COMPUTER SCIENCE

Database Management System

File Org. & Indexing

Lecture_2

Vijay Agarwal sir

TOPICS TO BE COVERED

01 File Structure

02 Indexing & its Type

Blocking factor

1. SPANNED ORG
2. Unspanned Org

ORDERED File $\Rightarrow \lceil \log_2 B \rceil$

Unordered (Heap) file $\Rightarrow$ Avg $= B/2$

Worst $= B$.

File Structures & Indexing

Database

Files

Blocks

Records

Fields

- Database is collection of files.
- Each file is a collection of Records.
- Each record is a sequence of fields.

- DB is divided into number of blocks.
- Each block is divided into records.
- Record can be stored in a blocks.

Record Blocking and Spanned Versus Unspanned Records

- Blocking factor
  - Average number of records per block for the file

  Spanned organization

Record Blocking and Spanned Versus Unspanned Records

- Blocking factor
  - Average number of records per block for the file.

  Unspanned organization

Note:

In spanned roganization No memory is waste but I/O cost is more (Block Access Increase).

But in unspanned organization memory is waste but input output cost is less compared to spanned organization.

Default organization is unspanned organization.

Note:

I/O Cost: Input Output cost means Number of Blocks transferred from secondary memory to Main memory in order to access some records.

Search Key: Attribute used to access the Data from DB

Organization of records in a file: (1) ORDERED file organization

(2) Unordered file organization

Files of Ordered Records (Sorted Files)

- Ordered (sequential) file
  - Records sorted by ordering field
    - Called ordering key if ordering field is a key field
- Advantages
- Reading records in order of ordering key value is extremely efficient
- Finding next record
- Binary search technique

NOTE:
To Access a record the average number of
Block Access = $\log_2 B$
(B: Data Blocks)

Files of Unordered Records (Heap Files)

- Heap (or pile) file
  - Records placed in file in order of insertion
- Inserting a new record is very efficient
- Searching for a record requires linear search
- Deletion techniques
  - Rewrite the block
  - Use deletion marker

NOTE:
To Access a record the average number of
Block Access = $\frac{B}{2}$
(B: Data Blocks)

Access Times for Various File Organizations

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

Average access times for a file of b blocks under basic file organizations

Indexing

- Index are used to Improve the Searching efficiency.
- To Reduce the I/O Cost.
- Provide a faster way to Access the Data.
- Index is an Ordered file.

Indexing

Each Index Record Contain 2 (Two) fields.

Topic Name	Page No.
Search key	Block Pointer (BP)

Block Pointer (BP) → Point to a block where key is available

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

Given

$$\boxed{\text{Block Size} = 100\,\text{Byte}}$$

Record Size = 40 Byte

Block factor $= \left\lfloor \frac{100B}{40B} \right\rfloor = \lfloor 2.5 \rfloor$

= 2 Record Per Block

DB

Assume

Key = 16 Byte Bp = 4 Byte

One Index Record Size = 16 + 4 = 20 Byte

Block factor of Index file $= \left\lfloor \frac{100}{20} \right\rfloor$ = 5 Index Record Per Block.

Q.1 How Cost is Reduced ?

Q.2 Using Index Avg # Block Access ?

Q.3 How We Find Number of Index entries ?

Q.4 Practical Example with Diagram ?

Q.5 Bp & key Size is given or Not ? ⇒ Given in Question.

Q.6 How searching is Done ?

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

Book 1100 Page
(eg Index 30-50 Pages)

- 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]

Example:

- Suppose that:
  - record size R = 150 bytes, block size B = 512 bytes,
    r = 30000 records
- Then, we get:
  - blocking factor Bfr = $\left\lfloor \frac{512}{150} \right\rfloor$ = 3 Record Per Block.
  - number of file blocks b = $\left\lceil \frac{30000}{3} \right\rceil$ = 10,000 Data Block.

Example:

- Suppose that:
- record size R = 150 bytes, block size B = 512 bytes,

  r = 30000 records
- Then, we get:
- blocking factor Bfr = B div R = 512 div 150 = 3 records/block
- number of file blocks $b = (r/\text{Bfr}) = (30000/3) =$ 10000 blocks

Example:

Given the following data file

EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)

Suppose that:

- record size R=150 bytes, block size B=512 bytes r=30000 records
- For an index on the SSN field, assume the field size V_{SSN} =9 bytes, assume the record pointer size P_R=7 bytes. Then:

key

One Index Record Size = 9+7 = 16 Byte

Block factor of Index file = $\left\lfloor \frac{512 B}{16 B} \right\rfloor$ = 32 Index Record Per Block

Total # Index Entries = 30,000.

Total # Index Block = $\left\lceil \frac{30000}{32} \right\rceil$ = 938 Index Block.

GATE 2023

To Access a Index Avg # Block Access $= \lceil \log_2 B_i \rceil \Rightarrow \lceil \log_2 938 \rceil$

$= 10$ Block Access

To Access a Record Avg # Block Access $= \log_2 B_i + 1$

$\Rightarrow \lceil \log_2 938 \rceil + 1$

$= 10 + 1$

$= 11$ Block Access.

Example:

Given the following data file

EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)

Suppose that:

- record size R=150 bytes, block size B=512 bytes r=30000 records
- For an index on the SSN field, assume the field size V_{SSN} =9 bytes, assume the record pointer size P_R=7 bytes. Then:
  - index entry size $R_1=(V_{SSN}+P_R)=$ 9+7=16
  - index blocking factor Bfr_1=B div R_1= $\boxed{\frac{512}{16}}$ entries / block
  - number of index blocks $b=(r/Bfr_1)=$ = blocks
  - binary search needs $\log_2 bI=\log_2 938$ = block accesses

Example:

$\#\text{Blocks} = \frac{30000}{3} = 10000\ \text{DB Block}$

Given the following data file

EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)

Suppose that:

- record size R=150 bytes, block size B=512 bytes r=$\boxed{\text{30000 records}}$
- For an index on the SSN field, assume the field size V_{SSN} =9 bytes, assume the record pointer size P_R=7 bytes. Then:
  - index entry size $R_1=(V_{SSN}+P_R)=(9+7)=16$ bytes
  - index blocking factor Bfr_1=B div R_1=512 div16=$\boxed{\text{32 entries / block}}$ Index
  - number of index blocks $b=(r/Bfr_1)=(30000/32)=$ 938 blocks
  - binary search needs $\log_2 bI=\log_2 938$ =10 block accesses

To Access a Record Using Index = $\log_2 Bi + 1 = 10+1 = 11$ Ans

Dense Index

Index entry created for every Search key Value

SPARSE Index

Index entry is created for some search key Value.

Category of Index

1) Dense Index Files

Number of Index entries = Number of DB Records

2) Sparse Index Files

Number of Index entries = Number of Blocks

Dense Index Files

- Dense Index - Index record appears for every search-key values in the file.
- Example - index on *ID* attribute of *instructor* relation

10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

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	

Sparse Index Files

- Sparse Index: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value K we:
  - Find index record with largest search-key value $< K$
  - Search file sequentially starting at the record to which the index record points

10101	
32343	
76766	

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	

Types of Index

Single-level Ordered Indexes

- Primary indexes
- Clustering indexes
- Secondary indexes

Multilevel Indexes

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

Q.1

Assume a relational database system that holds relation: C(colleges) with the following characteristics

- Records are stored as fixed length, fixed format records, length is 256 bytes.
- There are 16384 records.
- Records contains key attribute CollegeNumber (C.N), length 22 bytes and other fields.
- Unspanned organization is used to store the information or record.

Let's suppose we want to build a sparse primary index on C.N then how many numbers of 4096-byte blocks are needed to store the primary index when block pointer size is 10 bytes _____?

A. 7

B. 8

C. 9

D. 10

File Organization:

- Data Records can be :
  1) Fixed Length Records
  2) Variable Length Records

≈ 200 bytes

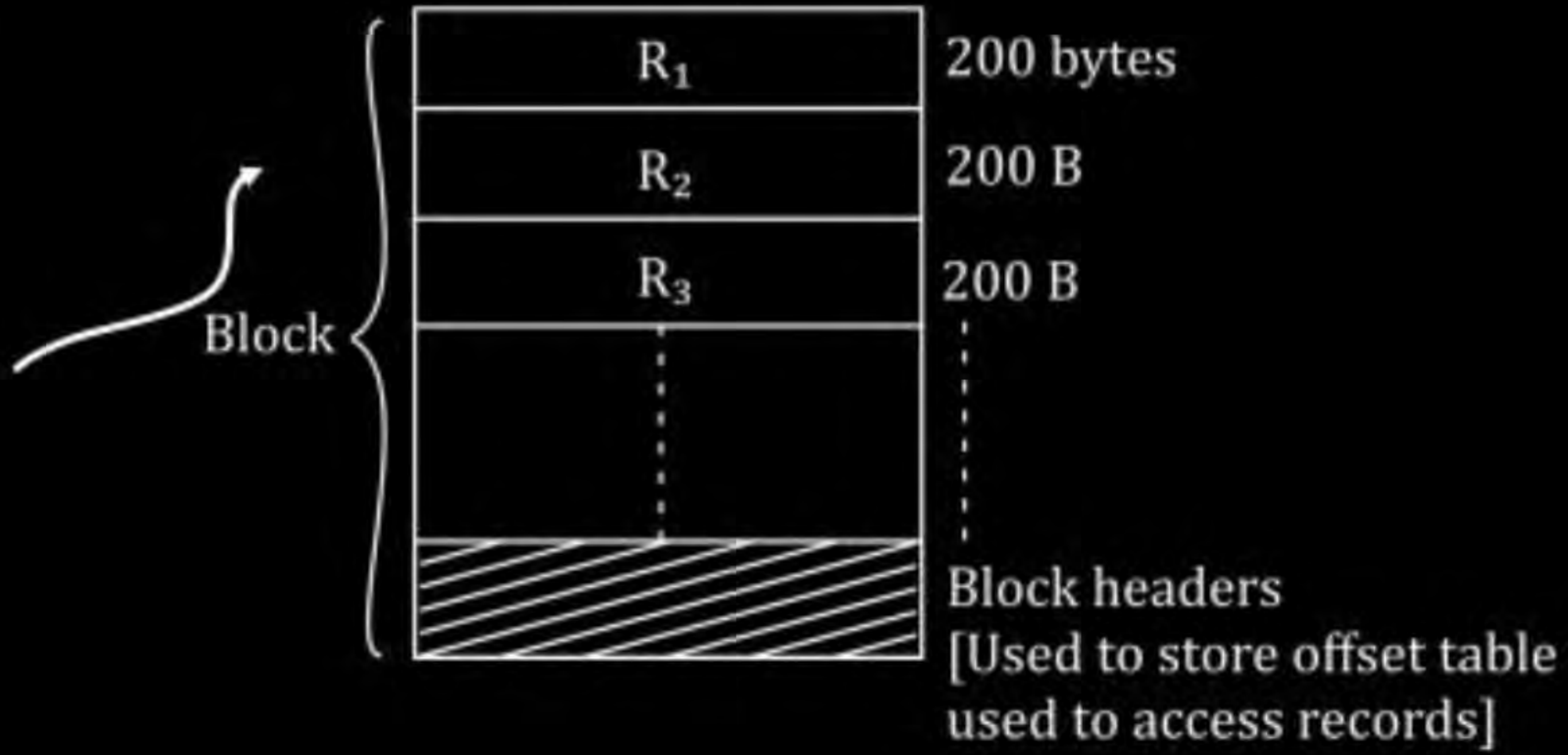
1) Fixed Length Records

Create table R

(A char (100),

B char (50),

C char (50));



File Organization:

2) Variable Length Records

Create table S

(D char (100),

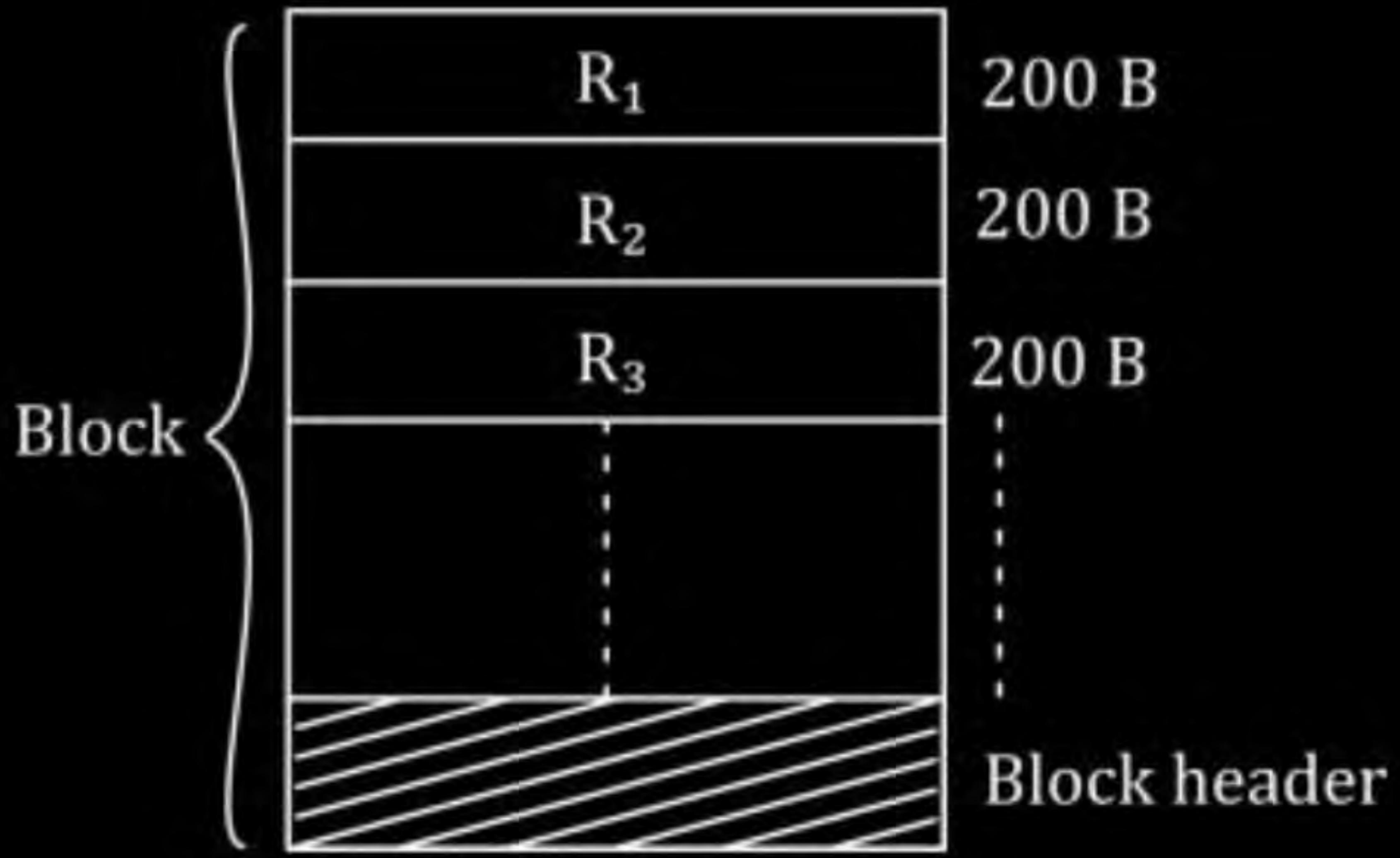
E char (50),

F text

);

$\therefore$ size (D) + Size (E) + size(F)

100 50 not fixed



File Organization:

$\left\{\begin{array}{l}\text{❑ DB File with all records fixed length} \\ \text{❑ DB file with variable length records}\end{array}\right\} \Rightarrow$ both are possible in RDBMS

Record Organization:

(a) Spanned Organization

Record allow to span in more than one block

Example: Block size 1000B Record size 400B

Advantage:

⇒ Possible to allocate file with no internal fragmentation.

Record Organization:

(a) Spanned Organization

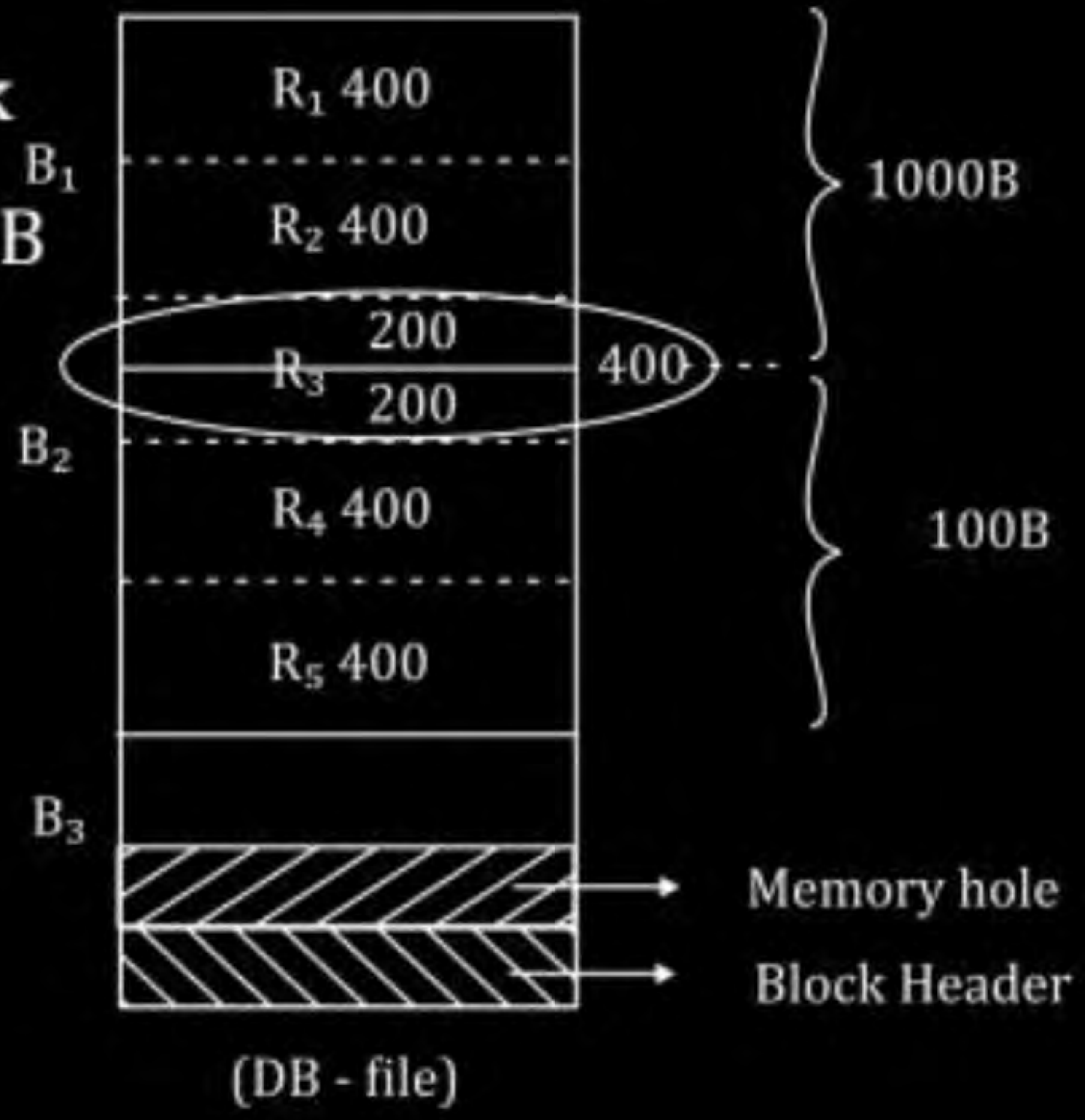
Record allow to span in more than one block

Example: Block size 1000B; Record size 400B

$$\text{Block Factor} = \frac{1000 - 0}{400} = 2.5\ R/B$$

⇒ Too complex to manage records

⇒ More access cost to access records.



Advantage:

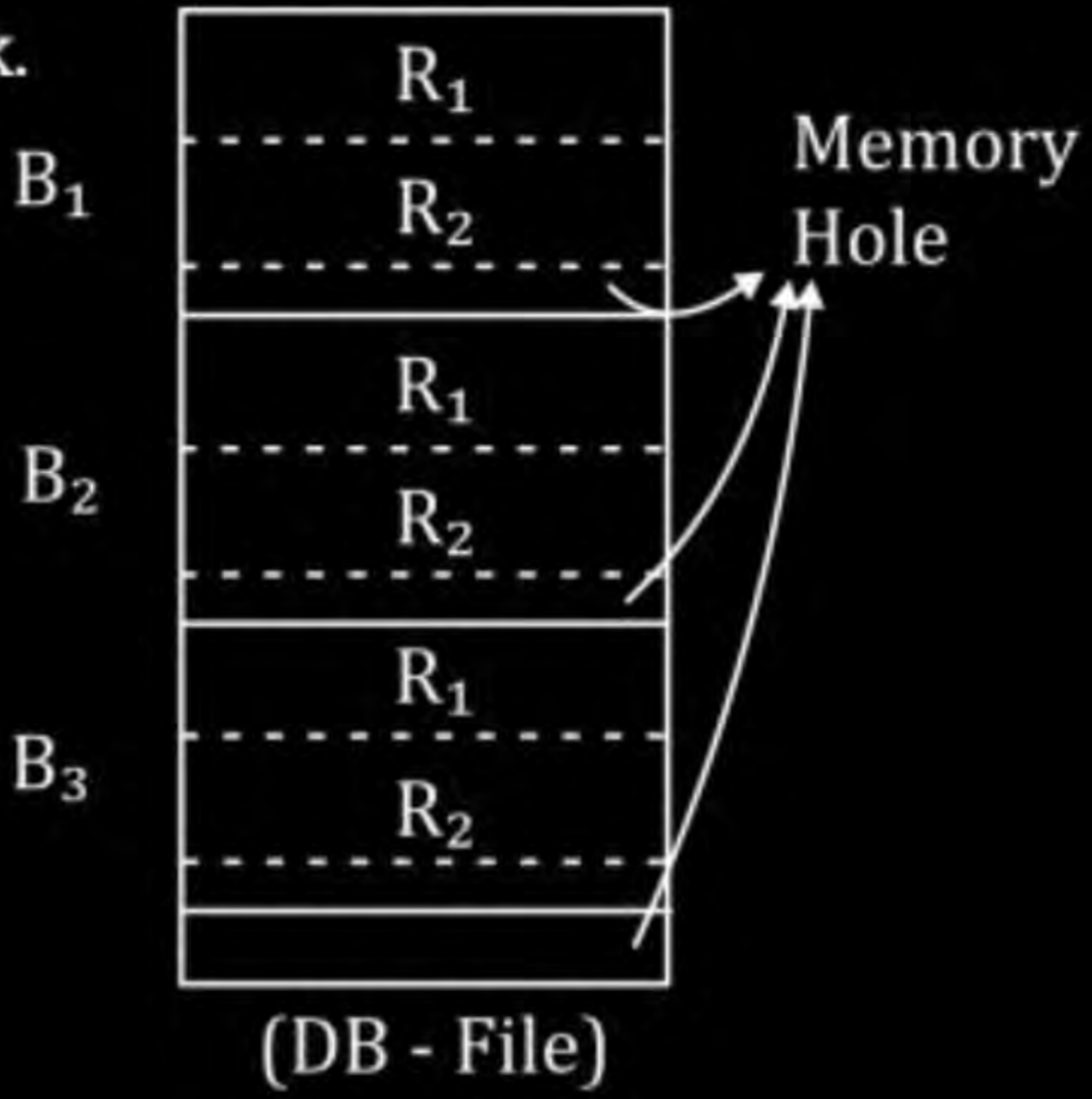
⇒ Possible to allocate file with no internal fragmentation

Record Organization:

(a) Unspanned Organization

Complete Record must be stored in one Block.

Block Factor $= \left\lfloor \frac{1000-0}{400} \right\rfloor = 2\ R/B$



⇒ Easy to Manage records

⇒ Access cost is also less.

⇒ May not possible to avoid internal fragmentation.

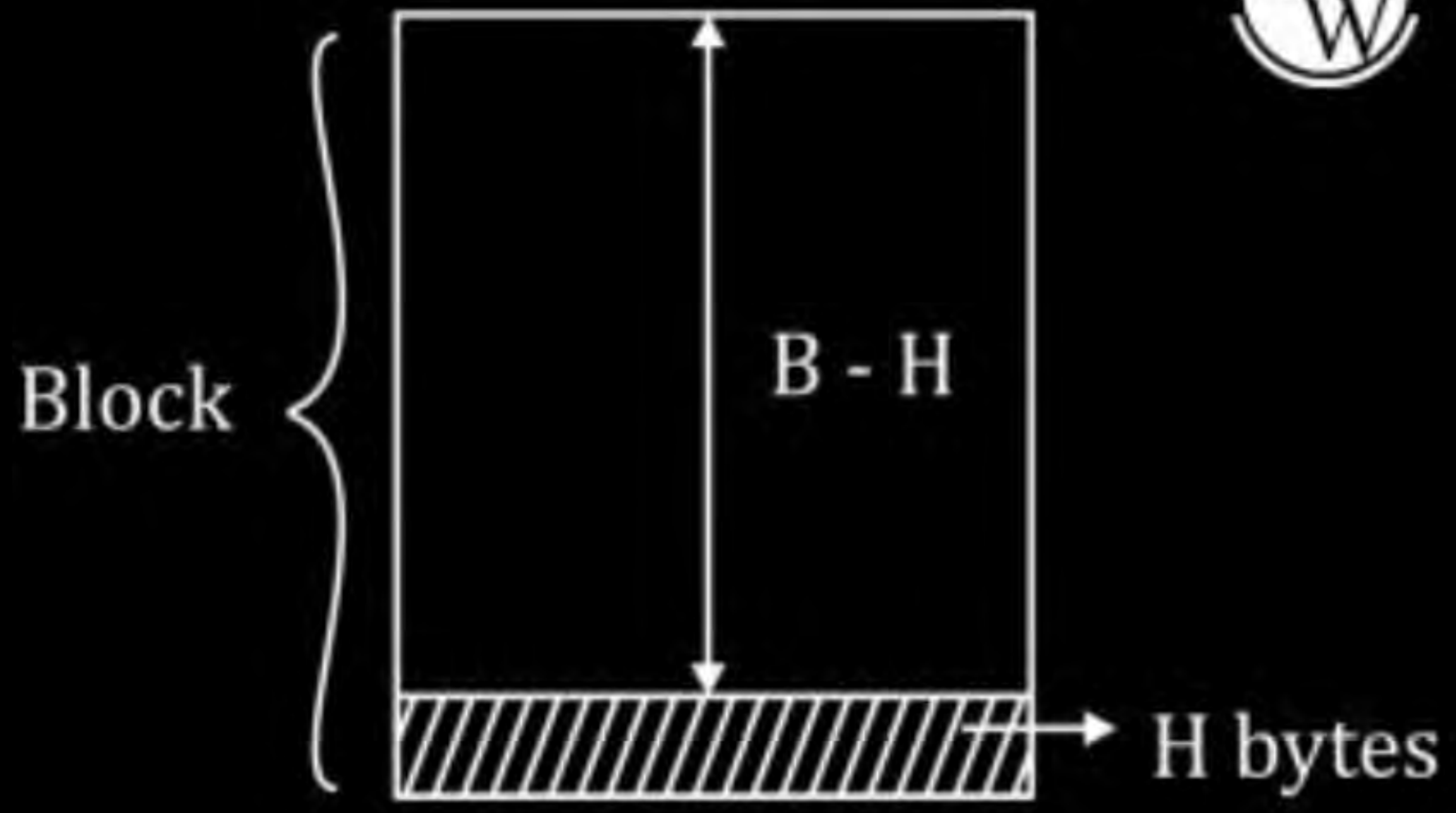
Block Factor

Maximum Possible Records per Block.

