Dynamic Multilevel Indexes Using B-Trees

B-Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order *p* can have at most *p−1* search values

B-Trees

- B-tree of order **p** has the following properties
- Every node has at most **p** children

  Min ***children***: leaf → 0

  Root →2

  Internal nodes → $\lceil p/2 \rceil$
- Every node has max (**p-1**) ***keys***

  Min keys: root node →1

  All other nodes → $\lceil p/2 \rceil$ -1 key

B-Trees

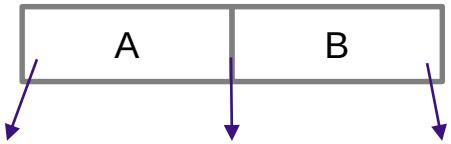
- B-tree of order **p=3**
- Every node has at most **3** children

Min ***children***: leaf → 0

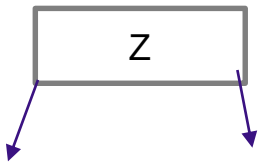
Root →2
Internal nodes →2

- Every node has max ***2 keys***

Internal nodes min 1 key, root node min 1 key



max 2 keys, 3 children



min 1 key, 2 children

B-Trees

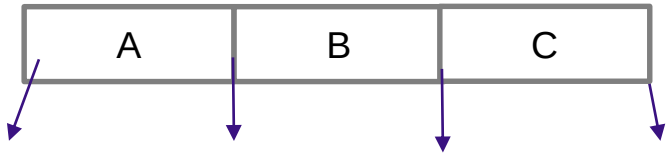
- B-tree of order **p=4**
- Every node has at most **4** children

Min ***children***: leaf → 0

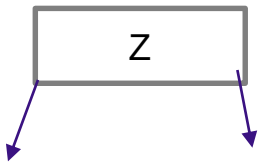
Root →2
Internal nodes →2

- Every node has max **3 *keys***

Internal nodes min 1 key, root node min 1 key



max 3 keys, 4 children



min 1 key, 2 children

B-Trees

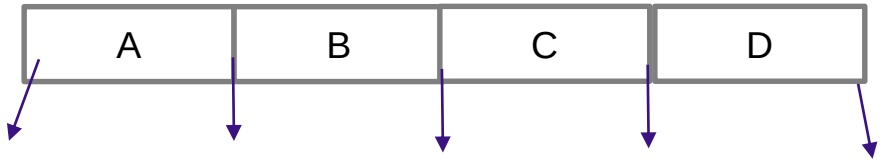
- B-tree of order **p=5**
- Every node has at most **4** children

Min ***children***: leaf → 0

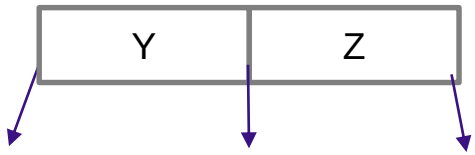
Root →2
Internal nodes →XX

- Every node has max **4 *keys***

Internal nodes min 2 key, root node min 1 key



max 4 keys, 5 children



min 2 key, 3 children

Dynamic Multilevel Indexes Using B-Trees

- An insertion into a node that **is not full** is quite efficient
- If a node **is full** the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- **When overflow**: split and promotion

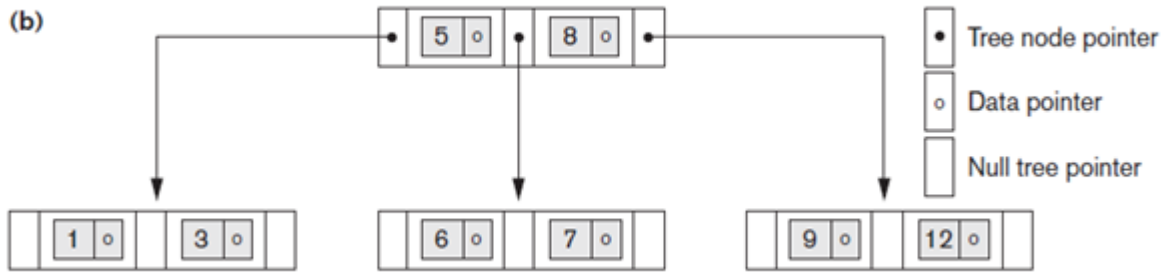
  Split the overflow page into two nodes

