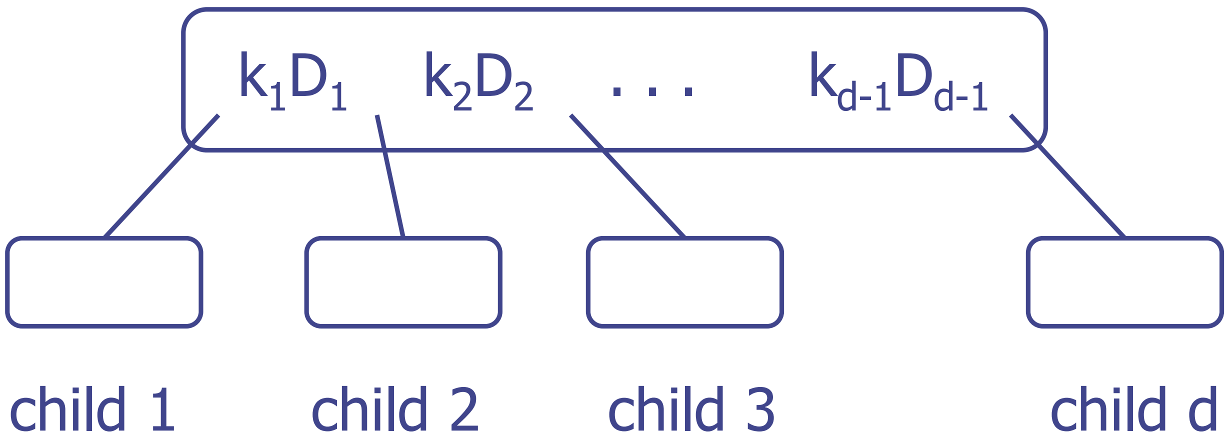
Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)

Rule: Number of children = 1 + number of data items in a node

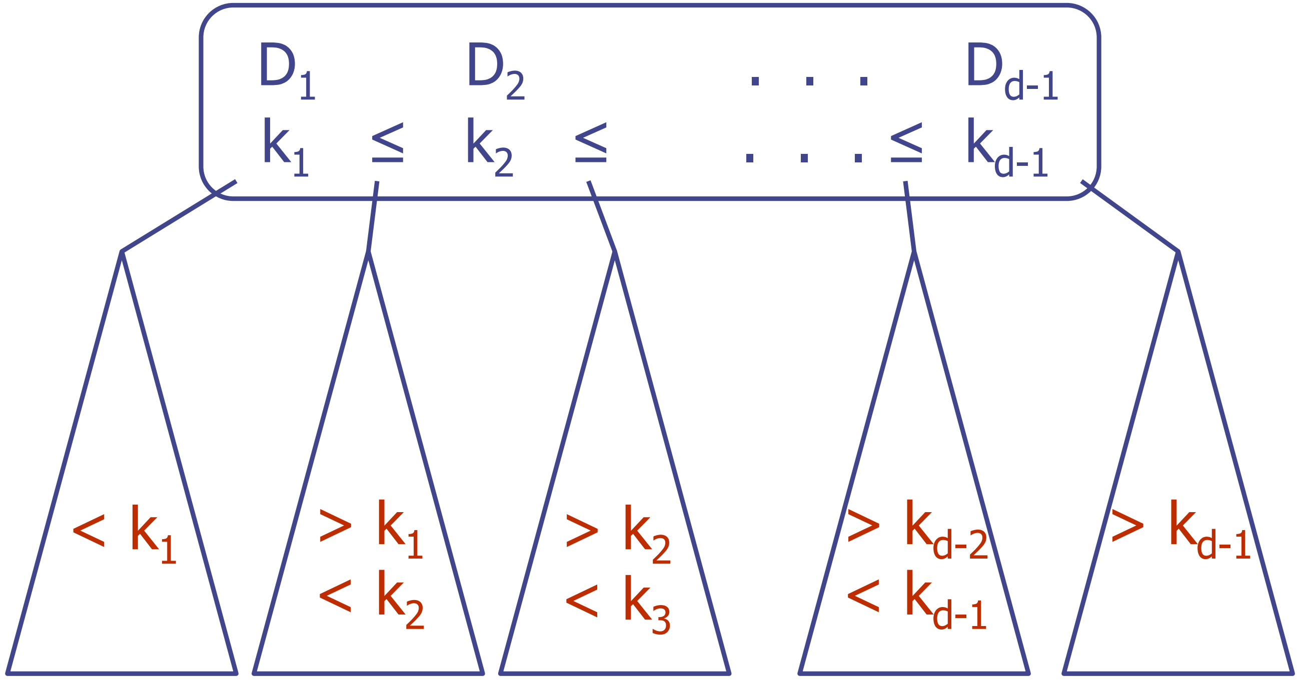


d is the degree or order of the tree

Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

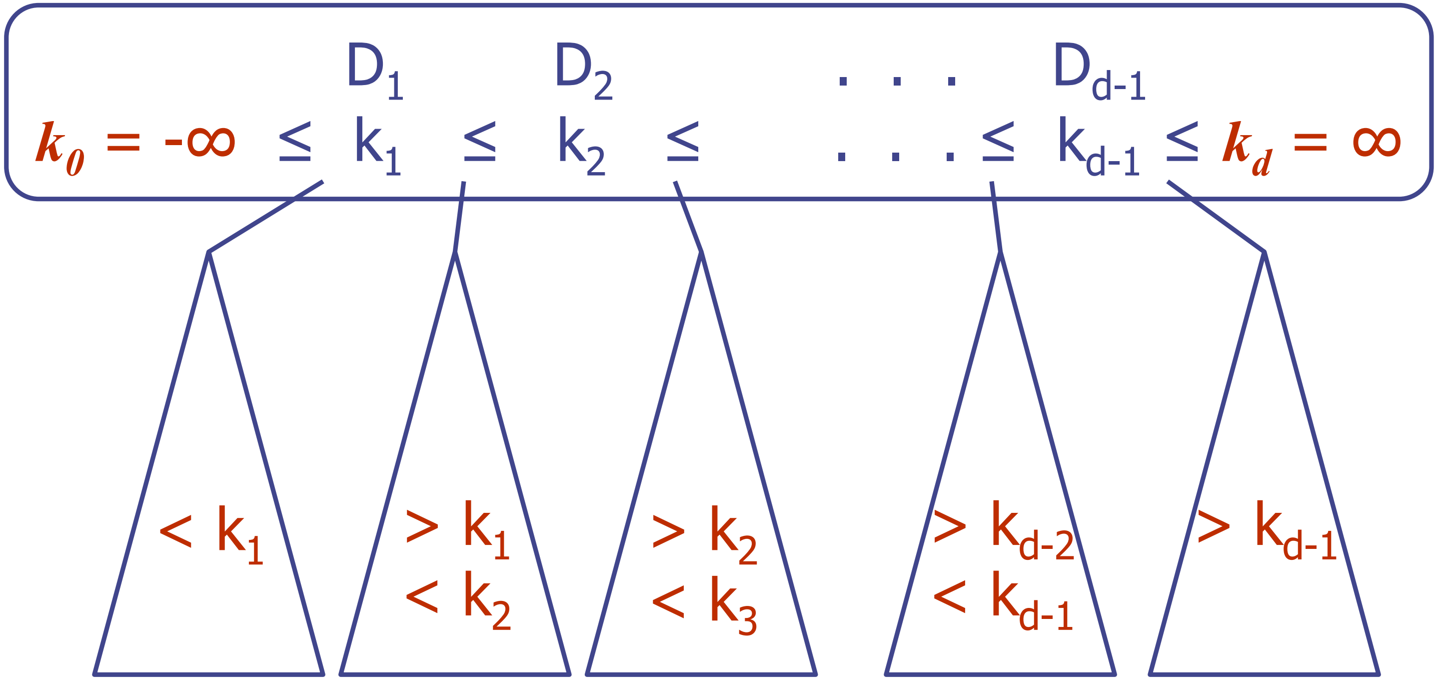
- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)
- An internal node storing keys $k_1 \le k_2 \le \ldots \le k_{d-1}$ has d children $v_1\, v_2\, \ldots\, v_d$ such that



Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

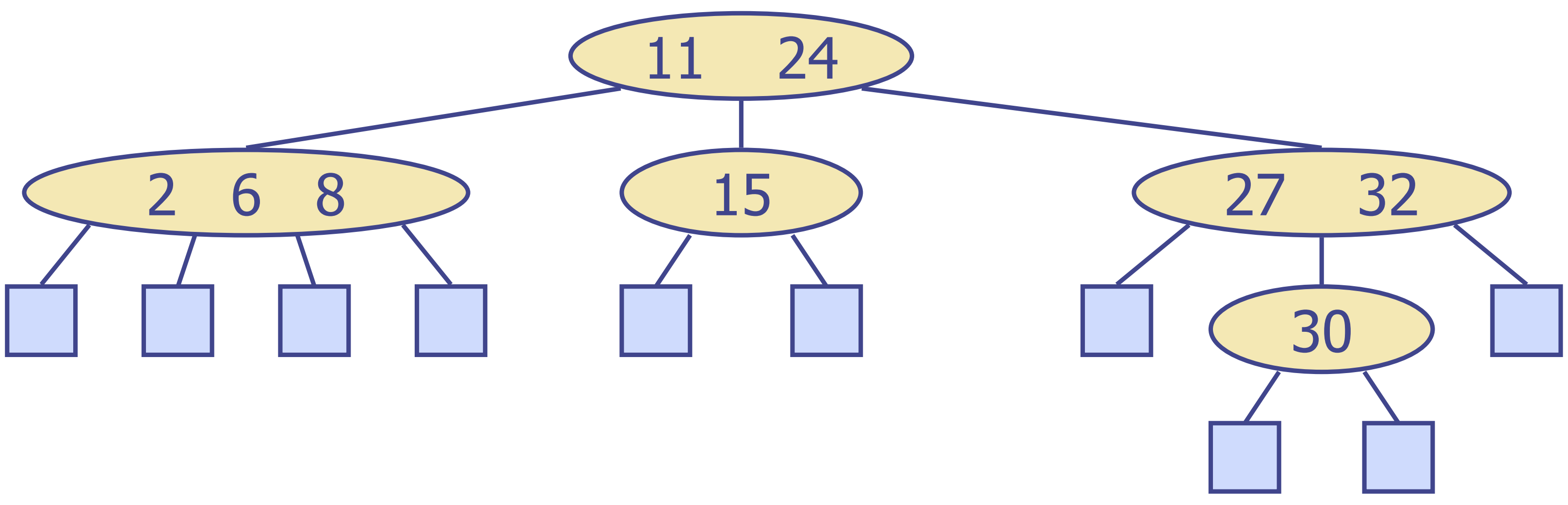
- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)
- An internal node storing keys $k_1 \le k_2 \le \ldots \le k_{d-1}$ has d children $v_1\, v_2\, \ldots\, v_d$ such that
- By convenience we add sentinel keys $k_0 = -\infty$ and $k_d = \infty$



