CS & IT ENGINEERING

Operating System

File System (Part - 02)

[redacted]

Revision

Lecture No.- 14

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic File System Interface

Topics to be Covered

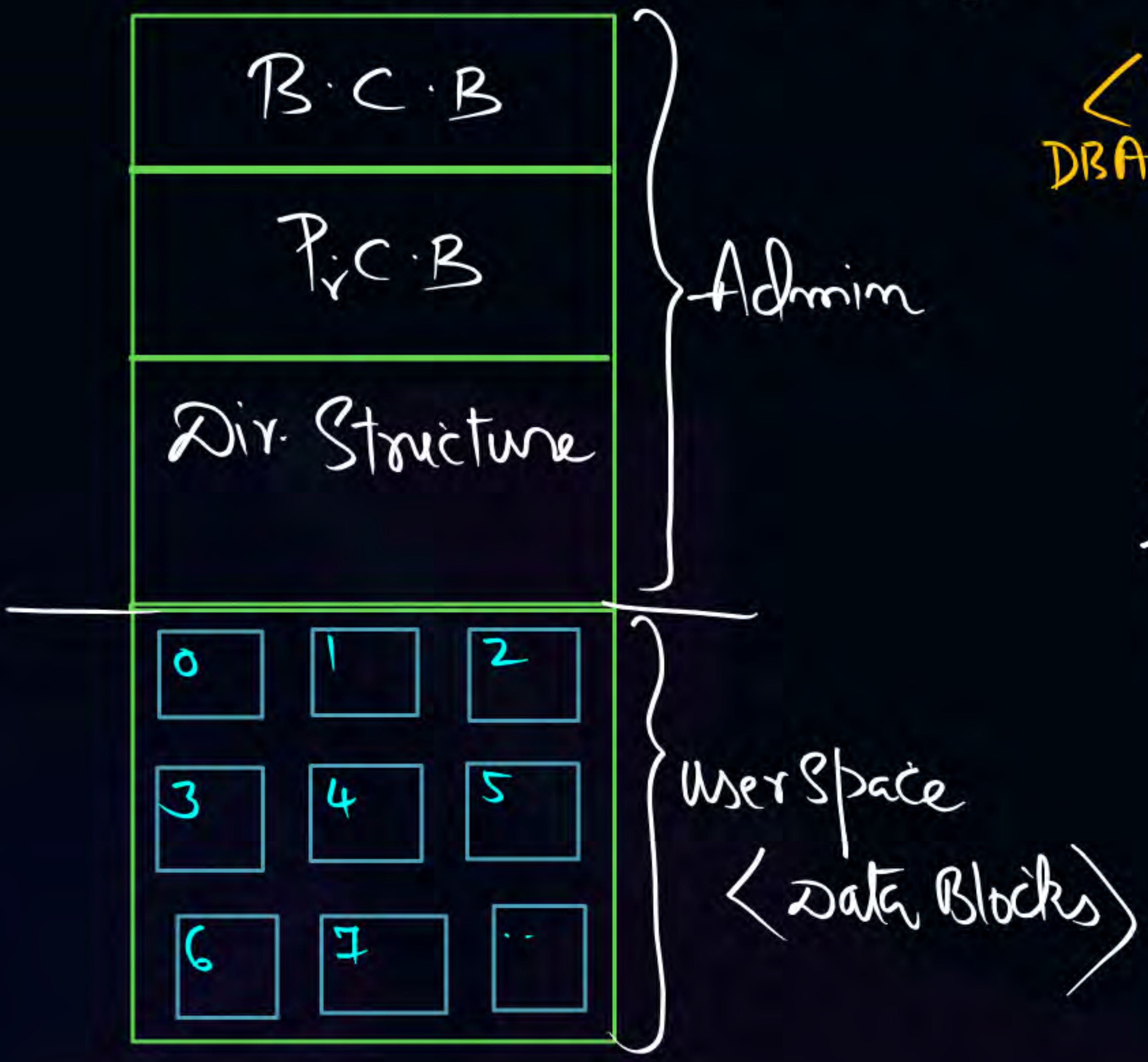
- Topic: Allocation Methods
- Topic: Free Space Management
- Topic: Case Studies ⟨UNIX/LINUX ; DOS/WIN⟩
- Topic: Problem Solving

