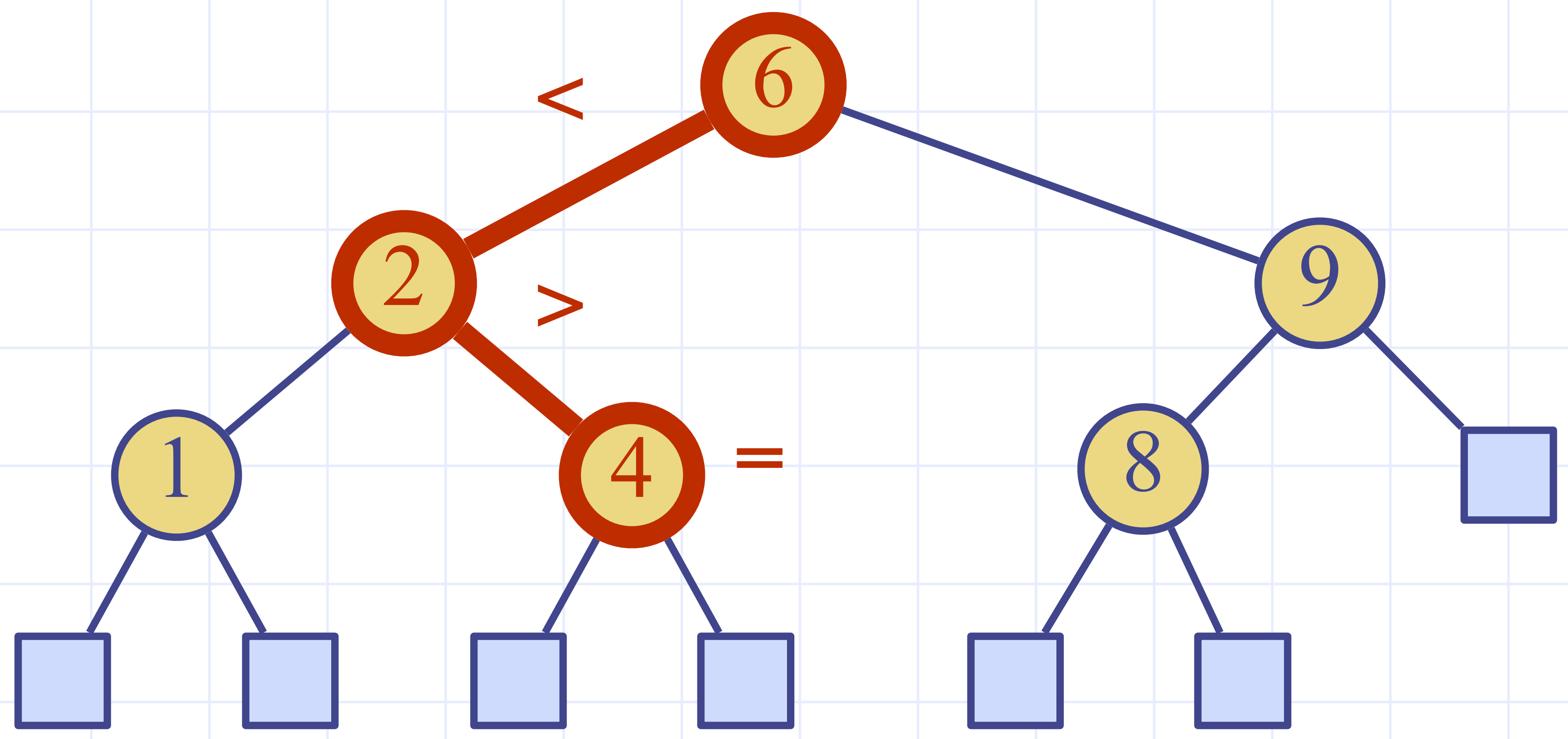
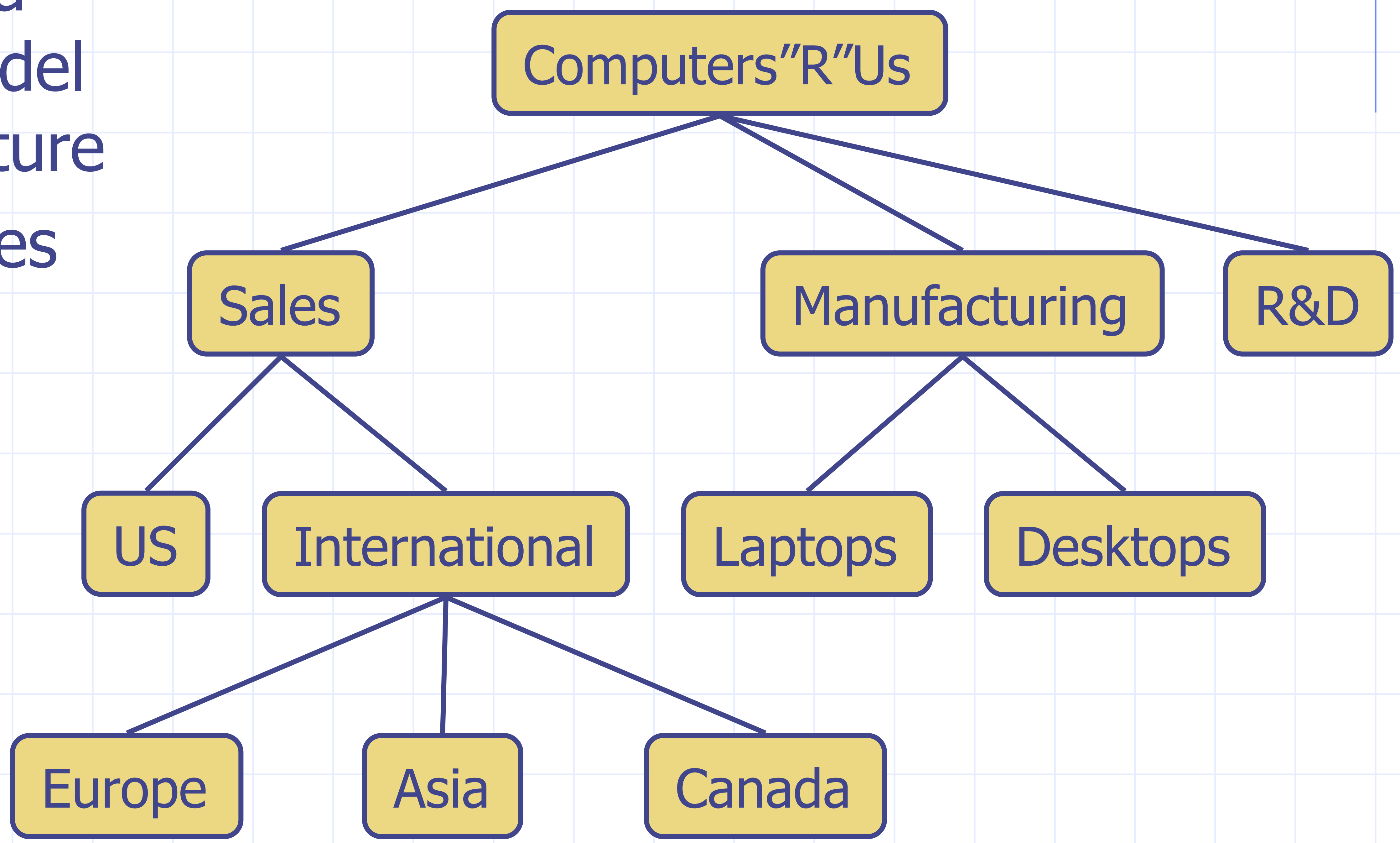
Binary Search Trees



