# $B^+$-Trees

# $B^+$-Trees

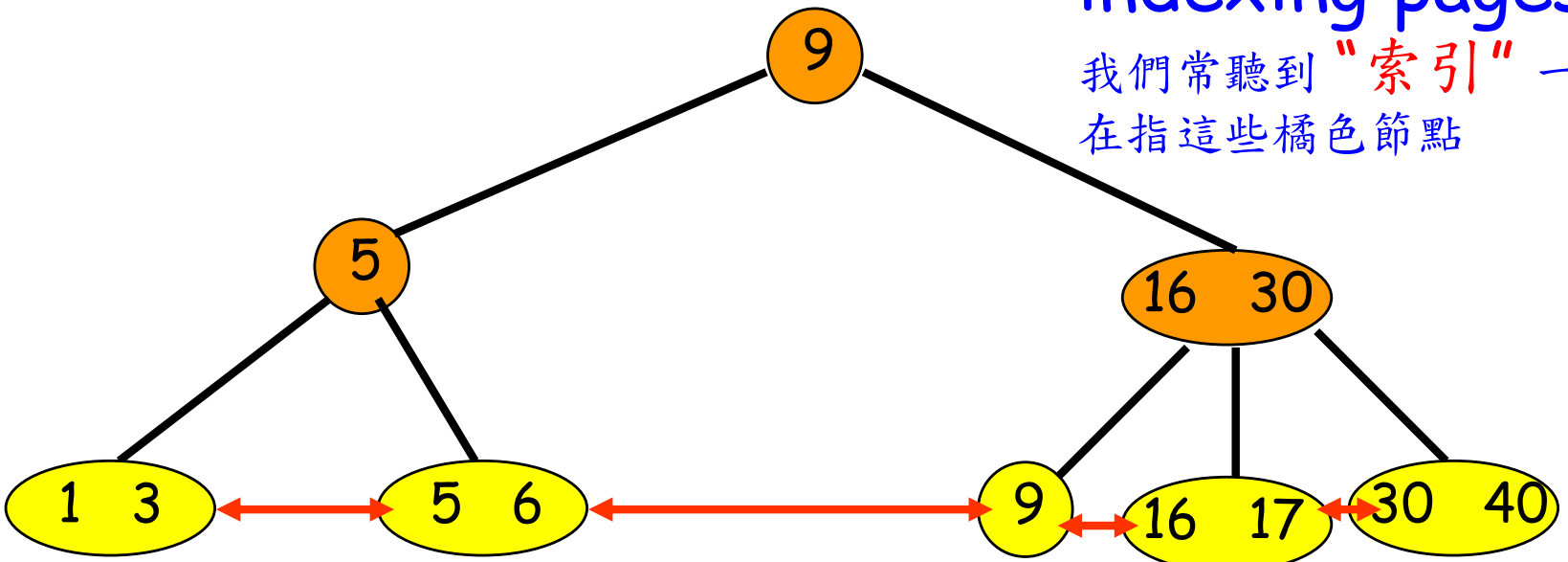
- 本質同B-trees. //B- and B+-trees兩詞有時沒有明顯區別
- Data pairs are in leaves only.
  - Leaves form a doubly-linked list.
- Remaining nodes have following structure:

$$j\ a_0\ k_1\ a_1\ k_2\ a_2\ \ldots\ k_j\ a_j$$

//只有keys
//沒有values

- j = number of keys in node.
- $a_i$ is a pointer to a subtree.
- $k_i$ "$>$" largest in $a_{i-1}$ (e.g., $k_2$)
- $k_i$ "$<=$" smallest key in subtree $a_i$ (e.g., $k_2$)

# Example B+-tree




indexing pages

我們常聽到"索引"一詞，即在指這些橘色節點

data pages

index node

leaf/data node

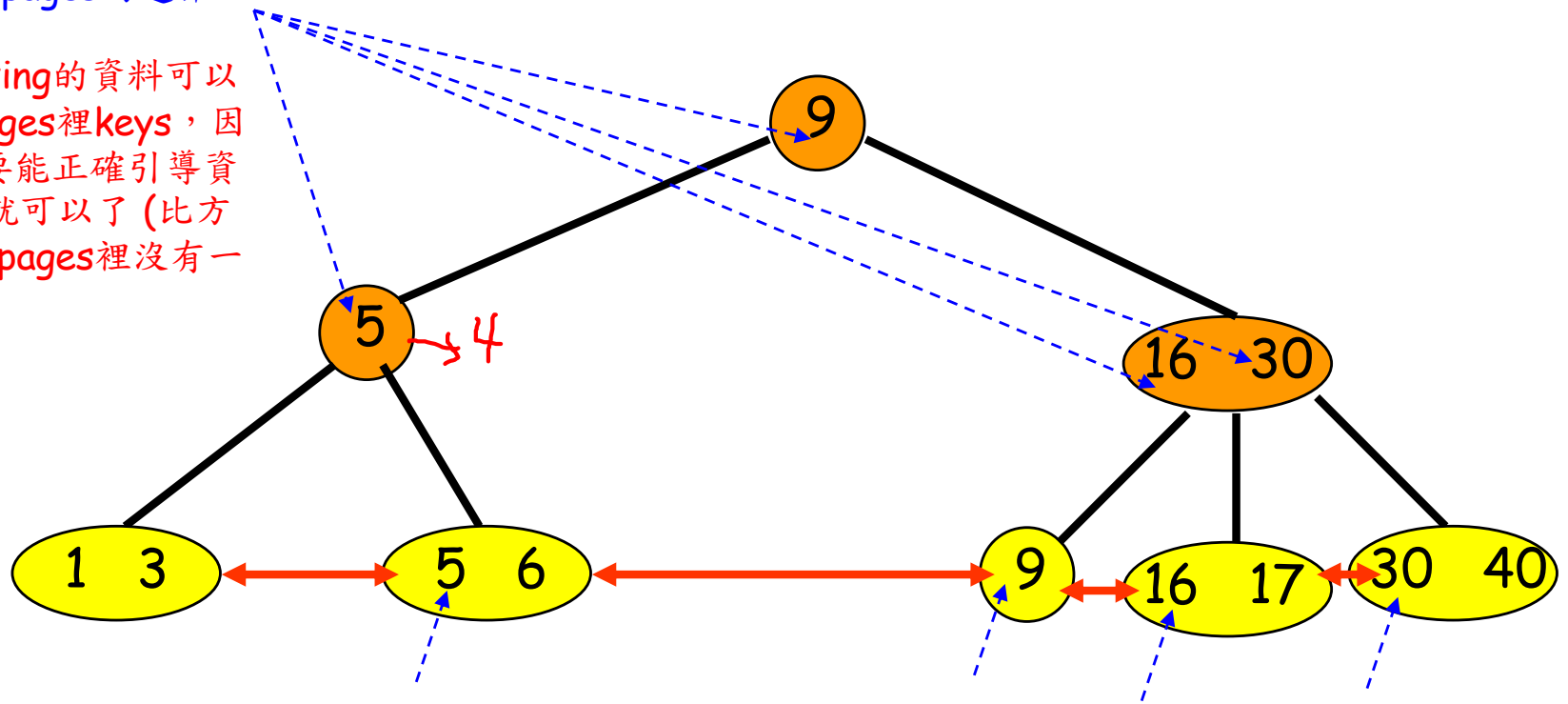
Pages:

- Taught in OS
- 記憶體區塊配置的單位, 4 Kbytes each, for example

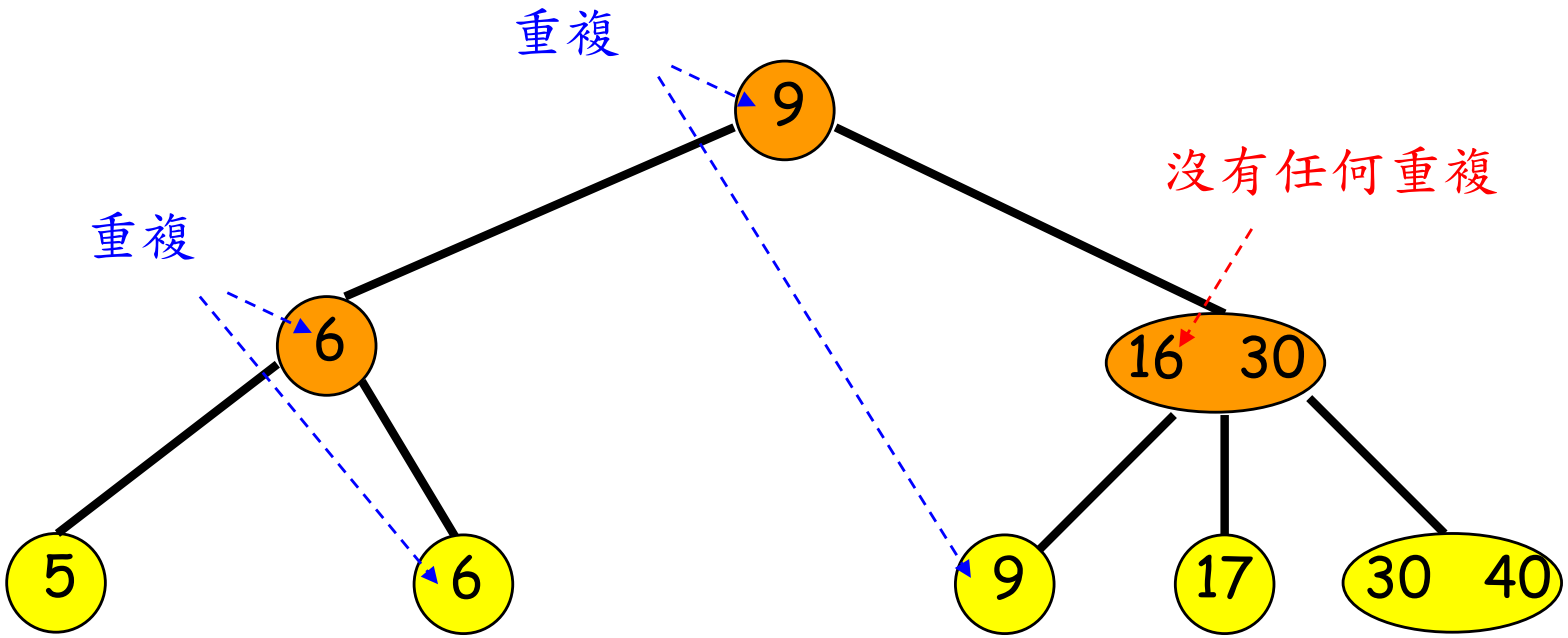
# B+-Tree的一些觀察

- 可能部分資料的keys成為indexing
- 該些indexing的資料"有時"為data pages的邊界資料
- 搜尋資料時，須走到data page使能取得真正的value值

