Trees
Advanced Trees

Dr. Nguyen Hua Phung

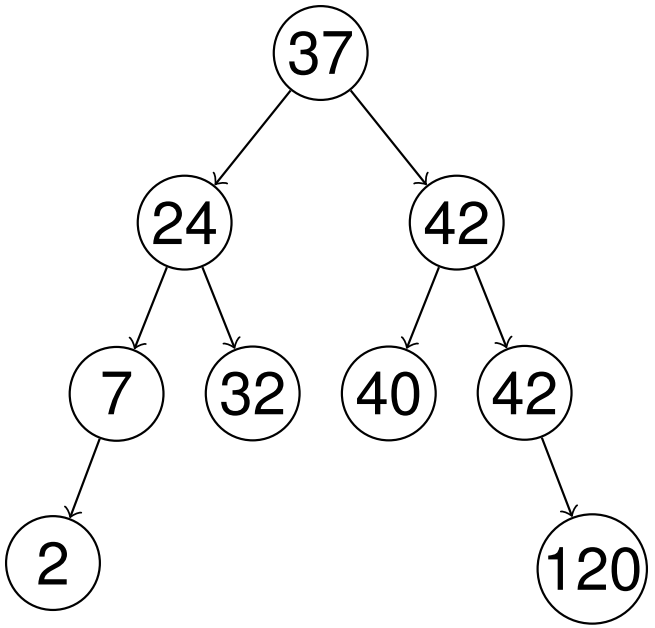
HCMC University of Technology, Viet Nam

01, 2020

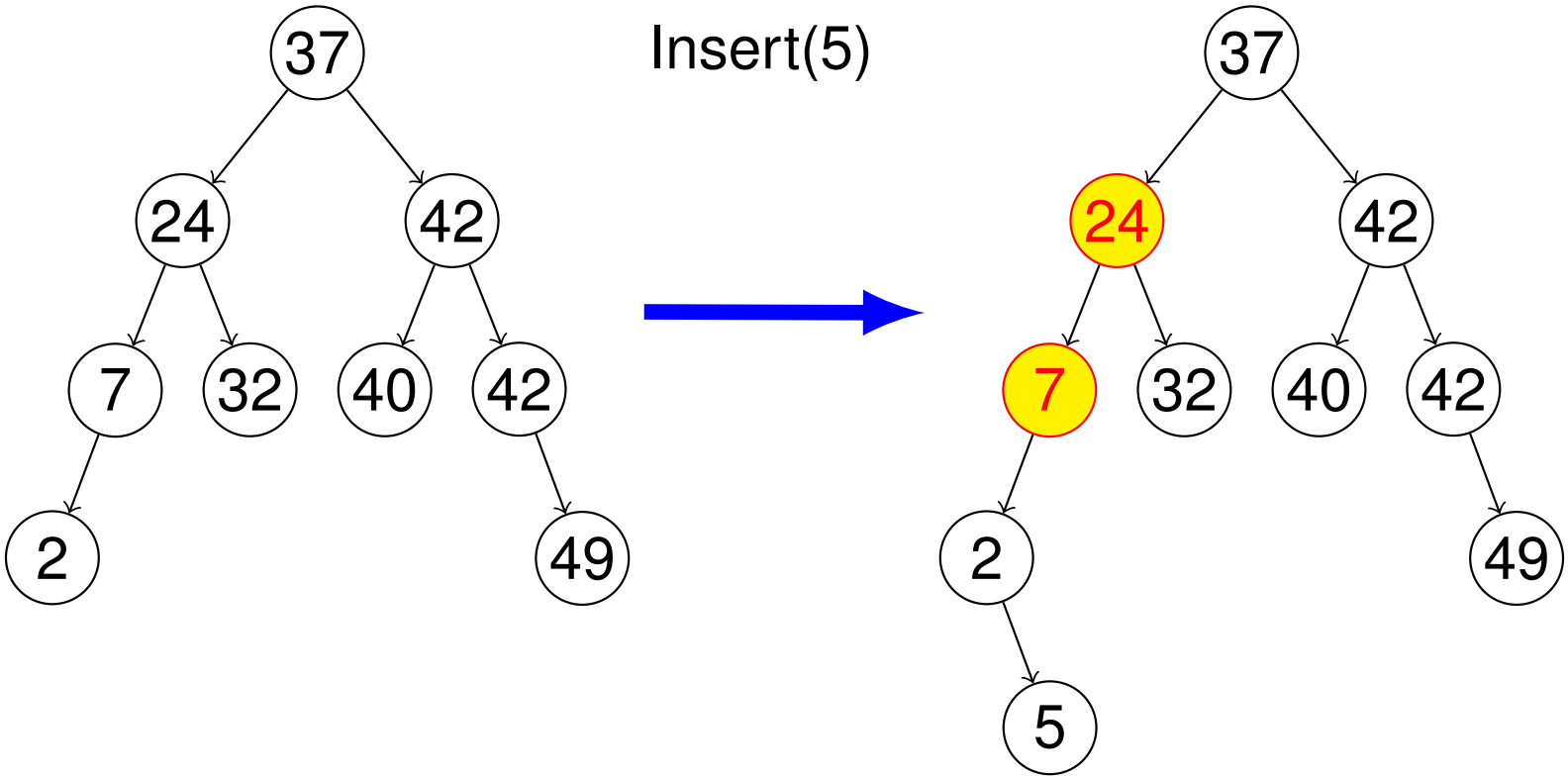
Outline

AVL Trees

- if BST is balanced, searching costs O(log n)
- AVL (Adelson-Velskii and Landis)
  - BST
  - balanced: for every node, $|H_L - H_R| \leq 1$

