Recursion on Trees

Structure of Trees

- Two views of a tree:
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***child nodes*** of the root, each of which is the root of its own tree
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***subtrees*** of the root, each of which is another tree

Structure of Trees

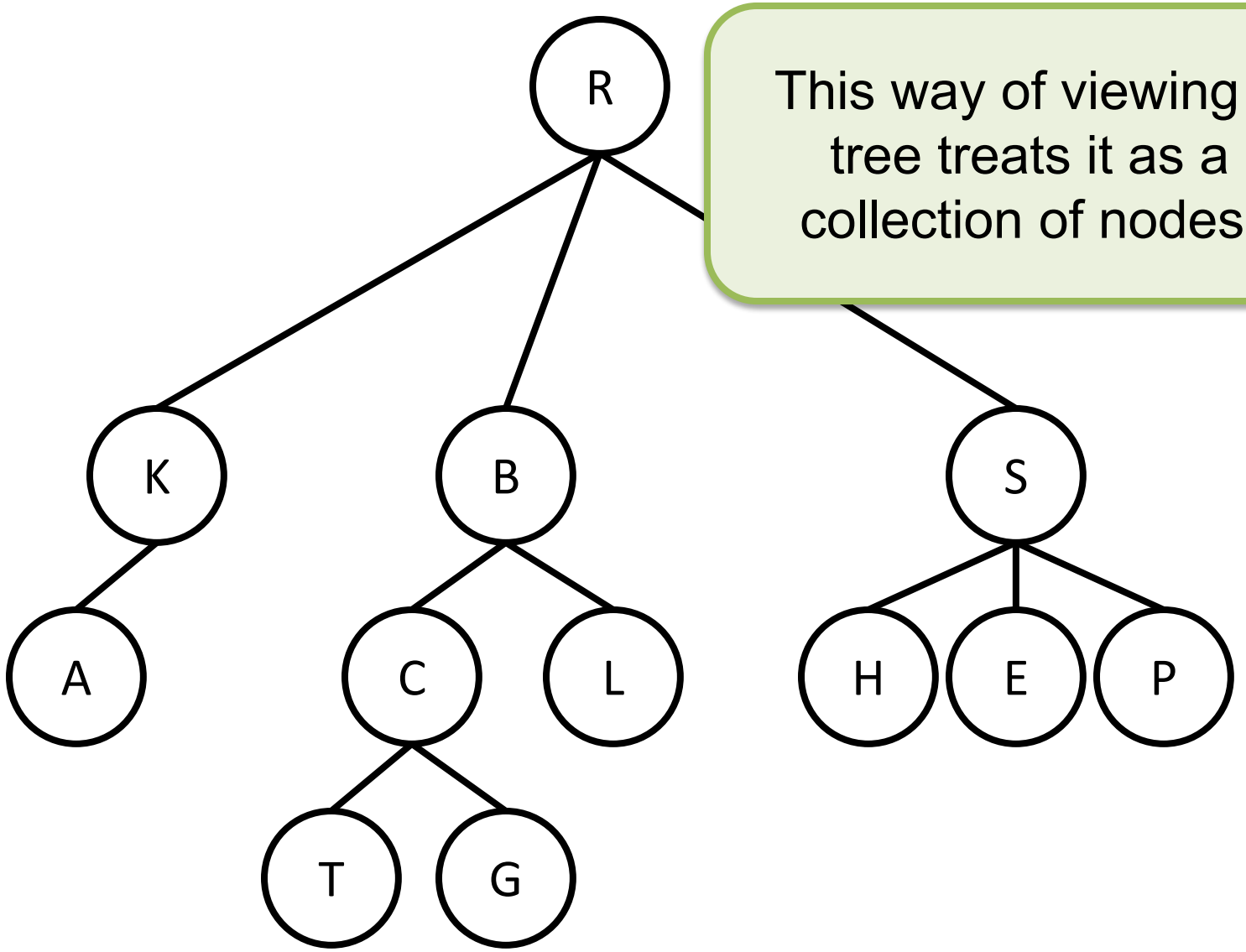
- Two views of a tree:
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***child nodes*** of the root, each of which is the root of its own tree
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***subtrees*** of the root, each of which is another tree

This way of viewing a tree treats it as a collection of nodes.

Structure of Trees

- Two views of a tree:
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***children nodes*** of the root, each of which is the root of its own tree
  - A ***tree*** is made up of:
    - A ***root node***
    - A string of zero or more ***subtrees*** of the root, each of which is another tree

