COMPUTER SCIENCE

Database Management System

File Org. & Indexing

Lecture_1

Vijay Agarwal sir

01 File Structure

02 Indexing & its Type

Query Language

- RA
- SQL → 2nd Highest
- TRC

P.k & F.k

JOIN.

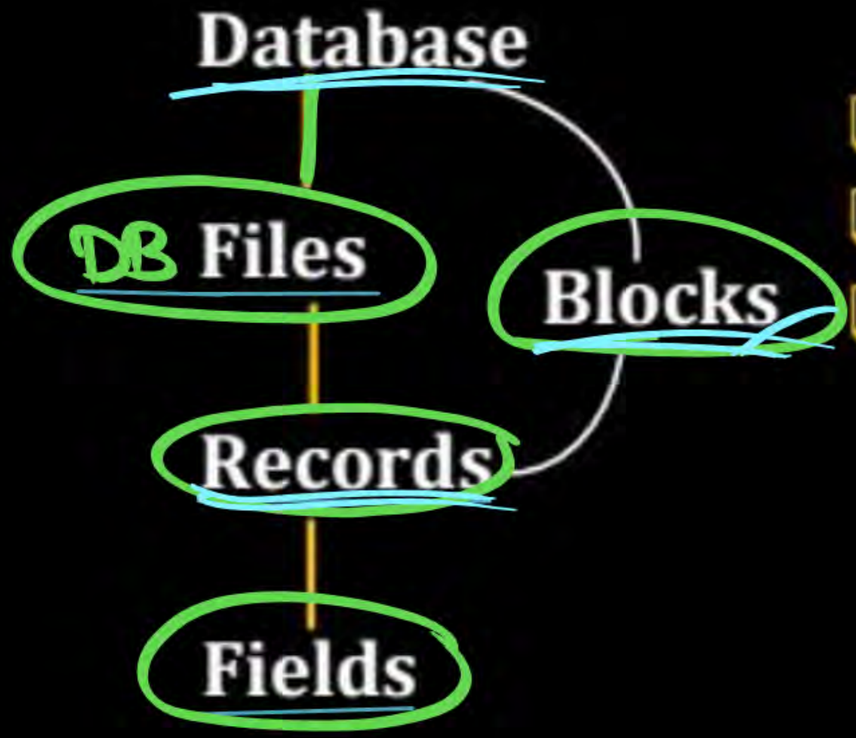
Minimum
Maximum

#Tuples

1. FD & Normalization
2. Transaction & Concurrency Control
3. ER Model & Foreign key Concept
4. Query language.
5. File Org & Indexing.

File Org & Indexing

File Structures & Indexing



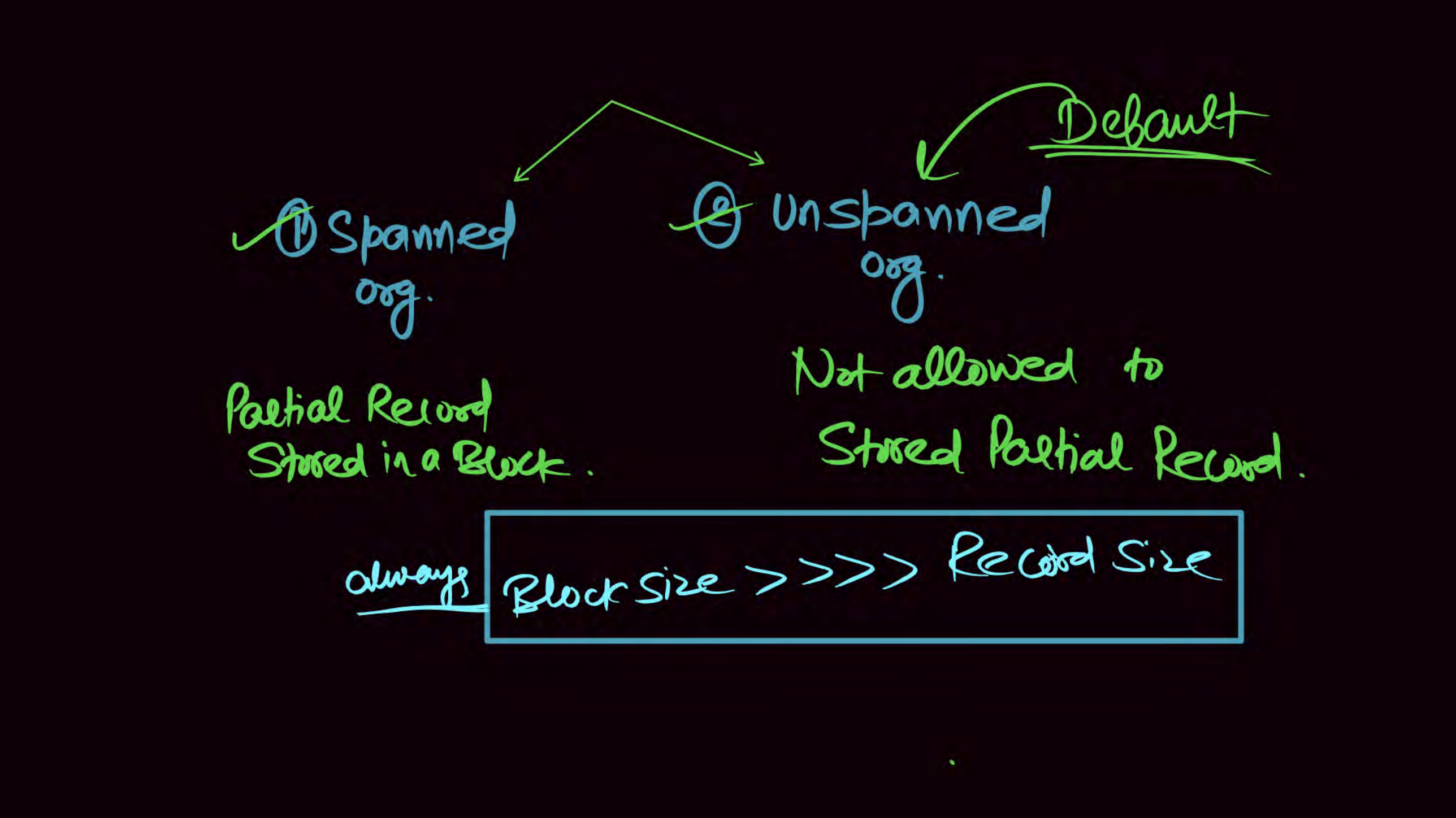
- Database is collection of files.
- Each file is a collection of Records.
- Each record is a sequence of fields.

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

Blocking factor : Average Number of Records Per Block.

Blocking factor = 4 [4 Records Per Block]

Blocking factor = 2 [2 Record Per Block]


Default
① Spanned org.
② Unspanned org.
Partial Record Stored in a Block.
Not allowed to Stored Partial Record.
always
Block Size >>>> Record Size

Record Blocking and Spanned Versus Unspanned Records

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

(1) Spanned organization :

· A Record Can be Stored More than One Block.

· It allow Partial Part of Record Stored in a Block

Advantage

No Wastage of Memory.

Disadvantage.

Number of Block Access Increase (for R_3 B_1 & B_2 Block Required)

Block Size = 100 Byte

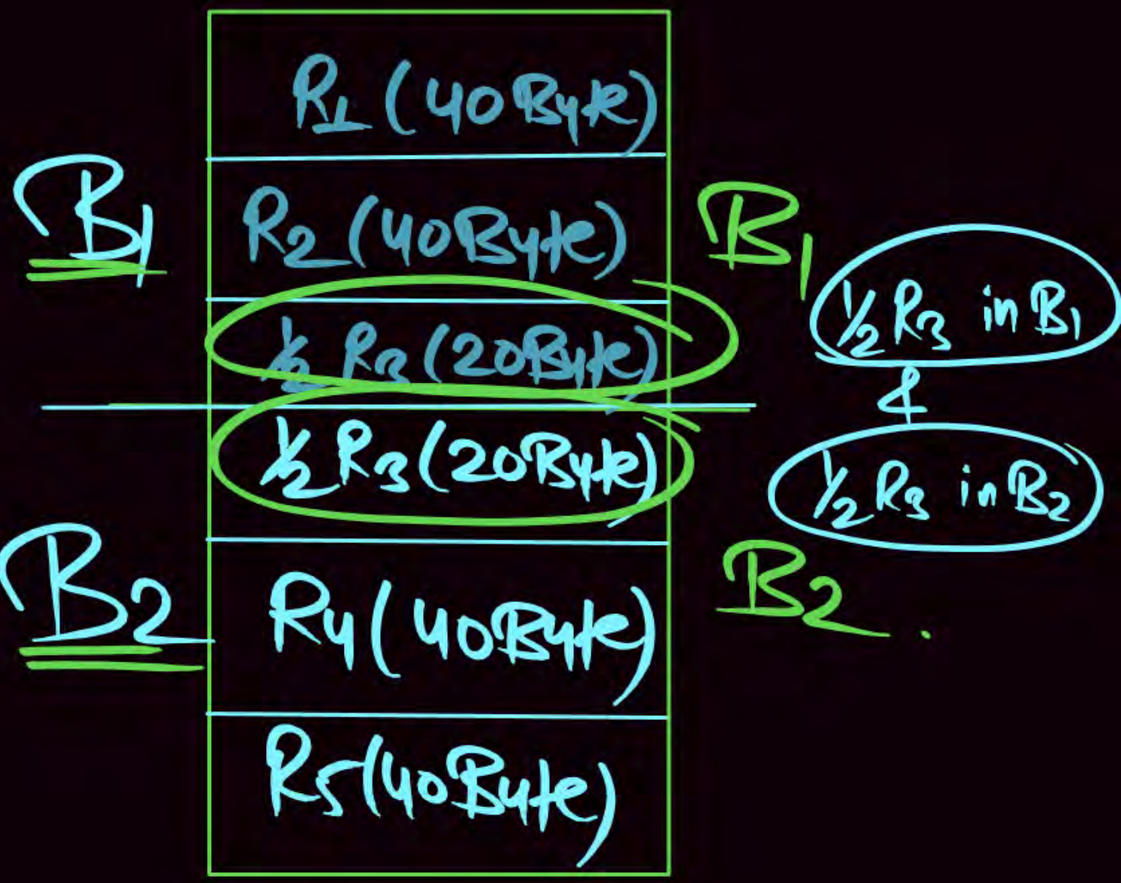
Record Size = 40 Byte.

$$\text{Blocking factor} = \frac{\text{Block Size}}{\text{Record Size}}$$

$$= \frac{100\,\text{Byte}}{40\,\text{Byte}}$$

$$Bf = 2.5$$

2.5 Records per Block



Record Blocking and Spanned Versus Unspanned Records

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

(2) Unspanned organization :

· A Records Can Stored in One Particular Block.
· It does not allow Partial Part of Record Stored in a Block.

Advantage

· Block Access Decrease.

Disadvantage

· Wastage of Memory (Internal Fragmentation).

Block Size = 100 Byte

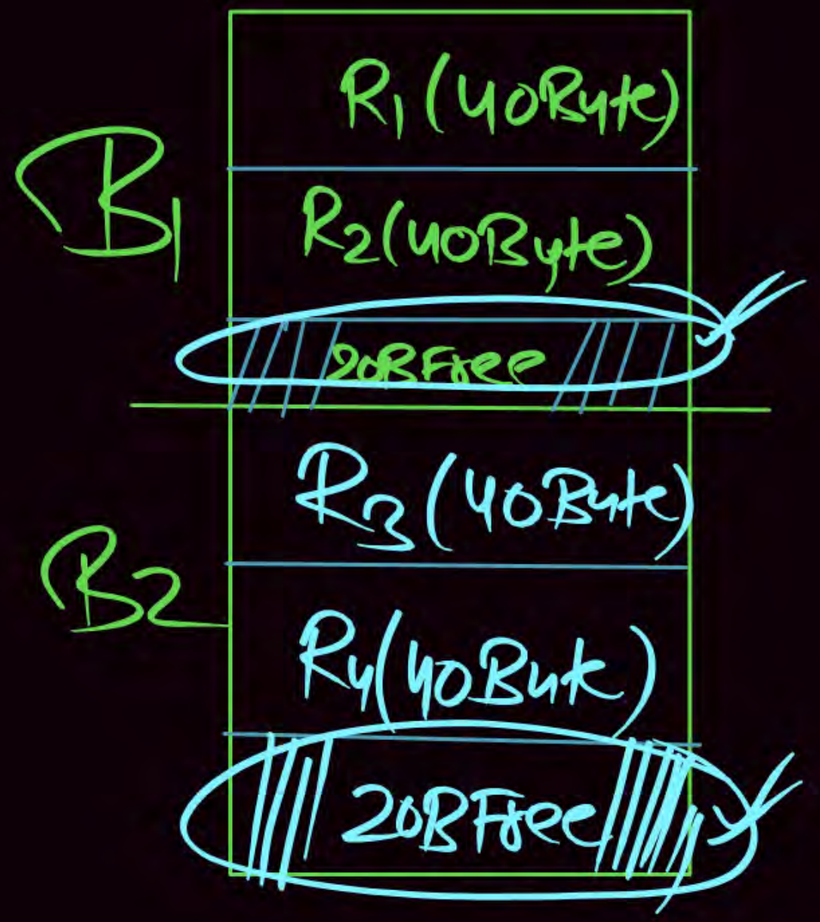
Record Size = 40 Byte.

$$\text{Blocking factor} = \left\lfloor \frac{\text{Block Size}}{\text{Record Size}} \right\rfloor$$

$$= \left\lfloor \frac{100\,\text{Byte}}{40\,\text{Byte}} \right\rfloor$$

$$= \lfloor 2.5 \rfloor$$

Blocking factor = 2 2 Record per Block



512 Byte

1024 Byte

1% to 2-1.

Wastage

2^{30} = 1GB (Giga Byte)

Storage

TB (Tera Byte)	2^{40}
PB (Peta Byte)	2^{50}
EB (Exa Byte)	
ZB	
YB	

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.

Note

Default organization is unspanned organization.

