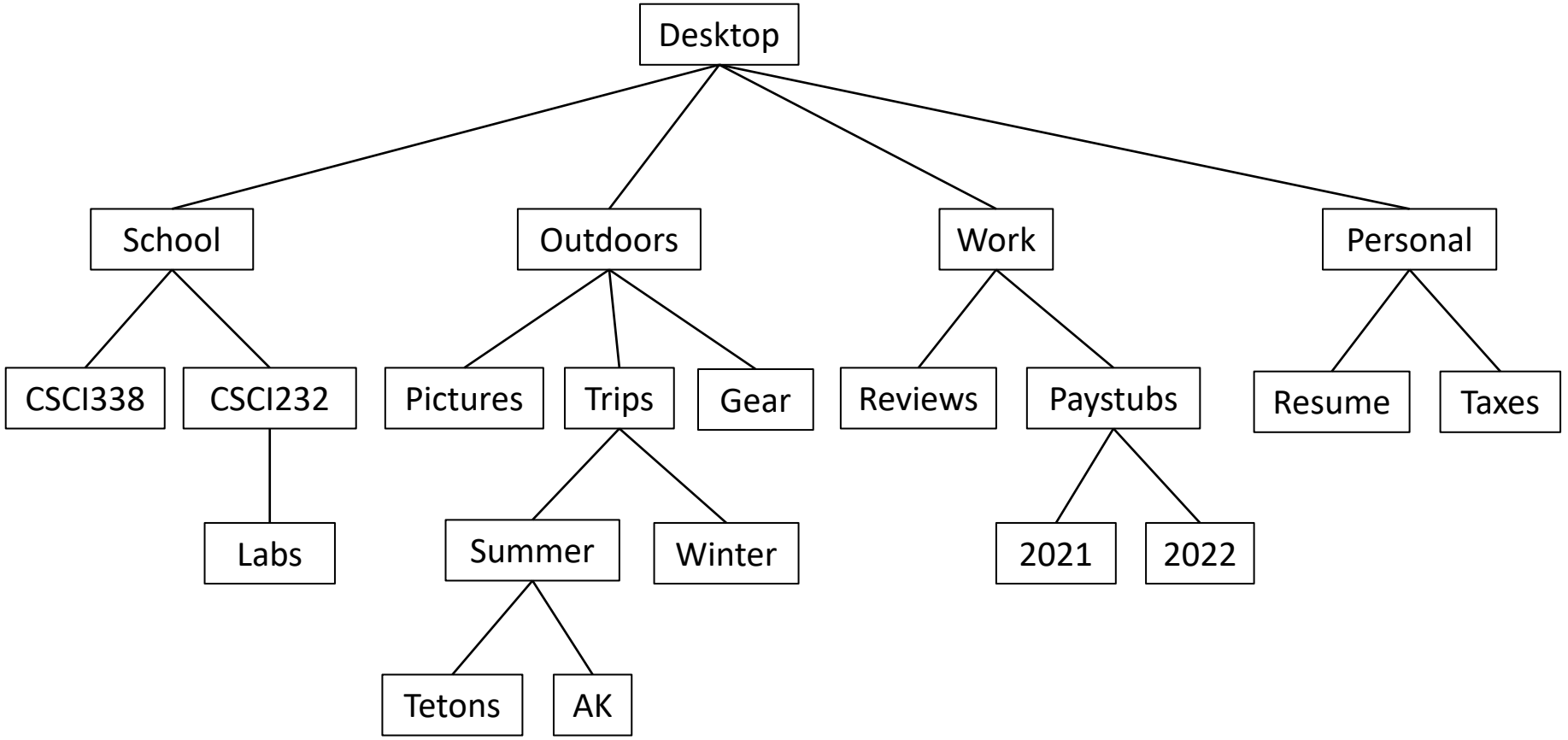
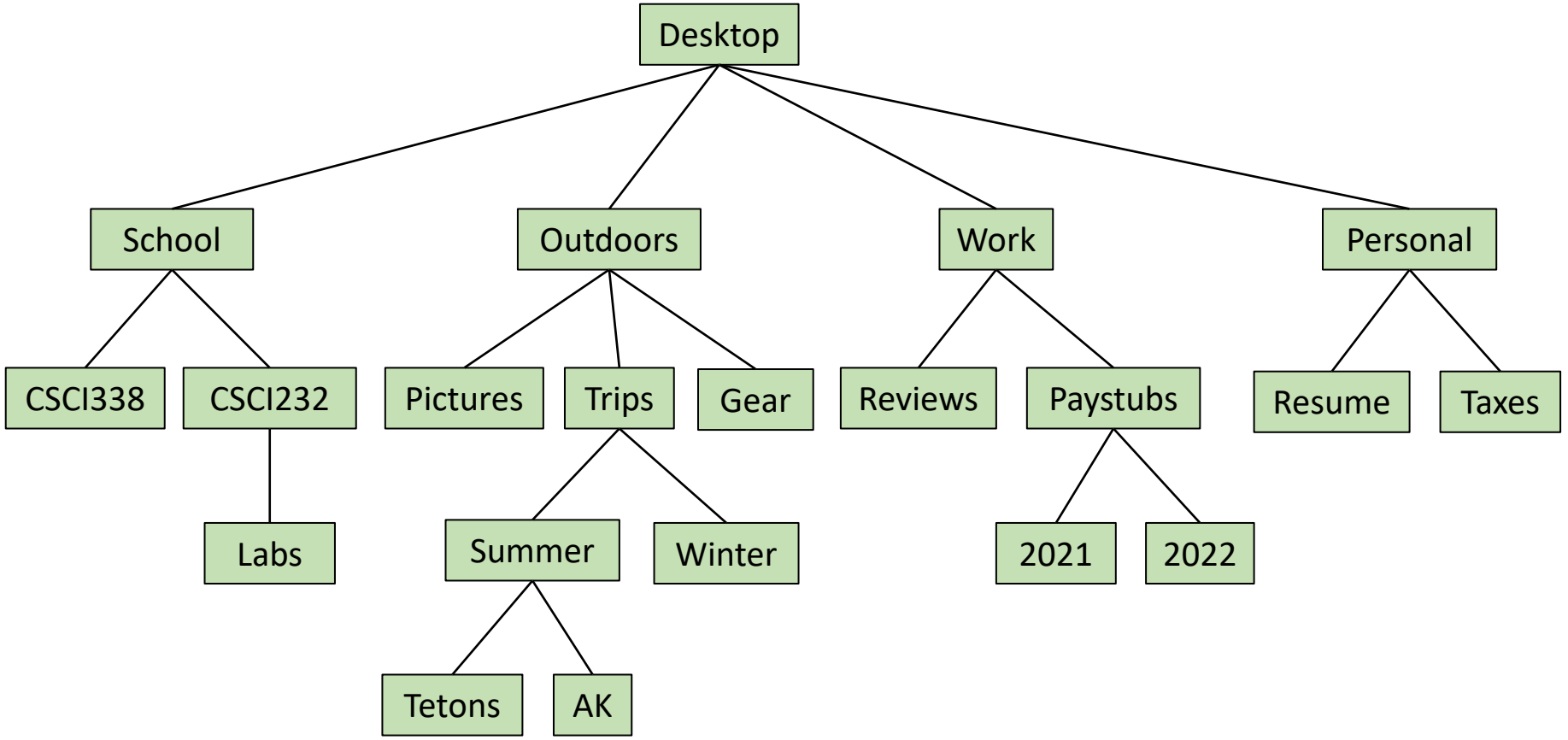
# Trees
## CSCI 232

# Trees



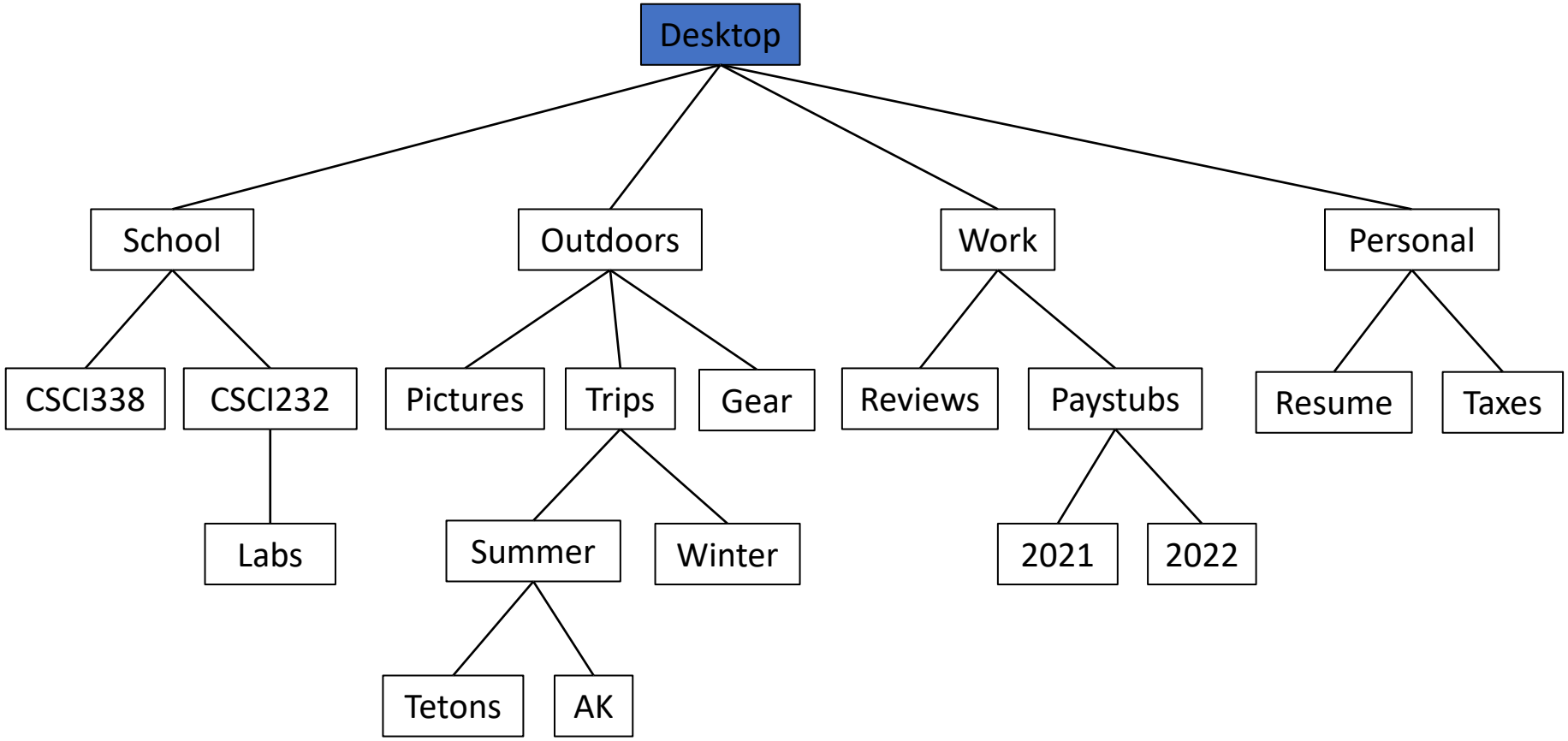

Trees are data structures use to store elements hierarchically (not linearly like arrays and linked lists).

