Software 2 (Theory)

Lecture 10: B-trees

THE UNIVERSITY of York

Today's Topics

- Discuss structure of B-Trees
- The properties of B-Trees
- Basic operations on B-Trees
  - Searching, Insertion and Deletion
- Understand some applications of B-Trees

Introduction

- B-trees are powerful data structures used in various applications, including databases and file systems.
- A B-tree is a self-balancing tree data structure that maintains sorted data and facilitates efficient searching, insertion, and deletion operations.
- The "B" in B-tree can stand for "balanced," "binary," or "Bayer," after its inventors Rudolf Bayer and Edward M. McCreight.
- Unlike binary search trees, B-trees can have multiple keys and child pointers per node, leading to a more balanced structure.

Structure of B-Tree

- Each node in a B-tree can have multiple keys and child pointers.
  - Also known as “large key” trees.
- The keys within a node are stored in sorted order, facilitating efficient searching.
  - Offers larger branching factor and shallower height for faster search and insertion.
- All leaf nodes are at the same level.
  - ensuring a balanced tree structure.
- The number of keys within a node is constrained by properties such as the minimum and maximum number of keys allowed per node.
  - Maintaining a balance which guarantees the time complexity for its operations.

Time Complexity of B-Tree

- **Searching**: Search for a key by traversing the tree from the root downwards, following the appropriate child pointers based on the key values.
- **Insertion**: Insert a new key into the B-tree by finding the appropriate leaf node and inserting the key while maintaining the sorted order.
- **Deletion**: Delete a key from the B-tree by first locating the key and then removing it from the appropriate node, adjusting the structure if necessary to maintain balance.

Algorithm	Time Complexity
Search	O(log n)
Insert	O(log n)
Delete	O(log n)

Properties of B-Trees

- **Balanced Structure**: B-trees maintain a balanced structure by redistributing data among nodes as necessary, ensuring efficient operations.
- **Efficient Searching**: Searching in a B-tree is efficient, typically requiring logarithmic time complexity due to the balanced nature of the tree.
- **Efficient Insertion and Deletion**: B-trees support efficient insertion and deletion operations by dynamically adjusting the structure while maintaining balance.
- **Optimal Disk I/O**: B-trees are well-suited for applications requiring persistent storage on disk, as they minimise disk I/O by ensuring a balanced tree structure.

B-Trees shape properties

A B-tree of order m is defined to have the following shape properties:

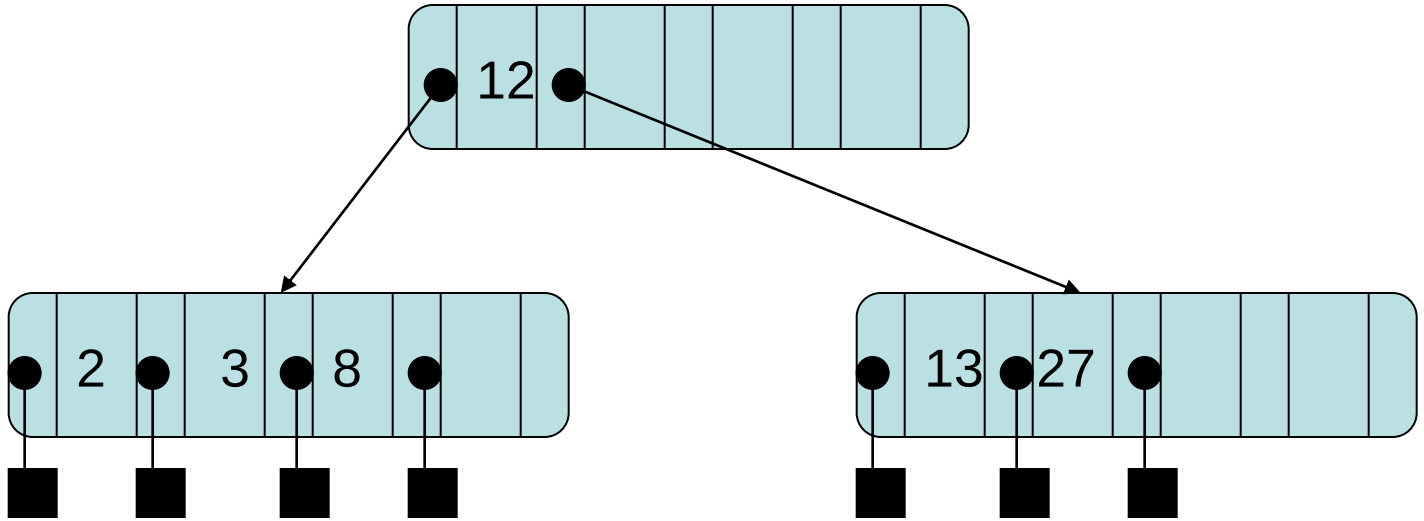
- Every node has at most m children.
- Every non-leaf node (except the root node) has at least $m = 2$ children.
- The root node, if it is not a leaf node, has at least two children.
- A non-leaf node with m children contains $m - 1$ search keys.
- Each internal node, except for the root, has between $\lceil {}^{m}/_{2} \rceil$ and m children.
- All leaves are at the same level in the tree, carries information and hence, the tree is always height balanced.

