COMPUTER SCIENCE

Database Management System

File Org. & Indexing

Lecture_4

Vijay Agarwal sir


TOPICS TO BE COVERED
01
Multi level Indexing
02
B & B+ Tree

File org.

'Blocks' ≡ (Pages)

① SPANNED ORG (Variable Length Record)

② UnSpanned ORG (Fixed Length Record)

↳ No Record can be Partially stored More than One Block.

① ORDERED File

$\lceil \log_2 B \rceil$

② Unordered File

Avg = $B/2$

Worst = B

→ unordered (Hash function)

Indexing

↳ ordered File.

One Index Record Size = Size of key + Size of Pointer.

To Access Index Block Avg # Block Access = $\lceil \log_2 B_i \rceil$

To Access a Record using Index Avg # Block Access = $\lceil \log_2 B_i \rceil + 1$

Dense Index

Index Entries = # Records.

SPARSE (Non Dense) Index

Index Entries = # DB Blocks.

Types of Index

Single-level Ordered Indexes

- Primary indexes
- Clustering indexes
- Secondary indexes

Multilevel Indexes

Dynamic multilevel indexes Using B-Tress and B^+ Trees.

① Primary Index (Key + ordered DB File)

② Clustering Index (Non key + ordered DB File)

③ Secondary Index (① Non key + Unordered DB File
② key + ordered DB File)

Indexing (Basic Concepts)

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

- Index files are typically much smaller than the original file.
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Index file block size is same as DB file Block Size

Block Size of Index File = Block Size of DB file

One Index Record Size = Size of Search Key + Size of Block Pointer

NOTE: To Access a Record Average number of block access

$= \log_2 B_i + 1$

Index Block access

Data Block access

[Bi : Index Block]

Multilevel Index: [ISAM]

'SPARSE'

1) 1st level index is index to DB file and 2nd level onward Index to index file until 1 block index at last level.
2) Idle access cost to access record using multi level index is (n + 1) blocks, n is number of level in index.

Categories of Index:

1) Dense Index [More entries in Index File]
2) Sparse Index [Less entries in Index File]

(Non Dense)

Category of Index

1) Dense Index Files

Number of Index entries = Number of DB Records

2) Sparse / Non Dense Index Files

Number of Index entries = Number of Blocks

Types of Index

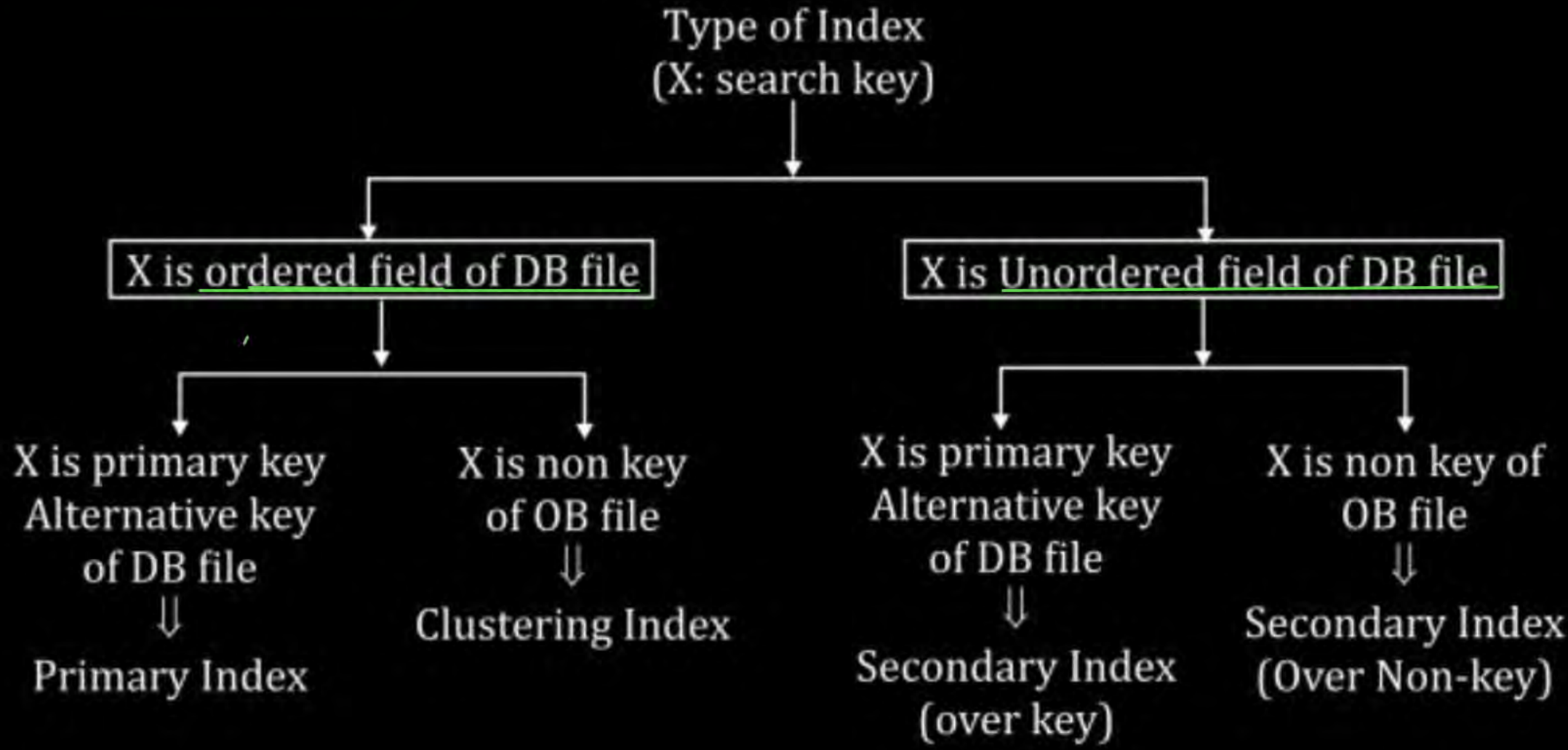
Single-level Ordered Indexes

- Primary indexes
- Clustering indexes
- Secondary indexes

Multilevel Indexes

Dynamic multilevel indexes Using B-Tress and B^+ Trees.

Types of Index

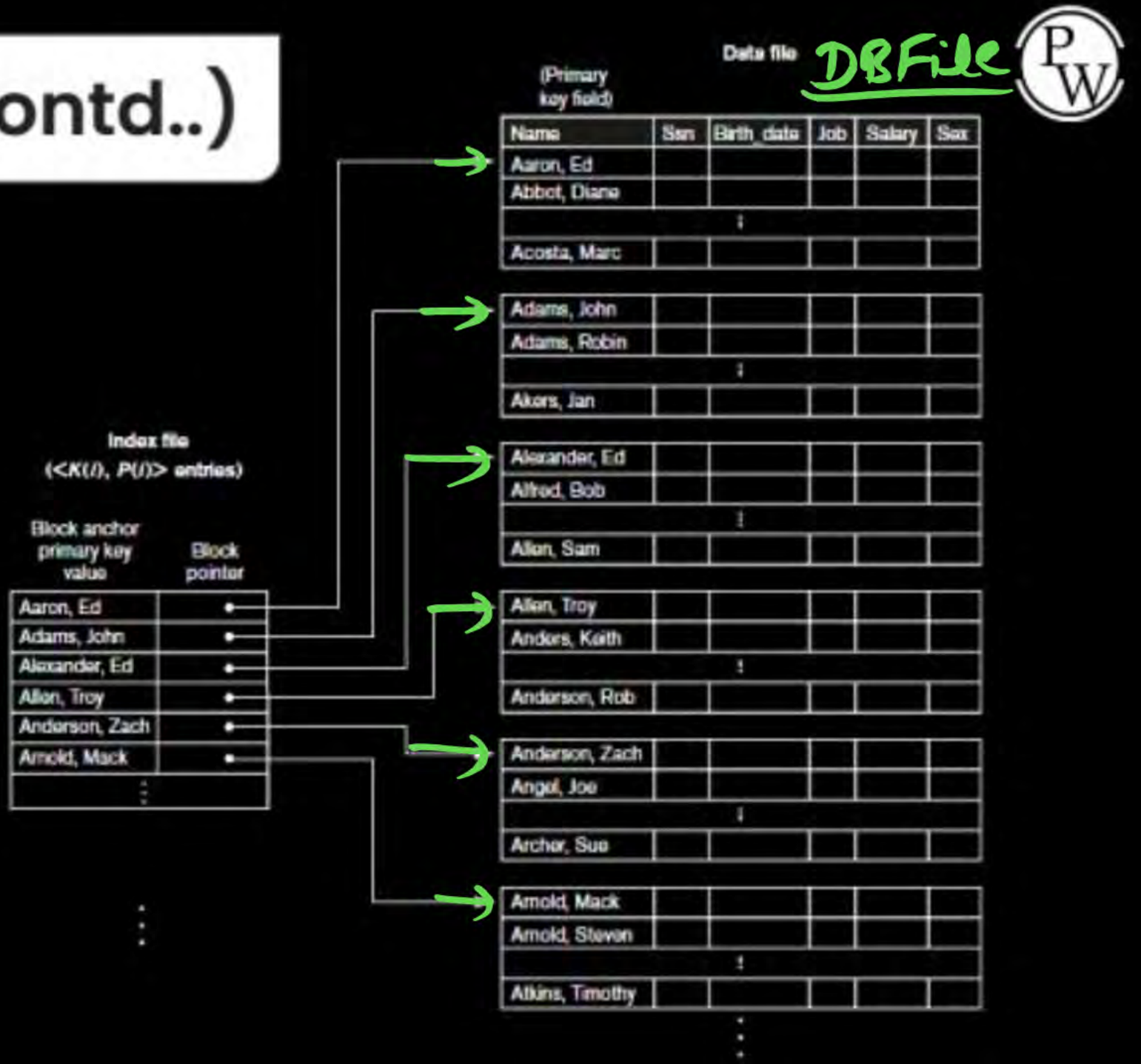


Primary Indexes

- Ordered file with two fields
  - Primary key, K(i)
  - Pointer to a disk block, P(i)
- One index entry in the index file for each block in the data file
- Indexes may be dense or sparse
  - Dense index has an index entry for every search key, value in the data file
  - Sparse index has entries for only some search values

