CSC 225

Algorithms and Data Structures I

Rich Little
rlittle@uvic.ca
ECS 516

Trees

- A (*rooted*) *tree T* is a set of *nodes* in a *parent-child relationship* with the following properties:
    - *T* has a special node *r*, called the *root* of *T*
    - Each node *v* of *T* different from *r* has a unique *parent* node *u*

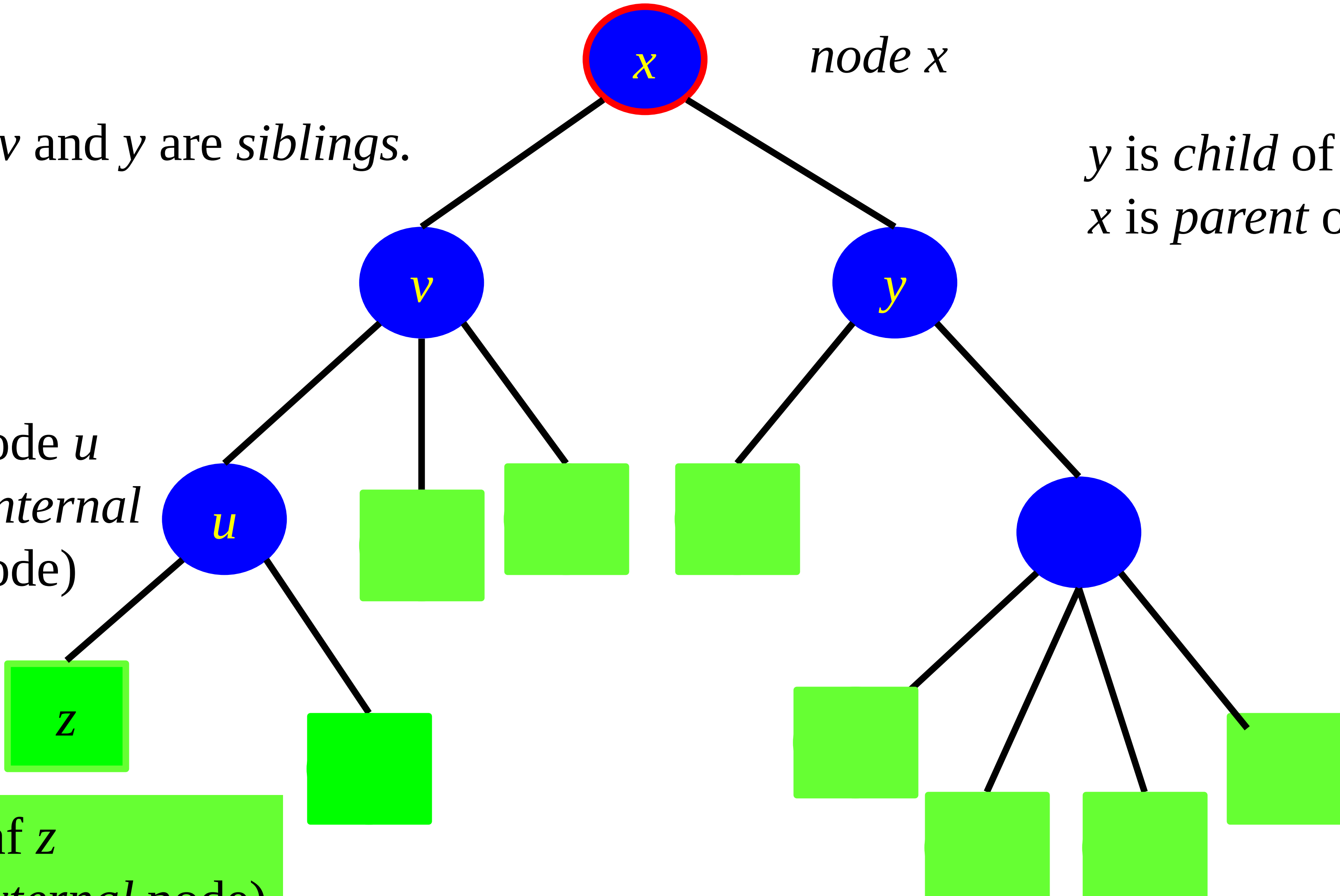
Trees

node x

v and *y* are *siblings*.

