on disk data structures used in file system implementation:

disk → secondary memory.

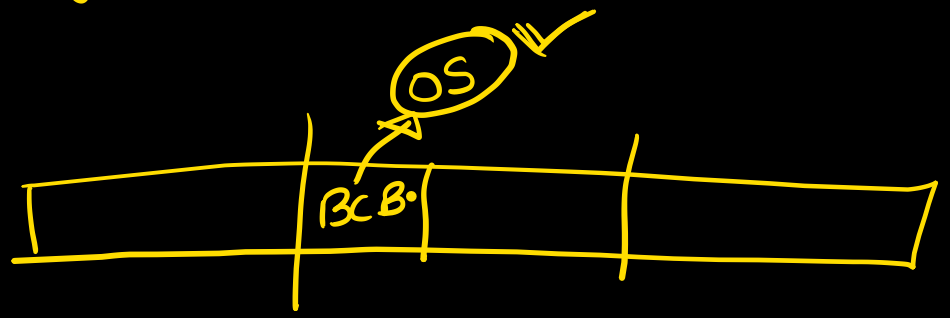
Several in memory and on disk structures are used to implement a file system. These structures vary from OS to OS.

Boot Control block: It contains information needed by the system to boot an OS from a partition. In unix it is called boot block.

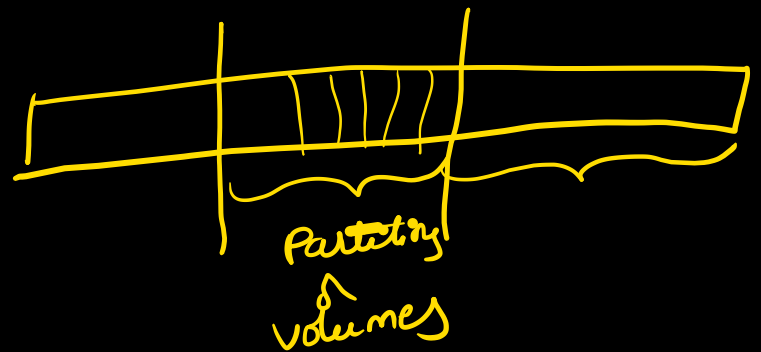
boot → load

In NTFS it is called partition block sector.

NTFS → New technology File system (window)



Volume control block:



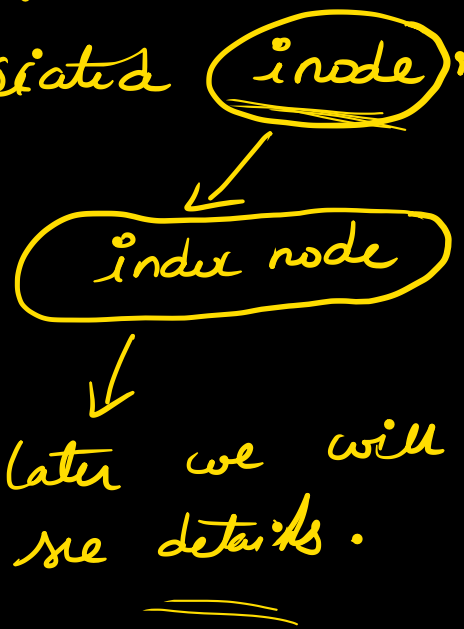
It contains volume (partition) details such as number of blocks in the partition, size of each block.

In unix, it is called super block

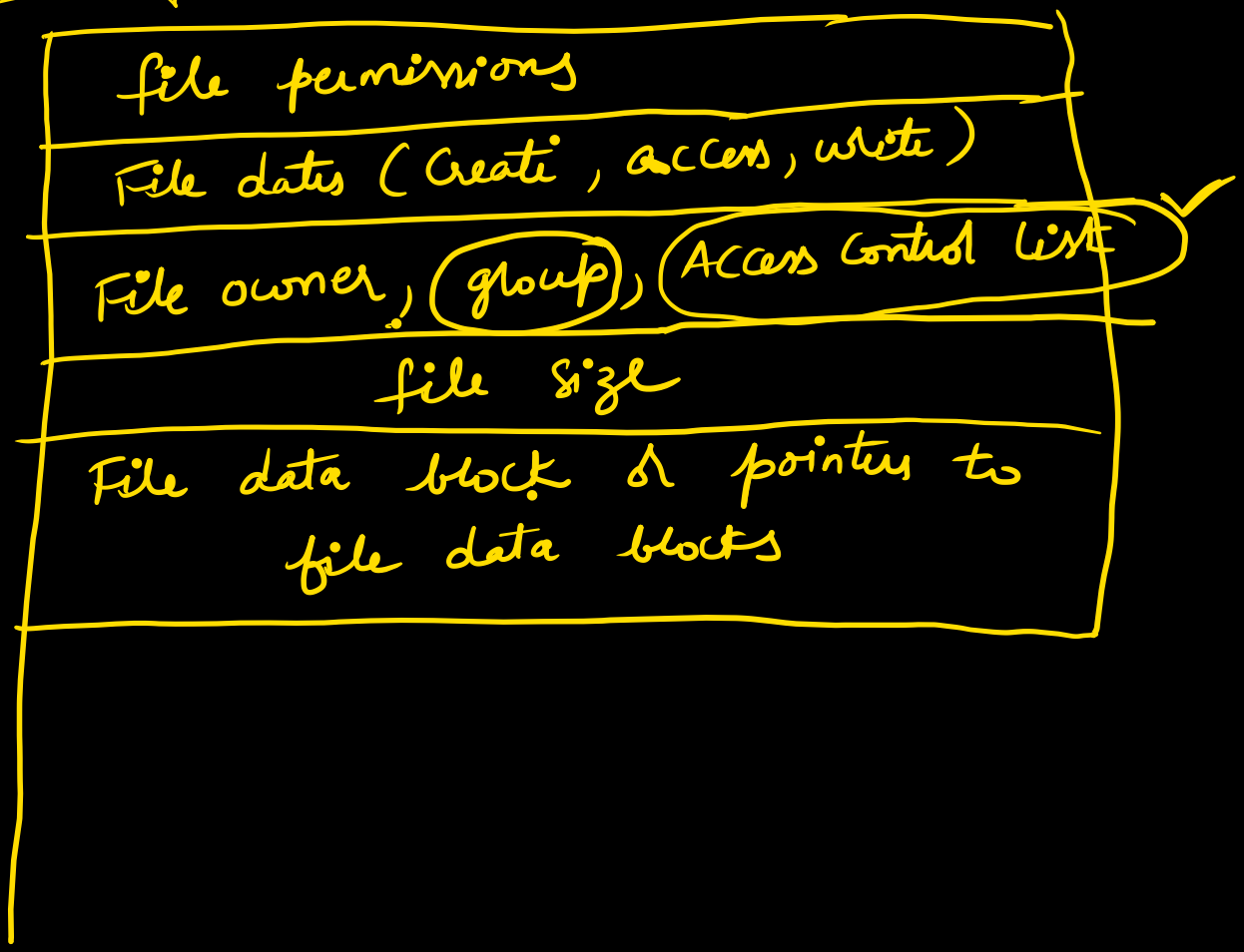
In NTFS, it is called master file table.

