2-3 Trees
(Multiway or K-way Search Trees)

- 我們談2-3 trees
- 可推廣到2-3-4 trees
- Red-Black trees基於2-3-4 trees

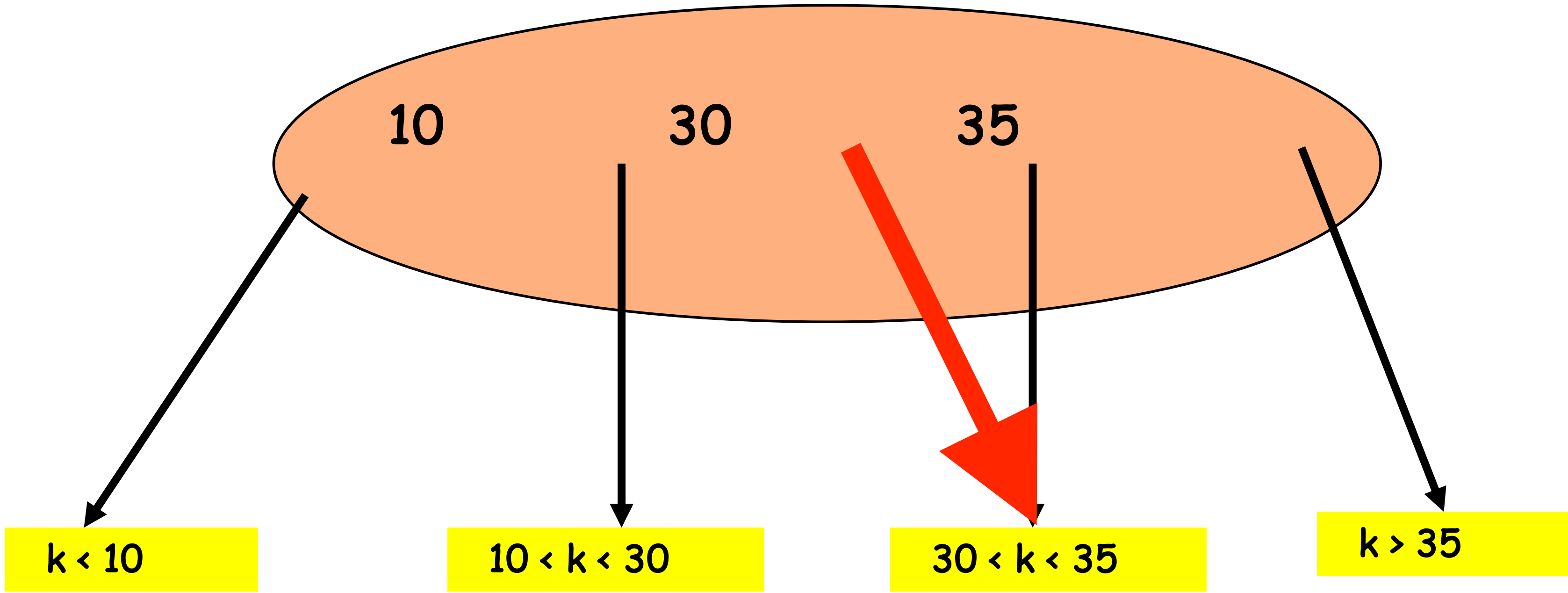
- …
- **2-3 tree…**we are here
- Red-Black…next
- B+-tree…
- …

Red-Black是2-3 tree的變形 (i.e., 2-3-4 tree essentially)

M-way Search Trees

- M-way search trees
  - Search trees, essentially
    - Recall: binary search trees
  - Each node has up to **m - 1** pairs and **m** children
    - m = 2 => binary search tree

4-way (Degree=4) Search Tree



Maximum # of Data Pairs

- Happens when all internal nodes are m-nodes.
- Full degree m tree.
- # of nodes $= 1 + m + m^2 + m^3 + \ldots + m^{h-1}$
  $= (m^h - 1)/(m - 1)$.
- Each node has $m - 1$ data pairs.
- So, # of data pairs $= m^h - 1$.

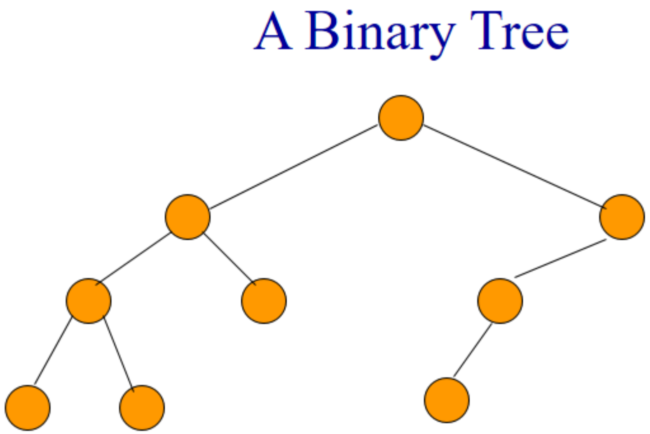
Capacity of m-Way Search Tree

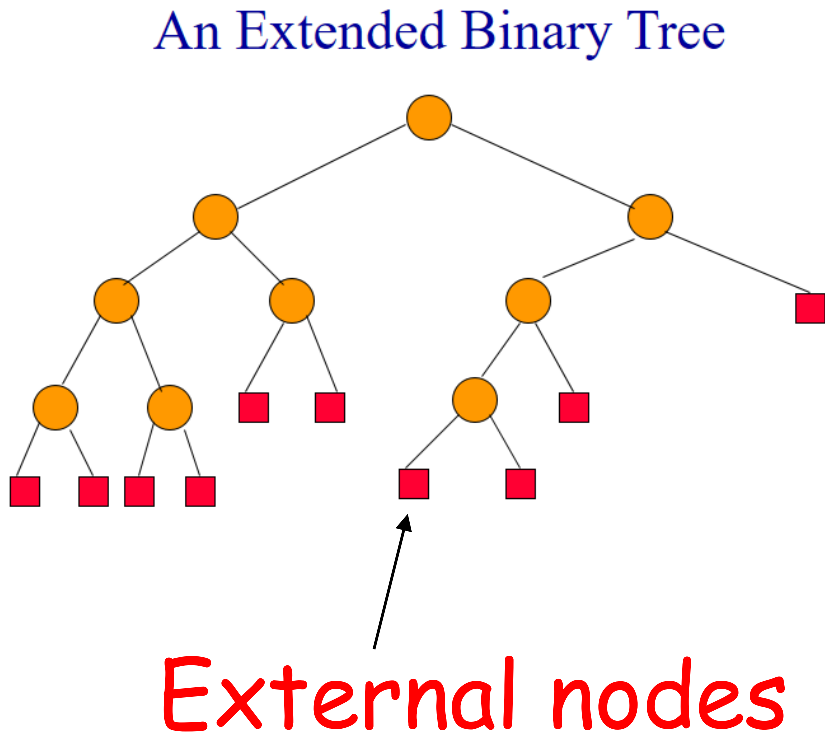
	m = 2	m = 200
h = 3	7	$8 * 10^6 - 1$
h = 5	31	$3.2 * 10^{11} - 1$
h = 7	127	$1.28 * 10^{16} - 1$

Definition of B-Tree

- Definition assumes external nodes (extended m-way search tree).
- B-tree of order m.
  - m-way search tree.
  - Not empty => root has at least 2 children.
  - Remaining "internal nodes" (if any) have at least ceil(m/2) children. //希望每個內部節點有效運用空間使至少裝滿近乎一半的指標/資料
  - "External nodes" on same level. //希望樹夠扁平
- B denotes "Balanced"

Extended Binary Trees





2-3 and 2-3-4 Trees

- B-tree of order m. //見前述
  - m-way search tree.
  - Not empty => root has at least 2 children.
  - Remaining internal nodes (if any) have at least ceil(m/2) children.
  - External nodes on same level.

degree of 2

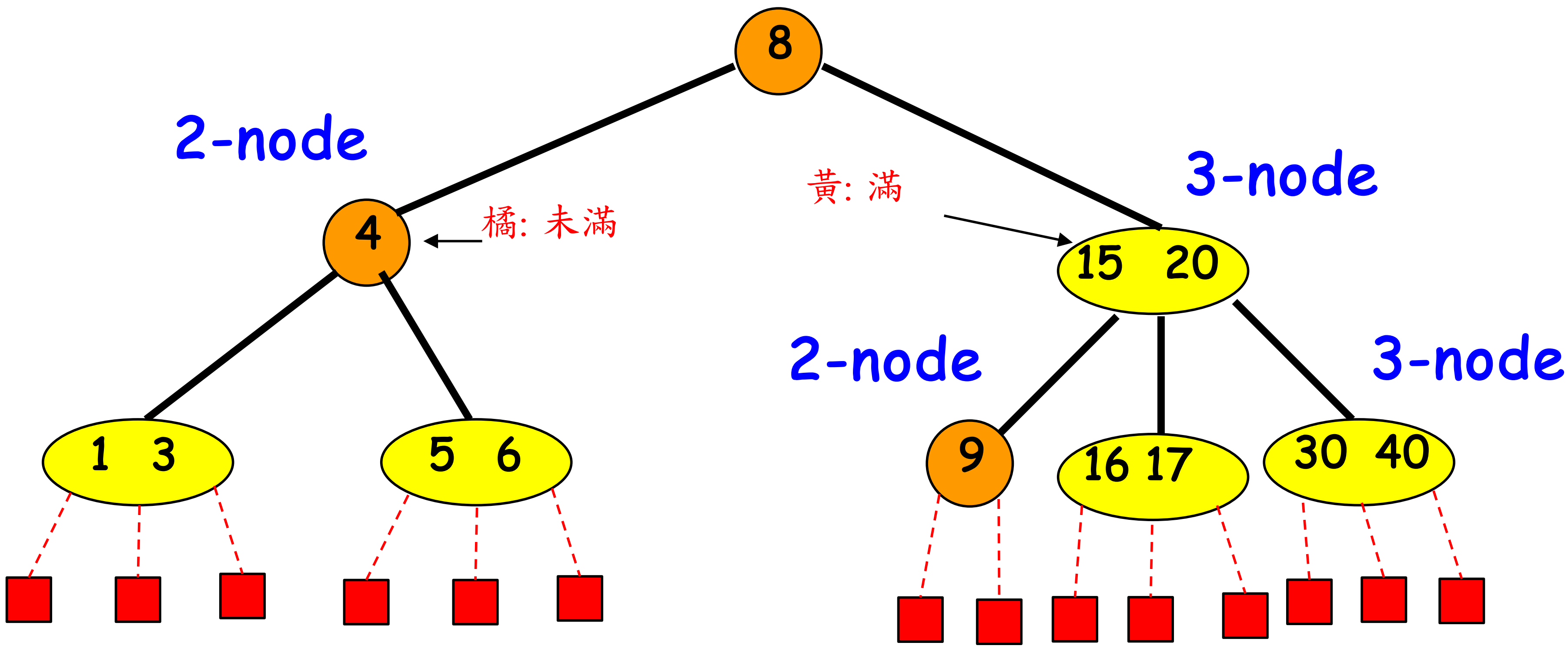
degree of 3

- 2-3 tree is B-tree of order 3. //2-node+3-node, m=3
- 2-3-4 tree is B-tree of order 4. //2-node+3-node+4-node, m=4

B-Trees of Order 5 and 2 (自行檢驗)

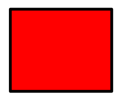
- B-tree of order 5 is 3-4-5 tree (root may be 2-node though).
- B-tree of order 2 is full binary tree.
  - Because of external nodes on the same level

Insert (2-3 tree)



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

//往後我們就不會標出external nodes



Split an Overfull Node

$m\ a_0\ p_1\ a_1\ p_2\ a_2\ \ldots\ p_m\ a_m$ ← 從m-1變成m筆資料
(從m-way變成(m+1)-way)

- a_i is a pointer to a subtree.
- p_i is a dictionary pair. //資料，即，(key, value) pair
- m 筆資料 //目前存在節點內的資料筆數

Split

原來的節點內部變更為
(滿足節點至少有ceil(m/2) degree)

$\mathrm{ceil}(m/2)-1\ a_0\ p_1\ a_1\ p_2\ a_2\ \ldots\ p_{\mathrm{ceil}(m/2)-1}\ a_{\mathrm{ceil}(m/2)-1}$

"新"的節點
(滿足節點至少有ceil(m/2) degree)

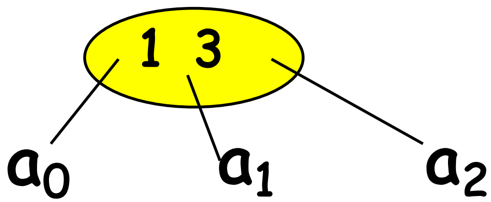
$m-\mathrm{ceil}(m/2)\ a_{\mathrm{ceil}(m/2)}\ p_{\mathrm{ceil}(m/2)+1}\ a_{\mathrm{ceil}(m/2)+1}\ \ldots\ p_m\ a_m$

單獨節點，含$p_{\mathrm{ceil}(m/2)}$ plus pointer to new node（藍）inserted in parent.

