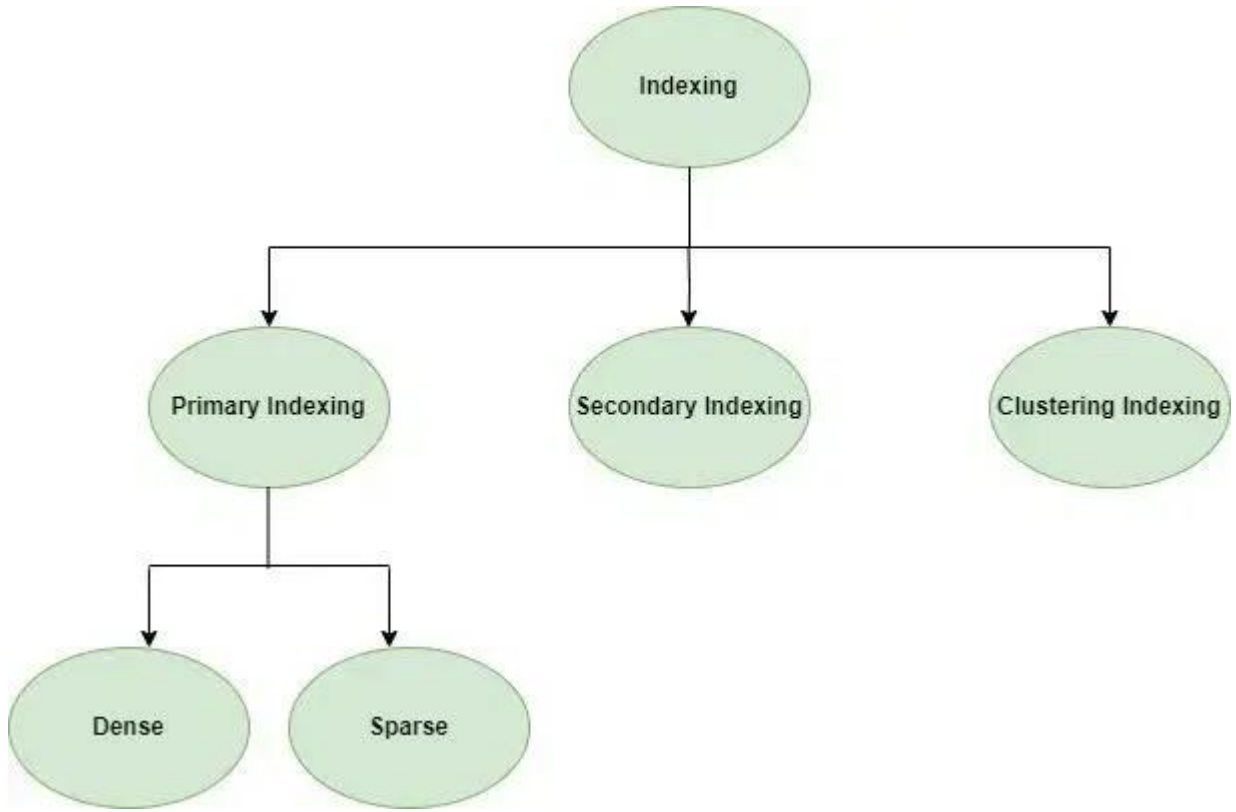
Indexing

- **Indexing** is a crucial technique used in databases to optimize **data retrieval operations**. It improves query performance by minimizing **disk I/O operations**, thus reducing the time it takes to locate and access data.
- Essentially, indexing allows the **database management system** (DBMS) to locate data more efficiently without having to scan the entire dataset.
- **Indexes** are organized data structures that allow quick searching based on key values. When an index is created for a database table, it maintains a sorted order of key values along with pointers to the actual data rows.

Attributes of Indexing

- **Access Types:** This refers to the type of access such as value-based search, range access, etc.
- **Access Time:** It refers to the time needed to find a particular data element or set of elements.
- **Insertion Time:** It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time:** Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead:** It refers to the additional space required by the index.


Indexing
Primary Indexing
Secondary Indexing
Clustering Indexing
Dense
Sparse

File Organization in Indexing

- **Sequential File Organization (Ordered Index File)**
- In this type of organization, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organizations might store the data in a dense or sparse format.
- **Dense Index**: Every search key value in the data file corresponds to an index record. This method ensures that each key value has a reference to its data location.

