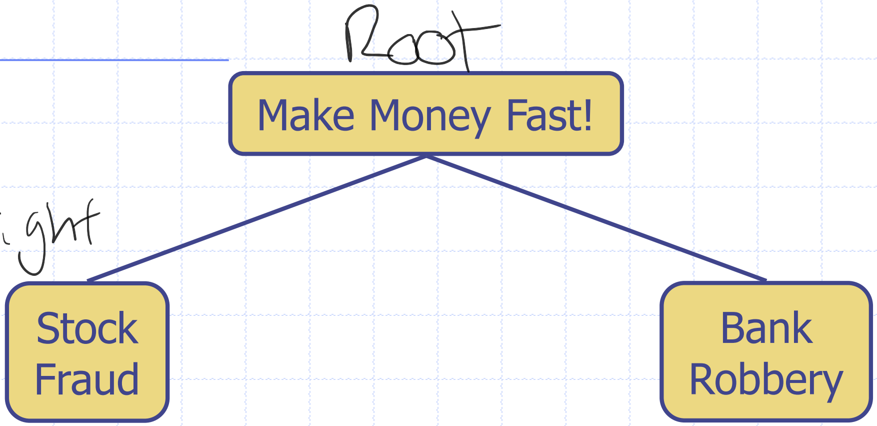
Array
↳linear
↳related to neighbors

Trees
↳only 1 parent
↳non-linear
↳heirarchy → levels

Binary Trees

↳only 2 branches per node

↳order → left to right

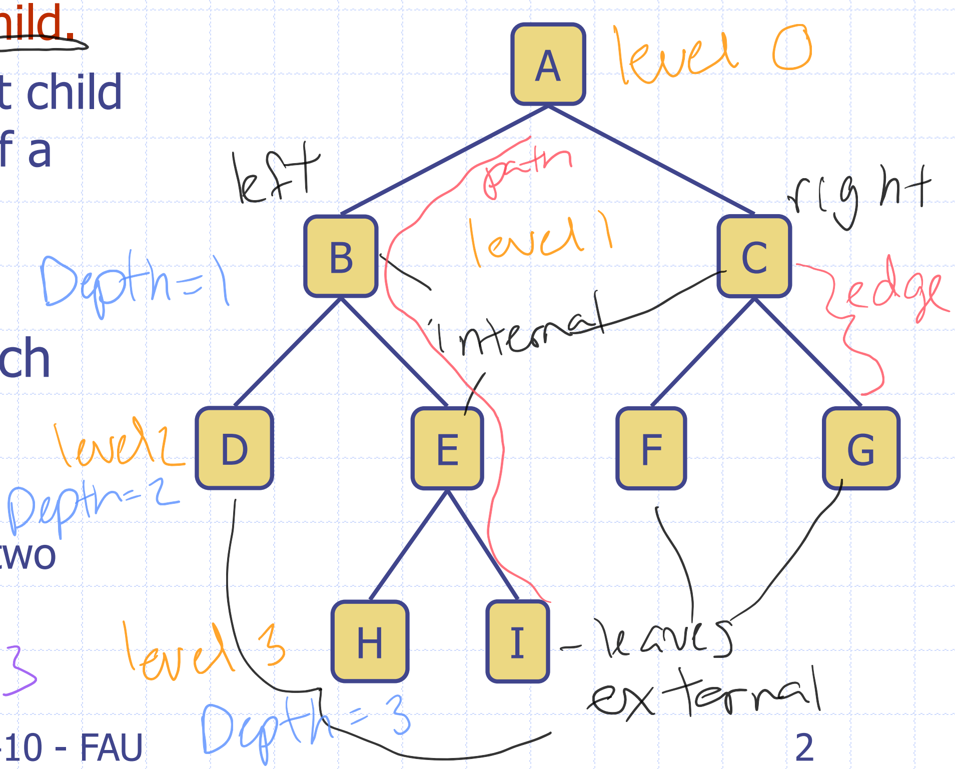


Sareh Taebi
COP3410 – Florida Atlantic University

Binary Trees

- A binary tree is a tree with the following properties:
  - Each node has at most two children
  - Each child node is labeled as being either a left child and right child.
  - A left child precedes the right child in the order of the children of a node.
- Proper(Full) binary tree: If each node has either zero or two children.
  - Every internal node has exactly two children.