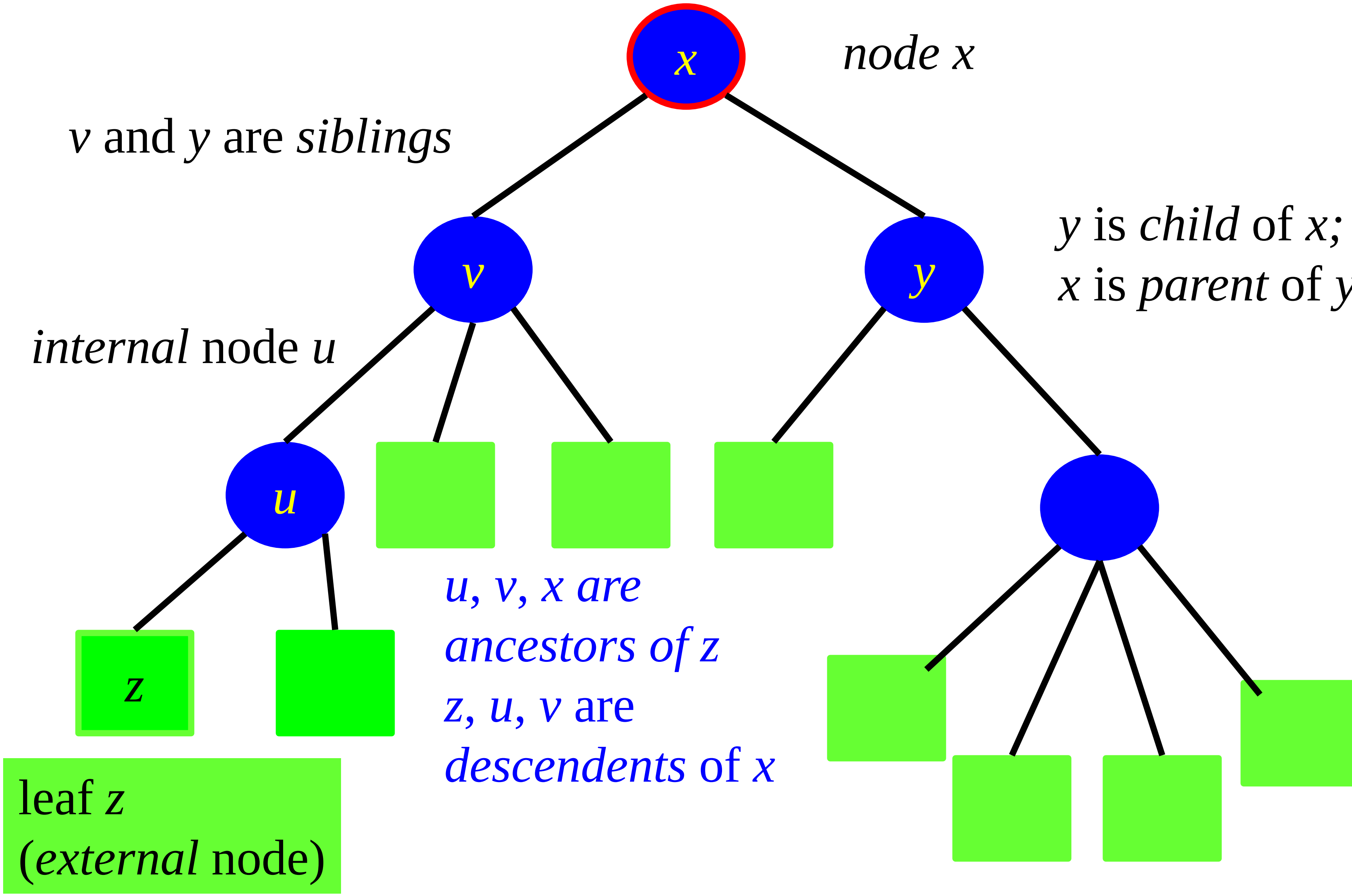
y is *child* of *x*;
x is *parent* of *y*.

node *u*
(*internal*
node)



leaf *z*
(*external* node)