  Promote a key to a parent node
- If the promotion in the previous step causes additional overflow, then repeat the **split-promotion**

B-Tree Structures

Figure 17.10 B-tree structures

(b) A B-tree of order *p*=*3*. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6



B-tree

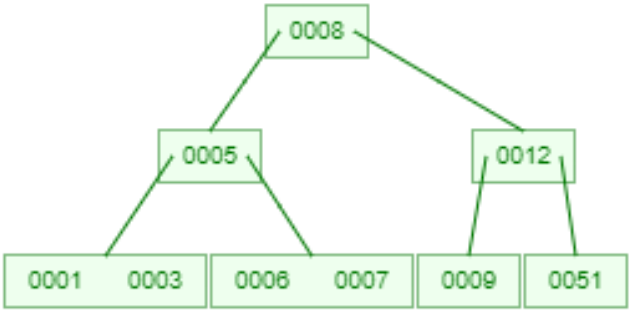
- Create a B tree with p=3.

51, 8, 5, 1, 7, 3, 12, 9, 6

B-tree

- Create a B tree with m=3.

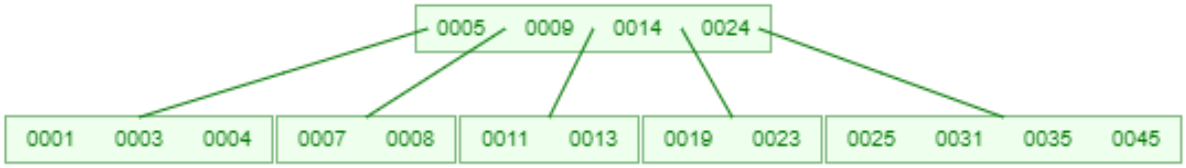
51, 8, 5, 1, 7, 3, 12, 9, 6



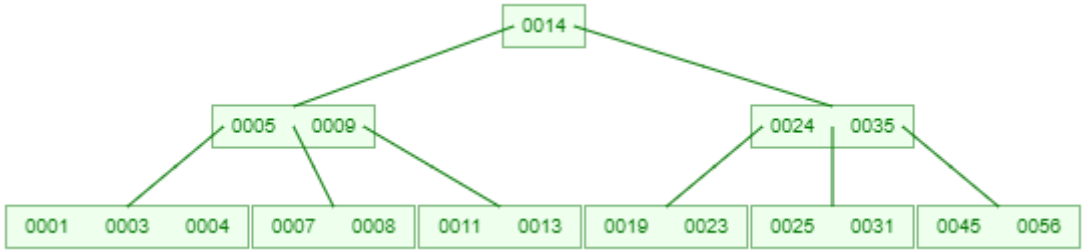
B-Trees example

- Insert the following keys to a B-Tree of order 5.

{3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56 }



56



B-tree delete

Follow the below steps to delete a key in any B-tree:

- Search B-tree to find **the key** to be deleted.
- Swap the key with its immediate successor, if the key is not in a leaf page.

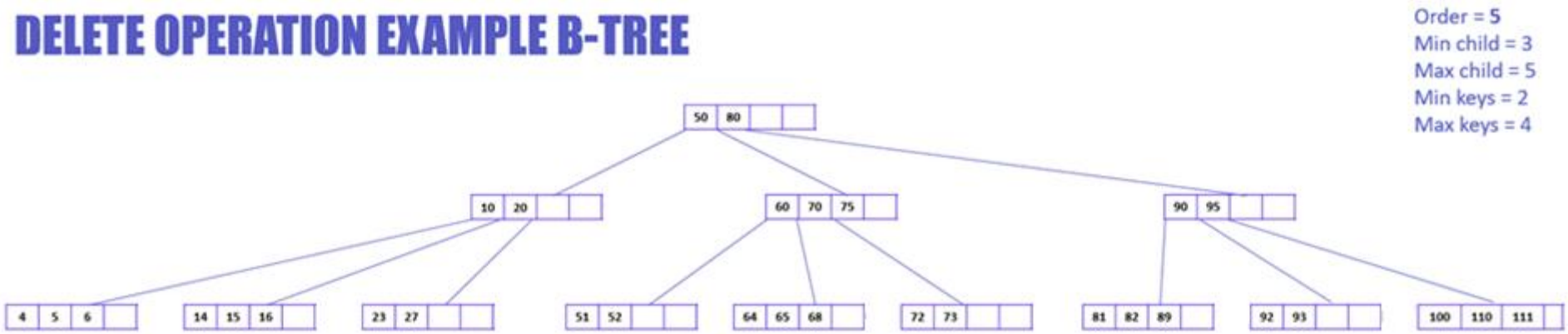
  Note only keys in a leaf may be deleted.
- **When underflow**: redistribution or concatenation

Redistribute keys among an adjacent sibling page, the parent page, and the underflow page if possible (need a rich sibling).

