CS & IT ENGINEERING

Database Management System

Lecture No. 10

By- Ravindrababu Ravula Sir

Recap of Previous Lecture

Topic ER Model

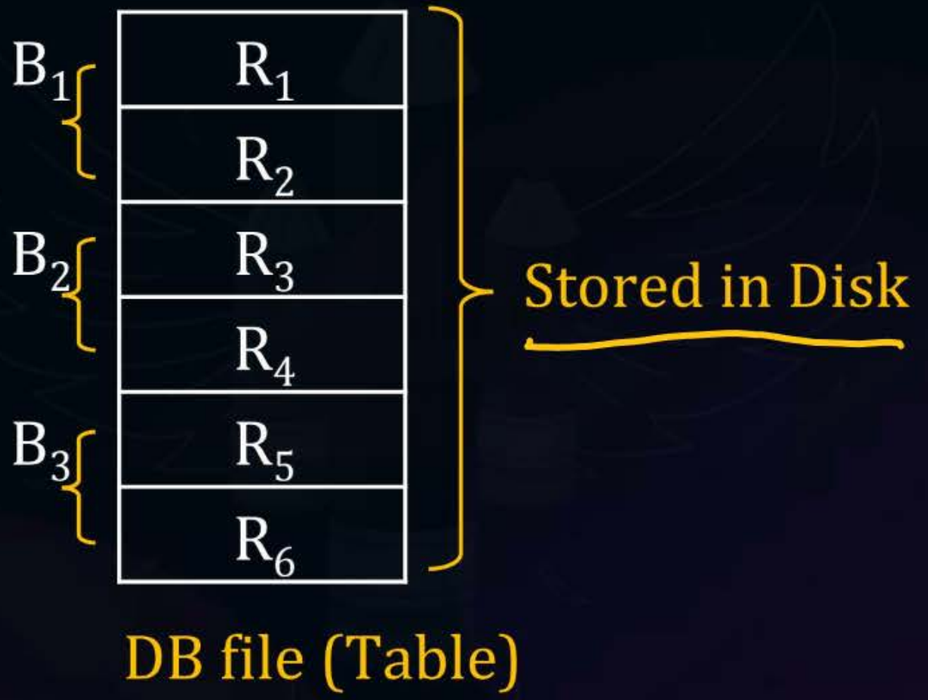
Topics to be Covered

Topic	File Organisation & Indexing
Topic	Types of Index
Topic	Primary Index
Topic	Clustering Index
Topic	Secondary Index
Topic	Dynamic Multilevel Index

Topic: File Org and Indexing

File Organization & Indexing

- DB is collection of Files (Tables)
- File is collection of Blocks (Pages) → sm
- Block is collection of records.



Topic: File Org and Indexing

- Data access from DISK to Main Memory is Block by Block.

Records in DB File

Fixed Length Records

Variable Length Records

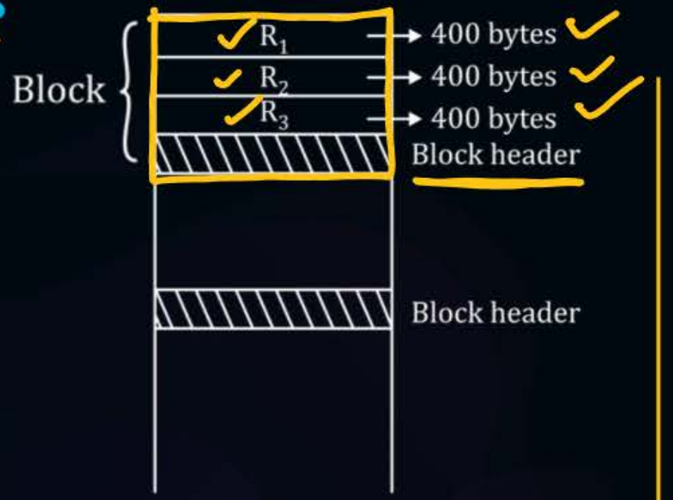
- Number of records in a block is dependent on the size of the Block

Topic: File Org and Indexing

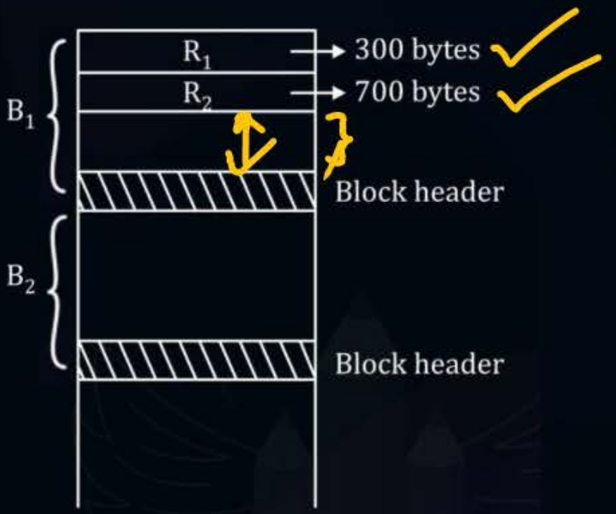
Records in DB file

Fixed length Records

.



Variable length records



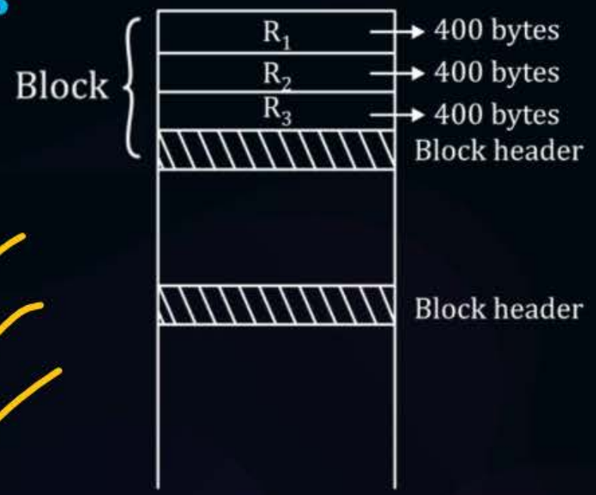
Topic: File Org and Indexing

Records in DB file

Fixed length Records

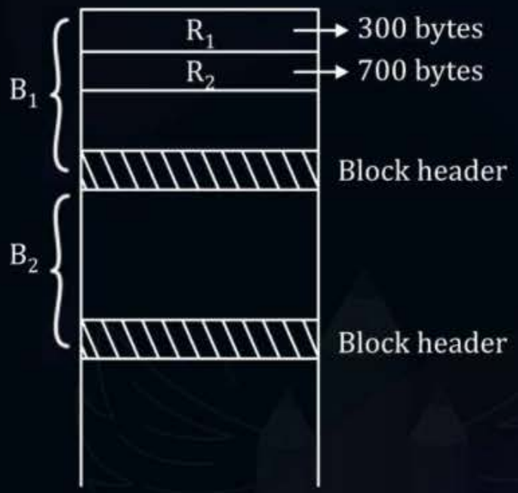
```
create table R
(
        A    char(100),
        B    char(100),
        C    char(200)
)
```

400



Variable length records

```
create table S
(       D char(200),
        E char(50),
        F text
);
```



Topic: File Org and Indexing

Records in DB file

Fixed length Records

create table R

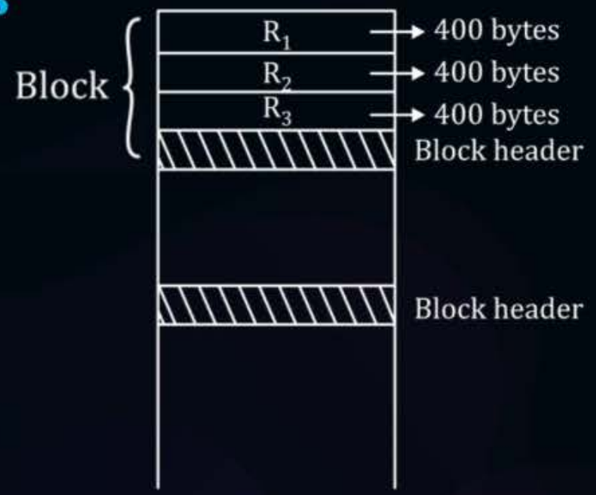
(

A char(100),

B char(100),

C char(200)

)



Now, even if the incoming data uses less memory, the length allocated will remain same.

Variable length records

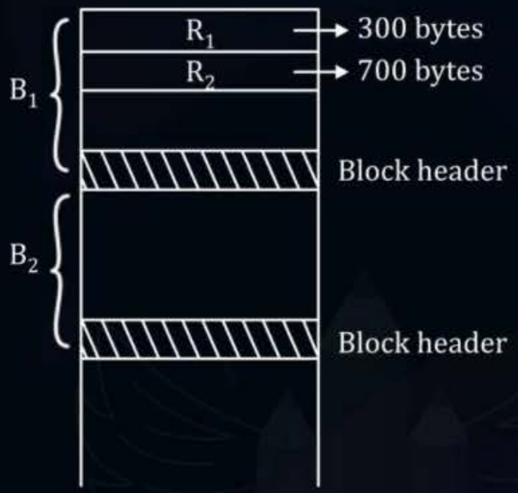
create table S

(D char(200),

E char(50),

F text

);



In case of F, allocation of space is not fixed, it depends on data stored here. If data is dynamic, then use Variable length records.

Topic: File Org and Indexing

Records in DB file

Fixed length Records

create table R

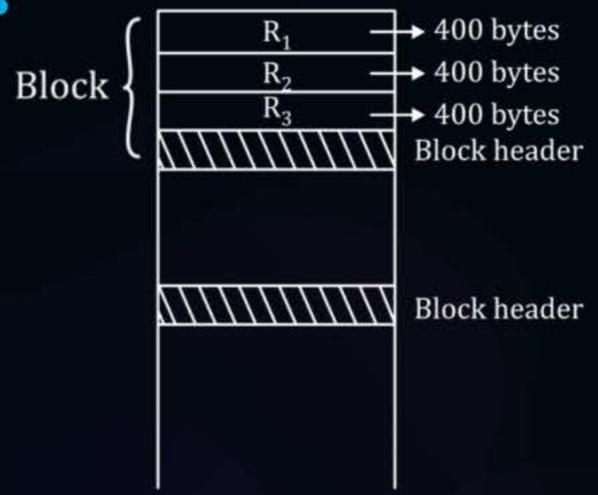
(

A char(100),

B char(100),

C char(200)

)



Now, even if the incoming data uses less memory, the length allocated will remain same.

Note that, **BLOCK HEADER** is used to store offset table to access records. Block header is necessary, without it, we can't manage the records of the block.

Variable length records

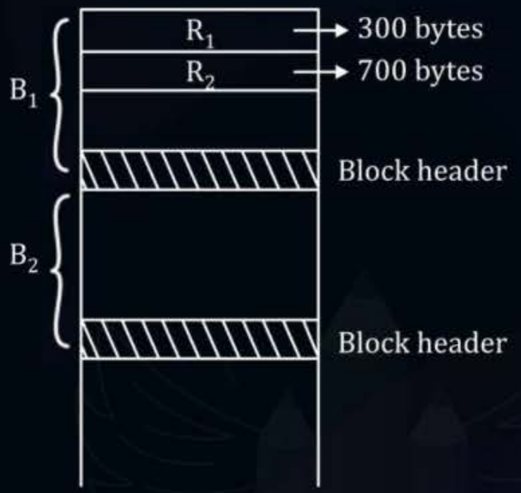
create table S

(D char(200),

E char(50),

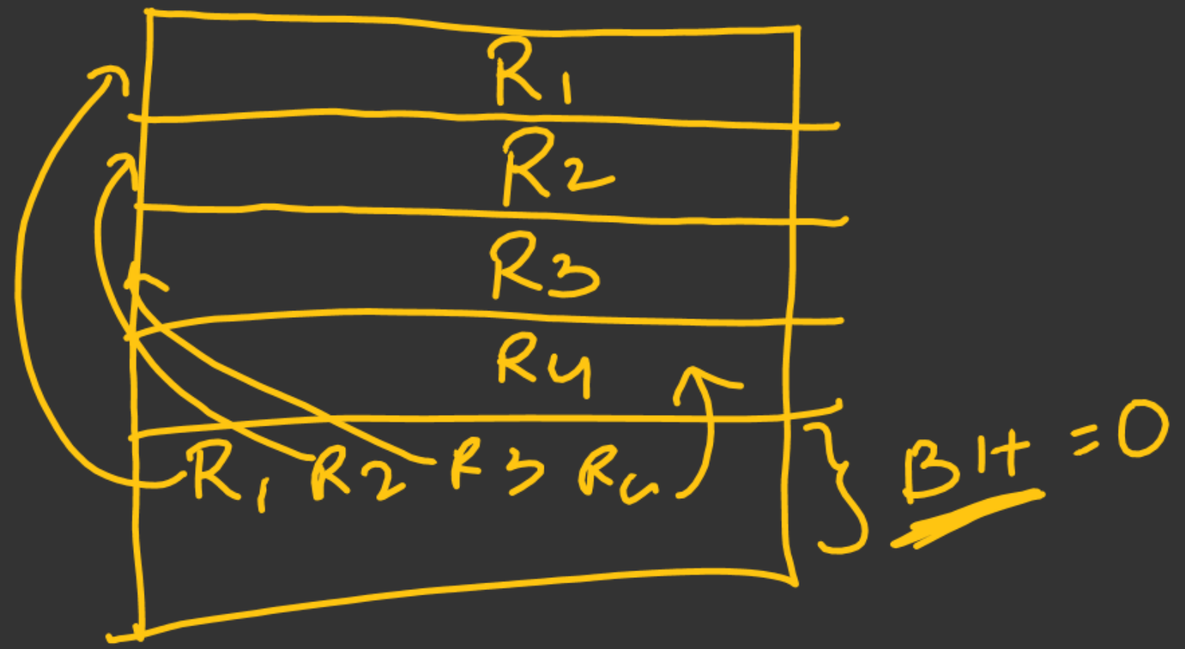
F text

);



In case of F, allocation of space is not fixed, it depends on data stored here. If data is dynamic, then use Variable length records.

Block header helps us access record in database file.


R1
R2
R3
R4
R1 R2 R3 R4
Bit =0

Inspiring Stories : Hugh Glass

Background: **Frontiersman in USA.**

Struggle: **Attacked by a grizzly bear, left for dead by his team.**

Achievements: **Crawled 200 miles to safety.**

Impact: **A symbol of never giving up.**

Topic: File Org and Indexing

Record Organization in DB File

Unspanned Org.

If complete record must be stored in one block

Say, Block size: 1000 byte

Record : 400 byte

Spanned Org.

Record allowed to span more than one block.

Topic: File Org and Indexing

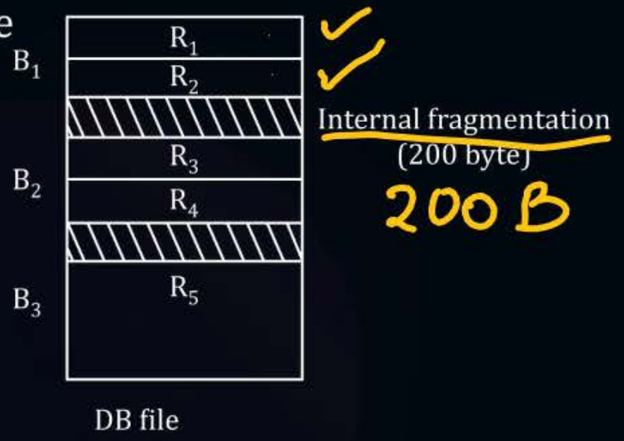
Record Organization in DB File

Unspanned Org.