B-Tree Example with order 5 (m = 5)

The root has between 2 and *m* children.

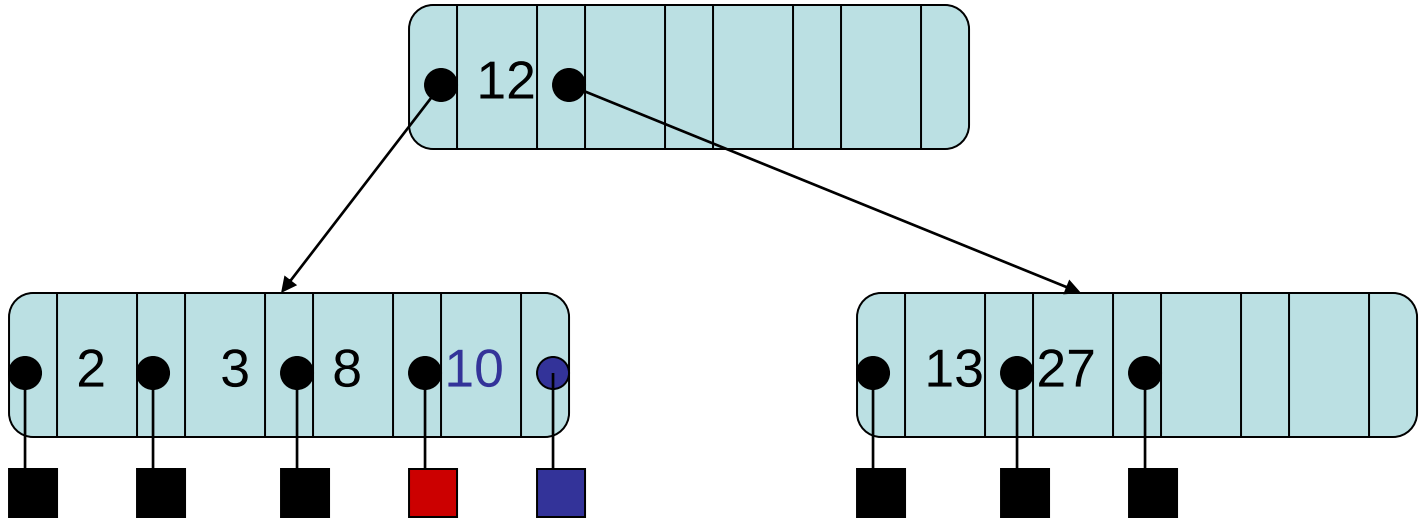
Each non-root internal node has between $\lceil m/2 \rceil$ and *m* children.

All external nodes are at the same level. (External nodes are represented by null pointers in implementations.)