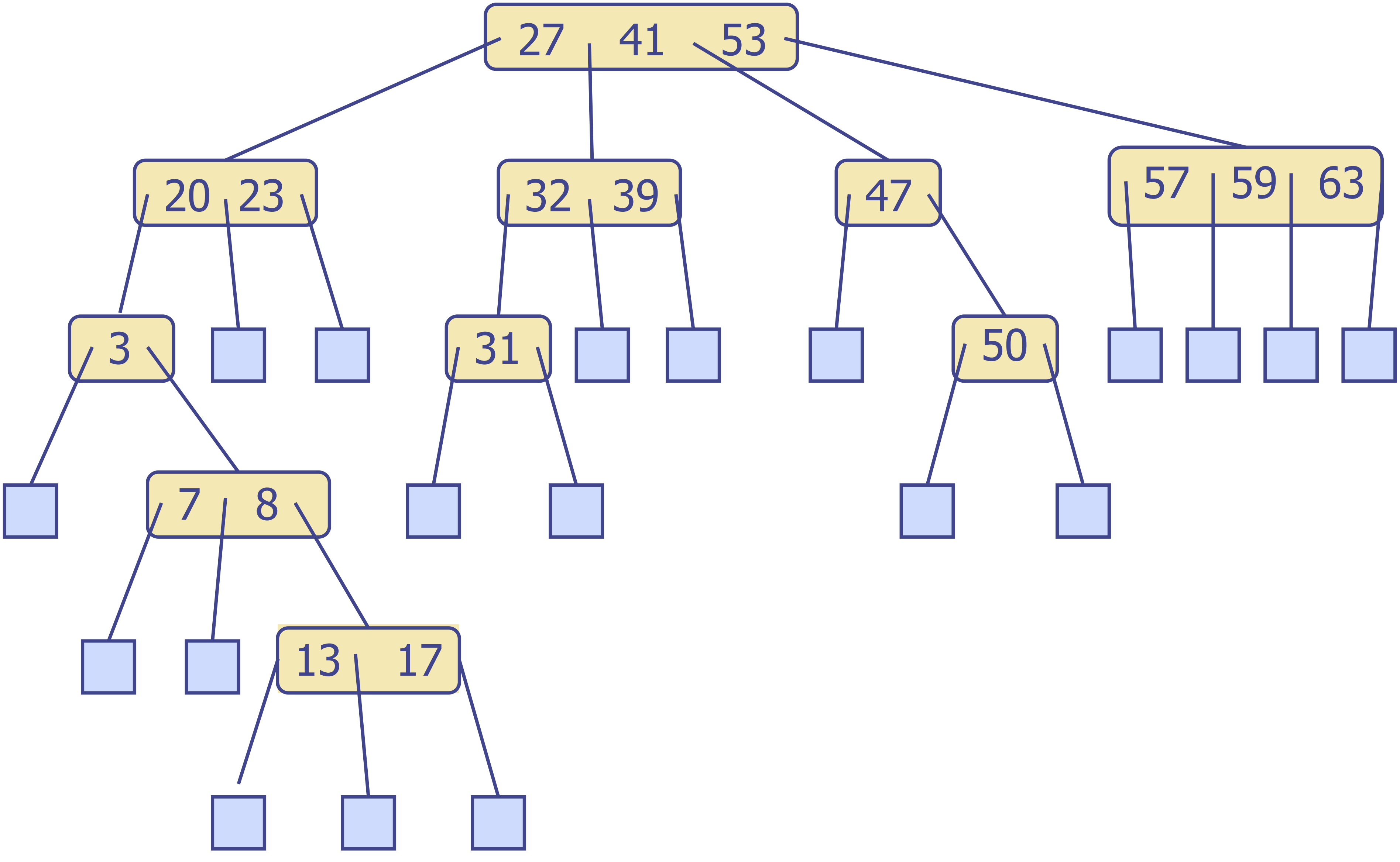
Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

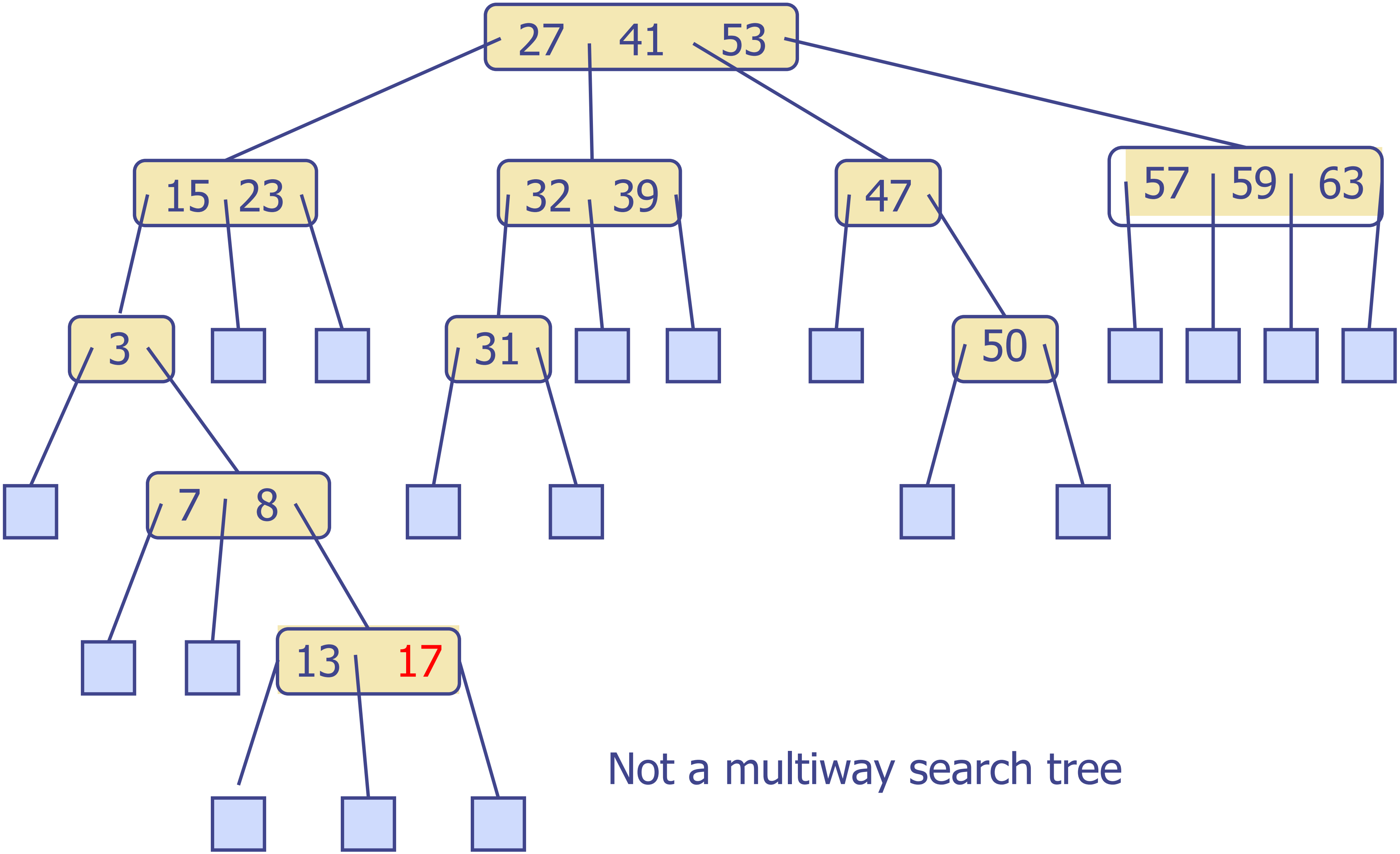
- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)
- An internal node storing keys $k_1 \leq k_2 \leq \ldots \leq k_{d-1}$ has d children $v_1\, v_2\, \ldots\, v_d$ such that
- By convenience we add sentinel keys $k_0 = -\infty$ and $k_d = \infty$
- The leaves store no items and serve as placeholders



Multi-Way Search Tree of Degree 4?



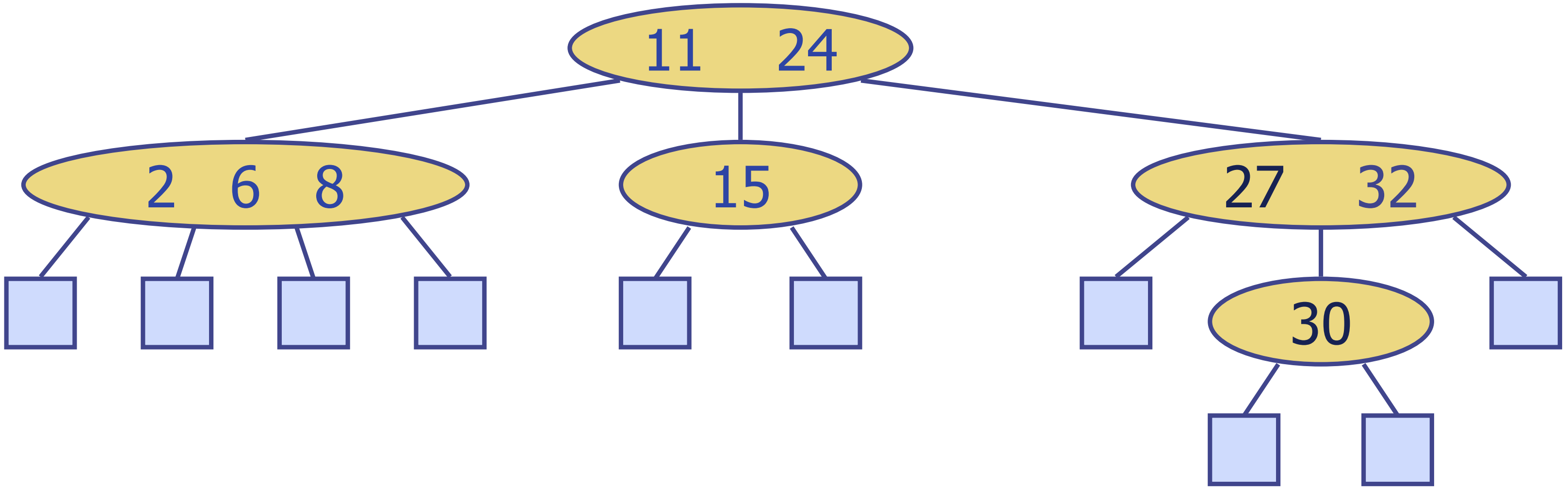
Multi-Way Search Tree of Degree 4?



Not a multiway search tree

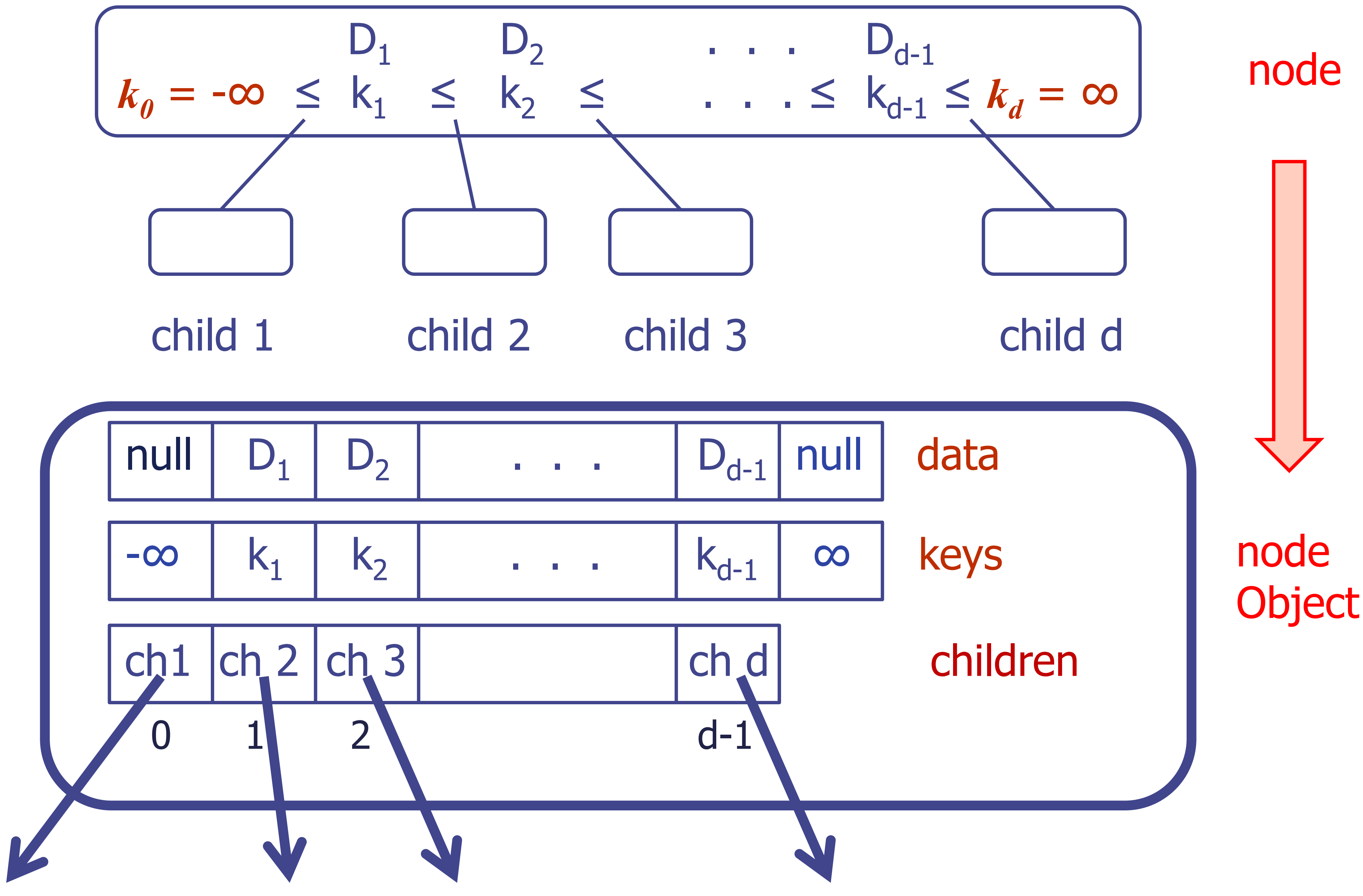
Multi-Way Inorder Traversal

- We can extend the notion of inorder traversal from binary trees to multi-way search trees
- An inorder traversal of a multi-way search tree visits the keys in increasing order