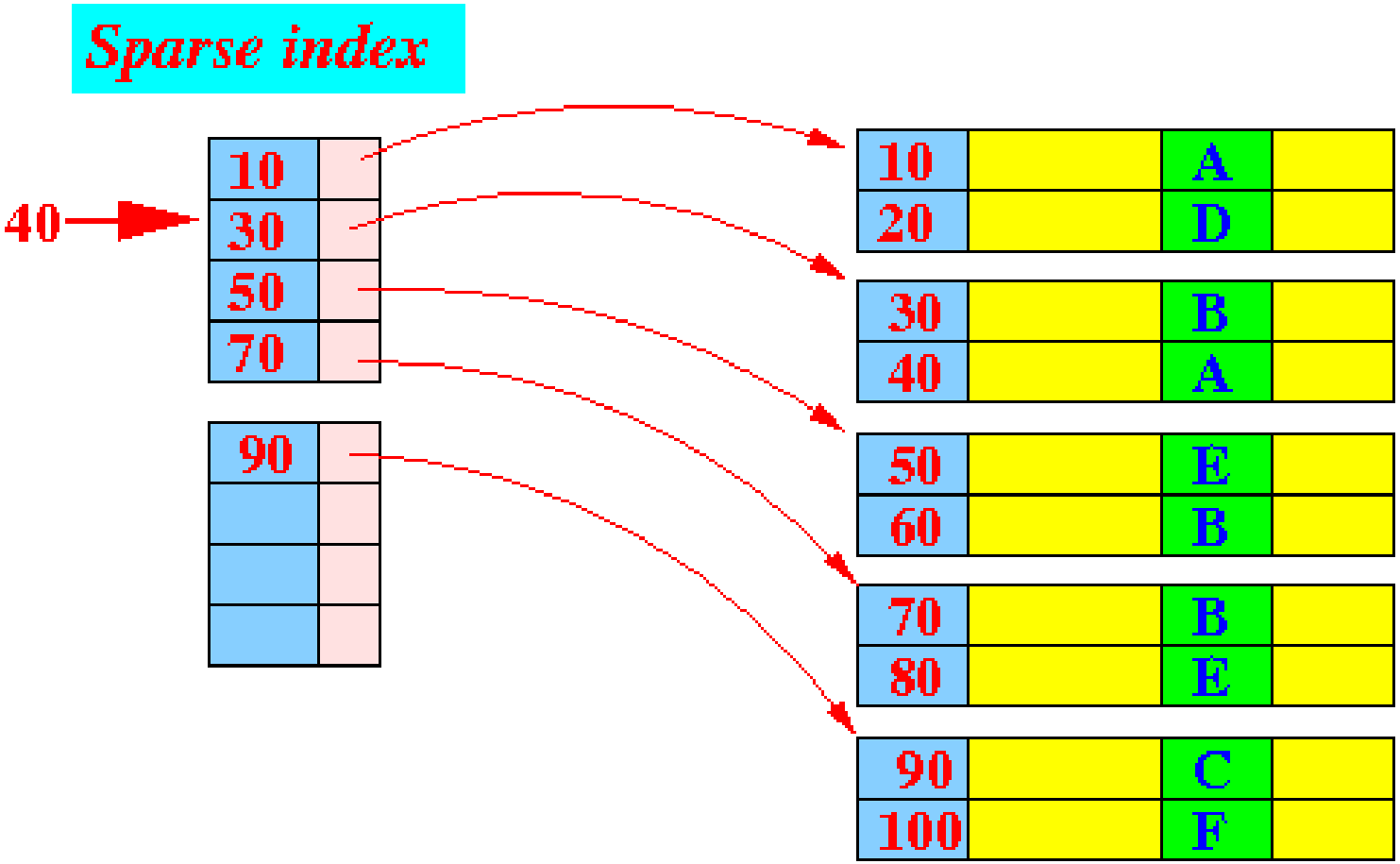
  Example: If a table contains multiple entries for the same key, a dense index ensures that each key value has its own index record.

Dense Index


Brighton
Downtown
Mianus
Perryridge
Redwood
Round Hill
Brighton A-217 750
Downtown A-101 500
Downtown A-110 600
Mianus A-215 700
Perryridge A-102 400
Perryridge A-201 900
Perryridge A-218 700
Redwood A-222 700
Round Hill A-305 350

