Mod 5 - Lecture 1:

File Organization

- Deepak Poonia
(IISc Bangalore; *GATE AIR 53 & 67*)

Content of this Lecture:

1. File Organization
2. Practice Questions

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

Instructor:

Deepak Poonia

IISc Bangalore

GATE CSE AIR 53; AIR 67; AIR 206; AIR 256

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

GO CLASSES

GATE 2024
COMPLETE COURSE CS-IT
(1 YEAR + 3 MONTHS)

GATE 2024 COMPLETE COURSE CS - IT

(1 YEAR + 3 MONTHS)

GO CLASSES

GATE 2023
COMPLETE COURSE CS-IT
(6 MONTHS)

GATE 2023 COMPLETE COURSE CS-IT

(6 MONTHS)

GO CLASSES

GATE 2025
COMPLETE COURSE CS-IT
(2 YEARS)

GATE 2025 COMPLETE COURSE CS - IT

(2 YEARS + 3 MONTHS)

2023 Discrete Mathematics

★ ★ ★ ★ ★ 5.0 (62 ratings)

Deepak Poonia (MTech IISc Bangal...

Free

2023 C Programming

★ ★ ★ ★ ★ 5.0 (59 ratings)

Sachin Mittal (MTech IISc Bangalor...

Free

2023 Discrete Mathematics

Learn, Understand, Discuss. "GO" for the Best.

★★★★★ 5.0 (62 ratings)

5997 Learners Enrolled

Language: English

Instructors: Deepak Poonia (MTech IISc Bangalore, GATE CSE AIR 53; 67)

FREE

On “GATE-Overflow ” Website

Join **GO+ GO Classes Combined Test Series** for BEST quality tests, matching GATE CSE Level:

Visit www.gateoverflow.in website to join Test Series.

1. **Quality Questions:** No Ambiguity in Questions, All Well-framed questions.
2. Correct, **Detailed Explanation**, Covering Variations of questions.
3. **Video Solutions.**

NOTE :

Complete Discrete Mathematics & Complete C-Programming

Courses, by GO Classes, are **FREE** for ALL learners.

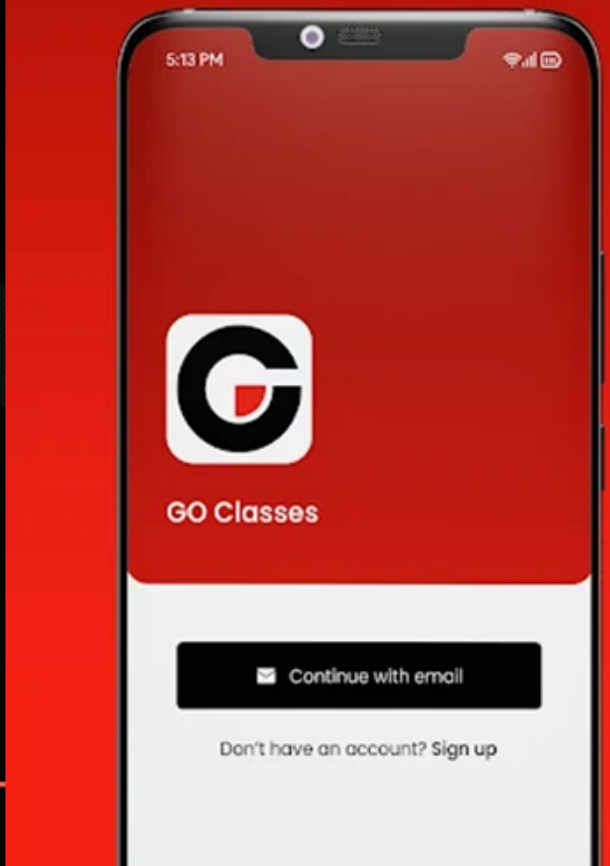
Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

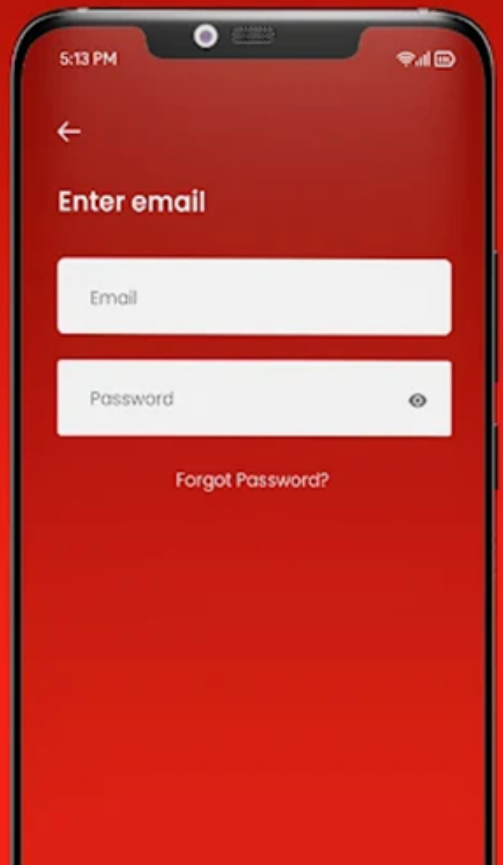
Download the GO Classes Android App:

https://play.google.com/store/apps/details?id=com.goclasses.courses

Search "GO Classes" on Play Store.

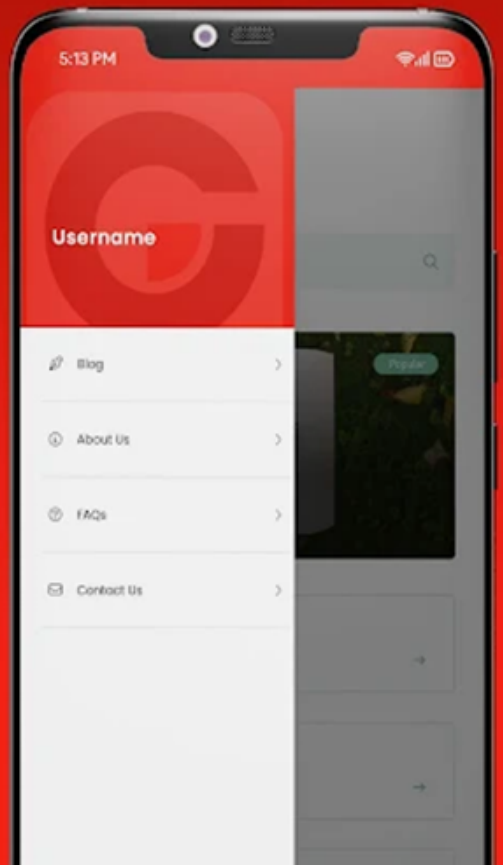


Welcome to
GO Classes
Learn at your own pace



Learn from
Top educators
Anytime, anywhere!

Easy to use
Navigation Panel
with all essentials



We are on Telegram. Contact Us for any help.

Join GO Classes Resources, Notes, Content, information Telegram Channel:

Public Username: GOCLASSES_CSE

Join GO Classes **Doubt Discussion** Telegram Group :

Username: GATECSE_Goclasses

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: gateoverflow_cse

Disk : (Magnetic Disk)

- Platters, Surface
- tracks, Cylinder
- Sectors
- Sector Size

} Properties of Disk

Disk : (Magnetic Disk)

- Platters, Surface
- tracks, Cylinder
- Sectors
- Sector Size

DBMS Decides the Block Size.

Block : Some Consecutive sectors within a Track

Chapter 1:

File Organization

Next Topic:

File Organization Storing a Database on Disk

NOTE:

In this chapter we will study "File Organization", In next chapter we will study Indexing.

DON'T CONFUSE/Mix File Organization with Indexing.

NOTE:

"File Organization" : → Actual Data

It tells you HOW Database is Stored in the Disk.

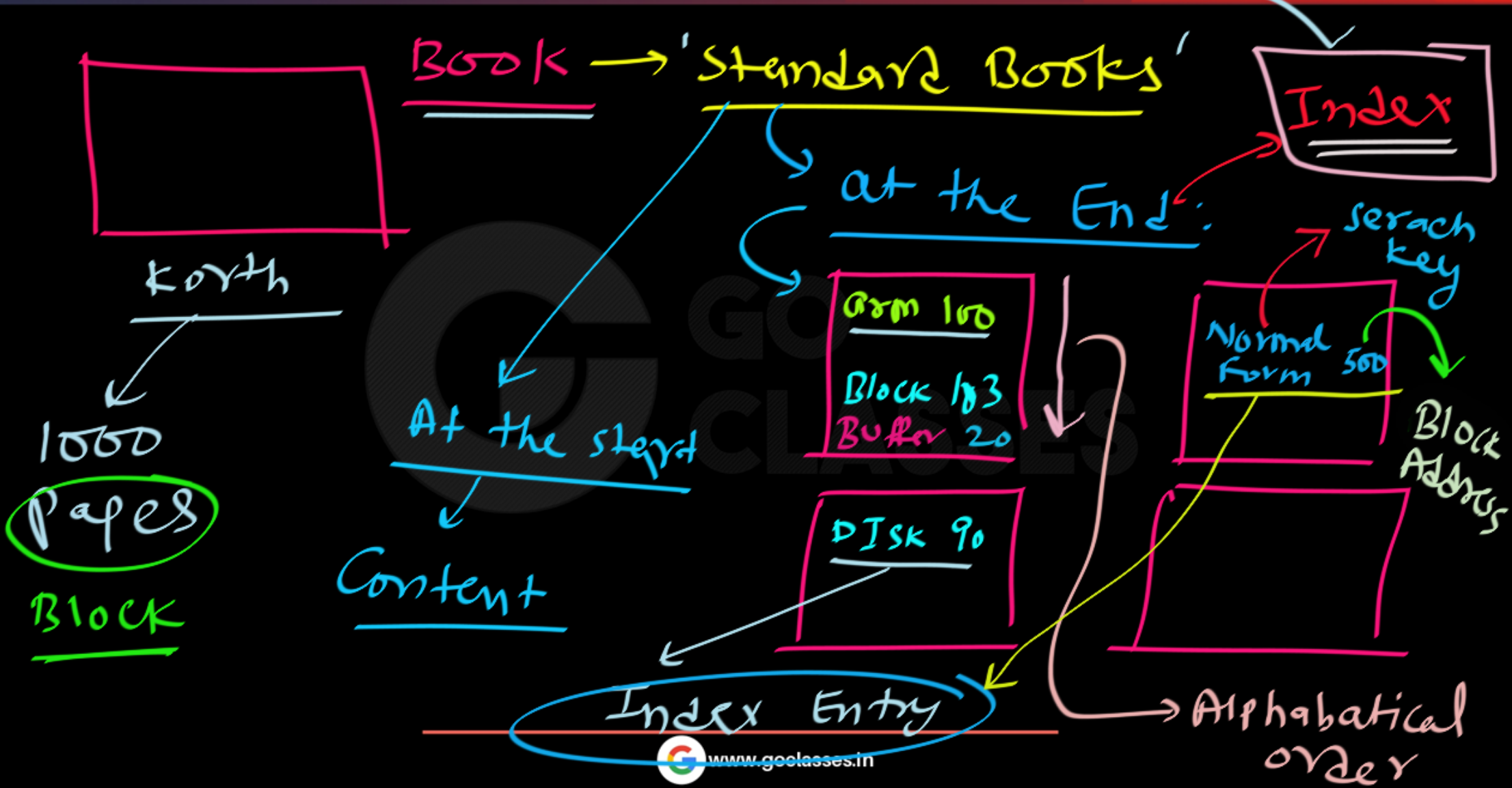
Indexing:

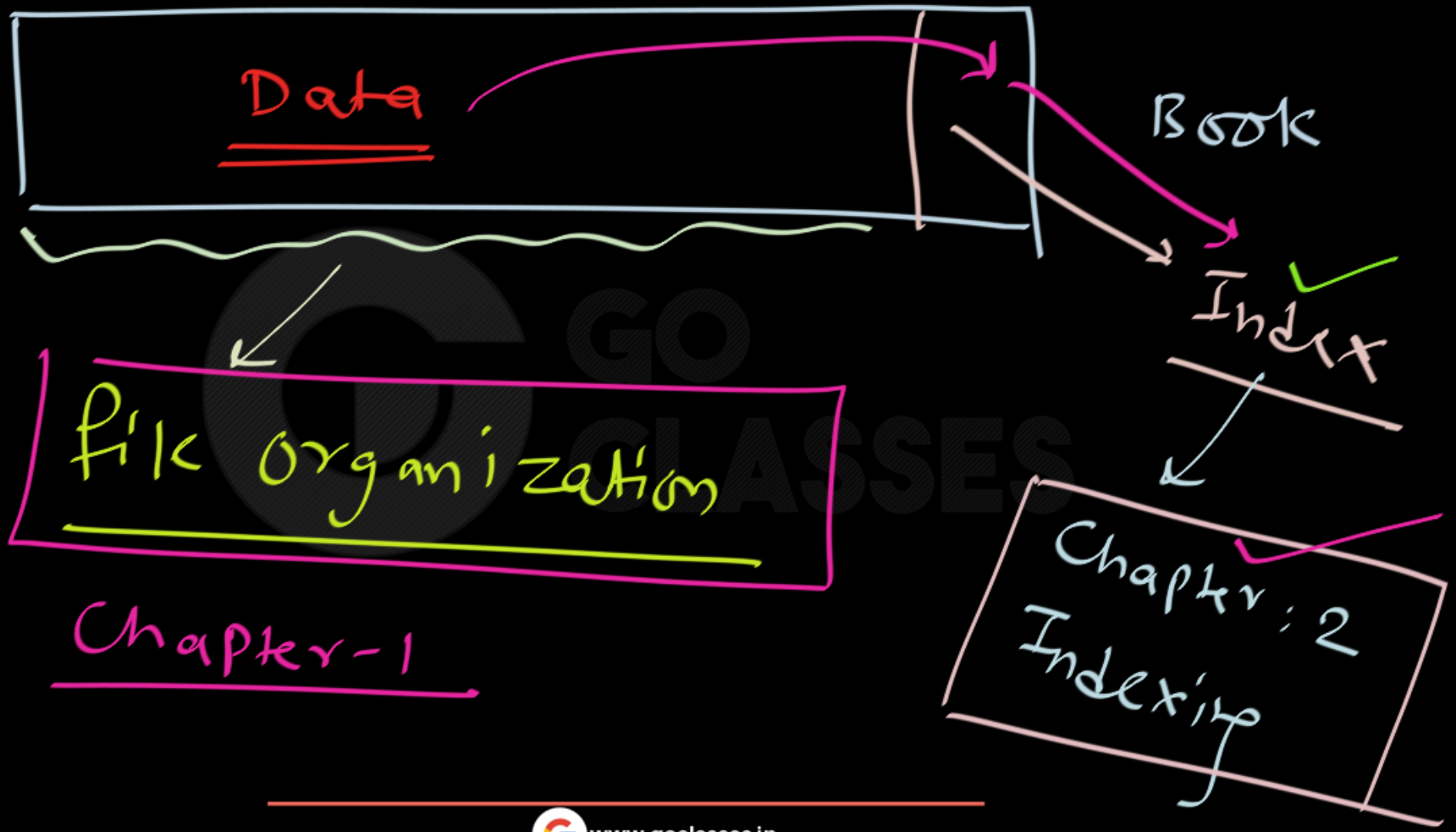
It ONLY simplifies our data access.

Analogy:

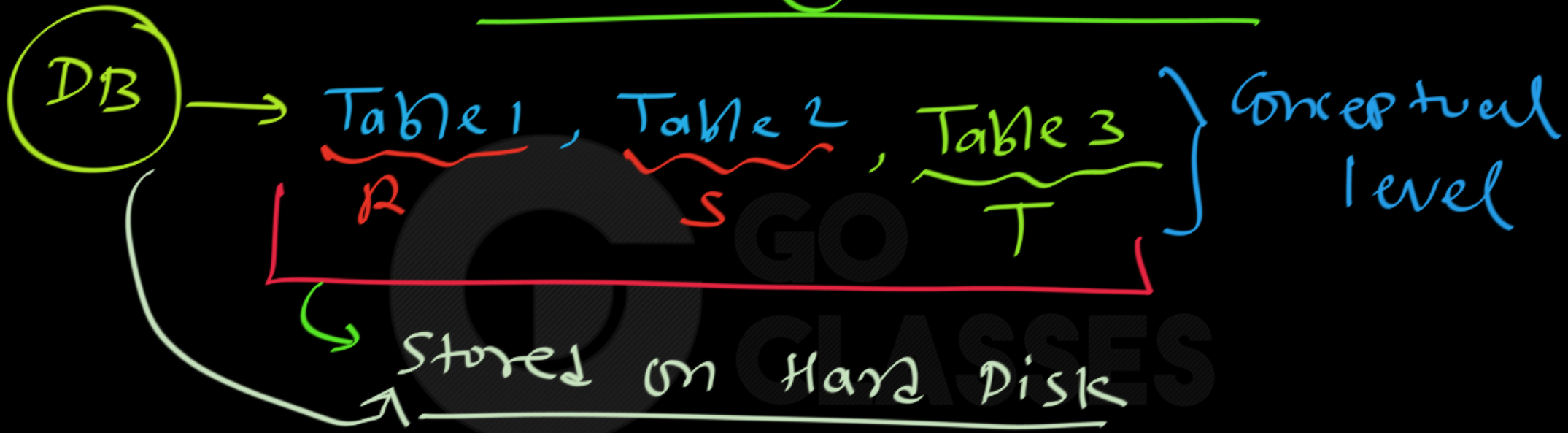
File Organization: Actual BOOK

Indexing: The Last 2-3 pages of the book, called Index.


