m-way search trees

Ch. 11.1-11.2

- **Ch 10.2 AVL tree**
- **Ch 11.2 B-tree**
  - 2-3 trees (B-tree of order 3)
  - 2-3-4 tree (B-tree of order 4)

We are here

- **Ch 10.3 Red-black tree** (An extension of 2-3-4 trees)
- **Ch 11.3 B^+-tree**

2-3 tree/2-3-4 tree/Red-Black tree

Degree of a node: the number of its children

Properties

- Root degree ≥ 2
- Degree of nonroot node: $\left\lceil \frac{m}{2} \right\rceil \sim m$
- External nodes should be at the same level.

Time complexity of operations

- Insertion: O(log *n*)
- Delete: O(log *n*)

M-way search tree

- A search tree
- Each node has up to **m-1** data pairs and **m** children.
  - m=2: a binary search tree

$$n, A_0, E_1, A_1, E_2, A_2, \ldots, E_n, A_n$$

- n: number of data pairs ($n < m$)
- A_i : pointer to a subtree
- E_i : a data pair (key, value)

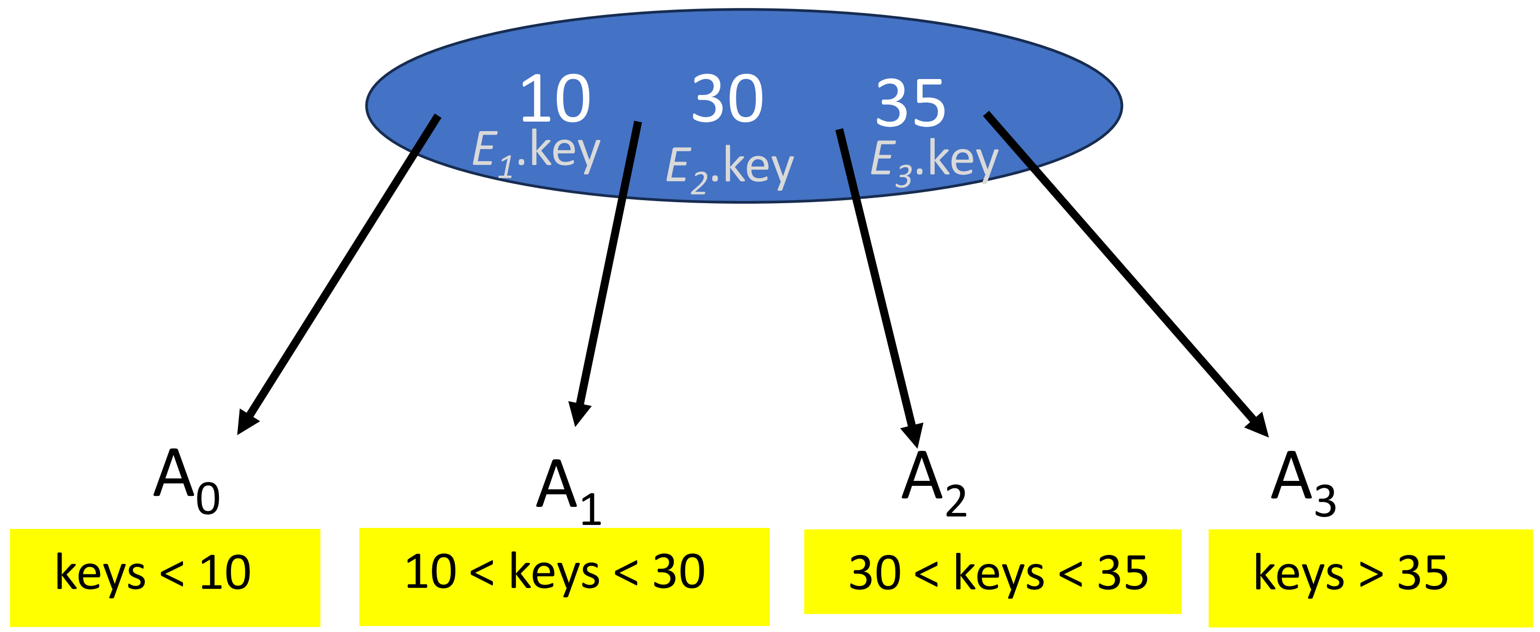
- Key of left data pair < key of right data pair

$$E_i.\text{key} < E_{i+1}.\text{key}$$

- E_i.key < All keys in the subtree A_i < E_{i+1}.key
- The subtrees are also m-way search trees.

Example of *m*-way search tree

- 4-way search tree (m=4)



Maximum number of data pairs

- Happens when all internal nodes are m-nodes.
- Full degree m tree

degree of node = m

$$\# \text{ of nodes} = 1 + m + m^2 + m^3 + \ldots + m^{h-1}$$
$$= (m^h - 1)/(m - 1).$$

Note: Sum of geometric progression

- Each node has $m - 1$ data pairs

➔ Number of data pairs = $m^h - 1$

Capacity of m-way search tree

	m = 2	m = 200
h = 3	7	$8 * 10^6 - 1$
h = 5	31	$3.2 * 10^{11} - 1$
h = 7	127	$1.28 * 10^{16} - 1$

- To achieve best performance of m-way search tree, the search tree should be balanced.

B-tree and B^+-tree

Definition of B-Tree

- Assume the tree has external nodes. (extended *m*-way search tree)
- B-tree of order *m*
  - *m*-way search tree
  - Empty or satisfying these properties:
    - Root degree ≥ 2
    - Degree of other internal nodes ≥ $\left\lceil \frac{m}{2} \right\rceil$
    - External nodes on the same level
- B denotes "Balanced"

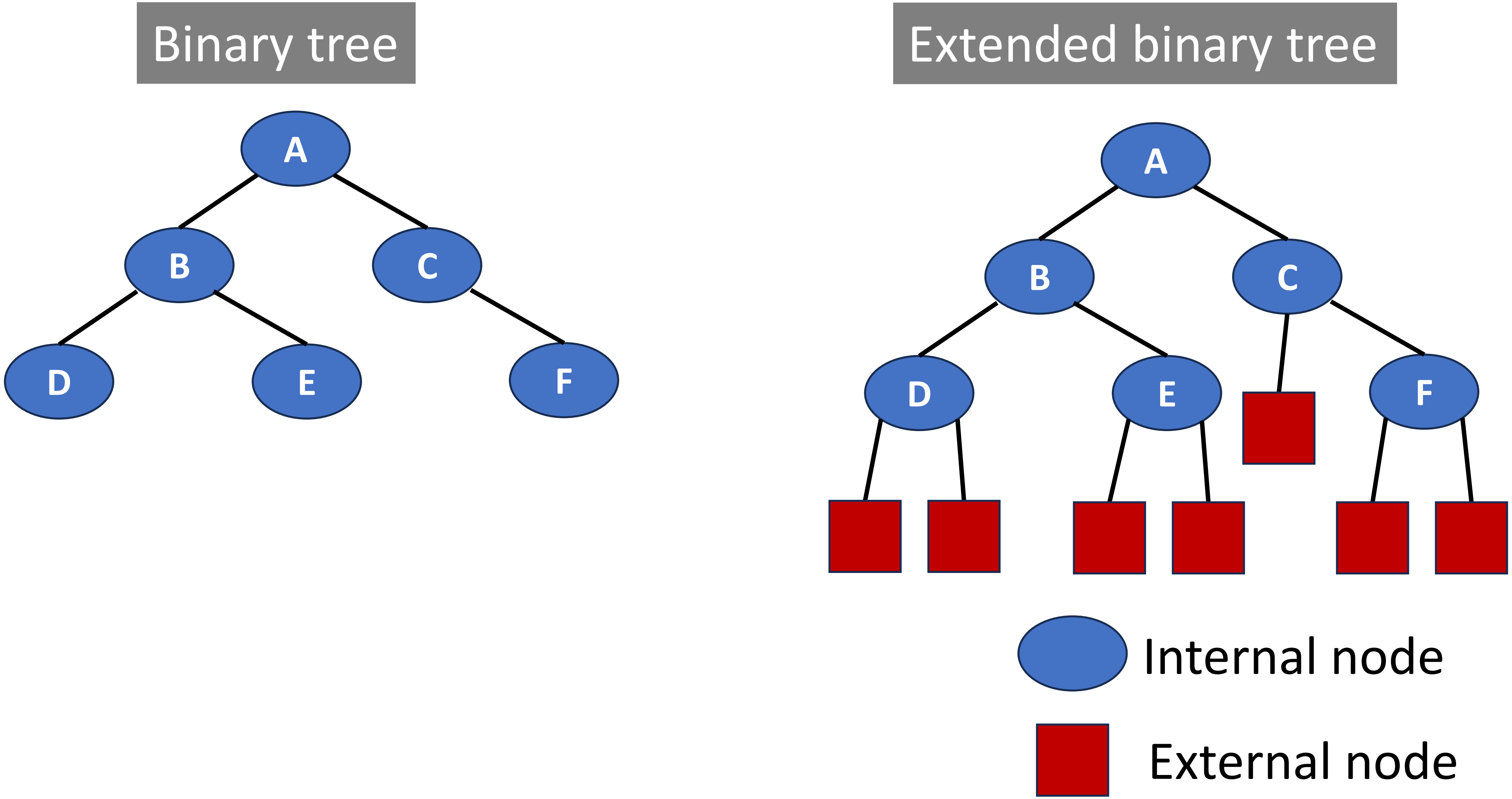
At least $\left\lceil \frac{m}{2} \right\rceil$ children and $\left\lceil \frac{m}{2} \right\rceil$-1 data pairs

At least half of the space in an internal node is utilized.

A balanced tree

Recall of extended binary tree

- Introduced in the lecture of leftist tree



B-tree of order 2 to 5

- 2-3 tree is **B-tree** of order 3 (m=3).
  - The internal nodes are either 2-node or 3-node.

degree of node = 2

degree of node = 3

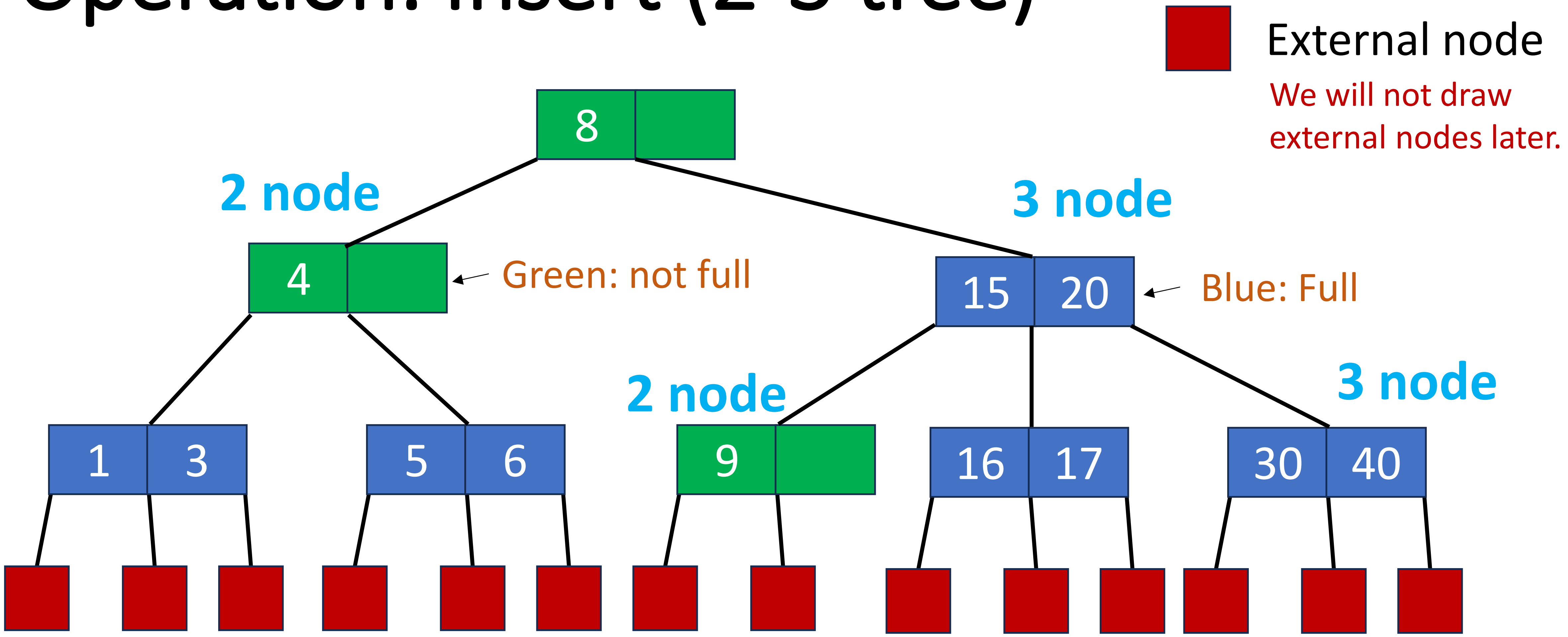
- 2-3-4 tree is **B-tree** of order 4 (m=4).
  - The internal nodes are 2-node, 3-node, or 4-node.
- 3-4-5 tree is **B-tree** of order 5 (m=5).
  - The internal nodes are 3-node, 4-node, or 5-node.
  - The root may be a 2-node.