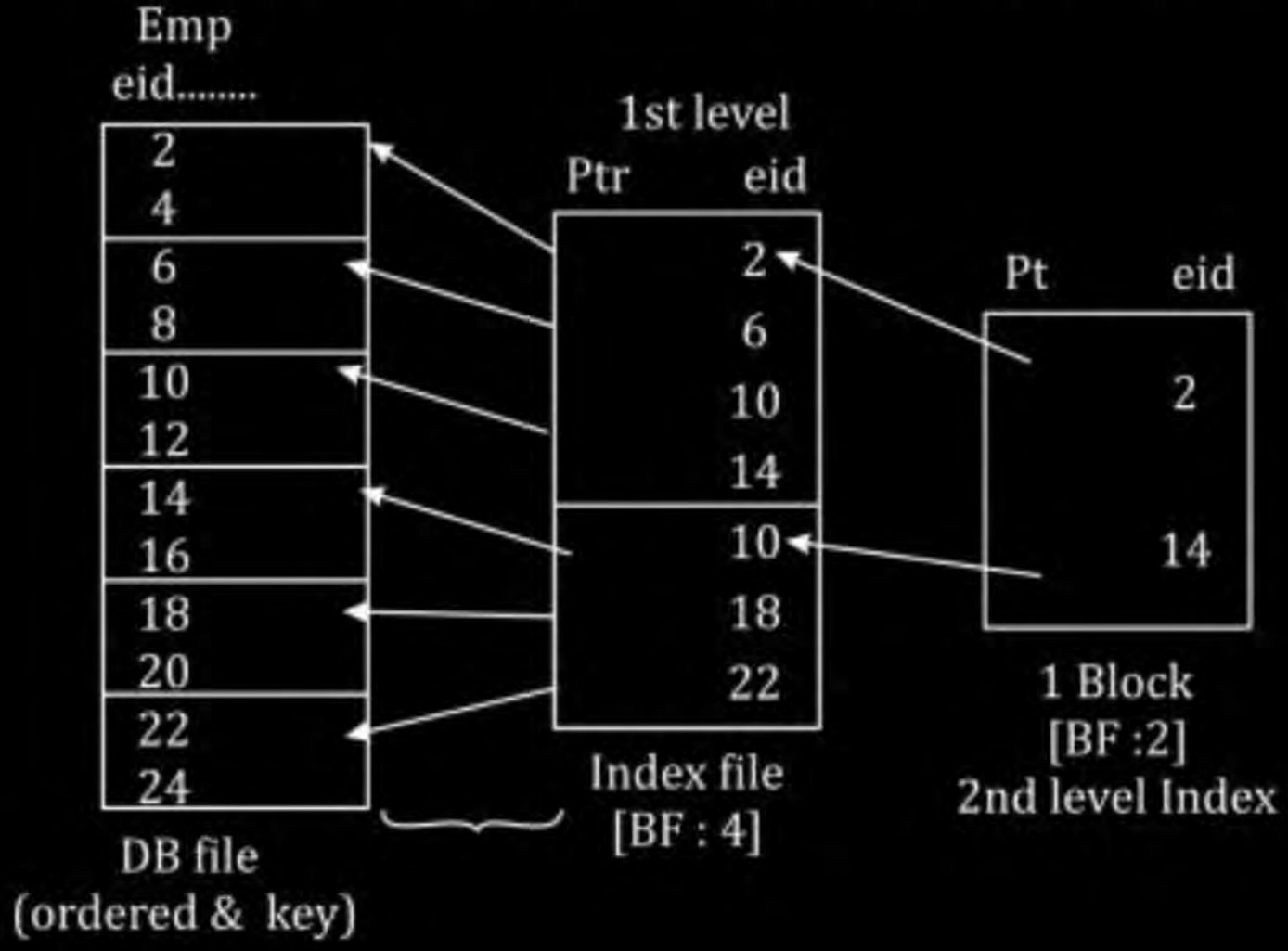
Primary Index (Contd..)

Primary index on the ordering key field of the file shown in Figure



(I) Primary Indexing:

Search key: Ordered field and key of the file.



⇒ Access cost to access record with PI with multilevel Index:

(K + 1) blocks

- Primary Index can be Dense or sparse

[sparse PI can be preferred]

- For any database relation at most one PI is possible

[because of Index over ordered field]

Q.1 Suppose that we have Ordered file of 30,000 records, stored on a disk with Block Size 1024 Byte, file records are of fixed length & unspanned of size 100 Byte (Record size) and suppose that we Have created a primary index on the key field of the file of size 9 Byte and Block pointer of size 6 Byte then find the average number of Block Access to search for a record using with and Without Index ?

#DB Blocks = 3000.

ORDERED File: To Access a Record Avg #Block Access $= \lceil \log_2 3000 \rceil = 12$

B_{fi} (Block factor of Index File) = 68 Index entries per Record.

PT: SPARSE (Total #Index entries = #DB Blocks [3000]

$$\#\text{Index Block } (B_i) = \left\lceil \frac{3000}{68} \right\rceil = 45 \text{ Index Block}$$

To Access Index Avg # Block Access $= \lceil \log_2 B_i \rceil \Rightarrow \lceil \log_2 45 \rceil$

$\Rightarrow$ 6 Avg

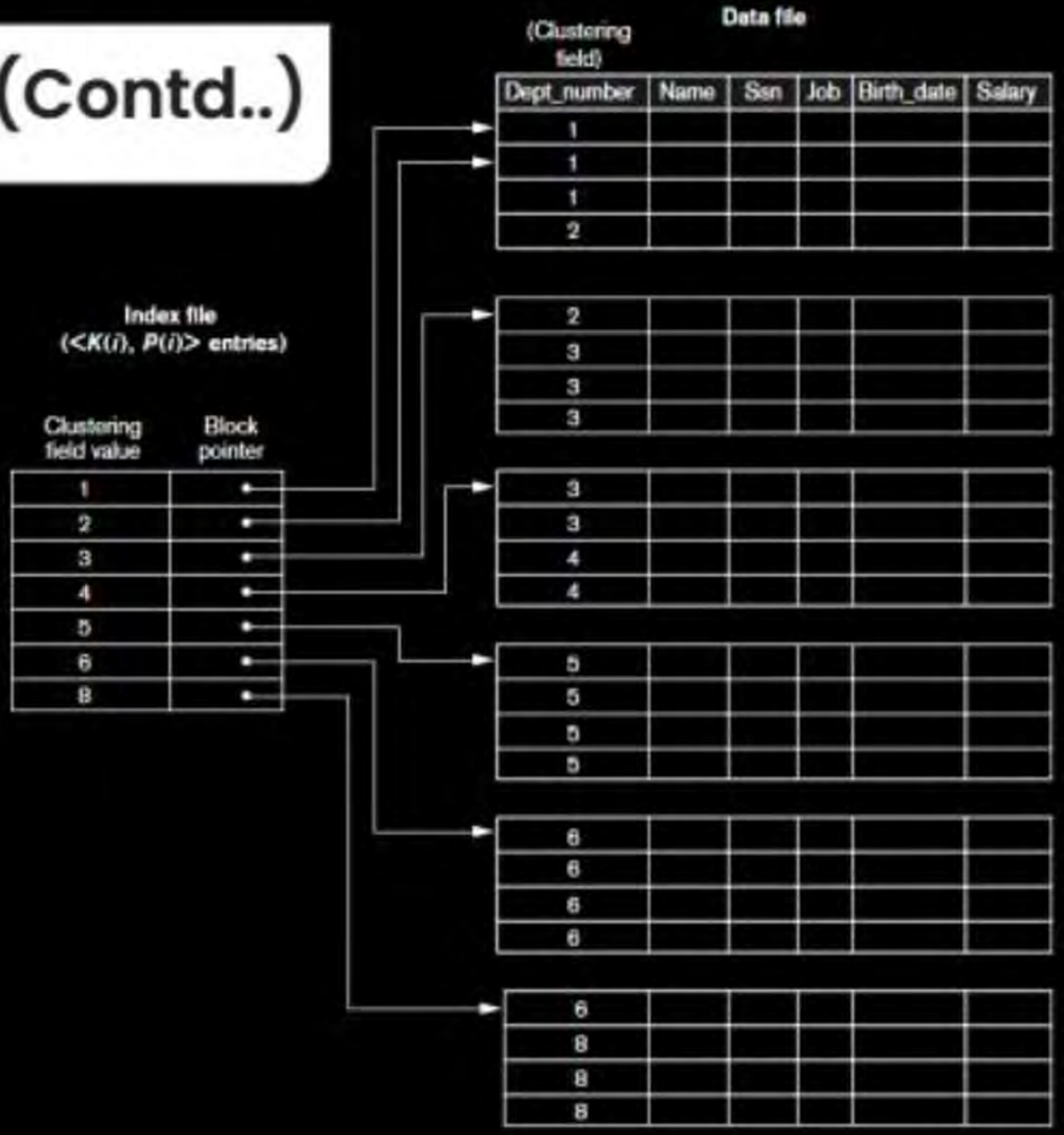
To Access a Record using Index Avg # Block Access $= \log_2 B_i + 1 \Rightarrow 6+1 = 7$ Avg

Clustering Indexes

- Clustering field
  - File records are physically ordered on a nonkey field without a distinct value for each record
- Ordered file with two fields
  - Same type as clustering field
  - Disk block pointer

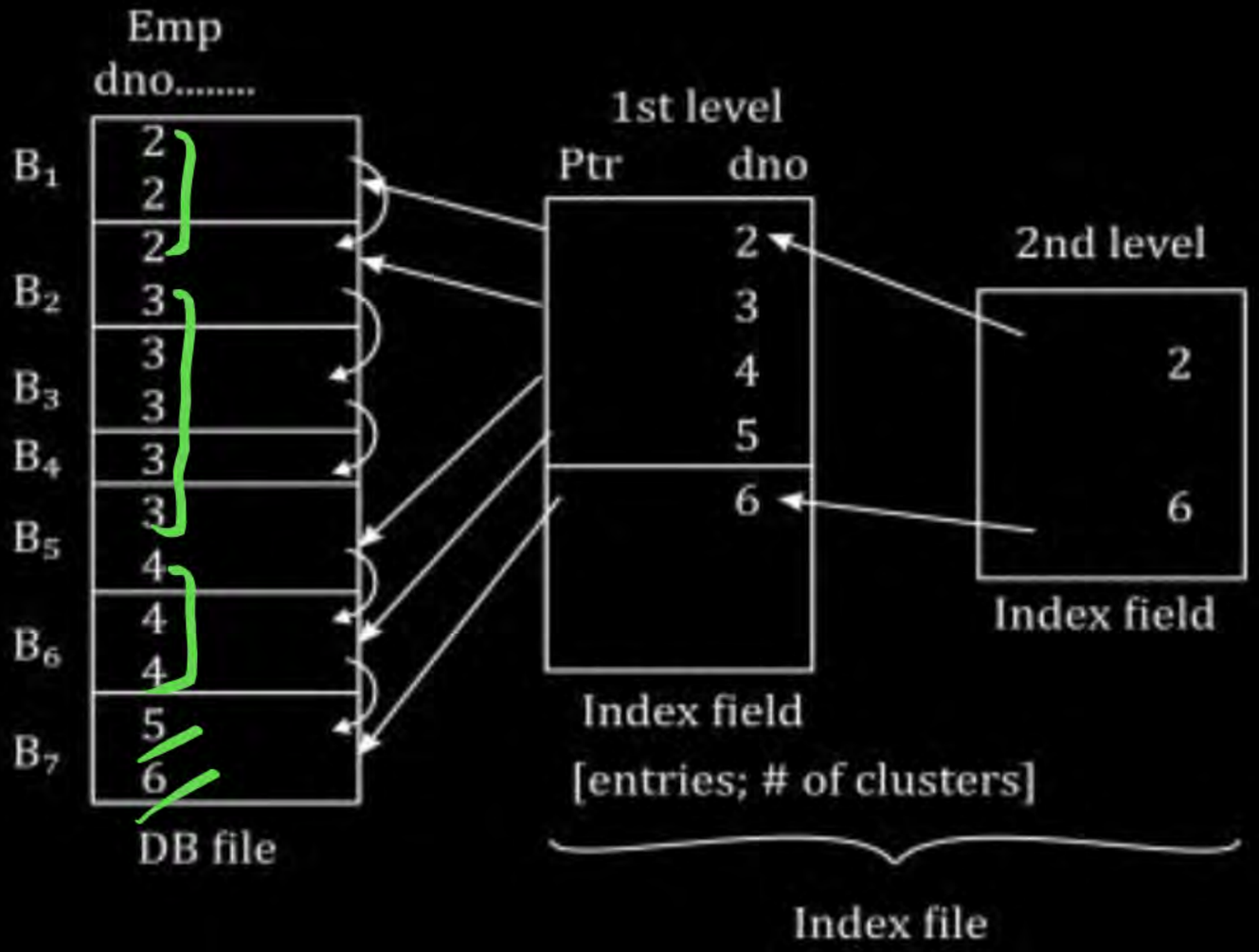
Clustering Indexes (Contd..)

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



Clustering Index:

Search key: Ordered field & Non- key.



- Clustering Index mostly sparse Index [Dense CI also possible If each cluster with one record]
- At most one CI is possible for any Database relation [ordering required]
- For any DB relation can build either PI or CI but not both.

