LET'S START WITH DBMS :)

B and B+ trees

Clustered and non-clustered indexes are concepts that describe how data is stored and accessed in a database, but B-trees (and B+ trees) are the data structures that actually implement these indexes. Knowing B-trees gives you insight into how these indexes work under the hood.

Understanding B-trees helps you understand why certain queries perform well or poorly based on the structure of the index. For example, how a B-tree's balanced nature affects search times or why range queries are efficient with B-tree indexes

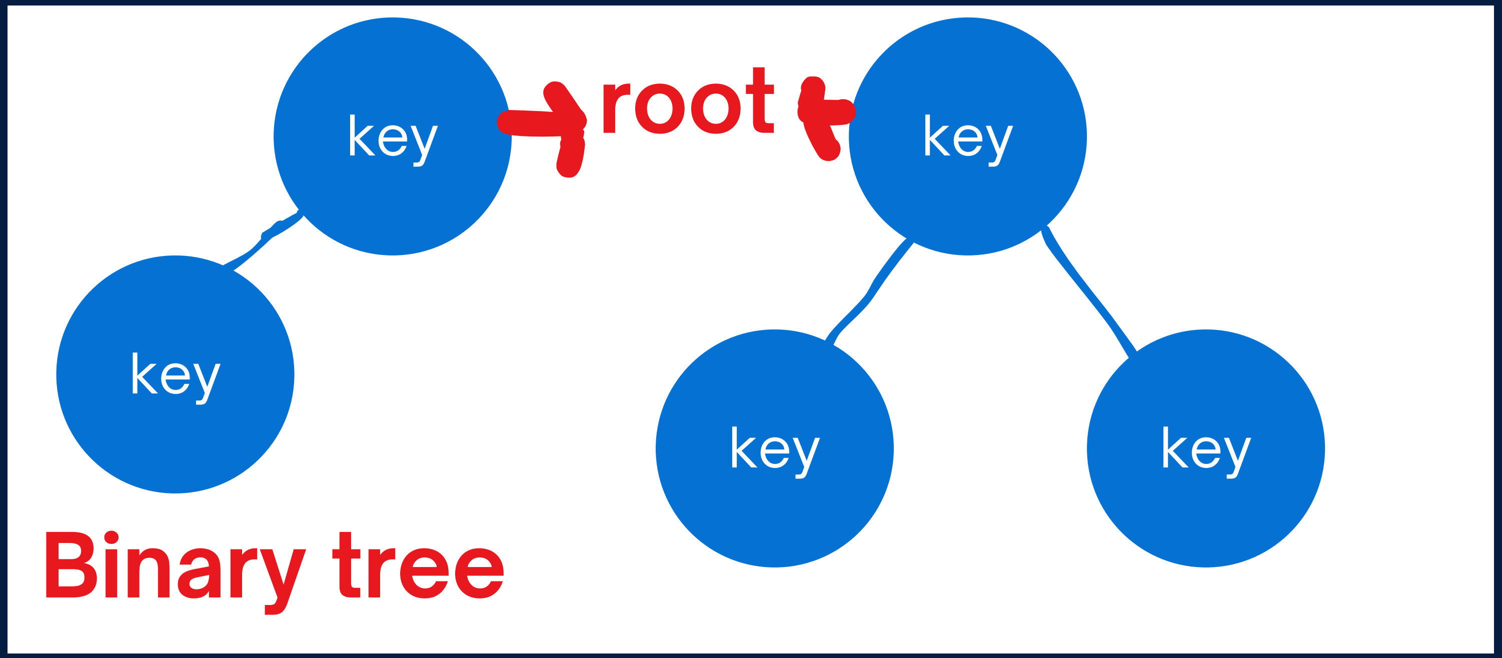
B-trees provide the balanced, efficient structure that makes these types of indexes performant, ensuring that operations like search, insert, delete, and update are done in logarithmic time (O(log n)). This makes B-tree indexing crucial for optimizing database queries, whether in clustered or non-clustered index scenarios.

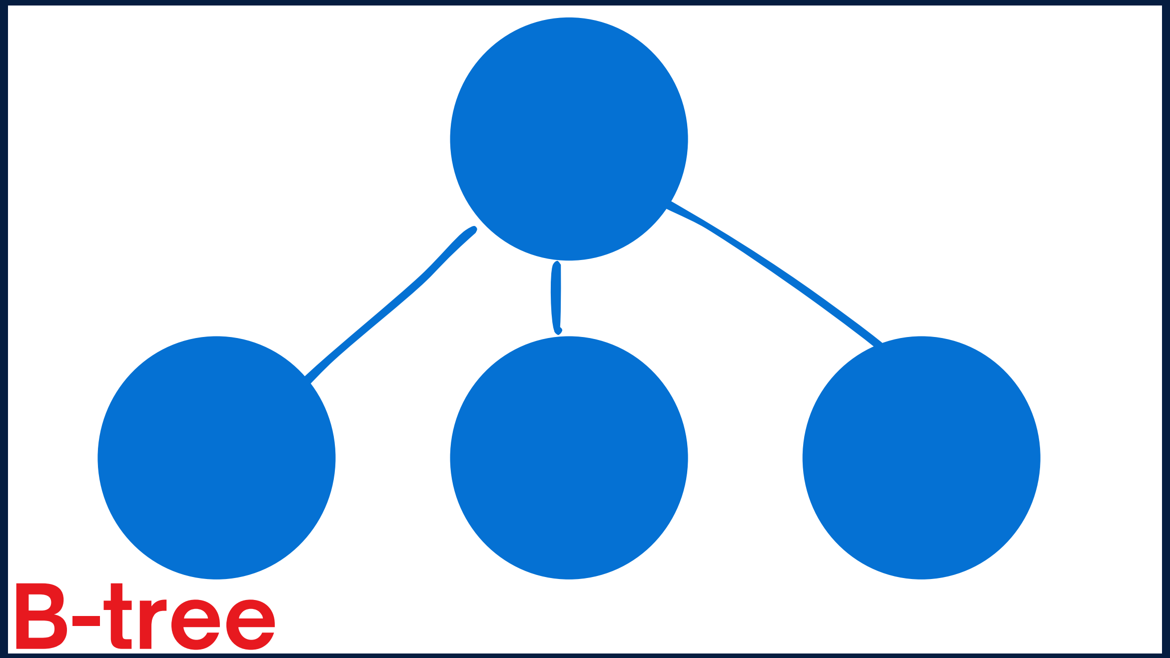
LET'S START WITH DBMS :)

B trees(M-way tree)

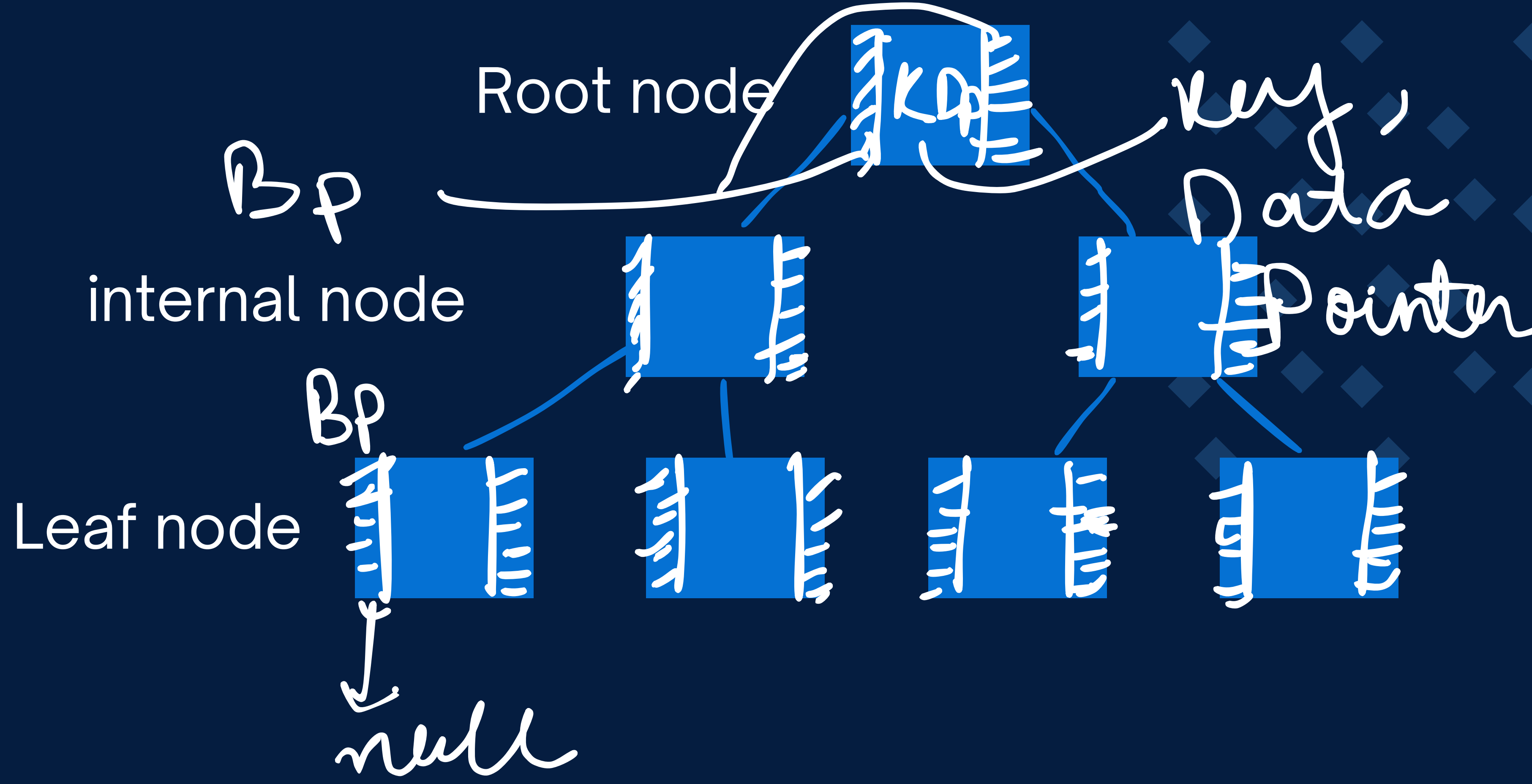
B-trees are self-balancing tree data structures that maintain sorted data and allow searches, sequential access, insertions, and deletions in logarithmic time. All leaf nodes are at the same level.