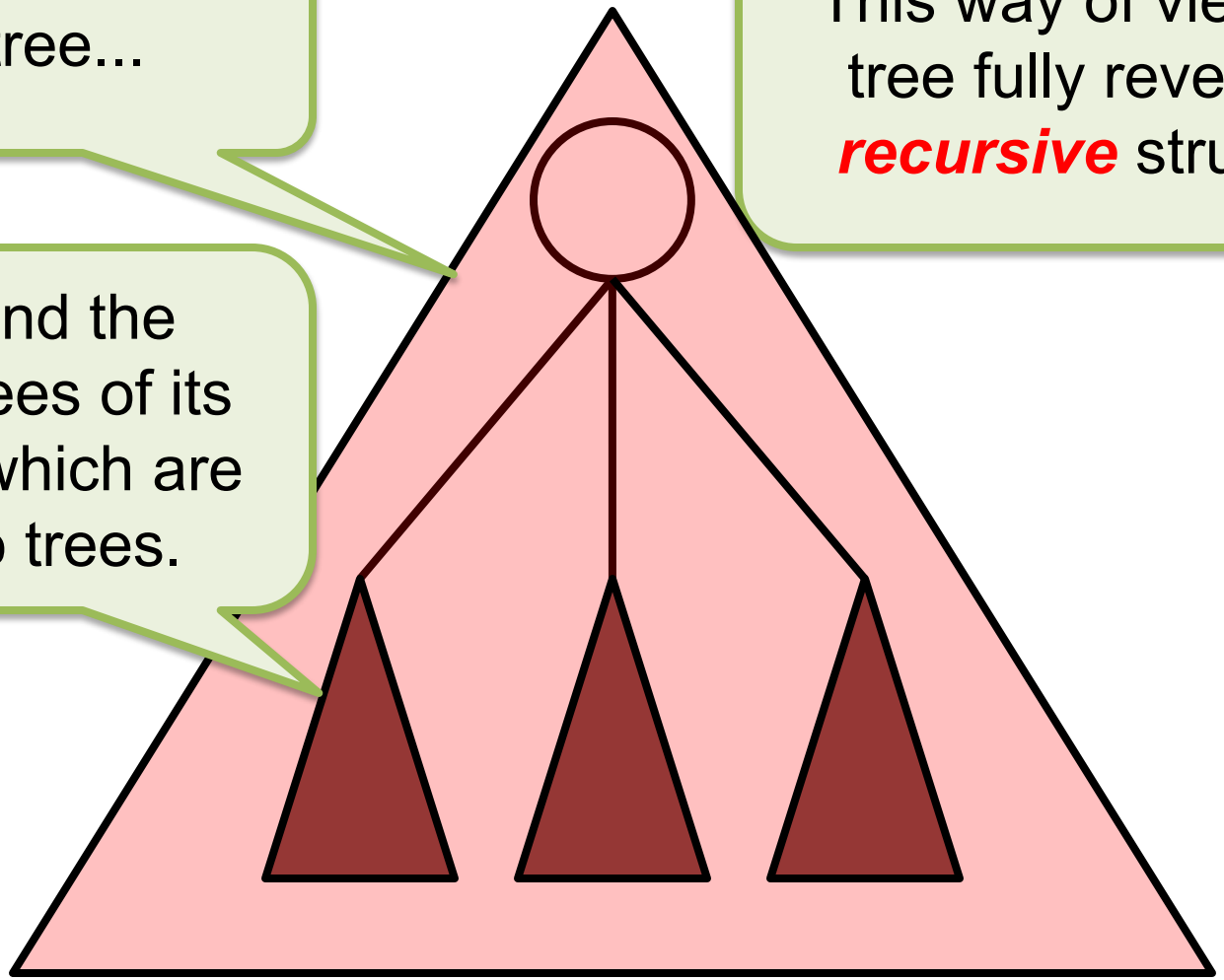
This way of viewing a tree fully reveals its ***recursive*** structure.



This way of viewing a tree treats it as a collection of nodes.

A tree...

... and the subtrees of its root, which are also trees.



This way of viewing a tree fully reveals its ***recursive*** structure.

Recursive Algorithms

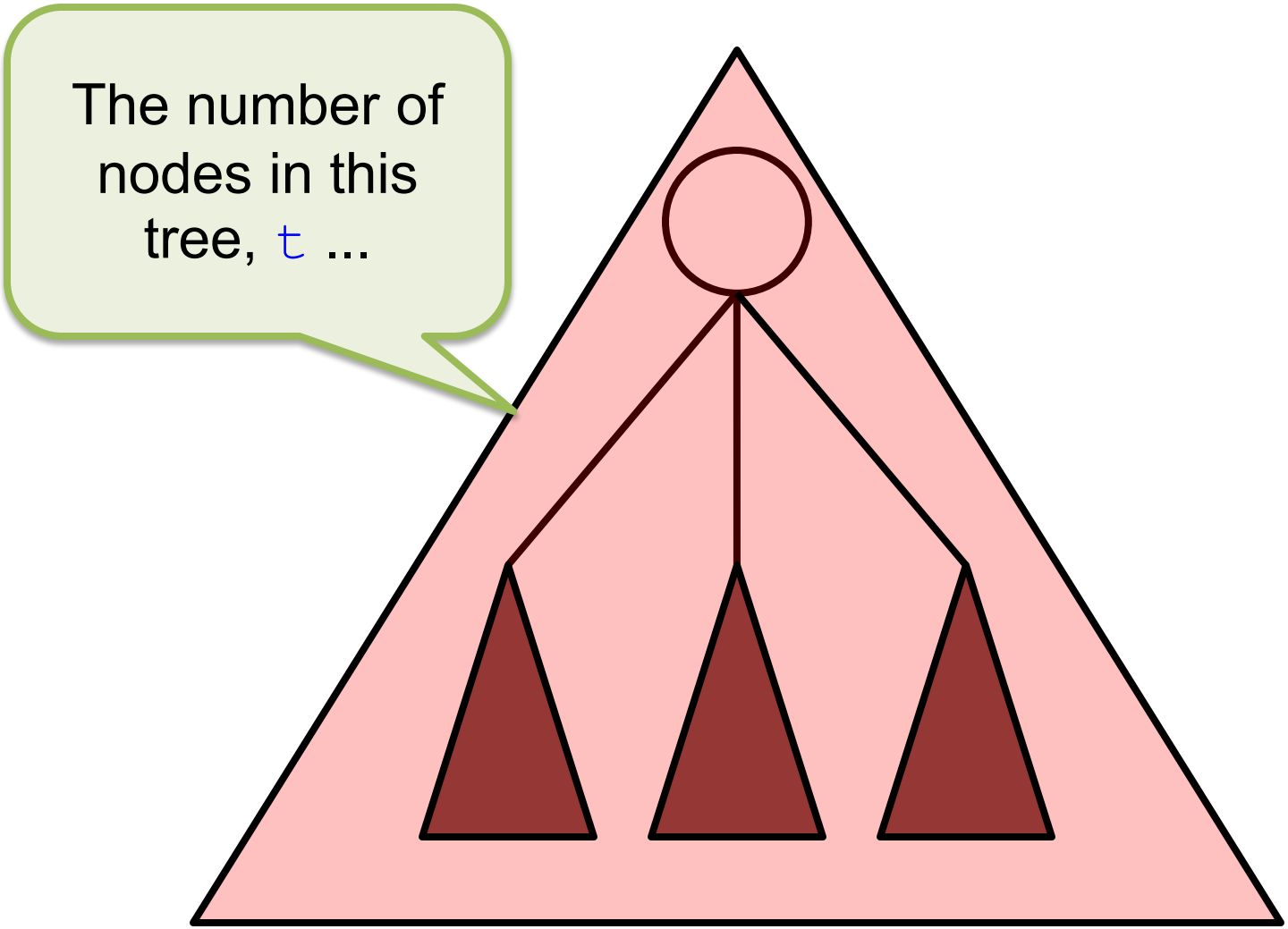
- The “in-your-face” recursive structure of trees (in the second way to view them) allows you to implement some methods that operate on trees using recursion
  - Indeed, this is sometimes the only sensible way to implement those methods