Book → 'Standard Books'
Index
at the End:
serach key
Normal Form 500
Block Address
Block No 3
Buffer 20
Disk 90
Korth
1000
Pages
Block
At the start
Content
Index Entry
Alphabatical order
www.goclasses.in


Data
Book
Index
File Organization
Chapter-1
Chapter: 2
Indexing

file Organization :

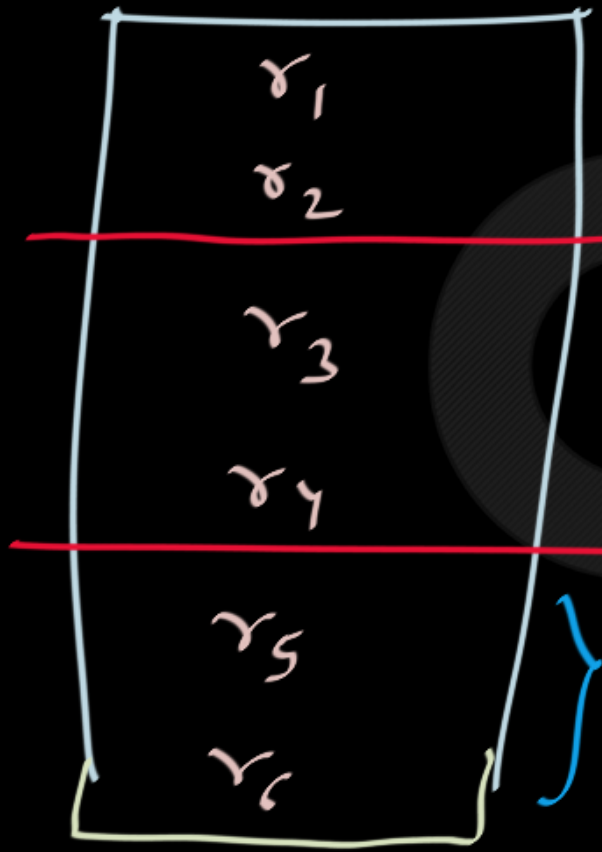


R :⟶ Simple file 1

S ⟶ " " 2

T ⟶ Simple file 3

Relation R ⟶ Stored on Disk as a Simple file

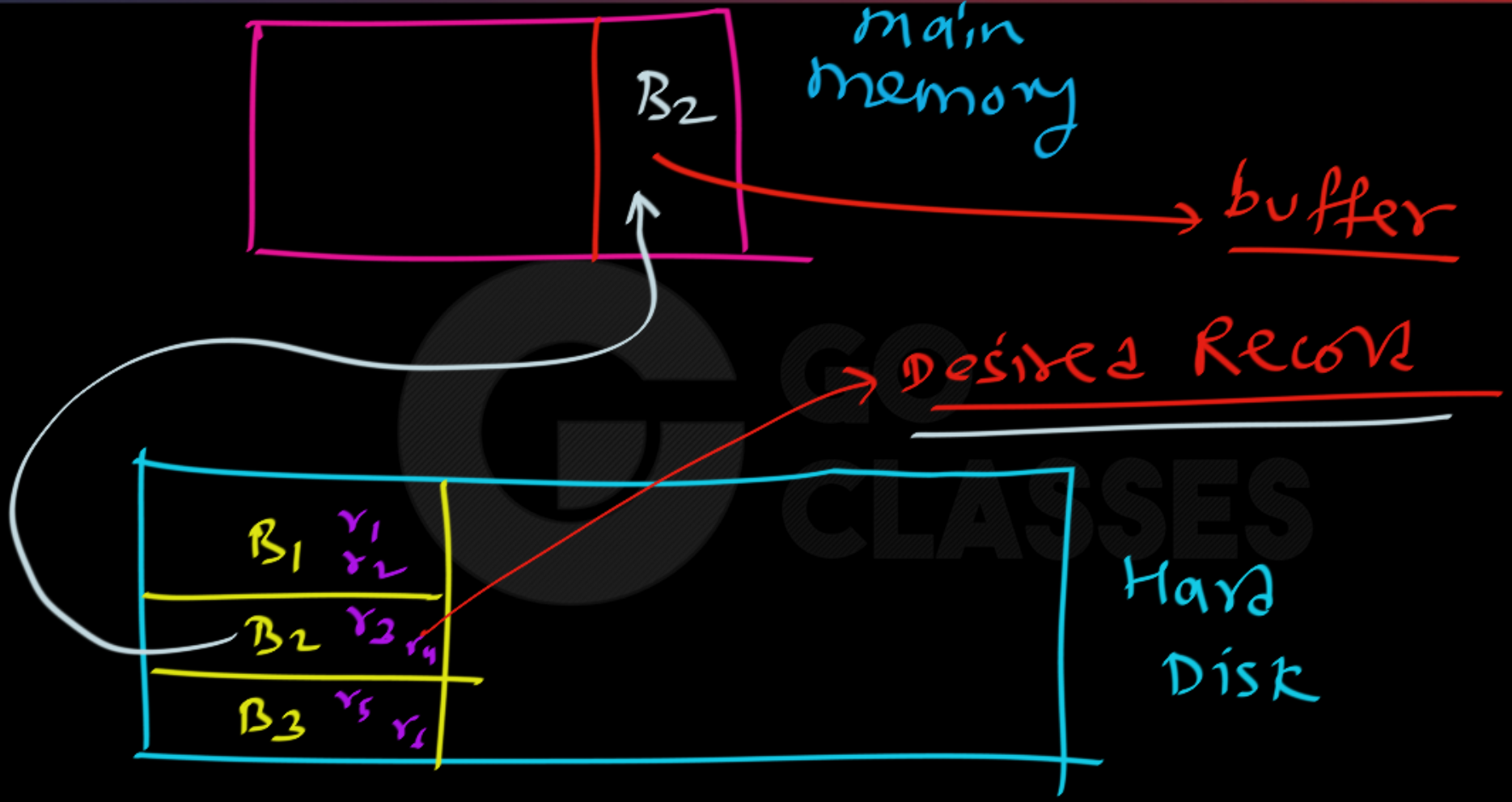


Sequence of Consecutive Block

Logical Concept

Block: Some Consecutive Sectors

Block size fixes by DBMS


Main memory
B_2
buffer
Desired Record
B_1 r_1 r_2
B_2 r_3 r_4
B_3 r_5 r_6
Hard Disk

Hard Disk :

Sector

Smallest Unit of Data Access

DBMS

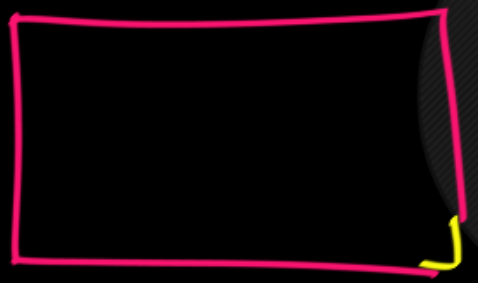
(Let) Block Size = 3 sectors

Smallest Unit of Data Access for DBMS

Although a database system provides a high-level view of data, ultimately data have to be stored as bits on one or more storage devices. A vast majority of databases today store data on magnetic disk and fetch data into main memory for processing.

The physical characteristics of storage devices play a major role in the way data are stored, in particular because access to a random piece of data on disk is much slower than main memory access: Disk access takes tens of milliseconds, whereas memory access takes a tenth of a microsecond.

Whenever a certain portion of the data is needed, it must be located on disk, copied to main memory for processing, and then rewritten to the disk if the data is changed.



a Block → main memory

"Access Cost"

Blocks needed to be Transferred From disk to main mem.

Disk stores Everything (programs, OS, settings, movies etc)...

But we care about HOW does Disk store a Database??

We now turn to the matter of how disks are used store databases.

We have studied Hard Disk.

Remember these??

Sector

Block

Track

Cylinder

Surface, Platter etc

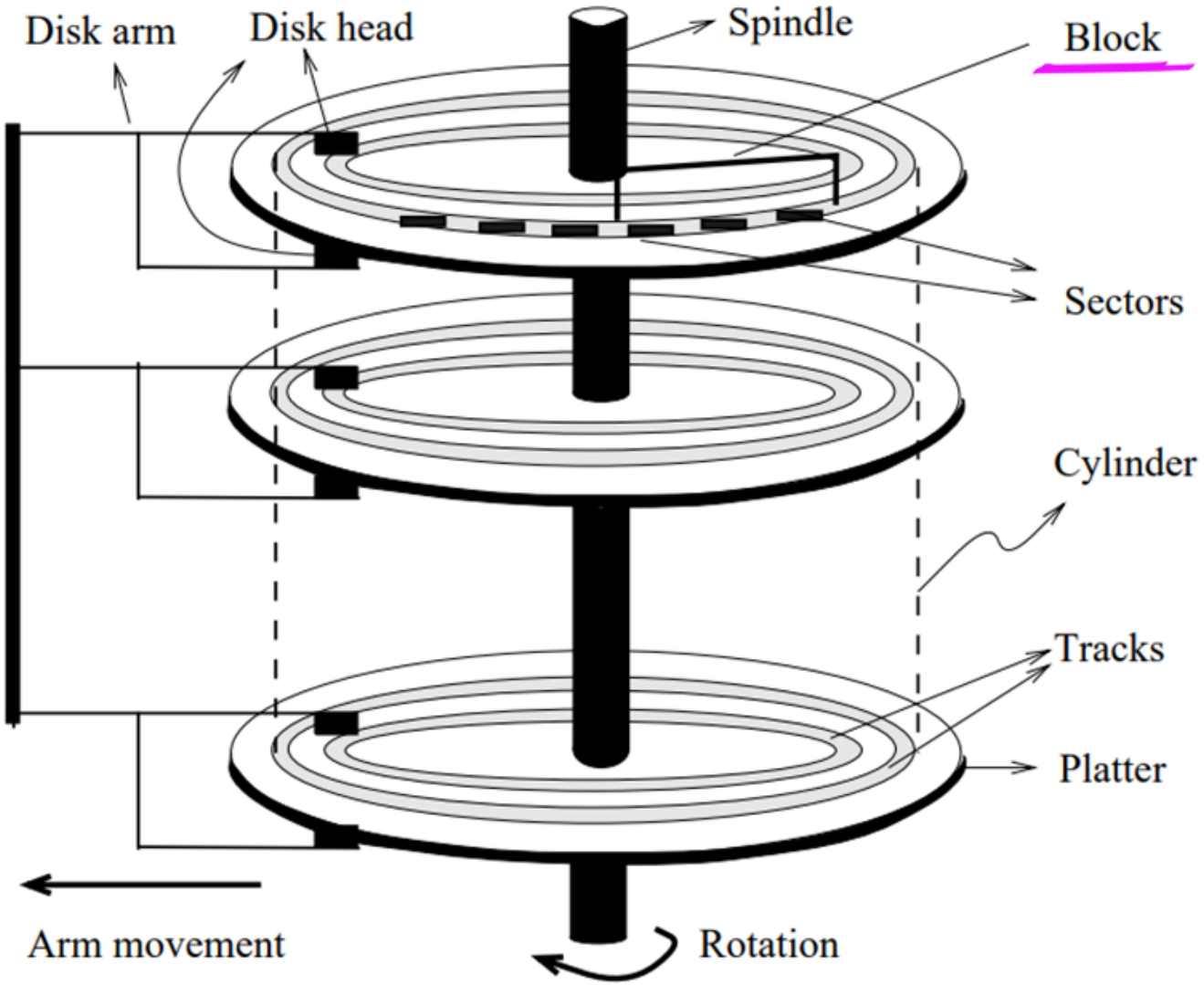


Figure 7.2 Structure of a Disk

- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes

"In General:"

Sector is the Smallest Unit of Data to access (Read Or Write) in a Disk.

BUT for DBMS (or DBMS s/w), Block is the smallest unit of data to access...

For DBMS(or DBMS software) :

Block is the Smallest Unit of Data to access (Read Or Write)

Block Size is set by DBMS. Block Size is NOT a property of the Hard Disk.

For DBMS(or DBMS software) :

Block is the Smallest Unit of Data to access (Read Or Write)

For Databases, Data is stored on disk in units called disk blocks. A disk block is a contiguous sequence of bytes and is the unit in which data is written to a disk and read from a disk.

Block is also called Page:

(NOT same as the page in paging)

A block is a logical unit consisting of a fixed number of contiguous sectors. Data are transferred between disk and main memory in units of blocks.

The term page is often used to refer to blocks.
But we will stick to term Block.

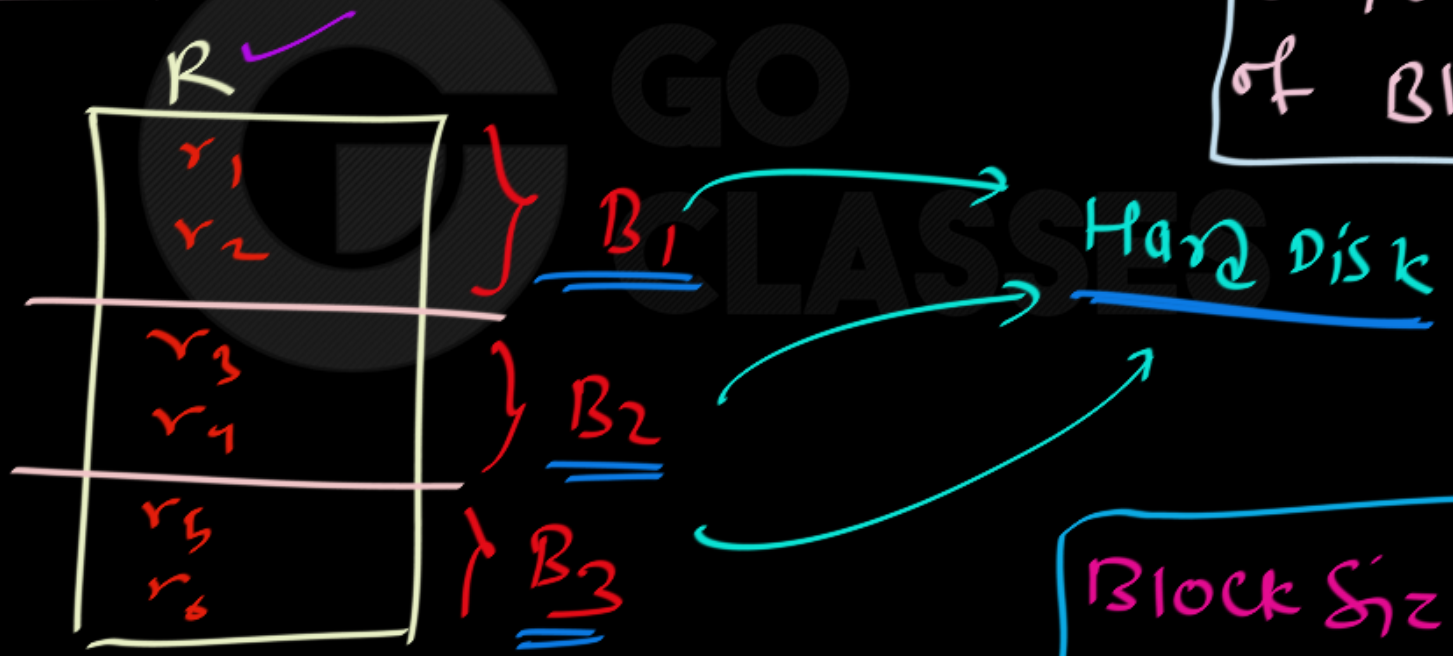
"Block" ≡ page

NOT the page from OS paging Concept

Some sectors

File organization:

One Relation → stored as → one file ⇒ Some sequence of Blocks



Block Size ≥ record size

We now turn to the matter of how disks are used store databases:

Databases are stored physically as files of records, which are typically stored on magnetic disks.

One database relation is stored as one file.

A database is mapped into a number of different files.

These files reside permanently on disks.

A file is organized logically as a sequence of records.

These records are mapped onto disk blocks.

Exercise 7.5: Raghu RamaKrishnan DBMS Book

Consider a disk with a sector size of 512 bytes, 2,000 tracks per surface, 50 sectors per track, 5 double-sided platters, average seek time of 10 msec.

1. What is the capacity of a track in bytes?
2. Give examples of valid block sizes. Is 256 bytes a valid block size? 2,048? 51,200?

Exercise 7.5: Raghu RamaKrishnan DBMS Book

#Tracks = 2000 × 10 = 20000

Consider a disk with a sector size of 512 bytes, 2,000 tracks per surface, 50 sectors per track, 5 double-sided platters, average seek time of 10 msec.

1. What is the capacity of a track in bytes? ⇒ 50 × 512 B
2. Give examples of valid block sizes. Is 256 bytes a valid block size? 2,048? 51,200?

✓ 4 sectors

✗

✗ Not multiple of Sector size

Block:

Consecutive Sectors within a Track

1. bytes/track = bytes/sector * sectors/track = 512 * 50 = 25K
   bytes/surface = bytes/track * tracks/surface = 25K * 2000 = 50, 000K
   bytes/disk = bytes/surface * surfaces/disk = 50, 000K * 5 * 2 = 500, 000K
2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 is. 51200 is not a valid block size in this case because block size cannot exceed the size of a track, which is 25600 bytes.

Each file is also logically partitioned into fixed-length storage units called blocks, which are the units of both storage allocation and data transfer.

A block may contain several records.

No record is larger than a block.

