Mod 5 - Lecture 4,5:

Indexing

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

Content of this Lecture:

1. Indexing Overall Idea
2. Single Level Ordered Indices
3. Multilevel Indices

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.

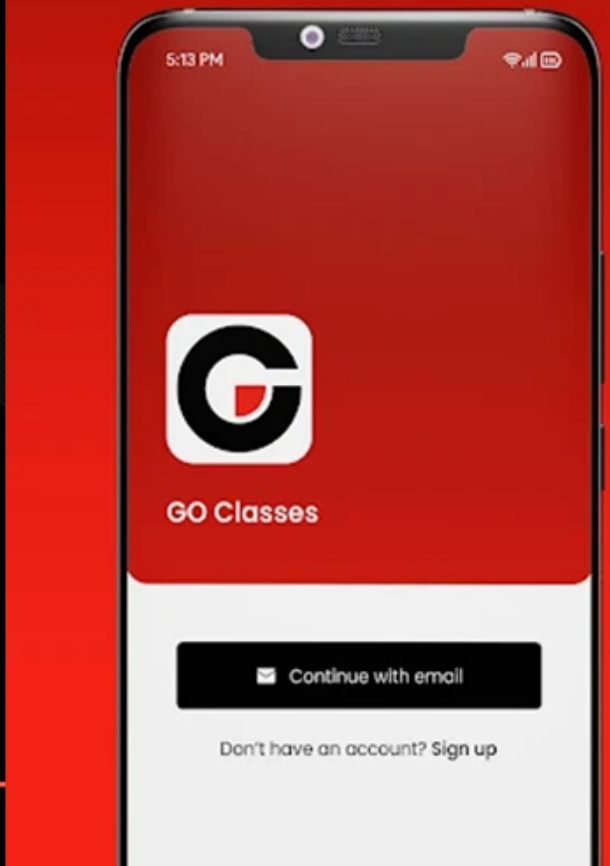
Visit here to watch : https://www.goclasses.in/s/store/

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

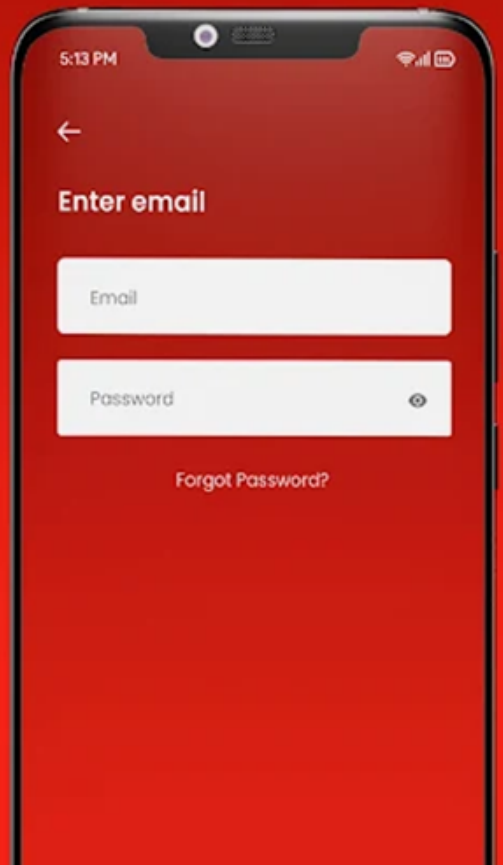
Download the GO Classes Android App:

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

Search "GO Classes" on Play Store.

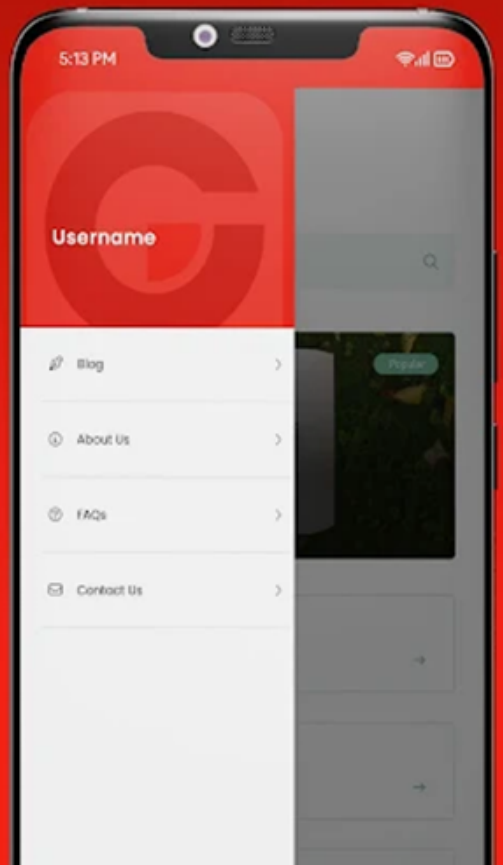


Welcome to
GO Classes
Learn at your own pace



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 :

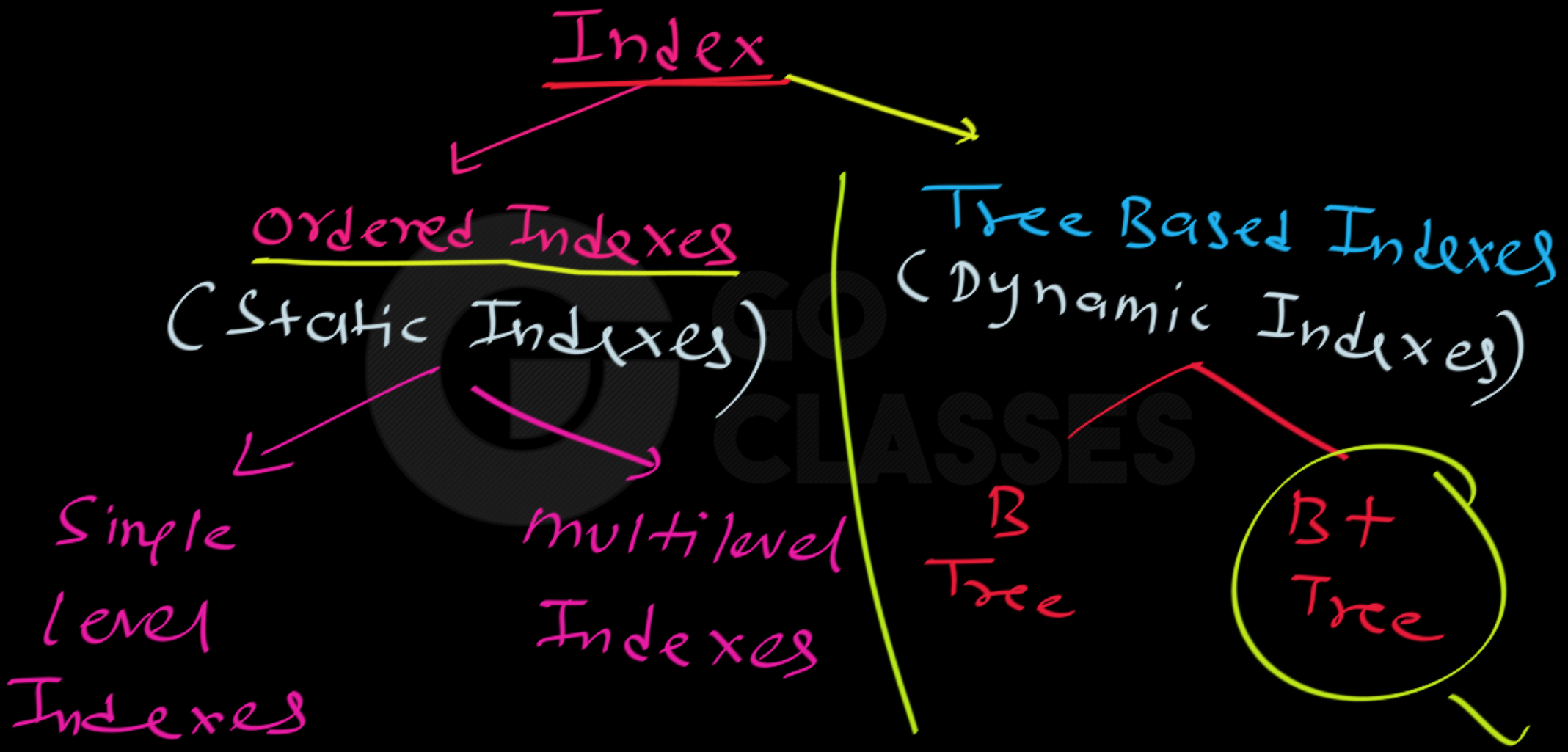
Username: gateoverflow_cse

Indexing is a topic which many students think they understand BUT they lack the CRYSTAL Clarity.

We'll study in a CRYSTAL Clear Way, like ALWAYS.

Chapter 2:

Index Structures


Index
Ordered Indexes
(Static Indexes)
Single level Indexes
Multilevel Indexes
Tree Based Indexes
(Dynamic Indexes)
B Tree
B+ Tree

7. Indexing

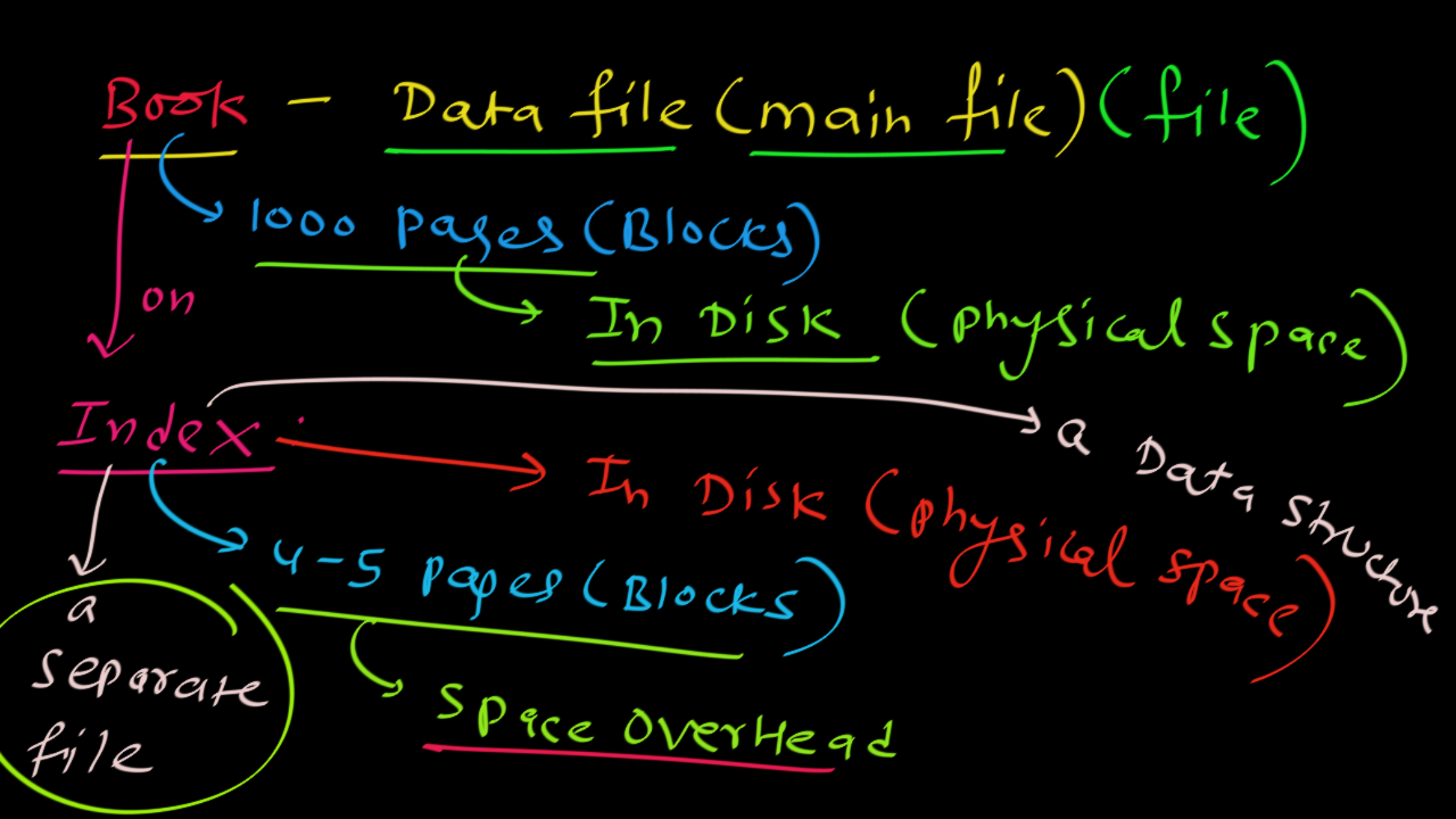
Contents:

- Single-Level Ordered Indexes
- Multi-Level Indexes
- B^+ Tree based Indexes

Next Topic:

Index Structures

Overall Idea


Book - Data file (main file) (file)
1000 pages (Blocks)
on
In Disk (physical space)
Index :
a Data structure
In Disk (physical space)
4-5 pages (Blocks)
a separate file
Space Overhead

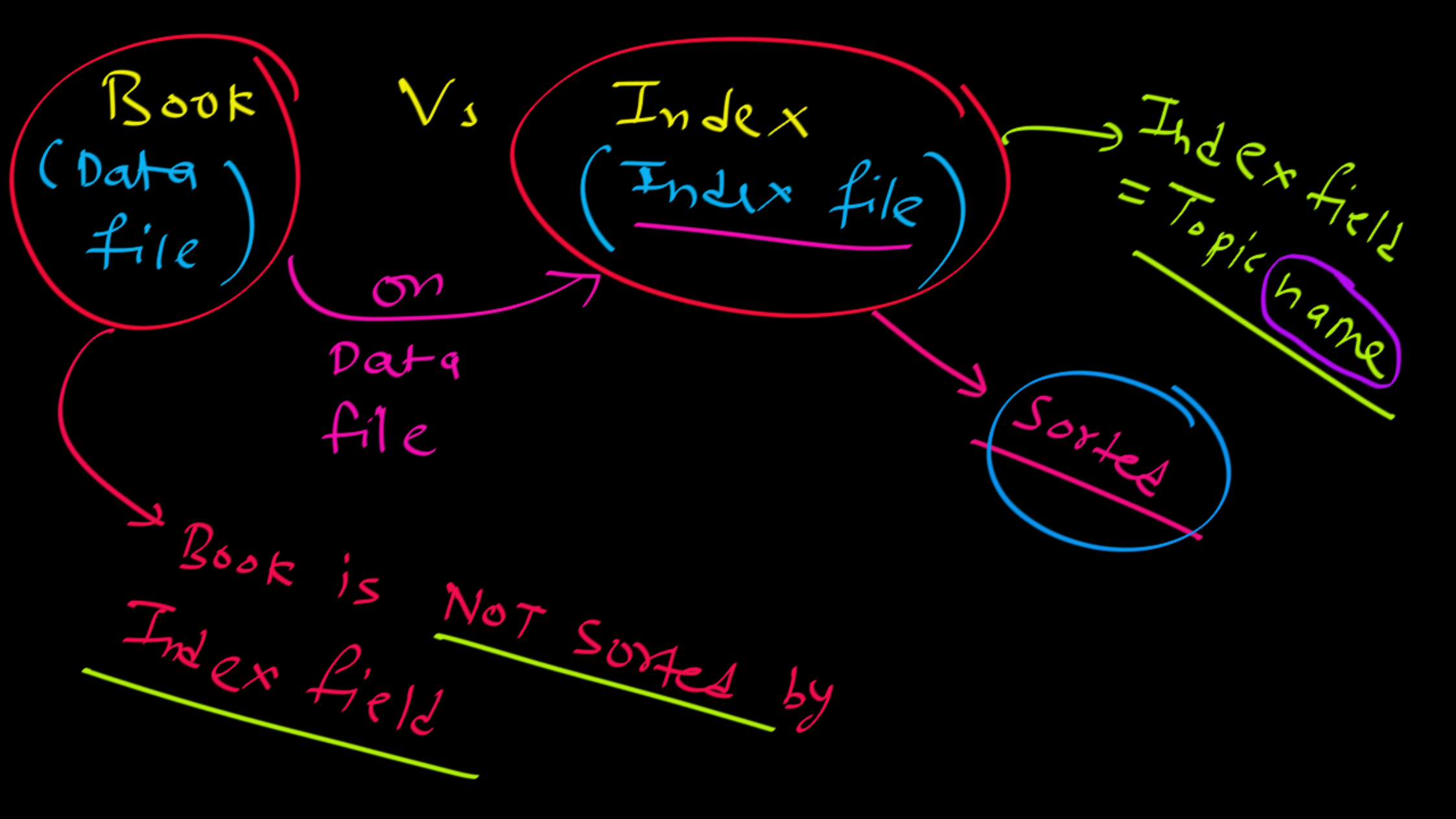
You already understand Indexing...

