Mod 5 - Lecture 9:

Multi-Level Indexes

ISAM

- Deepak Poonia

(IISc Bangalore; *GATE AIR 53 & 67*)

Content of this Lecture:

1. Multilevel Indices
2. ISAM
3. Practice + GATE 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.

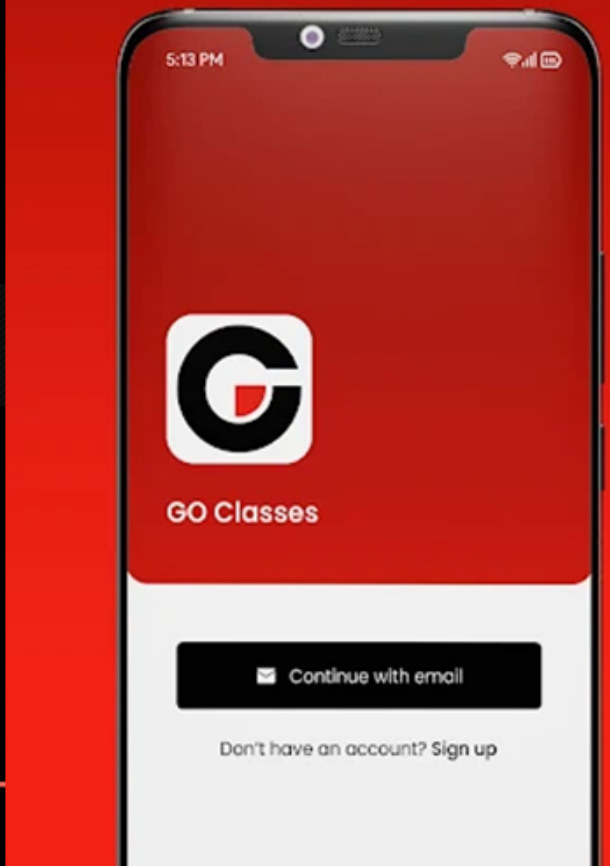
Visit here to watch : https://www.goclasses.in/s/store/

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

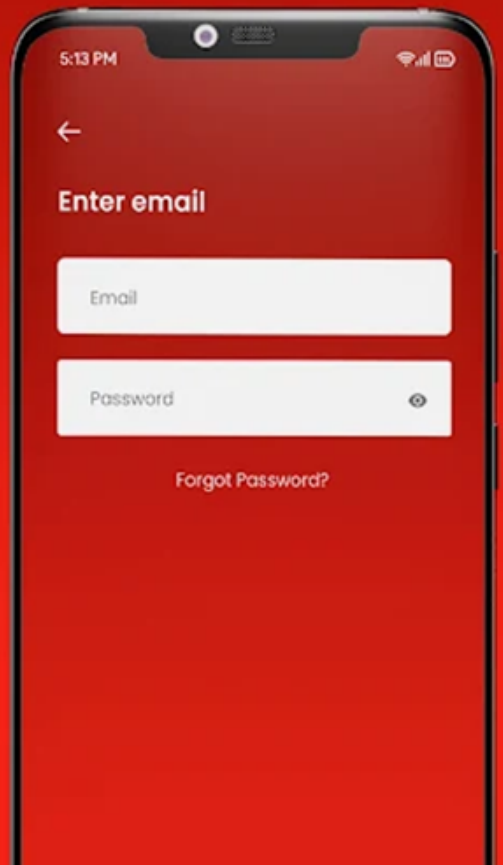
Download the GO Classes Android App:

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

Search "GO Classes" on Play Store.

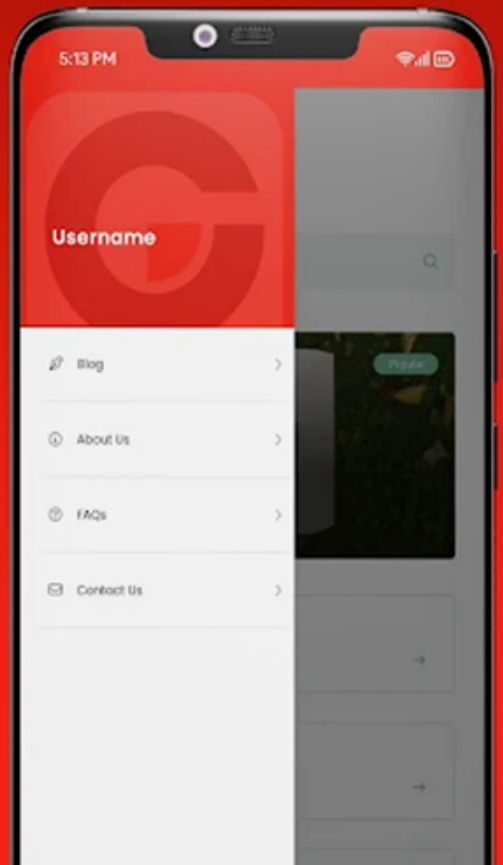


Welcome to
GO Classes
Learn at your own pace



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

Types of Single level Ordered Indexes:

Summary

Index Structures

Index: A disk data structure
– enables efficient retrieval of a record
given the value (s) of certain attributes
– indexing attributes

Primary Index:
Index built on *ordering key* field of a file

Clustering Index:
Index built on *ordering non-key* field of a file

Secondary Index:
Index built on any *non-ordering* field of a file

17.1.4 Summary

To conclude this section, we summarize the discussion of index types in two tables. Table 17.1 shows the index field characteristics of each type of ordered single-level index discussed—primary, clustering, and secondary. Table 17.2 summarizes the properties of each type of index by comparing the number of index entries and specifying which indexes are dense and which use block anchors of the data file.

Table 17.1 Types of Indexes Based on the Properties of the Indexing Field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.2 Properties of Index Types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Next Topic:

Ordered Indexes

Multi-Level

So far:

Index is ordered

Single level ordered Indexes { PI, CI, SI [on key, on Non key]

I index Blocks, B Data Blocks

Access Cost (I/O Cost): $\lceil \log_2 I \rceil + 1$

Binary Search on Index

for Data Block

I/O Cost : $\lceil \log_2 I \rceil + 1$ → binary search

To Reduce I/O Cost :

(1) Put Index in main memory

Problem 1 : Index larger than main memory

Problem 2 : Even if Index Size < mm size

then  → mm has many things to accomodate

So mm may not be able to take index

I/O Cost : $\lceil \log_2 I \rceil + 1$ → binary search

Data Block

To Reduce I/O Cost :

(1) Put Index in main memory : ⇒ I/O Cost : 1

Good only when Index Size is very small

at most 2-3 Index Blocks

I/O Cost : $\lceil \log_2 I \rceil + 1$ → binary search

To Reduce I/O Cost :

(1) Put Index in main memory
↳ may be okay for Primary Index → Small size

(2) Multilevel Index ✓
Reduce it

+1 → for Data block

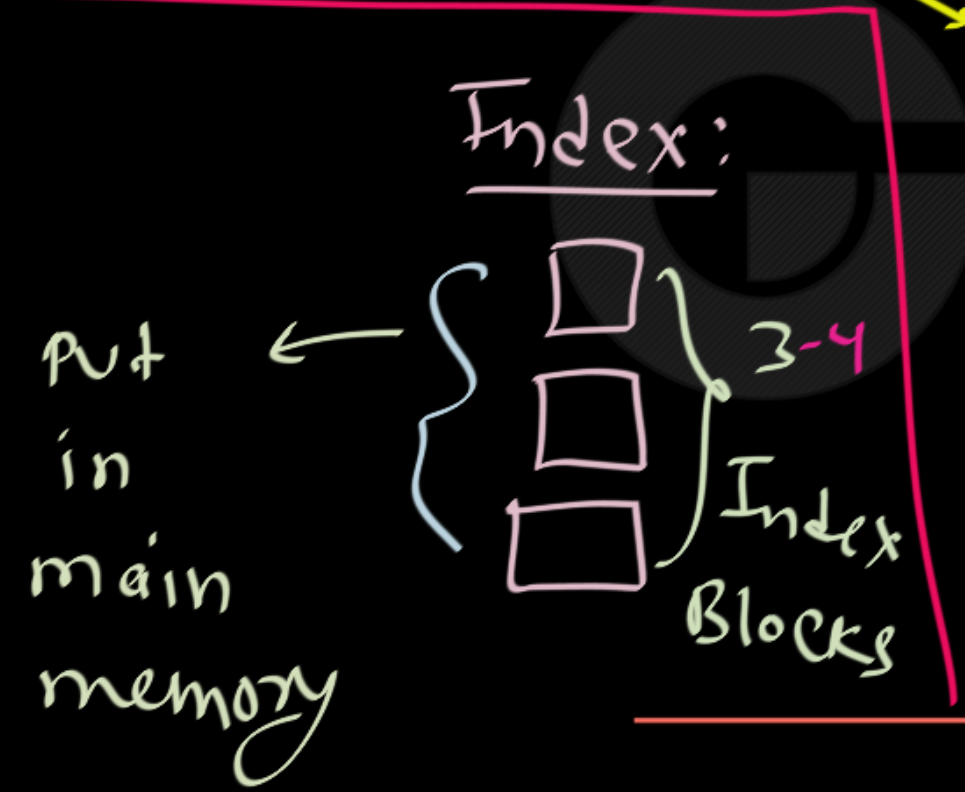
Q: we go for multilevel index because index doesn't fit in one block?

→ Reason for multilevel index??

Q: we go for multilevel index because 'index doesn't fit in one block ?

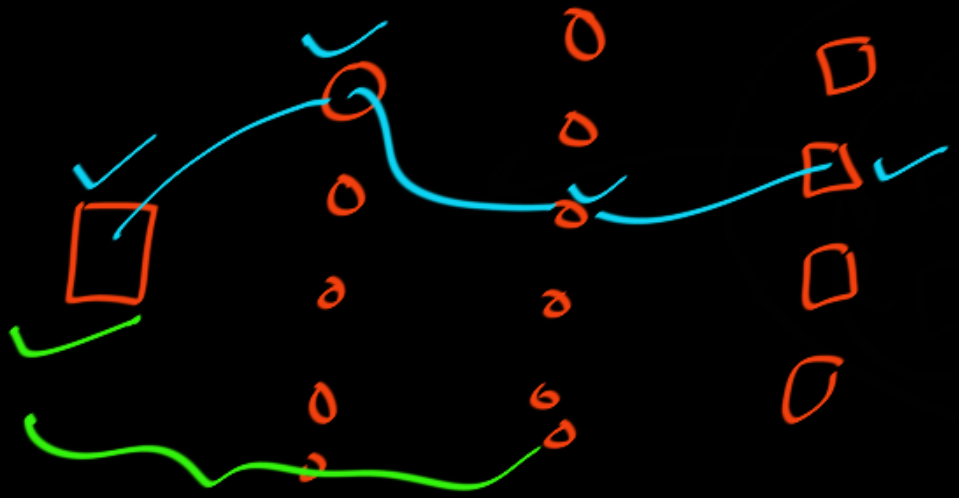
No

Reason for multilevel index ??



No

Once we Decide to go for multilevel Index then by default Outer most level has one Block.