- Applications:
  - arithmetic expressions
  - decision processes
  - searching

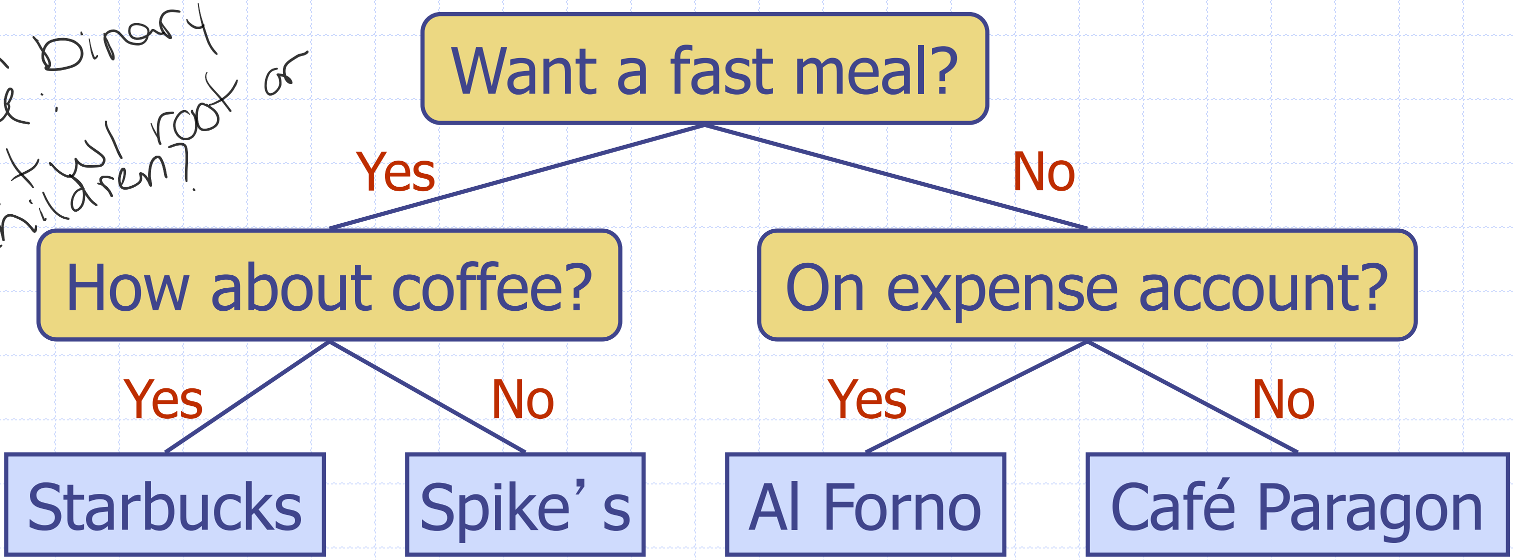


Decision Tree

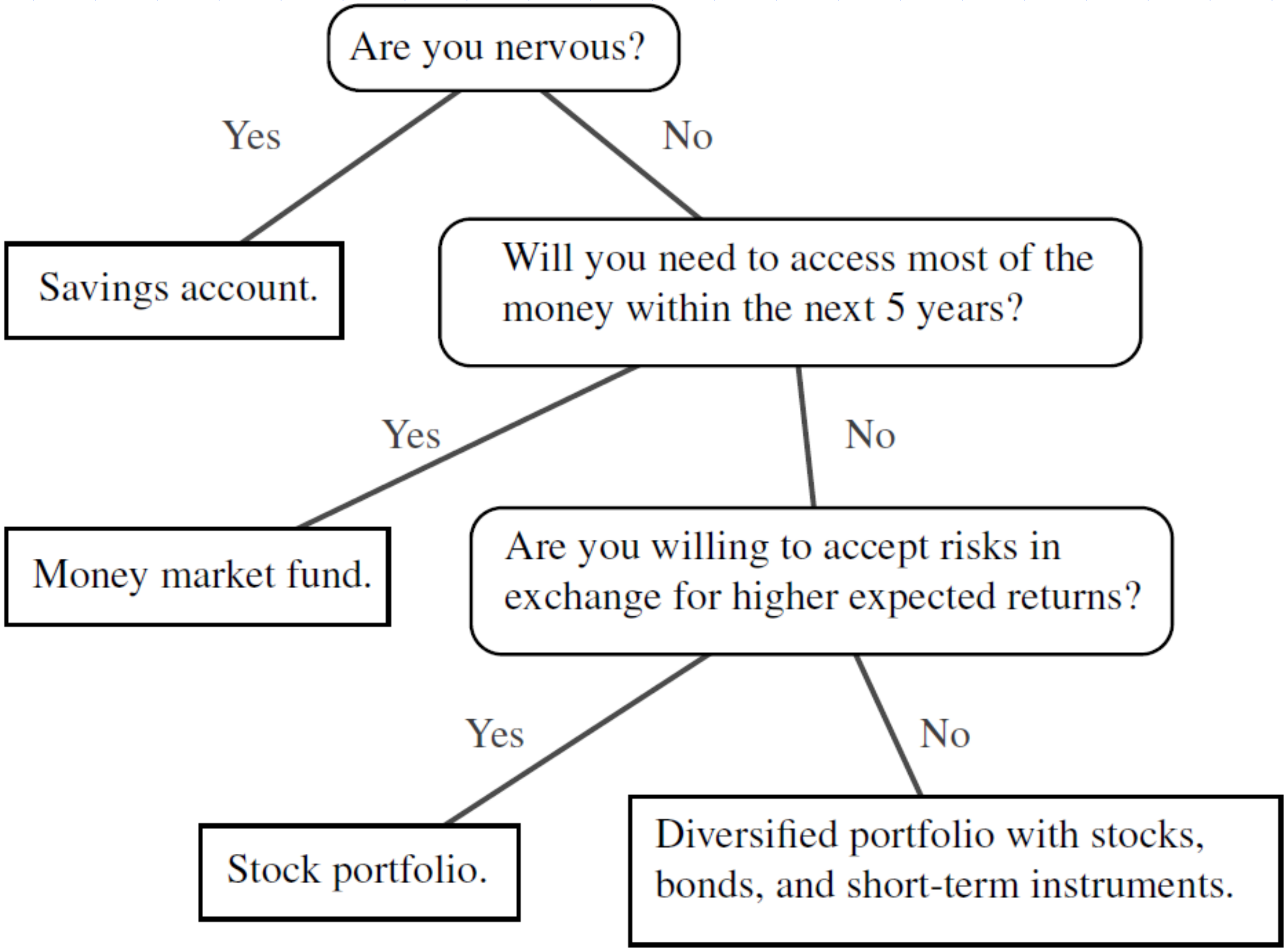
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

creating paths based on answers

search binary tree:
↳ start w/ root or children?