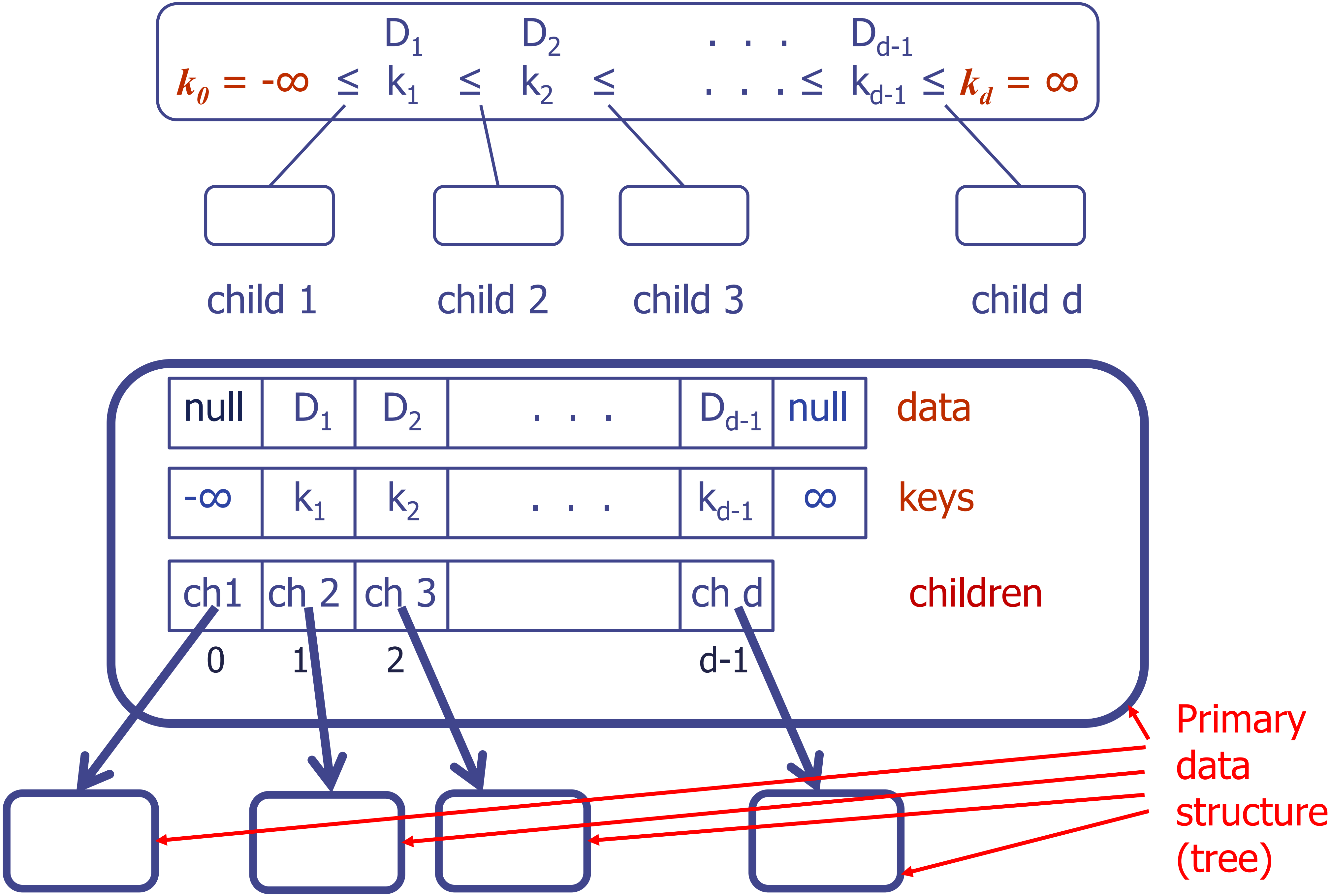


Inorder traversal:

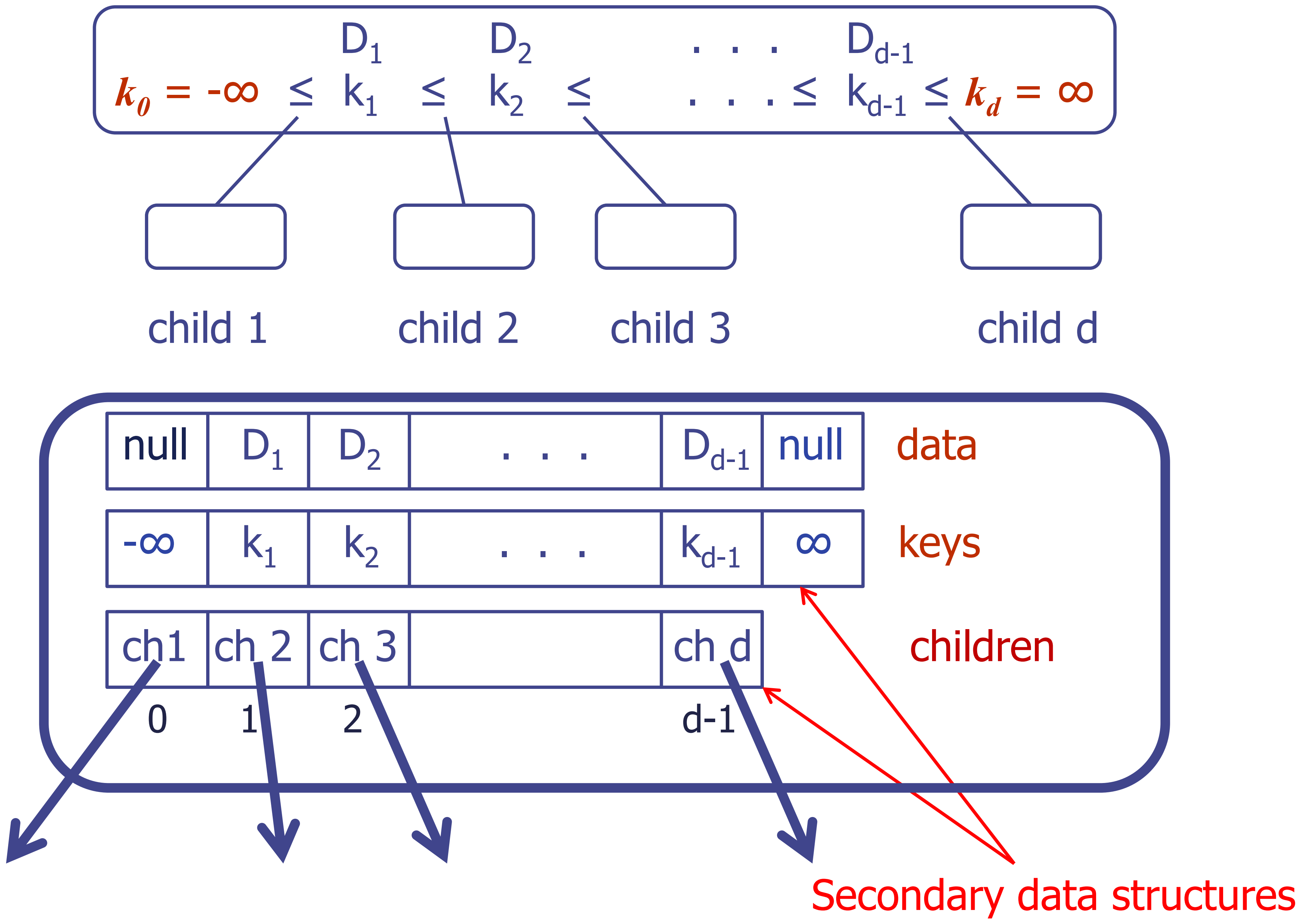
Data Structures for Multi-Way Search Trees