I/o cost = 4 = 3 + 1

3 → Index levels

1 → Data Block

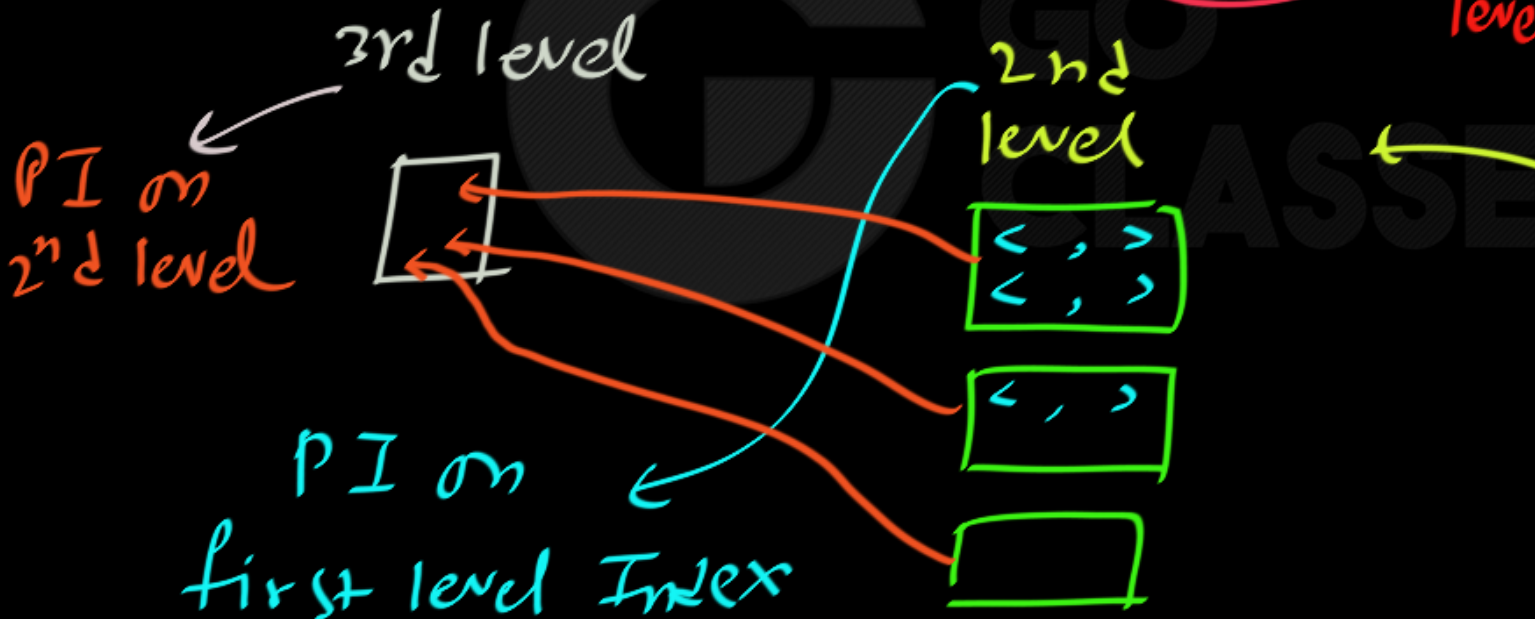
17.2 Multilevel Indexes

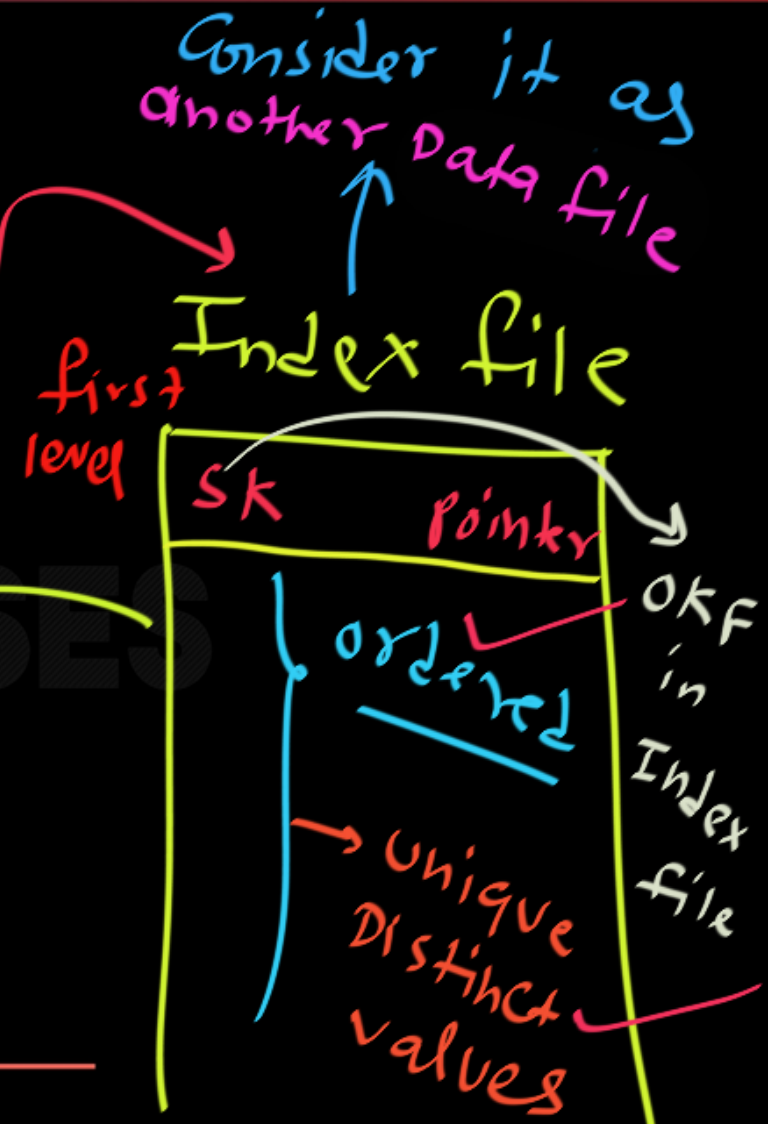
The indexing schemes we have described thus far involve an ordered index file. A binary search is applied to the index to locate pointers to a disk block or to a record (or records) in the file having a specific index field value. A binary search requires approximately ($\log_2 b_i$) block accesses for an index with b_i blocks because each step of the algorithm reduces the part of the index file that we continue to search by a factor of 2. This is why we take the log function to the base 2.

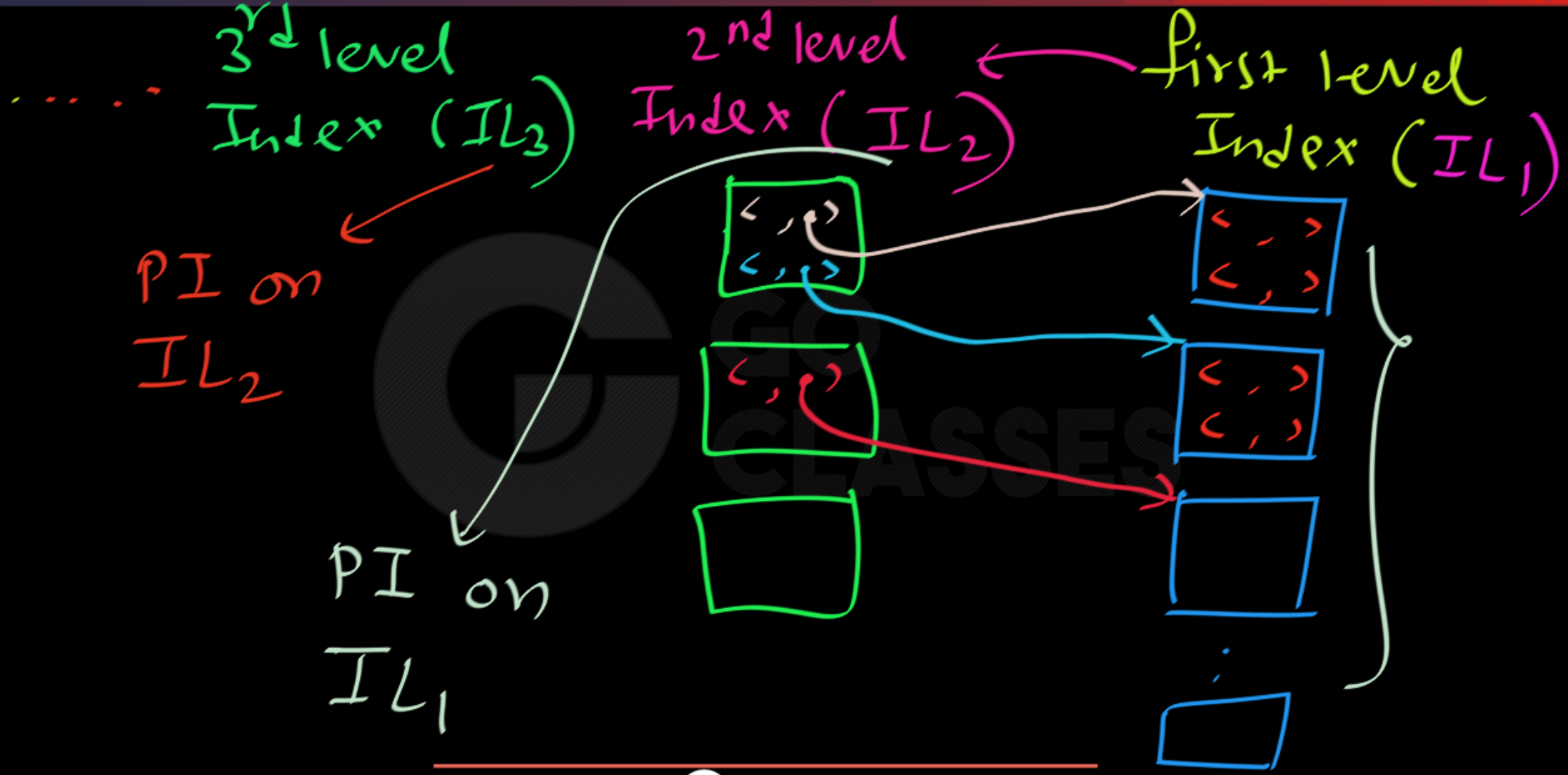
If an index is small enough to be kept entirely in main memory, the search time to find an entry is low. However, if the index is so large that not all of it can be kept in memory, index blocks must be fetched from disk when required. (Even if an index is smaller than the main memory of a computer, main memory is also required for a number of other tasks, so it may not be possible to keep the entire index in memory.) The search for an entry in the index then requires several disk-block reads.

Single level Indexes :

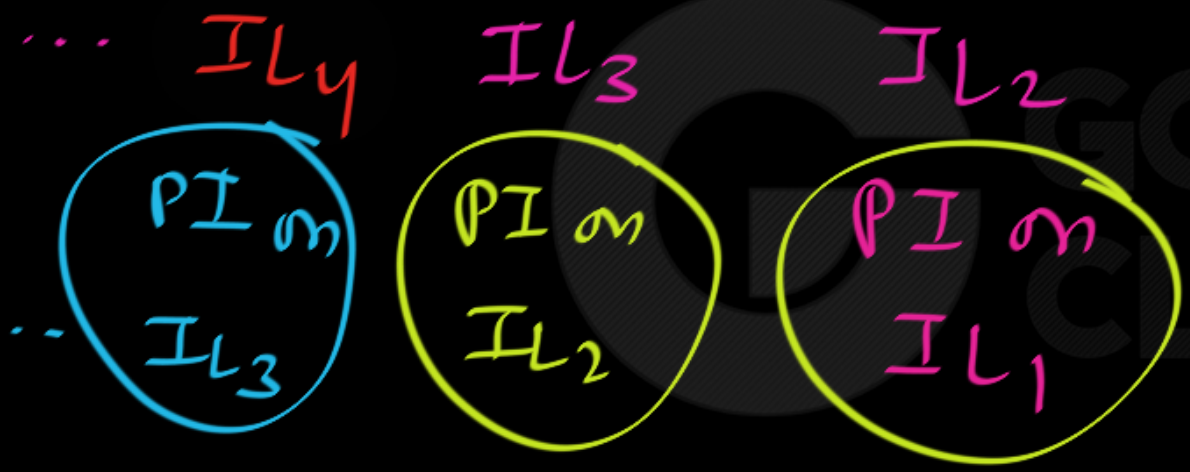
Primary Index, CI, SI

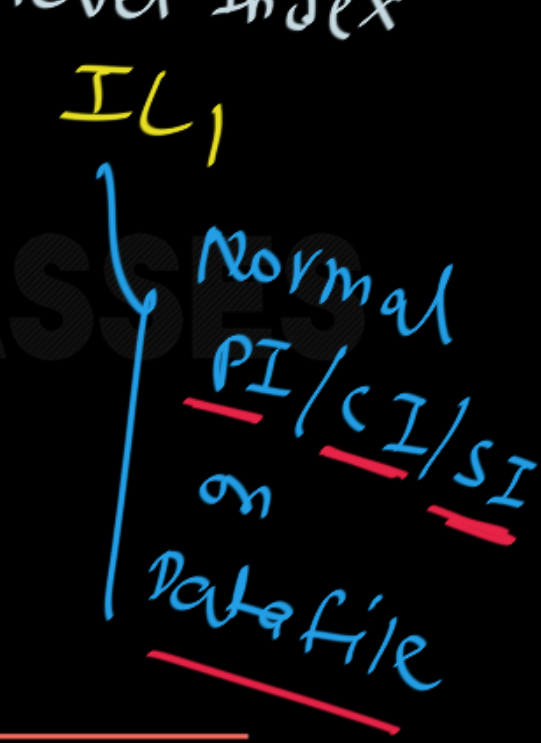


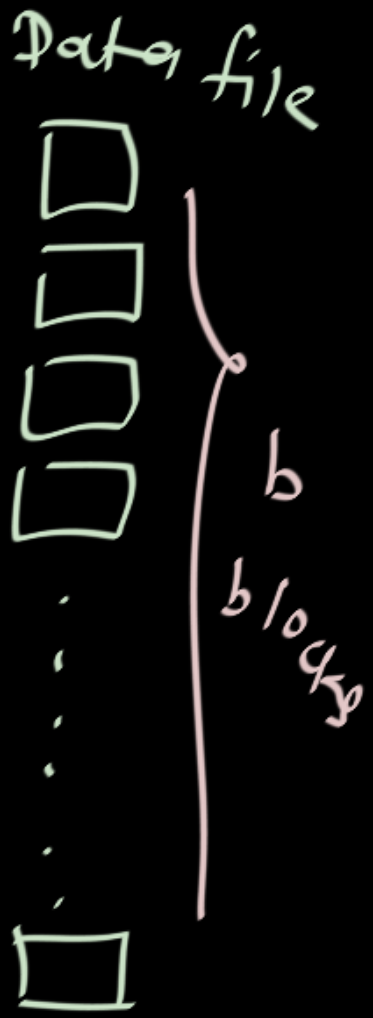



3rd level
Index (IL3)
2nd level
Index (IL2)
first level
Index (IL1)
PI on
IL2
PI on
IL1