Floor & Ceiling Concept

$\lfloor \rfloor$ Floor

$\lfloor 2.0 \rfloor = 2$

$\lfloor 3.0 \rfloor = 3$

$\lfloor 2.9 \rfloor = 2$

$\lfloor 2.8 \rfloor = 2$

$\lfloor 2.7 \rfloor = 2$

$\lfloor 2.6 \rfloor = 2$

$\lfloor 2.1 \rfloor = 2$

Ceiling $\lceil \rceil$

$\lceil 2.1 \rceil = 3$

$\lceil 2.2 \rceil = 3$

$\lceil 2.3 \rceil = 3$

$\lceil 2.4 \rceil = 3$

$\lceil 2.5 \rceil = 3$

$\lceil 2.9 \rceil = 3$

$\lceil 3.07 \rceil = 3$

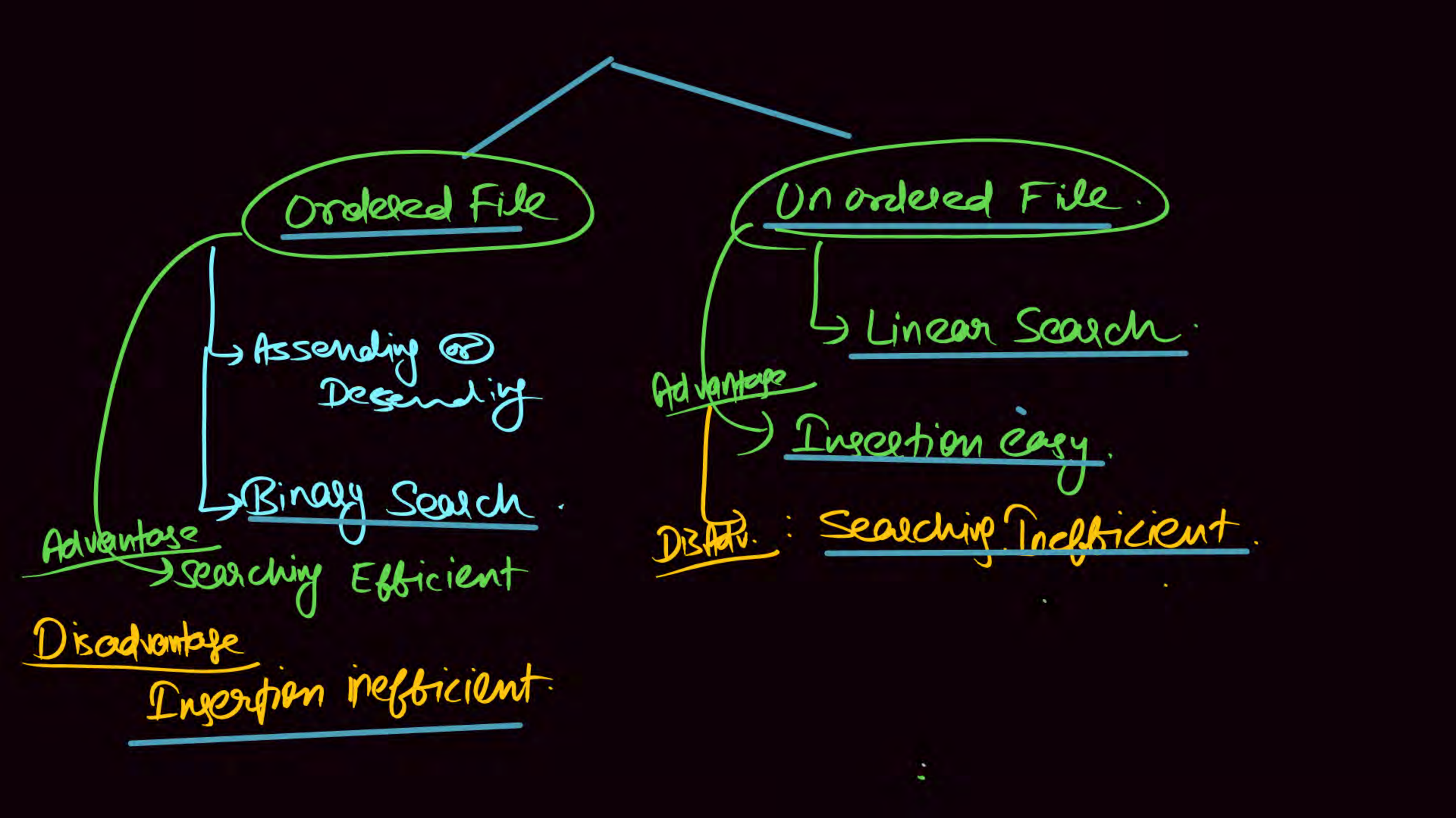
Note:

① I/O Cost: Input Output cost means Number of Blocks transferred from secondary memory to Main memory in order to access some records.

② Search Key: Attribute used to access the Data from DB

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

(2) Unordered file organization


Ordered File
Ascending or Descending
Binary Search
Advantage
Searching Efficient
Disadvantage
Insertion inefficient
Unordered File
Linear Search
Advantage
Insertion easy
Disadv.
Searching Inefficient

Ordered File

→ Binary Search.

To Access a Record
Avg No. of Block Access $= \lceil \log_2 B \rceil$

B: Number of DB Block
(Data Block)

Unordered File

→ Linear Search.

Avg Case $= B/2$

Worst Case $= B$

B: DB Block

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 org.)

- 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

Worst Case = B

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	~~$b/2$~~ $\log_2 b$
Ordered	Binary search	$\log_2 b$

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

Indexing

- To Improve the searching efficiency
- To Record the I/O Cost.

Block size of Index File is same as Block size of DB File.

One Index Record contain (search key) & (Pointer (BP.))

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

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".

V.V.Imp.

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]

Revision

Blocking factor.

① Spanned org

② Unspanned org

① ORDERED File

↳ $\log_2 B$

② Unordered (Heap) file

↳ Avg Case = $B/2$

Worst Case = B

Indexing

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

Index is an order file.

Block Size of Index file	=	Block size of DB File

To Access a Record using Index Average No. of Block Access $= \log_2 B_i + 1$

Example:

- Suppose that:
  - record size R = 150 bytes, block size B = 512 bytes,
  - r = 30000 records
- Then, we get:
  - blocking factor Bfr = $\lfloor \frac{\text{Block size}}{\text{Record size}} \rfloor \Rightarrow \lfloor \frac{512B}{150B} \rfloor$ = 3 Records Per Block
  - number of file blocks b = $\frac{30,000}{3}$ = 10000 DATA Block

b = 10,000 Block. Ans

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:

With Indexing

Example:

$938 \le 2^n$

$n=9 \Rightarrow 2^9 = 512$

$n=10 \quad 2^{10} = 1024$ ✓

Given the following data file

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

Block size of DB File ≡ Block size of Index file.

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:

Q.1 ❖ index entry size $R_1=(V_{SSN}+P_R)=$ 9+7 = 16 Byte

Q.2 ❖ index blocking factor $Bfr_1=B \text{ div } R_1=\left\lfloor \frac{512 \text{ Byte}}{16 \text{ Byte}} \right\rfloor$ entries / block = 32 Index Record Per Block.

Q.3 ❖ number of index blocks (Dense) $b=\lceil r/Bfr_1 \rceil =$ = blocks

Q.4 ❖ binary search (Block) needs $\log_2 bI = \log_2 938 =$ 10 block accesses

To Access a Record Using Index Alg. No. of Block Access = $\log_2 Bi + 1 \Rightarrow \log_2 938 + 1$

$= 10 + 1 = 11$ Ans

Fetch

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:

Soln

(i) index entry size $R_1=(V_{SSN}+P_R)=(9+7)=16$ bytes Ans

(ii) index blocking factor $Bfr_1=B \text{ div } R_1=512 \text{ div } 16=32$ entries / block Ans

(iii) number of index blocks $b=(r/Bfr_1)=(30000/32)=938$ blocks

(iv) binary search needs $\log_2 bI=\log_2 938$ =10 block accesses

Category of Index

1. SPARSE Index : Index entries is created for some search key value/some Record.
2. DENSE Index : Index entries is created for each/every Record.

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

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

#Records = 30,000

#Blocks = 10,000

If Dense Index

Total Number of Index entries = 30,000 (#Records)

② SPARSE Index

Total Number of Index entries = 10,000 (#DB Blocks)

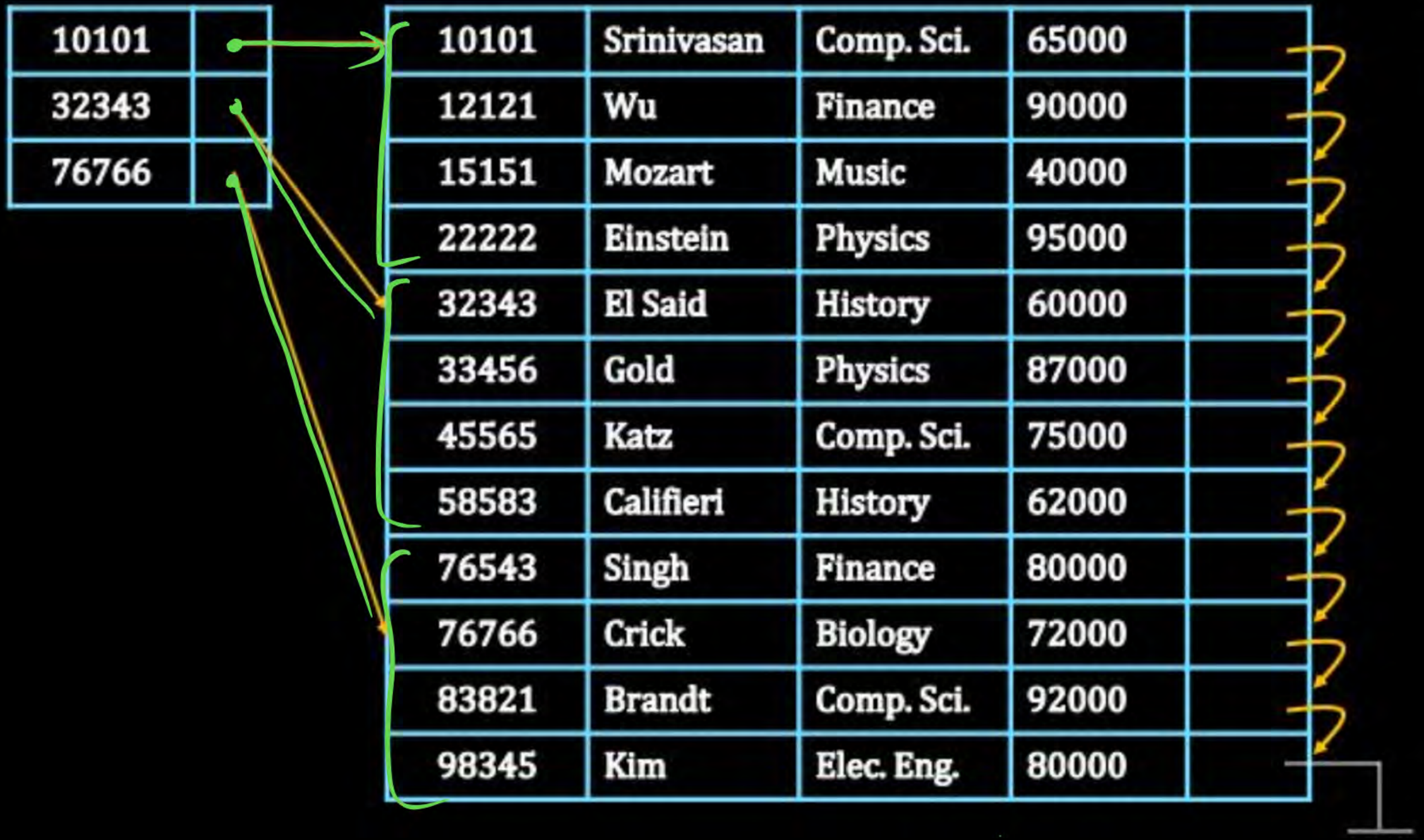
Dense Index Files

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

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

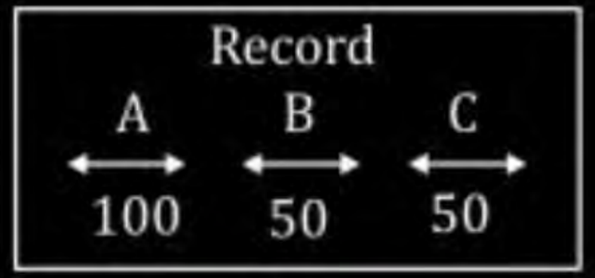
Dynamic multilevel indexes Using B-Tress and B^+ Trees.

Fixed length Record

Variable length Record.

File Organization:

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



≈ 200 bytes

1) Fixed Length Records

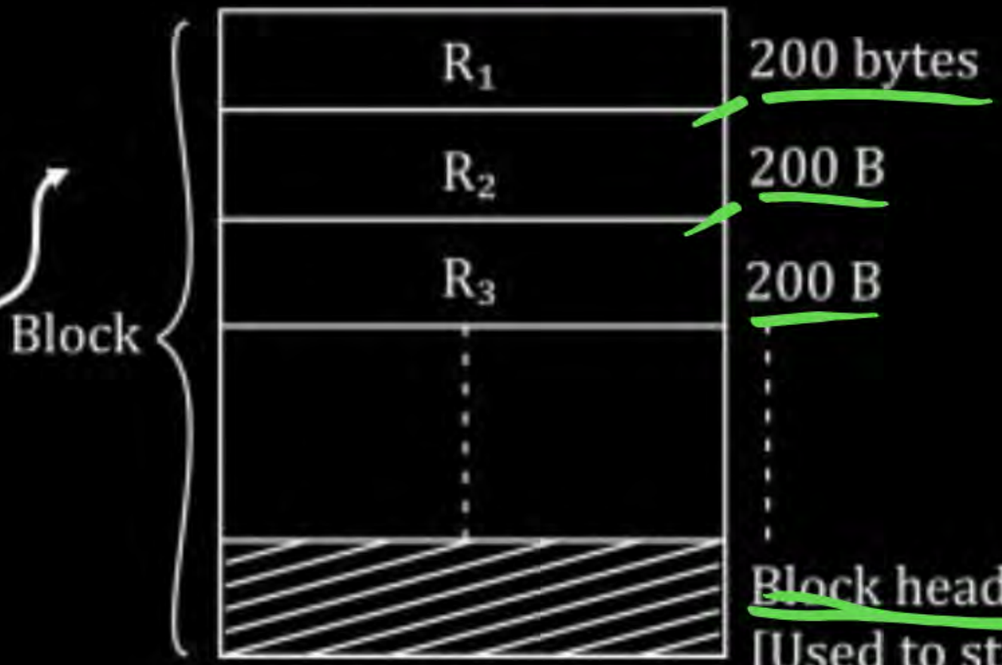
Create table R

(A char (100),

B char (50),

C char (50));

200 Byte



File Organization:

100 + 50 + (Variable) Text

2) Variable Length Records

Create table S

(D char (100),

E char (50),

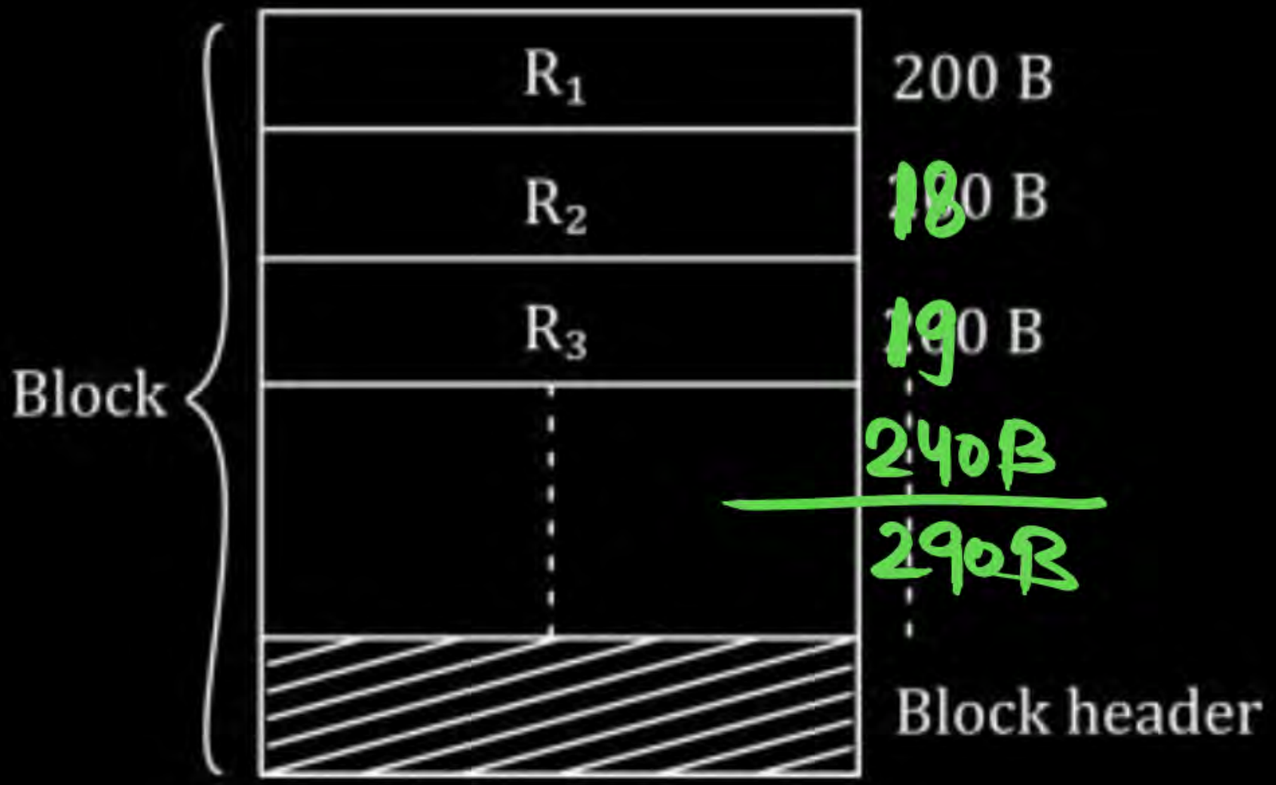
F text

);

∴ size (D) + Size (E) + size(F)

100 50 not fixed

n(variable)



File Organization:

- DB File with all records fixed length
- DB file with variable length records

$\Rightarrow$ 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

By Default H = 0Byte

Advantage:

⇒ Possible to allocate file with no internal fragmentation.

Block Header = H

$$B.F = \frac{\text{Block Size}}{\text{Record Size}}$$

$$\frac{B-H}{R}$$

Record Organization:

(a) Spanned Organization

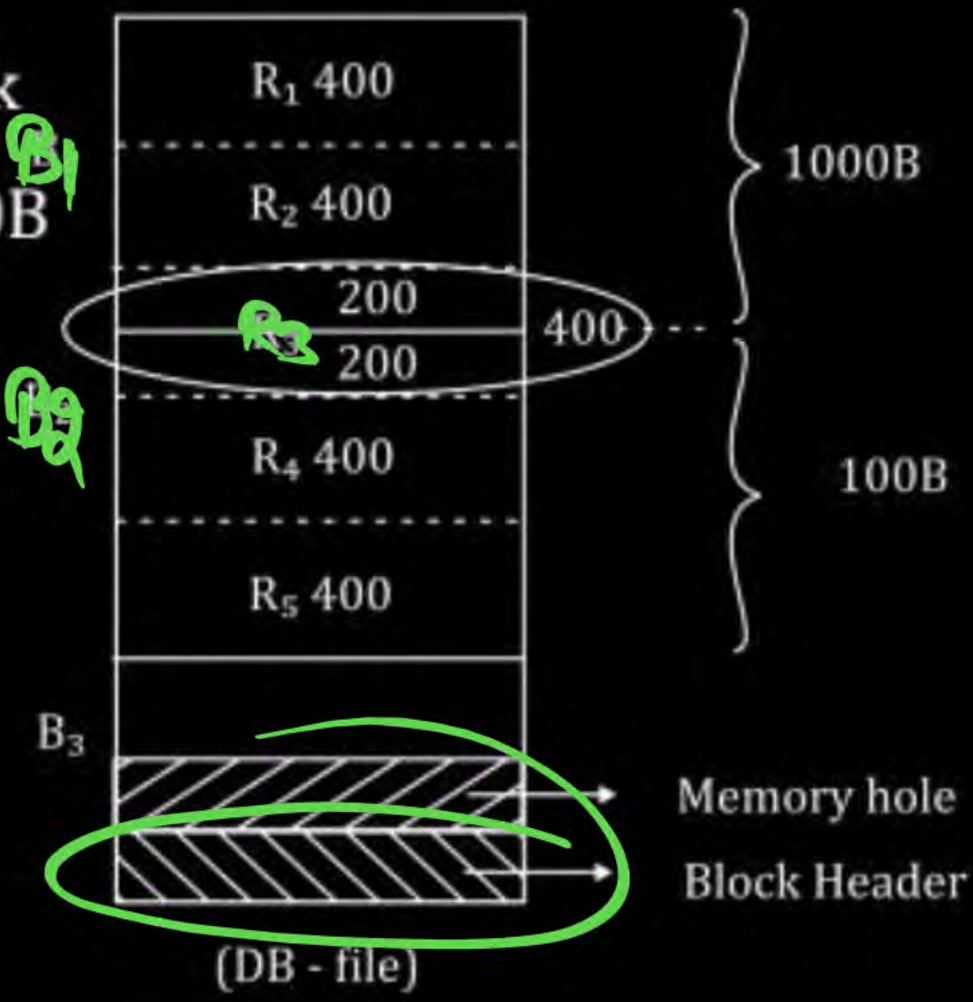
Record allow to span in more than one block

Example: Block size 1000B; Record size 400B

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

⇒ 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

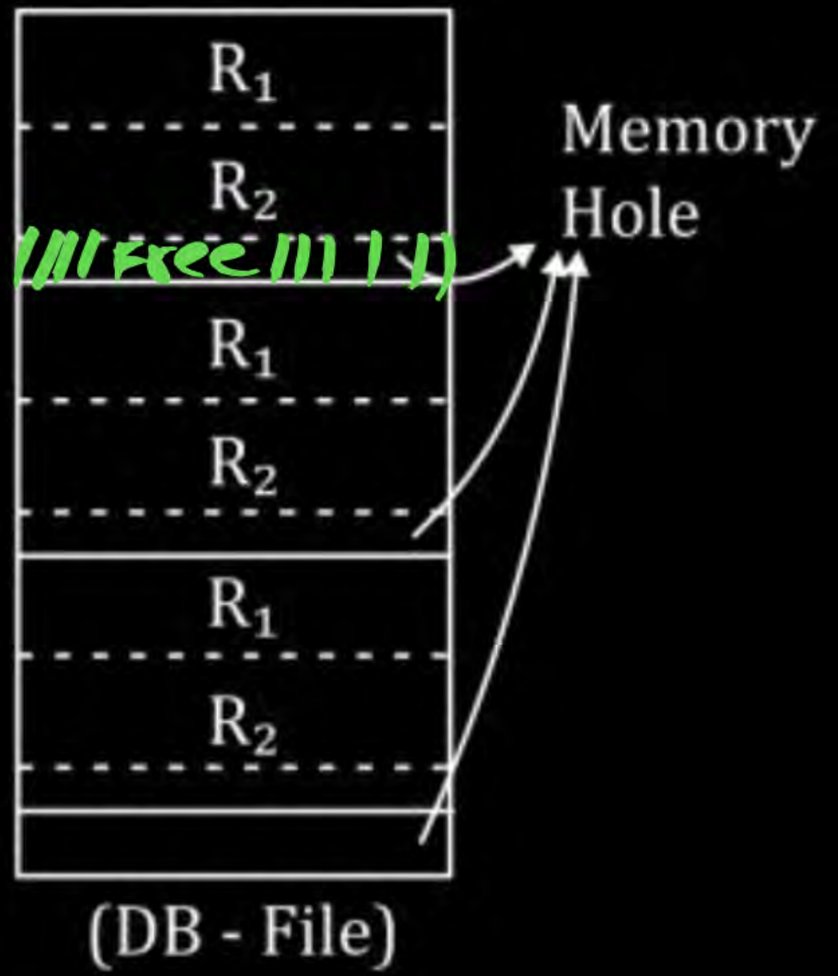
Complete Record must be stored in one Block.

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.



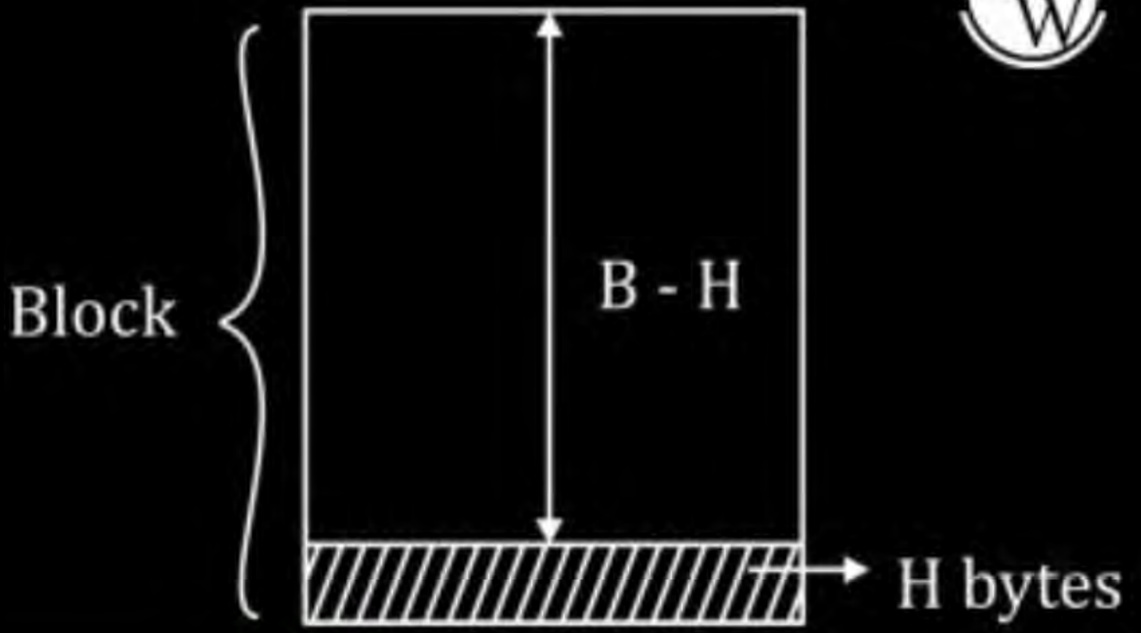
Block Factor

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.

By default Unspanned org Poefered [Fixed Length Record]

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

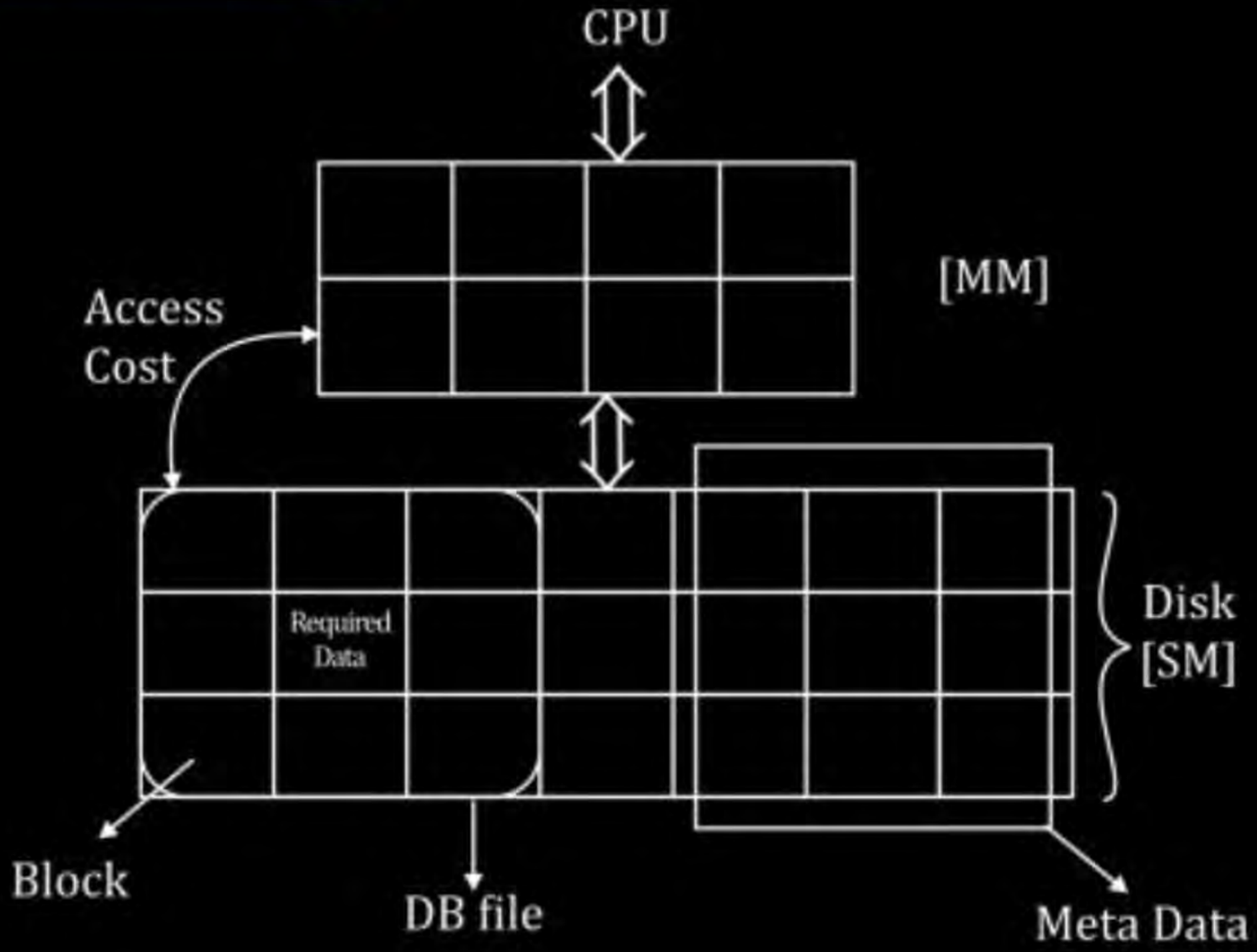
[Bi : Index Block]

