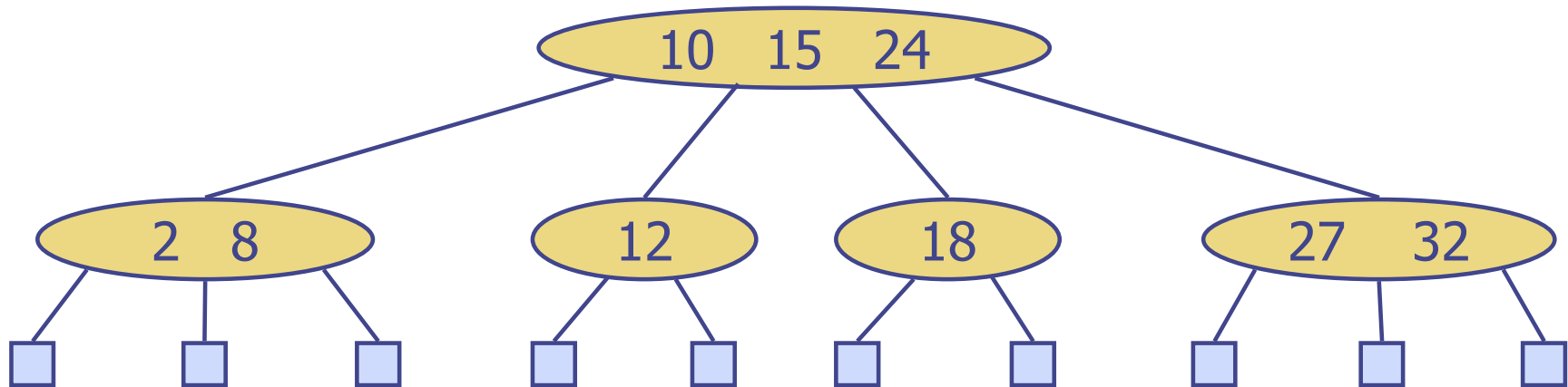
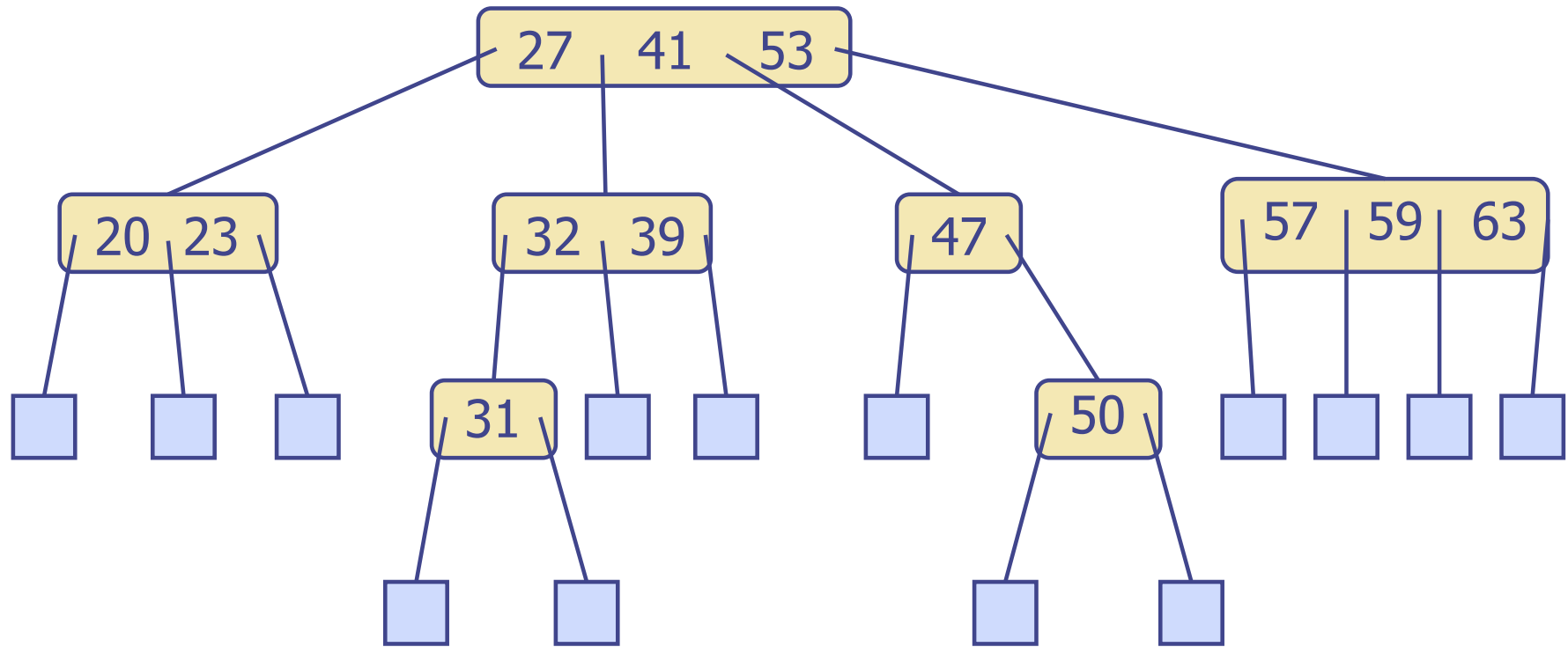


