CSCI 232:
Data Structures and Algorithms

Trees (Part 2), Tree Traversal

Reese Pearsall

Summer 2025

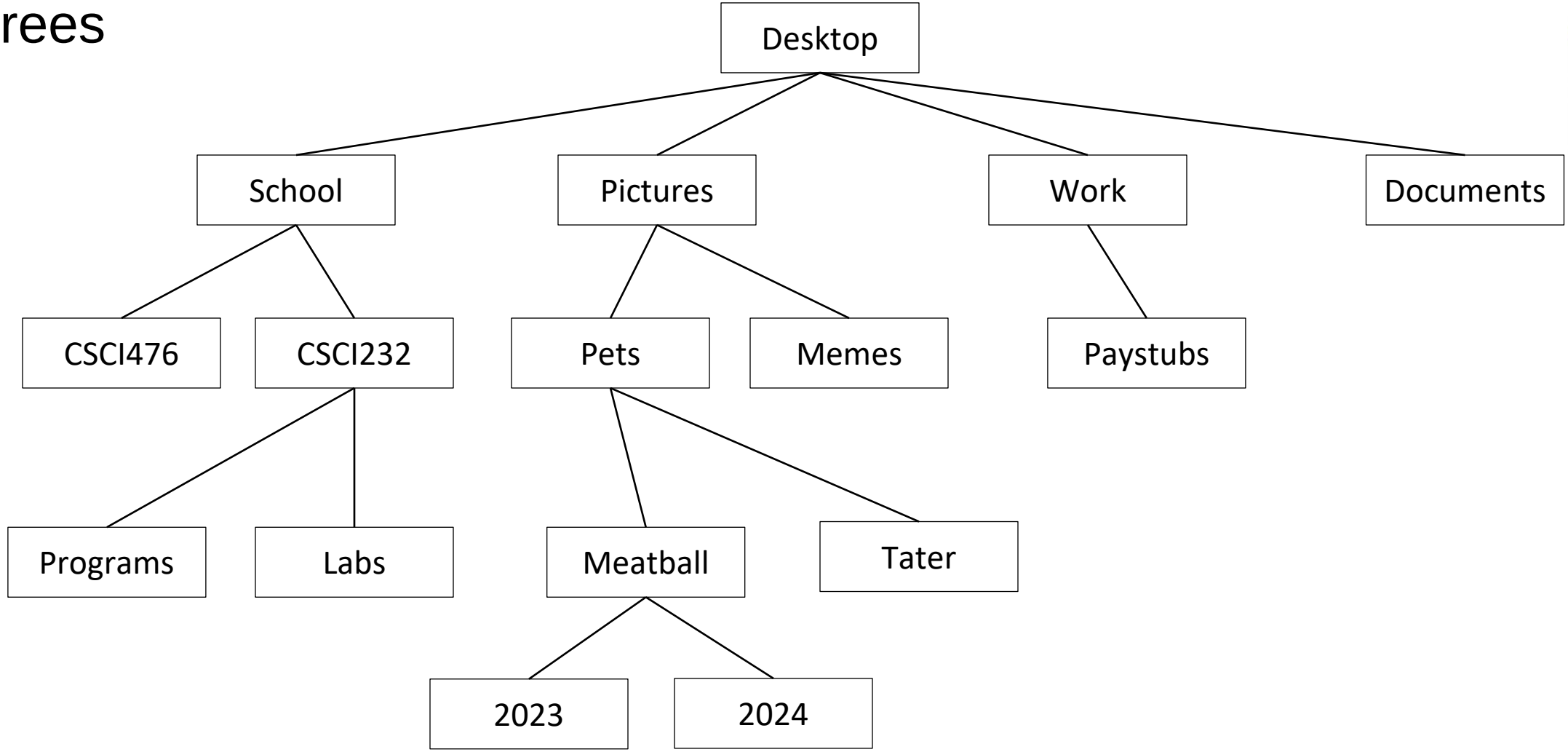
Announcements

Lab 1 due **tonight** at 11:59 PM

Lab 2 posted, due Tuesday at 11:59 PM

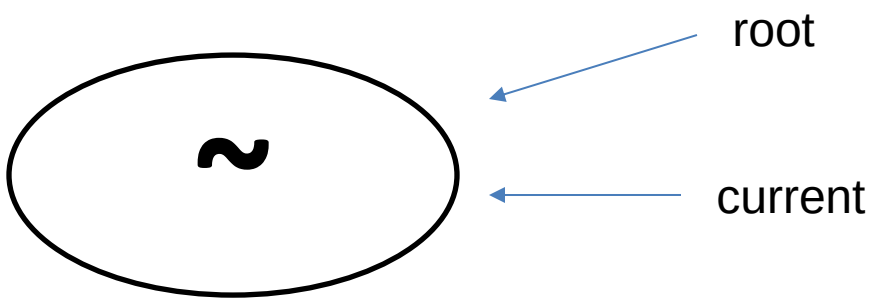
No class on Monday (Memorial Day)

Trees

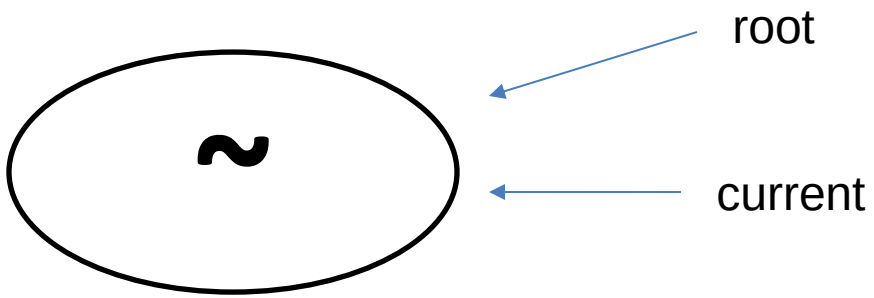


Trees are data structures used to store elements hierarchically
(not linear like arrays and linked lists)

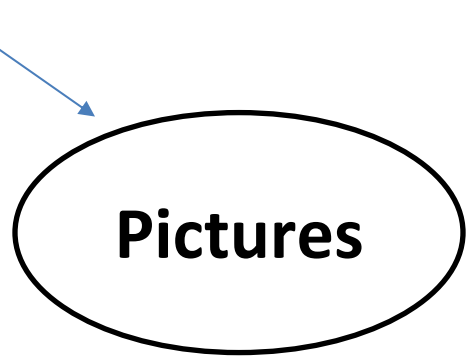
```
public FileTree() {
    this.root = new Node("~");
    this.current = root;
}
```



```
public FileTree() {
    this.root = new Node("~");
    this.current = root;
}
```