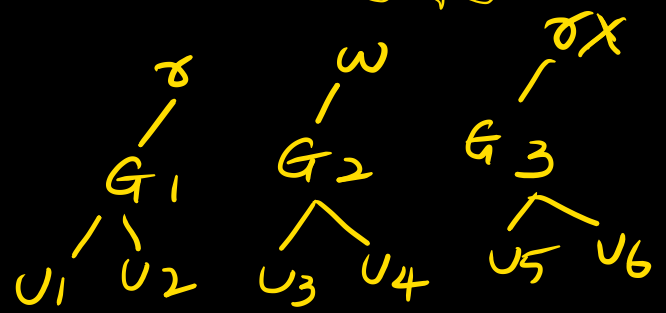
Directory structure: It is used to organize the files.

In unix, this includes file names and associated inode numbers



File control block : FCB : It Contains details about the file.





In memory data structures used in File system implementation:

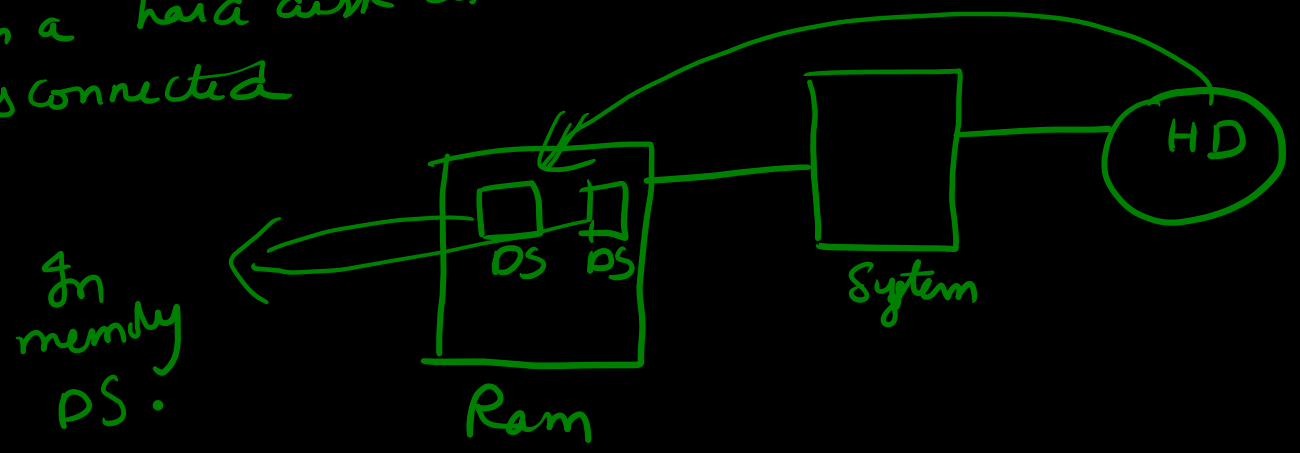
RAM

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

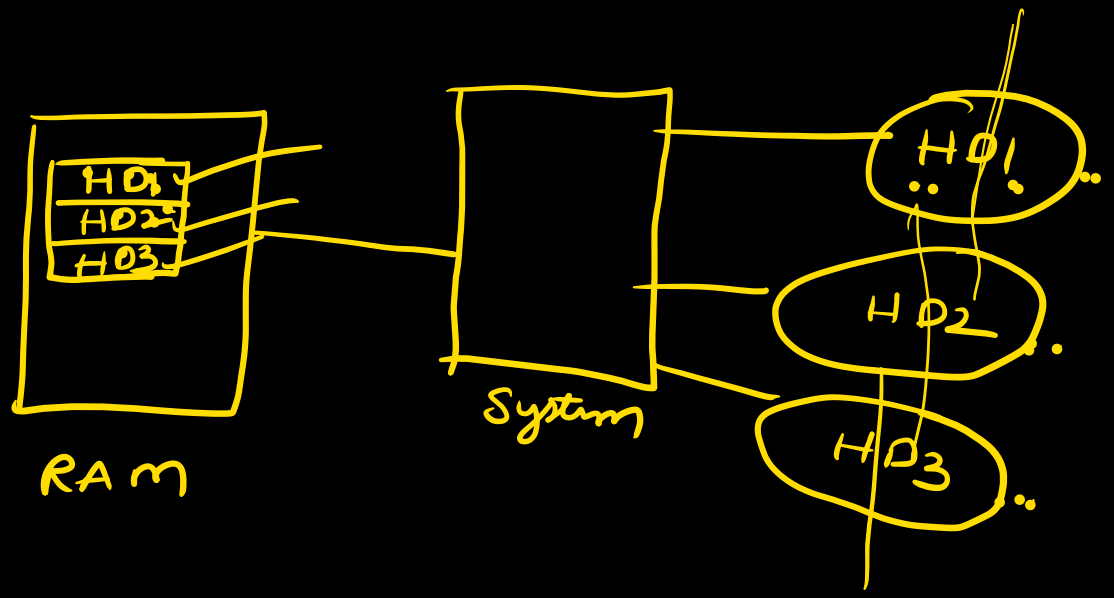
This data structures is loaded at mount time and discarded at dismount time

mount time → when a hard disk is connected

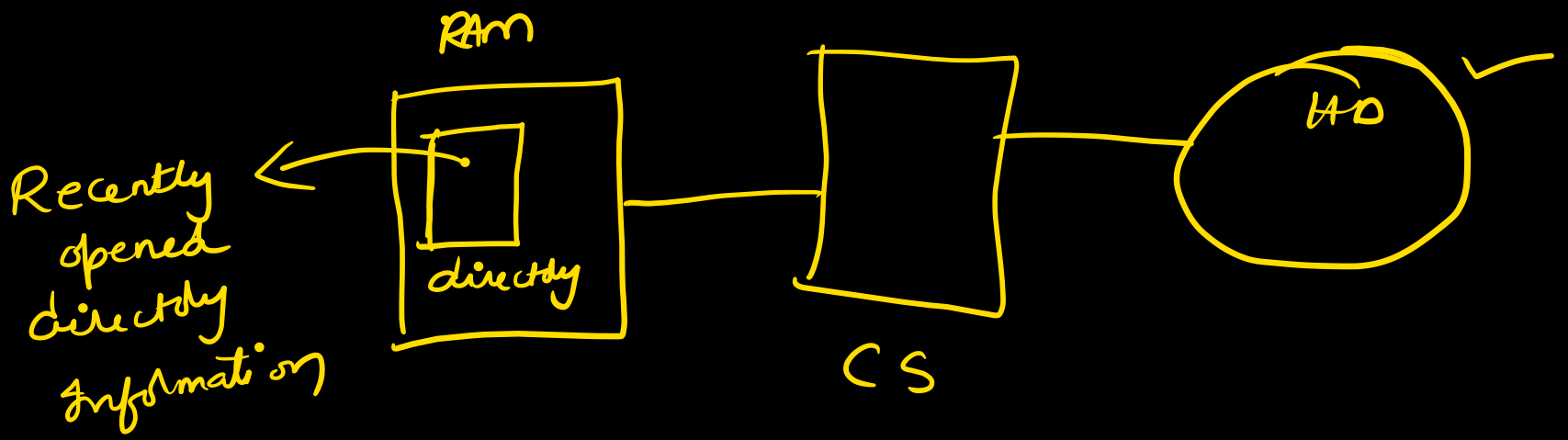
dismount time → when a hard disk is disconnected



