Tree Representations

Agenda

- Positional trees
- Complete k-ary tree
- Rooted Tree representations
  - Single array representation
    - Complete Binary Tree
  - Linked List representation

Positional Trees

- By extending the positioning information that distinguishes binary trees from ordered trees to **trees with more than 2 children per node**.
- ***Positional tree***, the children of a node are labeled with **distinct positive integers**. The ith child of a node is ***absent*** if no child is labeled with integer i.
- A **k-*ary* tree** is a positional tree in which for every node, all children with **labels greater than k are missing.**
- Binary tree is a **k-ary** tree with **k = 2.**

*

complete k***-ary tree***

- A ***complete*** k***-ary tree*** is a k-ary tree in which all leaves have the same depth and all internal nodes have **degree k.**
- The root has k children at depth 1, each of which has k children at depth 2, etc.
- How many leaves does a complete k-ary tree of **height h** have?
- Number of leaves at depth h is $\mathbf{k^h}$.
- Number of internal nodes of a complete k-ary tree of height h ?

Complete binary tree

- A **complete binary tree** is a binary tree in which all **leaves have the same depth** and **all internal nodes have degree 2.**
- Number of **internal nodes** in a complete binary tree ?
- Number of **Leaf nodes** ?
- **Total number of nodes (n)** in a complete binary tree of height h, $2^{h+1} - 1$
- What is the **height of a complete binary tree** with n number of nodes?

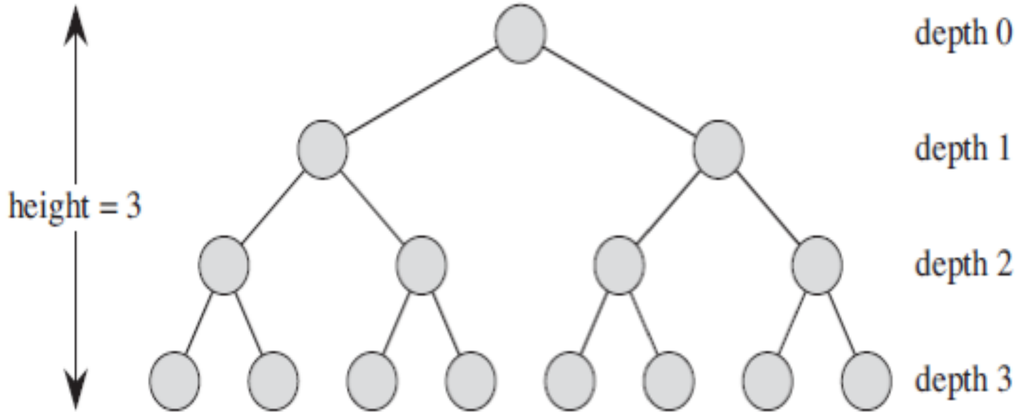


Figure B.8 A complete binary tree of height 3 with 8 leaves and 7 internal nodes.

Rooted tree representations

Representation of Rooted Trees

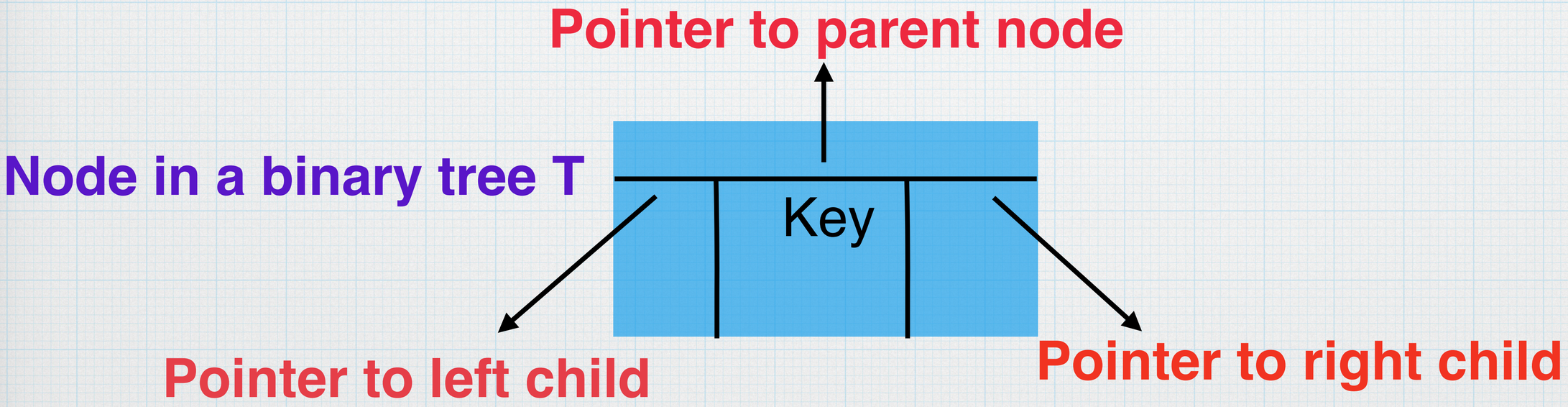
- Representing Binary trees by **linked data structures**
- Representing Rooted trees by **linked data structures**,
  - Rooted trees - Nodes having **arbitrary number of children**