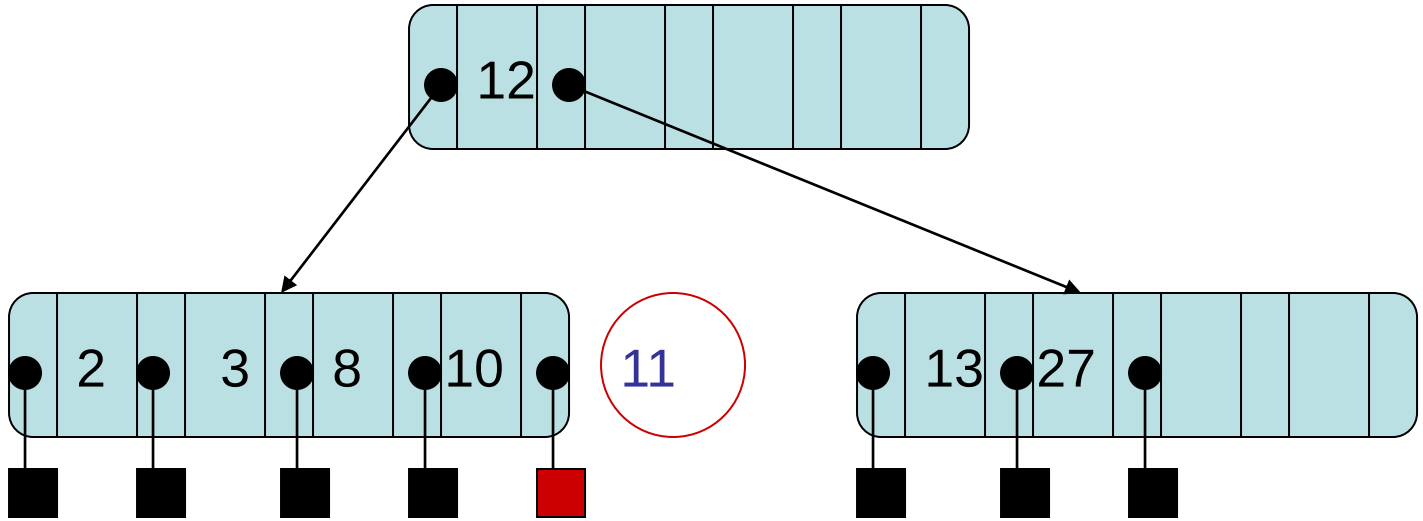
B-Tree Insertion : insert 10

- We find the location for 10 by following a path from the root using the stored key values to guide the search.
- The search falls out the tree at the 4th child of the 1st child of the root.
- The 1st child of the root has room for the new element, so we store it there.