Block Size: B bytes

Block Header size: H bytes

Record size: R bytes



- Block factor for unspanned: $\left\lfloor \frac{B-H}{R} \right\rfloor$ record / block

  because for fixed length record unspanned is preferred.

- Block Factor for spanned: $\frac{B-H}{R}$ record / block

NOTE:

1) To organize DB file with fixed length record unspanned organization is preferred.
2) To organize DB file with variable length record spanned organization preferred.

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 $= \log_2 B_i + 1$

$\log_2 B_i$ → Index Block access

$+1$ → Data Block access

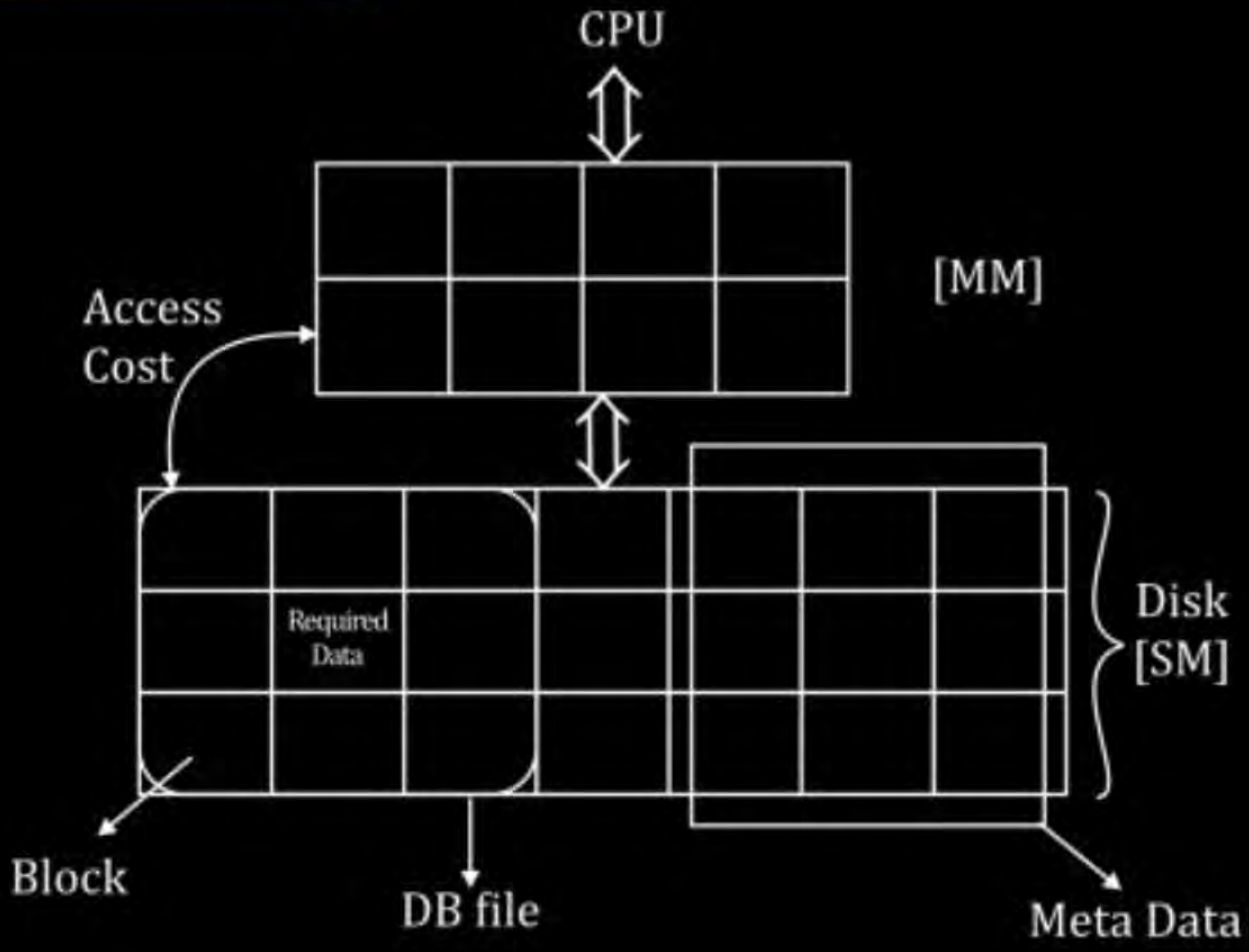
[Bi : Index Block]

Indexing

Used to reduce access cost or I/O cost.

Access Cost: Number of SM(secondary Memory) Blocks (Disk blocks) to transfer from SM to MM in order to access required data.

Indexing



CPU
[MM]
Access
Cost
Required
Data
Disk
[SM]
Block
DB file
Meta Data

Meta Data [Data Dictionary]

→ record format

→ Field format

→ Number of blocks allocated for file

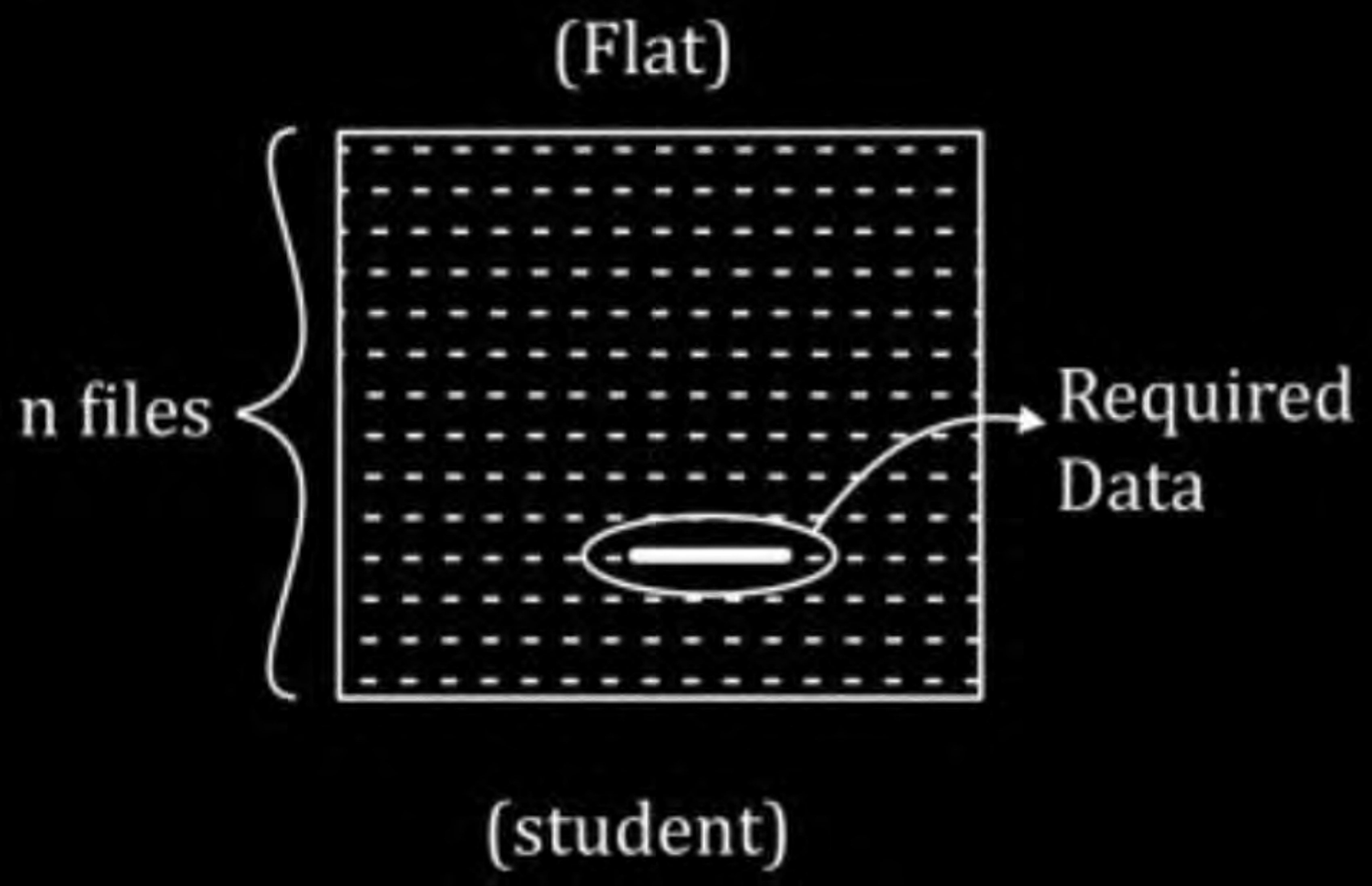
→ Number of record in file

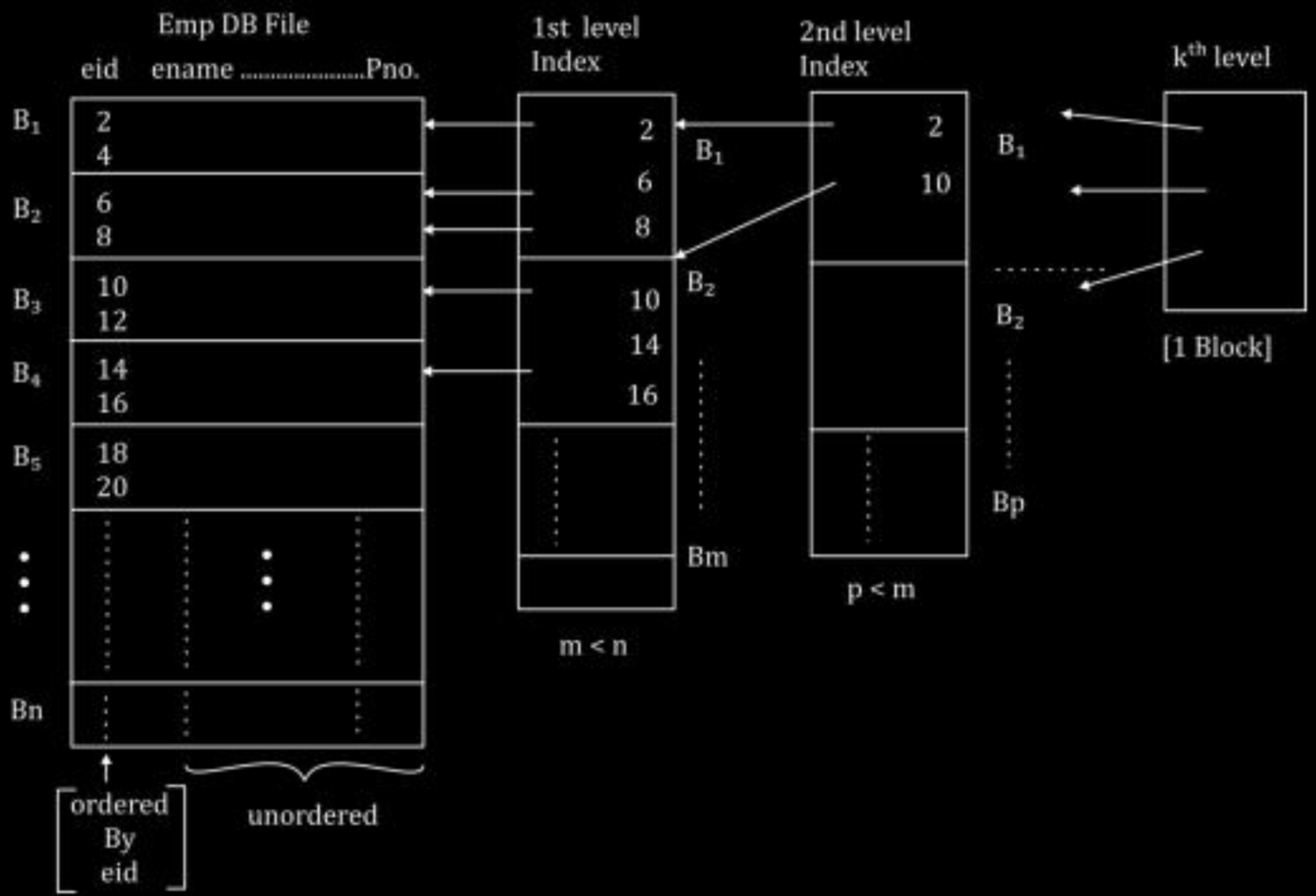
→ Ordered field of file, etc

Indexing

Huge access cost to access data from flat file system.

