# Data Structures:
## Trees: Introduction, Tree Traversal

YoungWoon Cha

(Slide credits to Won Kim)

Spring 2022

# Trees

# Real Tree

leaves

root

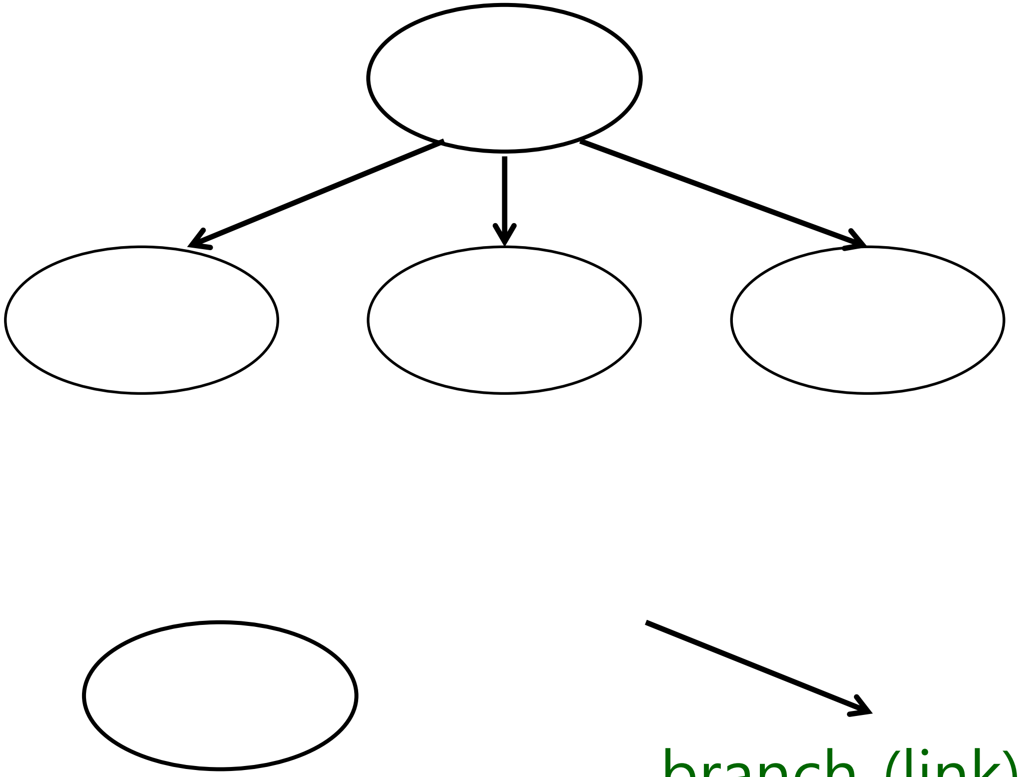
# Definitions for
# Trees in
# Software Data Structures

# Tree Data Structure

root

leaves

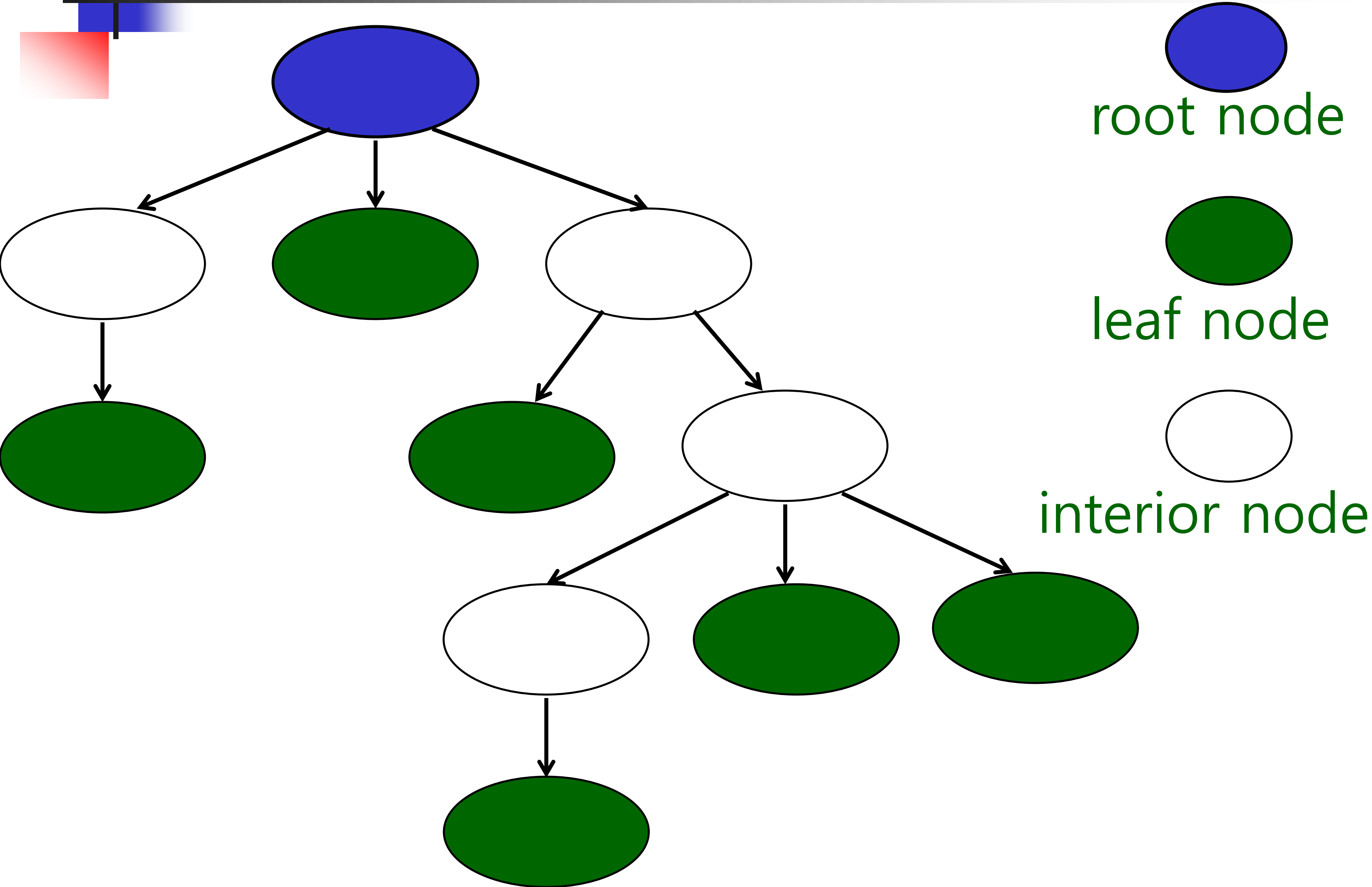
# Nodes and Branches (Links)



node (stores data)

branch (link)
(pointer stored in a node)

