B Tree Data Structure

261217 Data Structures for Computer Engineers

Patiwet Wuttisarnwattana, Ph.D.

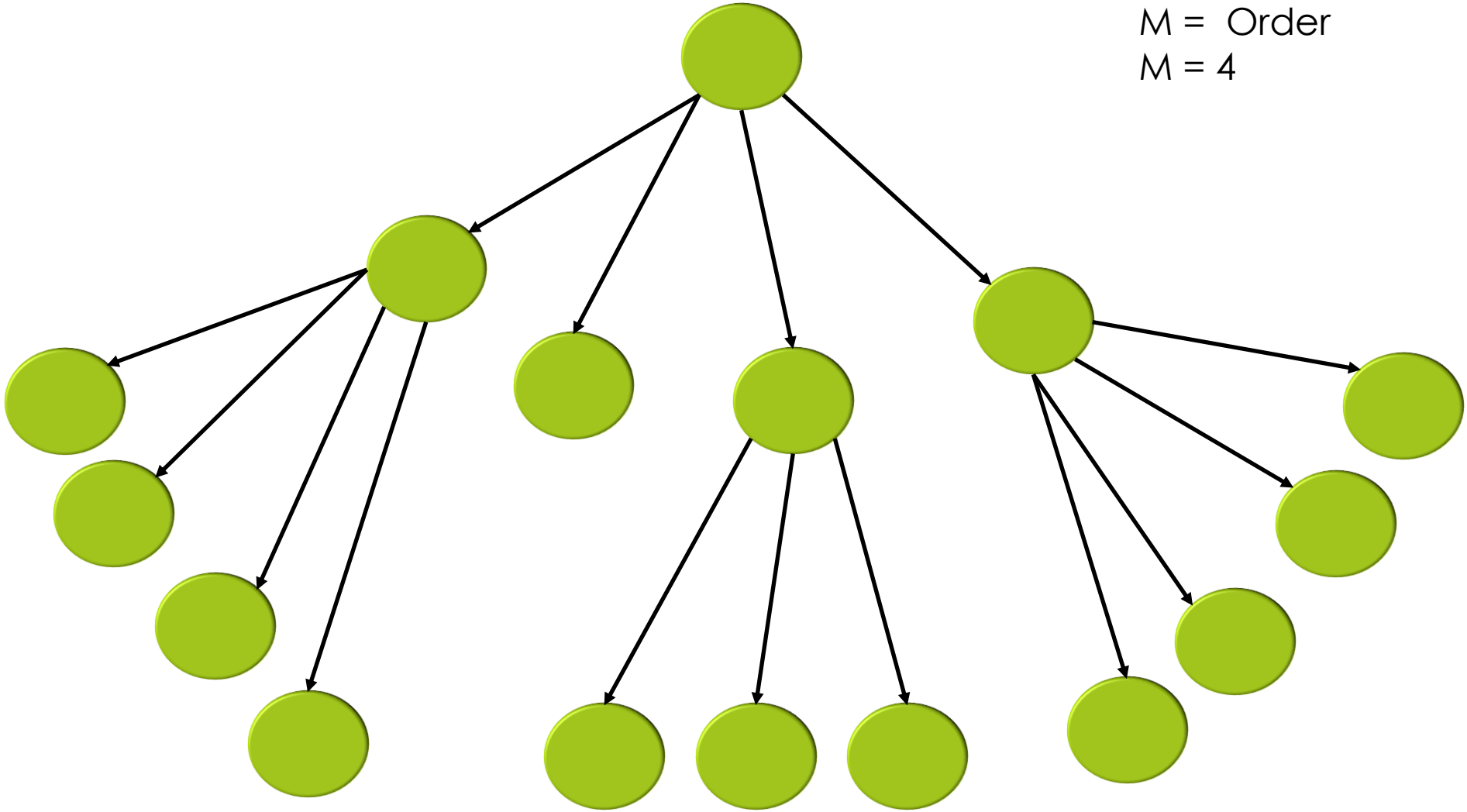
patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

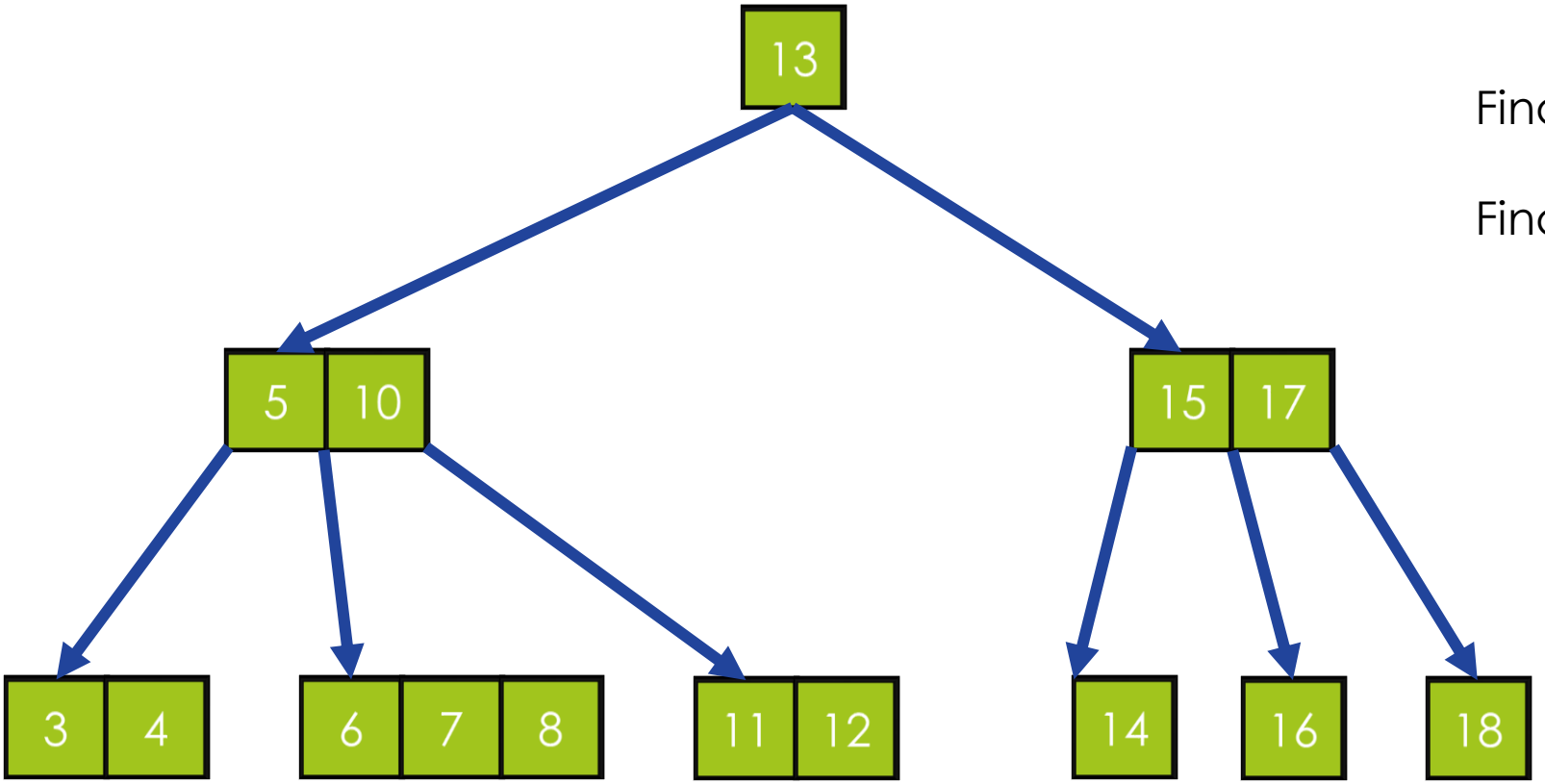
B-Tree

- B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
- The B-tree is a generalization of a binary search tree in that a node can have more than two children.
- B-tree is optimized for systems that read and write large blocks of data.
- B-trees are a good example of a data structure for external memory. It is commonly used in databases and filesystems.

M-way Tree (M-ary Tree)



B-Tree of order M (4)



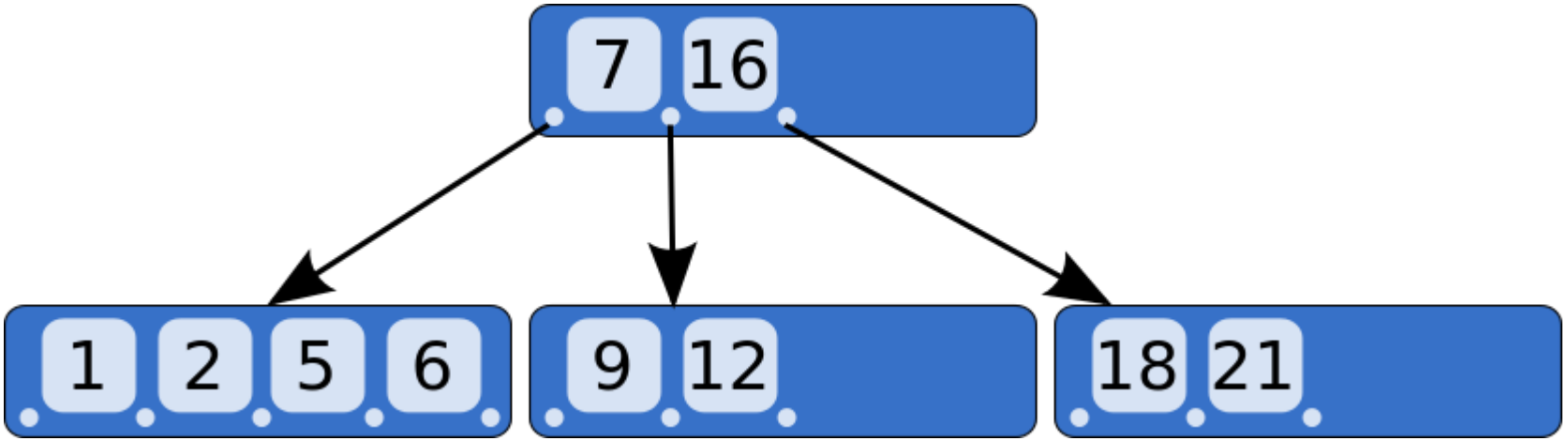
Find(16)

Find(8)

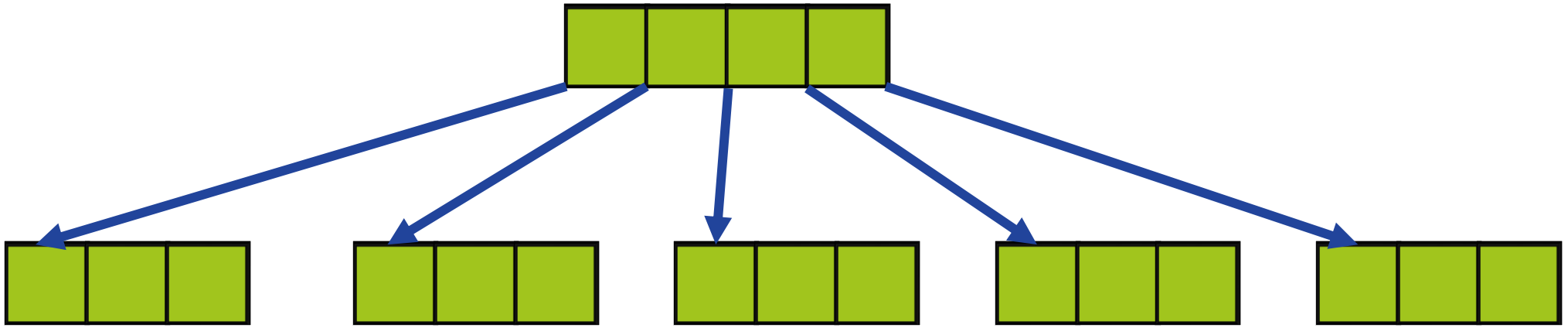
B Tree with order M

1. Every node has at most *M* children
2. Any node with *k* keys will have *k+1* children
3. The root node has at least two children unless it is a leaf
4. Internal nodes must have at least $\lceil M/2 \rceil$ children and $\lceil M/2 \rceil - 1$ keys
5. All leaves are on the same level
6. (Optional) Each leaf node should have at least $\lceil M/2 \rceil - 1$ keys

B-Tree of order 5



B-Tree



- All leaves are at the same level
- How many nodes?
- A node with k keys, how many children does it have?