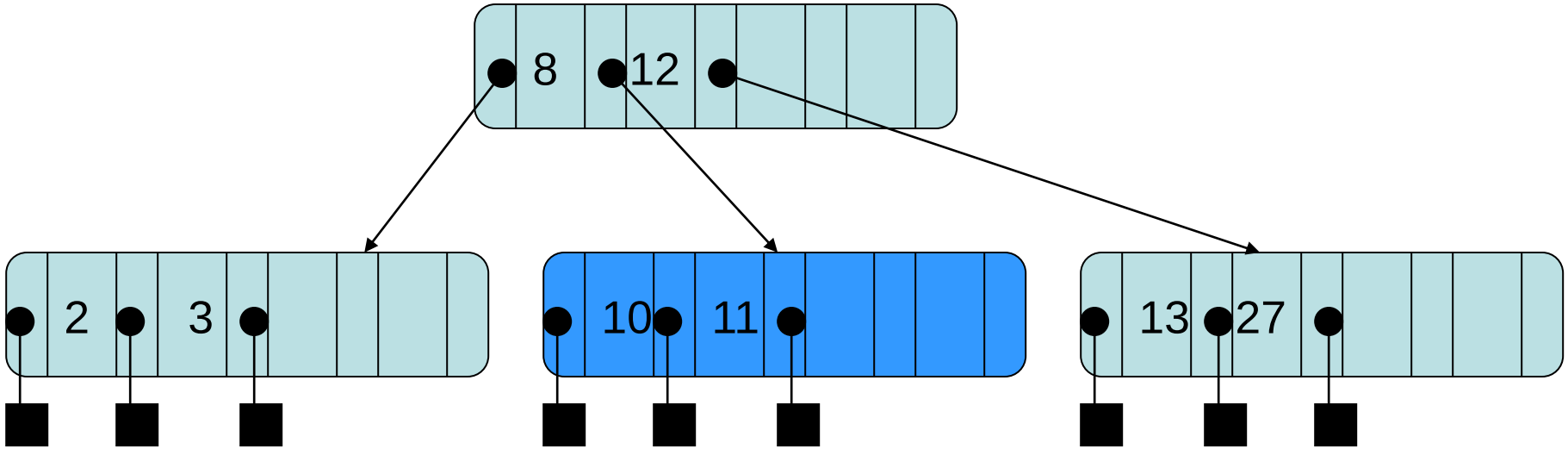


B-Tree Insertion : insert 11

- We fall out of the tree at the child to the right of key 10.
- But there is no more room in the left child of the root to hold 11.
- Therefore, we must split this node.



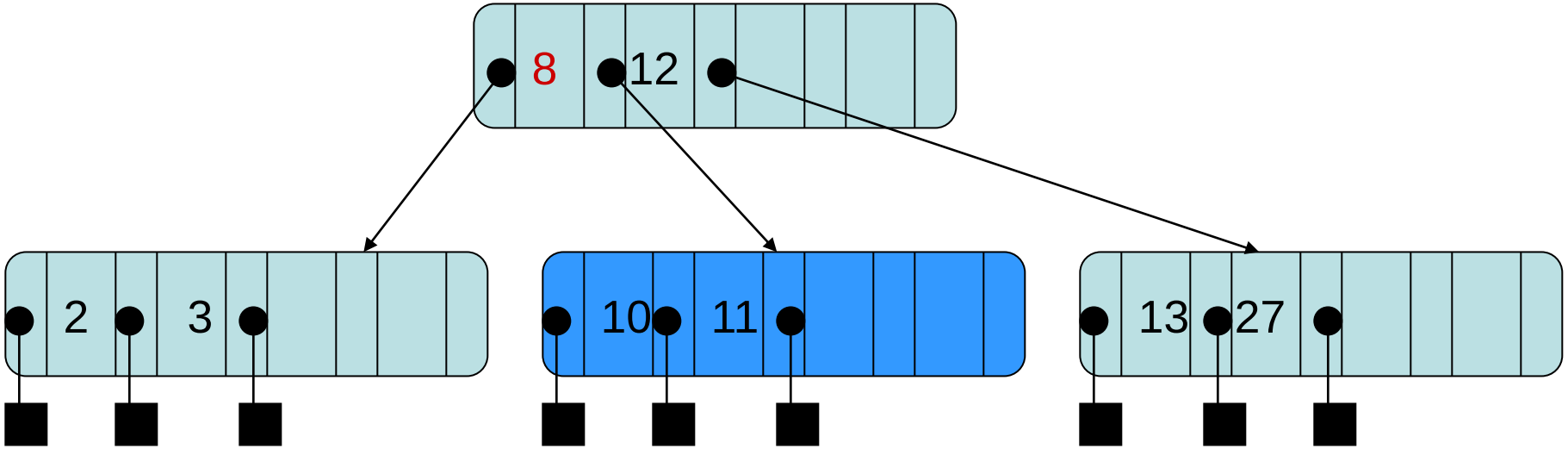
B-Tree Insertion : insert 11 (continued)



The m + 1 children are divided evenly between the old and new nodes.

