File organization

Database

Database

Database
Is a collection of files

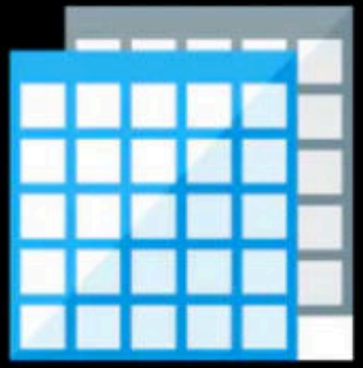
Database

Database
Is a collection of files

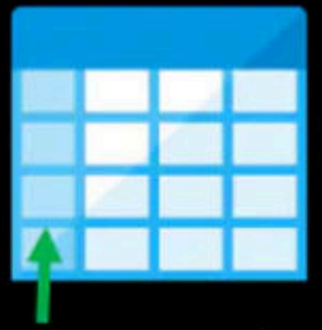
Files
are collection of records

Database

Database
Is a collection of files



Files
are collection of records



Record
is collection of fields

- A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.
- Blocking factor is the average number of records per block.
- Suppose the size of the block is 1024 B and size of record is 4B

Then,

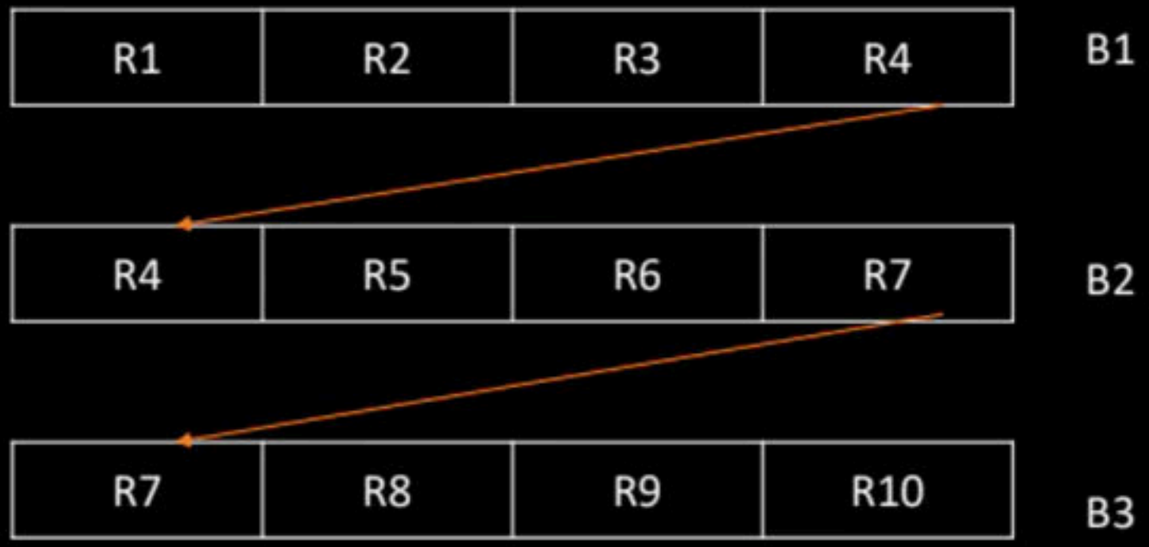
Blocking factor = 1024 / 4 = 2^8 = 256

These many records can be stored in one block

Strategies of storing file of records into block:

1.)Spanned Strategy:

It allows partial part of record to be stored in a block.



Advantage : **No wastage of memory**

Disadvantage : **Number of block accesses to access a record increases**

This strategy is suitable for variable length records

Strategies of storing file of records into block:

1.)Unspanned Strategy:

No record can be stored in more than 1 block.

Advantage : **Number of block accesses to access reduced to access a record**

Disadvantage : **Wastage of memory**

This strategy is suitable for fixed length records

Organization of records in a file:

1.) Ordered file organization:

All records in a file are ordered on some search value

Searching : Binary Search

Advantage : Searching a record is efficient.

Disadvantage : Insertion is expensive due to reorganization of the entire file.

Organization of records in a file:

1.) Unordered file organization:

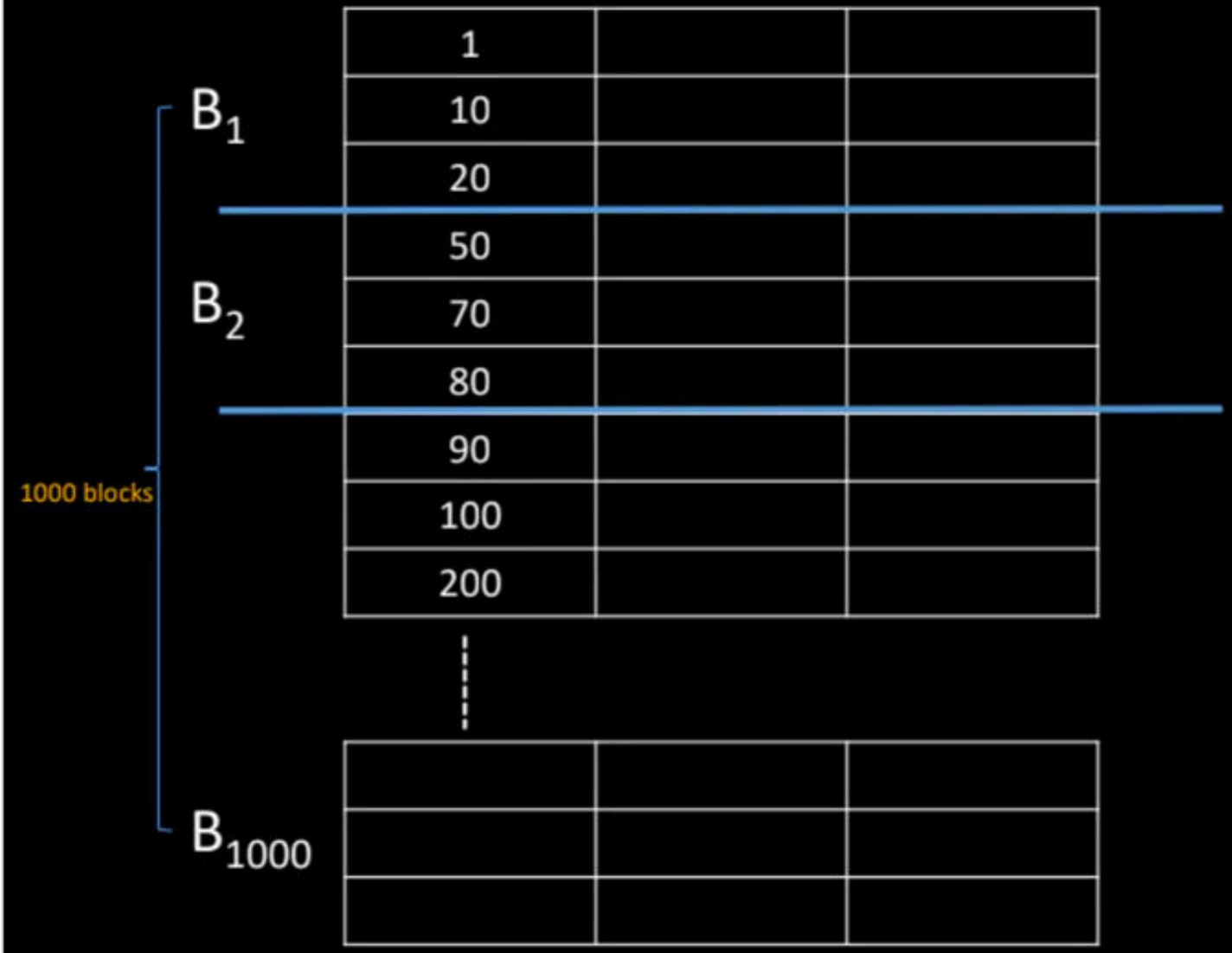
All records in a file are inserted where ever the place is available , usually at the end of file.

Searching : Linear Search

Advantage : Inserting a record is efficient.