If complete record must be stored in one block

Say, Block size: 1000 byte

Record : 400 byte

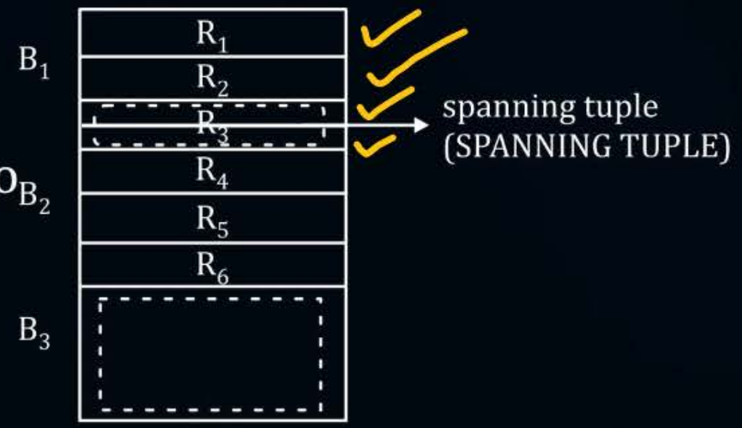


BS = 1000B

Rec = 400B

Spanned Org.

Record allowed to span more than one block.



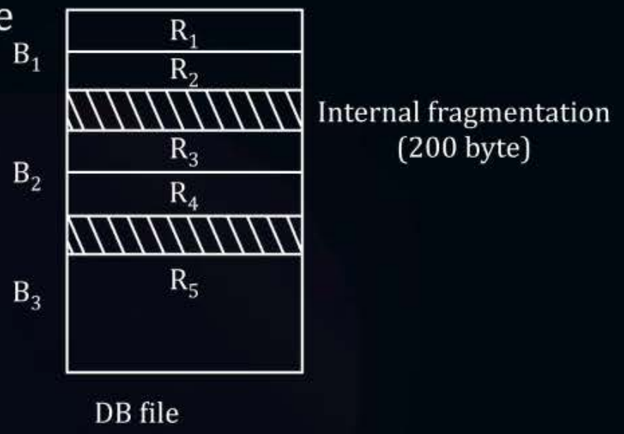
Topic: File Org and Indexing

Record Organization in DB File

Unspanned Org.

If complete record must be stored in one block

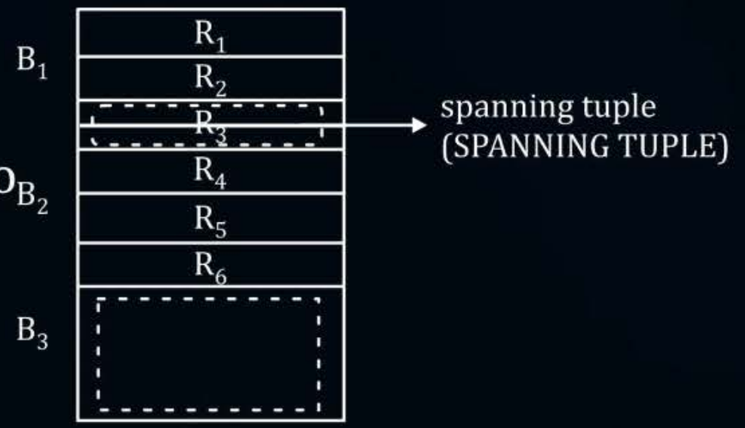
Say, Block size: 1000 byte
Record : 400 byte



- 1000 – 2 (400) = 200 (Internal Fragmentation)
- Less Access Cost
- To organize database file with fixed length records, we prefer Unspanned Organization.

Spanned Org.

Record allowed to span more than one block.



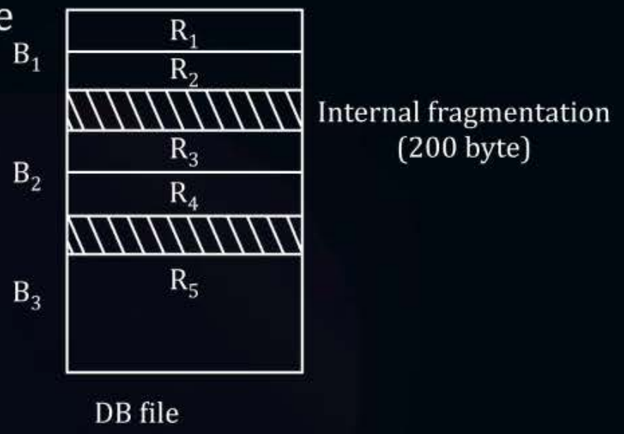
Topic: File Org and Indexing

Record Organization in DB File

Unspanned Org.

If complete record must be stored in one block

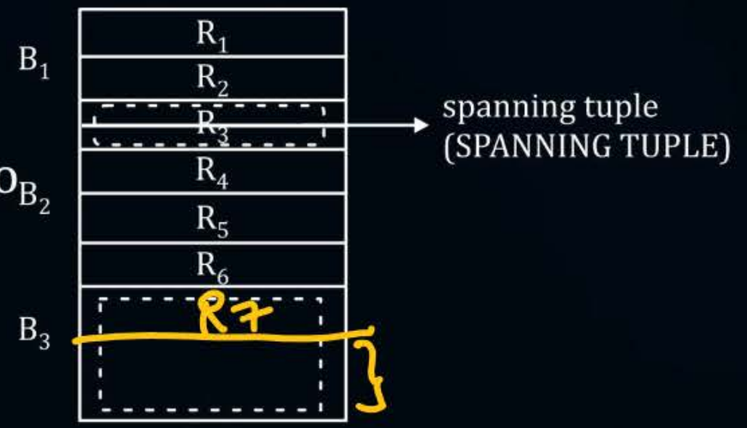
Say, Block size: 1000 byte
Record : 400 byte



- 1000 – 2 (400) = 200 (Internal Fragmentation)
- Less Access Cost
- To organize database file with fixed length records, we prefer Unspanned Organization.

Spanned Org.

Record allowed to span more than one block.



- Internal fragmentation happens only in last block.
- Space utilization is better.
- **NOTE:** data access happen from block to block. If we want Record R_3, we will need to access both Block B_1 and Block B_2, in that particular case Unspanned is better. But Spanned is more space efficient.
- More access cost.
- To organize database file with variable length records, we prefer Spanned organization.

Topic: File Org and Indexing

Block Factor of DB File

- Maximum possible records per block is called Block Factor of DB.

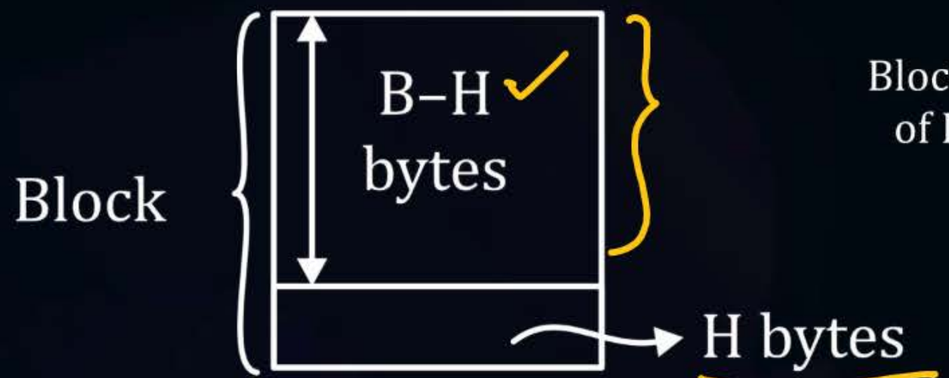
Block Size: B bytes Record Size: R bytes (Fixed len) Block Header Size: H bytes

Topic: File Org and Indexing

Block Factor of DB File

- Maximum possible records per block is called Block Factor of DB.

Block Size: B bytes Record Size: R bytes (Fixed len) Block Header Size: H bytes



unspanned org.

$$\text{Block factor of DB file} = \underbrace{\left\lfloor \frac{B-H}{R} \right\rfloor}_{\text{records/block}}$$

If header is not mentioned in exam, assume header size is zero.

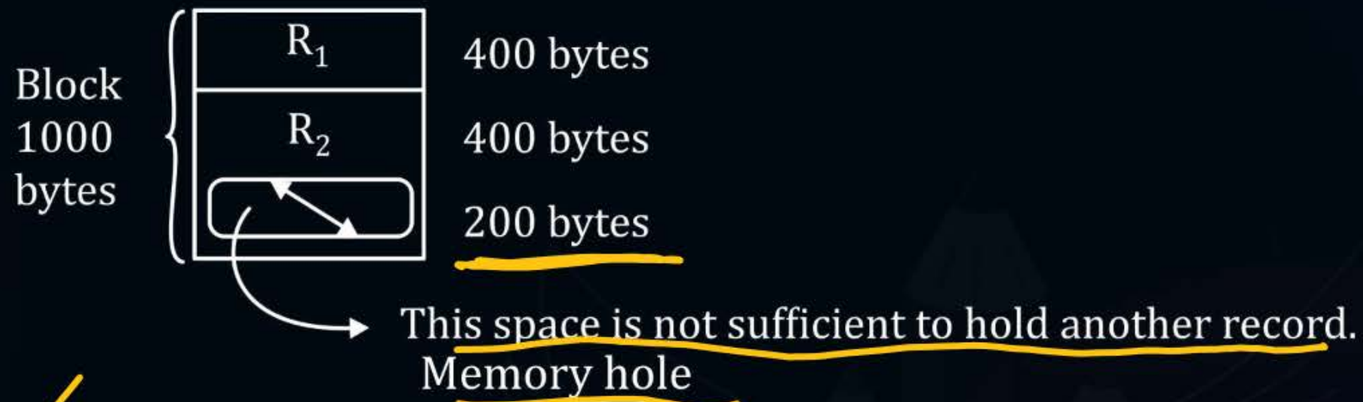
Topic: File Org and Indexing

Ex: Block : 1000 byte ,Record : 400 byte (Fixed length), what is its Block Factor?

$$\text{Bf of DB} = \left\lfloor \frac{B - H}{R} \right\rfloor$$

$$= \frac{1000 - 0}{400}$$

$$= 2 \text{ records/blocks}$$



There is no Block Factor for Spanned records, If BF is mentioned, then it is unspanned by default

Topic: File Org and Indexing

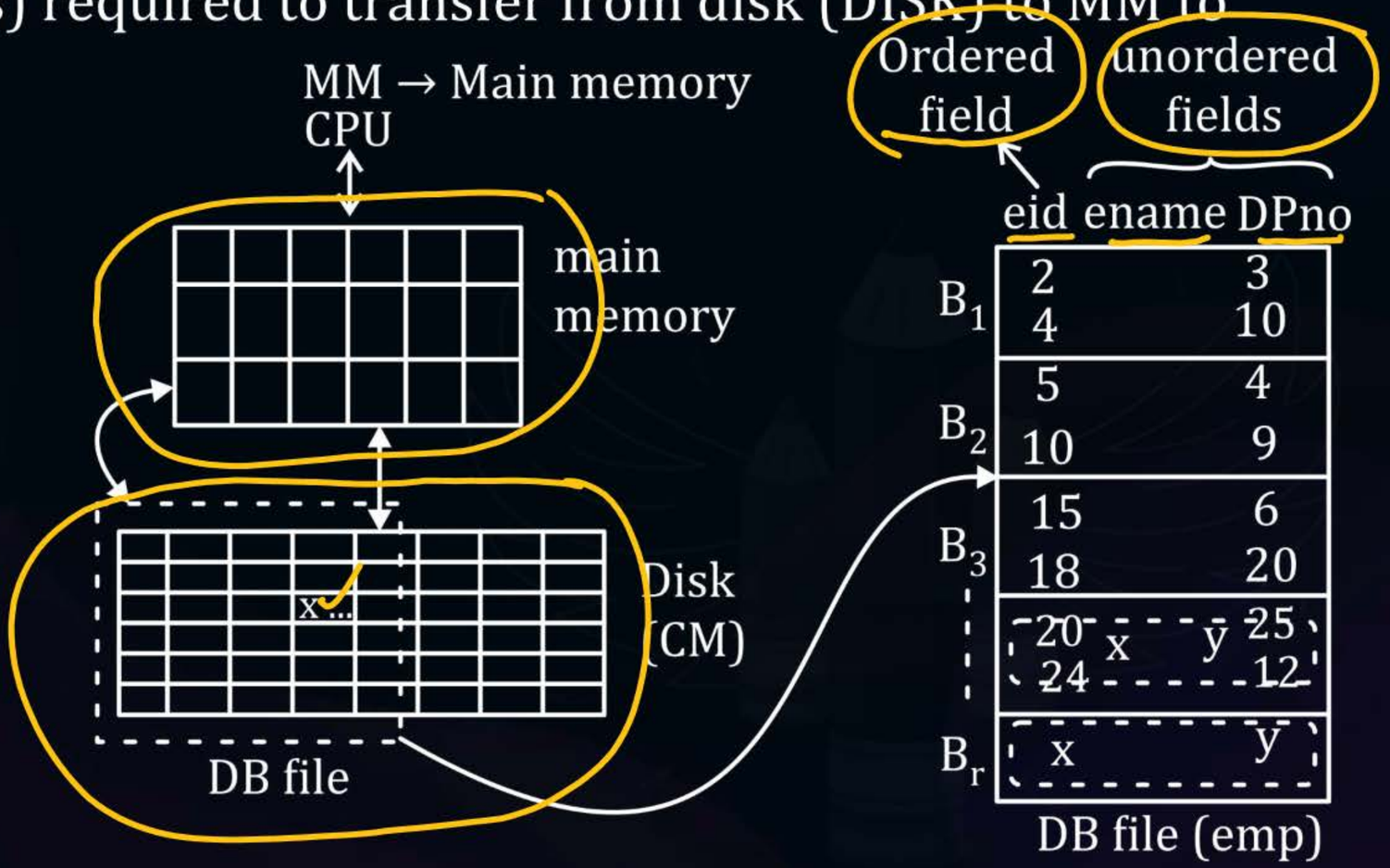
Access Cost (I/O cost)

No. of disk blocks (DB File blocks) required to transfer from disk (DISK) to MM to access required record.

Topic: File Org and Indexing

Access Cost (I/O cost)

No. of disk blocks (DB File blocks) required to transfer from disk (DISK) to MM to access required record.



