B-TREES

CSC212: Data Structures

B-Trees: Why?

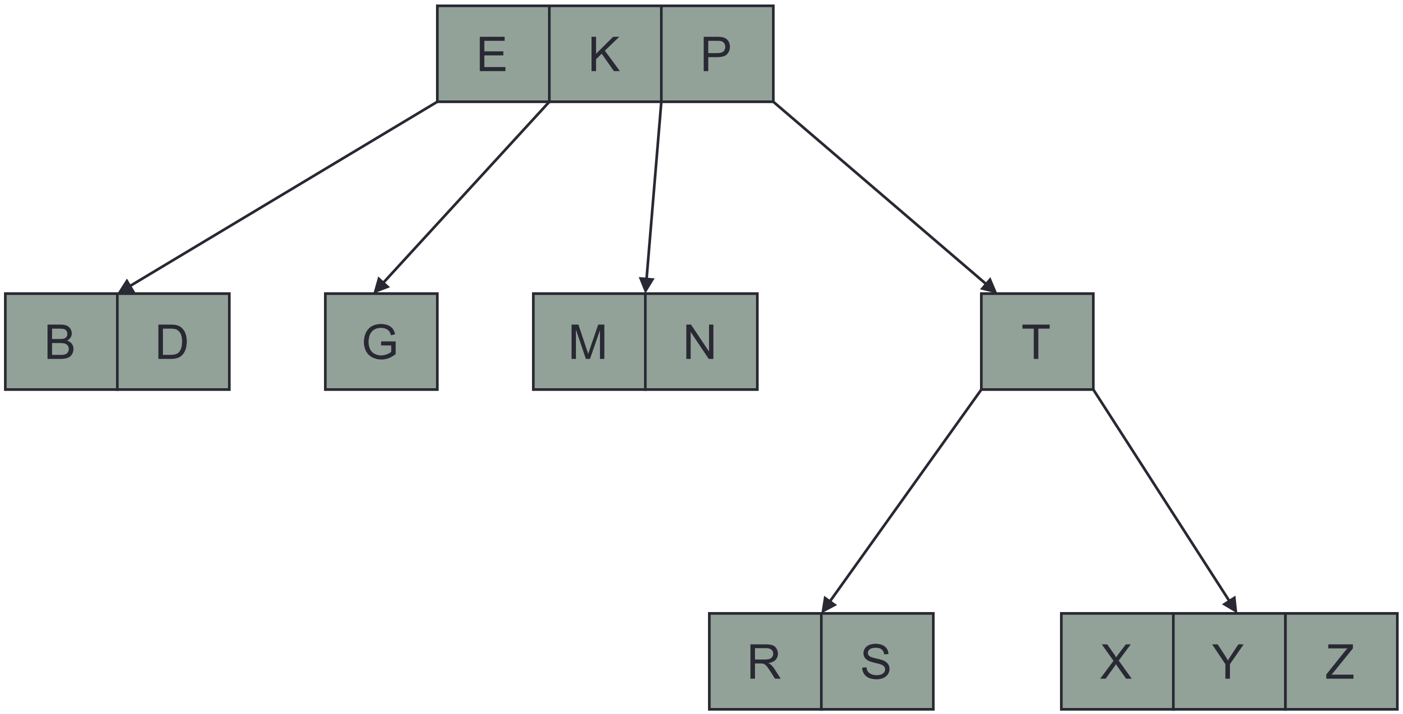
- Best tree discussed so far – AVL Tree:
  - Important operation *Findkey*() can be implemented in **O(logn)** time.
- AVL Tree has problems for large data
  - The size of the AVL tree increases and may not fit in the system's main memory.
  - The height of the AVL tree also increases – Findkey() operation no more efficient.

B-Trees: Why?

- To overcome these problems, m-way trees have been created.
- M-way tree allows:
  - Each node to have at the most m children (or sub-trees)
  - Each non-leaf node has (*k-1*) keys if it has *k* children.
  - M-way tree is ordered and could be balanced like an AVL tree

M-Way Tree

M-way tree of order 4



B-Trees: Why?

- Because in a m-way tree, a node can have more than two children and more than one data element in it, the overall size (i.e. number of nodes) decreases → height decreases.
- Also, at any time only a part of the tree can be loaded into the main memory → the rest of the tree can remain in disk storage.
- B-trees are a kind of m-way trees.
- Special types of B-trees:
  - B+ Tree.
  - B* Tree.
- Database files are represented as B-trees.

B+ Tree: Properties

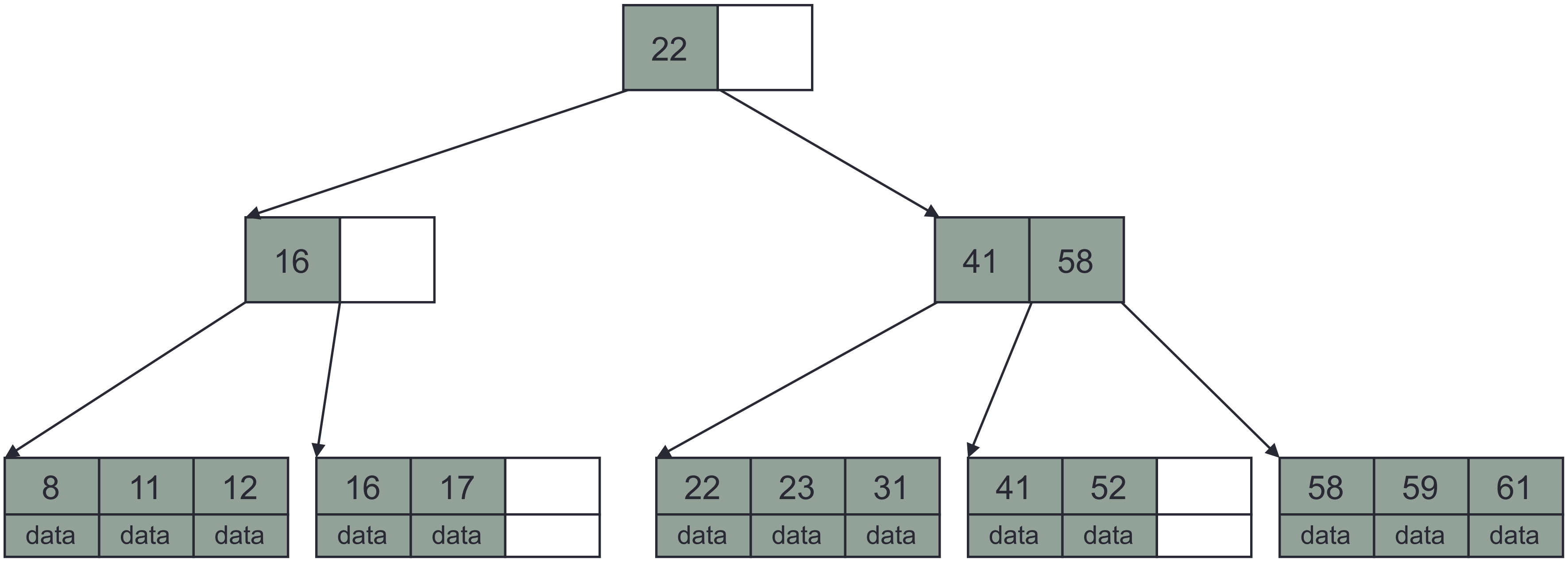
- B+ Tree of order **M** has following properties:
  1. Root is either a leaf or has 2 to M children.
  2. Non-leaf nodes (except the root)
     - have ⌈M/2⌉ to M children
     - → which means they can have from ⌈M/2⌉-1 to M-1 keys stored in them.
  3. Non-leaf store at the most M-1 keys to guide search; key i represents the smallest key in the subtree i + 1.
  4. All leaves are at the same depth or level
  5. Data elements are stored in the leaves only and have between ⌈M/2⌉ and M data elements.

B+ Tree: Properties

Notes:

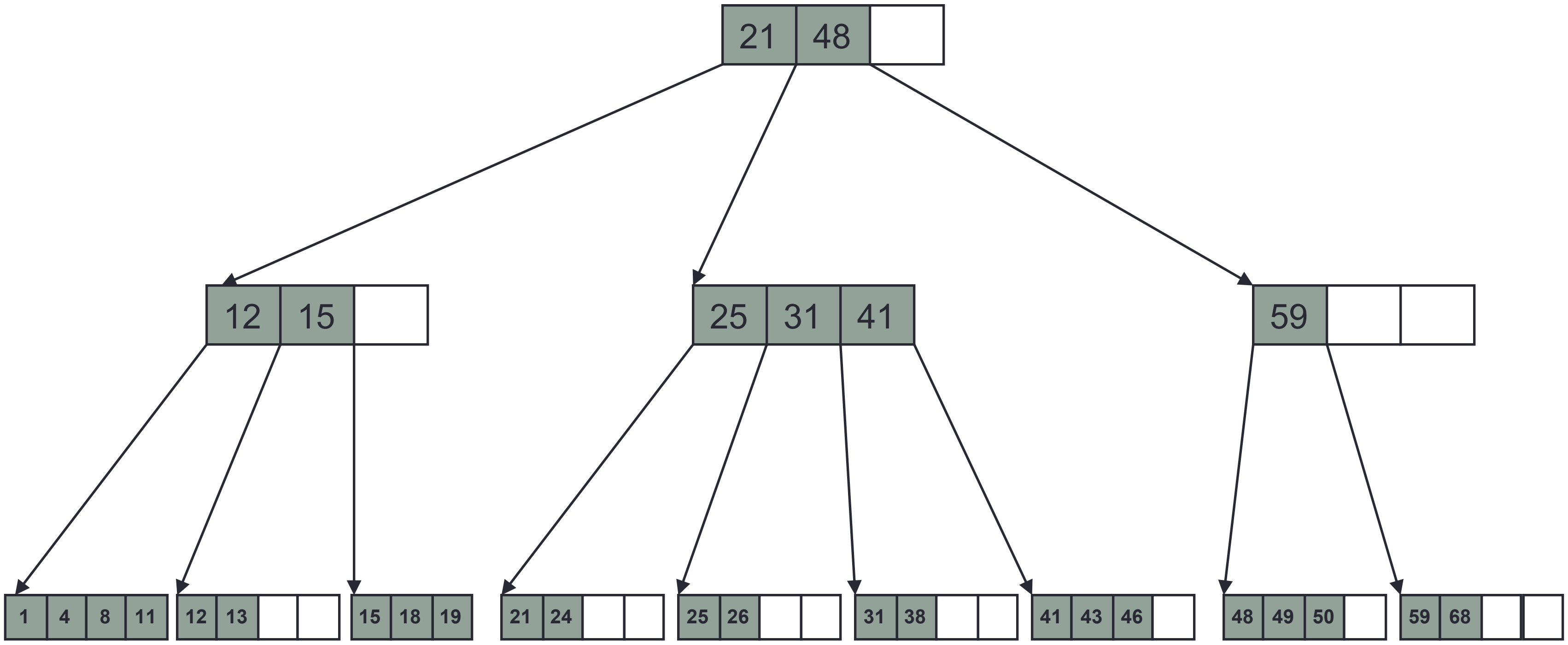
1. Actually leaf nodes can have up to L data elements. To simplify we assume L is equal to M.
2. Choice of parameters L and M depends on the data being stored in the B+ Tree.

B+ Tree: Example 1



B+ tree of order 3 (M=3)

B+ Tree: Example 2



B+ tree of order 4 (M=4)

B+ Tree: Search

- How is FindKey operation performed in a B+ Tree?
- Almost as in a BST
- The keys in the non-leaf node are used for guidance.
- The data element is always in the leaves.
- Search path gets traced from the root to the leave, where data element is found or not found.

B+ Tree: Insert

1. Search for a leaf node N into which new data element D will be inserted.
2. Insert D in N in sorted order.
   - If N has space for D, insert is complete.
   - Otherwise, if there is no space in N “overflow” takes place. Overflow is dealt with by:
     - Transferring a datum (or a subtree) to one of the close sibling nodes.
     - Or, by splitting N, which may lead to other splits

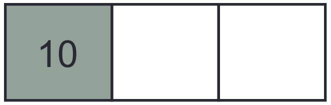
B+ Tree: Insert

Insert 10

B+ tree of order 3 (M=3)

B+ Tree: Insert

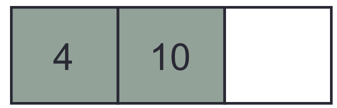
Insert 4



B+ tree of order 3 (M=3)

B+ Tree: Insert

Insert 90



B+ tree of order 3 (M=3)

B+ Tree: Insert

Insert 8

4	10	90

B+ tree of order 3 (M=3)

B+ Tree: Insert

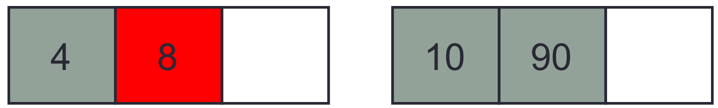
Insert 8 (Overflow)

4	8	10	90

B+ tree of order 3 (M=3)

B+ Tree: Insert

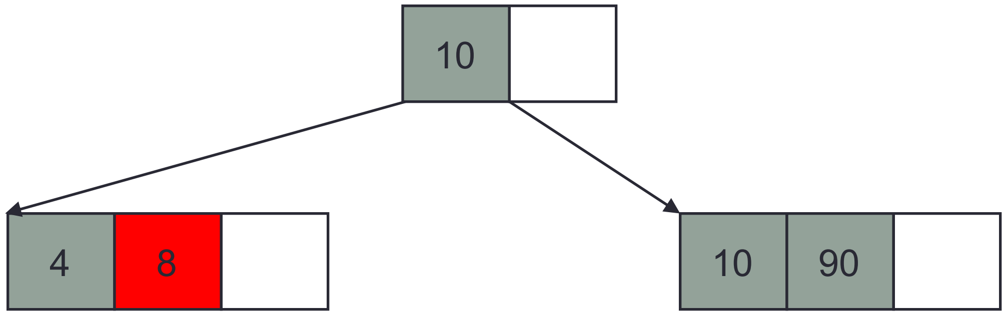
Insert 8 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert

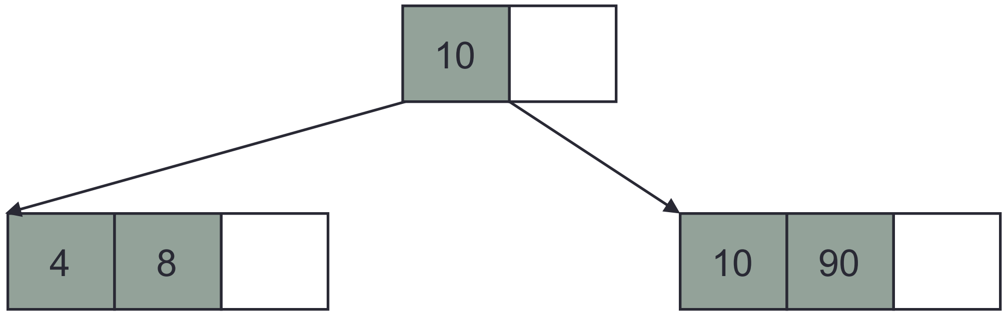
Insert 8 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert

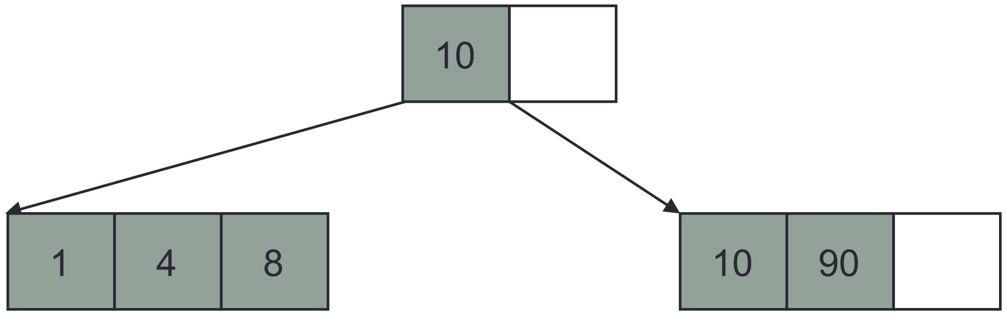
Insert 1



B+ tree of order 3 (M=3)

B+ Tree: Insert

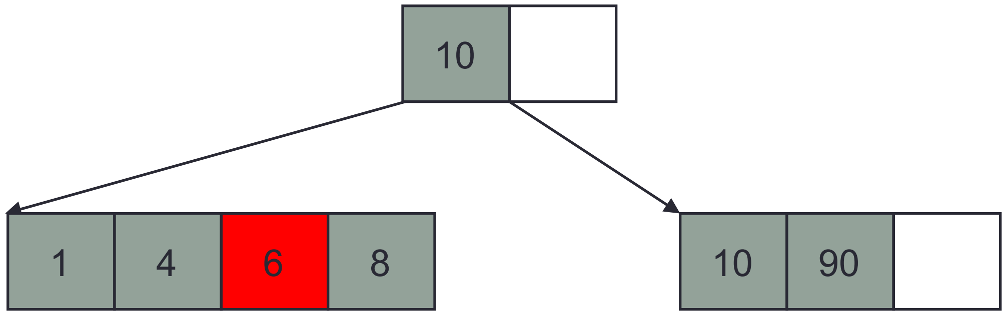
Insert 6



B+ tree of order 3 (M=3)

B+ Tree: Insert

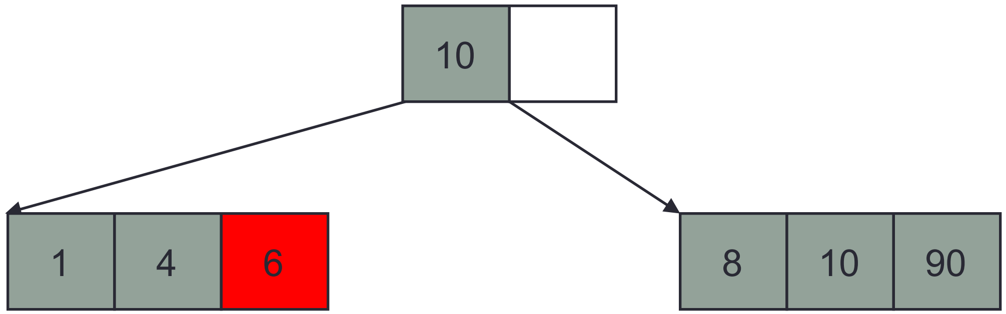
Insert 6 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

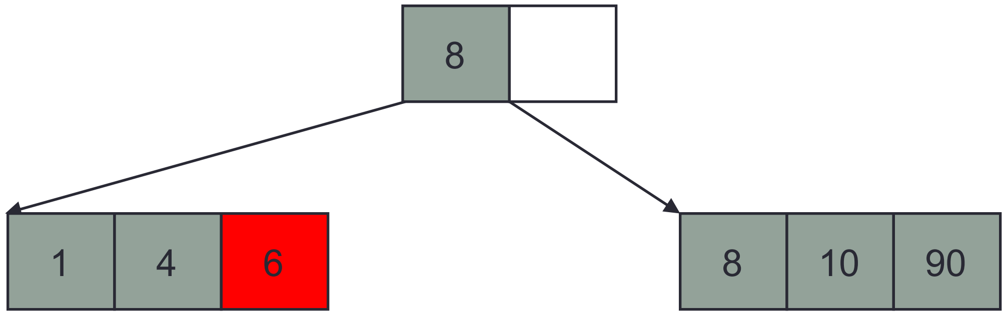
Insert 6 (Transfer)



B+ tree of order 3 (M=3)

B+ Tree: Insert

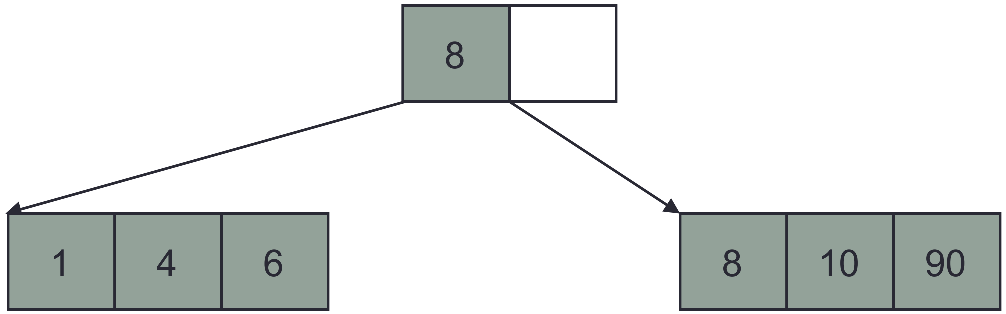
Insert 6 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert

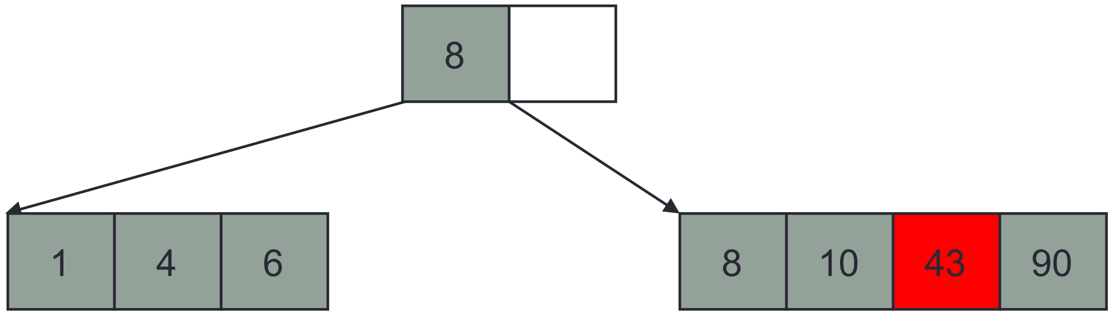
Insert 43



B+ tree of order 3 (M=3)

B+ Tree: Insert

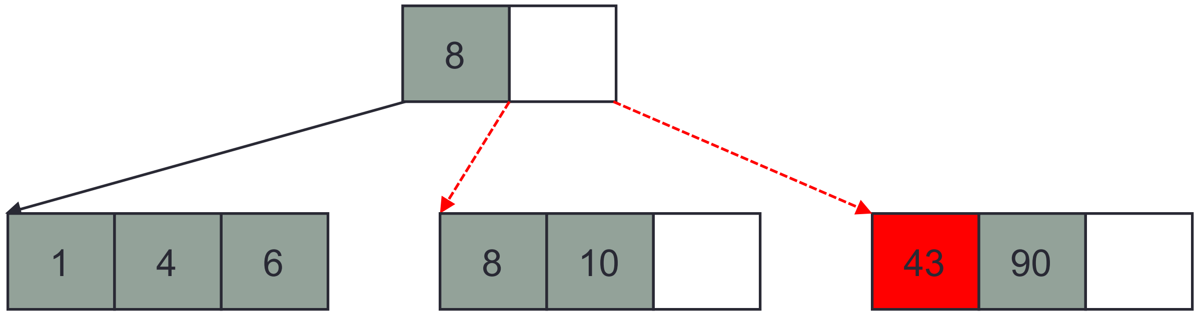
Insert 43 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

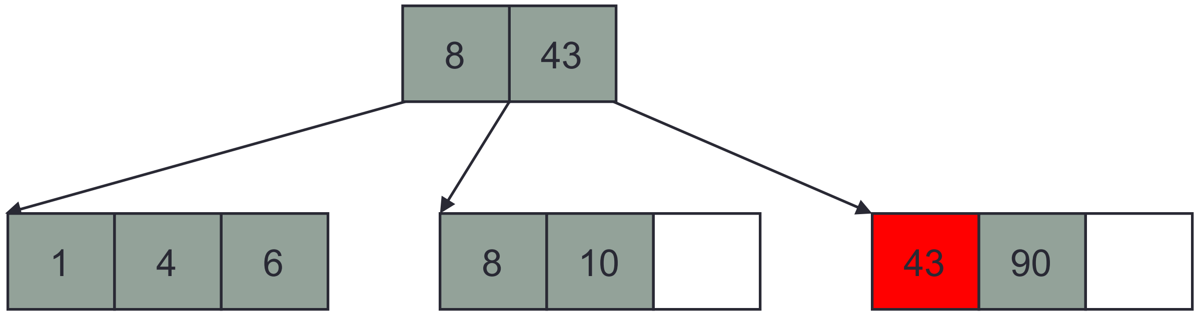
Insert 43 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert

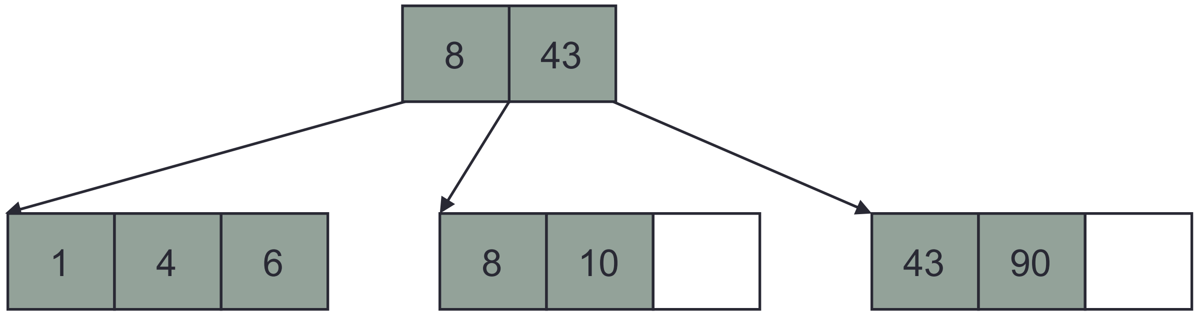
Insert 43 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert

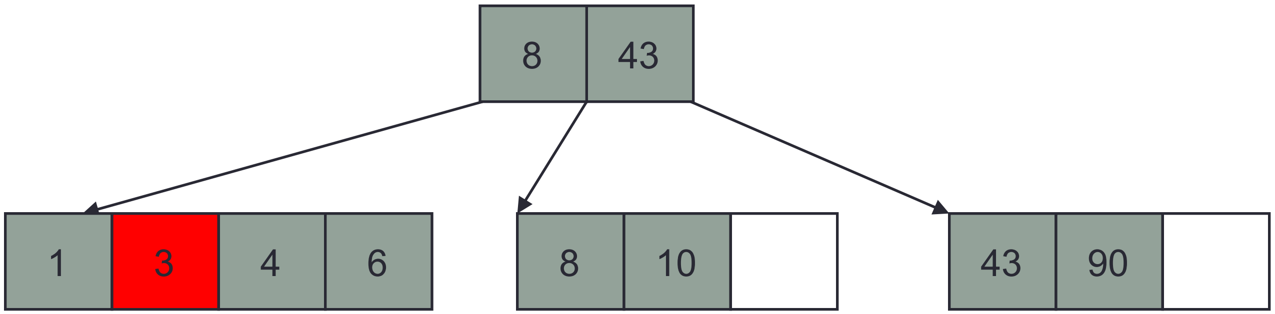
Insert 3



B+ tree of order 3 (M=3)

B+ Tree: Insert

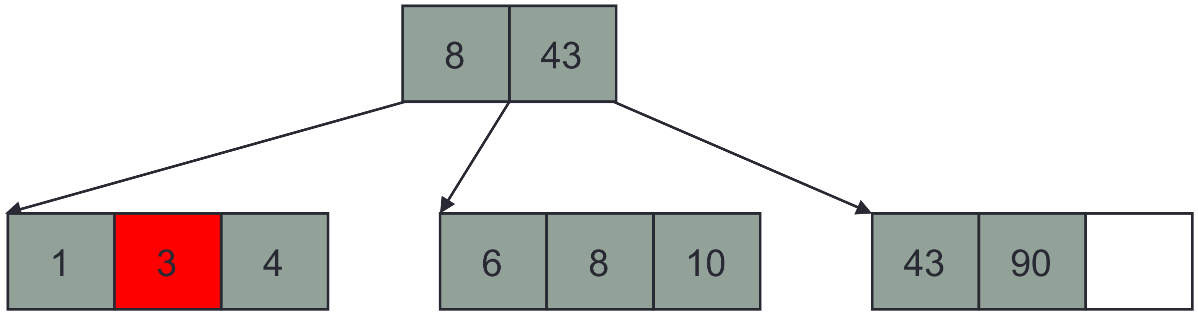
Insert 3 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