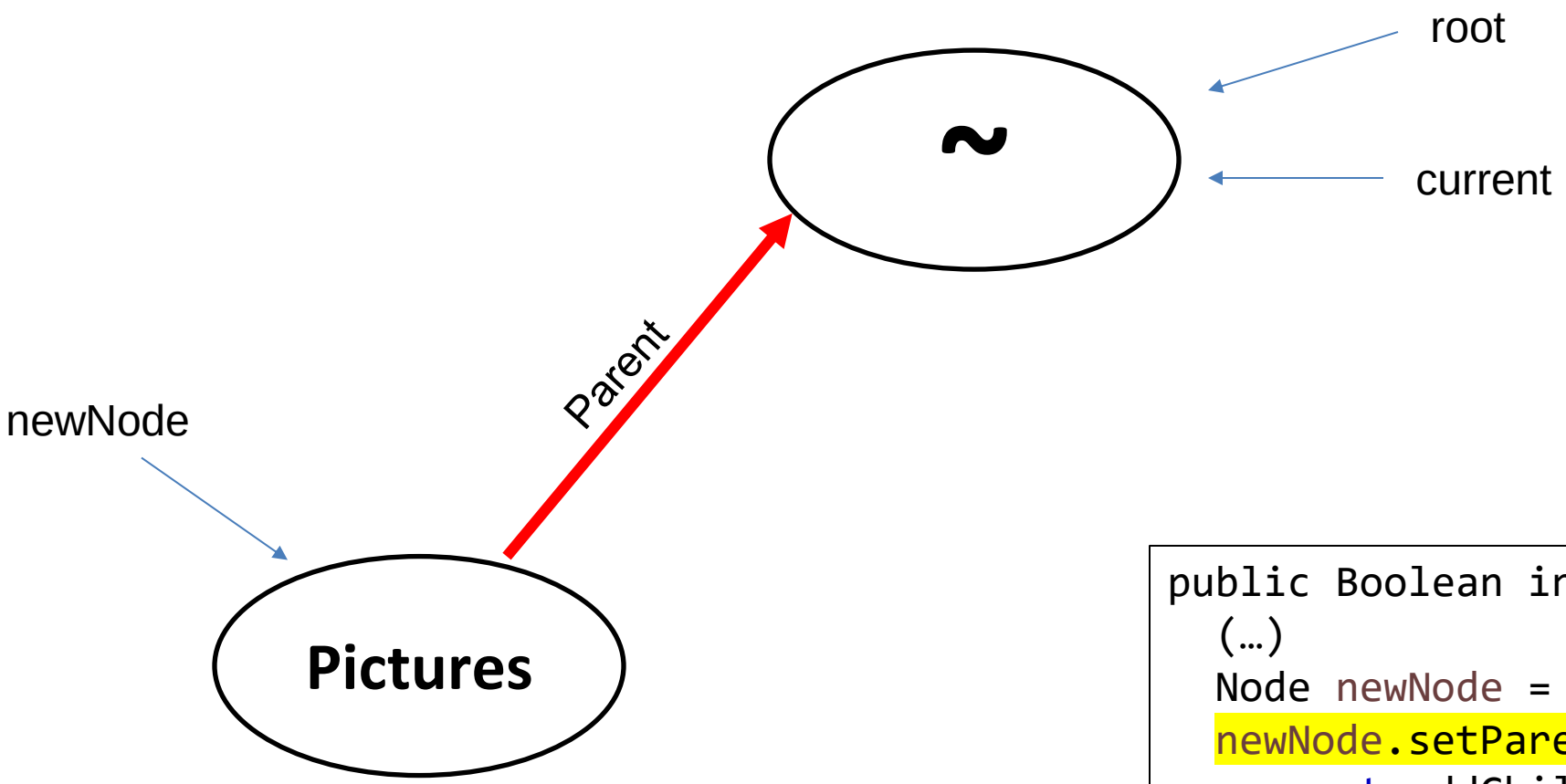


newNode



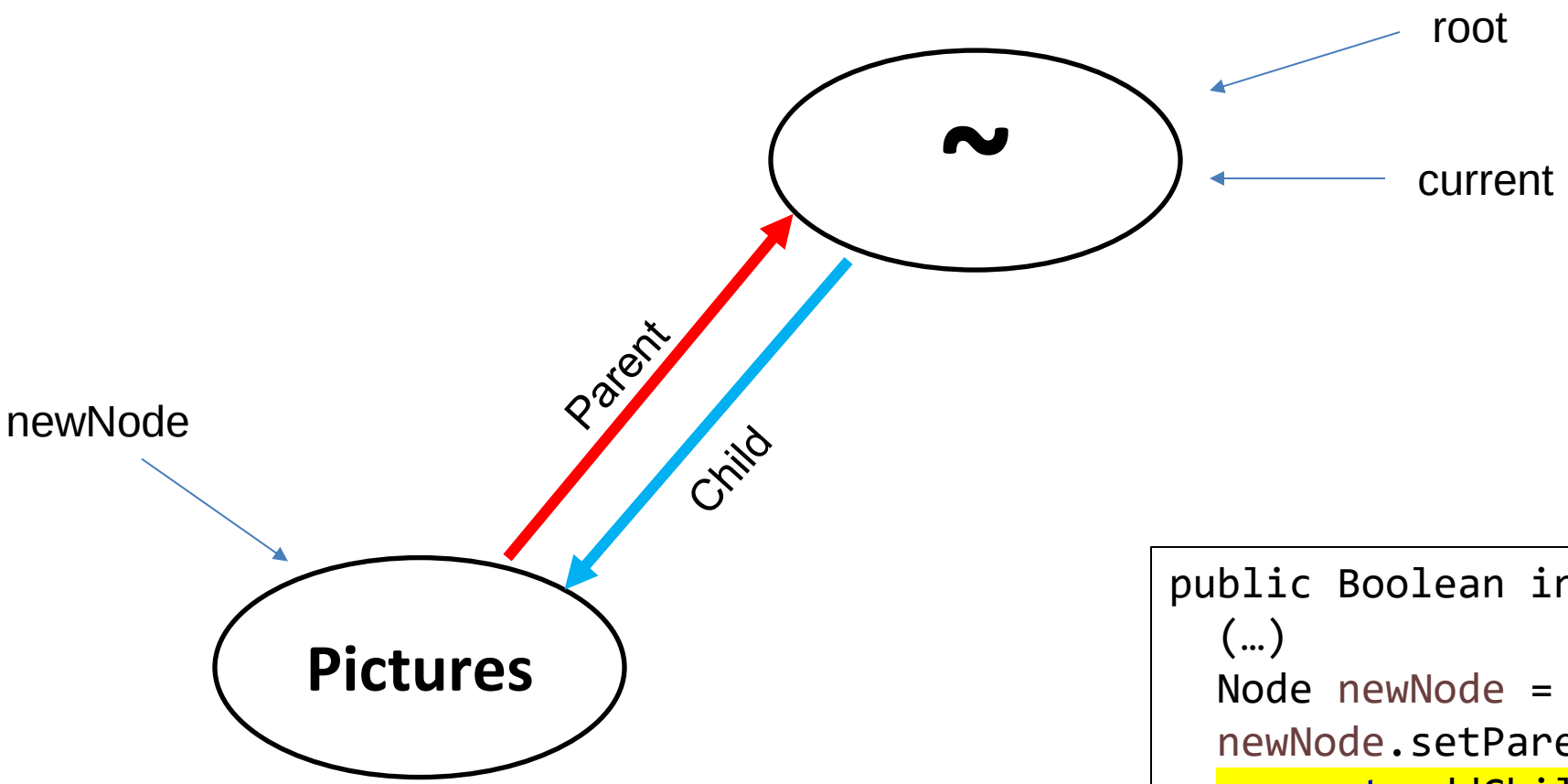
New node that needs to be added?

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



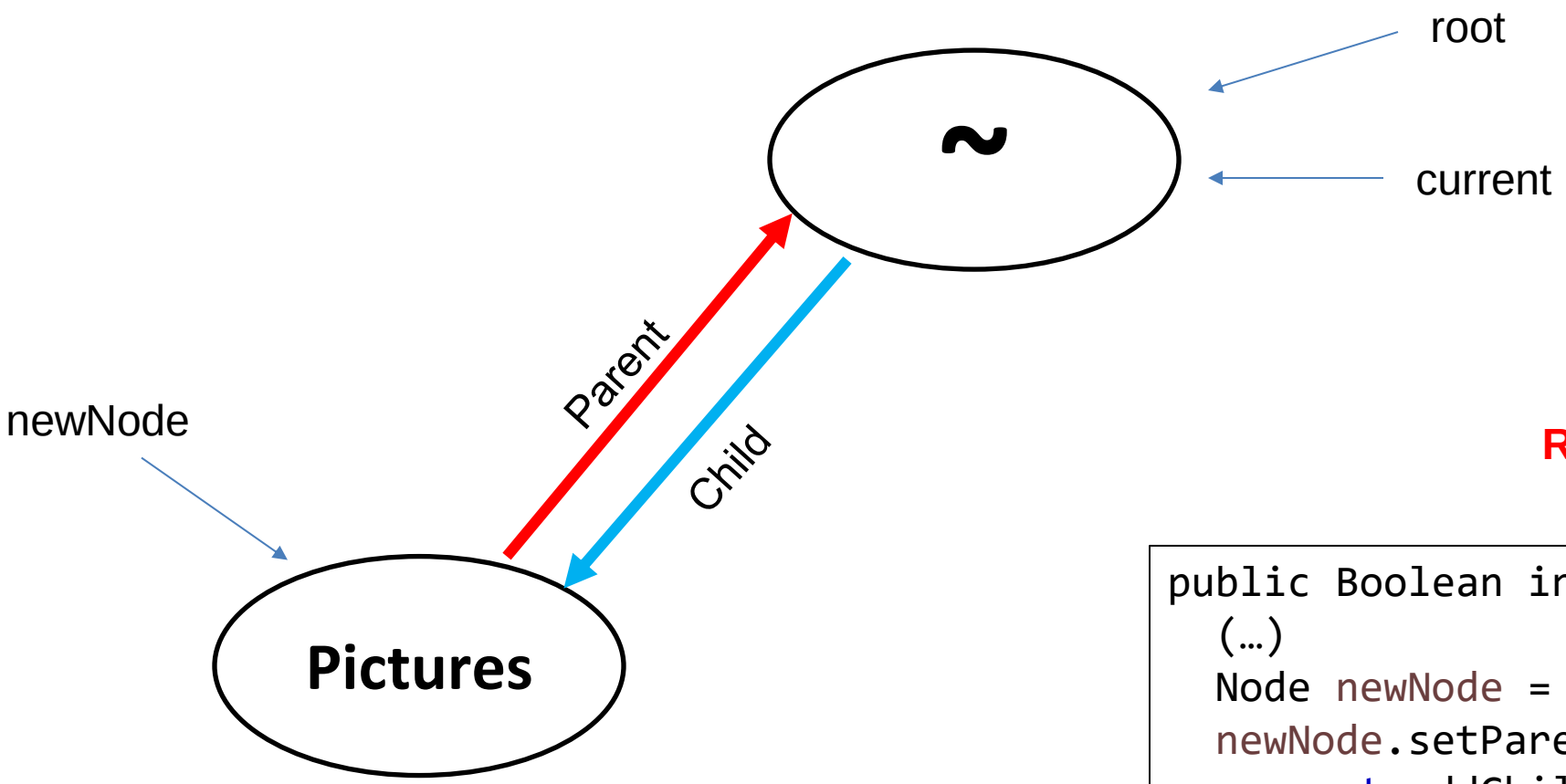

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

New node that needs to be added?




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

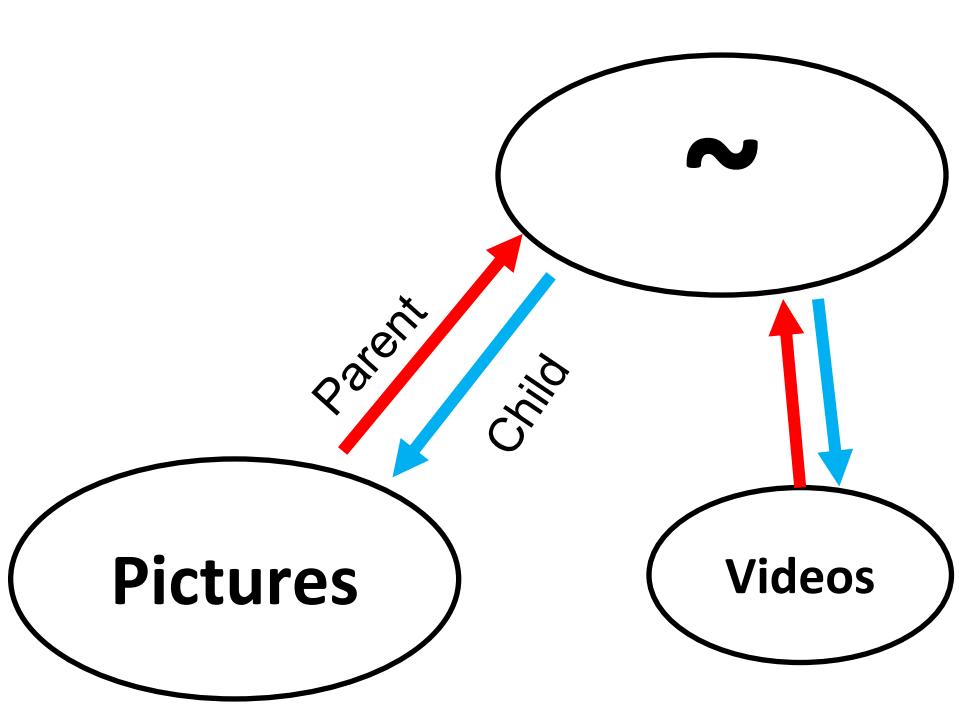
New node that needs to be added?



