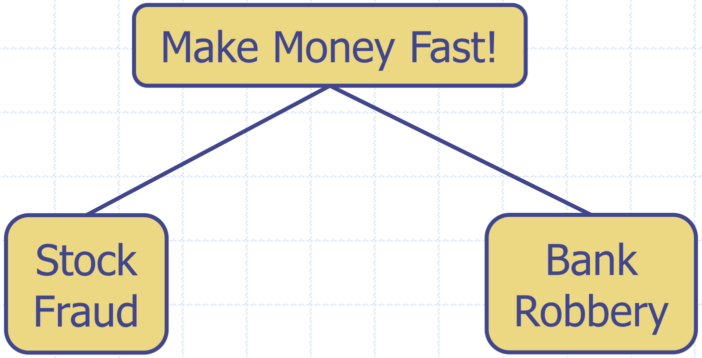
Binary Trees

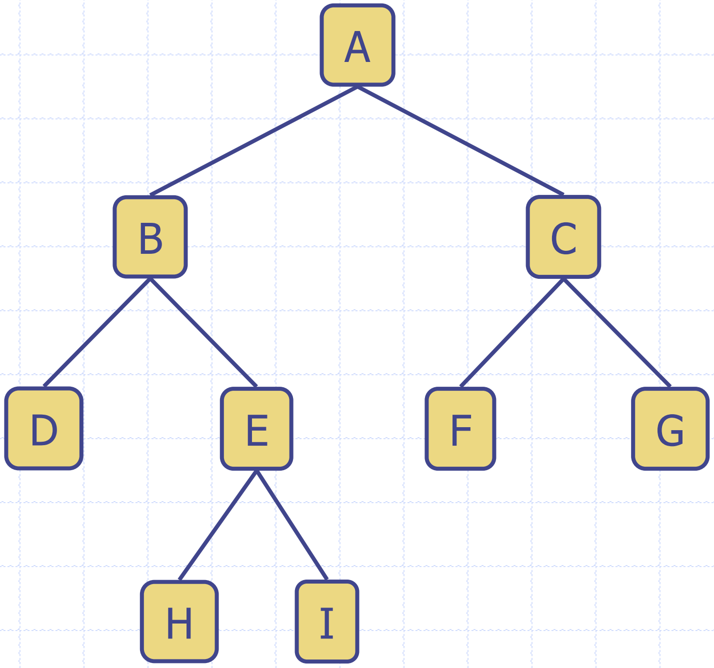
Section 7.3



Binary Trees

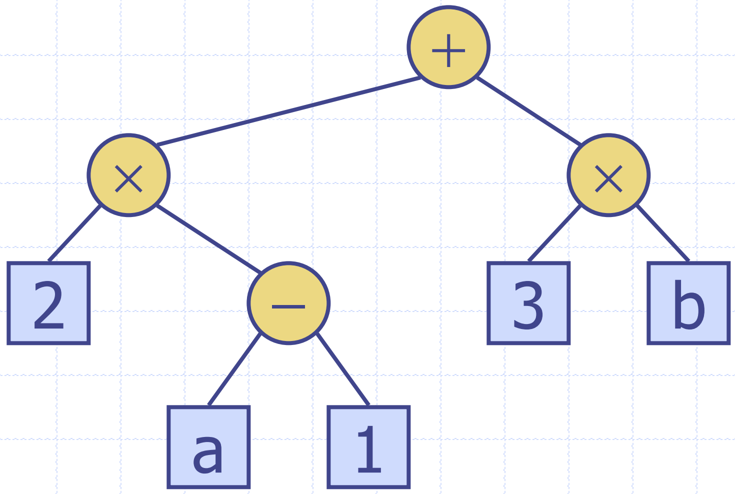
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary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