B-Tree Insertion

1. Start from the root, Find the inserting location like the Binary Search Tree. The destination will be a leaf node
2. Check if the insertion causes Overflow:
    1. If the node is not full (full if M-1 keys), then orderly insert the new key into the node
    2. If the node is full,
        - temporarily insert the new key into the node
        - Use the middle key as the pivot point
        - Split the node using the pivot point
        - Recursively insert the pivot key to the parent node

Example: Construction of B-Tree

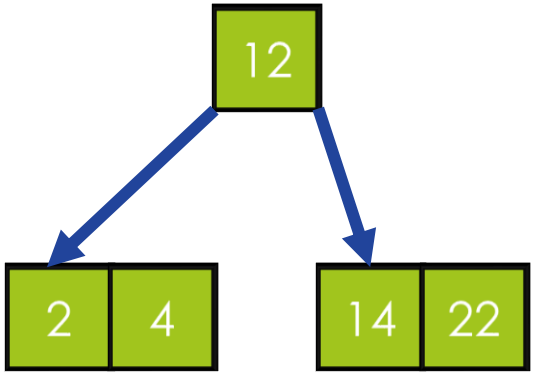
- Insert the following keys into a B-Tree of order 5
  - A node can have at most 5 children
  - A node can have at most 4 keys
  - An internal nodes (and leaves) must have at least 2 keys (3 children)
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32

Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32

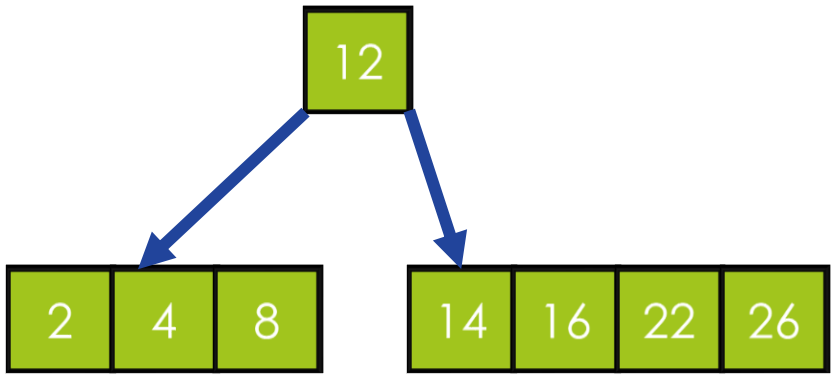
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



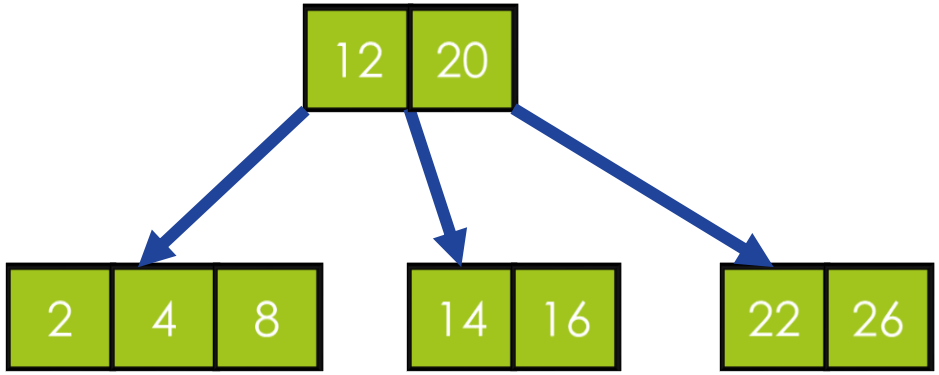
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



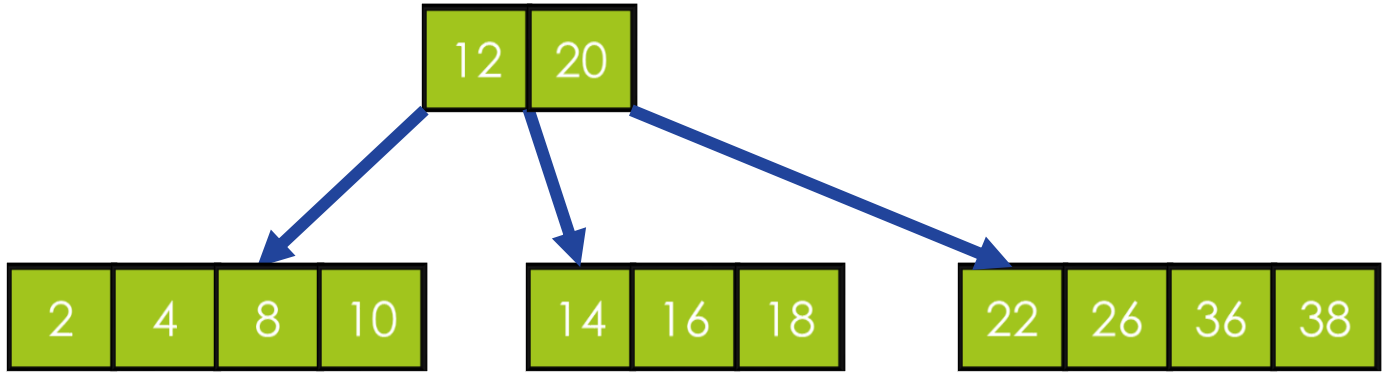
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



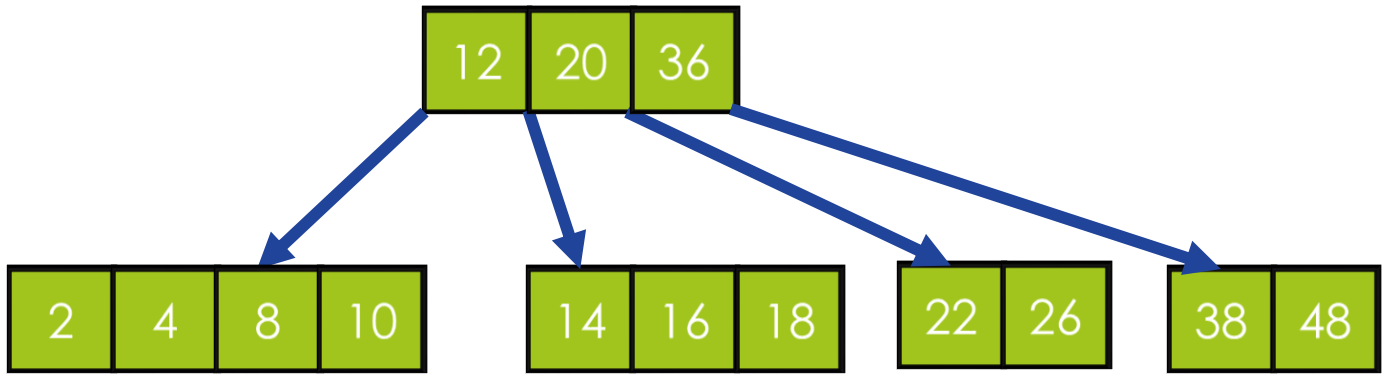
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



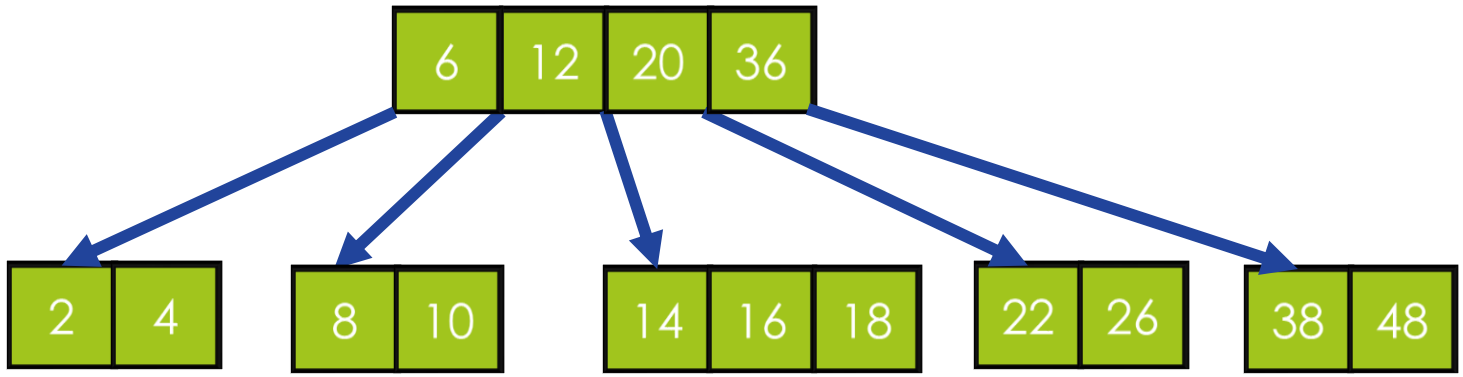
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



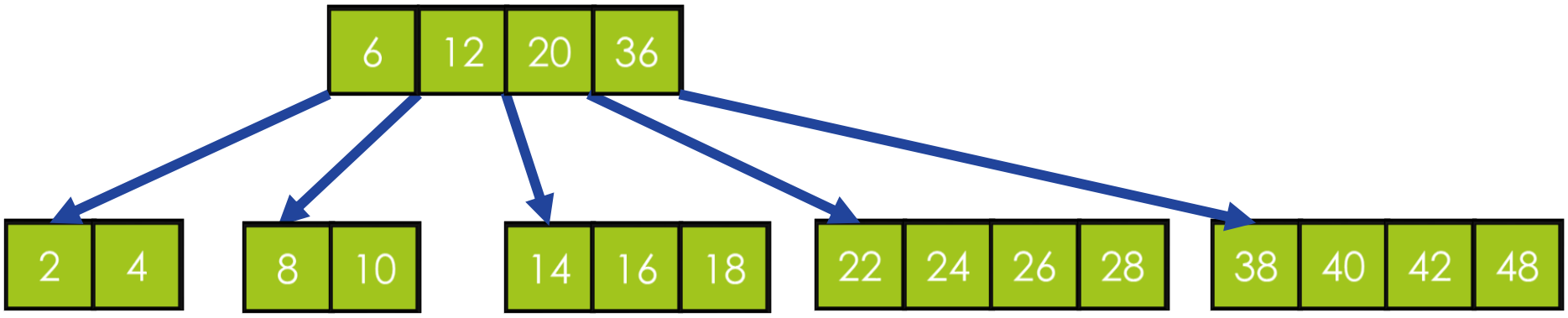
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



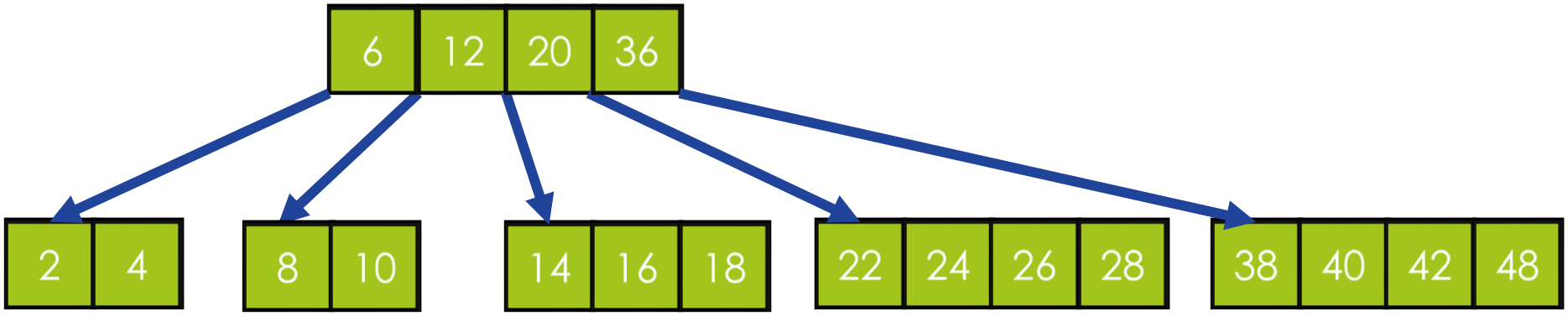
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



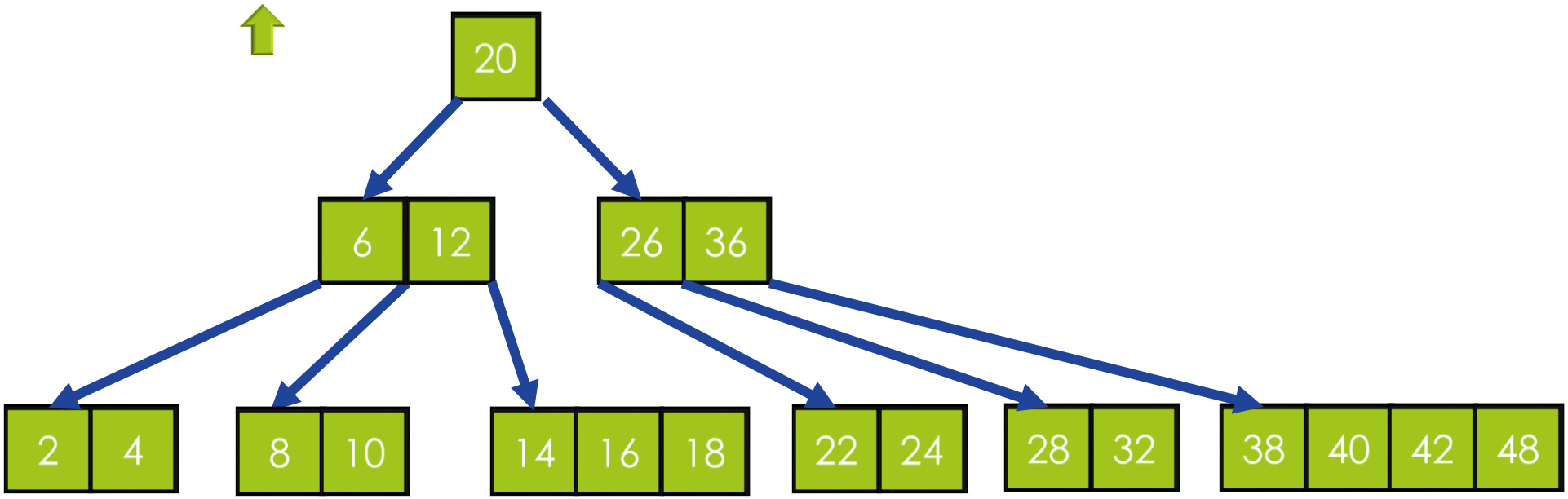
Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



Example: Construction of B-Tree

- Insert the following keys into a B-Tree
- 2, 14, 12, 4, 22, 8, 16, 26, 20, 10, 38, 18, 36, 48, 6, 24, 28, 40, 42, 32