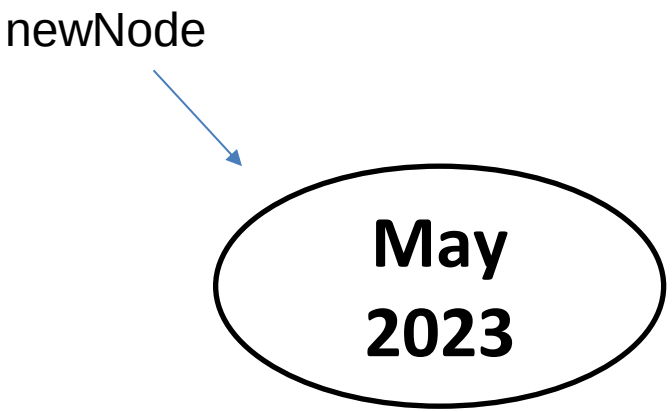
Running time?

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

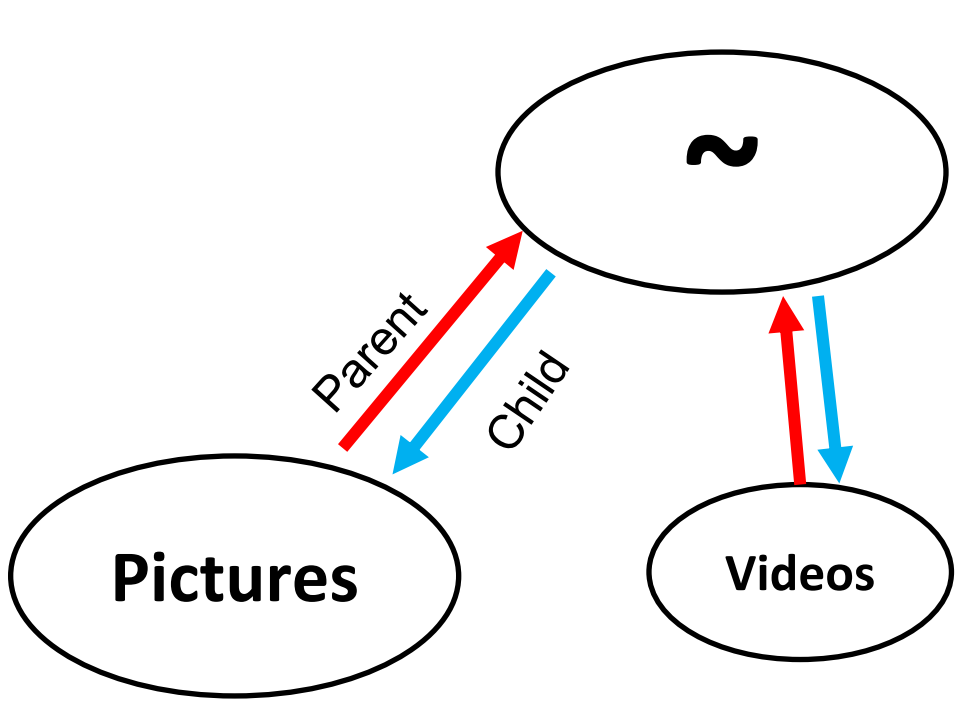
New node that needs to be added?




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

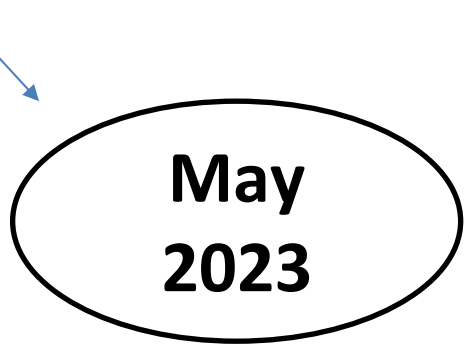


tree.moveDown(“Pictures”);



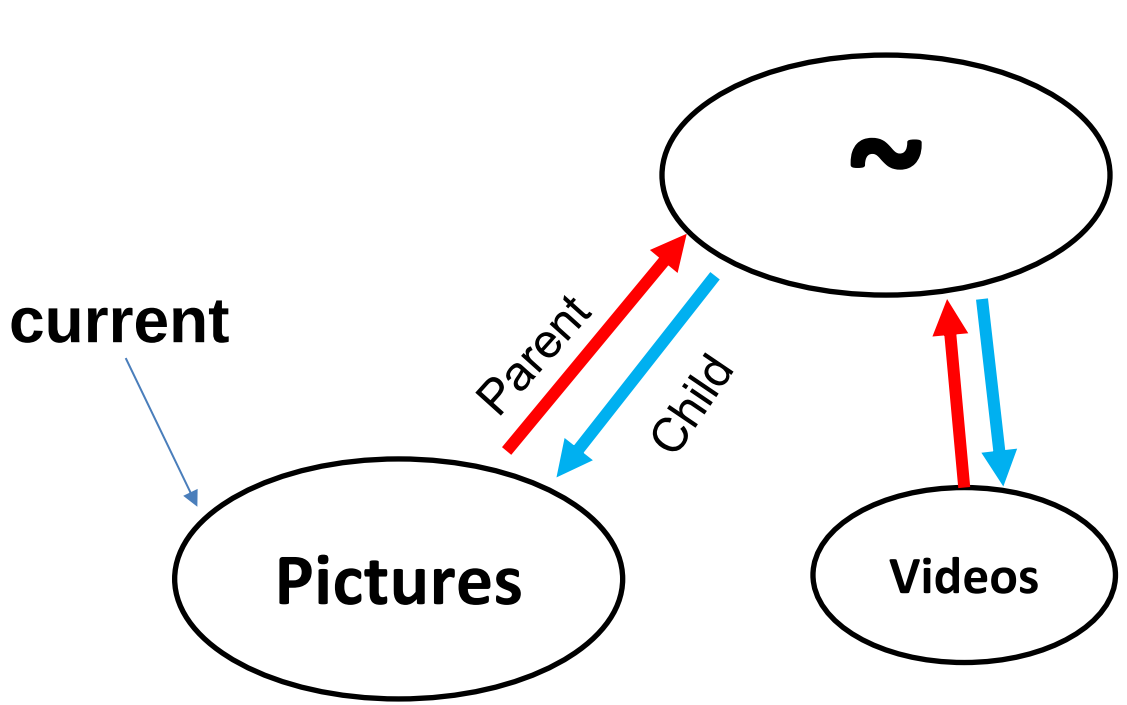

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

newNode



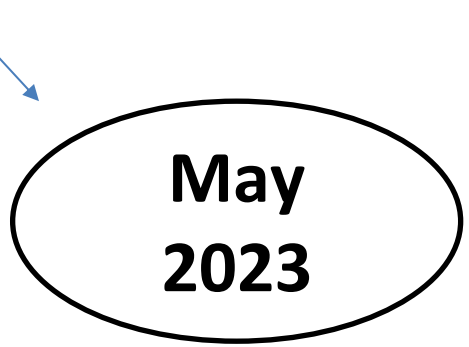

```
public boolean moveDown(String directory) {
   ArrayList<Node> children = current.getChildren();
      for(Node child: children) {
         if(child.getName().equals(directory)) {
            current = child;
            return true;
         }
   }
   return false;
}
```

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



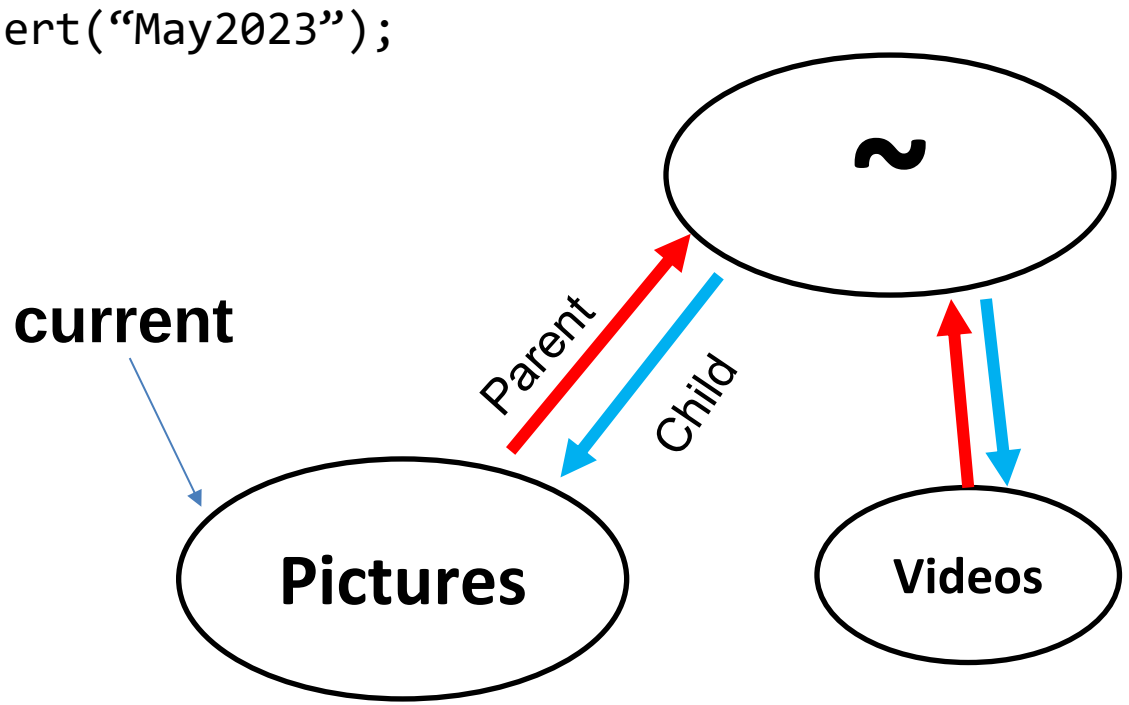

```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
  (…)
}
```