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

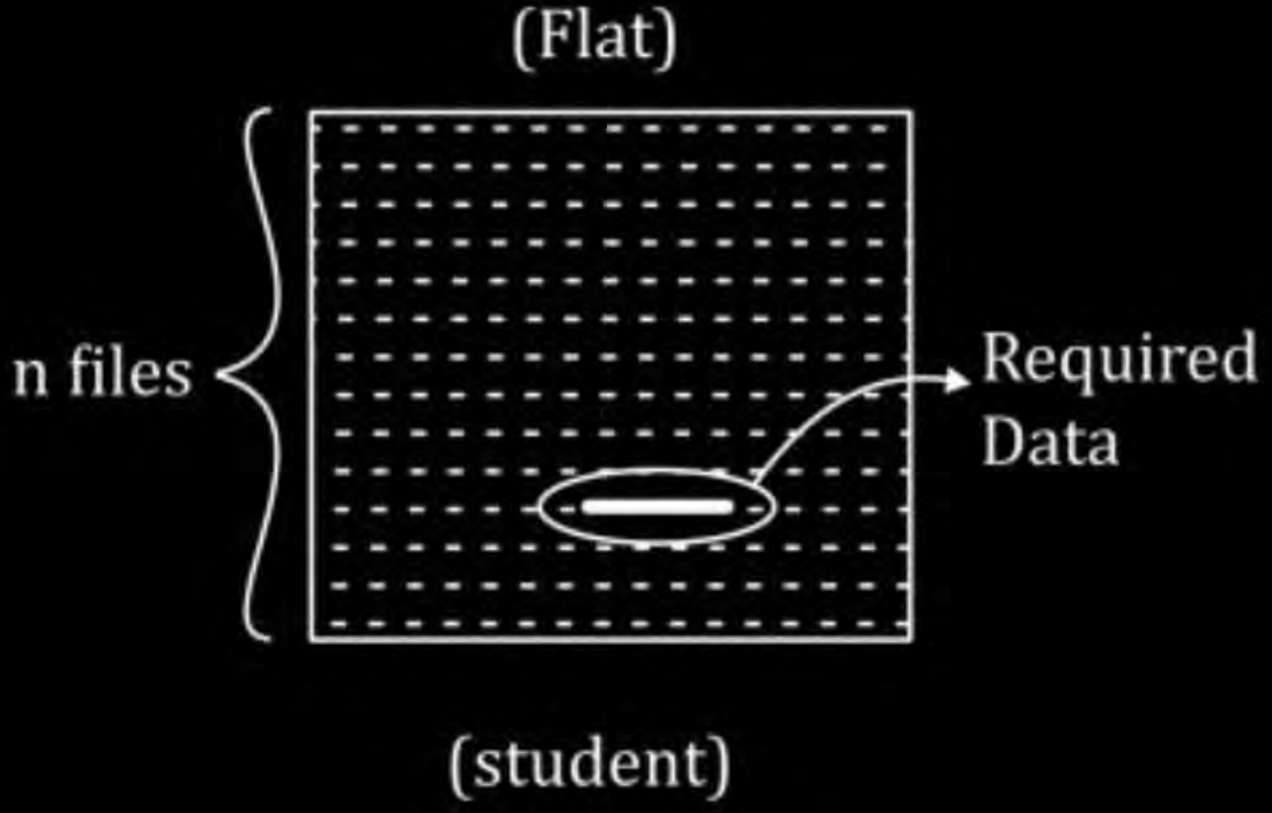
→ Number of record in file

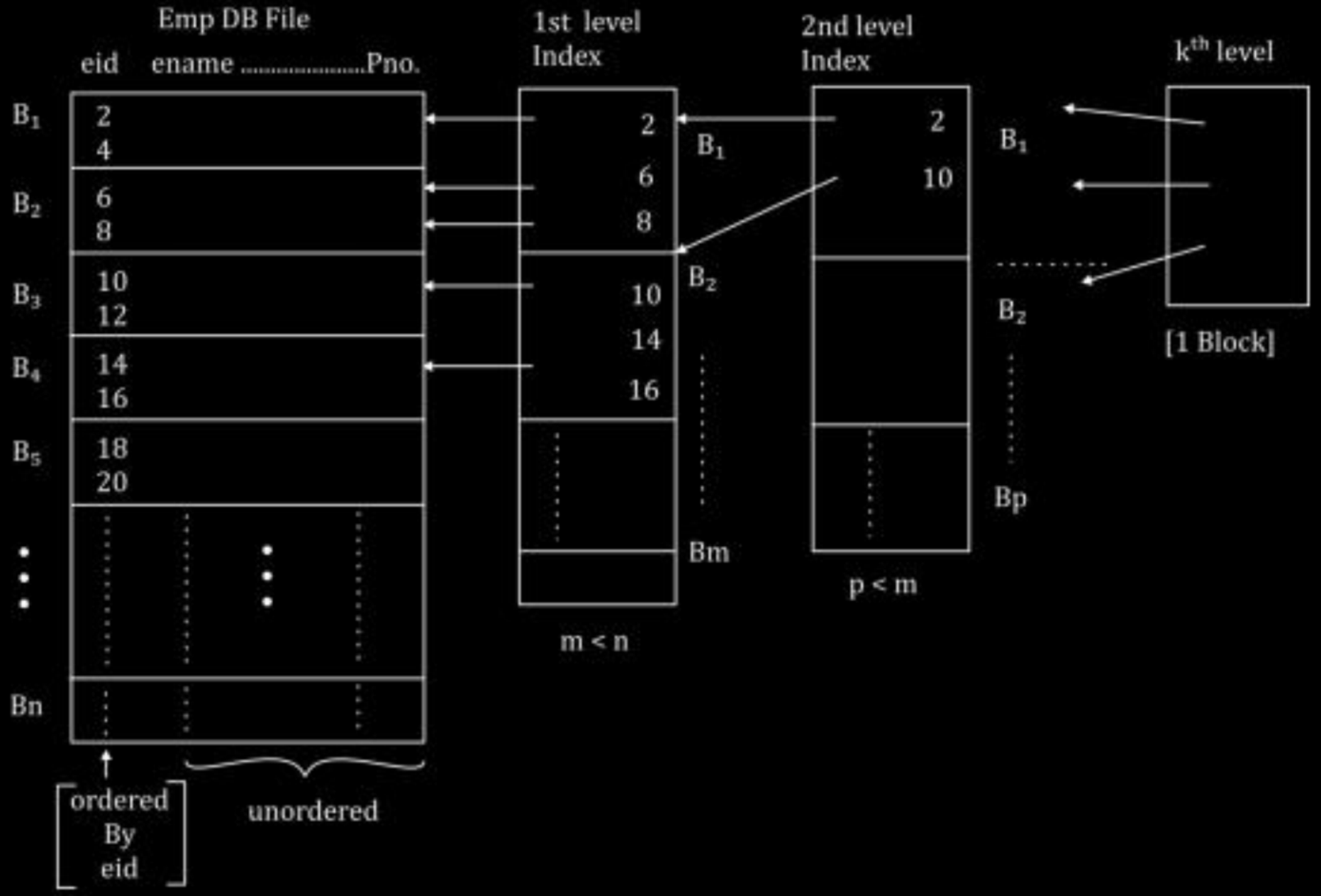
→ Ordered field of file, etc

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.
1st level Index
2nd level Index
k^th level
B1
B2
B3
B4
B5
Bn
2
4
6
8
10
12
14
16
18
20
2
6
8
10
14
16
B1
B2
Bm
m < n
2
10
B1
B2
Bp
p < m
[1 Block]
ordered By eid
unordered

Index File:

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

Block Size / Index Record Size

$\left\lfloor \frac{B-H}{\text{Key + Pointer}} \right\rfloor$

Index File

k_1	p_1
k_2	p_2
⋮	⋮

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

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

# of DB File block $\geq$ # of Index blocks

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

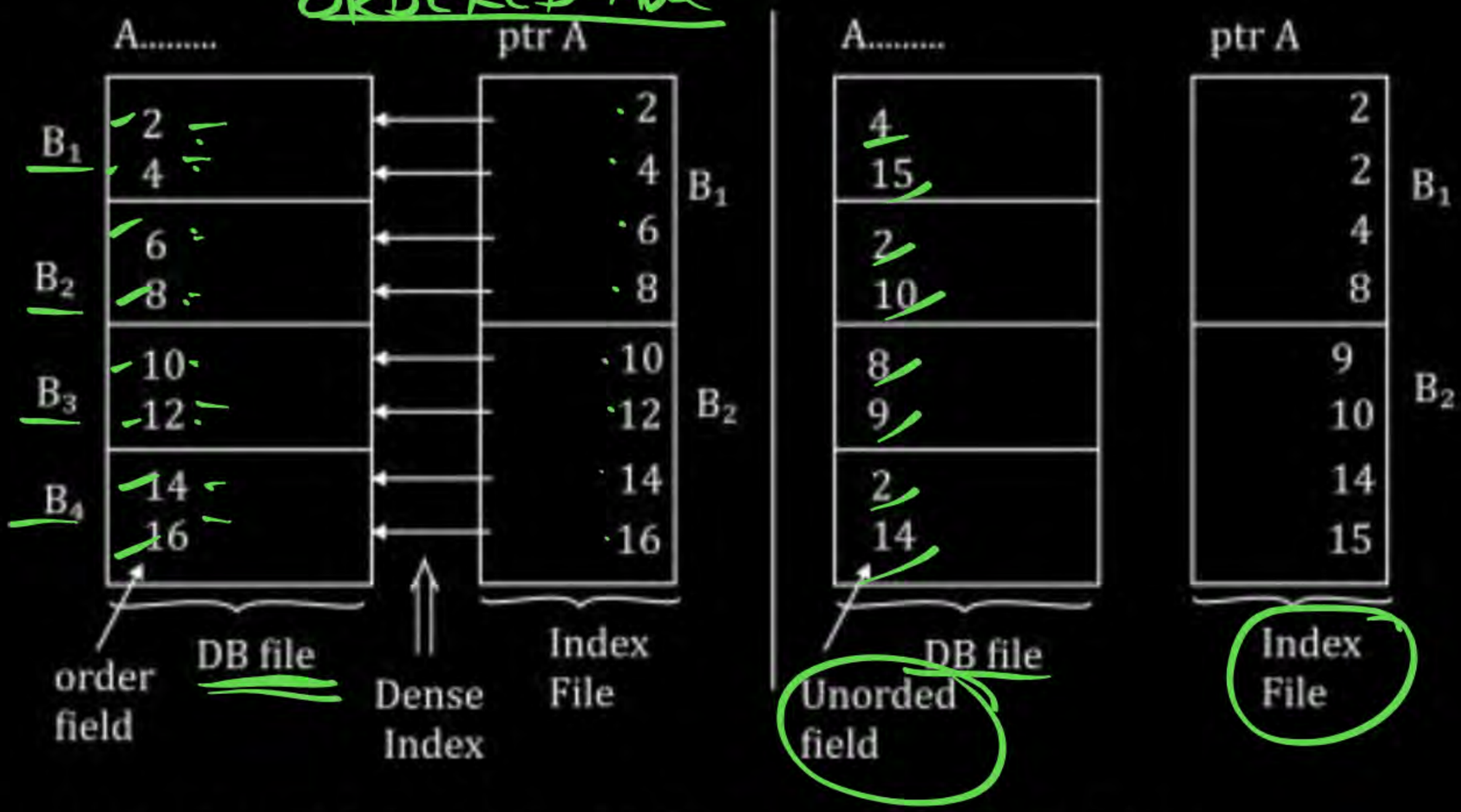
10101	10101	Srinivasan	Comp. Sci.	65000	
12121	12121	Wu	Finance	90000	
15151	15151	Mozart	Music	40000	
22222	22222	Einstein	Physics	95000	
32343	32343	El Said	History	60000	
33456	33456	Gold	Physics	87000	
45565	45565	Katz	Comp. Sci.	75000	
58583	58583	Califieri	History	62000	
76543	76543	Singh	Finance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

Dense Index → Ordered file
→ Unordered file.

