Exploring the use of B+ trees in Database Management Systems

We’re in the endgame now

INDEXING

Indexing is a key concept in Database Management Systems, as one optimize performance when a query is processed.

The time complexity heavily depends on the **data structure** used.

UNSORTED ARRAY

insert: O(1)

search: O(N)

SORTED ARRAY

insert: O(N)

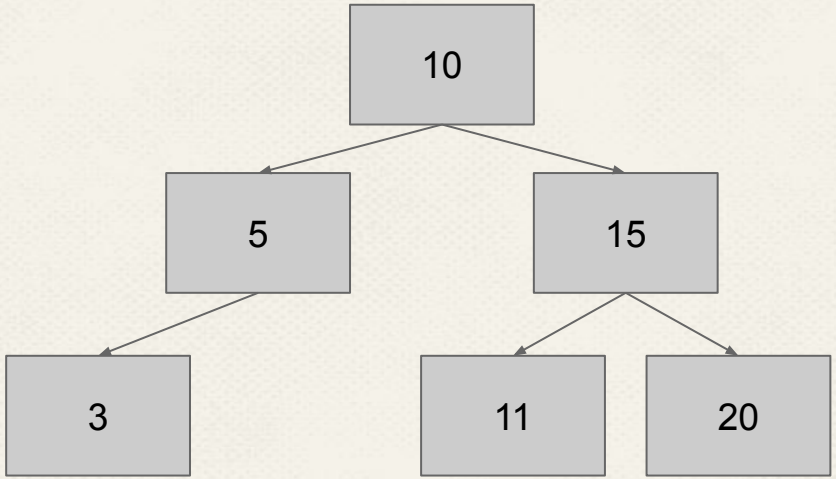
search: O(log n)

HASHMAP

insert: O(1)

search: O(1)

BINARY SEARCH TREES



insert: O(log N)

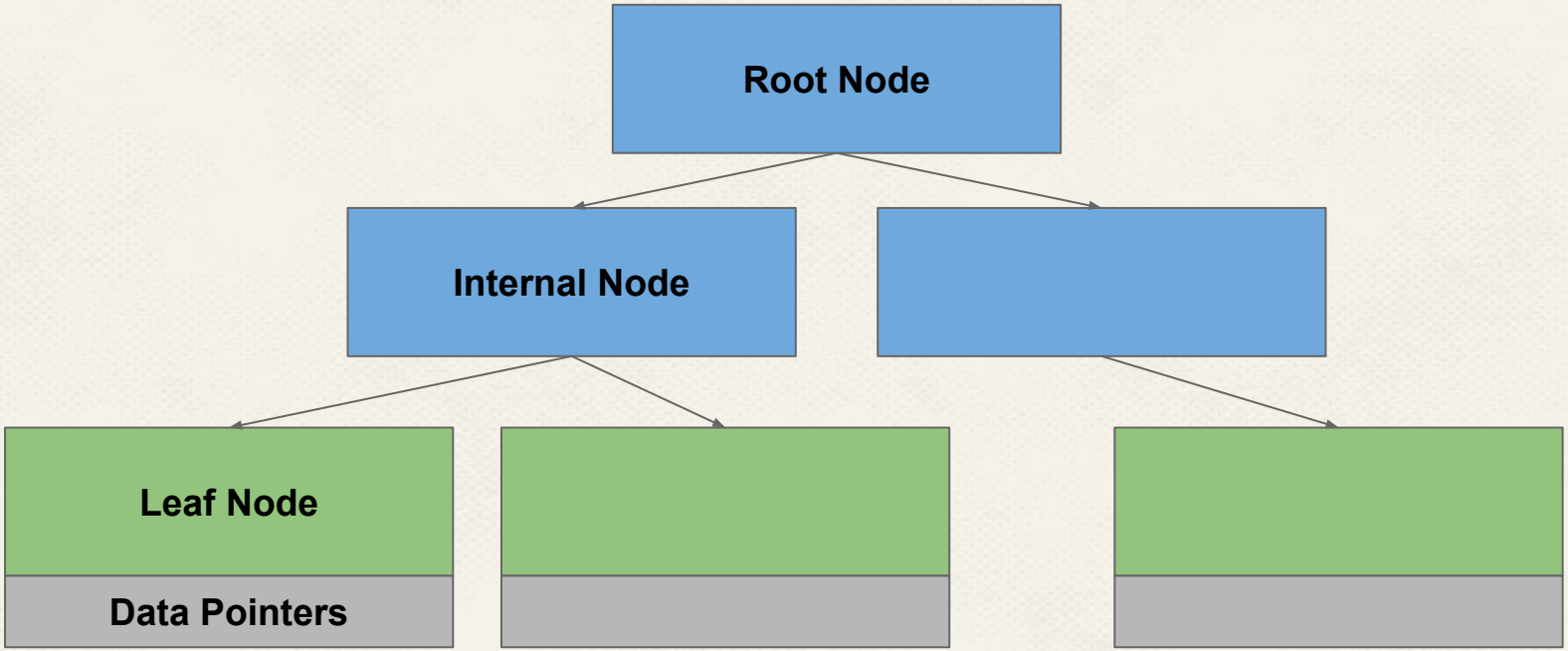
search: O(log N) / O(N)

Each time you retrieve a node from a tree you have to pull a block from memory. Since a BST node can only point to 2 children, it’s inefficient to retrieve the values stored in the BST as you need to make many memory calls.

In order to minimize the number of memory calls
B+ trees are used.