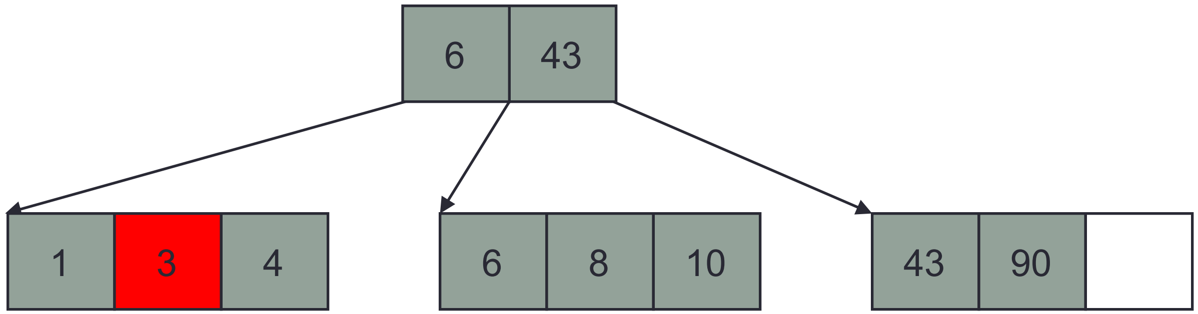
Insert 3 (Transfer)



B+ tree of order 3 (M=3)

B+ Tree: Insert

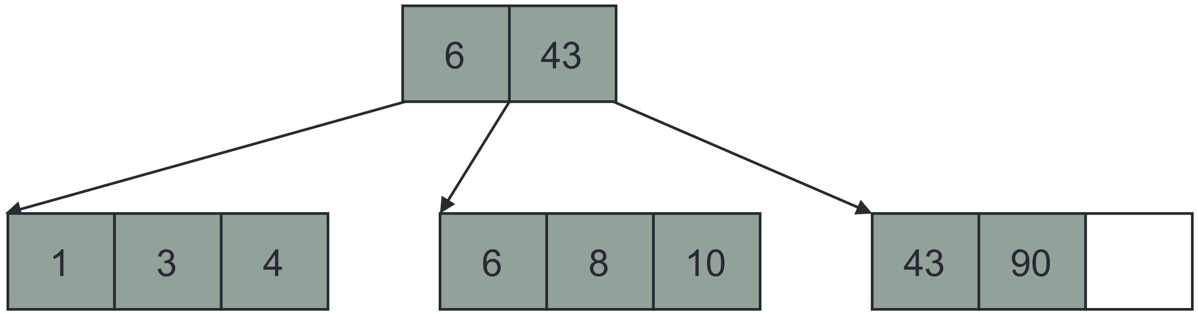
Insert 3 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert

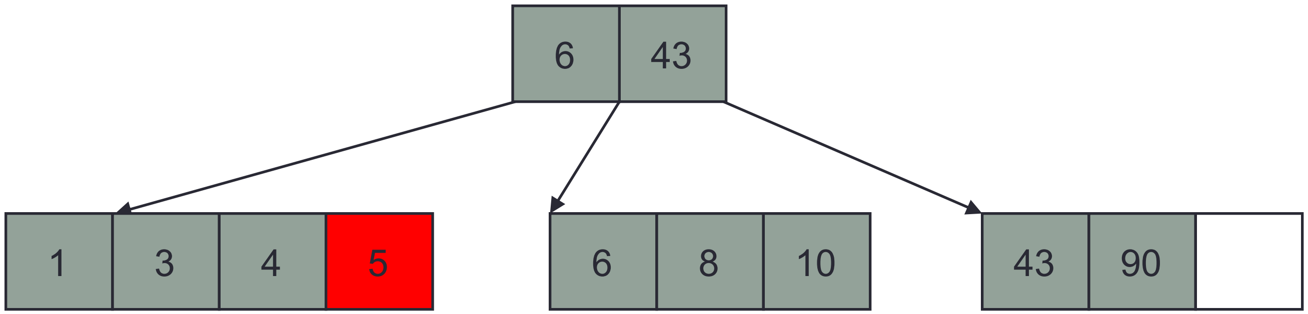
Insert 5



B+ tree of order 3 (M=3)

B+ Tree: Insert

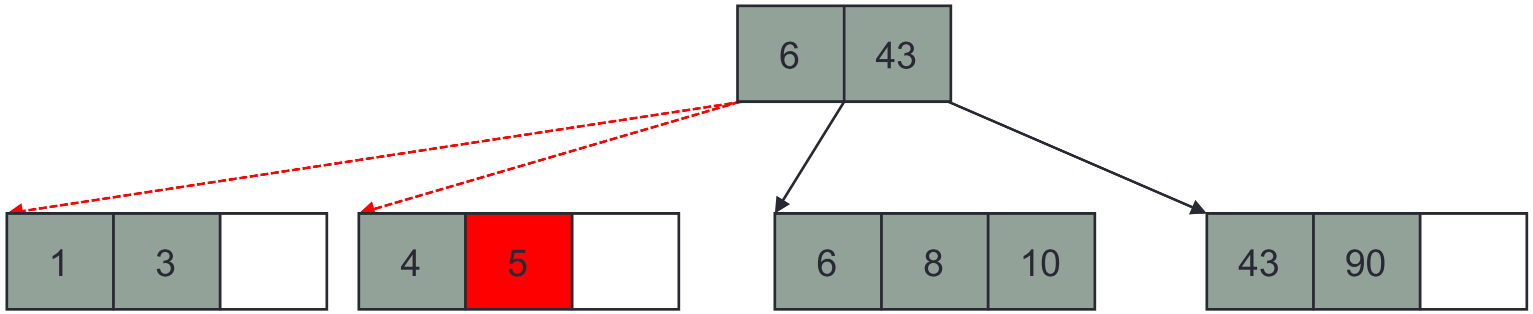
Insert 5 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

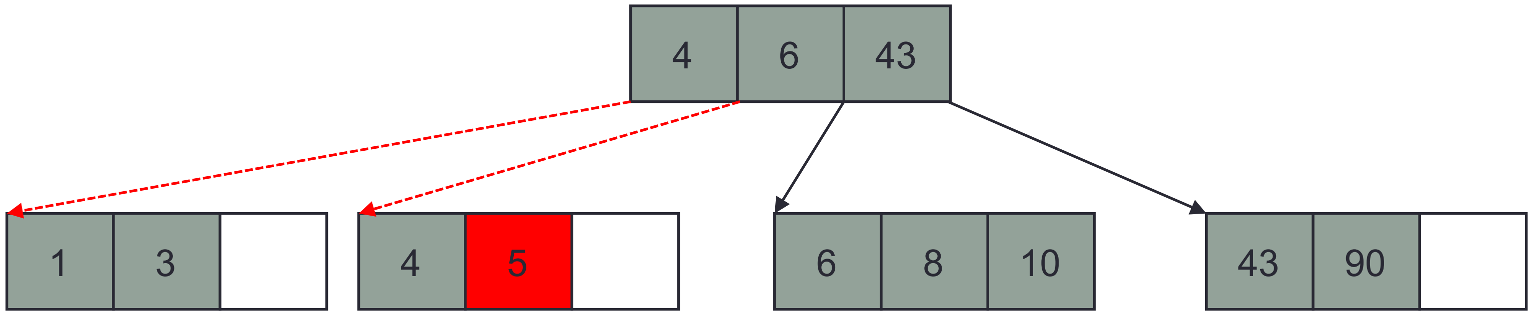
Insert 5 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert

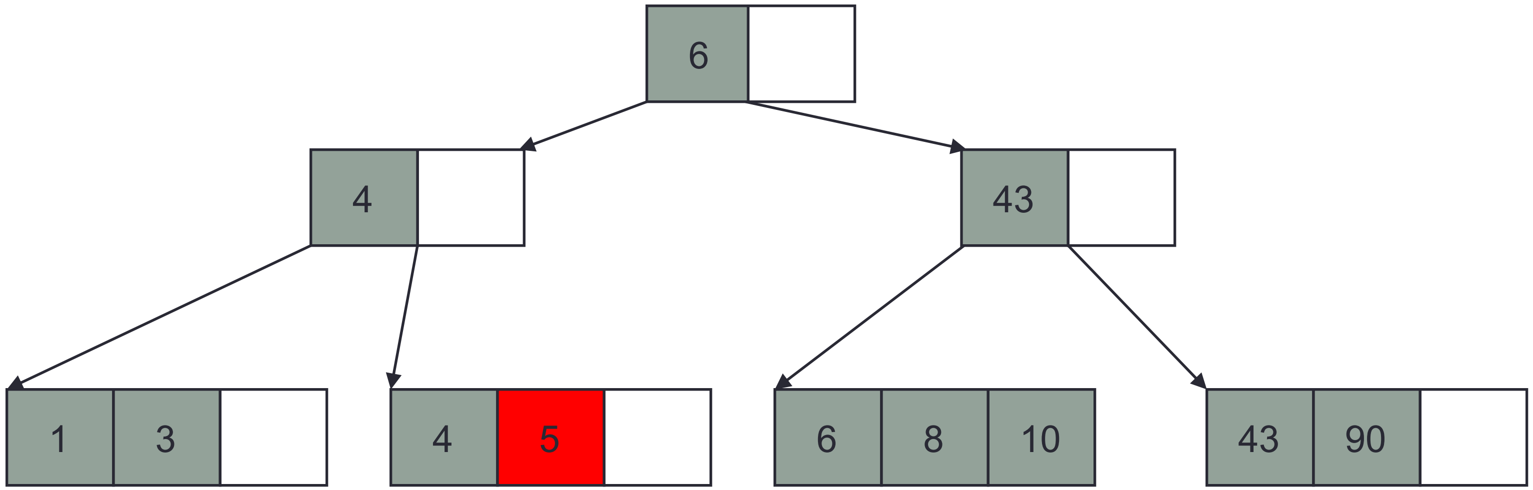
Insert 5 (Update - Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

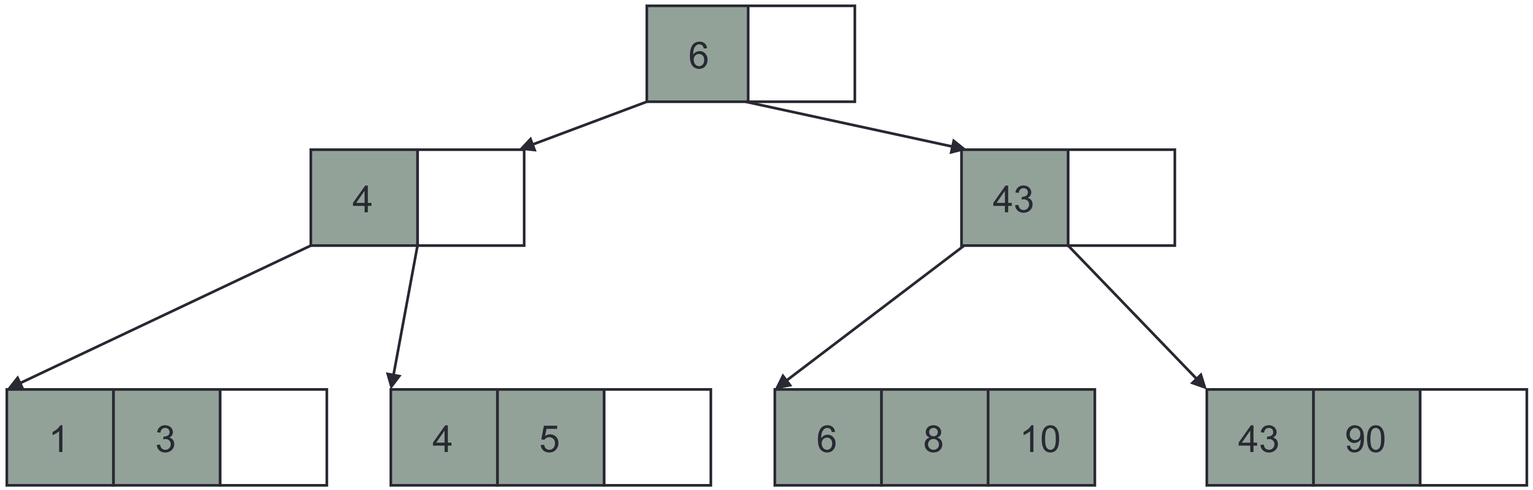
Insert 5 (Update – Overflow - Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert

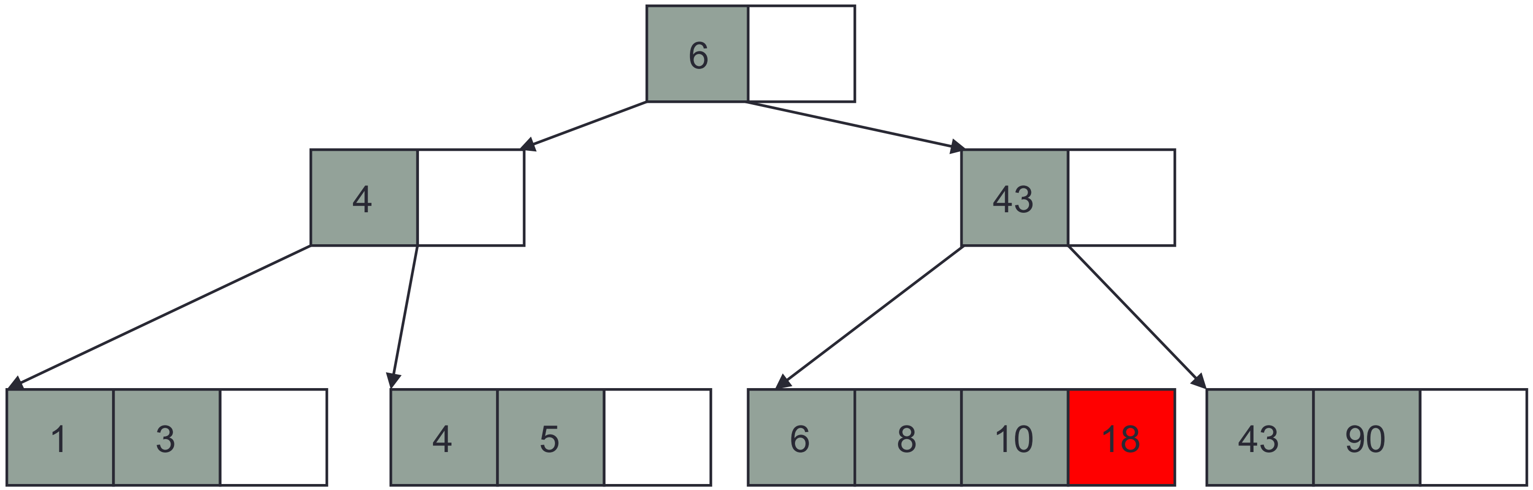
Insert 18



B+ tree of order 3 (M=3)

B+ Tree: Insert

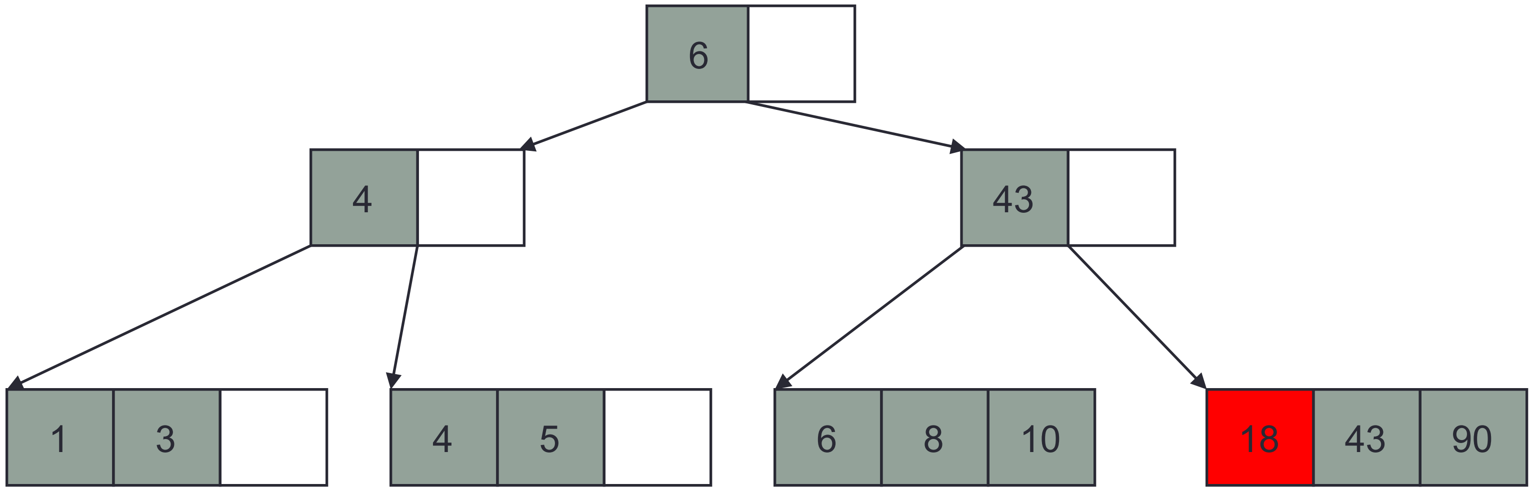
Insert 18 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

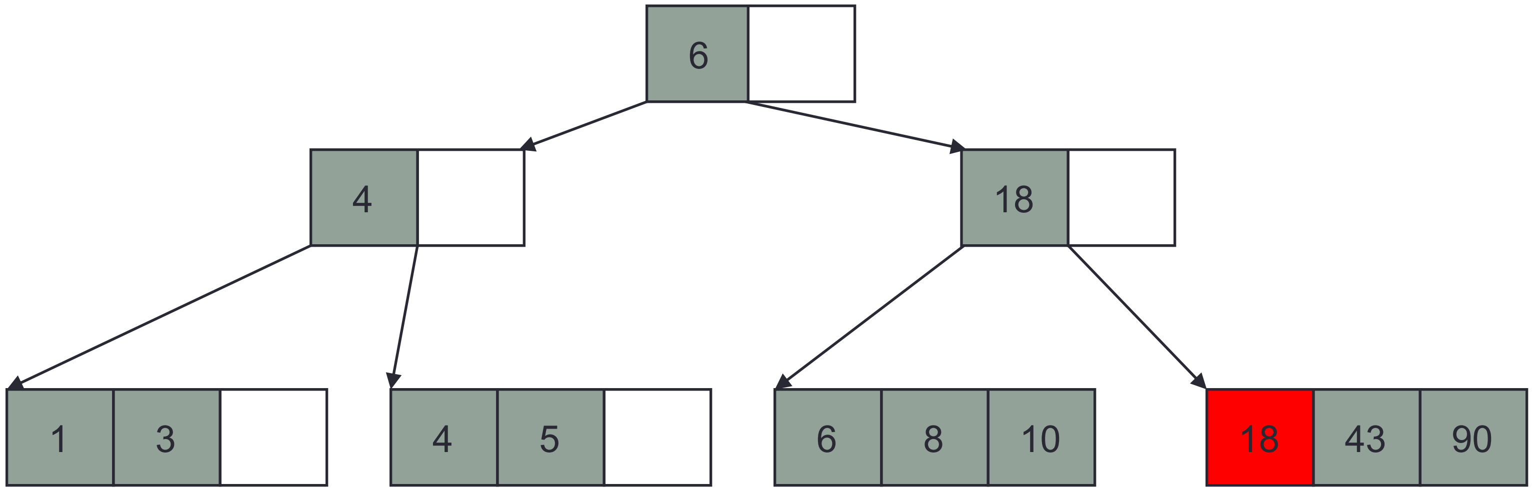
Insert 18 (Transfer)



B+ tree of order 3 (M=3)

B+ Tree: Insert

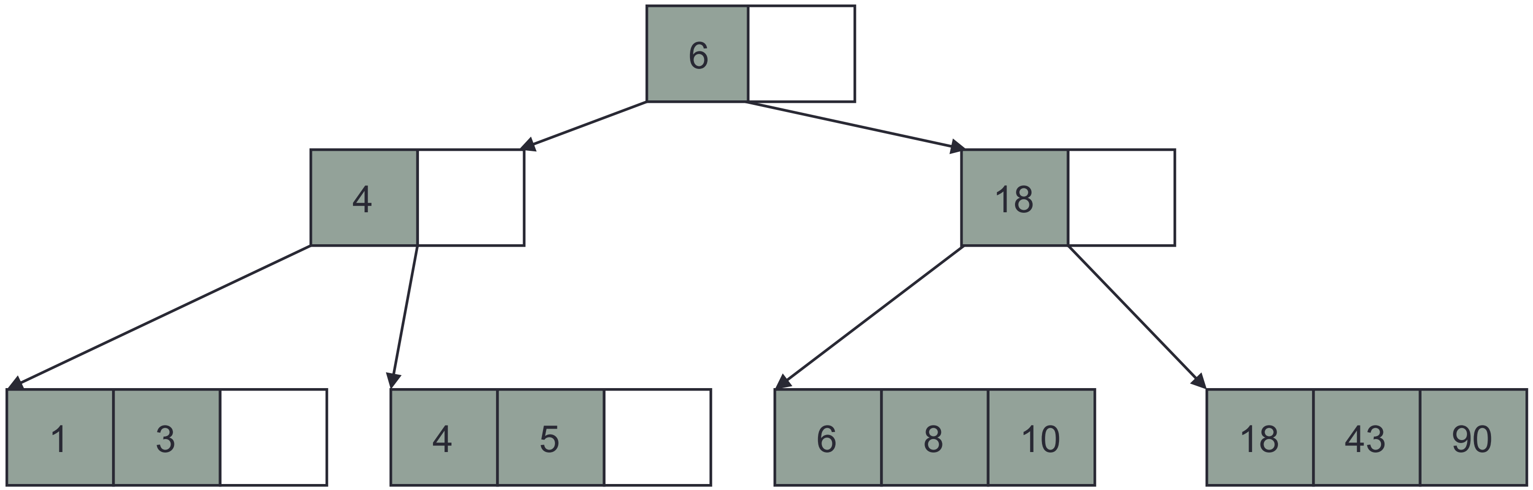
Insert 18 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert

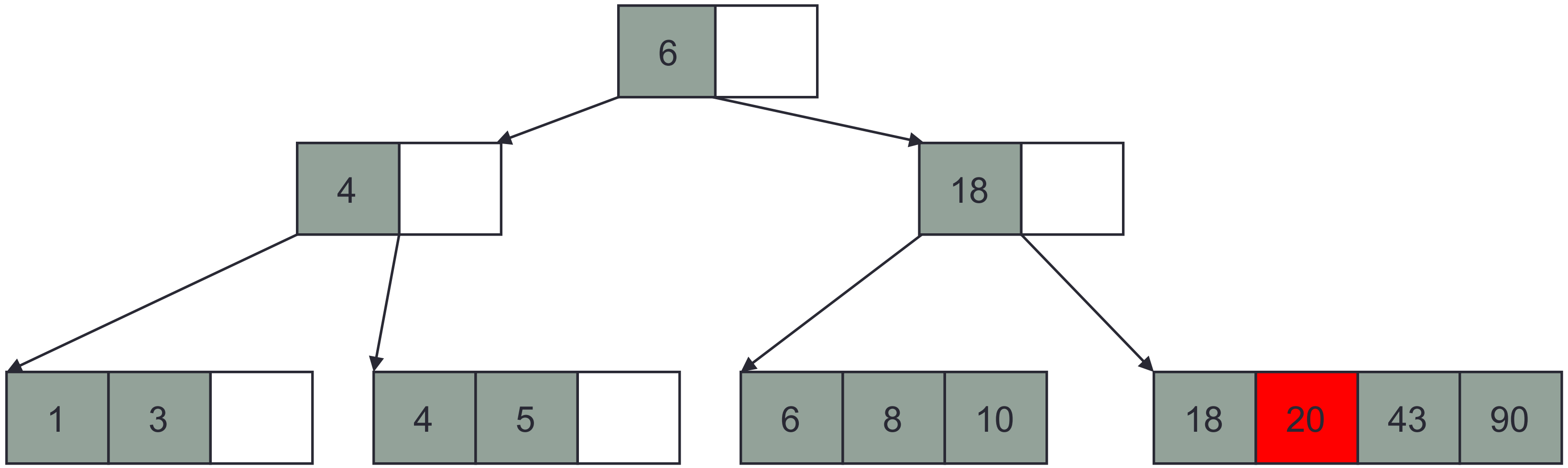
Insert 20



B+ tree of order 3 (M=3)

B+ Tree: Insert

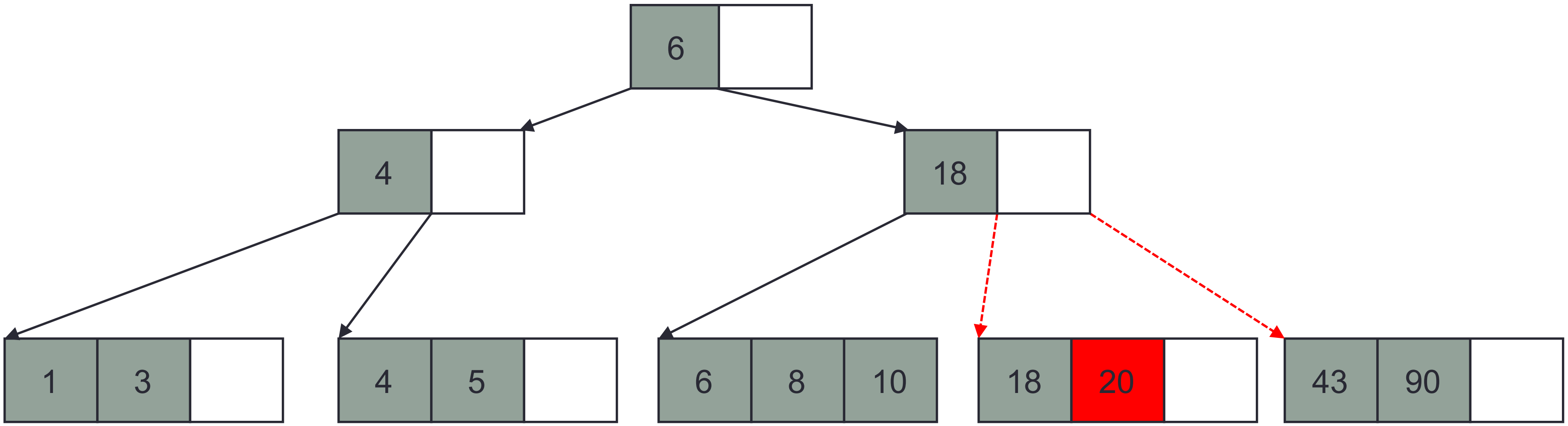
Insert 20 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert

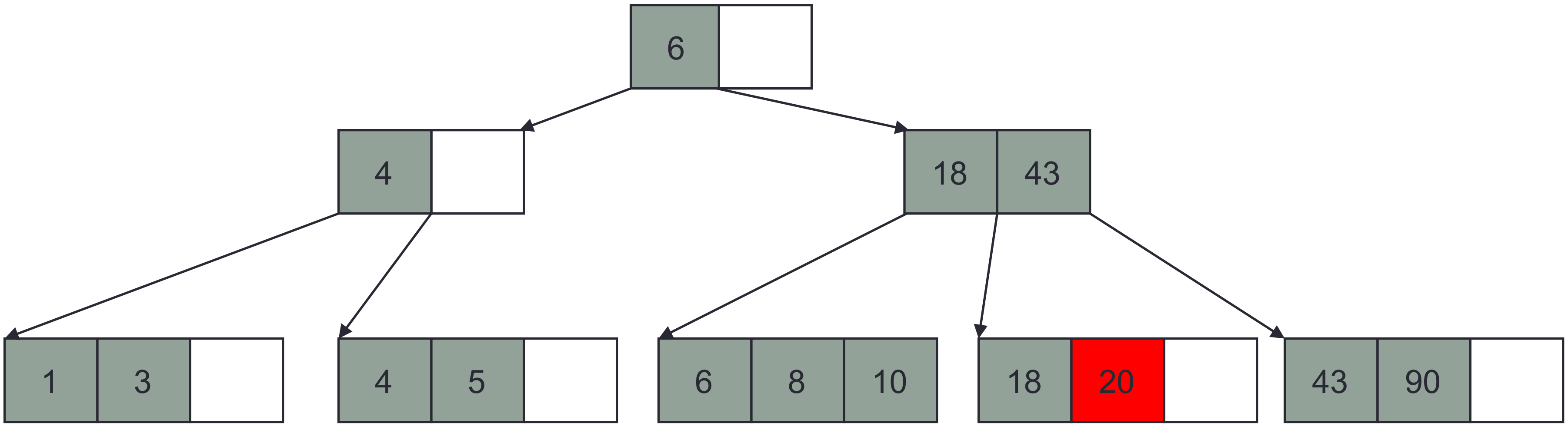
Insert 20 (Split)