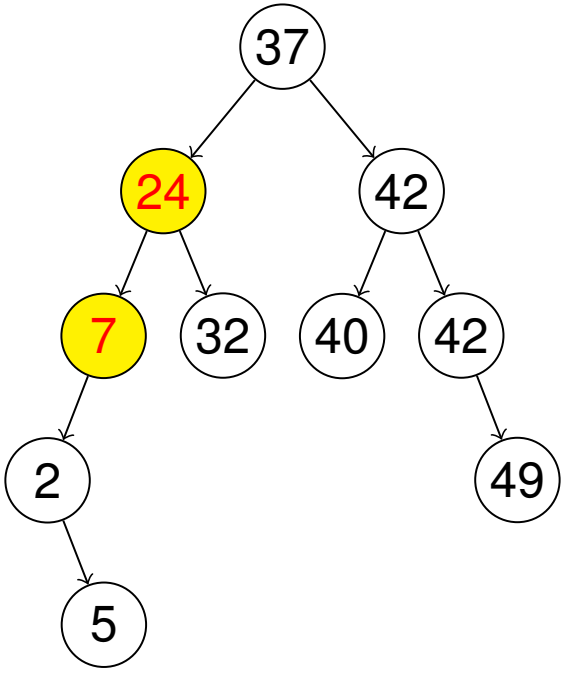


AVL Tree Insertion



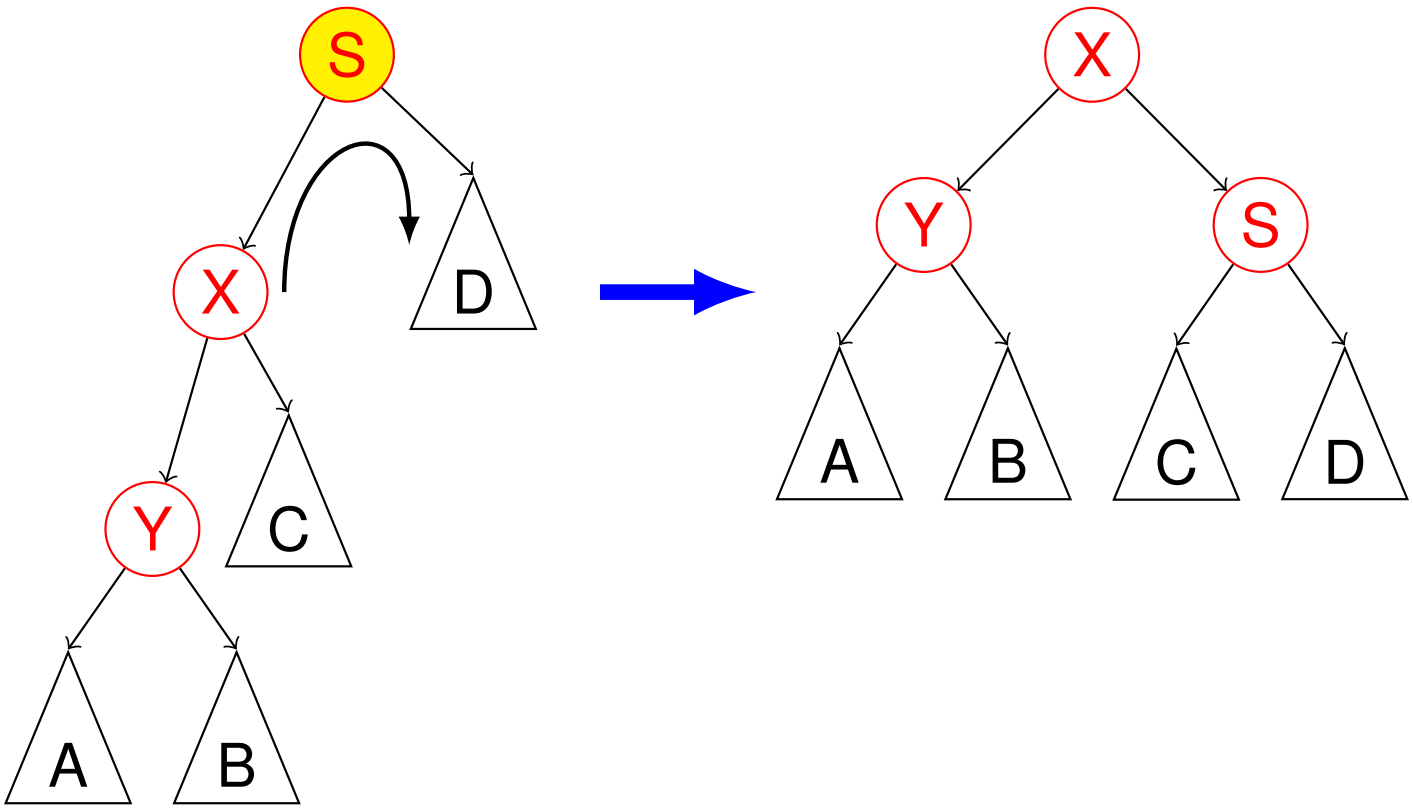
Re-balance AVL Trees

- For the bottommost unbalanced node S, the extra node is in:
  - the left child of the left child of S, or
  - the right child of the right child of S, or
  - the left child of the right child of S, or
  - the right child of the left child of S