(2,4) Trees

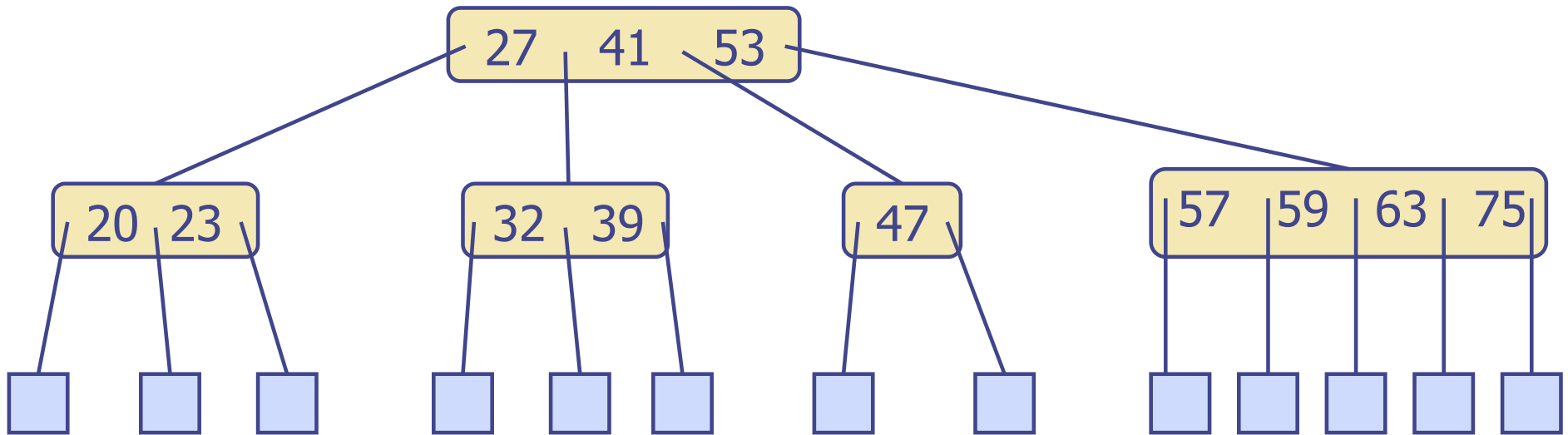
- ◆ A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search tree with the following properties
 - **Node-Size Property:** every internal node 2, 3, or 4 children
 - **Depth Property:** all the leaves are in the same level
- ◆ Depending on the number of children, an internal node of a (2,4) tree is called a 2-node, 3-node or 4-node



(2,4) Tree?



(2,4) Tree?



Height of a (2,4) Tree

◆ **Theorem:** A (2,4) tree storing n items has height $O(\log n)$

Proof:

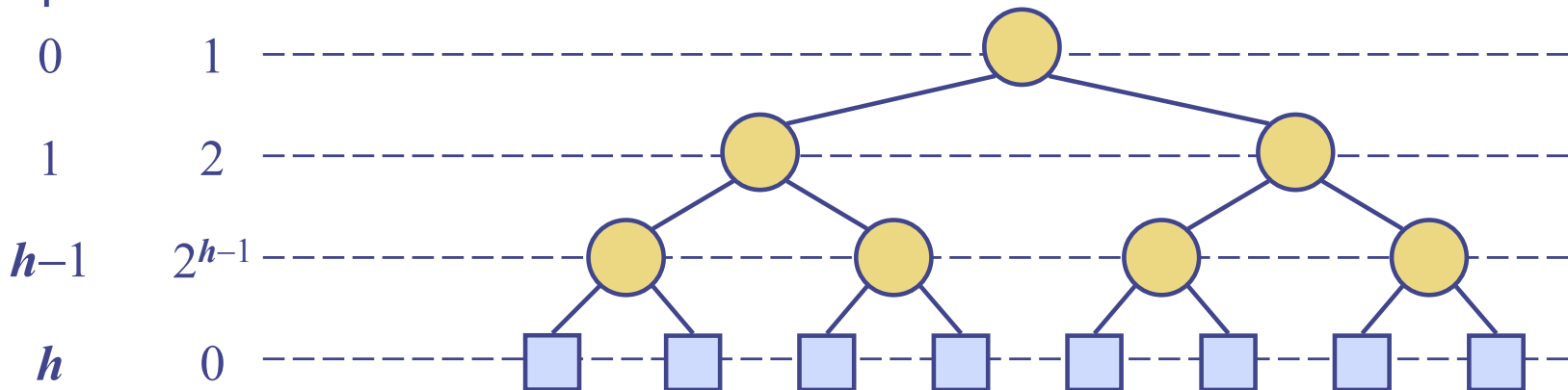
- Let h be the height of a (2,4) tree with n items
- Since there are at least 2^i items at depth $i = 0, \dots, h-1$ and no items at depth h , we have

$$n \geq 1 + 2 + 4 + \dots + 2^{h-1} = 2^h - 1$$

- Thus, $h \leq \log(n + 1)$

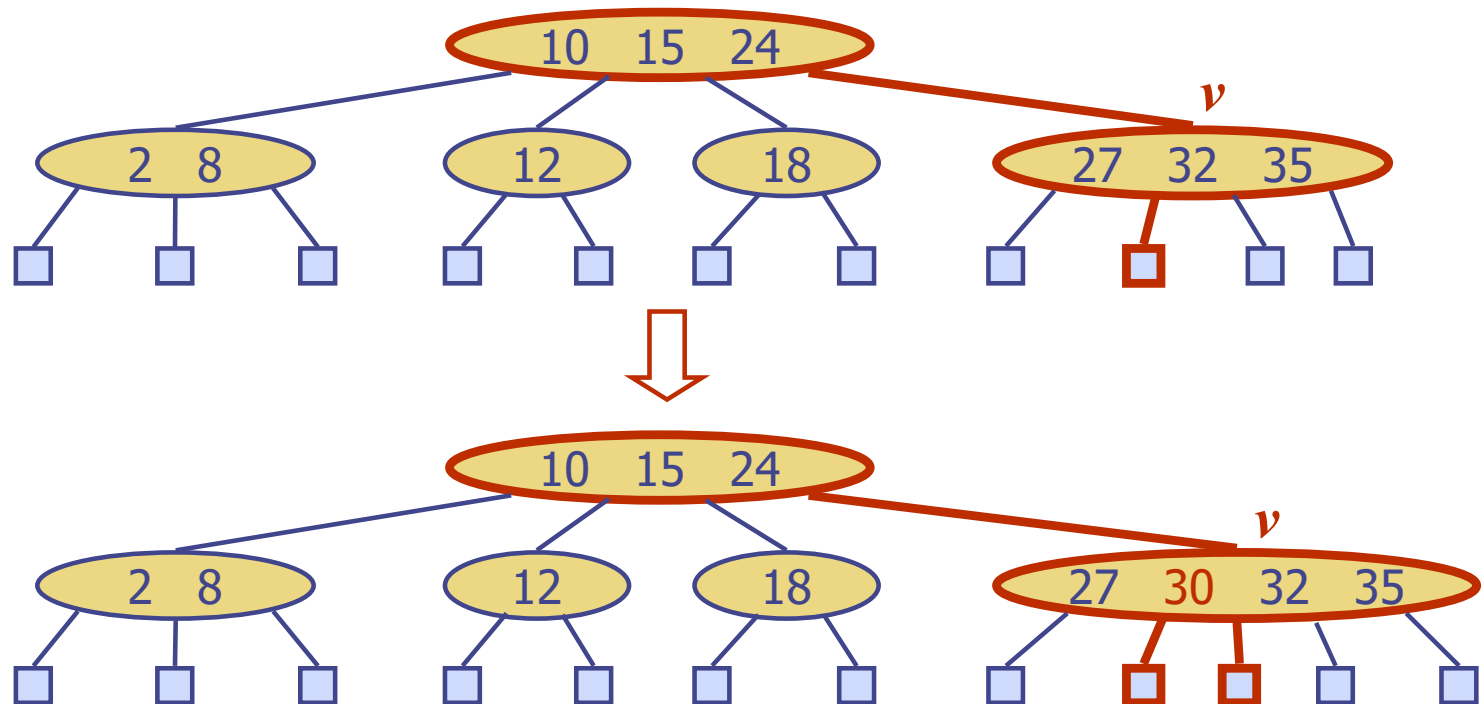
◆ Searching in a (2,4) tree with n items takes $O(\log n)$ time