Trees

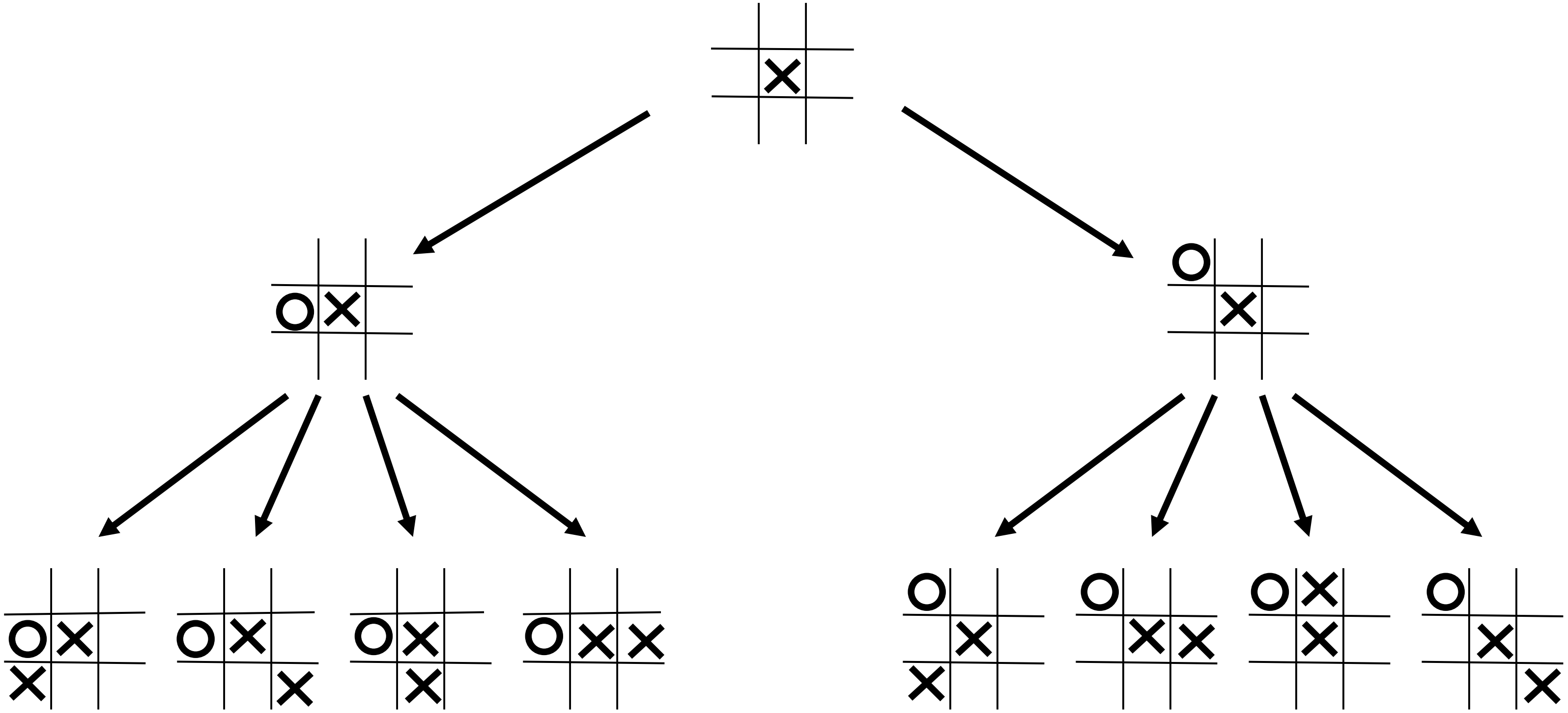

x
node x
v and y are siblings
v
y
y is child of x;
x is parent of y
internal node u
u
u, v, x are
ancestors of z
z, u, v are
descendents of x
z
leaf z
(external node)

Applications of Trees

- Phylogenetic trees
- Data structure (search trees)
- Search trees for exponential algorithms
- Visualization of algorithms (and tool for complexity analysis)
- Decision trees
- Parse trees
- Expression trees
- File systems
- Forests

Decision Trees

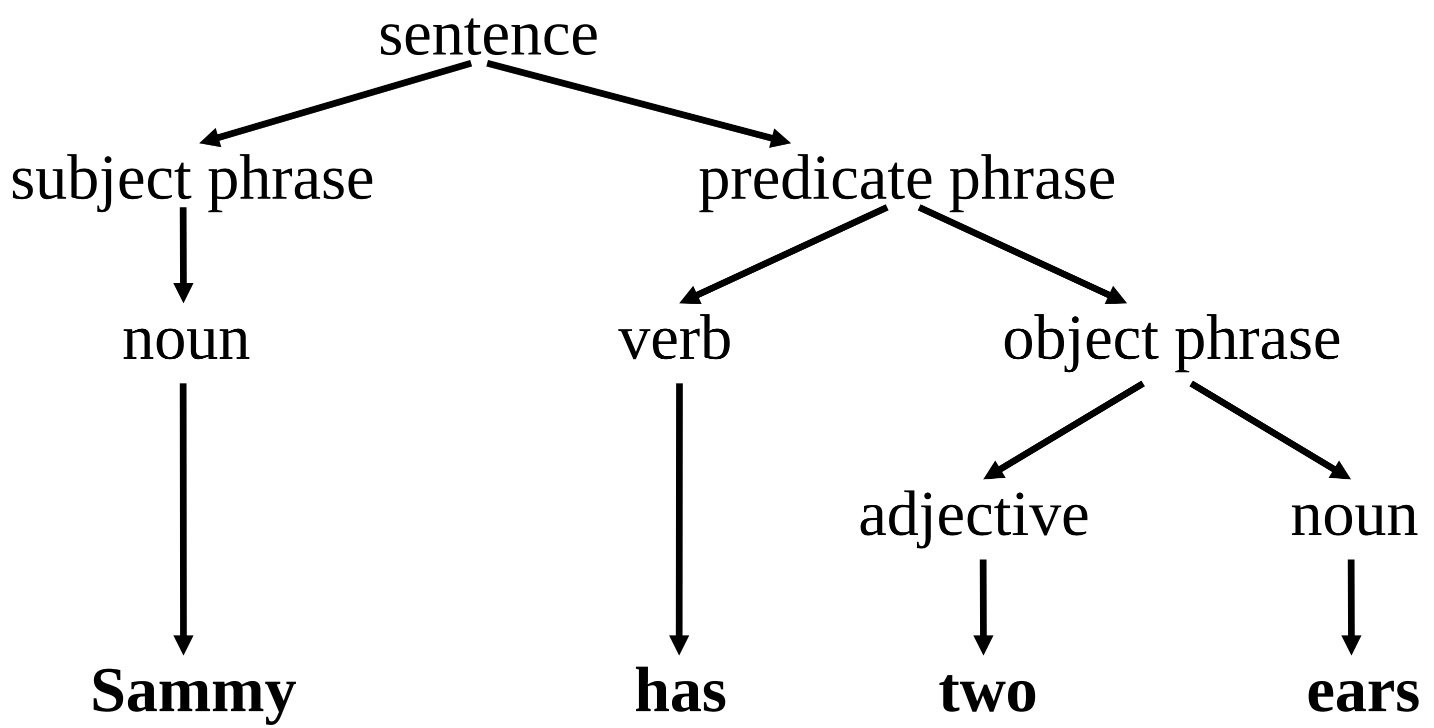
(after Gross & Yellen, 1999, p. 93)