B-Tree Deletion

1. Start from the root, Find the deleting location like the Binary Search Tree. The destination will be in any node.

2. If the key is in leave nodes, if yes, the key can be deleted. Rebalance the node if it is Underflow

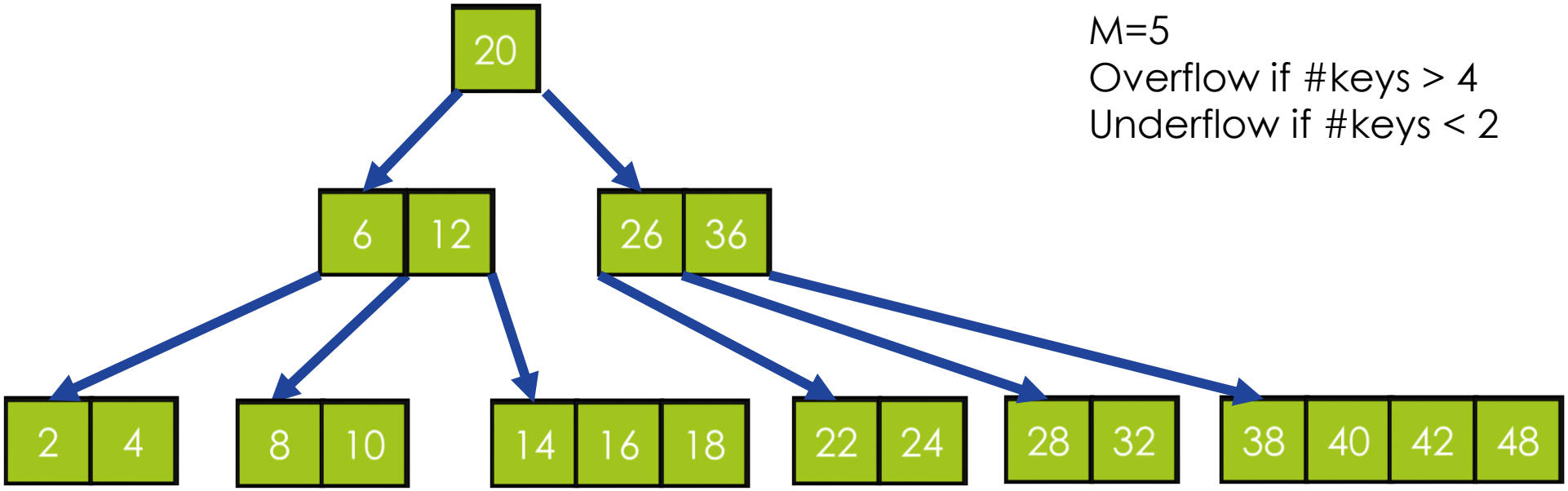
3. If the key is the internal nodes, replace the key with the min key from the right subtree. Recursively, delete the min key from the right subtree. Rebalance the right subtree if Underflow.
   - You may use max from left sub-tree, but this depends on your application

Underflow Correction

- An underflow node is a node that has number of keys below the minimum
  - Minimum at $\lceil M/2 \rceil - 1$ keys
- Check the right sibling first, if it is non-minimal node,
  - Rotate the minimum key of the right sibling node to the left
  - Rotation = Bring the parent key down + push the min key up
- If no, check the left sibling next, if it is a non-minimal node,
  - Rotate the maximum key of the left sibling node to the right
  - Rotation = Bring the parent key down + push the max key up
- If both sibling are at the minimal,
  - Merge the current node with the one of the siblings (right first before left) using the parent key as the pivot
  - Then recursively delete the parent key

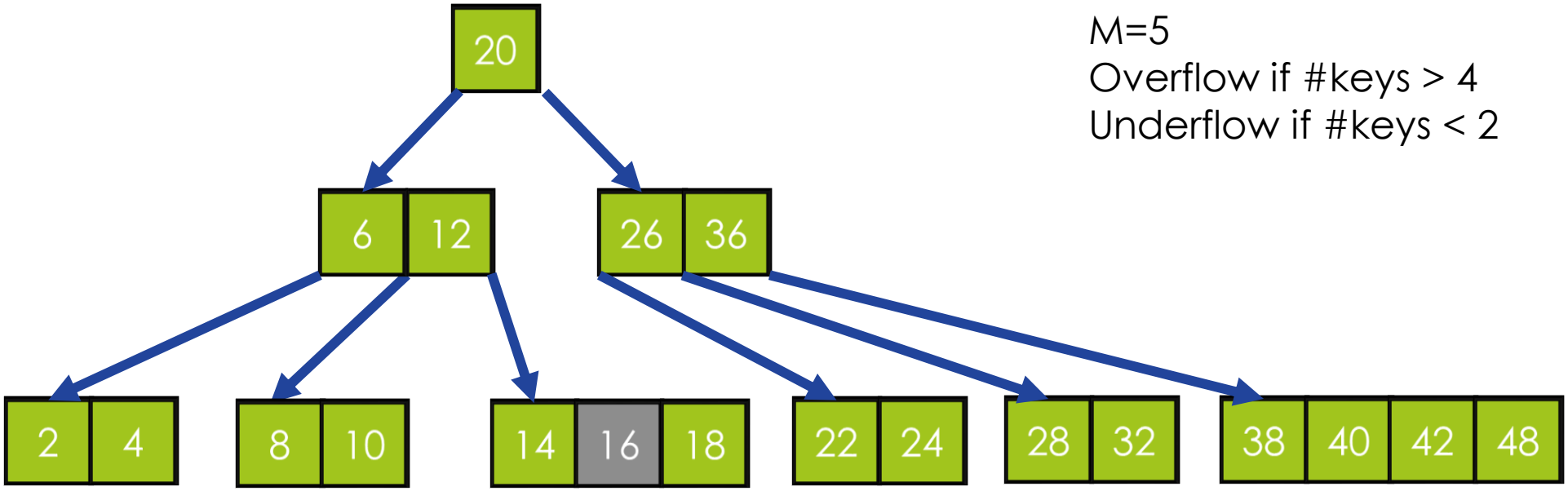
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(16)



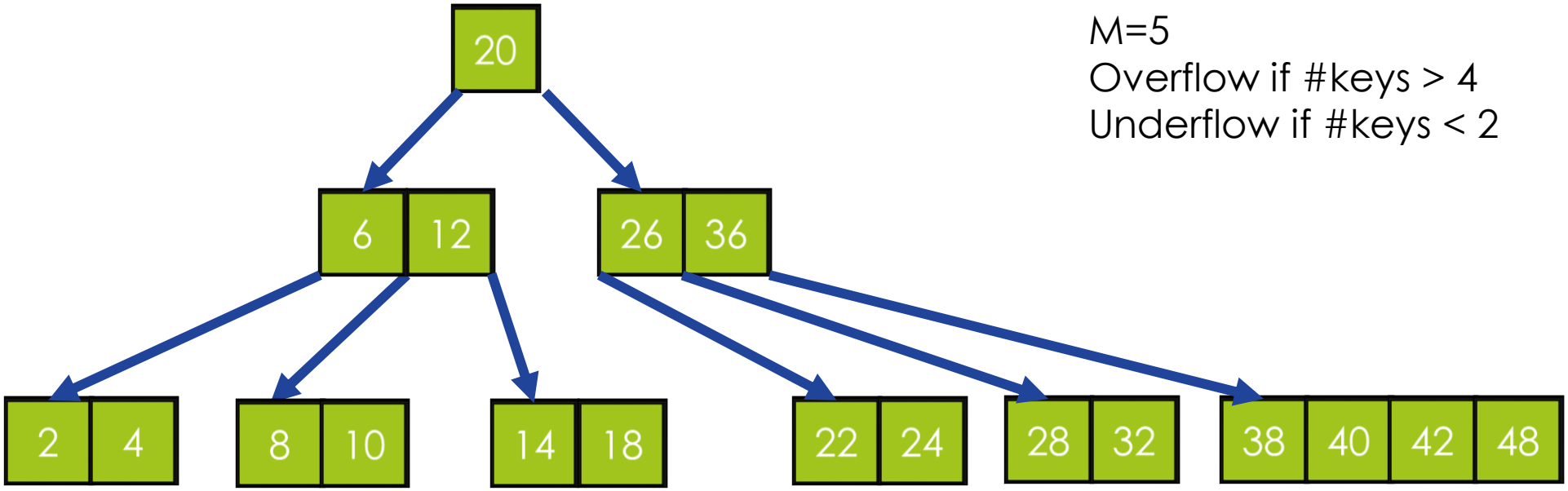
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(16)



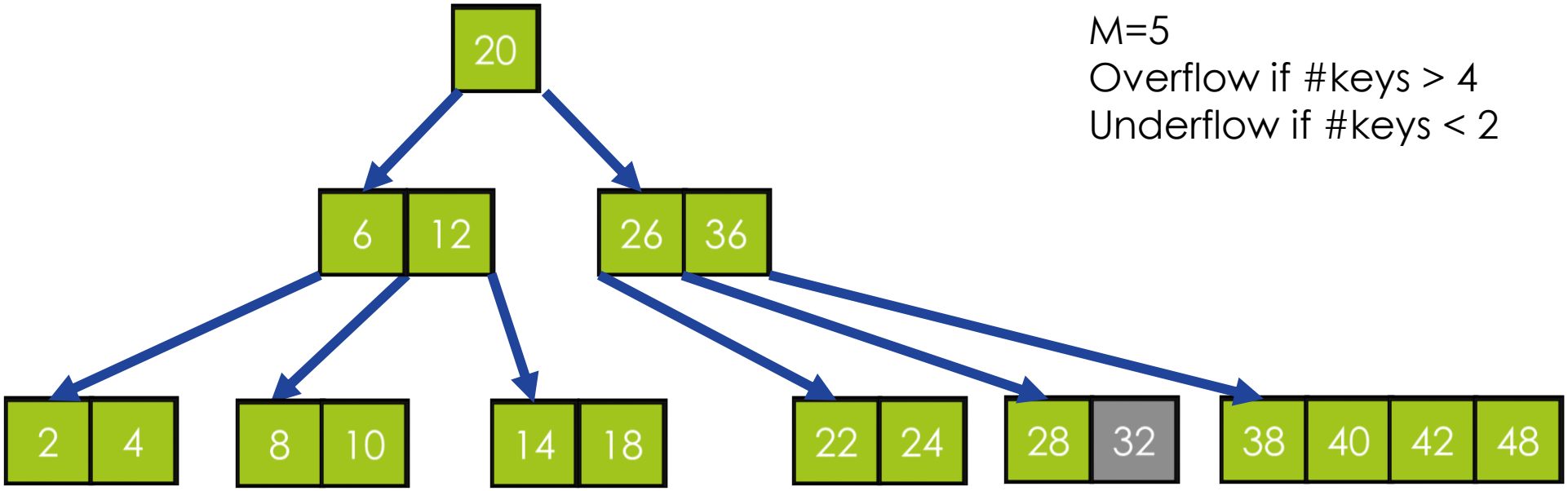
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(32)



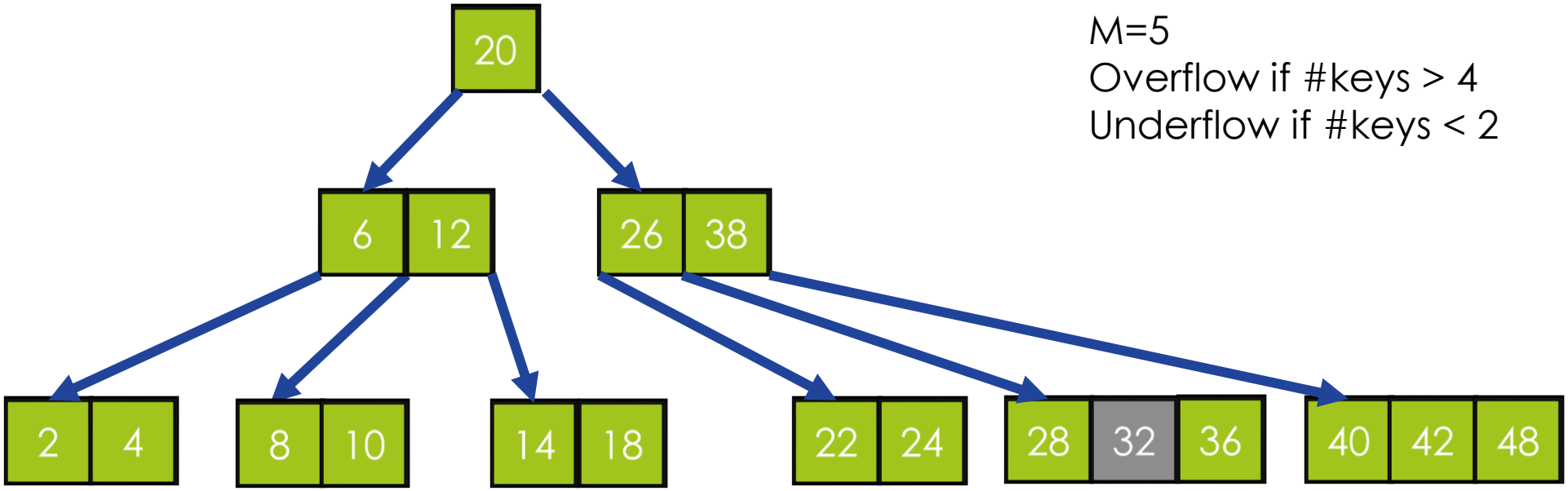
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(32)