For ANALOGY,

Let's see the Index at the end of the RaghuRama Krishnan Book of DBMS.

SUBJECT INDEX

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621


Book
(Data file)
Vs
Index
(Index file)
Index field = Topic name
on Data file
Sorted
Book is NOT sorted by Index field

You already understand Indexing...

For ANALOGY,

Let's see the Index at the end of the Korth Book of DBMS.

Index

2PC. See two-phase commit
3NF. See third normal form
3PC. See three-phase commit

abstract data types, 1127
access paths, 542
ACID properties. See atomicity; consistency; durability; isolation
Active Server Pages (ASP), 397
query optimization and, 597
query processing and, 566-567
ranking and, 192-195
relational algebra and, 235-239
representation of, 304
view maintenance and, 610-611
windowing, 195-197
Ajax, 390-391, 398, 867
disconnected operation and, 395-396
encryption and, 411-417
HyperText Markup Language (HTML) and, 378-380
HyperText Transfer Protocol (HTTP) and, 377-381, 383, 395, 404-406, 417
Java Server Pages (JSP) and, 377, 383-391

Index (Index file):

a particular entry (Index entry)

< index field value v, Block/record address of the record containing v >

Index Entry:

<search key value L, address of the data record containing search key value L>

Index Entry:

<search key, pointer>

Do you understand??

The Difference between a Book & Index on that book ??

If yes, Don't forget this difference (because most students forget it)

Index file Vs Data File

What most students don't understand:

Difference between a File & Index on that file.

Data File R Vs an Index on file R

Index file or Index

Vs

Data file or Main file or Indexed file or file

Consider the Students relation:

Data file

Student(sid, name, marks, state)

How many Indexes can we create on this relation??

at the same time

Consider the Students relation:

Data file

Student(sid, name, marks, state)

File ≡ Relation (DBMS point of view)

maximum

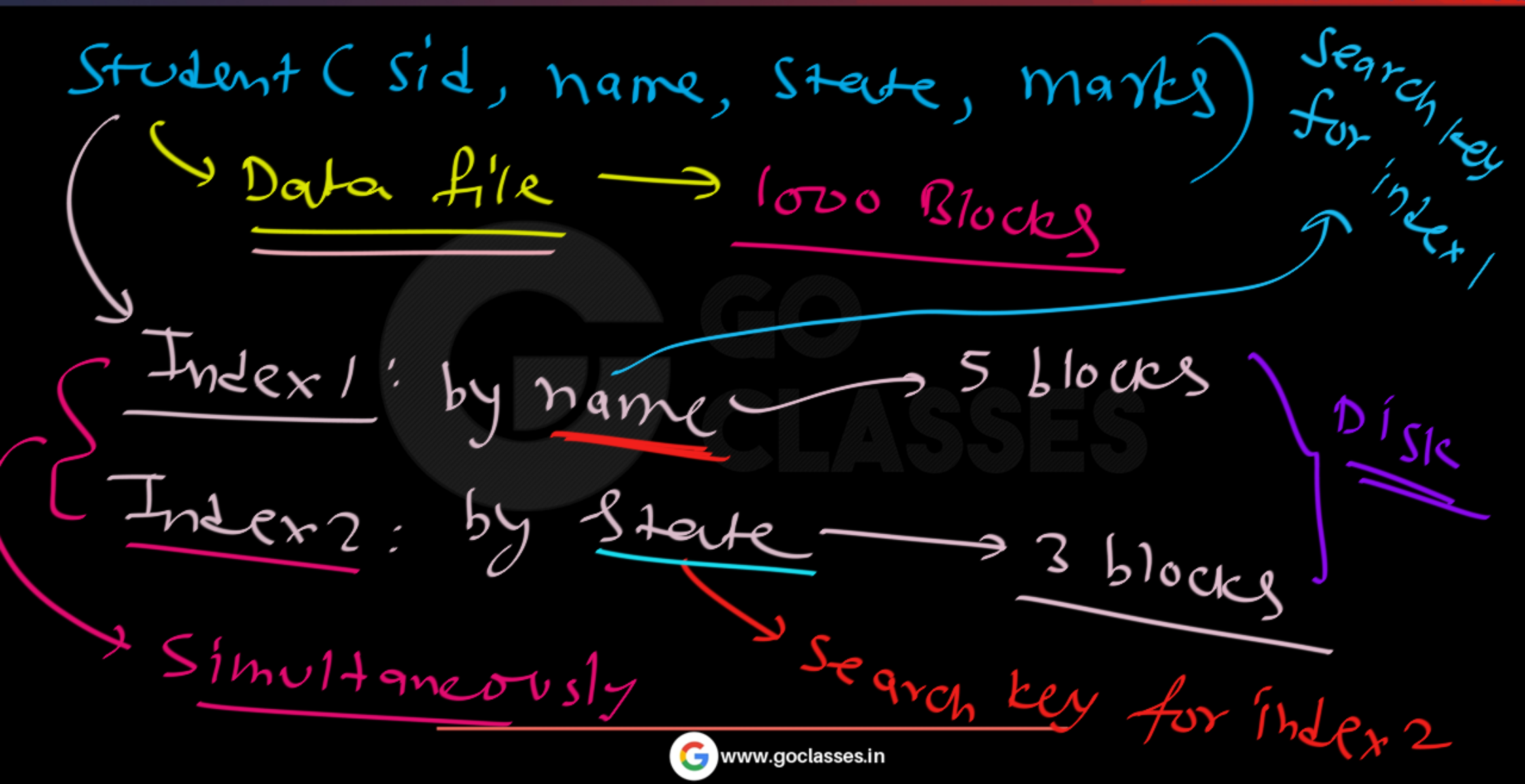
How many Indexes can we create on this relation?? = 4

Simultaneously

Navathe Book:

at the end

1. Index by topic name → 10 Pages
2. Index by Author name of topics → 3 Pages


Student (sid, name, state, marks)
Data file
1000 Blocks
Search key for index 1
Index 1 : by name
5 blocks
Index 2 : by state
3 blocks
Disk
Simultaneously
Search key for index 2

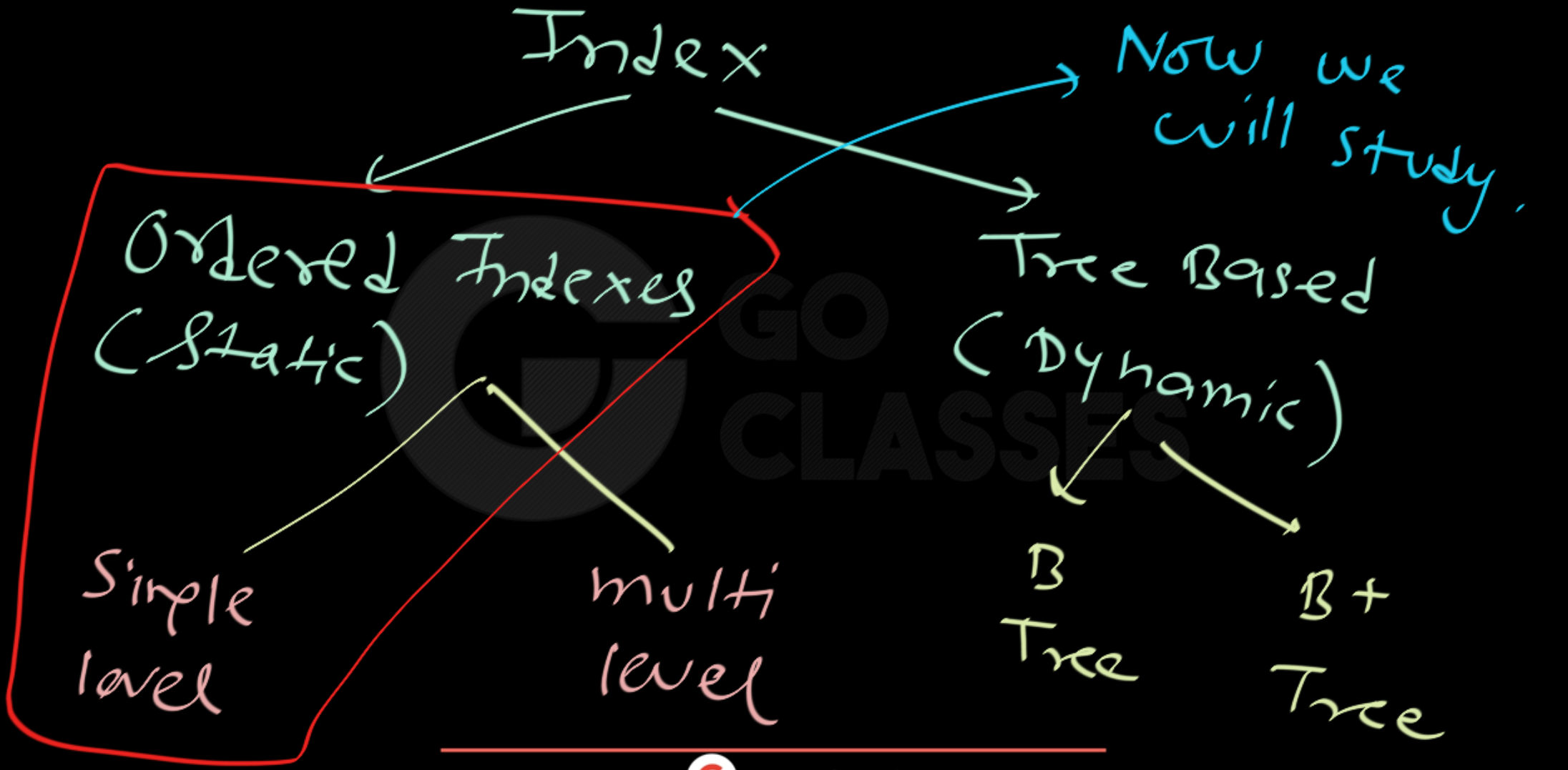
Search key ≡ Index field

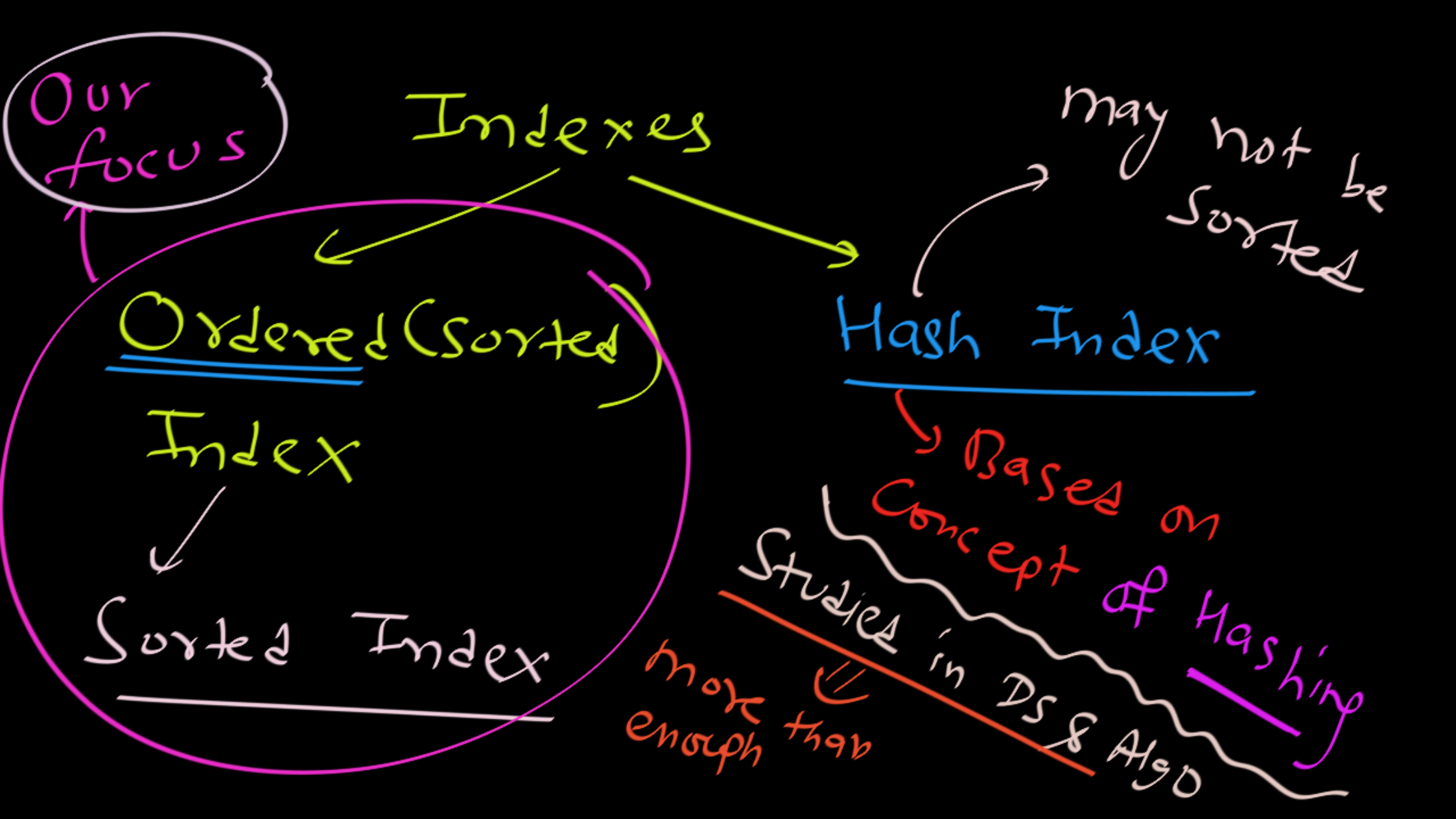
Index:

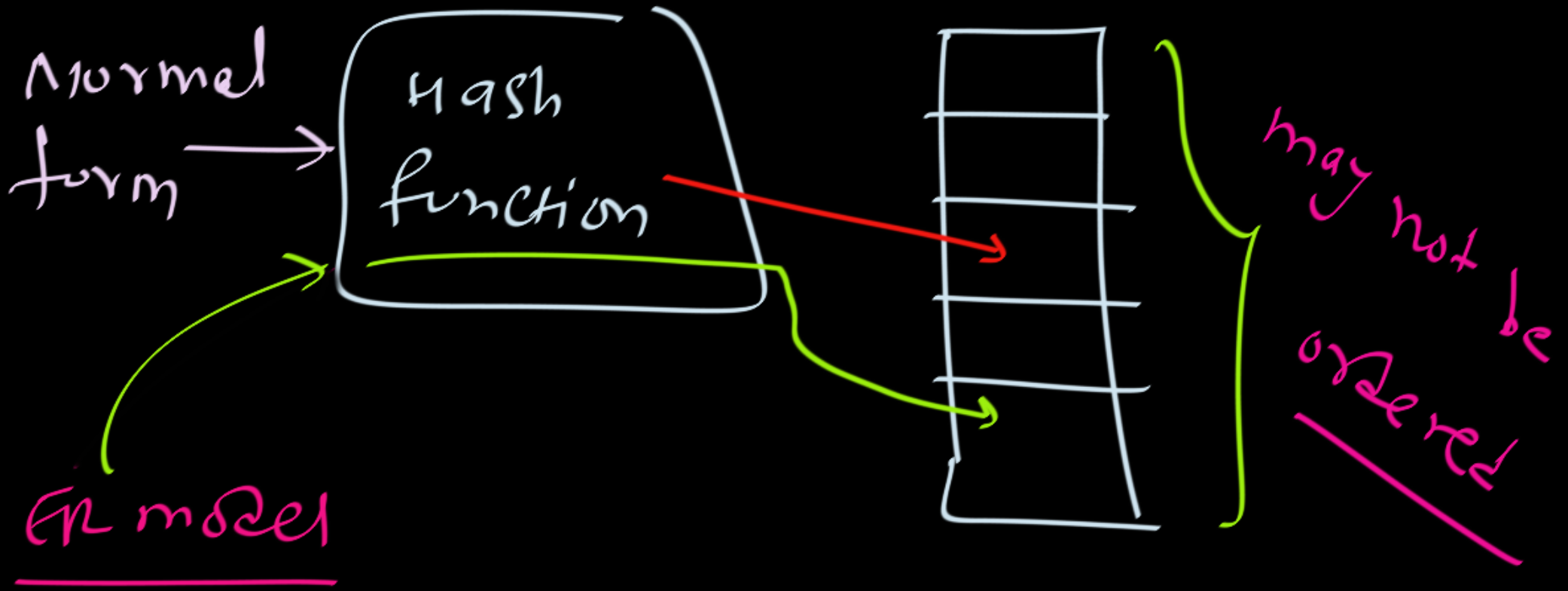
To gain fast random access to records in a file, we can use an index structure.