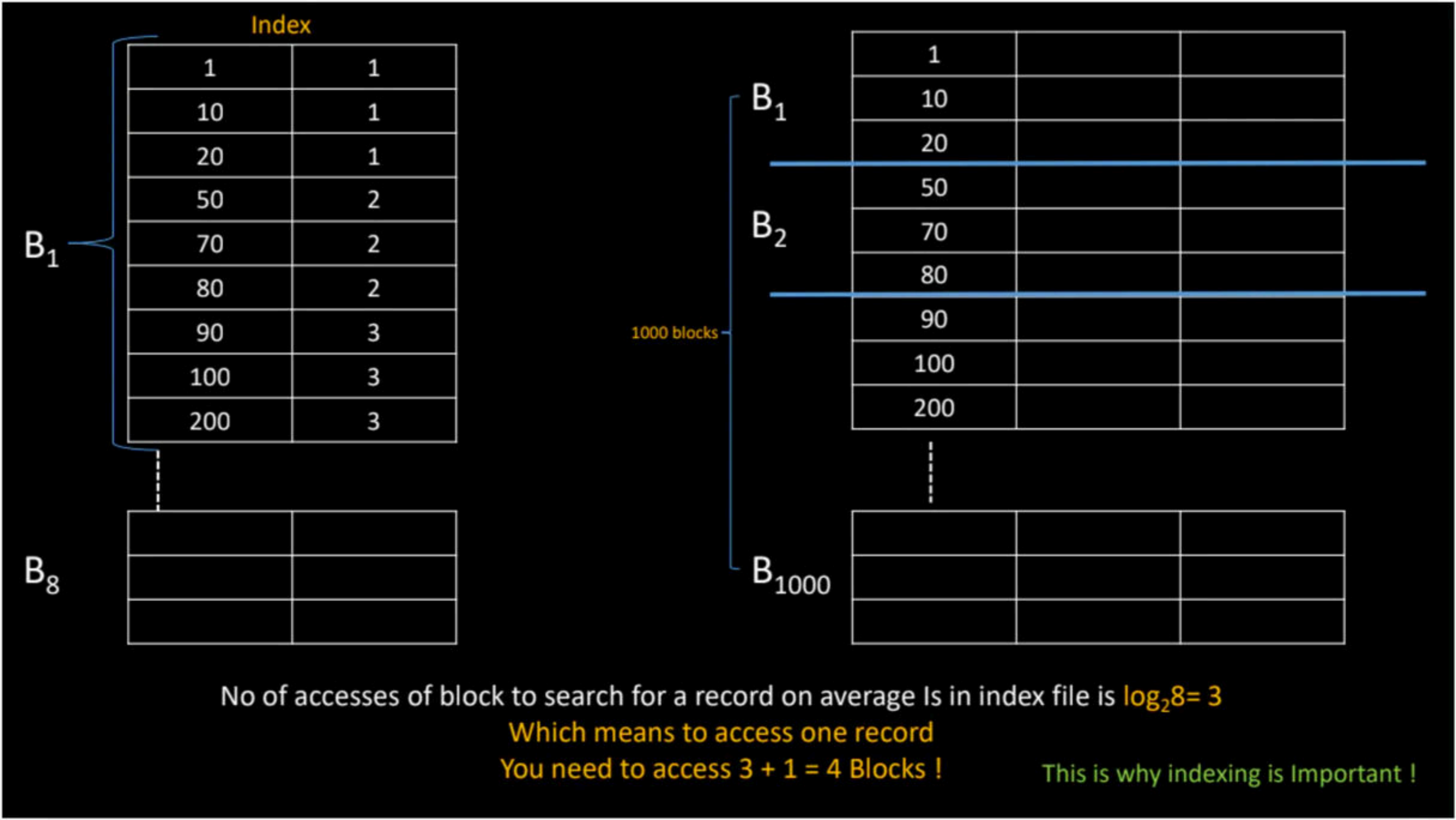
Disadvantage : Searching a record is inefficient compared to ordered file organization

Introduction to Indexing



No of accesses of blocks to search for a record (without index) on average is $\log_2 1000 = 10$

Which means to access one record You need to access 10 Blocks !



No of accesses of block to search for a record on average Is in index file is $\log_2 8 = 3$

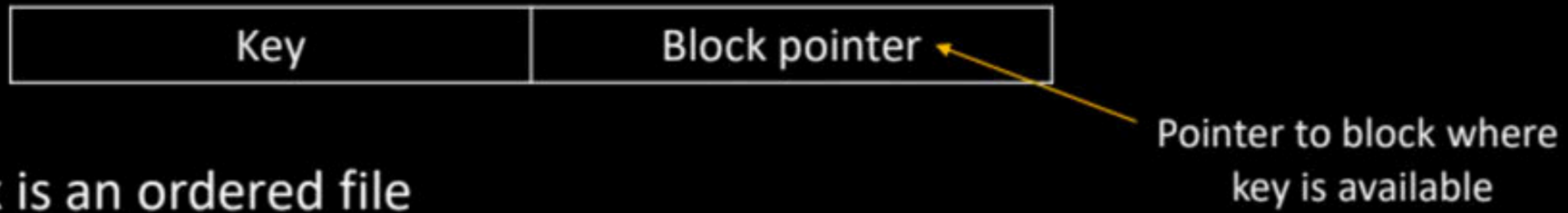
Which means to access one record

You need to access 3 + 1 = 4 Blocks !

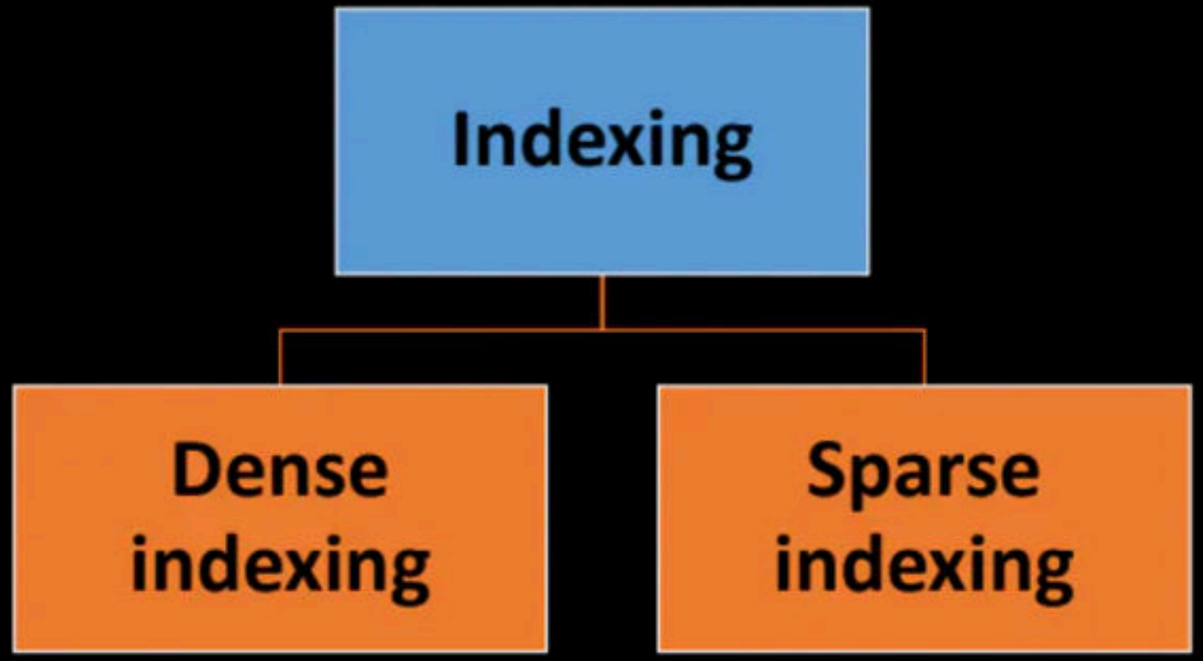
This is why indexing is Important !

Structure of index file

- Index record consists of 2 fields

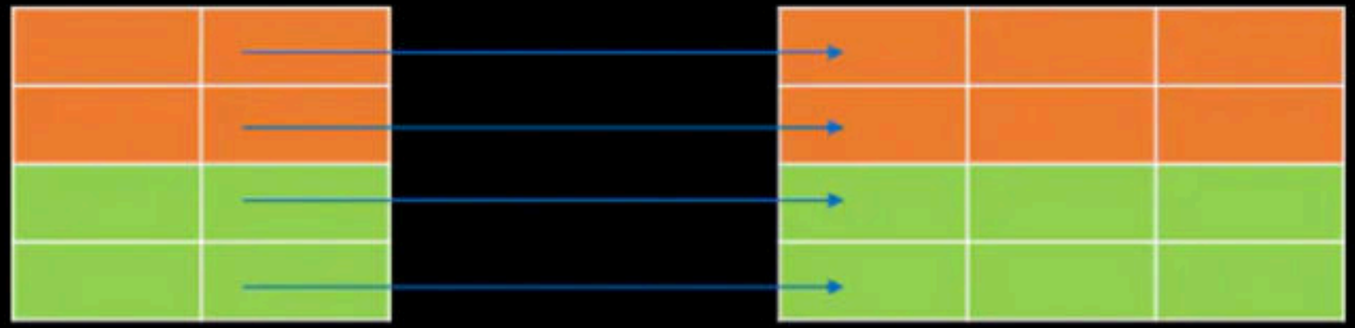


- Index is an ordered file
- Searching: Binary Search
- To access a record using index, the average number of block accesses $= \log_2 B_i + 1$
- Index can be created on any field of a relation (Primary key, Non key, candidate key)