Topic: File Org and Indexing

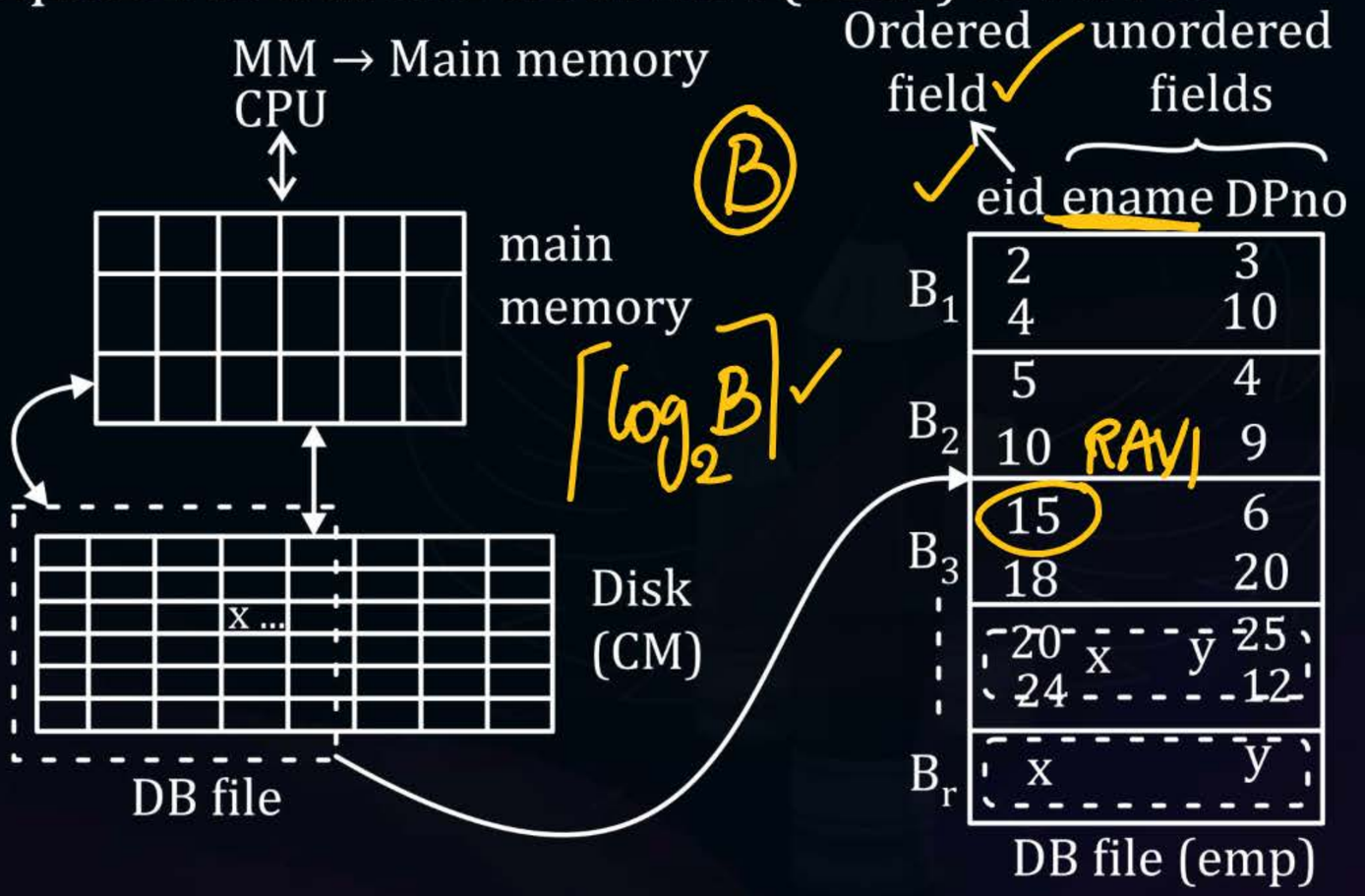
Access Cost (I/O cost)

No. of disk blocks (DB File blocks) required to transfer from disk (DISK) to MM to access required record.

CPU has to find which block to transfer to read X

If X is part of an Ordered key, then CPU can locate exact block, otherwise entire file has to be transferred and searched for X

Then the Access cost will be more



Topic: File Org and Indexing

I/O cost to access record from DB file without index:

(a) Using Ordered field:

```
Select *
From Emp
Where eid = x;
```

(b) Using Unordered field:

```
Select *
From Emp
Where ppno = y;
```

Topic: File Org and Indexing

I/O cost to access record from DB file without index:

(a) Using Ordered field:

```
Select *
From Emp
Where eid = x;
```

In case, if file is ordered by eid field we can apply binary search on eid

worst case I/O cost = $\lceil \log_2 n \rceil$ blocks.

Without Indexing, using ordered field = O(log n)

(b) Using Unordered field:

```
Select *
From Emp
Where ppno = y;
```

Topic: File Org and Indexing

I/O cost to access record from DB file without index:

(a) Using Ordered field:

```
Select *
From Emp
Where eid = x;
```

In case, if file is ordered by eid field we can apply binary search on eid

worst case I/O cost = $\lceil \log_2 n \rceil$ blocks.

Without Indexing, using ordered field = O(log n)

(b) Using Unordered field:

```
Select *
From Emp
Where ppno = y;
```

In case of unordered field, the worst case I/O cost would be n blocks. Because you will have to search through all records, one-by-one

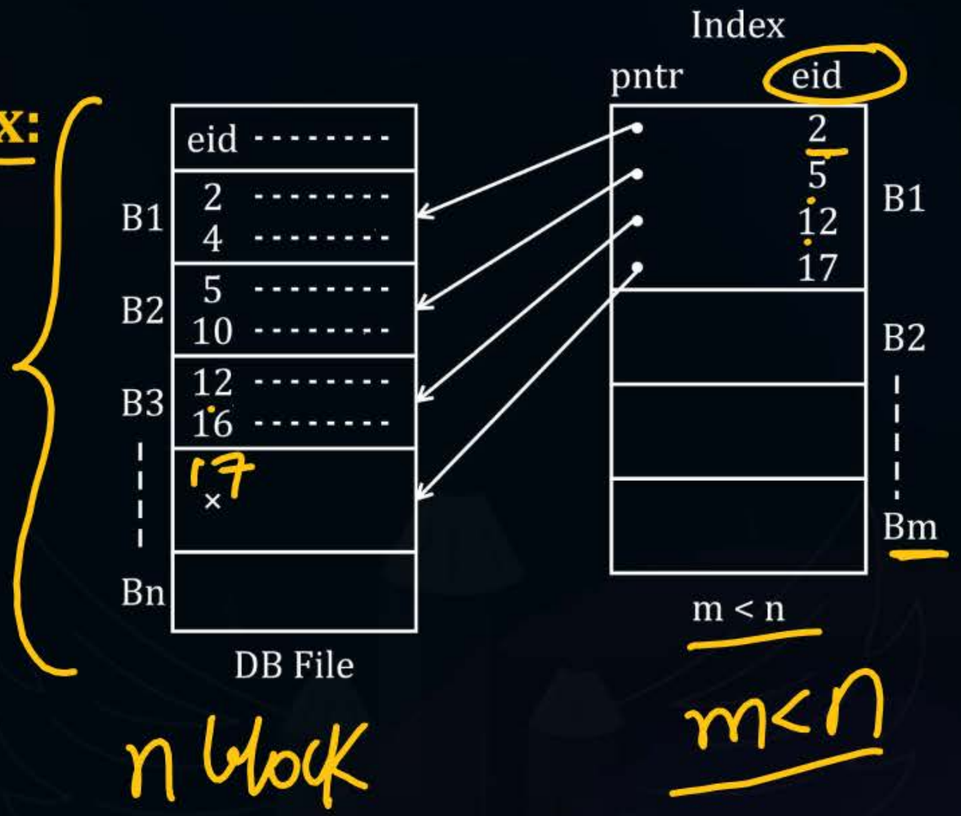
Without Indexing, using unordered field = O(n)

Topic: File Org and Indexing

Instead of directly fetching data as blocks from database file, we use something called Index file for it. We access required data block through index in lesser time

Topic: File Org and Indexing

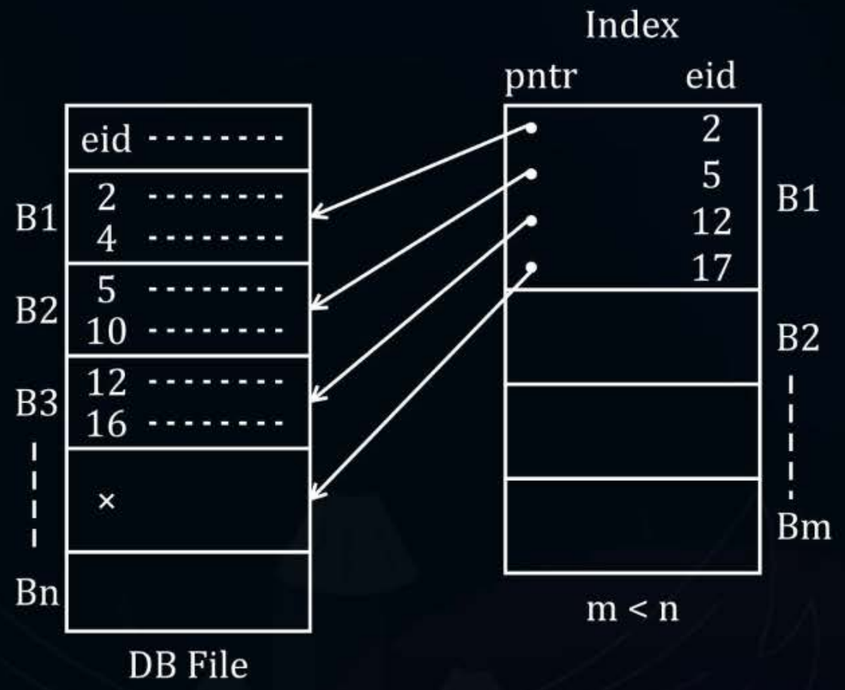
I/O cost to access record from DB file using index:



Topic: File Org and Indexing

I/O cost to access record from DB file using index:

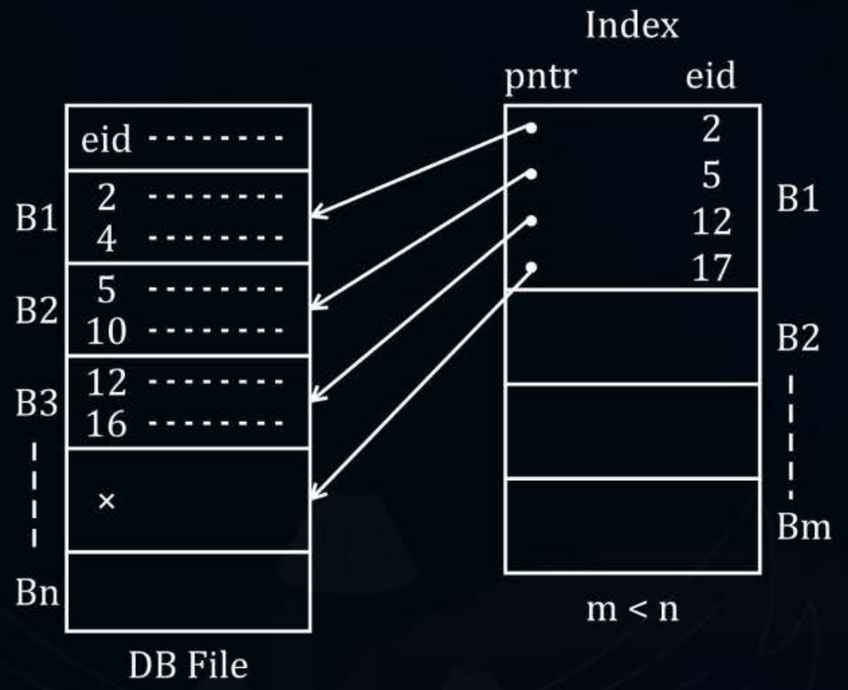
- We maintain a separate index file which consists of:
  1. search key i.e eid here
  2. block pointer of the block which contains search key record



Topic: File Org and Indexing

I/O cost to access record from DB file using index:

- We maintain a separate index file which consists of:
  1. search key i.e eid here
  2. block pointer of the block which contains search key record
- The index file is still in DISK, hence, it will be needed to retrieve to main memory.



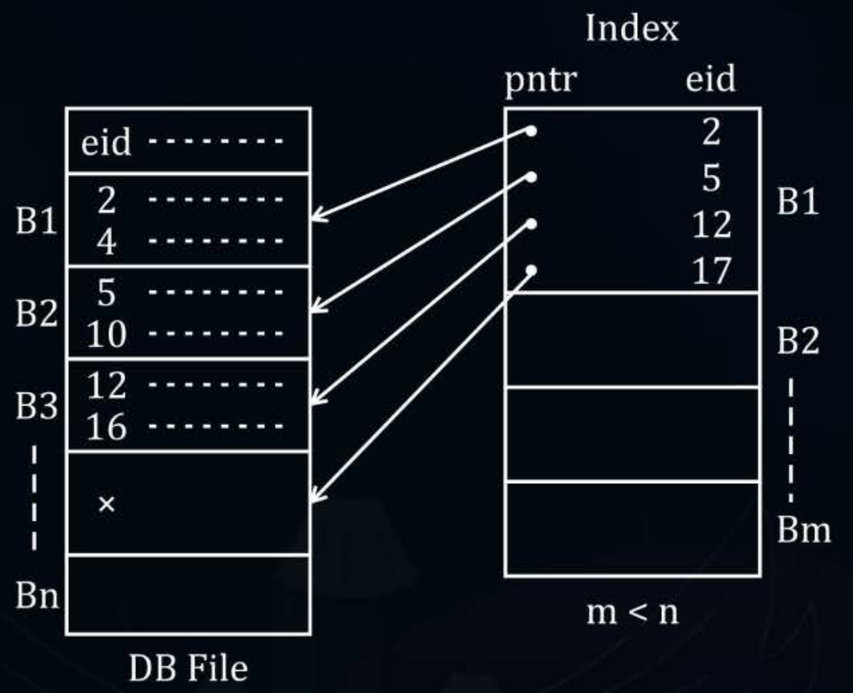
Topic: File Org and Indexing

I/O cost to access record from DB file using index:

- We maintain a separate index file which consists of:
  1. search key i.e eid here
  2. block pointer of the block which contains search key record
- The index file is still in DISK, hence, it will be needed to retrieve to main memory.

Why is m < n?

- Because we are just storing pointer and search key.



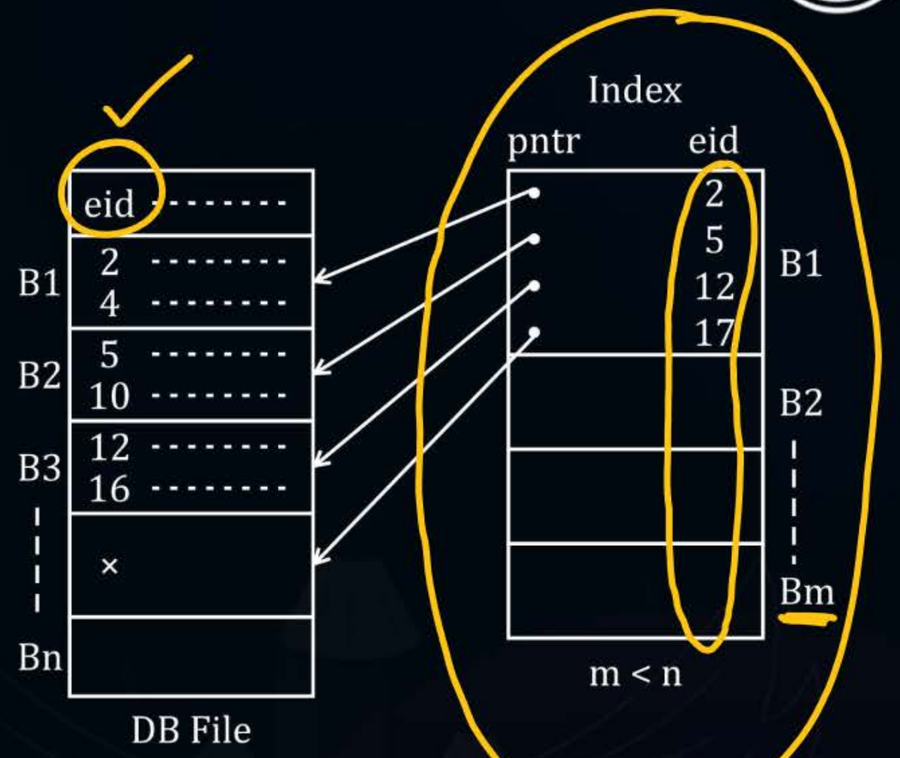
Topic: File Org and Indexing

I/O cost to access record from DB file using index:

- We maintain a separate index file which consists of:
  1. search key i.e eid here
  2. block pointer of the block which contains search key record
- The index file is still in DISK, hence, it will be needed to retrieve to main memory.

Why is m < n?

- Because we are just storing pointer and search key.