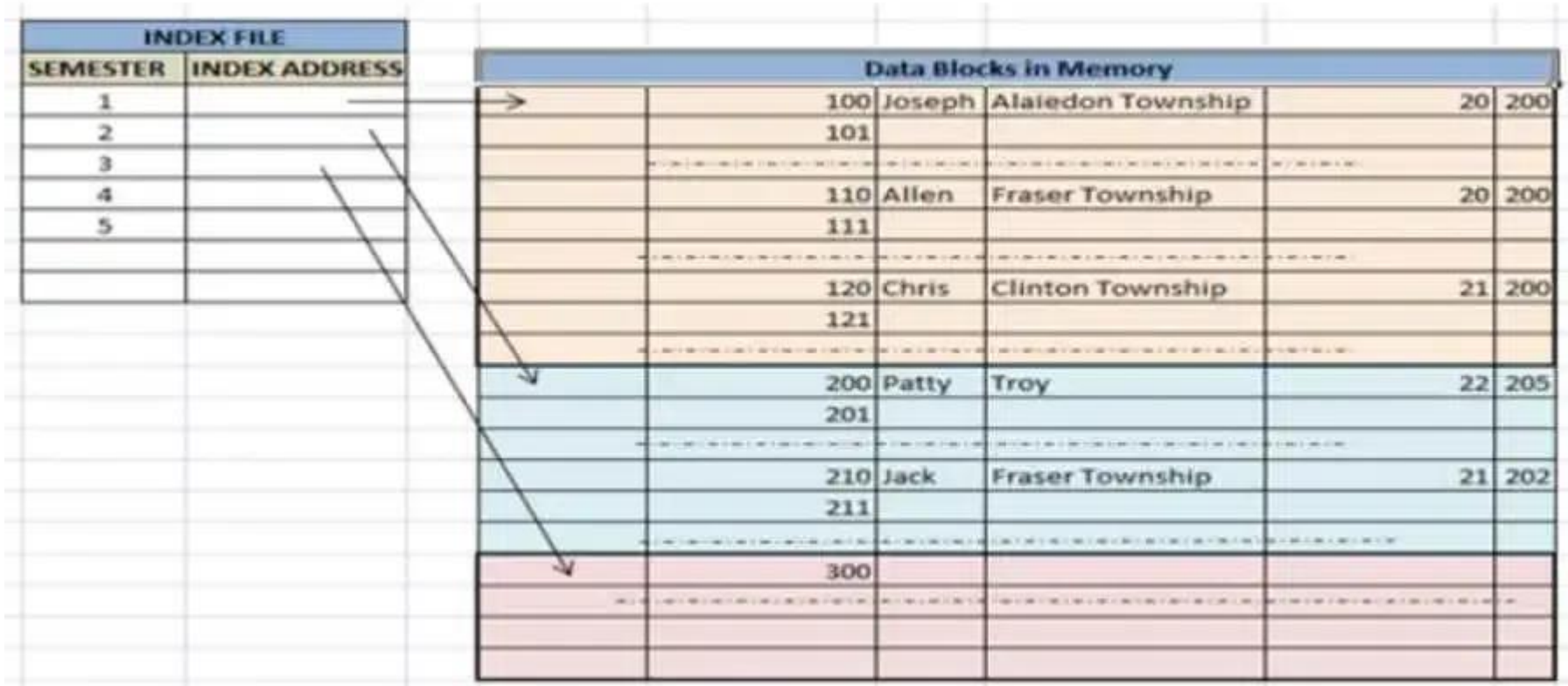
- **Sparse Index:** The index record appears only for a few items in the data file. Each item points to a block as shown. To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.

  Access Method: To locate a record, we find the index record with the largest key value less than or equal to the search key, and then follow the pointers sequentially.


Sparse index
40
10
30
50
70
90
10
A
20
D
30
B
40
A
50
E
60
B
70
B
80
E
90
C
100
F

Hash File Organization

- **1. Clustered Indexing**
- Clustered indexing is a technique where **multiple related records** are **stored together** in the **same file**. This helps reduce the cost of searching because related data is kept close to each other. Clustered indexing is especially useful when **multiple tables** or **records** need to be frequently joined. Storing related records together makes this process faster and more efficient.