Node in a Rooted Tree

- Represent **each node** of a tree **by an object**
  - Each **node** contains a **key attribute**
  - **Pointers** to other nodes (vary according to the type of trees)

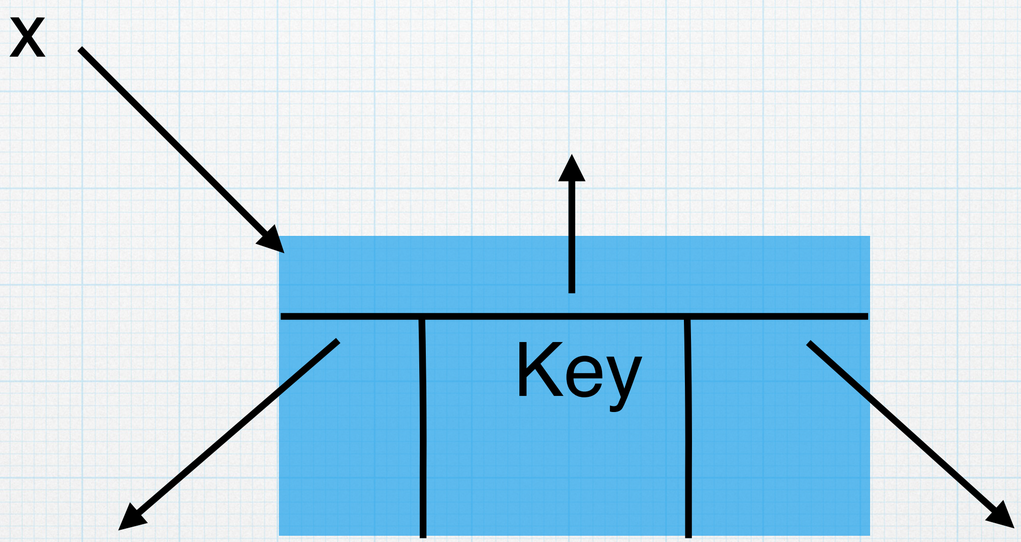
Binary Trees

- Represent **each node** of a tree **by an object**
  - Each **node** contains a **key attribute**
  - **Pointers** to **Parent node, left child and right child**