The first three moves of tic-tac-toe

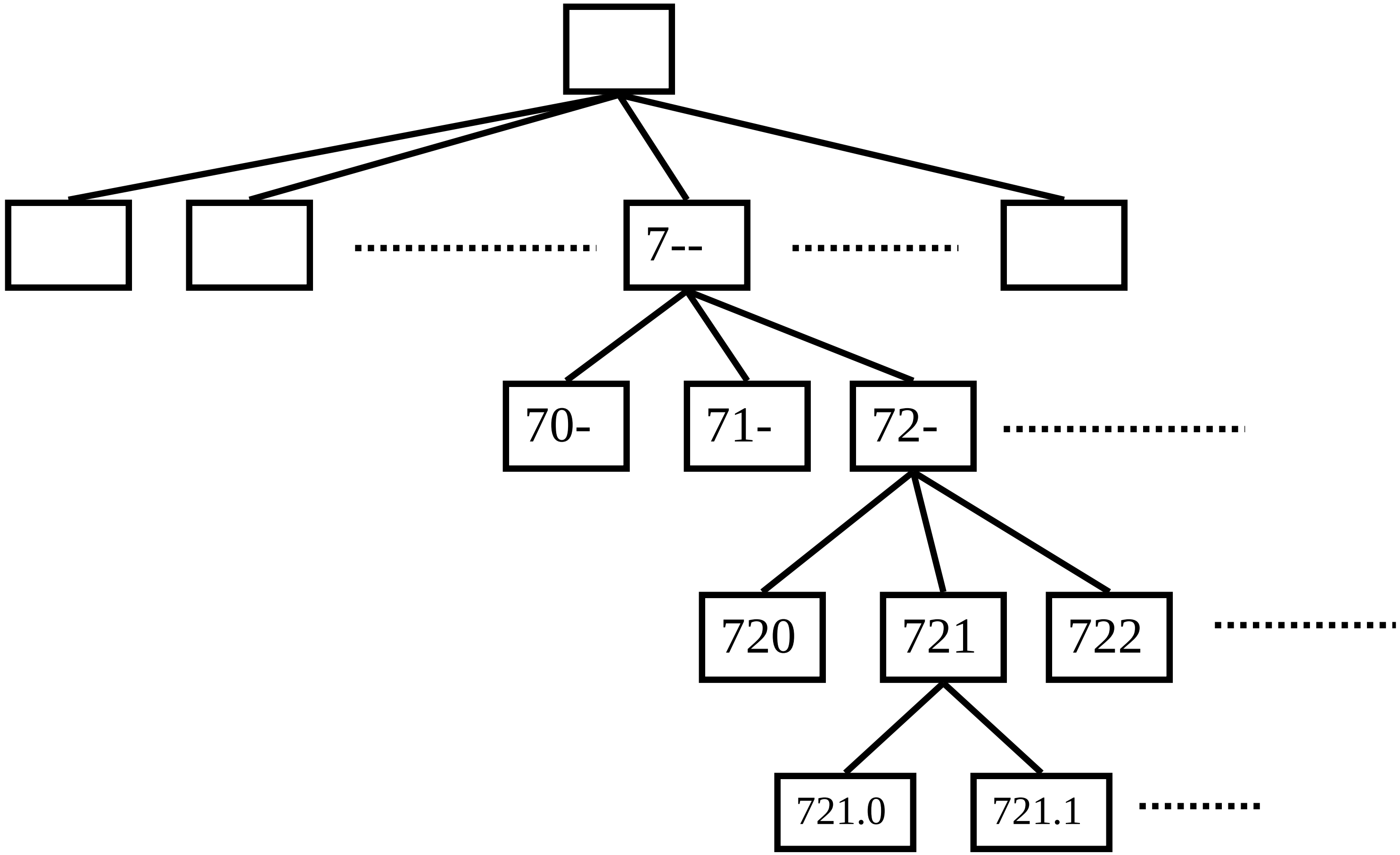
Sentence Parsing

(after Gross & Yellen, 1999, p. 93)