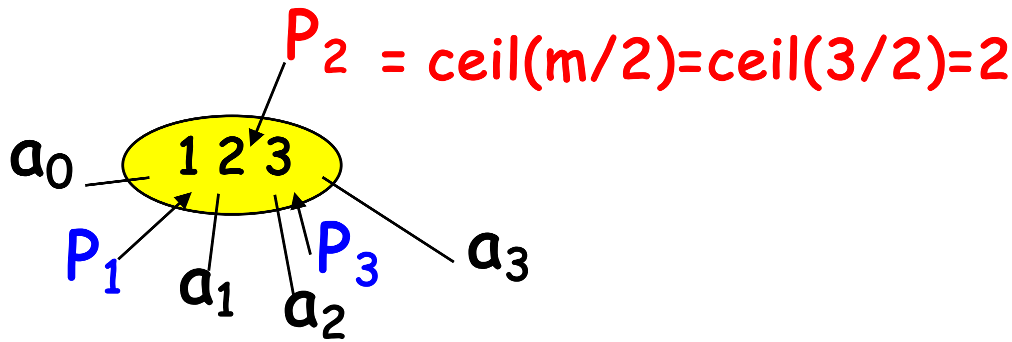
PS: 原來的a_i (i=0,1,2,…,m)不是在黃色就是在藍色節點

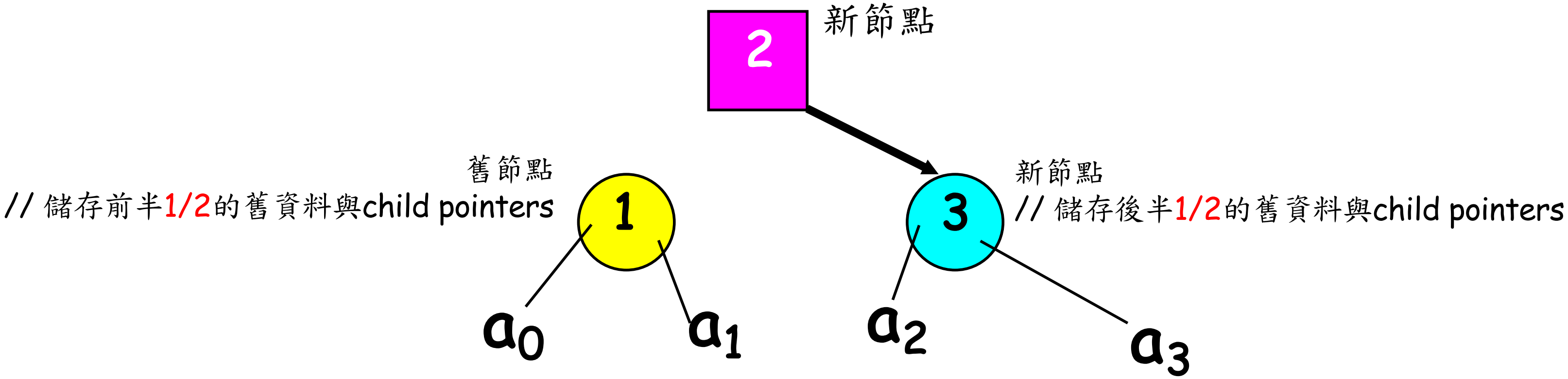
Example (2-3 Tree)

- 3-node

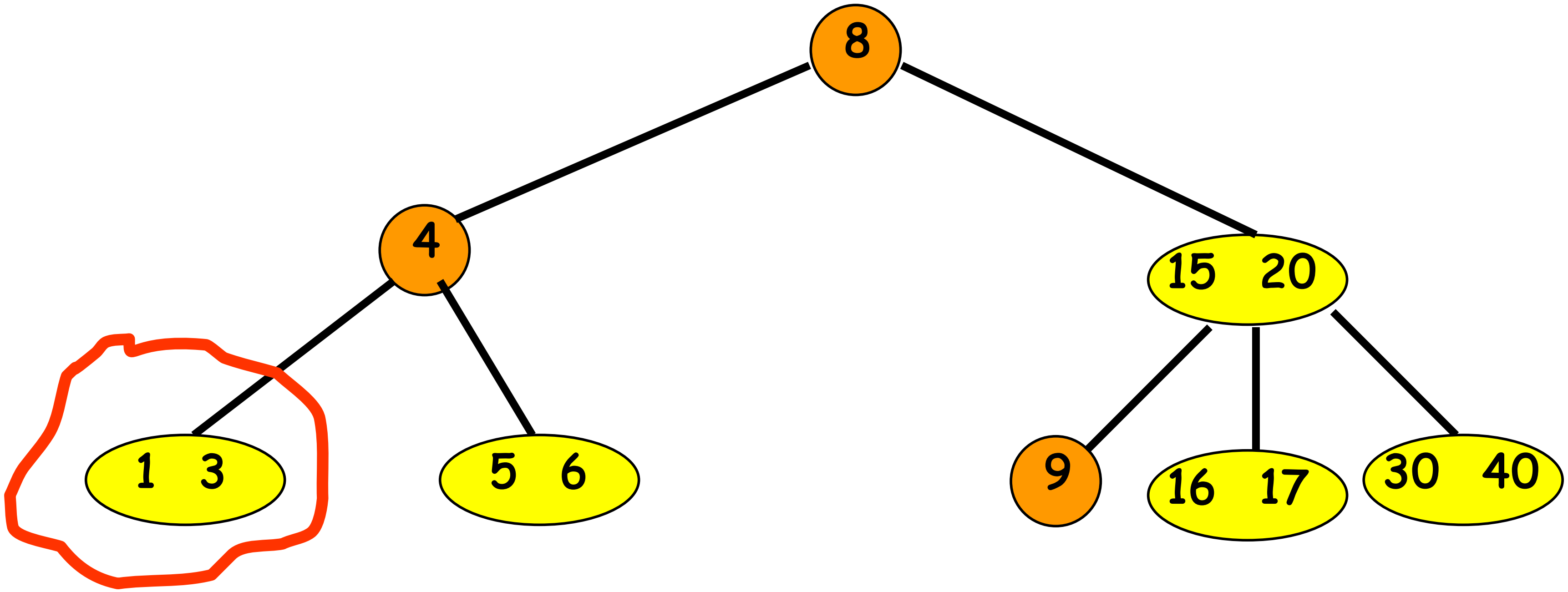


- 加了一筆資料 "2"
- Split





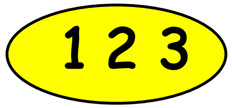
Insert (2-3 tree)



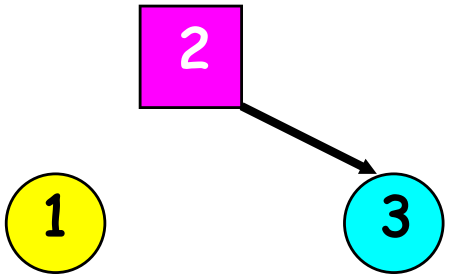
- Insert a pair with key = 2.
- New pair goes into a 3-node.

Insert into a Leaf 3-node

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

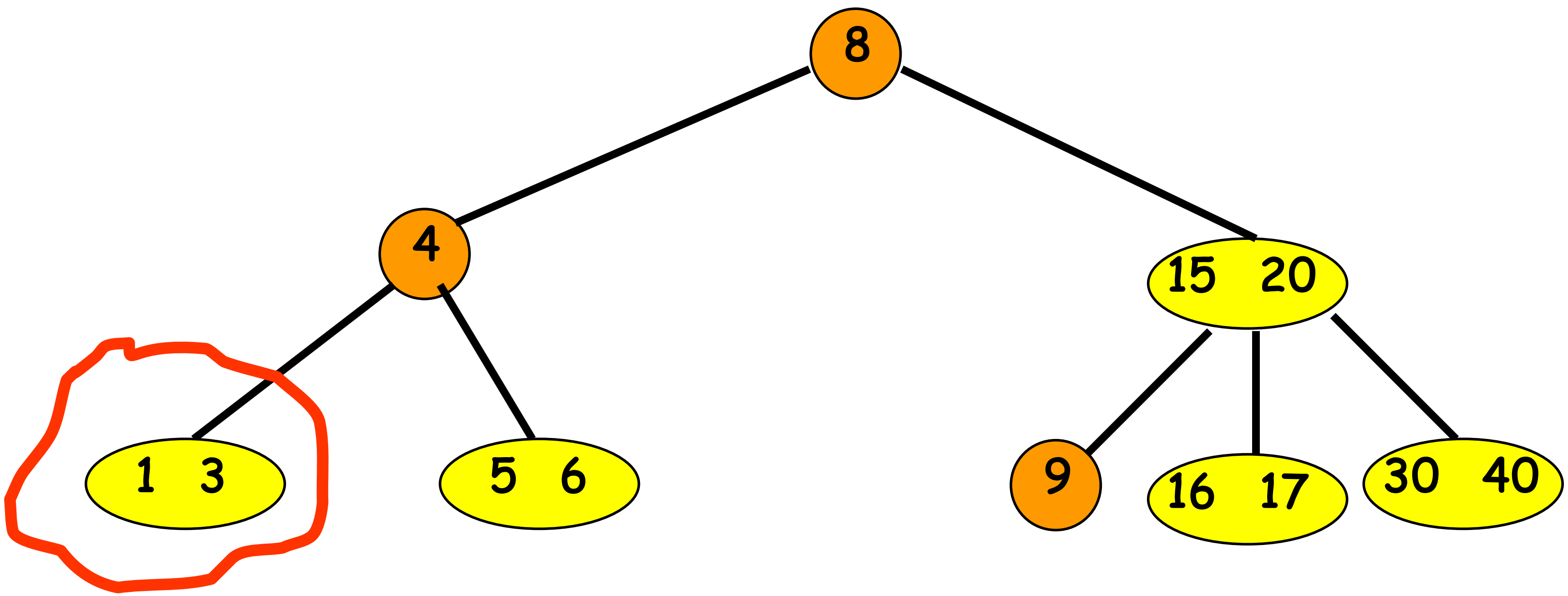


- Split overflowed node around middle key.



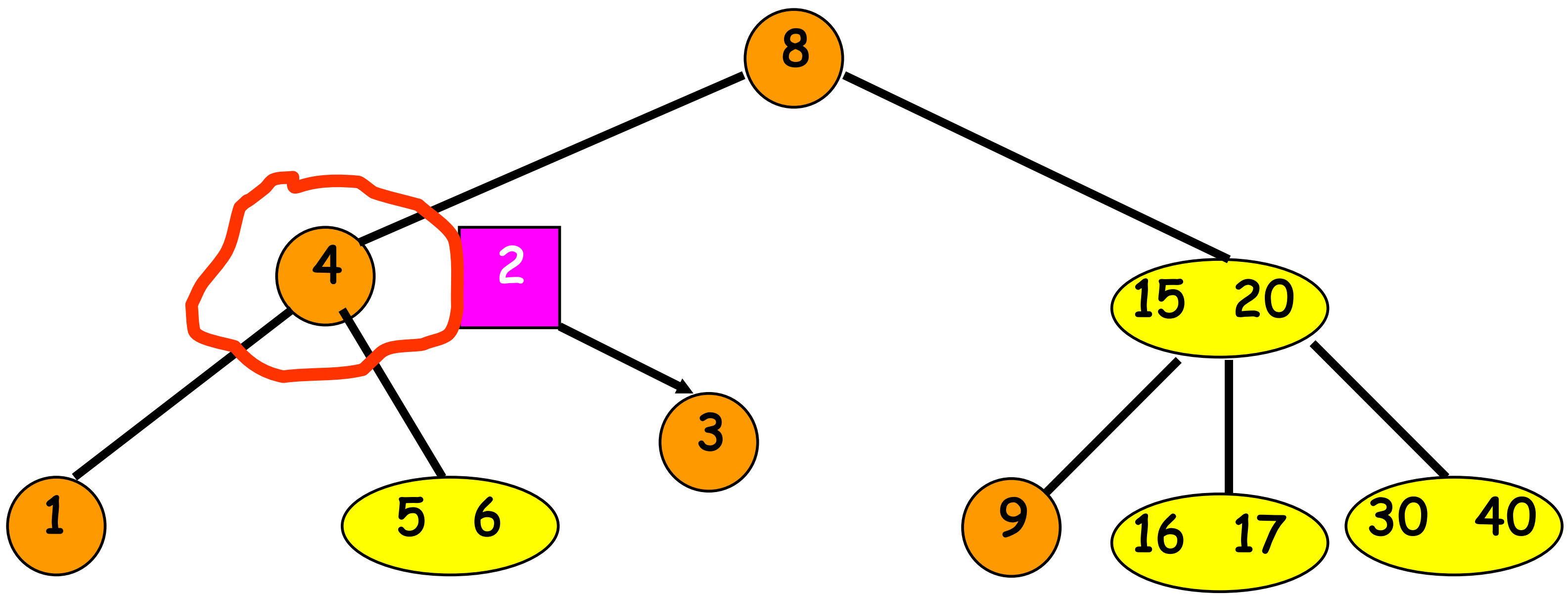
- Insert middle key and pointer to new node into parent.

