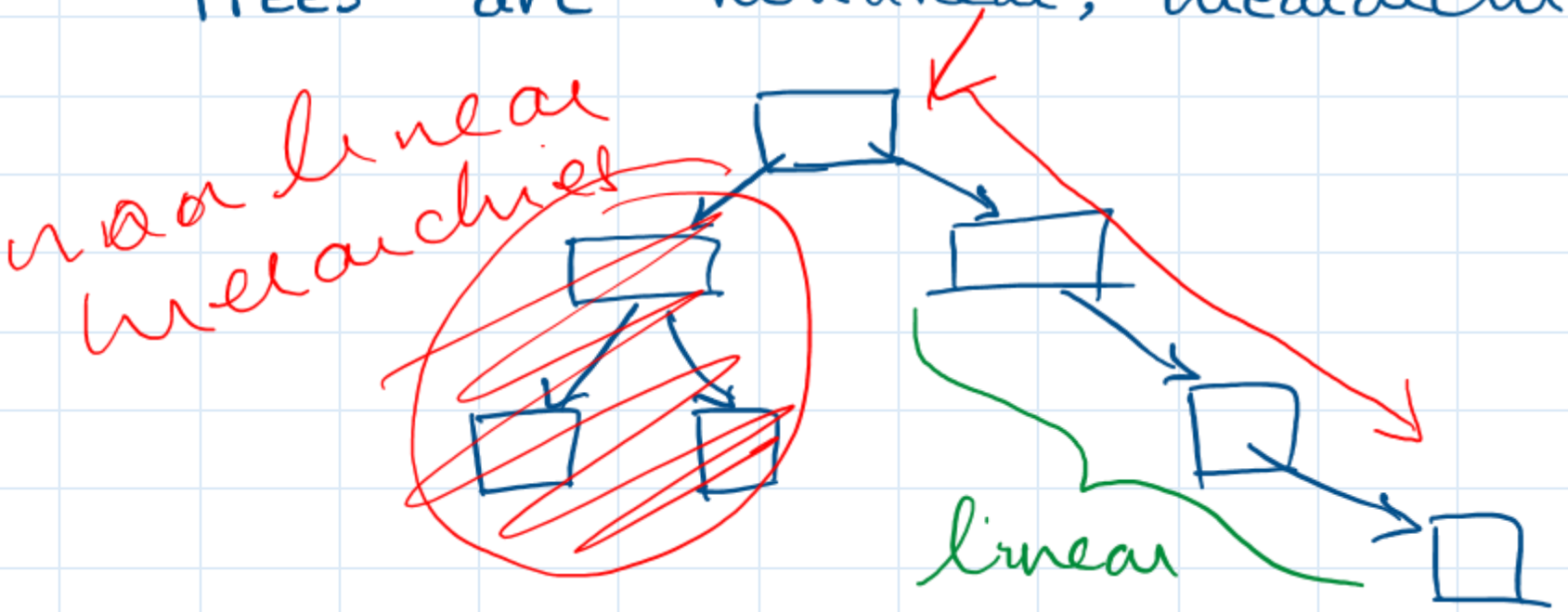
COL106 - Data Structures and Algorithms

From Linear to Hierarchical Data

linear

Lists are linear structures.
Growth by concatenation.