Data Organization: DDCS for libraries

(after Gross & Yellen, 1999, p. 93)



DDDC = *Dewey Decimal Classification System*

Tree ADT

- Stores *elements* at positions, which are defined relative to their neighbour positions (i.e., parents, children, siblings)
- The *positions* in a tree are its nodes
- Supported methods by a node/position object:
  - **root():** returns the root of the tree
  - **parent(*v*):** returns the parent of node *v;* an error occurs if *v* is root
  - **children(*v*):** returns an iterator to enumerate the children of node *v*

ADT Tree …

- **isInternal(*v*):** Test whether node *v* is internal
- **isExternal(*v*):** Test whether node *v* is external (i.e., leaf)
- **isRoot(*v*):** Test whether node *v* is root.
- **size():** returns the number of nodes in the tree
- **elements():** returns an iterator of all the elements stored at nodes of the tree (i.e., pre-, in-, post-, level-order)
- **positions():** returns an iterator of all the positions of the tree (i.e., pre-, in-, post-, level-order)
- **swapElements(*v*,*w*):** Swap the elements stored at nodes *v* and *w*
- **replaceElements(*v*,*e*):** Replace the element stored at node *v* with *e* and return the original element stored at node *v*

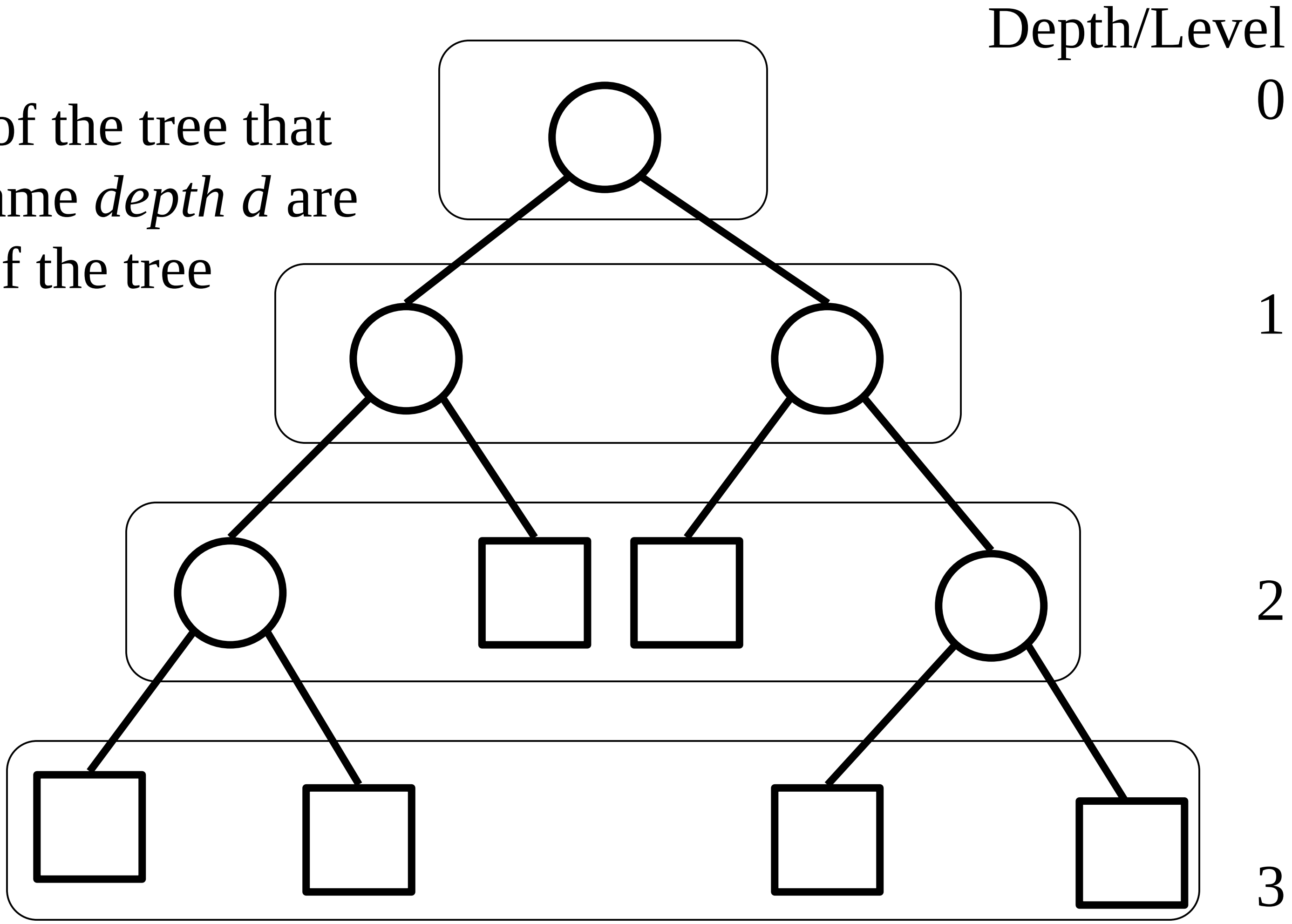
Depth of a Node

Definition: The *depth* of a node *v* is the number of ancestors of *v*. Recursively,

- If *v* is the root, then the depth of *v* is 0
- Otherwise, the depth of *v* is one plus the depth of the parent of *v*.