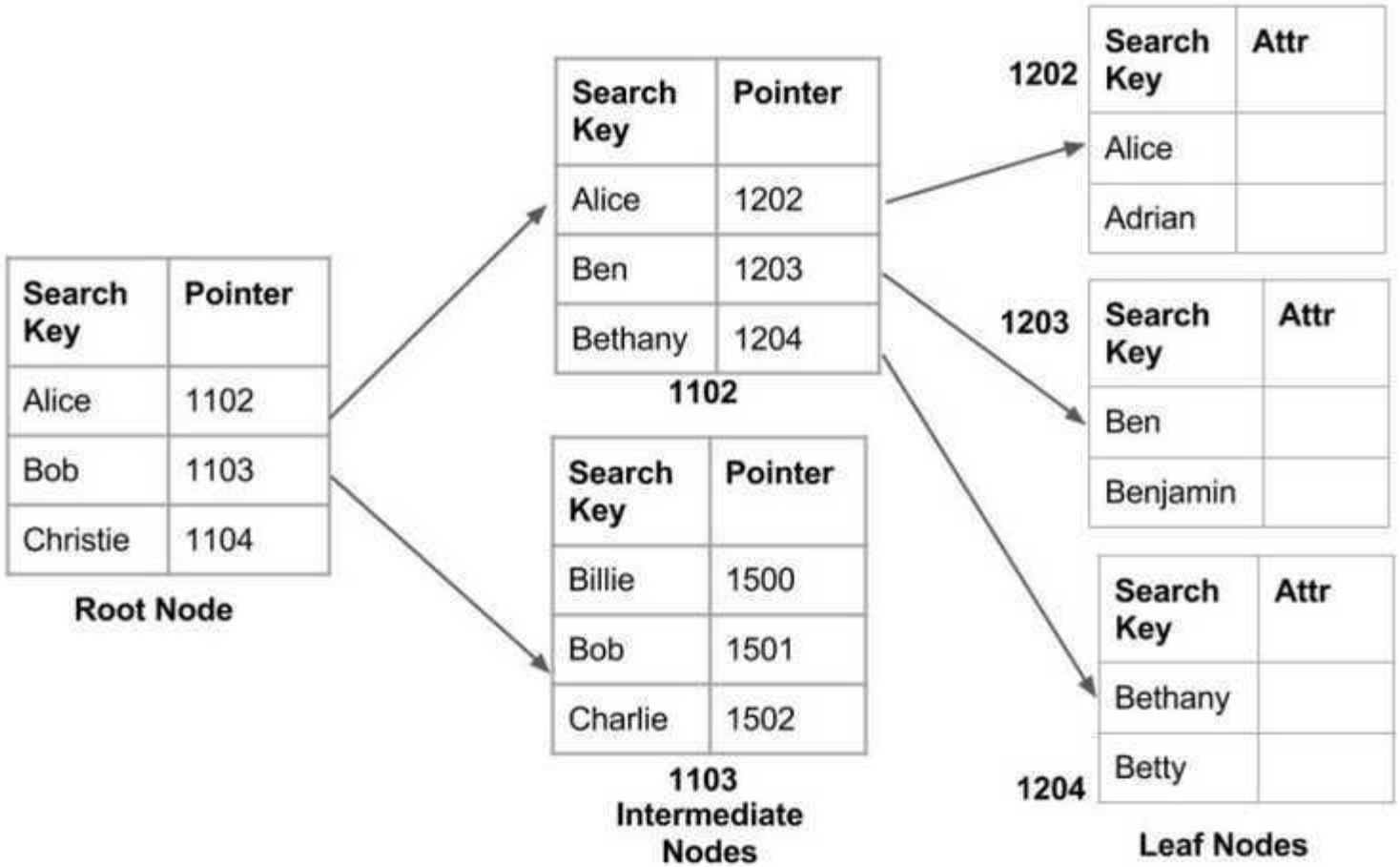

INDEX FILE
SEMESTER
INDEX ADDRESS
1
2
3
4
5
Data Blocks in Memory
100 Joseph Alaiedon Township 20 200
101
110 Allen Fraser Township 20 200
111
120 Chris Clinton Township 21 200
121
200 Patty Troy 22 205
201
210 Jack Fraser Township 21 202
211
300

2. Primary Indexing

- This is a type of Clustered Indexing wherein the data is sorted according to the search key and the **primary key** of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are **unique** and are stored in a sorted manner, the performance of the **searching operation** is quite efficient.
- **Key Features**: The data is stored in sequential order, making searches faster and more efficient.

3. Non-clustered or Secondary Indexing

- A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes.



Non clustered index

B-Tree

- A B-Tree is a specialized m-way tree designed to optimize data access, especially on disk-based storage systems.
- In a B-Tree of order **m**, each node can have up to m children and m-1 keys, allowing it to efficiently manage large datasets.
- The value of m is decided based on disk block and key sizes.

