CSCI 136
Data Structures & Advanced Programming

Lecture 17

Fall 2018

Instructor: Bills

Administrative Details

- Lab 7: PostScript
  - No partners this week
  - Review before lab; come to lab with design doc
  - Check out the javadoc pages for the 3 provided classes
    - Token – A wrapper for semantic PS elements,
    - Reader – An iterator to produce a stream of Tokens from standard input or a List of Tokens,
    - SymbolTable – A dictionary with String keys and Token values: For user-defined names

Last Time: Iterators & Ordered Structures

- Iterators Continued
- Iterating over Iterators
- Ordered Structures: Preview

Today: Ordered Structures

- Lab 7 Discussion
- Ordered Structures:
  - OrderedVector
  - OrderedList
- Trees: Introduction

Lab 7: PostScript Interpreter

- PostScript is a *stack-based* programming language
  - designed for vector graphics & printing
- Lab 7: Implement a small portion of a PS interpreter
  - Read a stream of "tokens"
  - Evaluate expressions using a stack
  - Allow for creation of variables (and procedures!) using a symbol table
- Provided:
  - Reader, Token, and SymbolTable class
  - You write an interpreter class
- Try out GhostScript: unix command: gs
  - Type gs –dNODISPLAY to suppress graphics window

Lab 7: Concept Overview

- Basic input unit: the *token*: There are multiple types
  - Number, Boolean, Symbol, Procedure (sorry, no Strings for us)
  - Implemented with class Token
- A PostScript program is a sequence of tokens
  - Tokens are processed as received
    - Numbers, booleans, procedures go on stack
    - A symbol should
      - Be put on stack (if preceded by /), or
      - Cause an operation to be performed if it is a built-in symbol (add, pstack, …), or
      - Cause its value to be looked up in symbol table and appropriate action taken
  - The SymbolTable class provides a symbol dictionry
  - The Reader class provides an iterator for producing a stream of tokens
    - Stream can come from standard input, a single Token, or a List of Tokens
- Your job: Write code to carry out the processing
  - Driven by a method (you write) *interpret(Reader r)*

Lab 7: Suggested Approach

1. Read Lab handout and description in text carefully
2. Read the Javadoc pages for the 3 provided classes: Using these classes well will help you a great deal!
3. Develop a plan. Here are some starting steps
    1. Write your interpret method so that it just reads a token stream from standard input and prints out each token.
    2. Handle numbers, booleans, and pstack/pop operators
    3. Follow the steps in the text in order
4. Debug as you go, use gs program to clarify expected behavior

Ordered Structures

- Until now, we have not required a specific ordering to the data stored in our structures
  - If we wanted the data ordered/sorted, we had to do it ourselves
- We often want to keep data ordered
  - Allows for faster searching
  - Easier data mining - easy to find best, worst, and median values, as well as rank (relative position)

Ordering Structures

- The key to establishing order is being able to compare objects
- We already know how to compare two objects…how?
- Comparators and `compare(T a, T b)`
- Comparable interface and `compareTo(T that)`
- Two means to an end: which should we use?

BOTH!

Ordered Vectors

- We want to create a Vector that is always sorted
  - When new elements are added, they are inserted into correct position
  - We still need the standard set of Vector methods
    - add, remove, contains, size, iterator, …
- Two choices
  - Extend Vector (as we did in sorting lab)
  - Create new class
    - Allows for more focused interface
    - Can have a Vector as an instance variable
    - Avoid corrupting order by controlled access to Vector
- We will implement a new class (OrderedVector)
  - Start with Comparables
  - Generalize to use Comparators instead of Comparables

OrderedVector Methods

```
public class OrderedVector<E extends Comparable<E>>
  implements OrderedStructure<E> {
   protected Vector<E> data;

   public OrderedVector() {
        data = new Vector<E>();
   }

   public void add(E value) {
        int pos = locate(value);
        data.add(pos, value);
   }

   protected int locate(E value) {
   //use modified binary search to find position of value
   //if not found, returns position where add should occur
   //uses iterative version of binary search (see text)
   }
```

OrderedVector Methods

```
public boolean contains(E value) {
    int pos = locate(value);
    return pos < size() && data.get(pos).equals(value);
}

public Object remove (E value) {
    if (contains(value)) {
          int pos = locate(value);
          return data.remove(pos);
    }
    else return null;
}
```

Performance:

- add - O(n)
- contains - O(log n)
- remove - O(n)

Adding Flexibility with Comparators

- We would like to be able to allow ordered structures to use different orders
- Idea: Add constructor that has a Comparator parameter
- Q: How does structure know whether to use the Comparator or the Comparable ordering?
- A: The NaturalComparator class....

An Aside: Natural Comparators

- NaturalComparators bridge the gap between Comparators and Comparables

```
class NaturalComparator<E extends Comparable<E>>
implements Comparator<E> {
    public int compare(E a, E b) {
          return a.compareTo(b);
    }
}
```

Generalizing OrderedVector

```
public class OrderedVector<E extends Comparable<E>>
   implements OrderedStructure<E> {
   protected Vector<E> data;
   protected Comparator<E> comp;

   public OrderedVector() {
        data = new Vector<E>();
        this.comp = new NaturalComparator<E>();
   }

   public OrderedVector(Comparator<E> comp) {
        data = new Vector<E>();
        this.comp = comp;
   }

   protected int locate(E value) {
        //use modified binary search to find position of value
        //return position
        //use comp.compare instead of compareTo
   }

   //rest stays same…
```

Ordered Lists

- Similar to OrderedVector
- Can't easily use SinglyLinkedList like OrderedVector used Vector (Why?)
- So, we just build a SinglyLinkedList-like structure
- add, contains, remove runtime?
  - All O(n)…why?