# Trees - Definitions



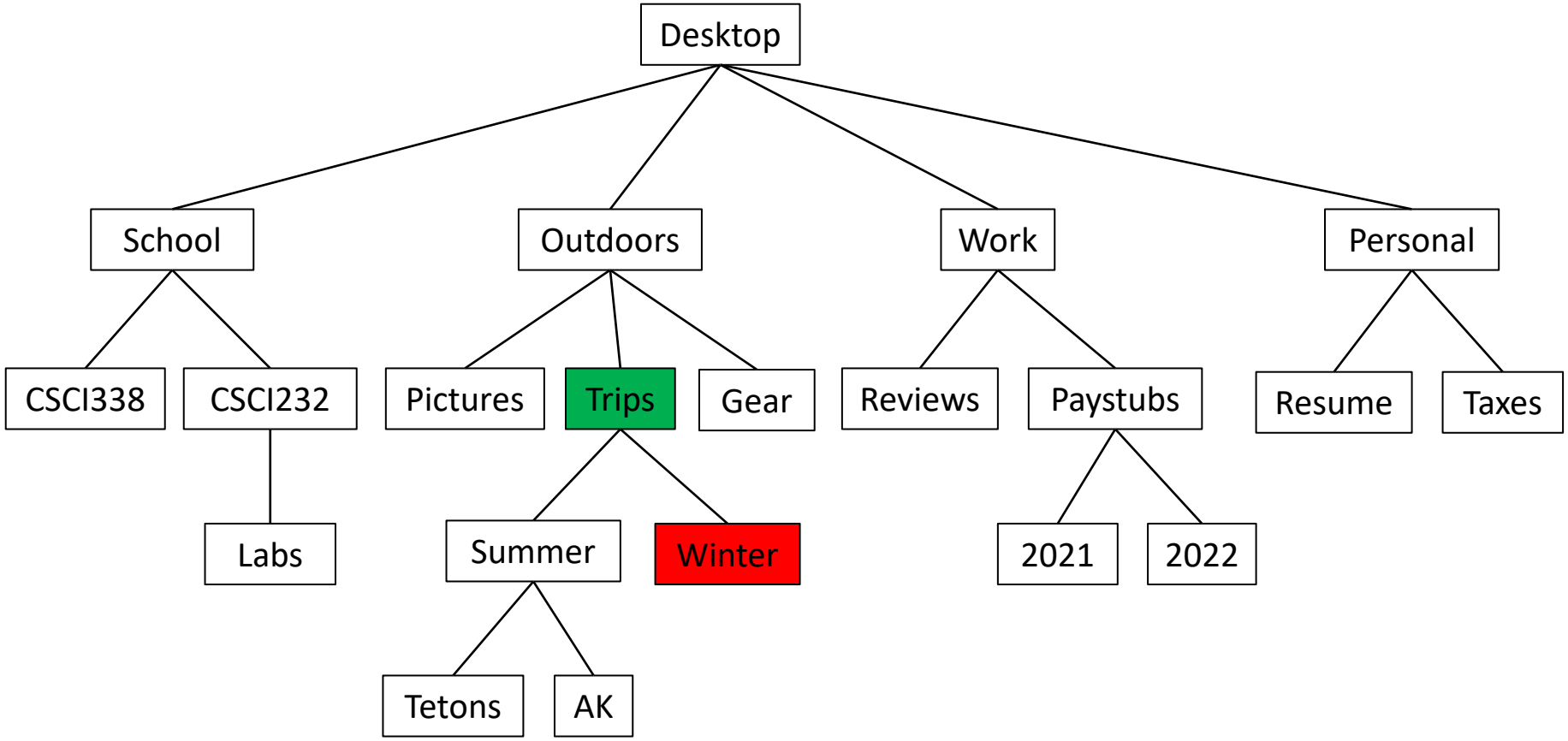

**Nodes**: The **entities** that make up the tree.

# Trees - Definitions



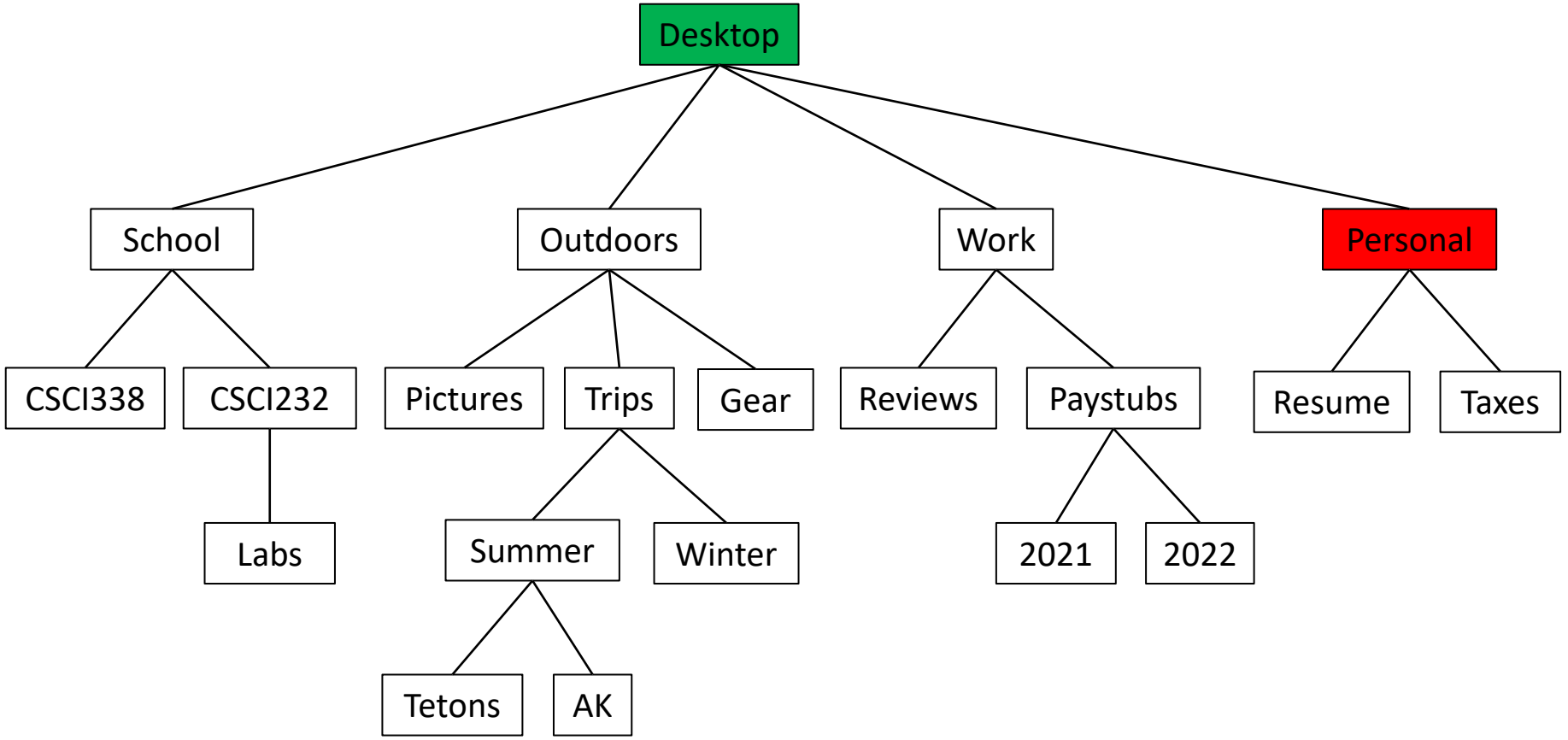

**Root**: The **node** at the top of hierarchy.

# Trees - Definitions



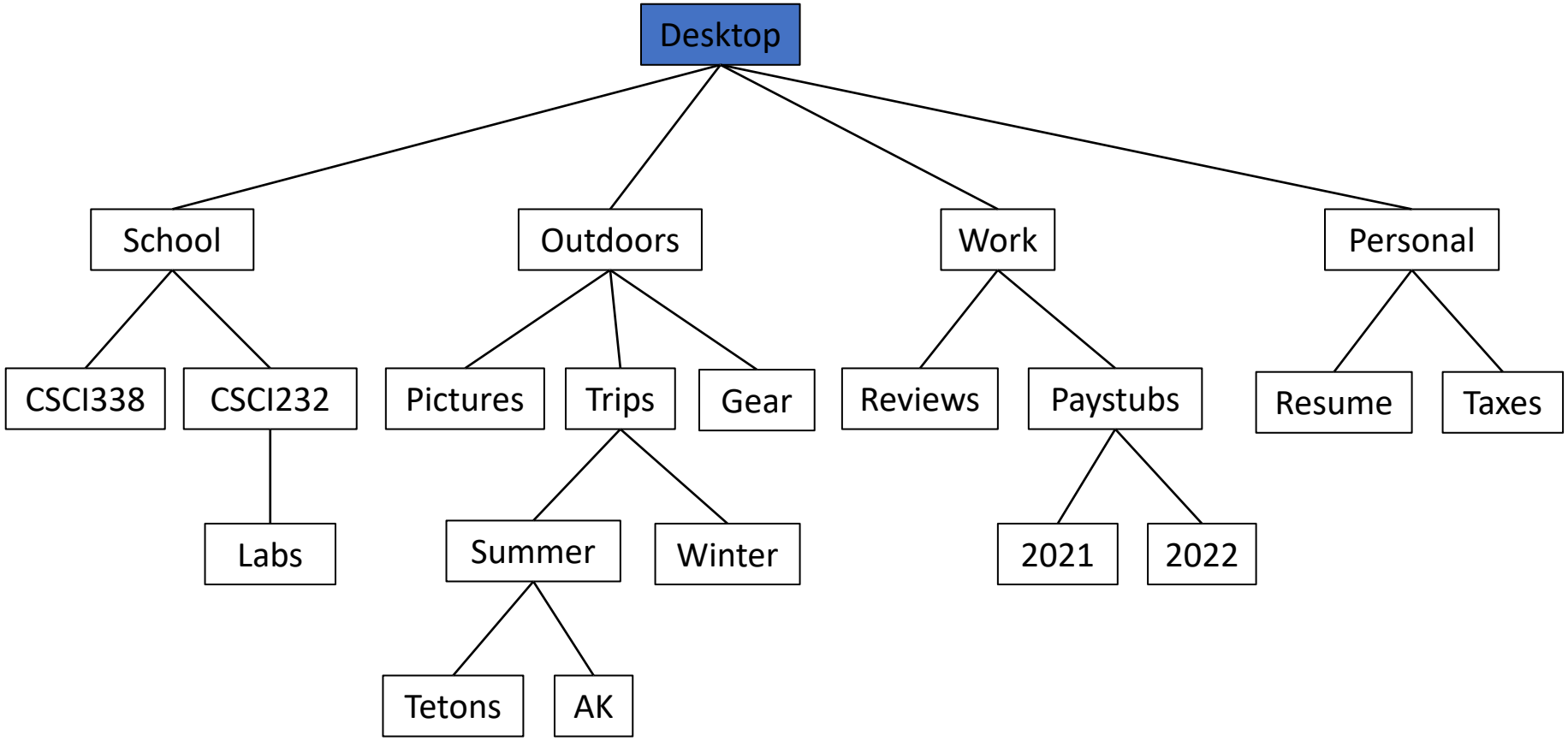

**Parent**: For a given **node**, its **parent** is the node that directly precedes it in the hierarchy.

# Trees - Definitions



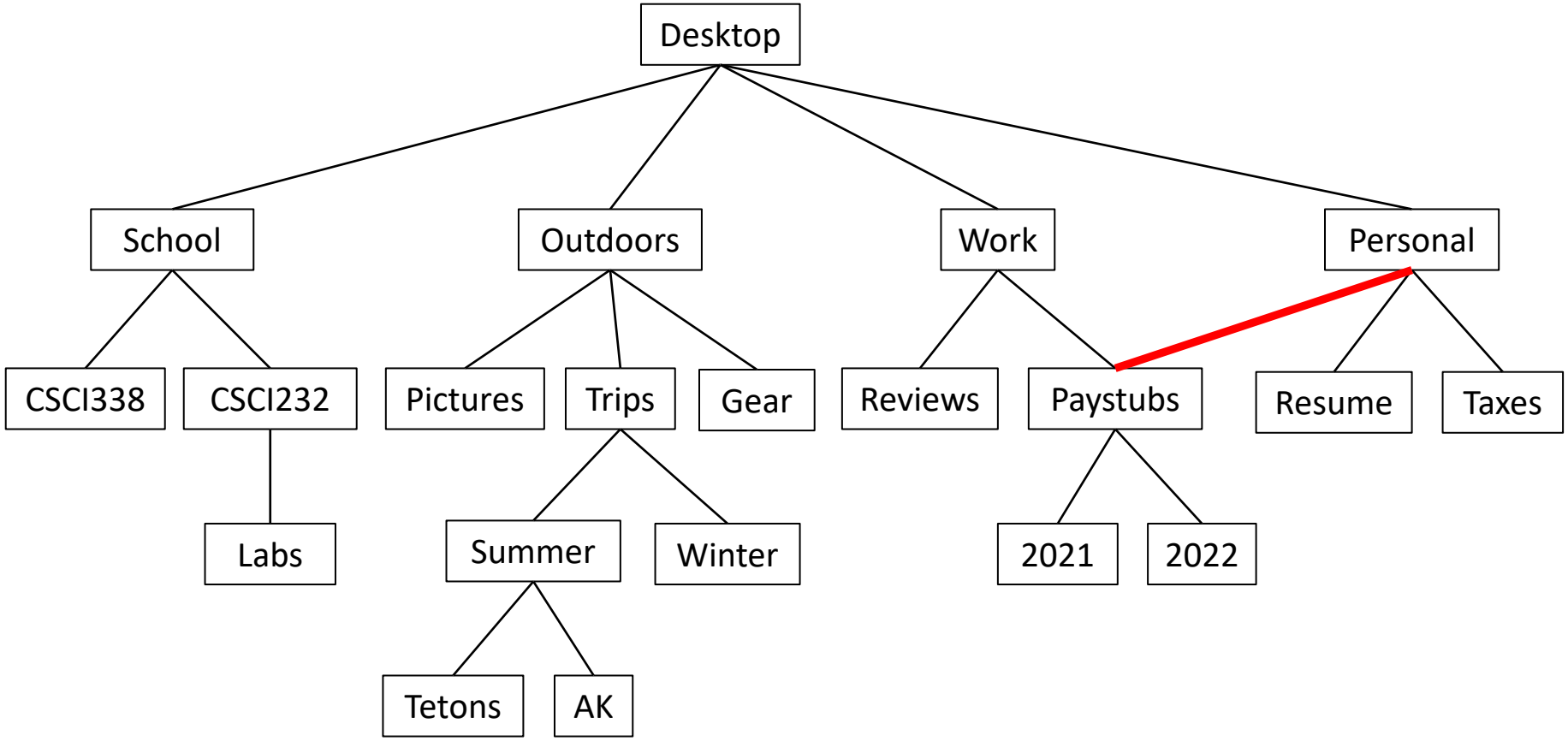

**Parent**: For a given **node**, its **parent** is the node that directly precedes it in the hierarchy.

# Trees - Definitions



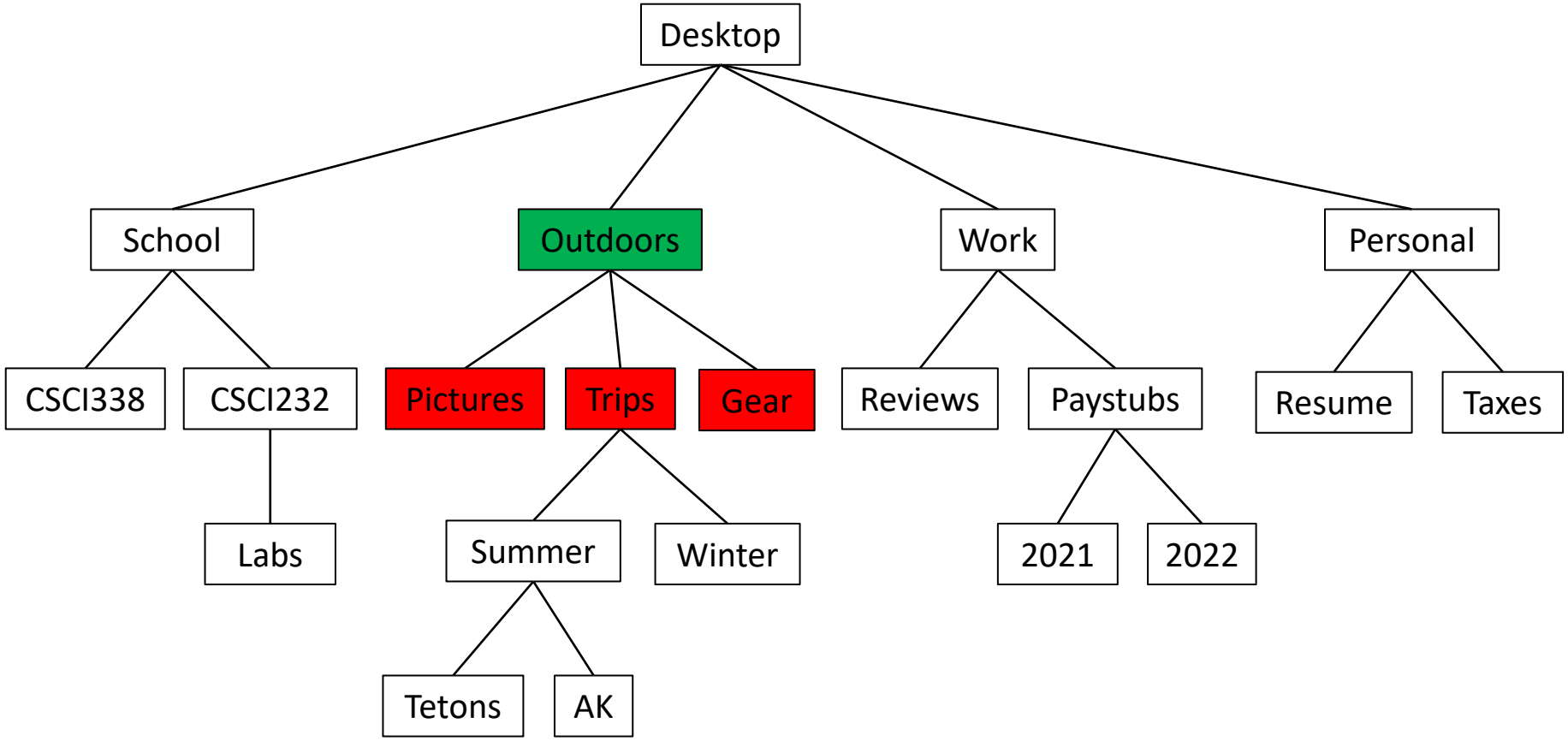

**Parent**: For a given node, its parent is the node that directly precedes it in the hierarchy. Every node has a parent except the **root**.

# Trees - Definitions



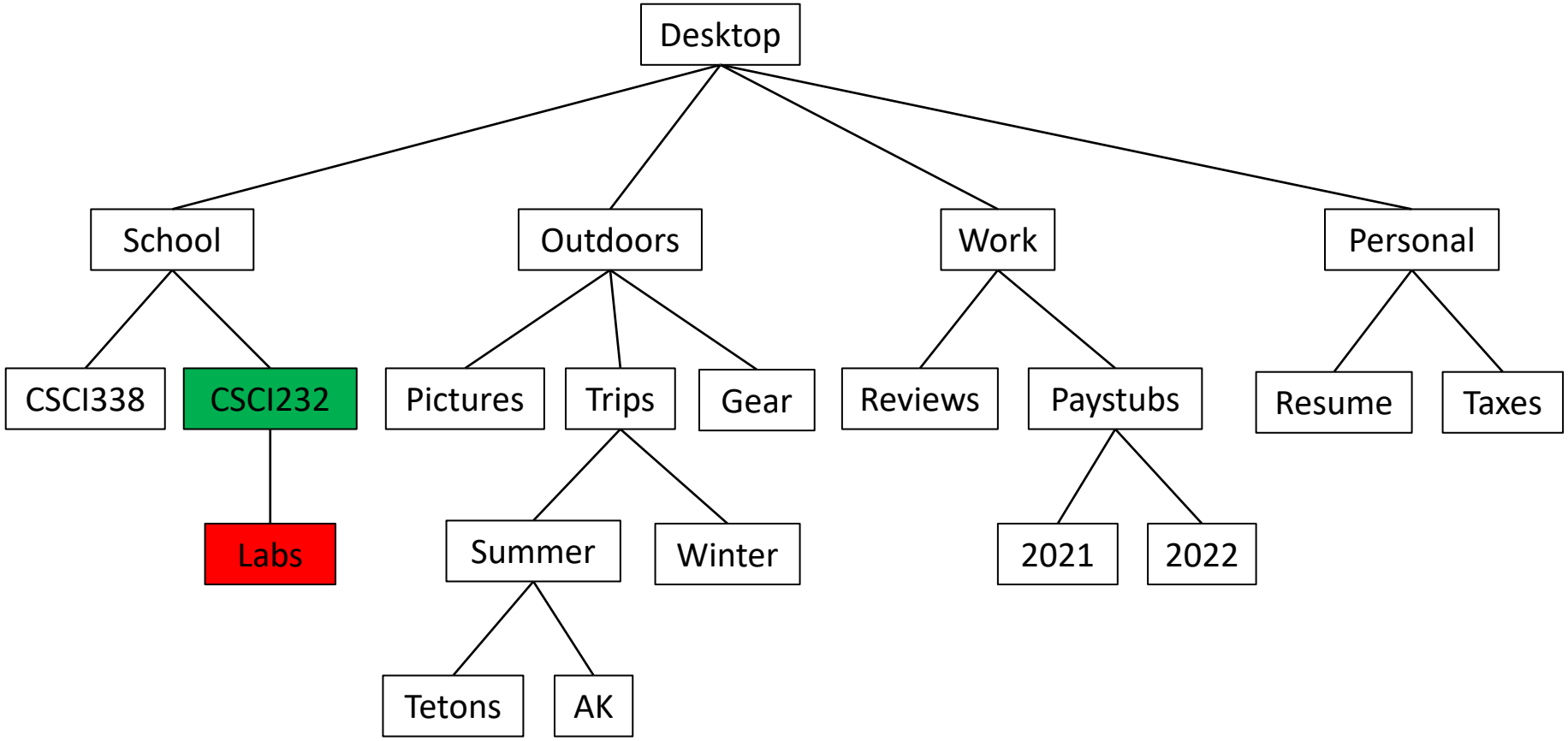

**Parent**: For a given node, its parent is the node that directly precedes it in the hierarchy. Every node has a parent except the root. Nodes may **not** have multiple parents.