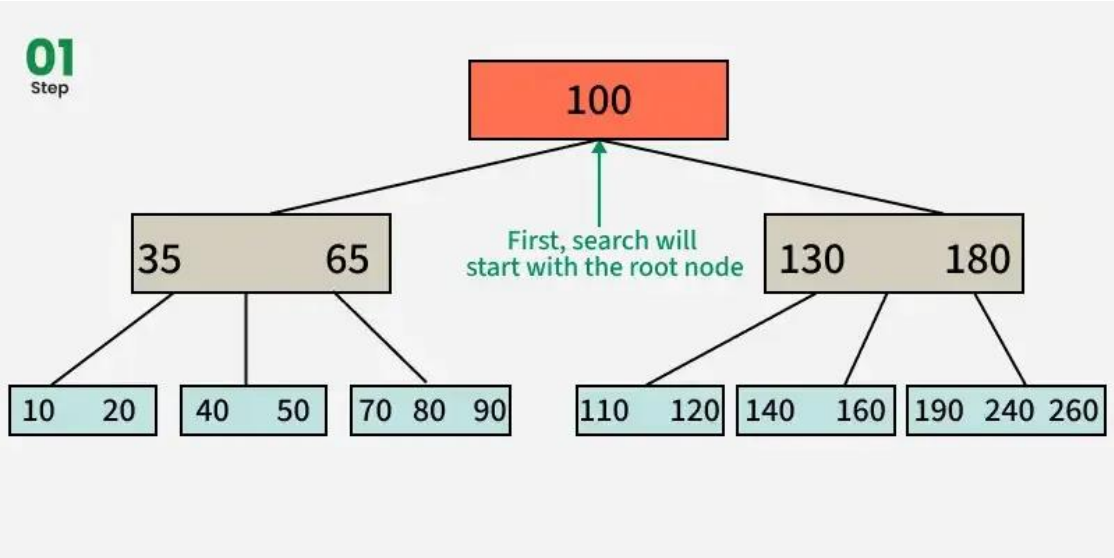
Properties of a B-Tree

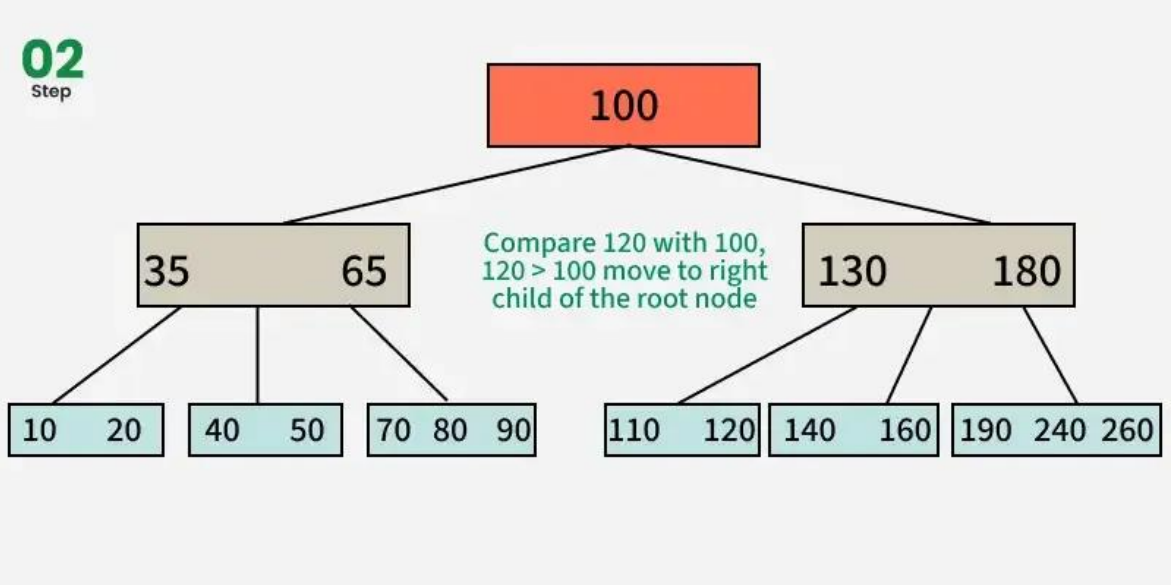
- A B Tree of order m can be defined as an m-way search tree which satisfies the following properties:
- All leaf nodes of a B tree are at the same level, i.e. they have the same depth (height of the tree).
- The keys of each node of a B tree (in case of multiple keys), should be stored in the ascending order.
- In a B tree, all non-leaf nodes (except root node) should have at least m/2 children.
- All nodes (except root node) should have at least m/2 - 1 keys.
- If the root node is a leaf node (only node in the tree), then it will have no children and will have at least one key. If the root node is a non-leaf node, then it will have at least 2 children and at least one key.
- A non-leaf node with n-1 key values should have n non NULL children.