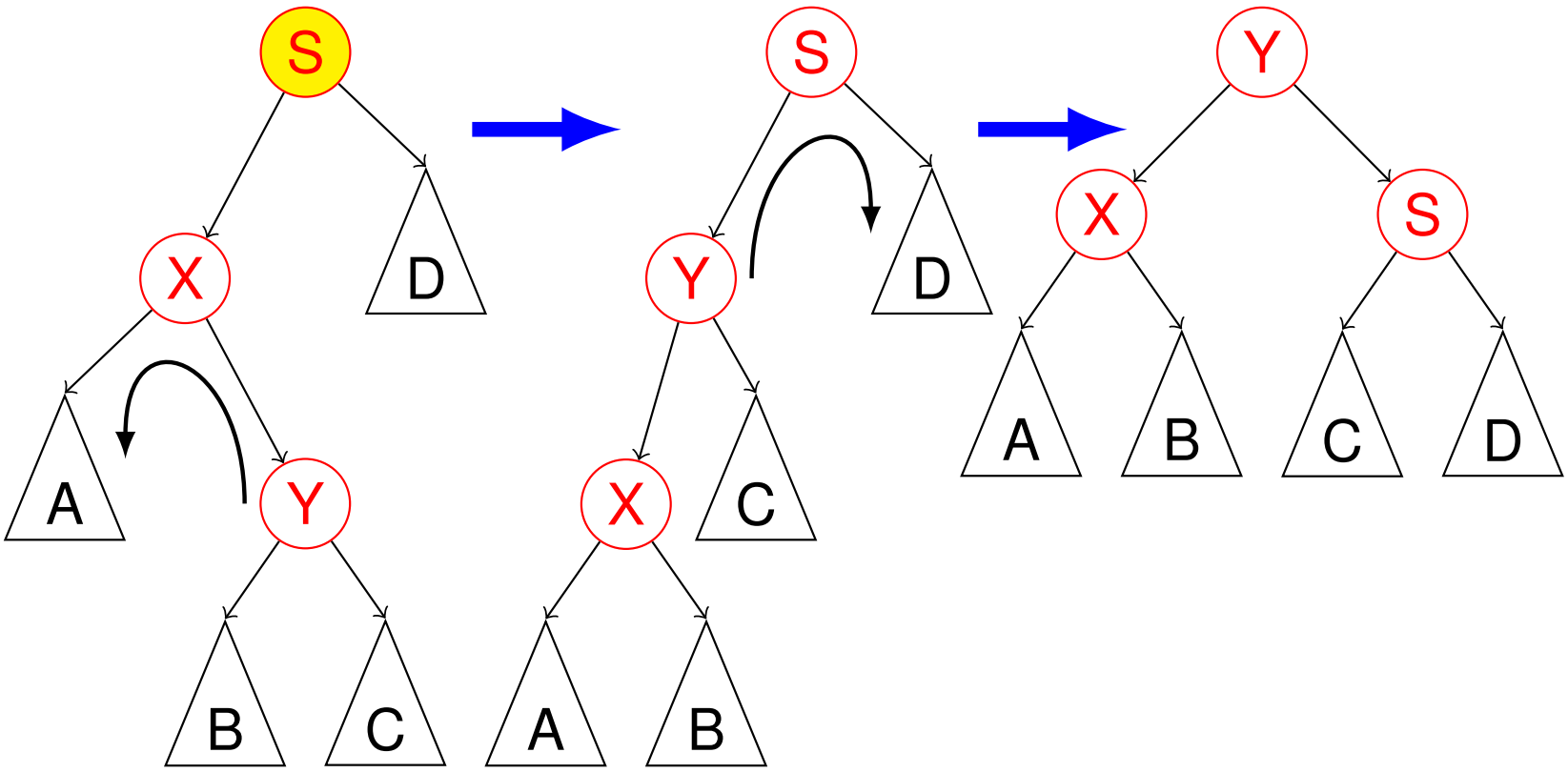
Single Rotation

- Apply for cases 1 and 2



Double Rotation

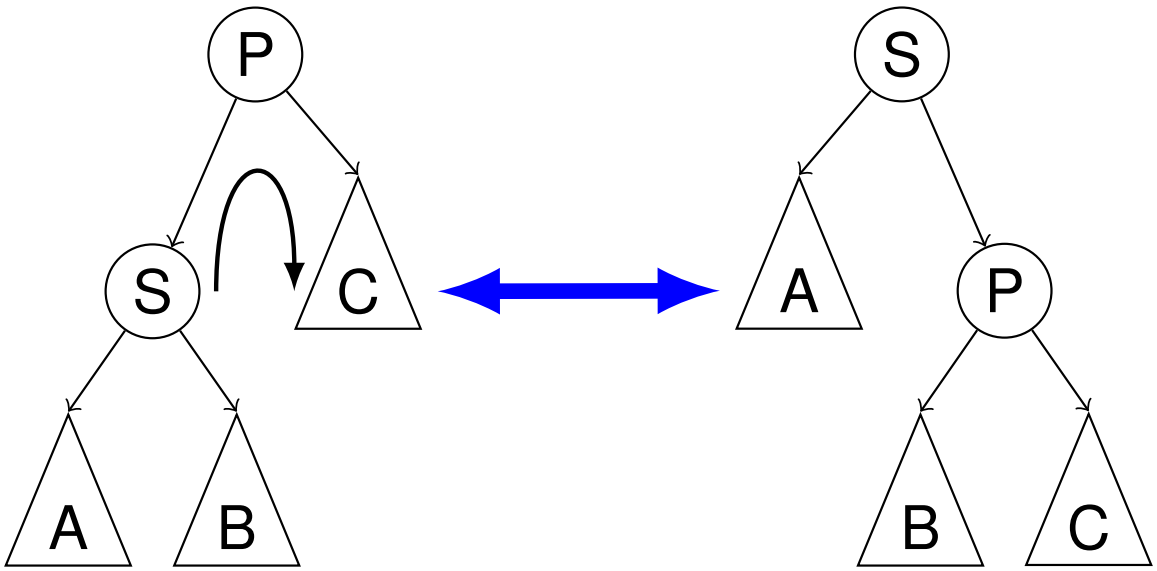
- Apply for cases 3 and 4



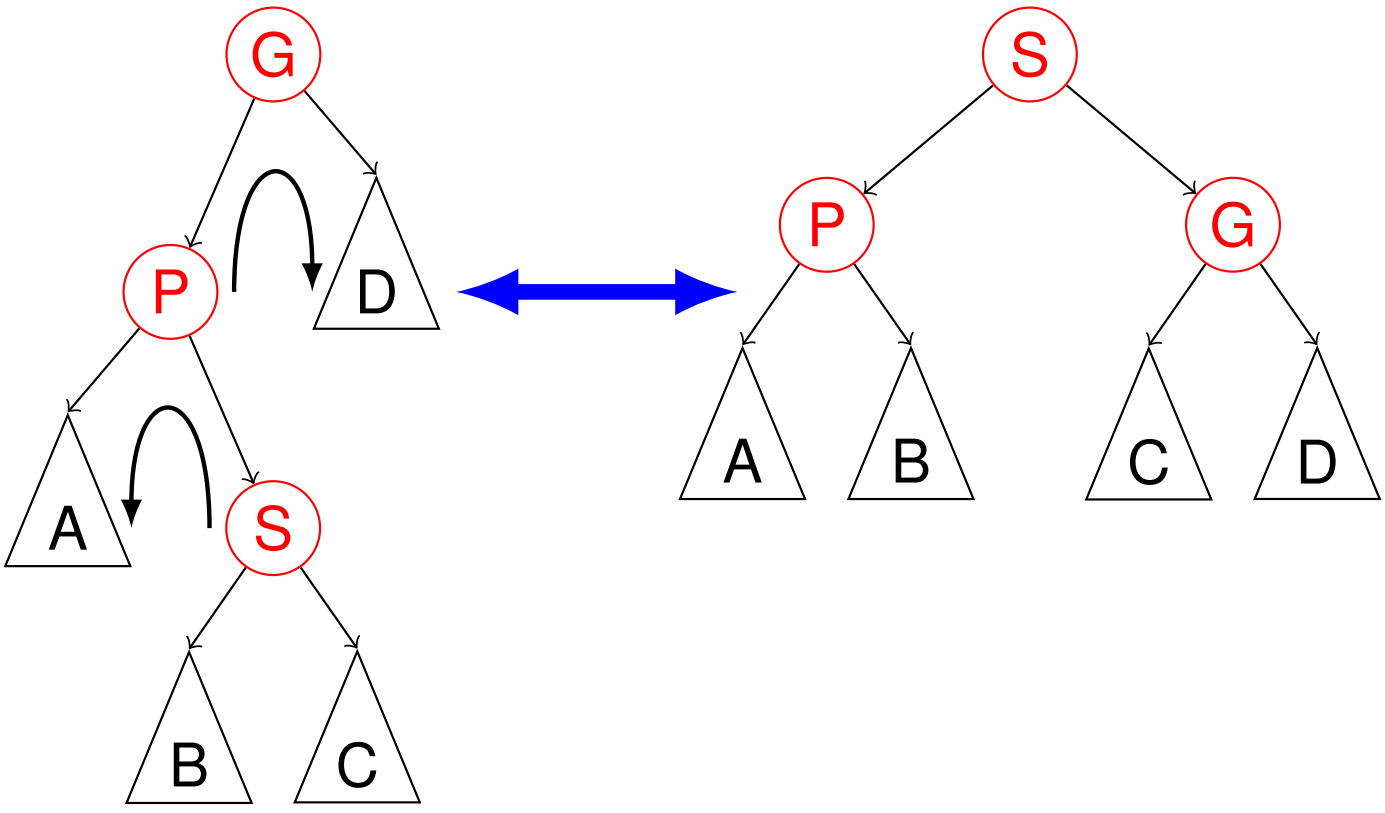
Splay Trees

- BST
- insert, delete, and search are modified to reach
  - O(mlog n) where m is number of operations and m>n
  - simpler than AVL as not guarantee balanced tree
- based on the 80/20 rule: "80% access to 20% records"
- move newly accessed nodes to the root

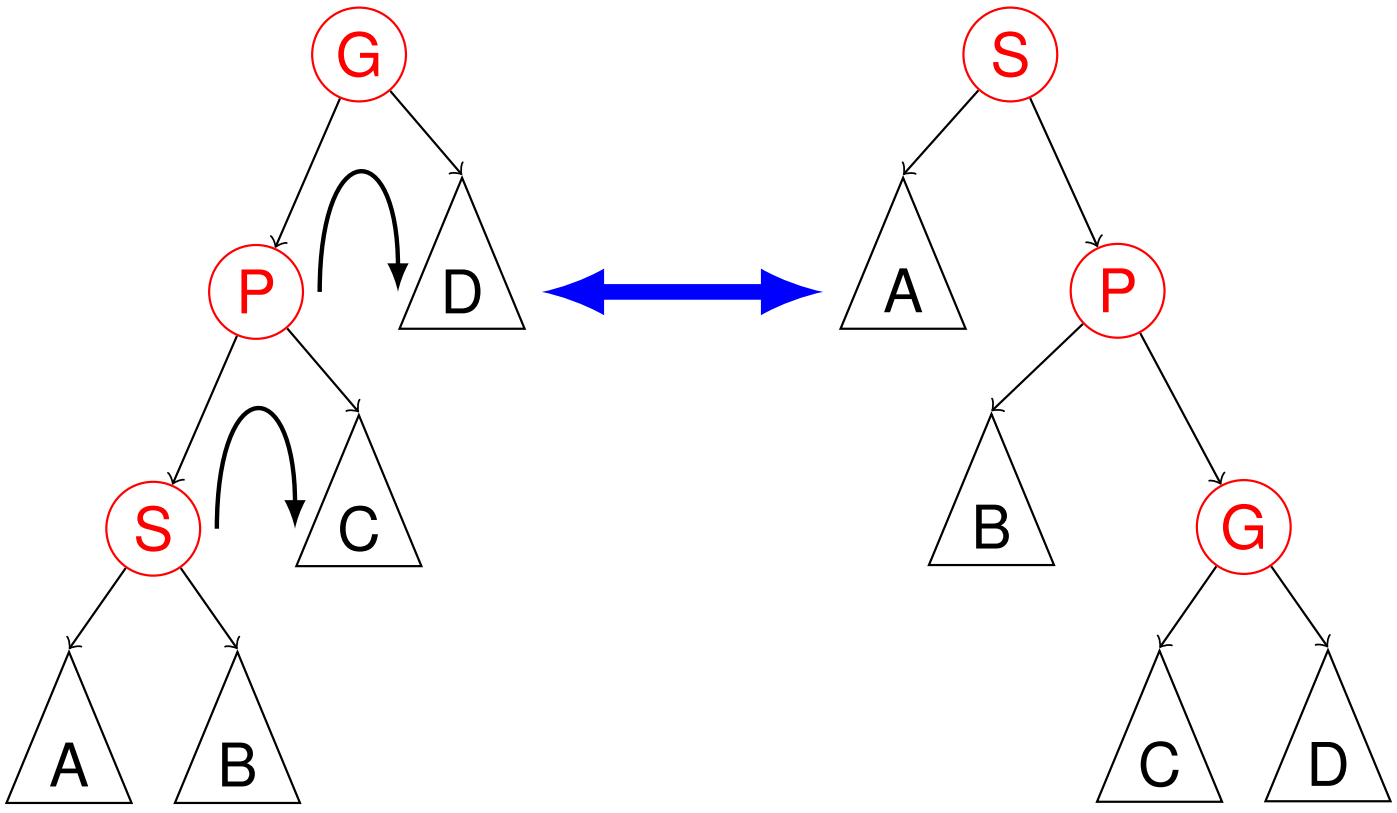
Splay Tree Single Rotation



Splay Tree Zigzag Rotation



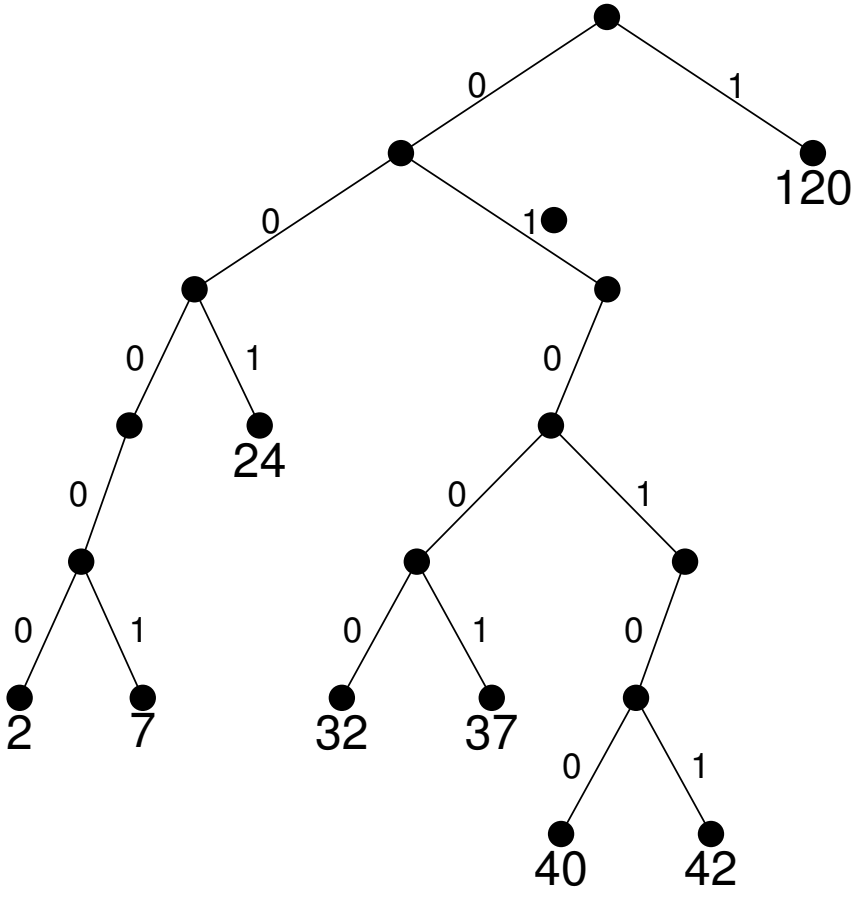
Splay Tree Zigzig Rotation



Tries

- Shape of BST depends on the order of data insertion
- Tries: predefine the distribution of value based on the range of values
  - value in the upper half of the range => right subtree
  - value in the lower half of the range => left subtree
  - value is kept just in the leaf
- Properties of Tries:
  - Shape of tries does not depend on the order of data insertion
  - The height of tries is limited by binary logarithm of the range