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;
}
```

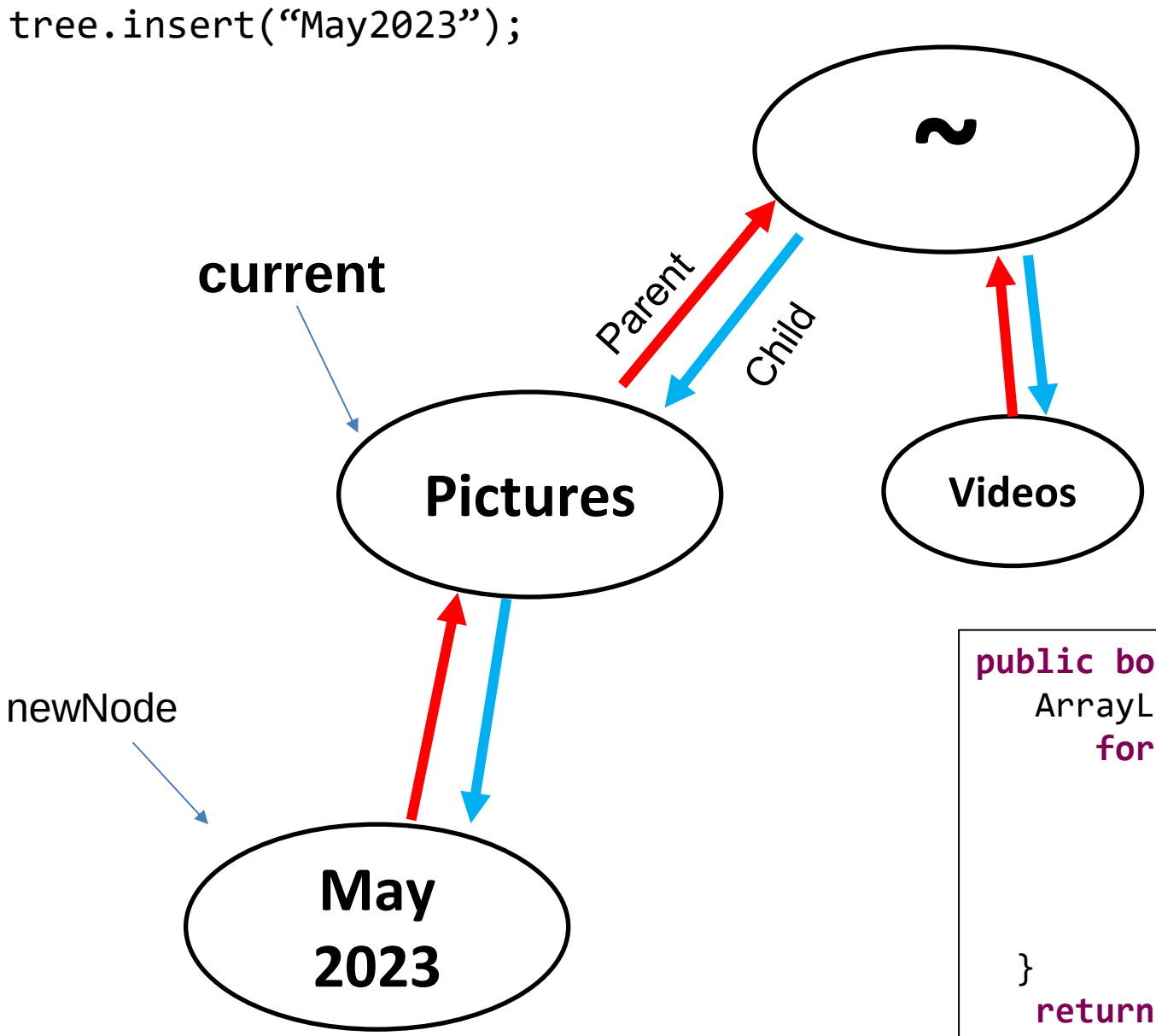
```
tree.moveDown("Pictures");
tree.insert("May2023");
```




```
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");
```

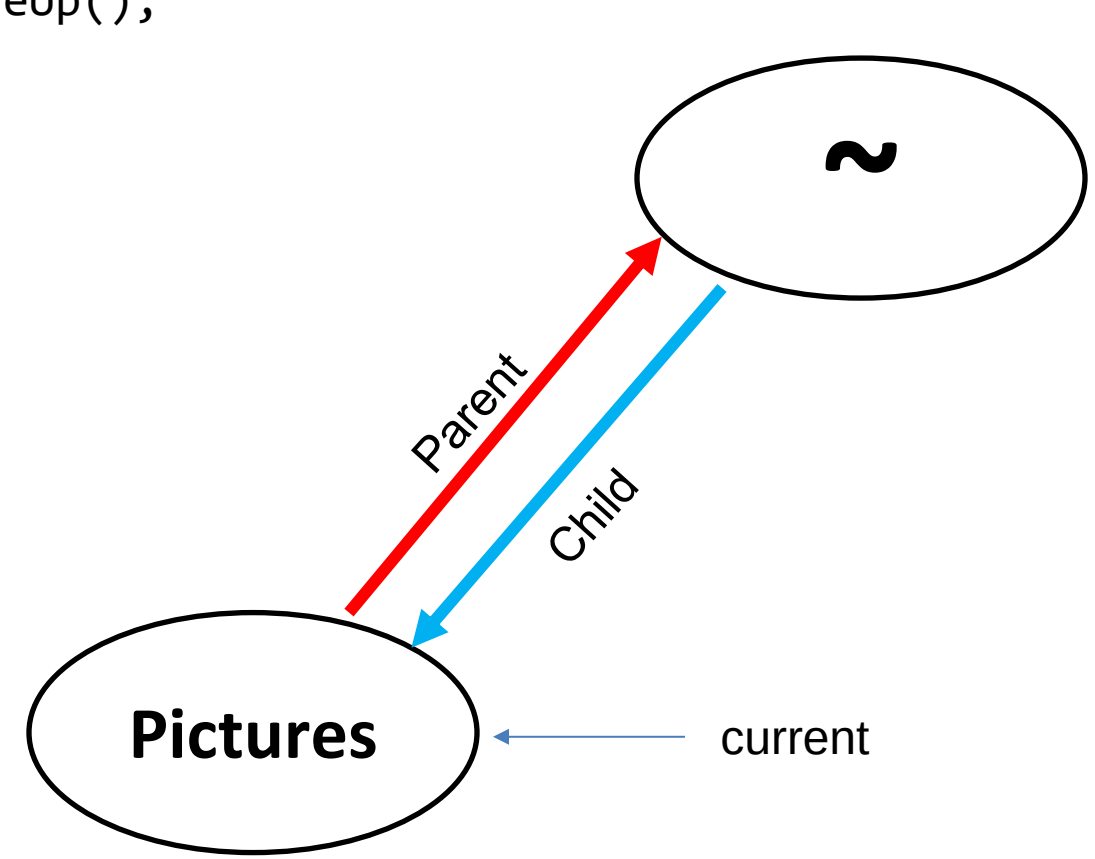



```
public Boolean insert(String directory){
   (…)
   Node newNode = new Node(directory);
   newNode.setParent(current);
   current.addChild(newNode);
   (…)
}
```

```
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;
}
```

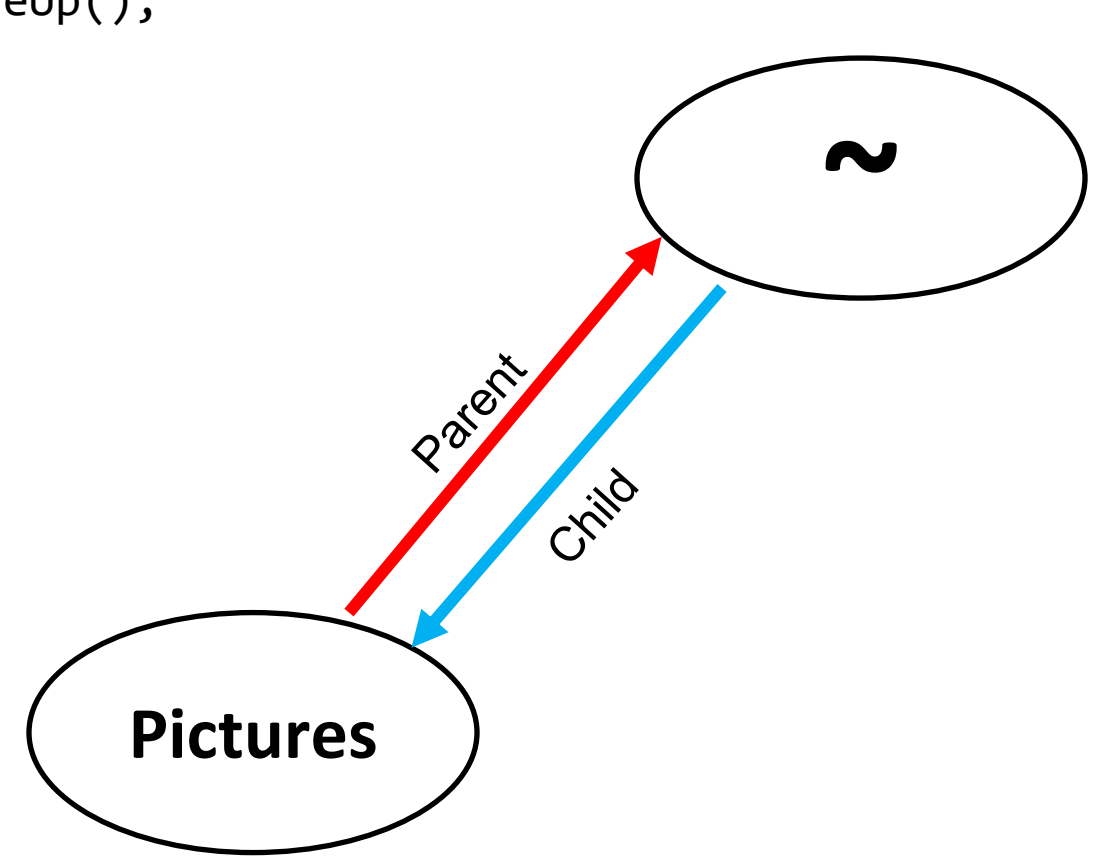
Running time?