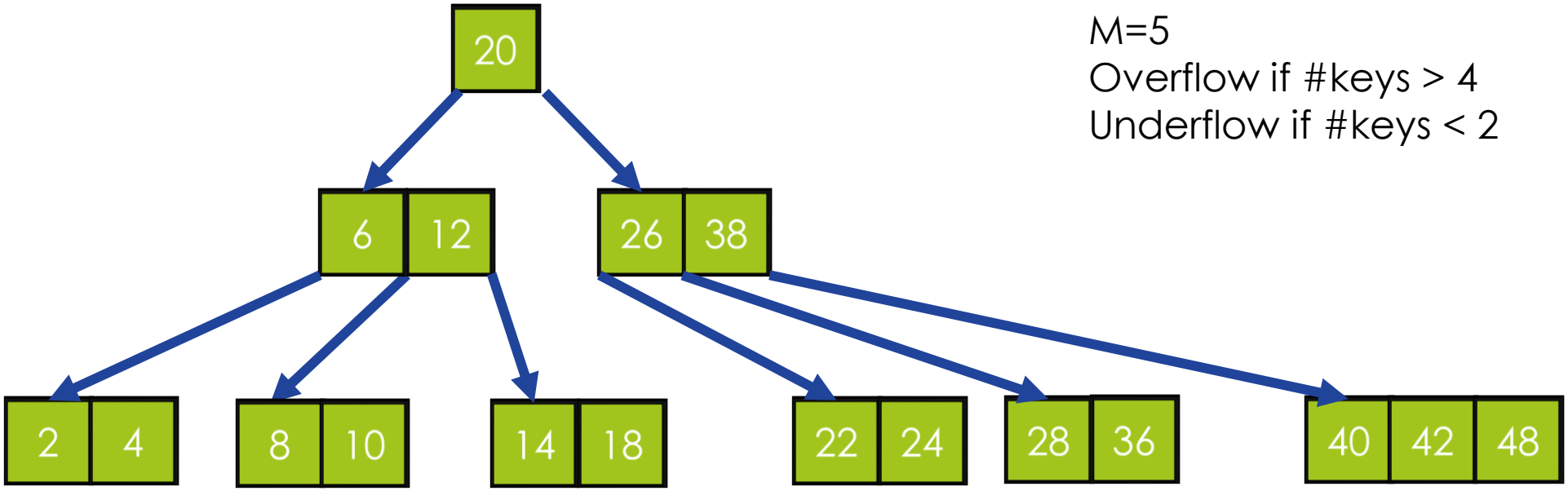
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(32)



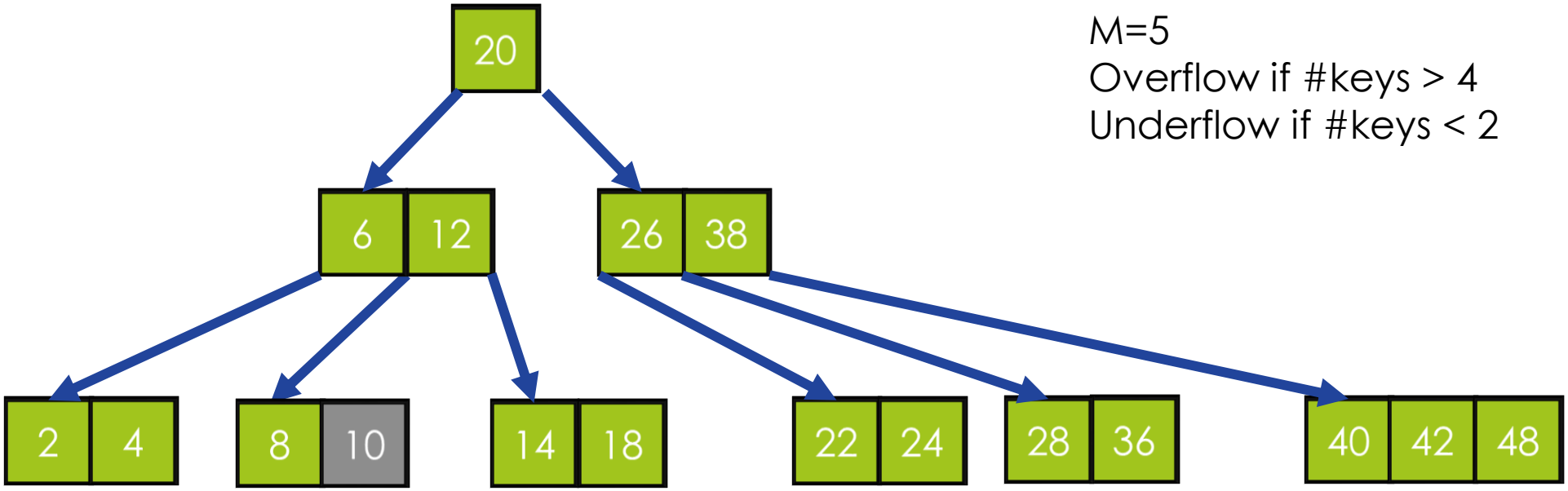
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(10)



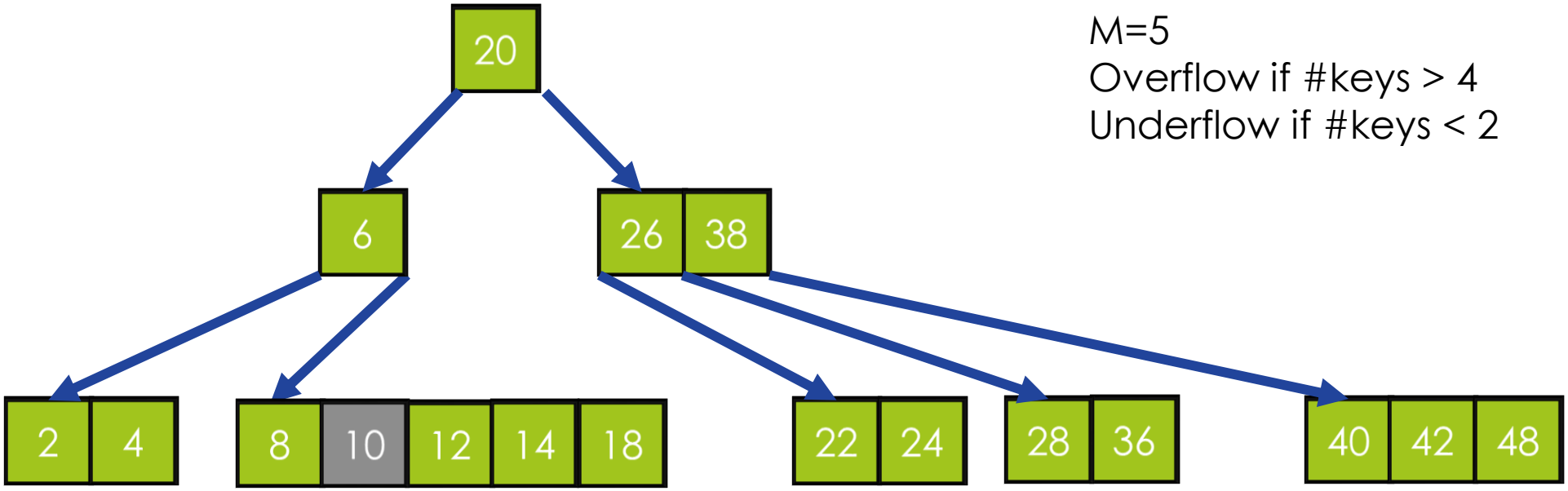
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(10)



Example: Deletion in B-Tree

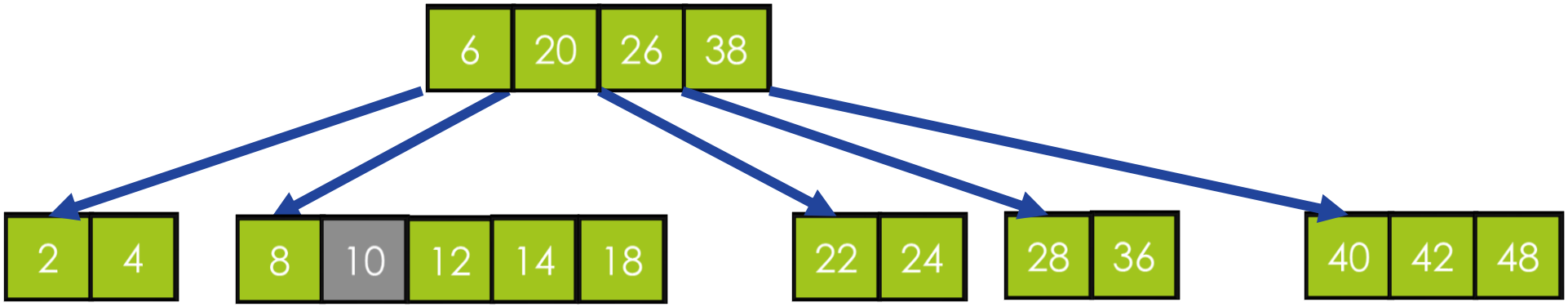
- Delete the following keys from the B-Tree
- Delete(10)



Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(10)

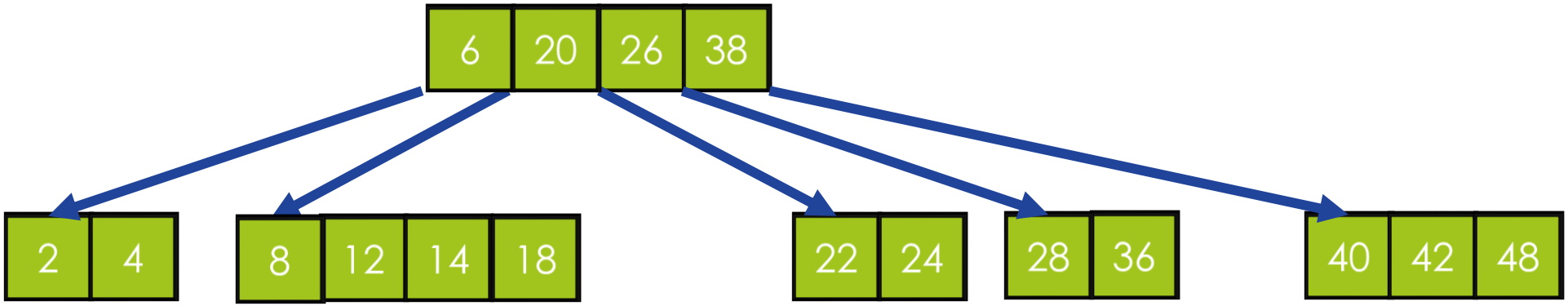
M=5
Overflow if #keys > 4
Underflow if #keys < 2



Example: Deletion in B-Tree

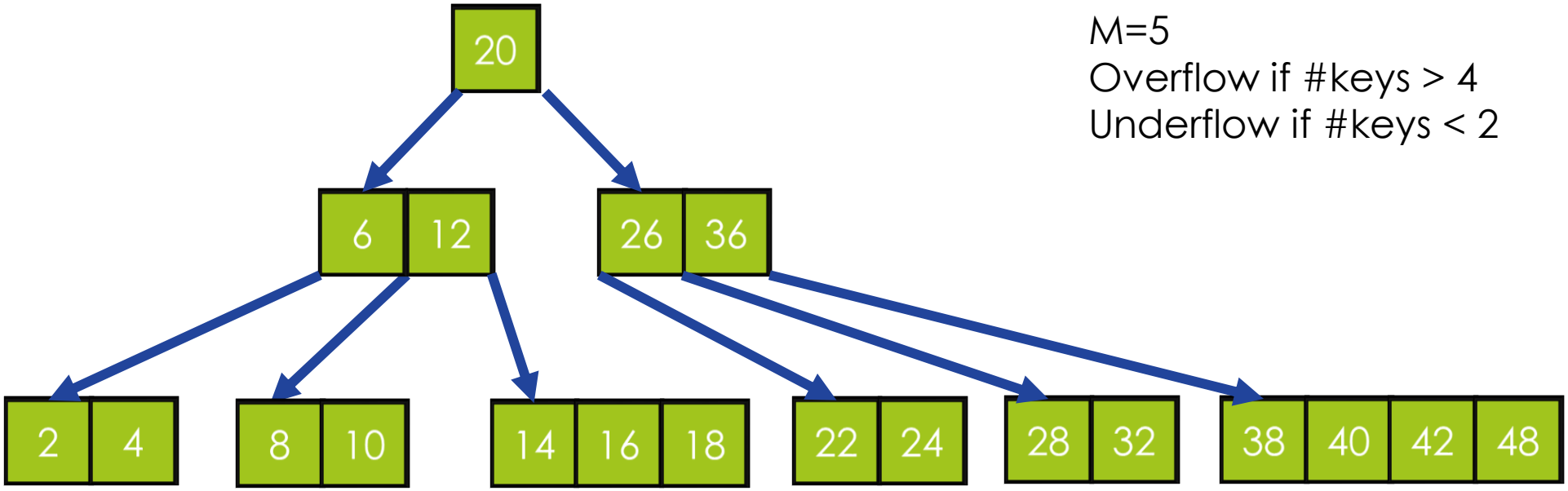
- Delete the following keys from the B-Tree
- Delete(10)

