# $B^+$-trees

Ch. 11.3

- **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)

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

- **Ch 11.3 $B^+$-tree**

**We are here**

# $B^+$-tree

- Similar with B-trees
- Two types of nodes: data node & index node
- Data pairs are in leaves (data node) only.
  - Leaves form a doubly-linked list.

* The capacity of a data node need **not** be the same as that of an index node.

- Index node has up to **m-1** keys and **m** pointers.

$$n, A_0, K_1, A_1, K_2, A_2, \ldots, K_n, A_n$$

- $n$: number of keys ($n < m$)
- $A_i$ : pointer to a subtree
- $K_i$ : a key

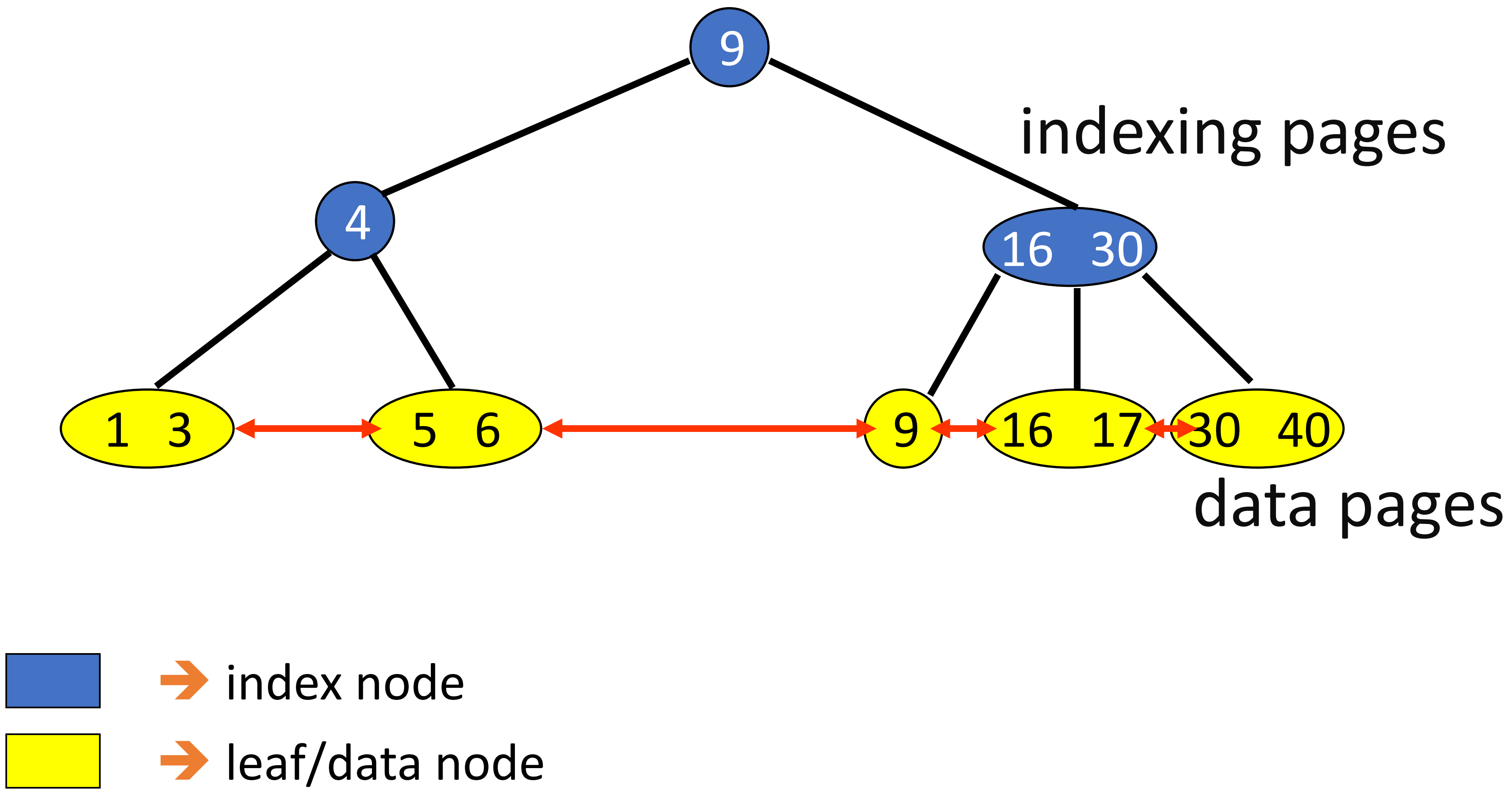
- Key of left data pair < key of right data pair

$$K_i < K_{i+1}$$

- $K_i \leq$ keys in the subtree $A_i$ < $K_{i+1}$

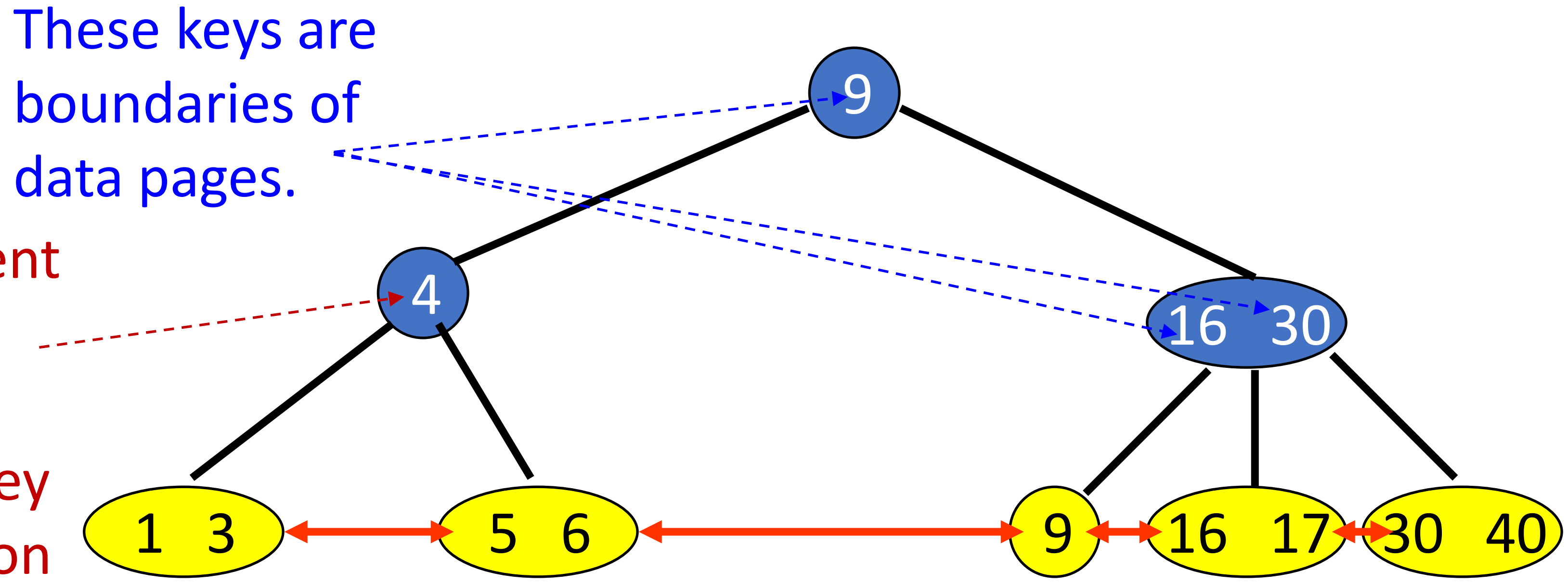
- Note that the smallest key of $A_i$ may be equal to $K_i$

# Example of B$^{+}$-tree



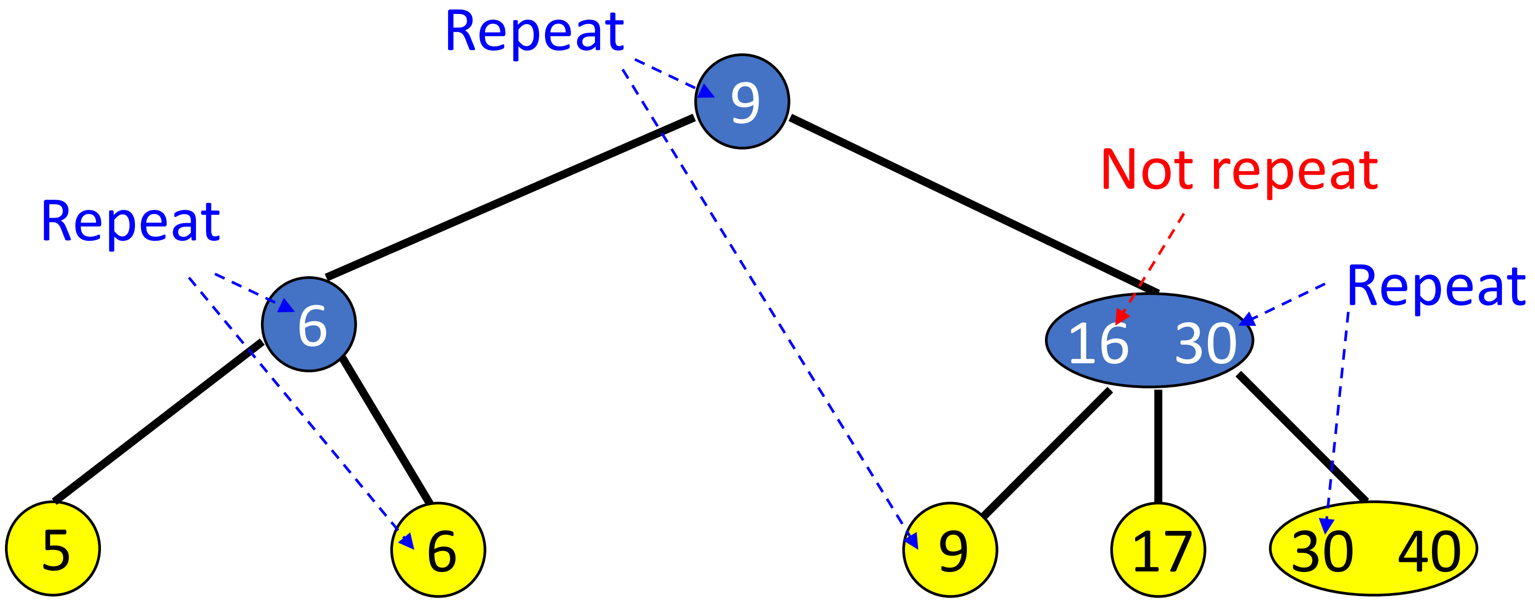
# Observations

- Some of the keys in data nodes are used in index nodes.
- Sometimes, keys in index nodes are boundary of data pages.
- When searching, we have to visit data pages to obtain values.



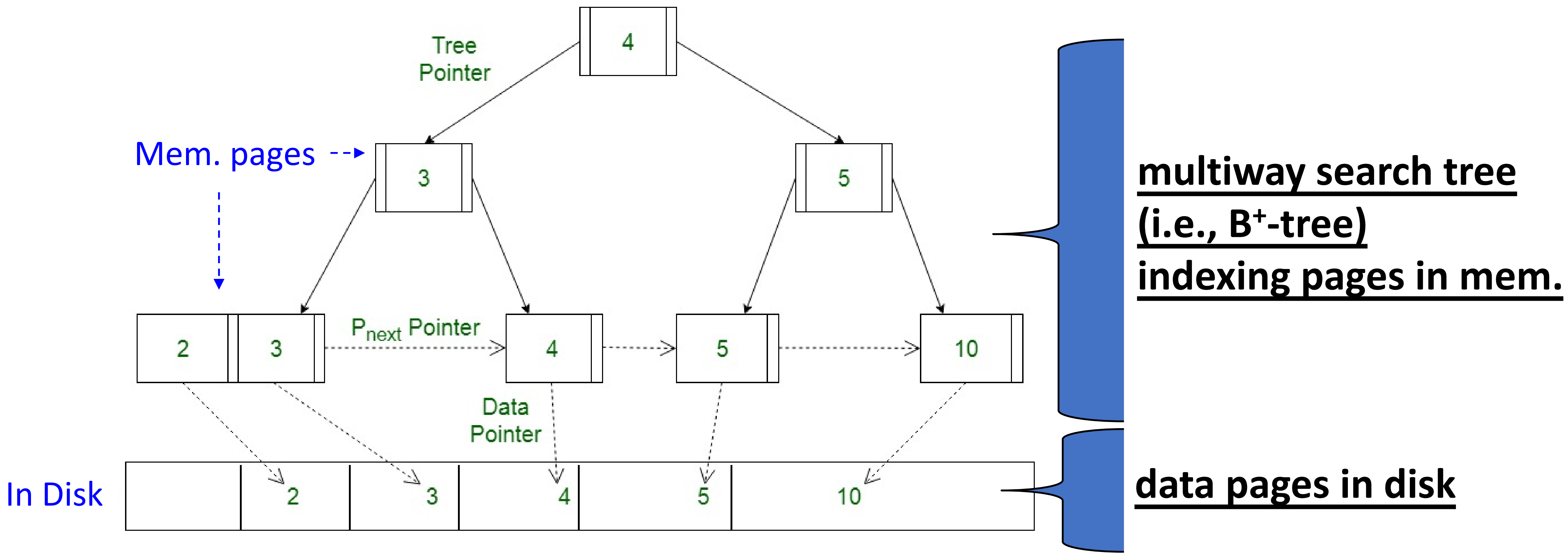

- Indices can be different from the keys in data pages.
- Indices are valid if they can guide the direction to correct data pages.

