Advanced Multi-Way Search Trees

2-3 Trees, 2-3-4 Trees, RB Trees & B-Trees

Prof. Young Park

M-way Search Trees

Trees

- **Tree** is an acyclic, connected, undirected graph.

  - **Binary Trees**
  - **M-way (N-ary) Trees**

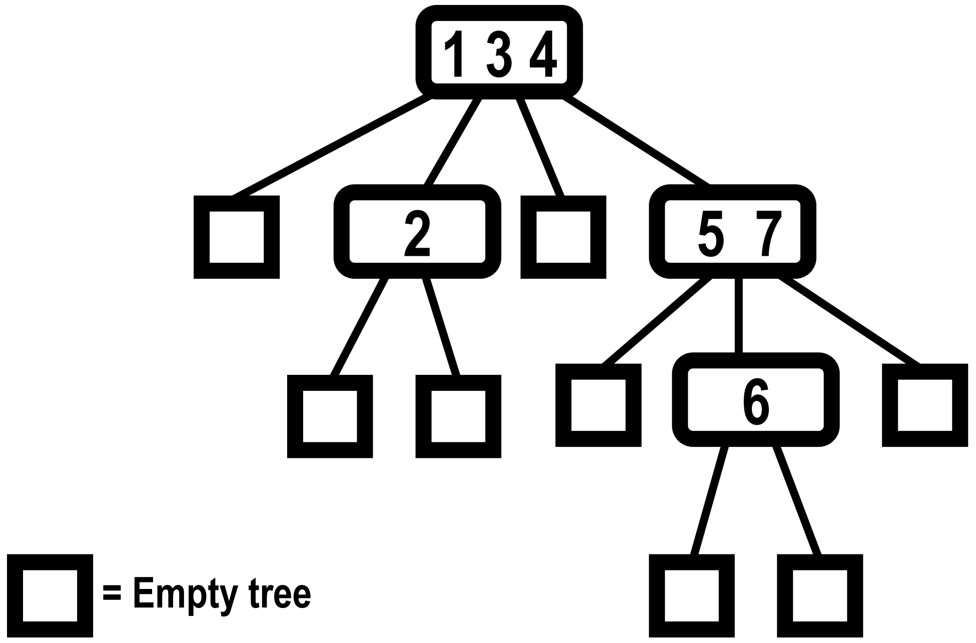
Search Trees

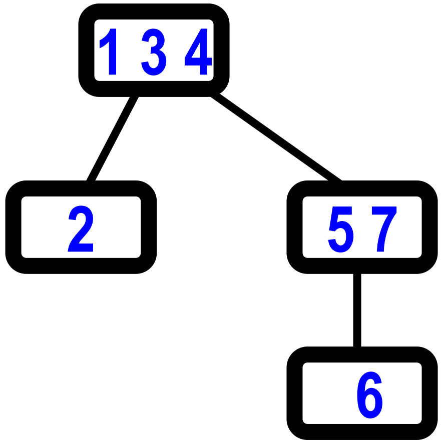
- A tree whose organization facilitates the search/retrieval of its items.
- **All data items are kept in sorted order!**
  - ➔ **Binary Search Trees**
  - ➔ **M-way Search Trees**

An M-Way Search Tree

- An **M-way search tree** is a rooted tree in which
  - Each node has **at most (m-1) sorted data items** and **m subtrees**.
    - The values in the leftmost subtree are less than the first node item.
    - The values in the second subtree are between the first node item and the second node item.
    - And so on ...
    - The values in the rightmost subtree are greater than the last node item.

Example: An M-way Search Tree with M=4





Advance M-way Search Trees

Why Advanced Search Trees?

- Why?
- The **search, insert, delete** operations on **BST and MST** are **O(n)** time at worst-case.

- **A better search, insert, delete operation?**
    - **O(log N) worst-case time**?
- Idea?

Advanced M-Way Search Trees

- How?
  - ➔ **Height Balanced** M-Way Search Trees

What Advanced M-Way Search Trees?

- **Height Balanced M-way Search Trees**
  - **2-3 trees**
  - **2-3-4 trees**
  - **B-trees**

- **Balanced Binary Search Tree**
  - **Red-Black Trees**

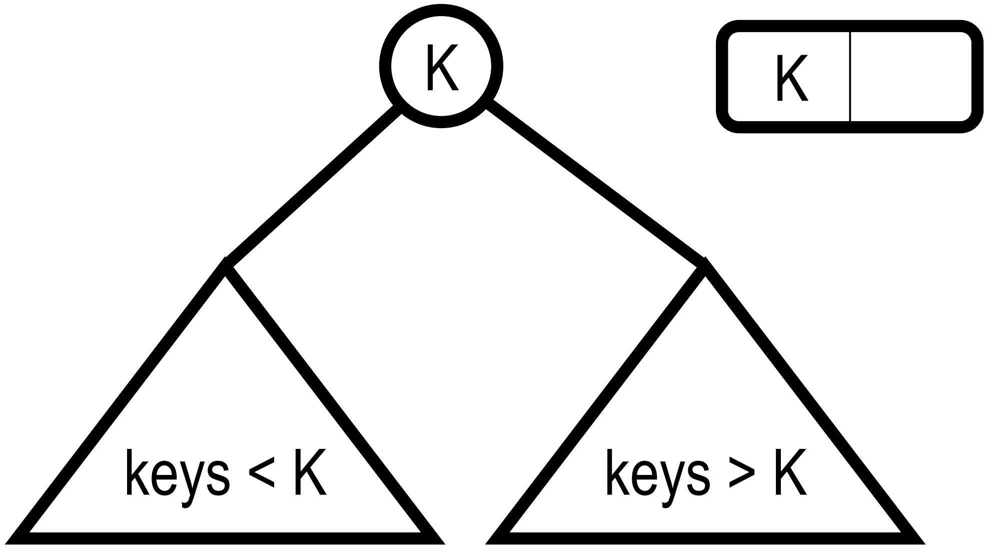
2-3 Trees

What is a 2-3 (Search) Tree?

- A **search tree** in which
  - Each nonleaf node has either two or three children. (**No single child**)
    - 2-nodes - **two** children (left & right child)
    - 3-nodes - **three** children (left, middle & right child)
  - **All leaves are at the same level (depth).**

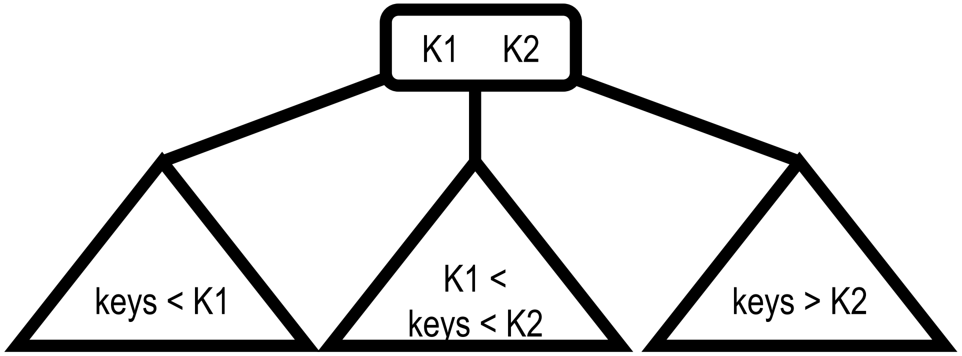
2-Node

- A 2-node must contain a single data item whose search key K
  - ➔is greater than the left child's search key(s) & less than the right child's search key(s).



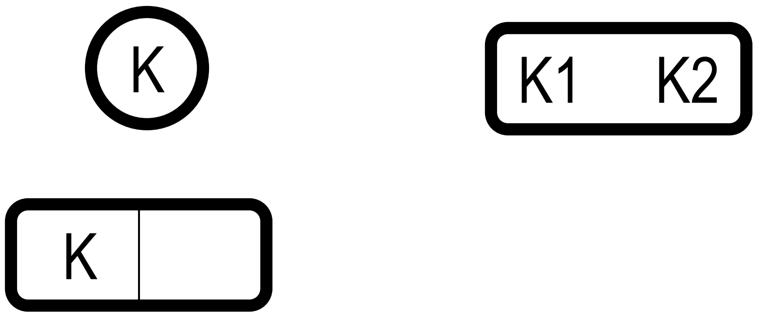
3-Node

- A 3-node must contain two data items whose search keys K1 and K2 are s.t.
  - ➔K1 is greater than the left child's search key(s) & less than the middle child's search key(s).
  - ➔K2 is greater than the middle child's search key(s) & less than the right child's search key(s).

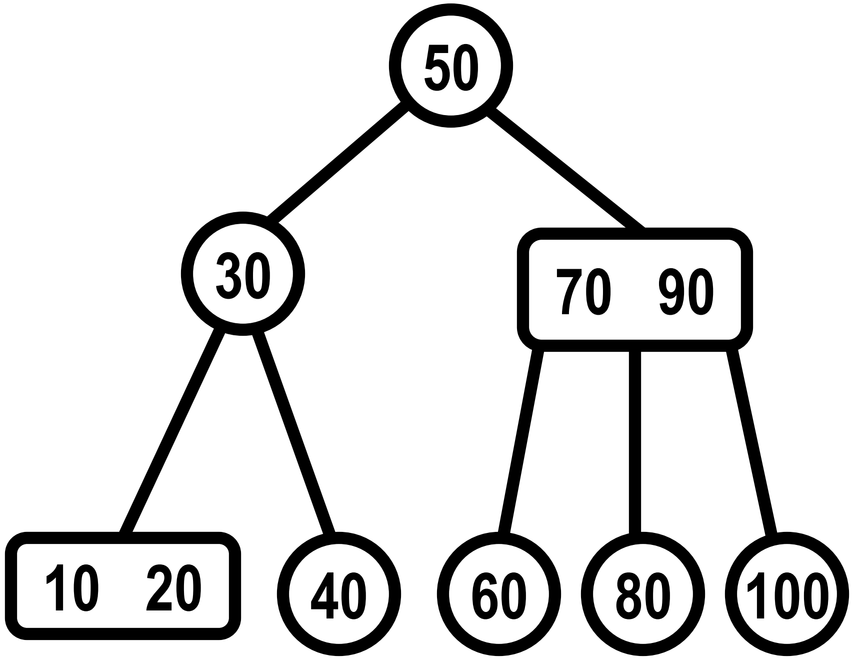


Leaf

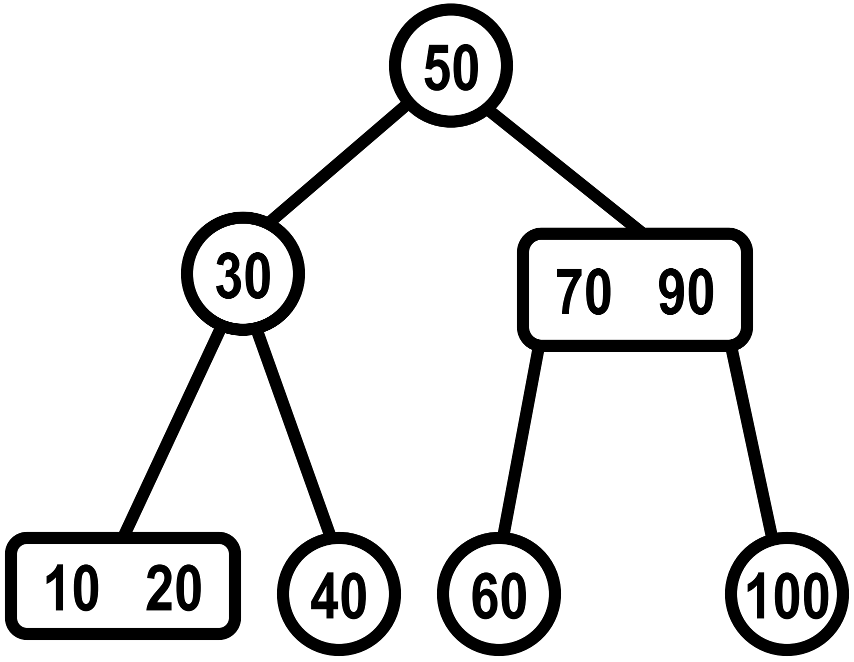
- A leaf may contain either one or two data items.



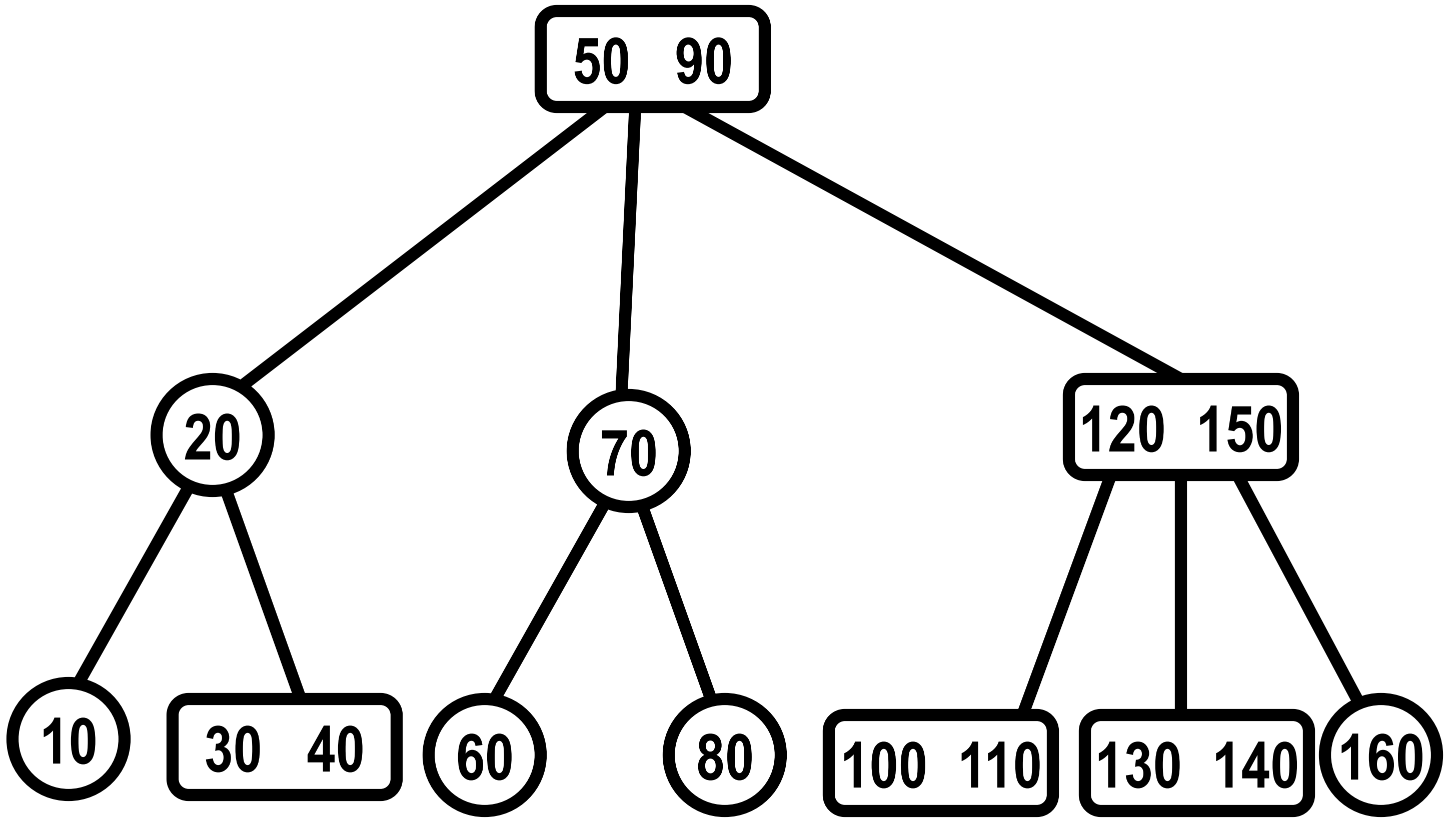
Example: A 2-3 tree?



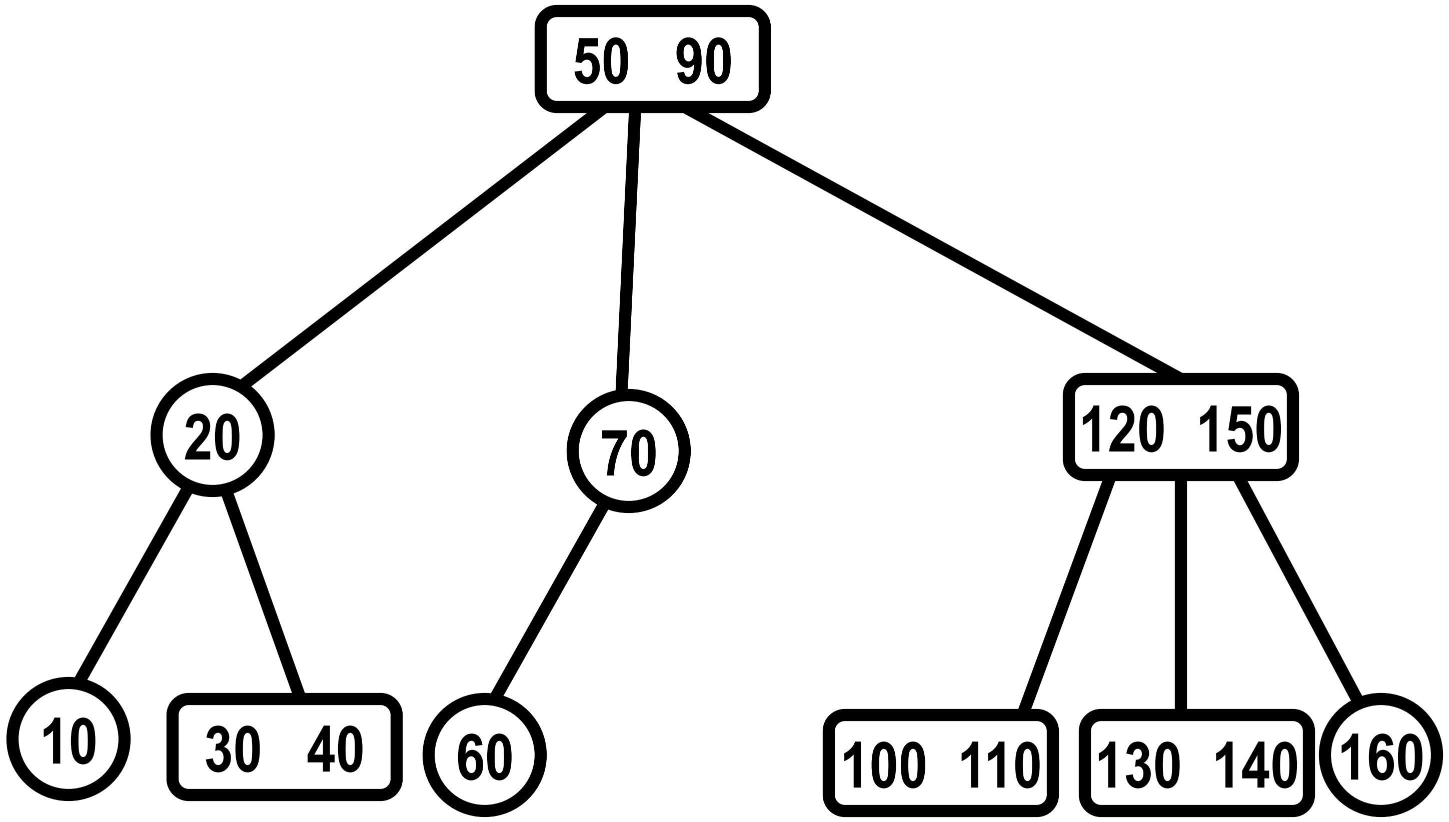
Example: A 2-3 tree?



Example: A 2-3 tree?



Example: A 2-3 tree?



Searching a 2-3 tree

- Similar to the search operation for a binary search tree.

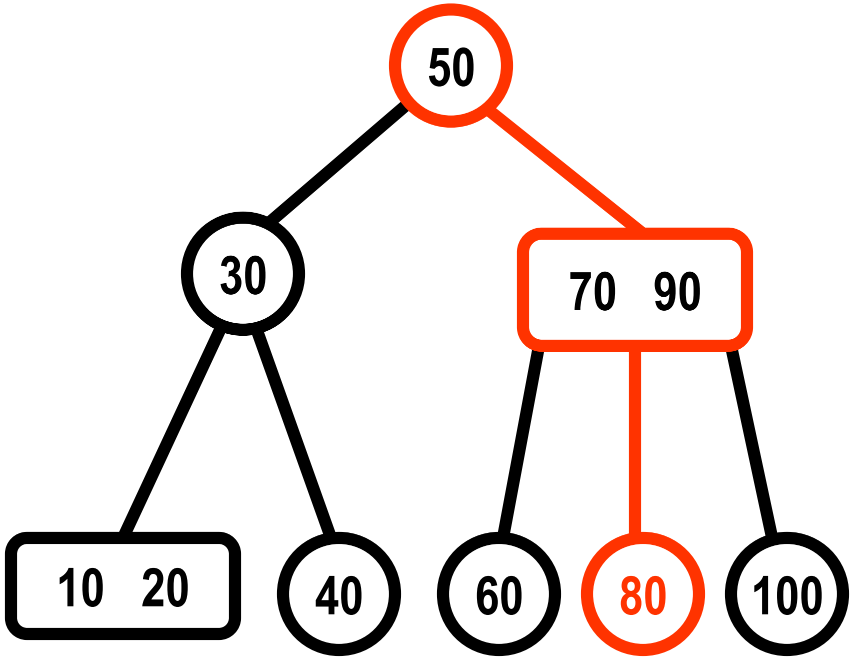
Search Operation

- Search(23T, SearchKey):
  - If SearchKey is in 23T's root node R then
    - Found (Successful search)
  - else if SearchKey is not in 23T's root node R & R is a leaf then
    - Not Found (Unsuccessful search)
  - else if R has one key then
    - If SearchKey < R's key then
      - Search (23T->LchildPtr, SearchKey)
    - else
      - Search (23T->RchildPtr, SearchKey)

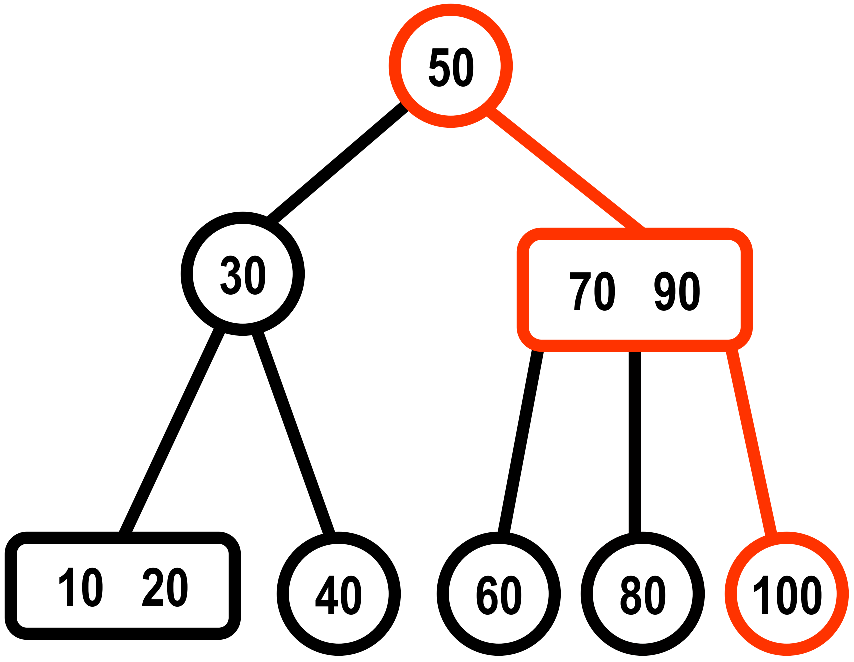
Search Operation

- else if R has two keys then
    - If SearchKey < R’s Smallkey then
        - Search (23T->LchildPtr, SearchKey)
    - else if SearchKey < R’s Largekey then
        - Search (23T->MchildPtr, SearchKey)
    - else
        - Search (23T->RchildPtr, SearchKey)

Example: Search for 80

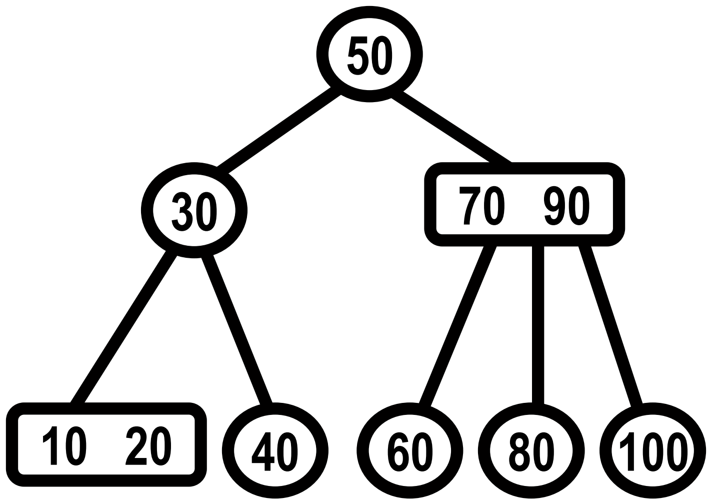


Example: Search for 110

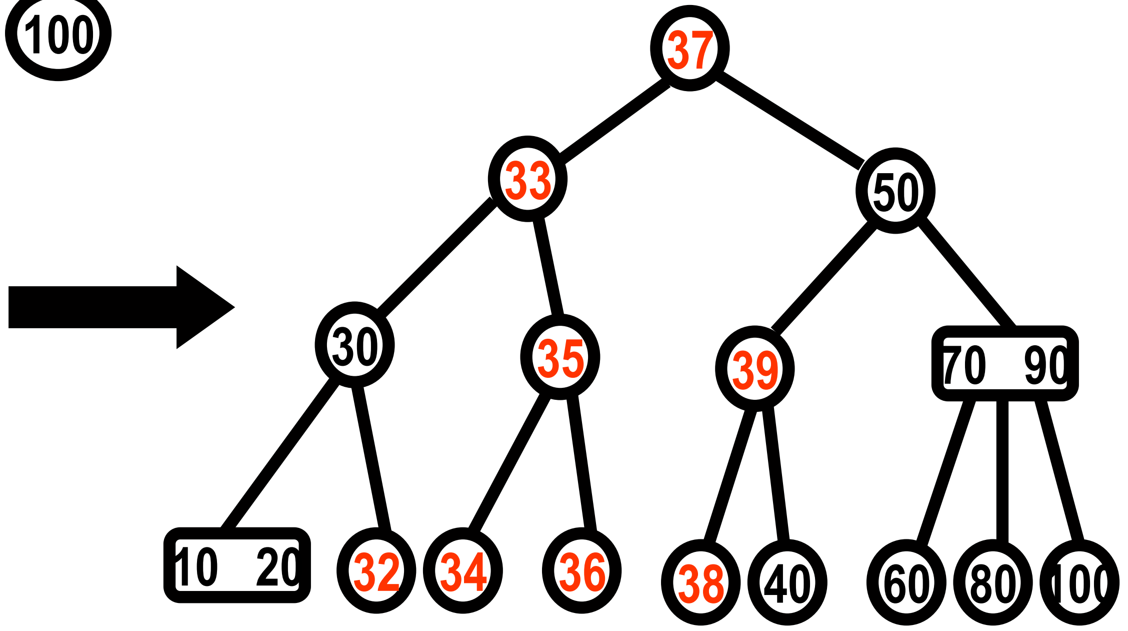


Inserting Items into a 2-3 Tree

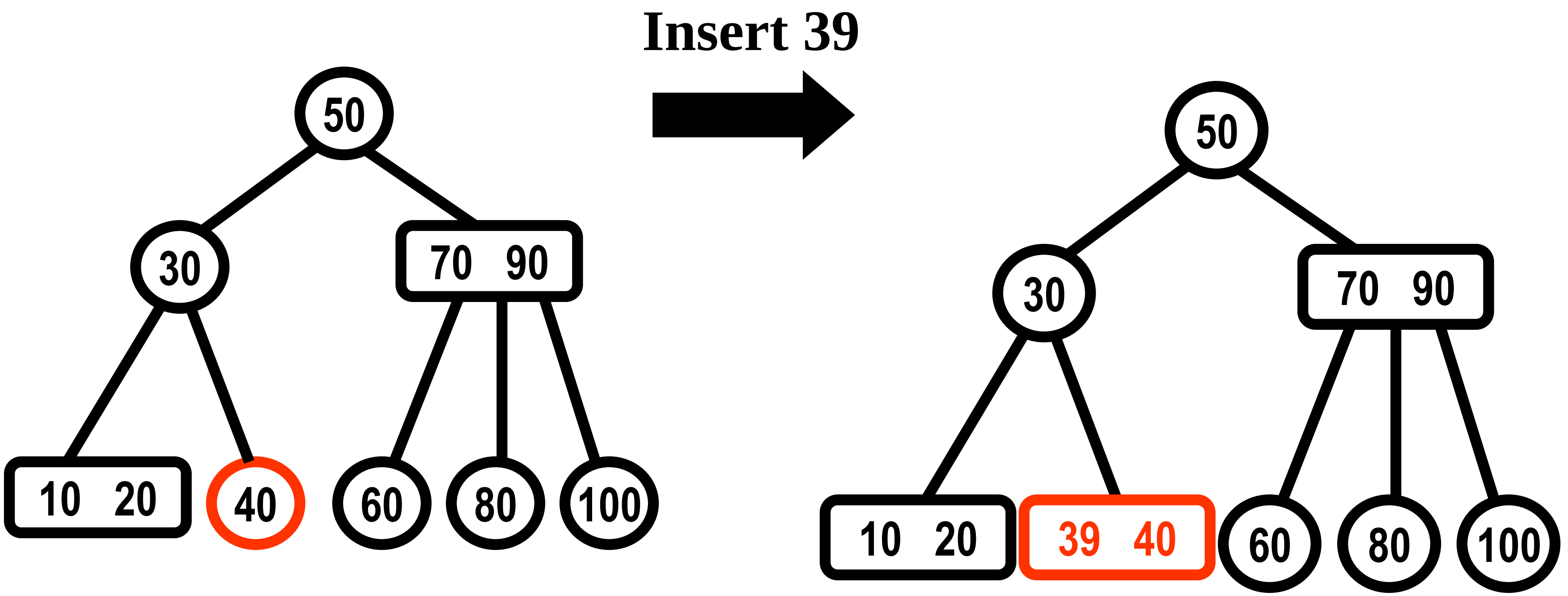
Example: Inserting Items into a 2-3 Tree