# Trees - Definitions



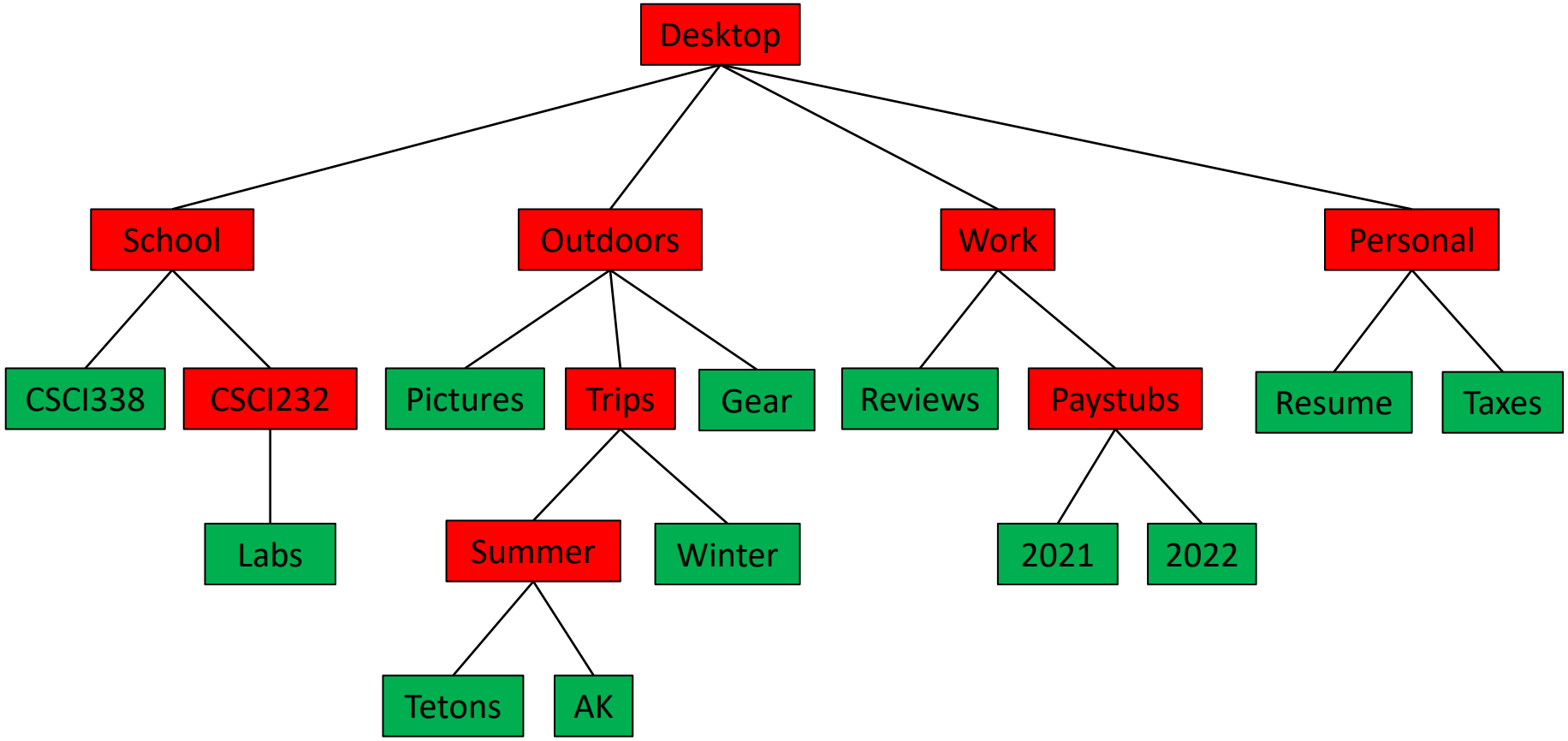

**Child**: For a given **node**, its children are the **node(s)** that directly follow it in the hierarchy.

# Trees - Definitions




**Child**: For a given **node**, its children are the **node(s)** that directly follow it in the hierarchy.

# Trees - Definitions

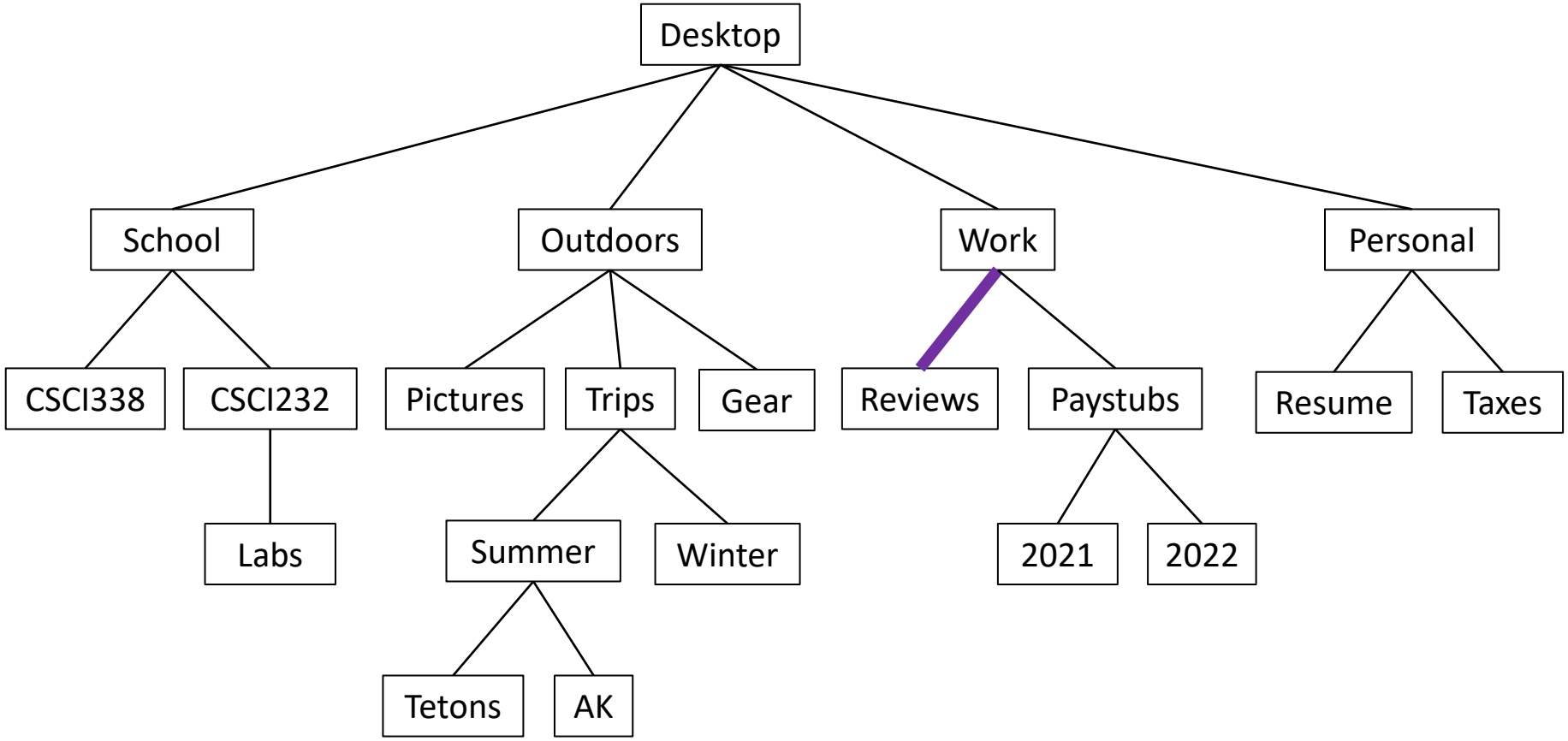



**Internal Node**: A node with at least one child (i.e., parent nodes).

**Leaf Node**: A node without children.

Every node is an internal node or a leaf node.

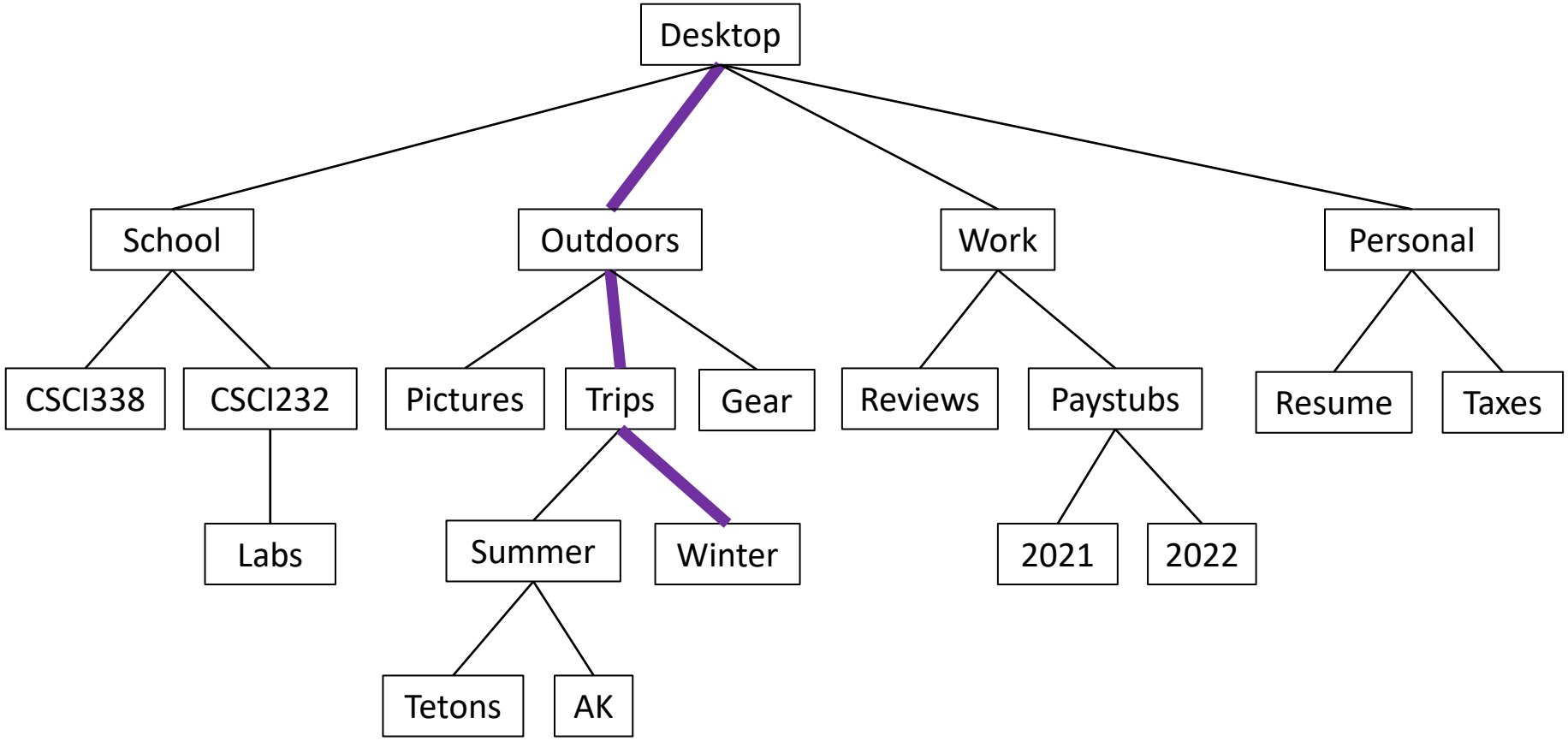
# Trees - Definitions




**Edge**: A pair of nodes such that one is the parent of the other.

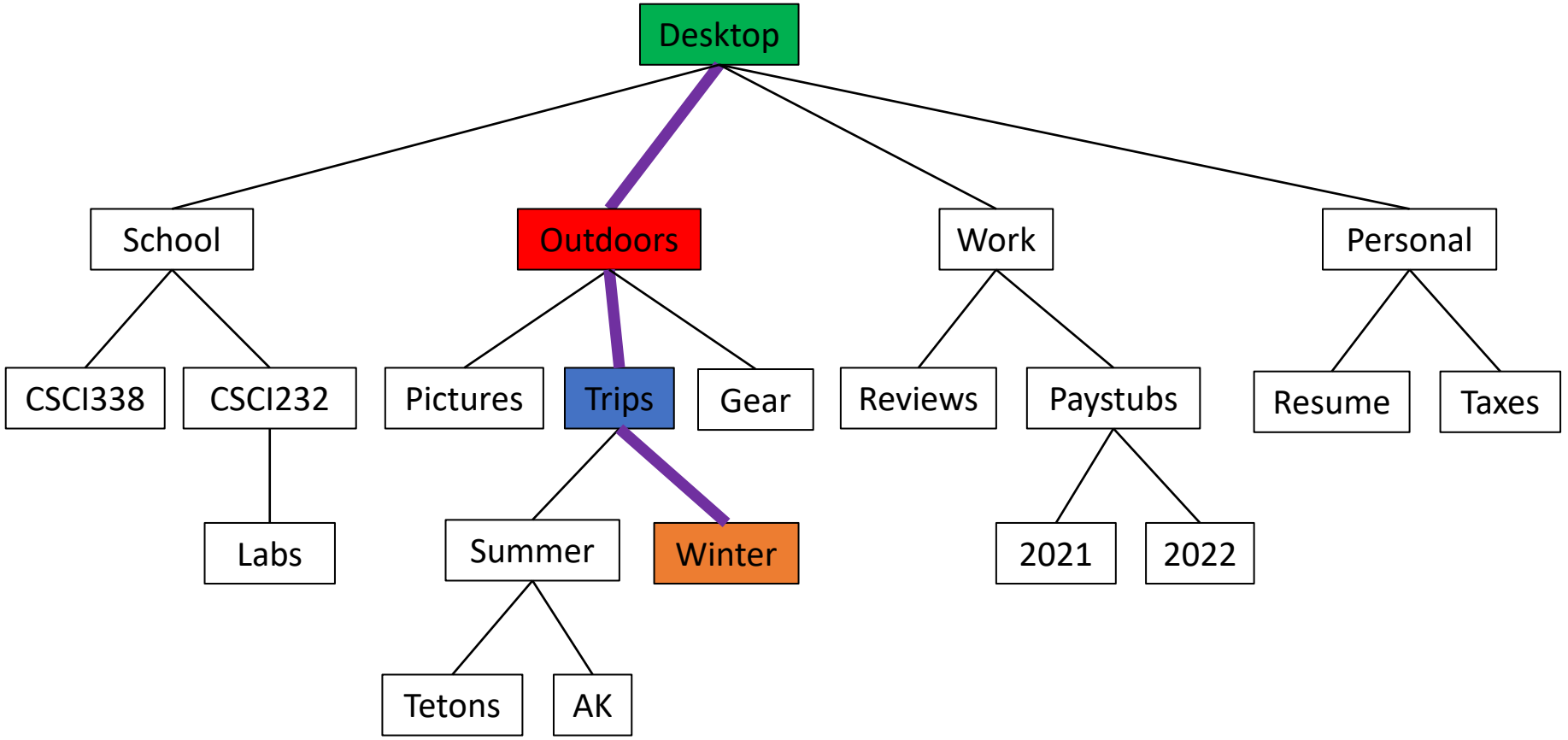
There is no edge between nodes that are not directly parent-child related.