# Another example of $B^+$-tree



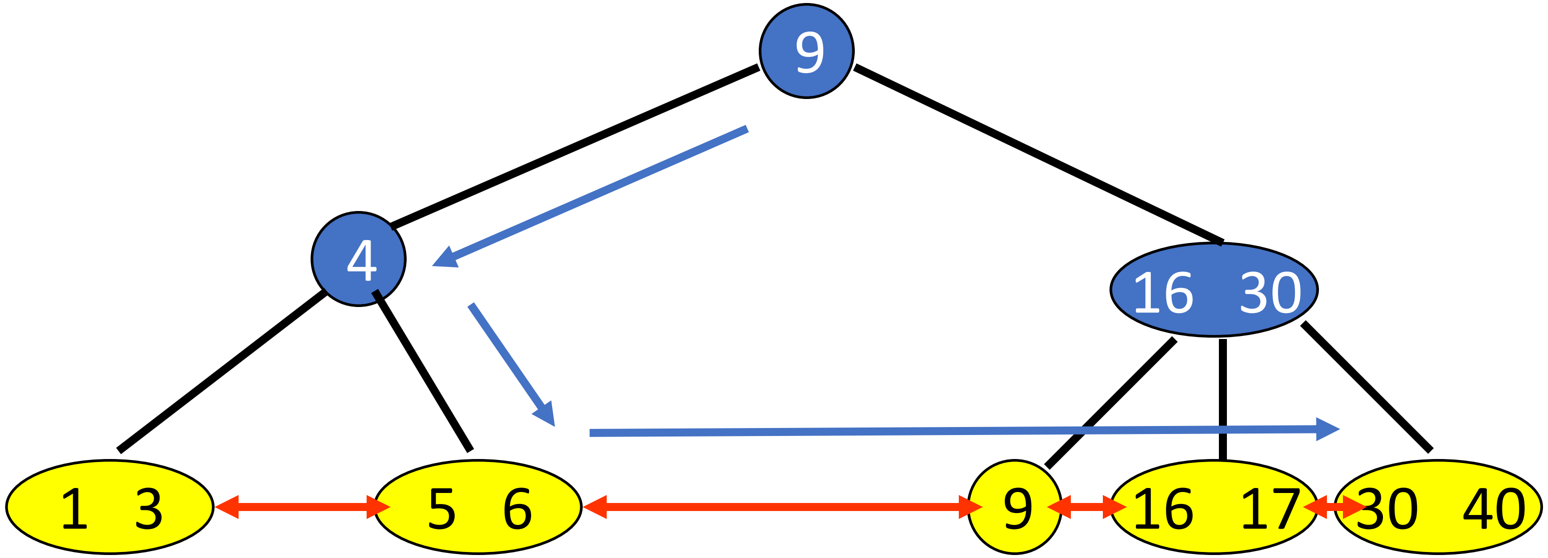
# Why $B^+$-tree (vs. B-tree)?

- Memory pages hold pointers only so as to maximize the numbers of pointers stored in memory, resulting in reduction of the number of disk pages accessed.



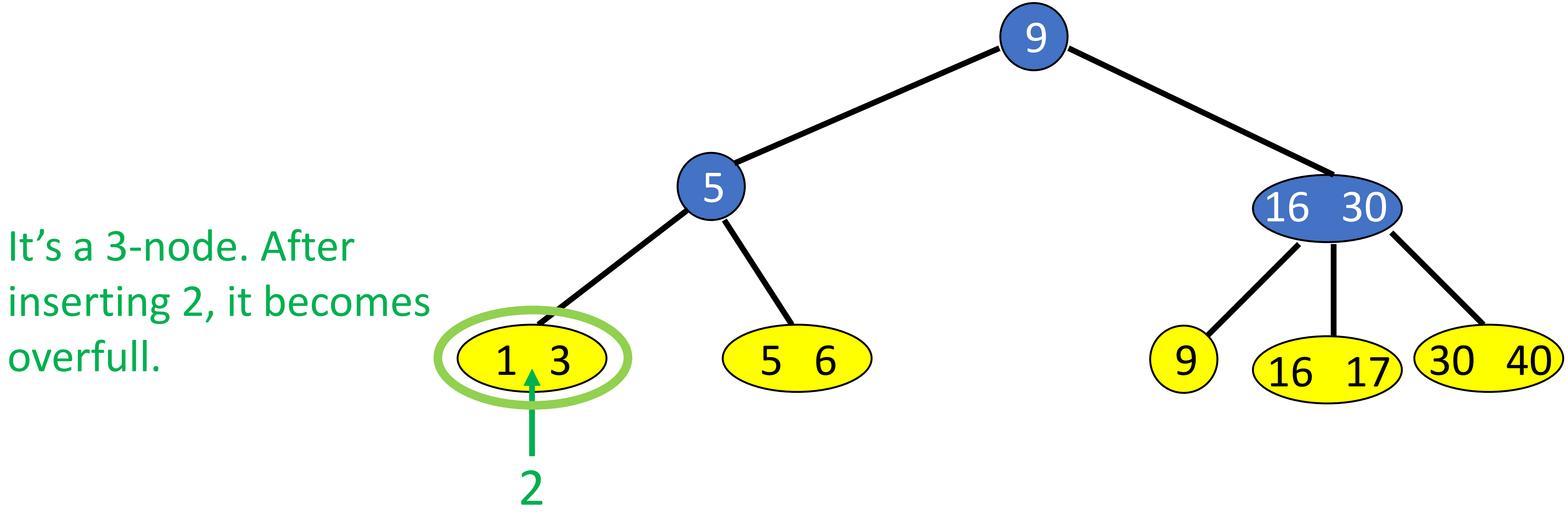
# Operation: Search

- Exact match (pin search): e.g., find key = 5
- Range search: e.g., find 6 ≤ key ≤ 20
  - Find the data node containing key = 6, then move rightward to find key exceeding 20.



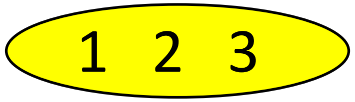
# Operation: Insert

- Similar with insertion algorithm of B-tree
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.



# Insert into a 3-node (for data node)

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




- Split into two data nodes.

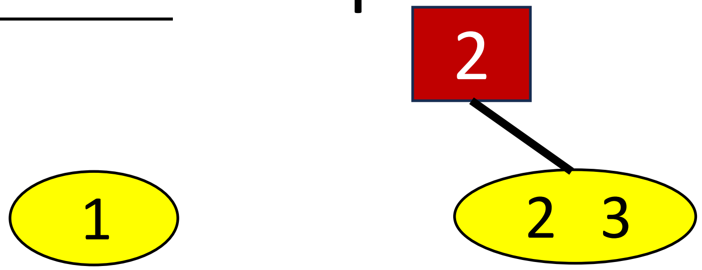

Also inserted into parent index node

$$m, \underbrace{E_1, \ldots, E_{[m/2]-1}}_{p \text{ node}}, \underbrace{E_{[m/2]}, E_{[m/2]+1}, \ldots, E_m}_{\text{New } q \text{ node}}$$

$[m/2]$-1 elements

m-$[m/2]$+1 data elements

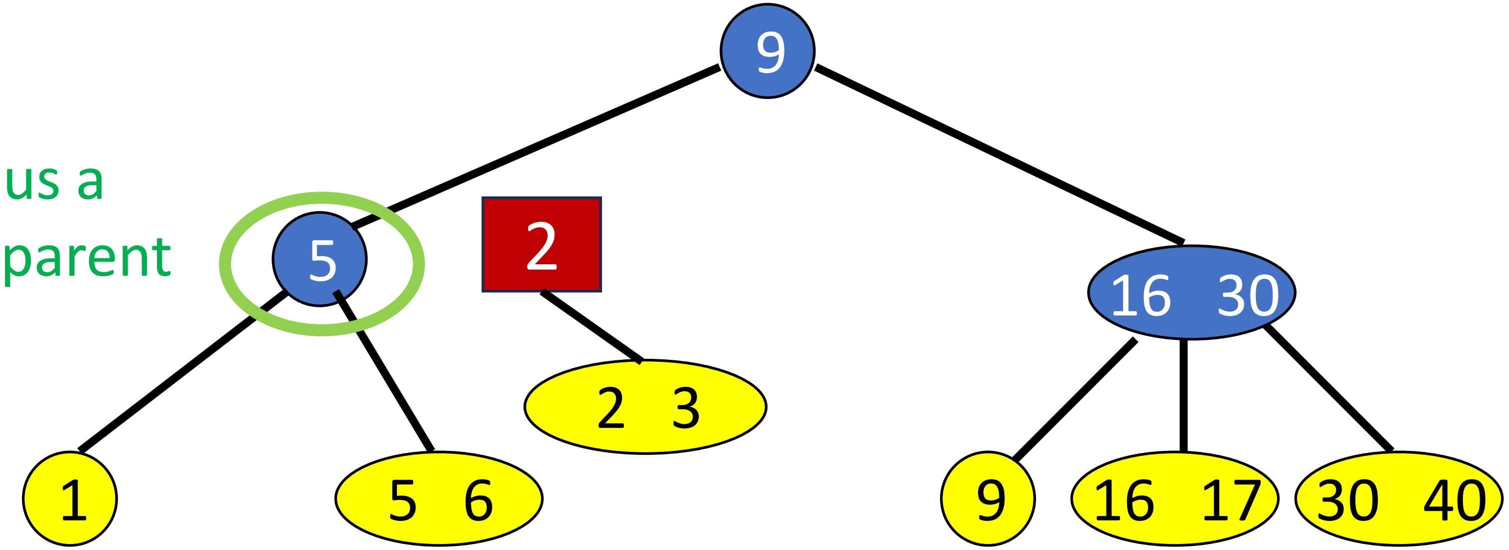
- Insert smallest key of new data node and a pointer to this new node into parent index node.



# Operation: Insert

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.

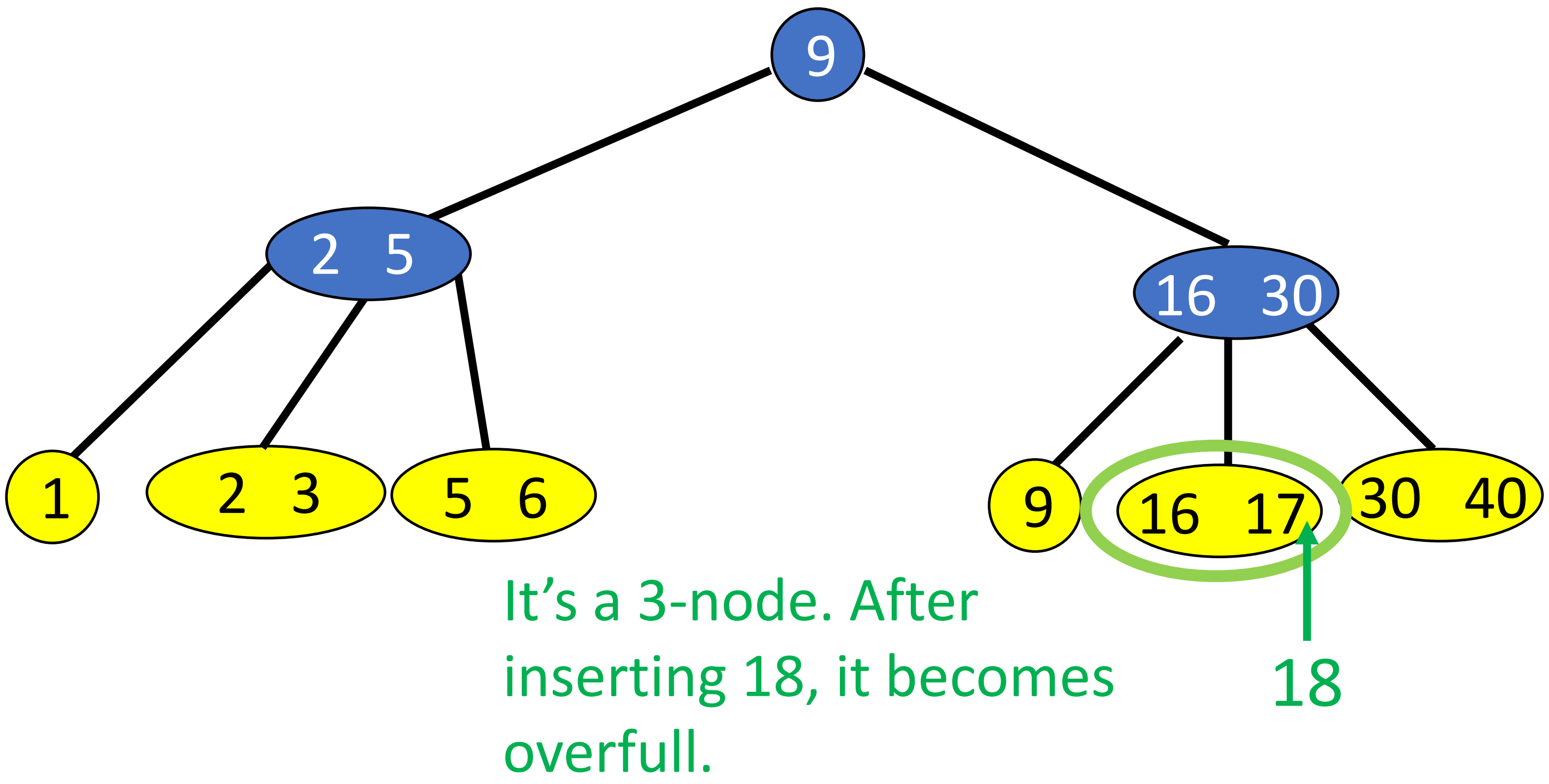
Insert a key=2 plus a pointer into the parent node.




Index node of $B^+$-tree: Insertion for index node is the same as insertion in B-tree.
The key moved to the parent node does not remain in the new index node.

# Operation: Insert

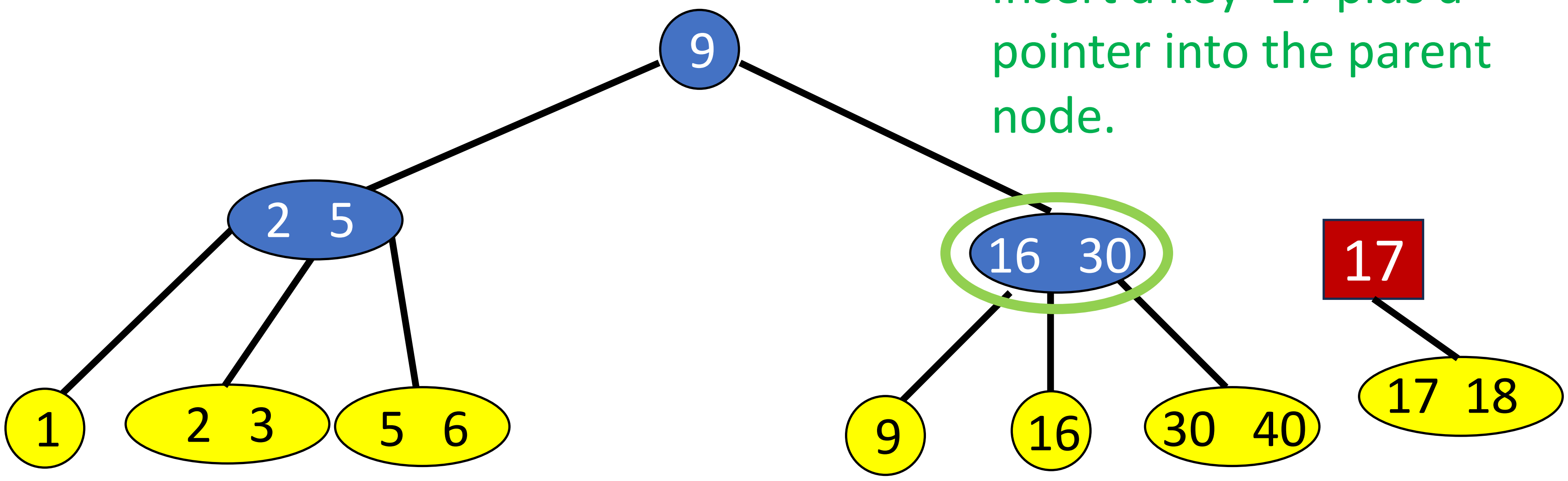
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.
  - Next, let insert a pair with key = 18. (1/4)



# Operation: Insert

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.
  - Next, let insert a pair with key = 18. (2/4)

Insert a key=17 plus a pointer into the parent node.