$\therefore$ Overall cost also reduces

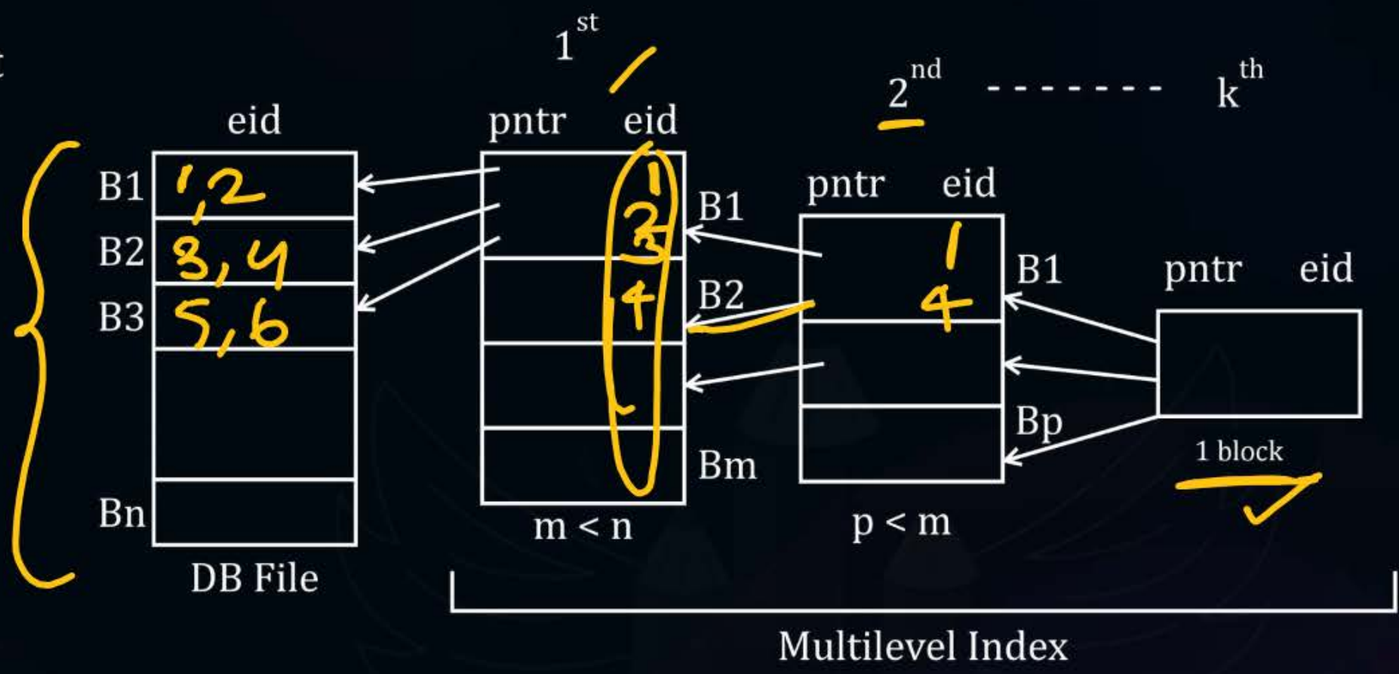
$\rightarrow$ I/O cost using 1 level indexing on a ordered search key = $\lceil \log_2 m \rceil + 1$ block.

To reduce cost further, we use Multilevel Index.

Topic: File Org and Indexing

Multilevel Index:

We introduce further index files to improve I/O cost



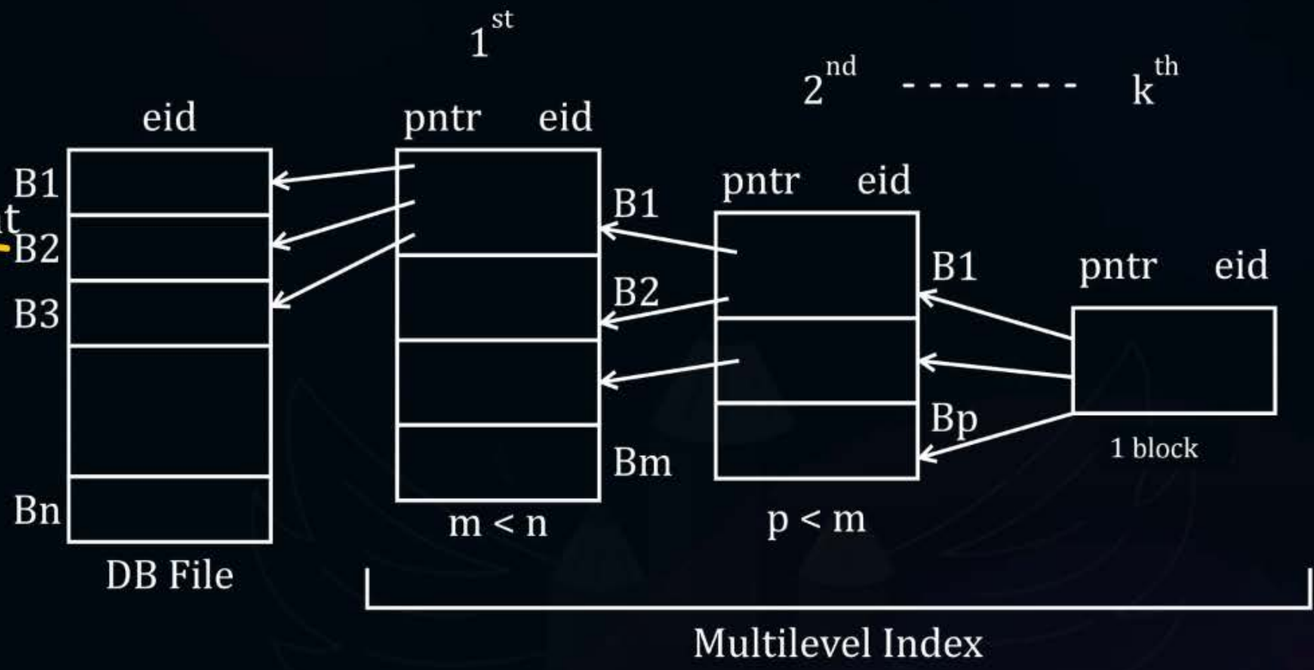
Topic: File Org and Indexing

Multilevel Index:

We introduce further index files to improve I/O cost

Concept:

We will keep introducing new level till k^{th} level, such that @ k^{th} level, we will have only one block.



Topic: File Org and Indexing

Multilevel Index:

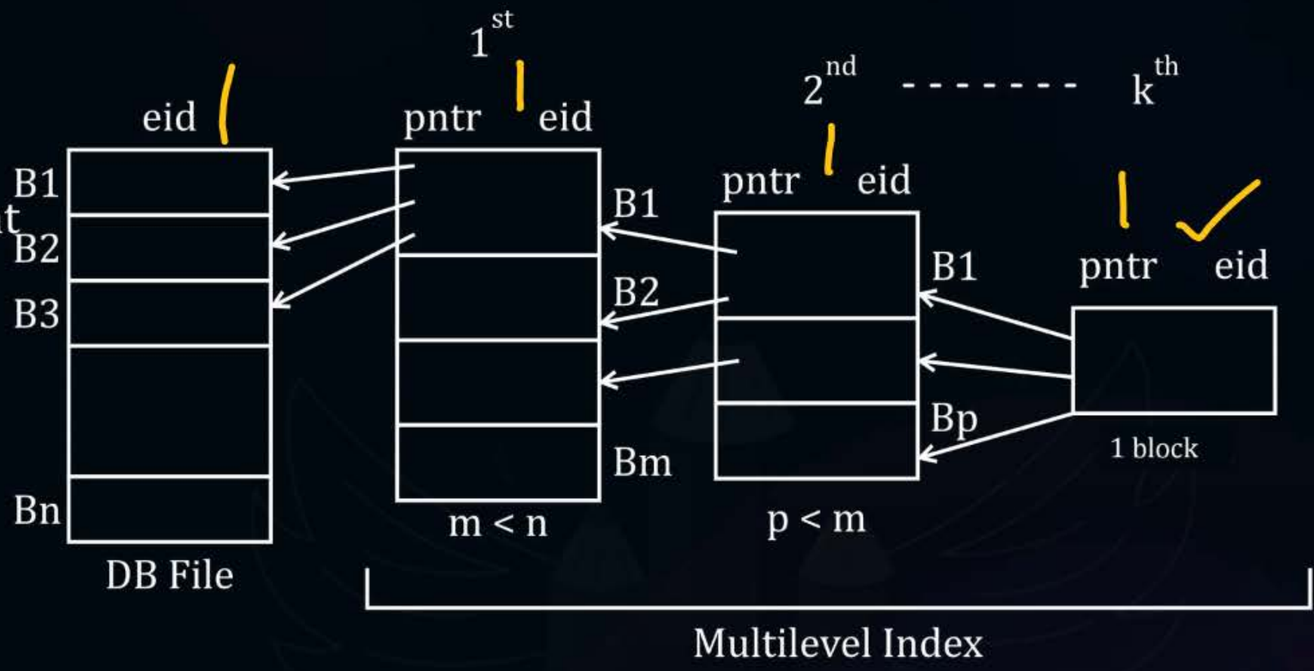
We introduce further index files to improve I/O cost

Concept:

We will keep introducing new level till k^{th} level, such that @ k^{th} level, we will have only one block.

I/O cost using Multilevel index:

(k + 1) blocks as we can see IO cost reduce significantly because we added multilevel index file.



Topic: File Org and Indexing

Index File:

Each entry of index has two fields, Block size is : B

<search key, pointer>

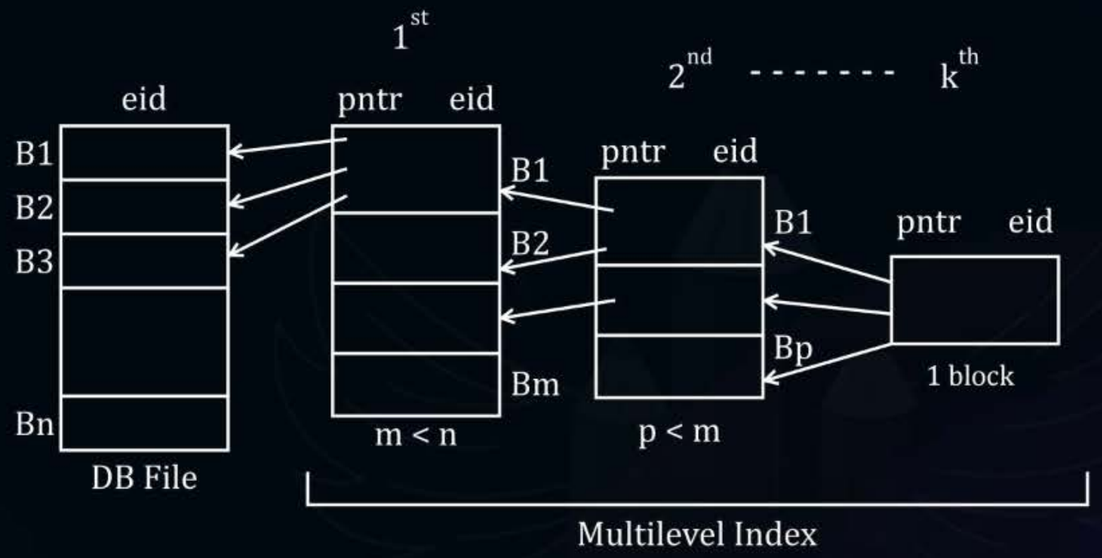
K bytes P bytes

Block factor of index file

Maximum possible index entries per block

Block factor of index : $\left\lfloor \frac{B-H}{K+P} \right\rfloor$ entries/block

$$\left\lfloor \frac{B-H}{K+P} \right\rfloor$$



Inspiring Stories : Tami Oldham

Background: **Sailor from USA.**

Struggle: **Yacht hit by hurricane in Pacific, fiancé lost, boat destroyed.**

Achievements: **Sailed alone for 41 days with no engine, reached land.**

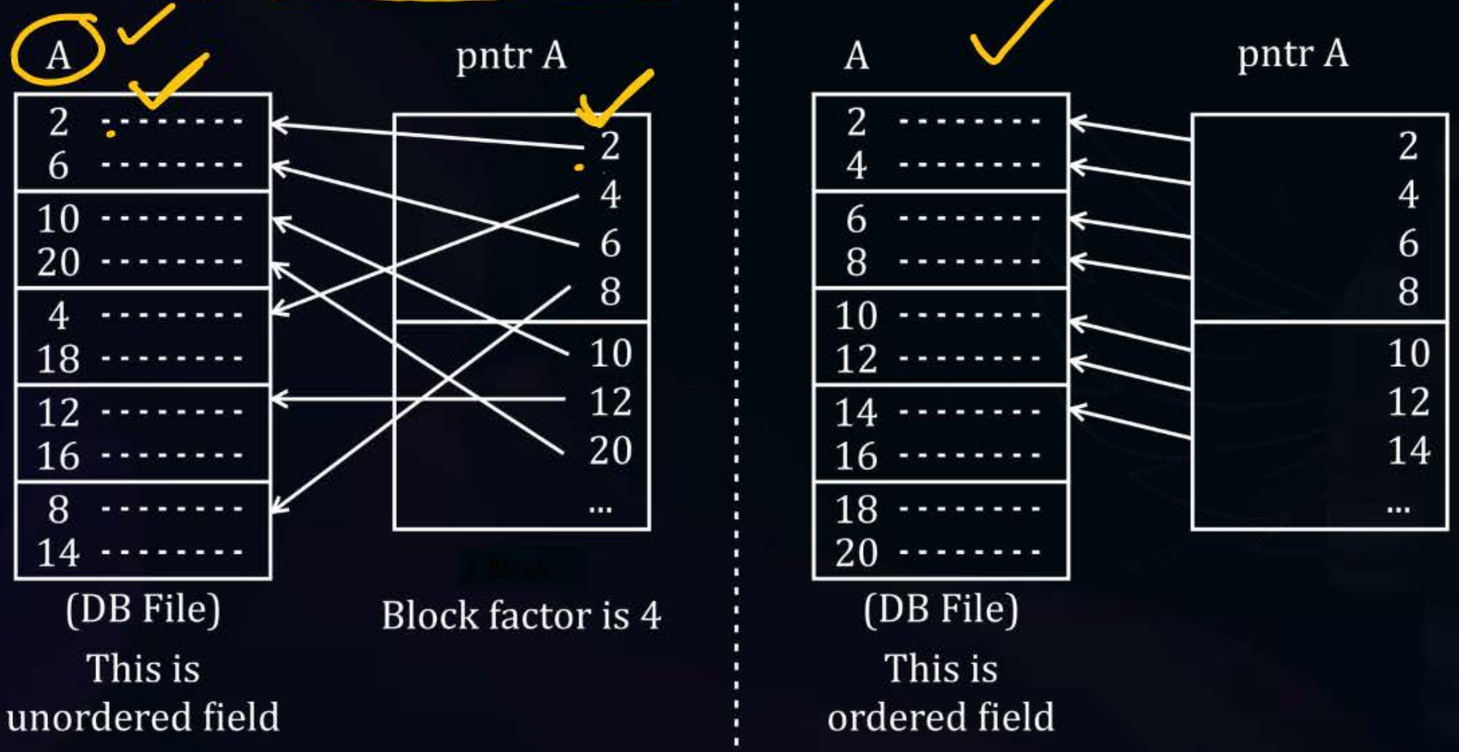
Impact: **Survived sea against all odds.**

Topic: File Org and Indexing

Categories of Index:

Dense Index (More no. of index entries):

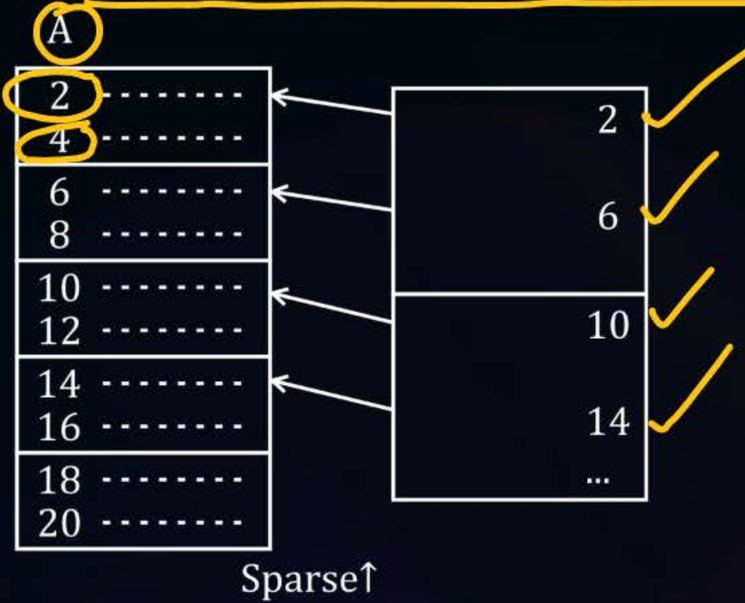
- For each record of DB file there must be entry in index file.
- No. of index entries = Records in DB file.