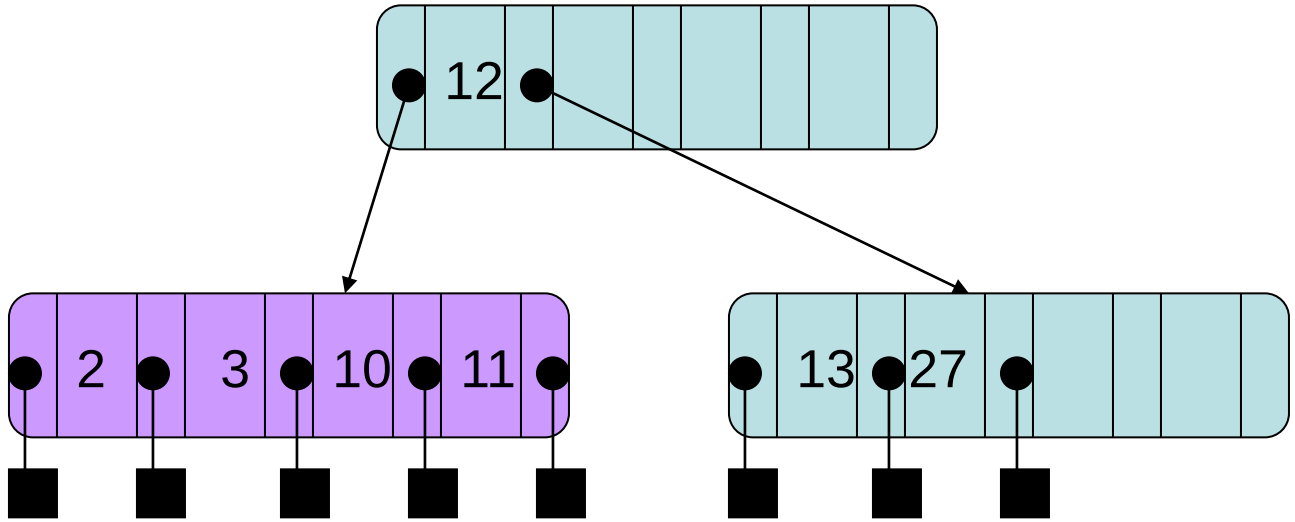
The parent gets one new child. (If the parent become overfull, then it, too, will have to be split).

B-Tree deletion : delete 8



