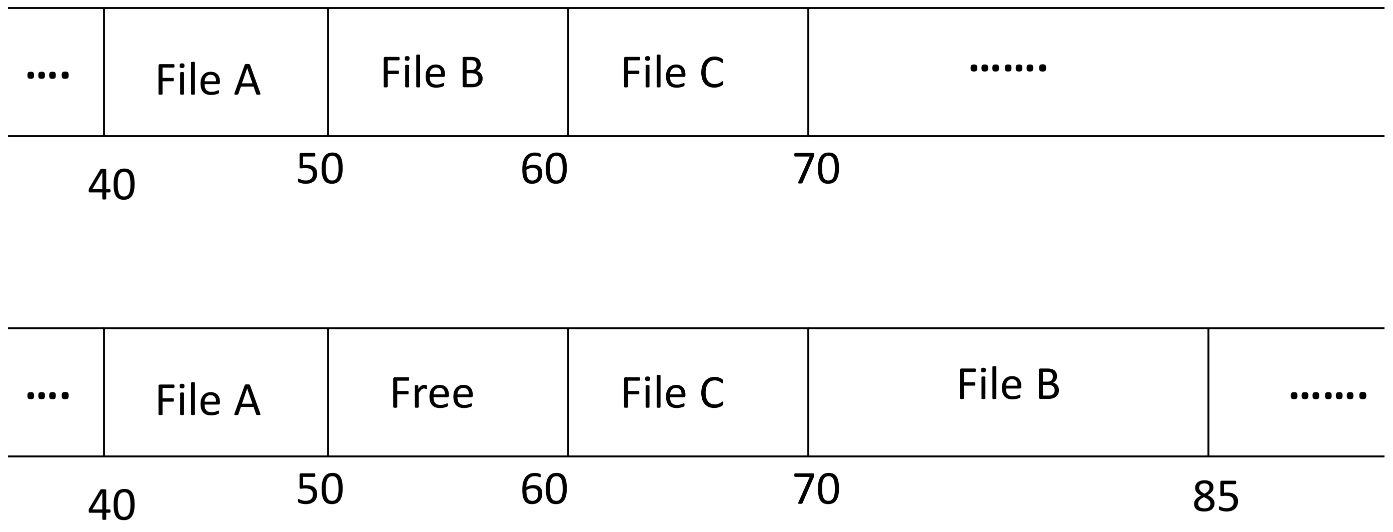
Hierarchical Organization of Files



Structure of a Regular File

- Table of contents in an inode
  - Location of a file's data on disk
  - A set of disk block #
    - Each block on a disk is addressable by number
  - Not contiguous file allocation strategy
    - Why ?
    - When a file expand or contract…
    - Fragmentations occur

Sample - Fragmentation



- File B was expanded
- Garbage collection – too high cost

Inodes

Index nodes usually stores the information (meta data) related to the UNIX files.

Information: File-size, its location, time of last access, time of last modification, permissions, etc.

It also contains pointers to the data blocks of the file.

Inode Table

Each file is represented by an inode.

Each inode entry is of 64 Bytes.

Contains only file metadata.

Inode Table (Continued…..)

- Type of file
- File Access Permissions (r w x)
- Owner of file (uid)
- Group to which owner belongs (gid)
- Date and Time of Last Access (atime)
- Date and Time of Creation (ctime)
- Date and Time of Last Modified (mtime)
- File Size (size)
- No. of blocks used by file (block count)
- Addresses of blocks