Record ≡ Tuple in a Relation

File ≡ Relation

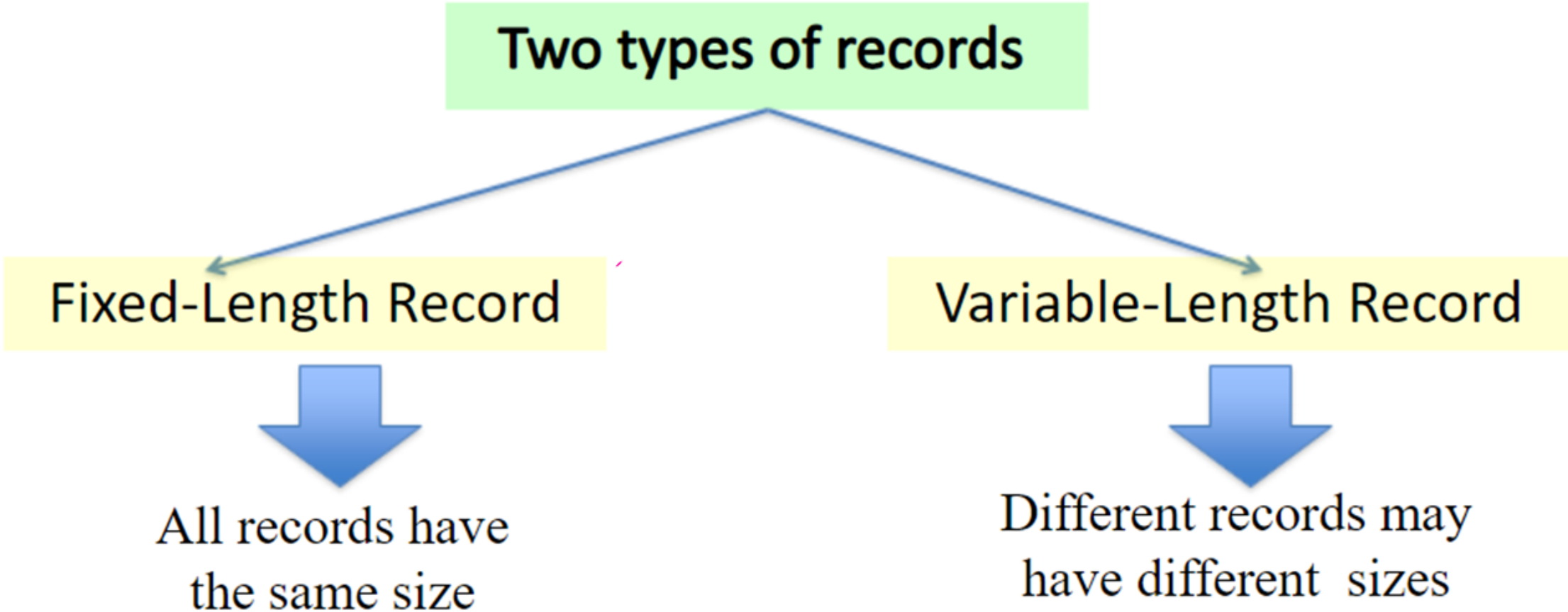
Next Topic:

File Organization

Types of Records

(Record Format)

How Records are Stored on Disk



How Records are Stored on Disk

Two types of records

Fixed-Length Record

All records have the same size

Variable-Length Record

Different records may have different sizes

Check the record's fields

If all fixed size → Fixed-Length record

if any field is variable size → Variable-Length record

Char(30) — Exactly 30 Characters (Bytes)

varchar(30) → At most 30 Character (Bytes)

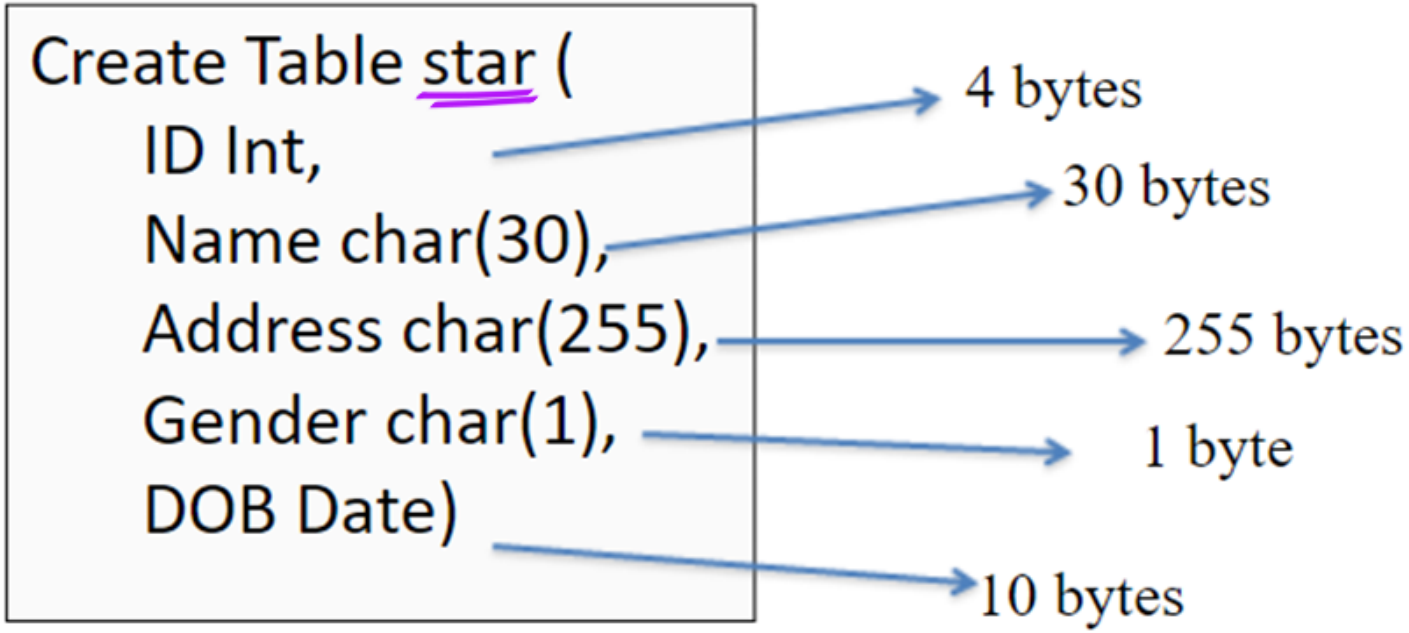
Variable Character Length

Fixed-Length Record Example

```
Create Table star (
        ID Int,
        Name char(30),
        Address char(255),
        Gender char(1),
        DOB Date)
```

ID	name	address	gender	birth date

Fixed-Length Record Example

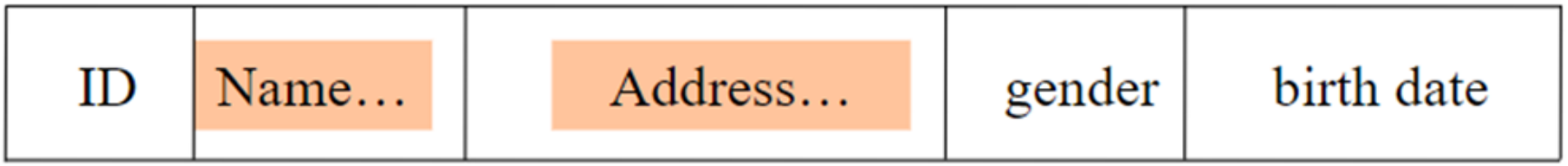


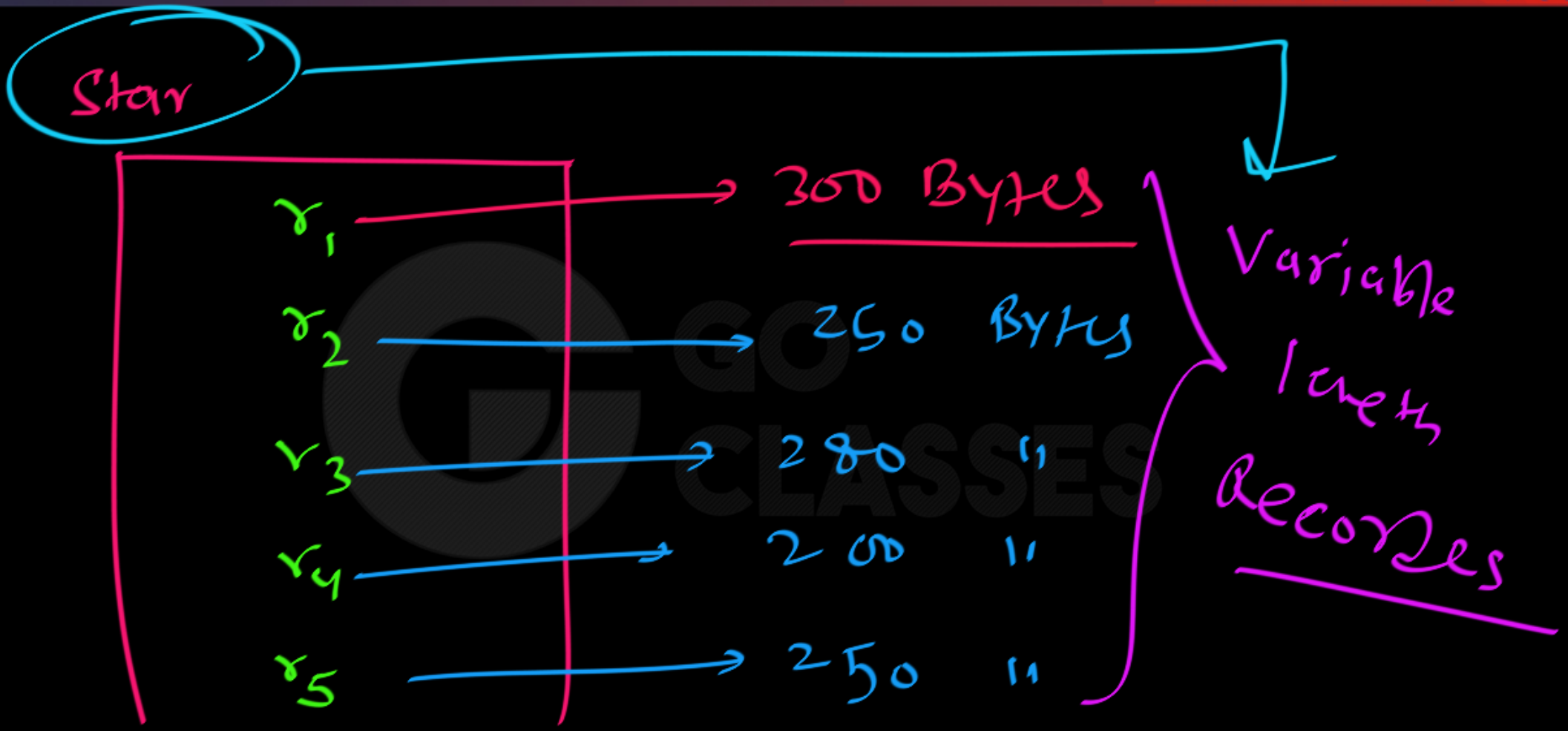
ID	name	address	gender	birth date

Variable-Length Record Example

```
Create Table star (
    ID Int,
    Name varchar2 (30),
    Address varchar2(255),
    Gender char(1),
    DOB Date)
```

Variable length and at most 255 bytes




Star
r_1
r_2
r_3
r_4
r_5
300 Bytes
250 Bytes
280 "
200 "
250 "
Variable length Records

If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records.

If different records in the file have different sizes, the file is said to be made up of variable-length records.

file Contains fixes length records

```
struct employee{
    char name[30];
    char ssn[9];
    int salary;
    int job_code;
    char department[20];
} ;
```

4B

Record size:

30 + 9 + 4 + 4 + 20 =

67 Bytes

```
struct employee{
 varchar name[30];
    char ssn[9];
    int salary;
    int job_code;
    char department[20];
} ;
```

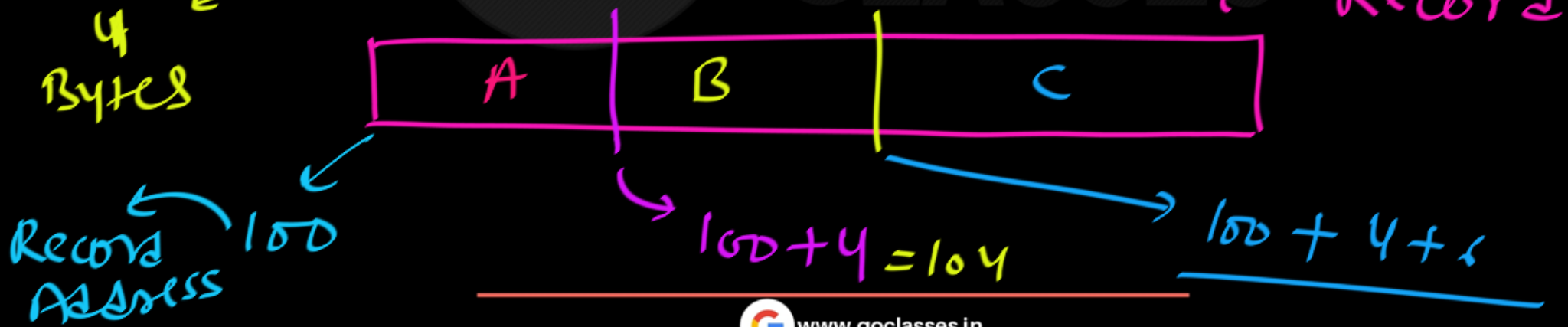
The file records are of the same record type, but one or more of the fields are of varying size (**variable-length fields**). For example, the Name field of EMPLOYEE can be a variable-length field.

How to find a particular field value inside a record

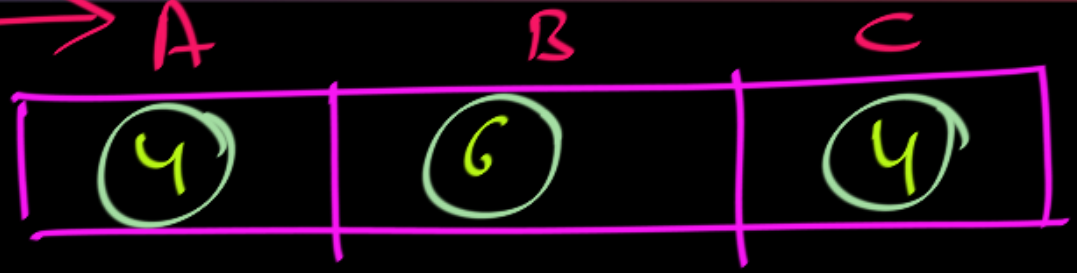
How to find a particular field value

inside a Fixed Length Record

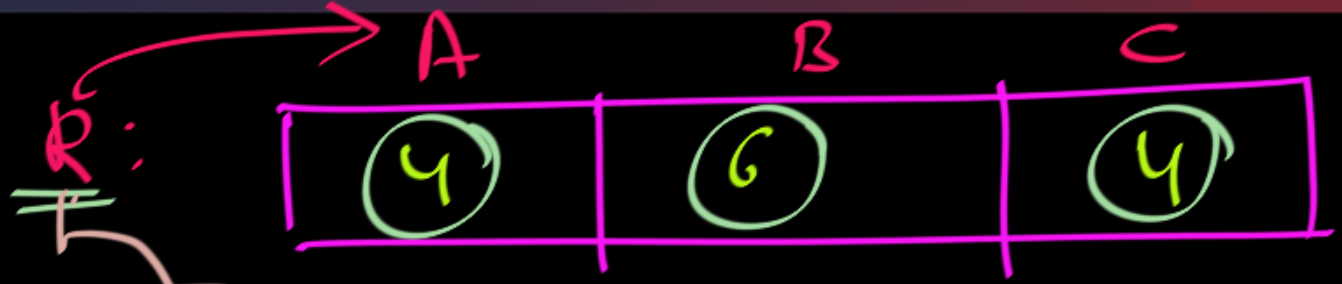
Table R (int A, char B[6], int C)



R:

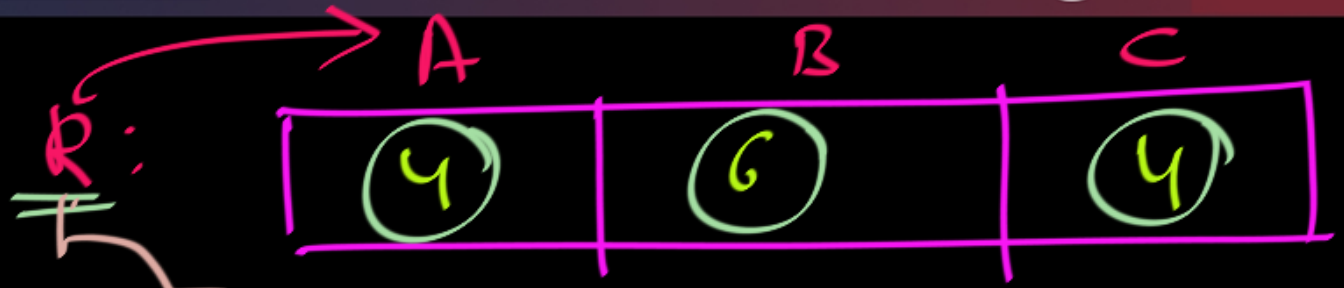


S:



Where stored !!

- How many fields are there ?
- Size of " ?
- time when you accesses the Record Recently ?



time when you accesses the Record Recently ?