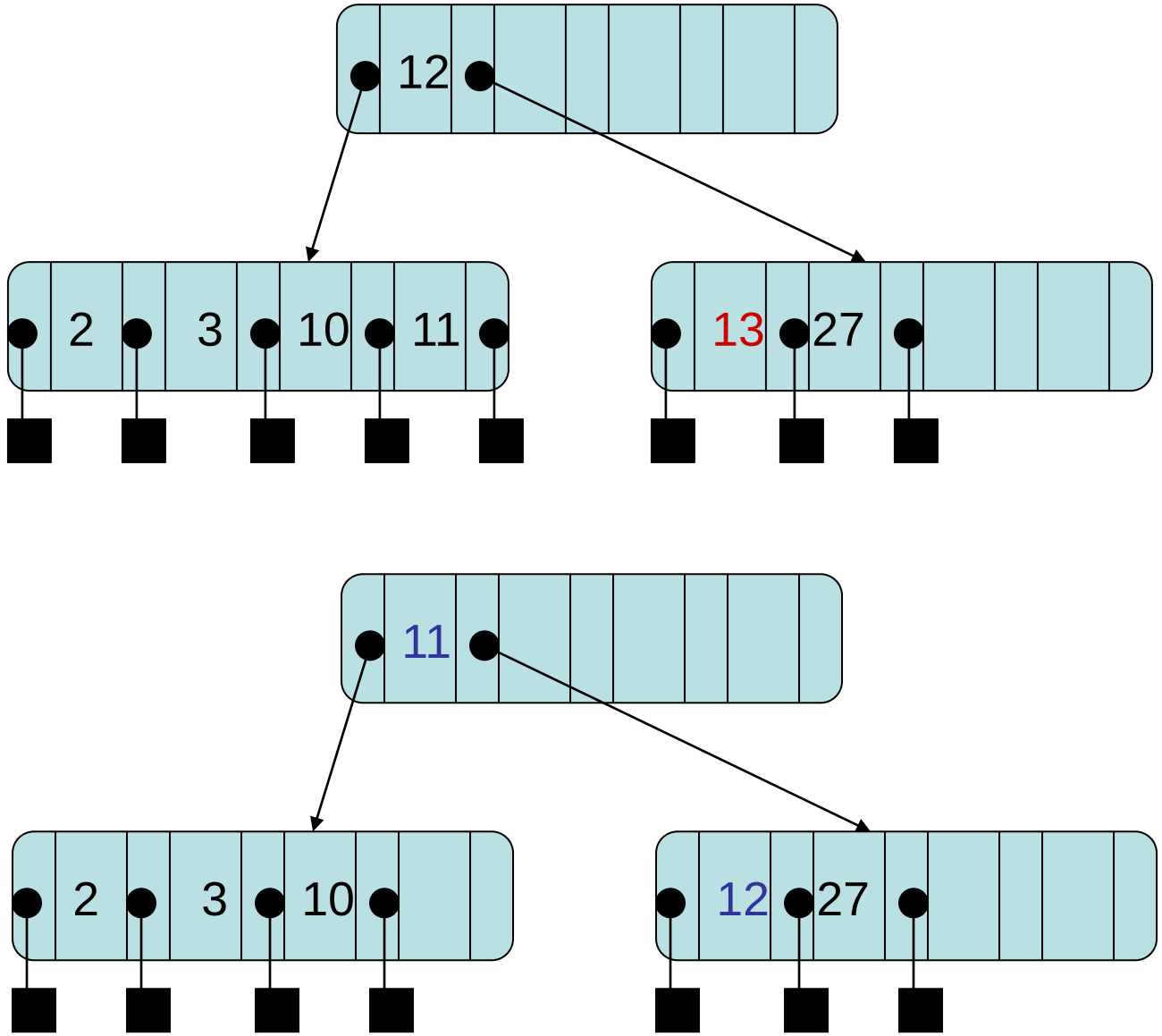
Removing 8 might force us to move another key up from one of the children. It could either be the 3 from the 1st child or the 10 from the second child.