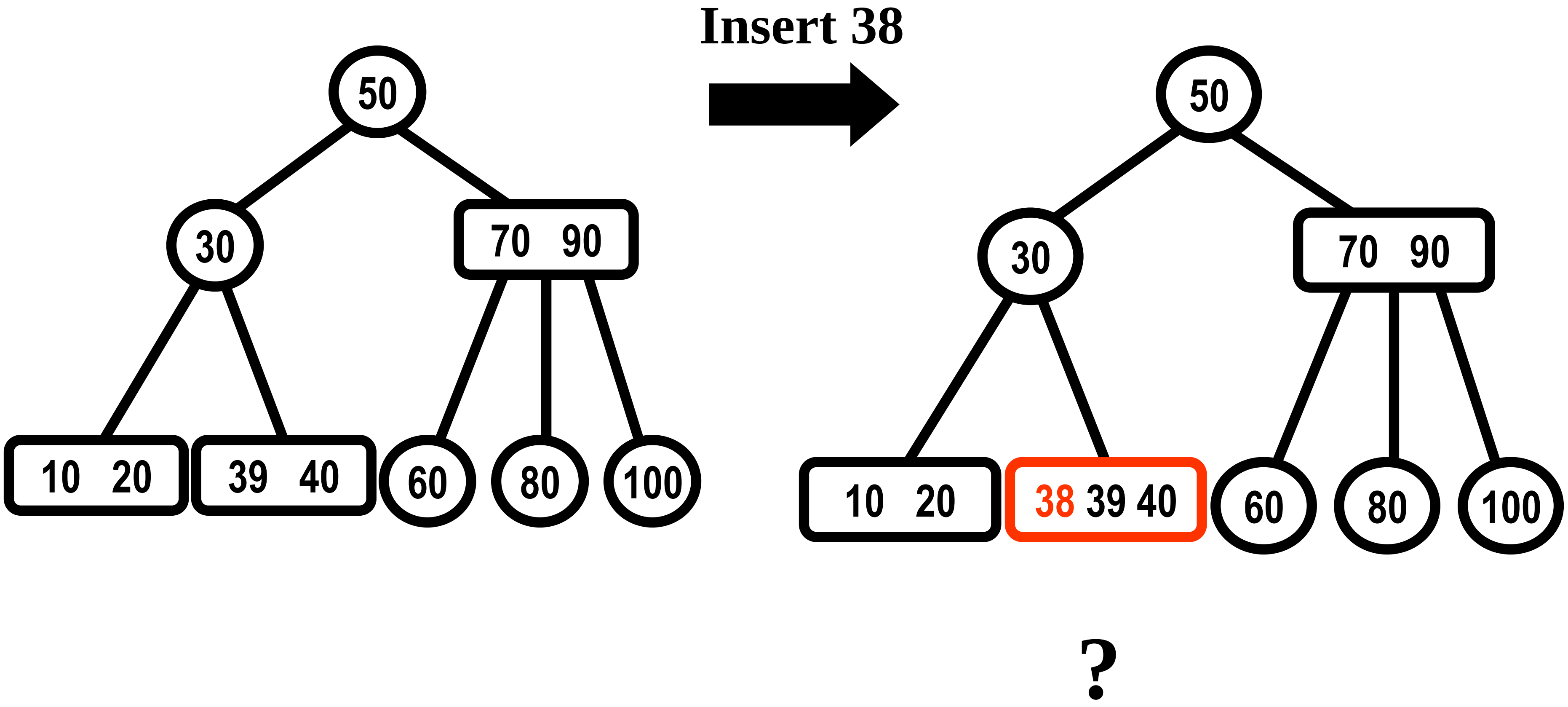
Insert: 39, 38, 37, 36, 35, 34, 33, 32



Example

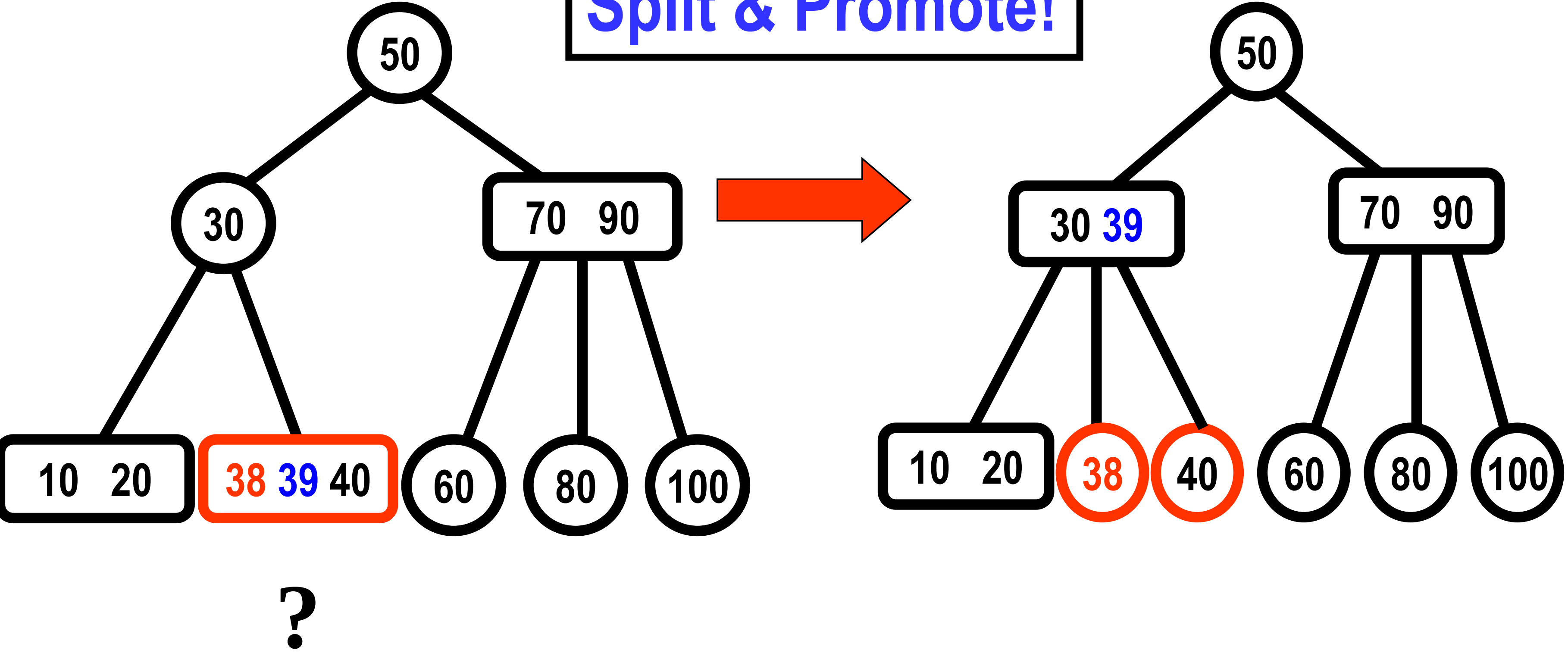

Insert 39
50
30
70 90
10 20
40
60
80
100
50
30
70 90
10 20
39 40
60
80
100

Example: Inserting Items into a 2-3 Tree



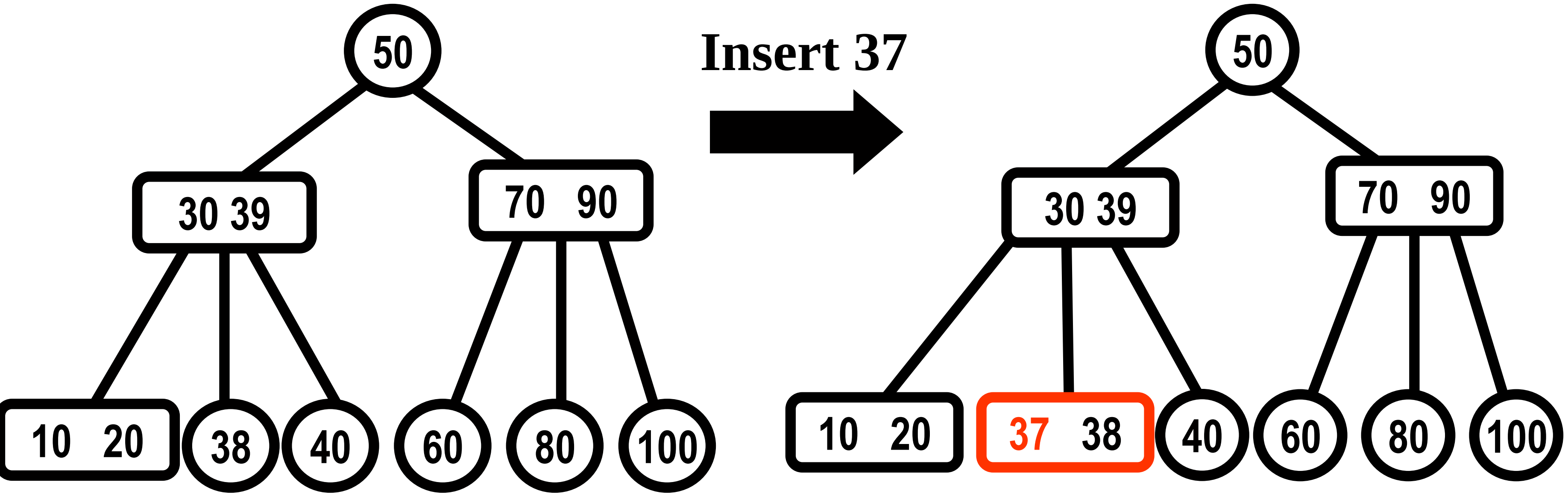
Example: Inserting Items into a 2-3 Tree

Split & Promote!

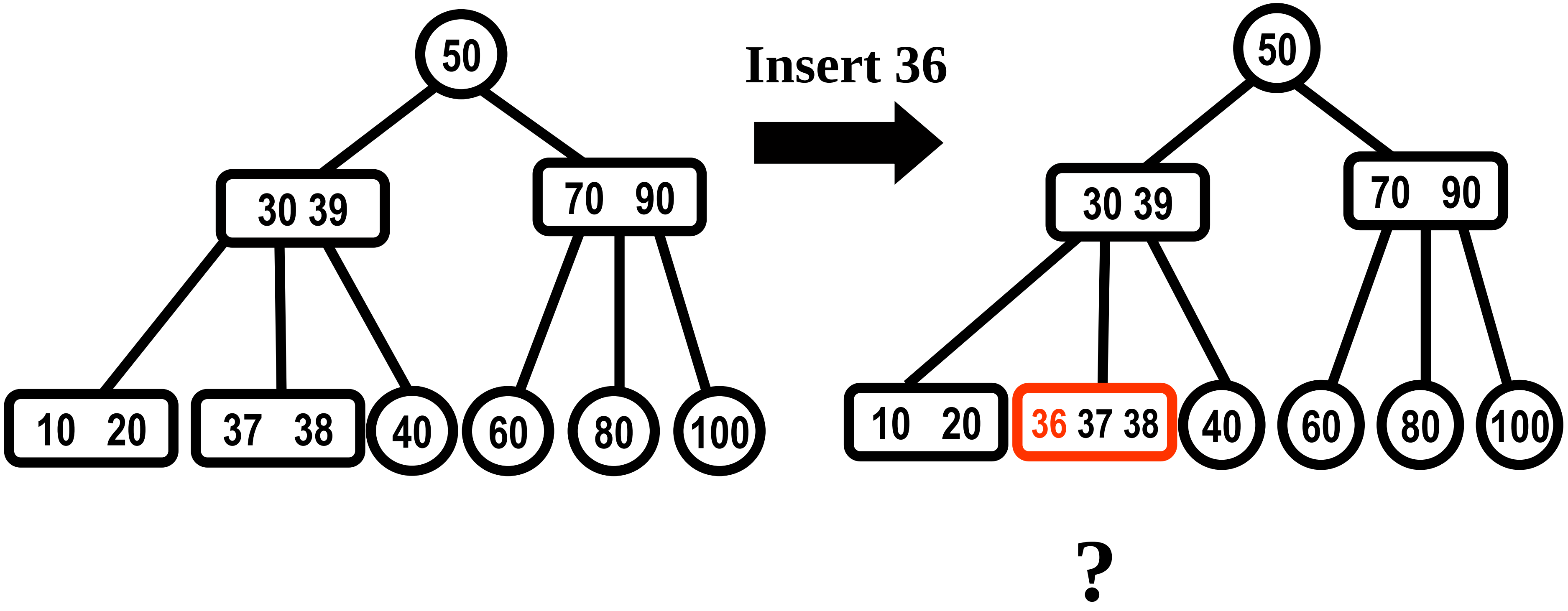


Median

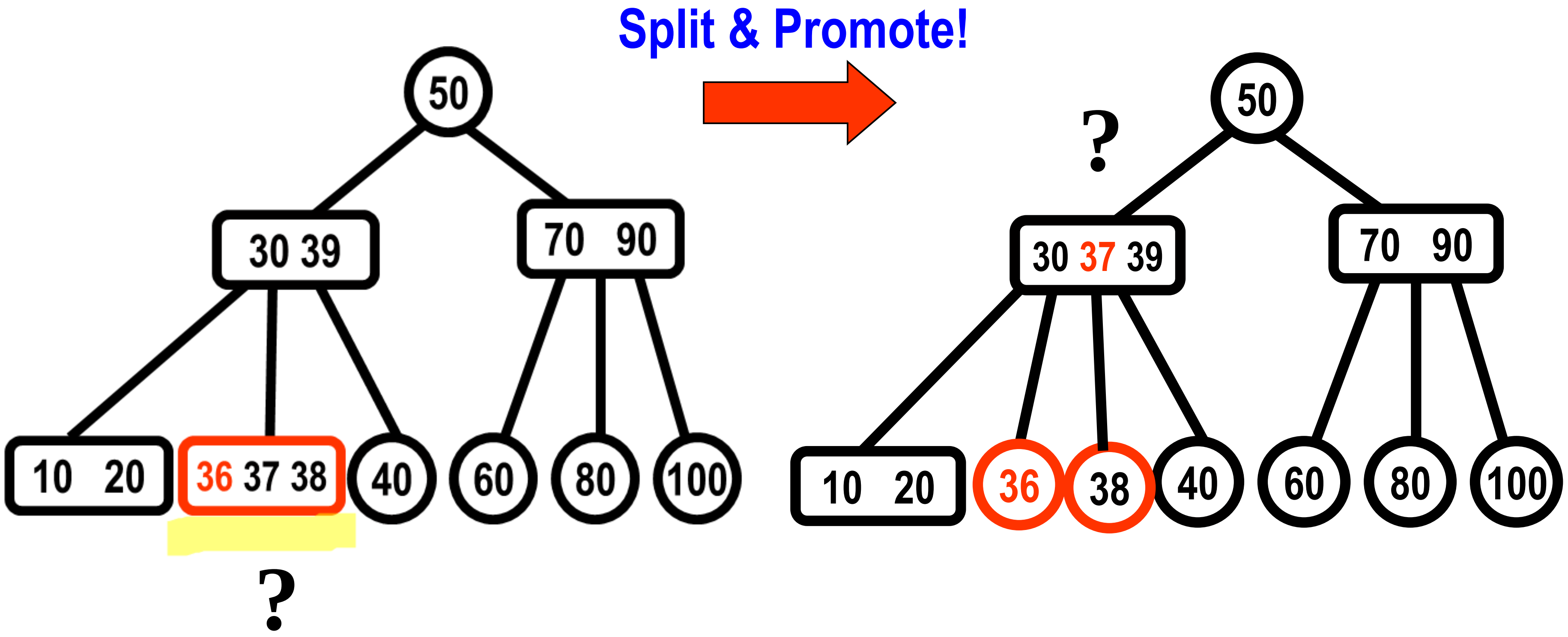
Example: Inserting Items into a 2-3 Tree



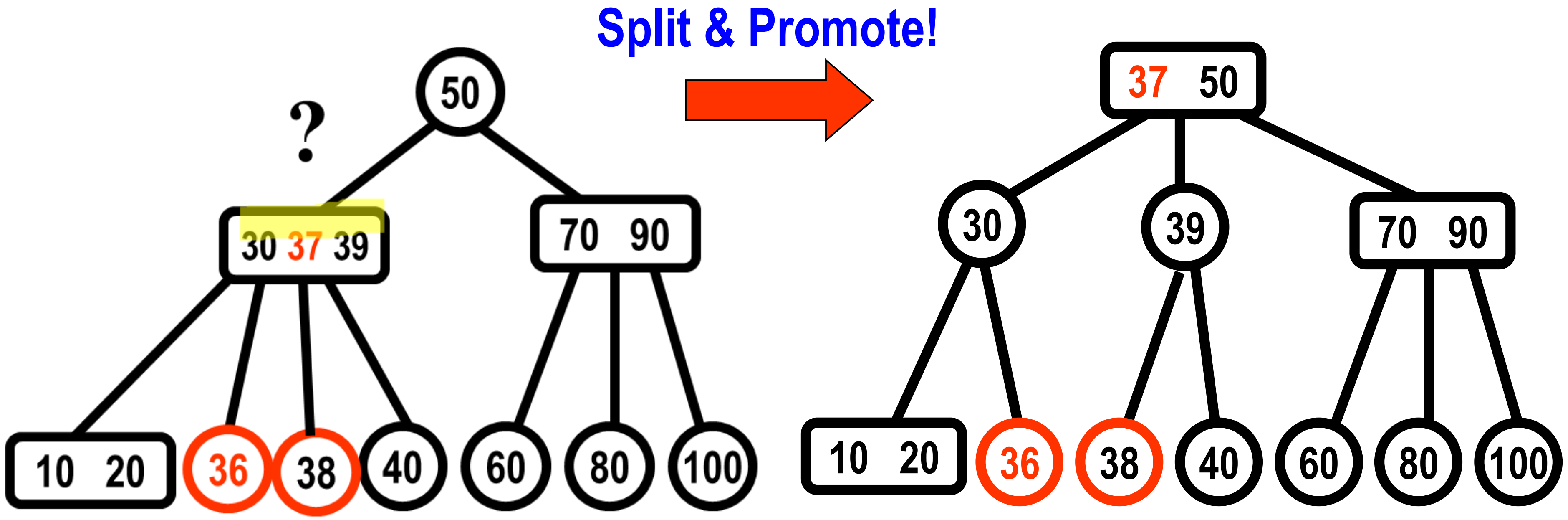
Example: Inserting Items into a 2-3 Tree



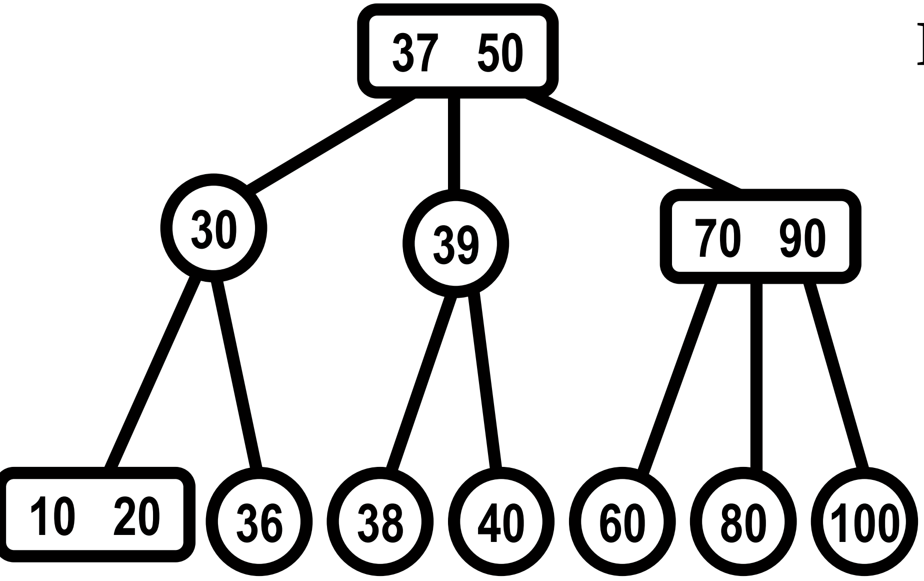
Example: Inserting Items into a 2-3 Tree



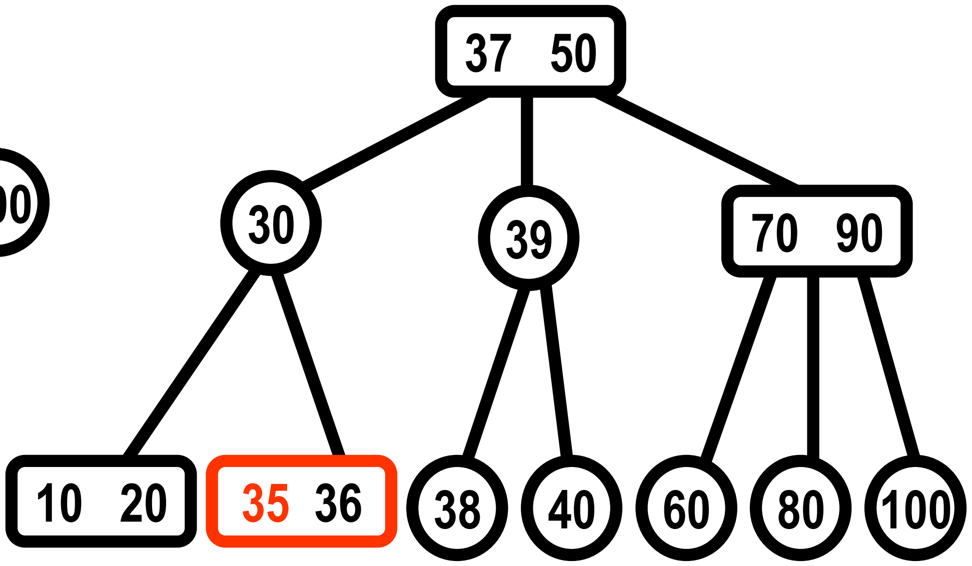
Example: Inserting Items into a 2-3 Tree



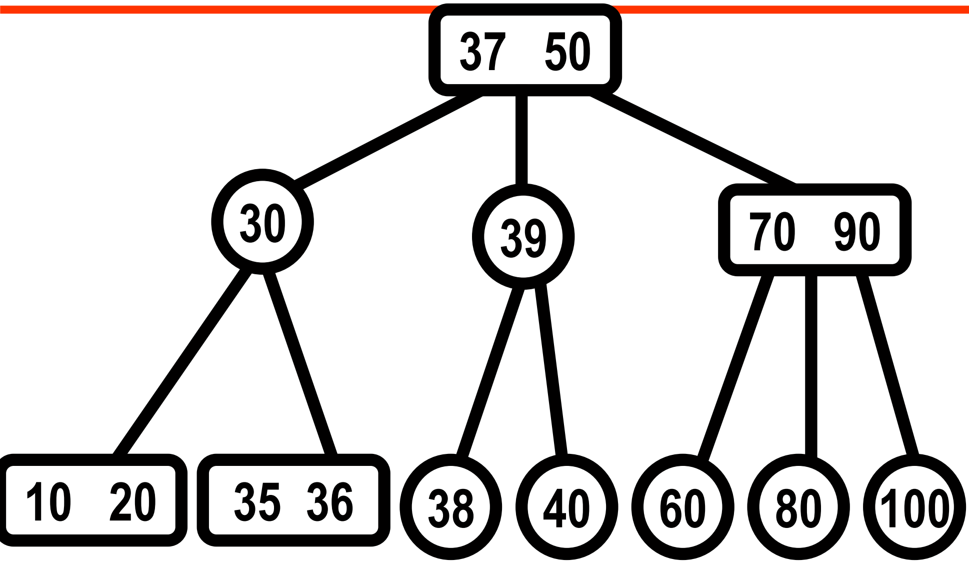
Example: Inserting Items into a 2-3 Tree



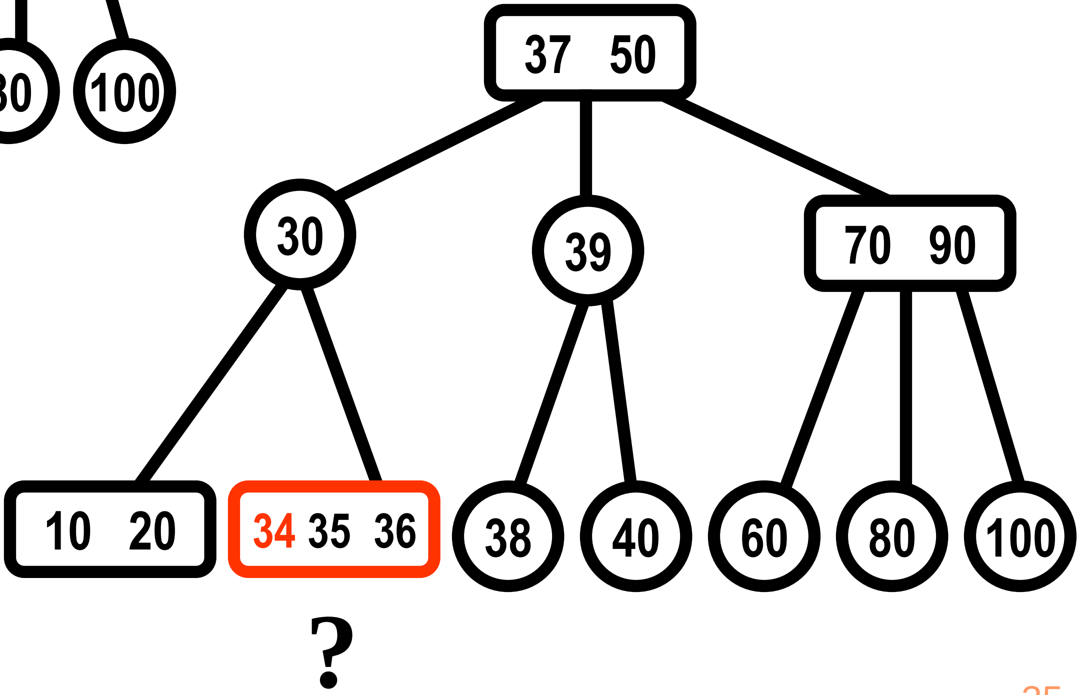
Insert 35



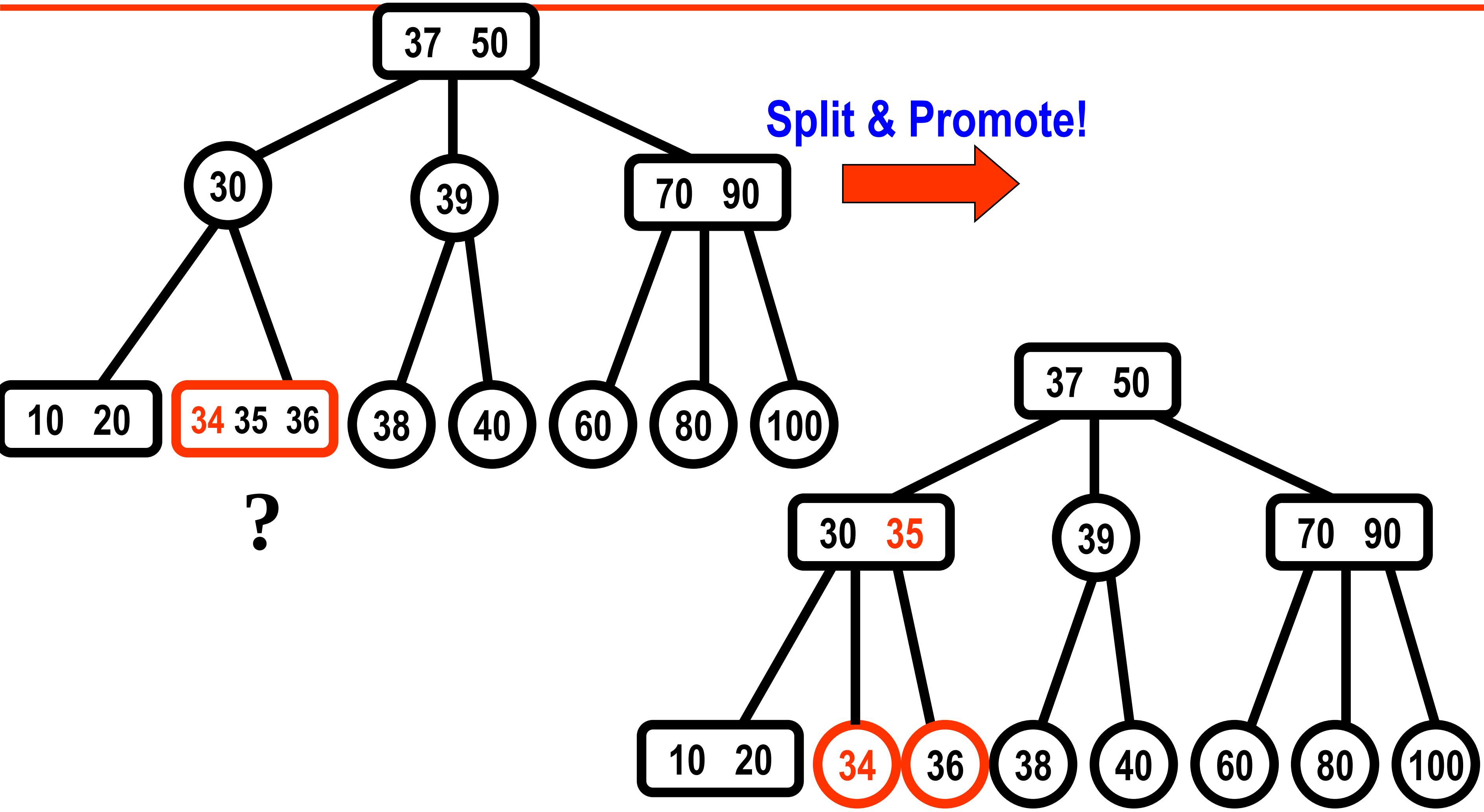
Example: Inserting Items into a 2-3 Tree



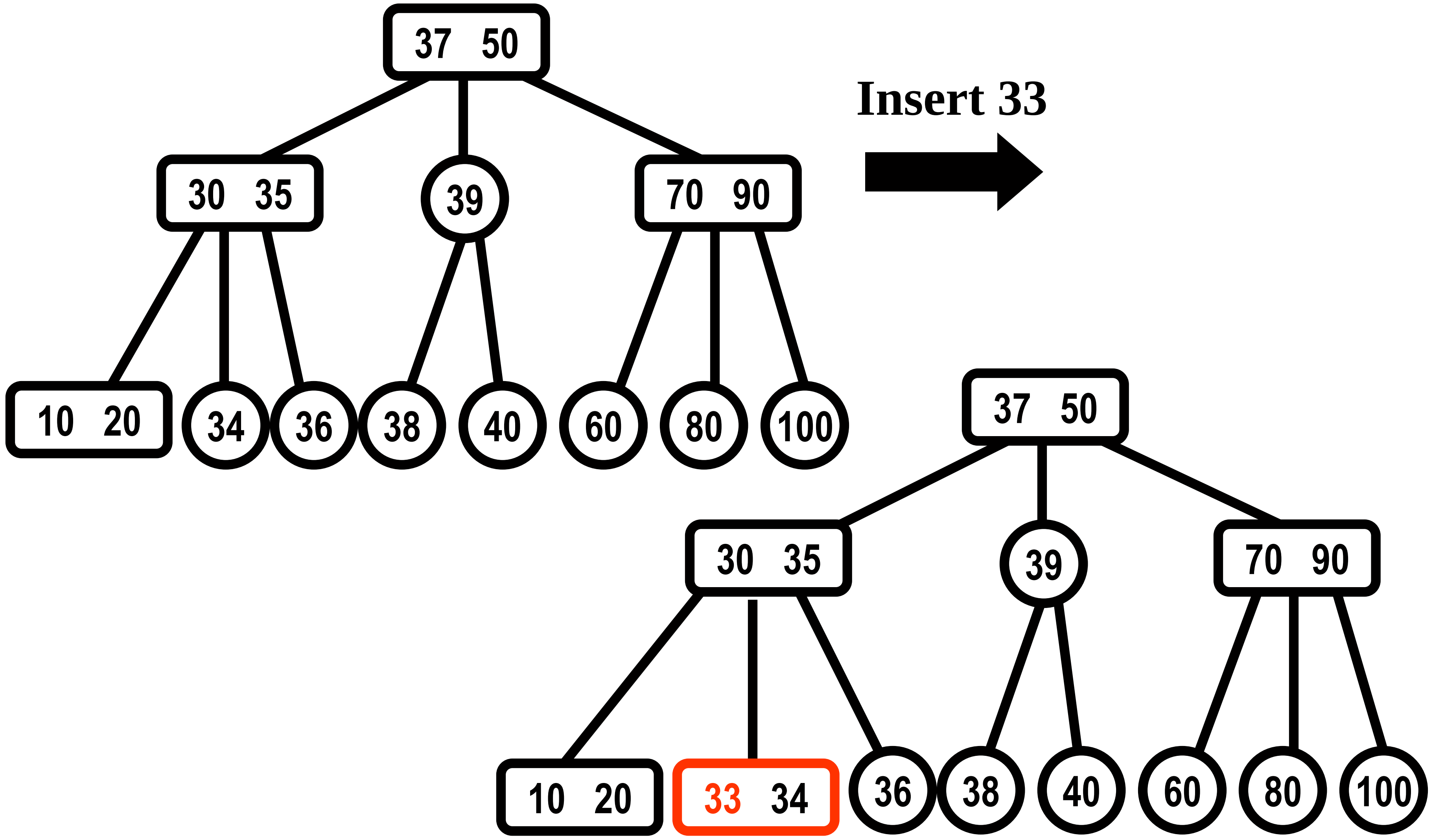
Insert 34



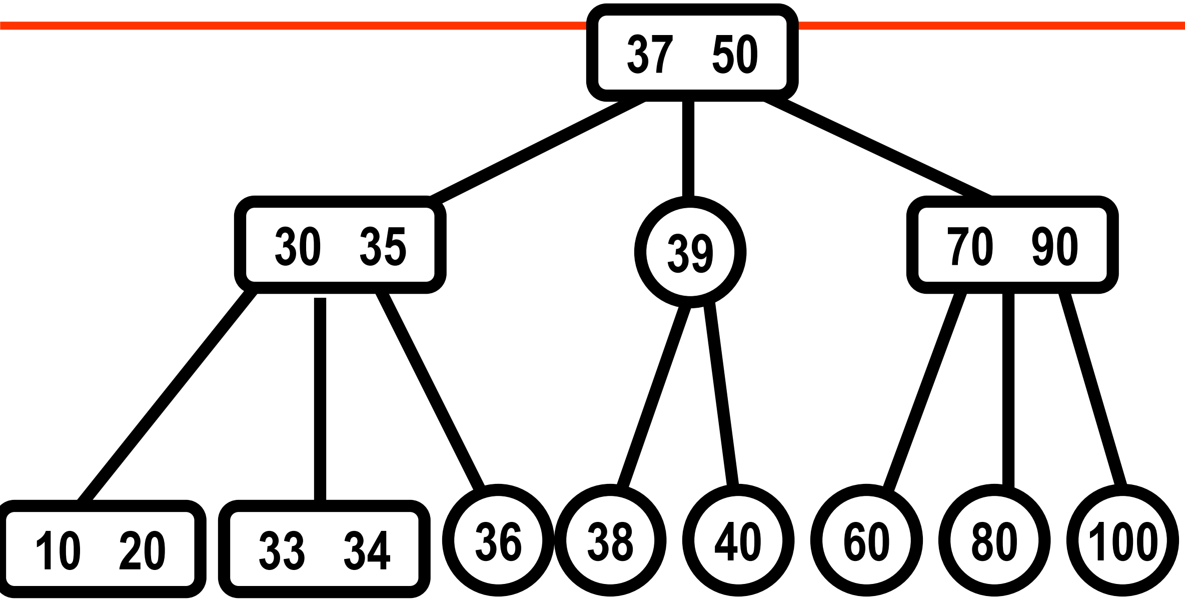
Example: Inserting Items into a 2-3 Tree



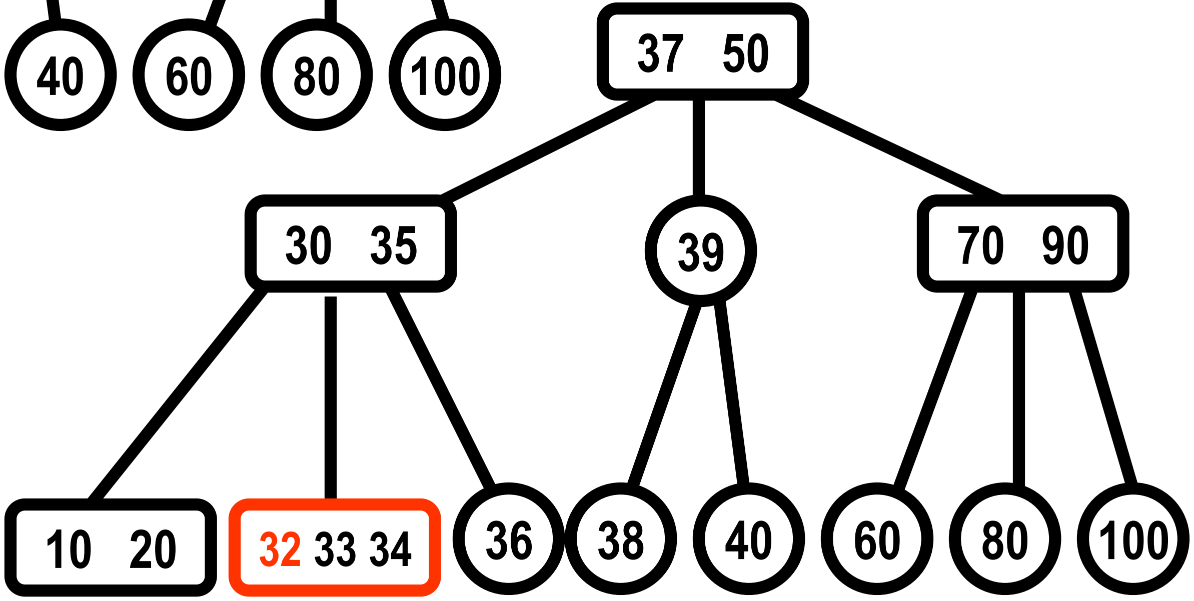
Example: Inserting Items into a 2-3 Tree