# Trees - Definitions




**Path**: A sequence edge-connected nodes.

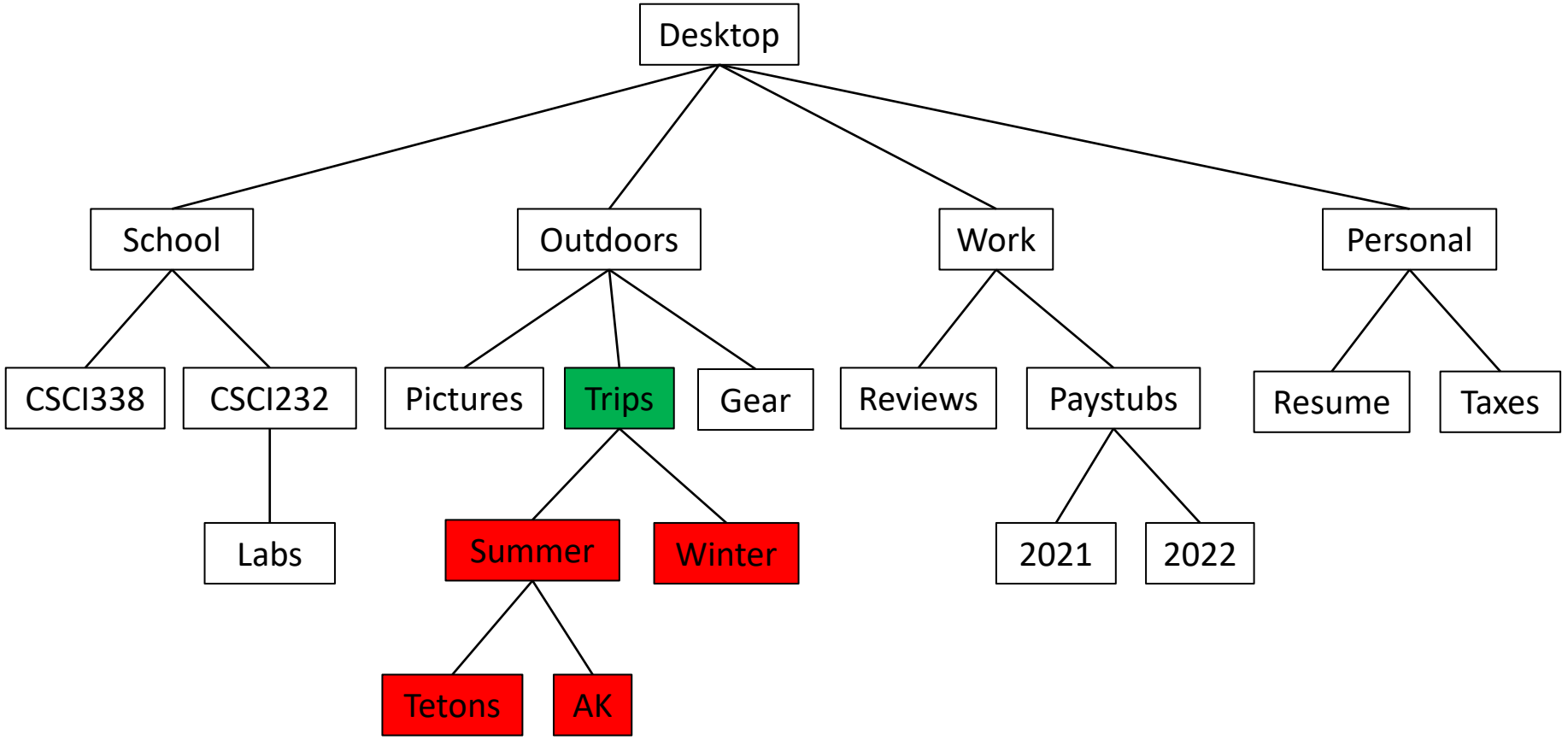
# Trees - Definitions




**Path**: A sequence edge-connected nodes.

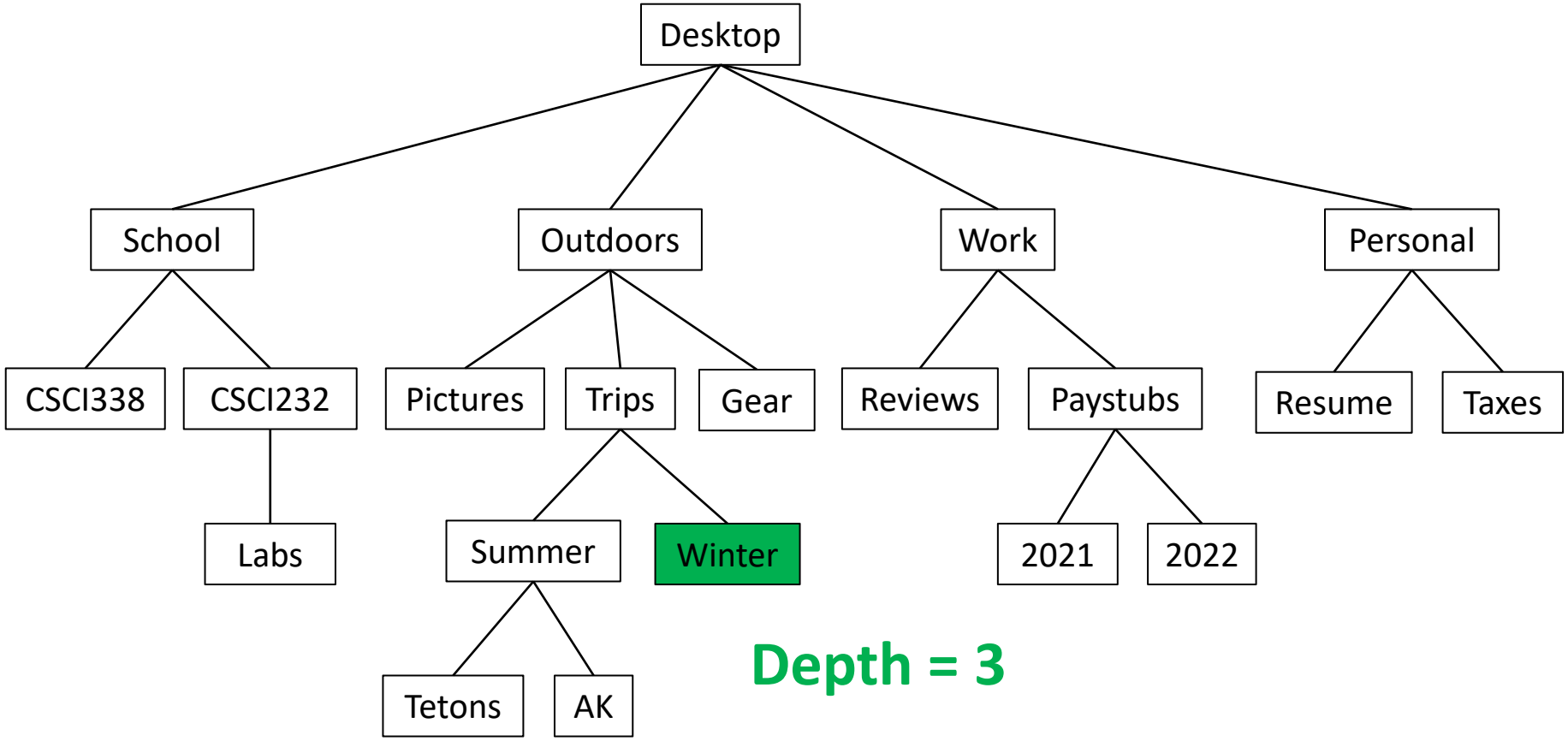
E.g., **Desktop**/**Outdoors**/**Trips**/**Winter**

# Trees - Definitions



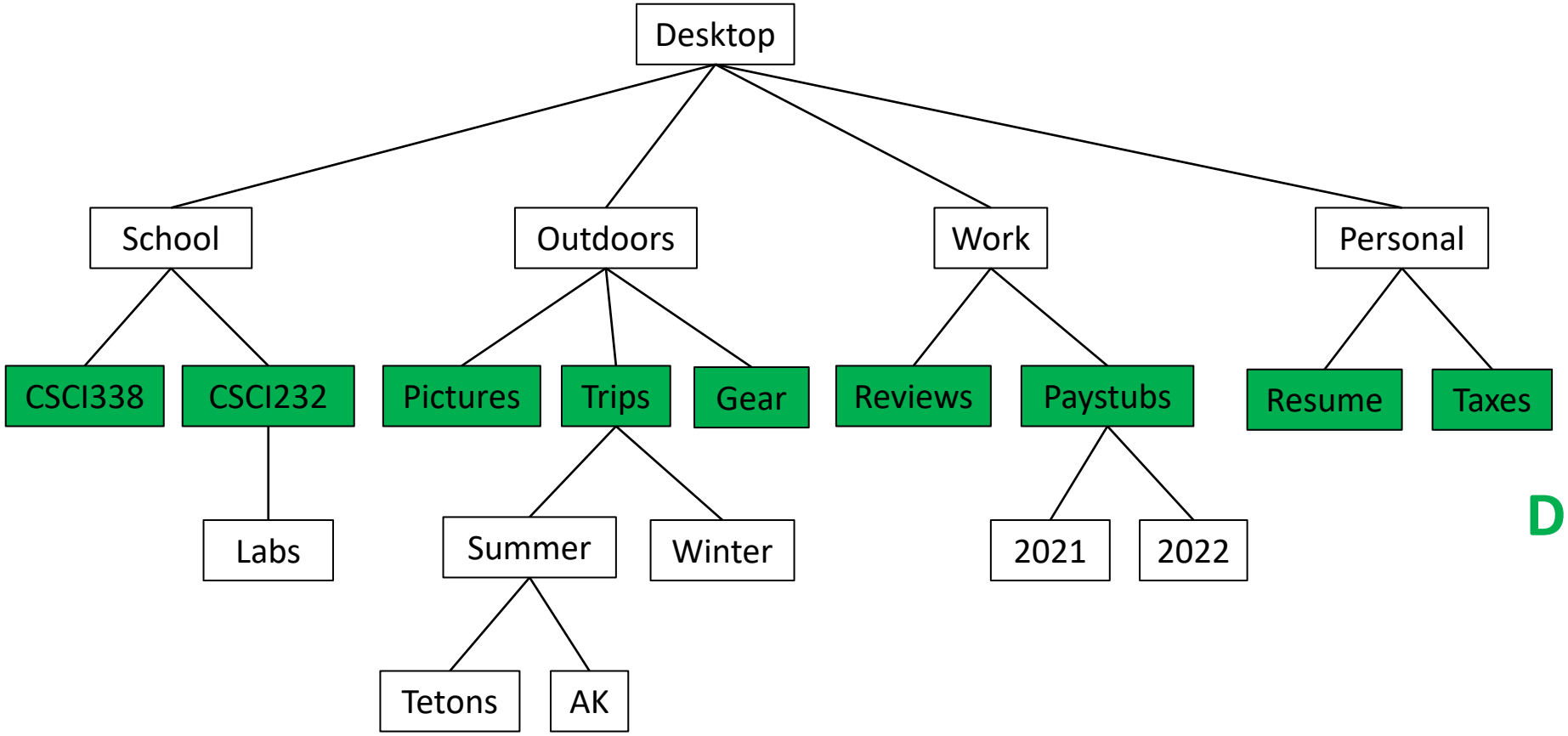

**Subtree**: A given **node** and all of its **descendants**.

# Trees - Definitions




**Depth**: For a given node, its depth is the number of edges in the (unique) path back to the root.

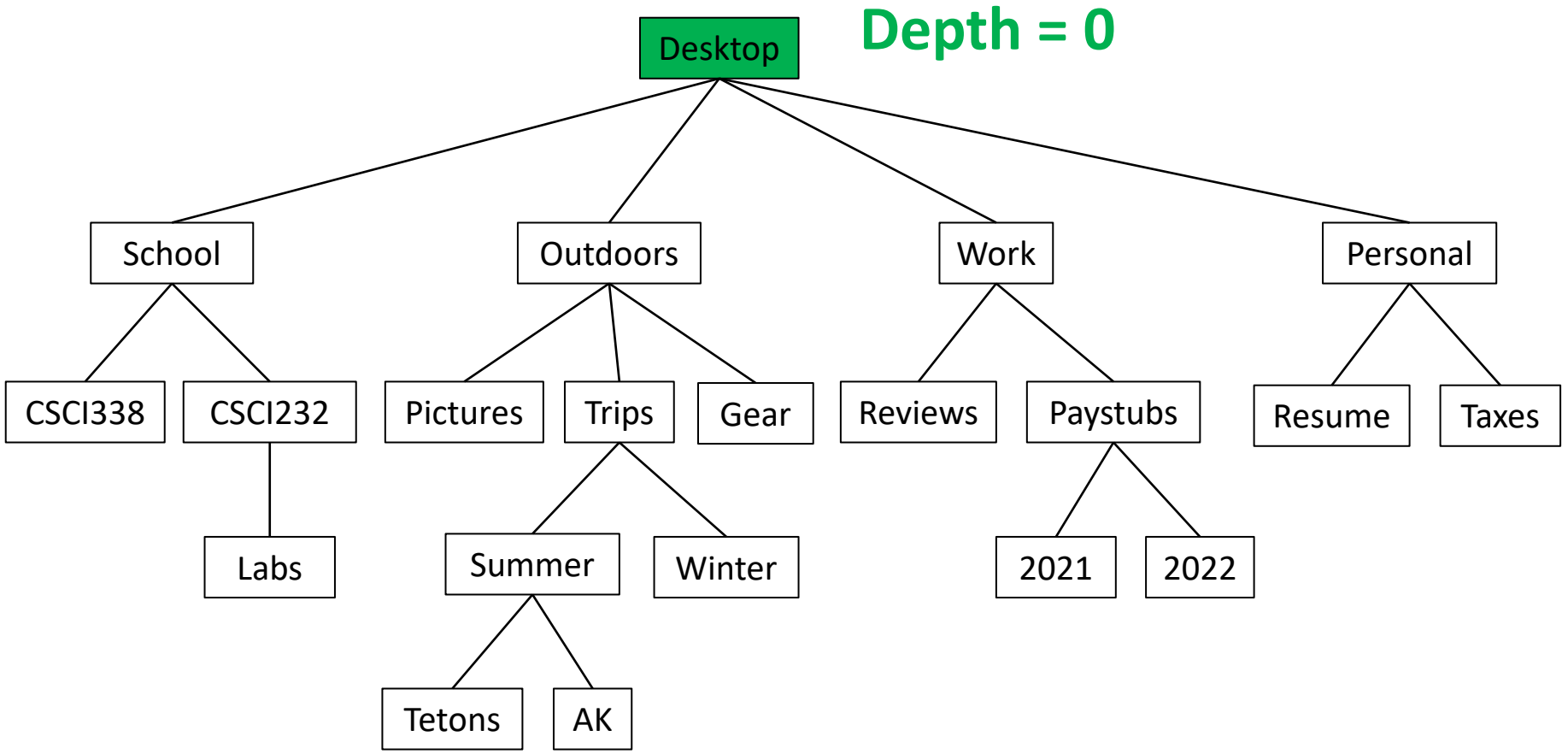
# Trees - Definitions




**Depth = 2**

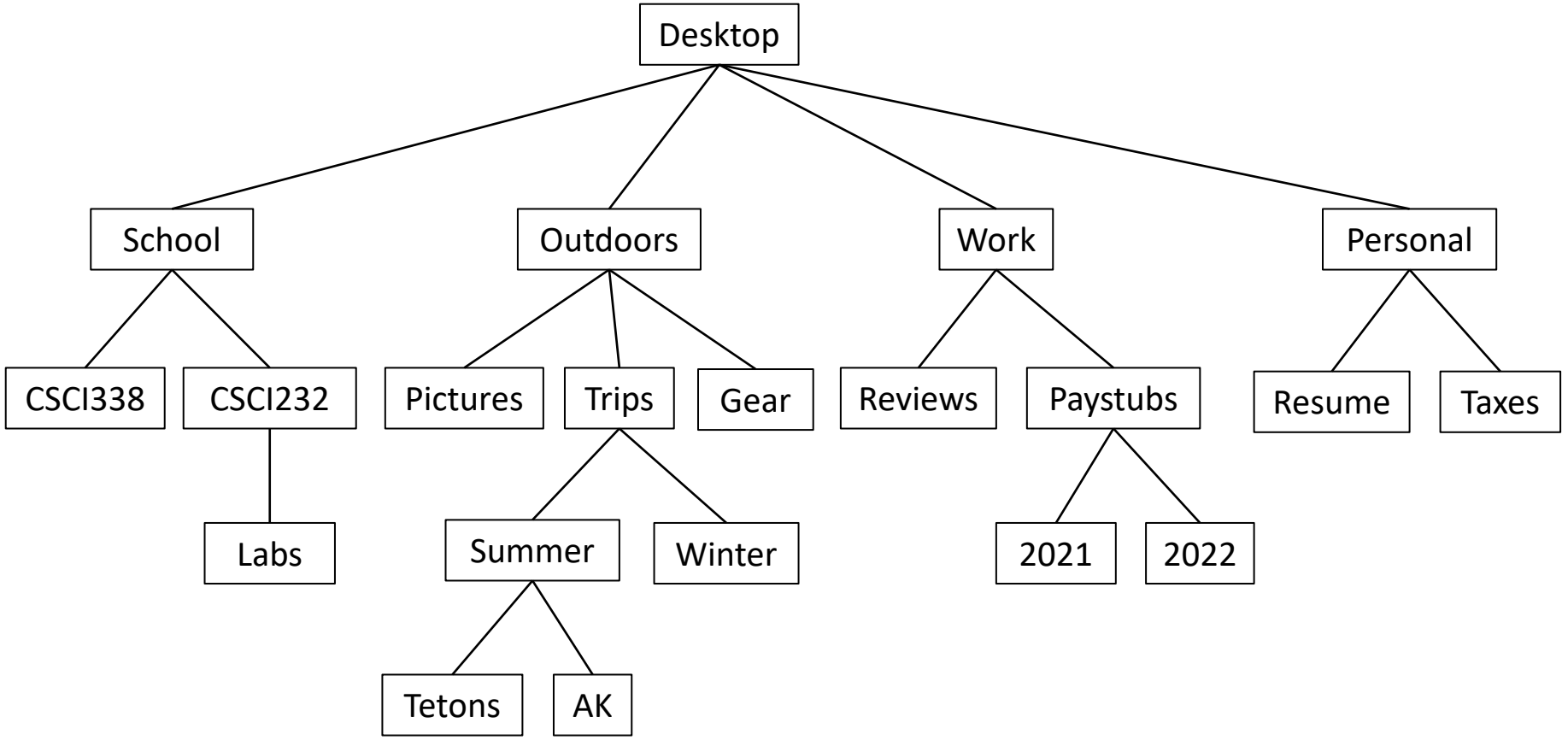
**Depth**: For a given node, its depth is the number of edges in the (unique) path back to the root.