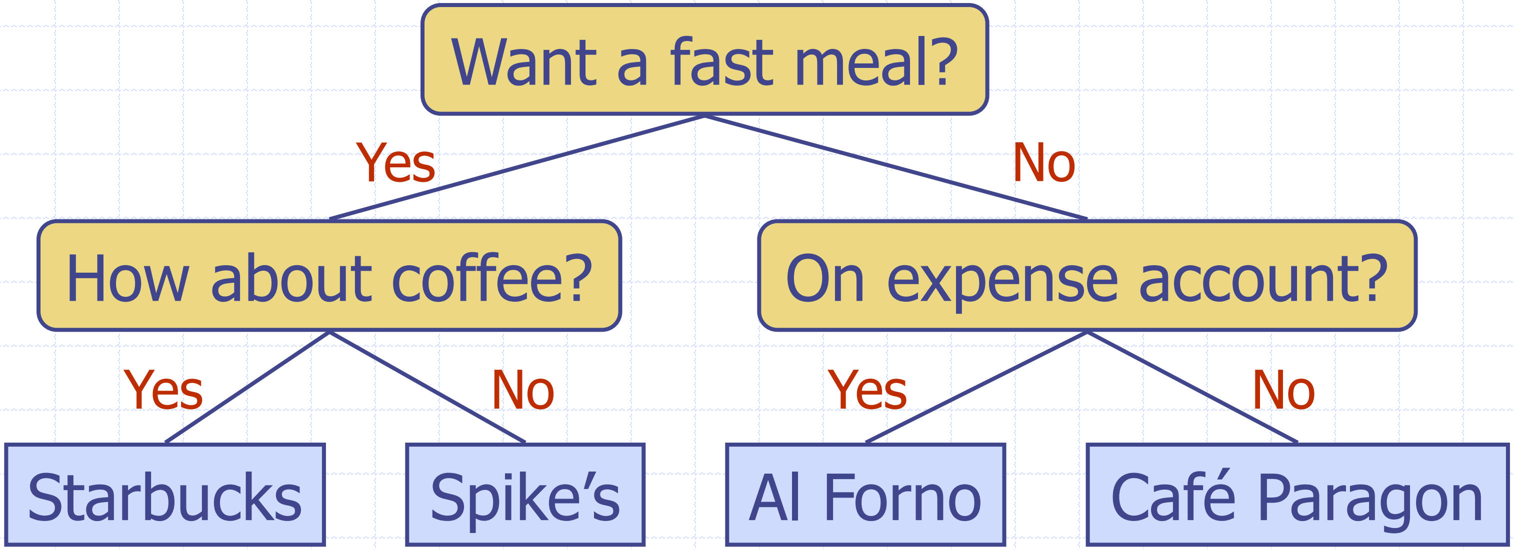
Arithmetic Expression Tree

- 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

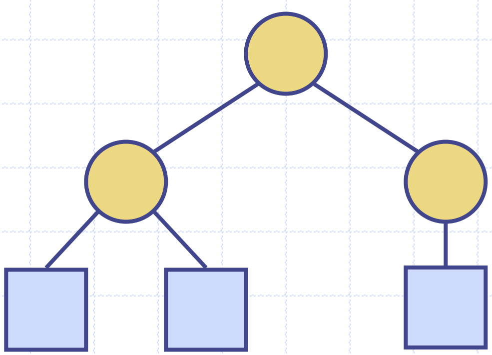
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

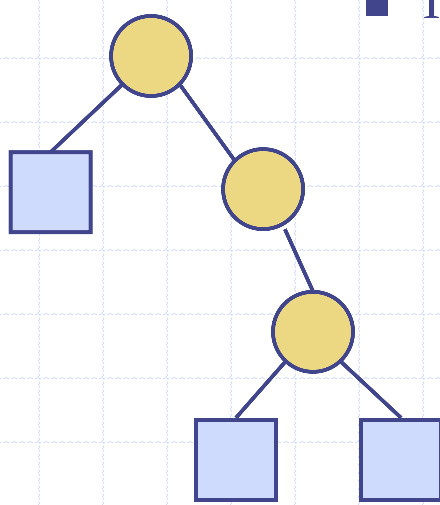


Properties of Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height

- Properties:
  - $h + 1 \le n \le 2^{h+1} - 1$
  - $1 \le e \le 2^{h}$
  - $h \le i \le 2^{h} - 1$
  - $\log(n+1) - 1 \le h \le n - 1$