Dense Index [More entries in Index File]

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

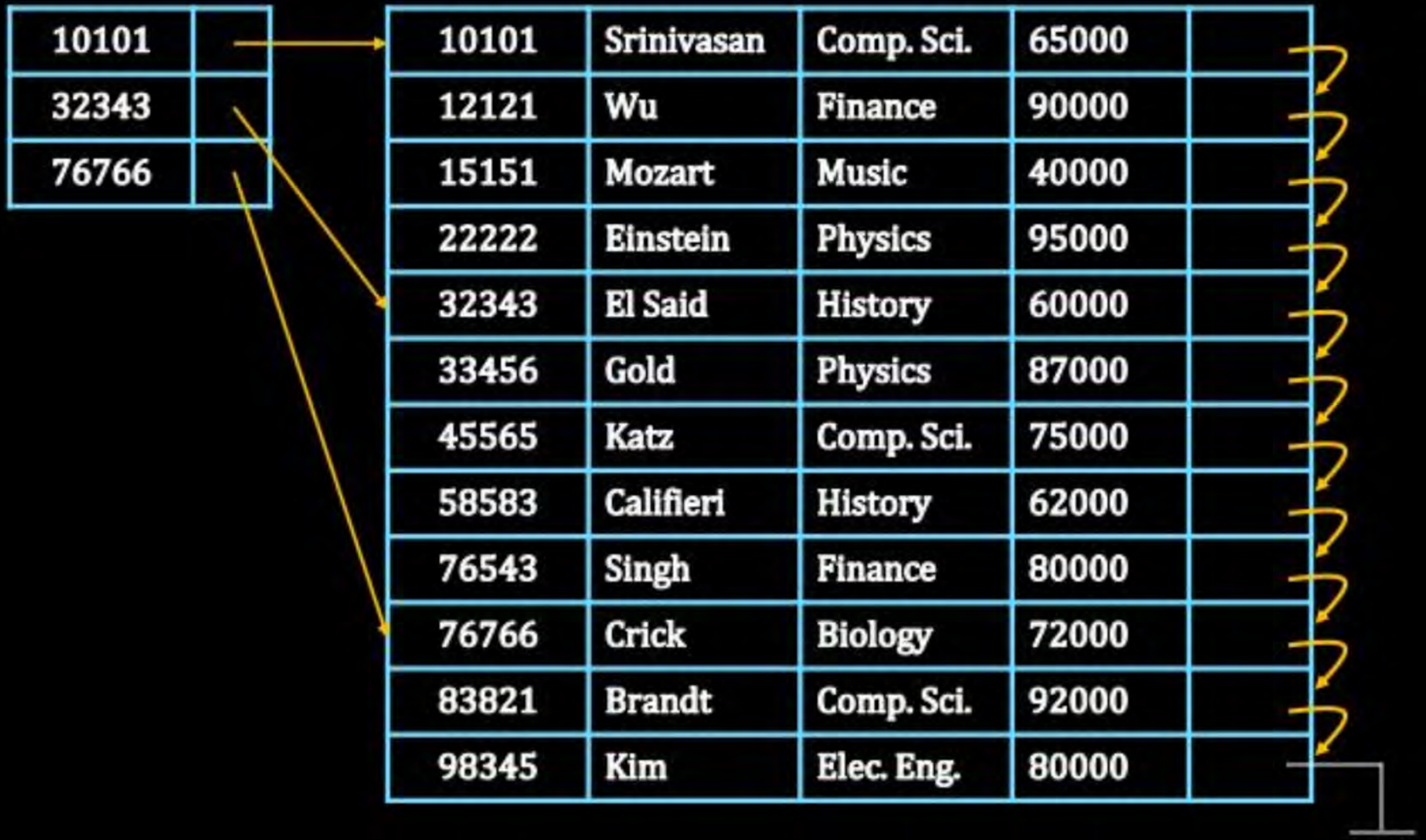
[Dense Index]



⇒ (# 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
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]

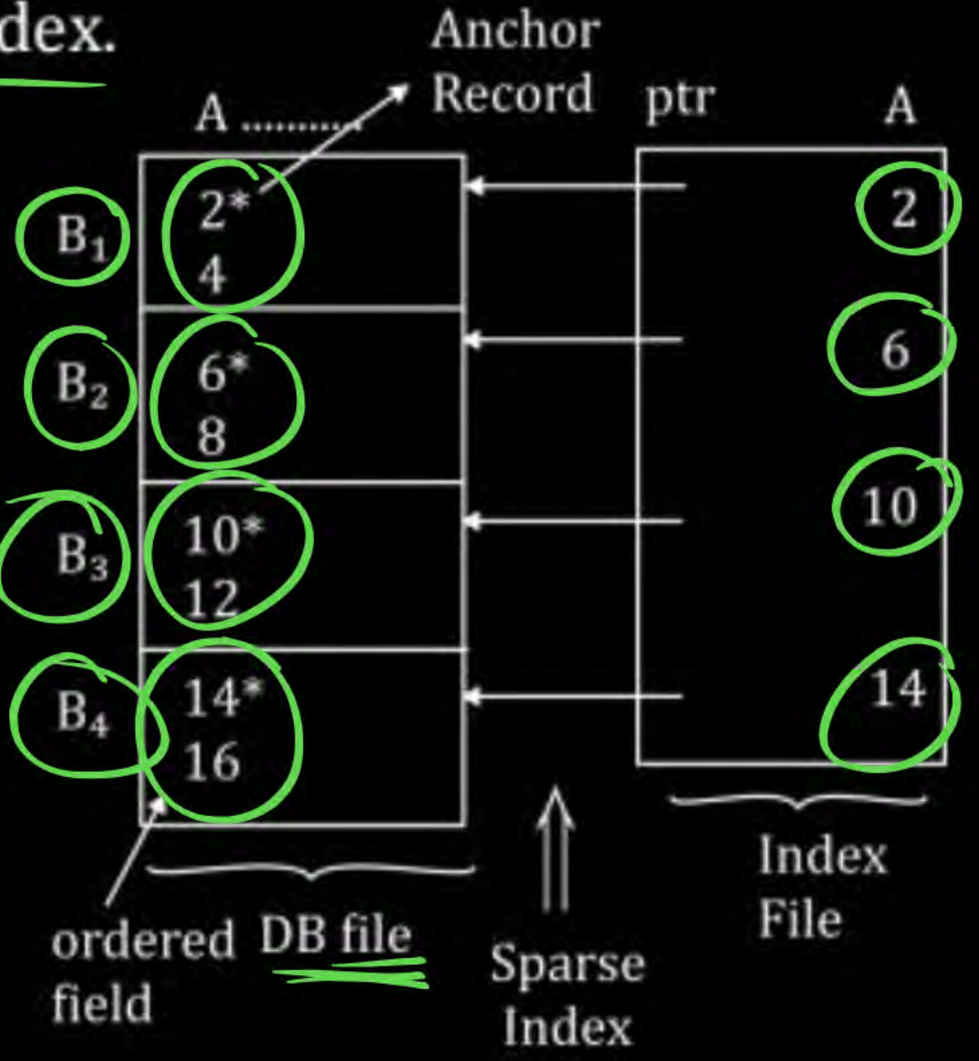
⇒ For set of DB records there exist entry in Index file such a Index is called Sparse Index.

⇒ 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]

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



Index File:

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

Index File

k_1	p_1
k_2	p_2
⋮	⋮

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

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

{# of DB File block} > {# of Index blocks}

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

Dense Index

#Entries = 30000 (#Record) Ans

SPARSE Index

#Entries = 10,000

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:

Dense

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 $Bfr_1=B \text{ div } R_1=512 \text{ div} 16=32$ entries / block
  - number of index blocks $b=(r/Bfr_1)=(30000/32)= 938$ blocks (Dense.)
  - 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} B]

Record size: 256 Bytes [2^8 B]

Search key size: 22 Bytes

Pointer : 10 Bytes

Index Record Size = 32 B [2^5]

(Home work)

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

(a) Ordered field = $\lceil \log_2 1024 \rceil = 10$ Ans

(b) Unordered Filed = Avg = $B/2 = 512$ Worst Case = 1024

⇒ 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] =

#Record = 16,384 $[2^{14}]$ Block Size = 4096 B $[2^{12}_{B}]$ Record Size = 256 B $[2^{8}_{Byte}]$

Index Record Size = 32 B $[2^{5}_{Byte}]$

Without Index

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

Total #Records = 16384 $[2^{14}]$

Total # Data Block [B] = $\frac{2^{14}}{16} = \frac{2^{14}}{2^{4}} \Rightarrow 2^{10} =$ 1024 Data Block

B = 1024

#Record = 16,384 $[2^{14}]$ Block Size = 4096 B $[2^{12}$ B$]$ Record Size = 256 B $[2^{8}$ Byte$]$

Index Record Size = 32 B $[2^{5}$ Byte$]$

With Index

Block factor of Index file = $\left\lfloor \frac{\text{Block Size}}{\text{One Index Record Size}} \right\rfloor = \frac{2^{12}}{2^5} = 2^7 \Rightarrow 128$ Index Record Per Block

Dense Index

Total # Index entries = 16,384 (# DB Records)

Block factor of Index file = 128 Index Record per Block

Total # Index Block = $\left\lceil \frac{16384}{128} \right\rceil = \frac{2^{14}}{2^7} = 2^7$

= 128 Index Block.

Ans

SPARSE Index

Total # Index entries = 1024 (# DB Block)

Block factor of Index file = 128 Index Record per Block.

Total # Index Block = $\left\lceil \frac{1024}{128} \right\rceil = \left\lceil \frac{2^{10}}{2^7} \right\rceil = 8$ Index Block.

Ans

Revision

① Spanned org

② Unspanned org.

---

① ORDERED File

↳ $\log_2 B$

② Unordered file

Avg = $B/2$ | worst Case = B

Indexing

Ordered file

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

Block Size of Index file = Block Size of DB File

To Access Index Block
Avg # of Block Access = $\log_2 B_i$

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

Dense Index

Total # of Index entries = # of DB Records

SPARSE Index

Total # of Index entries = # of DB Blocks

Types of Index

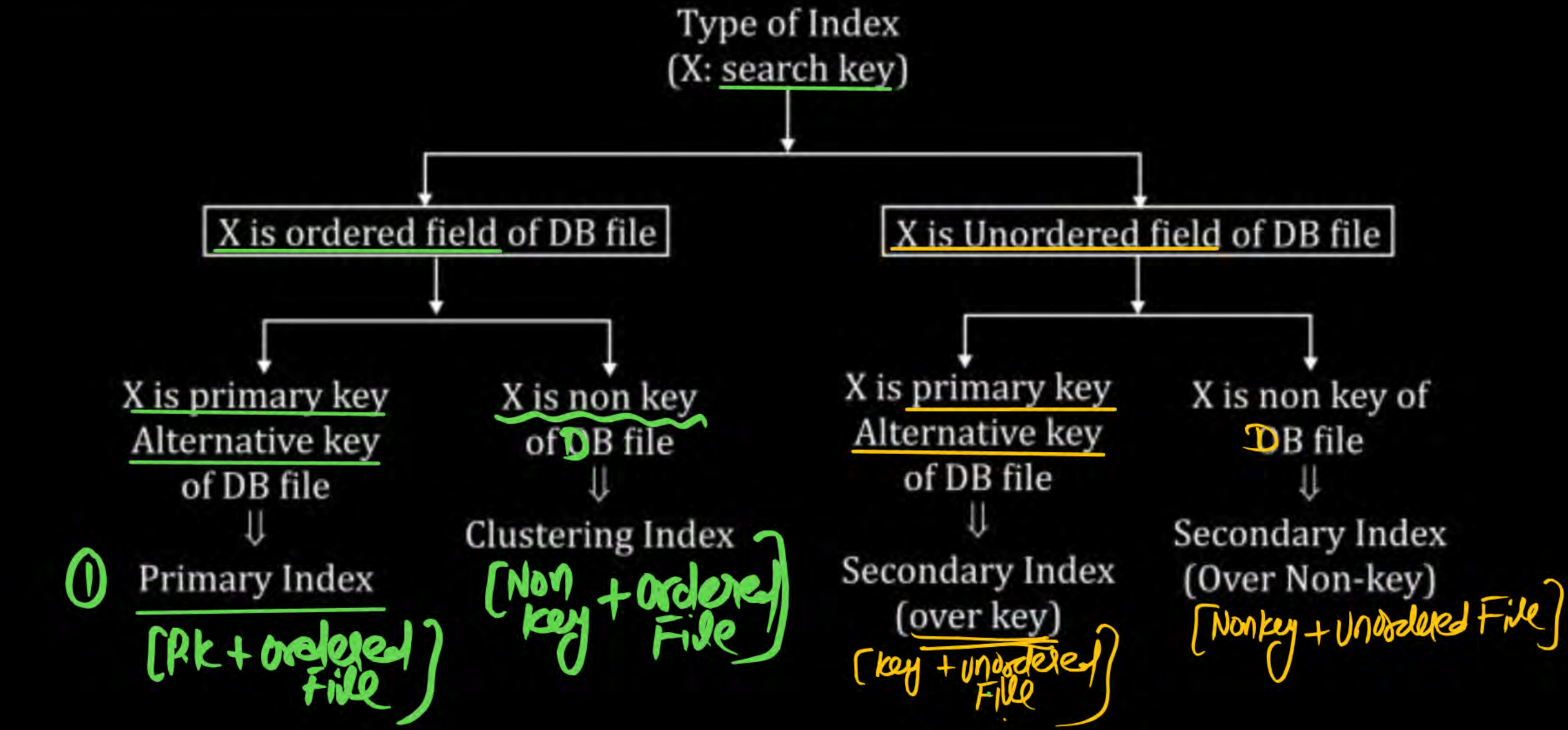
- Single-level Ordered Indexes
  - Primary indexes
  - Clustering indexes
  - Secondary indexes
- Multilevel Indexes
  - Dynamic multilevel indexes Using B-Tress and B^+ Trees.

① Primary Index (Search key / P.K [C.K] + Ordered File)

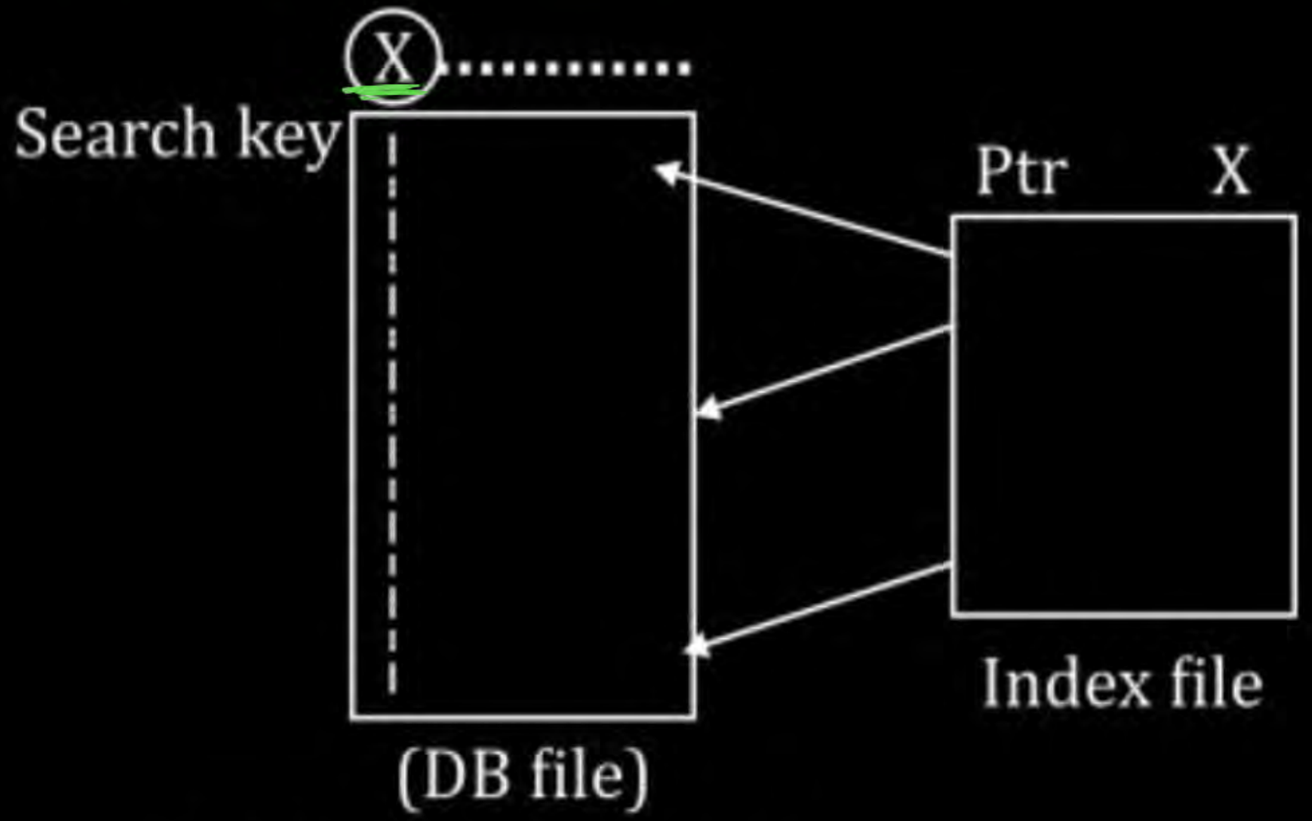
② Clustered Index (Non key + Ordered File)