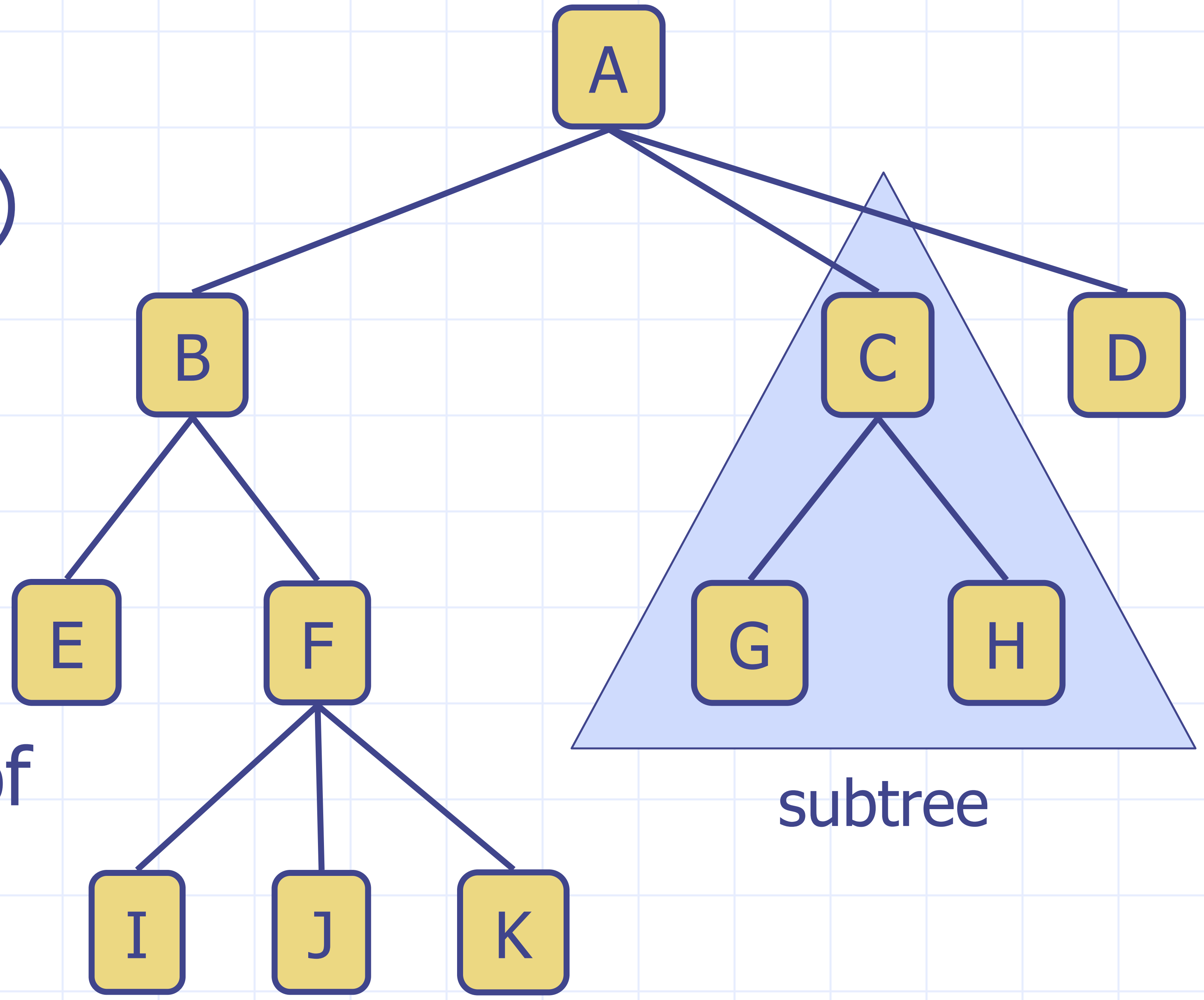
What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

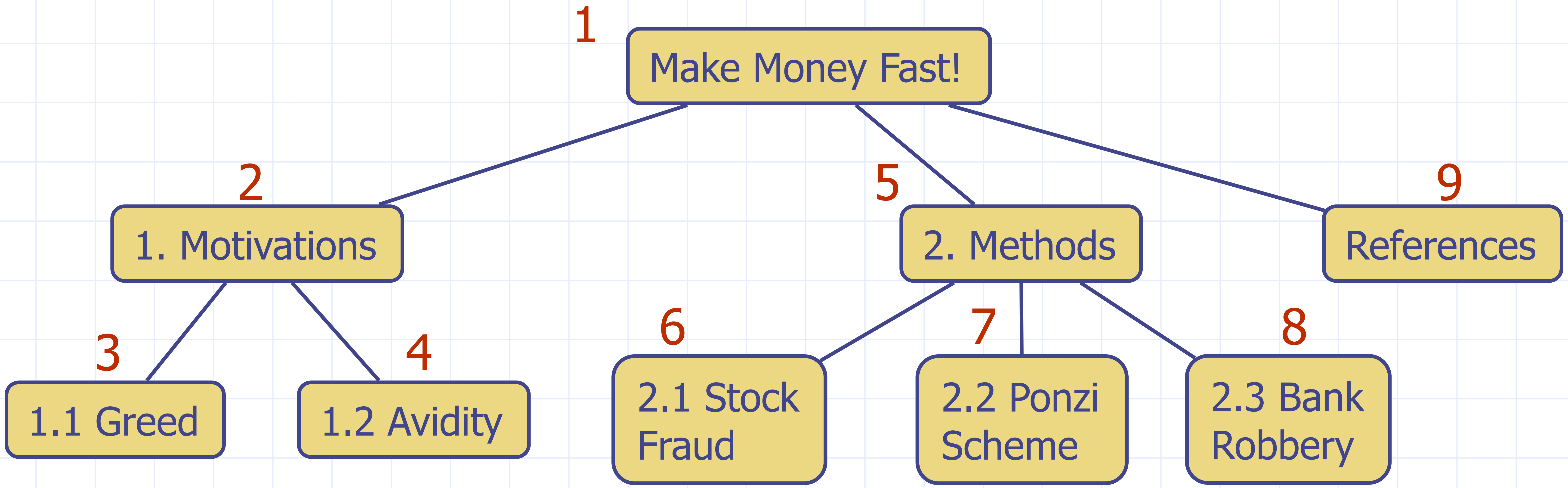
- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants



Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

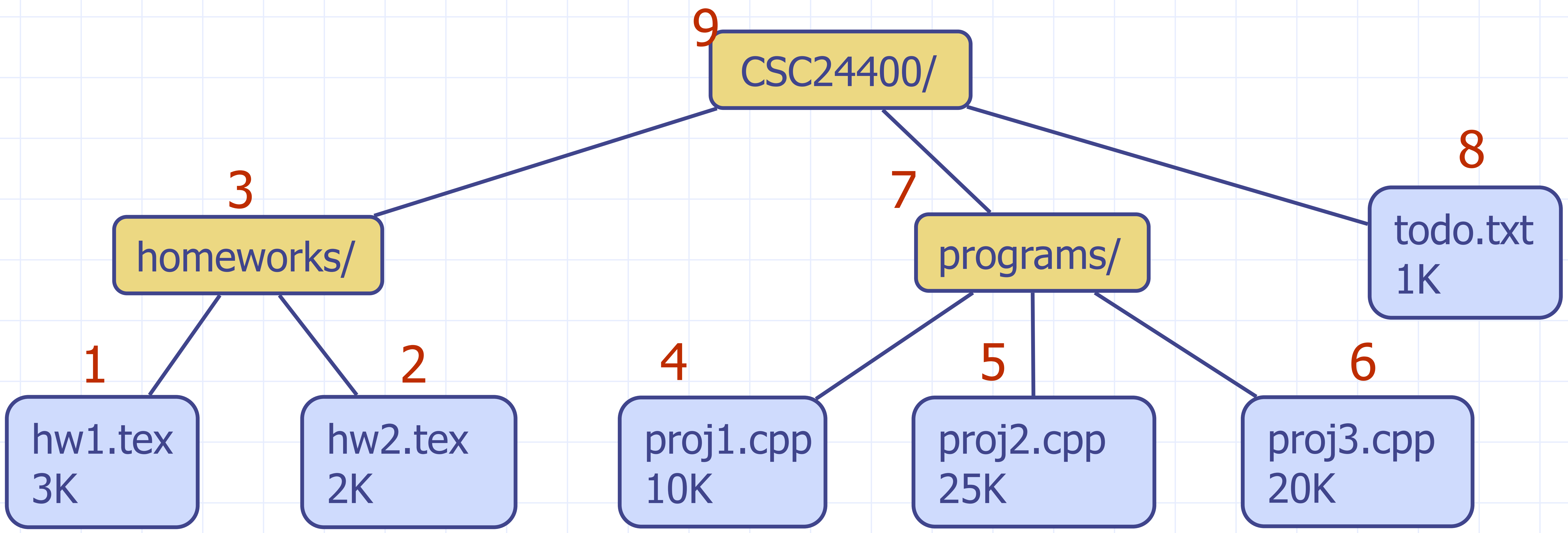
```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preOrder (w)
```



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

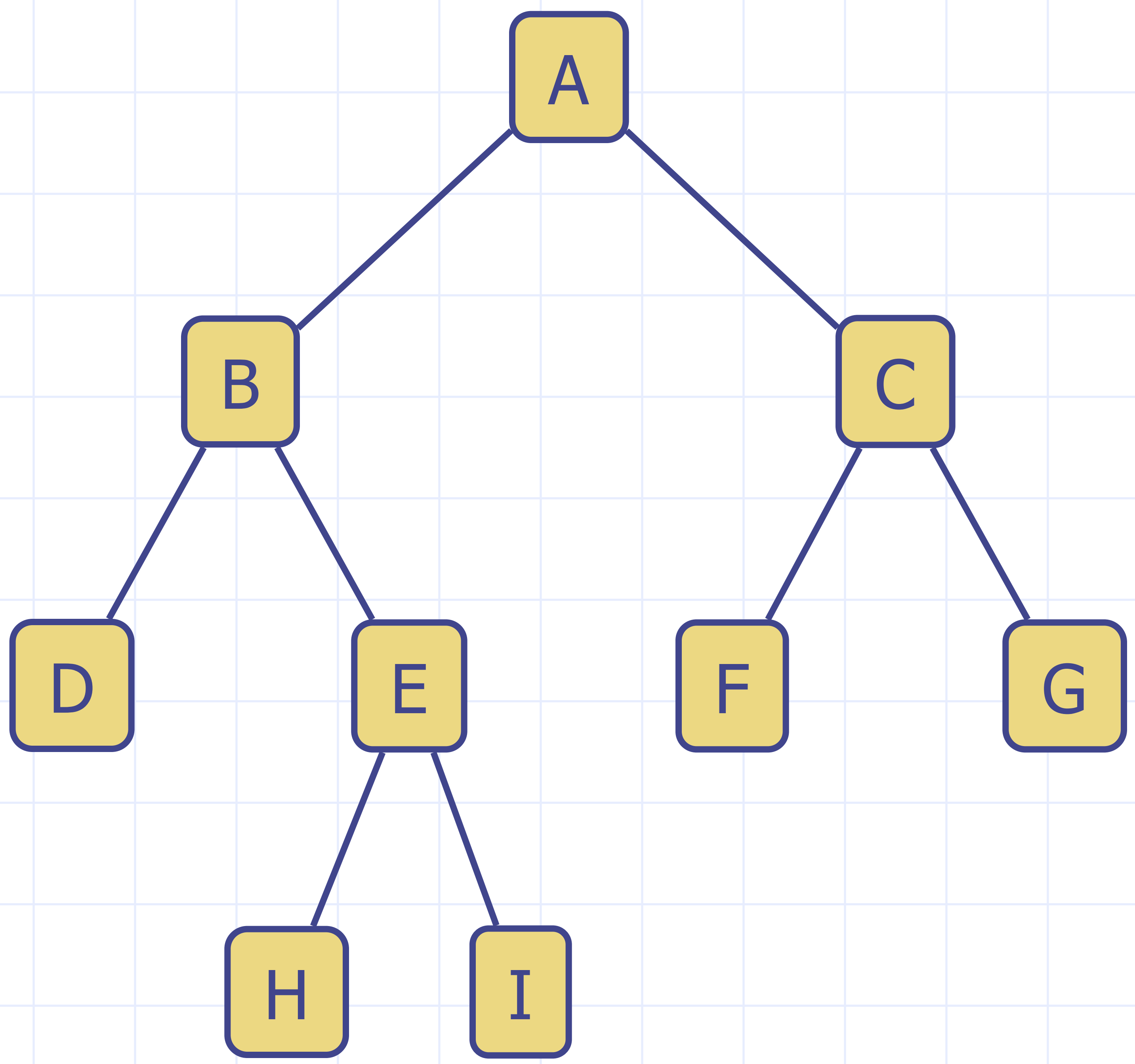
Algorithm *postOrder*(v)
    for each child w of v
        postOrder (w)
    visit(v)



