File System Implementation

File-System Implementation

The file system resides permanently on secondary storage, which is designed to hold a large amount of data permanently.

Here, we will study about how the file systems are implemented.

Different operating systems supports different file systems:

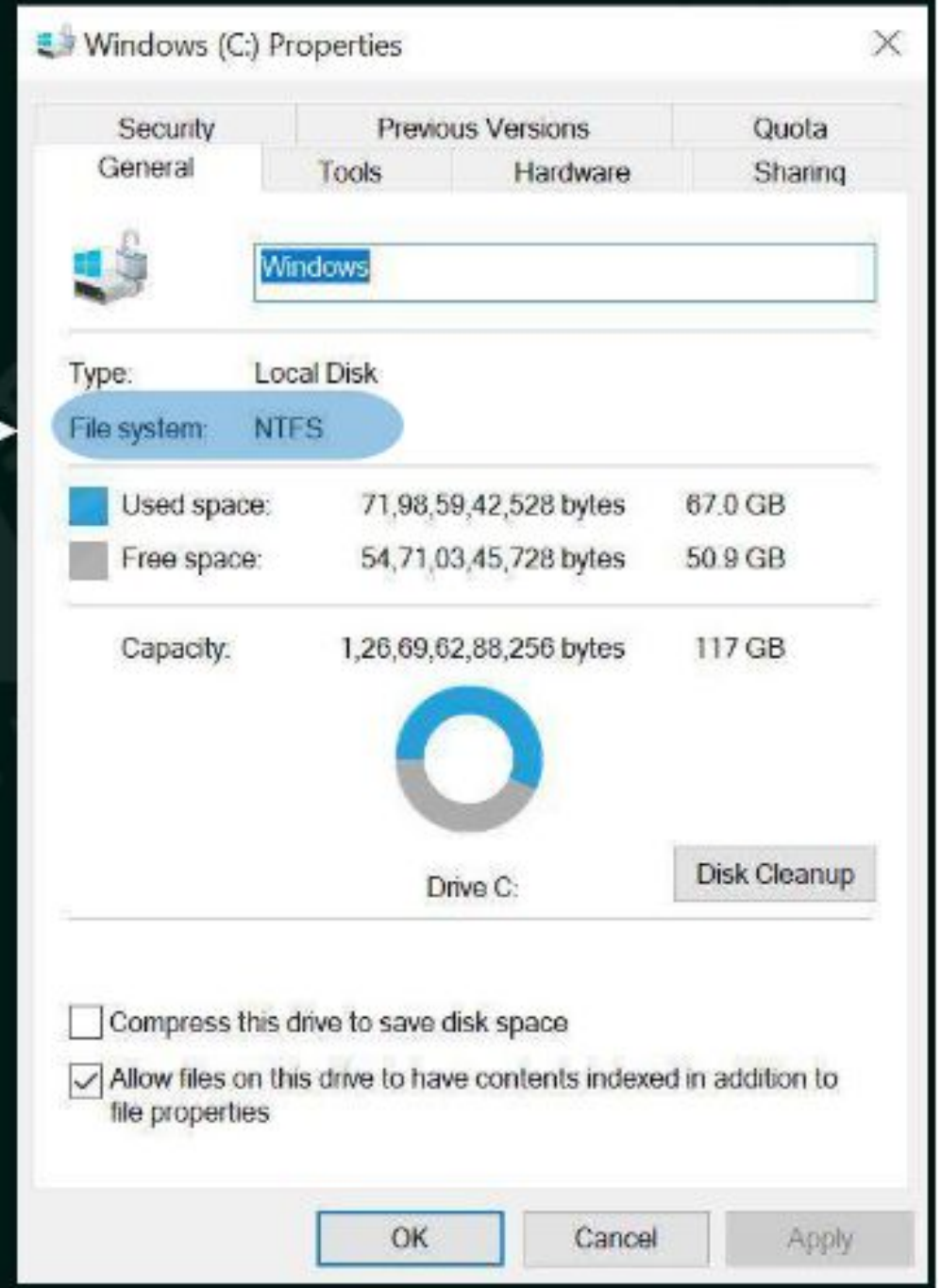
UNIX – Unix File System

Windows – FAT, FAT32, NTFS

Several on-disk and in-memory structures are used to implement a file system.

Right-Click

PROPERTIES



On-Disk Structures

On disk, the file system may contain information about:

- How to boot an operating system stored there
- The total number of blocks
- The number and location of free blocks
- The directory structure
- Individual files

Let's discuss some of these structures.

Boot Control Block

There is a boot control block for each volume.

- It is typically the first block of a volume.
- It contains information needed by the system to boot an operating system from that volume.
- If the disk does not contain an operating system, this block can be empty.

In Unix File System it is called the boot block.

In NTFS it is called the partition boot sector.

Volume Control Block

There is a volume control block for each volume.

It contains volume or partition details such as:

- The number of blocks in the partition
- Size of the blocks
- Free block count
- Free-block pointers
- Free FCB (File Control Block) count and FCB pointers

In Unix File System it is called a superblock.

In NTFS it is stored in the master file table.

Directory Structure per File System

This is used to organize the files.

In Unix File System it is includes file names and associated inode numbers.

In NTFS it is stored in the master file table.

Per-File FCB

It contains many details about the file such as:

- File permissions
- Ownership
- Size
- Location of the data blocks

In Unix File System it is called the inode.

In NTFS it is stored in the master file table.

Per-File FCB

It contains many details about the file such as:

- File permissions
- Ownership
- Size
- Location of the data blocks

In Unix File System it is called the inode.

In NTFS it is stored in the master file table.

A typical file control block

In-Memory Structures

The in-memory information is used for both file-system management and performance improvement via caching.

Here, the data are loaded at mount time

and

discarded at dismount.

The structures include:

- An in-memory mount table
- The system-wide open-file table
- An in-memory directory-structure cache
- The per-process open-file table

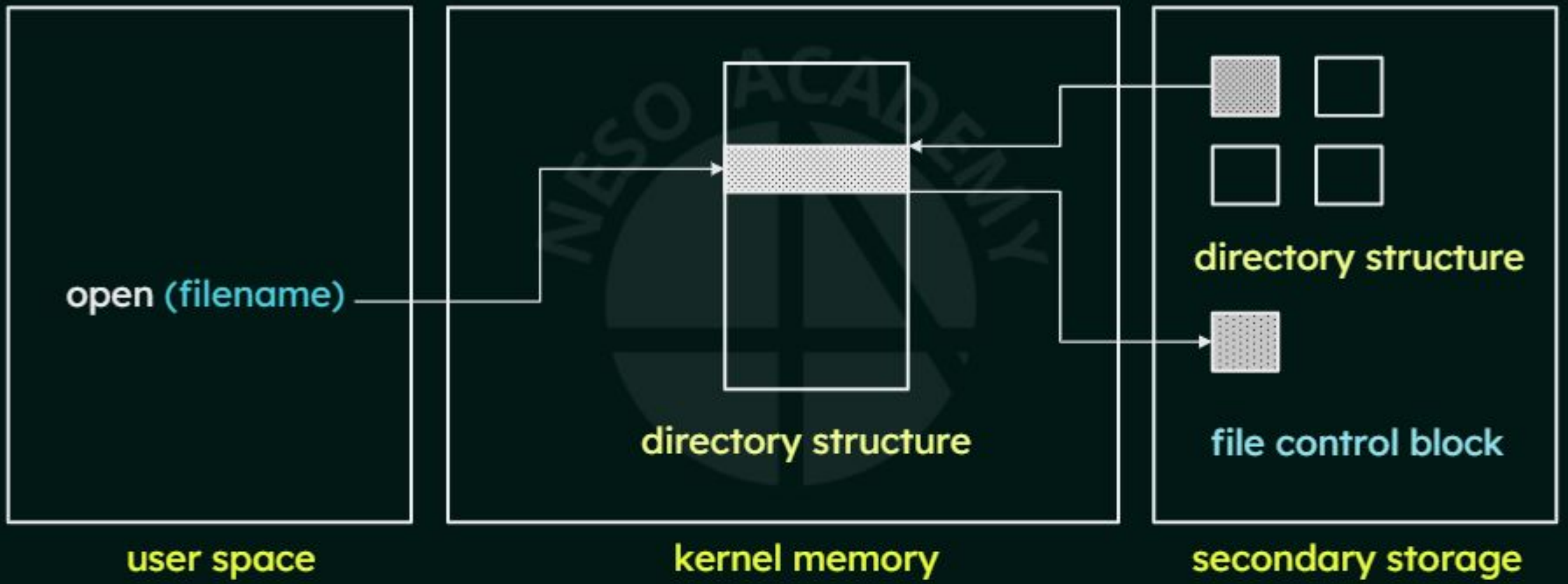
- **In-memory mount table:**
  Contains information about each mounted volume.