# Trees - Definitions




**Depth**: For a given node, its depth is the number of edges in the (unique) path back to the root.

# Trees - Definitions




**Height**: The height of a tree is the maximum depth of any of its nodes.

# Trees - Operations

What kinds of operations do you think we will need to do on a tree?

# Trees - Operations

What kinds of operations do you think we will need to do on a tree?

- Insert a node.
- Remove a node.
- Get the children of a node.
- Get the parent of a node.
- Some sort of a traversal/search.
- Get depth/height.

# Trees - Operations

What kinds of operations do you think we will need to do on a tree?

- Insert a node.
- Remove a node.
- **Get the children of a node.**
- **Get the parent of a node.**
- **Some sort of a traversal/search.**
- **Get depth/height.**

**Some** of these operations don't depend on the purpose of the tree.

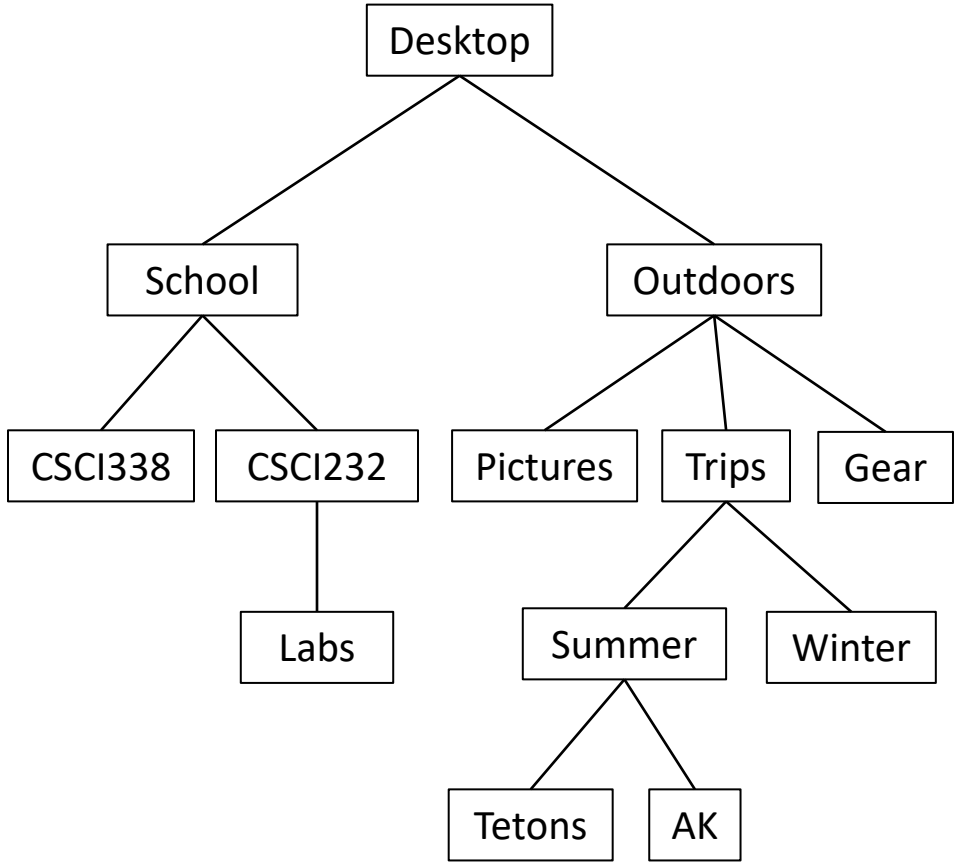
# Trees - Operations

What kinds of operations do you think we will need to do on a tree?

- **Insert a node.**
- **Remove a node.**
- Get the children of a node.
- Get the parent of a node.
- Some sort of a traversal/search.
- Get depth/height.

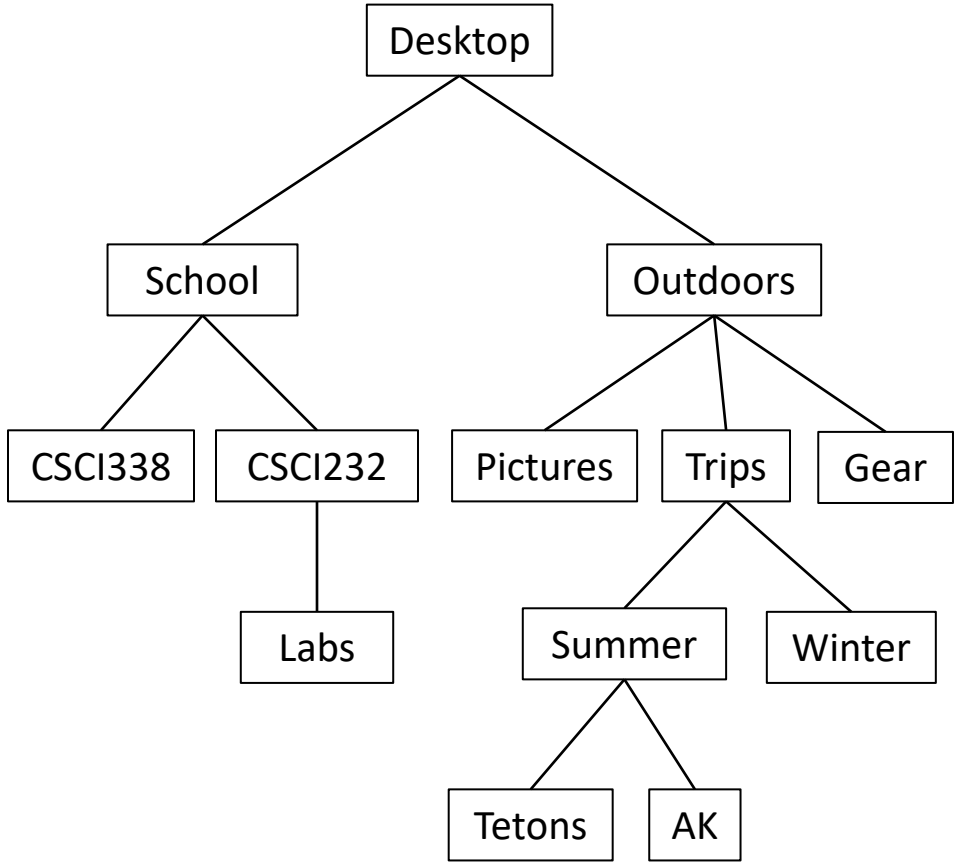
**Some** of these operations have implementations that depend on what the tree is supposed to do.

# Trees - Implementation



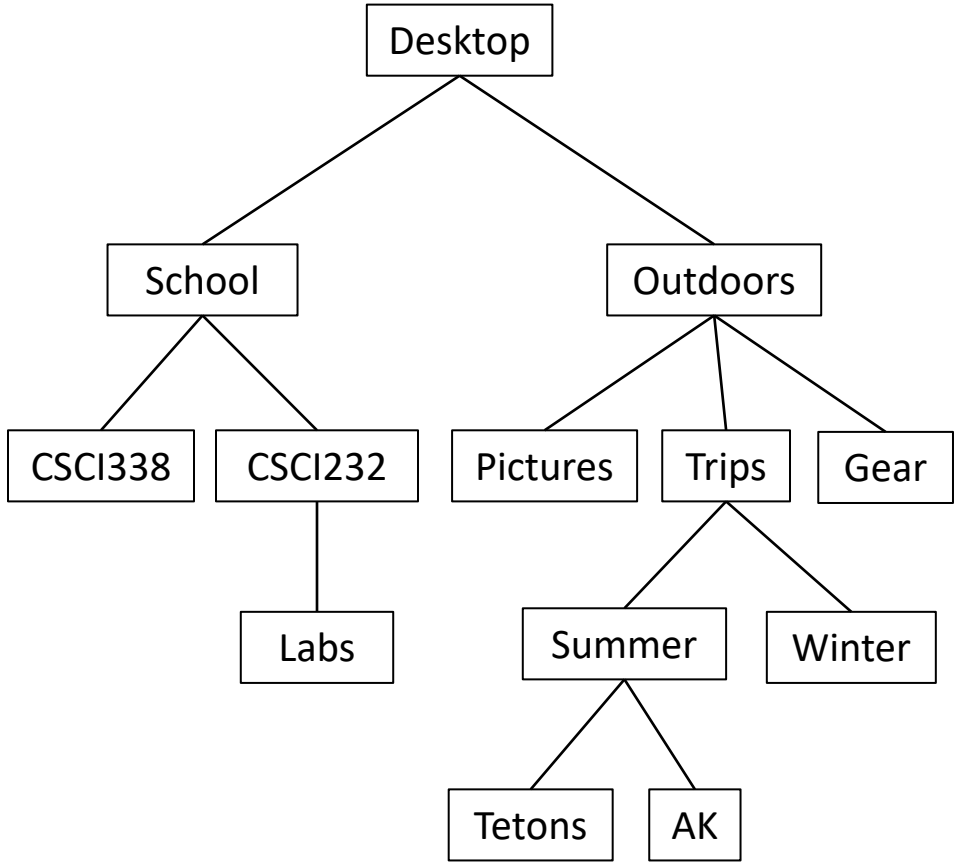

**How do we represent a tree in code?**

# Trees - Implementation



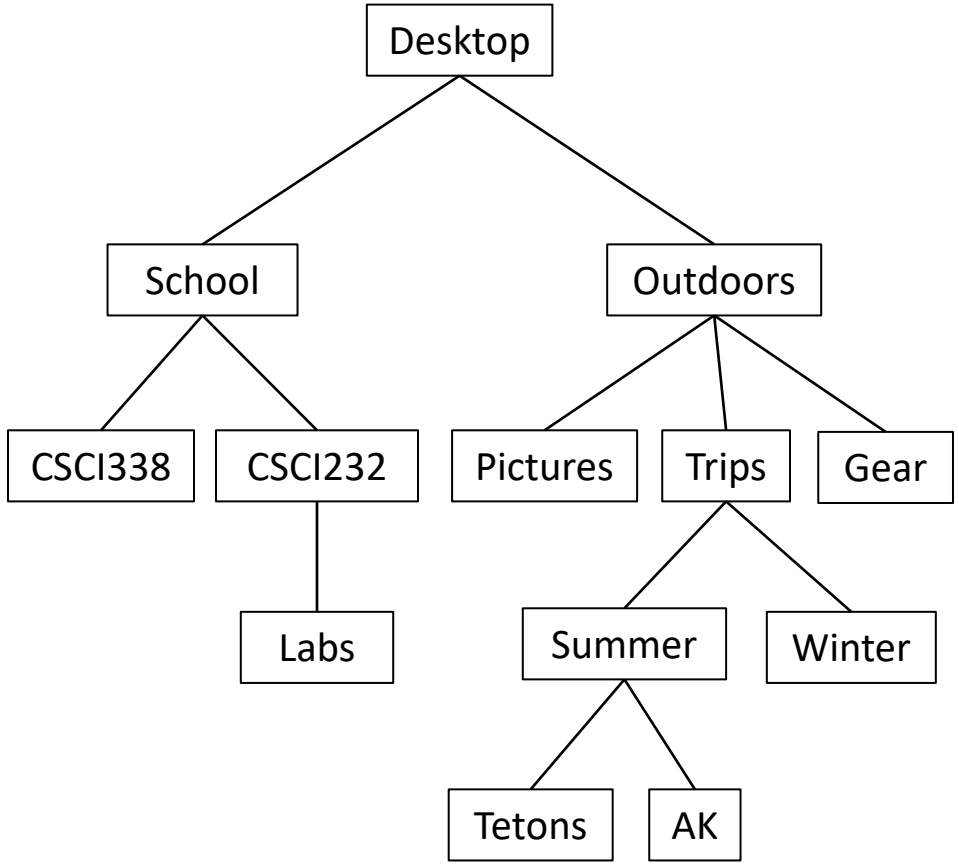

```
public class Node {



}
```

# Trees - Implementation



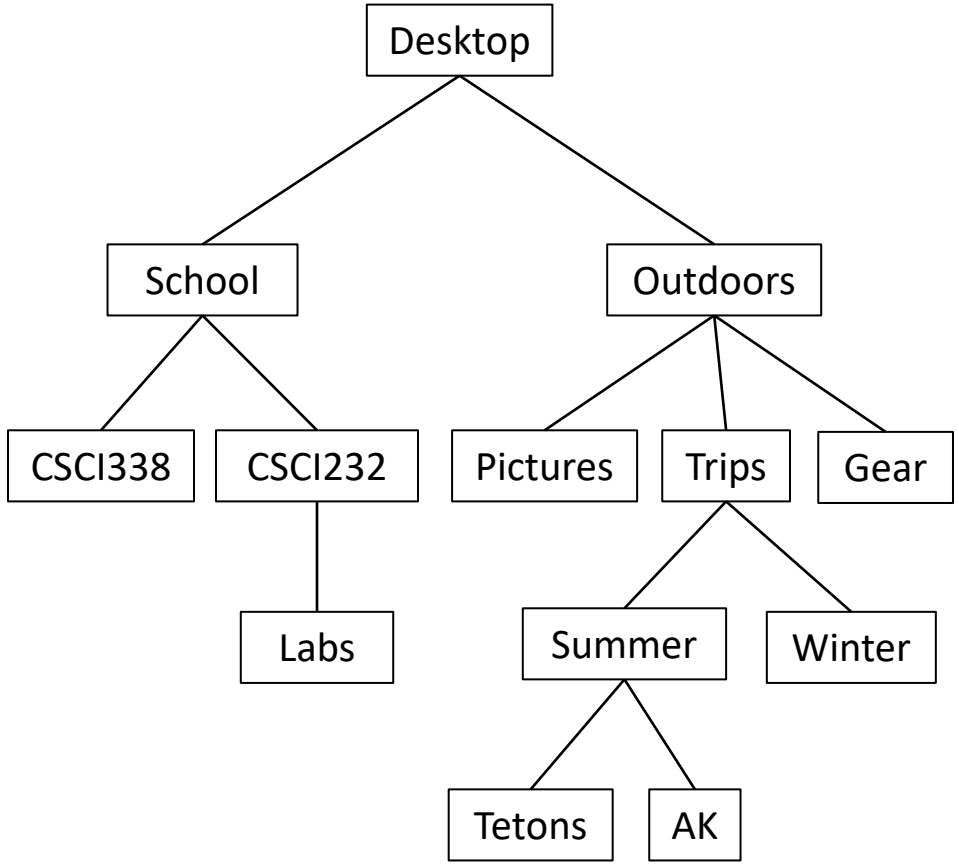

```
public class Node {

      instance variables?

}
```

# Trees - Implementation



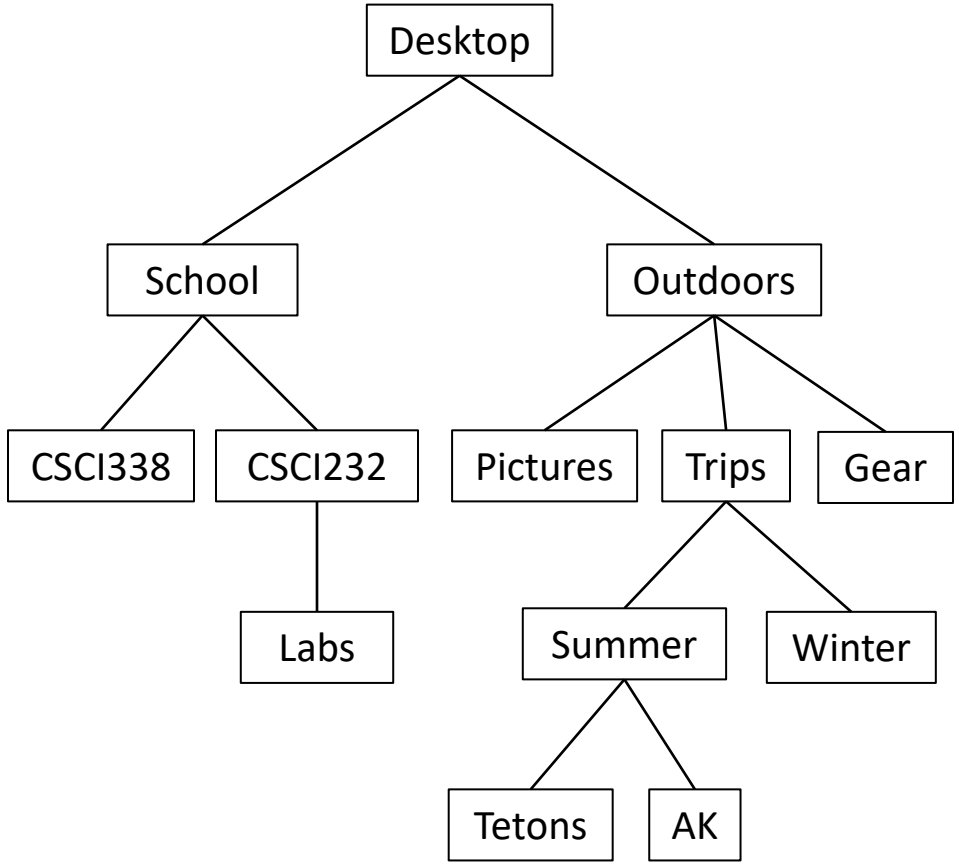

```
public class Node {

     private ???? parent;









}
```