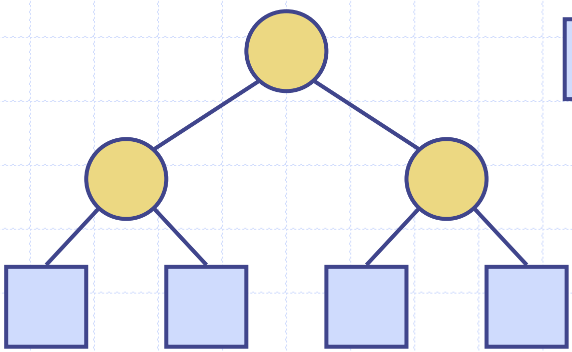


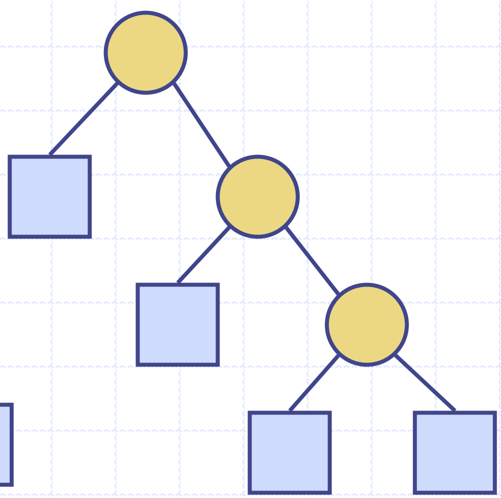


Properties of Proper Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height

- Properties:
  - $2h+1 \leq n \leq 2^{h+1}-1$
  - $h+1 \leq e \leq 2^h$
  - $\log(n+1)-1 \leq h \leq (n-1)/2$
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \geq \log_2 e$





BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position p.left()
  - position p.right()
- Update methods may be defined by data structures implementing the BinaryTree ADT

BinaryTree ADT

- There is a position class associated with the tree, to provide public access to nodes.
- Tree::Position supports member functions
  - element() Return the associated element.
  - left() Return the left child. Error if this is an external node.
  - right() Return the right child. Error if this is an external node.
  - parent() Return the parent. Error if this is the root.
  - isRoot() Return **true** if this is the root; else **false**.
  - isExternal() Return **true** if this is external; else **false**.

BinaryTree ADT

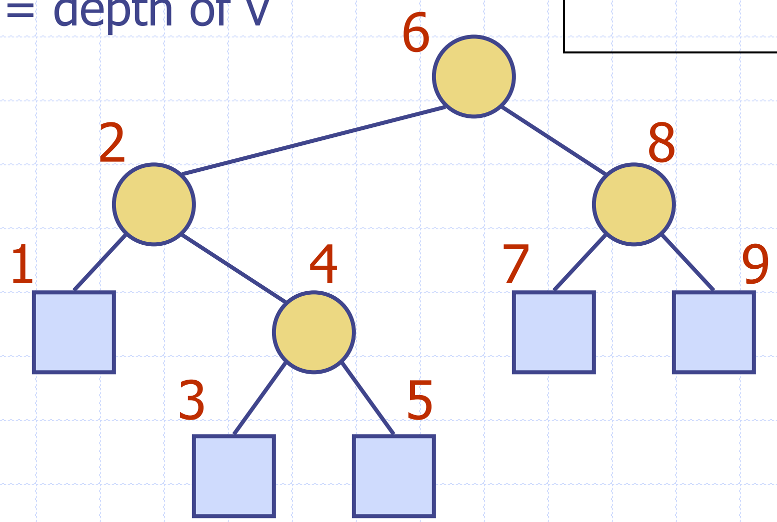
- The Tree itself supports member functions:
  - size() Return the number of nodes in the tree.
  - empty() Return **true** if the tree is empty; else **false**.
  - root() Return a position for the tree's root. Error if the tree is empty.
  - positions() Return a position list of all of the nodes of the tree.

- Note that we haven't defined update functions, so building a tree isn't possible right now. We'll do that later.