Depth and Levels in Trees

All nodes of the tree that have the same *depth d* are at *level d* of the tree



Tree Algorithms: depth

Algorithm depth(T,v):

***Input*:** Tree T, node v in T

***Output*:** the depth of v in T

if T.isRoot(v) **then**
    return 0
else
    return 1 + depth(T,T.parent(v))

Height of a Tree

Definition: The *height* of a tree T rooted at node v is (recursively) defined to be

- The height is 0 if v is a leaf node
- The height is equal to 1 plus the maximum height of any child of v, otherwise.

Tree Algorithms: height

Algorithm height(T,v):

***Input*:** Tree T, node v in T

***Output*:** the height of tree T rooted at node v

if T.isExternal(v) **then**

  return 0

else

  $h = 0$

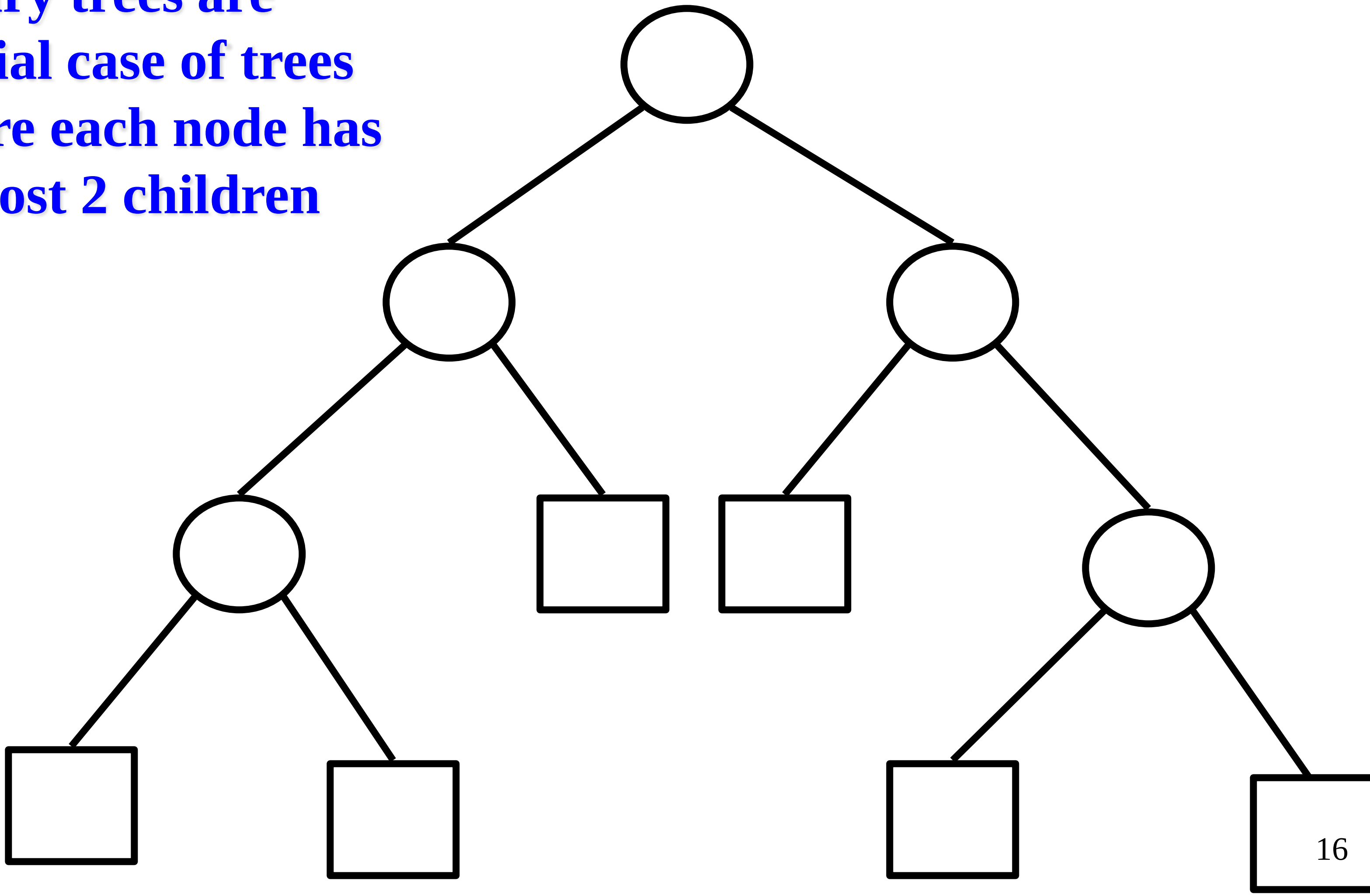