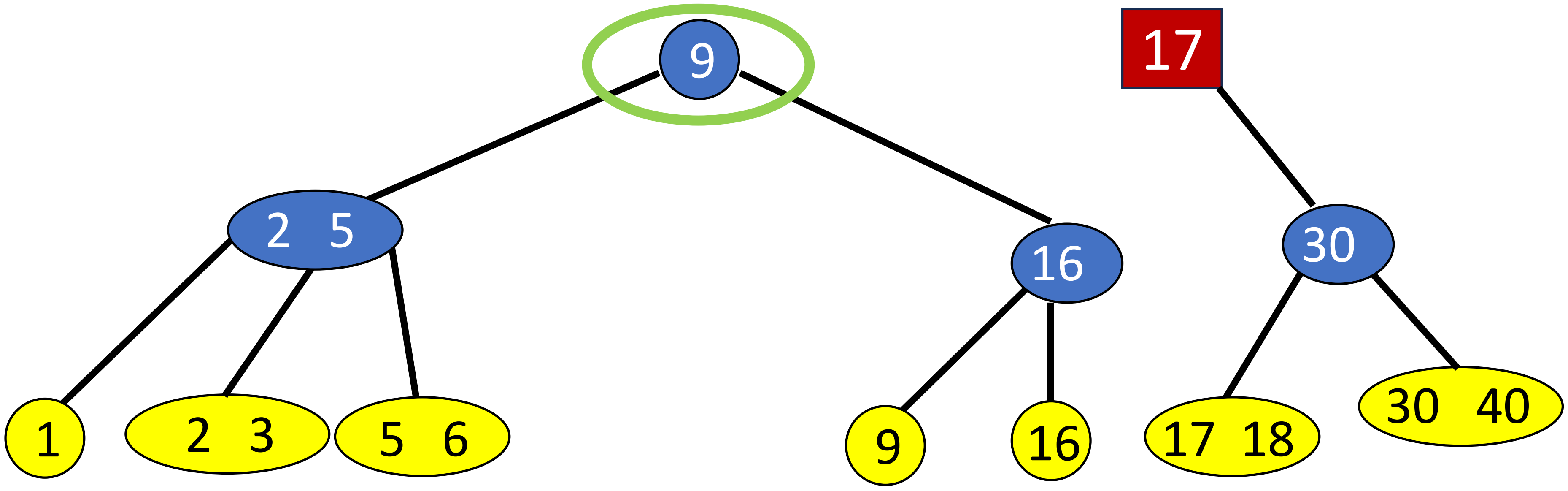



Index node of $B^+$-tree: Insertion for index node is the same as insertion in B-tree. The key moved to the parent node does not remain in the new index node.

# Operation: Insert

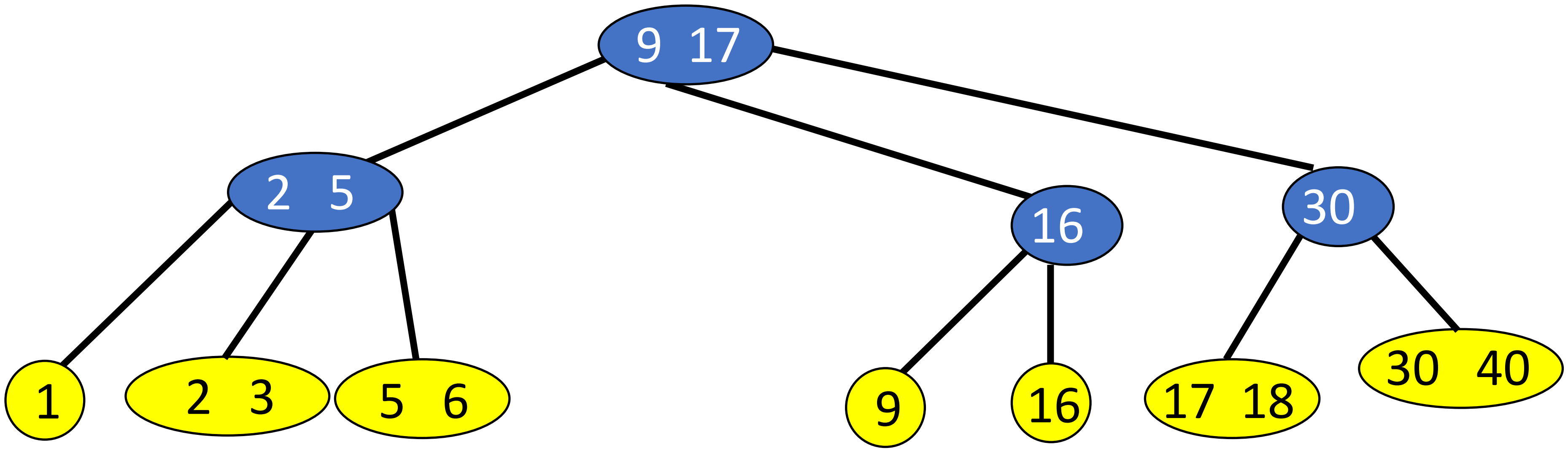
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.
  - Next, let insert a pair with key = 18. (3/4)

Insert a key=17 plus a pointer into the parent node.



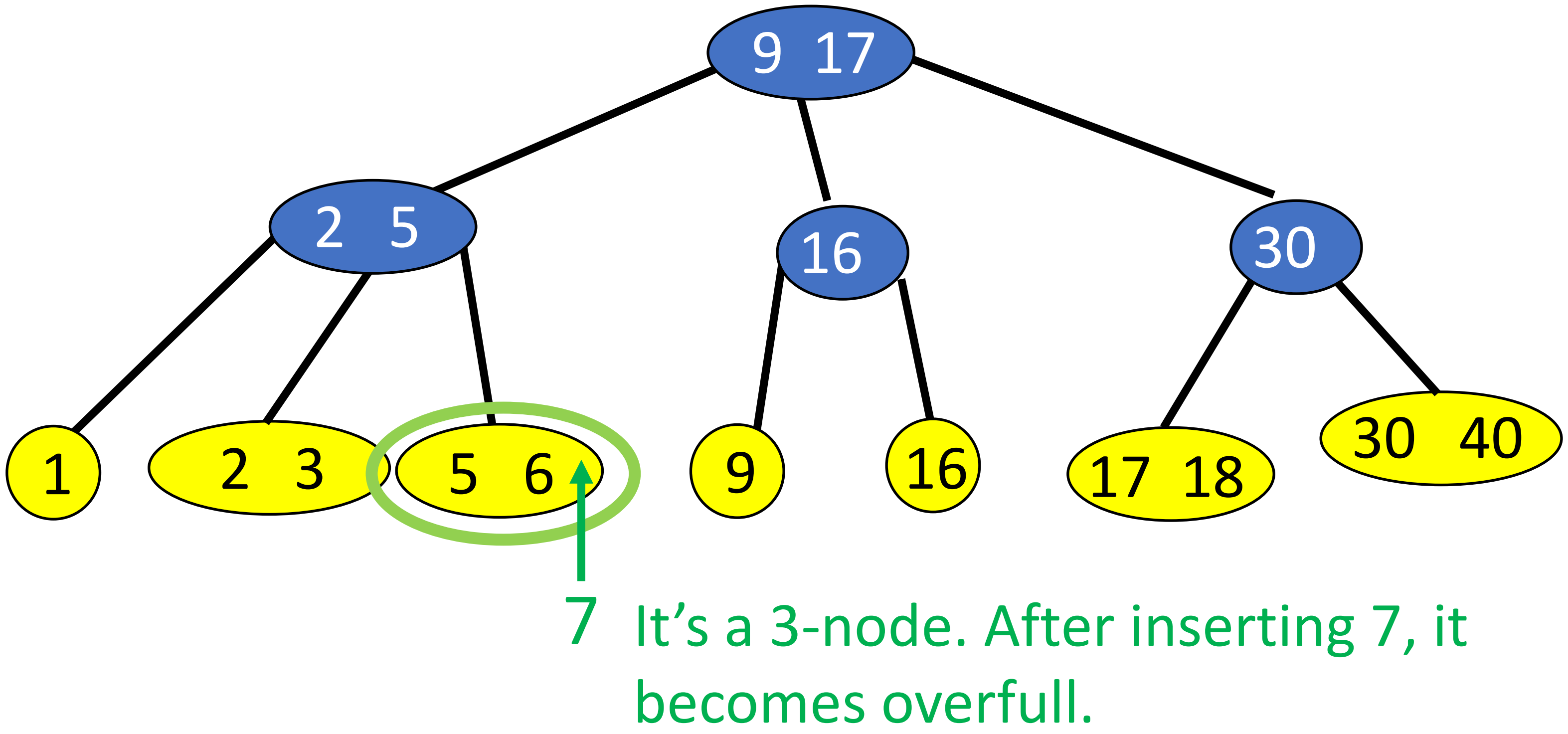
# Operation: Insert

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Insert a pair with key = 2.
  - Next, let insert a pair with key = 18. (4/4)



# Operation: Insert

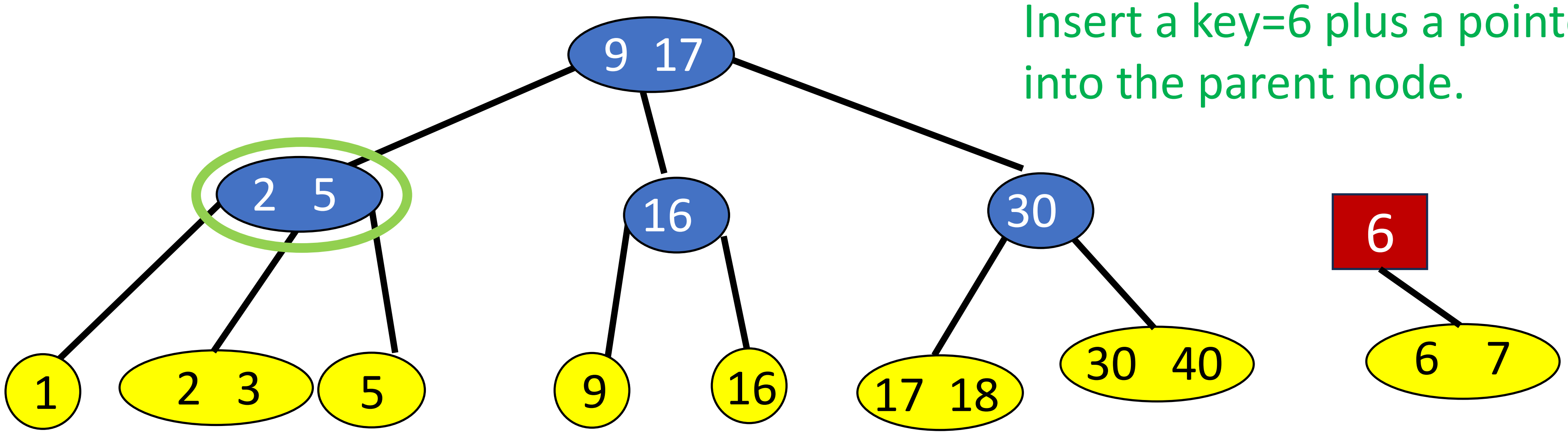
- Example:
  - Next, let insert a pair with key = 7. (1/4)



# Operation: Insert

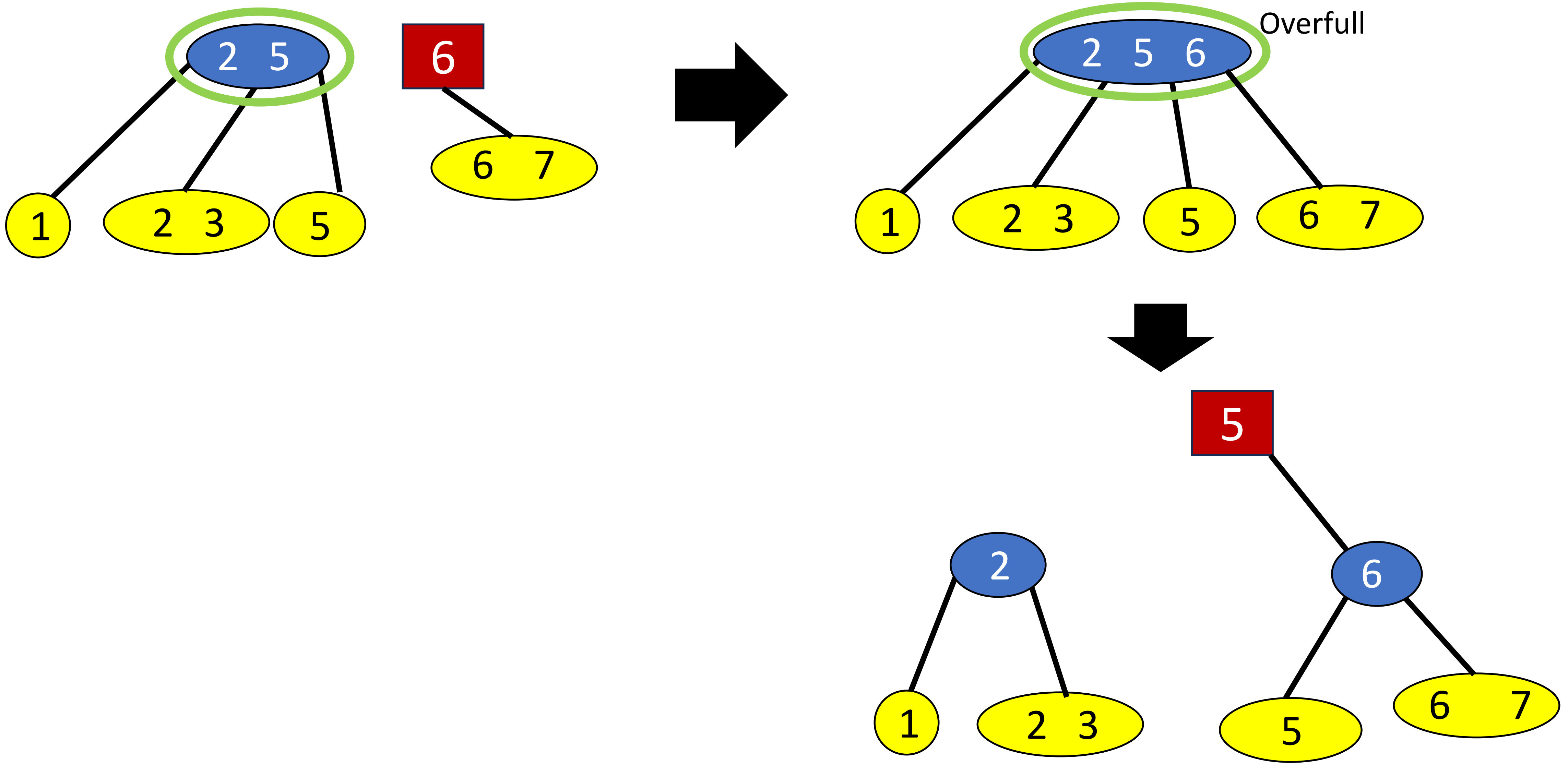
- Example:
  - Next, let insert a pair with key = 7. (2/4)

Insert a key=6 plus a pointer into the parent node.