Example: Inserting Items into a 2-3 Tree



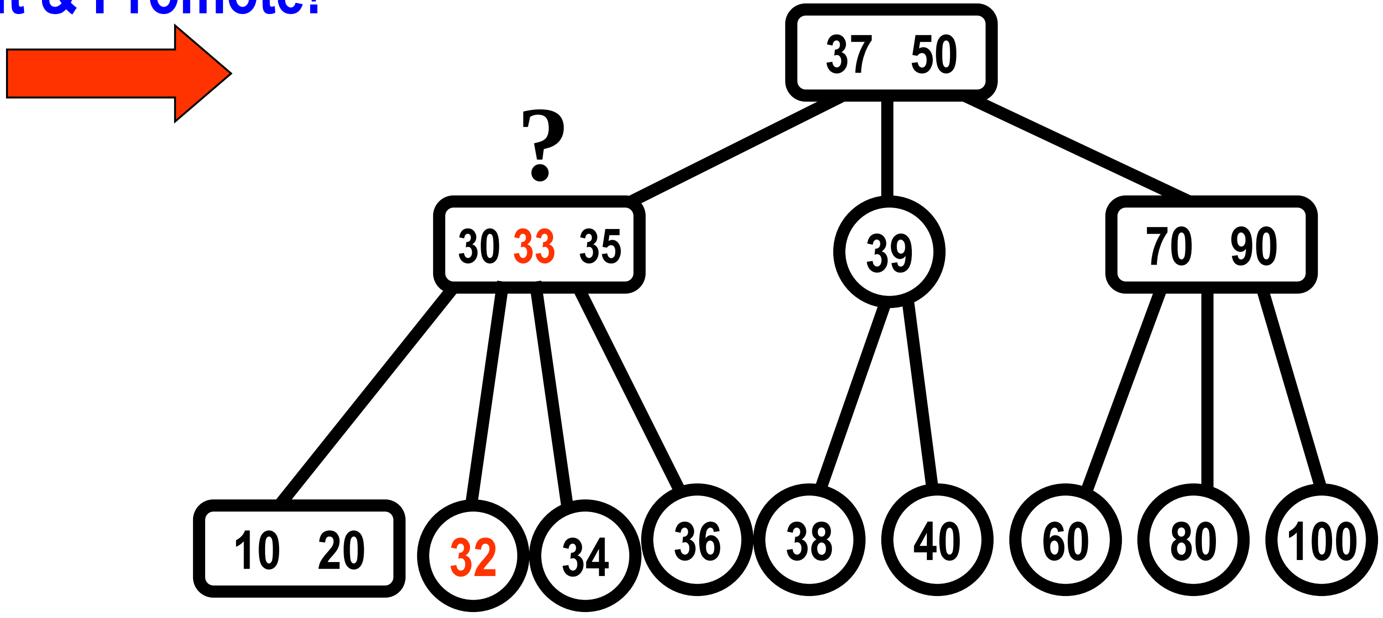
Insert 32



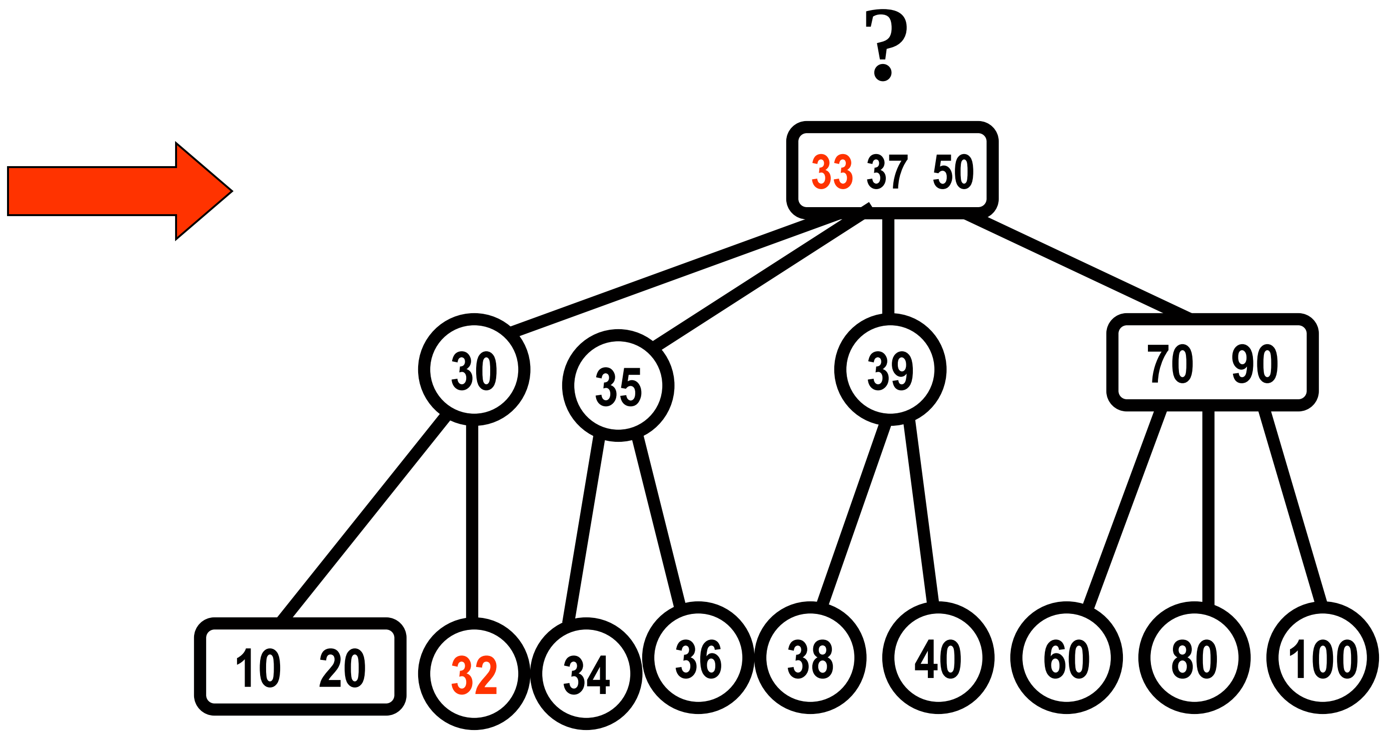
?

Example: Inserting Items into a 2-3 Tree

Split & Promote!

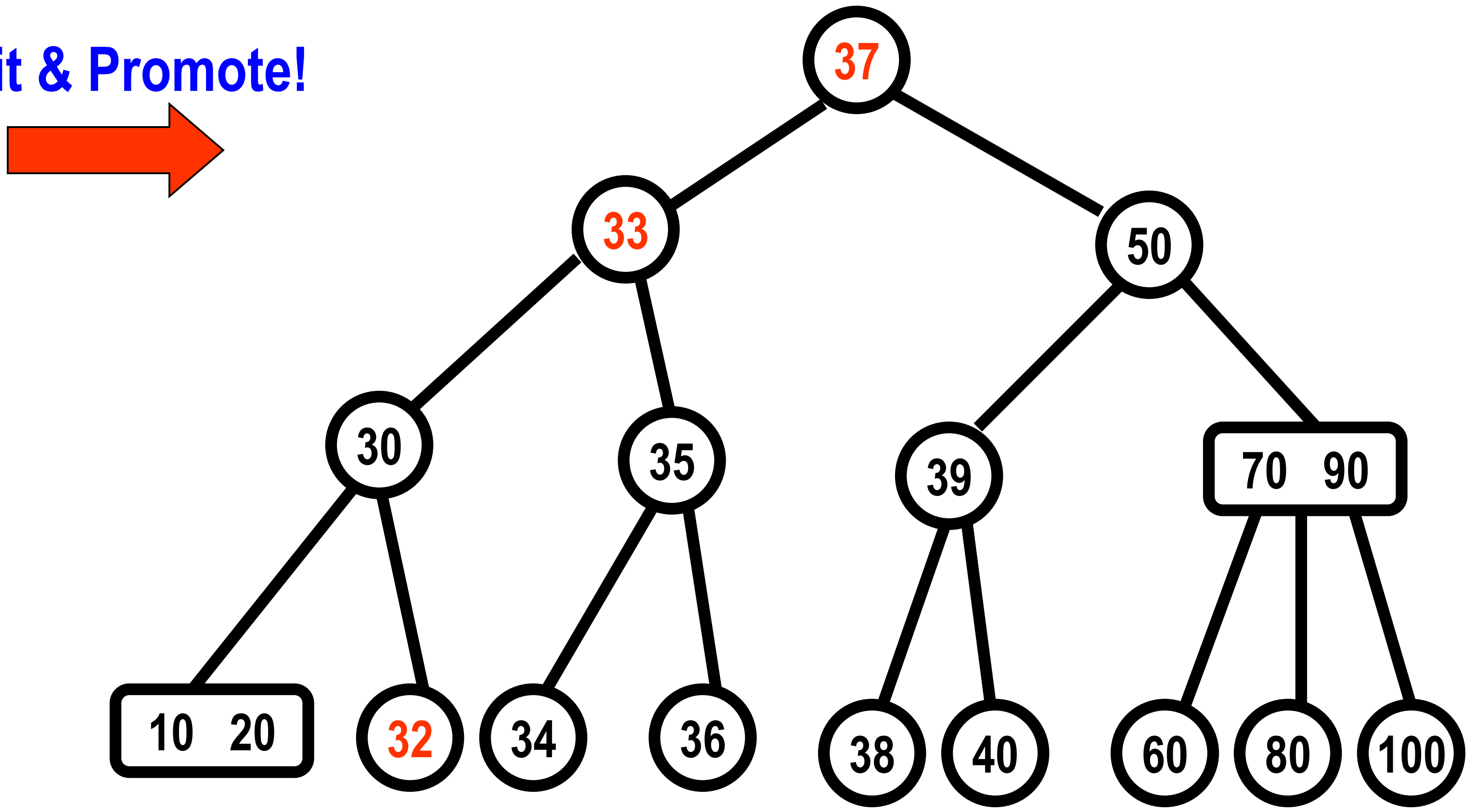


Example: Inserting Items into a 2-3 Tree



Example: Inserting Items into a 2-3 Tree

Split & Promote!



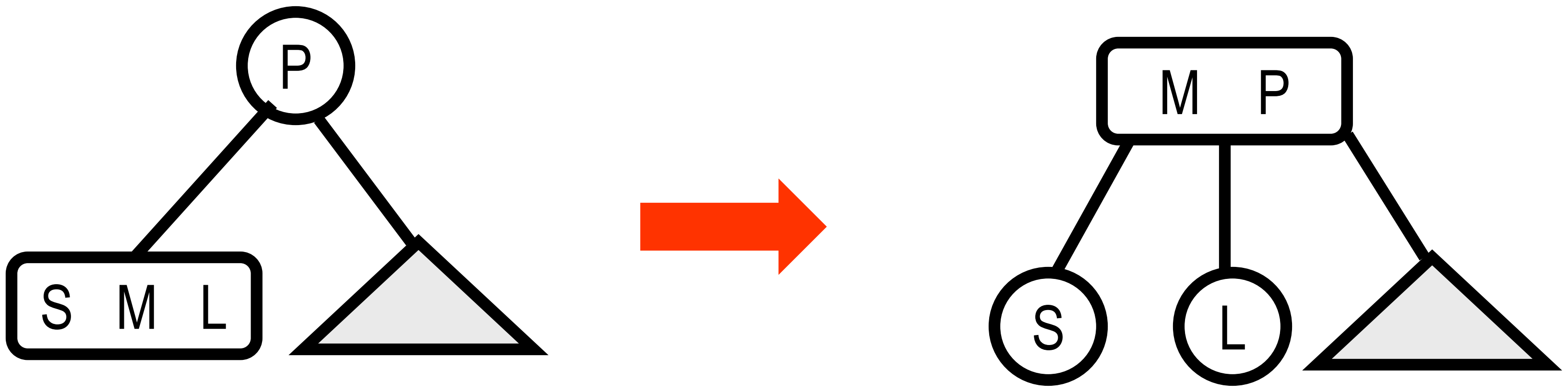
Insert Operation

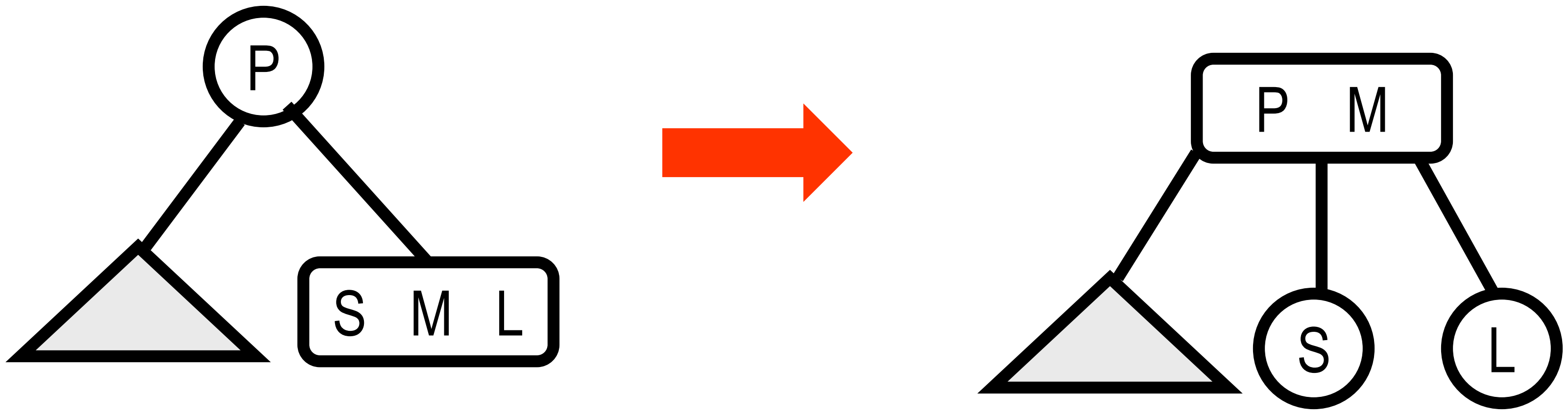
- Insert(23T, NewItem):
  - Search the leaf L in which the key of NewItem belongs.
  - Add NewItem to L.
  - If L now has three items then
    - **Split(L) & Promote the Median**
- **Two-pass process**

Split&Promote(N): Splitting Nodes & Promote the Median

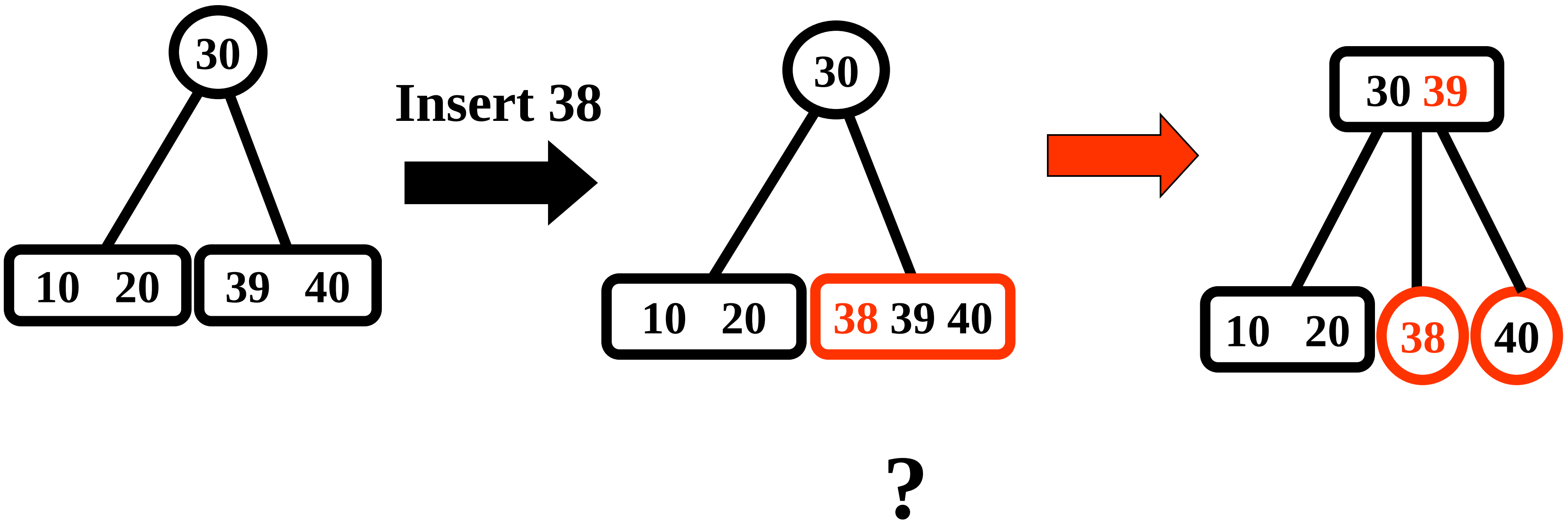
- Split&Promote(N)
- Three cases:
  - N = a leaf node
  - N = an internal node
  - N = the root