Insert (2-3 tree)



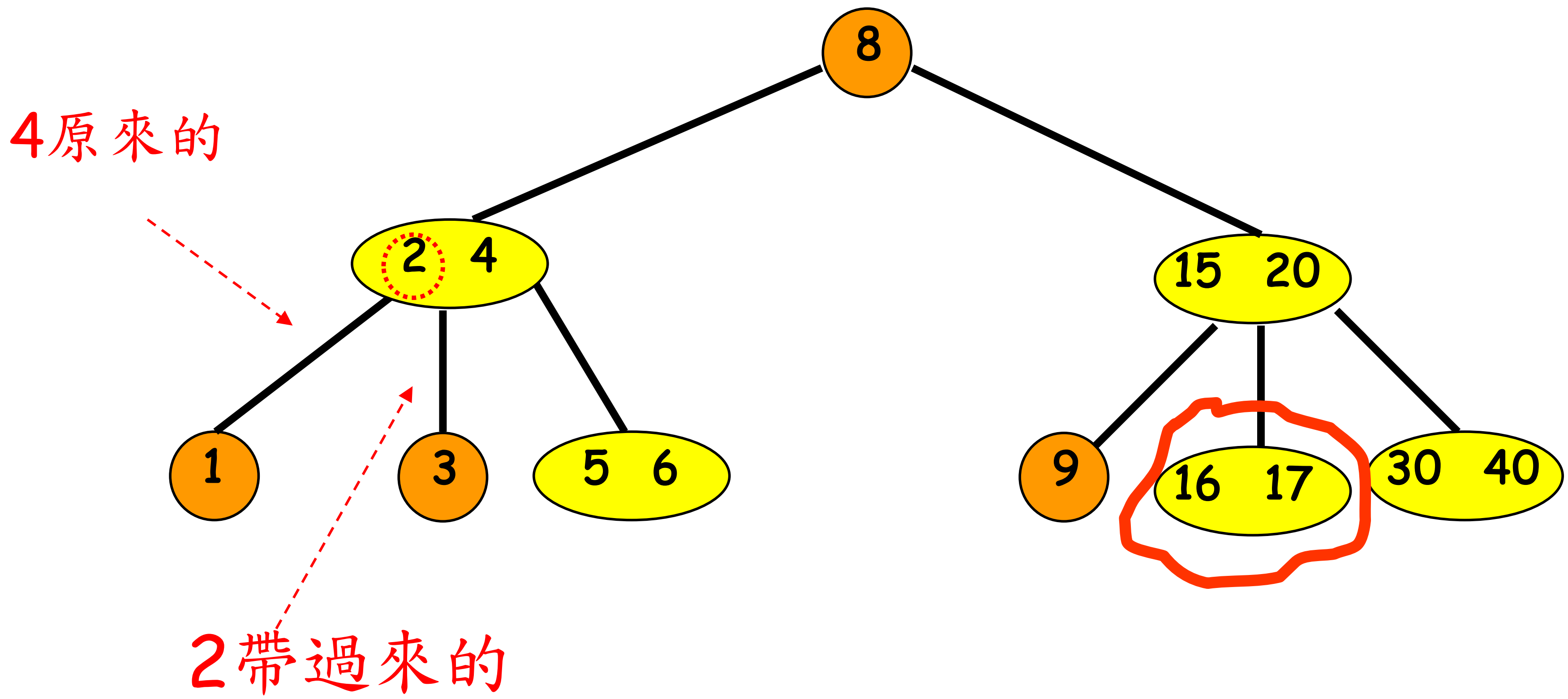
- Insert a pair with key = 2.

Insert (2-3 tree)



- Insert a pair with key = 2 plus a pointer into parent.

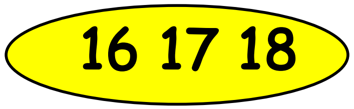
Insert (2-3 tree)



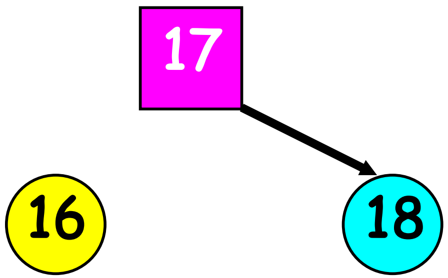
- Now, insert a pair with key = 18.

Insert into a Leaf 3-node

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

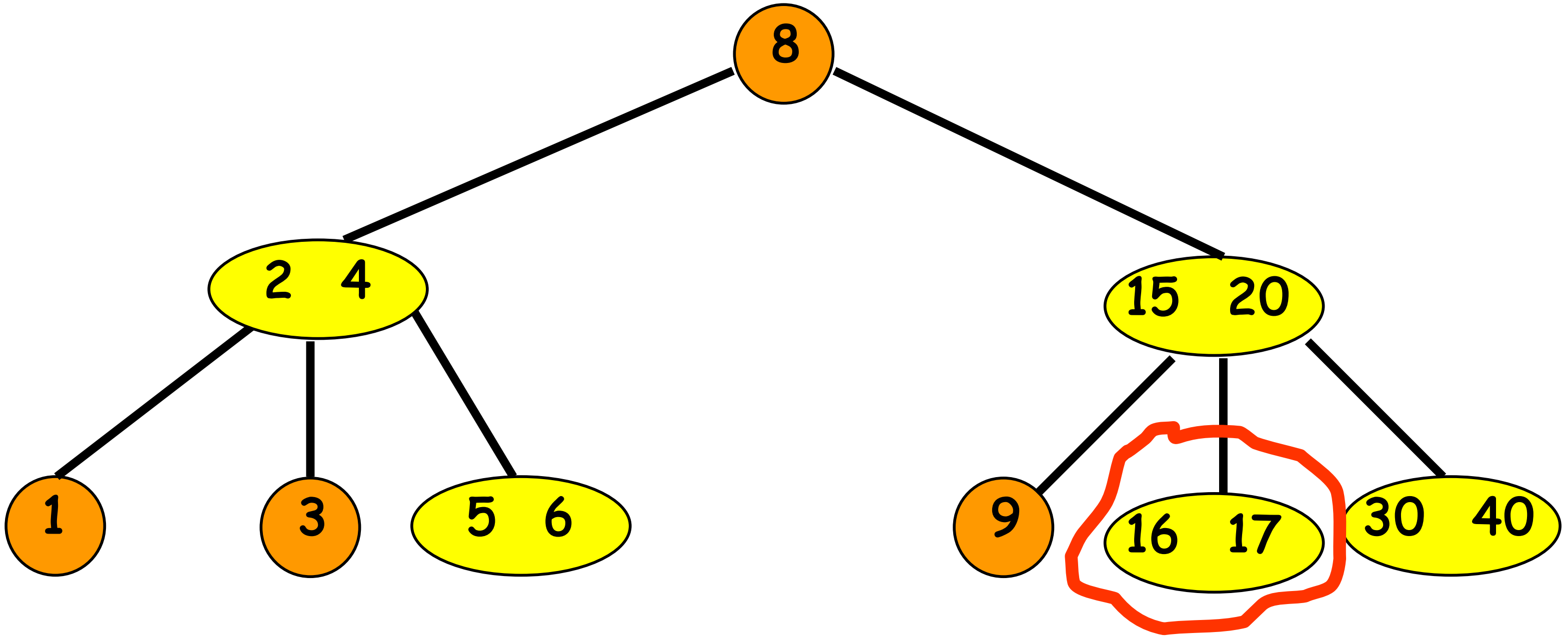


- Split the overflowed node.



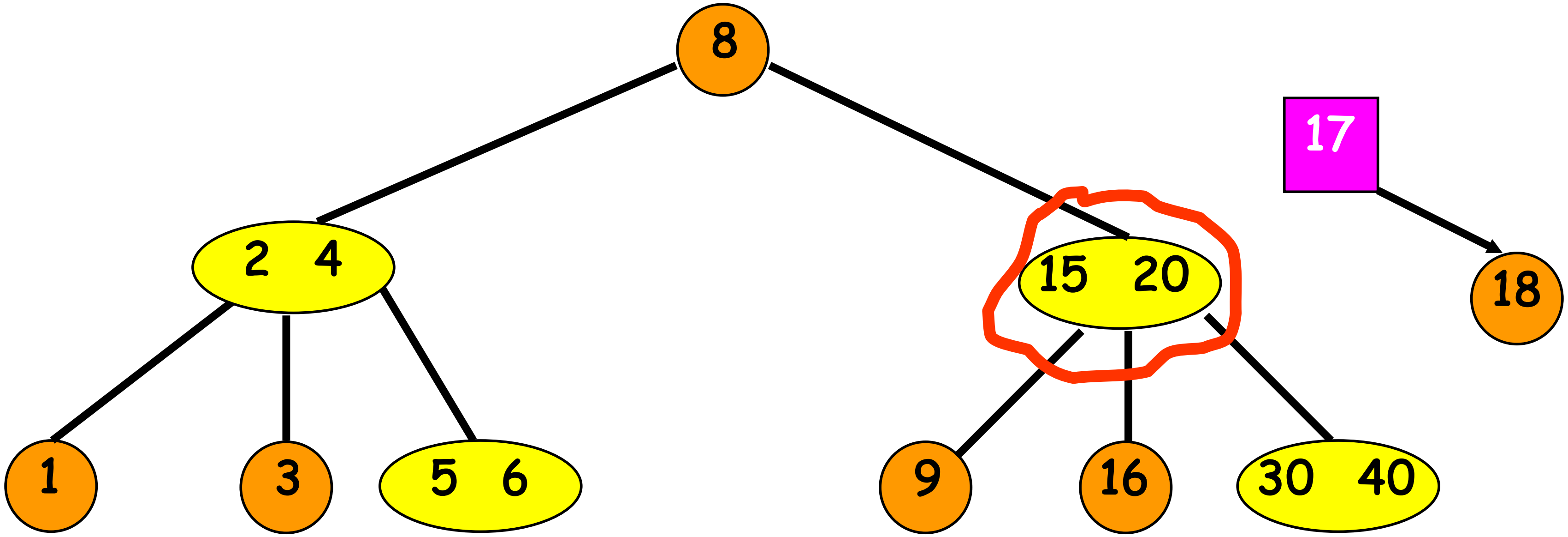
- Insert middle key and pointer to new node into parent.