# Trees - Implementation



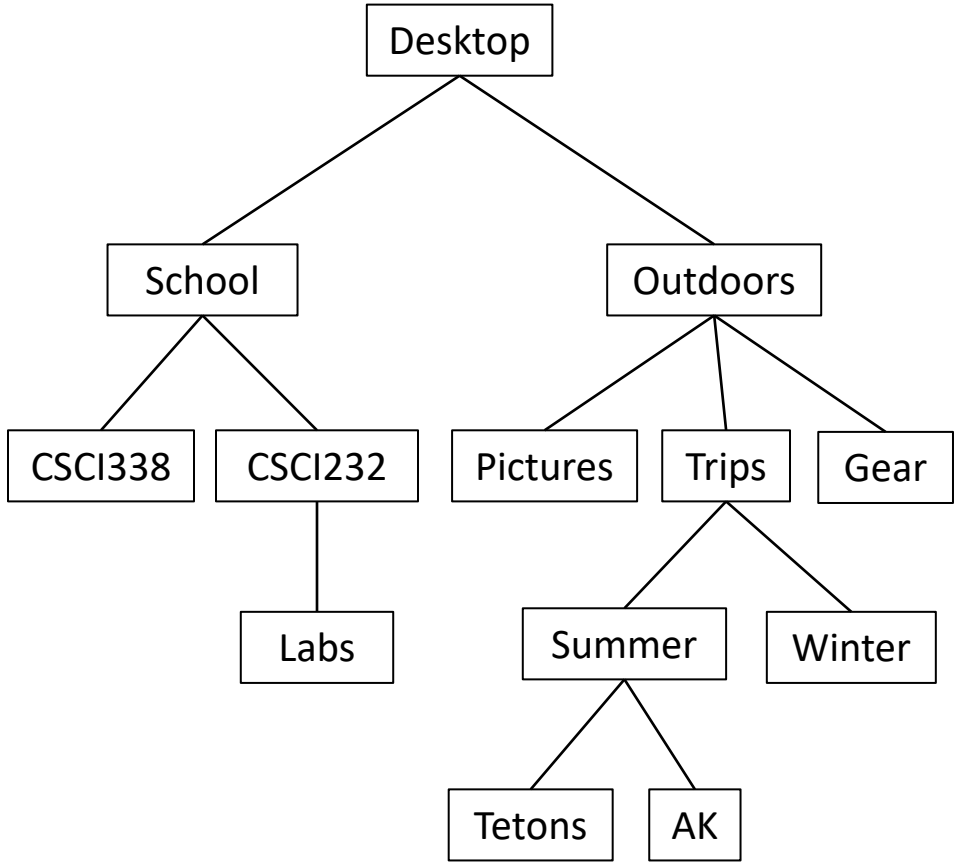

```
public class Node {

     private Node parent;




}
```

# Trees - Implementation



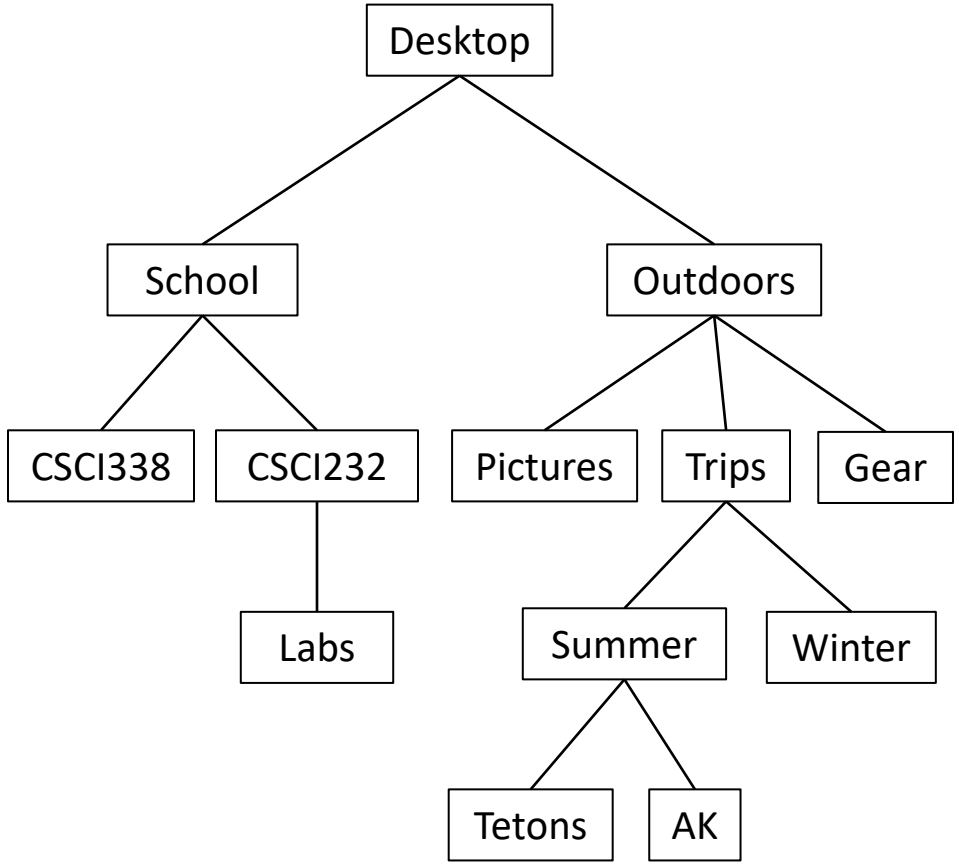

```
public class Node {

     private Node parent;
     private ?????????????? children;

}
```

# Trees - Implementation



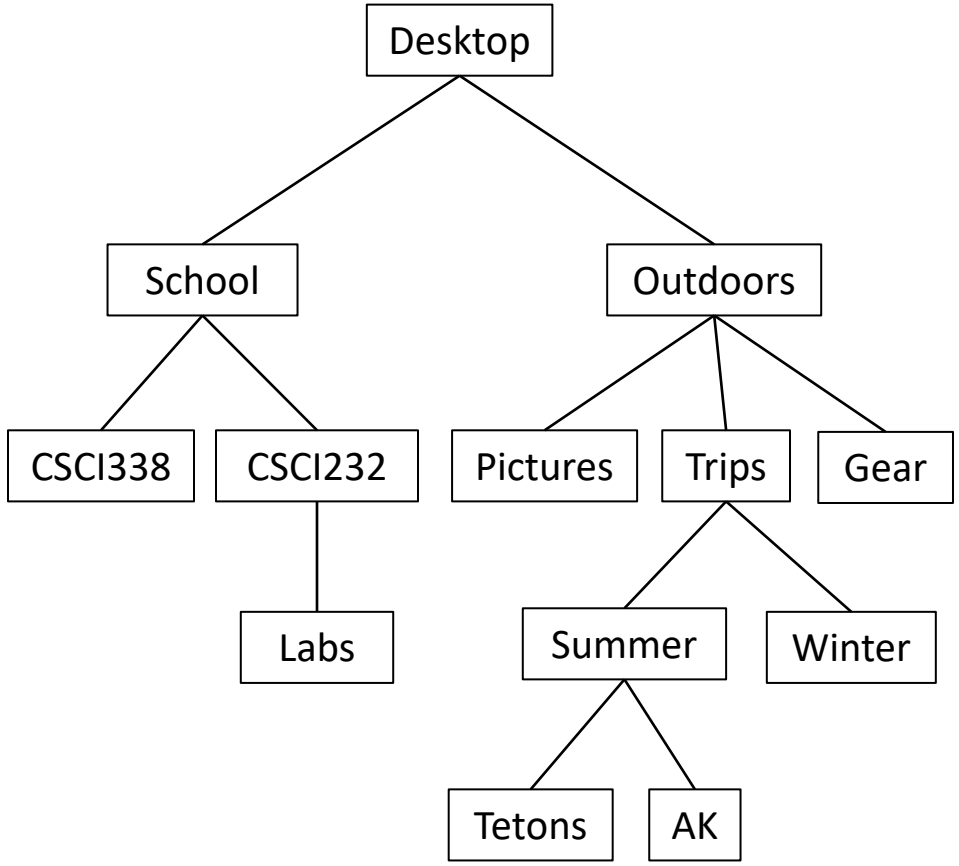

```
public class Node {

     private Node parent;
     private ArrayList<????> children;




}
```

# Trees - Implementation



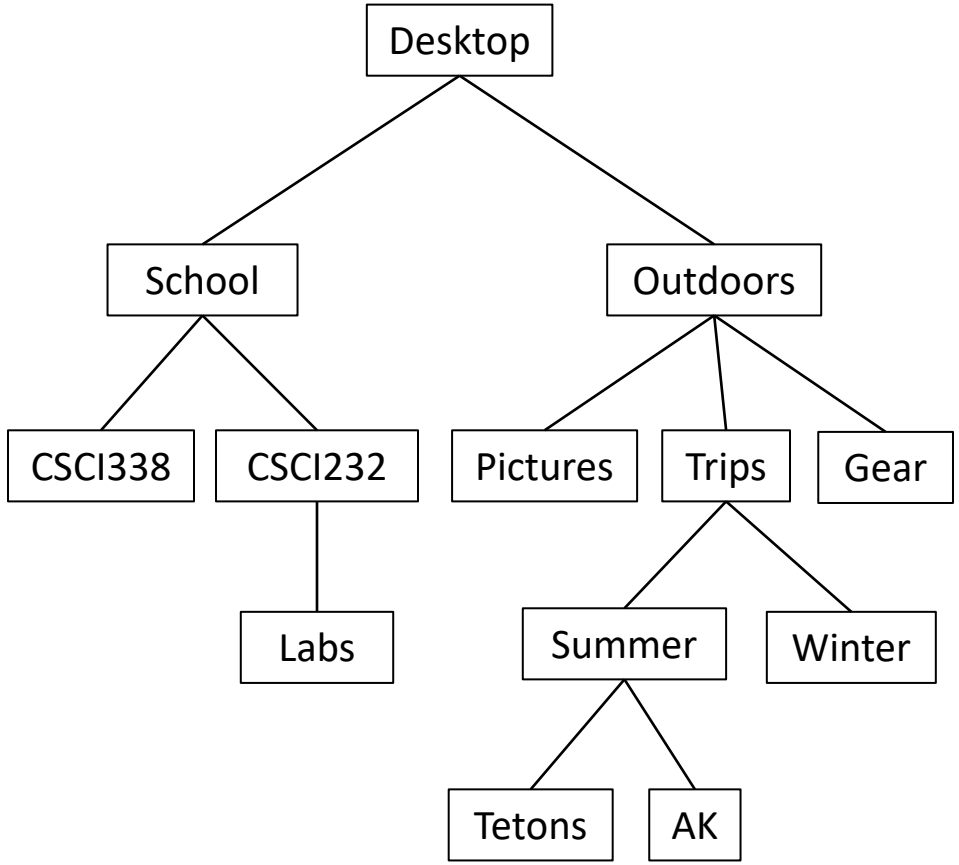

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;




}
```

# Trees - Implementation



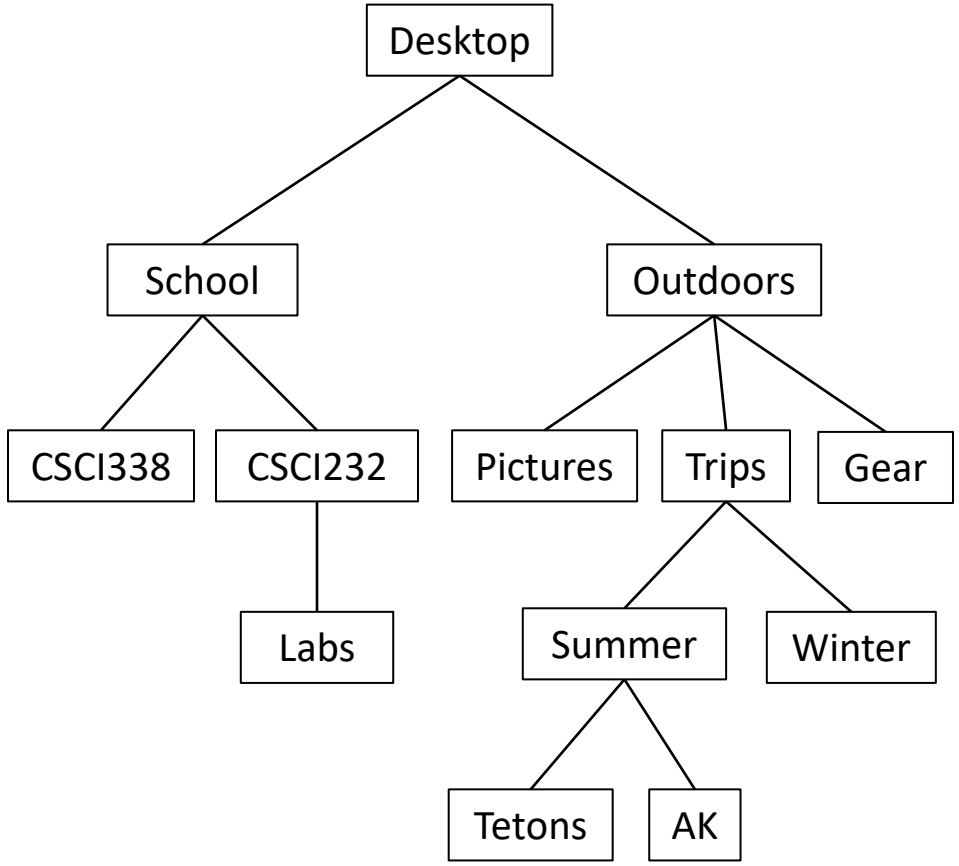

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     ????



}
```

# Trees - Implementation



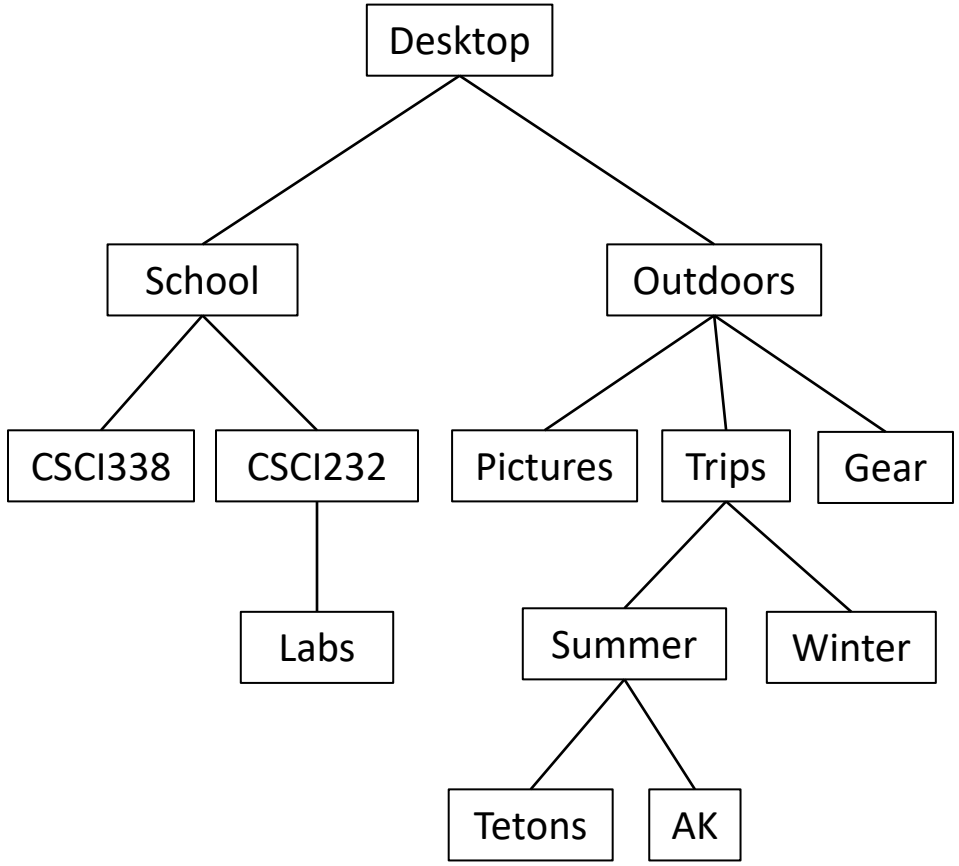

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;