M=5
Overflow if #keys > 4
Underflow if #keys < 2



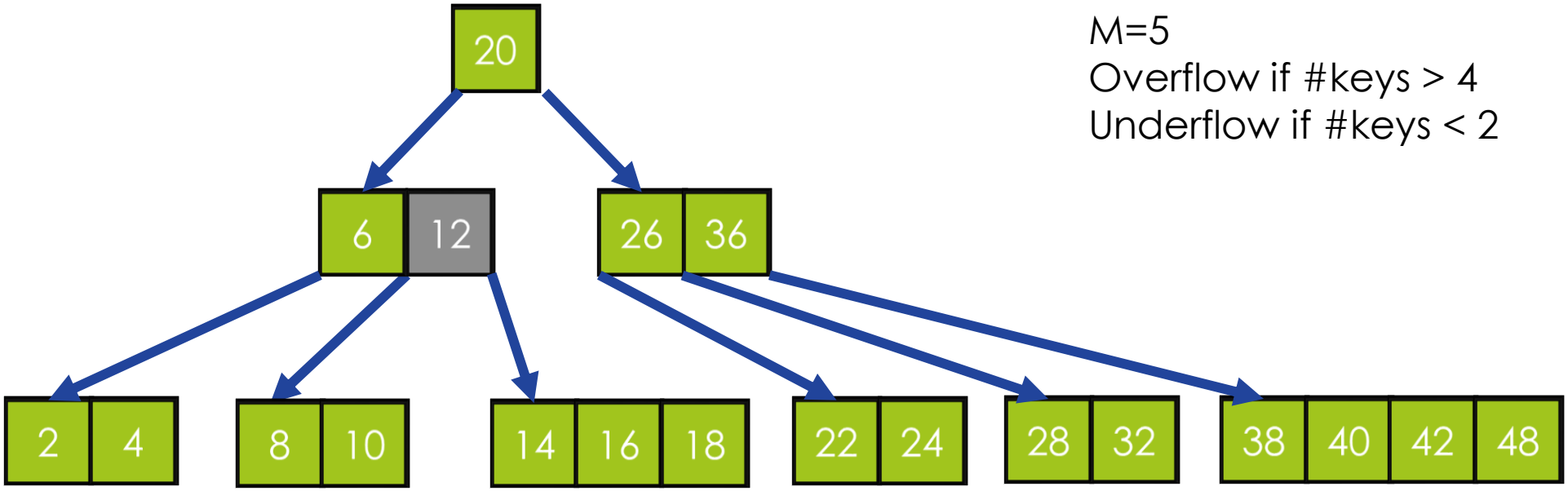
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(12)



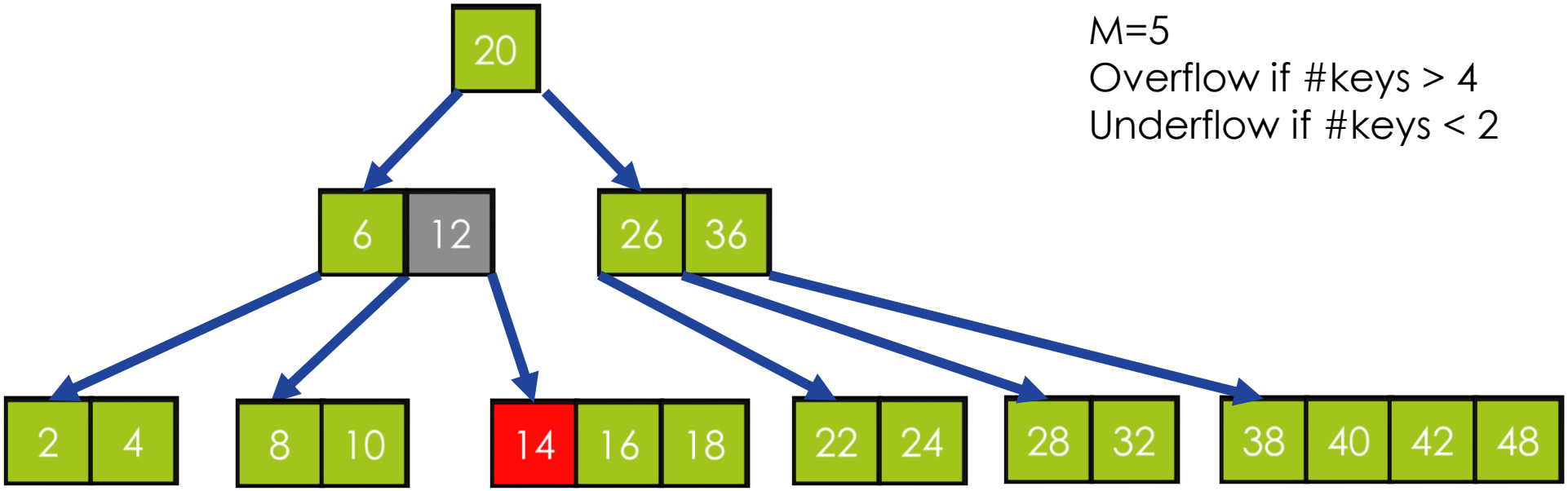
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(12)



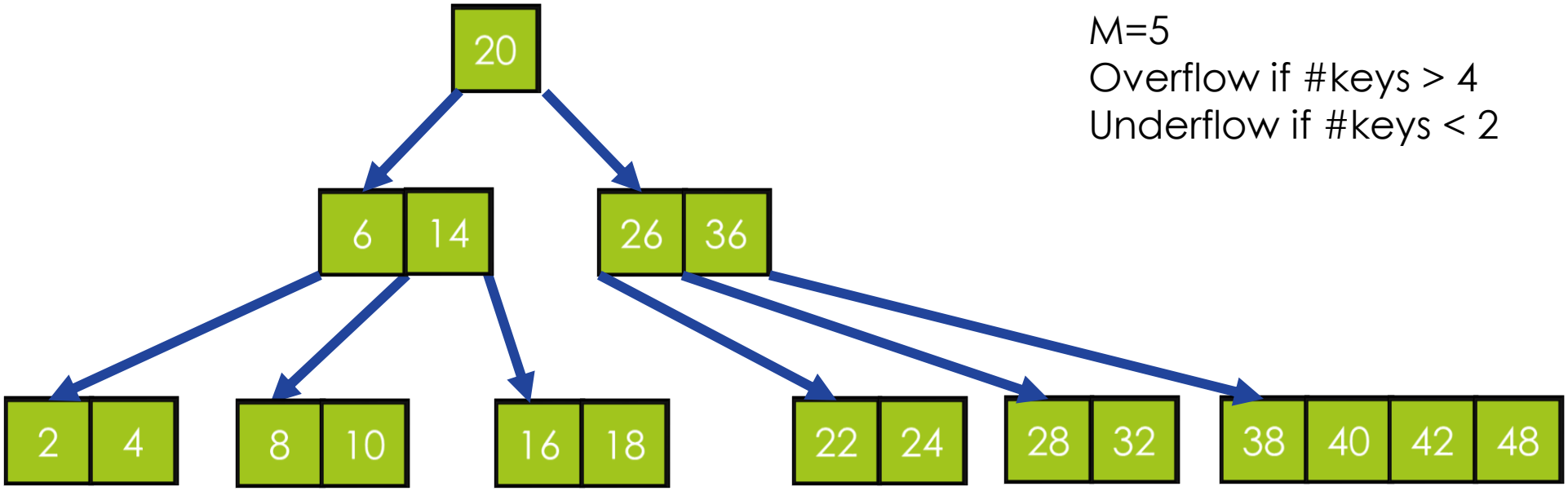
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(12)



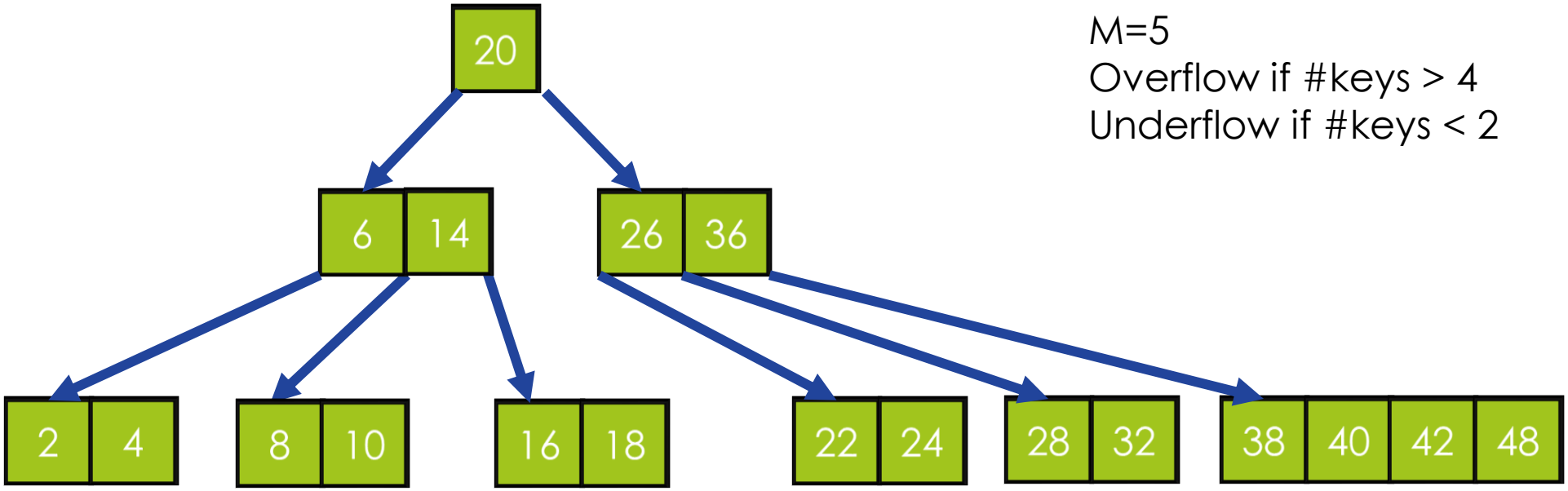
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(12)



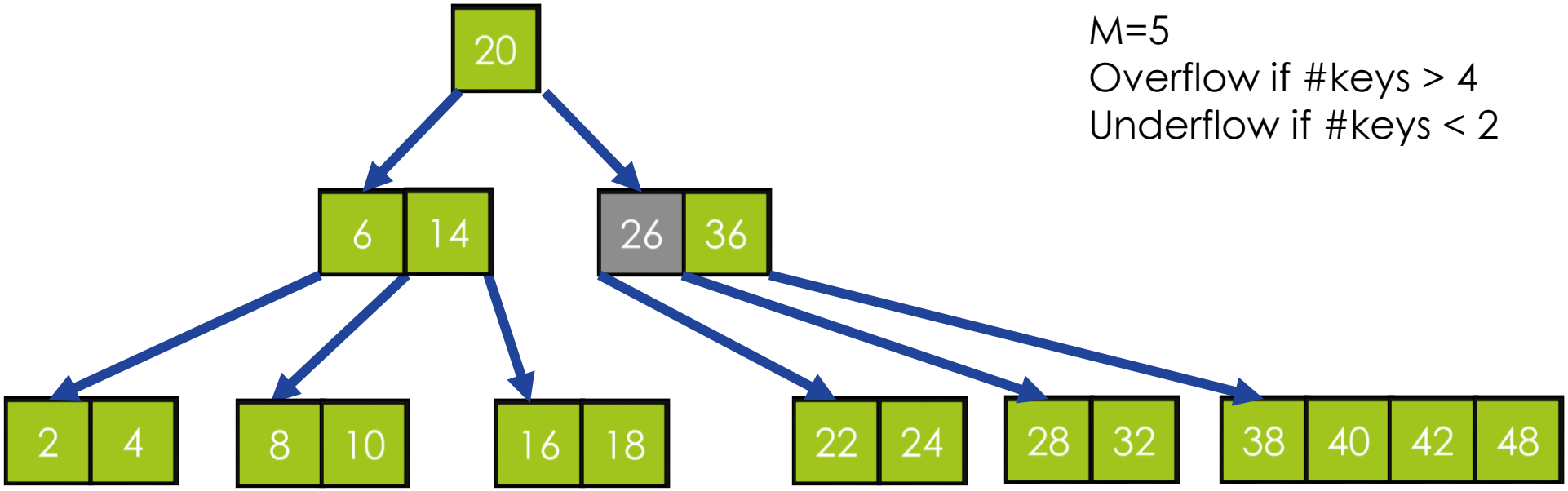
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



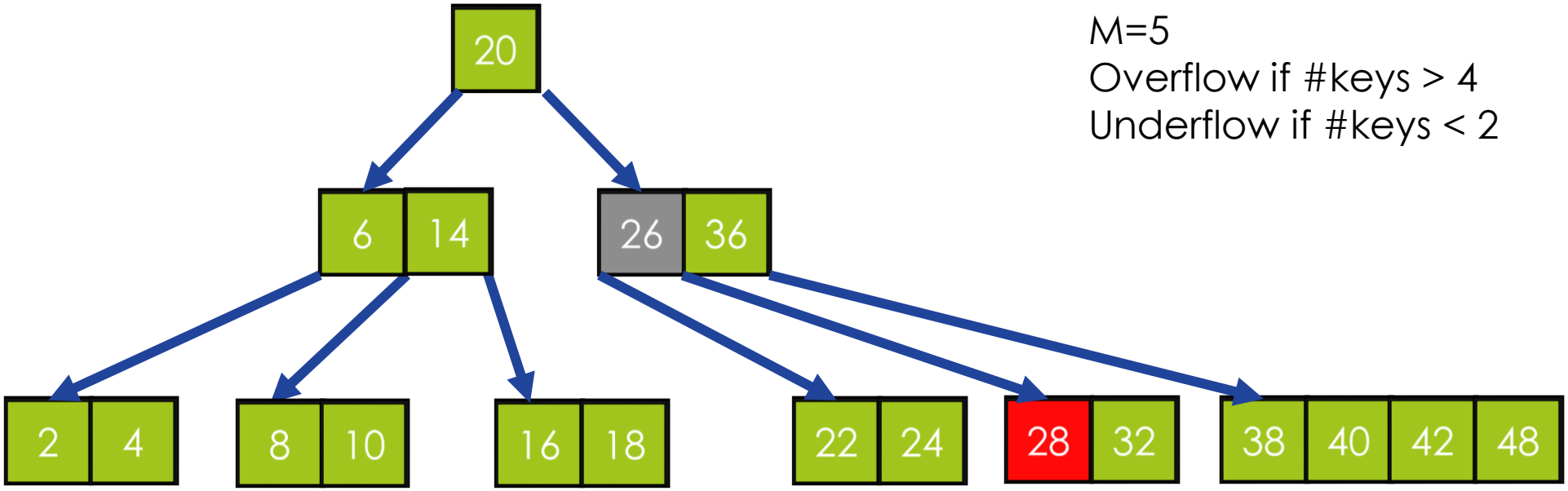
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