Indexing
Dense indexing
Sparse indexing

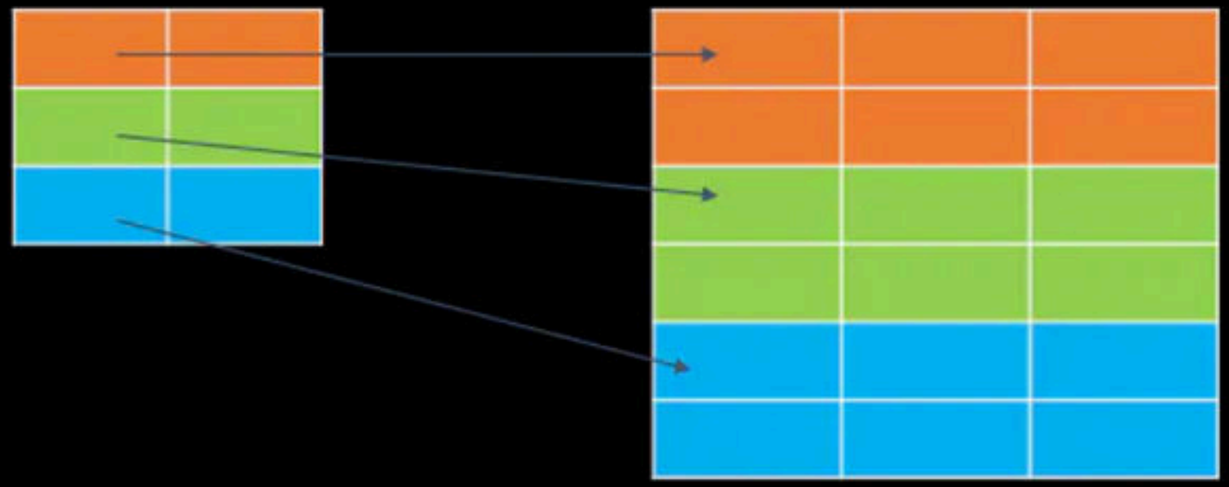
Dense index

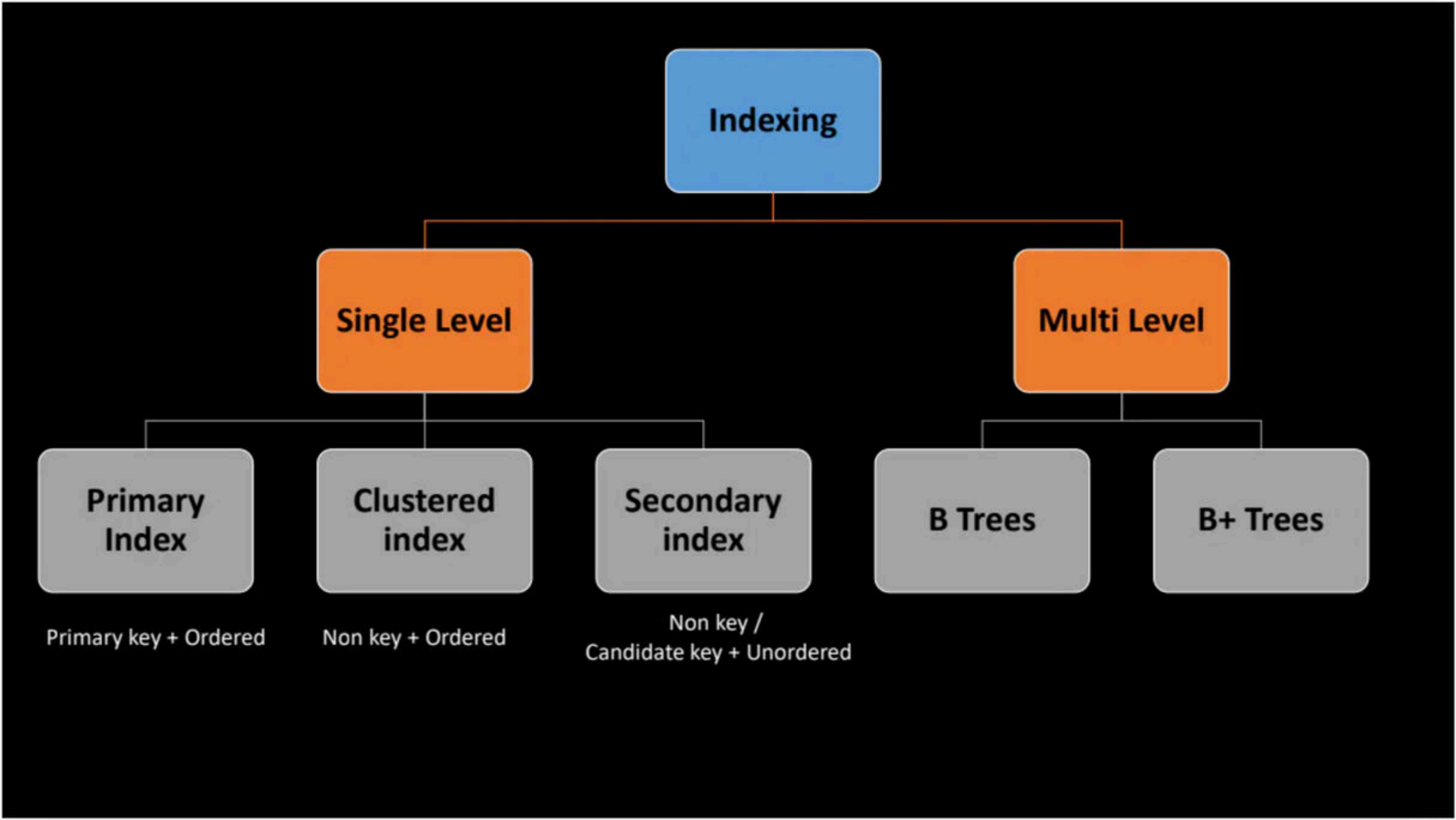
If an index entry is created for every search key value, then it is called dense index



Sparse index

If an index entry is created only for some records, then it is called sparse index




Indexing
Single Level
Multi Level
Primary Index
Clustered index
Secondary index
B Trees
B+ Trees
Primary key + Ordered
Non key + Ordered
Non key /
Candidate key + Unordered

Primary index (Primary key + ordered)

A primary index is an ordered file whose records are of fixed length with two fields.
The first field is same as primary key of data file and the second field is a pointer to data block,
Where the key is available.
Index created for first record of each block is called “block anchors”
The number of index entries is equal to the number of data blocks.
The average number of block accesses using index = $\log_2 Bi+1$

Bi is number of index blocks

Suppose that we have an ordered file of 30,000 records and these records are stored on a disk and block size is 1024 bytes files records of fixed length and unspanned of size 100 byte and suppose that we have created a primary index on key field of size 9 bytes and a block pointer of size 6 bytes then find the average number of block access required with or without index?

Block size=1024B
Record Size=100B
No of records per block= 1024/100=10.24 (but we can store only 10 records at max)
No of blocks required to store 30000 records=30000/10= 3000

Without Primary Index: No of block accesses= log(3000)= 12

Size of a Index record= 9+6=15B

No of index record per block= 1024/15= 68 (unspanned)

We have already calculated no of blocks needed to store all records= 3000

So total no of index records =3000

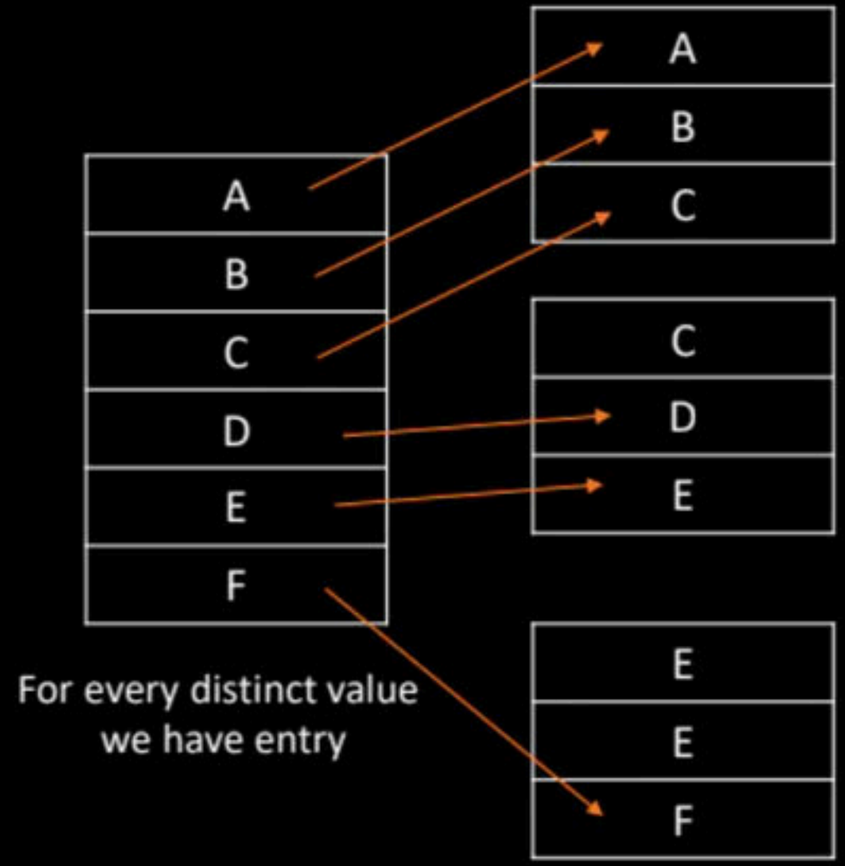
Since, 68 index records are present in 1 block.

3000 index records will be present in 3000/68= 45 blocks.