XMLTree

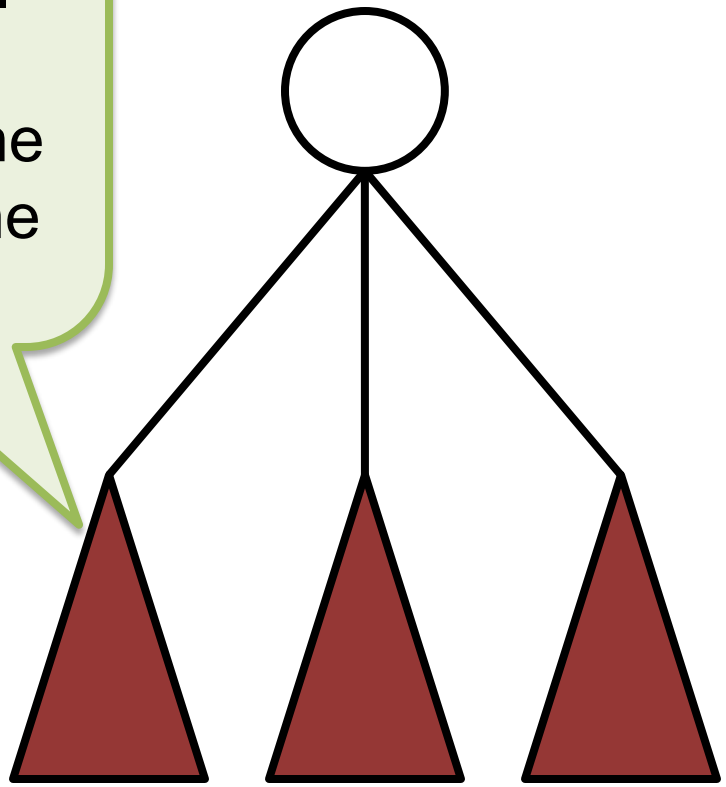
- The methods for XMLTree are named using the collection-of-nodes view of a tree, because most uses of XMLTree (e.g., the XML/RSS projects) do not need to leverage the recursive structure of trees
- But some uses of XMLTree demand that you use the recursive view...

Example

```
/**
 * Reports the size of an XMLTree.
 * ...
 * @ensures
 * size = [number of nodes in t]
*/
private static int size(XMLTree t) {...}
```


The number of nodes in this tree, t ...

... is 1 (the root) plus the total number of nodes in all the subtrees of the root of `t`.



Example

```
private static int size(XMLTree t) {
  int totalNodes = 1;
  if (t.isTag()) {
    for (int i = 0; i < t.numberOfChildren();
         i++) {
      totalNodes += size(t.child(i));
    }
  }
  return totalNodes;
}
```

Example

```
private static int size(XMLTree t) {
  int totalNodes = 1;
  if (t.isTag()) {
    for (int i = 0; i < t.numberOfChildren();
         i++) {
      totalNodes += size(t.child(i));
    }
  }
  return totalNodes;
}
```

This recursive call reports the size of a subtree of the root.

Example

```
/**
 * Reports the height of an XMLTree.
 * ...
 * @ensures
 * height = [height of t]
*/
private static int height(XMLTree t) {...}
```

Example

```
private static int height(XMLTree t) {
  int maxSubtreeHeight = 0;
  if (t.isTag()) {
    for (int i = 0; i < t.numberOfChildren();
         i++) {
      int subtreeHeight = height(t.child(i));
      if (subtreeHeight > maxSubtreeHeight) {
        maxSubtreeHeight = subtreeHeight;
      }
    }
  }
  return maxSubtreeHeight + 1;
}
```

Example

```
private static int height(XMLT
  int maxSubtreeHeight = 0;
  if (t.isTag()) {
    for (int i = 0; i < t.numbe
         i++) {
      int subtreeHeight = height(t.child(i));
      if (subtreeHeight > maxSubtreeHeight) {
        maxSubtreeHeight = subtreeHeight;
      }
    }
  }
  return maxSubtreeHeight + 1;
}
```

This recursive call reports the height of a subtree of the root.

Example

```
private static int height(XMLT
  int maxSubtreeHeight = 0;
  if (t.isTag()) {
    for (int i = 0; i <  .numbe
         i++) {
      int subtreeHeight = height(t.child(i));
      if (subtreeHeight > maxSubtreeHeight) {
        maxSubtreeHeight = subtreeHeight;
      }
    }
  }
  return maxSubtreeHeight + 1;
}
```

Why is it a good idea to store the result of the recursive call in a variable here?

Expression Trees

- There are many other uses for `XMLTree`
- Consider an ***expression tree***, which is a representation of a formula you might type into a Java program or into a calculator, such as:

```
(1 + 3) * 5 - (4 / 2)
```

Expression Trees

- There are many o
- Consider an ***expr*** representation of a into a Java program into a calculator, such as:

```
(1 + 3) * 5 - (4 / 2)
```

What is the ***value*** of this expression? Computing this value is what we mean by ***evaluating*** the expression.