B+ tree of order 3 (M=3)

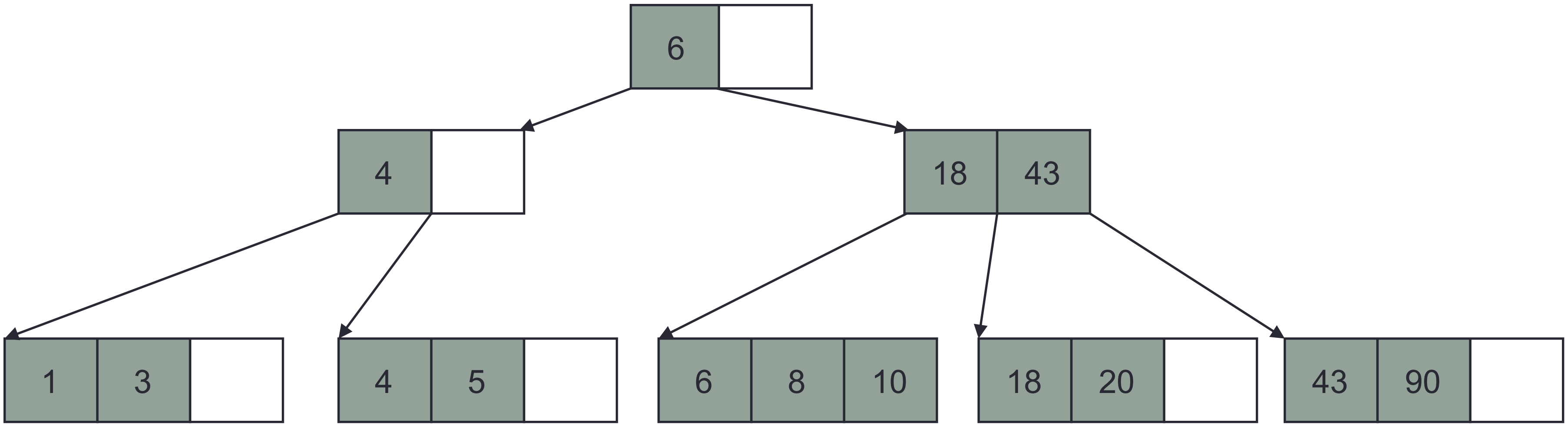
B+ Tree: Insert

Insert 20 (Update)



B+ tree of order 3 (M=3)

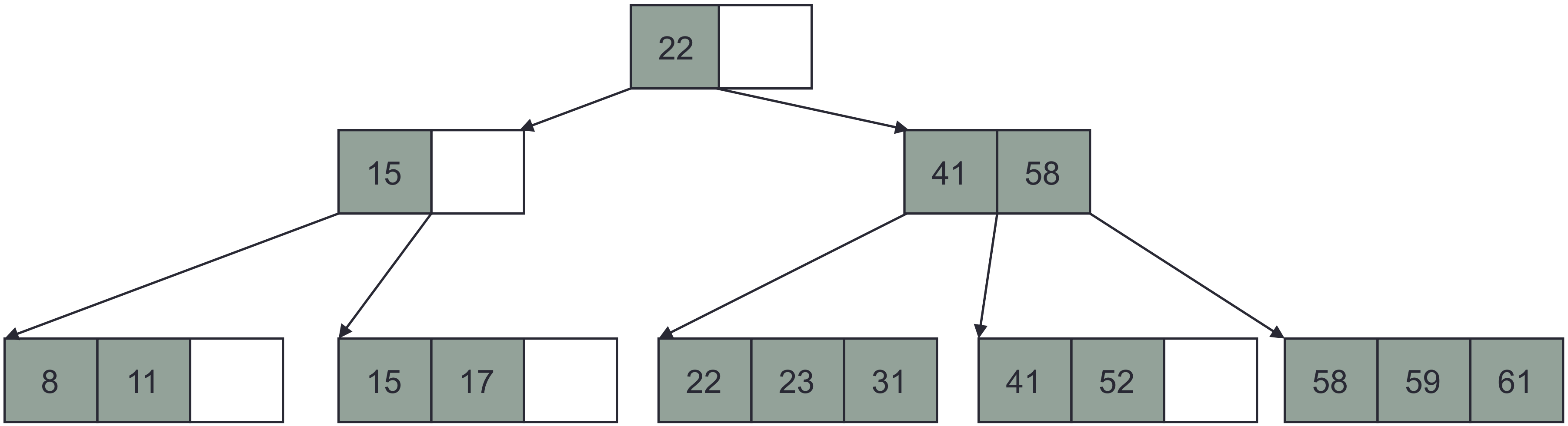
B+ Tree: Insert



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

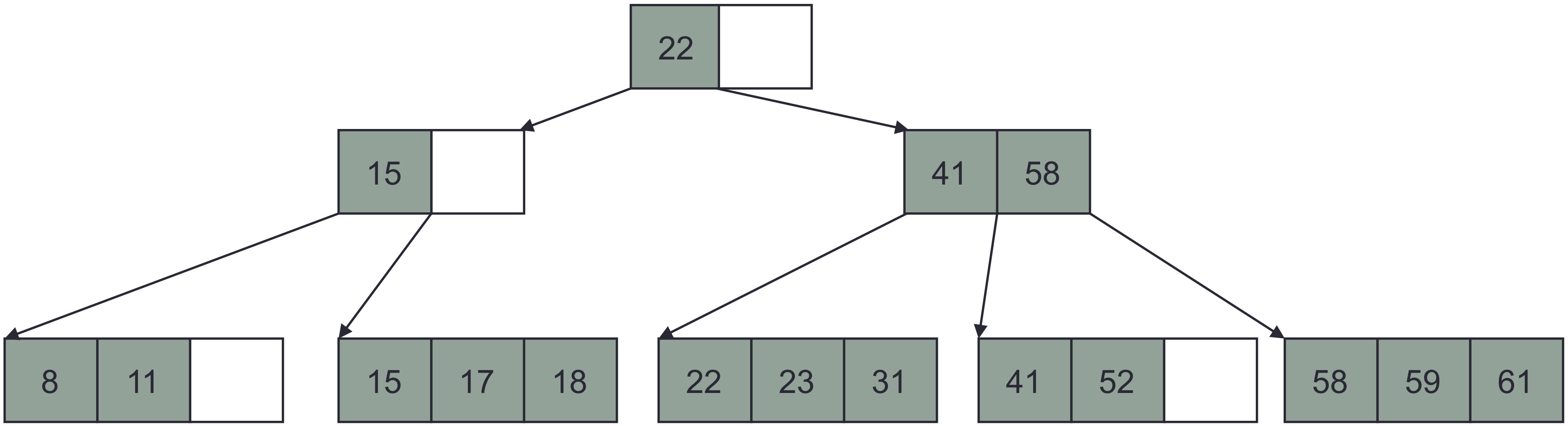
Insert 18



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

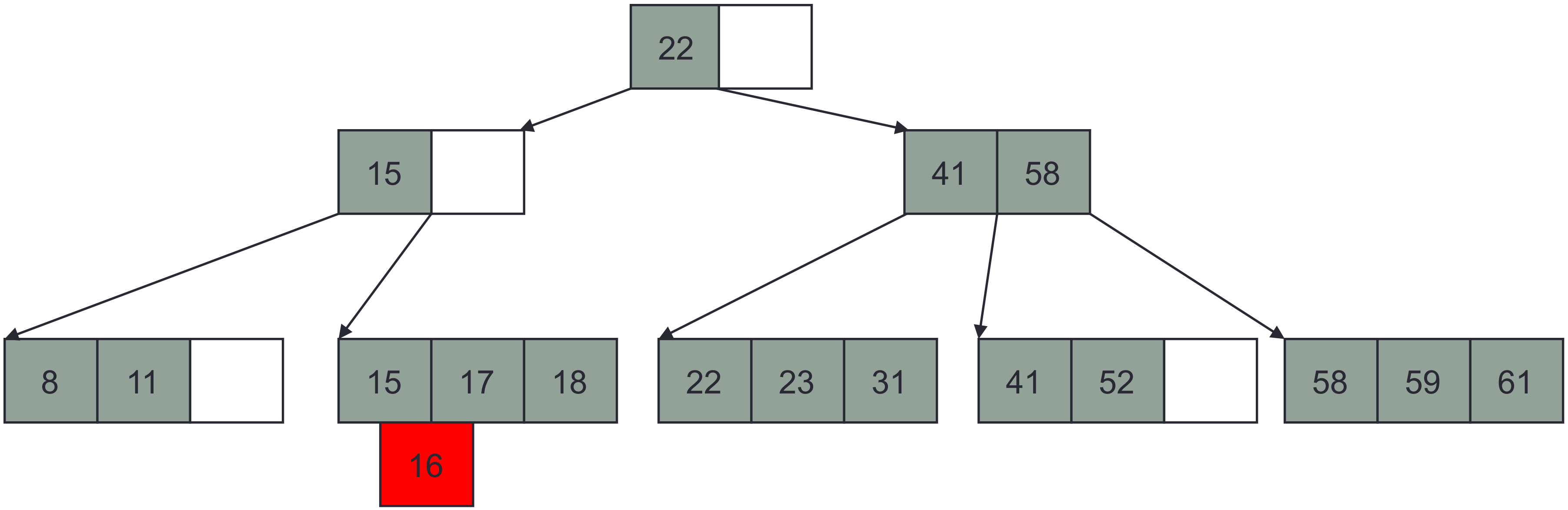
Insert 16



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

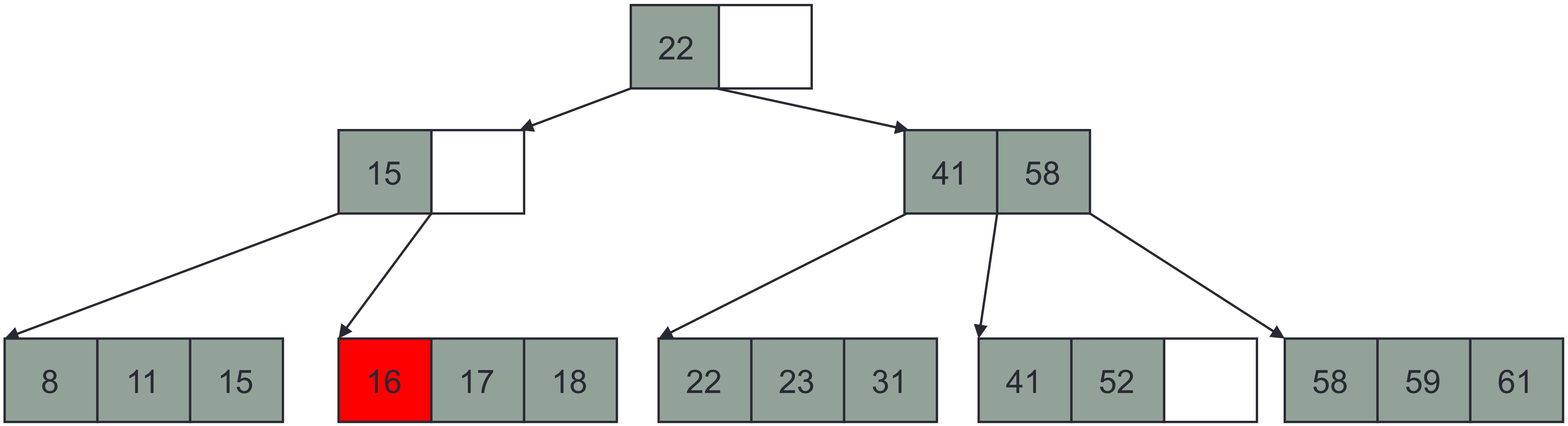
Insert 16 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

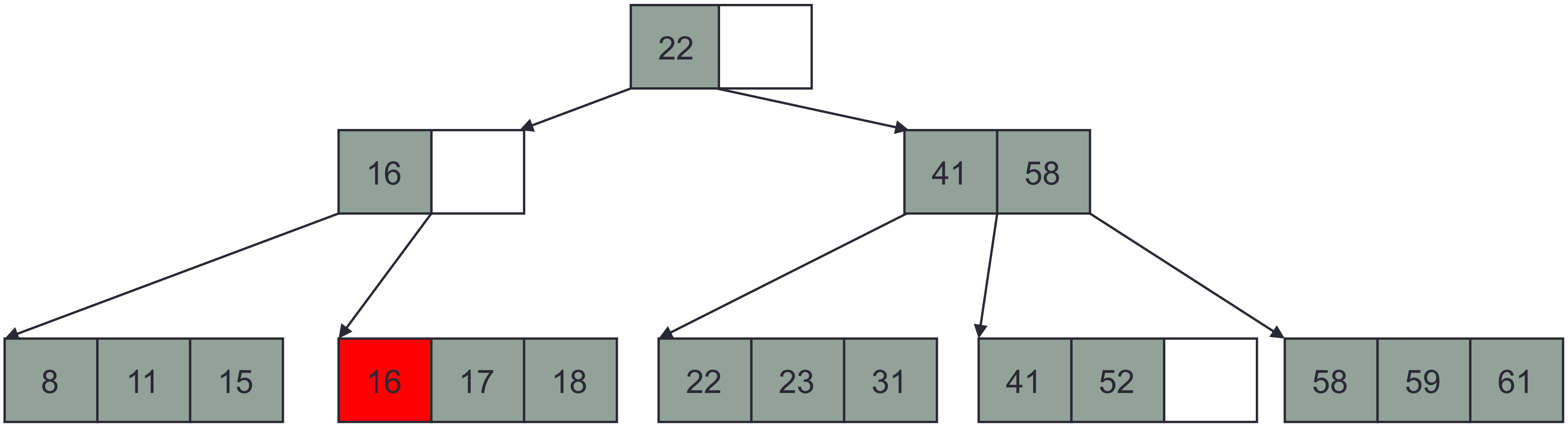
Insert 16 (Transfer)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

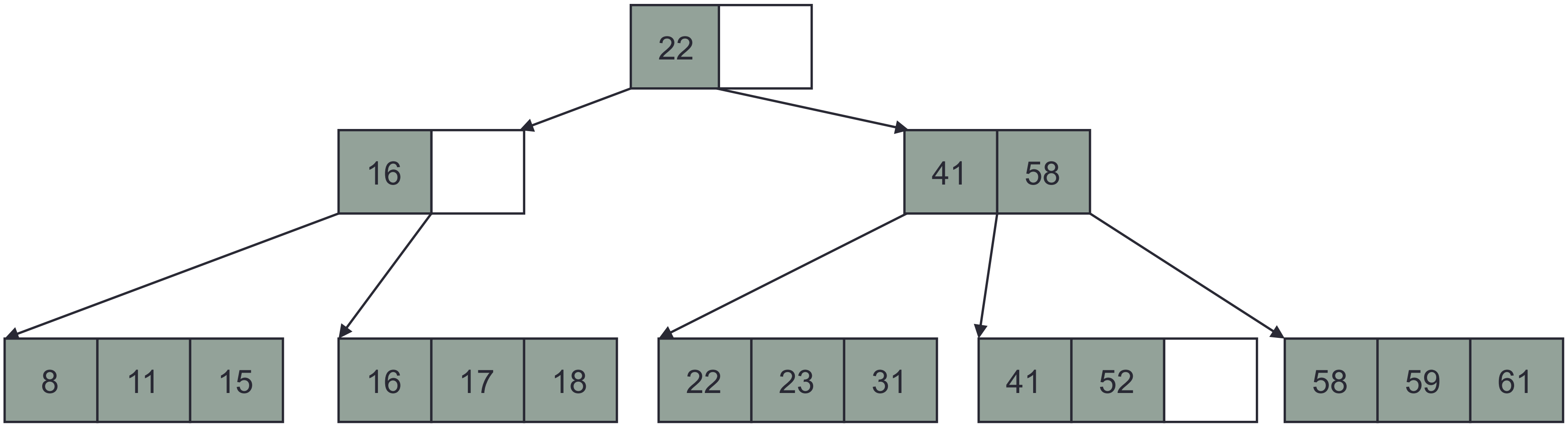
Insert 16 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

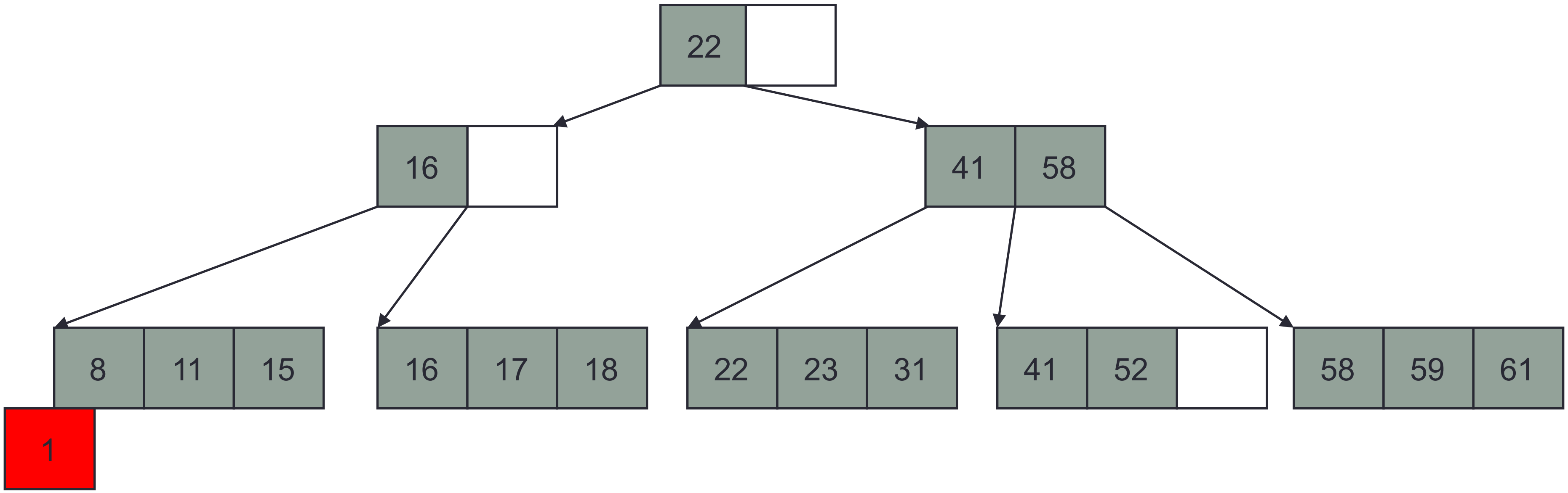
Insert 1



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

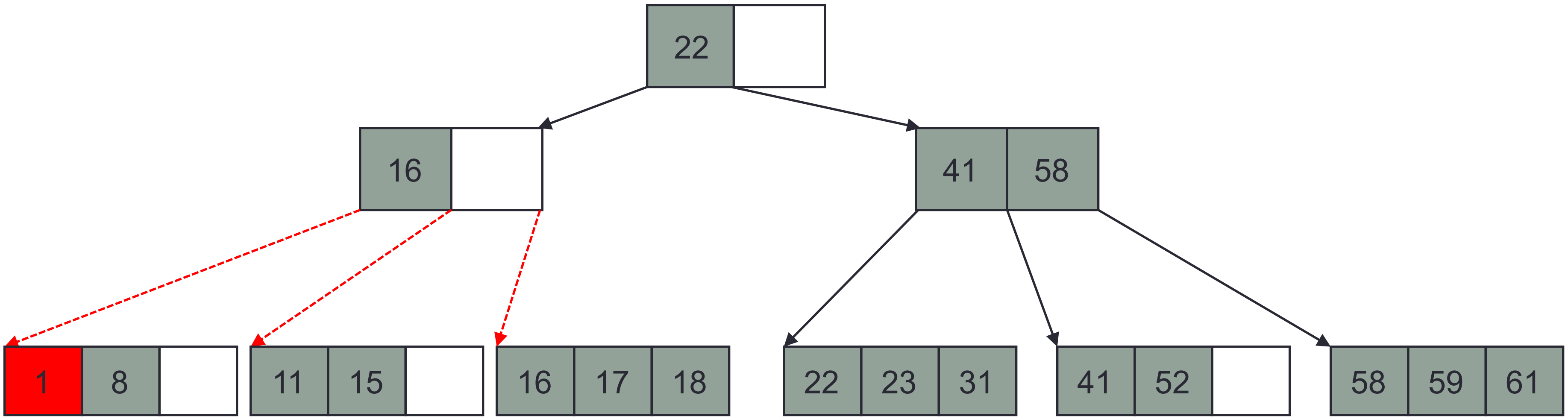
Insert 1 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

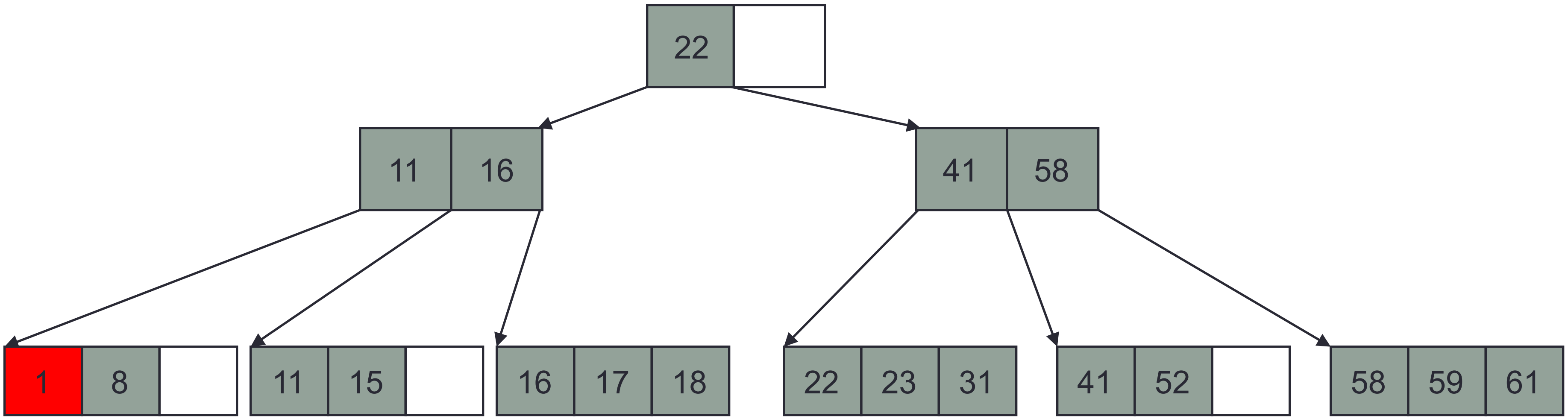
Insert 1 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

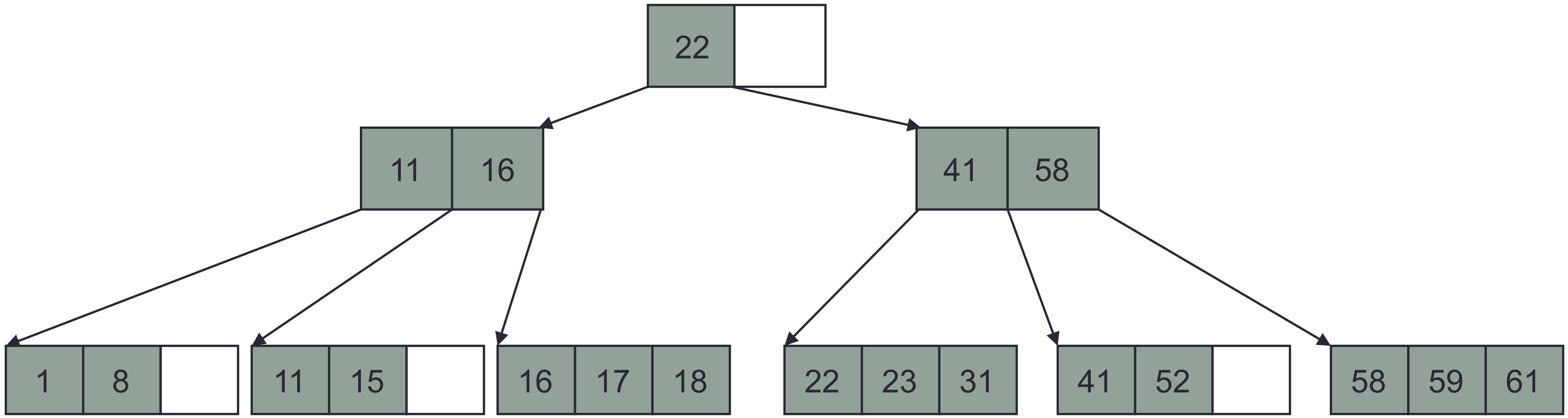
Insert 1 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