PI ✓ CI ✗

CI ✓ PI ✗

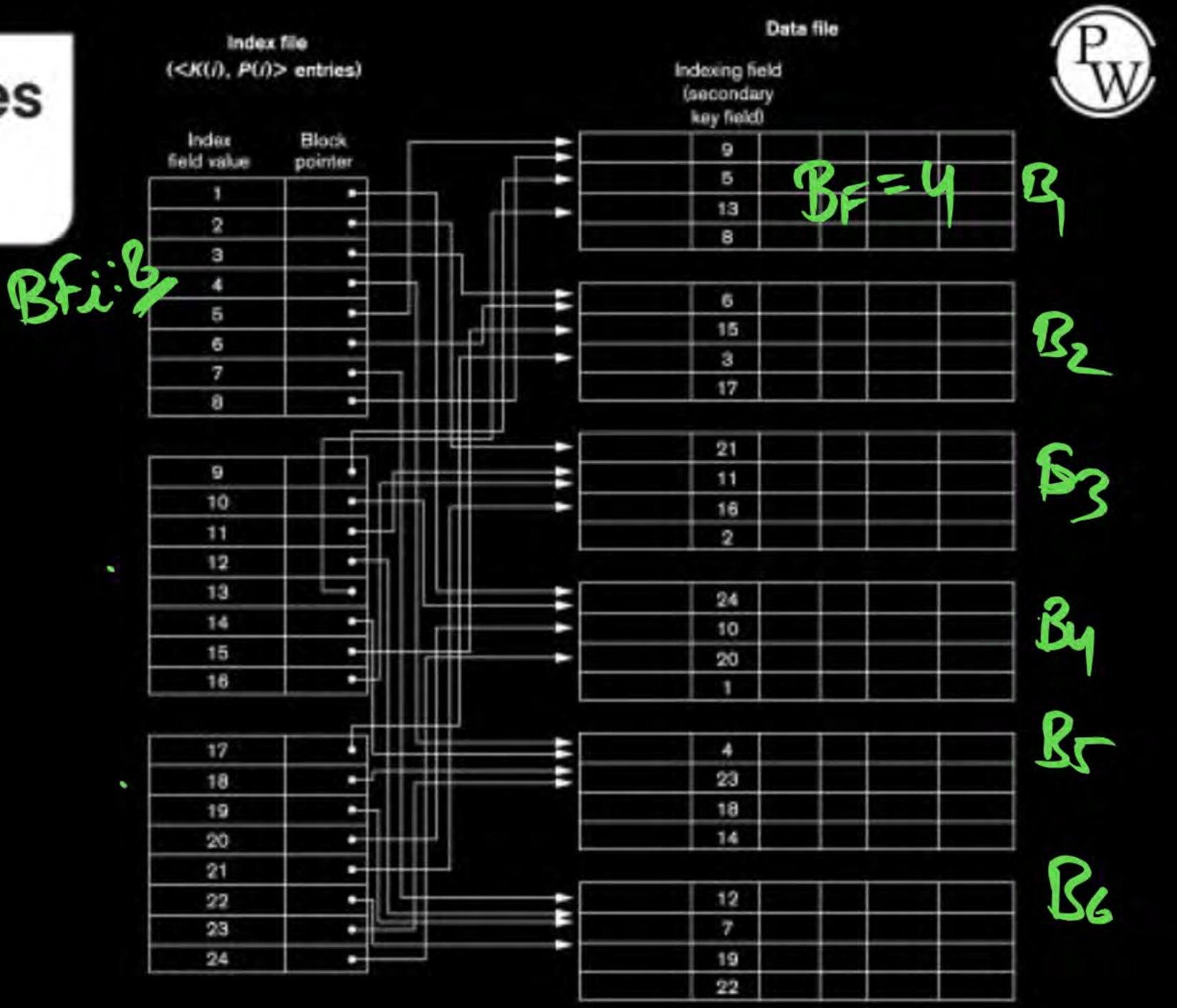
PI ✓ SI ✓

CI ✓ SI ✓

Secondary Indexes

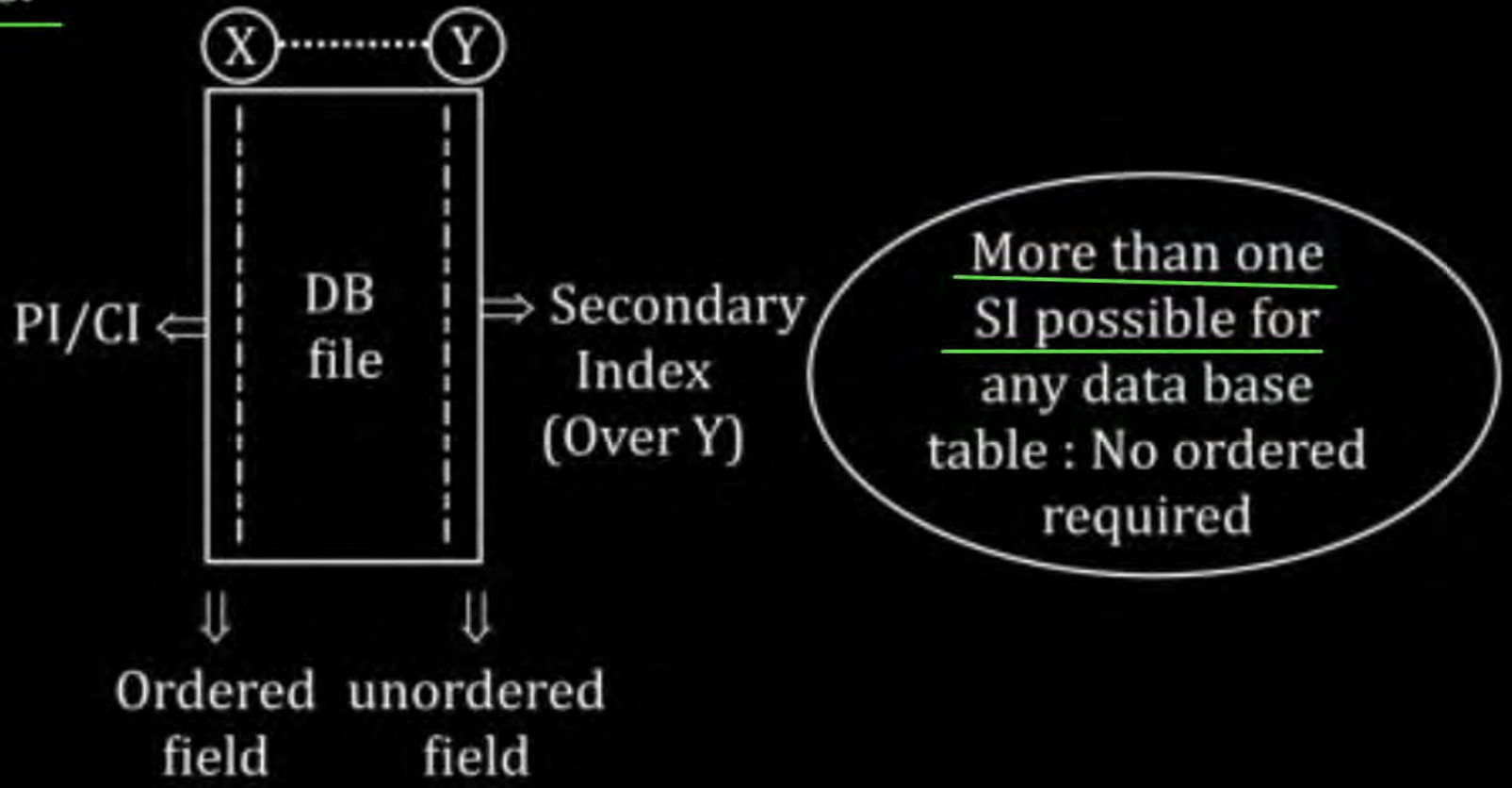
- Secondary Index
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some non-ordering field of the data file that is an indexing field.
    - The second field is either a block pointer or a record pointer.
    - There can be many secondary indexes (and hence, indexing fields) for the same file.
- Includes one entry for each record in the data file; hence, it is a dense index

Secondary Indexes (Contd..)

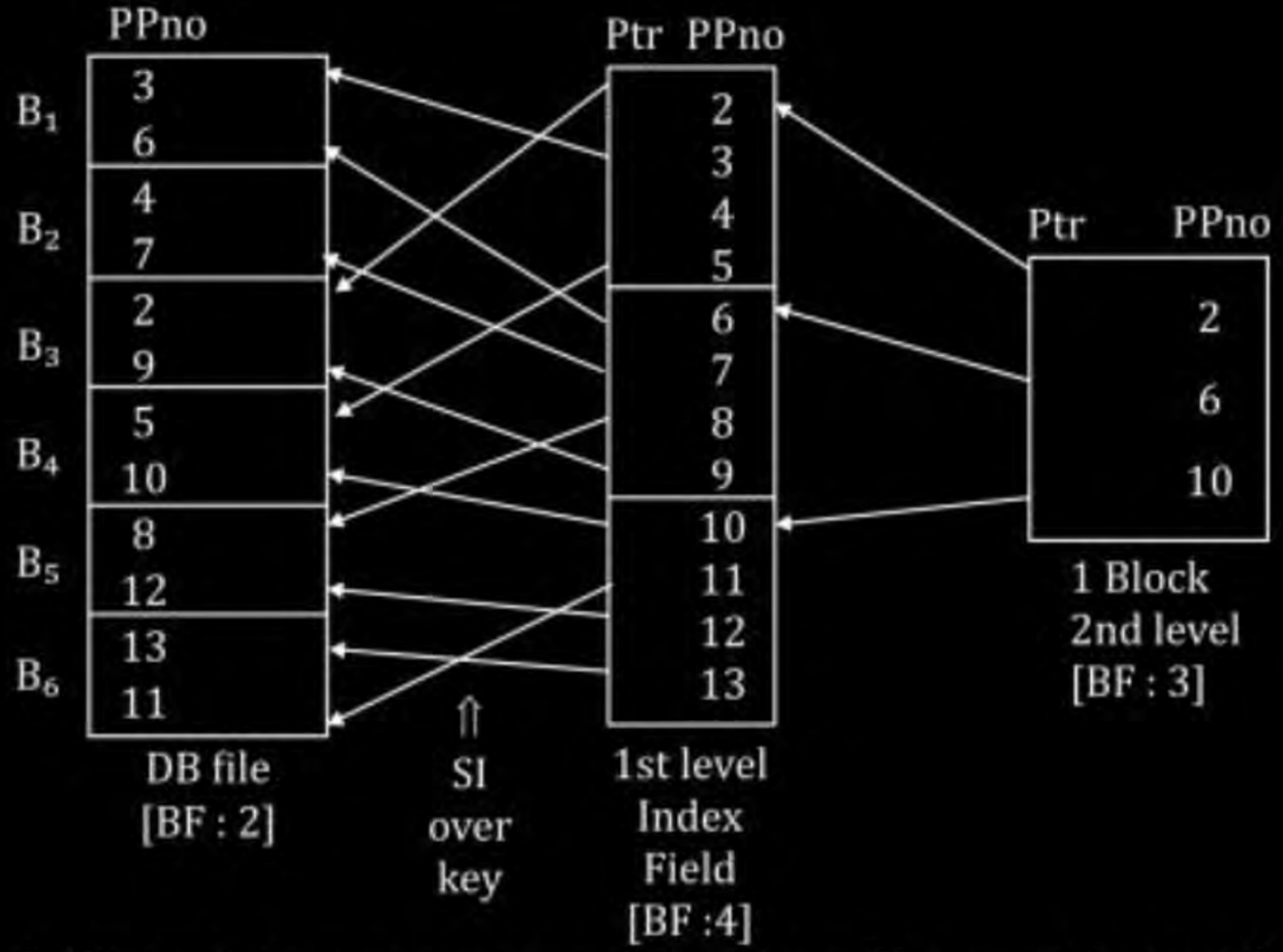


Secondary Index:

- Search key: Unordered field & Key/Non- key.
- Secondary possible way to access data using index even PI/CI indexes already exists.