- **In-memory directory-structure cache:**
  Holds the directory information of recently accessed directories. (For directories at which volumes are mounted, it can contain a pointer to the volume table.)

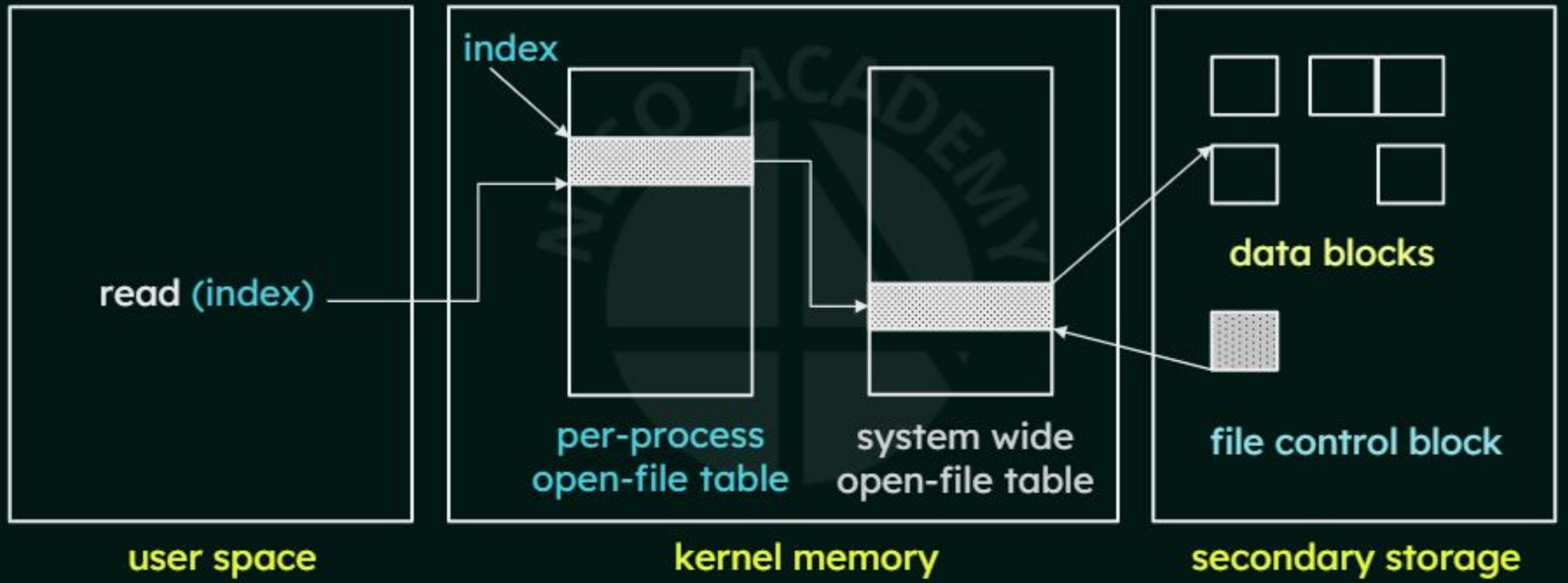
- **System-wide open-file table:**
  Contains a copy of the FCB of each open file, as well as other information.

- **Per-process open-file table:**
  Contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information.

File open


open (filename)
directory structure
directory structure
file control block
user space
kernel memory
secondary storage

File read


index
read (index)
per-process
open-file table
system wide
open-file table
data blocks
file control block
user space
kernel memory
secondary storage

Virtual File Systems

Modern operating systems must concurrently support multiple types of file systems.

BUT

🤔 How does an operating system allow multiple types of file systems to be integrated into a directory structure?

🤔 How can users seamlessly move between file-system types as they navigate the file-system space?

Object-Oriented Techniques

Most operating systems, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementation.

This allows:

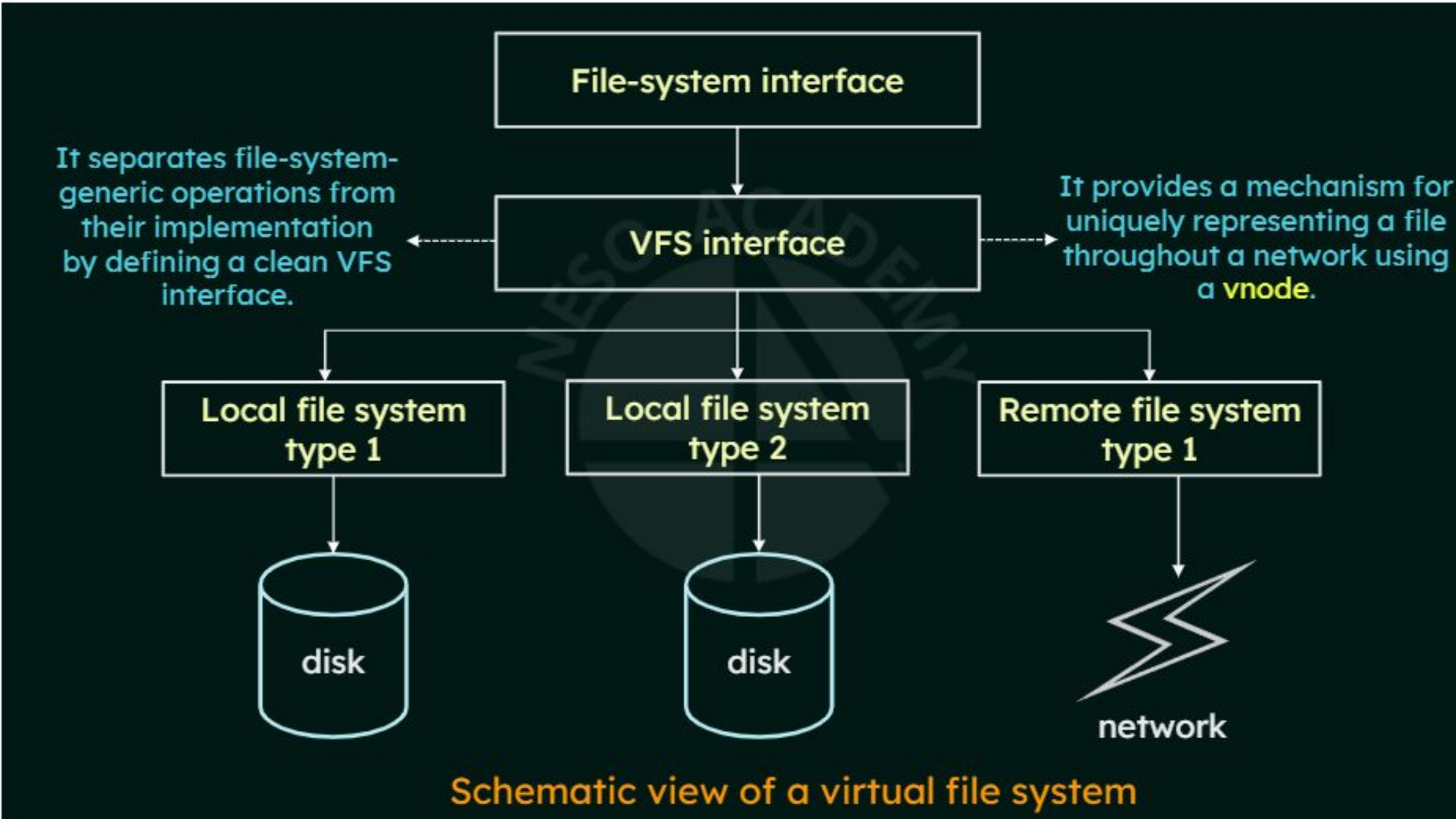
- Very dissimilar file-system types to be implemented within the same structure, including network file systems, such as NFS.
- Users to access files that are contained within multiple file systems on the local disk or even on file systems available across the network.

The file-system implementation consists of three major layers:

Layer 1: The file-system interface.

Layer 2: The virtual file system (VFS).

Layer 3: The layer implementing the file system type or the remote-file-system protocol.



Schematic view of a virtual file system

Example of VFS architecture in Linux

The four main object types defined by the Linux VFS are:

- The inode object → represents an individual file.
- The file object → represents an open file.
- The superblock object → represents an entire file system.
- The dentry object → represents an individual directory entry.

Every object of one of these types contains a pointer to a function table which has the addresses of the actual functions that implement the defined operations for that particular object.