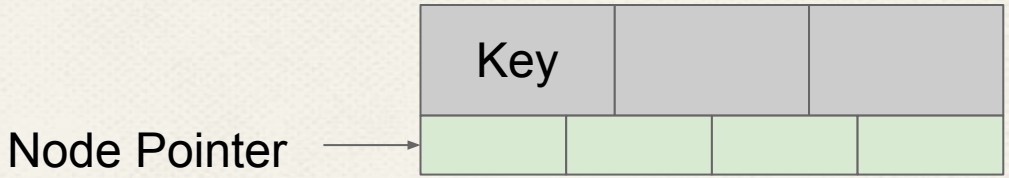
B+ Tree Terminology

B+ TREES



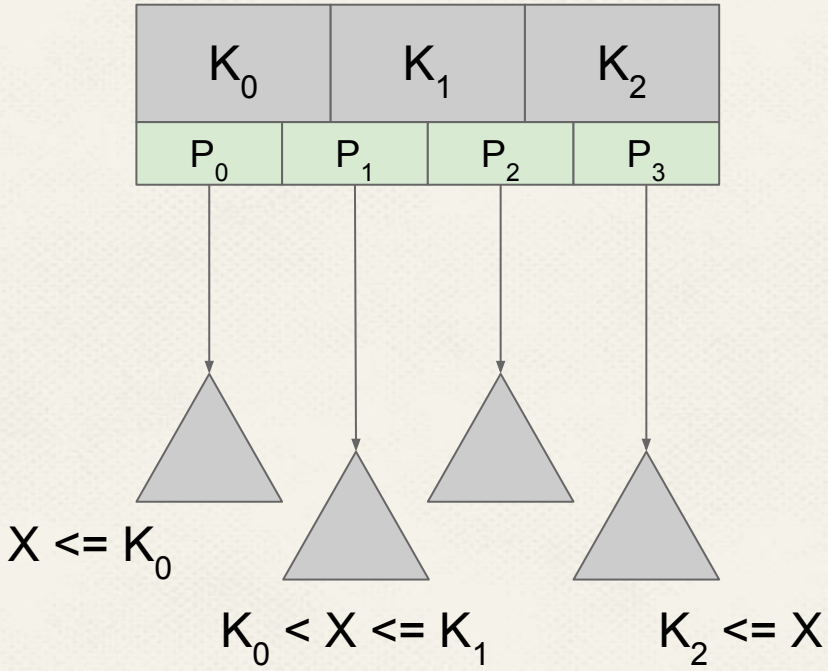
INTERNAL NODES

Internal Node



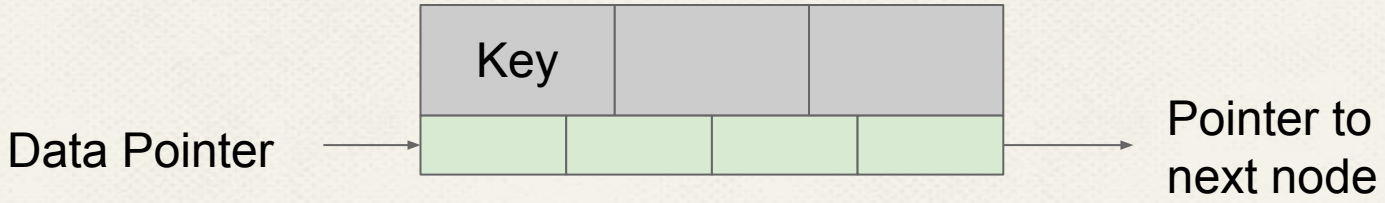
INTERNAL NODES

Internal Node



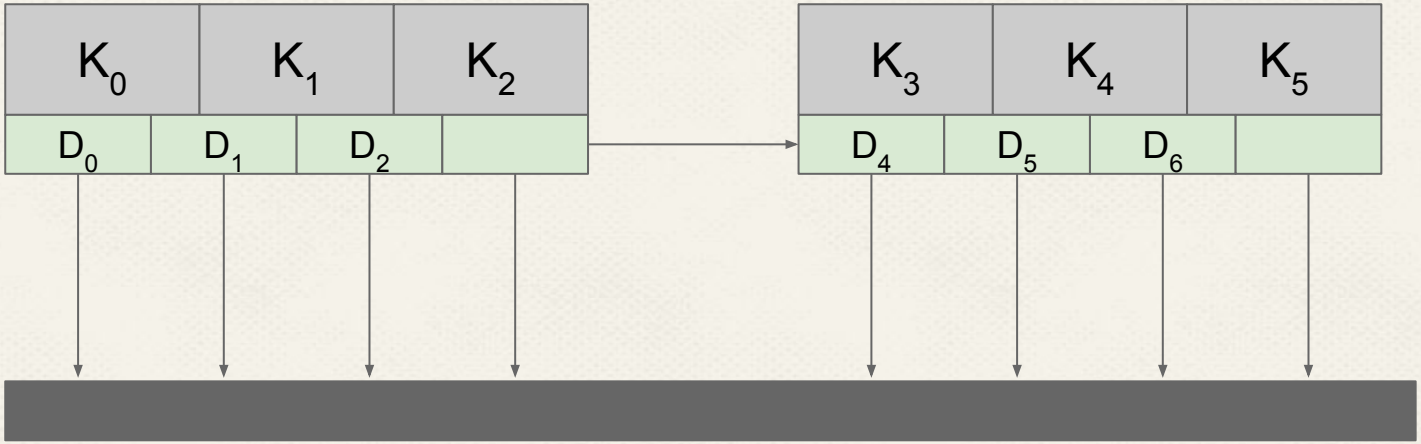
LEAF NODES

Leaf Node



LEAF NODES

Leaf Node



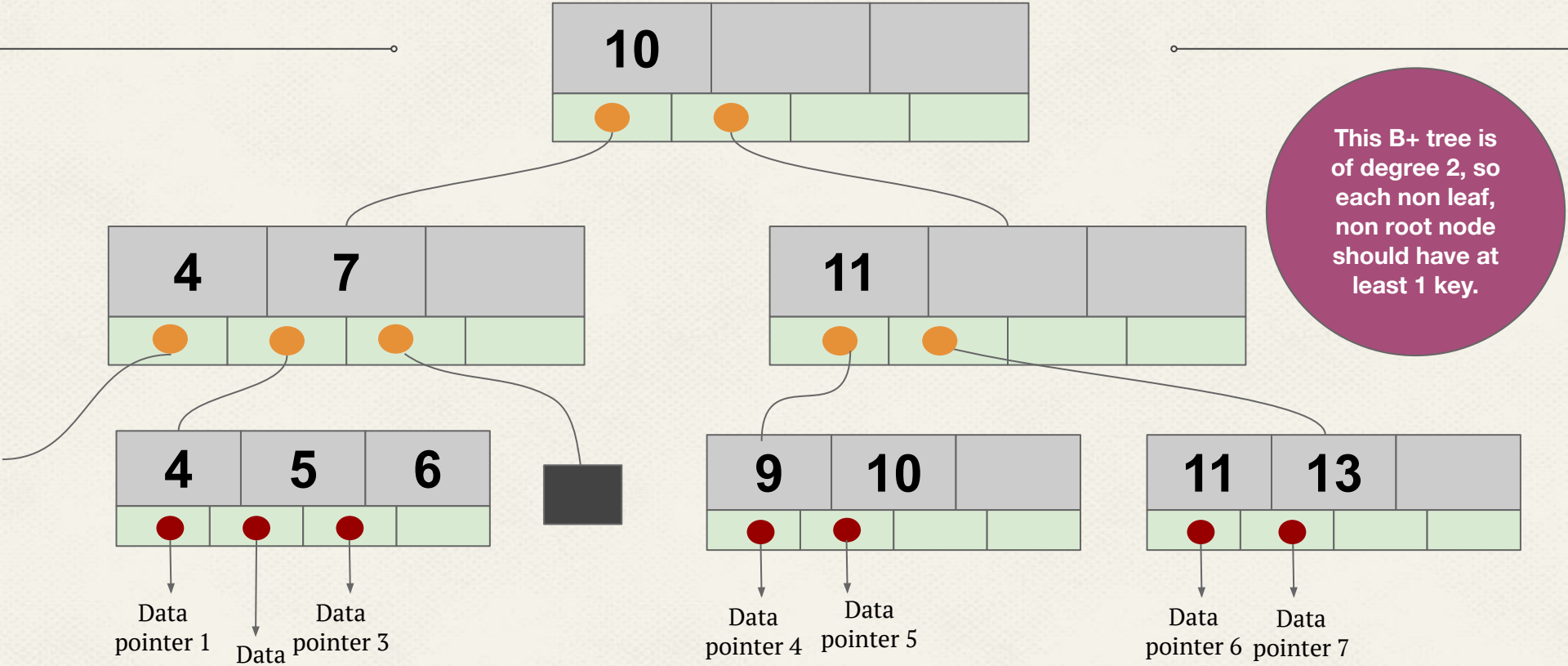
Degree and order

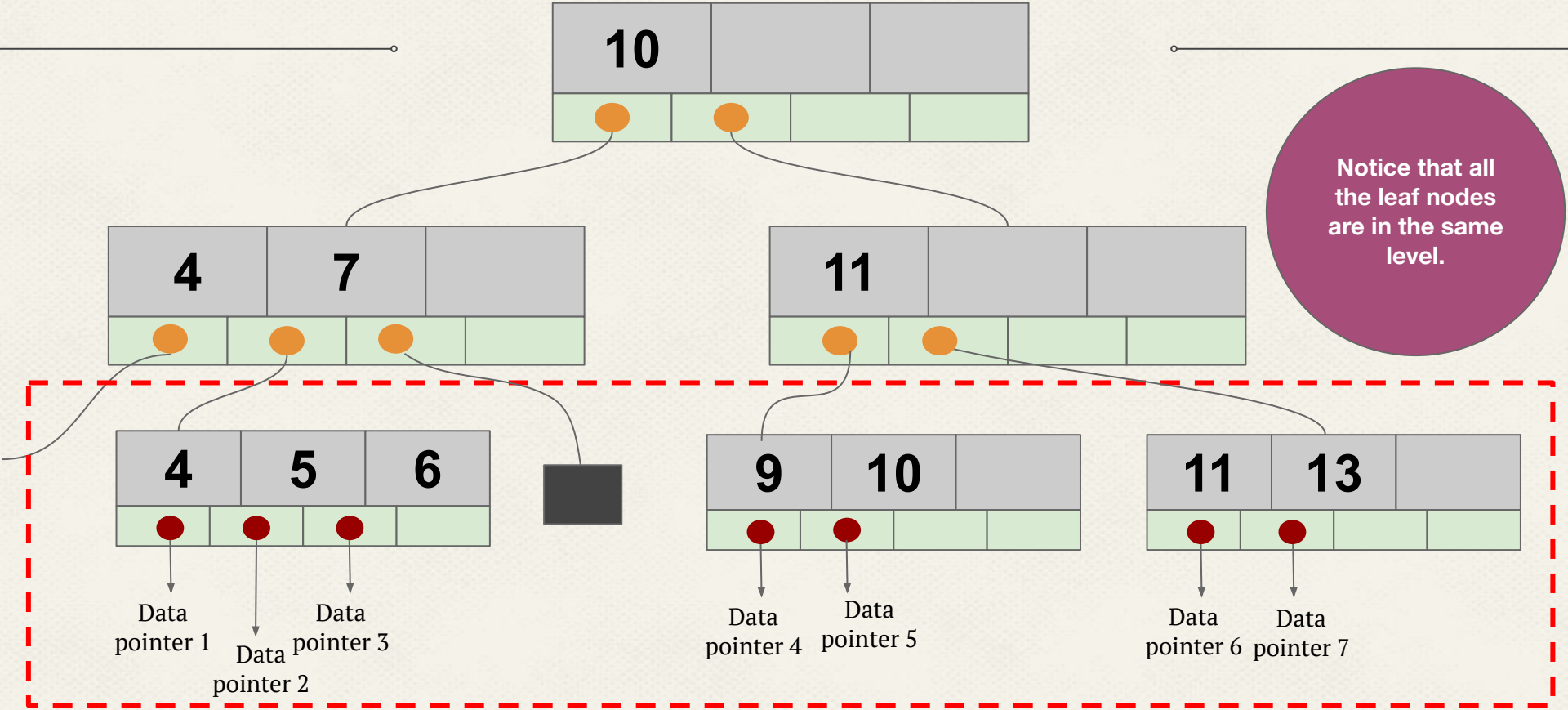
Order of B+ trees:
The maximum number of children that a B+ tree node can have.

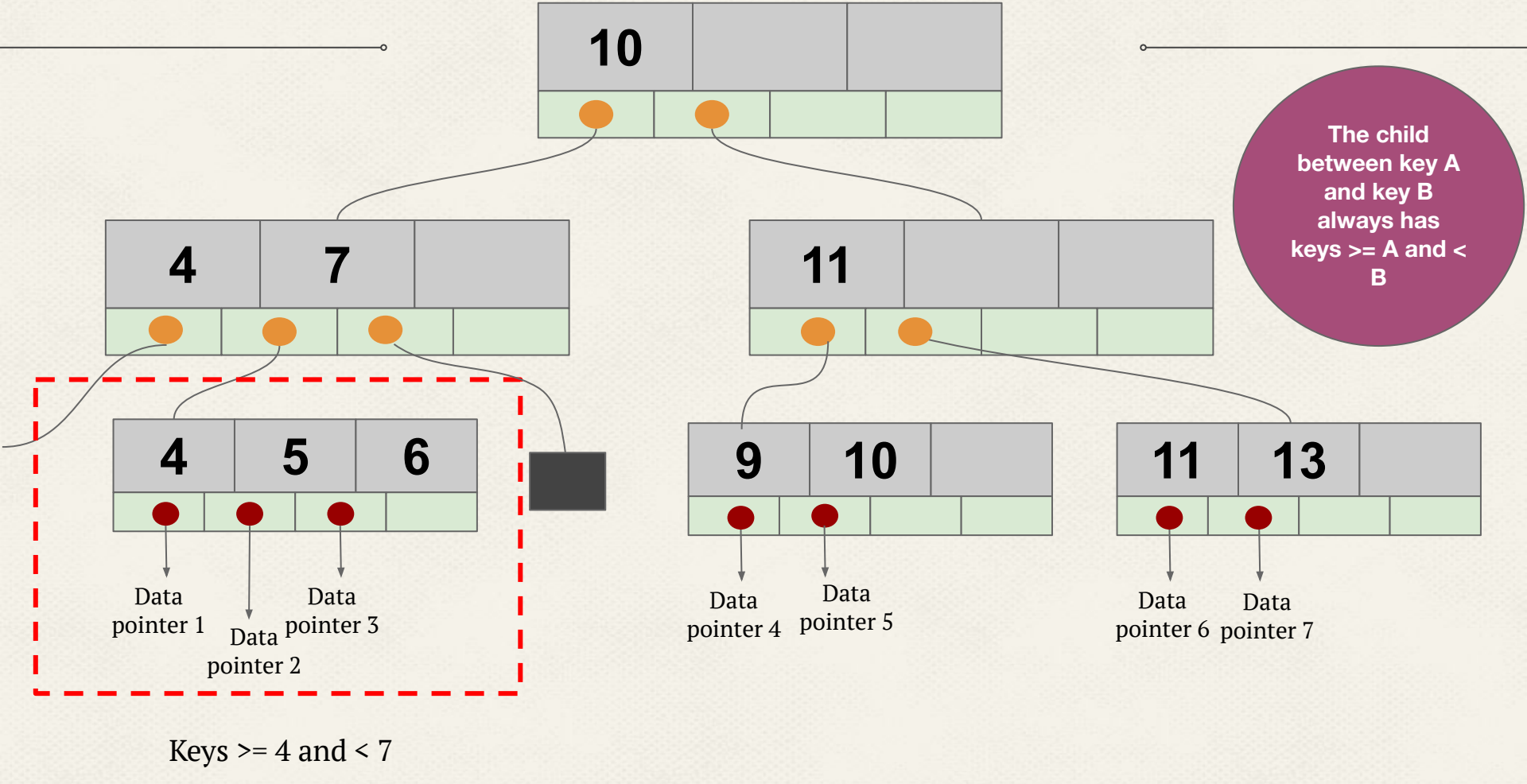
Degree of B+ trees:
For a B+ tree of degree **d**, each node must have **at least d-1 keys** can have **at most 2d-1 keys**.

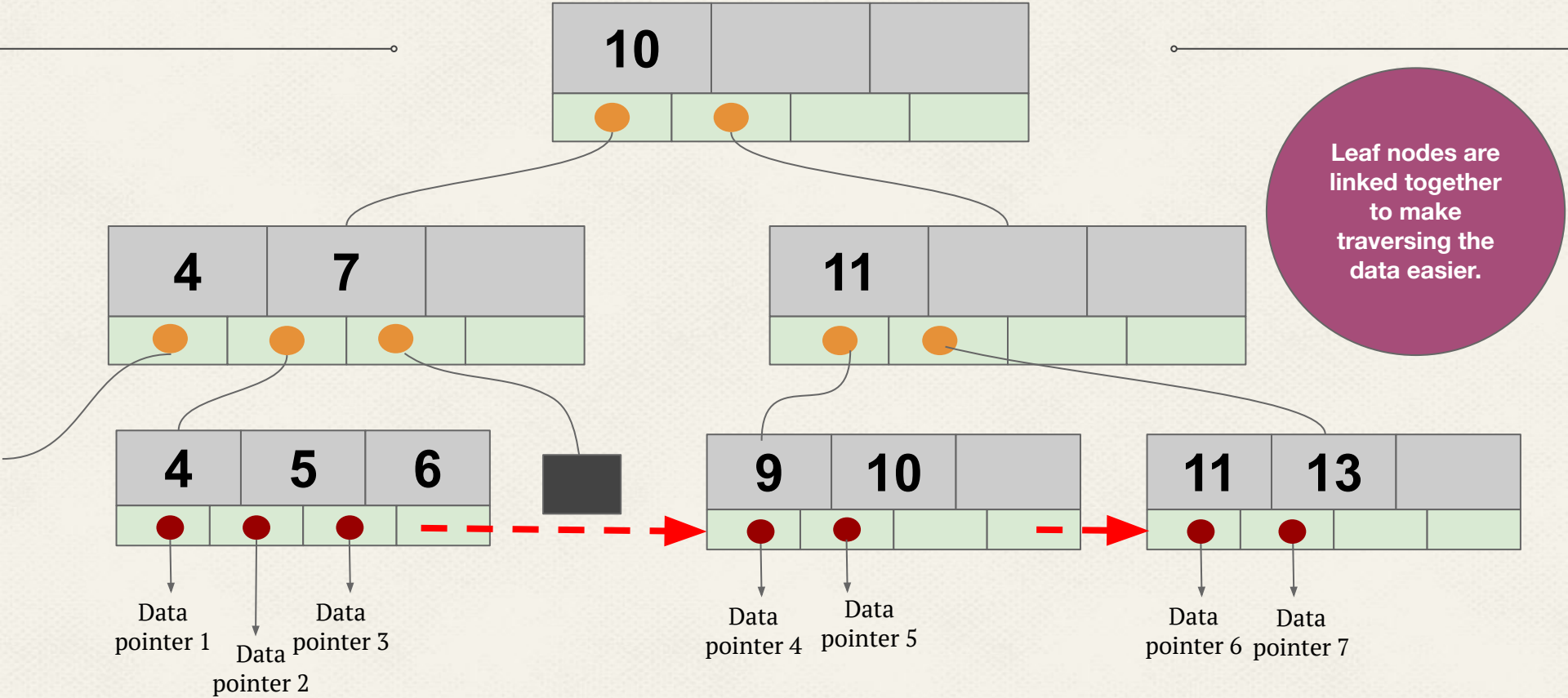
Example

B+ tree of degree 2


10
4
7
11
4
5
6
9
10
11
13
Data pointer 1
Data pointer 2
Data pointer 3
Data pointer 4
Data pointer 5
Data pointer 6
Data pointer 7
This B+ tree is of degree 2, so each non leaf, non root node should have at least 1 key.