Each index structure is associated with a particular search key.

A file may have several indices, on different search keys.


Index
Now we will study.
Ordered Indexes (Static)
Tree Based (Dynamic)
Simple level
multi level
B Tree
B+ Tree


Our focus
Indexes
May not be sorted
Ordered (Sorted) Index
Sorted Index
Hash Index
Based on Concept of Hashing
Studies in DS & Algo
More than enough


Normal form
Hash function
ER model
may not be ordered

Next Topic:

Index Structures

1. Single Level Ordered Indices

Ordered Indexes

↔ Sorted

What is "Ordered" here??

File or Index ???

→ may or may not be Ordered

Ordered Indexes

In an ordered index file, index entries are stored sorted by the search key value.

Ordered Indexes

By default:

Index file is Ordered.

Ordered Indexes

In an ordered index file, index entries are stored sorted by the search key value.

NOTE: The records in Data file may or may not be ordered...

Q:

Index at the end of RaghuRama Krishnana book (Or any other standard book), is Ordered Index ???? Yes.

Is the Data file (Book) ordered in the same order as the Index file?? ⇒ No

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create Index on a heap file ??

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Unordered file

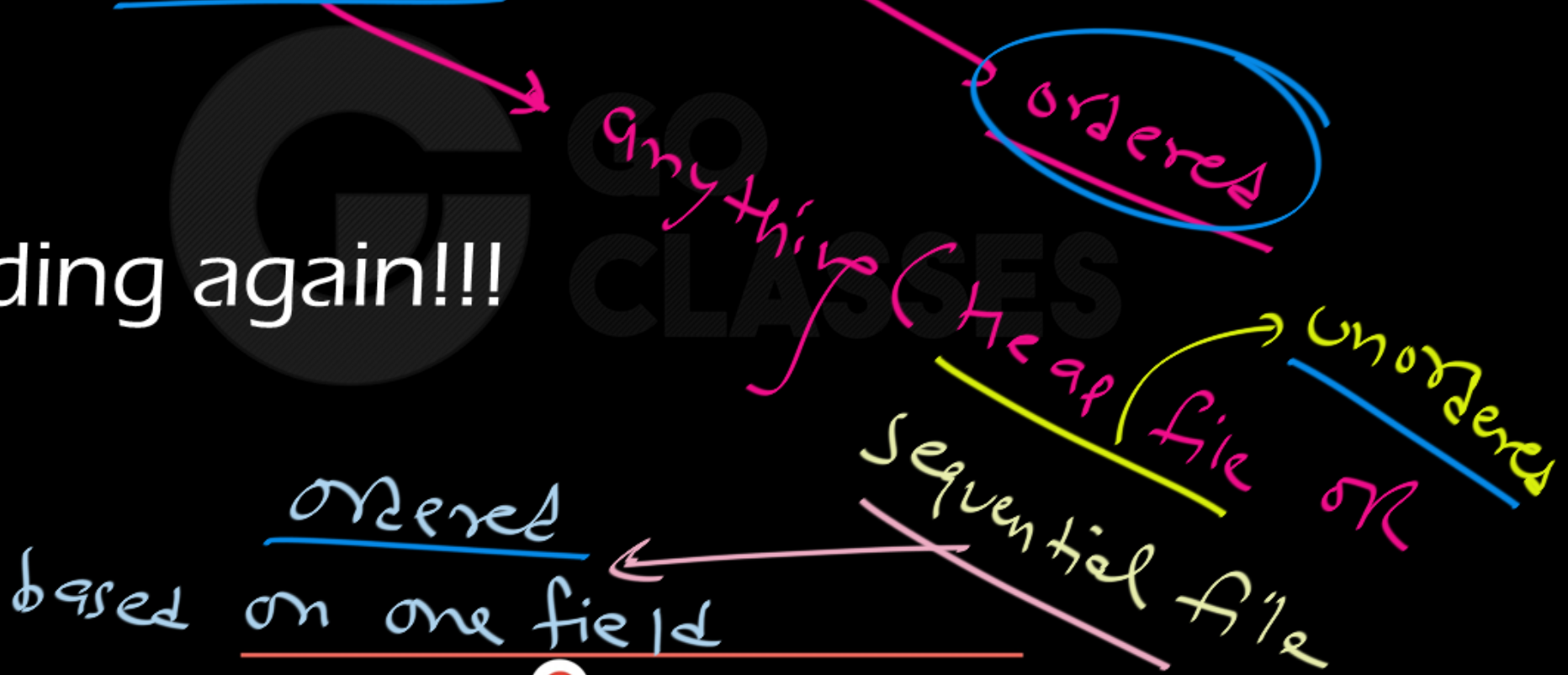
No field sorted

Can we create Index on a heap file ??

Yes.

I told you to Understand the difference between Data File and Index File..

Reminding again!!!



Ordered Indices:

An Ordered index stores the values of the search keys in sorted order, and associates with each search key the records that contain it.

The records in the indexed file(main file) may themselves be stored unsorted OR stored in some sorted order.

A file may have several indices, on different search keys.

Single Level Ordered Indexes

- Primary Index
- Clustering Index
- Secondary Index

Types of Single level Ordered Indexes:

1. Primary Index

Relation R: ⟶ Physically Stored on Disk.

R is Stored Sequentially (Sorted on some field)

OR

R is a Heap file (R Unordered)

No field ordered

Relation R : ⟶ Physically Stored on Disk in Sequential Manner based on ordering field A

If A = key in R

Index on A

primary Index

file is Physically Stored sorted by A.

Primary Index :

If Data file is Sorted on a key attribute A then Index on A is called Primary Index.

Primary Index:

A Index on field F

& CK of Data file

Data file sorted by F

Primary Index:

If the file containing the records is sequentially ordered by a key field F, then the Index using search key F is called Primary Index.

Simple attribute

Candidate key

Primary Index:

If the file containing the records is sequentially ordered by a key field F(i.e. F is the Ordering Key filed for data file), then the index whose search key is F is called Primary Index.

Primary Index:

A primary index is specified on the ordering key field of an ordered file of records.

Recall 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.

Q:

From DBMS point of view,
Index at the end of RaghuRama Krishnana book (Or any other standard book), is Primary Index ????

SUBJECT INDEX

Ordered Index

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621

Q:

From DBMS point of view,
Index at the end of RaghuRama Krishnana book (Or any other standard book), is Primary Index ???? ⇒ No.

Index: → Search key = Topic Name

Data file is NOT sorted on Topic Name.

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create Primary Index on a heap file ??

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Unordered file

Can we create Primary Index on a heap file ?? ⇒ No

NOTE:

So, to create Primary Index on a file R:

R MUST be PHYSICALLY Ordered by a KEY Field F...

Then a Index by F is called Primary Index.

Q:

Can we create Primary Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Primary Index??

Database Management System

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Database Management System

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543			80000
76766			72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

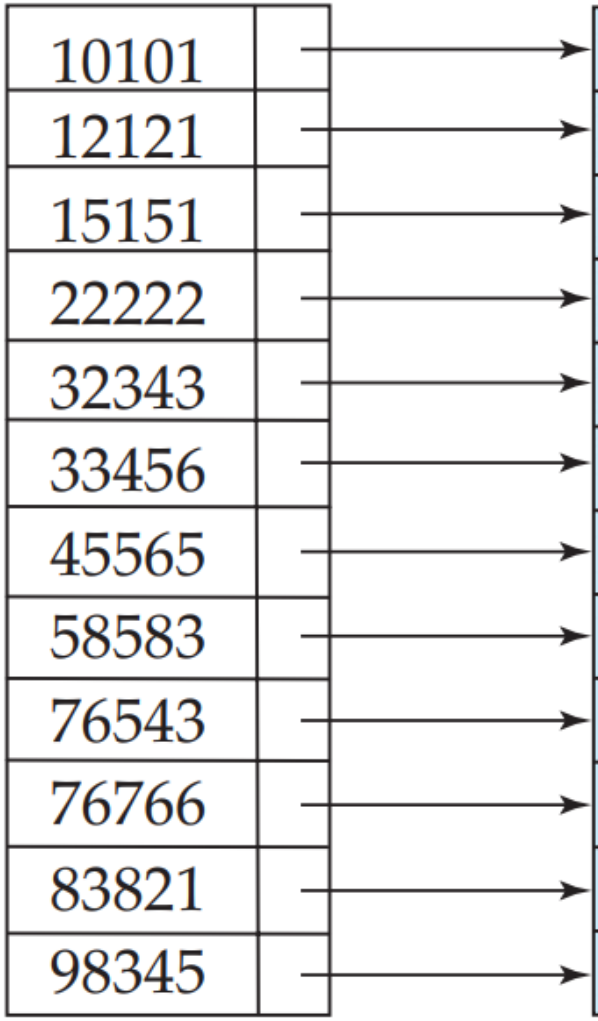
Id

key

So; Index on ID

primary Index

Physically sorted by Id.



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Q:

Can we create Primary Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Primary Index??

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Q: → No

Can we create Primary Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Primary Index?? ⇒ Can't create Primary Index

Q:

Index on Primary key is called Primary Index ??

Q:

Index on Primary key is called Primary Index ??

→ No (Data file may not be ordered by primary key)

Reason:

May be Data file is Heap file.

Data file : R(A, B, C)

Order (Sort) by A ⇒ ordering field

If A is key & file orders by A :-

ordering key field

Data file : R(A, B, C)

file
Order (Sort) by A ⟹ ordering field

If A is key ⟹ key field

Data file R(A, B, C)

How many ordering field can be there (at one point of time) ?

Order by A ⇒ Unordered by B, C

Order by B ⇒ Unordered by A, C

At most 1

At most 1 field we can keep ordered

Data file R(A, B, C)

file R ordered by C : ordering field of R

Assume B is key & file sorted by B :

ordering key field of R

Q:

How many Primary Indexes can be created on a file ?

Q:

How many Primary Indexes can be created on a file ?

⇒ At most one

Once one attribute becomes Ordering field then all remaining fields become Non-keys.

Order of Data file

is Same as

Order of primary Index

True

Primary Index:

In a sequentially ordered file, the index whose search key specifies the sequential order of the file, is called Primary Index.

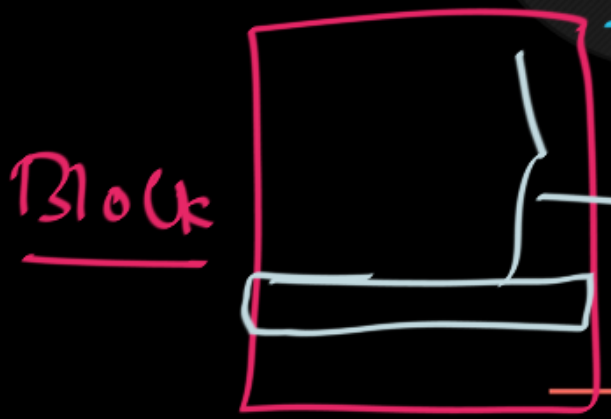
For a relation, there can be at most one primary index.

Do you have CRYSTAL Clarity about Primary Index??

Record address Vs Block address

Block address + offset

Block address → address of Block



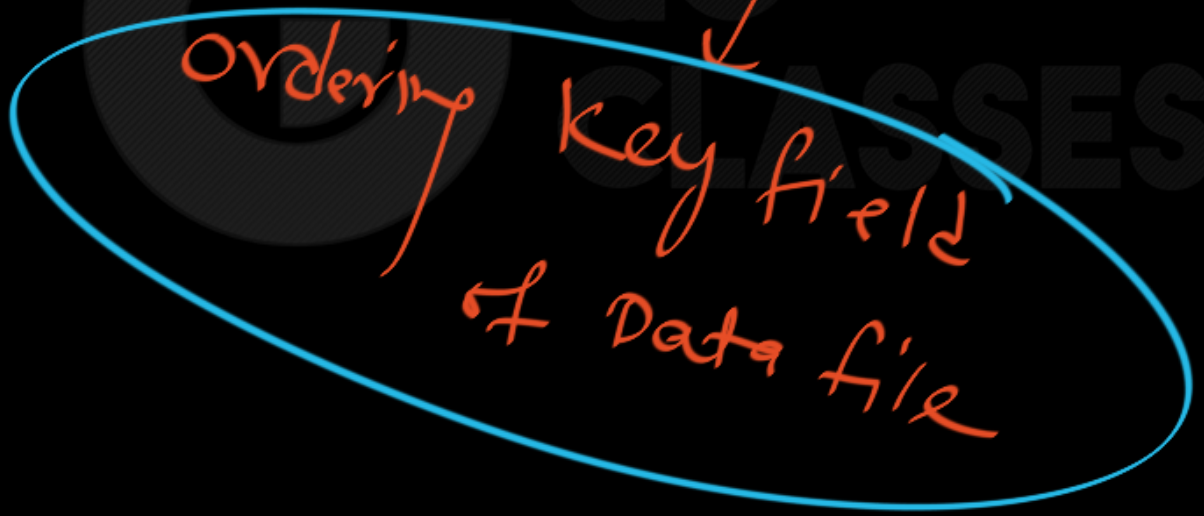
Primary Index:

Implementation

(Sparse Index)

Primary Index :

