CSCI 232:
Data Structures and Algorithms

More Trees, Tree Traversal

Reese Pearsall

Summer 2023

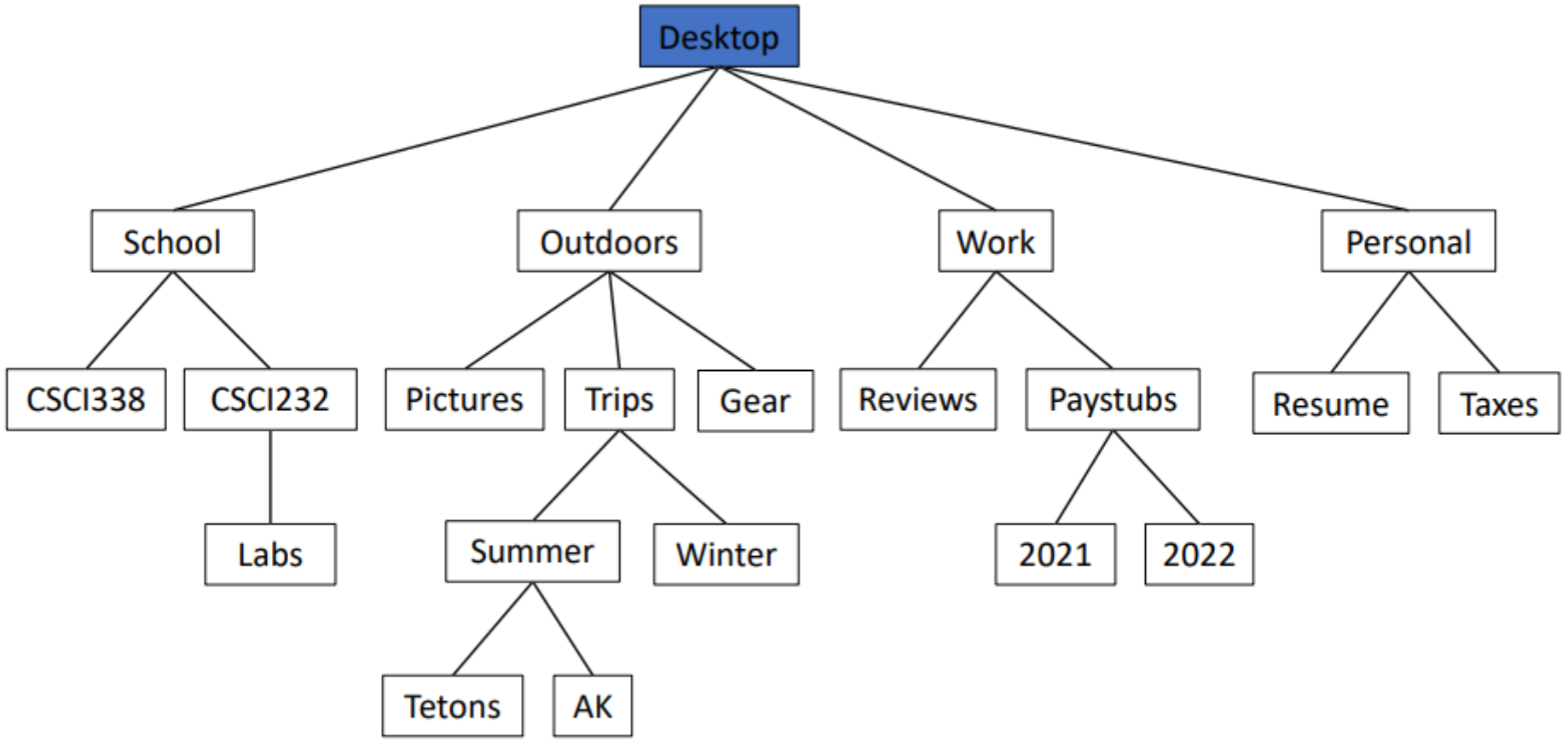
Announcements

Lab 1 due tonight

Lab 2 posted: Due Monday @ 11:59 PM

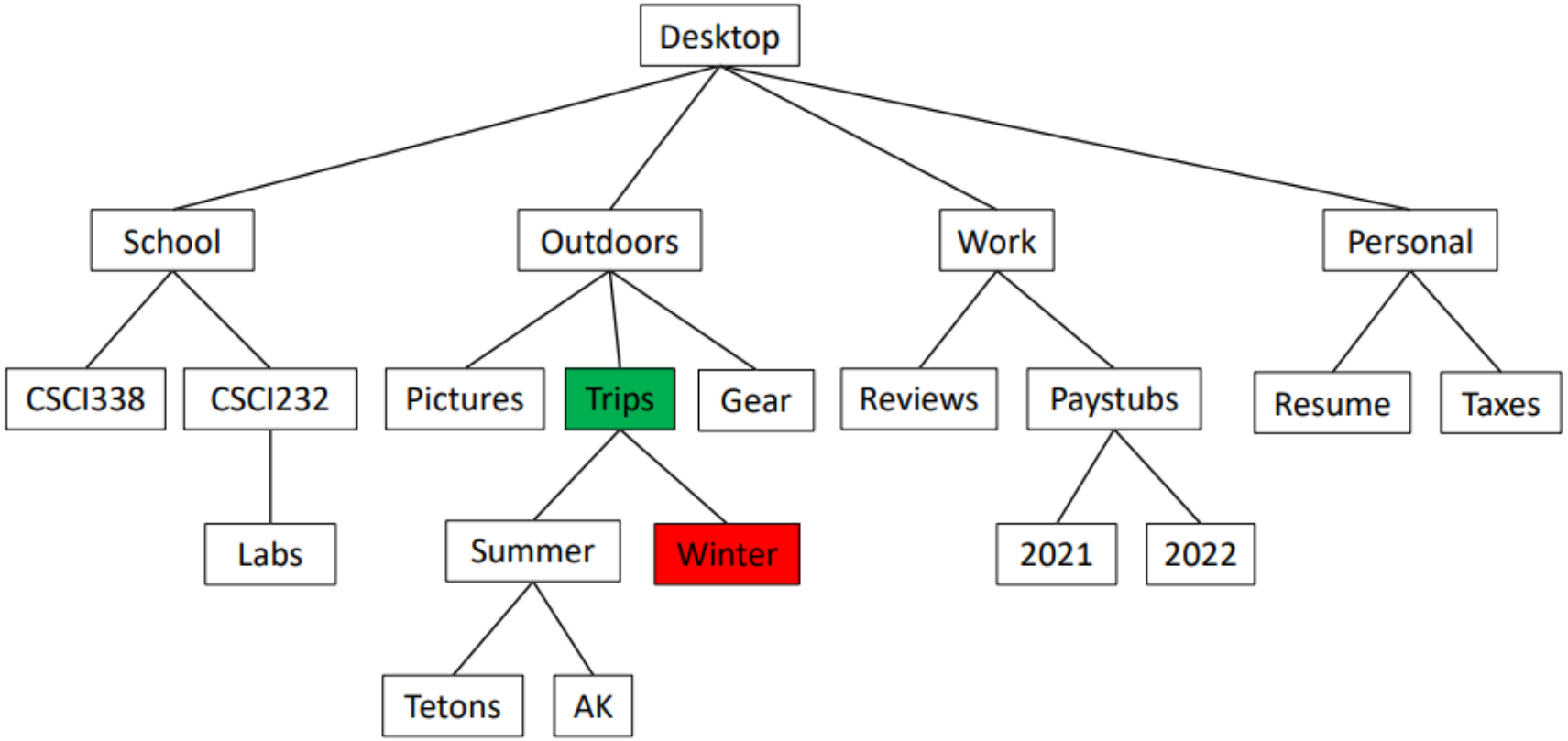
First program will be posted soon

Trees - Definitions



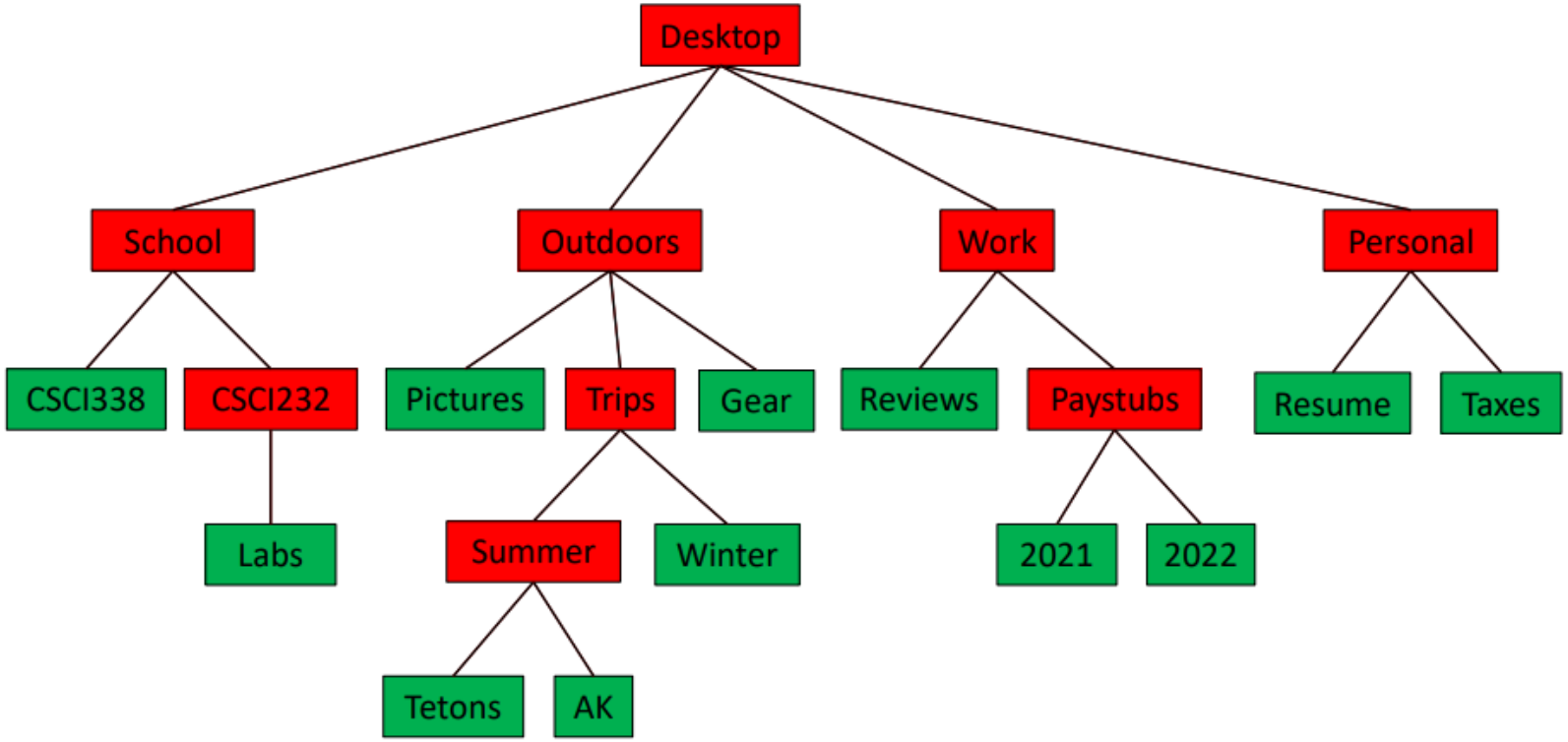
Root: The **node** at the top of hierarchy.

Trees - Definitions



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

Trees - Definitions

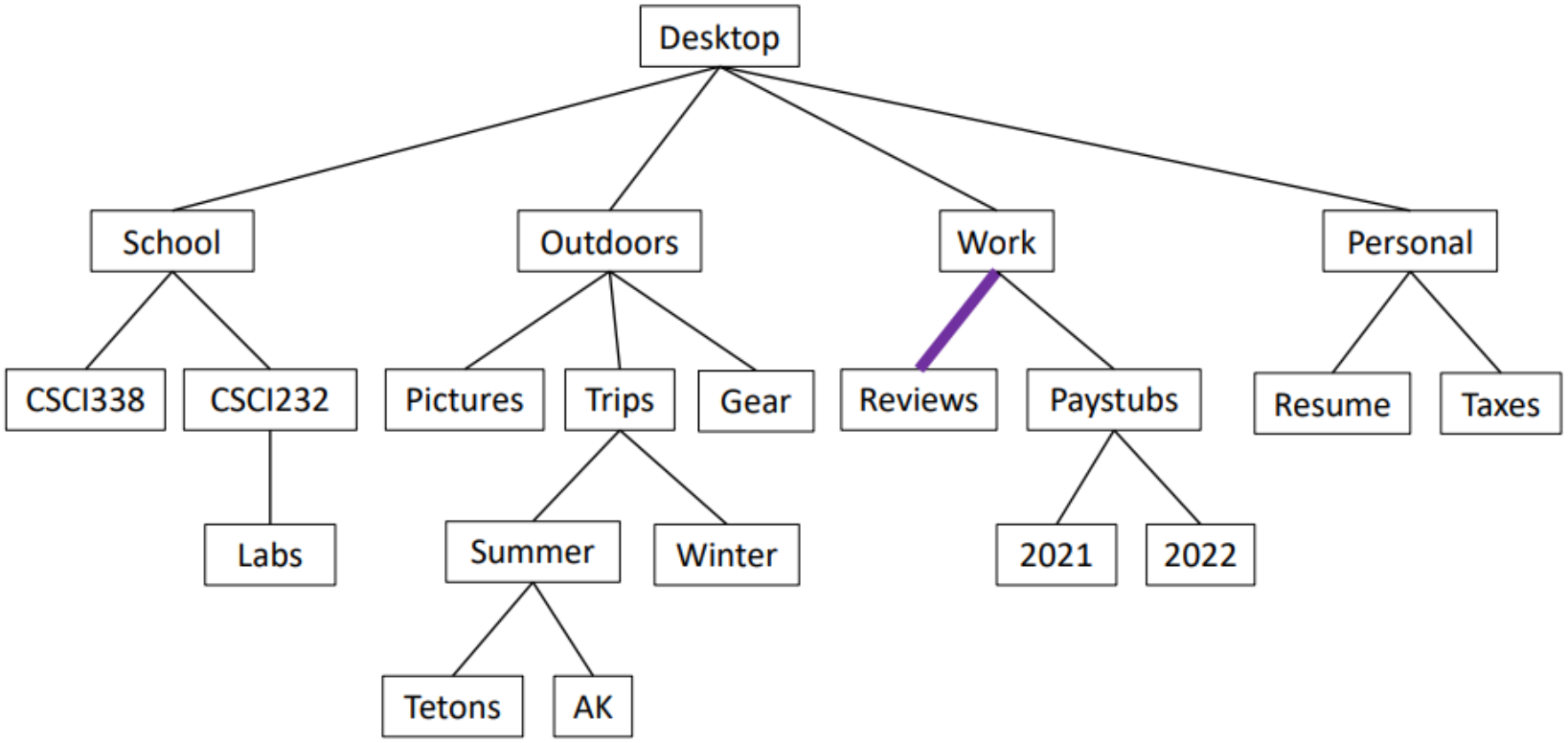


Internal Node: A node with at least one child (i.e., parent nodes).

Leaf Node: A node without children.

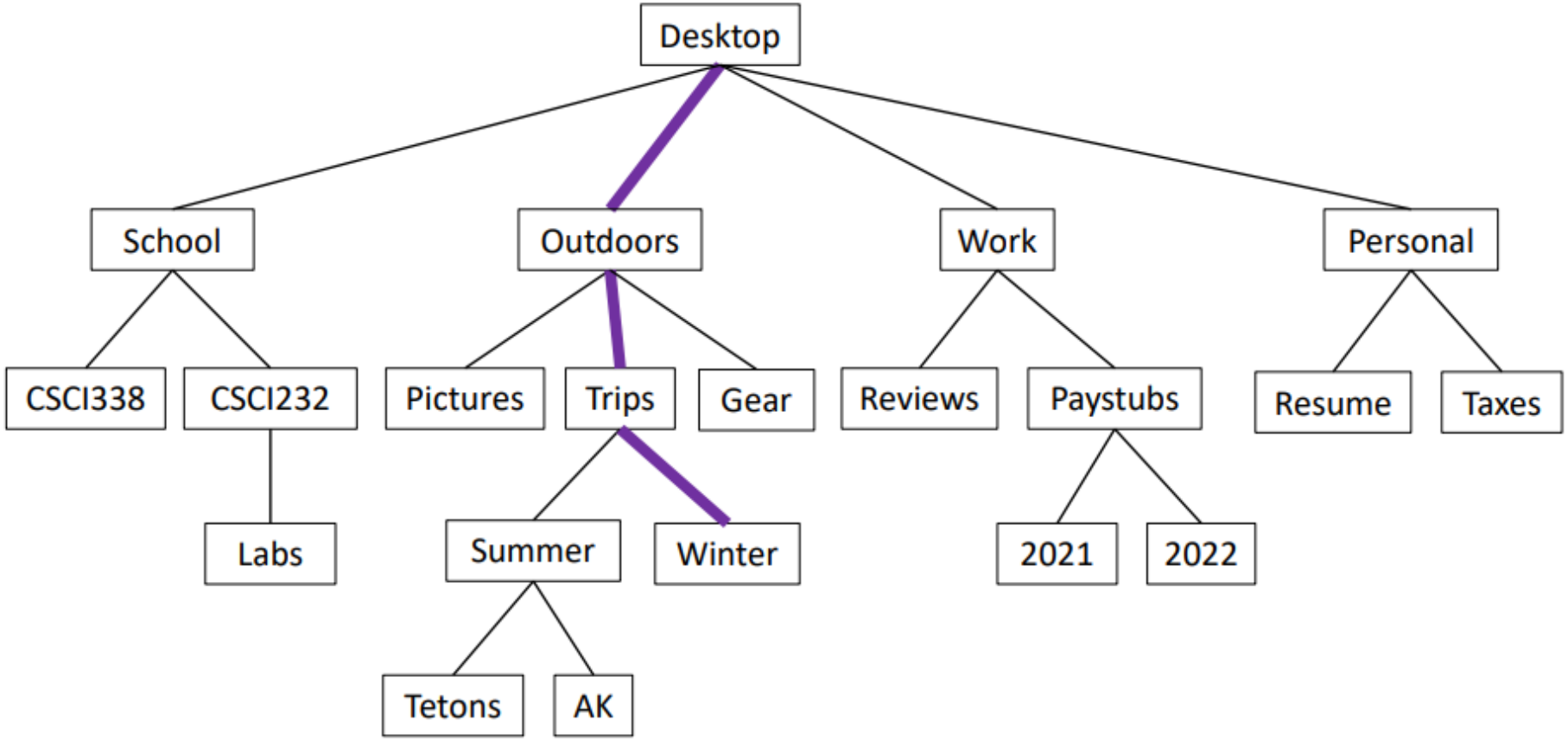
Every node is an internal node or a leaf node.

Trees - Definitions

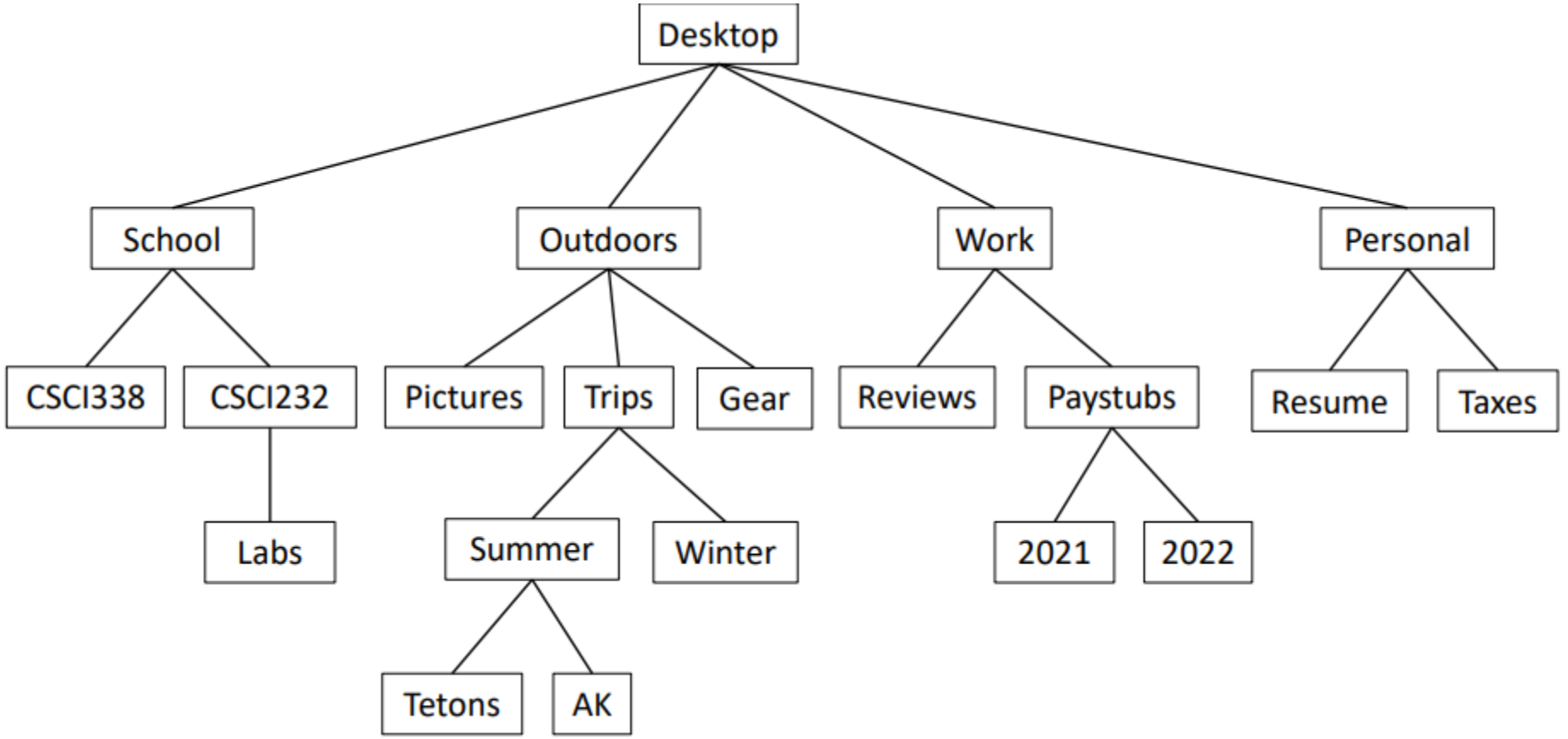


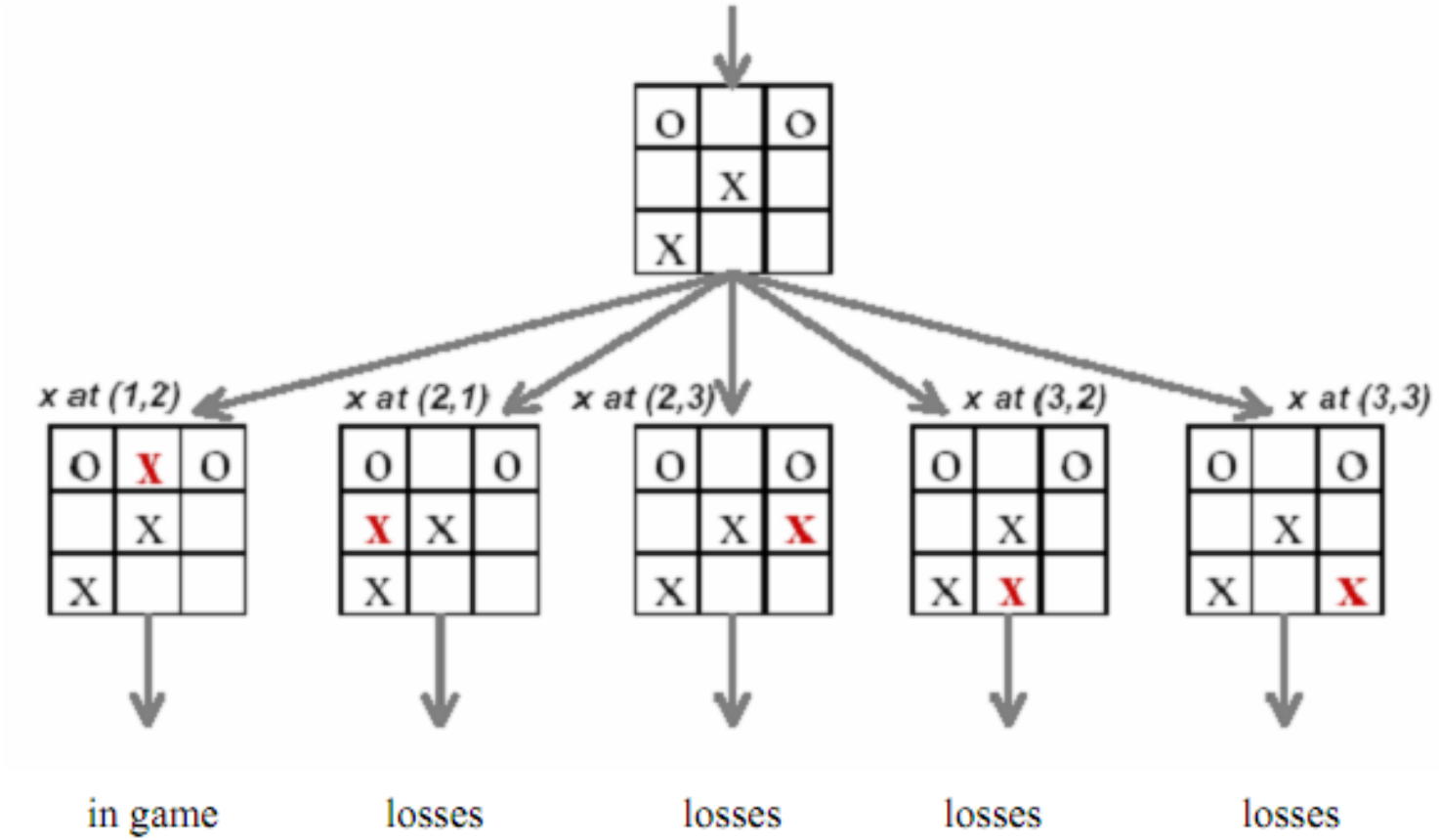
Edge: A pair of nodes such that one is the parent of the other. There is no edge between nodes that are not directly parent-child related.

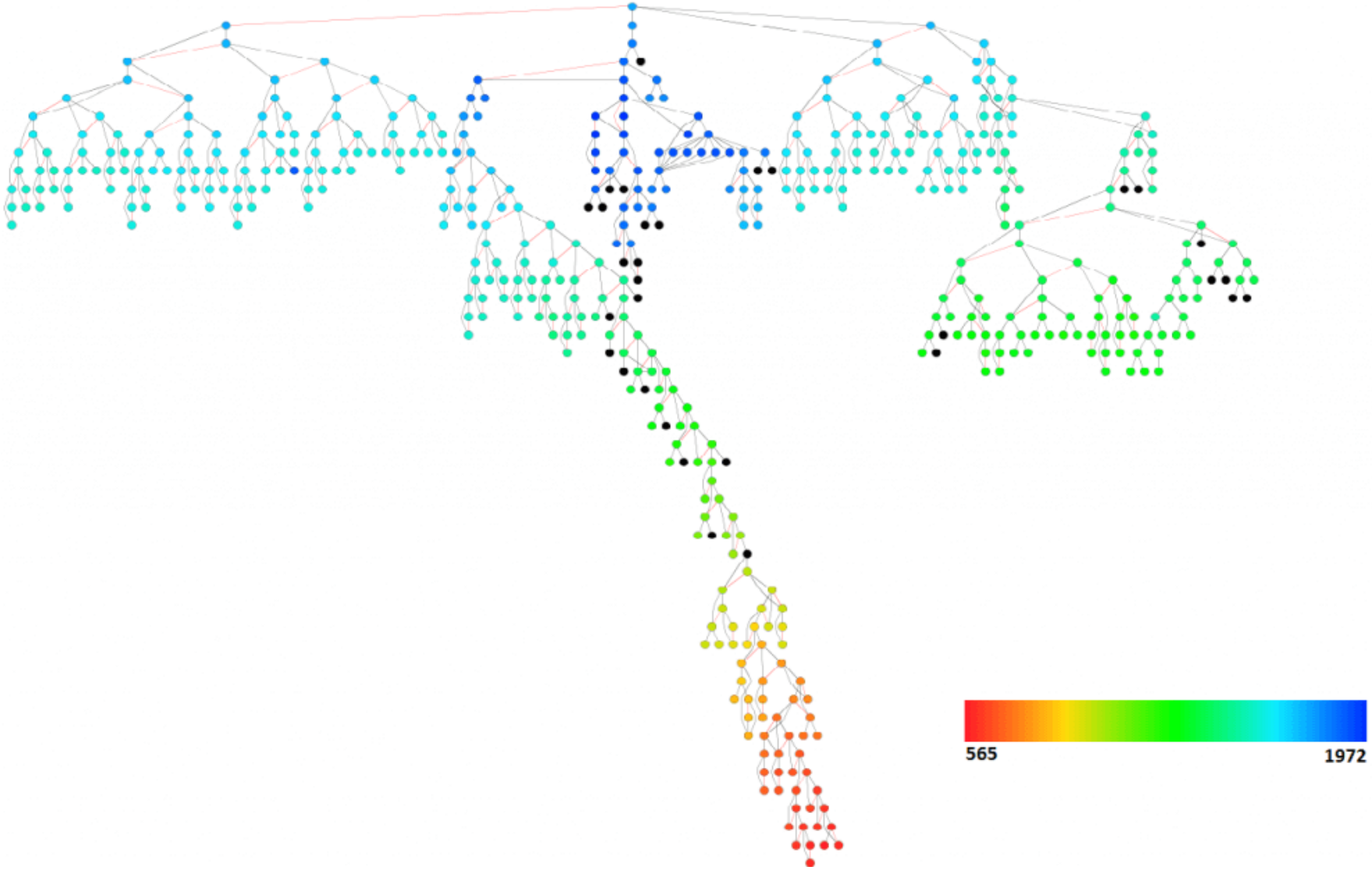
Trees - Definitions

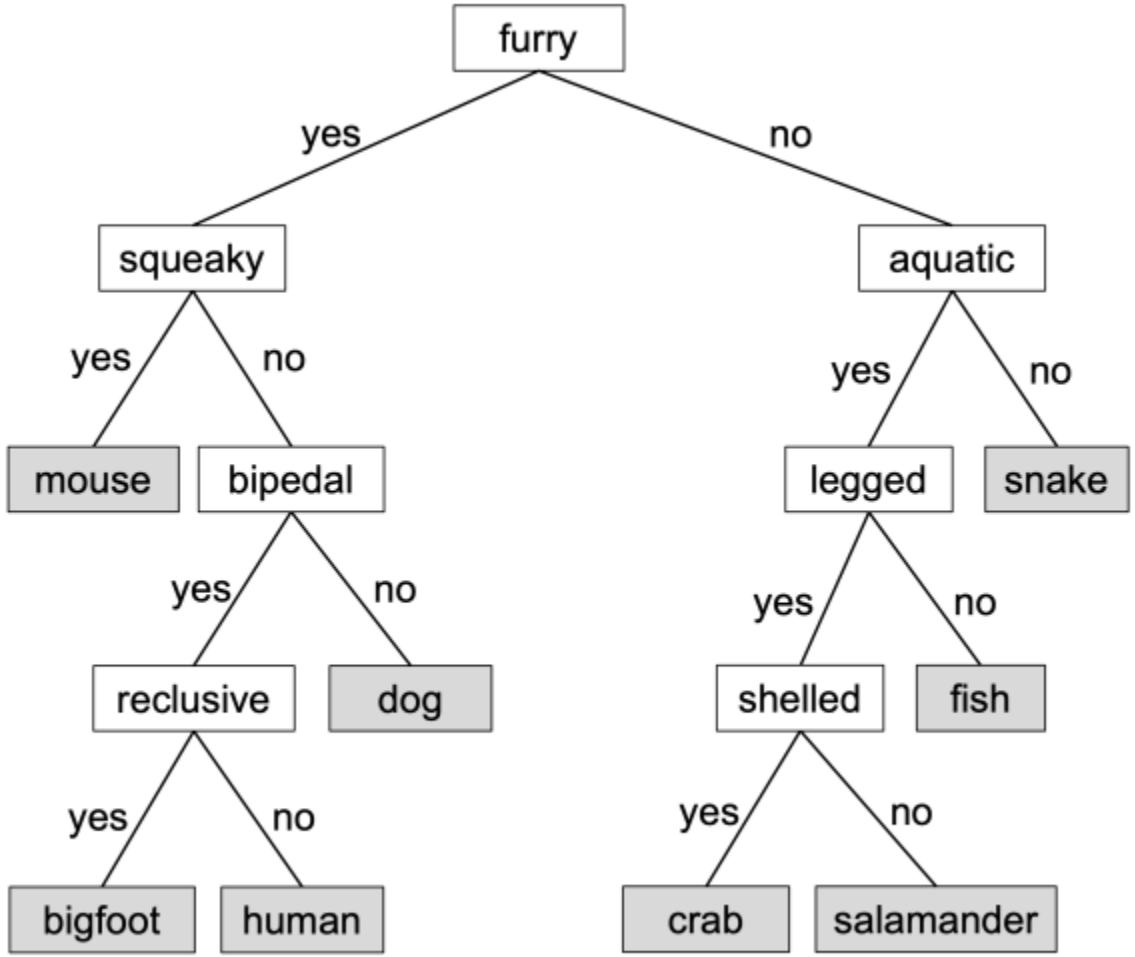


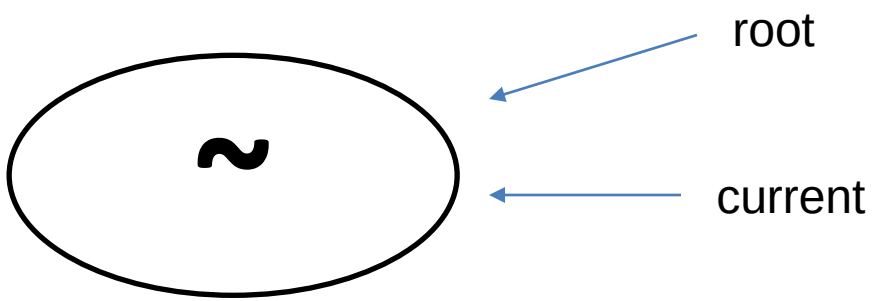
Path: A sequence edge-connected nodes.