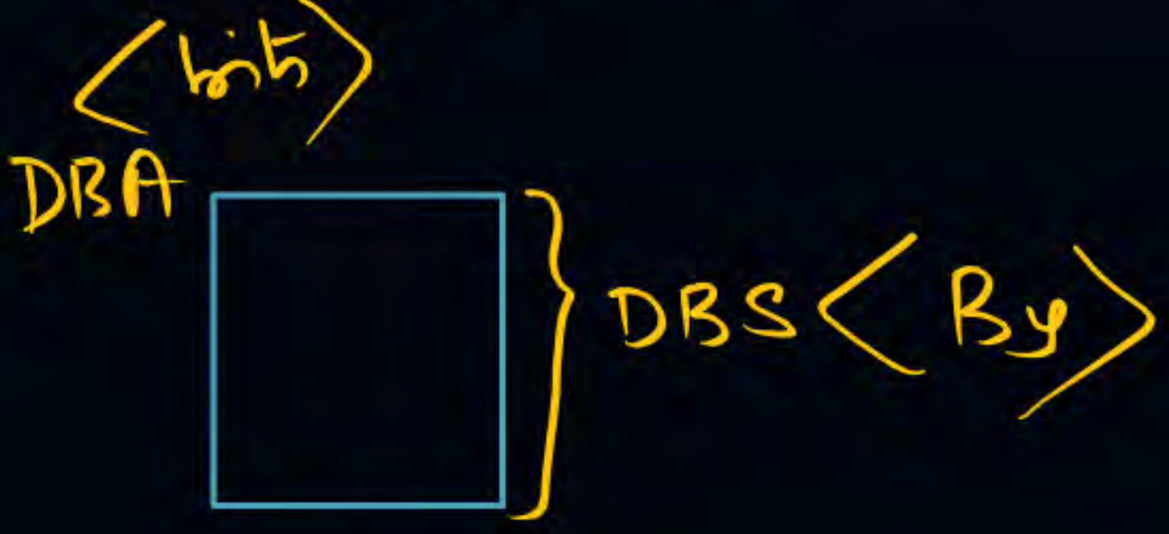
Disk Scheduling

Topic : File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in blocks of sectors (usually 512 bytes)
- File control block (FCB) – storage structure consisting of information about a file
- Device driver controls the physical device
- File system organized into layers

Partition Structure <Formatting Process>





DB

<Data/Disk Block>

Max. Possible Disk Size $\left(2^{DBA}\right) * DBS$

Max. File Size

Topic : Layered File System

application programs

↓

logical file system — Knows about Files & their FCB's / Dir. Stru

↓

file-organization module — Converts L.A's to P.A's (free Sp. Manager)

↓

basic file system — Blocks & their Addresses (I O)

↓

I/O control — Dev. Drivers & Interrupt Handlers

↓

devices — Aux/SEC STORAGE

Topic : File System Layers

- Device drivers manage I/O devices at the I/O control layer
- Given commands like
- read drive1, cylinder 72, track 2, sector 10, into memory location 1060
- Outputs low-level hardware specific commands to hardware controller
- Basic file system given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- File organization module understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

Topic : File System Layers (Cont.)

- Logical file system manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Logical layers can be implemented by any coding method according to OS designer

Topic : File System Layers (Cont.)

- 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

Topic : File-System Operations

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- Boot control block contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- Volume control block (superblock, master file table) contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

Topic : File Control Block (FCB)

- OS maintains FCB per file, which contains many details about the file
  - Typically, inode number, permissions, size, dates
  - Example

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Topic : In-Memory File System Structures

- Mount table storing file system mounts, mount points, file system types
- System-wide open-file table contains a copy of the FCB of each file and other info
- Per-process open-file table contains pointers to appropriate entries in system-wide open-file table as well as other info

Topic : Directory Implementation

- Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- Hash Table – linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Topic : Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous (CG)
  - Linked (NCG)
  - File Allocation Table (FAT)

Indexed

File : 16 KB

DBS : 512 B

# of Blks : $\frac{16\,KB}{512\,B} = \frac{2^{14}}{2^{9}} = 2^{5} = 32$

Topic : Contiguous Allocation Method

- An allocation method refers to how disk blocks are allocated for files:
- Each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include:
    - Finding space on the disk for a file,
    - Knowing file size,
    - External fragmentation, need for compaction off-line (downtime) or on-line

Topic : Contiguous Allocation (Cont.)

- Mapping from logical to physical
- (block size =512 bytes)

LA/512 → Q, R

- Block to be accessed = starting address + Q
- Displacement into block = R

First DBA

directory

# of Blks

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

Problems :

1) I.F : ✓
2) E.F : ✓
3) Inc. F. Size : Inflexible
4) Type of Access : (S+R) Faster Access