[complete file should transfer to mm to locate required data]





Emp DB File
eid ename ....Pno.
B1
2
4
B2
6
8
B3
10
12
B4
14
16
B5
18
20
Bn
ordered By eid
unordered
1st level Index
2
6
8
10
14
16
B1
B2
Bm
m < n
2nd level Index
2
10
p < m
B1
B2
Bp
kth level
[1 Block]

Index File:

$$\text{BF (Block Factor) of Index} = \left\lfloor \frac{B-H}{K+P} \right\rfloor \text{ entries / block}$$

k : search key
P: Pointer

Index File

k_1	p_1
k_2	p_2
⋮	⋮

$$\therefore \text{BF} = \left\lfloor \frac{B-H}{k-p} \right\rfloor$$

DB File

Index file

$$\left\lfloor \frac{B-H}{R} \right\rfloor \text{ record / block} < \left\lfloor \frac{B-H}{K+P} \right\rfloor \text{ entries / block}$$

$$\left\{ \begin{matrix} \text{\# of DB} \\ \text{File block} \end{matrix} \right\} > \left\{ \begin{matrix} \text{\# of Index} \\ \text{blocks} \end{matrix} \right\}$$

Multilevel Index:

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]

Category of Index

1) Dense Index Files

Number of Index entries = Number of DB Records

2) Sparse Index Files

Number of Index entries = Number of Blocks

Dense Index Files

- Dense Index - Index record appears for every search-key values in the file.
- Example - index on *ID* attribute of *instructor* relation

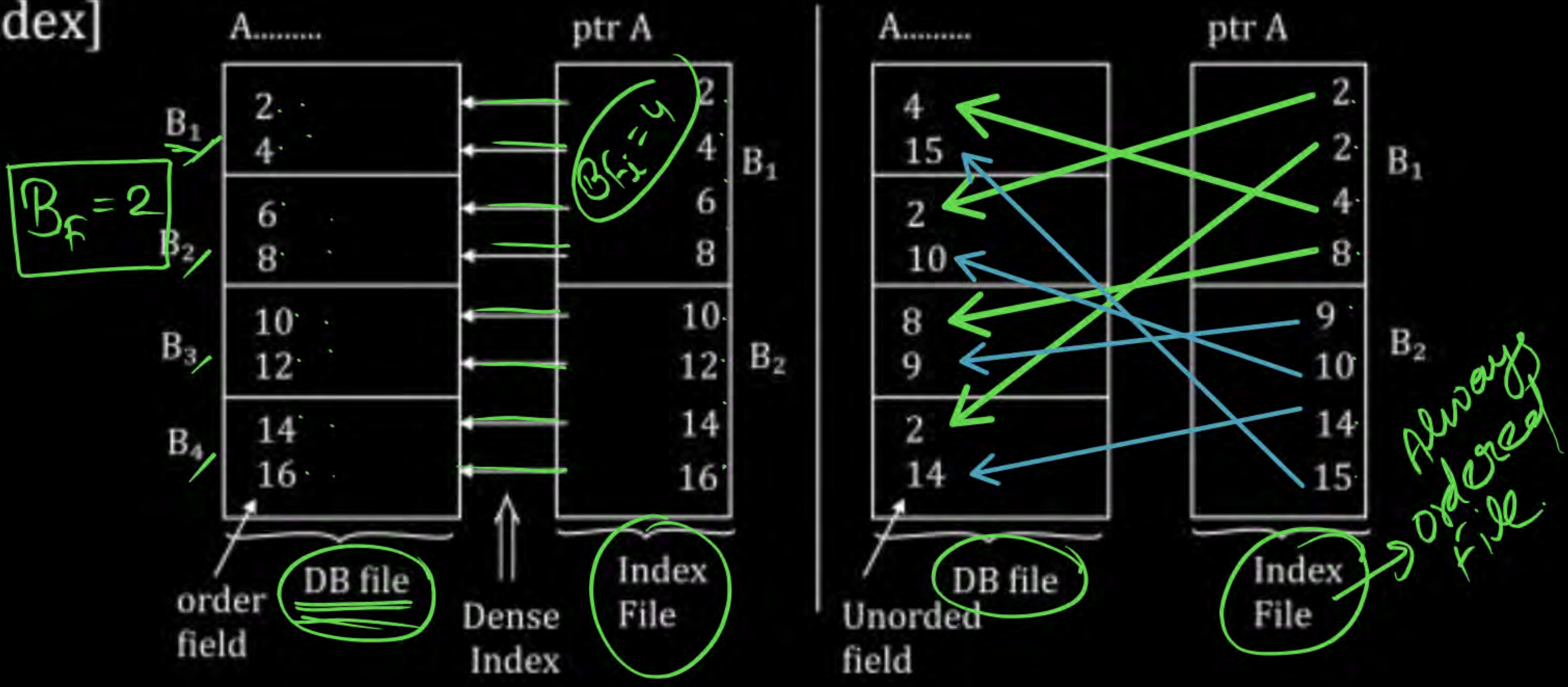
10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

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	

Dense Index [More entries in Index File]

⇒ For each DB record of DB file there exist entry in index file

[Dense Index]



⇒ (# of Index Entries in Index File) ≡ (# of DB records in DB file)

Sparse Index Files

- Sparse Index: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value K we:
  - Find index record with largest search-key value $< K$
  - Search file sequentially starting at the record to which the index record points

10101	
32343	
76766	

BF:4

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	

Sparse Index [Less entries in Index File]

⇒ For set of DB records there exist entry in Index file such a Index is called Sparse Index.

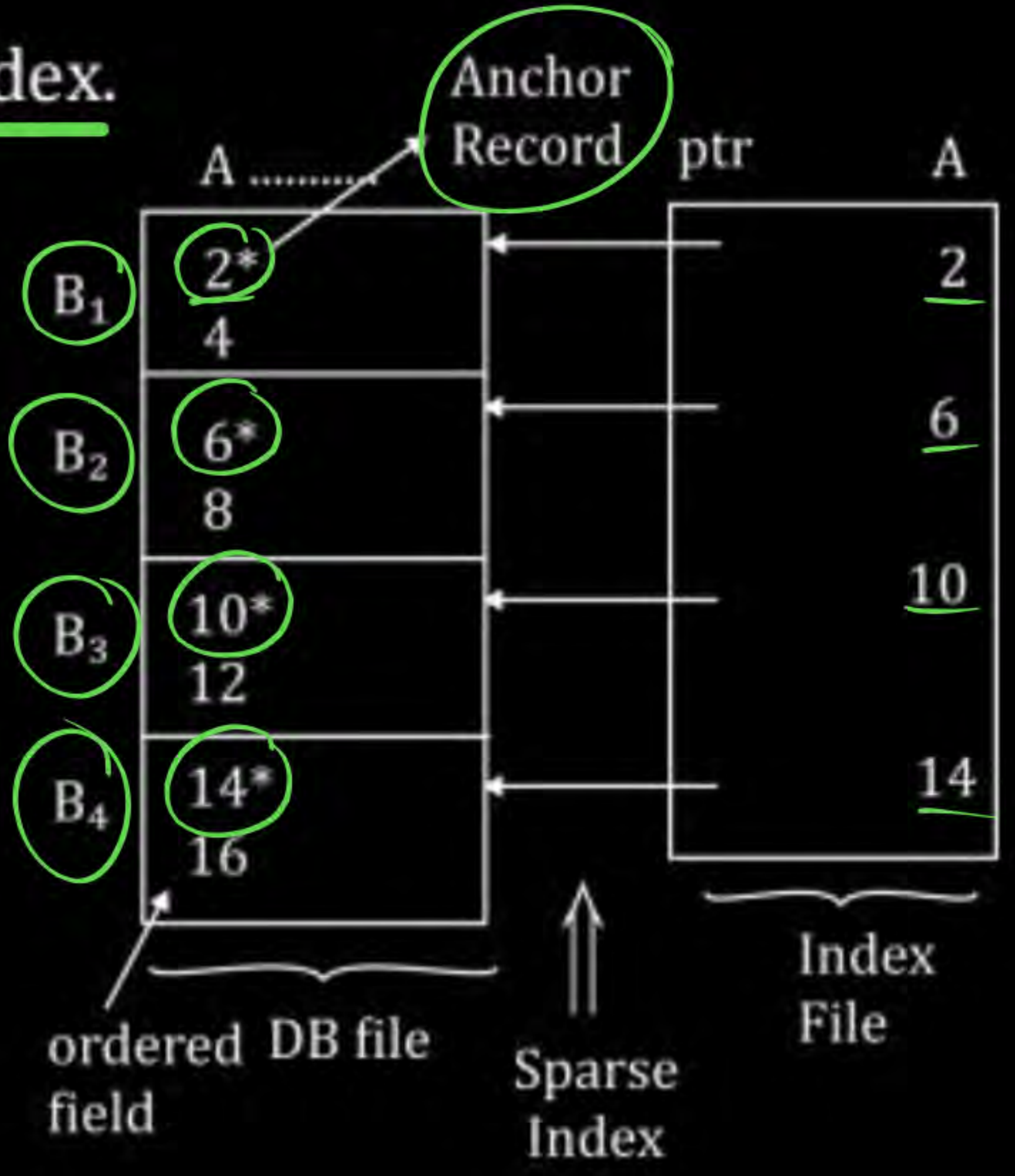
⇒ Sparse Index can build only over ordered field of file.

Anchor Record: First record of Block

[Sparse Index]

⇒ [# of entries in Index] < [# of DB records]

$$\left\{\begin{matrix}\text{\# of Index} \\ \text{entries}\end{matrix}\right\} \equiv \left\{\begin{matrix}\text{\# of DB} \\ \text{blocks}\end{matrix}\right\}$$



Index File:

$$\text{BF (Block Factor) of Index} = \left\lfloor \frac{B-H}{K+P} \right\rfloor \text{ entries / block}$$

$$\text{Index File} \left\{ \boxed{\begin{array}{cc} k_1 & p_1 \\ k_2 & p_2 \\ \vdots & \vdots \end{array}} \right.$$

$$\therefore \text{BF} = \left\lfloor \frac{\text{B}-\text{H}}{\text{k}-\text{p}} \right\rfloor$$

$$\left\lfloor \frac{B-H}{R} \right\rfloor \text{ record / block} < \left\lfloor \frac{B-H}{K+P} \right\rfloor \text{ entries / block}$$

$$\left\{ \begin{array}{l} \text{\# of DB} \\ \text{File block} \end{array} \right\} > \left\{ \begin{array}{l} \text{\# of Index} \\ \text{blocks} \end{array} \right\}$$

Example:

- Suppose that:
  - record size R = 150 bytes, block size B = 512 bytes, r = 30000 records
- Then, we get:
  - blocking factor Bfr = B div R = 512 div 150 = 3 records/block
  - number of file blocks $b = (r/\mathrm{Bfr}) = (30000/3) = 10000$ blocks

Example:

Given the following data file

EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)

Suppose that:

- record size R=150 bytes, block size B=512 bytes r=30000 records
- For an index on the SSN field, assume the field size V_{SSN} =9 bytes, assume the record pointer size P_R=7 bytes. Then:

Example:

Given the following data file

EMPLOYEE (NAME, SSN, ADDRESS, JOB, SAL, ...)

Suppose that:

- record size R=150 bytes, block size B=512 bytes r=30000 records
- For an index on the SSN field, assume the field size V_{SSN} =9 bytes, assume the record pointer size P_R=7 bytes. Then:
  - index entry size $R_1=(V_{SSN}+P_R)=(9+7)=16$ bytes
  - index blocking factor Bfr_1=B div R_1=512 div16=32 entries / block
  - number of index blocks $b=(r/Bfr_1)=(30000/32)=$ 938 blocks
  - binary search needs $\log_2 bI=\log_2 938=10$ block accesses

Q.