```
tree.moveUp();
```



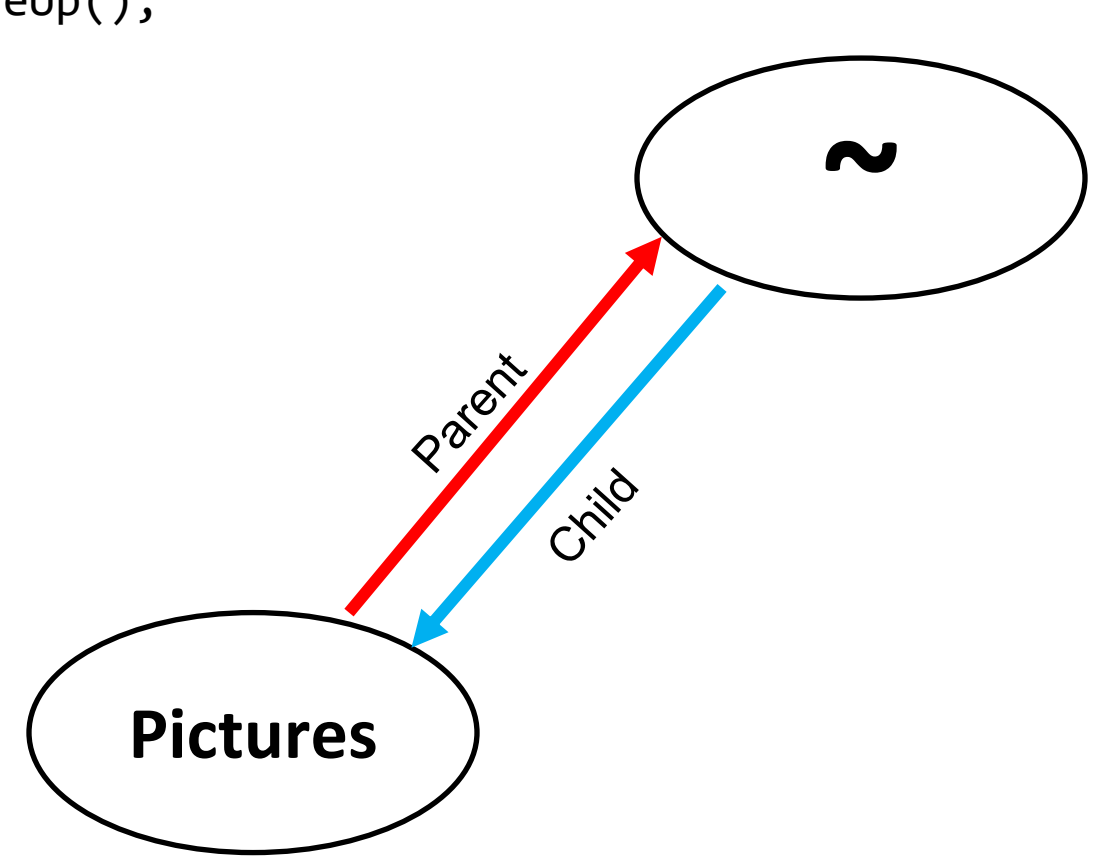

```
public void moveUp() {
    if( current != root ) {
        current = current.getParent();
    }
}
```

```
tree.moveUp();
```




```
public void moveUp() {
    if( current != root ) {
        current = current.getParent();
    }
}
```