Binary Trees

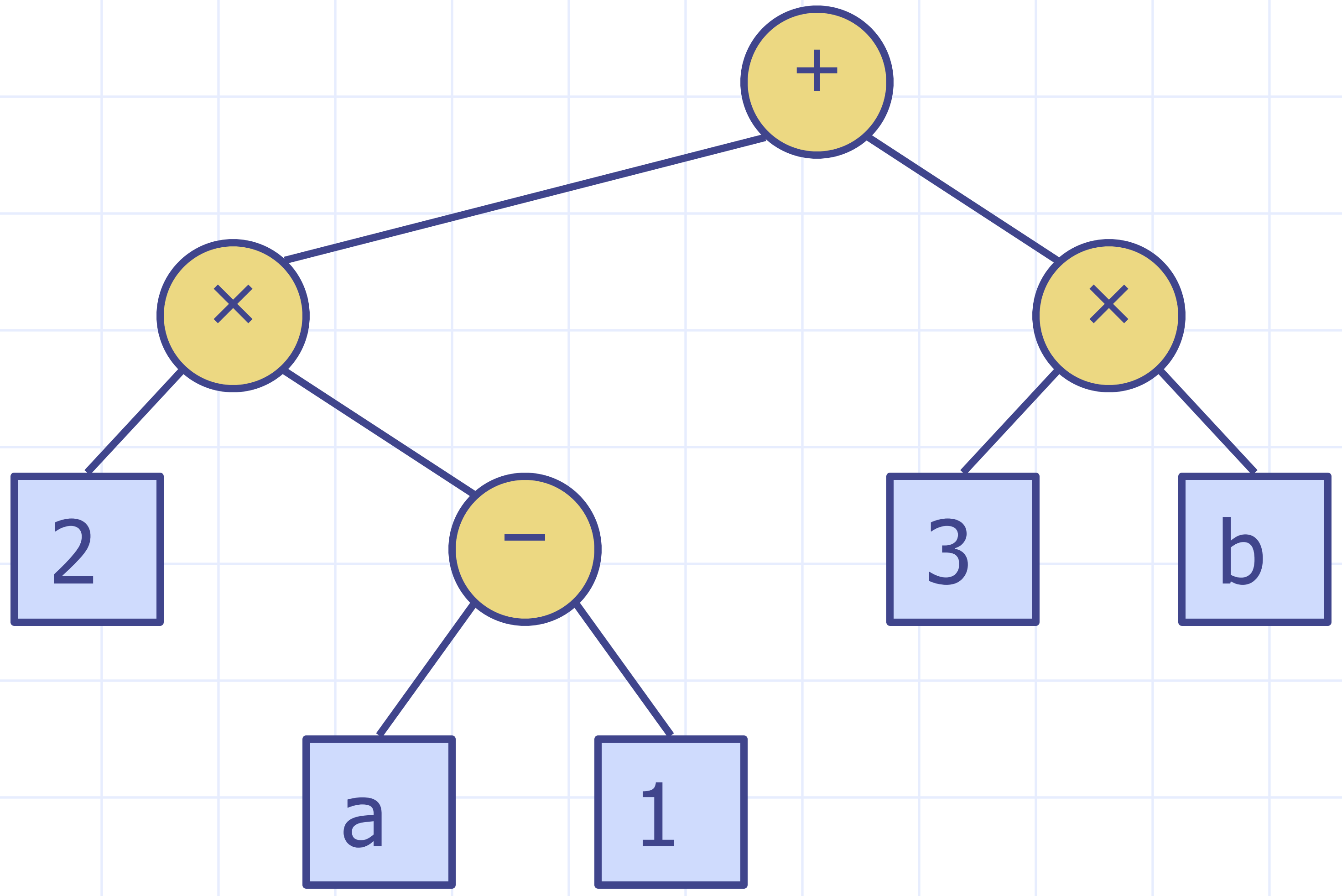
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



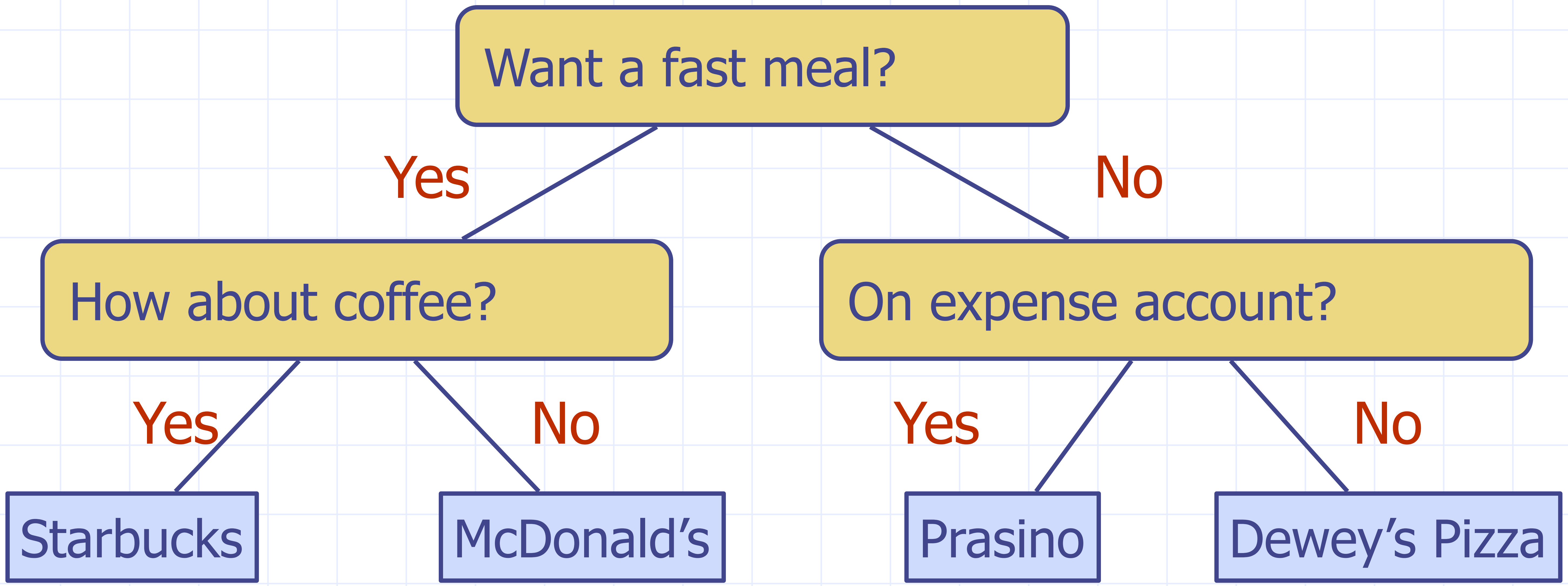
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