Otherwise, **concatenate** with an adjacent page, demoting a key from the parent page to the newly formed page.

- If the demotion causes **underflow**, repeat redistribution-concatenation.

B-tree delete



B-tree delete

Follow the below steps to delete a key in any B-tree:

- Search B-tree to find **the key** to be deleted.
- Three conditions are applied based on the location of the target key

CASE I: If the target key is in the leaf node

CASE II: If the target key is in an internal node

CASE III: If the target key is in a root node

B-tree delete

CASE I: If the target key is in the leaf node

a) Target is in the leaf node, more than min keys.

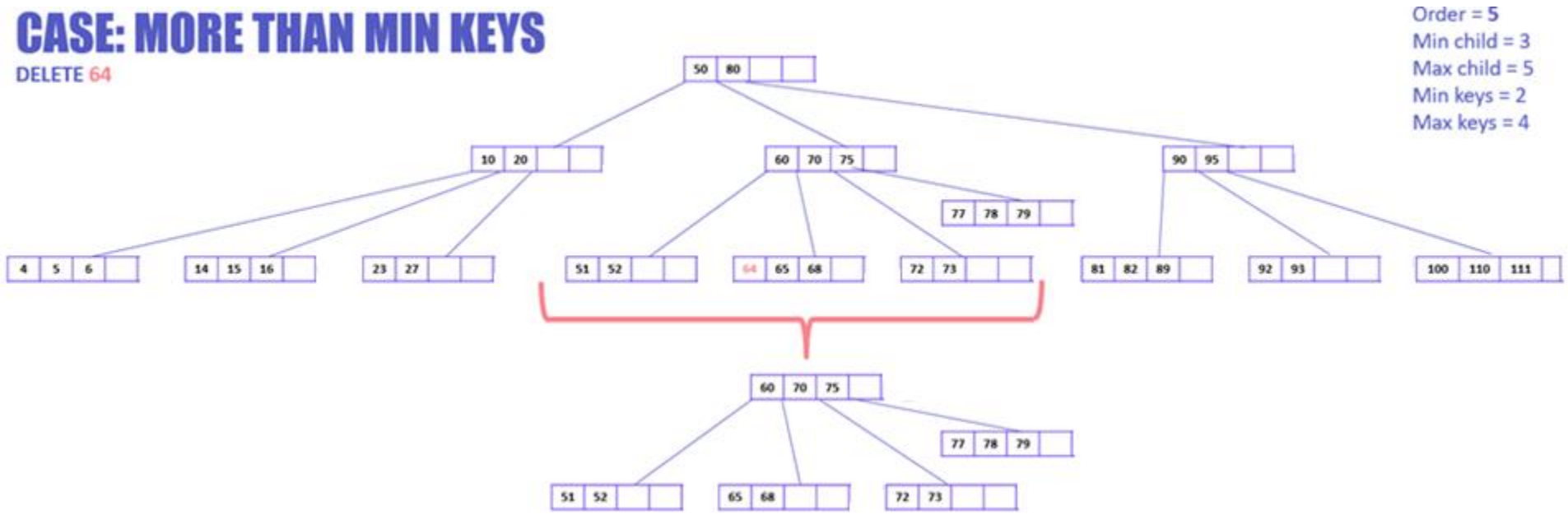
b) Target is in the leaf node, it has min key nodes

c) Target is in the leaf node, but no siblings have more than min number of keys

B-tree delete

CASE I: If the target key is in the leaf node

a) Target is in the leaf node, more than min keys.
Deleting this will not violate the property of B Tree



B-tree delete

CASE I: If the target key is in the leaf node

b) Target is in leaf node, it has min key nodes

Deleting this will violate the property of B Tree

Target node can borrow a key from the immediate left node, or the immediate right node (sibling)