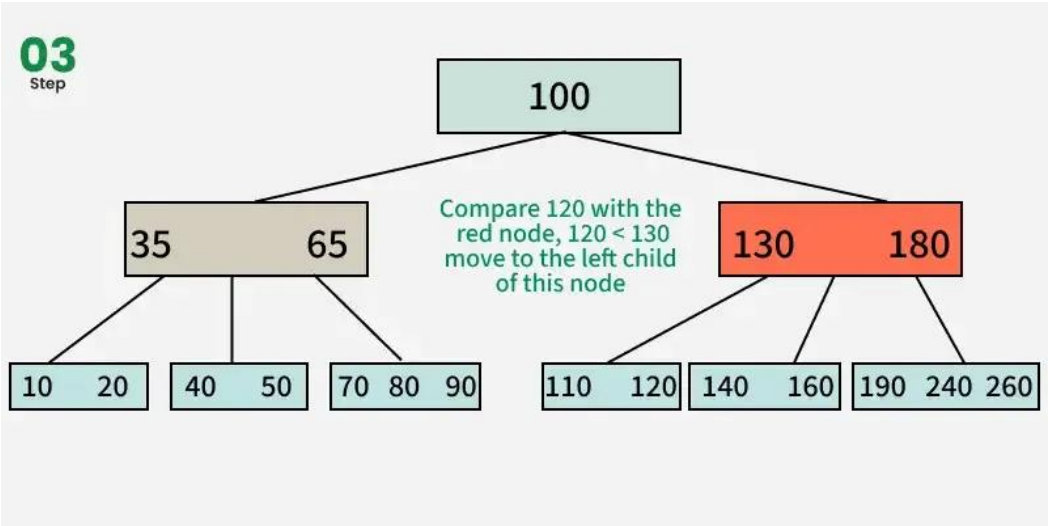
Search Operation in B-Tree

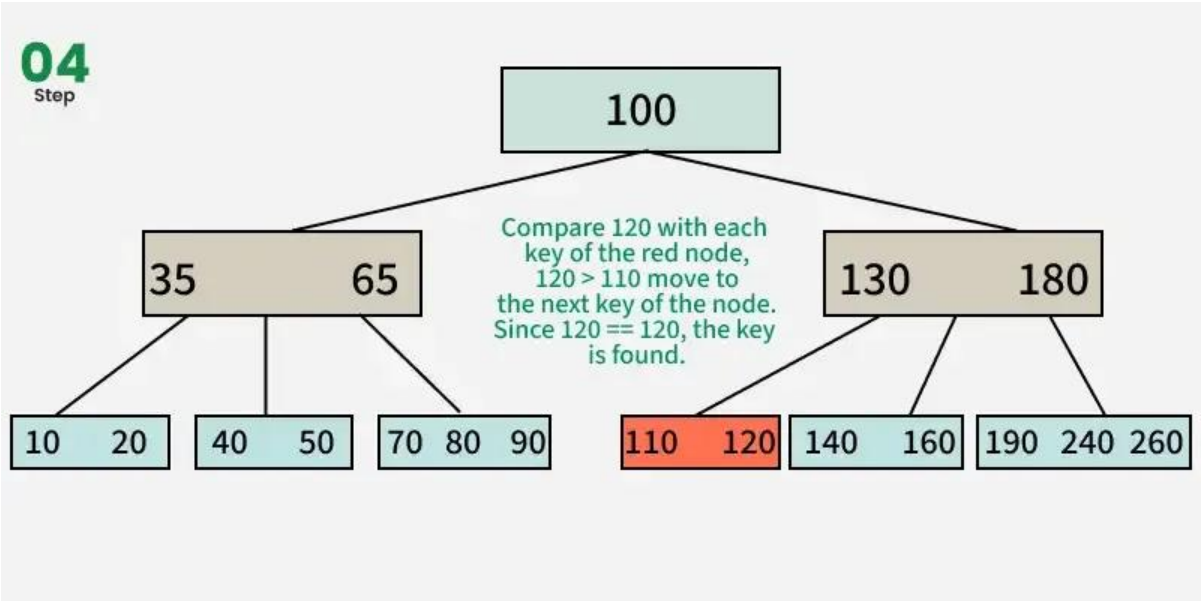
- Search is similar to the search in Binary Search Tree. Let the key to be searched is k.
- Start from the root and recursively traverse down.
- For every visited non-leaf node
  - If the current node contains k, return the node.
  - Otherwise, determine the appropriate child to traverse. This is the child just before the first key greater than k.
- If we reach a leaf node and don't find k in the leaf node, then return NULL.

Input: *Search 120 in the given B-Tree.*









- A new key is always inserted into a leaf node. To insert a key k, we start from the root and traverse down the tree until we reach the appropriate leaf node. Once there, the key is added to the leaf.
- Unlike Binary Search Trees (BSTs), nodes in a B-Tree have a predefined range for the number of keys they can hold. Therefore, before inserting a key, we ensure the node has enough space. If the node is full, an operation called splitChild() is performed to create space by splitting the node.