10
4
7
11
Notice that all the leaf nodes are in the same level.
4
5
6
9
10
11
13
Data pointer 1
Data pointer 2
Data pointer 3
Data pointer 4
Data pointer 5
Data pointer 6
Data pointer 7


10
4
7
11
4
5
6
9
10
11
13
Data pointer 1
Data pointer 2
Data pointer 3
Data pointer 4
Data pointer 5
Data pointer 6
Data pointer 7
The child between key A and key B always has keys >= A and < B
Keys >= 4 and < 7


10
4
7
11
4
5
6
9
10
11
13
Data pointer 1
Data pointer 2
Data pointer 3
Data pointer 4
Data pointer 5
Data pointer 6
Data pointer 7
Leaf nodes are linked together to make traversing the data easier.

INSERT

How do we add a value?

Insertion Algorithm

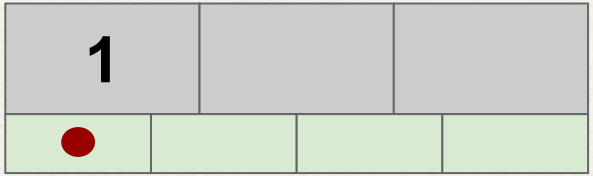
1. Find the node that the key should be inserted in.

2. Check if node is full.

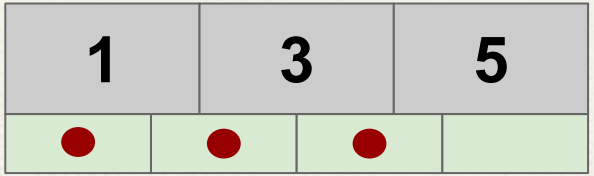
    a) If node is not full, insert the key in the correct position.

    b) If the node is full, split the node into two. Copy and insert the median key value into the parent if the node is a leaf node. Else, promote the median without copying

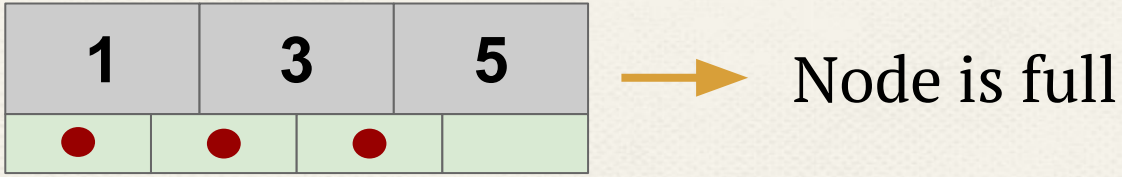
Insertion Algorithm
Case - 1: Insert 3 and 5



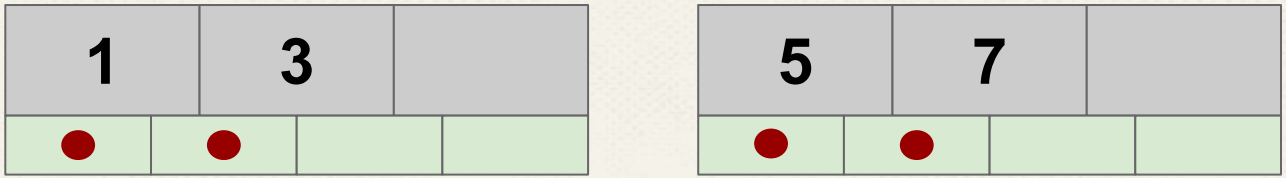
Insertion Algorithm
Case - 1: Insert 3 and 5