OrderedList Methods

```
public class OrderedList<E extends Comparable<E>>
    extends AbstractStructure<E> implements
OrderedStructure<E> {

    protected Node<E> data; // smallest value
    protected int count;    // size of list
    protected Comparator<? super E> ordering;

    public OrderedList() {
        this(new NaturalComparator<E>());
    }
    public OrderedList(Comparator<? super E> ordering){
        this.ordering = ordering;
        clear();
    }
```

OrderedList Methods

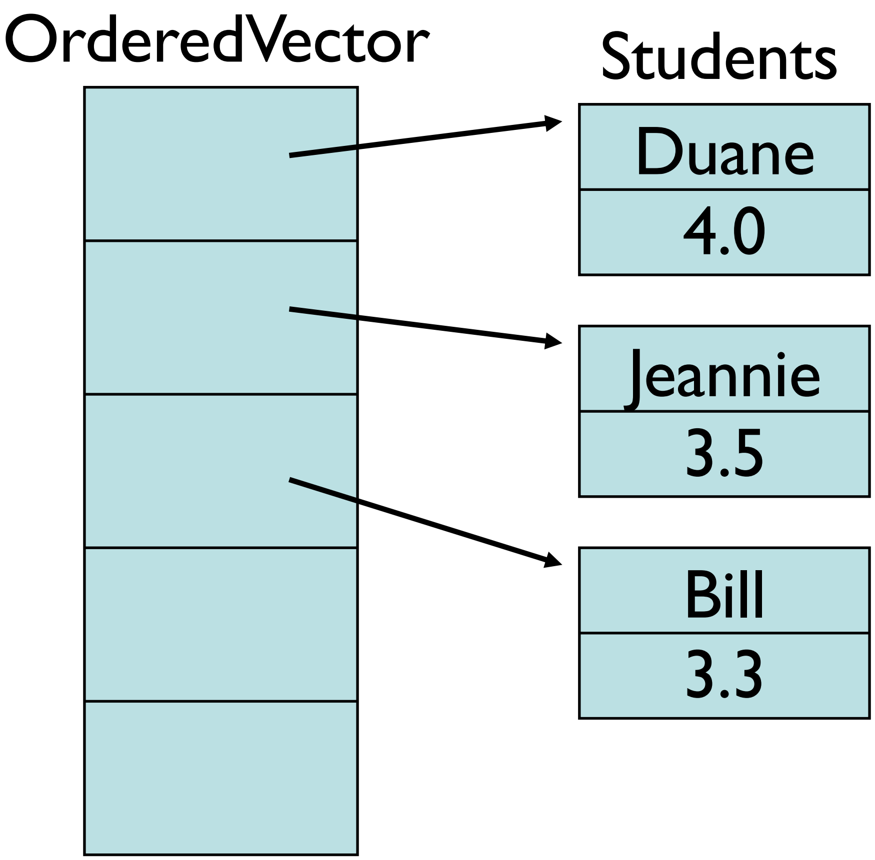
```
public void clear() {
      data = null;
      count = 0;
}
public boolean contains(E value) {
   Node<E> finger = data; // target

   while ((finger != null) &&
      ordering.compare(finger.value(),value)<0)

      finger = finger.next();

  return finger!=null && value.equals(finger.value());
}
```

What Could Go Wrong?



- Students compared to each other by GPA
- Suppose next semester I get a 3.7 and Jeannie gets a 3.3

What’s the problem?

- We have to recompute GPAs each semester
- What happens if the values are allowed to change?
- We may need to resort vector
  - But since this isn’t part of the interface, it may be forgotten
- Options:
  - Avoid changing values in OrderedStructures
  - Incorporate an update method that repositions element
  - Incorporate a resort method
    - This invites adding a “setComparator” method....
  - Always update a value by removing and re-adding

Type Safety & Generic Types

- Question: Since String extends Object, does List<String> extend List<Object>?
  - I.e., can I say List<Object> = new List<String>()?
- No. It would compromise the type system:

```
List<String> slist = new List<String>();
List<Object> olist = slist;    // If this were possible
olist.add(new Object());       // This would be bad!
```

- It generates a compiler error.
- On the other hand…

```
String[] sa = {“I”, “love”, “java”, “!”};
Object[] oa = sa;
oa[1] = new Object()); // This would be bad!
```

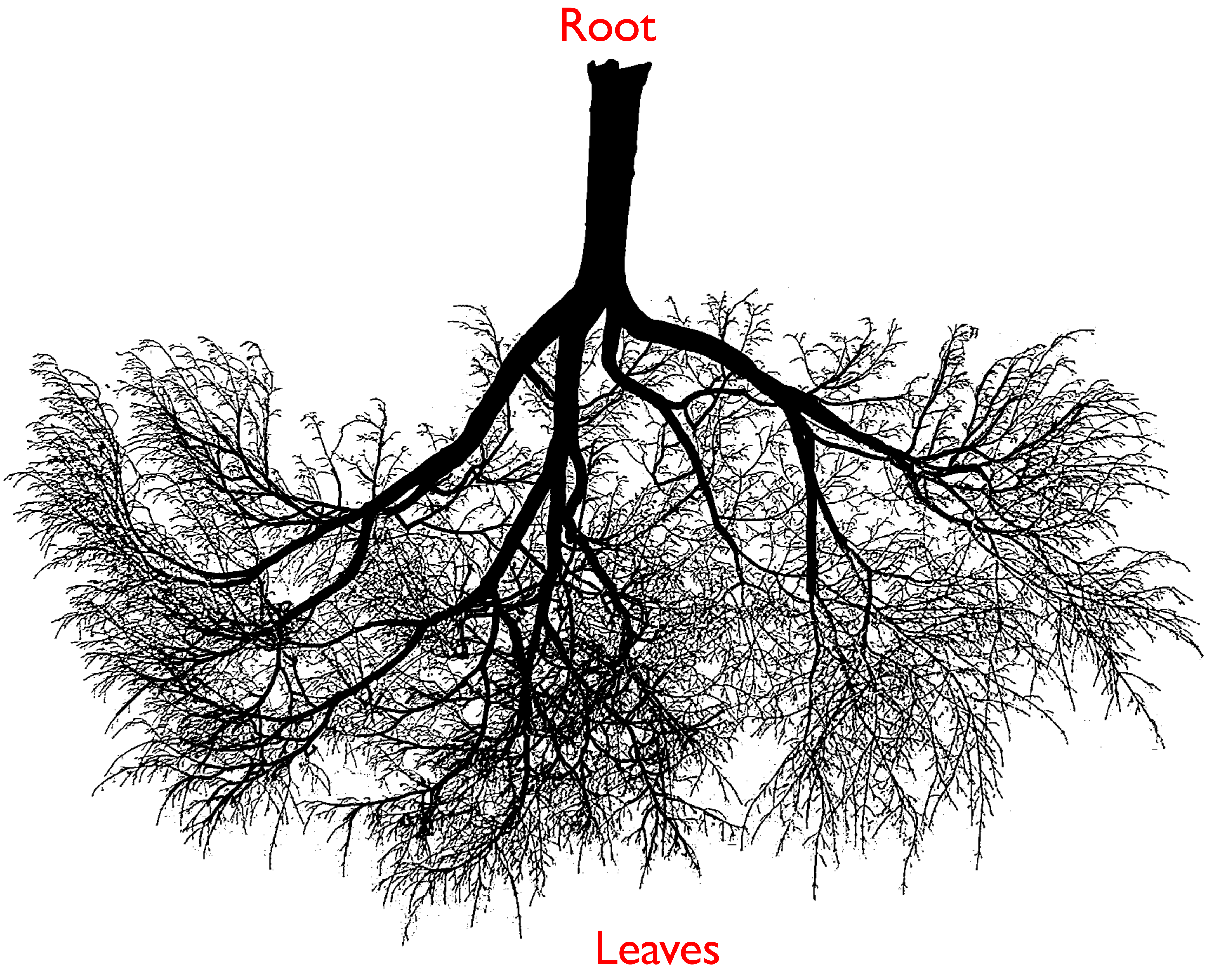
- …actually compiles
  - But causes a run-time error!