Topic: File Org and Indexing

Sparse Index (less no. of index entries):

- If the search key is ordered and a key then we can go with sparse index



Not possible, if used for index is unordered field and candidate key.

If sparse index over key

# of index file entries = # of blocks of DB file

Topic: File Org and Indexing

#Q. No. of records : 50000

Block : 1024 bytes Search key : 12 Bytes Record : 100 bytes Pointer size : 6 bytes

If dense index build

(a) no. of 1st level dense index blocks? (b) I/O cost using 1st level?

Topic: File Org and Indexing

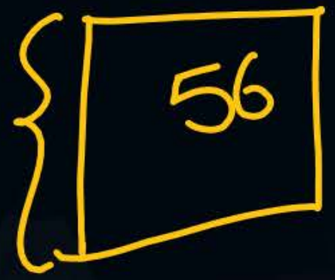
#Q. **No. of records : 50000**

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If dense index build

(a) no. of 1st level dense index blocks? (b) I/O cost using 1st level?

Dense index



Sol.

$$\text{Index file block factor} = \left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024+0}{12+6} \right\rfloor = 56 \text{ entries/block}$$

index

$$\text{No. of dense index blocks} = \left\lceil \frac{50000}{56} \right\rceil = 893 \text{ blocks}$$

$$\text{No. of searches in dense index} = \lceil \log_2 893 \rceil \approx 10 \text{ blocks}$$

So, 11 blocks are need to be transferred from DISK to main memory.

Topic: File Org and Indexing

#Q. **No. of records : 50000**

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If sparse index build

(a) no. of 1st level sparse blocks? (b) I/O cost (using 1st level)?

Topic: File Org and Indexing

#Q. **No. of records : 50000**

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If sparse index build

(a) no. of 1st level sparse blocks? (b) I/O cost (using 1st level)?

We need sparse index for 50,000 records in DB. Note that no. of index entries is equal to no. of blocks of DB.

$$\text{B. f. of DB} = \left\lfloor \frac{B - H}{R} \right\rfloor = \left\lfloor \frac{1024 - 0}{100} \right\rfloor = 10 \quad \text{R/blocks}$$

Topic: File Org and Indexing

#Q. No. of records : 50000

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If sparse index build

(a) no. of 1st level sparse blocks? (b) I/O cost (using 1st level)?

We need sparse index for 50,000 records in DB. Note that no. of index entries is equal to no. of blocks of DB.

$$\text{B. f. of DB} = \left\lfloor \frac{B - H}{R} \right\rfloor = \left\lfloor \frac{1024 - 0}{100} \right\rfloor = 10 \quad \text{R/blocks}$$

If one block can accommodate 10 records; then for 50000 records, we need 50,000/10 no. of blocks = 5000 blocks

if DB file is 5000 blocks $\Rightarrow$ 5000 index entries.

$$\text{Index file block factor} = \left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024+0}{12+6} \right\rfloor = 56 \text{ entries/block}$$

No. of index entries = No. of block of DB

Topic: File Org and Indexing

#Q. No. of records : 50000

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If sparse index build

(a) no. of 1st level sparse blocks? (b) I/O cost (using 1st level)?

We need sparse index for 50,000 records in DB. Note that no. of index entries is equal to no. of blocks of DB.

$$\text{B. f. of DB} = \left\lfloor \frac{B - H}{R} \right\rfloor = \left\lfloor \frac{1024 - 0}{100} \right\rfloor = 10 \quad \text{R/blocks}$$

If one block can accommodate 10 records; then for 50000 records, we need 50,000/10 no. of blocks = 5000 blocks

if DB file is 5000 blocks $\Rightarrow$ 5000 index entries.

$$\text{Index file block factor} = \left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024+0}{12+6} \right\rfloor = 56 \text{ entries/block}$$

No. of index entries = No. of block of DB

$$\text{So, Sparse index blocks} = \left\lceil \frac{5000}{56} \right\rceil = 90$$

Topic: File Org and Indexing

#Q. No. of records : 50000

Block : 1024 bytes **Search key : 12 Bytes** **Record : 100 bytes** **Pointer size : 6 bytes**

If sparse index build

(a) no. of 1st level sparse blocks? (b) I/O cost (using 1st level)?

We need sparse index for 50,000 records in DB. Note that no. of index entries is equal to no. of blocks of DB.

$$\text{B. f. of DB} = \left\lfloor \frac{B - H}{R} \right\rfloor = \left\lfloor \frac{1024 - 0}{100} \right\rfloor = 10 \quad \text{R/blocks}$$

If one block can accommodate 10 records; then for 50000 records, we need 50,000/10 no. of blocks = 5000 blocks

if DB file is 5000 blocks $\Rightarrow$ 5000 index entries.

Index file block factor $= \left\lfloor \frac{B-H}{K+P} \right\rfloor = \left\lfloor \frac{1024+0}{12+6} \right\rfloor = 56$ entries/block

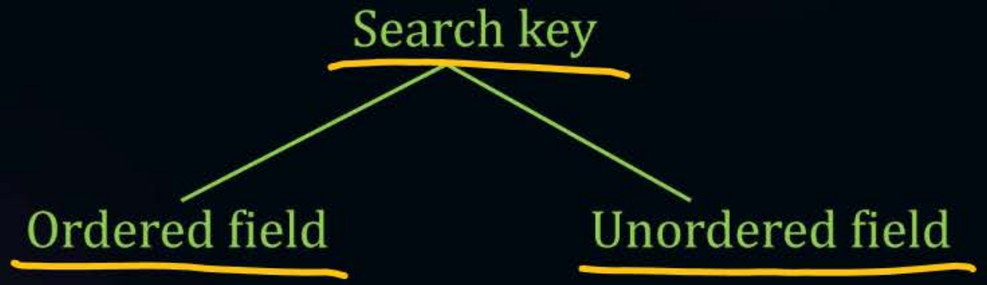
No. of index entries = No. of block of DB

So, Sparse index blocks $= \left\lceil \frac{5000}{56} \right\rceil = 90$ (ii) Number of searches in sparse index $= \lceil \log_2 90 \rceil \approx 7$ blocks

So, total number of blocks required to access a record is 8

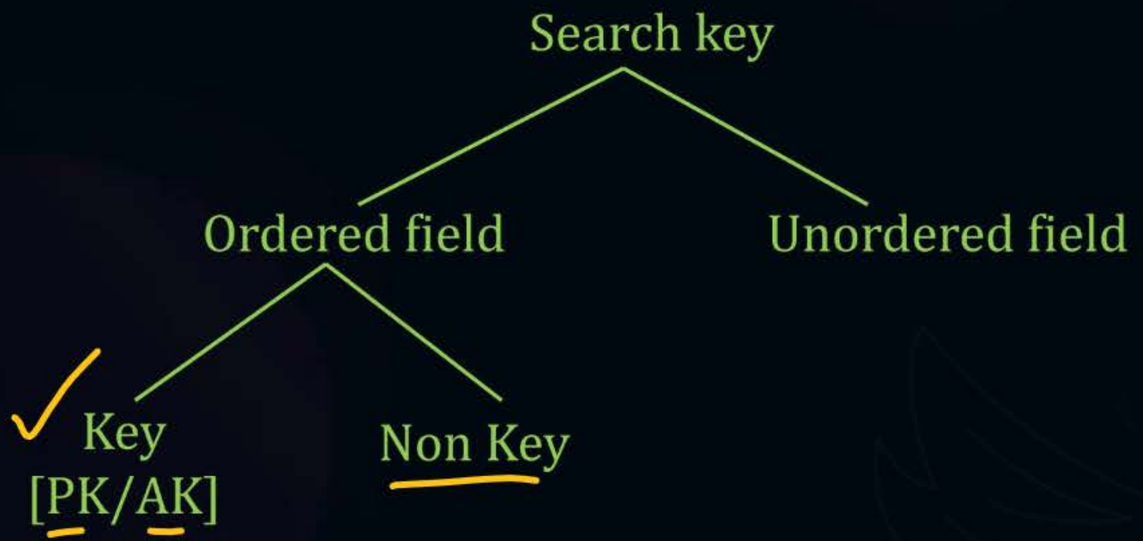
Topic: Types of Index

Any fields/attributes used for indexing is called search Key.



Topic: Types of Index

Any fields/attributes used for indexing is called search Key.

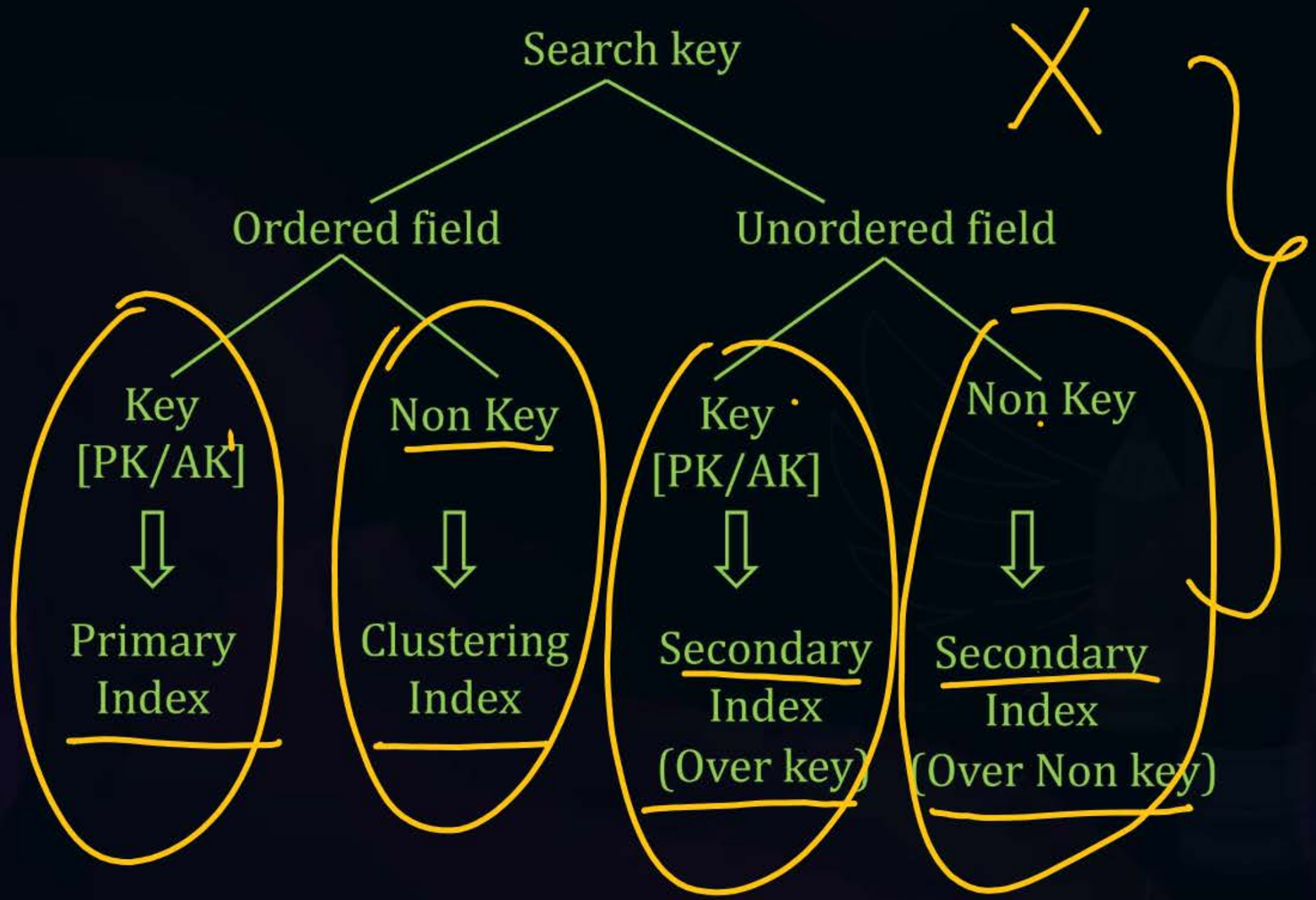


Topic: Types of Index

Any fields/attributes used for indexing is called search Key.

Topic: Types of Index

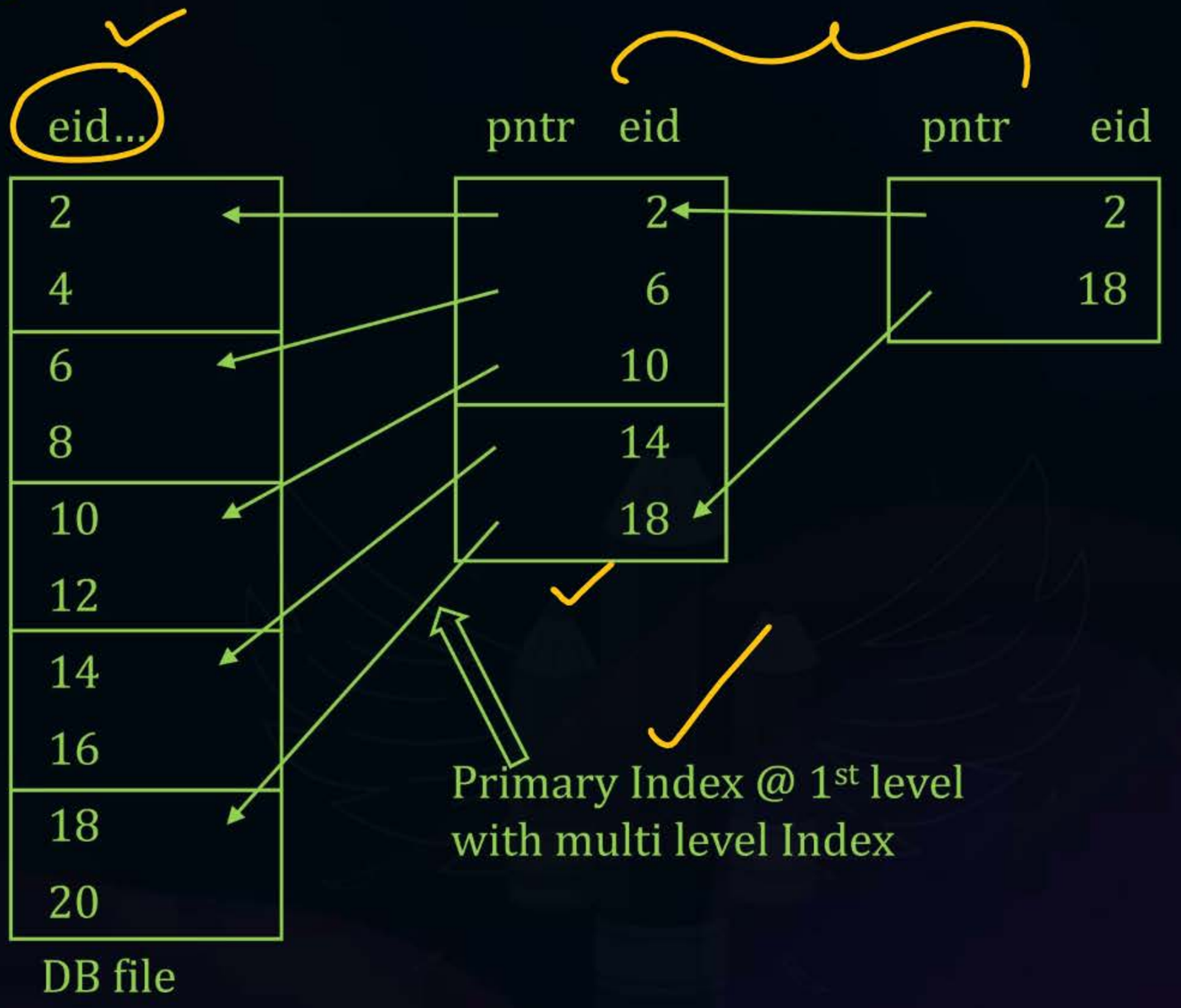
Any fields/attributes used for indexing is called search Key.