Insert (2-3 tree)



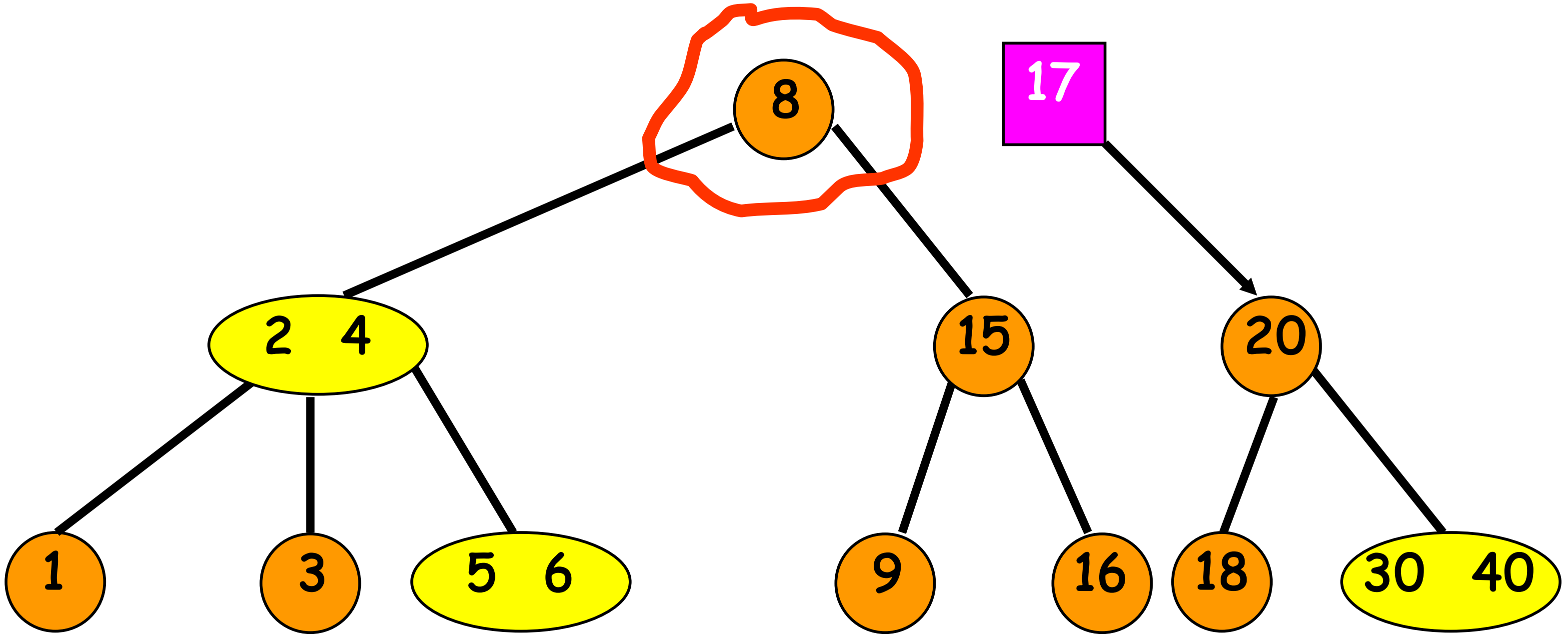
- Insert a pair with key = 18.

Insert (2-3 tree)



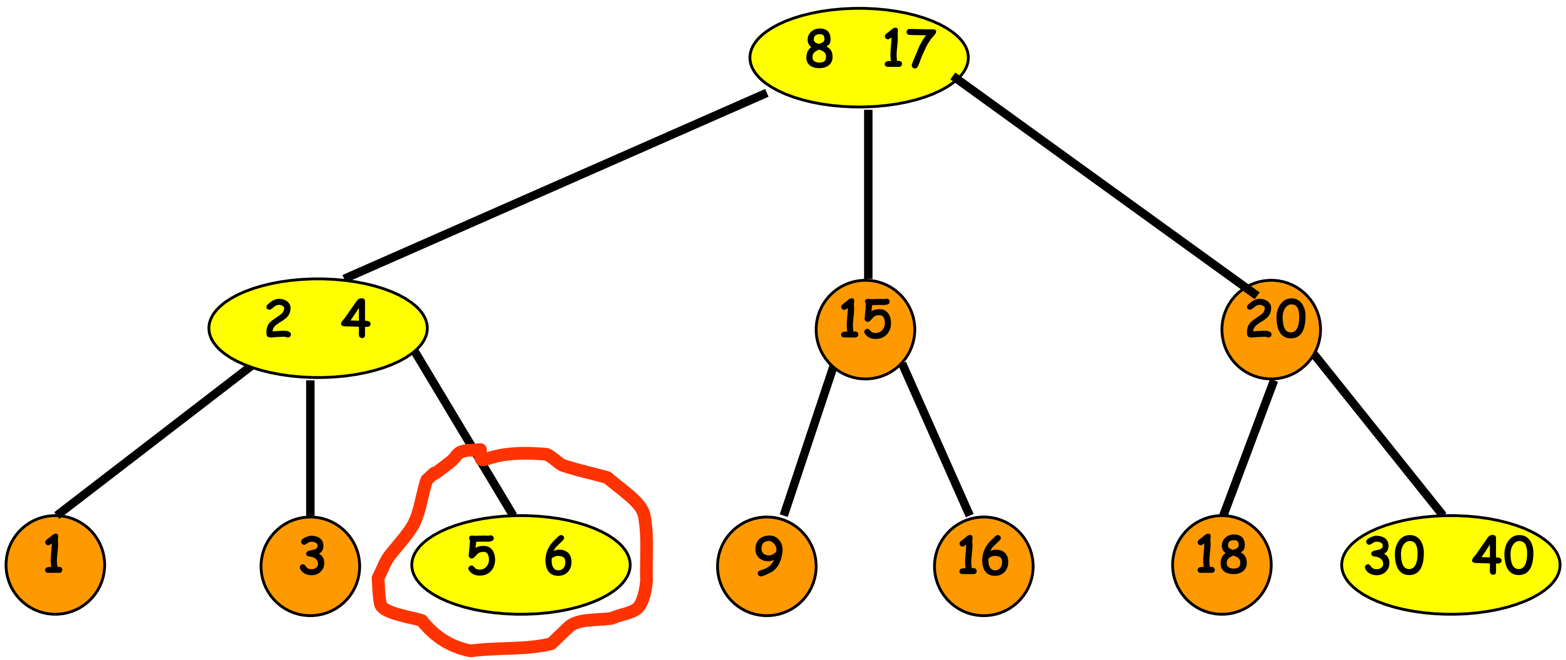
- Insert a pair with key = 17 plus a pointer into parent.

Insert (2-3 tree)



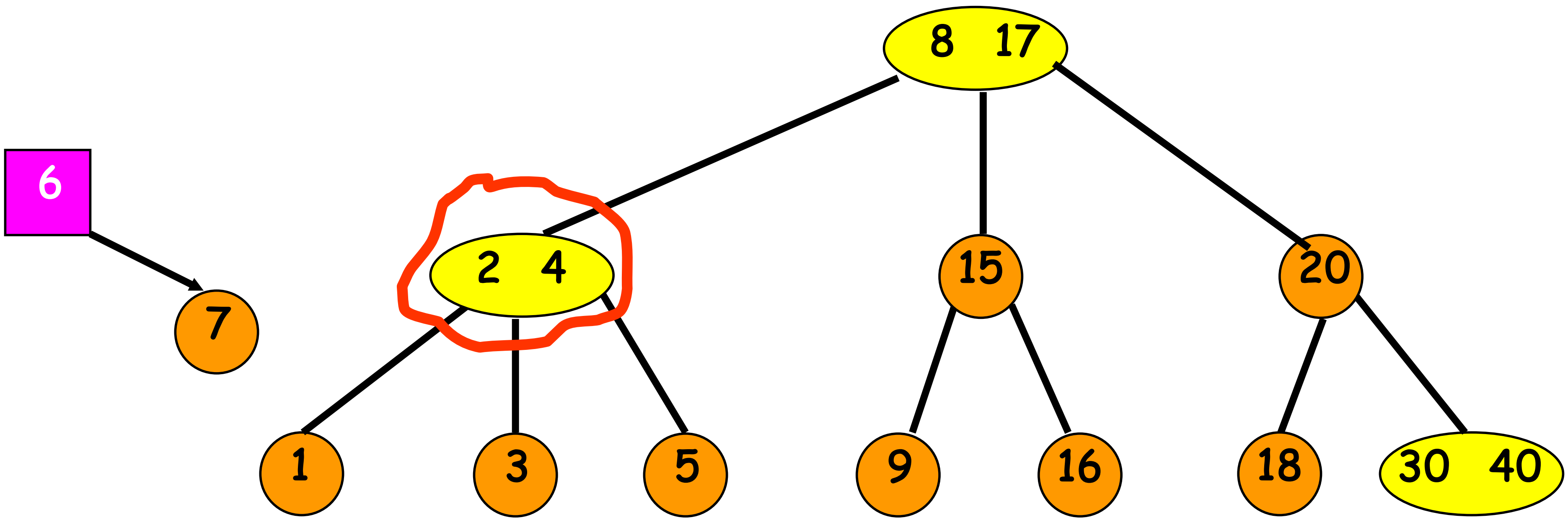
- Insert a pair with key = 17 plus a pointer into parent.

Insert (2-3 tree)



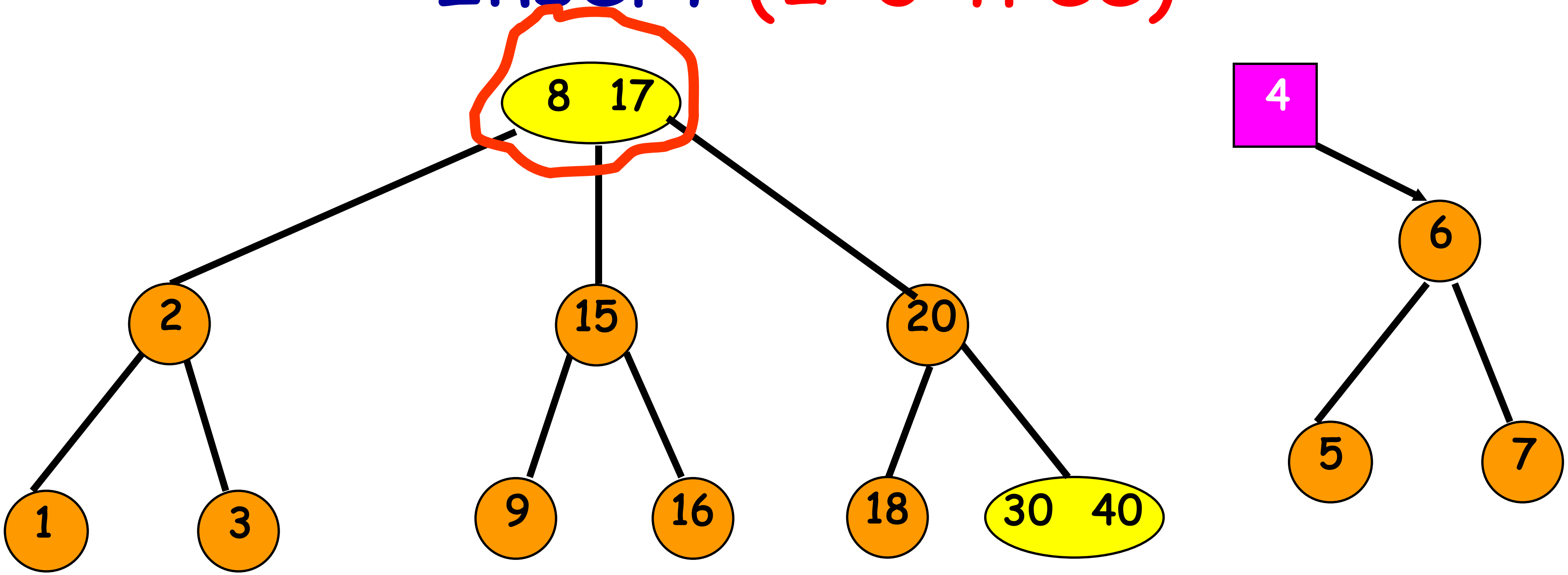
- Now, insert a pair with key = 7.