# Operation: Insert

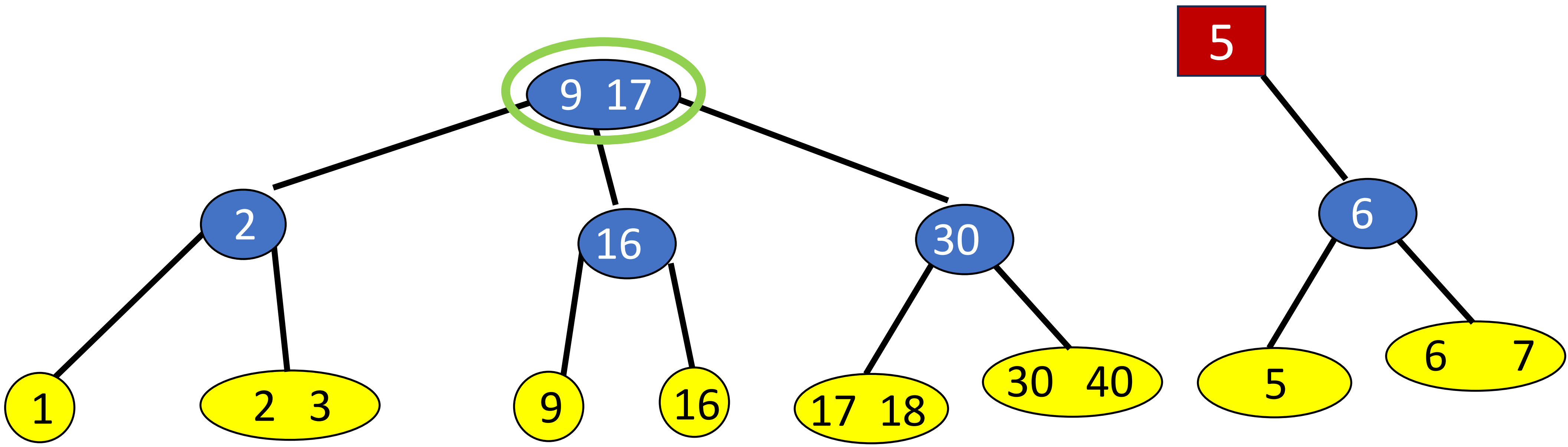
Insert a key=6 plus a pointer into the parent index node.



# Operation: Insert

- Example:
  - Next, let insert a pair with key = 7. (3/4)

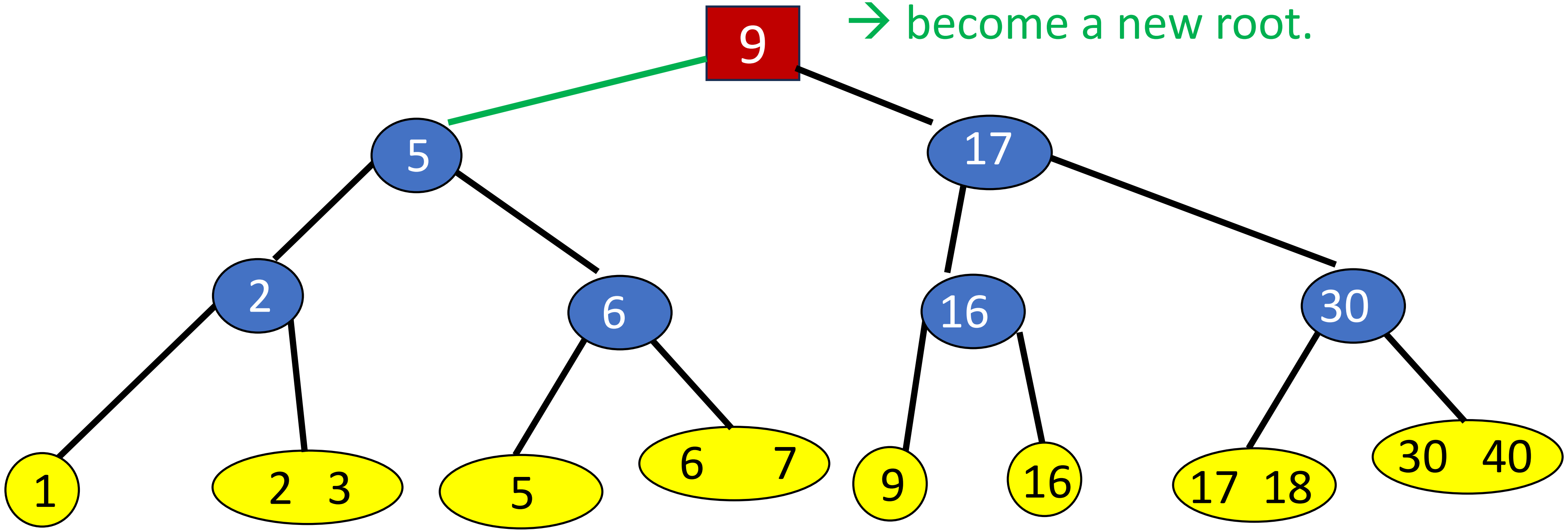
Insert a key=5 plus a pointer into the parent node.



# Operation: Insert

- Example:
  - Next, let insert a pair with key = 7. (4/4)

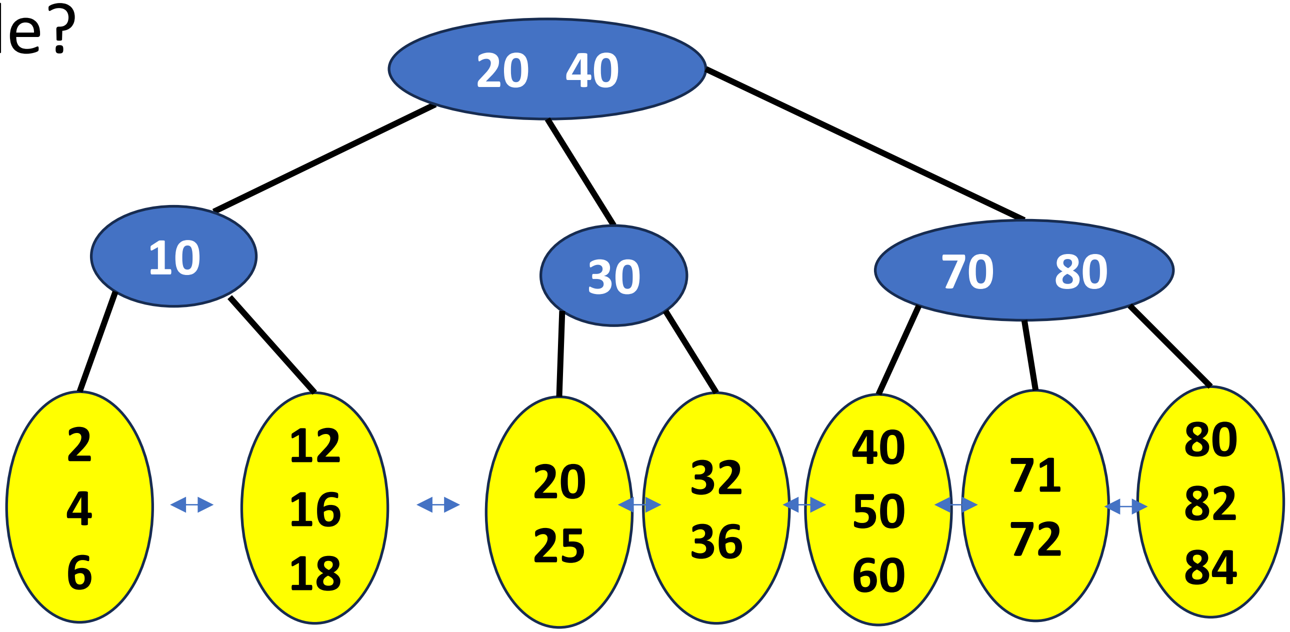
Insert a key=9 plus a pointer into the parent node.
→ There is no parent node.
→ become a new root.




The height is increased by one.

# Exercise

- Given the following $B^+$-tree of order 3 (2-3 tree). The capacity of a data node is 3.
  - Q1: Please insert 14. What are the keys in the second leaf node (from left to right)?
  - Q2: (Continue Q1) Please insert 86. What are the keys in the rightmost leaf node?



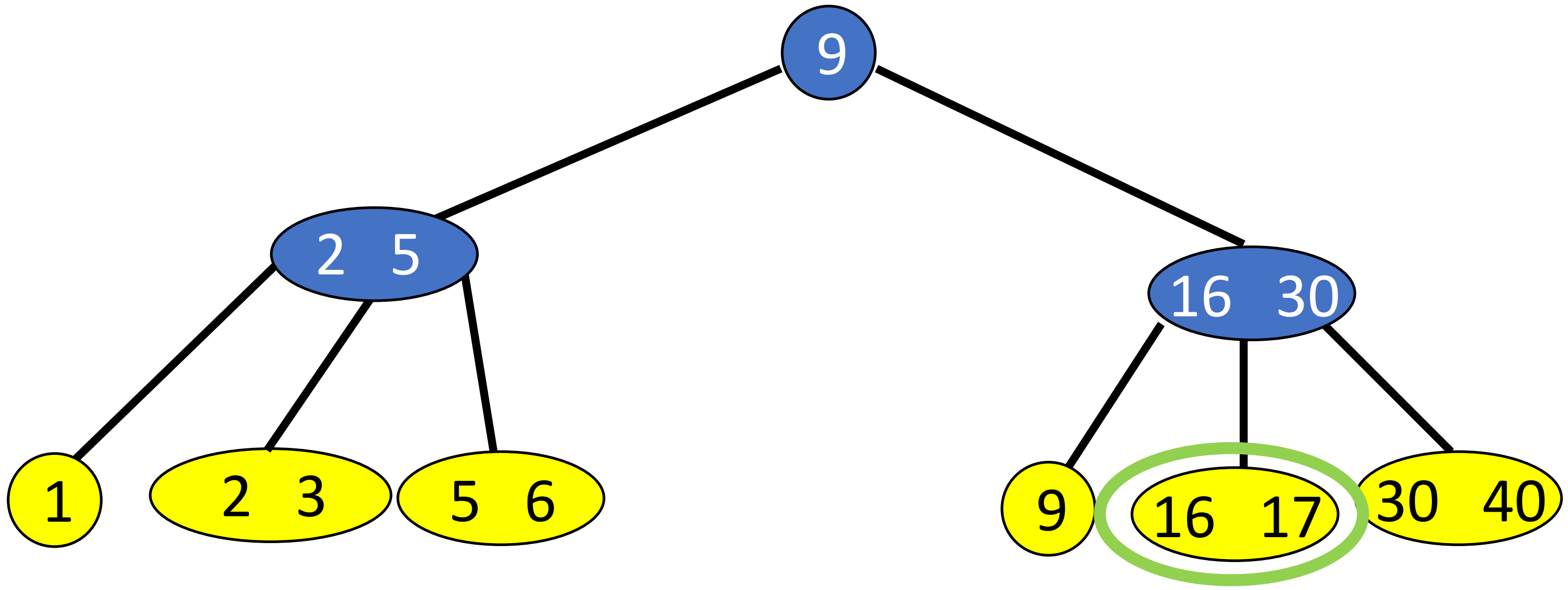

Please reply your answers of Q1-Q2 via the following link:

https://forms.gle/NyCVWyzZ6eXwVdYg9

Group members: 2~4 people

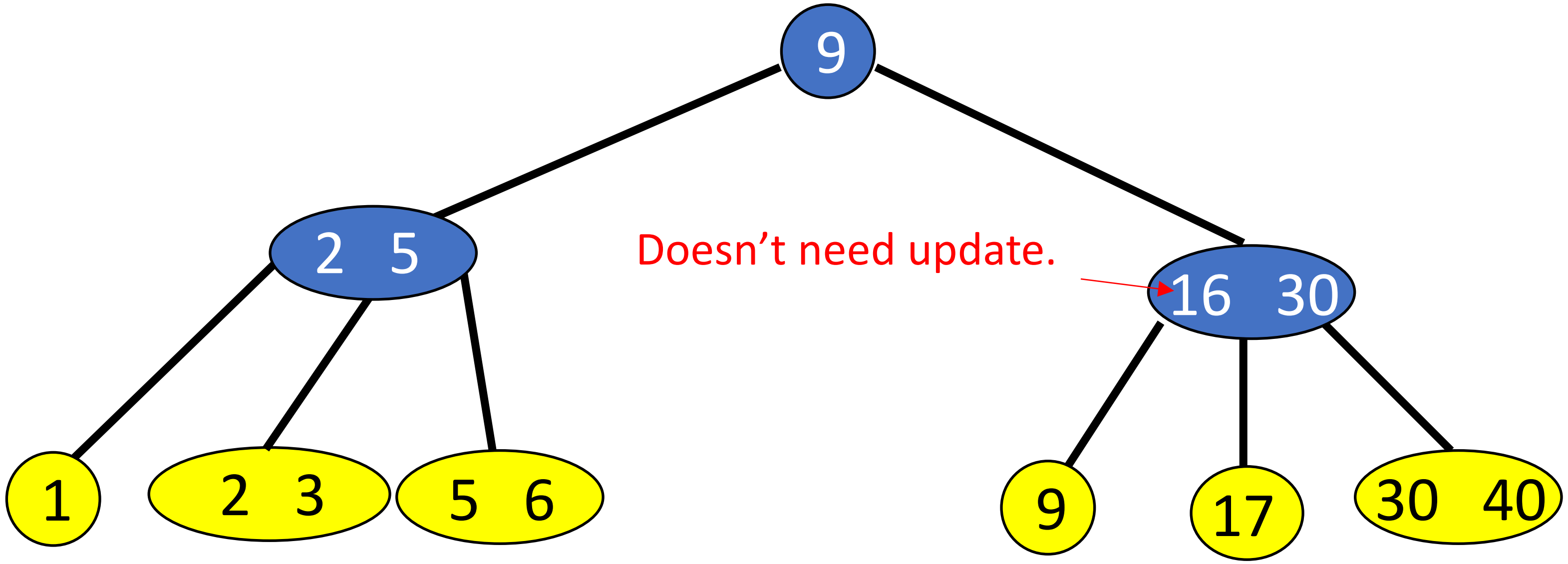
# Operation: Delete

- The deleted data pair is always in a leaf node.
- Delete the data pair and update the key in the parent node.
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete a pair with key = 16.