Trees are nonlinear, hierarchical

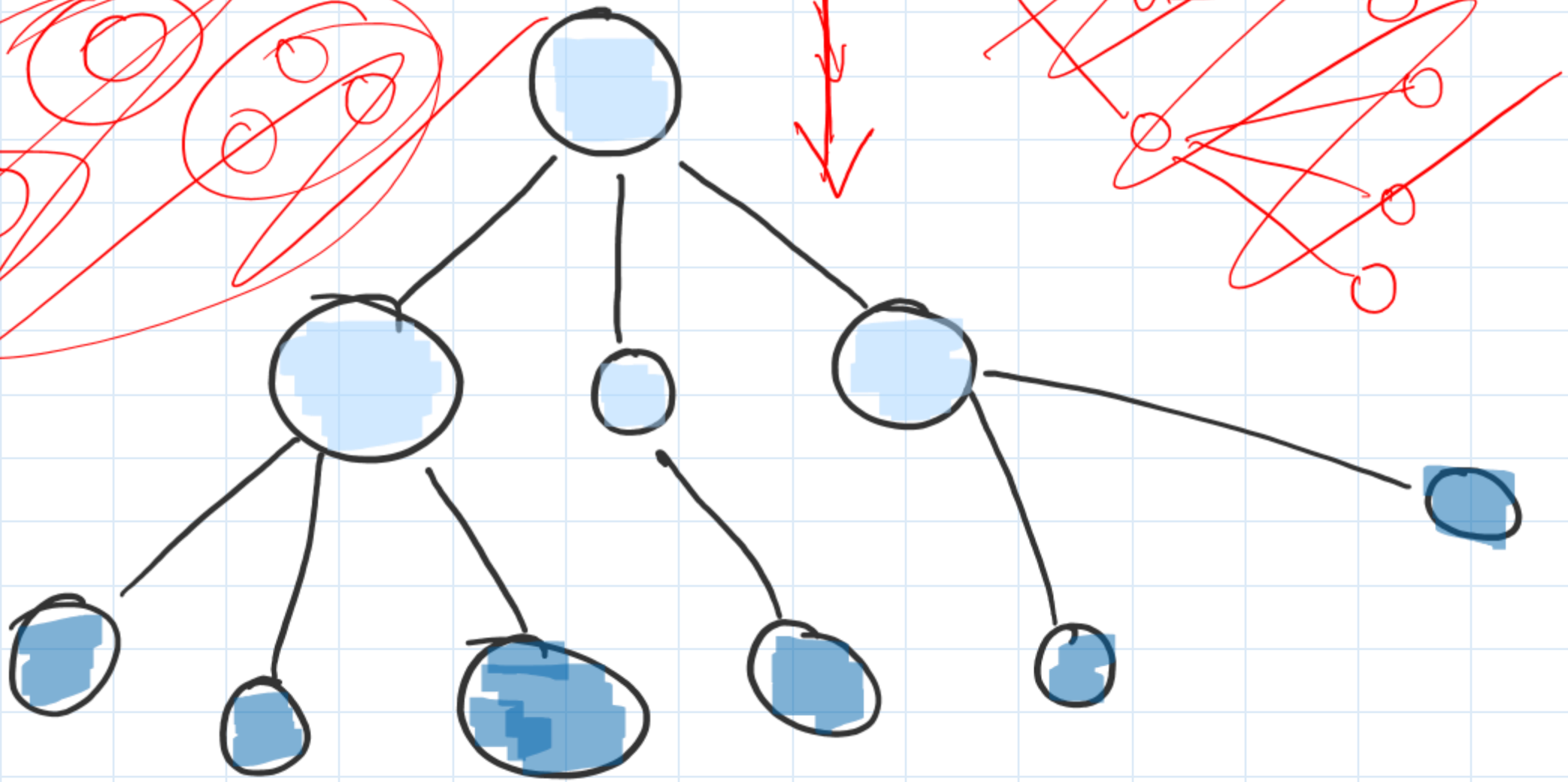


Every list is a tree. (degenerate)