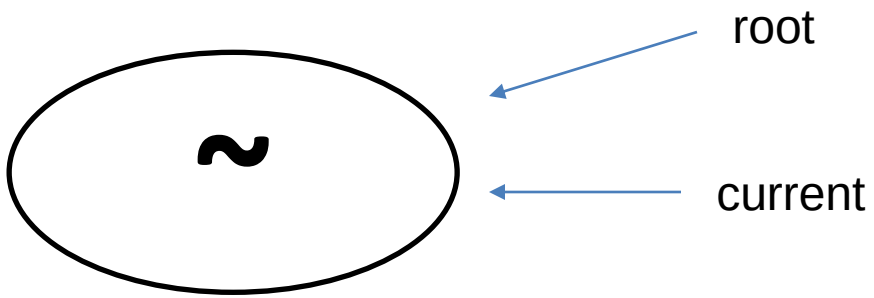

Desktop
School
Outdoors
Work
Personal
CSCI338
CSCI232
Pictures
Trips
Gear
Reviews
Paystubs
Resume
Taxes
Labs
Summer
Winter
2021
2022
Tetons
AK

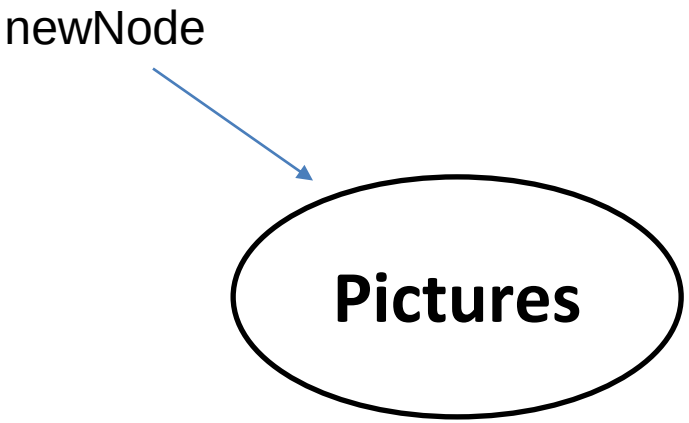

x at (1,2)
x at (2,1)
x at (2,3)
x at (3,2)
x at (3,3)
in game
losses
losses
losses
losses


565
1972


furry
yes
no
squeaky
aquatic
yes
no
yes
no
mouse
bipedal
legged
snake
yes
no
yes
no
reclusive
dog
shelled
fish
yes
no
yes
no
bigfoot
human
crab
salamander

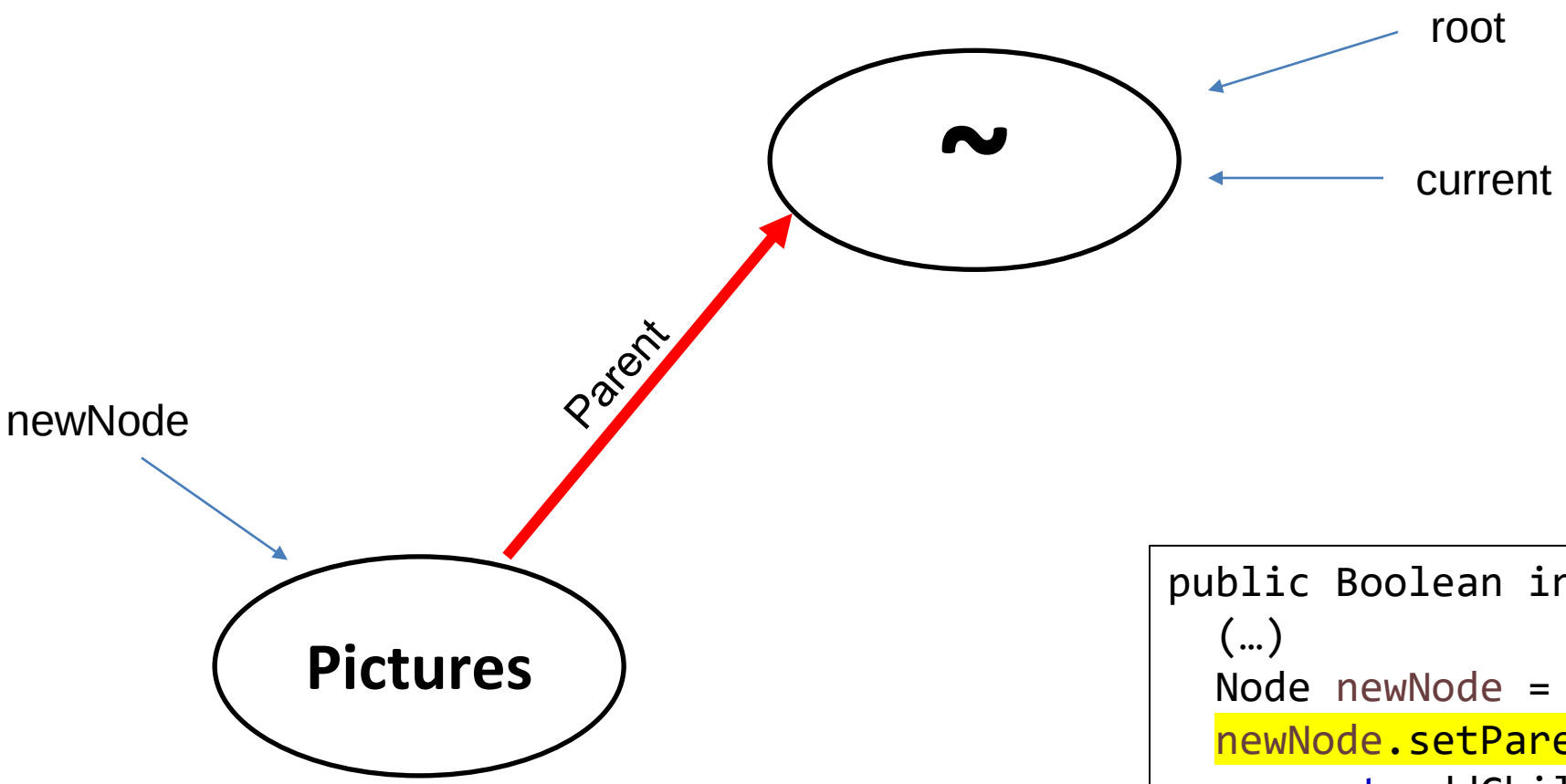

root
~
current





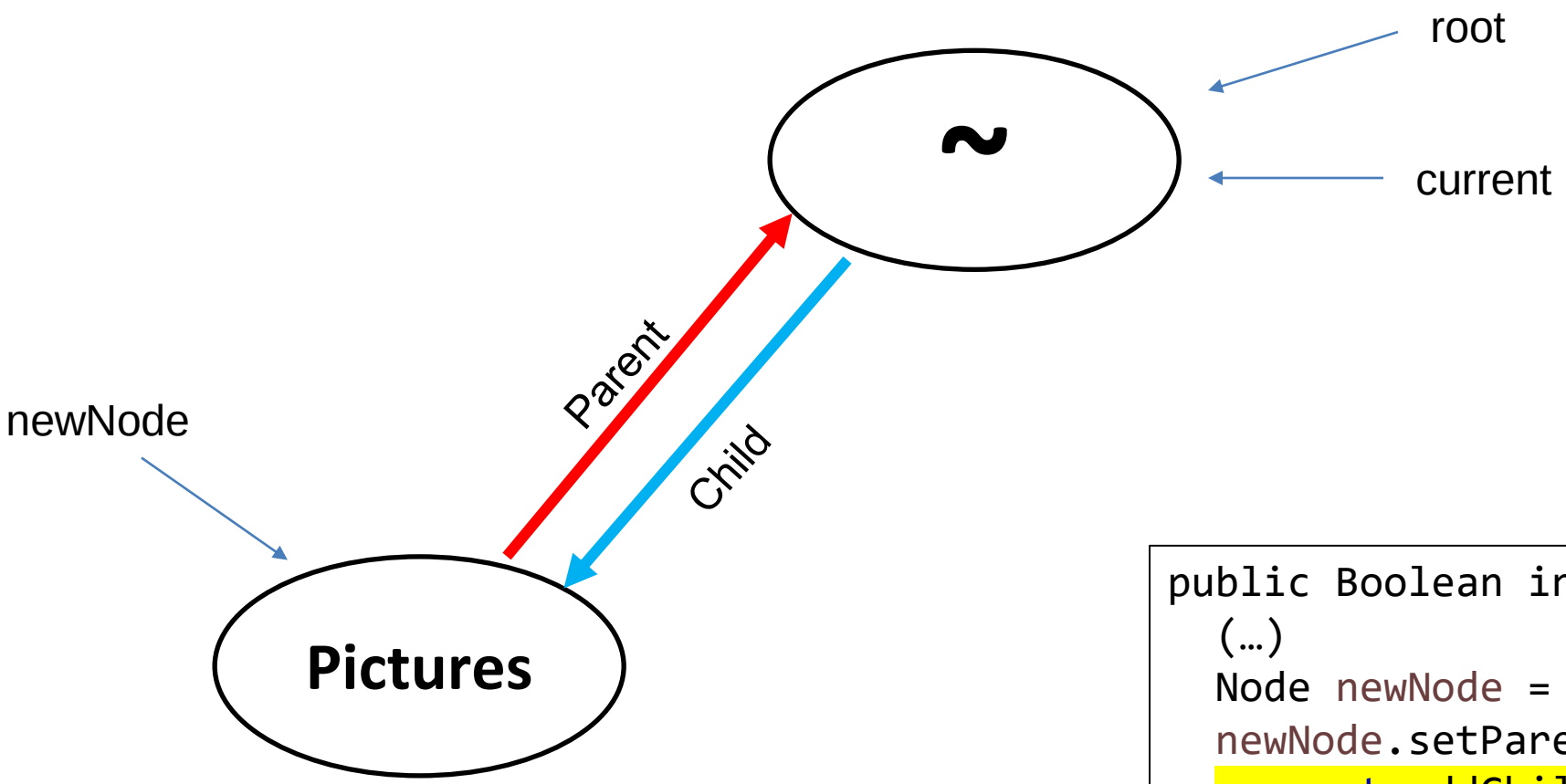
New node that needs to be added?

```
public Boolean insert(String directory){
  (…)
  Node newNode = new Node(directory);
  newNode.setParent(current);
  current.addChild(newNode);
 (…)
}
```



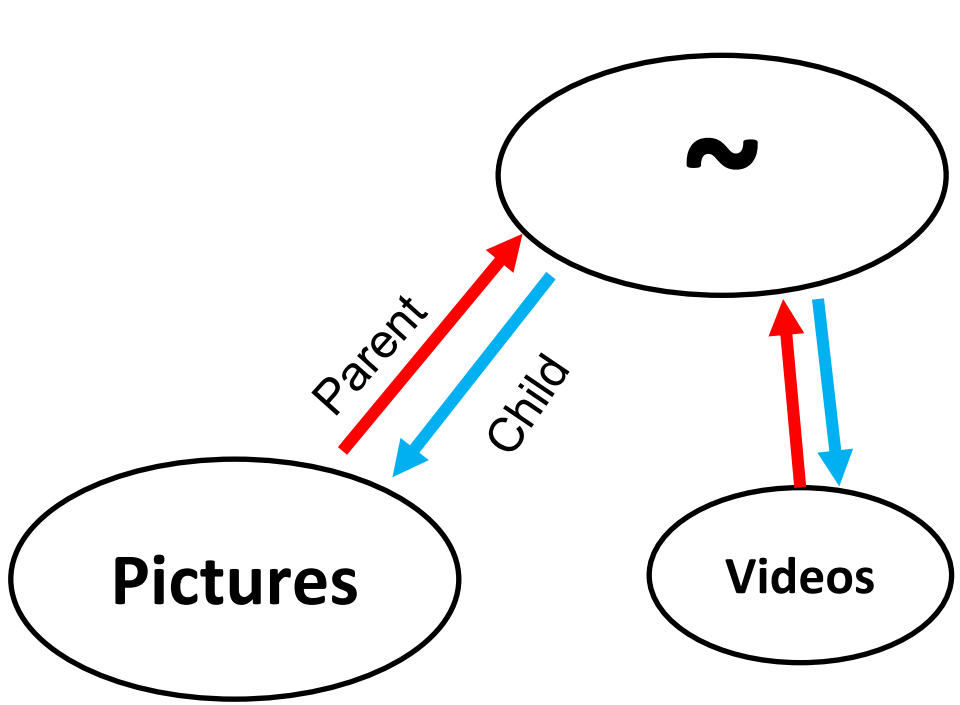
New node that needs to be added?

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

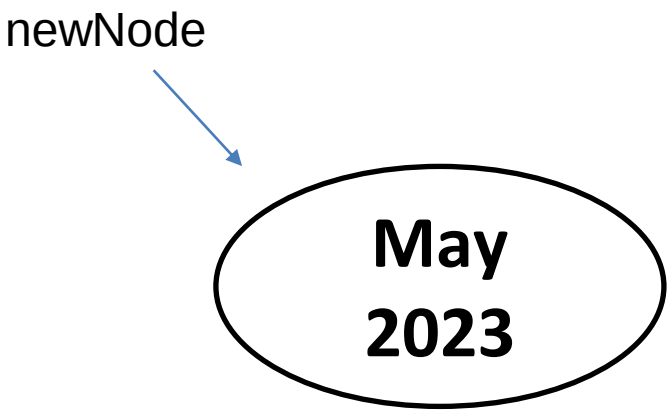



```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

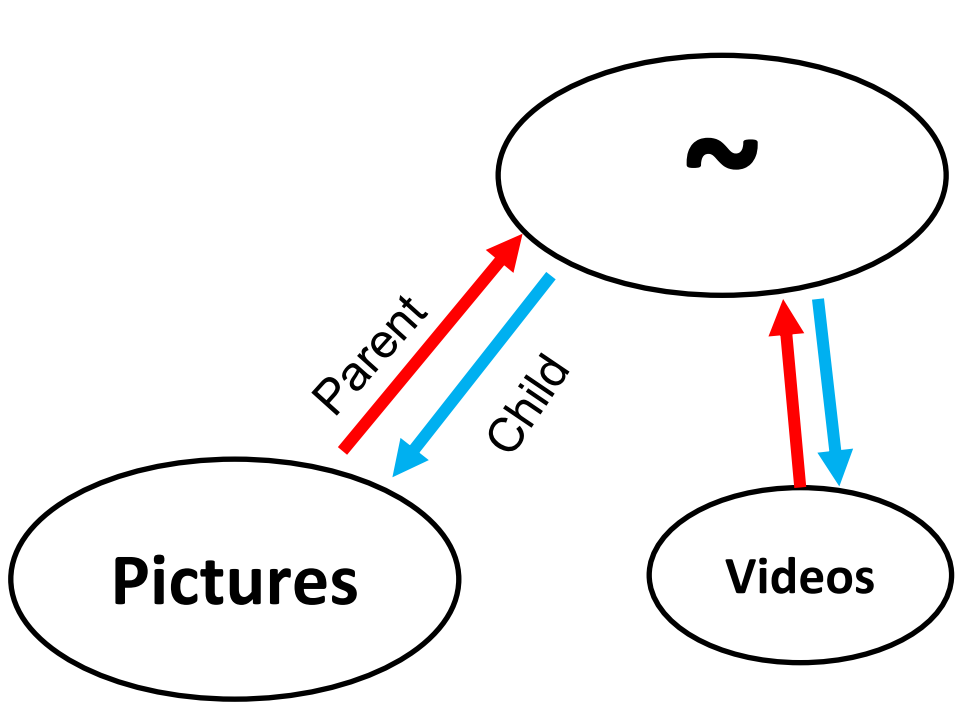
New node that needs to be added?




```
public Boolean insert(String directory){
  (…)
  Node newNode = new Node(directory);
  newNode.setParent(current);
  current.addChild(newNode);
 (…)
}
```

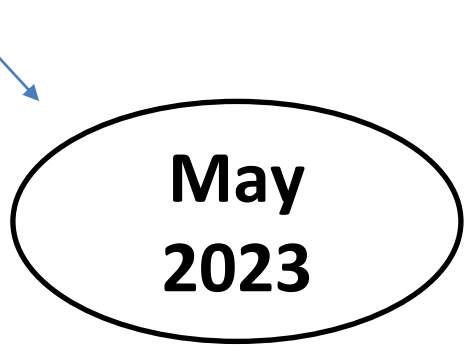


```
tree.moveDown("Pictures");
```



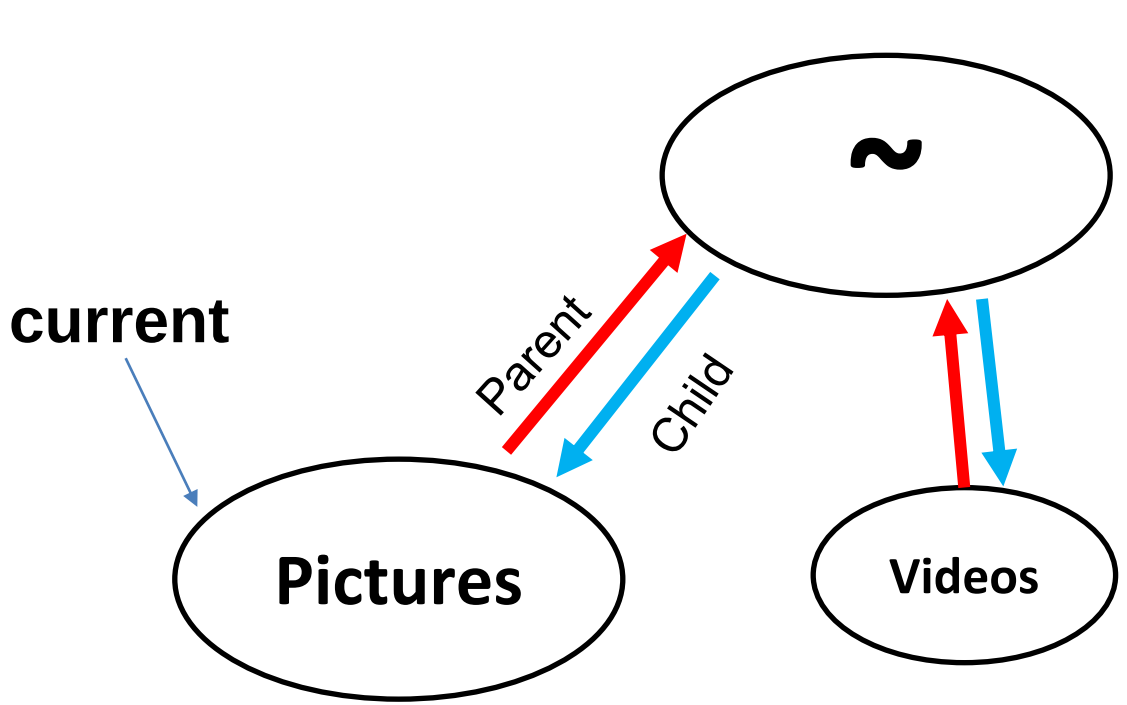

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

newNode



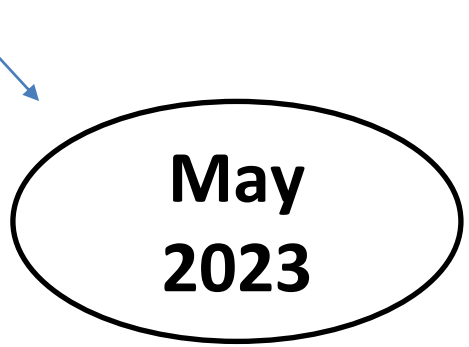

```
public boolean moveDown(String directory) {
   ArrayList<Node> children = current.getChildren();
      for(Node child: children) {
         if(child.getName().equals(directory)) {
            current = child;
            return true;
         }
   }
    return false;
}
```

```
tree.moveDown("Pictures");
```



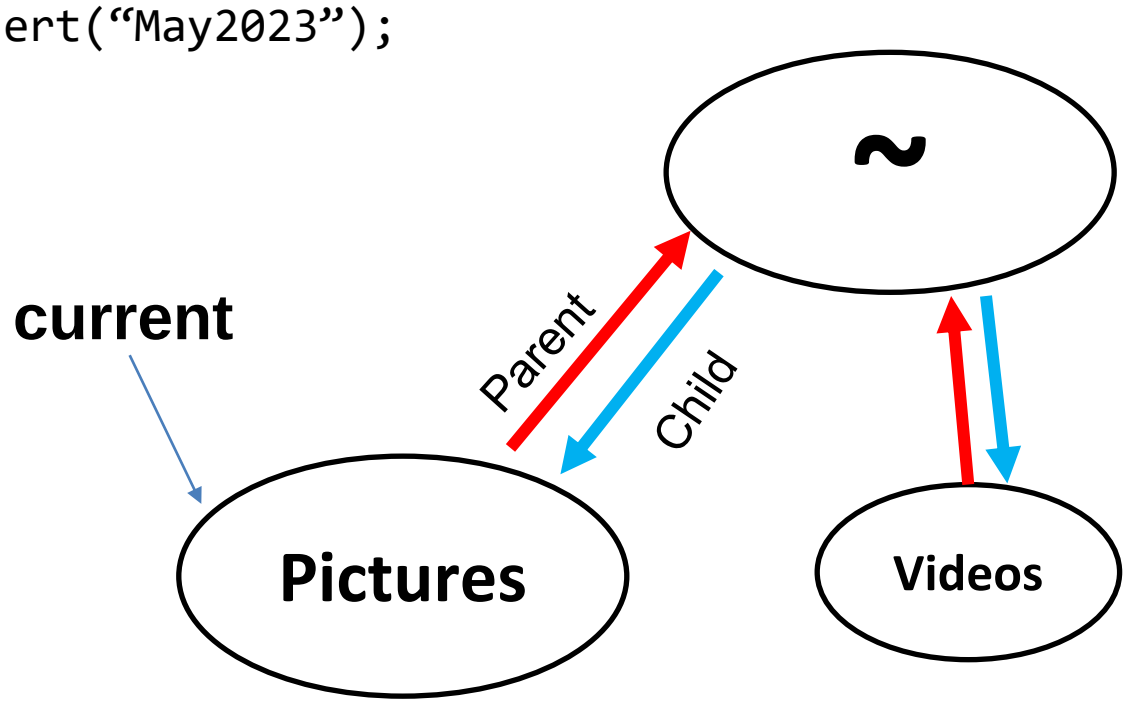

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

newNode




```
public boolean moveDown(String directory) {
   ArrayList<Node> children = current.getChildren();
      for(Node child: children) {
         if(child.getName().equals(directory)) {
            current = child;
            return true;
         }
   }
    return false;
}
```

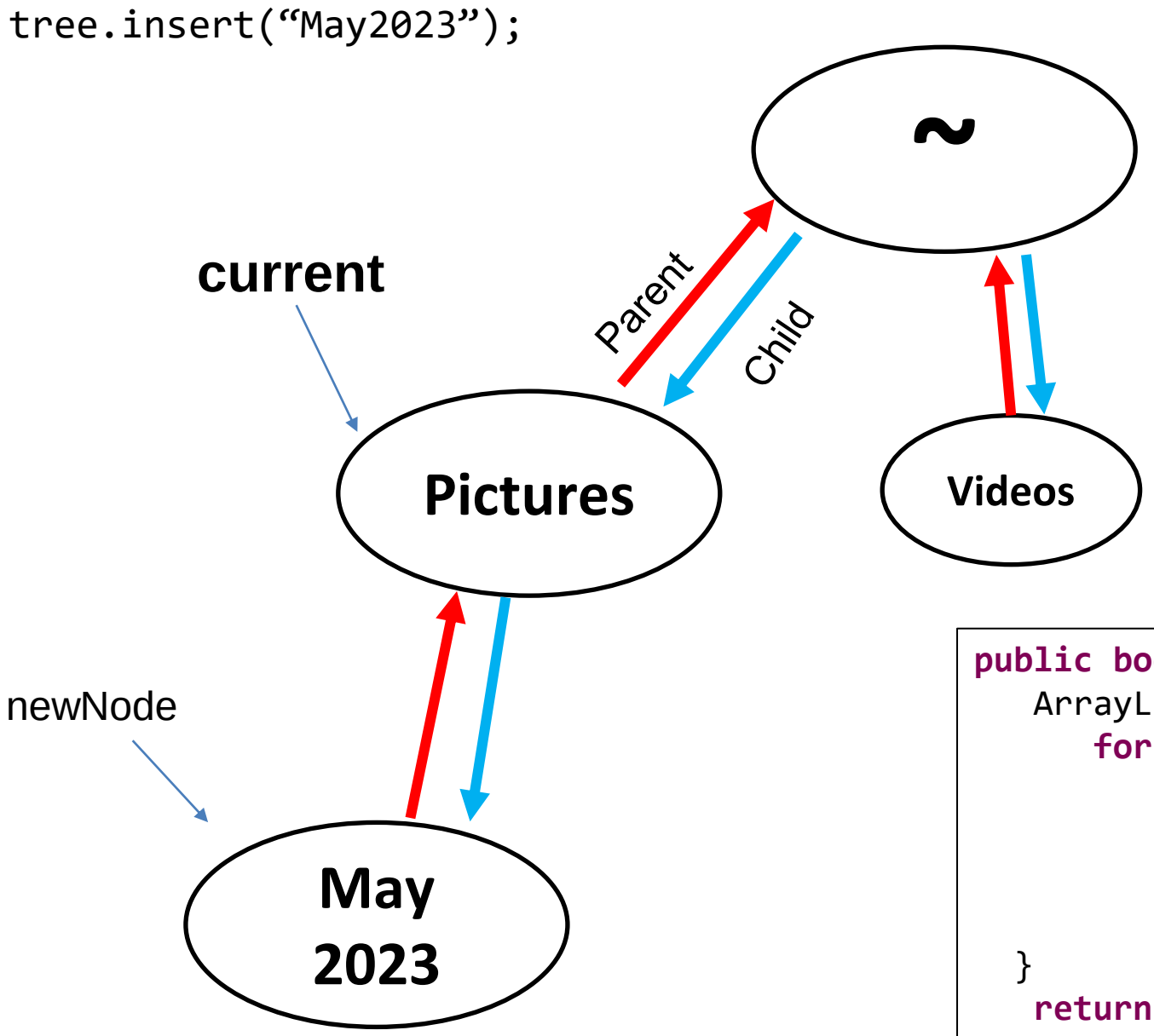
```
tree.moveDown("Pictures");
tree.insert("May2023");
```




```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

```
public boolean moveDown(String directory) {
   ArrayList<Node> children = current.getChildren();
      for(Node child: children) {
         if(child.getName().equals(directory)) {
            current = child;
            return true;
         }
   }
    return false;
}
```

```
tree.moveDown("Pictures");
tree.insert("May2023");
```




```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
   (…)
}
```

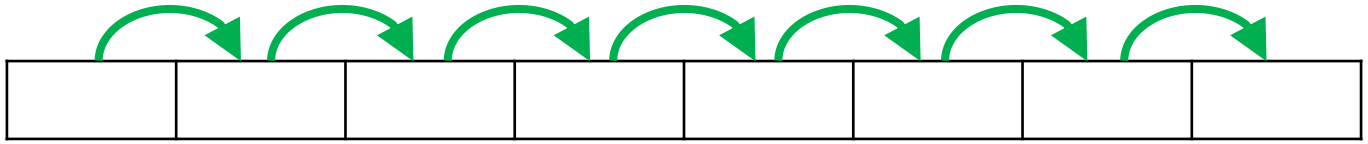
```
public boolean moveDown(String directory) {
   ArrayList<Node> children = current.getChildren();
      for(Node child: children) {
         if(child.getName().equals(directory)) {
            current = child;
            return true;
         }
   }
    return false;
}
```

Tree Traversal
CSCI 232

Tree Traversal

How could you search for a value in an array?

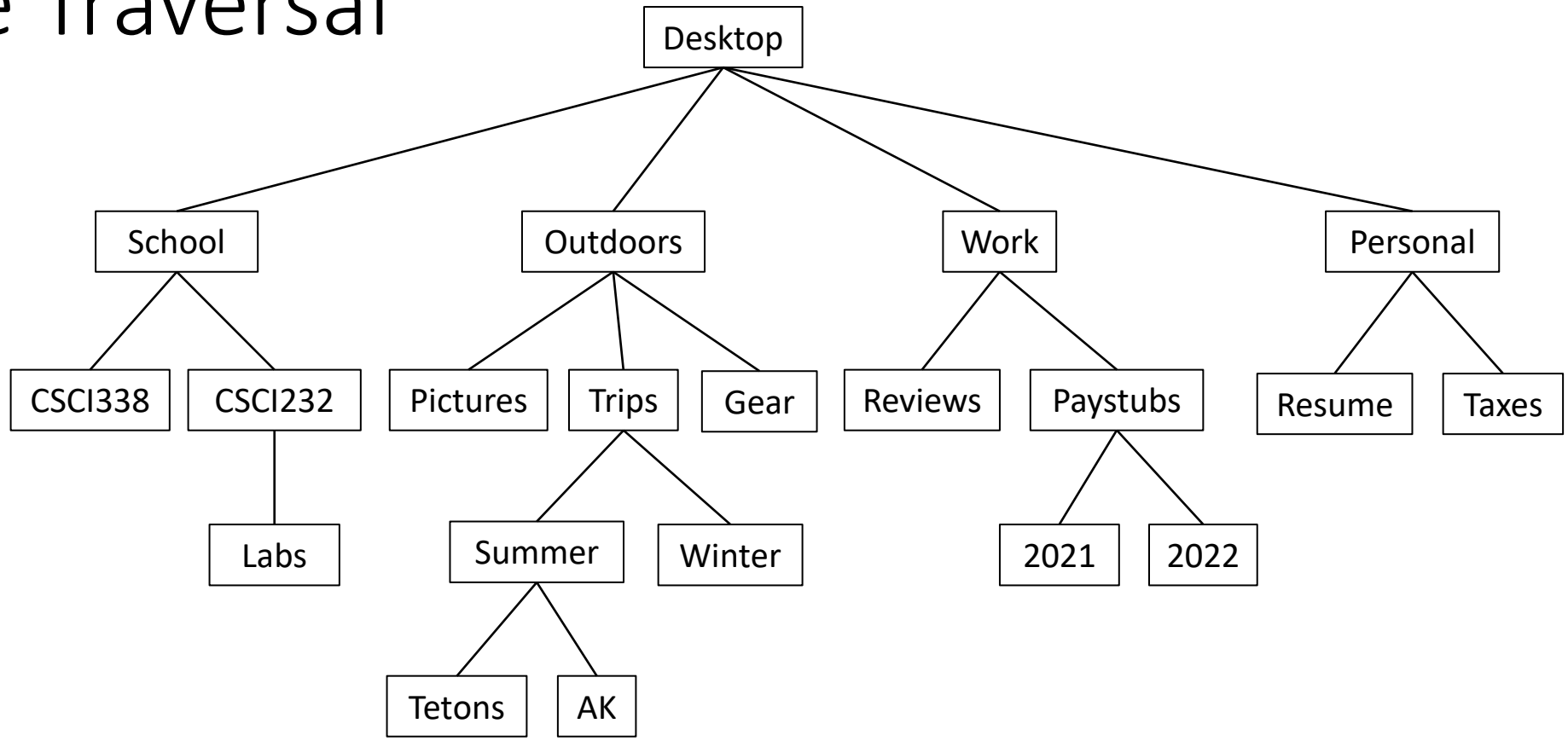
Tree Traversal



How could you search for a value in an array?

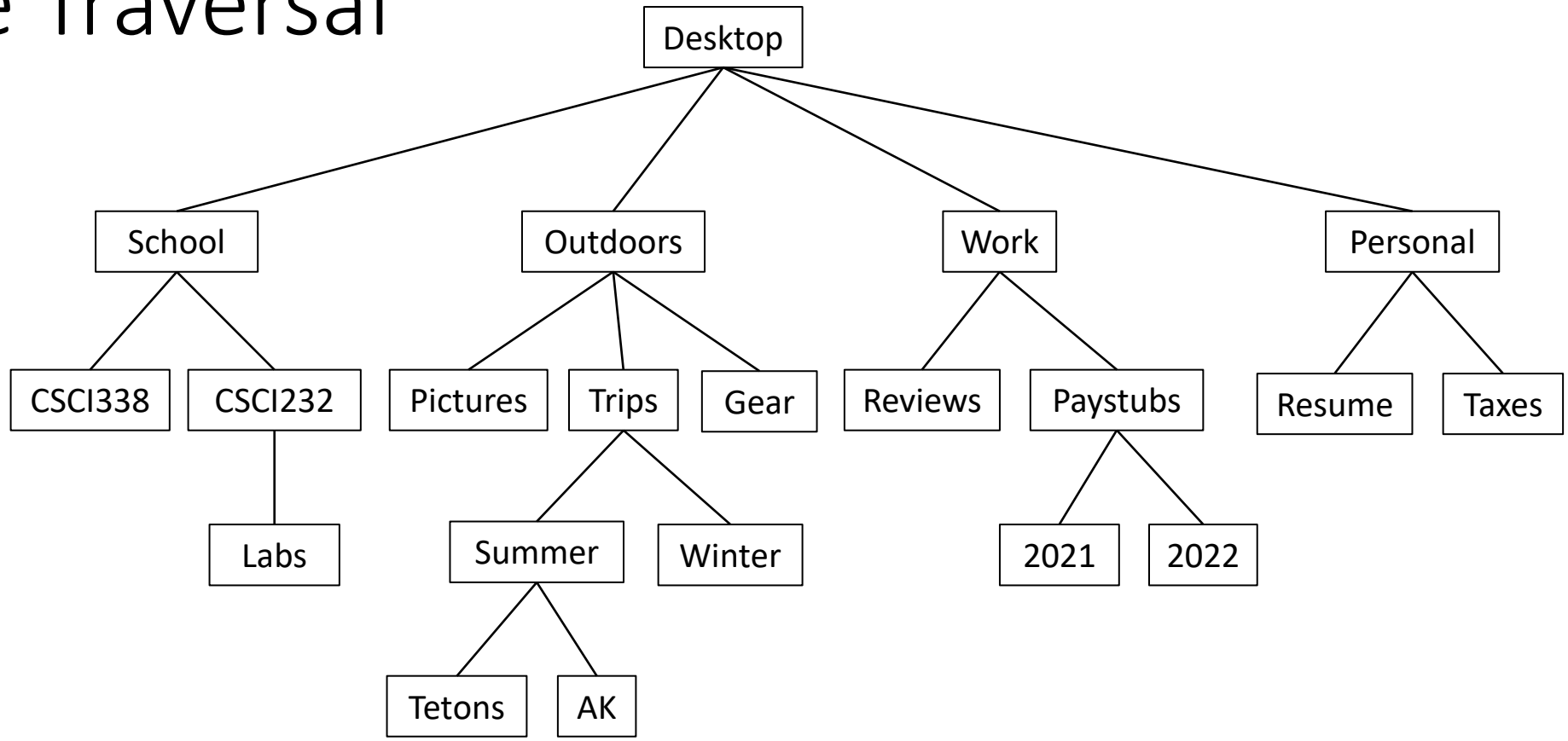
Go one element at a time and see if it is what you are looking for.

Tree Traversal



How could you search for a value in a tree?

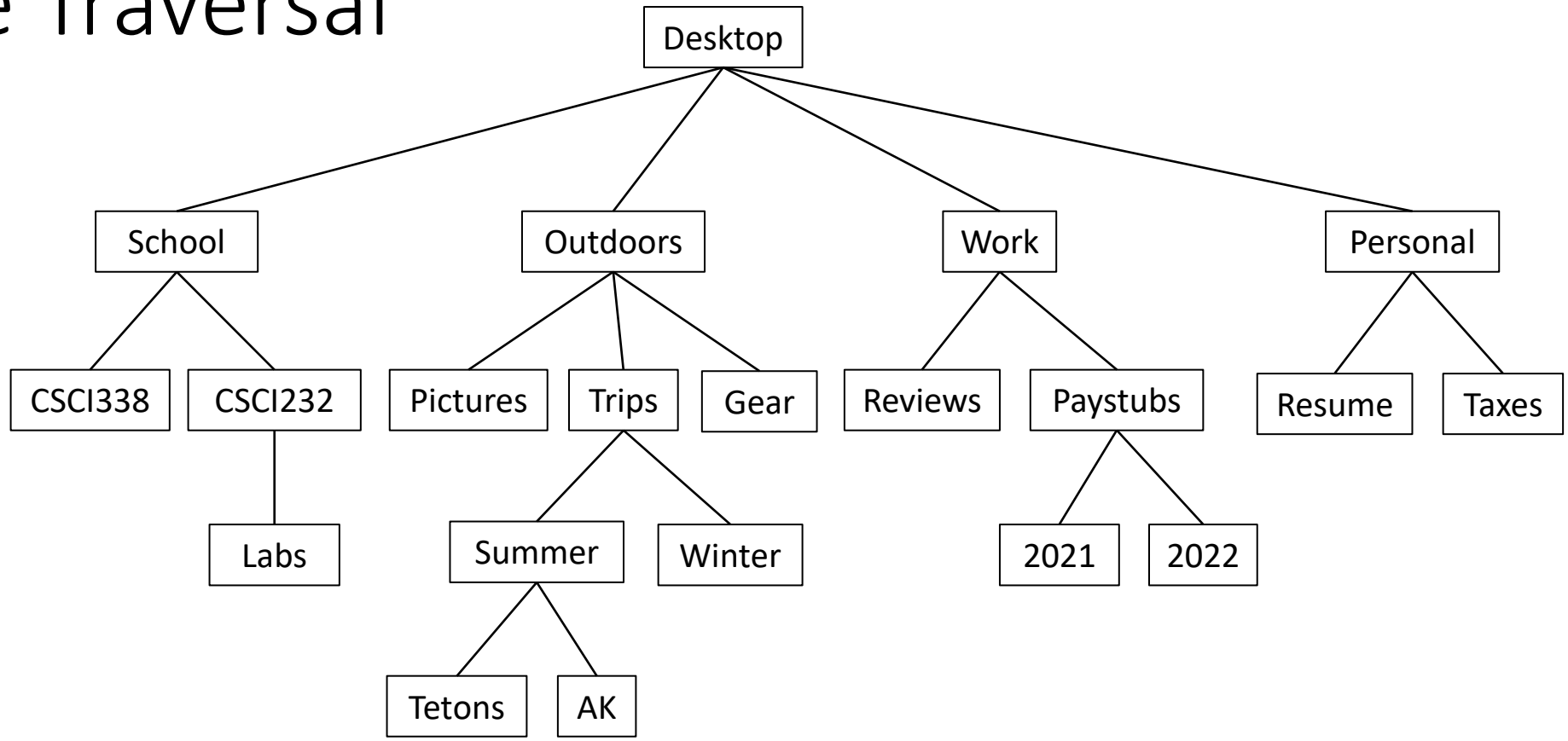
Tree Traversal



How could you search for a value in a tree?

1. Breadth-first. Visit all nodes at the same depth before progressing to next depth

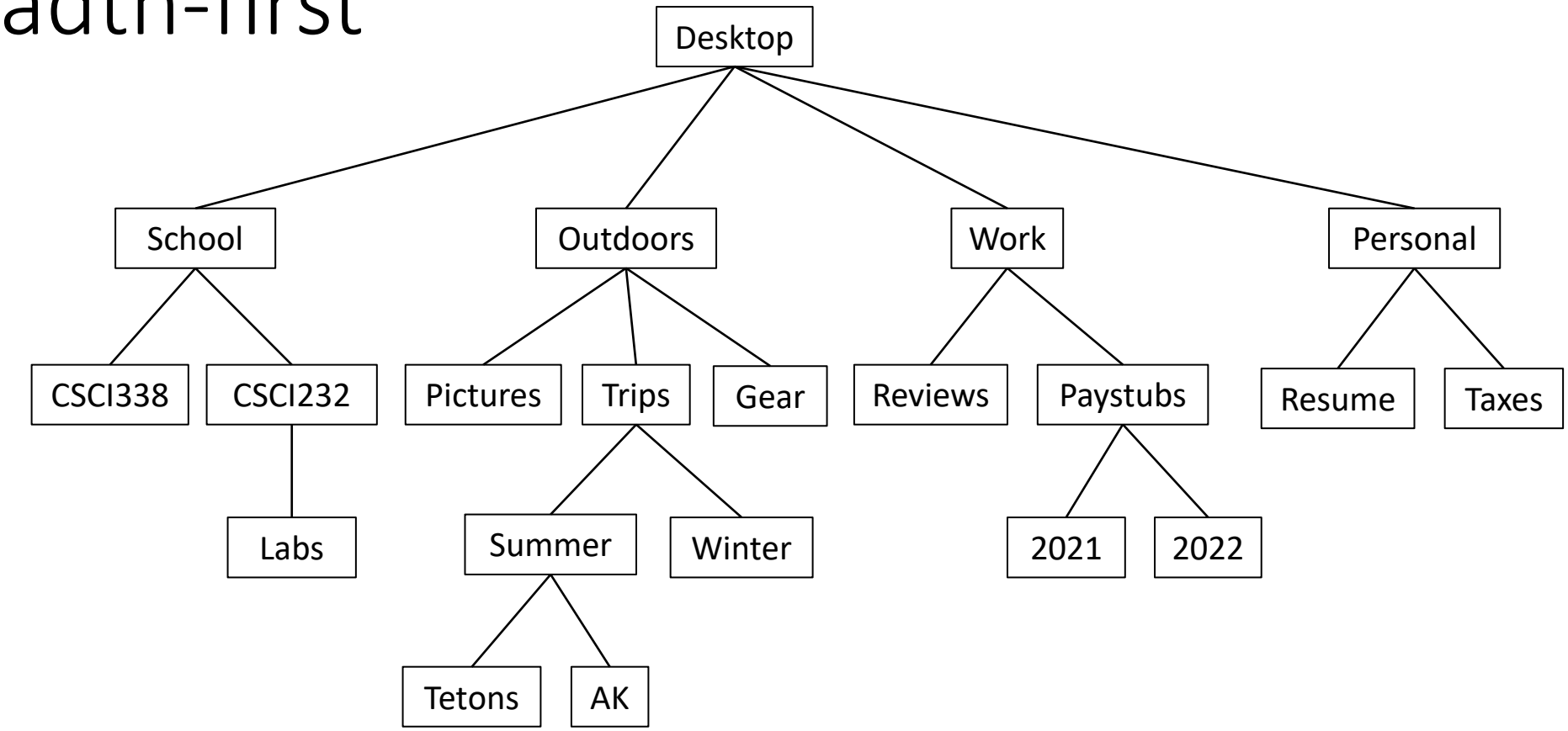
Tree Traversal



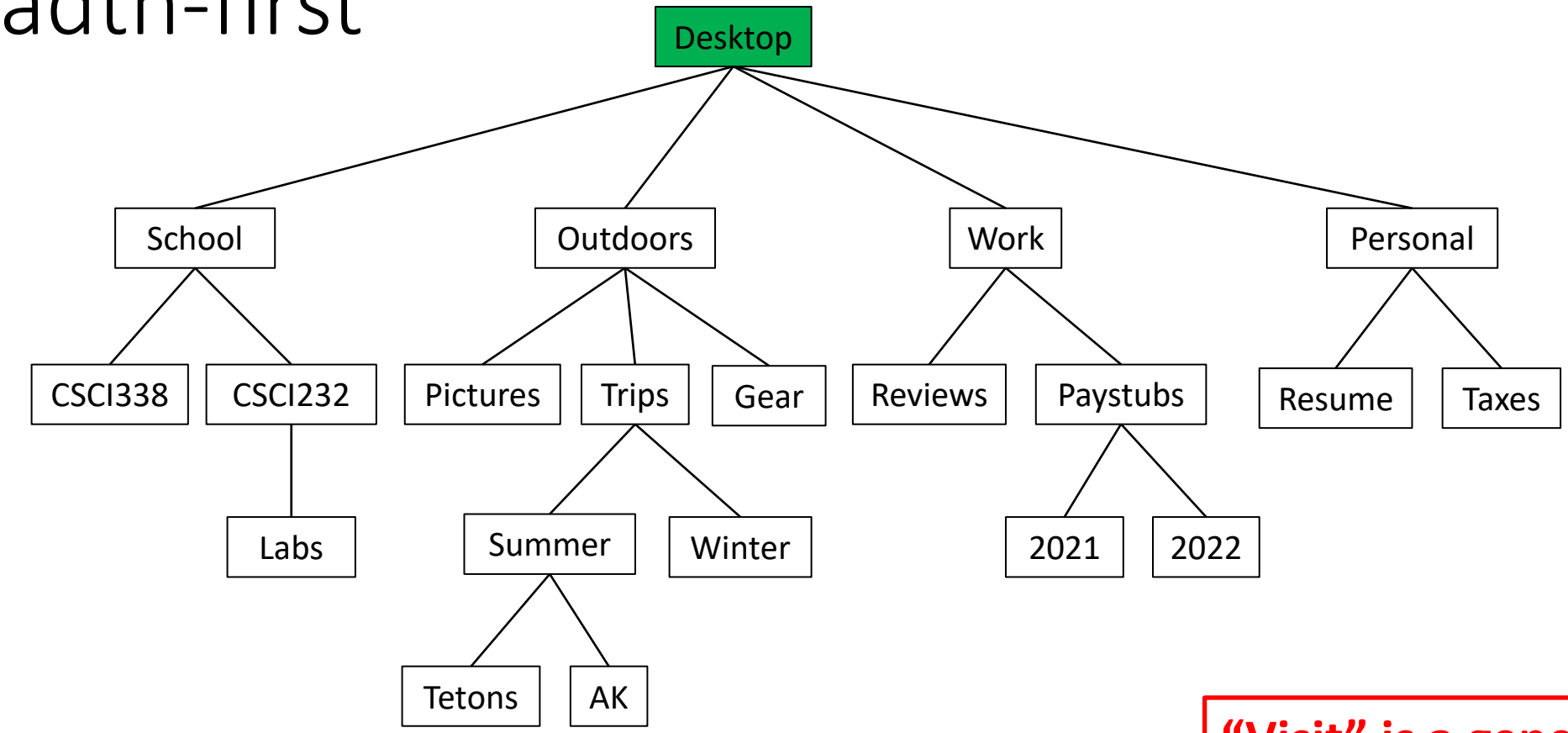
How could you search for a value in a tree?