Decision Tree: A proper binary tree

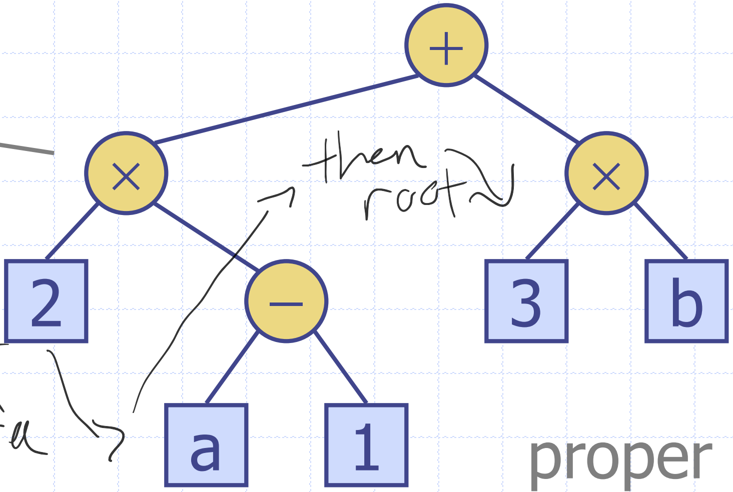

Are you nervous?
Yes
No
Savings account.
Will you need to access most of the money within the next 5 years?
Yes
No
Money market fund.
Are you willing to accept risks in exchange for higher expected returns?
Yes
No
Stock portfolio.
Diversified portfolio with stocks, bonds, and short-term instruments.

Arithmetic Expression Tree

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

① can you build this w/ binary tree?

② Would it be a proper binary tree?



binary ~~operands~~

Operators

then root

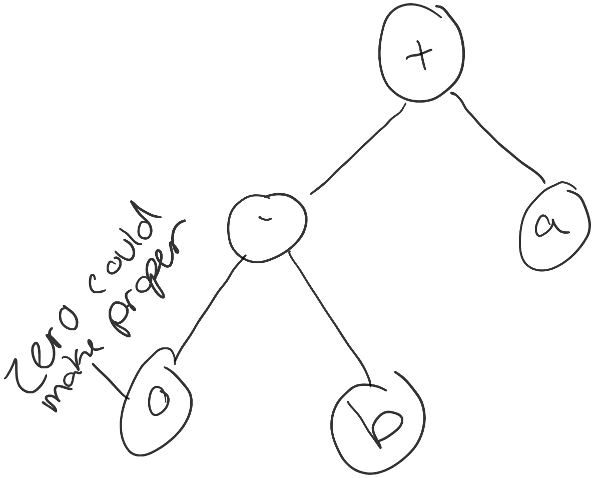
start w/ child

then sub tree

leaves → operands

proper binary tree

$-b + a$

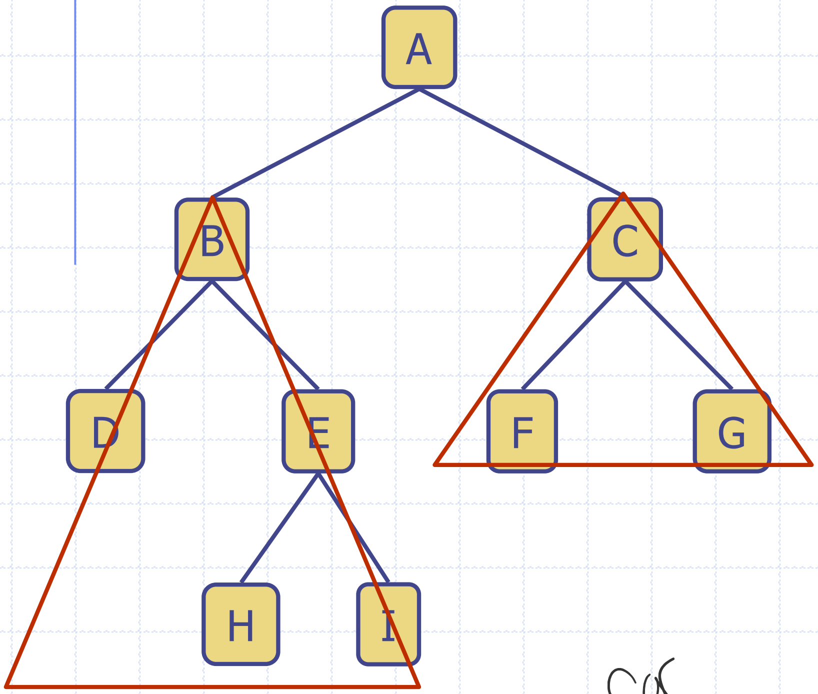


zero could make proper

↳ binary tree

↳ not proper

Recursive Binary Tree Definition



- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, called the root that stores an element
  - if a binary tree, called the left subtree
  - else a binary tree, called the right subtree