1. Split&Promote a Leaf Node

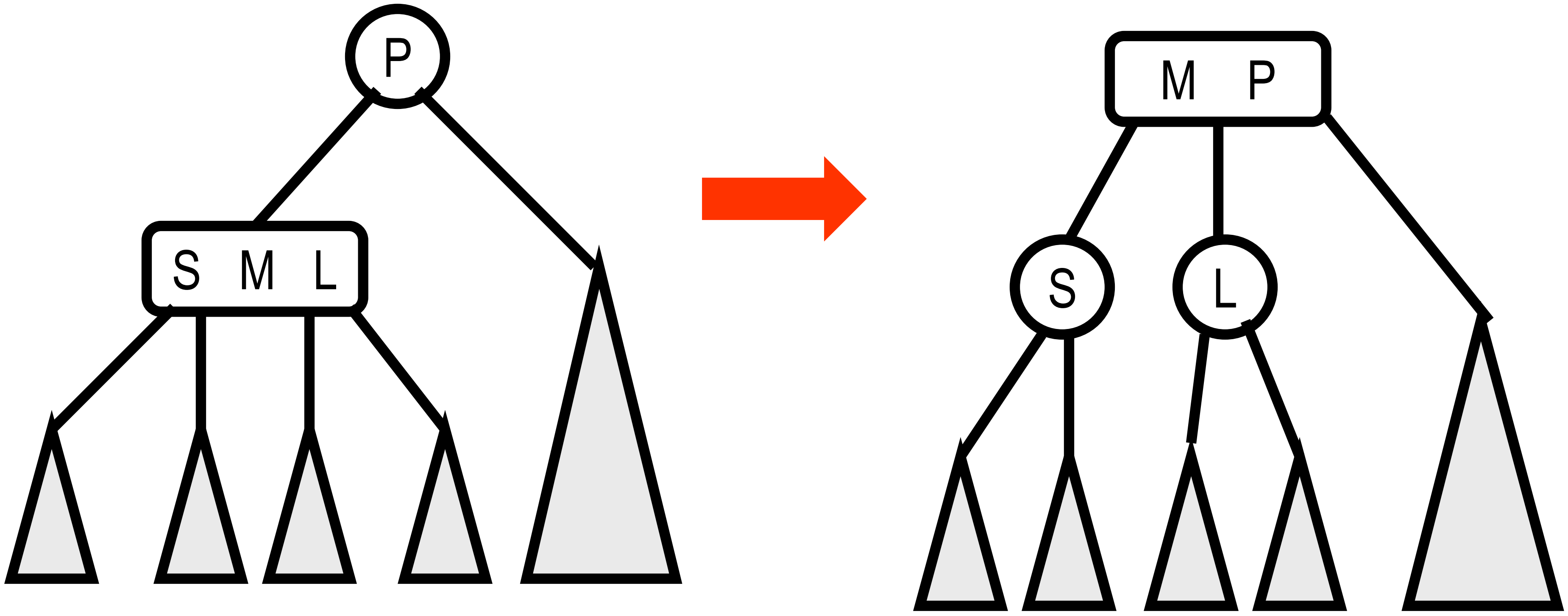




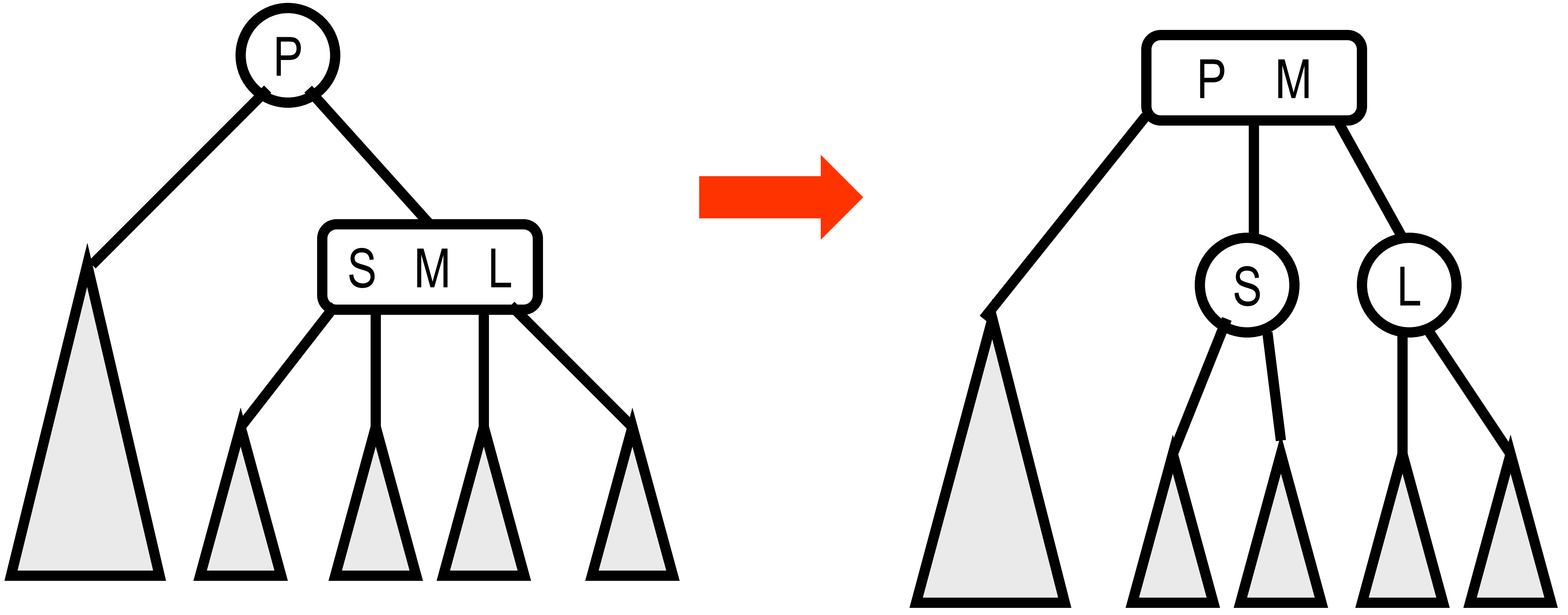
Example: Split & Promote



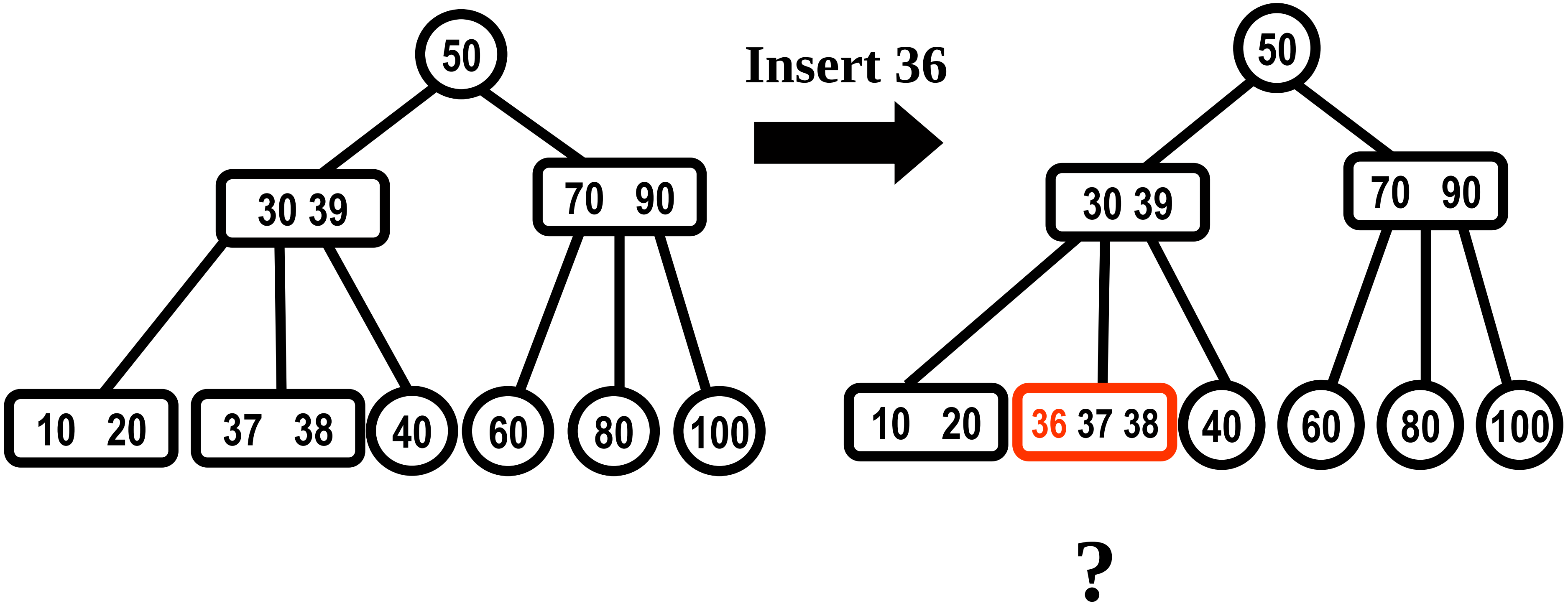
2. Split&Promote an Internal Node



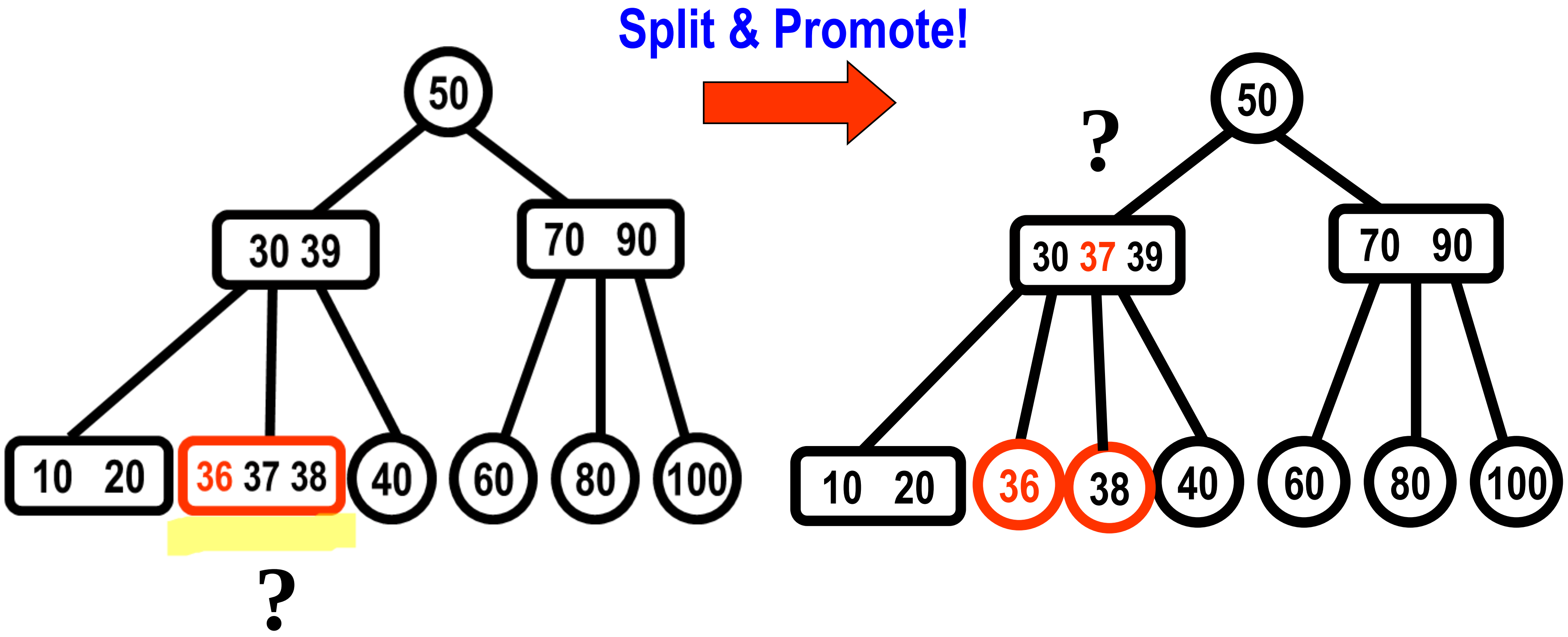
2. Split&Promote an Internal Node



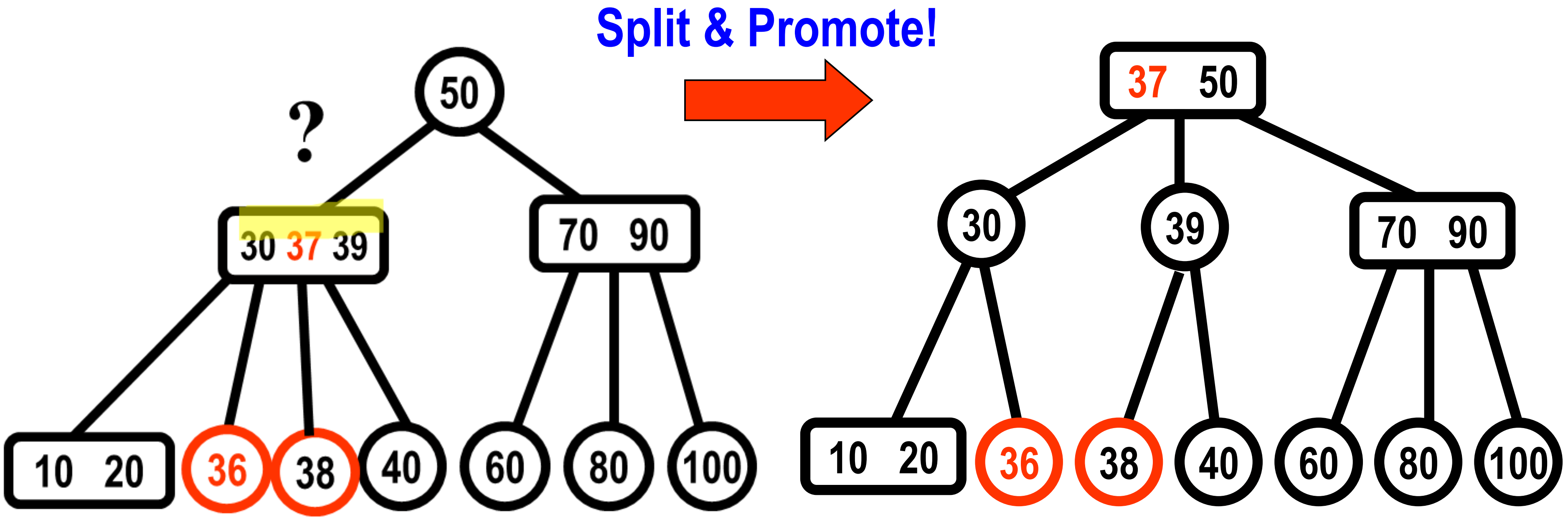
Example: Split & Promote



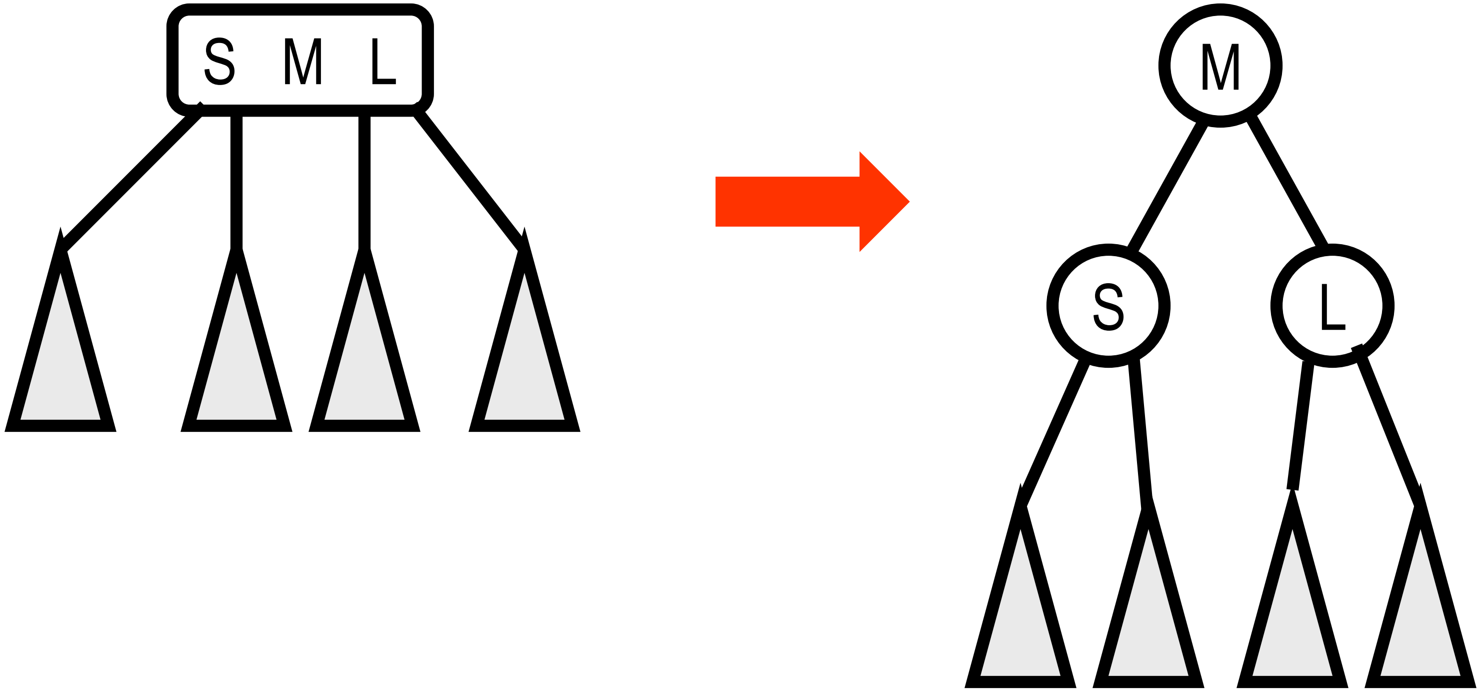
Example: Split & Promote



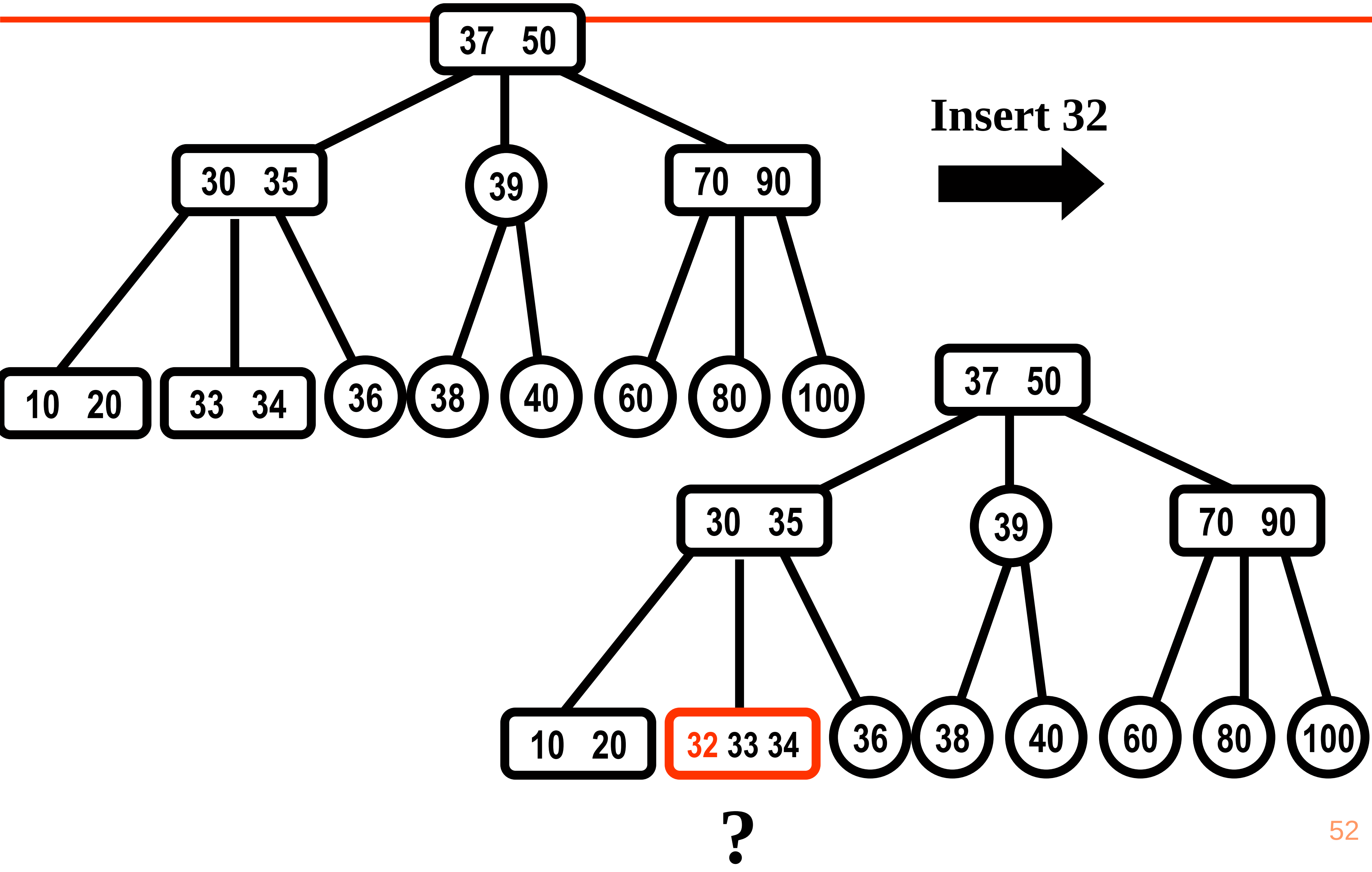
Example: Split & Promote



3. Split&Promote the Root Node

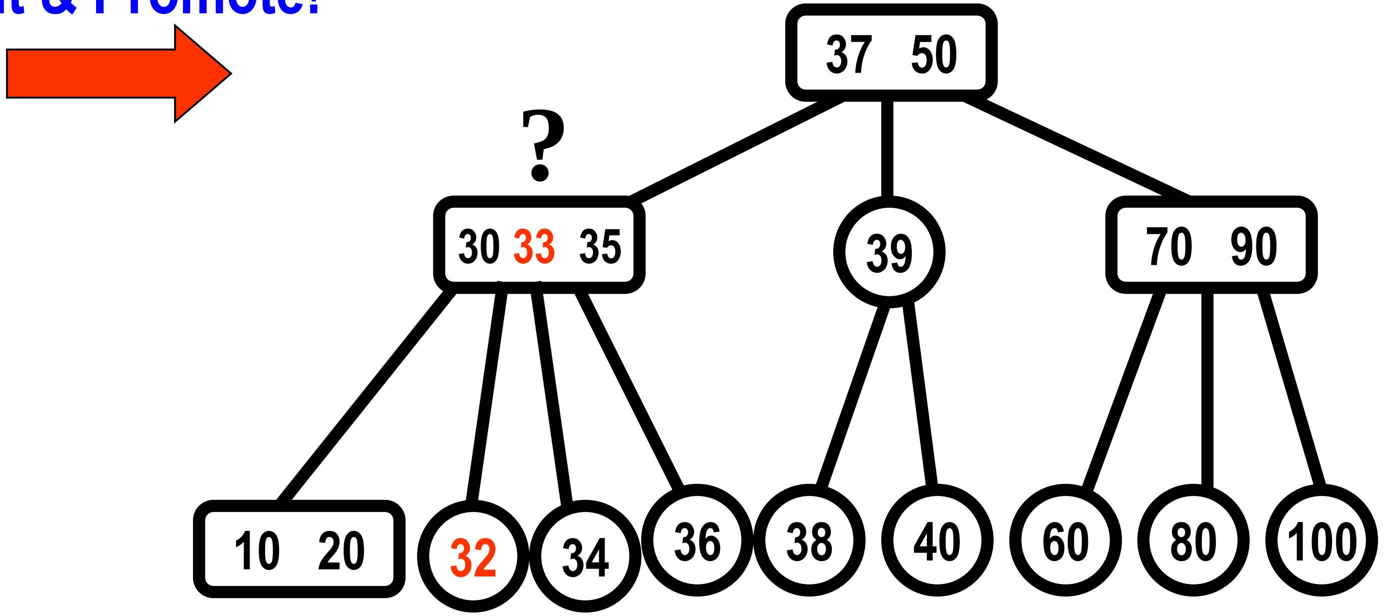


Example: Split & Promote

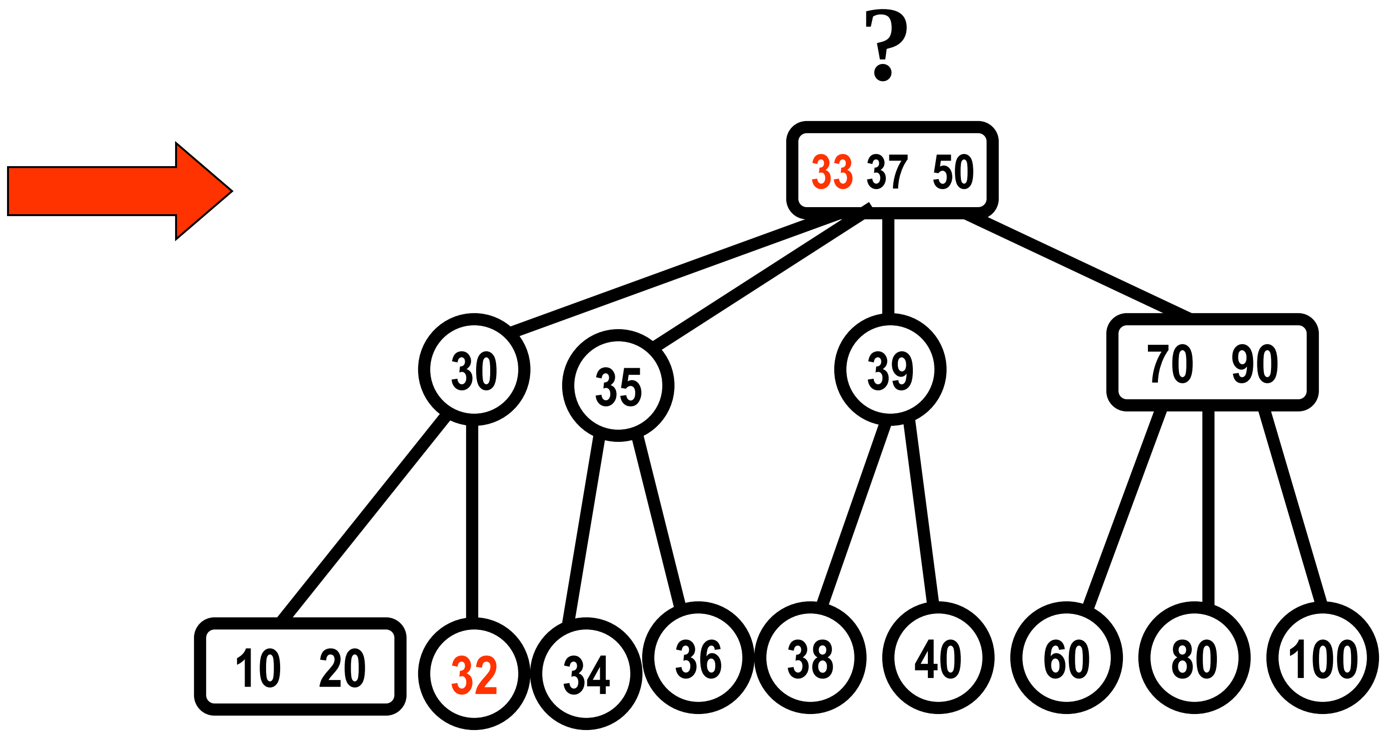


Example: Split & Promote

Split & Promote!

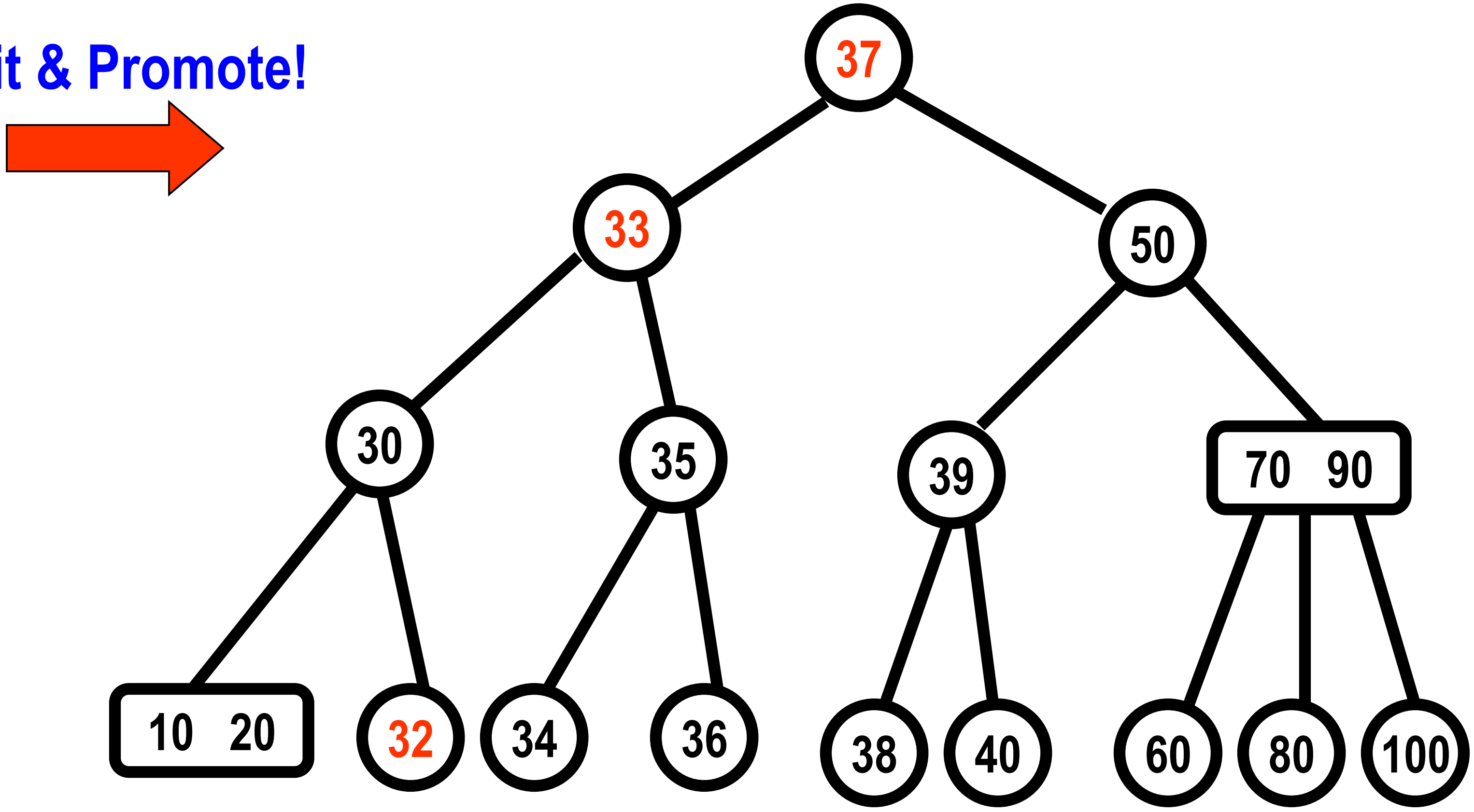


Example: Split & Promote



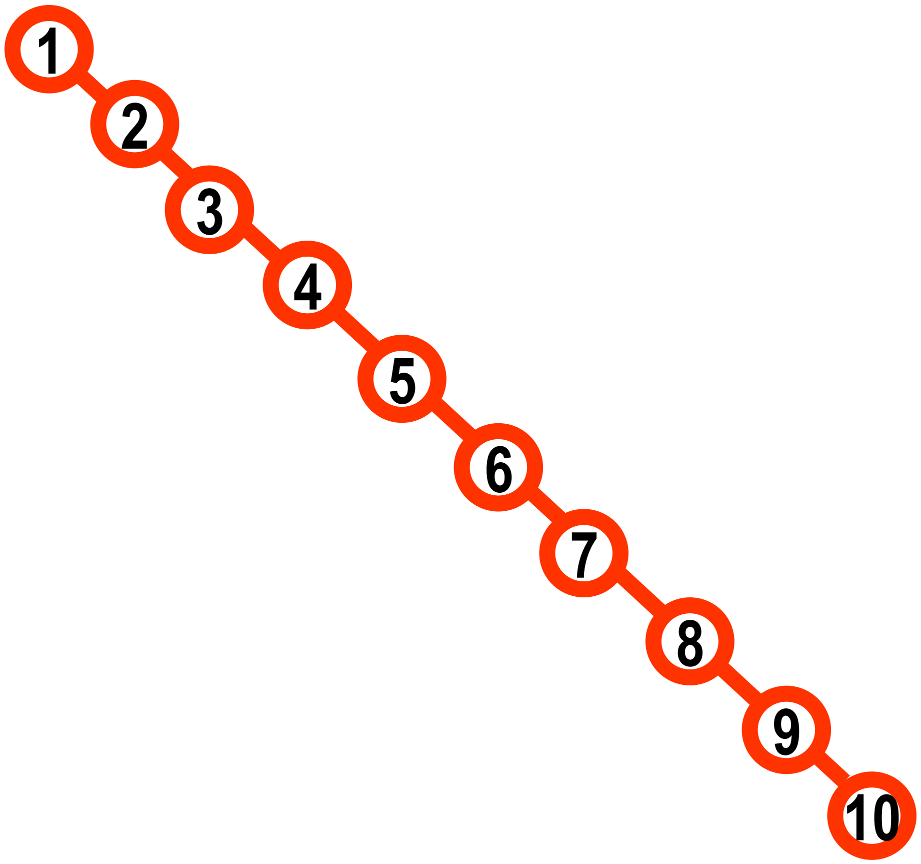
Example: Split & Promote

Split & Promote!



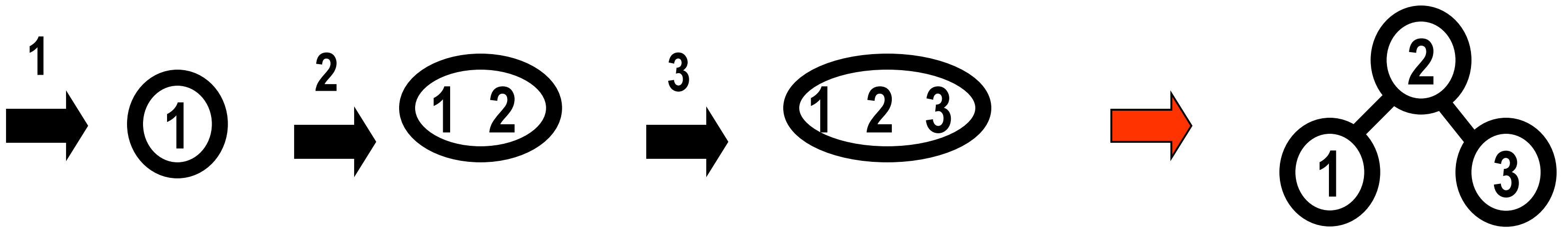
►QUIZ? Binary Search Tree?

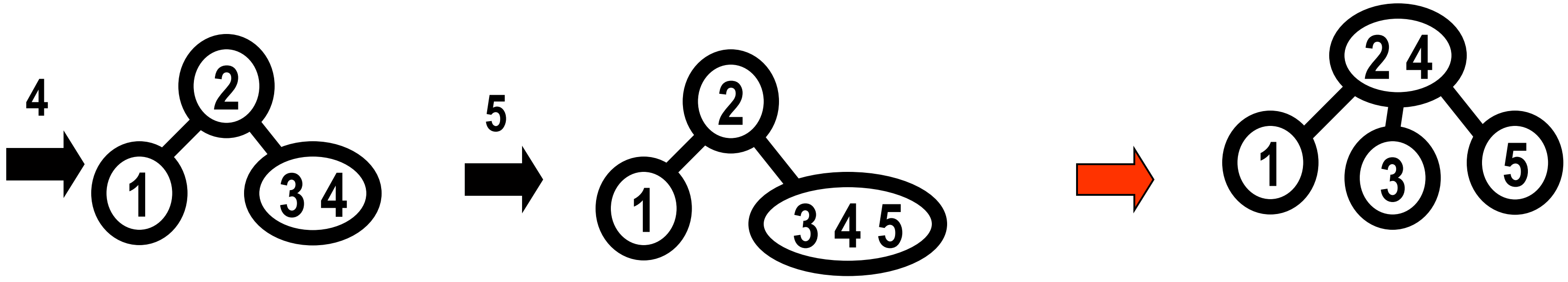
Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10



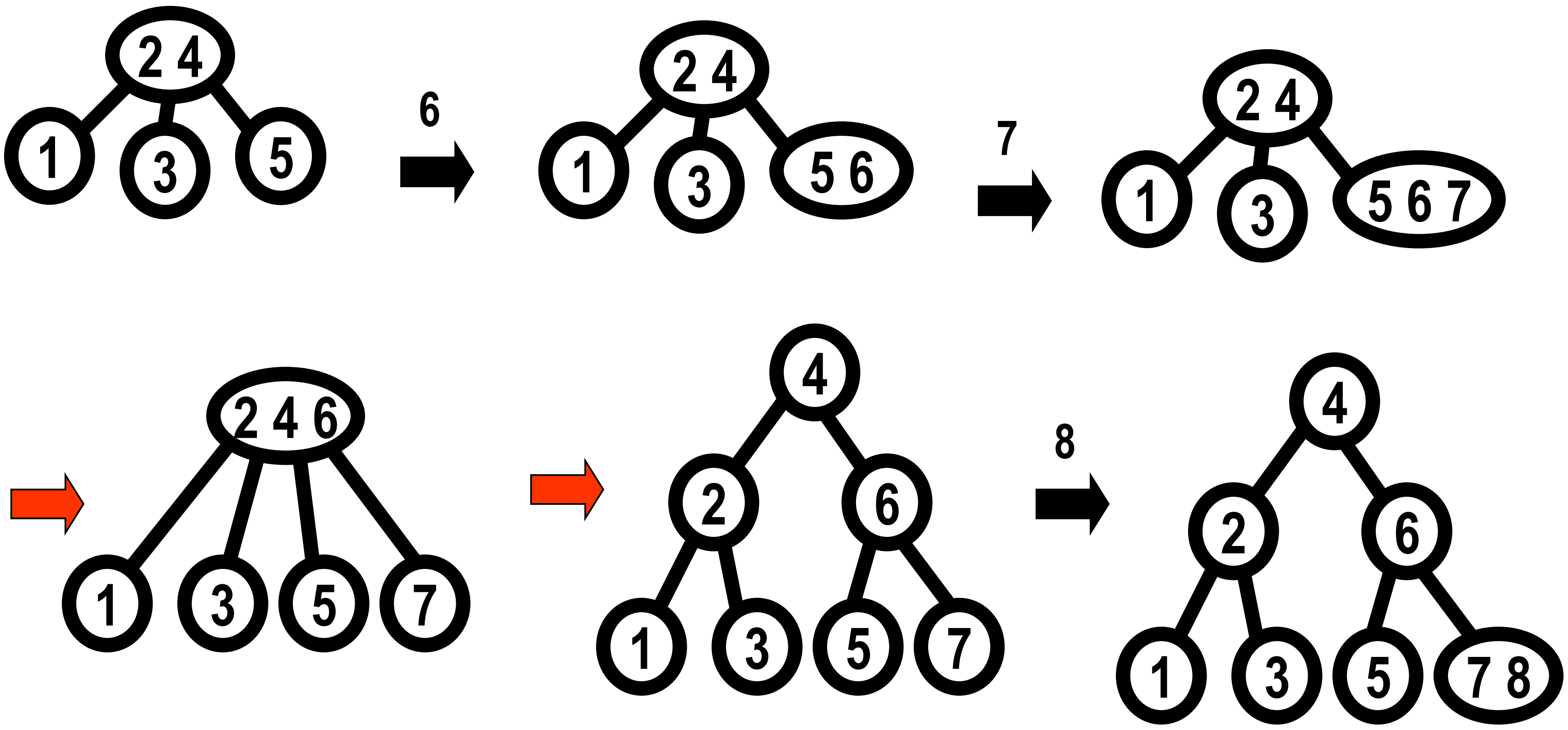
►QUIZ? 2-3 Tree?

Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10

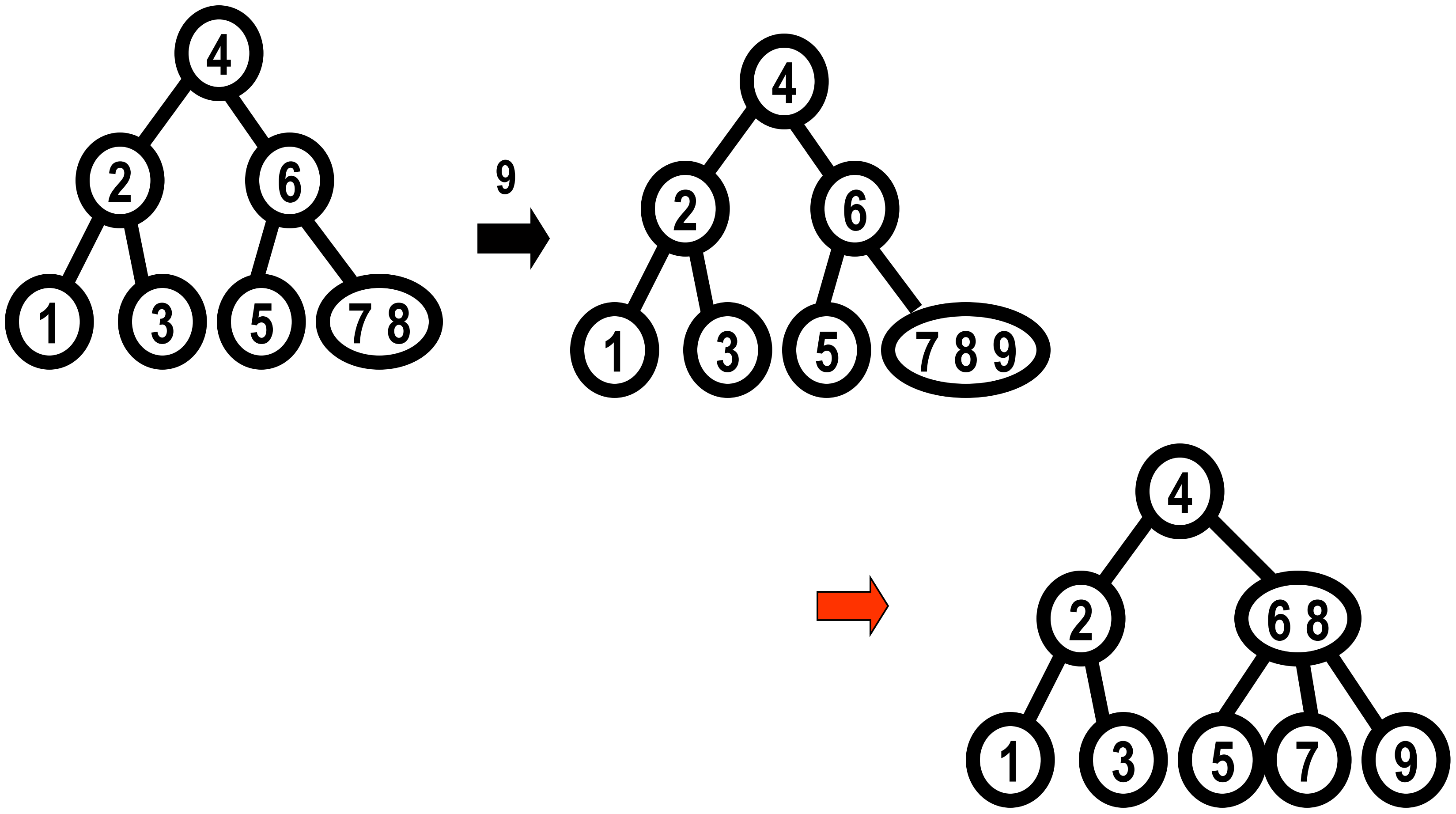




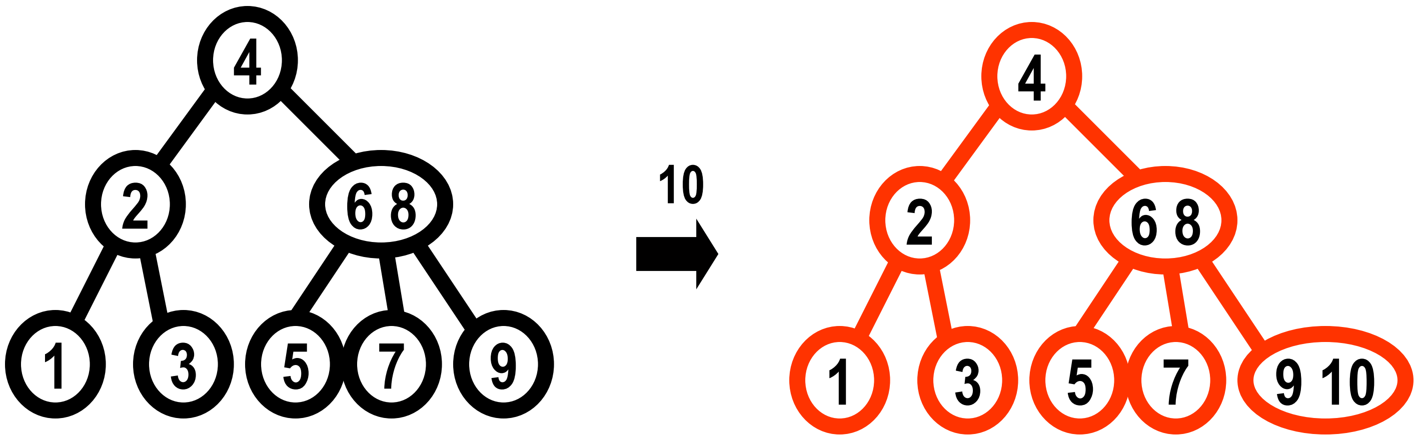
▶QUIZ? 2-3 Tree?



►QUIZ? 2-3 Tree?

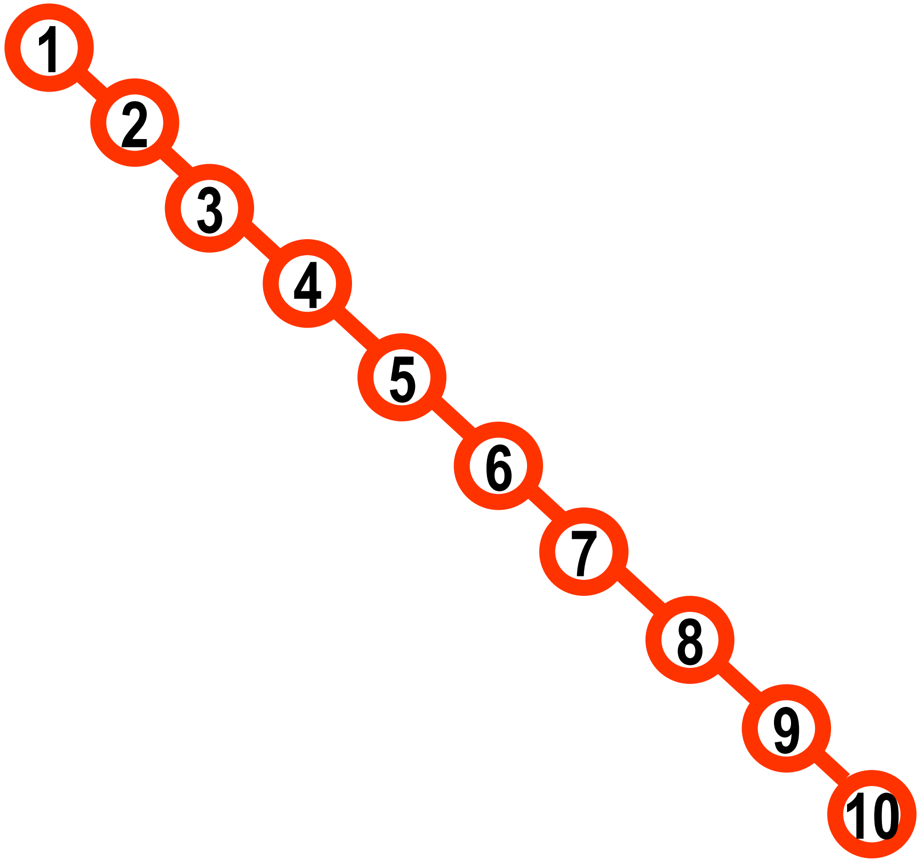


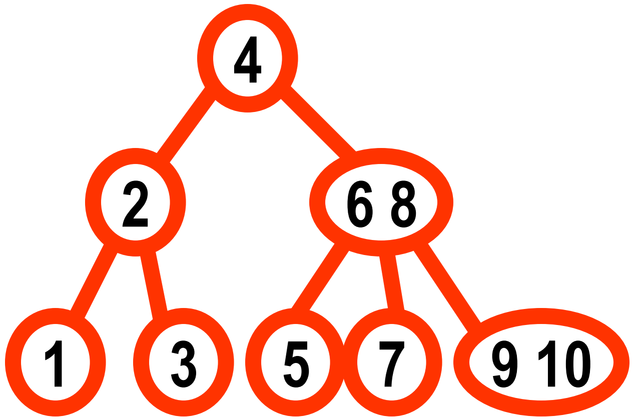
►QUIZ? 2-3 Tree?



BST vs 2-3 Tree

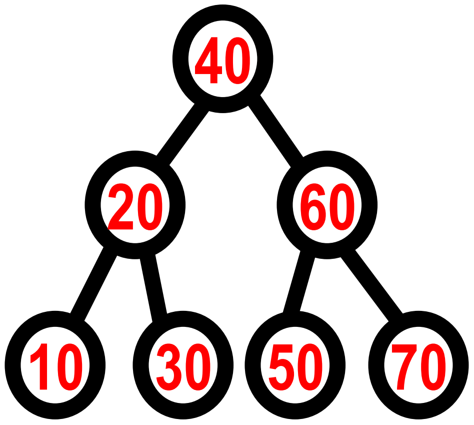
Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10





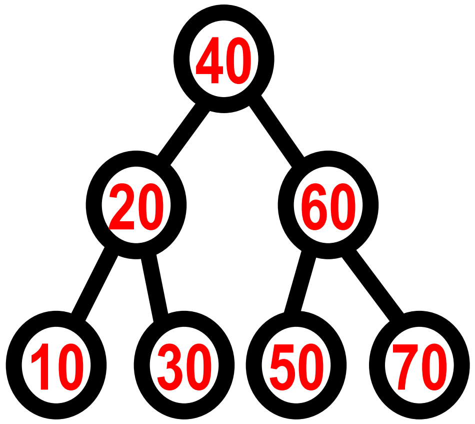
►QUIZ? 2-3Tree?

Insert 10, 20, 30, 40, 50, 60 and 70



►QUIZ? 2-3Tree?

Insert 70, 60, 50, 40, 30, 20 and 10



►QUIZ?