```
tree.moveUp();
```



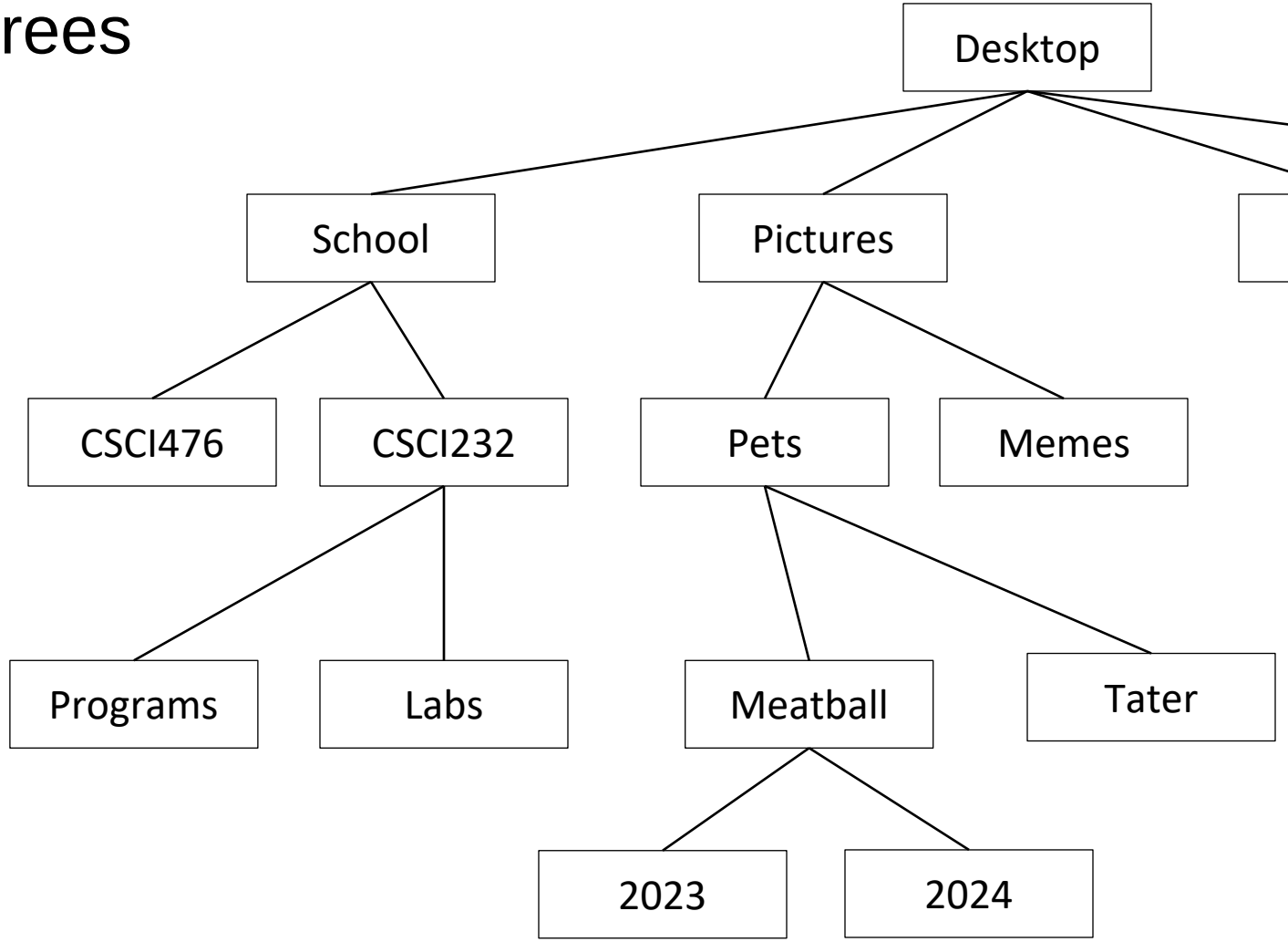

```
public void moveUp() {
    if( current != root ) {
        current = current.getParent();
    }
}
```

Running time?

Running time?

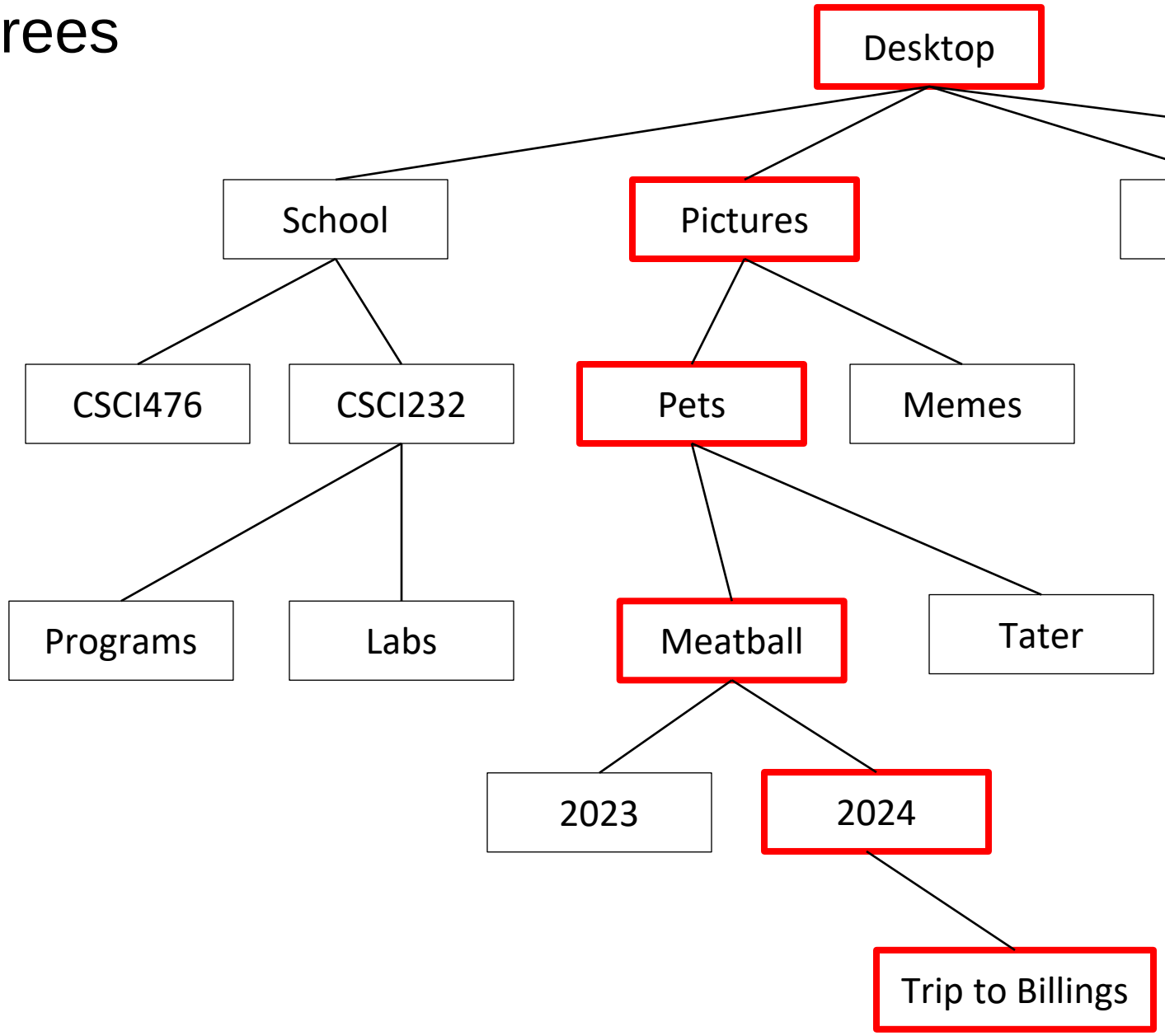
```
public void goHome() {
    current = root;
}
```

Trees



Running time for insert?
(no current)

Trees



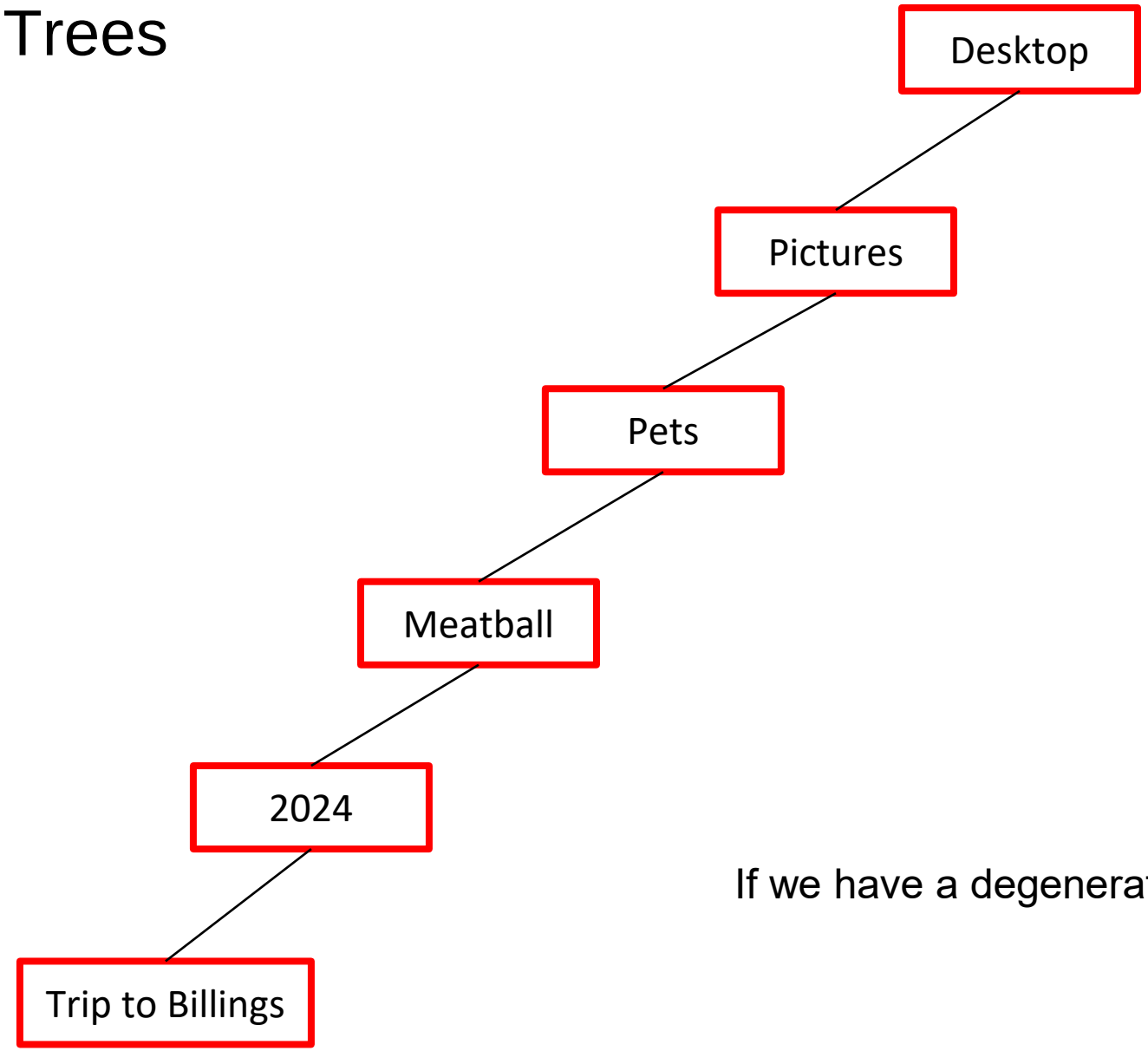
Running time for insert?
***(no current)* (no error checking)**

We may have to traverse the entire **height** of the tree

O(h) where h is the height of the tree

`addNode(Desktop/Pictures/Pets/Meatball/2024/Trip To Billings)`

Trees



We may have to traverse the entire **height** of the tree

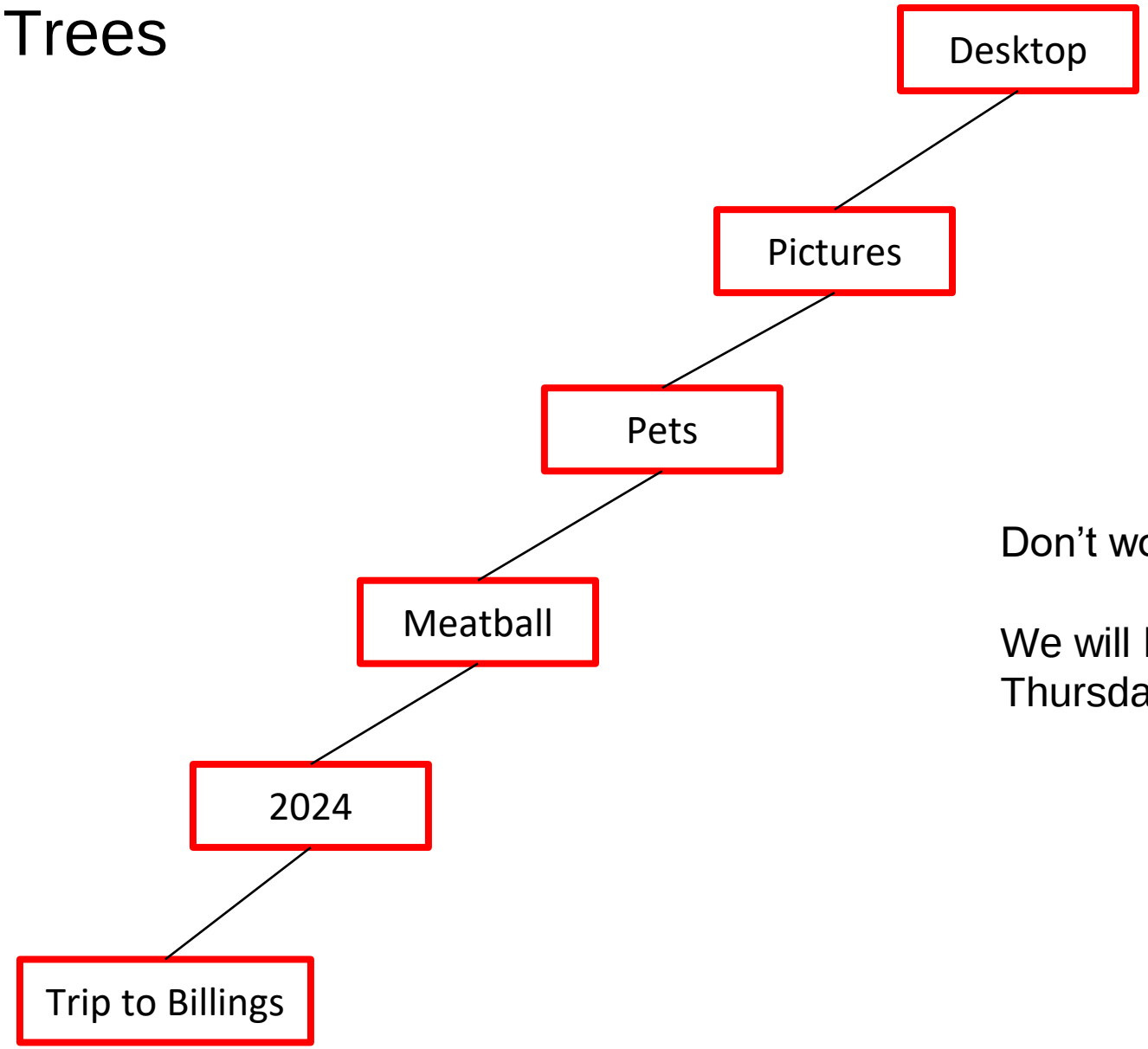
O(h) where h is the height of the tree

Height of tree = Number of nodes in tree - 1

If we have a degenerate tree, our running time really isn't that great

```
addNode(Desktop/Pictures/Pets/Meatball/2024/Trip To Billings)
```

Trees

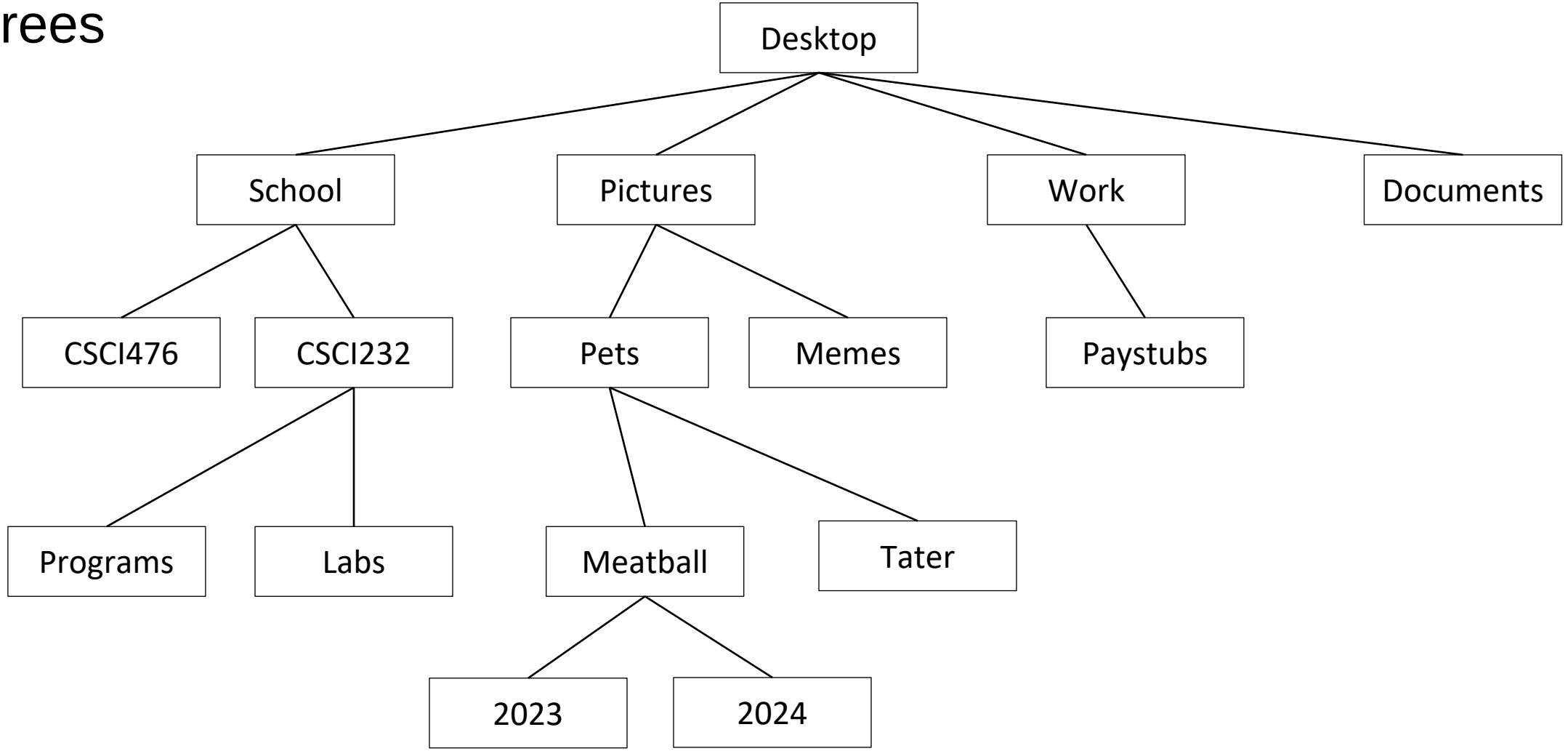


Don’t worry too much about running time for Trees.

We will look at some special types of trees on Thursday that have better running time