MLI : (multilevel Index) :







MLI : (multilevel Index) :

Note : 2nd level and greater are ALWAYS SPARSE

ALWAYS

MLI : (multilevel Index) :

Note : 2^{nd} level and greater are ALWAYS SPARSE

Note : Index level K is PI (sparse) on Index level K-1.

MLI : (multilevel Index) :

Note : 2nd level and greater are ALWAYS SPARSE

Note : Index level 1 is Normal → PI / CI / SI on Data file

MLI : (multilevel Index) :

Note : 2nd level and greater are ALWAYS SPARSE

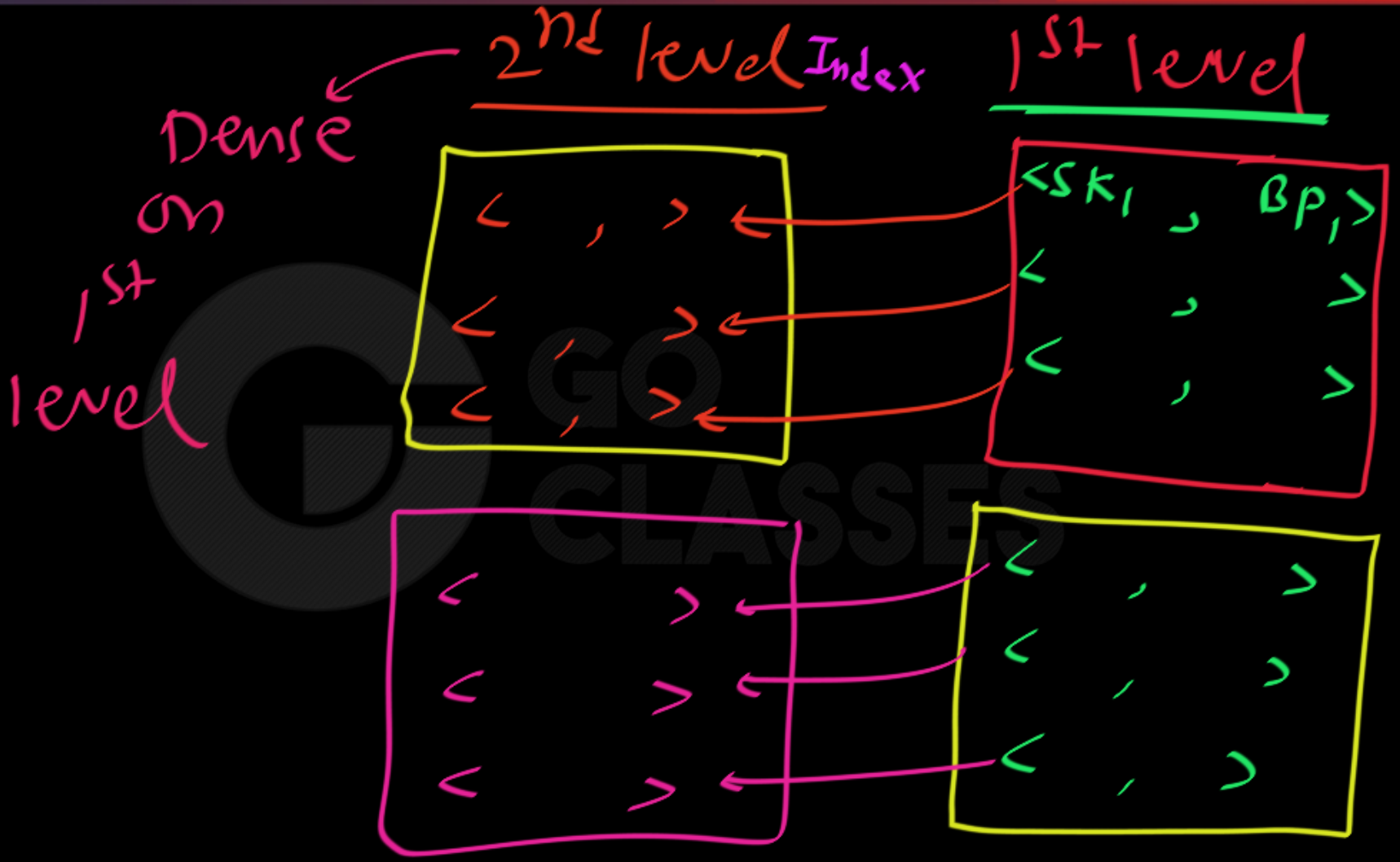
Note : Last (outermost) level is called Top level

Innermost level is 1st level

Q: Assume 1st level is PI on Datafile & 2nd level is Dense index on 1st level then Problem ??

Q: Assume 1st level is PI on Datafile & 2nd level is Dense index on 1st level then Problem ??

All Index levels Copy on each other

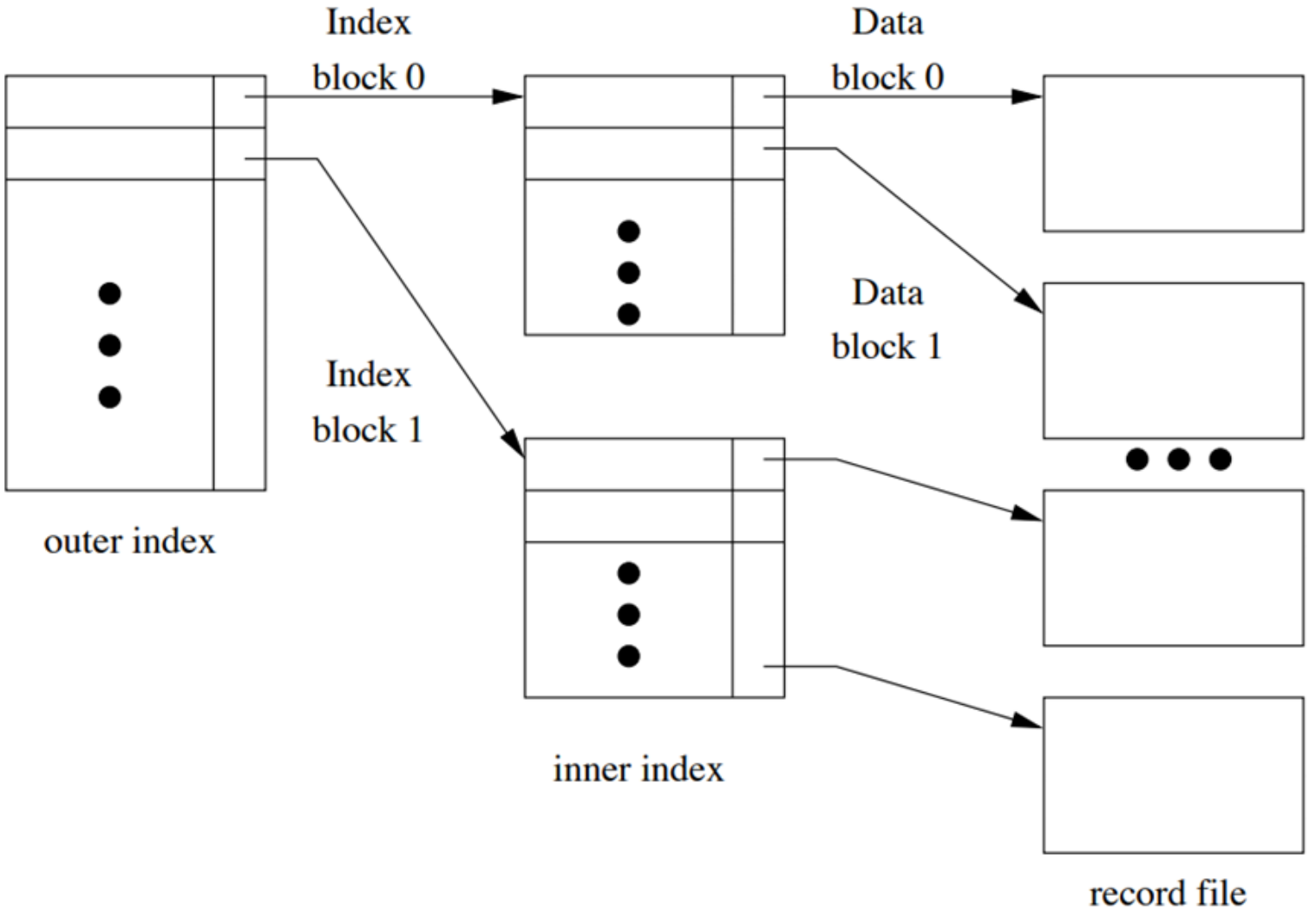

2nd level Index
1st level
Dense on 1st level
<SK1, BP1>

Multi-Level Index

- If primary index does not fit in memory, access to records becomes expensive
- To reduce number of disk accesses to index entries, treat primary index on disk as sequential file and construct a sparse index on it.
  - outer index $\longrightarrow$ a sparse index of primary index
  - inner index $\longrightarrow$ the primary index file

- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.

• Multilevel Index structure



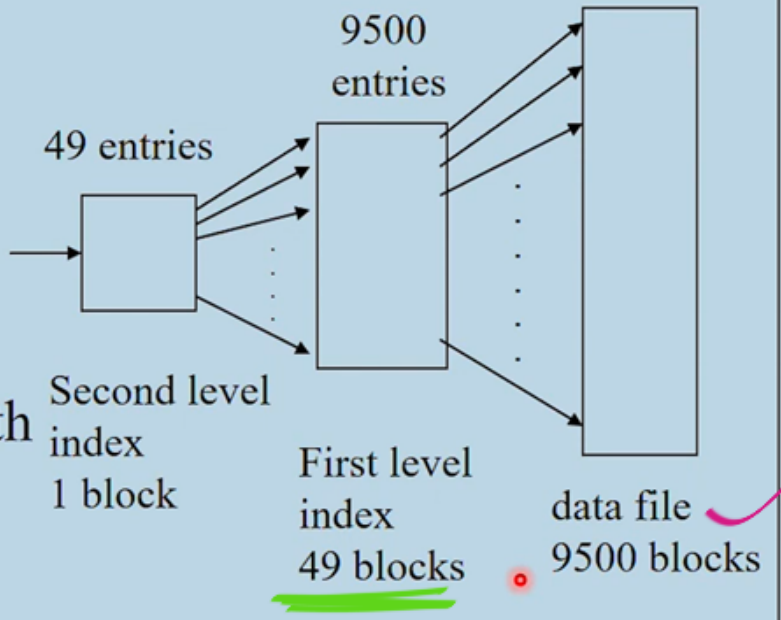
Making the Index Multi-level

Index file – itself an ordered file
– another level of index can be built

Multilevel Index –
Successive levels of indices are built till the last level has one block

height – no. of levels
block accesses: height + 1
(no binary search required)

For the example data file:
No of block accesses required with
multi-level primary index: 3
without any index: 14



Q: If Data file is onLevel : # Data Blocks

(1) Approach1 : without index :

I/O Cost = $\lceil \log_2 9500 \rceil = 14$

(2) App2 : With simple level index :

I/O Cost = $\lceil \log_2 I \rceil + 1 = 6 + 1 = 7$

I = 49

(3) App3 : MLI : 2 + 1 = 3

Note: MLI is better choice than Binary Search on Single level.