Index Entry : < Search key, pointer >

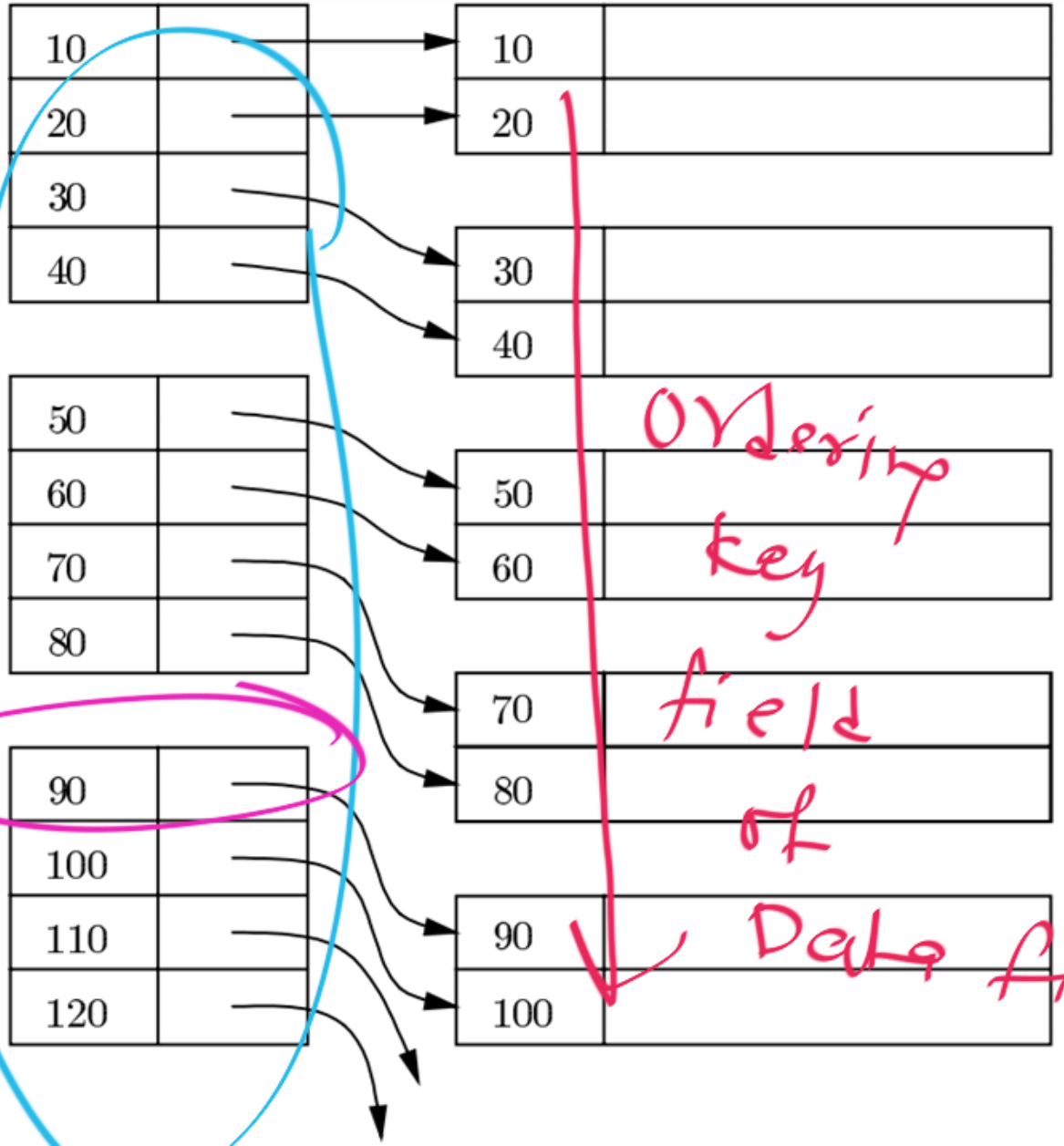


Primary Index

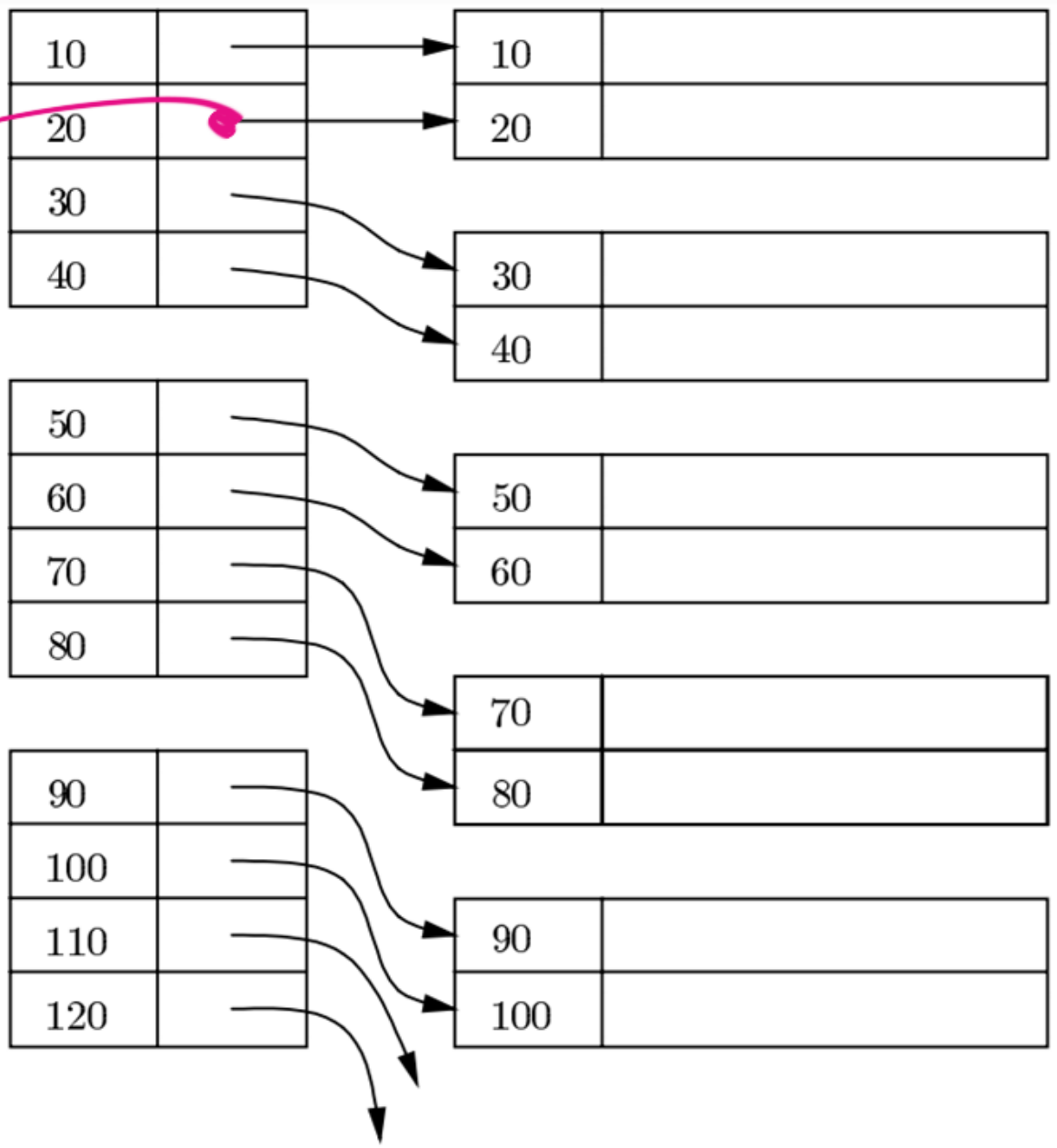
Index entry (Index Record)

Dense implementation:

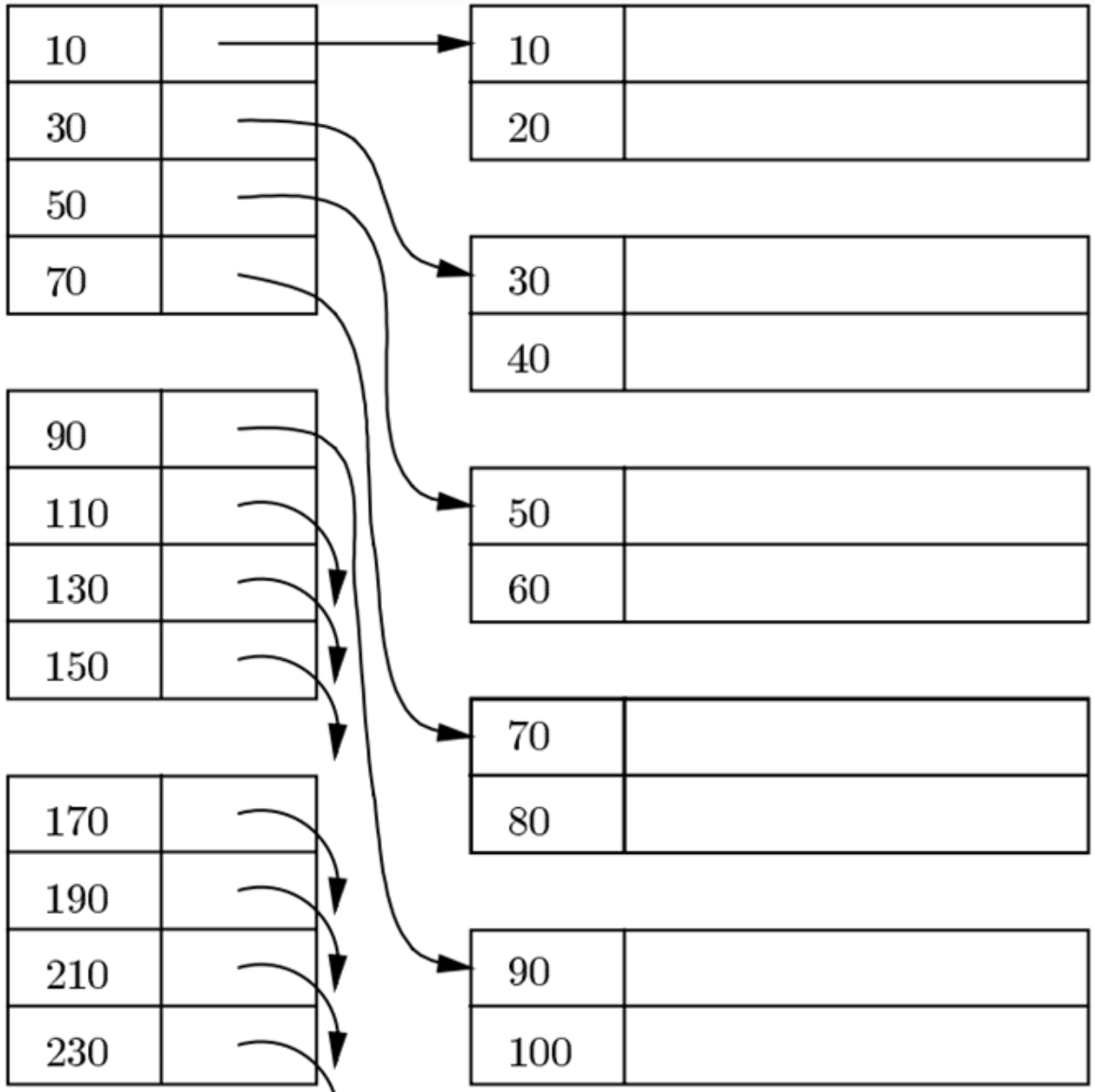
Ordering key field of Data file



Record address OR Block address

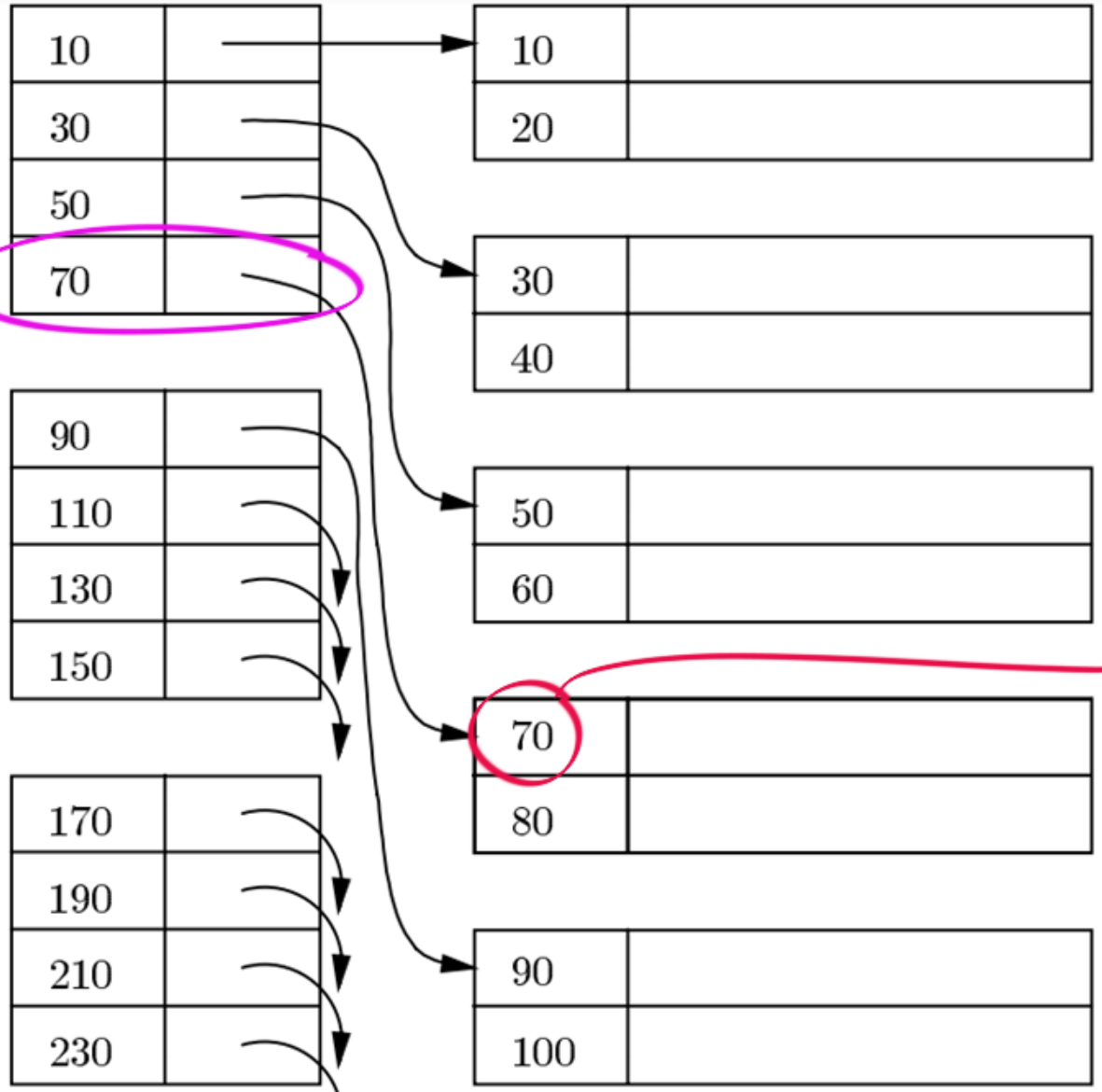


Dense implementation: for every Data record, one Index record



Sparse Implementation:

Database System Implementation, H. Garcia-Molina, J. Ullman, and J. Widom, Prentice-Hall, 2000.



Index entry

Sparse Implementation:

Block Anchor

Database System Implementation, H. Garcia-Molina, J. Ullman, and J. Widom, Prentice-Hall, 2000.

Primary Index (Simple level)

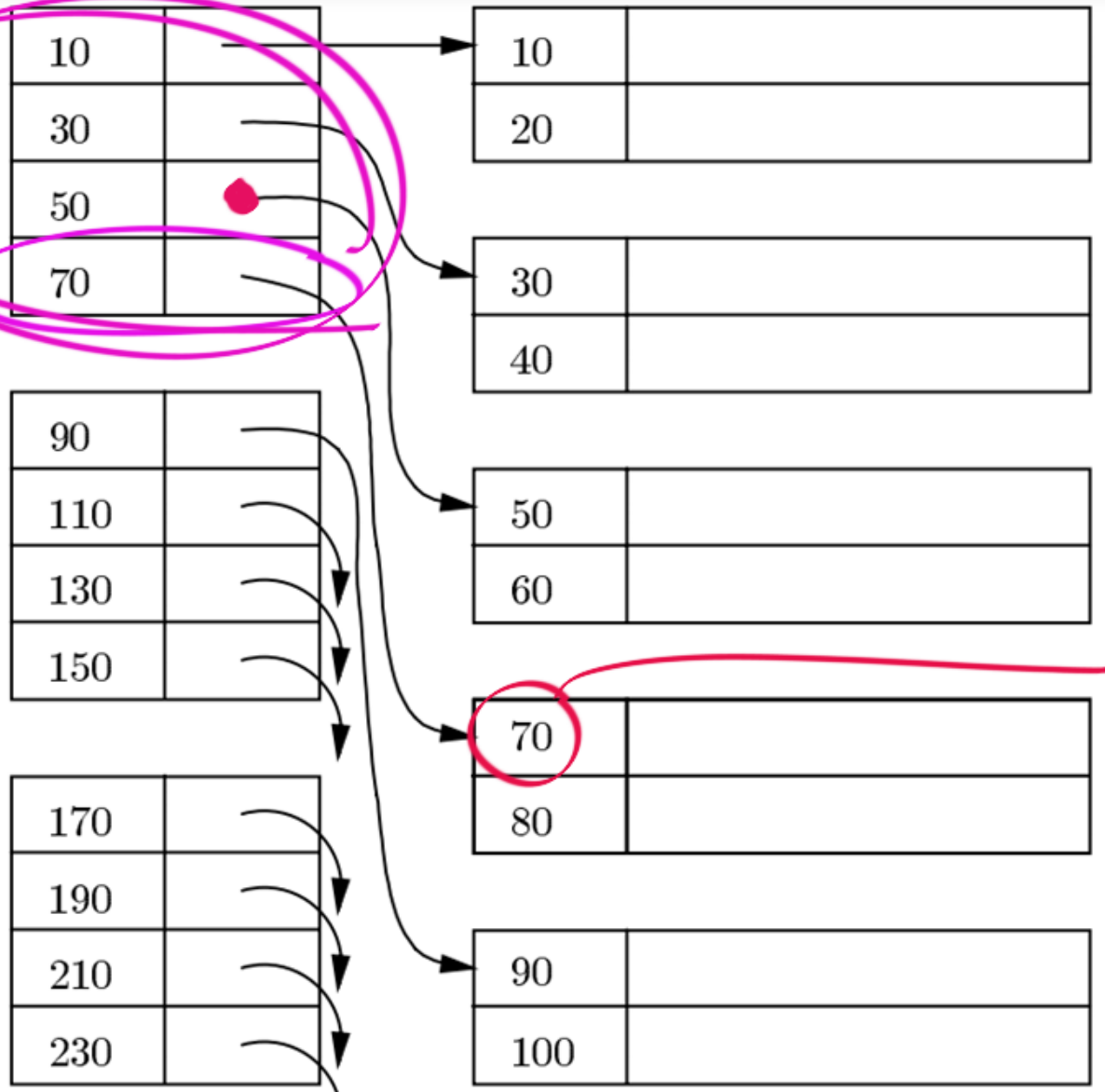
Search Algo :

(I) index Blocks

(1) Binary search on Index Blocks

(2) 1 more Access for Data Block.

$$\#\text{Block Access} = \lceil \log_2 I \rceil + 1$$



Sparse Implementation:

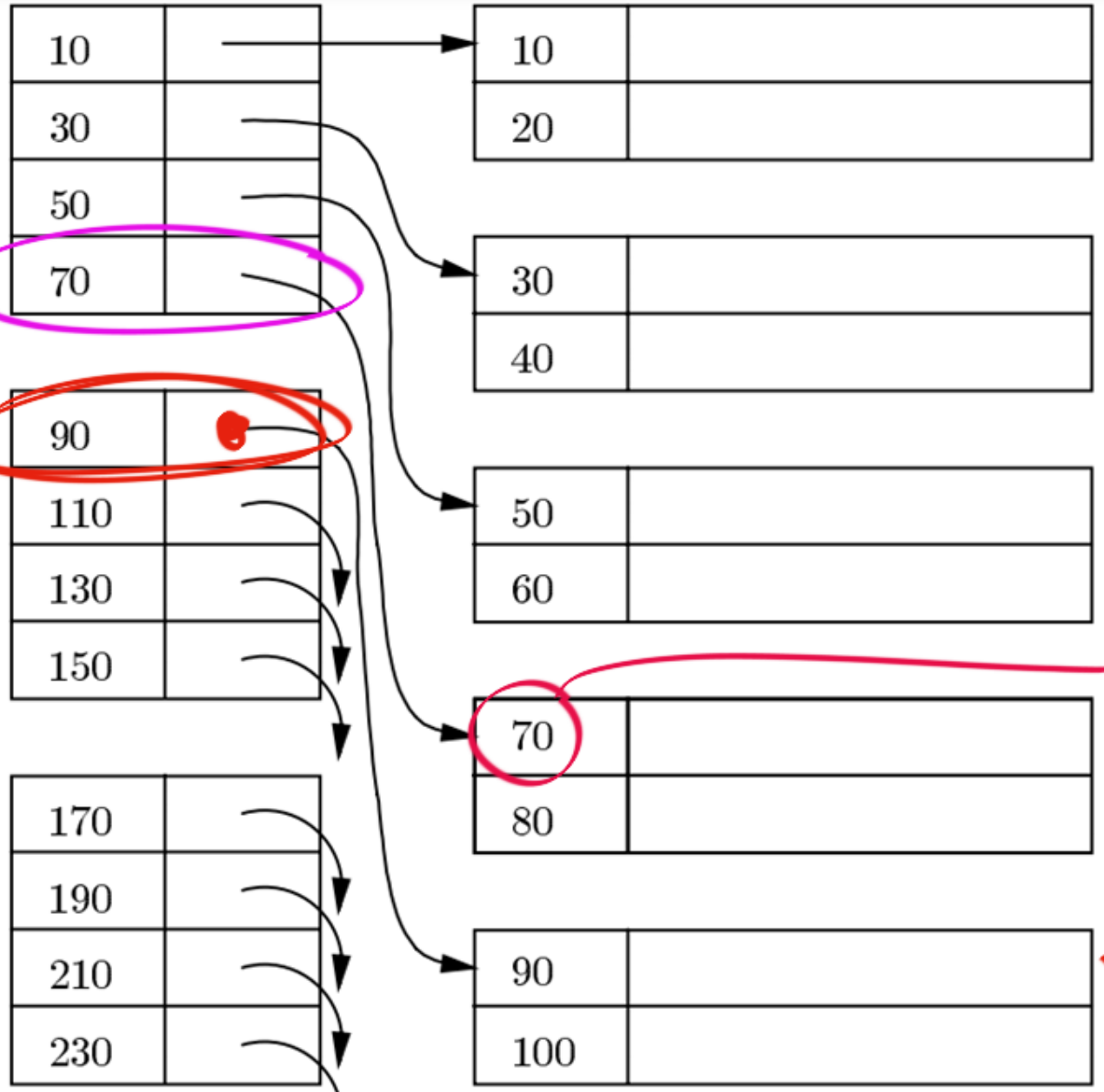
Index entry

Block Anchor

Search 60 in Index

#Block Access = 2 + 1

Database System Implementation, H. Garcia-Molina, J. Ullman, and J. Widom, Prentice-Hall, 2000



Sparse Implementation:

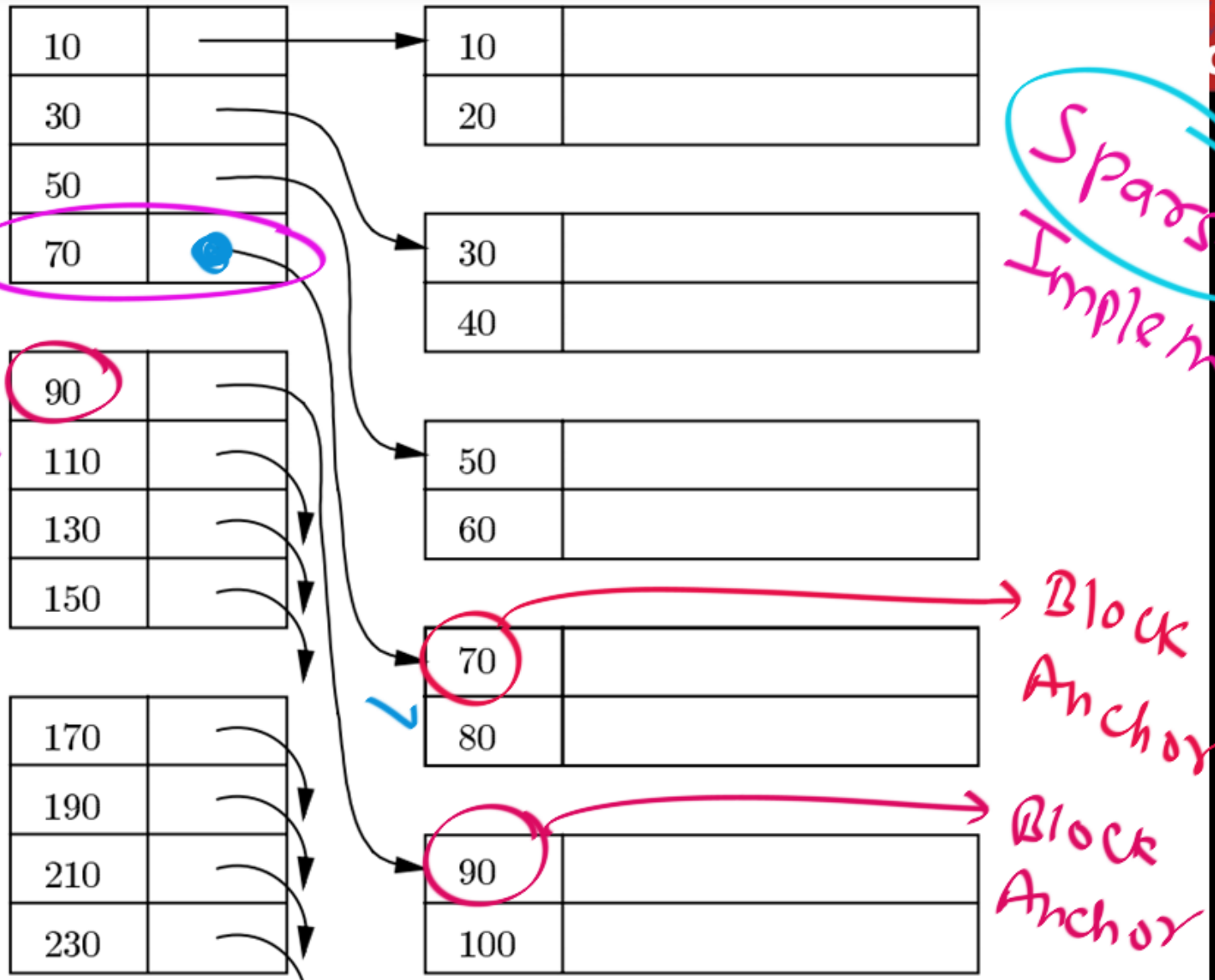
Index entry

Search 90 in Index

Block Anchor

Block access = 1+1

Database System Implementation, H. Garcia-Molina, J. Ullman, and J. Widom, Prentice-Hall, 2000.



Sparse Implementation:

Index entry

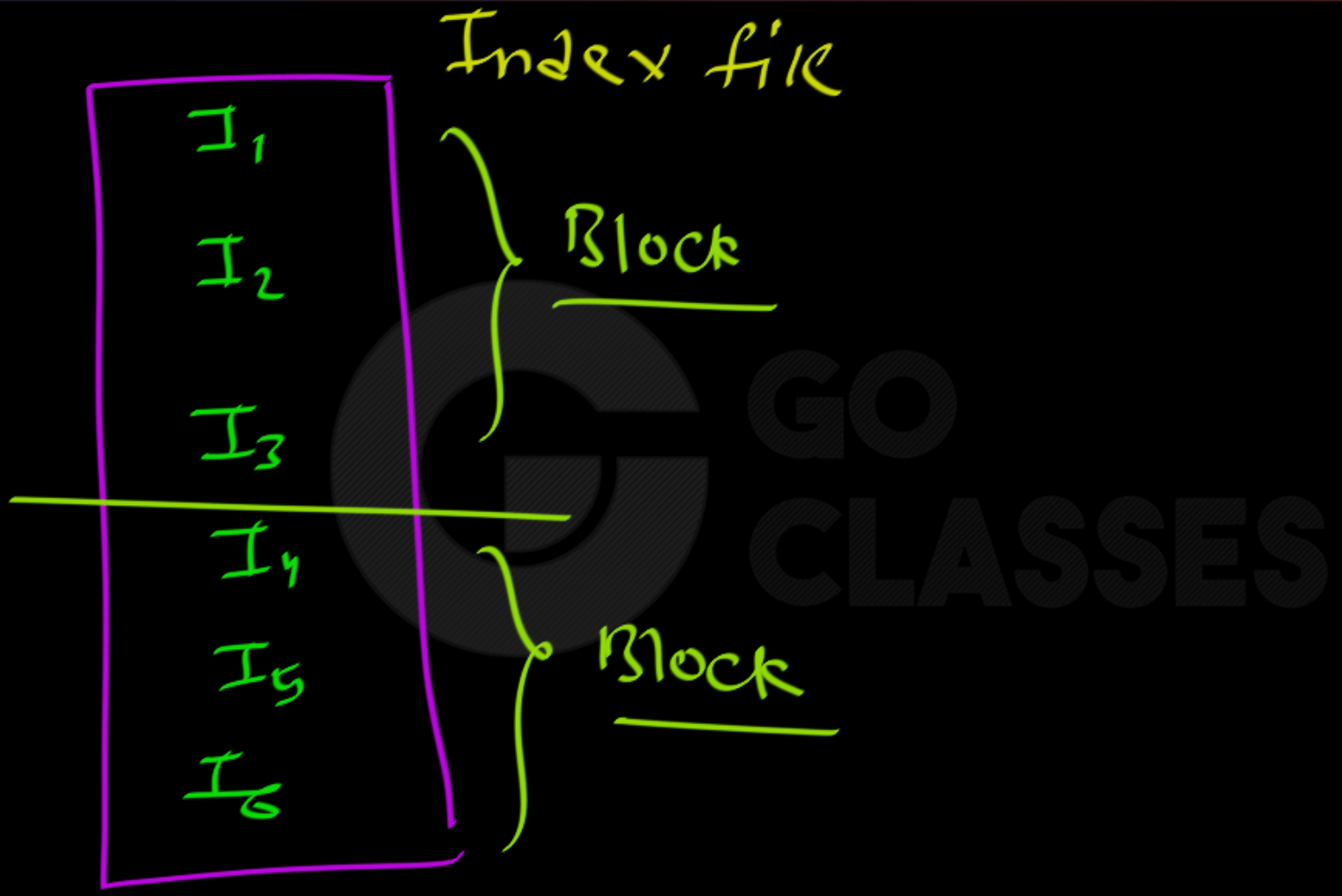
Block Anchor

Block Anchor

Search 85 in Index

Block Access = 2+1

Database System Implementation, H. Garcia-Molina, J. Ullman, and J. Widom, Prentice-Hall, 2000.


Index file
I_1
I_2
I_3
I_4
I_5
I_6
Block
Block

Primary Index:

- Dense implementation
  - index entry for every Record
  - possible
  - (But NOT used)
- Sparse implementation
  - for primary index, one index entry for one Block!
  - Uses

Primary Index is Sparse index ; True

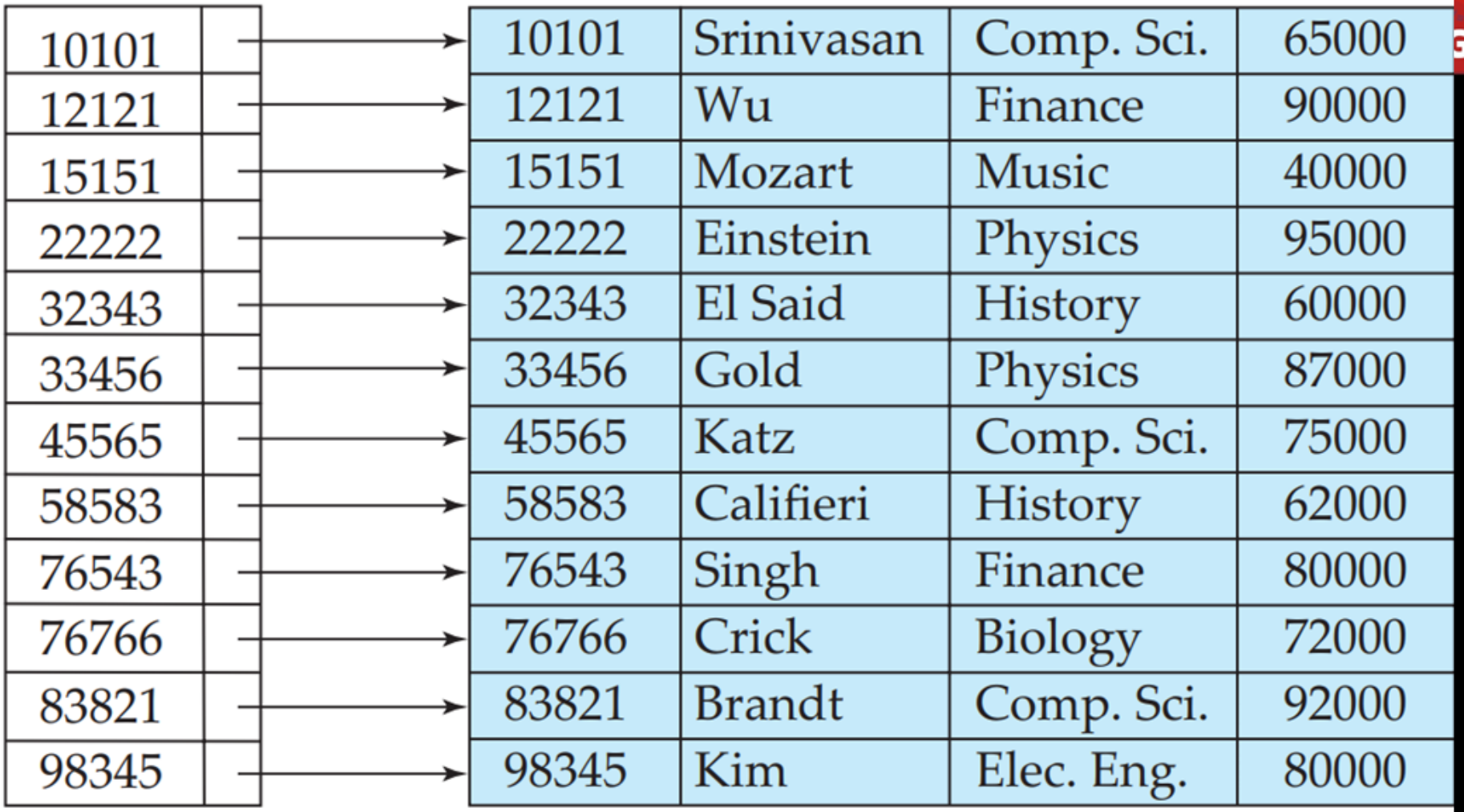


Figure 11.2 Dense index.