Tries Example

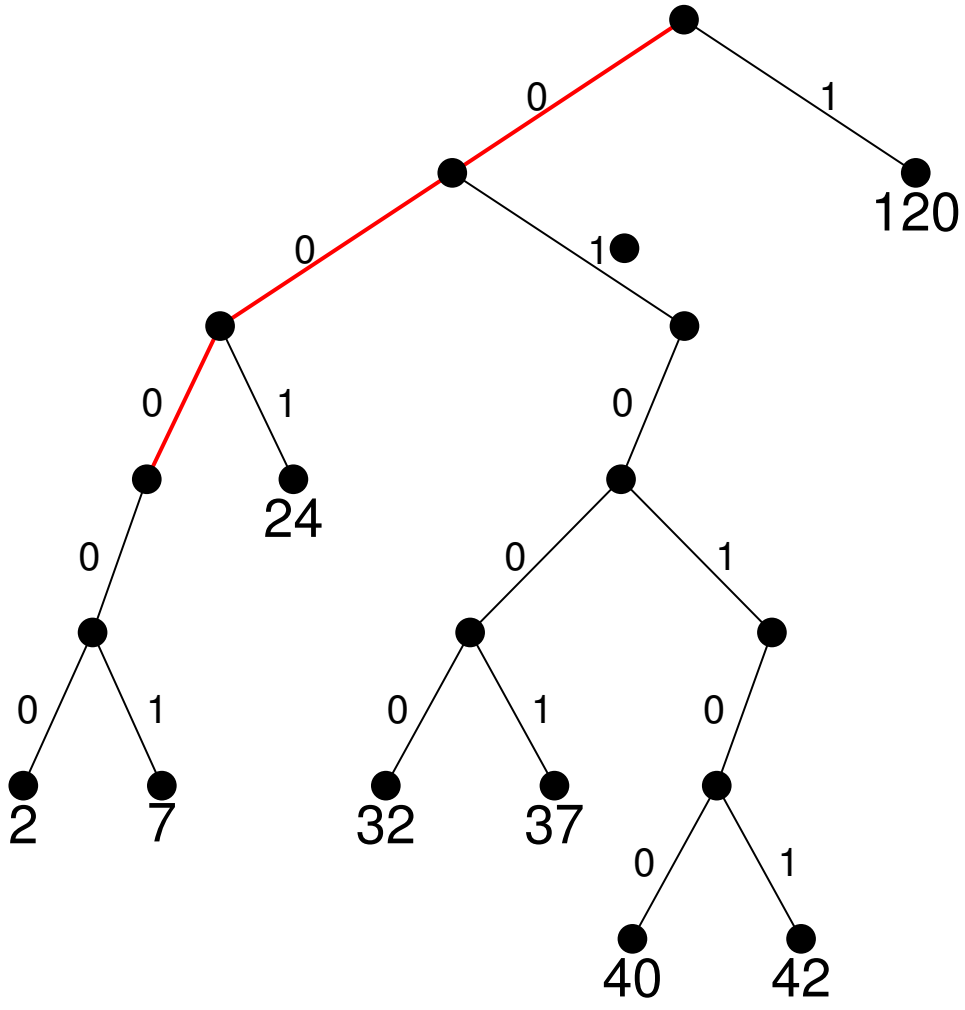


Range : 0-127 => 7 bits

Values : {2,7,24,32, 37,40,42,120}

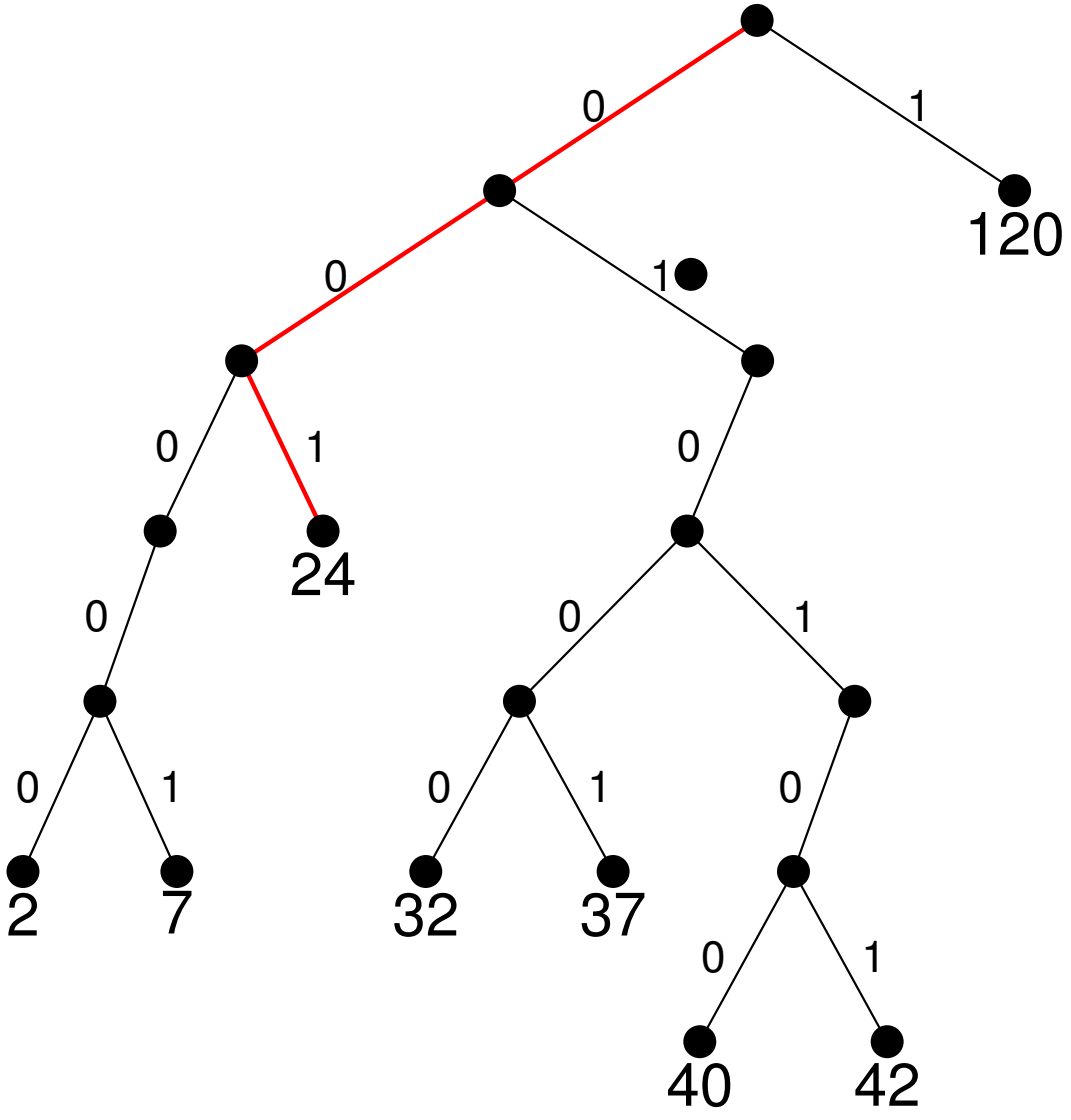
120	1~~1110000~~
42	0101010
40	0101000
37	0100101
32	0100000
24	0011000
7	0000111
2	0000010