Data Structures for Multi-Way Search Trees

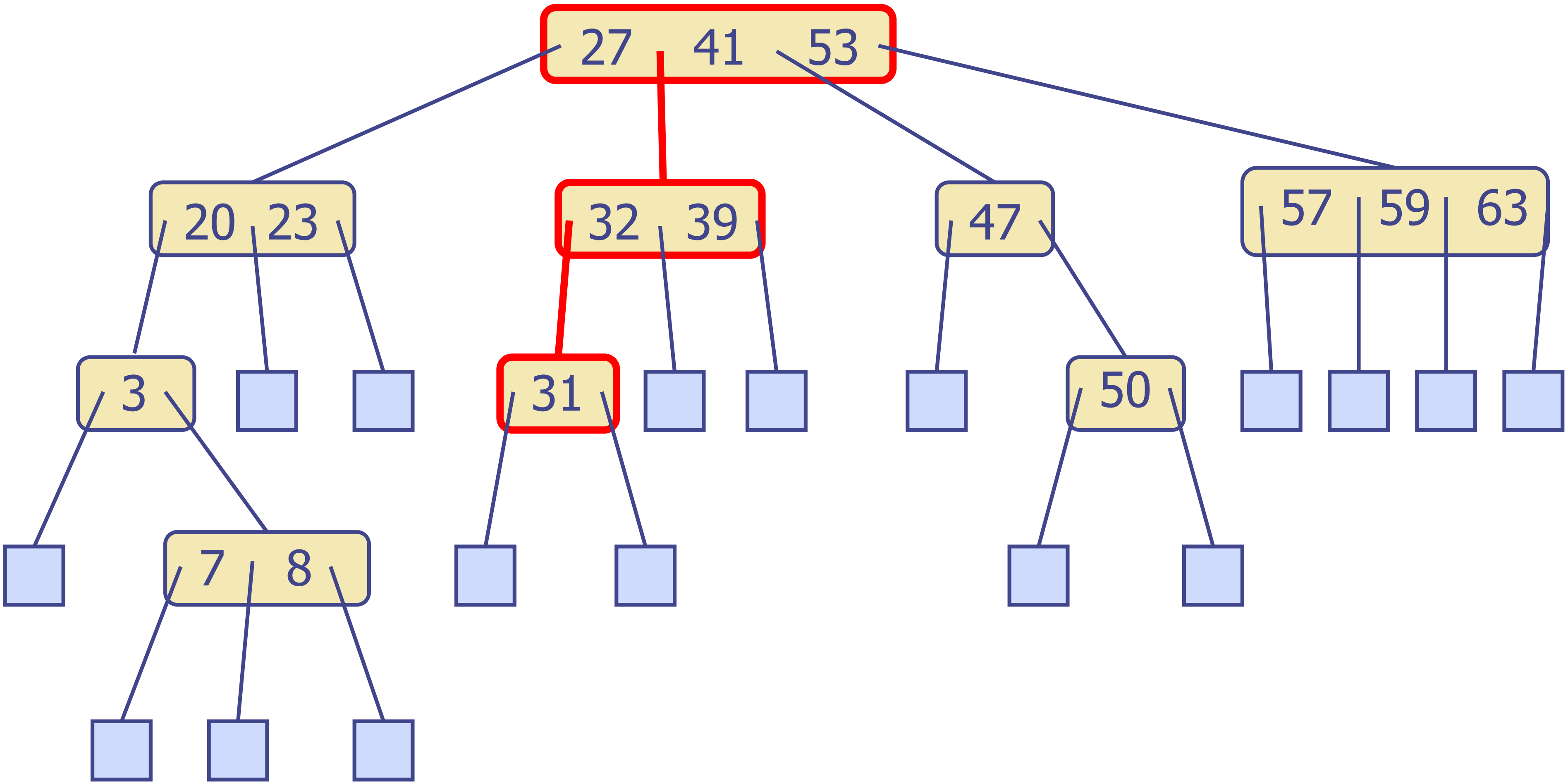


Data Structures for Multi-Way Search Trees



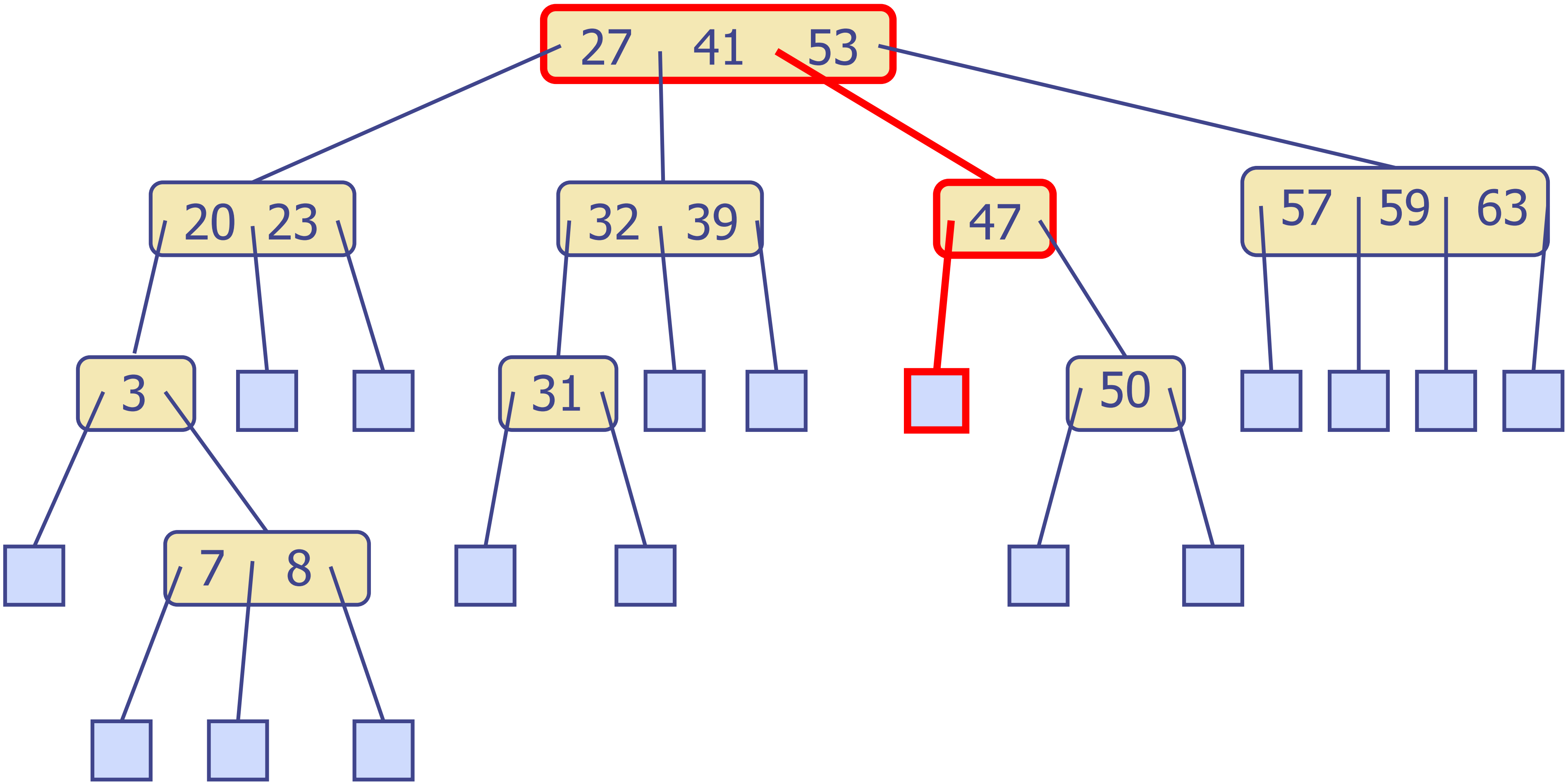
Multi-Way Searching

- Similar to search in a binary search tree
- Example: search for 31



Multi-Way Searching

- Similar to search in a binary search tree
- Example: search for 46



Multi-Way Searching

Algorithm get(r,k)

In: Root r of a multiway search tree, key k
Out: data for key k or null if k not in tree

if r is a leaf **then return** null

else {

Use binary search to find the index i such that either

- r.keys[i] = k, or
- r.keys[i] < k < r.keys[i+1]

if k = r.keys[i] **then return** r.data[i]

else return get(r.child[i],k)

}

Multi-Way Searching

Algorithm get(r,k)

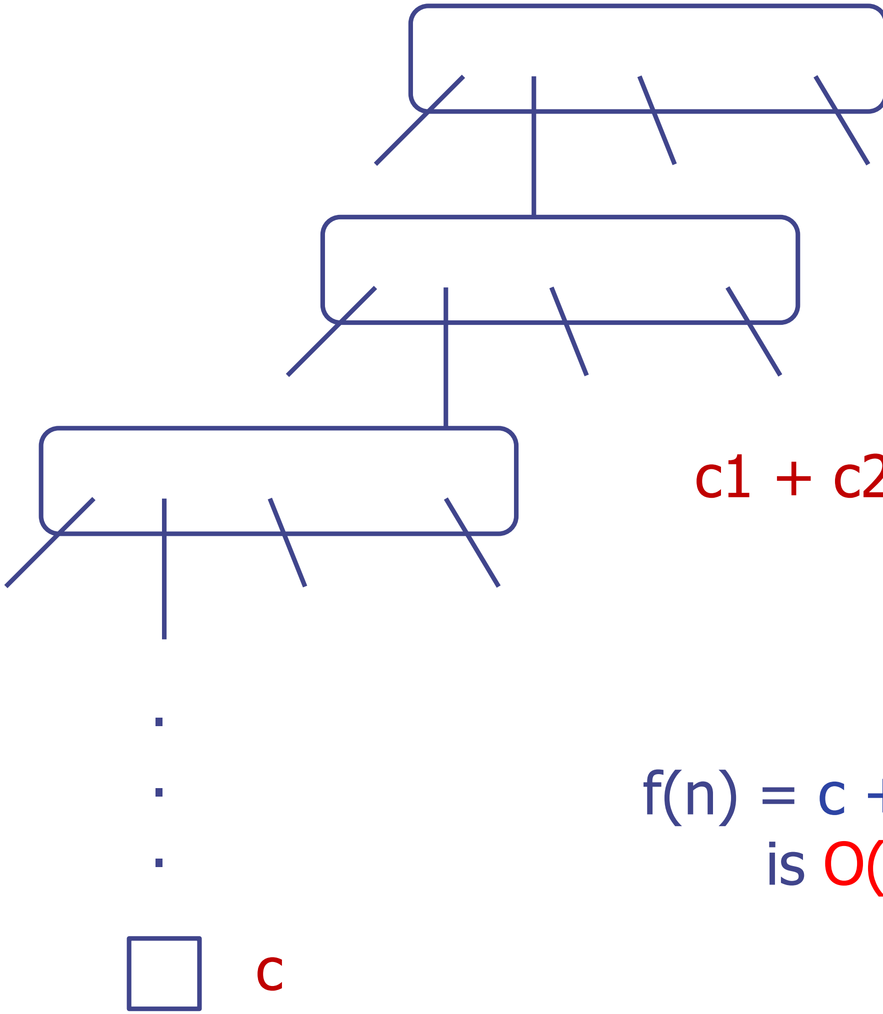
In: Root r of a multiway search tree, key k
Out: data for key k or null if k not in tree

```
if r is a leaf then return null
else {
    Use binary search to find the index i such that either
        • r.keys[i] = k, or
        • r.keys[i] < k < r.keys[i+1]
    if k = r.keys[i] then return r.data[i]
    else return get(r.child[i],k)
}
```