In B trees you can have minimum 2 children and max x children.
Now B-tree is a generalisation of Binary Search trees. In BST every node can have atmost 2 children(0,1,2) and only one key





LET'S START WITH DBMS :)



B trees

(Rp) Dp-> record/data pointer(where the record is present in secondary memory(disk)

(Bp) Tp-> block/tree pointer(links to the children nodes)

Structure of B-Tree:

- Nodes: A B-tree is composed of nodes, each containing keys and pointers (references) to child nodes. The keys within a node are sorted in ascending order.
- Root, Internal Nodes, and Leaves:

1. The root node is the topmost node in the tree.
2. Internal nodes contain keys and child pointers.
3. Leaf nodes contain keys and possibly pointers to records or other data.

- In indexing, each key in a B-tree node typically represents a value or range of values, and the associated pointer directs to a data block where records corresponding to the key(s) can be found.
- For example, in a database, the key might be a value in a column, and the pointer might direct to the location of a row or a set of rows in a table.

LET'S START WITH DBMS :)

B trees

A B-tree of order x can have:

- **The max no. of children**

For every node -> x i.e the order of tree.

- **The max no. of keys**

For every node -> x-1

- **The min no of keys**

a. root node -> 1

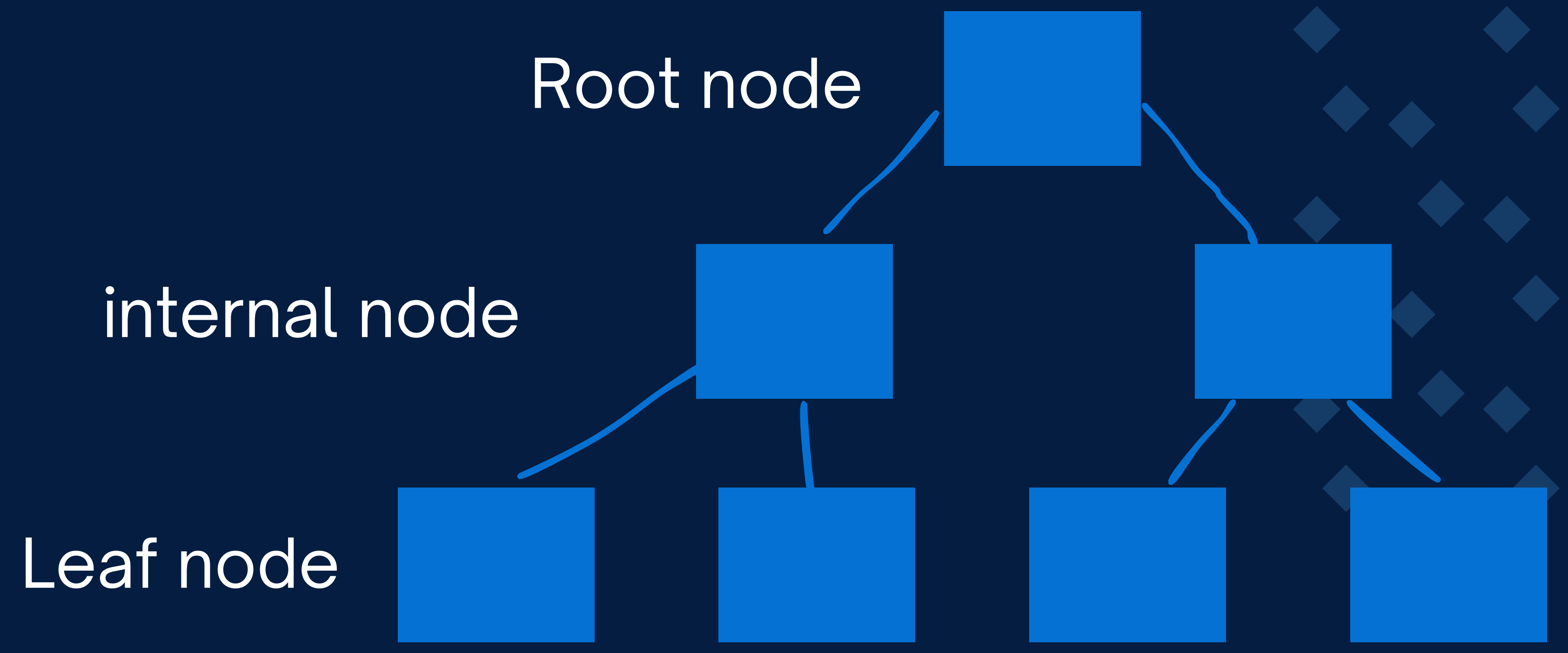
b. other nodes apart from root-> ceiling(m/2) -1

- **The min no. of children**

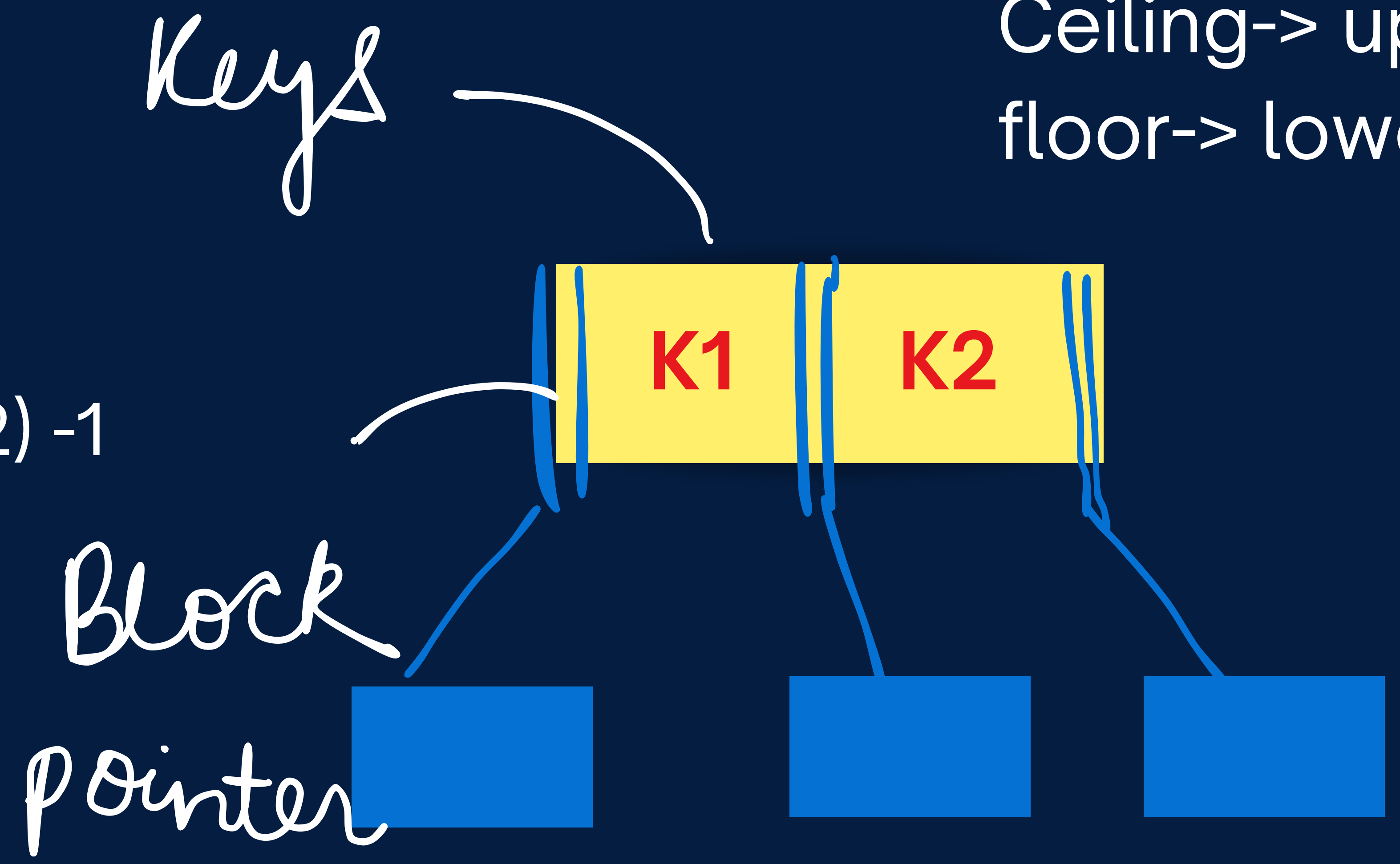
a. root node -> 2

b. Leaf node ->0

c. internal node-> ceiling(m/2)