Trees are not lists

Some Definitions

canonical structure



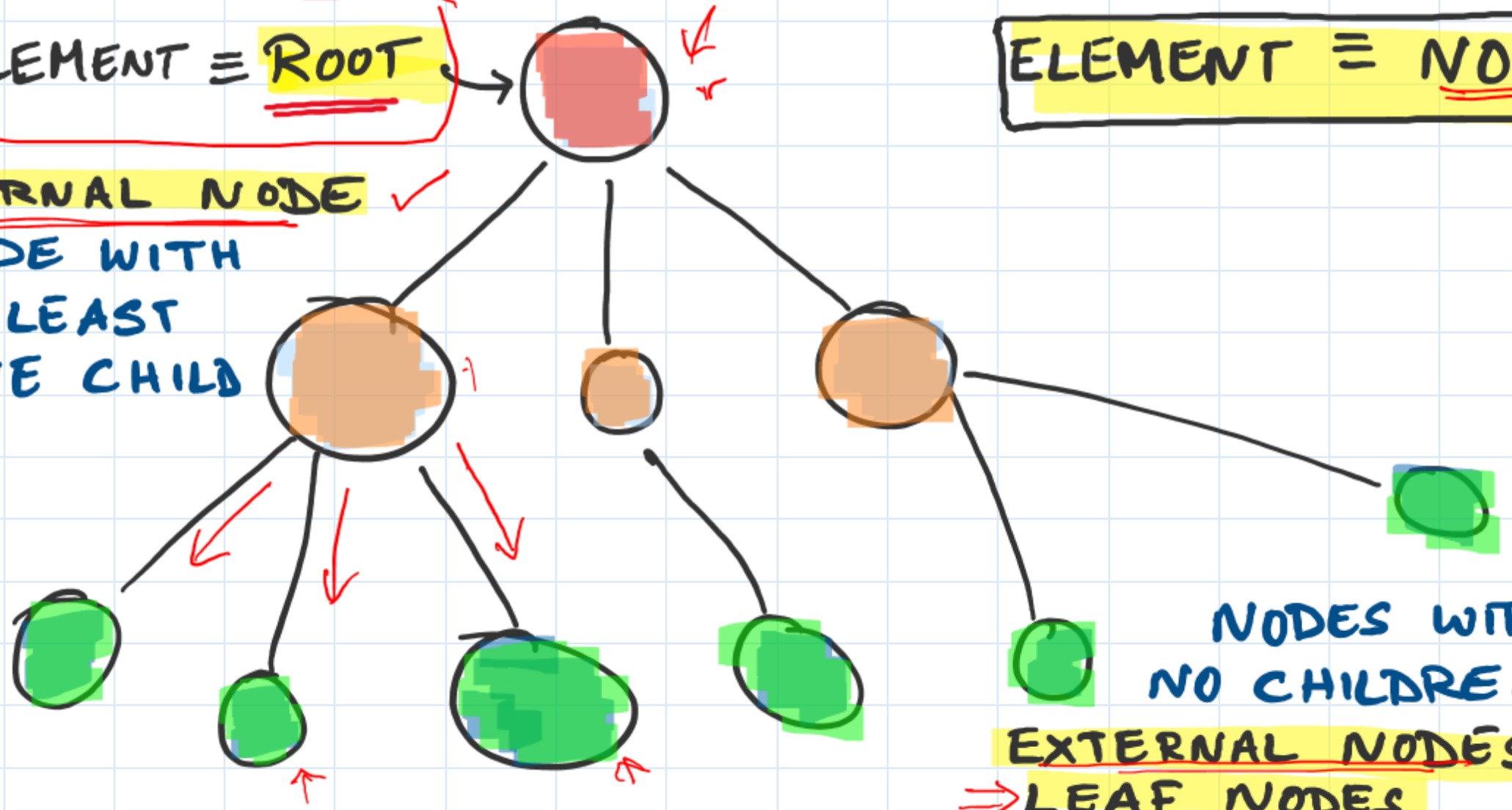
EXCEPT TOP ELEMENT EACH ELEMENT IN A TREE HAS A PARENT ELEMENT AND ZERO OR MORE CHILDREN