Topic : Extent-Based Systems

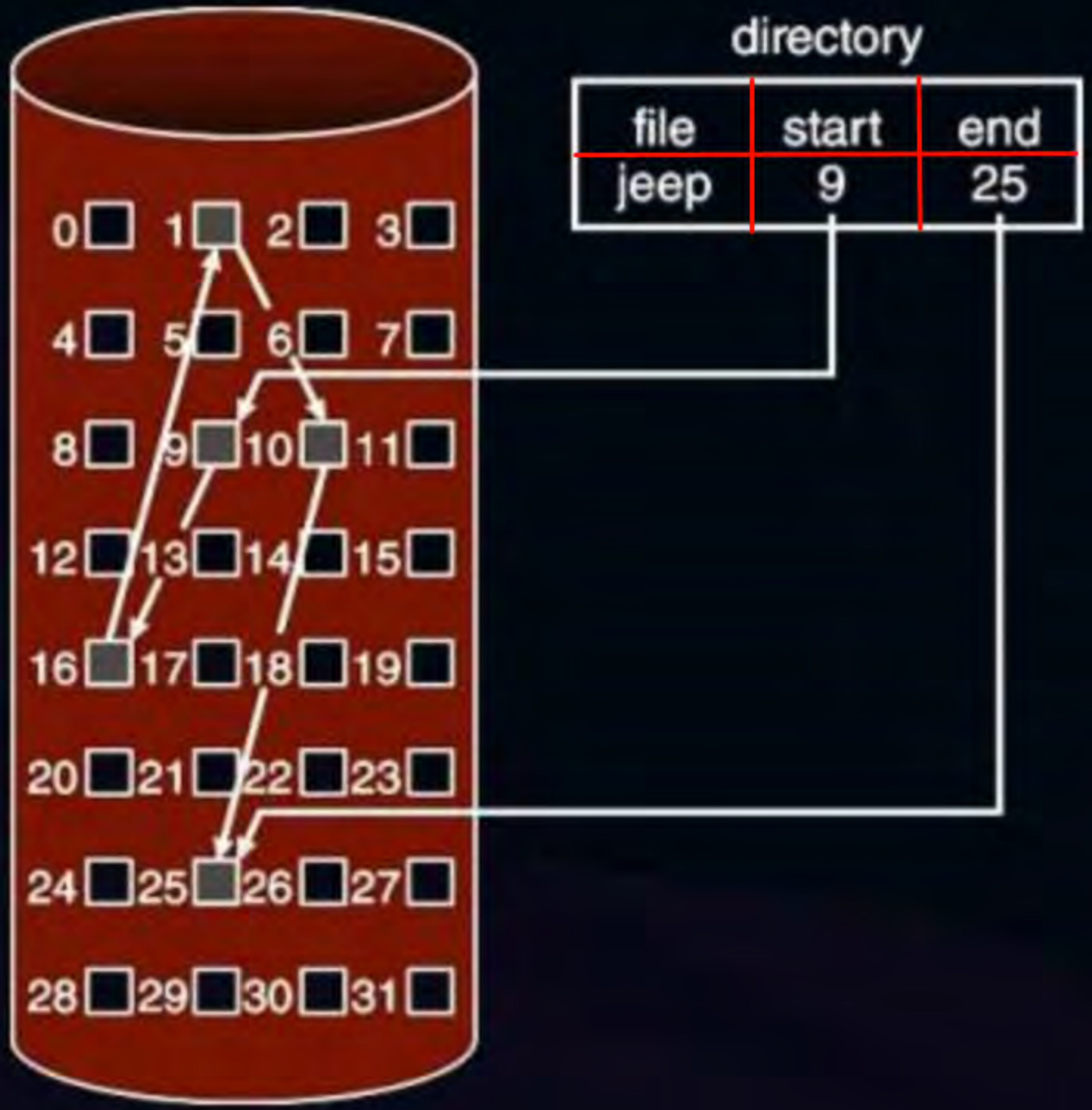
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Topic : Linked Allocation

- Each file is a linked list of blocks
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem ✓ Link can break
- Locating a block can take many I/Os and disk seeks

Topic : Linked Allocation Example

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- Scheme



Topic : Linked Allocation (Cont.)

- Mapping

$$LA/511 \begin{cases} Q \\ R \end{cases}$$

- Block to be accessed is the Q^{th} block in the linked chain of blocks representing the file.
- Displacement into block = R + 1

Topic : FAT Allocation Method

Case - Study

- Beginning of volume has table, indexed by block number
- Much like a linked list, but faster on disk and cacheable
- New block allocation simple

Tabular linked Allocation

DOS/WIN

F.A.T / M.F.T

FAT - 32

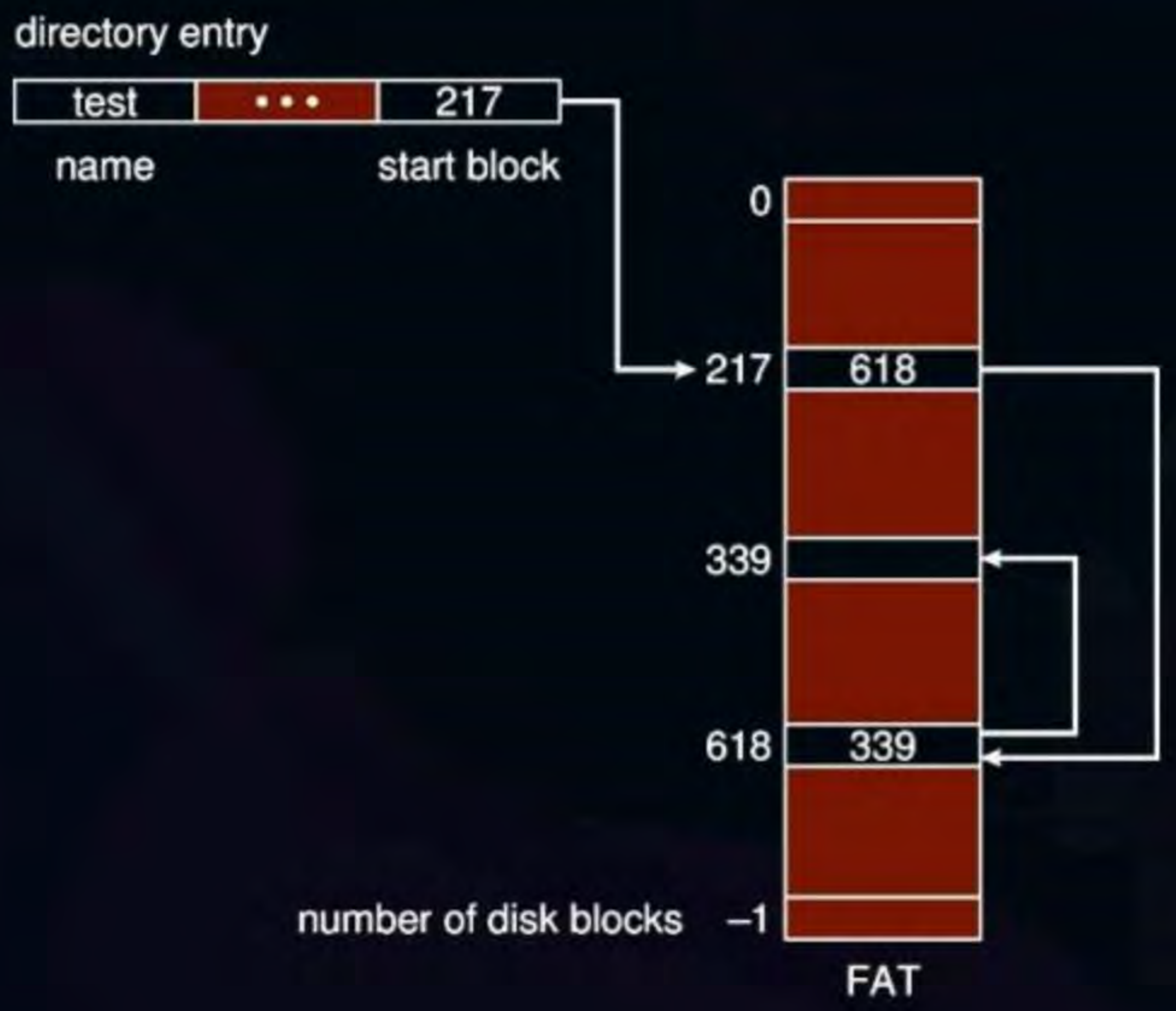
"DBA"