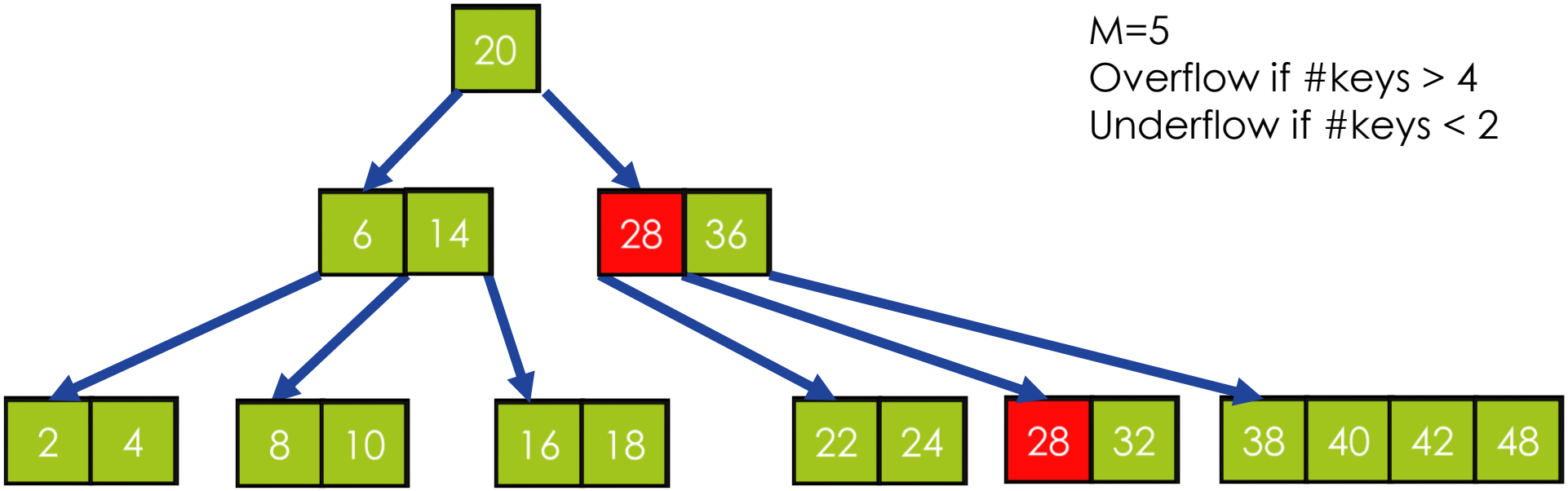
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



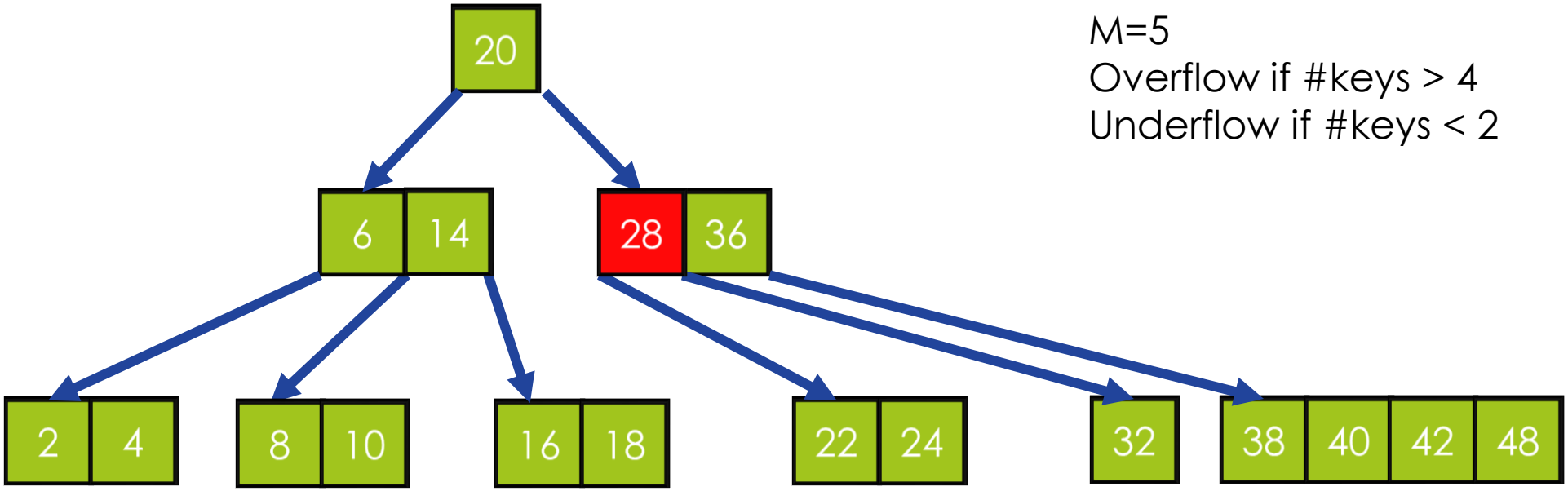
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



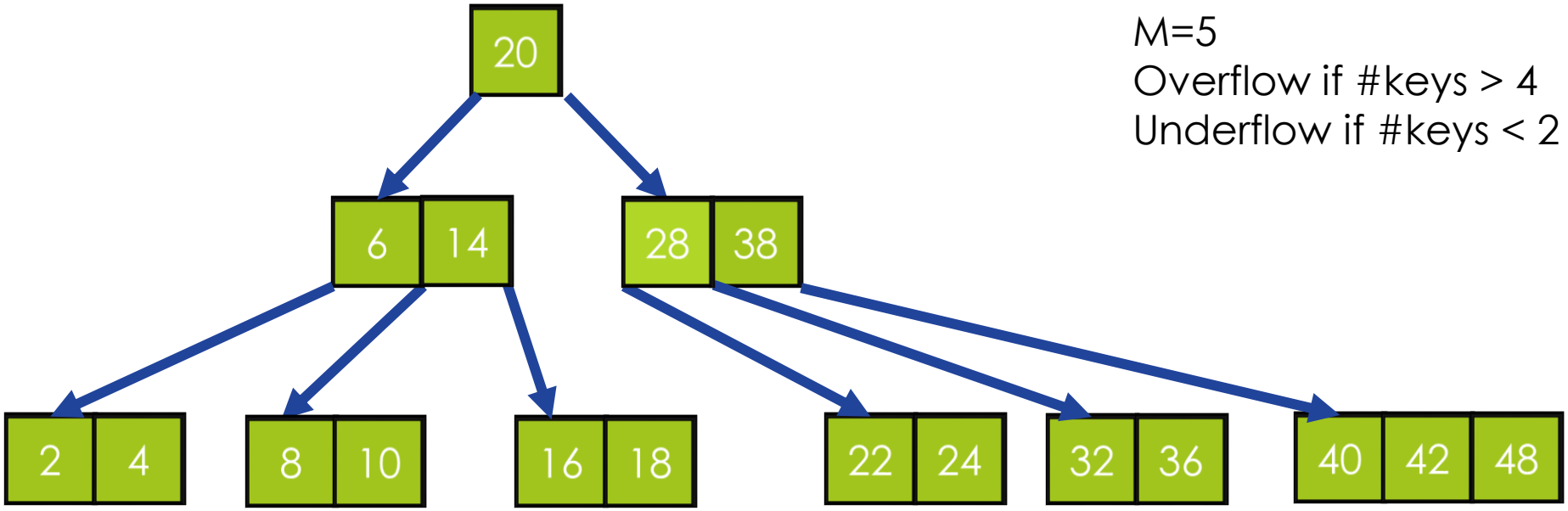
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



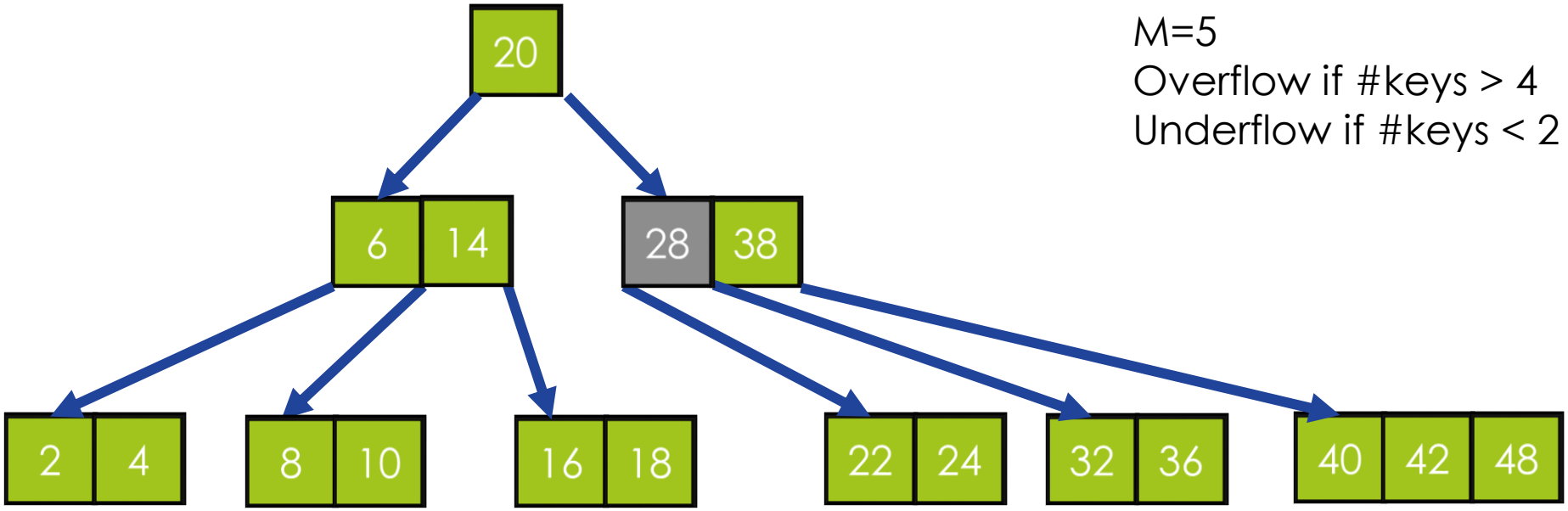
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(26)



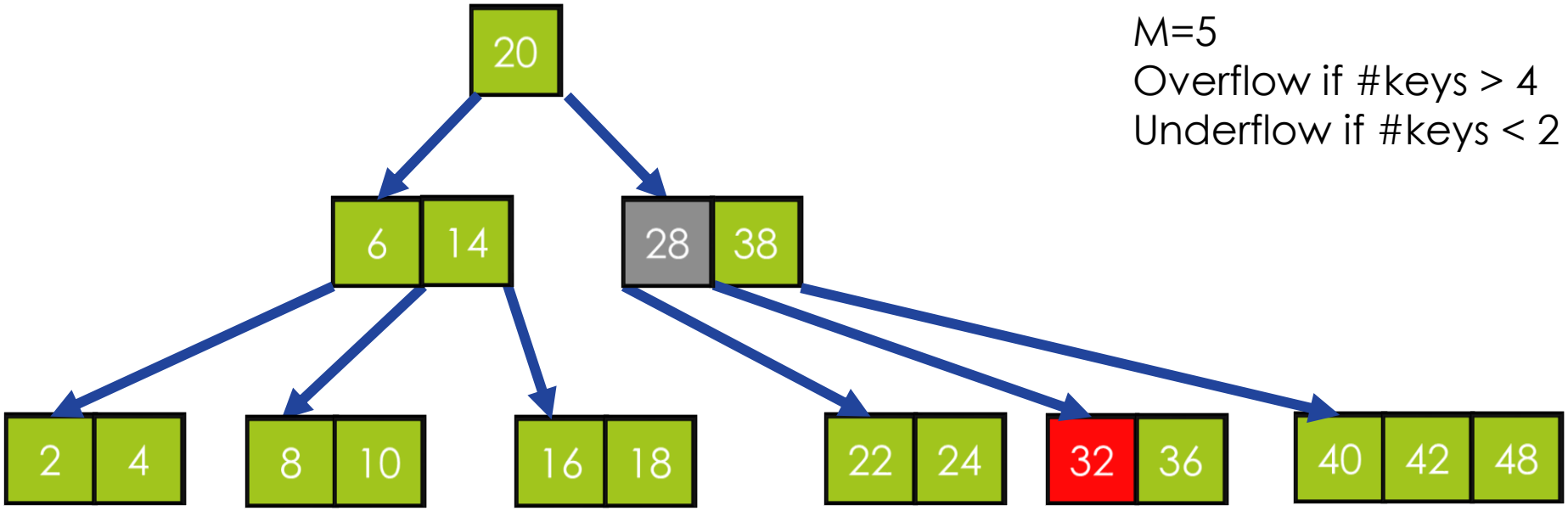
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(28)