- 剛好取自Data pages的邊界資料
- 事實上，indexing的資料可以不同於data pages裡keys，因為indexing只要能正確引導資料搜尋的方向就可以了 (比方5->4，但data pages裡沒有一筆資料是4)

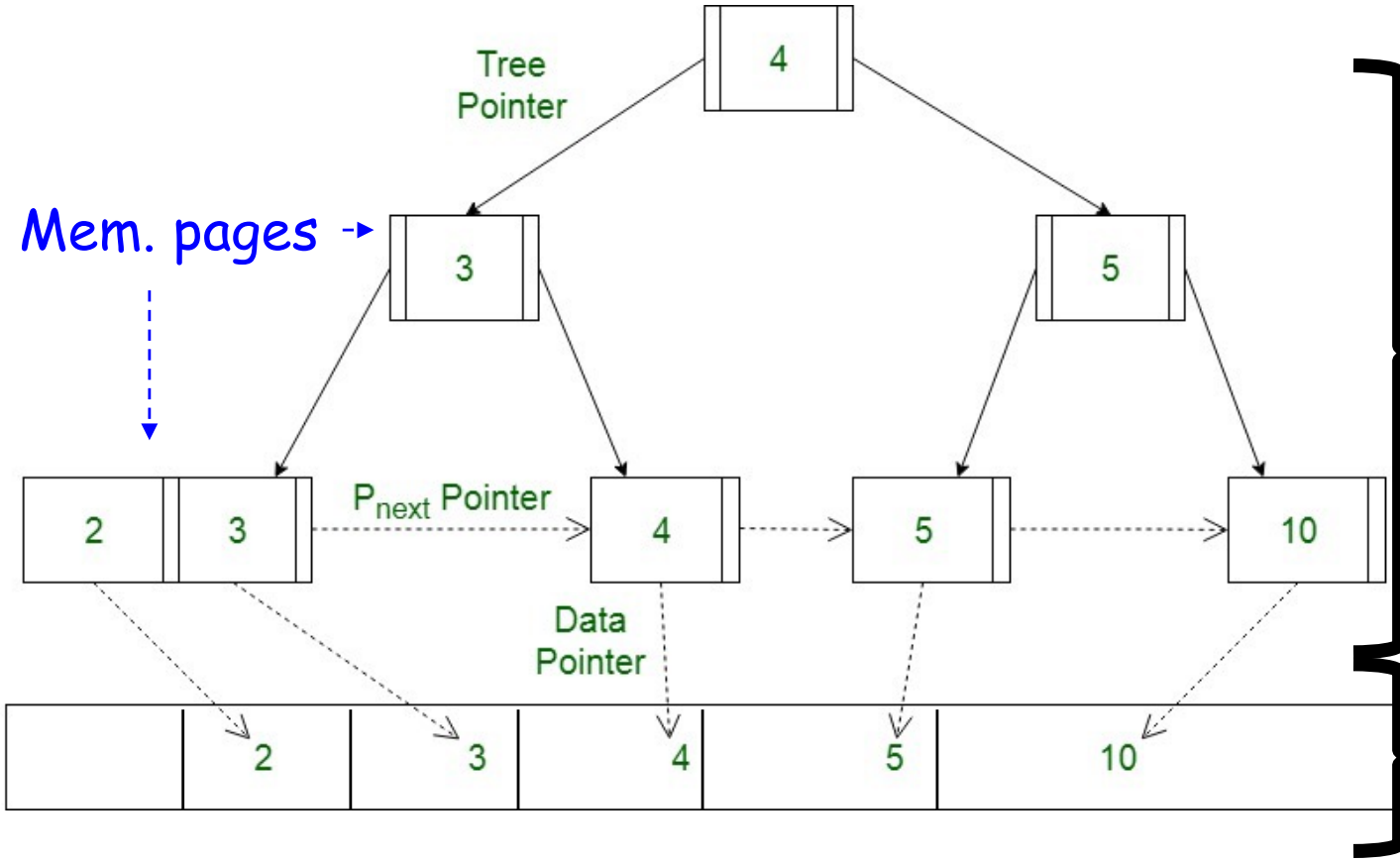


# Another Example



# Why B+-tree (vs B-tree)?

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

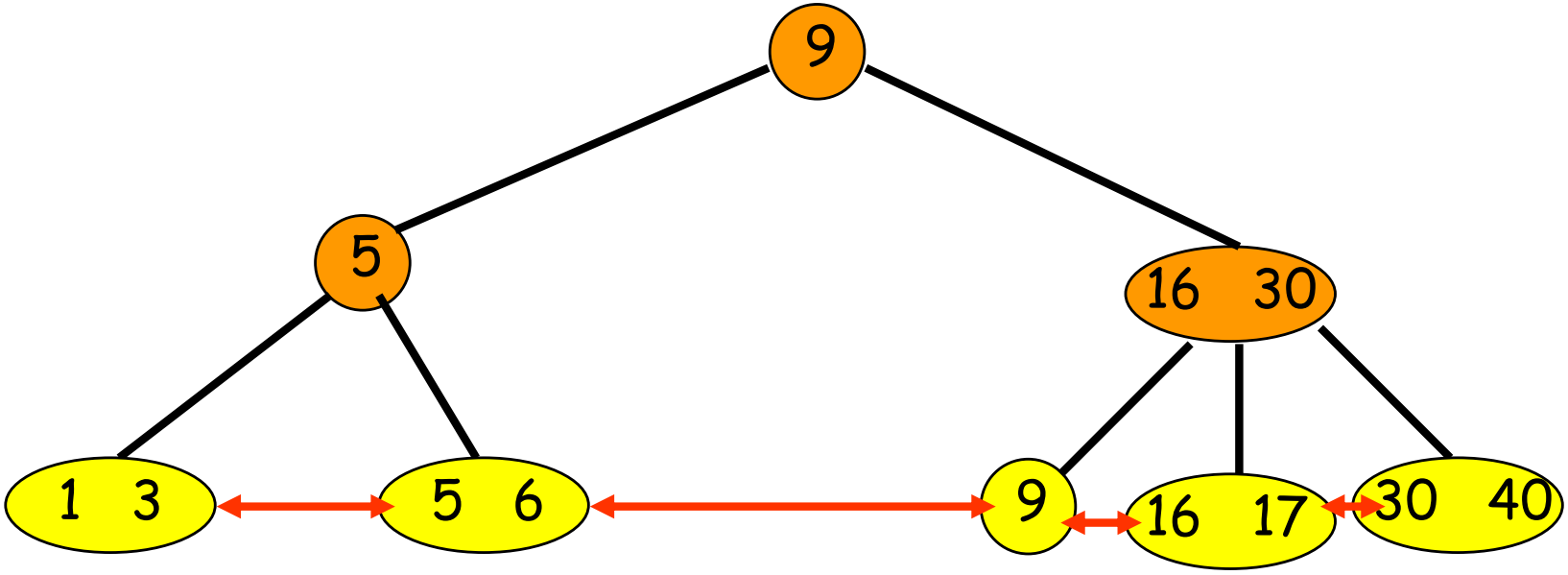



- multiway search tree (i.e., B-tree)
- indexing pages in mem.

兩者資料存取的行為不同

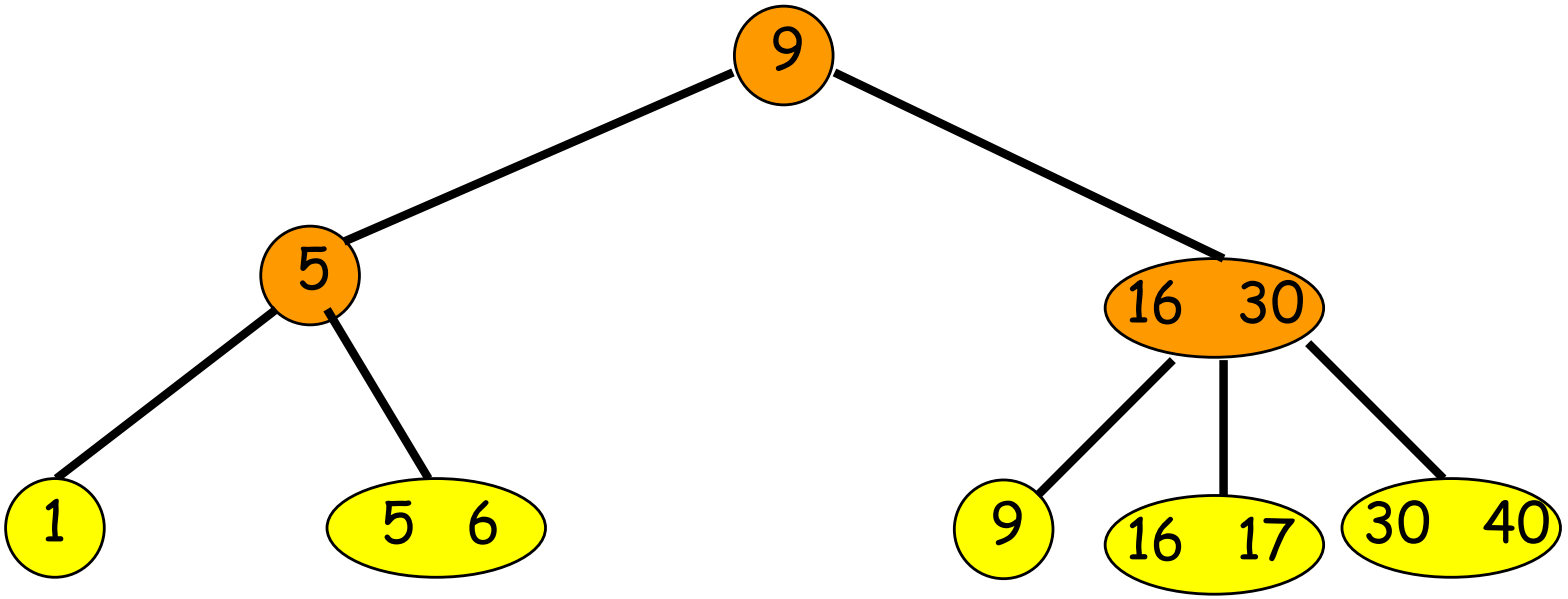
data pages in disk
// 部分會有副本在記憶體
// 使縮短存、取的時間