③ Secondary Index [Candidate key / Non key + Unordered File]

Types of Index

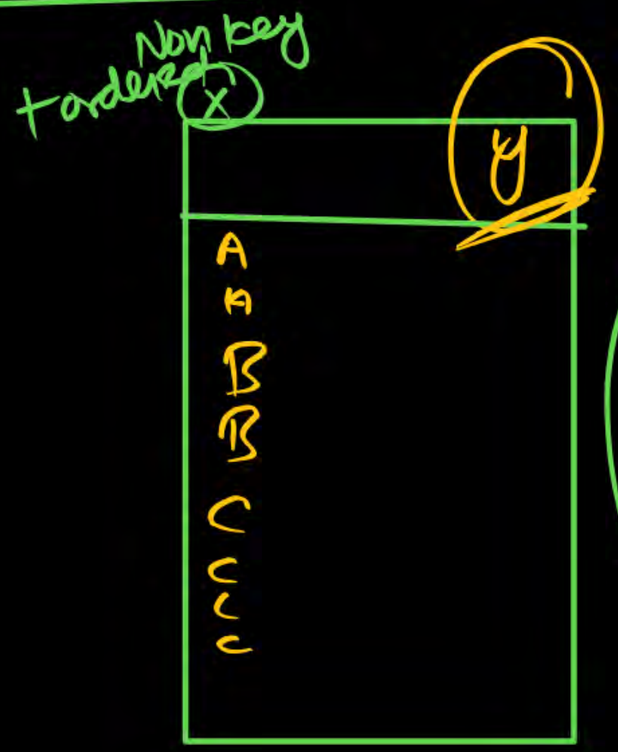


Types of Index



Types of Index

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



Key

Secondary Index over key

eg

Bank

A/c No ordered [key + ordered]

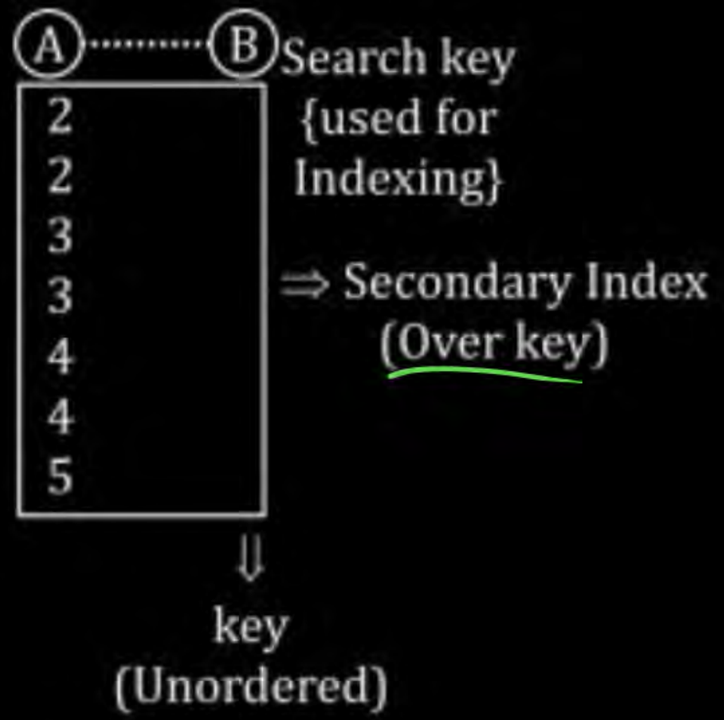
PAN → [key + unordered]

Aadhar → [key + unordered]

Name [Non key + unordered]

Types of Index

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



Types of Index

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

non-key: Clustering Index

Clustering Index [Non key + Ordered File]

Primary Indexes

[P.K + Ordered File]

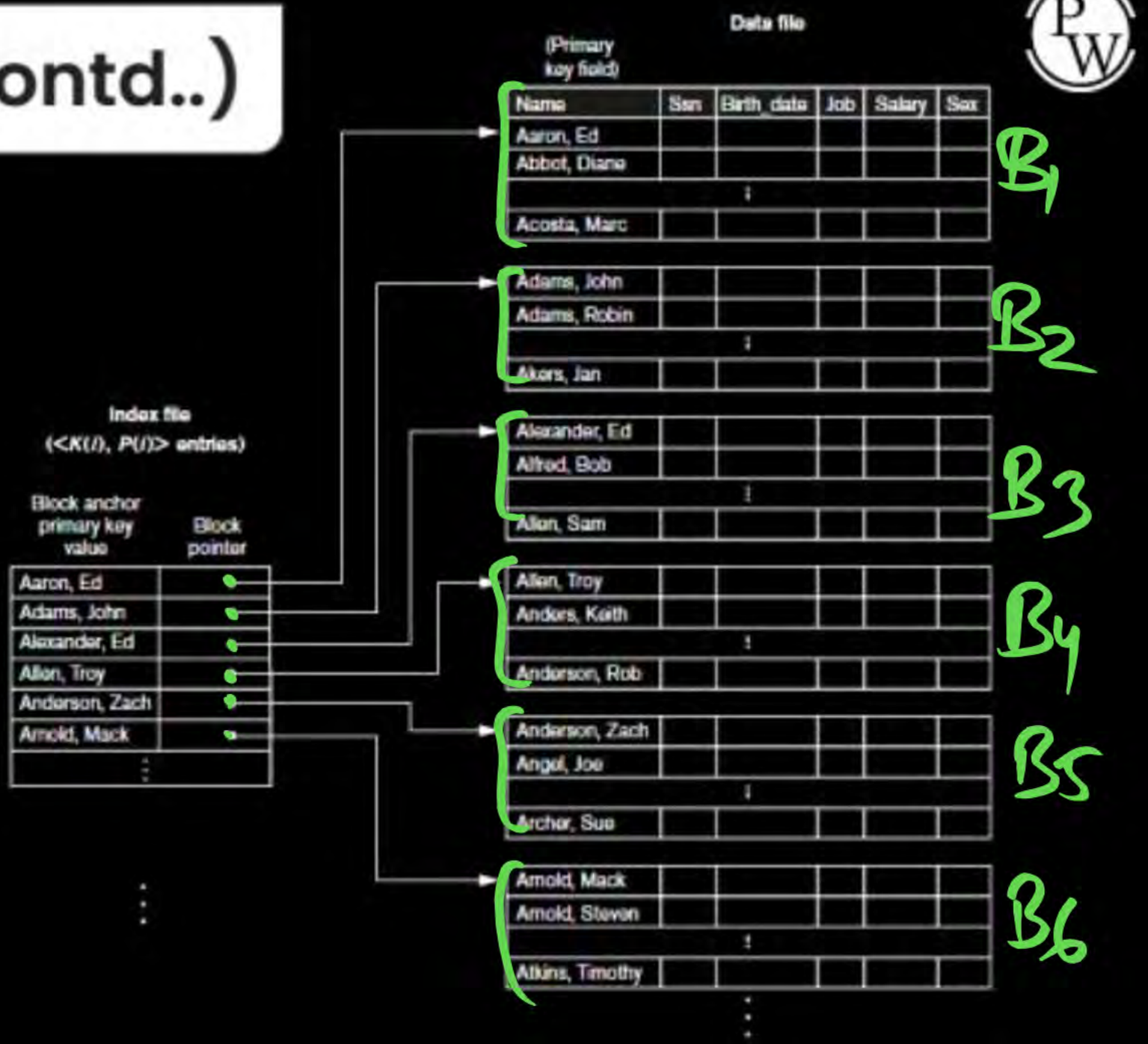
- Ordered file with two fields
  - Primary key, K(i)
  - Pointer to a disk block, P(i)
- One index entry in the index file for each block in the data file
- Indexes may be dense or sparse
  - Dense index has an index entry for every search key, value in the data file
  - Sparse index has entries for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

P.I : [SPARSE & Dense]
Mostly SPARSE.

(#DB Block)

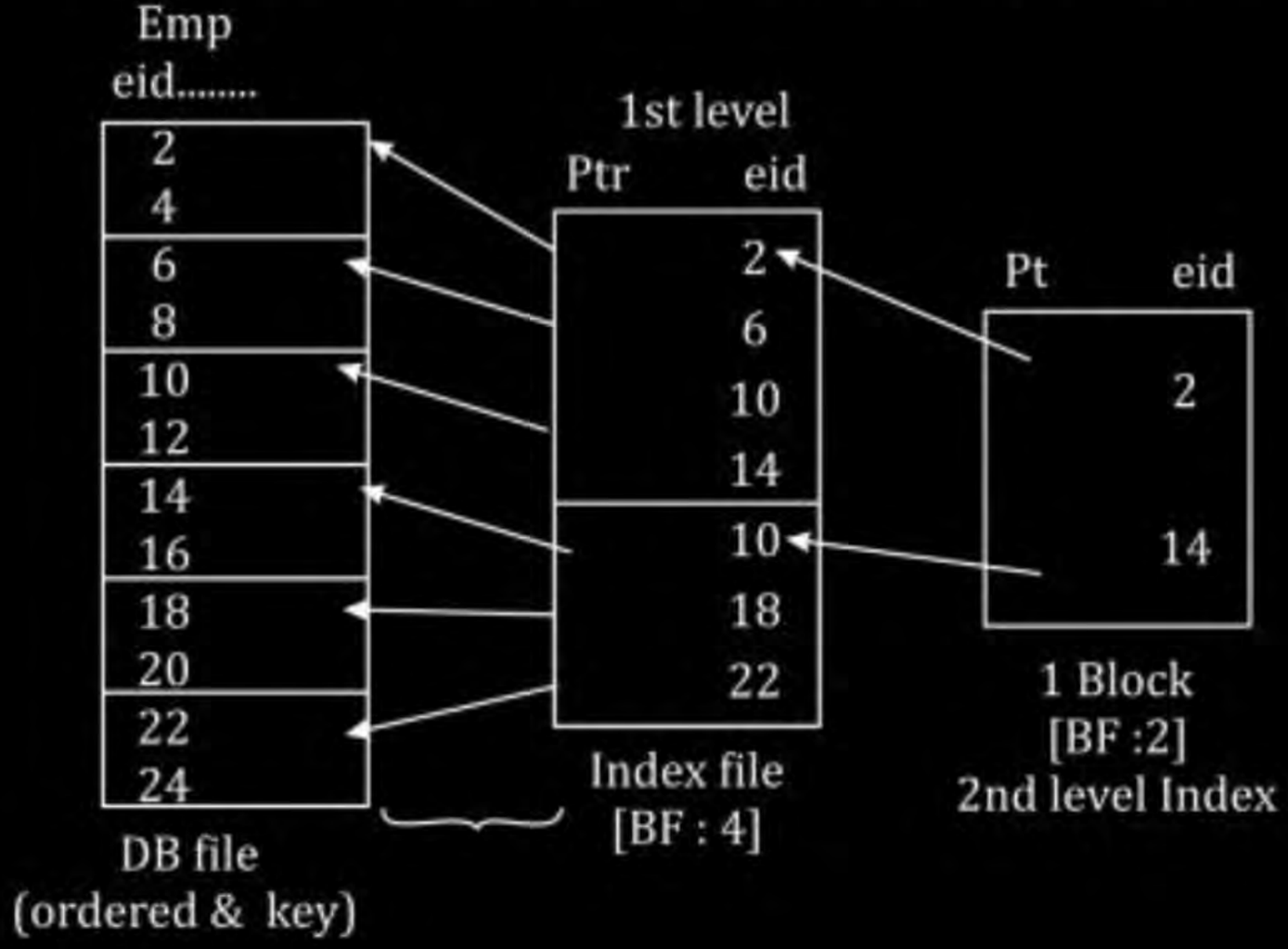
Primary Index (Contd..)

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 ?

[SPARSE]

[Home Work]

2 Clustering Indexes

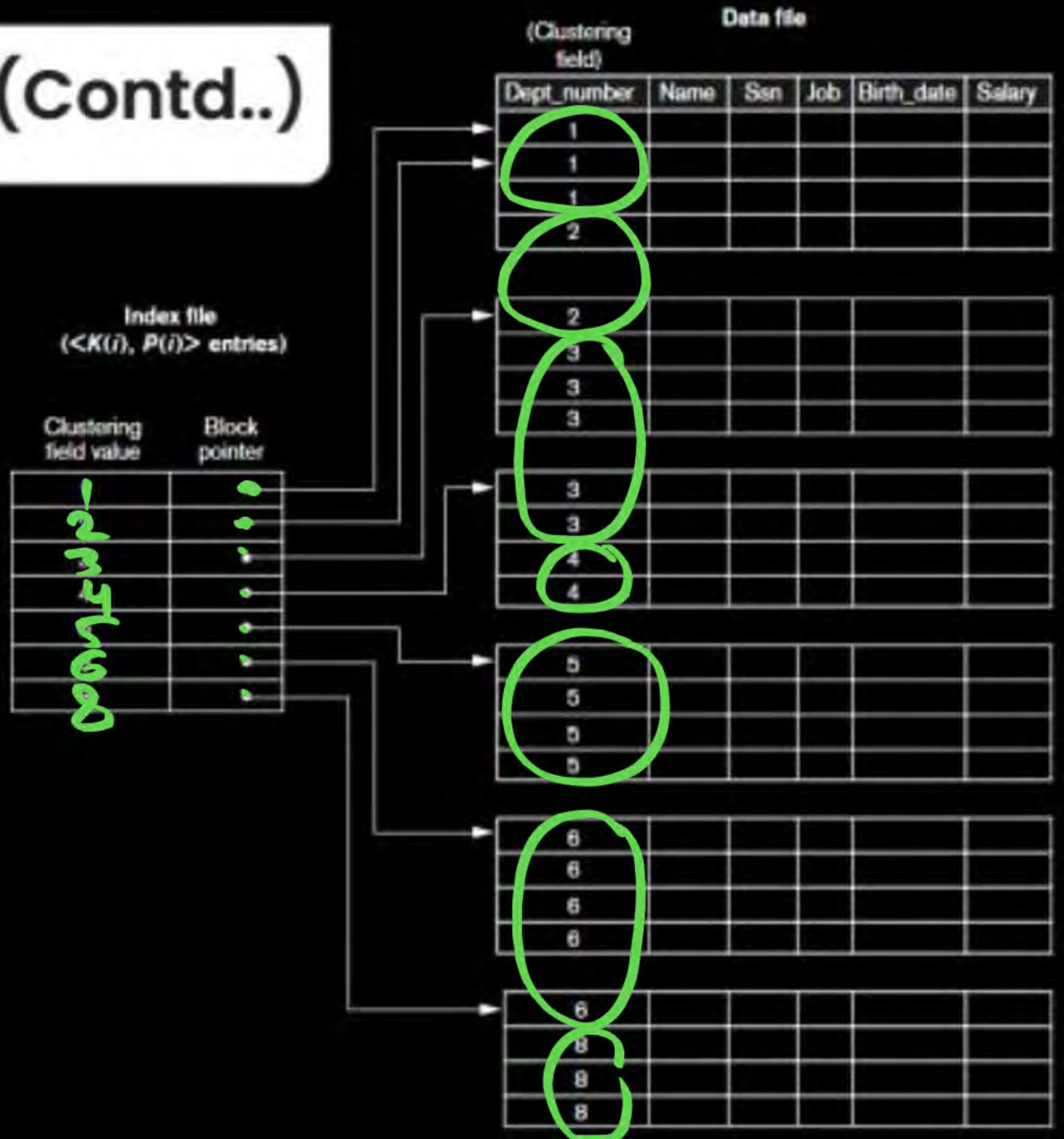
[Nonkey + Ordered File]

- 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..)