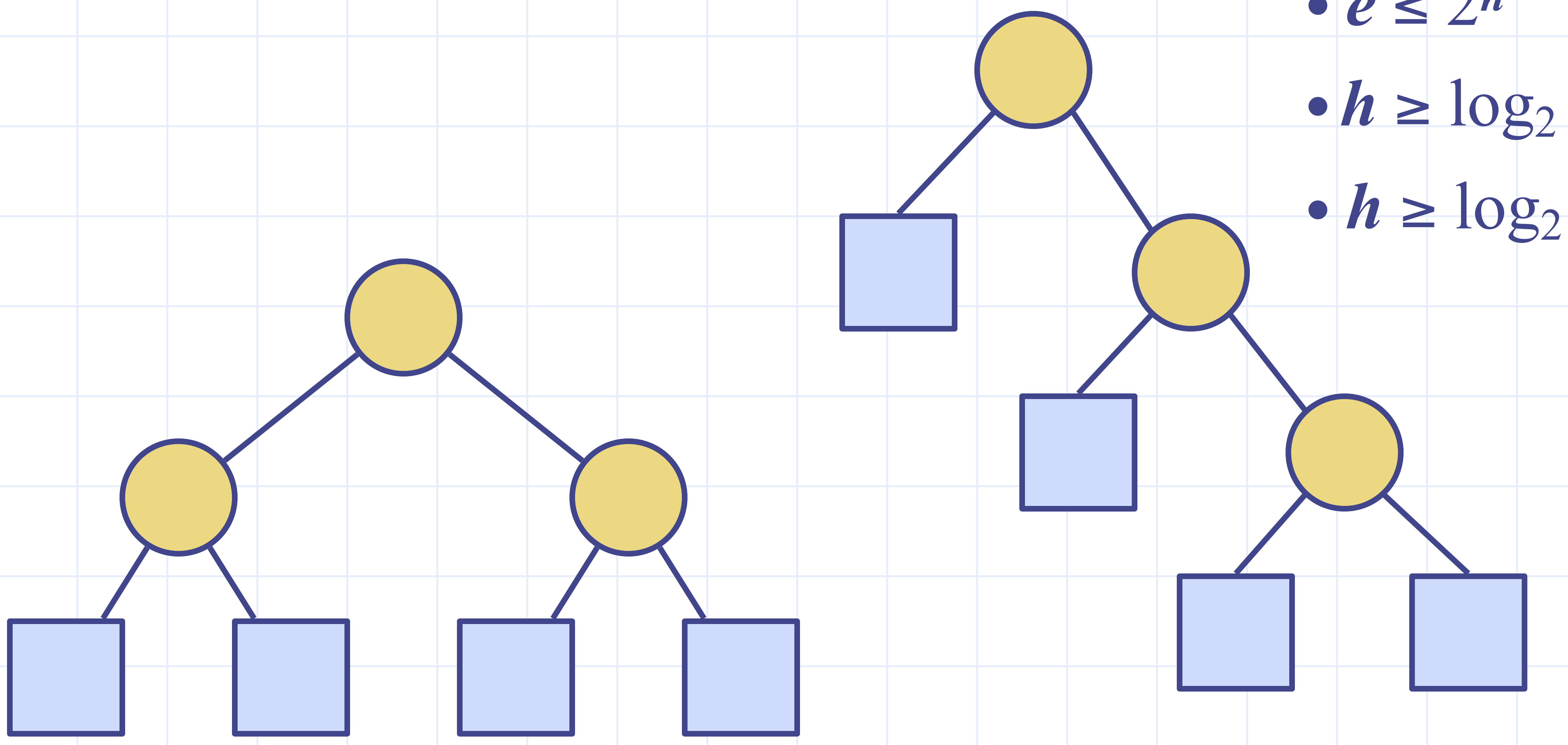
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Properties of Binary Trees

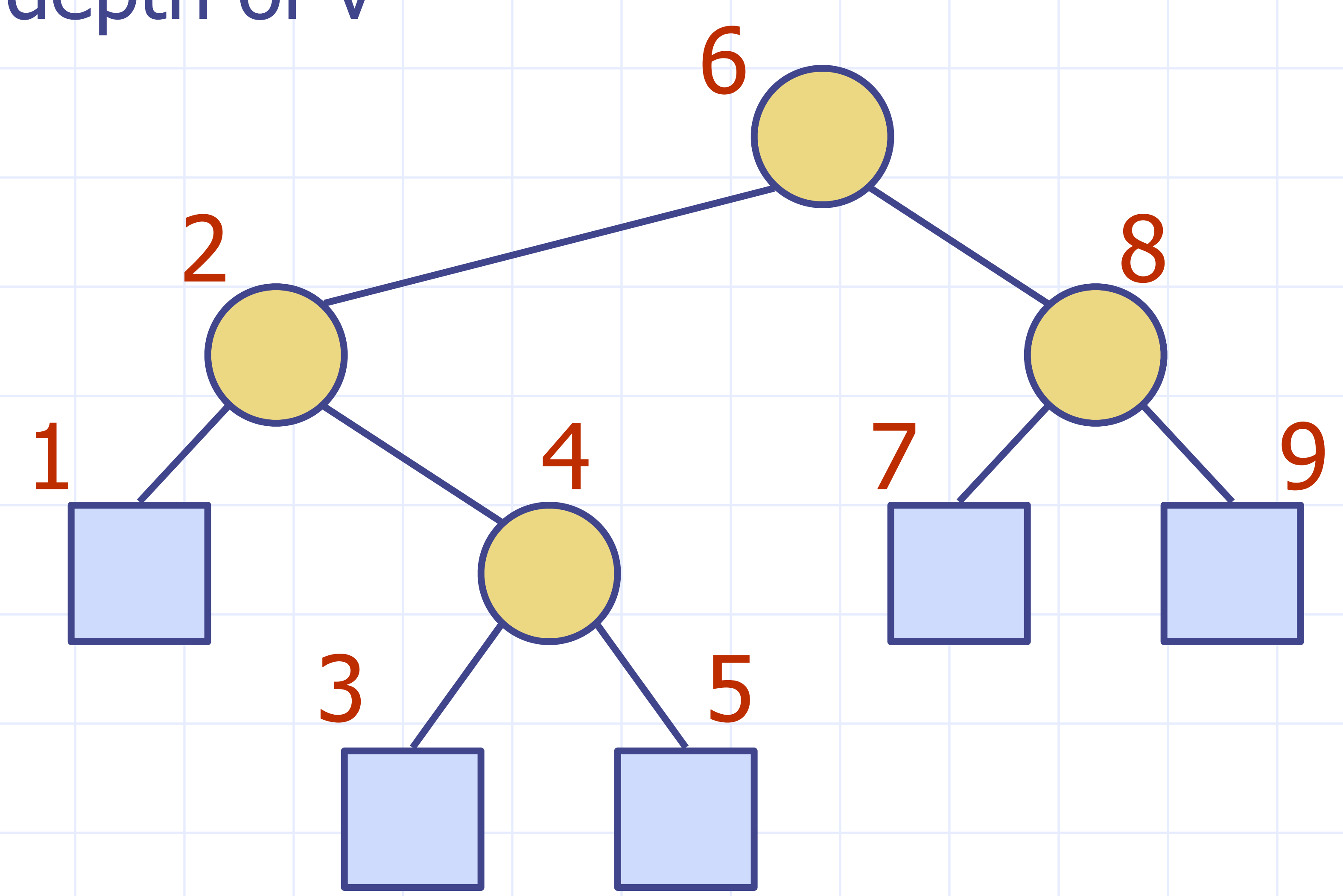
- Notation
  - n number of nodes
  - e number of leaf nodes
  - i number of internal nodes
  - h height
- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1 = \Omega(log(n))$



Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - $x(v)$ = inorder rank of v
  - $y(v)$ = depth of v

Algorithm *inOrder*(v)
    if !*v.isLeaf*()
        inOrder(*v.left*())
    visit(v)
    if !*v.isLeaf*()
        inOrder(*v.right*())

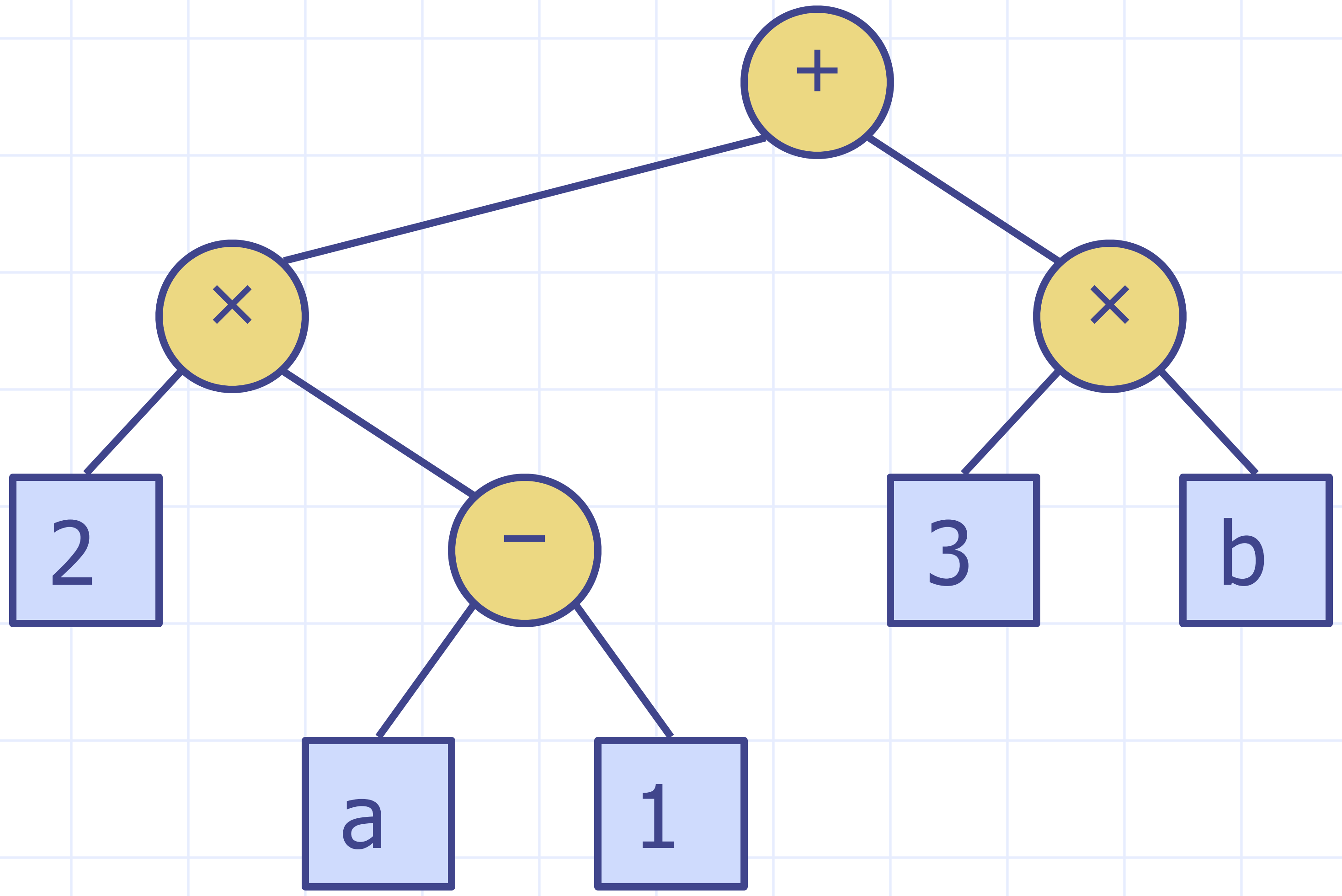


Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

Algorithm *printExpression(v)*

```
if !v.isLeaf()
    print("(")
    printExpression(v.left())
print(v.element())
if !v.isLeaf()
    printExpression(v.right())
    print (")")
```