$\left\lceil \frac{m}{2} \right\rceil$=3, so degree of 2 is not permissible.

- **B-tree** of order 2 (m=2) is full binary tree.

All external nodes on the same level

Operation: Insert (2-3 tree)



- Insertion into a full leaf triggers bottom-up node splitting pass.

Split an overfull node

- After insertion, the node *p* has *m* data pairs ($n=m$).

$$m, A_0, E_1, A_1, E_2, A_2, \ldots, E_m, A_m$$

- ○ A_i : pointer to a subtree
- ○ E_i : a data pair (key, value)

  - The number of data pairs should be at most m-1.

- Spit the node *p* into to two nodes *p* and *q*
  Then insert ($E_{\lceil m/2 \rceil}$, a *pointer to q*) into parent node.

$$m, \underbrace{A_0, E_1, A_1, \ldots, E_{\lceil m/2 \rceil -1}, A_{\lceil m/2 \rceil -1}}_{p \text{ node}}, E_{\lceil m/2 \rceil}, \underbrace{A_{\lceil m/2 \rceil}, E_{\lceil m/2 \rceil +1}, \ldots, E_m, A_m}_{\text{New } q \text{ node}}$$

p node

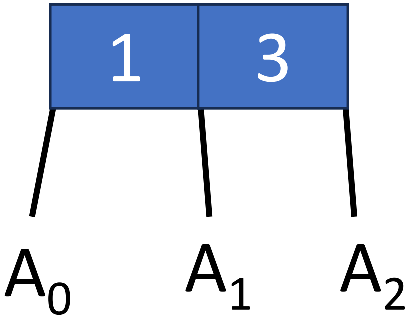
$\lceil m/2 \rceil$-1 data pairs

New *q* node

m-$\lceil m/2 \rceil$ data pairs

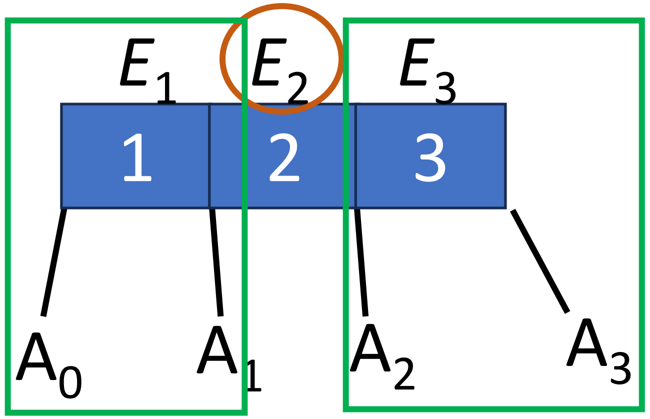
Example

- A 3-node



- Insert 2

p node



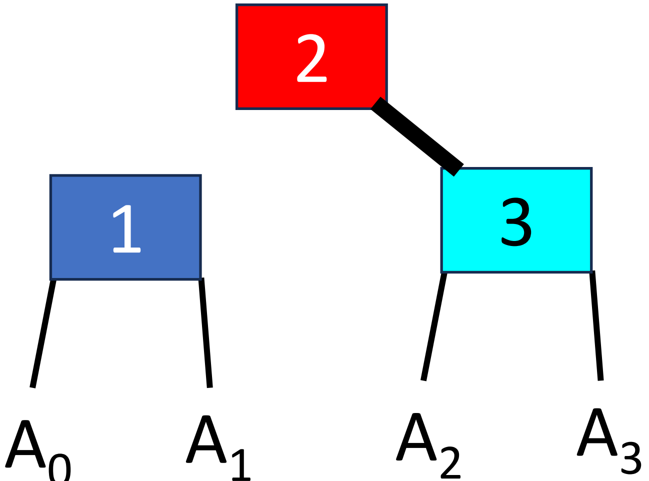
New *q* node

The data pair that will be moved to parent node.

$E_{\lceil m/2 \rceil} = E_{\lceil 3/2 \rceil} = E_2$

- Split

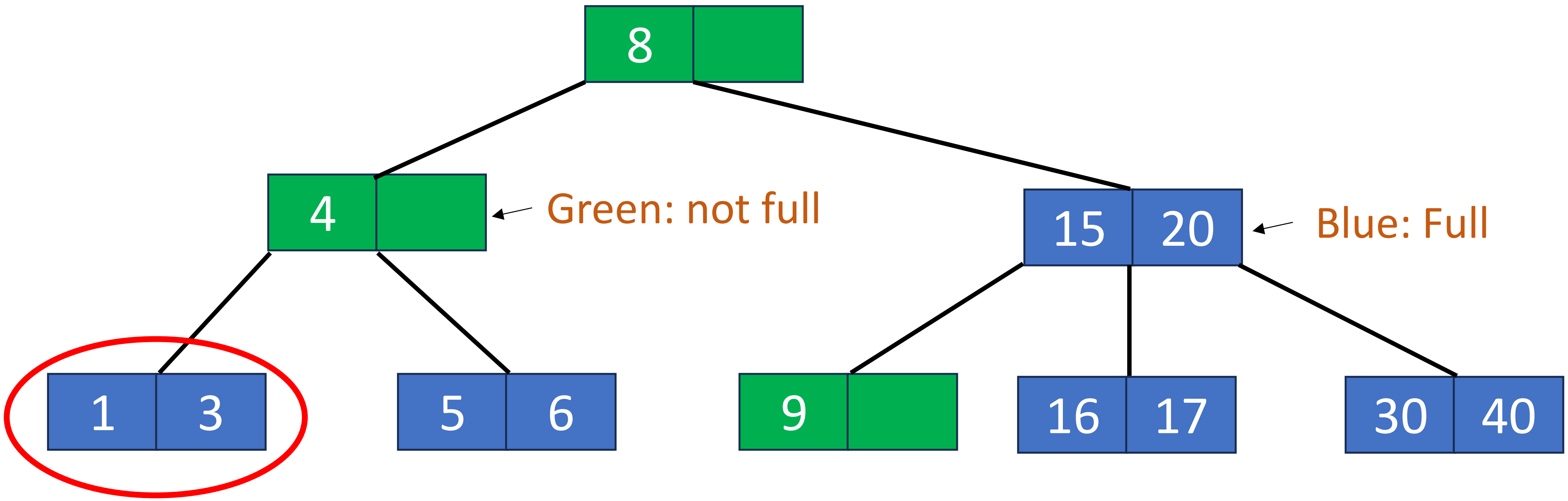
New node



Old node
// Store first 1/2 of old data and child pointers

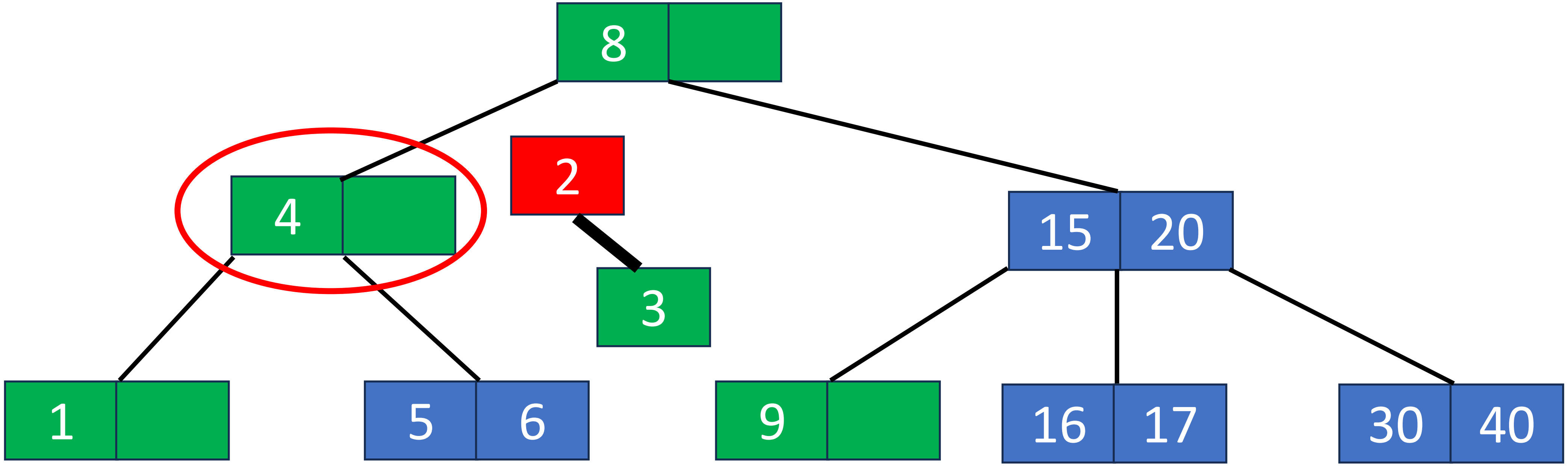
New node
// Store second 1/2 of old data and child points

Example: Insert (2-3 tree)



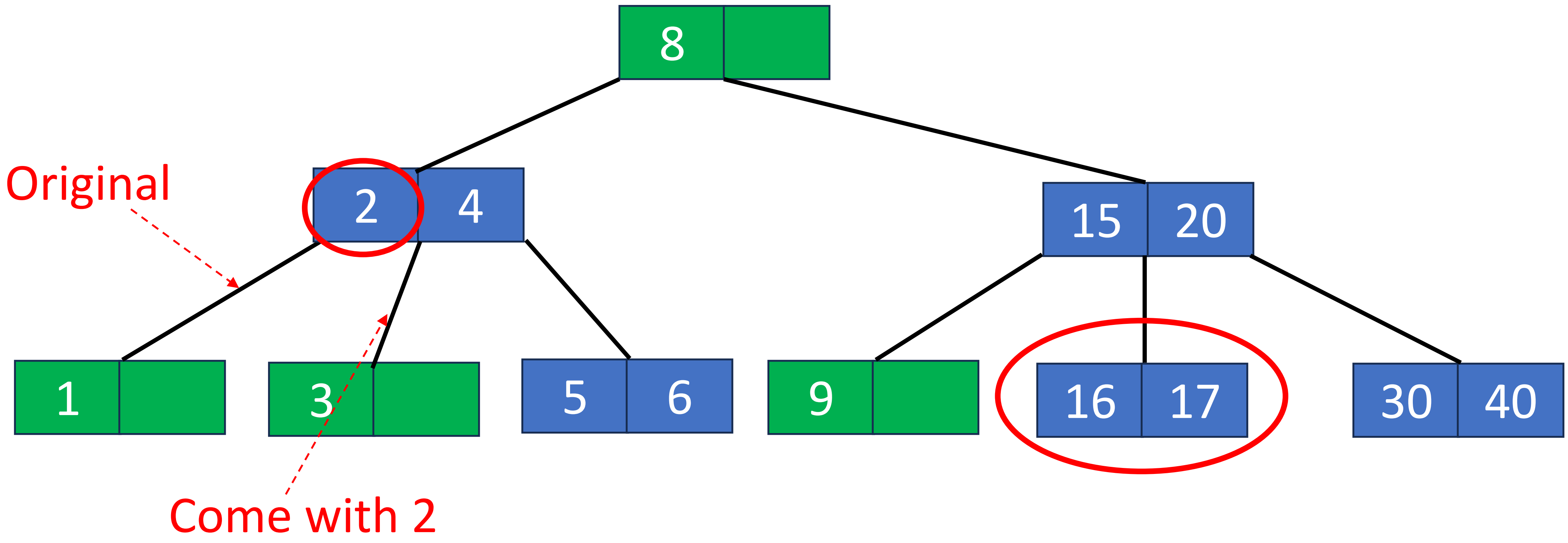
- Insert a pair with key = 2.
- New pair goes into a 3-node and causes overfull.