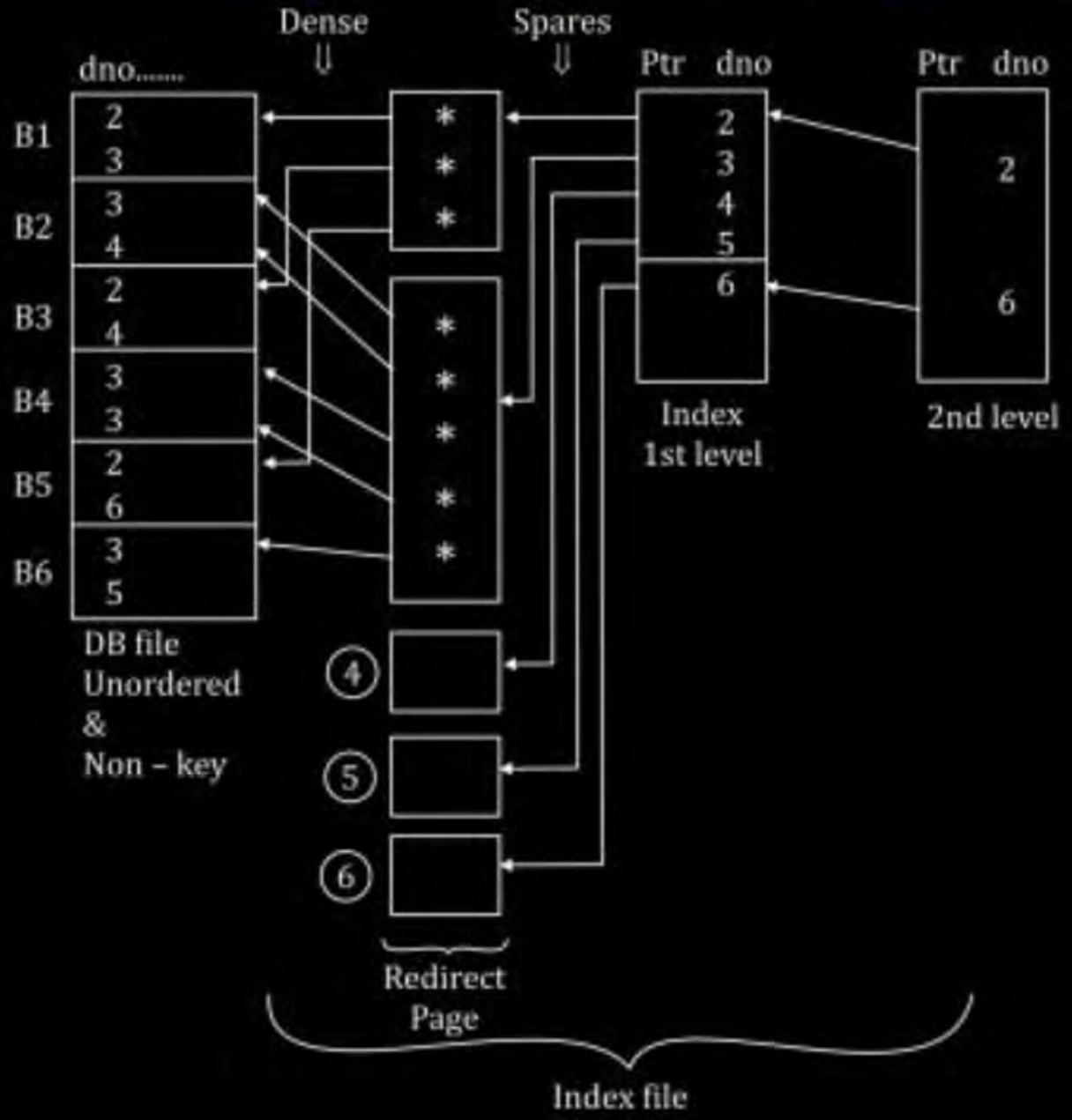


- Secondary Index (over key):



- I/O cost to access record using SI over key with MLI is (K + 1) blocks.

- ## Secondary Index (over Non-key):



- I/O cost to access record of some non-key using SI over non-key with MLI: {k+ # of blocks of DB equal to # of pointers in given redirect page}

Q.2 Consider a secondary Index on the key field of the file of question number 1, then find the Average number of Block Access to Access a record using with & without Index?

Number of records = 30000 Block size = 1024 B
record size = 1000 B
Key = 9B Bp = 6 Byte
Unspanned & Unordered.

Q.1 Suppose that we have ~~Ordered~~ Unordered file of 30,000 records, stored on a disk with Block Size 1024 Byte, file records are of fixed length & unspanned of size 100 Byte (Record size) and suppose that we Have created a ~~primary index~~ SI on the key field of the file of size 9 Byte and Block pointer of size 6 Byte then find the average number of Block Access to search for a record using with and Without Index ?

#DB Blocks = 3000.

ORDERED File : To Access a Record Avg #Block Access = $\frac{B}{2} = \frac{3000}{2} = 1500$

B_{fi} (Block factor of Index File) = GB Index entries per Record.

SI : Dense : (Total #Index entries = #DB Records = 30,000.

Total # Index Block = $\lceil \frac{30000}{68} \rceil$ = 442 Index Block

⇓

Now we Apply (ISAM) Multi Level Indexing.

⇓

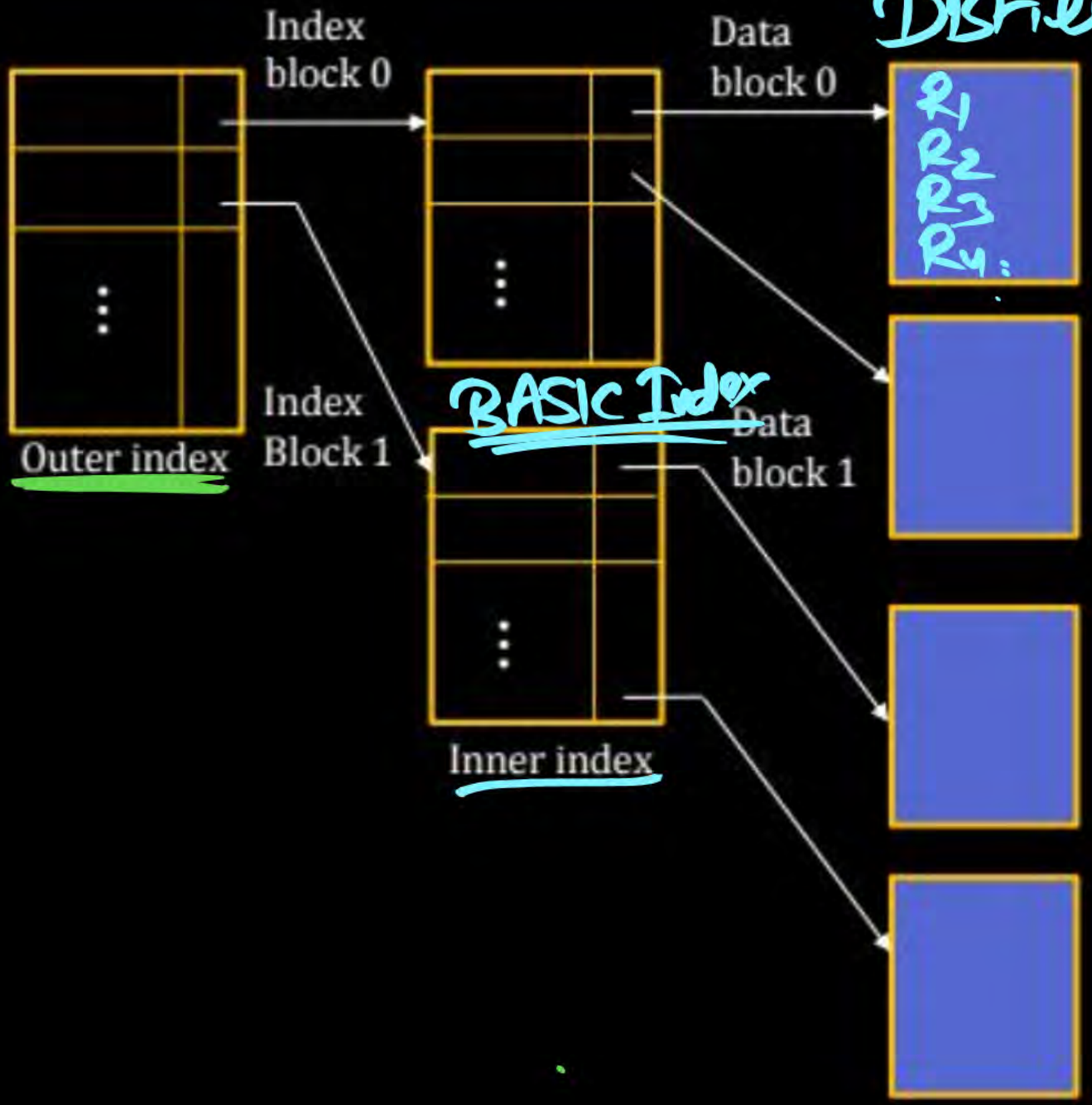
Now we Create SPARSE Index on Basic Index file & Repeat this Process untill all Index entries Fit into One Single Block.

Multilevel Index

(ISAM) Index Sequential Access Method.
(ORDERED File + Multilevel Primary Index.)

- If index does not fit in memory, access becomes expensive.
- Solution: treat index kept on disk as a sequential file and construct a sparse index on it.
  - outer index - a sparse index of the basic index
  - inner index - the basic 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.
- Indices at all levels must be updated on insertion or deletion from the file.

Multilevel Index (Contd..)



Multilevel Indexes

- Designed to greatly reduce remaining search space as search is conducted
- Index file
  - Considered first (or base level) of a multilevel index
- Second level
  - Primary index to the first level
- Third level
  - Primary index to the second level

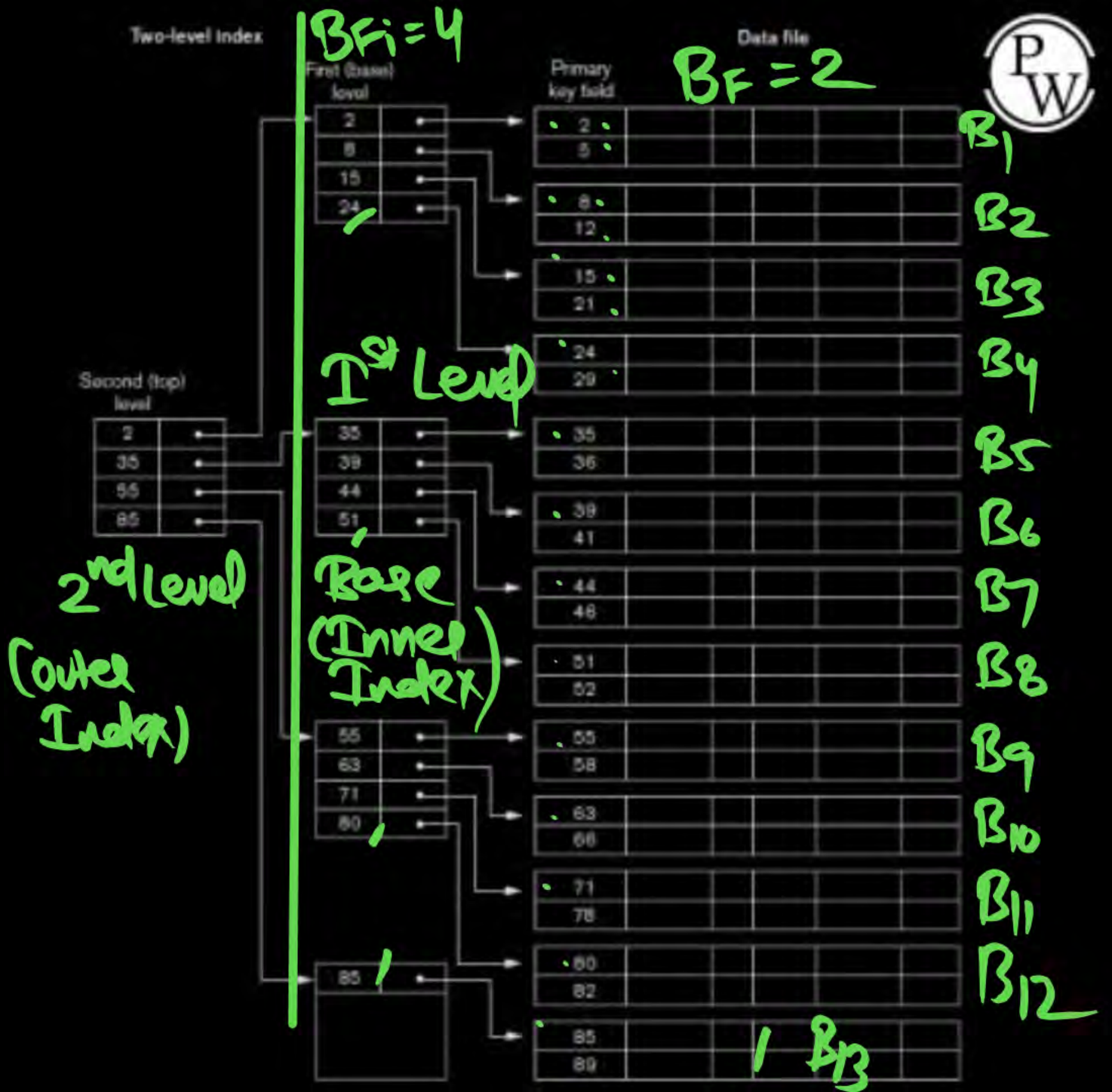
(Base Index file) I^st Level Index file ⇒ P.I (or) S.I

2nd Level : SPARSE Index (P.I) on I^st Level Index File (BASE File)

3rd Level : P.I (sparse Index) on II^nd Level Index file.