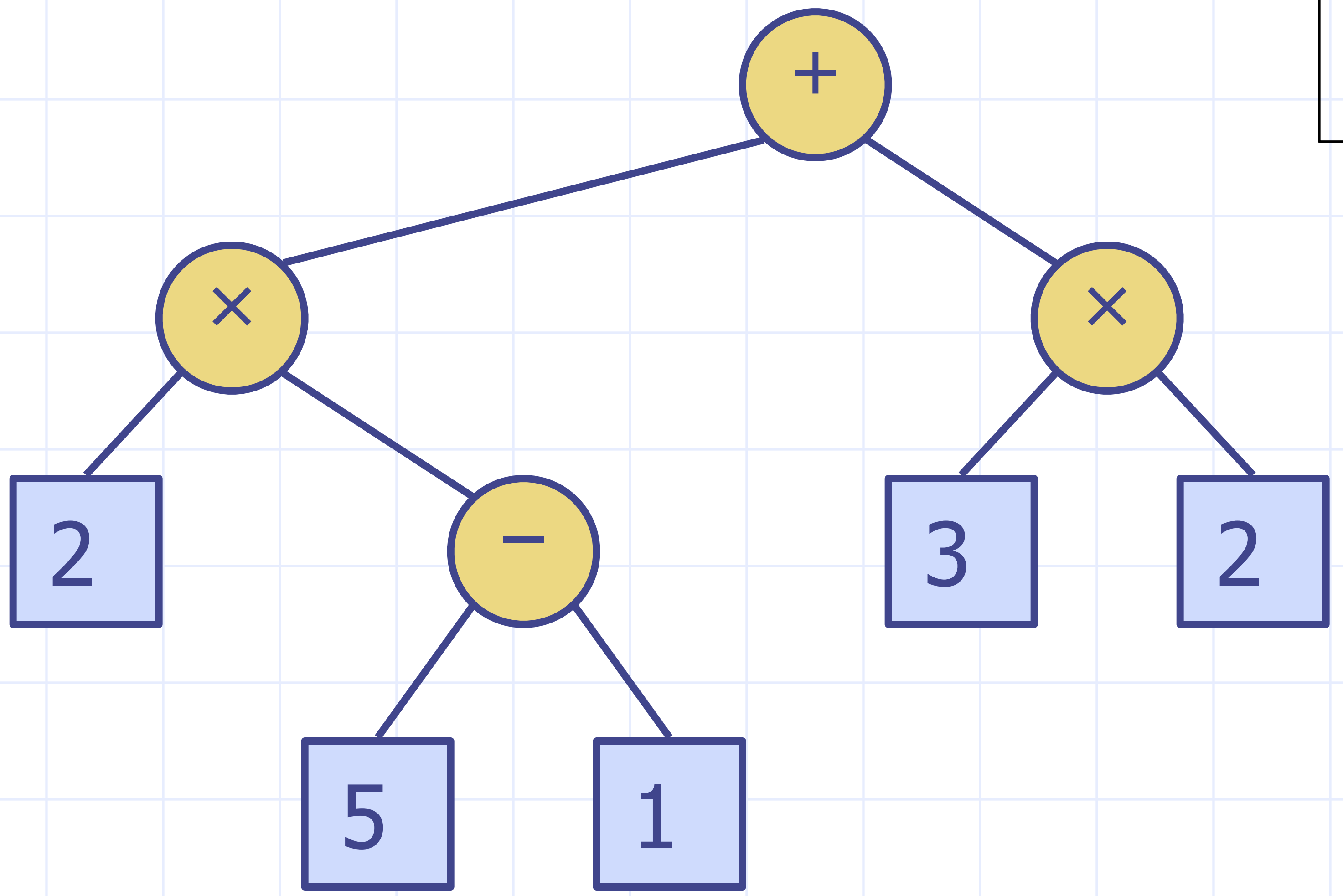


$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

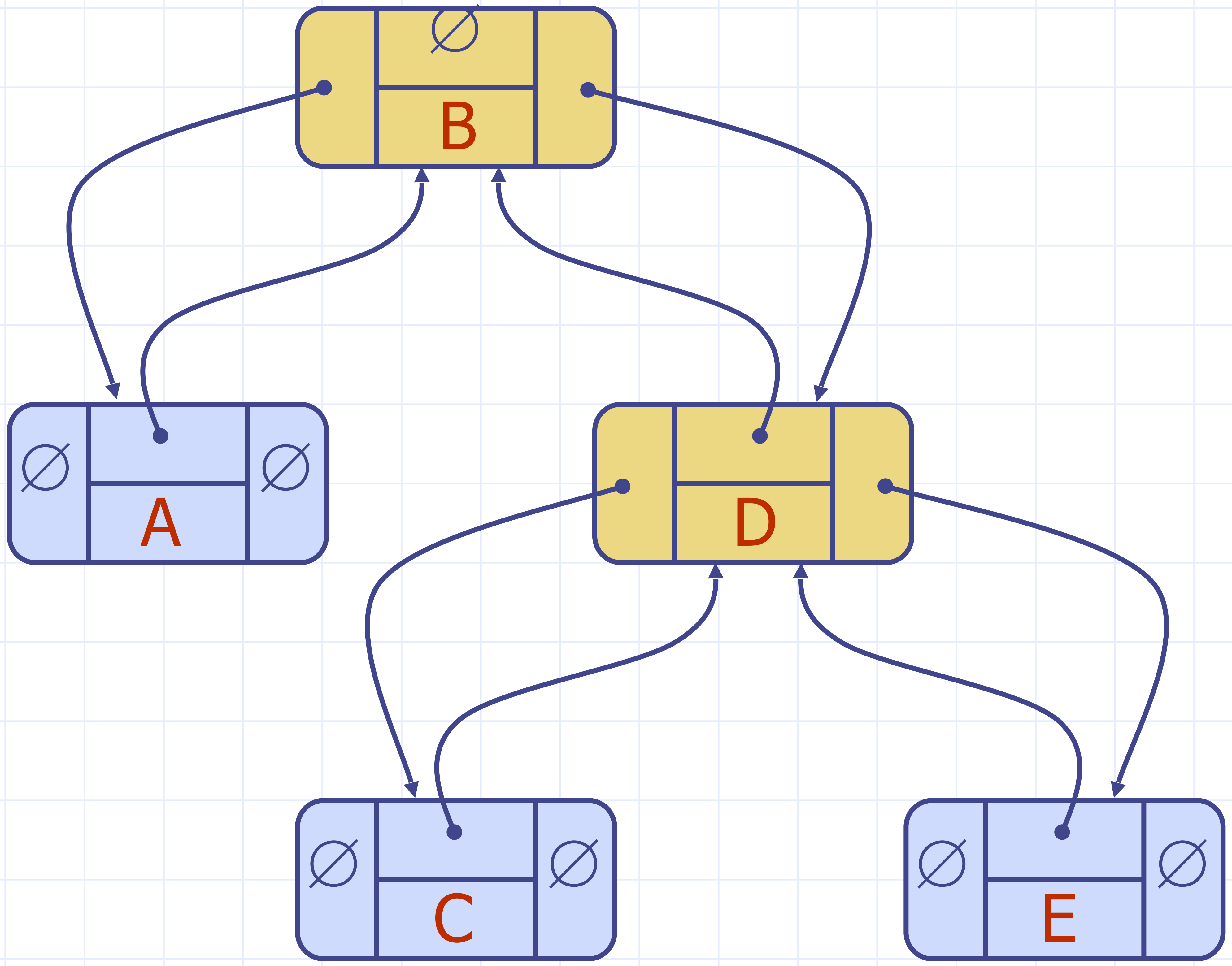
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