Topic: Primary Index

(PI → Primary Index)

- Ordered field and key.

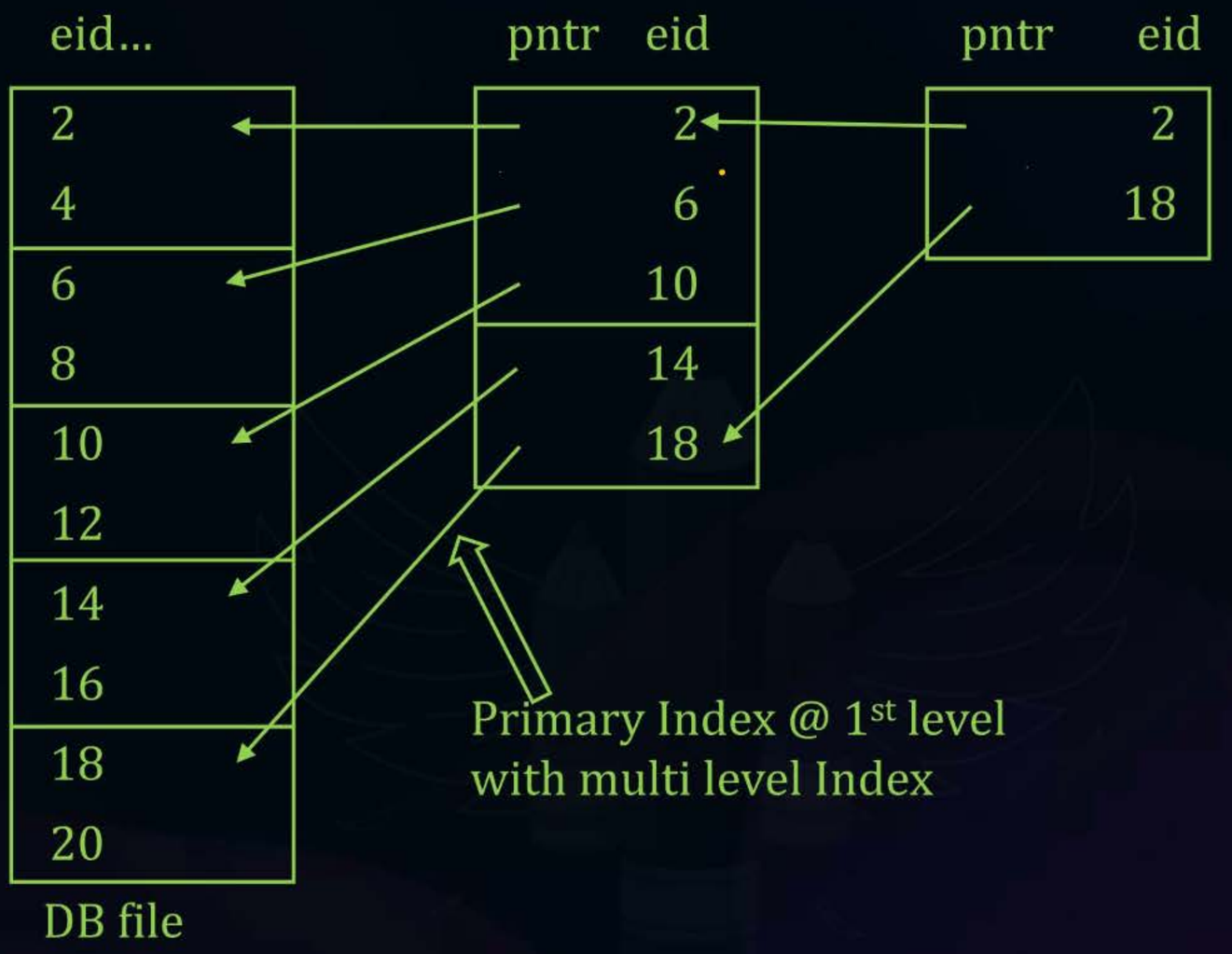


Topic: Primary Index

(PI → Primary Index)

- Ordered field and key.

⇒ I/O cost PI with MLI: K+1 blocks



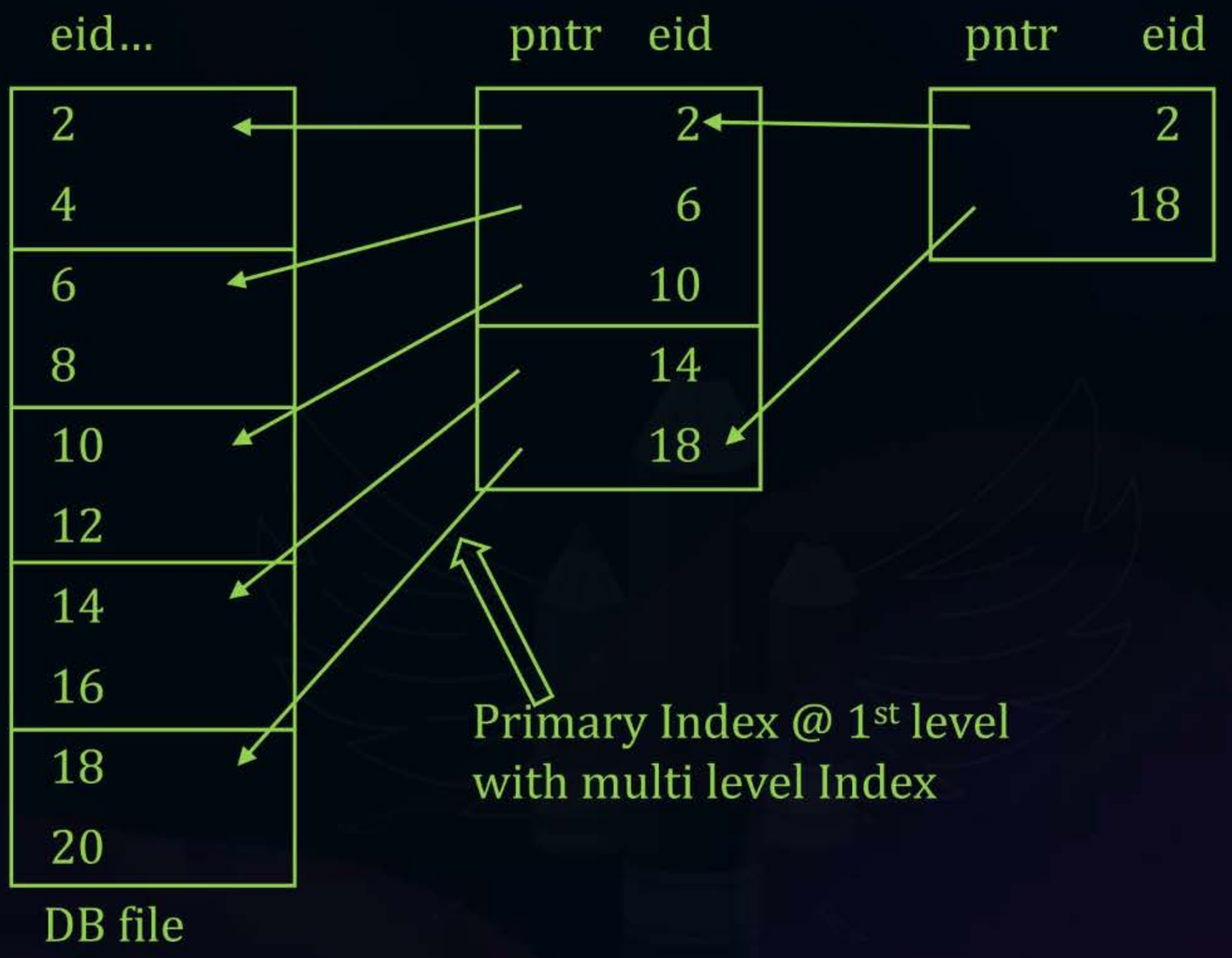
Topic: Primary Index

(PI → Primary Index)

- Ordered field and key. ✓

⇒ I/O cost PI with MLI:
K+1 blocks

⇒ PI can be dense / sparse
(sparse PI is preferred)



Topic: Primary Index

(PI → Primary Index)

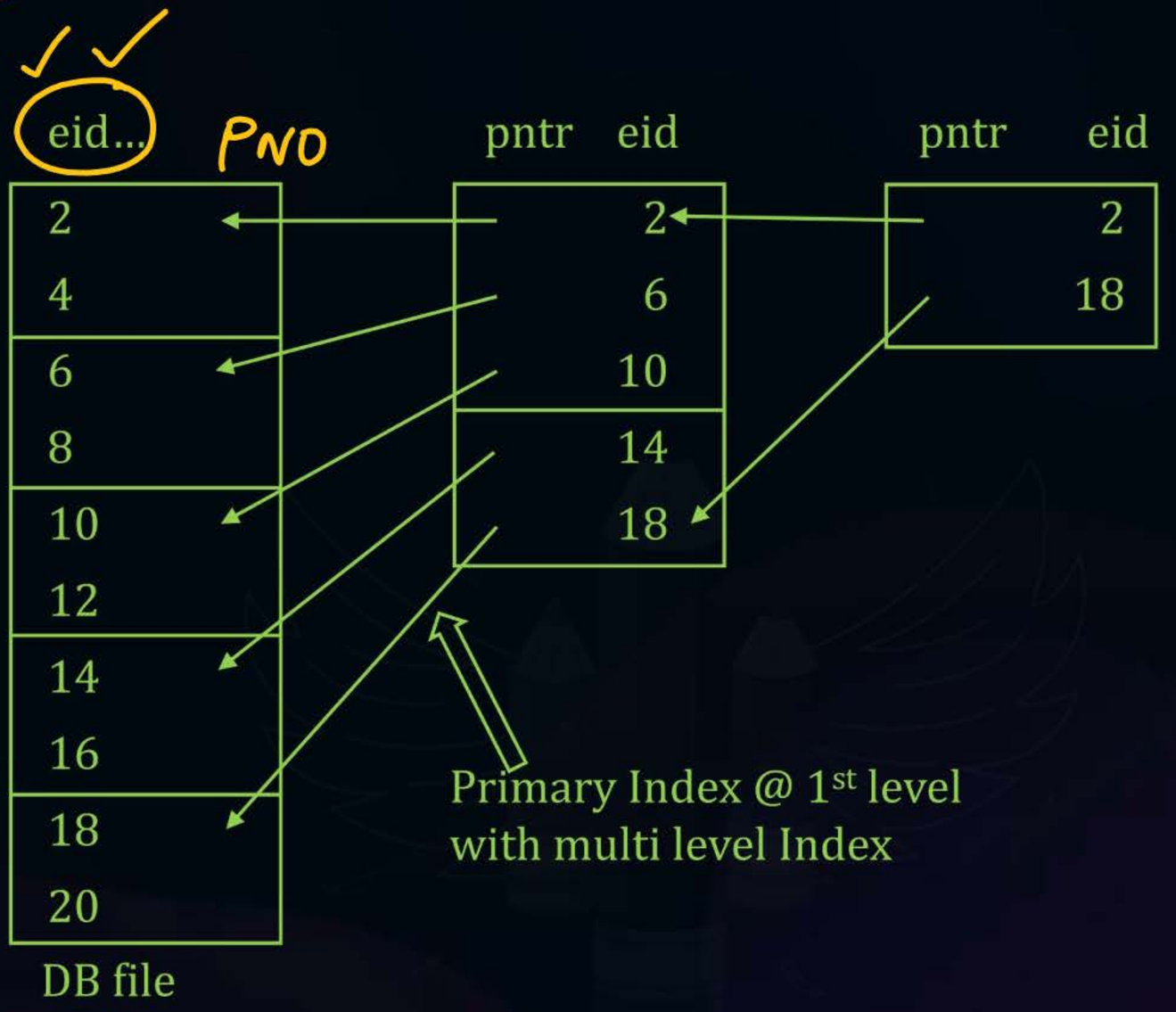
- Ordered field and key.

⇒ I/O cost PI with MLI:
K+1 blocks

⇒ PI can be dense / sparse
(sparse PI is preferred)

⇒ Atmost one PI for any DB file.

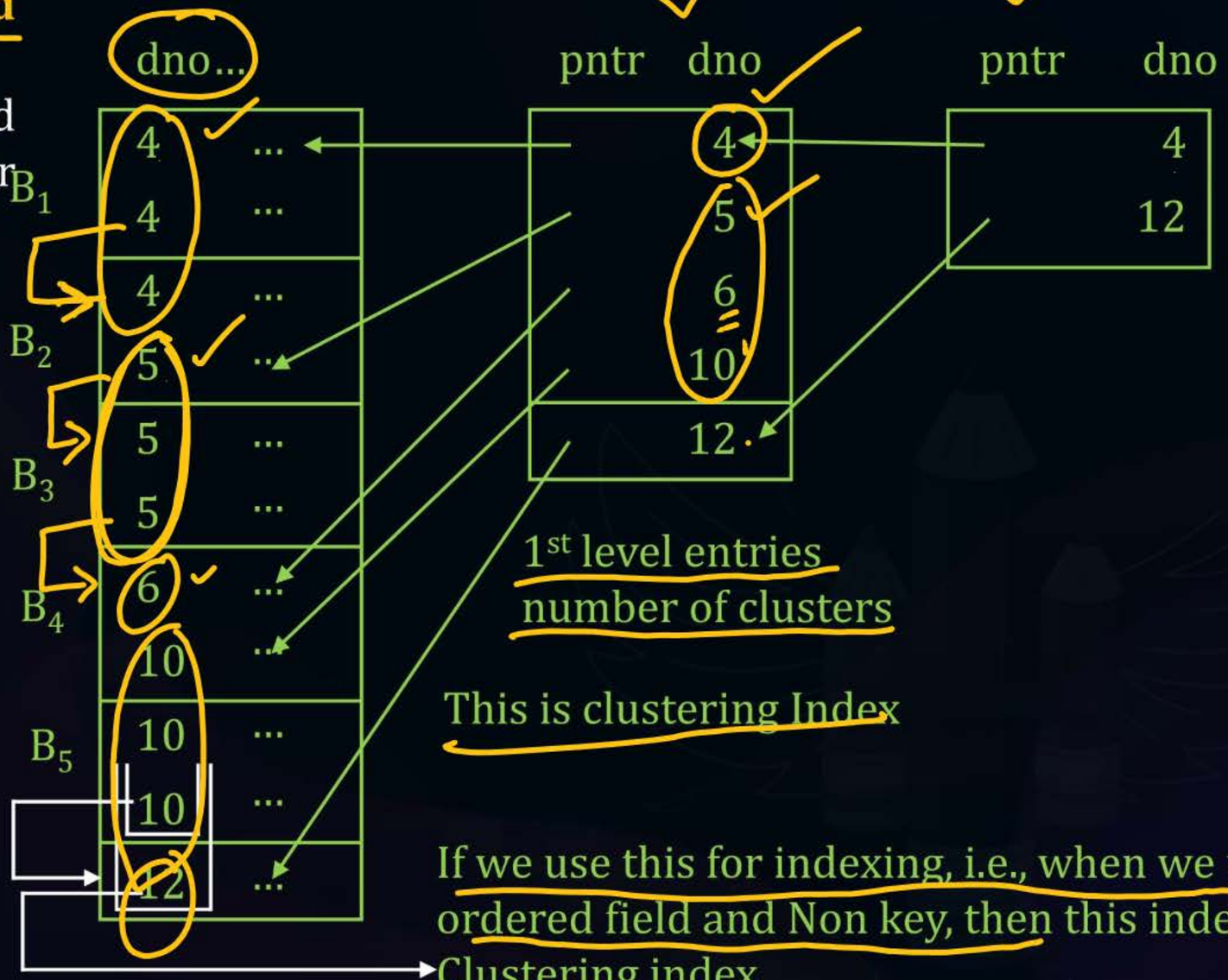
⇒ So, only one PI is possible



Topic: Clustering Index

Clustering Index [Ordered field & Non key]

Let's say records are ordered based on the department number (dno),
but dno is not CK of the table.