Insert 1 to 10 in a B-Tree of order 2 (t = 2)

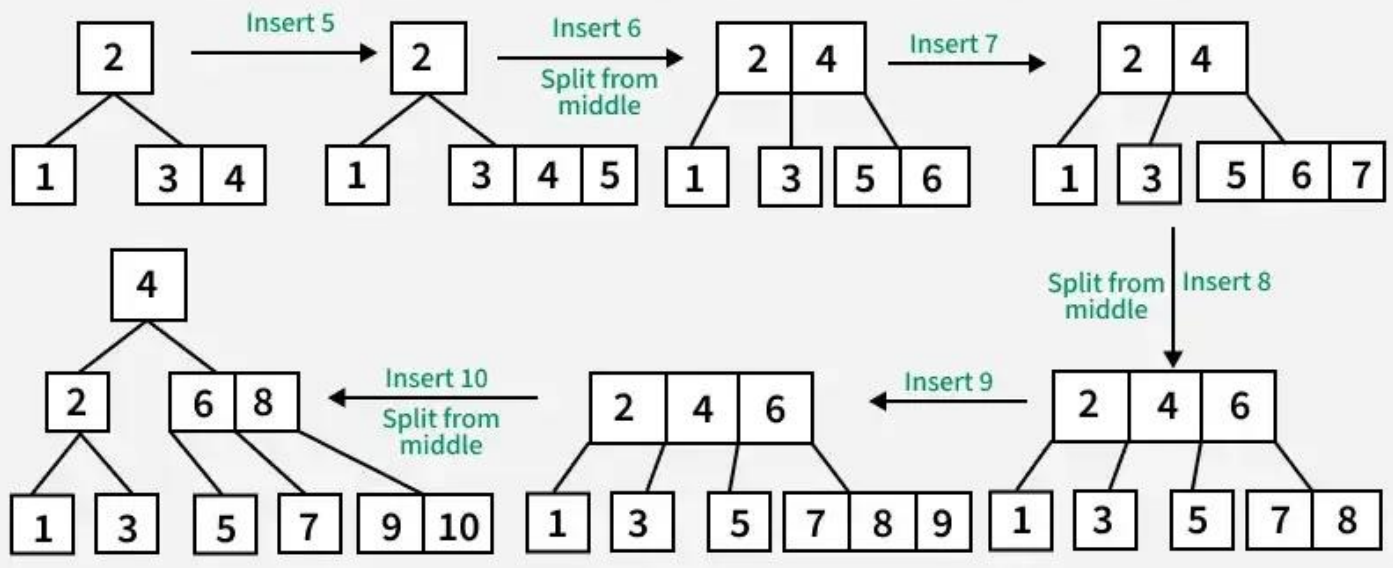
Minimum number of Keys = t - 1 i.e. 1

Maximum number of Keys = 2t - 1 i.e. 3

Maximum number of Children = 2t i.e. 4

Keys: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Insert 1 to 10 in a B-Tree of order 2 (t = 2)



B+ Tree

- **B + Tree** is a variation of the B-tree data structure. In a B + tree, data pointers are stored only at the leaf nodes of the tree. In this **t**ree, structure of a leaf node differs from the structure of internal nodes.

Properties of B+ trees

- Every node in a B+ Tree, except root, will hold a maximum of m children and (m-1) keys, and a minimum of [m2] children and [m−12] keys, since the order of the tree is m.

- The root node must have no less than two children and at least one search key.

- All the paths in a B tree must end at the same level, i.e. the leaf nodes must be at the same level.

- A B+ tree always maintains sorted data.