Search in Tries



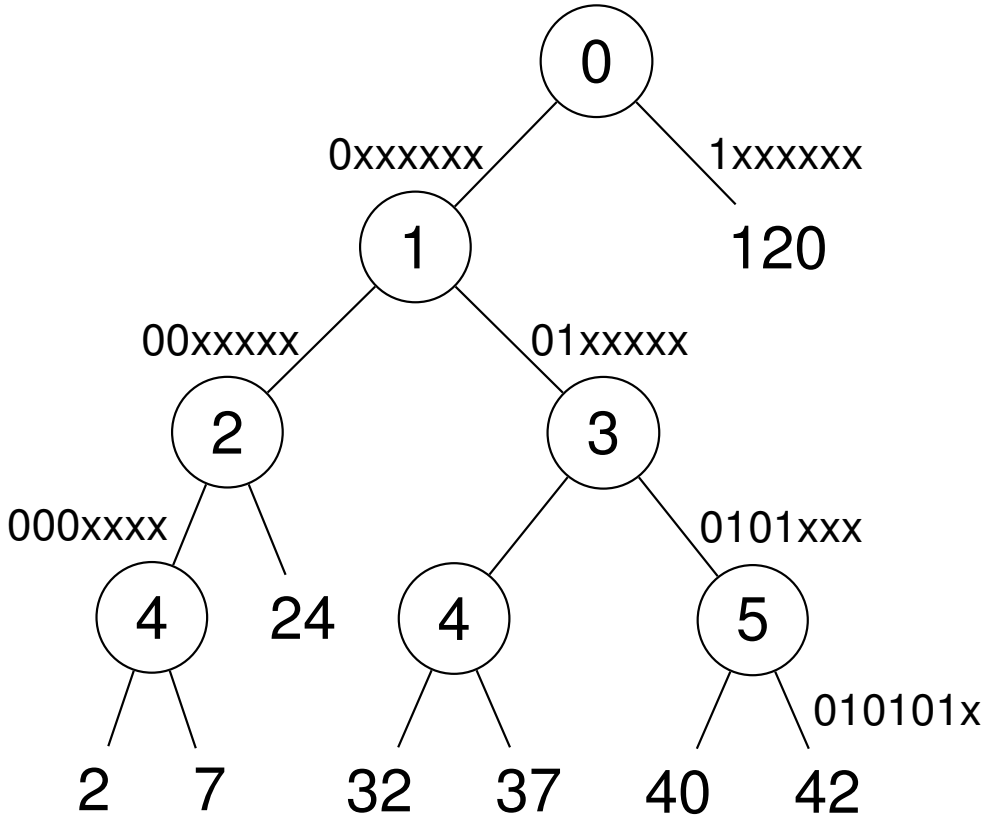
Search(15)
=> Search(0001111)
=> 15 not found

Search in Tries

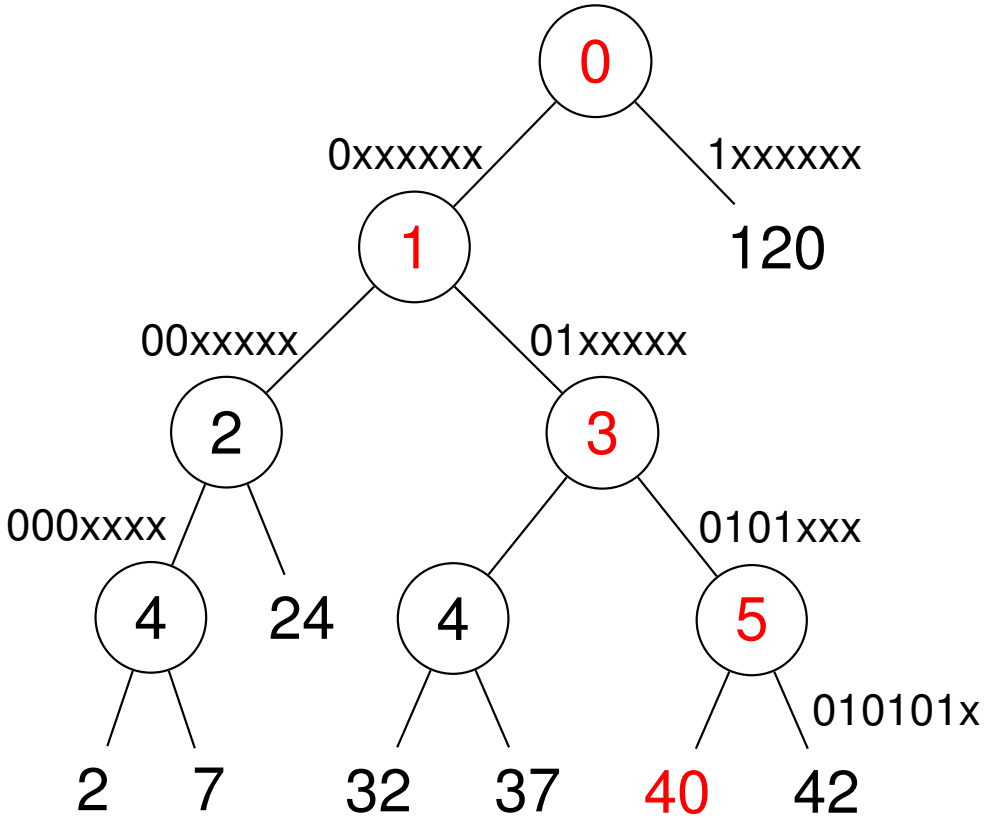


Search(24)
=> Search(0011000)
=> 24 found

PAT Tries

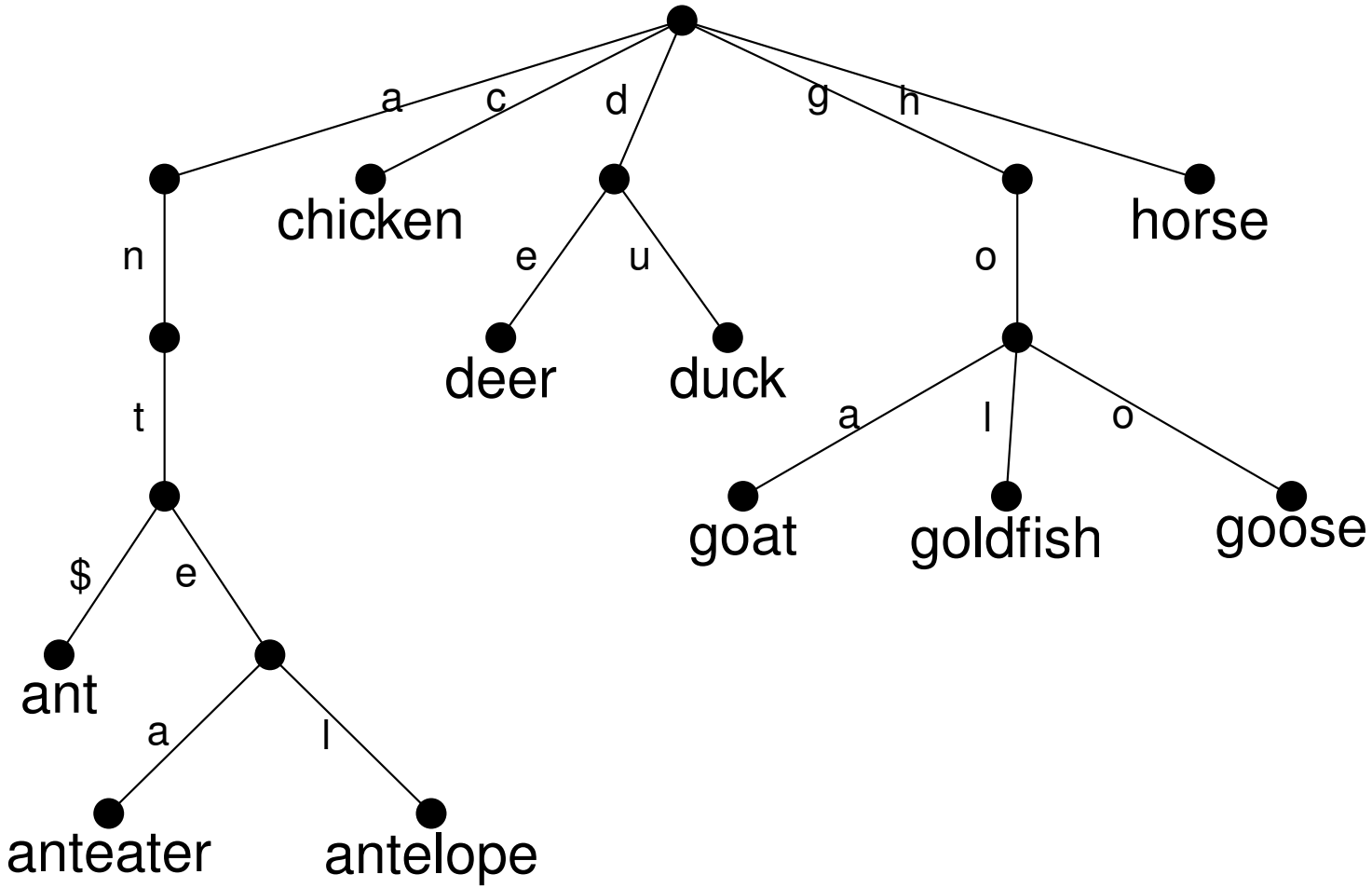

0
0xxxxxx
1xxxxxx
1
120
00xxxxx
01xxxxx
2
3
000xxxx
0101xxx
4
24
4
5
010101x
2
7
32
37
40
42