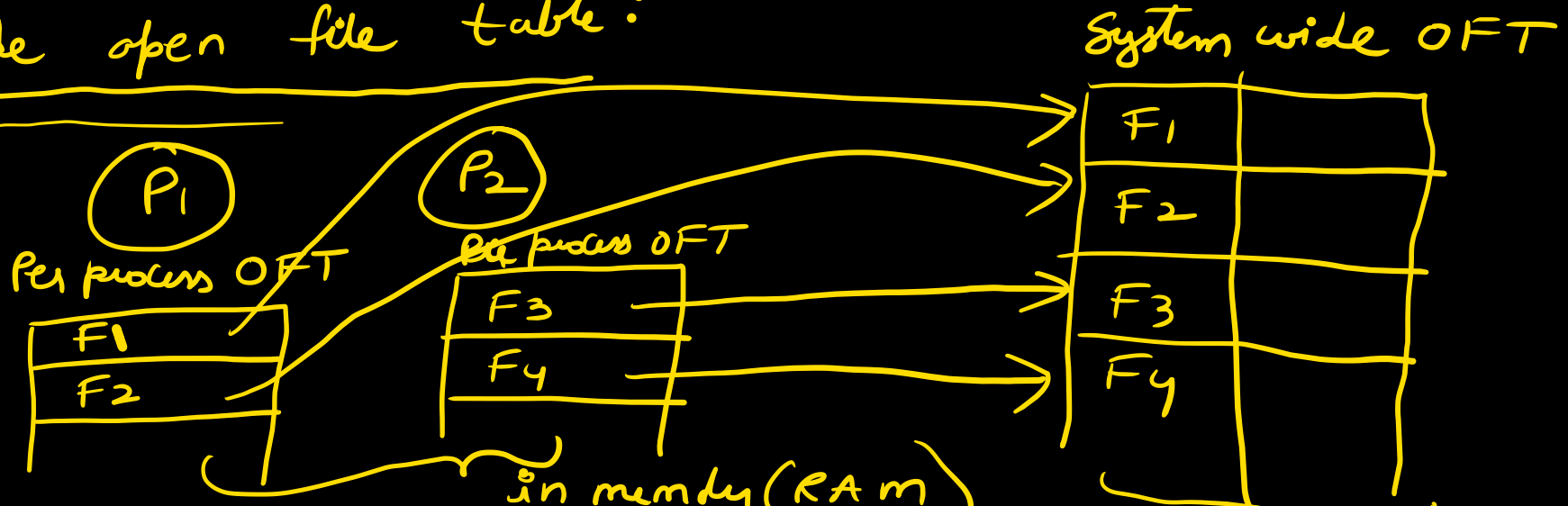
1) In memory mount table: It Contains information about each mounted volume.



2) In memory directory structure cache: It holds the directory information of recently accessed directory.



3) System wide open file table:

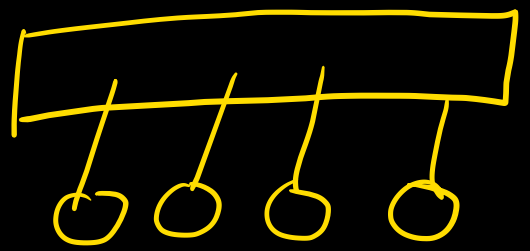


(4) Per process open file table:

It contains the pointer to appropriate entry in the system wide open file table.

Directory implementation:

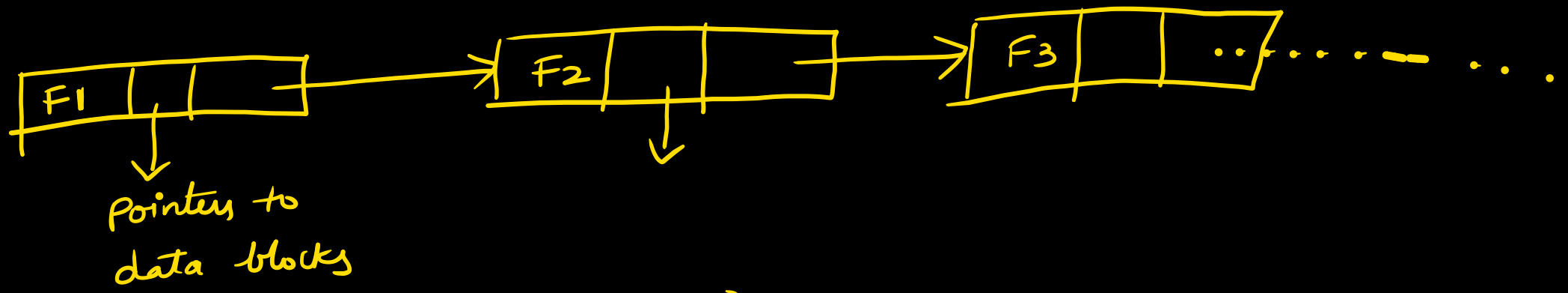
The selection of directory allocation and directory management algorithms significantly affects the performance and reliability of file system.



How to implement?