# Root, Leaf, Interior (Non-Leaf) Nodes

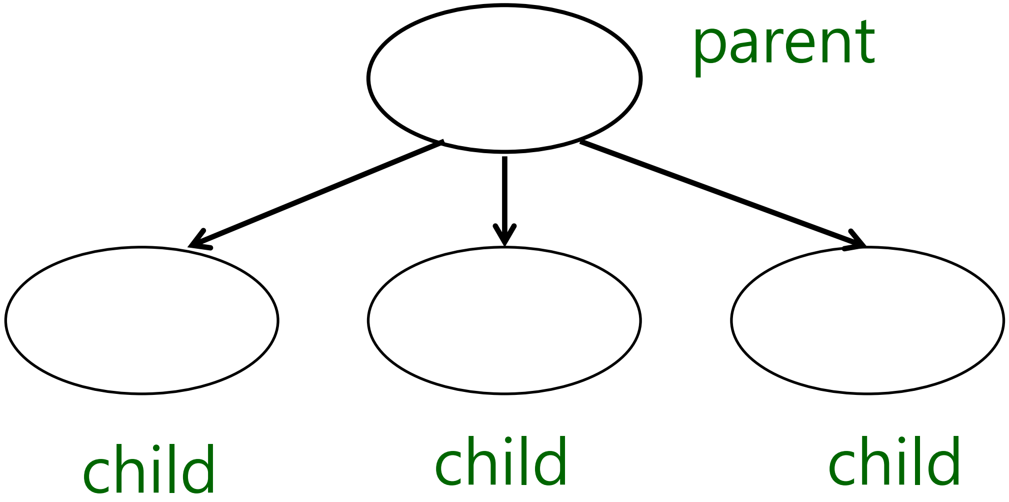


root node

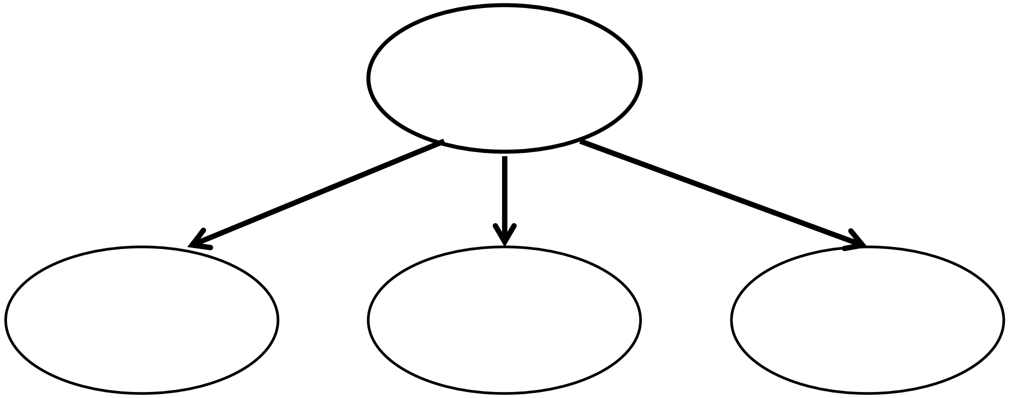
leaf node

interior node

# Parent and Child

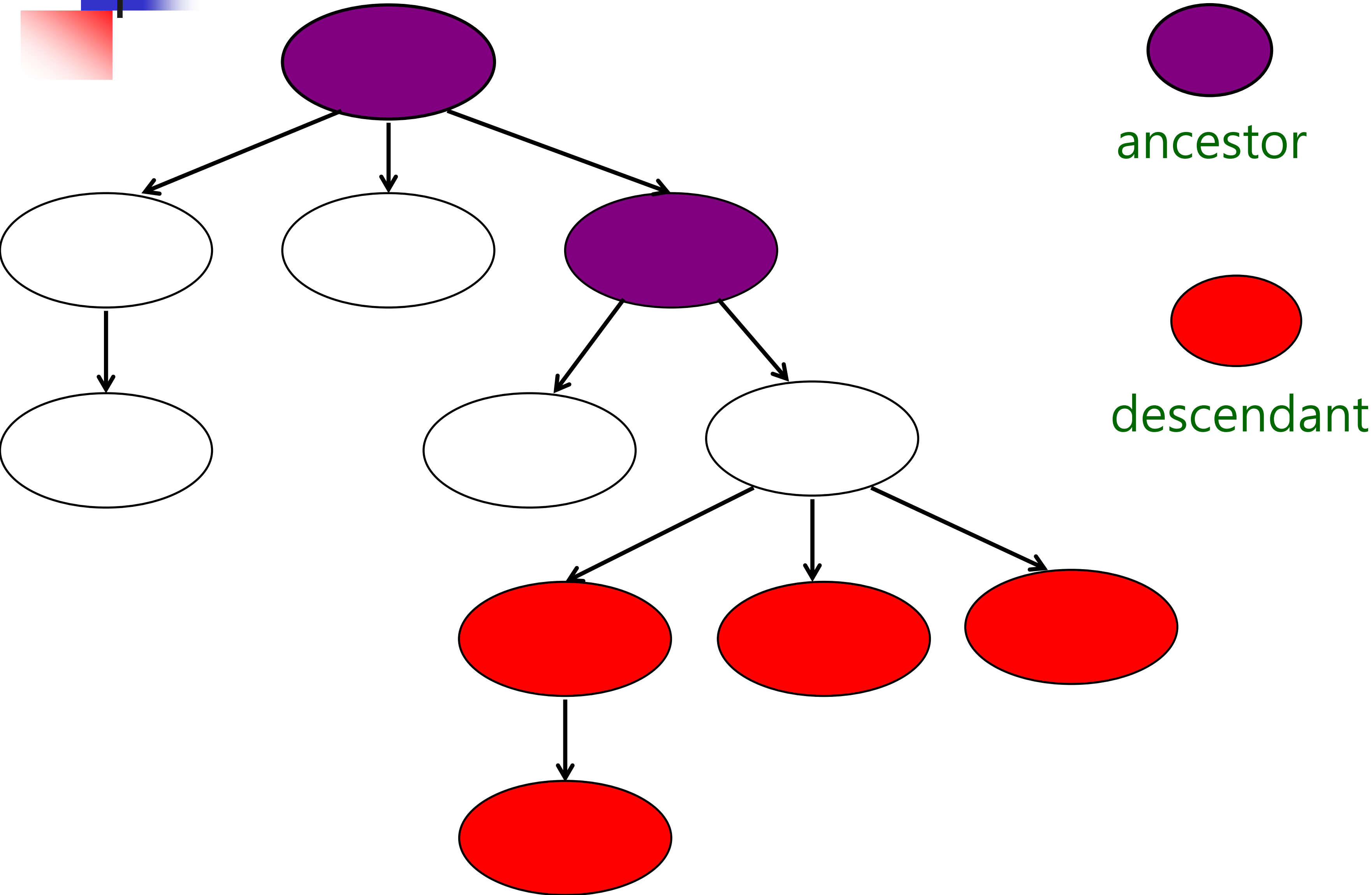


# Siblings (brothers and sisters)



sibling sibling sibling

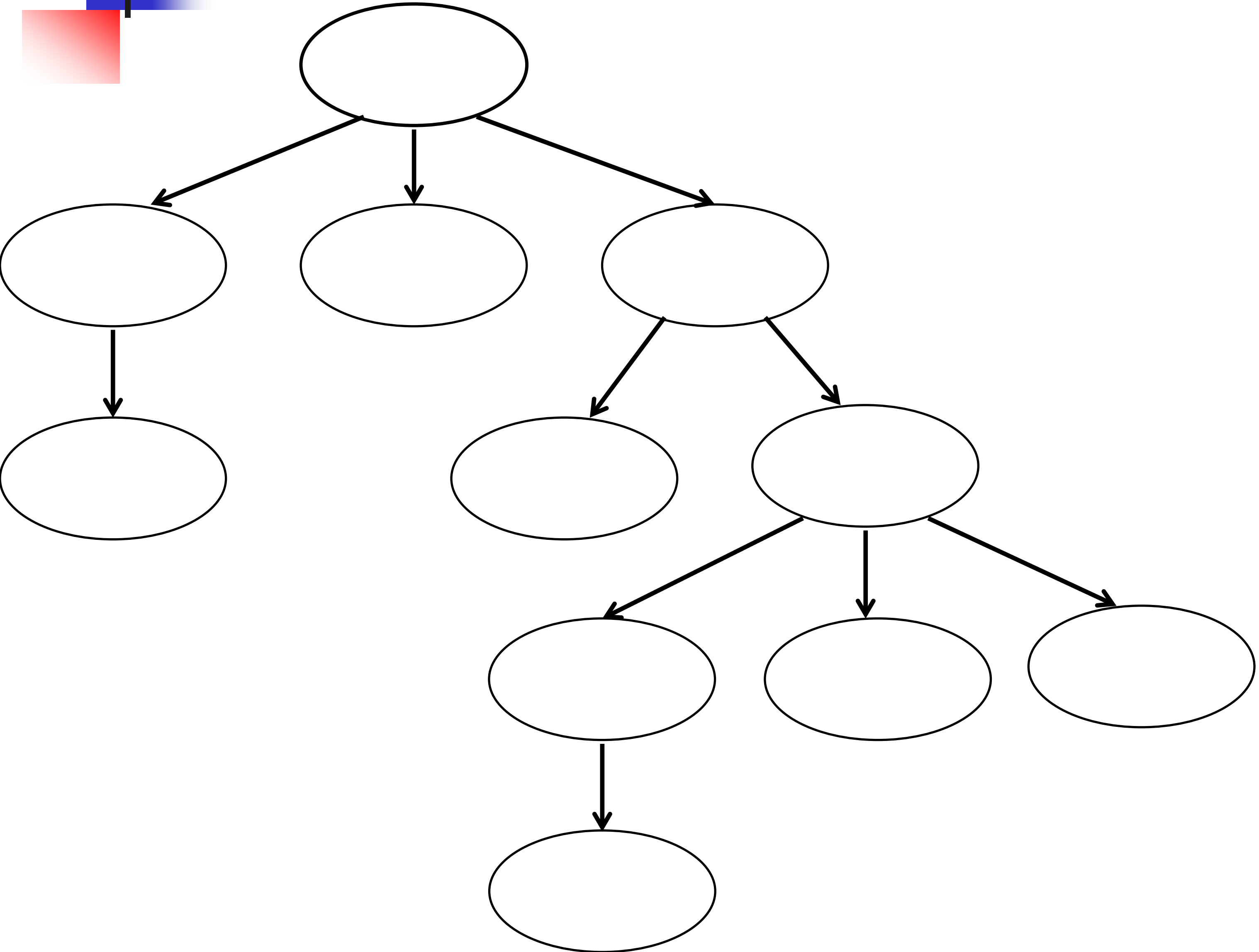
# Ancestors and Descendants



ancestor

descendant

# Depth (Height), Level



1 or 0

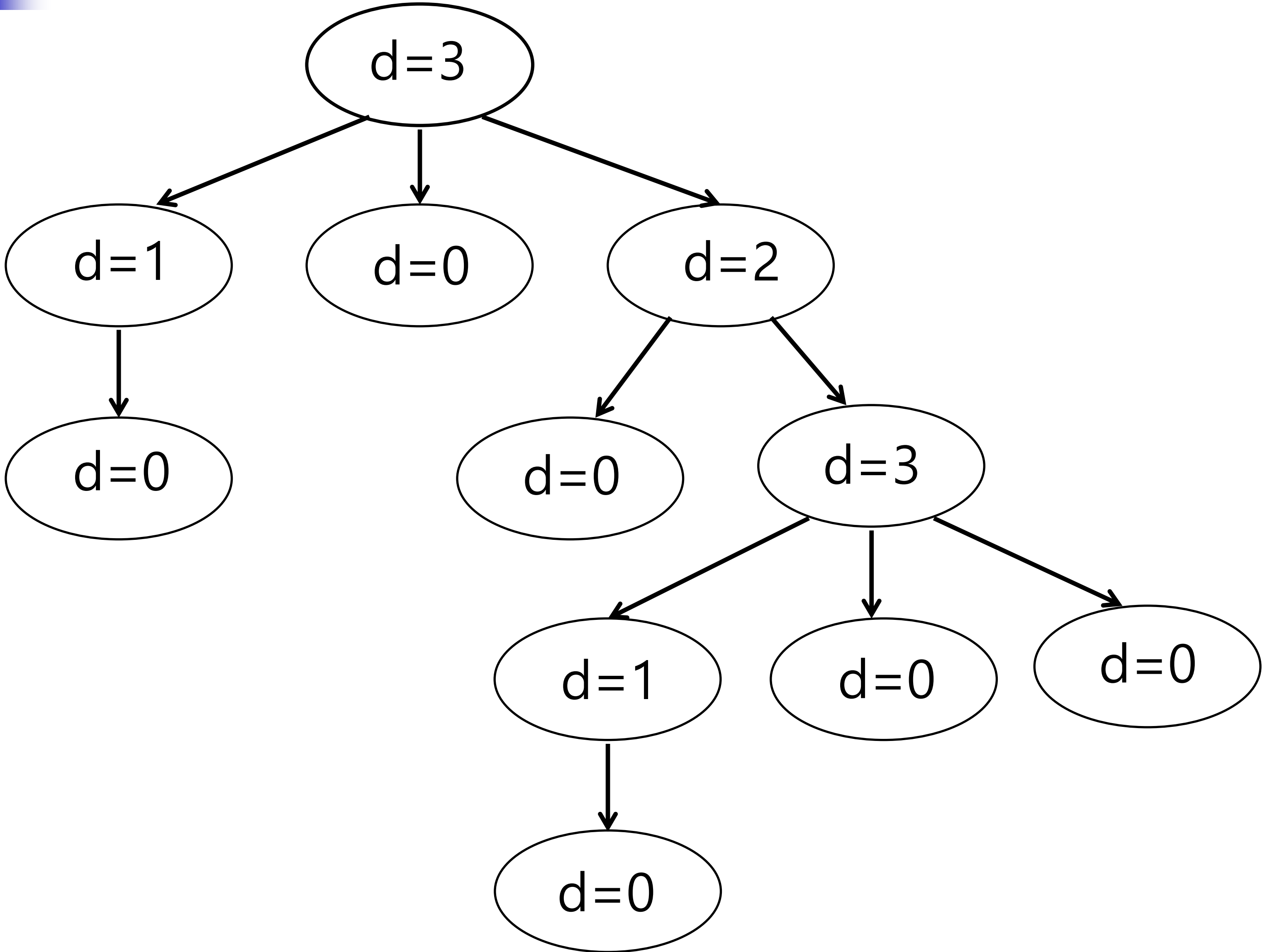
2 or 1

3 or 2

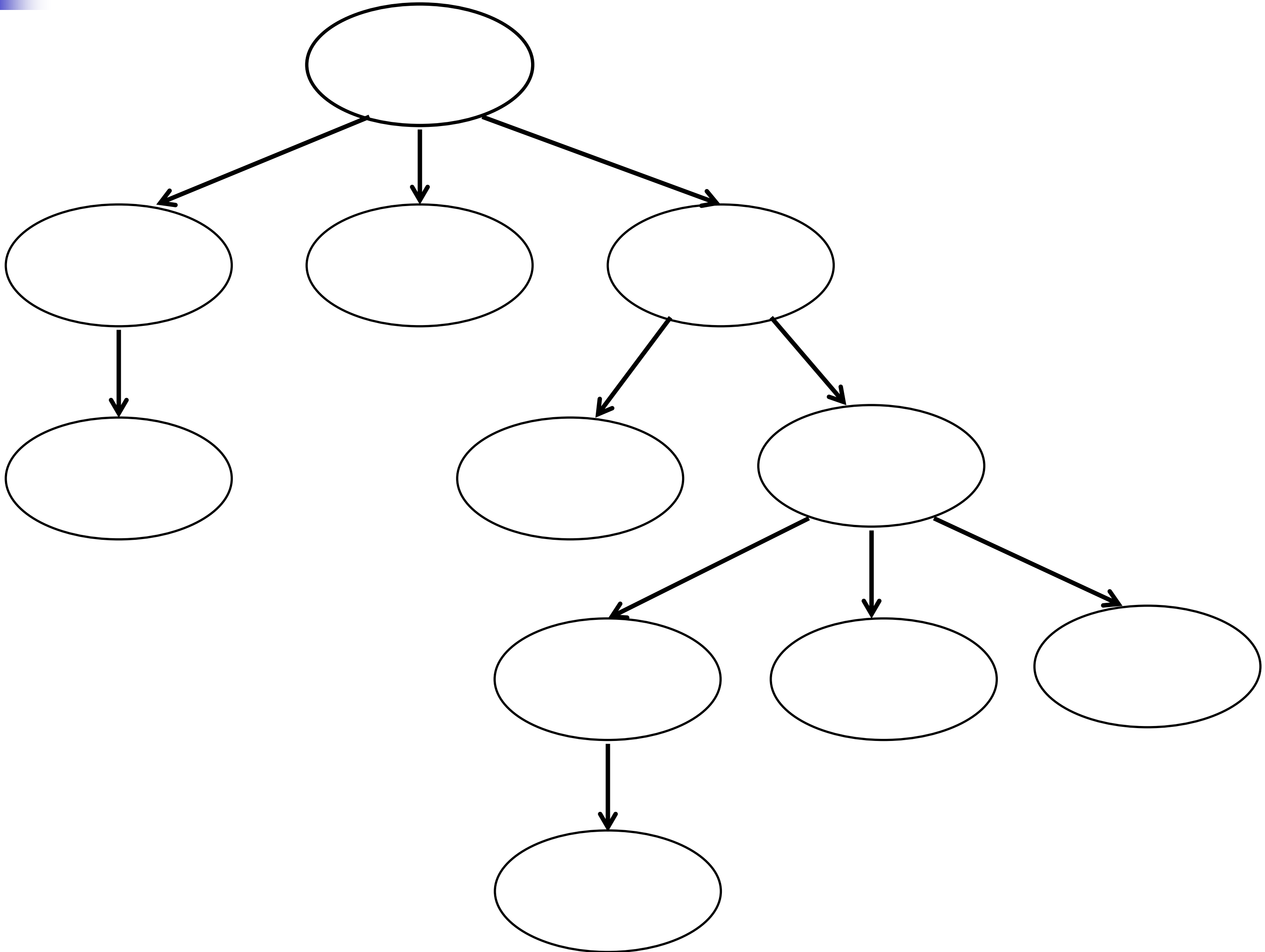
4 or 3

5 or 4

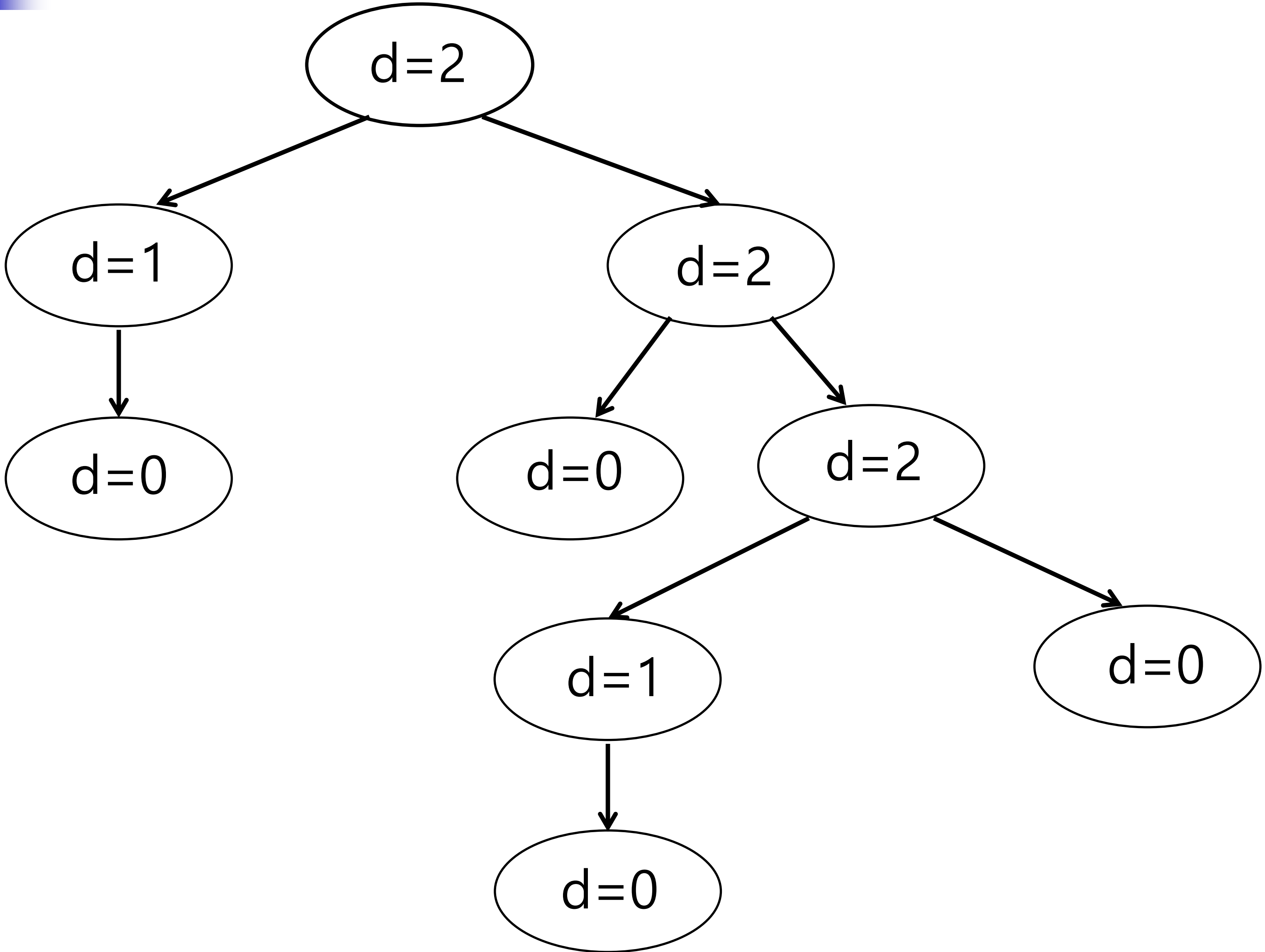
# Degree (Fanout)



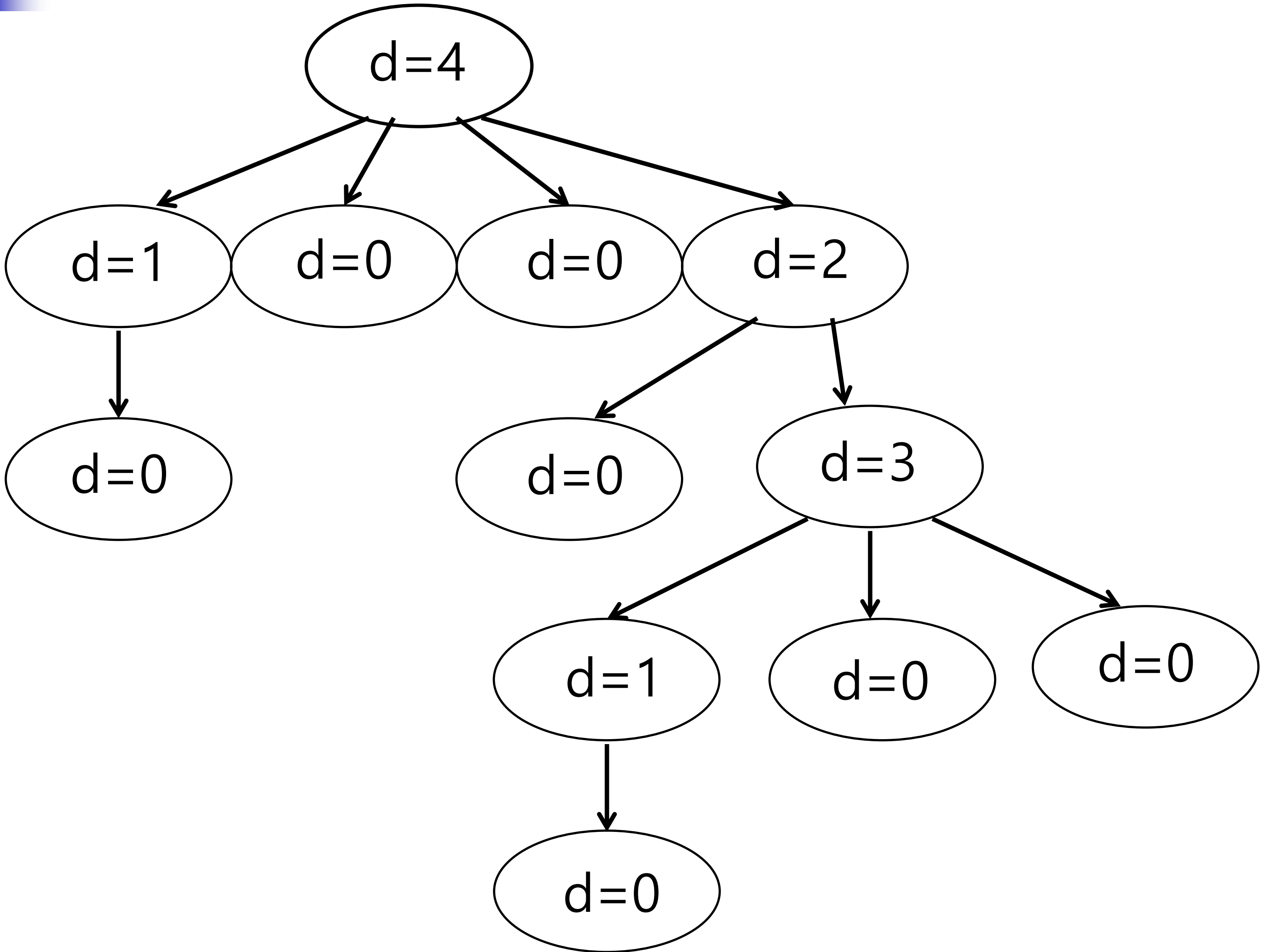
# A General Tree



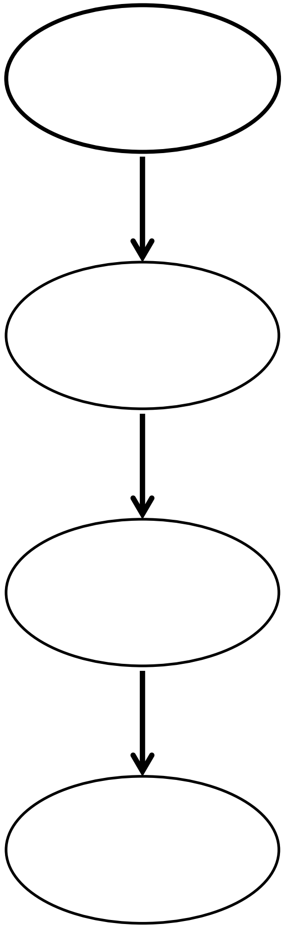
# A Binary Tree (Degree <= 2)



# A Quad Tree (Degree =< 4)

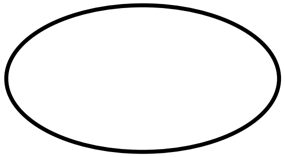


# A Skewed Tree (Degenerate Tree)

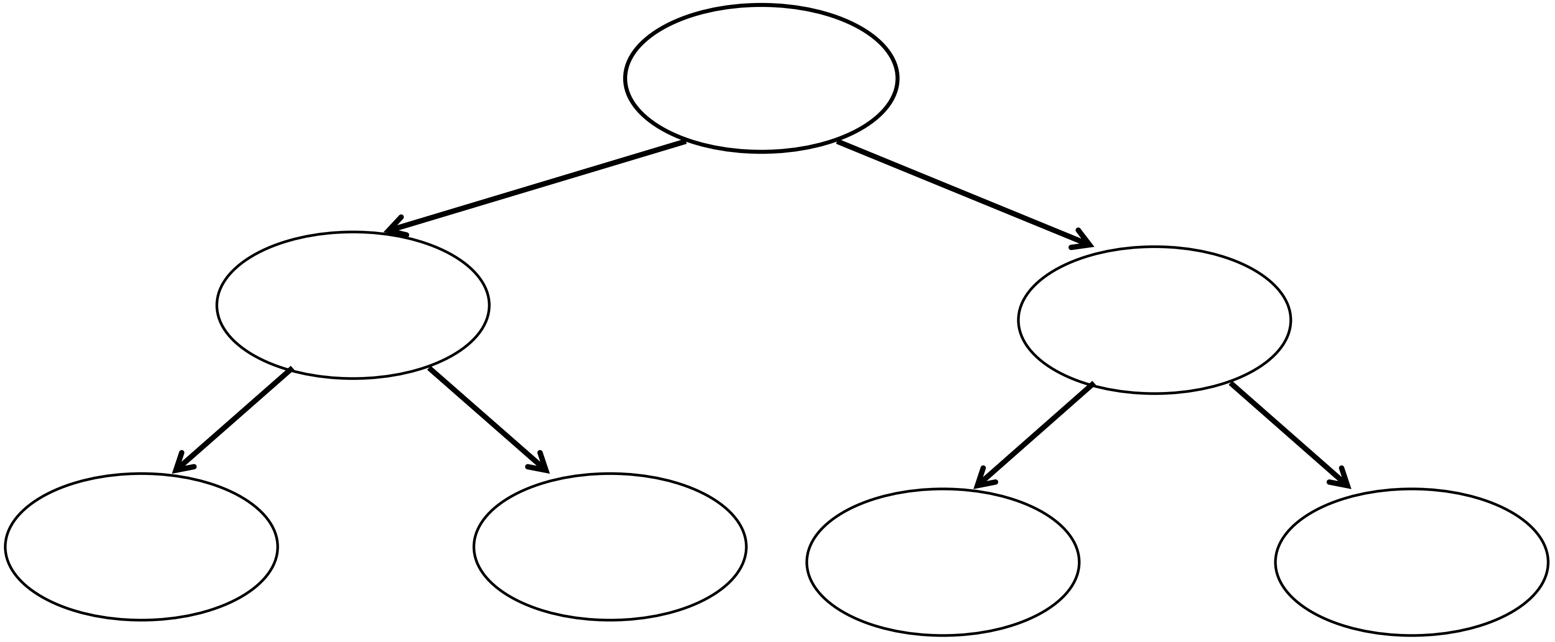


**Degree =< 1**

# (Theoretically A) Tree



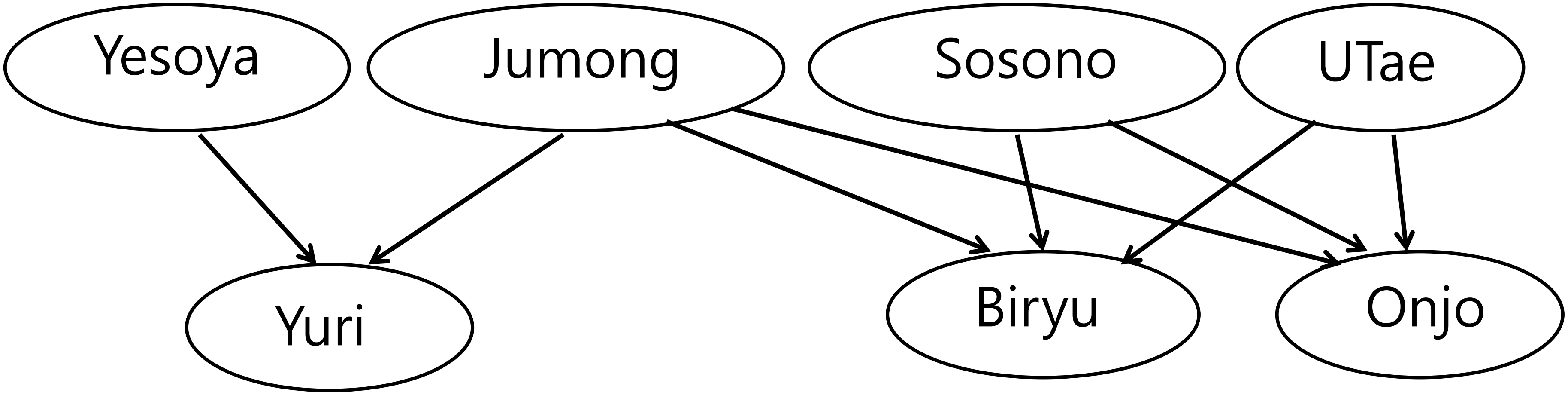
# A Full (Perfect) Tree



# Organization

- Each non-root node has only one parent node.
  - The root node has no parent node.
- Each non-leaf node has one or more child nodes.
  - Each leaf node has zero child node.
- A tree consists of
  - The root node, and j child nodes of the root node.
  - Each of the j child nodes is the root node of a tree.

# Not a Tree: General Genealogy



# Many Types of Tree Data Structures (* We will learn about the red highlighted trees in this course.)

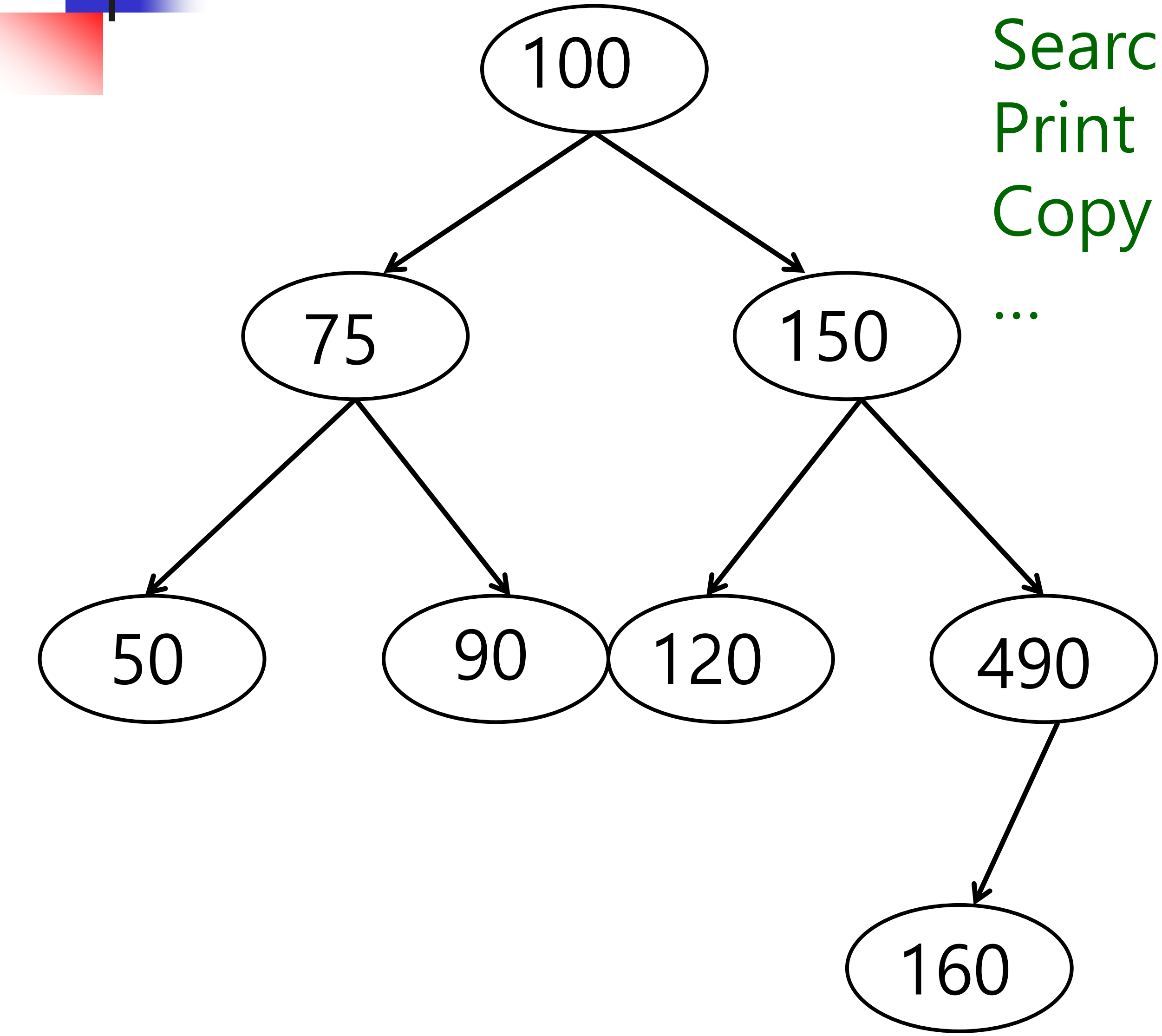
- Binary Tree
  - Binary Search Tree, Heap, Digital Search Tree, Trie
  - Red-Black Tree, AA Tree, Splay Tree,
- Height-Balanced Binary Trees
  - AVL Tree, T-Tree
- n-Way Tree
  - m-Way Trie
  - 2-3 Tree, 2-3-4 Tree
- Height-Balanced m-Way Tree
  - B-Tree, B+-Tree, K-d B-Tree
- Spatial Tree
  - Quad Tree, Oct Tree, K-d Tree, R-Tree, R* Tree

# Summary Definition of a Tree

- A tree is a linked list of data, where a data item is linked to other data items by a certain relationship.
- A search for a data item on a tree proceeds from one data item to another data item that satisfies a certain relationship.
- The hierarchy of a tree is a convenient visualization of the relationships among the data items.

# Tree Traversal

# Tree Traversal (Tree Walk)



Search for a Key
Print Each Key
Copy Each Key
…

# Two Basic Ways to Visit Each Node Once

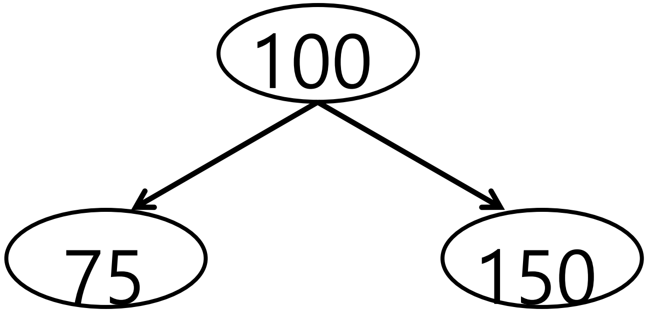
- Visiting a Node: Taking Some Action on the Node
  - printing the data, pushing the data onto a stack, copying data into another tree,…
- Depth-First (Traversal / Search)
- Breadth-First (Traversal / Search)

# Breadth-First Traversal / Search

- Level Order Traversal
    - Visit every node of a level, and move to the next level.

# Depth-First Traversal / Search

- Inorder Traversal
  - (Left Subtree, Visit the Root, Right Subtree)
- Postorder Traversal
  - (Left Subtree, Right Subtree, Visit the Root)
- Preorder Traversal
  - (Visit the Root, Left Subtree, Right Subtree)



# How to Memorize the 3 Types of Depth-First Traversal?

## Think of a Mother with Two Children

# Quiz: What are the 3 ways to share some cookies among the 3 people?

- Assumption: Mother likes the left child more.
- Your answers??