Insert (2-3 tree)



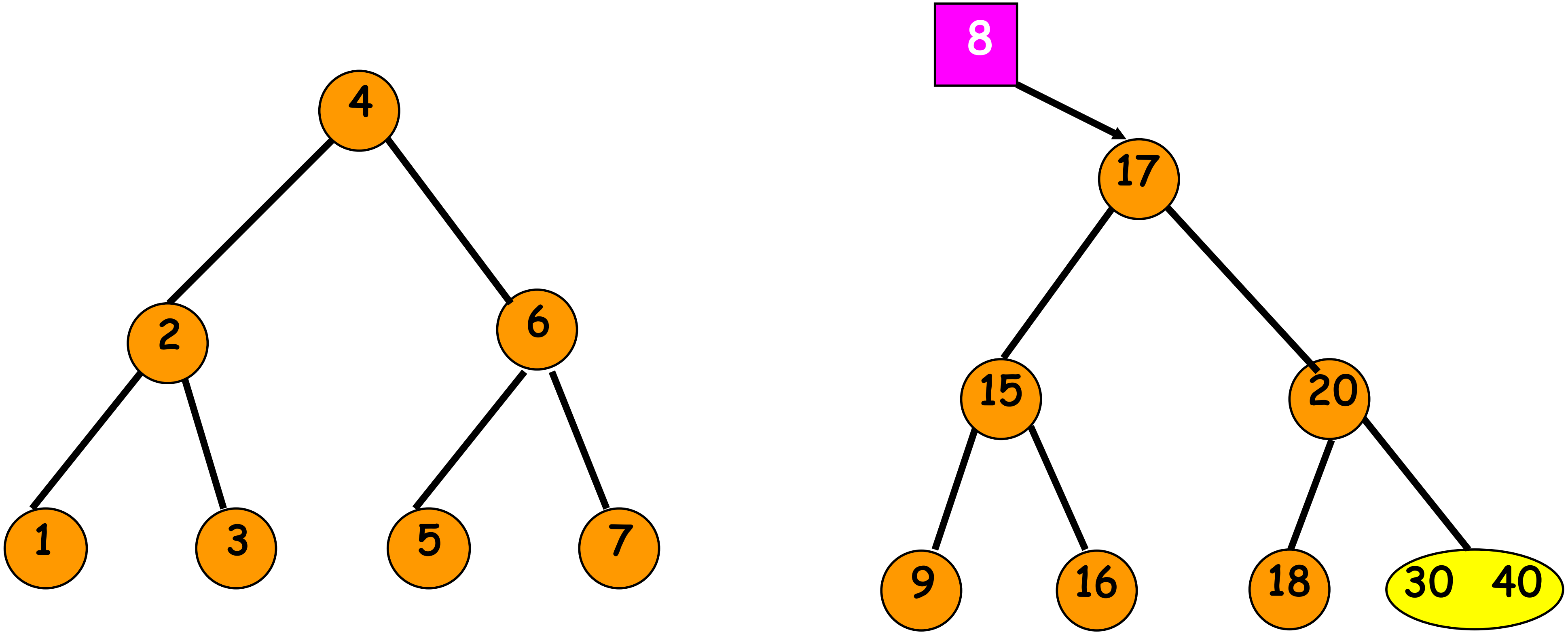
- Insert a pair with key = 6 plus a pointer into parent.

Insert (2-3 tree)



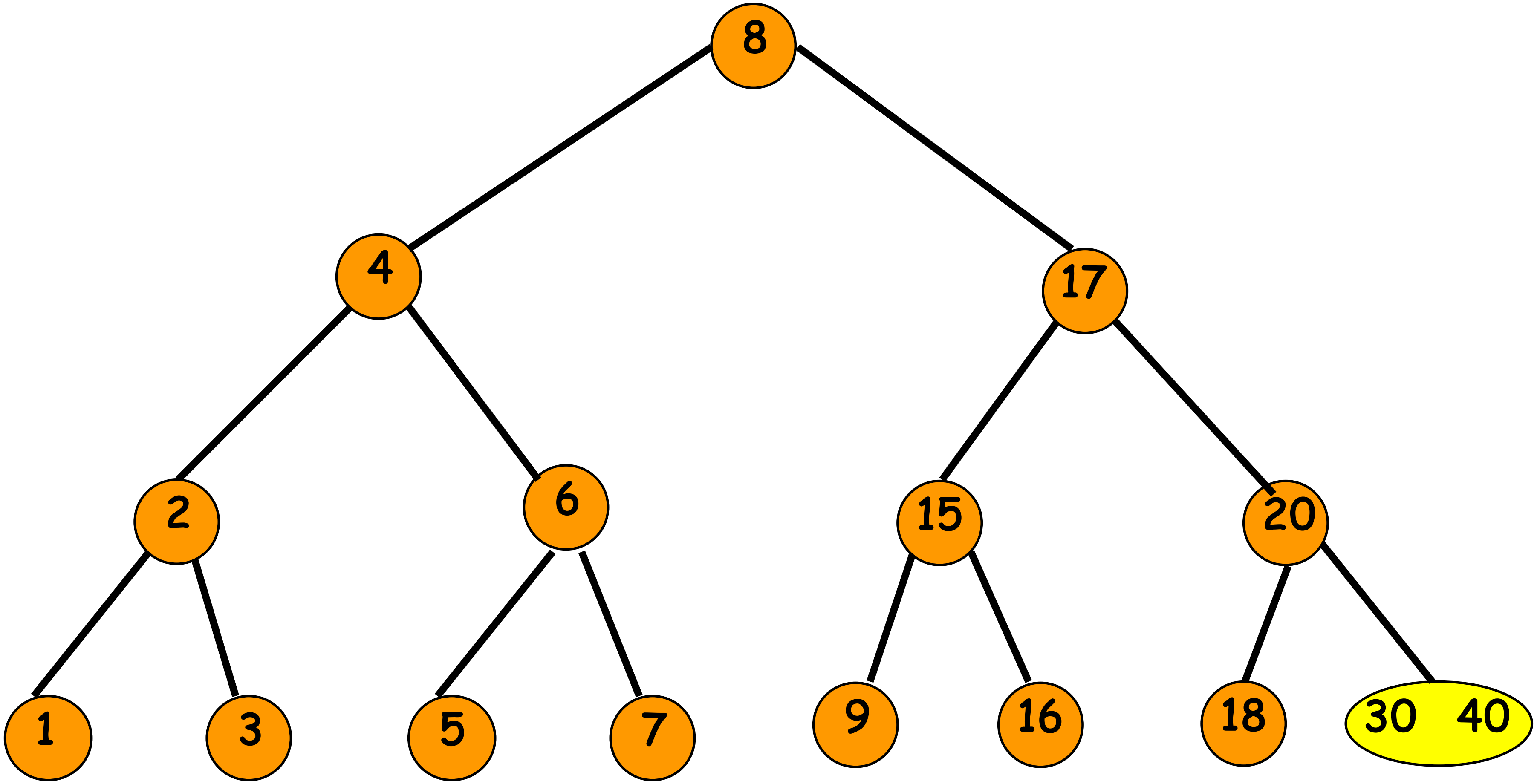
- Insert a pair with key = 4 plus a pointer into parent.

Insert (2-3 tree)



- Insert a pair with key = 8 plus a pointer into parent.
- There is no parent. So, create a new root.

Insert (2-3 tree)

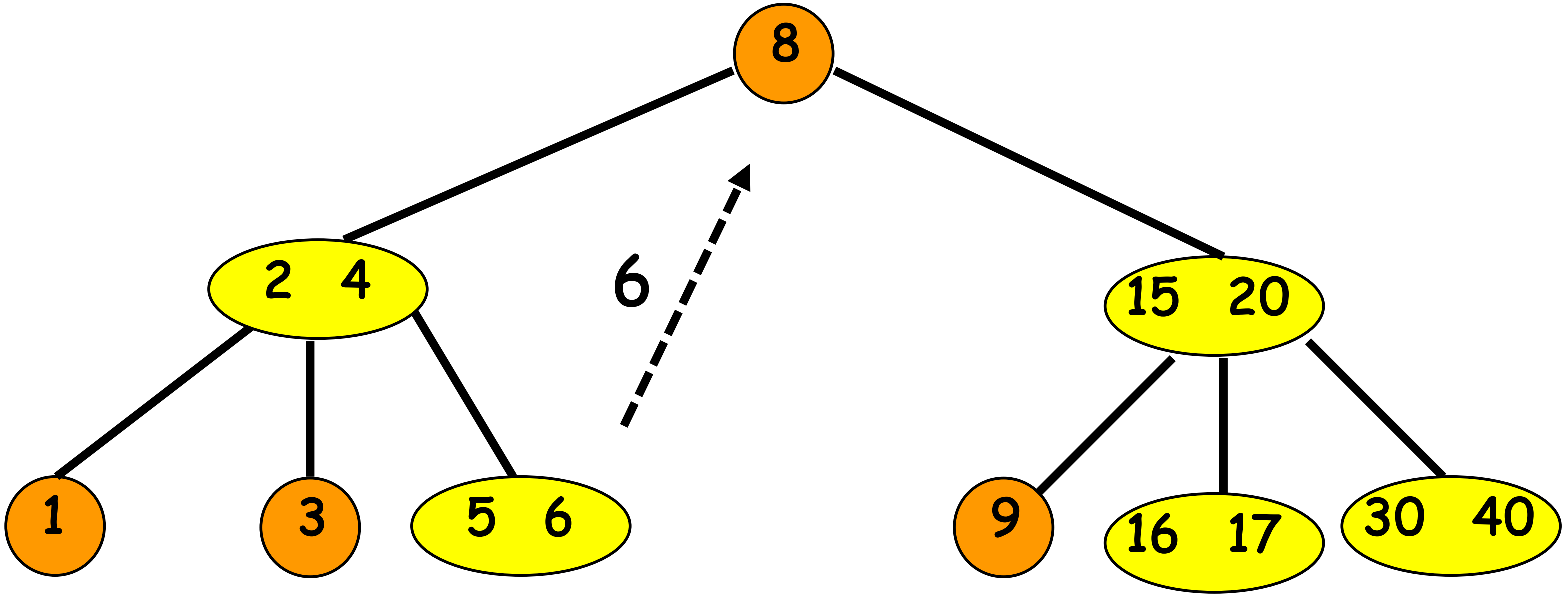


- Height increases by 1.

Deletion

- 做法並非唯一
- 底下我們僅討論一種可能的做法
- 我們事實上有如下"implicit" considerations
  - 盡可能善用已經存在的節點組織結構，盡量不影響樹的高度（減一），thus eliminating chain reaction
  - 盡可能搬動少量的資料在節點之間，因為資料的搬動（記憶體存取）的成本高
    - Memory wall

Delete (2-3 tree)

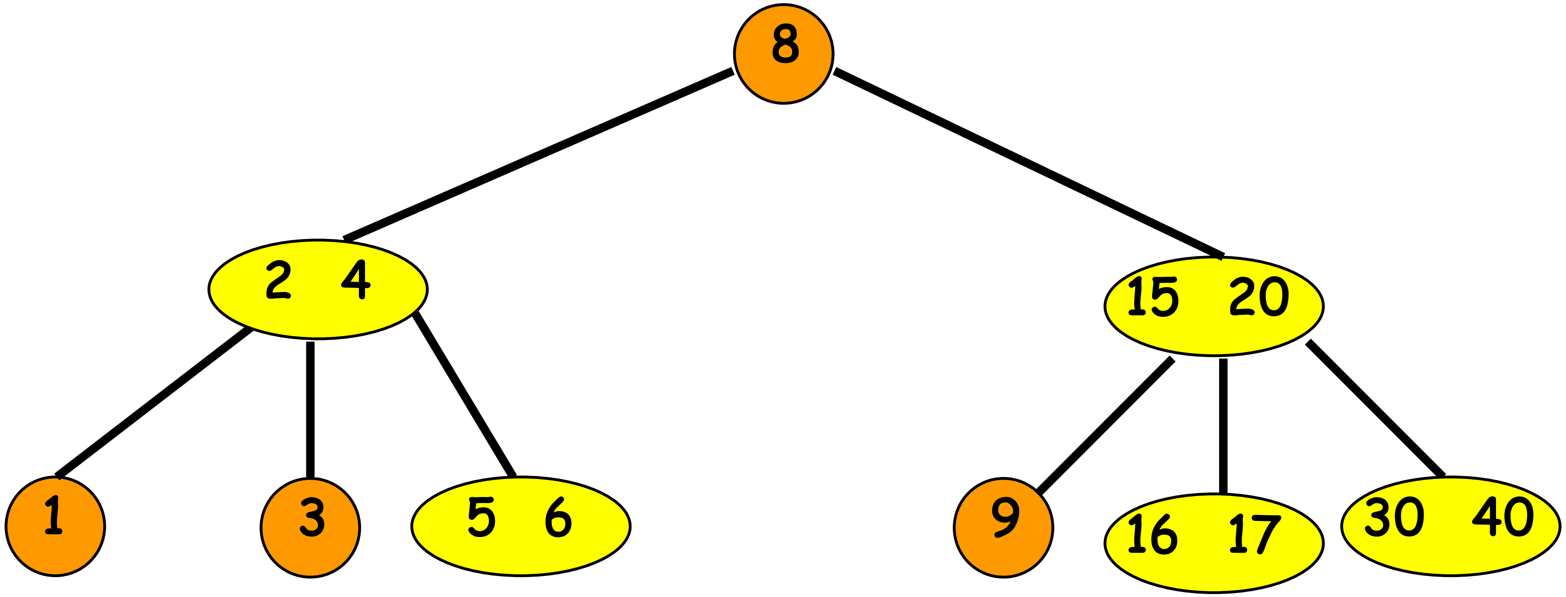


- Delete the pair with key = 8.
- Transform deletion from interior into deletion from a leaf.