FUNDAMENTALS OF Database Systems

7th Edition Ramez Elmasri, Shamkant B. Navathe

Exercise 17.18

(Multi part question)

Q 17.18: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

17.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

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: $b = 512$ Bytes $= 512$ B

Block pointer $= p = 6$ Bytes $= 6$ B

Record " $= p_R = 7$ B

#file Records $= r = 30,000$

Record Size $= 30 + 9 + 9 + 40 + 10 + 9 + 1 + 4 + 4 + 1$

$= 116$ B (Fixed length)

Blocking factor of file : bfr :

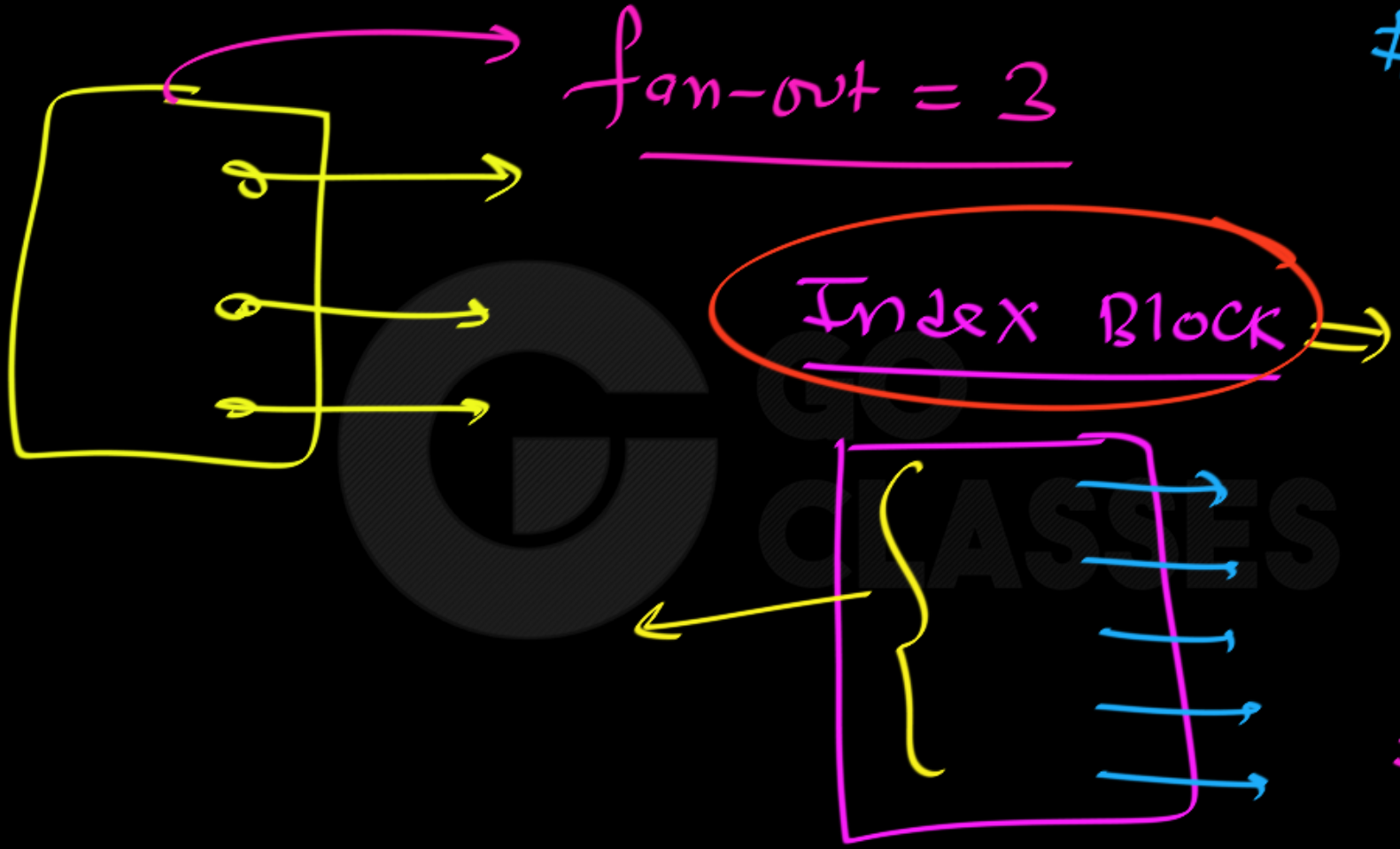
#Records / Block

$$\left\lfloor \frac{512\ B}{111\ B} \right\rfloor = 4$$

4 in one Block

#File Blocks needed :

$$\left\lceil \frac{30,000}{4} \right\rceil = 7500 \text{ Blocks}$$


fan-out = 3
Index Block
#Index Records per Block
Blocking factor
≡ fan-out
≡ fo

Q 17.18: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

17.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

c. Suppose that the file is *ordered* by the key field Ssn and we want to construct a *primary index* on Ssn. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out *fo*); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the primary index.

Block Size: $b = 512$ Bytes $= 512$ B

Block pointer $= p = 6$ Bytes $= 6$ B

Record " $= p_R = 7$ B

#file Records $= r = 30,000$

Record Size $= 30 + 9 + 9 + 40 + 10 + 9 + 1 + 4 + 4 + 1$

$= 116$ B (Fixed length)

File Blocking factor $= 4$; #file Blocks $= 7500$

Second level

Index Record Size
$= 9 + 6 = 15\ B$

Sk size → SSN

BP size

$\left\lfloor \frac{512\ B}{15\ B} \right\rfloor = 34$

Blocking factor of IL2 = 34

1st level Index

Index Record Size
$= 9 + 6 = 15\ B$

PI on Data file

IL1 Blocking factor

3rd level

#IL3 entries = 7 ✓

Blocking factor = 34 ✓ of IL_3

2nd level

#IL2 Entries = 221

#IL_2 Blocks = $\lceil \frac{221}{34} \rceil = 7$

1st level

#IL_1 entries = 7500 ✓

#IL_1 Blocks $\lceil \frac{7500}{34} \rceil = 221$

(i) 34

(ii) 7500 ; 221

(iii) 3 Index level

(iv) 1+7+221 = 229

(v) Search on SSN : I/O Cost = 3 + 1 = 4 ✓

on SSN, we have MLI

for 'index level'

Data Block

Search on Address = I/O Cost ⇒ Avg = $\frac{7500}{2}$ = 3750 ✓

Q 17.18: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

17.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

d. Suppose that the file is *not ordered* by the key field Ssn and we want to construct a *secondary index* on Ssn. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

Block Size: $b = 512$ Bytes $= 512$ B

Block pointer $= p = 6$ Bytes $= 6$ B

Record " $= p_R = 7$ B

#file Records $= r = 30,000$

Record Size $= 30 + 9 + 9 + 40 + 10 + 9 + 1 + 4 + 4 + 1$

$= 116$ B (Fixed length)

File Blocking factor $= 4$; #file Blocks $= 7500$

① Index Entry size:

first level index :⇒ ILI = SI on key

↳ 9 + 7 = 16 B

9 → SSN

7 → Record pointer

SI on key → Record pointer or Block pointer

by Default

2nd & Higher levels

↳ Index Record size = 9+6 = 15

(2) Blocking factor of IL1 :

$$\left\lfloor \frac{512\ B}{16\ B} \right\rfloor = 32$$

Blocking factor of IL2 & Higher levels :

Index Records Per Block

$$\left\lfloor \frac{512\ B}{15\ B} \right\rfloor = 34$$

(3) # first level Entries : # Records in file

Index = 30,000

SI on key

IL, Blocks =

$$\left\lceil \frac{30,000}{32} \right\rceil = 938$$

(3) # 2nd level Entries : # Blocks on IL_1

↓ Index = 938

$$\#IL_2 \text{ Blocks} = \left\lceil \frac{938}{34} \right\rceil = 28$$

3^{rd} level index entries = # Blocks on 2^{nd} level

= 28

Simple Block

level in MLI = 3

Index Blocks (total) = 1 + 28 + 938

= 967

(4) Access Cost:

Search on SSN:

3 + 1 → Index levels → Data Block

= 4

14.1.3 Secondary Indexes

- A secondary index is also an ordered file with two fields.
- The first field of the index is of the same data type as some *nonordering field* of the data file that is an indexing field.
- The second field of the index is either a block pointer or a record pointer.

(170 points) Consider a disk with block size B=512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 7 bytes long. A file has r = 30,000 EMPLOYEE records of fixed length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), DEPARTMENTCODE (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), JOBCODE (4 bytes), SALARY (5 bytes, real number).

4. (50 points) Suppose the file is **not ordered** by the key field SSN and we want to construct a **secondary index** on SSN. Repeat the previous exercise (part 3) for the secondary index and compare with the primary index. Assumption is the first level indexes use the **record pointers** and the higher-level indexes use the **block pointers**.

   a) the index blocking factor bfr (which is also the index fan-out fo)

   b) the number of first-level index entries and the number of first-level index blocks

   c) the number of levels needed if we make it into a multi-level index.

   d) the total number of blocks required by the multi-level index.

   e) the number of block accesses needed to search for and retrieve a record from the file--given its SSN value--using the secondary index.

Q 17.18: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

17.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

Part e: On Next Page

e. Suppose that the file is *not ordered* by the nonkey field Department_code and we want to construct a *secondary index* on Department_code, using option 3 of Section 17.1.3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out *fo*); (ii) the number of blocks needed by the level of indirection that stores record pointers; (iii) the number of first-level index entries and the number of first-level index blocks; (iv) the number of levels needed if we make it into a multilevel index; (v) the total number of blocks required by the multilevel index and the blocks used in the extra level of indirection; and (vi) the approximate number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the index.

SI on Non-key:

Three implementations

Variable length records

I_1 : Not preferred

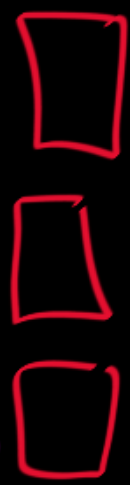
I_2 :

20 BP_1
20 BP_2
20 BP_3

I_3 : by implementation

(SK, BP_1, BP_2, BP_3)