# B+-tree—Search (Pin & Range)




- Pin search: e.g., key = 5
- Range search: e.g., 6 <= key <= 20

# B+-tree—Insert

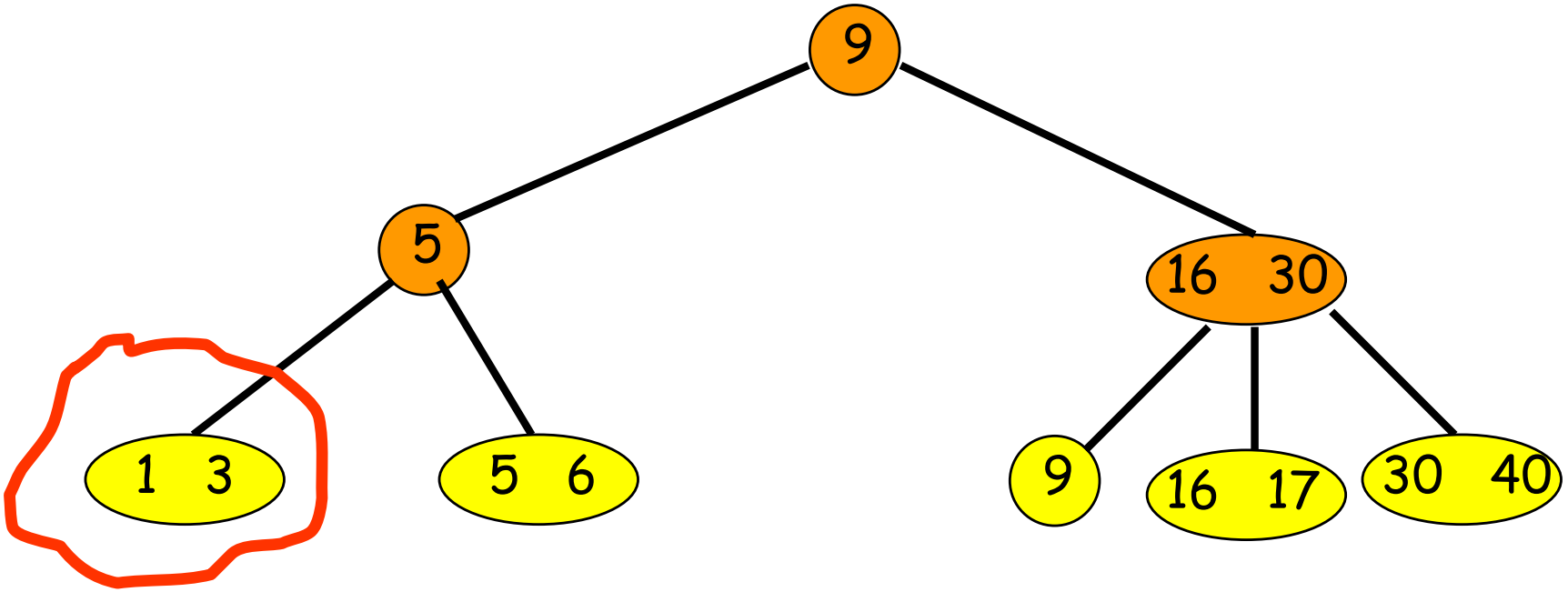



Insert 10

# Insert and Delete

- Multi-way search tree, essentially
  - 之前已經討論過, i.e., B-tree

# Insert



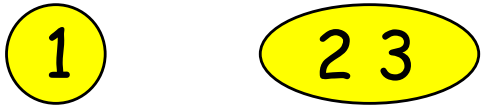

- Insert a pair with key = 2.
- New pair goes into a 3-node.

- 假設每個data page最多存兩筆資料，
- 並使用2-3 search tree

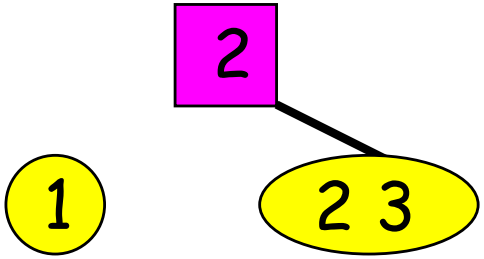
# Insert Into A 3-node

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

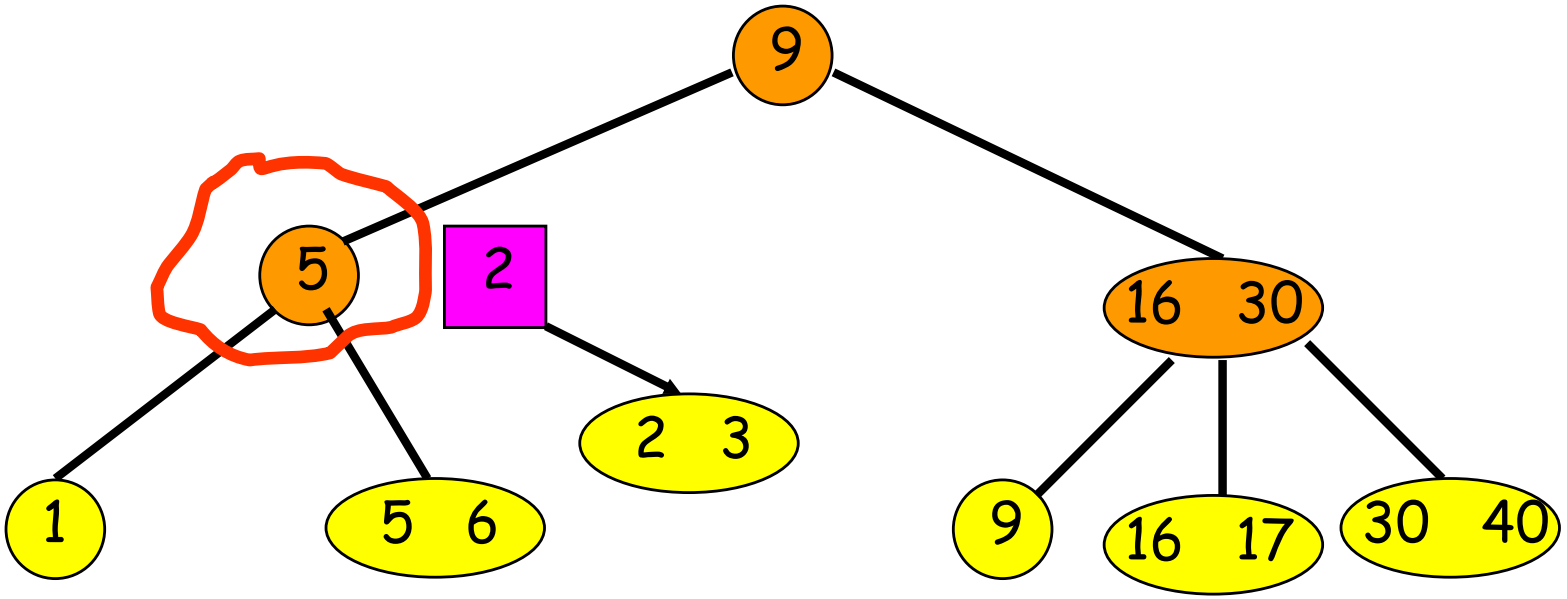

- Split into two nodes.




- Insert smallest key in new node and pointer to this new node into parent.