Insertion Algorithm
Case - 2: Insert 7

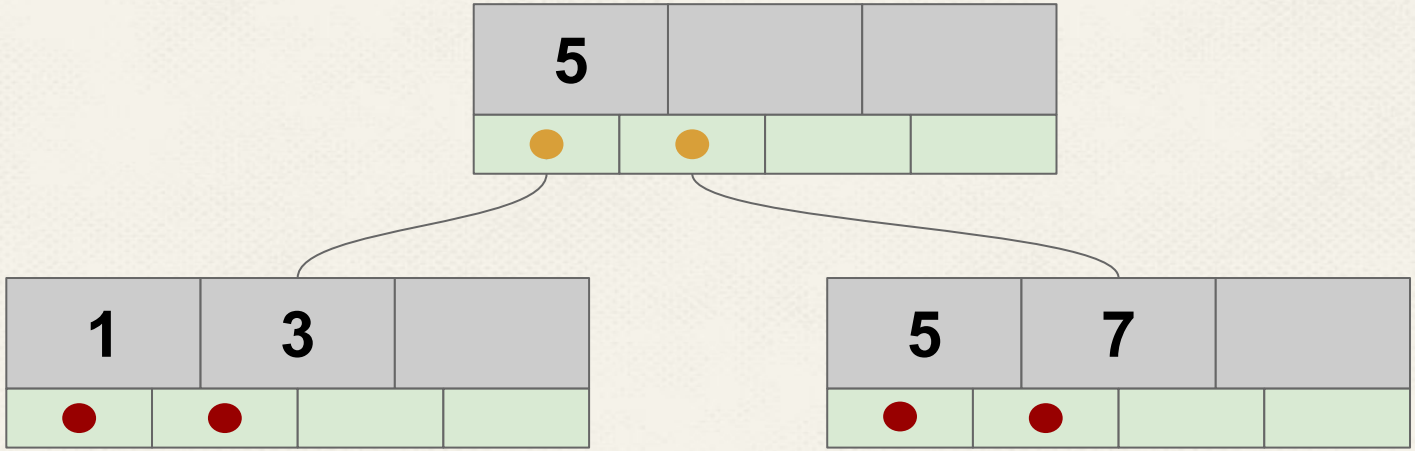


Insertion Algorithm
Case - 2: Insert 7



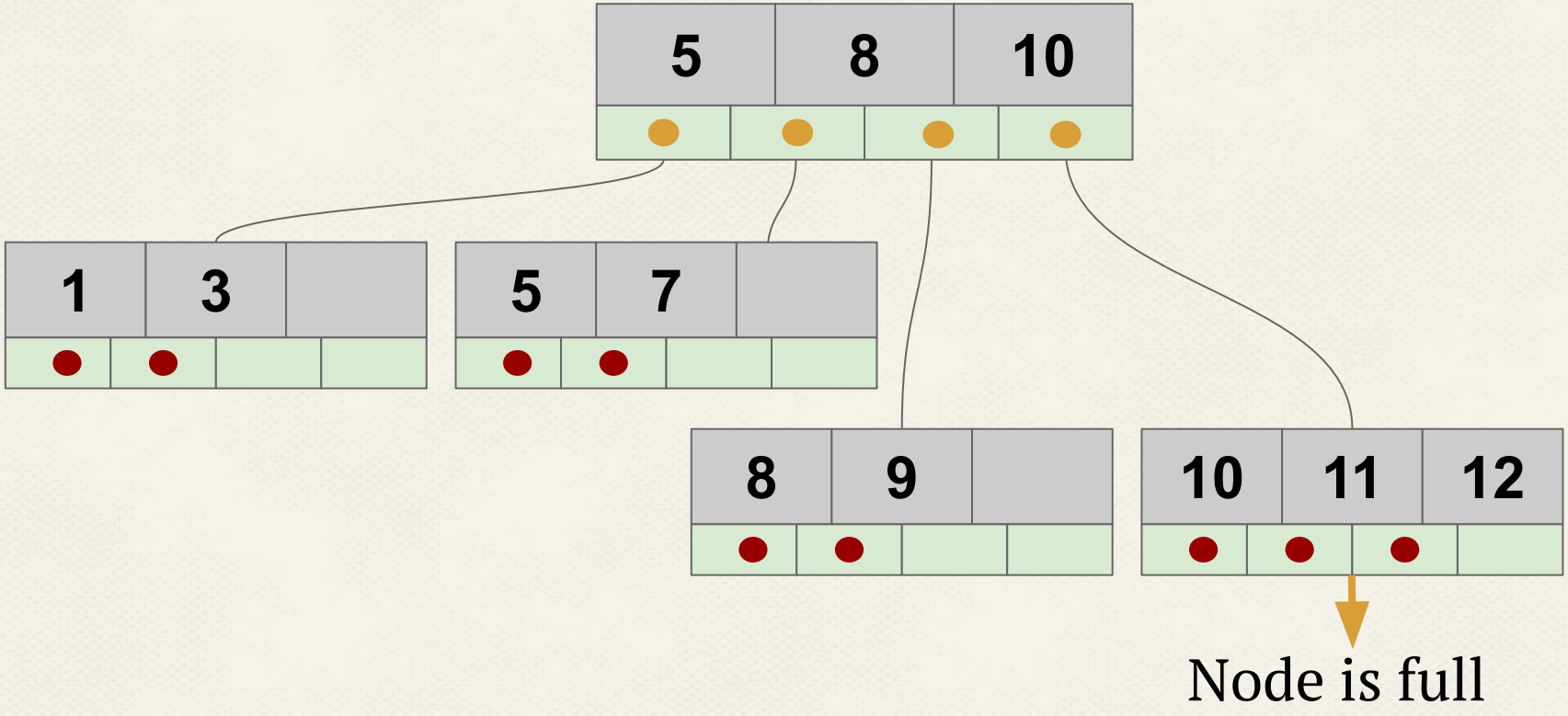
Insertion Algorithm
Case - 2: Insert 7

Insertion Algorithm
Case - 2: Insert 7

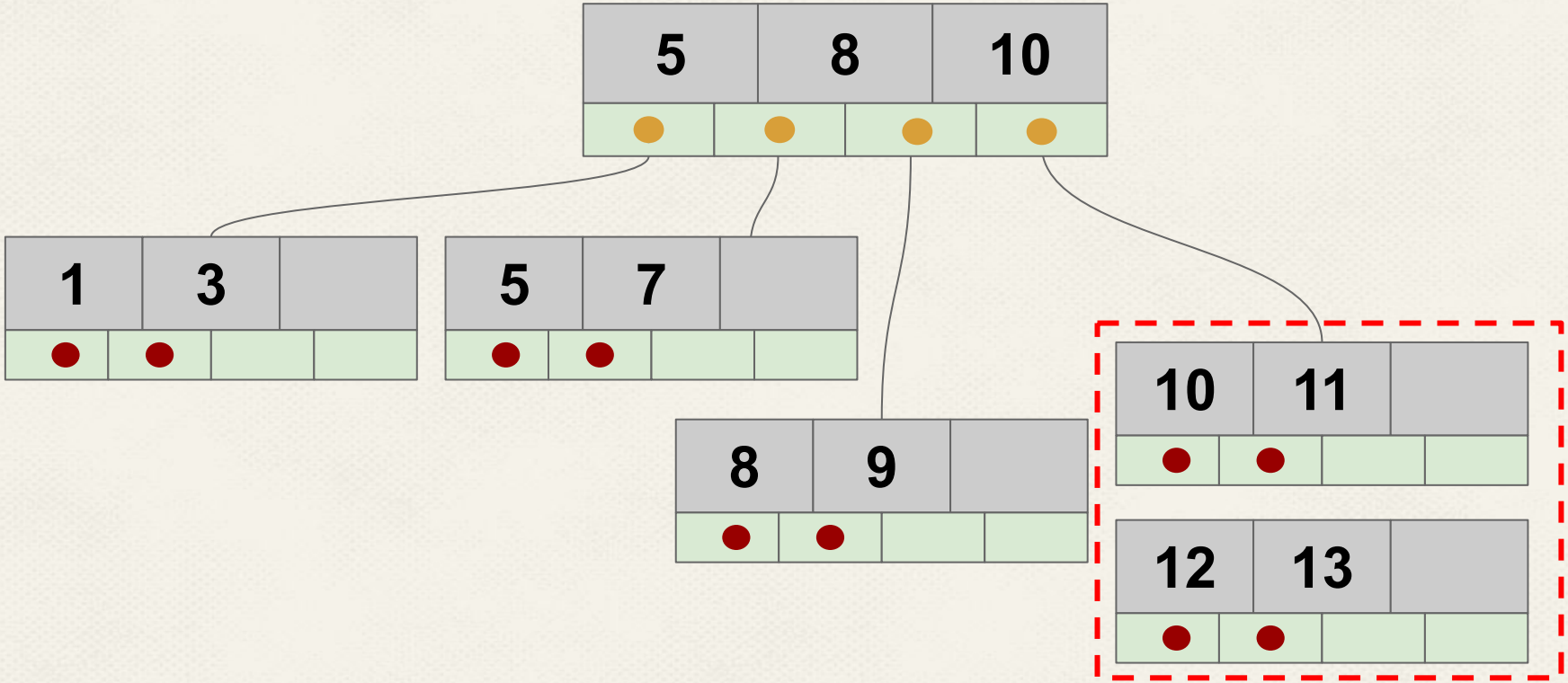


Fast forward

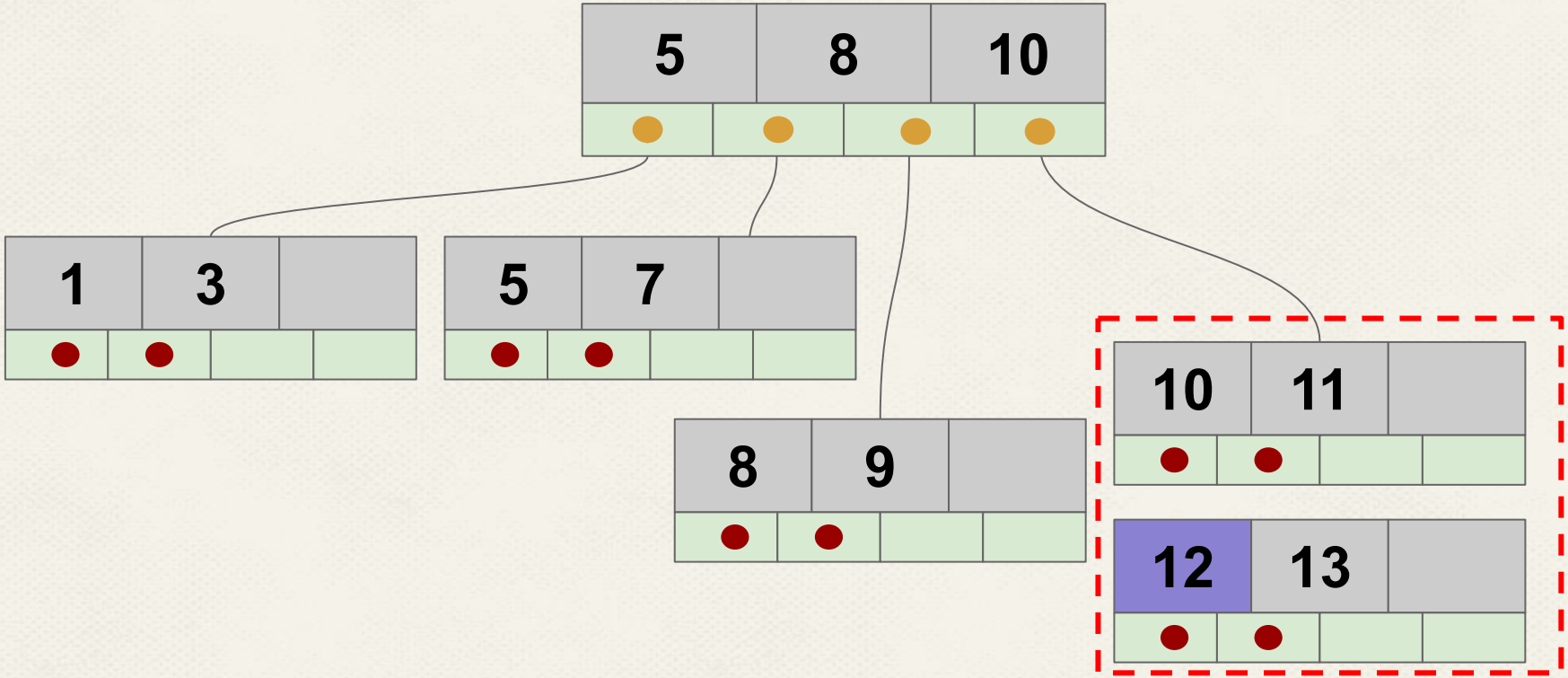
Insertion Algorithm
Case - 3: Insert 13



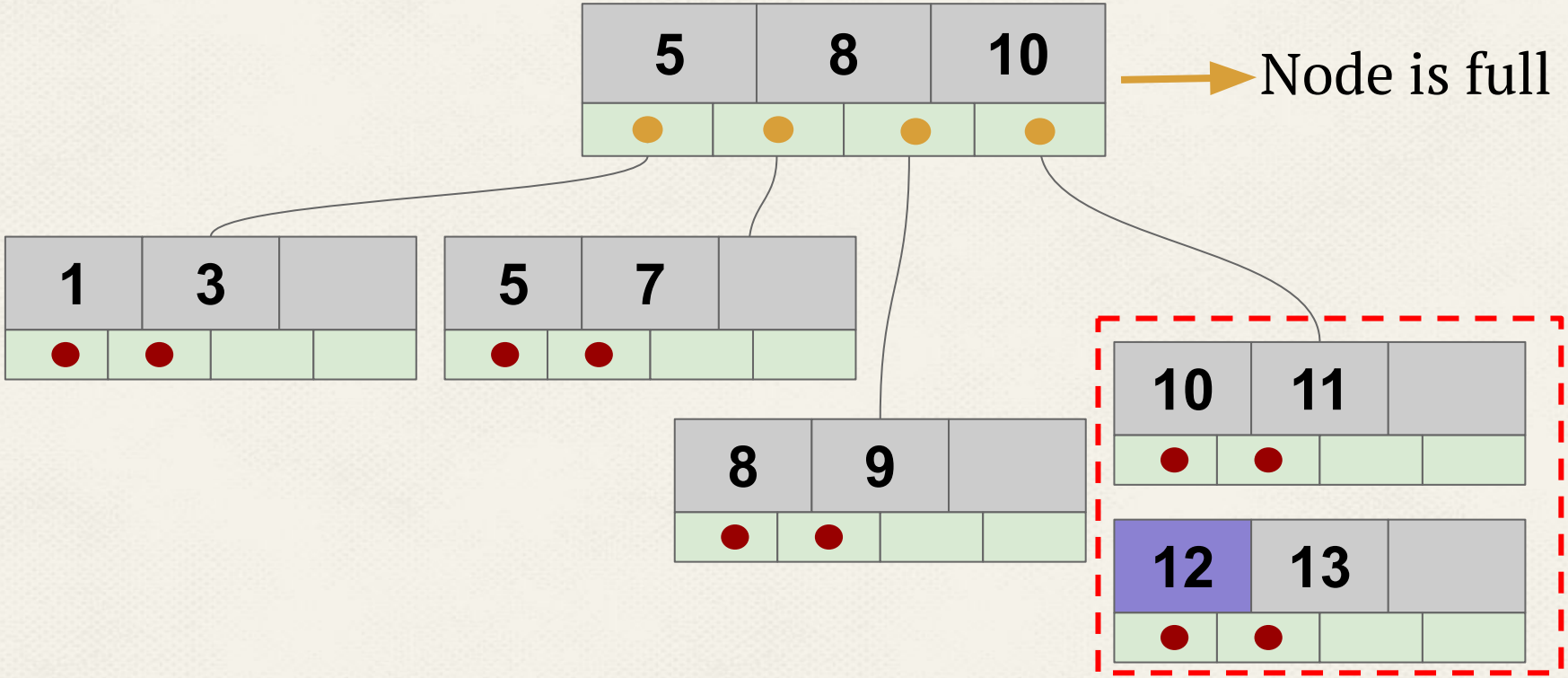
Insertion Algorithm
Case - 3: Insert 13



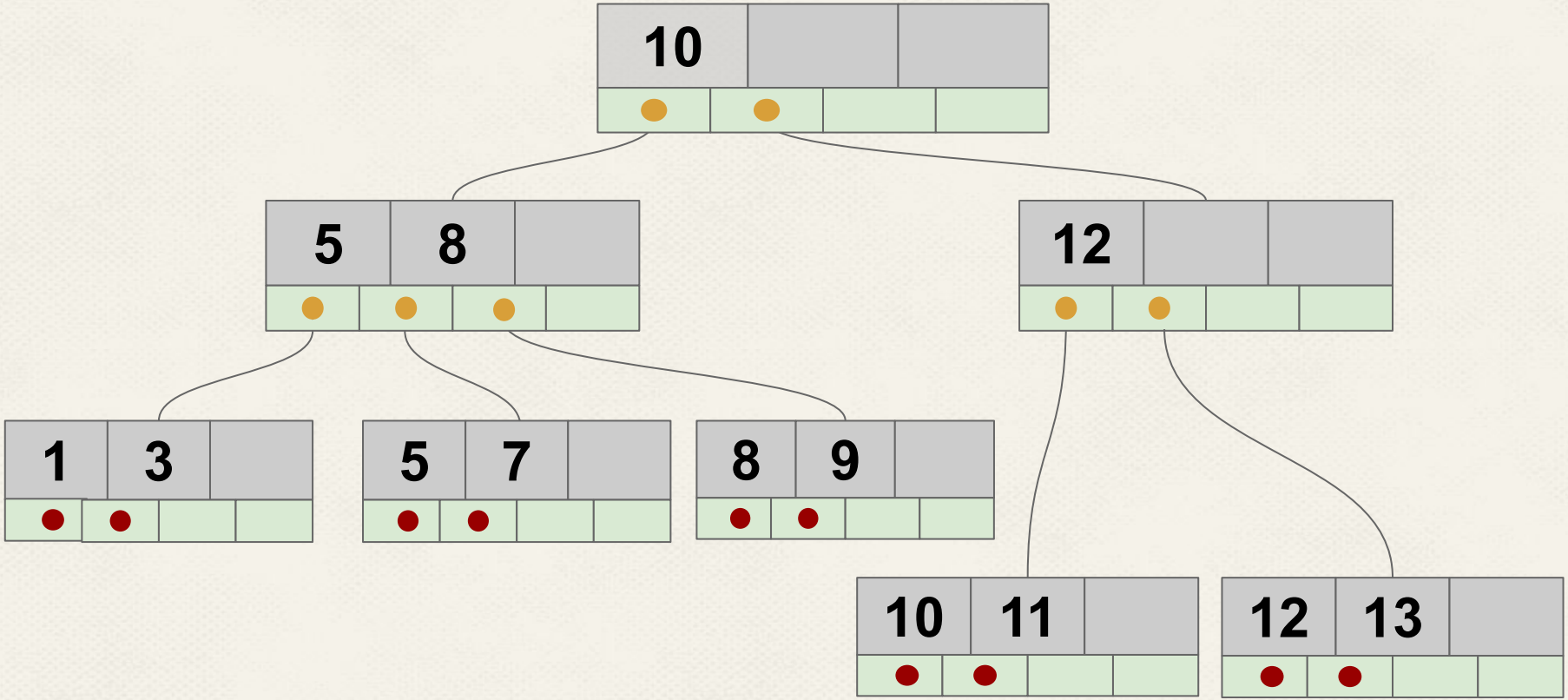
Insertion Algorithm
Case - 3: Insert 13



Insertion Algorithm
Case - 3: Insert 13



Insertion Algorithm
Case - 3: Insert 13



SEARCH

How do we find the data we want?