1. Breadth-first. Visit all nodes at the same depth before progressing to next depth
2. Depth-first. Explores full root-leaf paths.

Breadth-first



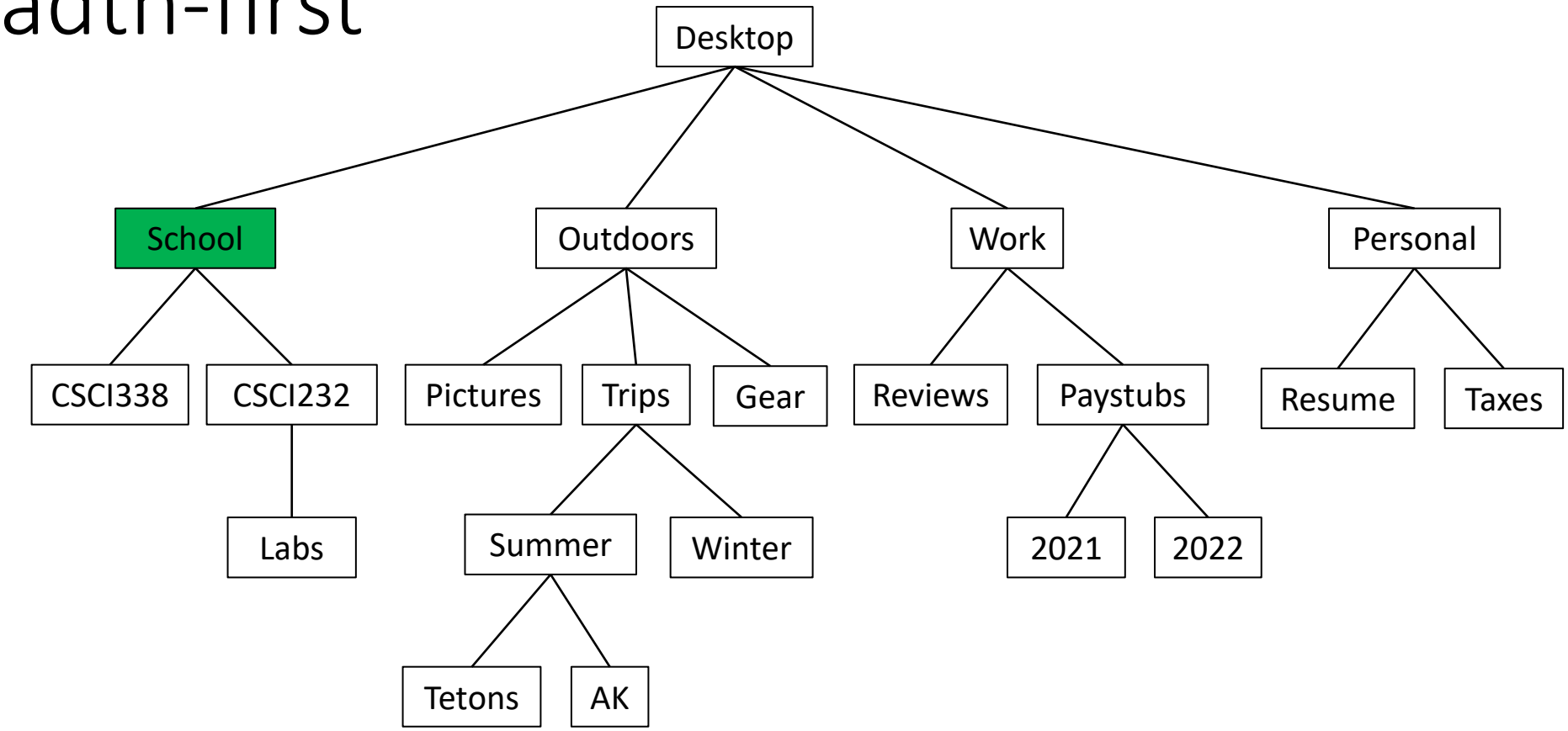
Breadth-first



1. Visit the root.

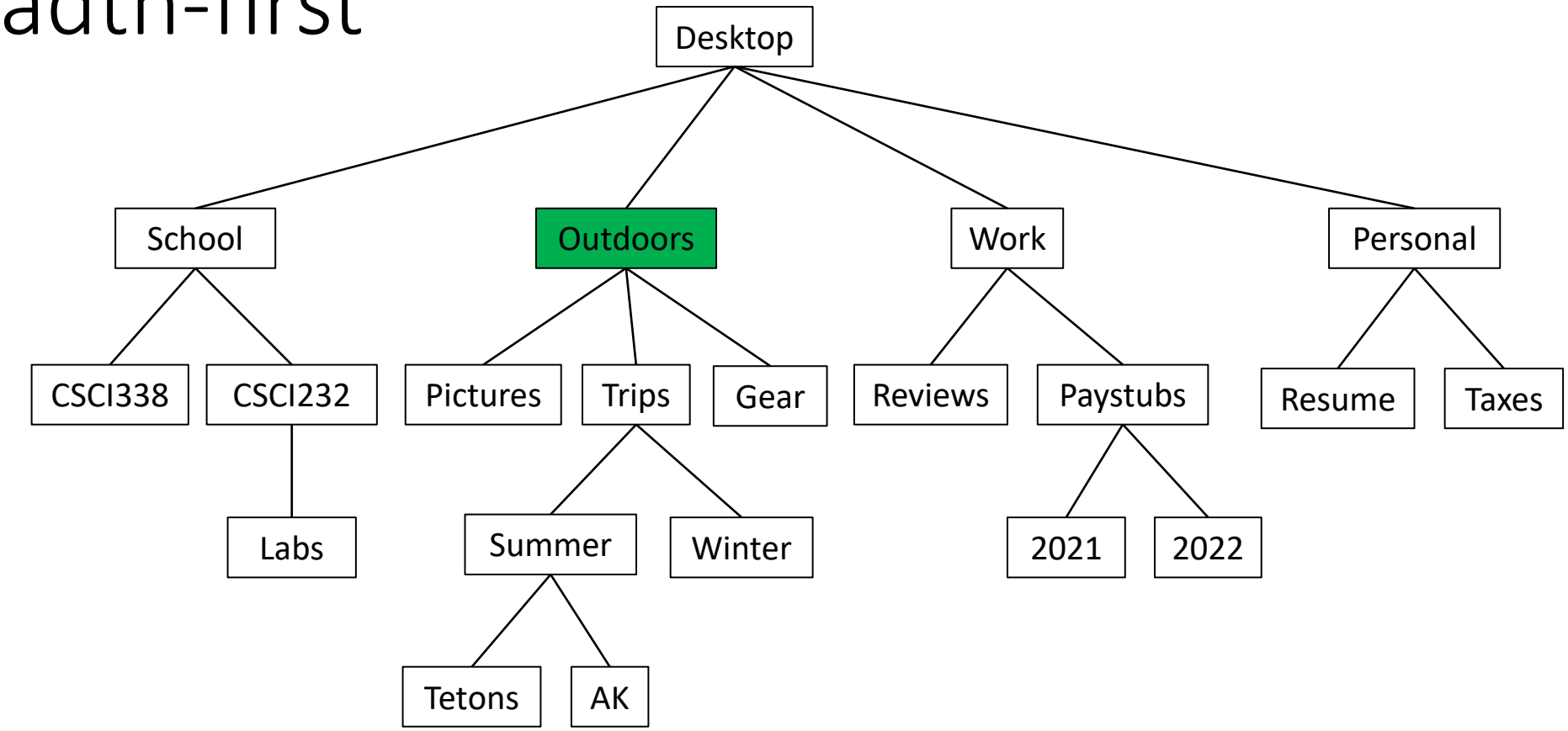
"Visit" is a generic action. The actual action depends on what the application is (ex: print node, compare to value…)

Breadth-first



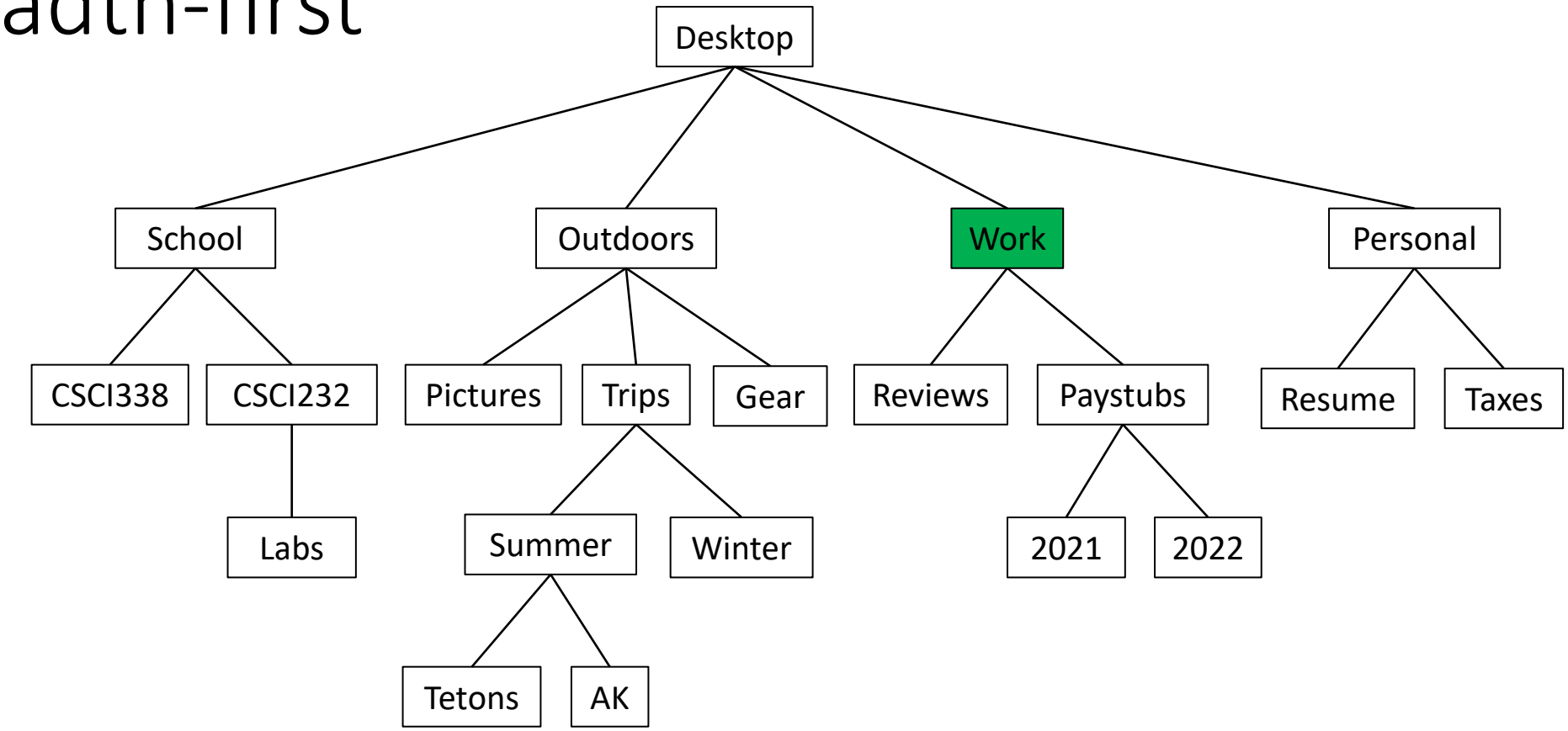
1. Visit the root.
2. Visit all depth 1 nodes (in order).

Breadth-first



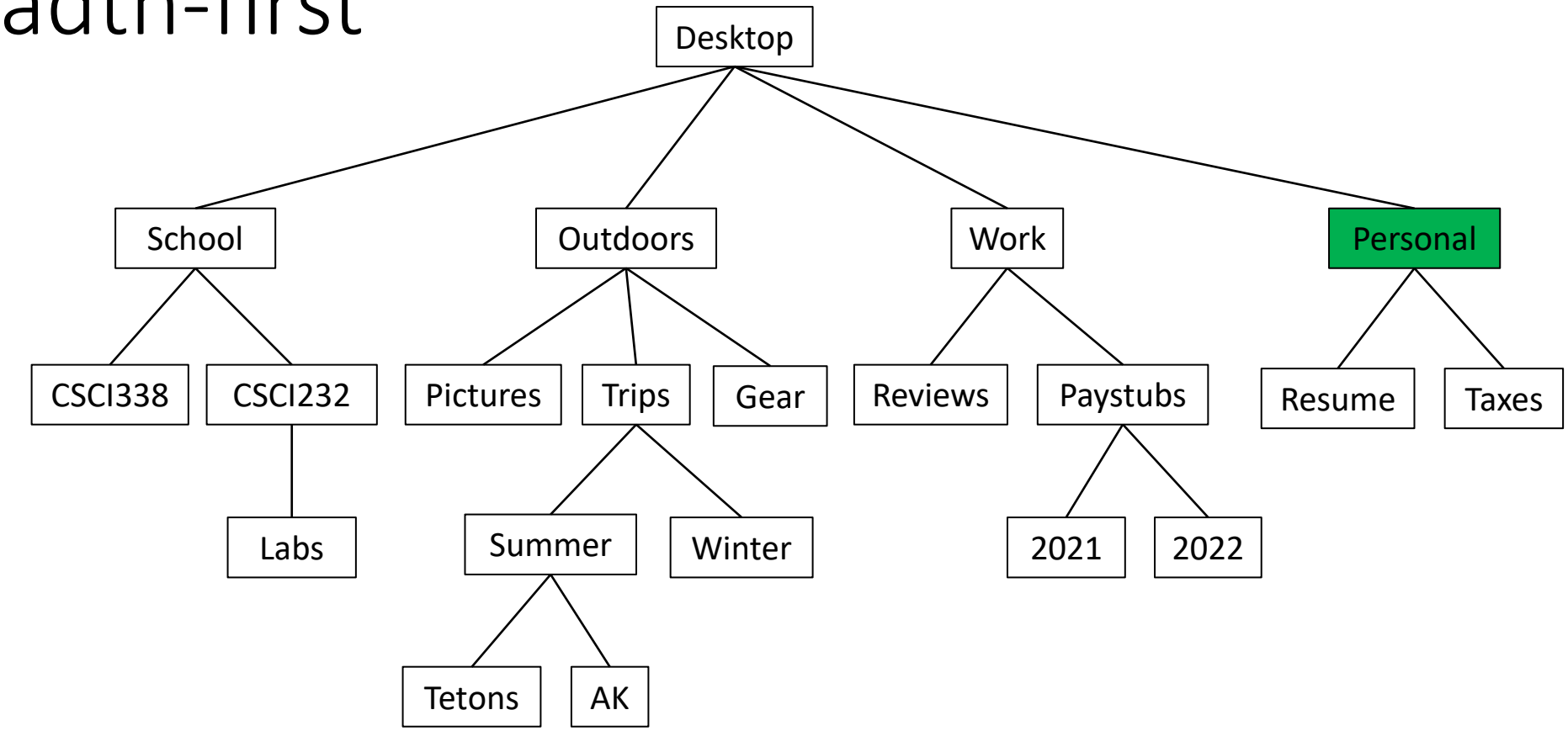
1. Visit the root.
2. Visit all depth 1 nodes (in order).

Breadth-first



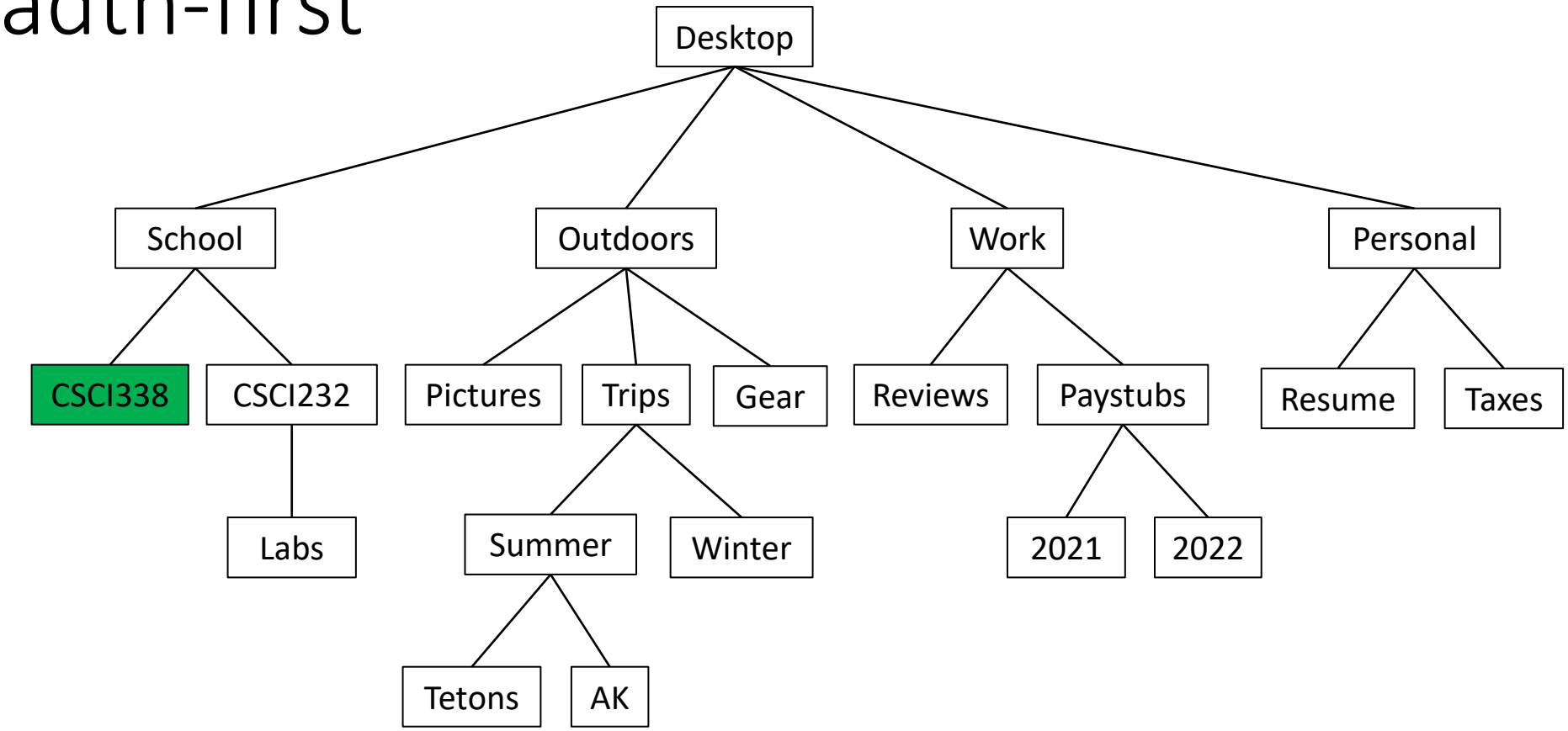
1. Visit the root.
2. Visit all depth 1 nodes (in order).

Breadth-first



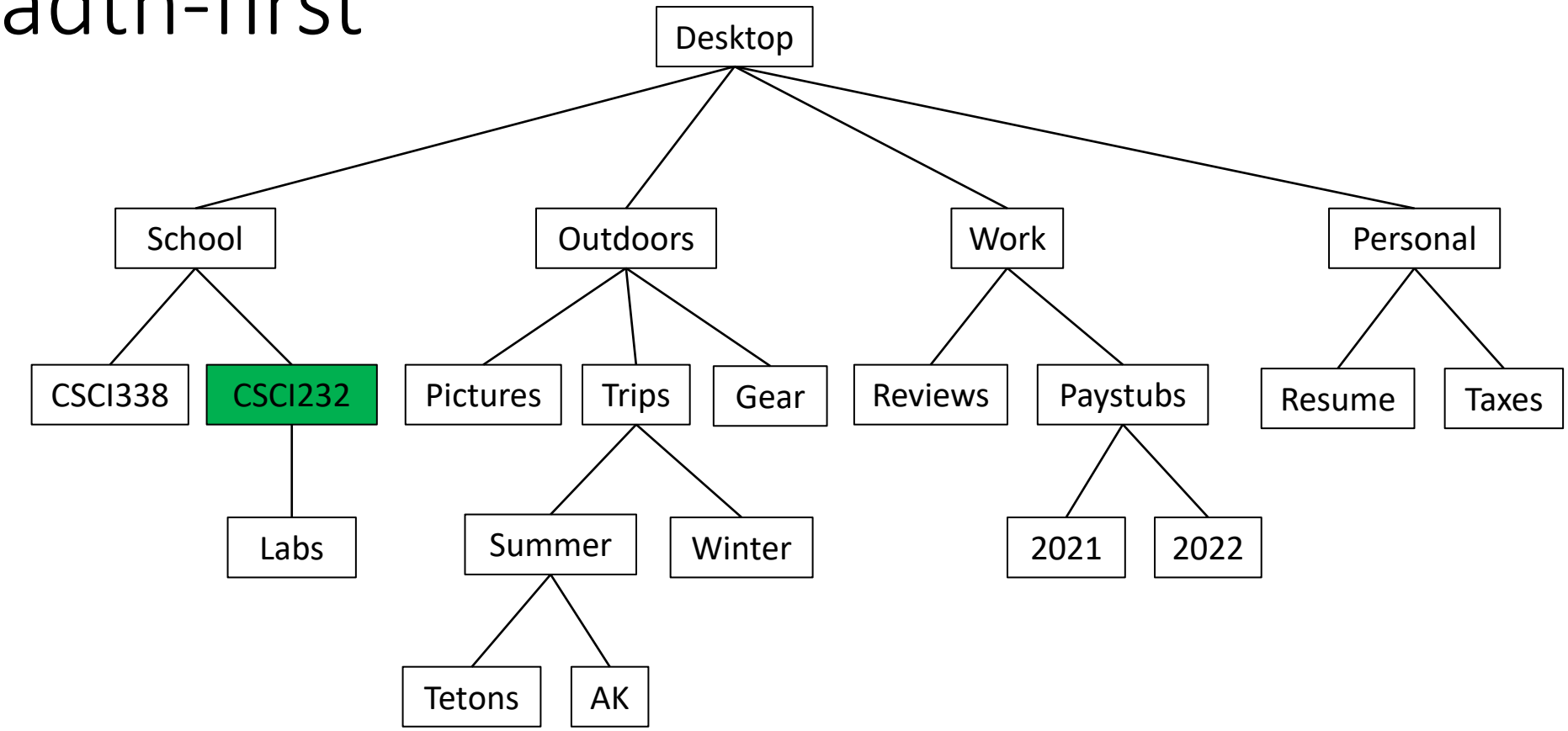
1. Visit the root.
2. Visit all depth 1 nodes (in order).

Breadth-first



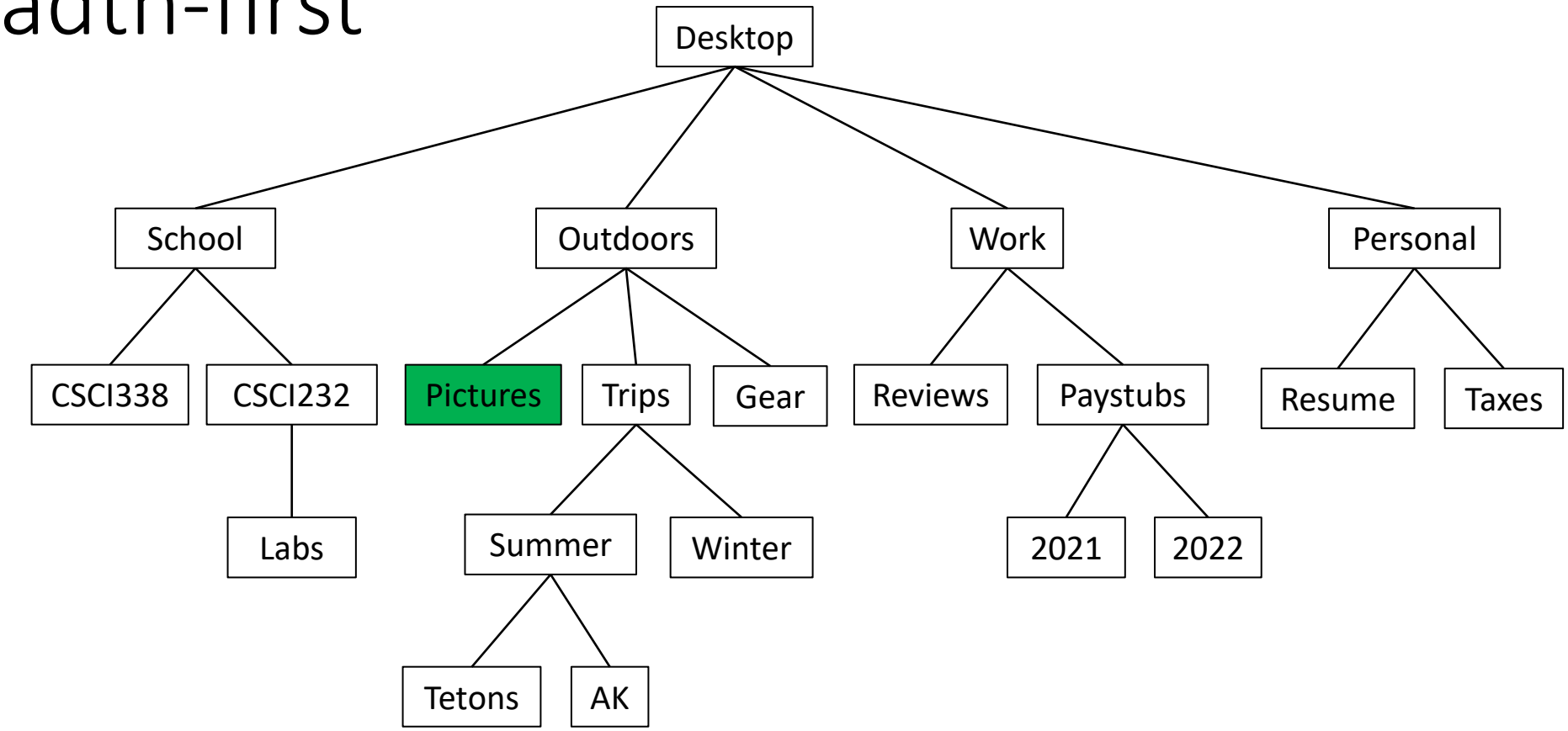
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



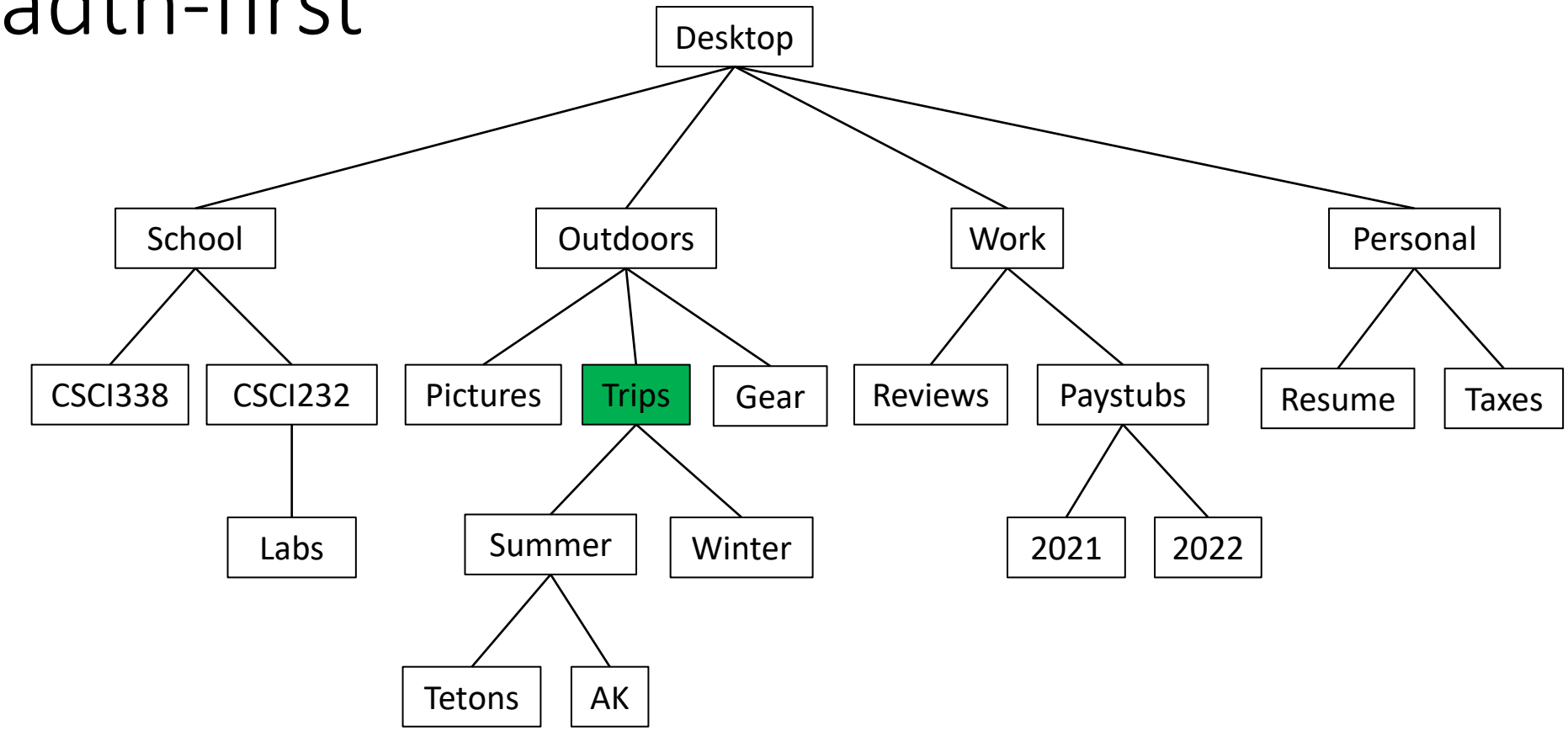
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



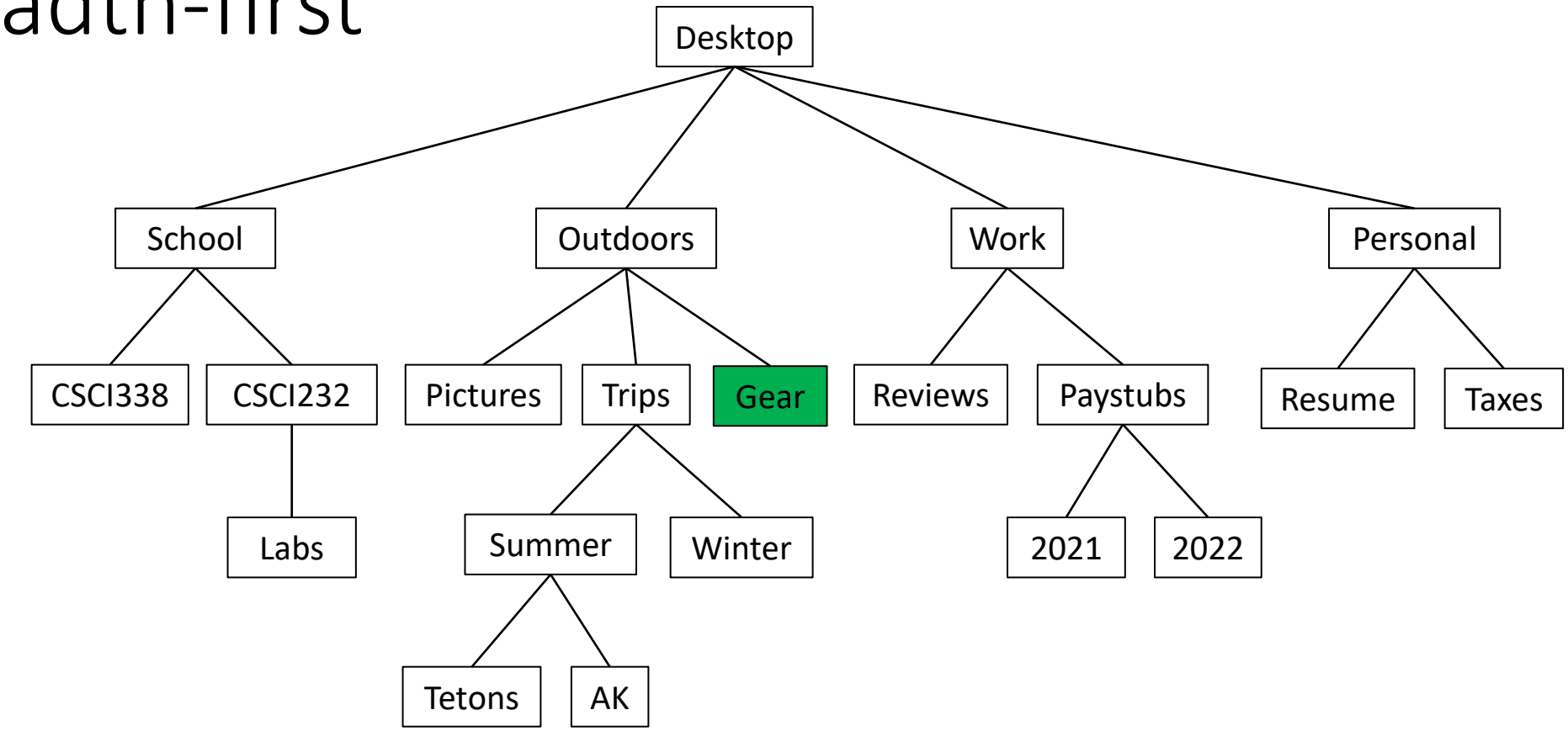
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



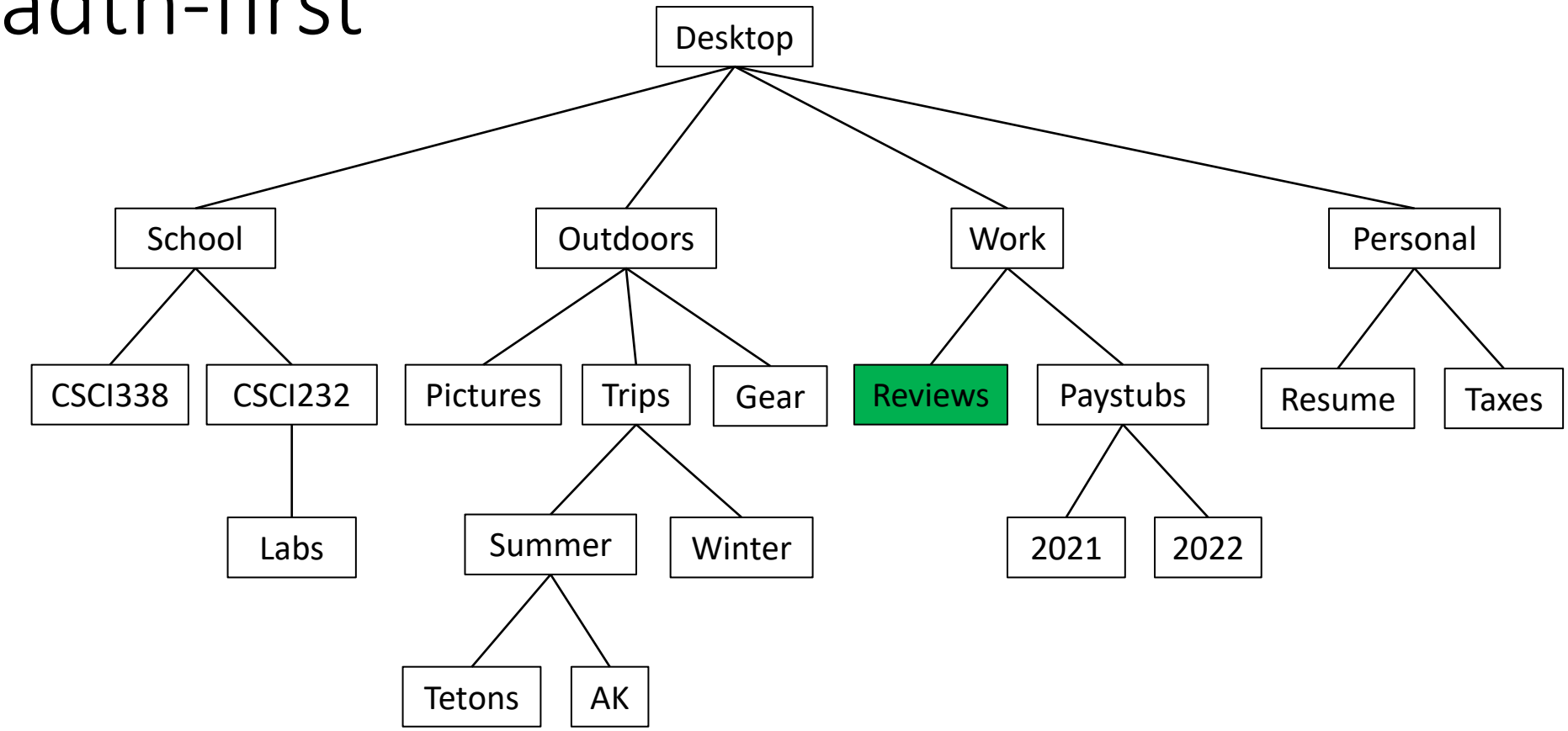
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