Untill Fit into 1 Block.

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



NOTE: We can Repeat the above process until index entries fit into One Block.

NOTE: If there are n level in multilevel index then the number of Block Access to search for a record = n + 1

(at each level One Index Block + 1 Data Block)

Q.3 Find the average number of block access required to search for a record if multilevel Index is created on the Data file of Question 2.

Block factor of Index file = 68 Index entries per Block.

1st Level Total #Index Block = 442.

2nd Level $= \lceil \frac{442}{68} \rceil = 7$ So Total 3 Level

3rd Level $= \lceil \frac{7}{68} \rceil = 1$

Total # Index Block = $\lceil \frac{30000}{68} \rceil$ = 442 Index Block

⇓

Now we Apply (ISAM) Multi Level Indexing.

⇓

Now we Create SPARSE Index on Basic Index file & Repeat this Process untill all Index entries Fit into One Single Block.

Q.3 Find the average number of block access required to search for a record if multilevel Index is created on the Data file of Question 2.

Block factor of Index file = 68 Index entries per Block

Ist Level: Total number of Index Block = 442 Index Block

IInd Level: number of Index Records (entries) = 442 (SPARSE number of Ist level block) & Block factor = 68 Index entries for block

$$\text{Total number Index Block} = \left\lceil \frac{442}{68} \right\rceil = 7 \text{ Index Block}$$

IIIrd Level: Number of Index Record = 7 (number of 2nd level block)

$$\text{Total number of index Block} = \left\lceil \frac{7}{68} \right\rceil = 1$$

Average Number of block Access = 1 + 1 + 1 + 1 = 4

Q.1 A clustering index is defined on the fields which are of type

[GATE-2008 : 1 Mark]

A. Non-key and ordering

B. Non-key and non-ordering

C. key and ordering

D. key and non-ordering

Q.2 Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is orderd 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 block pointer is 10bytes. If the secondary index is built on the key field of the file, and a multilevel index scheme is used to store the secondary index, the number of first-level and second-level block in the multilevel index are respectively [GATE-2008 : 2 Marks]

A 8 and 0

B 128 and 6

C 256 and 4

D 512 and 5

#Records = 16,384 [2^{14}] Record Size = 32B, Key = 6B, Bp = 10B, Block Size = 1024B, Secondary Index (Dense) & multilevel Index

---

One Index Record Size = 6 + 10 = 16 Byte.

Block factor of Index file [B_{Fi}] = $\left\lfloor \frac{1024B}{16B} \right\rfloor = \frac{2^{10}}{2^4} = 2^6$ = 64 Index entries per Block

Secondary Index (Dense): Total # Index entries = 16,384 [2^{14}] (# DB Records)

1st Level Total # Index Block = $\frac{2^{14}(16,384)}{2^6\,(64)} = 2^8 = 256$ Index Block

∴

2nd Level

#Index entries = 256 (# 1st Level Blocks)

Bfi = 64 Index entries Per Block

$$\text{Total \# Index Blocks} = \left\lceil \frac{256}{64} \right\rceil = 4$$

(256, 4) Ans

3rd Level

Total # Index entries = 4 & Bfi = 64 Index entries Per Block.

$$\text{Total \# Index Block} = \left\lceil \frac{4}{64} \right\rceil = 1$$

So Stop

Total 3 Level Required.

∴

Total 3 Level Indexing Required

To Access a Record Using Multilevel Index Avg # Block Access $= n + 1$

$= 3 + 1$

$= (4)$ Avg

$\Rightarrow 1 + 1 + 1 + 1$

1 → 3rd level

1 → 2nd

1 → 1st

1 → DATA BLOCK

Problem With Static Multi Level Indexing

Problem with Static Multi Level Indexing.

In Previous Q.1

University DB : 30,000 Students → 3000 DATA Block

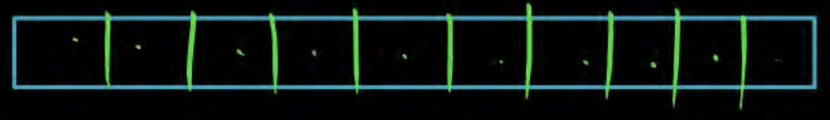
BFDB = 10 Record Per Block

Array : Static

int a[10]

↳ (i) 1000 Student (Delete) ⇒ OUT. ⇒ 29000 ⇒ 2900 Blocks

↳ (ii) 2000 Student (Insert) ⇒ New admission = 32000 ⇒ 3200 DATA Block



costly & Very Time Consuming

So Dynamic Multi Level Indexing

Cylinder

Q1

Role of Block size : if Block Size is very - very small

Sol^n Capacity Wasted [1 Completed Cylinder for only Few Byte

Platter
↳ Surface
↳ Track
↳ Sector 'DATA'

'OS & COA'

'Block' ⇒ GROUP of Sector.

Cylinder

Role of Block size : if Block Size is very-very Large?

Soln Next cylinder (Unnecessary Seek time + R.L Increase.)

Platter
↳ Surface
↳ Track
↳ Sector 'DATA'

Block ⇒ GROUP of Sector.

OS & COA'

Dynamic Multi Level Indexing.

- B Tree [Balanced Tree].
- B+ Tree

B Tree Definition

DSA

ORDER : P

Min keys = $\lceil P/2 \rceil - 1$

Max keys = $P - 1$

eg ORDER : 5

Min keys = $\lceil \frac{5}{2} \rceil - 1 \Rightarrow 3 - 1 = 2$ Ans

$\lceil 2.5 \rceil - 1$

Max key = $P - 1 \Rightarrow 5 - 1 = 4$ Ans

B Tree Definition

DSA

ORDER : P.

Min keys = $\lceil P/2 \rceil - 1$

Max keys = P − 1

eg ORDER : 6

Min keys = $\lceil 6/2 \rceil - 1 \Rightarrow 3 - 1$

= 2 Ans

Max key = P − 1

= 6 − 1

= 5 Ans

ORDER : '5'

DSA

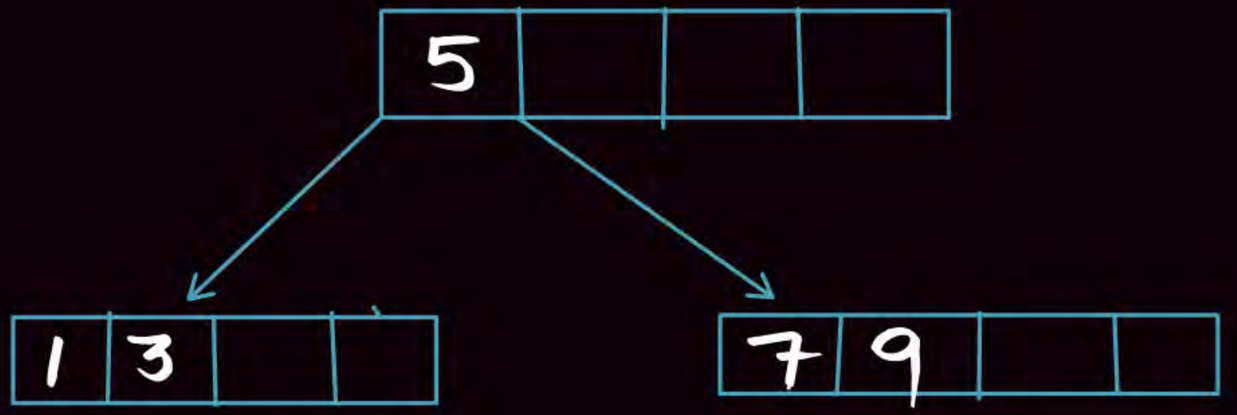
1, 5, 3, 7, 9, 11, 13, 16.

Min key = $\lceil \frac{5}{2} \rceil - 1 \Rightarrow 2$

Maximum keys = 5 - 1 = 4

← '9'

Violate B Tree Definition



ORDER : '5'

DBMS 1, 5, 3, 7, 9, 11, 13, 16.

Min keys = $\lceil \frac{5}{2} \rceil - 1 \Rightarrow (2)$

Maximum keys = 5 − 1 = 4

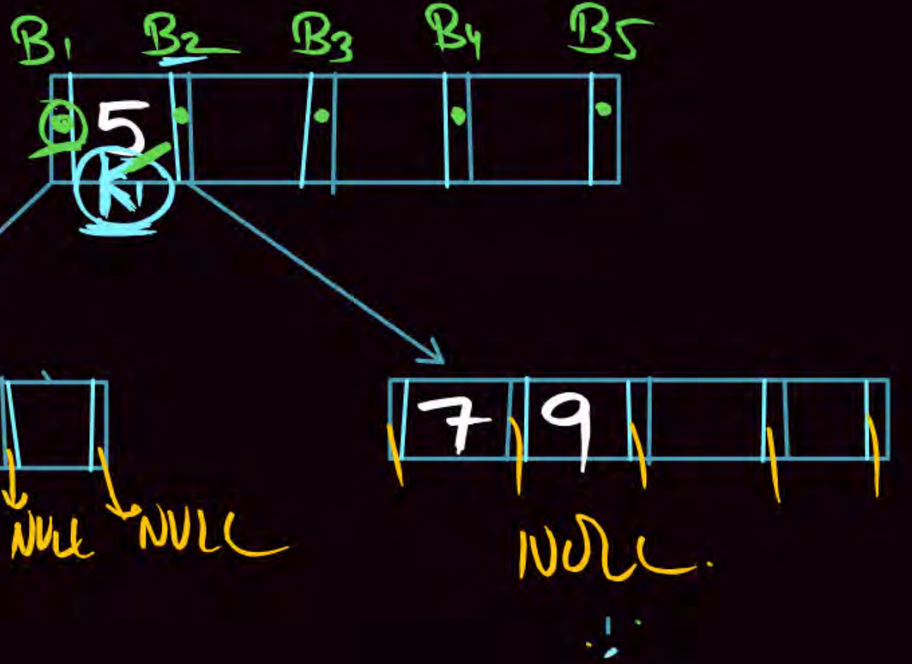
Violate B Tree Definition

ORDER : 5

$B_p = 5$

keys = 4

$R_p = 4$



ORDER : 5

Max keys = (4)