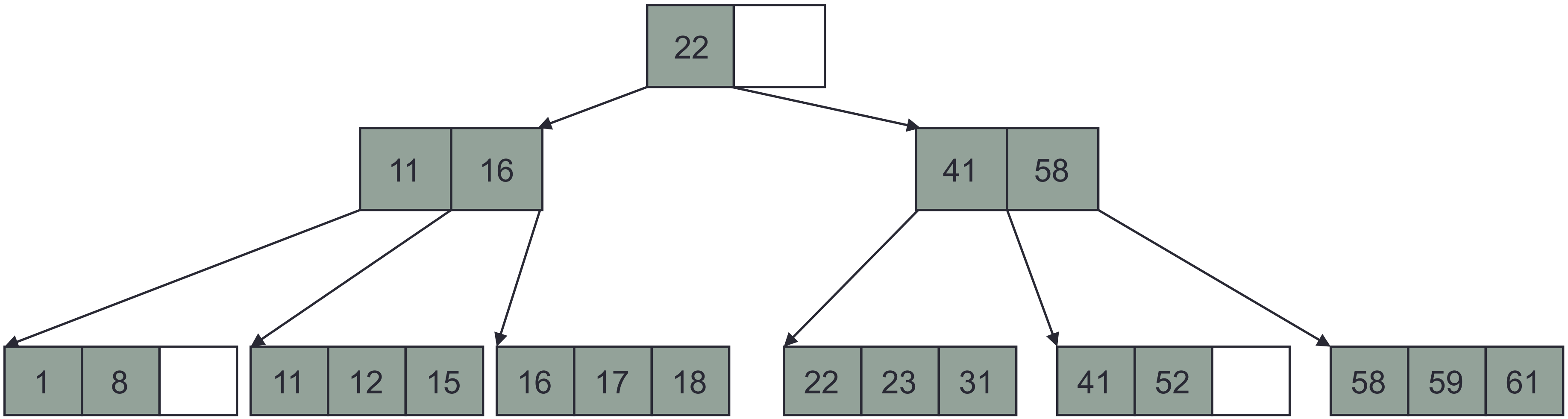
Insert 12



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

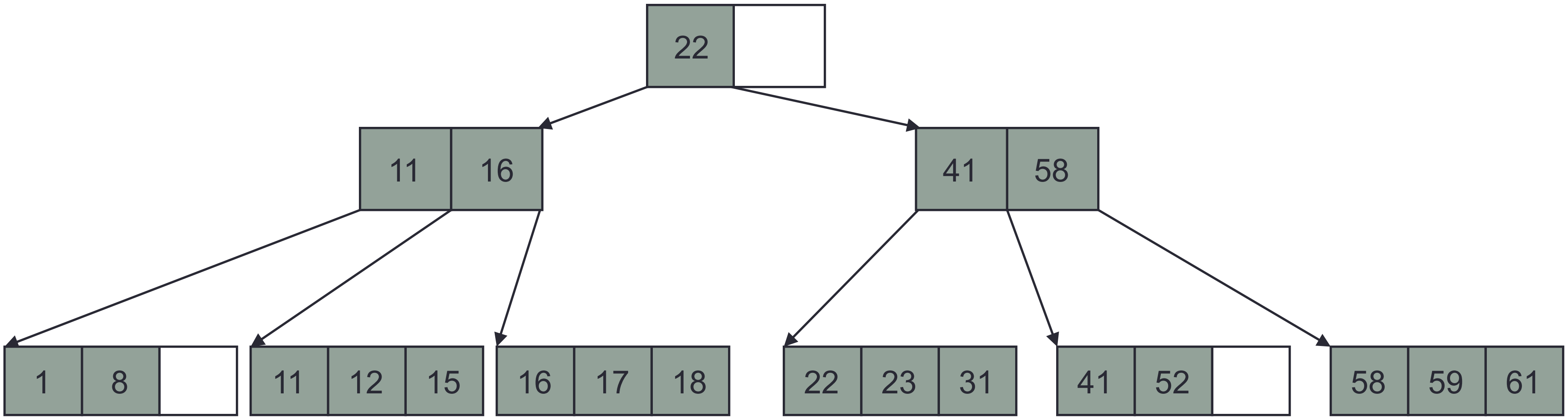
Insert 12



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

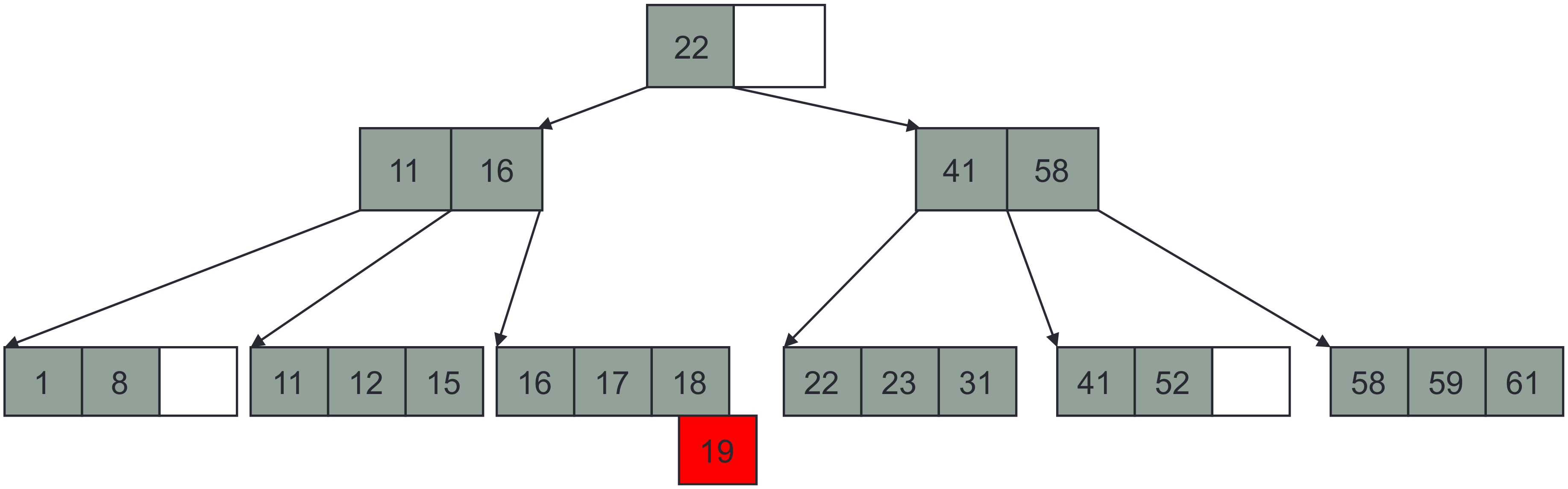
Insert 19



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

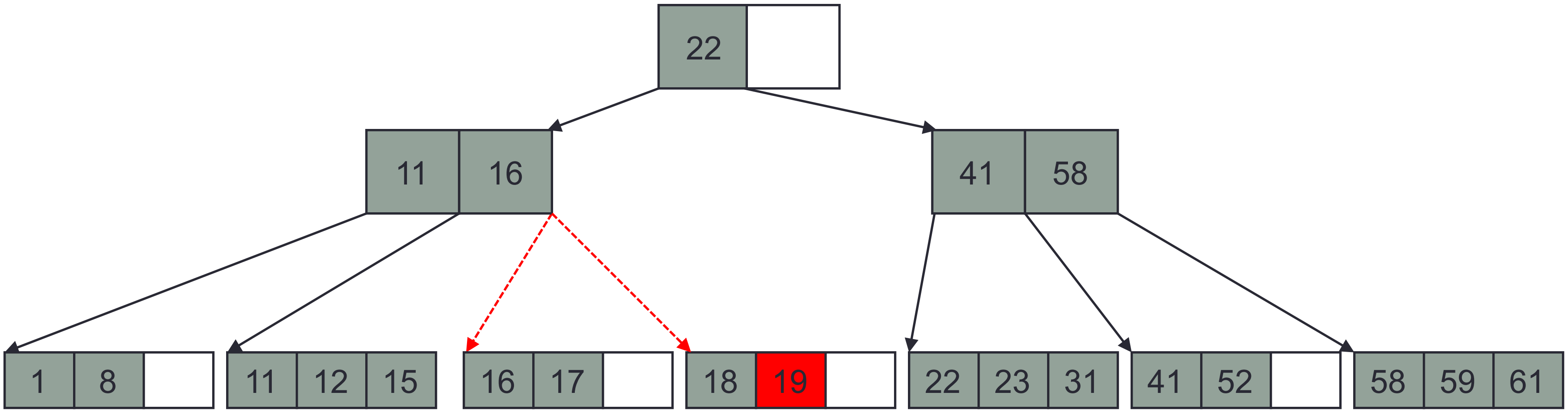
Insert 19 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

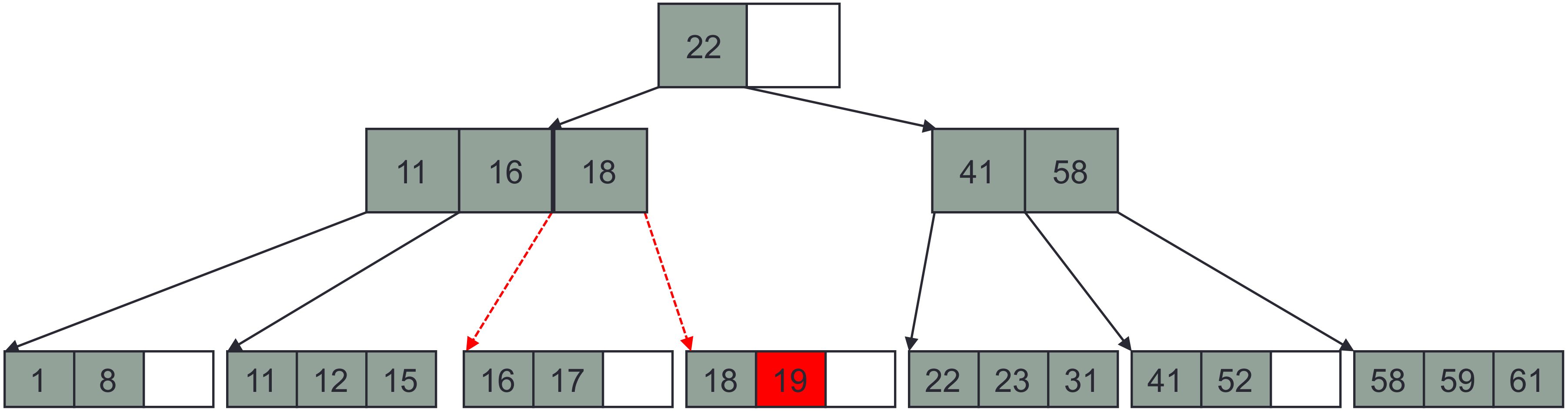
Insert 19 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

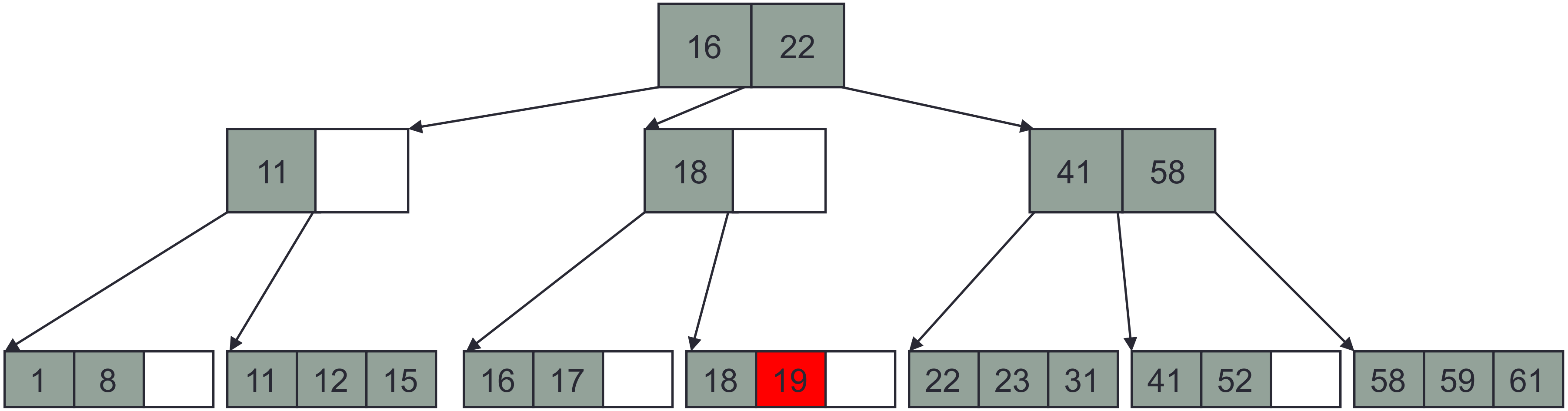
Insert 19 (Update - Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

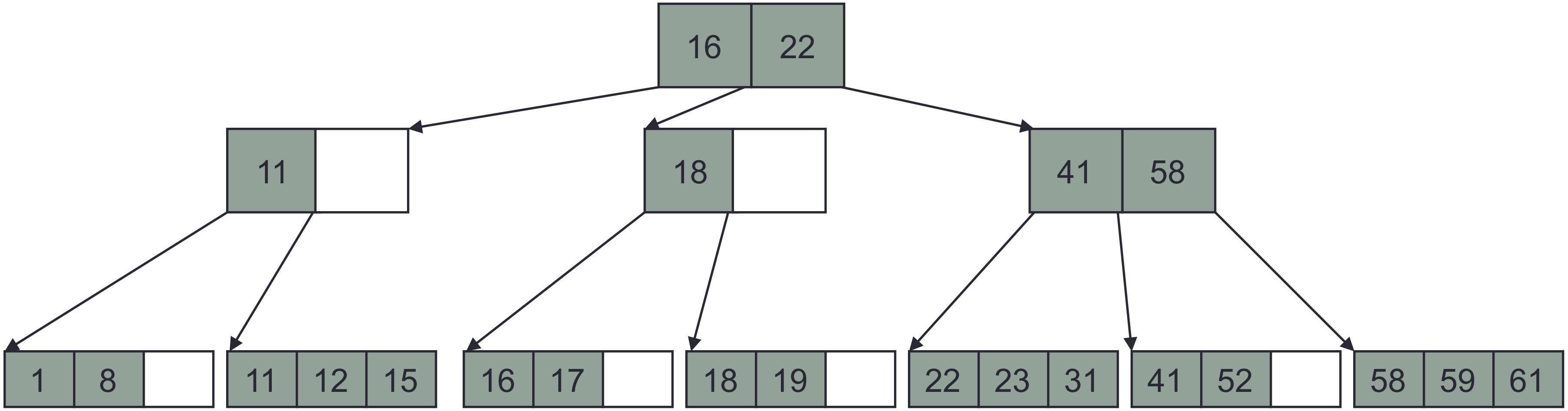
Insert 19 (Update – Overflow - Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

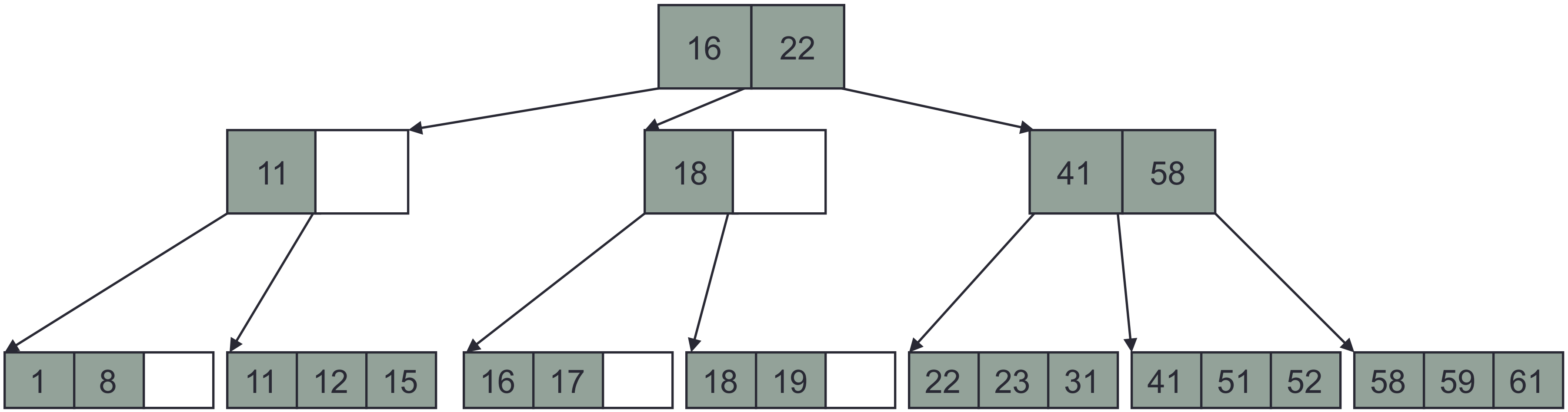
Insert 51



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

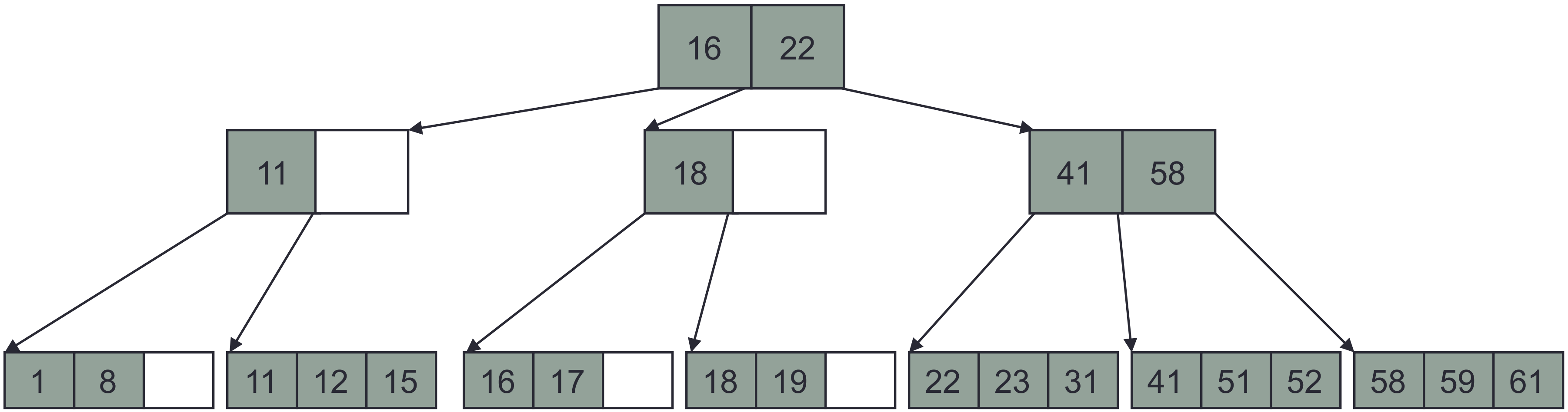
Insert 51



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

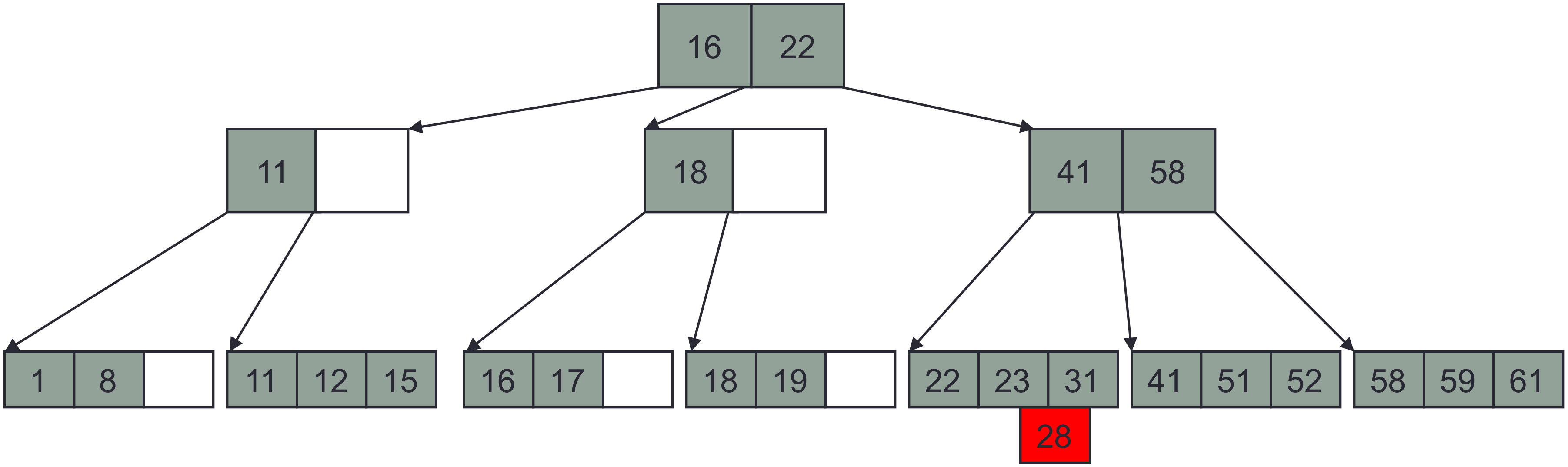
Insert 28



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

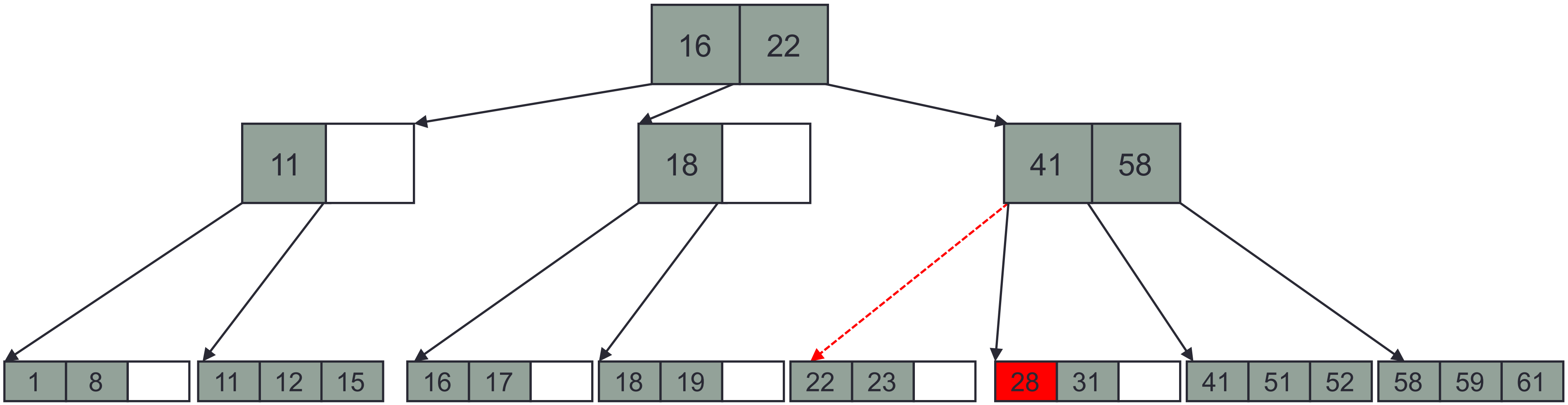
Insert 28 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

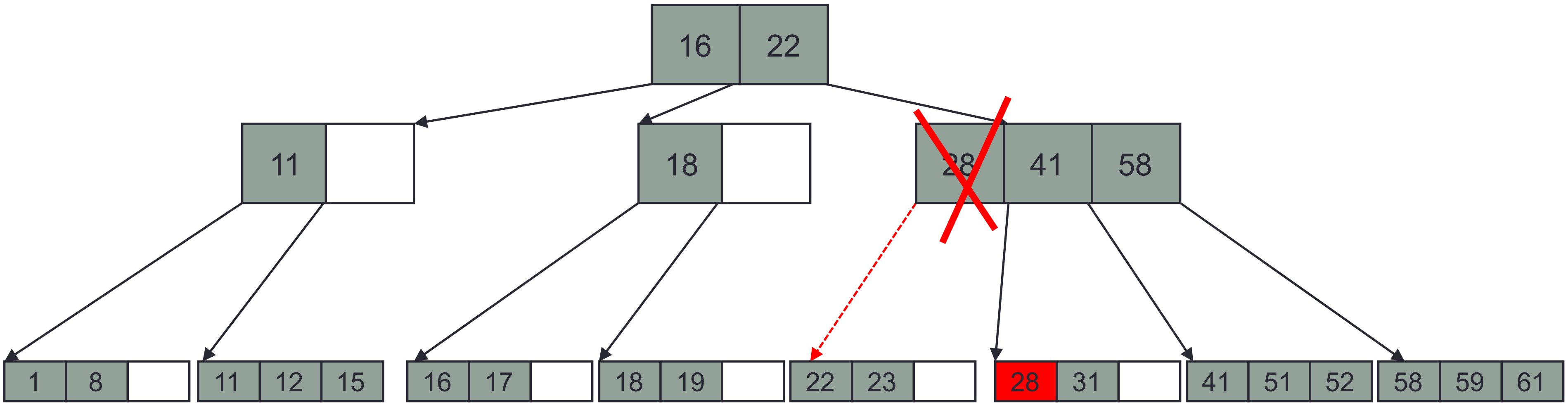
Insert 28 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

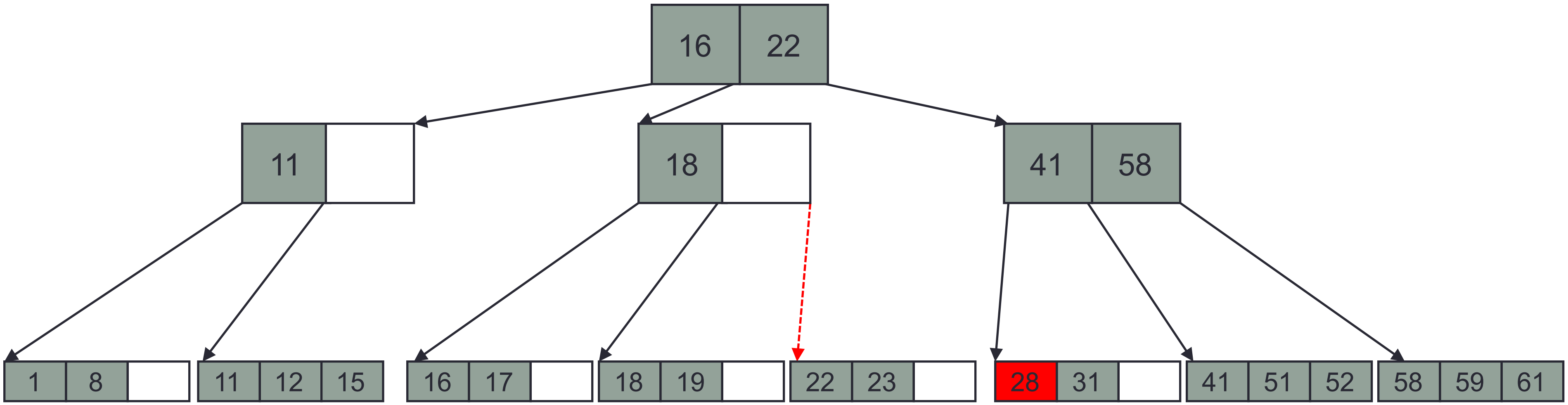
Insert 28 (Update - Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

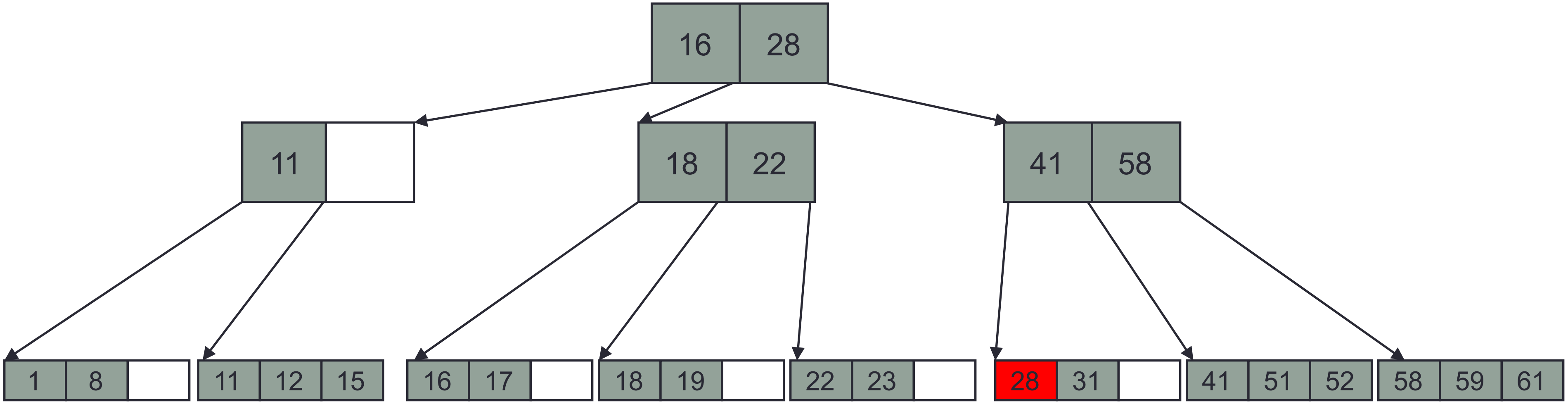
Insert 28 (Update – Overflow – Transfer Child)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

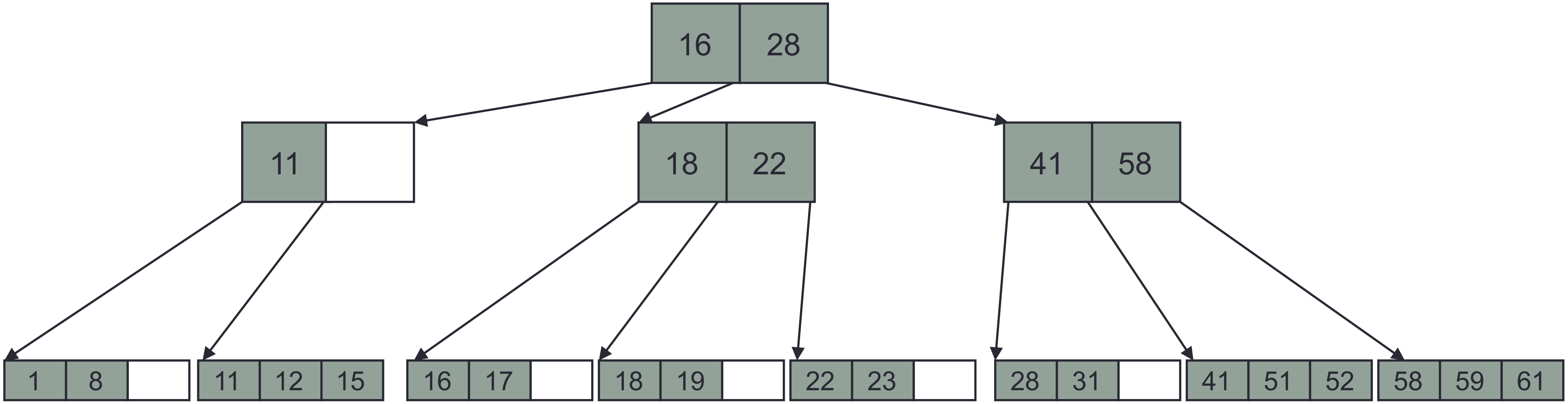
Insert 28 (Update – Overflow – Transfer Child - Update)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