Order of Evaluation

- What is the ***order of evaluation*** of subexpressions in this expression?

```
(1 + 3) * 5 - (4 / 2)
```

Order of Evaluation

- What is the ***order of evaluation*** of subexpressions in this expression?

```
(1 + 3) * 5 - (4 / 2)
```

- Let's ***fully parenthesize*** it to help:

```
((1 + 3) * 5) - (4 / 2)
```

Order of Evaluation

- What is the ***order of evaluation*** of subexpressions in this expression?

```
(1 + 3) * 5 - (4 / 2)
```

- Let's ***fully parenthesize*** it to help:

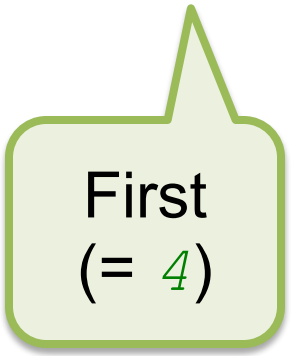
```
((1 + 3) * 5) - (4 / 2)
```

The fully parenthesized version is based on a convention regarding the ***precedence*** of operators (e.g., " * before – " in ordinary math).

Order of Evaluation

- What is the ***order*** in which the subexpressions in this expression are evaluated?

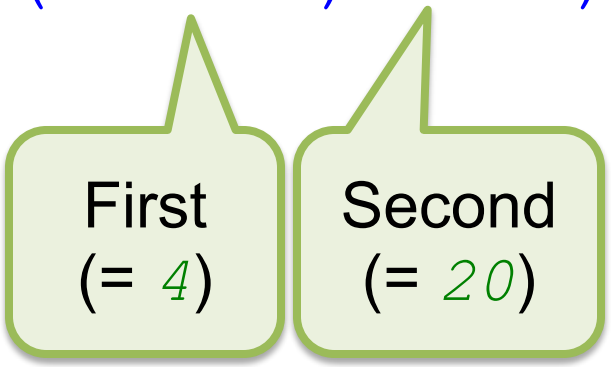
```
((1 + 3) * 5) - (4 / 2)
```



Order of Evaluation

- What is the ***order*** in which the subexpressions in this expression are evaluated?

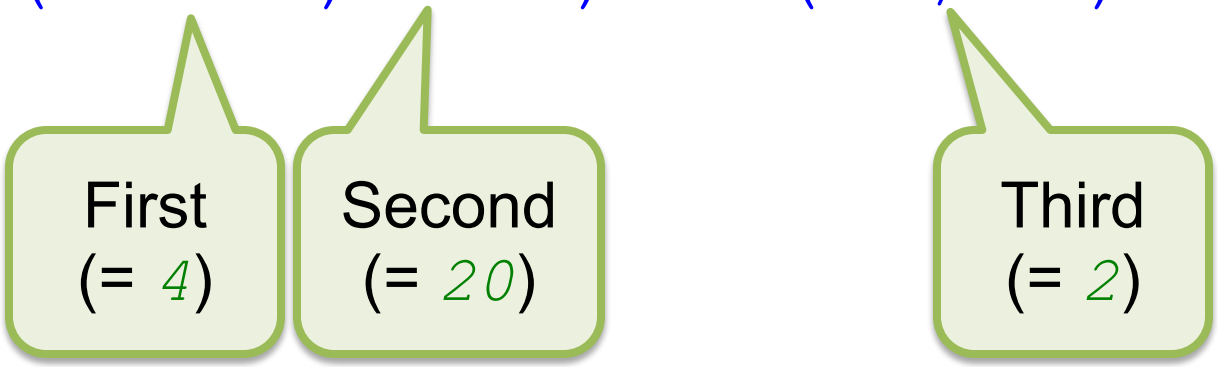
```
((1 + 3) * 5) - (4 / 2)
```



Order of Evaluation

- What is the ***order*** in which the subexpressions in this expression are evaluated?

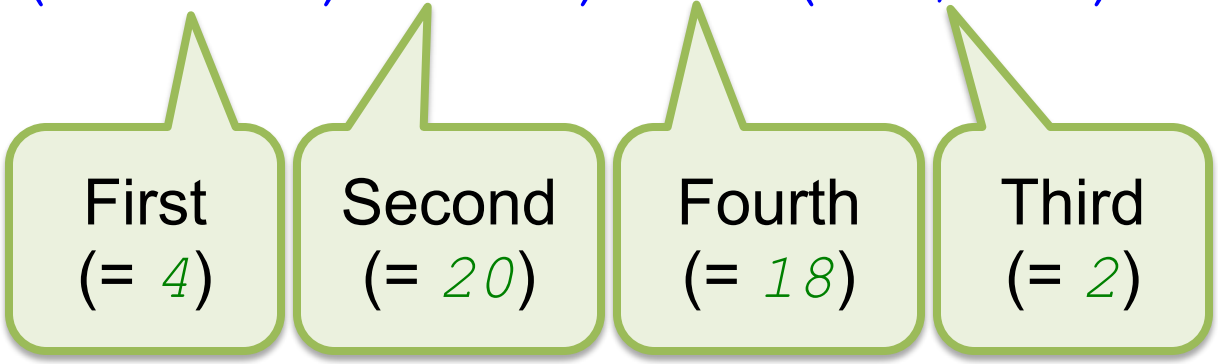
```
((1 + 3) * 5) - (4 / 2)
```



Order of Evaluation

- What is the ***order*** in which the subexpressions in this expression are evaluated?

```
((1 + 3) * 5) - (4 / 2)
```



- What i
  subexp
  evaluated?