Sample disk inode

owner mjb

group os

type regular file

perms rwxr-xr-x

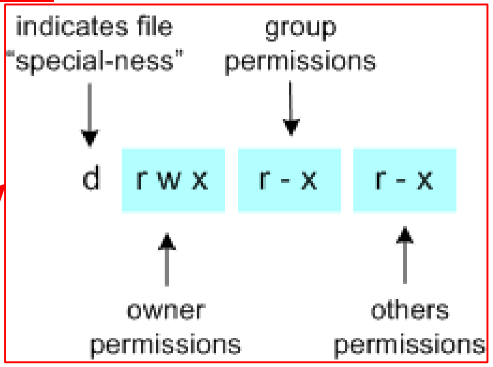
accessed Oct 23 1984 1:45 P.M

modified Oct 22 1984 10:30 A.M

created Oct 23 1984 1:30 P.M

size 6030 bytes

Disk addresses

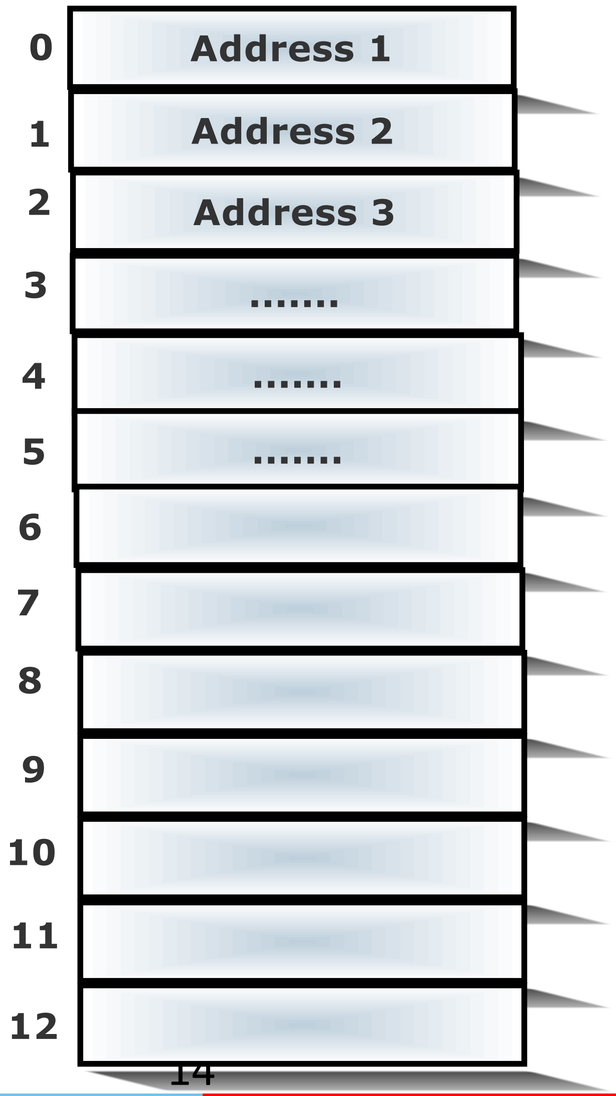


In-core copy of Inode

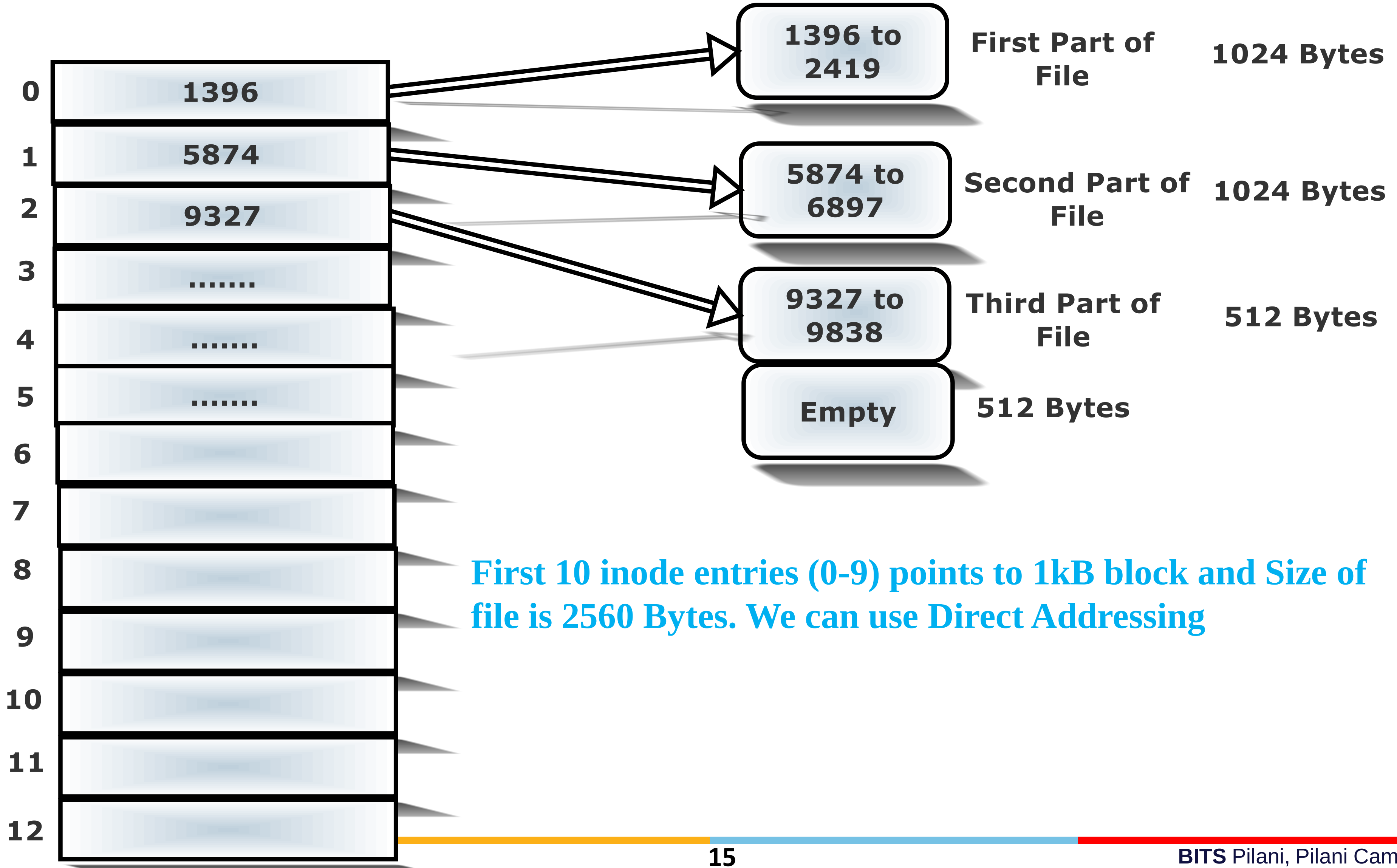
- in-core copy of the inode contains
  - status of the in-core inode
    - Inode is locked
    - Process is waiting for the inode to become unlocked
    - In-core representation of the inode differs from the disk copy as a result of a change to the data in the inode
    - In-core representation of the inode differs from the disk copy as a result of a change to the file data
    - File is a mount point
  - logical device number of file system
  - inode number (stored as a linear array on disk)
  - pointers to other in-core inodes
  - reference count (no. of instances of the file are active)

Unix File Storage And Access

- Each Inode entry contains the addresses of the blocks of memory.
- Each entry of Inode entry points to 1KB block of memory.
- Total 13 entries => at most 13KB



Unix File Storage And Access (Direct Addressing)