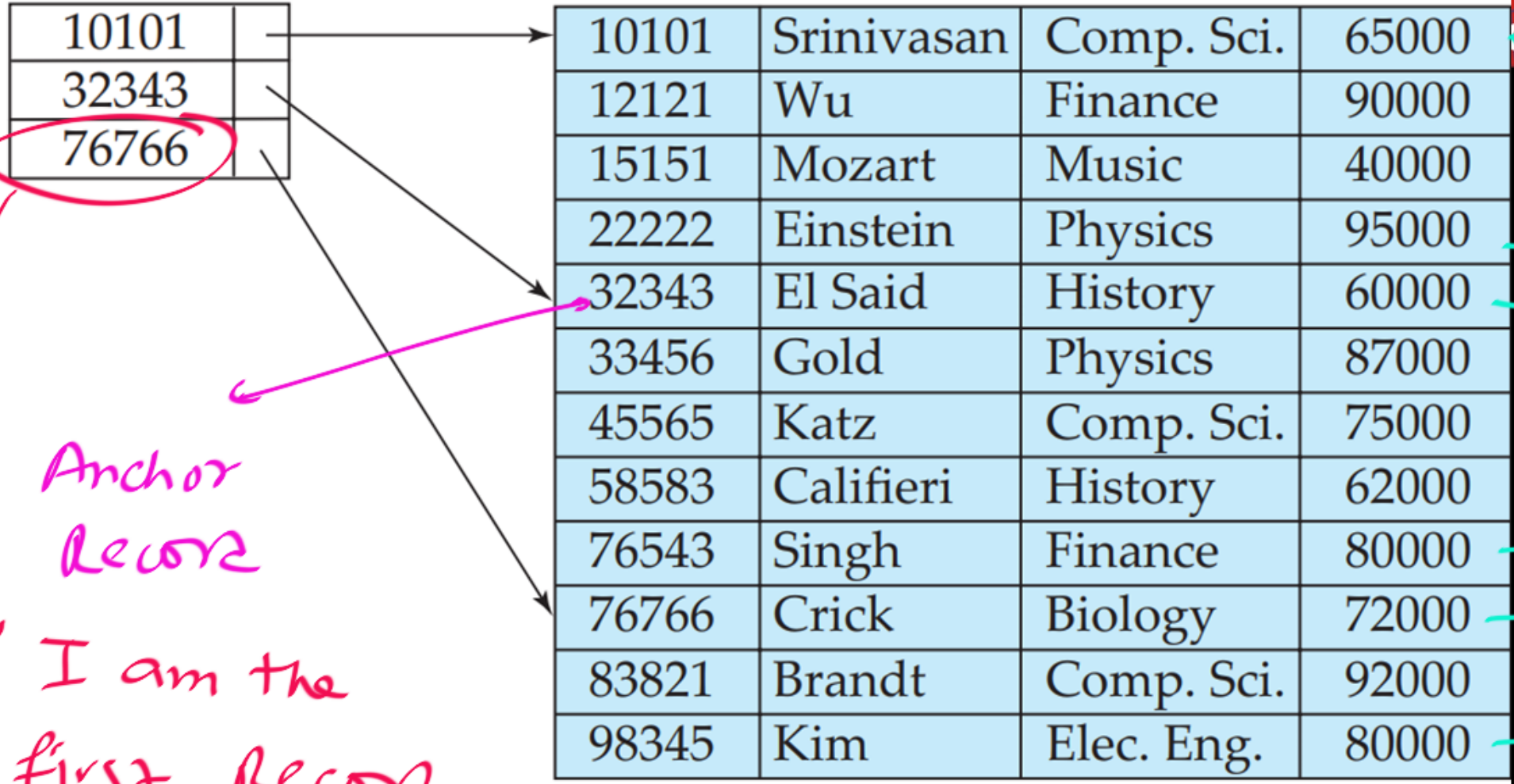
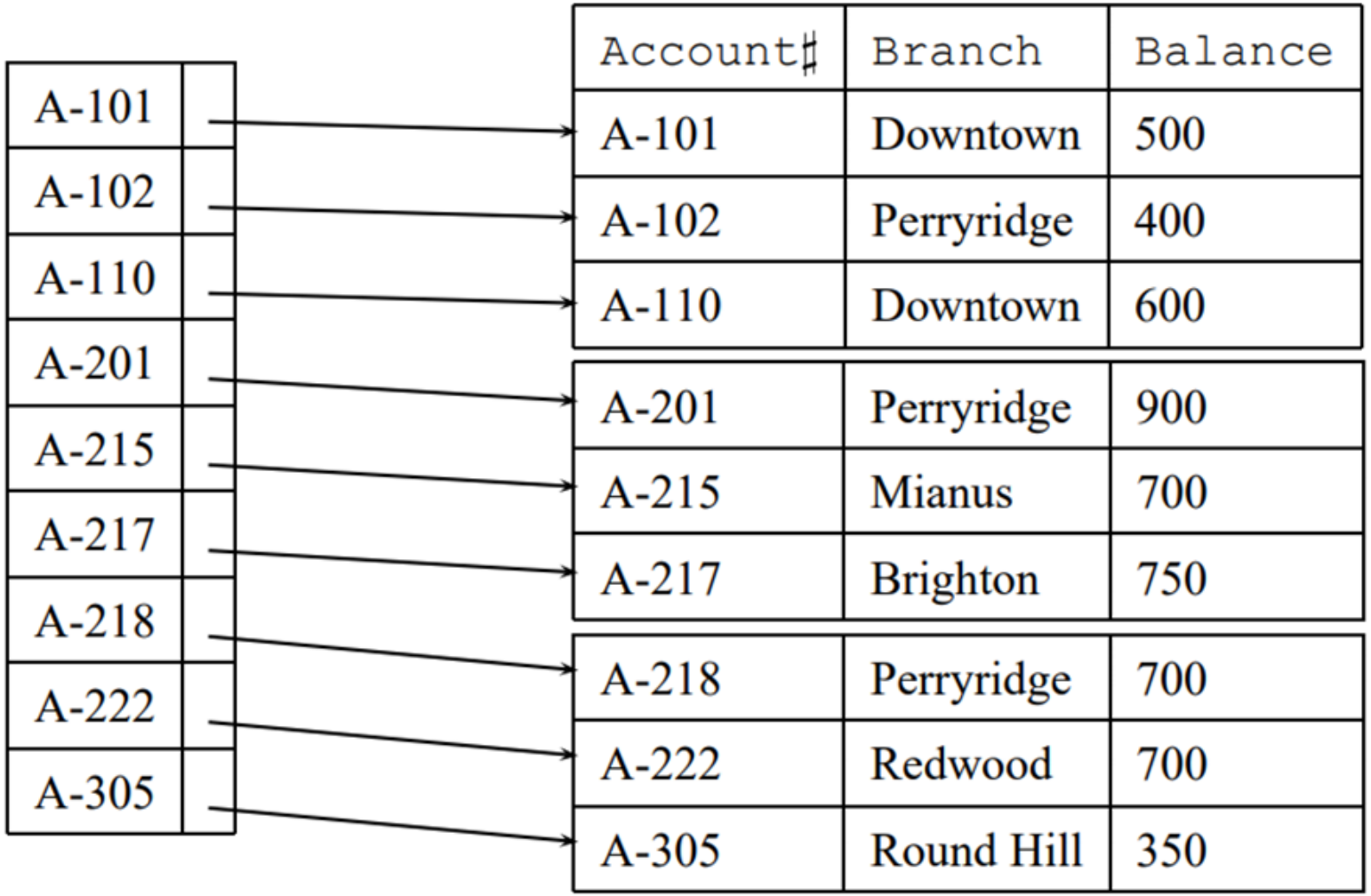


Figure 11.3 Sparse index.

Primary Dense Index: Example

Example of a **Primary Dense Index** with **Search Key**=Account#.



Primary Sparse Index: Example

Example of a **Primary Sparse Index** with **Search Key**=Account#.

I am the first record in my Block in Data file

Account#
A-101
A-201
A-218

Account#	Branch	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600
A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Block Anchor

Primary Sparse Index: Example

Example of a **Primary Sparse Index** with **Search Key**=Account#.

I am the first record in my Block in Data file

Account#
A-101
A-201
A-218

Account#	Branch	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600
A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Block Anchor

Search A-216

Two types of Indexes:

Dense

Sparse

Dense Index:

In a dense index, an index entry appears for every search-key value in the file.

Sparse index:

In a sparse index, an index entry appears for only some of the search-key values.

NOTE:

Primary Index is, by default, Sparse Index.

In primary index, we make One index entry for one Data block.

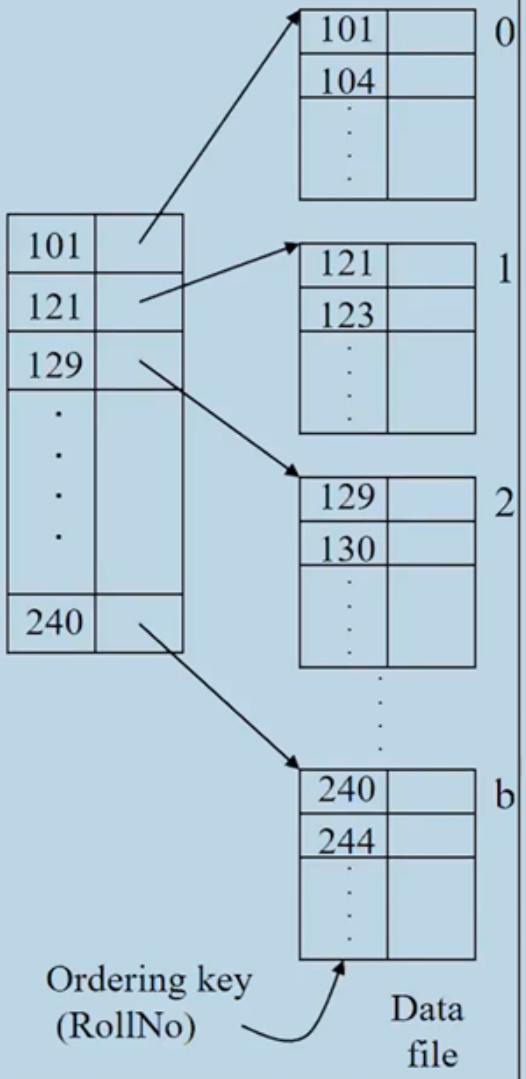
Primary Index

Can be built on ordered / sorted files

Index attribute – ordering key field (OKF)

Index Entry:

value of OKF for the first record of a block B_j	disk address of B_j



Primary Index:

Sparse index: (by Default)

(Serach key value of Anchor Record of Data Block

Can Not be a Record Pointer

Block Pointer

Primary Index:

Dense index: (Not Used)

(Search key value
For a Record)

Pointer)

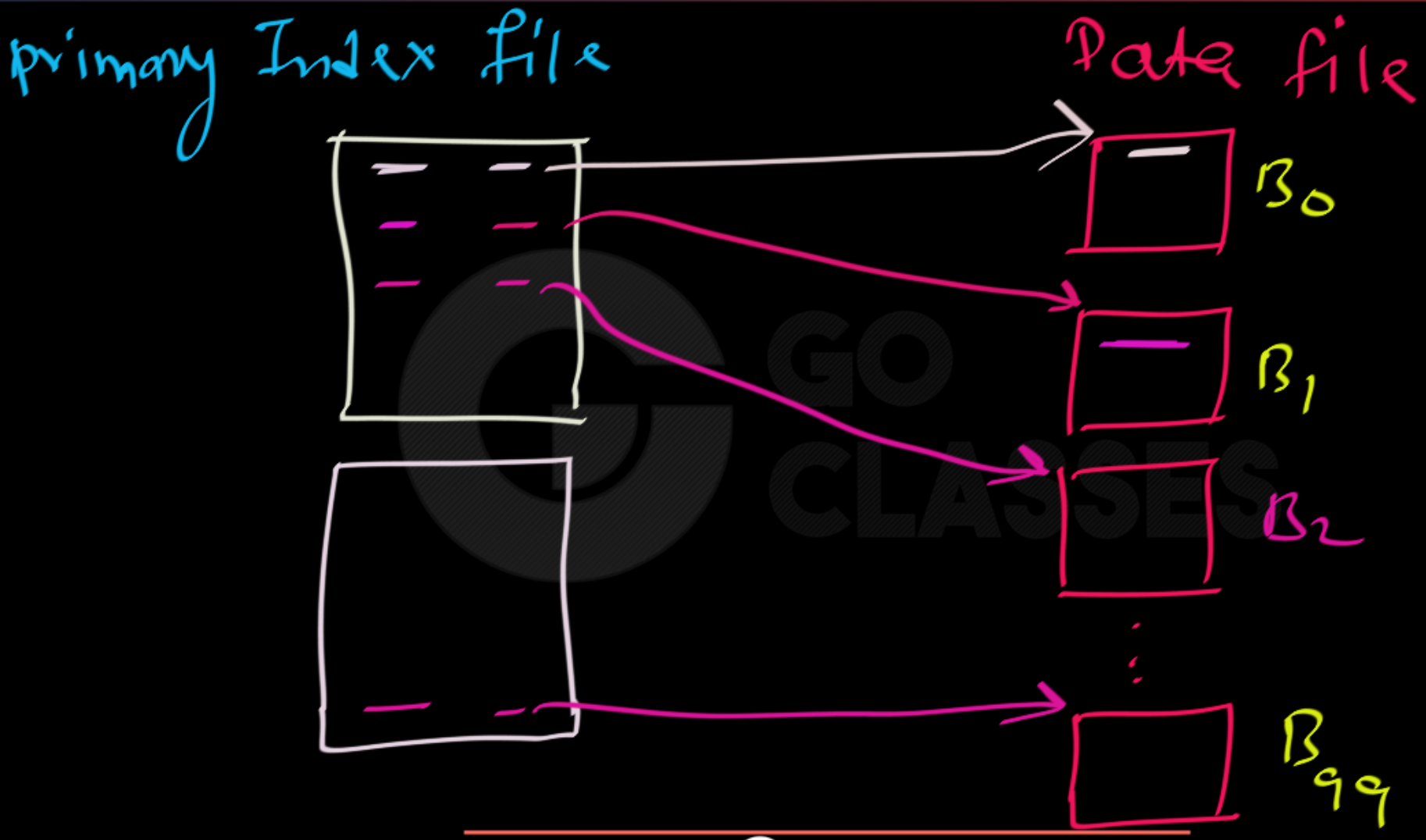
Record or Block Pointer

Note:

Assume Sequential file with 50000 Records;

Data Blocks = 1000

Index Entries in Primary index = # Data Blocks = 1000


primary Index file
Data file
B_0
B_1
B_2
B_{99}

Access Cost : #Blocks Access

Binary Search : On Blocks of a ordered file

Significant Amount of time

On Records inside a Block : Negligible.

We do Binary Search on Blocks of ordered file by

Primary Index

Can be built on ordered / sorted files

Index attribute – ordering key field (OKF)

Index Entry:

value of OKF for the first record of a block B_j	disk address of B_j

Index file: ordered file (sorted on OKF)

size: no. of blocks in the data file

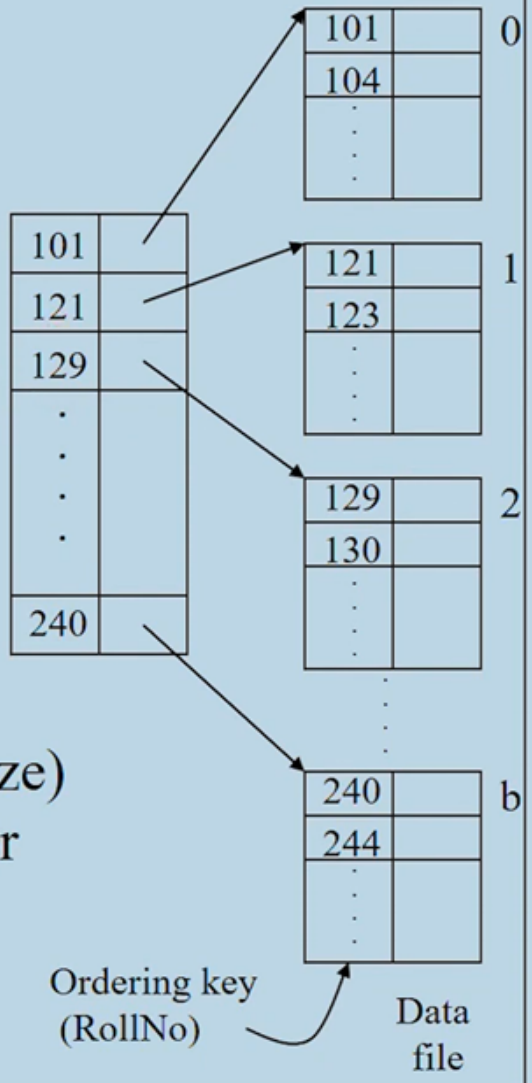
Index file blocking factor $BF_i = \lfloor B/(V+P) \rfloor$

(B-block size, V-OKF size, P-block pointer size)

- generally more than data file blocking factor

No of Index file blocks $b_i = \lceil b/BF_i \rceil$

(b - no. of data file blocks)



Indexes on Sequential Files

- **I**ndex on Sequential File, also called **P**rimary Index, when the Index is associated to a *Data File* which is in turn *sorted with respect to the search key*.
  1. A Primary Index forces a sequential file organization on the Data File;
  2. Since a Data File can have just one order there can be just one Primary Index for Data File.
- Usually used when the search key is also the primary key of the relation.
- Usually, these indexes fit in main memory.
- Indexes on sequential files can be:
  1. **D**ense: One entry in the index file for every record in the data file;
  2. **S**parse: One entry in the index file for each block of the data file.

Primary Index:

The total number of entries in the index is the same as the number of disk blocks in the ordered data file. (Because for Primary Index, we by default make Sparse Index)

The first record in each block of the data file is called the anchor record of the block, or simply the block anchor.

A **primary index** is an ordered file whose records are of fixed length with two fields, and it acts like an access structure to efficiently search for and access the data records in a data file. The first field is of the same data type as the ordering key field—called the **primary key**—of the data file, and the second field is a pointer to a disk block (a block address). There is one **index entry** (or **index record**) in the index file for each *block* in the data file. Each index entry has the value of the primary key field for the *first* record in a block and a pointer to that block as its two field values.