"Inner-most" parentheses first, but there may be some flexibility in the order of evaluation (e.g., / before + would work just as well, but not * before +, in this expression).

```
((1 + 3) * 5) - (4 / 2)
```

First (= *4*)

Second (= *20*)

Fourth (= *18*)

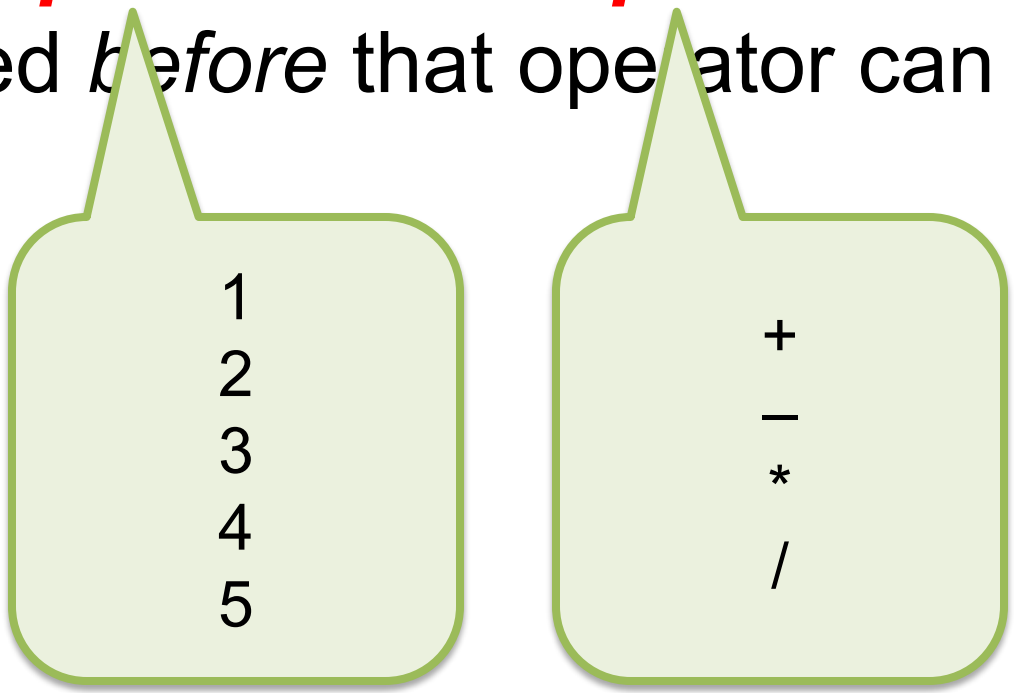
Third (= *2*)

Tree Representation of Expression

- Key: Each ***operand*** of an ***operator*** must be evaluated *before* that operator can be evaluated

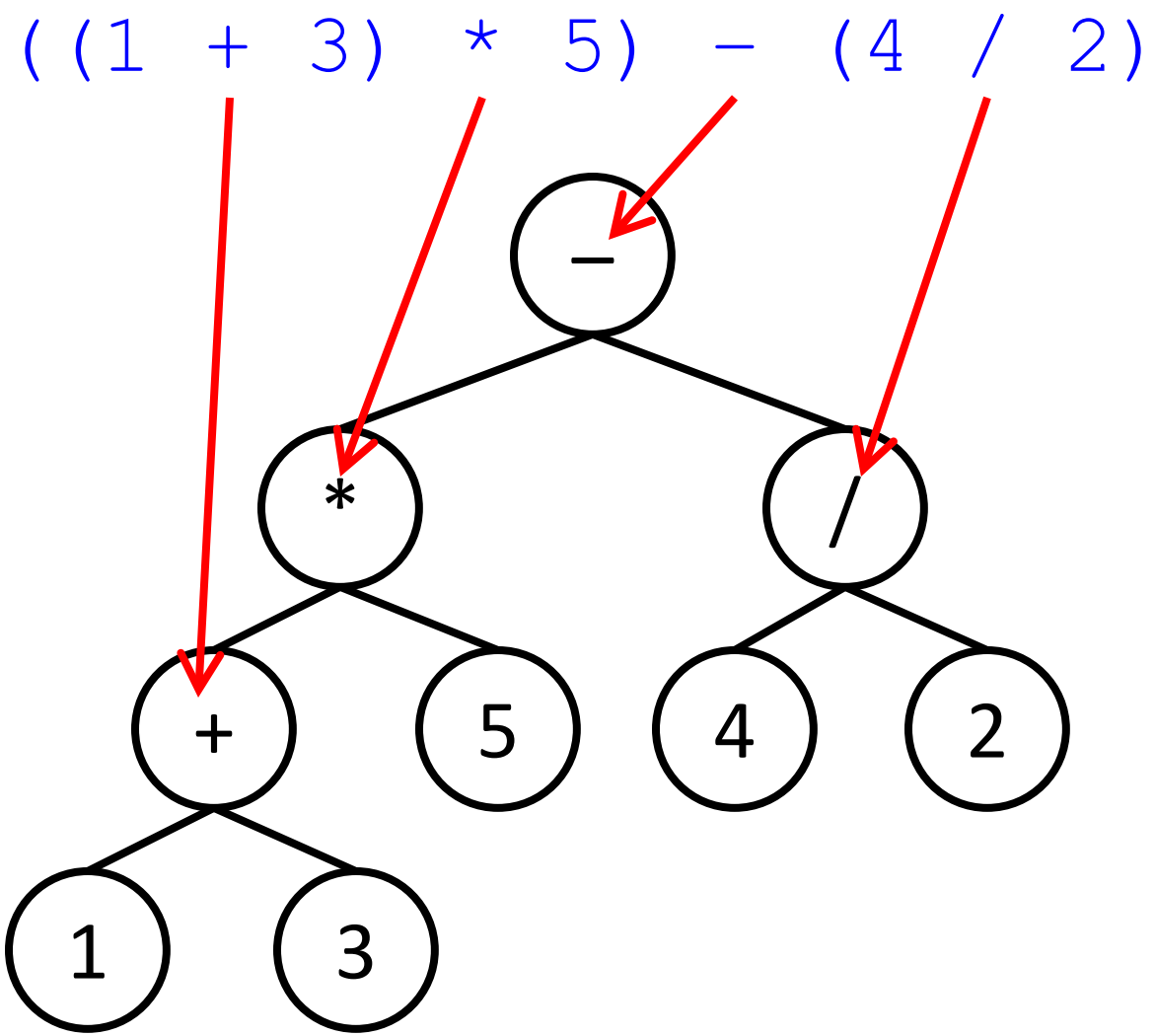
Tree Representation of Expression

- Key: Each ***operand*** of an ***operator*** must be evaluated *before* that operator can be evaluated