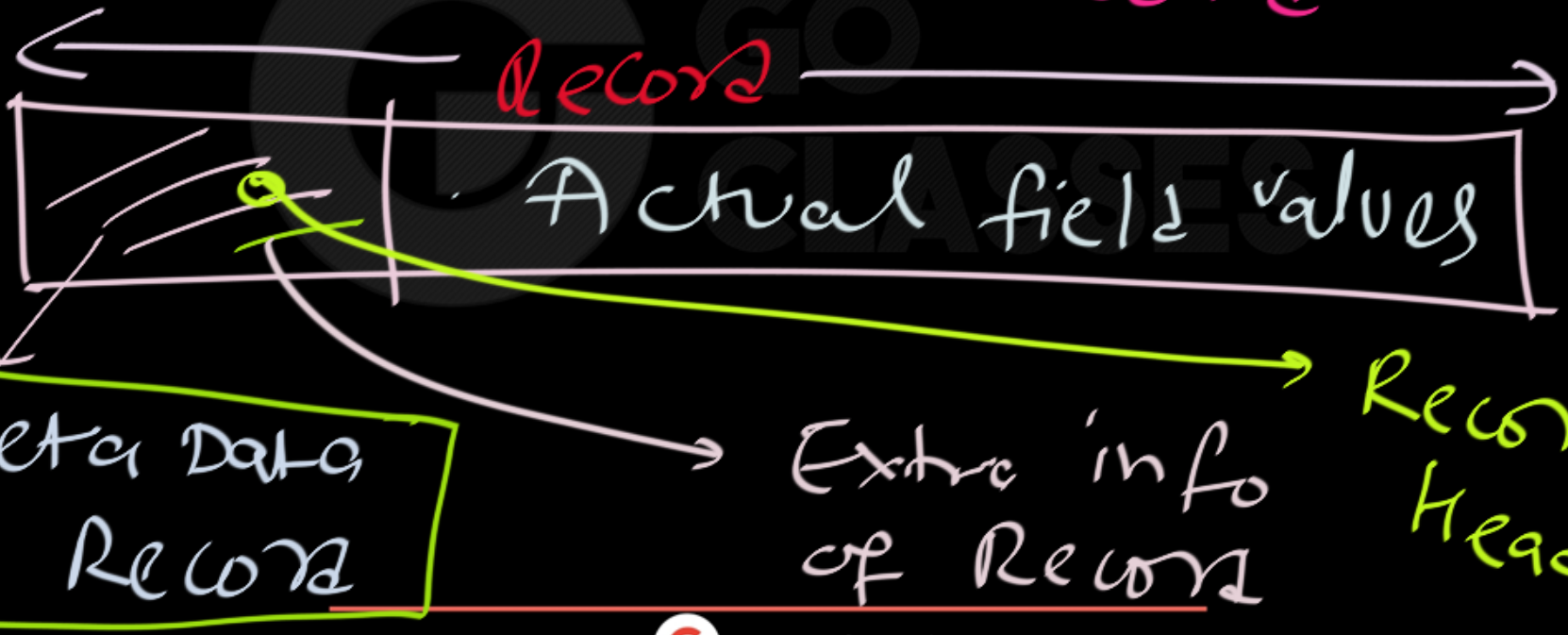
Record property

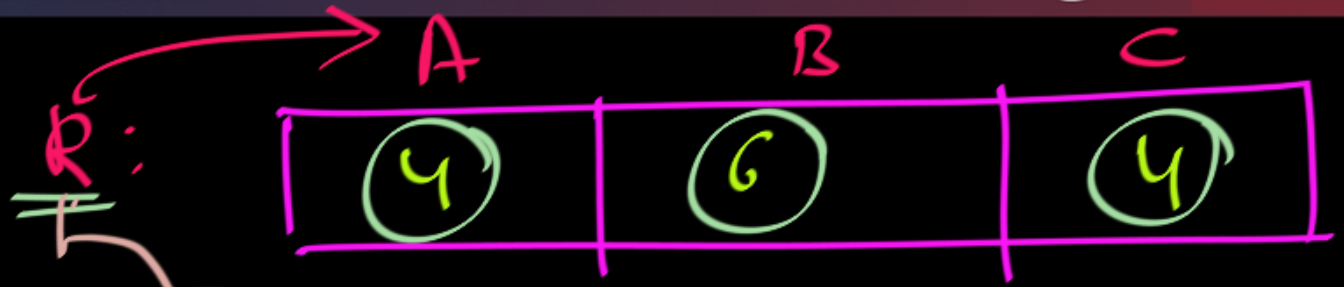
Stored inside Record

Record Property :

Eg: Time Stamp of Record

→ Stored in the Record

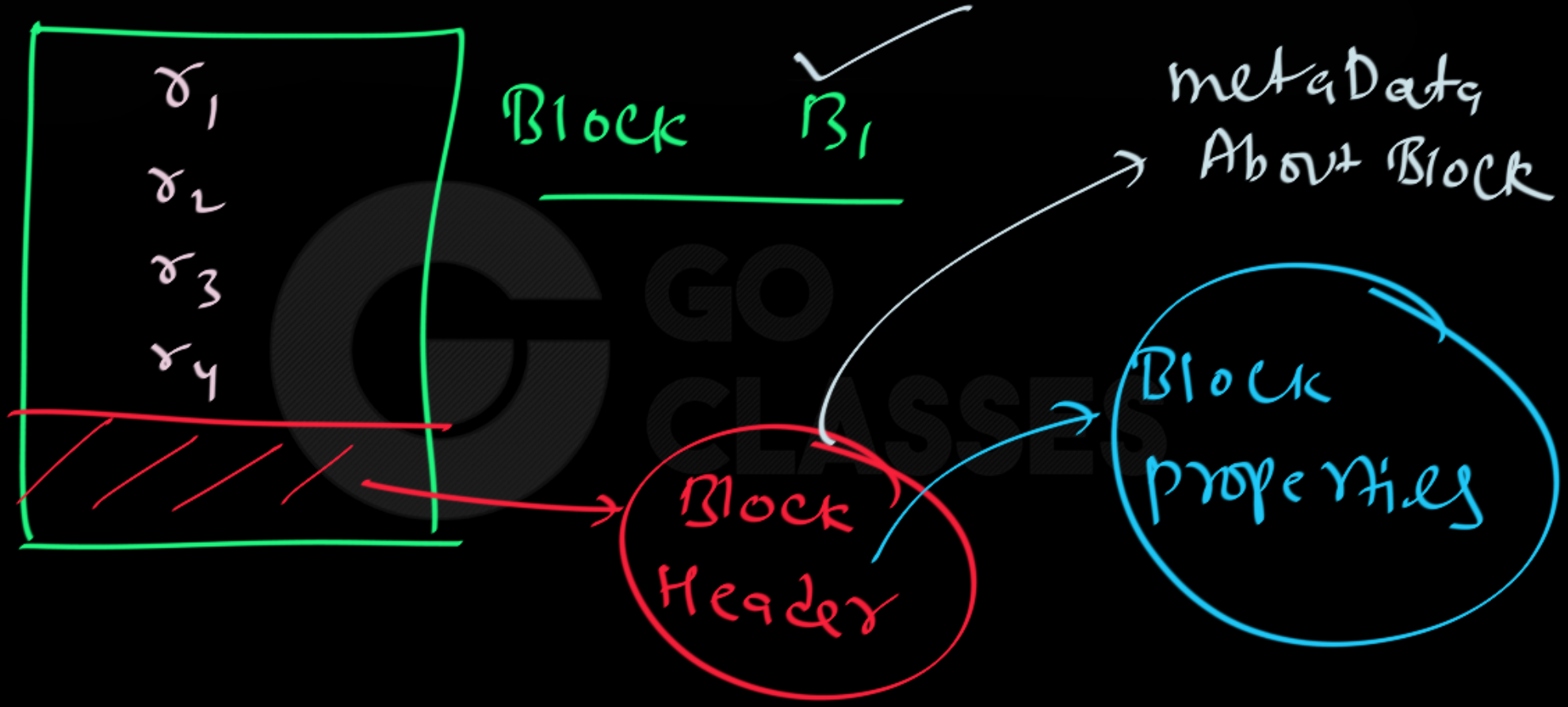


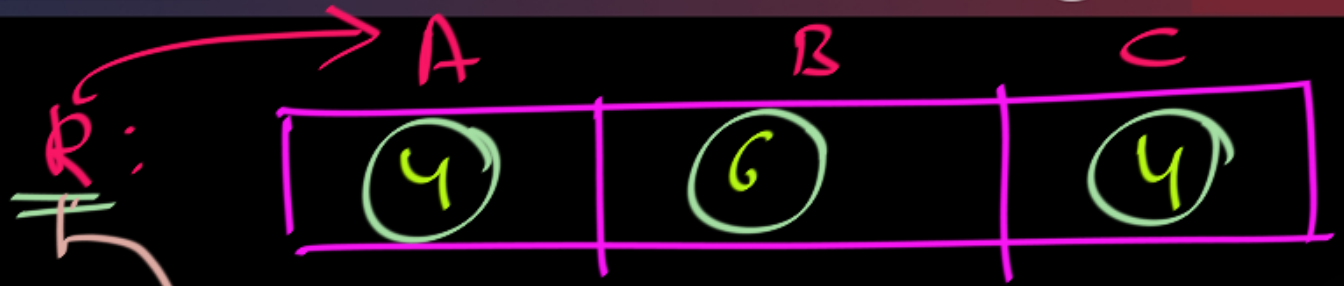


File Property

How many fields are there?

Block Header


r_1
r_2
r_3
r_4
Block B_1
metaData About Block
Block Header
Block properties



Variable length Records

Records header

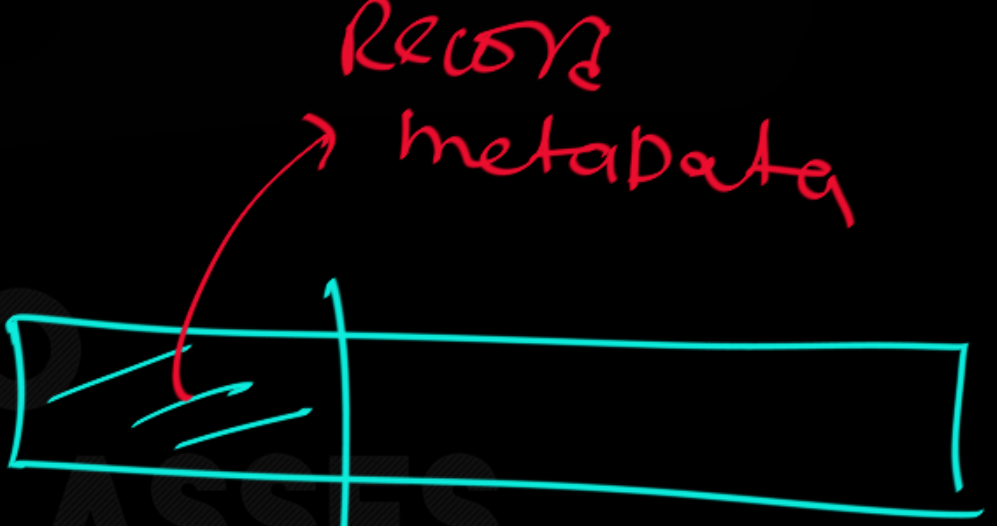
+ Size of every field ?

fixed length Records

Block Header

"Conclusion":

Record Header:



Block Header:

7.7.1 Fixed-Length Records

In a fixed-length record, each field has a fixed length (that is, the value in this field is of the same length in all records), and the number of fields is also fixed. The fields of such a record can be stored consecutively, and, given the address of the record, the address of a particular field can be calculated using information about the lengths of preceding fields, This record organization is illustrated in Figure 7.9.

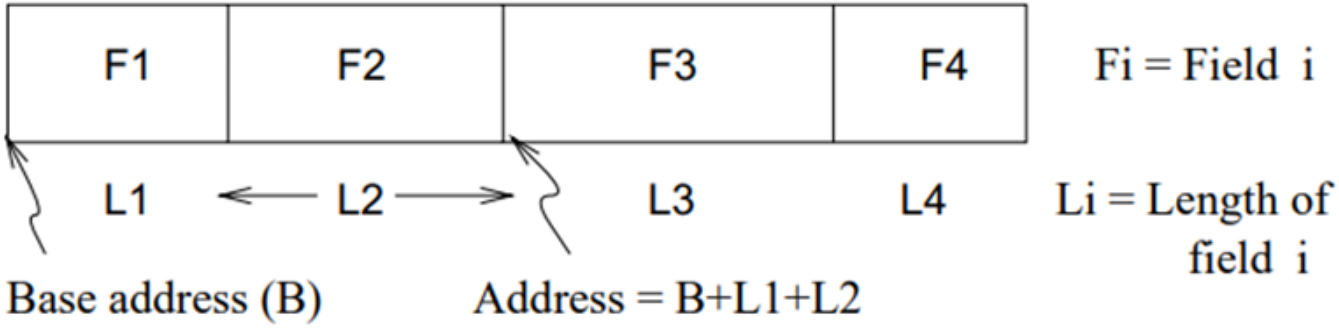
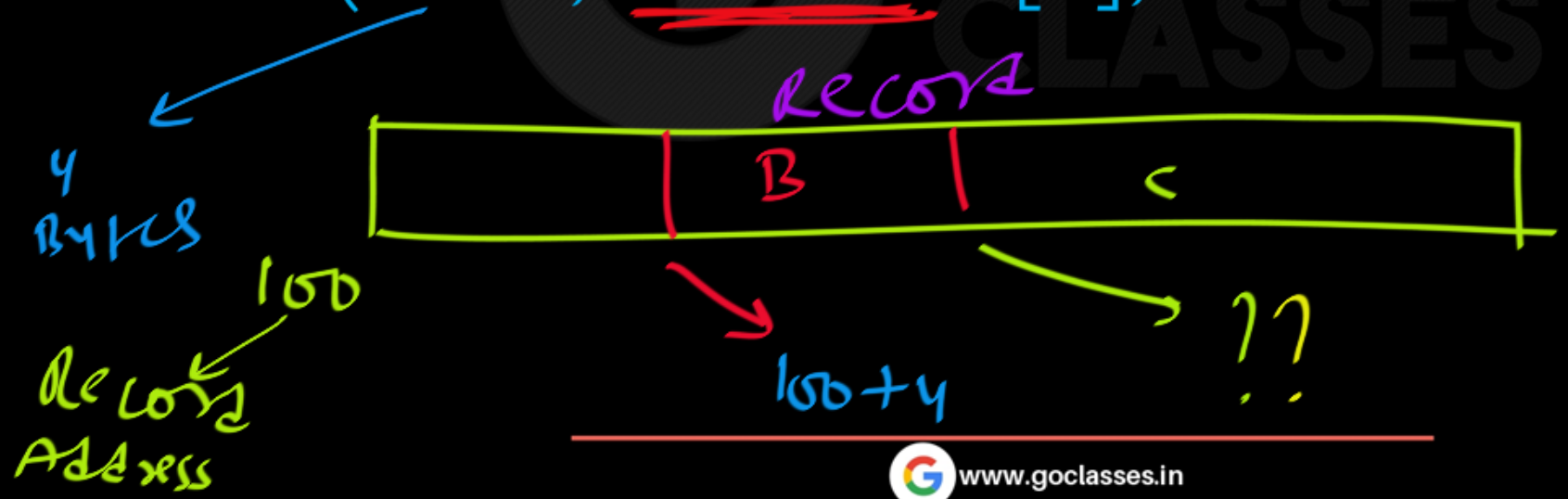


Figure 7.9 Organization of Records with Fixed-Length Fields

How to find a particular field value

inside a Variable Length Record

Table R (int A, varchar B[6], varchar C[10])

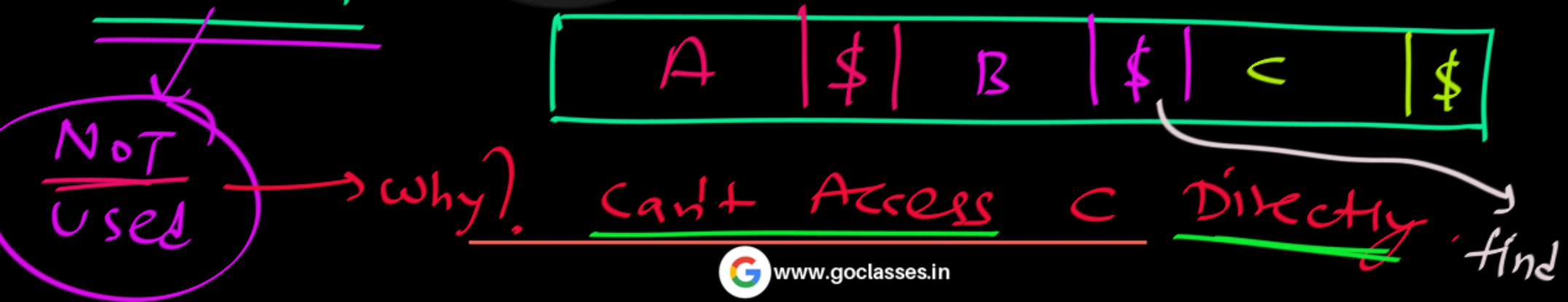


How to find a particular field value

inside a Variable Length Record

Table R (int A, varchar B[6], varchar C[10])

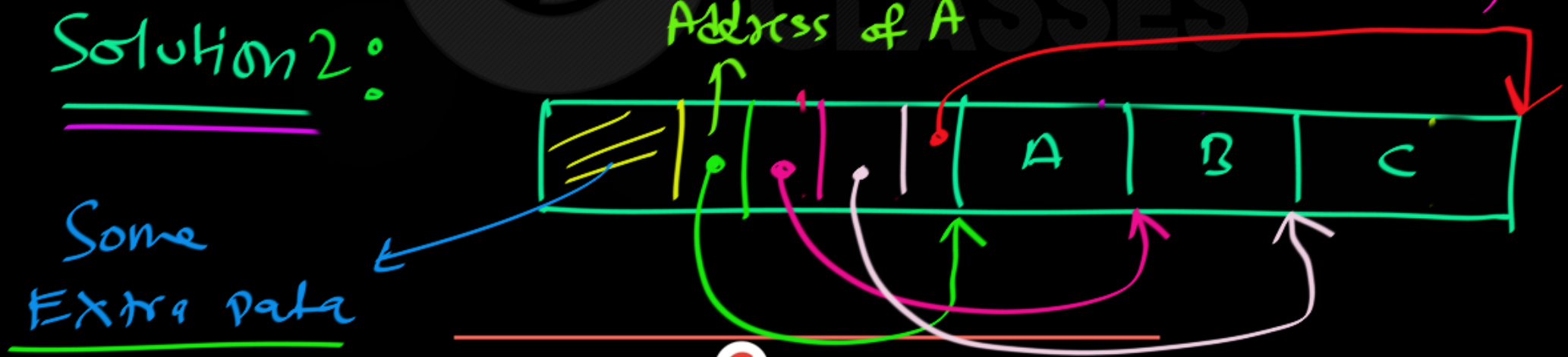
Solution 1:



How to find a particular field value inside a Variable Length Record

Table R (int A, varchar B[6], varchar C[10])

Solution 2:



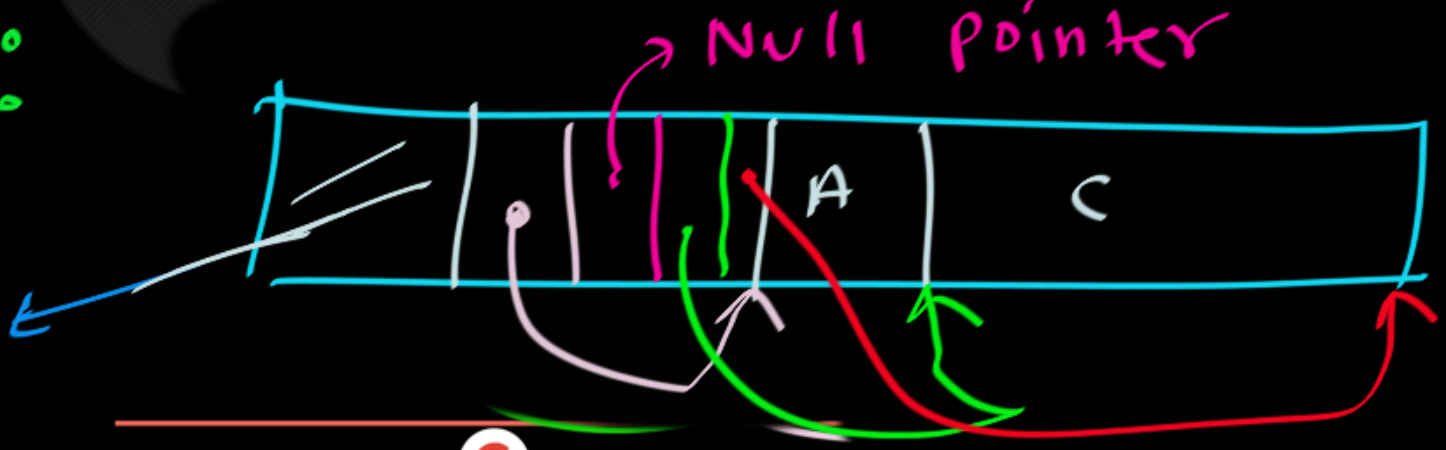
for John : B = Null

inside a Variable Length Record

Record Entries

Table R (int A, varchar B[6], varchar C[10])

Solution 2 :



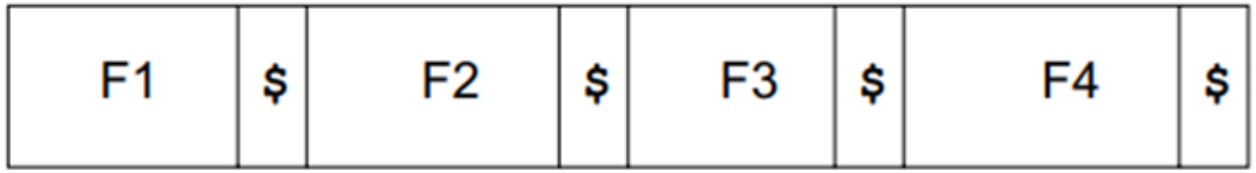
7.7.2 Variable-Length Records

In the relational model, every record in a relation contains the same number of fields. If the number of fields is fixed, a record is of variable length only because some of its fields are of variable length.

One possible organization is to store fields consecutively, separated by delimiters (which are special characters that do not appear in the data itself). This organization requires a scan of the record in order to locate a desired field.

How to find a particular field value inside a Variable Length Record

Approach 1:



Fi = Field i

Fields delimited by special symbol $

An alternative is to reserve some space at the beginning of a record for use as an array of integer offsets—the ith integer in this array is the starting address of the ith field value relative to the start of the record. Note that we also store an offset to the end of the record; this offset is needed to recognize where the last field ends. Both alternatives

How to find a particular field value

inside a Variable Length Record

Approach 2:

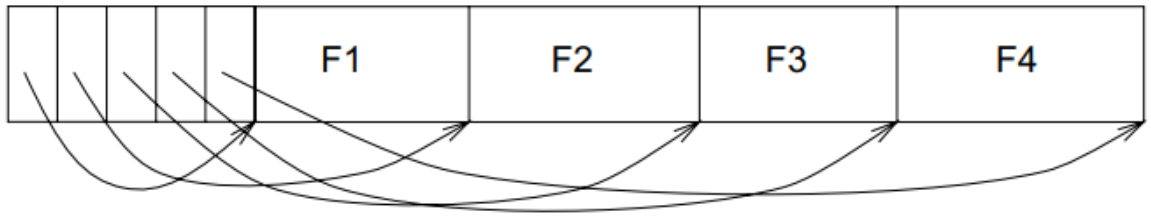


Figure 7.10 Alternative Record Organizations for Variable-Length Fields

Representing Null Values

Tuples often have fields that may be NULL. The record format of Fig. 13.23 offers a convenient way to represent NULL values. If a field such as address is null, then we put a null pointer in the place where the pointer to an address goes. Then, we need no space for an address, except the place for the pointer. This arrangement can save space on average

Next Topic:

File Organization

Spanned versus Unspanned Record Organization

Assume that the Disk Block Size is 100 Bytes.

Record size is 40 Bytes.

How many records can we keep in one block??

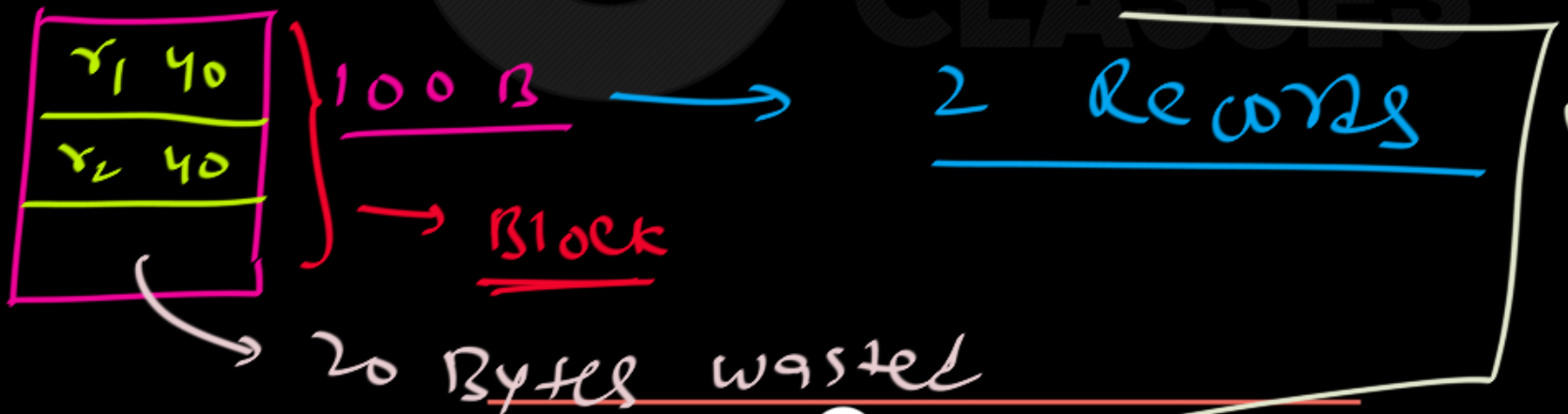
Assume that the Disk Block Size is 100 Bytes.

Record size is 40 Bytes.

How many records can we keep in one block??

2

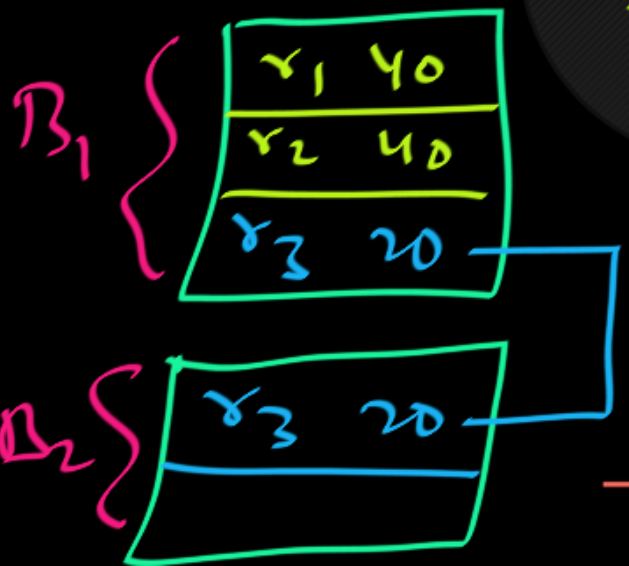
$\lfloor \frac{100}{40} \rfloor$



Assume that the Disk Block Size is 100 Bytes.

Record size is 40 Bytes.

How many records can we keep in one block??



Spanned org

$\frac{100}{40} = 2.5$

1.

UnSpanned Record Organization

Unspanned Organization:

Records are Not allowed to cross block boundaries, the organization is called unspanned.

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes,

How many records can we keep in one block, using unspanned record organization(which is By Default as well)??

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes,

How many records can we keep in one block, using unspanned record organization(which is By Default as well)??

$$\left\lfloor \frac{B}{R} \right\rfloor$$

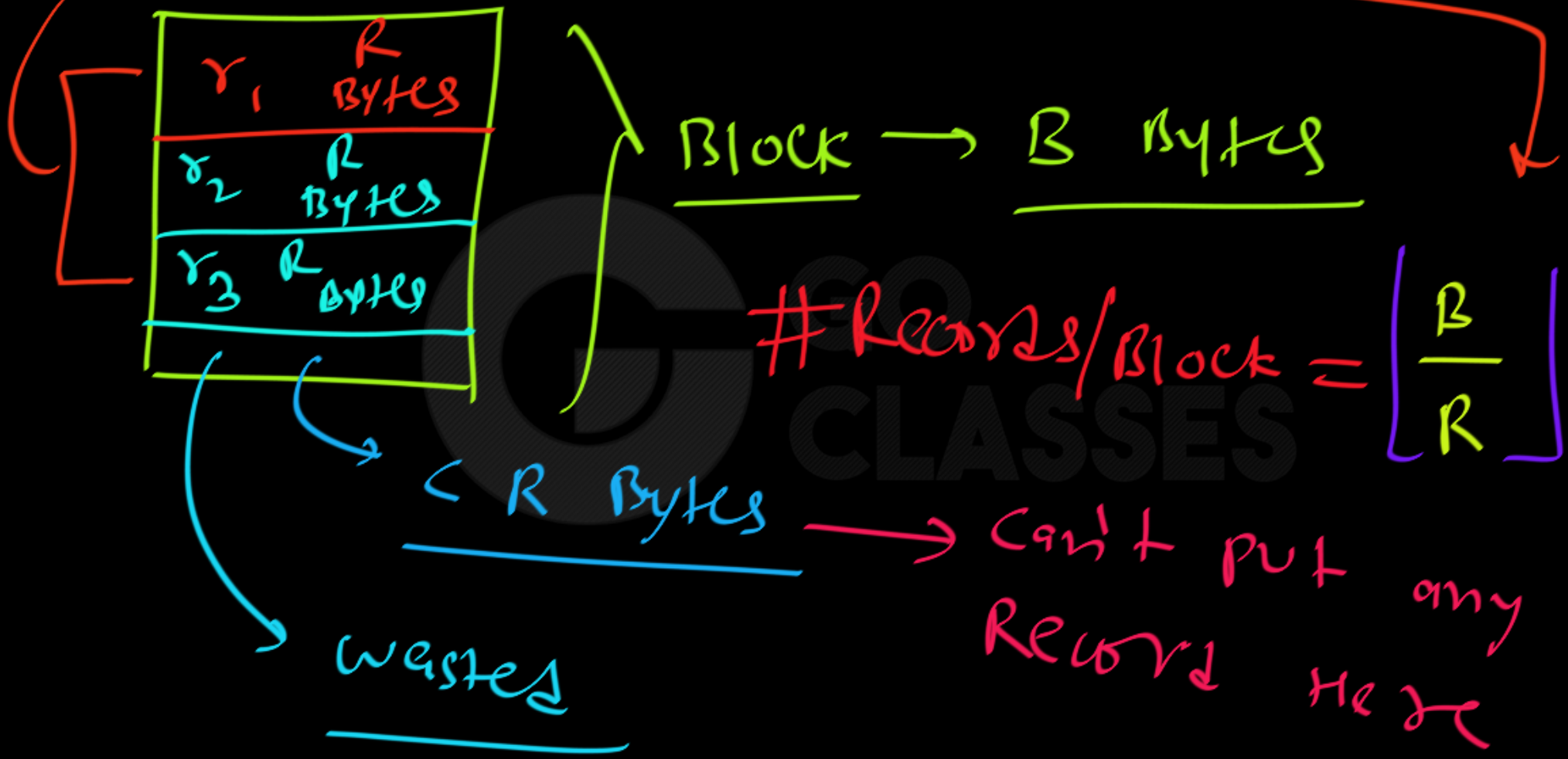
Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes,

How many records can we keep in one block, using unspanned record organization(which is By Default as well)??

Block Header Size = K Bytes

$$\#\text{Records/Block} = \left\lfloor \frac{B-K}{R} \right\rfloor$$

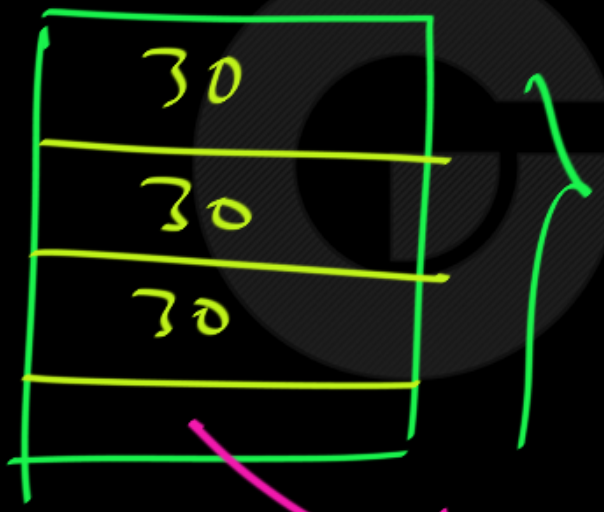

r_1 R Bytes
r_2 R Bytes
r_3 R Bytes
Block → B Bytes
#Records/Block = $\lfloor \frac{B}{R} \rfloor$
< R Bytes → Can't put any Record here
Wastes

$B = 100$ Bytes

$R = 30$ Bytes

→ Blocking factor

$$\#\text{Records/Block} = \left\lfloor \frac{100}{30} \right\rfloor = 3$$



100 Bytes

10 Bytes wastes

#Records / Block = Blocking factor

Blocking Factor:

- f_r, the blocking factor of relation r—that is, the number of tuples of relation r that fit into one block.

The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory. Each block can contain numerous records.

Suppose that the block size is B bytes. For a file of fixed-length records of size R bytes, with $B \geq R$,

we can fit bfr = $\lfloor B/R \rfloor$ records per block.

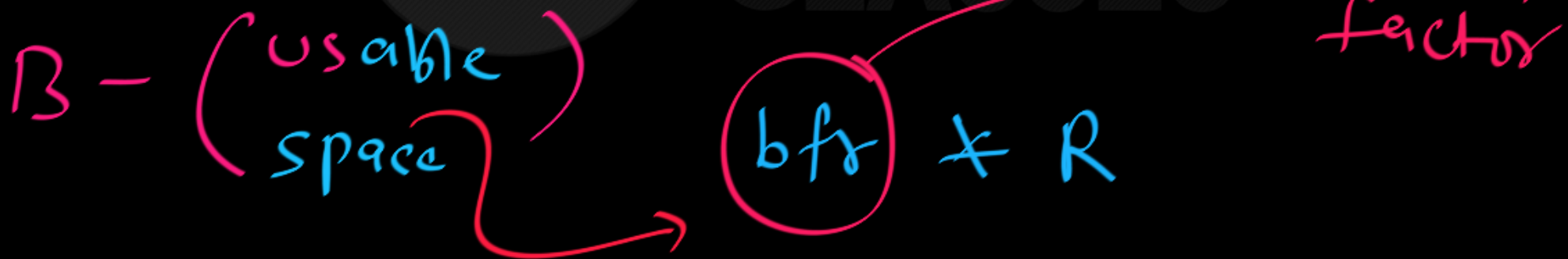
The value bfr is called the blocking factor for the file.

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes, How much unused space in each block, using unspanned record organization (which is By Default as well)??

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes, How much unused space in each block, using unspanned record organization (which is By Default as well)??



$$B - \left(\left\lfloor \frac{B}{R} \right\rfloor * R \right)$$

Blocking factor
(#Records per Block)

The value *bfr* is called the **blocking factor** for the file. In general, *R* may not divide *B* exactly, so we have some unused space in each block equal to

$$B - (bfr * R) \text{ bytes}$$

2.
Spanned Record Organization

Spanned Organization:

Records are allowed to cross block boundaries, the organization is called spanned.

We can store part of a record on one block and the rest on another. This organization is called spanned because records can span more than one block.

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes,

How many records can we keep in one block(i.e.

Blocking Factor), using spanned record organization??

Q: Assume that the Disk Block Size is 100 Bytes.

For a file of fixed-length records of size 40 Bytes,

How many records can we keep in one block(i.e.

Blocking Factor), using spanned record organization??