FAT - entries

No. of entries = No. of Blocks

FAT entry Contains "DBA"

Topic : File-Allocation Table

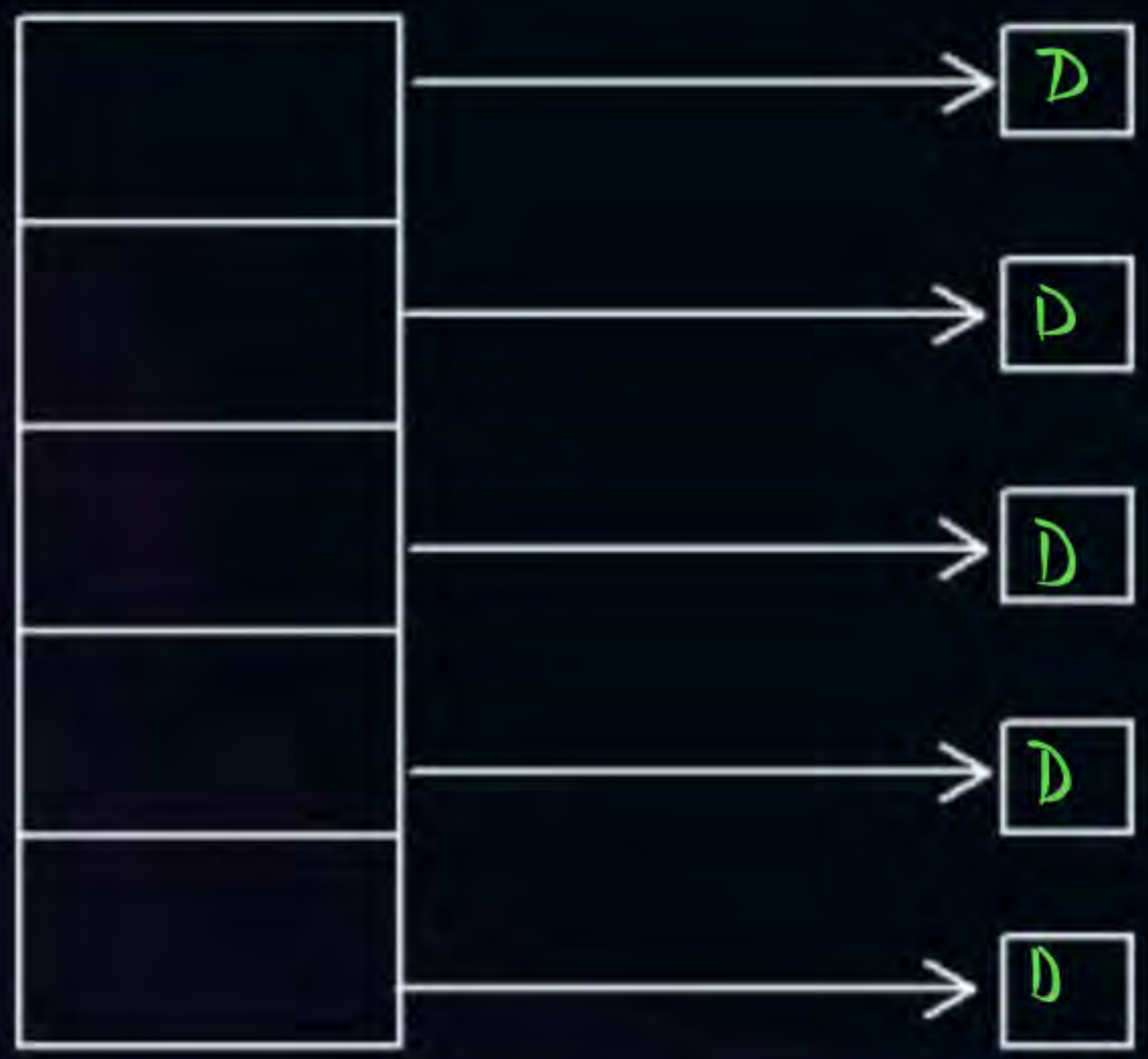


Topic : Indexed Allocation Method

<CG + NCG>

- Each file has its own index block(s) of pointers to its data blocks
- Logical view

Index Block



index table

$2^{64} \times 4KB$

DBS: 4KB

DBA: 64 bits = 8B

a) Max. possible F. Size

b) Max File Size with one Index Block 2MB

4KB

$\frac{4KB}{8B} = 512$

$512 \times 4KB = 2^9 \times 2^{12} = 2^{21}$

$DBS = 2KB$

$DBA = 32\ bits = 4B$

For a File of Size 32MB, How many Index Blks are Needed?