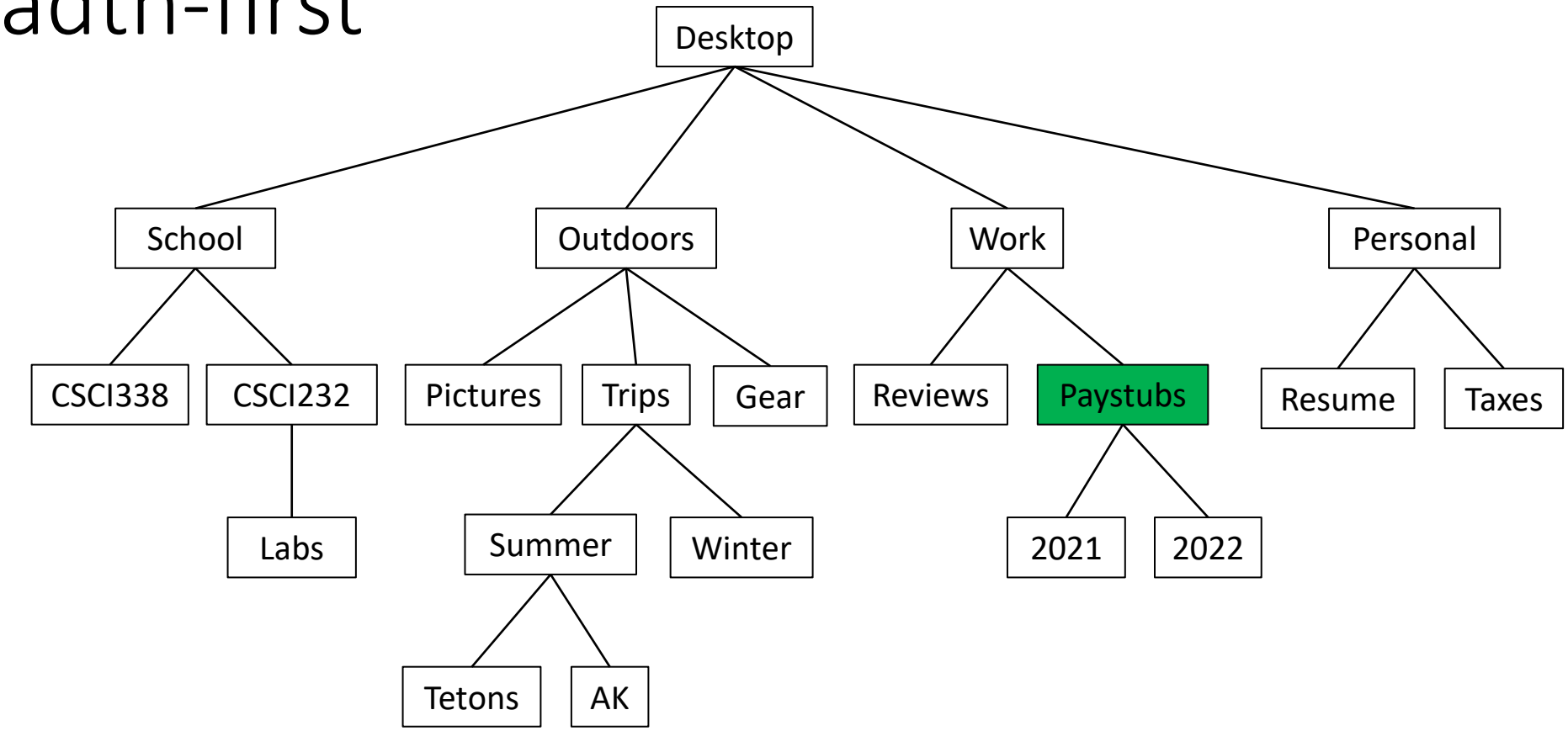
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



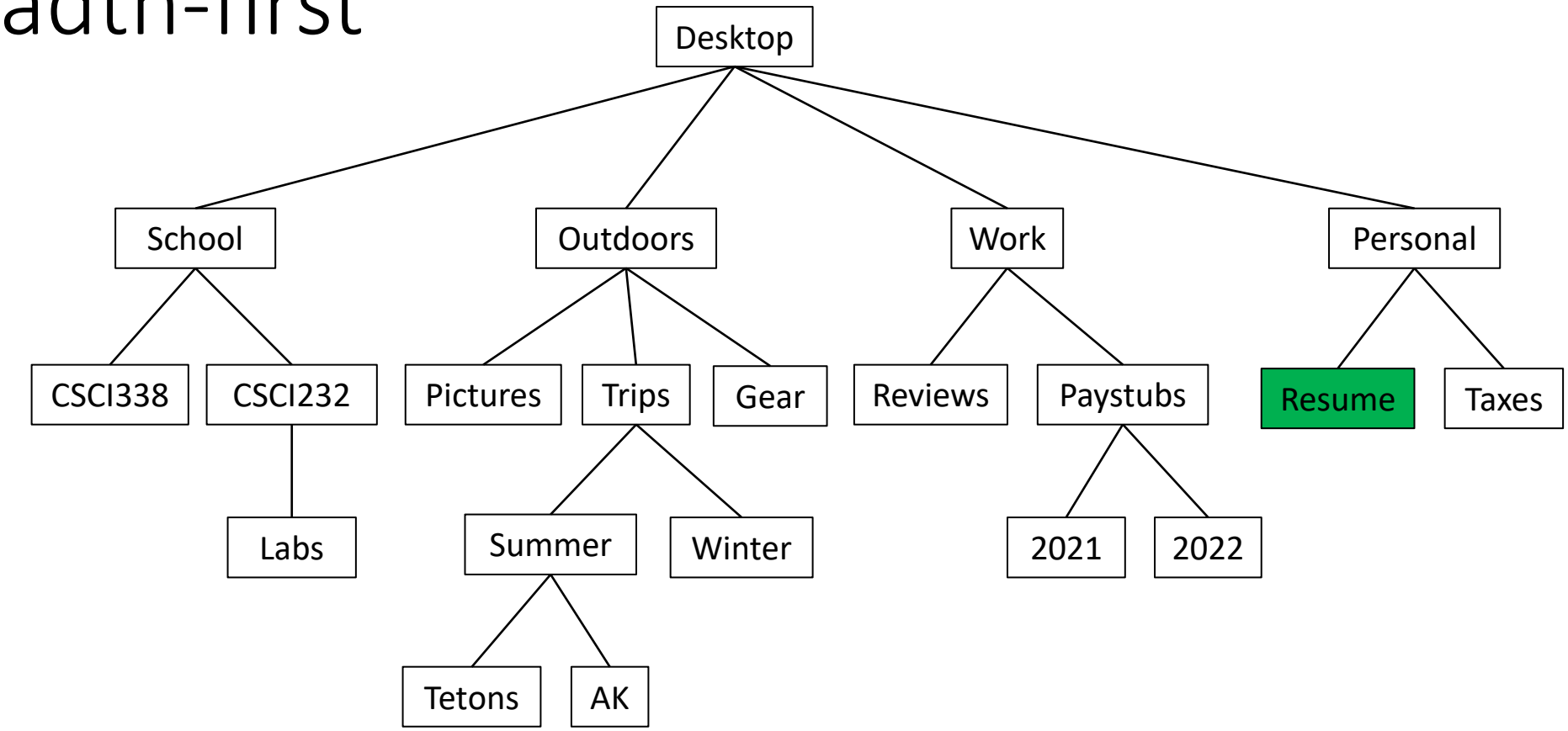
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



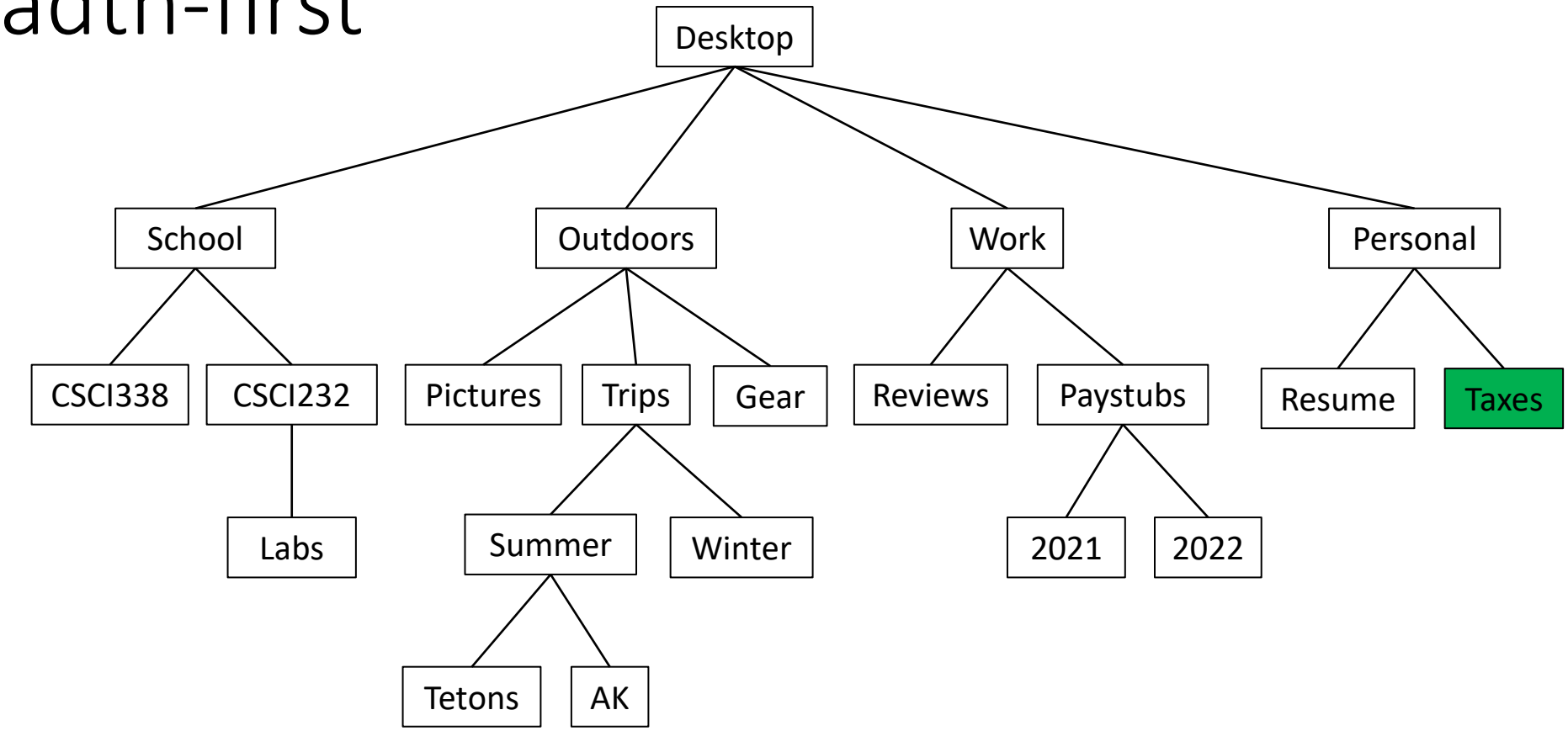
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



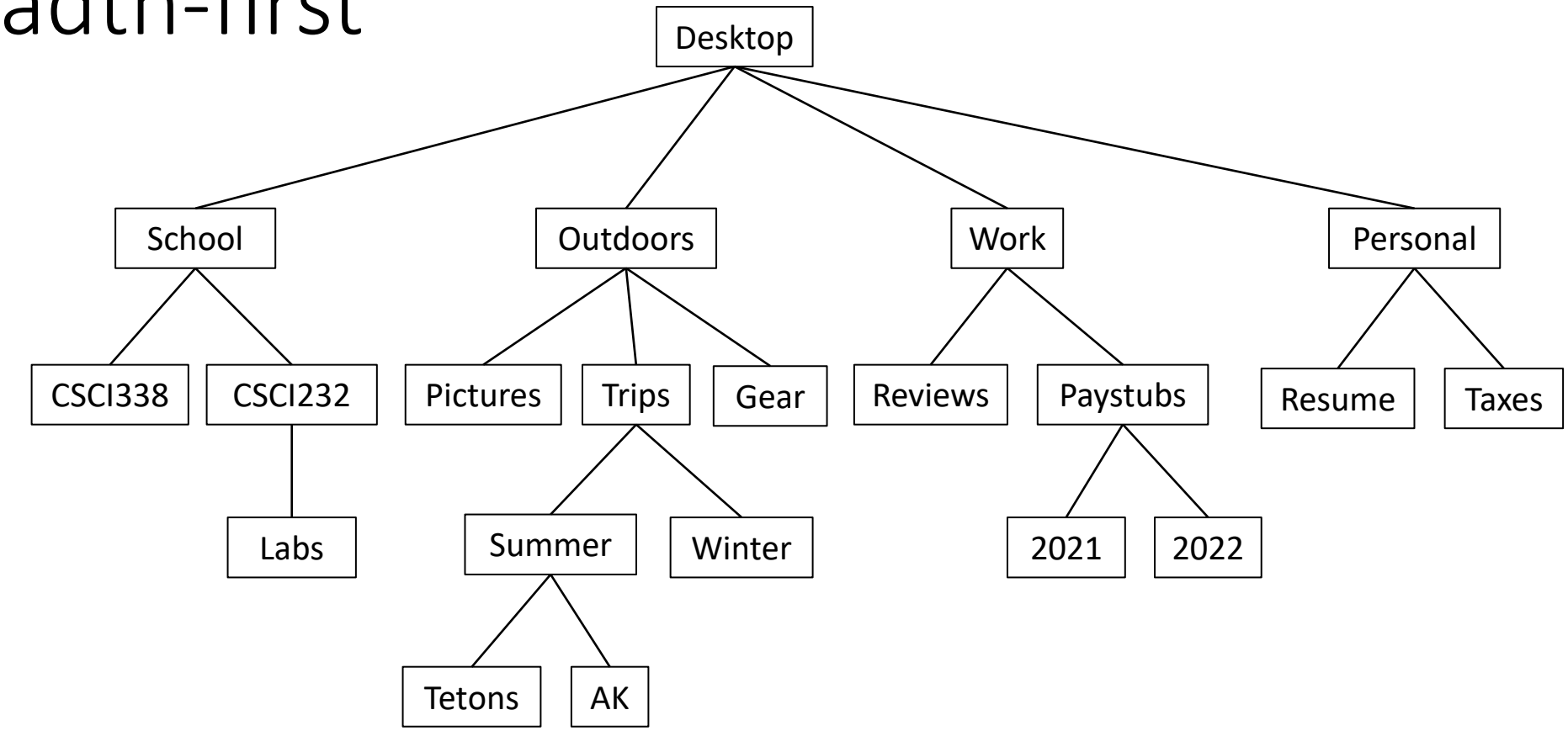
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



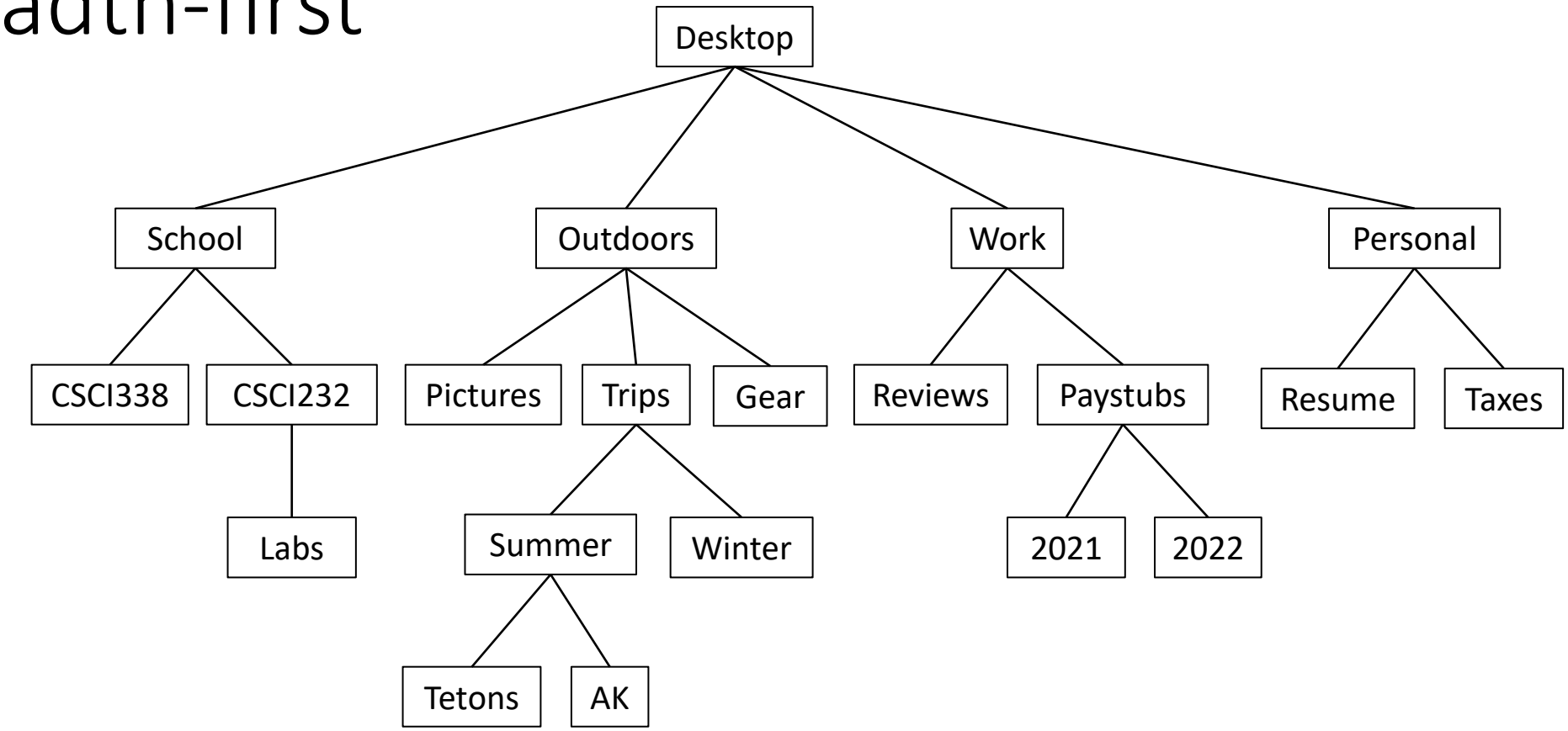
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).

Breadth-first



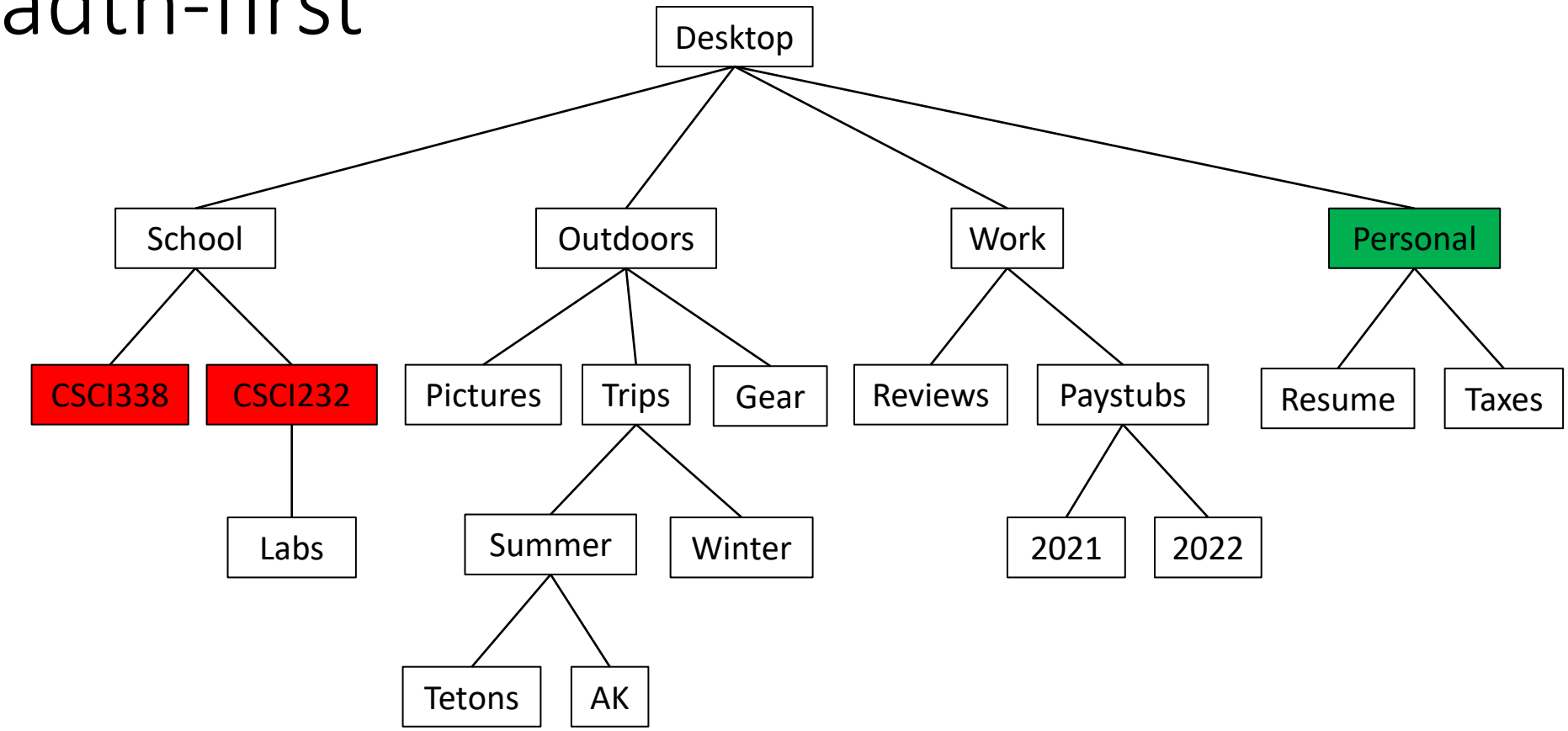
1. Visit the root.
2. Visit all depth 1 nodes (in order).
3. Visit all depth 2 nodes (in order).
4. …

Breadth-first



How to implement this?

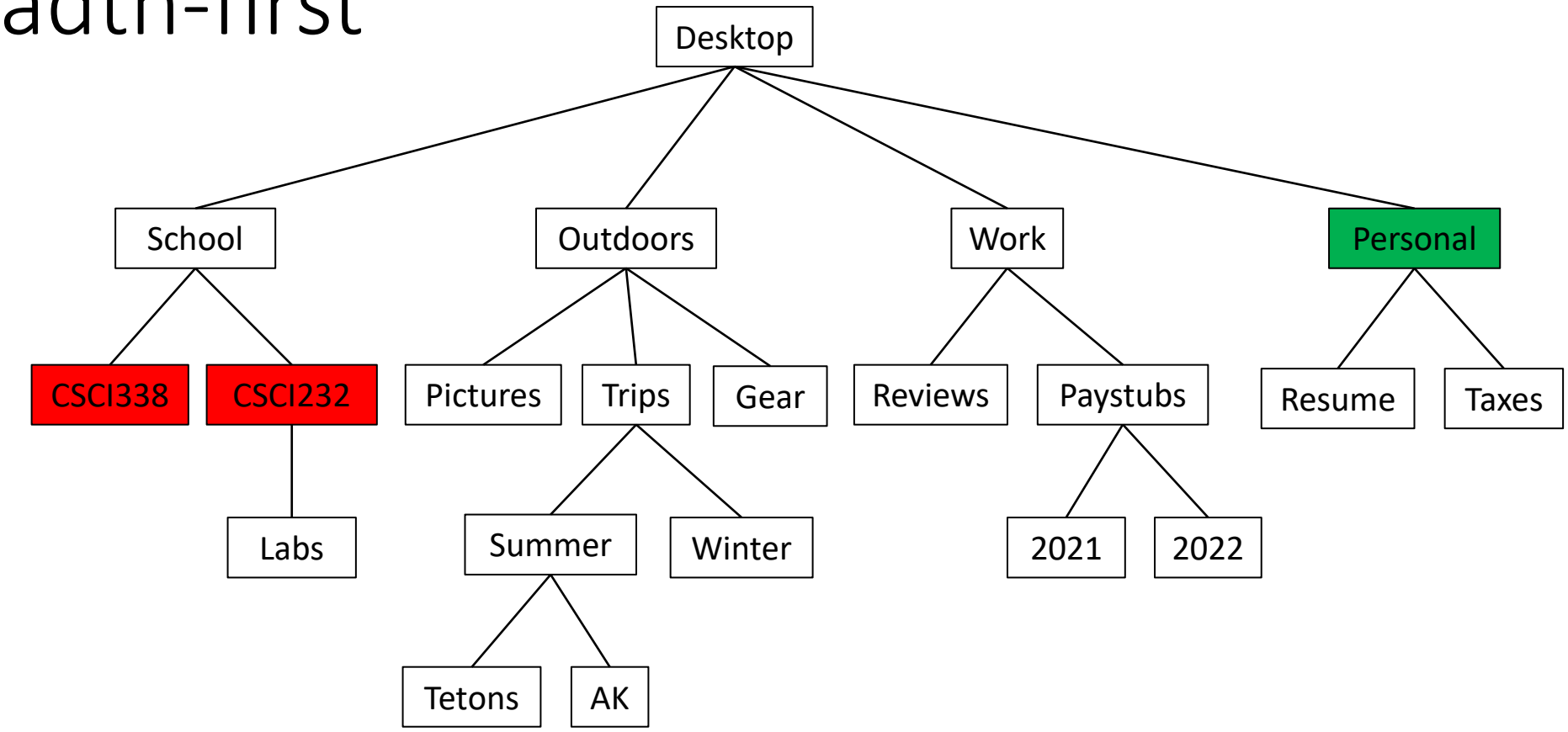
Breadth-first



How to implement this?

How do we know that the **children of School** are the nodes to visit after **Personal**?

Breadth-first

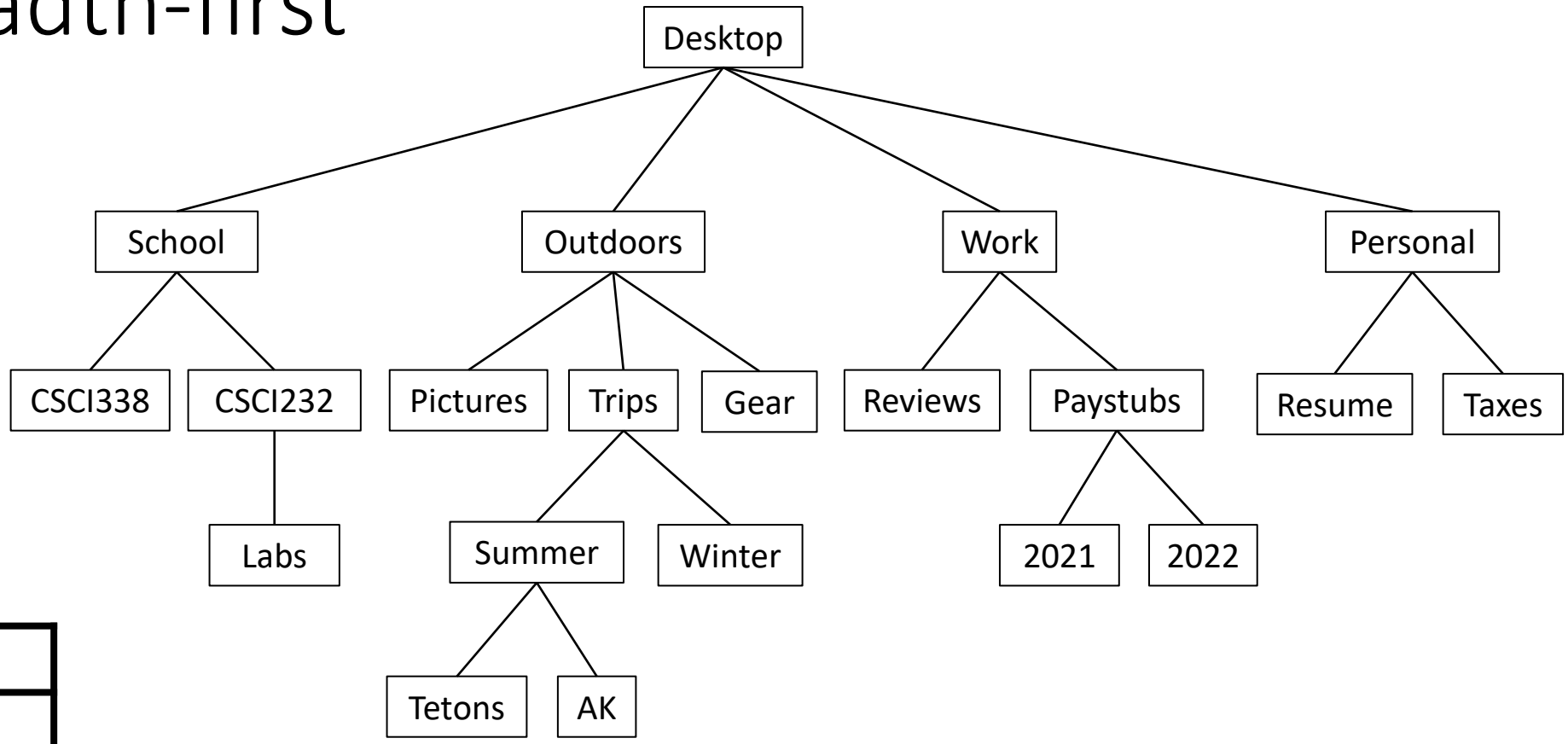


How to implement this?

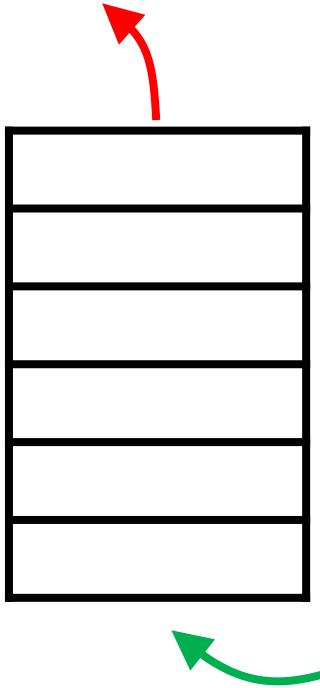
How do we know that the **children of School** are the nodes to visit after **Personal**?

What if we use a queue?

Breadth-first

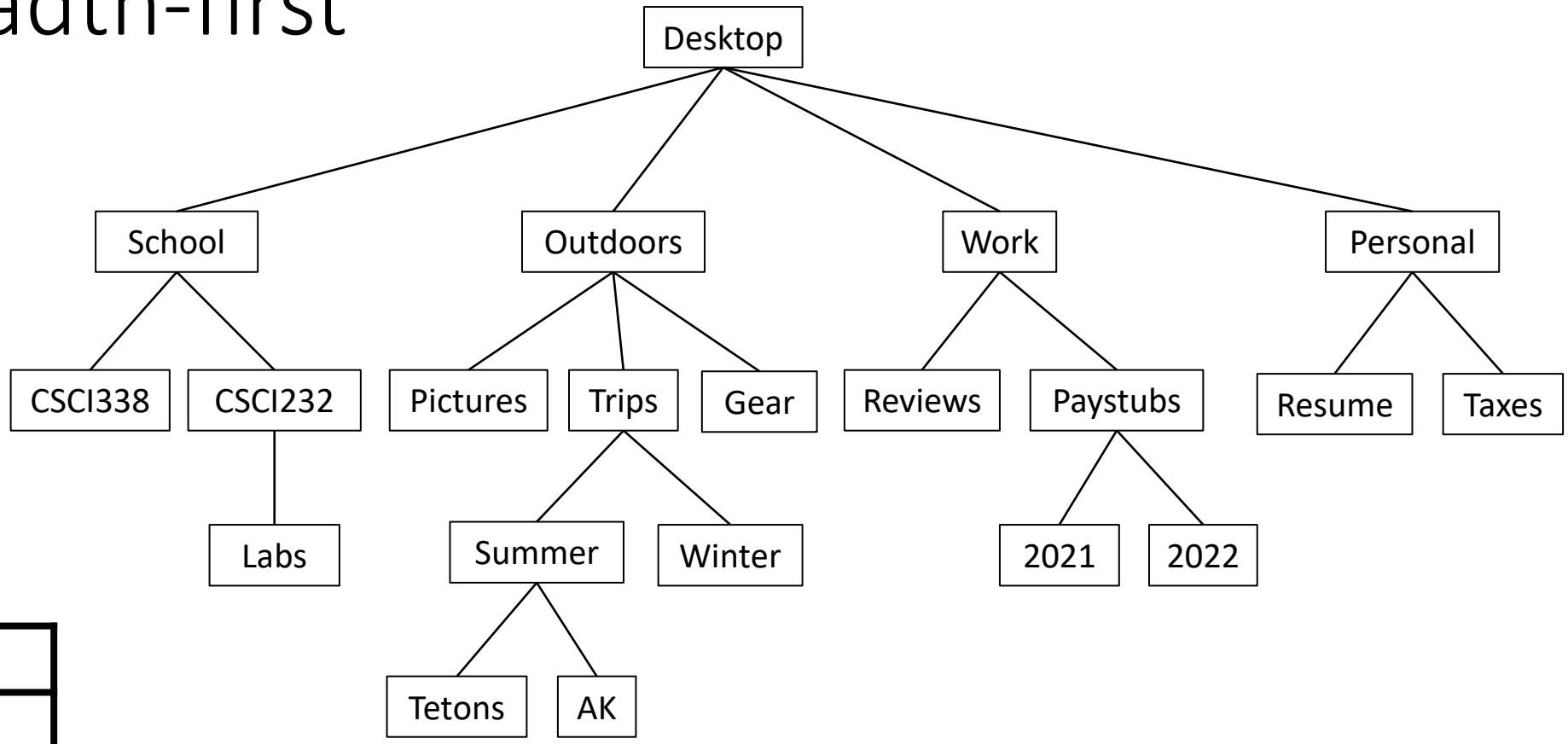


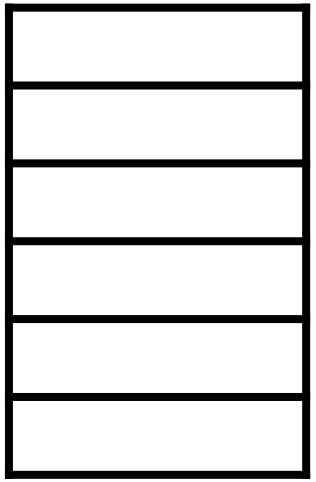
First out



First in

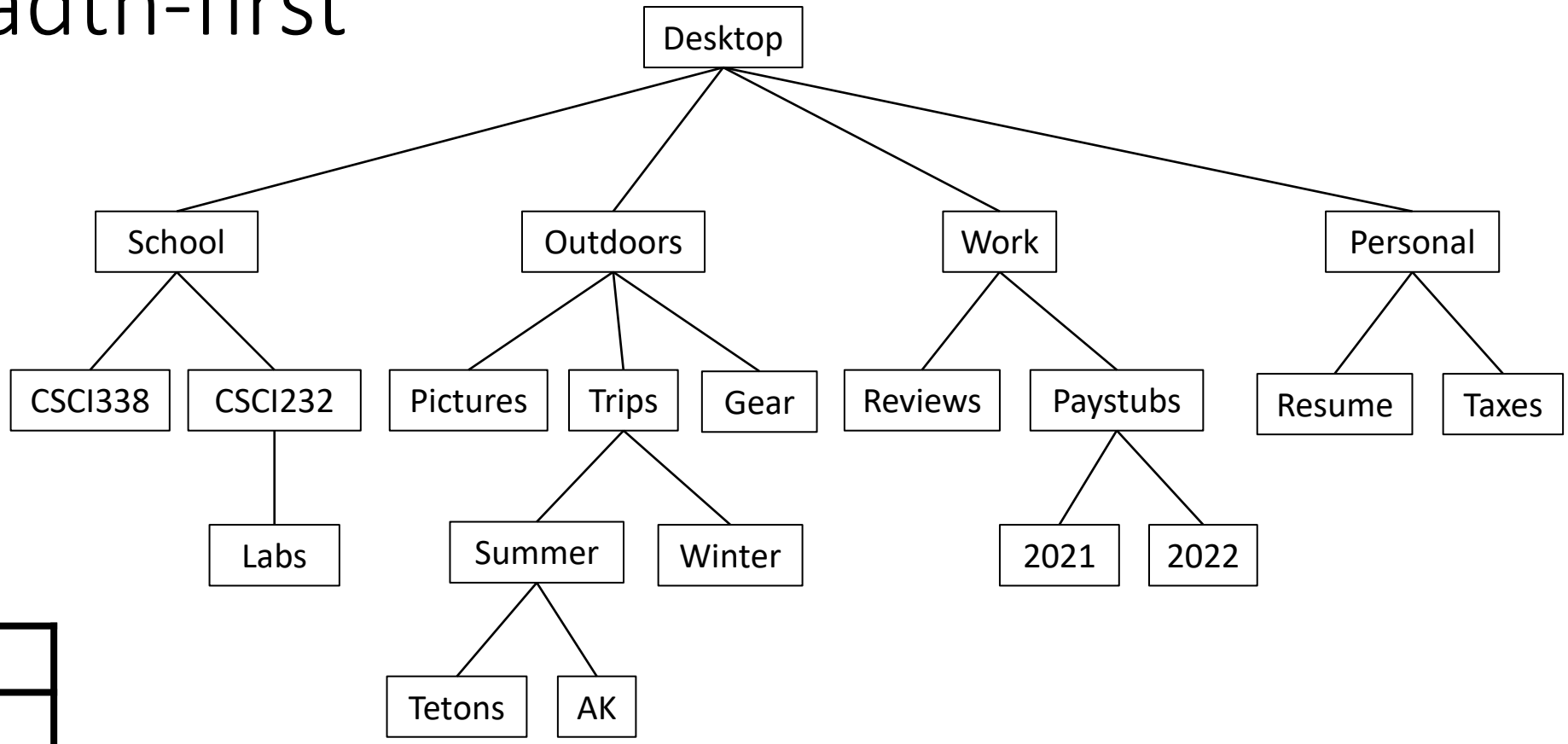
Breadth-first

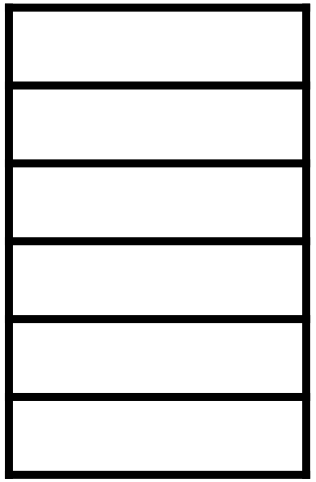




Every time we “visit” a node we:

Breadth-first

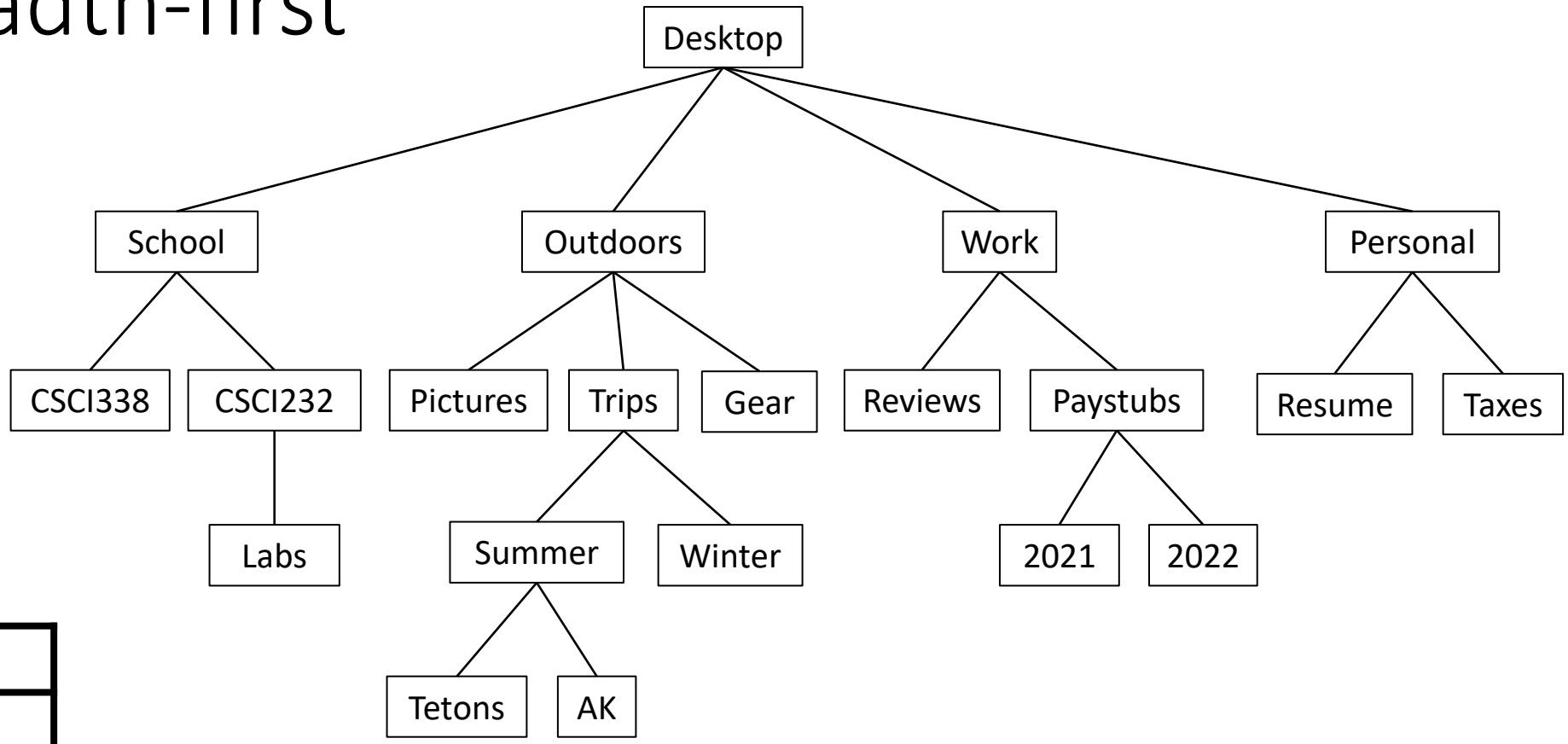


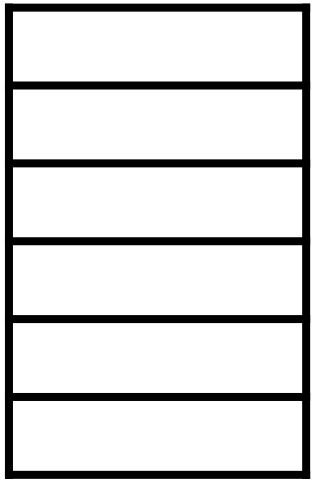


Every time we “visit” a node we:

1. Execute the action (e.g., print, compare,…).

Breadth-first

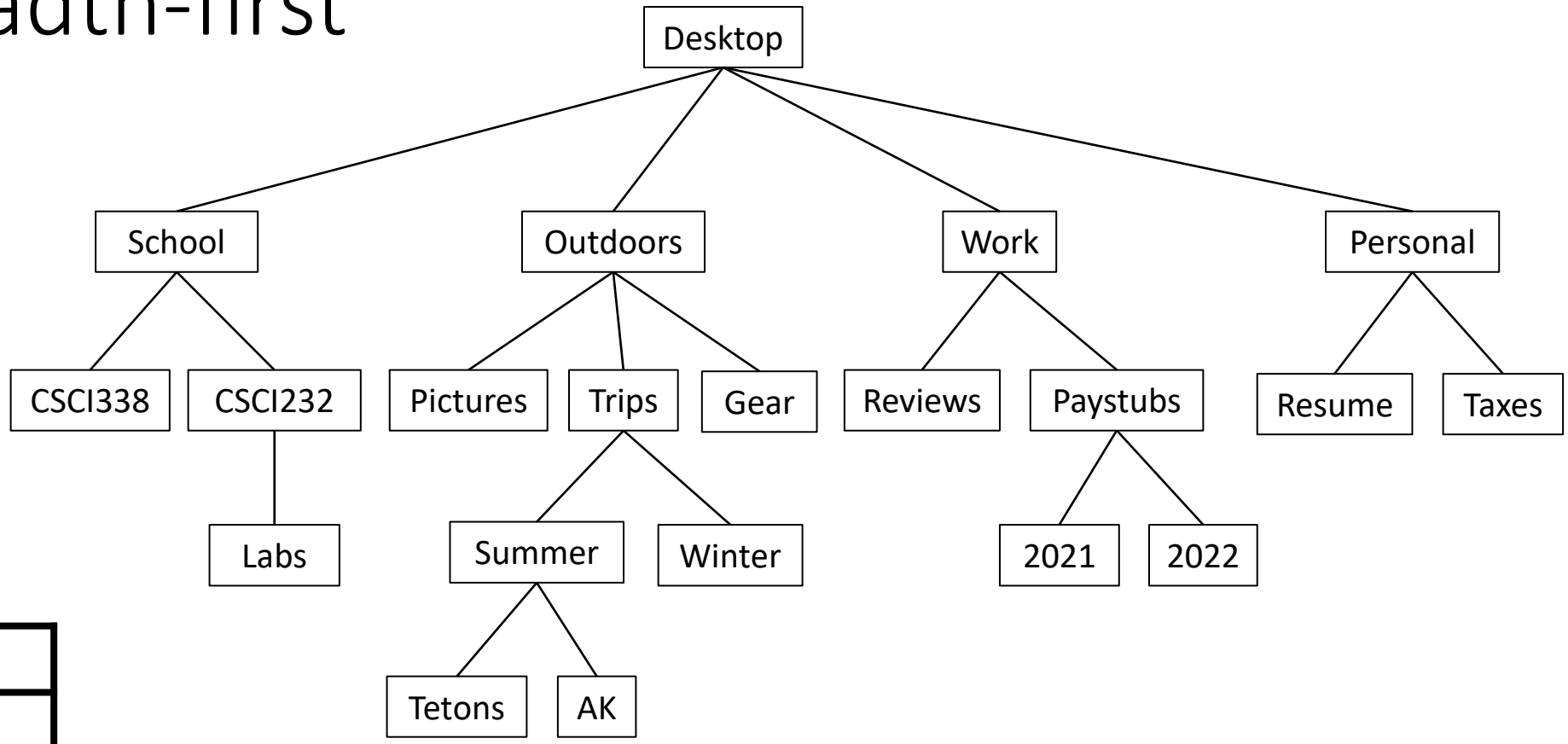


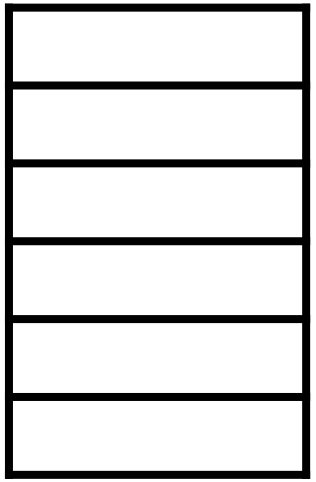


Every time we “visit” a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.

Breadth-first

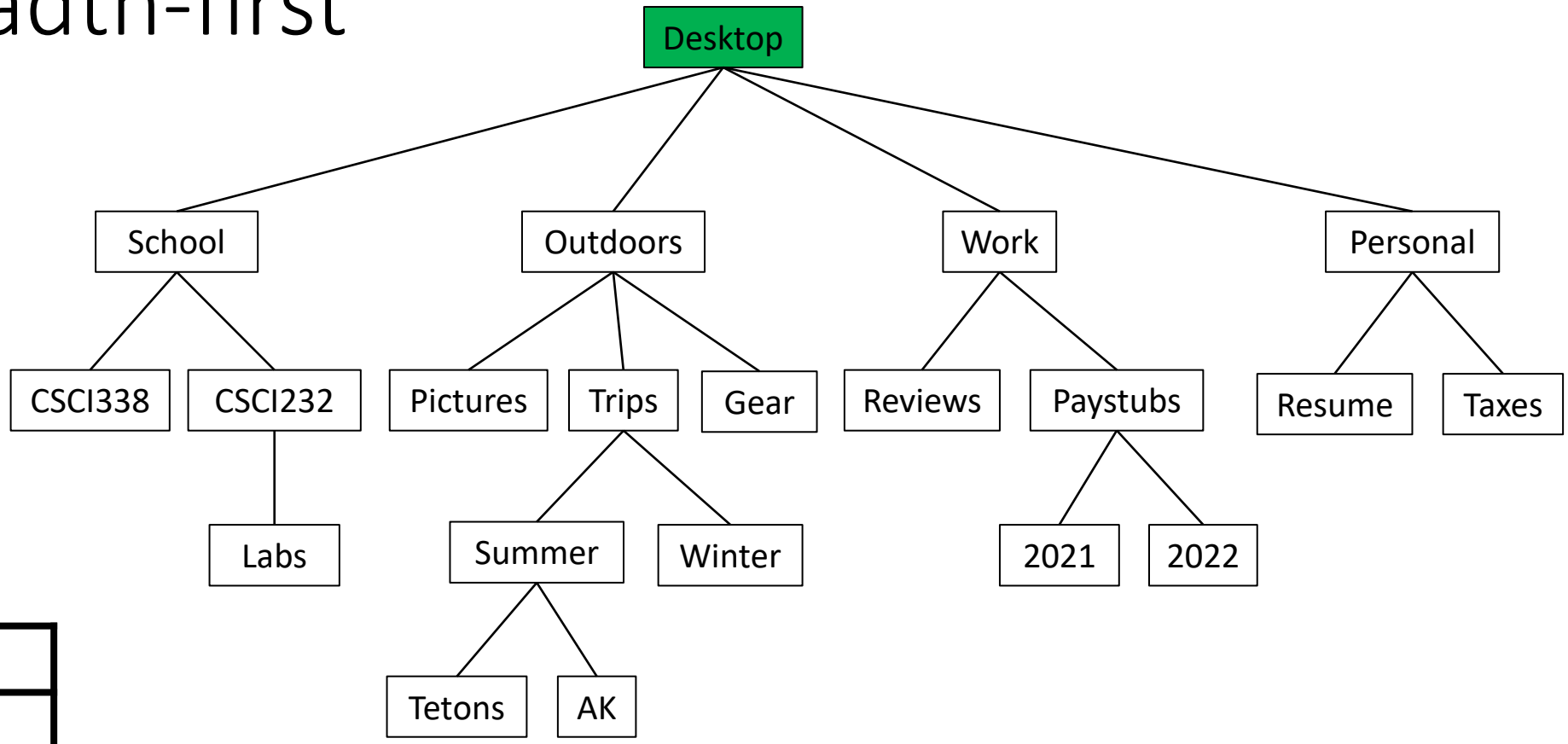




Every time we “visit” a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

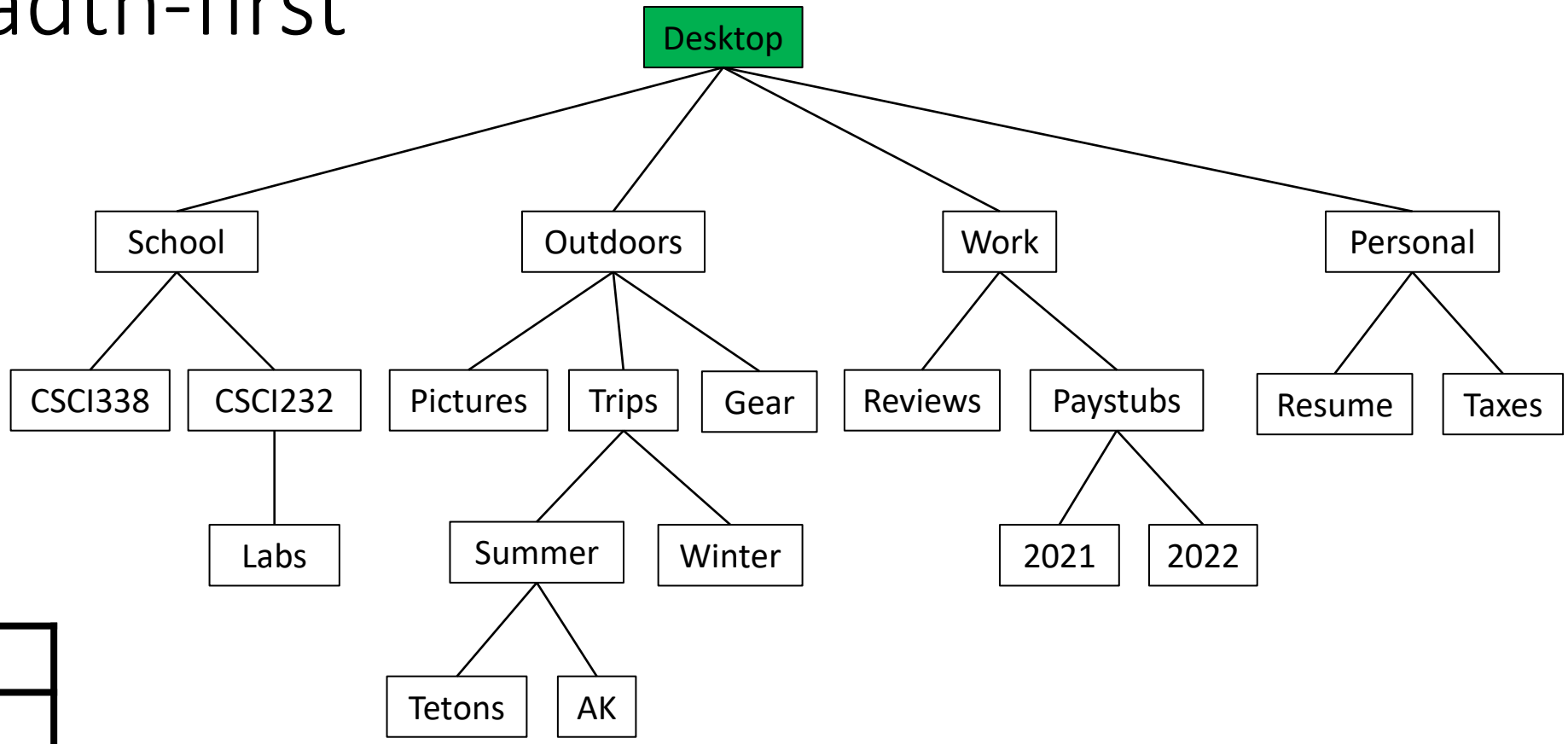


Desktop

Every time we “visit” a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

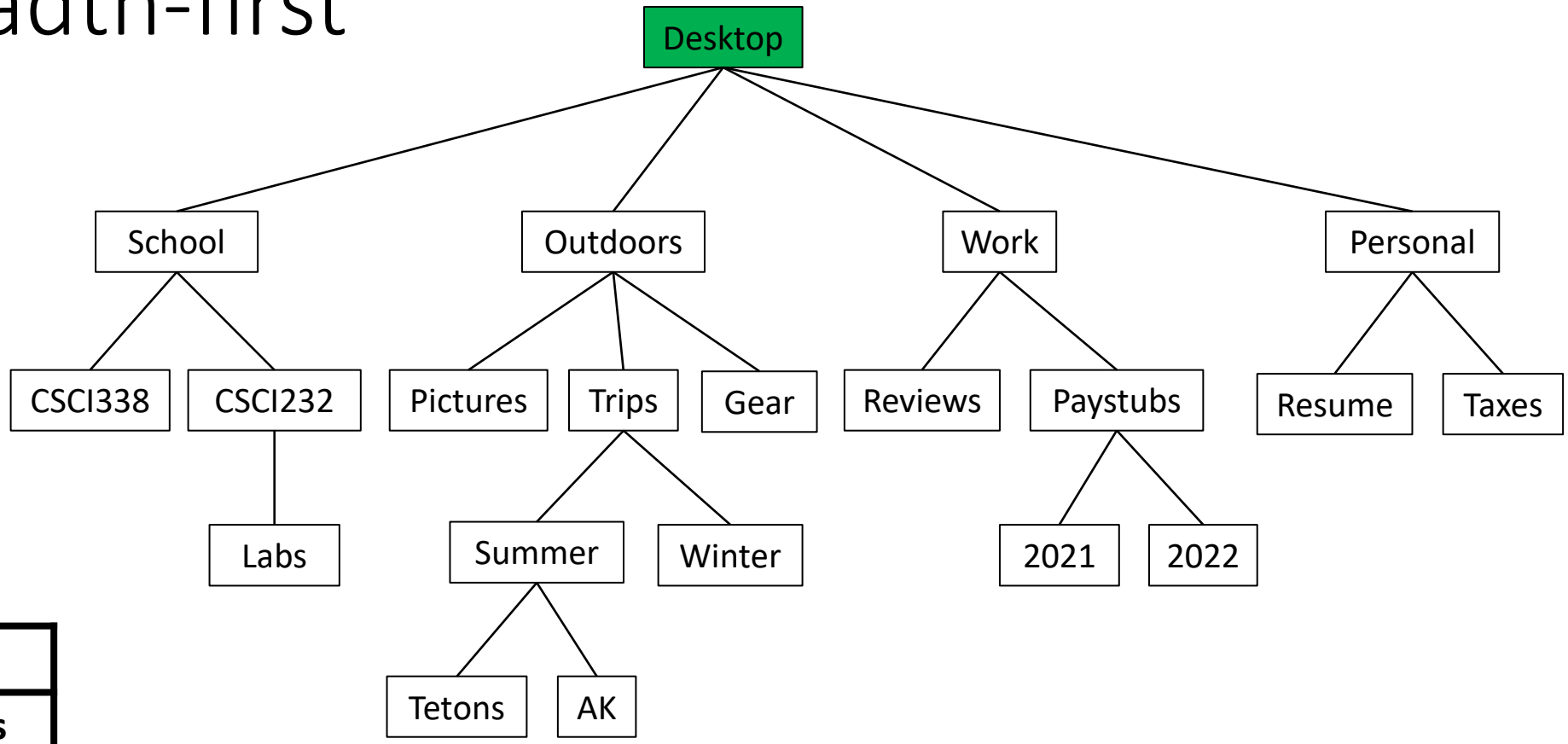


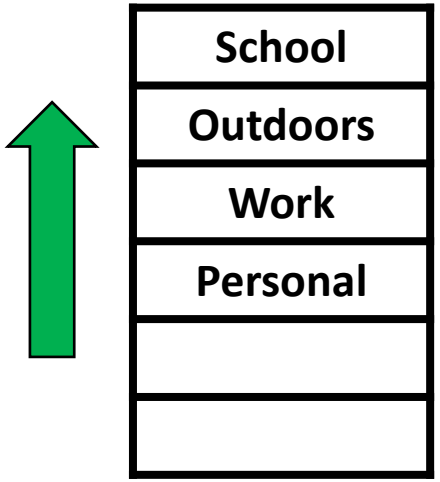
Desktop
School
Outdoors
Work
Personal

Every time we “visit” a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

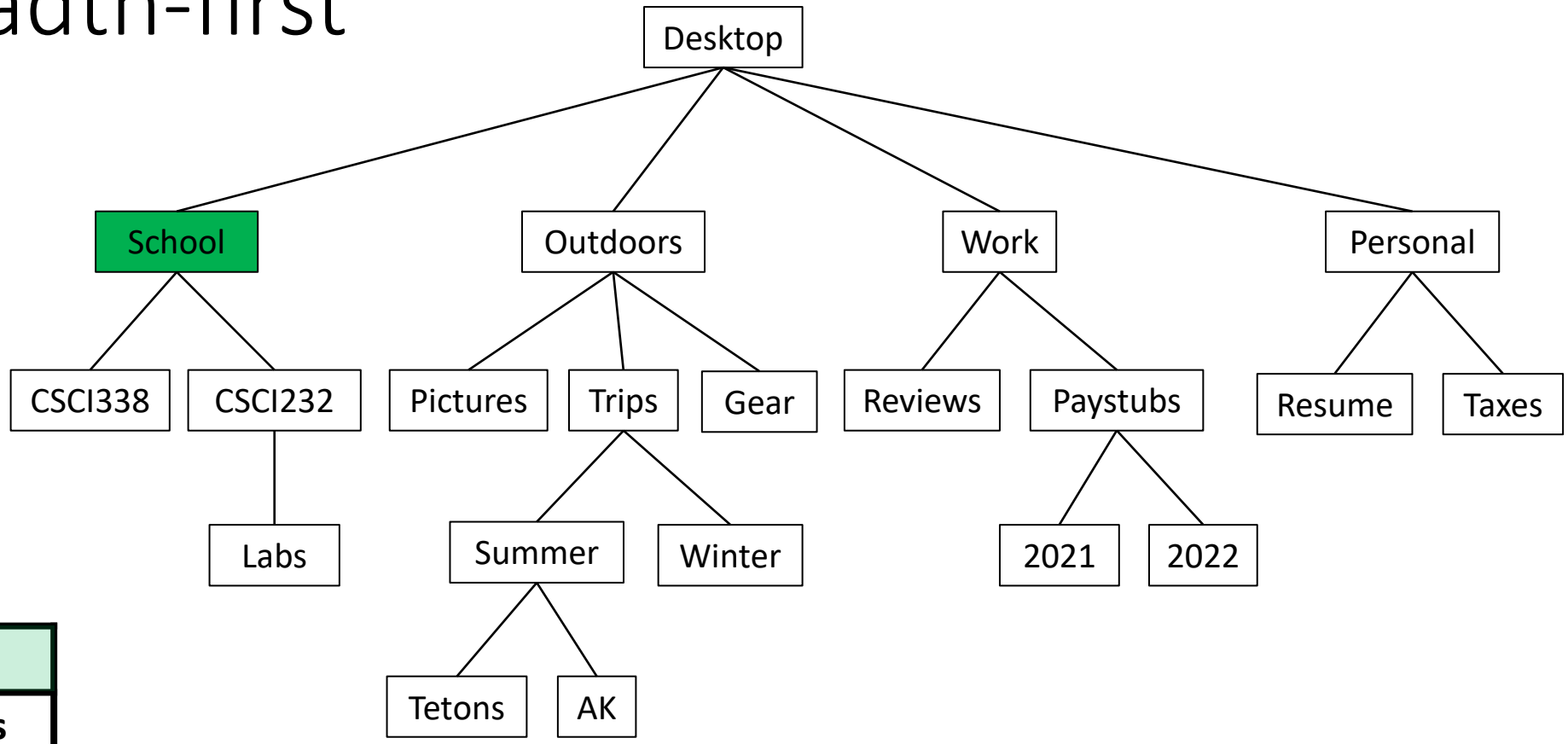




Every time we "visit" a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

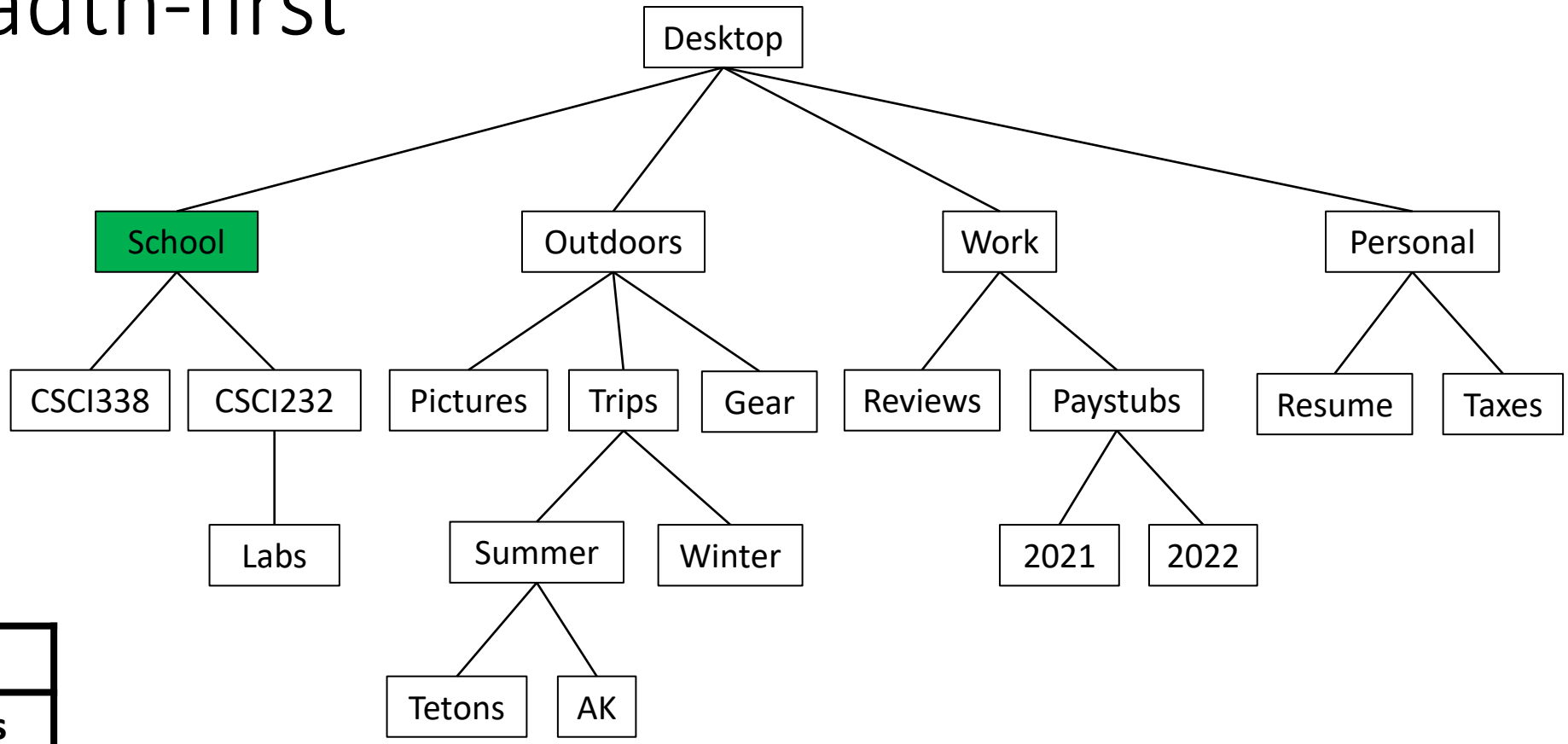


School
Outdoors
Work
Personal

Every time we "visit" a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

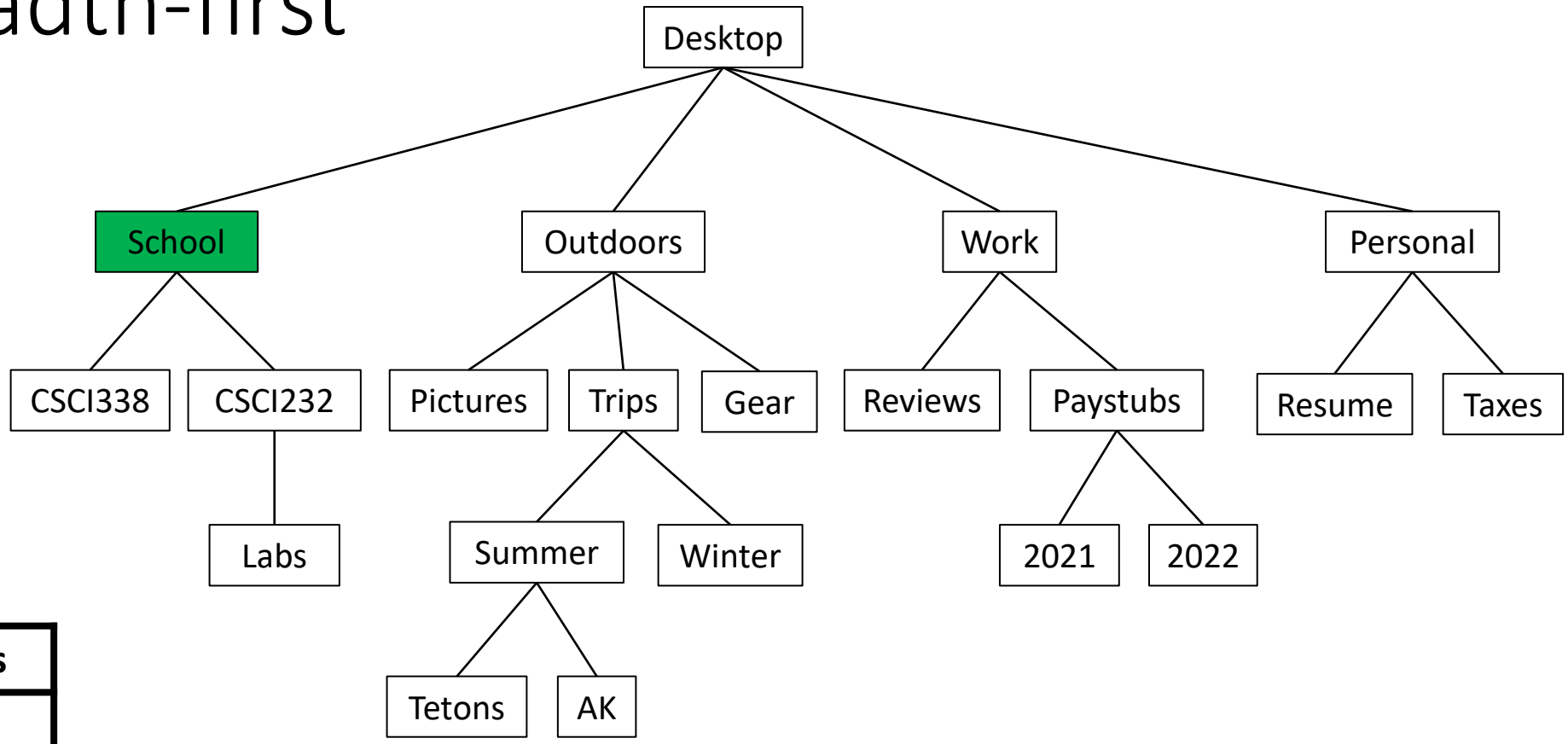


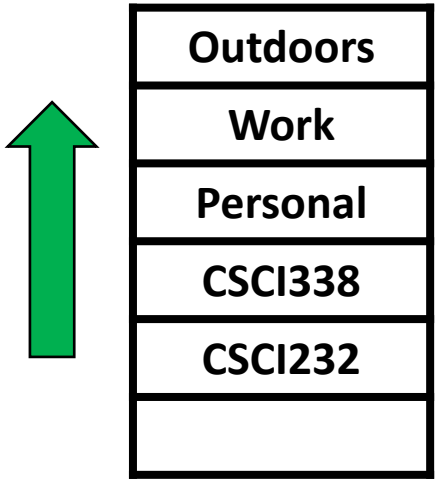
School
Outdoors
Work
Personal
CSCI338
CSCI232

Every time we "visit" a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Breadth-first

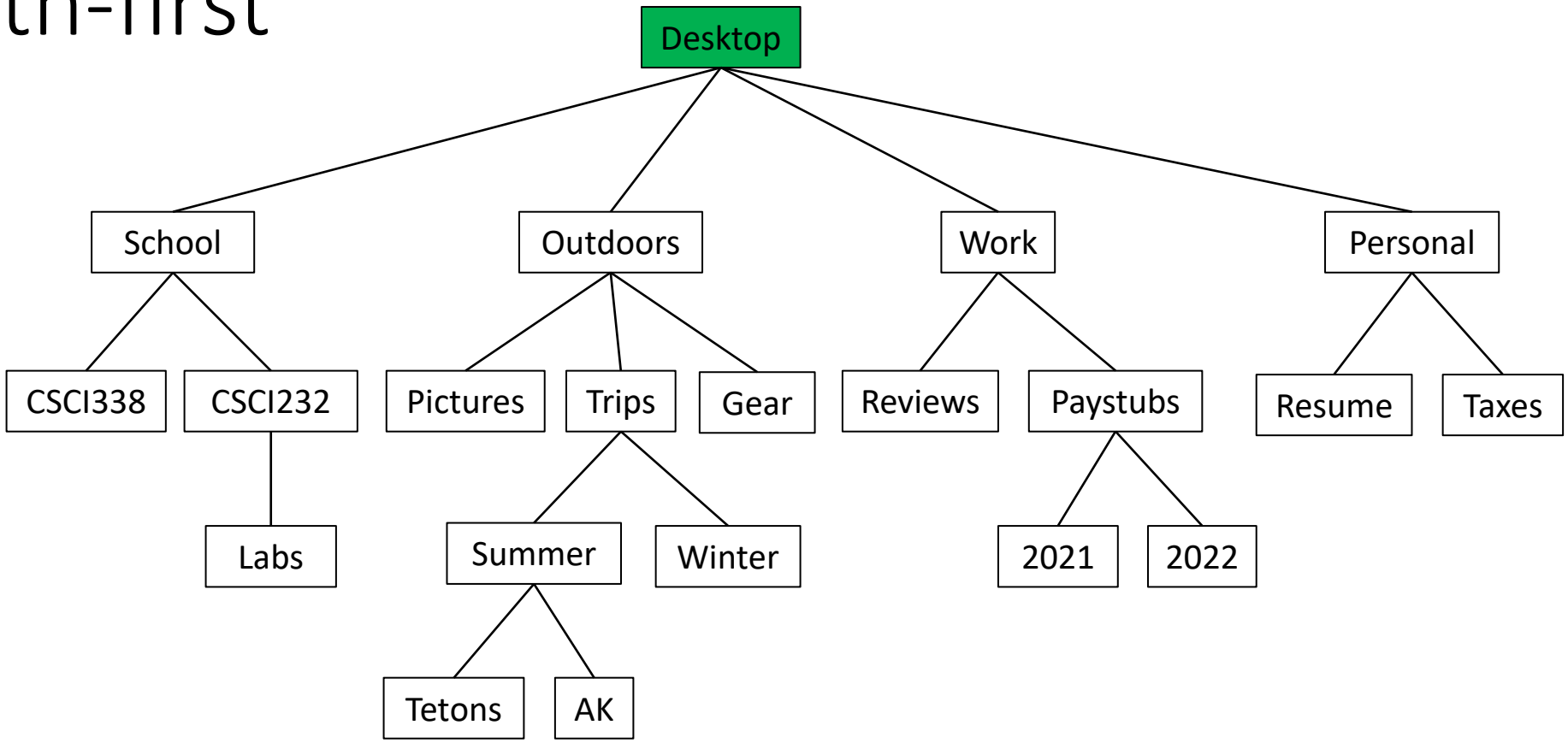




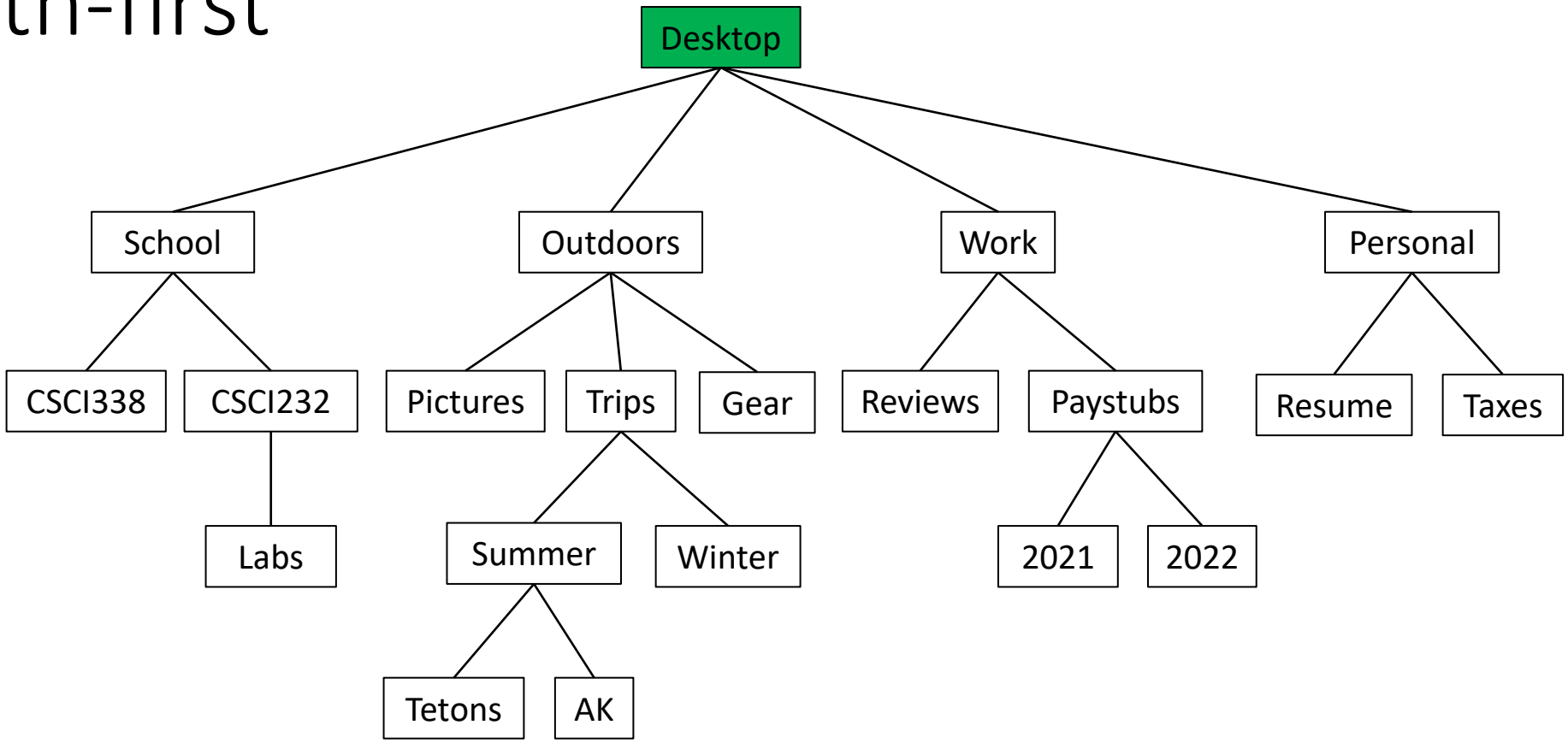
Every time we "visit" a node we:

1. Execute the action (e.g., print, compare,…).
2. Add all of its children to the queue.
3. Remove visited node from queue.