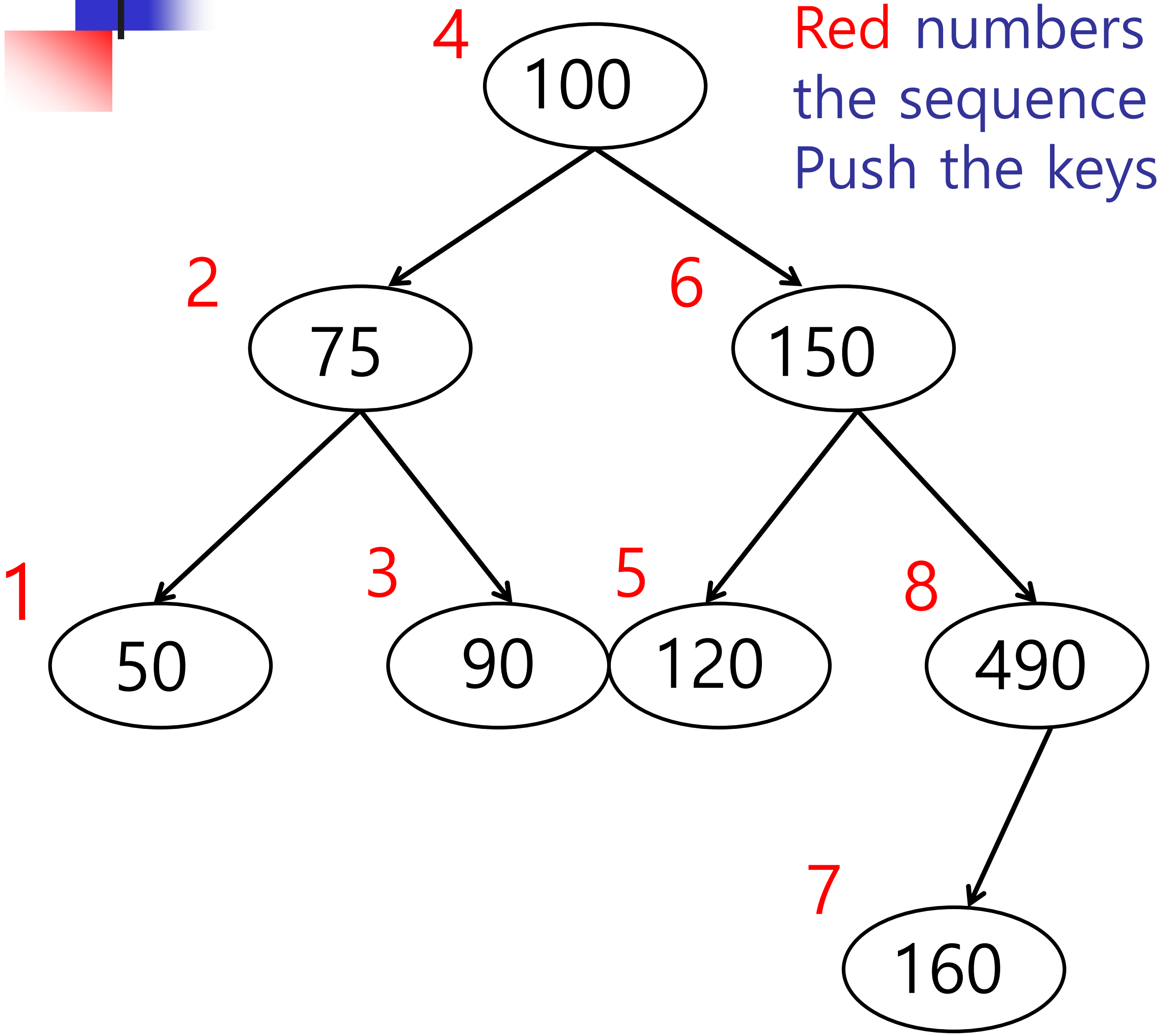
# Answers

- left child first, mother next, right child last
  - (Inorder)
- left child first, right child next, mother last
  - Unselfish mother (postorder)
- mother first, left child next, right child last
  - Selfish mother, (preorder)

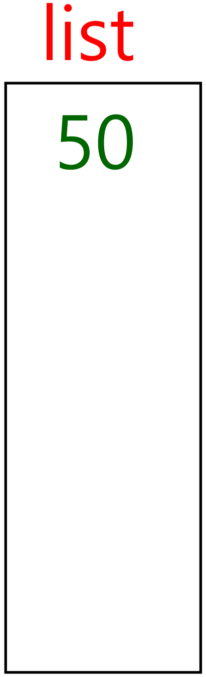
# Inorder Traversal

- First Child, Mother, Second Child
  - print or push the key of the visited node to a stack
- Applications
  - retrieval of a sorted sequence
- Makes Sense Only for a Binary Tree

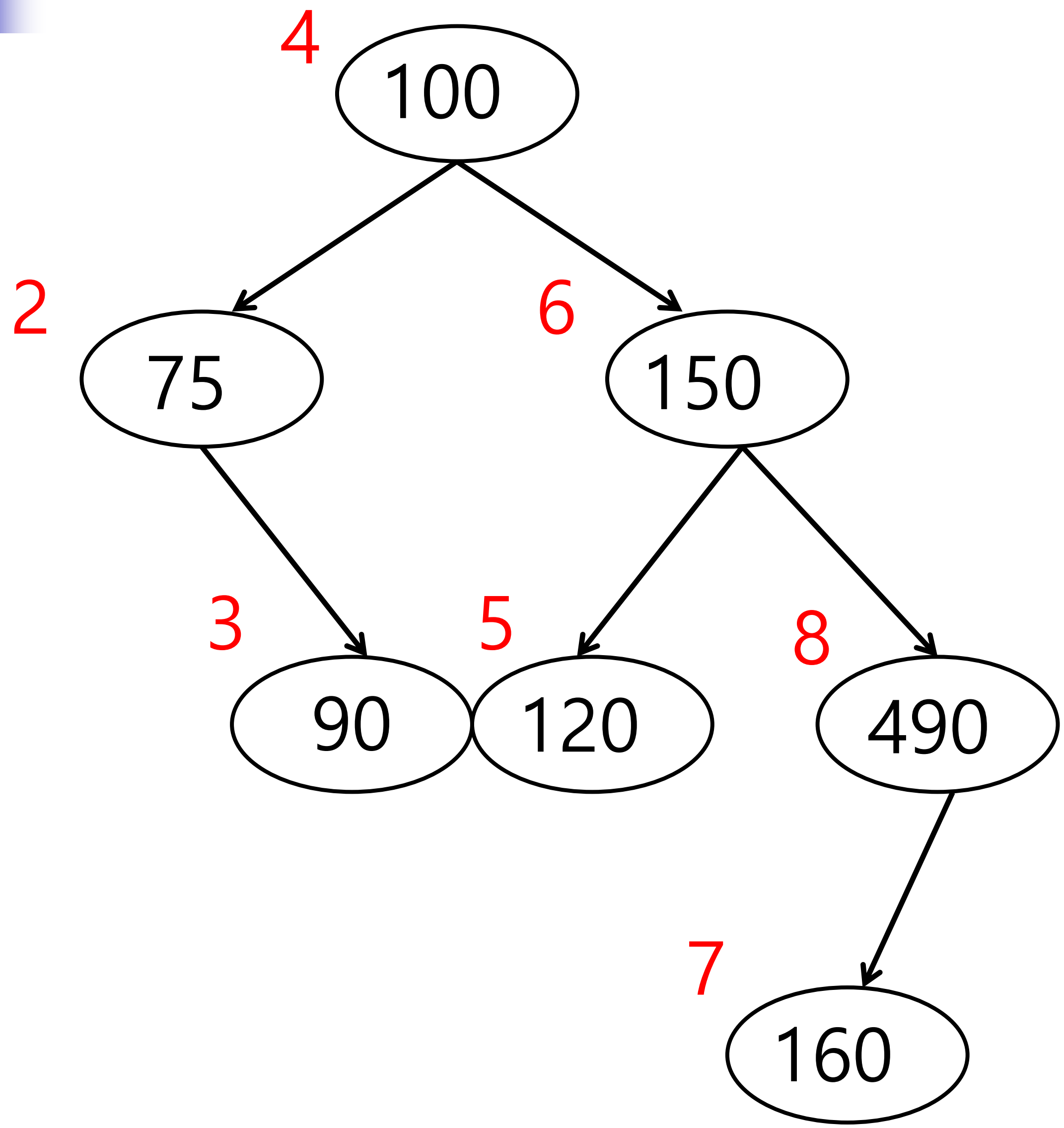
# Inorder Traversal: Example (1/6)