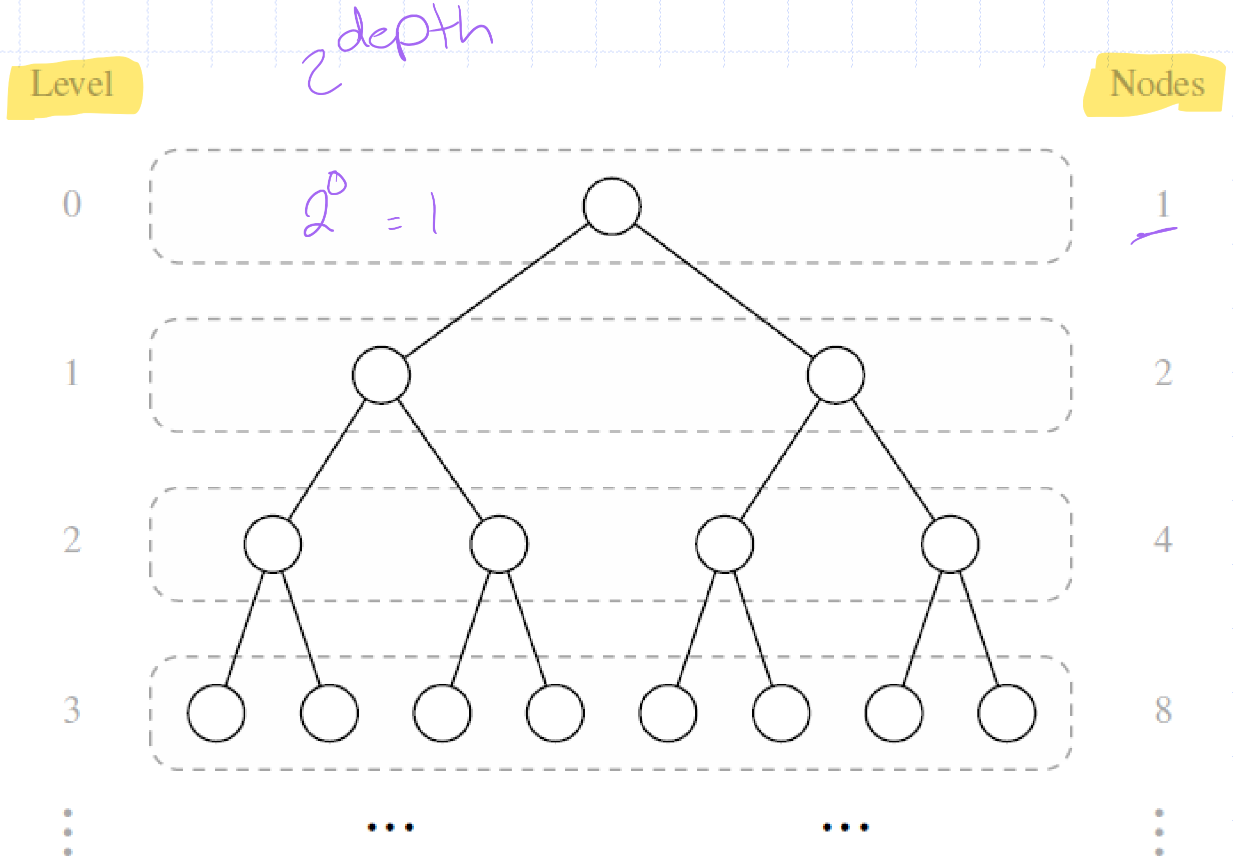
(possibly empty)

case for either empty

Level, Height and Depth

- **Level (d)**: Distance of the *node* from the root.
  - The root is always at level *zero*.
- **Height of the tree**: The maximum level in a tree determines its height.
- **Depth of the node**: The number of nodes along the path from the root to that node.