# of records of file: 16384 $[2^{14}]$

Block size: 4096 Bytes $[2^{12}$ Byte$]$

Record size: 256 Bytes $[2^{8}$ Byte

Search key size: 22 Bytes

Pointer : 10 Bytes

32B $(2^{5}$ Byte$)$

⇒ I/O cost to access record without index based on ____

(a) Avg Ordered field = $\log_2 B$

(b) Avg Unordered Filed = $B/2$

⇒ Index Dense [Using]

(a) # of index blocks at 1st level =

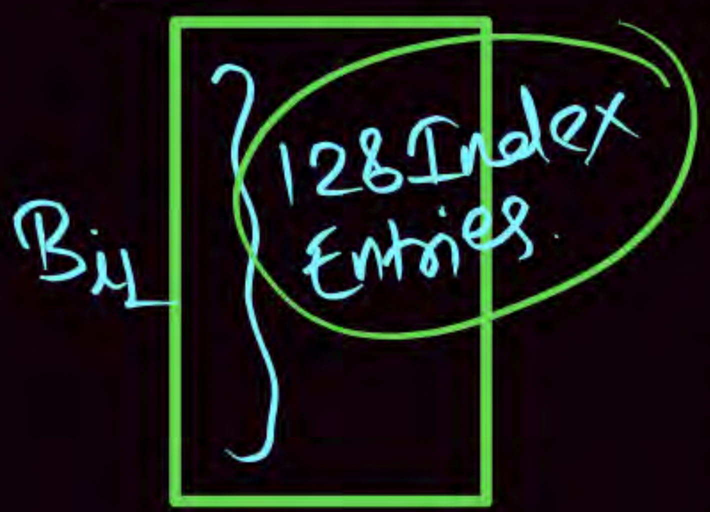
(b) I/O cost to access record: [Using 1st level] =

⇒ By using sparse Index:

(a) # of Index blocks at 1st level =

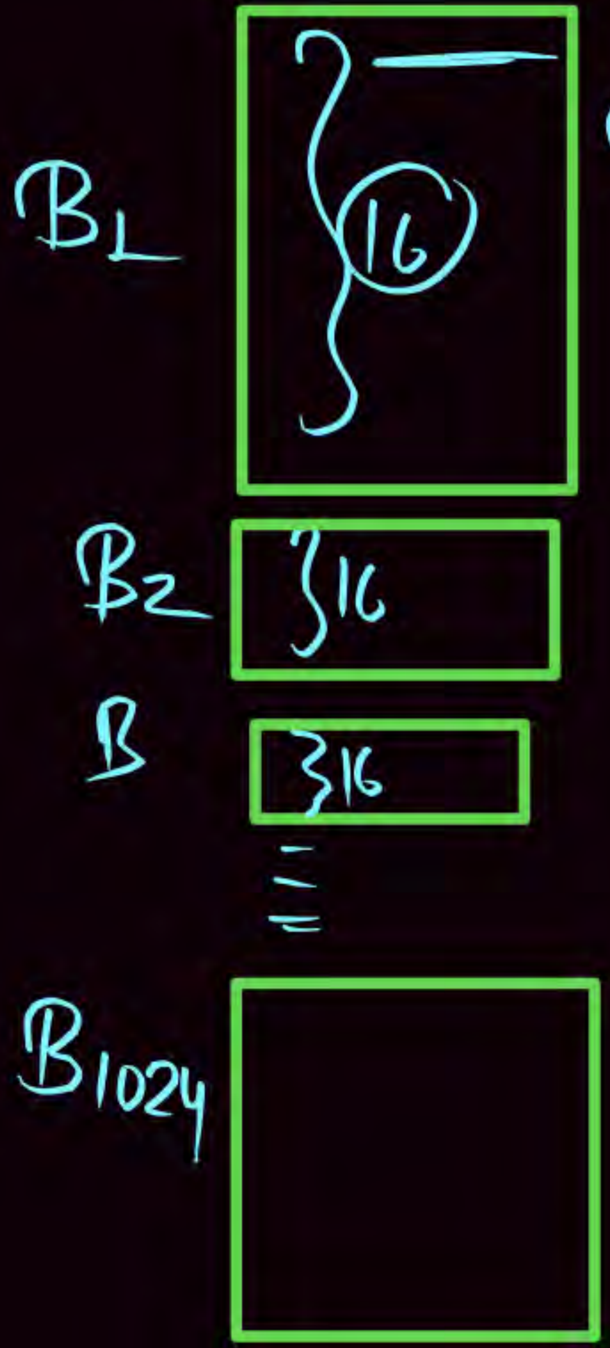
(b) I/O cost to access record: [Using 1st level] =

Index



Record = 32B

DB File



B_F = 16 Record Per Block.

Tuple (Record) = 256 Byte

Records = 16.384 [2^{14}]   Block Size = 4096 B [2^{12} B]

Record Size = 256 B [2^8 B]   key = 22 B   Pointer = 10 Byte

DB File

Block factor of DB File = $\left\lfloor \frac{\text{Block Size}}{\text{Record Size}} \right\rfloor \Rightarrow \left\lfloor \frac{4096\,B}{256\,B} \right\rfloor \Rightarrow \frac{2^{12}}{2^8} = 2^4$ = 16 Record Per Block

Total # DATA Blocks = $\left\lceil \frac{\#\text{Records}}{\text{Bf of DB File}} \right\rceil \Rightarrow \left\lceil \frac{16.384}{16} \right\rceil \Rightarrow \frac{2^{14}}{2^4} \Rightarrow 2^{10}$

$\boxed{B = 1024}$

B = 1024 DATA Block

Indexing   One Index Record Size = Size of key + Size of Pointer ⇒ 22+10 = 32 Byte [2^5 B]

Index File

Block factor of Index file = $\left\lfloor \frac{\text{Block Size}}{\text{Index Record Size}} \right\rfloor \Rightarrow \left\lfloor \frac{4096\,B}{32\,B} \right\rfloor = \frac{2^{12}}{2^5} = 2^7$

⇒ 128 Index entries Per Block.

$\# Records = 16.384\ [2^{14}]$ Block Size $= 4096\,B\ [2^{12}B]$

Record Size $= 256B\ (2^8 B)$ key $= 22B$ Pointer $= 10$ Byte

Without Index :

ORDERED File

To Access a Cost Avg #

Block Access $= \lceil \log_2 B \rceil$

$= \lceil \log_2 1024 \rceil$

$= 10$ Block Access

UNORDERED File

To Access a Record Avg. #

Block Access $= B/2$

$= \frac{1024}{2} = 512$ Block Access Avg

#Records = 16,384 [2^{14}] Block Size = 4096 B [2^{12} B]

Record Size = 256 B (2^8 B) key = 22 B Pointer = 10 Byte

With Index **Dense**

(#DB Records)

Total # Index entries = 16,384 [2^{14}]

Bf_i [Block factor of Index file] = 128 Index Record Per Block

Total Number of Index Block = $\left\lceil \frac{16,384}{128} \right\rceil \Rightarrow \frac{2^{14}}{2^7} = 2^7$

= 128 Index Block

With Index : SPARSE

Total # Index Entries = 1024 (#DATA Blocks)

Block factor of Index file [Bf_i] = 128 Index Record Per Block.

Total Number of Index Block = $\left\lceil \frac{1024}{128} \right\rceil = \frac{2^{10}}{2^7} = 2^3$

= 8 Index Block.

\# Records = 16,384 $[2^{14}]$ Block Size = 4096 B $[2^{12}B]$

Record Size = 256B $(2^8 B)$ key = 22B Pointer = 10 Byte

Index is ordered File:

Cost with Index Dense

To Access Index Avg # Block Access $= \lceil \log_2 B \rceil \Rightarrow \lceil \log_2 128 \rceil$

$= 7$

To Access a Record Using Index Avg Number of Block Access $\Rightarrow \log_2 B_i + 1$

$\Rightarrow \lceil \log_2 128 \rceil + 1$

$= 8$ Avg

With Index: SPARSE

To Access Index Avg # Block Access $= \lceil \log_2 8 \rceil = 3$

To Access a Record Using Index Avg Number of Block Access $= \lceil \log_2 B_i \rceil + 1$

$\Rightarrow \log_2 8 + 1$

$= 4$ Avg

Q.

\# of records of file: 16384 $[2^{14}]$

Block size: 4096 Bytes $[2^{12} B]$

Record size: 256 Bytes

Search key size: 22 Bytes

Pointer : 10 Bytes

⇒ I/O cost to access record without index based on ____ Average

(a) Ordered field = 10 Ans

(b) Unordered Filed = 512 Ans

⇒ Index Dense [Using]

(a) # of index blocks at 1st level = 128 Ans

(b) I/O cost to access record: [Using 1st level] = 8 Ans

⇒ By using sparse Index:

(a) # of Index blocks at 1st level = 8 Ans

(b) I/O cost to access record: [Using 1st level] = 4 Ans

Index

Single Level Index

1. Primary Index
2. Clustered Index
3. Secondary Index

Multi Level Indexing.

- Static Multilevel Indexing
- Dynamic Multi Level Indexing
  - B Tree
  - B+ Tree.

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 (Nonkey + ordered file)
3. Secondary Index (~~Nonkey~~ C.K [Cand. key] + Unordered file)

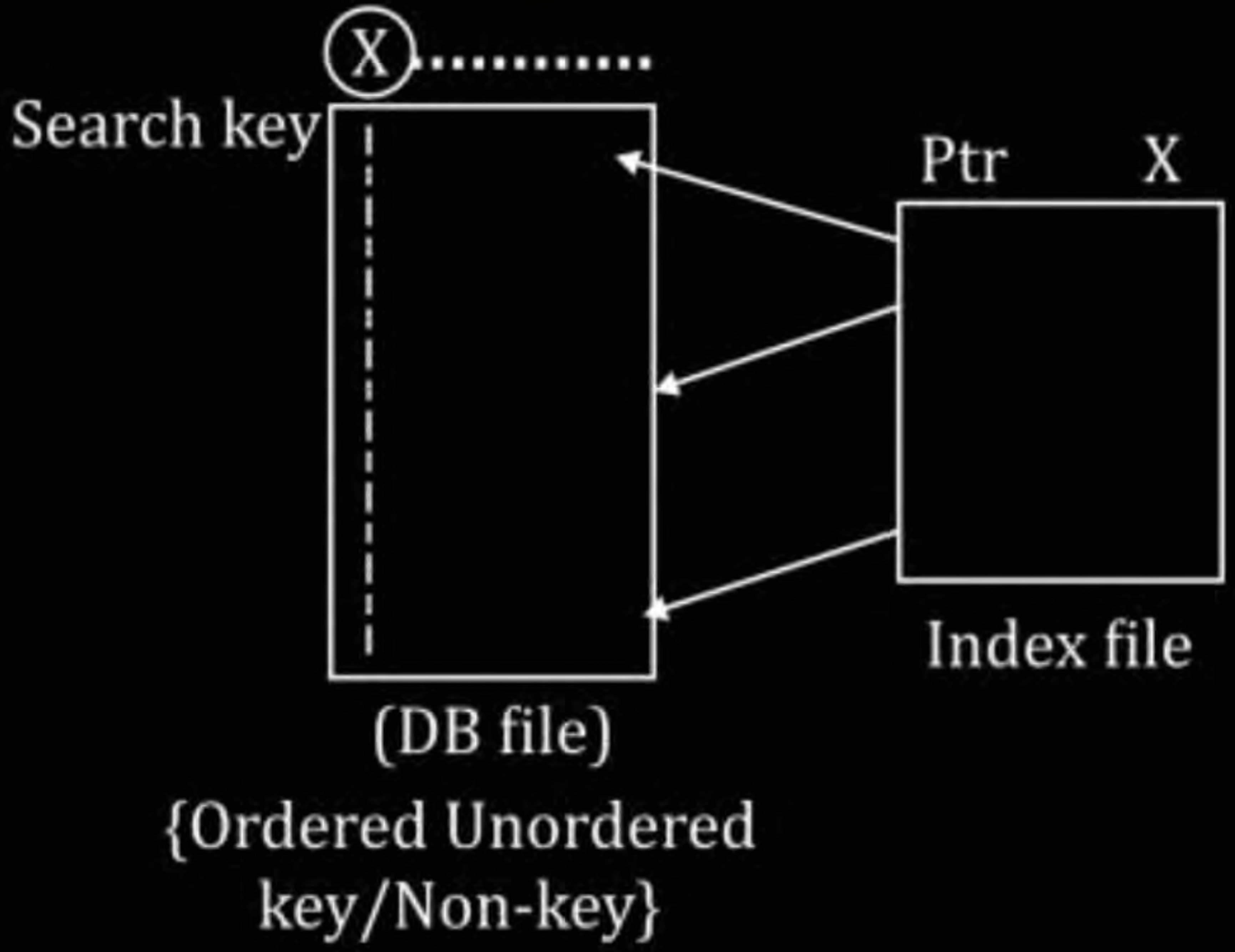
Types of Index

Type of Index
(X: search key)

- X is ordered field of DB file
  - X is primary key Alternative key of DB file ⇓ Primary Index
    key + ordered File
  - X is non key of DB file ⇓ Clustering Index
    (Non key + ordered File)
- X is Unordered field of DB file
  - X is primary key Alternative key of DB file ⇓ Secondary Index (over key)
  - X is non key of DB file ⇓ Secondary Index (Over Non-key)

Home work.