Example: Insert (2-3 tree)



- Split the overfull leaf node.
- Insert (the pair with key = 2, a pointer) to the parent.

Example: Insert (2-3 tree)

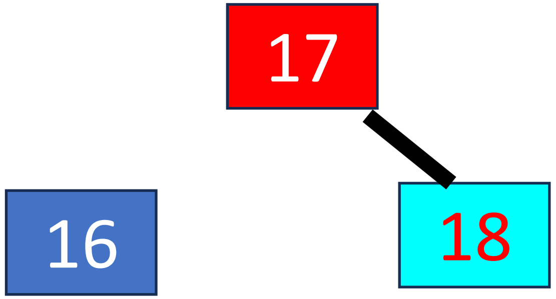


- Then, let's insert a pair with key =18

Insert into a leaf 3-node

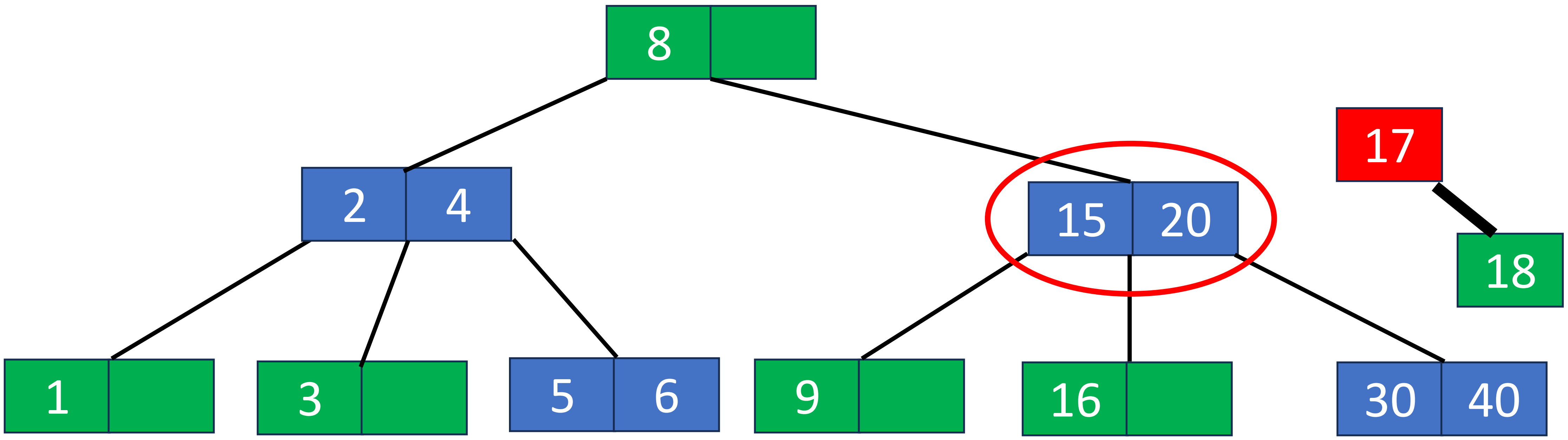
- Insert new pair so that the 3 keys are in ascending order.

- Split the overfull node.



- Insert the middle key and the pointer to the new node into parent

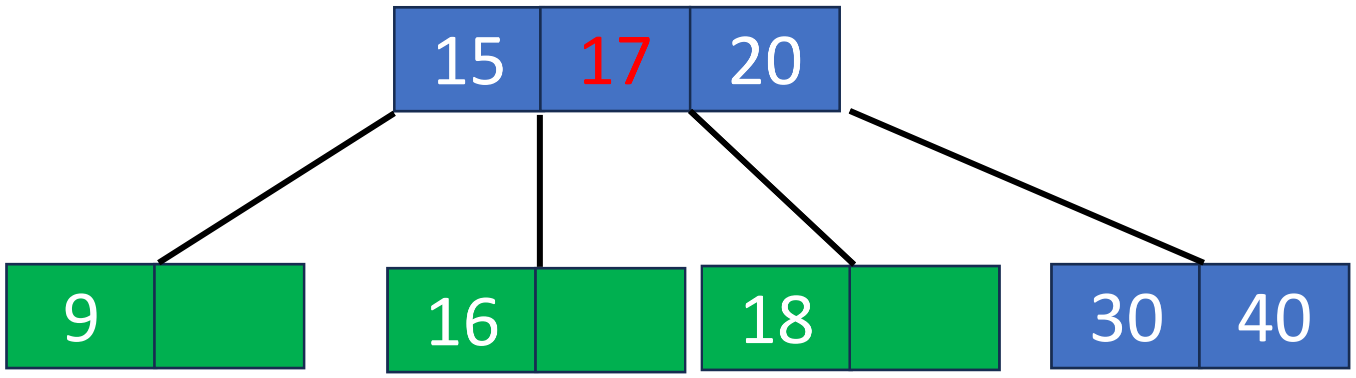
Example: Insert (2-3 tree)



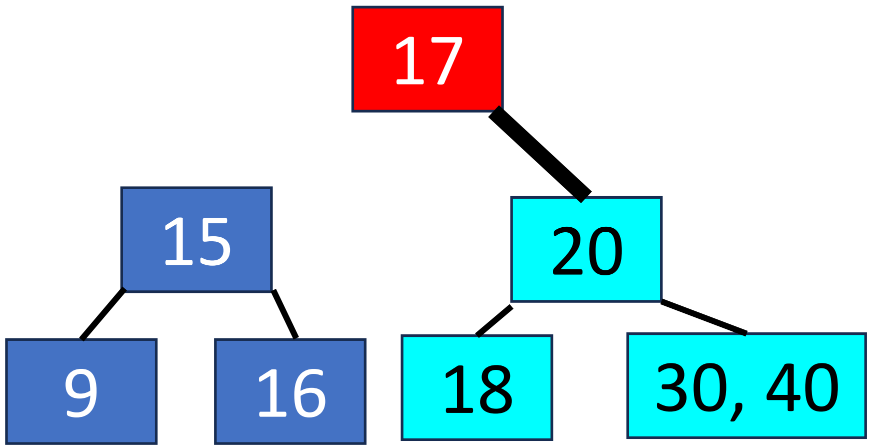
- Then, insert the data pair with key=17 and the pointer to the new node into parent

Insert into a 3-node

- Insert new pair so that the 3 keys are in ascending order.

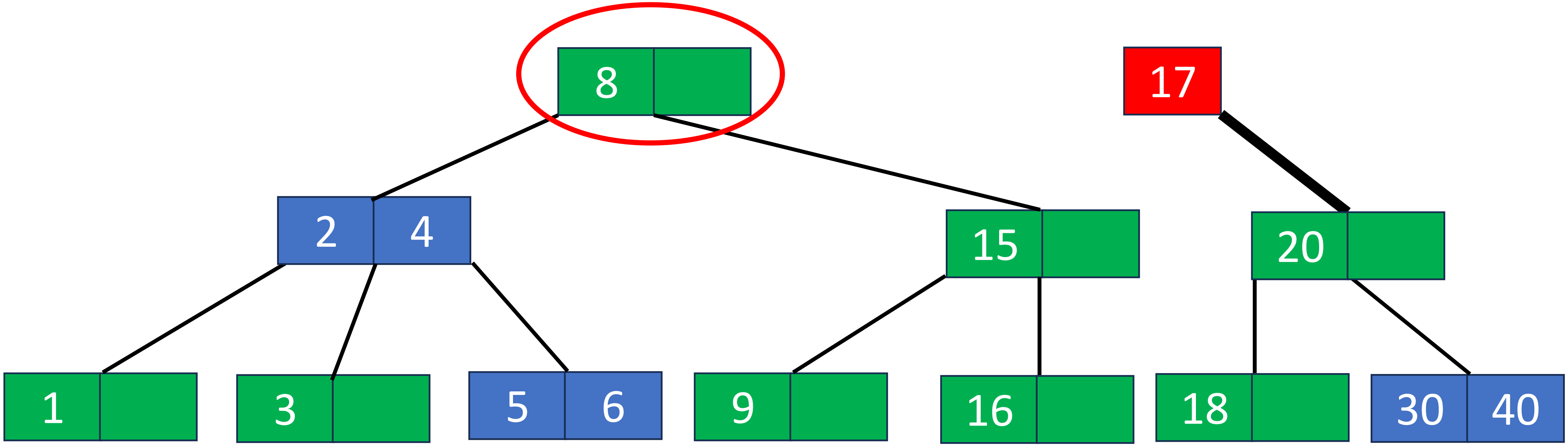


- Split the overfull node.



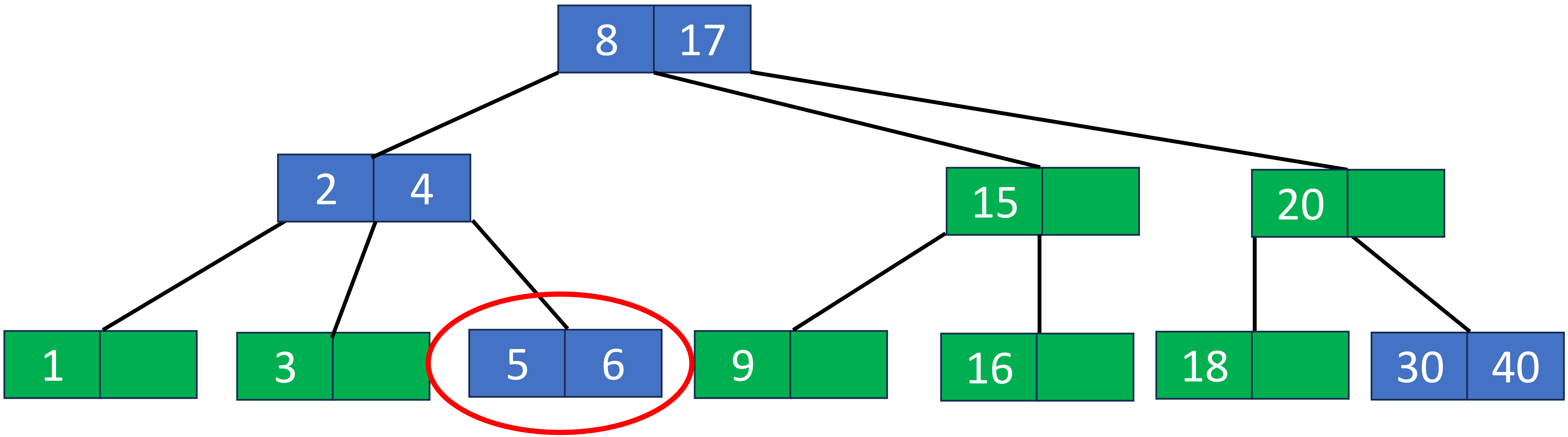
- Insert the middle key and the pointer to the new node into parent

Example: Insert (2-3 tree)