}
```

# Trees - Implementation



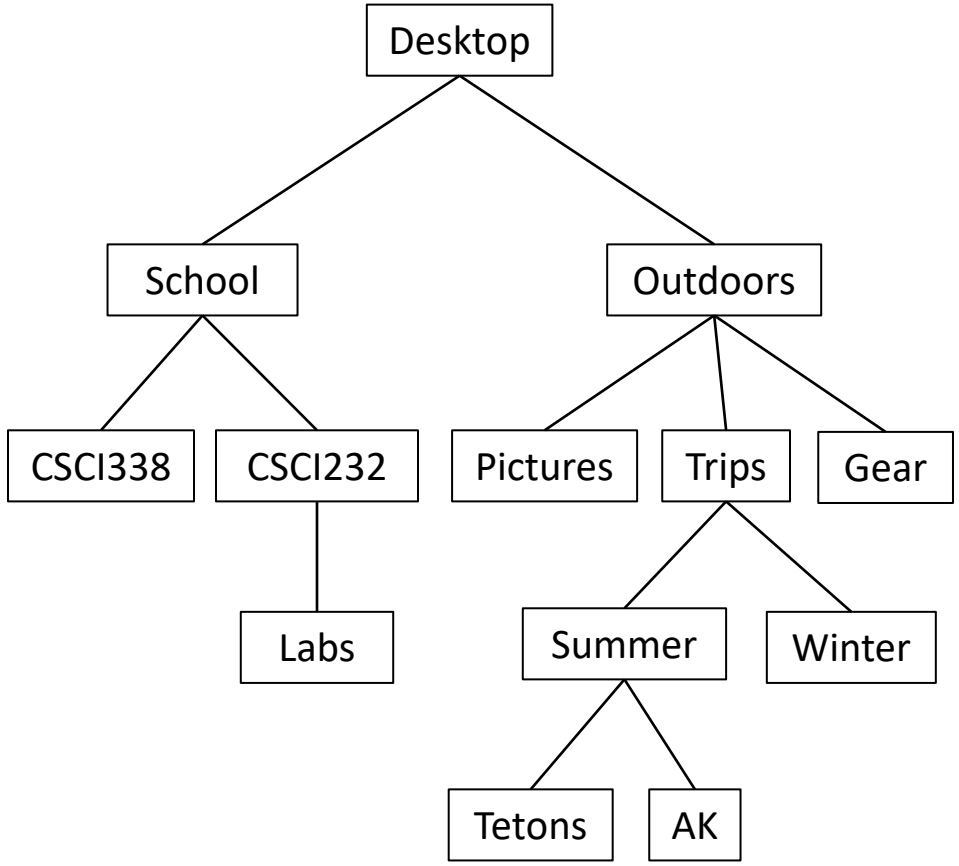

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(?????? ????) {


     }









}
```

# Trees - Implementation



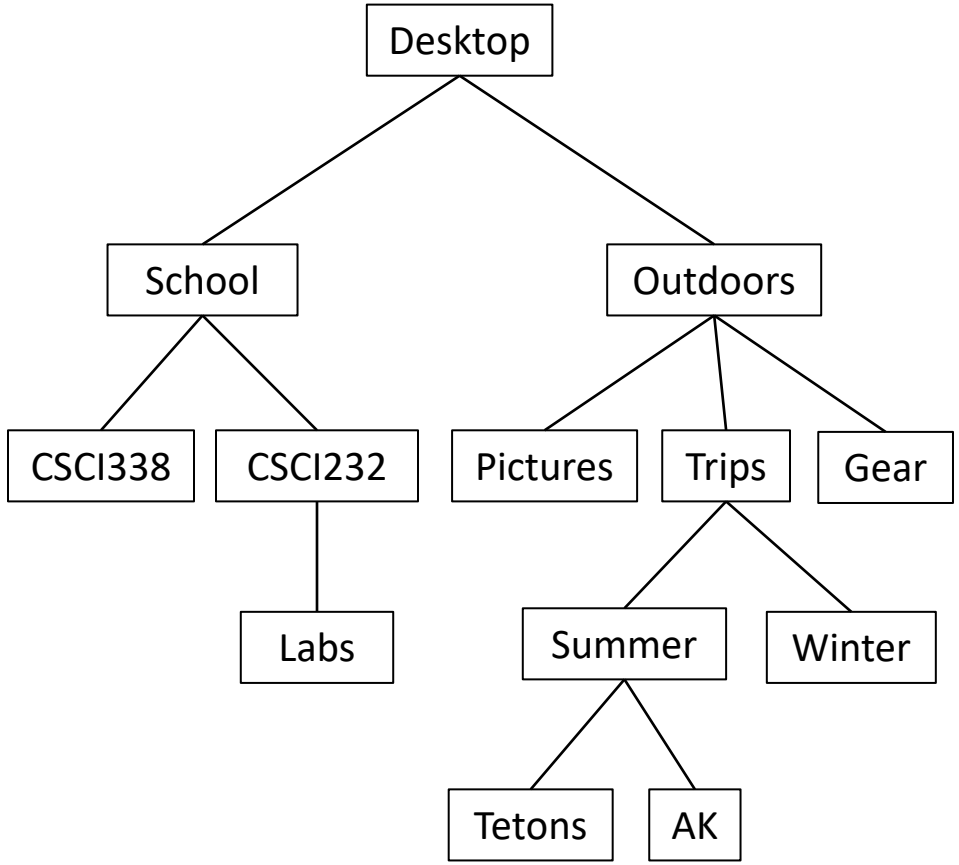

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          ????

     }

}
```

# Trees - Implementation



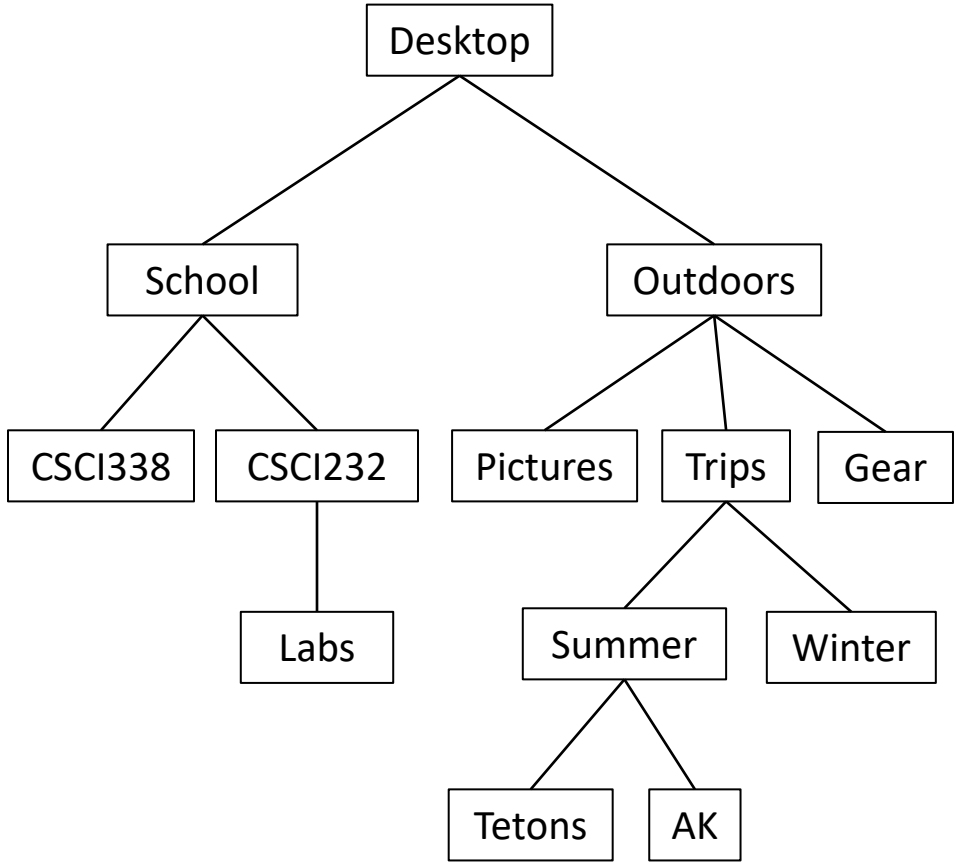

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          ????
     }




}
```

# Trees - Implementation



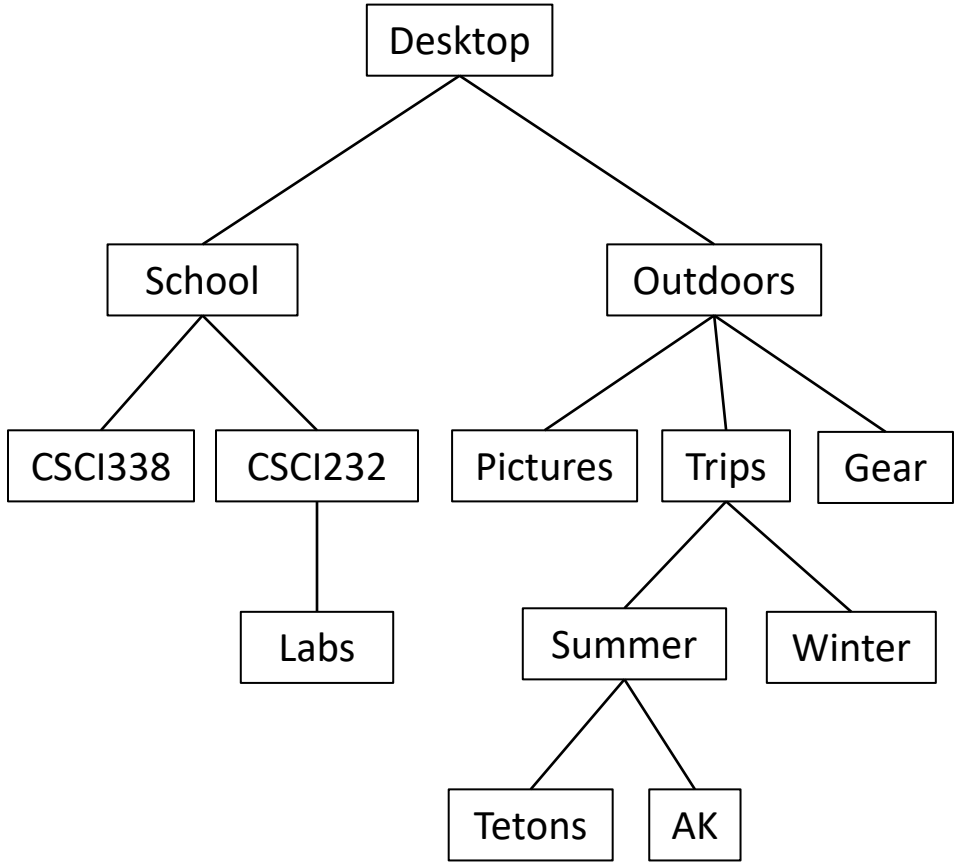

```
public class Node {

	private Node parent;
	private ArrayList<Node> children;

	private String name;

	public Node(String name) {
		this.name = name;
		children = new ArrayList<>();
	}









}
```

# Trees - Implementation



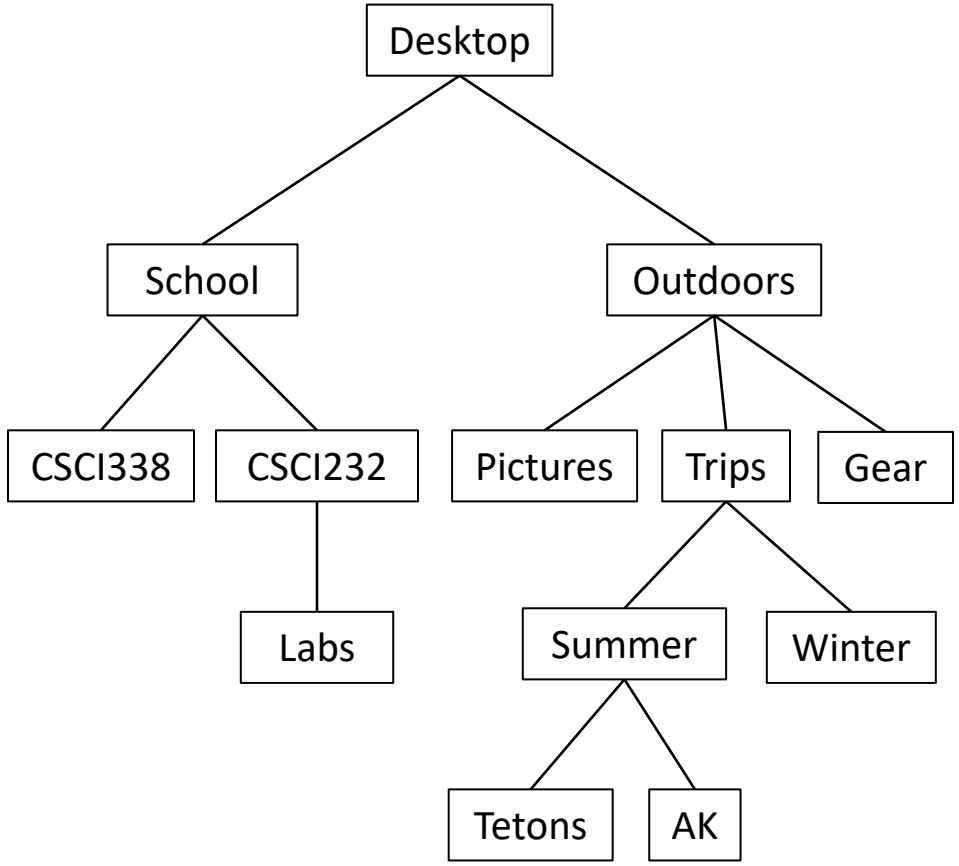

```
public class Node {

	private Node parent;
	private ArrayList<Node> children;

	private String name;

	public Node(String name) {
		this.name = name;
		children = new ArrayList<>();
	}

	// getName()
	// getParent()
	// getChildren()




}
```

# Trees - Implementation



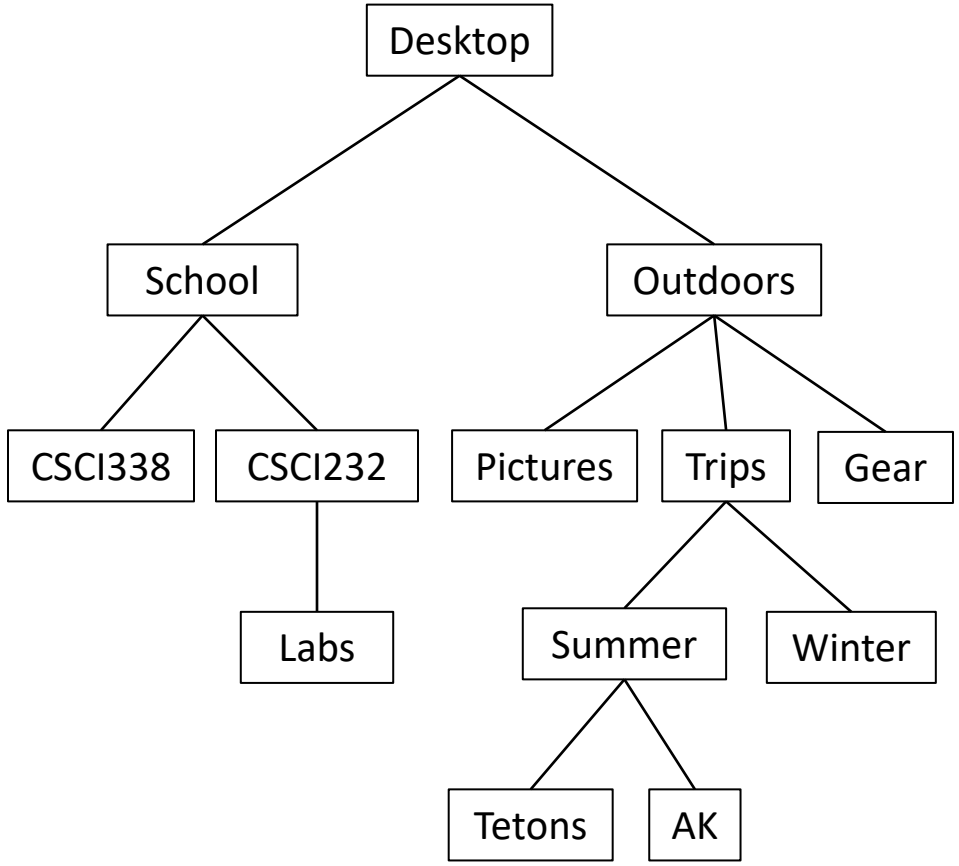

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          children = new ArrayList<>();
     }

     // getName()
     // getParent()
     // getChildren()

     // setParent()