  for each $w \in T$.children(v) **do**

    $h = \max(h, \text{height}(T,w))$

  return $1 + h$

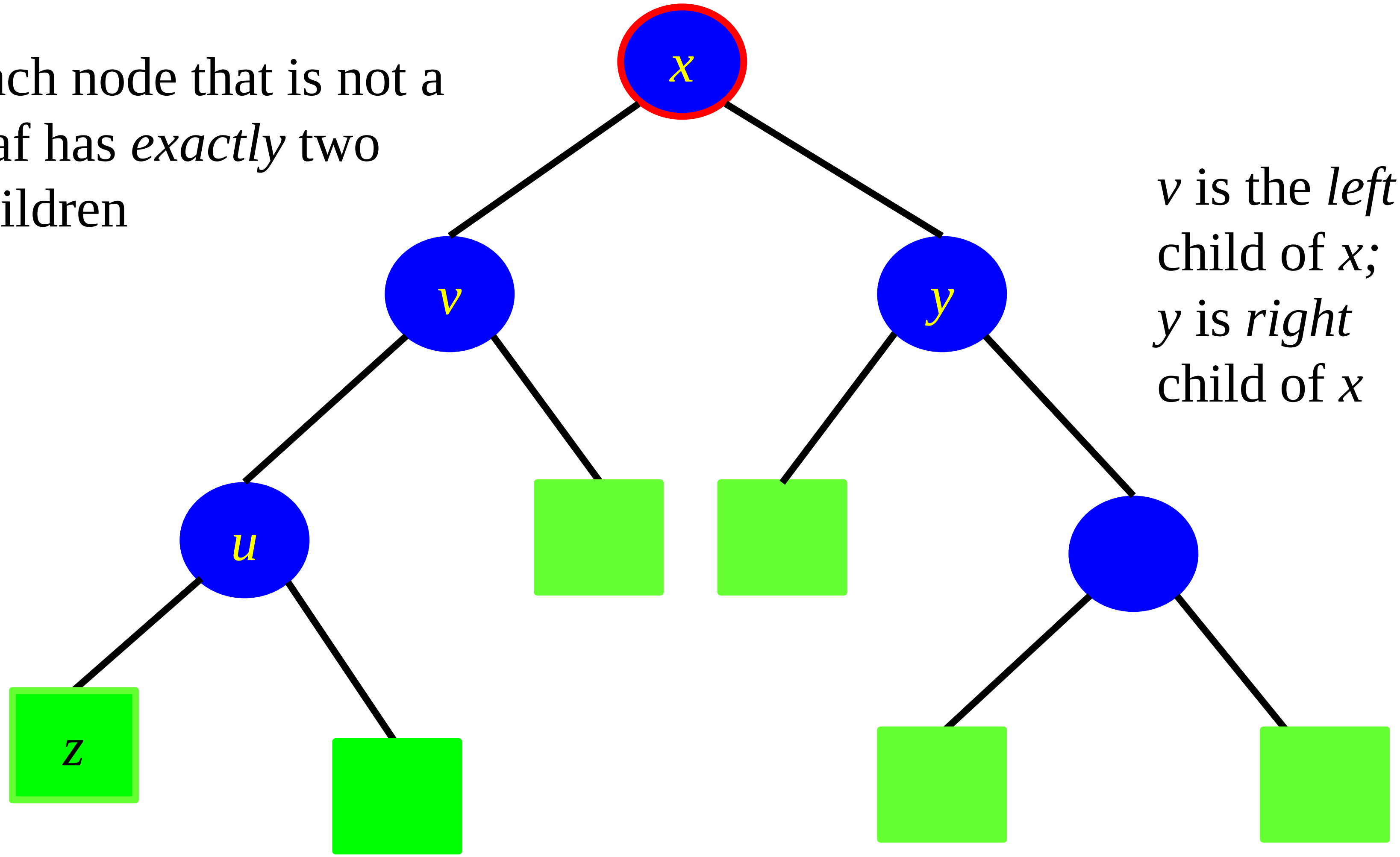
Binary Trees

Binary trees are special case of trees where each node has at most 2 children



Proper Binary Trees

Each node that is not a leaf has *exactly* two children



v is the *left* child of *x*; *y* is *right* child of *x*

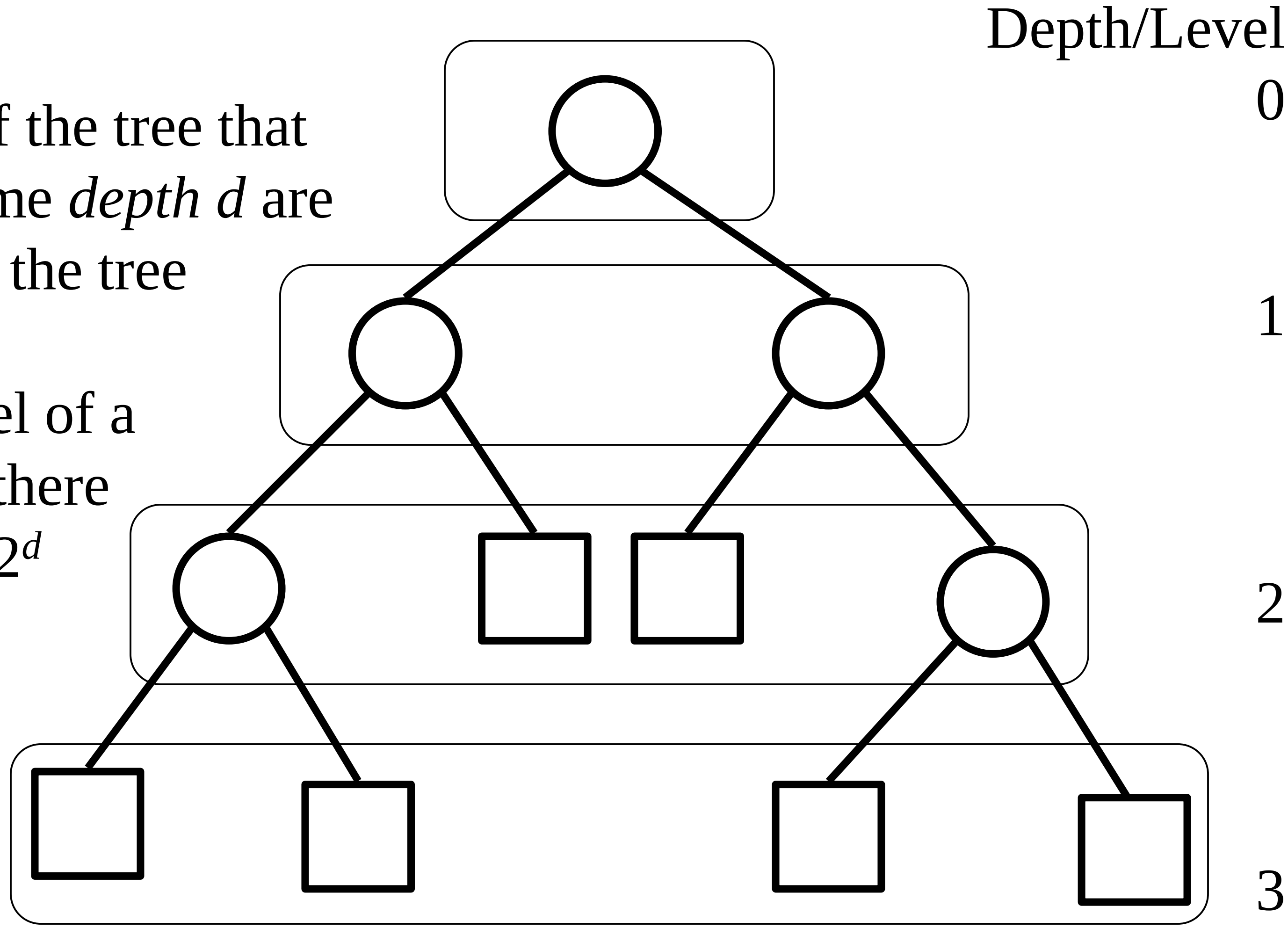
ADT Binary Tree

- Specialization of a proper binary tree ADT that supports the accessor methods
  - **leftChild(*v*):** returns the left child of *v;* an error occurs if *v* is a leaf.
  - **rightChild(*v*):** returns the right child of *v;* an error occurs if *v* is a leaf.
  - **sibling(*v*):** returns the sibling of *v;* an error occurs if *v* is the root.

Depth and Levels in Trees

All nodes of the tree that have the same *depth d* are at *level d* of the tree

At each level of a *binary* tree there are at most 2^d nodes



Properties of Binary Trees

- **Theorem:** Let T be a binary tree with n nodes and let h denote the height of T, then
  1. The number of *leaves* in T is at least $h + 1$ and at most 2^h
  2. The number of internal nodes in T is at least h and at most $2^h - 1$
  3. $2h + 1 \leq n \leq 2^{h+1} - 1$
  4. $\log(n+1) - 1 \leq h \leq (n-1)/2$
  5. # of leaves = 1 + # of internal nodes

Proof of 5

- Number of external nodes, e, is 1 more than the number of internal nodes, i. That is, $e = i + 1$.

Proof of 5 - continued

- Number of external nodes, e, is 1 more than the number of internal nodes, i. That is, $e = i + 1$.