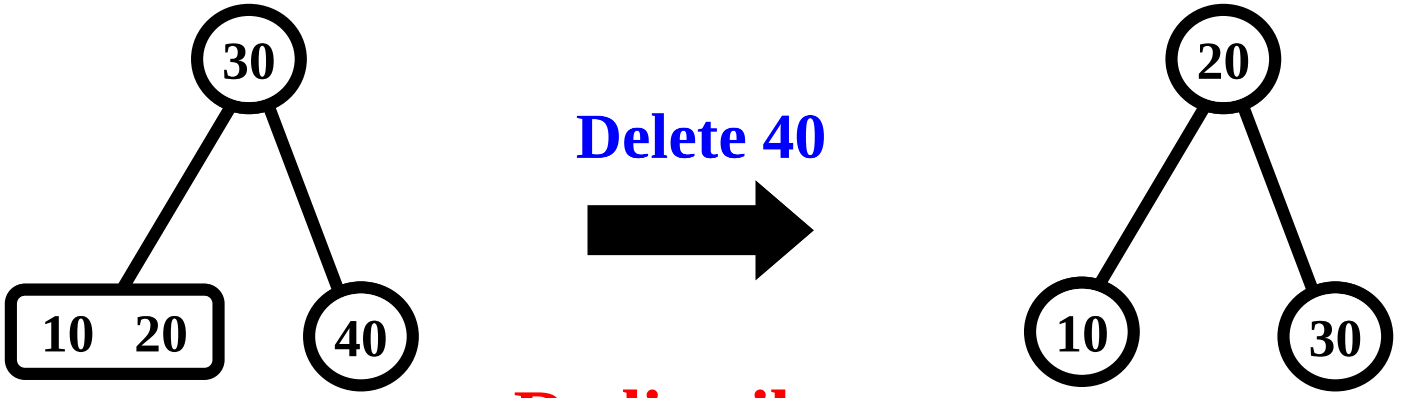
- Insert(23T, NewItem)?
  - Search the leaf L in which the key of NewItem belongs.
  - Add NewItem to L.
  - If L now has three items then
    - **Split(L) & Promote the Median**

  - **Two-pass process**

Delete Operation

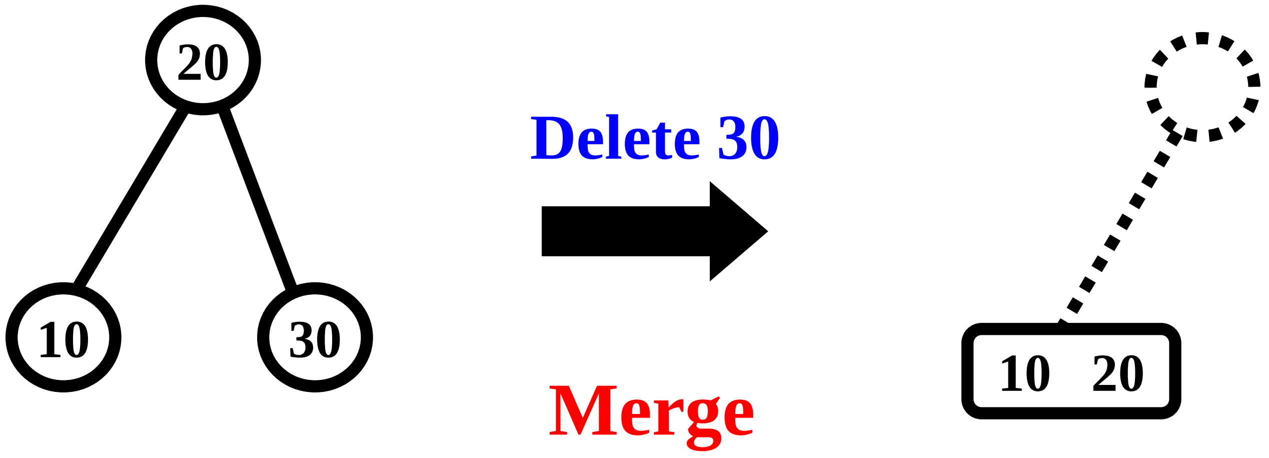
- Delete(23T, Item):
  - **Redistribute**
  - **Merge**

Delete Operation



Redistribute

Delete Operation



Representation of 2-3 Trees

- Similar to representation of binary trees, binary search trees and AVL trees

A Pointer-Based Representation of 2-3 Trees

```
template<class DataType>
class 23Tnode
{
public:
   23Tnode();

private:
   DataType SmallKey;
   DataType LargeKey;
   23Tnode<DataType>* LchildPtr;
   23Tnode<DataType>* MchildPtr;
   23Tnode<DataType>* RchildPtr;
};
```

Properties of 2-3 Trees

- A 2-3 tree is not a binary tree.
- A 2-3 tree does resemble a full binary tree.
- A 2-3 tree with N nodes never has height greater than the minimum height of a binary tree with N nodes.

Properties of 2-3 Trees

- What is the minimum number of nodes that a 2-3 tree of height h can have?
  - ➔ **2^h - 1**
- What is the maximum number of nodes that a 2-3 tree of height h can have?
  - ➔ **3^h - 1**

Properties of 2-3 Trees

- N= The number of nodes in a 2-3 tree.
- h = The height of a 2-3 tree.

➔ $2^h - 1 \leq N \leq 3^h - 1$

➔ $\log_3 (N+1) \leq h \leq \log_2 (N+1)$

➔ Lower Bound: $h = \Omega(\log N)$

➔ Upper Bound: $h = O(\log N)$

➔ **$h = \Theta(\log N)$**

Properties of 2-3 Trees

The height of a 2-3 tree with n nodes is Θ (log N).

2-3 Tree Operations -Analysis

- Search
  - ➔**O(log N)** worst-case
- Insert
  - ➔**Two pass: root-to-leaf & leaf-to-root (split)**
  - ➔**O(log N)** worst-case
- Delete
  - ➔Two pass: root-to-leaf & leaf-to-root **(merge)**
  - ➔**O(log N)** worst-case

2-3 Tree Visualization

- ***2-3 Tree Visualization***

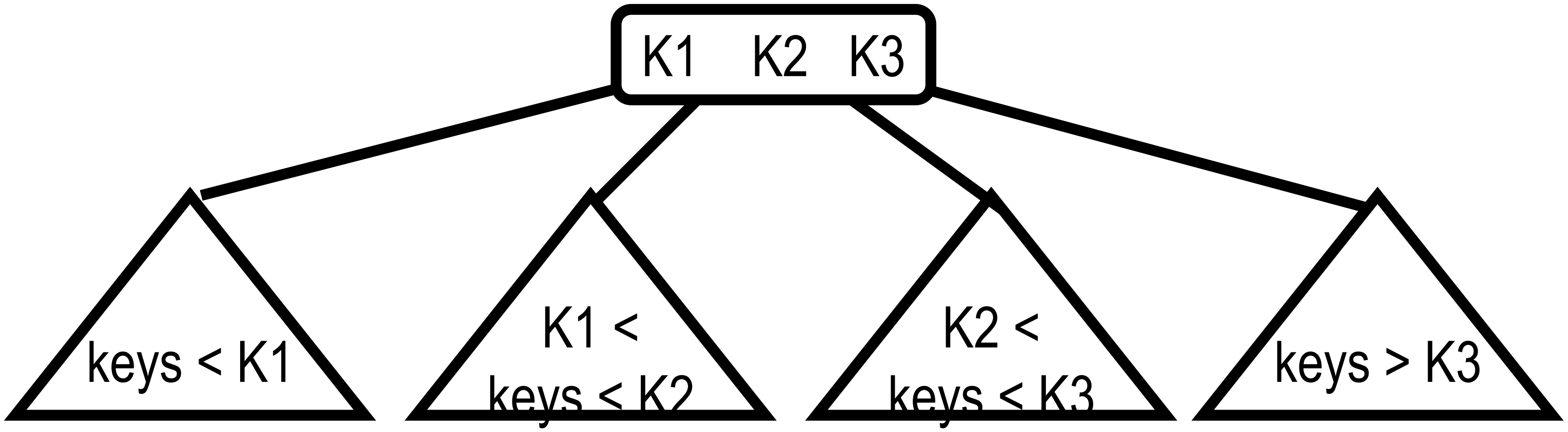
✓

2-3-4 Trees
(2-4 Trees)

What is a 2-3-4 (Search) Tree?

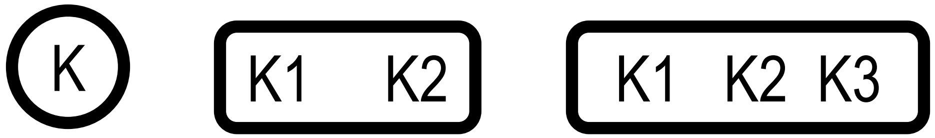
- A **search tree** s.t.
  - Each nonleaf node has either two (2-node), three (3-node) or four (**4-node**) children
  - **All leaves are at the same level (depth).**

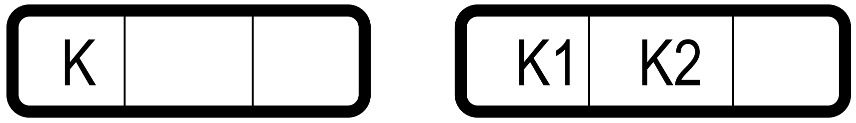
4-Node



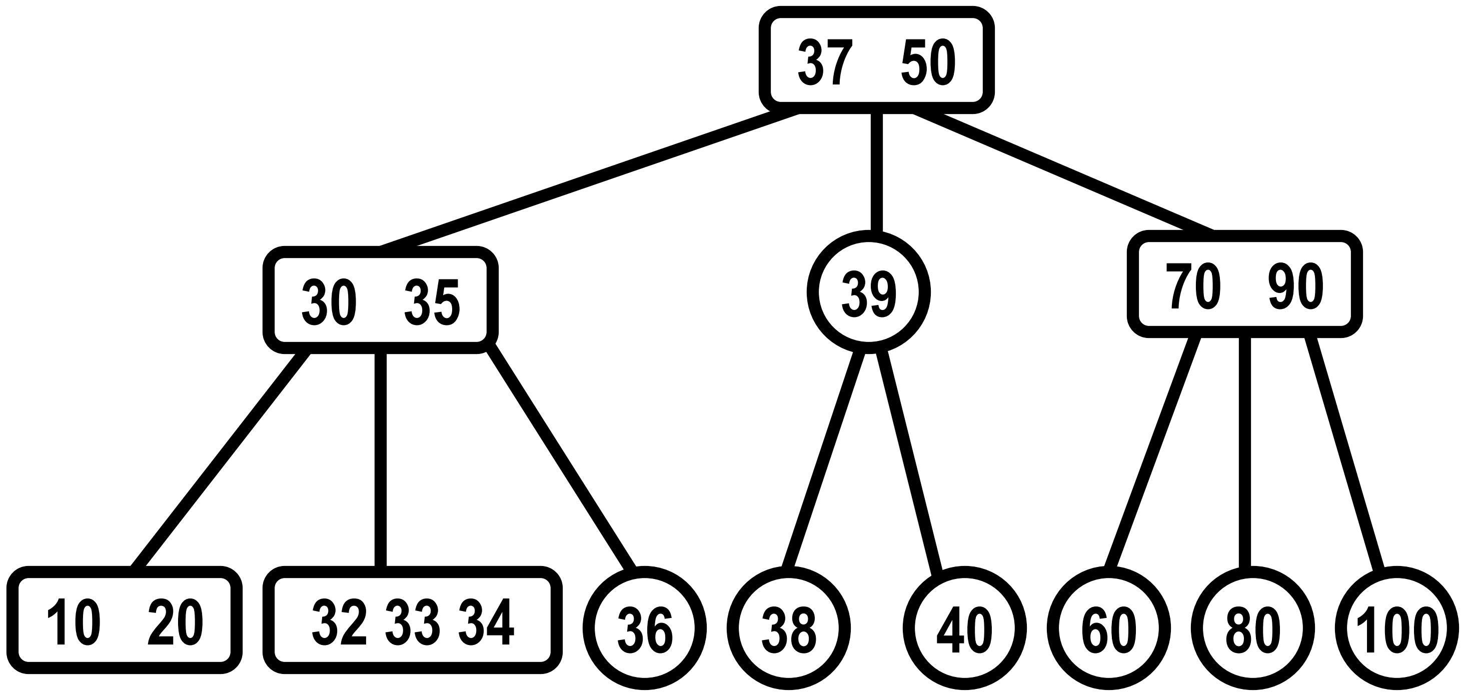
Leaf

- A leaf may contain either one or two or three data items.

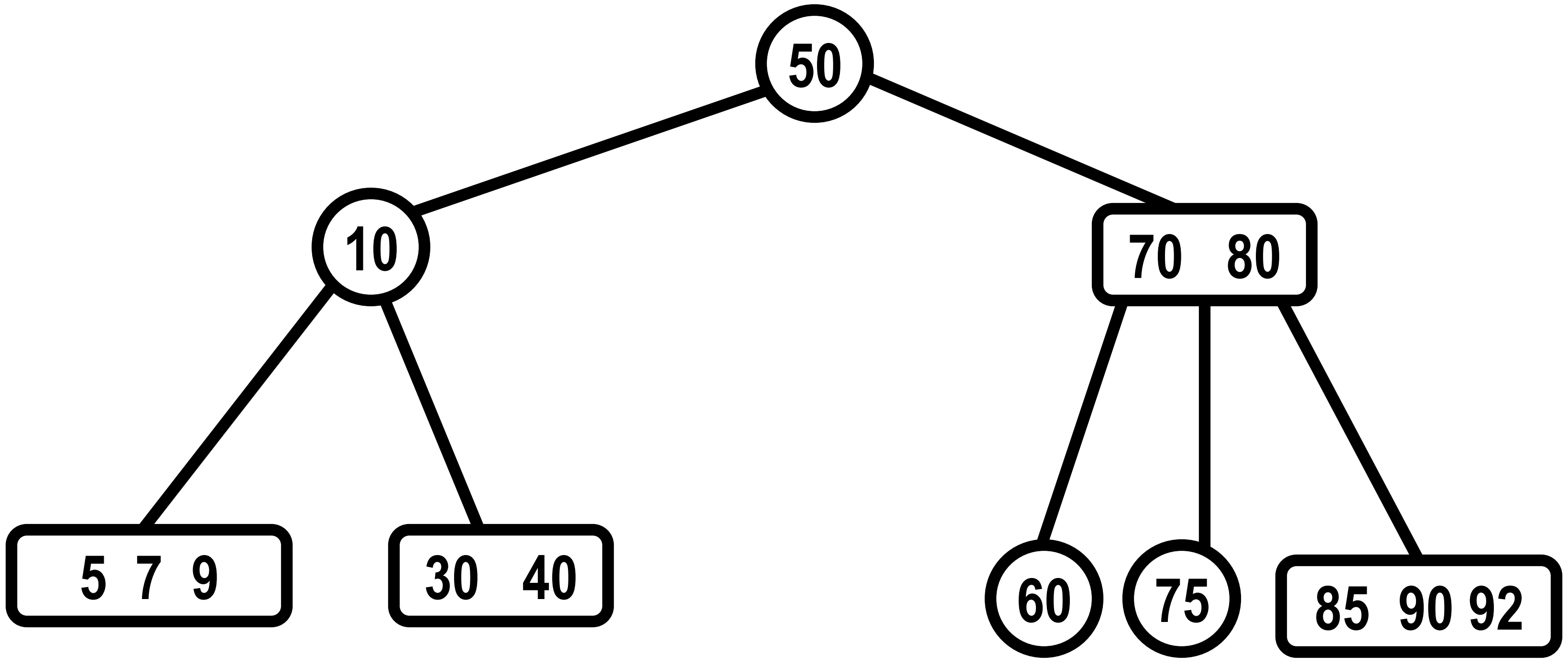




Example: A 2-3-4 tree?



Example: A 2-3-4 tree?



Searching a 2-3-4 tree

- Similar to the search operation for a 2-3 tree.

Inserting Items into a 2-3-4 Tree

- **Approach 1:**
  - Split & Promote a single Median!

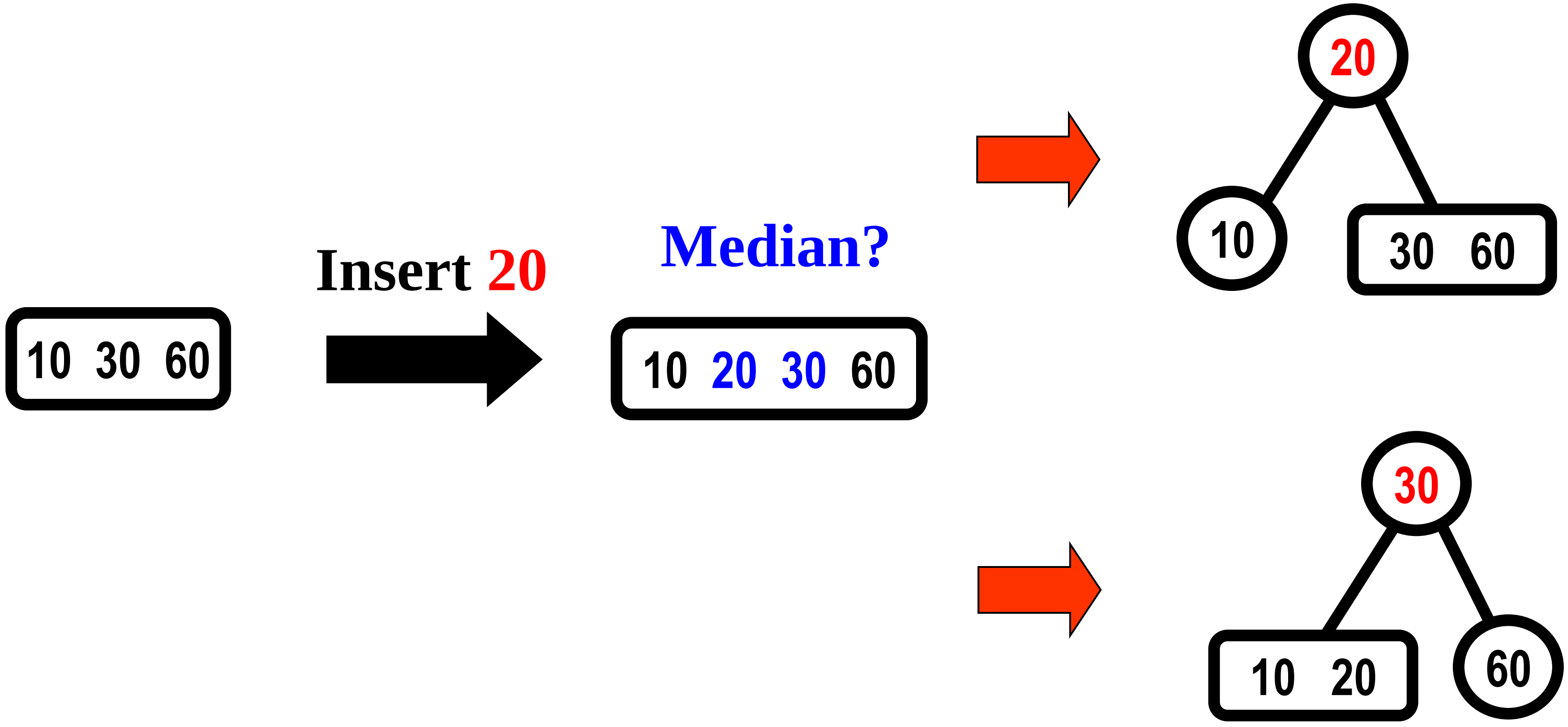
    (Like 2-3 Tree Insert)

  - **Two-pass process**

Example: Inserting Items into a 2-3-4 Tree

Insert: 20, 50, 40, 70, 80, 15, 90, 100

Example: Inserting Items into a 2-3-4 Tree

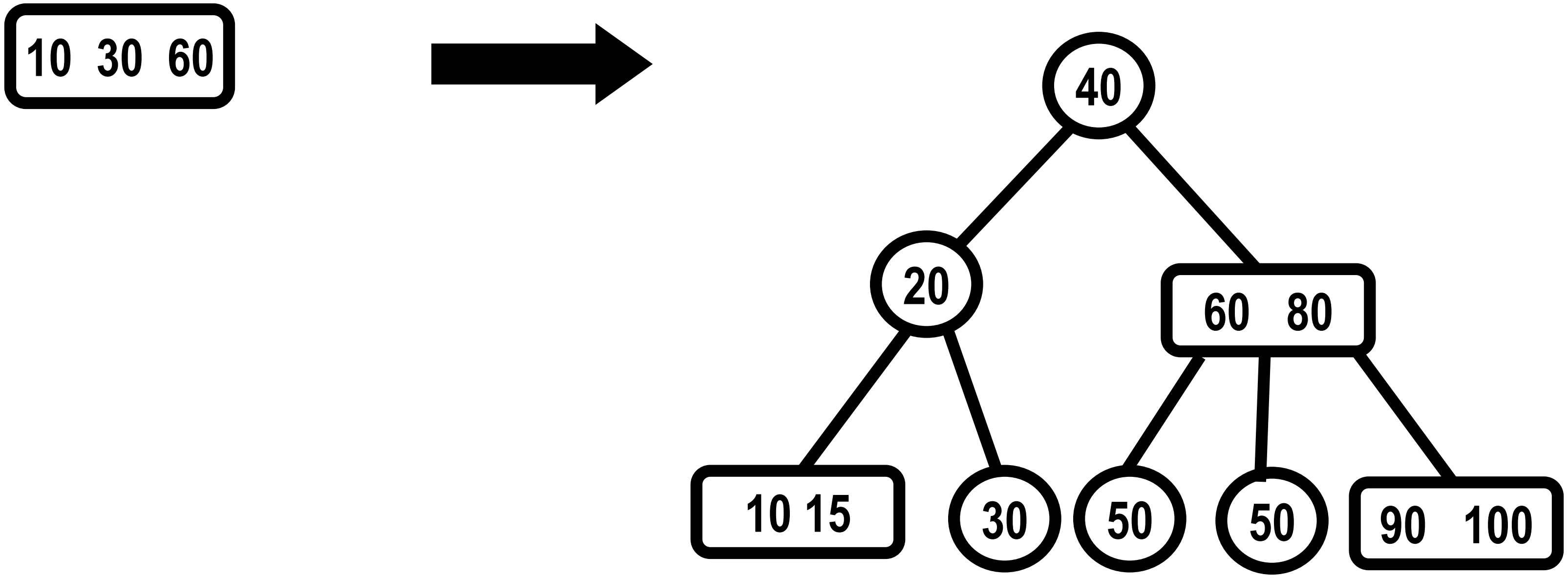


►QUIZ? Inserting Items into a 2-3-4 Tree

Insert: 20, 50, 40, 70, 80, 15, 90, 100

►QUIZ? Inserting Items into a 2-3-4 Tree

Insert: 20, 50, 40, 70, 80, 15, 90, 100



Inserting Items into a 2-3-4 Tree

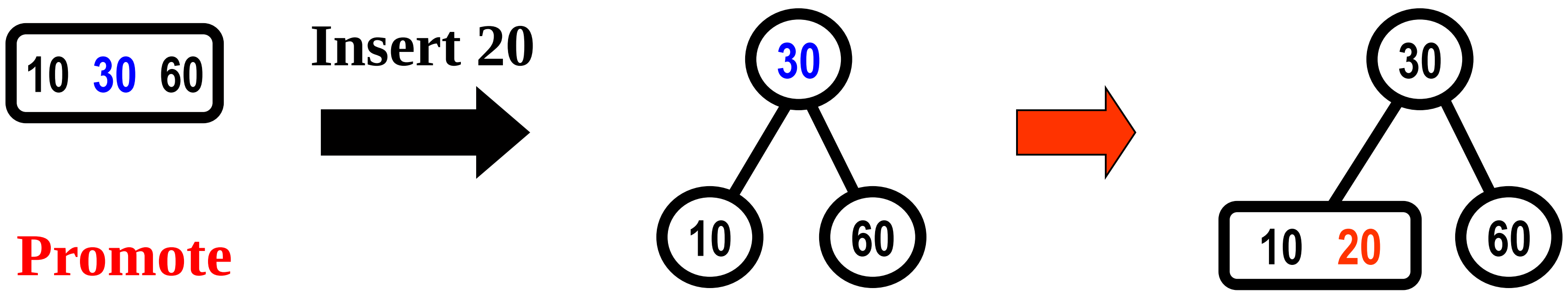
- **Approach 2:**
  - **Pre-Split** & Promote the Median!
  - **One-pass process**

Example: Inserting Items into a 2-3-4 Tree with Pre-Split

Insert: 20, 50, 40, 70, 80, 15, 90, 100

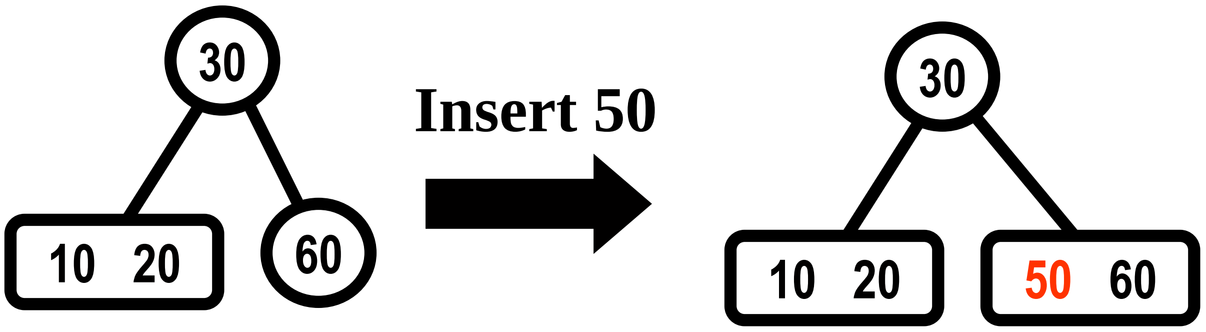
Example: Inserting Items into a 2-3-4 Tree with Pre-Split

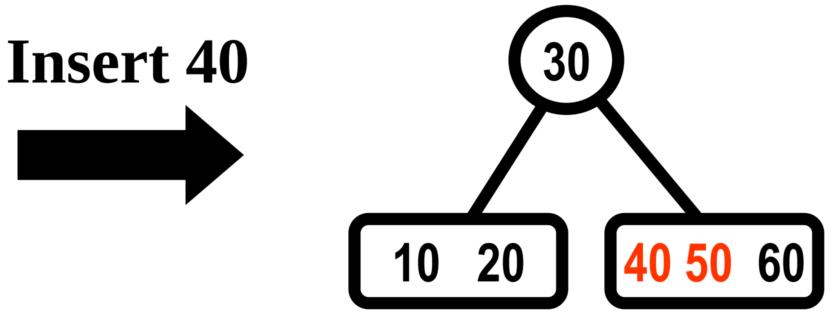
Pre-Splitting!



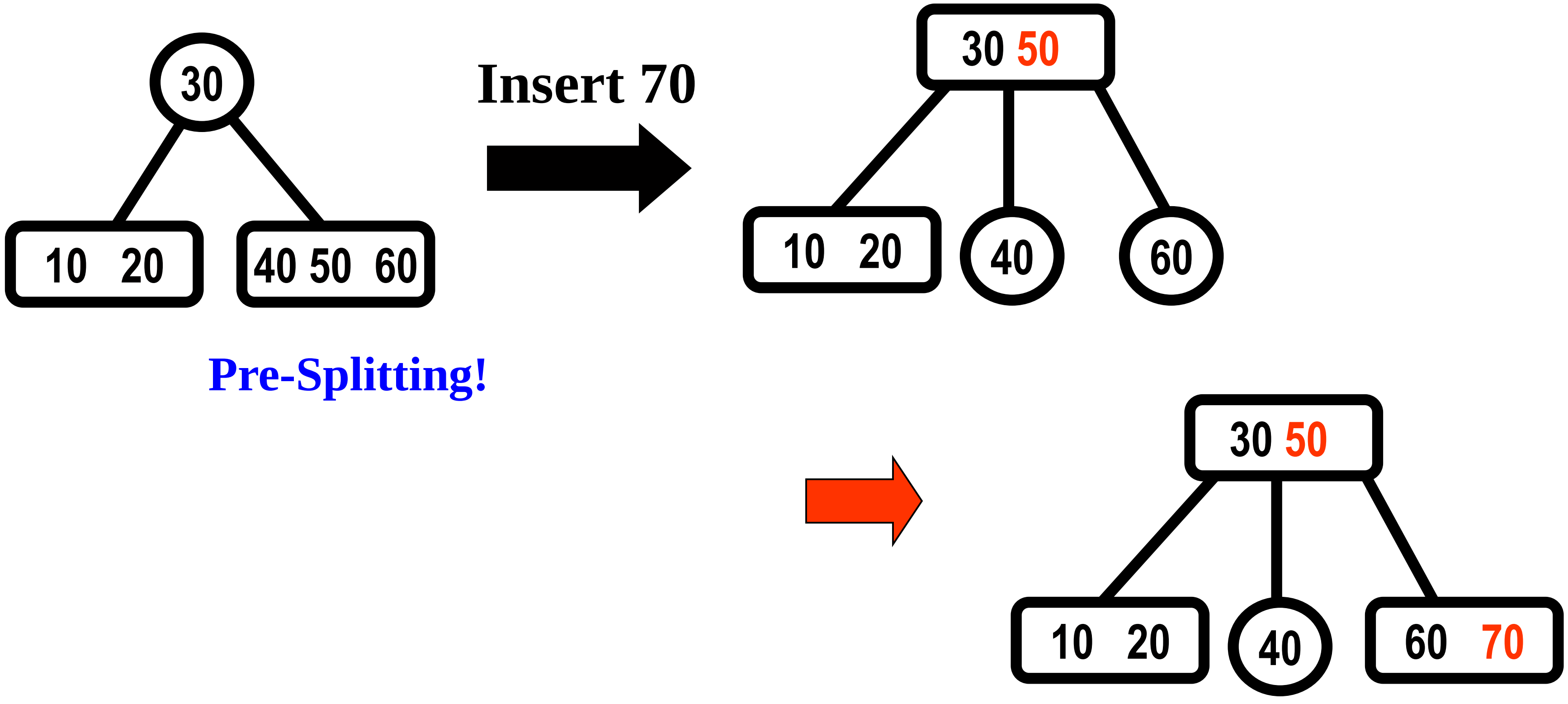
& Promote the Median!

Example: Inserting Items into a 2-3-4 Tree with Pre-Split

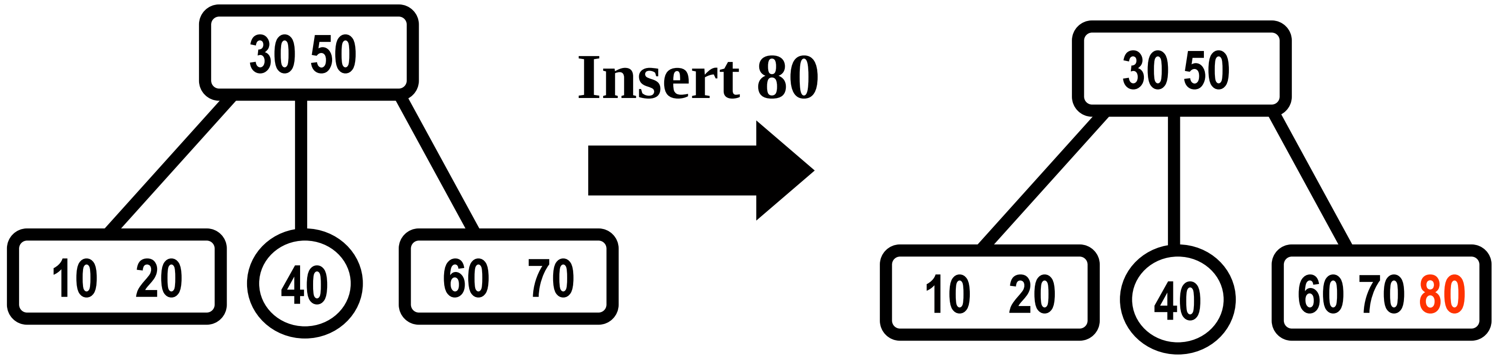




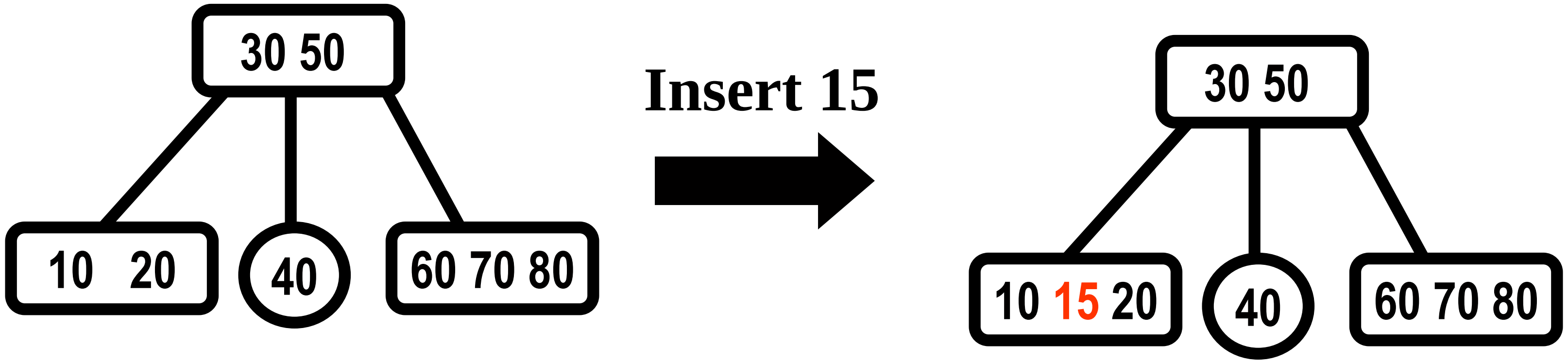
Example: Inserting Items into a 2-3-4 Tree with Pre-Split



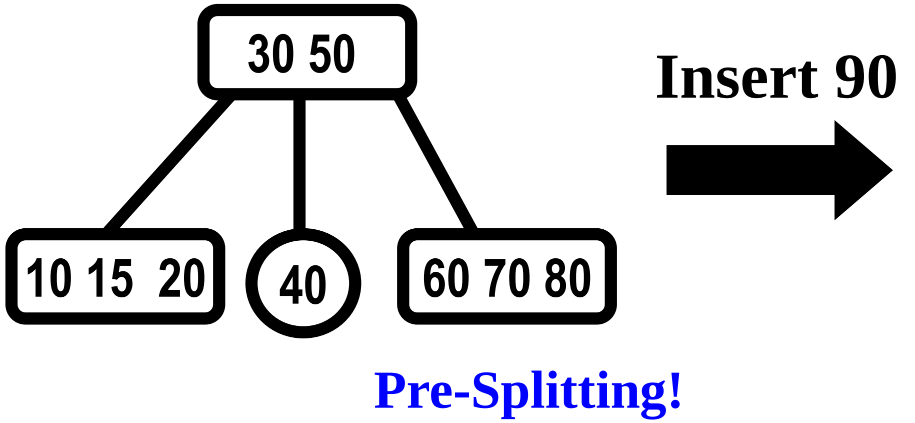
Example: Inserting Items into a 2-3-4 Tree with Pre-Split