Special nodes in the Binary tree

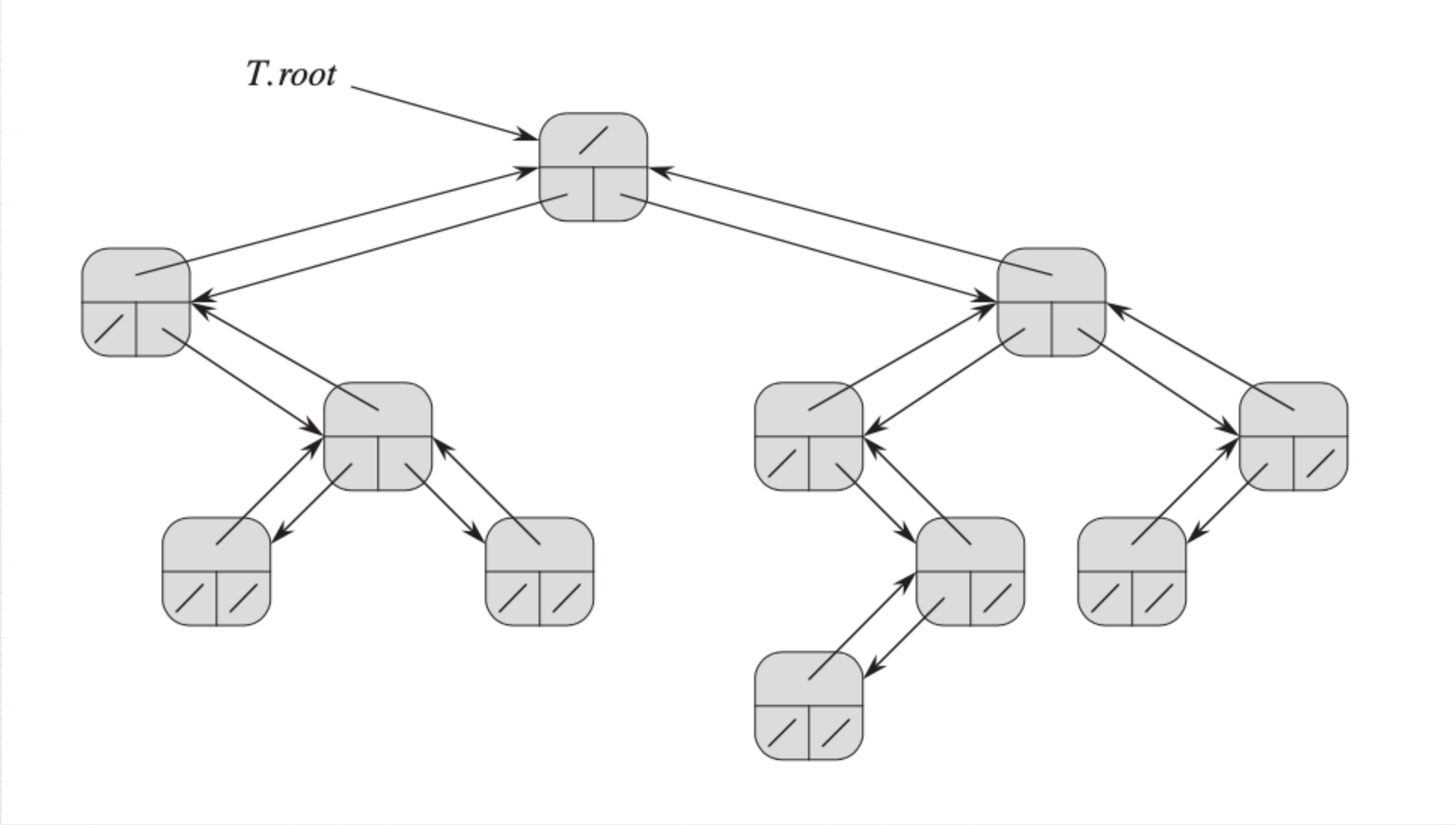
- Node x
- x.p = NIL ——> ?
- x.left = NIL
- x.right = NIL
- If both x.left and x.right are NIL, then ?



Root of the Binary Tree

- The **root of the entire tree T** is pointed to by the attribute **T.** ***root*****.**
- If **T.** ***root*** **= NIL**, then the **tree is empty.**