Introducing Trees

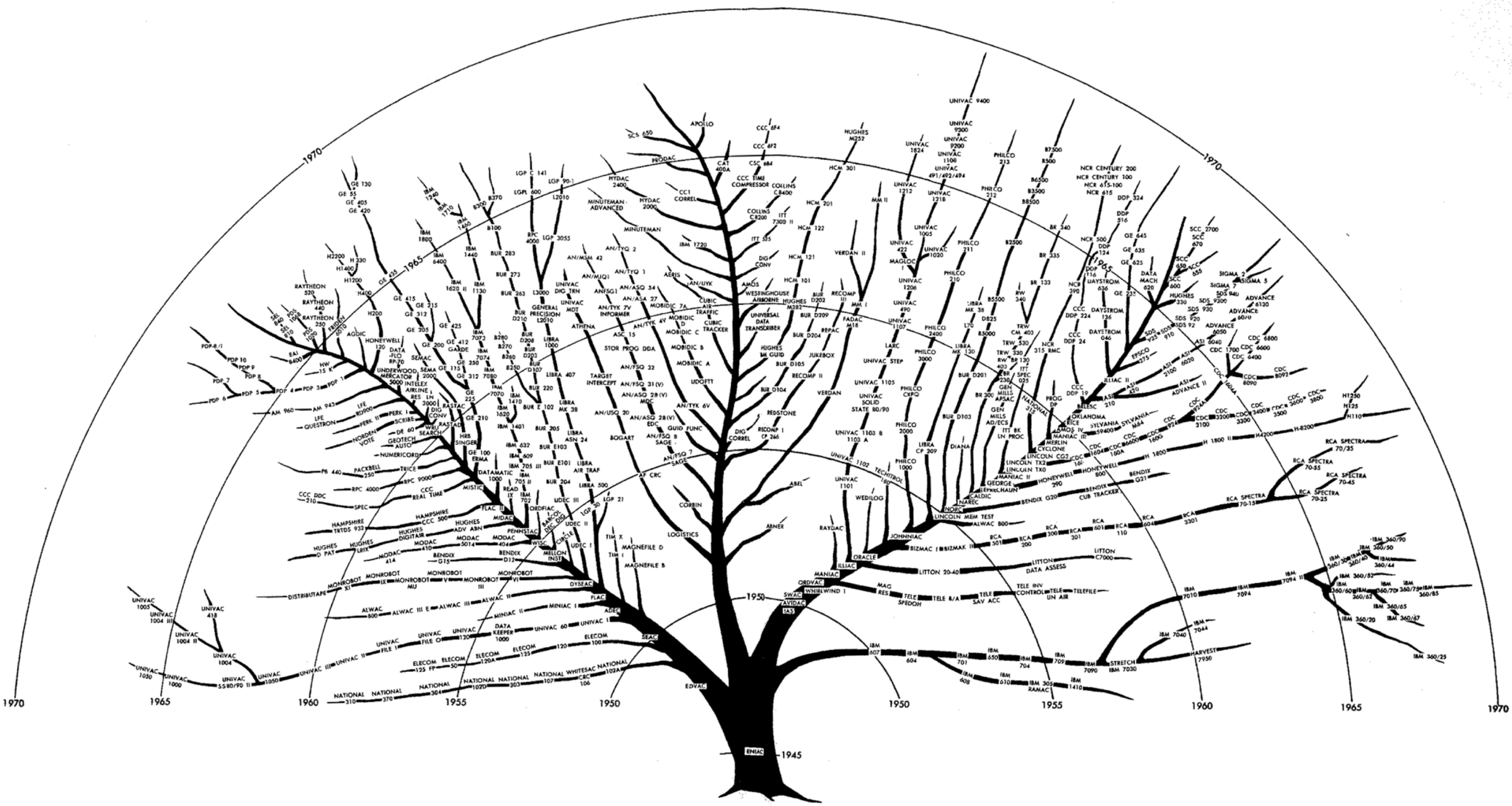
- Our structures have had a linear organization
  - Stacks, queues
  - Even ordered vectors, ordered lists, arrays, vectors, lists are visualized linearly
- By linear we essentially mean that each element has at most one successor and at most one predecessor…