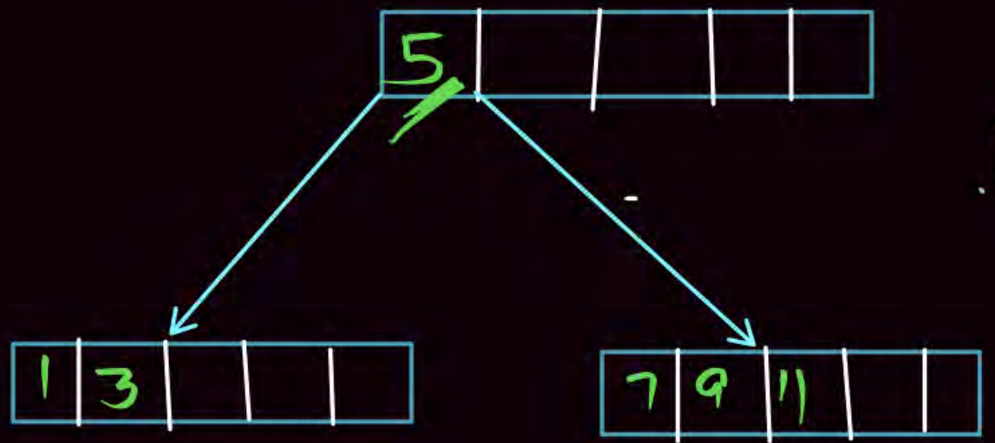
ORDER : 6

$\text{Min key} = \lceil 6/2 \rceil - 1 = 3 - 1 = 2$

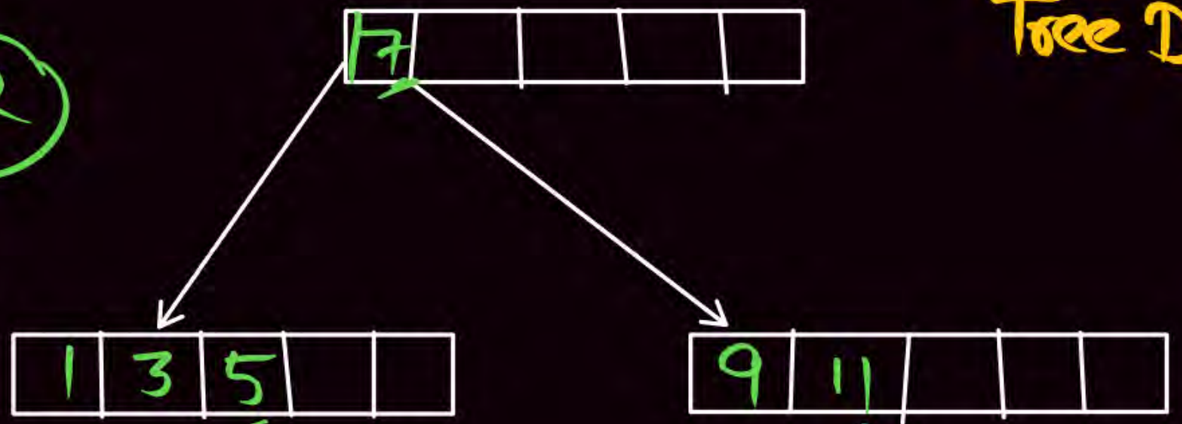
$\text{Maximum keys} = 6 - 1 = 5$

1, 5, 3, 7, 9, 11, 13, 16.

Violate B Tree Definition



OR



B Tree Definition :

ORDER : P ⇒ Maximum Number of Block Pointer is P.

ORDER : P

Bp : P

key = P - 1

Rp = P - 1

Bp : Block / Child / Tree / Index Pointer.

Rp : Read / Data / Record Pointer

K : Search key.

B Tree Definition

① ORDER : 2

$Bp = 2$

$key = 1$

$Rp = 1$

② ORDER : 3

$Bp = 3$

$key = 2$

$Rp = 2$

③ ORDER : 4

$Bp = 4$

$key = 3$

$Rp = 3$

④ ORDER : 5

$Bp = 5$

$keys = 4$

$Rp = 4$

⑤ ORDER : 6

$Bp = 6$

$key = 5$

$Rp = 5$

⑥ ORDER : P

$Bp = P$

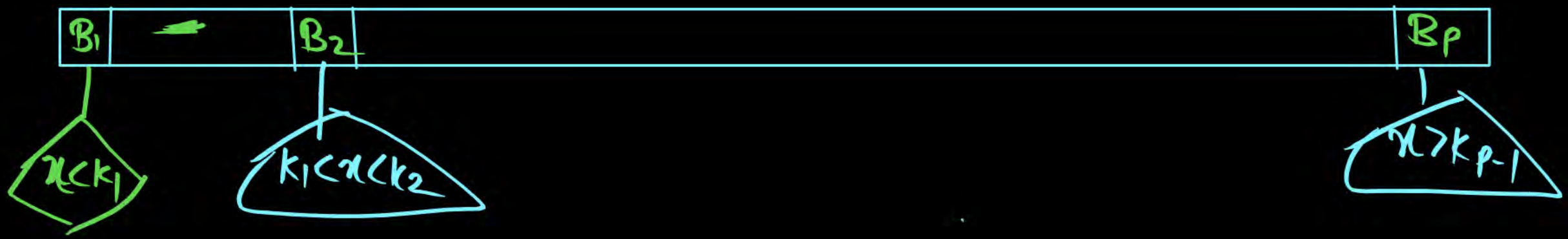
$key = P - 1$

$Rp = P - 1$

ORDER : P

Definition of B Tree

① structure of Internal Node



ORDER : P.

Definition of B Tree

② Structure of Leaf Node

B Tree Definition

ORDER : P.

Definition of B Tree

③ Every Internal Node Except the Root Node Contain at least(min) $\lceil P/2 \rceil$ Block Pointer (min keys $\lceil P/2 \rceil - 1$) and Maximum P Block Pointer (Maximum keys $P-1$).

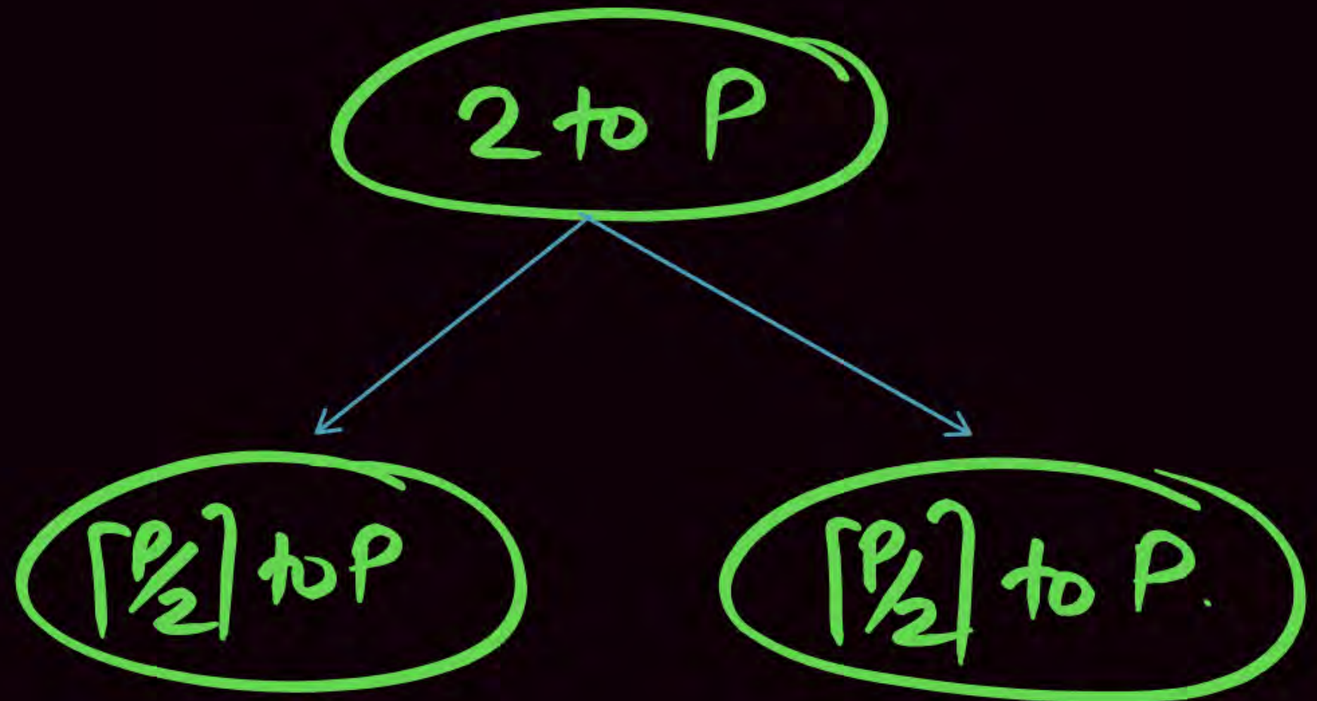
④ Root Can Contain atleast 2 Block Pointer (Min. 1 key in Root Node) & Maximum P Block Pointer (Max $P-1$ keys).

ORDER : P.

Definition of B Tree

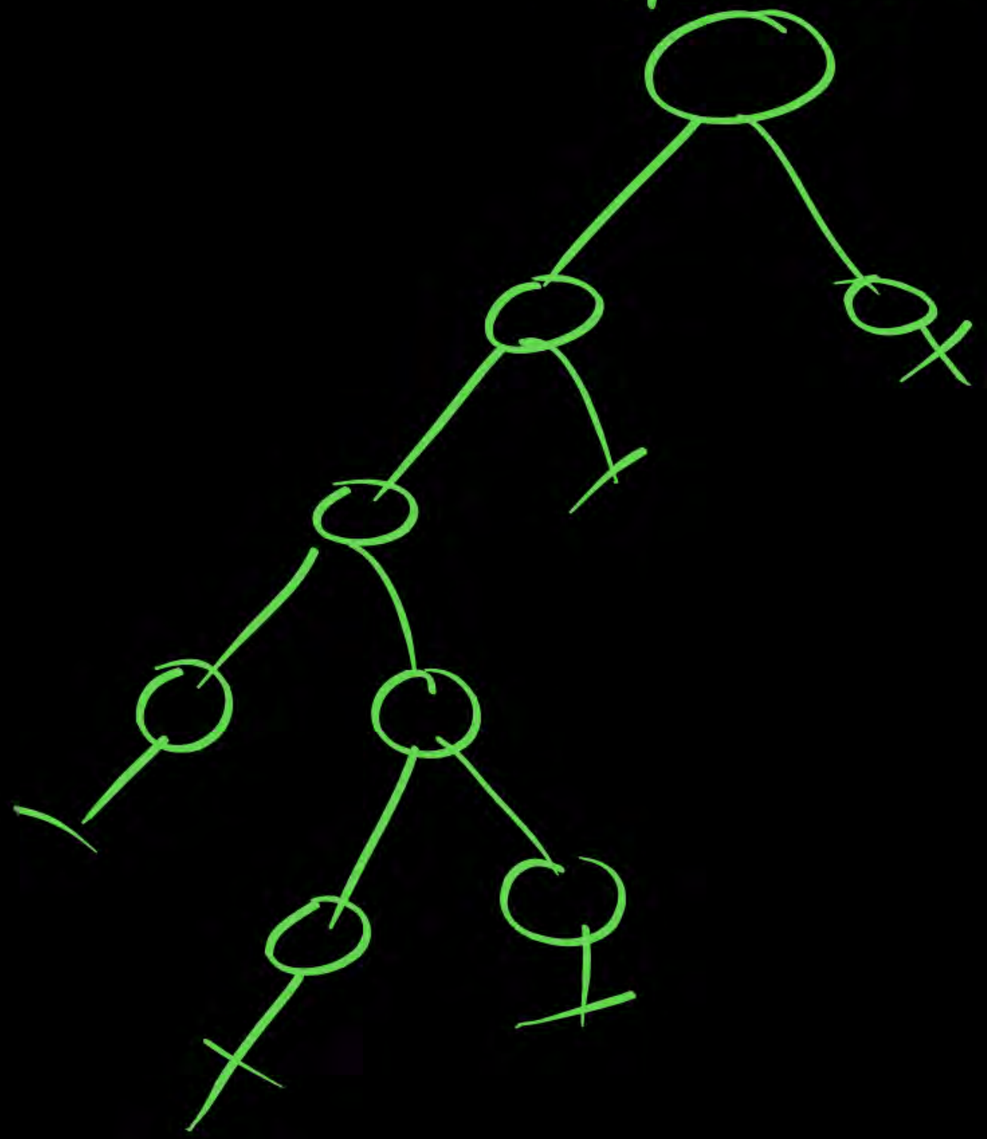
⑤ Keys within the Node should be in ascending order.

⑥ Every Leaf Node Should be at Same Level.