Representation of a Binary Tree

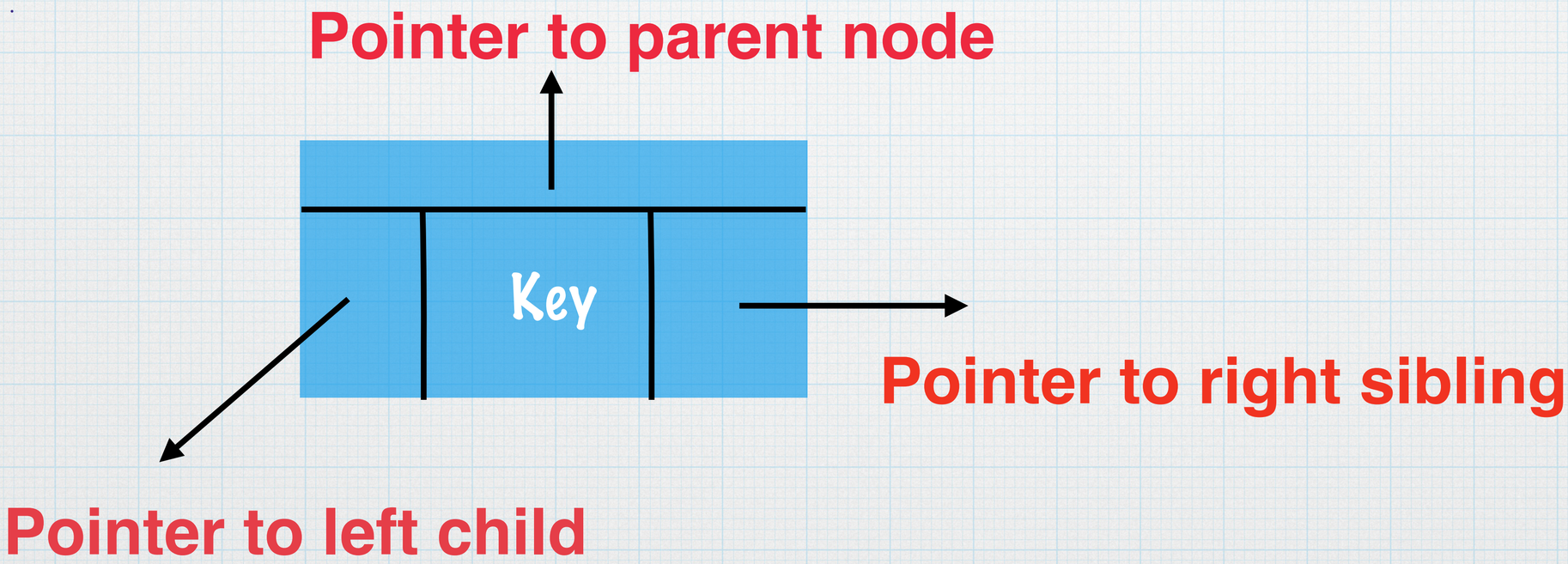


Rooted trees with bounded branching

- Extend the scheme of **representation of a binary tree** to **any class** of trees
- Trees in which the **number of children of each node** is **at most constant k**
- Replace the *left* and *right* attributes by $child_1, child_2, \ldots, child_k$
- Whether this scheme works, if the number of children of a node is unbounded ?
- Space requirement ?
- Even if the number of children k is bounded by a large constant but most nodes have a small number of children - waste of memory

Left-child, right-sibling representation

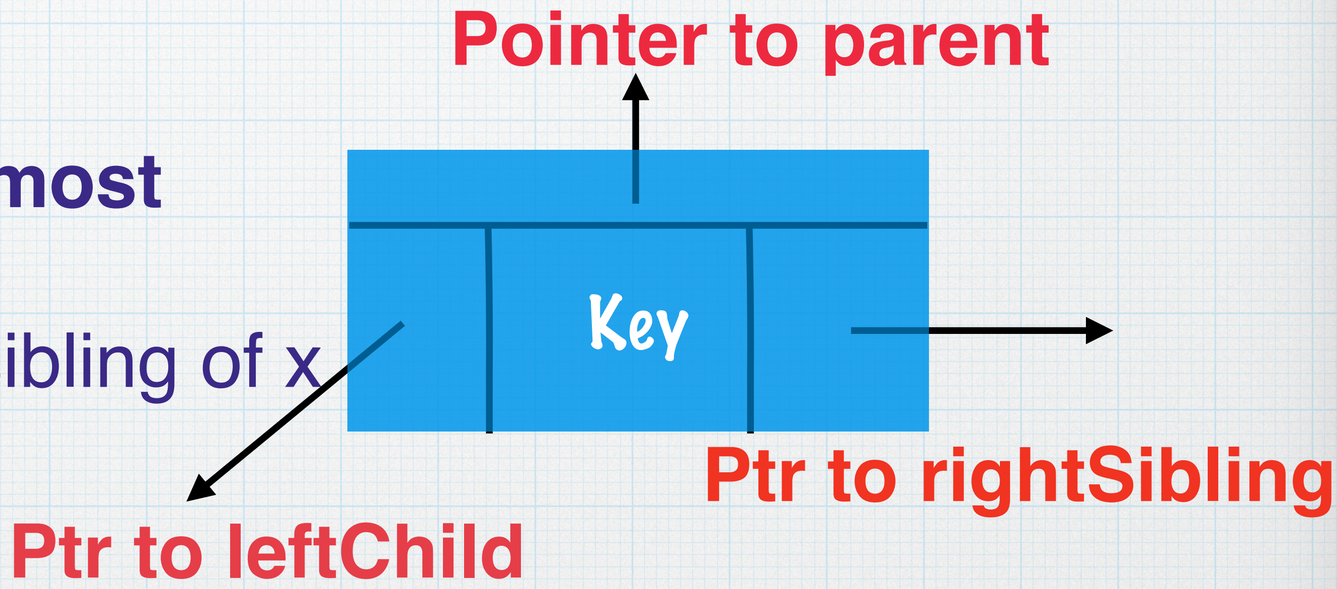
- Scheme to represent trees with arbitrary numbers of children
  - **The left-child, right-sibling representation**
- Each node contains a parent pointer p, and **T.*root*** points to the root of tree T.