32 Index Blocks

2KB

DB

$512 \times 2KB$

$= 1MB$

Will there be any Int. Frag. within Index Block? – NO –

$$\frac{2KB}{4B} = 512$$

A Disk Block of Size 4KB Can hold 128 Addresses of Data Blks;

What is Max. File Size over the Disk?

Disk Size = $\left(2^{DBA} * DBS\right)$ By

$2^{256} * 4KB$

$\frac{\text{Size of DBA}}{\text{bits}} = \frac{4KB}{128}$

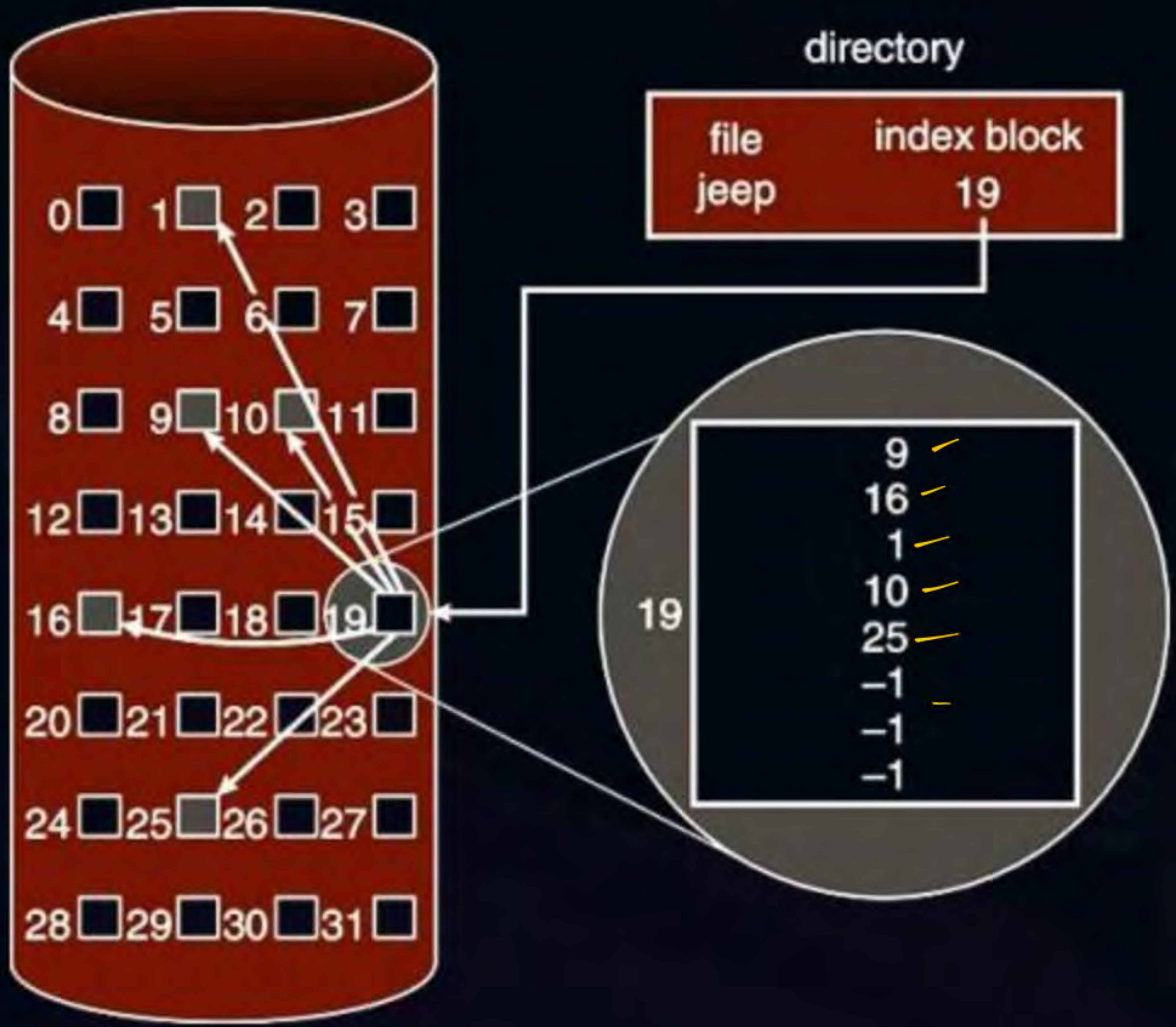
$= \frac{2^{12}}{2^7} = 2^5$

$= 32B$

$= 32 \times 8$ bits

$= 256$ bits

Topic : Example of Indexed Allocation



Topic : Indexed Allocation – Small Files

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table

Q

LA/512

R

- Calculation:
- Q = displacement into index table
- R = displacement into block

Topic : Indexed Allocation – Large Files

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
  - Linked scheme – Link blocks of index table (no limit on size)
  - Multi-level indexing