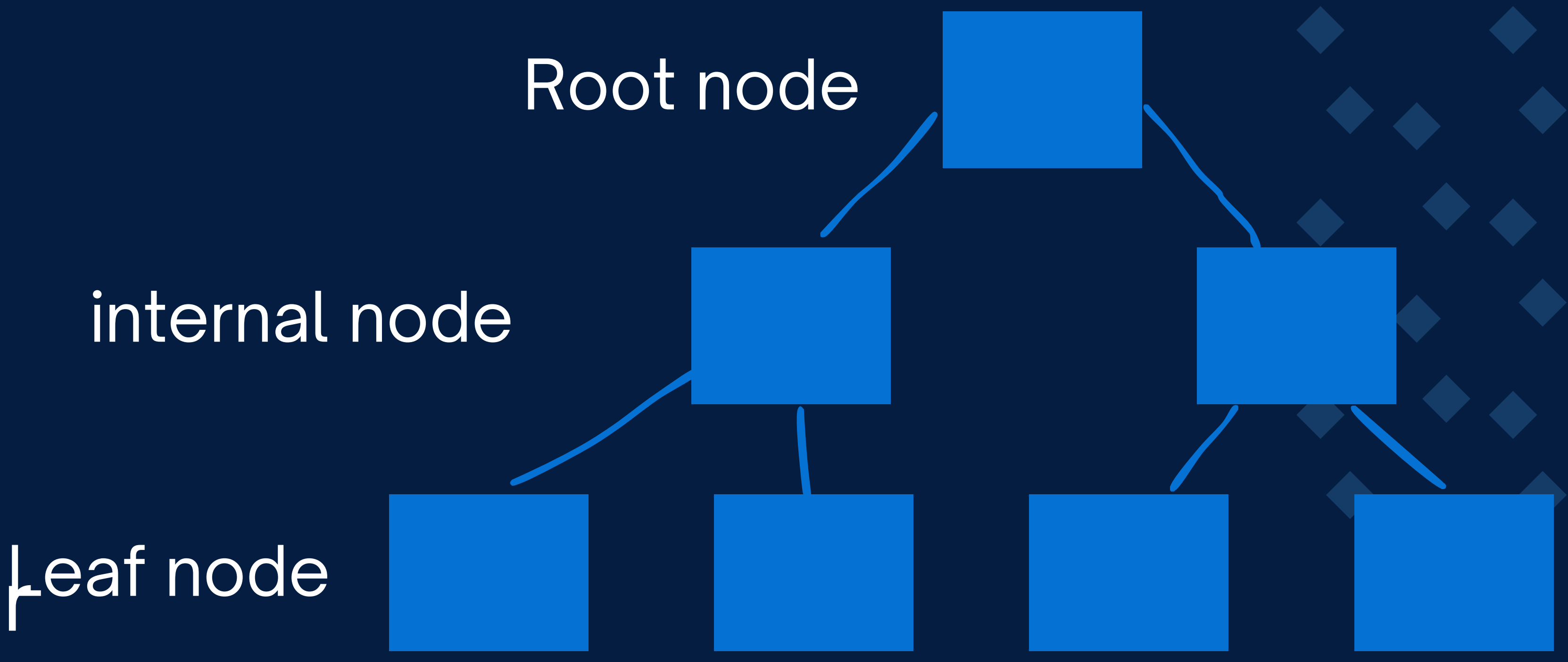
Ceiling-> upper value
floor-> lower value



Insertion in B-tree always happens from leaf node.

LET'S START WITH DBMS :)

B trees



To determine the order of a B-tree when the block size, block pointer size, and data pointer size are given :

$$m \times Pb + (m-1) \times (K+Pd) \leqslant B$$

B- Block size
m- Order of tree
Pb- Block pointer size
K- Key size
Pd- Data pointer size

For a B-tree node with m children:

- Number of Keys: A node with m children can have a maximum of m-1 keys.
- Number of Block Pointers: Each node has m block pointers (pointers to child nodes).
- Number of Data Pointers: Each key has an associated data pointer, so there are m-1 data pointers.

LET'S START WITH DBMS :)

B trees

Let's say you have the following:

- Block size (B) = 1024 bytes
- Block pointer size (Pb) = 8 bytes
- Data pointer size (Pd) = 12 bytes
- Key size (K) = 16 bytes

Find the order of the tree.

Formula to find order : $m \times Pb + (m-1) \times (K+Pd) \leqslant B$

mx8+(m-1)x(16+12) <=1024

So, the order of the B-tree is m = 29

To determine the order of a B-tree when the block size, block pointer size, and data pointer size are given :

$m \times Pb + (m-1) \times (K+Pd) \leqslant B$

B- Block size
m- Order of tree
Pb- Block pointer size
K- Key size
Pd- Data pointer size

LET'S START WITH DBMS :)

Insertion in B-TREE

Steps to insert values in B-tree

1.Start at the root and recursively move down the tree to find the appropriate leaf node where the new value should be inserted.

2. Insert the value into the leaf node in sorted order. If the leaf node has fewer than the maximum allowed keys (order - 1), this step is simple.

3. If the leaf node contains the maximum number of keys after the insertion, it causes an overflow.
Split the Node:
Divide the node into two nodes. The middle key (median) is pushed up to the parent node.

- The left half of the original node stays in place, while the right half forms a new node.
- Insert the Median into the Parent:
- If the parent node also overflows after this insertion, recursively split the parent node and propagate the median up the tree.

4.If the root node overflows (which can happen if it already has the maximum number of keys), split it into two nodes, and the median becomes the new root. This increases the height of the B-tree by one.

LET'S START WITH DBMS :)

Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Max children= order of tree=3
Max keys node can have=order-1=3-1=2

Insertion in B-tree happens from leaf node and values are also inserted in sorted order. The element in left of root would be less than root and the element in right would be greater than root

Step 1: Start at the root and recursively move down the tree to find the appropriate leaf node where the new value should be inserted. Insert the value into the leaf node in sorted order. If the leaf node has fewer than the maximum allowed keys (order - 1), this step is simple.

1	4

A B-tree of order x can have:

- Every node will have max x children i.e the order of tree.
- Every node can have max (x-1) keys
- The min no of keys

a. root node -> 1
b. other nodes apart from root-> ceiling(m/2) -1

- Min children

a. root node -> 2
b. Leaf node ->0
c. internal node-> ceiling(m/2)

Max children= order of tree=3
Max keys node can have=order-1=3-1=2

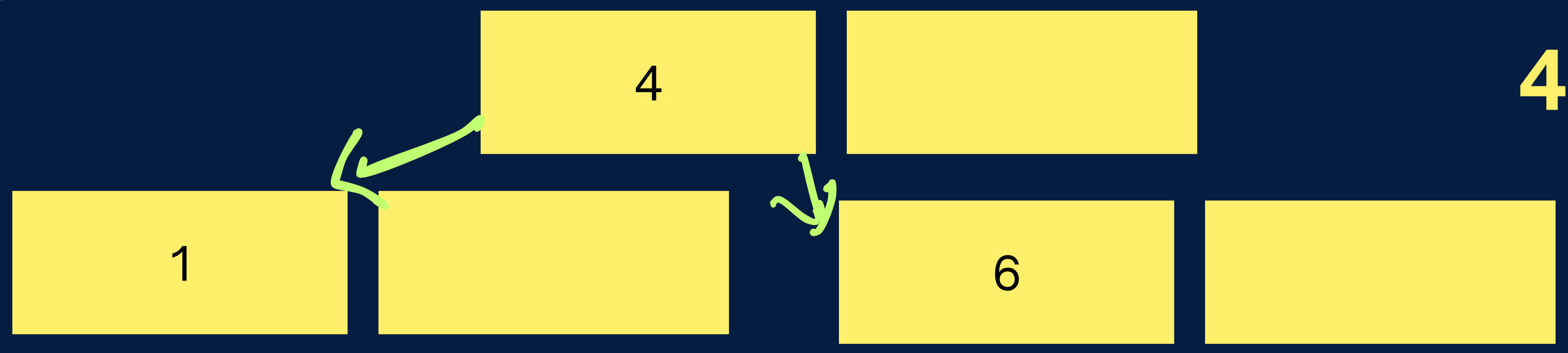
LET'S START WITH DBMS :)

Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Step 2: If the leaf node contains the maximum number of keys after the insertion, it causes an overflow.
Split the Node: Divide the node into two nodes. The middle key (median) is pushed up to the parent node.