Search on PAT Tries



Search(41)
=> Search(0101001)
=> 41 not found

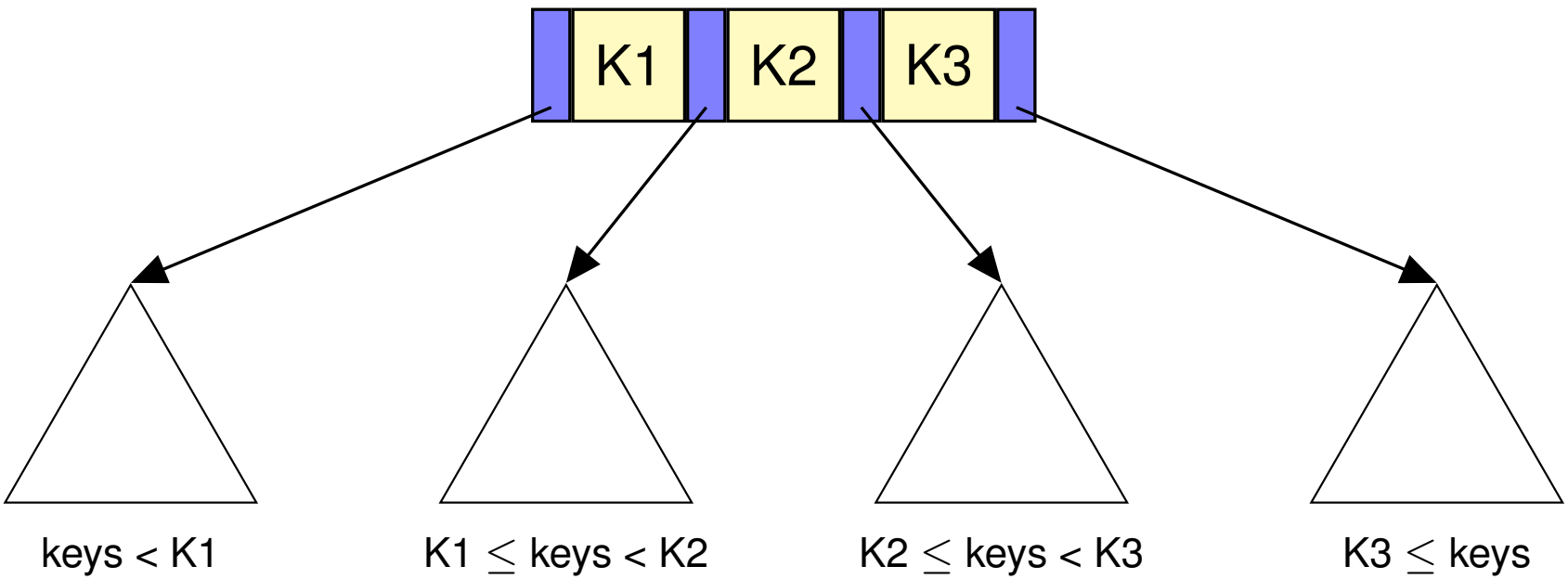
Alphabet Tries



Multiway Search Trees

- Tree whose outdegree is not restricted to 2 while retaining the general properties of binary search trees.
  - Each node has m - 1 data entries and m subtree pointers.
  - The key values in a subtree
    - $\geq$ the key of the left data entry
    - $<$ the key of the right data entry

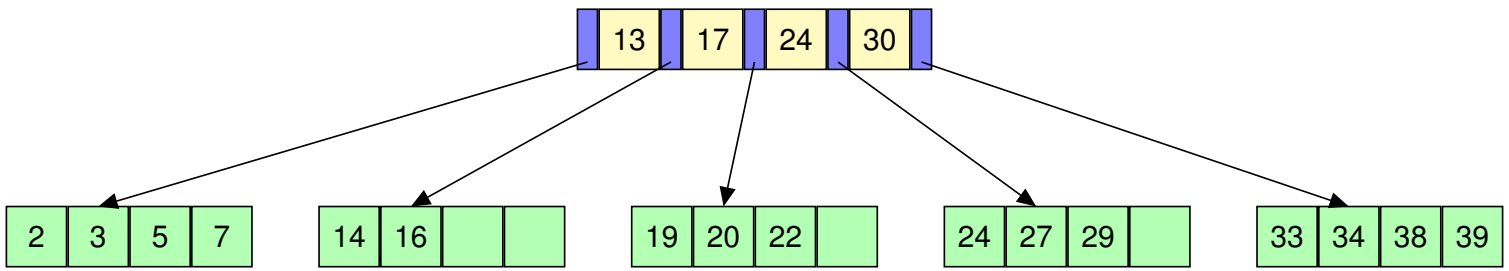
M-way Search Trees



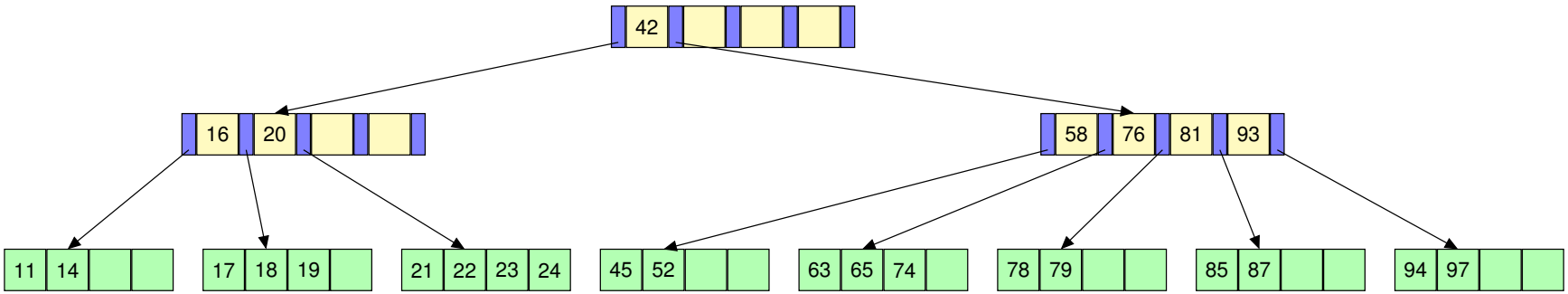
B-Tree

- A B-tree is an m-way tree with the following additional properties:
  - The root is either a leaf or has at least 2 and at most m subtrees.
  - All internal nodes have at least [m/2] and at most m subtrees.
  - A leaf node has at least [m/2] - 1 and at most m - 1 entries.
  - All leaf nodes are at the same level.

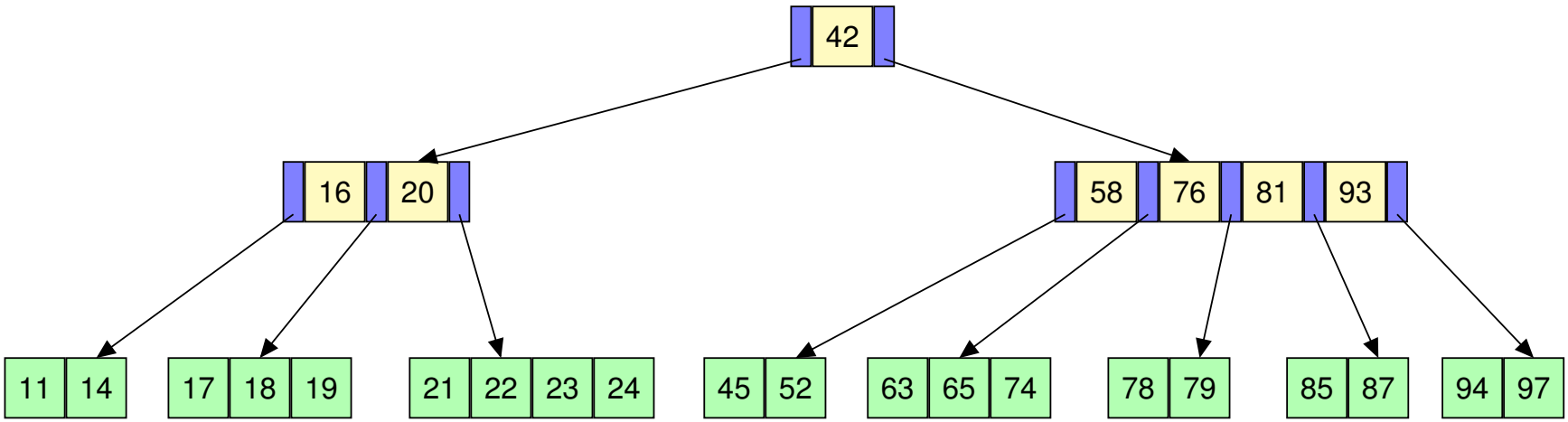
B-Tree (m=5)



B-Tree


42
16
20
58
76
81
93
11
14
17
18
19
21
22
23
24
45
52
63
65
74
78
79
85
87
94
97

B-Tree (m=5)

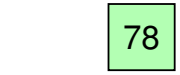

42
16
20
58
76
81
93
11
14
17
18
19
21
22
23
24
45
52
63
65
74
78
79
85
87
94
97

B-Tree Insertion

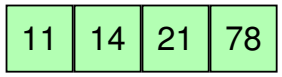
- Insert the new entry into a leaf node.
- If the leaf node is overflow, then split it and insert its median entry into its parent.
- If the internal node is overflow, do the same thing
- If the root is overflow, then create a new root containing the median entry.