Red numbers show the sequence of visited nodes. Push the keys to a list

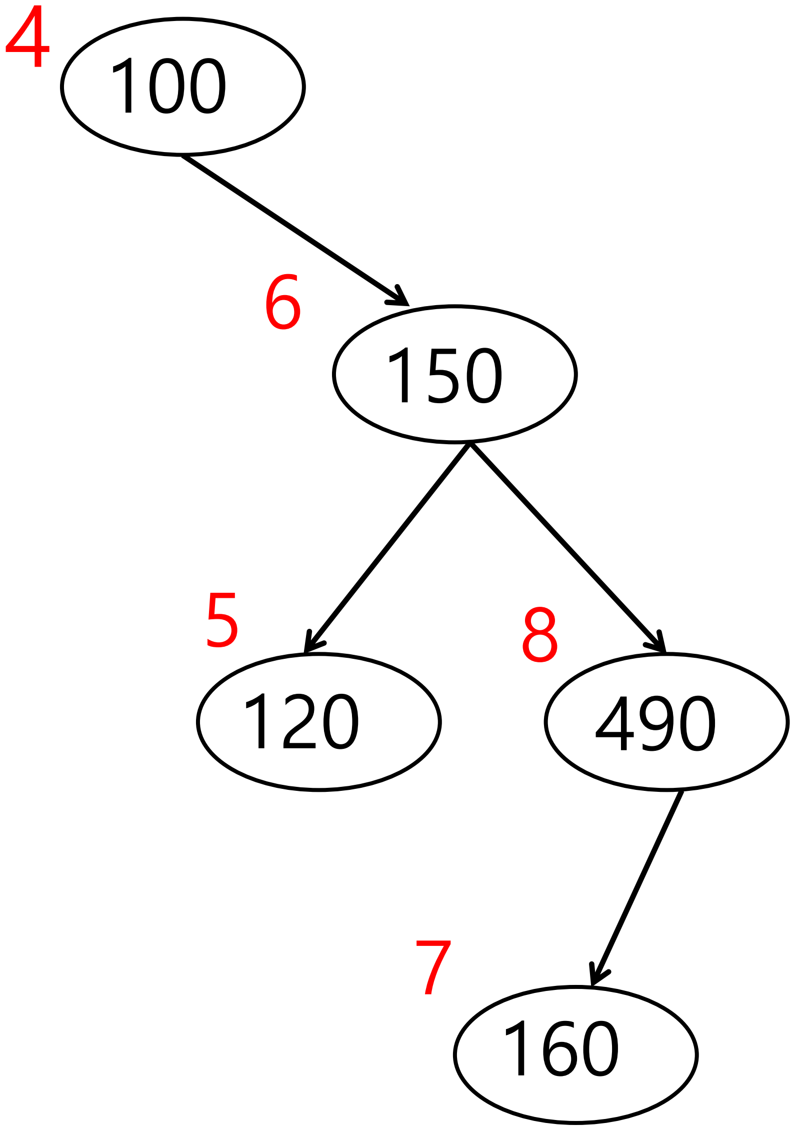


# Inorder Traversal: Example (2/6)



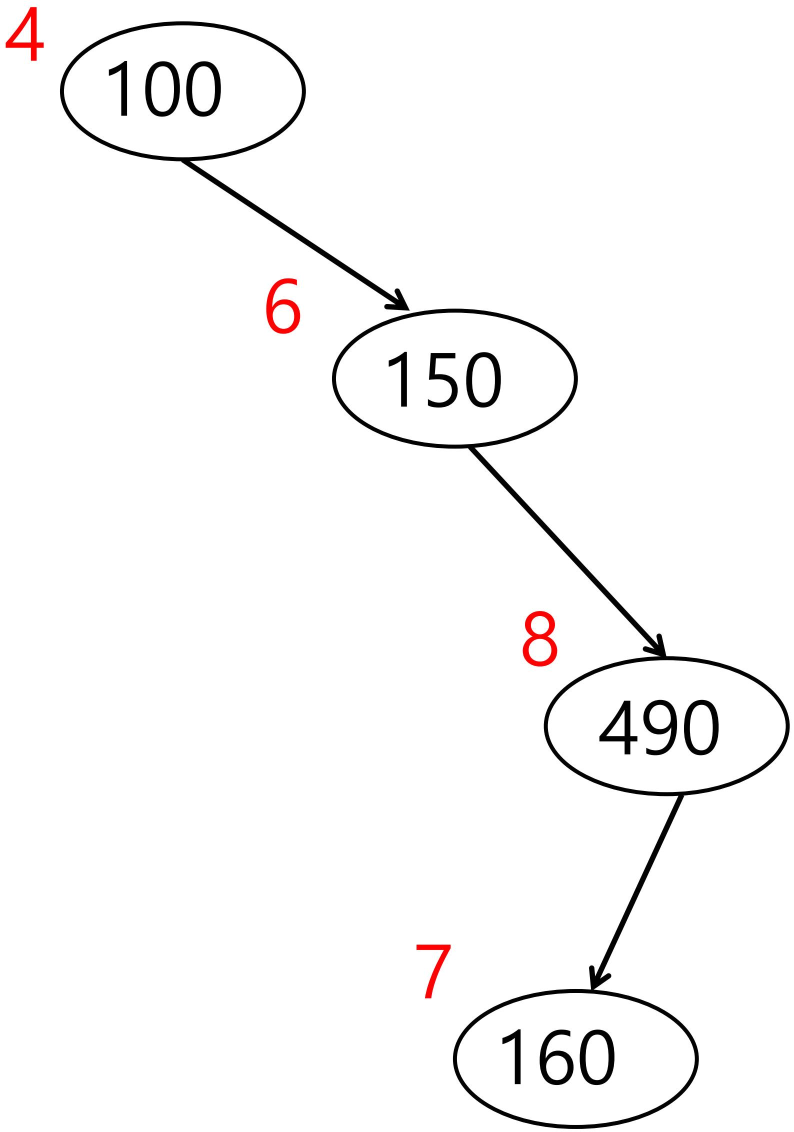
50
75
90

# Inorder Traversal: Example (3/6)



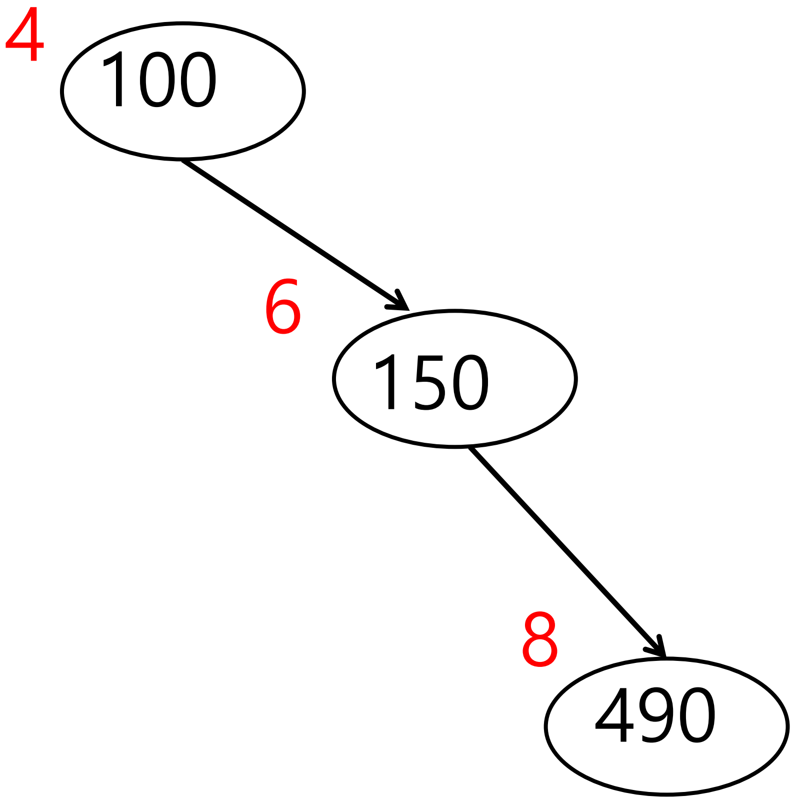
50
75
90
100
120

# Inorder Traversal: Example (4/6)



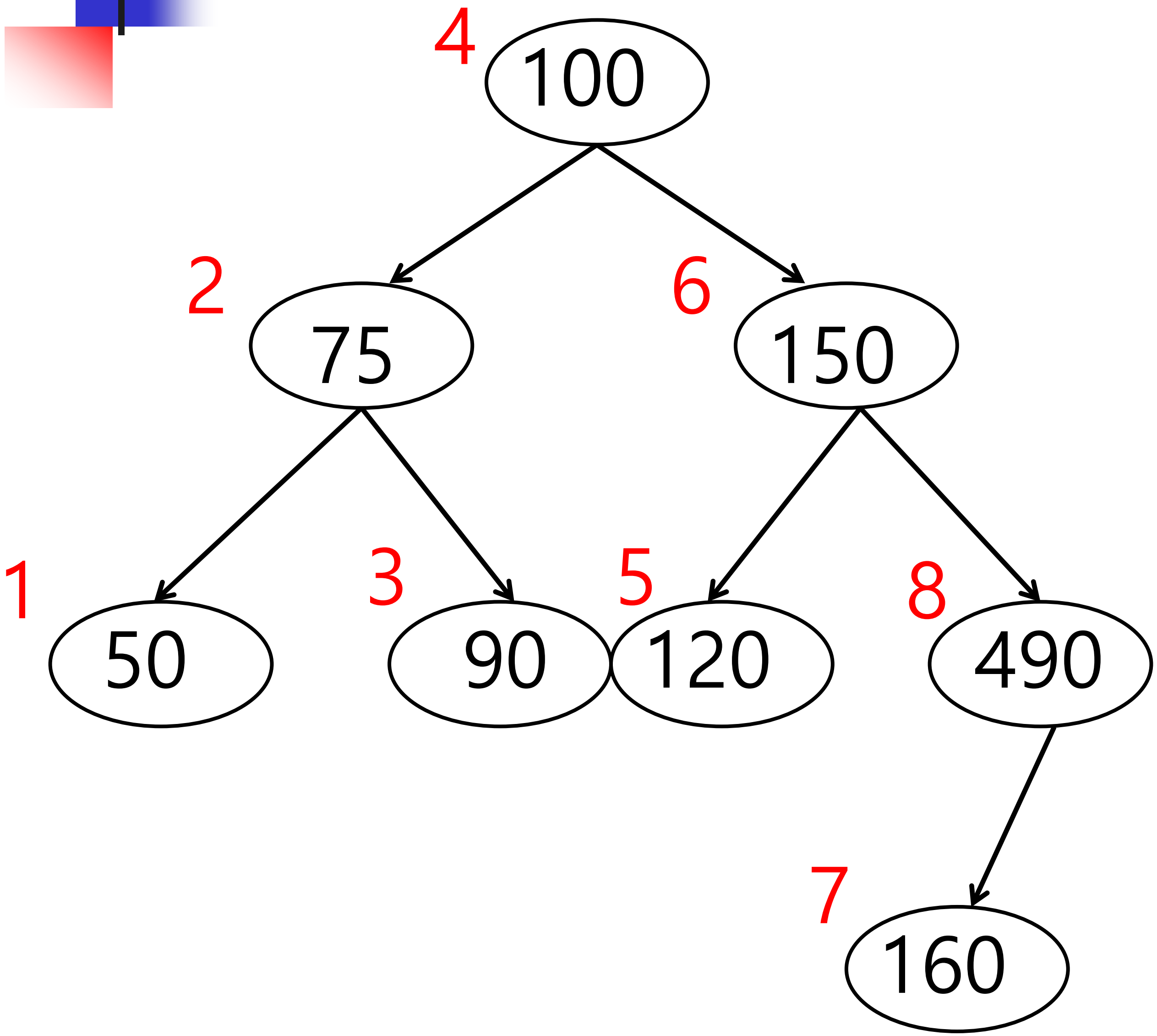
50
75
90
100
120
150
160

# Inorder Traversal: Example (5/6)



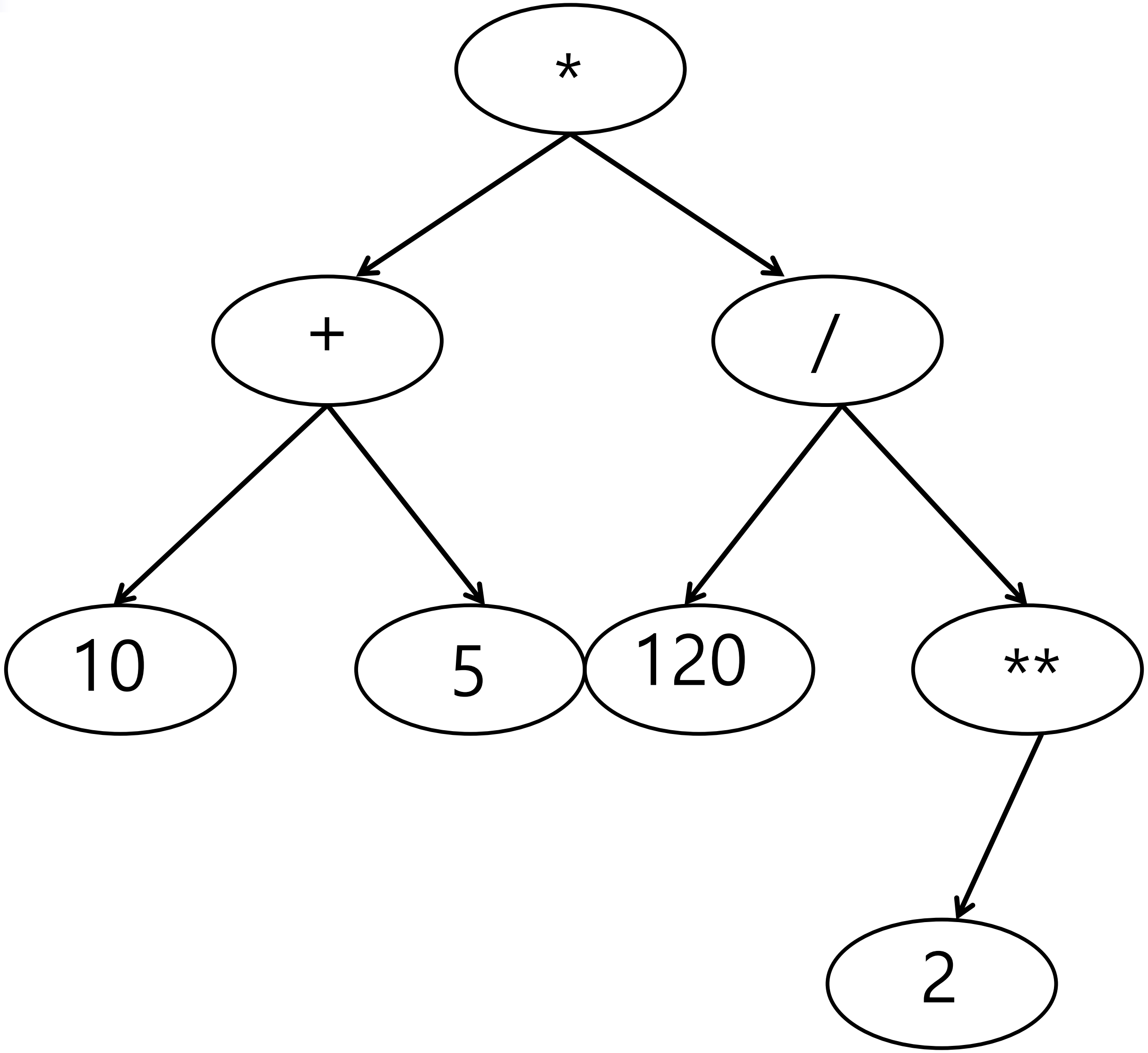
50
75
90
100
120
150
160
490

# Inorder Traversal: Example (6/6)

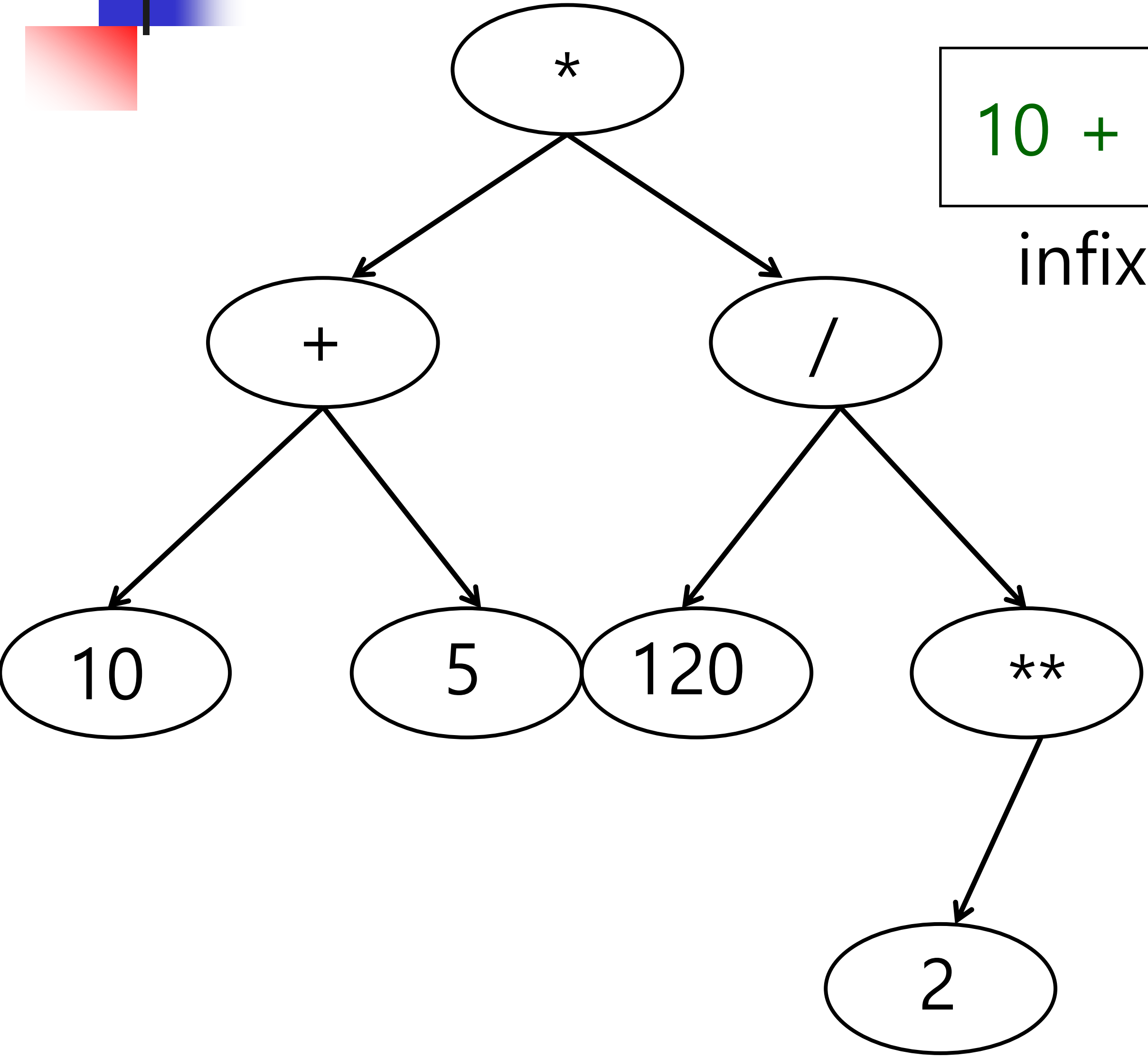


50
75
90
100
120
150
160
490

# Inorder Traversal: Exercise



# Inorder Traversal: Solution



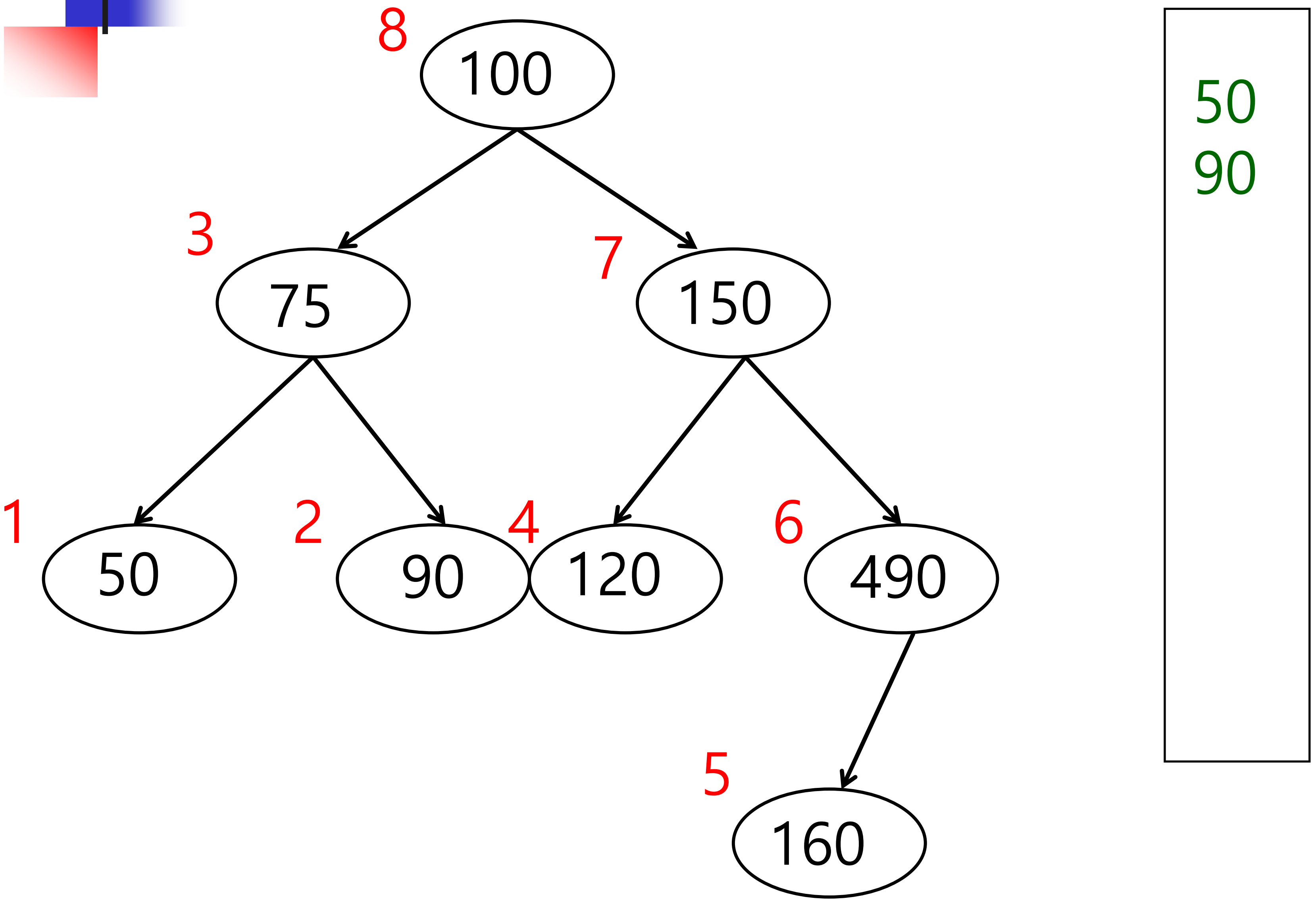
10 + 5 * 120 / 2 **

infix expression

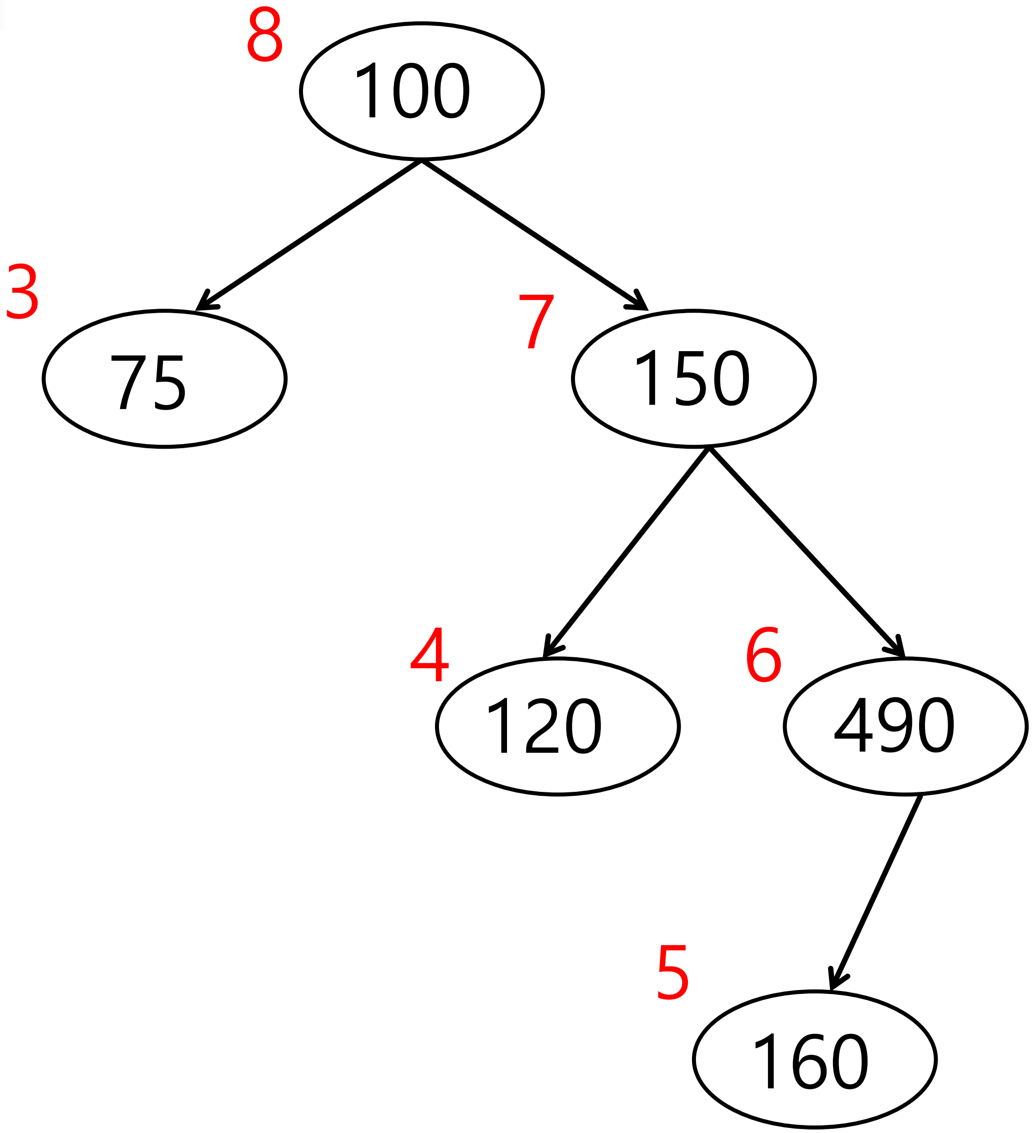
# Postorder Traversal

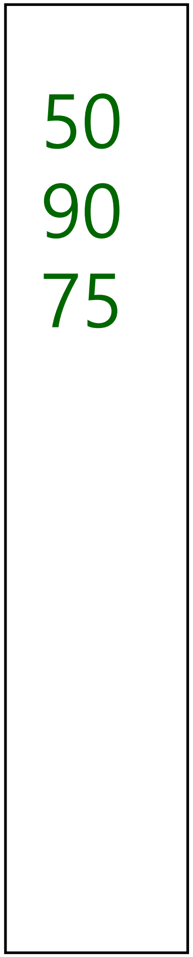
- First Child, Second Child, Mother
- Applications
  - (compiler) postfix expression evaluation
    - using a queue-like order, using a stack