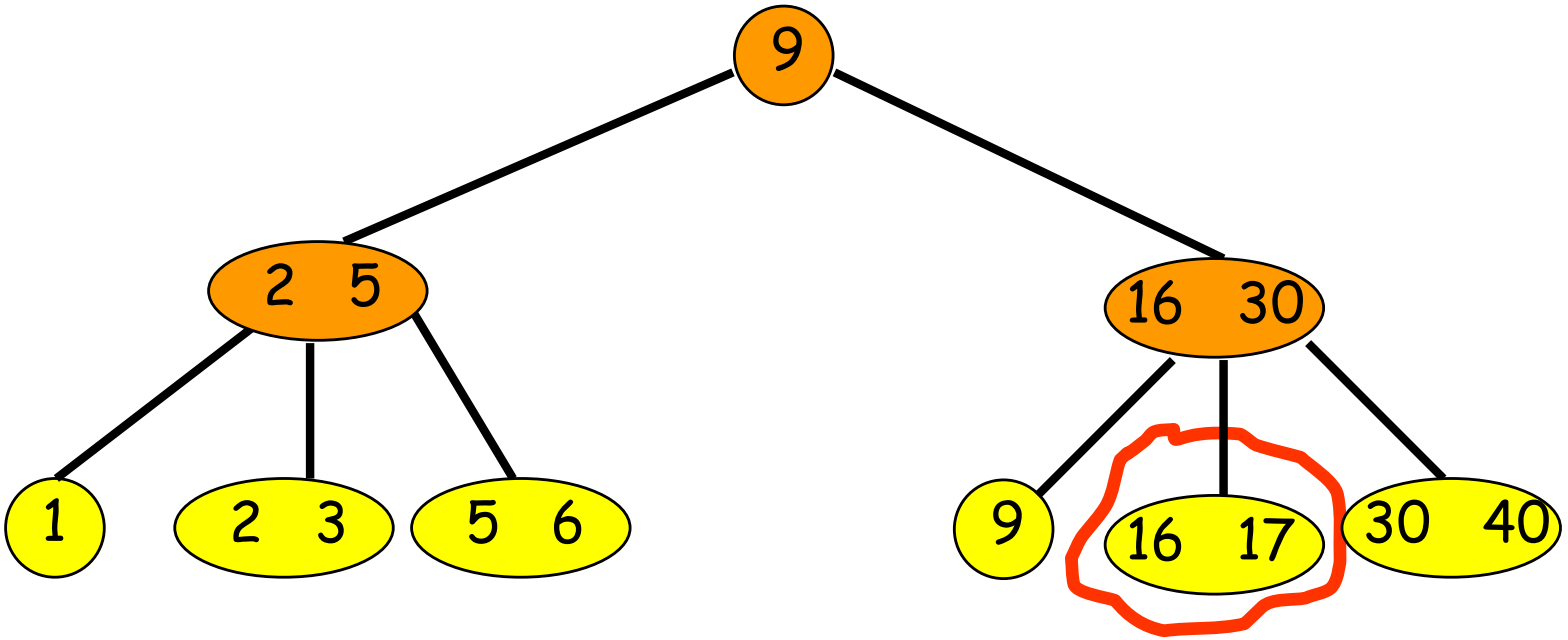


# Insert



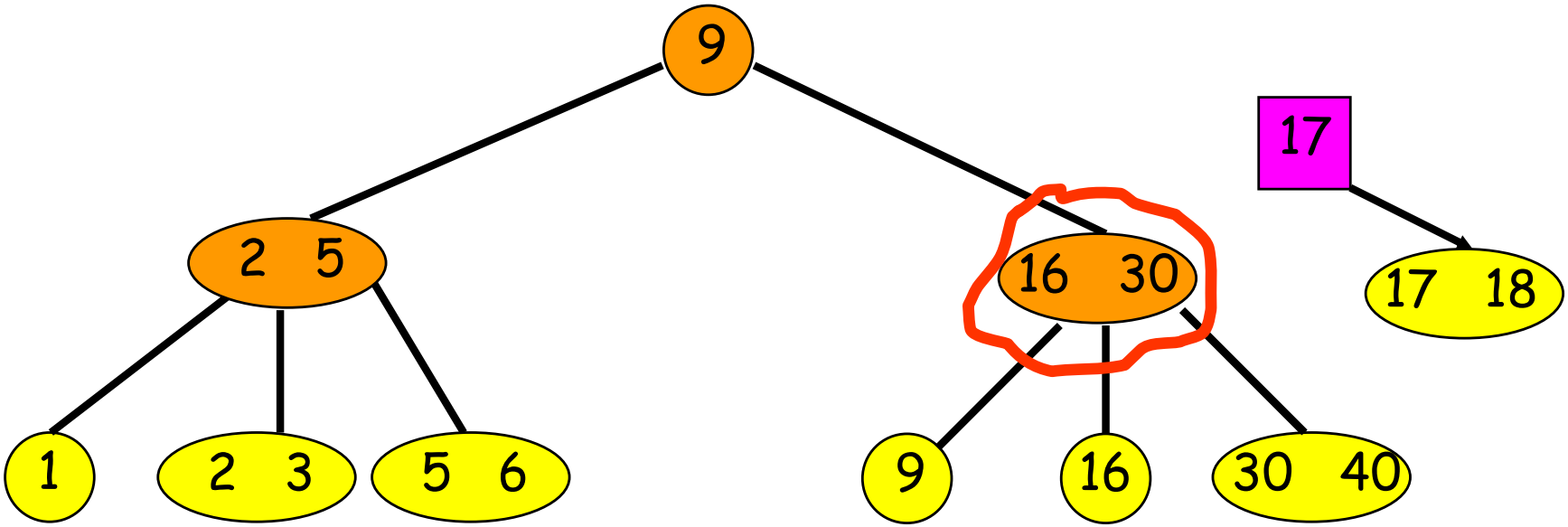

- Insert an index entry 2 plus a pointer into parent.

# Insert



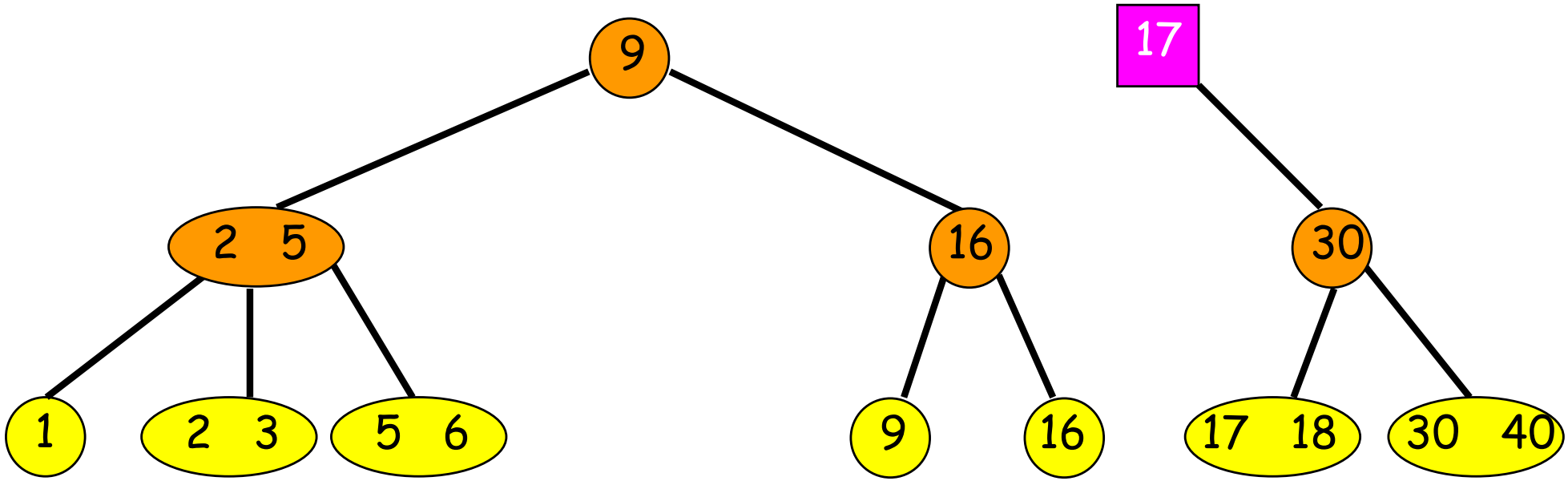

- Now, insert a pair with key = 18.

# Insert



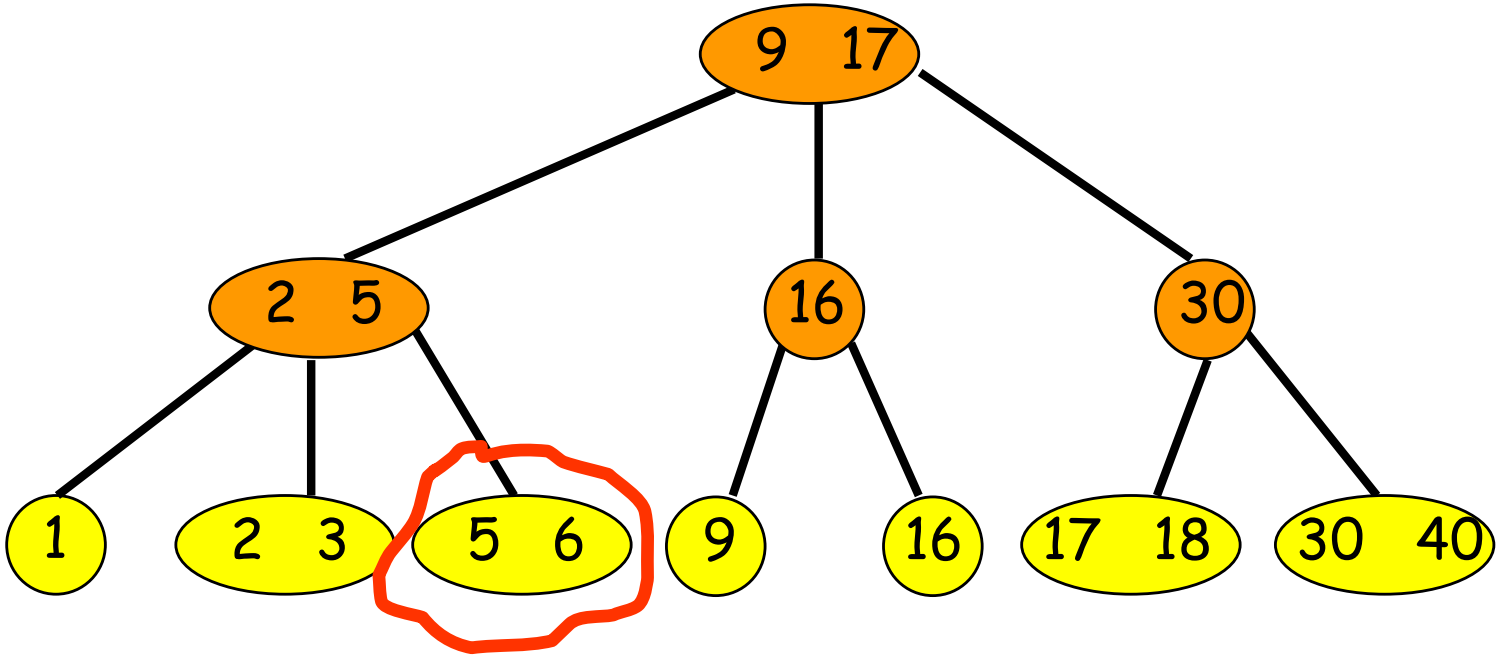

- Now, insert a pair with key = 18.
- Insert an index entry 17 plus a pointer into parent.

# Insert




- Now, insert a pair with key = 18.
- Insert an index entry 17 plus a pointer into parent.