Left Child, Right Sibling Representation

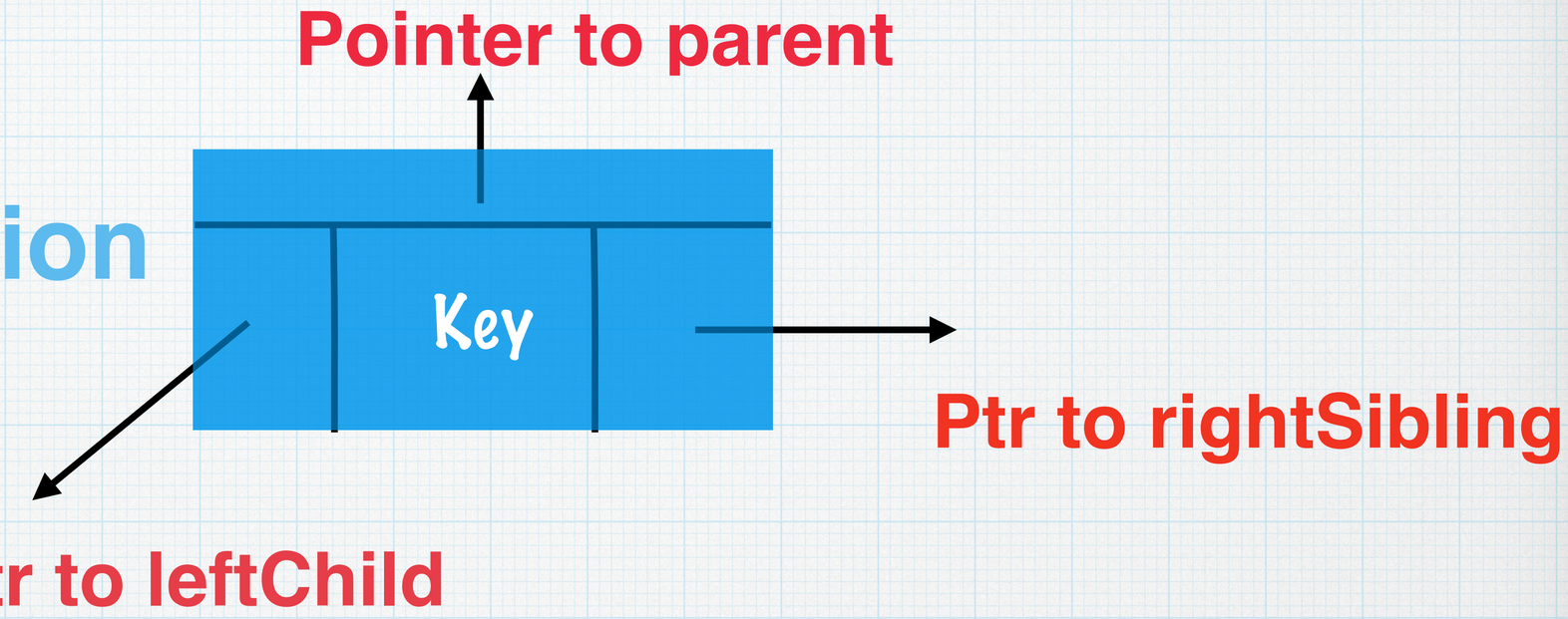
- Instead of having a pointer to each of its children, each **node x** has only **two pointers**:

- 1. **x.*leftChild*** points to the **left most child of node x**
  2. **x.*rightSibling*** points to the sibling of x immediately to its right.