Directory Implementation

The kind of directory-allocation & directory-management algorithms used in a system affects the efficiency, performance, and reliability of the file system.

Here we will discuss two methods of implementing a directory:

1. Linear List
2. Hash Table

Linear List

This is the simplest directory implementation method.

- Here, a linear list of file names is used which points to the data blocks.
- This is simple to implement but time consuming to execute.

To create a new file:

- Search the directory to be sure that no existing file has the same name.
- Add a new entry at the end of the directory.

To delete a file:

- Search the directory for the named file.
- Release the space allocated to it.

Biggest disadvantage → A linear search is always required to find the files. This will slow down the entire process as users frequently need to access directory information.

Hash Table

Here, a linear list stores the directory entries, but a hash data structure is also used.

- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- This greatly decrease the directory search time.

Problems to be considered:

- Collisions — situations in which two file names hash to the same location.
- Hash tables are generally fixed size and the hash functions depends on that size.

Example of Hash Function

Suppose we have 5 files:

Physics, Chemistry, Mathematics, Biology, Economics.

They are associated with keys (x) :

Physics - 126

Chemistry - 92

Mathematics - 34

Biology - 345

Economics – 453

	Key	Filename
0	345	Biology
1	126	Physics
2	92	Chemistry
3	453	Economics
4	34	Mathematics

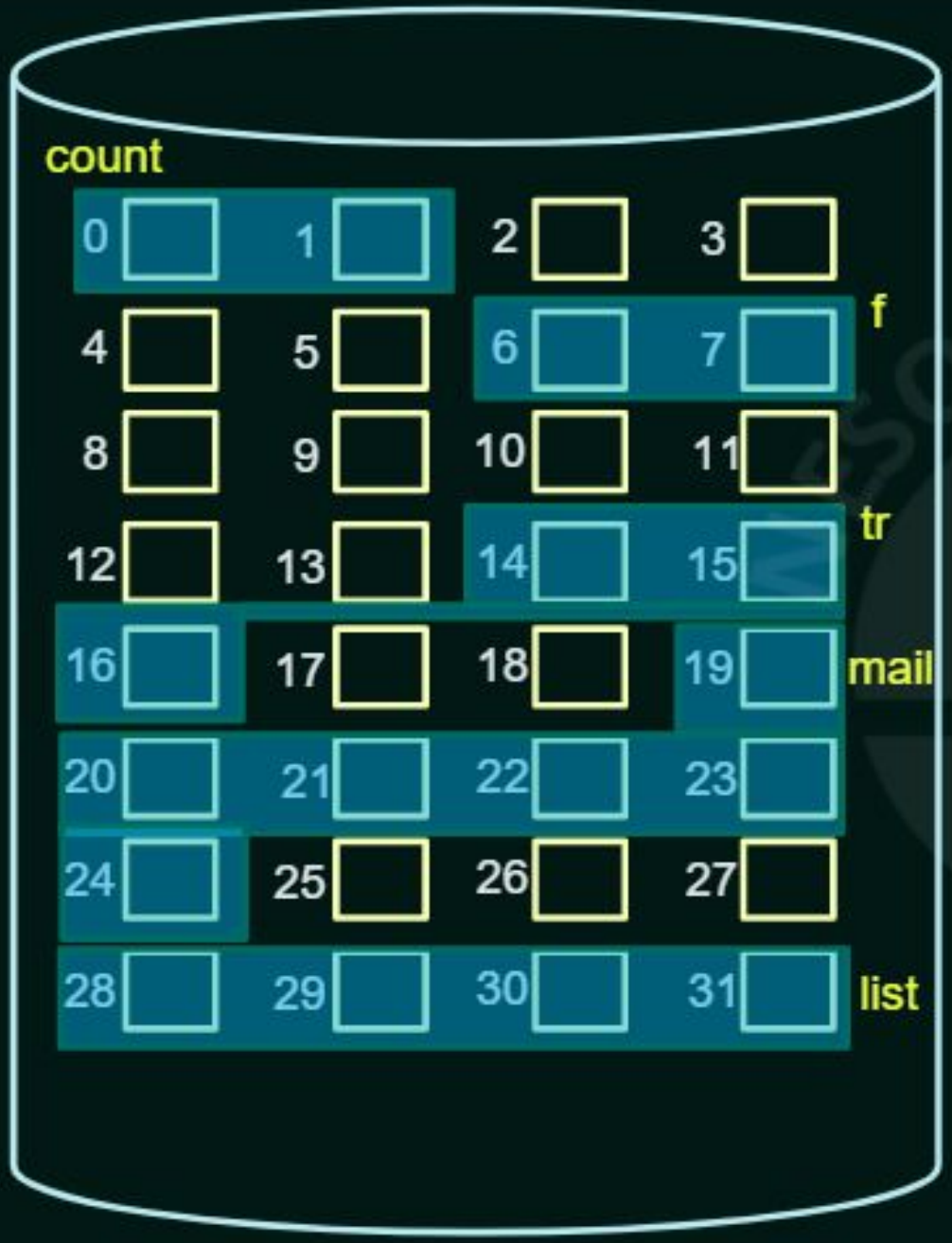
Hash function, h (x) = x mod 5

Contiguous Disk Space Allocation

Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk.

- Disk addresses define a linear ordering on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
- E.g. If a file is 'n' blocks long and starts at location 'b'

  Then, it occupies the blocks b, b+1, b+2, ..., b + n-1.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.



Directory

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

Contiguous allocation of disk space

Advantages

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.
- When necessary, it reads the next block.

For Direct Access:

- For accessing block ‘i’ of a file that starts at block ‘b’,
- We can immediately access block b + i.

Disadvantages

Finding space for a new file is difficult.

- First fit & Best fit algorithms can be used for selecting free holes from a set of available holes.
- Both these algorithms suffer from external fragmentation.
- As files are allocated and deleted, the free disk space is broken into little pieces.
- Some older PC systems used contiguous allocation on floppy disks but this was not a good solution as it consumes a lot of time.

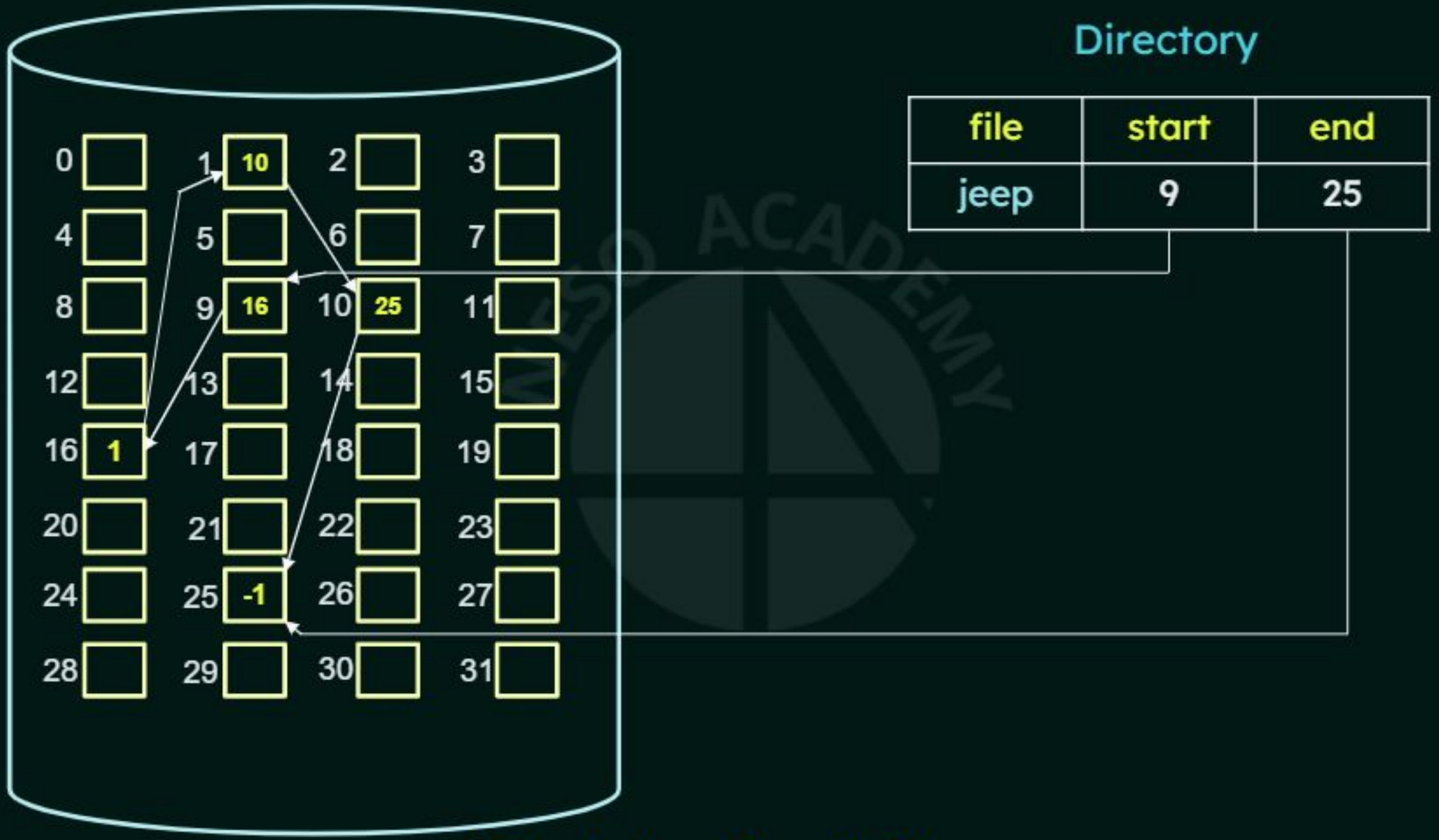
Determining how much space is needed for a file is a difficult task.