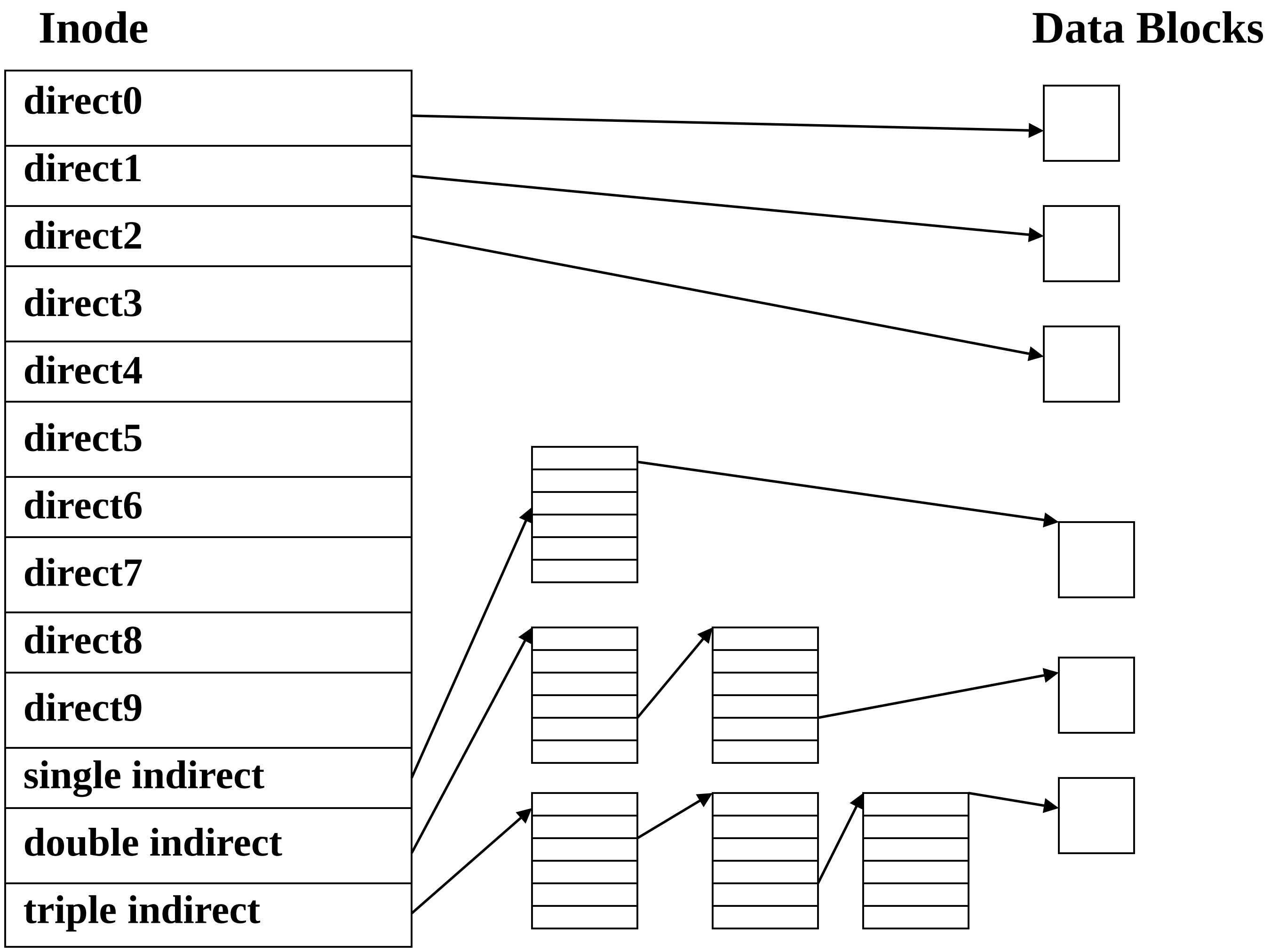


First 10 inode entries (0-9) points to 1kB block and Size of file is 2560 Bytes. We can use Direct Addressing

Single Indirection

$$\begin{tikzcd}
\text{0-9 entries} \arrow[r] & \text{Single Address} \arrow[r] & \text{File Content} \\
\text{10th entry} \arrow[r] & \text{Single Address} \arrow[r] & \text{256 Addresses} \arrow[d] \\
 & & \text{File Content}
\end{tikzcd}$$