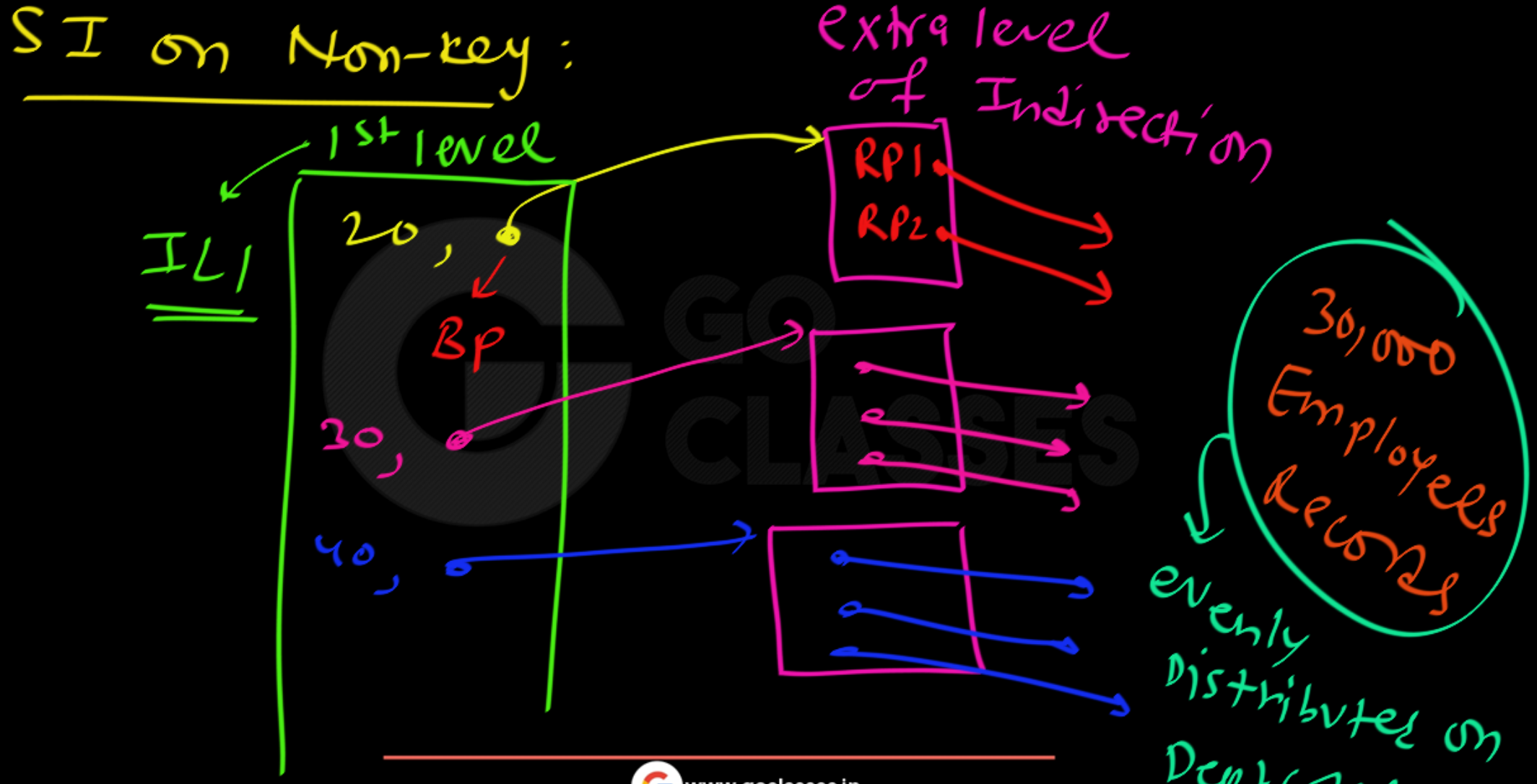
New level of indirection



PI, CI, SI

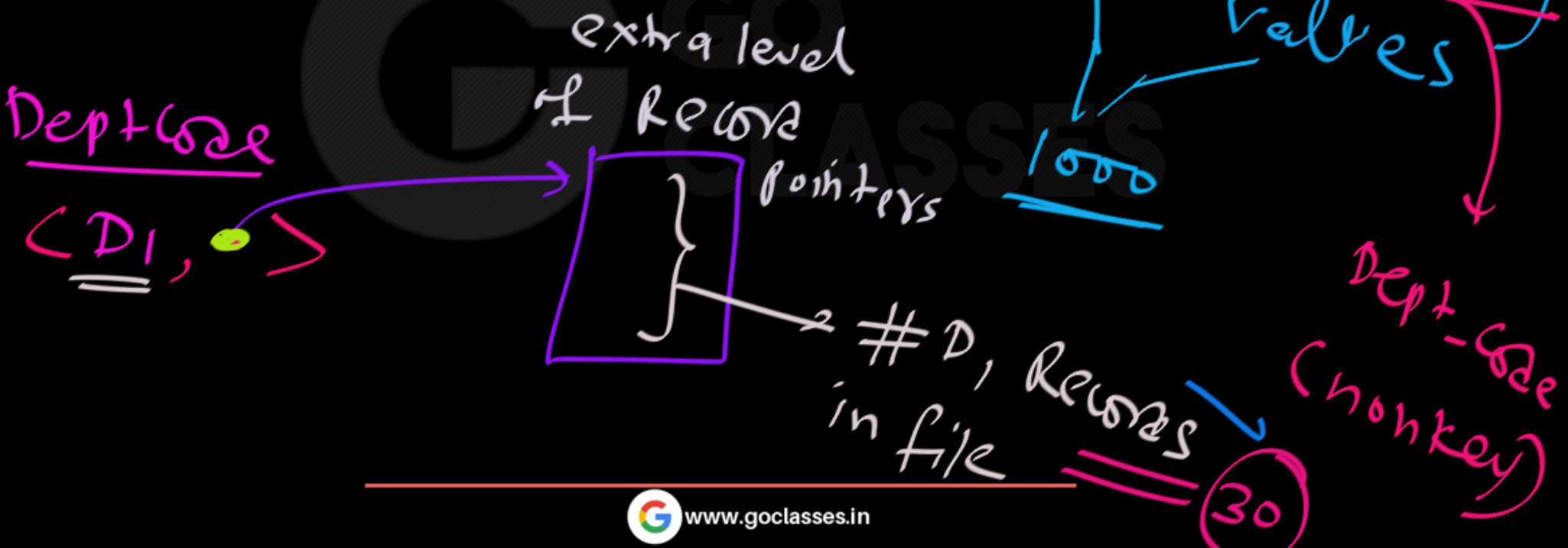
↳ on 1st level :

2nd → PI on IL1


SI on Non-key:
1st level
IL1
20,
Bp
30,
40,
Extra level of Indirection
RP1
RP2
30,000 Employees Records
evenly Distributed on Depts

1st level Index:

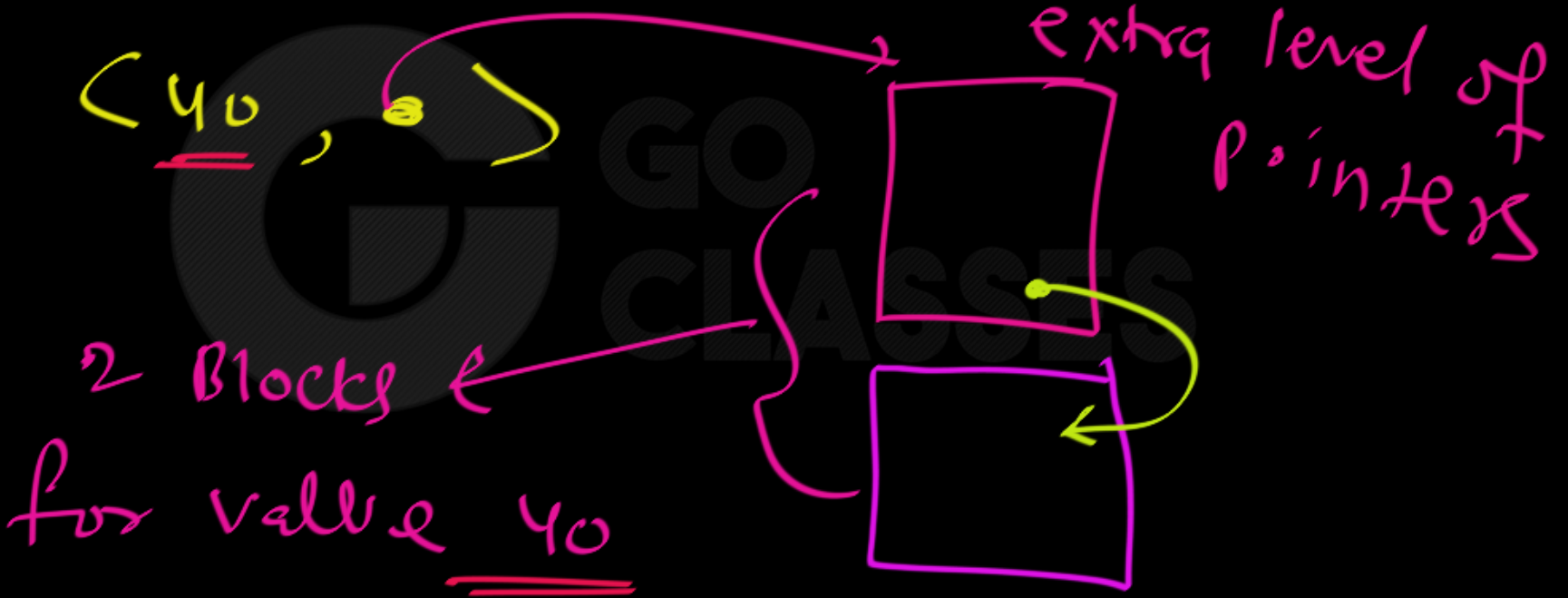
Index Entries = # Distinct search key values



Can we put 30 Record Points in a single Block ? ⇒ Yes ✓

max #RP Per Block = $\left\lfloor \frac{512\ B}{7\ B} \right\rfloor$ = 73

By Chance : SI on Monkey :



Coming Back to Question: GV

#Blocks in the Extra level of indirection = 1000 ✓

1st level of Index: IL1:

#Index entries = 1000

IL1 entry size = 9B + 6B = 15B

for Dept_Code size

Block Pointer size

for every level of Index:

Index entry size = 15B

Blocking factor = $\lfloor \frac{512}{15} \rfloor = 34$

1st level index Blocks = $\left\lceil \frac{1000}{34} \right\rceil = 30$

2nd level #entries = 30

2nd level # Blocks = $\left\lceil \frac{30}{34} \right\rceil = 1$

#levels = 2 + 1 ⟹ levels of index

Extra level for indirection

(i) 34 (ii) 1000 (iii) 1000 ; 30

(iv) 2 + 1 → extra level of indirection

Unordered ← for a specific dept (say 30 employee records)

(v) 1 + 30 + 1000 = 1031

(vi) 2 + 1 + (worst case : All 1000 Records in different Blocks = 1000 file Blocks)

↓ extra level of indirection

33 (worst case)

Q 17.18: FUNDAMENTALS OF Database Systems 7th Ed, Navathe, Elmasri

17.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

Part f: On Next Page

f. Suppose that the file is *ordered* by the nonkey field Department_code and we want to construct a *clustering index* on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the clustering index (assume that multiple blocks in a cluster are contiguous).

HW

Multi-level Secondary Indexes

Secondary indexes can also be converted to multi-level indexes

First level index

– as many entries as there are records in the data file

First level index is an ordered file

so, in the second level index, the number of entries will be equal to the number of *blocks* in the first level index rather than the number of *records*

Similarly in other higher levels

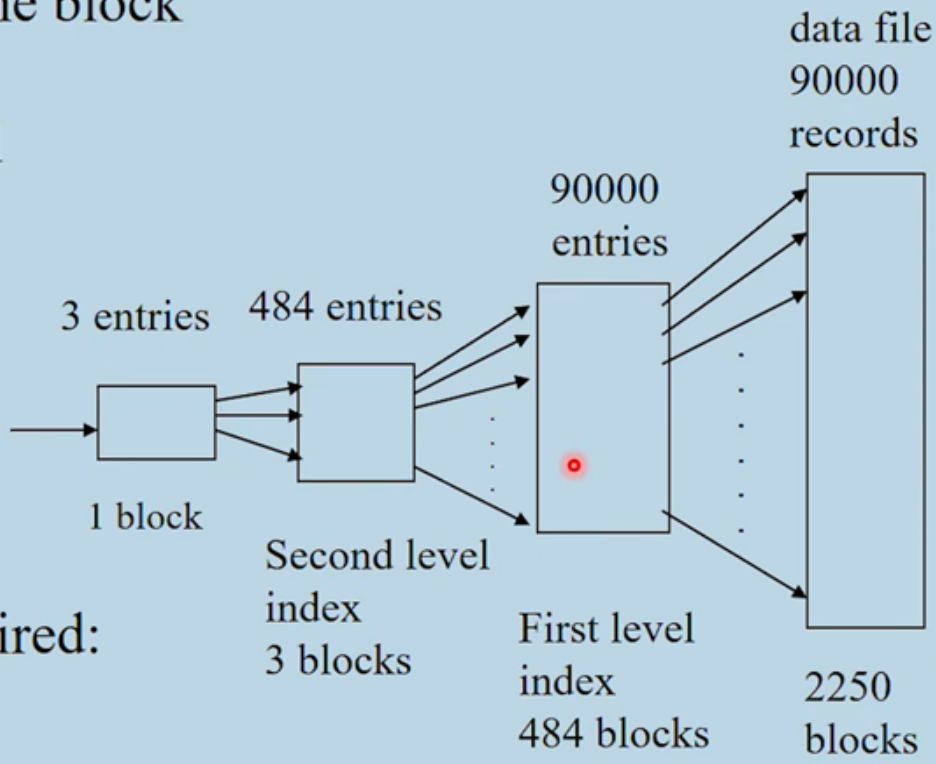
Making the Secondary Index Multi-level

Multilevel Index –

Successive levels of indices are built till the last level has one block

height – no. of levels

block accesses: height + 1



For the example data file:

No of block accesses required:

multi-level index: 4

single level index: 10

A multilevel index considers the index file, which we will now refer to as the **first** (or **base**) **level** of a multilevel index, as an *ordered file* with a *distinct value* for each

$K(i)$. Therefore, by considering the first-level index file as a sorted data file, we can create a primary index for the first level; this index to the first level is called the **second level** of the multilevel index. Because the second level is a primary index, we can use block anchors so that the second level has one entry for *each block* of the first level. The blocking factor bfr_i for the second level—and for all subsequent levels—is the same as that for the first-level index because all index entries are the same size; each has one field value and one block address. If the first level has r_1 entries, and the blocking factor—which is also the fan-out—for the index is $bfr_i = fo$, then the first level needs $\lceil (r_1/fo) \rceil$ blocks, which is therefore the number of entries r_2 needed at the second level of the index.