depth items



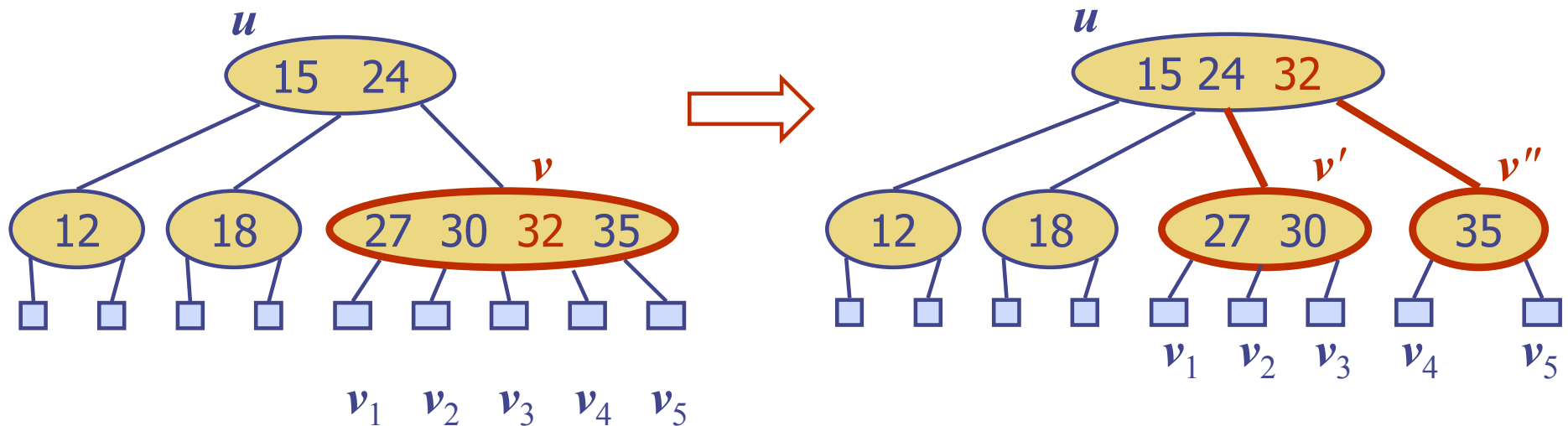
Insertion

- ◆ We insert a new item (k, o) at the parent v of the leaf reached by searching for k
 - We preserve the depth property but
 - We may cause an **overflow** (i.e., node v may become a 5-node)
- ◆ Example: inserting key 30 causes an overflow



Overflow and Split

- ◆ We handle an **overflow** at a 5-node v with a **split operation**:
 - let $v_1 \dots v_5$ be the children of v and $k_1 \dots k_4$ be the keys of v
 - node v is replaced nodes v' and v''
 - ◆ v' is a 3-node with keys $k_1 k_2$ and children $v_1 v_2 v_3$
 - ◆ v'' is a 2-node with key k_4 and children $v_4 v_5$
 - key k_3 is inserted into the parent u of v (a new root may be created)
- ◆ The overflow may propagate to the parent node u



Algorithm *put* (r, k, o)

In: Root r of a (2,4) tree, data item (k, o)

Out: {Insert data item (k, o) in (2,4) tree

Search for k to find the **lowest** insertion **internal** node v

Add the new data item (k, o) at node v

while node v *overflows* **do** {

if v is the root **then**

 Create a new empty root and set as parent of v

 Split v around the second key k' , move k' to parent, and
 update parent's children

$v \leftarrow$ parent of v

}

Algorithm *put* (r, k, o)

In: Root r of a (2,4) tree, data item (k, o)

Out: {Insert data item (k, o) in (2,4) tree

Search for k to find the **lowest** insertion **internal** node v $O(\log n)$

Add the new data item (k, o) at node v $O(1)$

while node v *overflows* **do** {

if v is the root **then**

 Create a new empty root and set as parent of v $O(1)$

 Split v around the second key k' , move k' to parent, and update parent's children