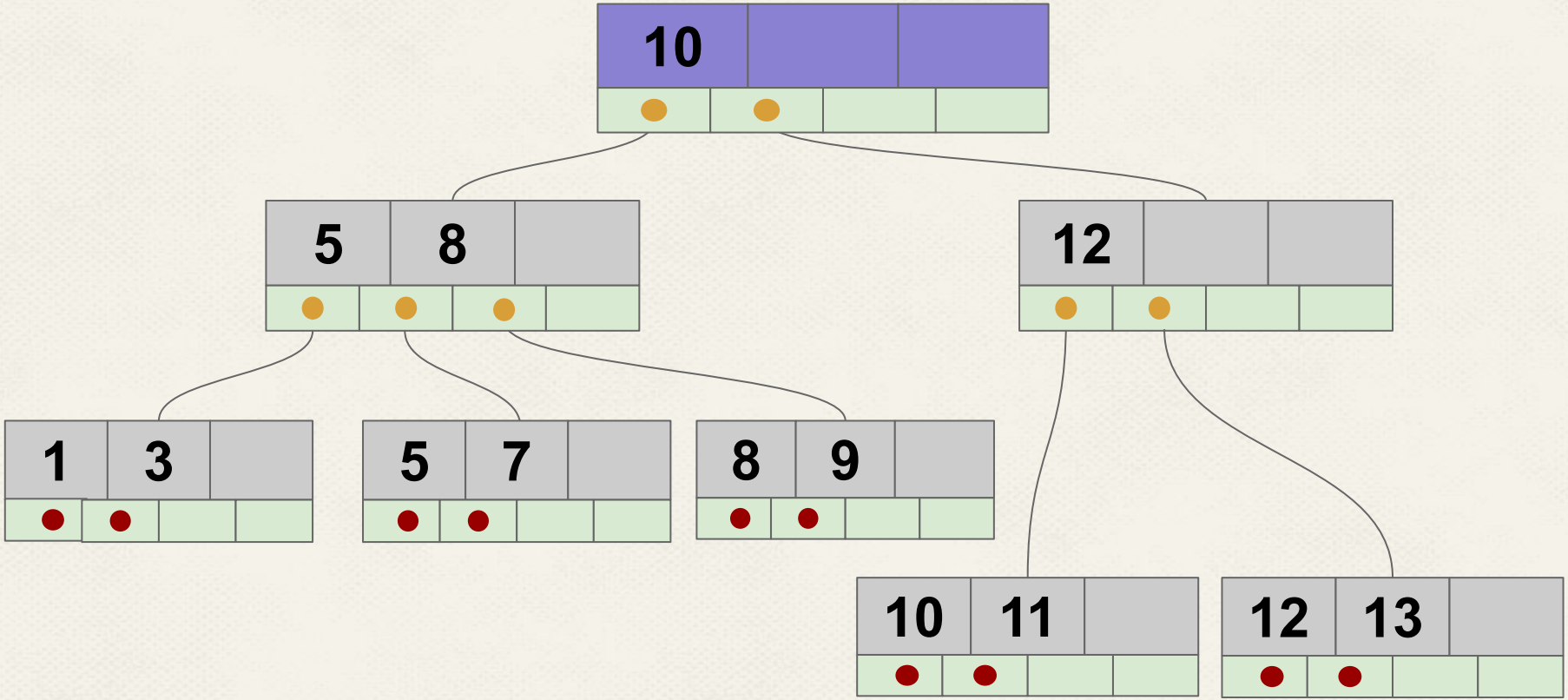
Search Algorithm

1. Perform binary search

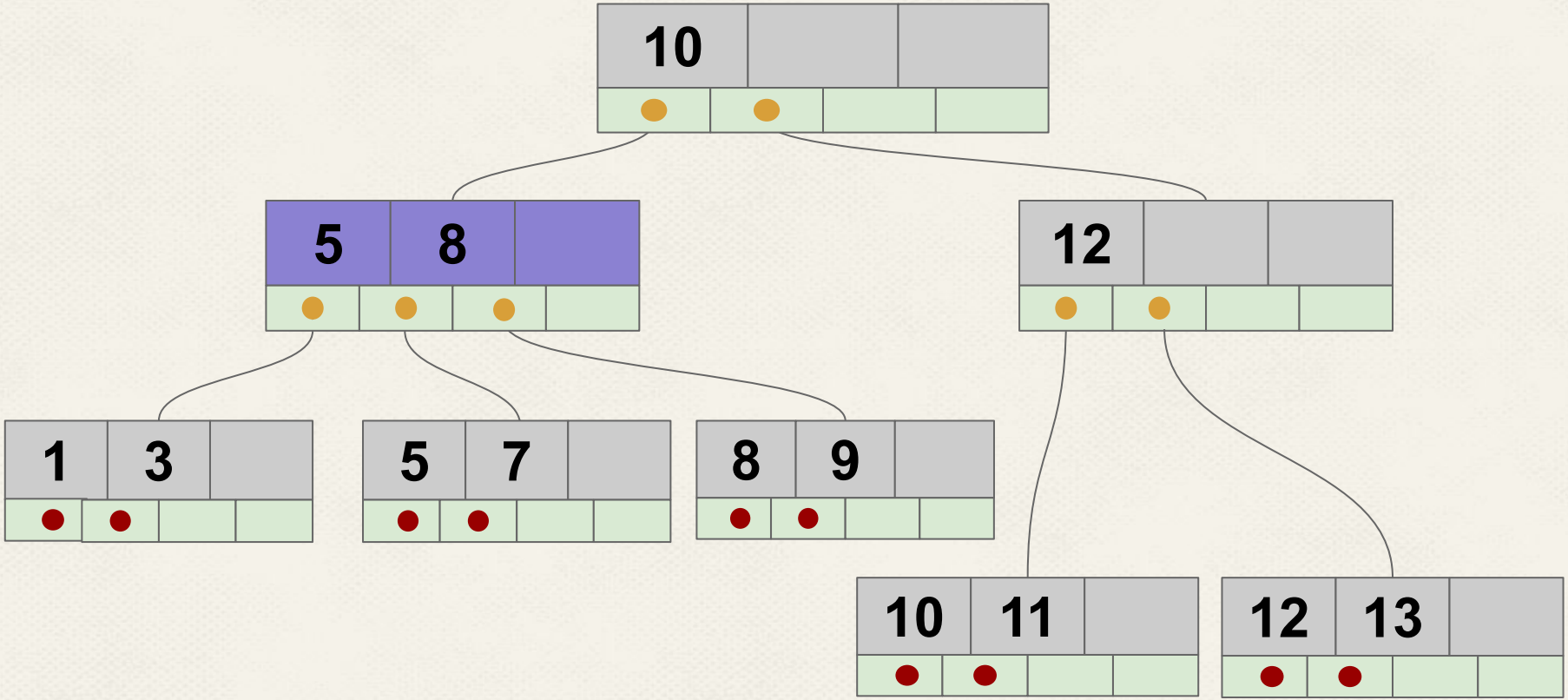
    a. If the key is found on a leaf node, the data can be directly retrieved.

    b. If the key is in an internal node, the branch must be traversed further.