- If the space allocated for a file is too less, and it needs more space later, then,
  - » Either the user program is terminated & the user must allocate more space & run the program again. OR
  - » The system finds a larger hole, copies the content to the new space & releases the previous space.

Linked Disk Space Allocation

Linked allocation solves all problems of contiguous allocation.

- Each file is a linked list of disk blocks (which may be scattered anywhere on the disk).
- The directory contains a pointer to the first and last blocks of the file.
- Each block contains a pointer to the next block.
- E.g. If each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.



Linked allocation of disk space

Advantages

- There is no external fragmentation here.
- The size of a file need not be declared when that file is created.
- Creating, reading & writing to files can be easily done.

Creating a file:

- Simply create a new entry in the directory.
- Each directory entry has a pointer to the first disk block of the file.
- This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file.

Writing to a file:

- Find a free block using the free-space management system.
- This new block is written to and is linked to the end of the file.

Reading a file:

- Simply read blocks by following the pointers from block to block.

Disadvantages

It can be used effectively only for sequential-access files.

To find the i^{th} block of a file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block.

Space required for the pointers.

If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers.

Solution: Allocate clusters (multiples of blocks) rather than just blocks.

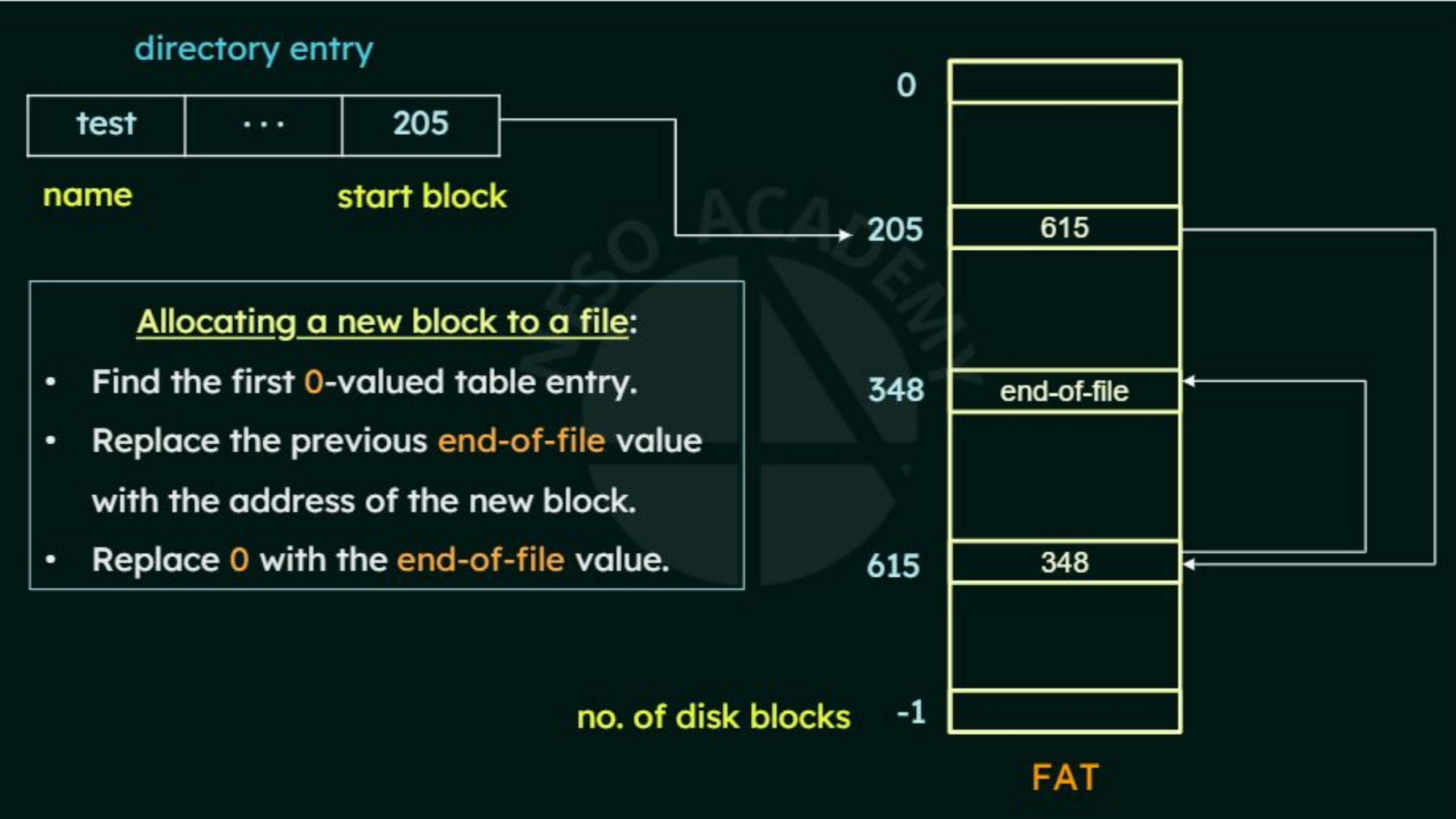
Reliability.

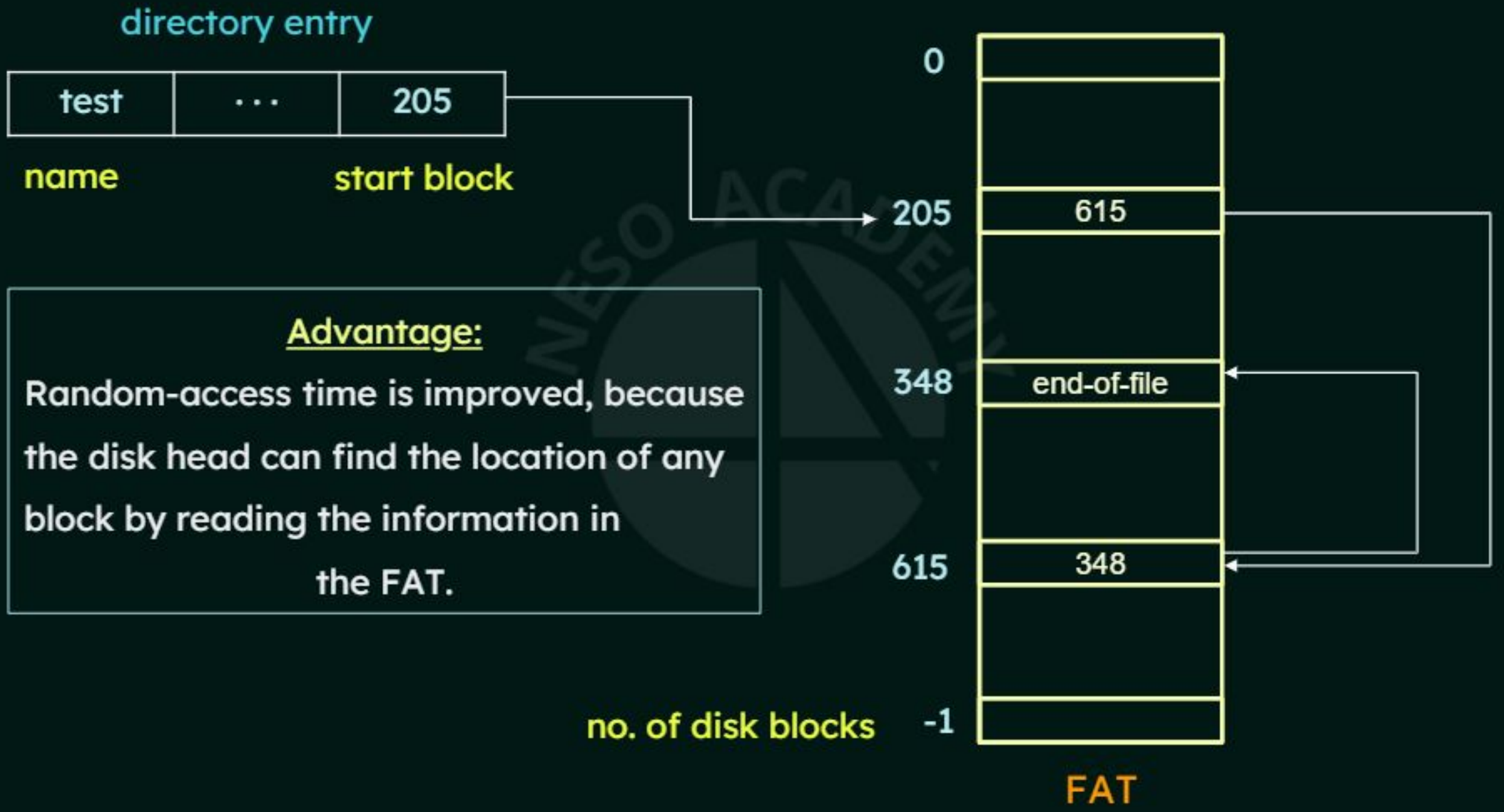
If pointers which are used for linking the files gets lost or damaged or if wrong pointers are picked up, this will lead to problems.