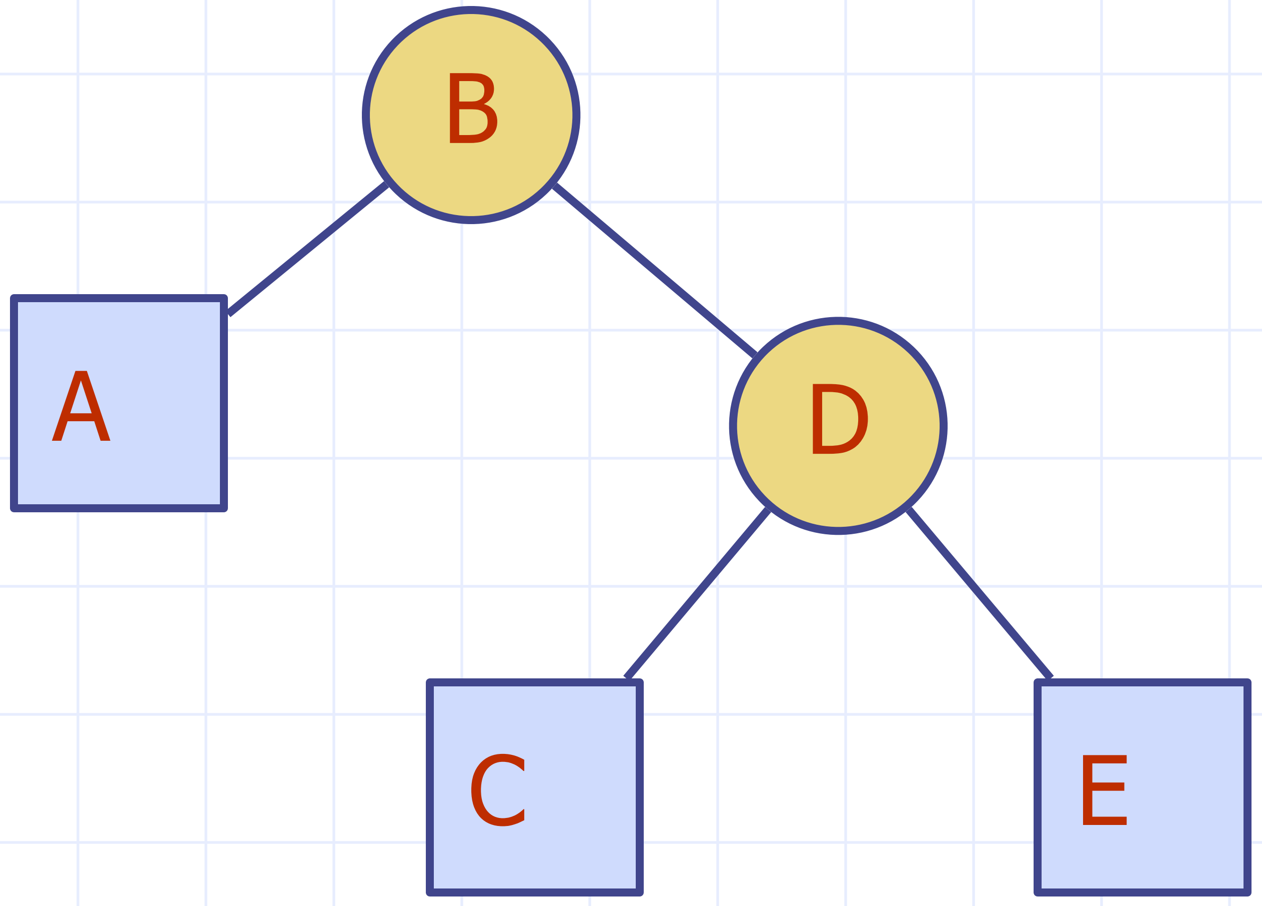
Algorithm *evalExpr*(v)
  if *v.isLeaf*()
    return *v.element*()
  else
    $x \leftarrow$ *evalExpr*(*v.left*())
    $y \leftarrow$ *evalExpr*(*v.right*())
    $\Diamond \leftarrow$ operator stored at v
    return $x \Diamond y$



Linked Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node (optional)
  - Left child node
  - Right child node

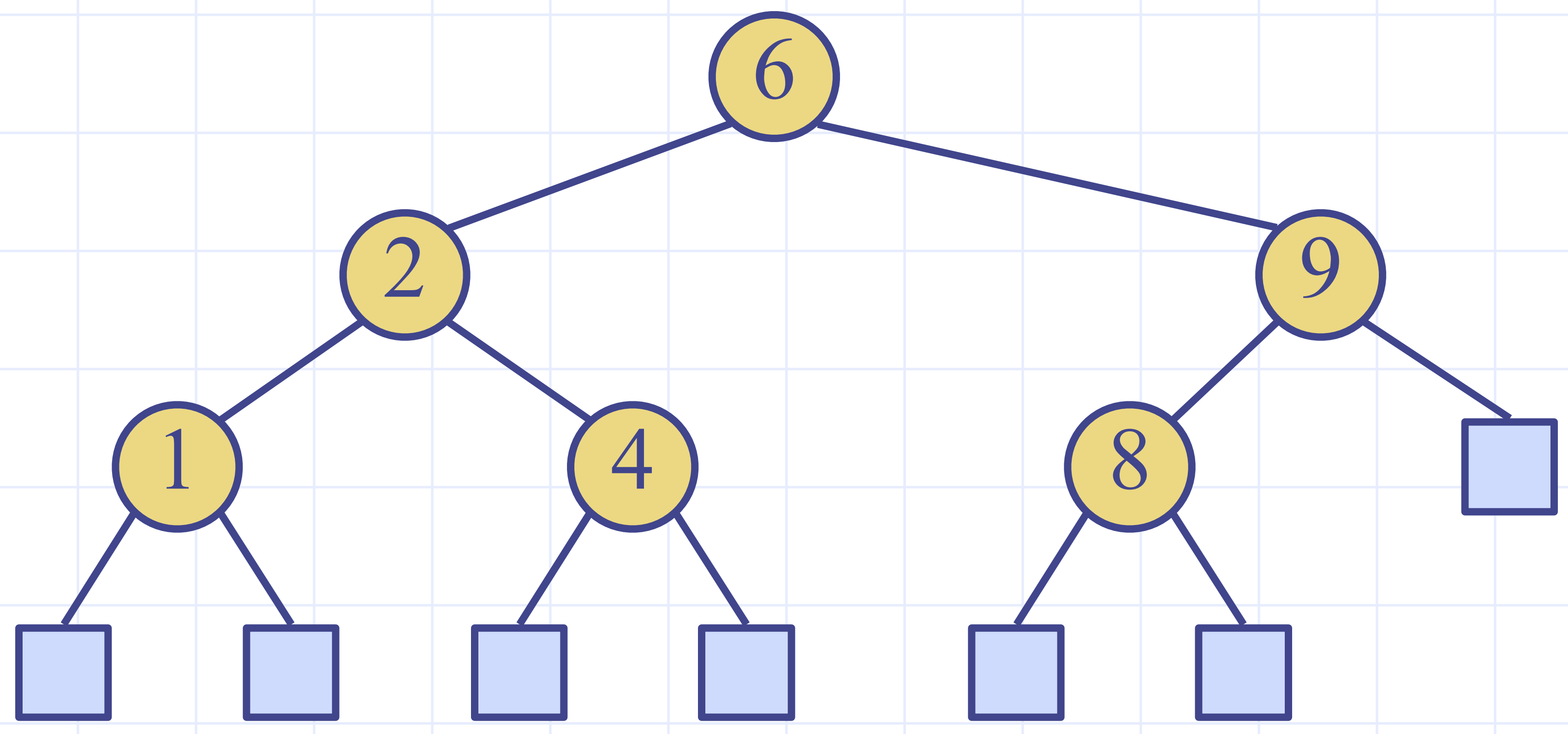




Binary Search Trees

- A binary search tree is a binary tree storing keys (or key-value entries) at its internal nodes and satisfying the following property:
  - Let u, v, and w be three nodes such that u is in the left subtree of v and w is in the right subtree of v. We have $key(u) \leq key(v) \leq key(w)$
- External nodes are usually represented immediately as null pointers, so they do not store items