Indexes can also be characterized as dense or sparse. A **dense index** has an index entry for *every search key value* (and hence every record) in the data file. A **sparse** (or **nondense**) **index**, on the other hand, has index entries for only some of the search values. A sparse index has fewer entries than the number of records in the file. Thus, a primary index is a nondense (sparse) index, since it includes an

entry for each disk block of the data file and the keys of its anchor record rather than for every search value (or every record).

Q: Access Cost of Ordered file without Primary Index:

We know that, to search in Ordered file, we can use Binary Search on disk blocks of the file.
If file contains b blocks then access cost in terms of number of disk block access is??

Q: Access Cost of Ordered file without Primary Index:

Worst Case

We know that, to search in Ordered file, we can use Binary Search on disk blocks of the file.

If file contains b blocks then access cost in terms of number of disk block access is?? $\Rightarrow \lceil \log_2 b \rceil$

Q: Access Cost of Ordered file with Primary Index:

We create Primary Index on a Sequential (Ordered) file. We know that, to search in Ordered file, we can use Binary Search on disk blocks of the file. Assume that we have created a Primary Index on this file, with Index file having I blocks.

If file contains b blocks then access cost in terms of number of disk block access is??

Q: Access Cost of Ordered file with Primary Index:

We create Primary Index on a Sequential (Ordered) file. We know that, to search in Ordered file, we can use Binary Search on disk blocks of the file. Assume that we have created a Primary Index on this file, with Index file having I blocks.

If file contains b blocks then access cost in terms of number of disk block access is?? $\Rightarrow \lceil \log_2 I \rceil + 1$

An Example

Data file:

- No. of blocks b = 9500
- Block size B = 4KB
- OKF length V = 15 bytes
- Block pointer length p = 6 bytes

Index file on OFK :

- No. of records r_i =
- Size of entry
- Blocking factor BF_i =
- No. of blocks b_i =

Max No. of block accesses for getting record using the primary index

Max No. of block accesses for getting record without using primary index

Prof P Sreenivasa Kumar
Department of CS&E, IITM

An Example

Data file:

- No. of blocks b = 9500
- Block size B = 4KB
- OKF length V = 15 bytes
- Block pointer length p = 6 bytes

Index file

- No. of records r_i =
- Size of entry
- Blocking factor BF_i =
- No. of blocks b_i =

Max No. of block accesses for getting record using the primary index

Max No. of block accesses for getting record without using primary index

Prof P Sreenivasa Kumar
Department of CS&E, IITM

Primary Index:

(1) #index entries = #Blocks in Data file

= 950

(2) Size of index entry = Ofk Size + Block Pointer

(Ofk Size → search key)

= 15 + 6 = 21 B

(3) Blocking factor of Index file :

→ #Records per Block

#index entries per Block

2^{10}

Index entry size = 21 B
Block size = 4 kB

$$\left\lfloor \frac{4\,kB}{21\,B} \right\rfloor = 195$$

(4) # Blocks needed for Index file:

One Block ⇒ 195 index entries

→ 9500 index entries

$$\left\lceil \frac{9500}{195} \right\rceil = 49 \text{ Index Blocks}$$

(5) max Access Cost using primary index :

$$\lceil \log_2 49 \rceil + 1 = 7$$

for Data Block

7 Blocks need to access in worst case

(5) max Access Cost without primary index :

$\lceil \log_2 9500 \rceil = 14$

Binary Search on Data Blocks

9500 Data Blocks

Blocks need to access in worst case

An Example

Data file:

- No. of blocks b = 9500
- Block size B = 4KB
- OKF length V = 15 bytes
- Block pointer length p = 6 bytes

Index file

- No. of records $r_i = 9500$
- Size of entry V + P = 21 bytes
- Blocking factor $BF_i = \lfloor 4096/21 \rfloor = 195$
- No. of blocks $b_i = \lceil r_i/BF_i \rceil = 49$

Max No. of block accesses for getting record using the primary index | $1 + \lceil \log_2 {}^{b_i} \rceil = 7$

Max No. of block accesses for getting record without using primary index | $\lceil \log_2{}^{b} \rceil = 14$

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Assume the file is ordered by a Key field V, & we make a search query to find the record having V=7 then what is the worst case number of disk blocks we need to access for this query??

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Assume the file is ordered by a Key field V, & we make a search query to find the record having V=7 then what is the worst case number of disk blocks we need to access for this query??

∴ Binary Search on ordered file

Record Size = 100 B

Block " = 4096 B

$$\#\text{Records per Block} = \left\lfloor \frac{4096}{100} \right\rfloor = 40$$

$$\#\text{Blocks} = \left\lceil \frac{30,000}{40} \right\rceil = 750$$

$$\text{Access cost} = \lceil \log_2 750 \rceil = 13 \text{ Blocks}$$

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Assume the file is ordered by a Key field V, & we make a search query to find the record having A = 7 then what is the worst case number of disk blocks we need to access for this query?? ⇒ 1

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Assume the file is ordered by a Key field V, & we make a search query to find the record having A = 7 then what is the worst case number of disk blocks we need to access for this query??

⇒ 7500 Blocks

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes.
Now suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.

The number of Blocks needed to store the Primary Index is??

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes.

Now suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.

on V

The number of Blocks needed to store the Primary Index is??

Index Record Size: 9 + 6 = 15 B

Search key size = 9 B

Block pointer

#Index Records = #Data Blocks = 750

#Index Entries = 750

#Index Blocks ?

#index entries per Block : $\left\lfloor \frac{4096\ B}{15\ B} \right\rfloor$

$= 273$

#Index Blocks : $\left\lceil \frac{7500}{273} \right\rceil = 28$ Blocks

Q: Suppose that we have an ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes.

Now suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.

We make a search query to find the record having V=7 then what is the worst case number of disk blocks we need to access for this query??

Access Cost using primary Index:

$$= \lceil \log_2 I \rceil + 1$$

$$= 5 + 1 = 6$$

Blocks

without Index:

Access Cost = 13 Blocks

Example 1. Suppose that we have an ordered file with $r = 300{,}000$ records stored on a disk with block size $B = 4{,}096$ bytes.[5] File records are of fixed size and are unspanned, with record length $R = 100$ bytes. The blocking factor for the file would be $bfr = \lfloor (B/R) \rfloor = \lfloor (4{,}096/100) \rfloor = 40$ records per block. The number of blocks needed for the file is $b = \lceil (r/bfr) \rceil = \lceil (300{,}000/40) \rceil = 7{,}500$ blocks. A binary search on the data file would need approximately $\lceil \log_2 b \rceil = \lceil (\log_2 7{,}500) \rceil = 13$ block accesses.

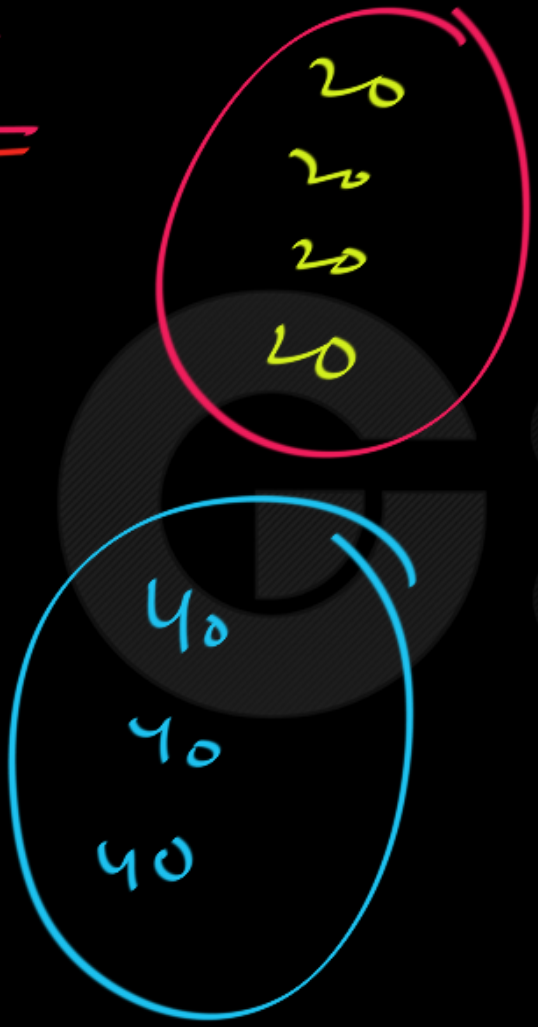
Now suppose that the ordering key field of the file is $V = 9$ bytes long, a block pointer is $P = 6$ bytes long, and we have constructed a primary index for the file. The size of each index entry is $R_i = (9 + 6) = 15$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4{,}096/15) \rfloor = 273$ entries per block. The total number of index entries r_i is equal to the number of blocks in the data file, which is 7,500. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (7{,}500/273) \rceil = 28$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 28) \rceil = 5$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of 5 + 1 = 6 block accesses—an improvement over binary search on the data file, which required 13 disk block accesses. Note that the index with 7,500 entries

The index file for a primary index occupies a much smaller space than does the data file, for two reasons. First, there are *fewer index entries* than there are records in the data file. Second, each index entry is typically *smaller in size* than a data record because it has only two fields, both of which tend to be short in size; consequently, more index entries than data records can fit in one block. Therefore, a binary search on the index file requires fewer block accesses than a binary search on the data file. Referring to Table 16.3, note that the binary search for an ordered data file required $\log_2 b$ block accesses. But if the primary index file contains only b_i blocks, then to locate a record with a search key value requires a binary search of that index and access to the block containing that record: a total of $\log_2 b_i + 1$ accesses.

Types of Single level Ordered Indexes:

2. Clustering Index

Clusters:



Clustering Index:

If the file containing the records is sequentially ordered by a Non-key field F, then the Index using search key F is called Clustring Index.

17.1.2 Clustering Indexes

If file records are physically ordered on a nonkey field—which *does not* have a distinct value for each record—that field is called the **clustering field** and the data file is called a **clustered file.** We can create a different type of index, called a **clustering index**, to speed up retrieval of all the records that have the same value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a *distinct value* for each record.

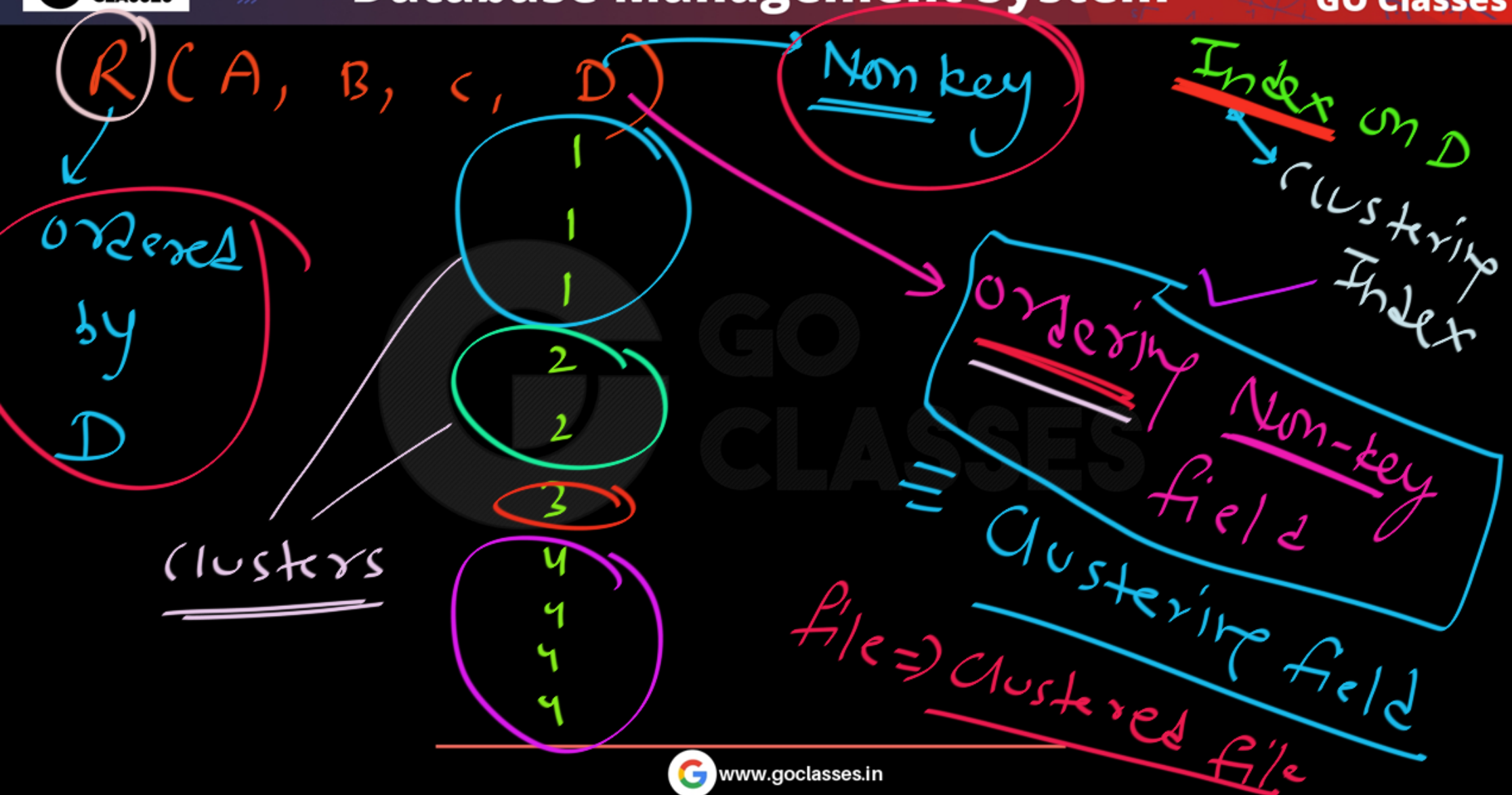
file R(A, B, C, D)

Non-key

key

R physically ordered on D

ordering field for R


R(A, B, C, D)
Ordered by D
Clusters
1
1
1
1
2
2
3
4
4
4
4
Non key
Index on D
Clustering Index
Ordering Non-key field
≡ Clustering field
File ⇒ Clustered file

Clustering Index:

Index on Ordering Non-key field

Q:

From DBMS point of view, Index at the end of RaghuRama Krishnana book (Or any other standard book), is Clustering Index ???? No ⇒ because Data file is NOT sorted on search key : Topic Name

SUBJECT INDEX

1NF, 430
2NF, 434
2PC, 628, 630
  blocking, 630
  with Presumed Abort, 631
2PL, 542
  distributed databases, 624
3NF, 432, 440, 443
3PC, 632
4NF, 447
5NF, 449
A priori property, 710
Abandoned privilege, 504
Abort, 525–526, 548, 556, 574, 584, 628
Abstract data types, 742
ACA schedule, 531
Access control, 8, 497–498
Access methods, 217
Application servers, 647
Architecture of a DBMS, 18
ARIES recovery algorithm, 571, 587
Armstrong's Axioms, 427
Array chunks, 693, 760
Arrays, 746
Assertions in SQL, 163
Association rules, 714, 716
  use for prediction, 718
  with calendars, 717
  with item hierarchies, 715
Asynchronous replication, 611, 620–621, 681
  Capture and Apply, 622
  change data table (CDT), 622
  conflict resolution, 621
  peer-to-peer, 621
  primary site, 621

{ Book Vs Index

Data file Vs Index file }

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

Can we create a Clustering Index on a heap file ??

Q:

Consider a Heap file (Do you remember what a Heap file is ???)

NOT ordered by Any field

Can we create a Clustering Index on a heap file ??

NO

NOTE:

So, to create Clustering Index on a file R:

R MUST be PHYSICALLY Ordered by a Non-Key Field F...

Then a Index by F is called Clustering Index.

Q:

Yes.

Can we create Clustering Index on the following File? (Assume the Order of recorded shown is the physical order of records)

If yes, then on which search key we should create Clustering Index??

subject name

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Q:

How many Clustering Indexes can be created on a file ?

Q:

How many Clustering Indexes can be created on a file ? ⇒ At most 1

Q:

On a file, Can we create both Primary as well as Clustering Index ??

Q:

On a file, Can we create both Primary as well as Clustering Index ??