TOP ELEMENT ≡ ROOT

ELEMENT ≡ NODE

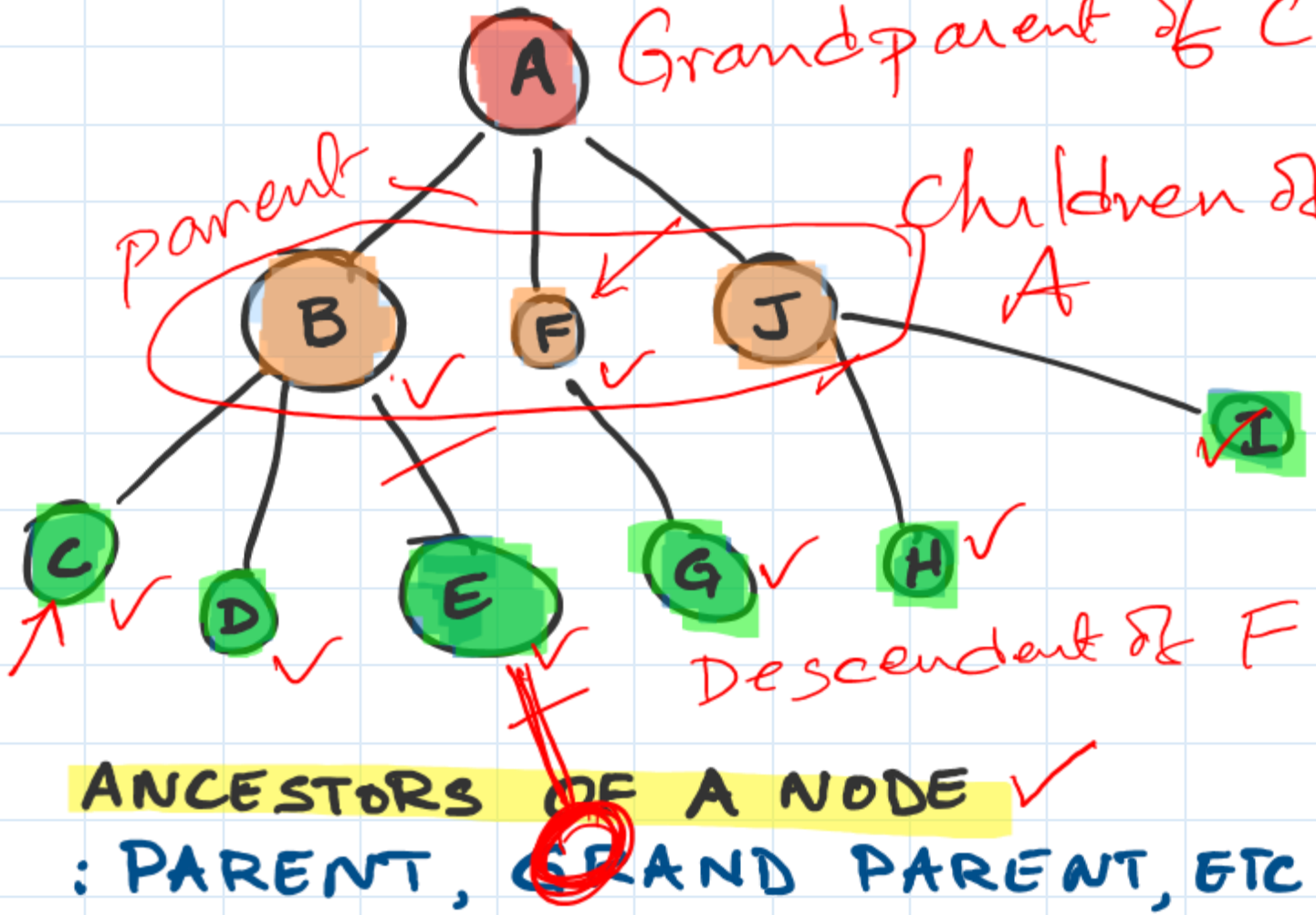
INTERNAL NODE

NODE WITH AT LEAST ONE CHILD



NODES WITH NO CHILDREN

EXTERNAL NODES OR LEAF NODES



ELEMENT ≡ NODE

TOP ELEMENT ≡ ROOT

INTERNAL NODE

EXTERNAL NODES OR LEAF NODES

ANCESTORS OF A NODE ✓

: PARENT, GRAND PARENT, ETC

DESCENDENTS ✓

: CHILD, GRANDCHILD, GREAT-GRANDCHILD ETC.