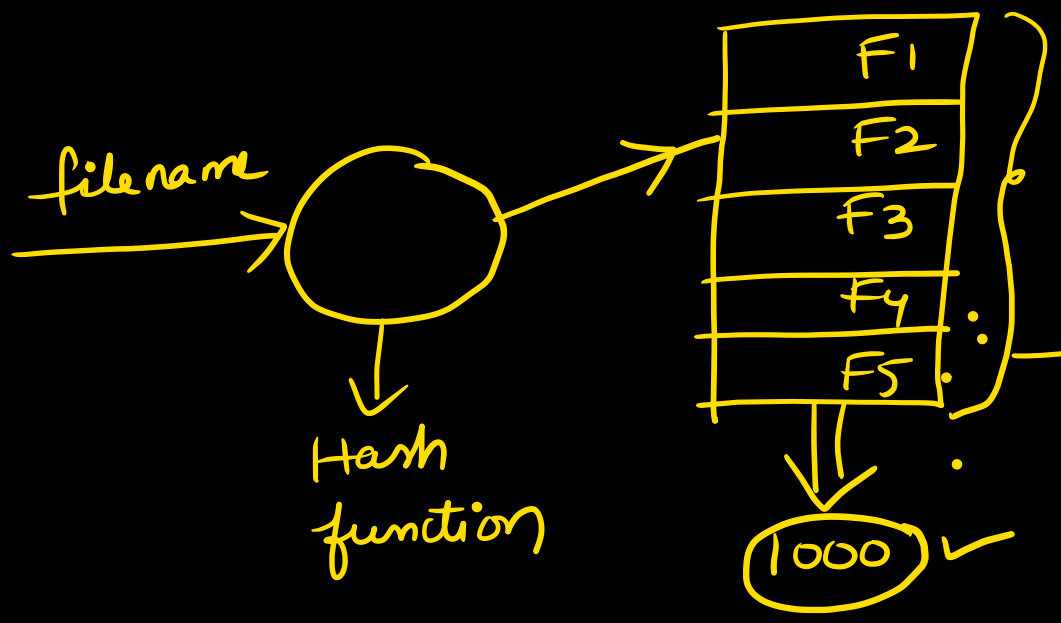
algorithms? Data structures?

1) Linear list :



Problem: linear search → O(n)

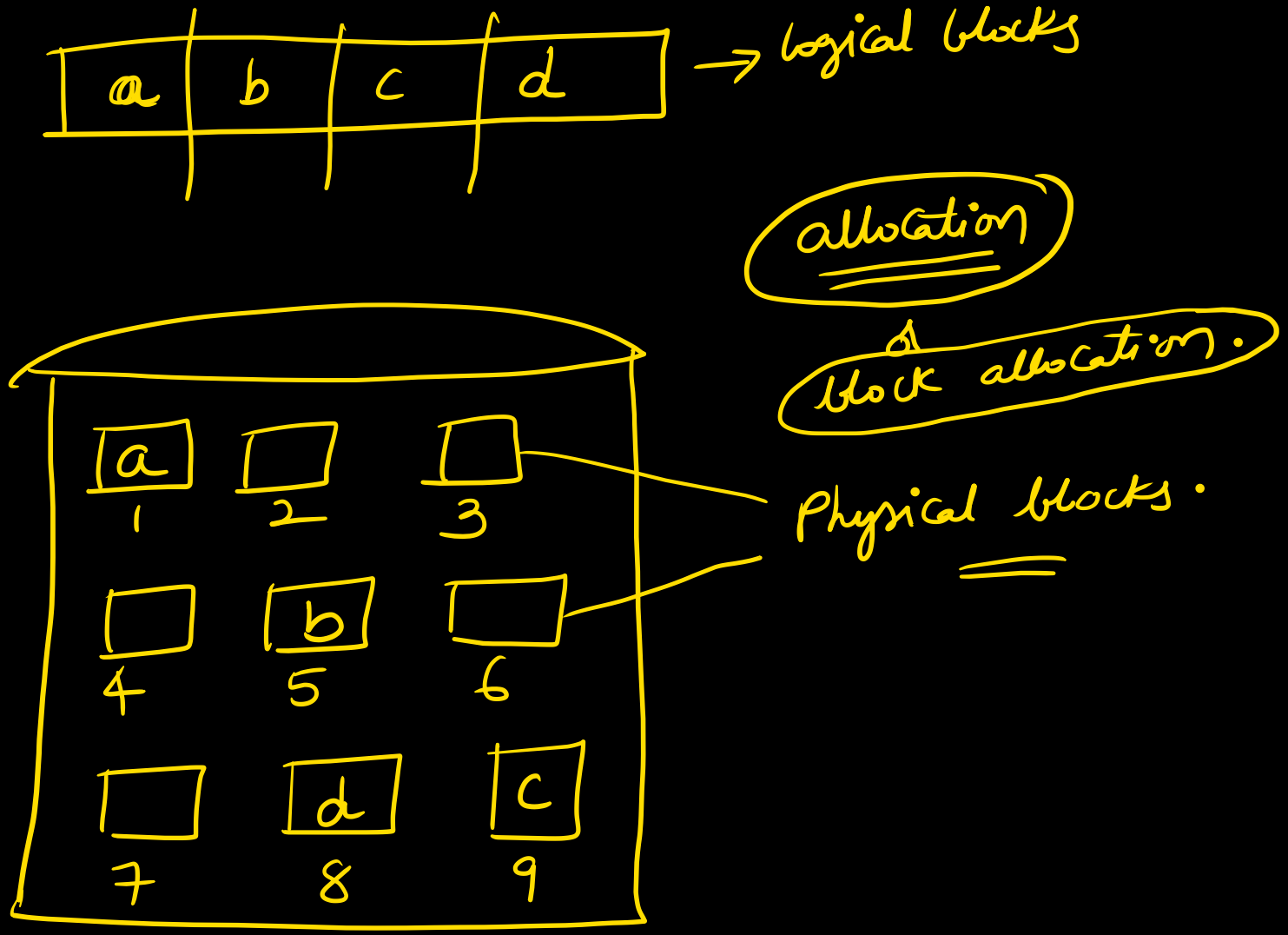
2) Hash table:



Search = O(1)

~~Disadv~~ Disadv:

- Fixed size.
- Collisions.
  - Data structure → we will discuss.

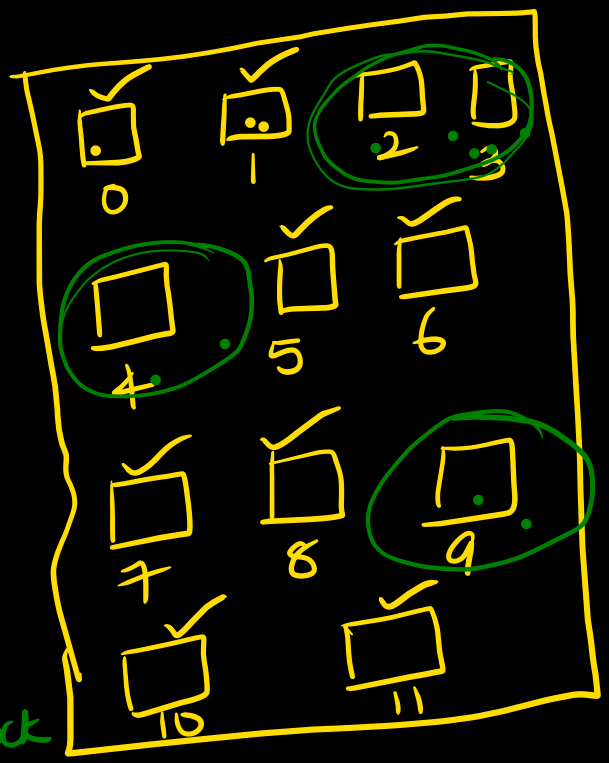

a
b
c
d
logical blocks
allocation
&
block allocation.
a
1
2
3
4
b
5
6
7
d
8
c
9
Physical blocks.