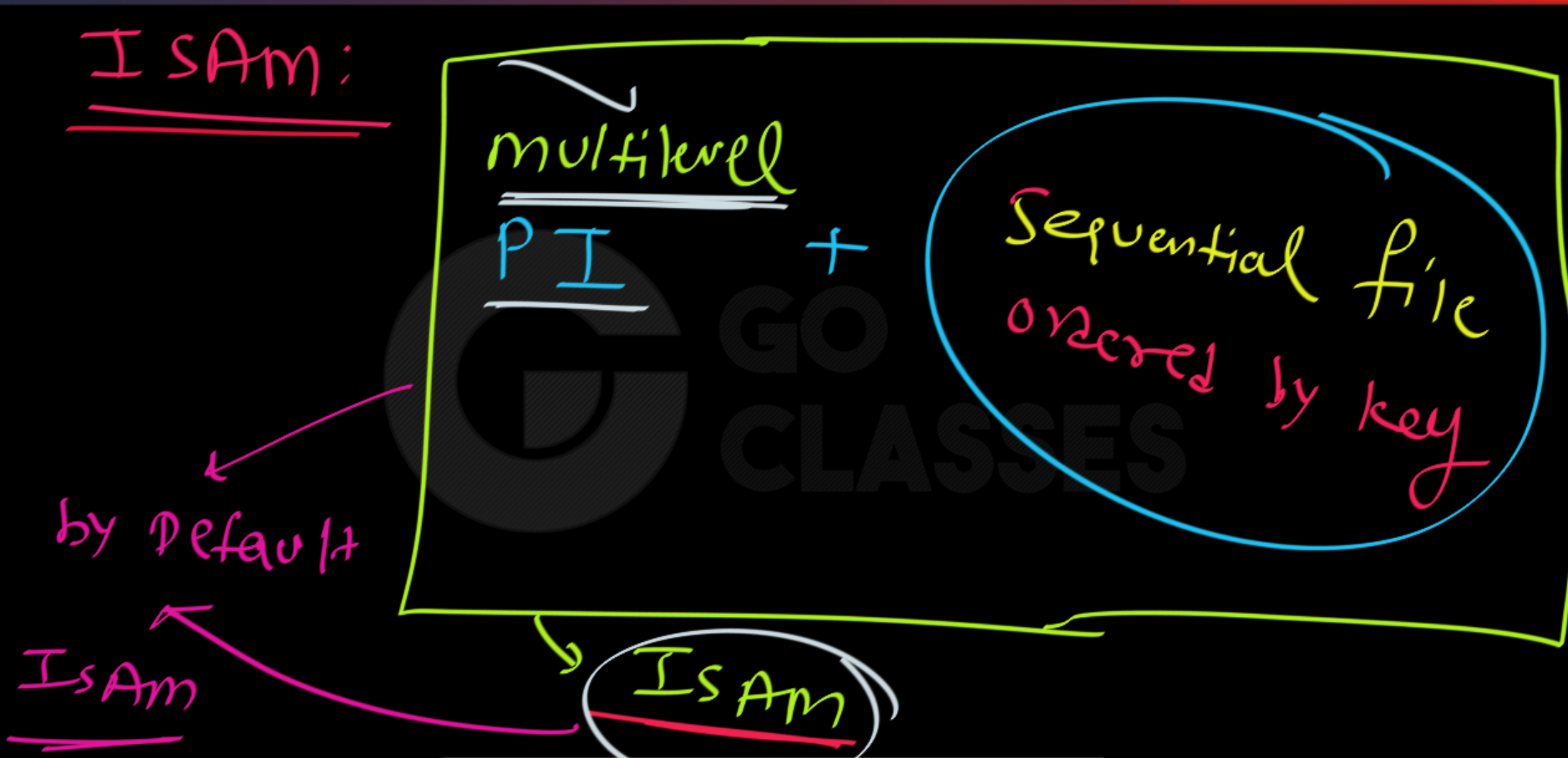
We can repeat this process for the second level. The **third level**, which is a primary index for the second level, has an entry for each second-level block, so the number of third-level entries is $r_3 = \lceil (r_2/fo) \rceil$. Notice that we require a second level only if the first level needs more than one block of disk storage, and, similarly, we require a third level only if the second level needs more than one block. We can repeat the preceding process until all the entries of some index level t fit in a single block. This block at the tth level is called the **top** index level.[6] Each level reduces the number of entries at the previous level by a factor of fo—the index fan-out—so we can use the formula $1 \leq (r_1/((fo)^t))$ to calculate t. Hence, a multilevel index with r_1 first-level entries will have approximately t levels, where $t = \lceil (\log_{fo}(r_1)) \rceil$. When searching the index, a single disk block is retrieved at each level. Hence, t disk blocks are accessed for an index search, where t is the *number of index levels*.

The multilevel scheme described here can be used on any type of index—whether it is primary, clustering, or secondary—as long as the first-level index has *distinct values for K(i) and fixed-length entries*. Figure 17.6 shows a multilevel index built over

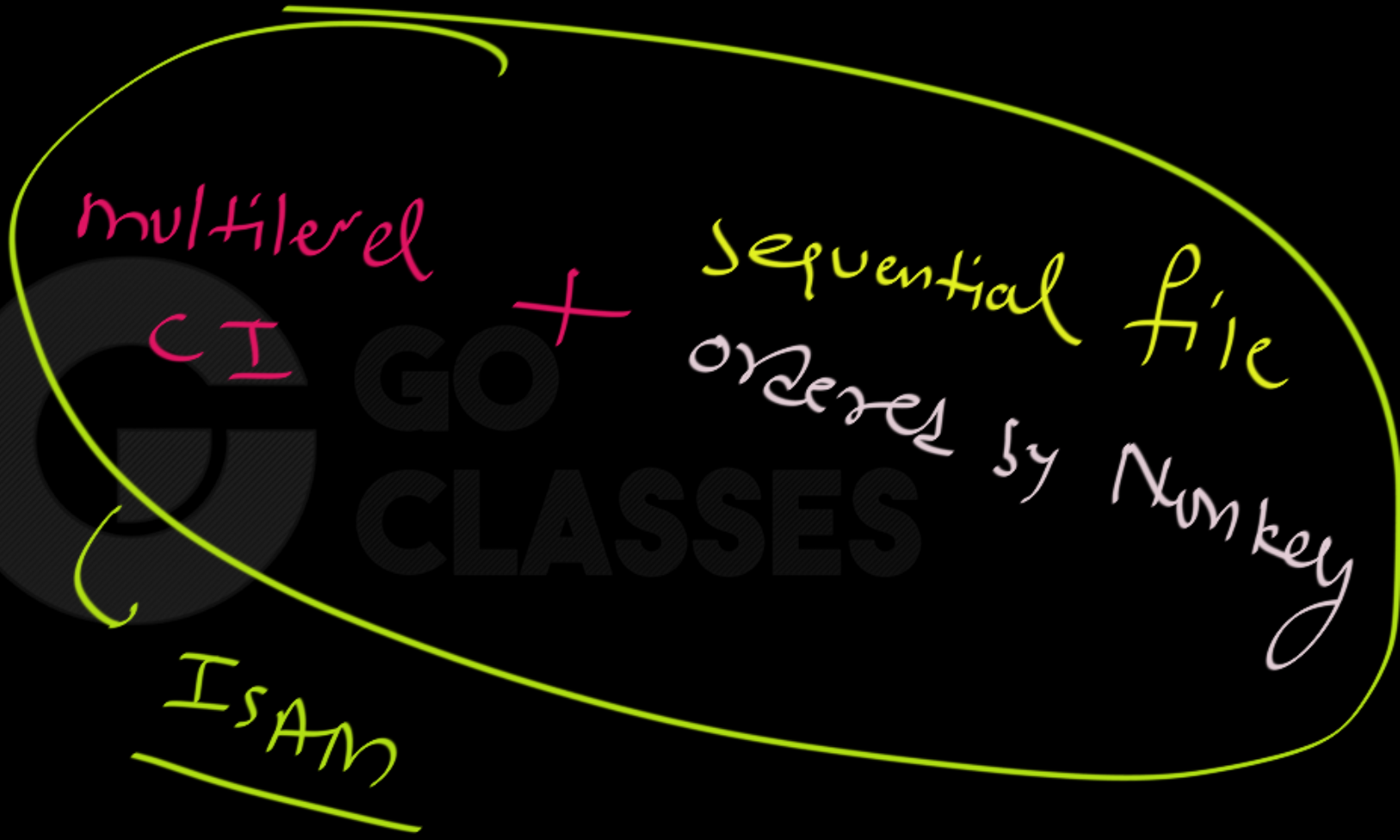
Next Topic:

ISAM

Indexed sequential access method


ISAM:
multilevel
PI
+
Sequential file
ordered by key
by Default
IsAm
IsAm

ISAM:



A common file organization used in business data processing is an ordered file with a multilevel primary index on its ordering key field. Such an organization is called an **indexed sequential file** and was used in a large number of early IBM systems.

ISAM

ISAM file : ordered file with multilevel primary Index

A common file organization used in business data processing is an ordered file with a multilevel primary index on its ordering key field. Such an organization is called an **indexed sequential file** and was used in a large number of early IBM systems.

ISAM

ISAM file : A file with multilevel primary Index

Index Sequential Access Method (ISAM) Files

ISAM files –

Ordered files with a multilevel primary/clustering index

Most suitable for files that are relatively static

If the files are dynamic, we need to go for dynamic multi-level index structures based on B^+- trees