Inorder Traversal

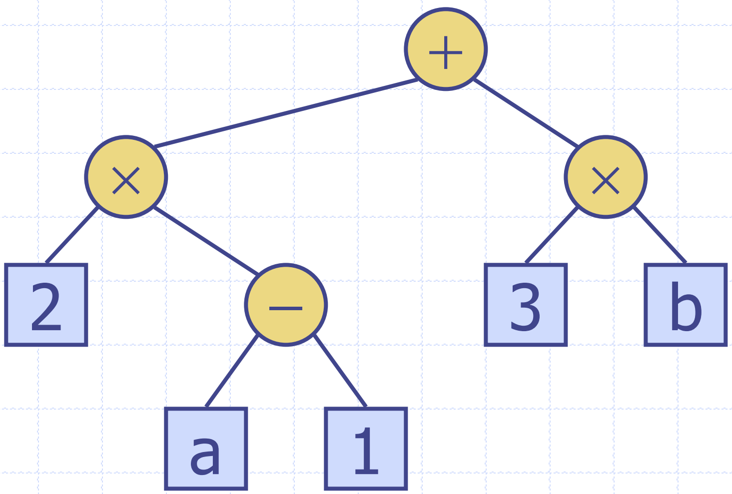
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
    if ¬ *v.isExternal*()
        inOrder(*v.left*())
    visit(*v*)
    if ¬ *v.isExternal*()
        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



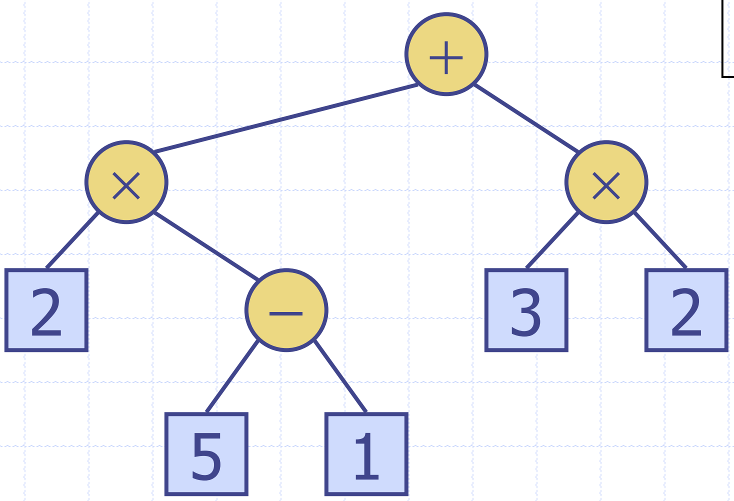
Algorithm *printExpression*(*v*)
    if ¬*v.isExternal*()
        print("(")
        printExpression(*v.left*())
    print(*v.element*())
    if ¬*v.isExternal*()
        printExpression(*v.right*())
        print (")")

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

Evaluate Arithmetic Expressions

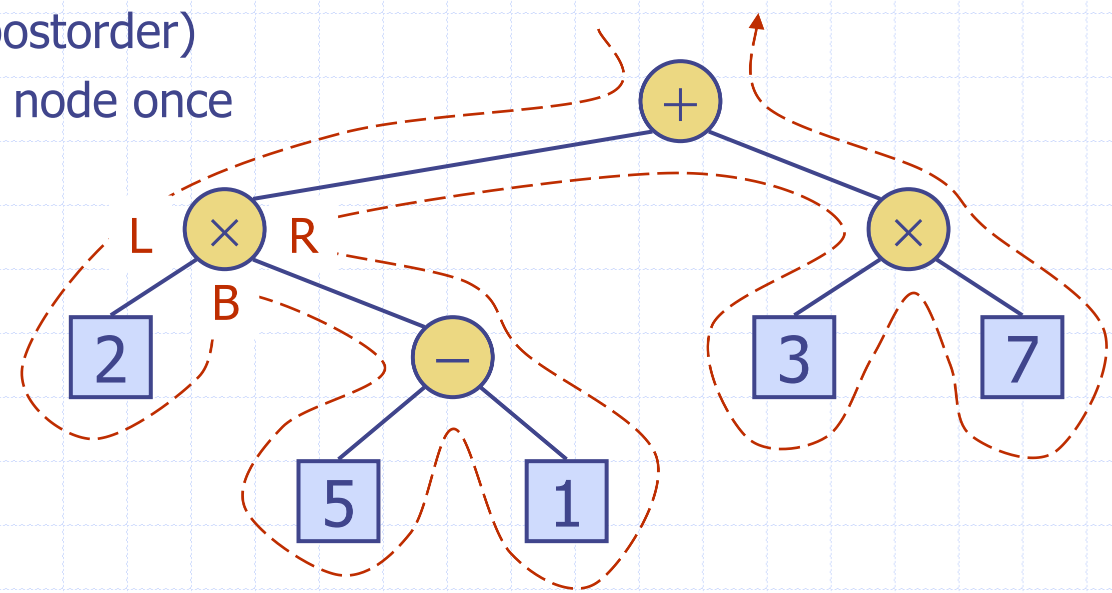
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm *evalExpr*(*v*)
    if *v.isExternal*()
        return *v.element*()
    else
        x ← *evalExpr*(*v.left*())
        y ← *evalExpr*(*v.right*())
        ◊ ← operator stored at *v*
        return *x* ◊ *y*