Branching Out: Trees

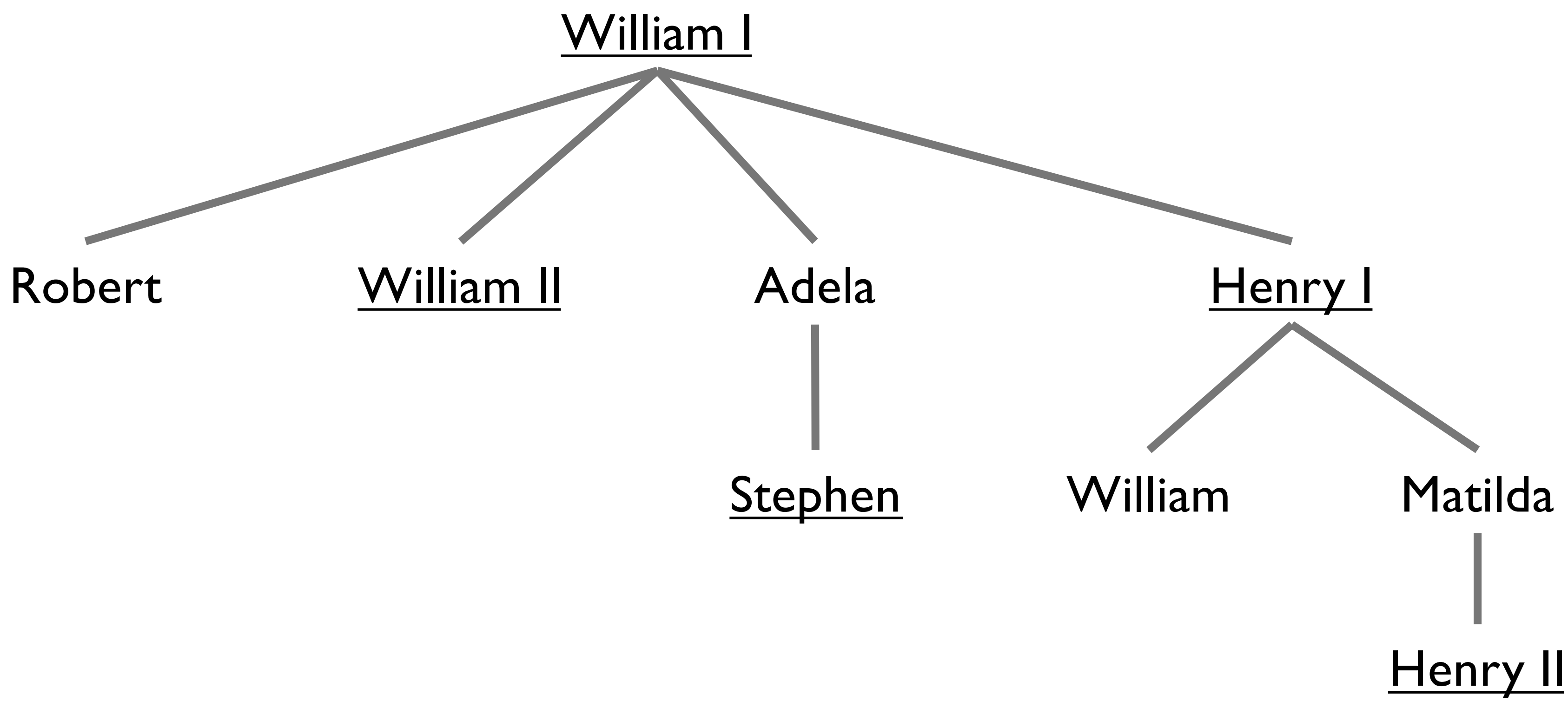
- A tree is a data structure where elements can have multiple successors (called children)
- But still only one predecessor (called parent)