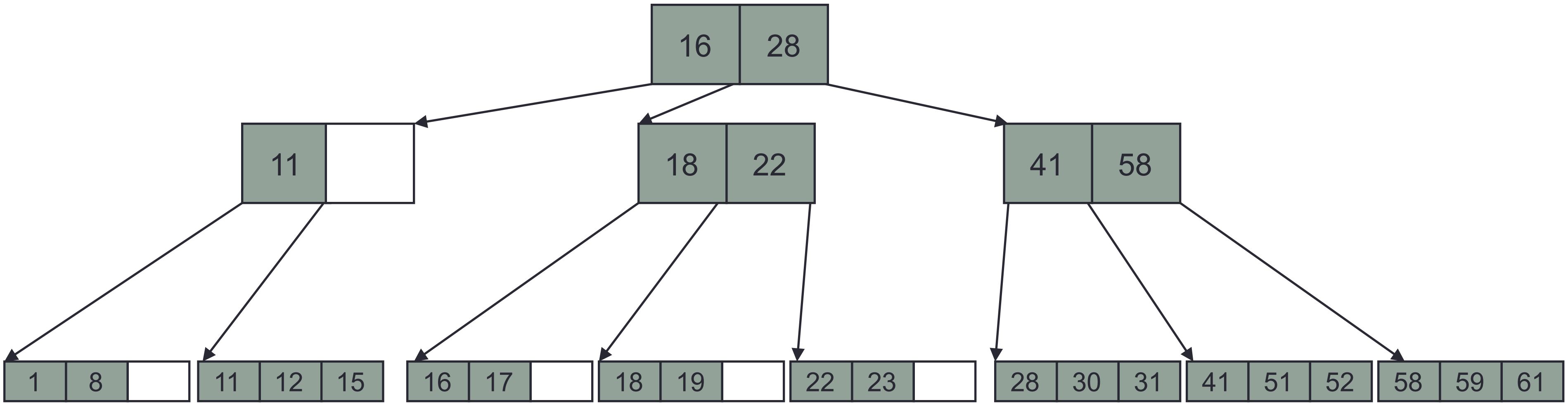
Insert 30



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

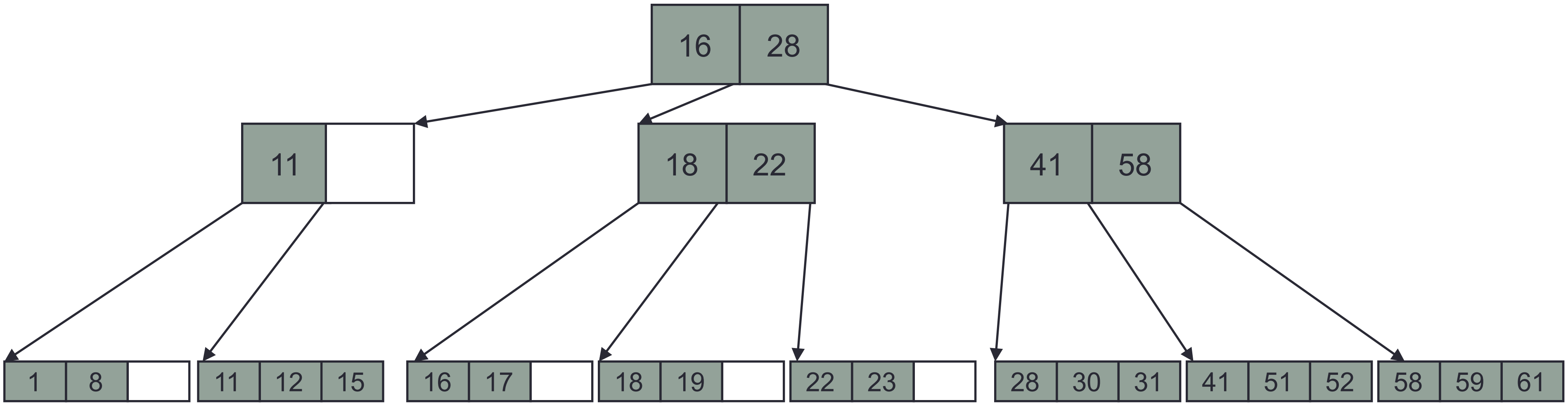
Insert 30



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

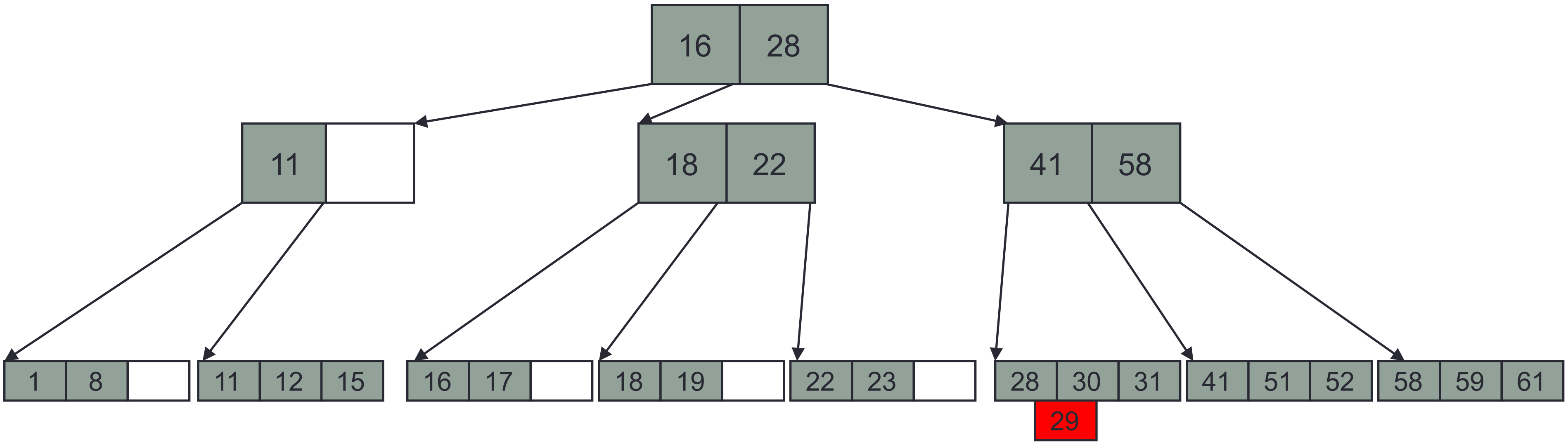
Insert 29



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

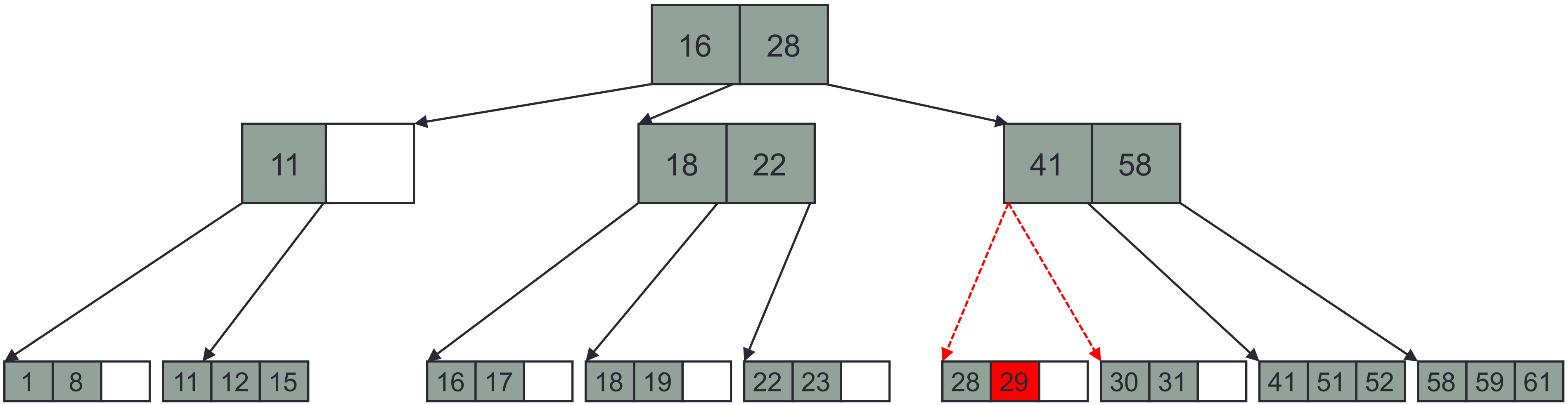
Insert 29 (Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

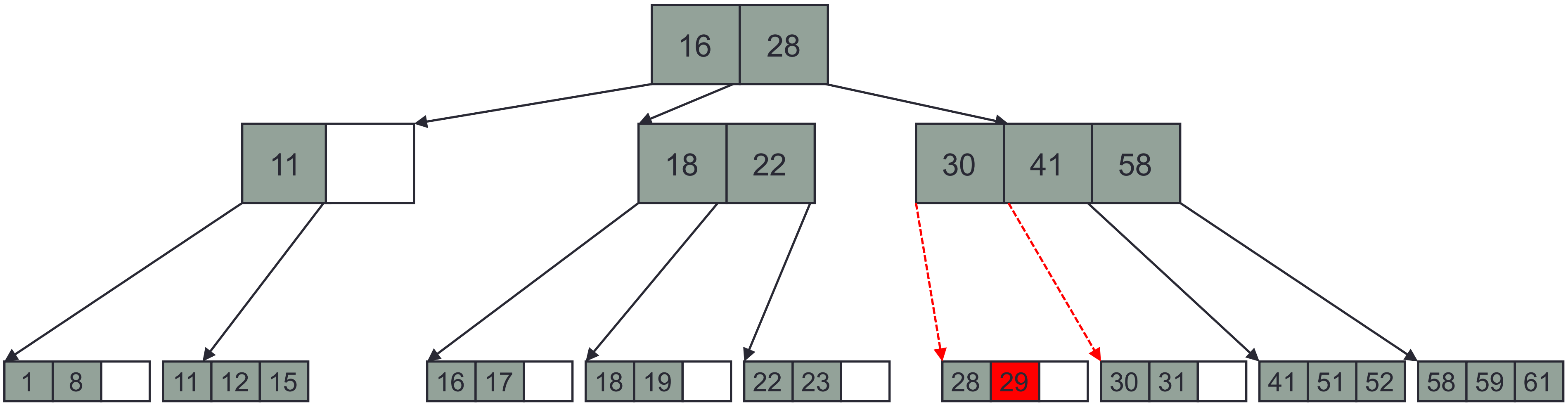
Insert 29 (Split)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

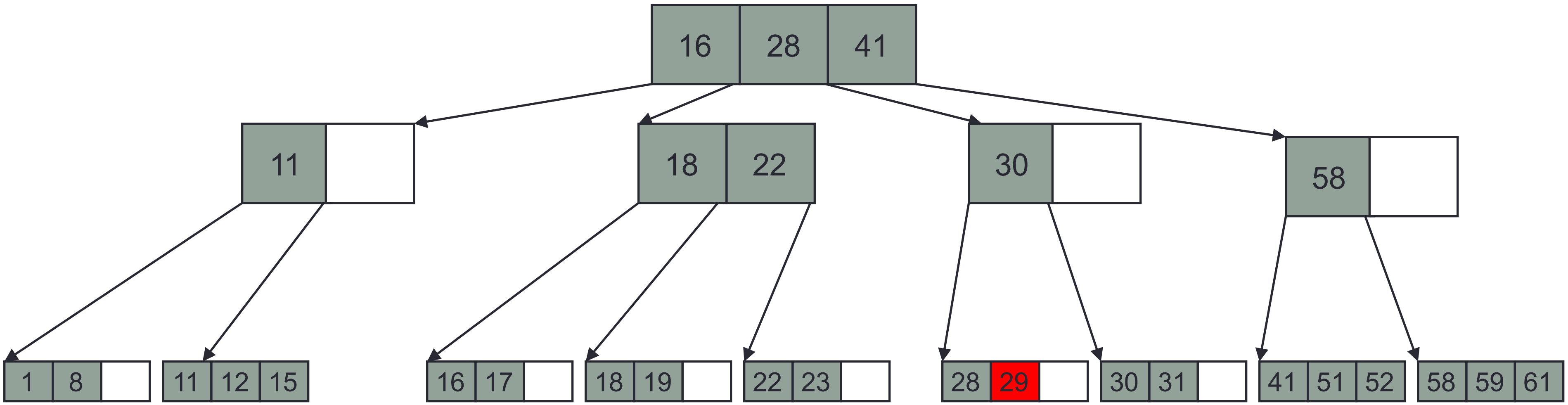
Insert 29 (Update - Overflow)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)

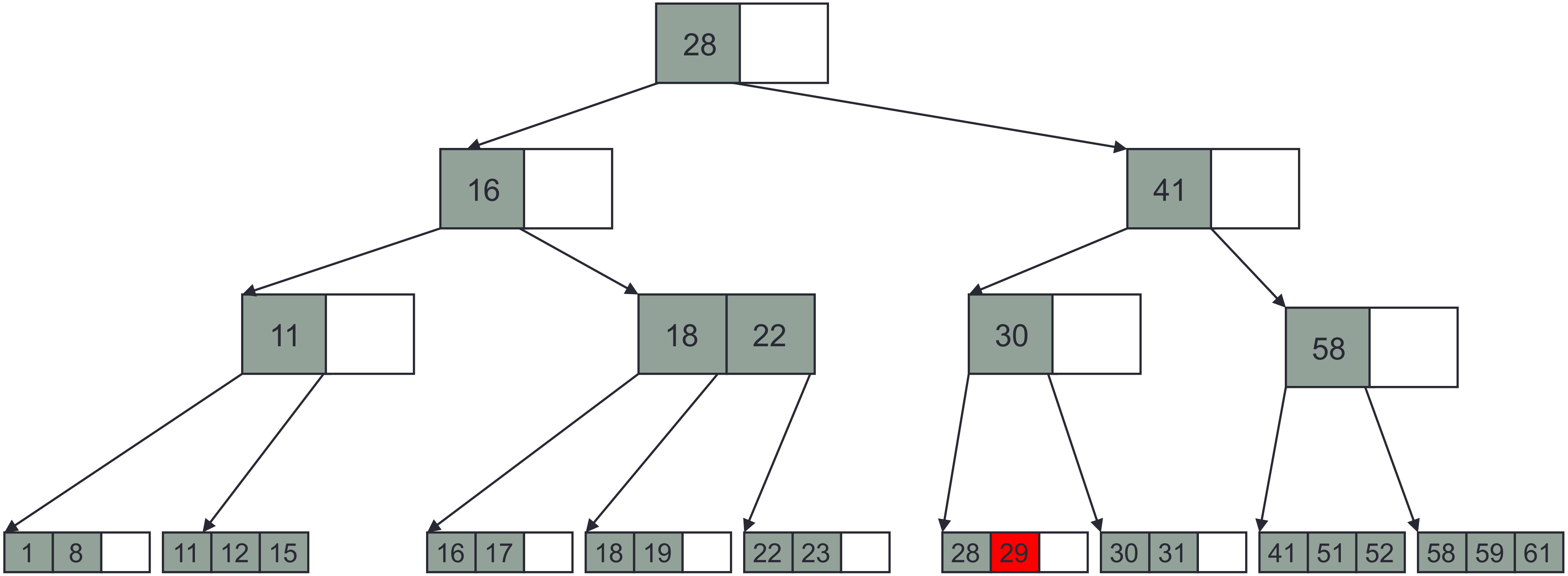
Insert 29 (Update – Overflow - Split)



B+ tree of order 3 (M=3)

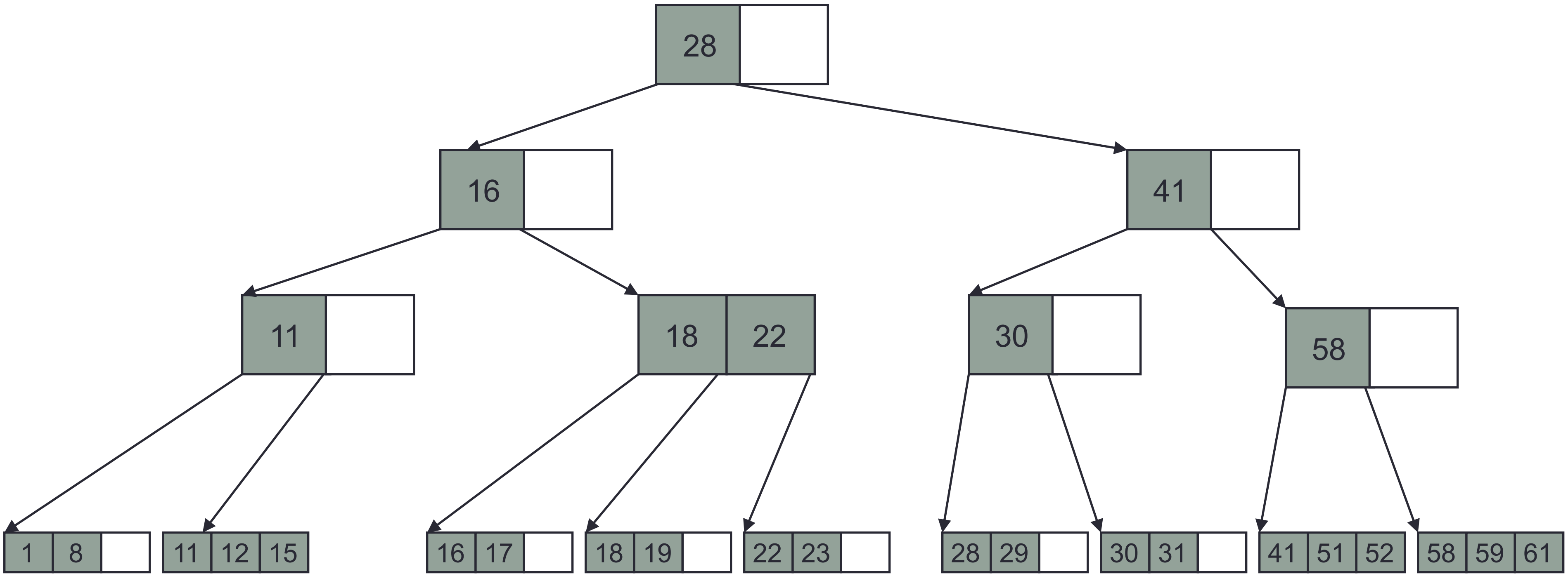
B+ Tree: Insert (Example 2)

Insert 29 (Update – Overflow – Split Again)



B+ tree of order 3 (M=3)

B+ Tree: Insert (Example 2)



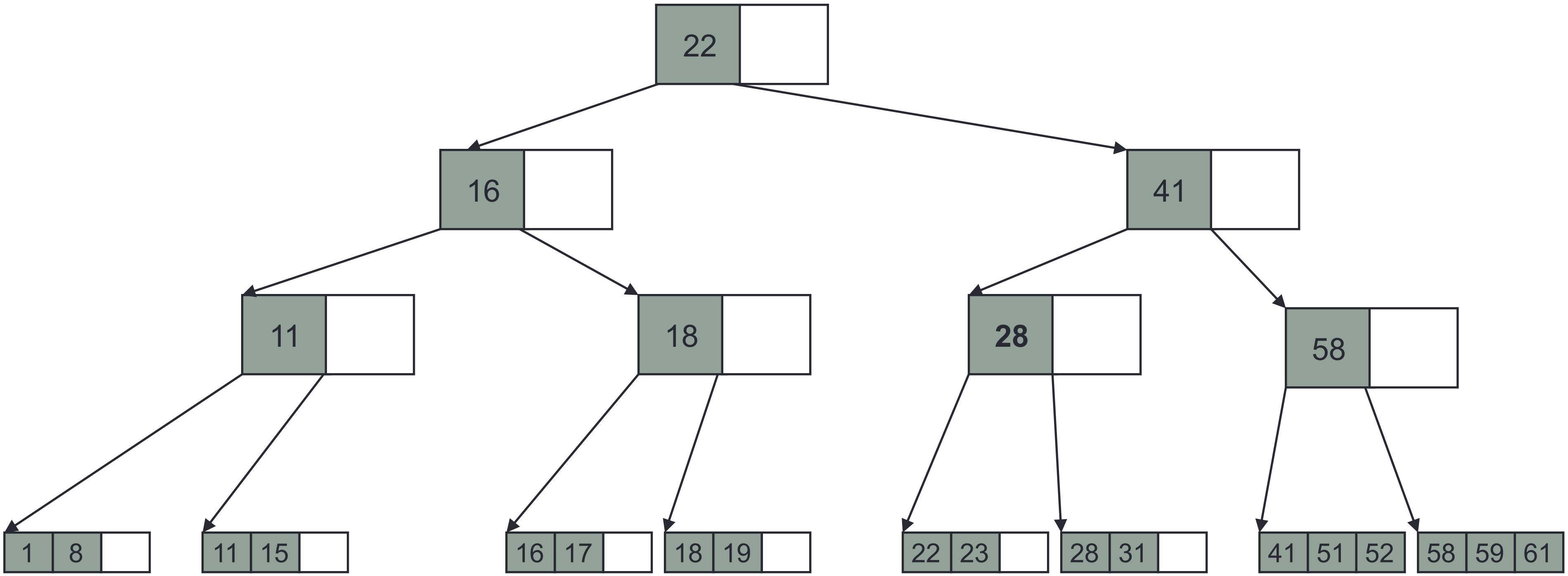
B+ tree of order 3 (M=3)

B+ Tree: Delete

1. Search for a leaf node N from which data with key K is to be deleted.
2. Delete K from N.
   - If N has minimum number of data elements, delete is complete.
   - Otherwise, if there are fewer data elements "underflow" takes place. Underflow is dealt with by:
     - Borrowing a datum (or a subtree) from one of the close sibling nodes.
     - Or, by merging N with one of its close siblings.

B+ Tree: Delete

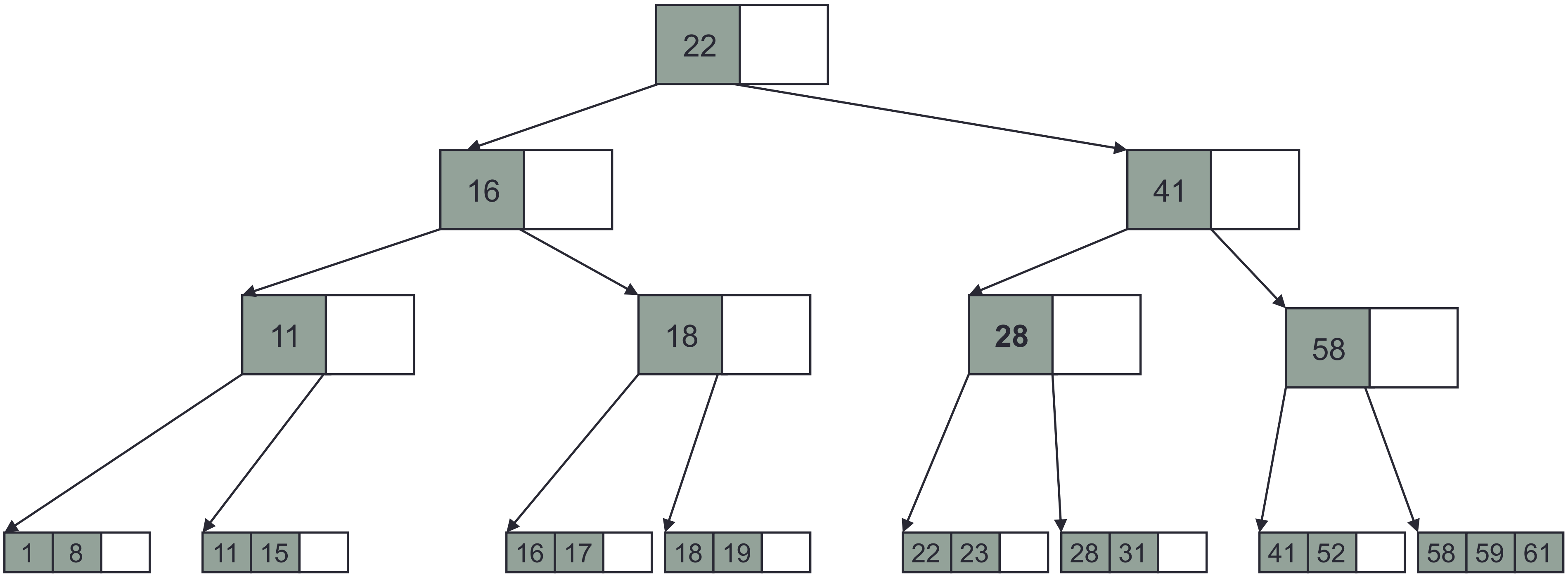
Delete 51



B+ tree of order 3 (M=3)

B+ Tree: Delete

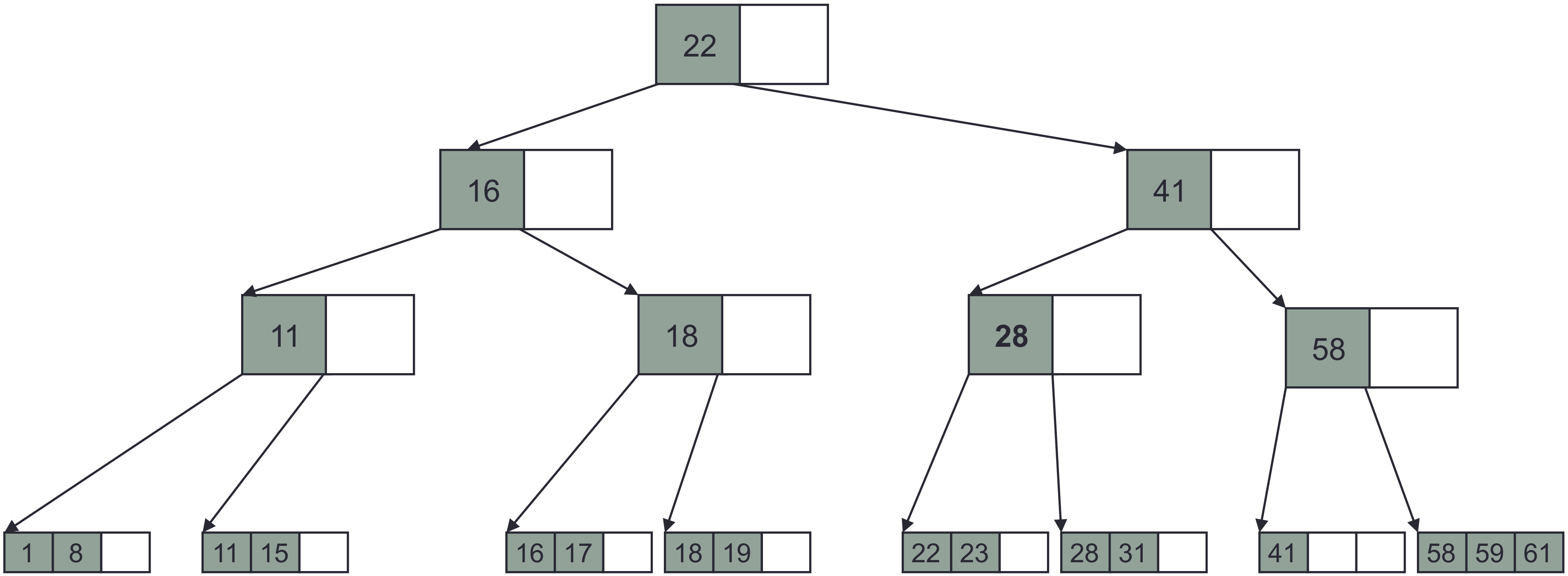
Delete 52



B+ tree of order 3 (M=3)