# Postorder Traversal: Example (1/8)

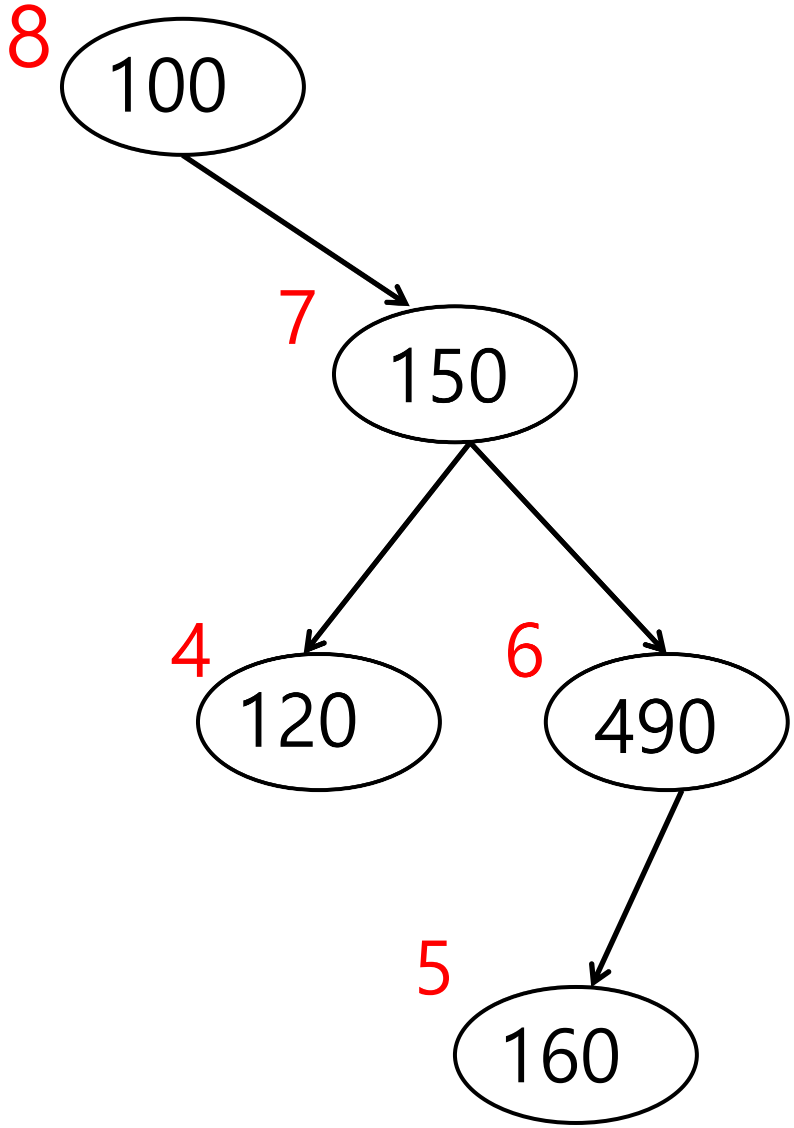


# Postorder Traversal: Example (2/8)



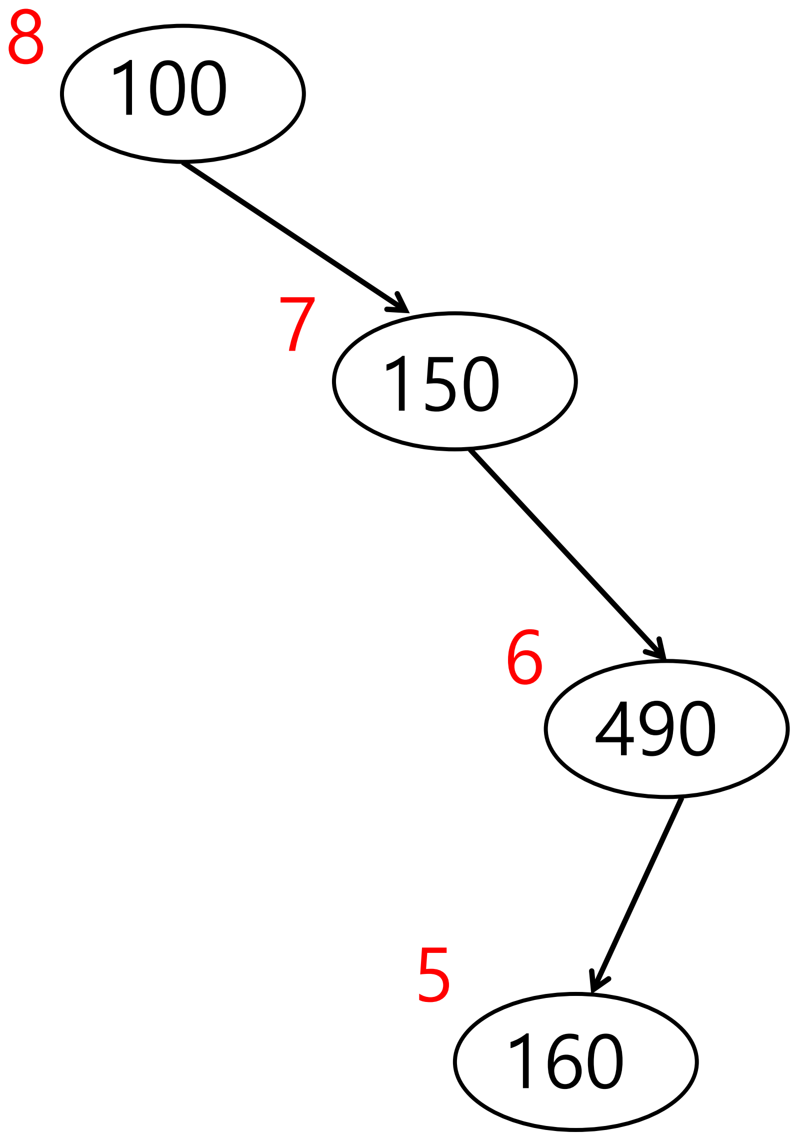


# Postorder Traversal: Example (3/8)



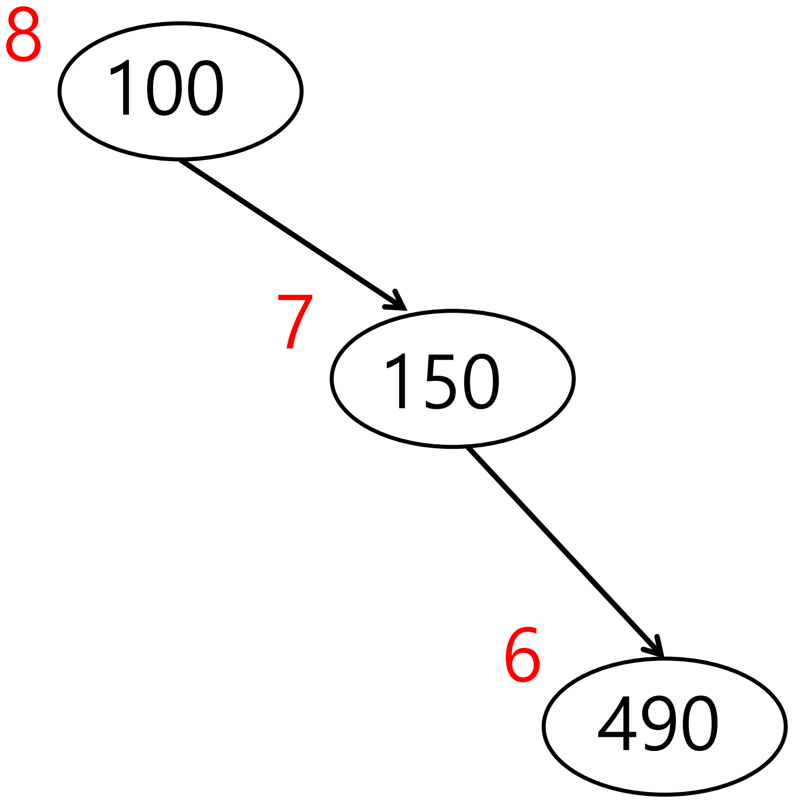
50
90
75
120

# Postorder Traversal: Example (4/8)



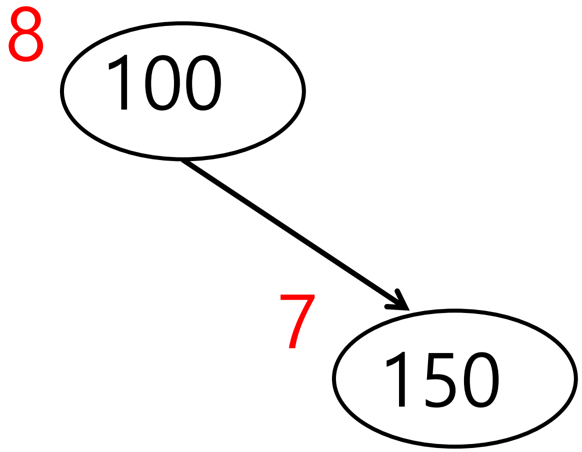
50
90
75
120
160

# Postorder Traversal: Example (5/8)



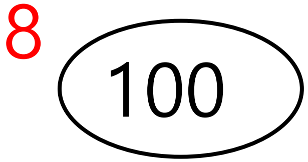
50
90
75
120
160
490

# Postorder Traversal: Example (6/8)



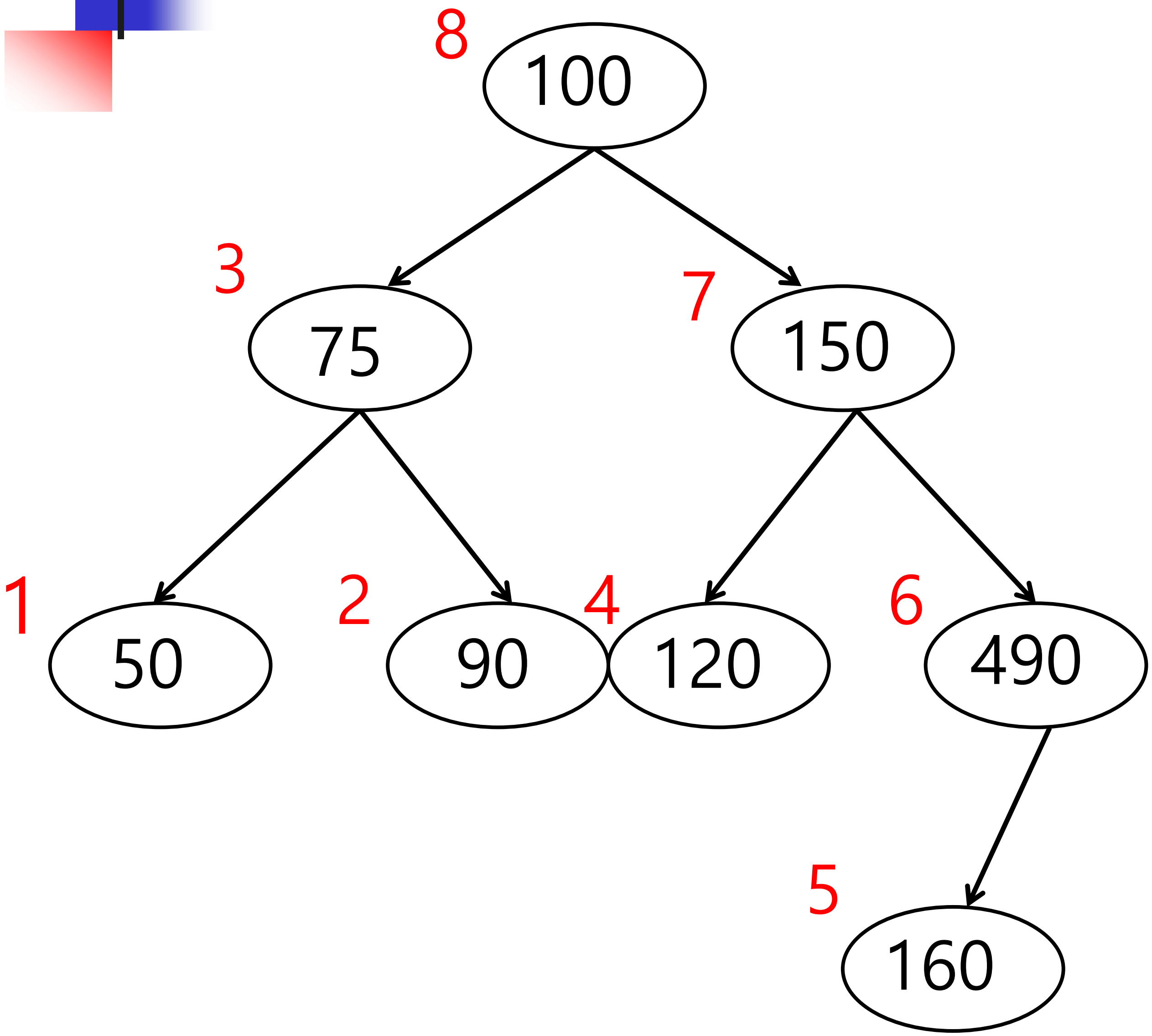
50
90
75
120
160
490
150

# Postorder Traversal: Example (7/8)



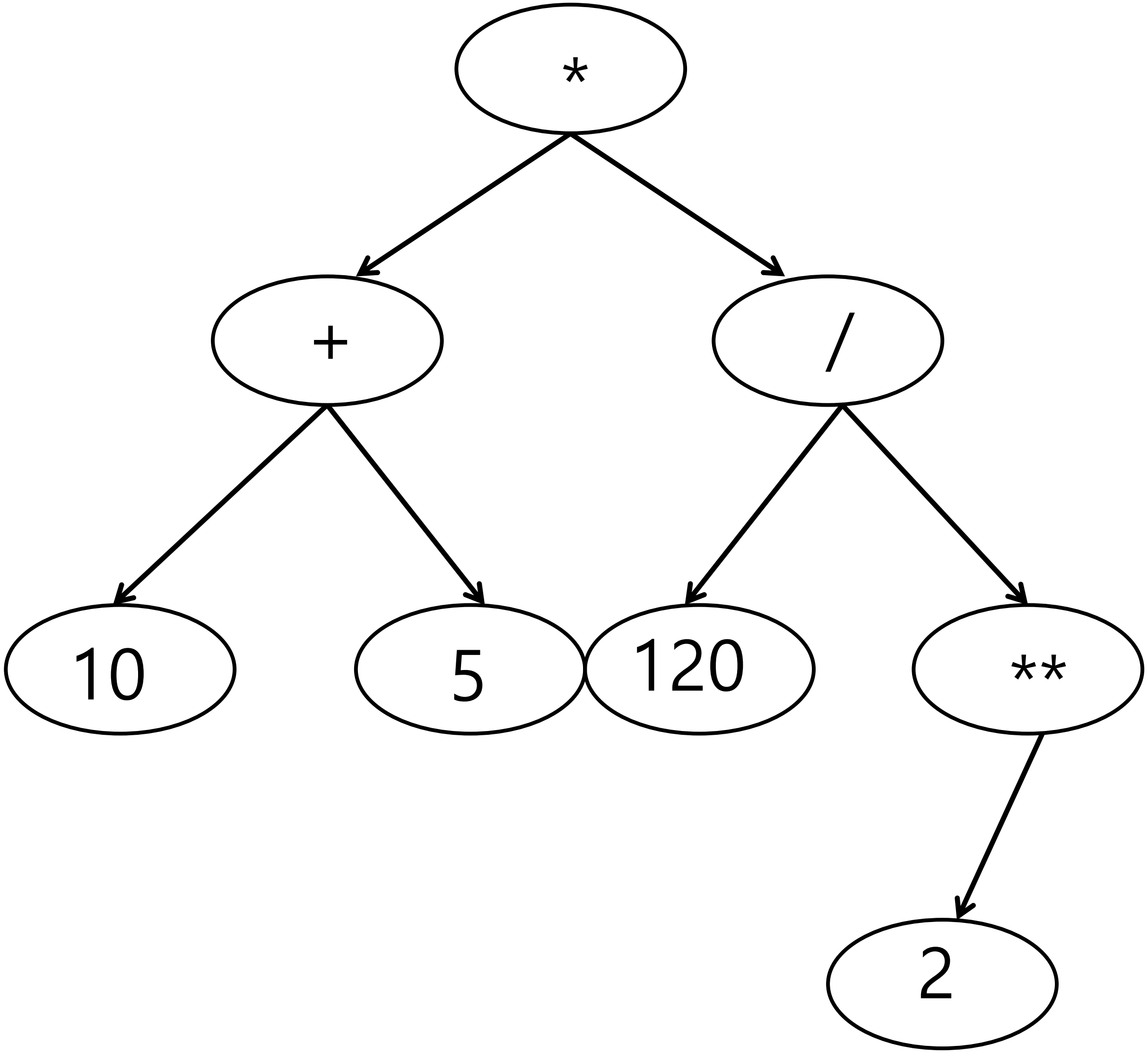
50
90
75
120
160
490
150
100

# Postorder Traversal: Example (8/8)

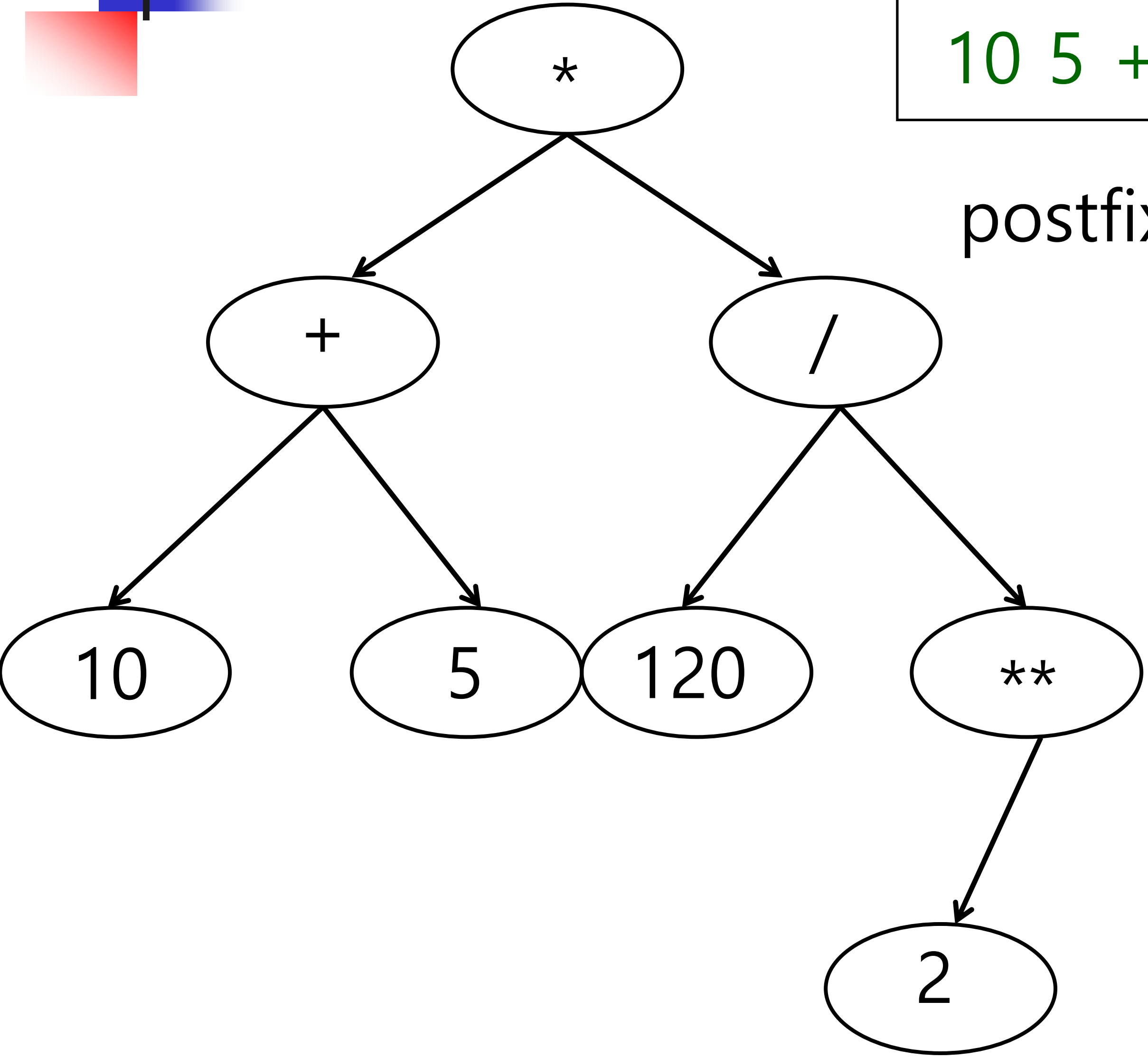


50
90
75
120
160
490
150
100

# Postorder Traversal: Exercise



# Postorder Traversal: Solution



10 5 + 120 2 ** / *

postfix expression

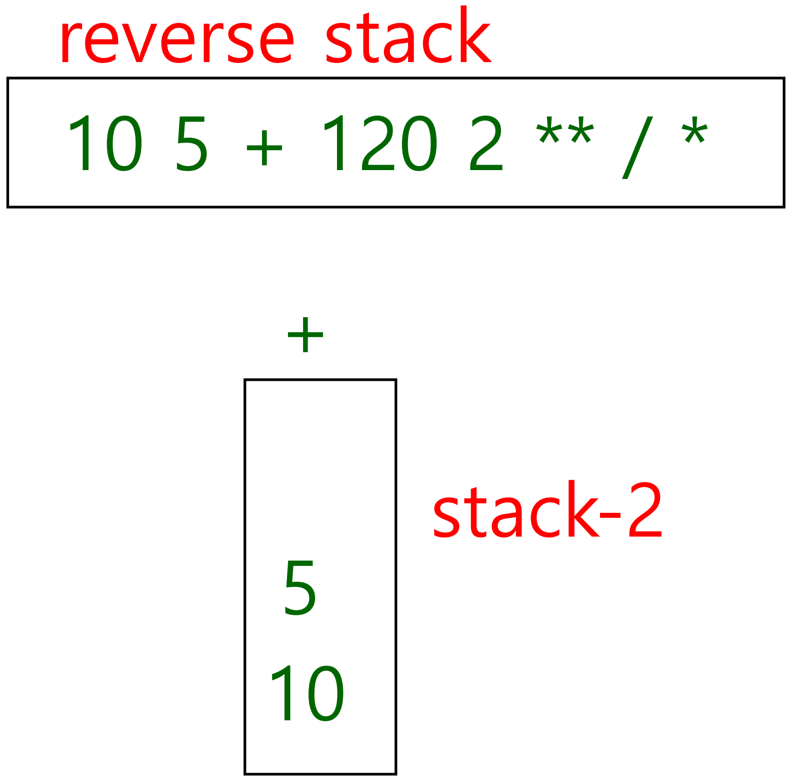
# Evaluating a Postfix Expression Using a Stack (How compilers evaluate expressions) (1/5)

Pop stack-1.

If it is a number,
    push it to stack-2

If it is an operator,
    pop stack-2 and compute,

Push the result to stack-2



reverse stack

10 5 + 120 2 ** / *

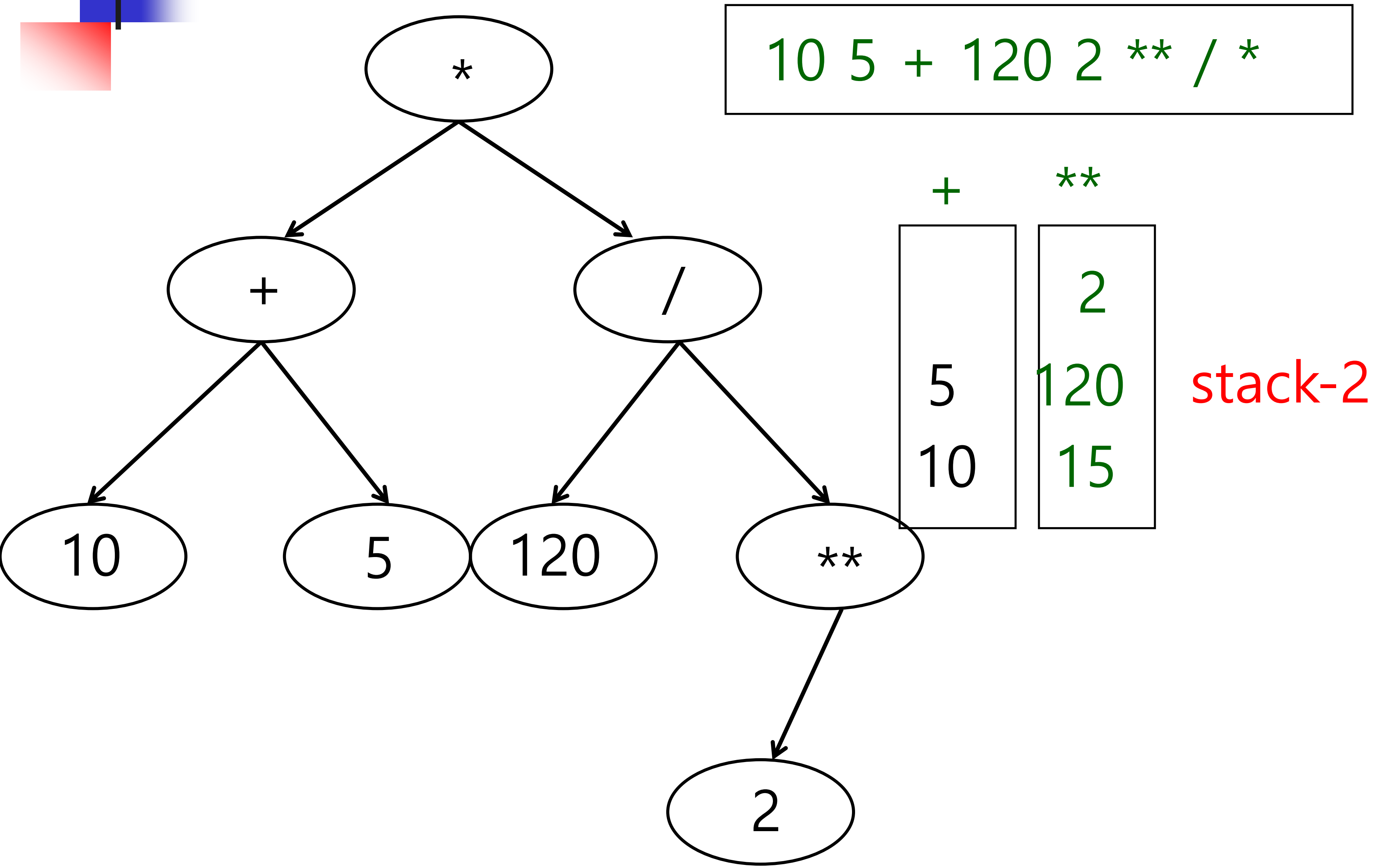
+

5

10

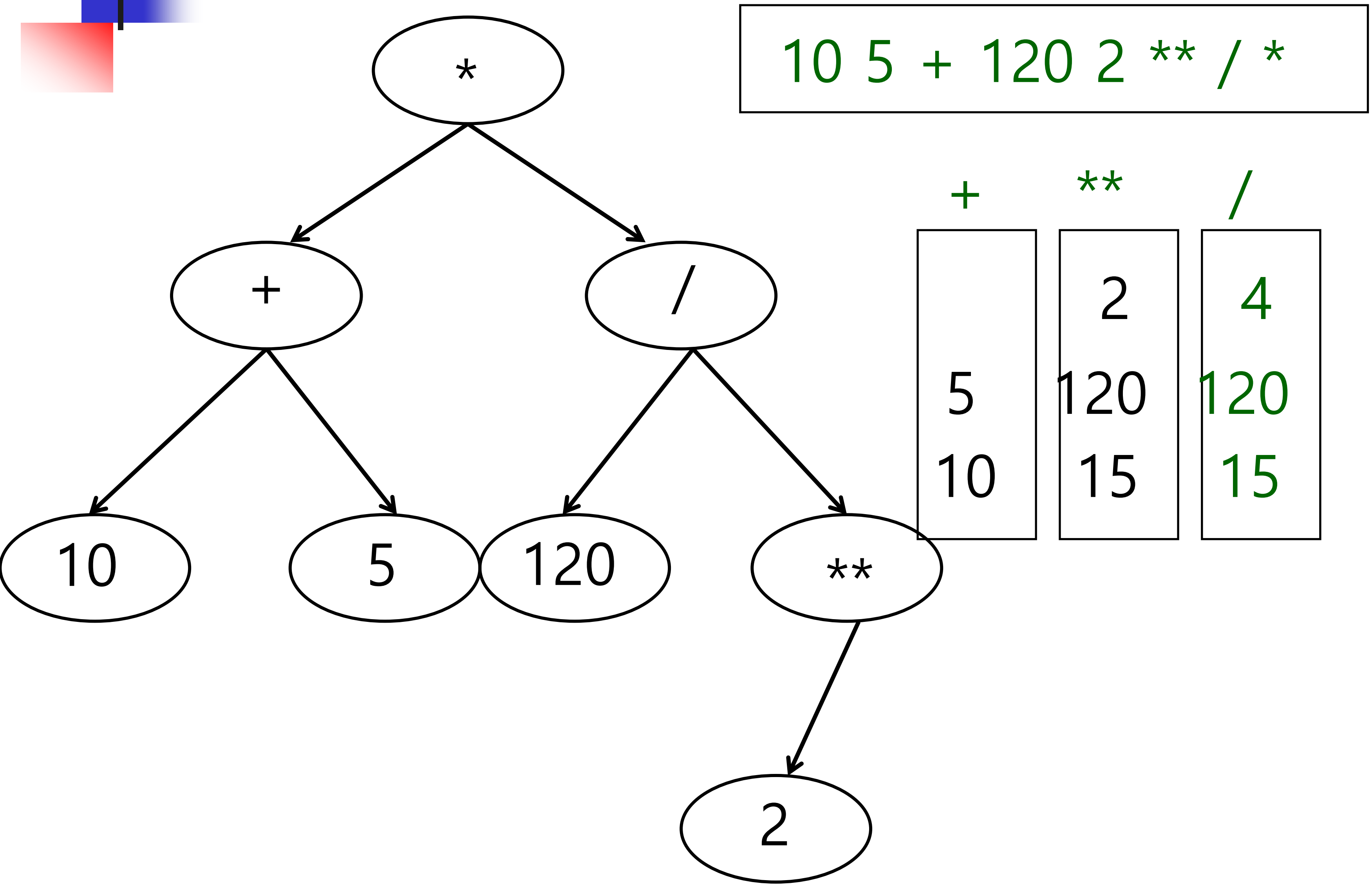
stack-2

# (2/5)



10 5 + 120 2 ** / *

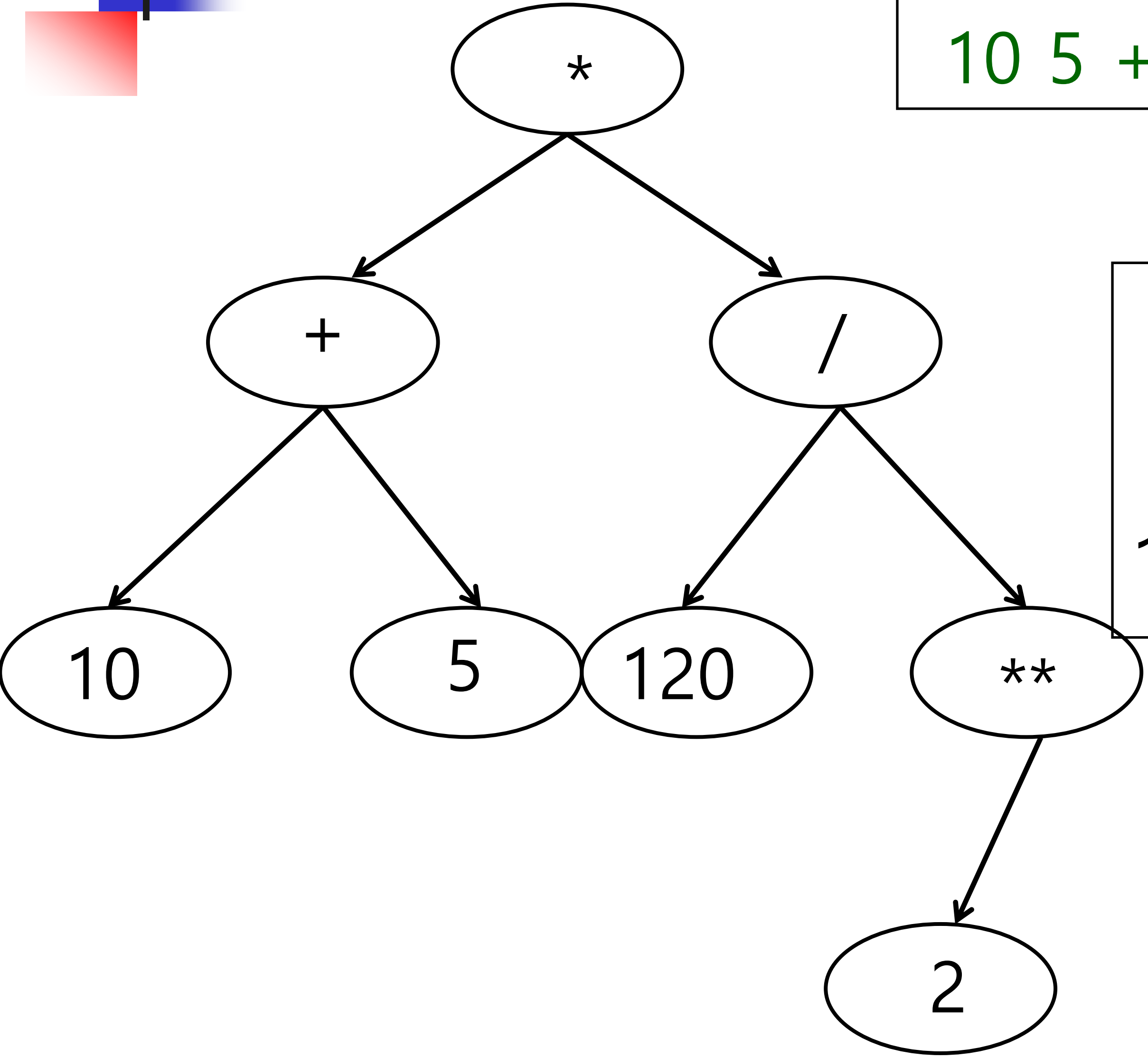
# (3/5)