Direct and indirect blocks in inode



Assume that a logical block on the file system holds 1K bytes and that a block number is addressable by a 32 bit integer, then a block can hold up to 256 block numbers

- 13 entries in the inode table of contents
  - 10 direct, 1 indirect, 1 double indirect, 1 triple indirect block
- Assume
  - a logical block = 1KBytes
  - a block number is addressable by a 32 bit (4 bytes) integer
  - a block can hold up to 256 block numbers
- Byte Capacity of a File

10 direct blocks with 1K bytes each=	10K bytes
1 indirect block with 256 direct blocks= 1K*256=	256K bytes
1 double indirect block with 256 indirect blocks= 256K*256=	64M bytes
1 triple indirect block with 256 double indirect blocks= 64M*256=	16G bytes

Example

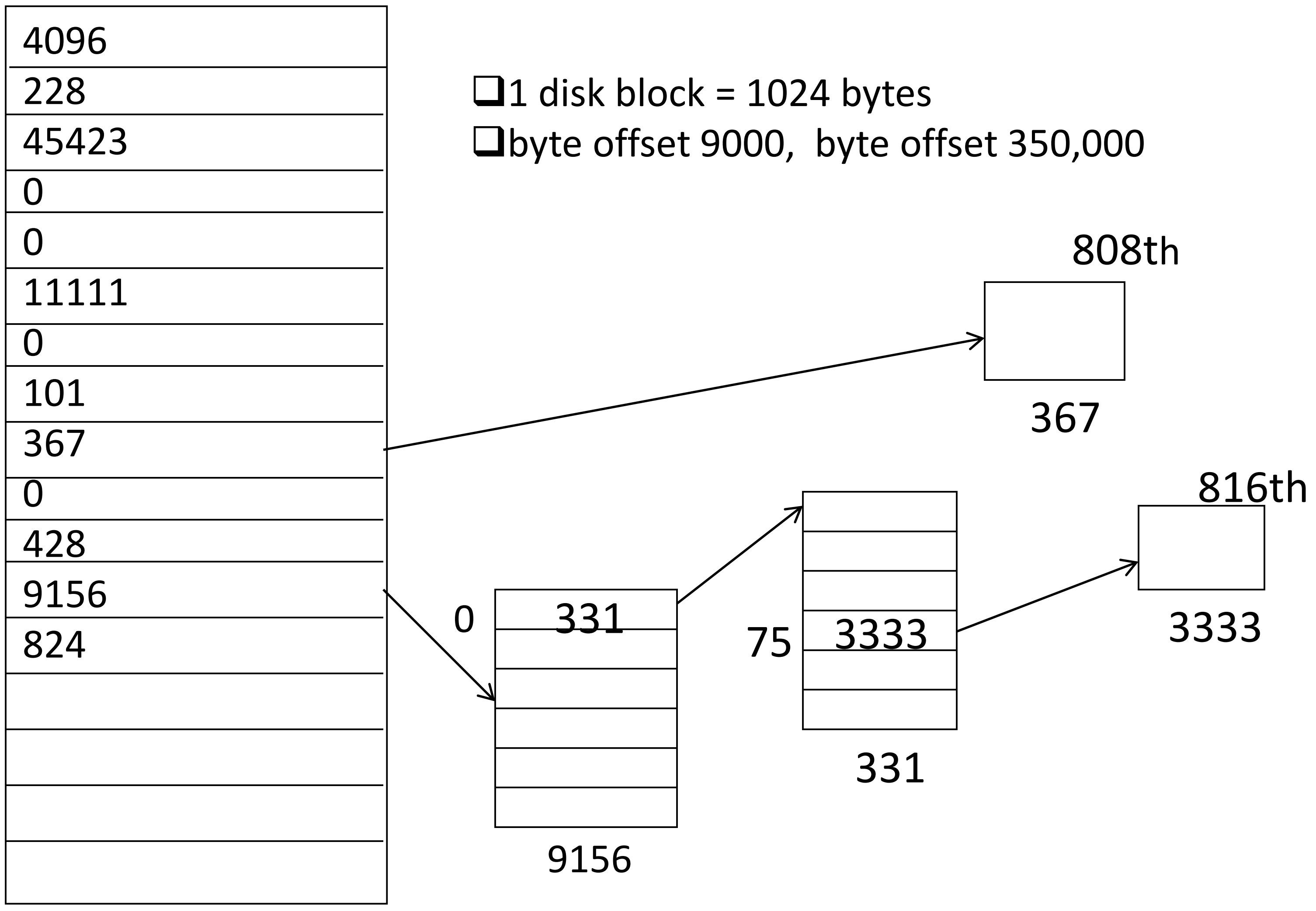
- Disk block size = 1024 Bytes or 1KB
- 4 bytes are required to store one block number
- 256 block numbers can be stored in one index block (1024/4)