Depth-first

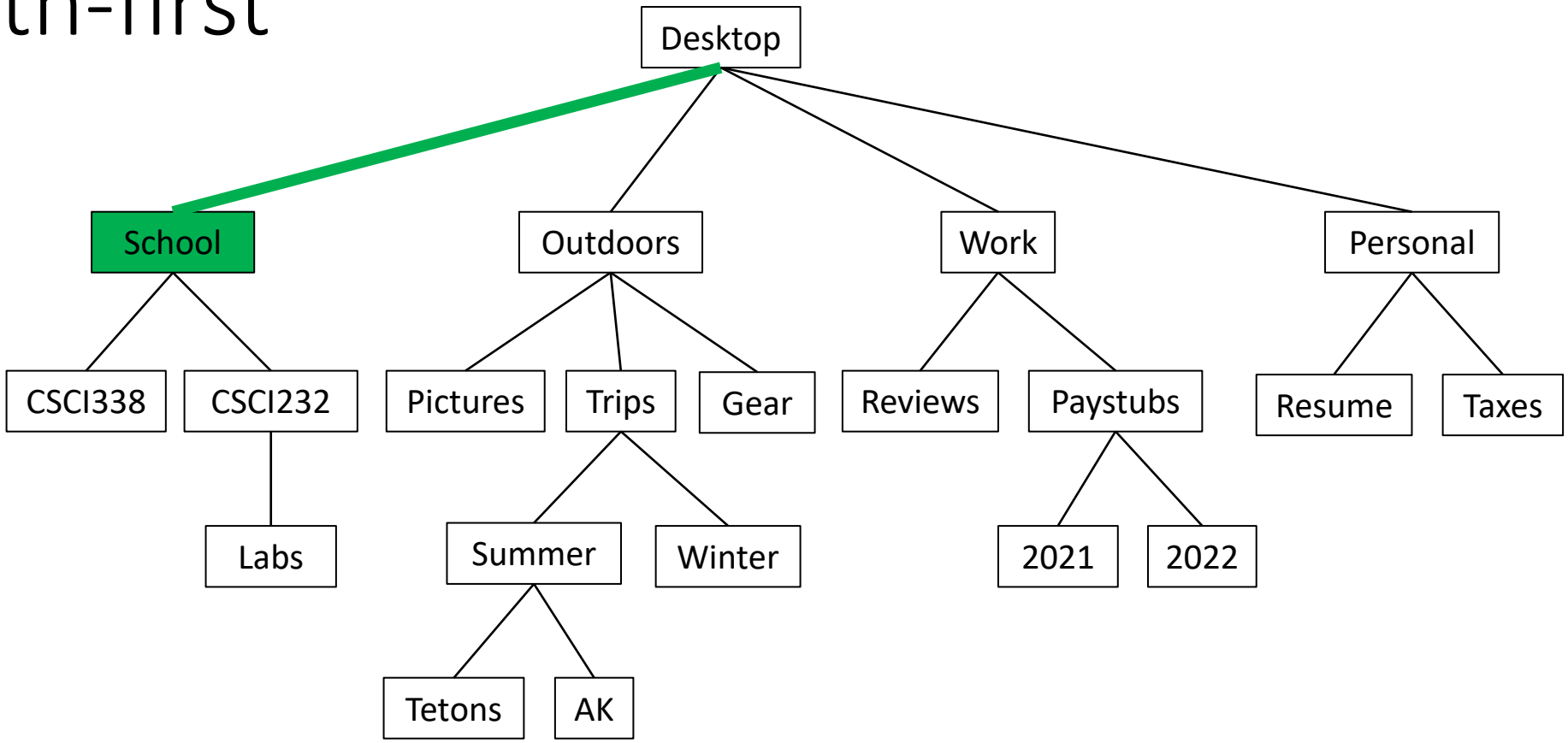

Desktop
School
Outdoors
Work
Personal
CSCI338
CSCI232
Pictures
Trips
Gear
Reviews
Paystubs
Resume
Taxes
Labs
Summer
Winter
2021
2022
Tetons
AK

Depth-first



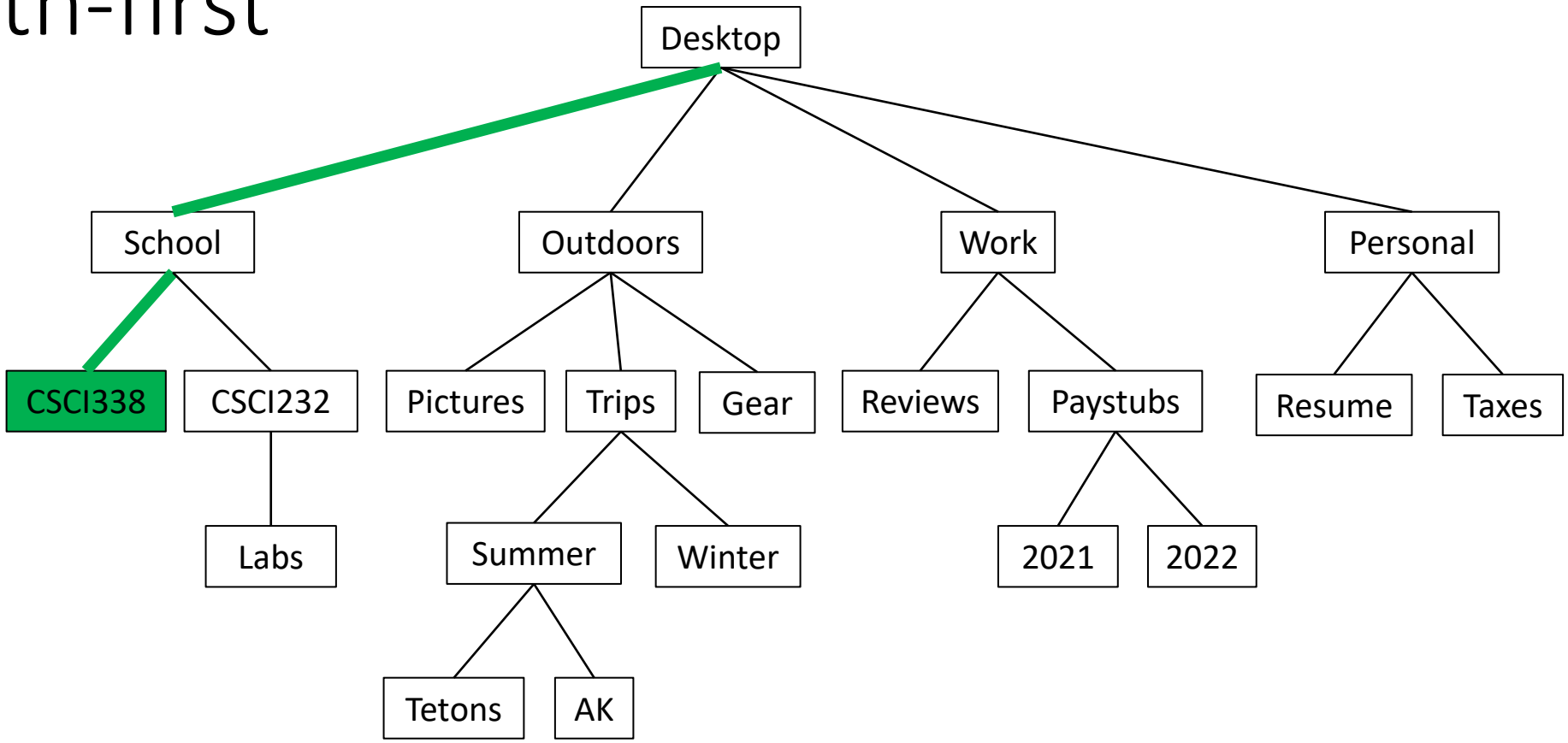
1. Go all the way down to the “first” leaf.

Depth-first



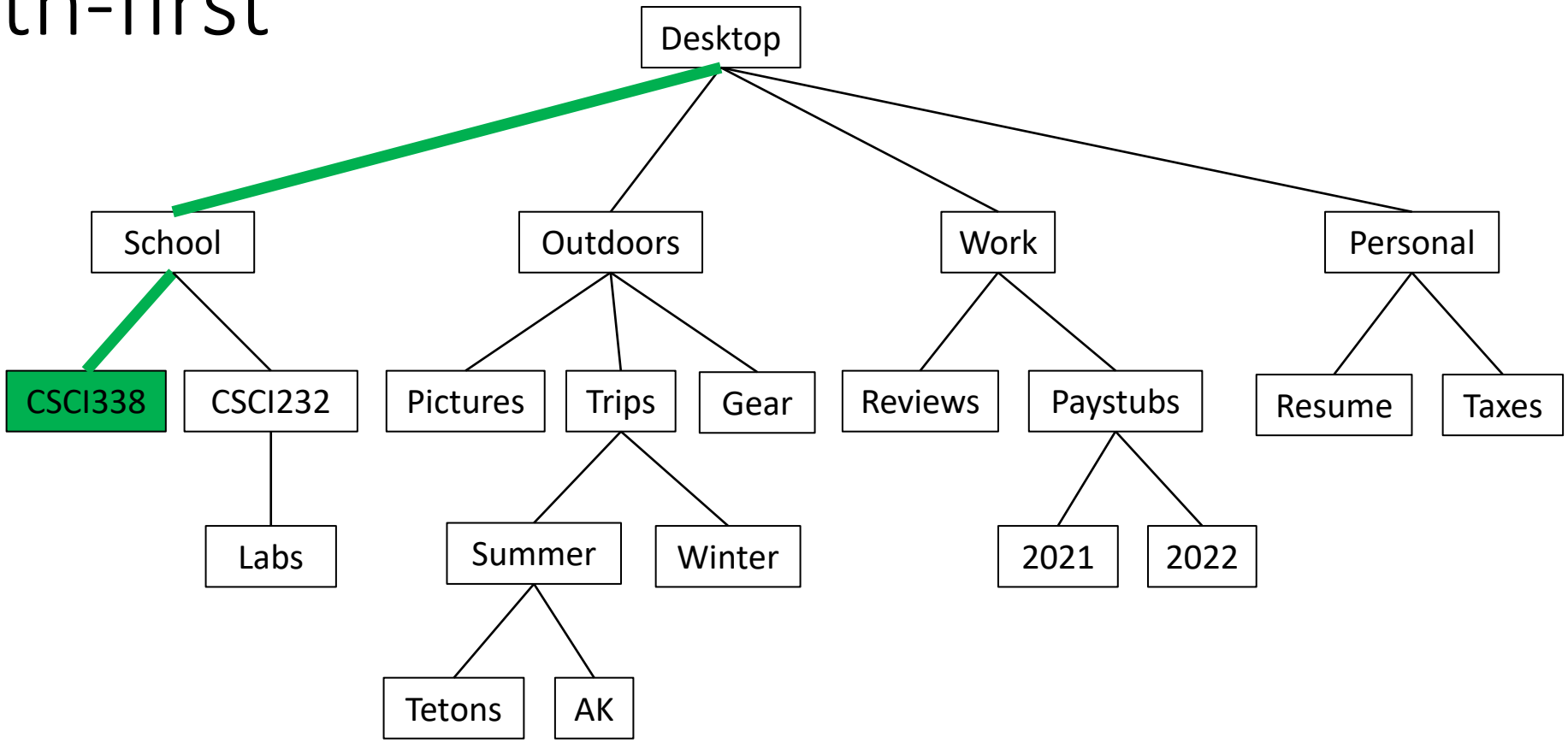
1. Go all the way down to the “first” leaf.

Depth-first



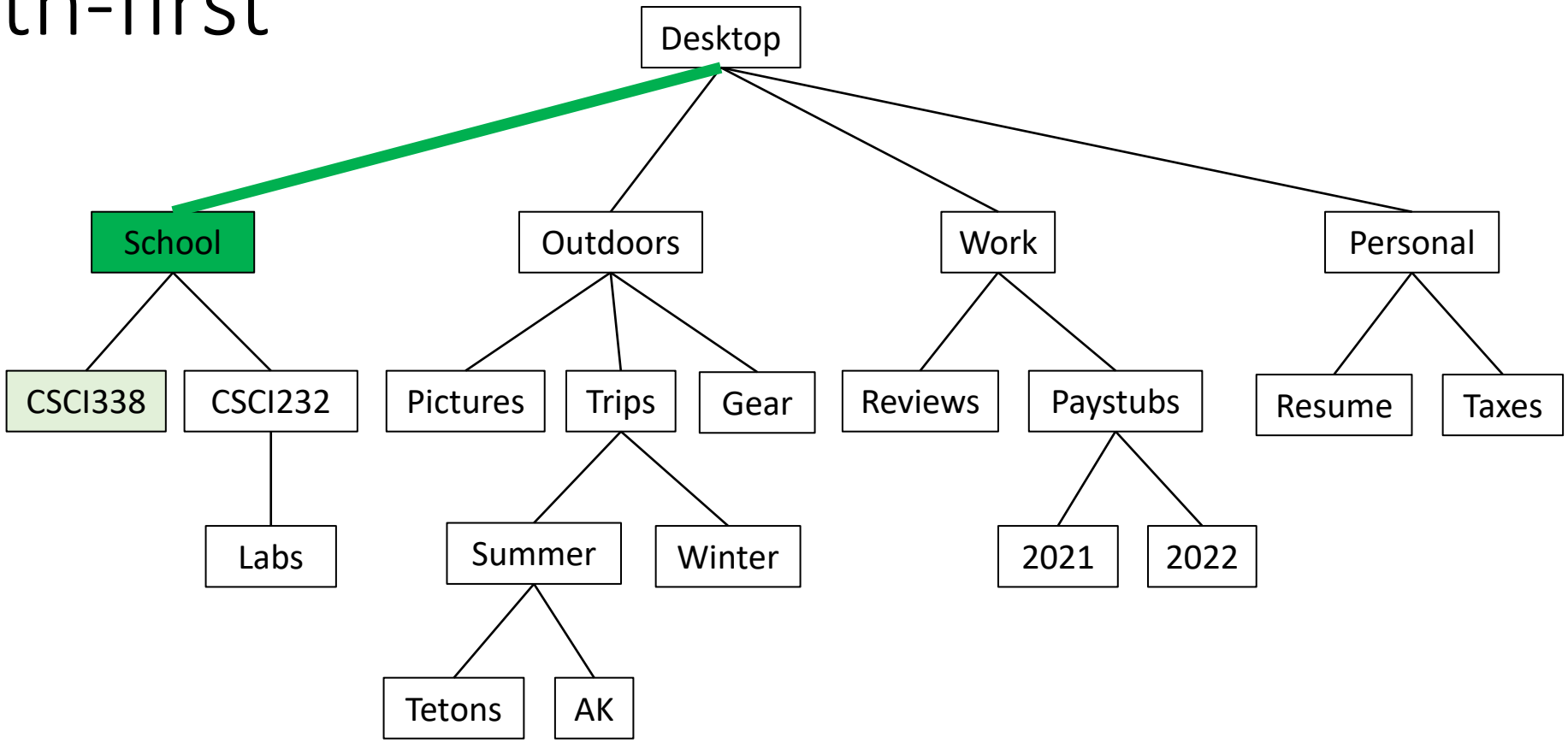
1. Go all the way down to the “first” leaf.

Depth-first



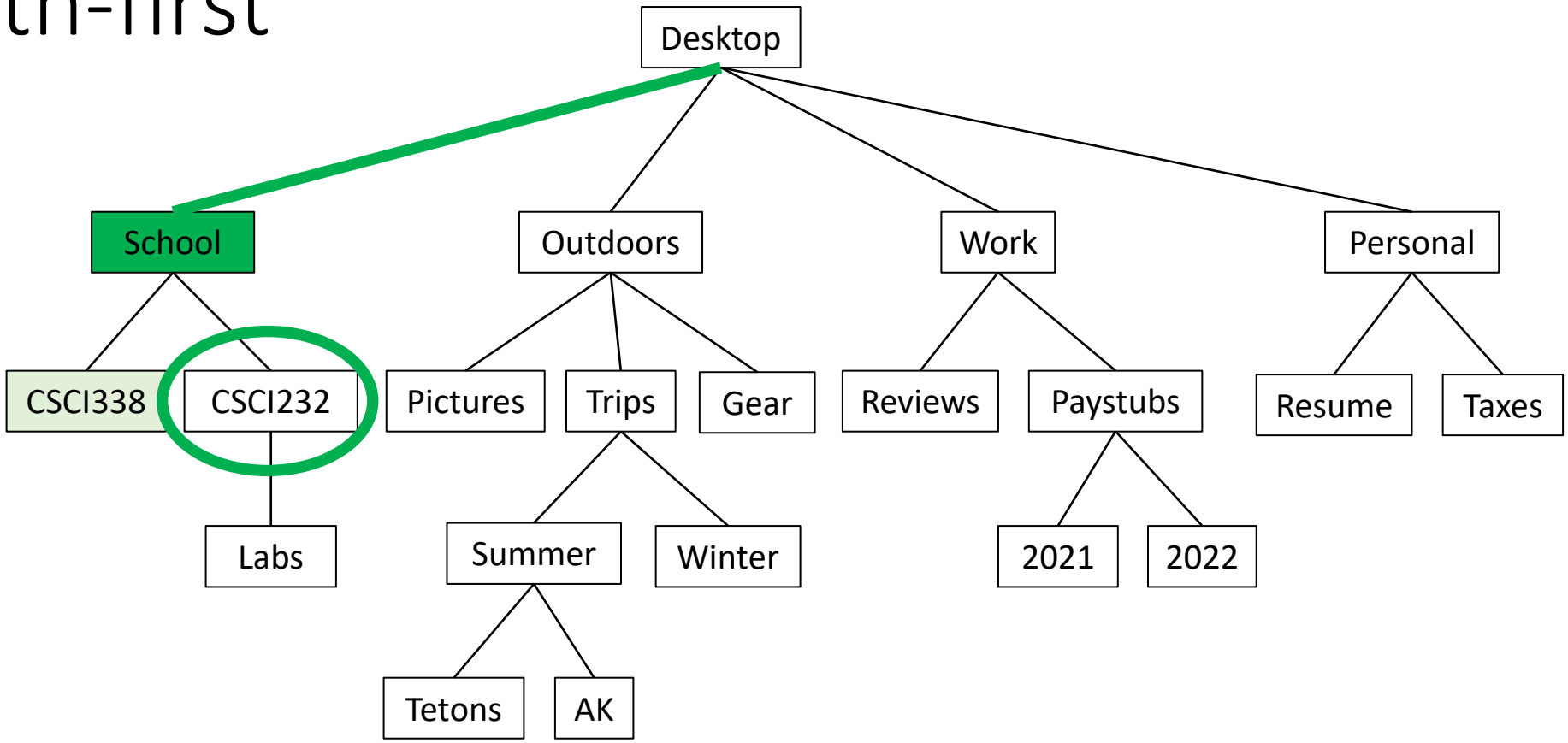
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.

Depth-first



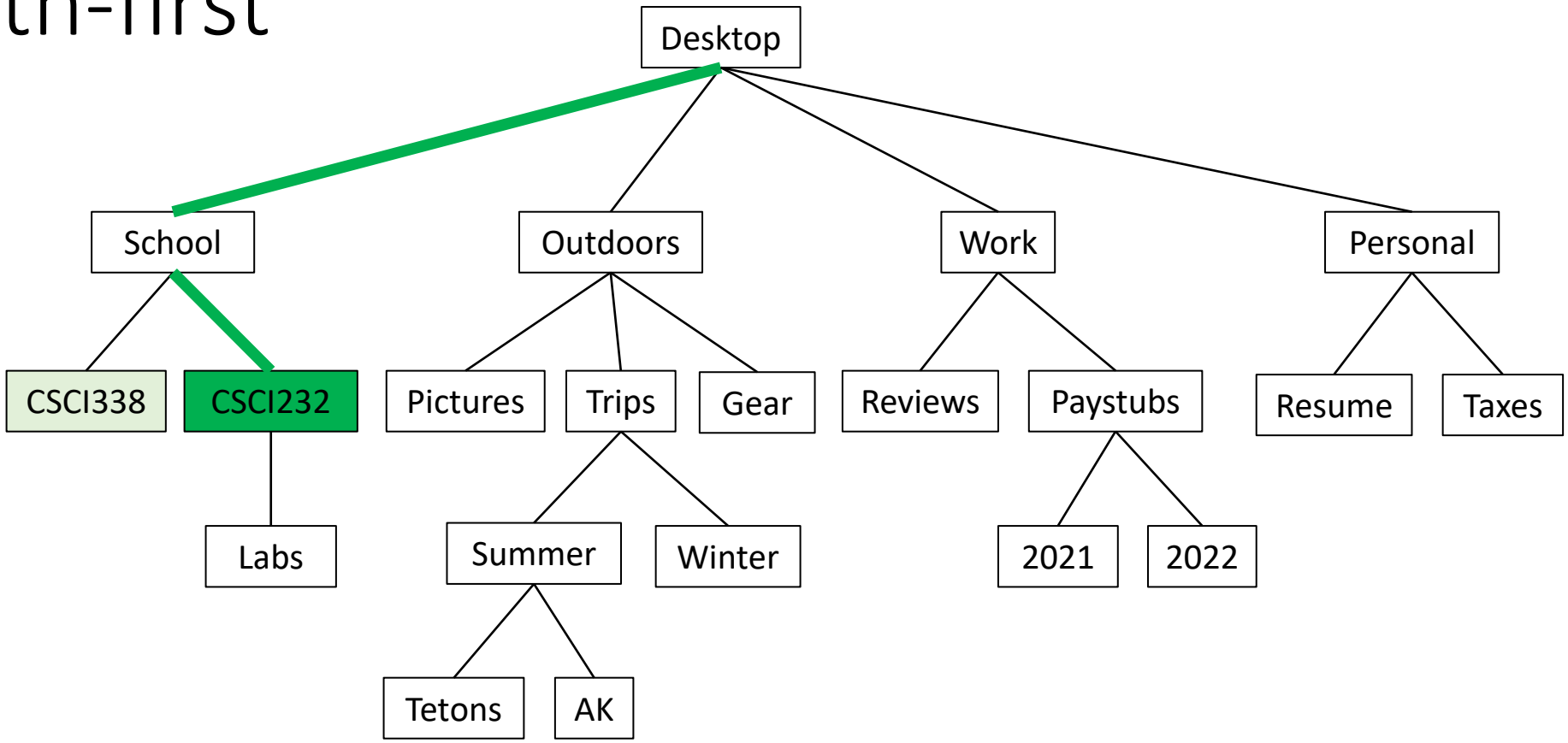
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.

Depth-first



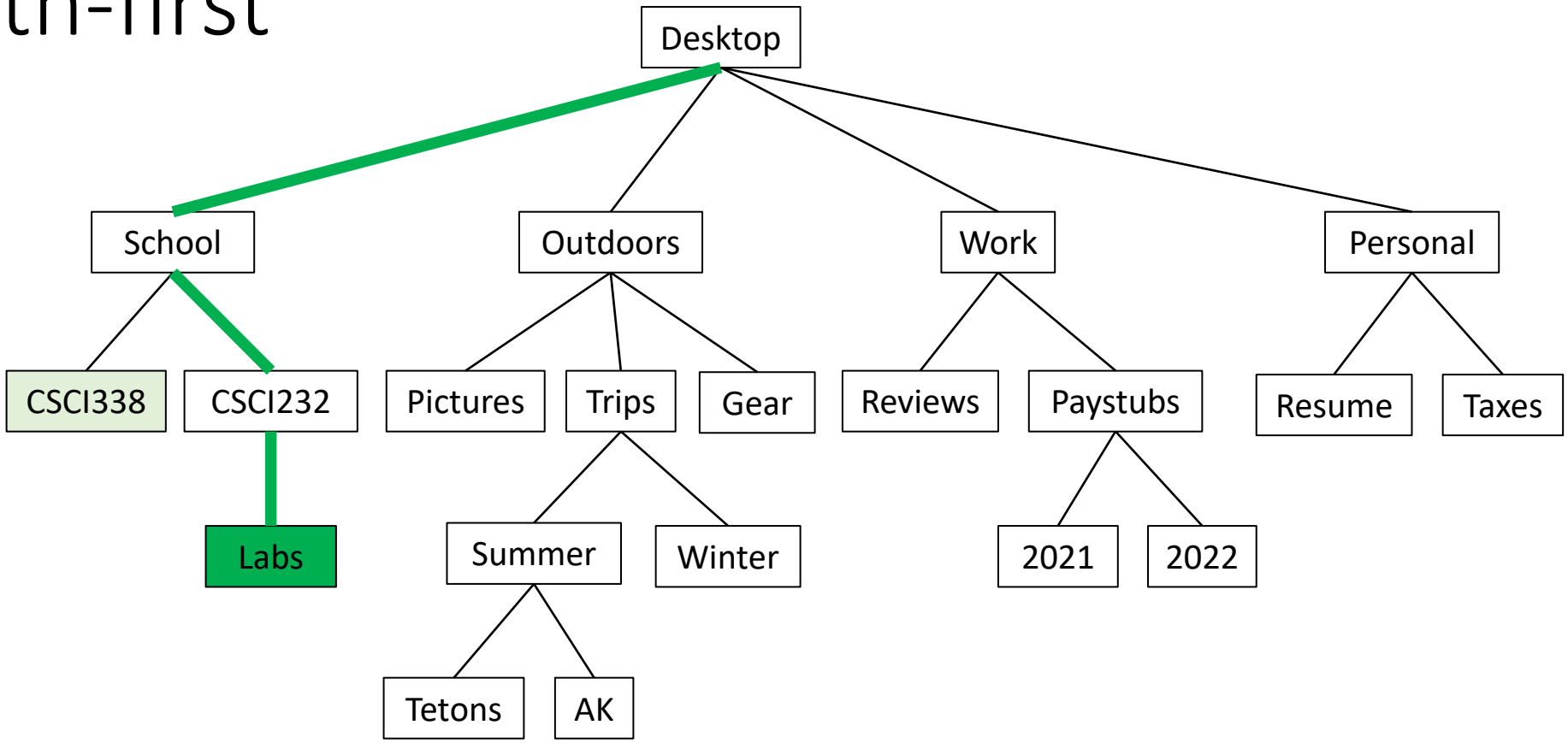
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.

Depth-first



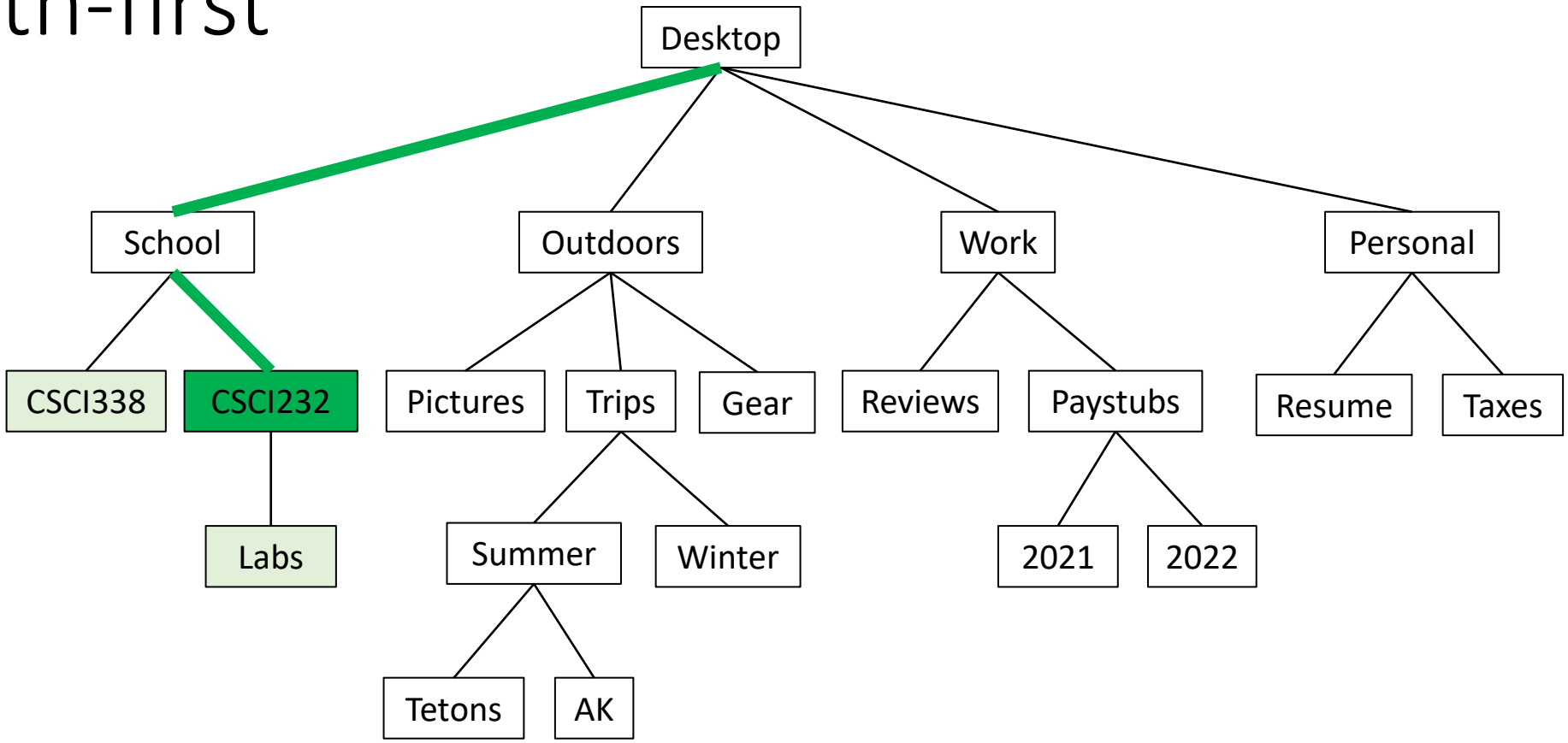
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



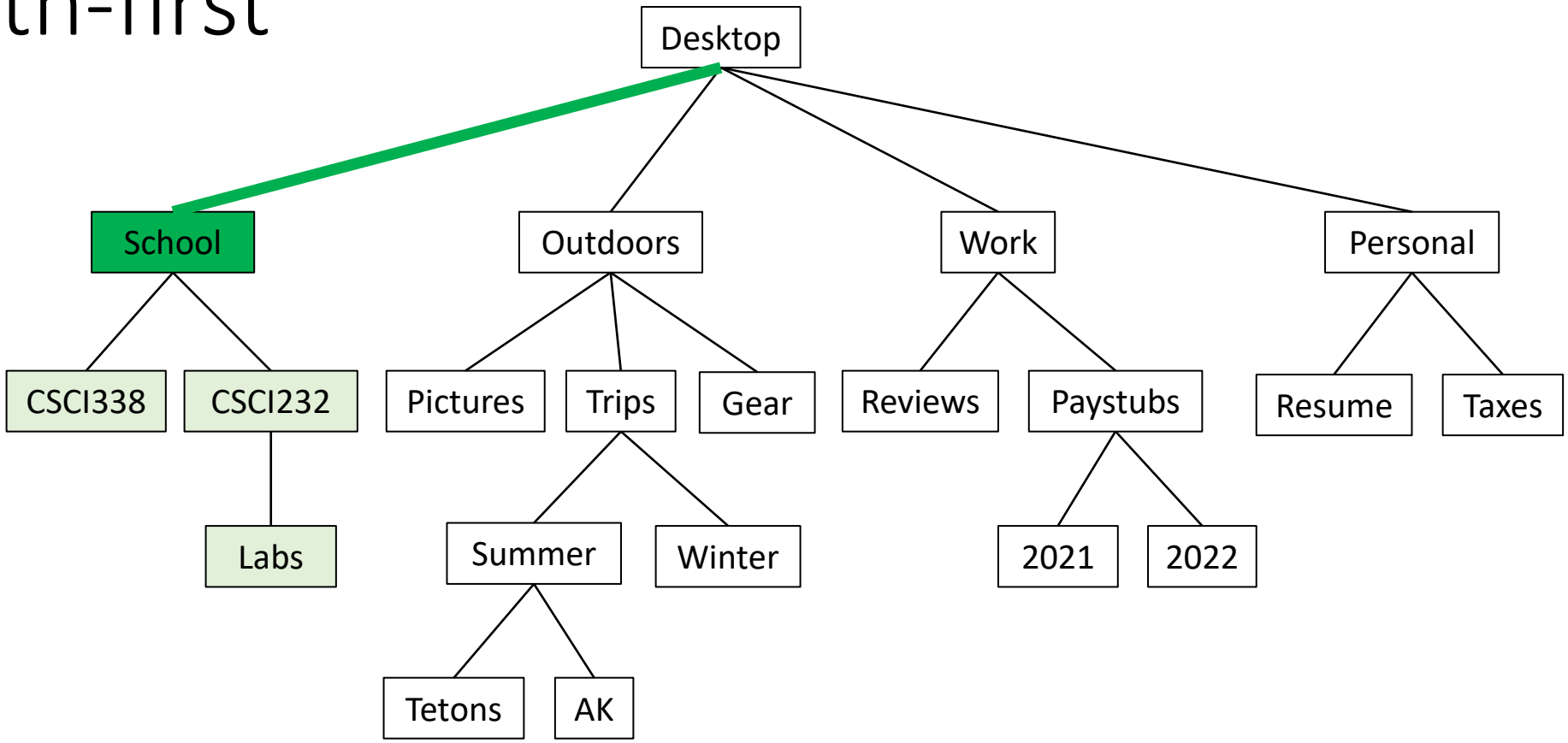
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



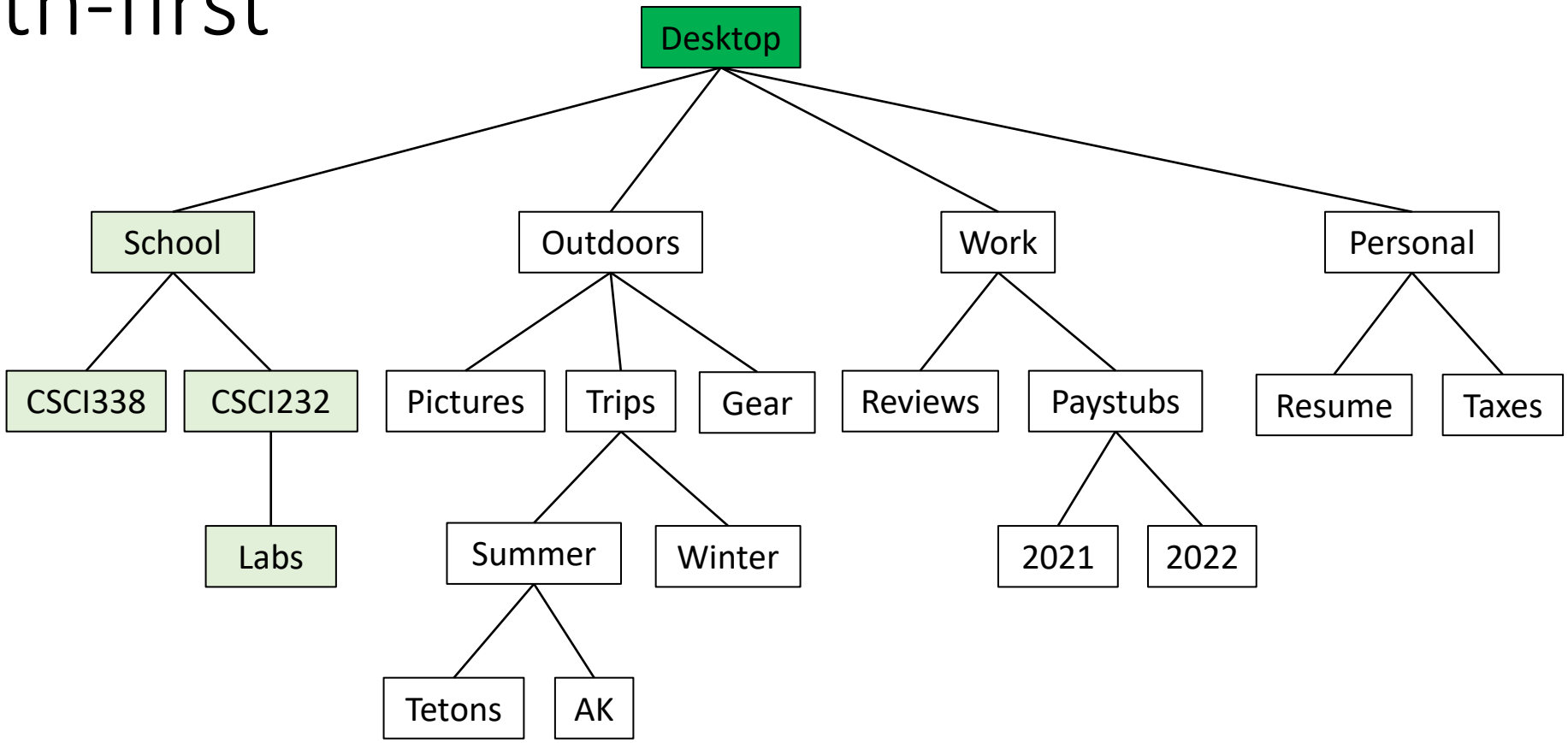
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



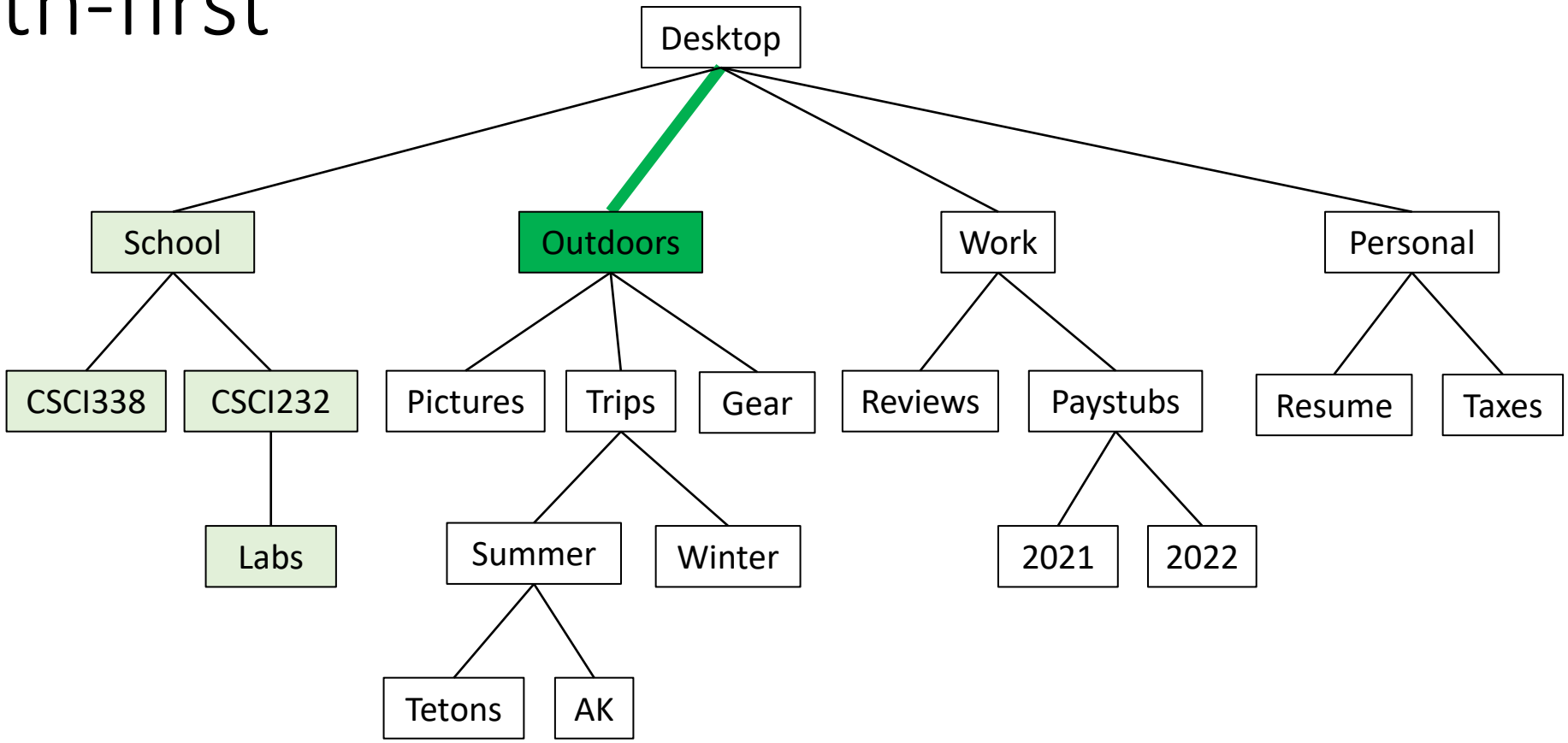
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