B-Tree Insertion

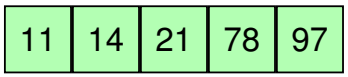
Insert 78

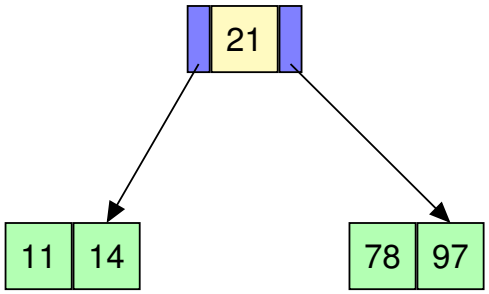


Insert 21,14,11



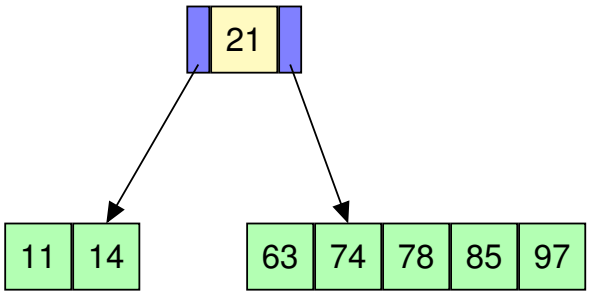
Insert 97

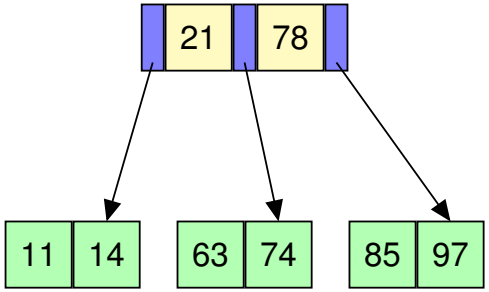




overflow

Insert 85,74,63

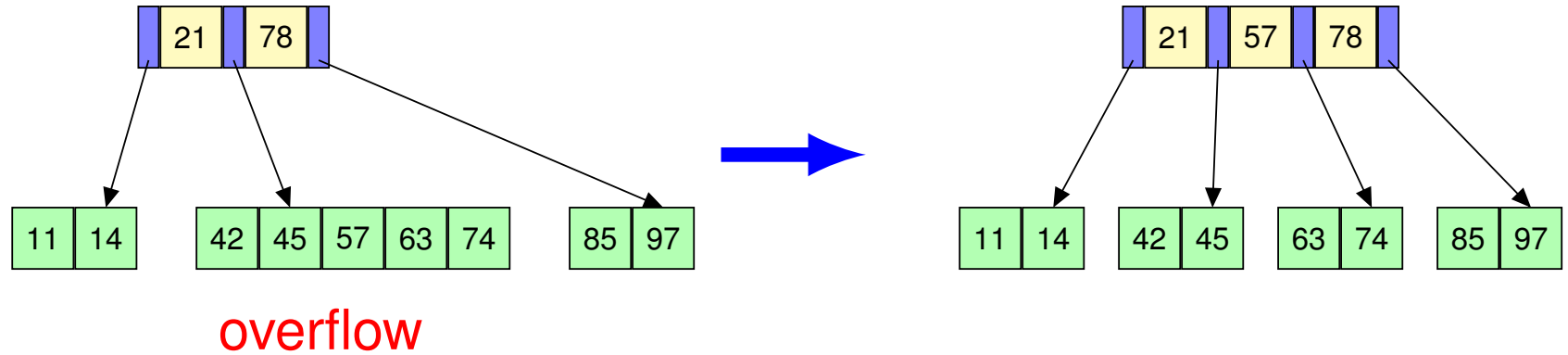




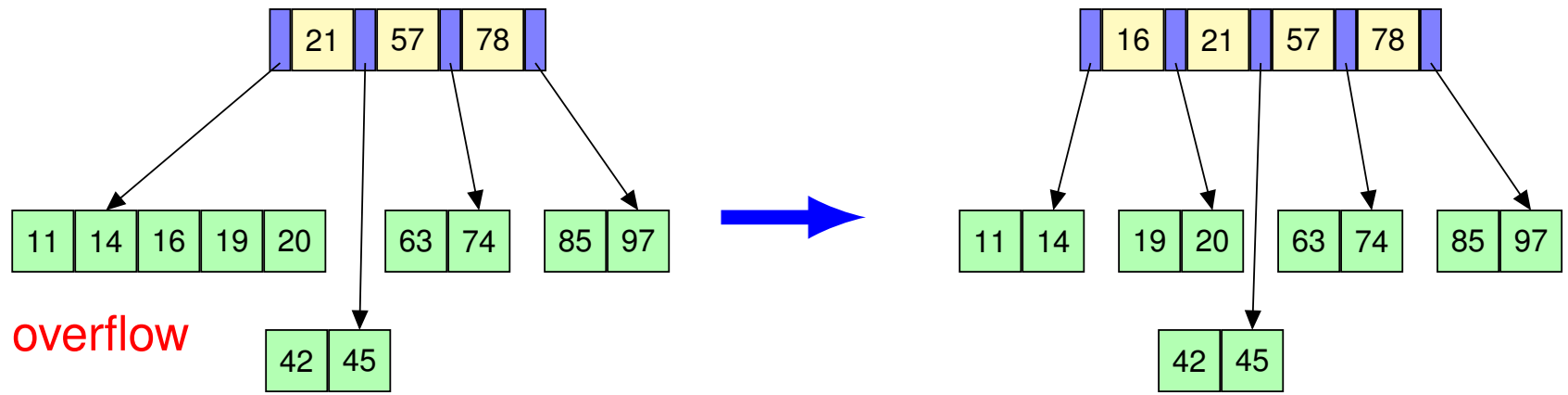
overflow

B-Tree Insertion

Insert 45,42,57

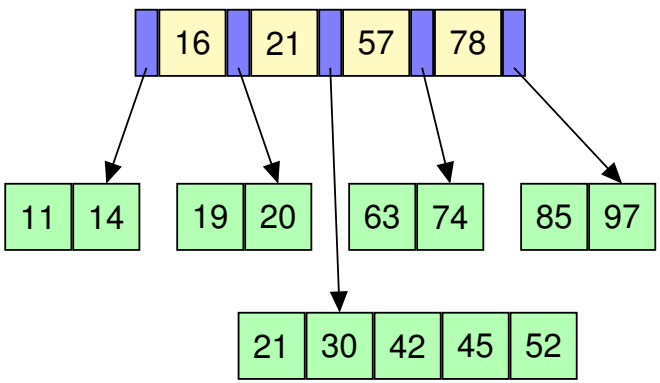


Insert 20,16,19

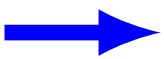


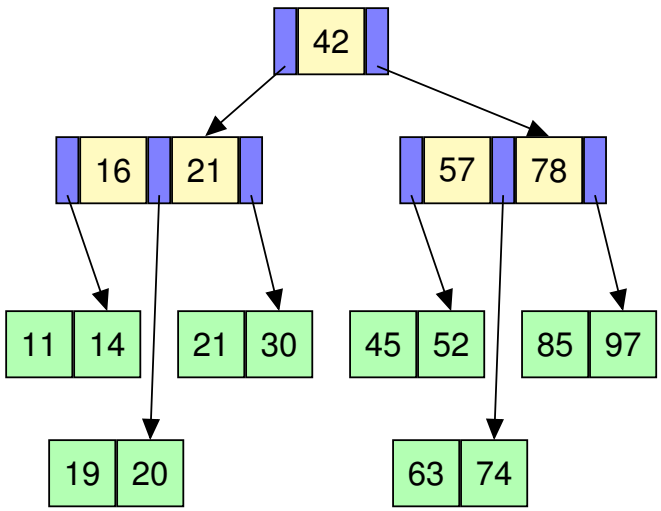
B-Tree Insertion

Insert 52,30,21



overflow



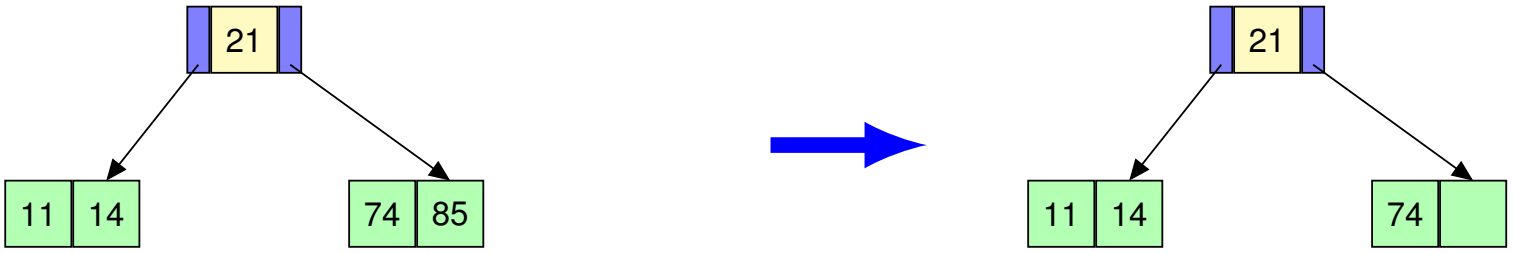


B-Tree Deletion

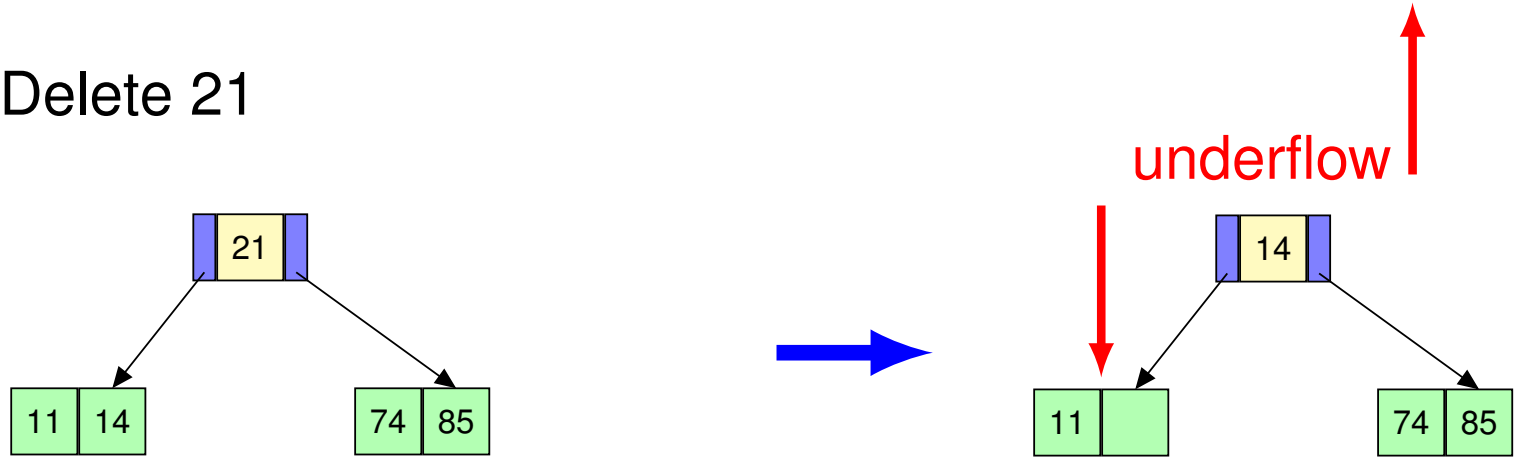
- It must take place at a leaf node
- If the data to be deleted is not in a leaf node, then replace that entry by the largest entry on its left subtree.

B-Tree Deletion

Delete 85



Delete 21



Reflow

- To guarantee each node have sufficient number of entries:
  - Balance: shift data among nodes.
  - Combine: join data from nodes.

Balance

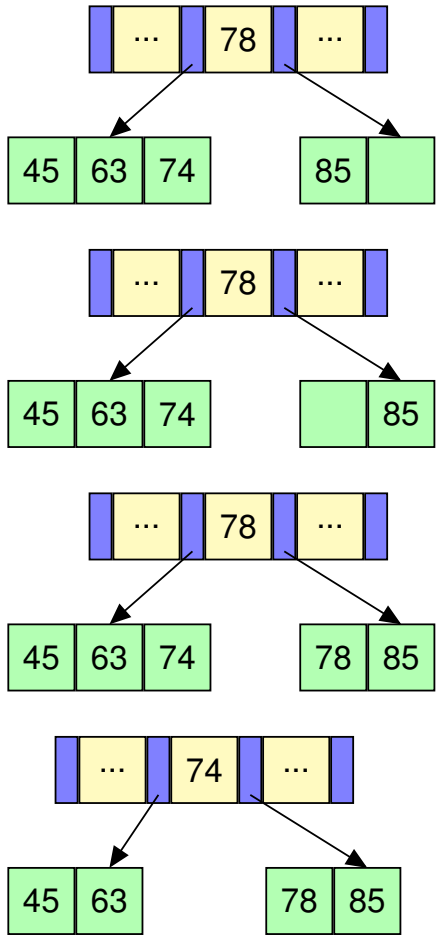
Borrow from left

Original node