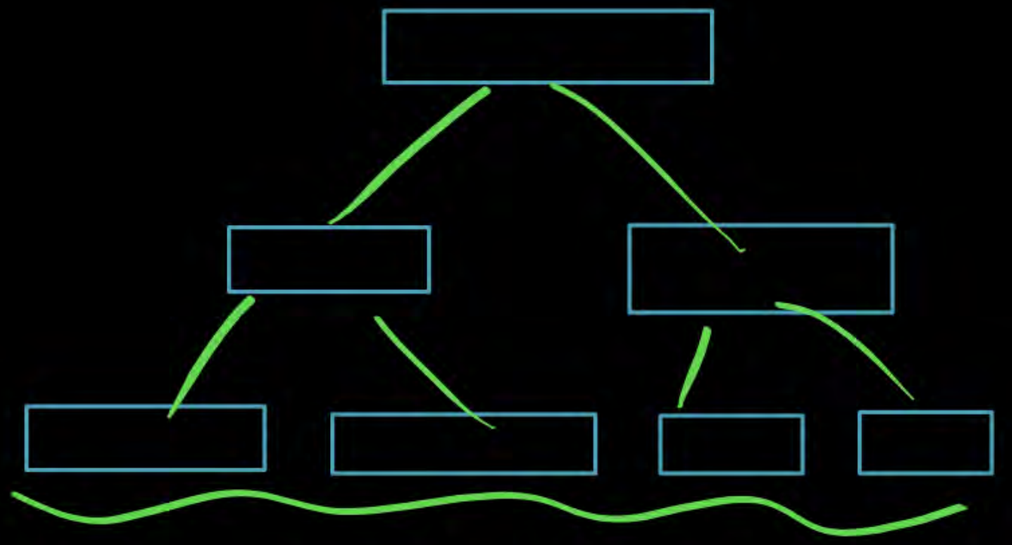

2 to P
⌈P/2⌉ to P
⌈P/2⌉ to P.

B Tree Definition

General Tree



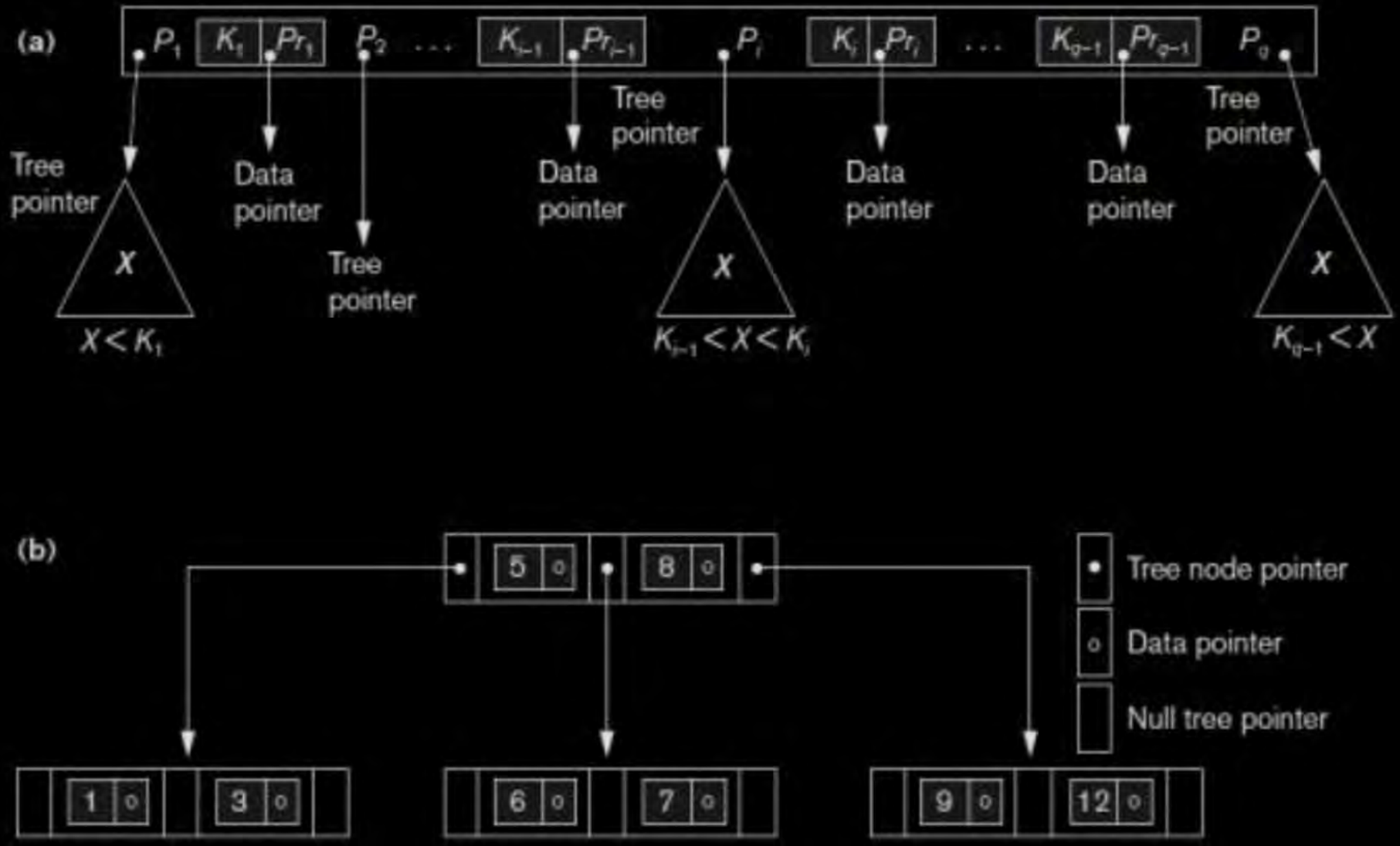
B & B+ Tree



- Leaf Node

B Tree Structure

B-tree structures (a) A node in a B-tree with q−1 search values (b) A B-tree of order p=3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6

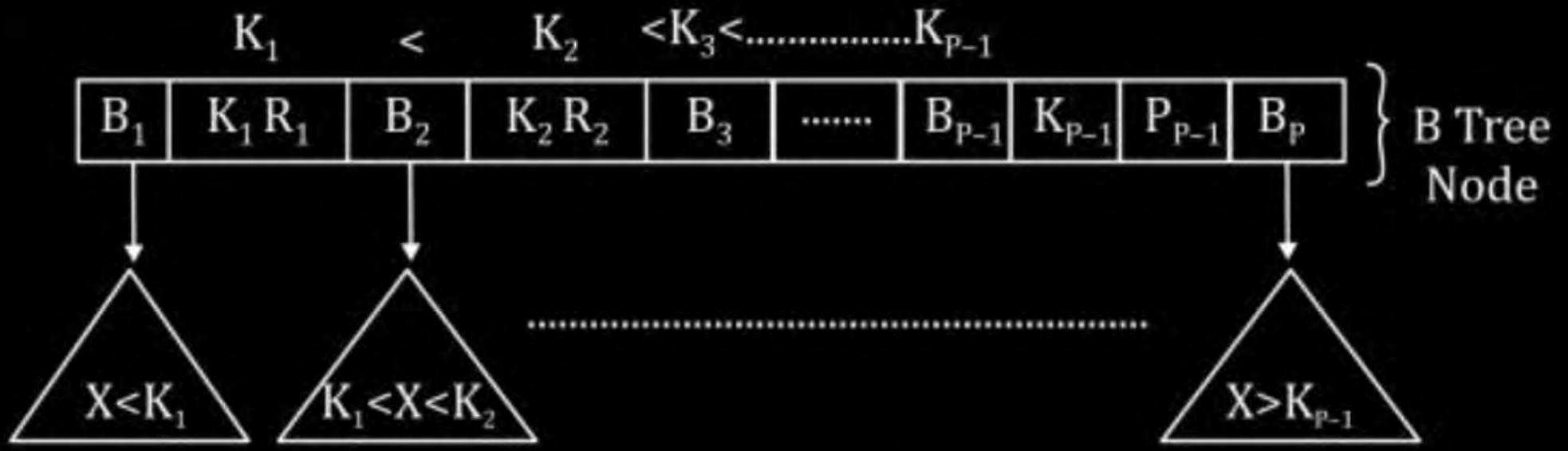


B Tree Definition

Order P: Max possible child pointers

[degree] can store in B Tree node.

(1) Node structure:

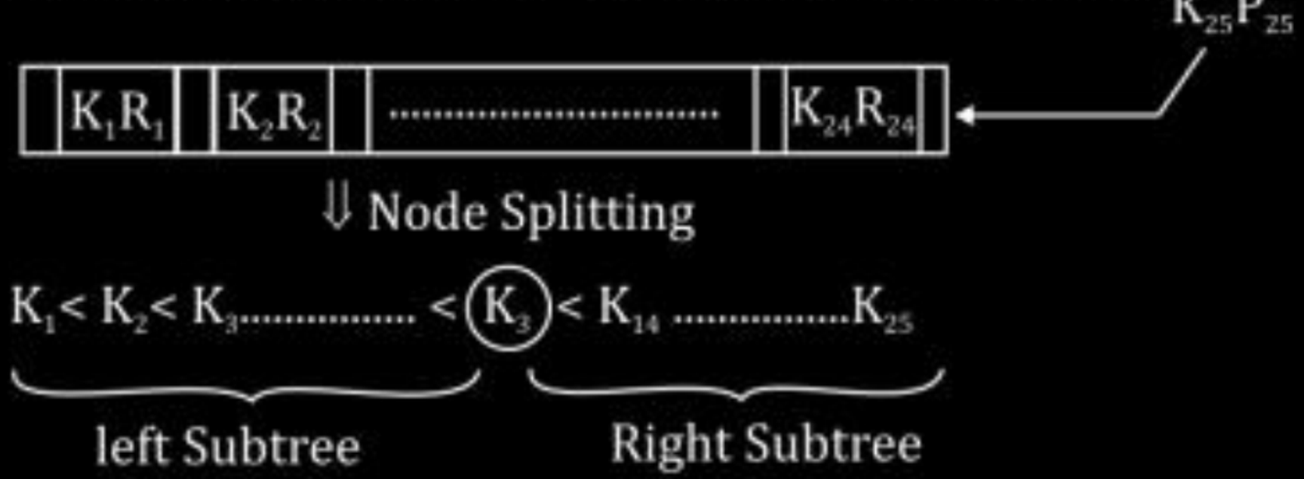


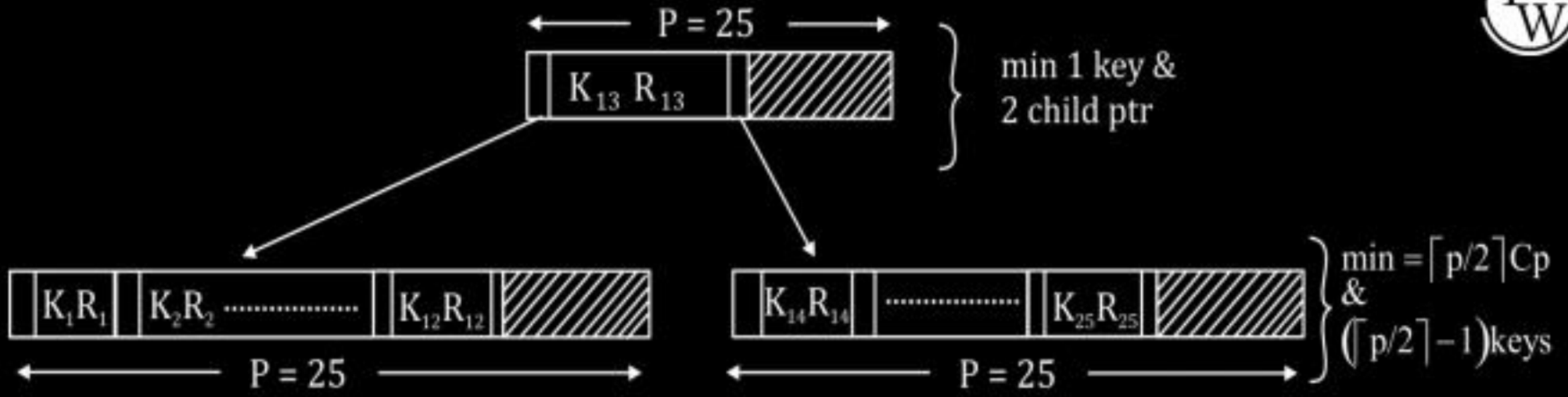
<P.Cp, (p-1)keys, (p-1)Rp>

(2) Every internal node except root must be at least $\lceil P/2 \rceil$ child pointer with $(\lceil P/2 \rceil - 1)$ key's and at most P children pointer with (P - 1) key's must.