```
addNode(Desktop/Pictures/Pets/Meatball/2024/Trip To Billings)
```

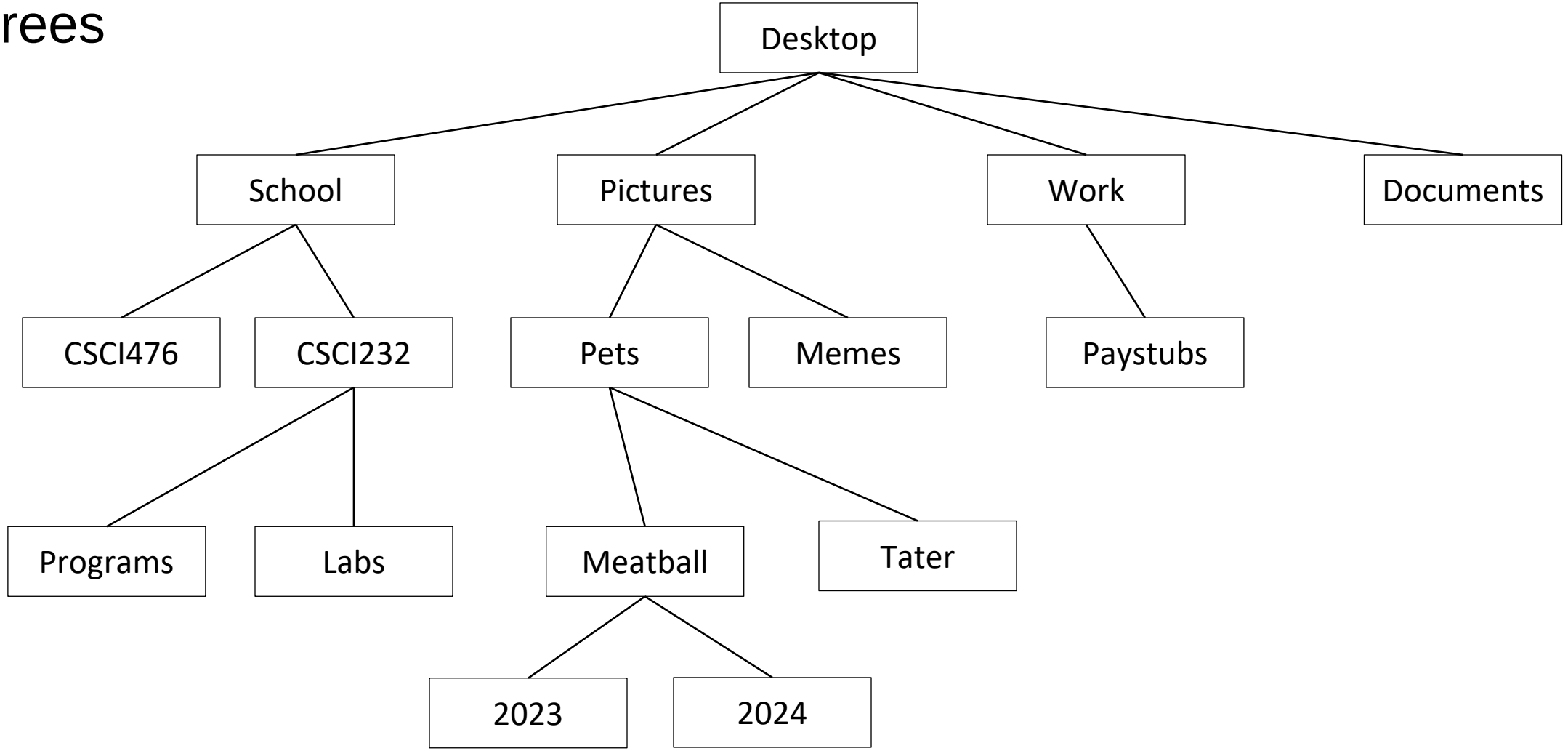
Trees



How could you search for a value in a tree?