- The left half of the original node stays in place, while the right half forms a new node.
- Insert the Median into the Parent:
- If the parent node also overflows after this insertion, recursively split the parent node and propagate the median up the tree.



4 (left- 1 , right- 6)

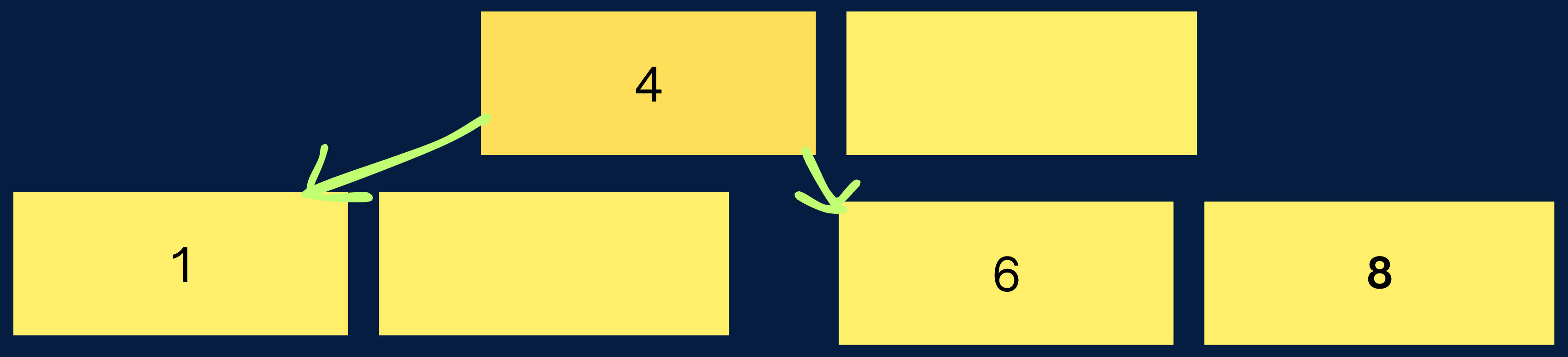
Max children= order of tree=3
Max keys node can have=order-1=3-1=2

LET'S START WITH DBMS :)

Insertion in B-TREE

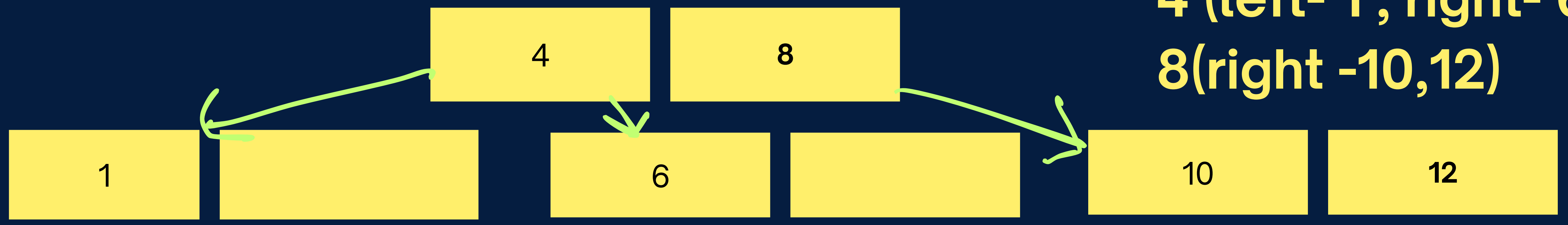
Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

4 (left- 1 , right- 6,8)



Follow step-2 again

4 (left- 1 , right- 6)
8(right -10,12)



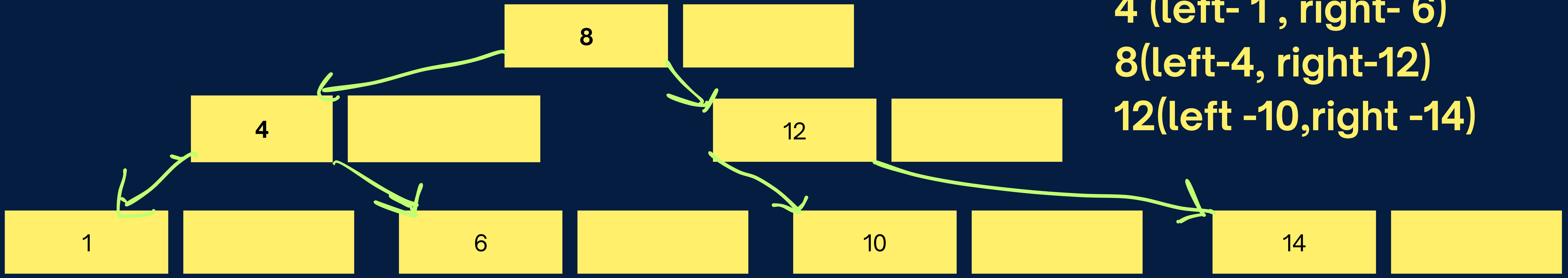
LET'S START WITH DBMS :)

Max children= order of tree=3
Max keys node can have=order-1=3-1=2

Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Step 3: If the root node overflows (which can happen if it already has the maximum number of keys), split it into two nodes, and the median becomes the new root. This increases the height of the B-tree by one.



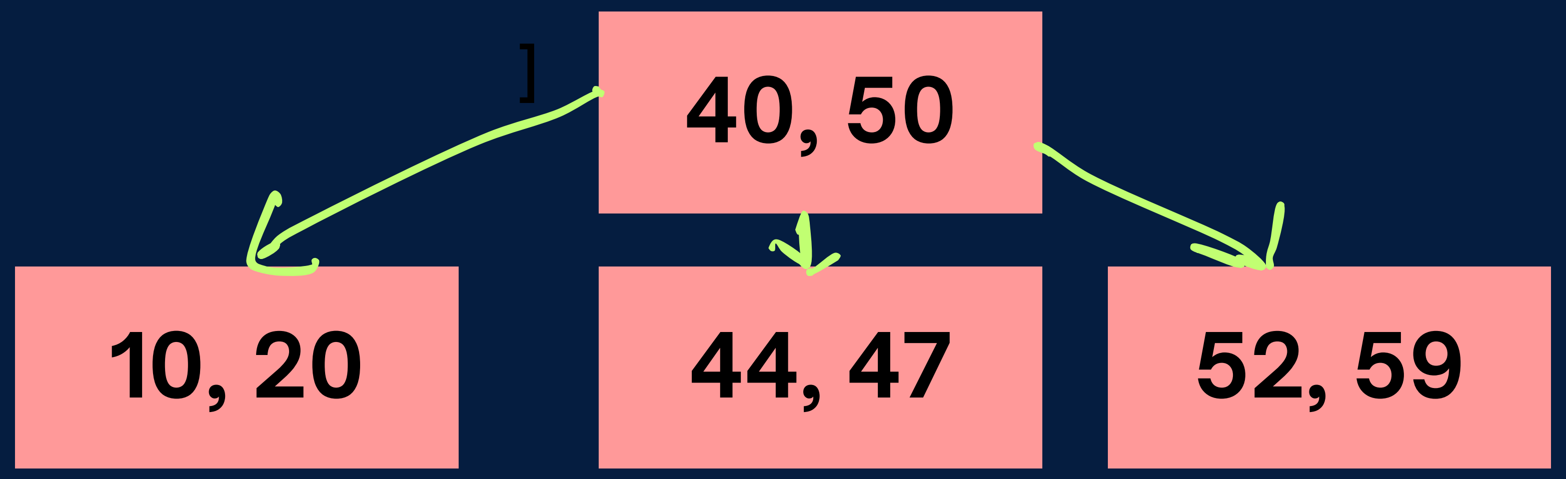
4 (left- 1 , right- 6)
8(left-4, right-12)
12(left -10,right -14)

LET'S START WITH DBMS :)