Types of Index



Types of Index

Q. Data records ordered based on non-key (x) and Index built over key field (y) of DB Table :

Types of Index

Q. Data records ordered based on non-key and Index order key field of DB Table :

A	B
2	
2	
3	
3	
4	
4	
5	

B: Search key {used for Indexing}

$\Rightarrow$ Secondary Index (Over key)

$\Downarrow$

key (Unordered)

Types of Index

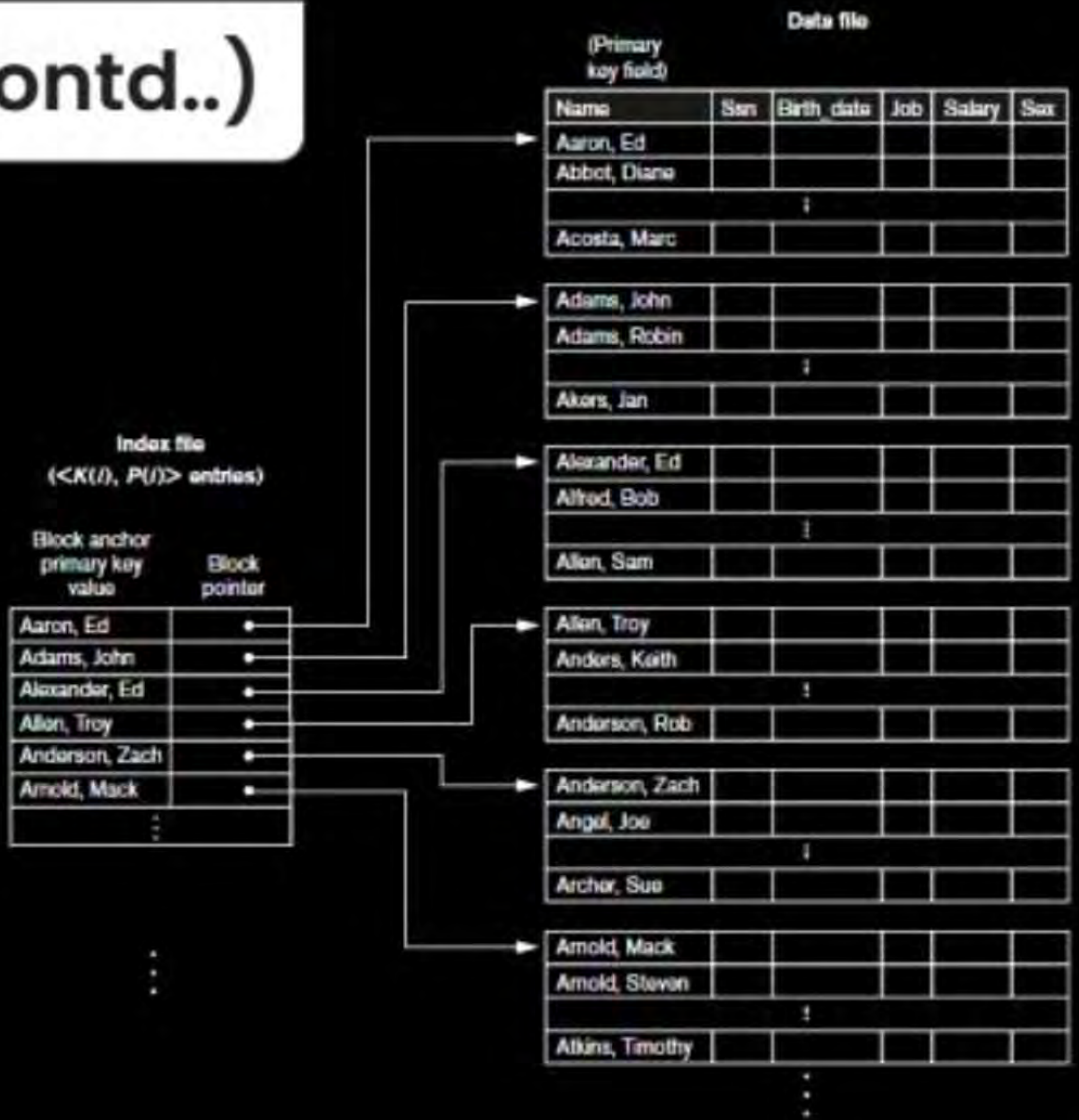
Q. Data records ordered based on non-key and Index build over same non-key: Clustering Index

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

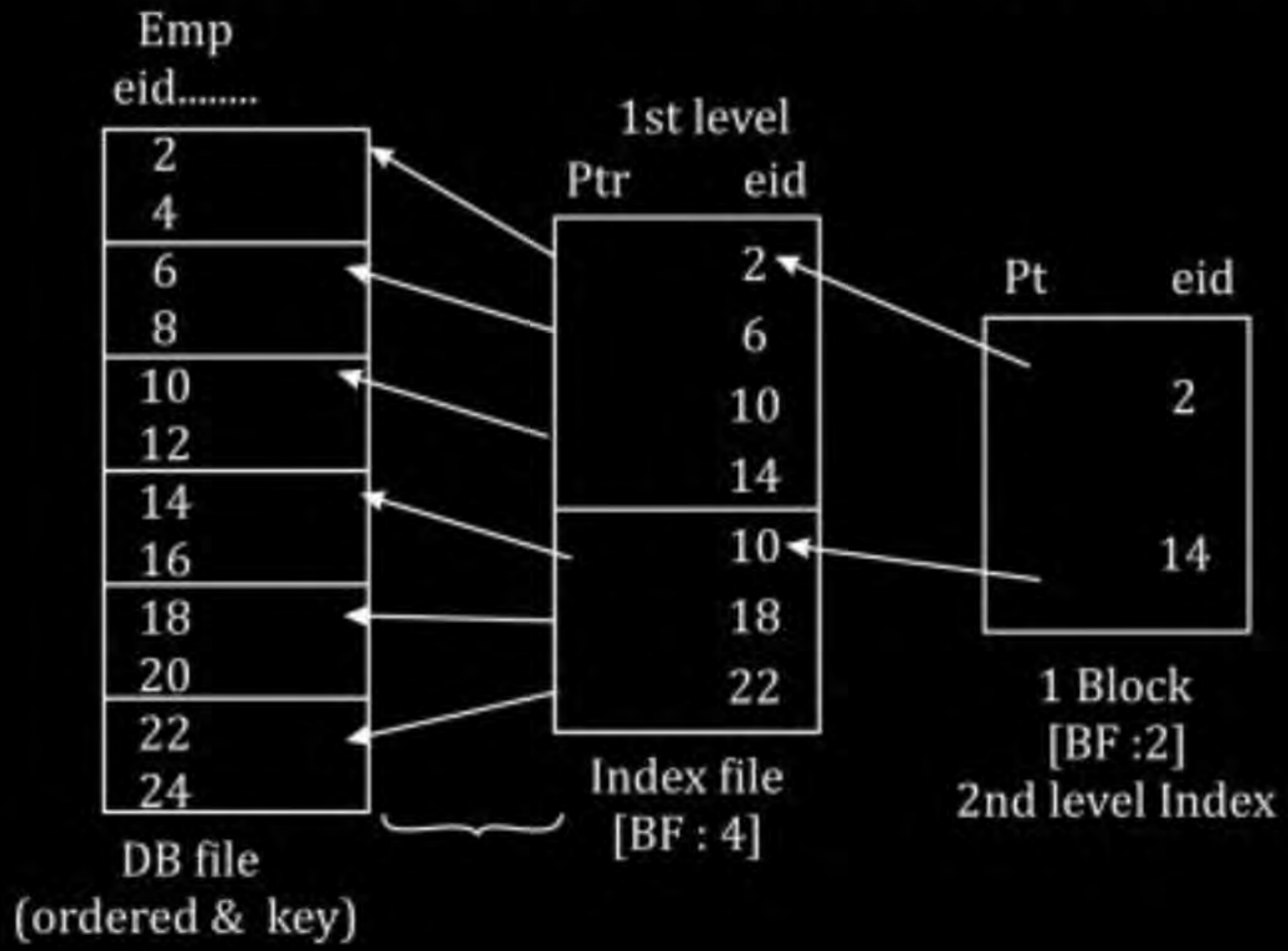
Primary Index (Contd..)

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.



$\Rightarrow$ 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 ?