NPTEL IITM

Q: file Data information

ISAM

Indexed - Sequential Data file

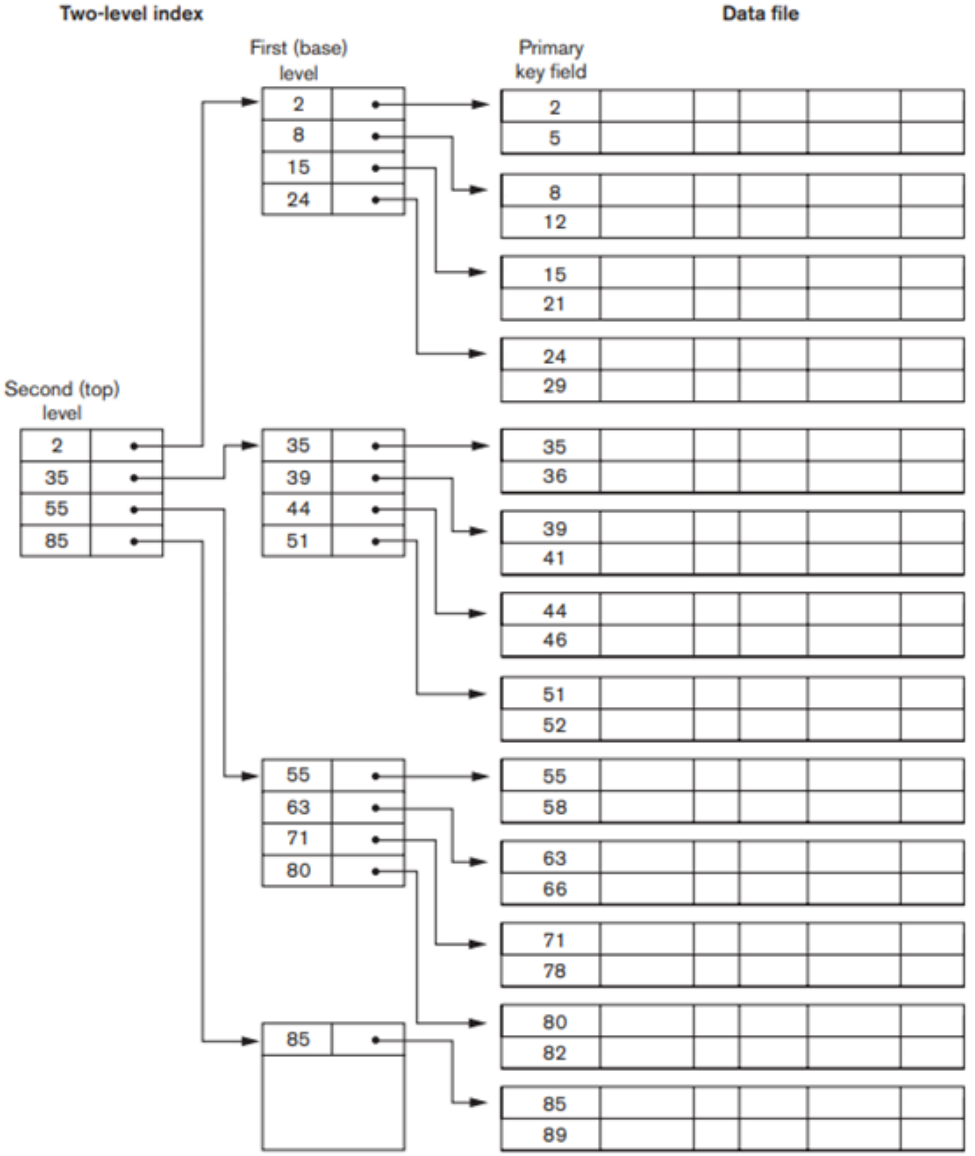
ISAM

file ⇒ ordered on key + multi level index on file

ISAM ≡ multilevel primary/clustering Index

≡ Index sequential file

Figure 17.6
A two-level primary index resembling ISAM (indexed sequential access method) organization.



Multilevel SI :

1st level : SI

Remaining level : Sparse

↳ k^{th} level is PI on $k-1$ level

Multilevel CI :

1st level : CI

Remaining level : Sparse

kth level is PI on k-1 level

Multilevel PI :

1st level : PI

Remaining level : Sparse

↳ k^{th} level is PI on $k-1$ level

ISAM file ≡ Indexed Sequential file

file Ordered + multilevel Primary Index

3.7.3 Indexing: GATE CSE 1993 | Question: 14 top

https://gateoverflow.in/2311

An ISAM (indexed sequential) file consists of records of size 64 bytes each, including key field of size 14 bytes. An address of a disk block takes 2 bytes. If the disk block size is 512 bytes and there are $16K$ records, compute the size of the data and index areas in terms of number blocks. How many levels of tree do you have for the index?

multilevel primary index

3.7.3 Indexing: GATE CSE 1993 | Question: 14 top

https://gateoverflow.in/2311

An ISAM (indexed sequential) file consists of records of size 64 bytes each, including key field of size 14 bytes. An address of a disk block takes 2 bytes. If the disk block size is 512 bytes and there are $16K$ records, compute the size of the data and index areas in terms of number blocks. How many levels of tree do you have for the index?

ISAM file

multilevel primary index

+ ISAM

3.7.3 Indexing: GATE CSE 1993 | Question: 14 top

https://gateoverflow.in/2311

An ISAM (indexed sequential) file consists of records of size 64 bytes each, including key field of size 14 bytes. An address of a disk block takes 2 bytes. If the disk block size is 512 bytes and there are $16K$ records, compute the size of the data and index areas in terms of number blocks. How many levels of tree do you have for the index?

3 levels

2000 Blocks

64 Blocks

file :

Record Size = 64 B

Key field = 14 B

Block address (BP) = 2 B

Block Size = 512 B

Records = 16000

file Blocks

$$= \left\lceil \frac{16000}{8} \right\rceil$$

= 2000 Blocks

→ Blocking factor of file = # Records / Block

$$\left\lfloor \frac{512}{64} \right\rfloor = 8$$

Index: multilevel primary index

Index entry size: (whatever level of index)

Told by ISAM file

14 + 2B = 16B

key field size

Index Blocking factor :

$$\left\lfloor \frac{512\ B}{16\ B} \right\rfloor = 32$$ Index entries Per Block

1st level Index :

#index entries = #Blocks in file

= 2000

$$\#IL_1\ Blocks = \left\lceil \frac{2000}{32} \right\rceil = 63$$ IL_1 Blocks

2^{nd} level index:

#IL2 entries = #Blocks in IL1

= 63

#IL2 Blocks = $\lceil \frac{63}{32} \rceil = 2$

3^{rd} level index: → 1 Block ; 2 entries

Size of Data file : 2000 Blocks

Size of Index file : 1 + 2 + 63 = 66 Blocks

ISAM file: multilevel primary Index

master index

3.7.2 Indexing: GATE CSE 1990 | Question: 10b top

https://gateoverflow.in/85691

One giga bytes of data are to be organized as an indexed-sequential file with a uniform blocking factor 8. Assuming a block size of 1 Kilo bytes and a block refrencing pointer size of 32 bits, find out the number of levels of indexing that would be required and the size of the index at each level. Determine also the size of the master index. The referencing capability (fanout ratio) per block of index storage may be considered to be 32.

File Blocking factor

File with multilevel Primary index

3.7.2 Indexing: GATE CSE 1990 | Question: 10b

https://gateoverflow.in/85691

One giga bytes of data are to be organized as an indexed-sequential file with a uniform blocking factor 8. Assuming a block size of 1 Kilo bytes and a block refrencing pointer size of 32 bits, find out the number of levels of indexing that would be required and the size of the index at each level. Determine also the size of the master index. The referencing capability (fanout ratio) per block of index storage may be considered to be 32.

#index entries = 32

Total Index size

Subjective

Total index size = $2^{15} + 2^{10} + 2^5 + 1$ Blocks

4

3.7.2 Indexing: GATE CSE 1990 | Question: 10b

https://gateoverflow.in/85691

One giga bytes of data are to be organized as an indexed-sequential file with a uniform blocking factor 8. Assuming a block size of 1 Kilo bytes and a block referencing pointer size of 32 bits, find out the number of levels of indexing that would be required and the size of the index at each level. Determine also the size of the master index. The referencing capability (fanout ratio) per block of index storage may be considered to be 32.

1st level: 2^{15} Blocks

2nd: 2^{10}

3rd: 2^5

4th: 1 Block

Data file : 1GB ⇒

Blocking factor : 8 Records / Block

Block size = 1kB

Block pointer = 4B

#Blocks = $\frac{1GB}{1kB} = 1m = 2^{20}$ Blocks

Record size = $\frac{1kB}{8} = 128B$

1st level index:

#Entries = 2^{20} = #file Blocks

Blocking factor of index = 32

#IL1 Blocks = $\lceil \frac{2^{20}}{2^5} \rceil = 2^{15}$ Blocks

2^{nd} level index:

$\#\text{Entries} = 2^{15} = \#\text{IL}_1\text{ Blocks}$

Blocking factor of index = 32

$$\#\text{IL}_1\text{ Blocks} = \left\lceil \frac{2^{15}}{2^5} \right\rceil = 2^{10}\text{ Blocks}$$

3rd level index:

$\#\text{Entries} = 2^{10} = \#\text{IL2 Blocks}$

Blocking factor of index = 32

$\#\text{IL}_1\text{ Blocks} = \left\lceil \frac{2^{10}}{2^5} \right\rceil = 2^5$ Blocks

4th level index:

$\#\text{Entries} = 2^5 = \#\text{IL2 Blocks}$

Blocking factor of index = 32

$\#\text{IL1 Blocks} = \left\lceil \frac{2^5}{2^5} \right\rceil = 1 \text{ Blocks}$

$\#\text{Index levels} = 4$ ✓

3.7.6 Indexing: GATE CSE 2008 | Question: 70 top

https://gateoverflow.in/259

Consider a file of 16384 records. Each record is 32 *bytes* long and its key field is of size 6 *bytes*. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 *bytes*, and the size of a block pointer is 10 *bytes*. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively

A. 8 and 0
B. 128 and 6
C. 256 and 4
D. 512 and 5

gate2008-cse databases indexing normal

3.7.6 Indexing: GATE CSE 2008 | Question: 70 top

https://gateoverflow.in/259

Consider a file of 16384 records. Each record is 32 *bytes* long and its key field is of size 6 *bytes*. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 *bytes*, and the size of a block pointer is 10 *bytes*. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively

A. 8 and 0
B. 128 and 6
C. 256 and 4
D. 512 and 5

file is ordered by Non-key field ✓

Index on key field : SI on key

search key of Index

gate2008-cse | databases | indexing | normal

File :

16384 Records.

Record Size = 32 B

key field = 6B

Block Size = 1024 B

" Pointer = 10 B

SI on key :

Index entry size : (Every level) $6B + 10B = 16B$

Index Blocking factor (Every level) $= \left\lfloor \frac{1024B}{16B} \right\rfloor = 64$

1st level index:

$\#$ IL1 entries $= 16384$

$\#$ Blocks (IL1) $= \left\lceil \frac{16384}{64} \right\rceil = 256$ Blocks in level 1

2nd level index:

#IL2 entries = 256

#Blocks (IL2) = $\left\lceil \frac{256}{64} \right\rceil$ = 4 Blocks in level 2

3 level index:

IL3 entries = 4

Blocks in IL3 = $\left\lceil \frac{4}{64} \right\rceil$ = 1 Block in level 3

Next Topic:

Indexing

Recap (Self-Read)

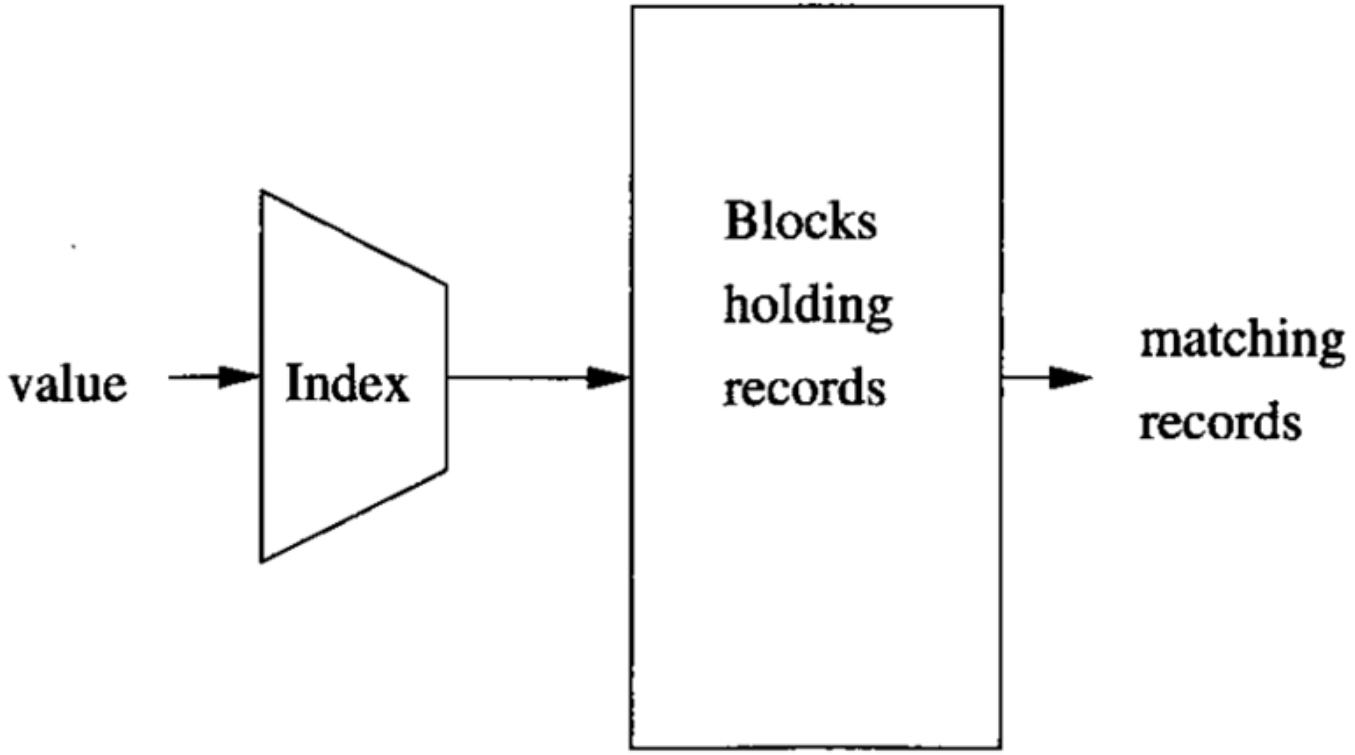


Figure 14.1: An index takes a value for some field(s) and finds records with the matching value

Many queries reference only a small proportion of the records in a file. For example, a query like "Find all instructors in the Physics department" or "Find the total number of credits earned by the student with *ID* 22201" references only a fraction of the student records. It is inefficient for the system to read every tuple in the *instructor* relation to check if the *dept_name* value is "Physics". Likewise, it is inefficient to read the entire *student* relation just to find the one tuple for the *ID* "32556,". Ideally, the system should be able to locate these records directly. To allow these forms of access, we design additional structures that we associate with files.