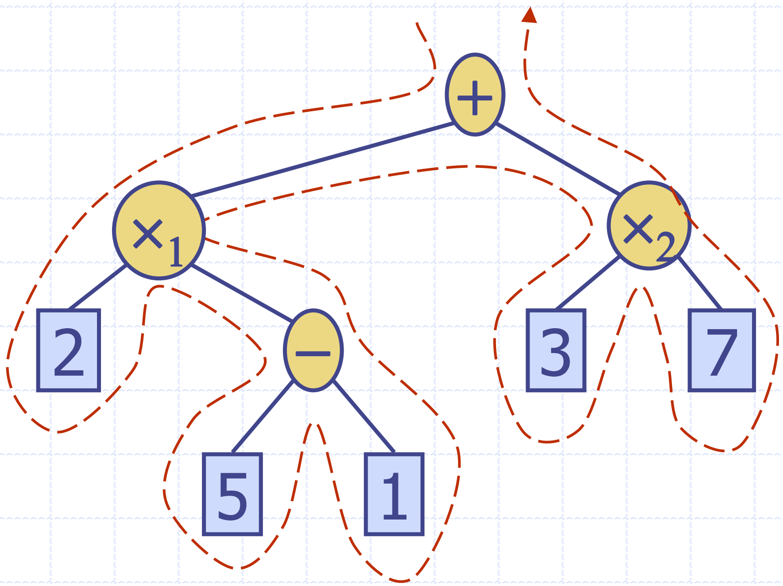


Euler Tour Traversal

- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each internal node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)
- Visit each external node once