Apply

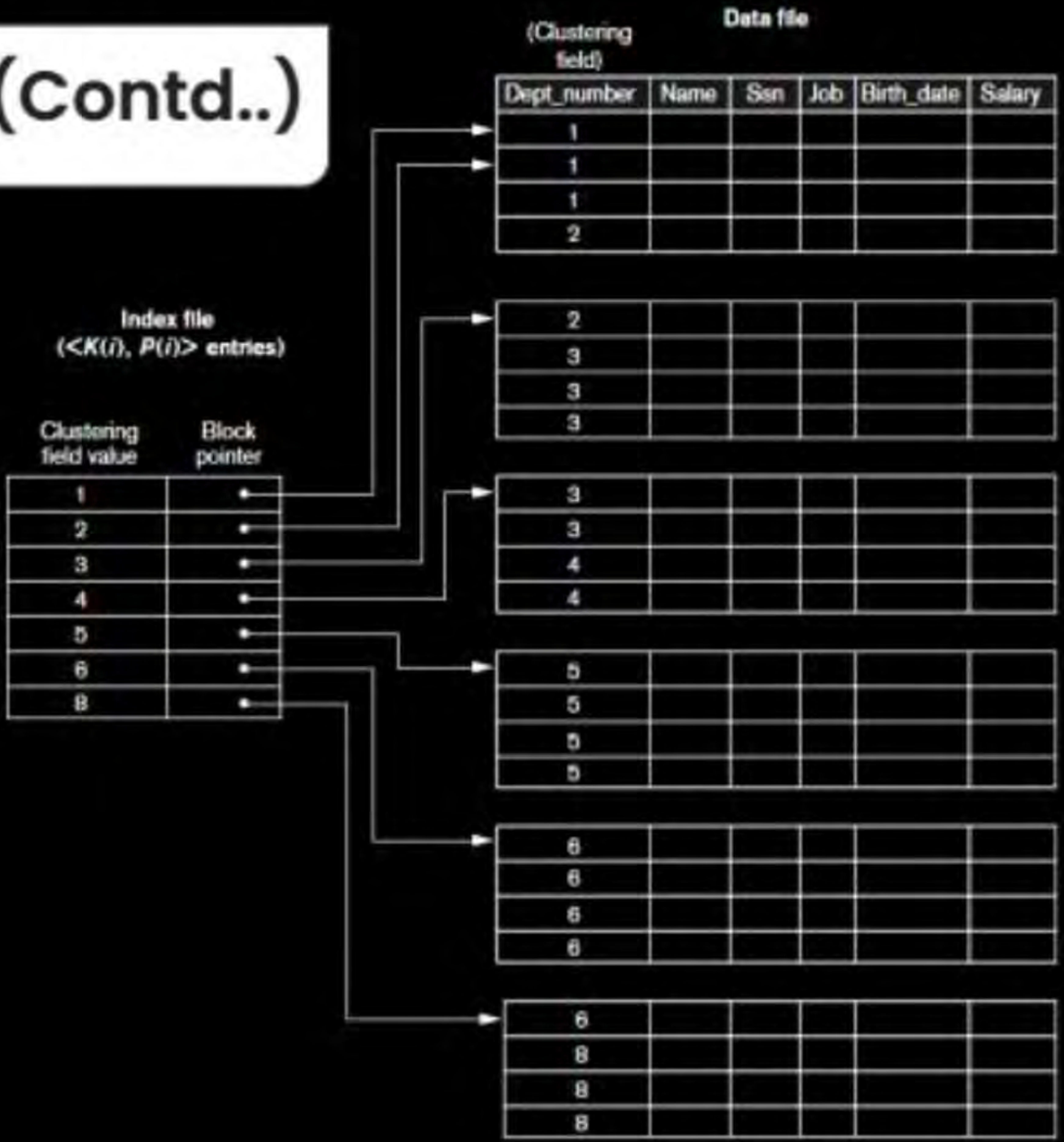
SPARSE

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

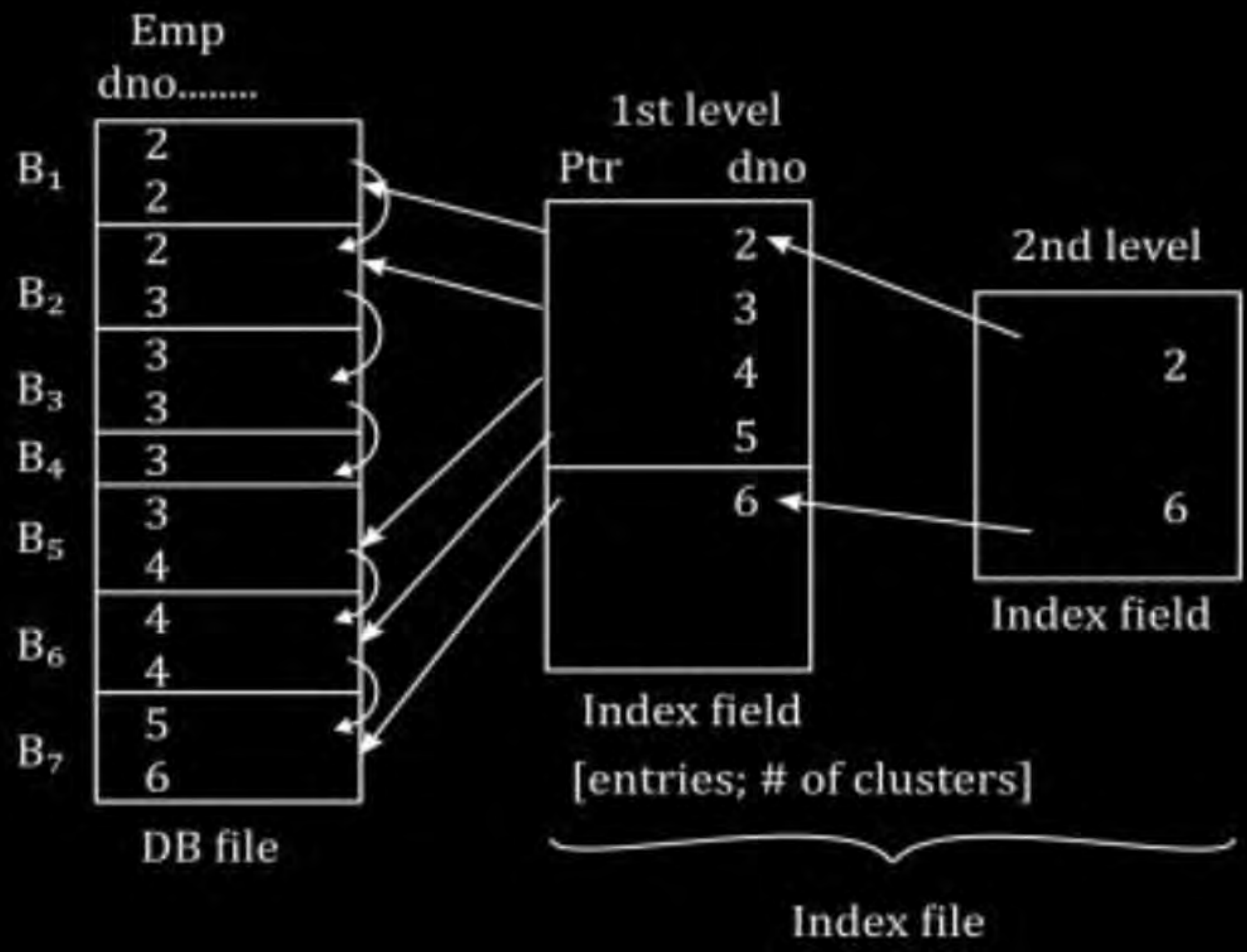
Clustering Indexes (Contd..)

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



Clustering Index:

Search key: Ordered field & Non- key.

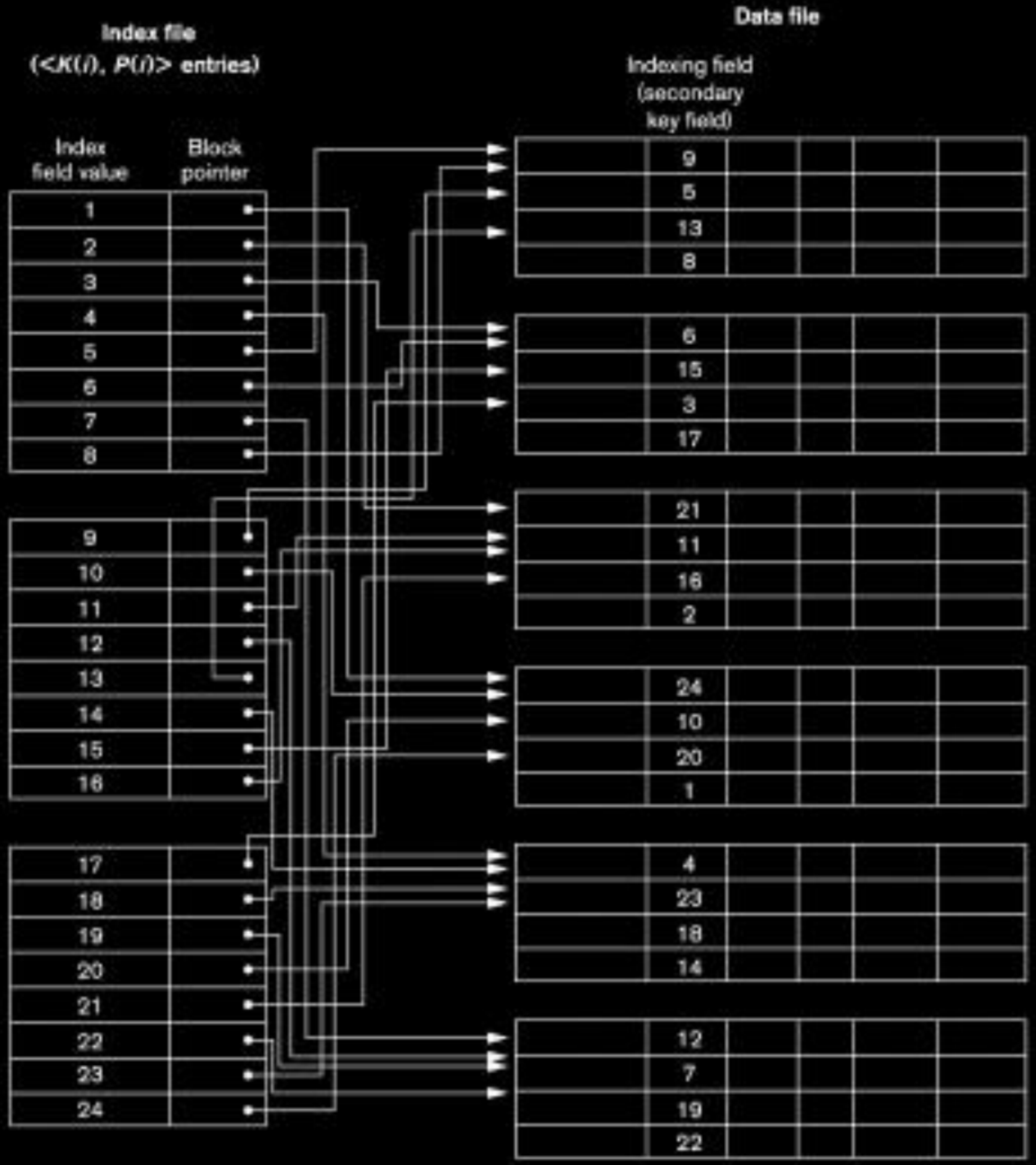


- Clustering Index mostly sparse Index [Dense CI also possible If each cluster with one record]
- At most one CI is possible for any Database relation [ordering required]
- For any DB relation can build either PI or CI but not both.

Secondary Indexes

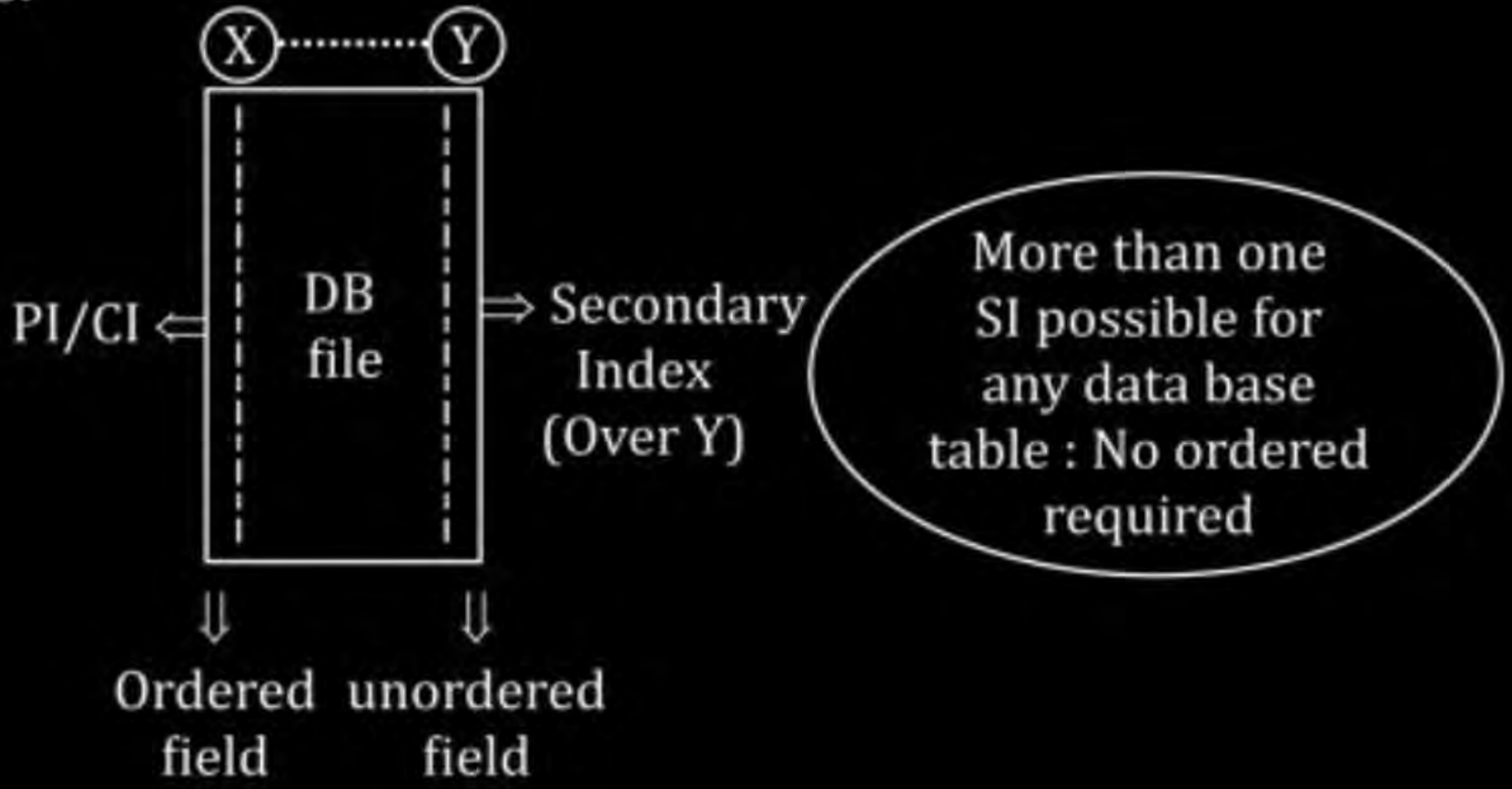
- Secondary Index
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
  - The index is an ordered 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..)

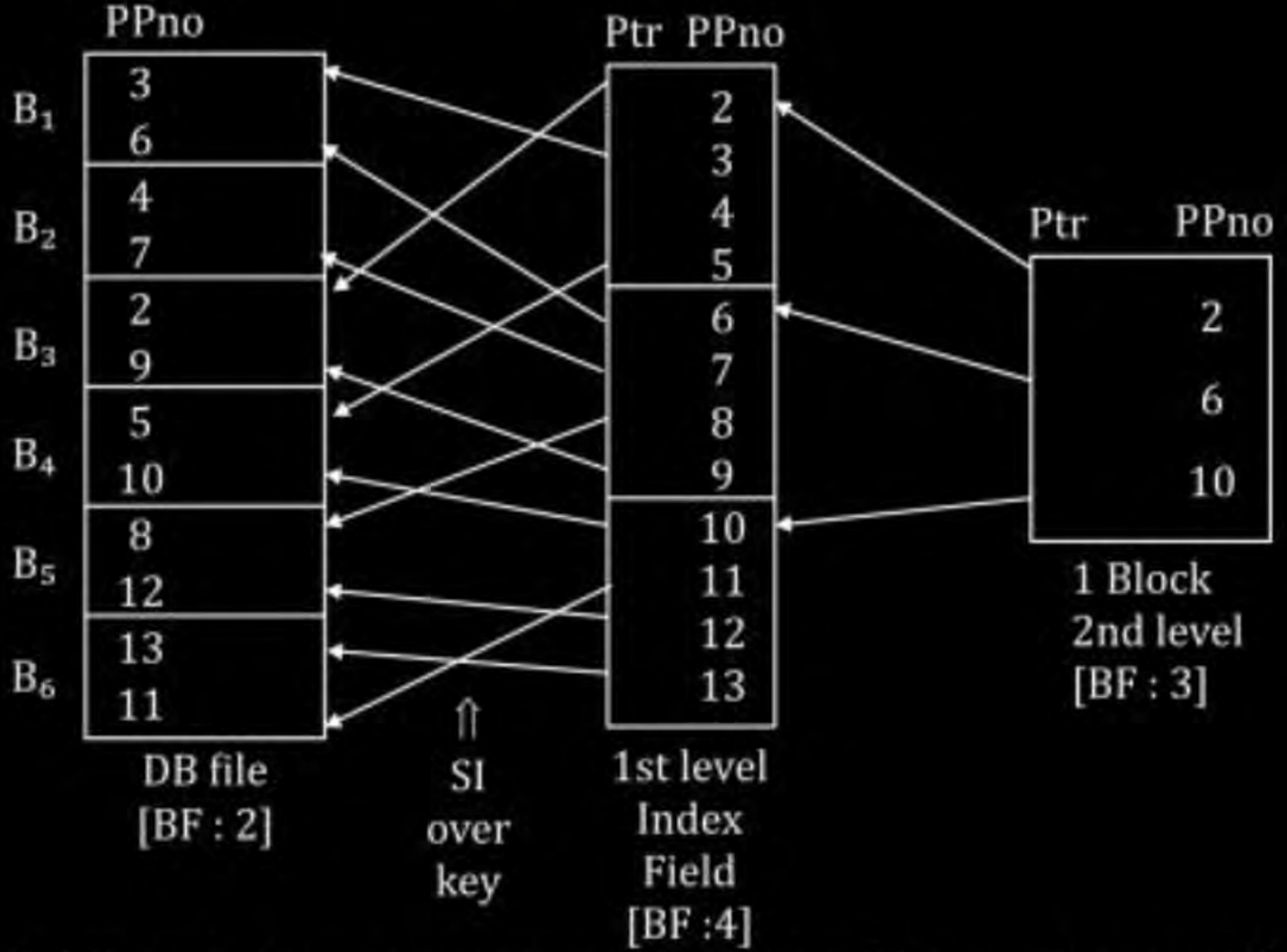


Secondary Index:

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

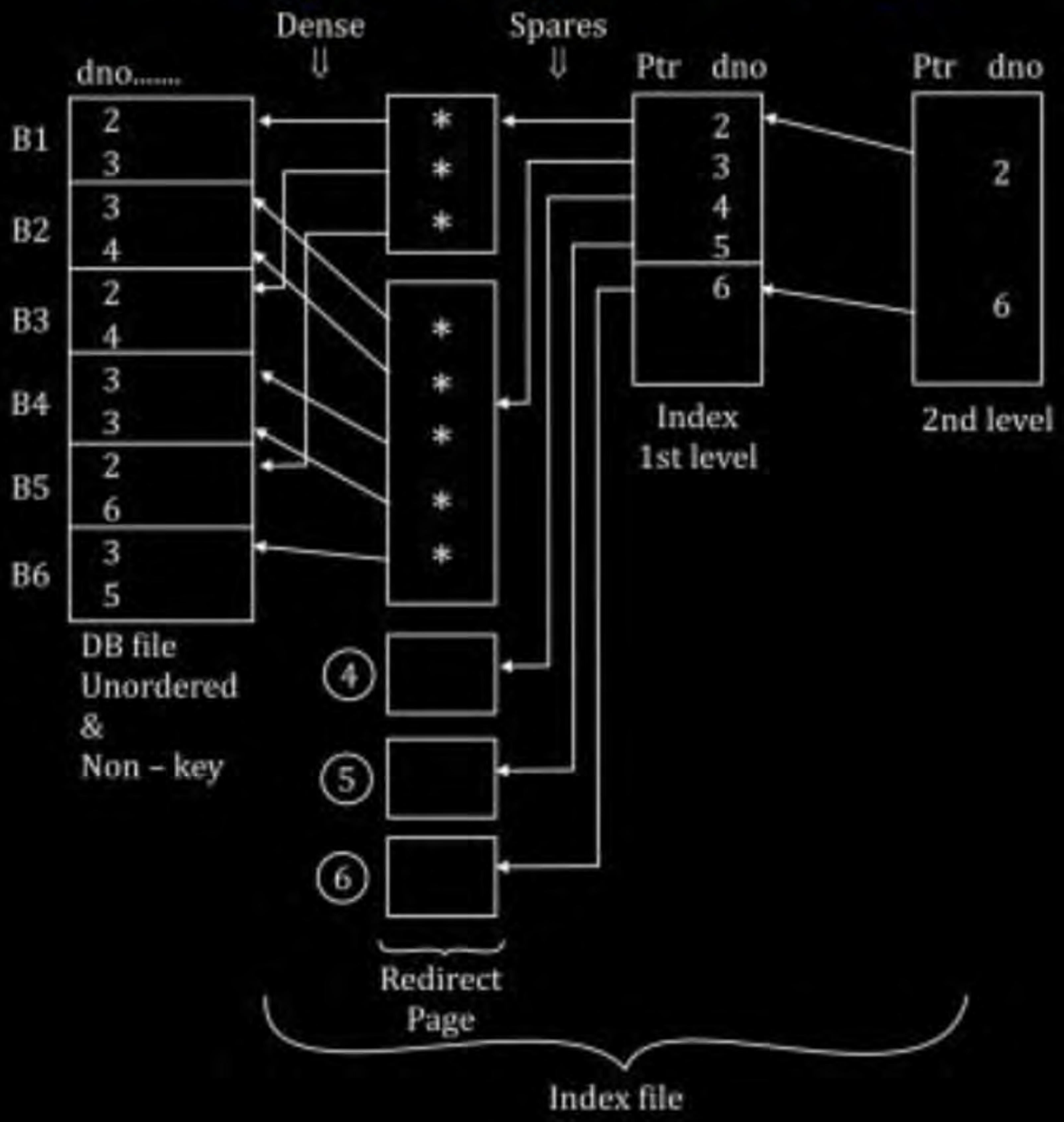


- Secondary Index (over key):



- 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

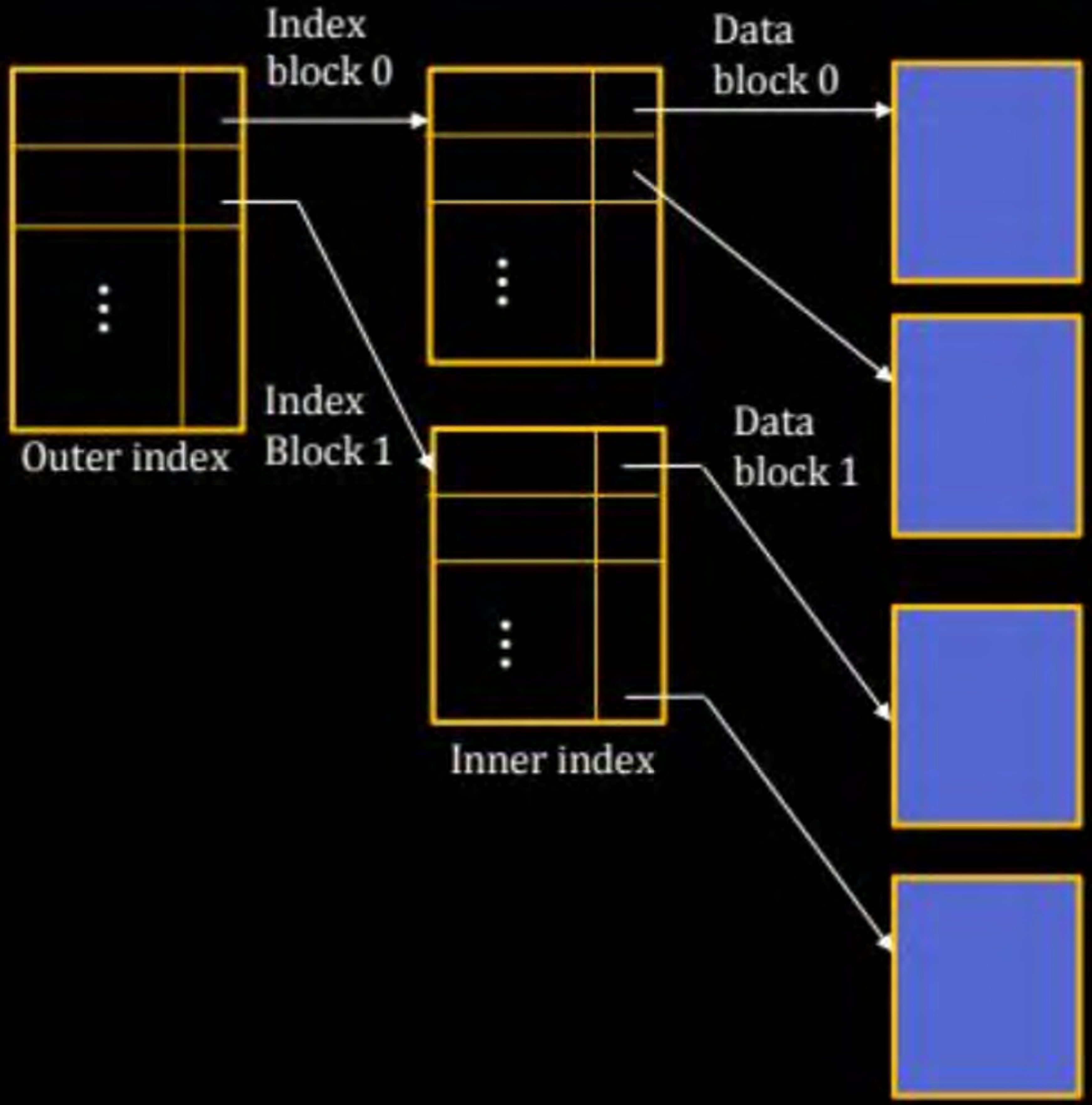
Unspanned & Unordered.

Dense

Multilevel 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.

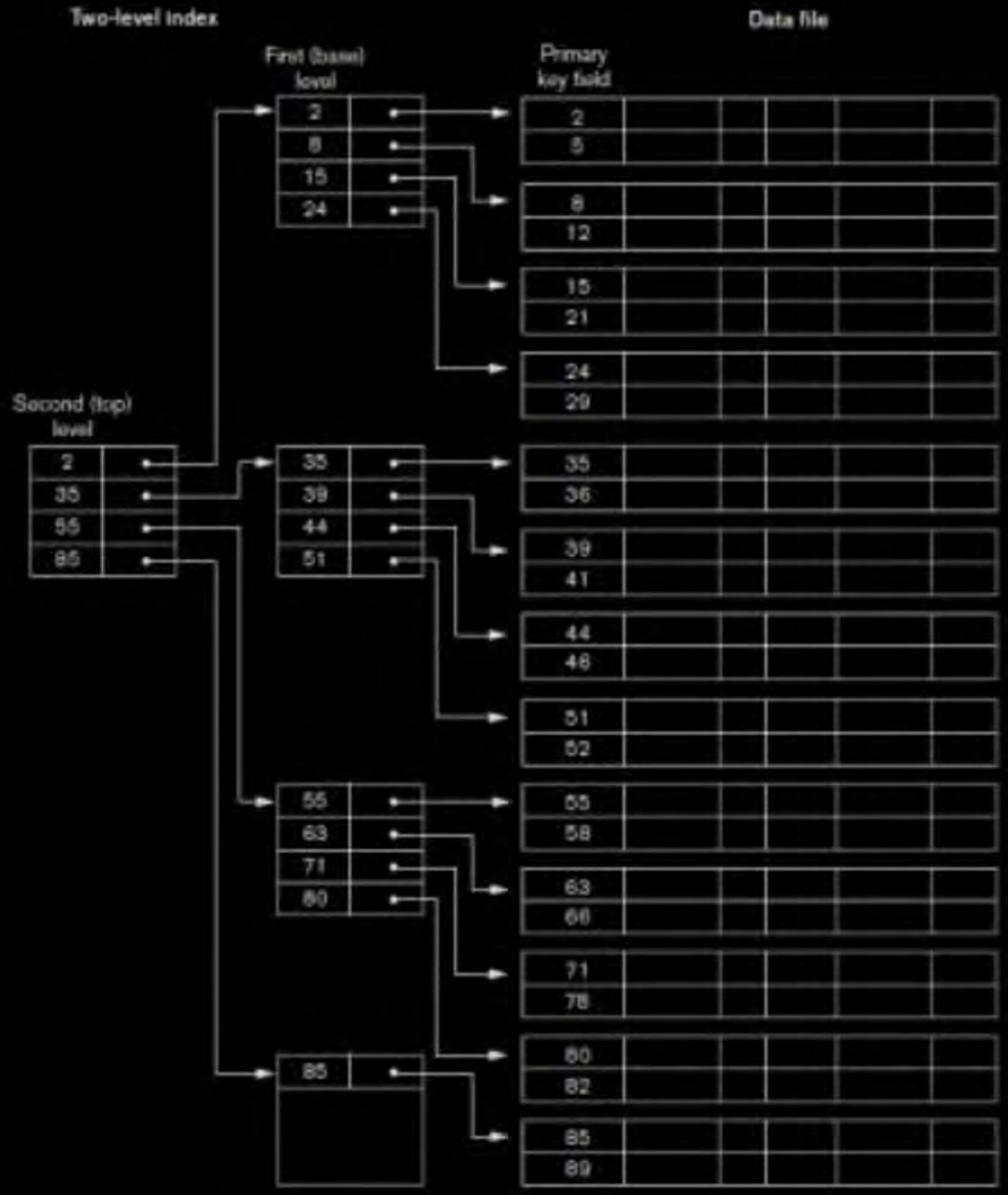
Multilevel Index (Contd..)



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

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.

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

I^{st} Level: Total number of Index Block = 442 Index Block

II^{nd} Level: number of Index Records (entries) = 442 (SPARSE number of I^{st} level block) & Block factor = 68 Index entries for block

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

III^{rd} Level: Number of Index Record = 7 (number of 2^{nd} level block)

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

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

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

THANK YOU!