Deletion in B-TREE

maximum of (m-1)=3 keys
and minimum of (ceil(m/2)-1)=1 key

Consider a B-tree of order 4

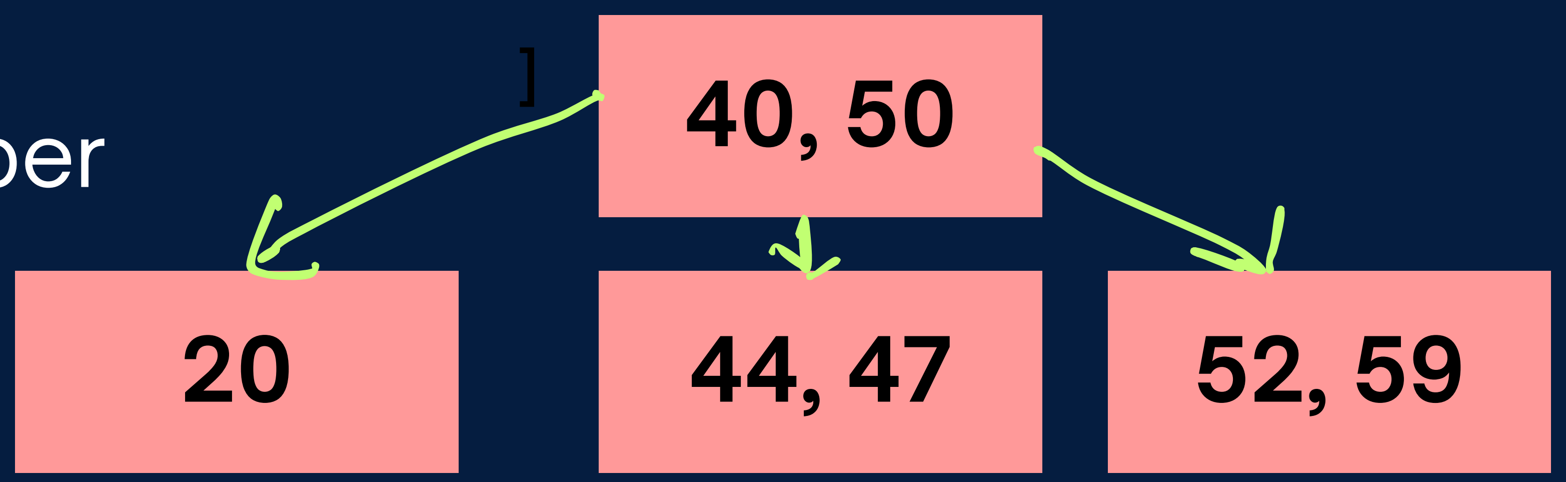


Step 1: Begin at the root and recursively move down the tree to find the node that contains the key to be deleted.

Step 2 :

Case 1: The Key is in a Leaf Node **(Delete 10)**

- Simply remove the key from the leaf node.
- If the node still has the minimum required number of keys (i.e., at least ceil(order/2) - 1 keys), the deletion is complete.



LET'S START WITH DBMS :)

Deletion in B-TREE

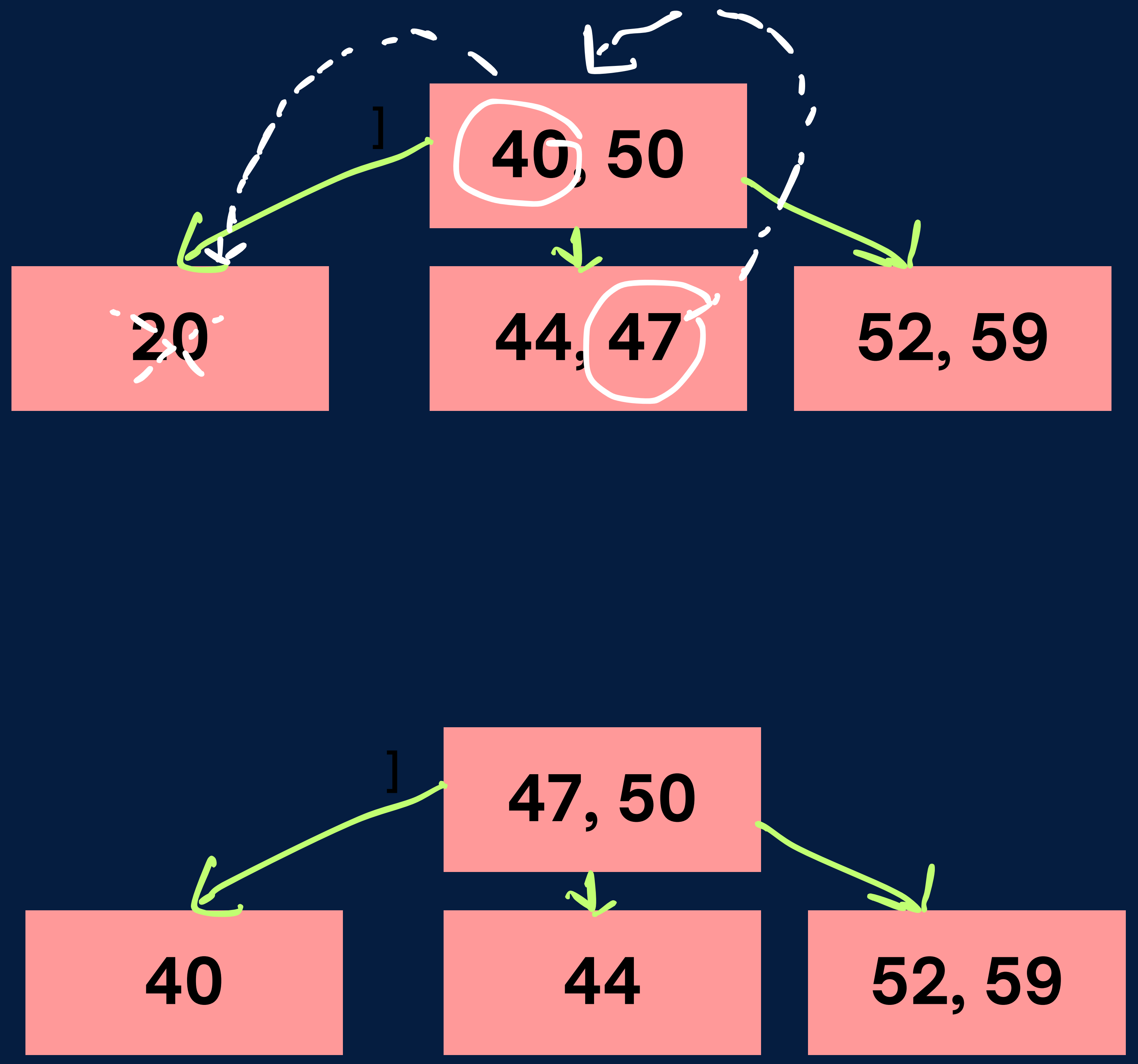
Consider a B-tree of order 4

maximum of (m-1)=3 keys
and minimum of (ceil(m/2)-1)=1 key

Step 2 :
Case 2: The Key is in a Leaf Node **(Delete 20)**

- If the deletion causes the node to have fewer than the minimum number of keys, proceed to the Borrowing or Merging step.

1. If the node has a sibling with more than the minimum number of keys, you can borrow a key from this sibling. The parent key between the node and the sibling moves down to the node, and a key from the sibling moves up to the parent.



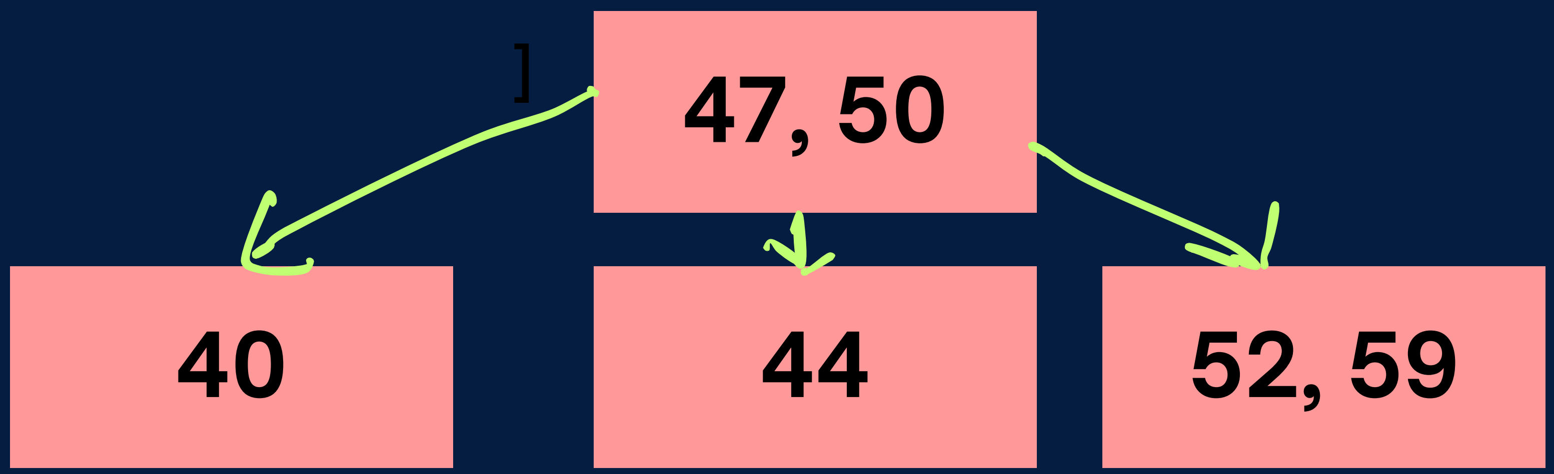
LET’S START WITH DBMS :)