// recall "delete an internal node in binary search tree"

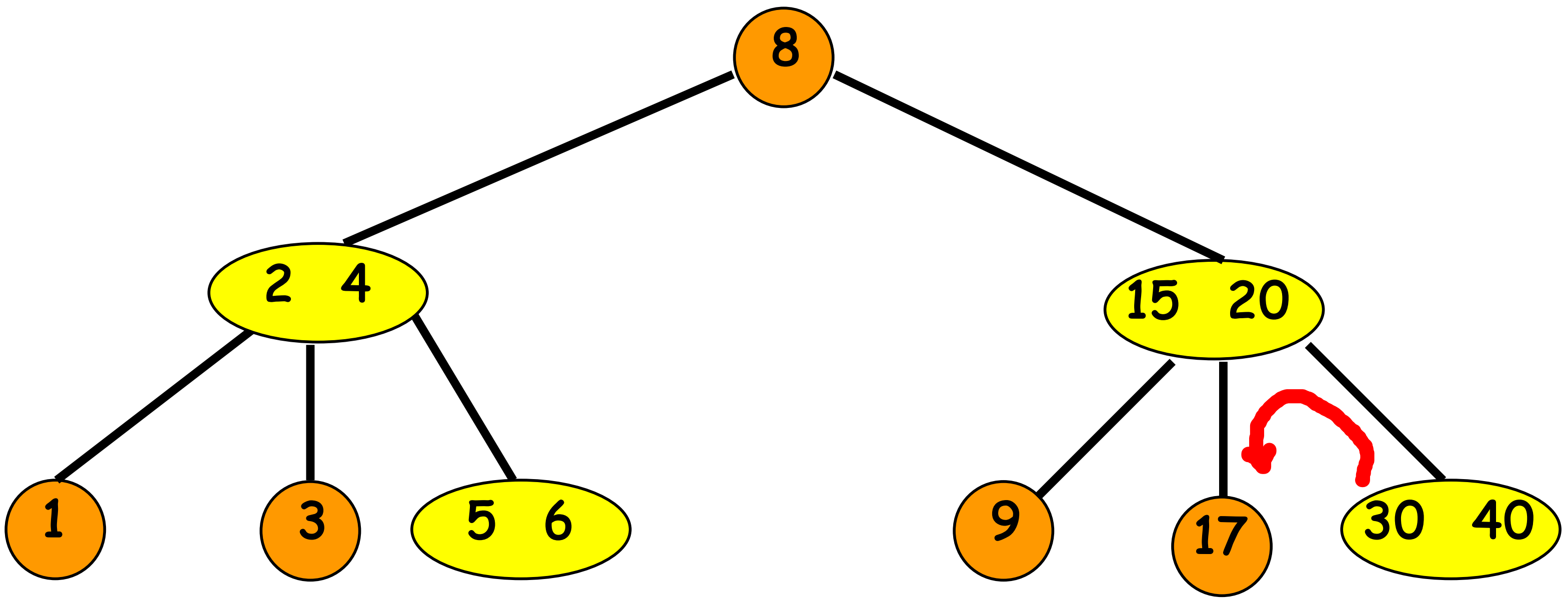
- Replace by largest in left subtree.

Delete from a Leaf
(往後僅討論刪除一筆在leaf的資料)



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

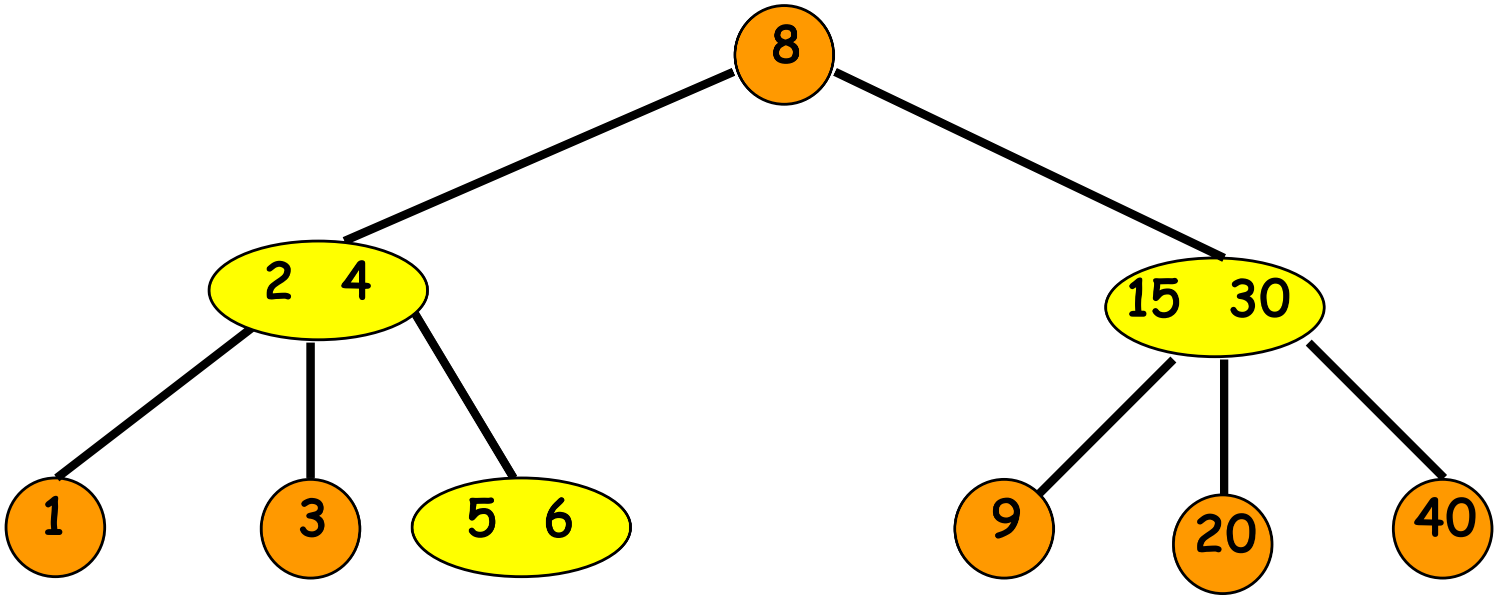
Delete from a Leaf



組織{9,15,20,30,40}至多5個nodes，至少3個nodes。此處原本有4個nodes，則我們盡可能利用這4個nodes的結構 if possible

- Delete the pair with key = 17.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If so borrow a pair and a subtree via parent node.

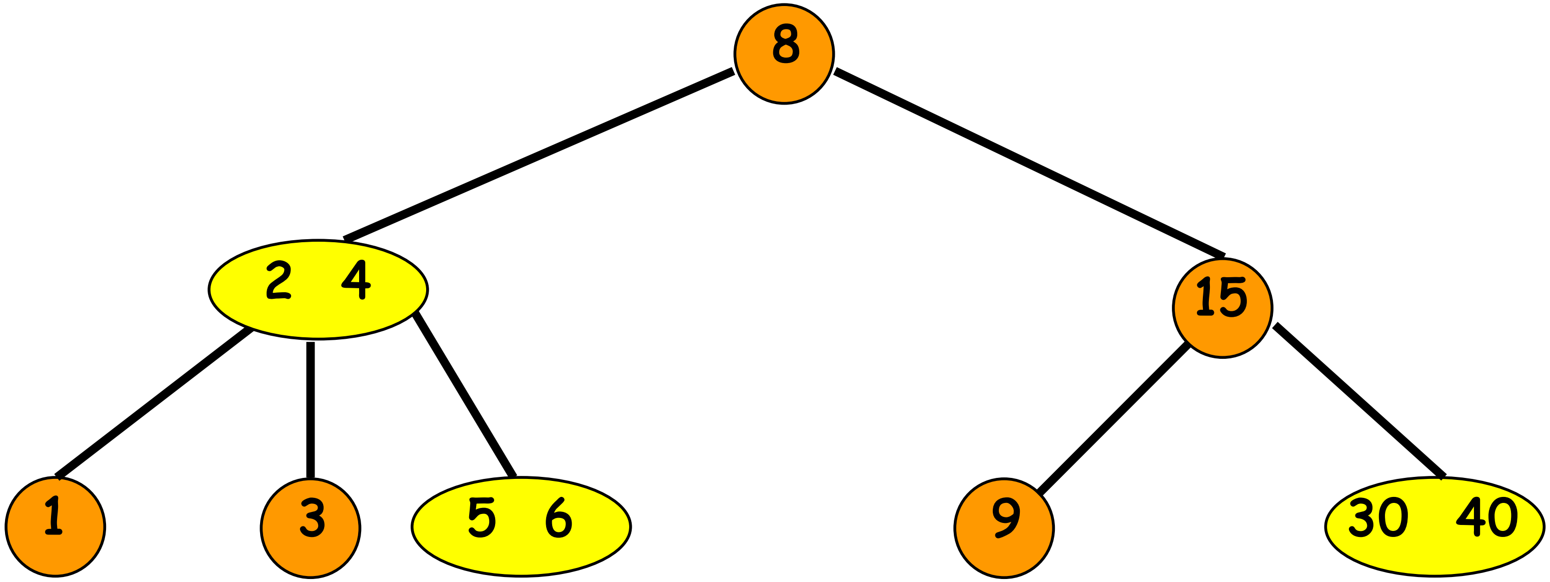
Delete from a Leaf



組織{9,15,30,40}至多4個nodes，至少2個nodes。此處原本有4個nodes，則我們盡可能利用這4個nodes的結構 if possible。但若為4個nodes，當中每個為2-node，則我們無法組織成一個子樹。因此改變成3個nodes，可行嗎？

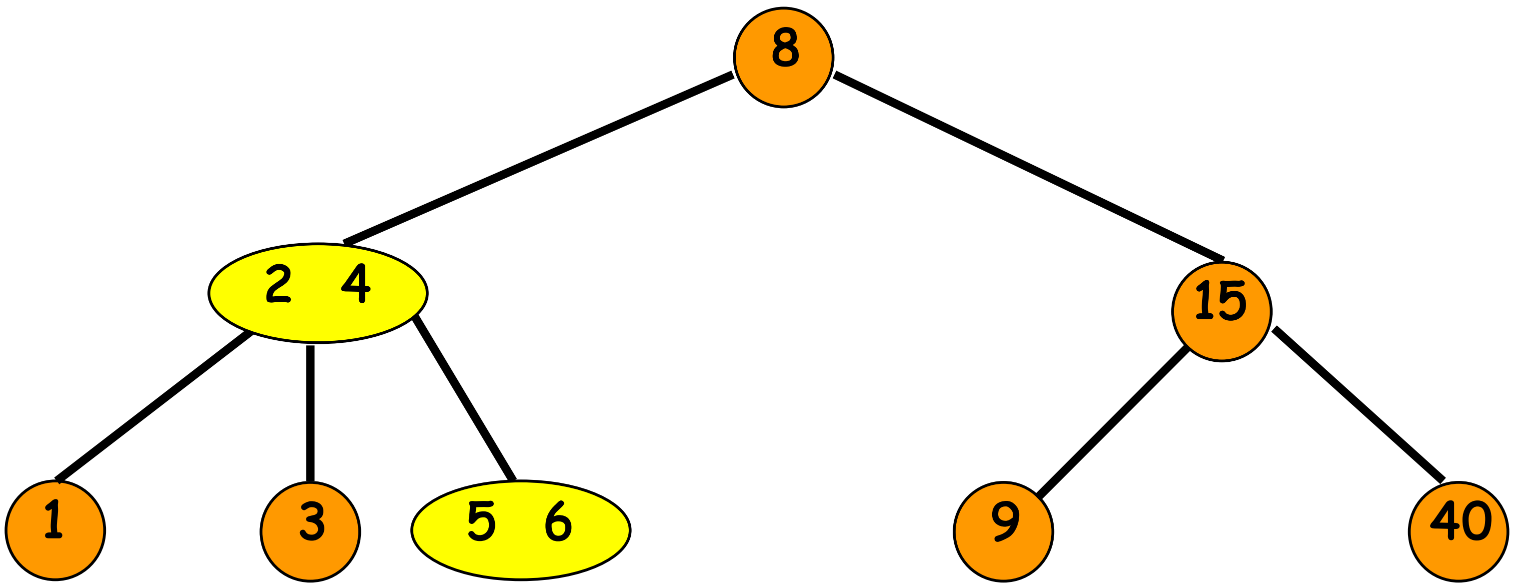
- Delete the pair with key = 20.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete from a Leaf