Allocation methods:

How to allocate space to the files so that disk space is utilized efficiently and files can be accessed quickly.

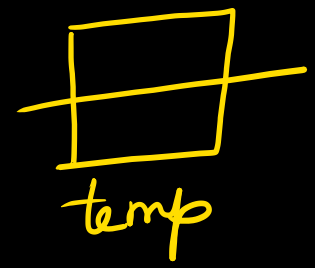
1) Contiguous allocation:

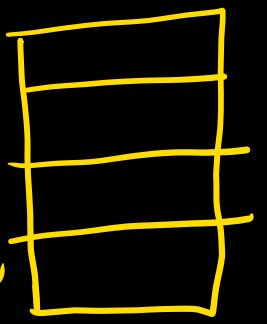
file → 4 DB

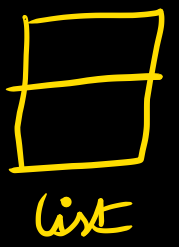


directory

file	start	size
Tem	0	2
tr	5	4
List	10	2







Adv: Simple to implement, read, performance is excellent.

Disadv: fragmentation.

(internal, external)

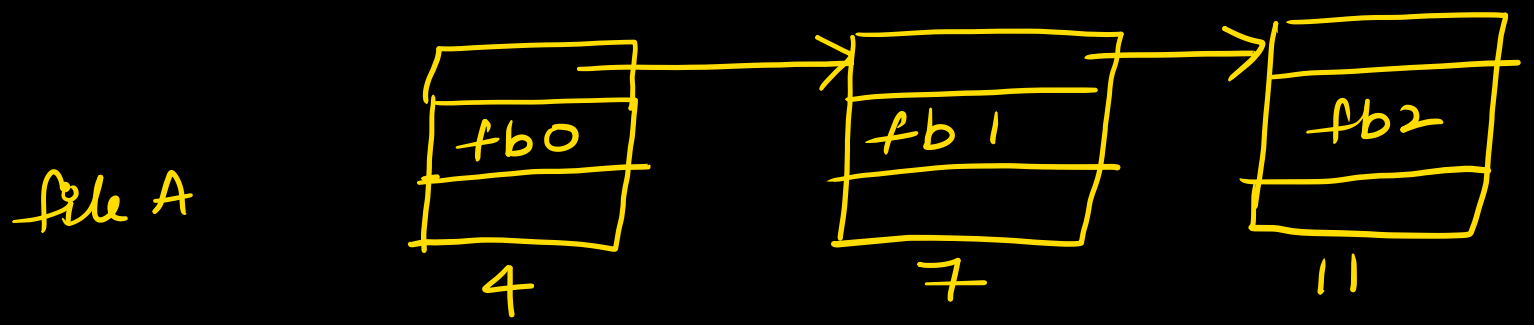
↓

Can't be avoided

Data block

internal fragmentation

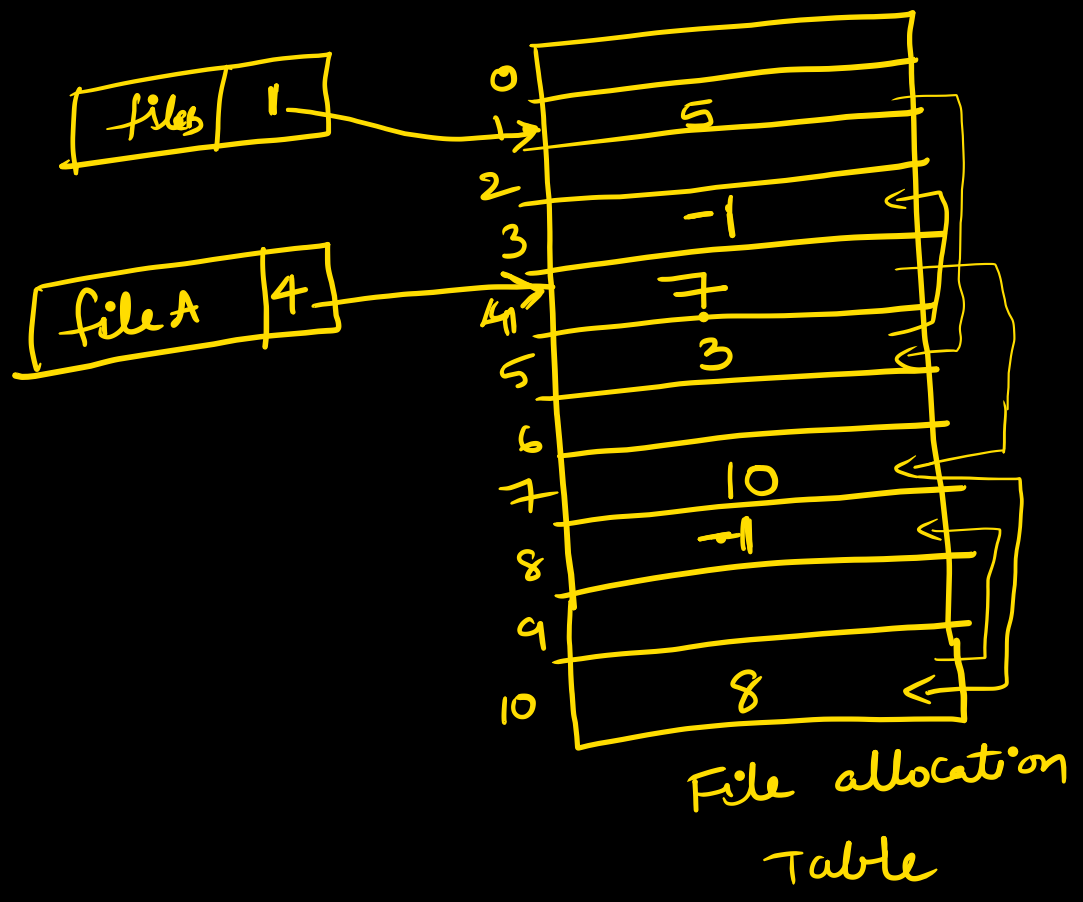
2) Linked list allocation:



adv: no need to be contiguous
no space is wasted bc of external fragmentation

Dis: Random acc is not possible (linear access)
Pointers will take extra space.