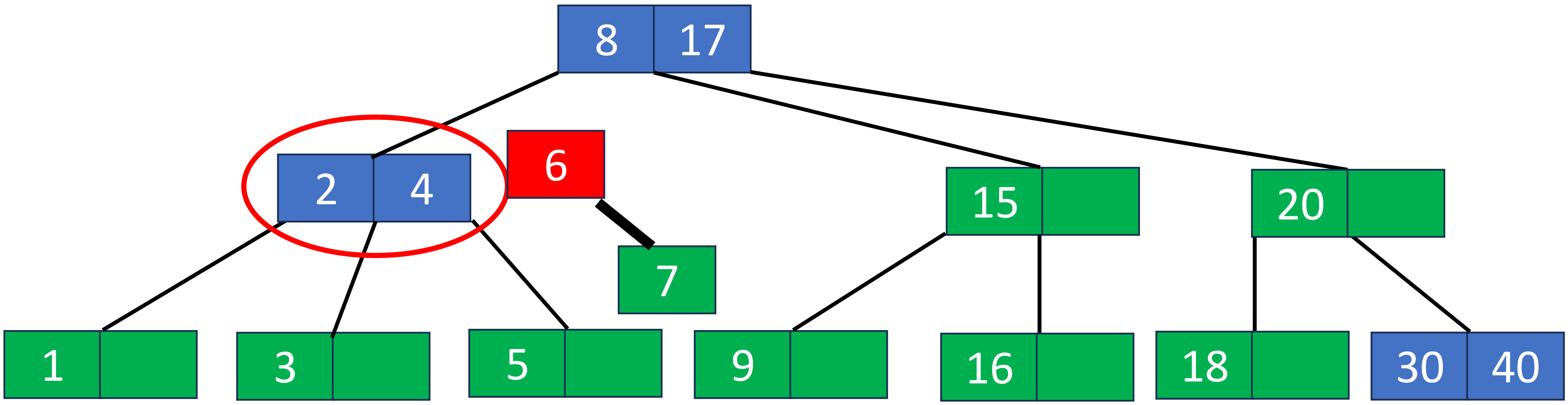
- Then, insert the data pair with key=17 and the pointer to the new node into parent

Example: Insert (2-3 tree)



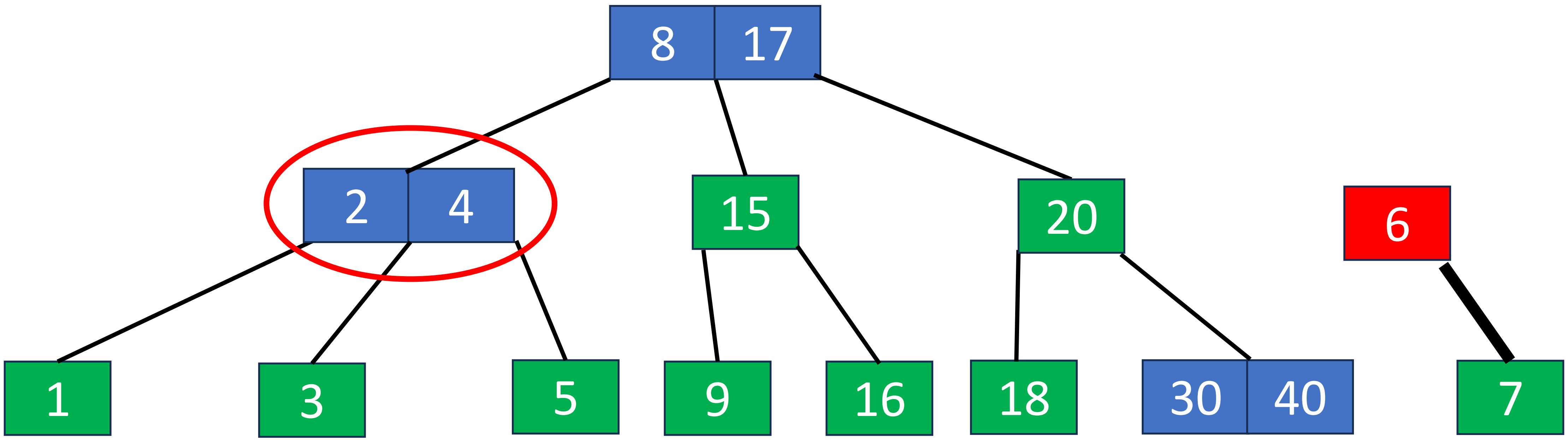
- Let's insert a pair with key=7.

Example: Insert (2-3 tree)



- Then, insert the data pair with key=6 and the pointer to the new node into parent

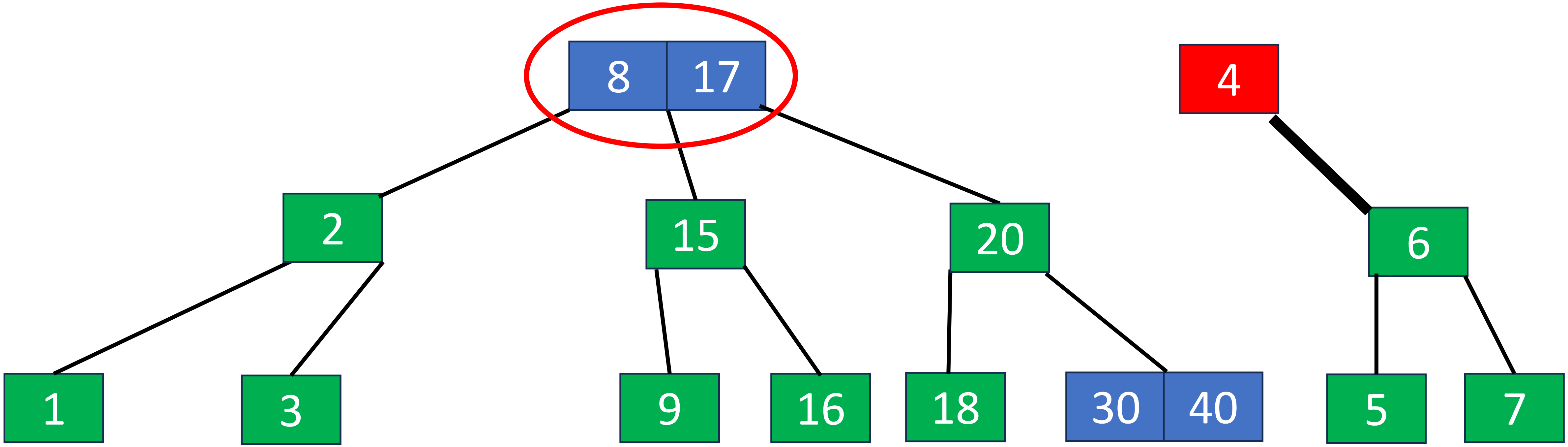
Example: Insert (2-3 tree)



- Then, insert the data pair with key=6 and the pointer to the new node into parent.

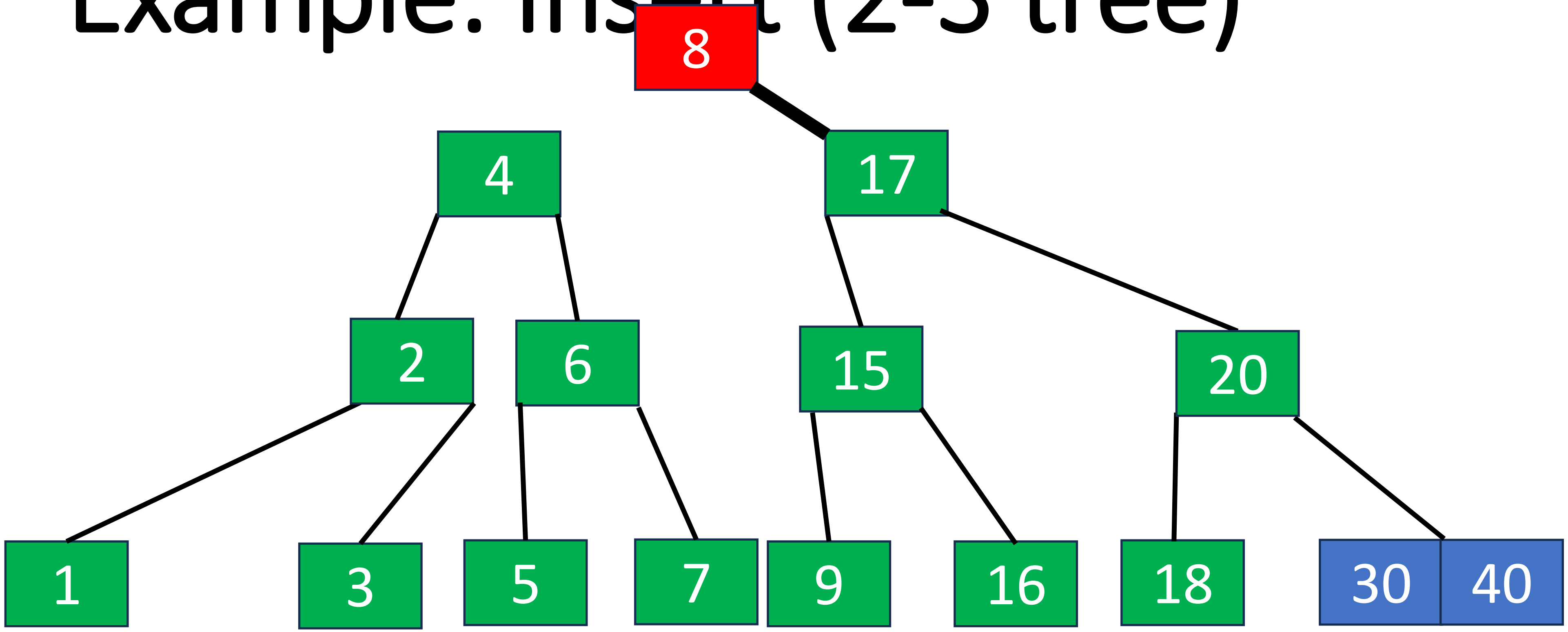
\\ We simplify the way to draw the representation. Use color to represent 2-node and 3-node

Example: Insert (2-3 tree)



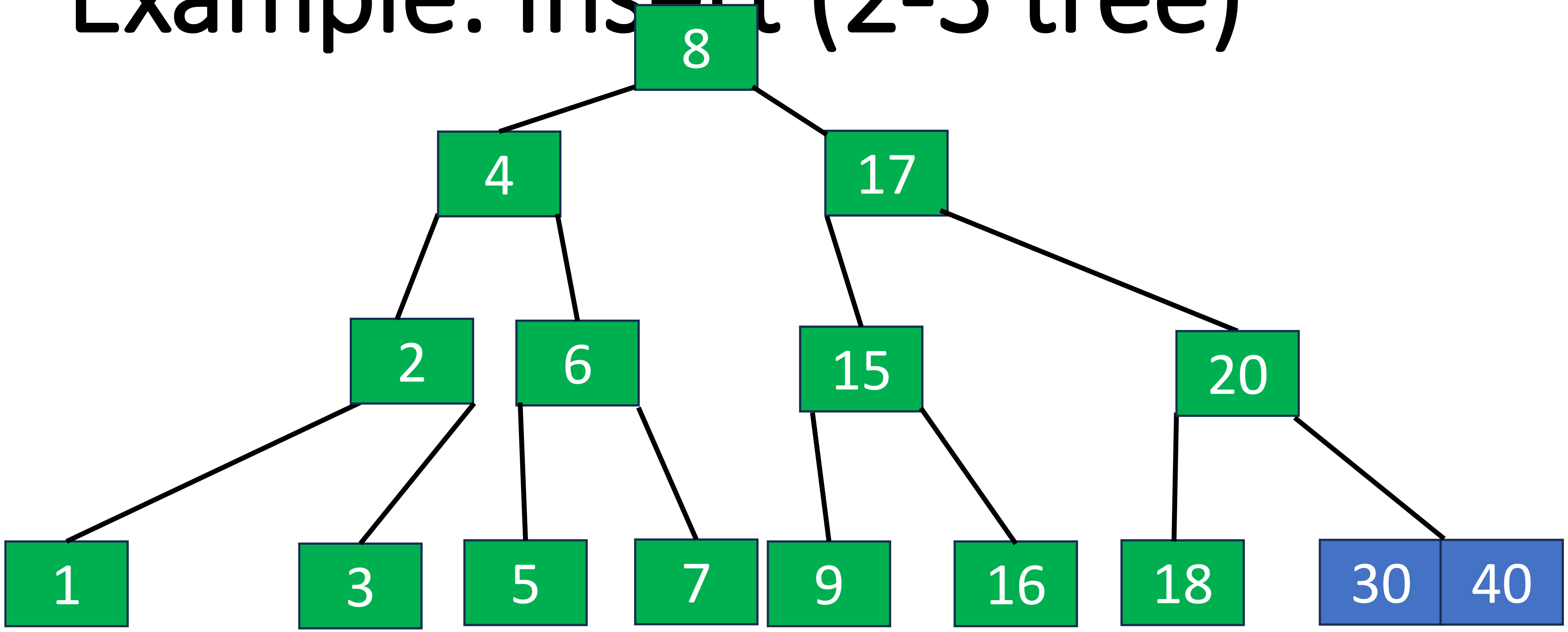
- Then, insert the data pair with key=4 and the pointer to the new node into parent.