The sibling will say yes if it has more than the minimum number of keys

The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value

B-tree delete

CASE I: If the target key is in the leaf node

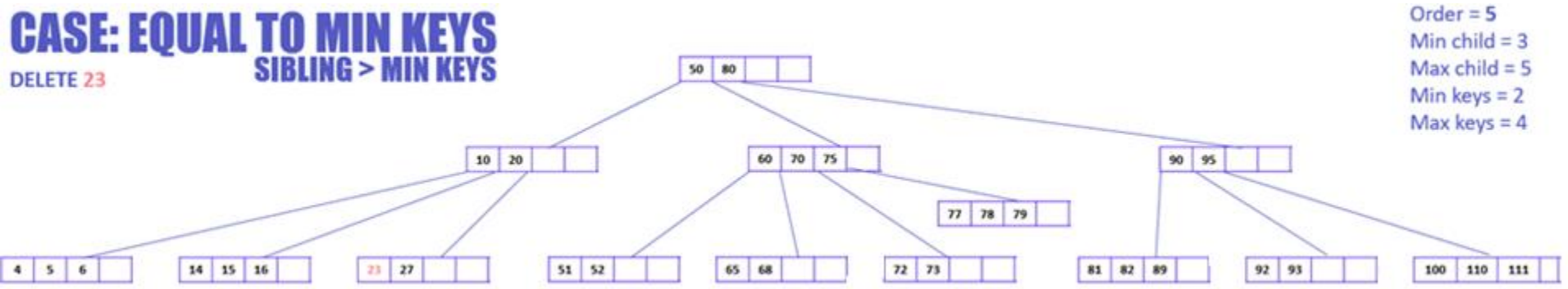
b) Target is in leaf node, it has min key nodes

Deleting this will violate the property of B Tree

Target node can borrow a key from the immediate left node, or the immediate right node (sibling)

The sibling will say yes if it has more than the minimum number of keys

The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value



The target node has keys equal to minimum keys, so cannot delete it directly as it will violate the conditions

B-tree delete

CASE I: If the target key is in the leaf node

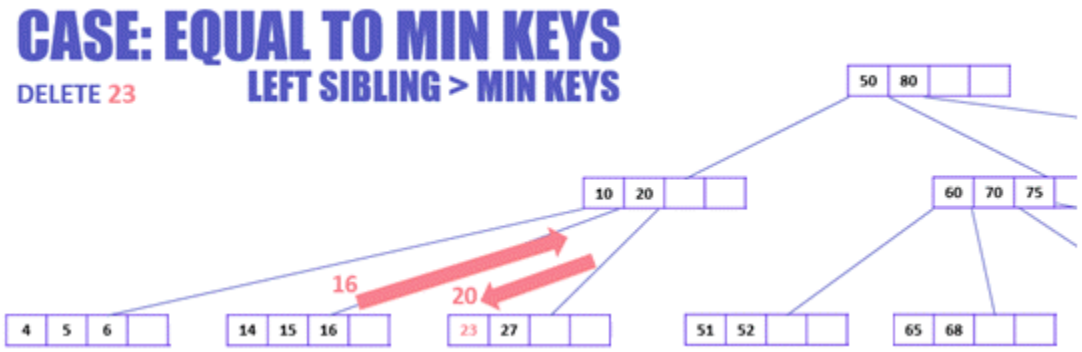
b) Target is in leaf node, it has min key nodes

Deleting this will violate the property of B Tree

Target node can borrow a key from the immediate left node, or the immediate right node (sibling)