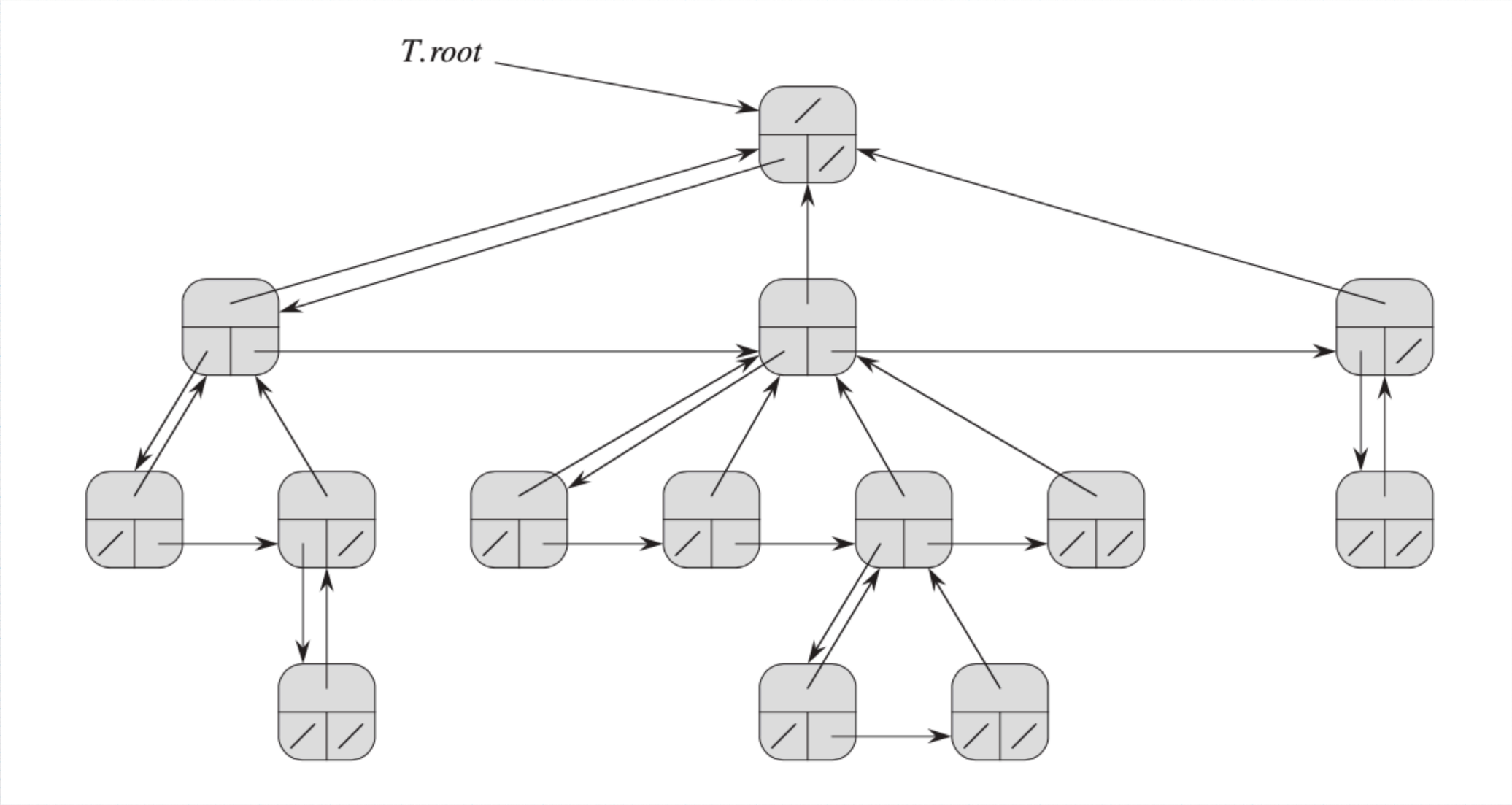
- If **node x** has no children, **x.*leftChild* = NIL**
- **I**f node x is the rightmost child of its parent, then **x.*rightSibling* = NIL**.

Left Child, Right Sibling Representation



- Node x
- x.p = NIL and x.rightSibling = NIL then x is the root node.
- If x has no children then x.leftChild = NIL
- If x is right most child of its parent then x.rightSibling = NIL

Left-child and Right-Sibling Representation



Implementation Details

```
struct node
{
        elemType Key;
        struct node *p;
        struct node *left;
        struct node *right;
};
```

```
struct binaryTree
{
 struct node *root;
};
```

Space requirement for Left-child right-sibling representation

- Only O(n) space for any n-node rooted tree

Reference

* CLRS Book