$\frac{100}{40} = 2.5 \rightarrow$ Blocking factor

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes,

How many records can we keep in one block(i.e.

Blocking Factor), using spanned record organization??

$\frac{B}{R}$ Straight

Blocking Factor:

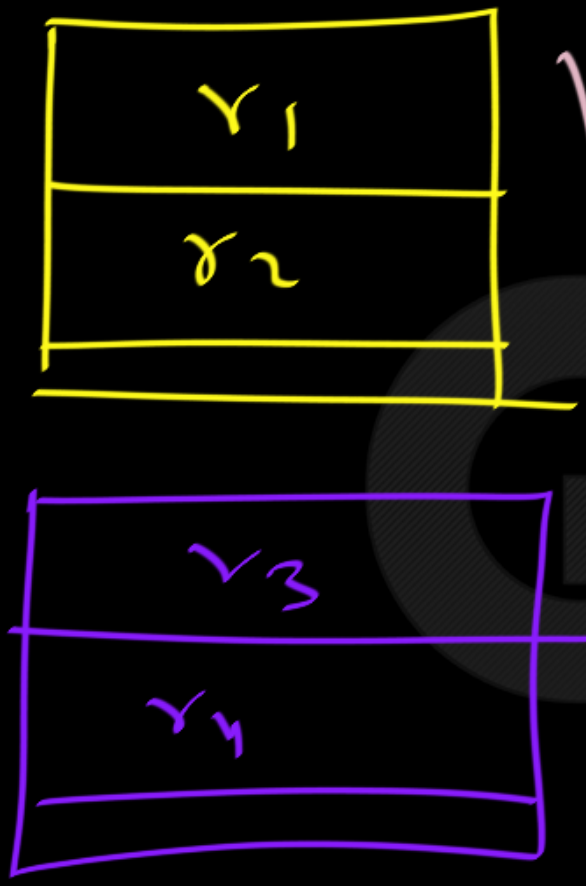
- f_r, the blocking factor of relation r—that is, the number of tuples of relation r that fit into one block.

Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes, How much unused space in each block, using spanned record organization?

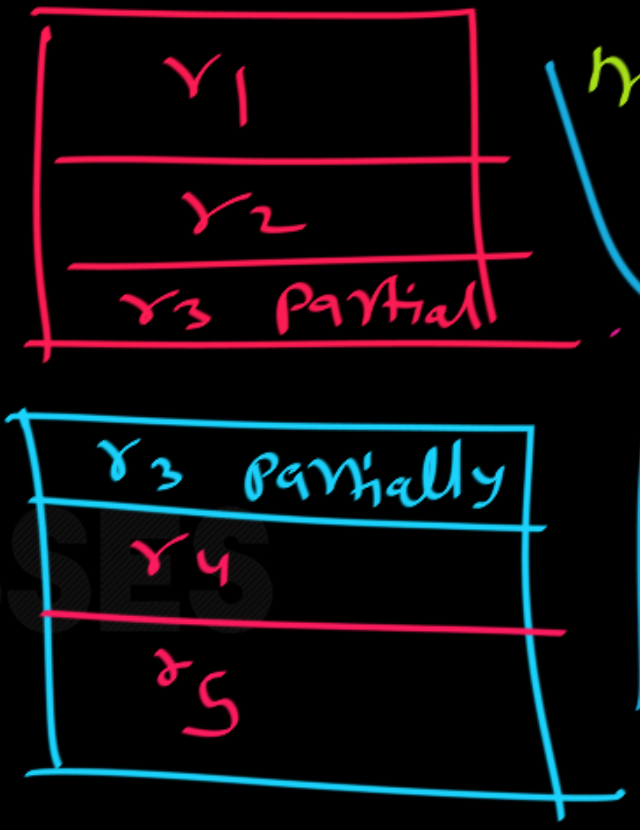
Q: Assume that the Disk Block Size is B Bytes.

For a file of fixed-length records of size R Bytes, How much unused space in each block, using spanned record organization?



1 Block Access for one Record

Unspanned



Multiple Block Access for one Record

Spanned

NOTE:

Unspanned Organization is generally used with fixed-length records because it makes each record start at a known location in the block, simplifying record processing.

For variable-length records, either a spanned or an unspanned organization can be used.

NOTE:

1. Unspanned Organization is By Default taken.

2. Fixed-length records is By Default consideration.

NOTE:

For variable-length records using spanned organization, each block may store a different number of records. In this case, the blocking factor bfr represents the average number of records per block for the file.

Q:

For variable-length records using spanned organization, Let the blocking factor bfr represent the average number of records per block for the file.

We can use bfr to calculate the number of blocks b needed for a file of r records.

HOW??

Q:

For variable-length records using spanned organization, Let the blocking factor bfr represent the average number of records per block for the file.

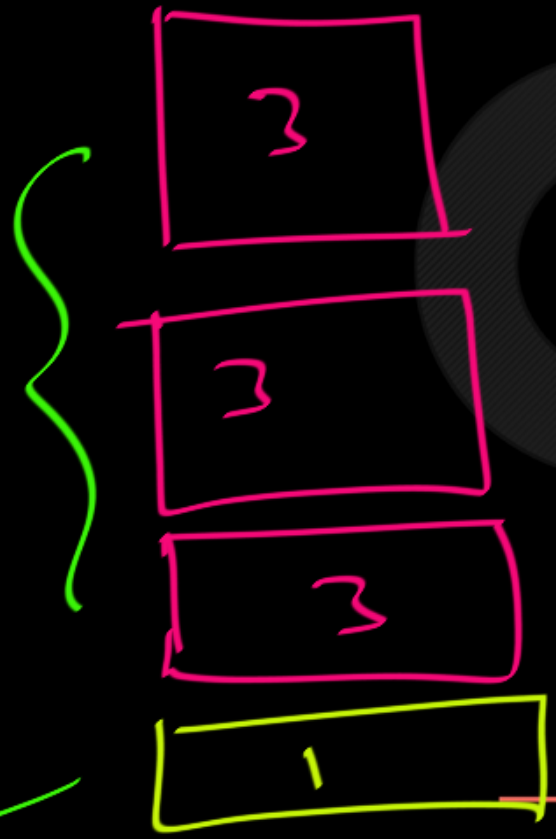
We can use bfr to calculate the number of blocks b needed for a file of r records.

HOW??

bfr = 3

10 Records

10 Records ; 1 Block → on Average 3 Records



$\left\lceil \frac{10}{3} \right\rceil$ = 4 Blocks for this file of 10 Records.

Q:

For variable-length records using spanned organization, Let the blocking factor bfr represent the average number of records per block for the file.

We can use bfr to calculate the number of blocks b needed for a file of r records.

HOW??

bfr: Average number of Records Per Block

File of r Records:

$$\#\text{Blocks needed} = \left\lceil \frac{r}{bfr} \right\rceil$$

Ans:

For variable-length records using spanned organization, Let the blocking factor bfr represent the average number of records per block for the file. We can use bfr to calculate the number of blocks b needed for a file of r records.

$$b = \lceil (r/bfr) \rceil \text{ blocks}$$

where the $\lceil (x) \rceil$ (*ceiling function*) rounds the value x up to the next integer.

Q:

For variable-length records, different records have different sizes. Let the average record size in a file be R, and number of records in the file be r. Assume that Block size is B bytes, and K bytes are used for Block Header. Using spanned organization, find the number of blocks needed to store this file?

Q:

For variable-length records, different records have different sizes. Let the average record size in a file be R, and number of records in the file be r. Assume that Block size is B bytes, and K bytes are used for Block Header. Using spanned organization, find the number of blocks needed to store this file?

Average Record Size : R Bytes

→ Consider it as fixed Record size.

#Records Per Block :

$$\frac{B-k}{R} = bfr$$

Blocking factor

Spanned or

$$\left\lceil \frac{r}{bfr} \right\rceil = \text{\#Blocks needed}$$

$$= \left\lceil \frac{r}{(B-k)/R} \right\rceil = \left\lceil \frac{rR}{(B-k)} \right\rceil$$

Next Topic:

File Organization

Organization of Records in Files

A relation is a set of records.

Given a set of records, the next question is how to organize them in a file(in a disk, in which order).

Determine how the file records are Physically Placed on the disk.

Conceptual level : ⇒ Order of Records Does Not matter

Physical level : ⇒ How we put Records, in what order, matters

Determine how the file records are Physically Placed on the disk:

1. Put Unordered:

Files of Unordered Records (Heap Files):

In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a heap file.


Heap
Organisation
Pile
Organisation

Determine how the file records are Physically Placed on the disk:

1. Put Unordered:

Files of Unordered Records (Heap Files):

Benefits: Insertion Efficient

Disadvantage: Searching Inefficient

Determine how the file records are Physically Placed on the disk:

1. Files of Unordered Records (Heap Files):

Inserting a new record is very efficient. The last disk block of the file is copied into a main memory buffer, the new record is added, and the block is then rewritten back to disk. The address of the last file block is kept in the file header.

Determine how the file records are Physically Placed on the disk:

1. Files of Unordered Records (Heap Files):

However, searching for a record using any search condition involves a linear search through the file block by block—an expensive procedure.

Determine how the file records are Physically Placed on the disk:

1. Files of Unordered Records (Heap Files):

If only one record satisfies the search condition, then, on the average, a program will read into memory and search half the file blocks before it finds the record. For a file of b blocks, this requires searching (b/2) blocks, on average. If no records or several records satisfy the search condition, the program must read and search all b blocks in the file.

Heap file $\Rightarrow$ (b) blocks

Searching time :

- Worst Case : b blocks
- Average Case : $b/2$ blocks

Determine how the file records are Physically Placed on the disk:

2. Put Ordered based on some field:

Files of Ordered Records (Sequential or Sorted Files):

We can physically order the records of a file on disk based on the values of one of their fields—called the ordering field. This leads to an ordered or sequential file.

Block 1

Name	Ssn	Birth_date	Job	Salary	Sex
Aaron, Ed					
Abbott, Diane					
		⋮			
Acosta, Marc					

Block 2

Adams, John					
Adams, Robin					
		⋮			
Akers, Jan					

Block 3

Alexander, Ed					
Alfred, Bob					
		⋮			
Allen, Sam					

Determine how the file records are Physically Placed on the disk:

2. Files of Ordered Records (Sequential or Sorted Files):

Ordered records have some advantages over unordered files. First, reading the records in order of the ordering field values becomes extremely efficient because no sorting is required.

Determine how the file records are Physically Placed on the disk:

2. Files of Ordered Records (Sequential or Sorted Files):

Can use Binary Search on the blocks while searching for a record based on ordering field value.

NOTE:

A binary search for disk files can be done on the blocks rather than on the records.

Determine how the file records are Physically Placed on the disk:

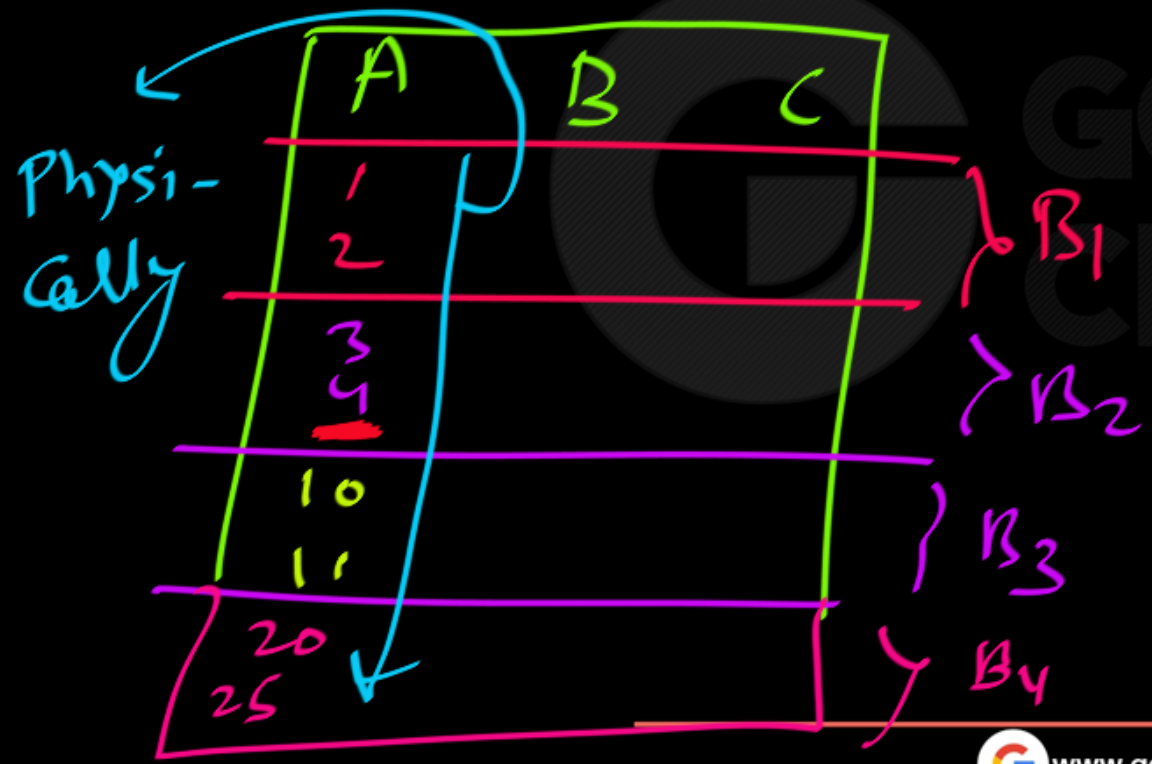
2. Files of Ordered Records (Sequential or Sorted Files):

Using a search condition based on the value of an ordering field results in faster access when the binary search technique is used, which constitutes an improvement over linear searches.

Ordered files are blocked and stored on contiguous cylinders to minimize the seek time.

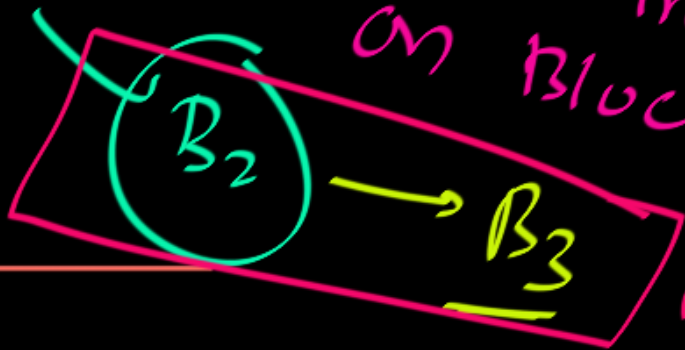
Sequential organization:

Ordered by field A:



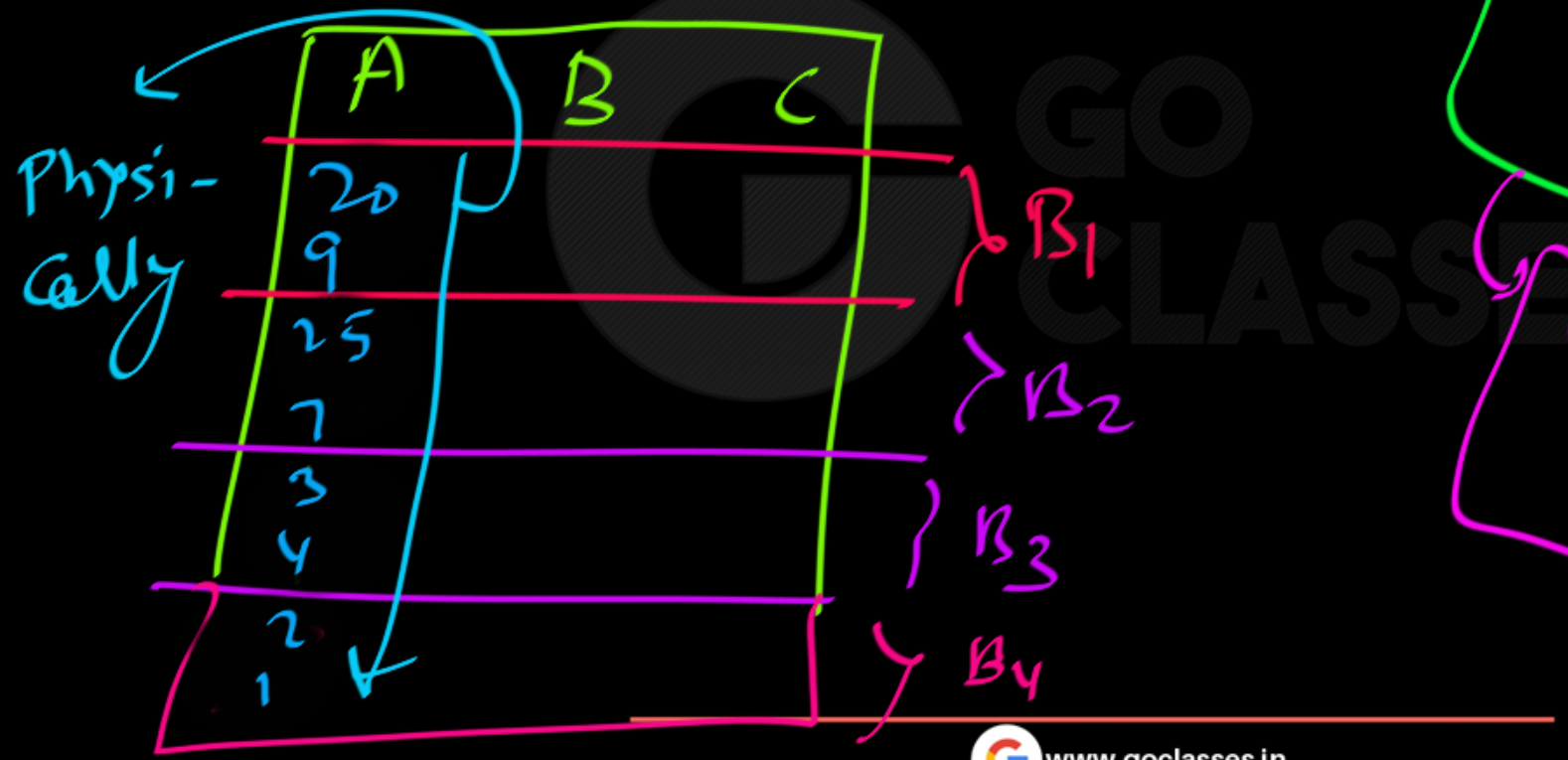
Search where A = 6

Binary search on Blocks:



Heap organization:

Ordered by field A:



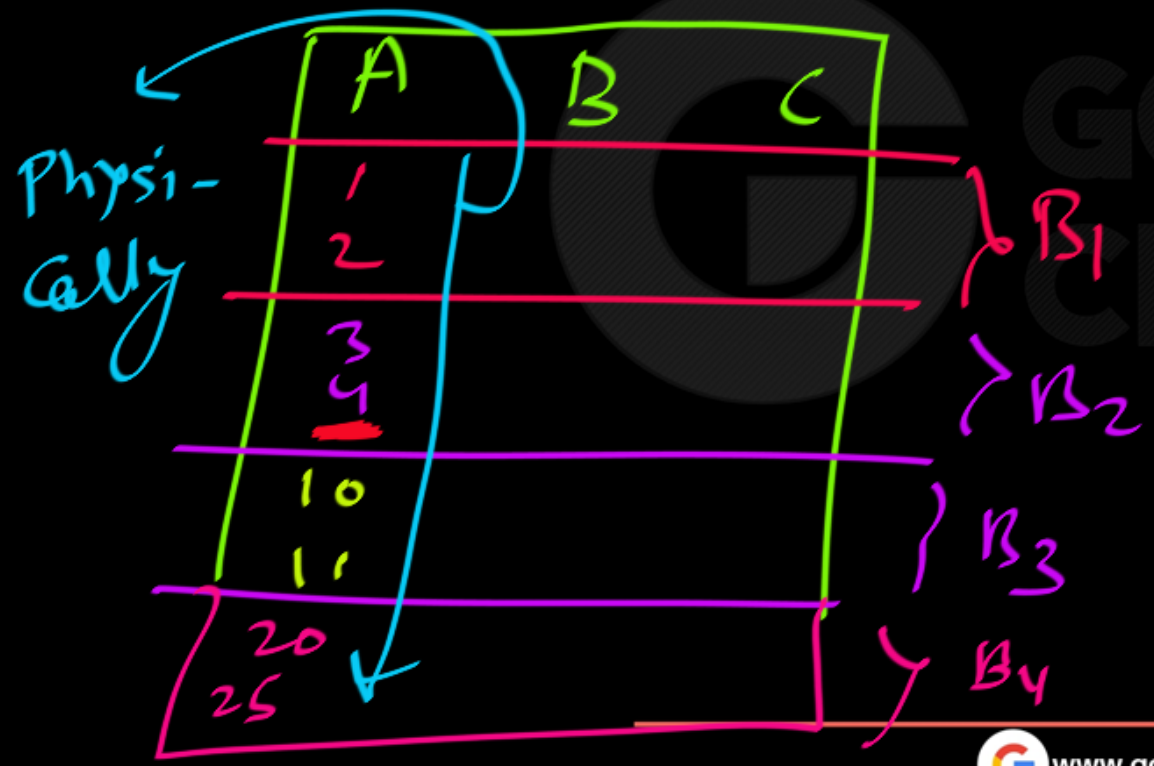
Worst Case

Search where A = 6

4 Blocks Access

Sequential organization:

Ordered by field A:



4 Block Access

Search where B = 6

from B Point of view; Heap org.

Sequential organization useful when searching is done based on Ordering field.

Determine how the file records are Physically Placed on the disk:

2. Files of Ordered Records (Sequential or Sorted Files):

NOTE:

Ordering does not provide any advantages for access of the records based on values of the other nonordering fields of the file.

when searching based on unordered attribute.

Table 16.3 Average Access Times for a File of b Blocks under Basic File Organizations

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

Organization of Records in Files

So far, we have studied how records are represented in a file structure. A relation is a set of records. Given a set of records, the next question is how to organize them in a file. Several of the possible ways of organizing records in files are:

- **Heap file organization**. Any record can be placed anywhere in the file where there is space for the record. There is no ordering of records. Typically, there is a single file for each relation.
- **Sequential file organization**. Records are stored in sequential order, according to the value of a "search key" of each record. Section 10.6.1 describes this organization.

There are several **file organizations**, which determine how the file records are *physically placed* on the disk, *and hence how the records can be accessed.* A *heap file* (or *unordered file*) places the records on disk in no particular order by appending new records at the end of the file, whereas a *sorted file* (or *sequential file*) keeps the records ordered by the value of a particular field (called the *sort key*).

Next Topic:

File Organization

Block and Record Addresses

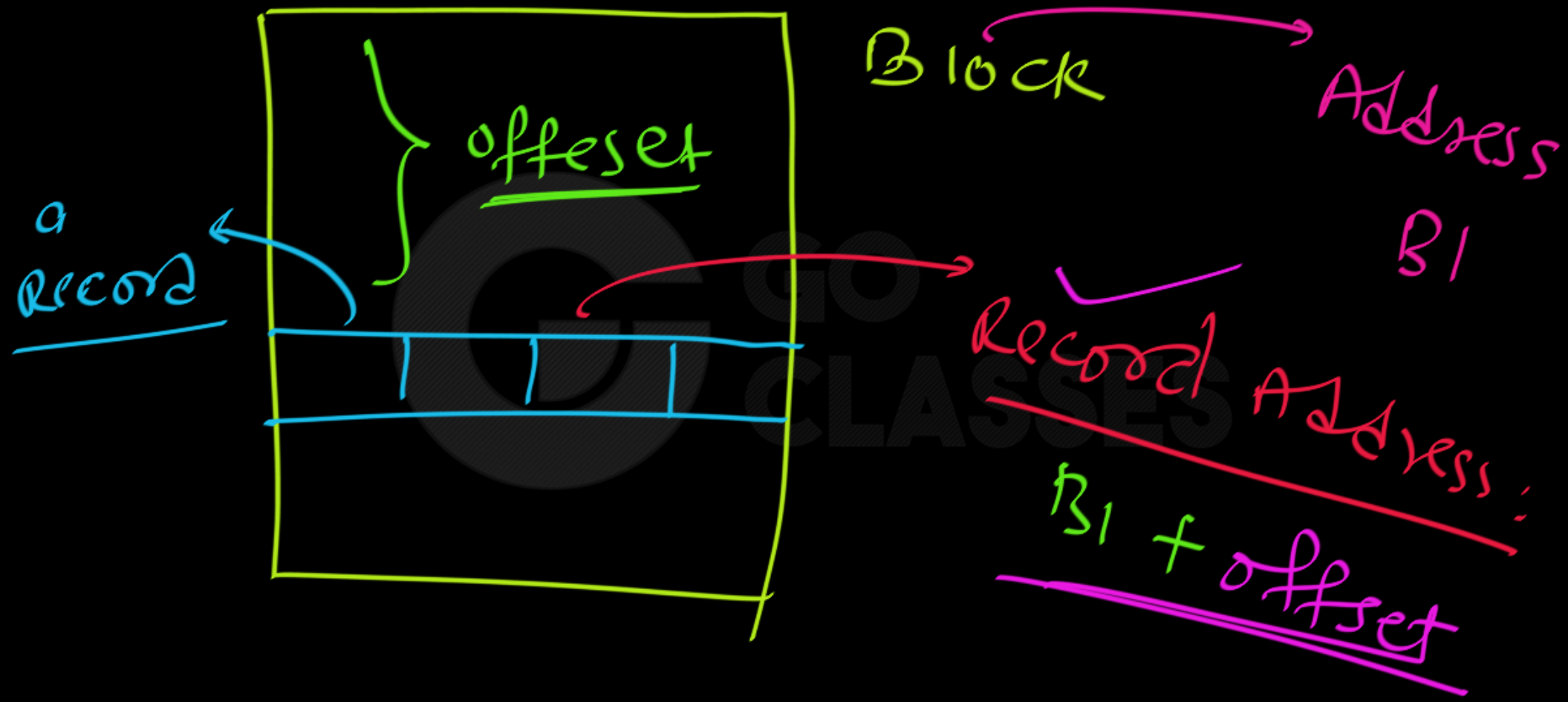
Block Address:

Address of the first sector of that block.

<c,h,s> address OR LBA address (We studied in Disk chapter)

Record Address:

A record can be identified by giving its block address and the offset of the first byte of the record within the block.


offeset
a
Record
Block
Address
B1
Record Address :
B1 + offset

Next Topic:

File Organization

Practice

Q: DATABASE SYSTEMS The Complete Book Ullman , Garcia-Molina, Widom ; Second Edition

13.5.3 Exercises for Section 13.5

Exercise 13.5.1: Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, a SQL date, and a SQL time (no decimal point). How many bytes does the record take if:

(SQL dates require 10 bytes, and SQL times require 8 bytes)

a) Fields can start at any byte.

b) Fields must start at a byte that is a multiple of 4.

c) Fields must start at a byte that is a multiple of 8.

Q: DATABASE SYSTEMS The Complete Book Ullman, Garcia-Molina, Widom ; Second Edition

13.5.3 Exercises for Section 13.5

Exercise 13.5.1: Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, a SQL date, and a SQL time (no decimal point). How many bytes does the record take if:

(SQL dates require 10 bytes, and SQL times require 8 bytes)

a) Fields can start at any byte.

b) Fields must start at a byte that is a multiple of 4.

c) Fields must start at a byte that is a multiple of 8.

address

Field ≡ Attributes

(a) Record Size:

15 + 2 + 10 + 8 = 35 Bytes

(b) Every field starts at a byte address multiple of 4:

Attribute 16 + 4 + 12 + 8 = 40 Bytes

Round 15 to multiple of 4 ≡ 16 (1 Byte Padding)

(a) Record Size:

15 + 2 + 10 + 8 = 35 Bytes

(c) Every field starts at a byte address multiple of 8:

Attribute

16 + 8 + 16 + 8 = 48 Bytes

Round 15 to multiple of 8 ≡ 16 (1 Byte Padding)

C :⇒ Structure Alignment → All variation studies in c-course.

PYQ

a)

The bytes required by each of the fields is $15 + 2 + 10 + 8 = 35$.

b)

Round each of the four field lengths up to a multiple of 4, to get $16 + 4 + 12 + 8 = 40$.

c) Round up again, to a multiple of 8, to get $16 + 8 + 16 + 8 = 48$.

Q: DATABASE SYSTEMS The Complete Book Ullman , Garcia-Molina, Widom ; Second Edition

13.5.3 Exercises for Section 13.5

Exercise 13.5.1: Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, a SQL date, and a SQL time (no decimal point). How many bytes does the record take if:

a) Fields can start at any byte.

b) Fields must start at a byte that is a multiple of 4.

c) Fields must start at a byte that is a multiple of 8.

(SQL dates require 10 bytes, and SQL times require 8 bytes)

Exercise 13.5.2: Repeat Exercise 13.5.1 for the list of fields: a real of 8 bytes, a character string of length 17, a single byte, and a SQL date.

Q: DATABASE SYSTEMS The Complete Book Ullman , Garcia-Molina, Widom ; Second Edition

13.5.3 Exercises for Section 13.5

Exercise 13.5.1: Suppose a record has the following fields in this order: A character string of length 15, an integer of 2 bytes, a SQL date, and a SQL time (no decimal point). How many bytes does the record take if:

a) Fields can start at any byte. — 35 B

b) Fields must start at a byte that is a multiple of 4. — 40 B

c) Fields must start at a byte that is a multiple of 8. — 48 B

(SQL dates require 10 bytes, and SQL times require 8 bytes)

Exercise 13.5.2: Repeat Exercise 13.5.1 for the list of fields: a real of 8 bytes, a character string of length 17, a single byte, and a SQL date.

R(8 B, 17 B, 1 B, 10 B) → 4 fields

(a) 8 + 17 + 1 + 10 = 36 B

(b) every field starts at byte address multiple of 4:

8 + 20 + 4 + 12 = 44 B

R(8 B, 17 B, 1 B, 10 B) → 4 fields

(a) 8 + 17 + 1 + 10 = 36 B

(b) every field starts at byte address multiple of 8 :

8 + 24 + 8 + 16 = 56 B

Q 16.34: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

Problem 1

Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size B=512 bytes, interblock gap size G=128 bytes, number of blocks per track=20, number of tracks per surface=400. A disk pack consists of 15 double-sided disks.

(a) What is the total capacity of a track and what is its useful capacity (excluding interblock gaps)?

(b) How many cylinders are there?

(c) What is the total capacity and the useful capacity of a cylinder?

(d) What is the total capacity and the useful capacity of a disk pack?

(e) Suppose the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what are the transfer rate in bytes/msec and the block transfer time *btt* in msec? What is the average rotational delay rd in msec?

(f) Suppose the average seek time (time to go to the correct track mechanically) is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block given its block address?

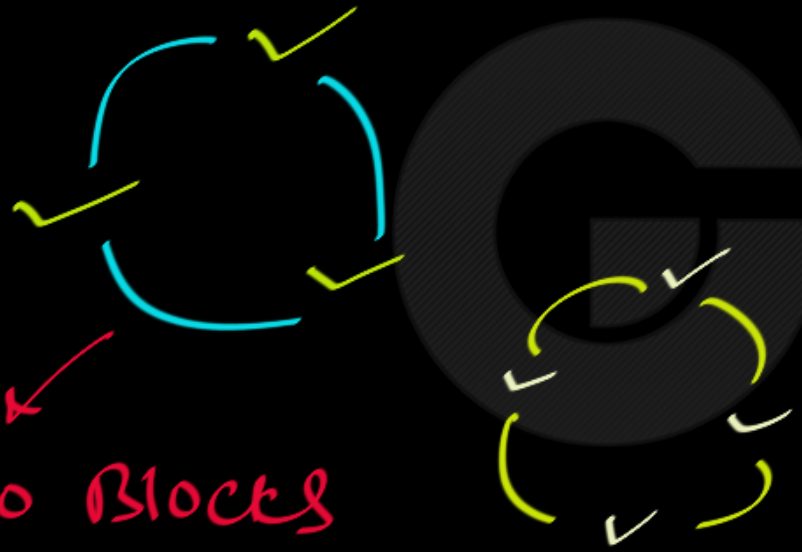
(g) Calculate the average time it would take to transfer 20 random blocks and compare it with the time it would take to transfer 20 consecutive blocks using double buffering to save seek time and rotational delay.

Double buffering:

↳ Two buffers in main memory

(Simple seek, || Rotational Delay)

(a) Track Size: $20 \times (512 + 128)$ B



(b)

#Cylinders ≠ #Tracks

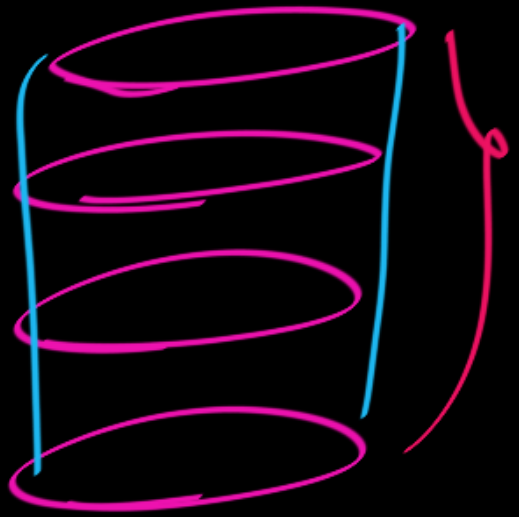
#Cylinders = 400

#Tracks per surface = 400

#Tracks = 400 per surface × 30 surfaces

= 12000 Tracks

(c) Total cylinder Capacity =



30 surfaces X Total Track Size

= 30 X [20 X (512 + 128) B]

Useful Cylinder Capacity :

30 X 20 X 512 B = 30 X Useful Track Size

(d) Total Disk Capacity:

#cylinders X Total Cylinder Capacity

= 400 X

Useful Disk Capacity:

= 400 X Useful cylinder capacity

(e) 2400 RPM :

2400 Rotations — 60 sec

1 Rotation — $\frac{60}{2400}$ sec

Avg Rotational Delay = Avg Latency =

$= \frac{1}{2}$ (one Rotation Time)

$= \frac{1}{2}$ (25 ms)

$= 12.5$ ms

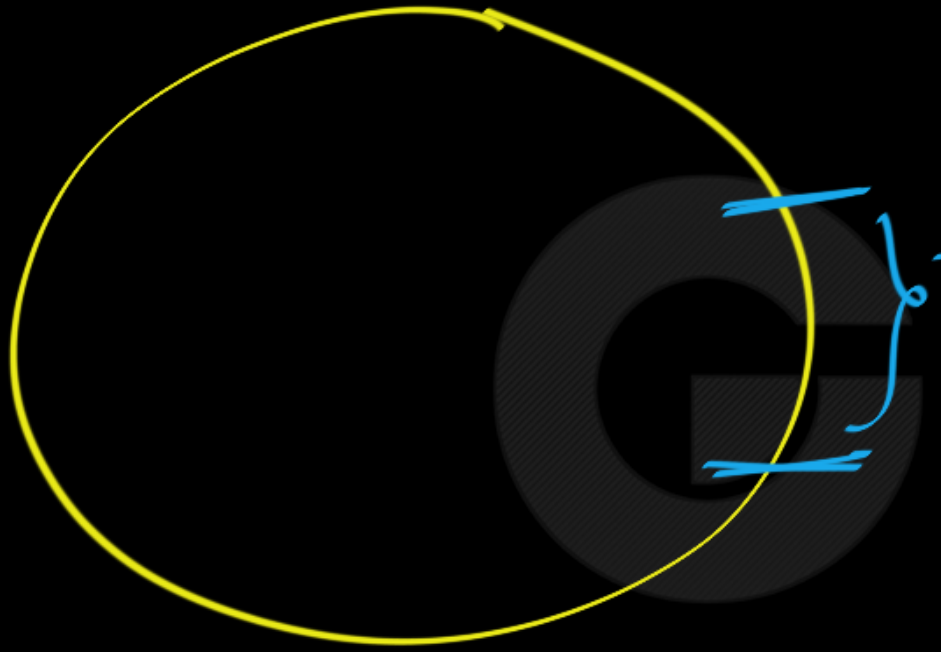
(e) Transfer Rate : Data / sec

1 Rotation → Track Size = 20 × (512 + 128)

25 ms

1 ms → $\frac{20 \times (640)}{25}$ Bytes

= 512 Bytes



k bytes → s ms

"Transfer Rate":