# Operation: Delete

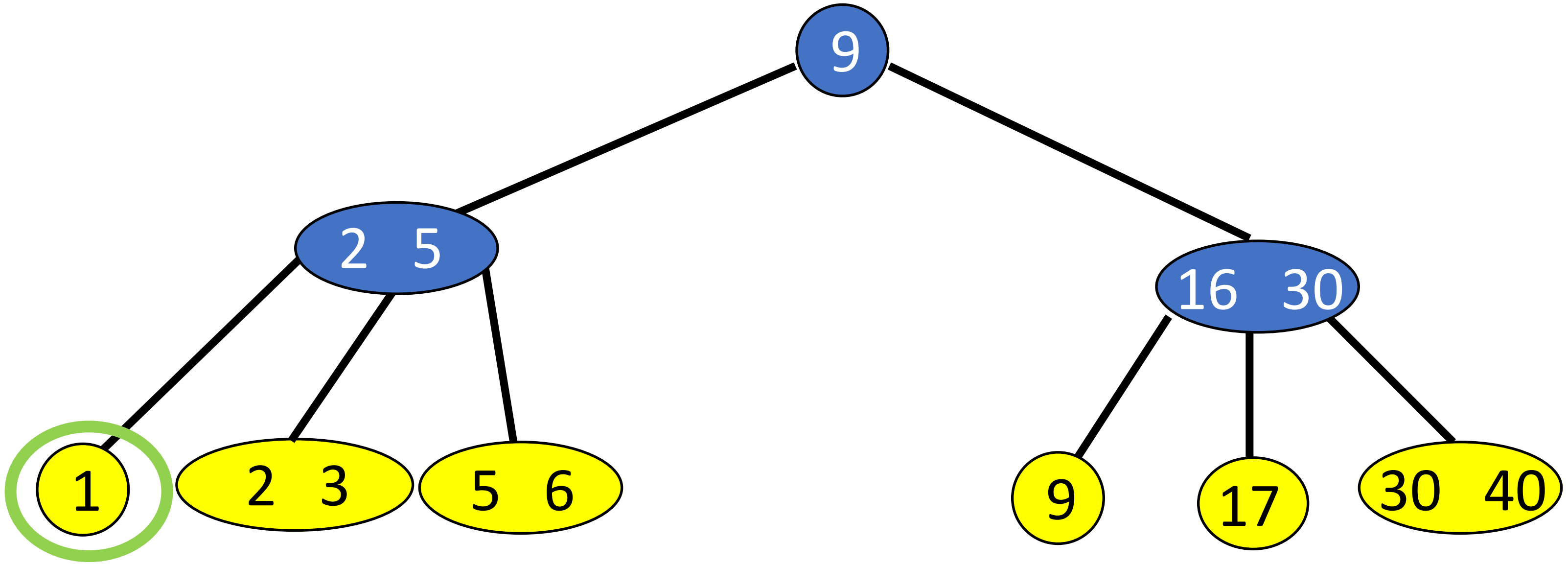
- The deleted data pair is always in a leaf node.
- Delete the data pair and update the key in the parent node.
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete a pair with key = 16.




The keys in the index nodes are not always the keys in the data nodes.

# Operation: Delete

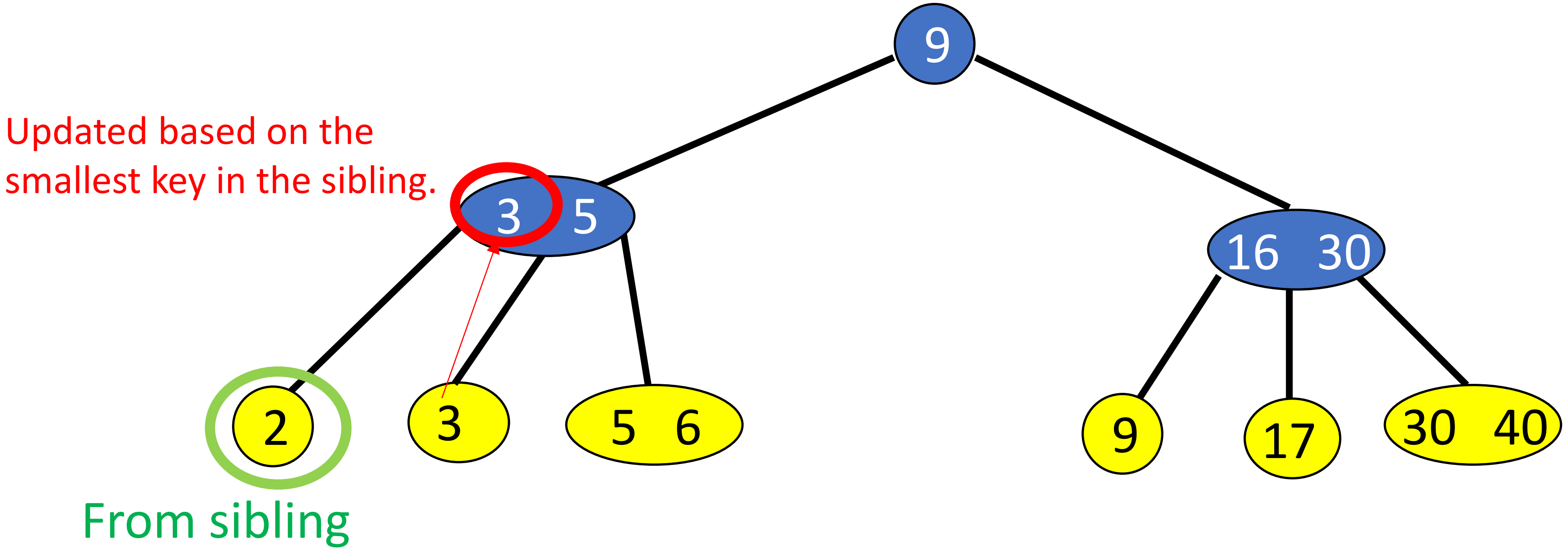
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Next, let's delete the pair with key = 1. (1/2)




After deletion, the node becomes deficient.
→ Obtain a data pair with a key ≥ 1 from sibling and update parent key.

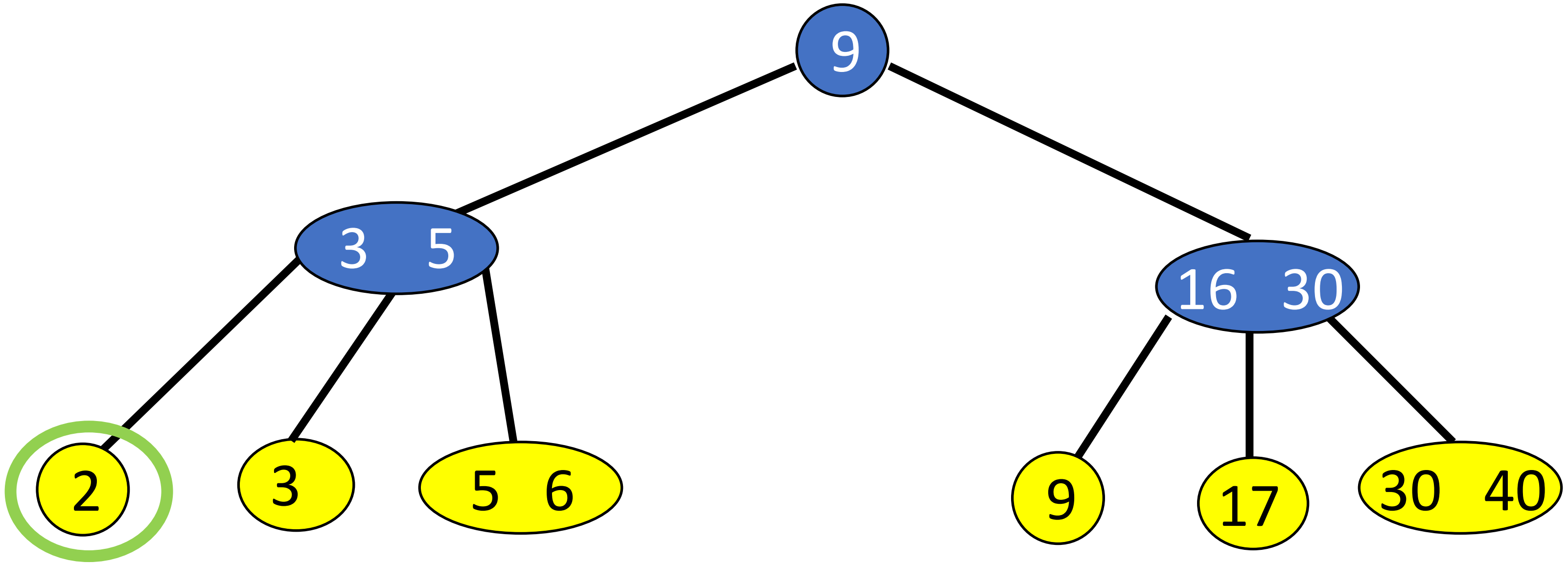
# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Next, let's delete the pair with key = 1. (2/2)



# Operation: Delete

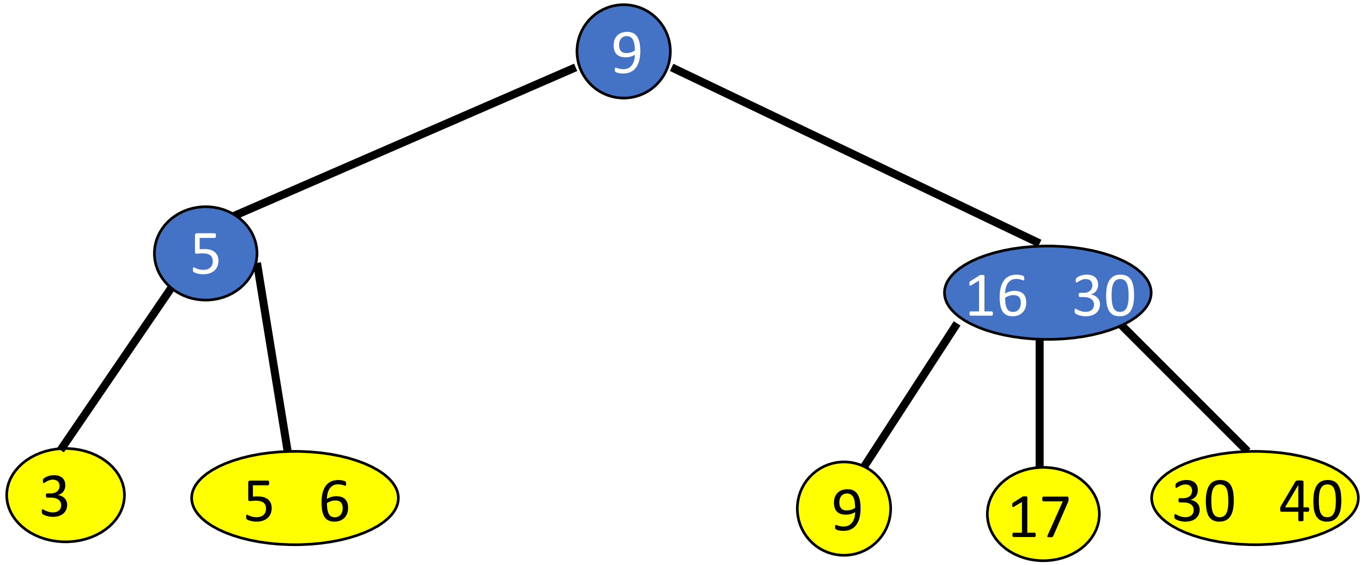
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 2. (1/2)




- After deletion, the node becomes deficient.
- Its sibling doesn't have enough data pair to move. (Cannot do rotation)
- Combine with sibling and delete in-between key in parent node. (Similar with combine)

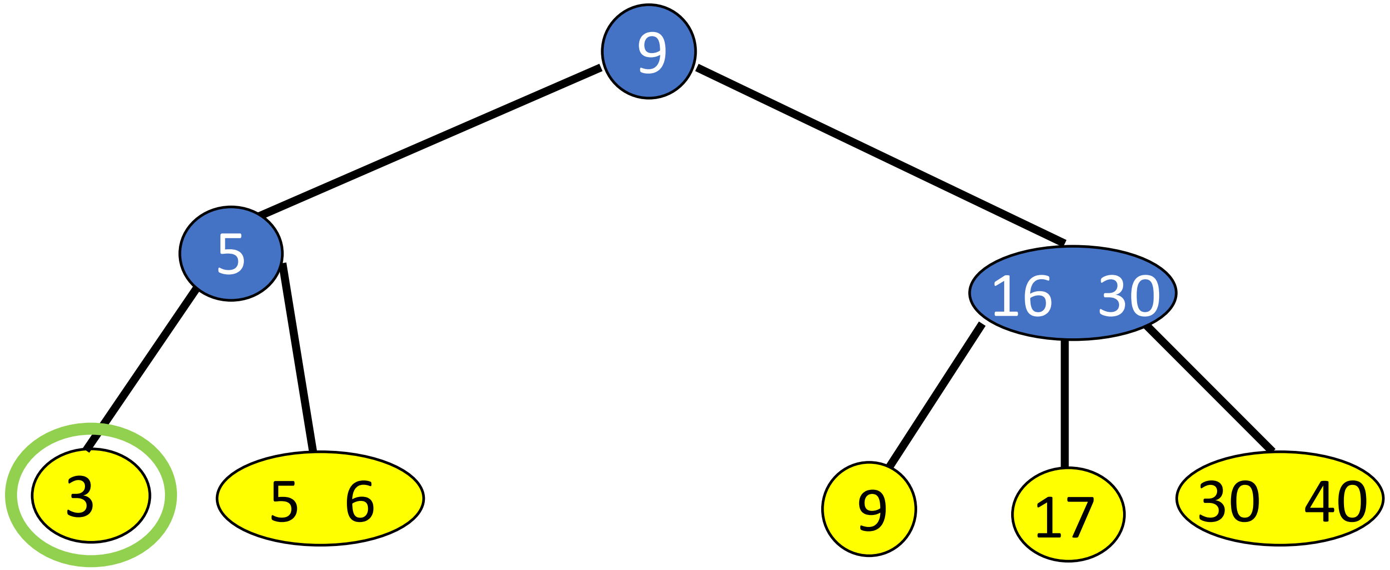
# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 2. (2/2)