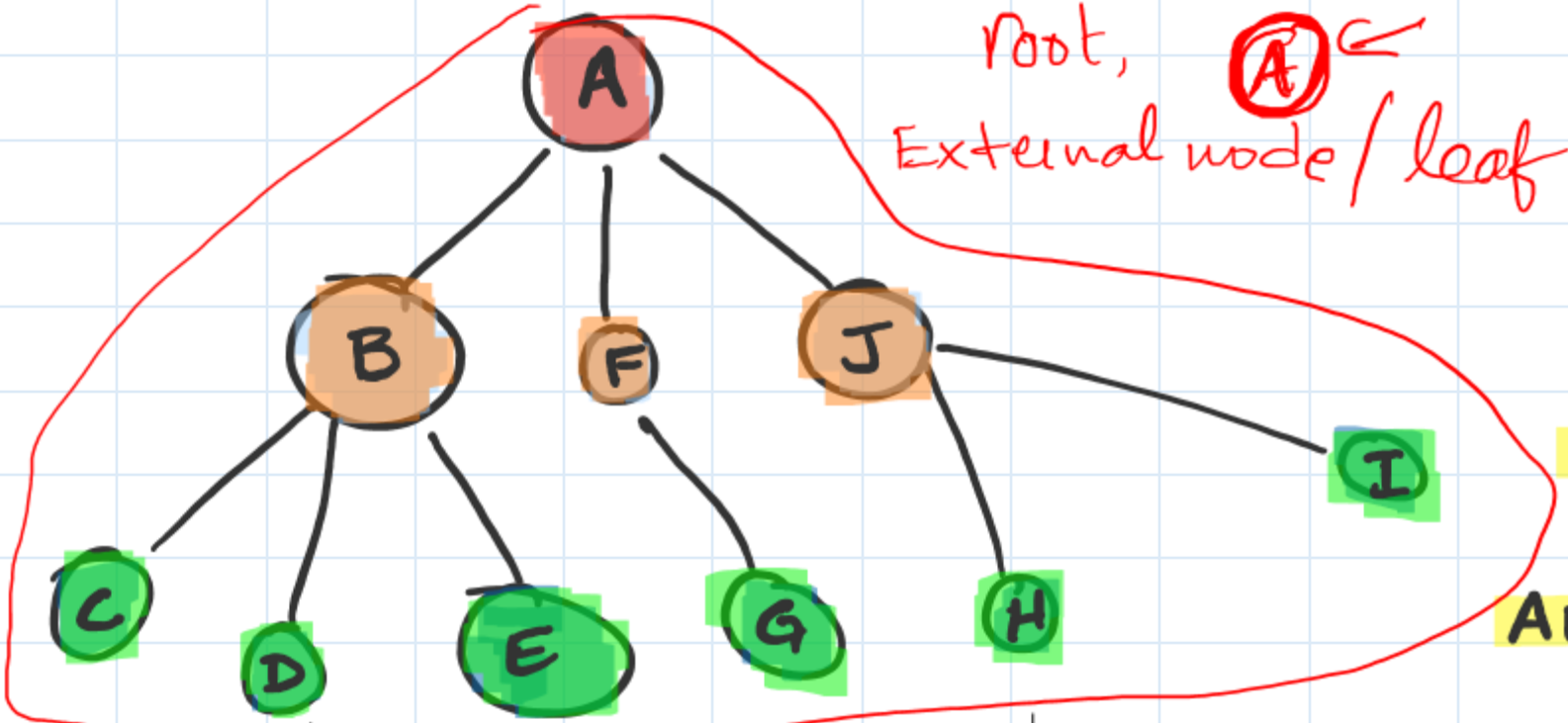
DEPTH OF A NODE

: NUMBER OF ANCESTORS

depth of E = 2

HEIGHT OF A TREE

MAXIMUM DEPTH OF ANY NODE



Root, (A) ← External node / leaf.