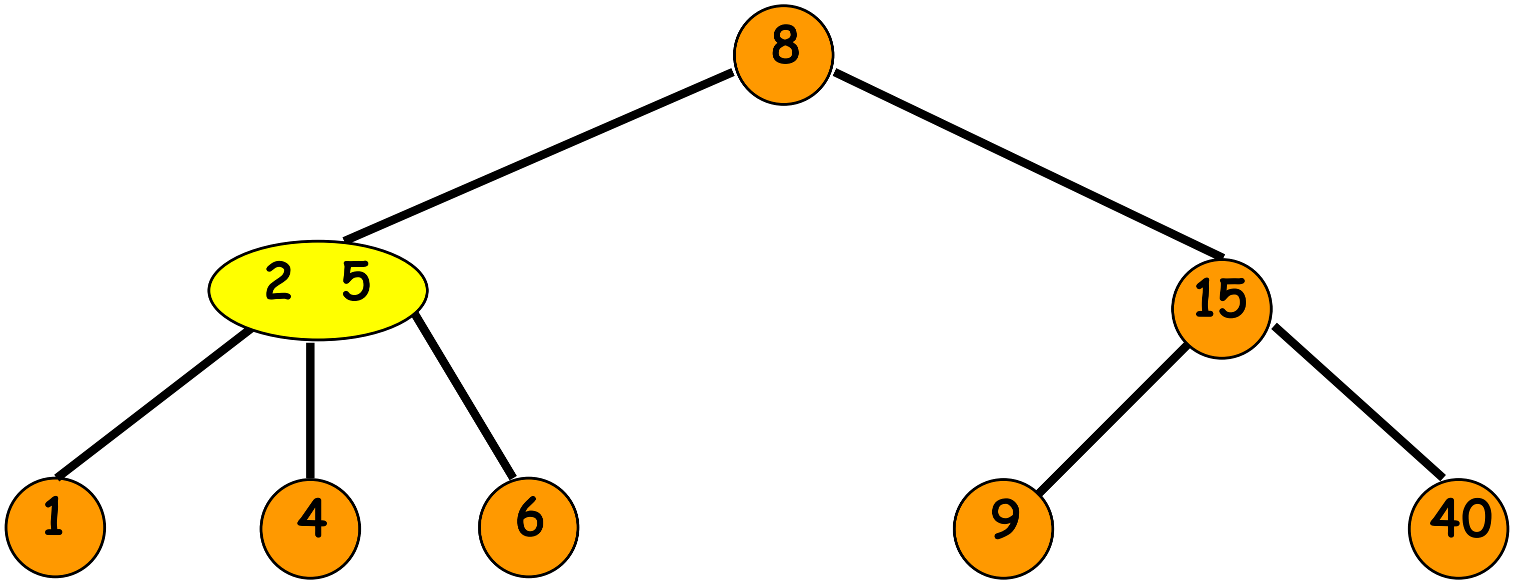
- Delete the pair with key = 30.
- Deletion from a 3-node.
- 3-node becomes 2-node.

Delete from a Leaf



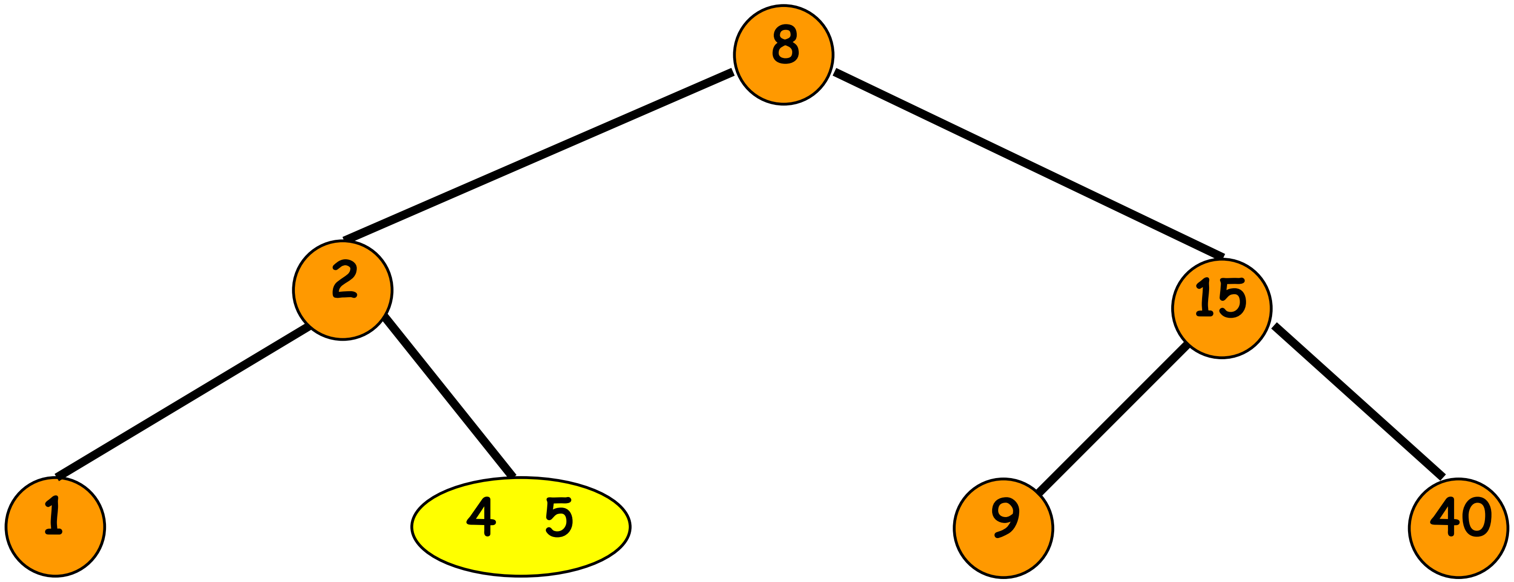
- Delete the pair with key = 3.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If so borrow a pair and a subtree via parent node.

Delete from a Leaf



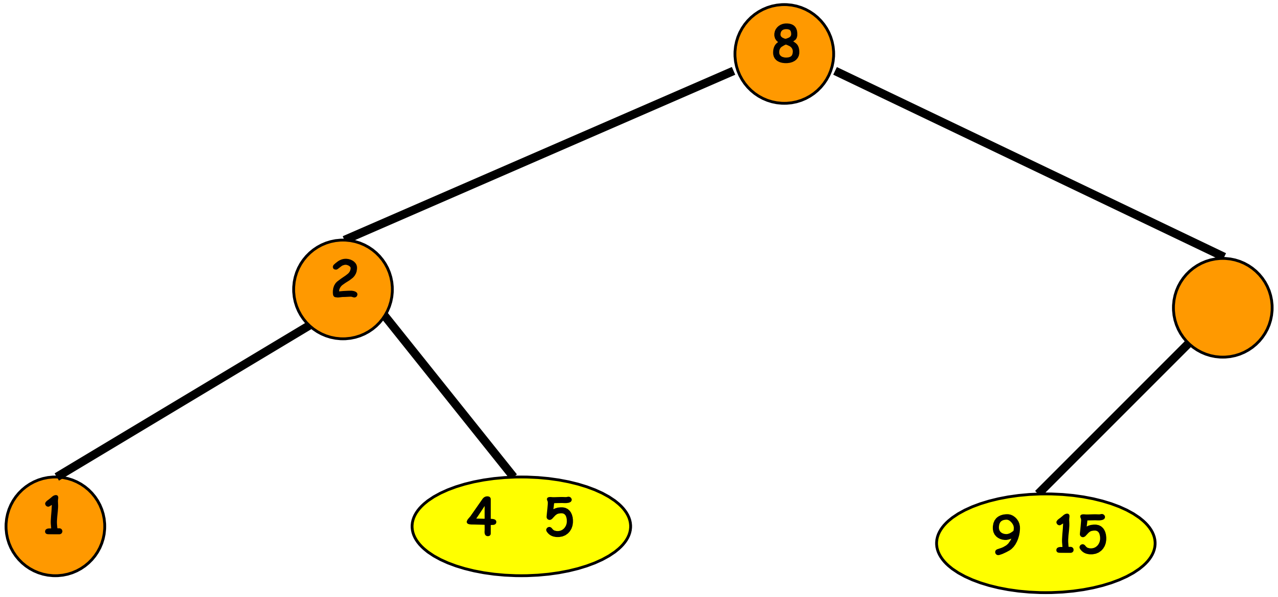
- Delete the pair with key = 6.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete from a Leaf



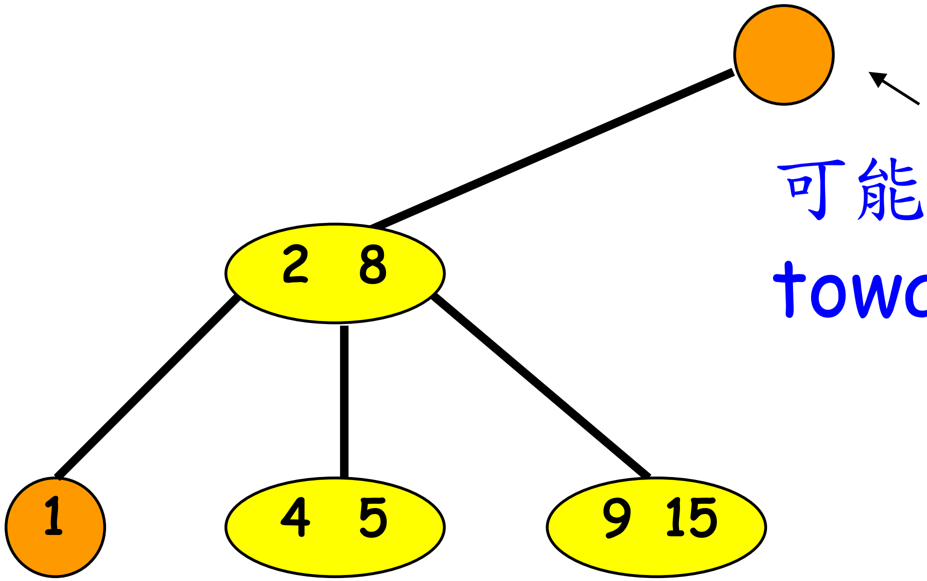
- Delete the pair with key = 40.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete from a Leaf