Problem 1:

- If a process makes a request to access 9000 byte offset, then find out which index block contains this byte information.
- 9000 / 1024 = 8 , 9000 mod 1024 = 808
- That means 9000 – 808 = 8192 byte offset is last offset in 8th index block
- Since 808 < 1024 , therefore byte offset 9000 is surely present in 9th index block with block no. 367 and in 808th byte

Block Layout of a Sample File and its Inode

- 1 disk block = 1024 bytes
- byte offset 9000, byte offset 350,000

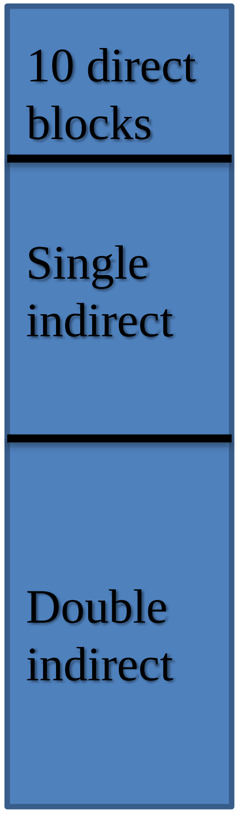


Problem 2: If a process makes a request to access 350,000 byte offset, then find out which index block contains this byte information.

- First 10 direct blocks can access 10240 bytes or 10K bytes.
- 350,000 > 10240, therefore lets look into single indirect block.
- Single indirect can address 256 blocks, each block contains 1024 bytes, so 256 * 1024= 262144 bytes can be accessed = 256KB
- So 10 K + 256 K = 266 K = 272,384 bytes
- Still 350,000 > 272,384 , Go ahead and search in double indirect.
- 350,000 – 272,384 = 77,616 byte no in double indirect.
- First entry in double indirect block points to first single indirect block which holds 256 block nos.., each block contains 1024 bytes, and hence points to 256K byte addresses i.e 262,144 byte address.
- Since 77,616 < 262,144, 350,000 will be there in one of the blocks pointed to by single indirect.
- 77616/1024 = 75, 77616 mod 1024 = 816
- It comes out to be 75th block and 816th byte entry.

Example2

If a process makes a request to access 350,000 byte offset, then find out which index block contains this byte information.



10 DIRECT BLOCKS *(10240 bytes or 10K bytes.)*

350,000 > 10240, therefore lets look into single indirect block

SINGLE INDIRECT can address 256 blocks,

each block contains **1024 bytes**, so *256 * 1024= 262144 bytes* can be ***= 256KB***

So 10 K + 256 K = 266 K **= 272,384 bytes**

350,000 > 272,384 , Go ahead and search in double indirect.

DOUBLE INDIRECT BLOCK contain two single indirect block

Take first single indirect block

*256 * 1024= 262144 bytes* can be ***= 256KB***

Since **77616 < 262144**

77616/1024 = 75, 77616 mod 1024 = 816

It comes out to be 75th block and 816th byte entry.