	IN?	EN?	ANC.	DESC	DEPTH
(B)	✓	X	{A}	{C, D, E}	1
(G)	X	✓	{F, A}	{ }	2
(A)	✓	X	{ }	{all nodes}	0

ELEMENT ≡ NODE

TOP ELEMENT ≡ ROOT

INTERNAL NODE

EXTERNAL NODES OR LEAF NODES

ANCESTORS OF A NODE

DESCENDENTS

DEPTH OF A NODE

HEIGHT OF A TREE

DEGREE OF A NODE

OF CHILDREN

TREE T

is a set of nodes storing elements such that nodes have parent-child relationship.

RECURSIVE DEFINITION OF A TREE

(a) THERE IS ONE SPECIAL NODE root OF THE TREE

(b) REST OF THE NODES ARE PARTITIONED $m \geq 0$ DISJOINT SETS $T_1, T_2 \ldots T_m$ WHERE EACH T_i IS A TREE

- T can be empty ✓
- T can be non empty with just one node root ✓
- each node v, except the root, has a unique parent

TREE T

is a set of **nodes** storing elements such that nodes have **parent-child** relationship.

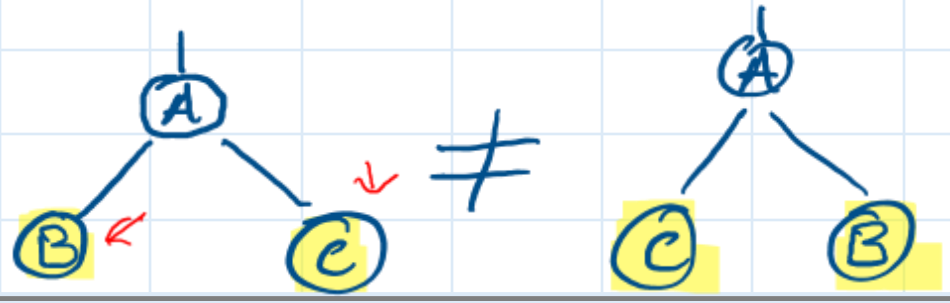
(a) THERE IS ONE SPECIAL NODE root OF THE TREE