The sibling will say yes if it has more than the minimum number of keys

The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value



B-tree delete

CASE I: If the target key is in the leaf node

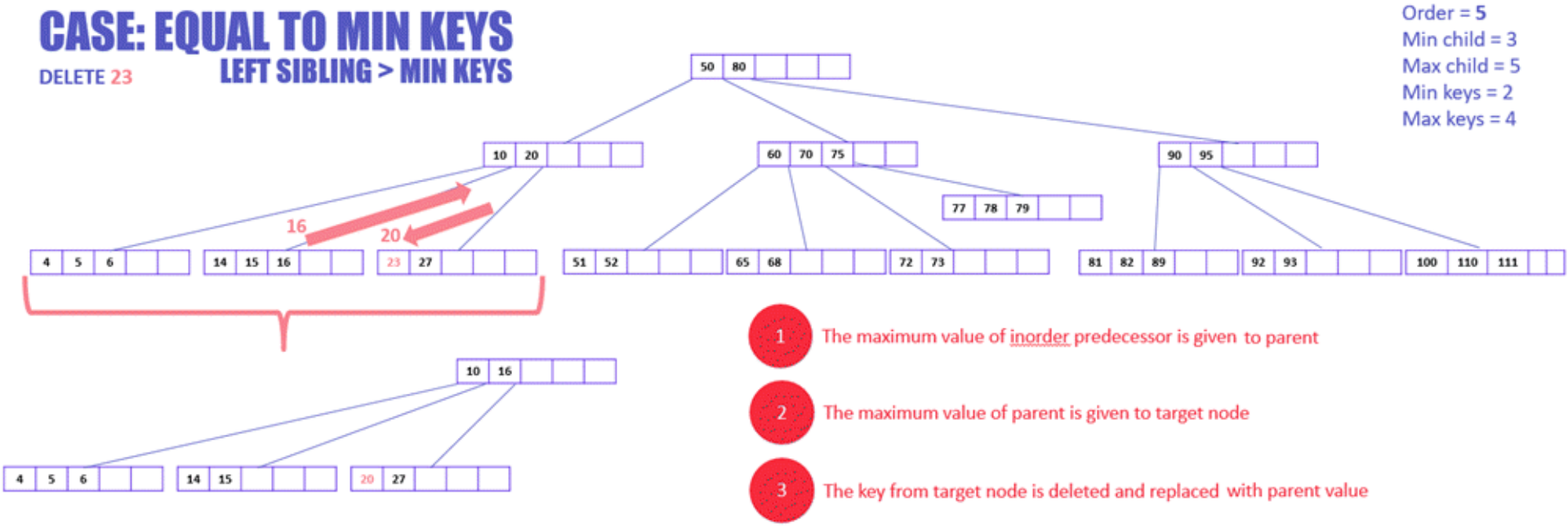
b) Target is in leaf node, it has min key nodes

Deleting this will violate the property of B Tree

Target node can borrow a key from the immediate left node, or the immediate right node (sibling)

The sibling will say yes if it has more than the minimum number of keys

The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value



B-tree delete

CASE I: If the target key is in the leaf node

c) Target is in the leaf node, but no siblings have more than min number of keys

Search for key

Merge with siblings and the minimum of parent nodes

Total keys will be now more than min

The target key will be replaced with the minimum of a parent node

B-tree delete

CASE I: If the target key is in the leaf node

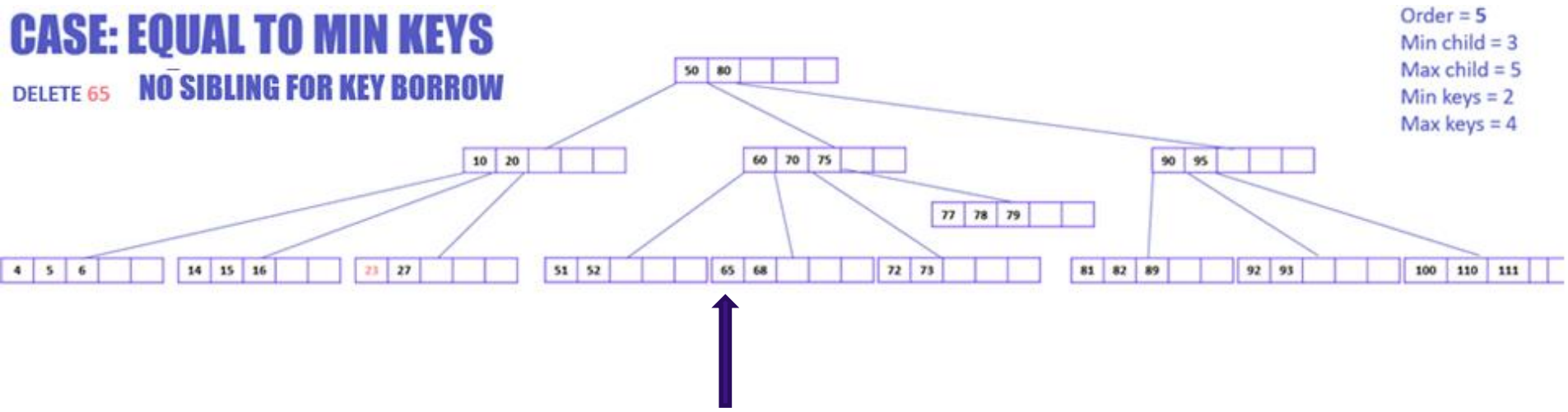
c) Target is in the leaf node, but no siblings have more than min number of keys