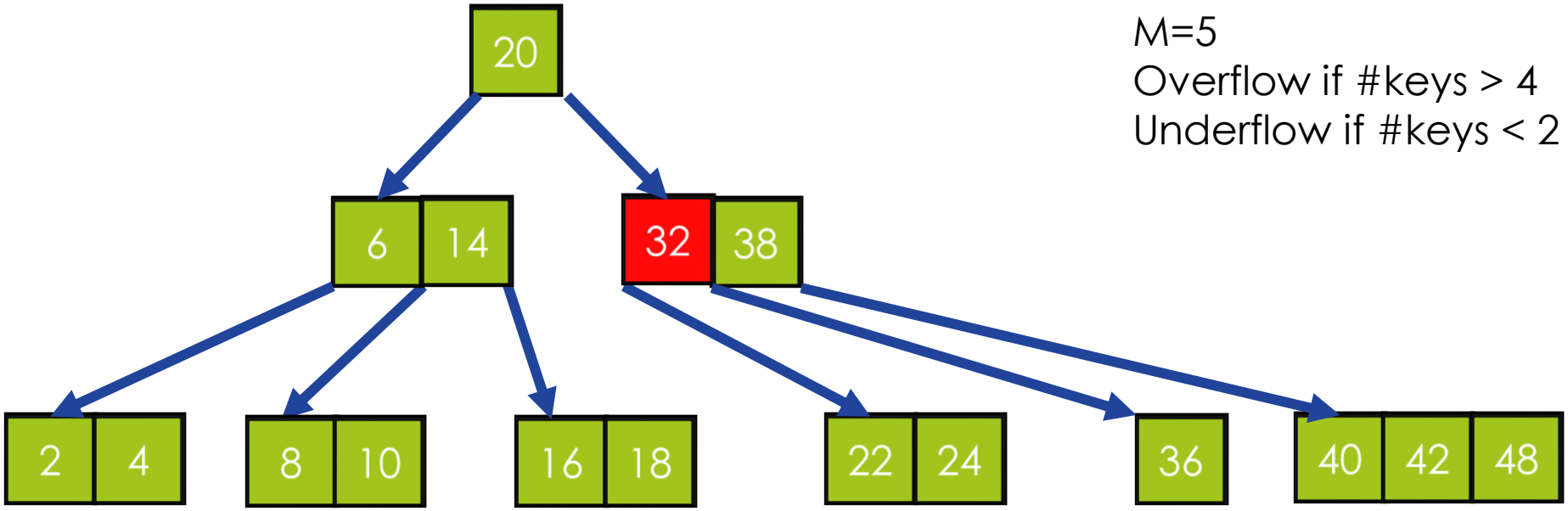
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(28)



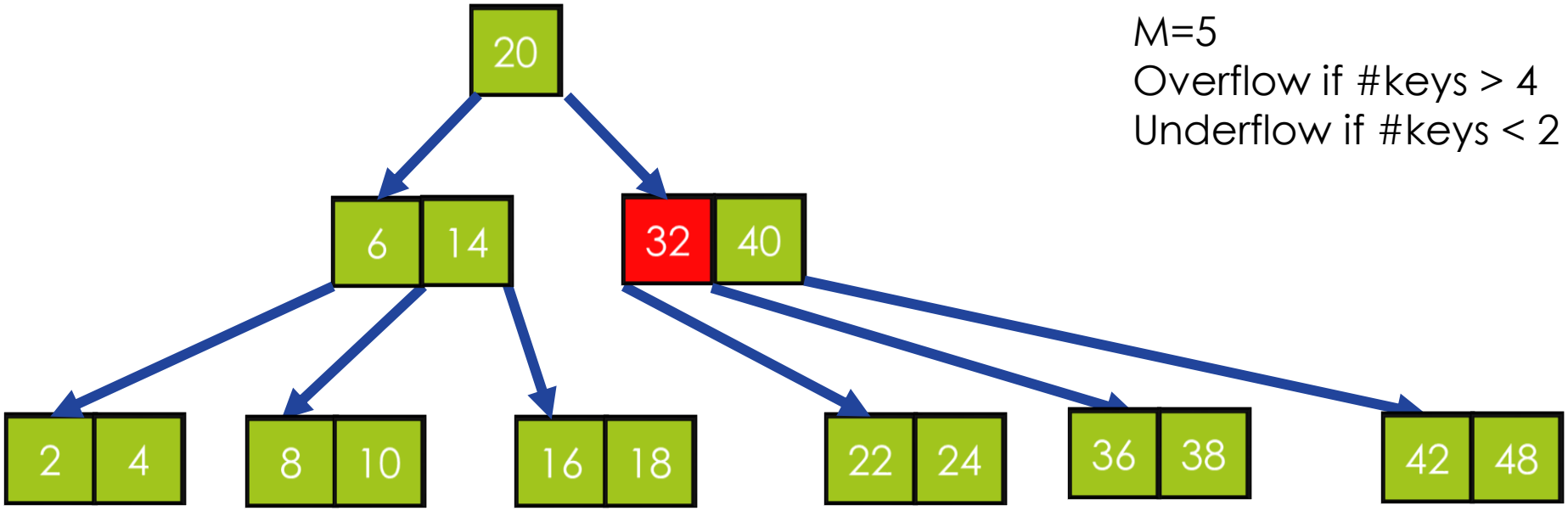
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(28)



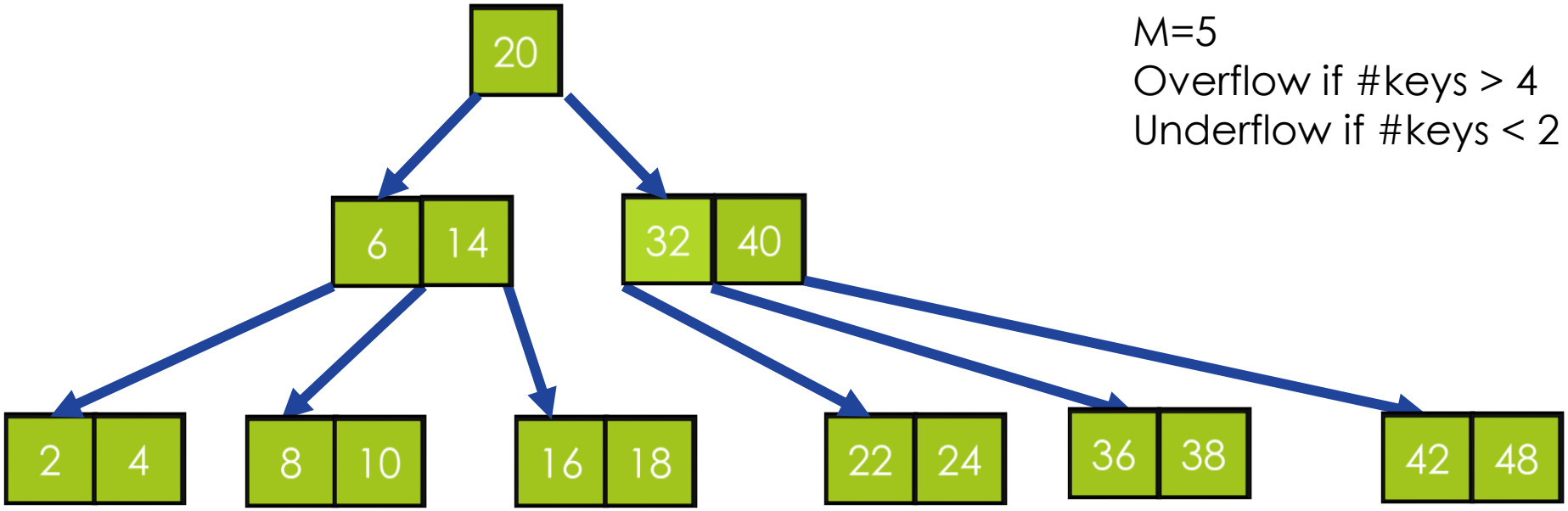
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(28)



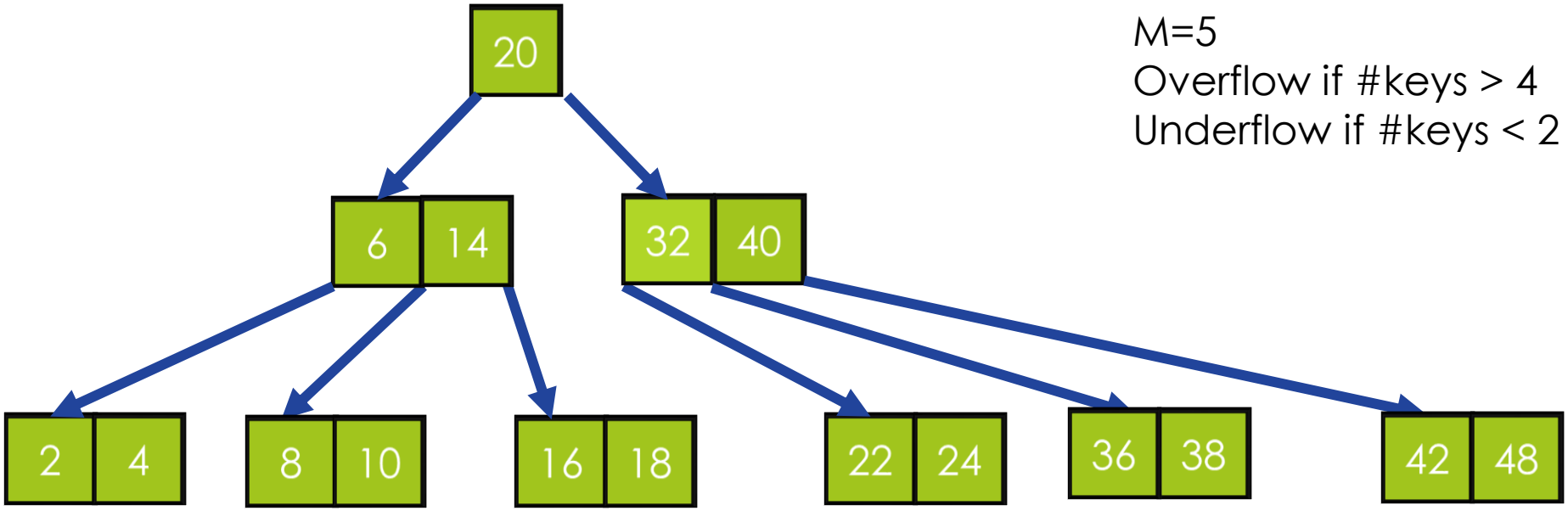
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(28)



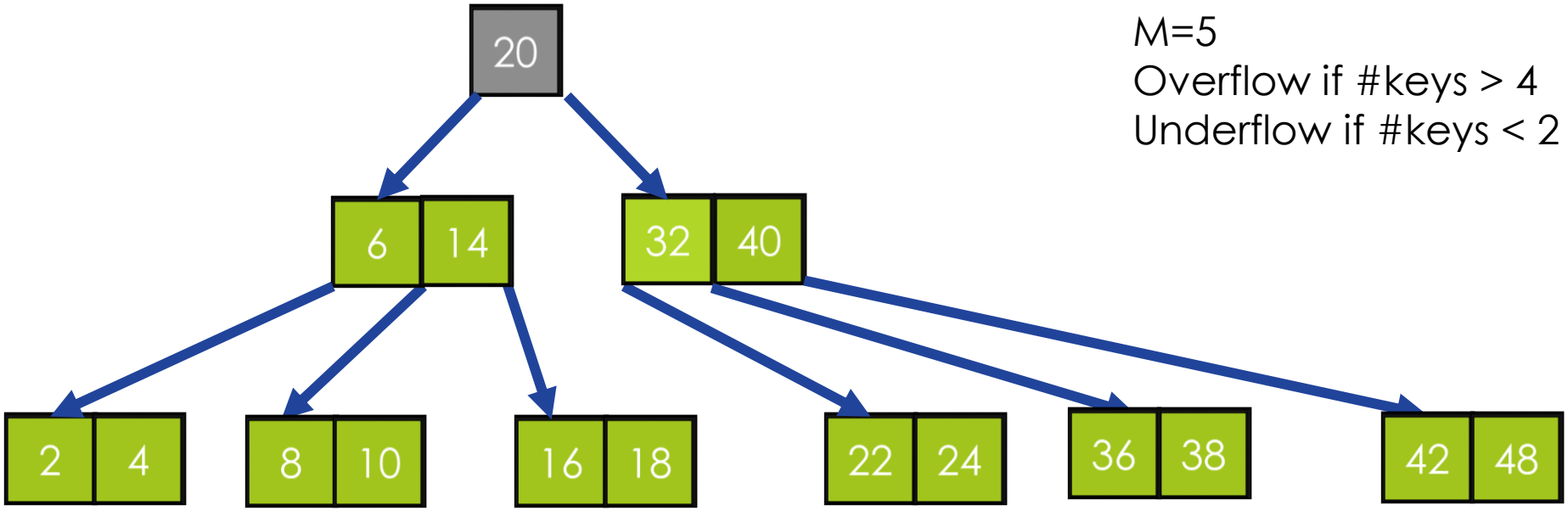
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)