(b) REST OF THE NODES ARE PARTITIONED $m \geq 0$ DISJOINT SETS $T_1, T_2 \ldots T_m$ WHERE EACH T_i IS A TREE

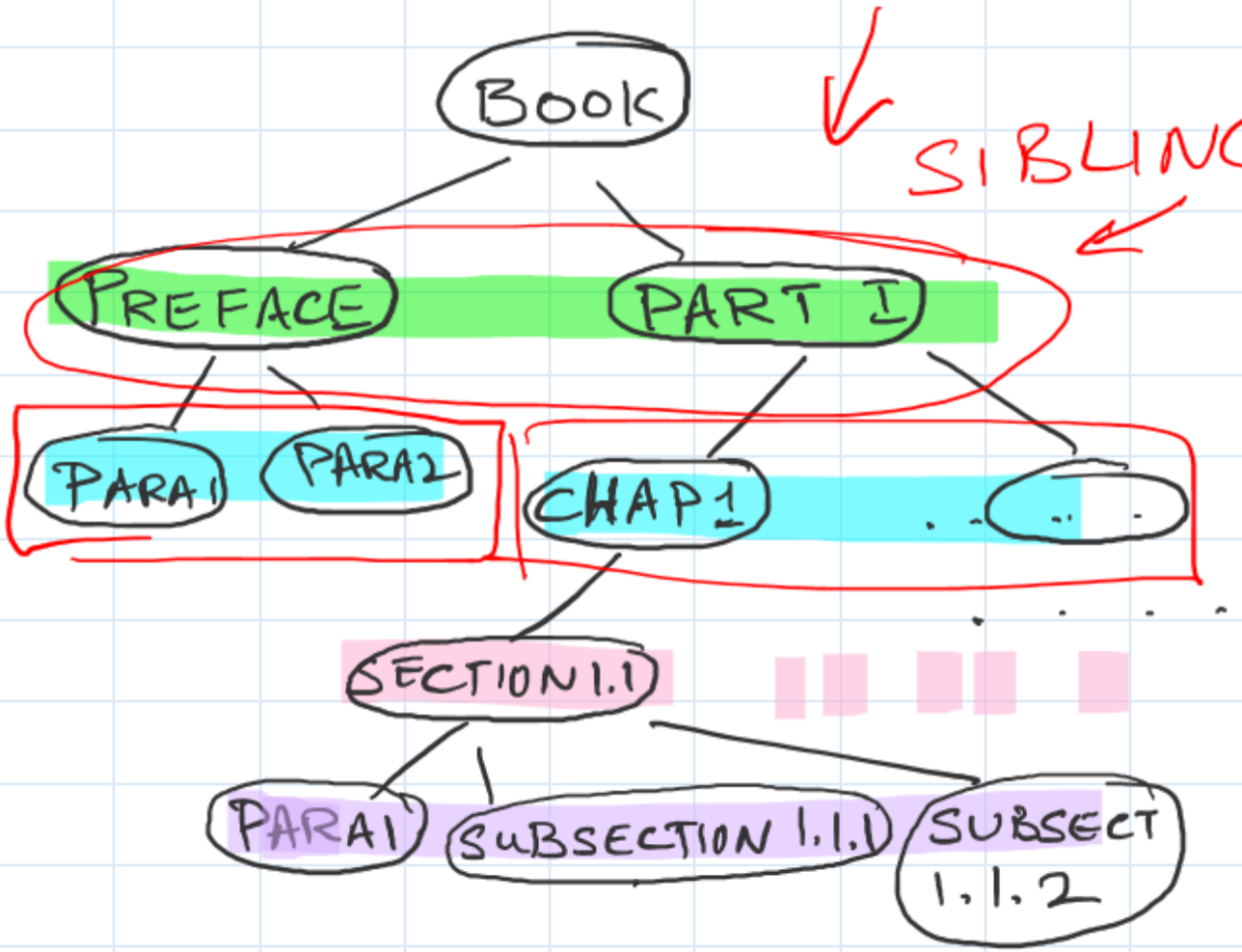
- T can be empty
- T can be nonempty with just one node **root**
- each node v, except the root, has a **unique parent**

ORDERED TREE

A TREE IN WHICH THE BRANCHES OF EACH NODE ARE ORDERED.