c operations

Ignoring recursive calls: $c_1 \log d + c_2$ operations

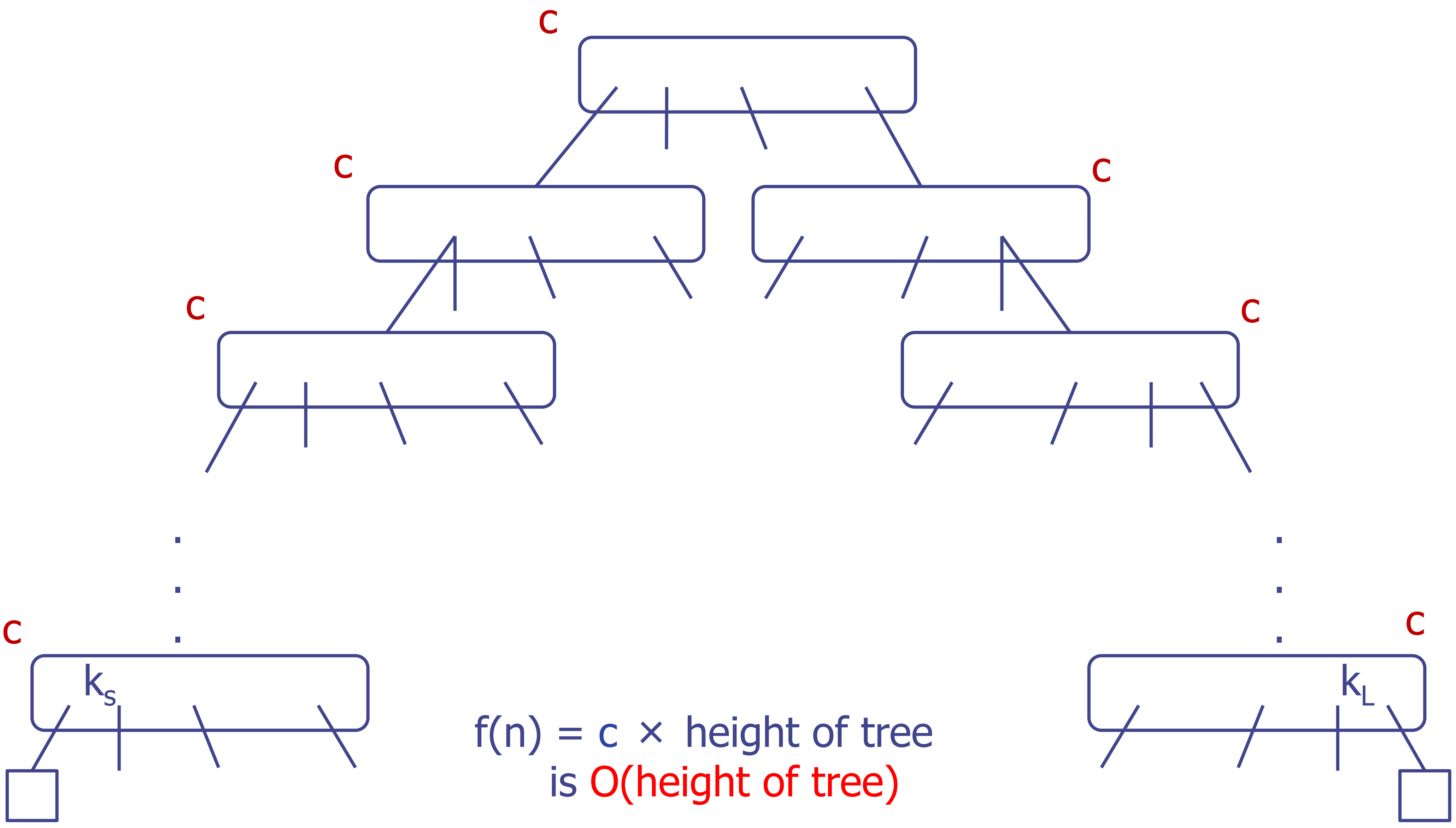
Time Complexity of get Operation



$f(n) = c + (c1 + c2 \log d) \times$ height of tree

is $O(\log d \times$ height of tree$)$

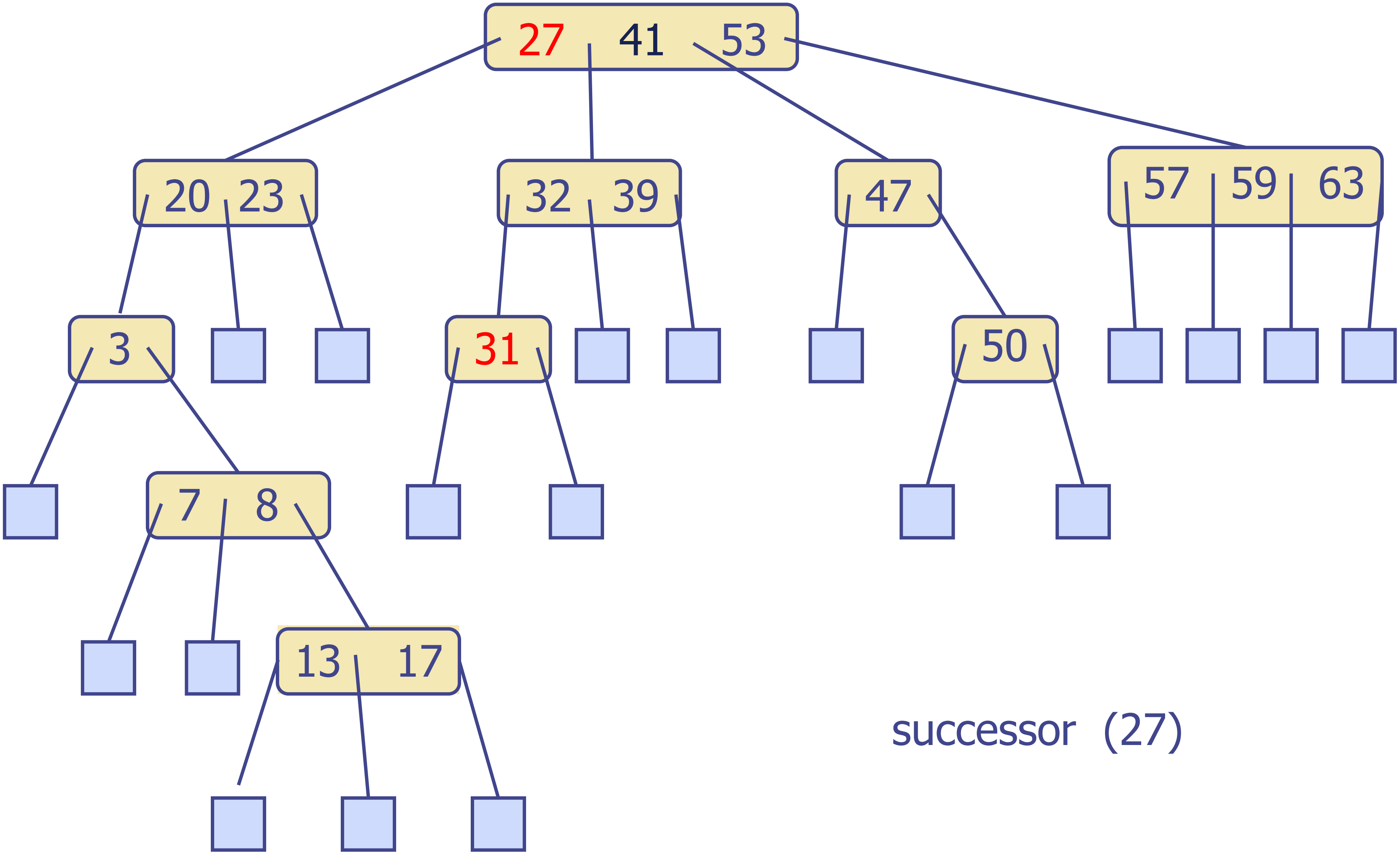
Smallest and Largest Operations



$f(n) = c \times$ height of tree

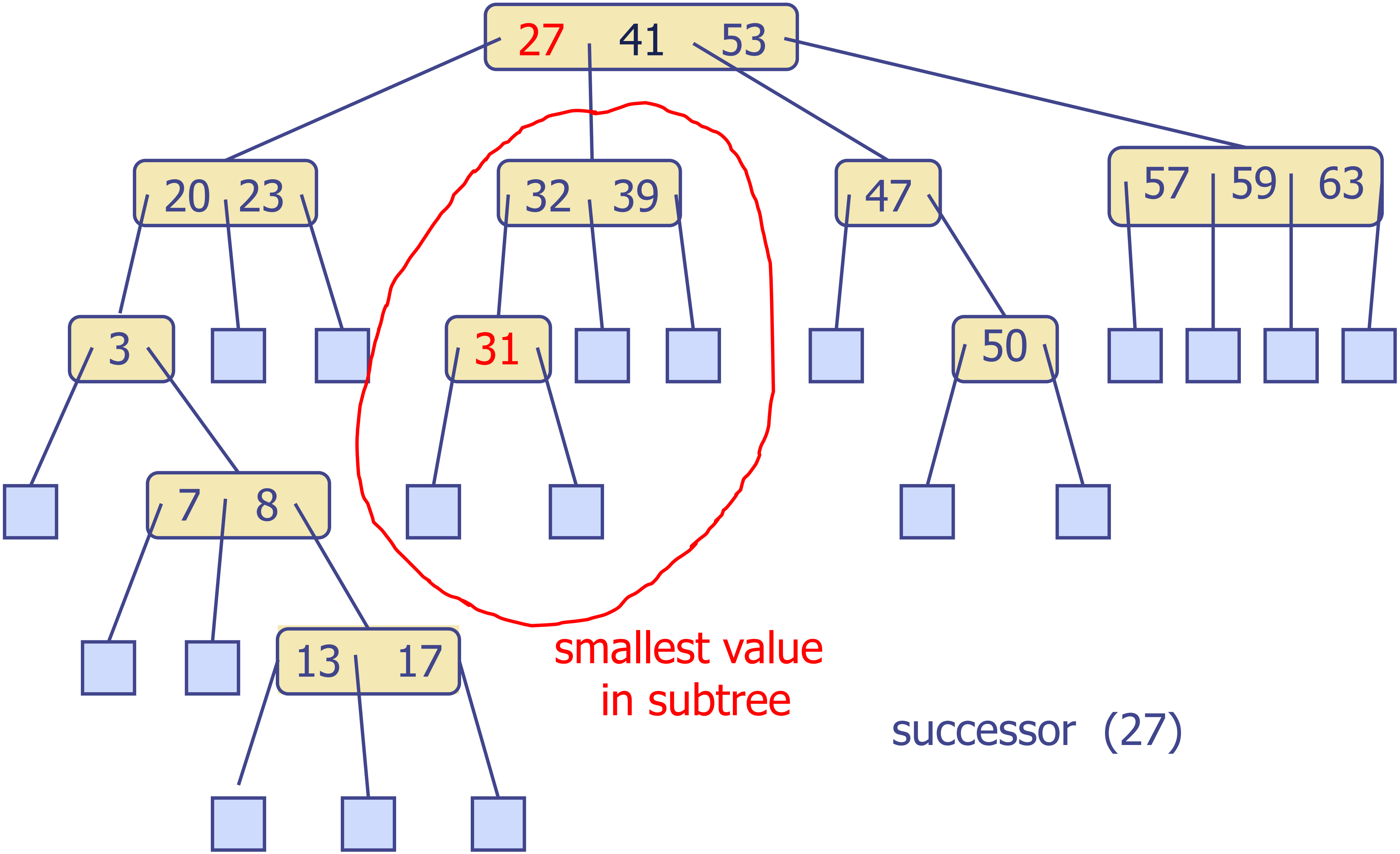
is O(height of tree)

Successor Operation

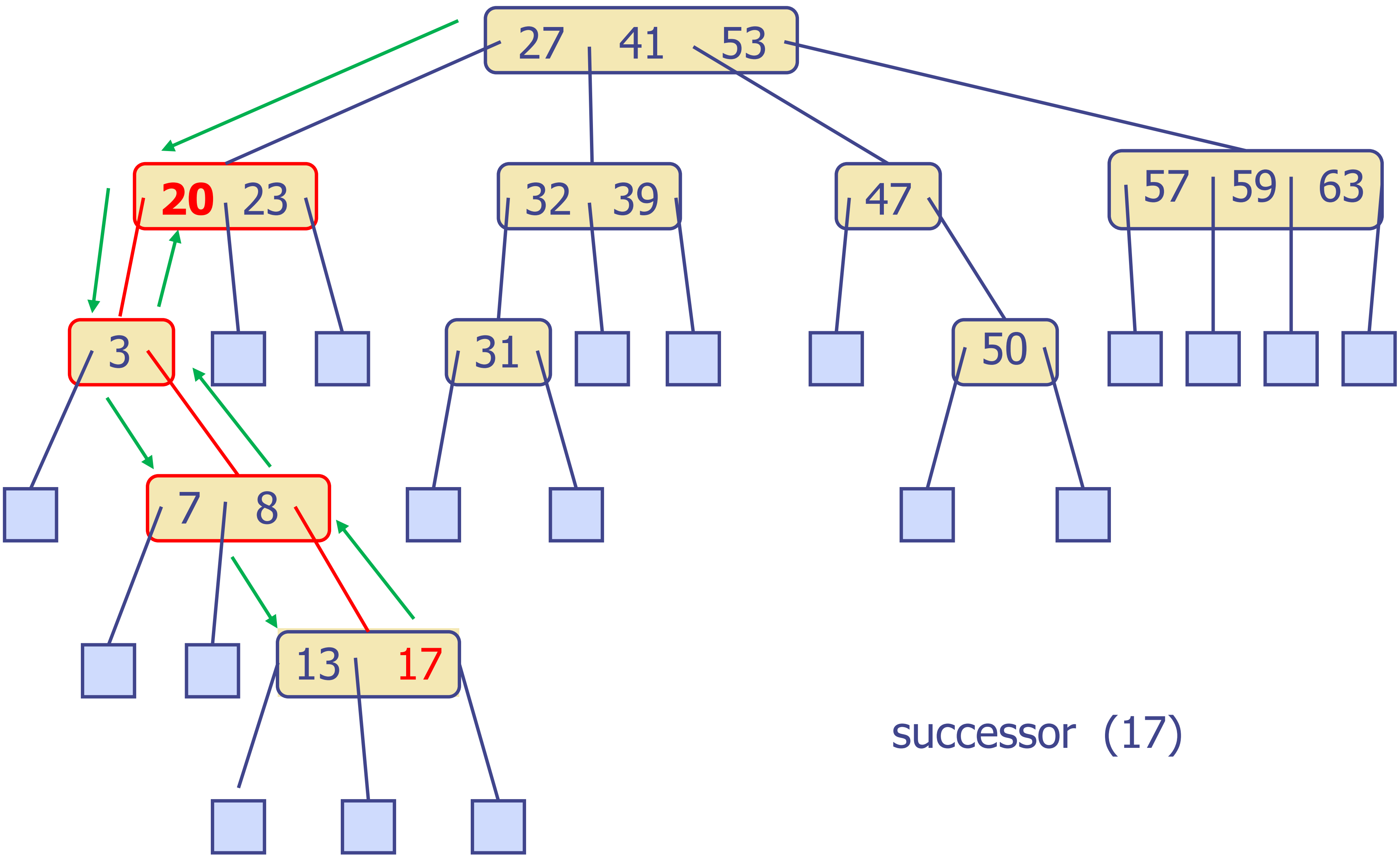


successor (27)