Example: Insert (2-3 tree)



- Then, insert the data pair with key=8 and the pointer to the new node into parent.
- There is not parent. So, create a new root.

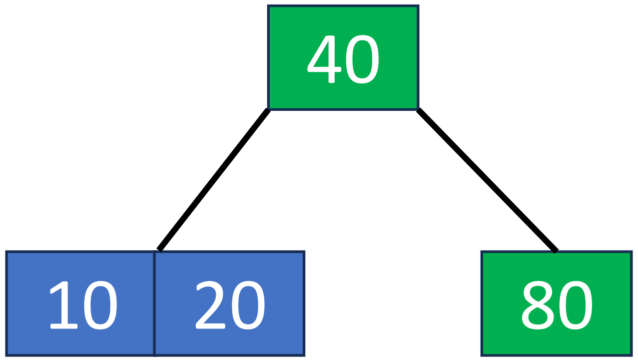
Example: Insert (2-3 tree)



- The height increases by 1.

Exercise

- Given the following 2-3 tree.
  - Q3: Please insert 70. What will be the keys of data pairs of node at index 3?
  - Q4: (Continue Q3) Then further insert 30. How many nodes are in level 2?
  - Q5: (Continue Q4) Finally, insert 60. How many nodes are in the tree?

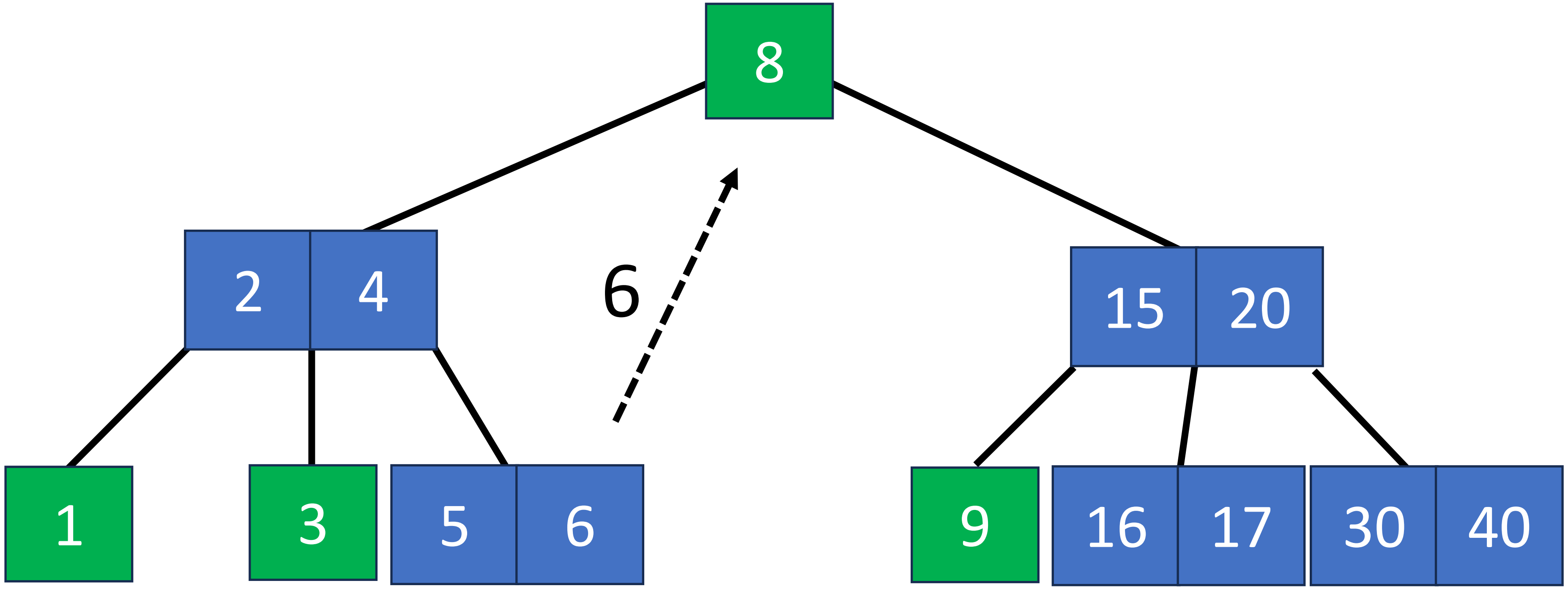


Please reply your answers of Q3-Q5 via the following link:

https://forms.gle/AkBqtrRHYHu9VvL89

Group members: 2~4 people

Operation: Delete (2-3 tree)



- Delete the pair with key = 8.
- Transform deletion from <u>interior</u> into deletion from a <u>leaf</u>.
  // similar to "delete an internal node in binary tree"
- Replace by largest in left subtree.

Operation: Deletion from a leaf node

Deletion a data pair *x* from a leaf node *p*

q: the nearest neighbor of *p* (if any)

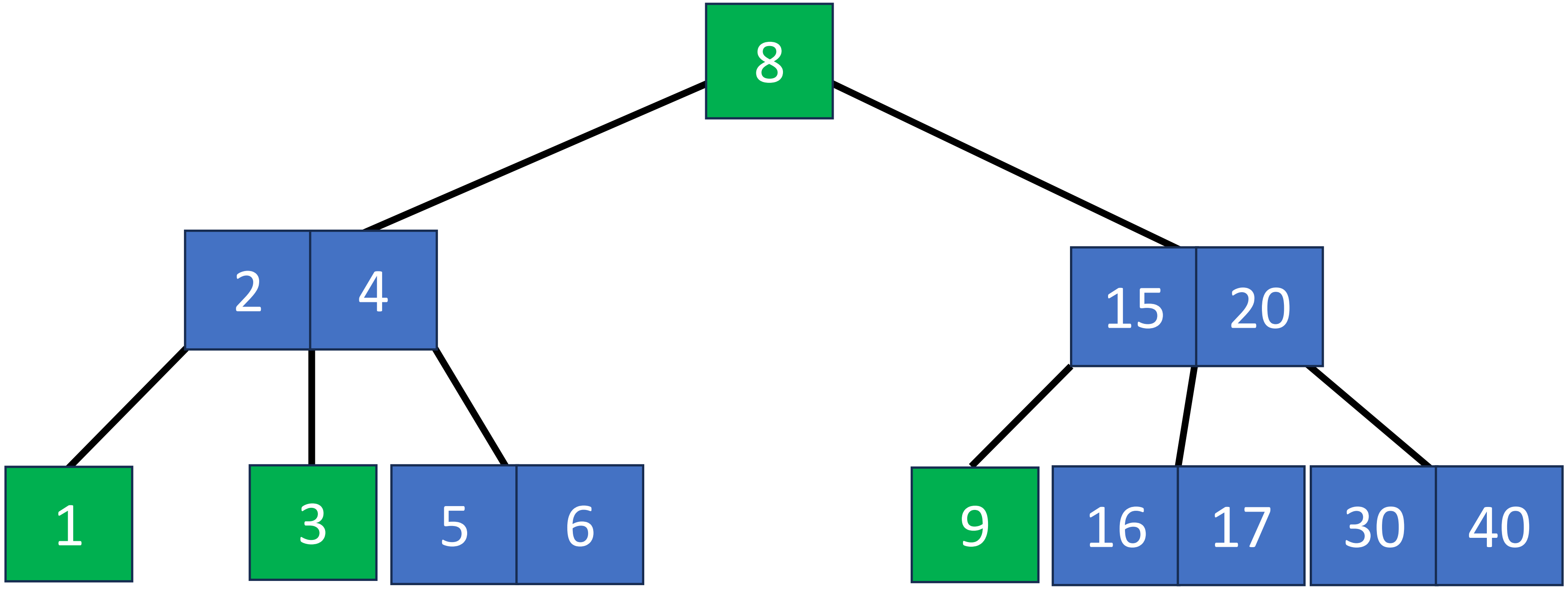
- *p* is the root. → remove *x*
- *p* is not in the root.
  - Case 1: *p* has at least $\lceil m/2 \rceil$ data pairs
    → remove *x*
  - Case 2: *p* has $\lceil m/2 \rceil$-1 data pairs and *q* has at least $\lceil m/2 \rceil$ data pairs
    → remove *x* and do rotation
  - Case 3: *p* has $\lceil m/2 \rceil$-1 data pairs and *q* has $\lceil m/2 \rceil$-1 data pairs
    → remove *x* and do combine
    → check the parent

After deletion, *p* has at least $\lceil m/2 \rceil$-1 data pairs.

After deletion, *p* has $\lceil m/2 \rceil$-2 data pairs. → deficient

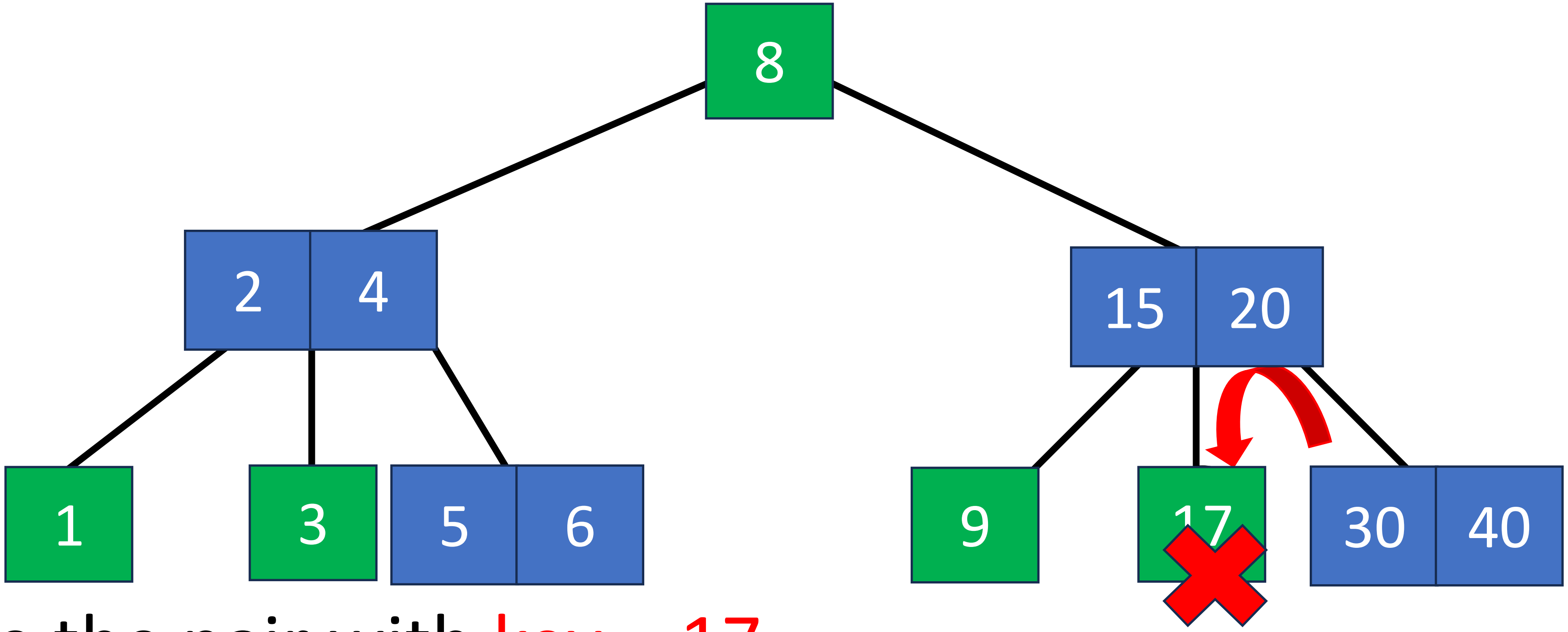
The minimum number of data pairs required by a nonroot node.