Topic: Secondary Index

Secondary Index

Index for unordered field and (Key/Non Key)

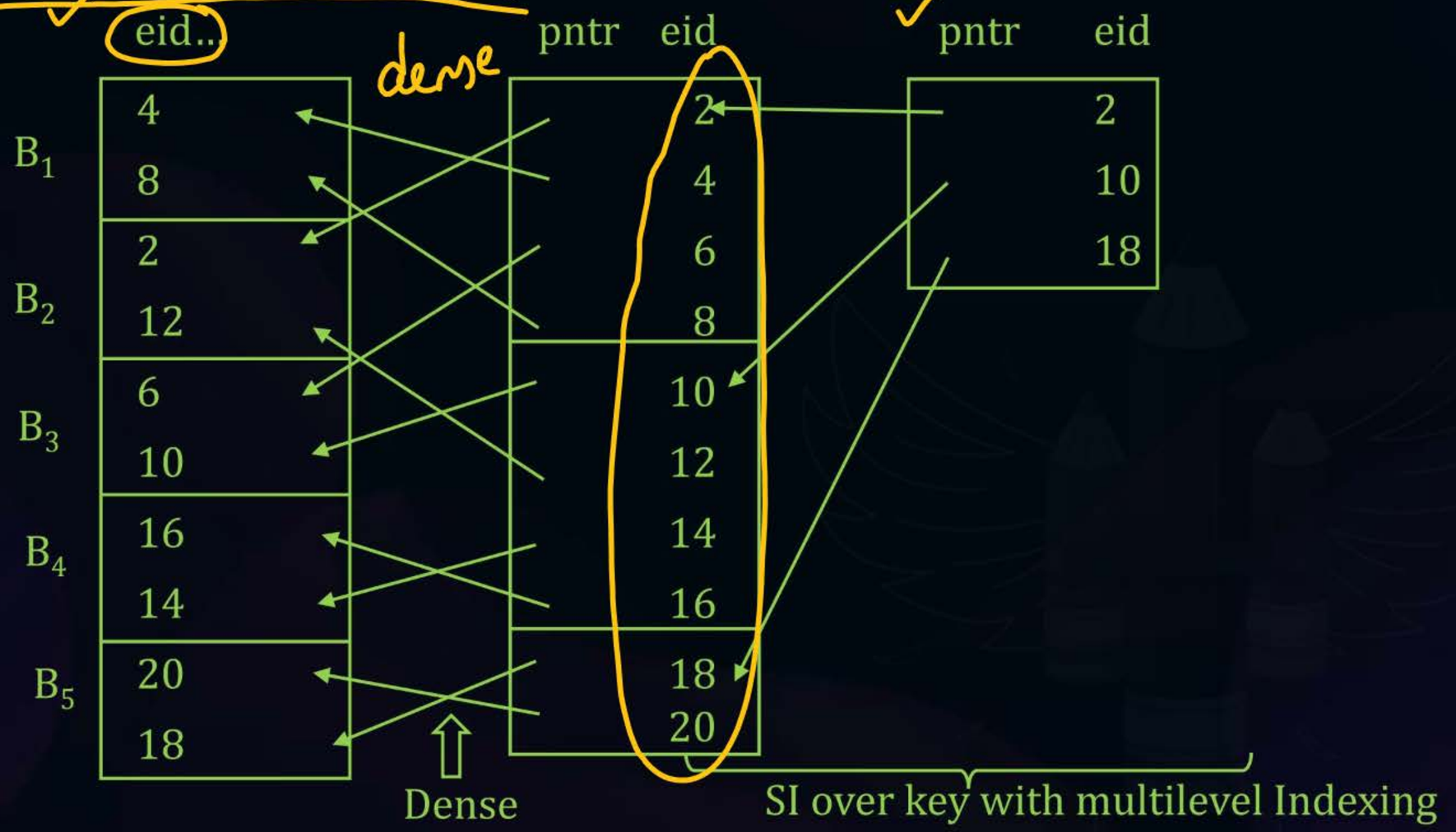
NOTE that many secondary index possible for DB file unlike primary index and clustering index, as we say, "atmost one Primary index, atmost one clustering index". But secondary index is not like that, we can have multiple secondary index. WE can build many secondary Index as per our requirement to access data.

⇒ At most one Primary Index, At most one Clustering index but Many Secondary Index are possible.

Topic: Secondary Index

Secondary index with key:

When we have unordered field and key.



Topic: Secondary Index

⇒ Secondary Index over key is always Dense.

Why?

Because we have unordered field, there is no way to point them as a group. we will have to point to each record individually.

But from 2nd level, we can have Sparse Index since the index file is has eid in ordered fashion

⇒ I/O cost with SI over Key & MLI (Multi-level Indexing) is (K+1) blocks.

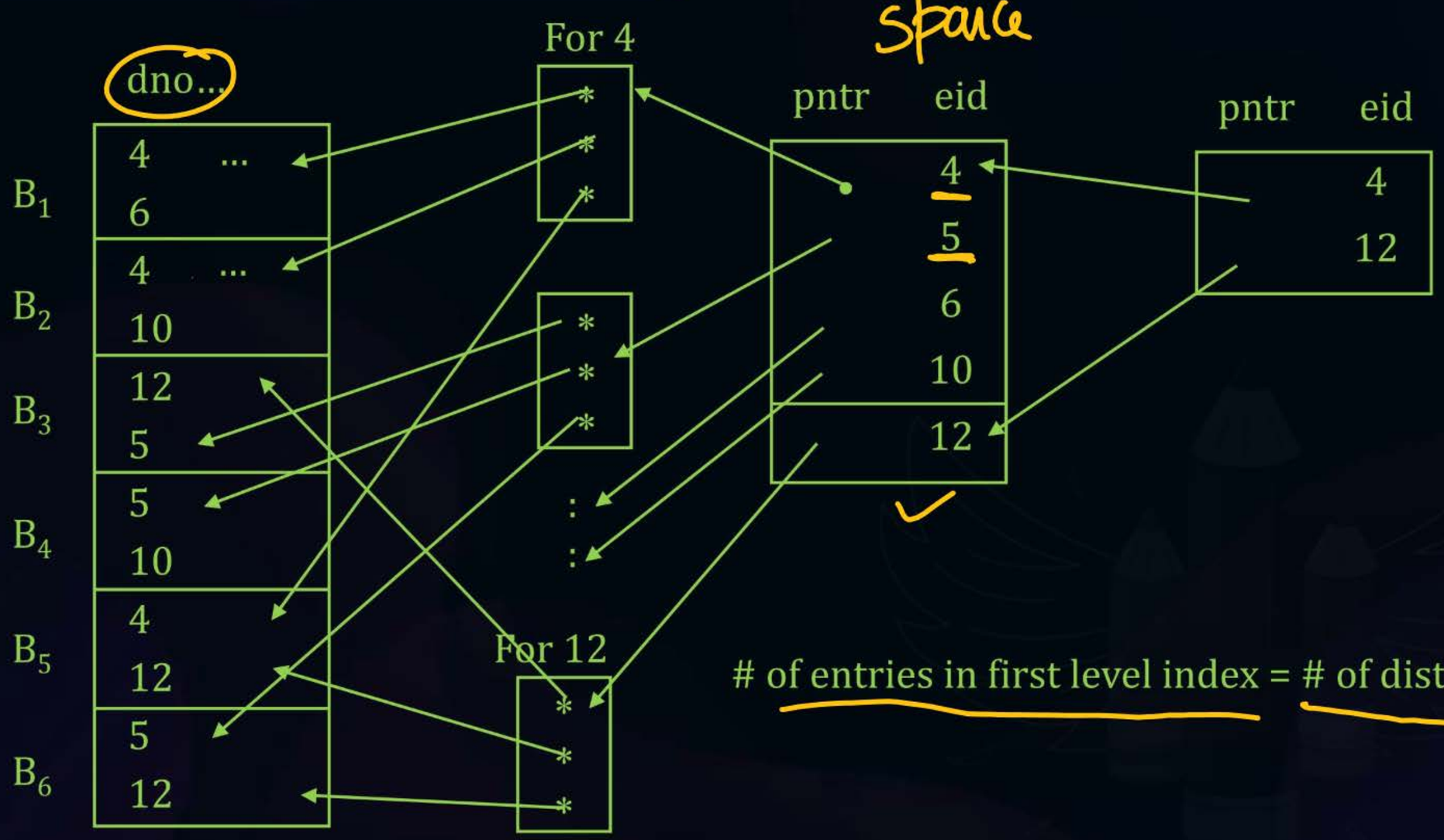
∵ To access a record, we will have to go through all K levels, then access content from DB file, hence, (K+1) blocks.

Topic: Secondary Index

Secondary Index over Non Key:

⇒ when we have unordered field and Non Key.

Topic: Secondary Index



# of entries in first level index = # of distinct values

Unordered field +non key

Redirect Pages
(REDIRECT PAGES)

Topic: Secondary Index

Since we have non key, and unordered file, a record with same search key value can be present in multiple blocks, it can be in middle, top or bottom. So, to keep a record of their location we maintain Redirect Pages, which has pointers to all the records of 4.

For example-

Index file just have to point to redirect page of 4, and that page will point to all the records of 4 in DB file.

⇒ SI over unordered non-Key is mostly sparse index, Why?

Because observe from diagram, there are 12 records in DB file but 5 index entries in index file. # index entries < # record in DB file

Topic: Secondary Index

⇒ Primary Index and secondary index over-Key looks almost similar, only difference- Primary index, 1st level is SPARSE index.
Secondary index over Key, 1st level is DENSE index.

⇒ Clustering Index and Secondary index over non Key is almost similar, Only difference:

- Clustering Index, Index points directly to Database.
- Secondary Index over non Key, Index points directly to Redirect pages, which then points to DB file.

Inspiring Stories : Annette Herfkens

Background: **Dutch woman on plane in Vietnam.**

Struggle: **Plane crashed in jungle, everyone else died..**

Achievements: **Survived 8 days with broken bones, drinking rainwater.**

Impact: **Only survivor of 30 passengers.**

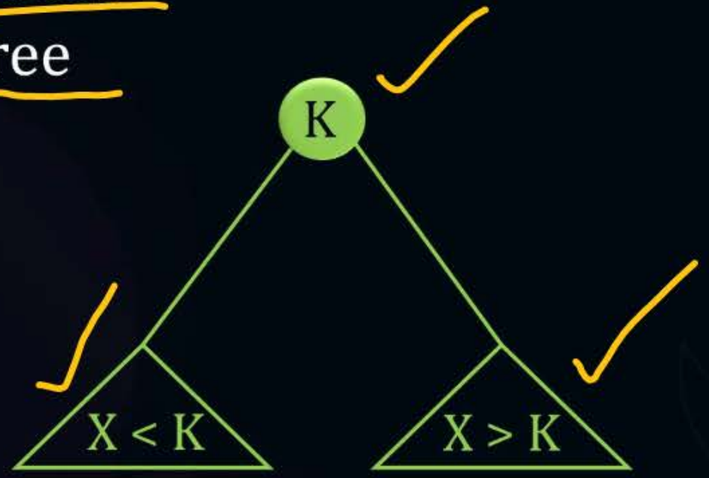
Topic: Dynamic Multilevel Index

To Insert, Delete or Update using Normal indexing takes more cost, so we use Dynamic Multi-level indexing like

B – Trees and B^+ - Trees

B→ Balanced Search Tree

⇒ Search Tree:

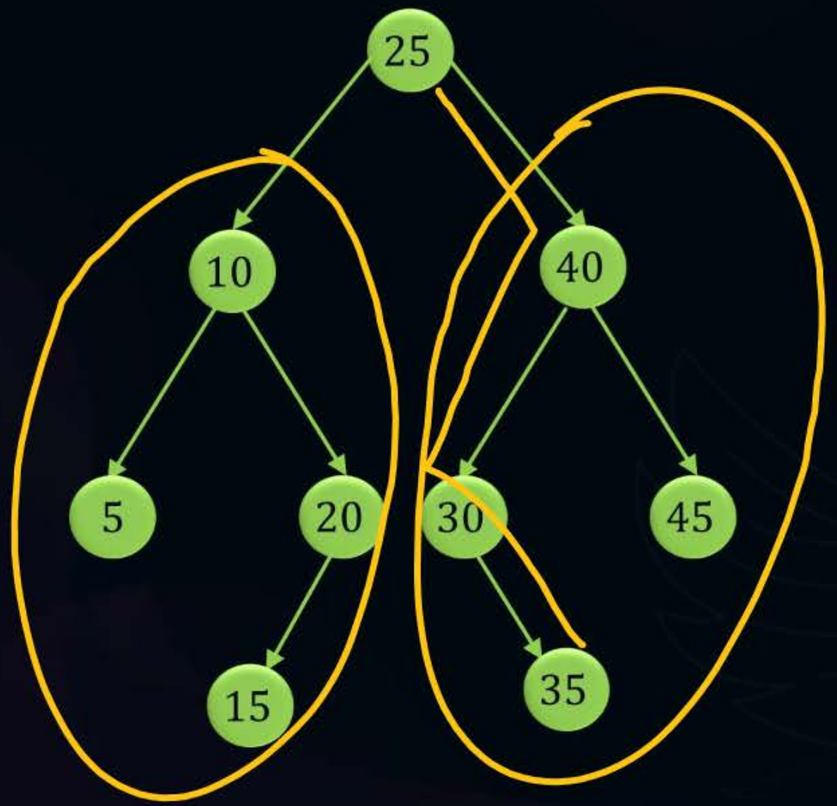


Keys of left subtree < K < keys of right subtree

Topic: Dynamic Multilevel Index

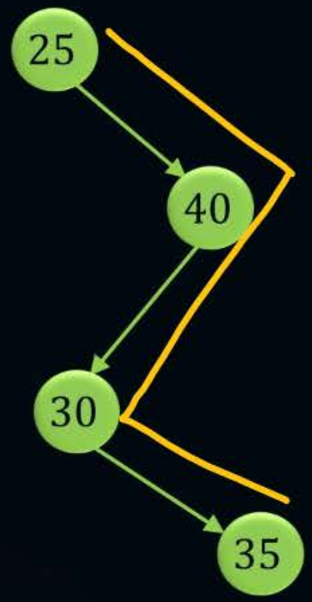
Example:

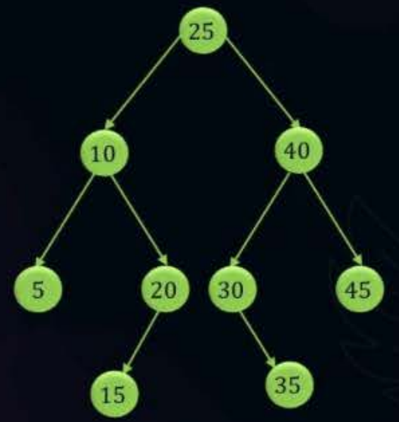
Binary Search Tree:



Topic: Dynamic Multilevel Index

- We find out whether 35 is present on the BST, by accessing one node at a time.