Successor Operation

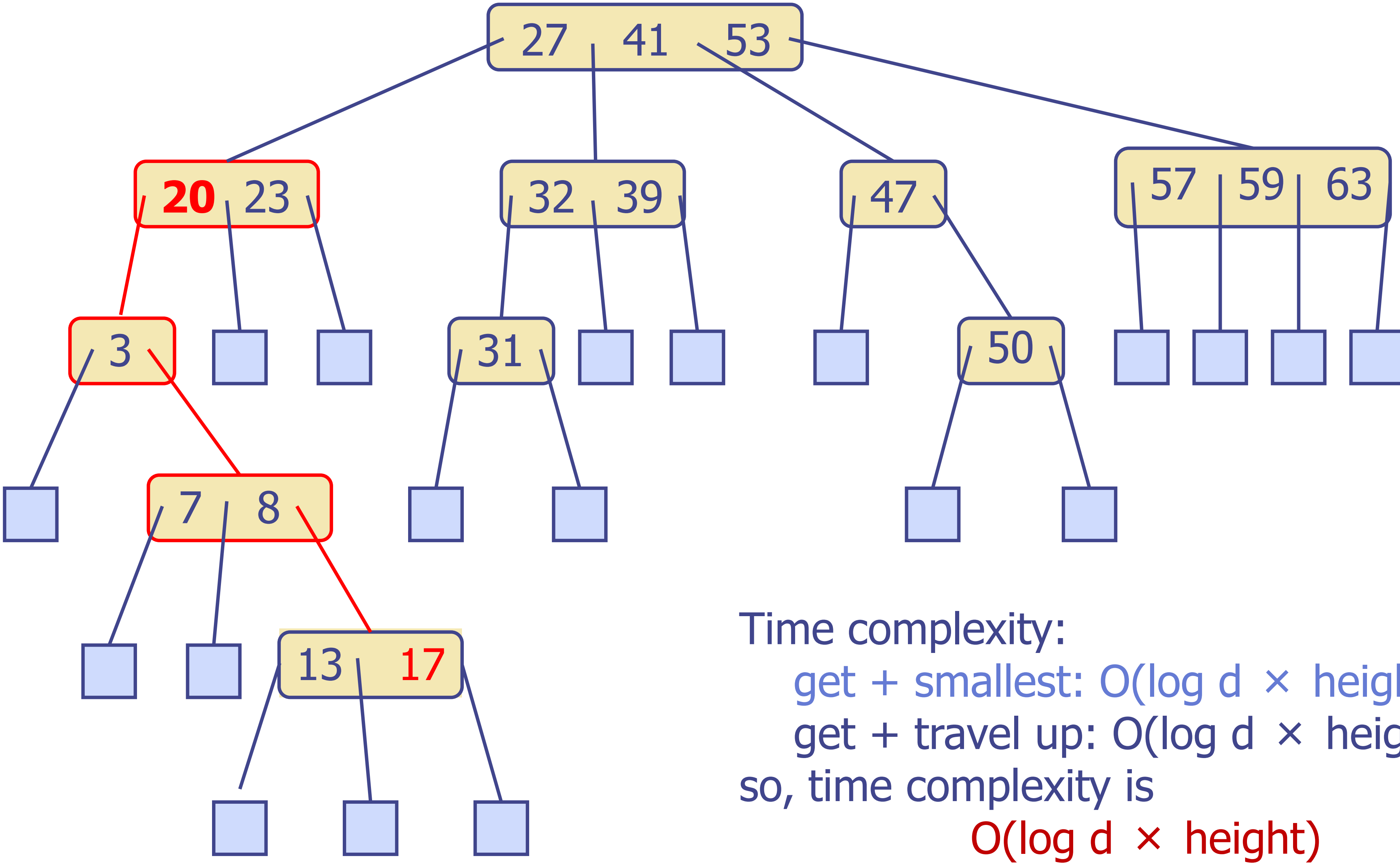


Successor Operation



successor (17)

Successor Operation



Time complexity:

get + smallest: O(log d × height), or

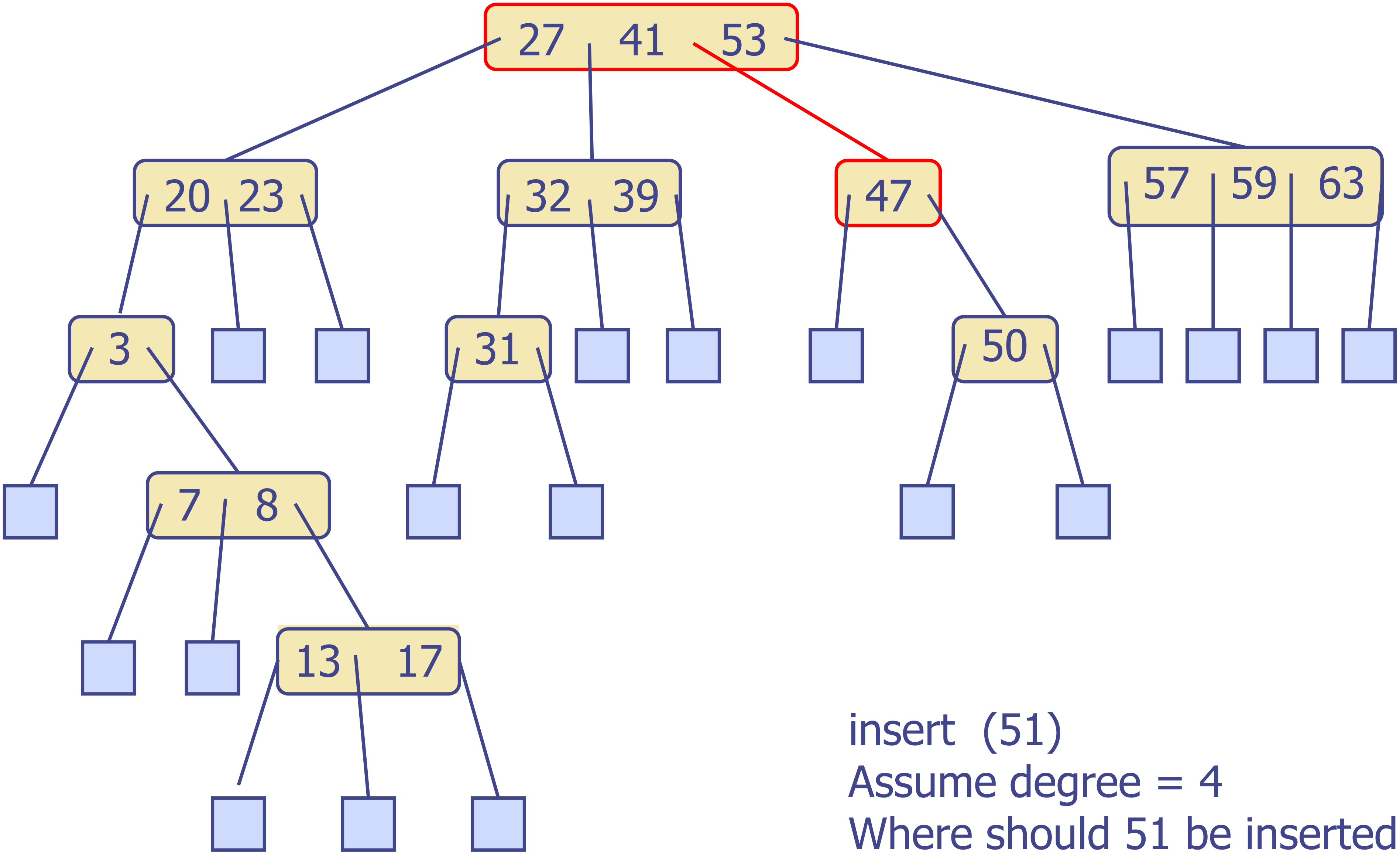
get + travel up: O(log d × height)

so, time complexity is

O(log d × height)

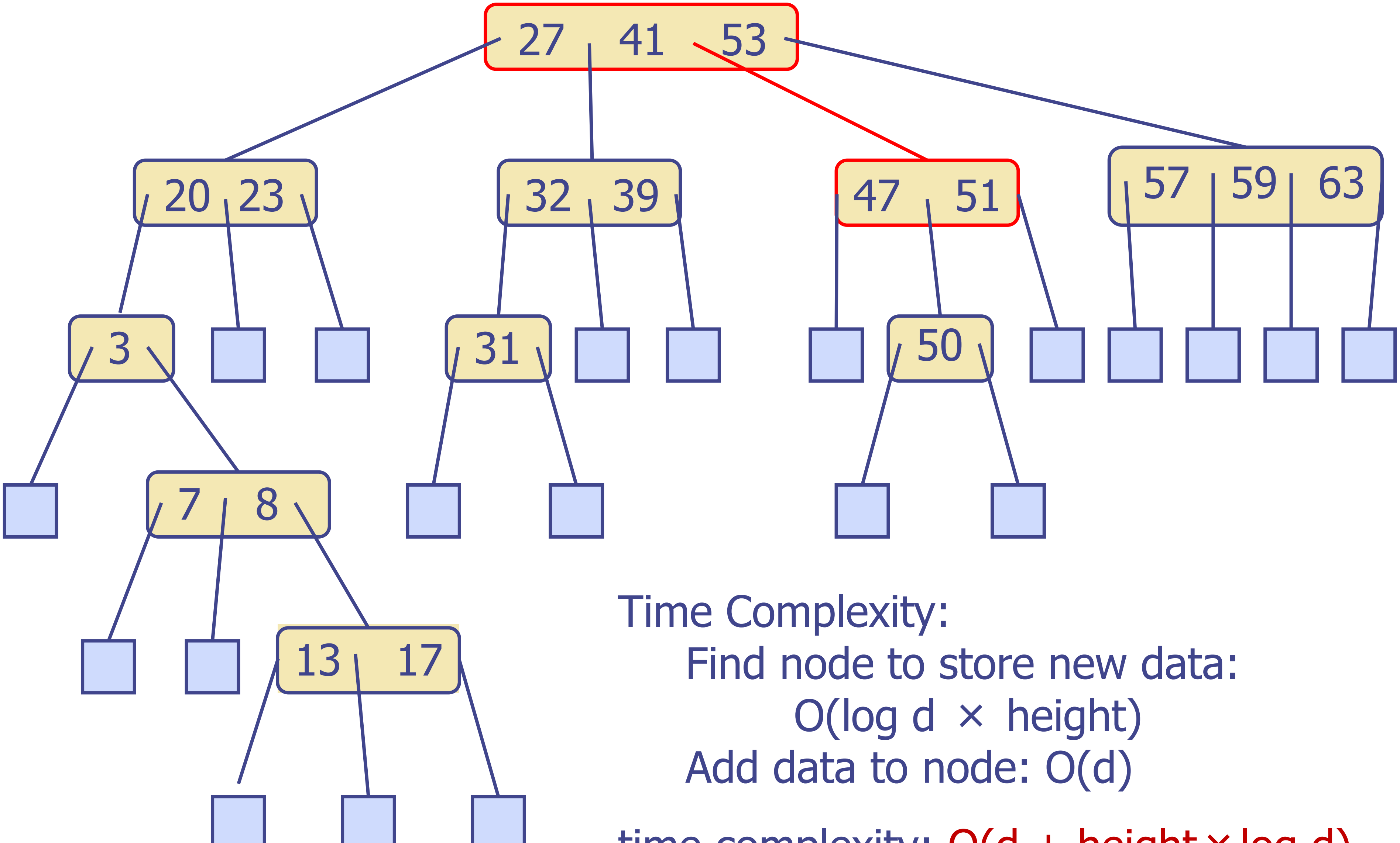
```
Algorithm successor(r,k)
In: Root r of a multiway search tree, key k
Out: Successor of k, or null if k has no successor
    p ← get(r,k)
    Use binary search to find index i such that p.keys[i] = k
    if p.children[i] is an internal node then
            return smallest(p.children[i])
     else if p.keys[i] is not the last key in p then return p.keys[i+1]
            else {
                p ← parent of p
                while p ≠ null do {
                    if p has a key k' > k then return k'
                    else p ← parent of p }
                }
                return null
            }
```

Put Operation



insert (51)
Assume degree = 4
Where should 51 be inserted?