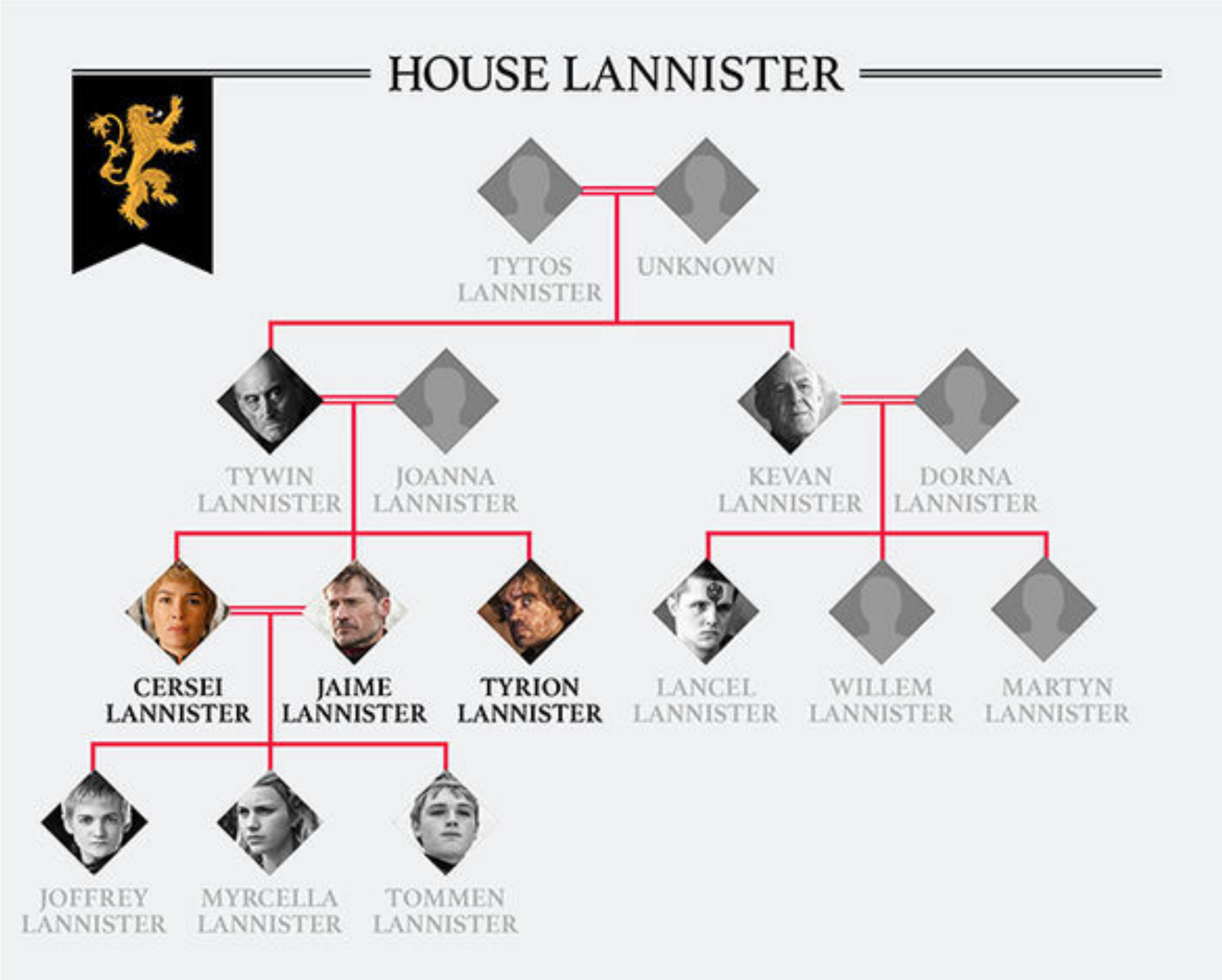

Root
Leaves

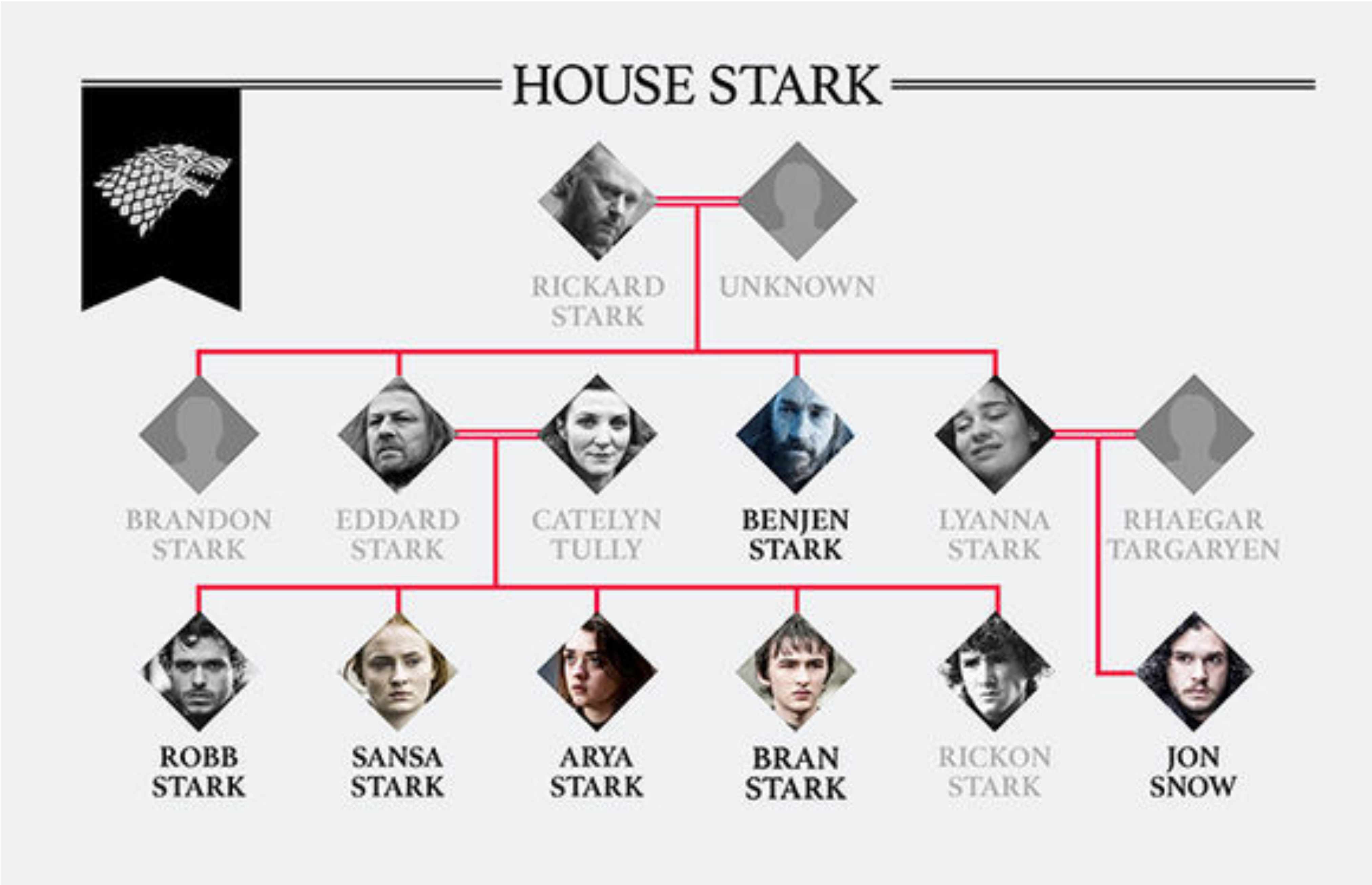
"Computer Tree"



House of Normandy, Battle of Hastings, 1066




HOUSE LANNISTER
TYTOS LANNISTER
UNKNOWN
TYWIN LANNISTER
JOANNA LANNISTER
KEVAN LANNISTER
DORNA LANNISTER
CERSEI LANNISTER
JAIME LANNISTER
TYRION LANNISTER
LANCEL LANNISTER
WILLEM LANNISTER
MARTYN LANNISTER
JOFFREY LANNISTER
MYRCELLA LANNISTER
TOMMEN LANNISTER