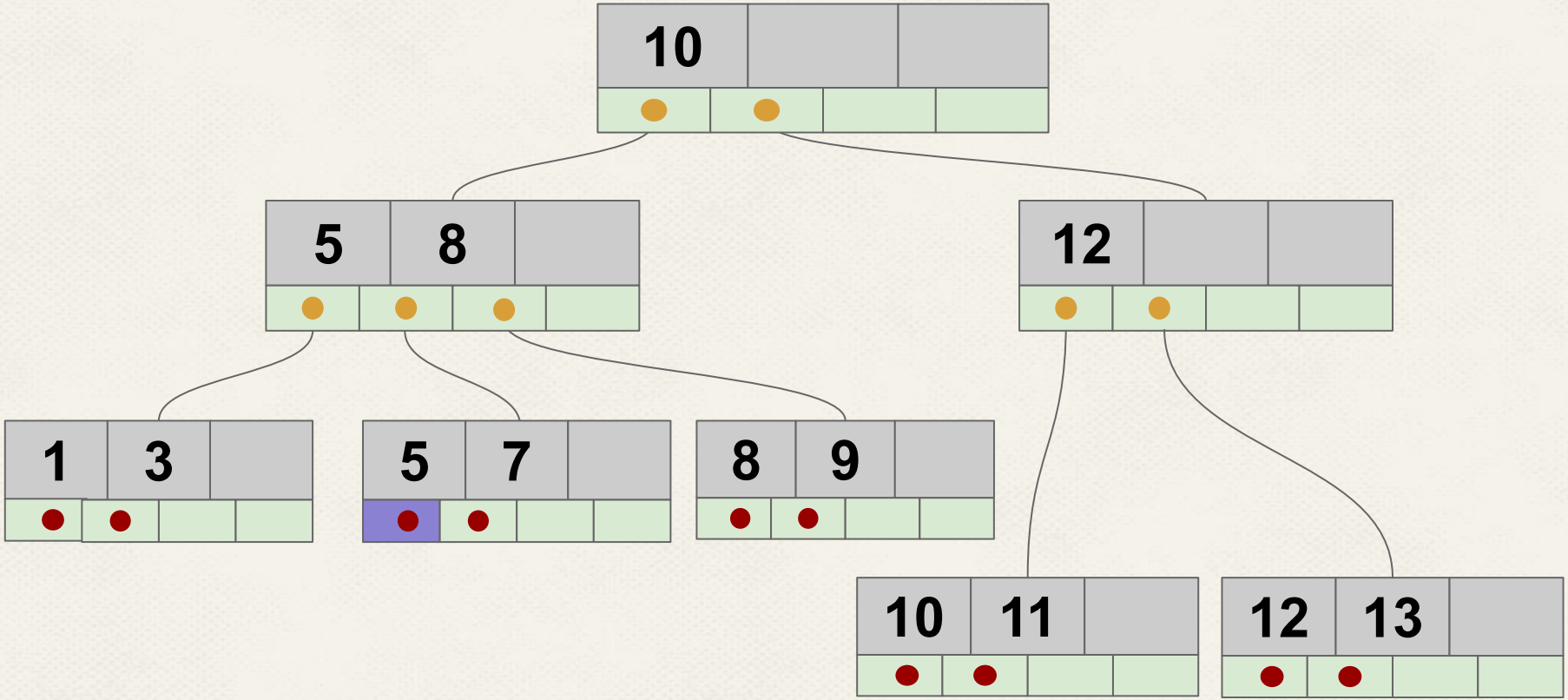
Search for 5



Search for 5



Search for 5



DELETE

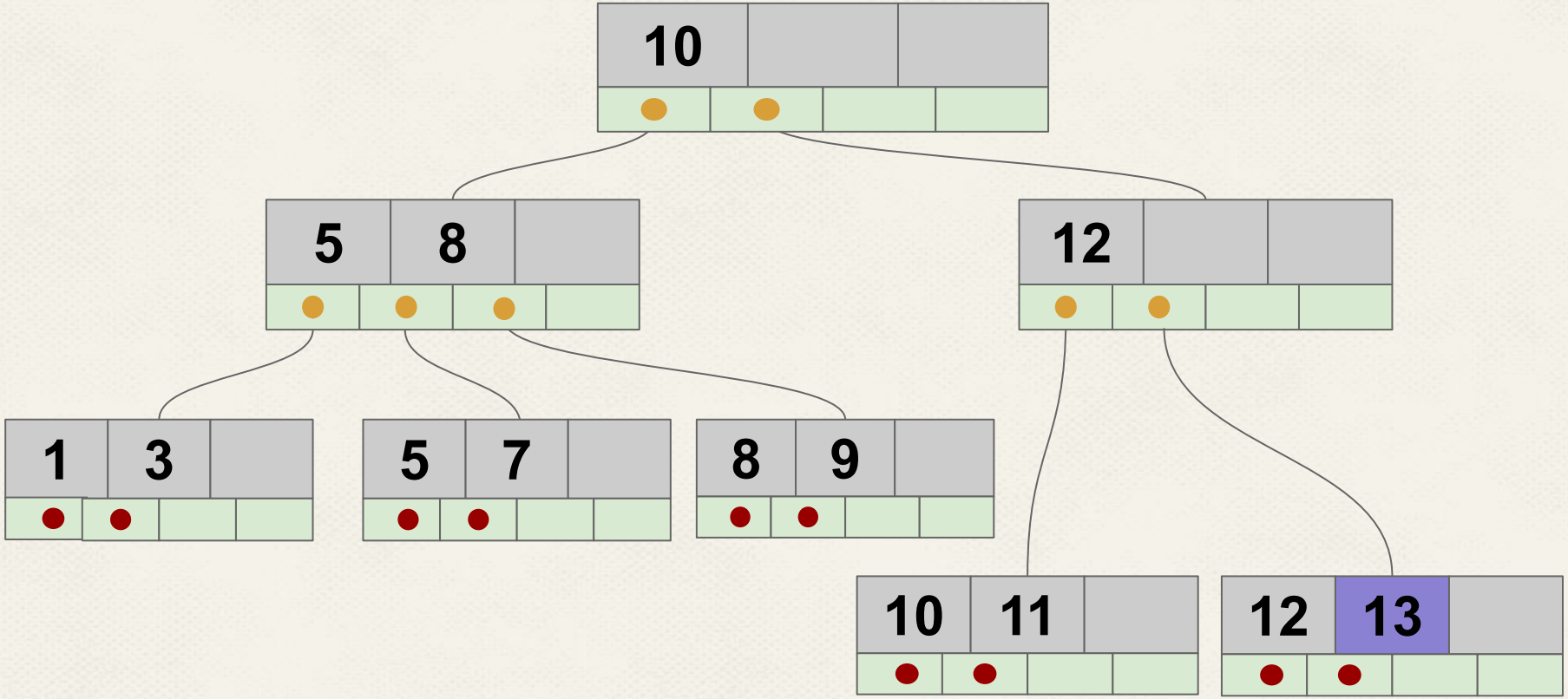
How do we delete a value

Delete Algorithm

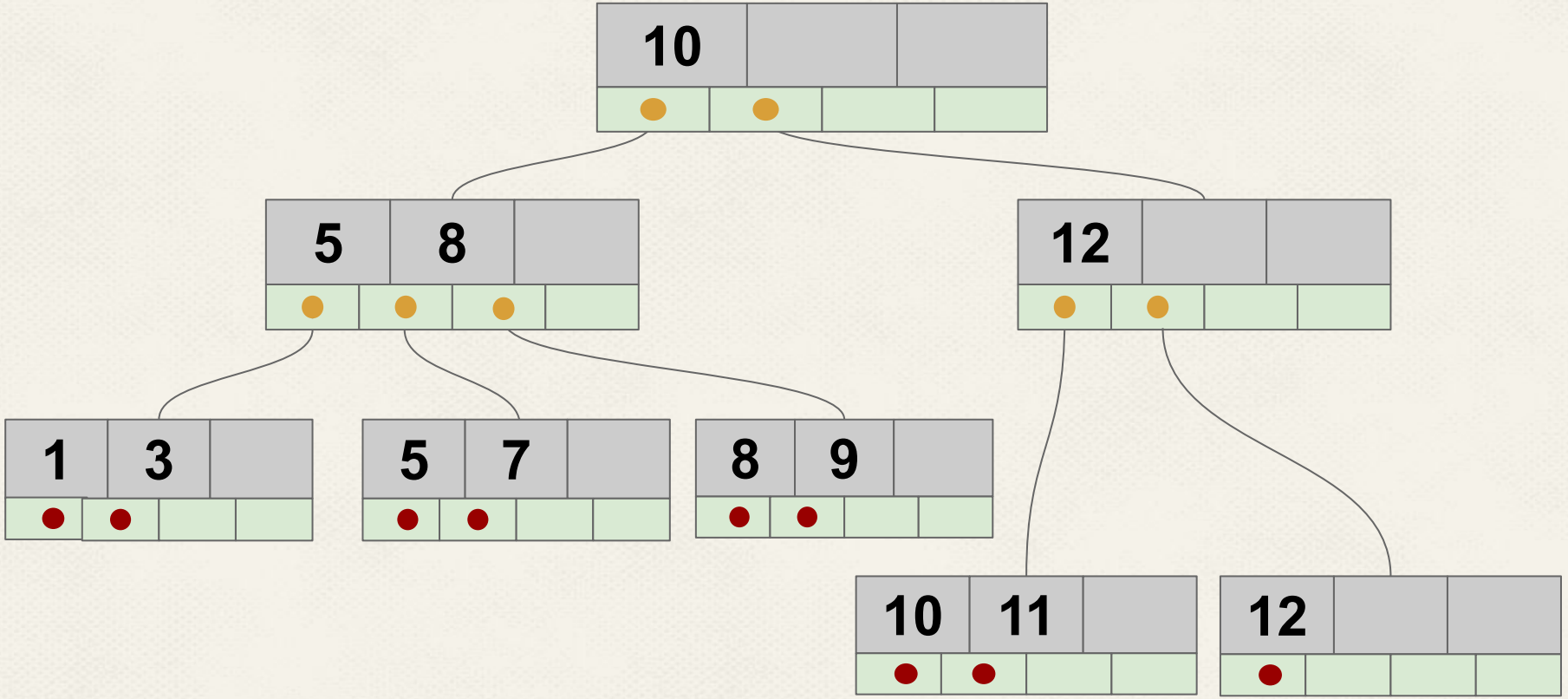
If deleting a key does not violate any of the B+ Tree properties

Key and associated data is removed from the node, values in the node and shift to remove any null spaces.

Delete Algorithm
Case-1: Delete 13



Delete Algorithm
Case-1: Delete 13



Delete Algorithm

If the deleted key appears in both an internal node and a leaf node

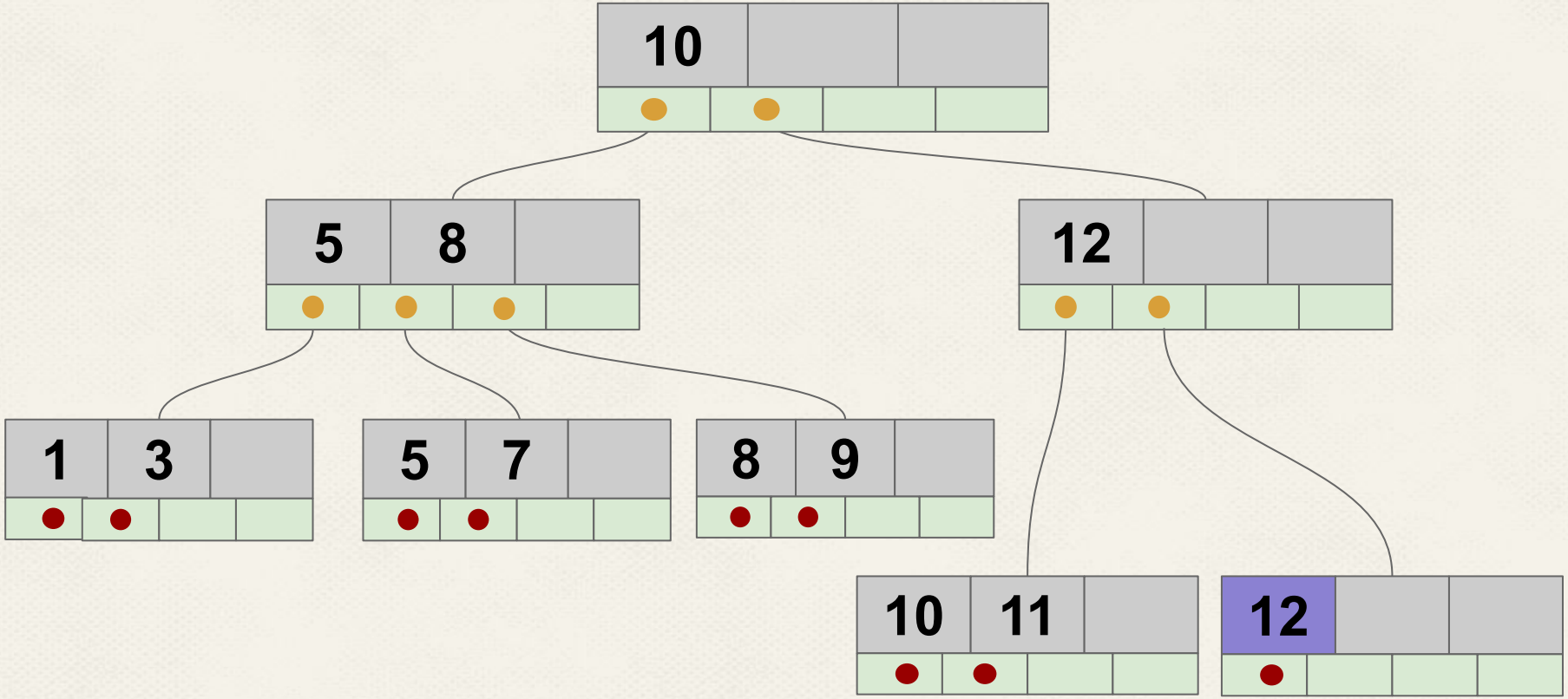
- The key must be deleted in the internal node must also be deleted

Delete Algorithm

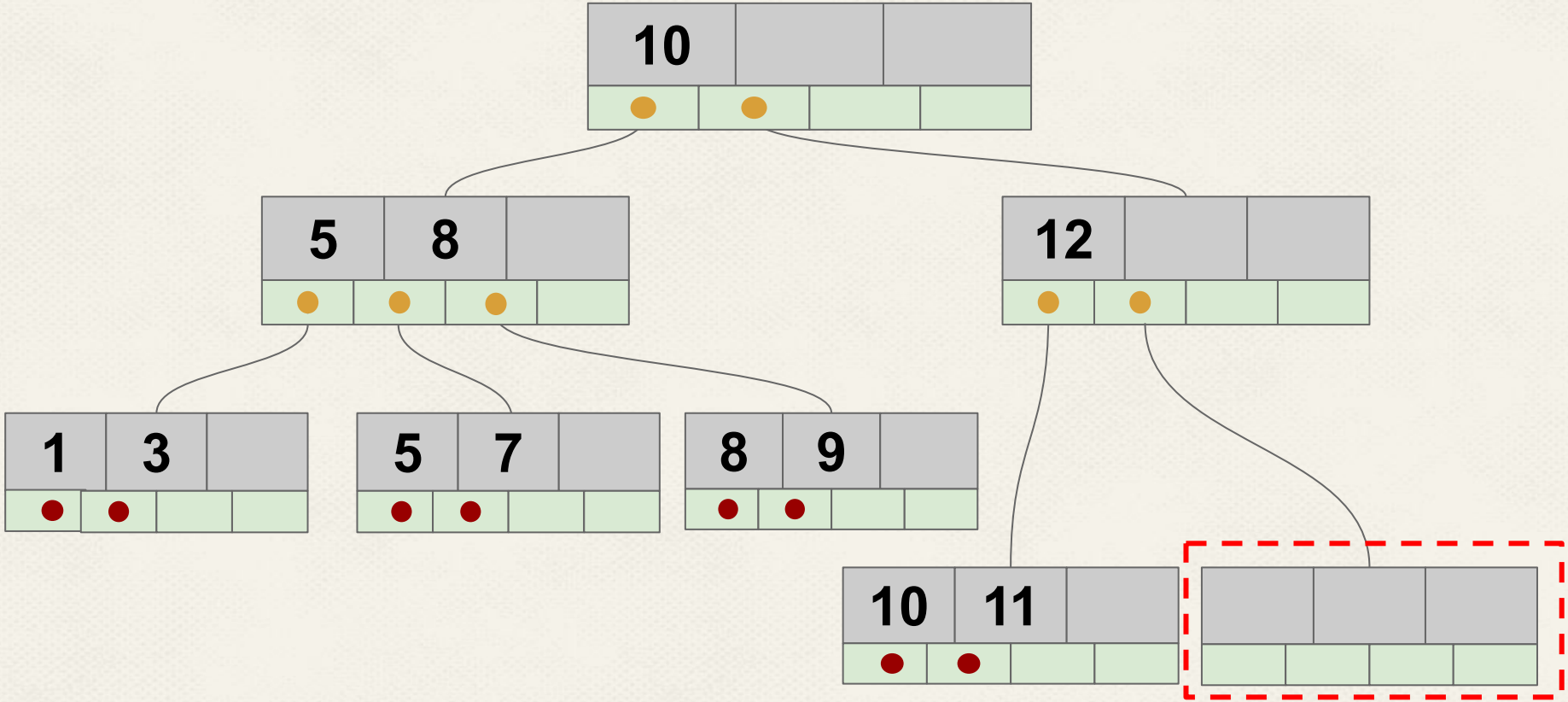
If deleting the key will lead to the node having fewer than d/2 keys