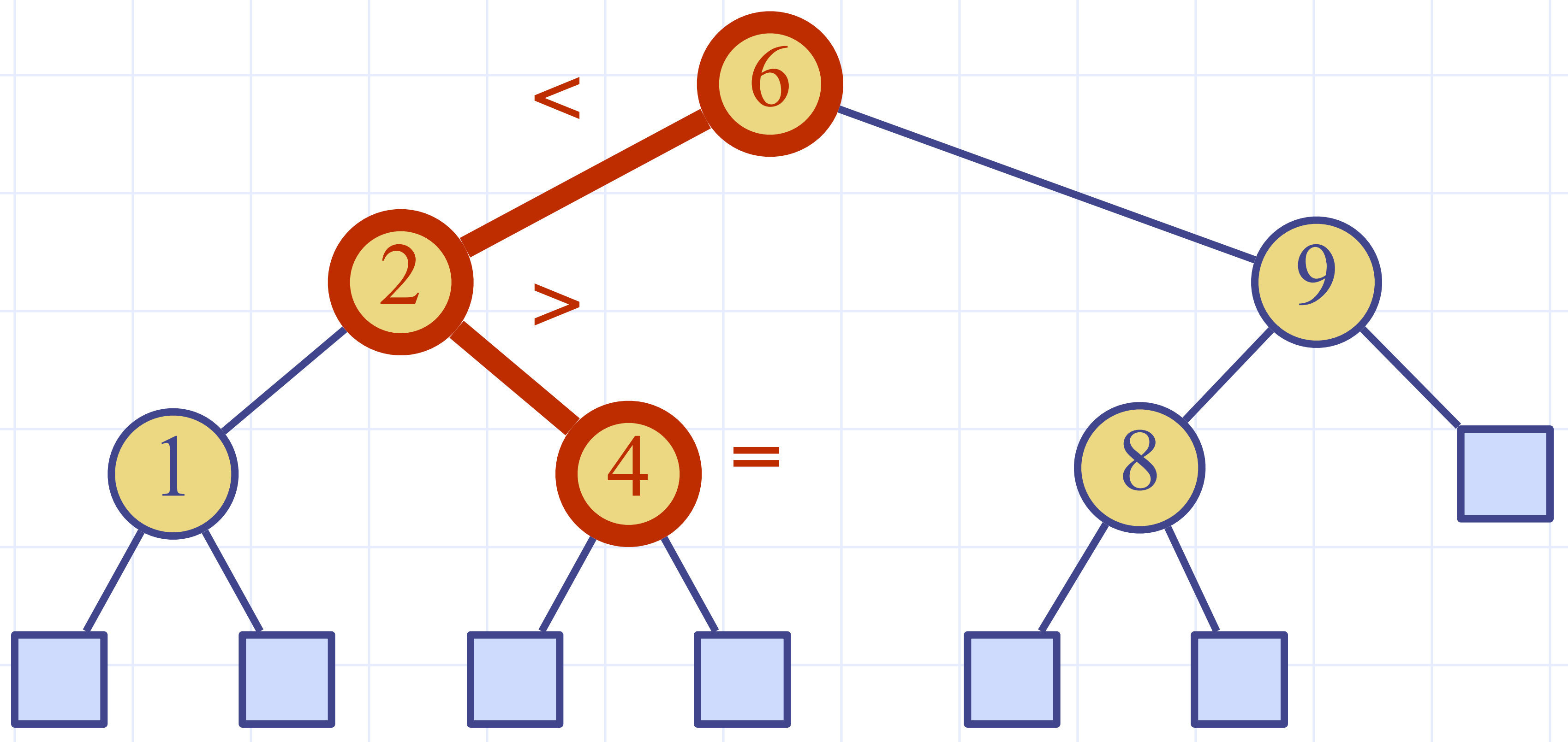
- An inorder traversal of a binary search trees visits the keys in increasing order



Search

- To search for a key k, we trace a downward path starting at the root
- The next node visited depends on the comparison of k with the key of the current node
- If we reach a leaf, the key is not found
- Example: find(4):
  - Call TreeSearch(4,root)

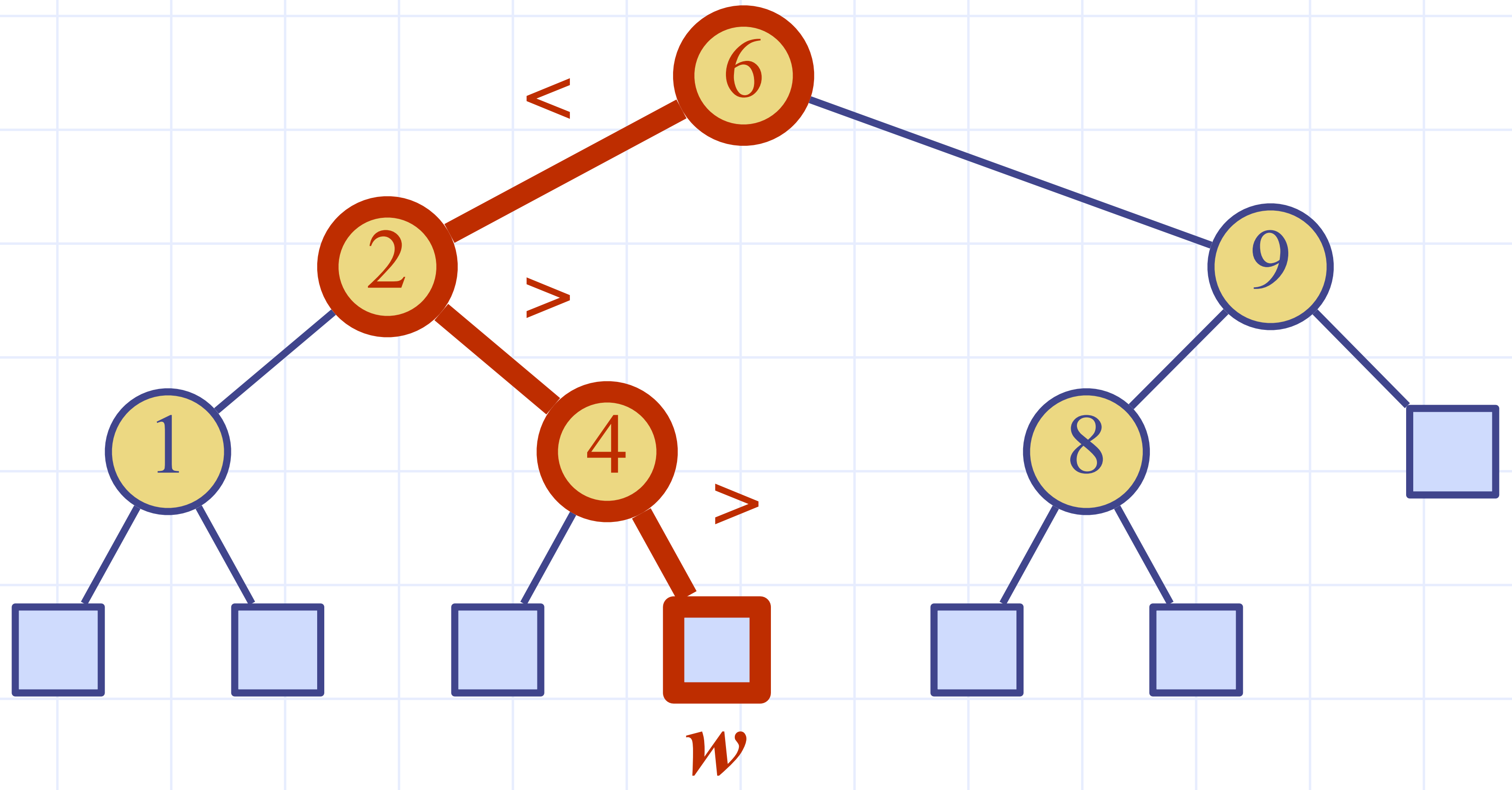
```
Algorithm TreeSearch(k, v)
  if v.isLeaf ()
    return v  //null? depends on implementation
  if k < v.key()
    return TreeSearch(k, v.left())
  else if k = v.key()
    return v
  else // k > v.key()
    return TreeSearch(k, v.right())
```

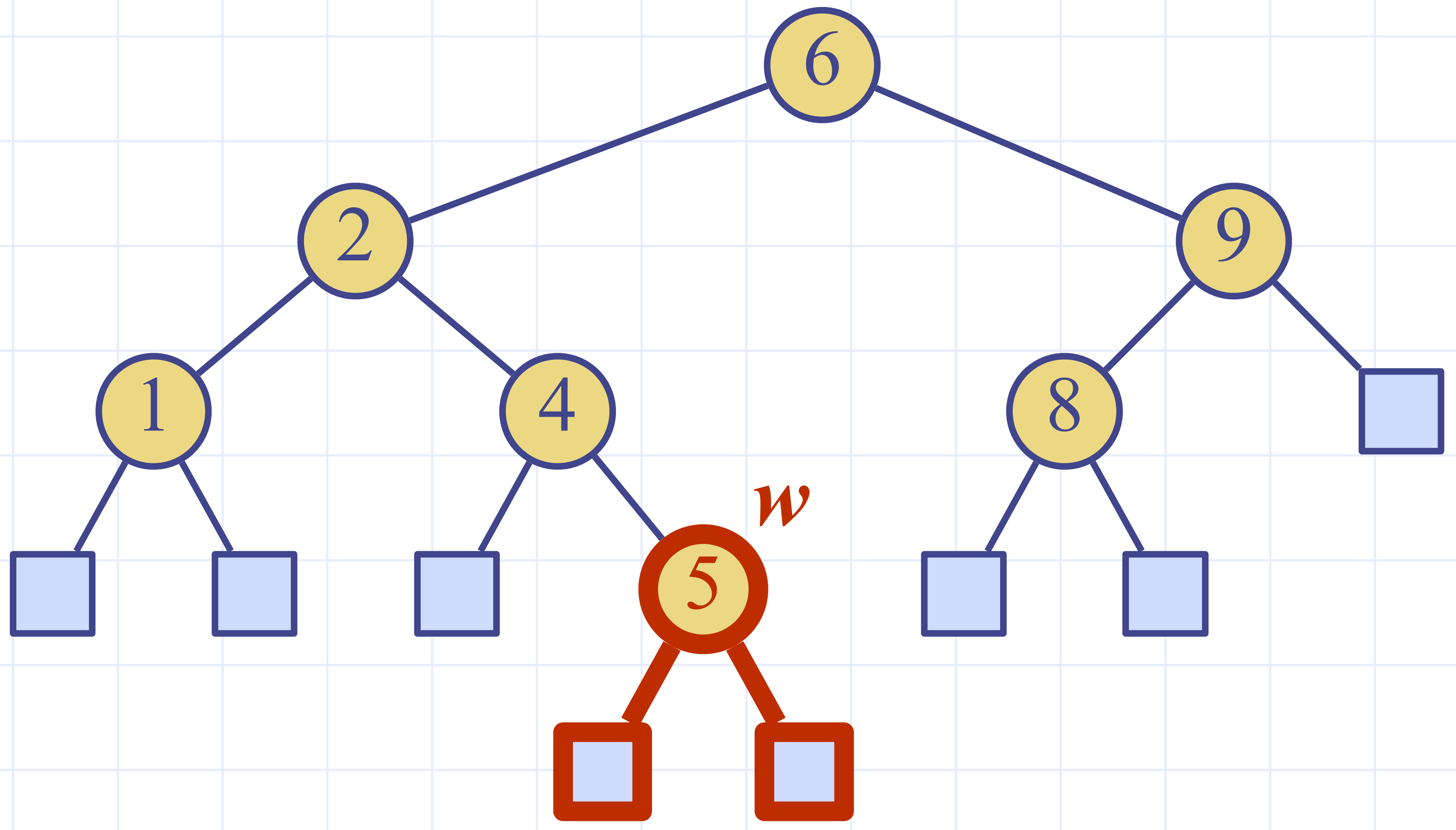


Insertion

- To perform operation insert(k), we search for key k
- Assume k is not already in the tree, and let w be the leaf reached by the search

  note that we'll always get to a leaf - **<u>why?</u>**
- We insert k at node w by making w's "parent" the node previously found in the search.
  - don't forget to update w's parent's appropriate child!
- Example: insert 5