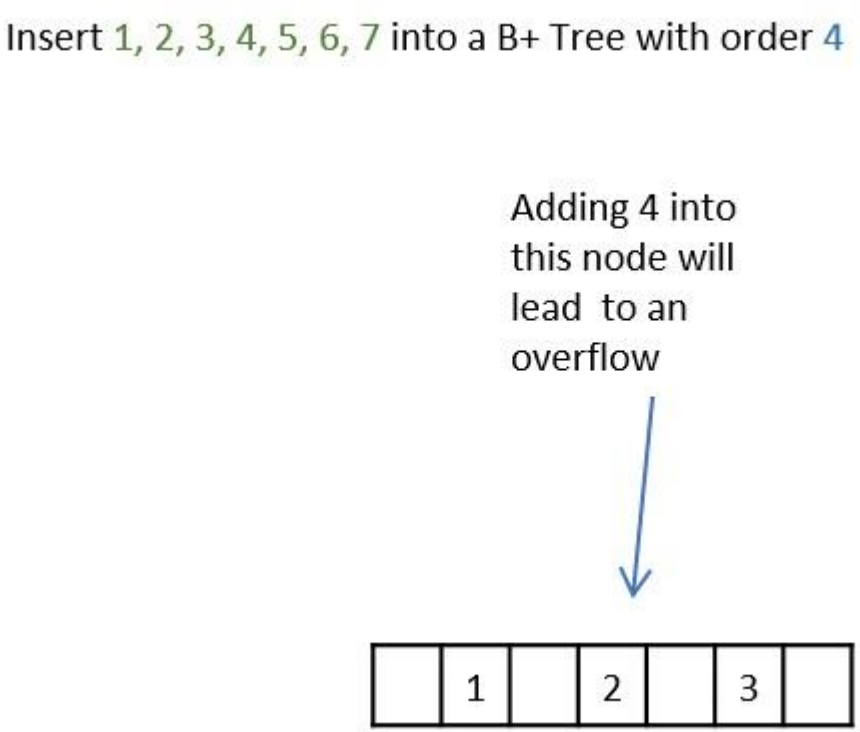
Insertion

- Step 1 − Calculate the maximum and minimum number of keys to be added onto the B+ tree node.

Insert 1, 2, 3, 4, 5, 6, 7 into a B+ Tree with order 4

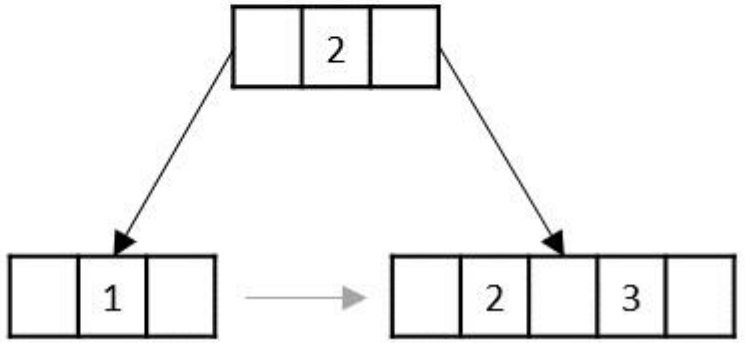
Order = 4
Maximum Children (m) = 4
Minimum Children ($\left\lceil \frac{m}{2} \right\rceil$) = 2
Maximum Keys (m - 1) = 3
Minimum Keys ($\left\lceil \frac{m-1}{2} \right\rceil$) = 1

- Step 2 − Insert the elements one by one accordingly into a leaf node until it exceeds the maximum key number.

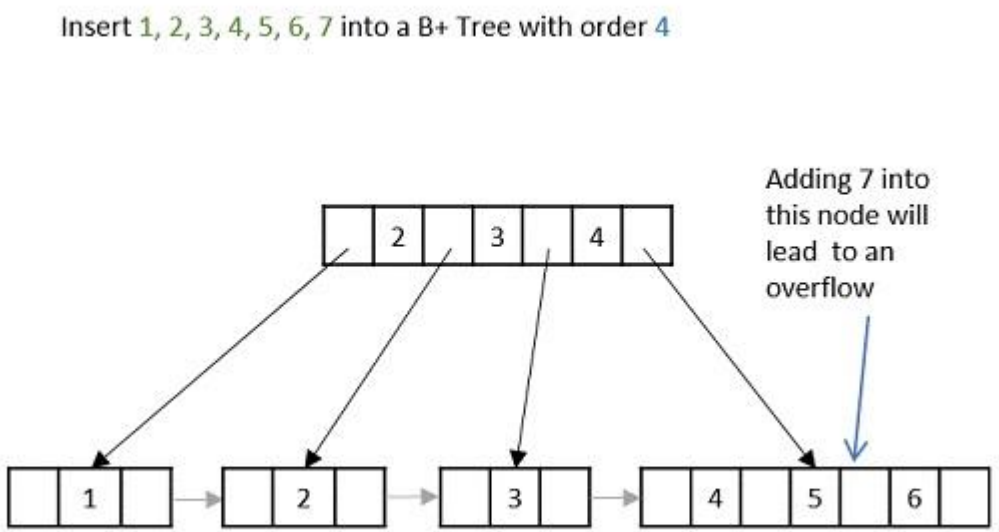


- Step 3 − The node is split into half where the left child consists of minimum number of keys and the remaining keys are stored in the right child.

Insert 1, 2, 3, 4, 5, 6, 7 into a B+ Tree with order 4



- Step 4 − But if the internal node also exceeds the maximum key property, the node is split in half where the left child consists of the minimum keys and remaining keys are stored in the right child. However, the smallest number in the right child is made the parent.



- Step 5 − If both the leaf node and internal node have the maximum keys, both of them are split in the similar manner and the smallest key in the right child is added to the parent node.