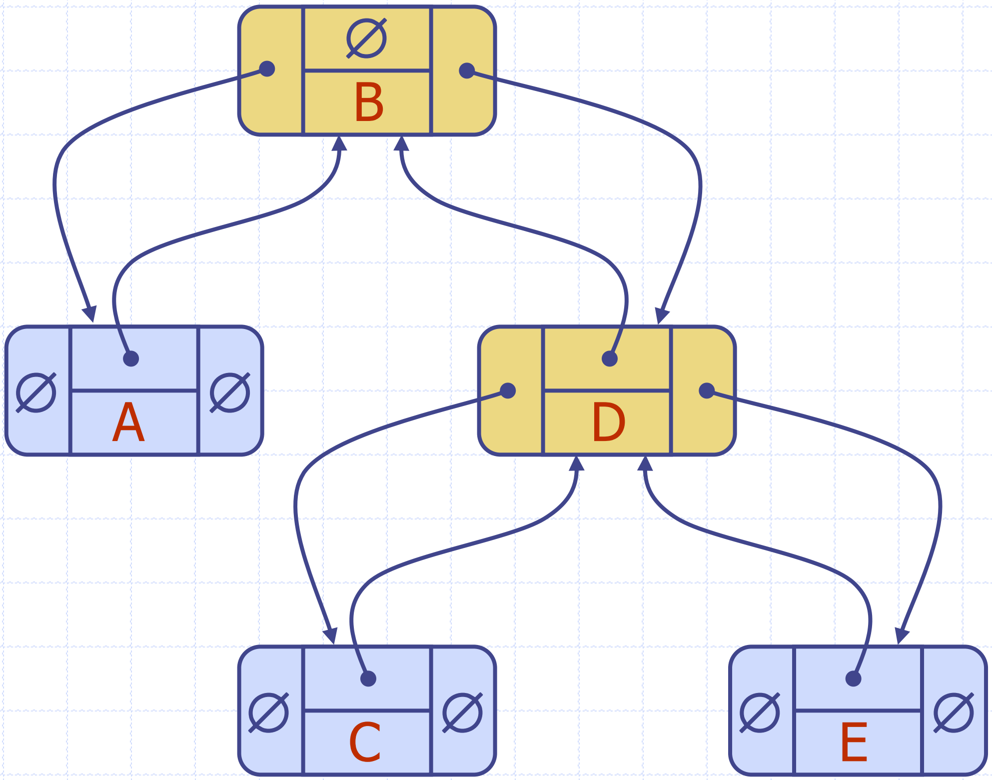
Euler Tour Traversal

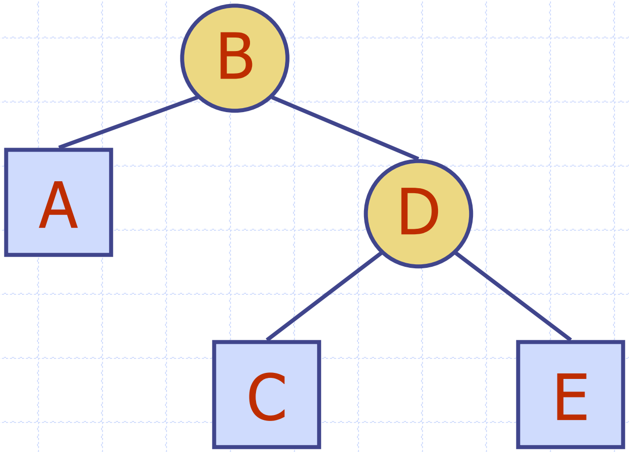


visitLeft(+)
visitLeft($\times_1$)
visit(2)
visitBottom($\times_1$)
visitLeft(–)
visit(5)
visitBottom(–)
visit(1)
visitRight(–)
visitRight($\times_1$)