File Allocation Table (FAT)

This is an important variation on linked allocation method.

- This method is used by the MS-DOS and OS/2 operating systems.
- A section of disk at the beginning of each volume is set aside to contain the table.
- The table 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.
- The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until the last block, which has a special end-of-file value as the table entry.
- Unused blocks are indicated by a 0 table value.


directory entry
test
· · ·
205
name
start block
Allocating a new block to a file:
• Find the first 0-valued table entry.
• Replace the previous end-of-file value with the address of the new block.
• Replace 0 with the end-of-file value.
0
205
348
615
no. of disk blocks
-1
615
end-of-file
348
FAT


directory entry
test
· · ·
205
name
start block
Advantage:
Random-access time is improved, because the disk head can find the location of any block by reading the information in the FAT.
0
205
348
615
615
end-of-file
348
no. of disk blocks
-1
FAT

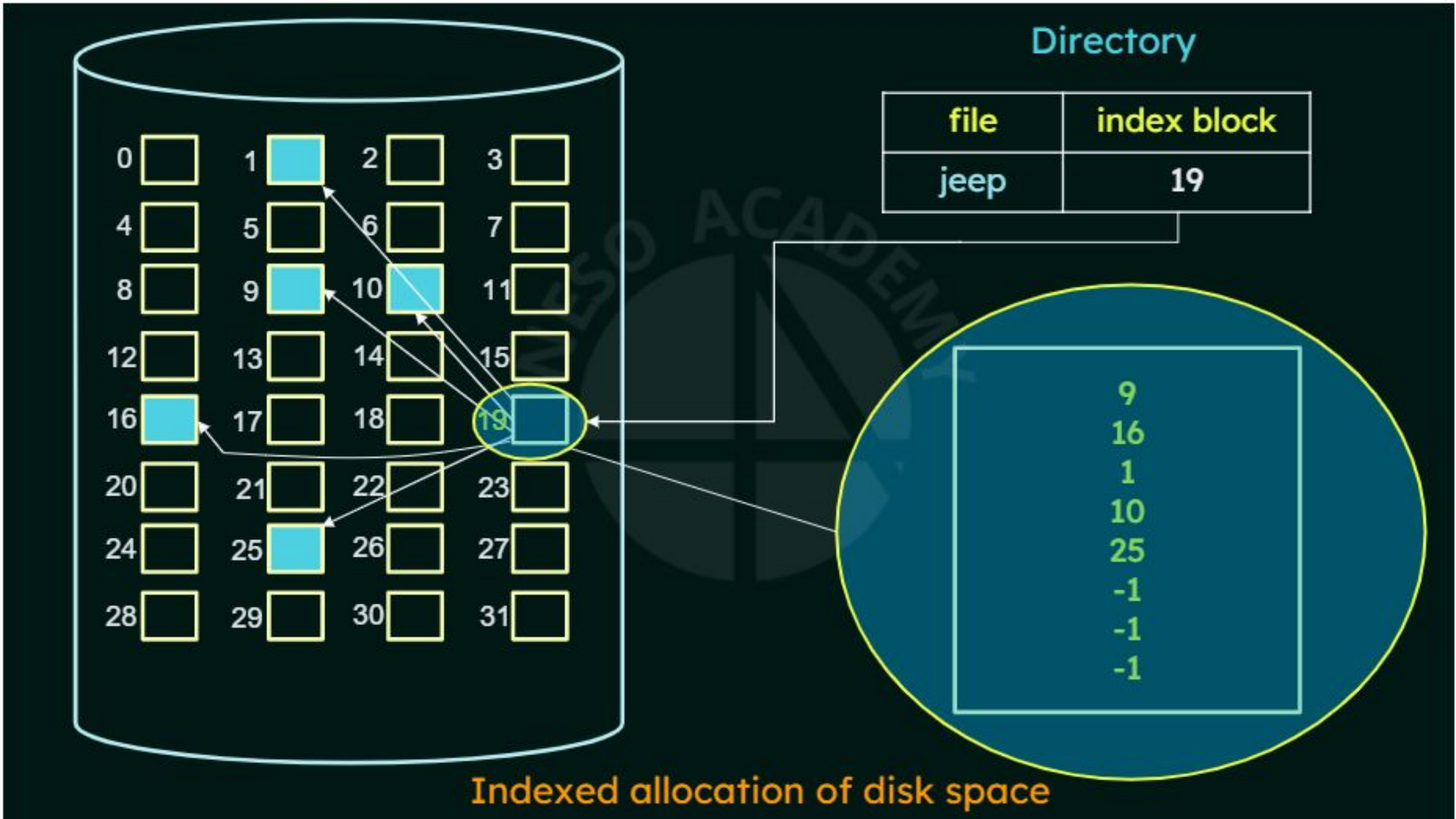
Indexed Disk Space Allocation

The main disadvantage of Linked Allocation:

It cannot support efficient direct access.

Indexed allocation solves this problem by bringing all the pointers together into one location: **the index block**

- Each file has its own index block → an array of disk-block addresses.
- The i^{th} entry in the index block points to the i^{th} block of the file.
- The directory contains the address of the index block.
- To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.



Indexed allocation of disk space

While Creating a File

- All pointers in the index block are set to nil.
- When the i^{th} block is first written, a block is obtained from the free-space manager.
- Its address is put in the i^{th} index-block entry.

Advantages

- Supports direct access, without suffering from external fragmentation.
- Any free block on the disk can satisfy a request for more space.

Disadvantage

- It suffers from wasted space.
- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

How large should the index blocks be?

- If it is too small, it will not be able to hold enough pointers for a large file.
- If its is too large (larger than what is required for the file) it is a waste of space.

Some mechanisms to deal with this issue:

- Linked scheme.
- Multilevel index.
- Combined scheme.

Performance of Disk Space Allocation Methods

We have discussed about 3 main disk space allocation methods:

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

All these methods vary in their storage efficiency and data-block access times.

Important criteria in selecting the proper method or methods for an operating system to implement.

Before selecting an allocation method, we need to determine how the systems will be used.

E.g. A system with mostly sequential access should not use the same method as a system with mostly random access.

Contiguous Allocation: Requires only one access to get a disk block for any type of access (sequential or direct). Only the initial block address has to be kept in memory.

- ✔ Sequential Access
- ✔ Direct Access

Linked Allocation: The address of the next block can be kept in memory & can be read directly. Hence, it is good for sequential access. But an access to the i^{th} disk block will require i disk reads.

- ✔ Sequential Access
- ✘ Direct Access

Before selecting an allocation method, we need to determine how the systems will be used.

Some systems make use of both Contiguous & Linked Allocation methods:

- They support direct-access files by using contiguous allocation and sequential access by linked allocation.
- Here, the type of access to be made must be declared when the file is created.
- A file created for sequential access will be linked and cannot be used for direct access.
- A file created for direct access will be contiguous and can support both direct access and sequential access (maximum length must be declared when it is created).
- The O.S. must have appropriate data structures and algorithms to support both allocation methods.