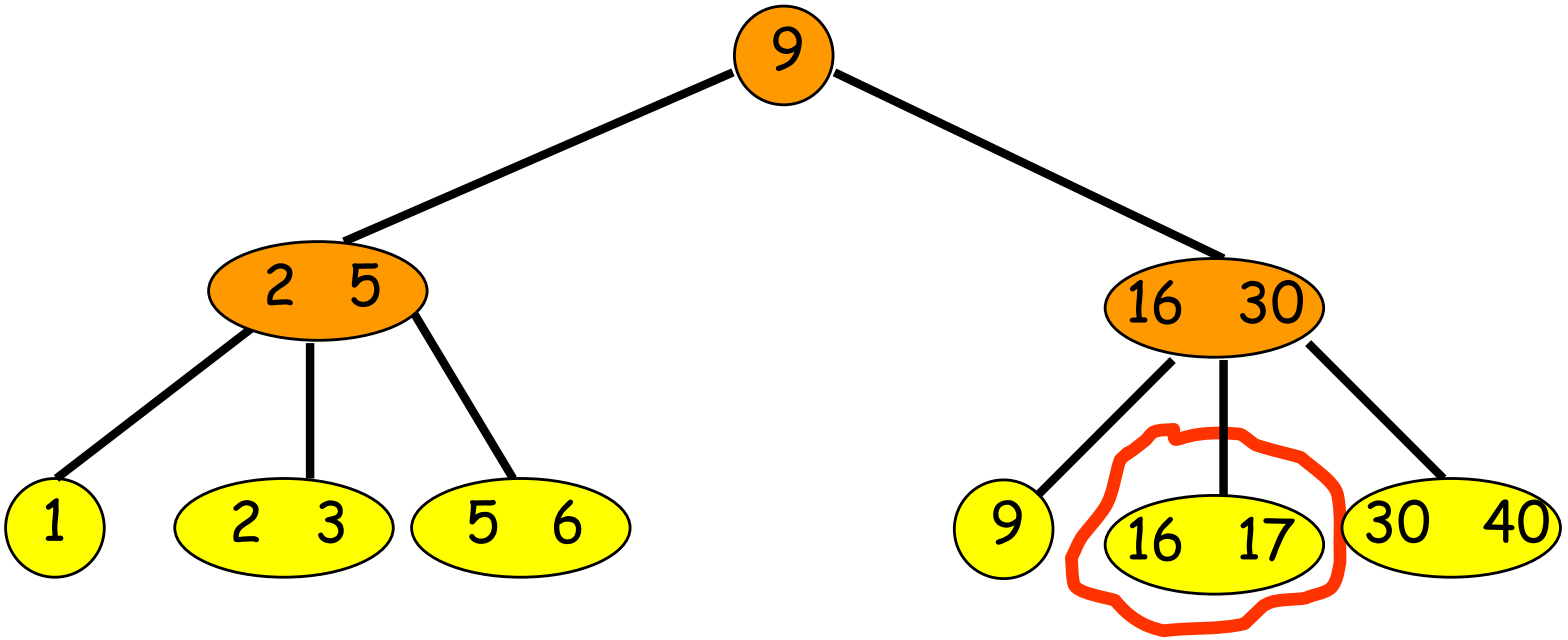
# Insert




- Now, insert a pair with key = 7.

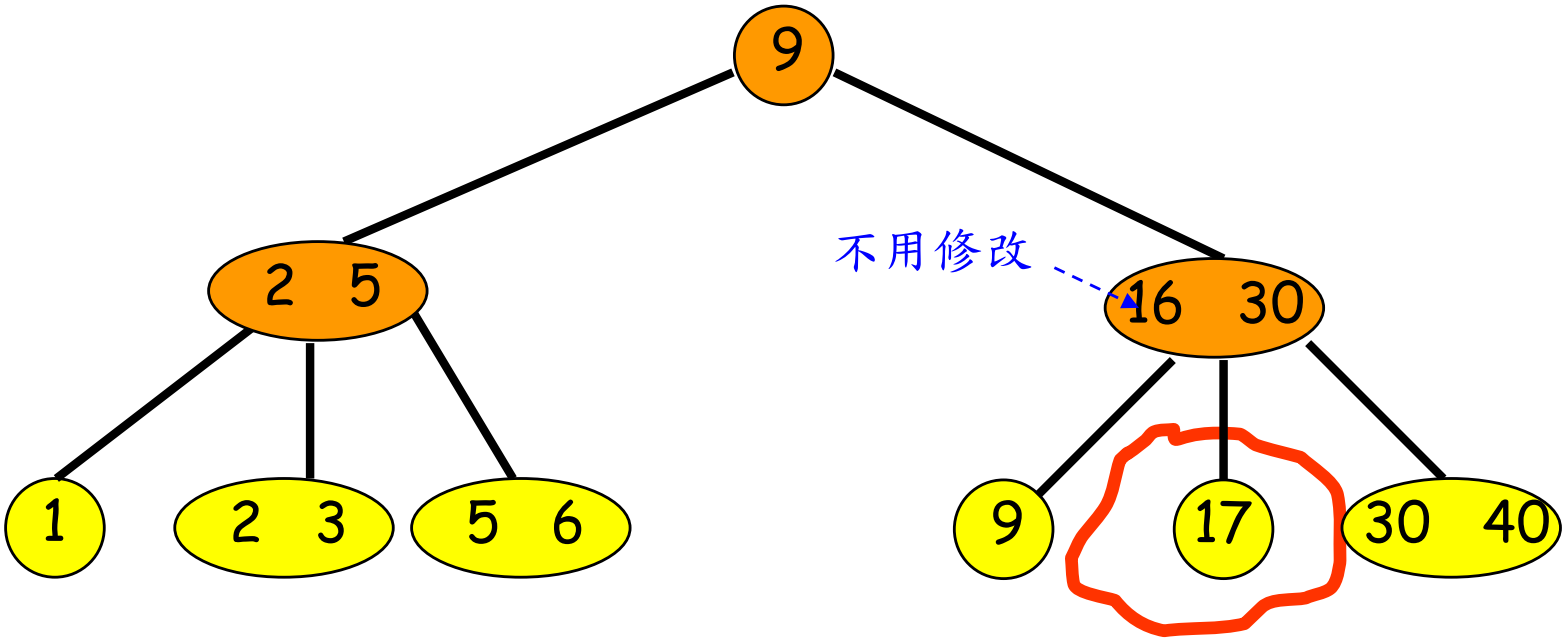
- 到目前為止，橘色節點的資料乃有部分insert的key-value pairs的keys所構成