visitBottom(+)
visitLeft($\times_2$)
visit(3)
visitBottom($\times_2$)
visit(7)
visitRight($\times_2$)
visitRight(+)

Linked Structure for Binary Trees

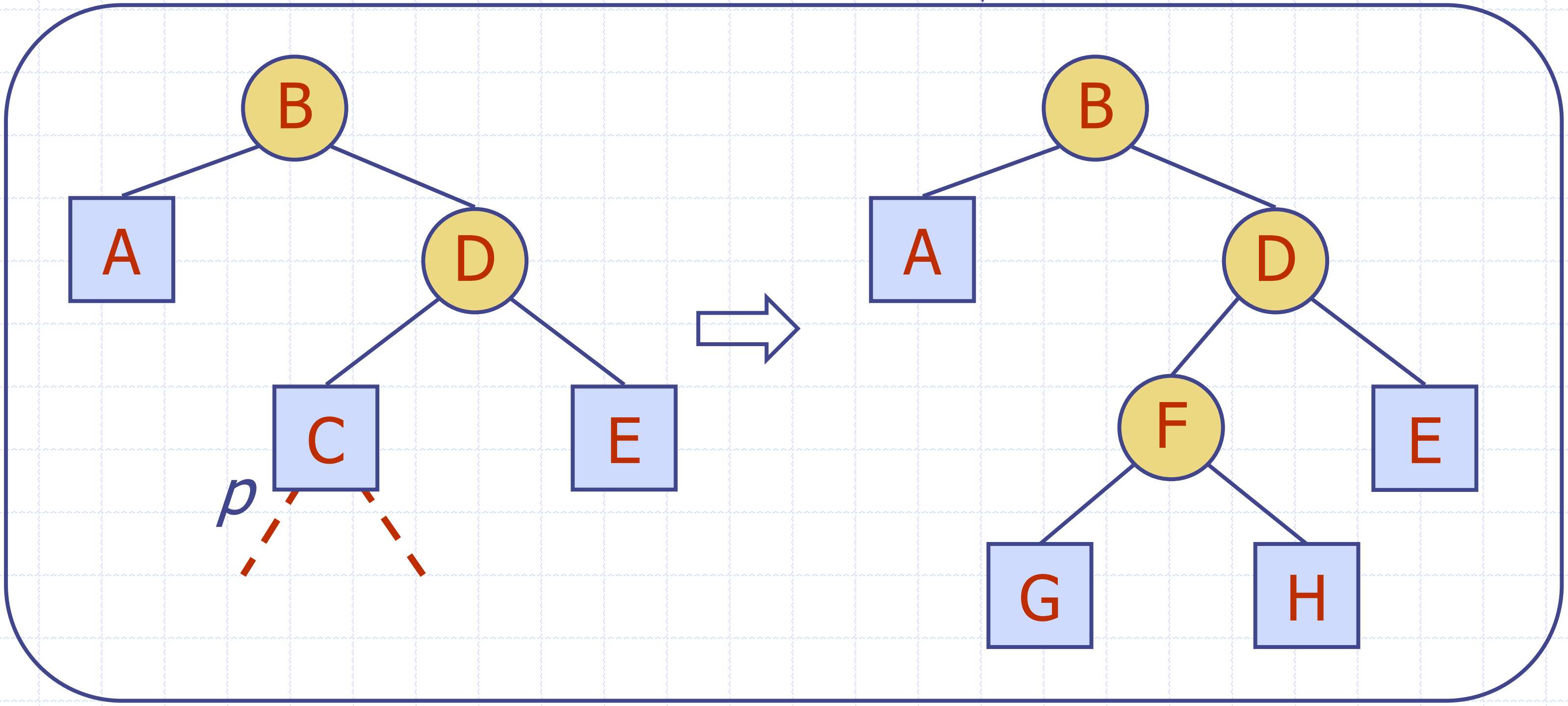
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