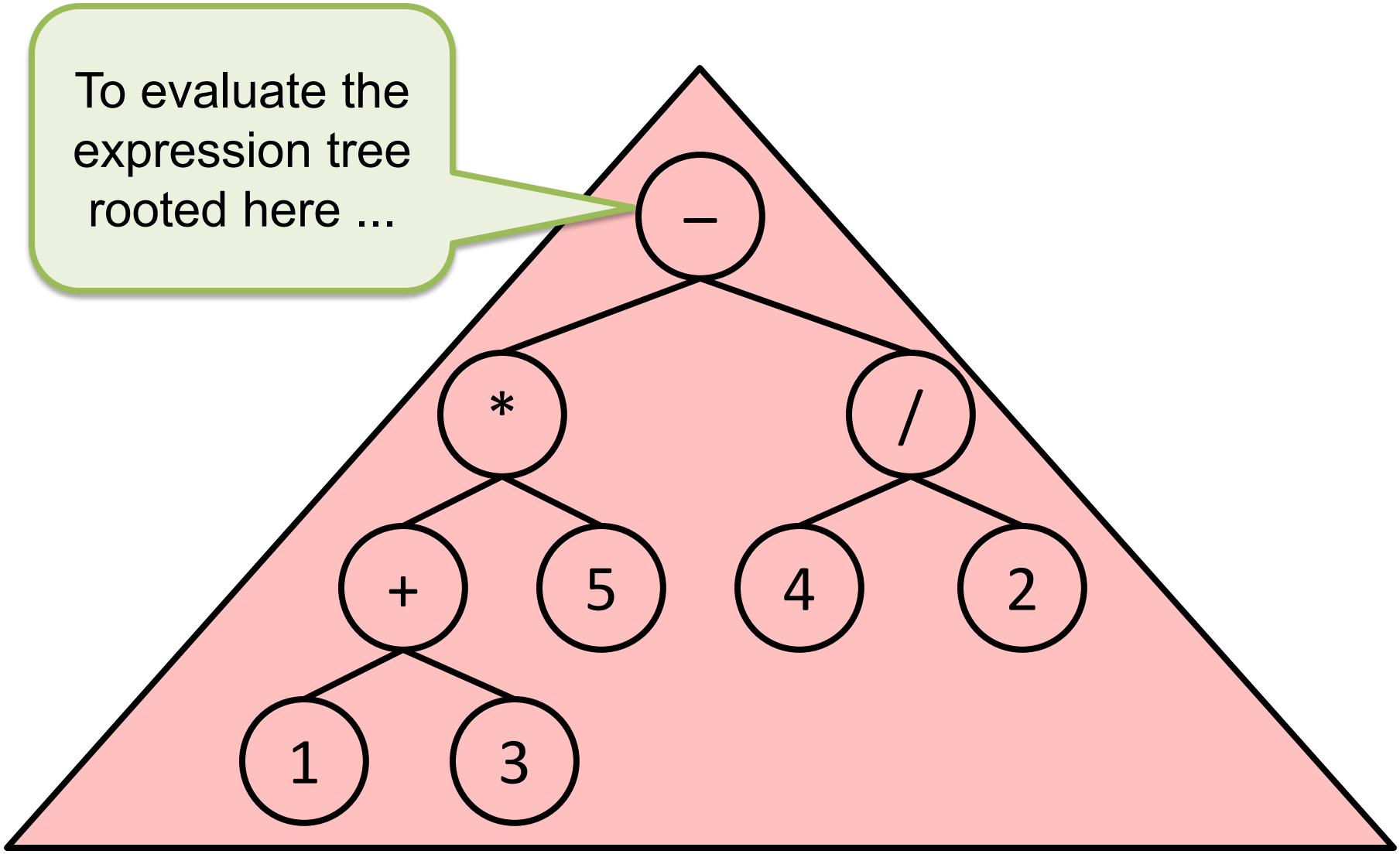
Tree Representation of Expression

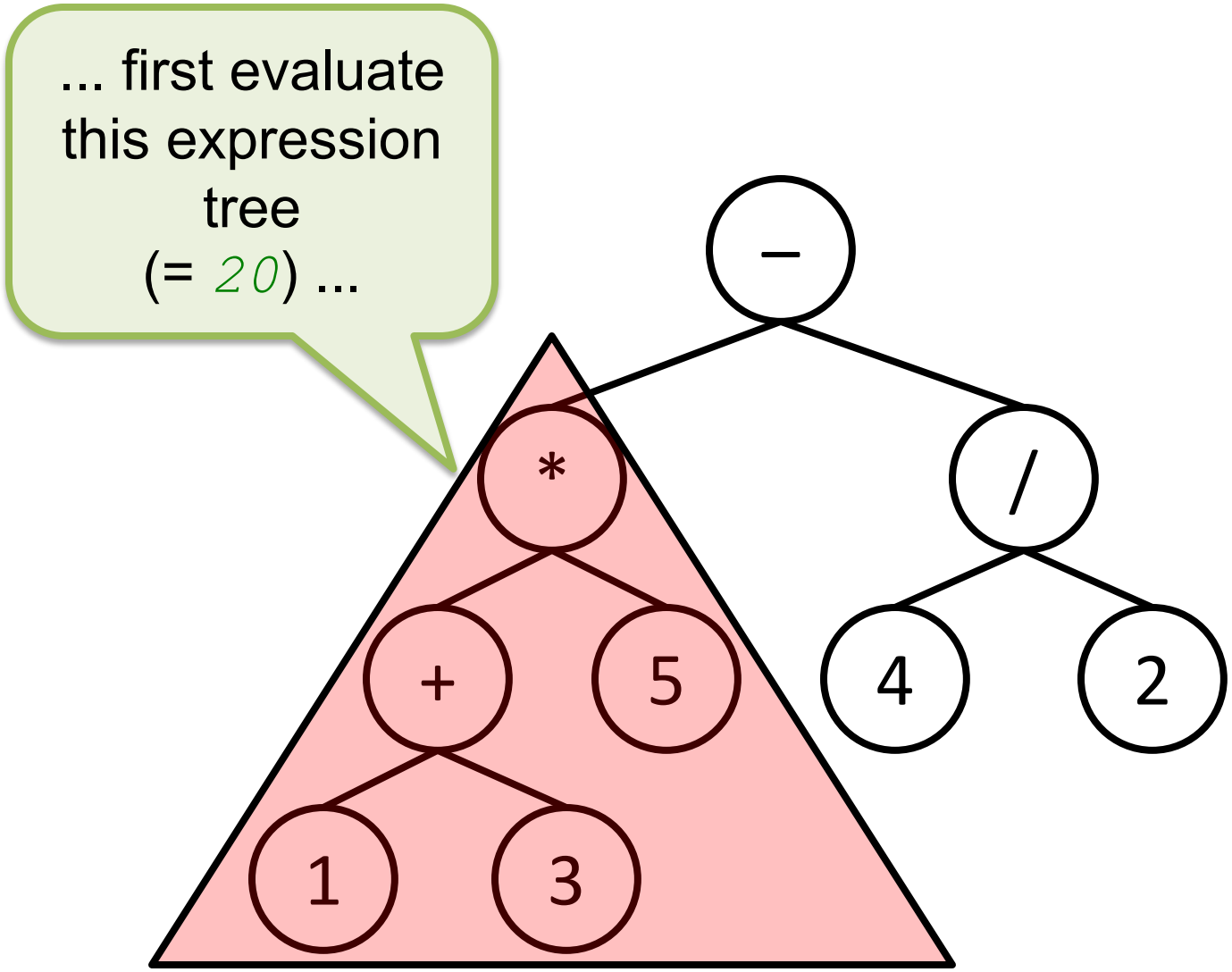
- So, this approach works:
  - *Last* operator evaluated is in root node
  - Each operator's left and right operands are its two subtrees (i.e., each operator has two subtrees, each of which is a subexpression in the larger expression)