With primary index: Total no of block accesses =ceil(log(45))+1=6+1=7

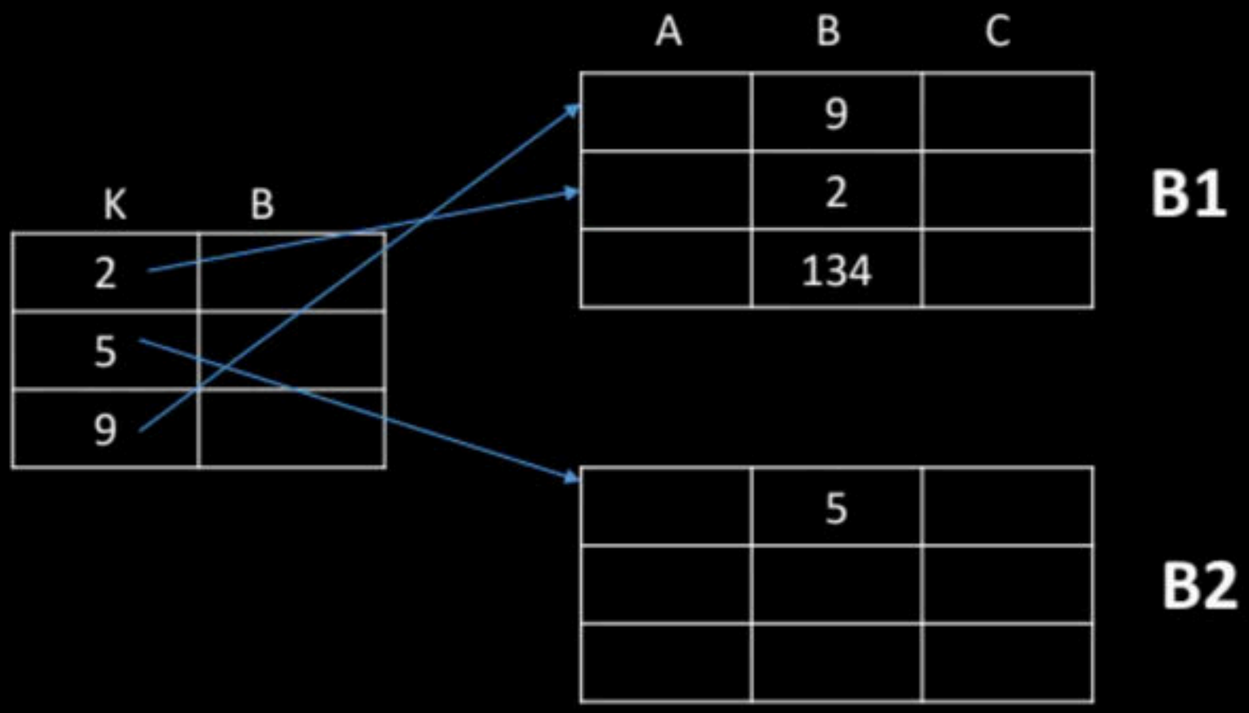
Clustering index

Clustering index is created on data file whose file records are physically ordered on a non key field which does not have distinct value for each record (that field is called clustering field).



The number of block accesses using clustering index >=$\log_2 Bi + 1$

Secondary index



- Secondary index provides a secondary means of accessing a file for which primary access already exists.
- Index created for each record in a file.
- Number of index entries = Number of records

Suppose that we have an ordered file of 30,000 records and these records are stored on a disk and block size is 1024 bytes files records of fixed length and unspanned of size 100 byte and suppose that we have created a secondary index on key field of size 9 bytes and a block pointer of size 6 bytes then find the average number of block access required with or without index?

Suppose that we have an ordered file of 30,000 records and these records are stored on a disk and block size is 1024 bytes files records of fixed length and unspanned of size 100 byte and suppose that we have created a secondary index on key field of size 9 bytes and a block pointer of size 6 bytes then find the average number of block access required with or without index?

BF=10 records/block; 3000 blocks of data

Secondary index with each entry = 15 bytes

So the BF = 1024/15 = 68 entries per block

This will be a dense index

30,000 entries with 68 entries/block: 30,000/68 = 442 blocks for the index

How many block accesses are required?

log 442 = 9 block accesses plus the additional one to access the data

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

A.)non-key and ordering

B.)non-key and non-ordering

C.)key and ordering

D.)key and non-ordering

GATE 2008

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

A.)non-key and ordering

B.)non-key and non-ordering

C.)key and ordering

D.)key and non-ordering

GATE 2008

Create index files, fields could be non-key attributes and which are in ordered form so as to form clusters easily.

An index is clustered, if

A.) it is on a set of fields that form a candidate key.

B.) it is on a set of fields that include the primary key.

C.) the data records of the file are organized in the same order as the data entries of the index.

D.) the data records of the file are organized not in the same order as the data entries of the index.

GATE 2013

An index is clustered, if

A.) it is on a set of fields that form a candidate key.

B.) it is on a set of fields that include the primary key.

C.) the data records of the file are organized in the same order as the data entries of the index.

D.) the data records of the file are organized not in the same order as the data entries of the index.

GATE 2013

Option (A) and Option (B) are not correct because, index can be created using any column or combination of columns, which need not be unique.

Basically, Indexed column is used to sort the rows of table. Whole data record of file is sorted using index so the correct option is (C).

B trees and B^+ trees

- Generalization of multilevel indexing.
- These are multilevel indexing with small modifications

Terminologies :

1.) Node Pointer / Block Pointer – Points to block (Node of a tree)

2.) Record Pointer – Points the record we are searching for

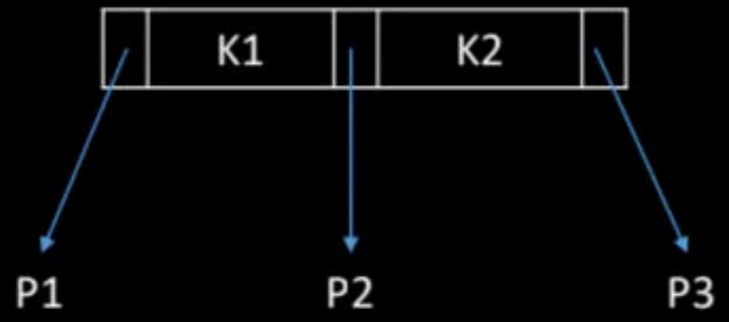
p is the order of tree

Order of tree means maximum number of children a node can have.

B+ trees

- Internal node has [p/2] to p children
- Internal node has [p/2 -1] to p-1 search keys

Case for n = 3

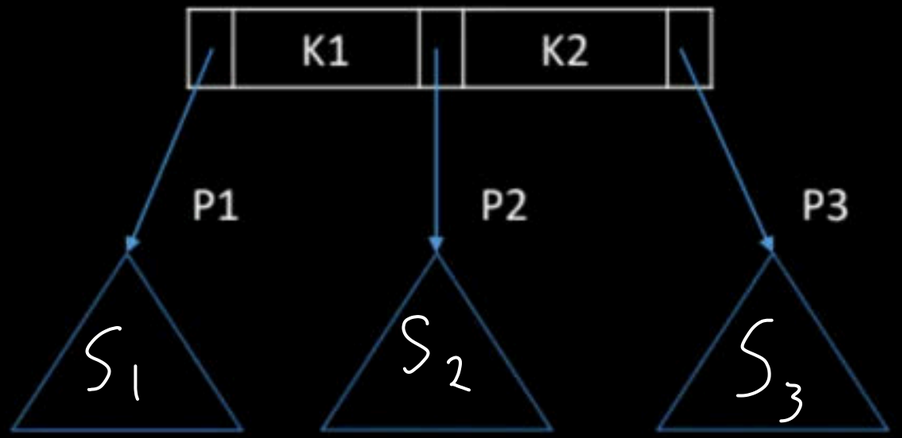


Non – leaf node :

Each key – search values in subtrees Si pointed by P_i, $K_i >= K_{i-1}$

Key Values in $S_1 < K_1$

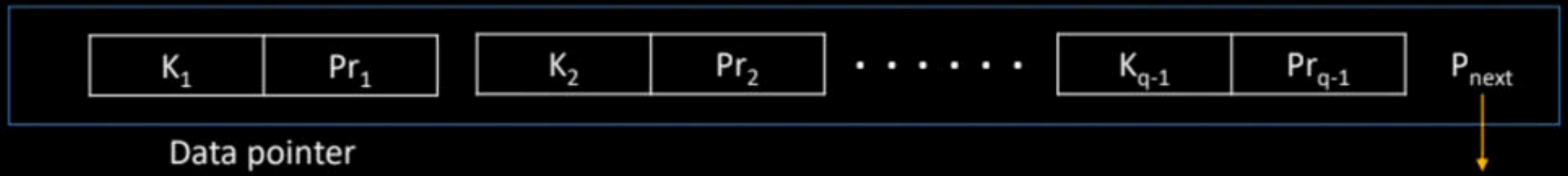
K_1 <=Key values in $S_2 < K_2$



Leaf node :

Each leaf node is of the form

$<<K_1,Pr_1>, <K_2,Pr_2>,\ldots\ldots, <K_{q-1},Pr_{q-1}><,P_{next}>$



Example 1 :

Search key field is 9 bytes, the block size is 512 B, a record pointer is 7B,
A block pointer is 6 bytes. What is the order of internal node and leaf node ?