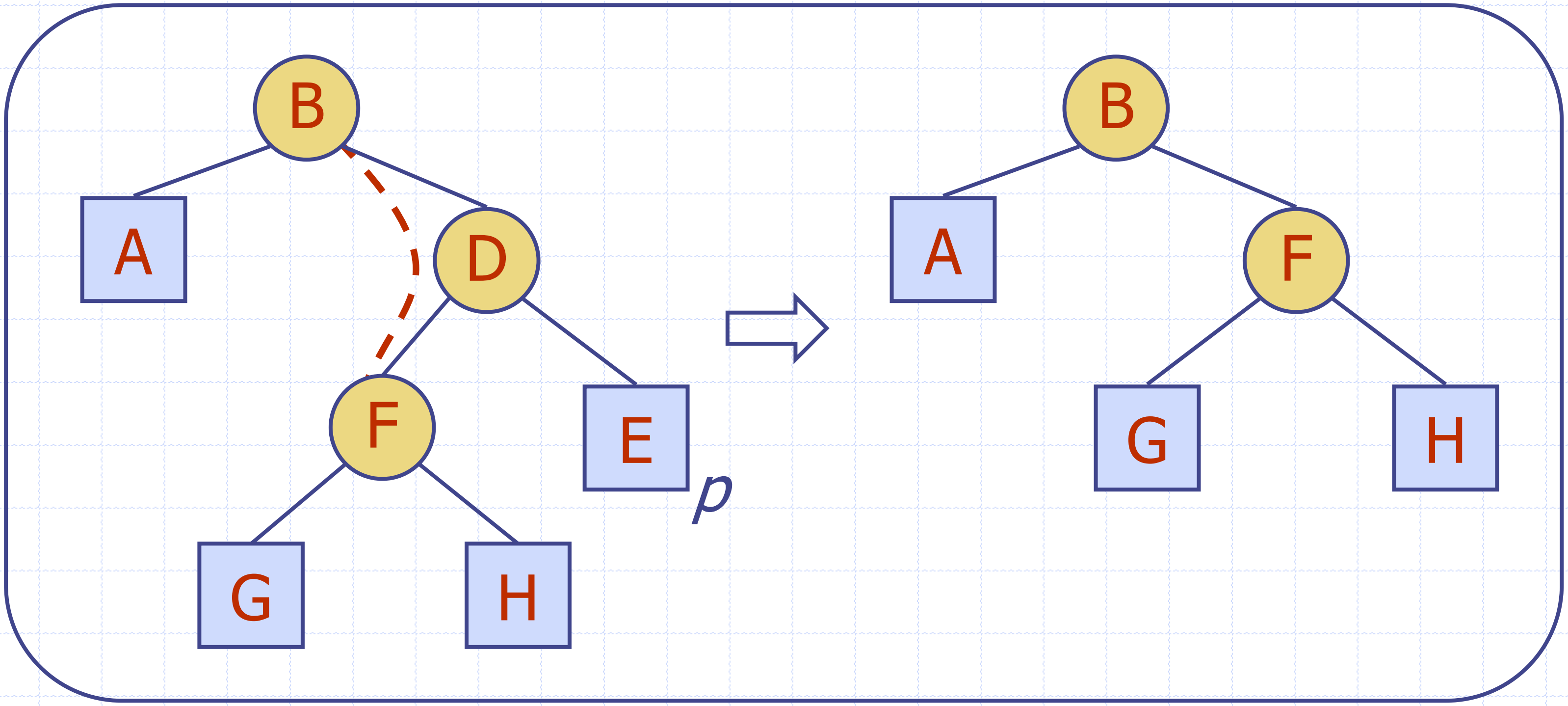
Proper Binary Tree Updates

- createRoot()
- expandExternal(p)
- removeAboveExternal(p)



Proper Binary Tree Updates

- removeAboveExternal(p)



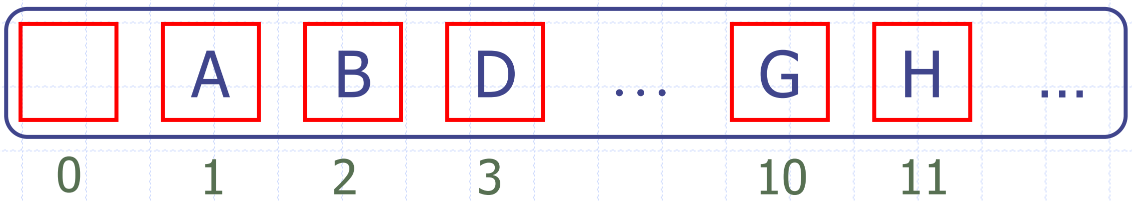
Binary Tree Performance Using Linked Structure

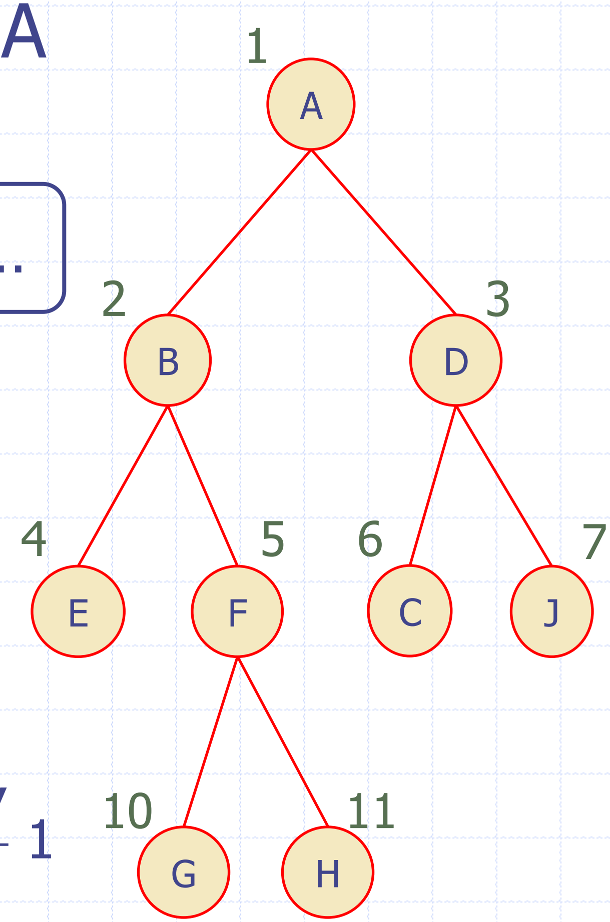
Operation	*Time*
left, right, parent, isExternal, isRoot	O(1)
size, empty	O(1)
root	O(1)
expandExternal, removeAboveExternal	O(1)
positions	O(n)

- Space usage is O(n).

Array-Based Representation of Binary Trees

- Nodes are stored in an array A





- Node v is stored at A[rank(v)]
  - rank(root) = 1
  - if node is the left child of parent(node), rank(node) = 2 · rank(parent(node))
  - if node is the right child of parent(node), rank(node) = 2 · rank(parent(node)) + 1