- Parent pair was from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

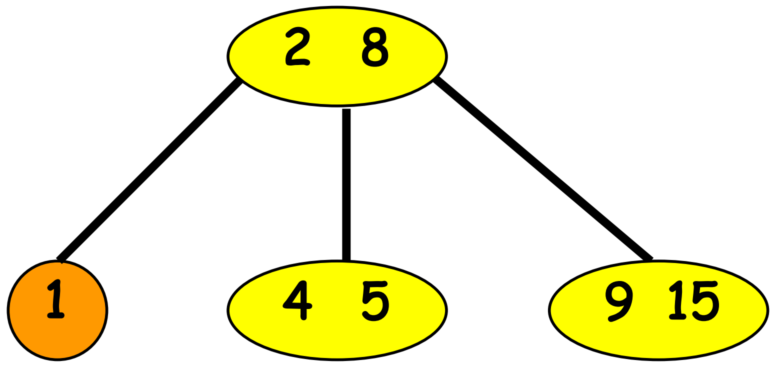
Delete from a Leaf



可能造成chain reaction towards ancestors

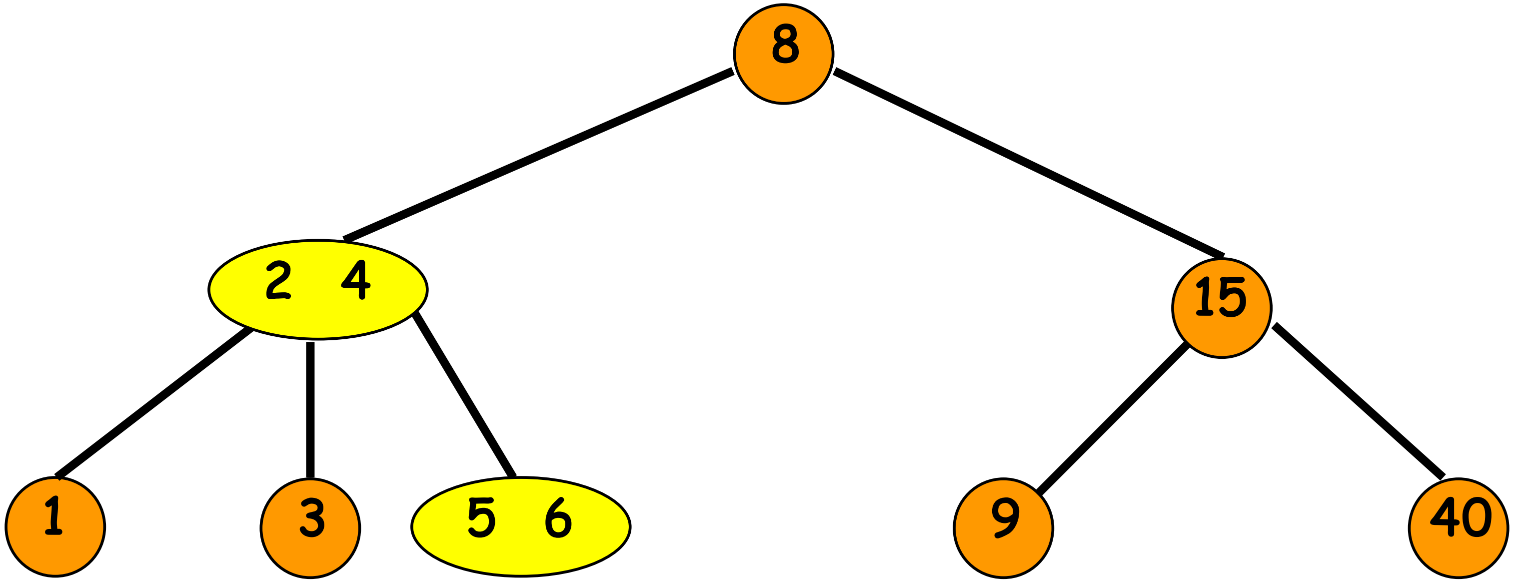
- Parent pair was from a 2-node.
- Check one sibling and determine if it is a 3-node.
- No sibling, so must be the root.
- Discard root. Left child becomes new root.

Delete from a Leaf



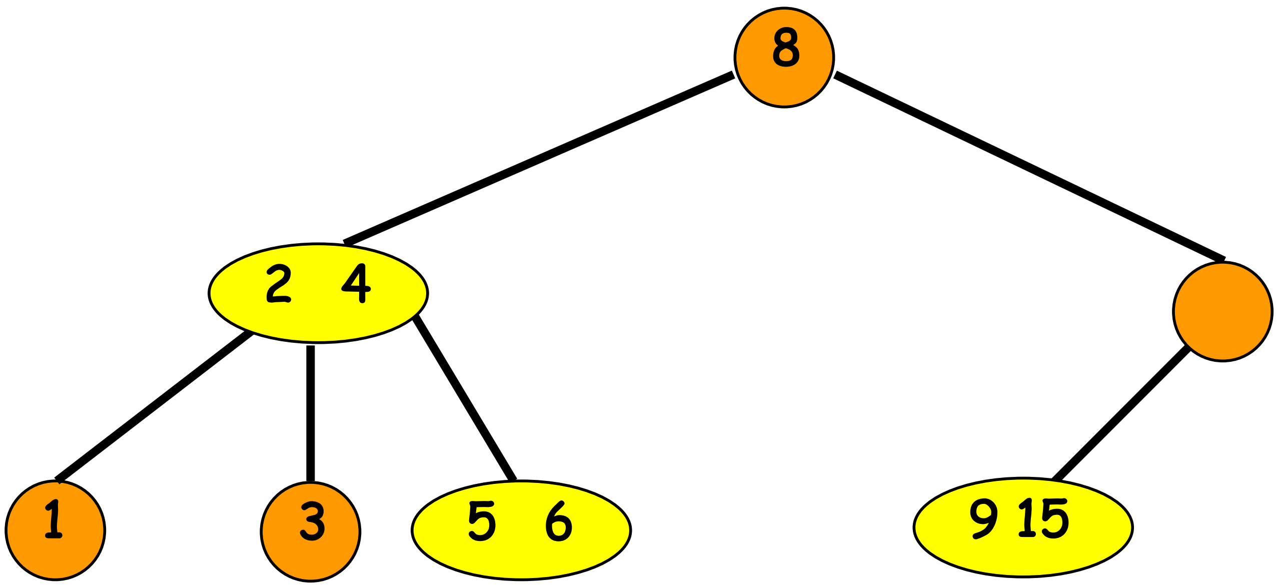
- Height reduces by 1.

Delete from a Leaf



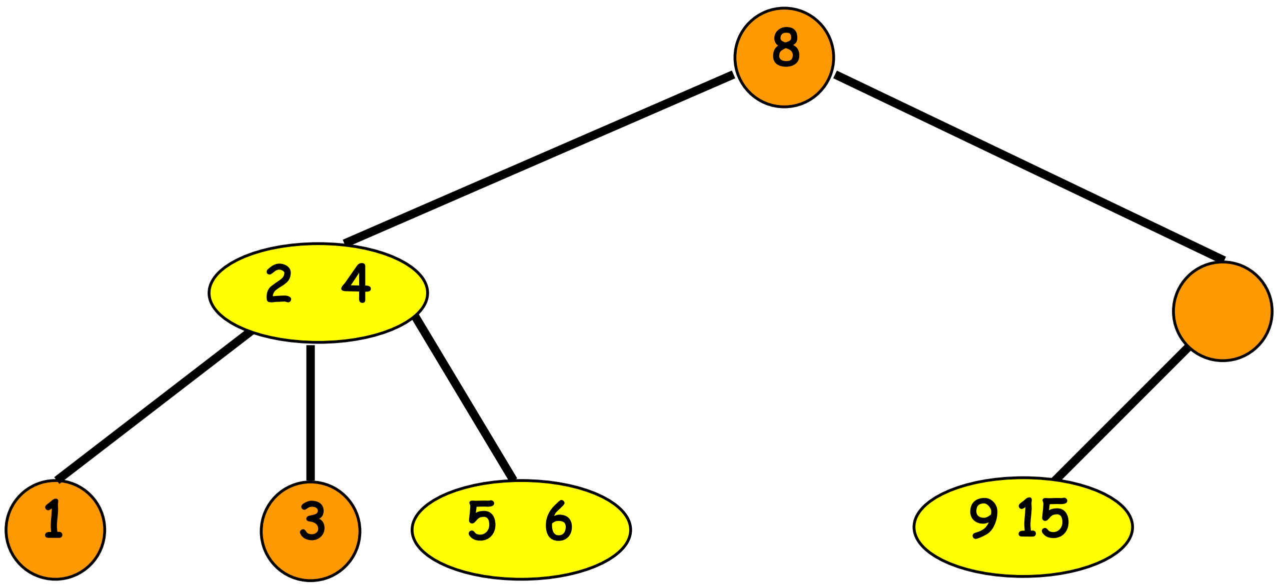
- Delete the pair with key = 40.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete from a Leaf

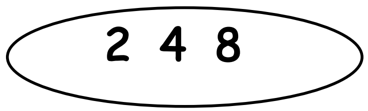


- Delete the pair with key = 40.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete from a Leaf

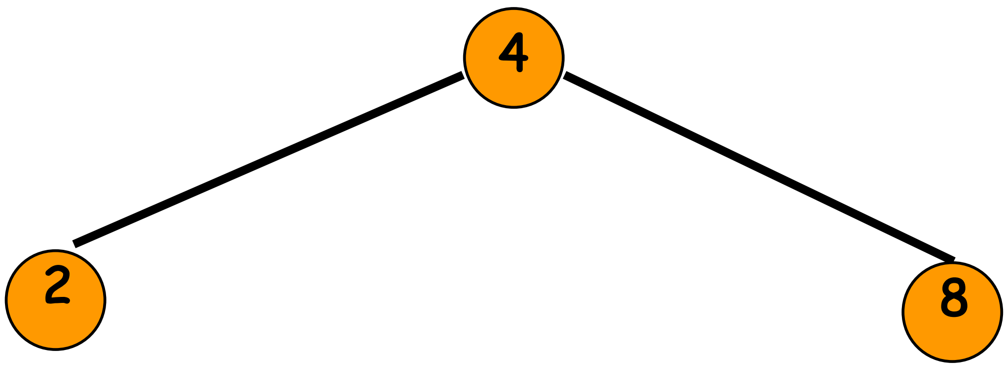


- Combine

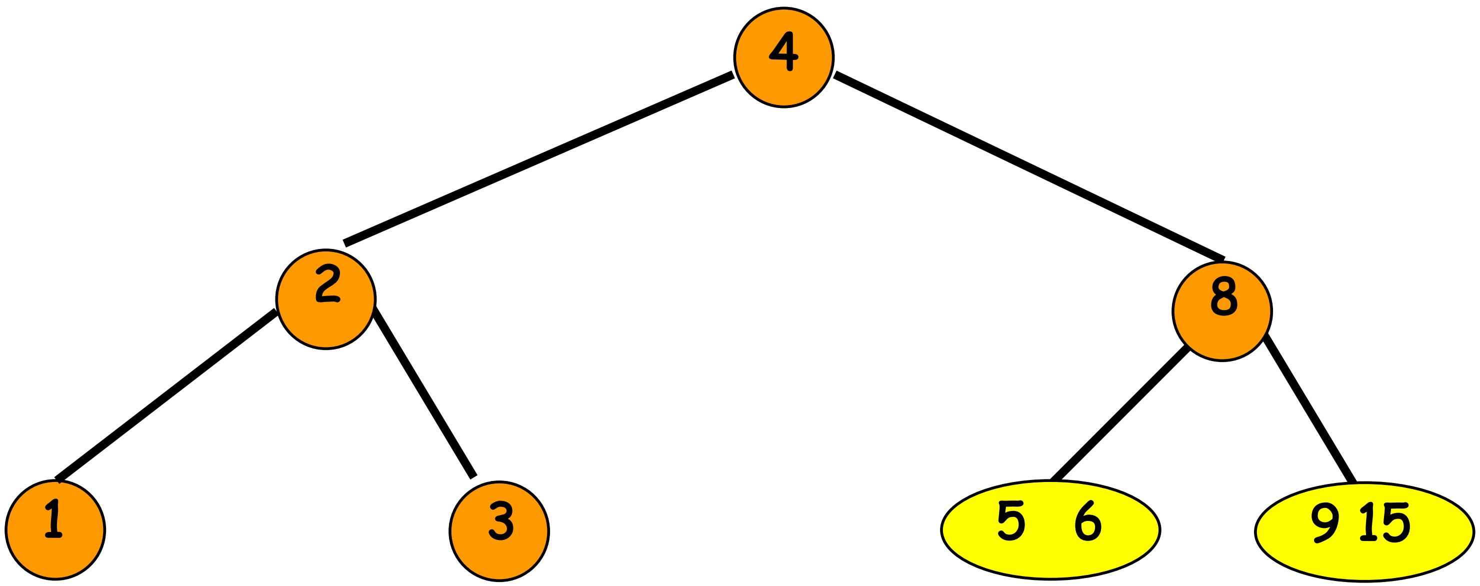


{2,4,8}可形成3個2-node's或1個2-node+1個3-node。我們從後者成為前者。

- Then, split



Delete from a Leaf



Operations

- Insert
  - Combine
  - Split
- Delete
  - Rotate
  - Combine
  - Split

How about Node Structure?

- 試想：若一個node節點內的資料量很大 (i.e., m很大時), 則該節點內部資料的查找變成是個不可忽視的成本
- 何不每個節點只儲存一筆資料 -> binary tree
- Thus, Red-Black tree
- Red-Black tree **為binary tree，它模擬2-3-4 tree (4-way search tree)**