B+ Tree: Delete

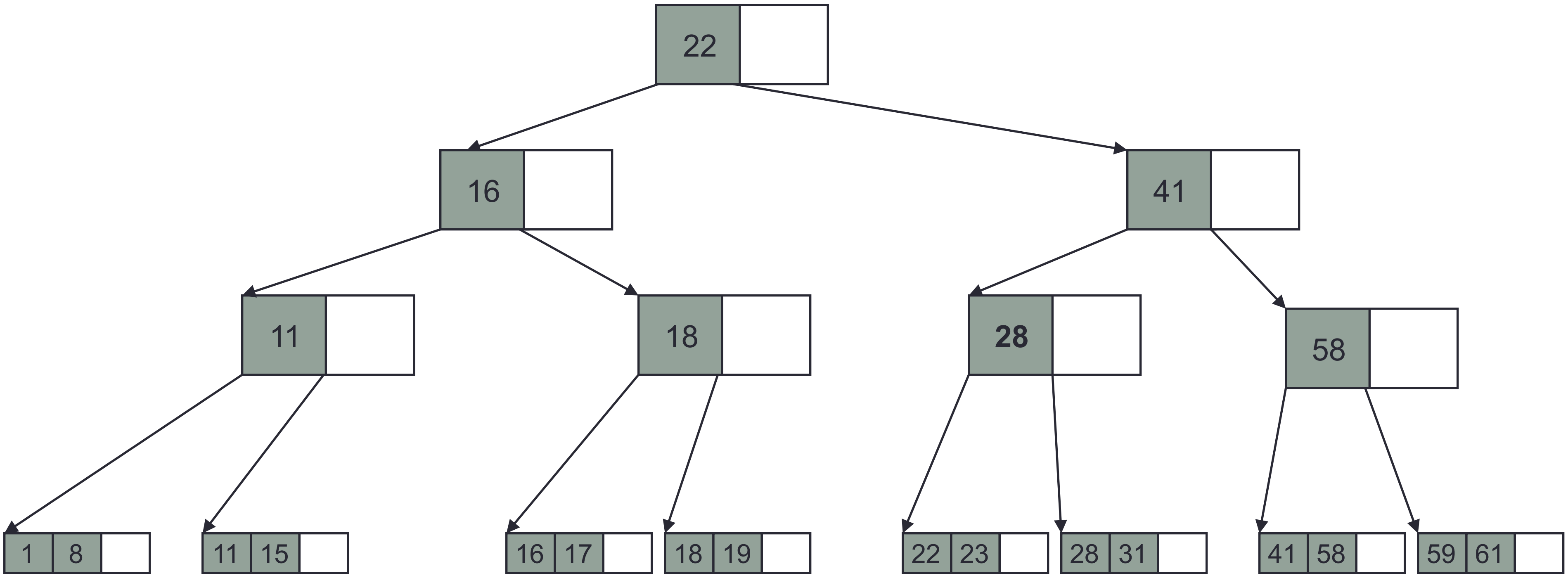
Delete 52 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

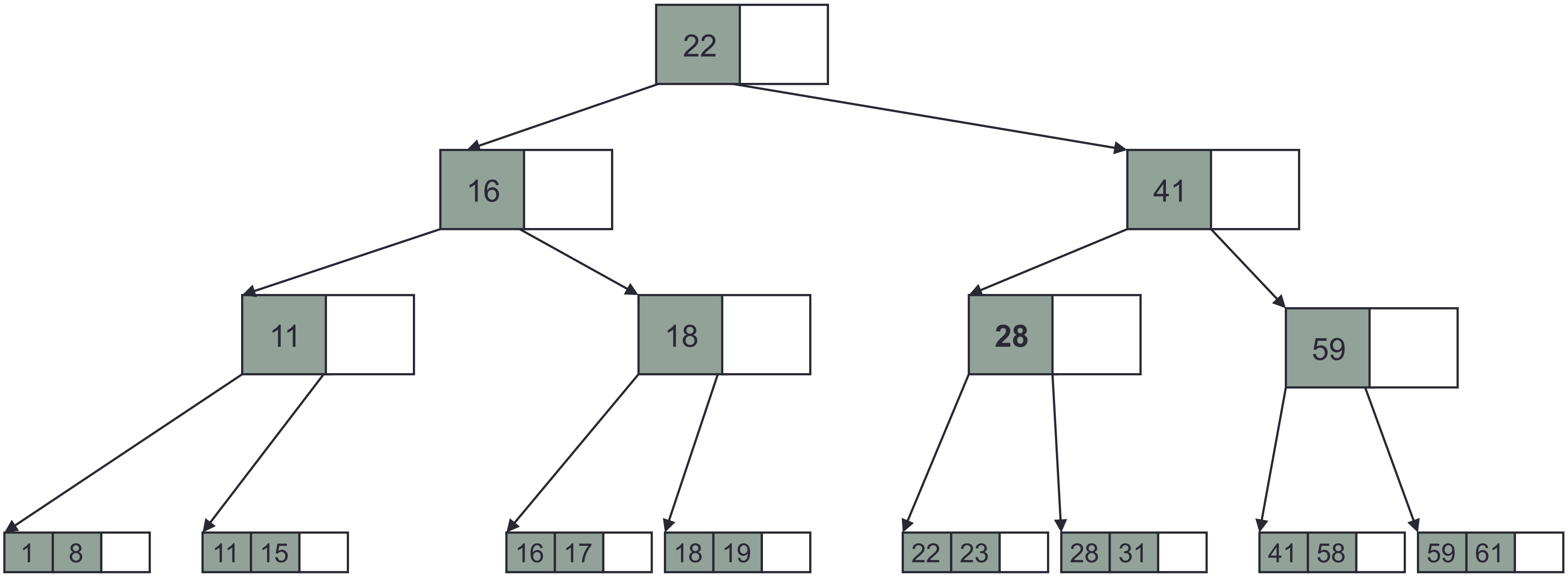
Delete 52 (Borrow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

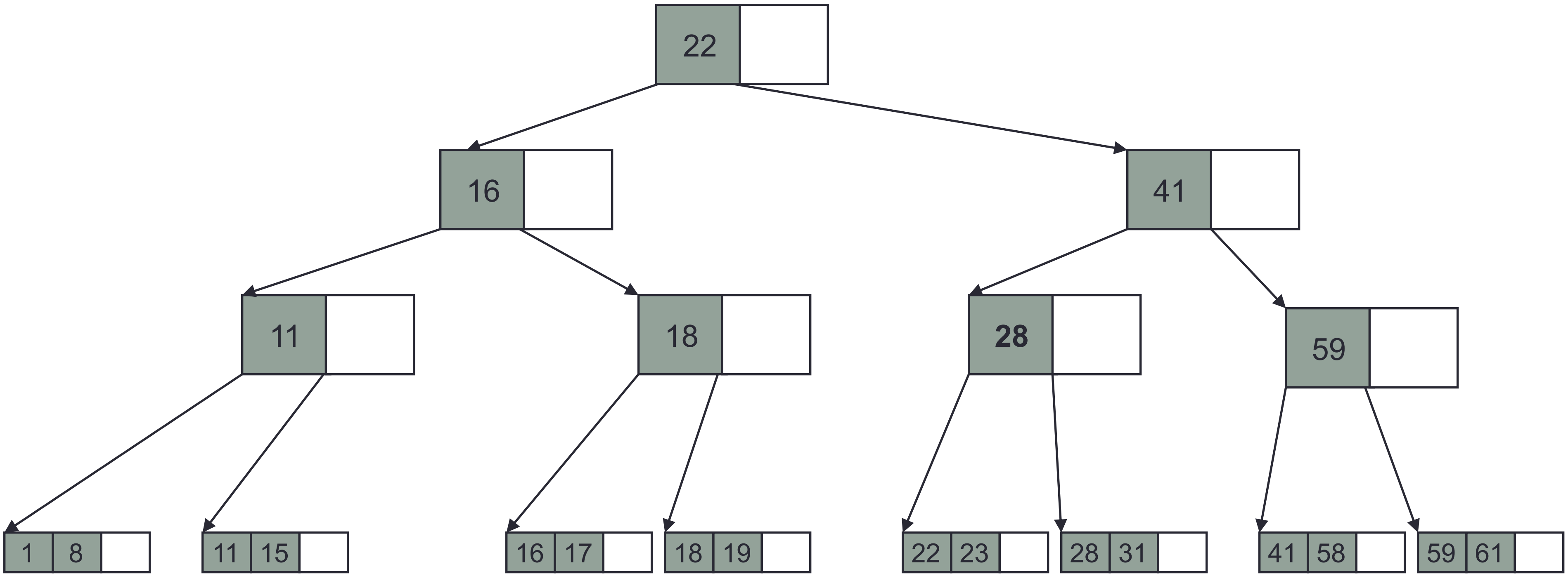
Delete 52 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Delete

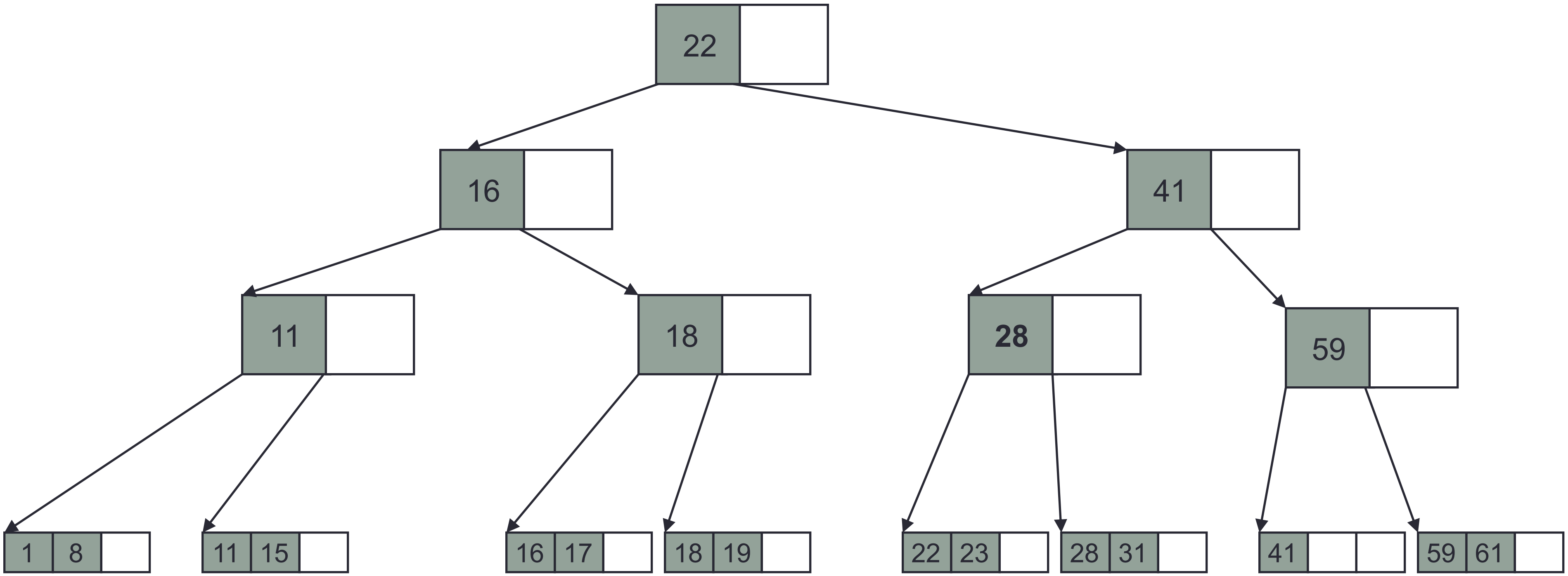
Delete 58



B+ tree of order 3 (M=3)

B+ Tree: Delete

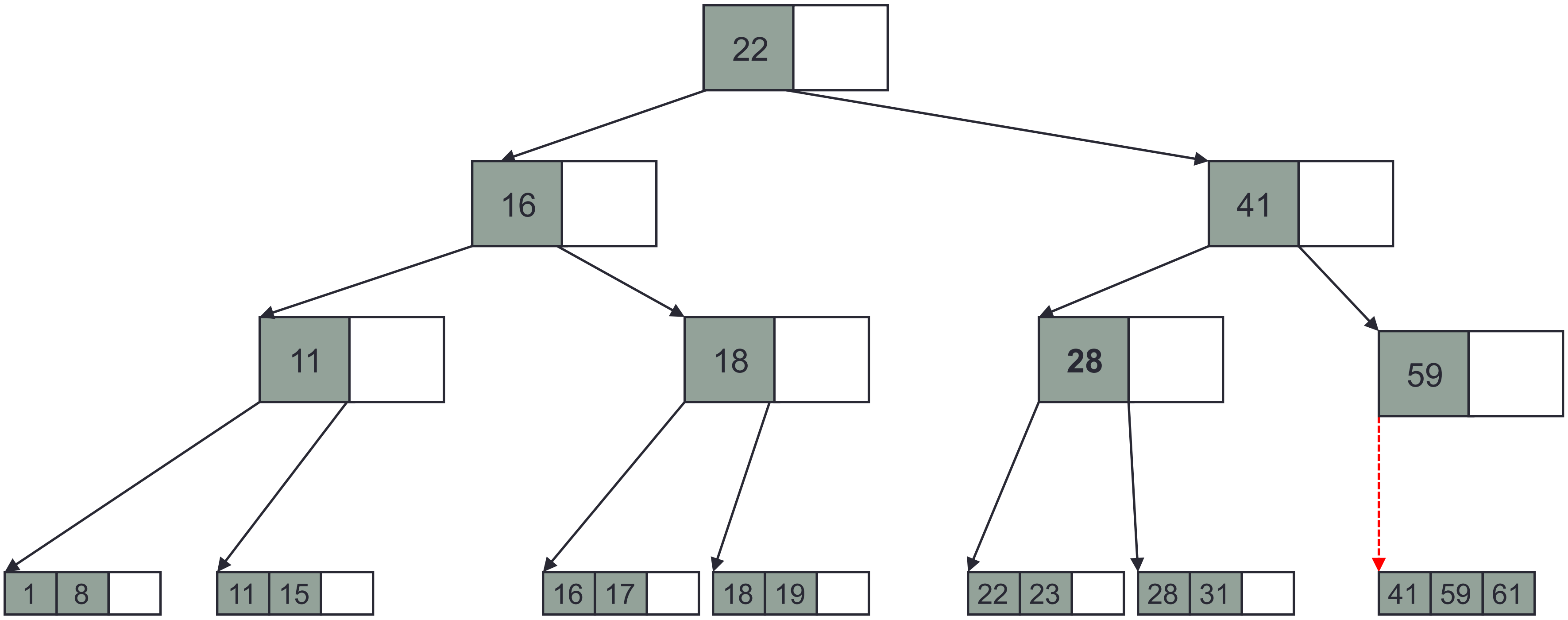
Delete 58 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

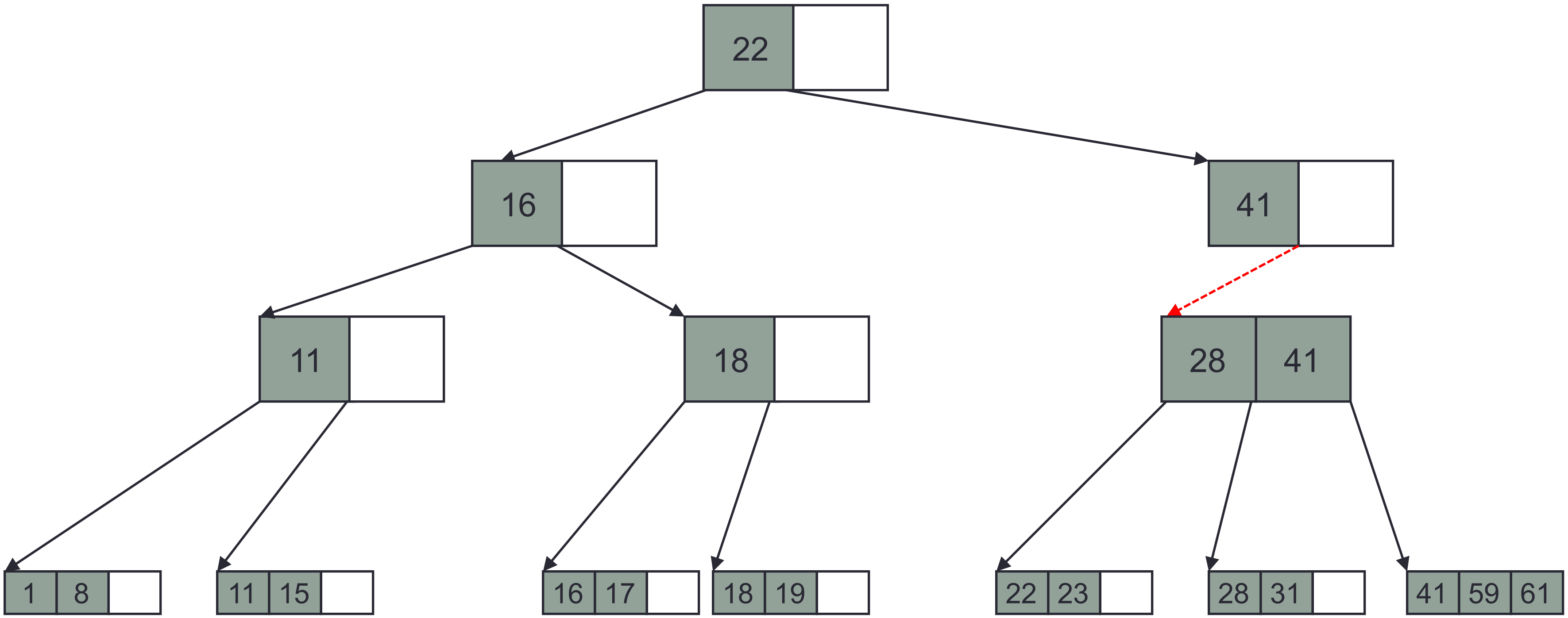
Delete 58 (Merge #1)



B+ tree of order 3 (M=3)

B+ Tree: Delete

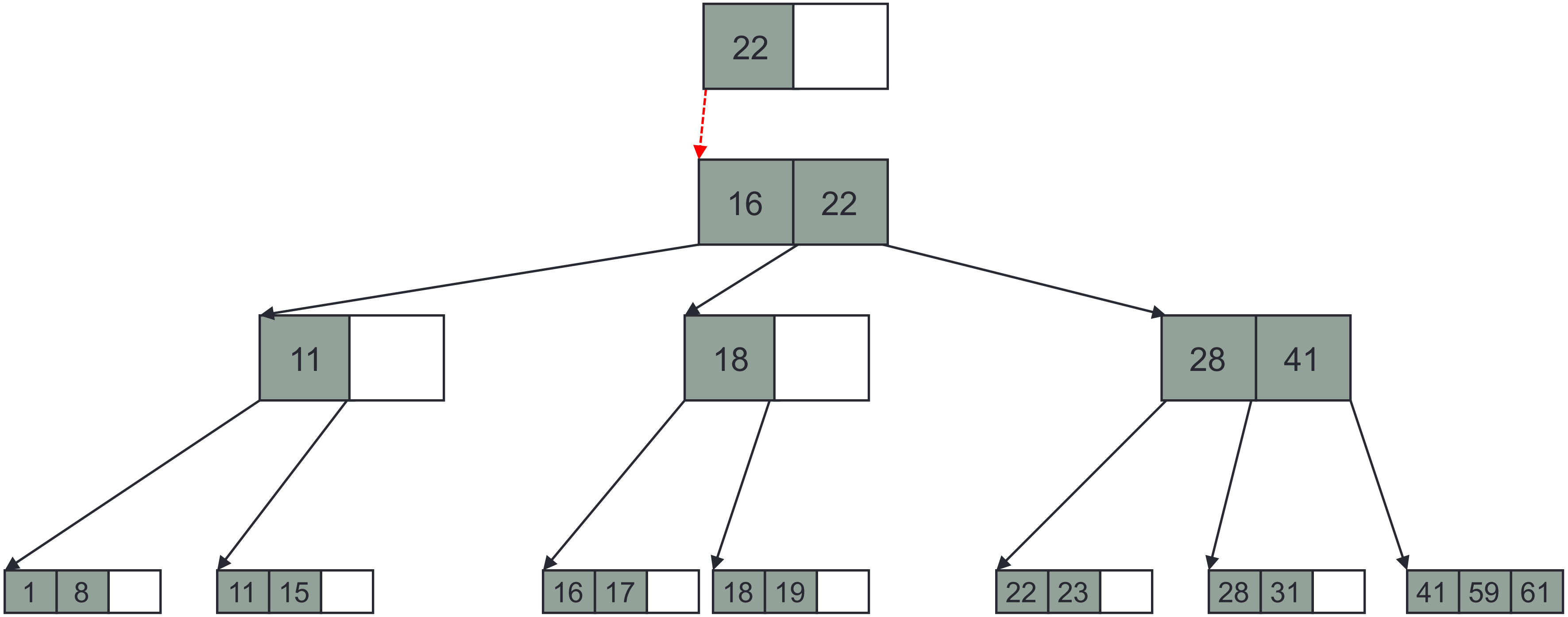
Delete 58 (Update/Merge #2)



B+ tree of order 3 (M=3)

B+ Tree: Delete

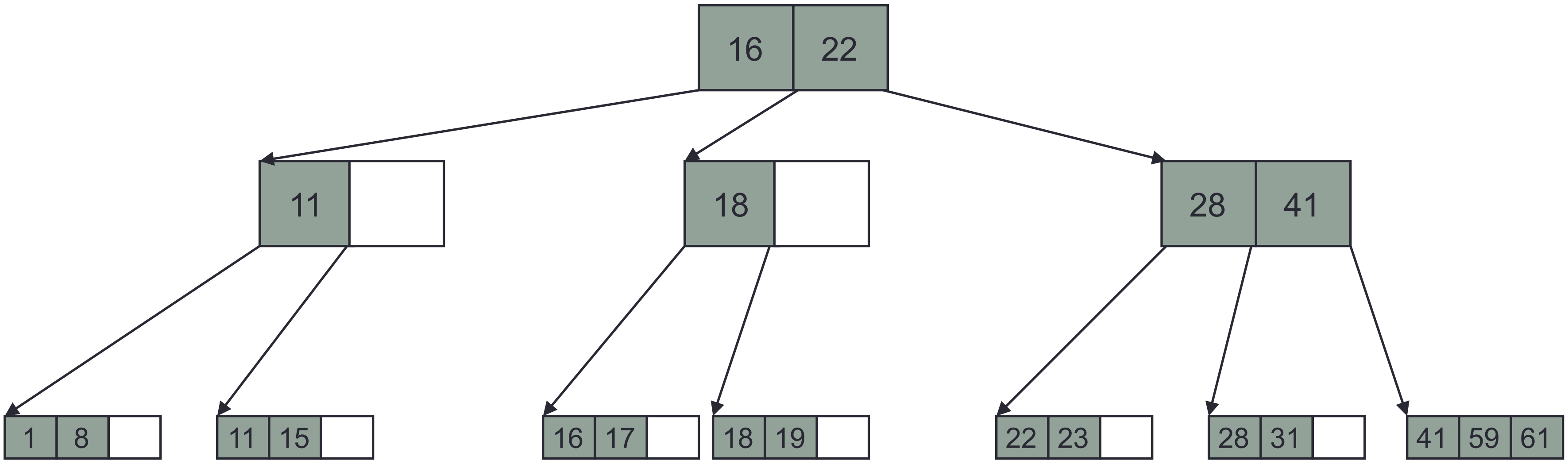
Delete 58 (Update/Merge #3)



B+ tree of order 3 (M=3)

B+ Tree: Delete

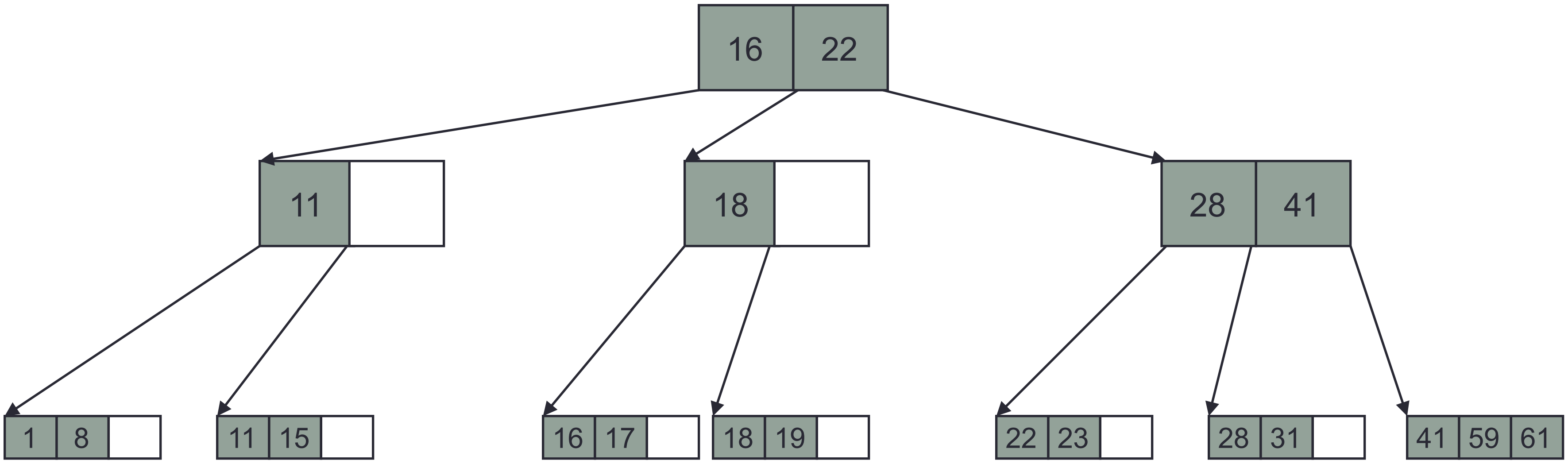
Delete 58 (Update/Merge #3)



B+ tree of order 3 (M=3)

B+ Tree: Delete

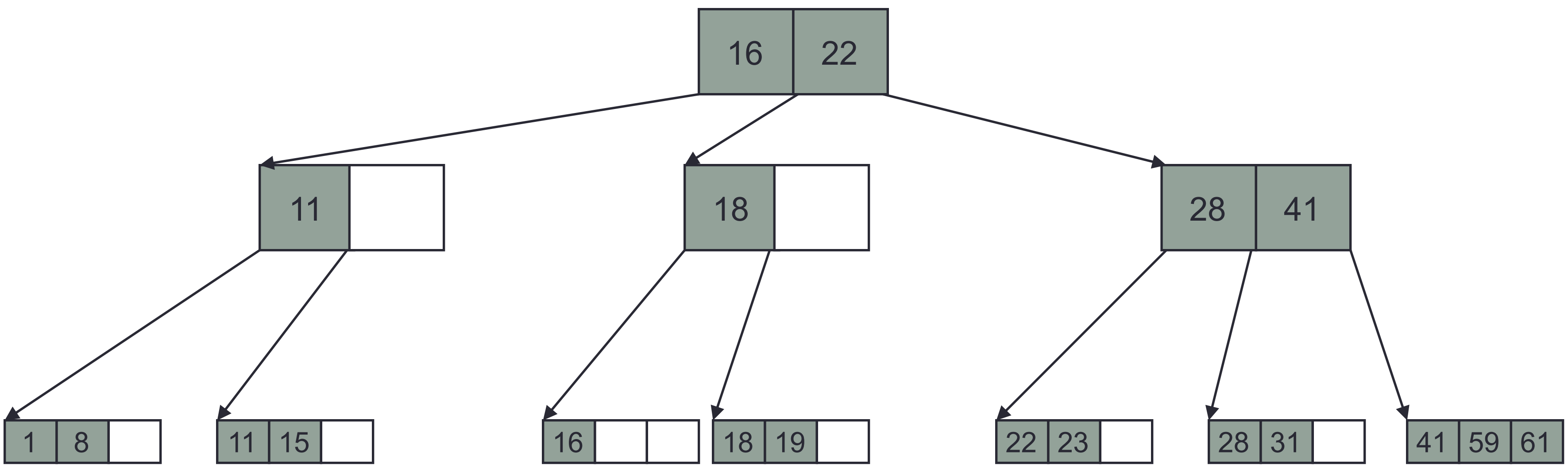
Delete 17



B+ tree of order 3 (M=3)