3) File allocation table:



File allocation Table

file A → 4, 7, 10, 8

file B → 1 5 3

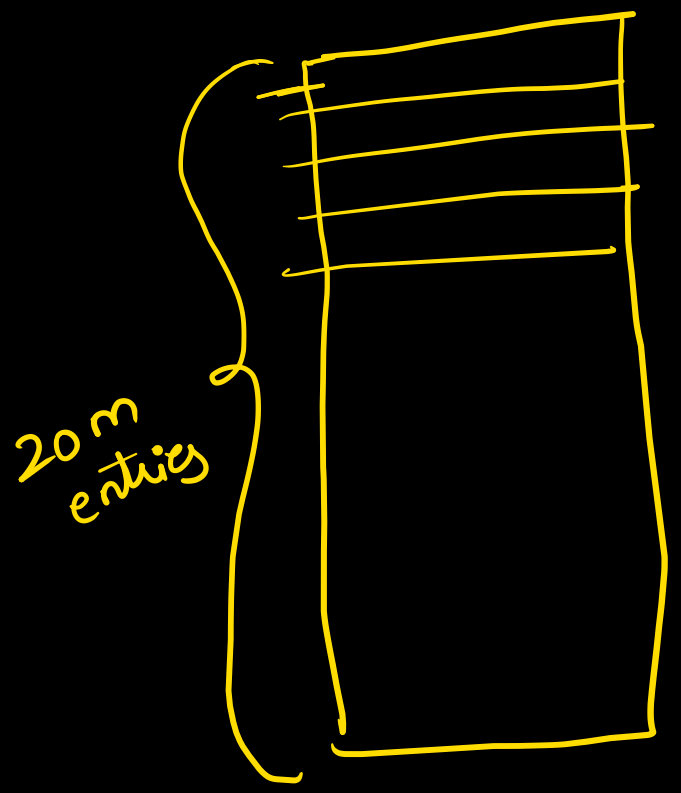
adv: Random access is easier

disadv: Size might be too big that it could consume lot of memory.

Ex: Size of HD = 20GB, Block size = 1KB, FAT entry = 4B.

$$\#\text{ blocks} = \frac{20 \times 2^{30}}{2^{10}} = 20\text{M}$$

$$\text{FAT size} = 20\text{M} \times 4\text{B} = 80\text{MB}.$$



Siz = 80MB ✓

↓

in memory data structure

80MB of RAM

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

FAT Size = no of entries × size of each entry

$$= \frac{(100 \times 10^6)}{10^3} \times 4 = 0.4 \times 10^6 \text{ B}.$$

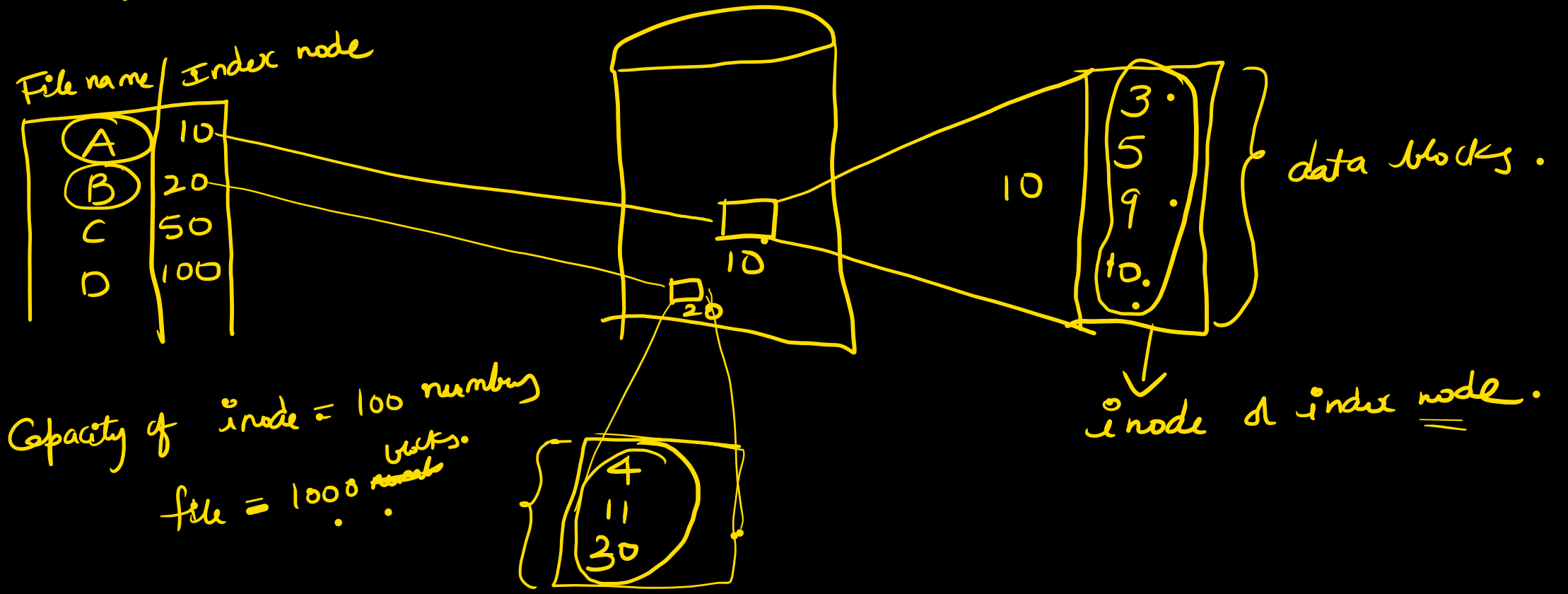
Total Size = 100×10^6.

max File Size = Total − FAT size

$$= (100 - 0.4)(10^6 \text{ B}) = 99.6 \times 10^6 \text{ B}.$$

Indexed allocation:

It brings all pointers together into one location

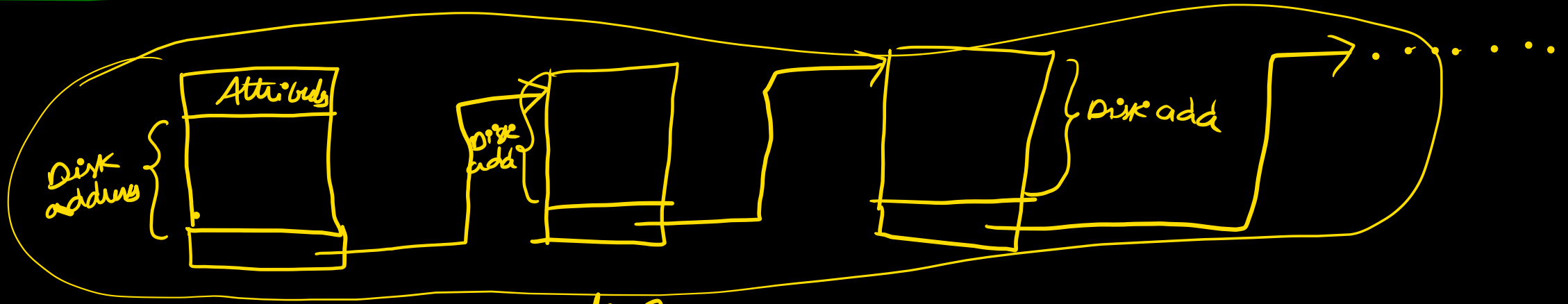


Capacity of inode = 100 numbers

file = 1000 blocks.