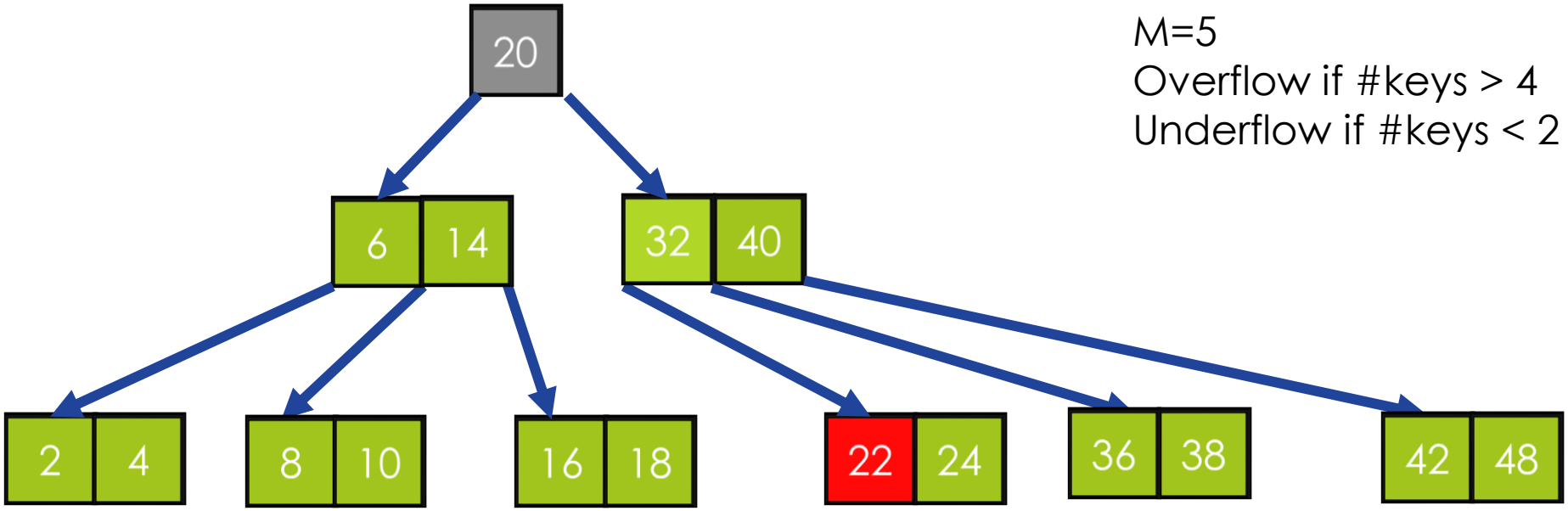
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)



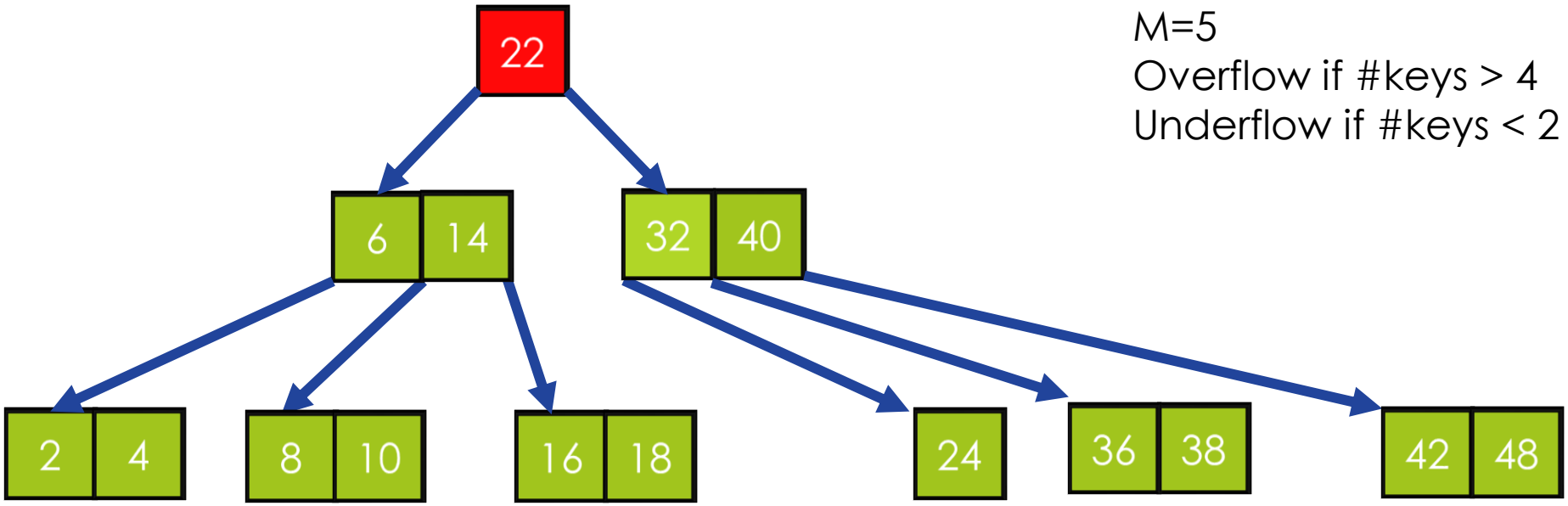
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)



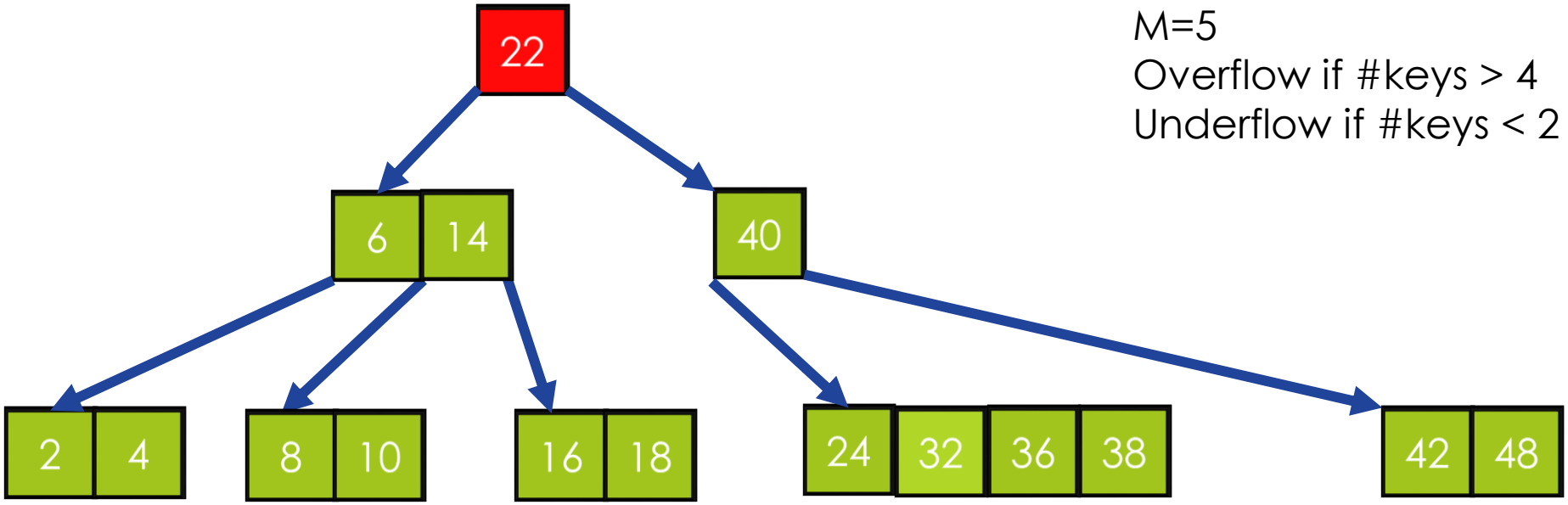
Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)



Example: Deletion in B-Tree

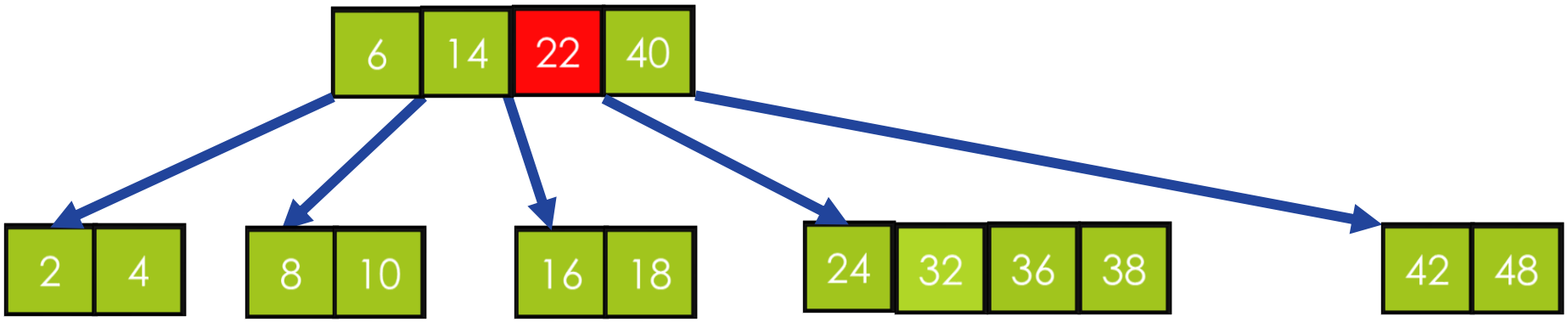
- Delete the following keys from the B-Tree
- Delete(20)



Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)

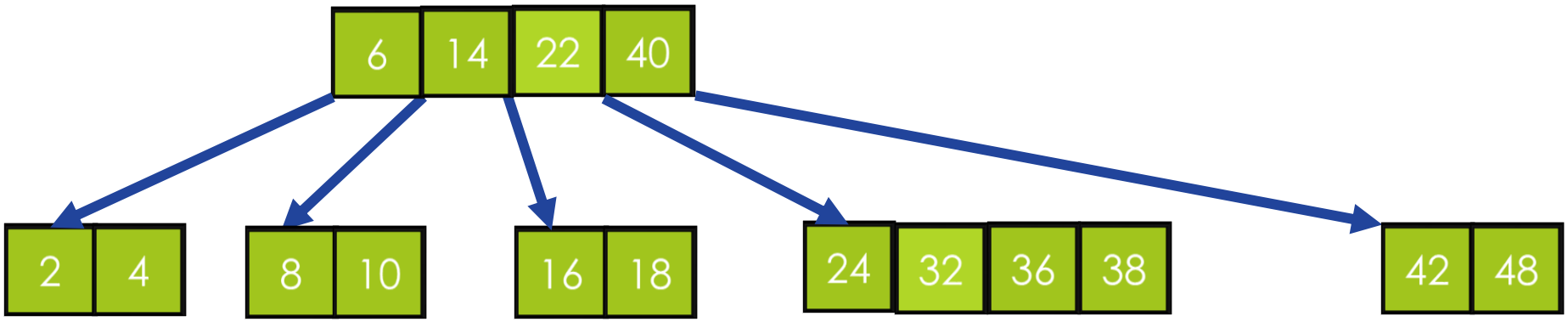
M=5
Overflow if #keys > 4
Underflow if #keys < 2



Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(20)

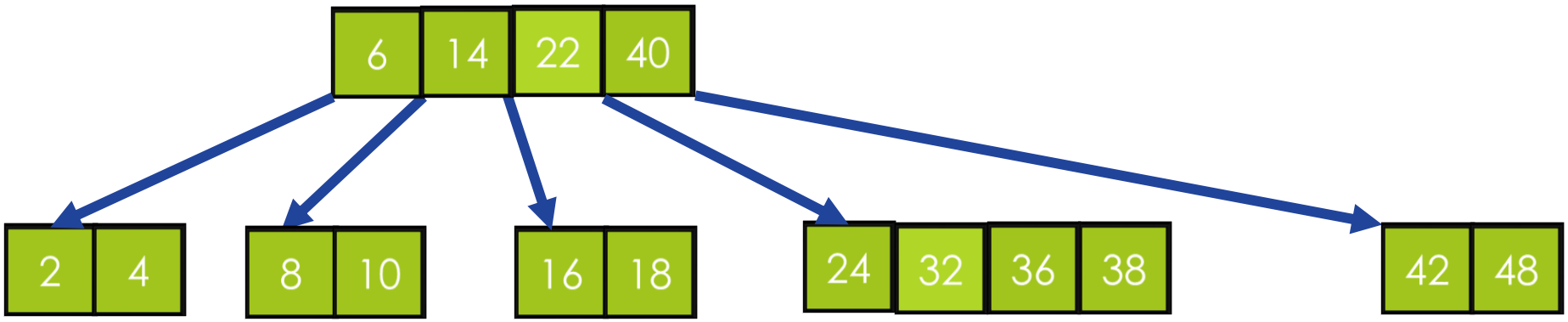
M=5
Overflow if #keys > 4
Underflow if #keys < 2



Example: Deletion in B-Tree

- Delete the following keys from the B-Tree
- Delete(2)?

M=5
Overflow if #keys > 4
Underflow if #keys < 2



Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  - Each internal node has up to M-1 keys to search
  - Each internal node has between $\lceil M/2 \rceil$ and M children
  - Depth of B-Tree storing N items is $O(\log_{\lceil M/2 \rceil} N)$
- Find: Run time is:
  - O(log M) to binary search which branch to take at each node. But M is small compared to N.
  - Total time to find an item is O(depth*log M) = O(log N)

Summary of Search Trees

- Problem with Binary Search Trees: Must keep tree balanced to allow fast access to stored items

- AVL trees: Insert/Delete operations keep tree balanced
- Splay trees: Repeated Find operations produce balanced trees
- Multi-way search trees (e.g. B-Trees):
  - More than two children per node allows shallow trees; all leaves are at the same depth.
  - Keeping tree balanced at all times.
  - Excellent for indexes in database systems.