# Operation: Delete

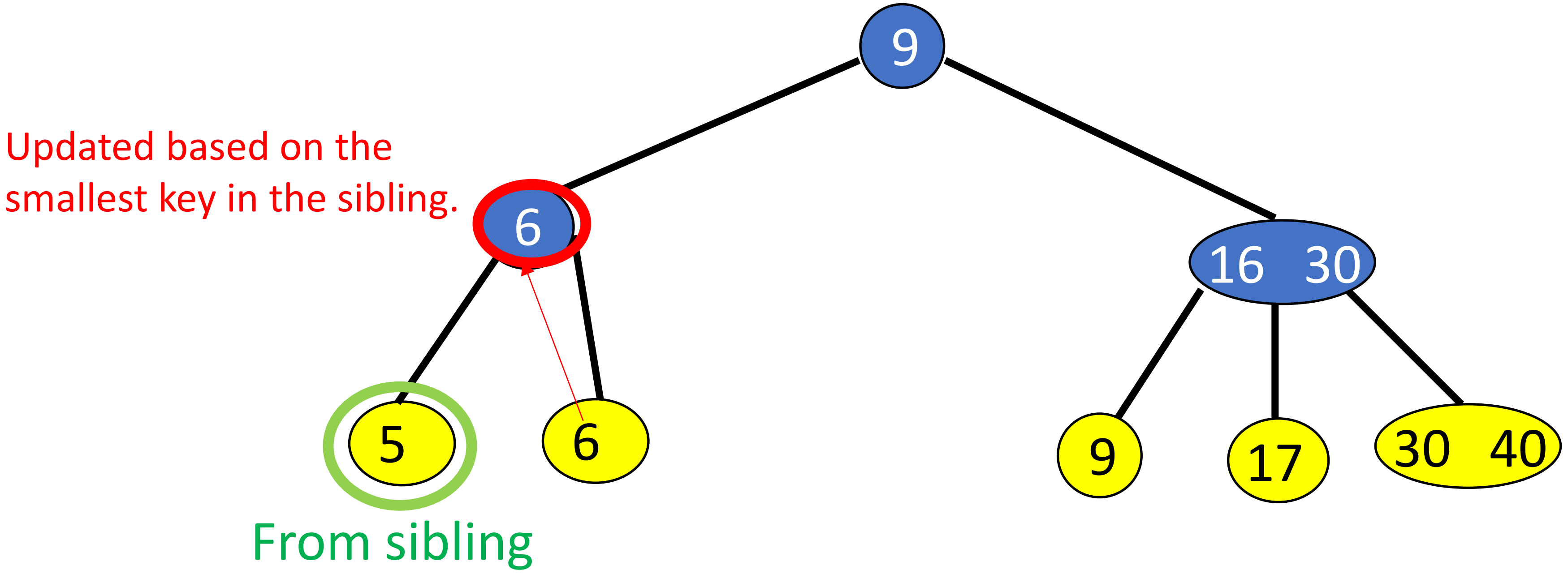
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 3. (1/2)




- After deletion, the node becomes deficient.
- Its sibling has enough data pairs to move. Get data pair from sibling and update keys in parent node. (Similar with rotation)

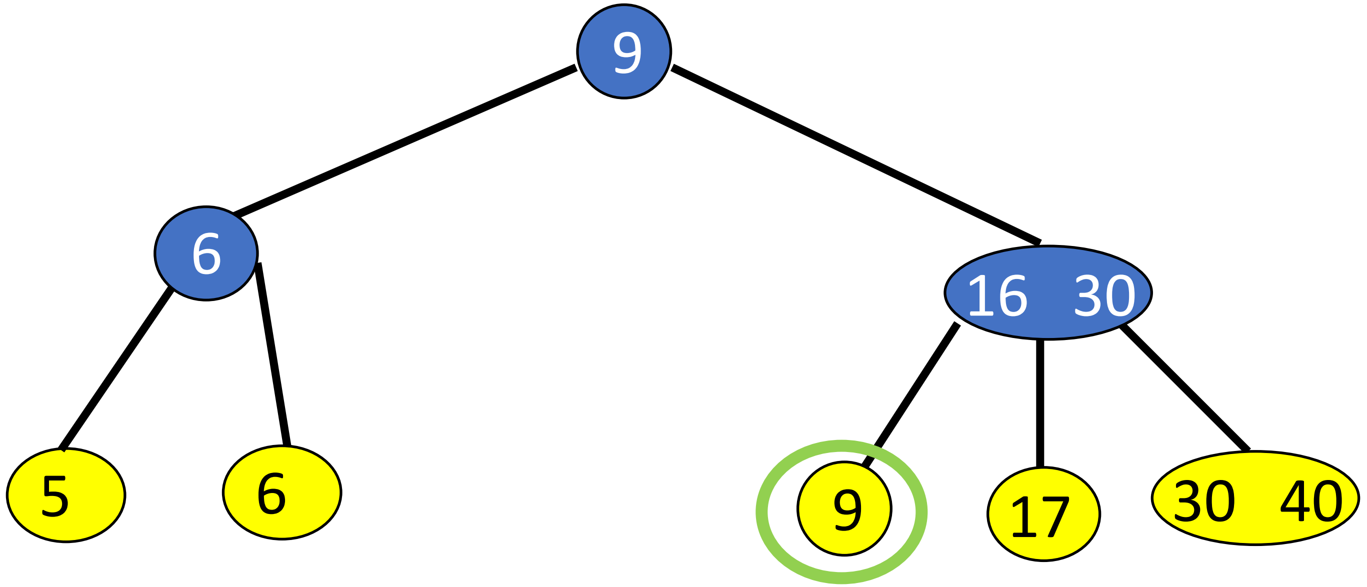
# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 3. (2/2)



# Operation: Delete

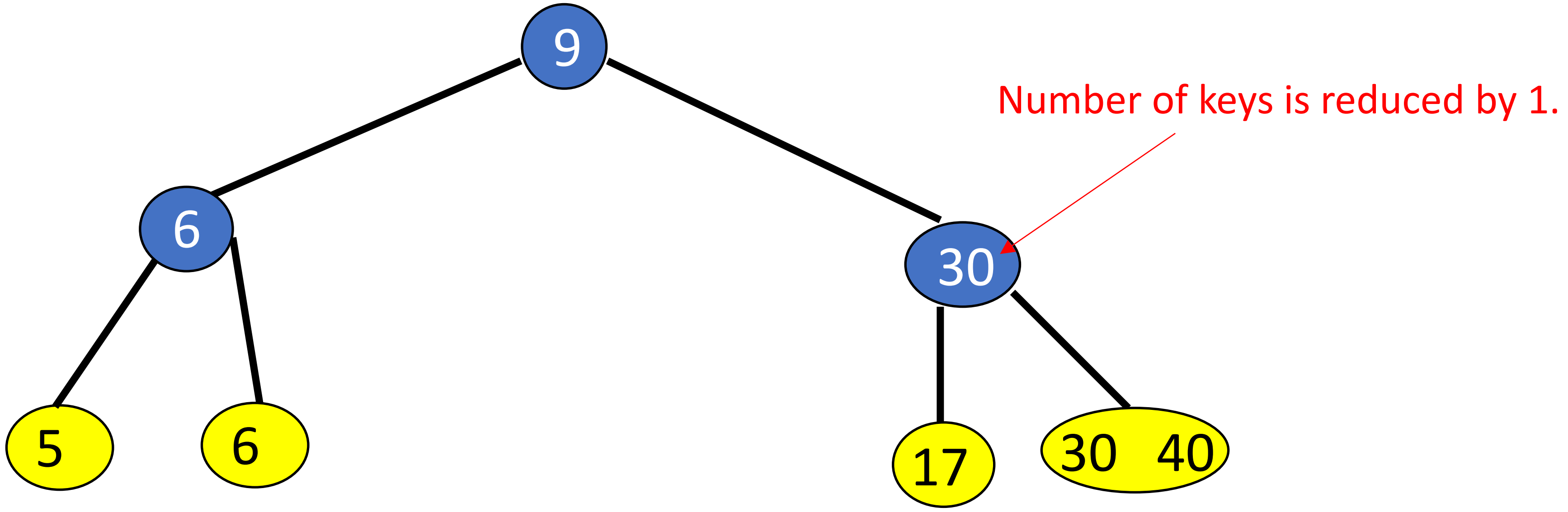
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 9. (1/2)




- After deletion, the node becomes deficient.
- Its sibling doesn't have enough data pair to move. (Cannot do rotation)
- Combine with sibling and delete in-between key in parent node. (Similar with combine)

# Operation: Delete

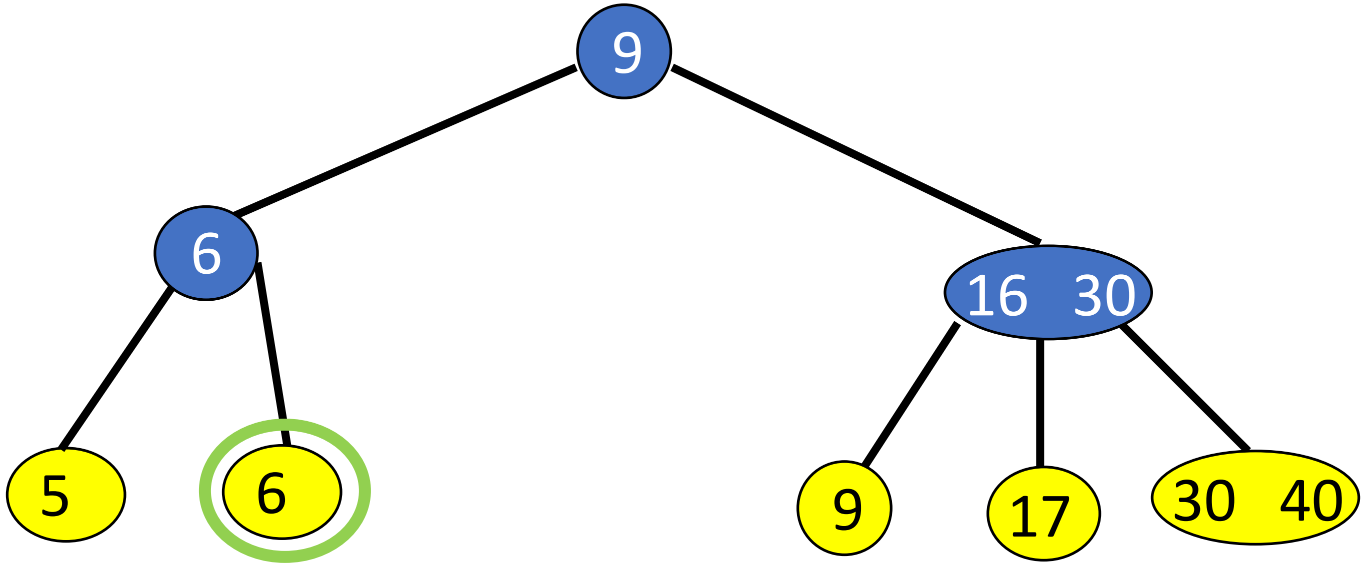
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 9. (2/2)




- After deletion, the node becomes deficient.
- Its sibling doesn't have enough data pair to move. (Cannot do rotation)
- Combine with sibling and delete in-between key in parent node. (Similar with combine)

# Operation: Delete

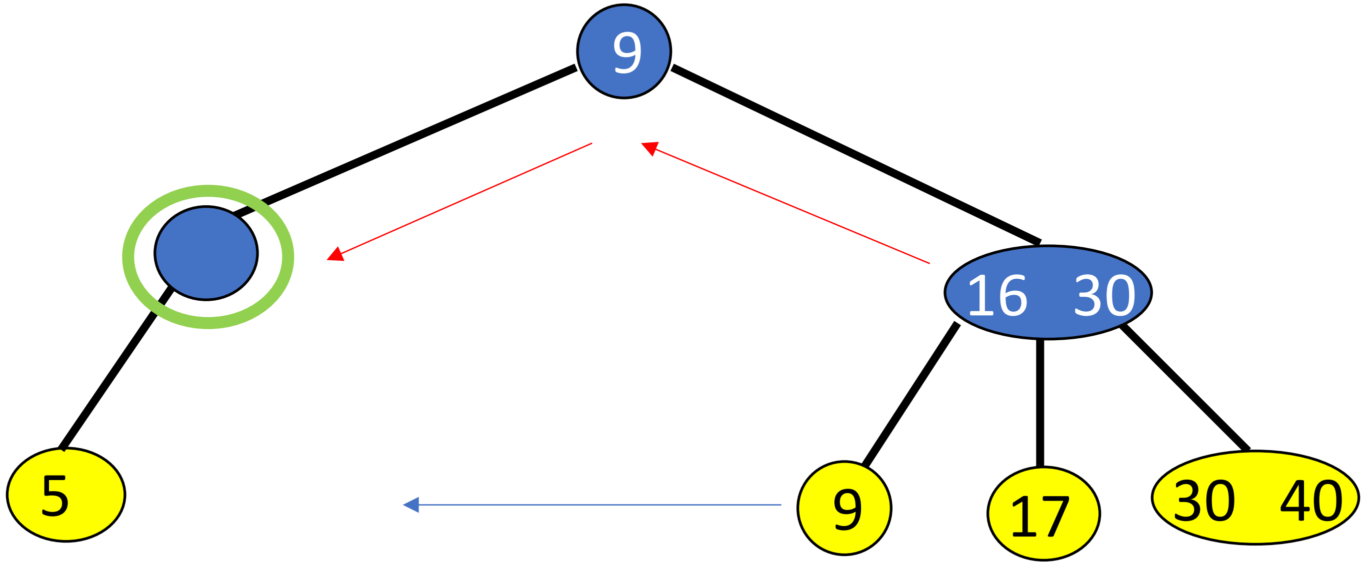
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 6. (1/3)




Merge with sibling, delete in-between key in parent.

# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 6. (2/3)

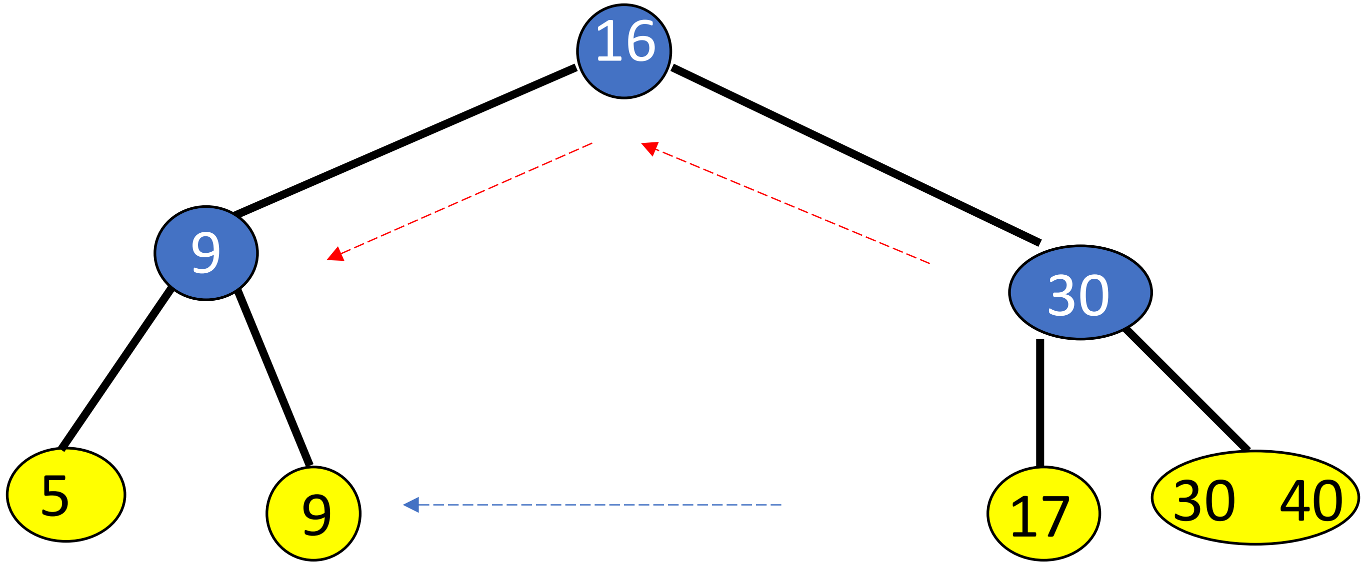



The index node becomes deficient.

Get data pair from its sibling, get parent's key, and move a key in the sibling to parent.

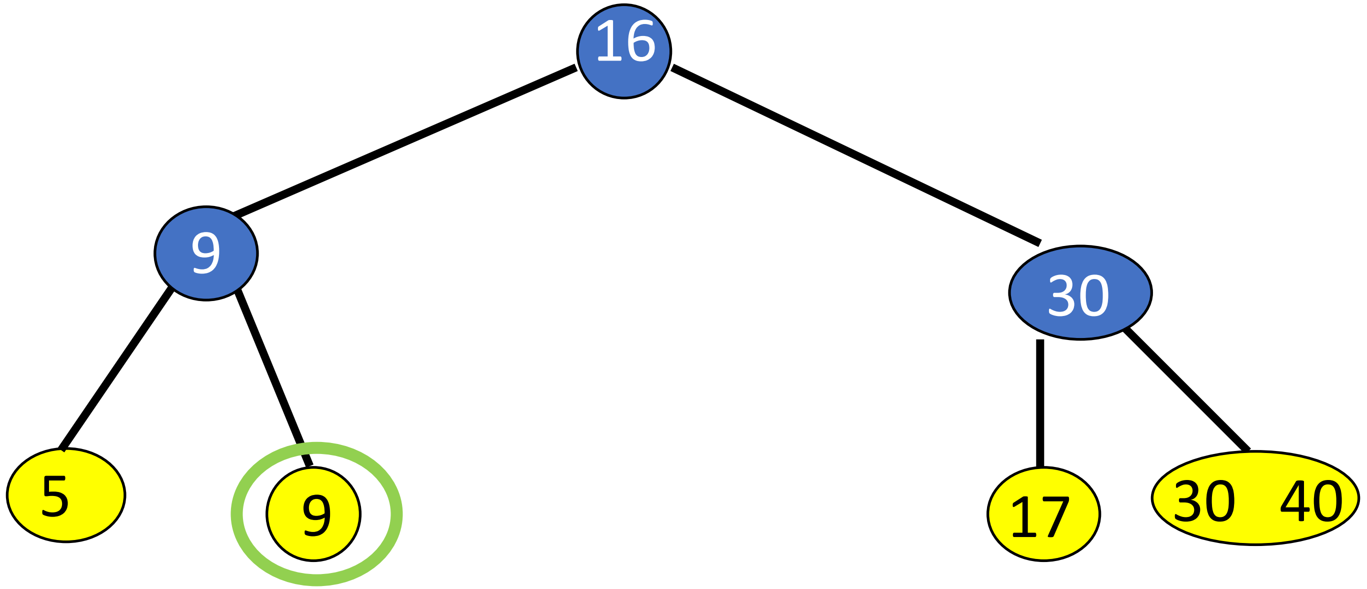
# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 6. (3/3)



# Operation: Delete

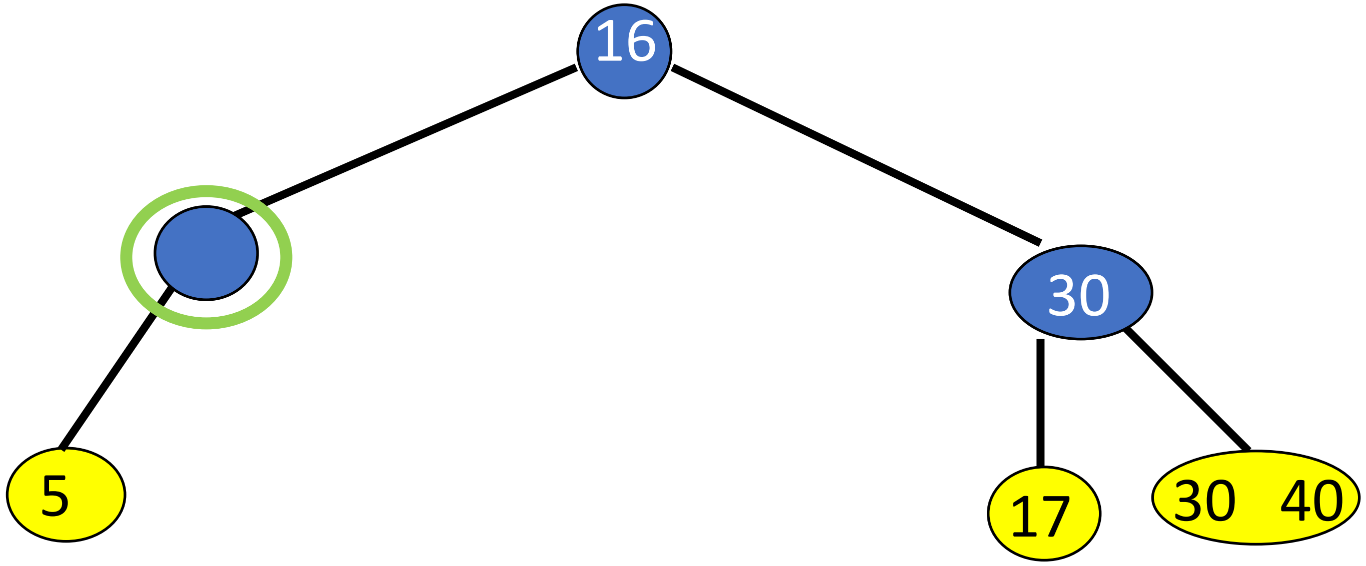
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 9. (1/3)




- After deletion, the data node becomes deficient.
- Its sibling doesn't have enough data pair to move. (Cannot do rotation)
- Combine with sibling and delete in-between key in parent node. (Similar with combine)

# Operation: Delete

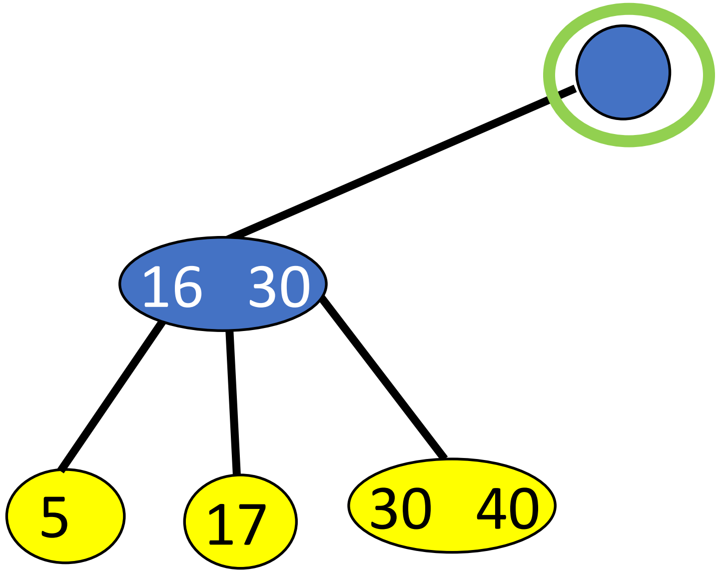
- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 9. (2/3)




- The index node becomes deficient.
- Its sibling doesn’t have enough keys to move. (Cannot do rotation)
- Combine with sibling and in-between key in parent node. (Similar with combine)

# Operation: Delete

- Example:
  - Assume that we have a 2-3 search tree and the capacity of a data node is 2.
  - Delete the pair with key = 9. (3/3)




- The index node becomes deficient.
- It's root. Discard.

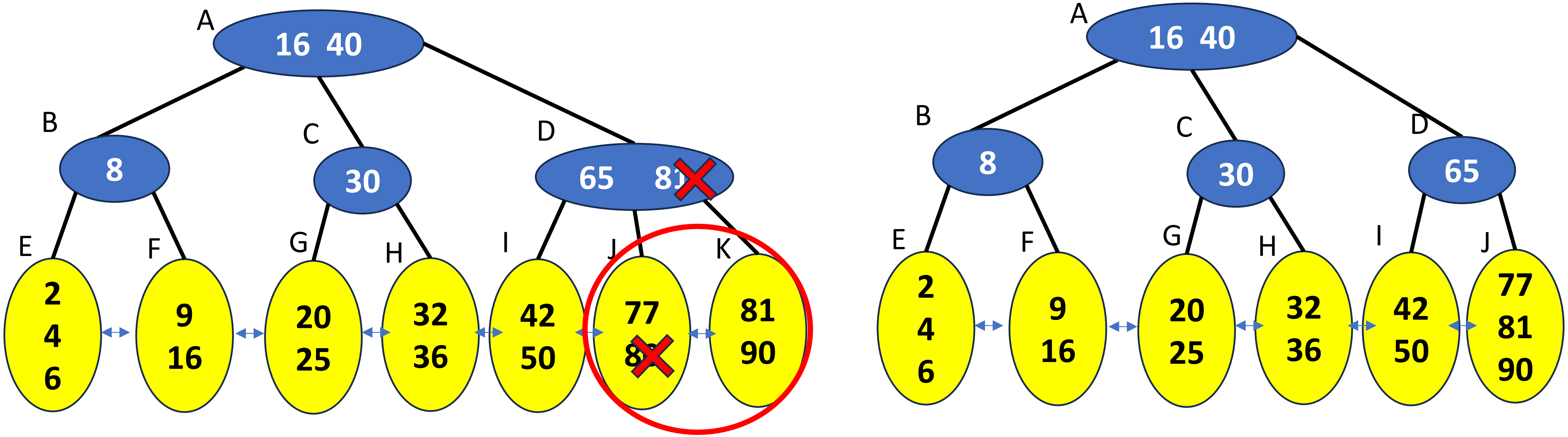
# Operation: Delete

- To increase efficiency, we can define the minimum occupancy for a data node.

- For example:
  - Capacity of a data node is $c$.
  - Each data node should have at least $\lceil c/2 \rceil$ data pairs.
  - When deletion causes the number of pairs < $\lceil c/2 \rceil$,
    - *Option 1*: Get data pairs from sibling and update parent key.
    - *Option 2*: Merge with siblings and delete parent key.

# Example of deletion

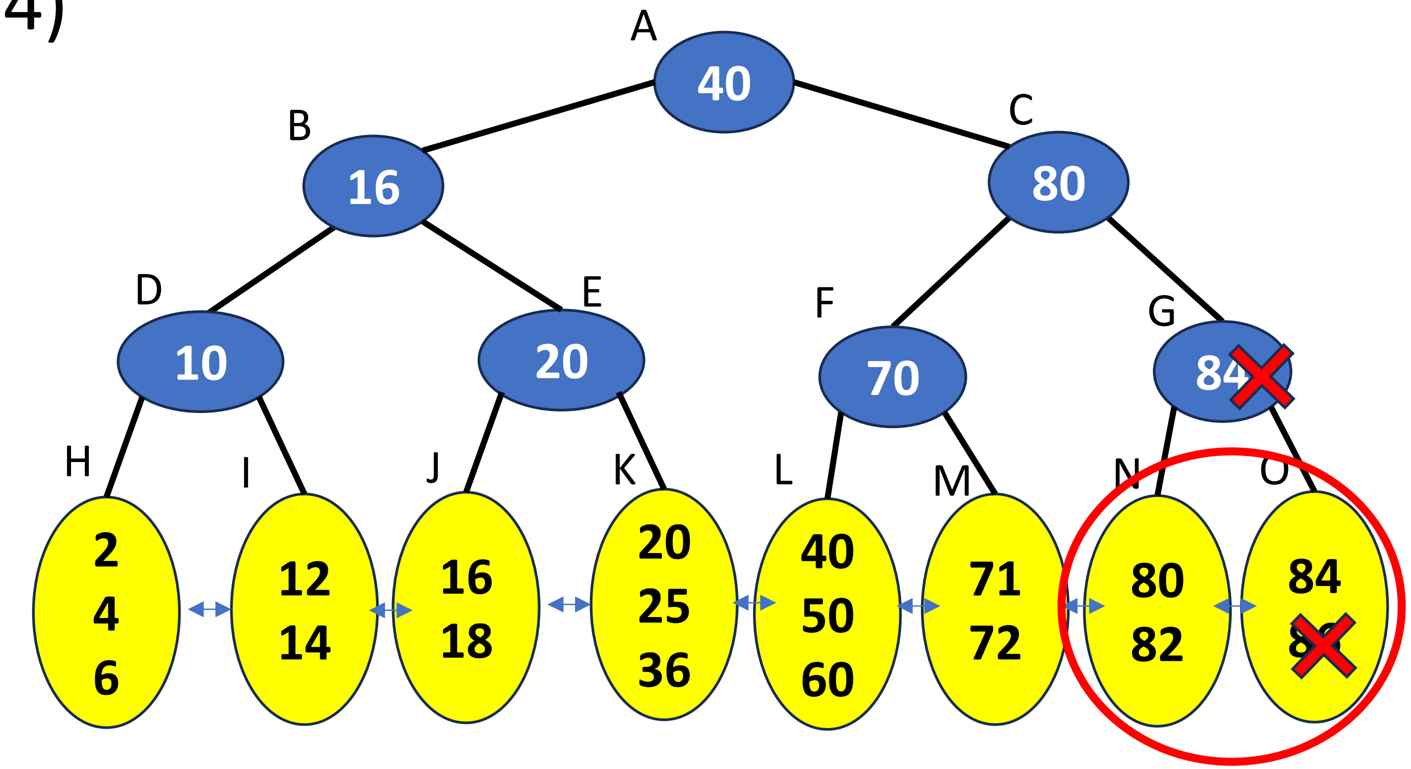
- B+-tree of order = 3. Capacity of a data node $c$ = 3.
- Minimum occupancy for a data node = $\lceil c/2 \rceil$ = 2.
- Delete 80.