For B+ tree with order n,
For a non leaf node it can be given that
$n*Pb + (n-1)*(k) <= B$

$n*6 + (n-1)*9 <= 512$
$n <= 34.77 = 34$ (Order of non leaf)

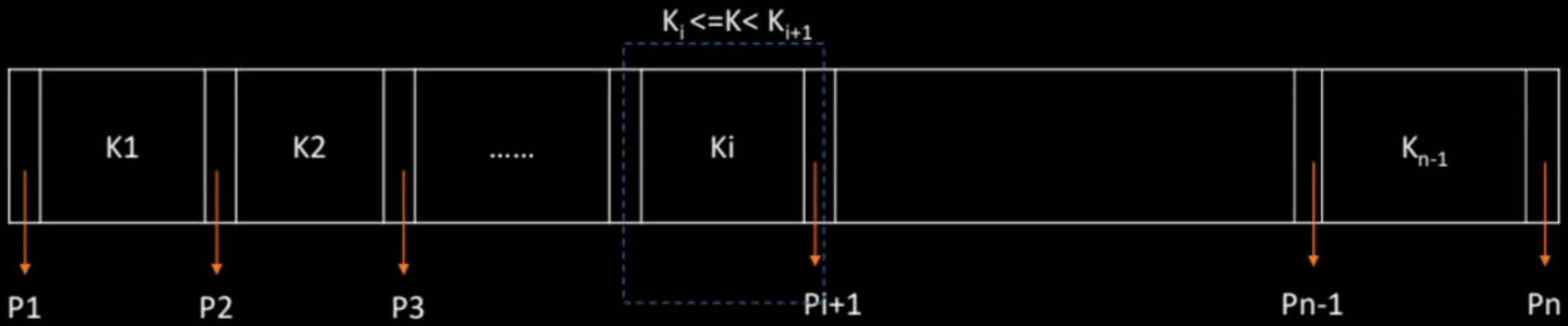
For leaf,
$m(k+Pr)+Pb <= B$

$m(9+7) + 6 <= 512$
$16m <= 506$
$M=506 / 16 = 31.62 = 31$ (Order of leaf)

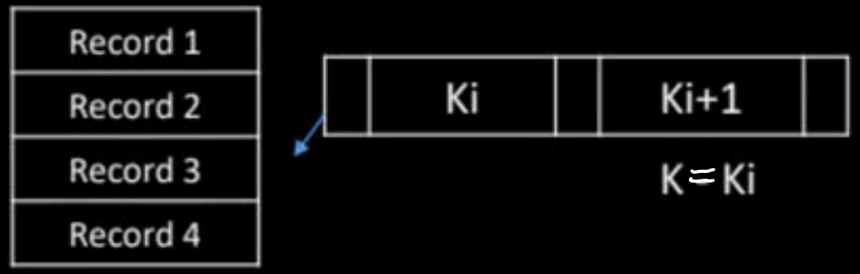
Searching B+ trees Algorithm for a key-value K

1.) Start from the root, look for the largest key value (K_i)
in the node <=k

2.) Follow the pointer P_{i+1} to next value, until reach the leaf node.



3.)If k is found to be equal to Ki in the leaf, follow Pr_i to search record.

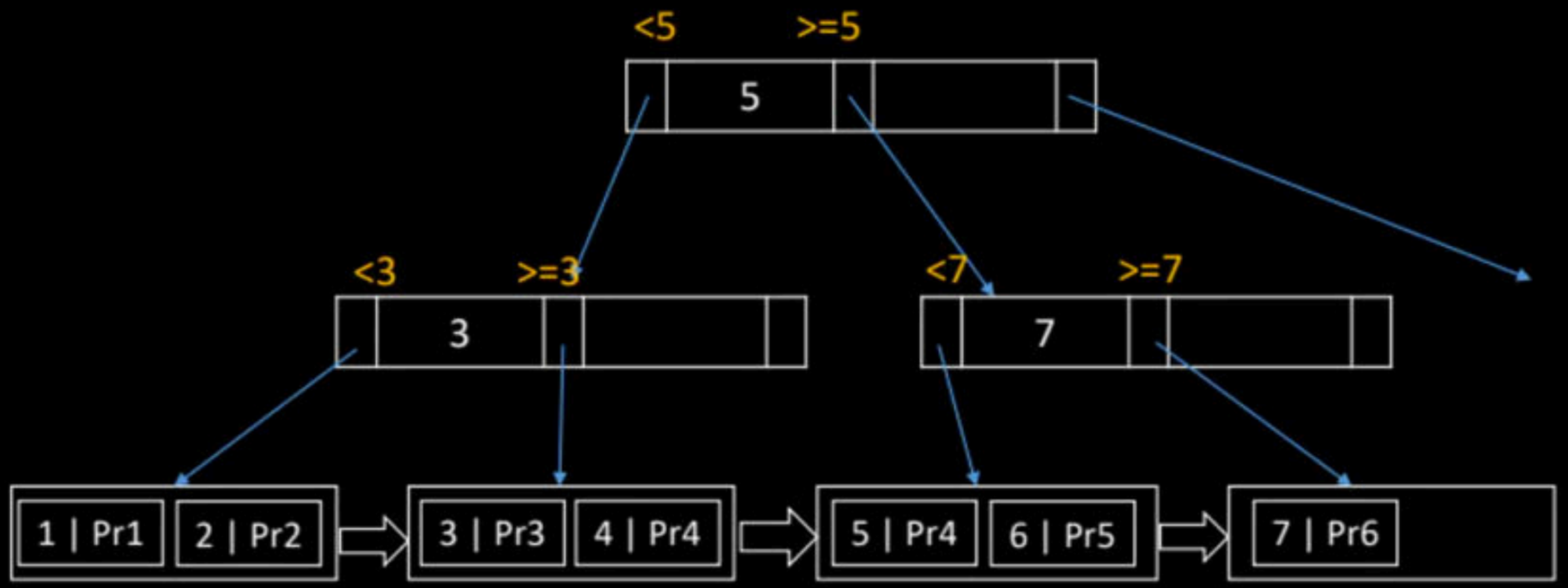


$$T.C. = \log_p n$$

Searching - B+ trees

EXAMPLE

B + tree with p = 3 and p_{leaf} = 2



B+ Trees are considered BALANCED because

(A) the lengths of the paths from the root to all leaf nodes are all equal.
(B) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1.
(C) the number of children of any two non-leaf sibling nodes differ by at most 1.
(D) the number of records in any two leaf nodes differ by at most 1.

[GATE 2016]

B+ Trees are considered BALANCED because

(A) the lengths of the paths from the root to all leaf nodes are all equal.
(B) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1.
(C) the number of children of any two non-leaf sibling nodes differ by at most 1.
(D) the number of records in any two leaf nodes differ by at most 1.

Answer: (A)

Explanation: In both B Tree and B+ trees, depth (length of root to leaf paths) of all leaf nodes is same. This is made sure by the insertion and deletion operations.
In these trees, we do insertions in a way that if we have to increase height of tree after insertion, we increase height from root. This is different from BST where height increases from leaf nodes.
Similarly, if we have to decrease height after deletion, we move the root one level down. This is also different from BST which shrinks from bottom.

[GATE 2016]

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

(A) 1

(B) 2

(C) 3

(D) 4

[GATE 2010]

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

(A) 1

(B) 2

(C) 3

(D) 4

Solution: Assuming order of B+ tree as p, maximum number of keys will be (p – 1).

As it is given that,

p – 1 = 5 => p = 6

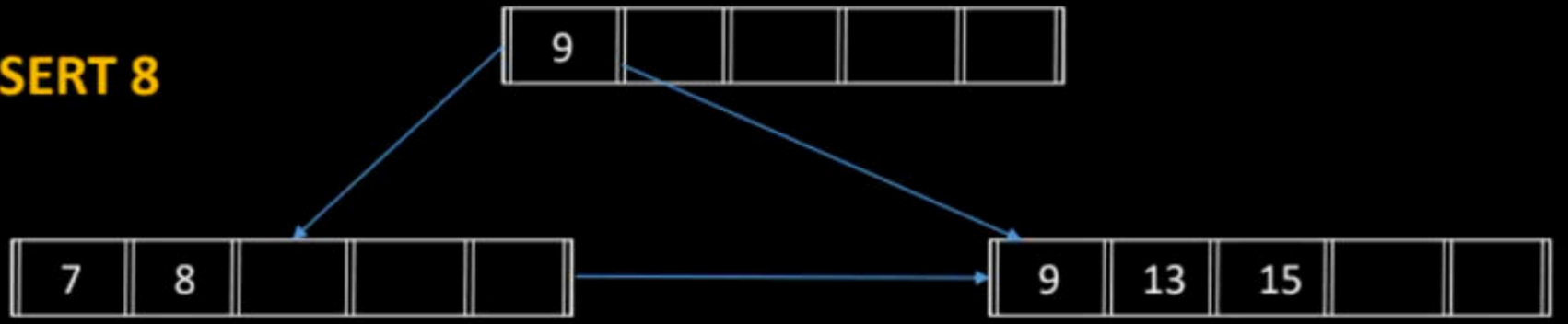
Therefore, minimum number of keys:

ceil(p/2) – 1 = 2

[GATE 2010]

ORDER 5 B+ TREE

INSERT 8



Overflow : When number of search key values exceed “p-1”

Leaf Node :

Split into two nodes

1.) 1st node contains ⌈ p-1/2 ⌉ values

2.) 2nd node contains remaining values

3.) Copy the smallest search key value of the 2nd node to the parent node

Non - Leaf Node :