Topic : Indexed Allocation – Linked Scheme

- Link blocks of index table (no limit on size)

$$LA / (512 \times 511) \begin{cases} Q_1 \\ R_1 \end{cases}$$

- Outer-level mapping scheme
  - Q_1 = block of index table
  - R_1 is used as follows

$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

- Inner-level mapping scheme
  - Q_2 = displacement into block of index table
  - R_2 displacement into block of file

Topic : Indexed Allocation – Two-level Scheme

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)

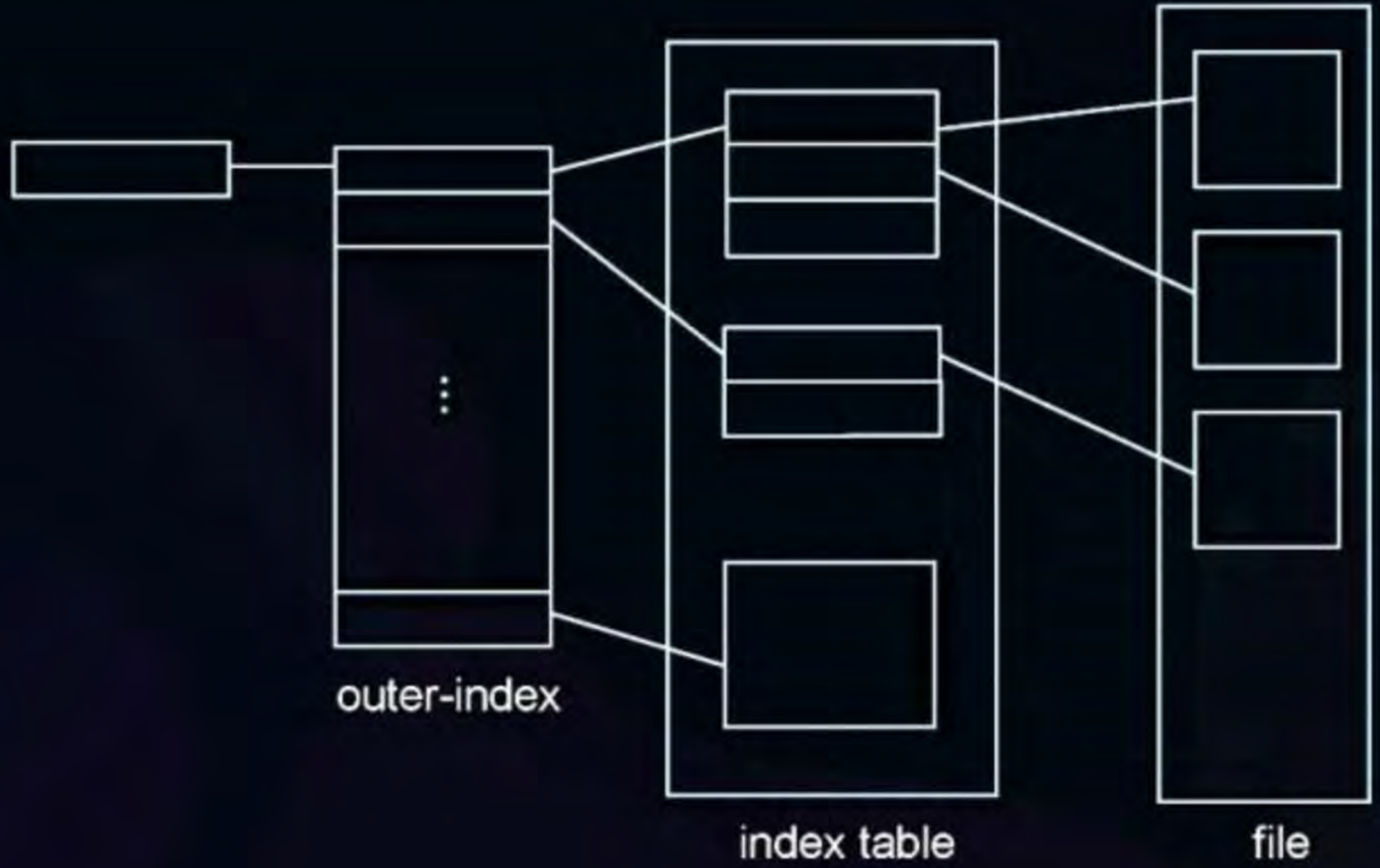
$$LA / (512 \times 512) \left\langle \begin{matrix} Q_1 \\ R_1 \end{matrix} \right.$$

- Mapping scheme for outer-index:
  - Q_1 = displacement into outer-index
  - R_1 is used as follows:

$$R_1 / 512 \left\langle \begin{matrix} Q_2 \\ R_2 \end{matrix} \right.$$

- Mapping scheme for index level:
  - Q_2 = displacement into block of index table
  - R_2 displacement into block of file