Example: Delete from a leaf (2-3 tree)



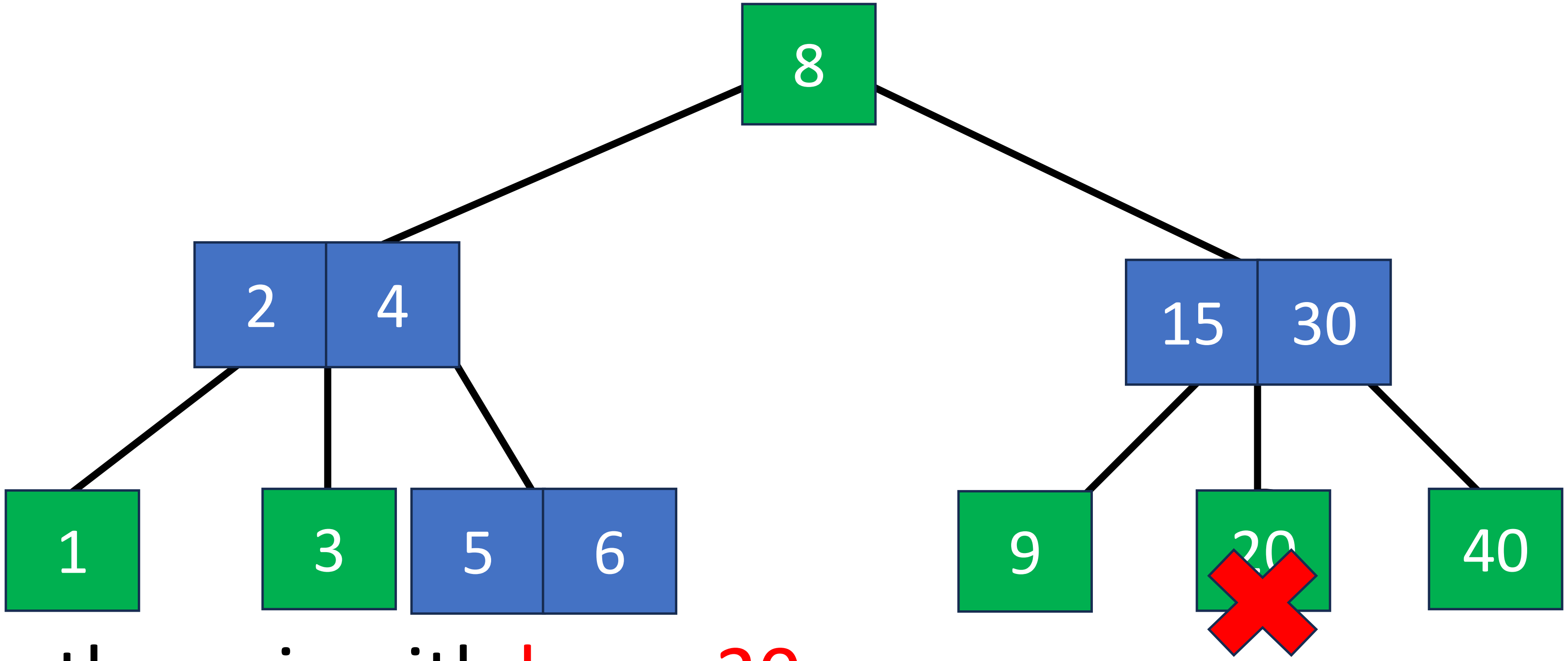
- Delete the pair with key = 16.
- A 3-node becomes 2-node.

Example: Delete from a leaf (2-3 tree)



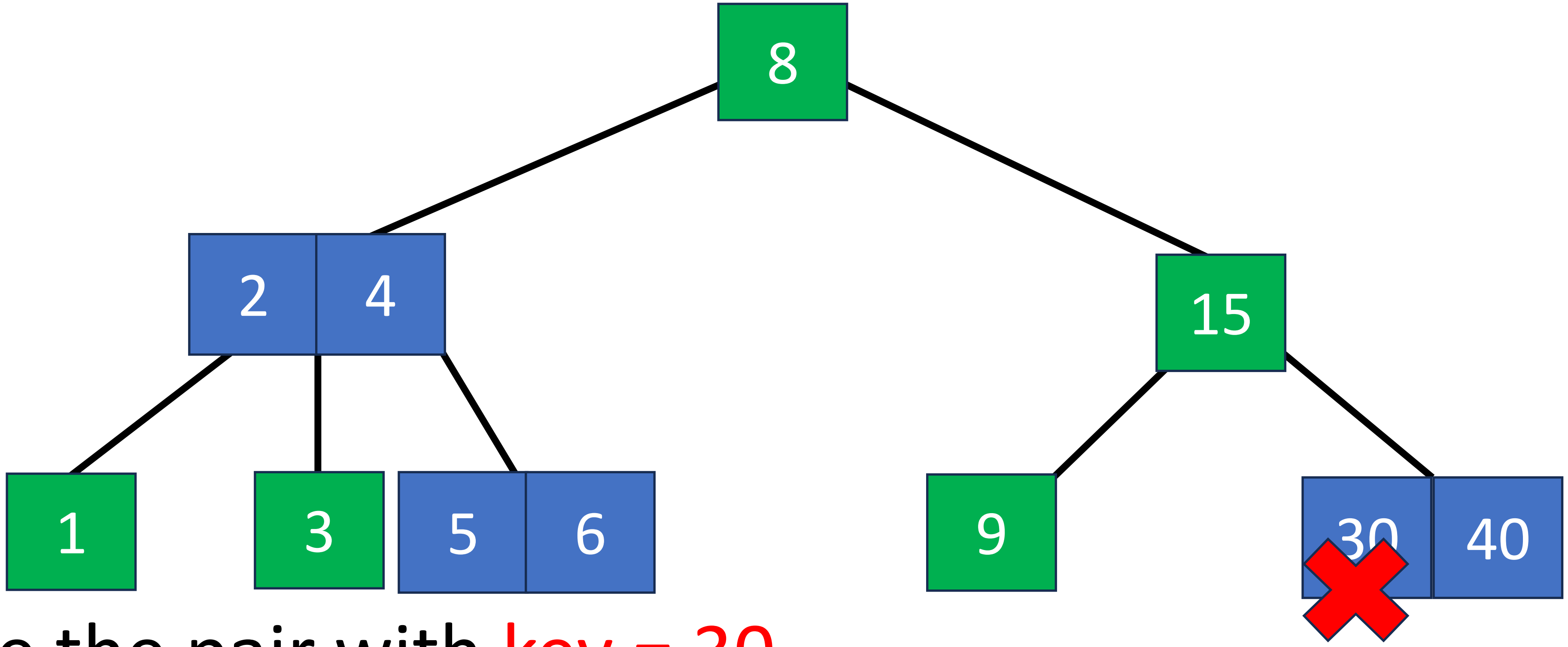
- Delete the pair with key = 17.
- A 2-node is deleted.
- If it has a 3-node sibling, we borrow a data pair and a subtree via parent node using rotation.

Delete from a leaf (after rotation)



- Delete the pair with key = 20.
- A 2-node is deleted.
- It has no 3-node sibling. We combine the node with its sibling and parent pair.

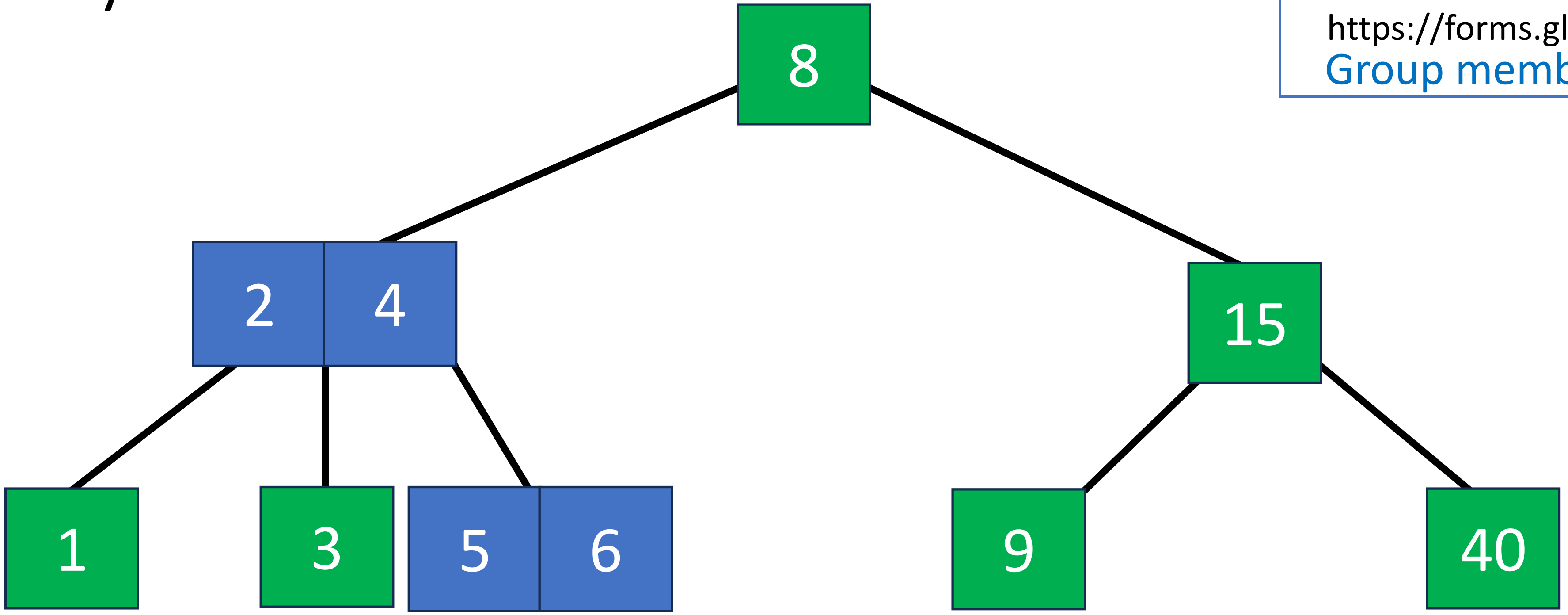
Delete from a leaf (after Combine)



- Delete the pair with key = 30.
- Deletion from a 3-node.
- 3-node becomes 2-node.

Exercise

- Given the following 2-3 tree.
  - Q6: Please delete the pair with key = 3. How many children do the left child of the root have?
  - Q7: (Continue Q6) Please delete the pair with key = 6. How many children do the left child of the root have?