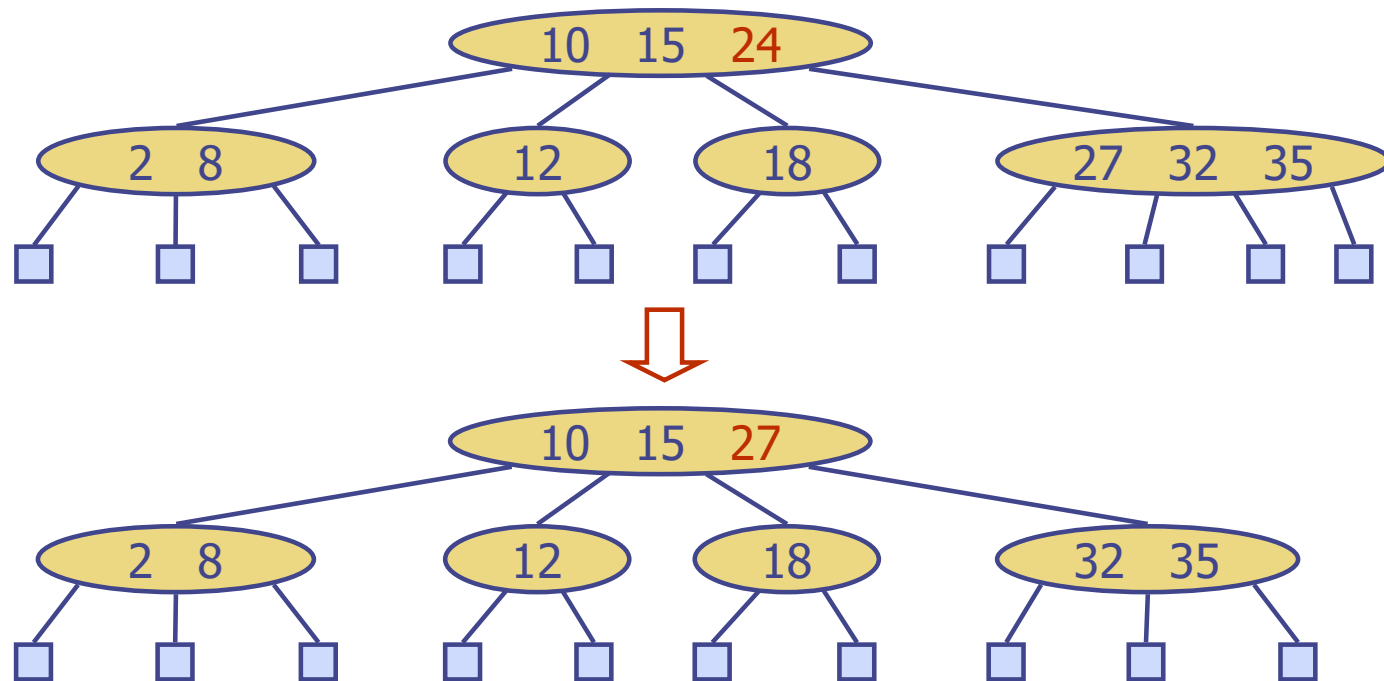
$v \leftarrow$ parent of v

}

Time complexity of put is $O(\log n)$

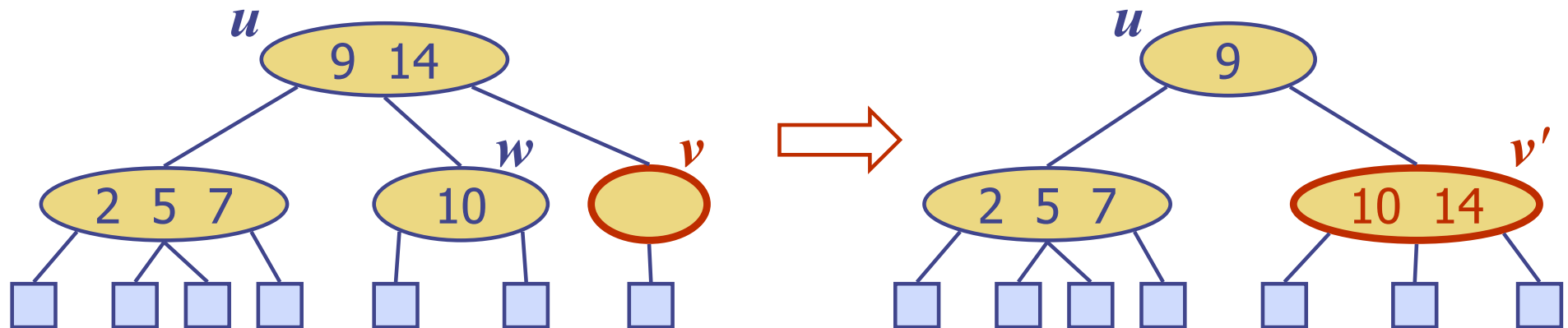
Deletion

- ◆ We reduce deletion of an entry to the case where the item is at the node with leaf children
- ◆ Otherwise, we replace the entry with its inorder successor (or, equivalently, with its inorder predecessor) and delete the latter entry
- ◆ Example: to delete key 24, we replace it with 27 (inorder successor)



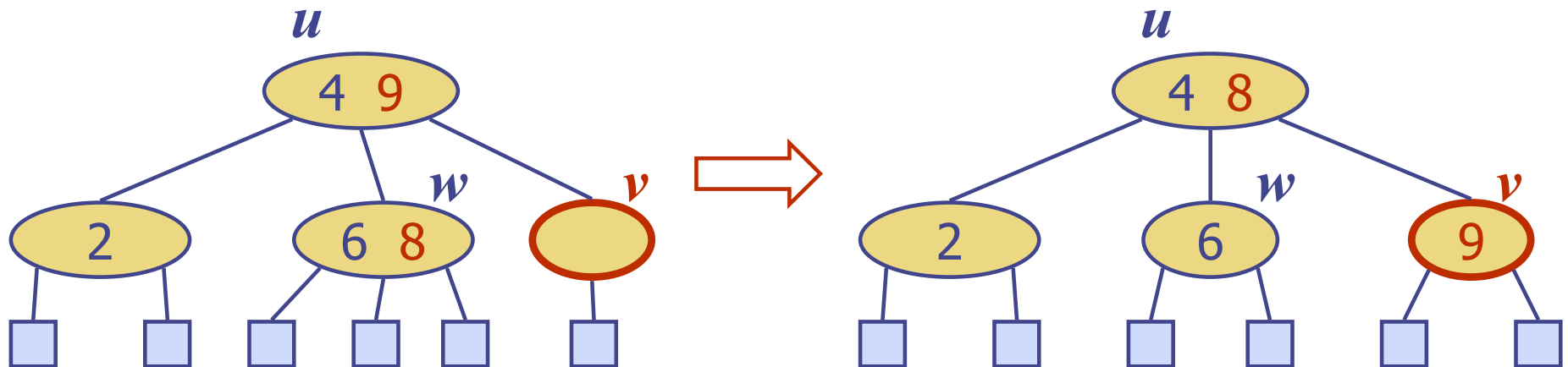
Underflow and Fusion

- ◆ Deleting an entry from a node v may cause an **underflow**, where node v becomes a 1-node with one child and no keys
- ◆ To handle an underflow at node v with parent u , we consider two cases
- ◆ **Case 1:** the adjacent siblings of v are 2-nodes
 - **Fusion operation:** we merge v with an adjacent sibling w and move an entry from u to the merged node v'
 - After a fusion, the underflow may propagate to the parent u



Underflow and Transfer

- ◆ To handle an underflow at node v with parent u , we consider two cases
- ◆ **Case 2:** an adjacent sibling w of v is a 3-node or a 4-node
 - **Transfer operation:**
 1. we move a child of w to v
 2. we move an item from u to v
 3. we move an item from w to u
 - After a transfer, no underflow occurs



Algorithm *remove*(r, k)

In: Root r of a (2,4) tree, key k

Out: {remove data item with key k from the tree}

Find the node v storing key k

Remove (k, o) from v replacing it with successor if needed

while node v *underflows* **do** {

if v is the root **then**

 make the first child of v the new root

else if a sibling has at least 2 keys **then**

 perform a transfer operation

else {

 perform a fusion operation

$v \leftarrow$ parent of v

 }

}

Algorithm *remove*(r, k)

In: Root r of a (2,4) tree, key k

Out: {remove data item with key k from the tree}

Find the node v storing key k } $O(\log n)$

Remove (k, o) from v replacing it with successor if needed }

while node v *underflows* **do** { $O(\log n)$

if v is the root **then**

 make the first child of v the new root

else if a sibling has at least 2 keys **then**

 perform a transfer operation

else {

 perform a fusion operation

$v \leftarrow$ parent of v

 }

} Time complexity of remove: $O(\log n)$

$O(1)$