Shift entries right

Rotate parent data down

Rotate data up



when the left sibling of the underflow node has more than minimum number of entries

Balance

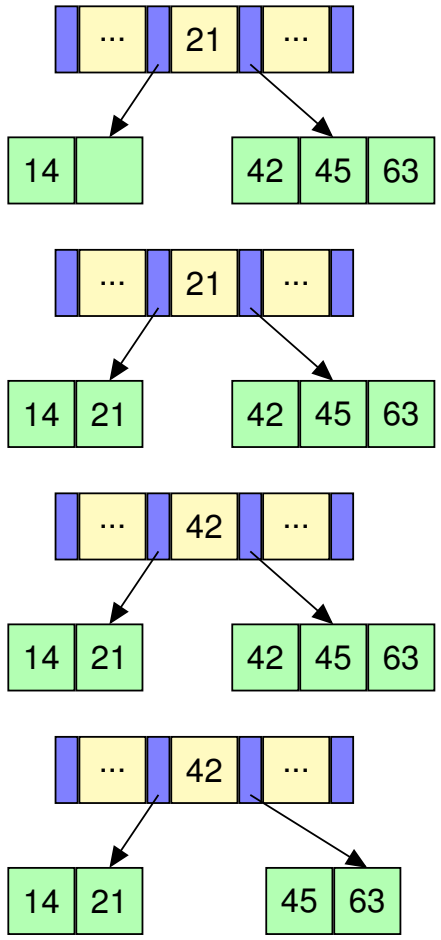
Borrow from right

Original node

Rotate parent data down

Rotate data to parent

Shift entries left

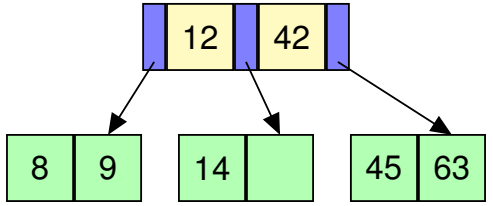


when the right sibling of the underflow node has more than minimum number of entries

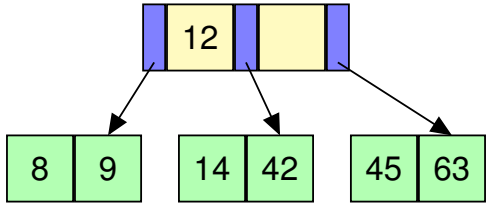
Combine

- when both left and right sibling nodes of the underflow node have minimum number of entries

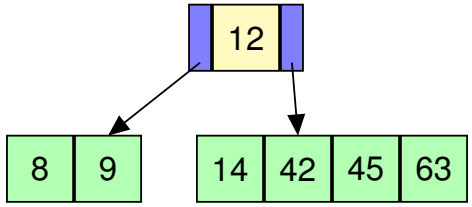
Choose one of its sibling



move the separator down to the underflow node

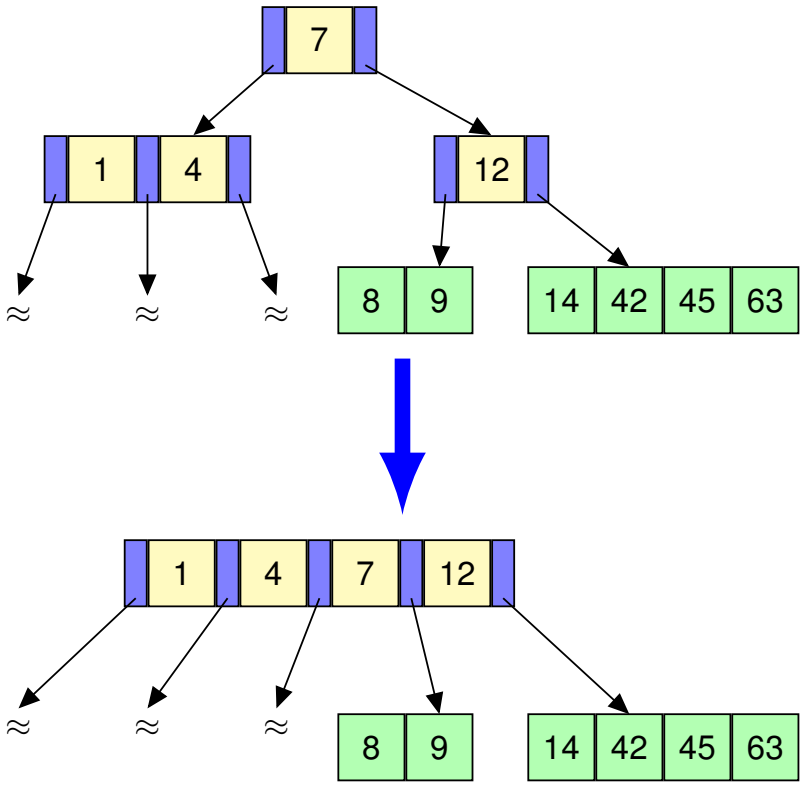


combine the underflow with the chosen sibling



Combine (cont'd)

- if the parent node is underflow, repeat this combination until the root



B-Tree Variations

- B+-Tree:
  - Each data entry must be represented at the leaf level.
  - Each leaf node has one additional pointer to move to the next leaf node
- B*-Tree: the minimum number of (used) entries is two thirds