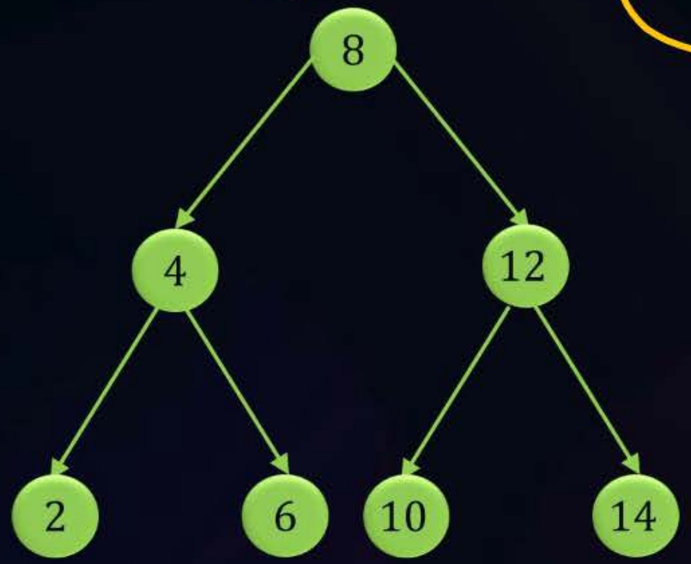
- 25, 40, 30, 35 is a Probe Sequence. we access one node from each level to check if element is present on the tree.

Topic: Dynamic Multilevel Index

Height of search Tree for n distinct Keys:

Min height
$O(\log_2 n)$

Max height
$O(n)$



Search cost O(logn)

Search cost O(n)

Topic: Dynamic Multilevel Index

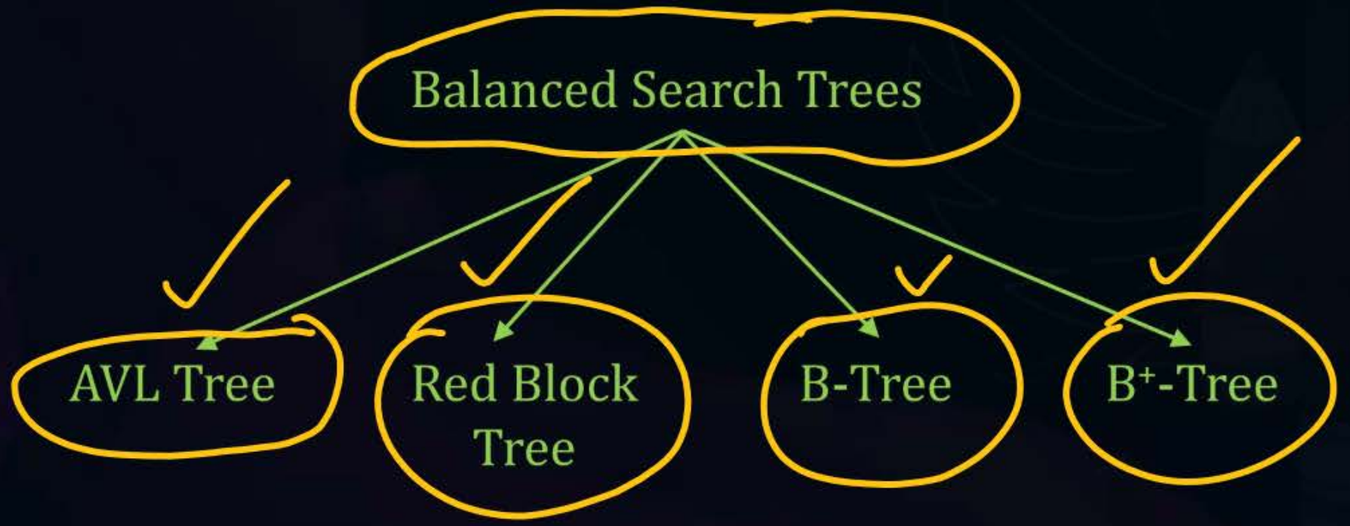
Disadvantage:

Worst case Search cost to perform Search of Key : O(n)

Balanced Search Tree: (Height restricted Search Tree)

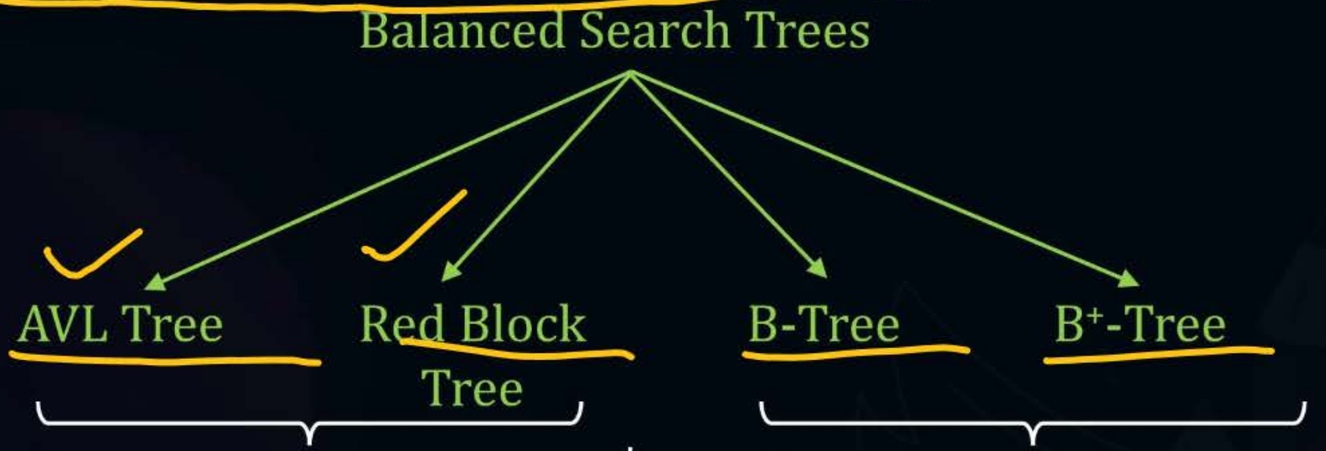
⇒ Maximum height of Search Tree should not exceed O(log n) for n distinct Keys.

⇒ In this case, the worst case search cost would be O(log n).

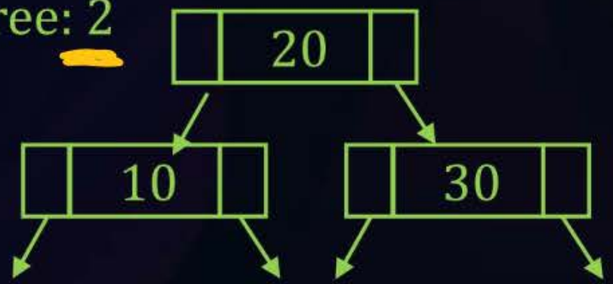


Topic: Dynamic Multilevel Index

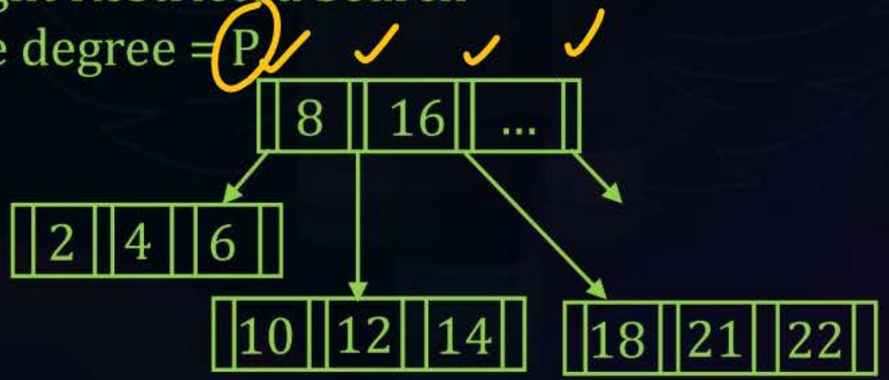
⇒ All of these many types of Balanced Search trees but All of them have one thing in common, they all are Height restricted Search Tree.



Height restricted Binary Search Trees.
degree: 2



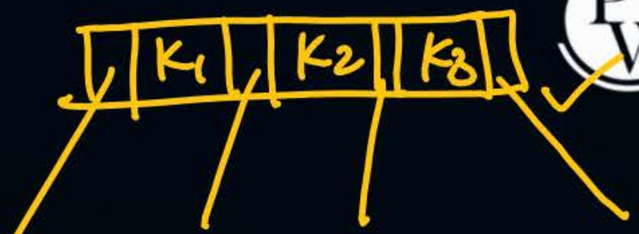
Height Restricted Search Tree degree = P



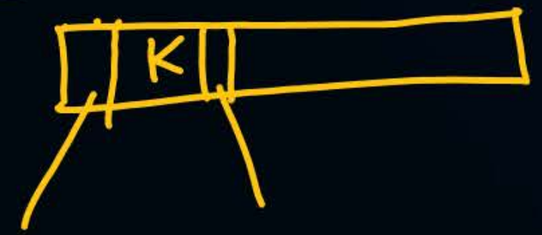
Topic: Dynamic Multilevel Index

⇒ For DB index, B and B+ trees are preferred because degree of nodes is higher i.e. P can be any number.

Topic: Dynamic Multilevel Index



#Q. Why B/B+ trees are preferred for DB index rather than balanced Binary Search Tree (AVL Tree)?



Sol:

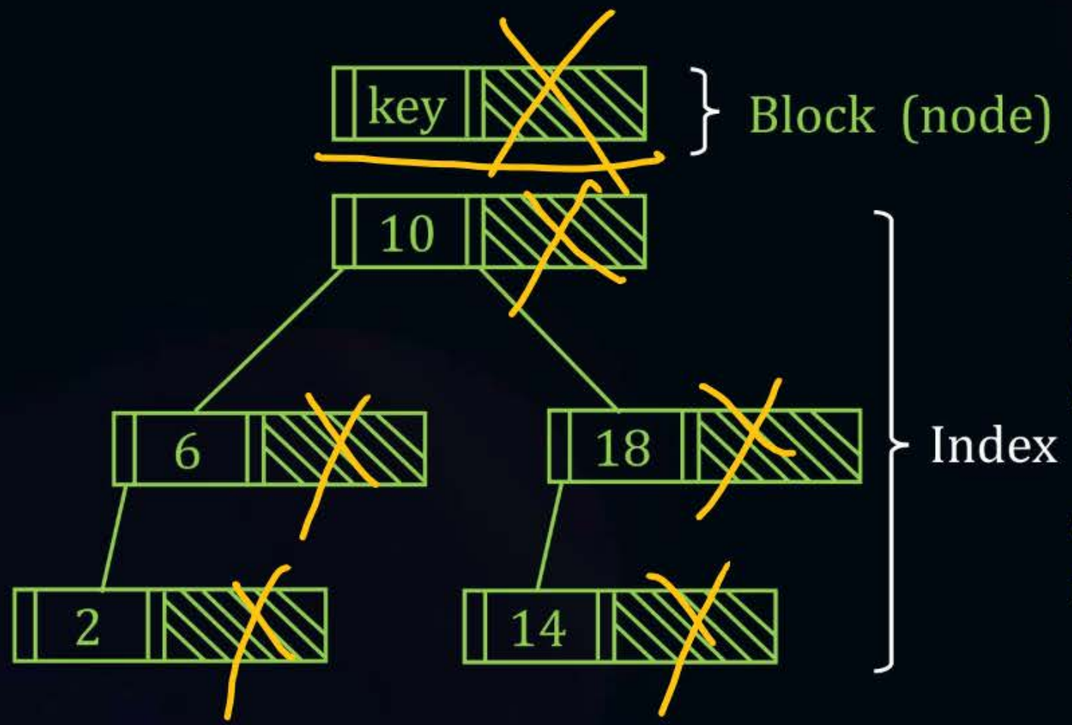
⇒ Data access from DISK happens block by block.

⇒ DB file must be stored in DISK & index to DB file must be in DISK. We can't have DB file on DISK and index file on Main Memory and vice versa.

⇒ If Balanced "Binary" search Tree is used for DB index, then

- we will have more I/O cost.
- In Balanced Binary Search Tree, one node can only store up to one key. So, if DB file has data with 5 keys, then there has to be 5 nodes to store the 5 keys.

Topic: Dynamic Multilevel Index



More levels of index is required, since one index can only accommodate one Key, since more levels

are required, the I/O cost increases.

Topic: Dynamic Multilevel Index

- If B/B$^+$ trees used for DB index

⇒ In case of B/B+ trees, each node is capable of accommodating many keys, and because we can accommodate many keys less levels of index is required, means less I/O cost.

⇒ I/O cost depends on no. of levels of index.

Conclusion!

⇒ This was the reason to use B/B+ trees.

⇒ B/B+ trees less levels required because one node can accommodate many keys.

⇒ The overall I/O cost reduces.

Topic: Dynamic Multilevel Index

Which data structure we use for search depend on type of memory we are dealing with

Search tree application
(DS used for search)

MM application main memory

- DS stored is MM used for search

(DB index)
SM application (Secondary memory)

- DS sorted in DISK, DISK block should access based on Search

Topic: Dynamic Multilevel Index

Which data structure we use for search depend on type of memory we are dealing with

Search tree application
(DS used for search)

MM application main memory
- DS stored is MM used for search

1. Less number of keys
   n : 100's
2. Huge no. of keys
   n : lakh's

(DB index)
SM application (Secondary memory)
- DS sorted in DISK, DISK block should access based on Search

3. Less no. of keys
   n : 100's
4. Huge no. of blocks
   n : B^+ trees lakhs

Topic: Dynamic Multilevel Index

Which data structure we use for search depend on type of memory we are dealing with

Search tree application
(DS used for search)

MM application main memory
- DS stored is MM used for search

1. Less number of keys
   n : 100's
   ⇩
   AVL Tree Best

2. Huge no. of keys
   n : lakh's
   ⇩
   B Tree Best

(DB index)
SM application (Secondary memory)
- DS sorted in DISK, DISK block should access based on Search

3. Less no. of keys
   n : 100's
   ⇩
   B^+ Tree Best

4. Huge no. of blocks
   n : B^+ trees lakhs
   ⇩
   B^+ Tree Best

Less I/O cost

Topic: Dynamic Multilevel Index

⇒ when we are dealing with MM application, with less keys we can go with AVL Tree,

⇒ when we have MM applications with huge no. of keys, we will use B Trees.

⇒ when we have SM application, we need to be mindful of I/O cost and because of that we use B+ trees for SM application, regardless of whether no. of Keys is more or less.

B^+

Inspiring Stories : Nguyen Van

Background: **Vietnamese farmer.**

Struggle: **Lost his hair young, so he stopped cutting it.**

Achievements: **Carried a hair braid 5 meters long, 10 kg heavy for 80 years, never cut it once.**

Impact: **Became a symbol of patience, discipline, and living true to belief.**

THANK - YOU