Split into two nodes

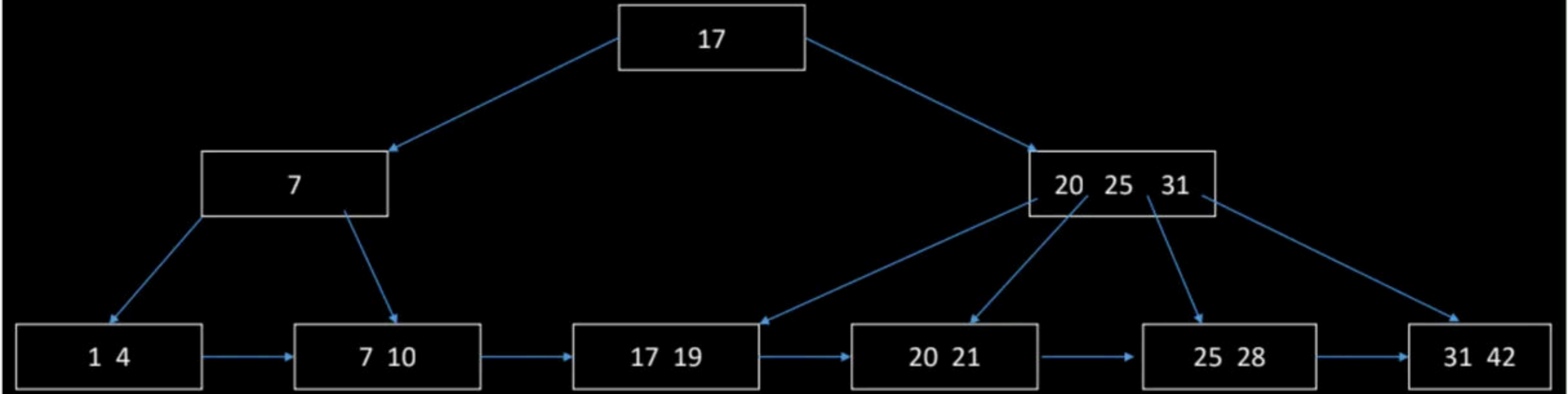
1.)1st nodes contains ⌈ n/2 ⌉ -1 keys

2.)Move the smallest of the remaining keys to the parent

3.)2nd node contains remaining keys

Construct B+ tree for 1,4,7,10,17,21,31,25,19,20,28,42 with P=4 (Order is 4)

Construct B+ tree for 1,4,7,10,17,21,31,25,19,20,28,42 with P=4 (Order is 4)



Why B/B+ trees preferred over Binary Search Trees like (AVL, Red black..etc) for storing index records and accessing them ?

- In Binary trees, every node can have only two children therefore they tend to grow in height wise
- Because of it even database is small, the number of levels we might have to access before we get to the final level is actually large.

Deleting data entries from B+ trees

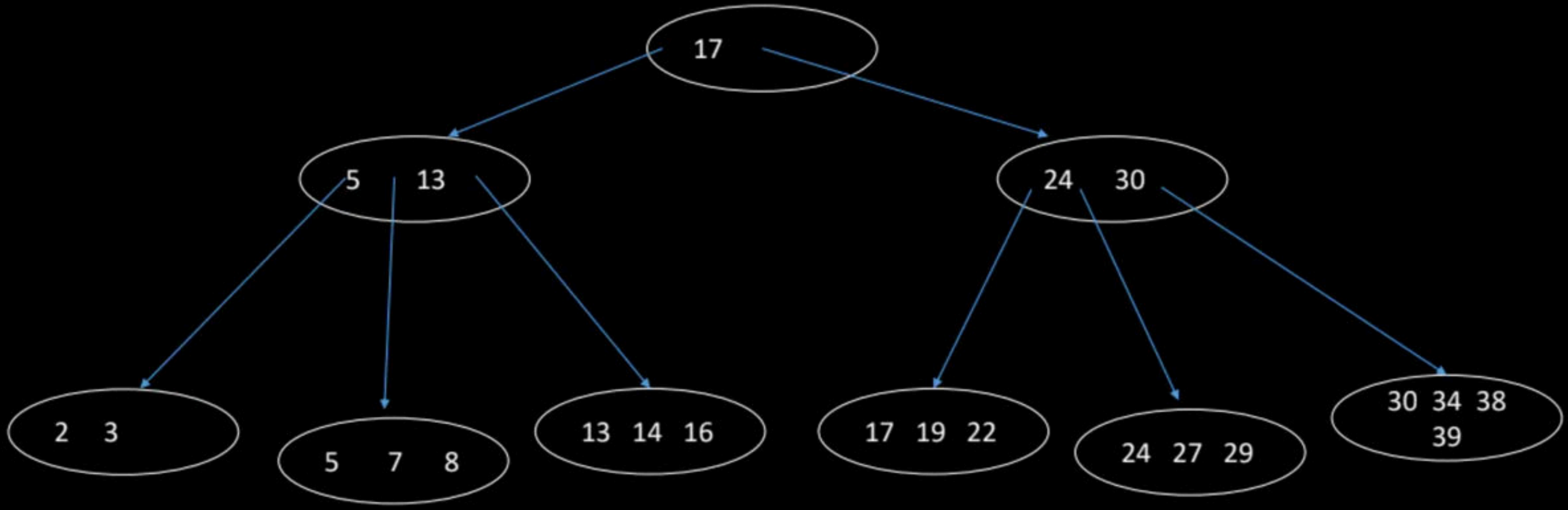
- Start at root, find leaf L where entry belongs
- Remove the entry
  - If L contains at least ⌈ p/2 ⌉ – 1entries, done !
  - If L has only ⌈ p/2 ⌉ – 2 entries.
    - Try to redistribute, borrowing from sibling (adjacent node with same parent as L)
    - If redistributing fails, merge L and a sibling

- If merge has occurred, then corresponding entry from parent must be deleted.

- Merge could propagate to root, decreasing height.

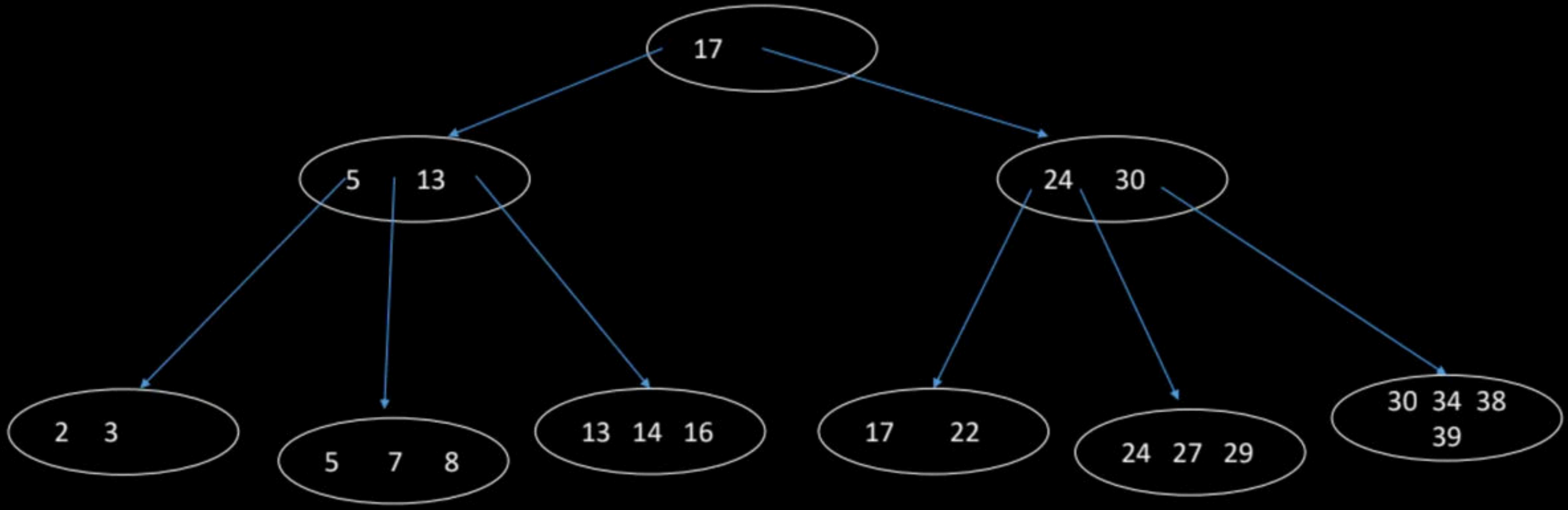
- If the deleted entry is present in internal node, replace it with inorder successor.