Please reply your answers of Q6-Q7 via the following link:

https://forms.gle/AkBqtrRHYHu9VvL89

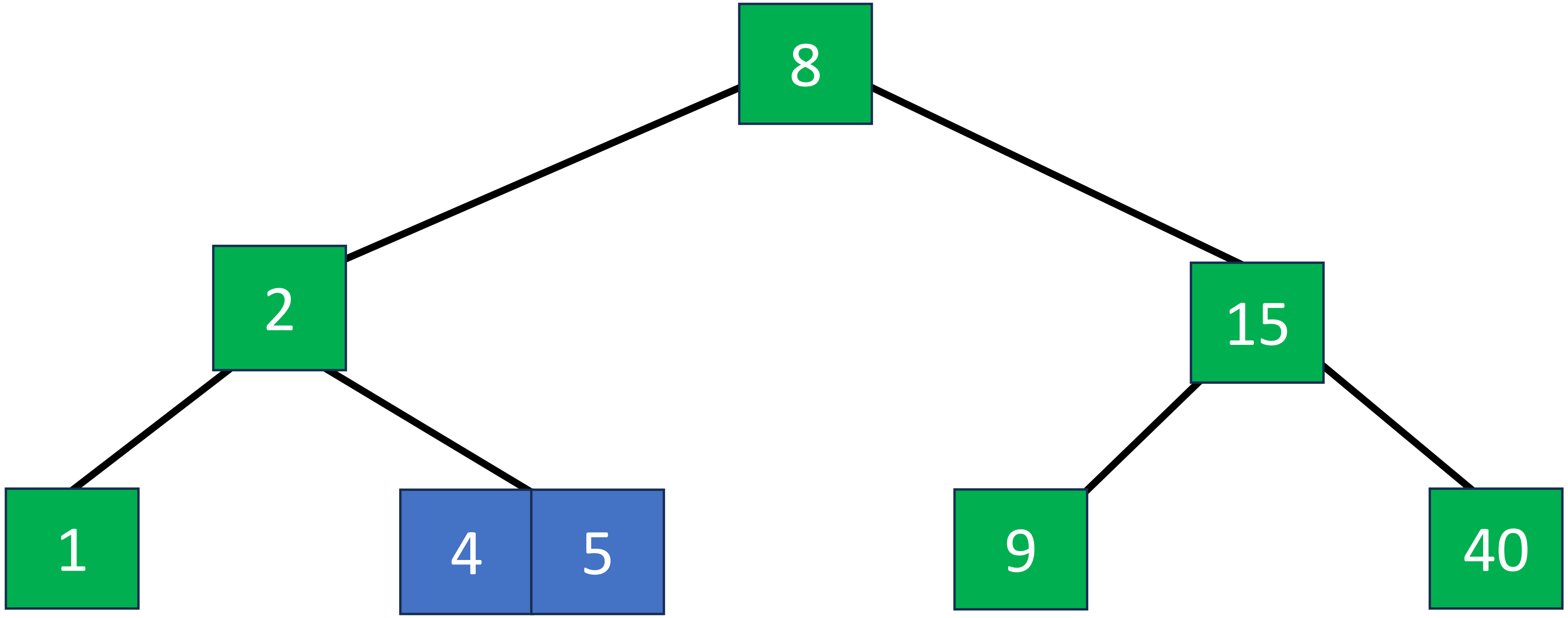
Group members: 2~4 people

Notes

- Rotation has higher priority.
  - The number of data pairs is enough.
  - If we can use the original space, let's use them.

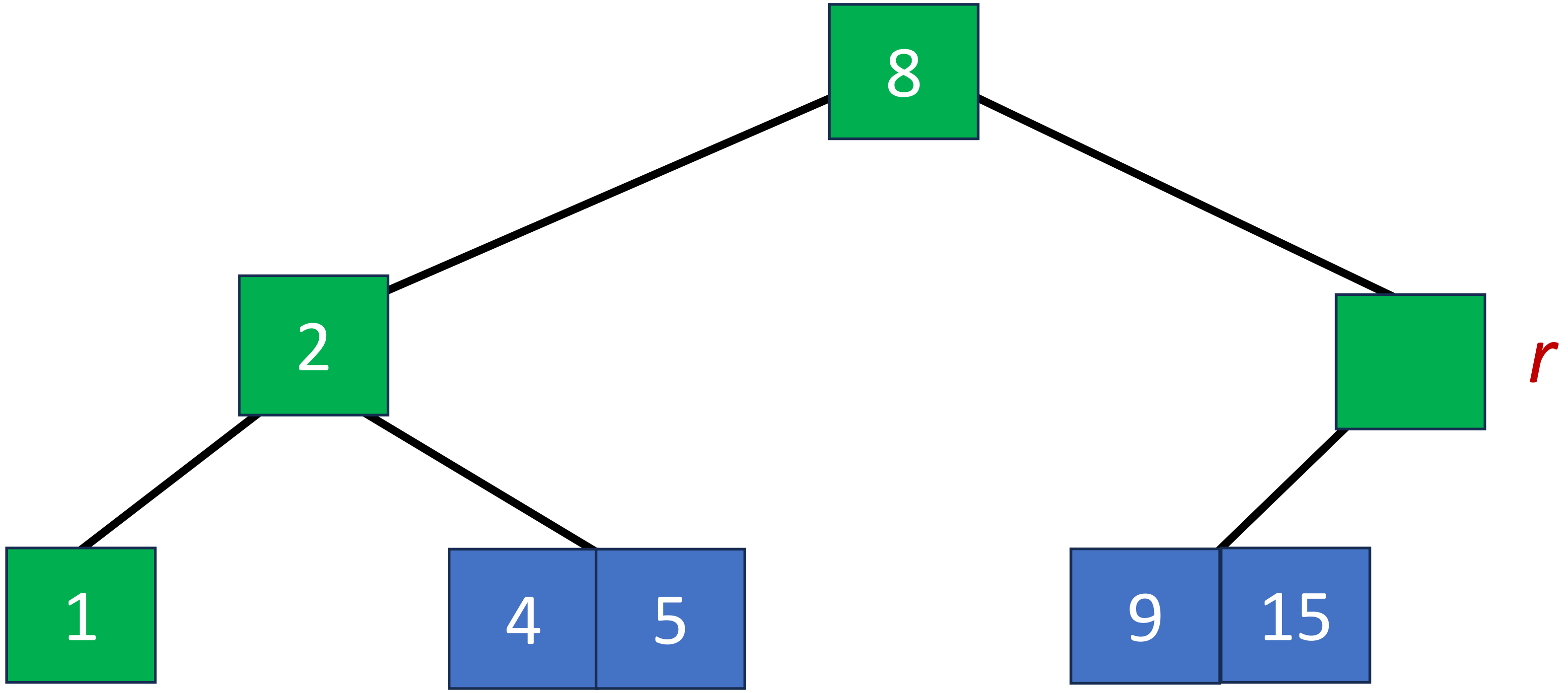
- When rotation is impossible, we do Combine.
  - The number of data pairs is <u>not</u> enough.
  - Return the memory space for a node. (The size may be large.)
  - It may reduce the height of the tree.

Delete from a leaf



- Delete the pair with key = 40.
- Deletion from a 2-node.
- It has no 3-node sibling, we combine its sibling and parent pair.

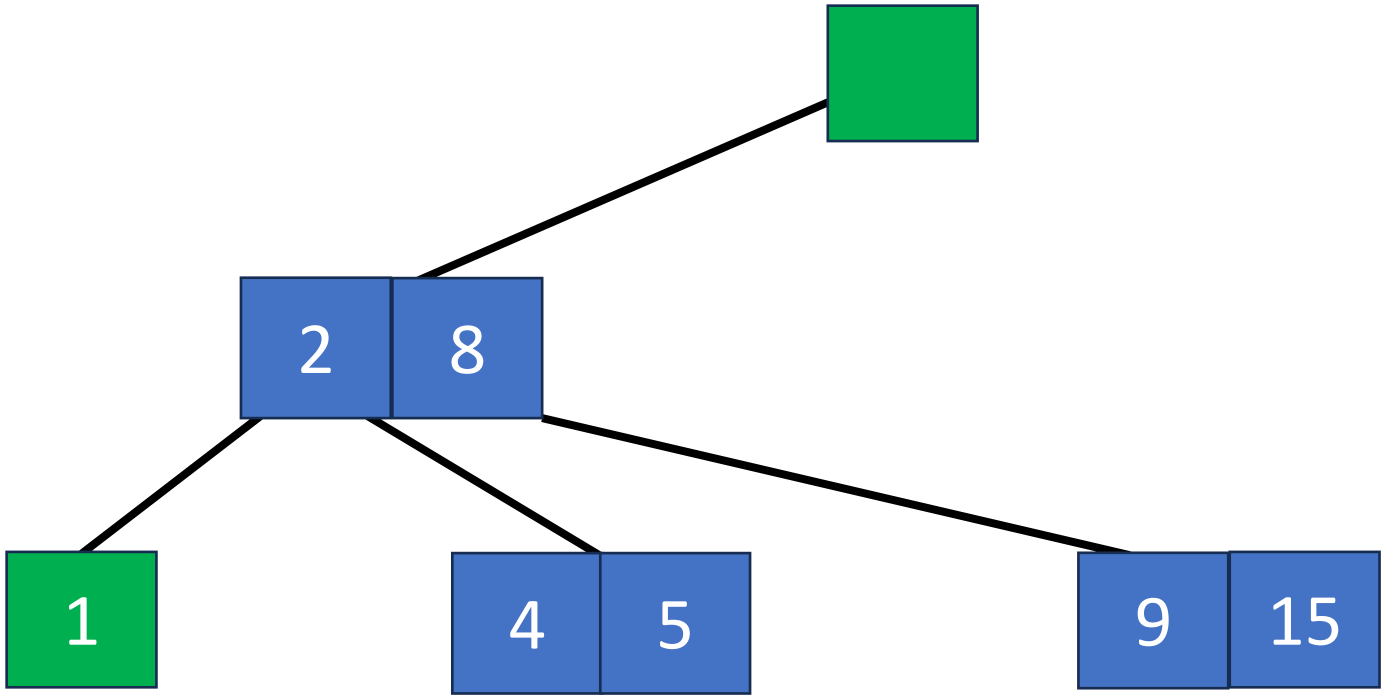
Delete from a leaf (after Combine)



- The combining operation reduces the number of data pairs in the parent node *r* by one.
- The parent pair was from a 2-node.
- ~~If the parent node has a 3-node nearest sibling, do rotation.~~

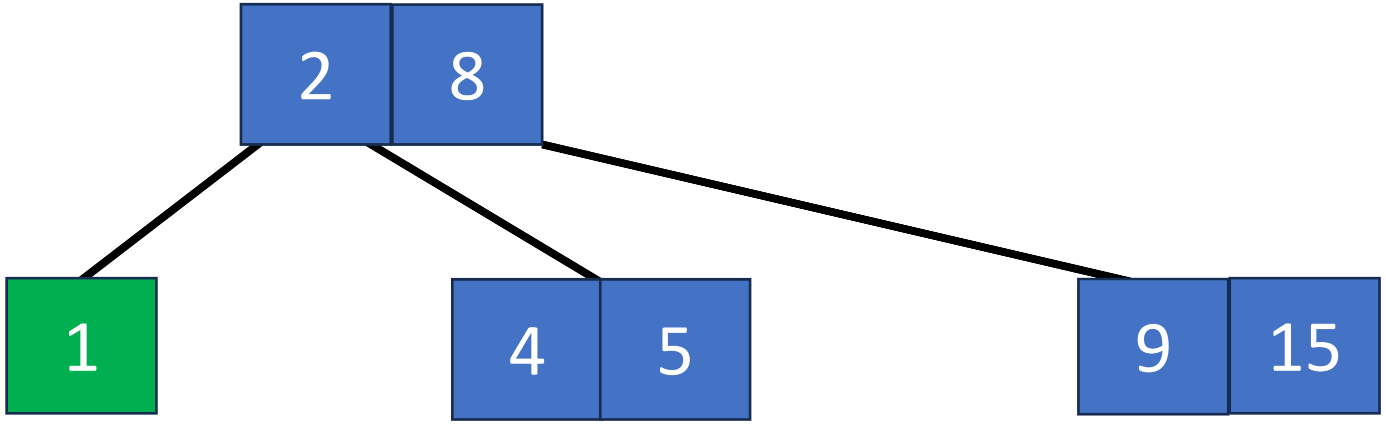
If the parent node has no 3-node nearest sibling, do combine.