Topic : Indexed Allocation – Two-Level Scheme



Topic : Combined Scheme : UNIX UFS

I-Node Structure

Multi-Level Index

- 4K bytes per block, 32-bit addresses

Extended Index Allocation

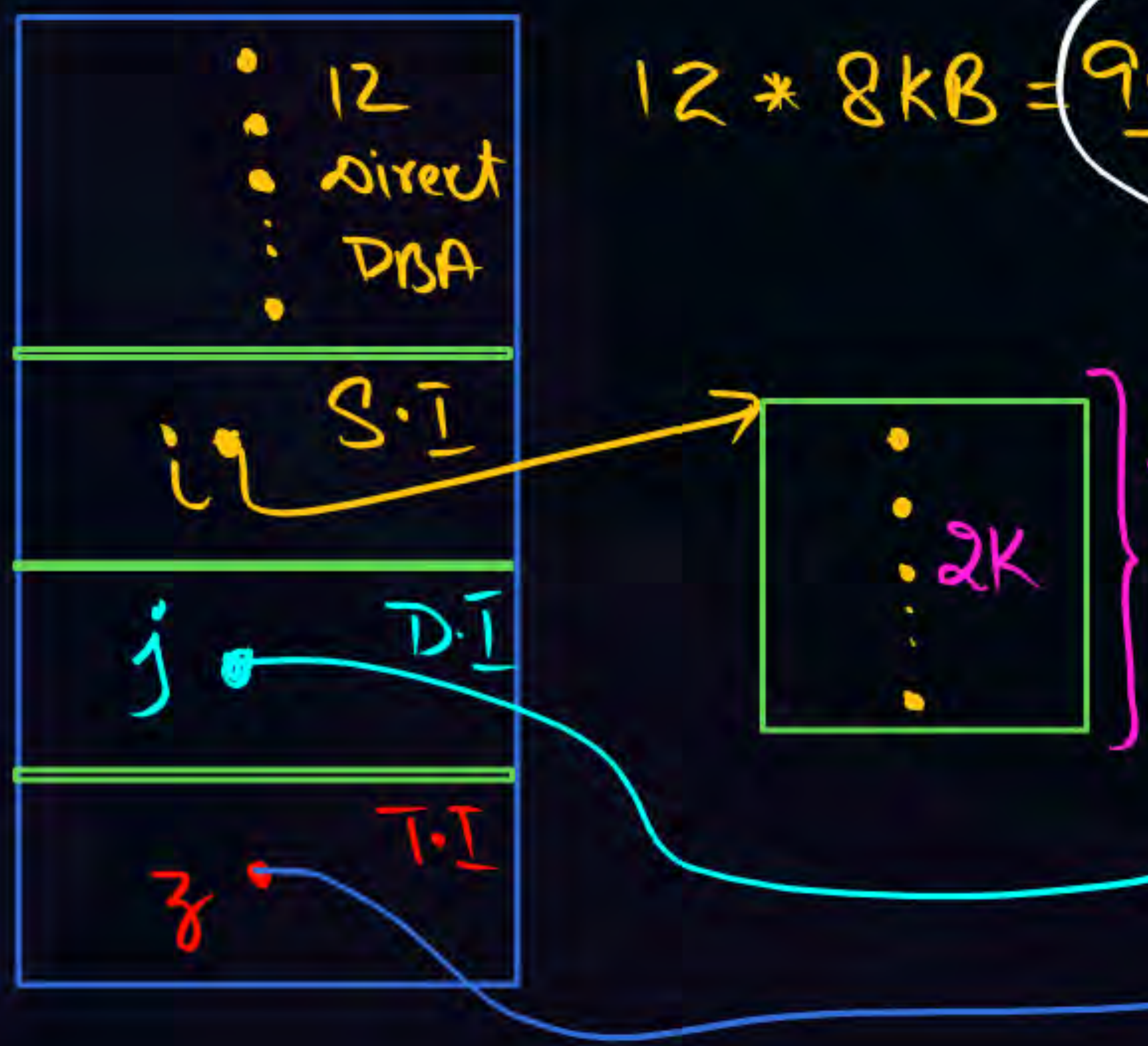
- More index blocks than can be addressed with 32-bit file pointer

Consider a file system that uses inodes to represent files. Disk blocks are 8 KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

$\frac{8KB}{4B} = 2K$

DBS = 8KB;
DBA = 4B;