Example 1: Consider B+ tree of order 5



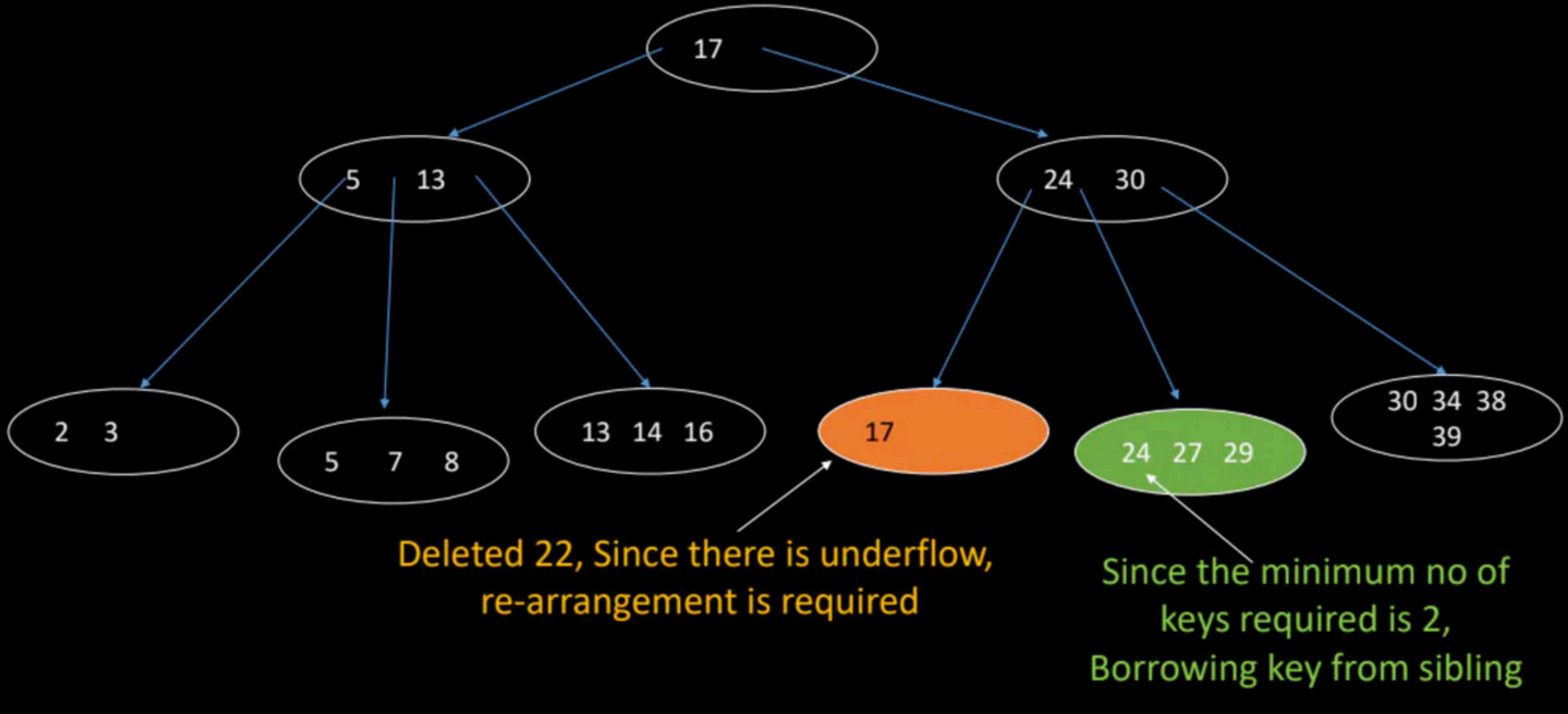
Delete 19,22

Example 1: Consider B+ tree of order 5

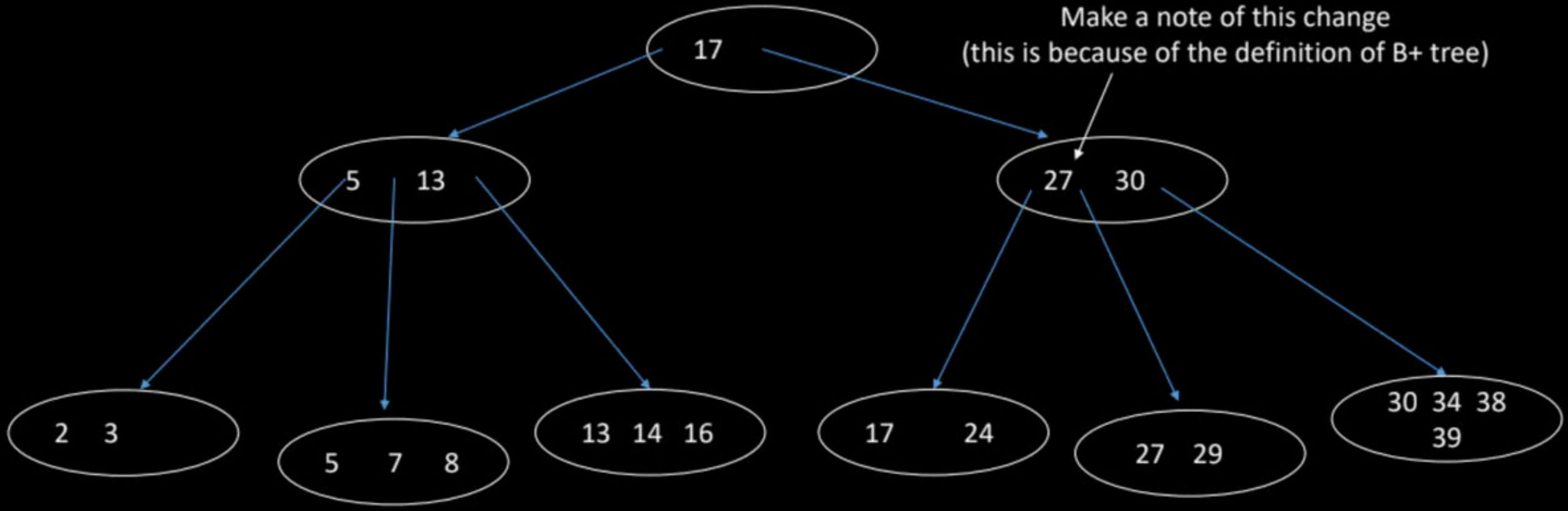


Deleted 19, No underflow, no re-arrangement is required

Example 1: Consider B+ tree of order 5

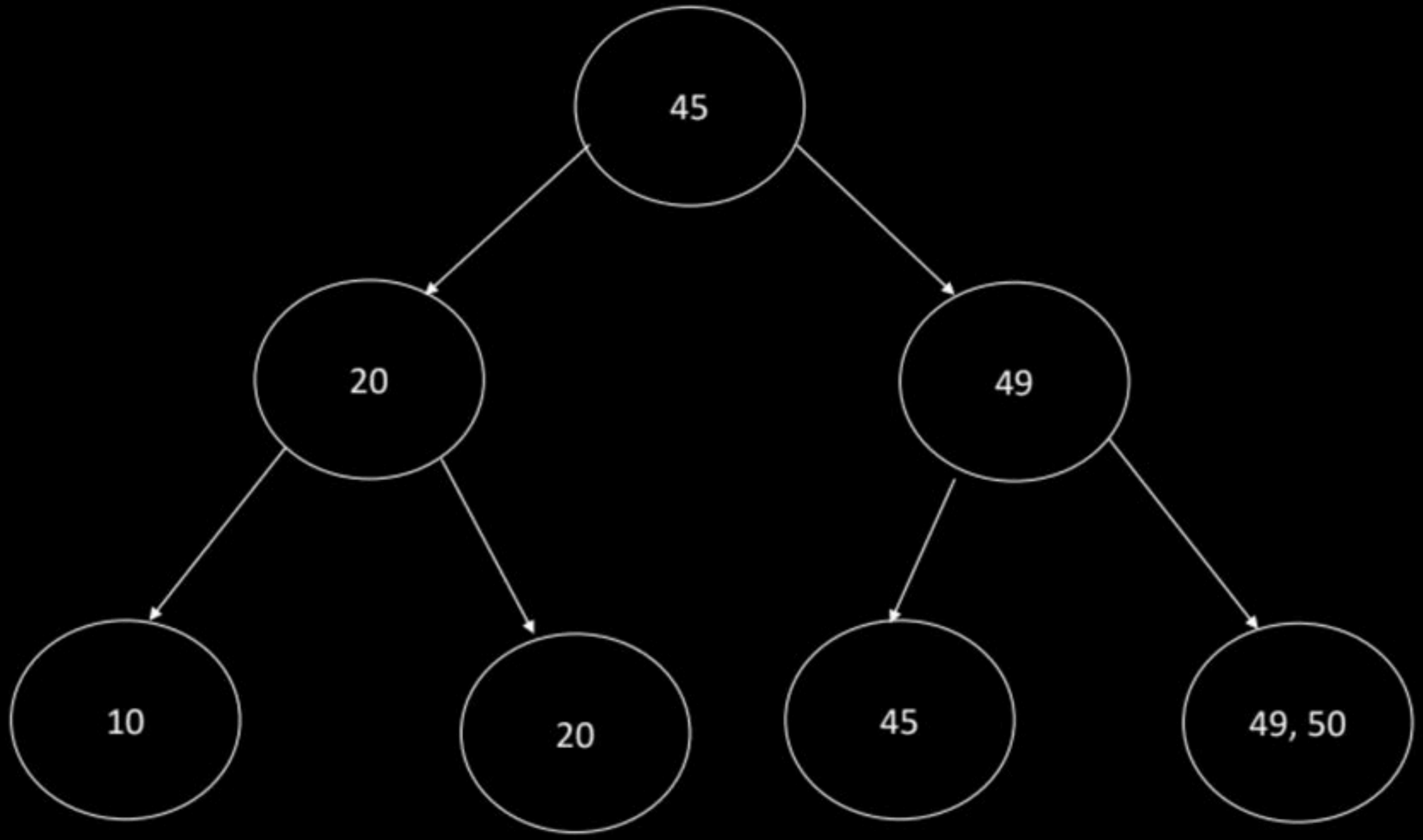


Example 1 : Consider B+ tree of order 5