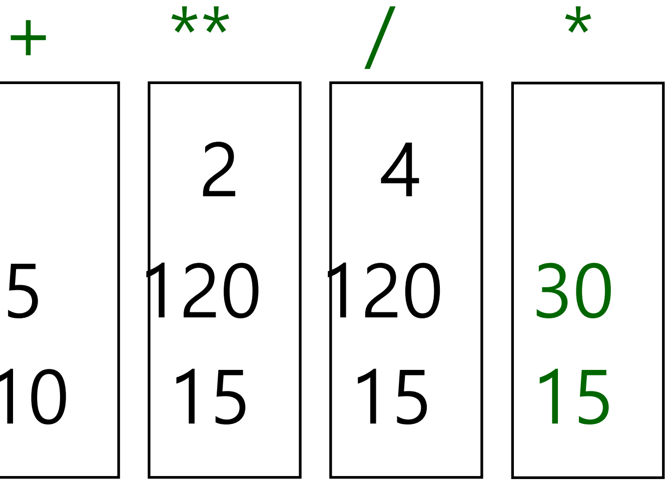


10 5 + 120 2 ** / *

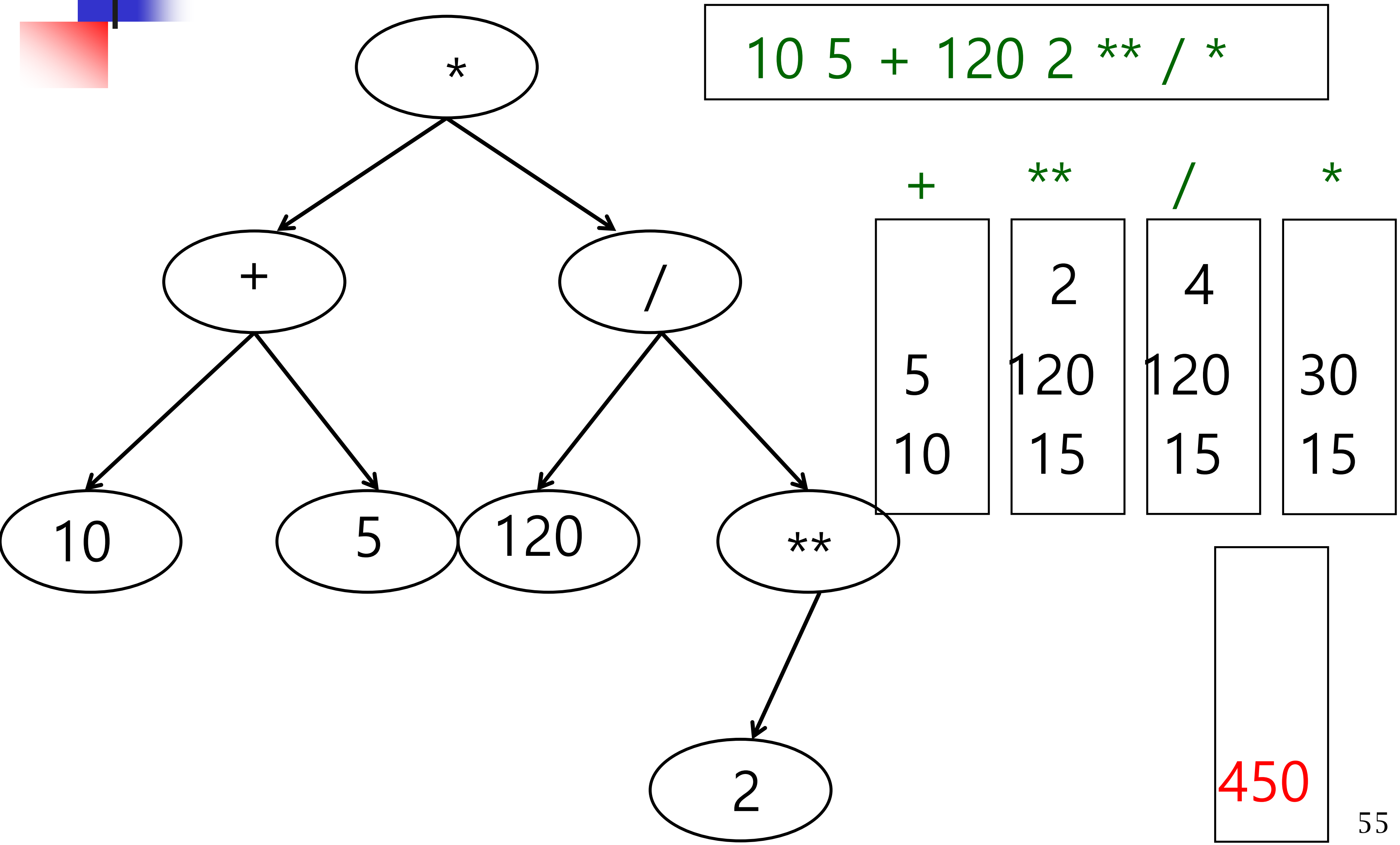
# (4/5)

10 5 + 120 2 ** / *





# (5/5)

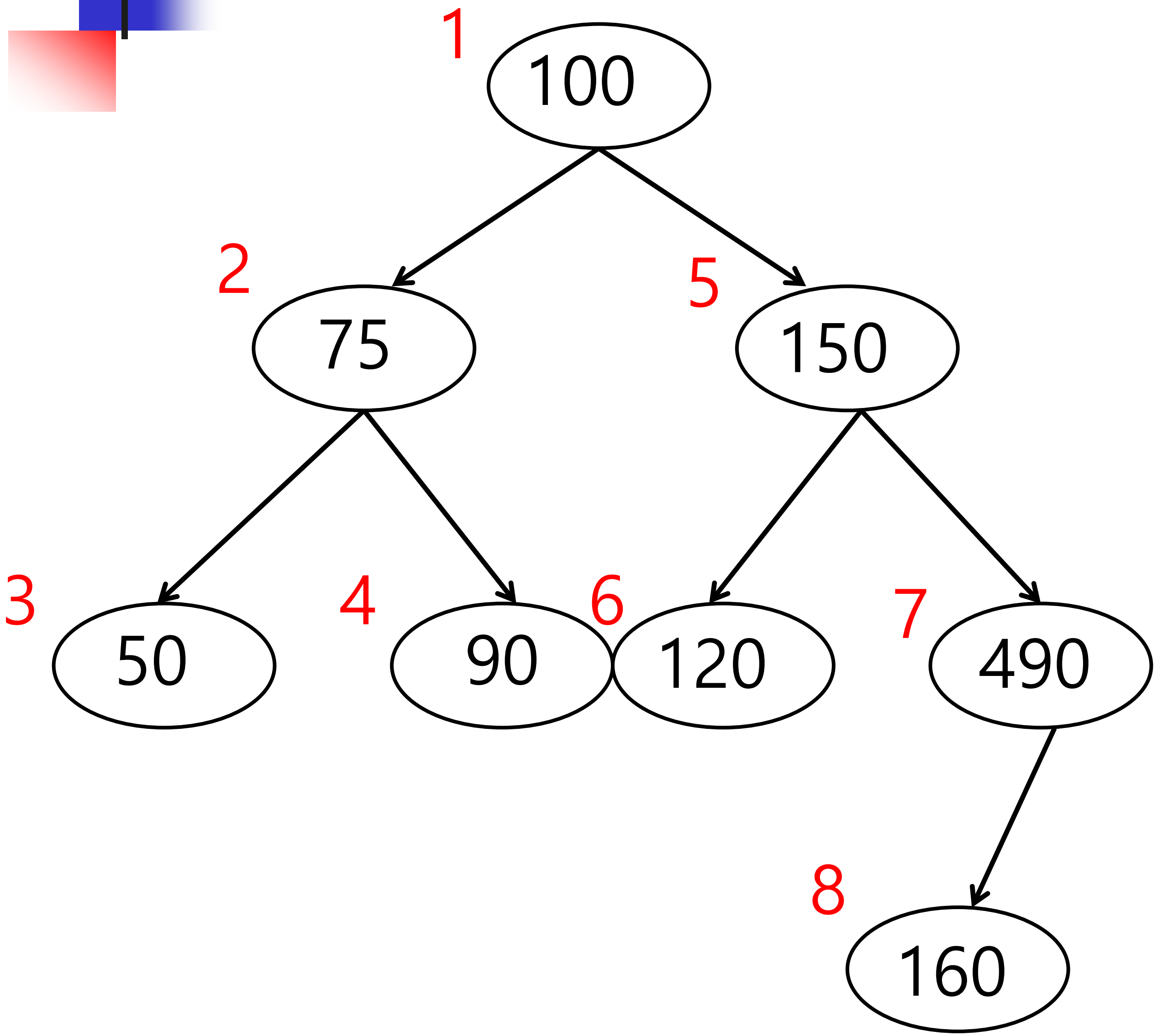


10 5 + 120 2 ** / *

# Preorder Traversal

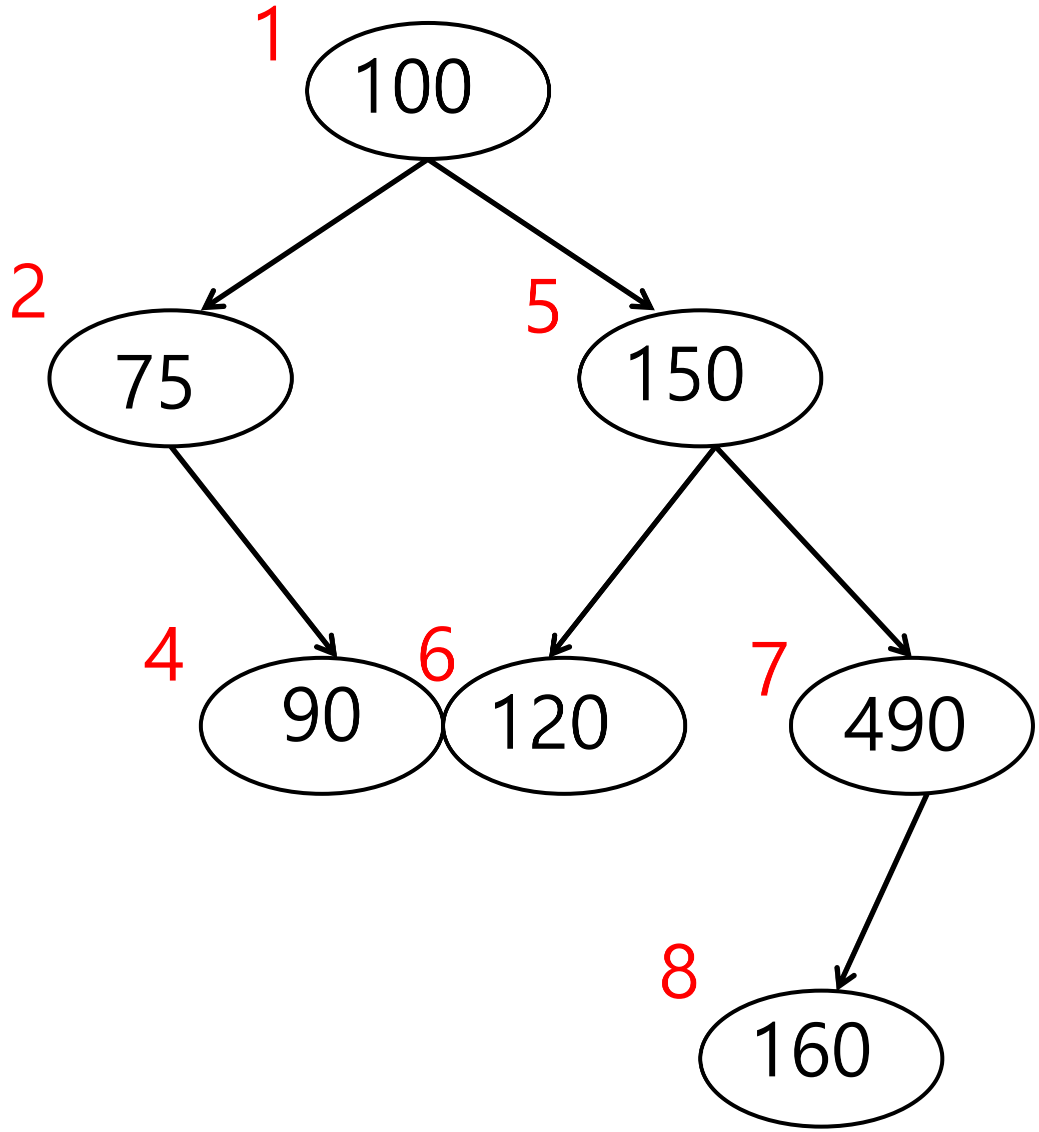
- Mother, First Child, Second Child
- Applications
  - Make a complete copy of a tree
  - (compiler) prefix expression evaluation
    - In stack-like (reverse) order, using a stack