Put Operation



Time Complexity:

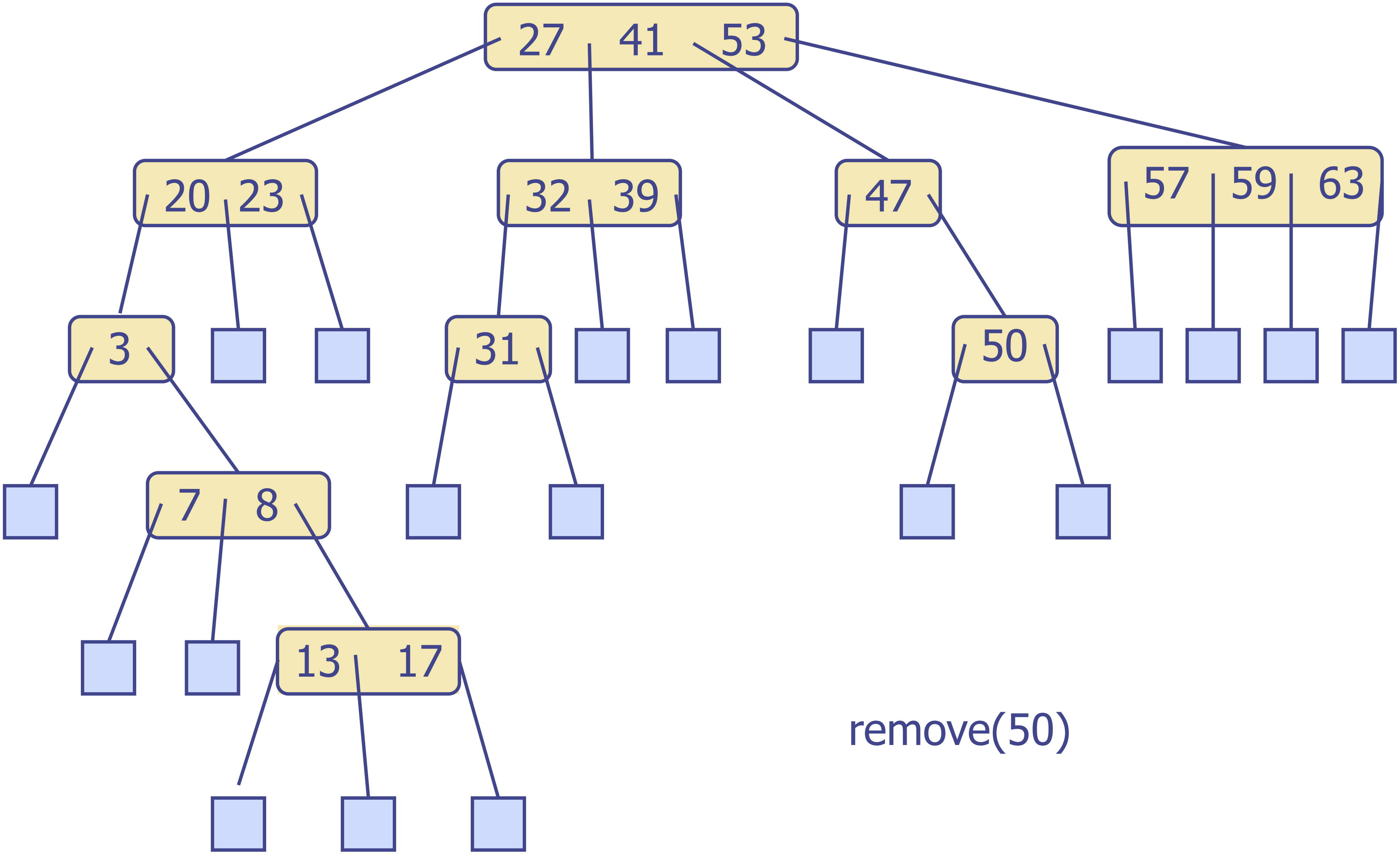
Find node to store new data:

O(log d × height)

Add data to node: O(d)

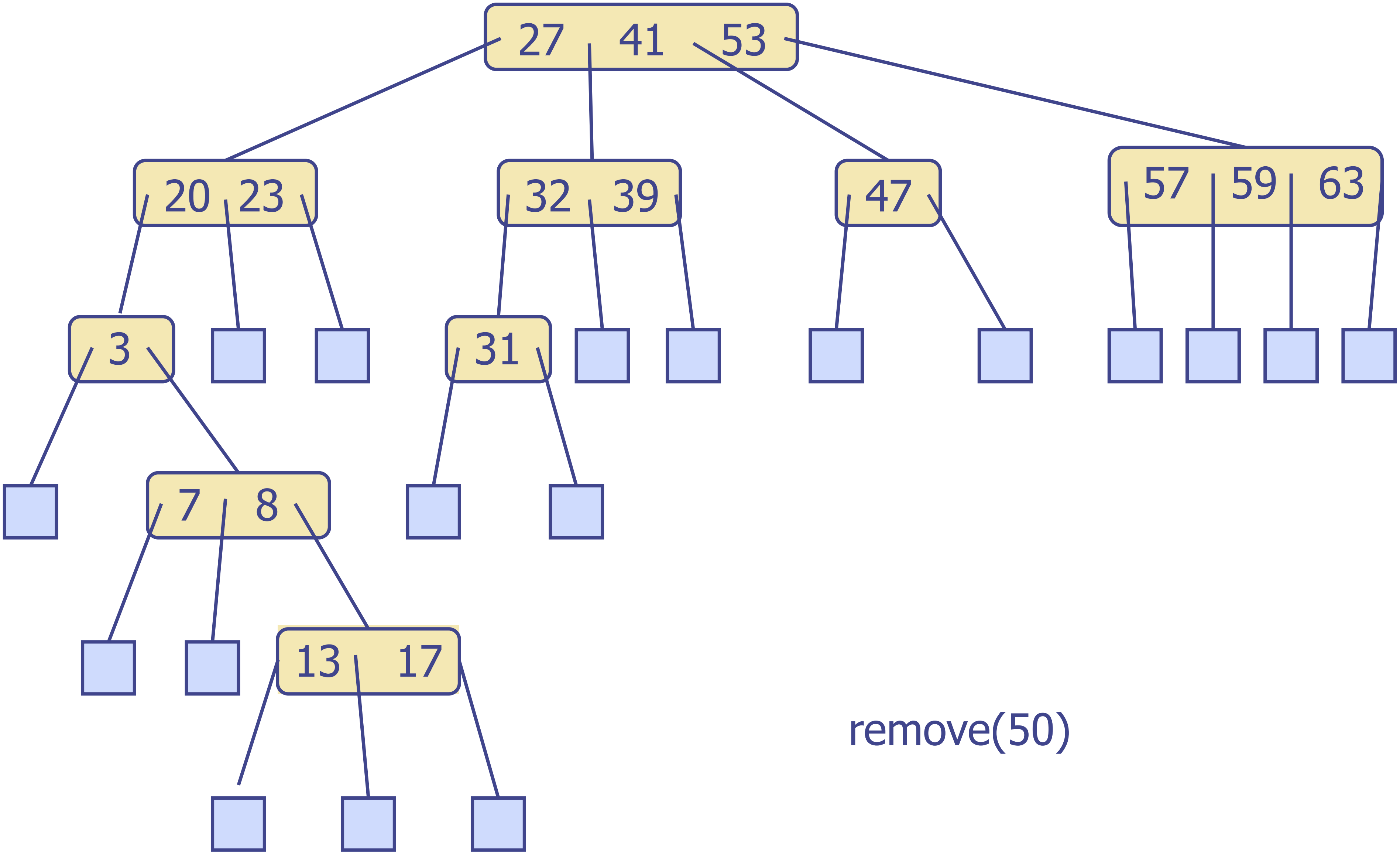
time complexity: O(d + height × log d)

Remove Operation



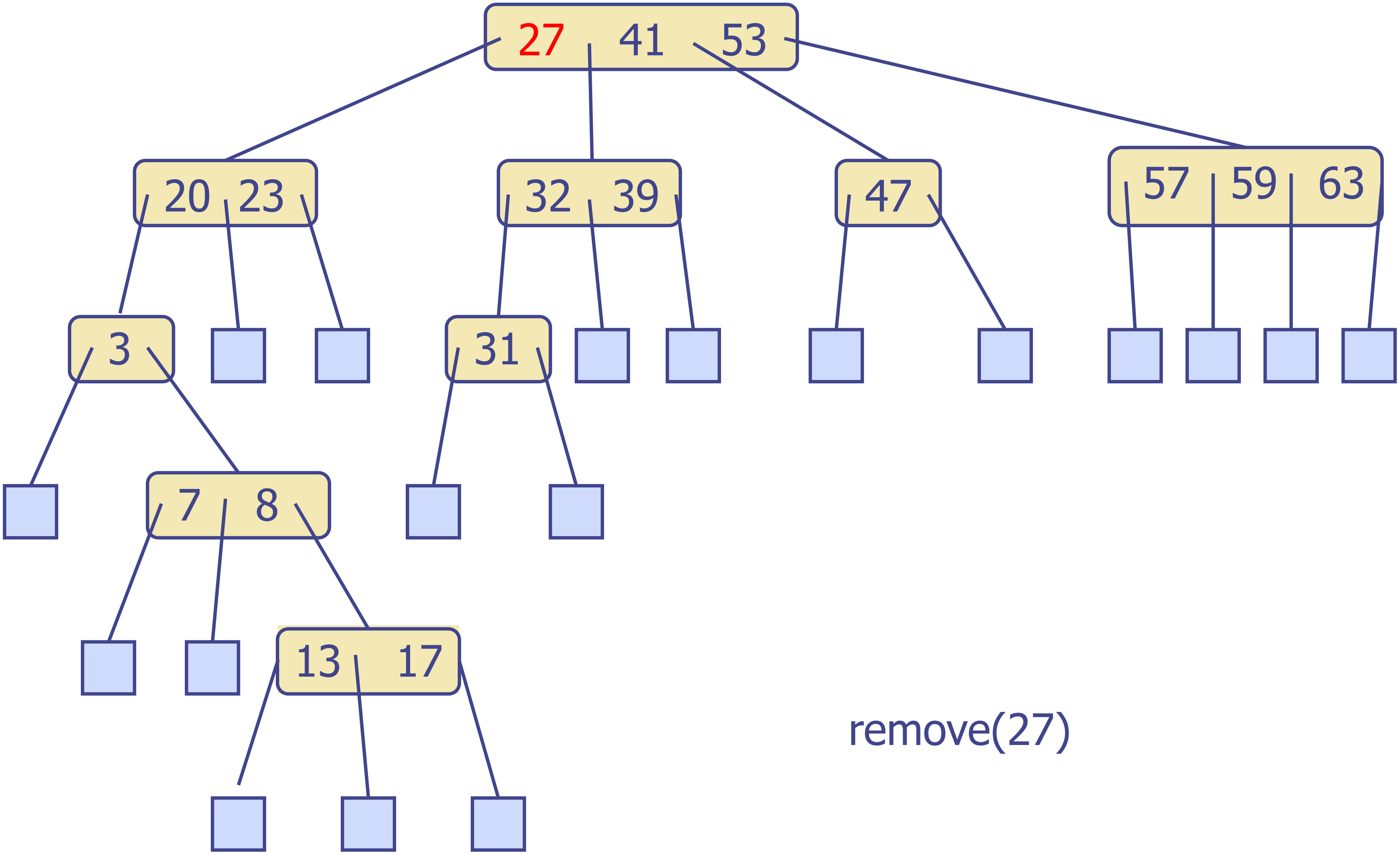
remove(50)