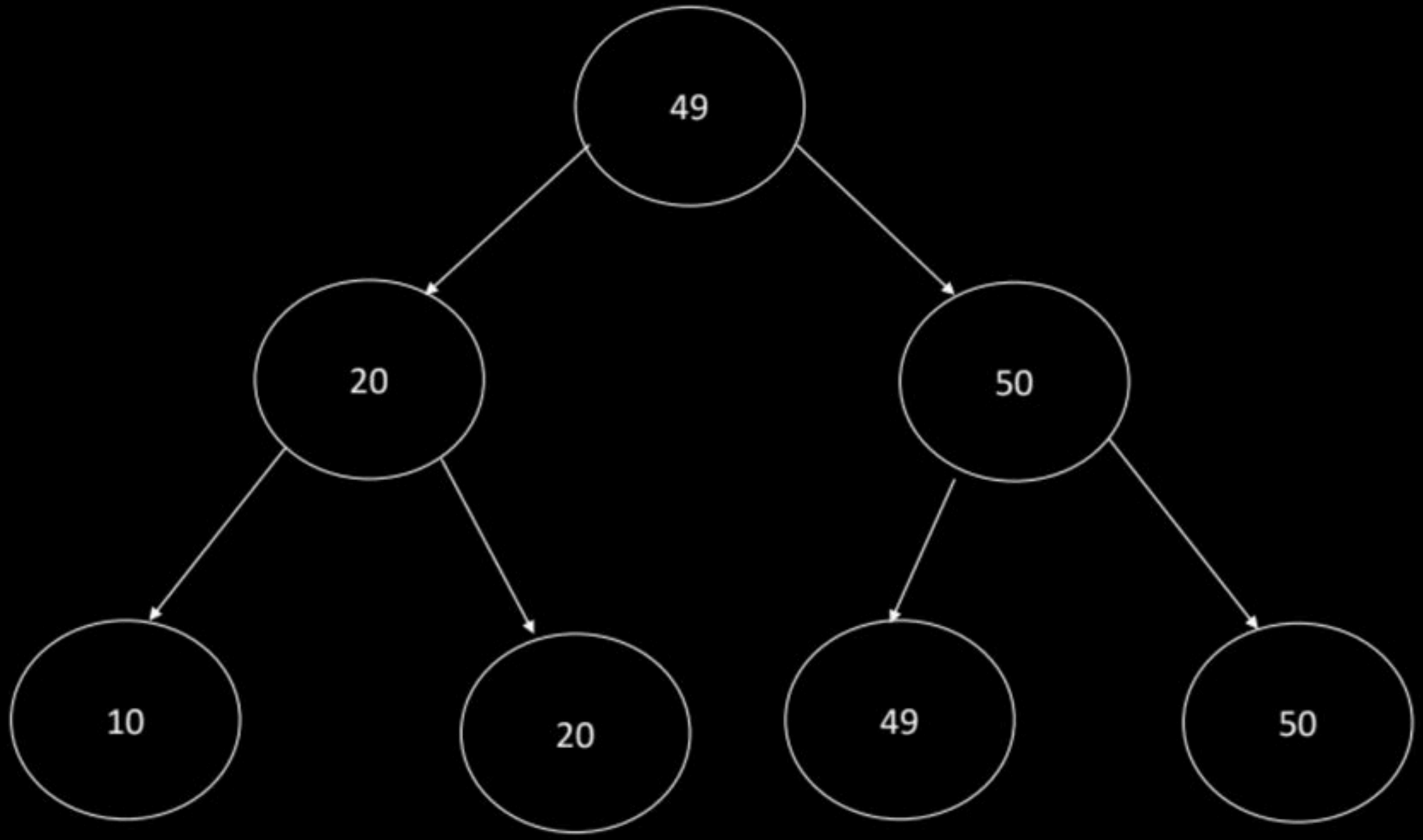
Tree after re-arrangement

Example 2 : Consider B+ tree of order 3



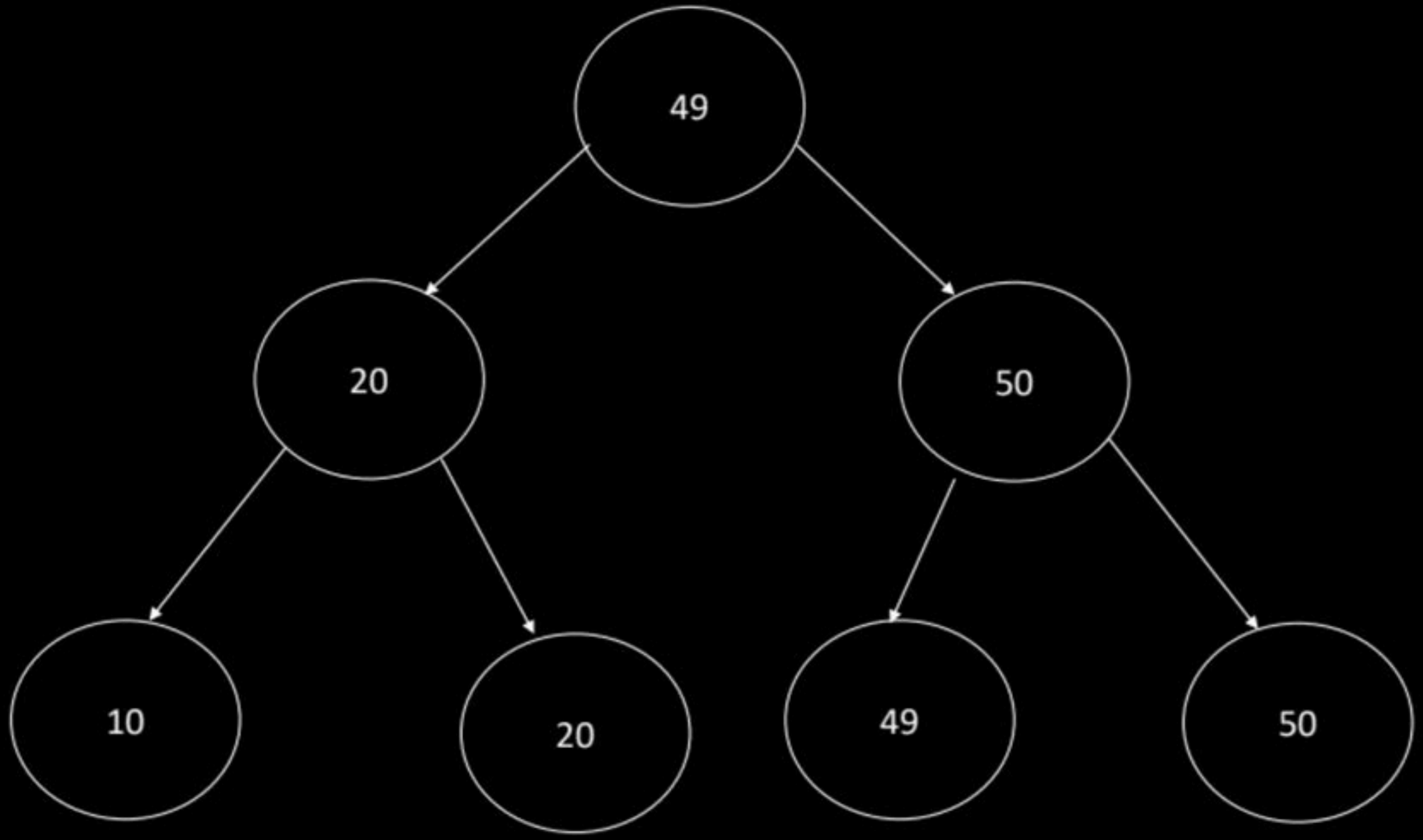
Delete 45

Example 2 : Consider B+ tree of order 3



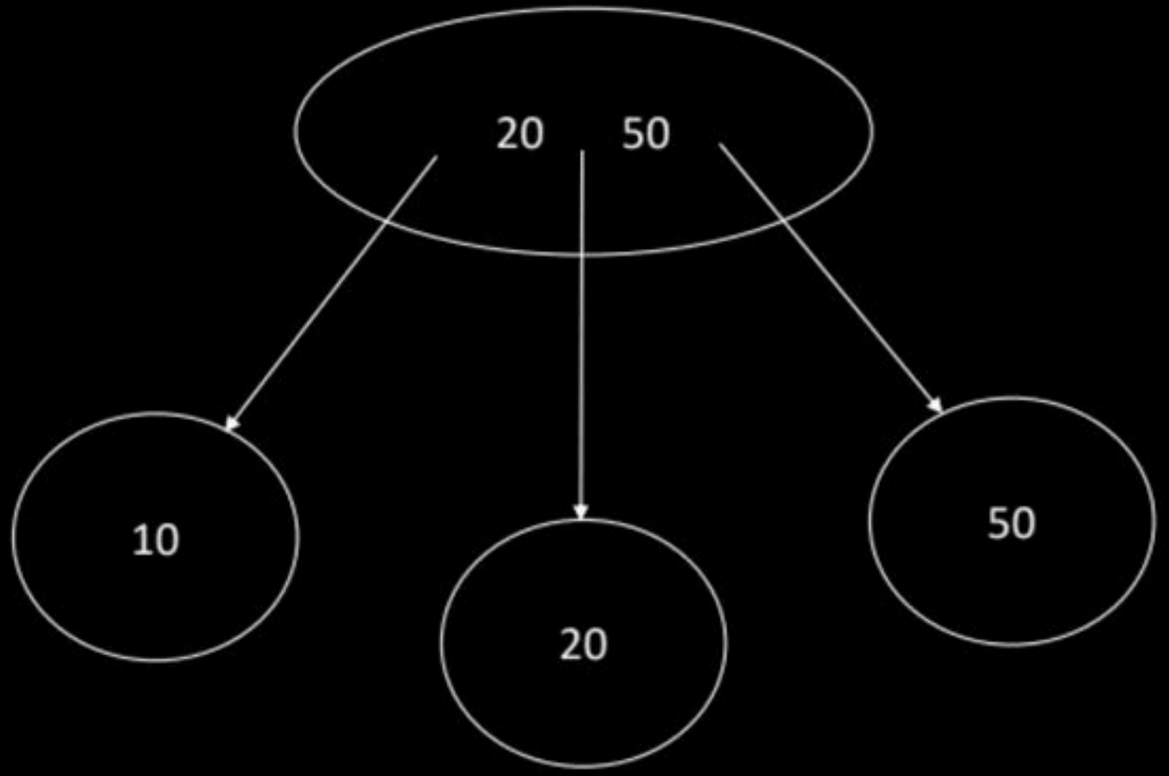
Tree after deletion of 45

Example 3 : Consider B+ tree of order 3



Delete 49

Example 3 : Consider B+ tree of order 3



Tree after deletion of 49