However, neither child has more than the minimum number of children ($m=\lceil m/2 \rceil = 3$) for m = 5, so the two nodes will have to be merged. Nothing moves up.



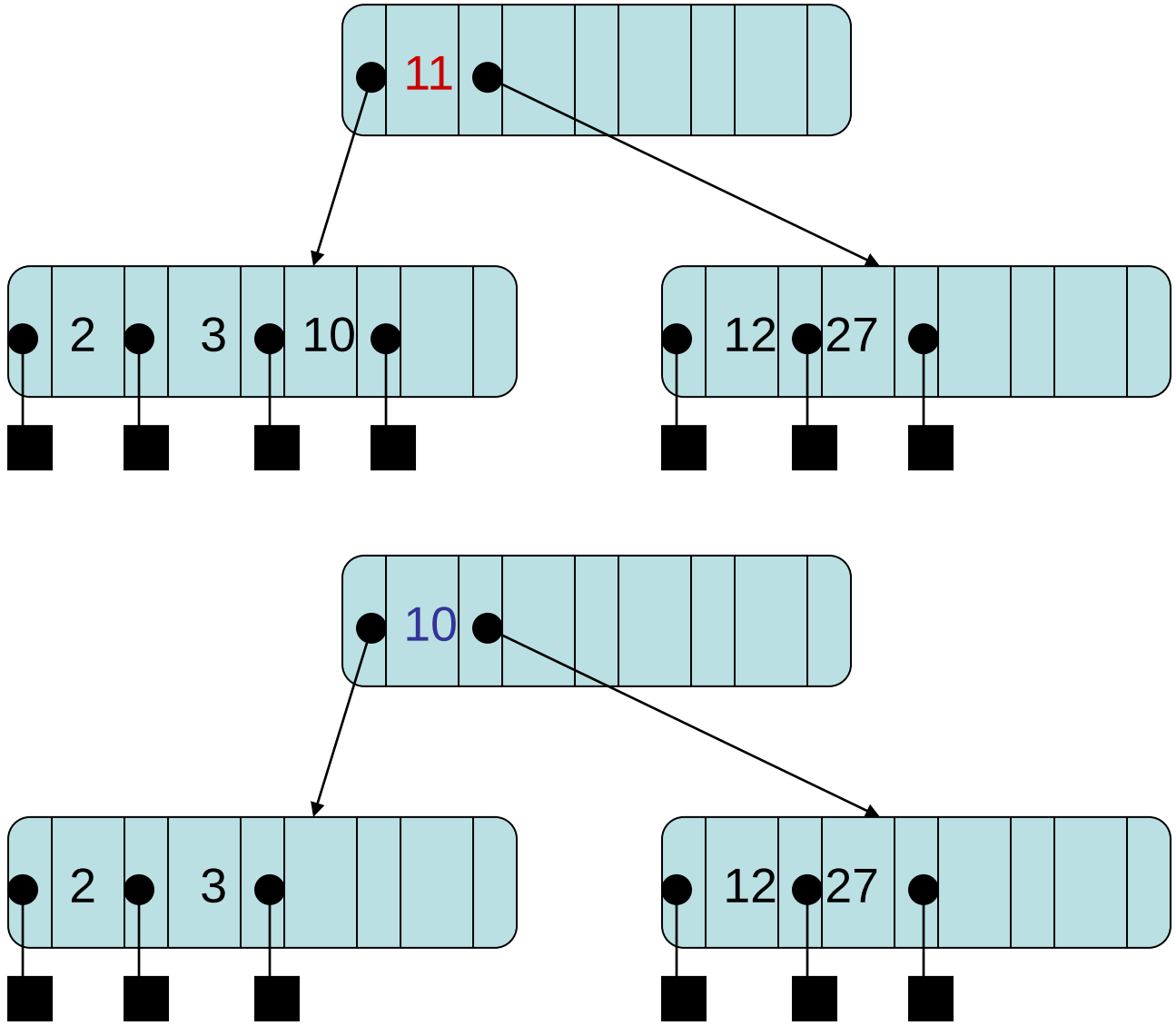
B-Tree Deletion : delete 13

- Deleting 13 would cause the node containing it to become underfill.
- To fix this, we try to reassign one key from a sibling that has spares.
- The 13 is replaced by the parent's key 12.
- The parent's key 12 is replaced by the spare key 11 from the left sibling.