The nodes must be **redistributed** or **merged** such that no node (except the root) ---

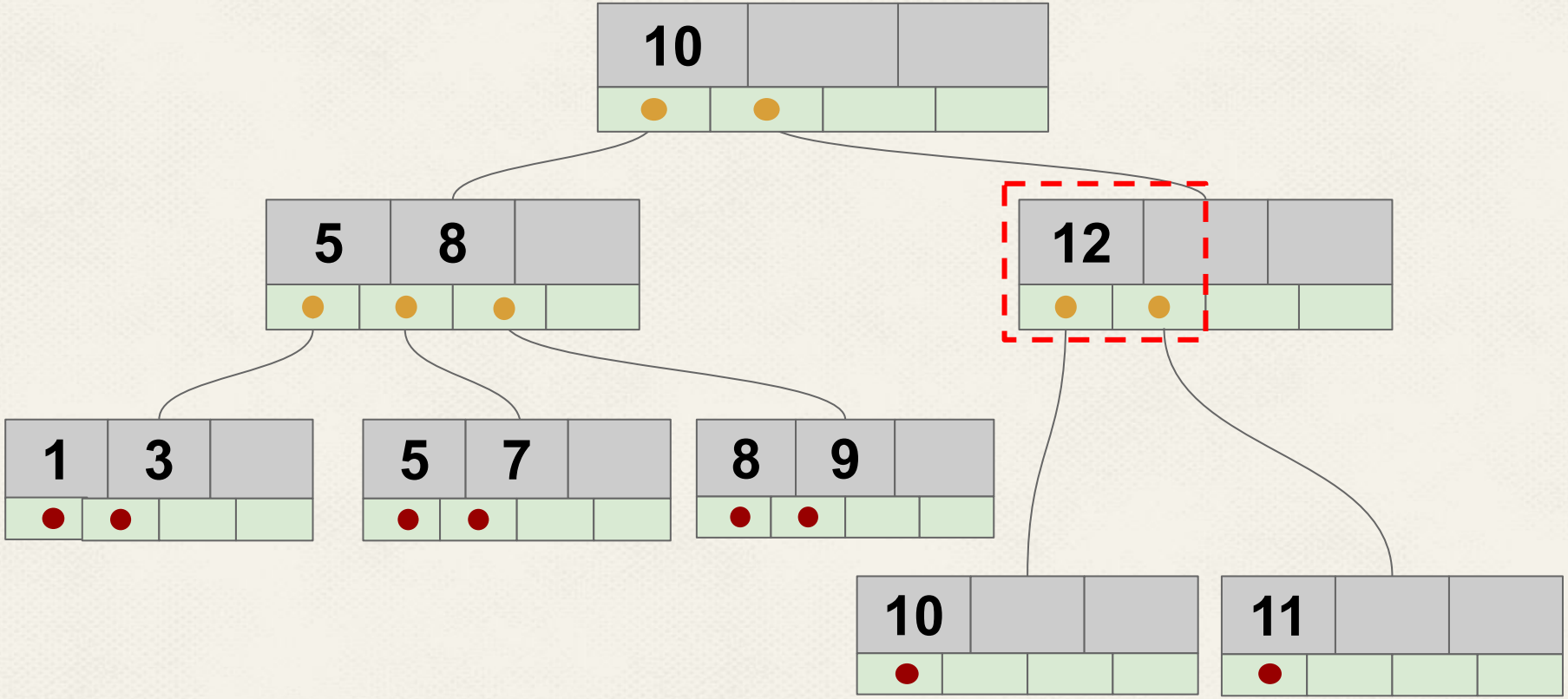
Delete Algorithm
Case-2: Delete 12



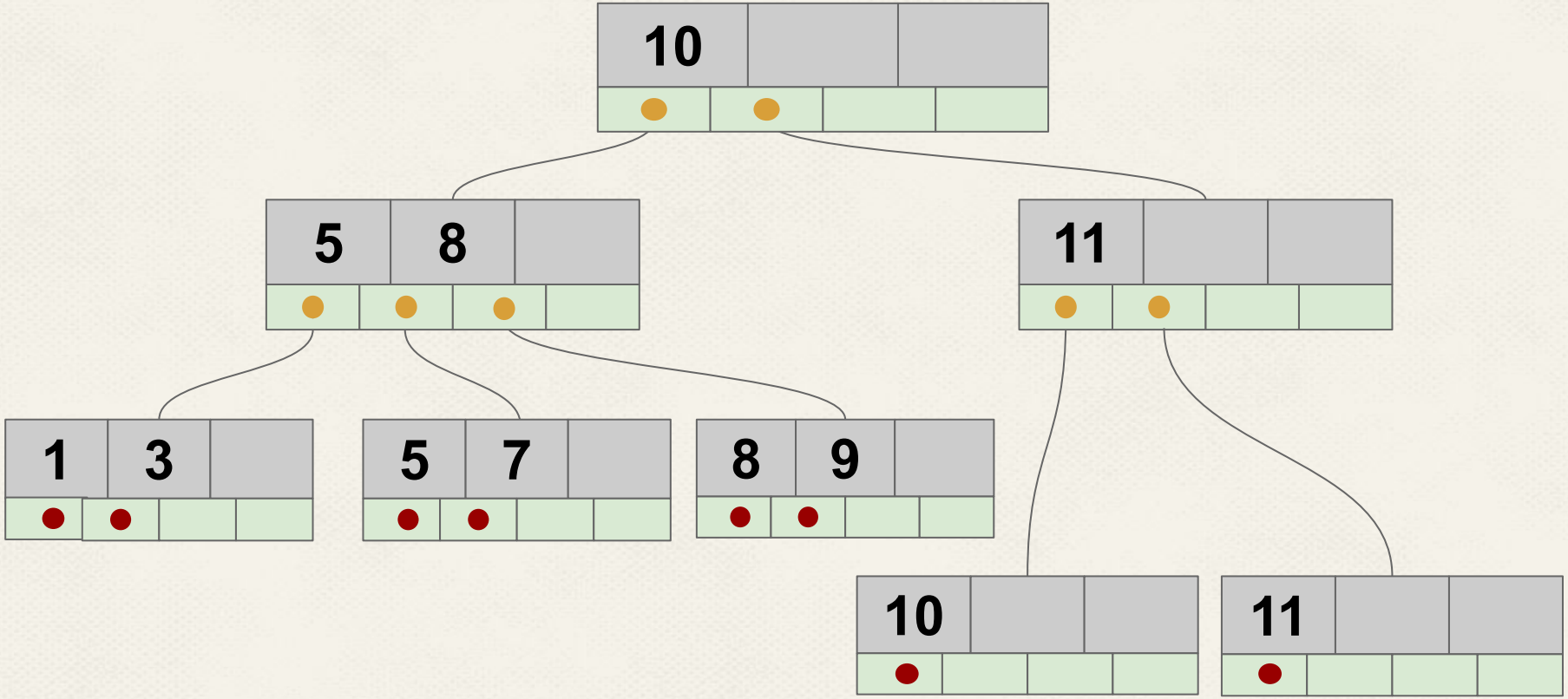
Delete Algorithm
Case-2: Delete 12



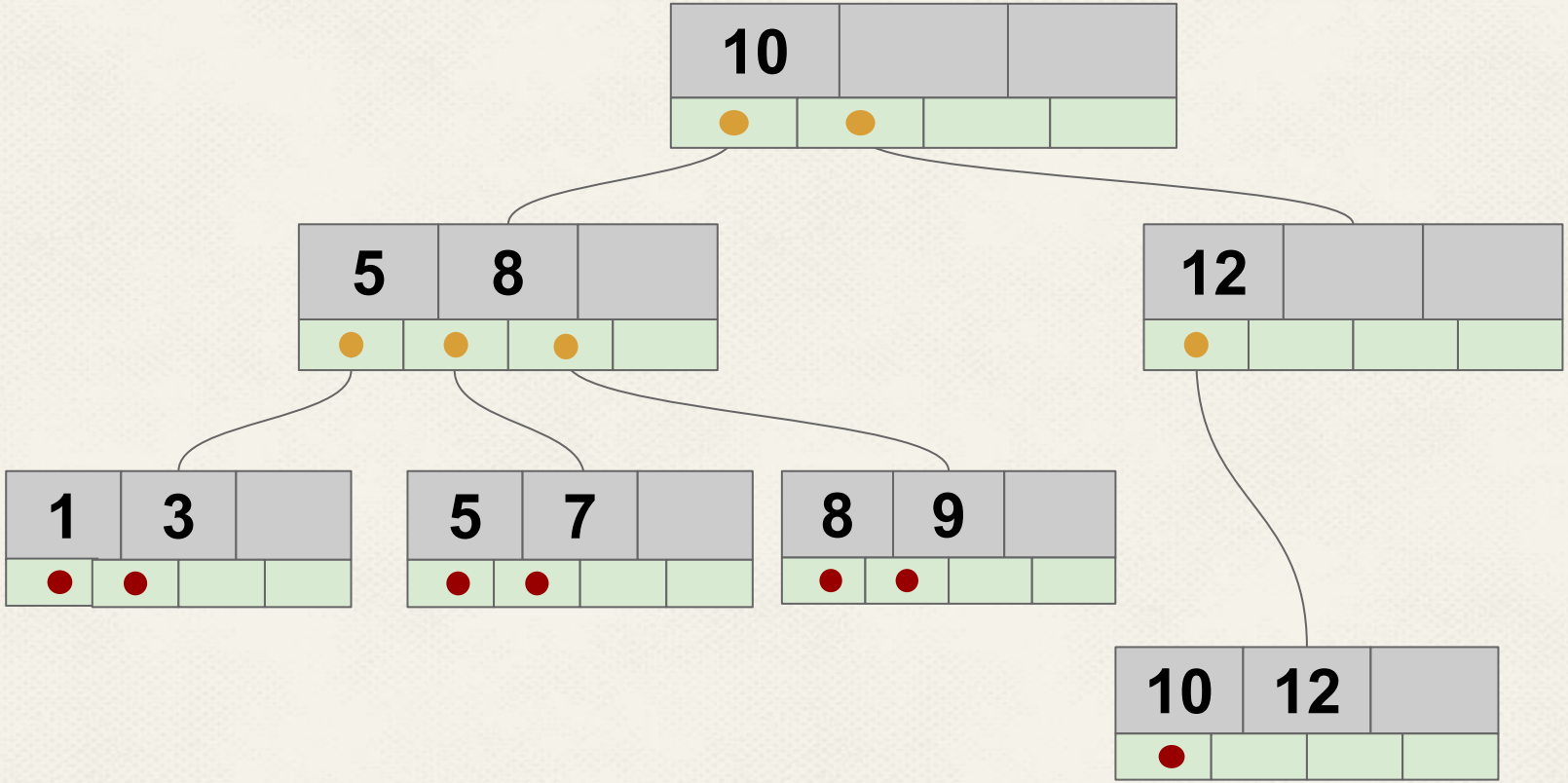
Delete Algorithm
Case-2: Delete 12



Delete Algorithm
Case-2: Delete 12



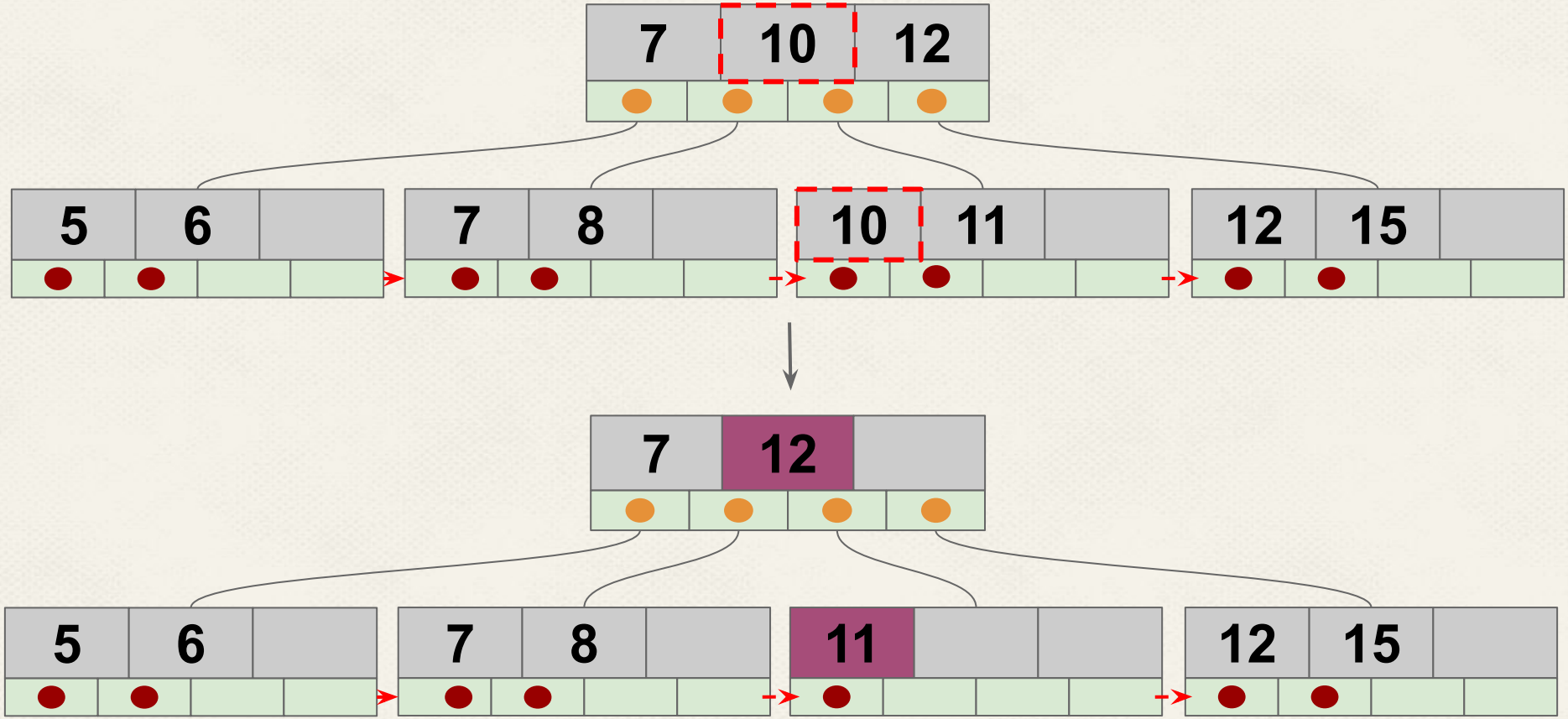
Delete Algorithm
Case-2: Delete 12



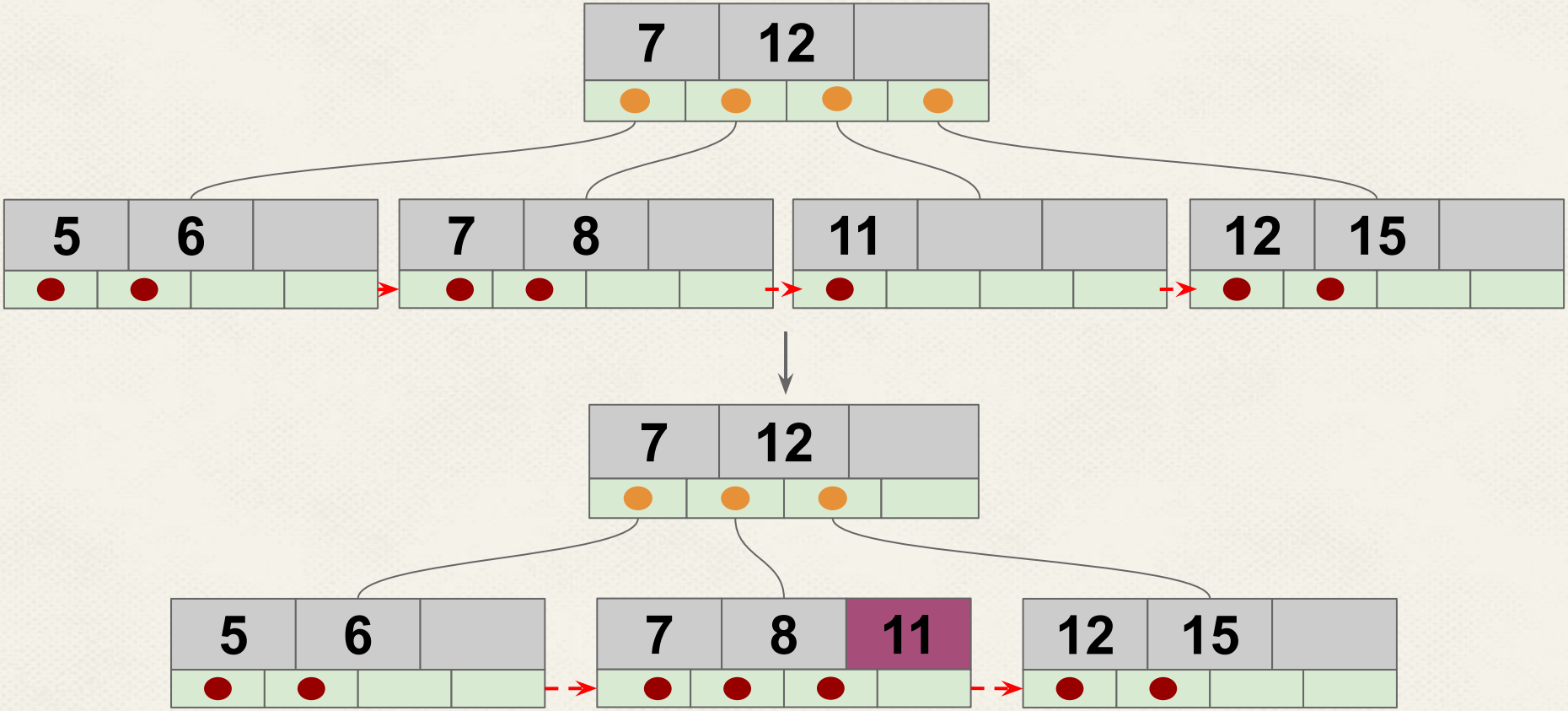
Thanks for listening!

Any questions?

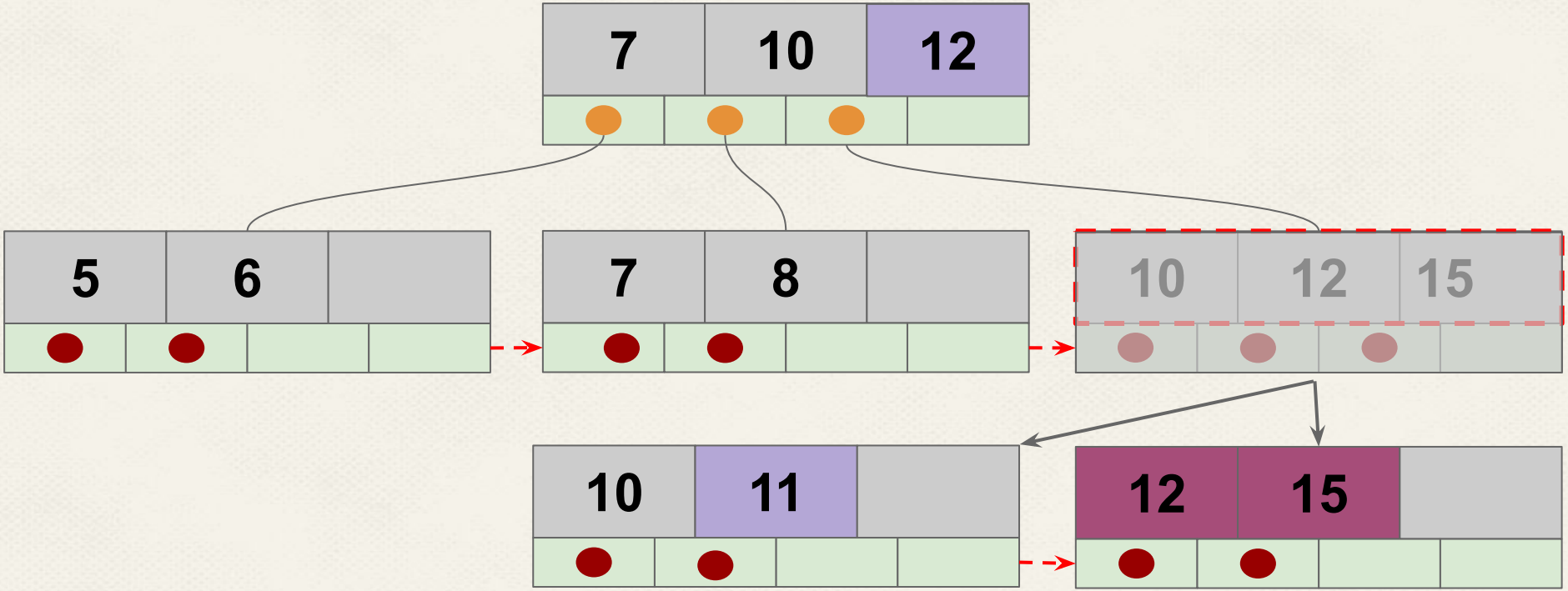
Deletion Algorithm



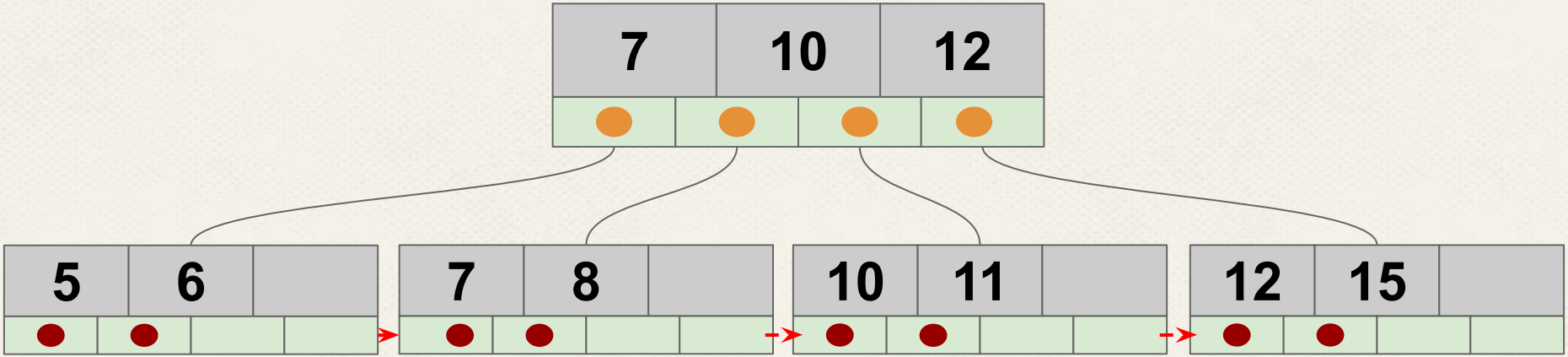
Deletion Algorithm (cont)



Insertion Algorithm



Insertion Algorithm (cont)



What is a B tree?

Self balancing binary search tree in which each node can have multiple children

1. The keys in a B-tree are always sorted

2. Every leaf of a B-tree is on the same level

3. A child between key-1 and key-2 will have keys greater than key-1 and less than key-2

What is a B tree?

Self balancing binary search tree in which each node can have multiple children

1. The keys in a B-tree are always sorted
2. Every leaf of a B-tree is on the same level
3. A child between key-1 and key-2 will have keys greater than key-1 and less than key-2

What is a B tree?

Self balancing binary search tree in which each node can have multiple children

1. The keys in a B-tree are always sorted
2. Every leaf of a B-tree is on the same level
3. A child between key-1 and key-2 will have keys greater than key-1 and less than key-2

What is a B+ Tree?

An extension of a B-tree

1. Data can only be stored in leaf nodes.
2. Leaf nodes are sometimes linked together.

B+ vs B Trees

1. Elements always deleted from leaf node.
   (Faster and simpler deletion)

2. Leaf nodes are linked.
   (Faster search operations)

3. Smaller height and remains balanced.