- After deletion, only one data pair in node J. → Node J becomes deficient.
- Check sibling node K. It has only $\lceil c/2 \rceil$ nodes. → Do combine
- We combine nodes J and K and delete the in-between key in the parent node D.

# Example of deletion

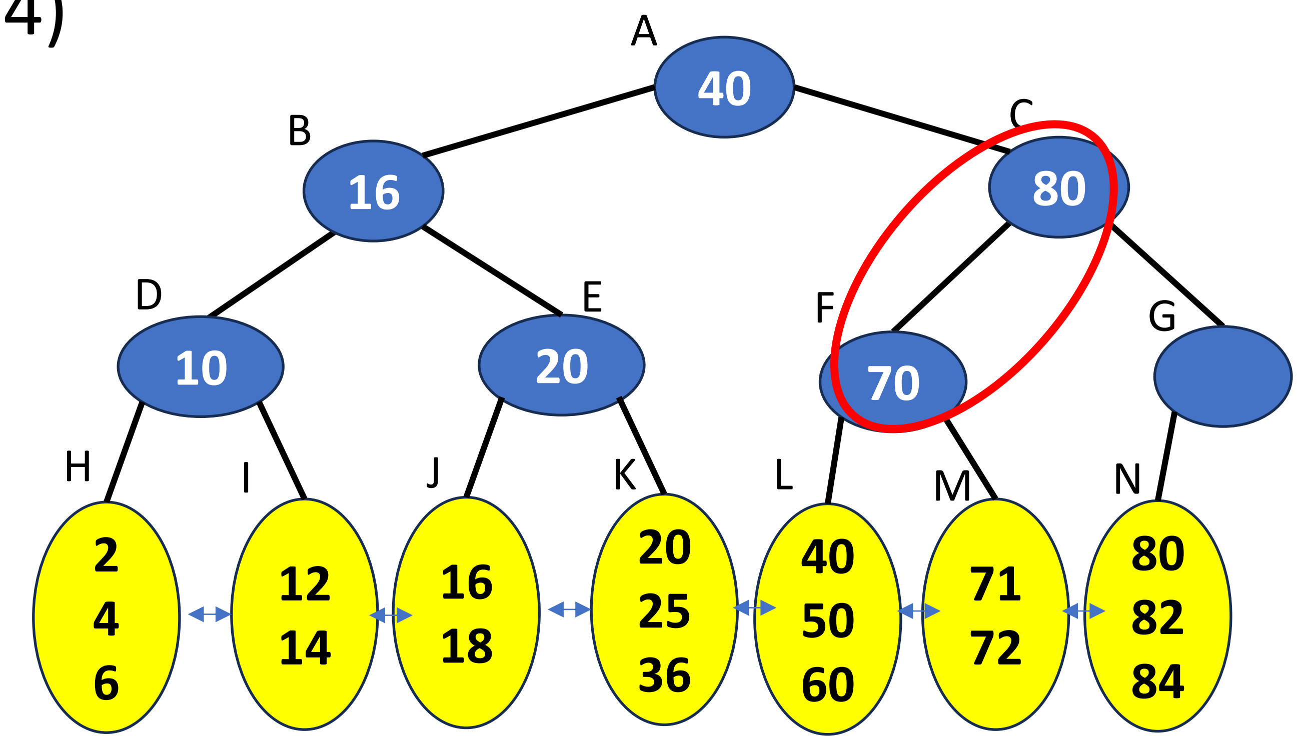
- $B^+$-tree of order = 3. Capacity of a data node $c$ = 3.
- Minimum occupancy for a data node = $\lceil c/2 \rceil$ = 2.
- Delete 86. (1/4)




- After deletion, only one data pair in node O. → Node O becomes deficient.
- Check sibling node N. It has only $\lceil c/2 \rceil$ nodes. → Do combine
- We combine nodes O and N and delete the in-between key in the parent node G.→ Node G becomes deficient.

# Example of deletion

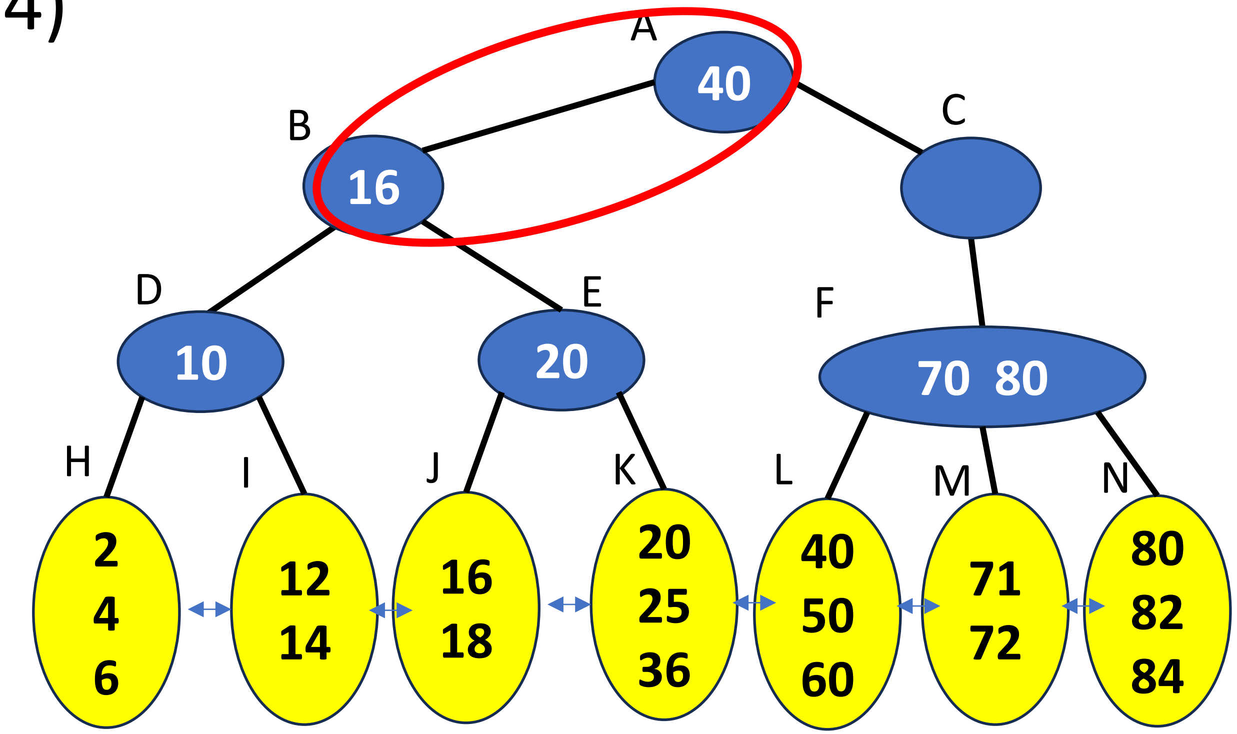
- $B^+$-tree of order = 3. Capacity of a data node $c$ = 3.
- Minimum occupancy for a data node = $\lceil c/2 \rceil$ = 2.
- Delete 86. (2/4)




- Check sibling node F. It has only $\lceil c/2 \rceil$ nodes. → Do combine
- We combine node F and the in-between key in the parent node C. → Node C becomes deficient.

# Example of deletion

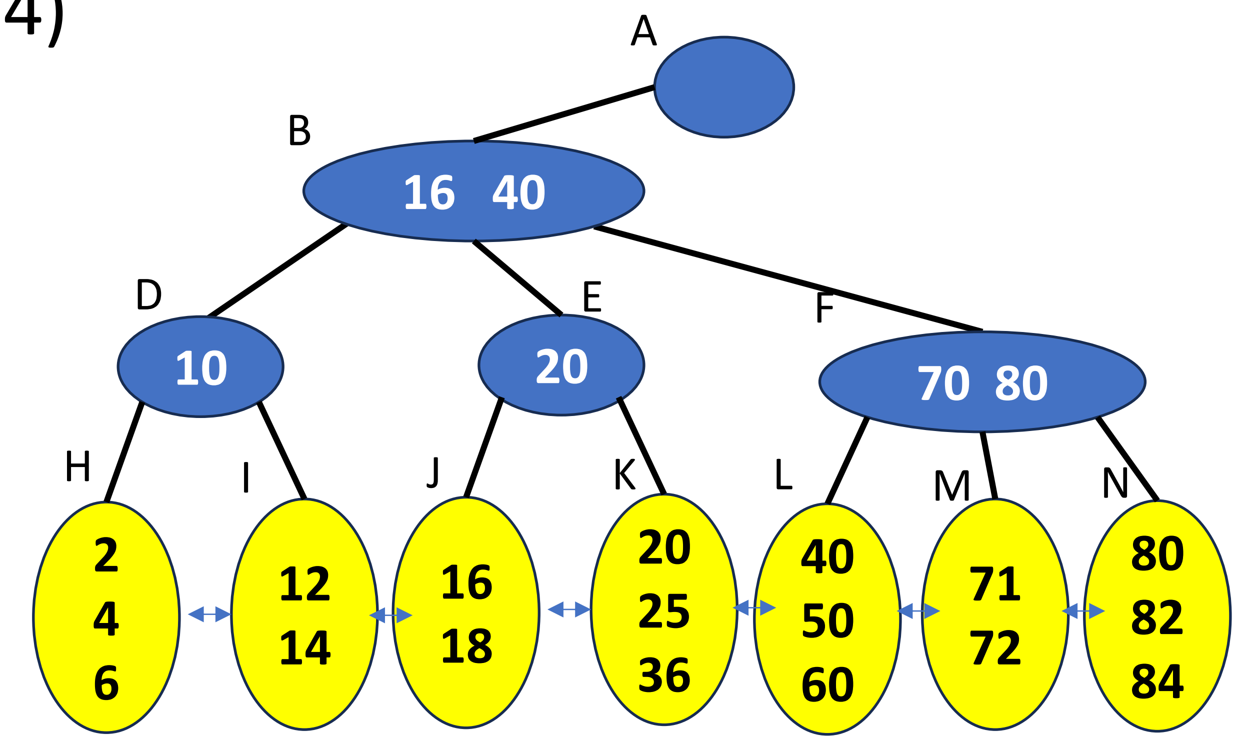
- B+-tree of order = 3. Capacity of a data node $c$ = 3.
- Minimum occupancy for a data node = $\lceil c/2 \rceil$ = 2.
- Delete 86. (3/4)




- We combine node F and the in-between key in the parent node C. → Node C becomes deficient.
- Check sibling node B. It has only $\lceil c/2 \rceil$ nodes. → Do combine
- We combine node B and the in-between key in the parent node A. → Node A becomes deficient.

# Example of deletion

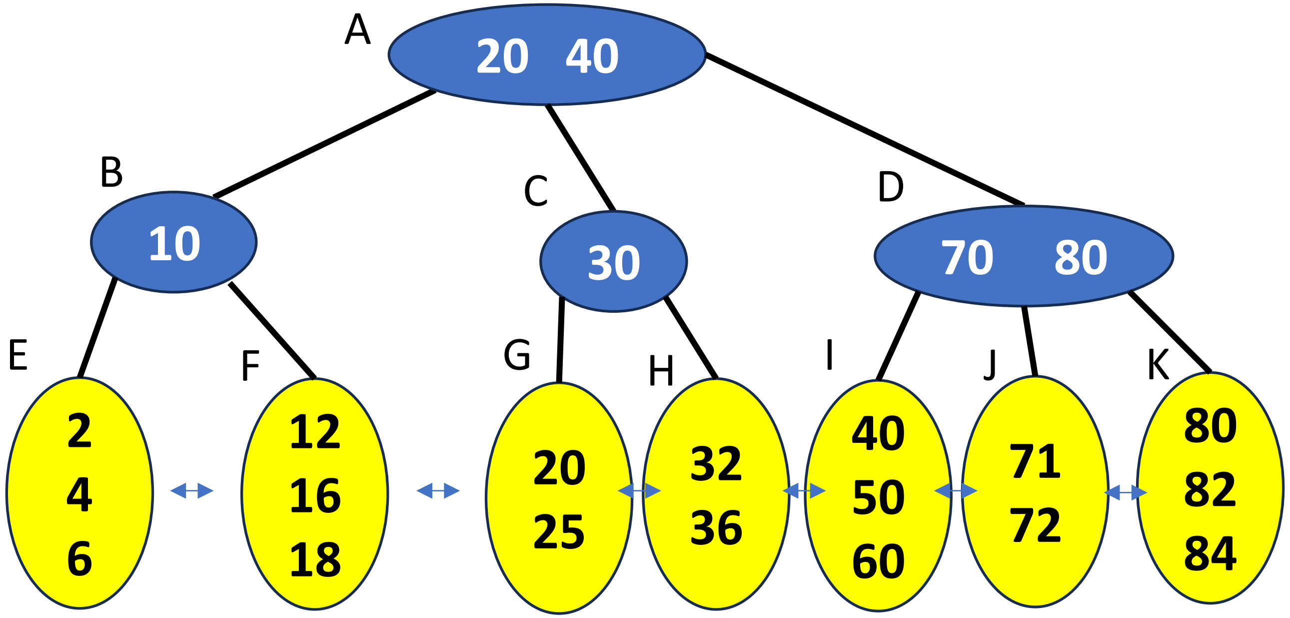
- $B^+$-tree of order = 3. Capacity of a data node $c$ = 3.
- Minimum occupancy for a data node = $\lceil c/2 \rceil$ = 2.
- Delete 86. (4/4)




- We combine node B and the in-between key in the parent node A. → Node A becomes deficient.
- Node A is root. Discard node A.

# Exercise

- Given the following B⁺-tree of order 3. The capacity *c* of a data node is 3. Minimum occupancy for a data node = $\lceil c/2 \rceil$.
  - Q3: After you delete the data pair with key=71, what are the keys in the node D?
  - Q4: (Continue Q3) What are the keys in node J?




Please reply your answers of Q3-Q4 via the following link:

https://forms.gle/NyCVWyzZ6eXwVdYg9

Group members: 2~4 people

# Summary

- $B^+$-tree
- Operations:
    - Insertion
    - Deletion