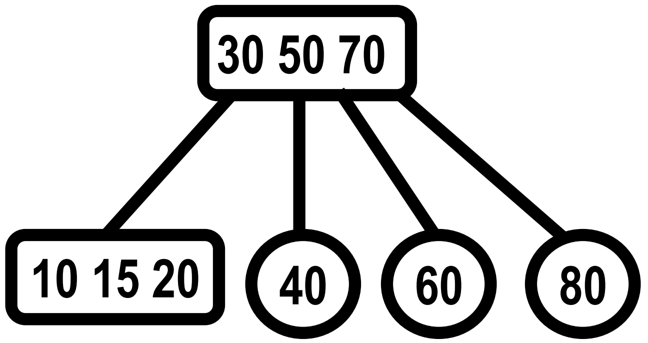


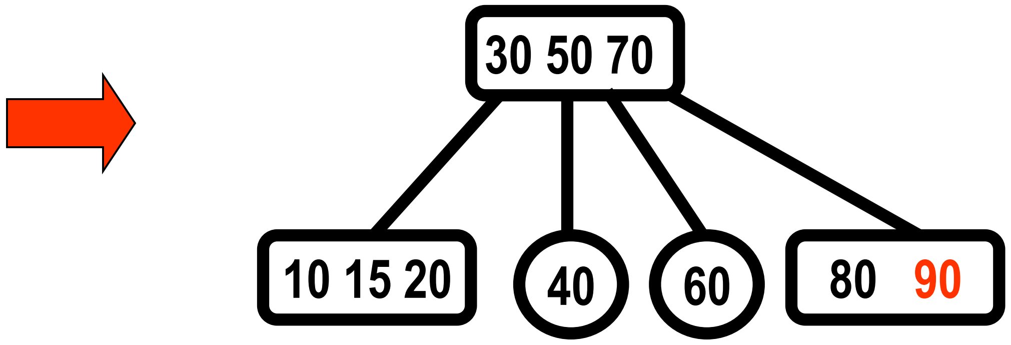
Example: Inserting Items into a 2-3-4 Tree with Pre-Split



Example: Inserting Items into a 2-3-4 Tree with Pre-Split



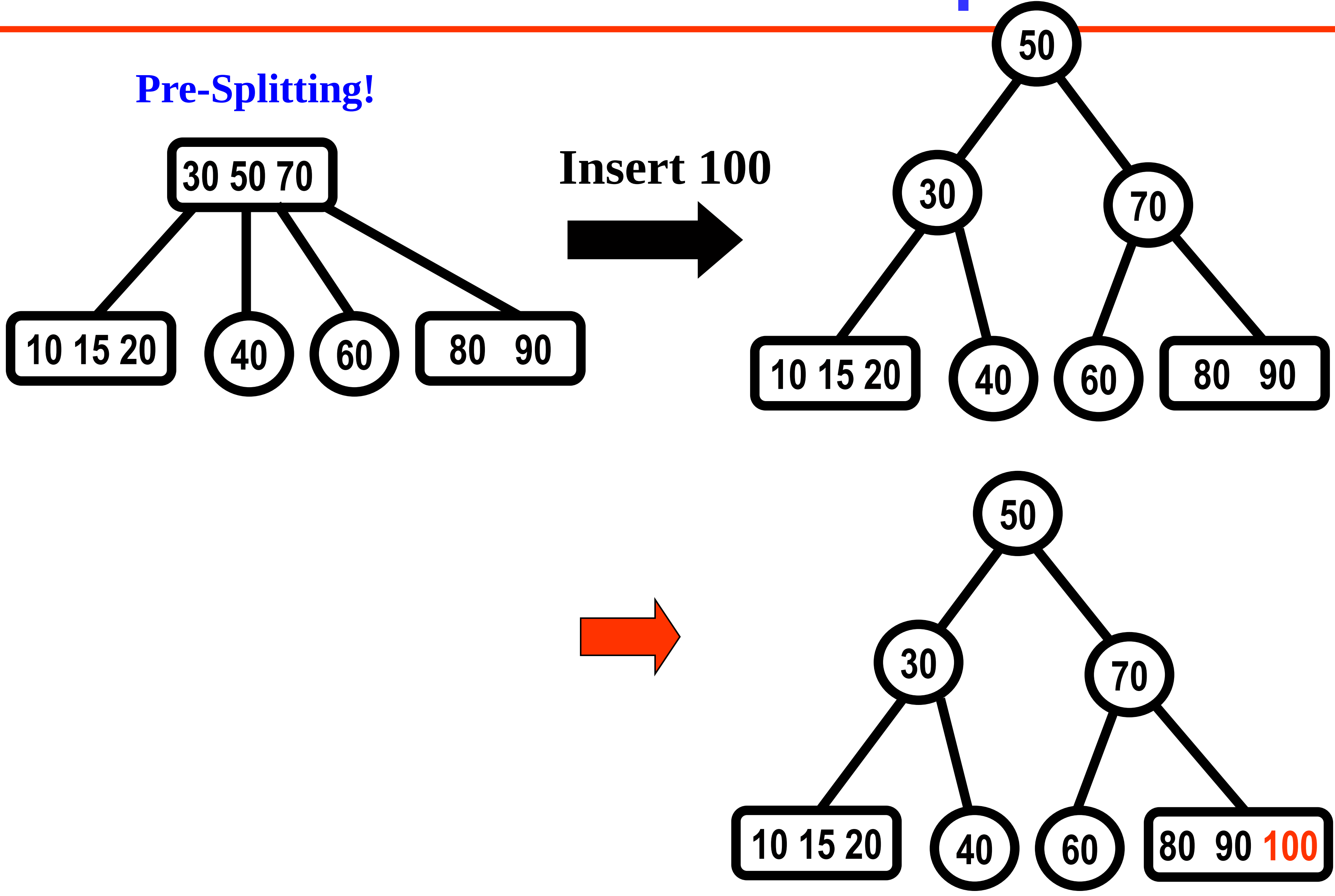




Example: Inserting Items into a 2-3-4 Tree with Pre-Split

Pre-Splitting!

Insert 100



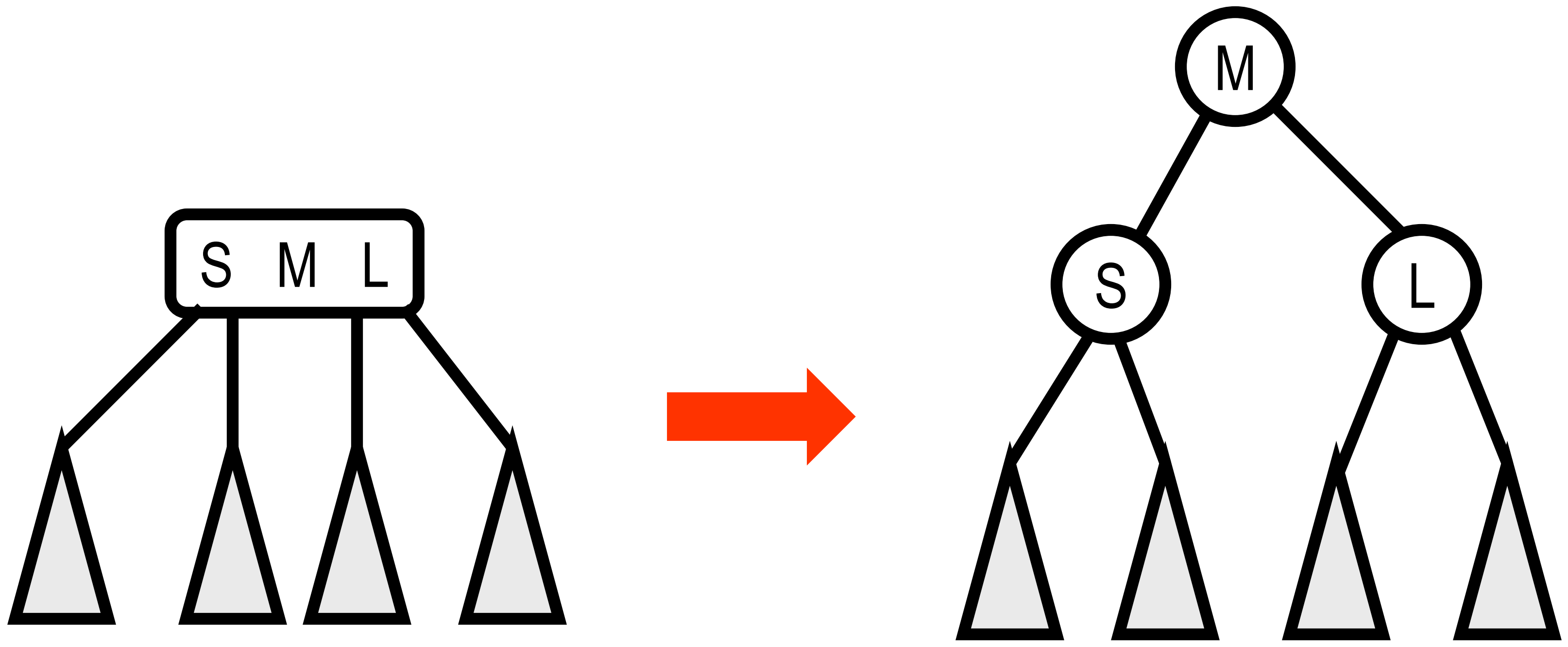
Insert Operation

- Insert(234T, NewItem):
  - Search the leaf L in which the key of NewItem belongs.
  - **Split each 4-node as soon as we encounter it** during the search from the root to a leaf that will accommodate the new item.
    - No 4-nodes!
  - Add NewItem to L.
    - No need to split through leaf-to-root pass!
    - **Preemtive-splitting!!!**

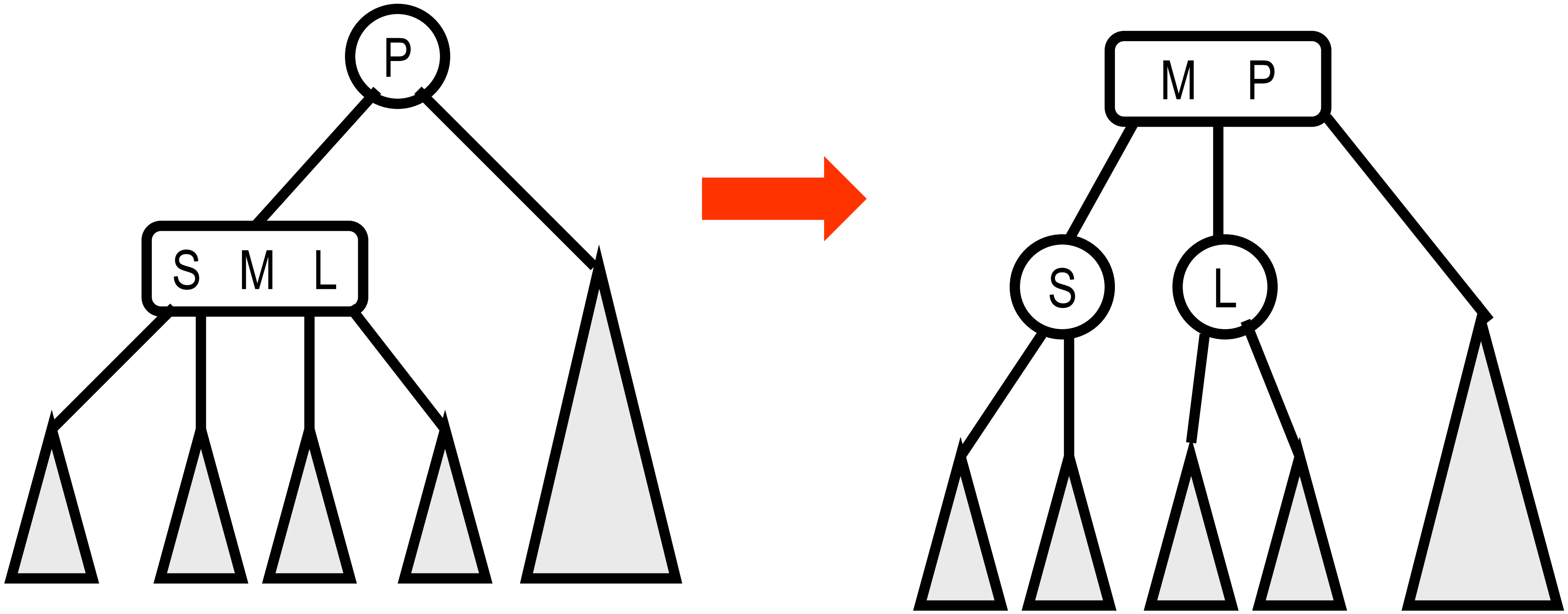
Pre-Splitting 4-Nodes

- The 4-node N will be
  - N = the root
  - N has a 2-node parent
  - N as a 3-node parent

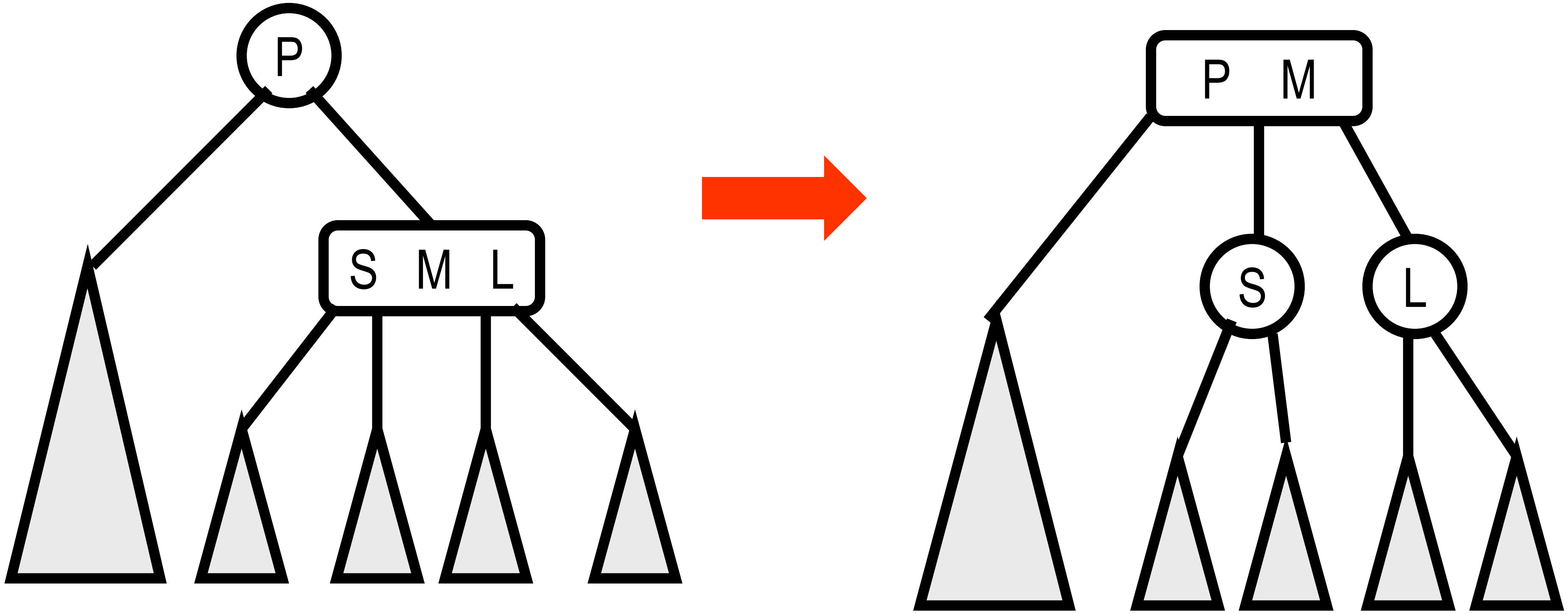
1. Pre-Splitting a 4-node root



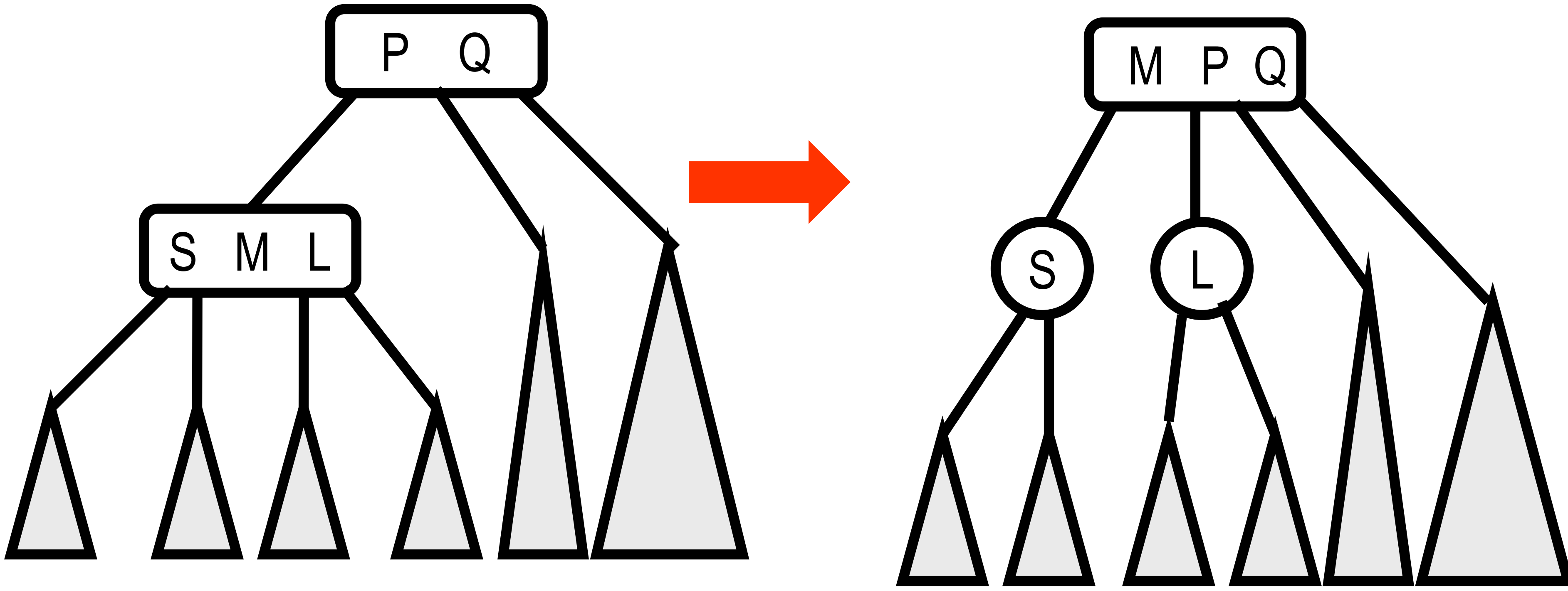
2. Pre-Splitting a 4-node with a 2-Node Parent



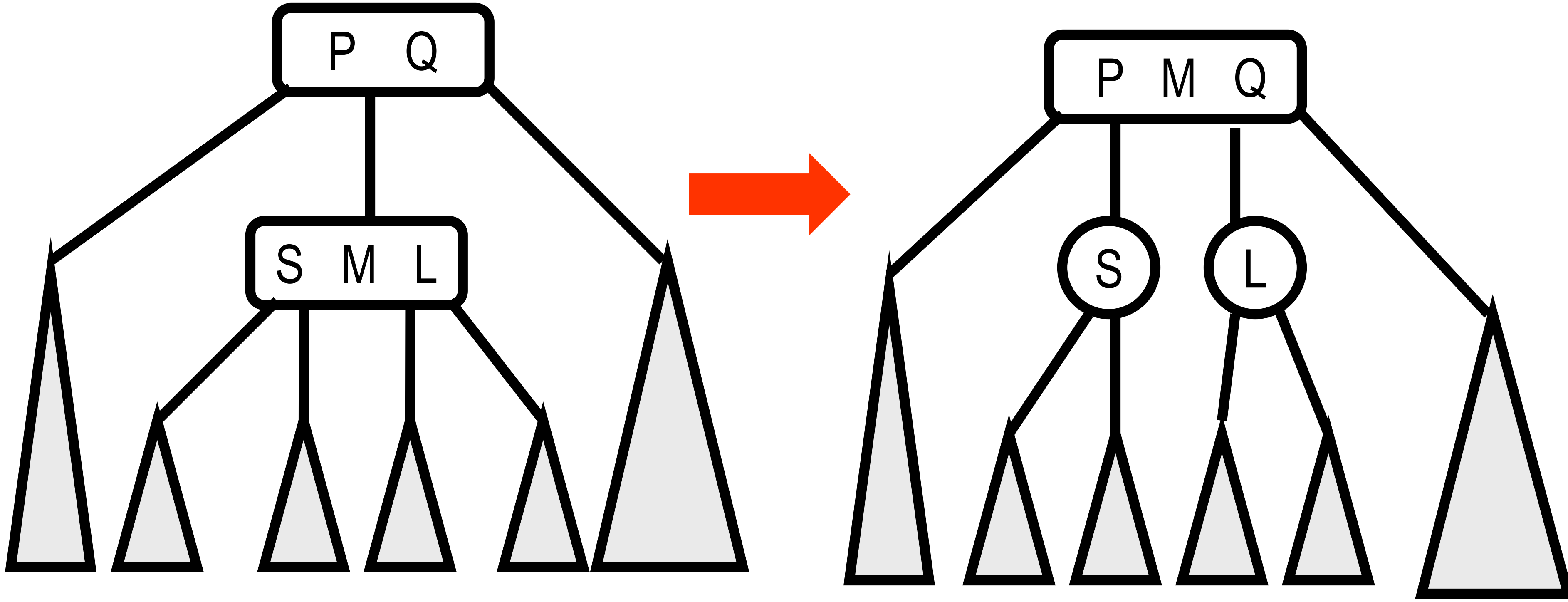
2. Pre-Splitting a 4-node with a 2-Node Parent



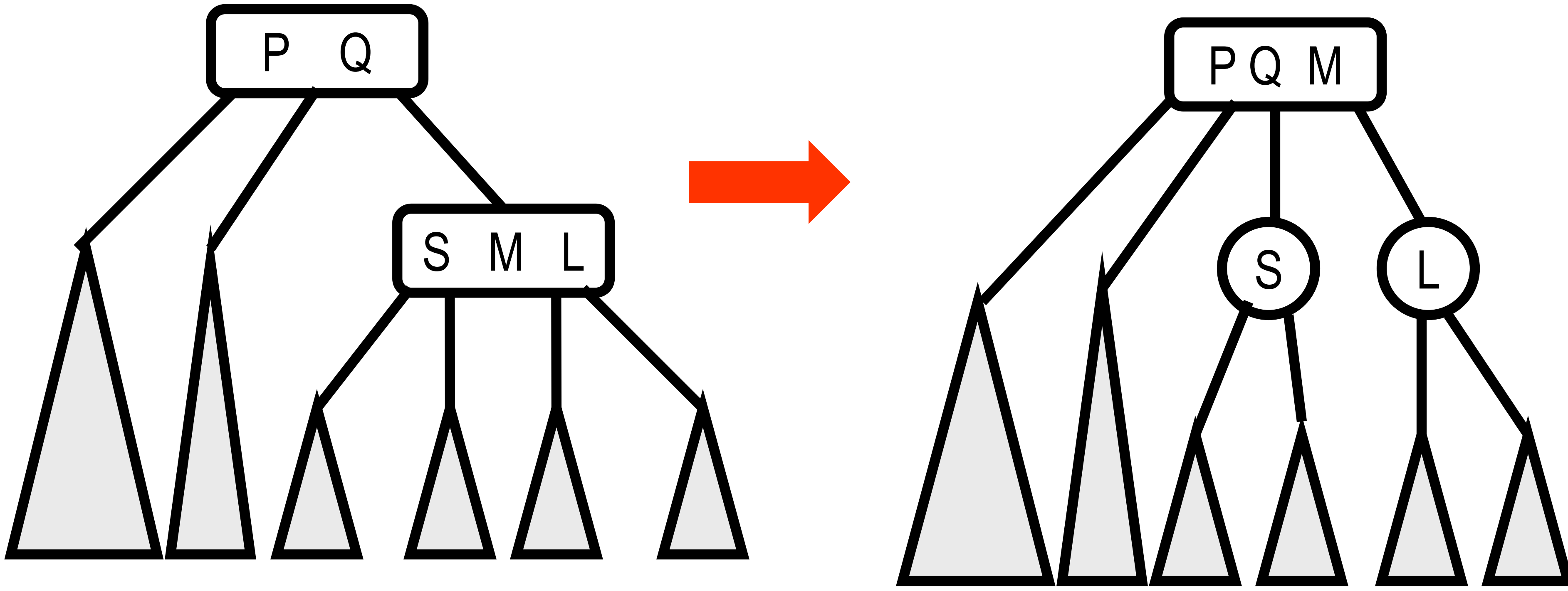
3. Pre-Splitting a 4-node with a 3-Node Parent



3. Pre-Splitting a 4-node with a 3-Node Parent

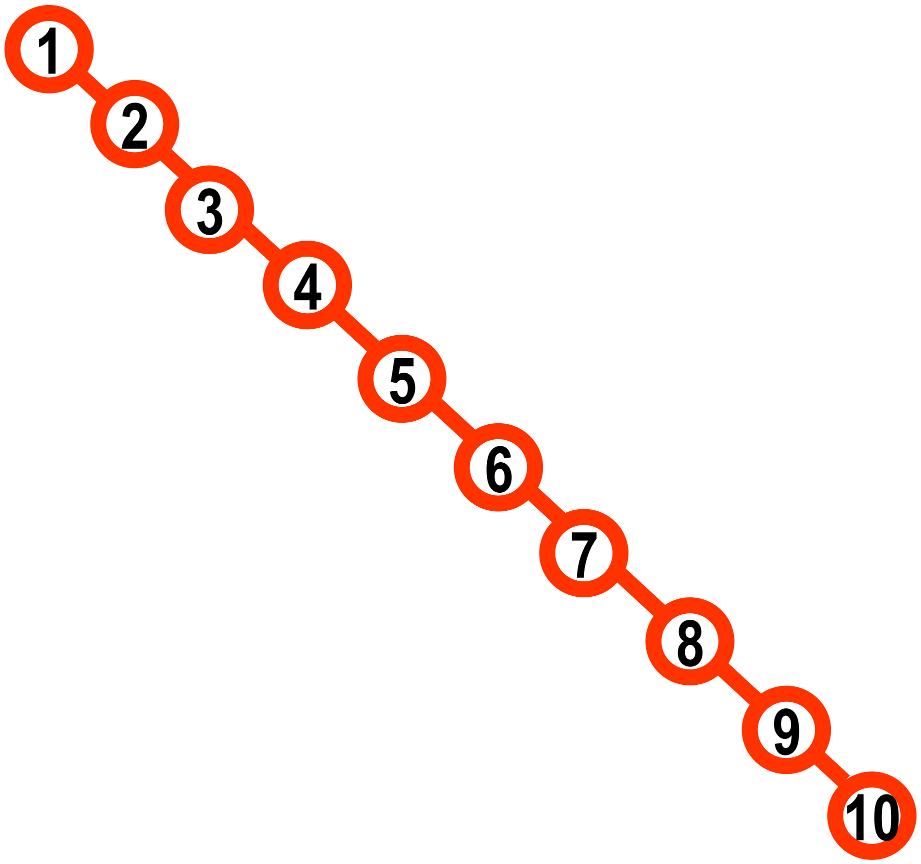


3. Pre-Splitting a 4-node with a 3-Node Parent



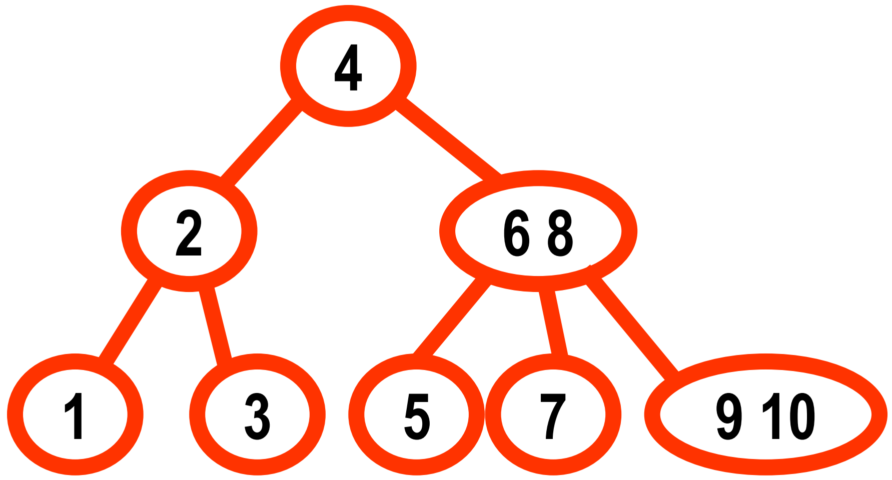
►QUIZ? Binary Search Tree?

Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10



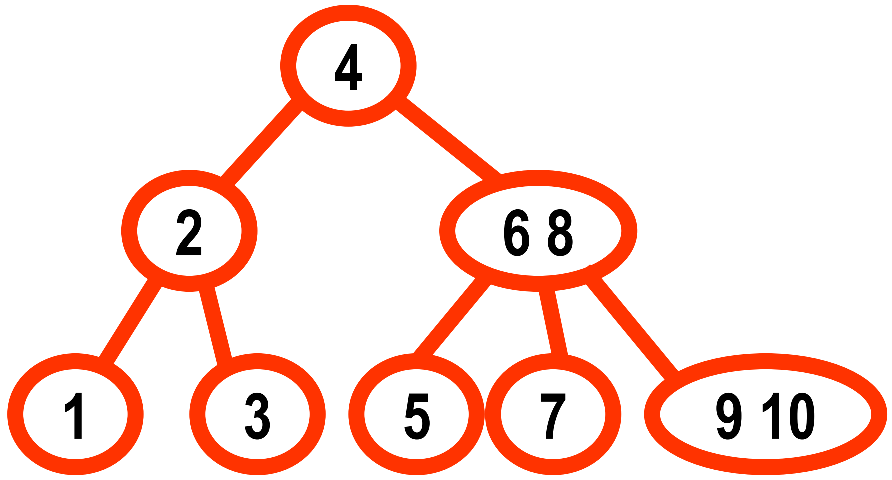
►QUIZ? 2-3-4 Tree without Pre-Split

Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10



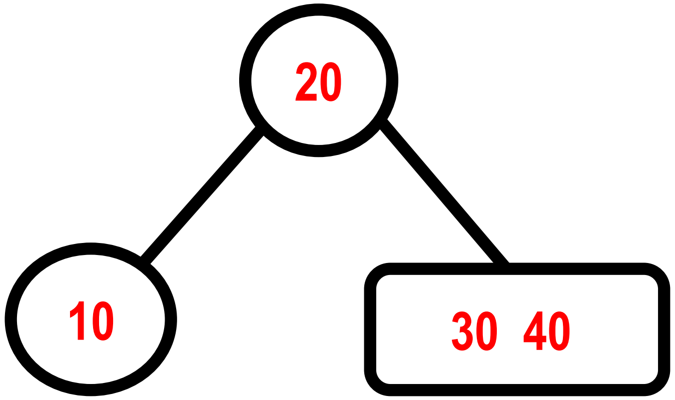
►QUIZ? 2-3-4 Tree with Pre-Split

Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10



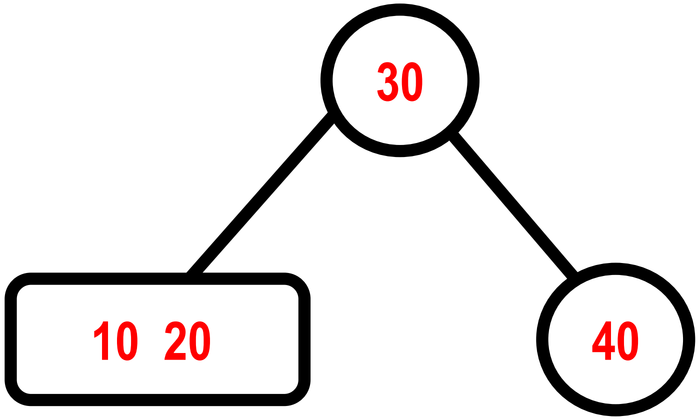
►QUIZ? 2-3-4Tree without Pre-Split

Insert 20, 30, 40 and 10



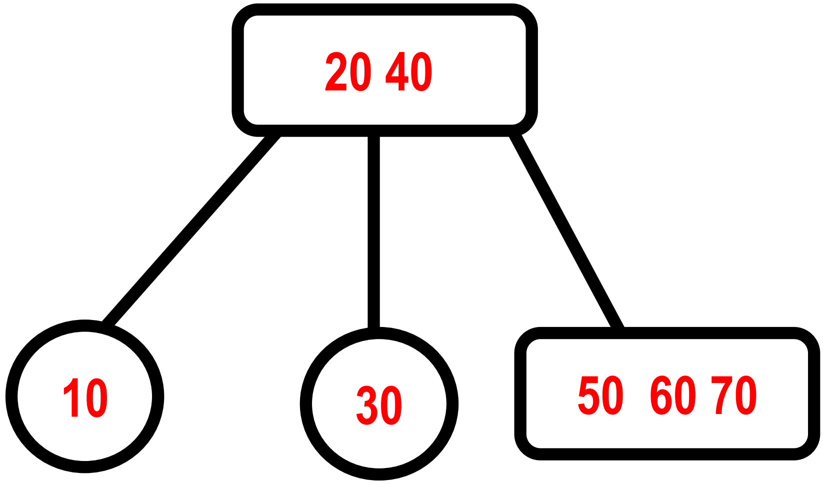
►QUIZ? 2-3-4Tree with Pre-Split

Insert 20, 30, 40 and 10



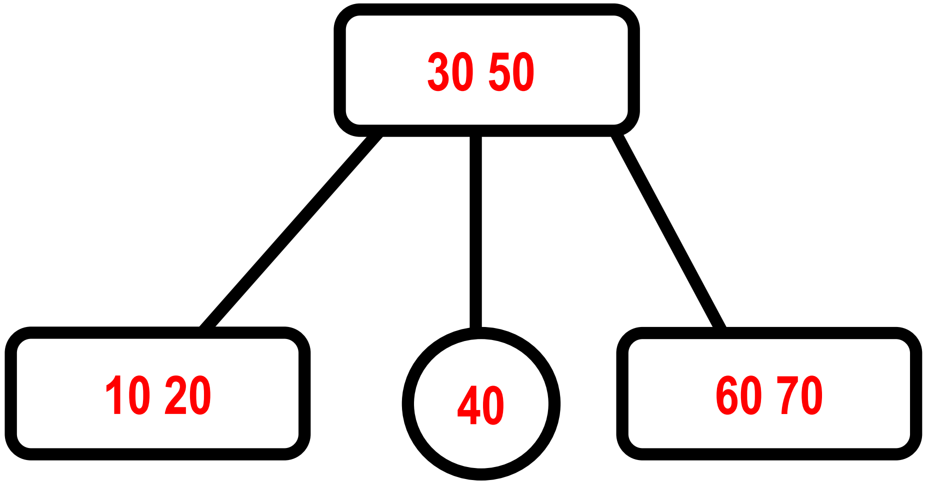
►QUIZ? 2-3-4Tree without Pre-Split

Insert 20, 30, 40, 10, 50, 60 and 70



►QUIZ? 2-3-4Tree with Pre-Split

Insert 20, 30, 40, 10, 50, 60 and 70



Inserting Items into a 2-3-4 Tree

- **Approach 1:**
  - Split & Promote a single Median! (Like 2-3 Tree Insert)
  - **Two-pass process**
- **Approach 2:**
  - **Pre-Split** & Promote the Median!
  - **One-pass process**

►QUIZ?