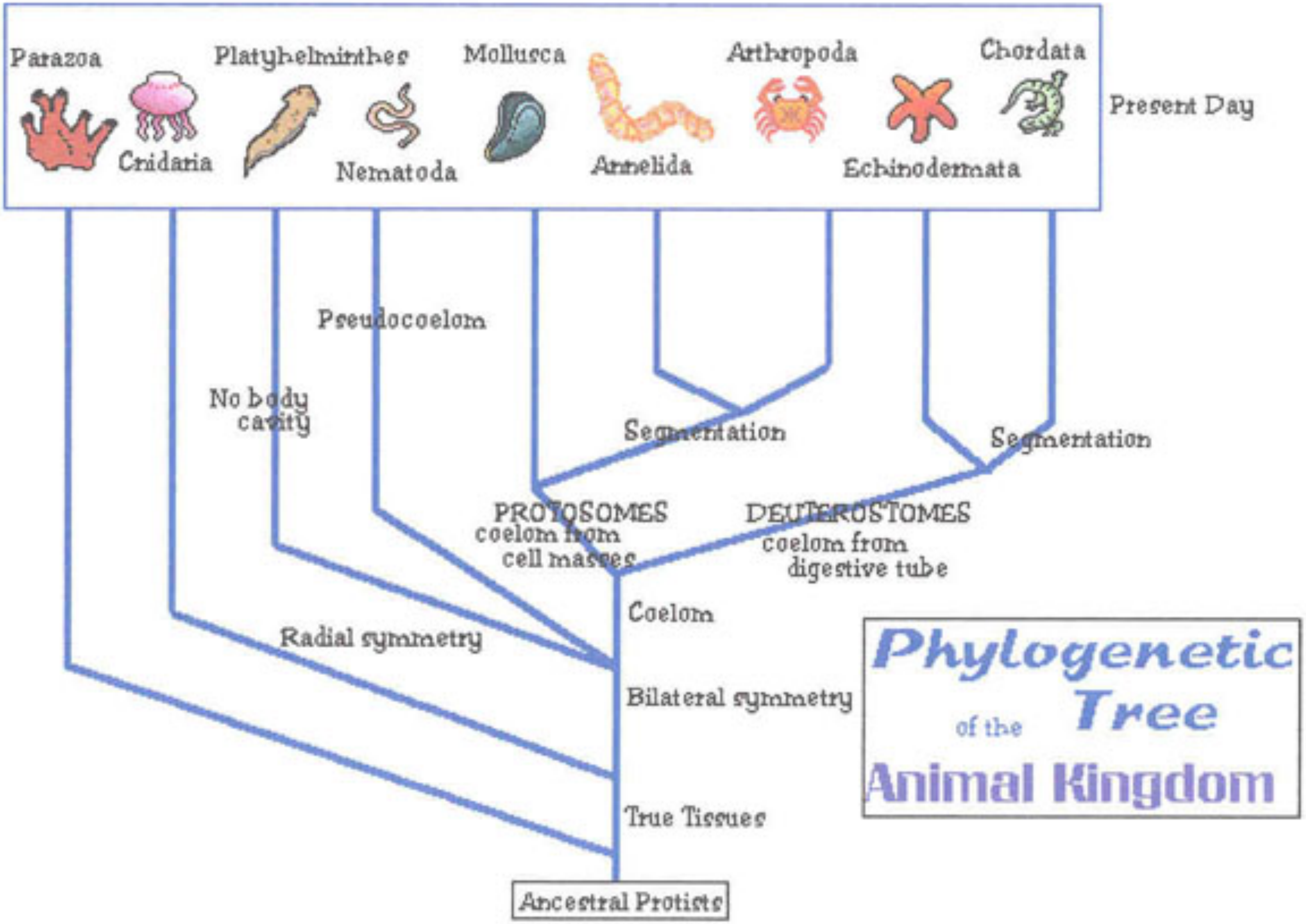

HOUSE STARK
RICKARD STARK
UNKNOWN
BRANDON STARK
EDDARD STARK
CATELYN TULLY
BENJEN STARK
LYANNA STARK
RHAEGAR TARGARYEN
ROBB STARK
SANSA STARK
ARYA STARK
BRAN STARK
RICKON STARK
JON SNOW