B+ Tree: Delete

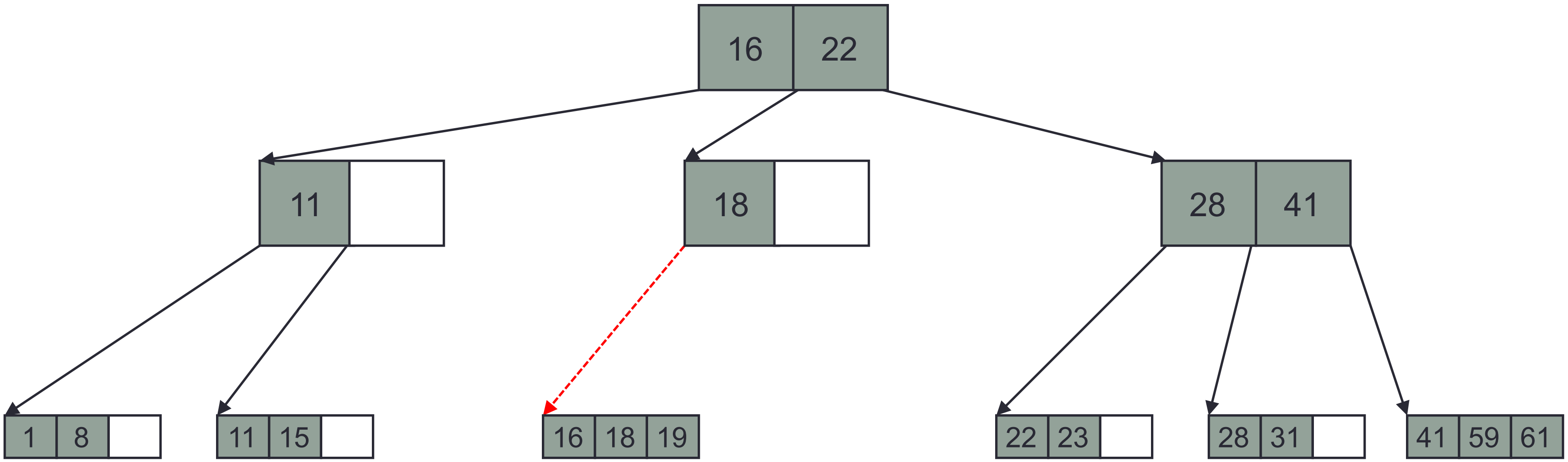
Delete 17 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

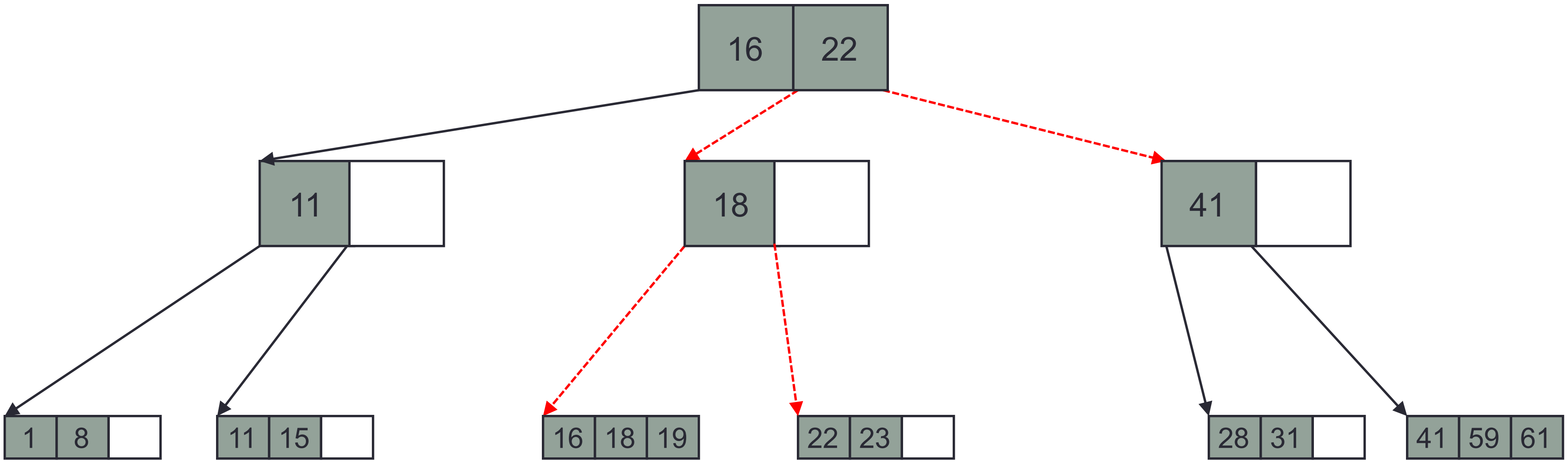
Delete 17 (Merge)



B+ tree of order 3 (M=3)

B+ Tree: Delete

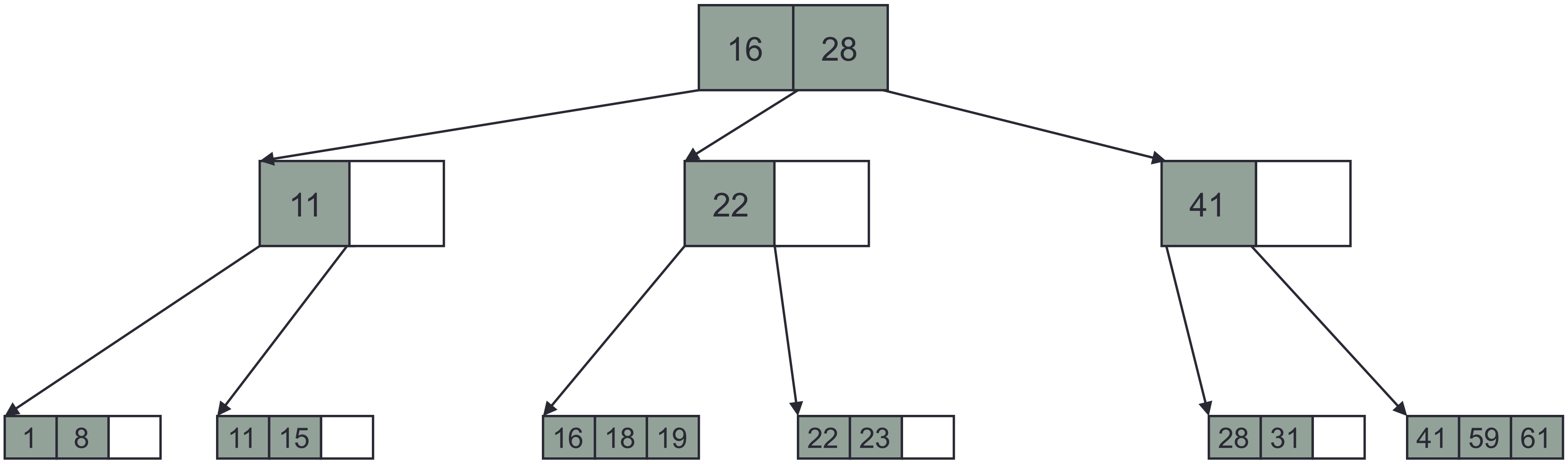
Delete 17 (Borrow Child)



B+ tree of order 3 (M=3)

B+ Tree: Delete

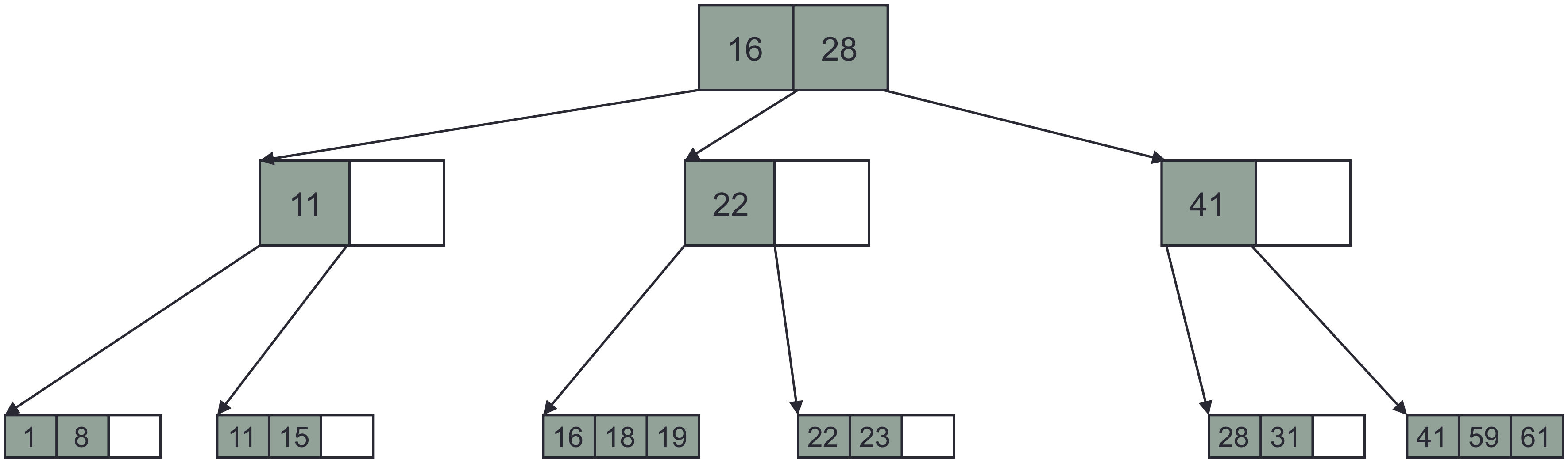
Delete 17 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Delete

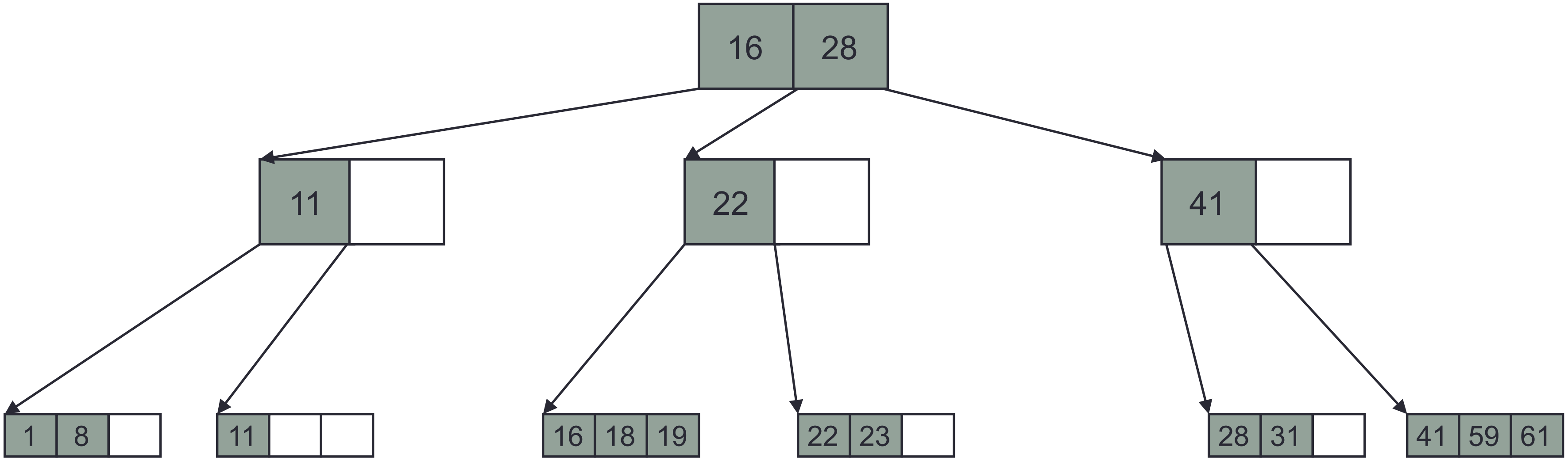
Delete 15



B+ tree of order 3 (M=3)

B+ Tree: Delete

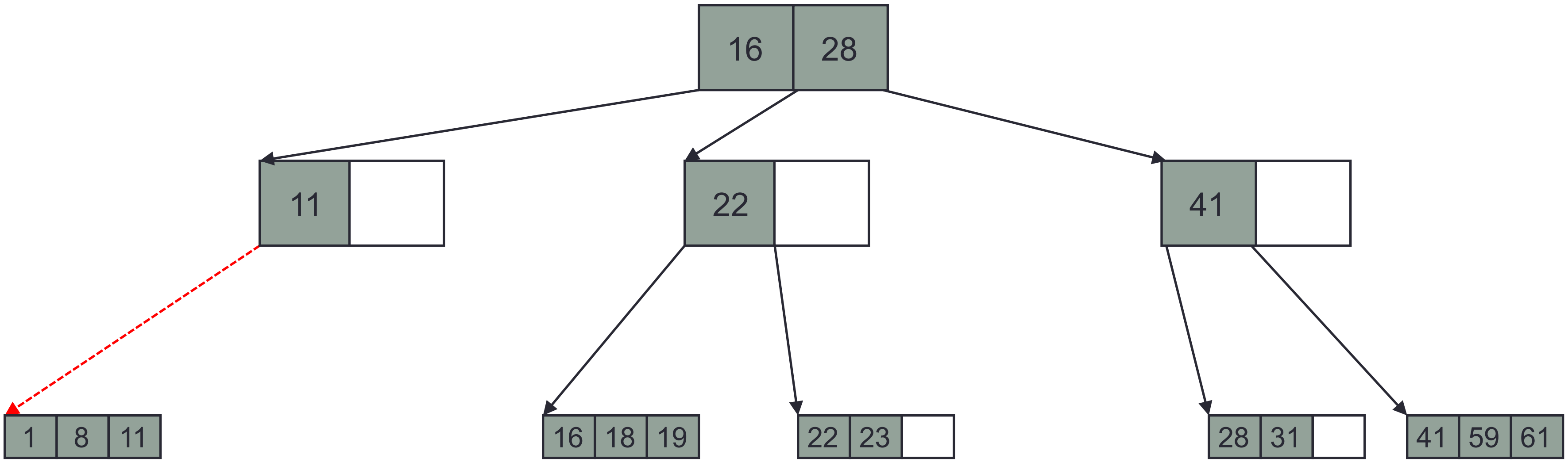
Delete 15 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

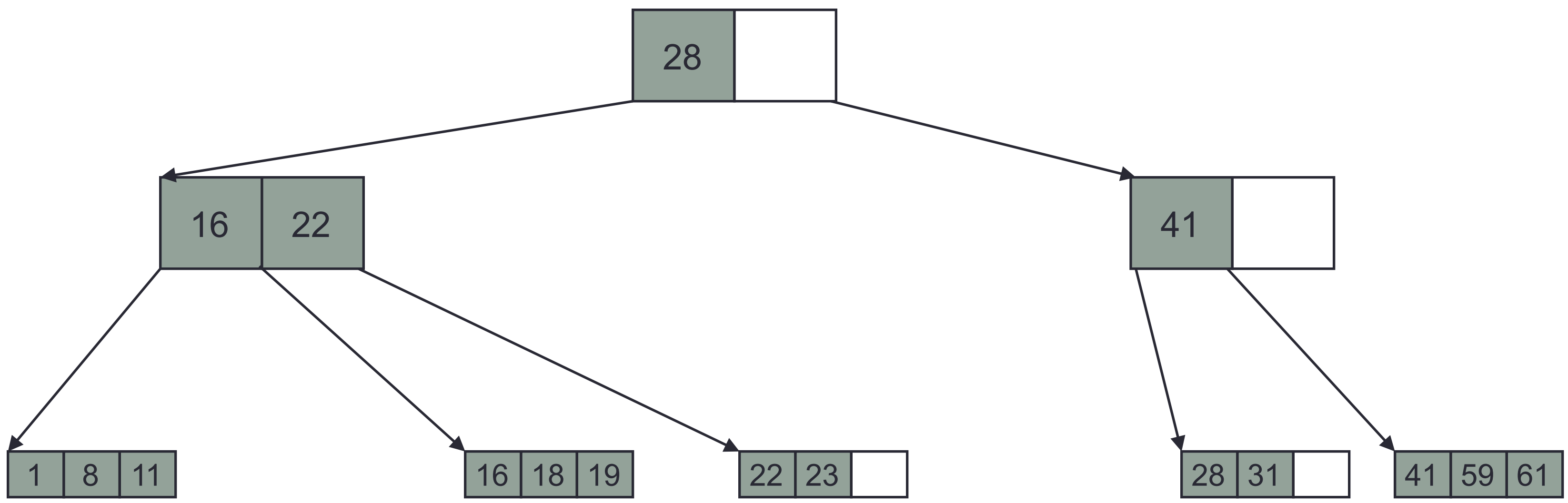
Delete 15 (Merge #1)



B+ tree of order 3 (M=3)

B+ Tree: Delete

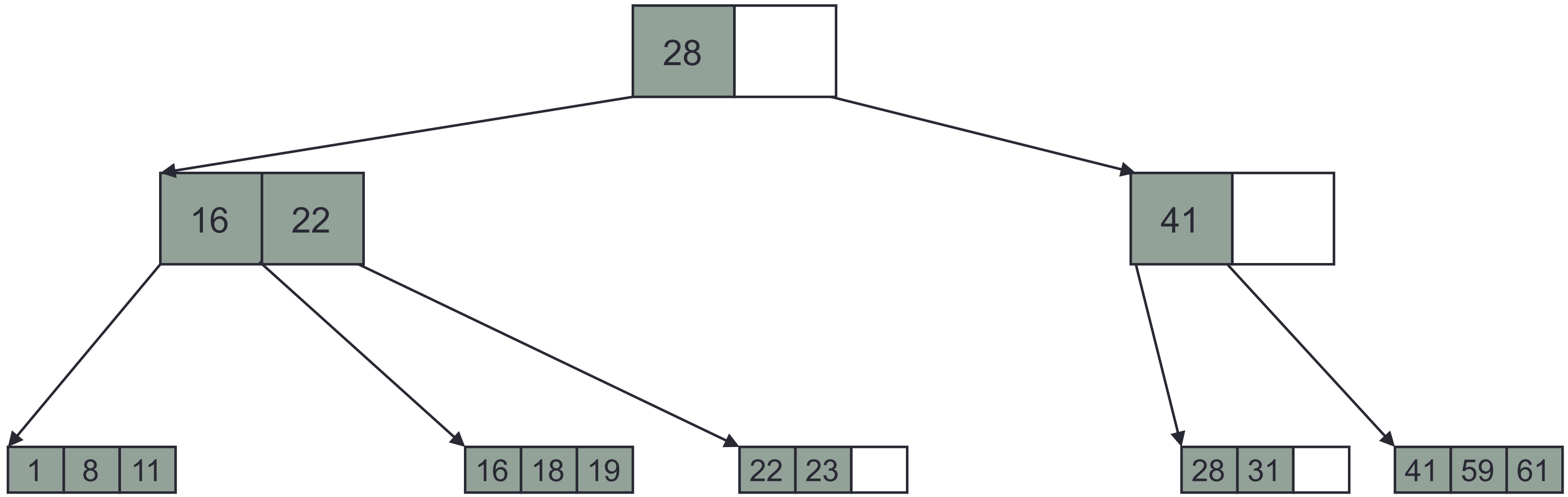
Delete 15 (Update/Merge #2)



B+ tree of order 3 (M=3)

B+ Tree: Delete

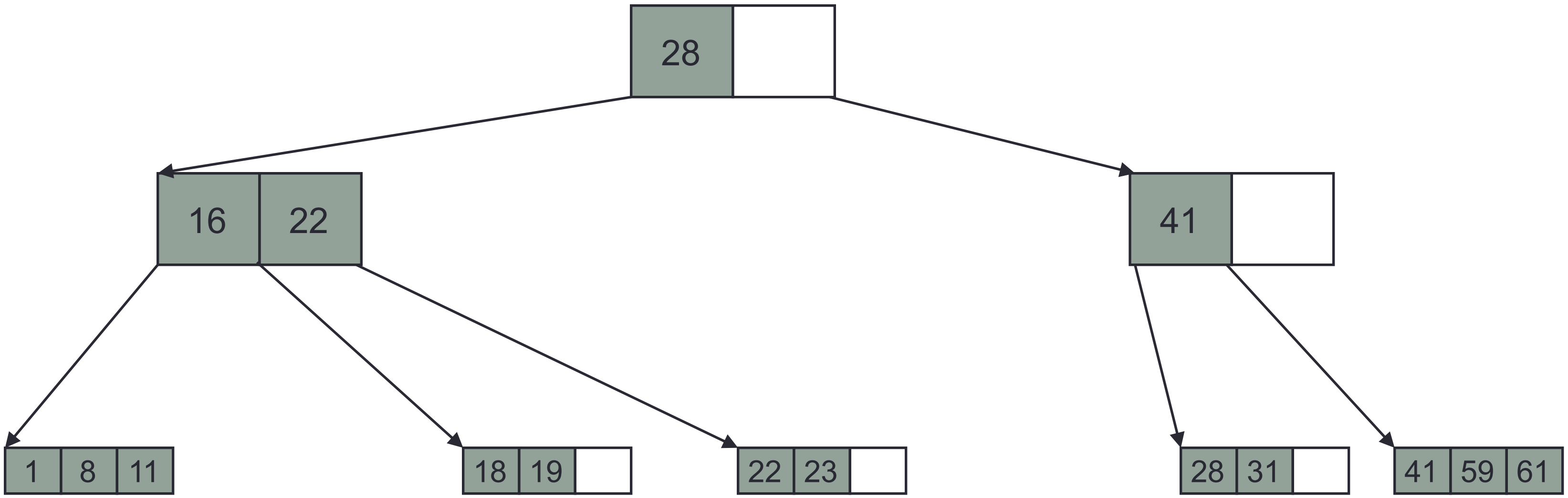
Delete 16



B+ tree of order 3 (M=3)

B+ Tree: Delete

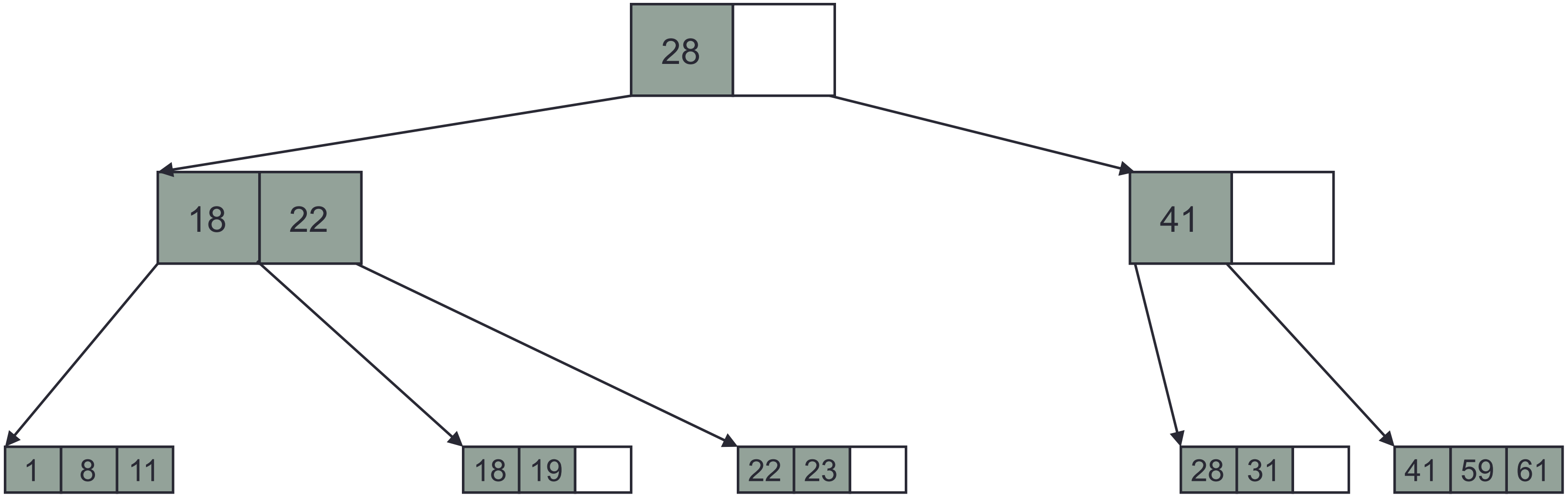
Delete 16 (Normal)



B+ tree of order 3 (M=3)

B+ Tree: Delete

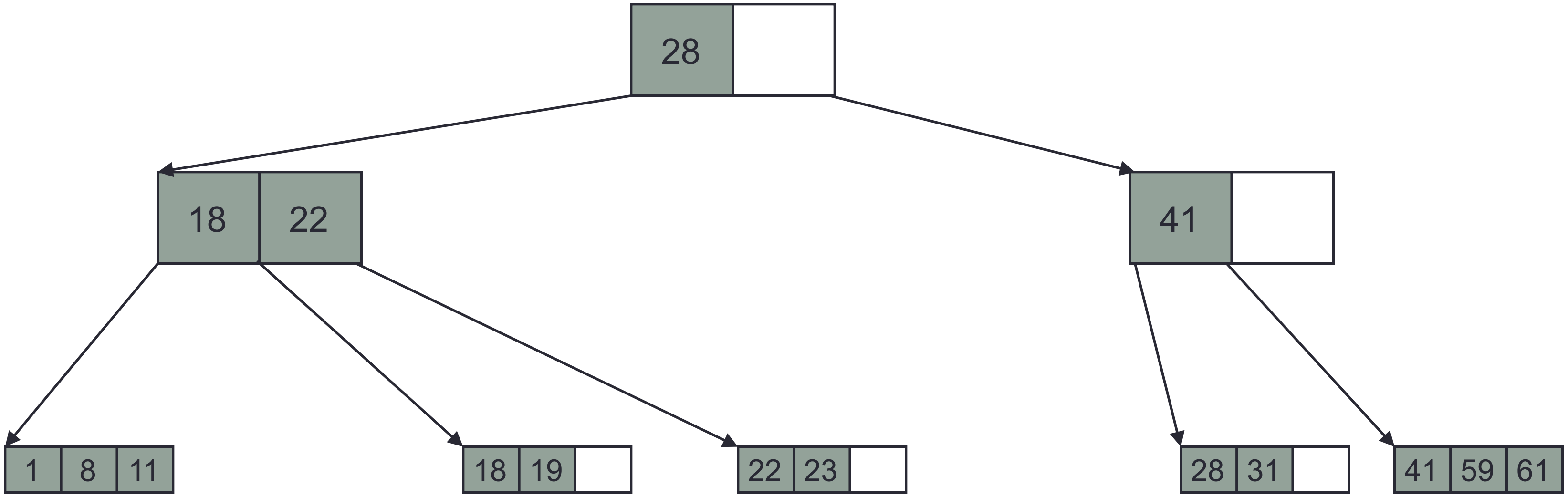
Delete 16 (Update)



B+ tree of order 3 (M=3)

B+ Tree: Delete

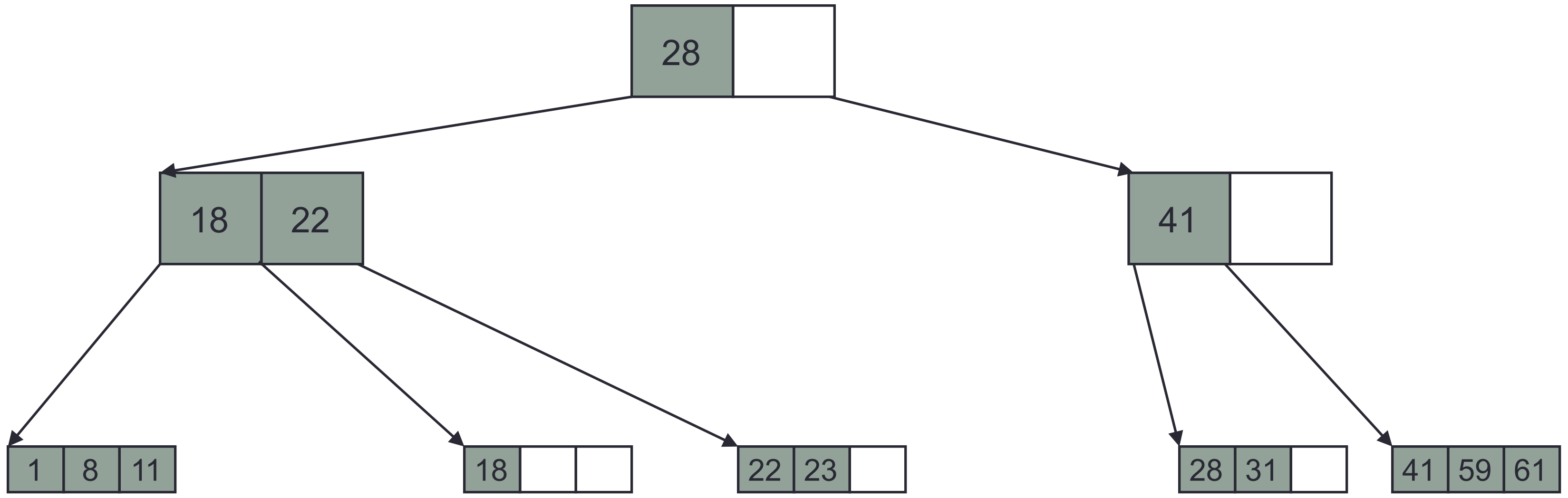
Delete 19



B+ tree of order 3 (M=3)