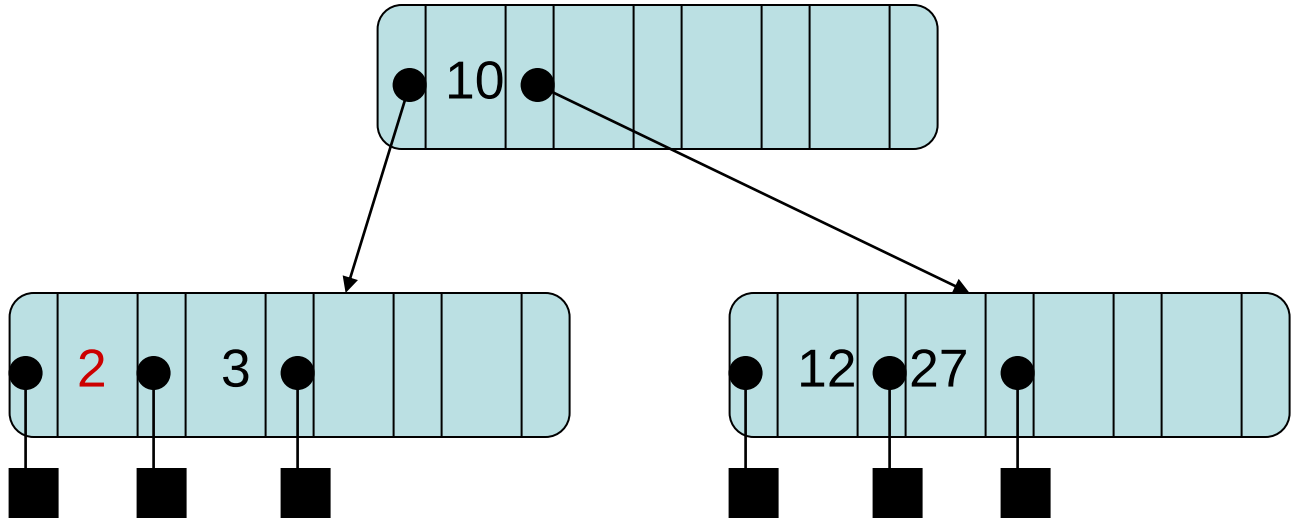
B-Tree Deletion : delete 11

- 11 is in a non-leaf, so replace it by the value immediately preceding: 10.
- 10 is at leaf, and this node has spares.



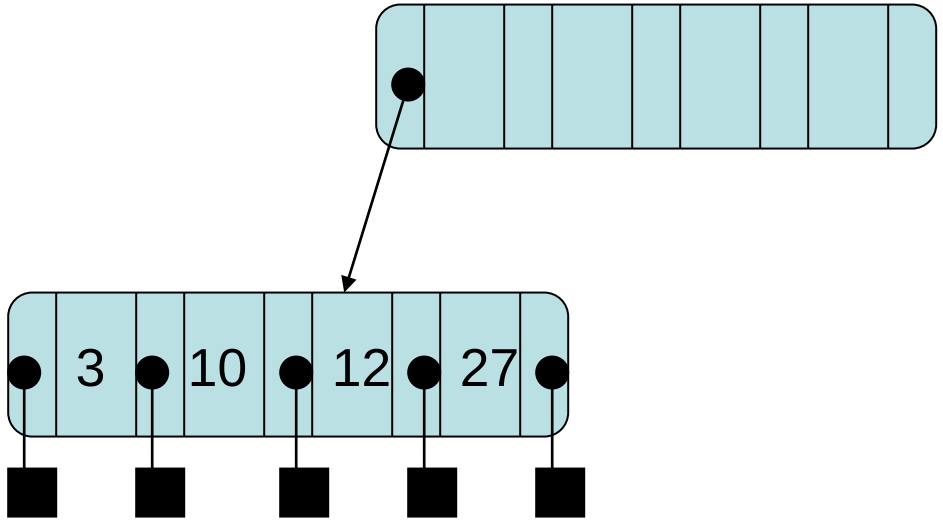
B-Tree Deletion : delete 2

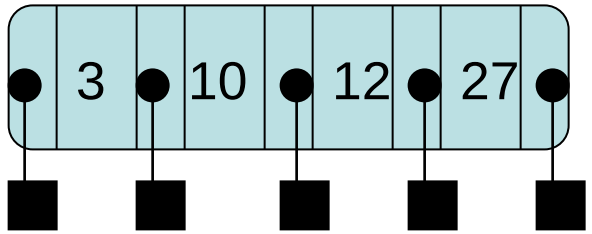
- Although 2 is at leaf level, removing it leads to an underfill node.
- The node has no left sibling. It does have a right sibling, but that node is at its minimum occupancy already.
- Therefore, the node must be merged with its right sibling.



B-Tree Deletion : delete 2

- This is no longer a B-tree, because the root does not have at least 2 children.
- Therefore, we must remove the root, making its child the new root.





Applications of B-Trees

- B-trees are widely used in databases and file systems due to their efficient searching, insertion, and deletion operations.
- They are also used in other applications such as indexing and caching, where fast access to large amounts of data is required.
- By maintaining a balanced structure and supporting efficient operations, B-trees enable high-performance applications in various domains.
- B-Trees are used in CAD systems to organize and search geometric data.
- B-Trees are also used in other areas such as natural language processing, computer networks, and cryptography.

Summary

- Understand:
  - B-Trees
  - B-Tree search, insertion and deletion
  - Similar time complex for the 3 operations
  - Some applications of B-Trees
- Read
  - Skiena, Section 15.1
  - Goodrick, Chapter 15
  - Cormen, Chapter 18

- Next
  - Revision Lecture