NEVER

Can Never happen

for Primary Index: file ordered by key

for Clustering Index: " " " Non-key

Clustering Index:

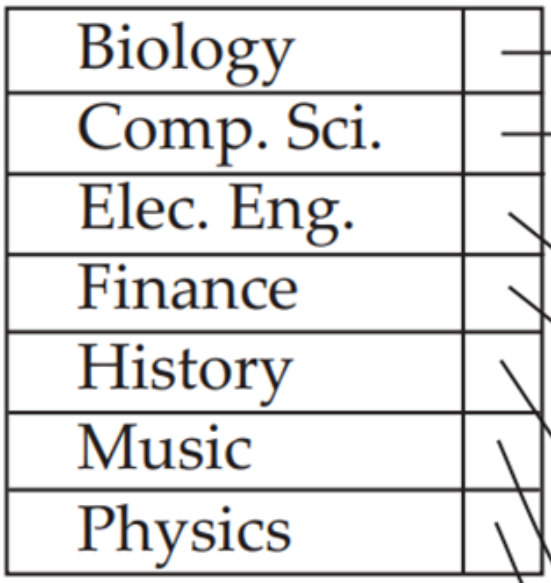
Implementation

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Clustring Index :

Compsci

History



Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

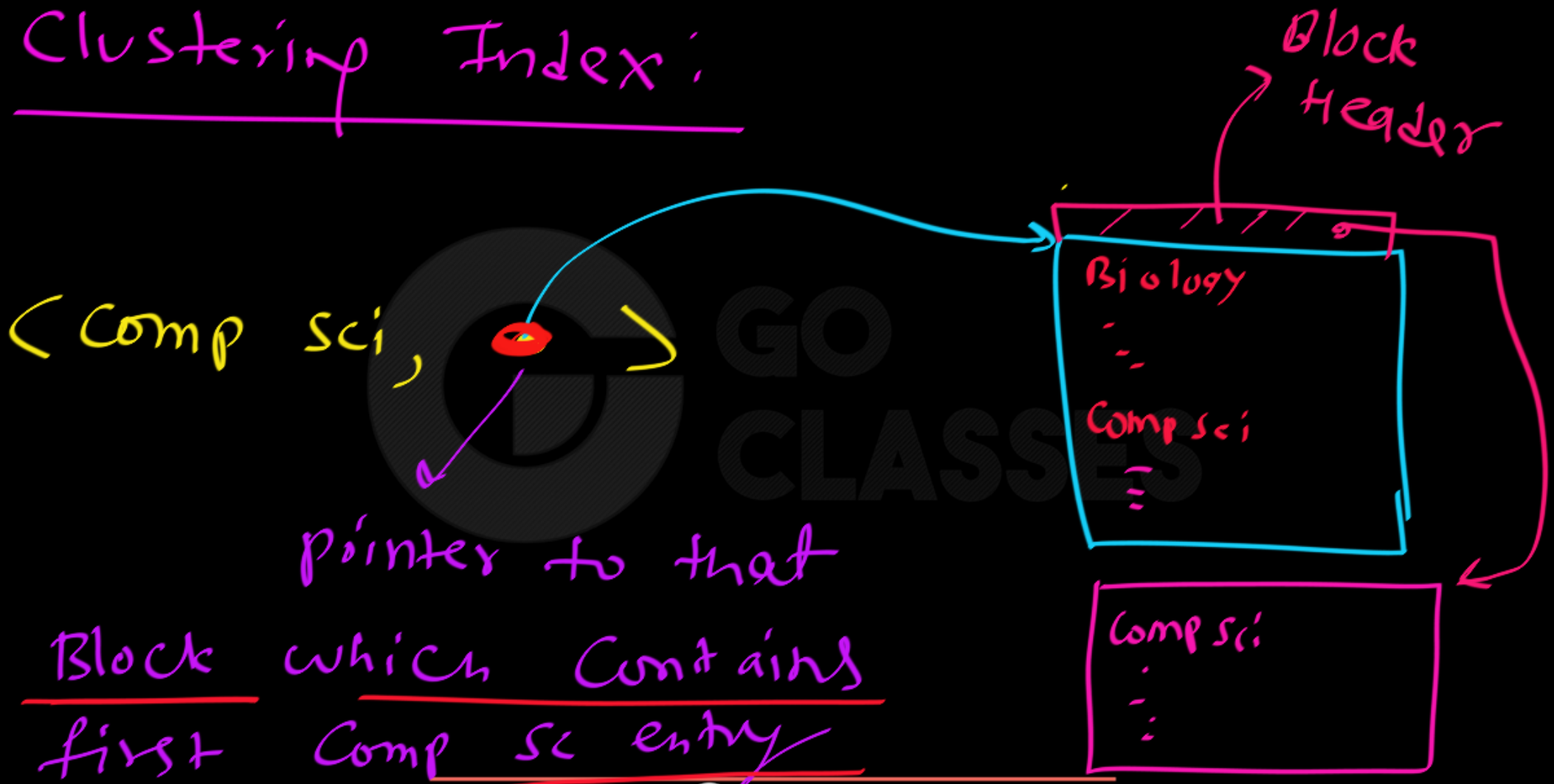
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Clustering Index:

Index entries = # of clusters
= # Distinct search key values

Index Entry : < search key value , Block pointer >

Clustering Index:



Clustering Index

Built on ordered files where ordering field is *not a key*

Index attribute: ordering field (OF)

Index entry:

Distinct value V_i of the OF	address of the first block that has a record with OF value V_i

Index file: Ordered file (sorted on OF)

size – no. of distinct values of OF

A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file, and the second field is a disk block pointer. There is one entry in the clustering index for each *distinct value* of the clustering field, and it contains the value and a pointer to the *first block* in the data file that has a record with that value for its clustering field. Figure 17.2 shows an

A clustering index is another example of a *nondense* index because it has an entry for every *distinct value* of the indexing field, which is a nonkey by definition and hence has duplicate values rather than a unique value for every record in the file.

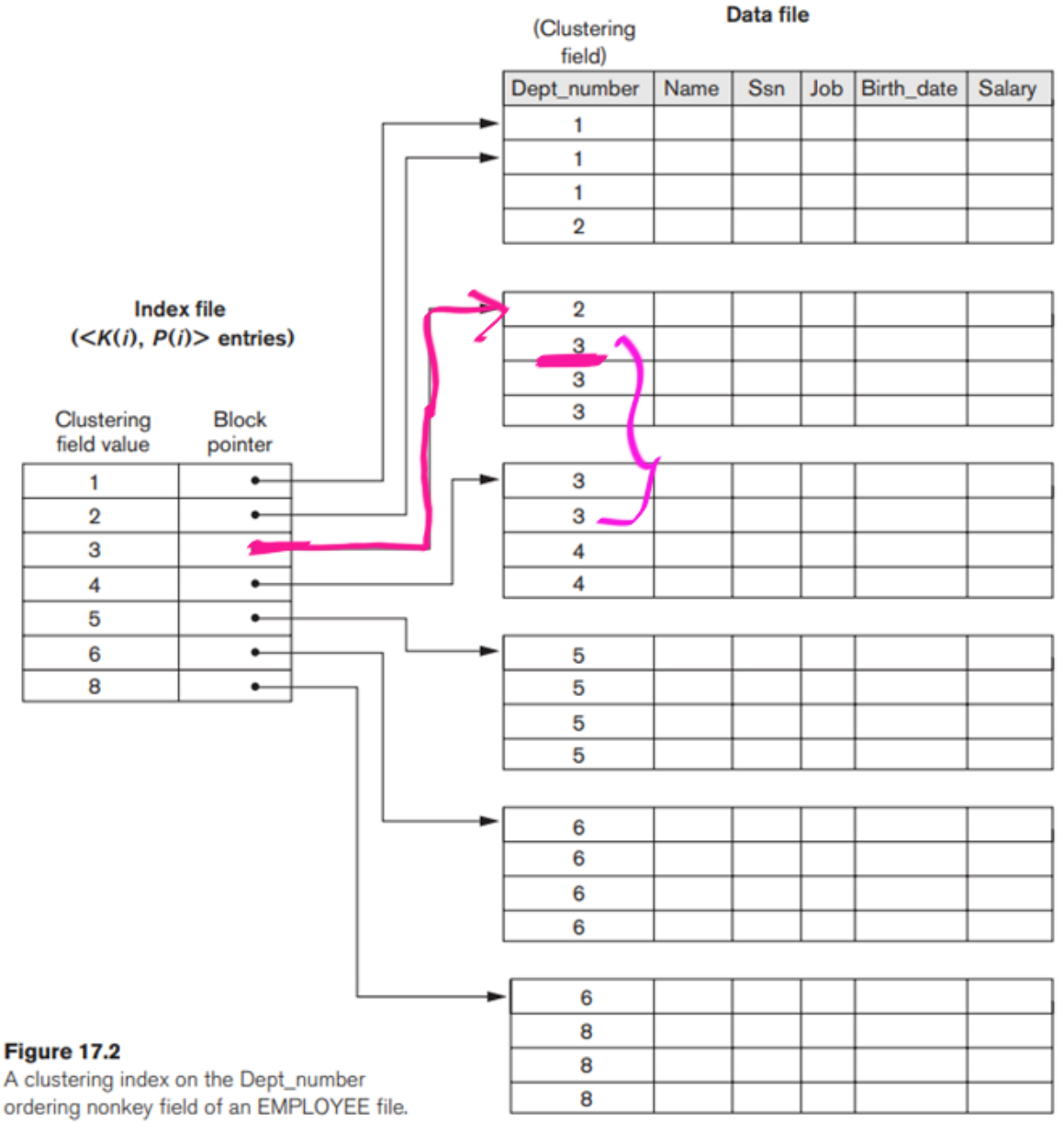
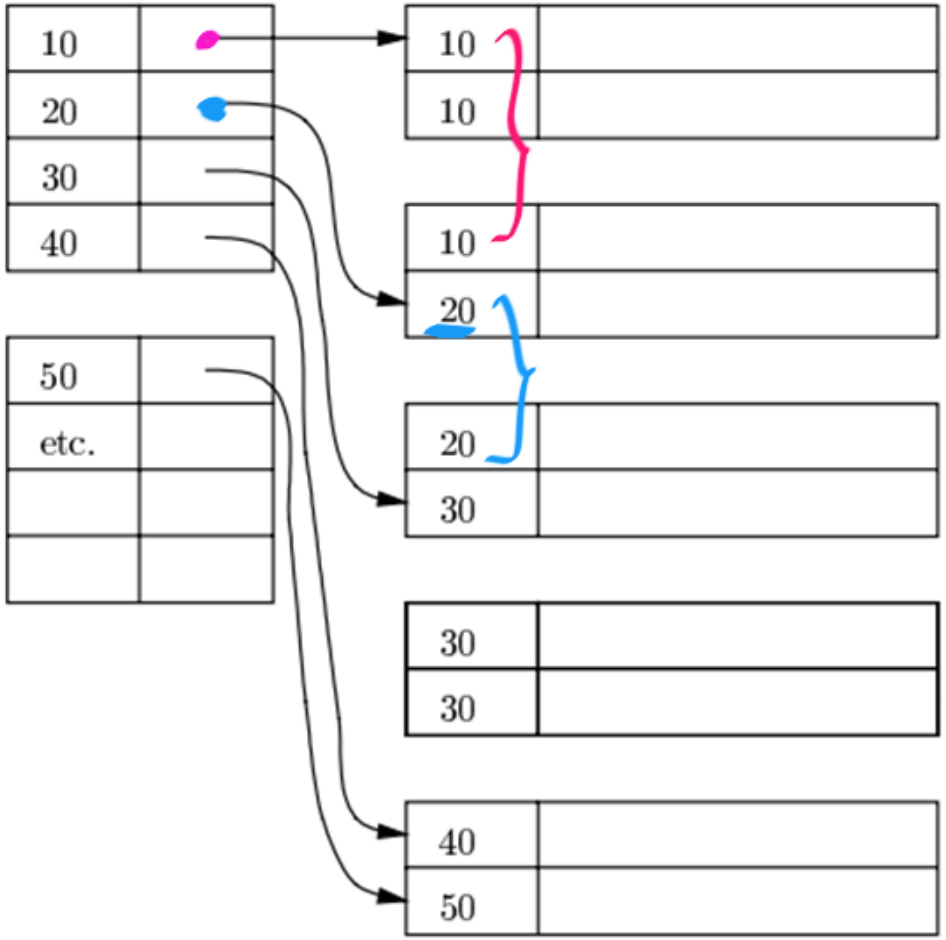
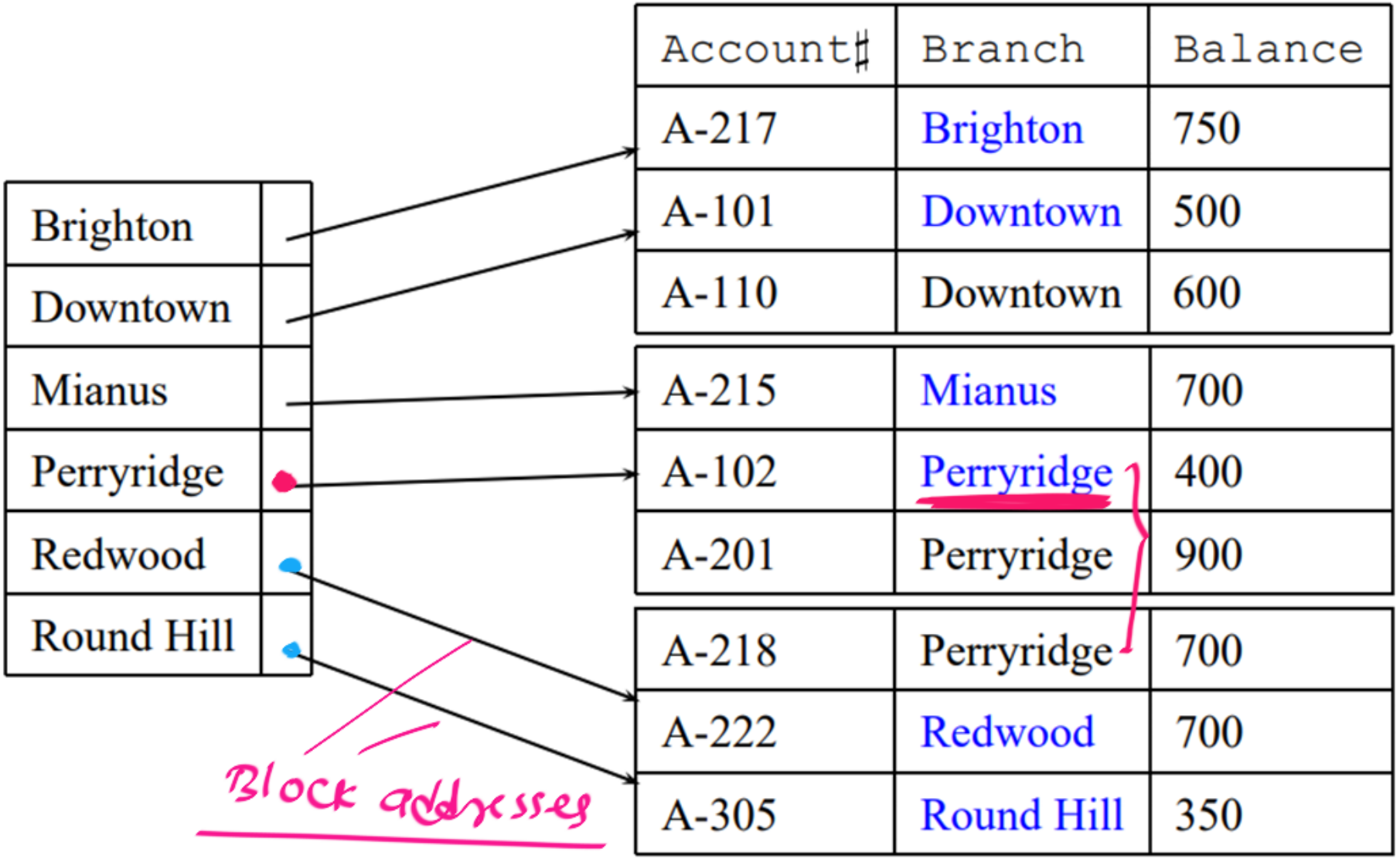


Figure 17.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

Lookup. Find the search key on the index, read the pointed disk block, possibly read successive blocks.





NOTE:

Clustering Index is a Sparse Index.

In Clustering index, we make One index entry for Every Distinct Value of search key in the data file.

Q: Suppose that we consider the same ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.) Assume 5-byte Zipcode and 6-byte block pointer.

We create Index on Zipcode. Number of Index entries? Blocking Factor of the Index?? Number of Index Blocks?

Example 2. Suppose that we consider the same ordered file with $r = 300{,}000$ records stored on a disk with block size $B = 4{,}096$ bytes. Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.) The index in this case has 1,000 index entries of 11 bytes each (5-byte Zipcode and 6-byte block pointer) with a blocking factor $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4{,}096/11) \rfloor = 372$ index entries per block. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (1{,}000/372) \rceil = 3$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 3) \rceil = 2$ block accesses. Again, this index would typically be loaded in main memory (occupies 11,000 or 11 Kbytes) and takes negligible time to search in memory. One block access to the data file would lead to the first record with a given zip code.