(3) Root node can be at least 2 children with 1 key, at most P children pointer and (P - 1) key's.

(4) Every leaf node must be at same level and keys with in node should be sequence order (Increasing order) Order P: 25 [Max Cp in node]





except Root Every
index node
atleast 50% occupied
in B/B* Tree index

Block B tree node
[Free]

Q. Construct B Tree with order P: 4[max Bp/cp (Block/child pointer) per node] and following sequence of key's _____

5, 10, 15, 1, 23, 34, 13, 8.

Q.3 Consider a table T in a relational database with a key field K. A B-tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in B-tree index node. Assume that K is 10 bytes long; disk block size is 512 bytes; each data pointer P_D is 8 bytes long and each block pointer P_B is 5 bytes long. In order for each B-tree node to fit in a single disk block, the maximum value of p is [GATE-2004 : 2 Marks]

A 20

B 22

C 23

D 32

Advantage:

1) B Tree index best suitable for Random access of some record.

```
select *
FROM R
WHERE A = 24 ;
```

One record access

I/O cost: K + 1 blocks

$\approx [\log_p n] + 1 = \theta(\log_p n)$

Disadvantage:

1) B Tree index not best suitable for sequency access of range of records.

```
select *
FROM R
WHERE A ≥ 30 and A ≤ 85;
```

Range of record Access required

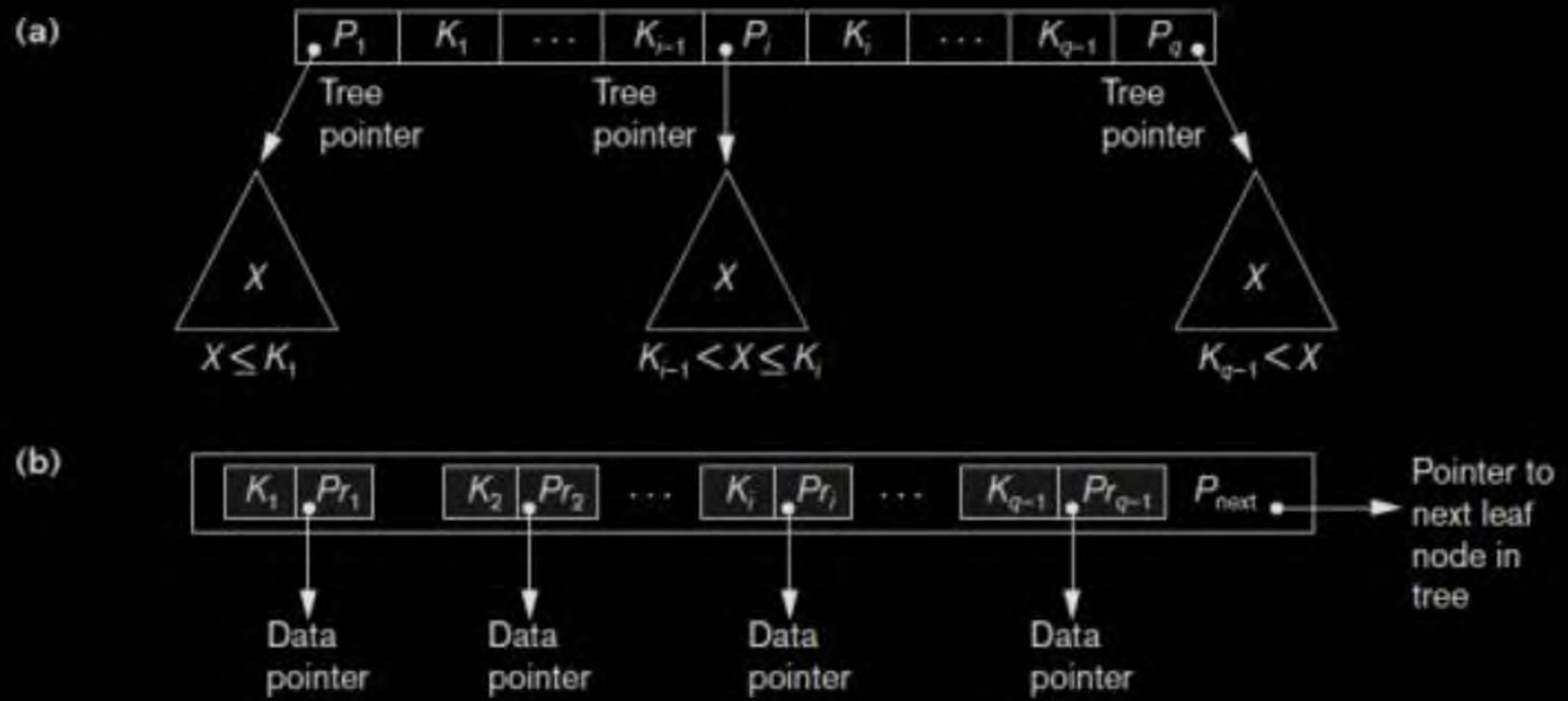
← X blocks of DB →

I/O cost: $x[\log_p n + 1]$ + cost of unsuccessful

More Access cost

[Unordered File DB]

B+ Tree



The nodes of a B+-tree (a) Internal node of a B+-tree with q−1 search values (b) Leaf node of a B+-tree with q−1 search values and q−1 data pointers

B+ Tree Definition

Order P: max possible pointers [degree] can store in B^+ Tree node.

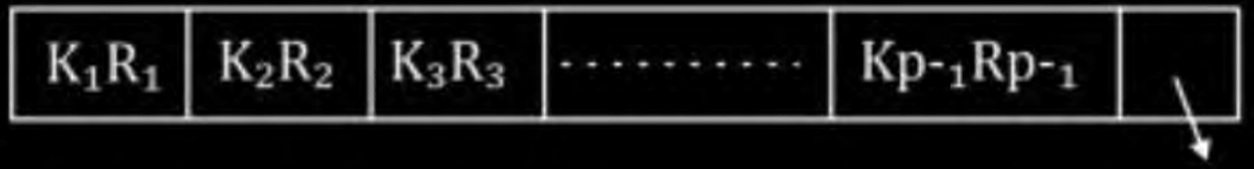
(1) Node Structure:

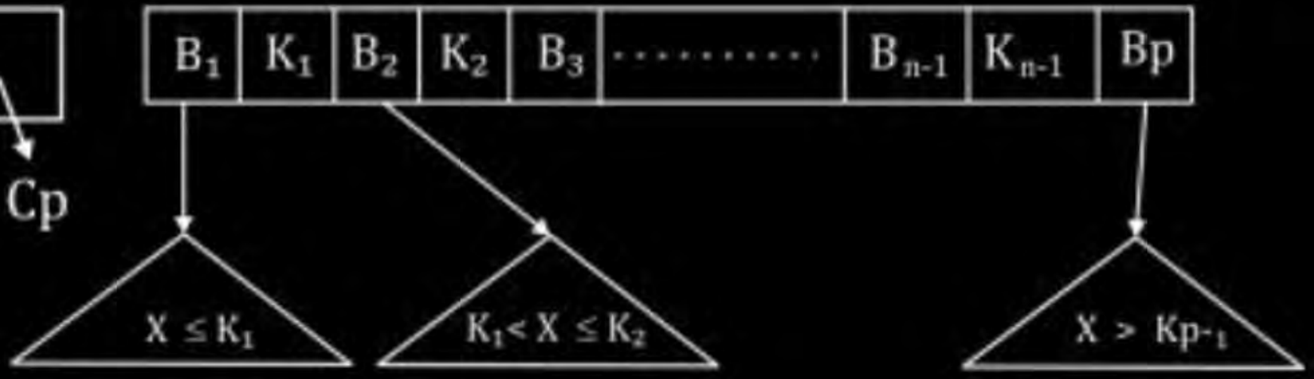
(a) Leaf Node

[set of (key, Rp) pair and only one block pointer pointed to next Leaf node]

(b) Internal Node

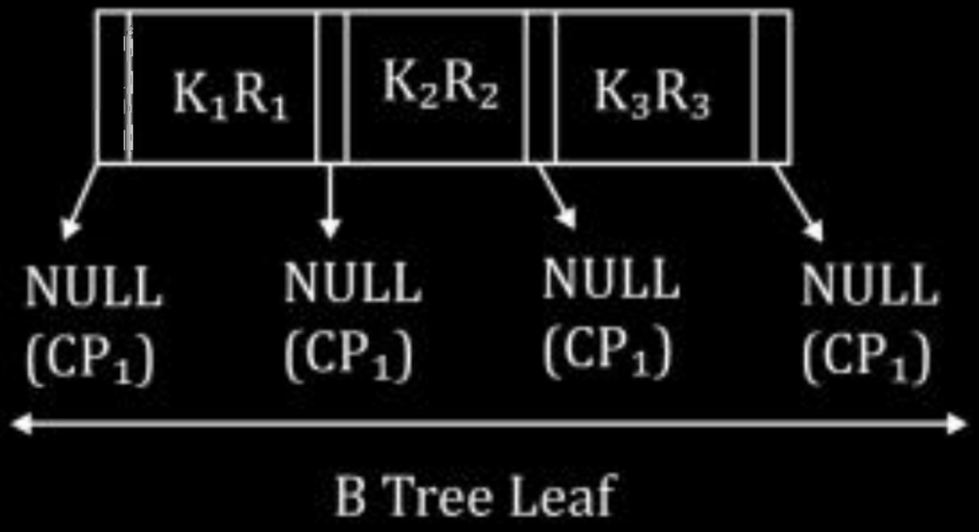
[key's and child pointer]
∴ No record pointer (Rp).

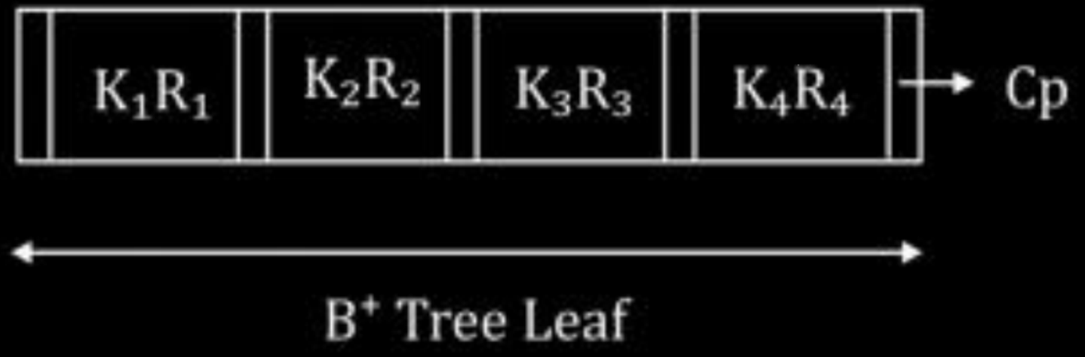




⇒ Left Biasing condition $k_1 < x \leq k_2$ .....

Right Biasing condition $k_1 \leq x < k_2$ ..... $x \geq k_{p-1}$





⇒ Remaining Balancing Condition same as B Tree

Advantage:

1) B^+ Tree index best suitable for sequence access of range of records.

   [Range Quires runs faster using B^+ Tree index]

   Select *

   FROM R

   WHERE A ≥ 30 and A ≤ 85;

   I/O cost: $[\log_p n + x + x]$

Q.4 The order of an internal node in a B^+tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

A 24

B 25

C 26

D 27

Q.5 Consider a B^+-tree in which the maximum number of keys in a node is 5 . What is the minimum number of keys in any non-root node?

A 1

B 2

C 3

D 4

THANK YOU!