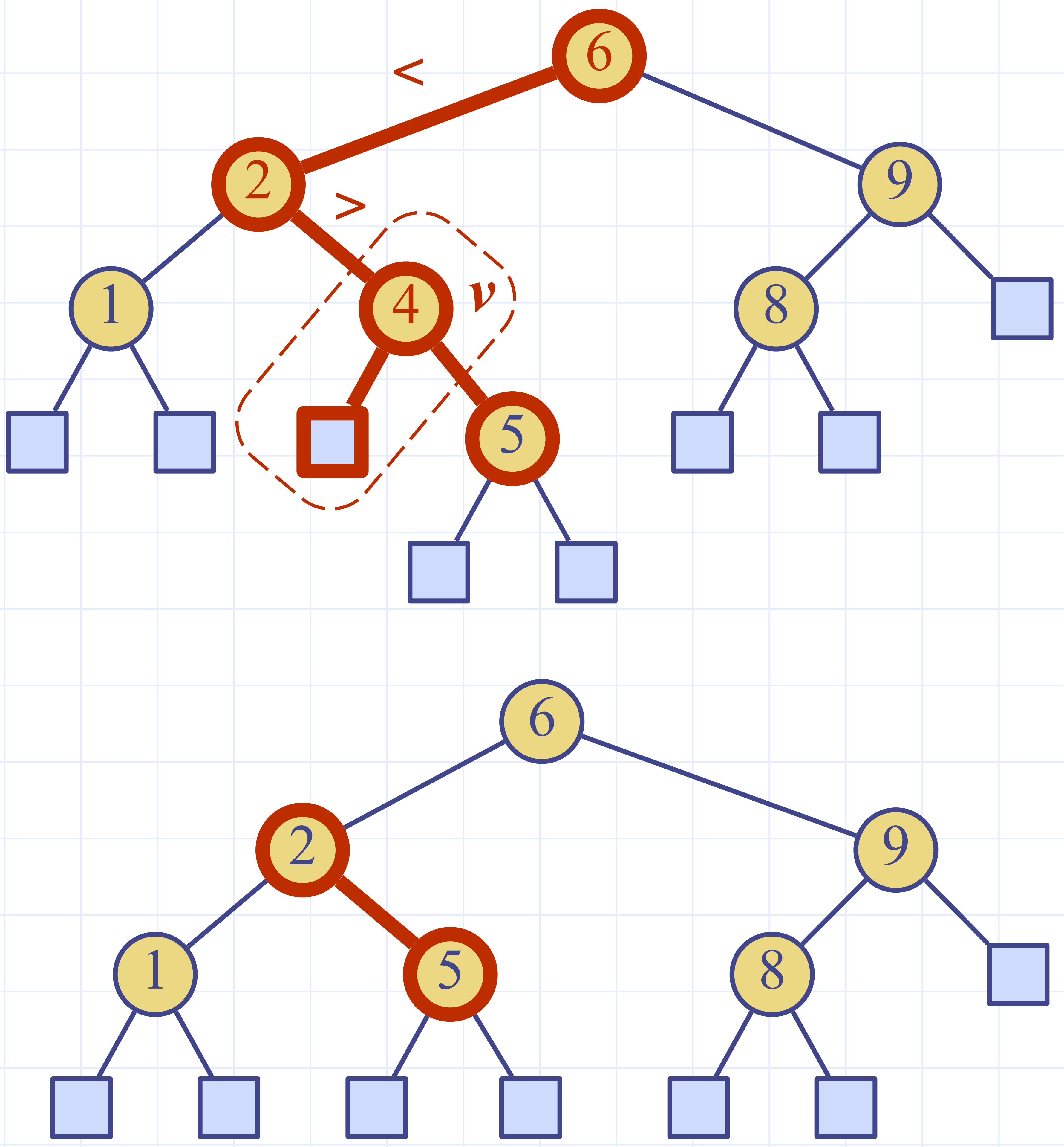




Deletion

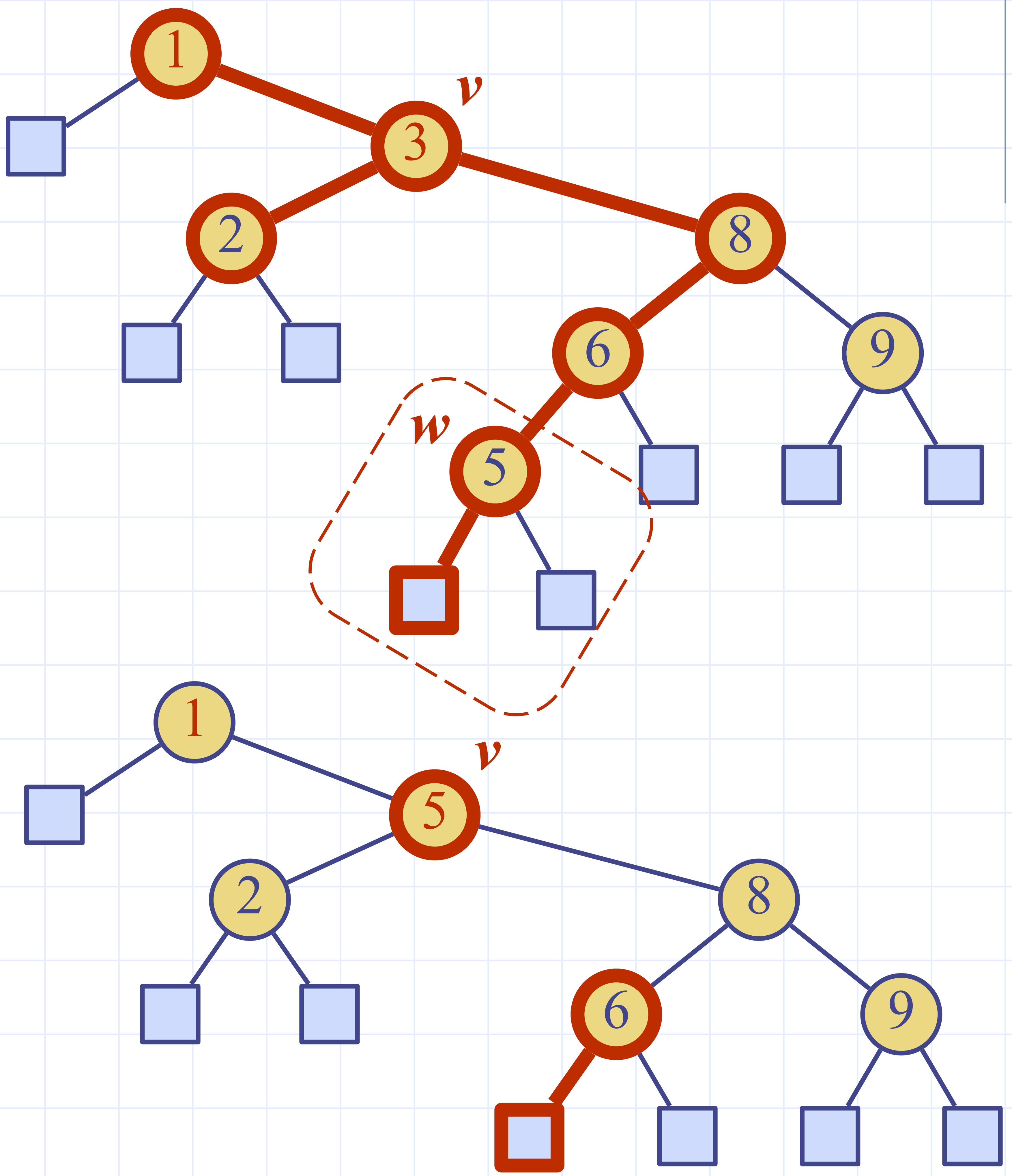
- To perform operation erase(k), we search for key k
- Assume key k is in the tree, and let let v be the node storing k
- If node v is a leaf, just tell the parent to make that child null
  - so simple, not shown here
- If node v has only one child, make that child's parent be v's parent and have v's parent's child point to v's child.

  Example: remove 4



Deletion (cont.)

- Now consider the case where the key k to be removed is stored at a node v whose children are both internal
  - find the internal node w that follows v in an inorder traversal
  - copy $key(w)$ into node v
  - remove node w (see previous slide)
    - Example: remove 3



Performance

- Consider a binary search tree of height h
  - the space used is $O(n)$
  - methods find, insert and erase take $O(h)$ time
- The height h is $O(n)$ in the worst case and $\Omega(\log n)$ in the best case