# Preorder Traversal: Example (1/6)



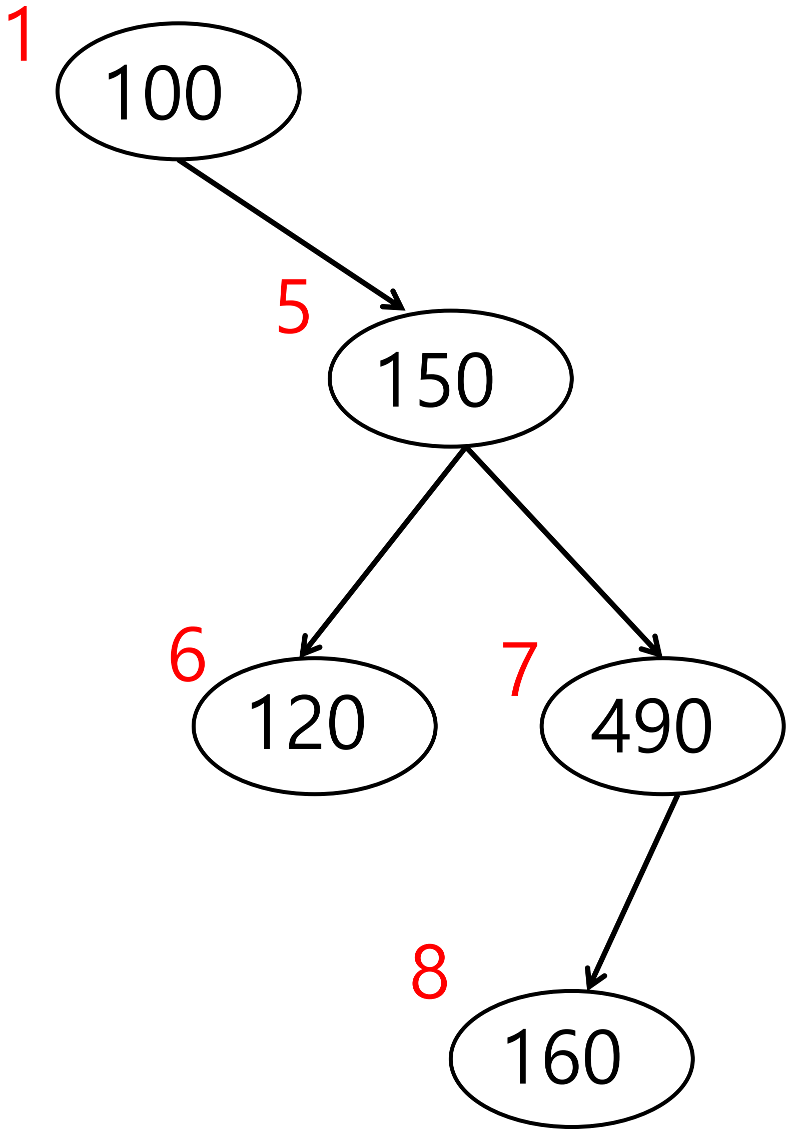
100
75
50

# Preorder Traversal: Example (2/6)



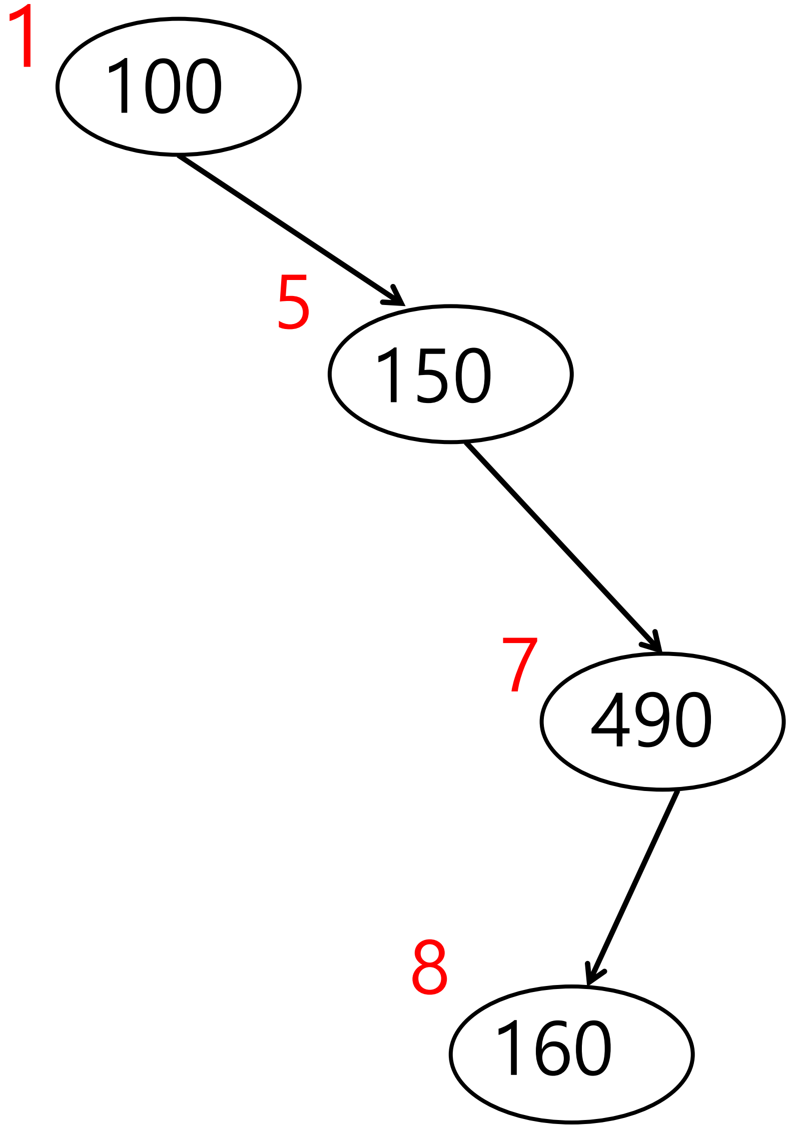
100
75
50
90

# Preorder Traversal: Example (3/6)



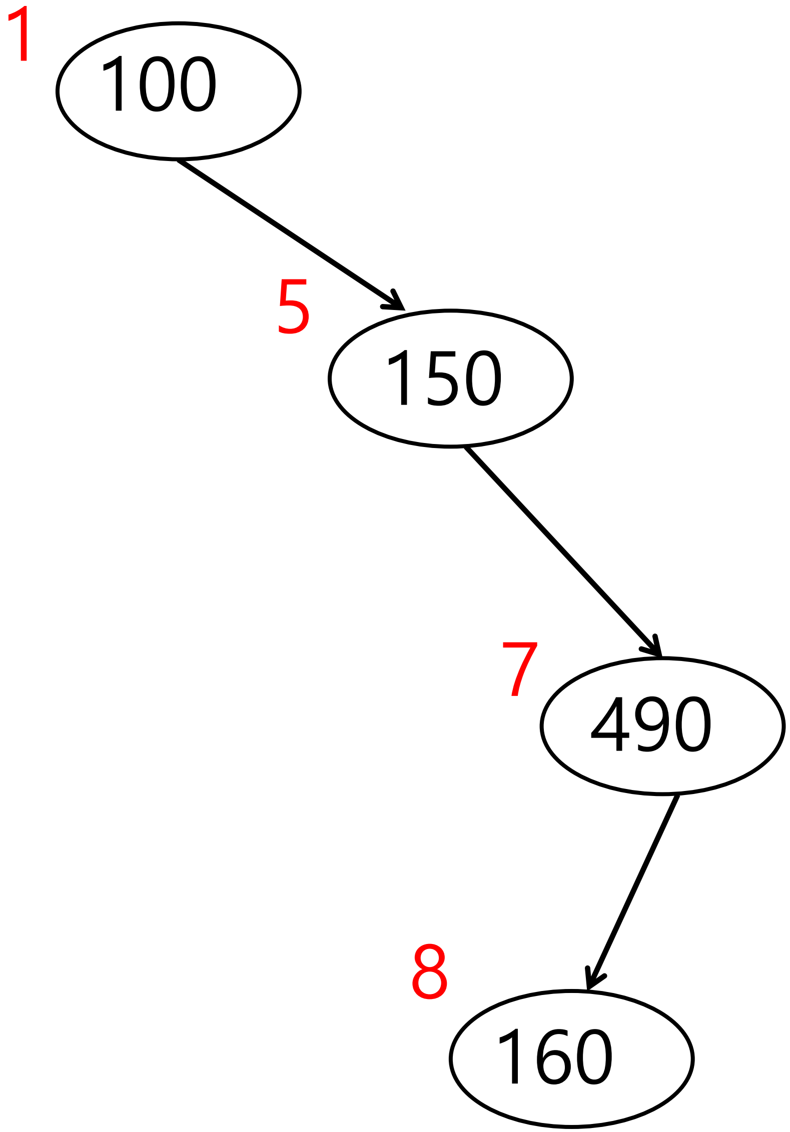
100
75
50
90
150

# Preorder Traversal: Example (4/6)



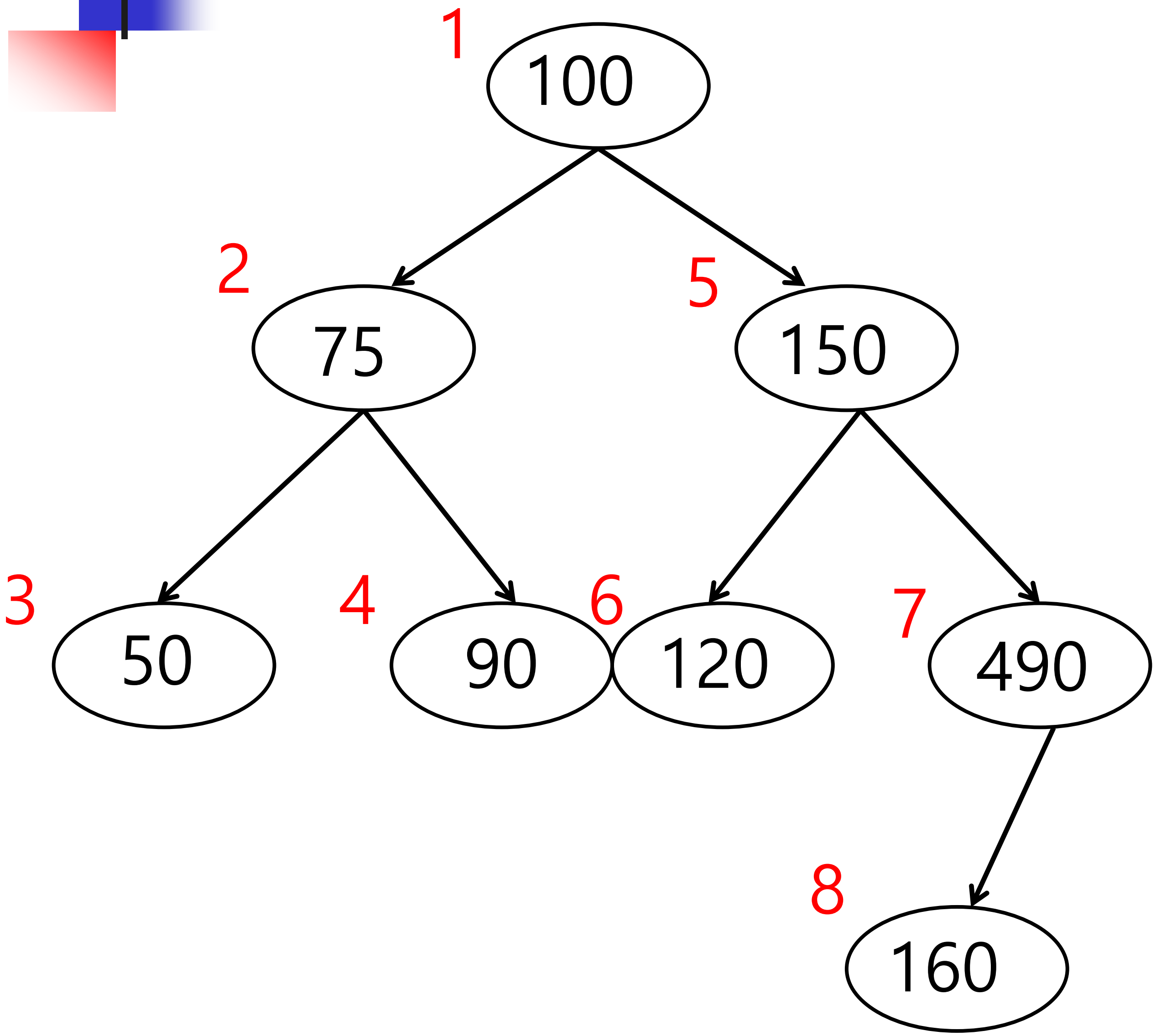
100
75
50
90
150
120

# Preorder Traversal: Example (5/6)



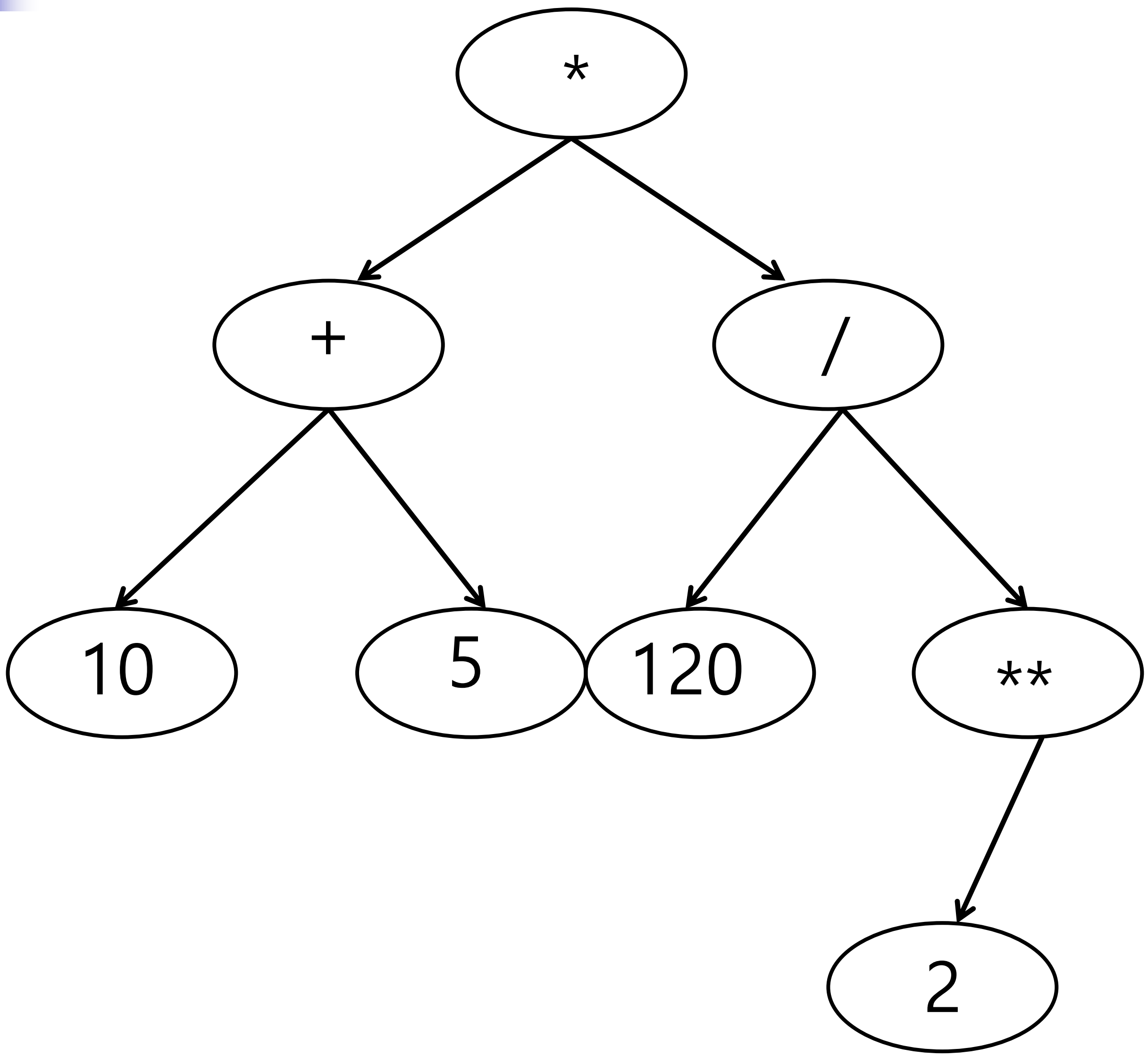
100
75
50
90
150
120
490
160

# Preorder Traversal: Example (6/6)



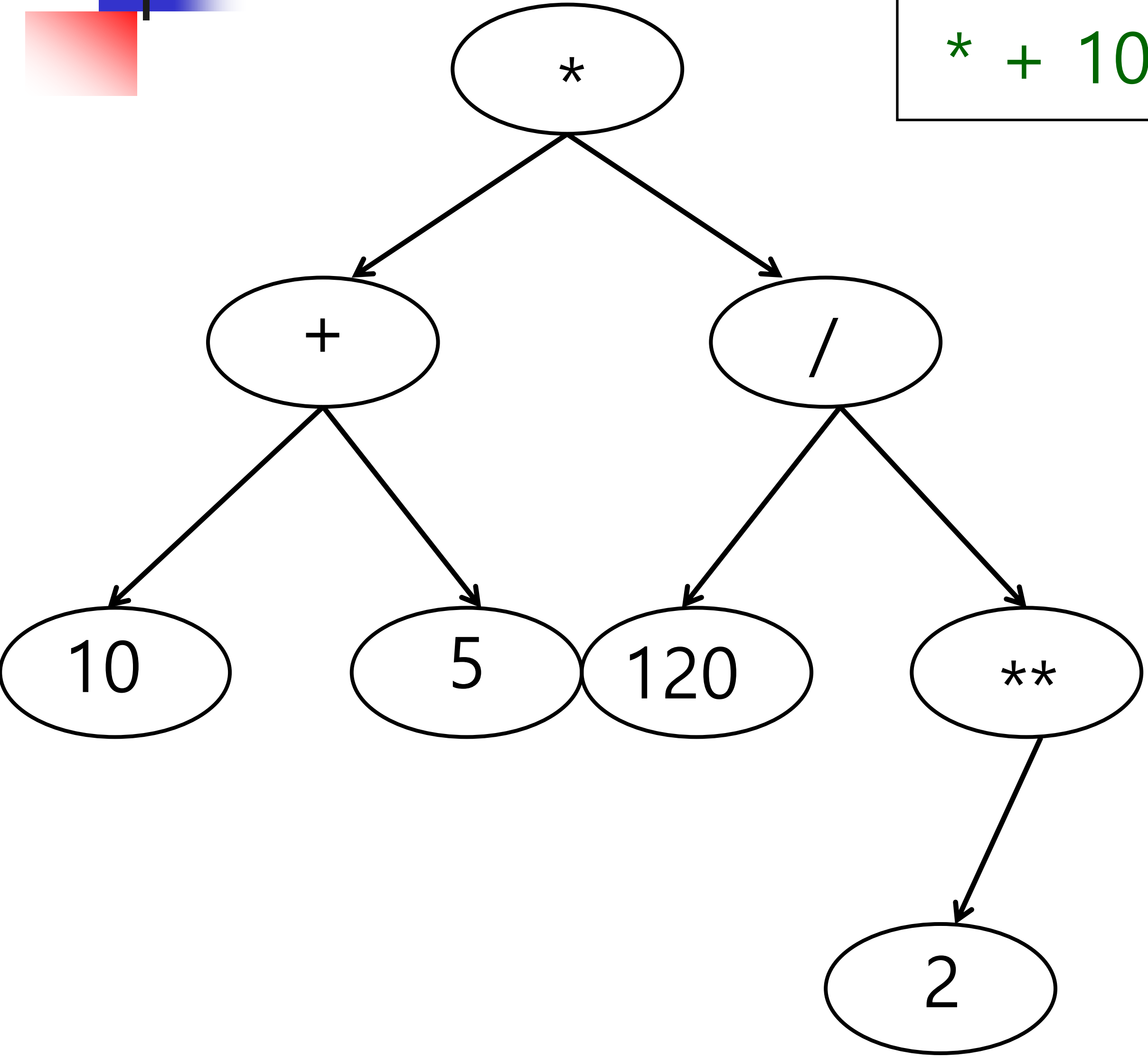
100
75
50
90
150
120
490
160

# Preorder Traversal: Exercise

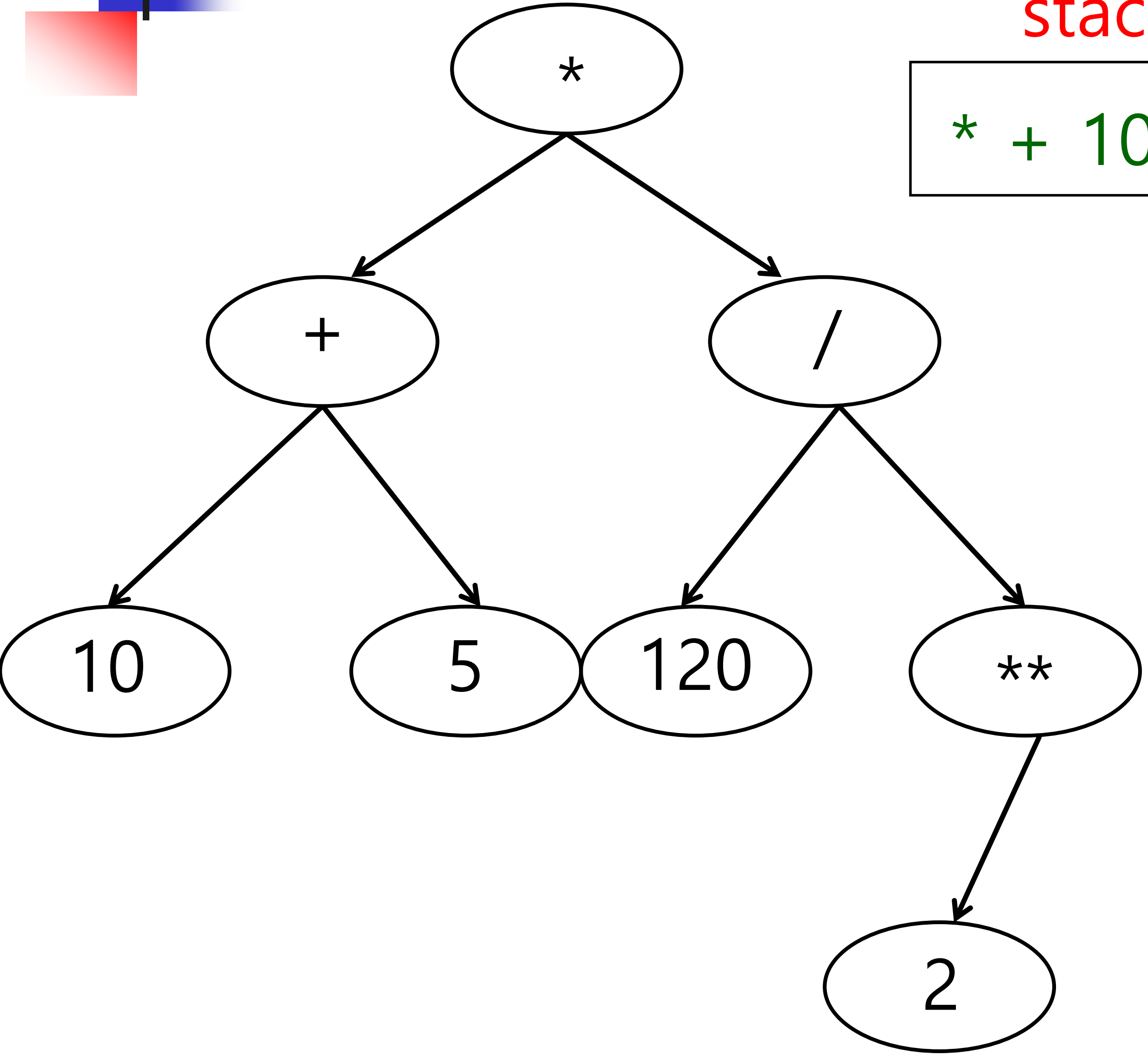


# Preorder Traversal: Solution

* + 10 5 / 120 ** 2

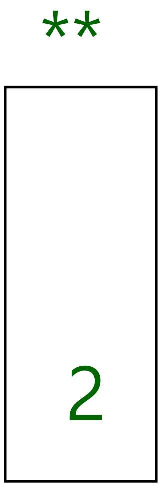


# Evaluating a Prefix Expression Using a Stack (1/5)



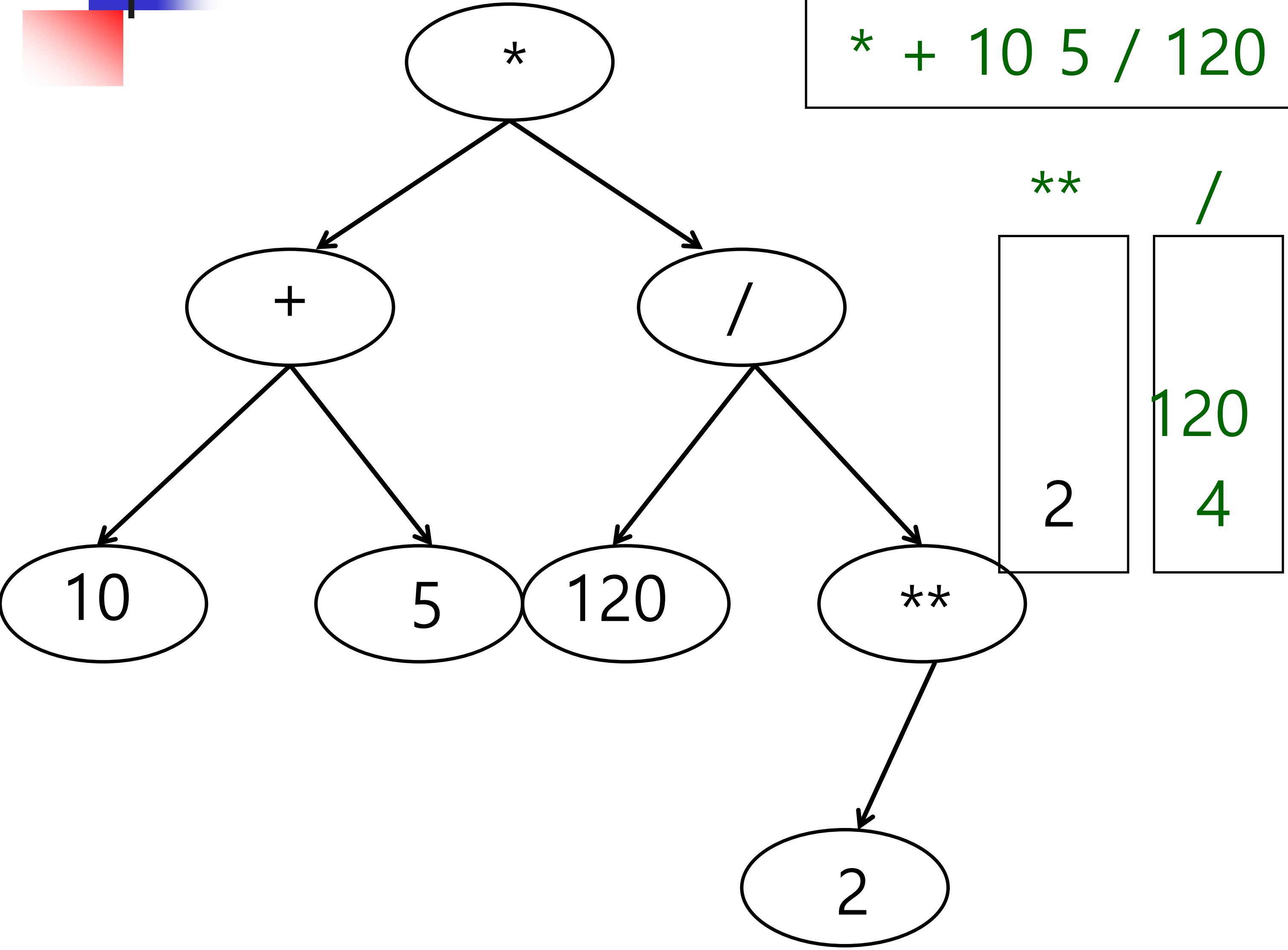
stack-1

* + 10 5 / 120 ** 2



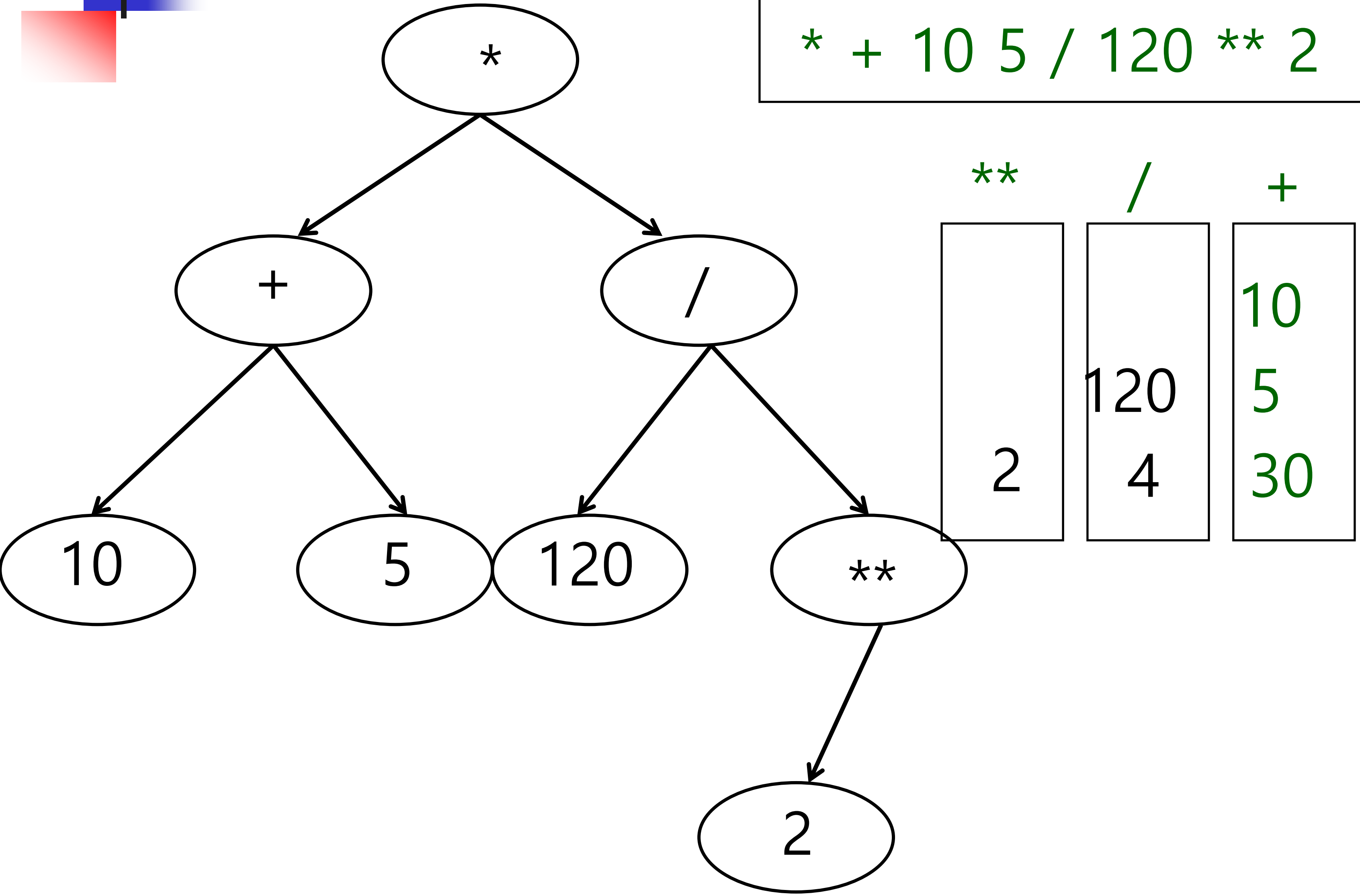
# (2/5)