Before selecting an allocation method, we need to determine how the systems will be used.

Indexed allocation – more complex. ✔ Sequential Access ✔ Direct Access

- If the index block is already in memory, then the access can be made directly.
- But, keeping the index block in memory requires considerable space.
- If this memory space is not available, then we may have to read first the index block and then the desired data block.
- For multi-level index blocks, there will be multiple index blocks reads necessary.
- E.g. For a very large file, accessing a block near the end of the file will need all the index blocks to be read first.
- So performance here depends on:

  - Index structure
  - Size of the file
  - Position of the desired block

Before selecting an allocation method, we need to determine how the systems will be used.

Some systems combine Contiguous Allocation with Indexed Allocation:

- They use contiguous allocation for small files (up to three or four blocks).
- It is automatically switched to an indexed allocation if the file grows large.
- Since most files are small, and contiguous allocation is efficient for small files average performance can be quite good.

The UNIX inode

An inode is a data structure in UNIX operating systems that contains important information pertaining to files within a file system.

- When a file system is created in UNIX, a specific amount of inodes are also created.
- Usually, about 1% of the total file system disk space is allocated to the inode table.

The inode structure

mode – Information regarding file type and permissions.

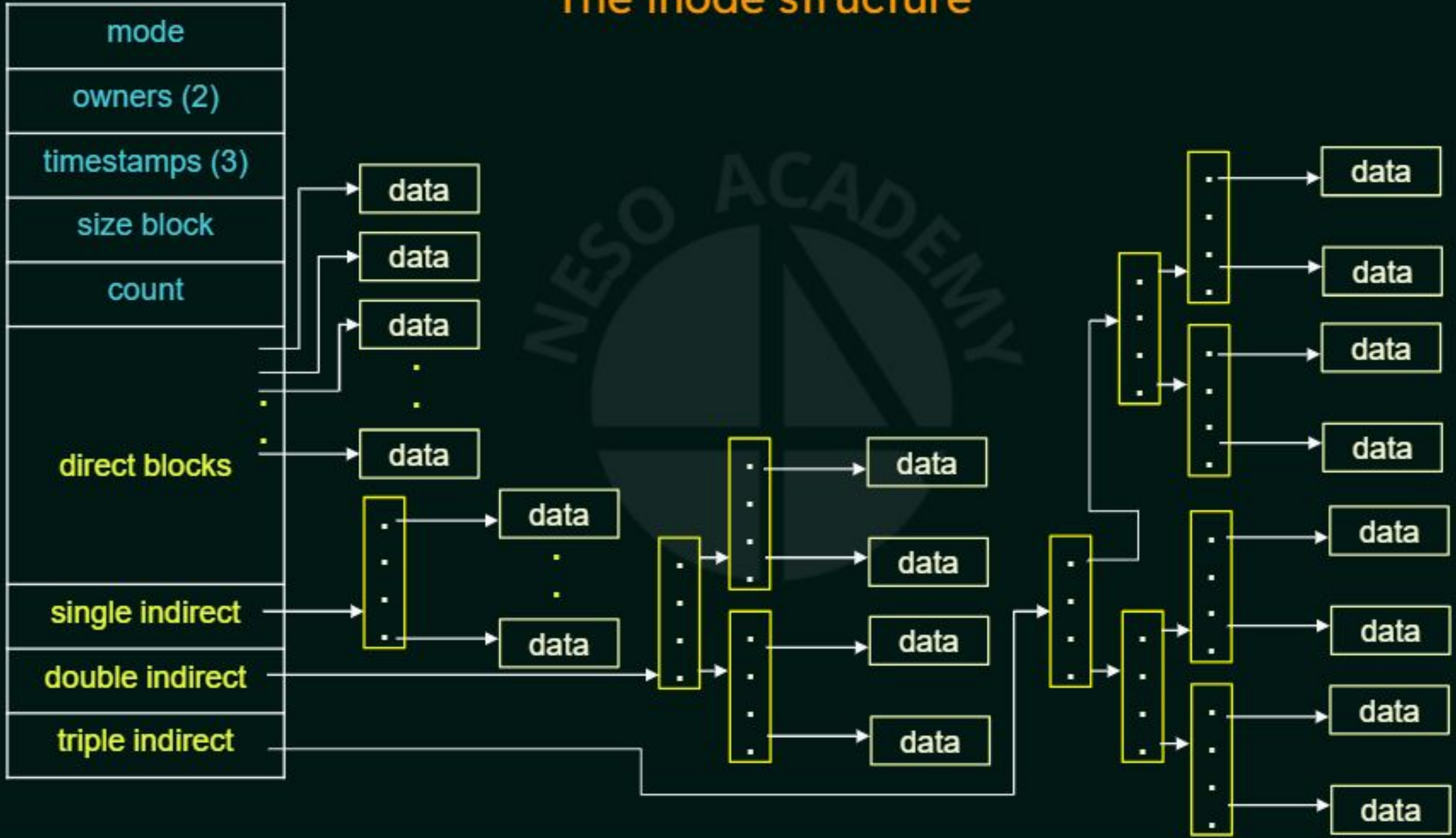
owners - Access details like owner of the file, group of the file etc.

timestamps – Stores the time of file access, modification, etc.

size block – Specifies the size of the blocks.

count – Number of blocks.

The inode structure



Free-Space Management

Disk space is limited.

So, we need to reuse the space from deleted files for new files, if possible.

To keep track of free disk space, the system maintains a free-space list which records all free disk blocks—those not allocated to some file or directory.

To create a file:

- We search the free-space list for the required amount of space.
- Allocate that space to the new file.
- This space is then removed from the free-space list.

When a file is deleted:

- Its disk space is added to the free-space list.

Bit Vector

The free-space list is implemented as a bit map or bit vector.

- Each block is represented by 1 bit.
- If the block is free, the bit is 1. If the block is allocated, the bit is 0.

E.g. Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free and the rest of the blocks are allocated.

The free-space bit map would be:

0 0 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 . . .

Advantage:

- It is relatively simple in finding the first free block or n consecutive free blocks on the disk.

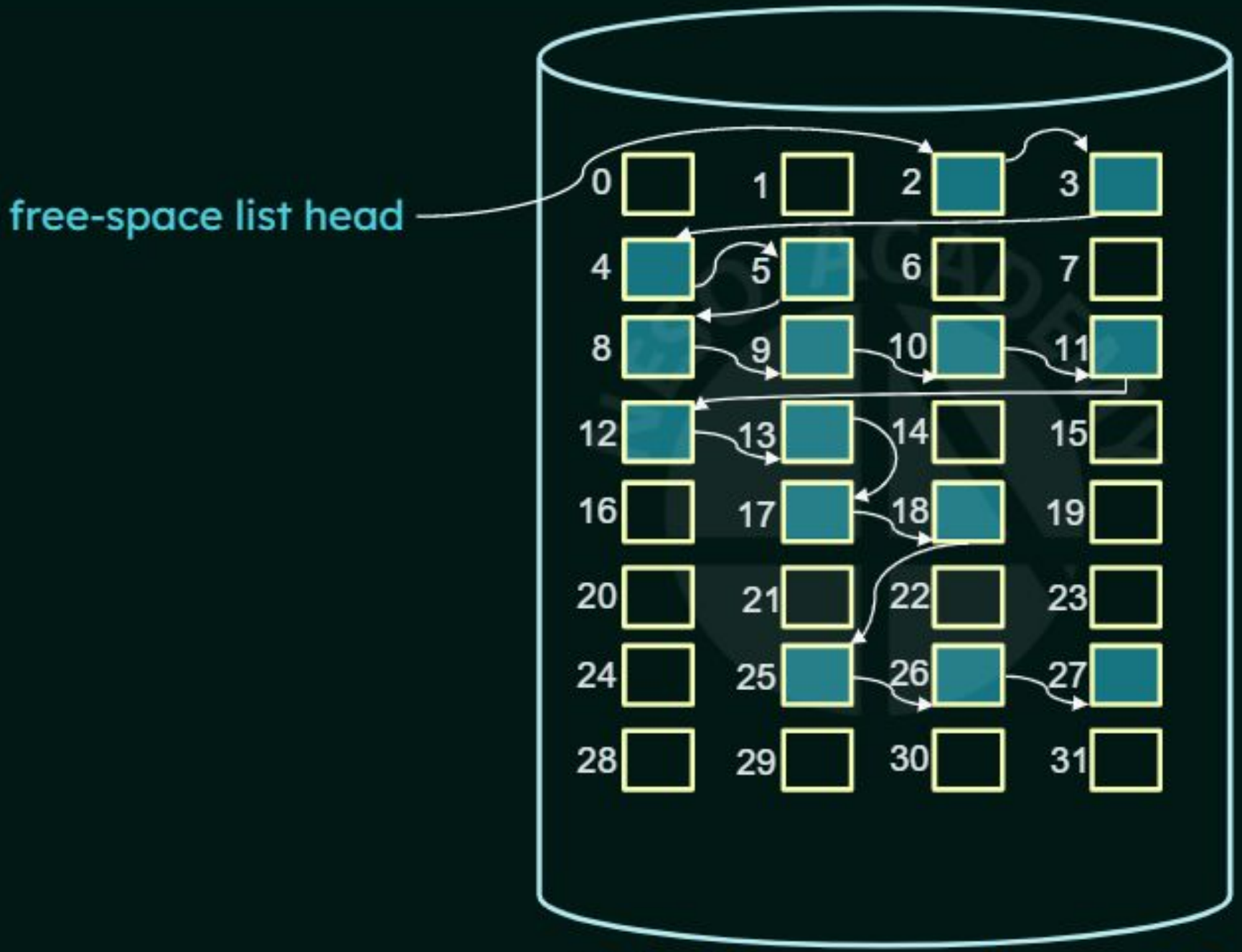
Disadvantage:

- It is inefficient unless the entire vector is kept in main memory.

Linked List

All the free disk blocks are linked together.

- A pointer pointing to the first free block is kept in a special location on the disk.
- It is cached in memory.
- This first block contains a pointer to the next free disk block, and so on.



Linked free-space list on disk.

Grouping

Store the addresses of n free blocks in the first free block.

- The first n-1 of these blocks are actually free.
- The last block contains the addresses of another n free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

Counting

Here, we take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.

- We 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.

Implementation of free space management list

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

File System Implementation

Solved Problem - 1

GATE 2004

A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointers. Disk block size is 1 KB, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size ?

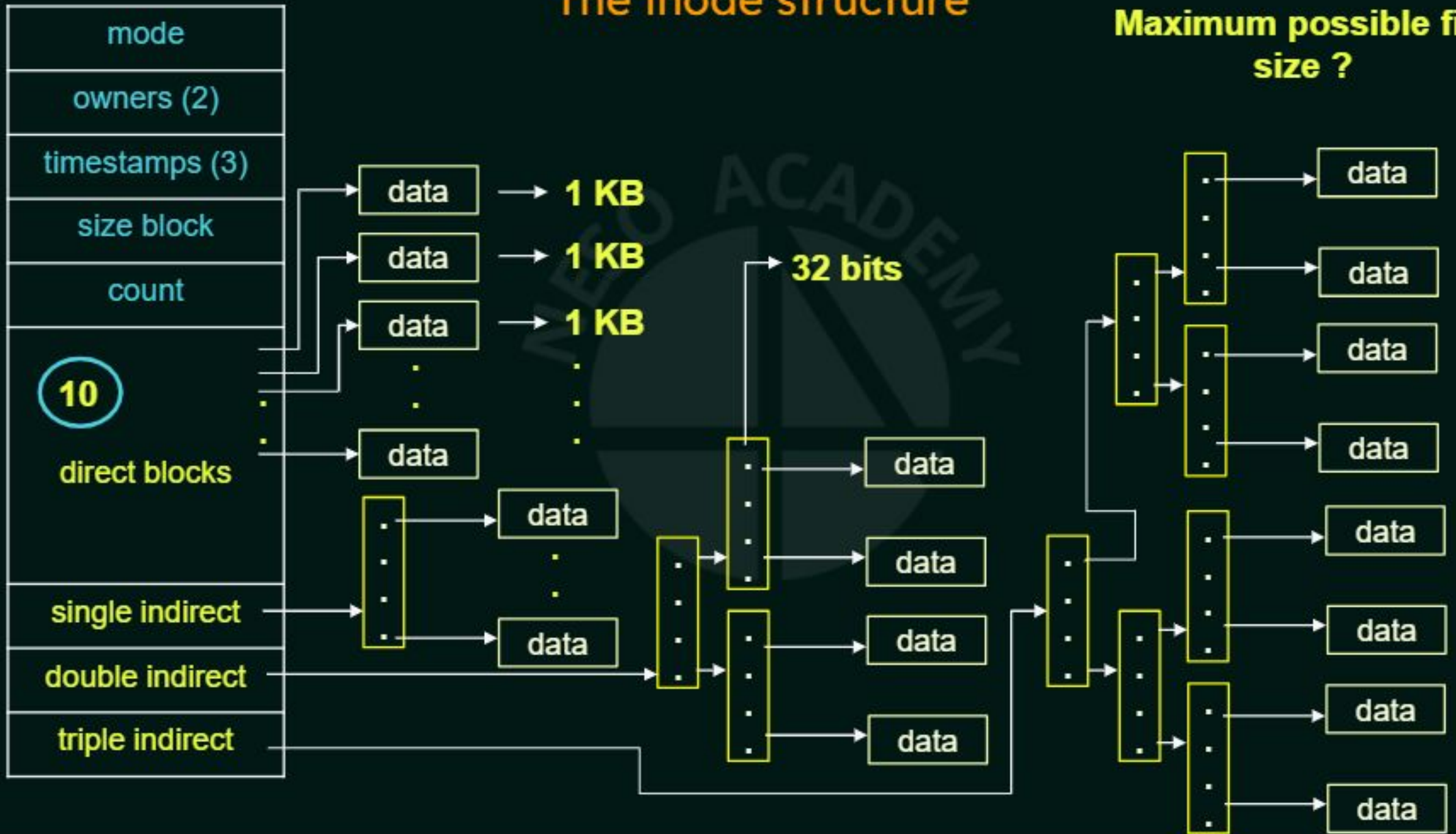
(A) 2^{24} bytes

(B) 2^{32} bytes

(C) 2^{34} bytes

(D) 2^{48} bytes

The inode structure



A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointers. Disk block size is 1 KB, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size ?

Size of disk block = 1 KB = 1024 bytes

Disk block address = 32 bits = 4 bytes

$$\text{Number of addresses per block} = \frac{\text{Block size}}{\text{Space occupied by each address}} = \frac{1024}{4} = 256 = 2^8$$

Maximum size of file = (10 Direct pointers x 1 KB) + (1 Single Indirect pointer x 1 KB) + (1 Double Indirect pointers x 1 KB) + (1 Triple Indirect pointers x 1 KB)

$= (10 \times 2^{10}) + (2^8 \times 2^{10}) + (2^8 \times 2^8 \times 2^{10}) + (2^8 \times 2^8 \times 2^8 \times 2^{10})$

$= 2^{13} + 2^{18} + 2^{26} + 2^{34}$

$\approx 2^{34}$ bytes

File System Implementation

Solved Problem - 1

GATE 2004

A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointers. Disk block size is 1 KB, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size ?