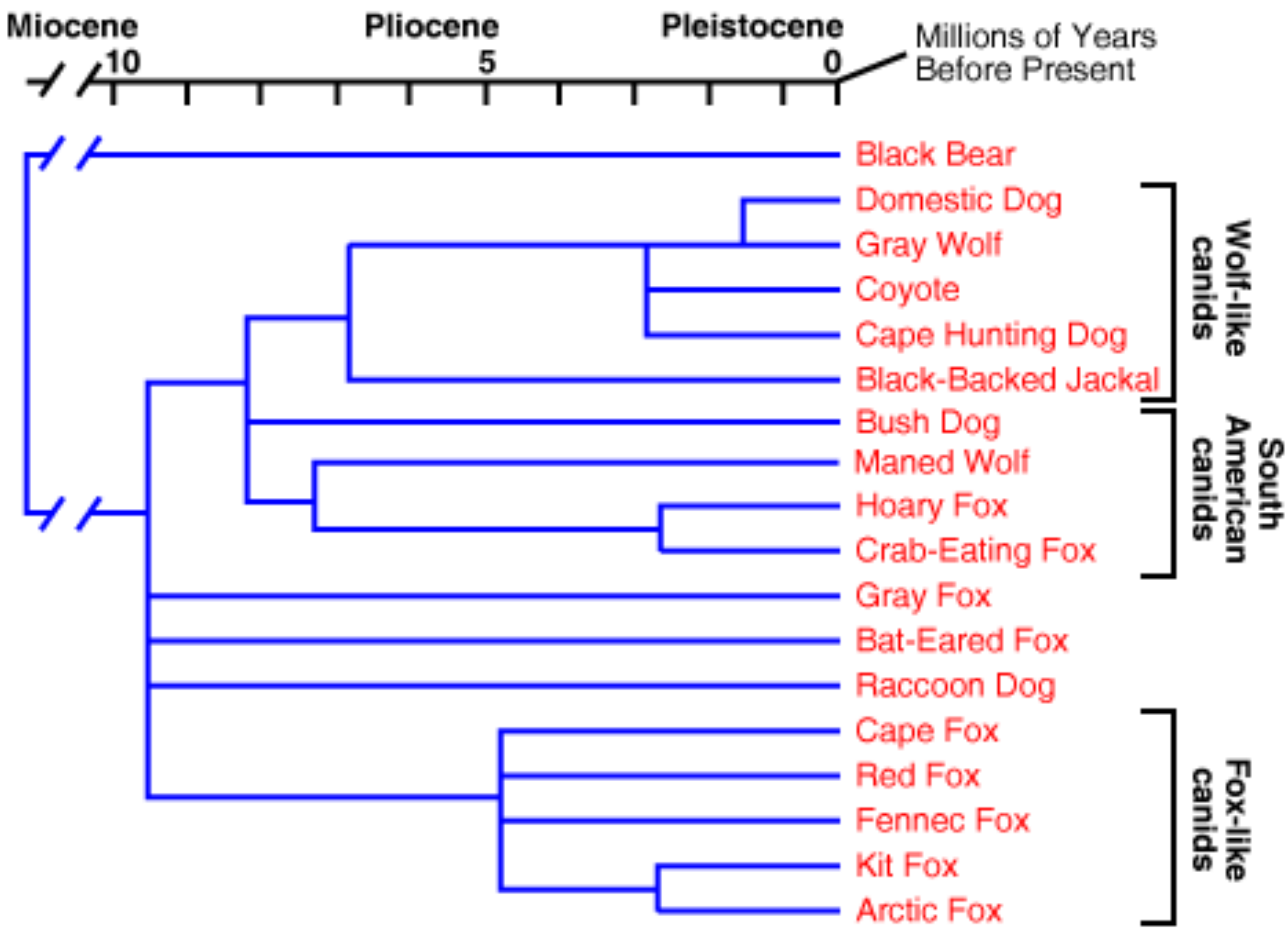
Tree Features

- Hierarchical relationship
- Root at the top
- Leaf at the bottom
- Interior nodes in middle
- Parents, children, ancestors, descendants, siblings
- Degree (of node): number of children of node
- Degree (of tree): maximum degree (across all nodes)
- Depth of node: number of *edges* from root to node
- Height of tree: maximum depth (across all nodes)