* + 10 5 / 120 ** 2

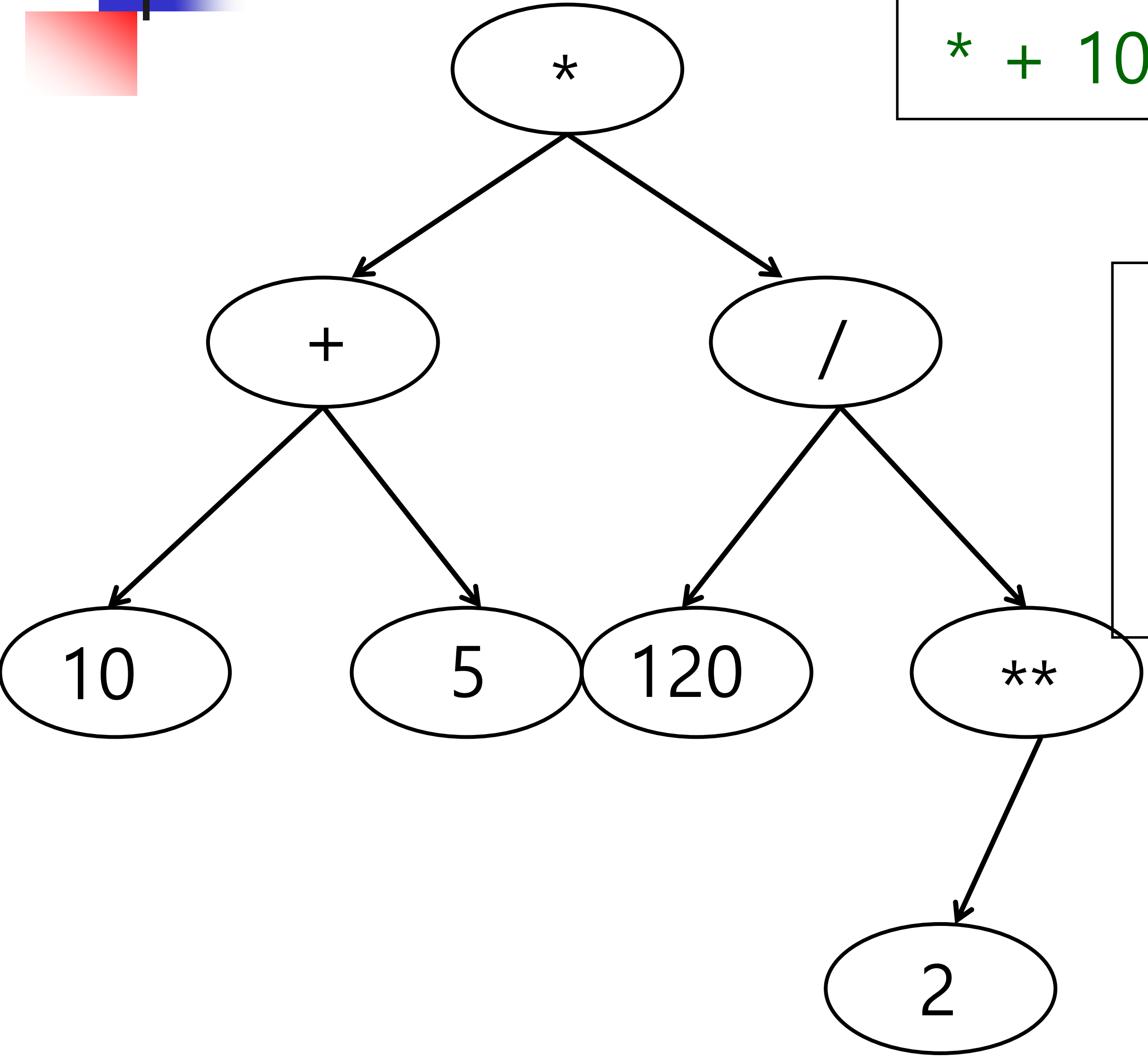


# (3/5)

* + 10 5 / 120 ** 2



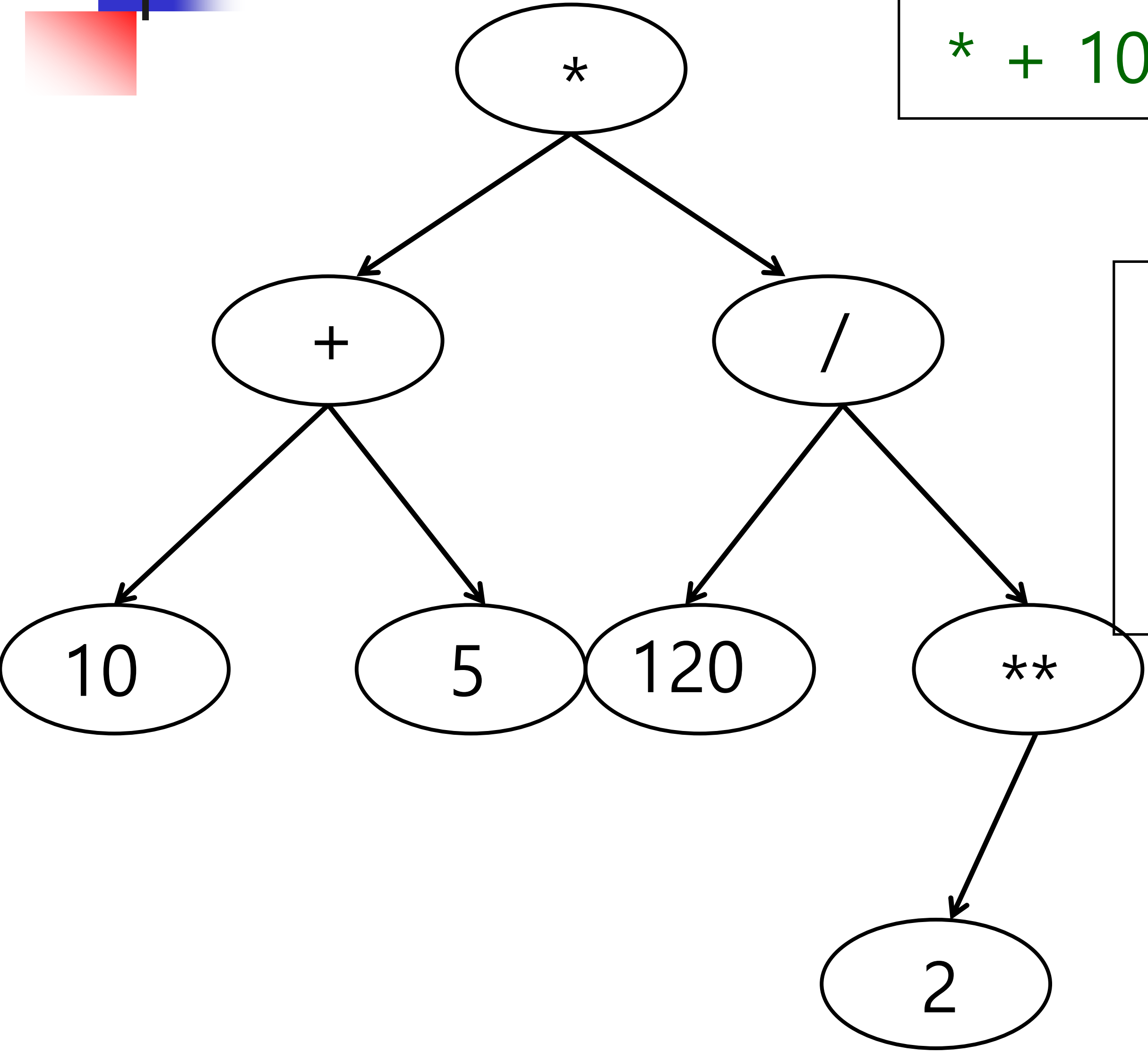
# (4/5)



* + 10 5 / 120 ** 2

| ** | / | + | * |
|---|---|---|---|
| | | 10 | |
| | 120 | 5 | 15 |
| 2 | 4 | 30 | 30 |

# (5/5)



* + 10 5 / 120 ** 2

| ** | / | + | * |
|---|---|---|---|
| | | 10 | |
| | 120 | 5 | 15 |
| 2 | 4 | 30 | 30 |

| |
|---|
| 450 |

# Level Order Traversal

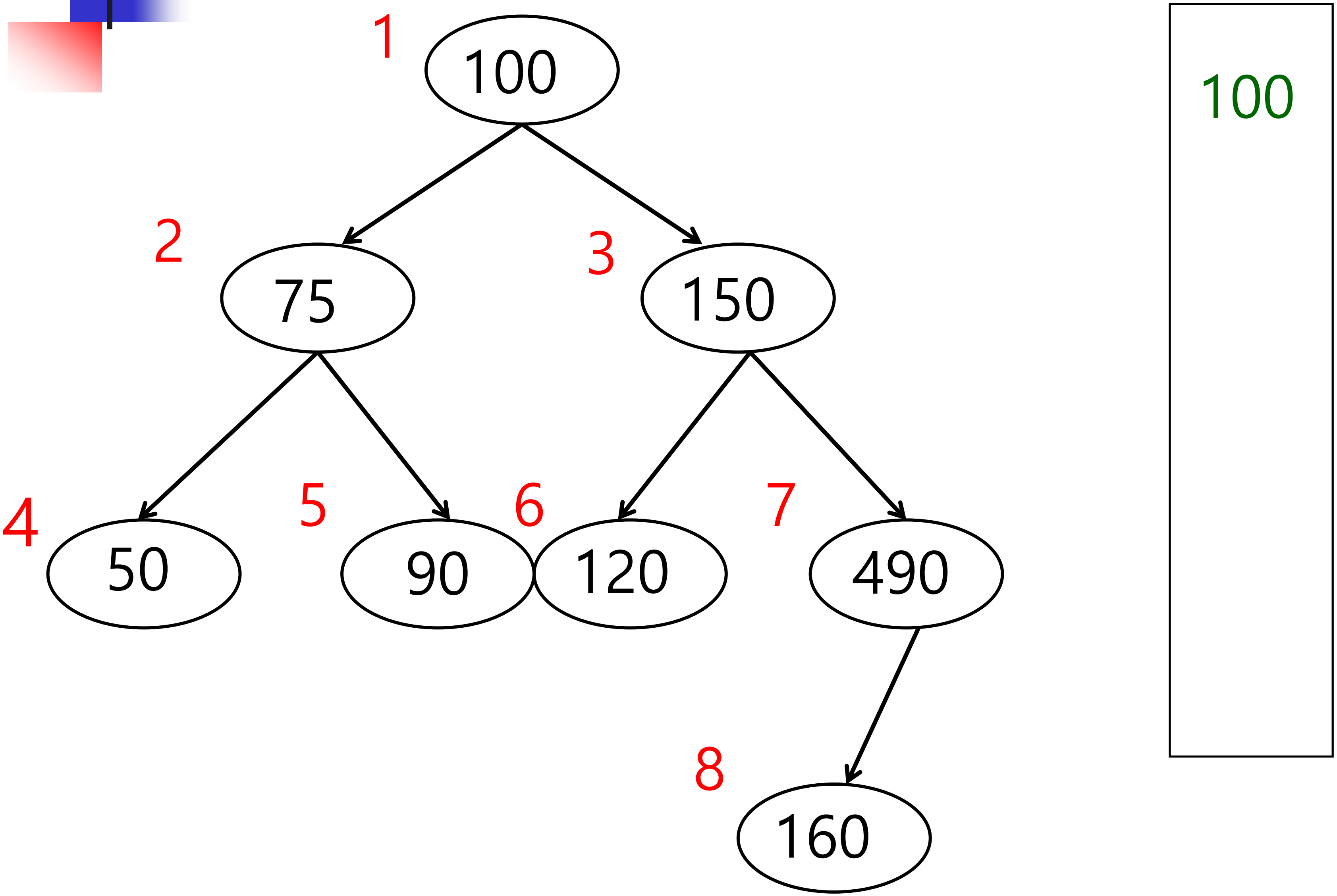
- Range (Level) Retrieval
  - People of the same rank on an organizational chart
  - Groups of the same rank on an organization chart
  - Possible next moves in a chess/go game

# Playing Chess

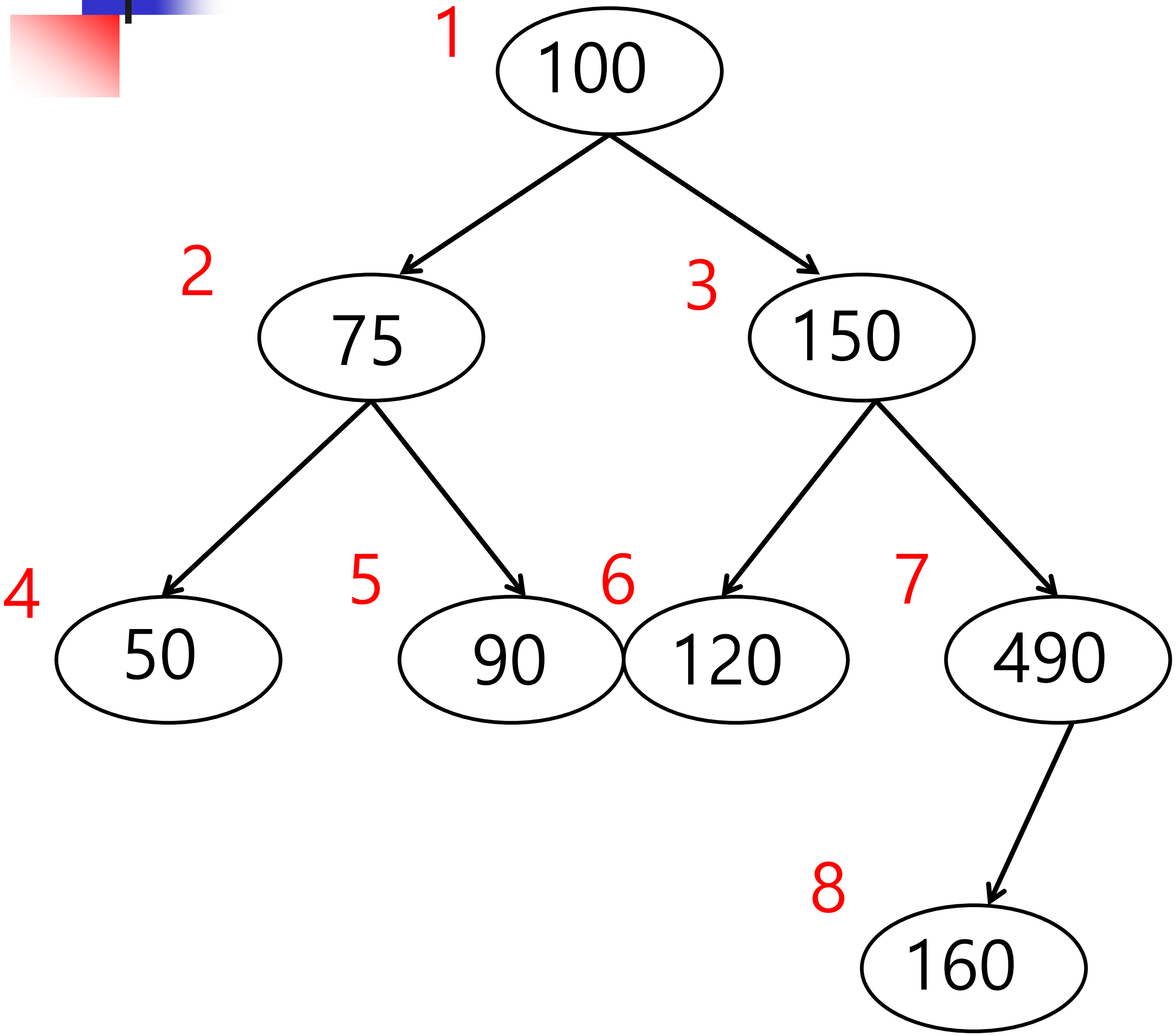
# IBM Deep Blue – Chess Playing Computer

# Playing Go

# Level Order Traversal: Example (1/4)

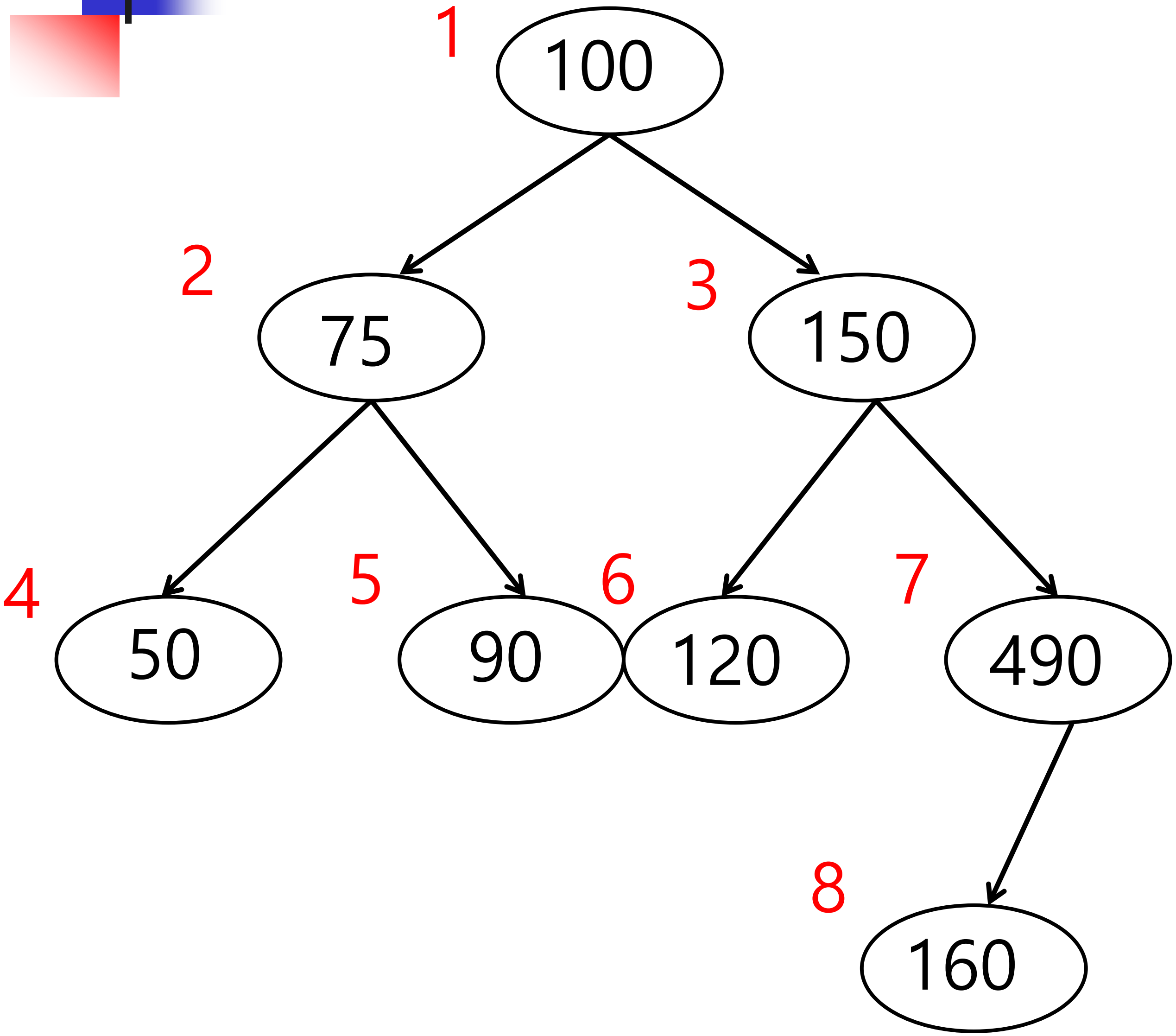


# Level Order Traversal: Example (2/4)



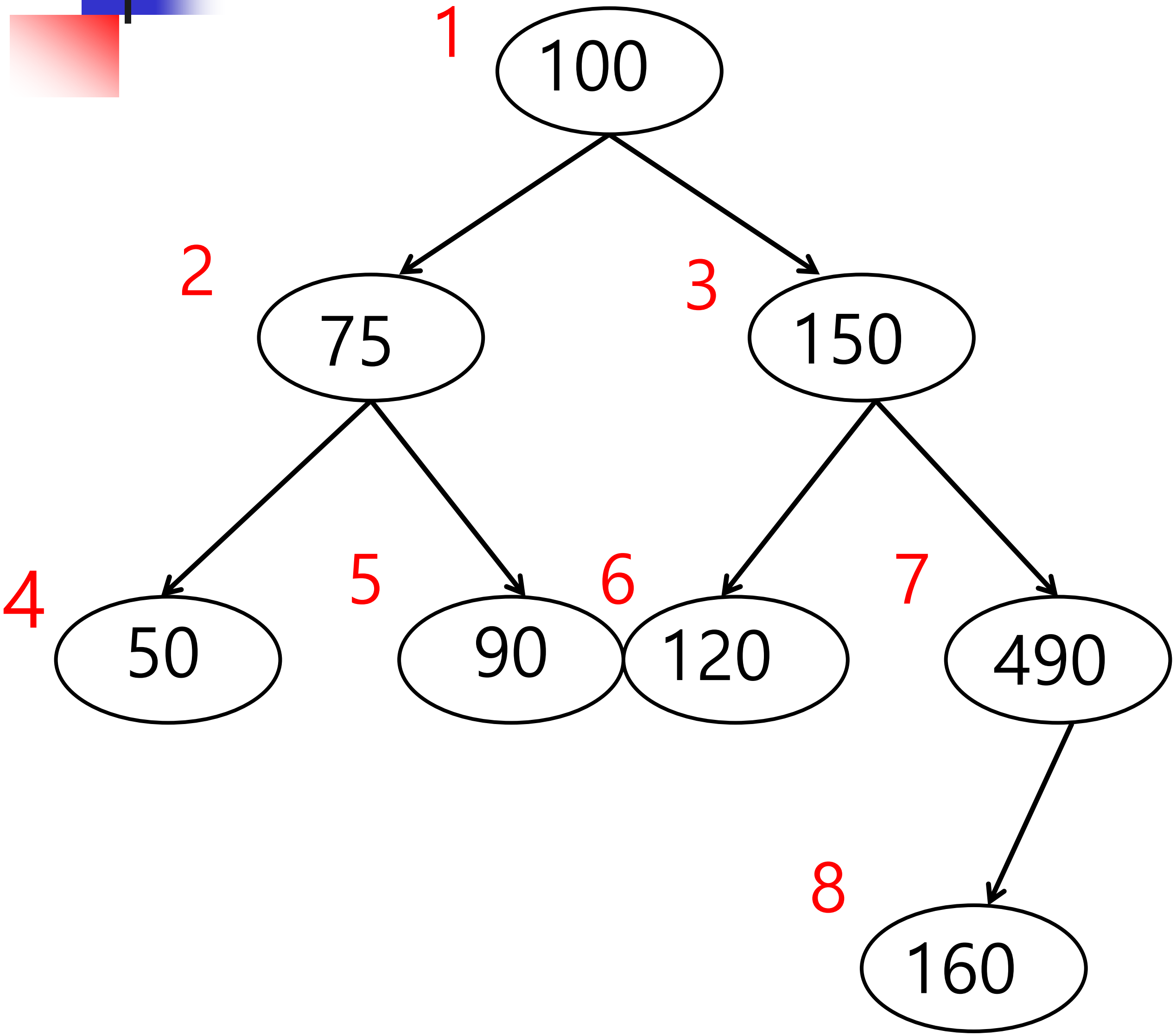
100
75
150

# Level Order Traversal: Example (3/4)



100
75
150
50
90
120
490

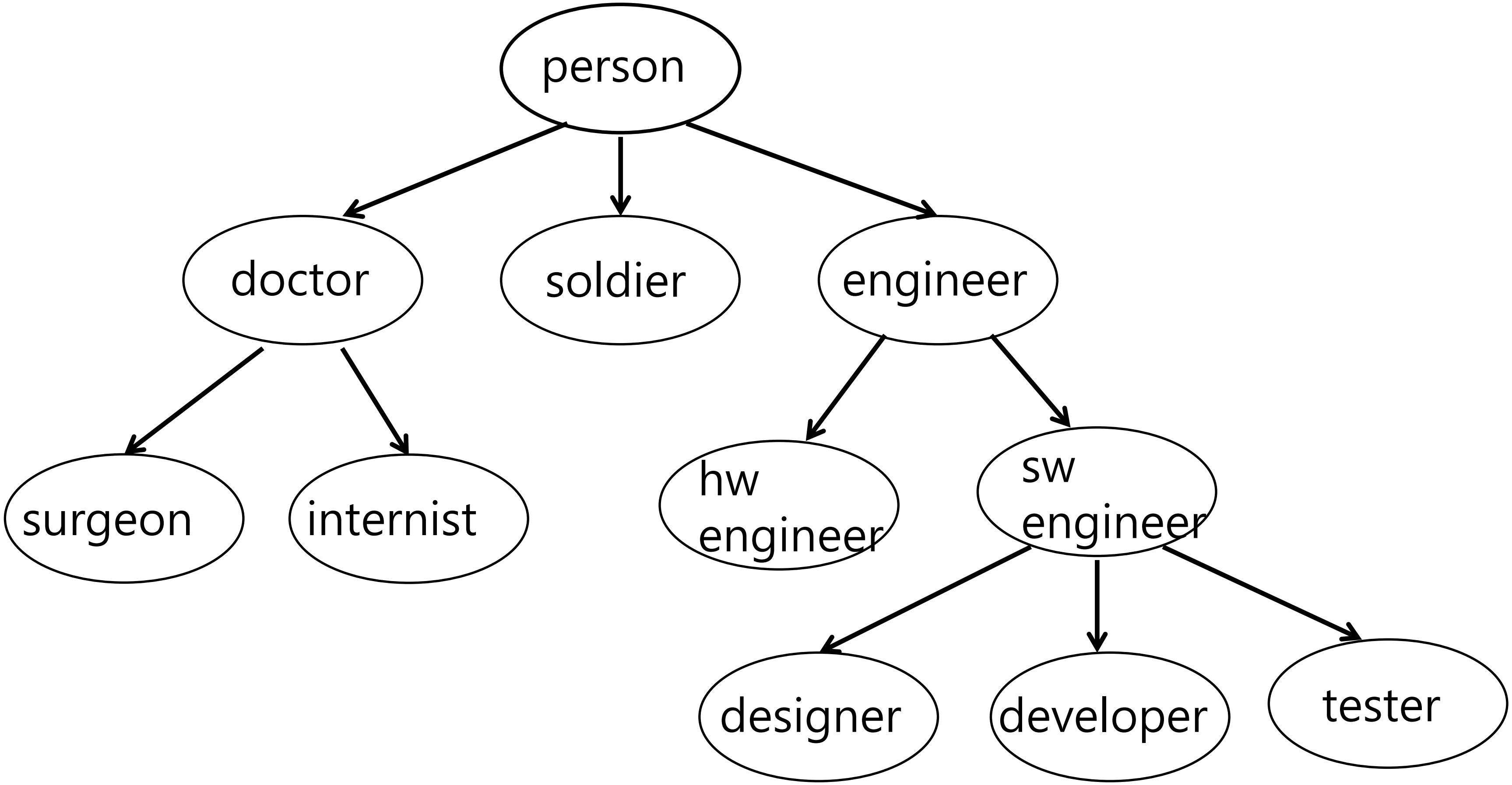
# Level Order Traversal: Example (4/4)



100
75
150
50
90
120
490
160

# Assignment 3

# HW3: What Are the Results of #1 Inorder, #2 Postorder, #3 Preorder, #4 level order Traversal?



# End of Lecture