Levels and # of nodes in Binary Tree



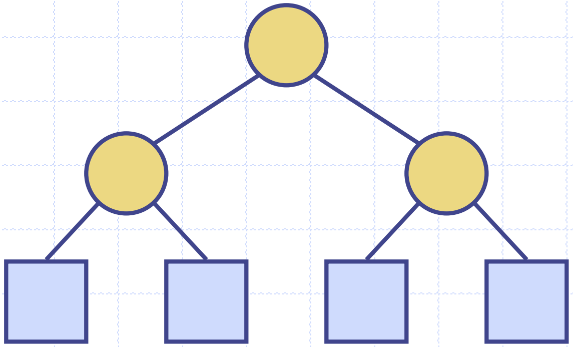
Properties of Binary Trees

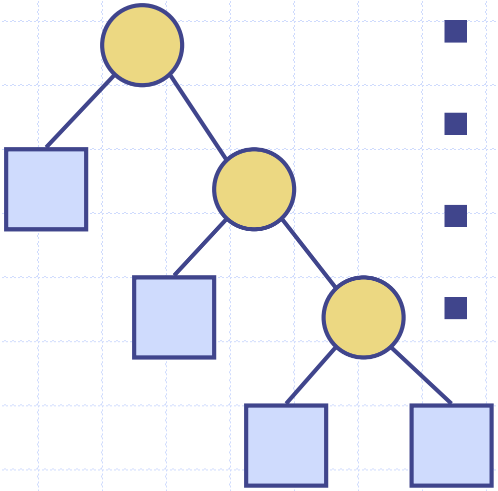
- If h is the height of the binary tree,
  - Max # of leaves = 2^h
  - Max # of nodes = $2^{h+1}-1$
- If a binary tree contain m nodes at level l, it contains at most $2m$ nodes at level $l+1$.
- A binary tree contains at most 2^l nodes at level l.
- The total # of edges in a proper binary tree with n nodes is $n+1$.

internal

Properties of Proper Binary Trees

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





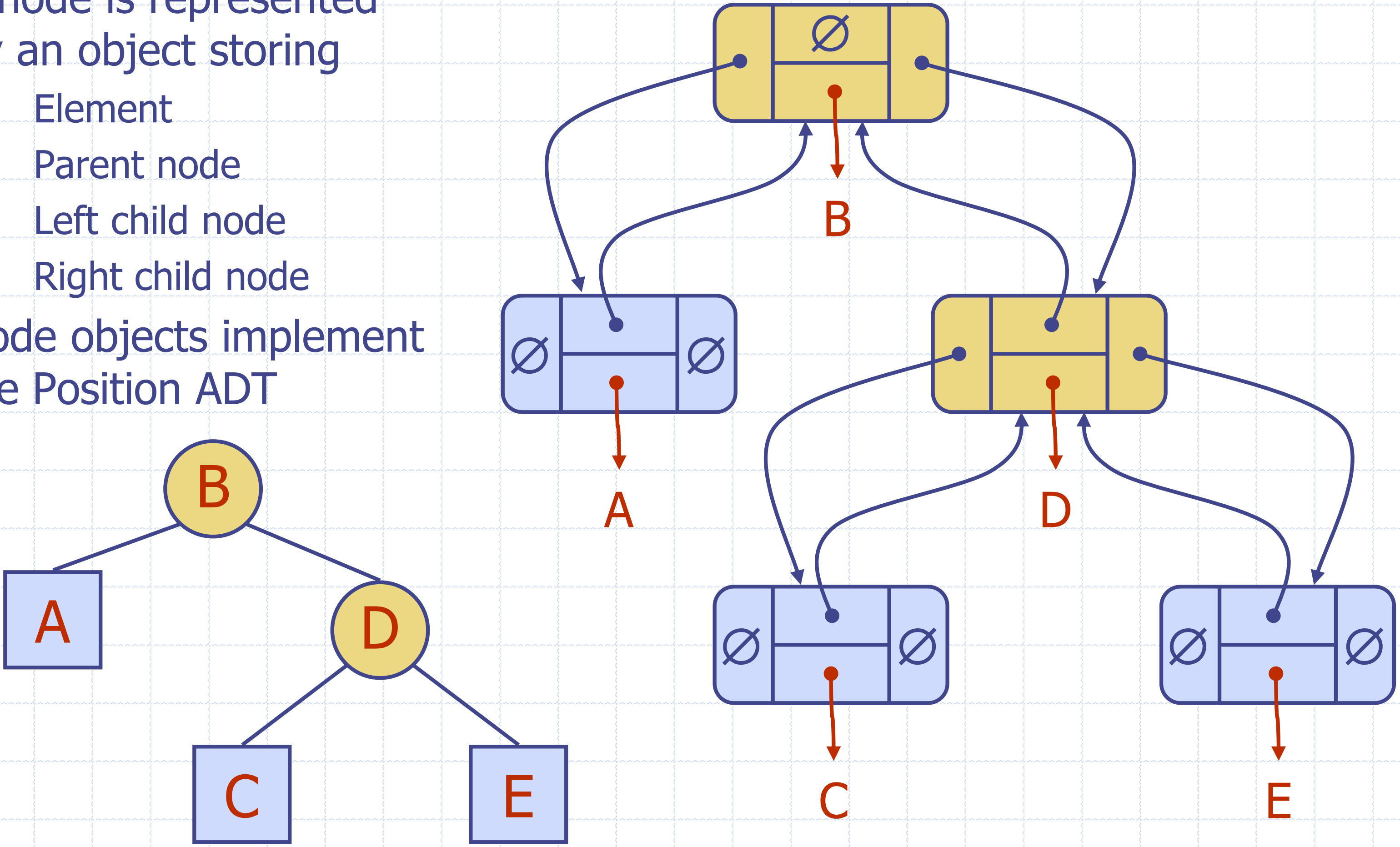
- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$ proper would be =
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$ $\log_2 e \leq h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