$\sim \boxed{64.032\,TB}$



$12 * 8KB = 96KB$

$2K * 8KB = 16MB$

$= 0.032TB$

$2K * 2K * 8KB = 32GB$

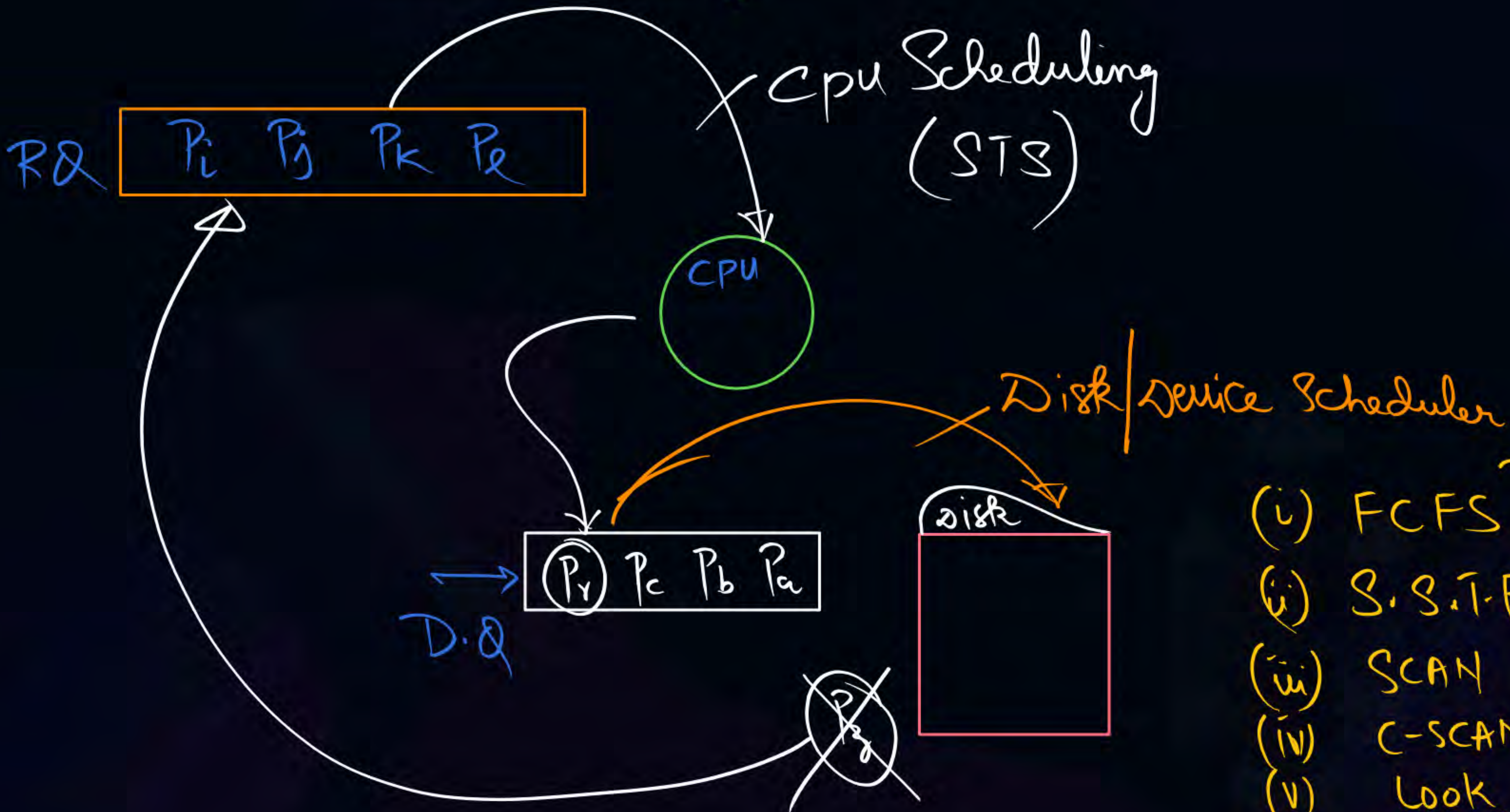
$2K * 2K * 2K * 8KB = 64TB$

Free Space Management

1) Free linked list
2) <u>Free list</u> :
3) $\dfrac{\text{Bit - Map}}{\text{Bit - vector}}$:
4) <u>Counter Method</u> :

< Majority of Blocks are Cn free >

Disk Scheduling



(i) FCFS
(ii) S.S.T.F
(iii) SCAN
(iv) C-SCAN
(v) LOOK
(vi) C-LOOK

Consider a file currently consisting of 100 blocks. Assume that the file-control block (and the index block, in the case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for contiguous, linked, and indexed (single-level) allocation strategies, if, for one block, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow at the beginning but there is room to grow at the end. Also assume that the block information to be added is stored in memory.

a. The block is added at the beginning.

b. The block is added in the middle.

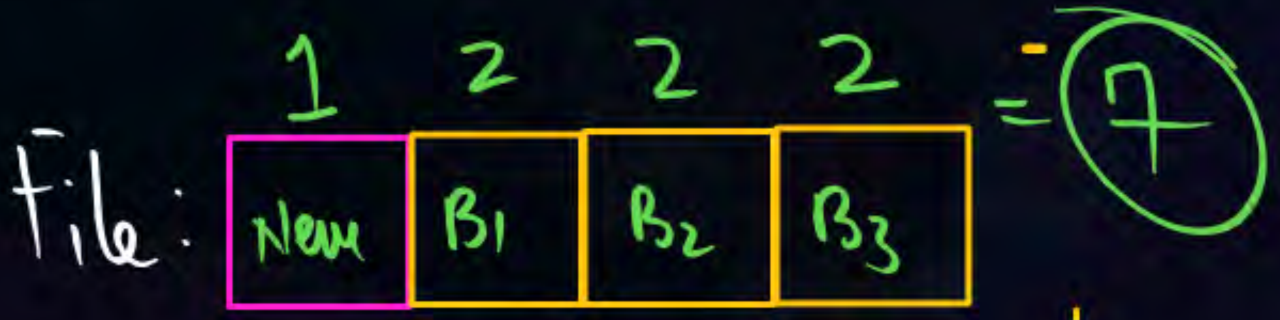
c. The block is added at the end.

d. The block is removed from the beginning.

e. The block is removed from the middle.

f. The block is removed from the end.

	CG	Linked	Indexed
a)	201	1	1



2 mins Summary

Topic One Allocation Methods

Topic Two Free Space Mgmnt

Topic Three Disk Scheduling

Topic Four Problem Solving

44

Topic Five

THANK - YOU