$\frac{k}{s}$ Bytes per ms

(e)

btt :

1 ms ⟶ 512 Bytes

Block Size

Tranfer time :

btt = 1 ms

(f) Block Address is Given;

Total time : Access time + Block Transfer time

Access time = Avg Seek time + Avg Rotational Latency

Avg Seek time = 30 ms

Avg Rotational Latency = 12.5 ms

Block Transfer time = 1 ms

= 43.5 ms

(I) 20 Random Blocks := 20 x (s + R + btt)

20 x 43.5 ms

per Block total time

20 Consecutive Blocks with Double buffering:

30 + 12.5 + 20 x (1 ms) → btt

Once

Solution:

(a) Total track size = 20 * (512+128) = 12800 bytes = 12.8 Kbytes
Useful capacity of a track = 20 * 512 = 10240 bytes = 10.24 Kbytes

(b) Number of cylinders = number of tracks = 400

(c) Total cylinder capacity = 15*2*20*(512+128) = 384000 bytes = 384 Kbytes
Useful cylinder capacity = 15 * 2 * 20 * 512 = 307200 bytes = 307.2 Kbytes

(d) Total capacity of a disk pack = 15 * 2 * 400 * 20 * (512+128) = 153600000 bytes = 153.6 Mbytes
Useful capacity of a disk pack = 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

(e) Transfer rate tr= (total track size in bytes)/(time for one disk revolution in msec)
tr= (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512 bytes/msec
block transfer time btt = B / tr = 512 / 512 = 1 msec
average rotational delay rd = (time for one disk revolution in msec) / 2 = 25 / 2 = 12.5 msec

(f) average time to locate and transfer a block = $s + rd + btt$ = 30+12.5+1 = 43.5 msec

(g) time to transfer 20 random blocks = 20 * (s + rd + btt) = 20 * 43.5 = 870 msec
time to transfer 20 consecutive blocks using double buffering = s + rd + 20*btt = 30 + 12.5 + (20*1) = 62.5 msec

Q 16.35: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

Problem 2

A file has r=20,000 STUDENT records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), MAJORDEPTCODE (4 bytes), MINORDEPTCODE (4 bytes), CLASSCODE (4 bytes, integer), and DEGREEPROGRAM (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk whose parameters are given in Problem 1 .

(a) Calculate the record size R in bytes.

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.

Block Size = 512 Bytes

(G) Record Size: fixes length; 9 fields

30 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 4 + 3

+ 1

= 113 Bytes

Addition Byte → Record Header

(b) Blocking factor: #Records / block

Block size : 512 B
Record " : 113 B

$\Rightarrow bfr = \left\lfloor \frac{512}{113} \right\rfloor$

$= 4$

(b) Blocking factor: #Records / block

Block size : 512 B
Record " : 113 B
} ⇒ bfr = ⌊512 / 113⌋ = 4 Records / Block

#Blocks for file : ∋ 20,000 / 4 = 5,000 Blocks

20,000 Records

Can we do this?

$$\#Blocks = \left\lceil \frac{Total\ File\ Size}{Block\ Size} \right\rceil$$

?!?

Can we do this? ⇒ NOOO

$$\#Blocks = \left\lceil \frac{\text{Total File Size}}{\text{Block Size}} \right\rceil$$ ✗

WRONG because of Unspanned Organization

$$\left\lceil \frac{20000 \times 113\,B}{512\,B} \right\rceil = 4415\,B$$

WRONG

Q:

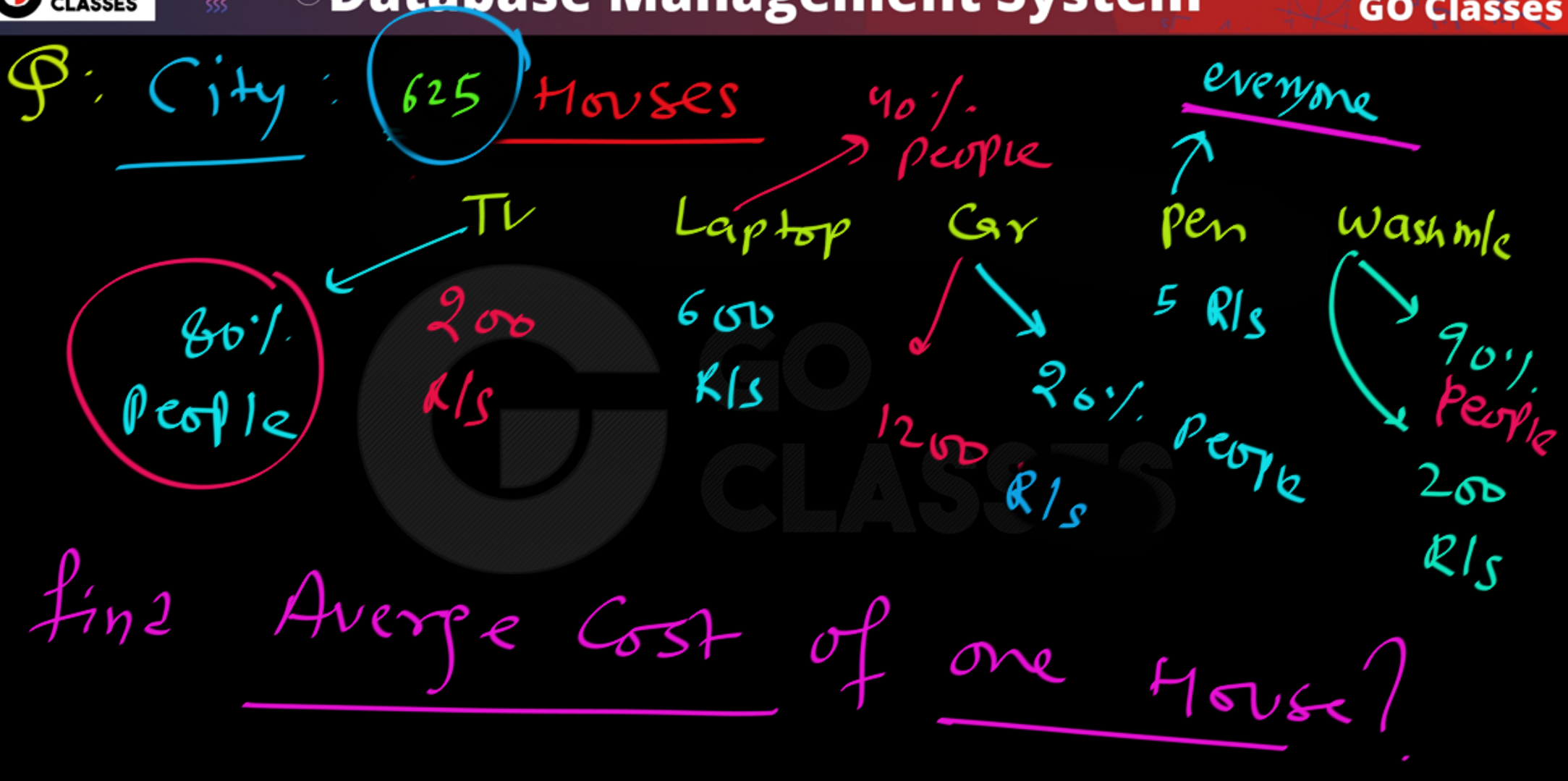
For variable-length records, different records have different sizes. Let the average record size in a file be R, and number of records in the file be r. Assume that Block size is B bytes, and K bytes are used for Block Header. Using spanned organization, find the number of blocks needed to store this file?

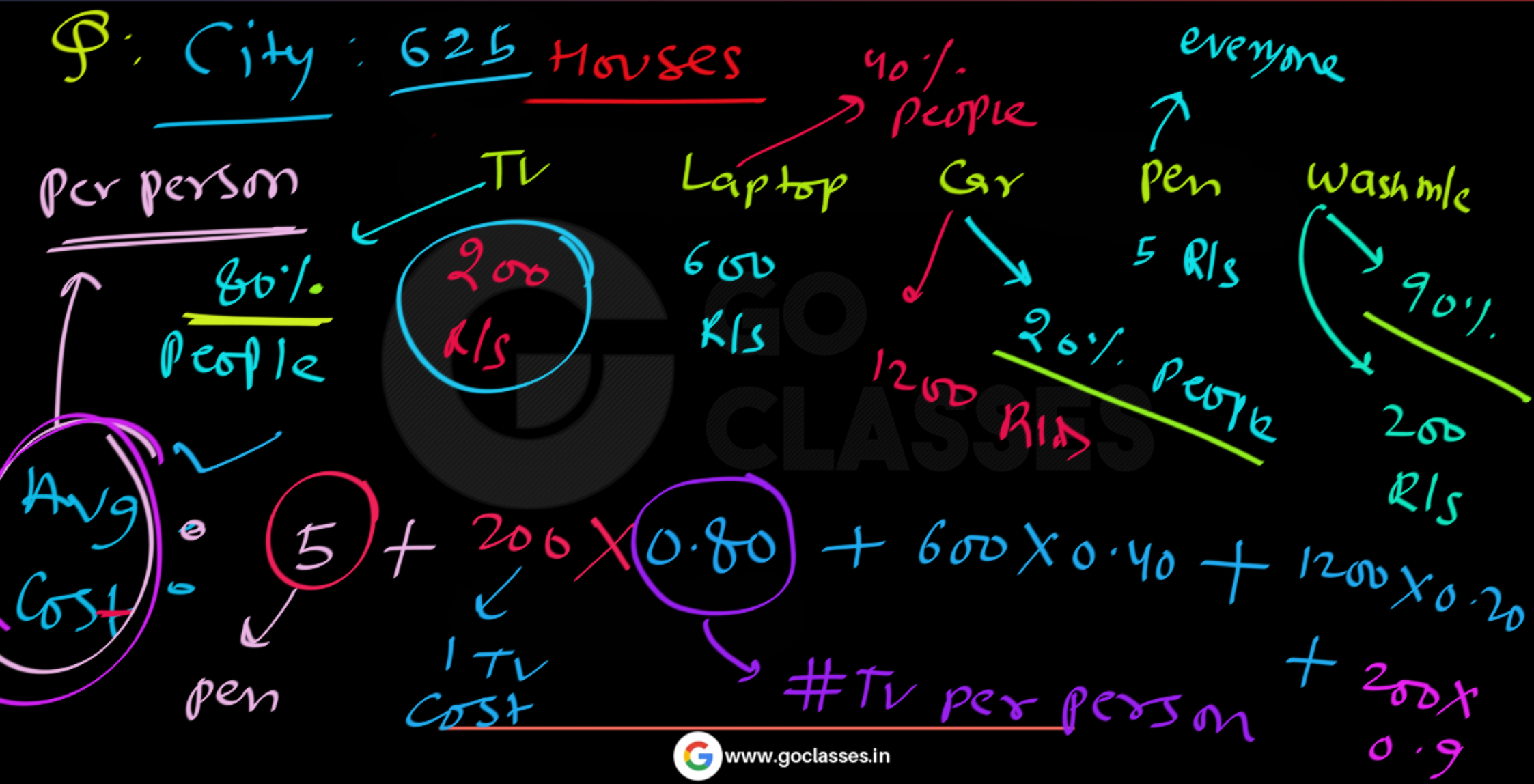
$bfr = \frac{B-k}{R}$ spanned org

#Blocks needed for file:

$$\left\lceil \frac{r}{bfr} \right\rceil = \left\lceil \frac{rR}{B-k} \right\rceil$$

rR → file size


P: City : 625 Houses
40% people
everyone
TV
Laptop
Car
Pen
Washmle
80% People
200 Rs
600 Rs
1200 Rs
20% people
5 Rs
90% People
200 Rs
Find Avergae Cost of one House?


P: City : 625 Houses
Per person
80% People
TV
200 Rs
Laptop
600 Rs
40% People
Car
1200 Rs
20% people
Pen
5 Rs
everyone
Washmle
90%
200 Rs
Avg Cost
=
5 + 200 X 0.80 + 600 X 0.40 + 1200 X 0.20 + 200 X 0.9
pen
1 TV Cost
TV per person

Average ⟶ find on one entity

⇒ 80% people have TV.

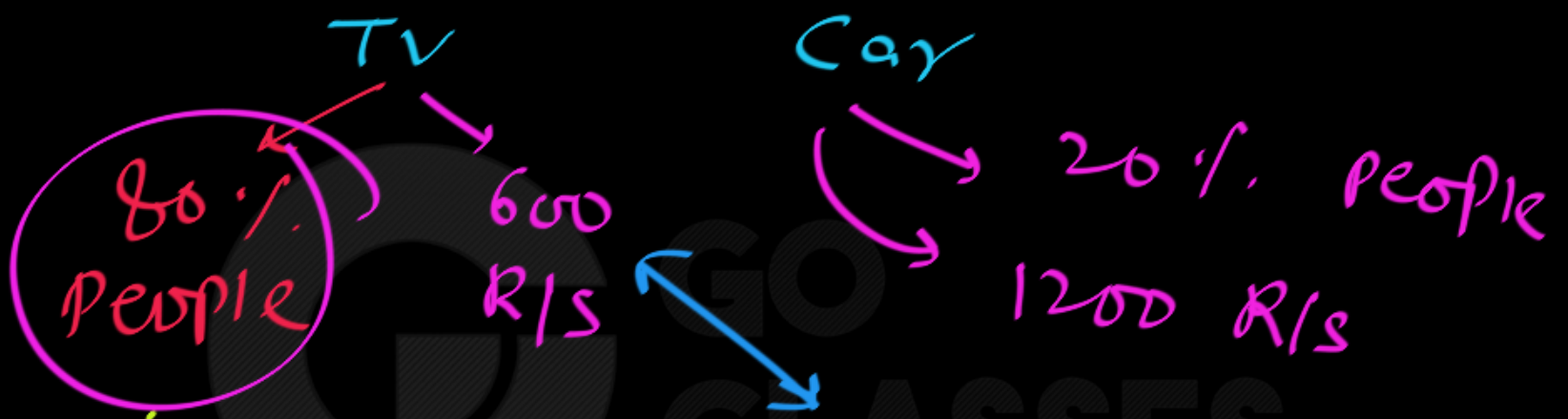
⇒ 1 person has How many TV? 0.8 TV

Q: 800 Houses;

If you have TV, 20 Rs Remote comes with it.

Average Cost per person?

Q: 800 Houses;



If you have TV, 20 Rs Remote comes with it.

Avg Cost per person = $1200 \times \underbrace{0.20}_{\#\text{Car per person}} + 0.80 \times (600+20)$

Q 16.36: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

16.36. Suppose that only 80% of the STUDENT records from Exercise 16.28 have a value for Phone, 85% for Major_dept_code, 15% for Minor_dept_code, and 90% for Degree_program; and suppose that we use a variable-length record file. Each record has a 1-byte *field type* for each field in the record, plus the 1-byte deletion marker and a 1-byte end-of-record marker. Suppose that we use a *spanned* record organization, where each block has a 5-byte pointer to the next block (this space is not used for record storage).

a. Calculate the average record length R in bytes.

b. Calculate the number of blocks needed for the file.

Block Size = 512 Bytes (STUDENT table information on next page)

A file has $r = 20{,}000$ STUDENT records of variable-length. Each record has the following fields: Name (30 bytes), Ssn (9 bytes), Address (40 bytes), PHONE (10 bytes), Birth_date (8 bytes), Sex (1 byte), Major_dept_code (4 bytes), Minor_dept_code (4 bytes), Class_code (4 bytes, integer), and Degree_program (3 bytes).

Q 16.36: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

16.36. Suppose that only 80% of the STUDENT records from Exercise 16.28 have a value for Phone, 85% for Major_dept_code, 15% for Minor_dept_code, and 90% for Degree_program; and suppose that we use a variable-length record file. Each record has a 1-byte *field type* for each field in the record, plus the 1-byte deletion marker and a 1-byte end-of-record marker. Suppose that we use a *spanned* record organization, where each block has a 5-byte pointer to the next block (this space is not used for record storage).

a. Calculate the average record length R in bytes.

b. Calculate the number of blocks needed for the file.

Block Header

Block Size = 512 Bytes (STUDENT table information on next page)

Students Table :

60% — Phone no.

85% — Major dept code

15% — Minor " "

90% — Degree " "

Average Record Size:

Variable-length

$= 30B + 9B + 40B + 0.80 \times (10+1)$

(name) (SSN) (Address)

$+1B \quad +1B \quad +1B$

$+ 8B + 1B + 0.85 \times (4+1)$

(BDAy) (set)

$+1B \quad +1B \quad + 0.15 \times (4+1)$

$0.90 \times (3+1) \quad + 4B + 1B +$

$+1B$

$+1B$

Record Header

Record Size : 0 Calculate from previous slide

=

#Blocks needed : $\left\lceil \frac{20,000 * 0}{512 - 5} \right\rceil$

Spanned or

File Organization

OVER...

Indexing

Next...