- **Insert** to a 2-3-4 trees?

Properties of 2-3-4 Trees

- What is the minimum number of nodes that a 2-3-4 tree of height h can have?
  - ➔ $2^h - 1$
- What is the maximum number of nodes that a 2-3-4 tree of height h can have?
  - ➔ $4^h - 1$

Properties of 2-3-4 Trees

- N= The number of nodes in a 2-3-4 tree.
- h = The height of a 2-3-4 tree.

➔ $2^h - 1 \leq N \leq 4^h - 1$

➔ $\log_4 (N+1) \leq h \leq \log_2 (N+1)$

➔ Lower Bound: $h = \Omega(\log N)$

➔ Upper Bound: $h = O(\log N)$

➔ **$h = \Theta(\log N)$**

Properties of 2-3-4 Trees

The height of a 2-3-4 tree with n nodes is

Θ (log N).

2-3-4 Tree Operations -Analysis

- Search
  - ➔ **O(log N)** worst-case
- Insert
  - ➔ **One pass: root-to-leaf (pre-emptive split)**
  - ➔ **O(log N)** worst-case
- Delete
  - ➔ **One pass: root-to-leaf (merge)**
  - ➔ **O(log N)** worst-case

2-3-4 Tree Visualization

- ***2-3-4 Tree Visualization***

✓

Red-Black Trees

(Height-Balanced Binary Search Trees)

2-3-4 (Search) Tree

- A 2-3-4 tree requires more storage than a binary search tree.
- A more efficient representation?
  - ➔ As a binary tree!

- **Can we get 2-3-4 tree advantages in a binary tree format?**

Red Black (Search) Tree

- A **red-black tree** is a **binary search tree** st.
  - Every node is colored either red or black.
  - The root is always black.
  - There are no two adjacent red nodes. If a node is red, then both its children must be black.
  - Every path from root to a NULL node has **the same number of black nodes**.

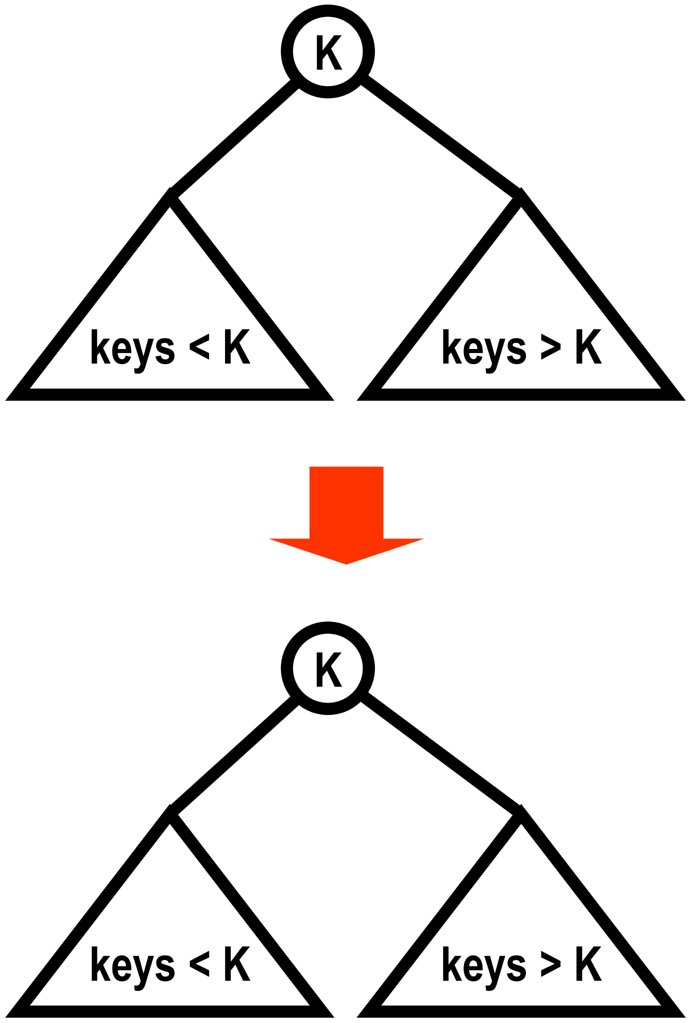
Red-Black Tree

- **Designed to represent 2-3-4 tree without the additional link overhead**.
- A red-black tree is a special binary search tree.
  - **A representation of a 2-3-4 tree as a binary tree** whose nodes are colored red or black.
  - A red-black tree requires less storage than a 2-3-4 tree.

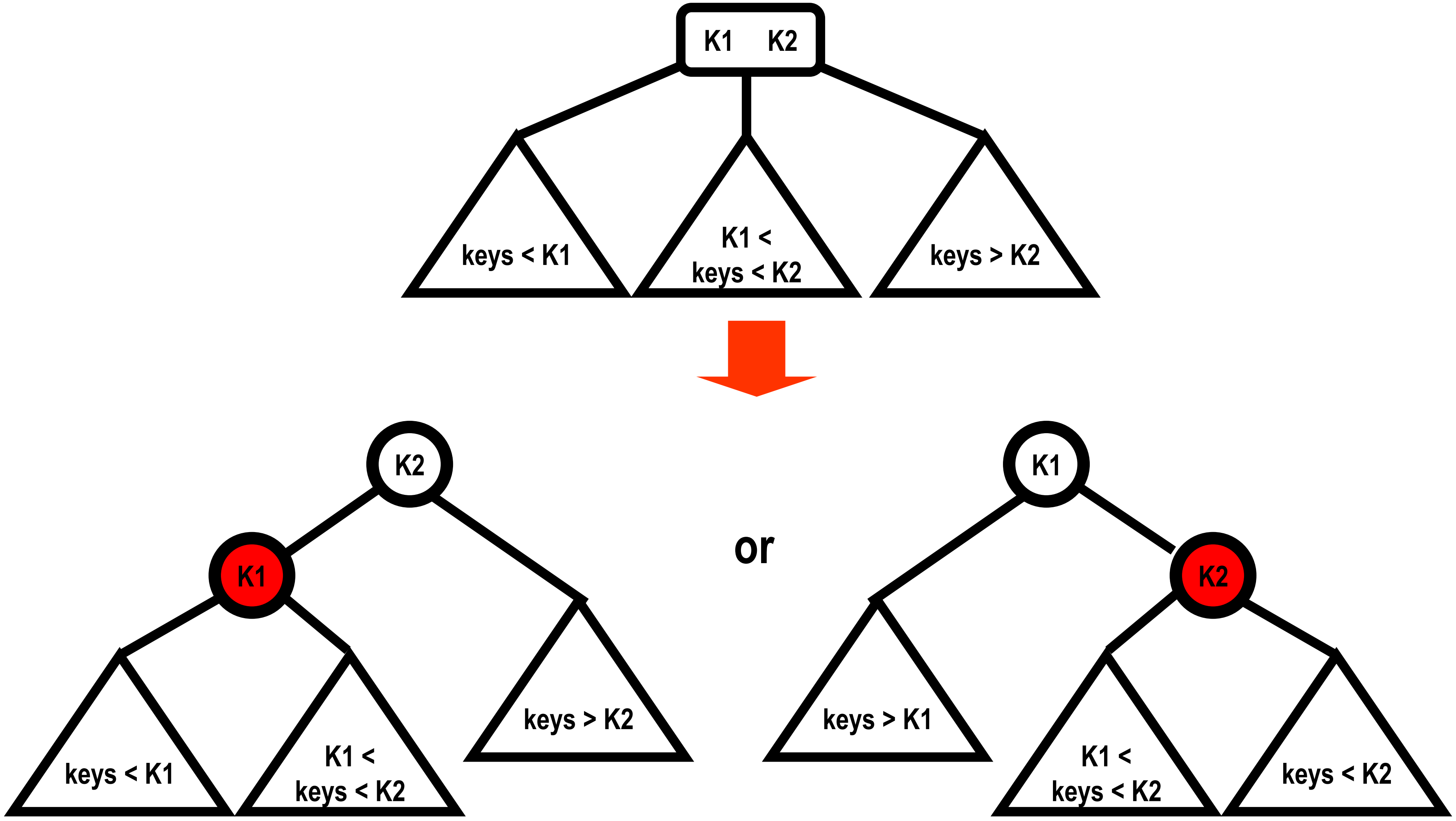
Red-Black Tree vs 2-3-4 Tree

- RB Trees represent 2-3-4 trees without the additional link overhead!
- Idea?
  - Red nodes represent the extra keys in 3-nodes and 4-nodes.
  - Red-black trees are not unique, but the corresponding 2-3-4 tree is unique!

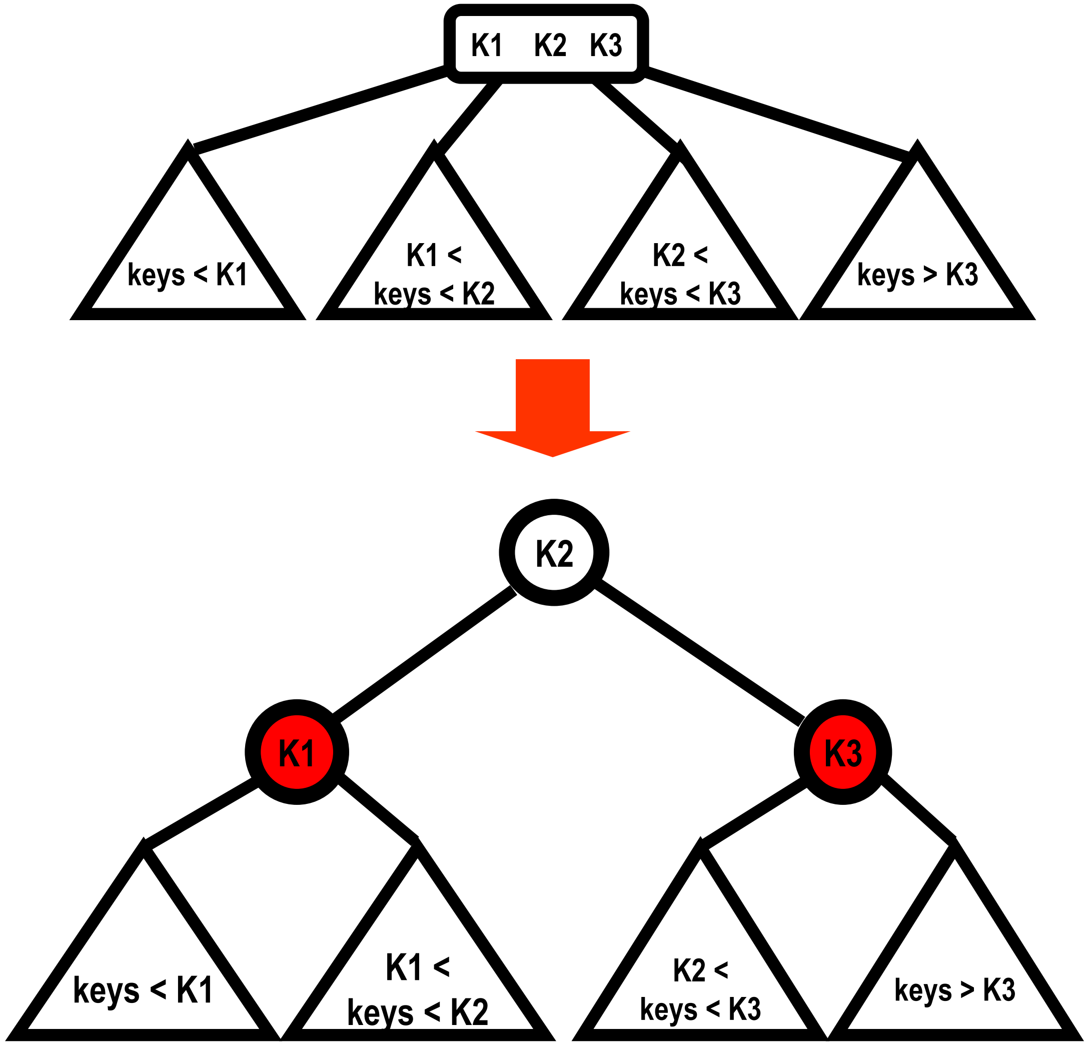
Representation of 2 Nodes



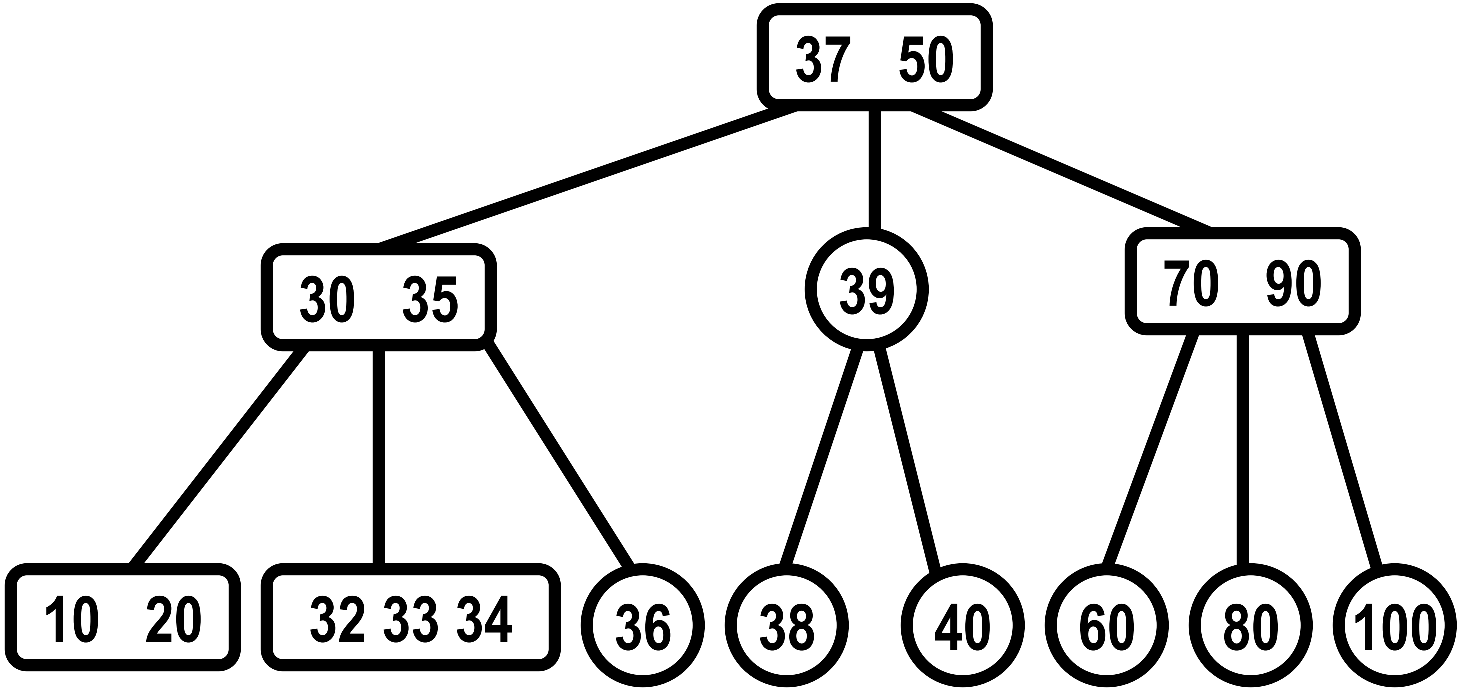
Representation of 3 Nodes



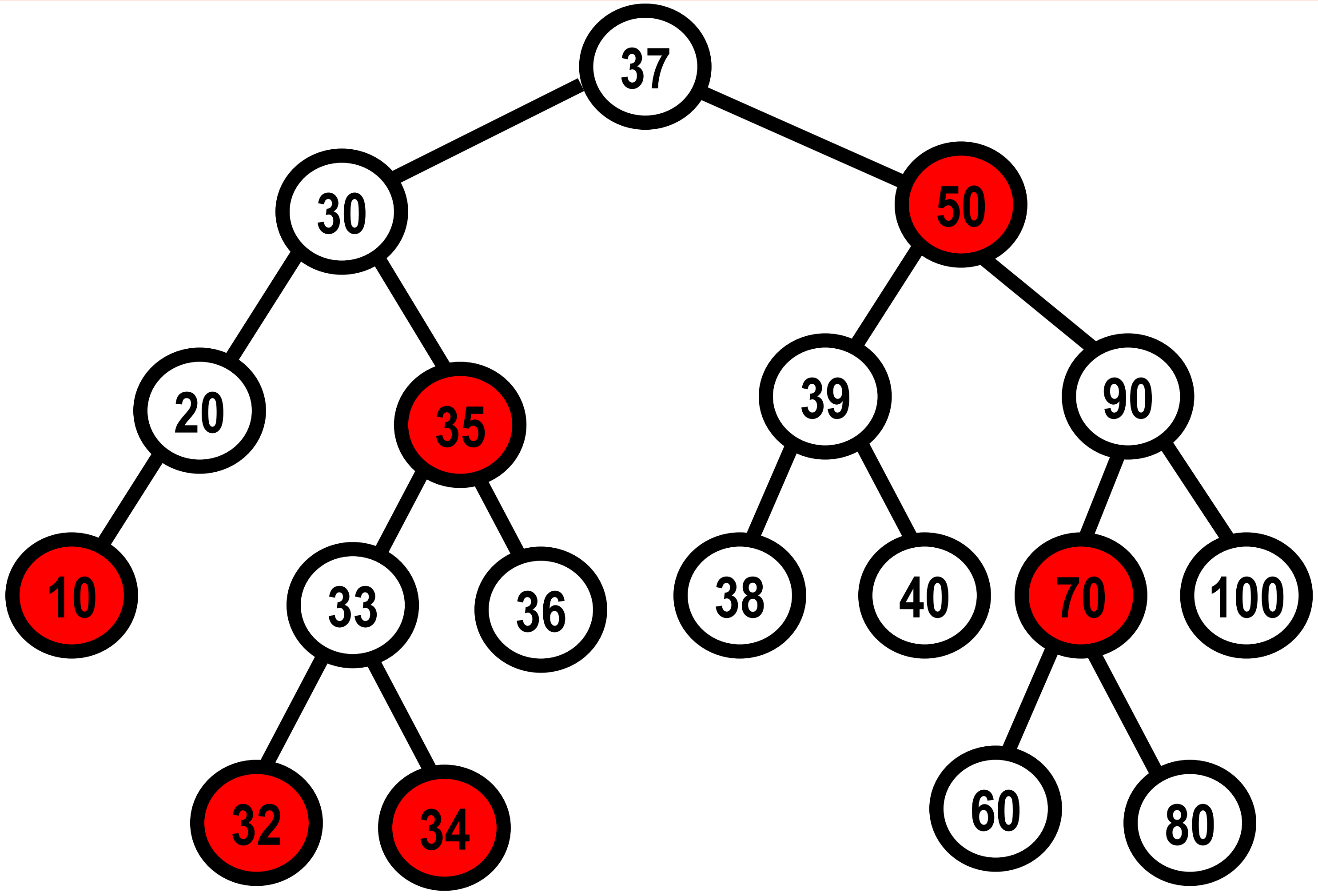
Representation of 4 Nodes



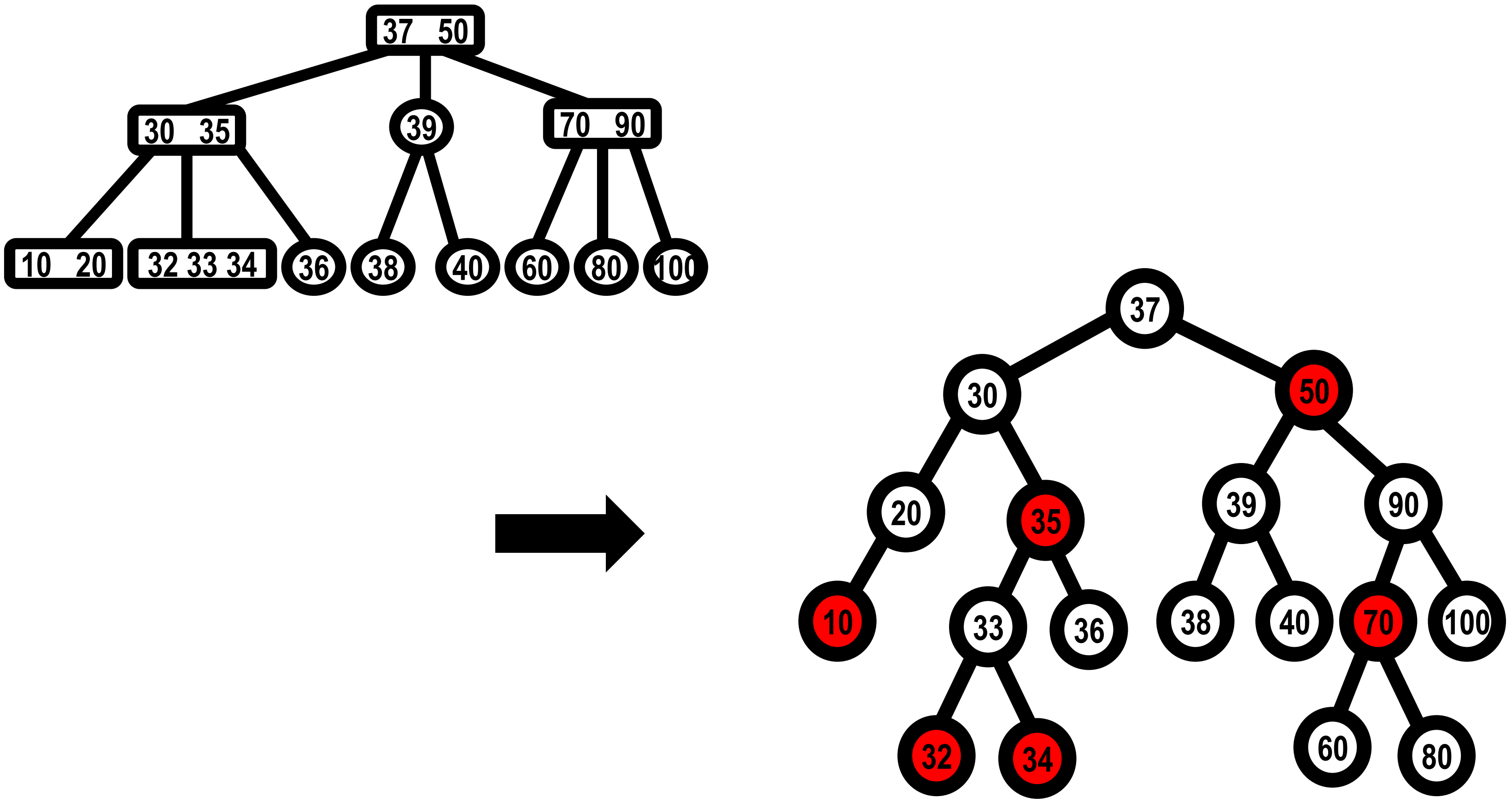
Example: A 2-3-4 tree



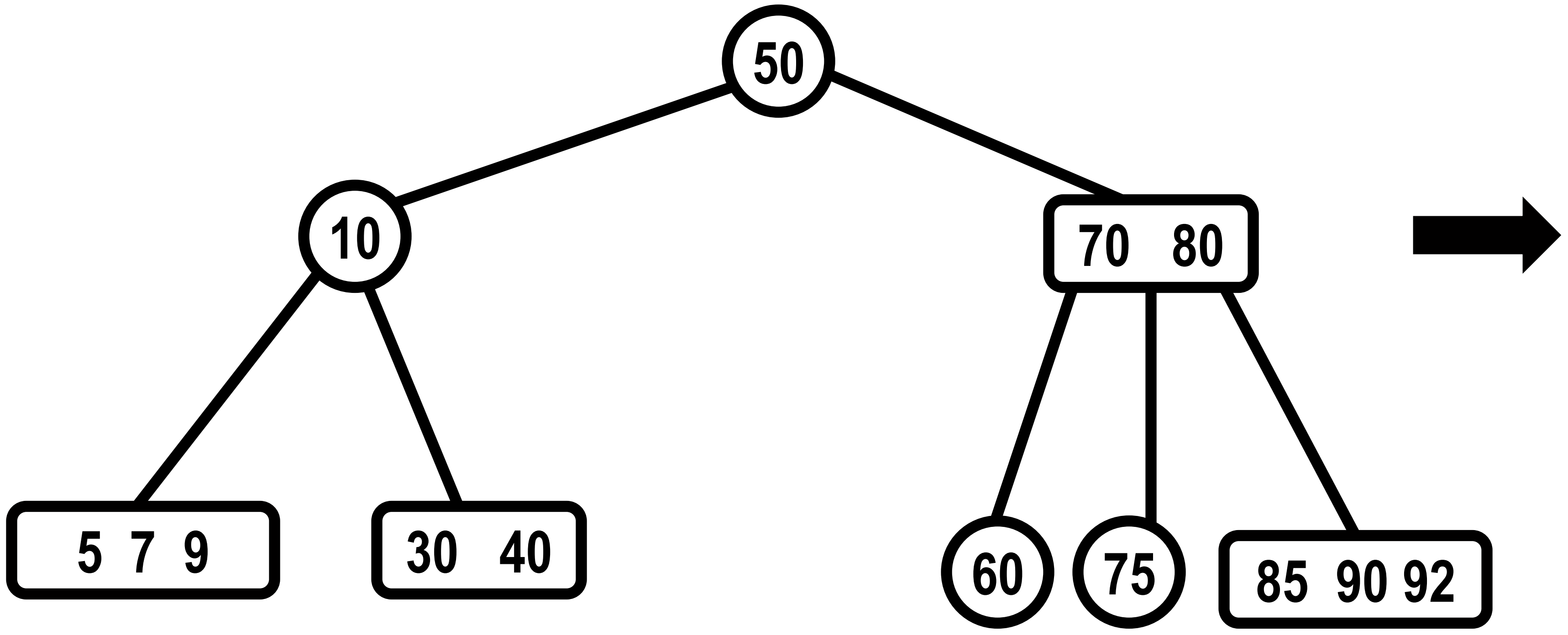
Example: Red-Black Tree?



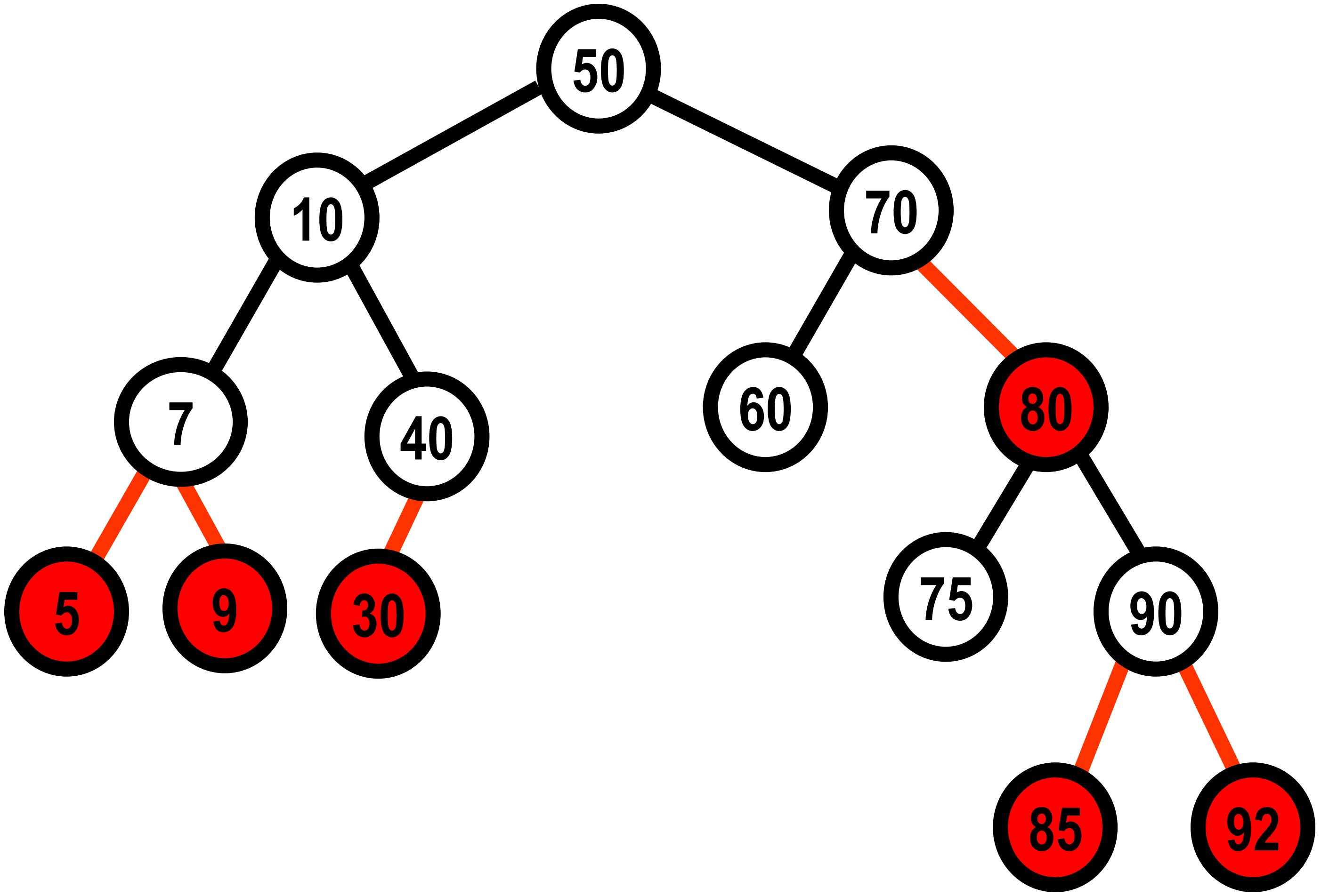
Example: A 2-3-4 tree vs A Red-Black Tree



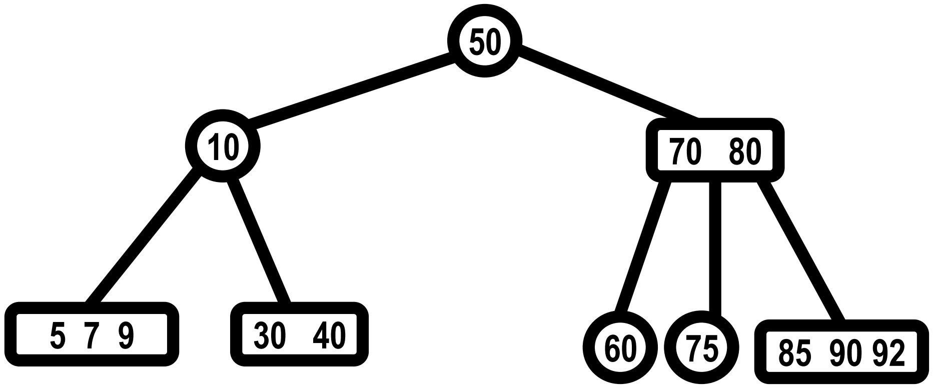
►QUIZ? A 2-3-4 tree?

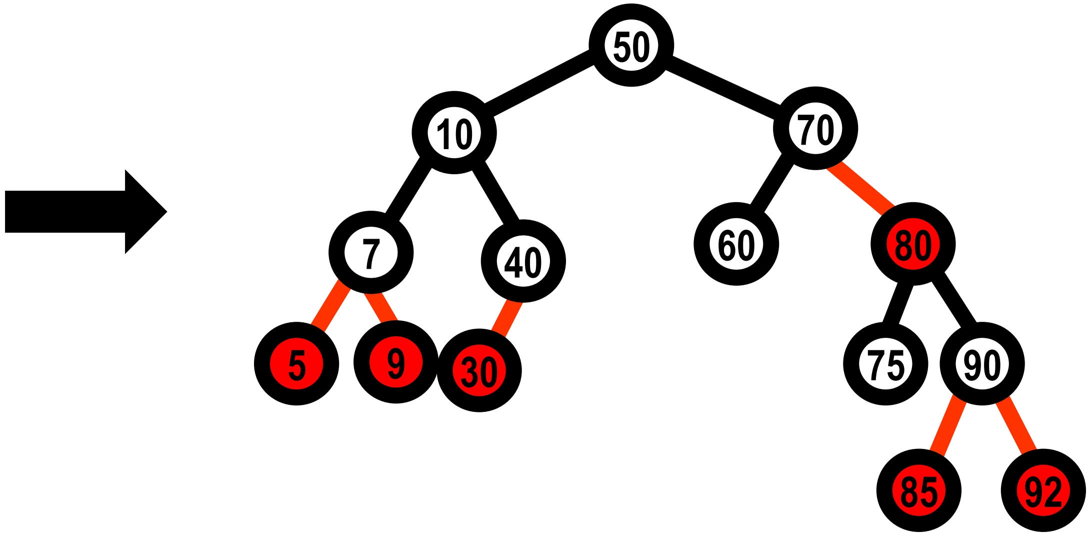


►QUIZ? Red-Black Tree?



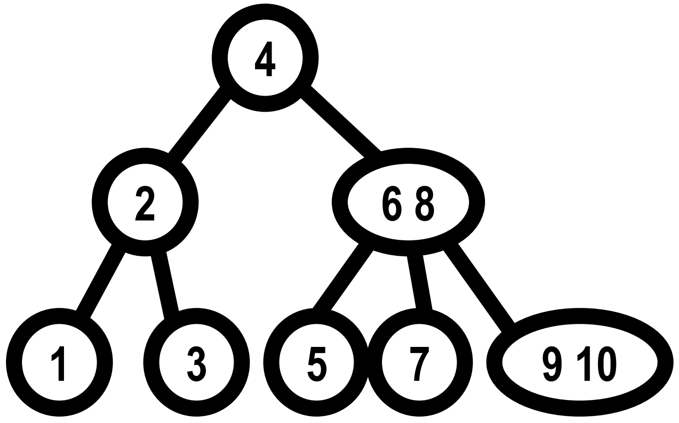
QUIZ: Red-Black Tree?