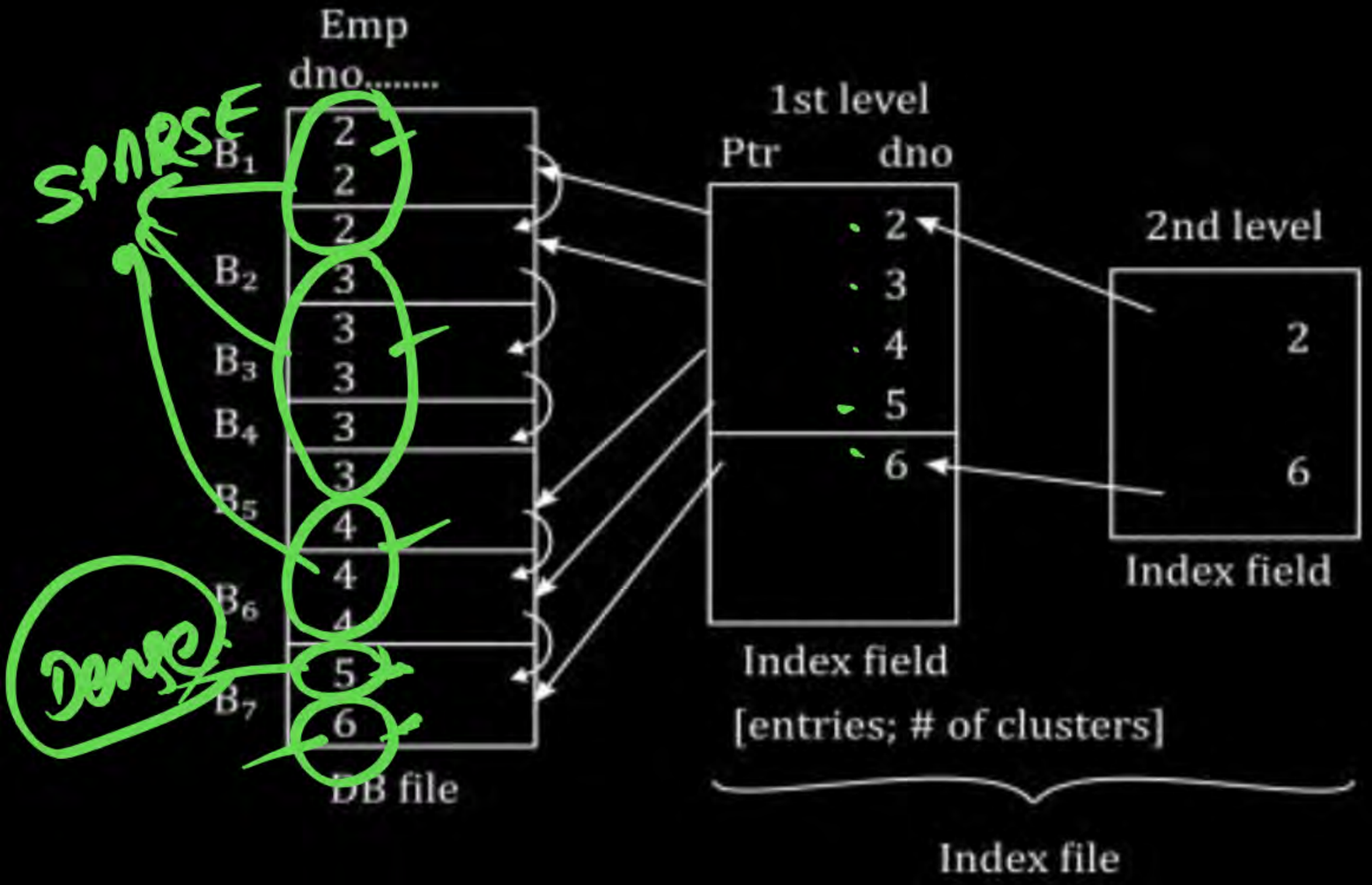
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file

SPARSE & Dense



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]

Note

- For any DB relation can build either PI or CI but not both.

In a Relation if PI ✓ CI X
CI ✓ PI X
PI ✓ SI ✓
CI ✓ SI ✓

PI : Primary Index
CI : clustered Index
SI : secondary Index

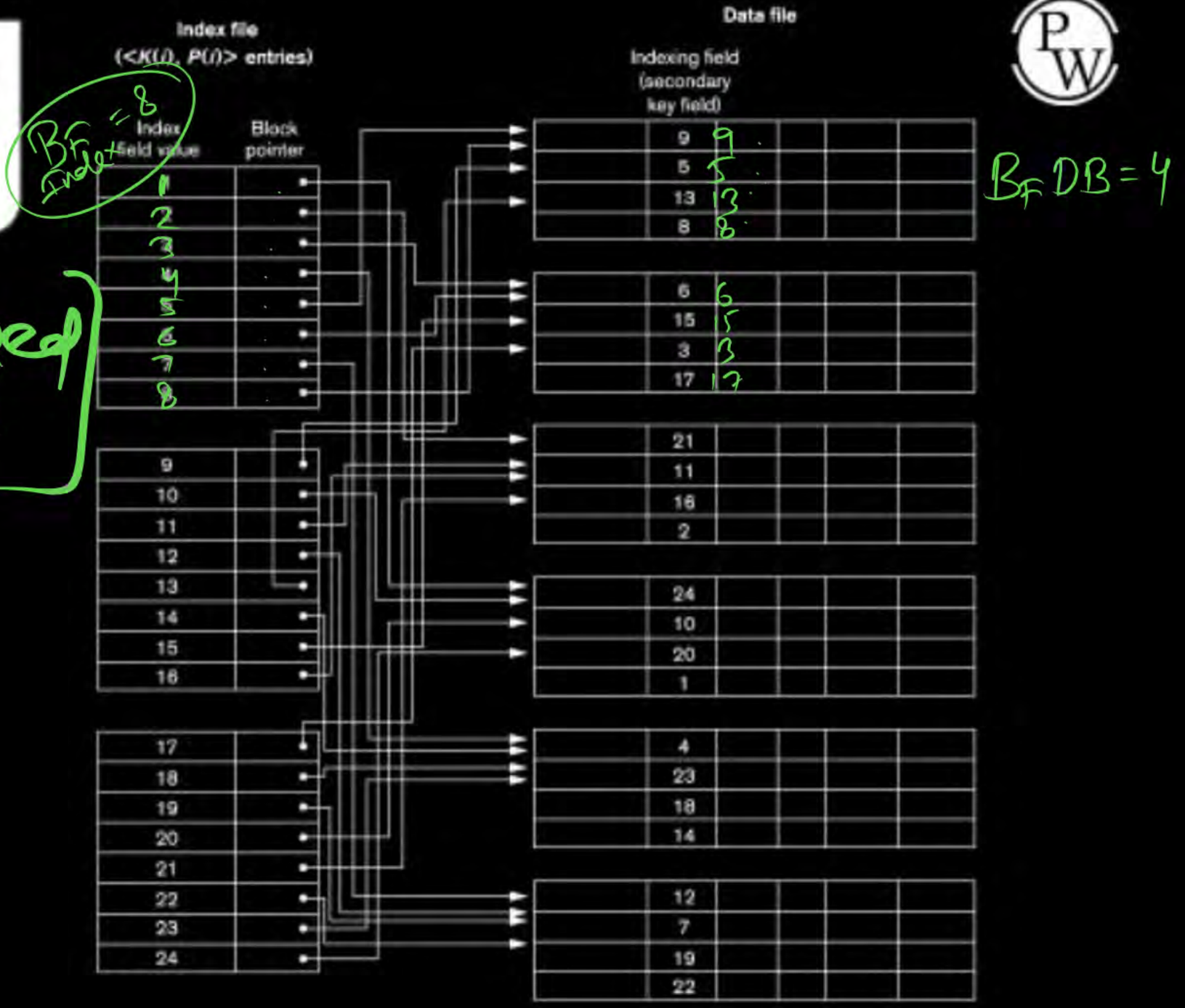
3 Secondary Indexes

- 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..)

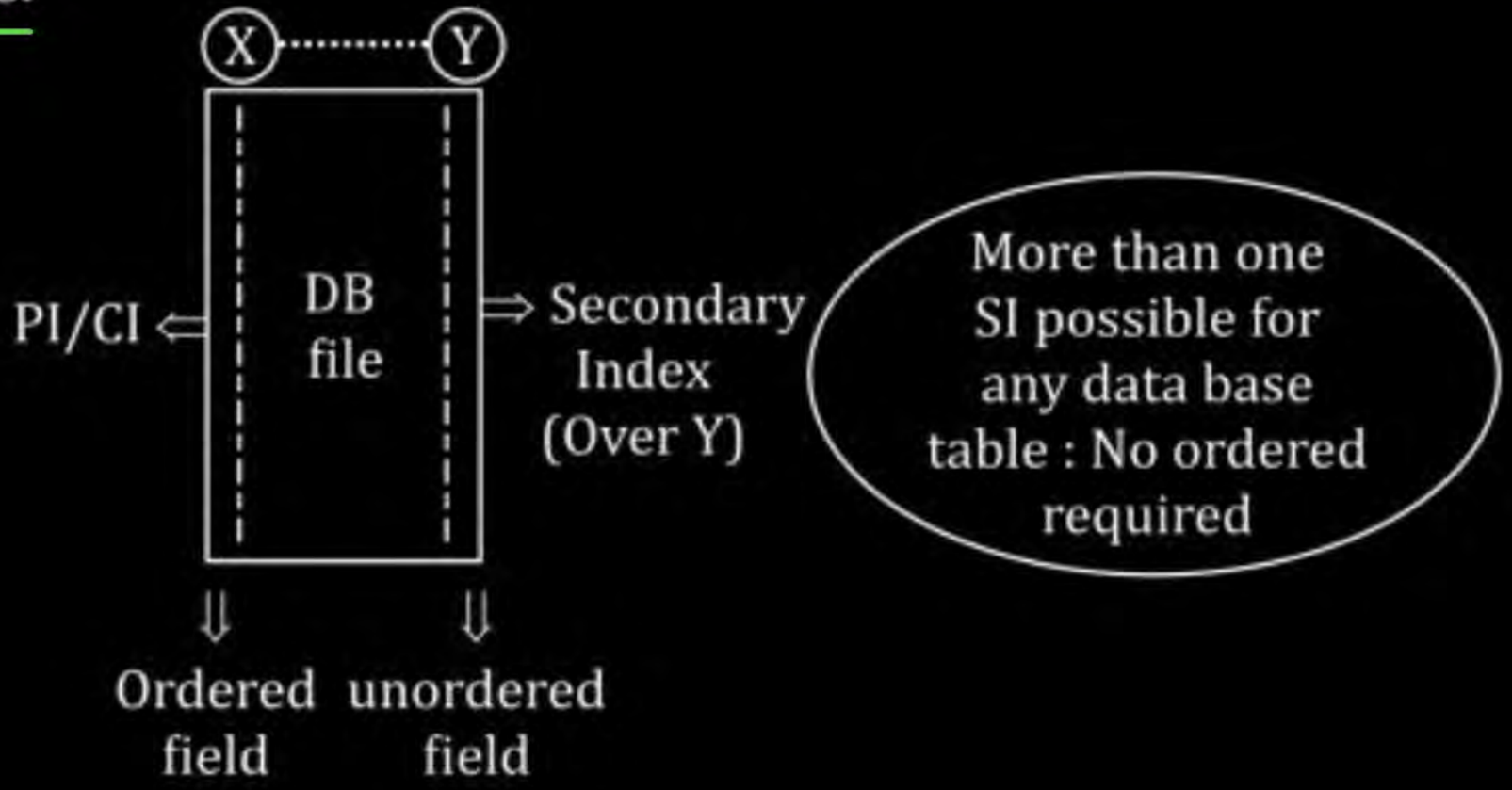
Nonkey / Candidate key + Unordered File

Dense.