# Delete




- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

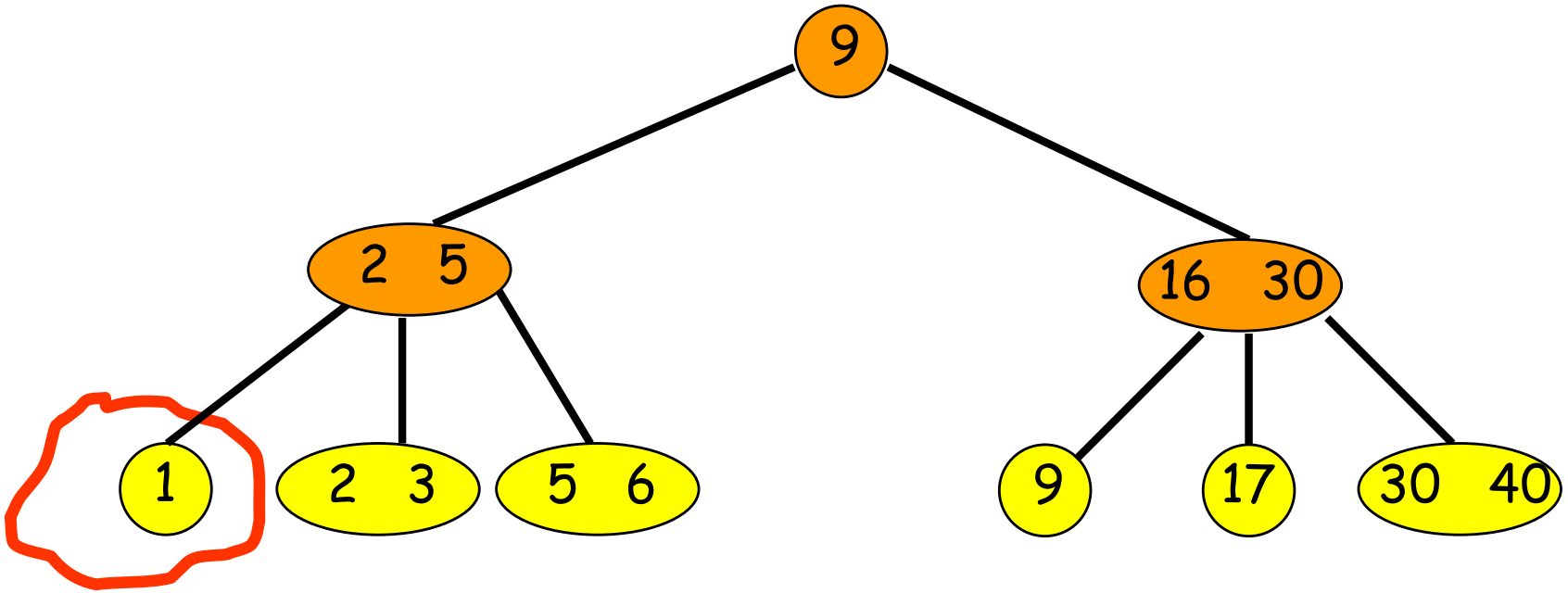
# Delete




- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

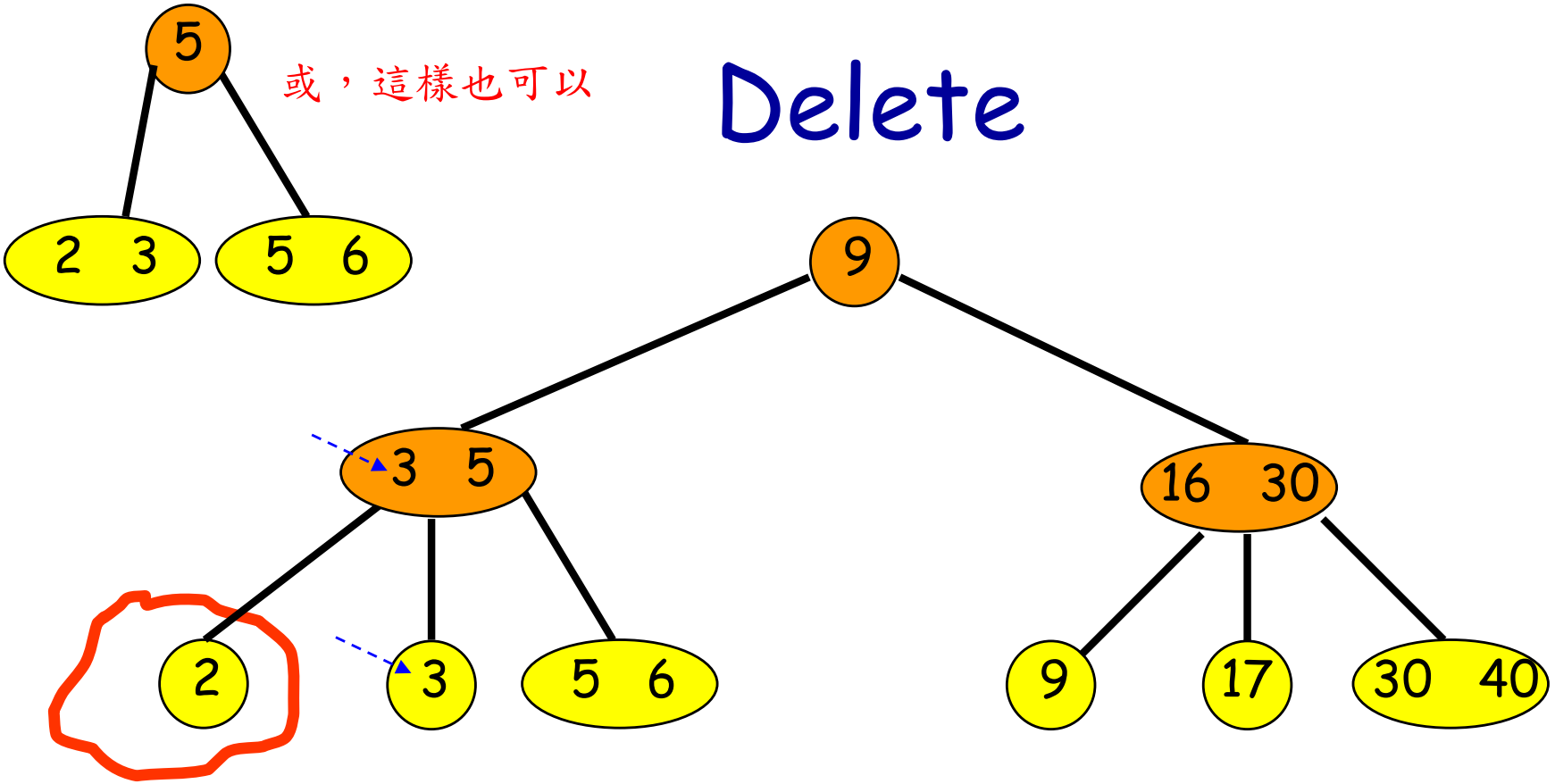
- 此時橘色節點內的keys值不完全是key-value pairs (in data pages) 裡的keys值

# Delete



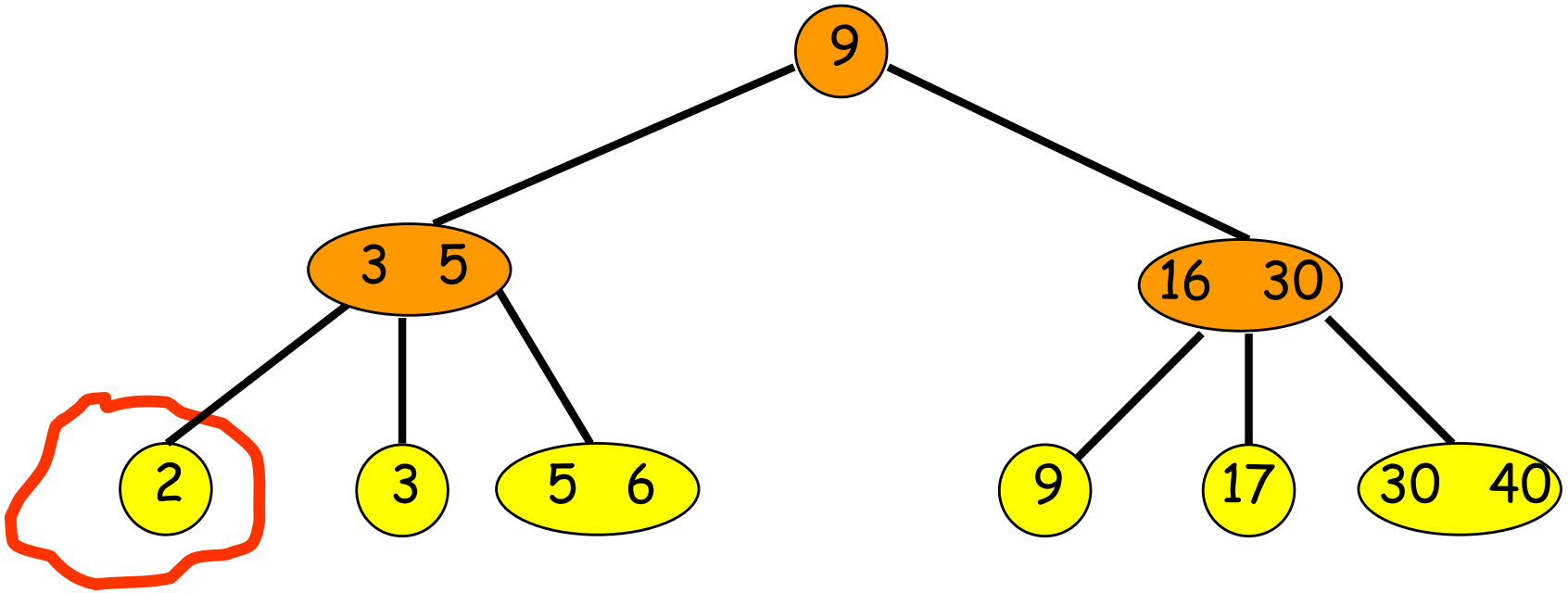

- Delete pair with key = 1.
- Get >= 1 from sibling and update parent key.

# Delete



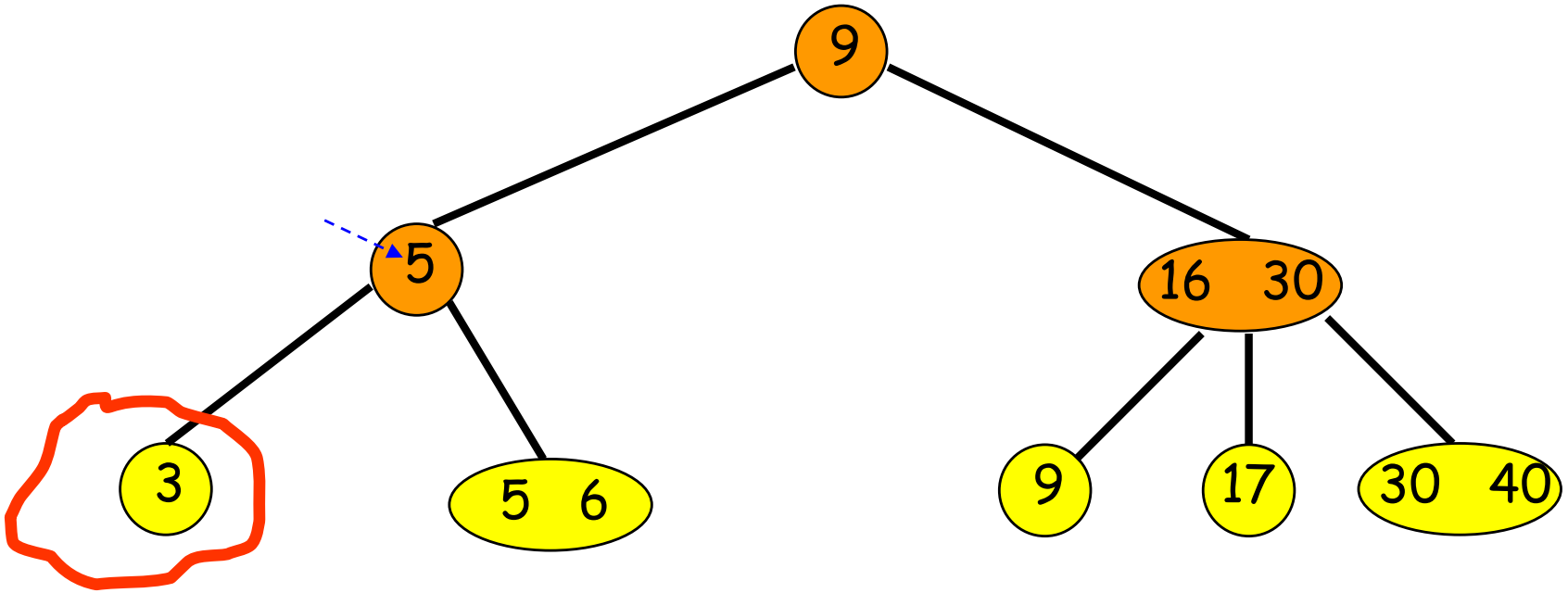

- Delete pair with key = 1.
- Get >= 1 from sibling and update parent key.