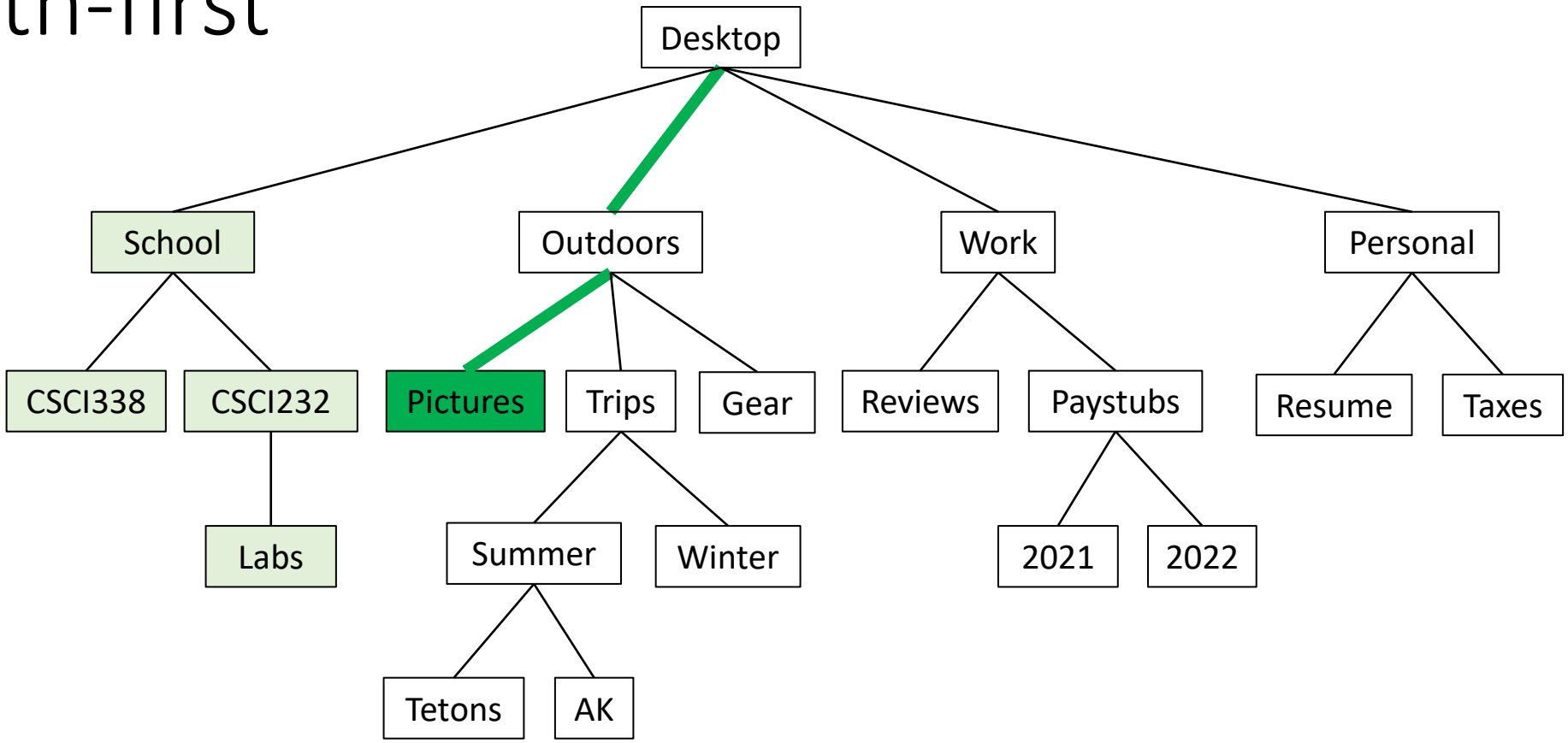
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



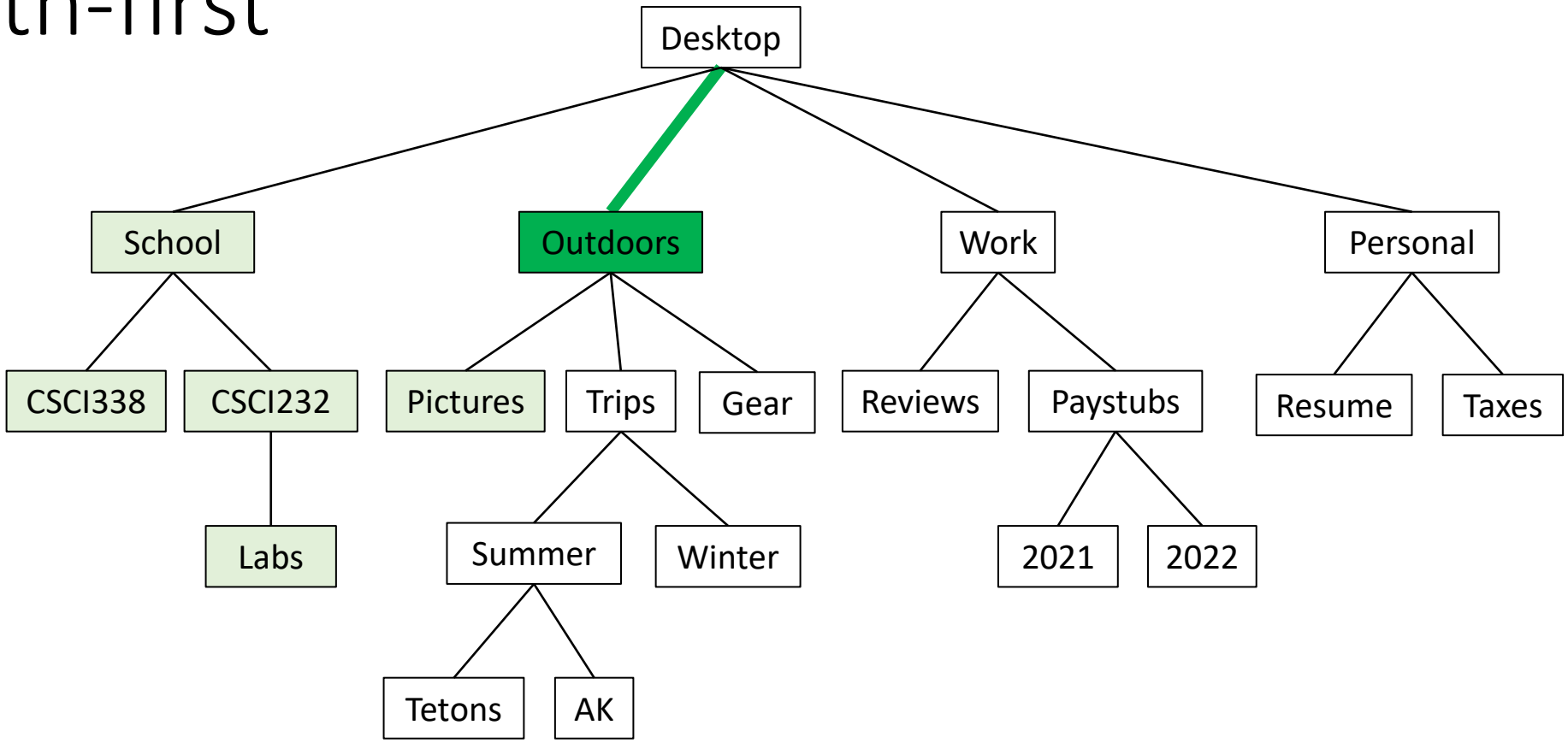
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



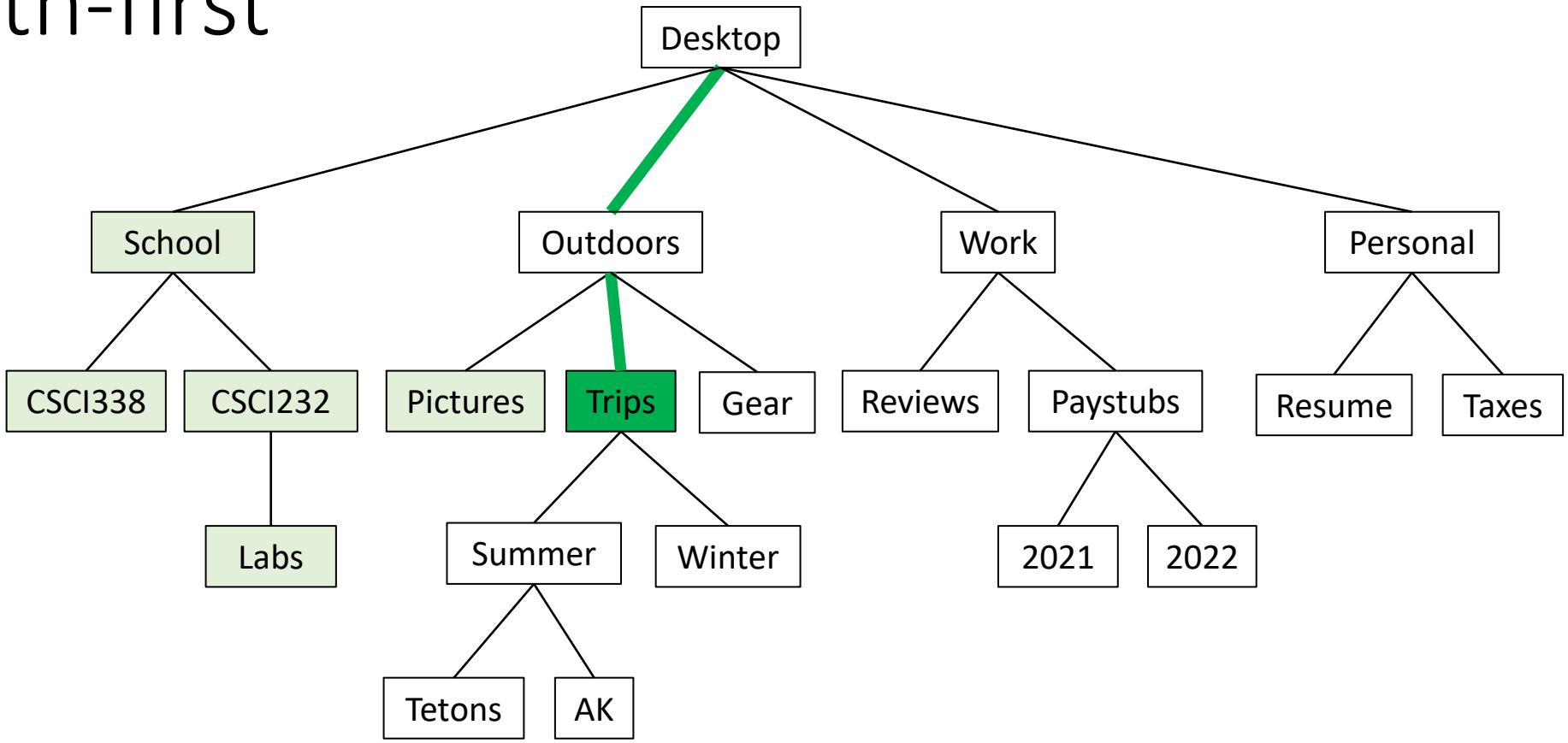
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



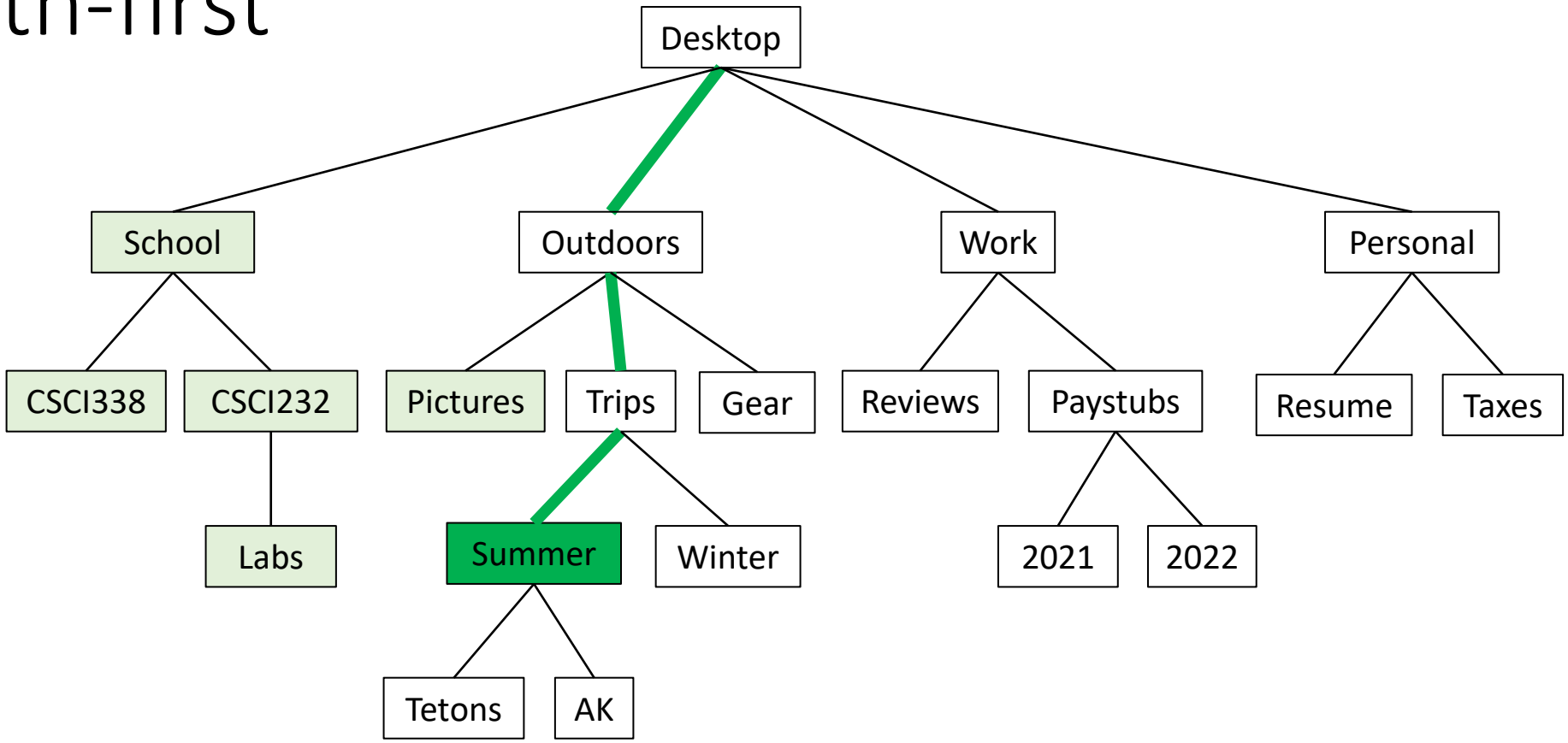
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



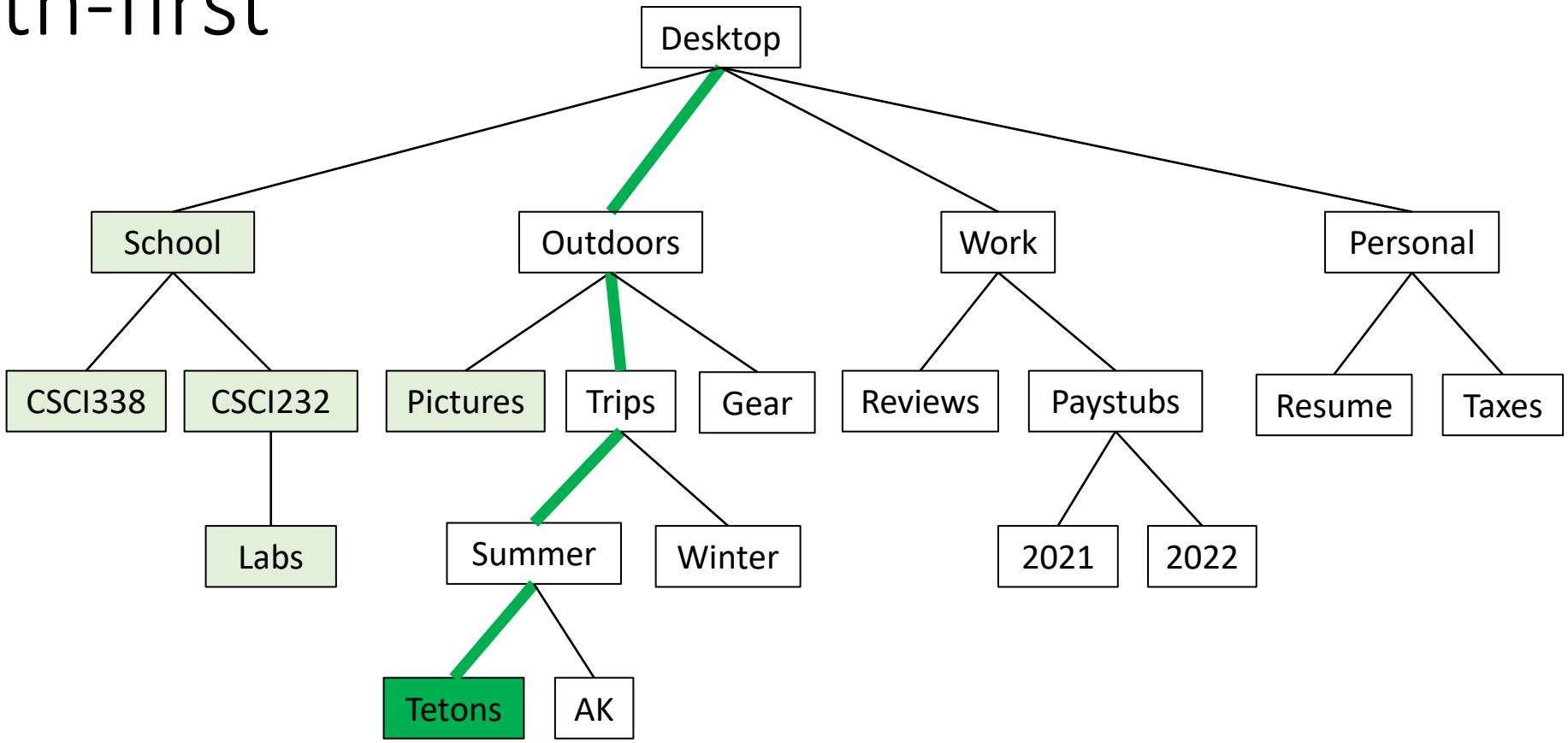
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



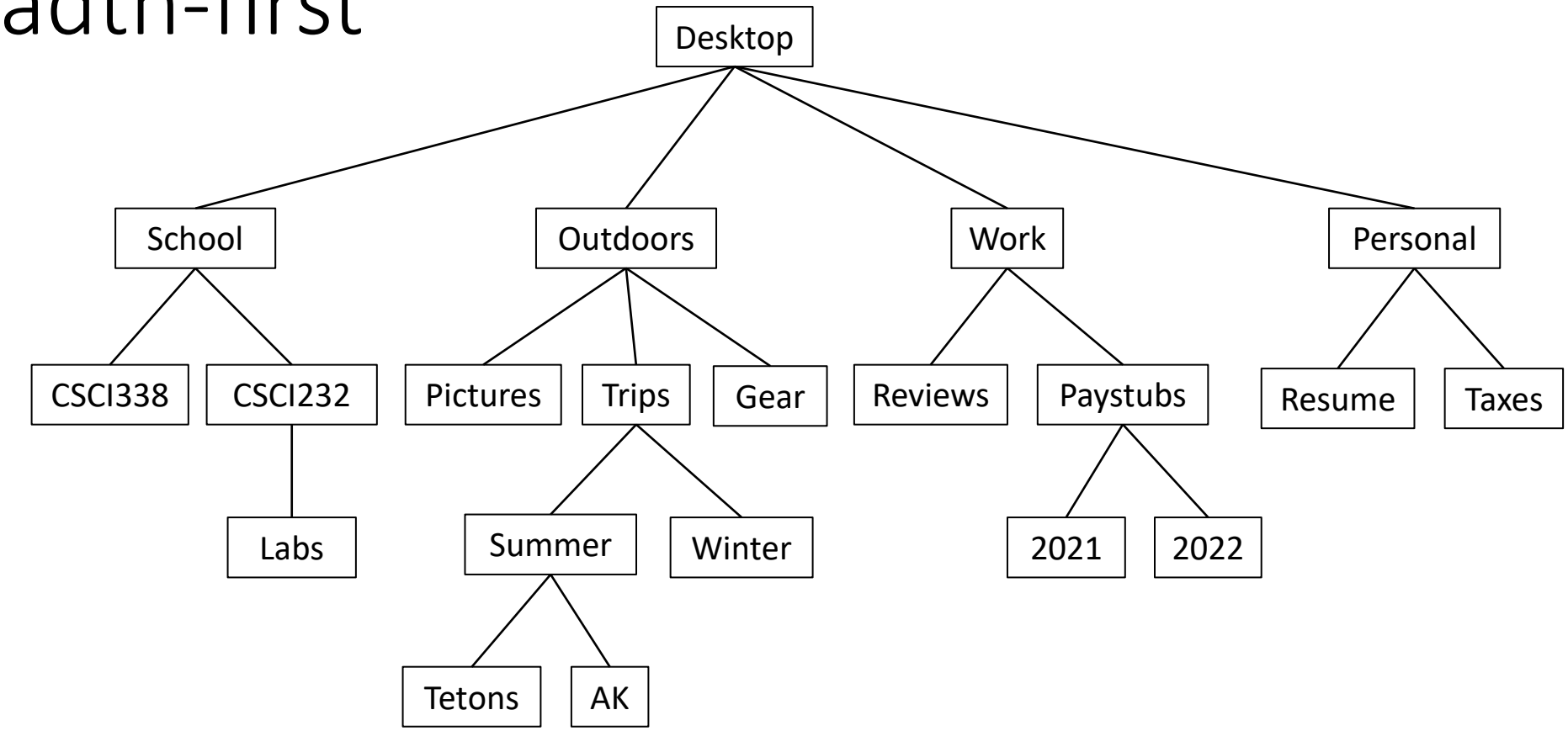
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Depth-first



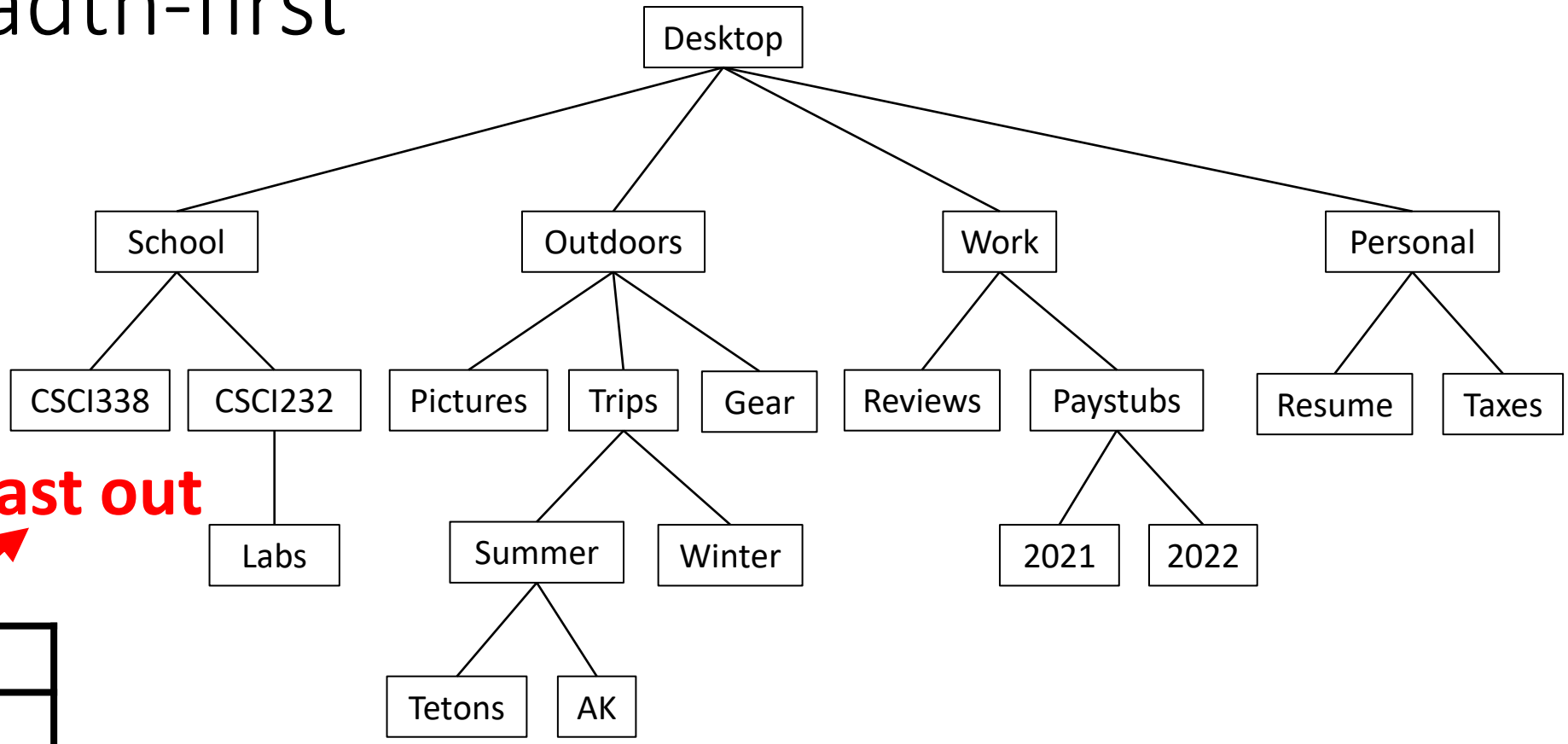
1. Go all the way down to the “first” leaf.
2. Backtrack until unvisited child is encountered.
3. Repeat.

Breadth-first

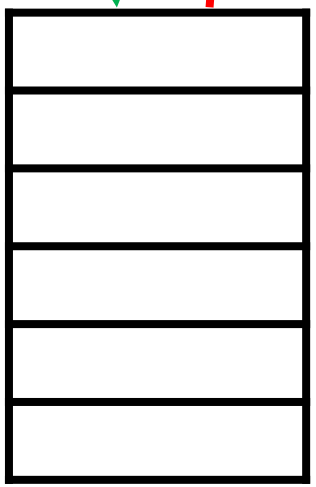


How to implement this?

Breadth-first



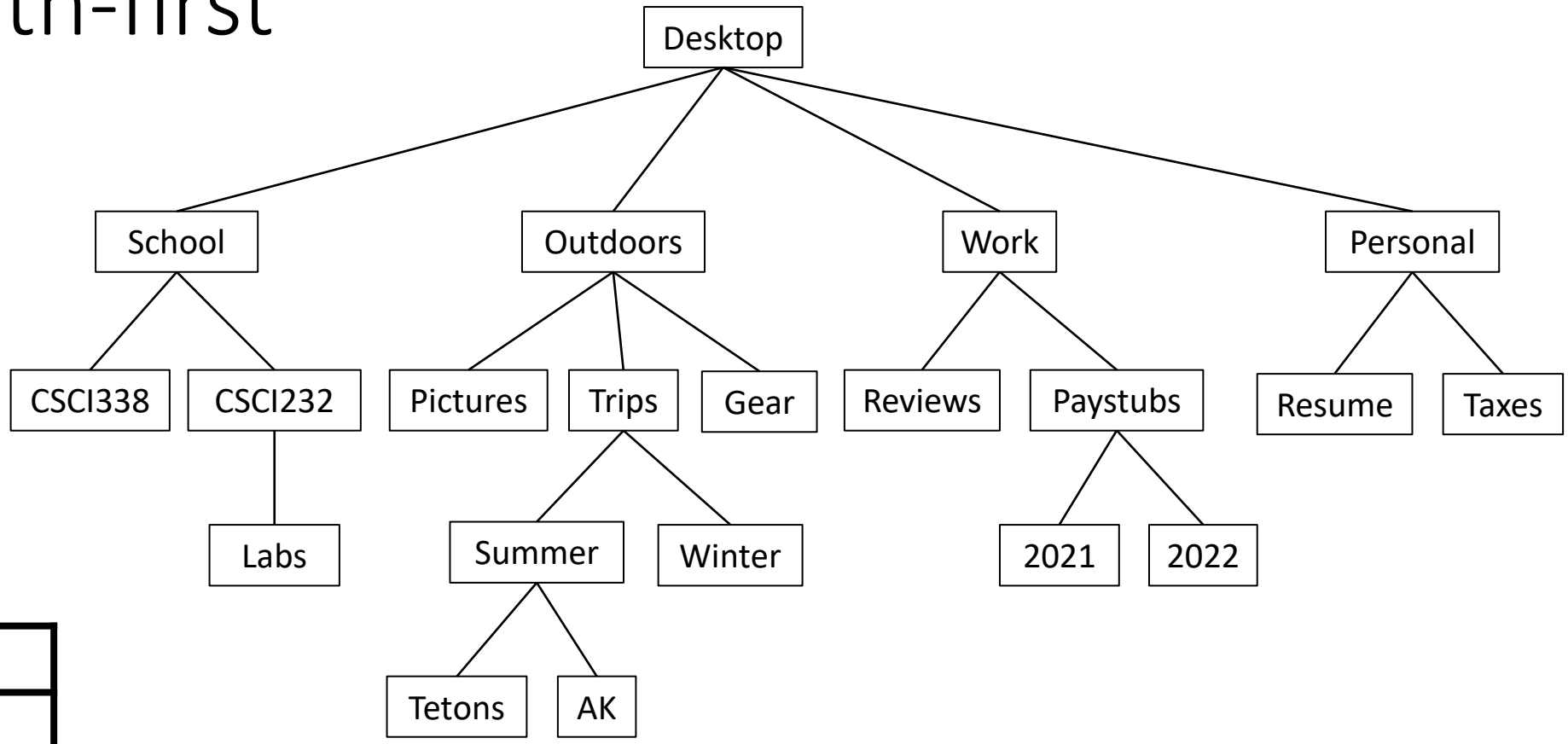
First in **Last out**

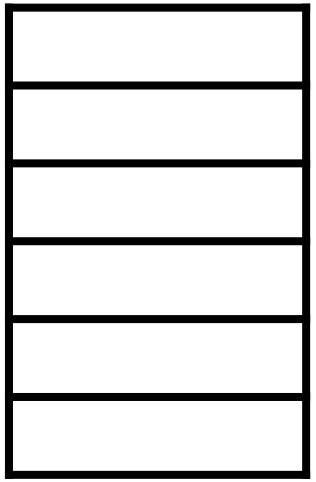


How to implement this?

Stack?

Depth-first

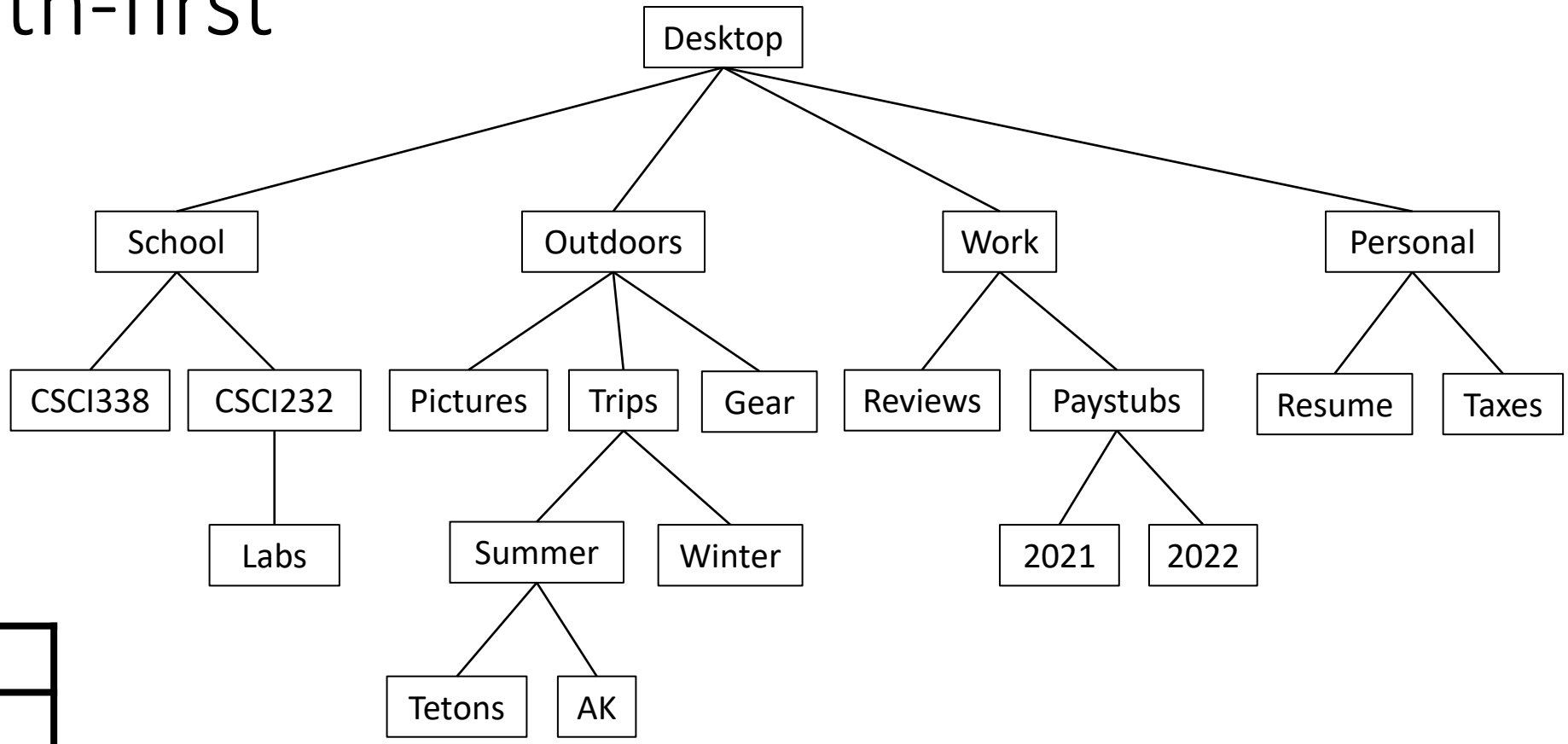


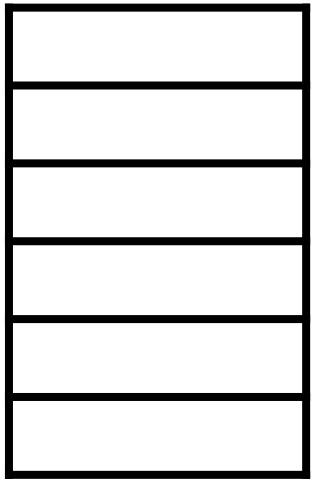


Every time we “visit” a node we:

1. Remove node from stack.

Depth-first

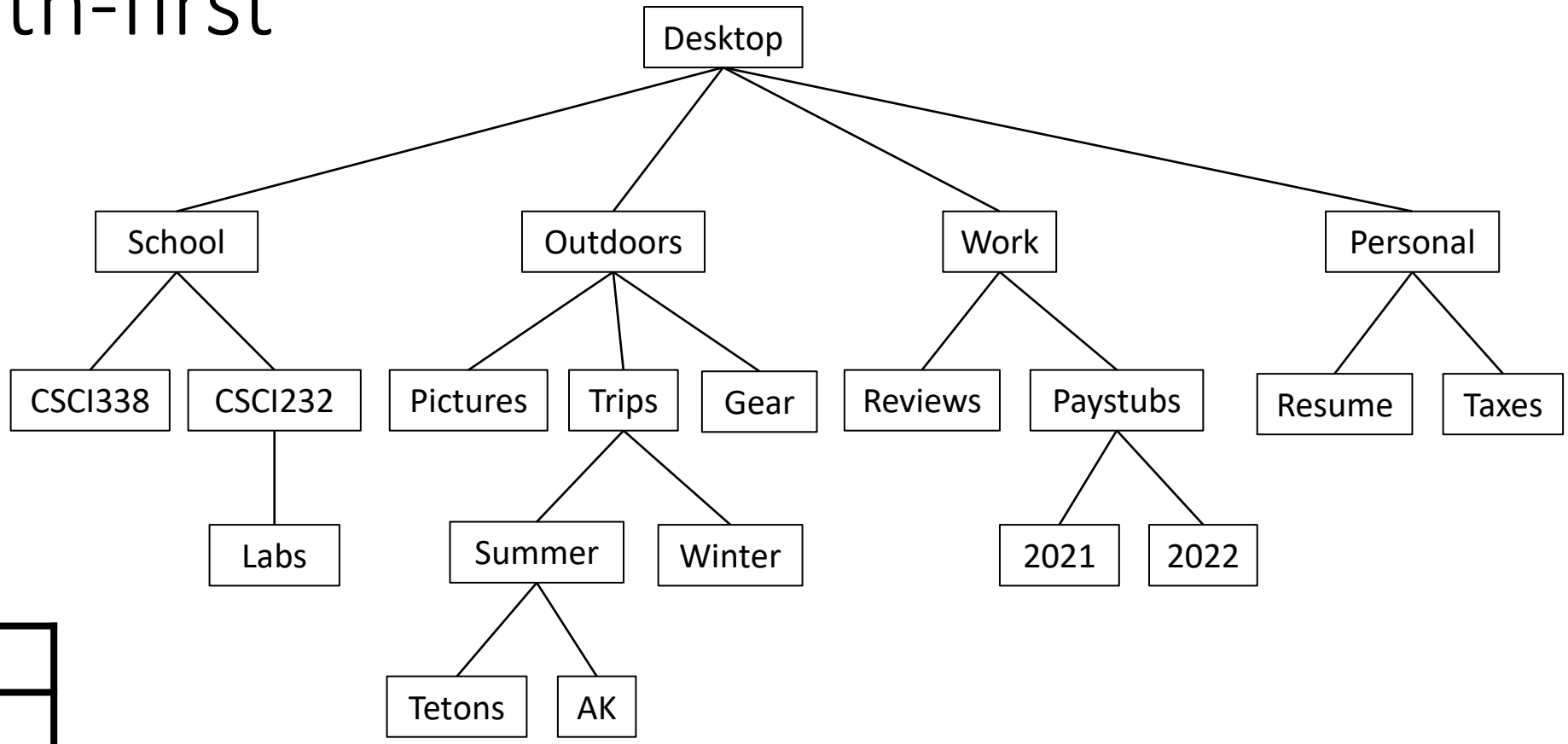


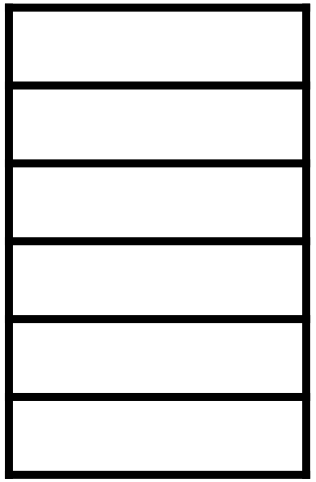


Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).