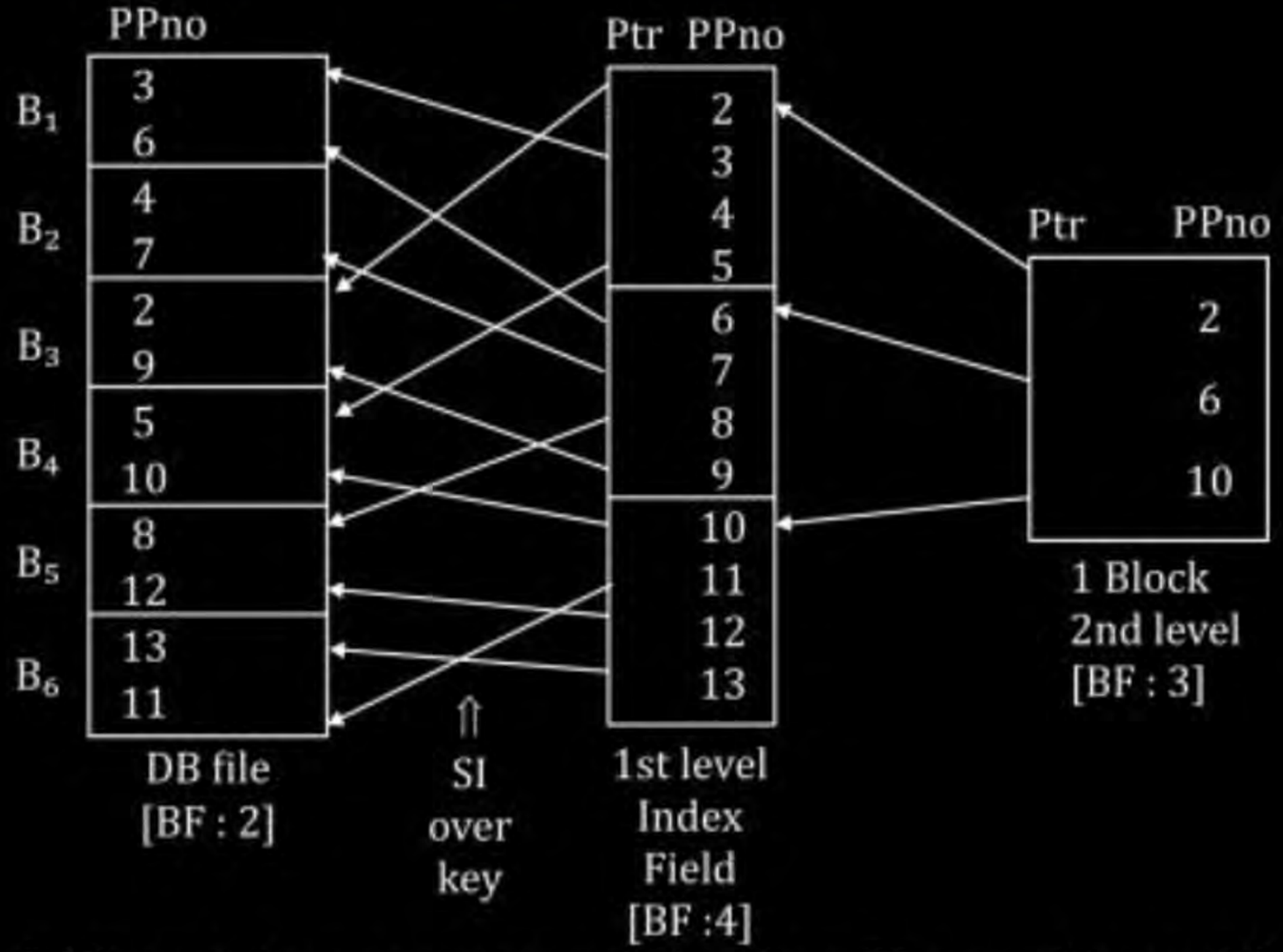


Secondary Index:

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

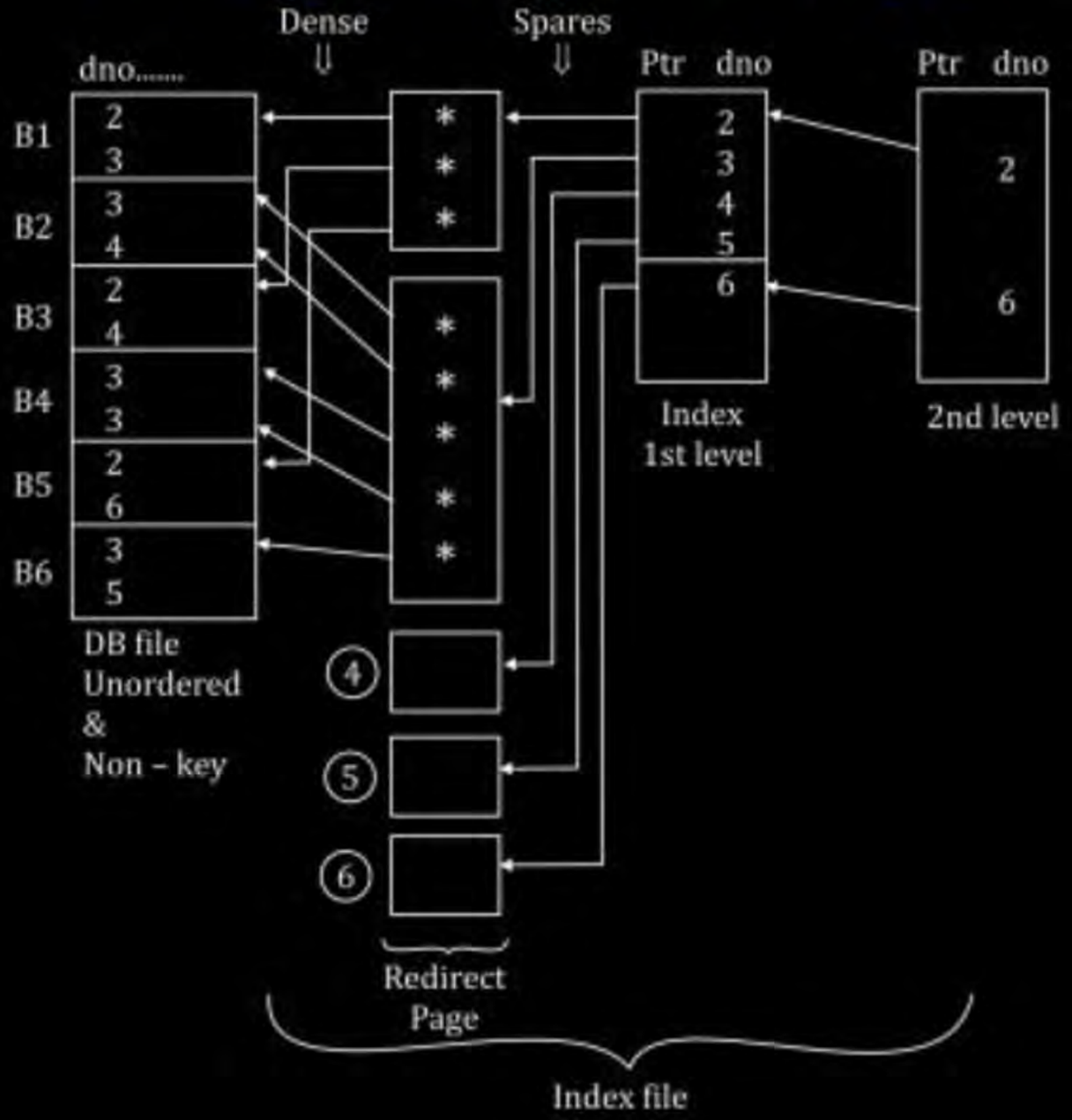


- Secondary Index (over key):



- 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}

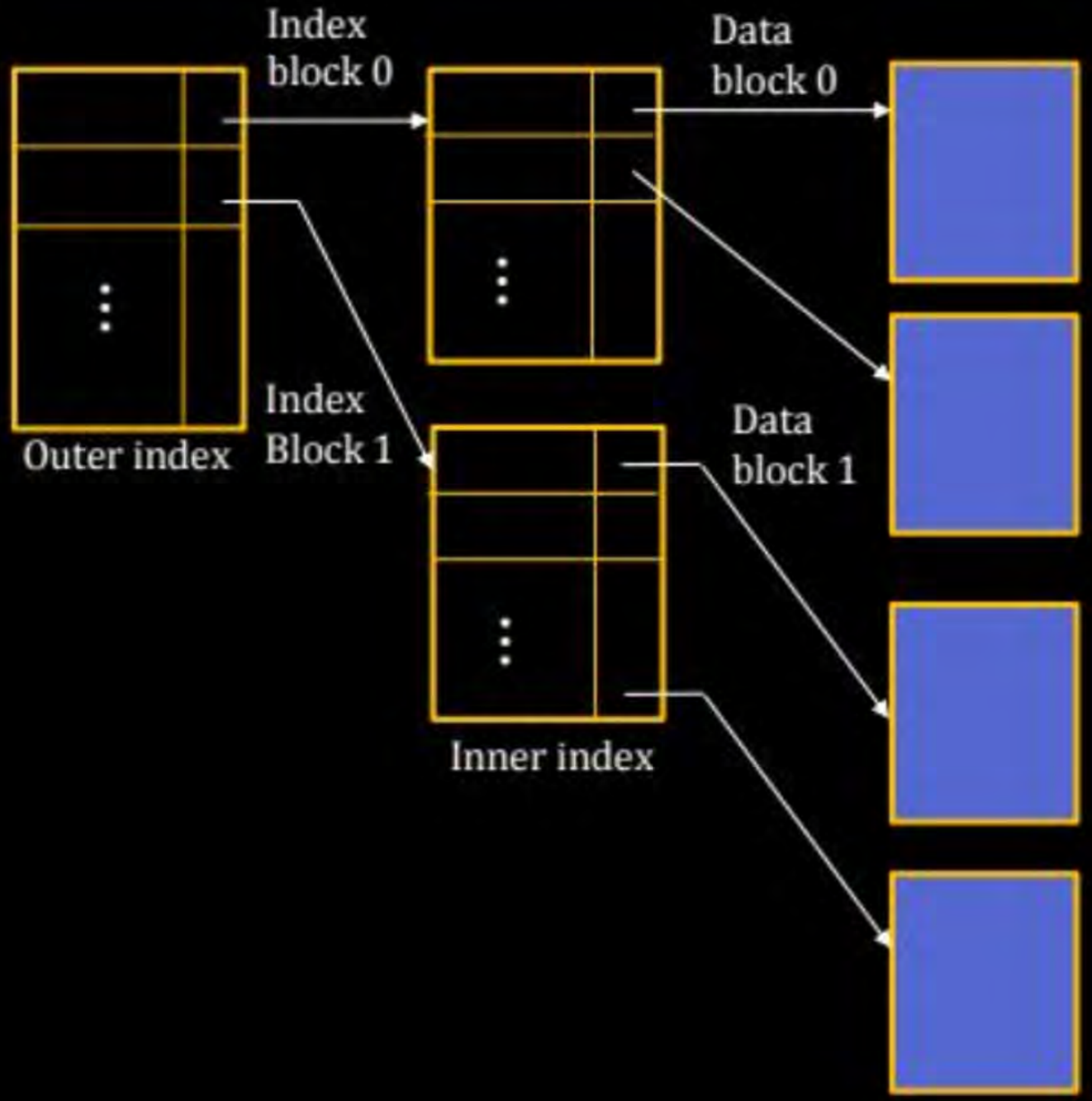
Q.2 Consider a secondary Index on the key field of the file of question number 1, then find the Average number of Block Access to Access a record using with & without Index?

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

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.

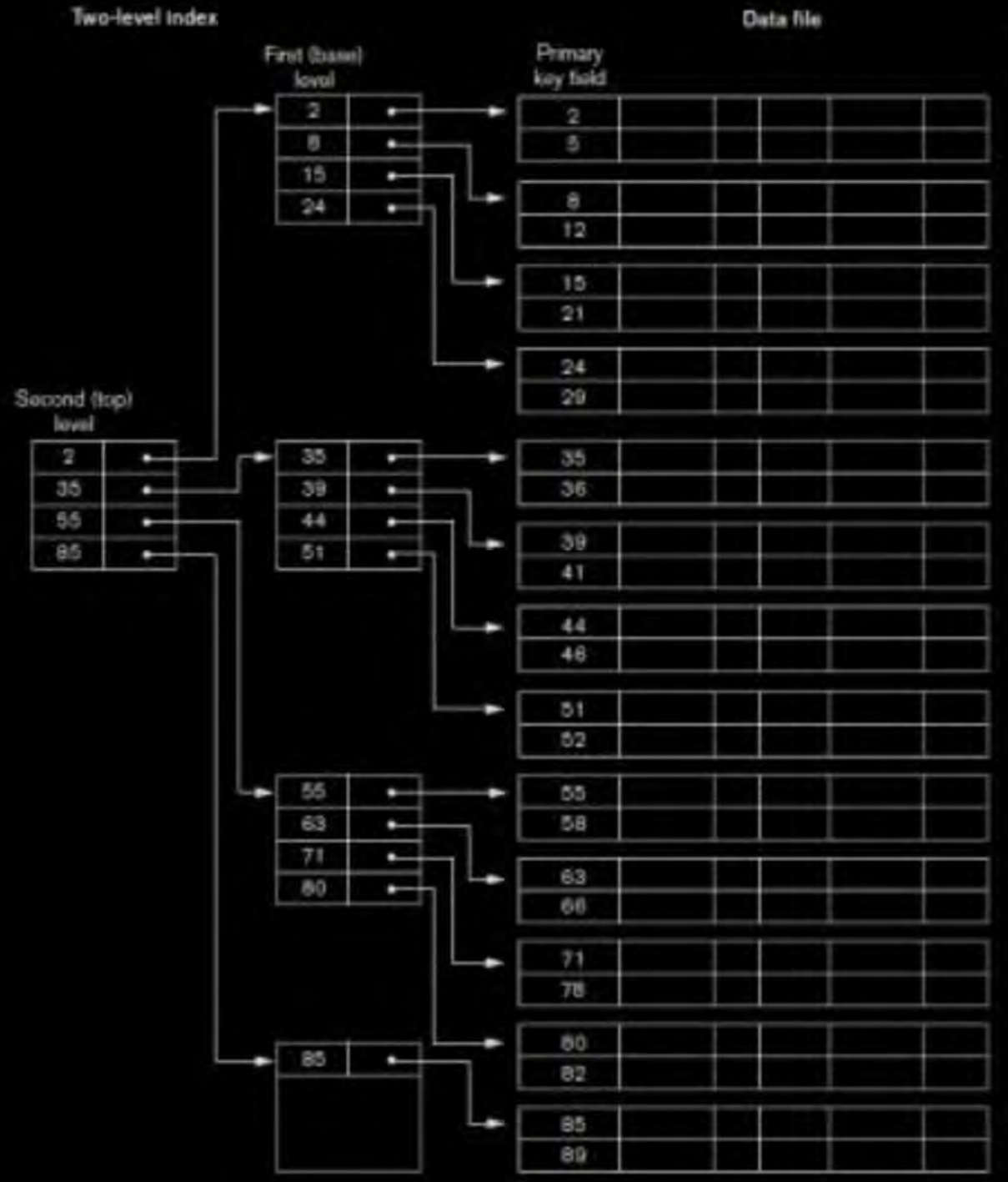
Multilevel Index (Contd..)



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.

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

THANK YOU!