If the index block is too small, it will not be able to hold many pointers for a large file.

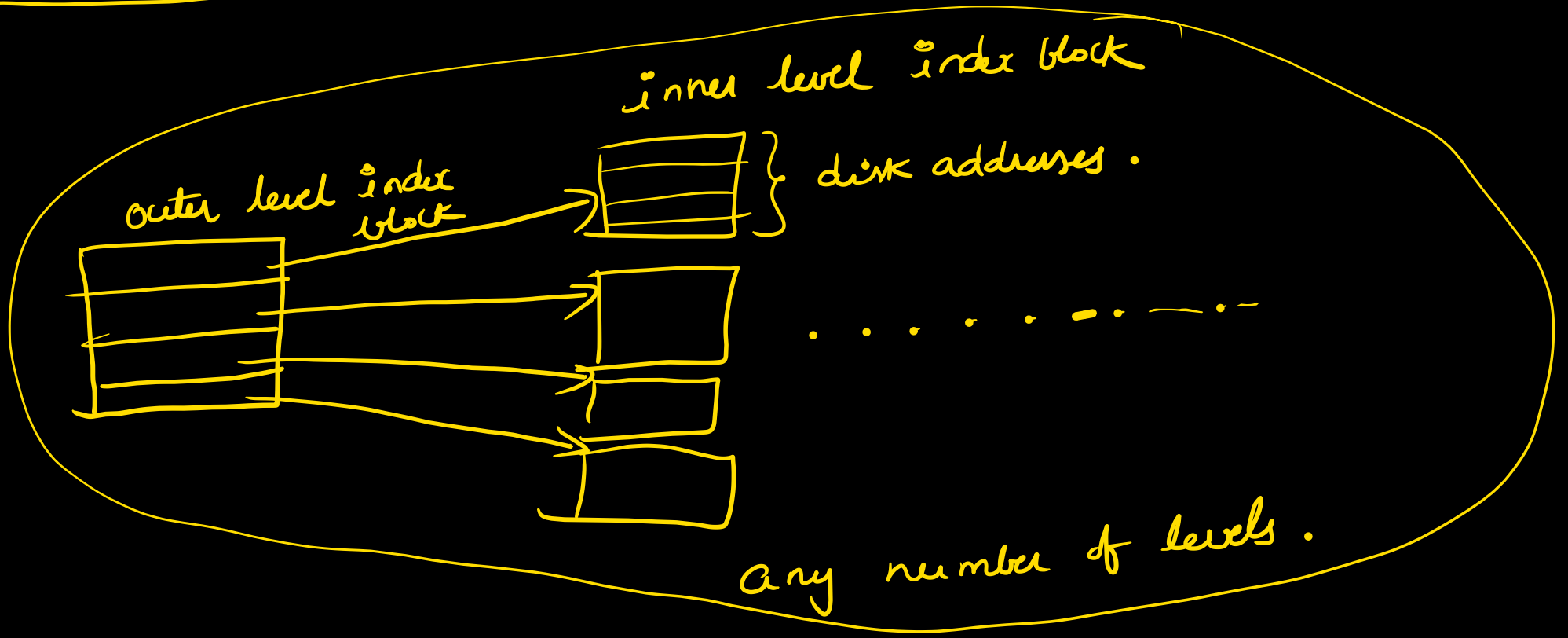
Linked schema: we can link several index blocks



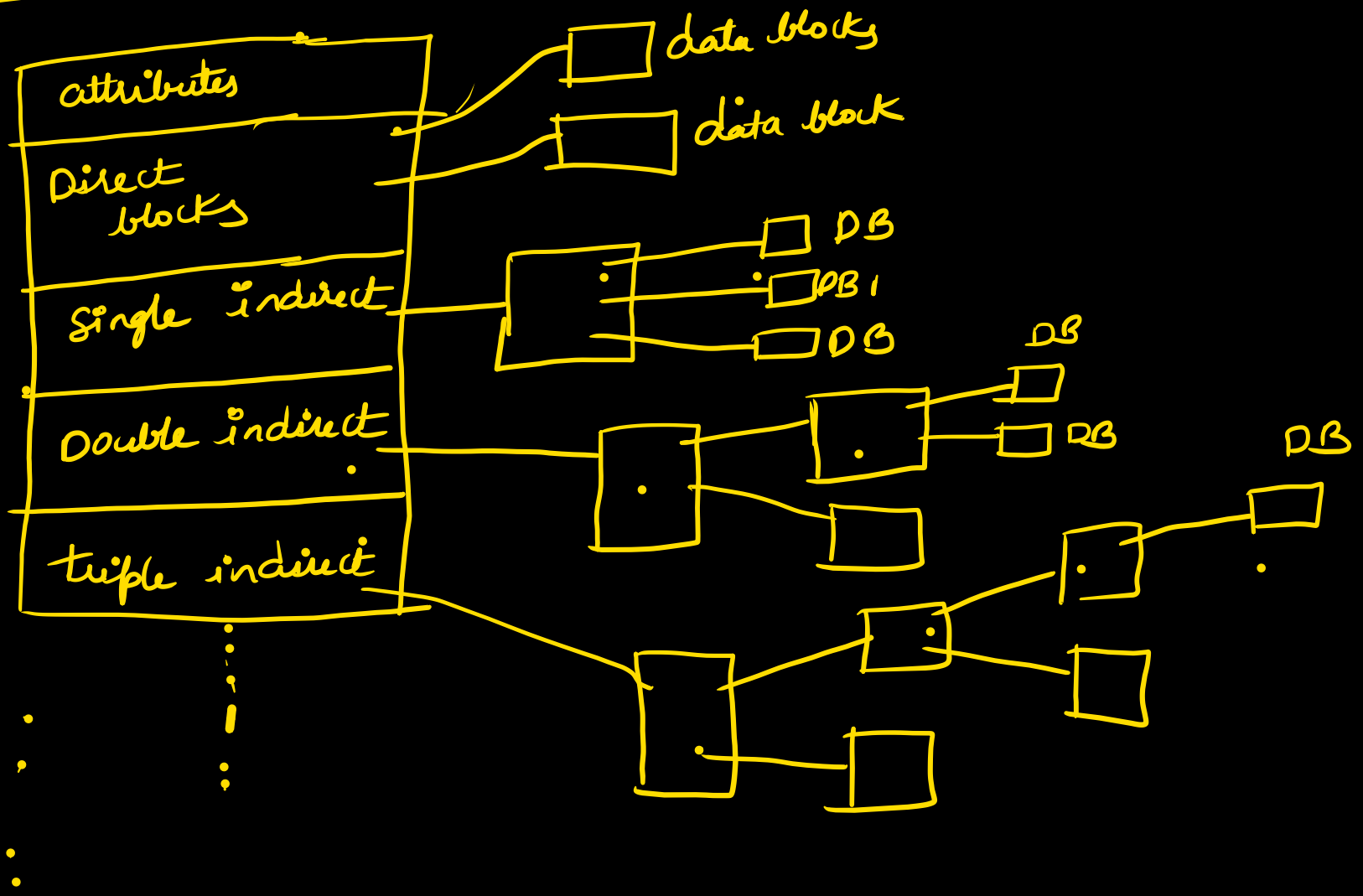
one implementation.