}
```

# Trees - Implementation



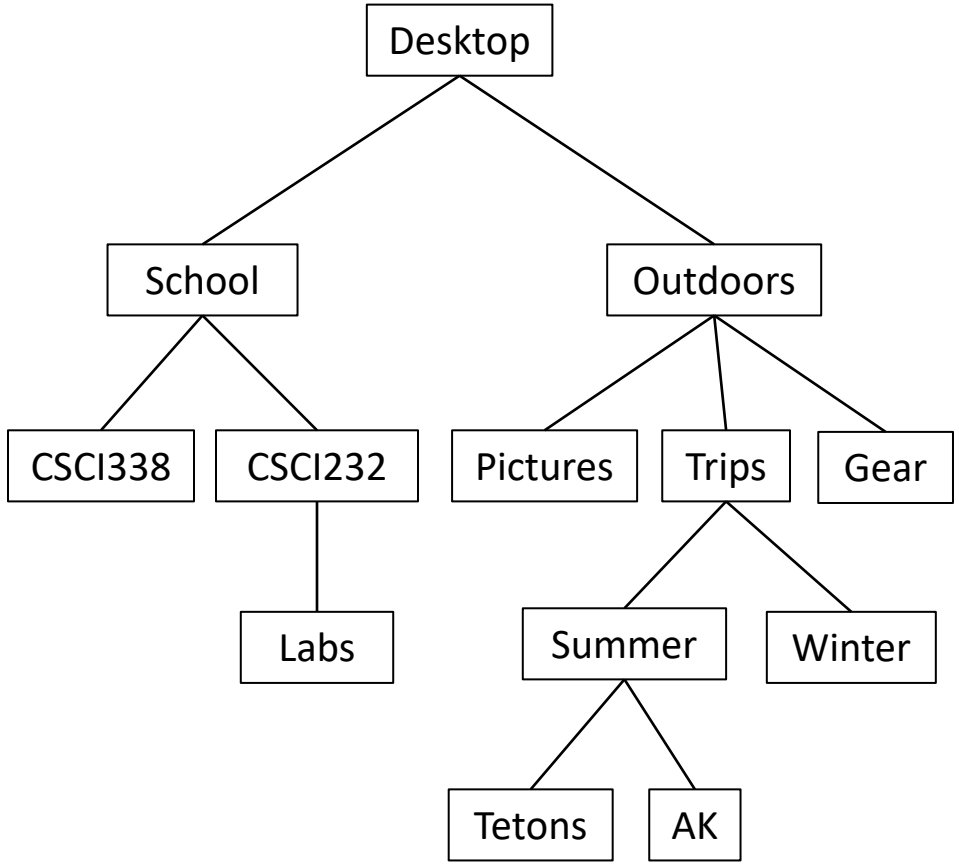

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          children = new ArrayList<>();
     }

     // getName()
     // getParent()
     // getChildren()

     // setParent()

     public ???? addChild(???? ?????) {

     }
}
```

# Trees - Implementation



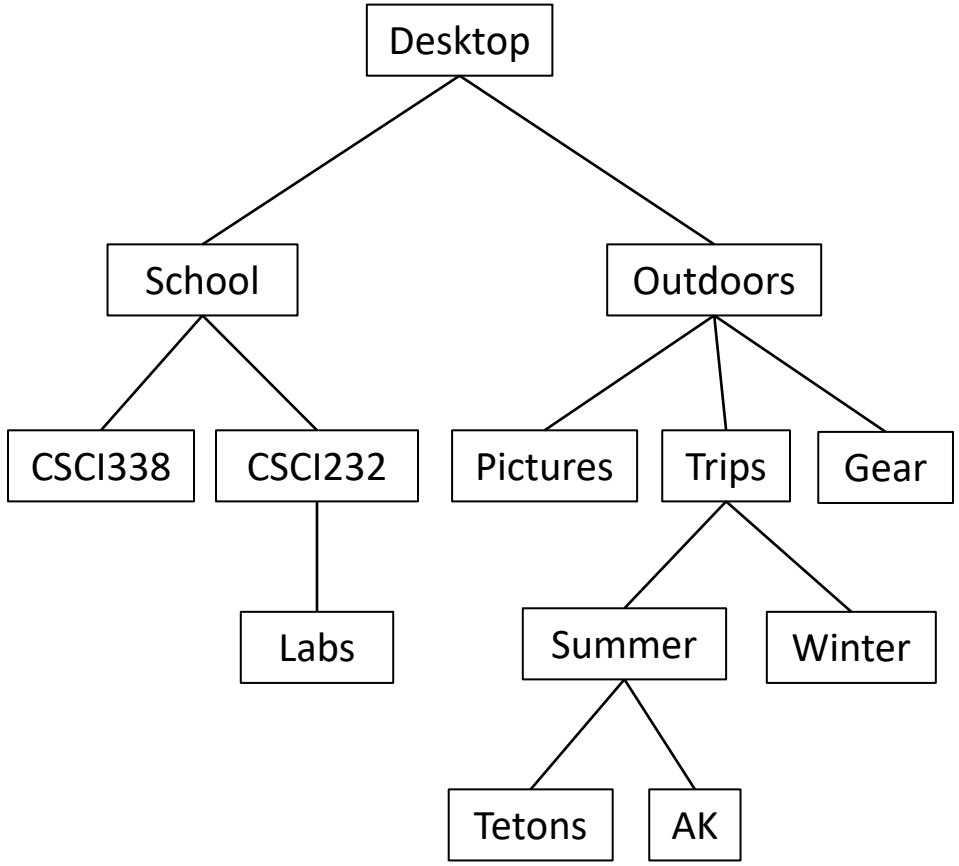

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          children = new ArrayList<>();
     }

     // getName()
     // getParent()
     // getChildren()

     // setParent()

     public ???? addChild(Node child) {
          ???
     }
}
```

# Trees - Implementation



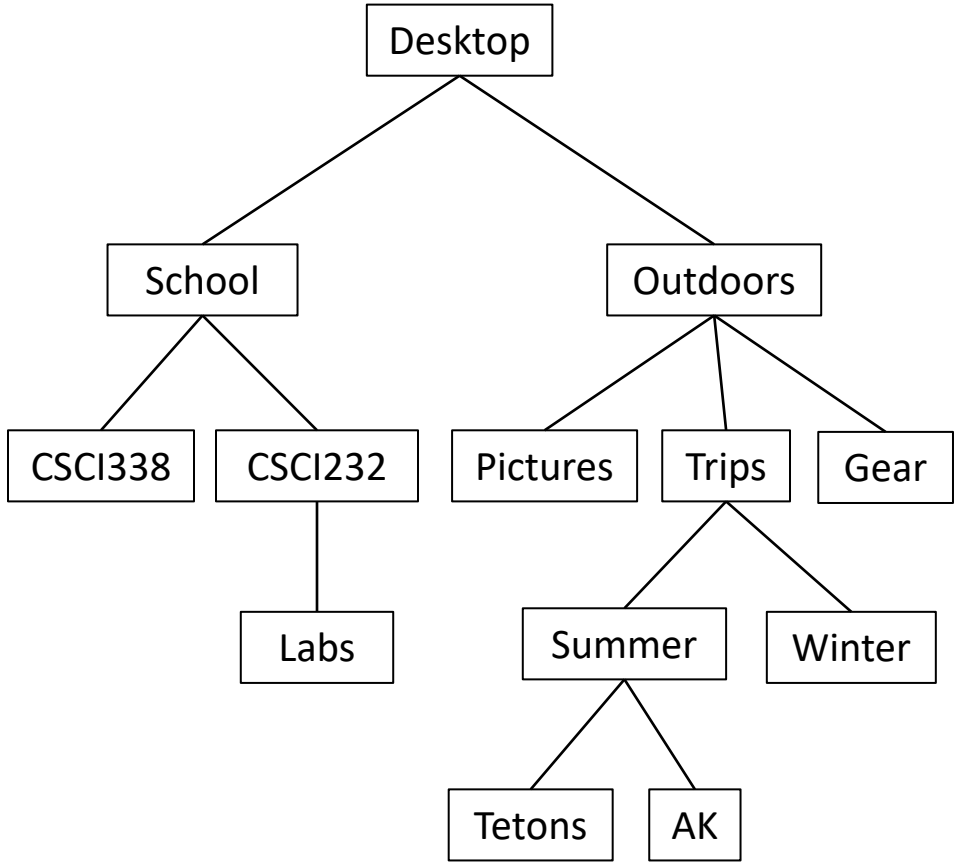

```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          children = new ArrayList<>();
     }

     // getName()
     // getParent()
     // getChildren()

     // setParent()

     public void addChild(Node child) {
          ???
     }
}
```

# Trees - Implementation




```
public class Node {

     private Node parent;
     private ArrayList<Node> children;

     private String name;

     public Node(String name) {
          this.name = name;
          children = new ArrayList<>();
     }

     // getName()
     // getParent()
     // getChildren()

     // setParent()

     public void addChild(Node child) {
          children.add(child);
     }
}
```

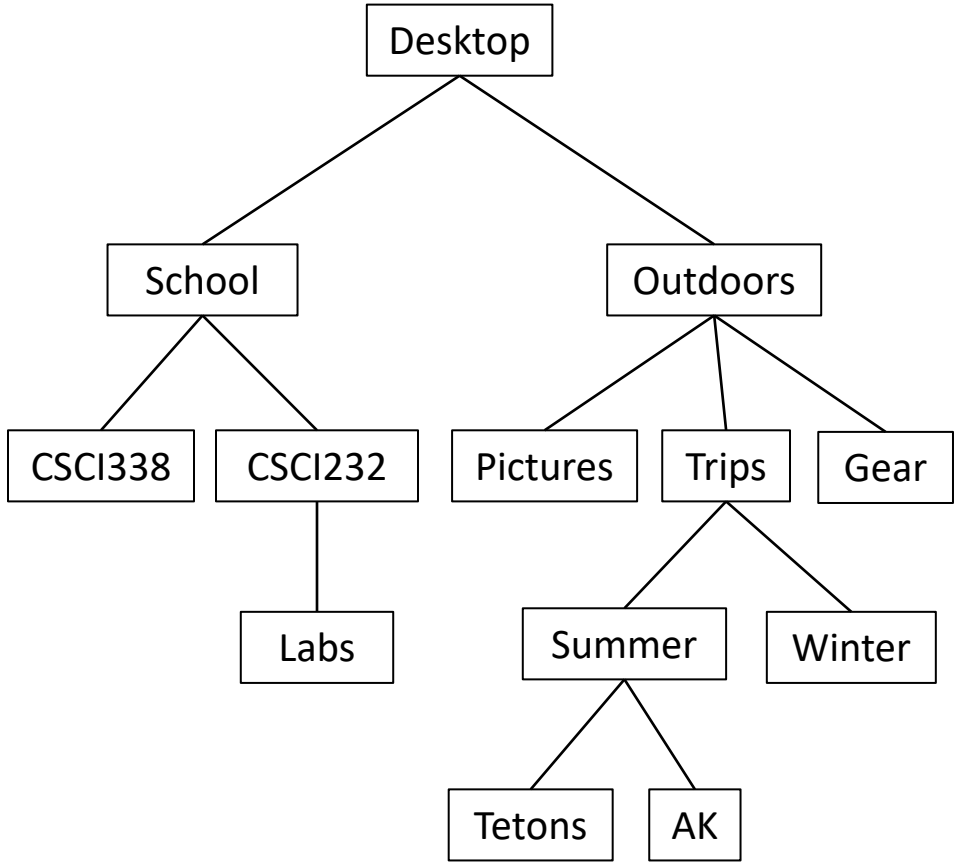
# Trees - Implementation




```
public class Node {
     ...
}
```

Represent individual data elements and their local relationships

# Trees - Implementation

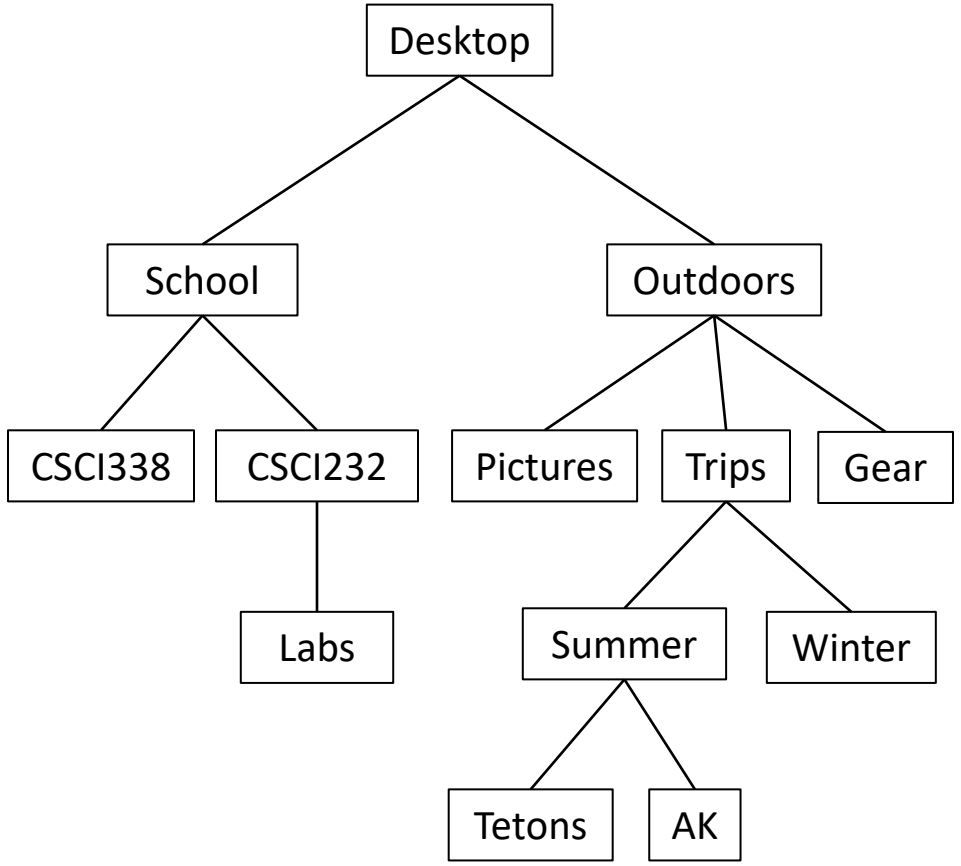



```
public class Node {
      ...
}
```

Represent individual data elements and their local relationships

**Now what?**

# Trees - Implementation



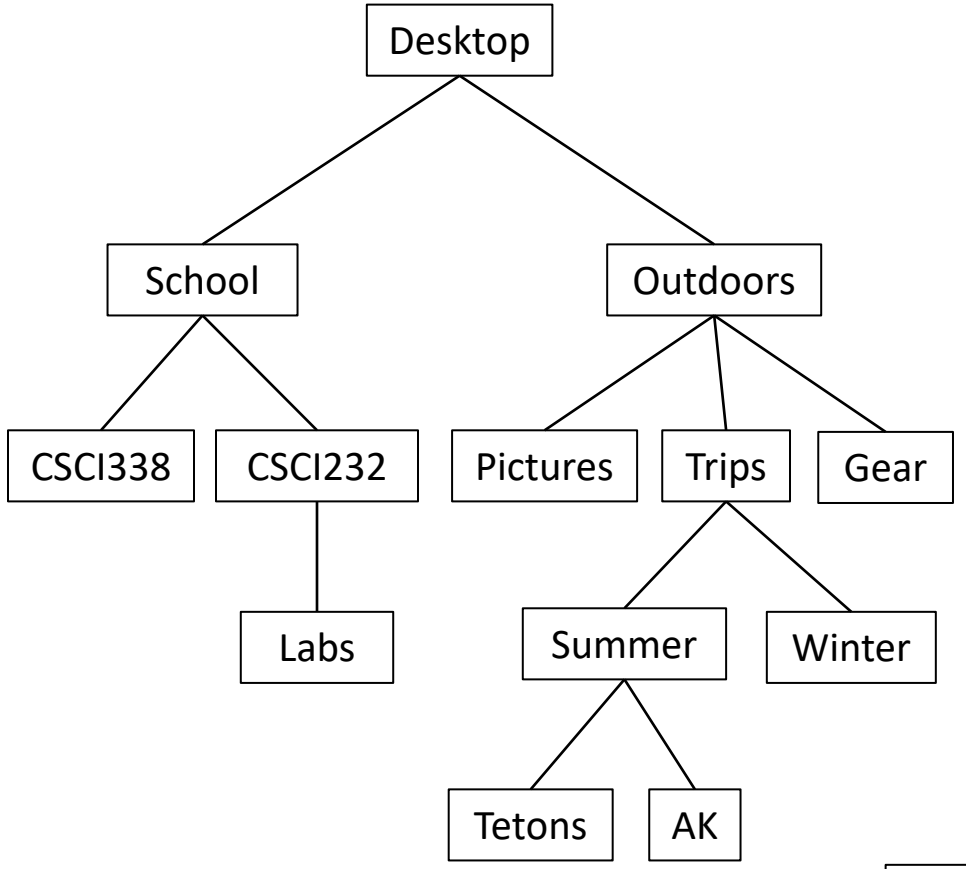

```
public class Node {
     ...
}
```

Represent individual data elements and their local relationships

```
public class Tree {
     ...
}
```

Represent the collection of data elements (tree). Insertion, deletion, navigation.

# Trees - Implementation




```
public class Node {
     ...
}
```

Represent individual data elements and their local relationships

```
public class Tree {
     ...
}
```

Represent the collection of data elements (tree). Insertion, deletion, navigation.

```
public class TreeManager {
     ...
}
```

Interface between user commands and tree object operations.