Remove Operation



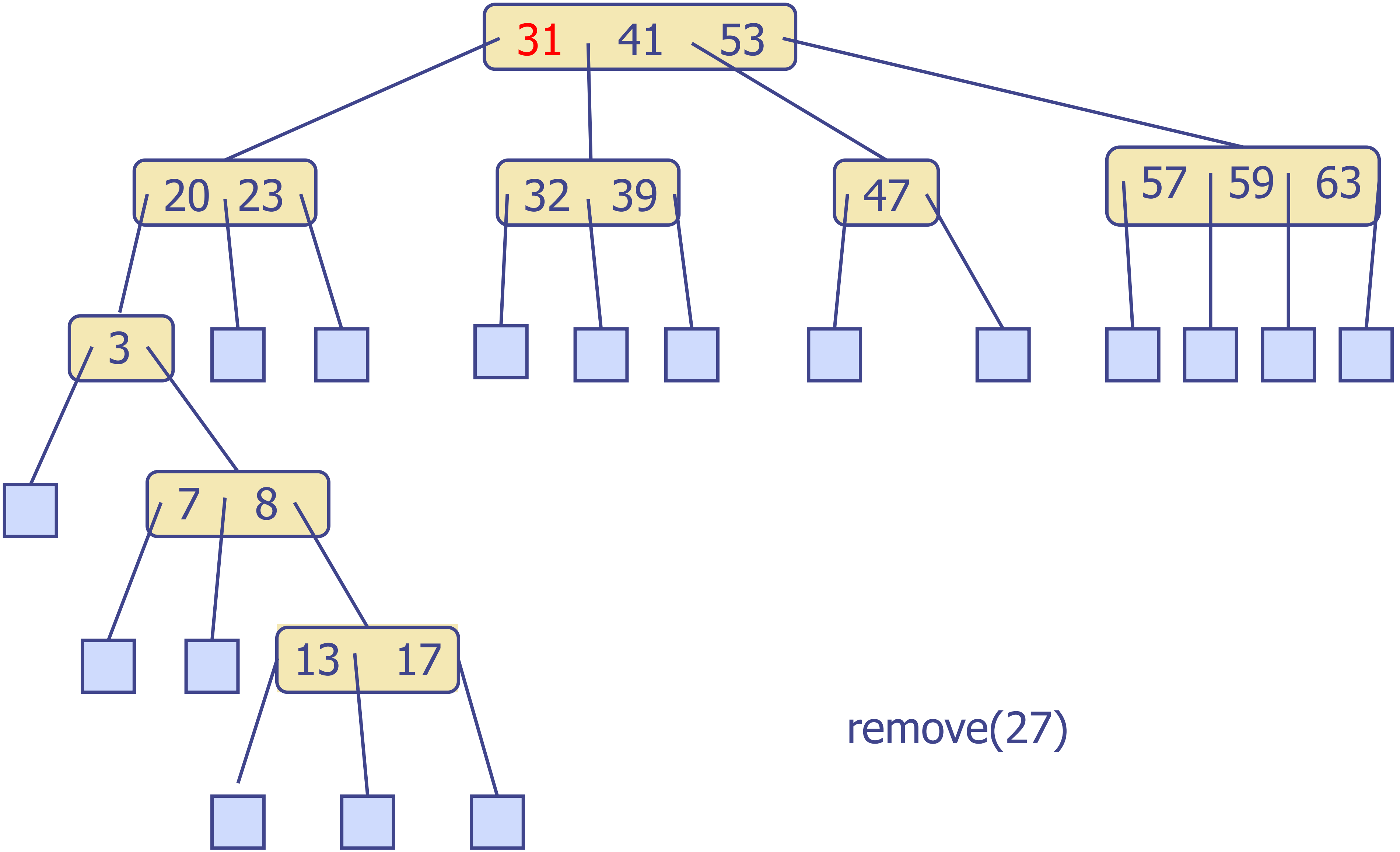
remove(50)

Remove Operation



remove(27)

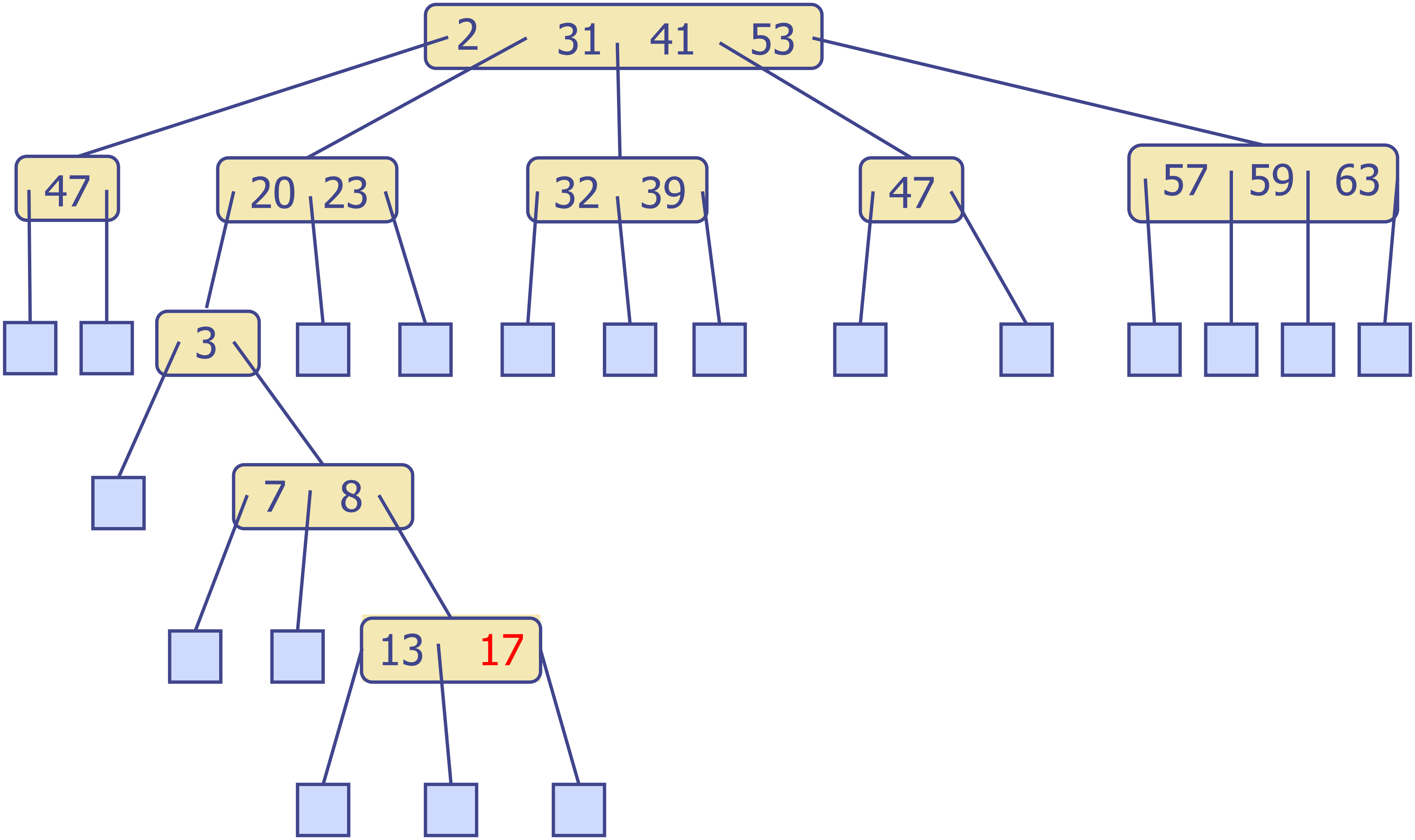
Remove Operation



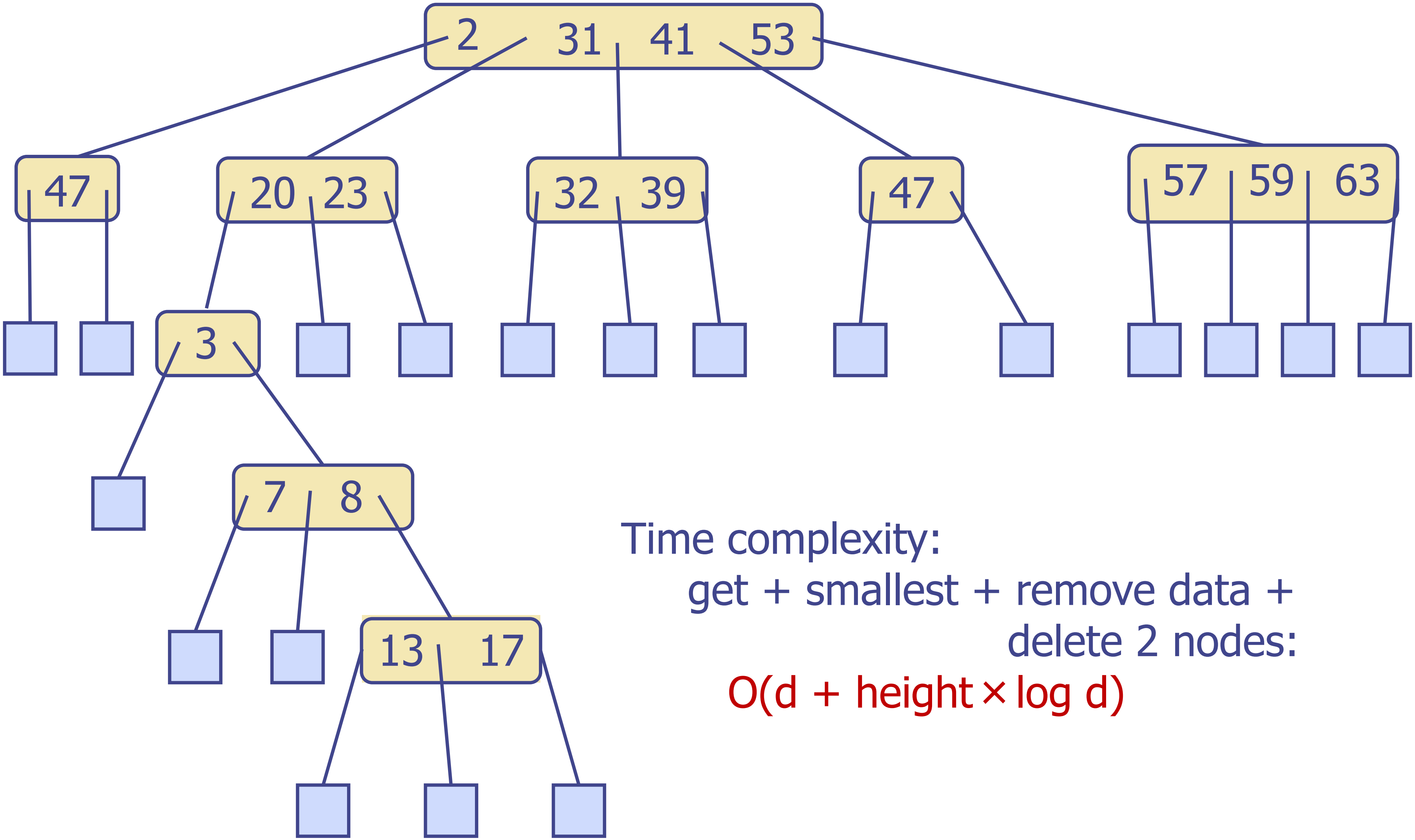
remove(27)

Remove Operation

remove(2)



Remove Operation



Time complexity:
get + smallest + remove data +
delete 2 nodes:
O(d + height × log d)

Ordered Dictionary Operations on a Multiway Search Tree of Degree d

smallest	O(height)
largest	O(height)
get	O(height $\times$ log d)
successor	O(height $\times$ log d)
predecessor	O(height $\times$ log d)
put	O(d + height $\times$ log d)
remove	O(d + height $\times$ log d)