Depth-first

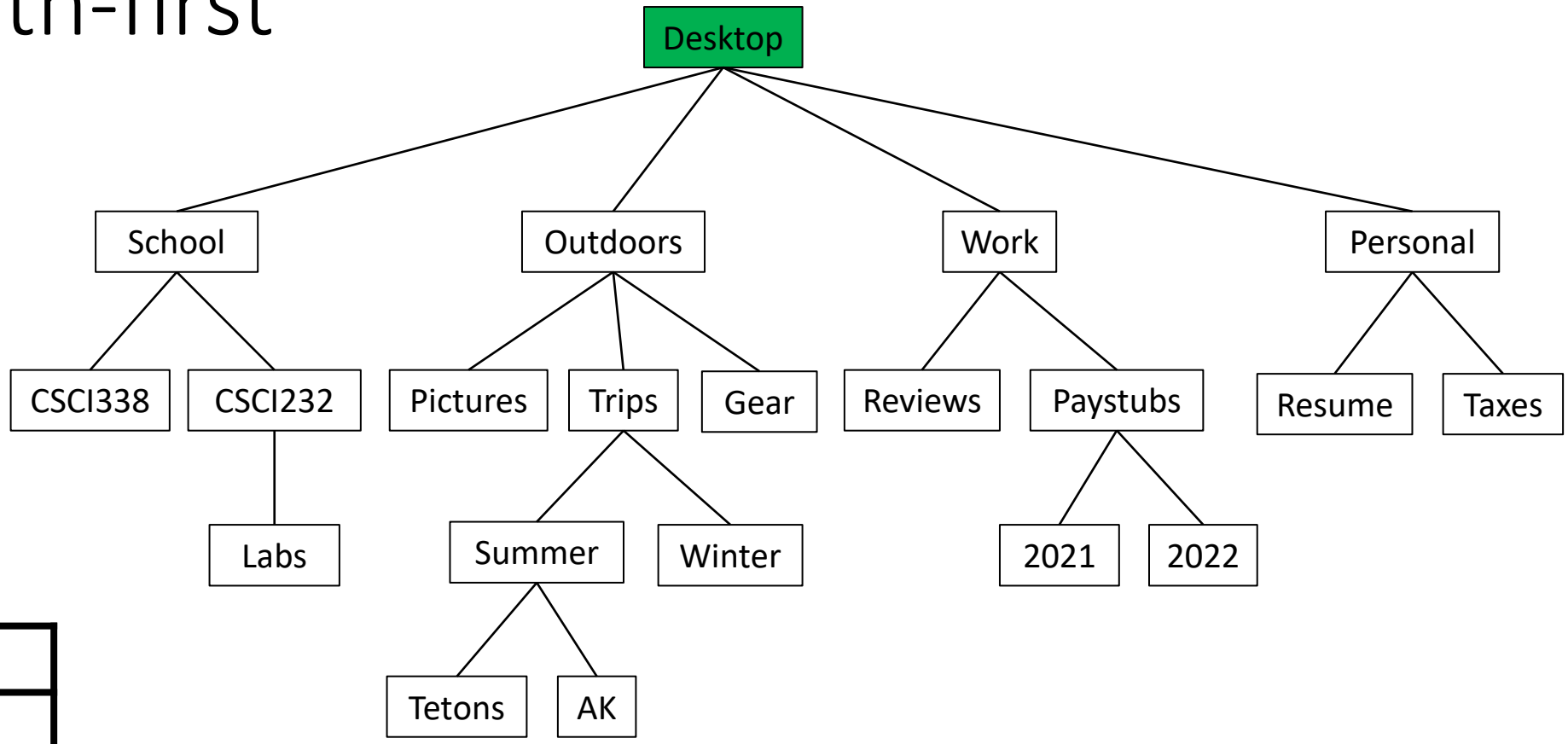


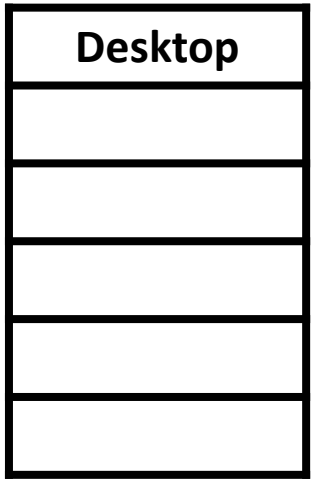


Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

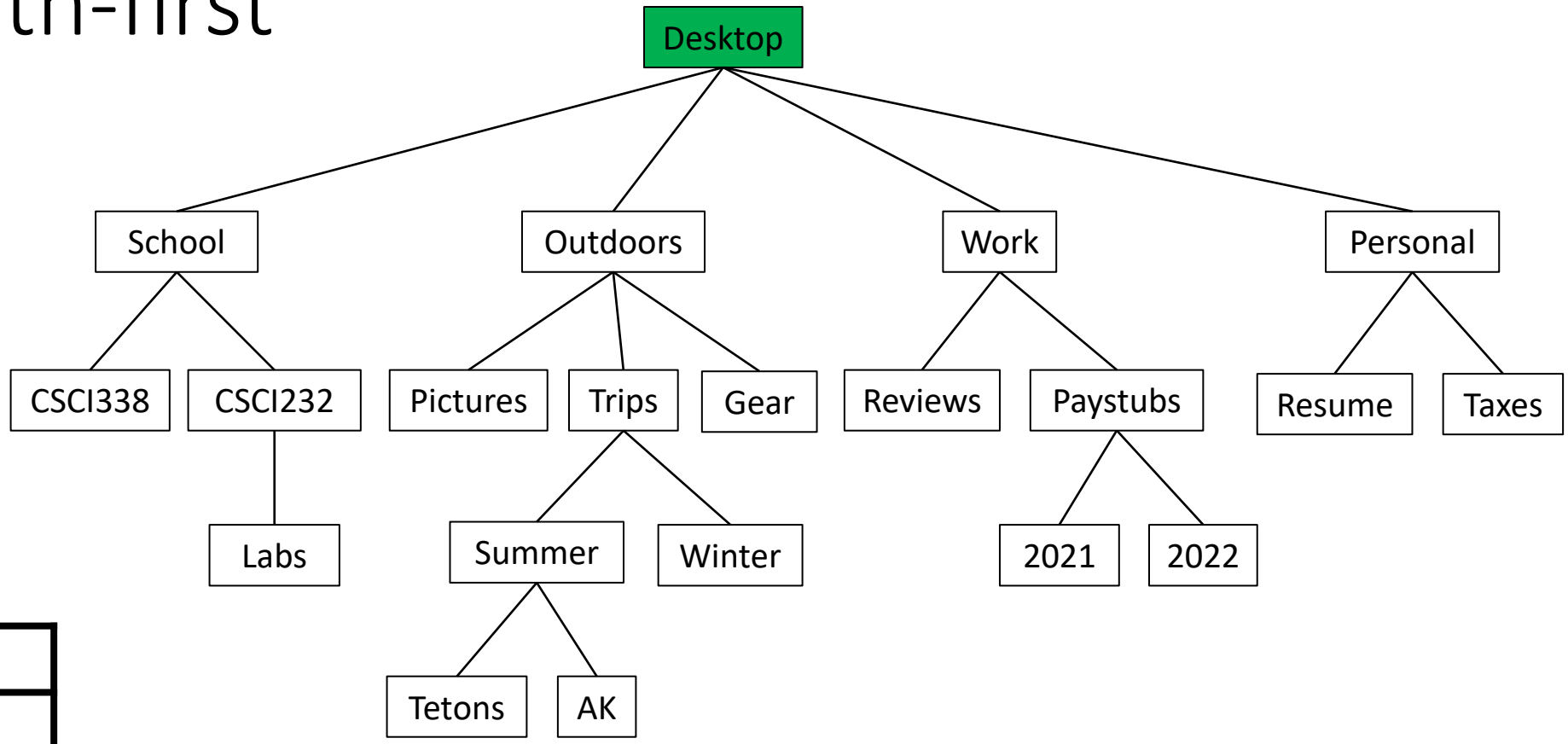


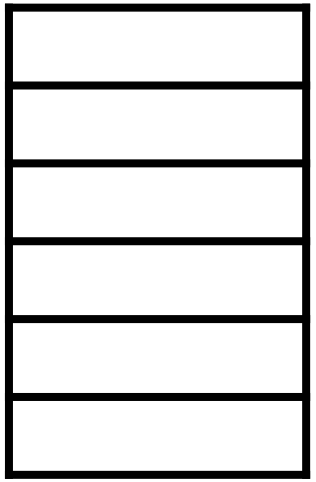


Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

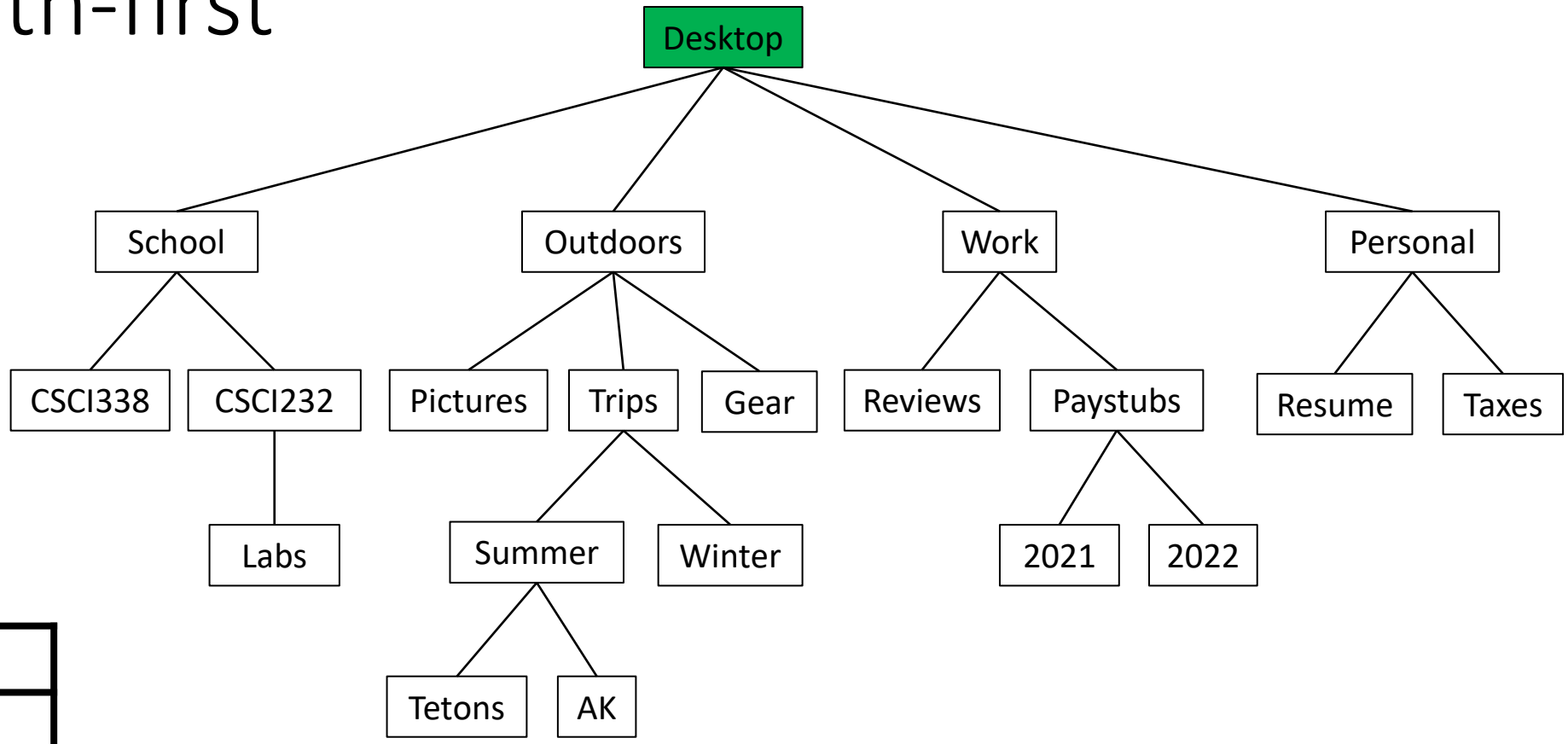


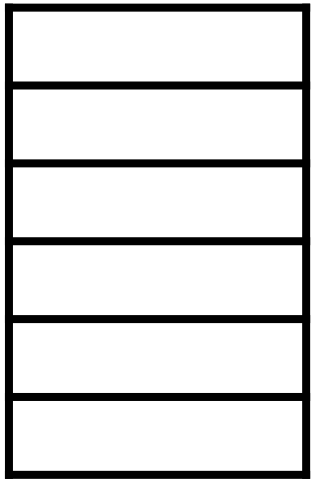


Every time we "visit" a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

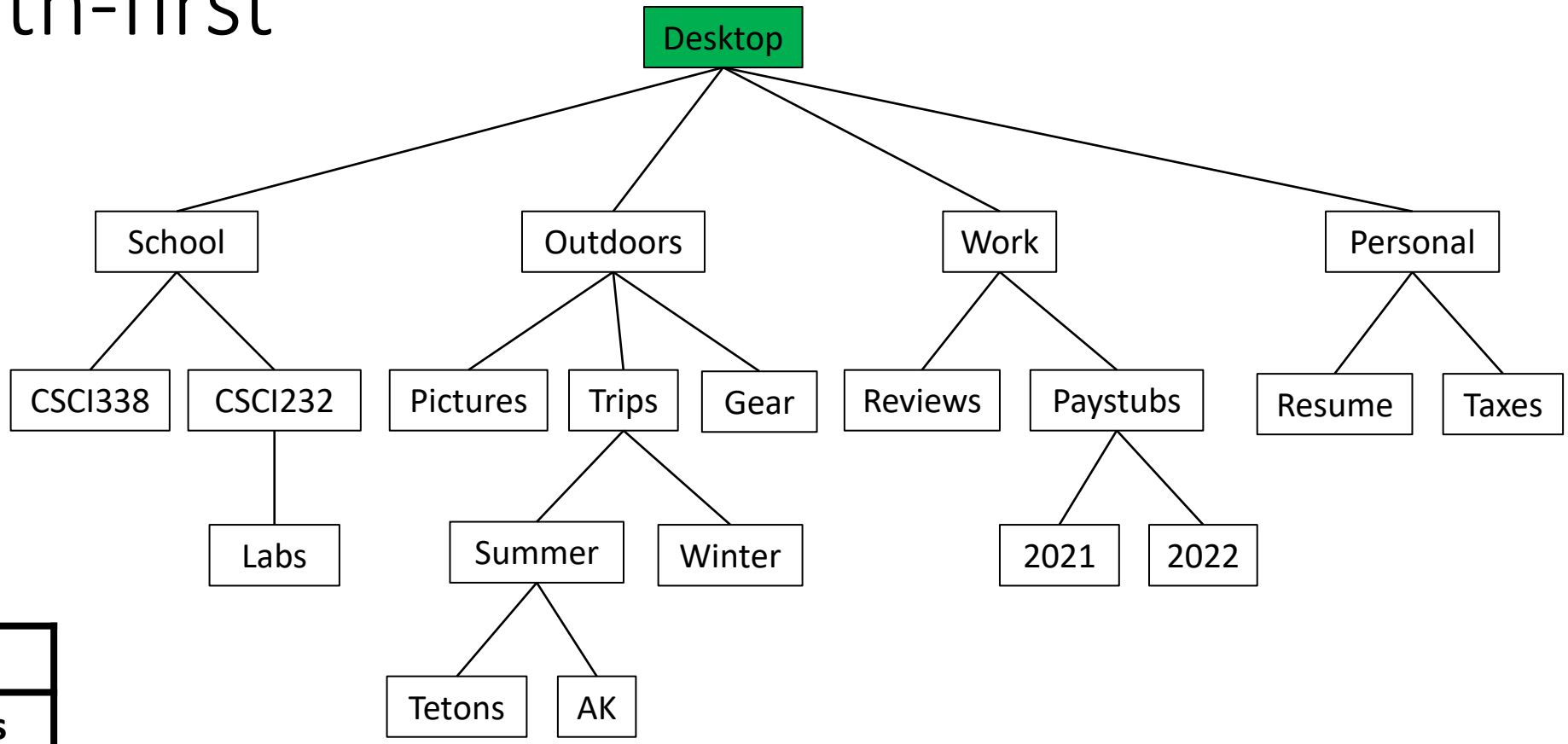




Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

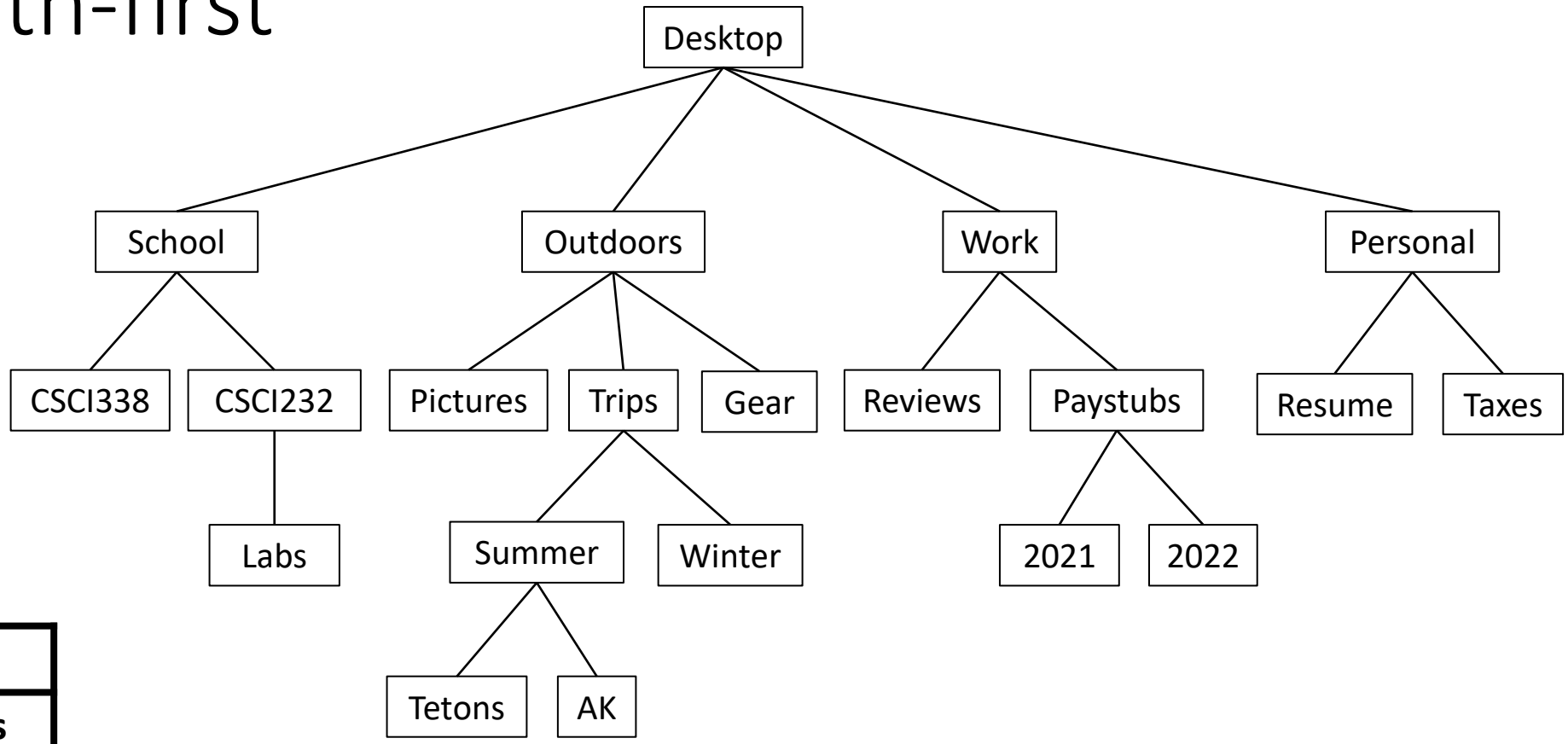


School
Outdoors
Work
Personal

Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

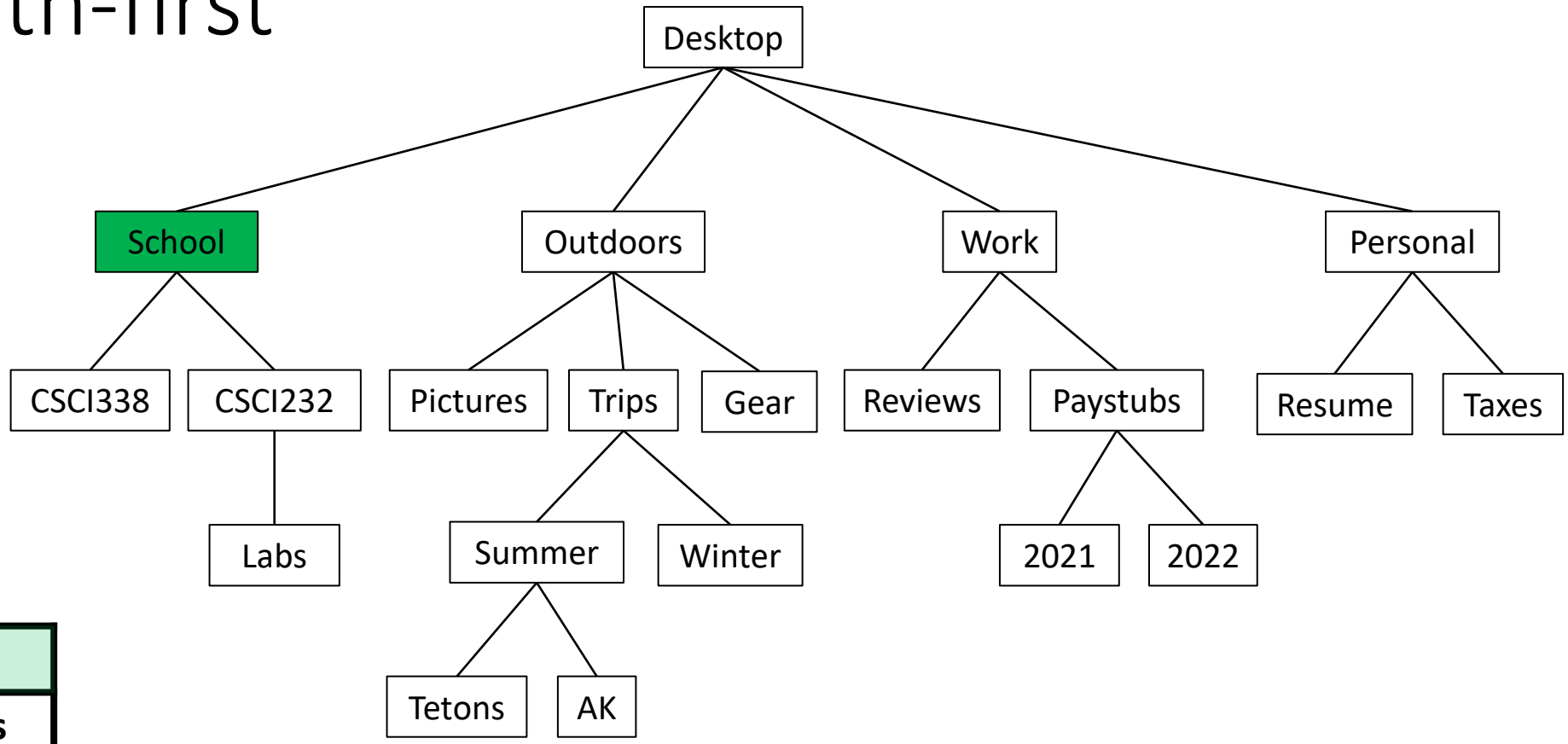


School
Outdoors
Work
Personal

Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

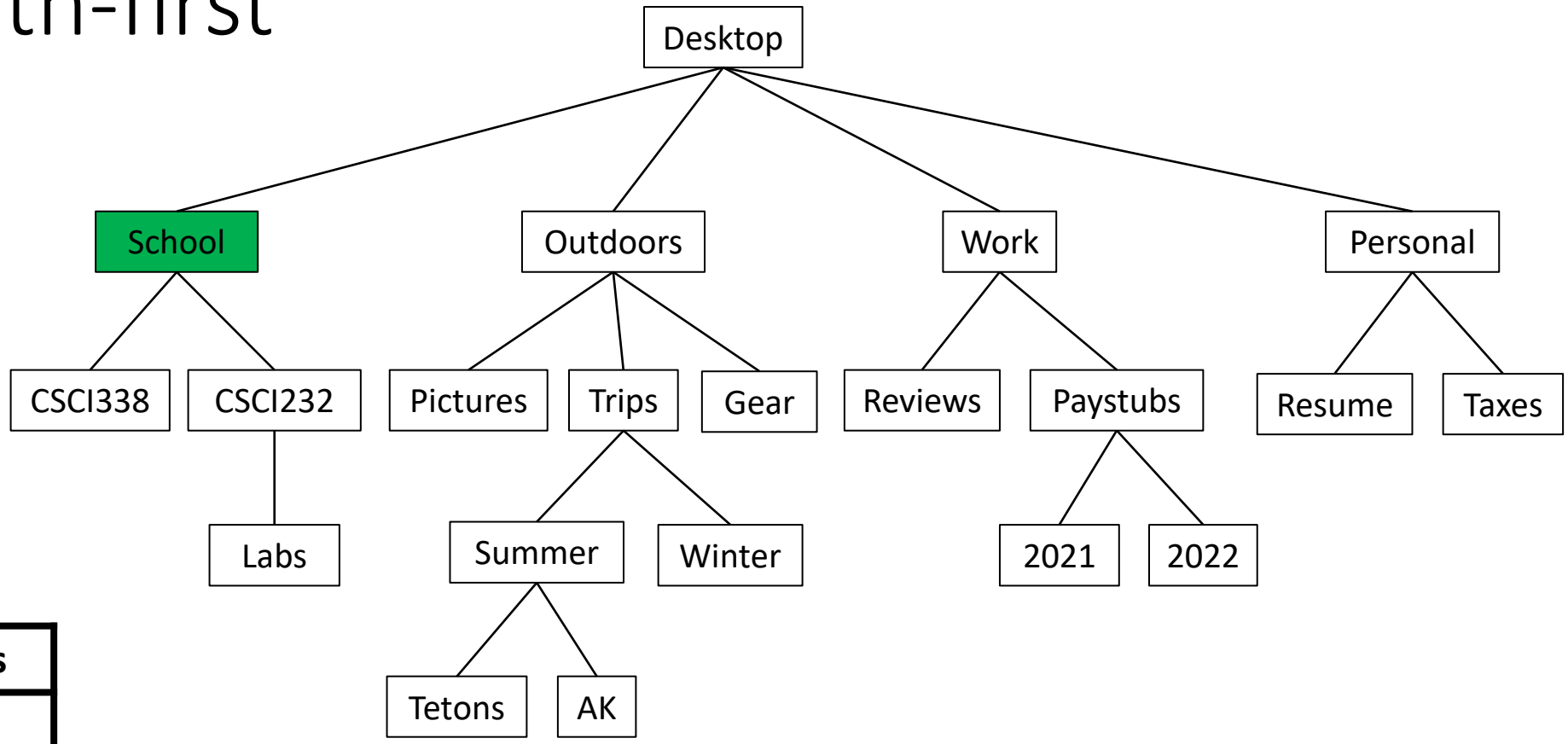


School
Outdoors
Work
Personal

Every time we "visit" a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

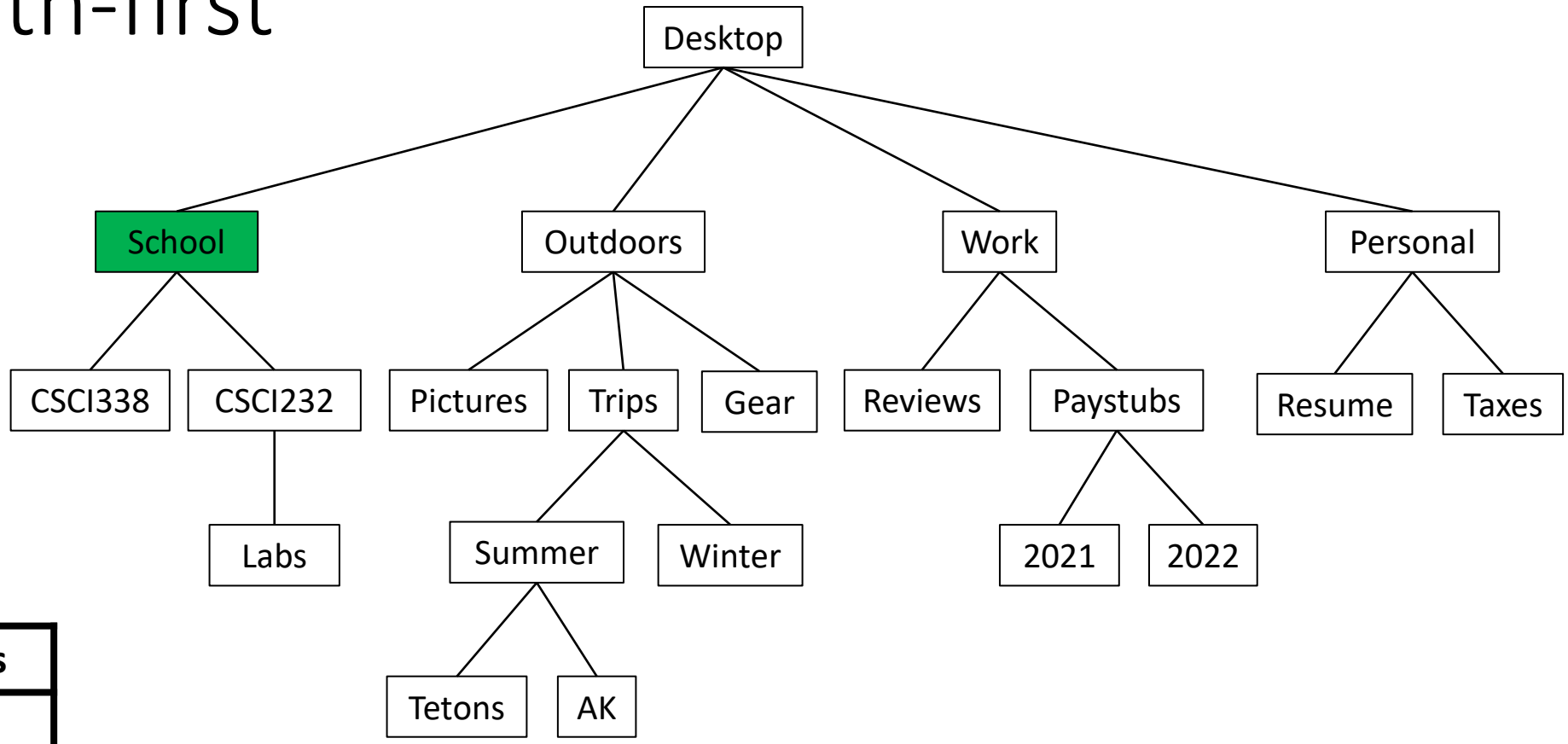


Outdoors
Work
Personal

Every time we “visit” a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

Depth-first

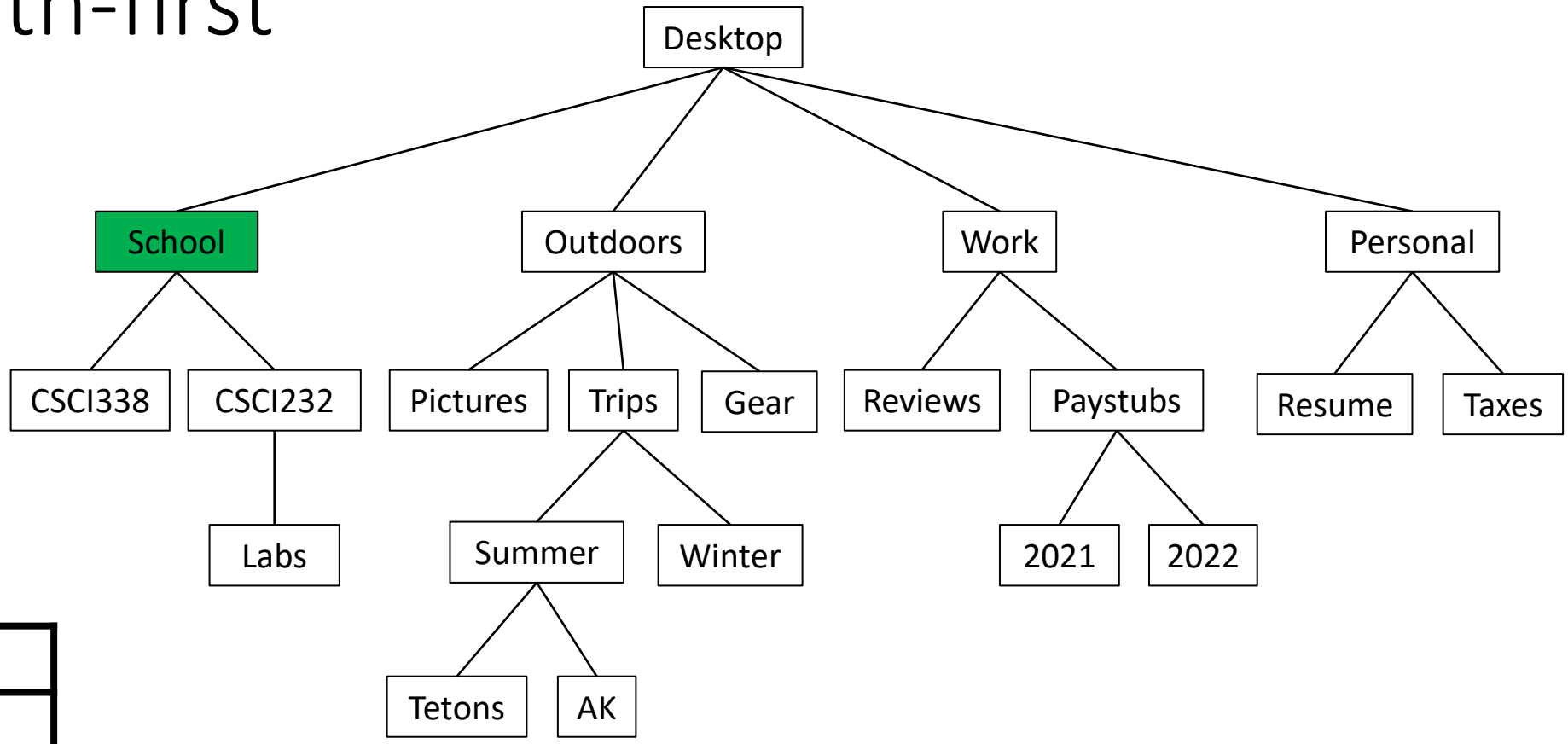


Outdoors
Work
Personal

Every time we "visit" a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.

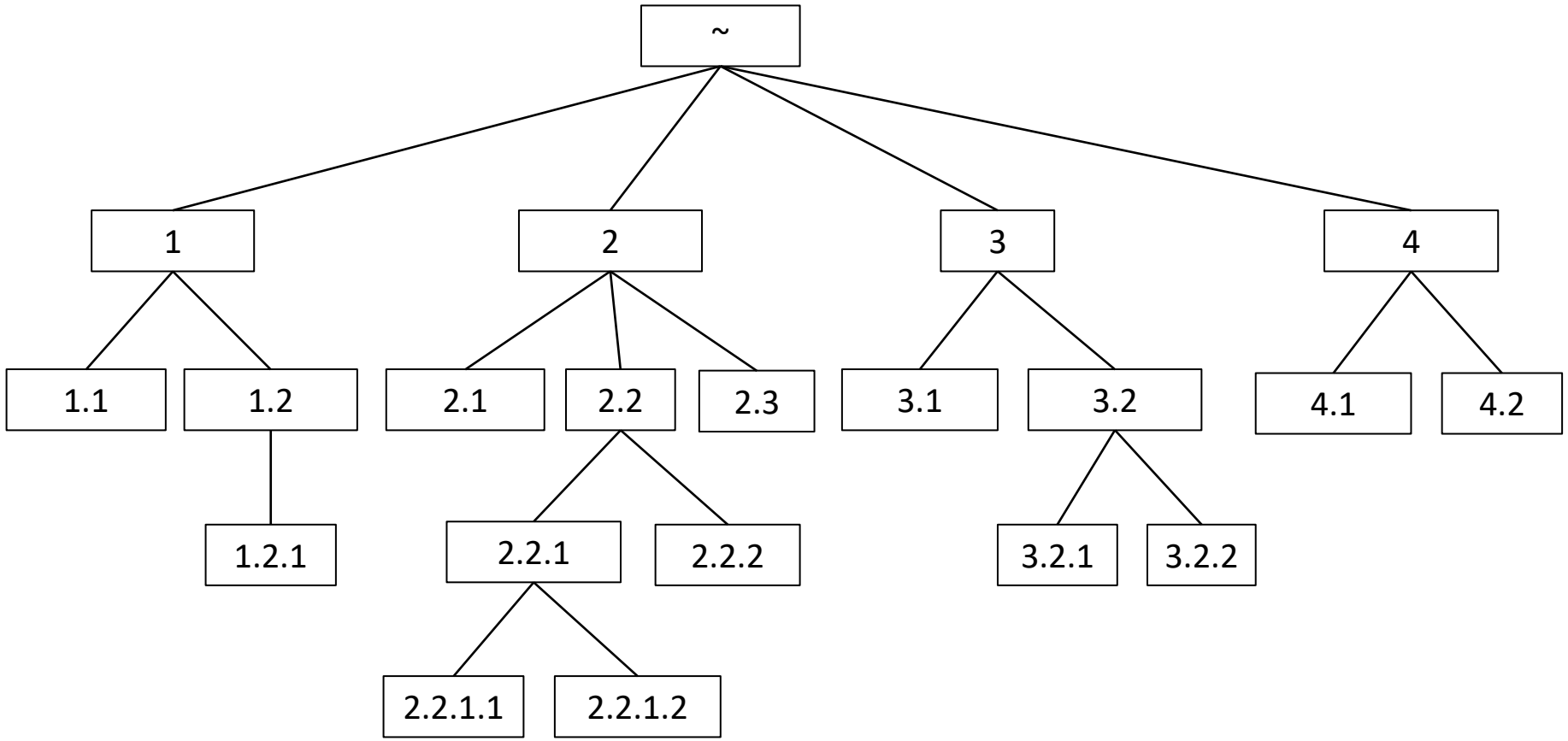
Depth-first



CSCI338
CSCI232
Outdoors
Work
Personal

Every time we "visit" a node we:

1. Remove node from stack.
2. Execute the action (e.g., print, compare,…).
3. Push all children to the stack.


~
1
2
3
4
1.1
1.2
2.1
2.2
2.3
3.1
3.2
4.1
4.2
1.2.1
2.2.1
2.2.2
3.2.1
3.2.2
2.2.1.1
2.2.1.2