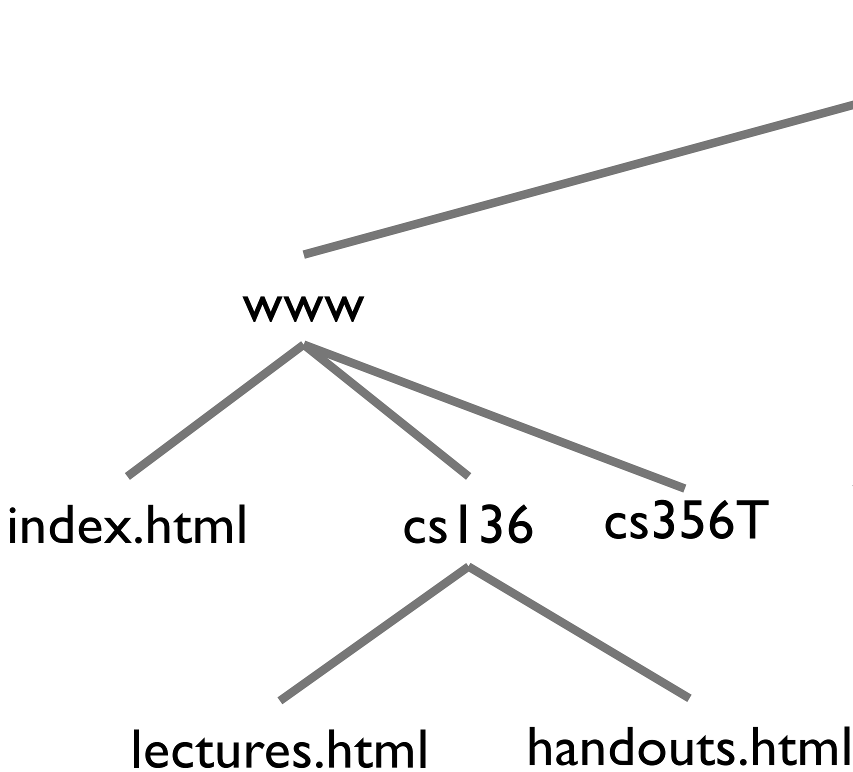
Other Trees

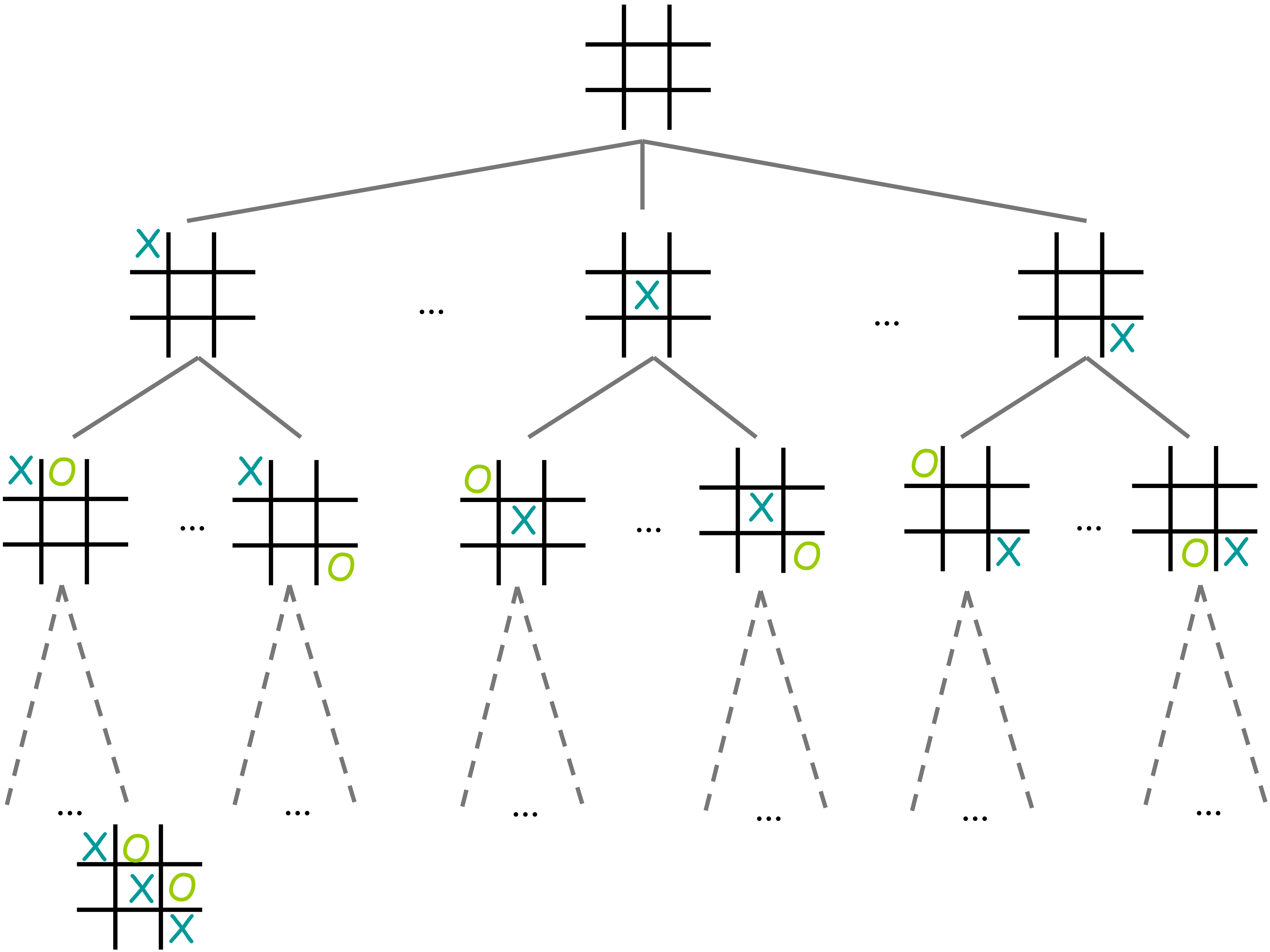
- Phylogenetic tree
- Directories of files
- Game trees
  - Build a tree
  - Search it for moves with high likelihood of winning
- Expression trees


Parazoa
Cnidaria
Platyhelminthes
Nematoda
Mollusca
Annelida
Arthropoda
Echinodermata
Chordata
Present Day
Pseudocoelom
No body cavity
Segmentation
Segmentation
PROTOSOMES
coelom from cell masses
DEUTEROSTOMES
coelom from digestive tube
Coelom
Radial symmetry
Bilateral symmetry
True Tissues
Ancestral Protists
Phylogenetic Tree
of the
Animal Kingdom



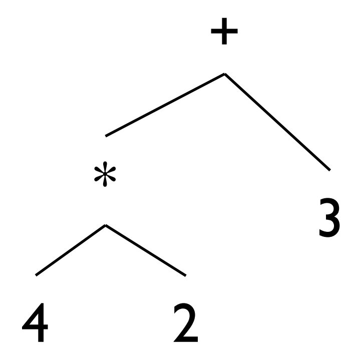
Miocene
Pliocene
Pleistocene
10
5
0
Millions of Years Before Present
Black Bear
Domestic Dog
Gray Wolf
Coyote
Cape Hunting Dog
Black-Backed Jackal
Bush Dog
Maned Wolf
Hoary Fox
Crab-Eating Fox
Gray Fox
Bat-Eared Fox
Raccoon Dog
Cape Fox
Red Fox
Fennec Fox
Kit Fox
Arctic Fox
Wolf-like canids
South American canids
Fox-like canids


~lenhart
www
research
papers
index.html
cs136
cs356T
...
...
lectures.html
handouts.html

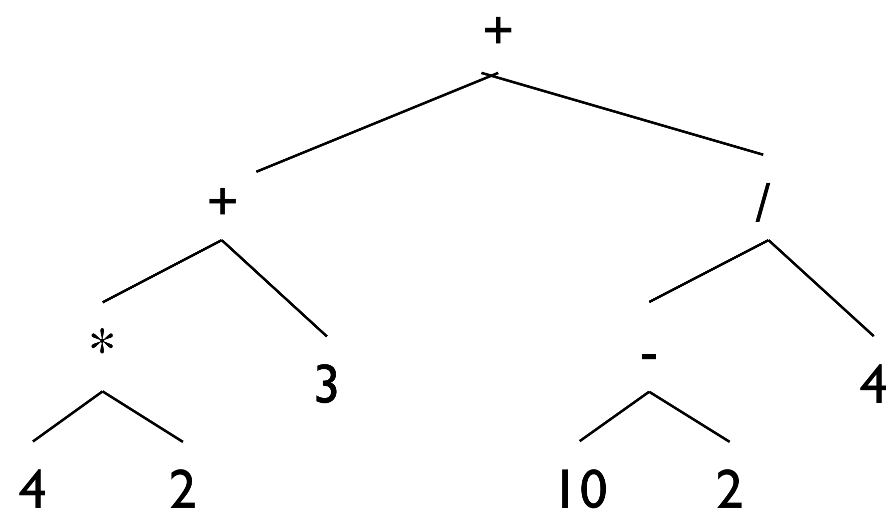


Expression Trees

4 * 2 + 3



(4 * 2 + 3) + ((10 – 2)/ 4)



Introducing Binary Trees

- Degree of all nodes <= 2
- Recursive nature of tree
  - Empty
  - Root with left and right subtrees
- SLL: Recursive nature was captured by nodes (Node<E>) on inside
- Binary Tree: No "inner" node class; single BinaryTree class does it all