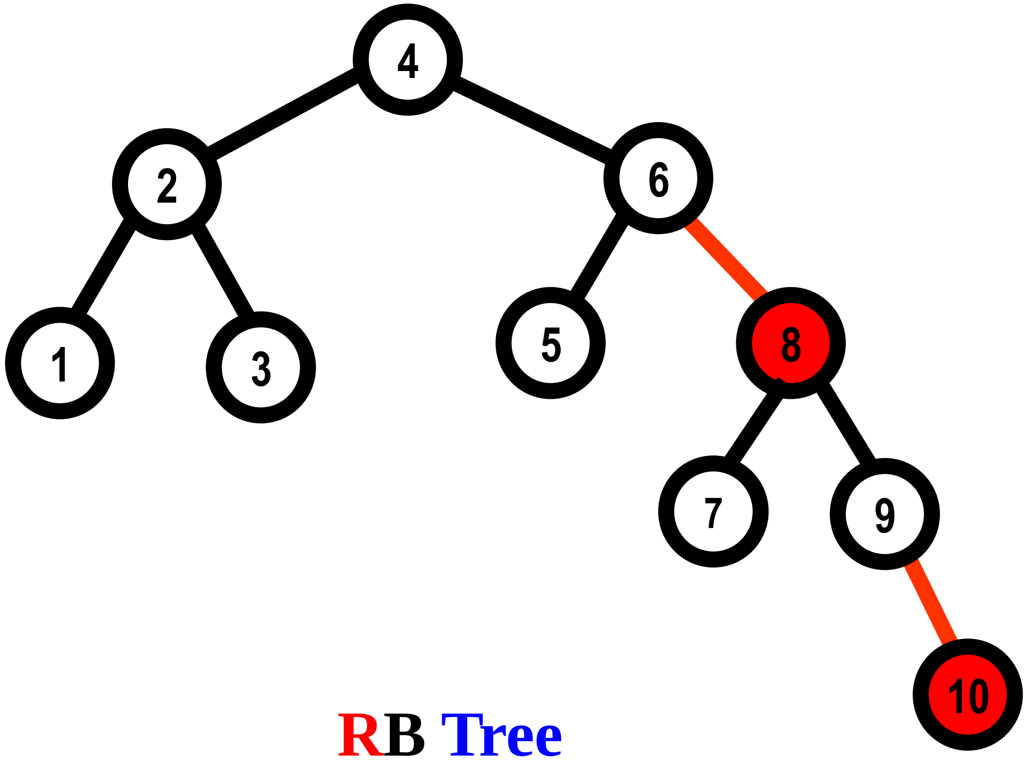


►QUIZ? 2-3-4 Tree vs RB Tree

Insert 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10



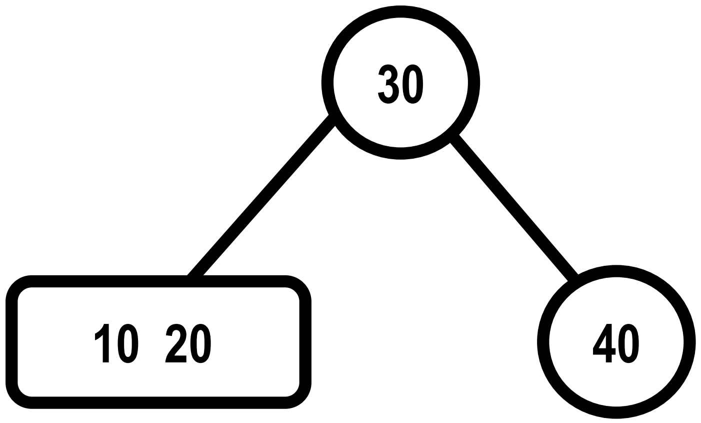
2-3-4 Tree



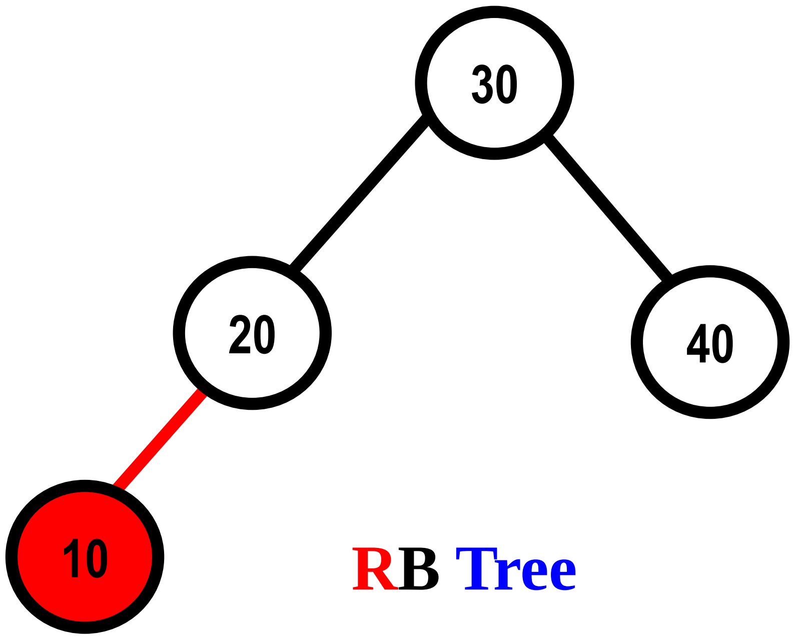
RB Tree

►QUIZ? 2-3-4 Tree vs RB Tree

Insert 20, 30, 40 and 10



2-3-4 Tree



RB Tree

Red Black Tree

- The height of a red-black tree with **n** internal nodes is between **log(n+1)** and **2 log(n+1)**.
- A red-black tree maintains a height close to the minimum **O(log n).**

Searching a Red-Black tree

- Same as with binary search trees
  - ➔ We can search a red-black tree almost as efficiently as a minimum-height binary search tree.
  - ➔ The colors don’t matter!

Red-Black Tree Operations - Analysis

- Insert & Delete
  - ➔**Recoloring**
  - ➔**Rotation**

- Search, Insert & Delete
  - ➔**O(log N)** worst-case

Set & Multiset and Map & Multimap

- Red-Black trees are used in many real-world libraries.
  - ➔C++ STL: map, set, multimap, multiset
  - ➔Java: java.util.TreeMap, java.util.TreeSet

Red-Black Tree vs 2-3-4 Tree vs B-tree of order 4

- A 2-3-4 tree as a binary tree is **isometry** of a red-black tree!
- The red-black tree is then structurally equivalent to a B-tree of order 4!

Red-Black Tree vs AVL Tree

- The AVL trees are
  - ➔more balanced compared to Red-Black trees.
  - ➔more rotations during insertion and deletion.
- If your application uses
  - ➔more frequent insertions and deletions, then Red Black trees!
  - ➔more frequent searches, then AVL trees!

►QUIZ?

- RB Tees vs 2-3-4 Trees?
- RB Trees vs AVL Trees?

Red-Black Tree History

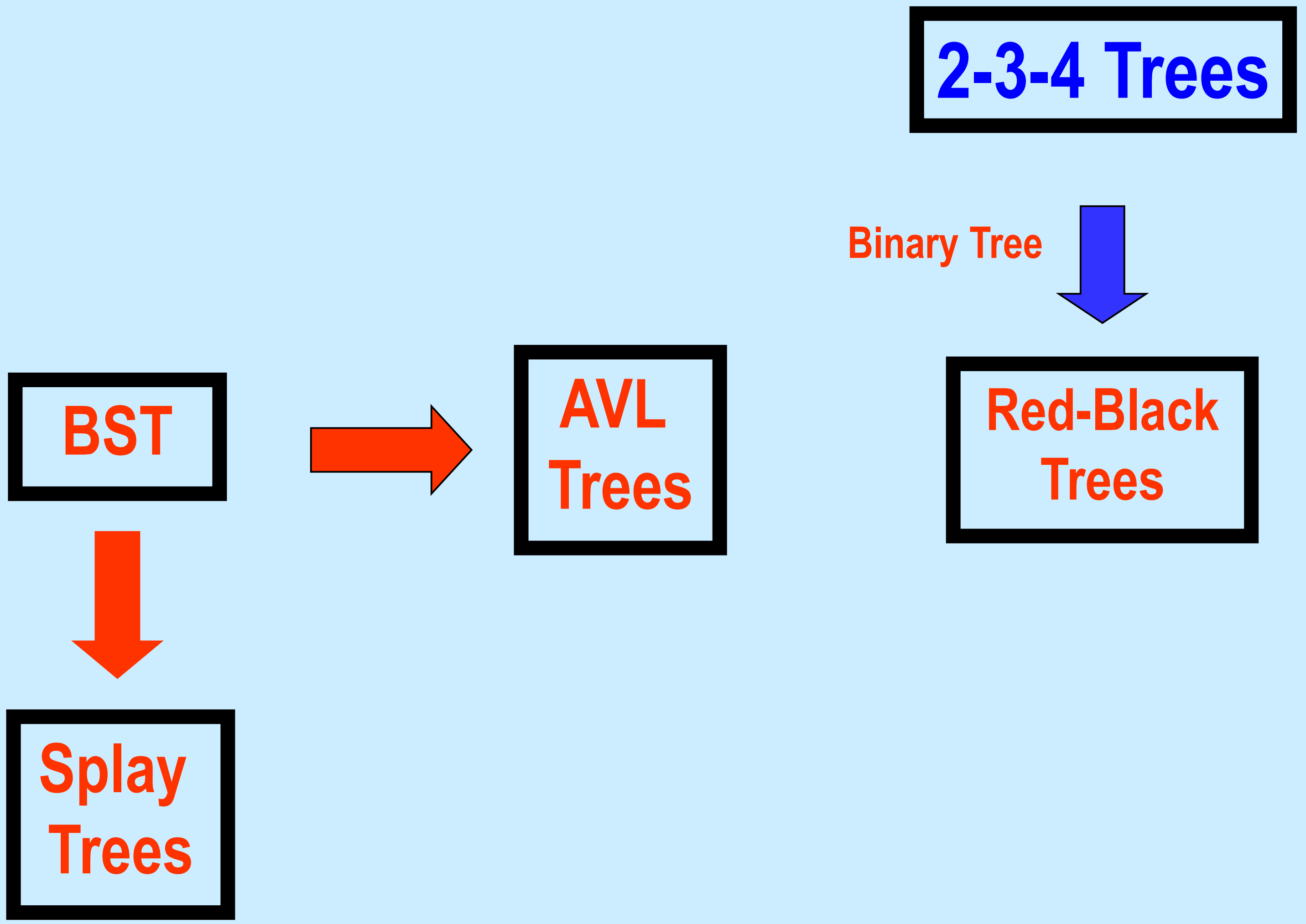
- Leonidas J. Guibas and Robert Sedgewick (1978). "A Dichromatic Framework for Balanced Trees". *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. pp. 8–21.

Red-Black Tree Visualization

- ***Red-Black Tree Visualization***

✓

The World of Balanced Search Trees



B-Trees

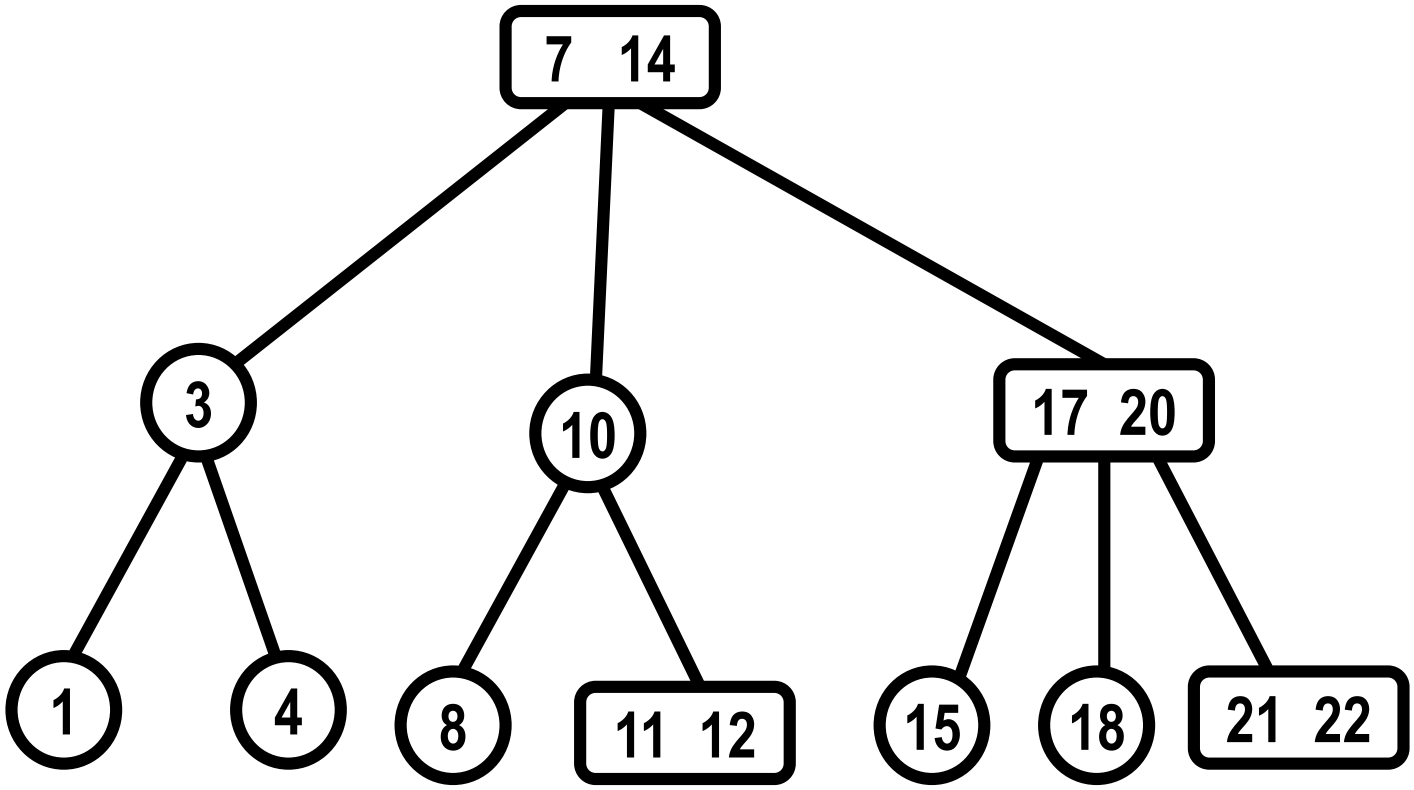
What & Why is a B-Tree?

- A B-tree is a **generalization** of 2-3 or 2-3-4 tree:
  - ➔ A large number of branches
  - ➔ A small height
  - ➔ Suitable for **huge data in disks**, **not in memory!**
  - ➔ Disk access is much slower than memory access.
  - ➔ As dynamic tree-based **Multi-level Indexes**

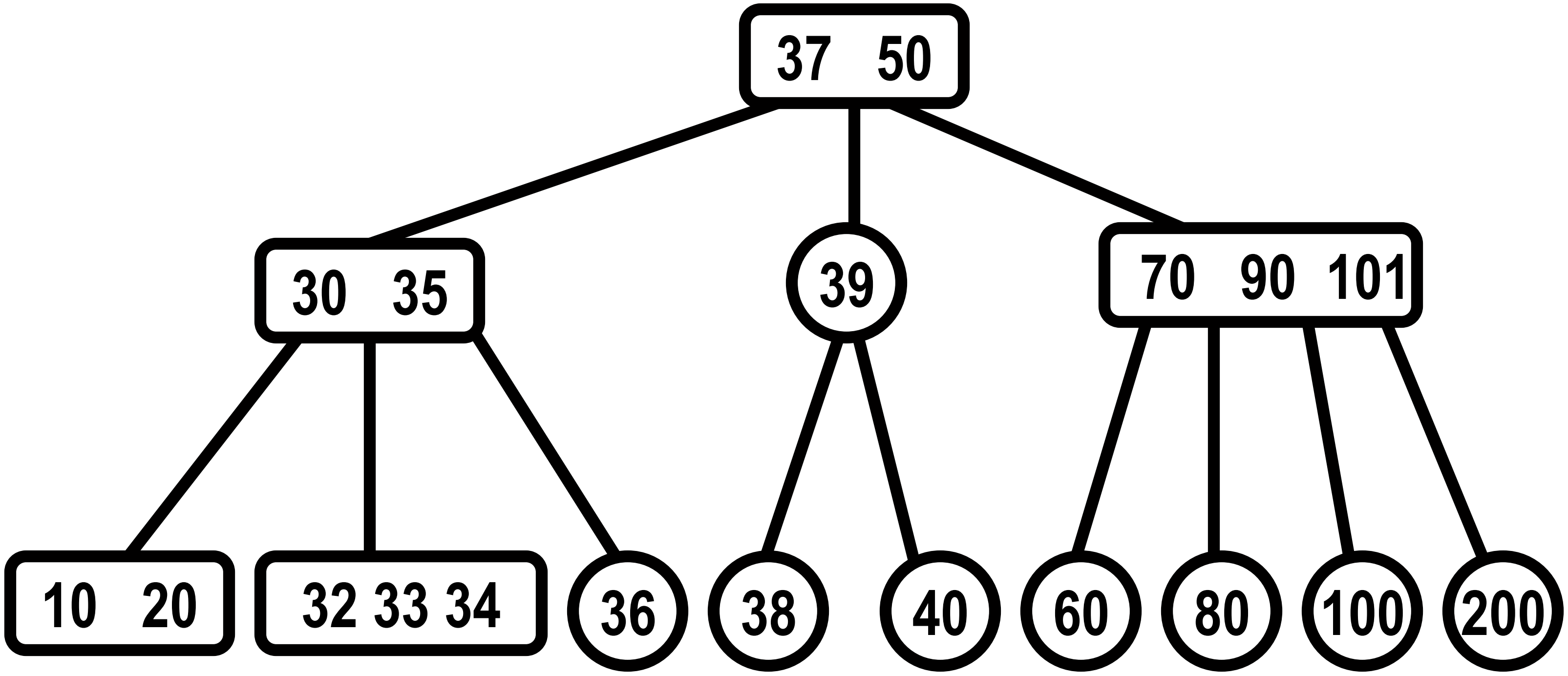
What is a B-Tree of Order m?

- **A B-tree of order (degree) m** is an m-way search tree in which
  - ➔ The root node has at least two subtrees and at most m subtrees.
  - ➔ All nonleaf nodes (other than the root) have between $\lceil m/2 \rceil$ and m children (subtrees) & between ($\lceil m/2 \rceil$ -1) and (m-1) sorted keys.
  - ➔ All leaf nodes store have between ($\lceil m/2 \rceil$-1) and (m-1) sorted keys.
  - ➔ **All leaves are at the same level.**

►QUIZ? A B-Tree of order (degree)?



►QUIZ? A B-Tree of order (degree)?



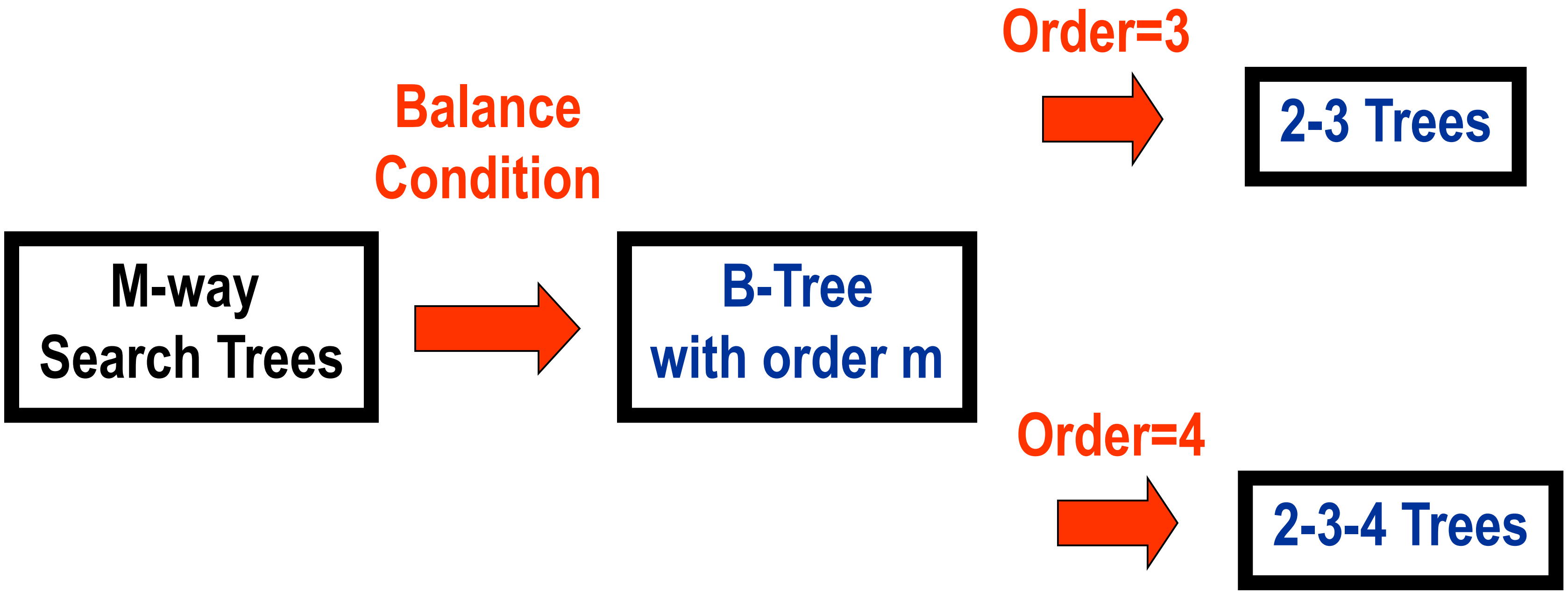
B-Trees vs 2-3 Trees vs 2-3-4 Trees

- What is a B-tree of order (degree) 3?
  - ➔2-3 tree
- What is a B-tree of order (degree) 4?
  - ➔2-3-4 tree

►QUIZ? B-Trees?

- What is a B-tree of order (degree) 5?
  - ➔3-4-5 tree
- What is a B-tree of order (degree) 2?
  - ➔Full binary tree

The World of Balanced Search Trees



B-Tree Operations

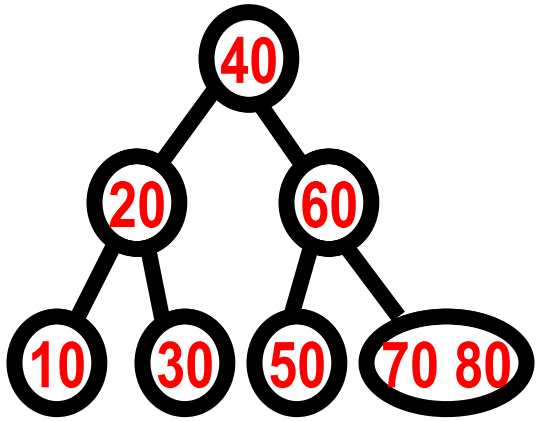
- The B-tree operations are a generalization of the 2-3 tree operations!

  ➔**Splitting/Promote** for insertion

  ➔**Redistribute/Merging** for deletion

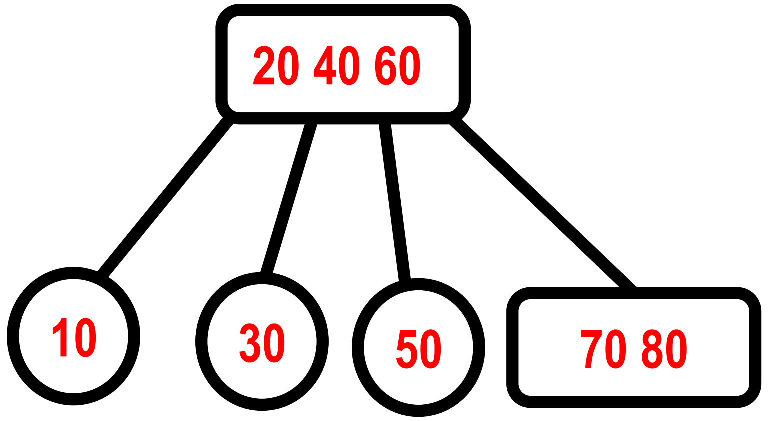
►QUIZ? B-Tree of degree 3?

Insert 10, 20, 30, 40, 50, 60, 70 and 80



►QUIZ? B-Tree of degree 4?

Insert 10, 20, 30, 40, 50, 60,70 and 80



Property of B-Trees

- What is the **minimum** number N_h of keys in a B-tree of order m and height h?
  - ➔ $N_h = 2\lceil m/2 \rceil^{h-1} - 1$

Properties of B-Trees of degree m

The **height** of a B-tree of degree m with **n** nodes is

Θ (log N).

B-Tree Operations - Analysis

- The worst-case running time of search, insertion and deletion of records from B-trees is
  - ➔**Θ (log N)**
- The worst-case running time of traversal of B-trees is
  - ➔**Θ (N)**

Variations of B-Trees

- **B+-trees**
- **B*-trees**
- …

- B-trees/B+ trees/B* trees are data structures used to implement dynamic tree-based multilevel indexes in databases and file systems!

B-Tree Visualization

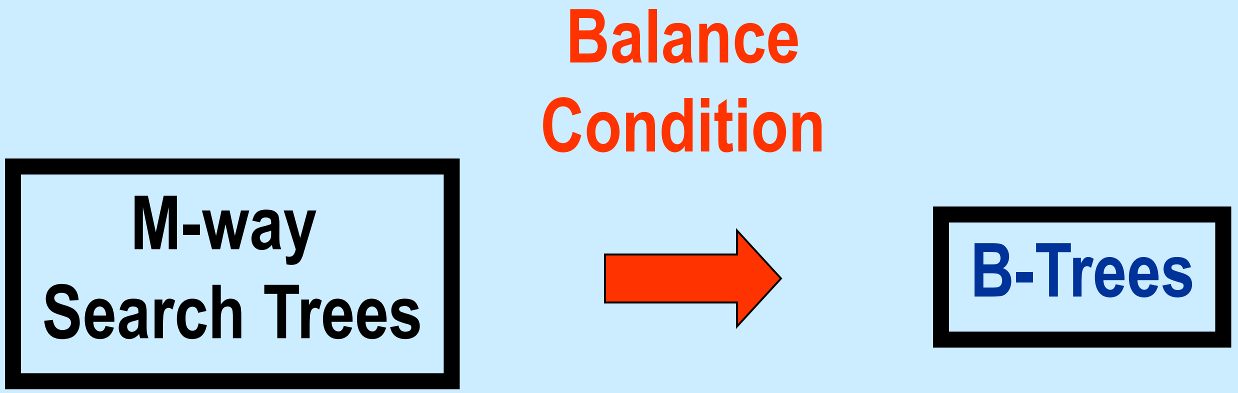
- ***B-Tree Visualization***

Self-Adjusting M-way Search Trees

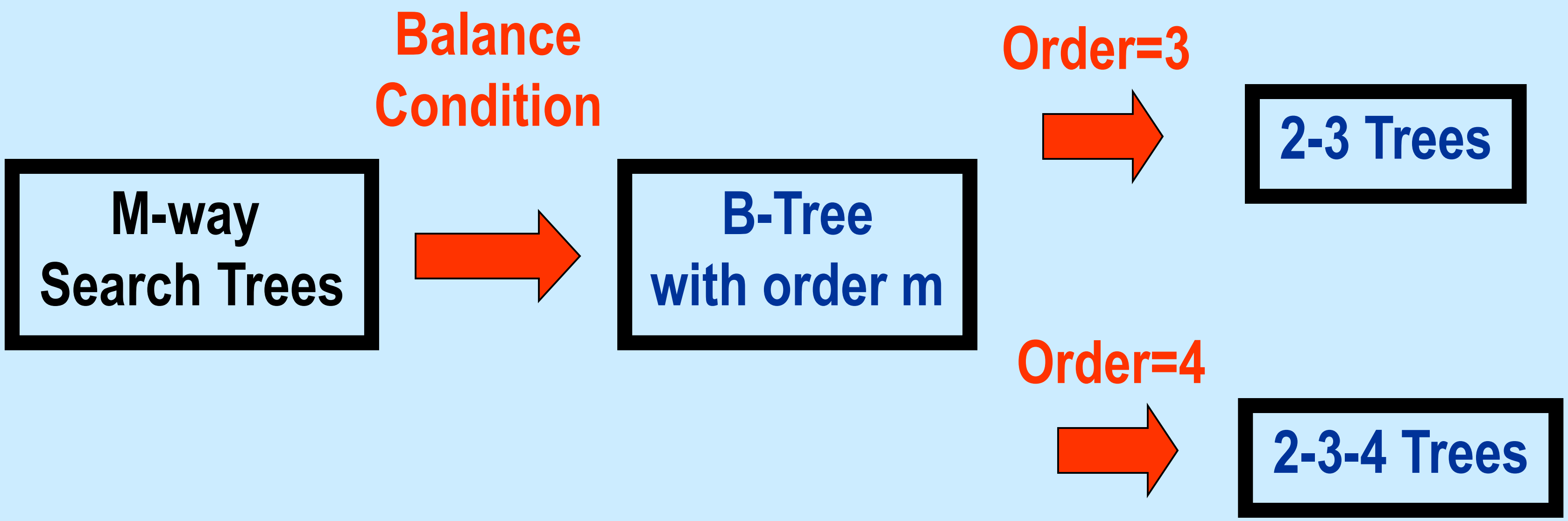
Self-Adjusting M-way Search Trees

- Any idea? You may propose a new DS!

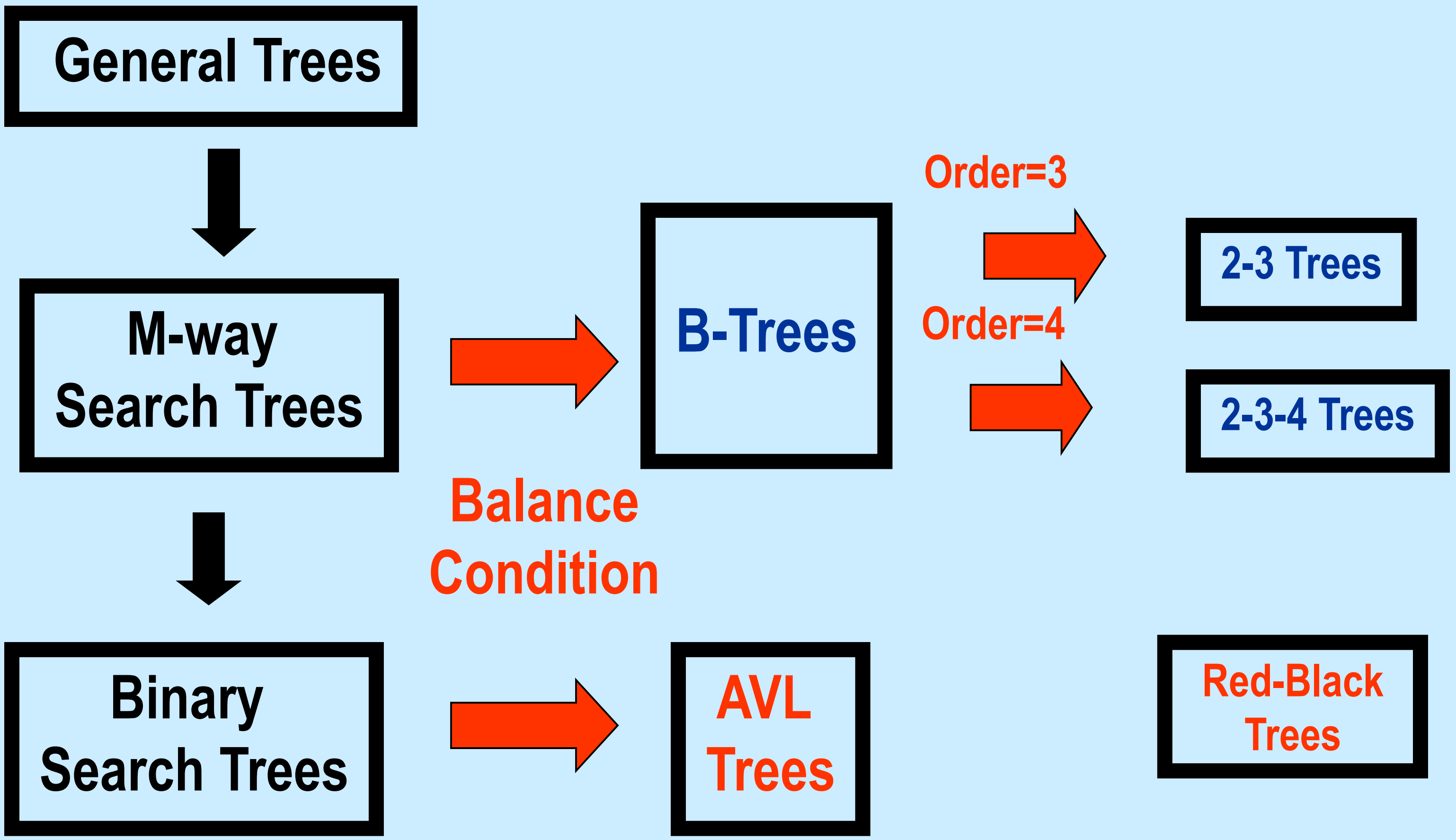
The World of Balanced Search Trees



The World of Balanced Search Trees



The World of Balanced Search Trees



Homework Assignment

►Homework Assignment?

- Draw the 2-3 tree that results when you insert items with the keys 5, 21, 8, 63, 69, 32, 7, 19 and 25 in that order into an initially empty 2-3 tree.

►Homework Assignment?

- Draw the 2-3-4 tree that results when you insert (one-pass insertion using preemptive splitting) items with the keys 1, 12, 8, 2, 25, 6, 14, 28, 17, 7 and 52 in that order into an initially empty 2-3-4 tree.

►Homework Assignment?

- Draw the 2-3-4 tree that results when you insert (two-pass insertion similar to the 2-3 tree insertion) items with the keys 1, 12, 8, 2, 25, 6, 14, 28, 17, 7 and 52 in that order into an initially empty 2-3-4 tree.

the right majors, minors & conc

ucation?

for students' academic and career success

ging for many students - *Many students change their*
during college!

e prediction of student success in MMC could
dual students

d their right MMC

chieve their academic goals

Y. PARK • DEPT OF ESBIS, BRADLEY UNIVERSITY 1/2

END

Prof. Young Park