BinaryTree ADT

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

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

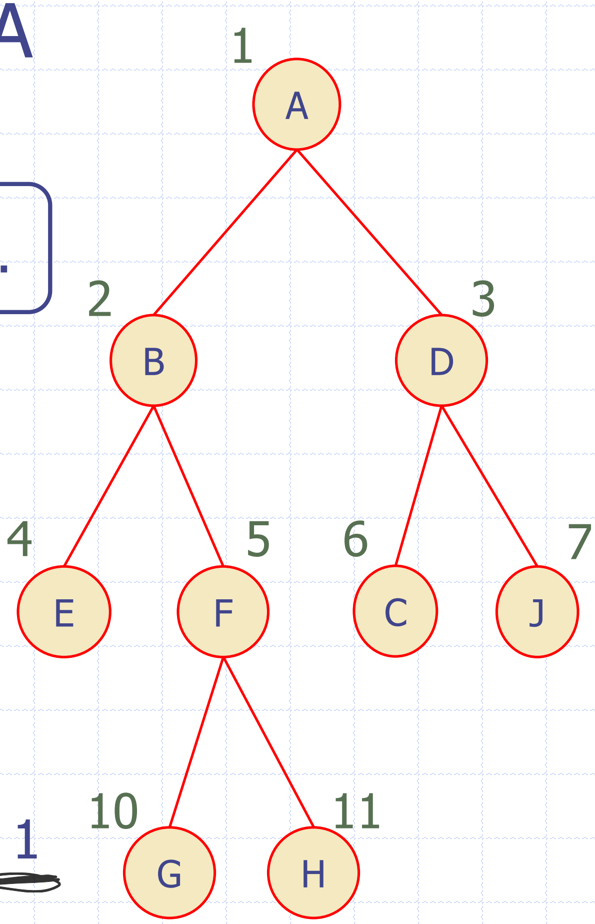


Array-Based Representation of Binary Trees

- ❑ Nodes are stored in an array A

	root	left	right		left of B	Right of B		
	A	B	D	…	G	H	…	
0	1	2	3		10	11		

- ❑ 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


1
A
2
B
3
D
4
E
5
F
6
C
7
J
10
G
11
H