Search for key

Merge with siblings and the minimum of parent nodes

Total keys will be now more than min

The target key will be replaced with the minimum of a parent node



B-tree delete

CASE I: If the target key is in the leaf node

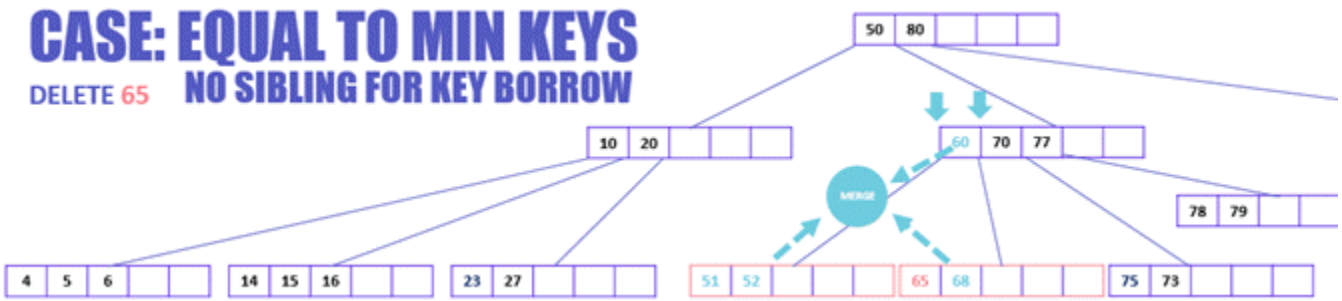
c) Target is in the leaf node, but no siblings have more than min number of keys

Search for key

Merge with siblings and the minimum of parent nodes

Total keys will be now more than min

The target key will be replaced with the minimum of a parent node



B-tree delete

CASE I: If the target key is in the leaf node

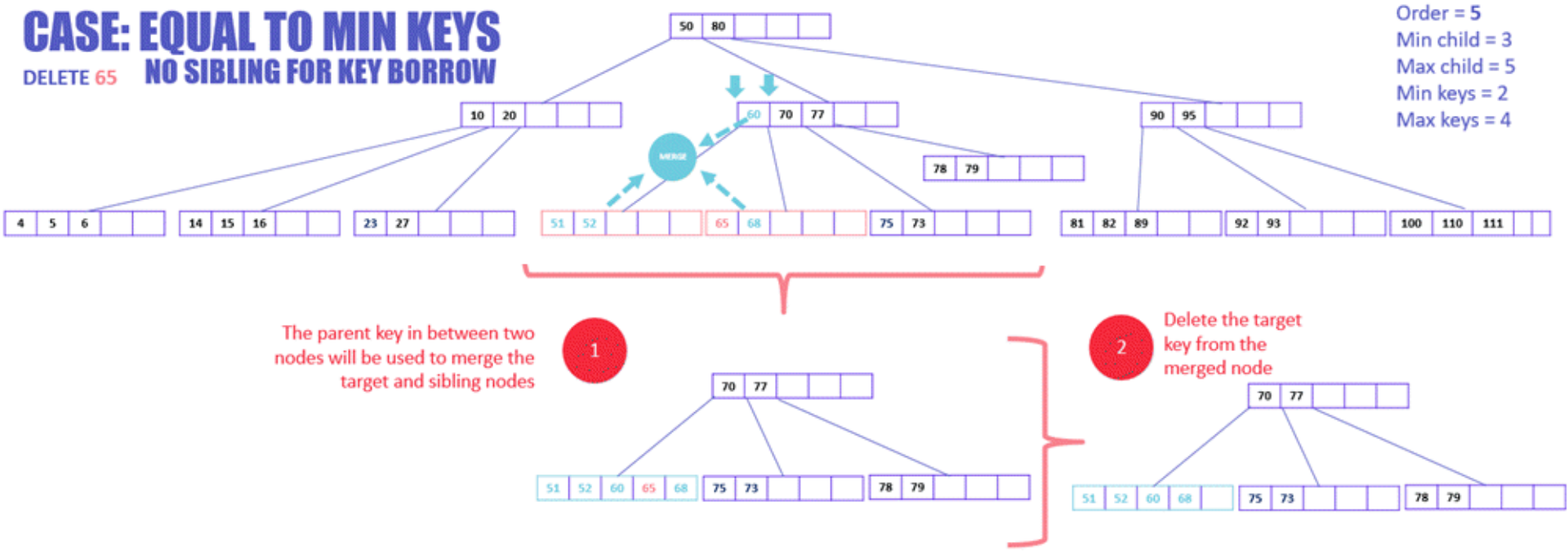
c) Target is in the leaf node, but no siblings have more than min number of keys