other one is multi-level index.

multi-level index:

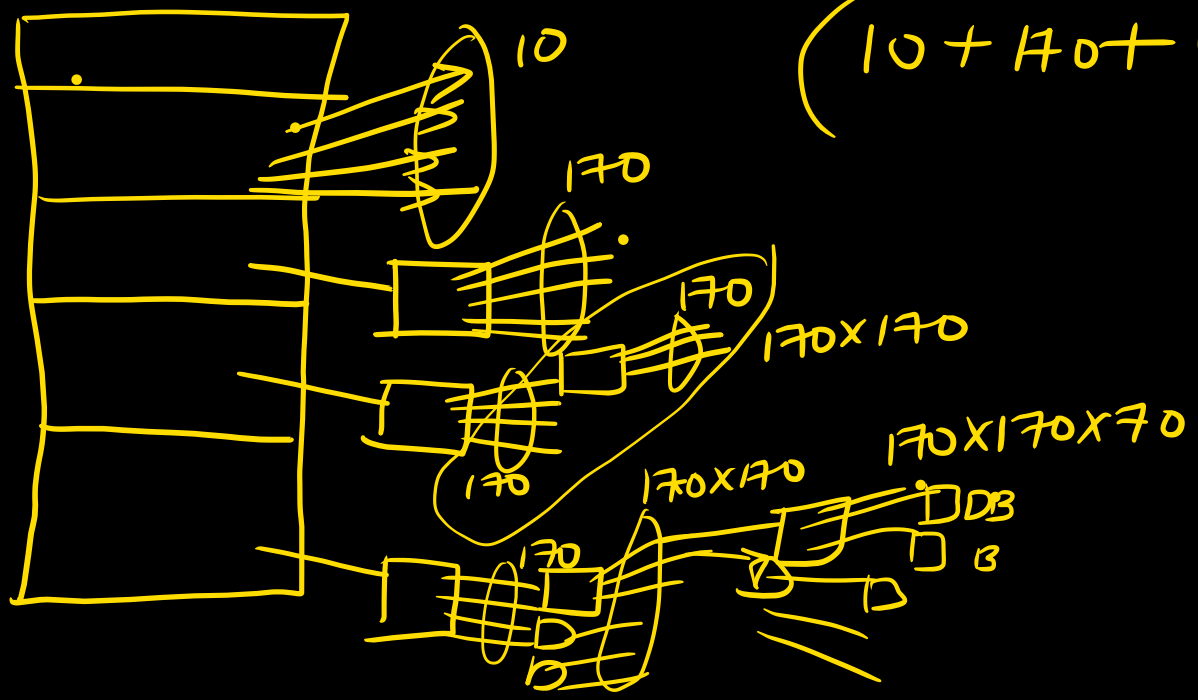


Hybrid schema:



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

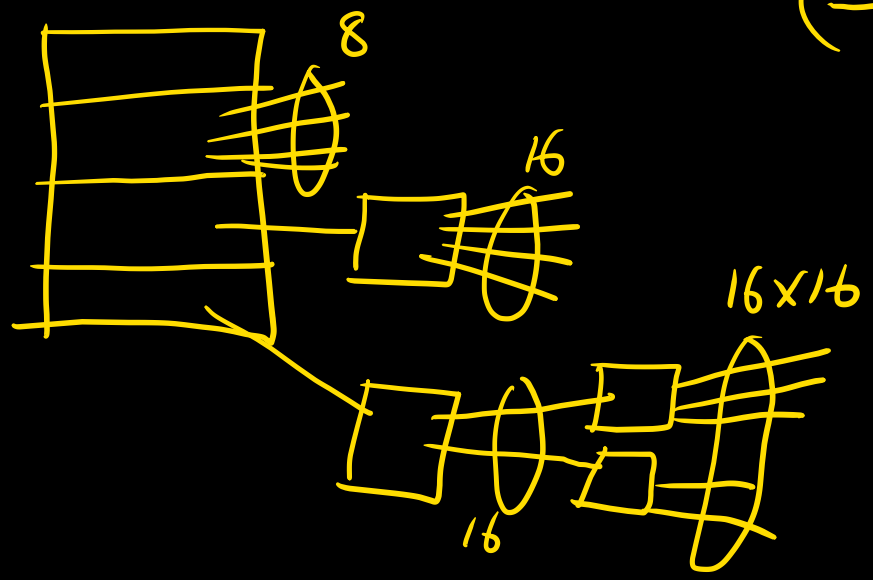
No of pointers in 1 block = $\frac{1024\,B}{6\,B}$ = 170 pointers.



$(10 + 170 + 170X170 + 170X170X170) \times 1024\,B.$

Gate: A file system with 300 GB uses a file descriptor with 8 direct block address. one indirect block address and one doubly indirect block add. The size of each disk block address is 8B. max possible file size is ——? (Size of each disk block = 128 B).

No of pointers = $\frac{128\ B}{8B} = 16$



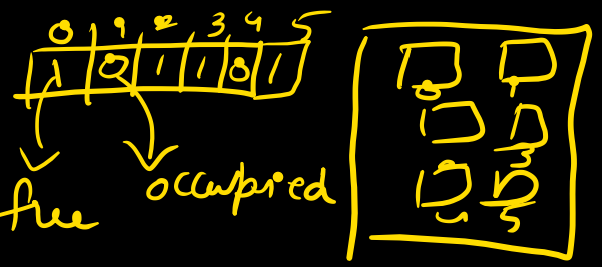
$(8 + 16 + 16 \times 16) \times 128B$

$= 35 KB$.

Free space management:

1) Bit vector / bit map:

Each block is represented by 1 bit. free / occupied



If block is free then bit is '1' else it is '0'

2) Linked list:

We link together all the free blocks.