In this chapter, we assume that a file already exists with some primary organization such as the unordered, ordered organizations that were described in last chapter.

We will describe additional auxiliary access structures called indexes, which are used to speed up the retrieval of records in response to certain search conditions.
The index structures are additional files on disk.

Indexes enable efficient access to records based on the indexing fields that are used to construct the index.

Basically, any field of the file can be used to create an index, and multiple indexes on different fields can be constructed on the same file.

To find a record or records in the data file based on a search condition on an indexing field, the index is searched, which leads to pointers to one or more disk blocks in the data file where the required records are located.

Different Kinds of "Keys"

There are many meanings of the term "key." We used it in Section 2.3.6 to mean the primary key of a relation. We shall also speak of "sort keys," the attribute(s) on which a file of records is sorted. We just introduced "search keys," the attribute(s) for which we are given values and asked to search, through an index, for tuples with matching values. We try to use the appropriate adjective — "primary," "sort," or "search" — when the meaning of "key" is unclear.

We often want to have more than one index for a file. For example, we may wish to search for a book by author, by subject, or by title.

An attribute or set of attributes used to look up records in a file is called a **search key**. Note that this definition of *key* differs from that used in *primary key*, *candidate key*, and *superkey*. This duplicate meaning for *key* is (unfortunately) well

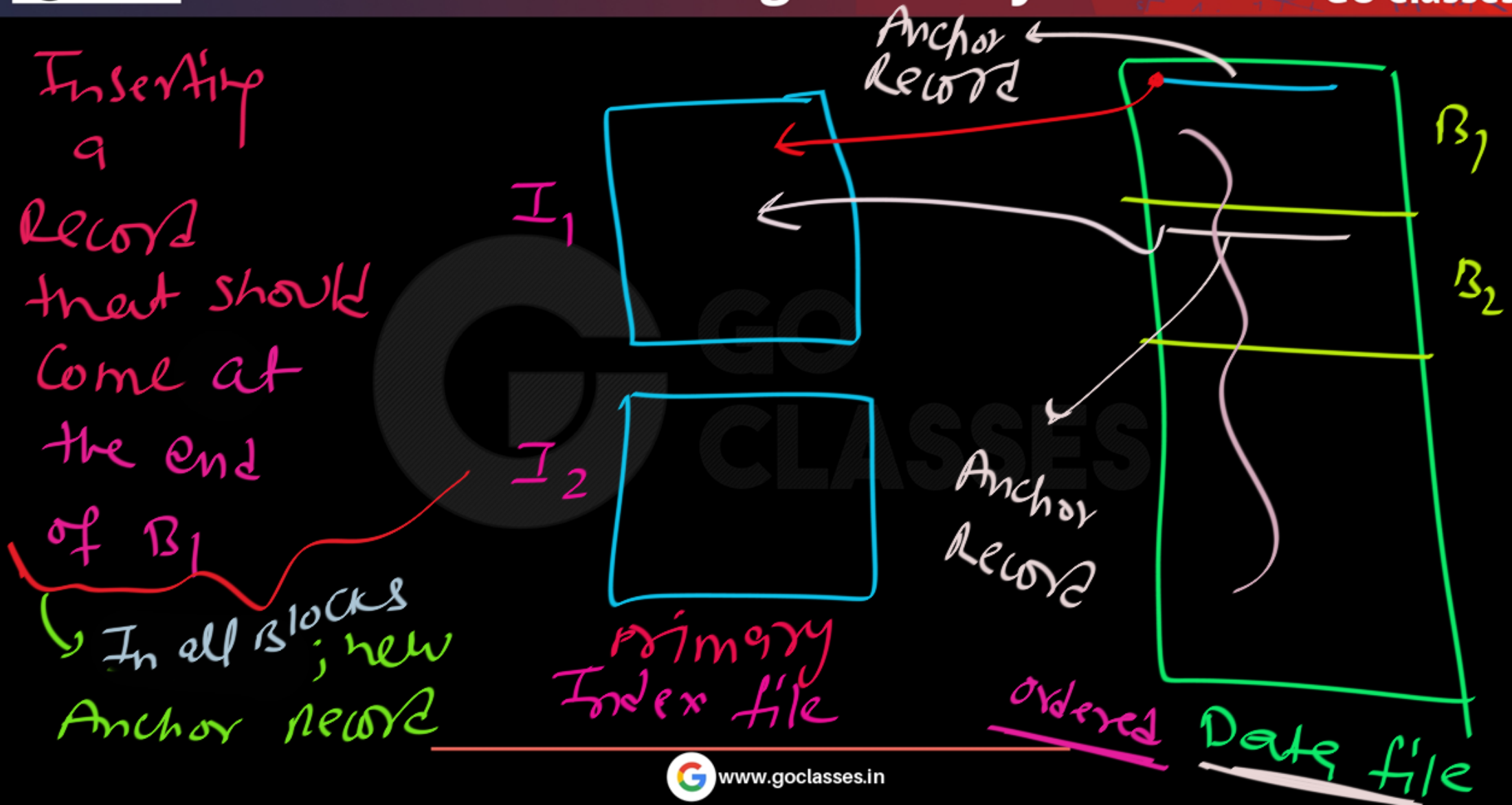
For a file with a given record structure consisting of several fields (or attributes), an index access structure is usually defined on a single field of a file, called an **indexing field** (or **indexing attribute**).[1] The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value. The values in the index are *ordered* so that we can do a *binary search* on the index. If both the data file and the index file are ordered, and since the index file is typically much smaller than the data file, searching the index using a binary search is a better option. Tree-structured multilevel indexes (see Section 17.2) implement

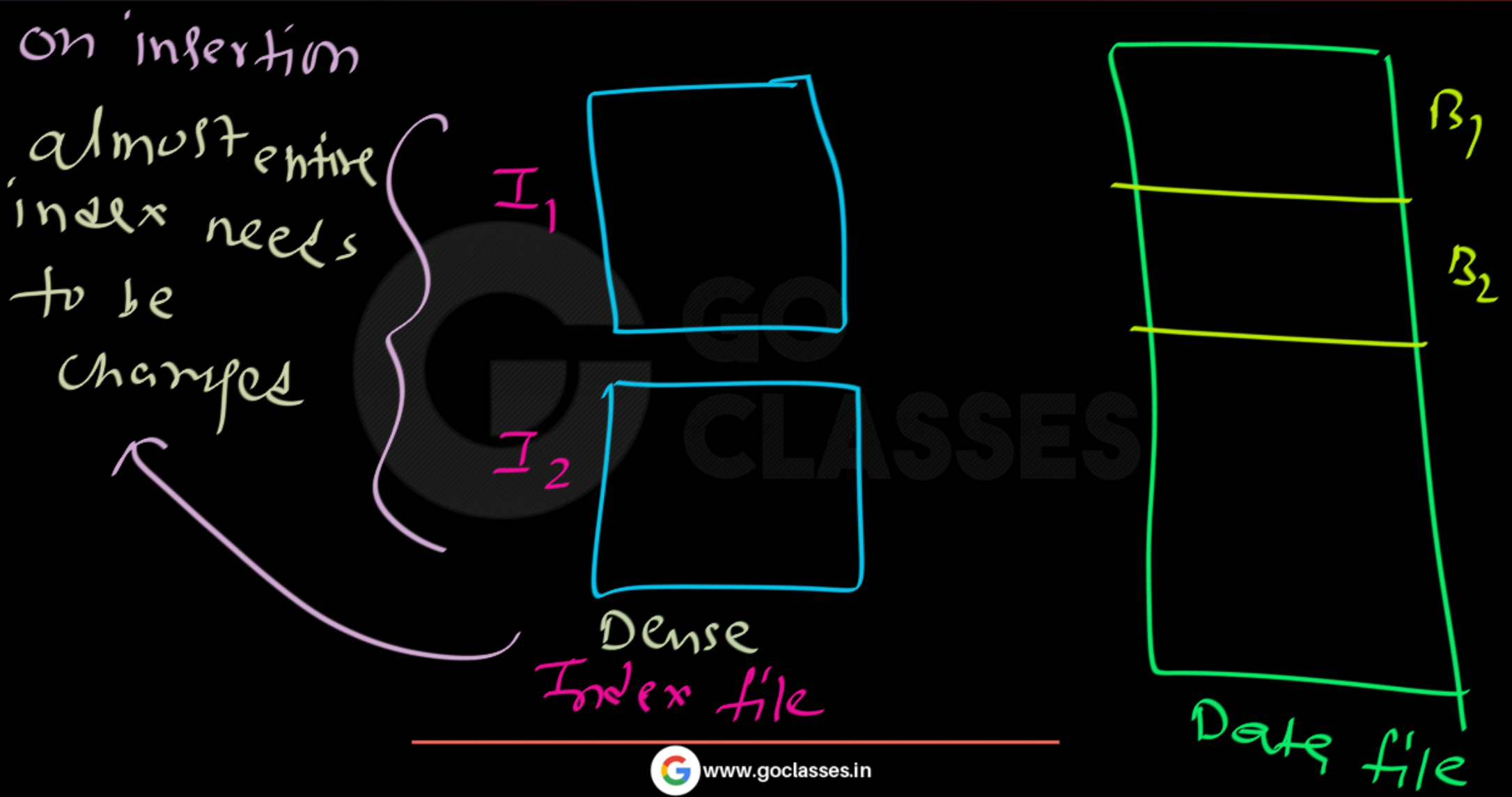
We describe different types of single-level ordered indexes—primary, secondary, and clustering—in Section 17.1. By viewing a single-level index as an ordered file, one can develop additional indexes for it, giving rise to the concept of multilevel indexes. A popular indexing scheme called **ISAM (indexed sequential access method)** is based on this idea. We discuss multilevel tree-structured indexes in Sec-

There are several types of ordered indexes. A **primary index** is specified on the *ordering key field* of an **ordered file** of records. Recall from Section 16.7 that an ordering key field is used to *physically order* the file records on disk, and every record has a *unique value* for that field. If the ordering field is not a key field—that is, if numerous records in the file can have the same value for the ordering field—another type of index, called a **clustering index**, can be used. The data file is called a **clustered file** in this latter case. Notice that a file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, *but not both*. A third type of index, called a **secondary index**, can be specified on any *nonordering* field of a file. A data file can have several secondary indexes in addition to its primary access method. We discuss these types of single-level indexes in the next three subsections.

Ordered Indexes :

- Problem (DisAdvantage)
  - modification is Very time Consuming (Very Hard)
    - insertion
    - Deletion
    - modify


Inserting a Record that should come at the end of B1
In all Blocks ; new Anchor record
Anchor Record
I1
I2
Primary Index file
Anchor Record
B1
B2
ordered
Data file


On insertion
almost entire
index needs
to be
changes
I_1
I_2
Dense
Index file
B_1
B_2
Data file

If Multilevel index ; modification becomes even more Harder.

When to prefer these indexes that we have seen so far ? ⇒ when modifications are Rare.

Static indexes

Data is mostly Static.

for Dynamically changing Data file,
we need some "modification-friendly"
index

B, B+ Tree Indexing

Next...