# Delete



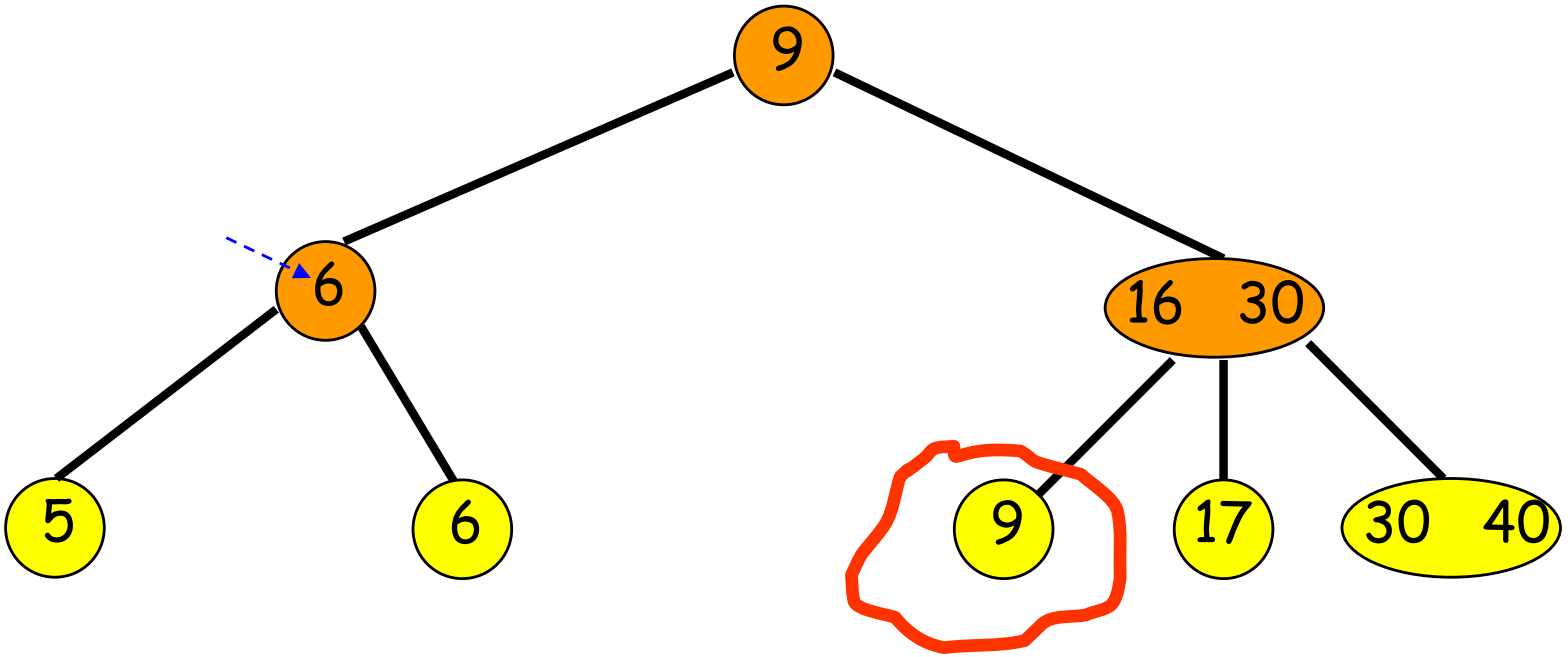

- Delete pair with key = 2.
- Merge with sibling, delete in-between key in parent.

# Delete



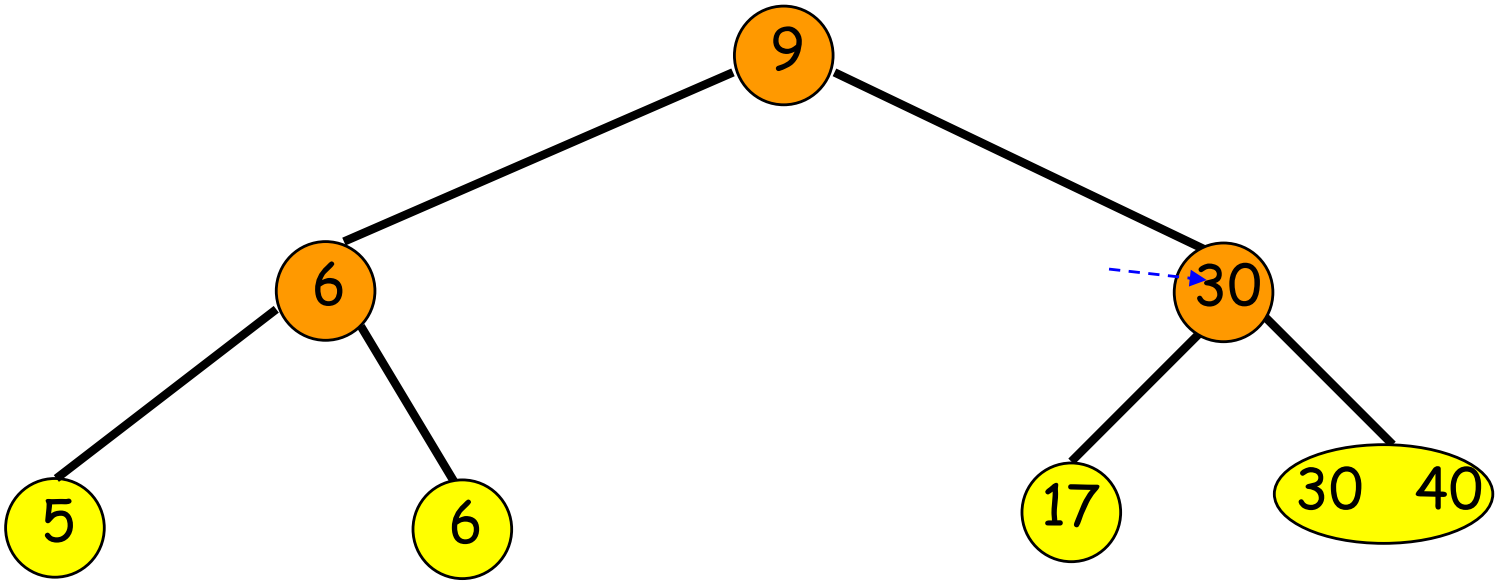

- Delete pair with key = 3.
- Get >= 1 from sibling and update parent key.

# Delete

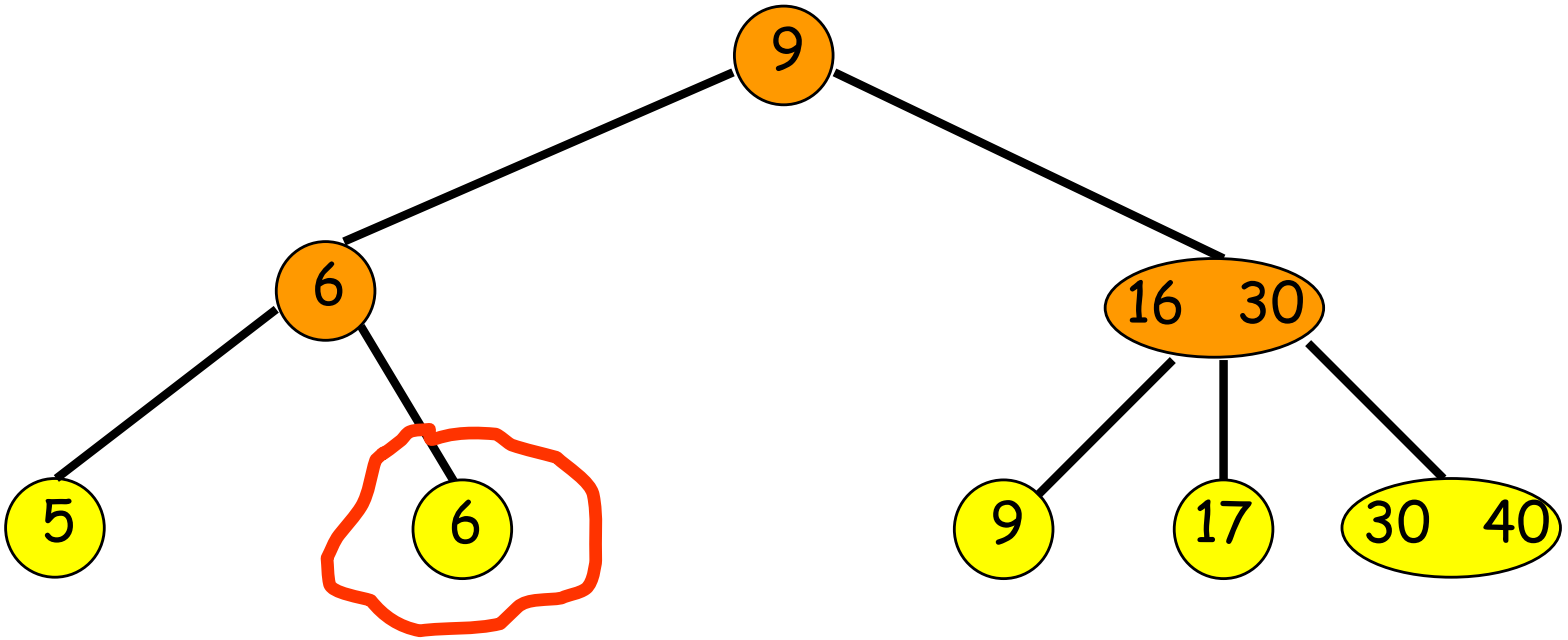



- Delete pair with key = 9.
- Merge with sibling, delete in-between key in parent.