Delete from a leaf (after Combine)



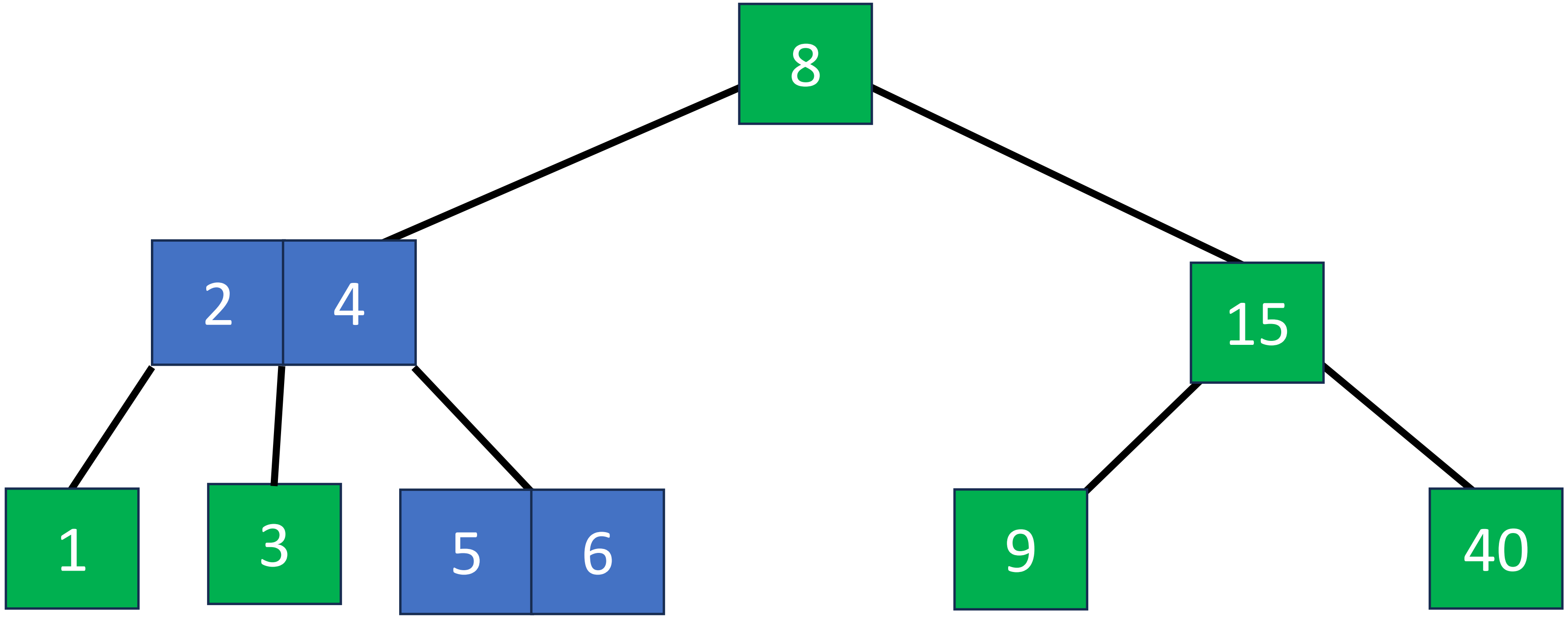
- Parent pair (key=8) was from a 2-node.
- Check the nearest sibling and determine if it is a 3-node.
- No sibling, the node must be the root.
- Discard root. Left child becomes new root.

Delete from a leaf



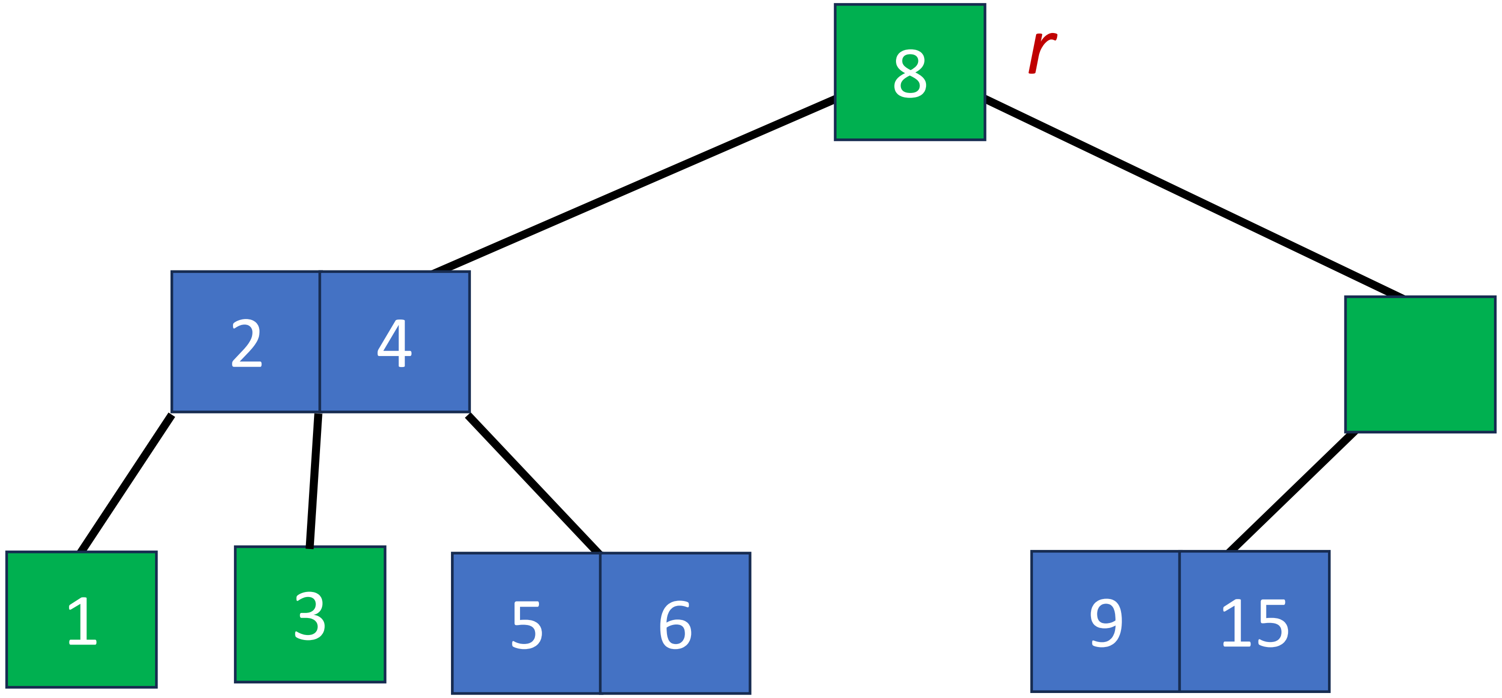
- The height reduces by 1.

Delete from a leaf



- Delete the pair with key = 40.
- Deletion from a 2-node.
- It has no 3-node nearest sibling, we combine its nearest sibling and parent pair.

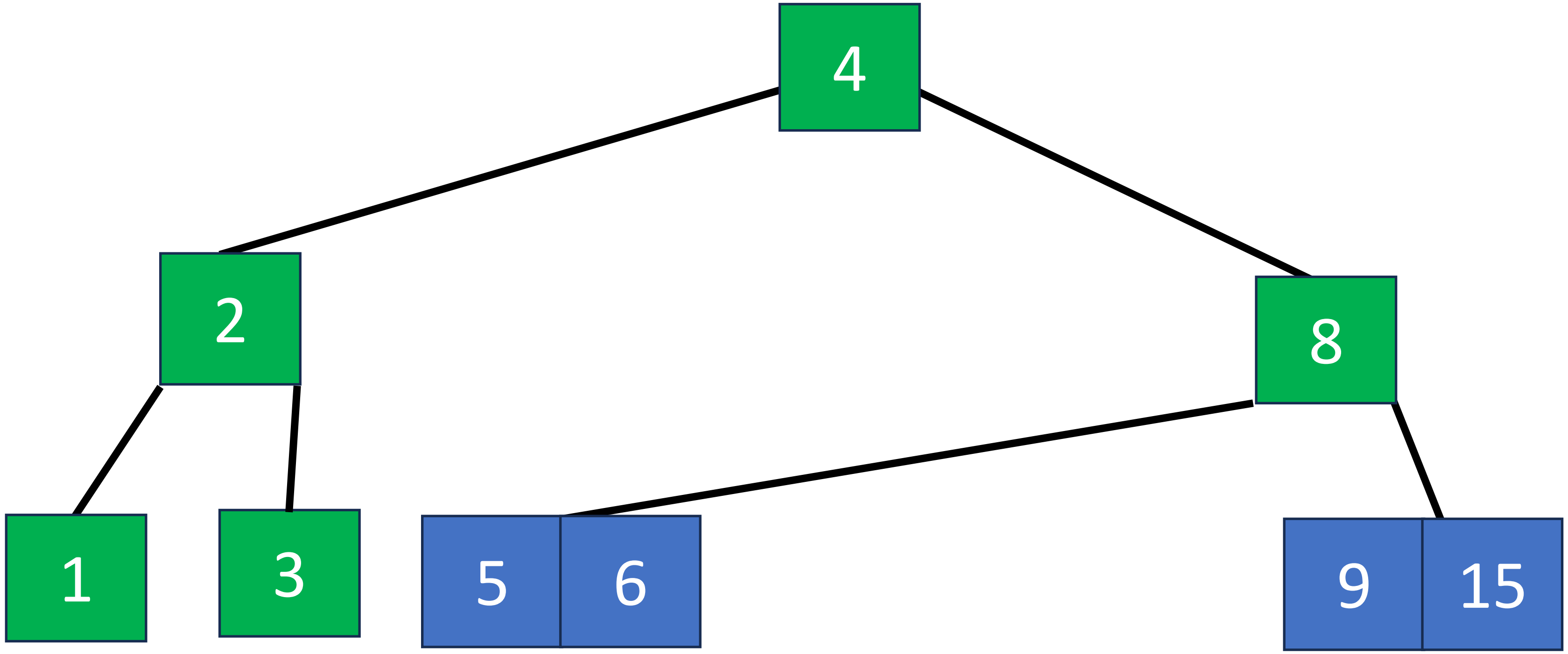
Delete from a leaf (After combine)



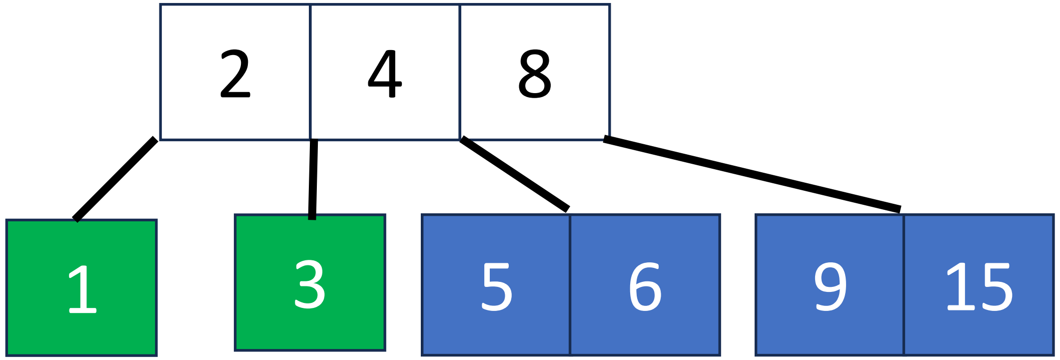
- The parent pair (key=15) was from a 2-node.
- If the parent node *r* has a 3-node nearest sibling, do rotation.

~~If the parent node *r* has no 3-node nearest sibling, do combine.~~

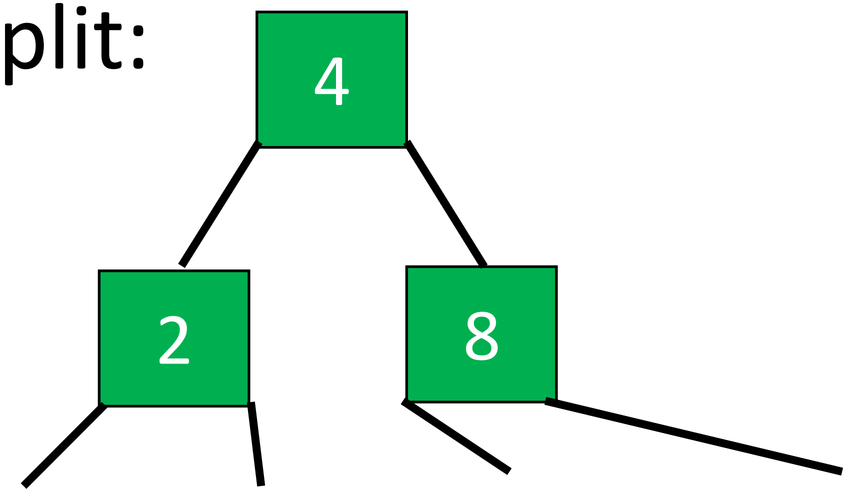
Delete from a leaf (After rotation)



- Combine:



- Then, split:



Summary

- Multiway search tree
- B-tree
- 2-3 tree
  - Insertion
  - Deletion