Search for key

Merge with siblings and the minimum of parent nodes

Total keys will be now more than min

The target key will be replaced with the minimum of a parent node



B-tree delete

CASE II: If the target key is in an internal node

Either choose, in- order predecessor or in-order successor

In case the of in-order predecessor, the maximum key from its left subtree will be selected. If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the **max of the in-order predecessor**

In case of in-order successor, the minimum key from its right subtree will be selected. If the target key's in-order predecessor does not have more than min keys, look for in-order successor's **minimum** key.

If the target key's in-order predecessor and successor both *have less than min keys*, then **merge the predecessor and successor**.

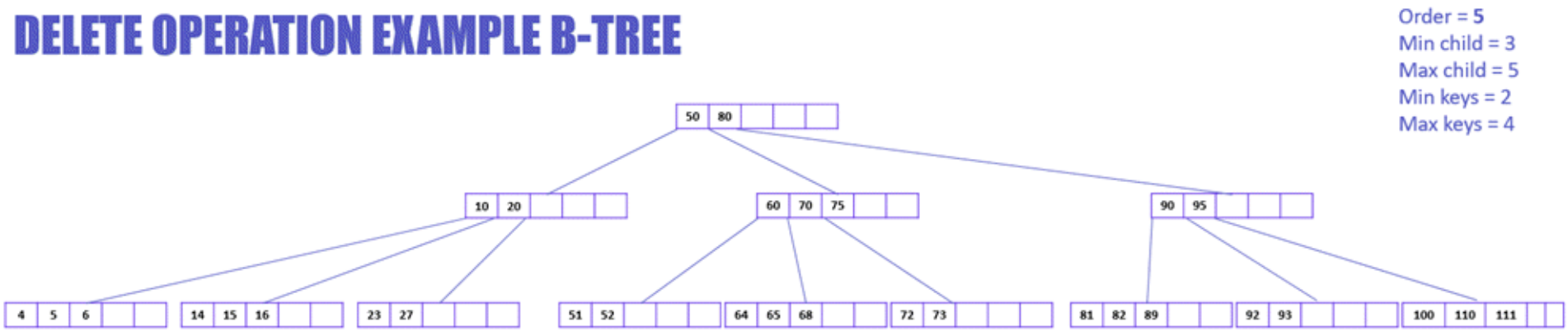
B-tree delete

CASE II: If the target key is in an internal node

Either choose, in- order predecessor or in-order successor

In case the of in-order predecessor, the maximum key from its left subtree will be selected. If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the **max of the in-order predecessor**

In case of in-order successor, the minimum key from its right subtree will be selected. If the target key's in-order predecessor does not have more than min keys, look for in-order successor's **minimum** key.