(A) 2^{24} bytes

(B) 2^{32} bytes

(C) 2^{34} bytes

(D) 2^{48} bytes

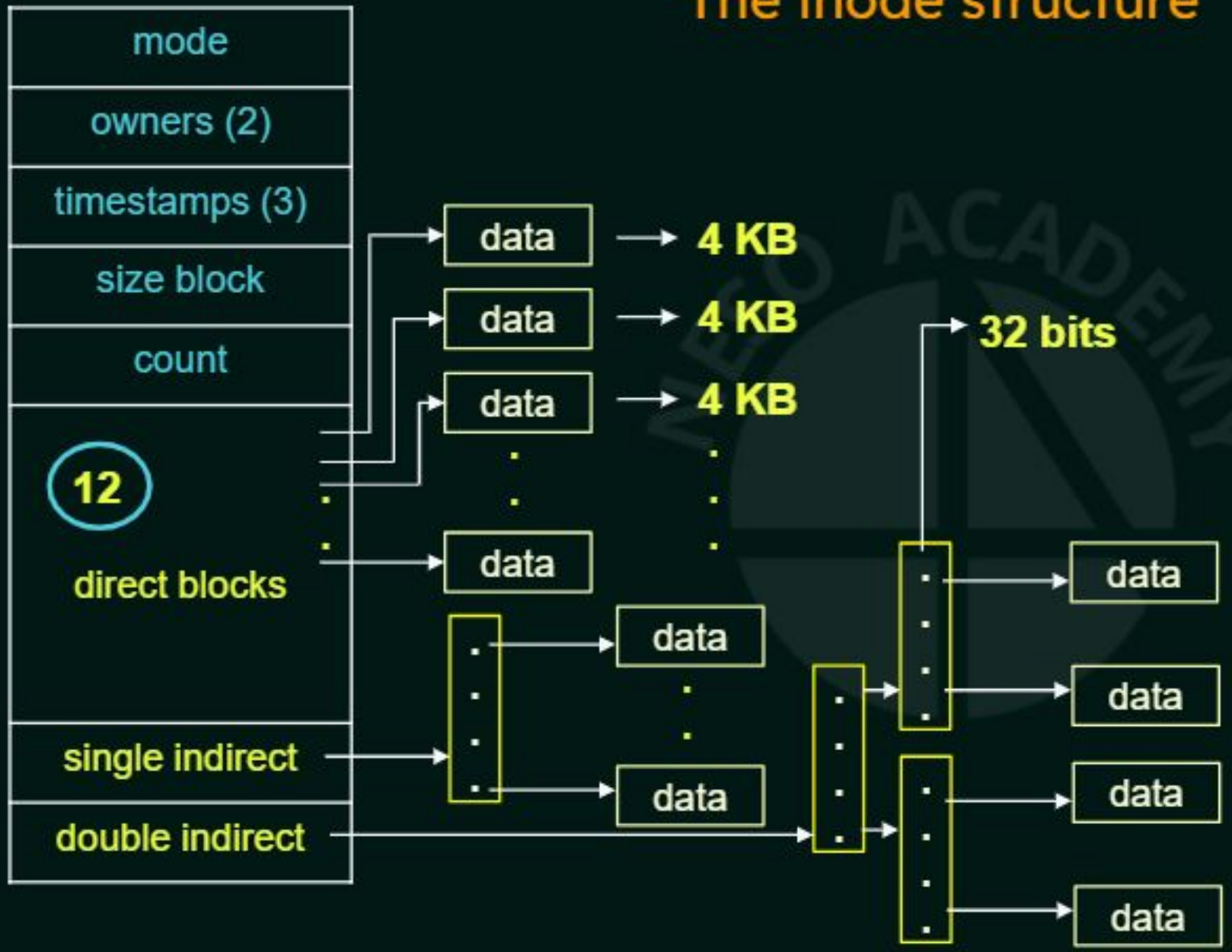
File System Implementation

Solved Problem - 2

GATE 2019

The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointer. The disk block size is 4 KB and the disk block addresses 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____GB.

The inode structure



Maximum possible file size ?

The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointer. The disk block size is 4 KB and the disk block addresses 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____ GB.

Size of disk block = 4 KB = 4096 bytes = 2^{12} bytes

Disk block address = 32 bits = 4 bytes

$$\text{Number of addresses per block} = \frac{\text{Block size}}{\text{Space occupied by each address}} = \frac{4096}{4} = 1024 = 2^{10}$$

Maximum size of file = (12 Direct pointers x 4 KB) + (1 Single Indirect pointer x 4 KB) + (1 Double Indirect pointers x 4 KB)

$= (12 \times 2^{12}) + (2^{10} \times 2^{12}) + (2^{10} \times 2^{10} \times 2^{12})$

$= 2^{15} + 2^{22} + 2^{32}$

$\approx 2^{32}$ bytes = 4 GB

File System Implementation

Solved Problem - 2

GATE 2019

The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointer. The disk block size is 4 KB and the disk block addresses 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____GB.

Answer: 4 GB

File System Implementation

Solved Problem - 3

GATE 2014

A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size is 10^3 bytes, the maximum size of a file that can be stored on this disk in units of 10^6 bytes is _____.

A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size is 10^3 bytes, the maximum size of a file that can be stored on this disk in units of 10^6 bytes is _____.

Total disk size = 100×10^6 bytes

Size of disk block = 10^3 bytes

Overhead for each FAT entry = 4 bytes

$$\text{Number of entries in the FAT} = \frac{\text{Total disk size}}{\text{Size of disk block}} = \frac{100 \times 106 \text{ bytes}}{10^3 \text{ bytes}} = 100 \times 10^3 = 10^5$$

Total space occupied by FAT = Number of entries in FAT x Overhead for each FAT entry

$= 10^5 \times 4$ bytes $= 0.4 \times 10^6$ bytes

Total disk size = 100×10^6 bytes

Size of disk block = 10^3 bytes

Overhead for each FAT entry = 4 bytes

$$\text{Number of entries in the FAT} = \frac{\text{Total disk size}}{\text{Size of disk block}} = \frac{100 \times 106 \text{ bytes}}{10^3 \text{ bytes}} = 100 \times 10^3 = 10^5$$

Total space occupied by FAT = Number of entries in FAT x Overhead for each FAT entry

$= 10^5 \times 4$ bytes $= 0.4 \times 10^6$ bytes

Maximum file size that can be stored on the disk

= Total disk size - Total space occupied by FAT

= $(100 \times 10^6 - 0.4 \times 10^6)$ bytes

= 99.6×10^6 bytes

File System Implementation

Solved Problem - 3

GATE 2014

A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size is 10^3 bytes, the maximum size of a file that can be stored on this disk in units of 10^6 bytes is _____.

Answer: 99.6 x 10^6 bytes

File System Implementation

Solved Problem - 4

GATE 2012

A file system with 300 GB disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 bytes and the size of each disk block address is 8 bytes. The maximum possible file size in this file system is _____.

(A) 3 KB

(B) 35 KB

(C) 280 bytes

(D) Dependent on the size of the disk

A file system with 300 GB disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 bytes and the size of each disk block address is 8 bytes. The maximum possible file size in this file system is_____.

Size of each disk block = 128 bytes

Each disk block address size = 8 bytes

Number of addresses that can be stored in a block = $\frac{\text{Disk block size}}{\text{Address size}} = \frac{128 \text{ bytes}}{8 \text{ bytes}} = 16$

Maximum file size = [(8 direct block addresses) + (1 indirect block address)
+ (1 doubly indirect block address)]

= [(8 x 128) + (16 x 128) + (16 x 16 x 128)] bytes

= [1024 + 2048 + 32768] bytes

= 35840 bytes = 35 KB

File System Implementation

Solved Problem - 4

GATE 2012

A file system with 300 GB disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 bytes and the size of each disk block address is 8 bytes. The maximum possible file size in this file system is _____.

(A) 3 KB

(B) 35 KB

(C) 280 bytes

(D) Dependent on the size of the disk

File System Implementation

Solved Problem - 5

GATE 1996

A file system with a one-level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk-block 0	File Allocation Table, consisting of one 8-bit entry per data block, representing the data block address of the next data block in the file.
Disk-block 1	Directory, with one 32-bit entry per file.
Disk-block 2	Data block 1;
Disk-block 3	Data block 2; etc.

(a) What is the maximum possible number of files?

(b) What is the maximum possible file size in blocks?

A file system with a one-level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk-block 0	File Allocation Table, consisting of one 8-bit entry per data block, representing the data block address of the next data block in the file.
Disk-block 1	Directory, with one 32-bit entry per file.
Disk-block 2	Data block 1;
Disk-block 3	Data block 2; etc.

(a) What is the maximum possible number of files?

(b) What is the maximum possible file size in blocks?

Size of each disk block = 4 KB

We have just 1 block that holds the details of every file in the system. Each entry in this block represents a file and each entry size = 32 bits.

So, maximum possible number of files = $\frac{\text{Disk block size}}{\text{Size of each entry in block}-1} = \frac{4\text{ KB}}{32\text{ bits}} = \frac{4096\text{ bytes}}{4\text{ bytes}}$

= 1024

A file system with a one-level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk-block 0	File Allocation Table, consisting of one 8-bit entry per data block, representing the data block address of the next data block in the file.
Disk-block 1	Directory, with one 32-bit entry per file.
Disk-block 2	Data block 1;
Disk-block 3	Data block 2; etc.

(a) What is the maximum possible number of files?

(b) What is the maximum possible file size in blocks?

The FAT contains the block address of the next data block of the file.

8 bits are used for representing each entry (address).

So, total number of block address details that can be accommodated in the FAT = 2^8.

Hence, maximum file size in blocks = 2^8 = 256.

File System Implementation

Solved Problem - 5

GATE 1996

A file system with a one-level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk-block 0	File Allocation Table, consisting of one 8-bit entry per data block, representing the data block address of the next data block in the file.
Disk-block 1	Directory, with one 32-bit entry per file.
Disk-block 2	Data block 1;
Disk-block 3	Data block 2; etc.

(a) What is the maximum possible number of files? → 1024

(b) What is the maximum possible file size in blocks? → 256