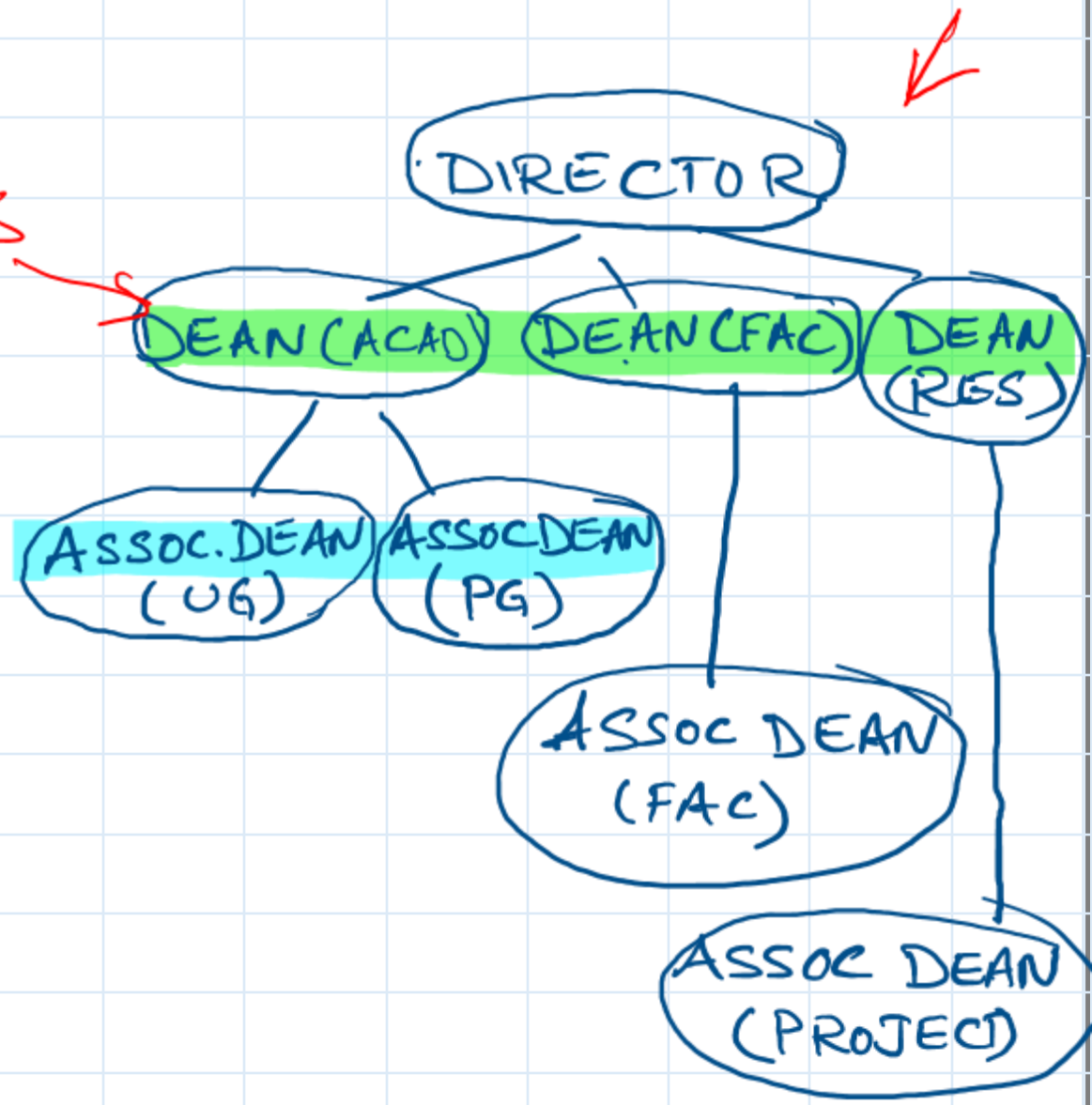


ORDERED TREE



SIBLINGS

UNORDERED TREE



TREE ADT (Focus on Accessor methods)

ACCESSOR METHODS

root() ✓

parent (p) get p's parent

children (p)

numChildren (p) ←

size() ✓

isEmpty()

POSITION (NODE)

getElement()

QUERY METHODS

isInternal (p) ←

isExternal (p) ✓

isRoot (p) ✓

positions/nodes () ✓

iterator ()

TREE ADT (Focus on Accessor methods)

ACCESSOR METHODS

root()

parent (p)

children (p)

numChildren (p)

Iterator over all children of p

POSITION (NODE)

getElement()

QUERY METHODS

isInternal (p)

isExternal (p)

isRoot (p)

size()

isEmpty()

positions/nodes ()

iterator ()

TREE IN JAVA

sun.util.Tree

⇒ THERE IS NO STANDARD CLASS / INTERFACE IN JAVA FOR TREE.

```
public interface Tree<E>
        extends Iterable<E> {
    TreeNode<E> root();
    TreeNode<E> parent(TreeNode<E> p)
            throws IllegalArgumentException;
      :
    Iterator<E> iterator();
    Iterable<TreeNode<E>> positions();
}
```

```
public abstract class
    AbstractTree<E>
        implements Tree<E>{
    public boolean
        isEmpty(){
        return (size()==0);
    }
      :
}
```