Deletion in B-TREE

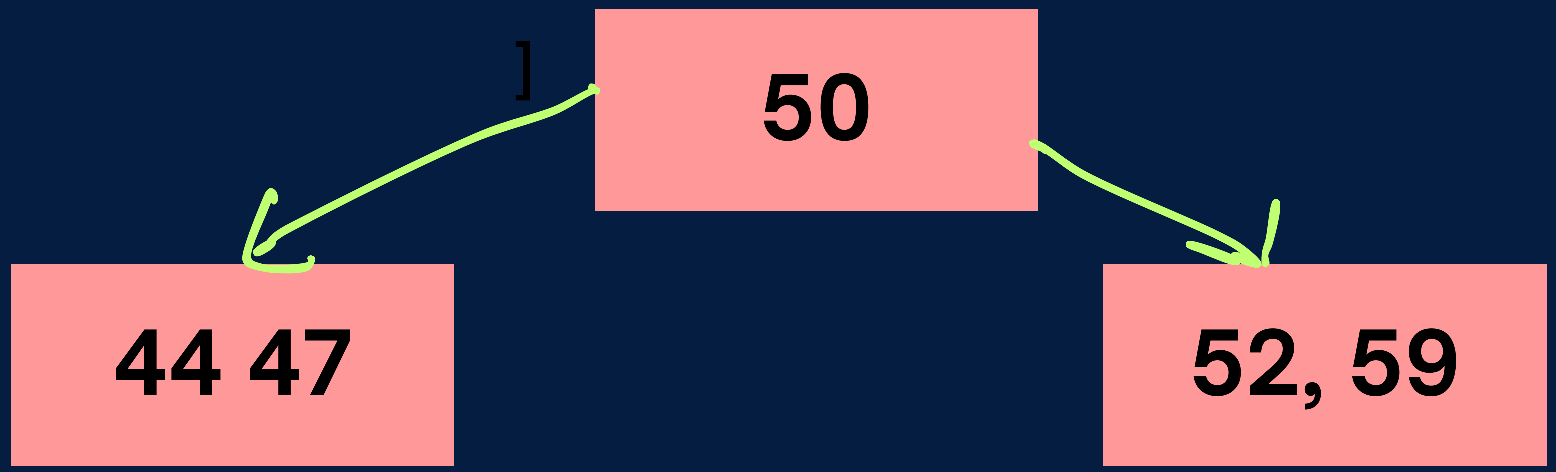
maximum of (m-1)=3 keys
and minimum of (ceil(m/2)-1)=1 key

Consider a B-tree of order 4

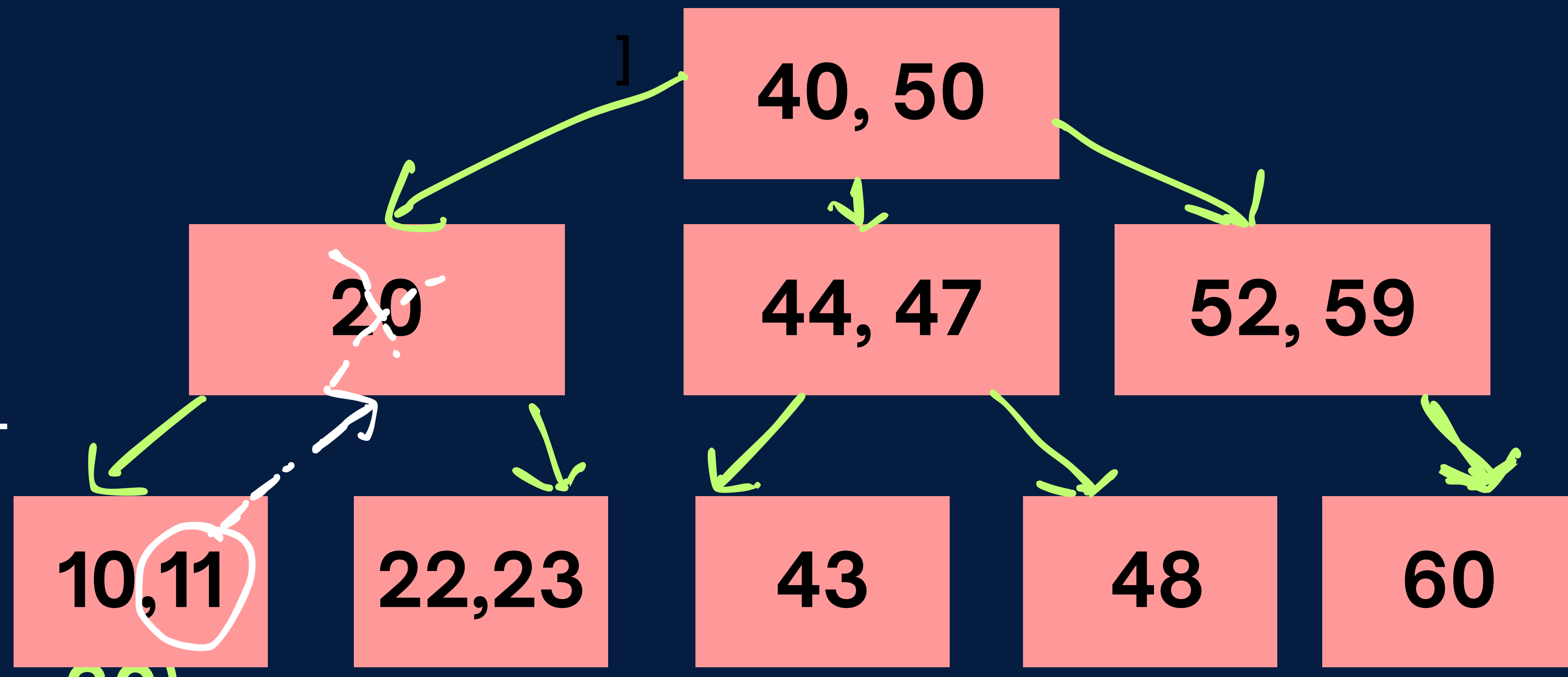
Step 2 :
Case 2: The Key is in a Leaf Node **(Delete 40)**



2. If borrowing is not possible (i.e., the sibling also has the minimum number of keys), merge the node with a sibling. The key from the parent that separates the two nodes moves down into the newly merged node.If this causes the parent to have too few keys, repeat the borrowing or merging process at the parent level.



LET'S START WITH DBMS :)



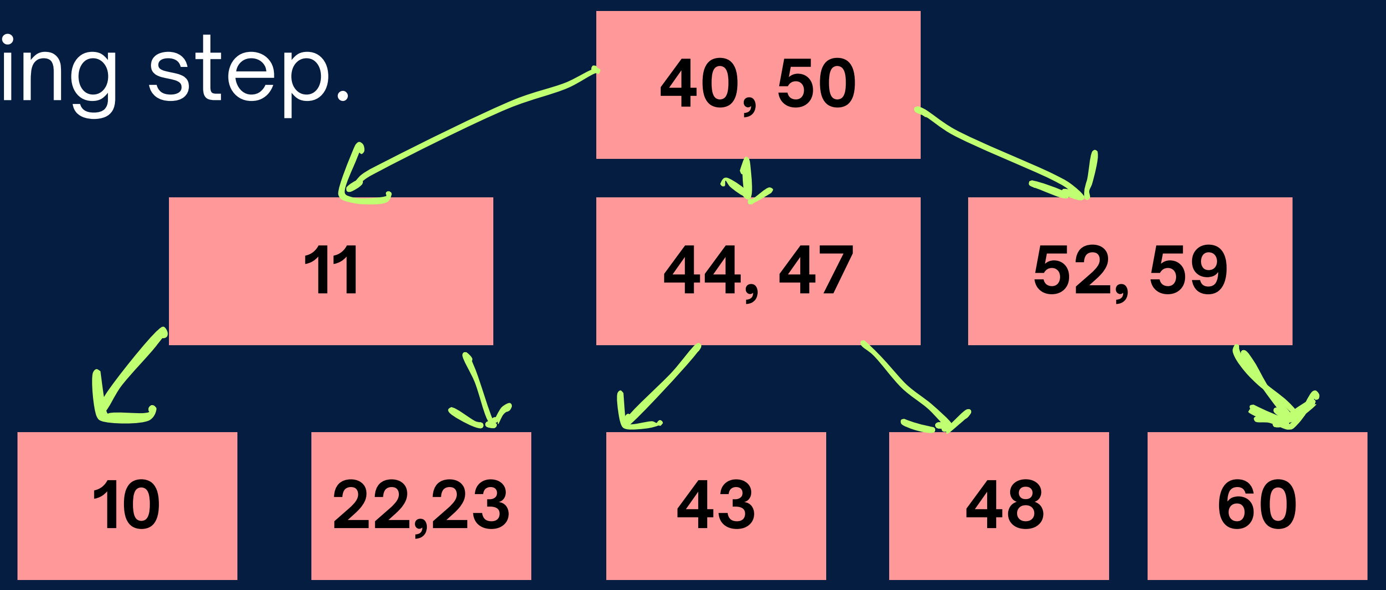
Deletion in B-TREE

Steps on how we can delete elements in B-tree

Step 2:

Case 2: The Key is in an Internal Node **(Delete 20)**

- Replace the key with its predecessor (the largest key in the left subtree) or its successor (the smallest key in the right subtree).
- Delete the predecessor or successor key from the corresponding subtree.
- If this causes an underflow (i.e., a node has too few keys), proceed to the Borrowing or Merging step.



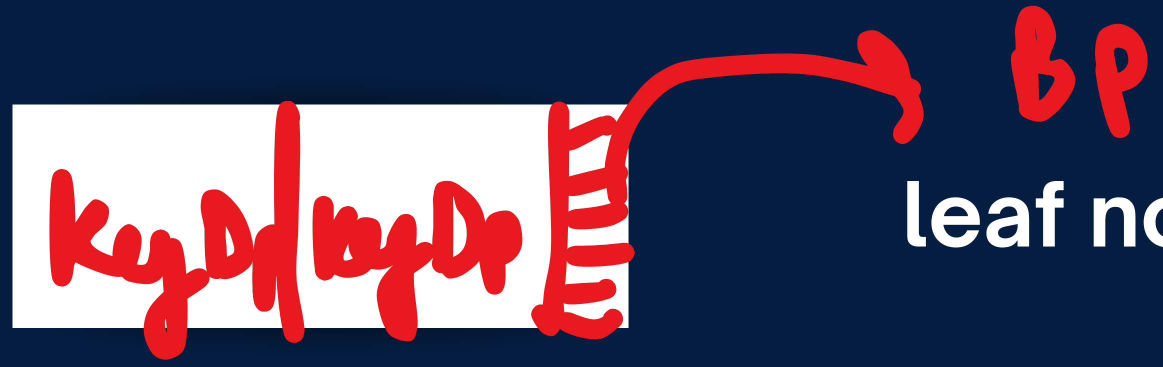
LET'S START WITH DBMS :)

B+ Tree

A B+ tree is an extension of the B-tree and is commonly used in databases and file systems to maintain sorted data and allow for efficient insertion, deletion, and search operations. B+ tree is a balanced tree, meaning all leaf nodes are at the same level

The key difference between a B+ tree and a B-tree lies in how they store data and how leaf nodes are structured.

1. In a B+ tree, all actual data (or references to data) are stored in the leaf nodes. Internal nodes only store keys

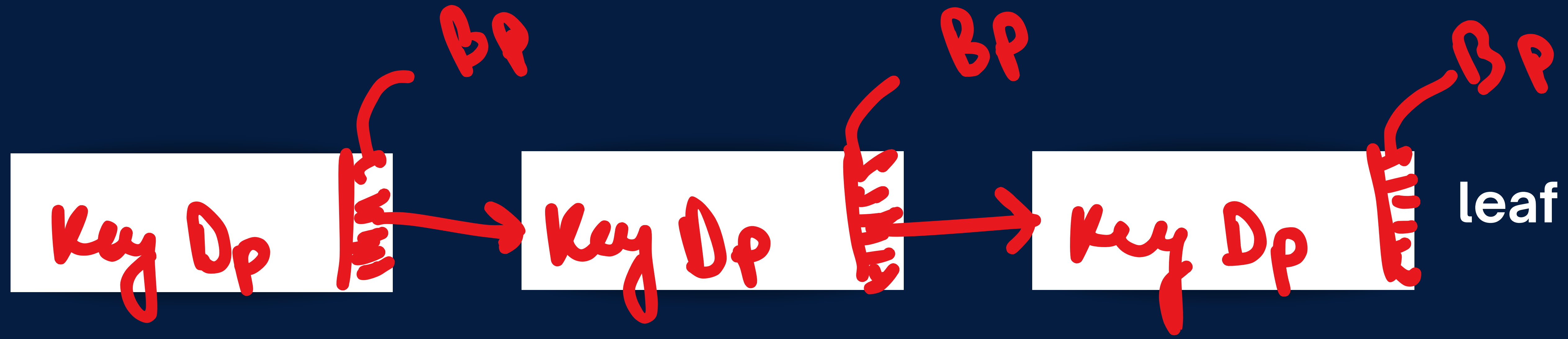


leaf node

LET'S START WITH DBMS :)

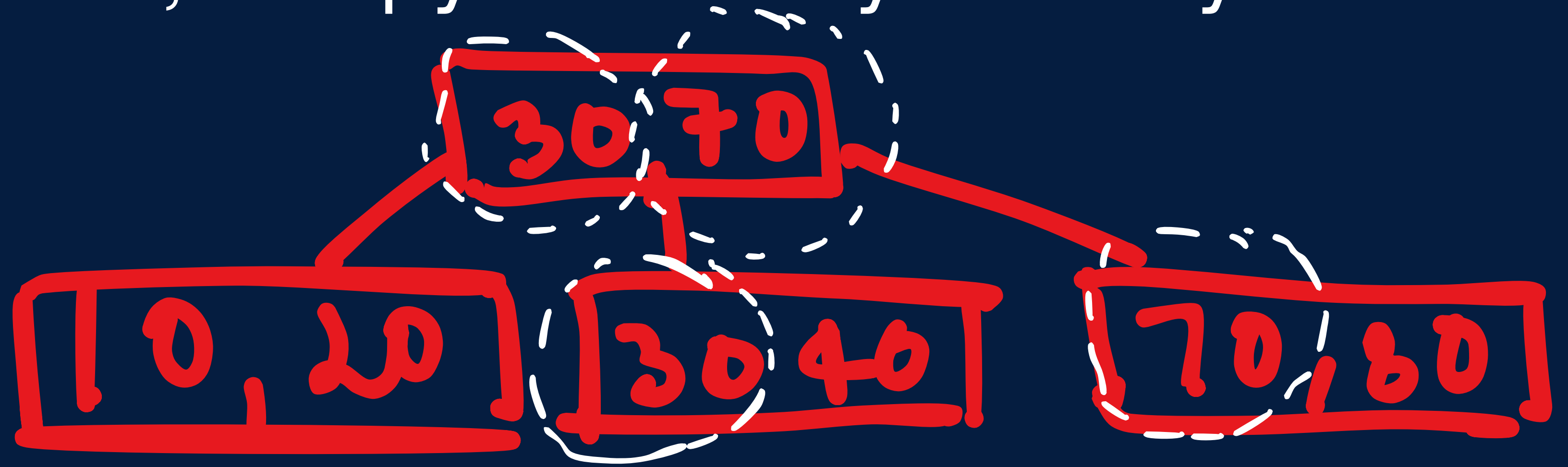
B+ Tree

2. In a B+ tree, Leaf nodes are linked together in a linked list fashion, allowing for efficient sequential access, one leaf node will have only 1 Bp.