# Delete

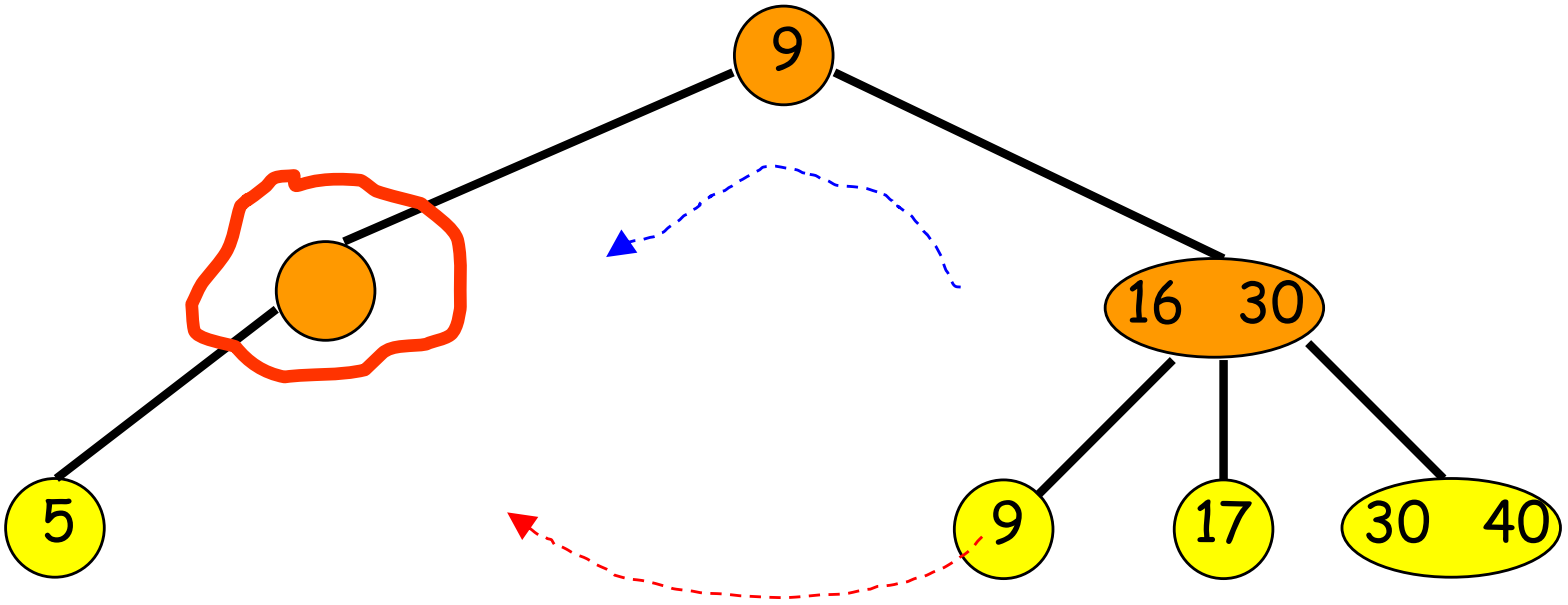


# Delete



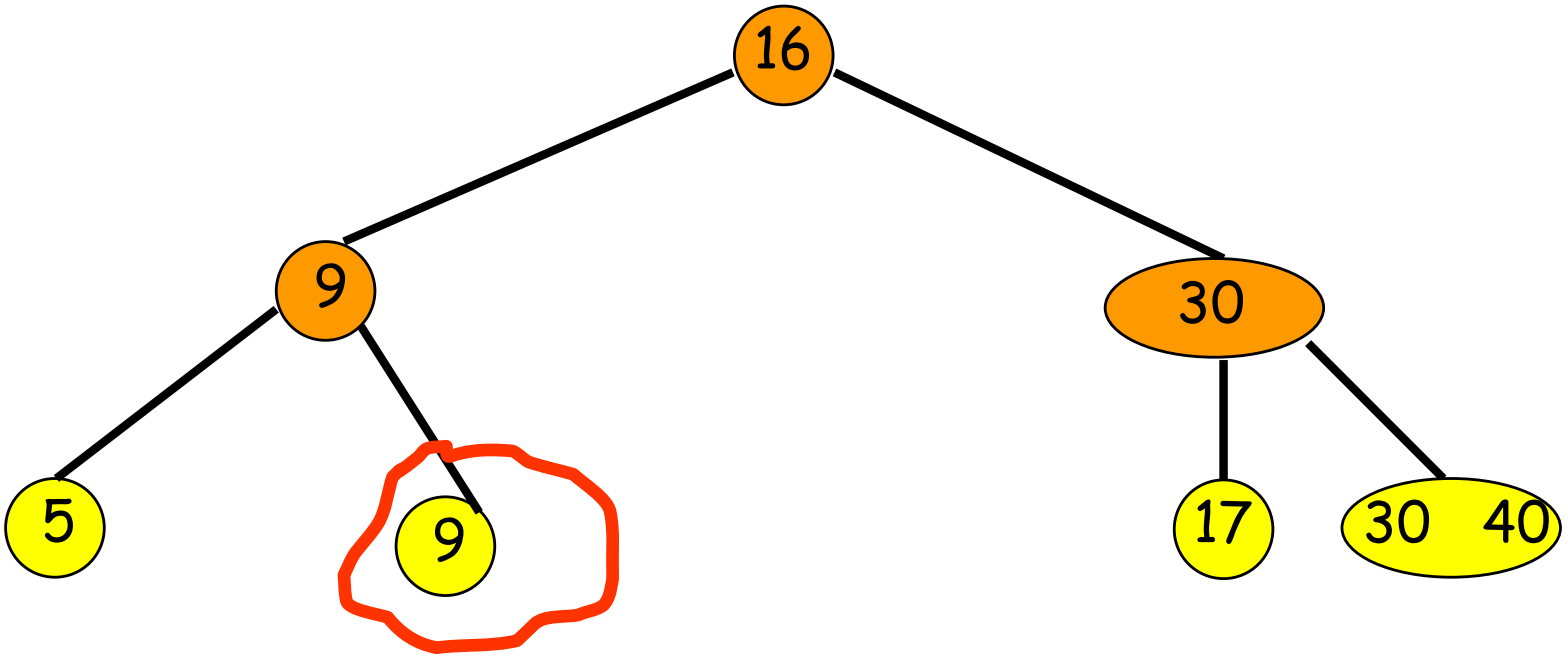

- Delete pair with key = 6.
- Merge with sibling, delete in-between key in parent.

# Delete



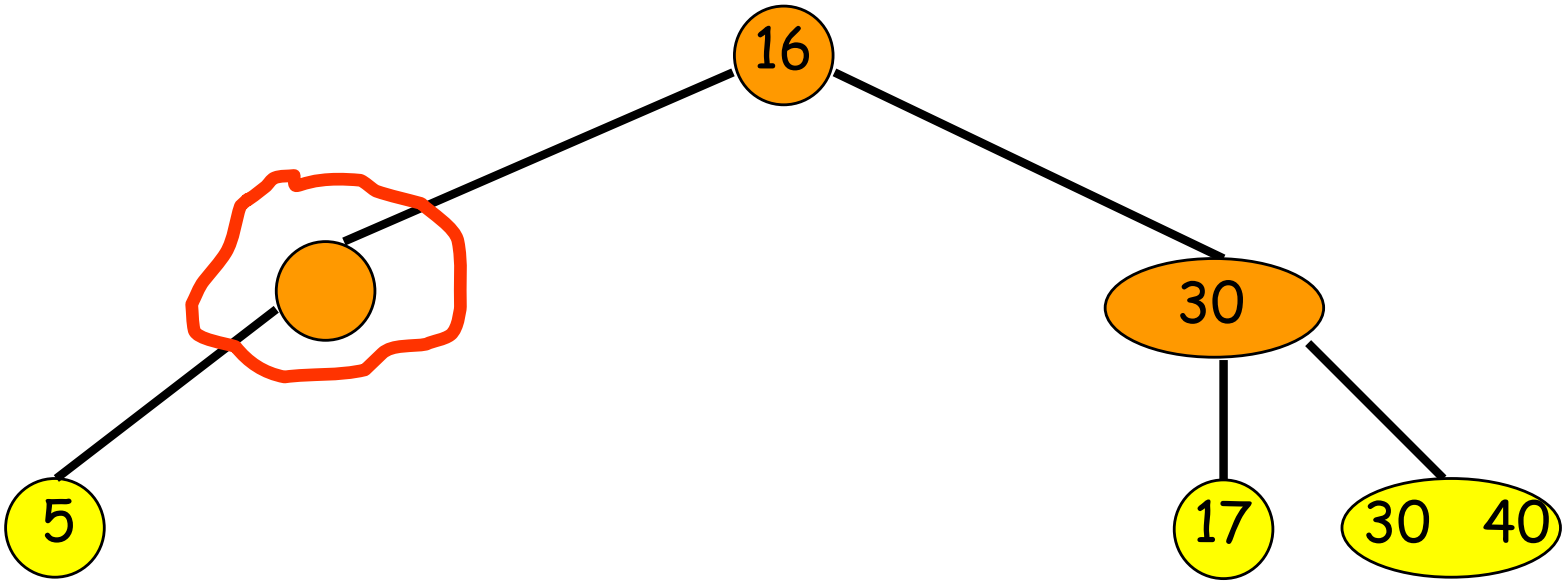

- Index node becomes deficient.
- Get >= 1 from sibling, move last one to parent, get parent key.

# Delete



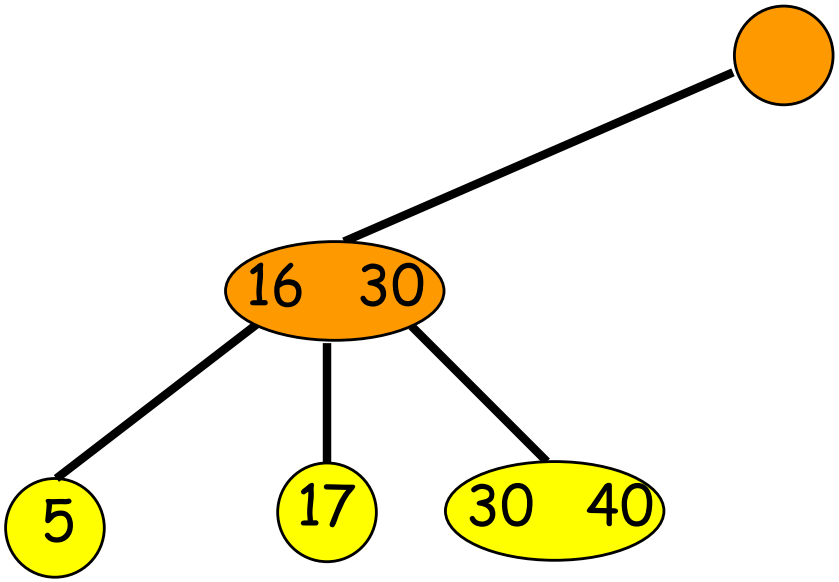

- Delete 9.
- Merge with sibling, delete in-between key in parent.

# Delete




- Index node becomes deficient.
- Merge with sibling and in-between key in parent.

# Delete




- Index node becomes deficient.
- It's the root; discard.