B+ Tree: Delete

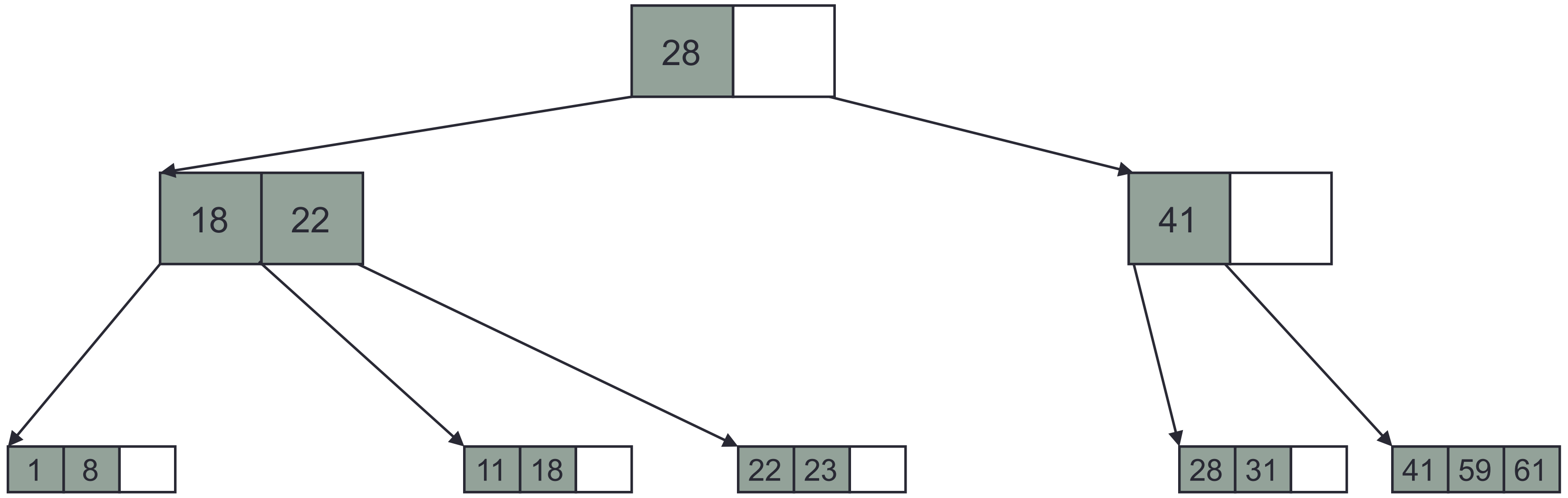
Delete 19 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

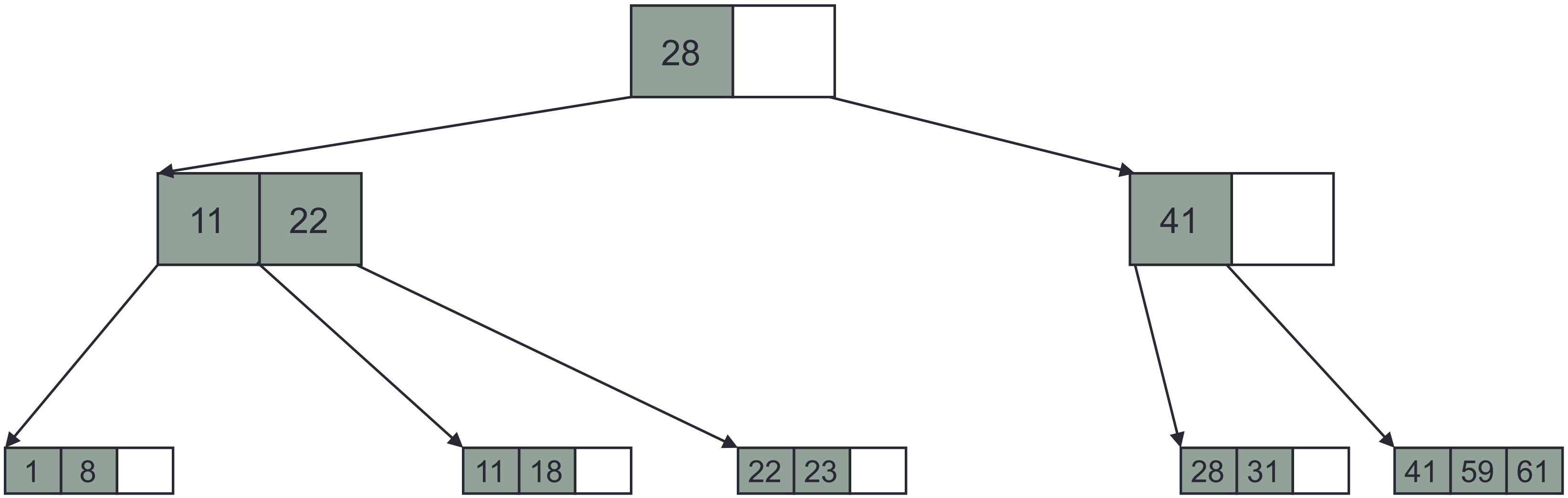
Delete 19 (Borrow)



B+ tree of order 3 (M=3)

B+ Tree: Delete

Delete 19 (Update)



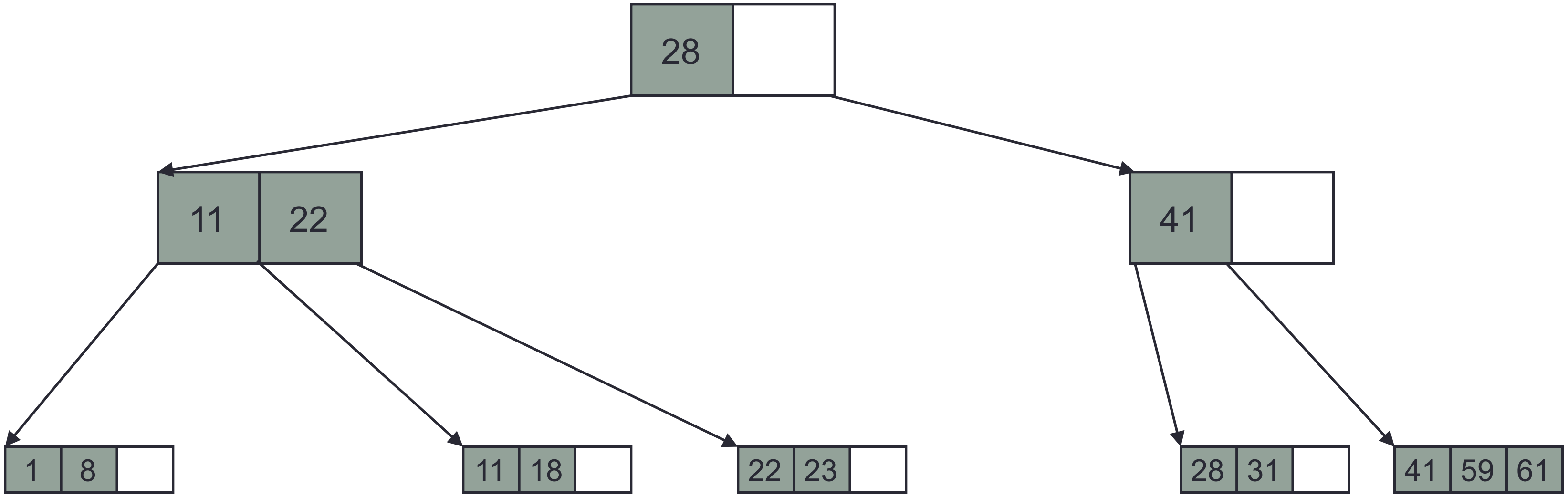
B+ tree of order 3 (M=3)

Reference

- Mark Allen Weiss, Data Structures & Problem Solving Using C++, Pages: 707-715. (Covers B+ trees in some detail)

B+ Tree: Homework

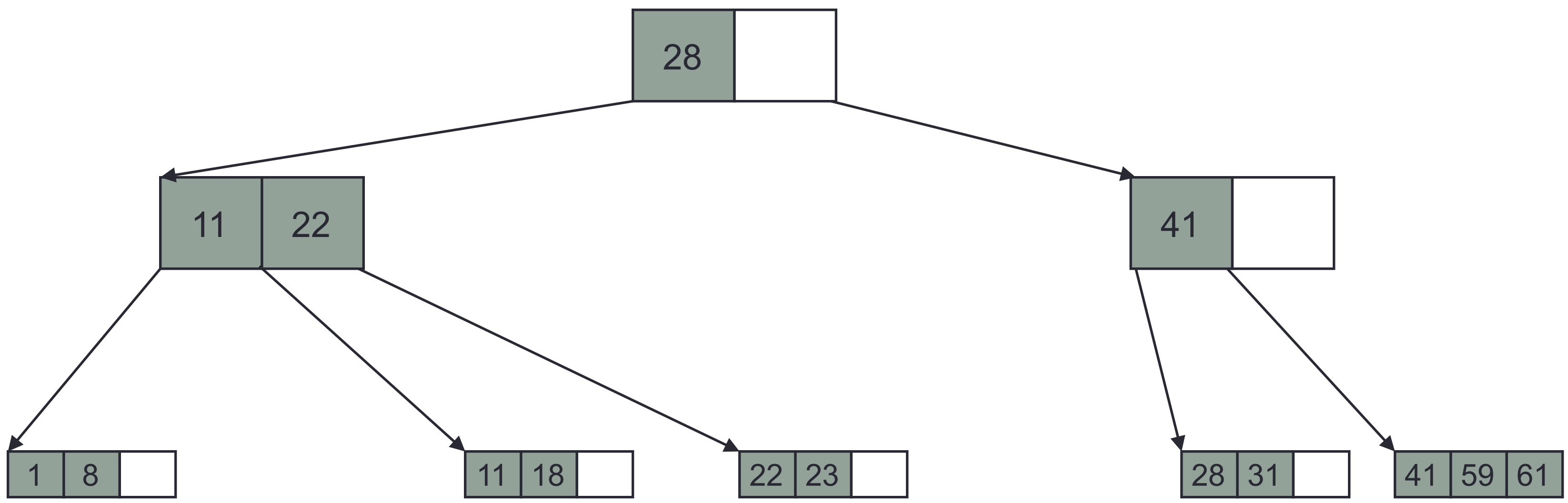
Delete 41
Delete 1
Delete 23



B+ tree of order 3 (M=3)

B+ Tree: Homework

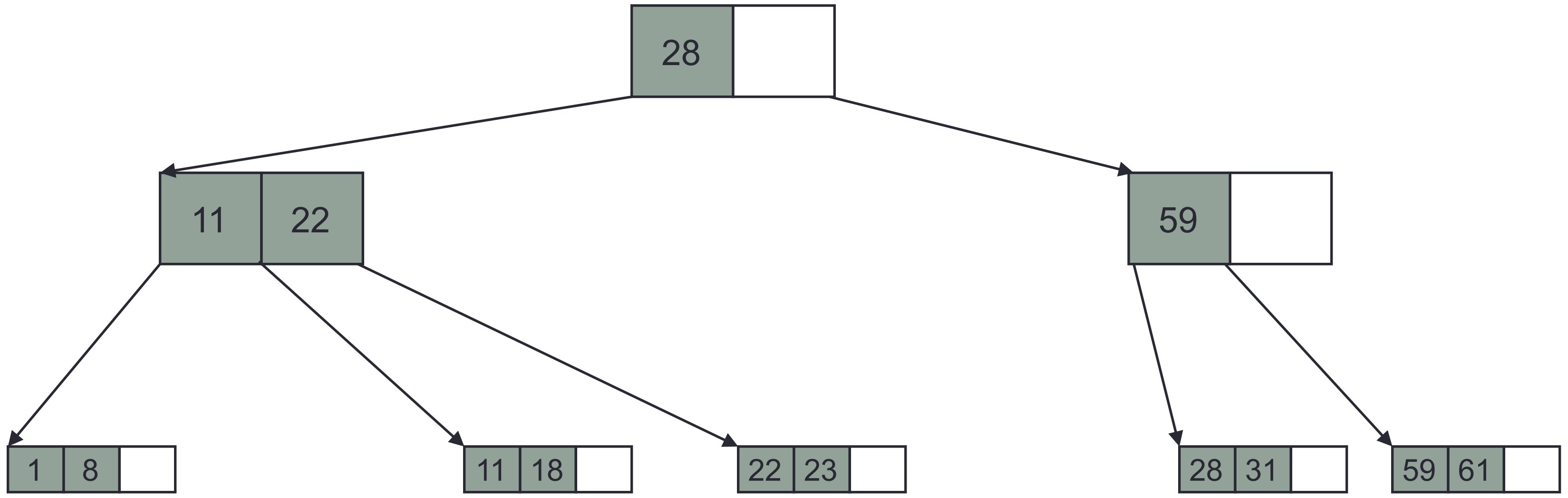
Delete 41



B+ tree of order 3 (M=3)

B+ Tree: Homework

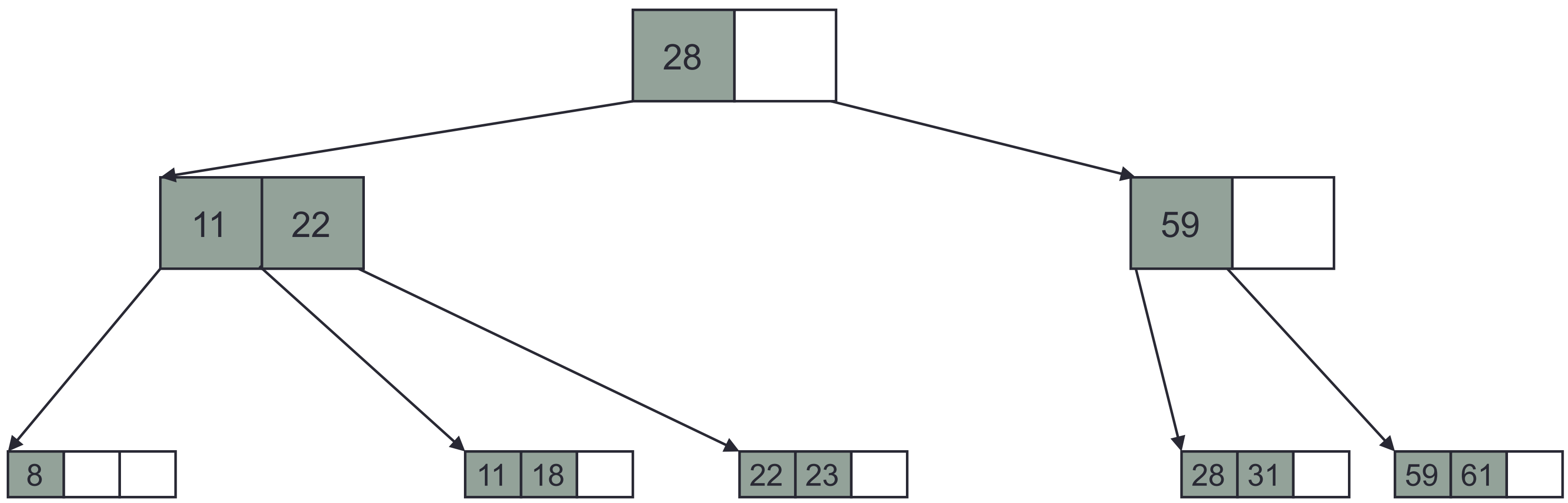
Delete 1



B+ tree of order 3 (M=3)

B+ Tree: Homework

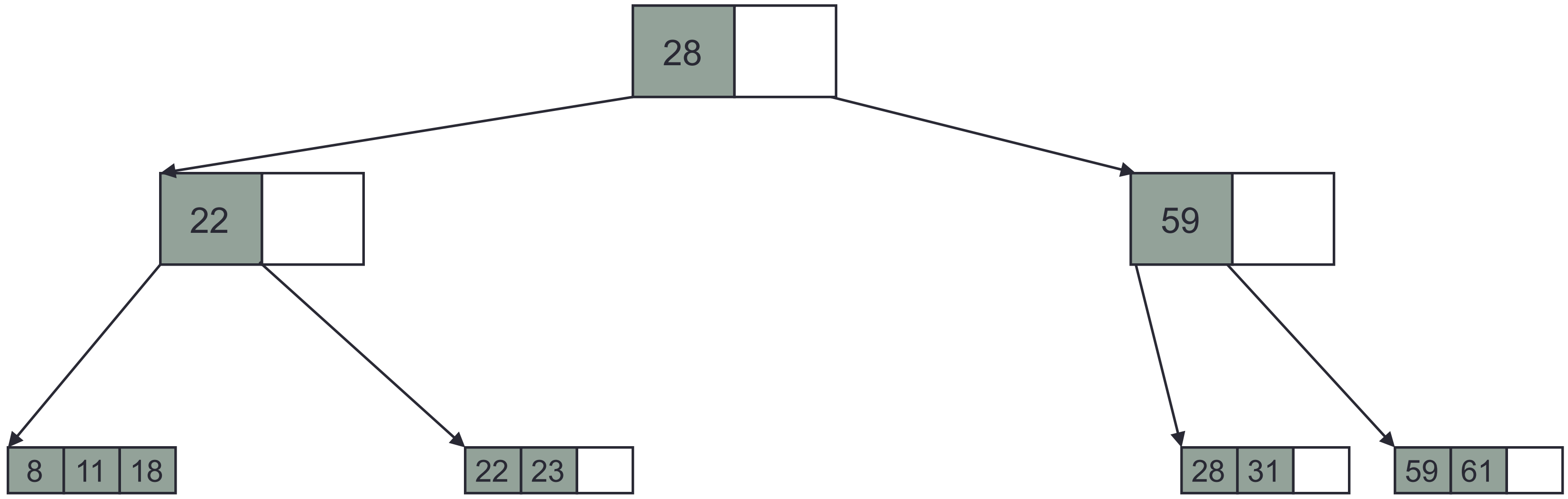
Delete 1 (Underflow)



B+ tree of order 3 (M=3)

B+ Tree: Homework

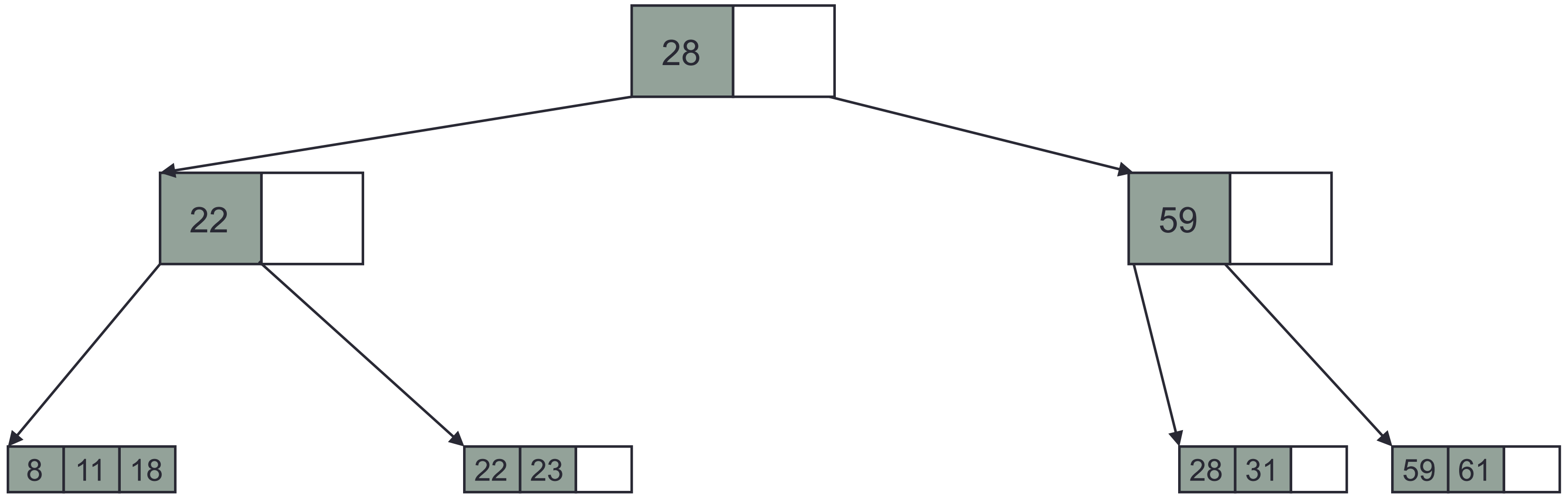
Delete 1 (Merge/Update)



B+ tree of order 3 (M=3)

B+ Tree: Homework

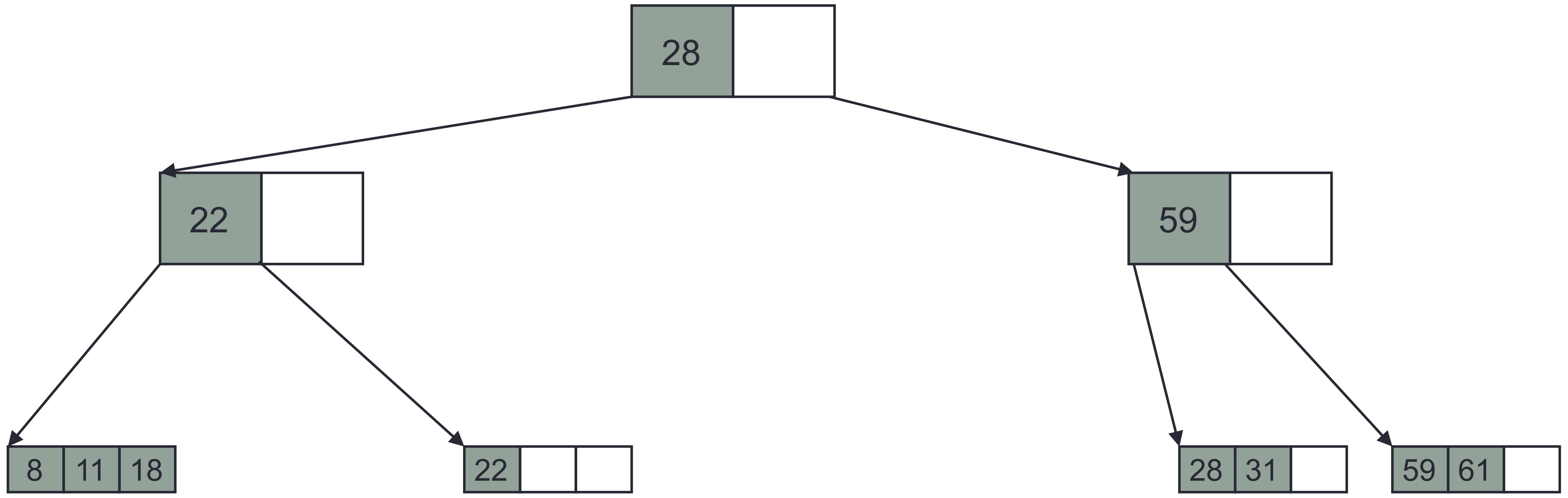
Delete 23



B+ tree of order 3 (M=3)

B+ Tree: Homework

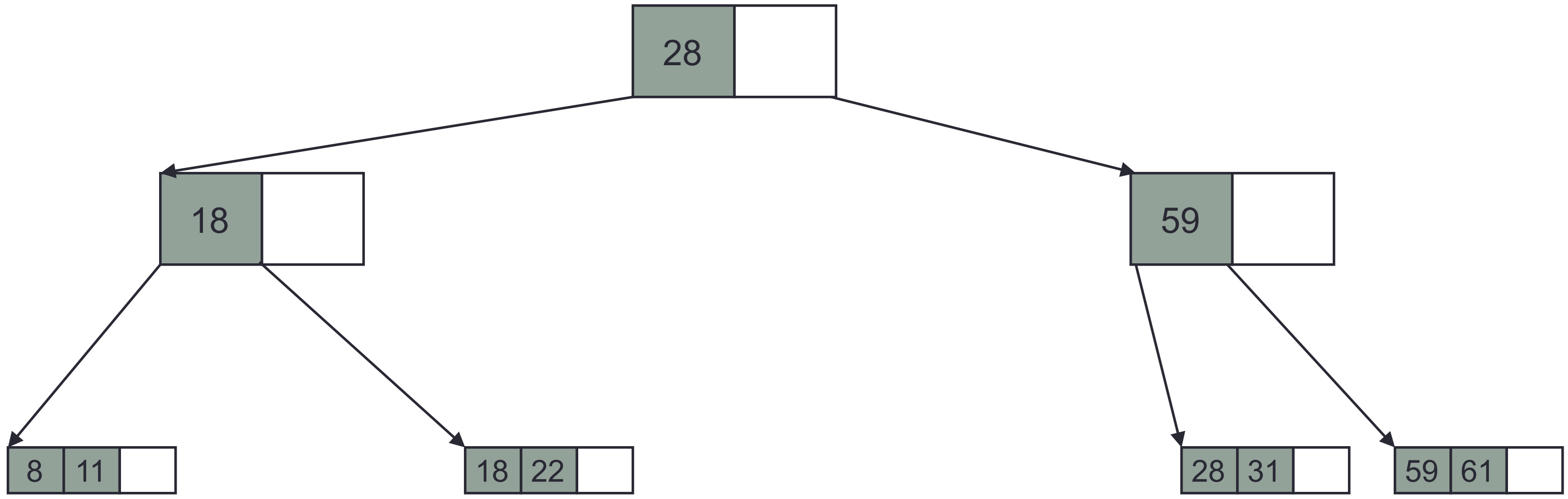
Delete 23 (Underflow)



B+ tree of order 3 (M=3)

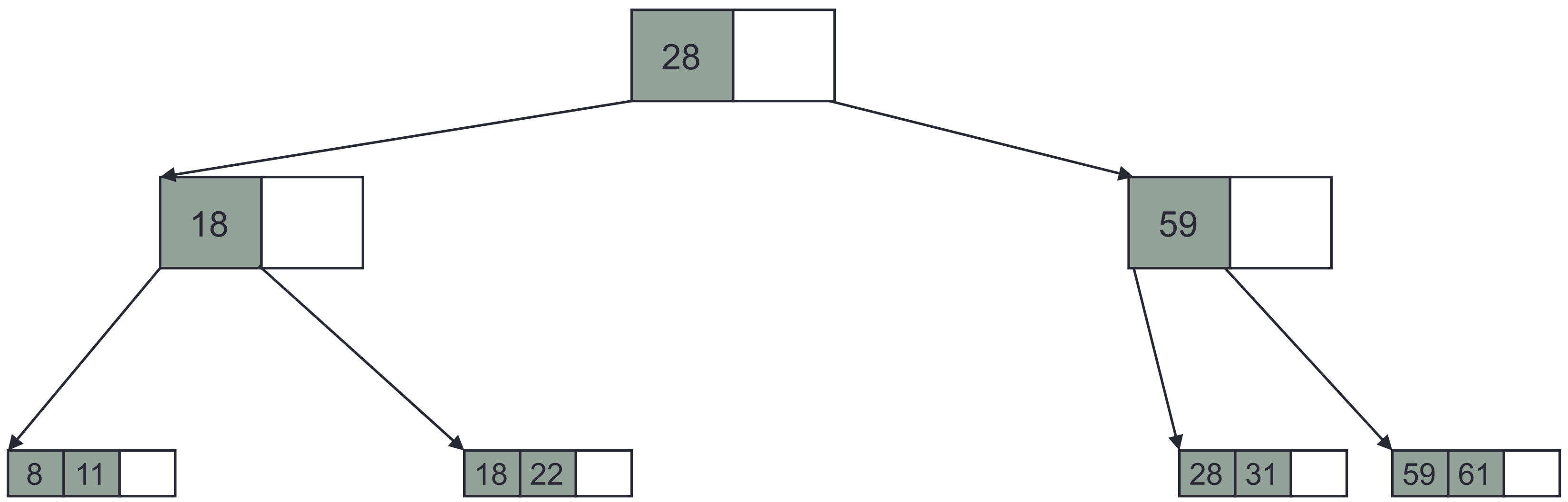
B+ Tree: Homework

Delete 23 (Borrow/Update)



B+ tree of order 3 (M=3)

B+ Tree: Homework



B+ tree of order 3 (M=3)