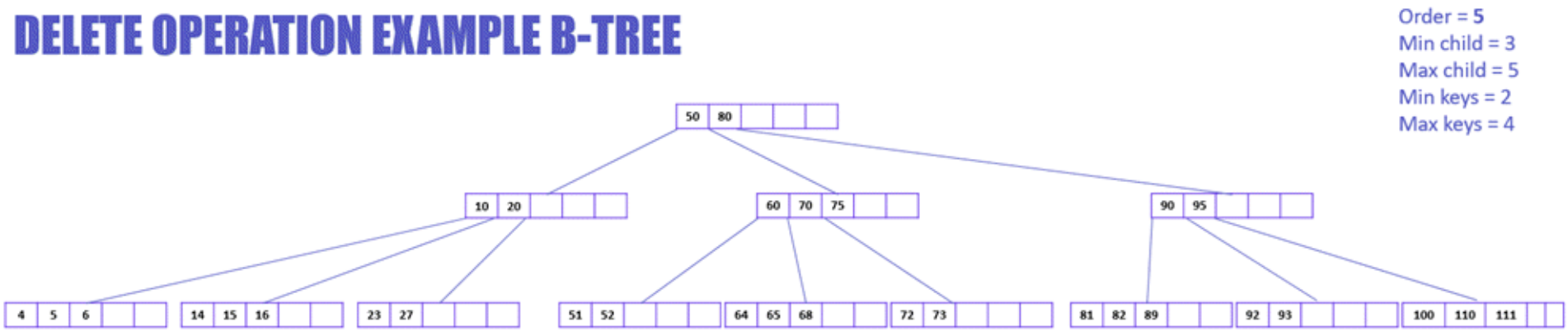
B-tree delete

CASE II: If the target key is in an internal node

Either choose, in- order predecessor or in-order successor

In case the of in-order predecessor, the maximum key from its left subtree will be selected. If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the **max of the in-order predecessor**

In case of in-order successor, the minimum key from its right subtree will be selected. If the target key's in-order predecessor does not have more than min keys, look for in-order successor's **minimum** key.



Delete 20
Delete 70

B-tree delete

CASE III: If the target key is in a root node

Replace with the maximum element of the in-order predecessor subtree

If, after deletion, the target has less than min keys, then the target node will borrow max value from its sibling via the sibling's parent.

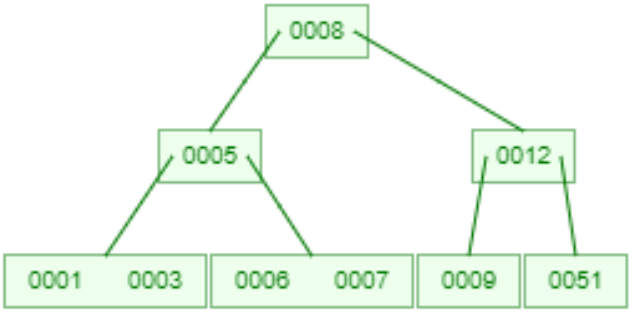
The max value of the parent will be taken by a target, but with the nodes of the max value of the sibling.

B-tree

- Delete 9

51, 8, 5, 1, 7, 3, 12, 9, 6

- 9



CASE I-c)
Target is in the leaf node, but no siblings have more than min number of keys

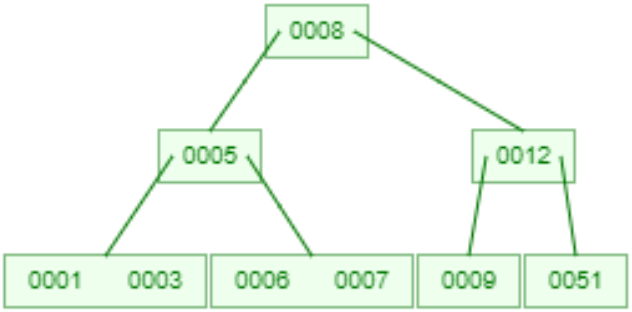
Merge with siblings and the minimum of parent nodes
Total keys will be now more than min
The target key will be replaced with the minimum of a parent node

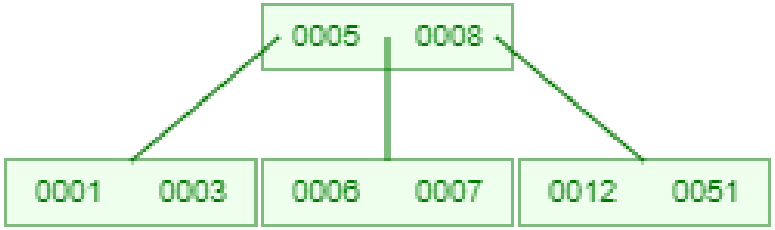
B-tree

- Delete 9

51, 8, 5, 1, 7, 3, 12, 9, 6

- 9





Dynamic Multilevel Indexes Using B, B+ Trees

- https://yangez.github.io/btree-js/
- https://www.cs.usfca.edu/~galles/visualization/BTree.html
- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html