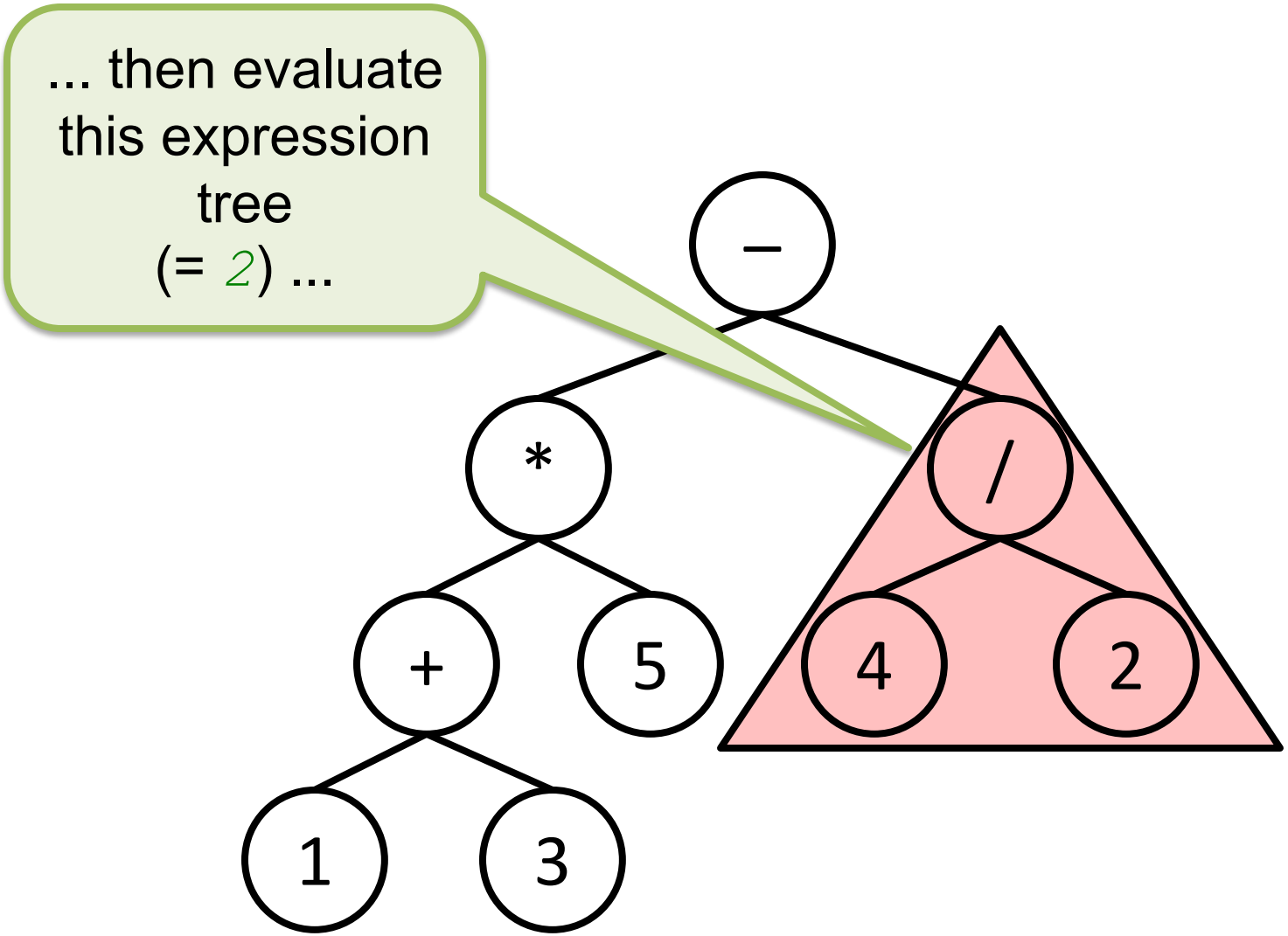

((1 + 3) * 5) - (4 / 2)
-
*
/
+
5
4
2
1
3

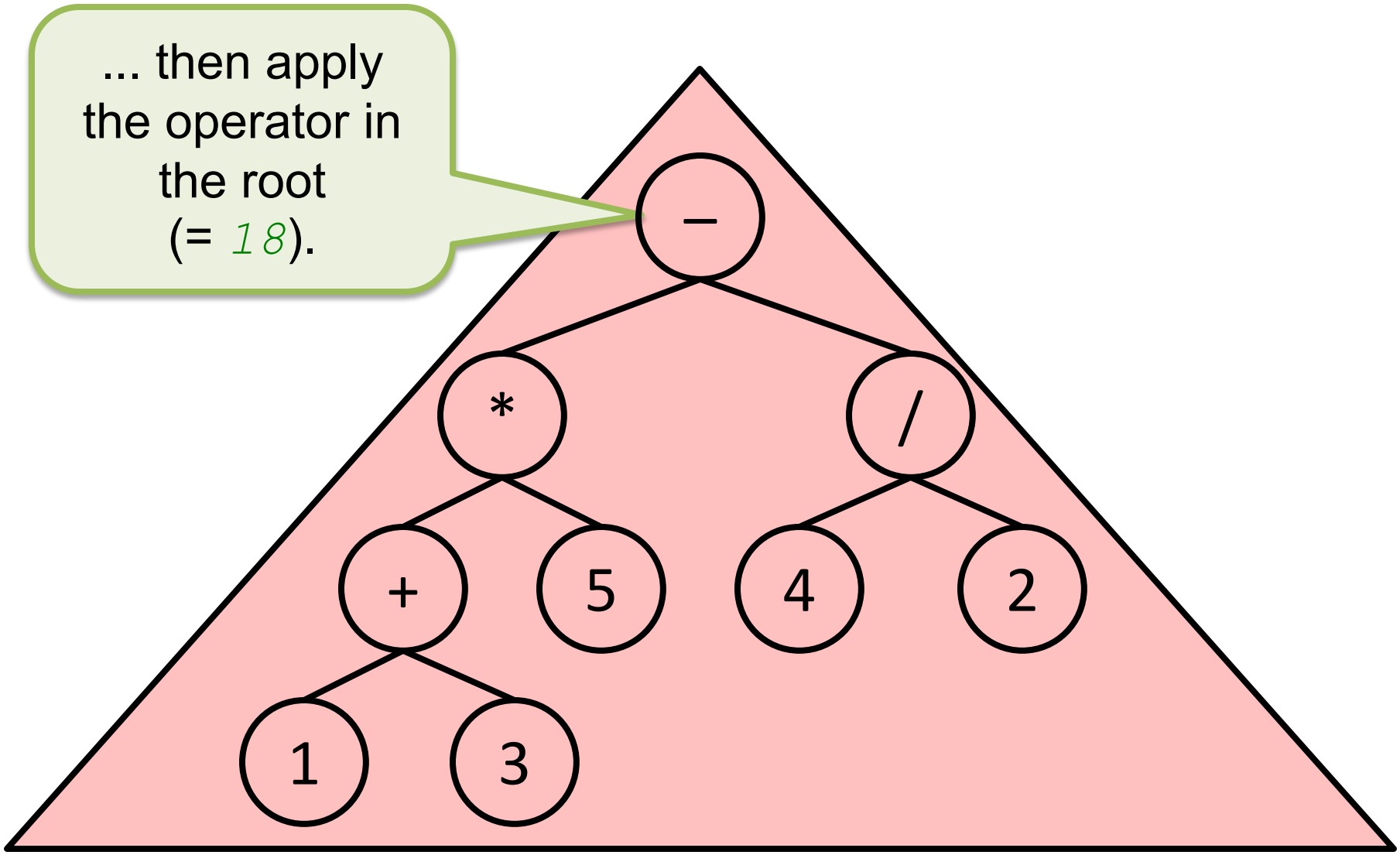
Evaluation of Expression Trees

- To ***evaluate*** any ***expression tree***:
  - If the root is an operator, then first ***evaluate*** the ***expression trees*** that are its left (first) and right (second) subtrees; then apply that operator to these two values, and the result is the value of the expression represented by the tree
  - If the root has no subtrees, then it must be an operand, and that operand is the value of the expression represented by the tree


To evaluate the expression tree rooted here ...
–
*
/
+
5
4
2
1
3


... first evaluate this expression tree (= 20) ...
–
*
/
+
5
4
2
1
3


... then evaluate this expression tree (= 2) ...
–
*
/
+
5
4
2
1
3


... then apply the operator in the root (= 18).
–
*
/
+
5
4
2
1
3

XML Encoding of Expressions

- The difference between an ***operator*** and an ***operand*** can be encoded in XML tags (e.g., "<operator>" and "<operand>")
  - The specific operator (e.g., "+", "–", "*", "/") can be either an attribute of an operator tag, or its content
  - Similarly, the value of an operand (e.g., "1", "34723576", etc.) ...
- Given such details for a specific XML encoding of expressions, you should be able to evaluate an expression given an `XMLTree` for its encoding