leaf nodes

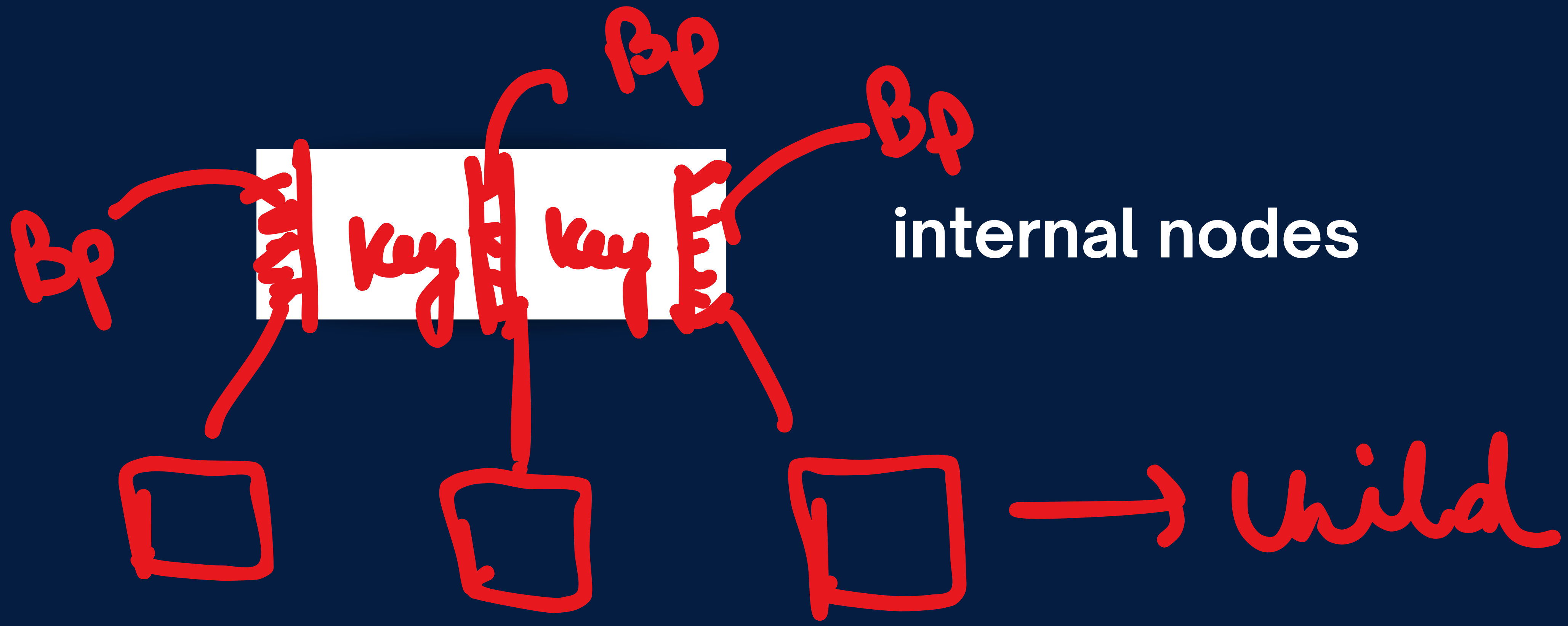
During inserting value in B+ tree, a copy of the key is always stored in the leaf node.



LET'S START WITH DBMS :)

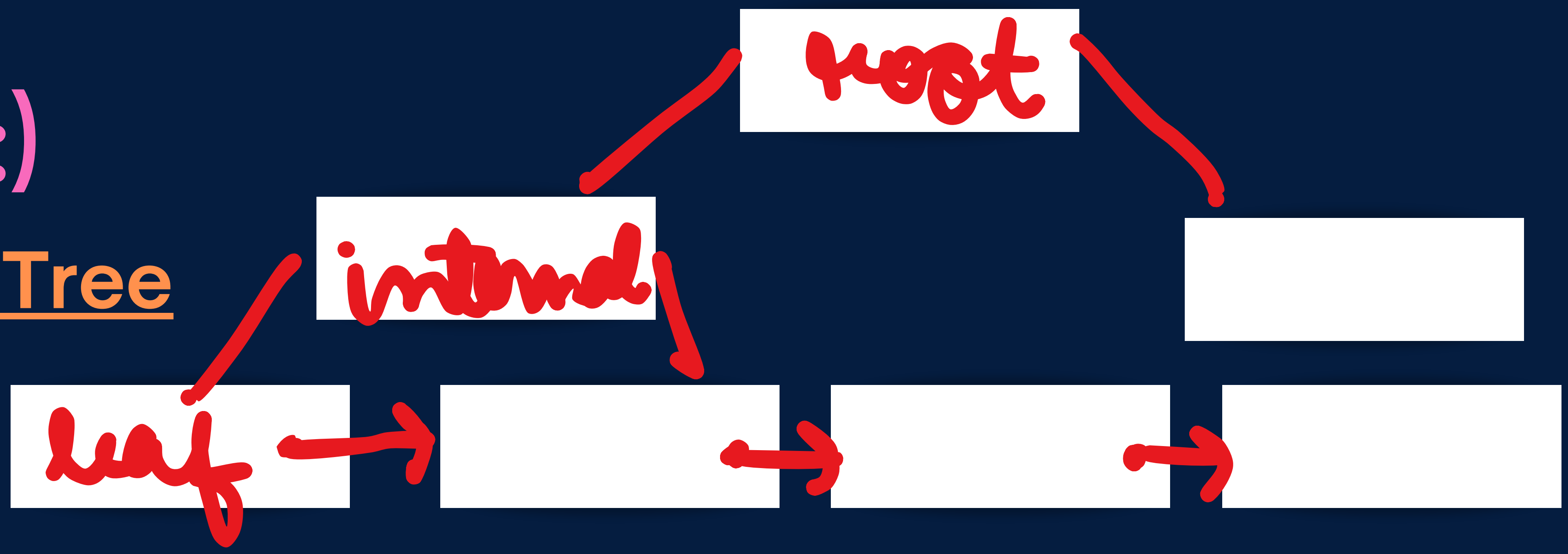
B+ Tree

3. Internal nodes do not store data pointers, only keys and child pointers. This allows more keys to be stored in each internal node, leading to a lower height and more efficient operations.



LET'S START WITH DBMS :)

B+ Tree



Structure of a B+ Tree:

- Root Node:
  - The top node of the B+ tree, which points to the first level of internal nodes or directly to leaf nodes if the tree has only one level.
- Internal Nodes:
  - These nodes contain only keys and pointers to child nodes. They guide the search process down to the correct leaf node.
- Leaf Nodes:
  - These nodes contain keys and data pointers. Each leaf node stores a pointer to the next leaf node, enabling quick traversal of records.

LET'S START WITH DBMS :)

B+ Tree

Advantages :

1. B+ trees have a balanced structure, meaning all leaf nodes are at the same level. This balance ensures that search operations require logarithmic time relative to the number of keys, making it very efficient even for large datasets.
2. The structure of the B+ tree allows for direct access to data. Since the internal nodes contain only keys, searching for a specific value can be done quickly by navigating through the tree down to the leaf node where the data is stored.

LET'S START WITH DBMS :)

B- Block size
m- Order of tree
Pb- Block pointer size
K- Key size
Pd- Data pointer size

B+ Tree

Order of B+ tree

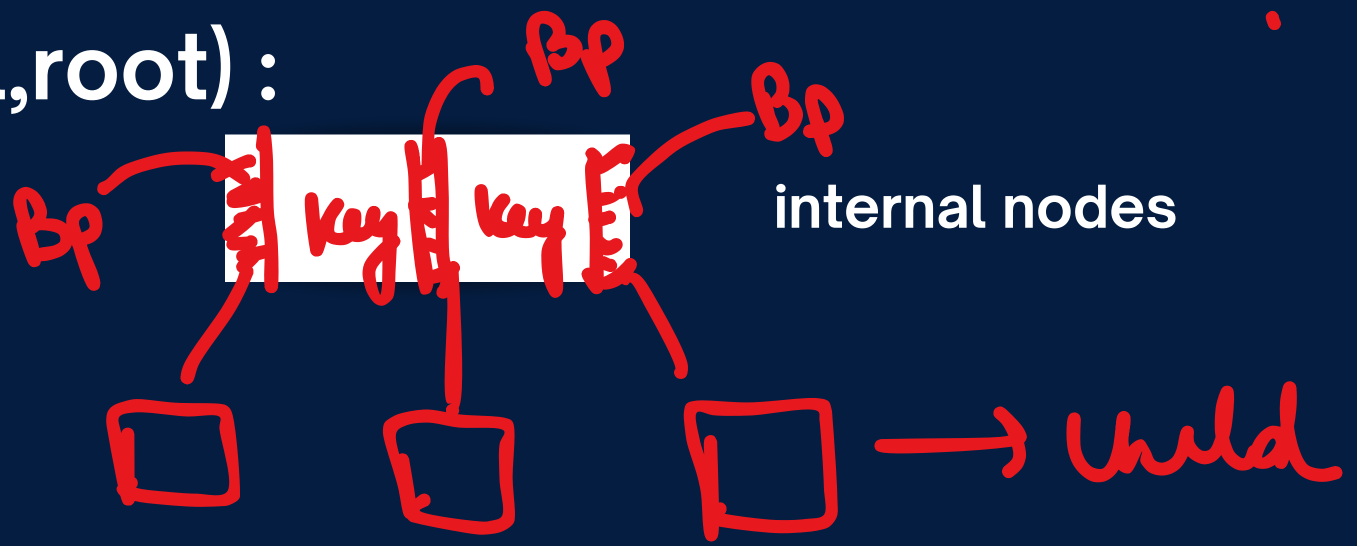
Order of Leaf node :

$1 \times Pb + M(k+Pd) <= B$

leaf nodes

Order of non-Leaf node(internal,root) :

$m \times Pb + (m-1)k <= B$



internal nodes