```
public Node searchForDirectory(“Meatball”);
```

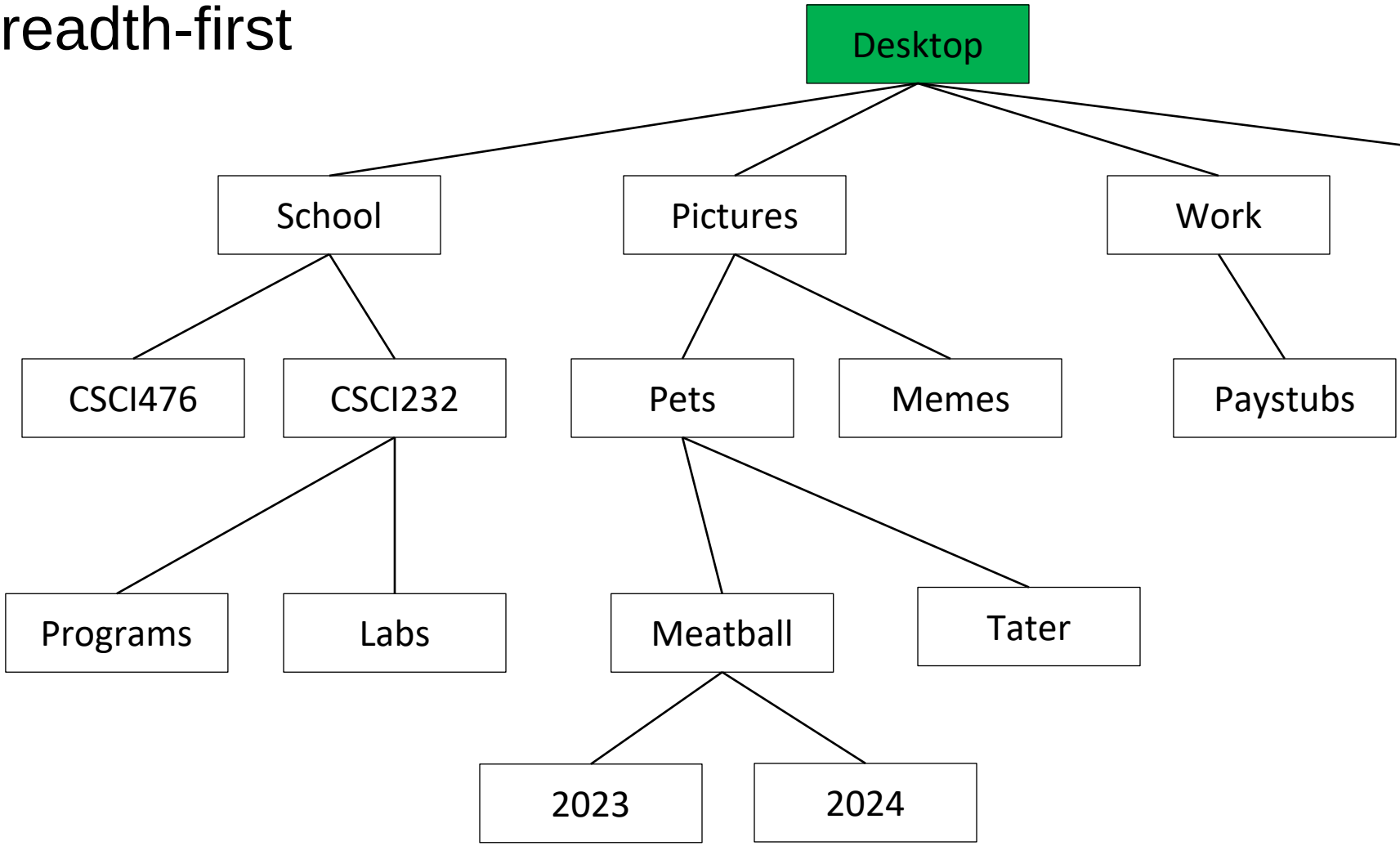
Trees



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

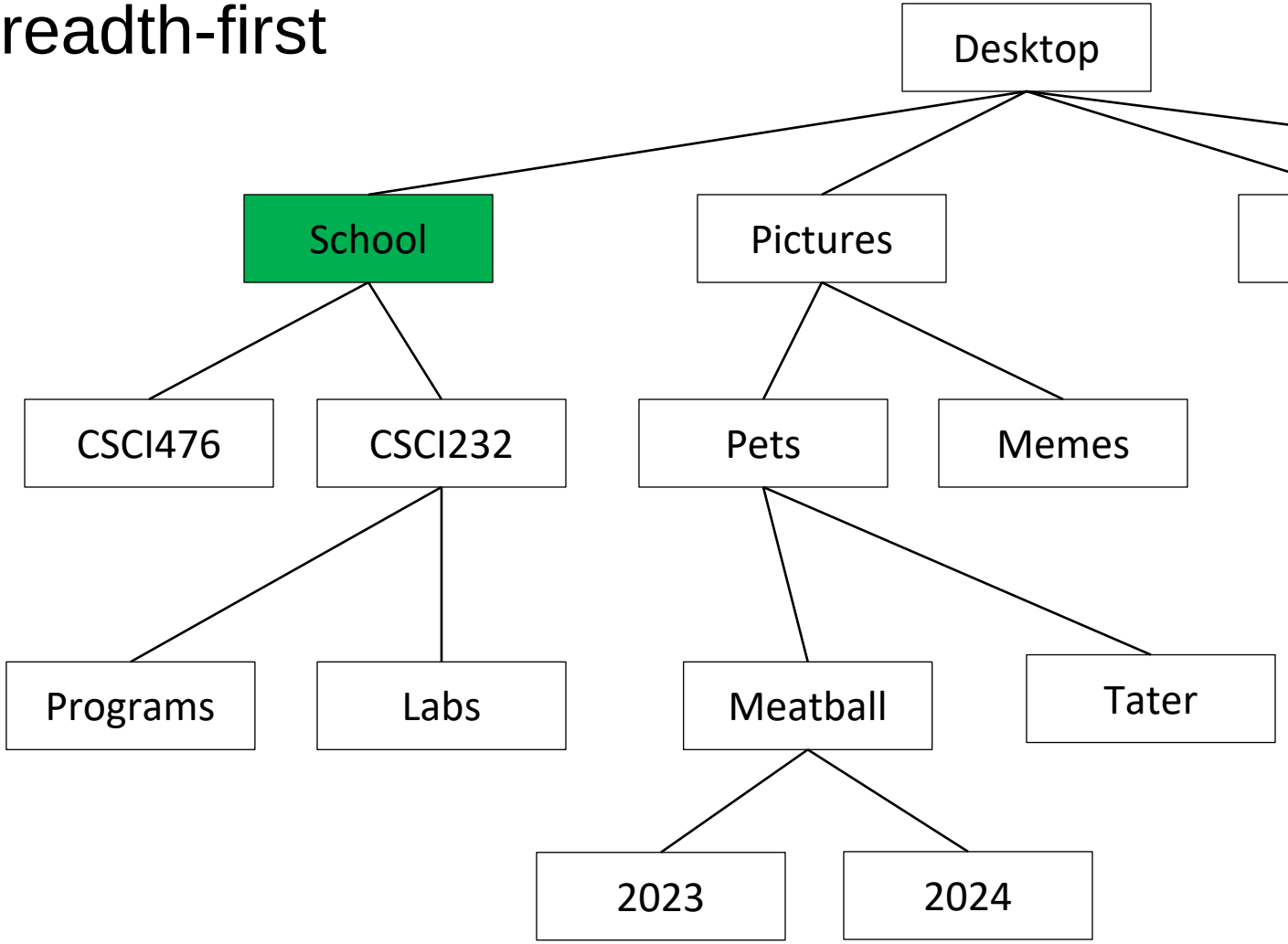
Breadth-first



1. **Visit Root**

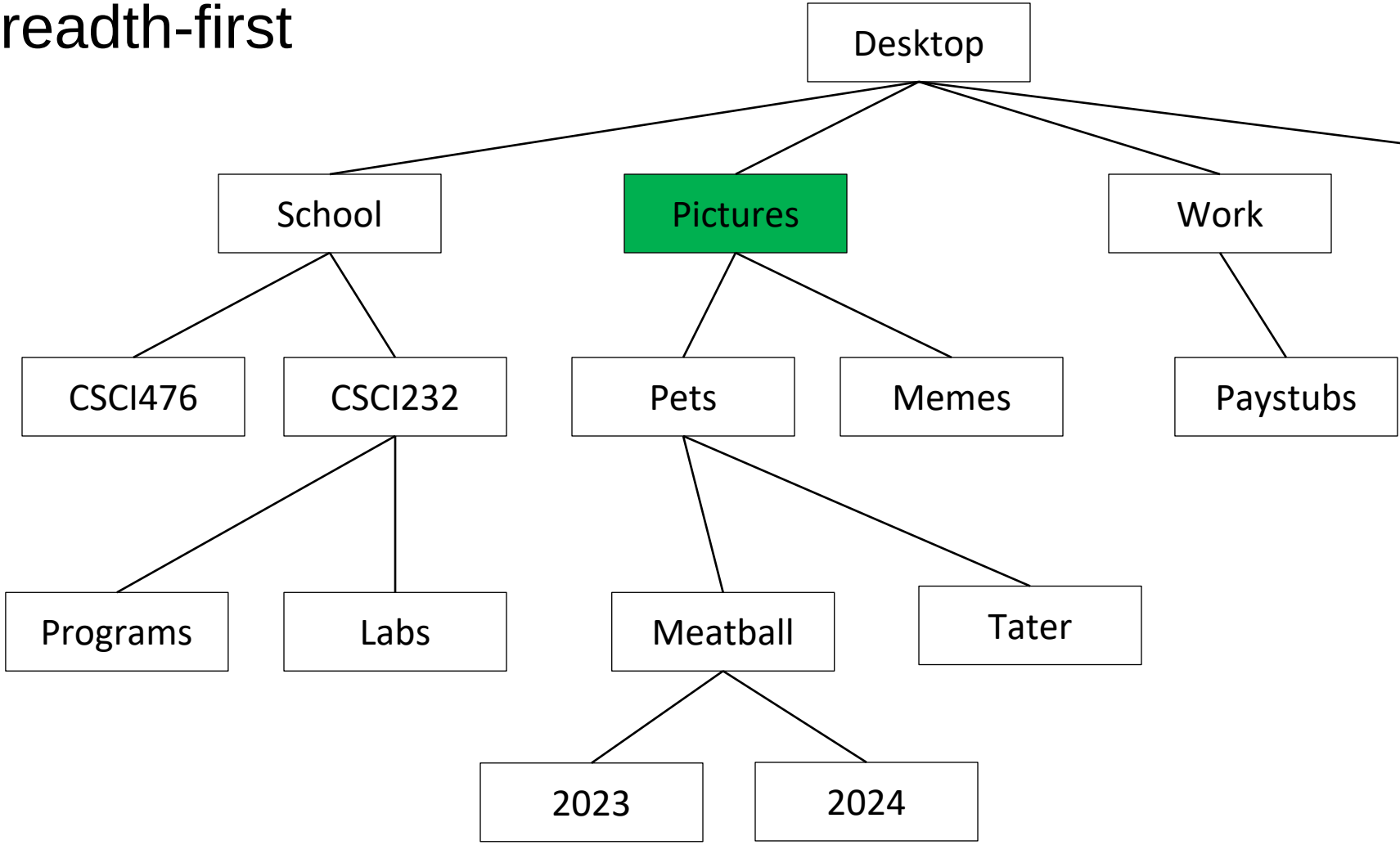
"Visit" is a generic action. The actual action depends on what the application is (ex: print node, compare, update)

Breadth-first



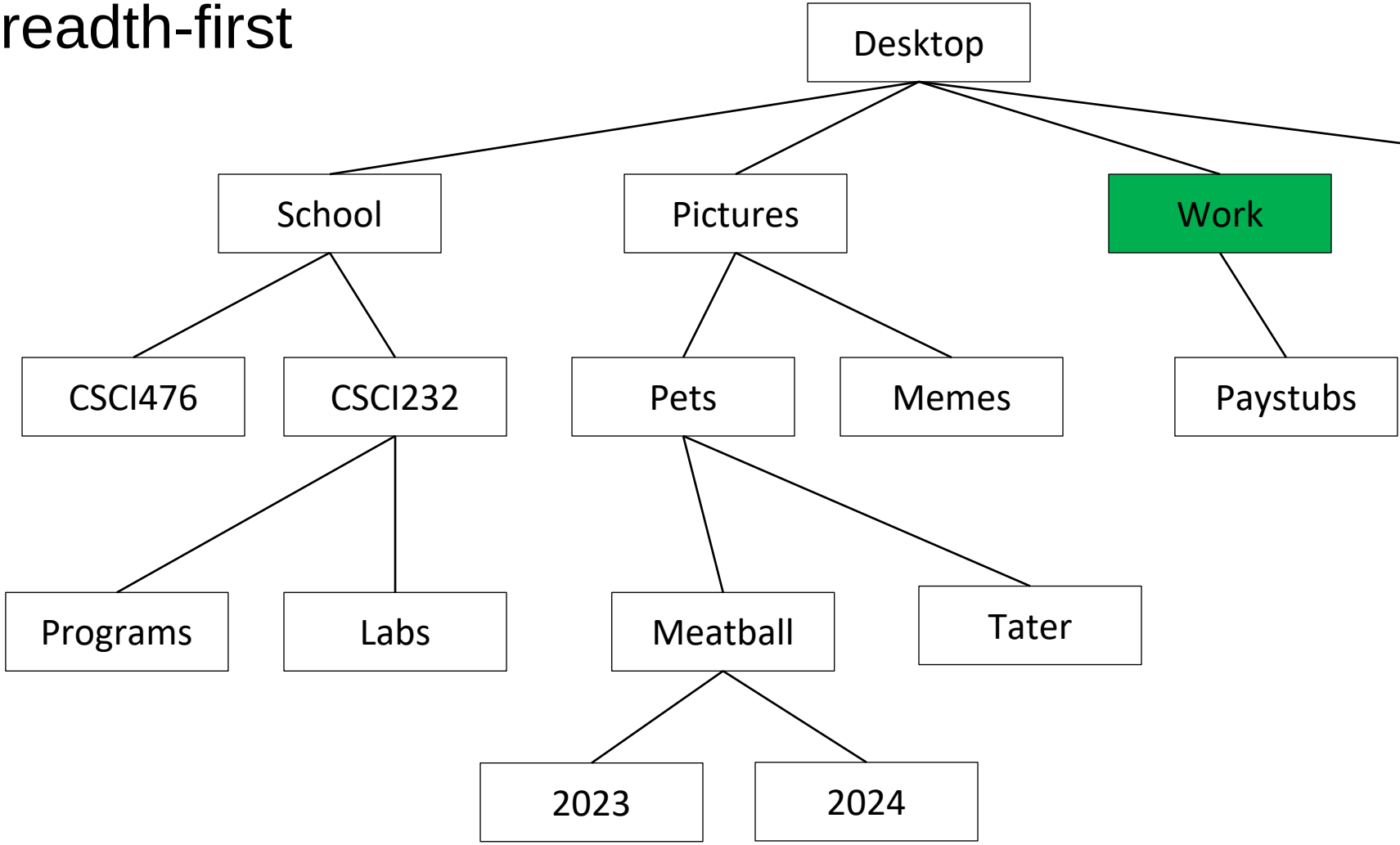
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**

Breadth-first



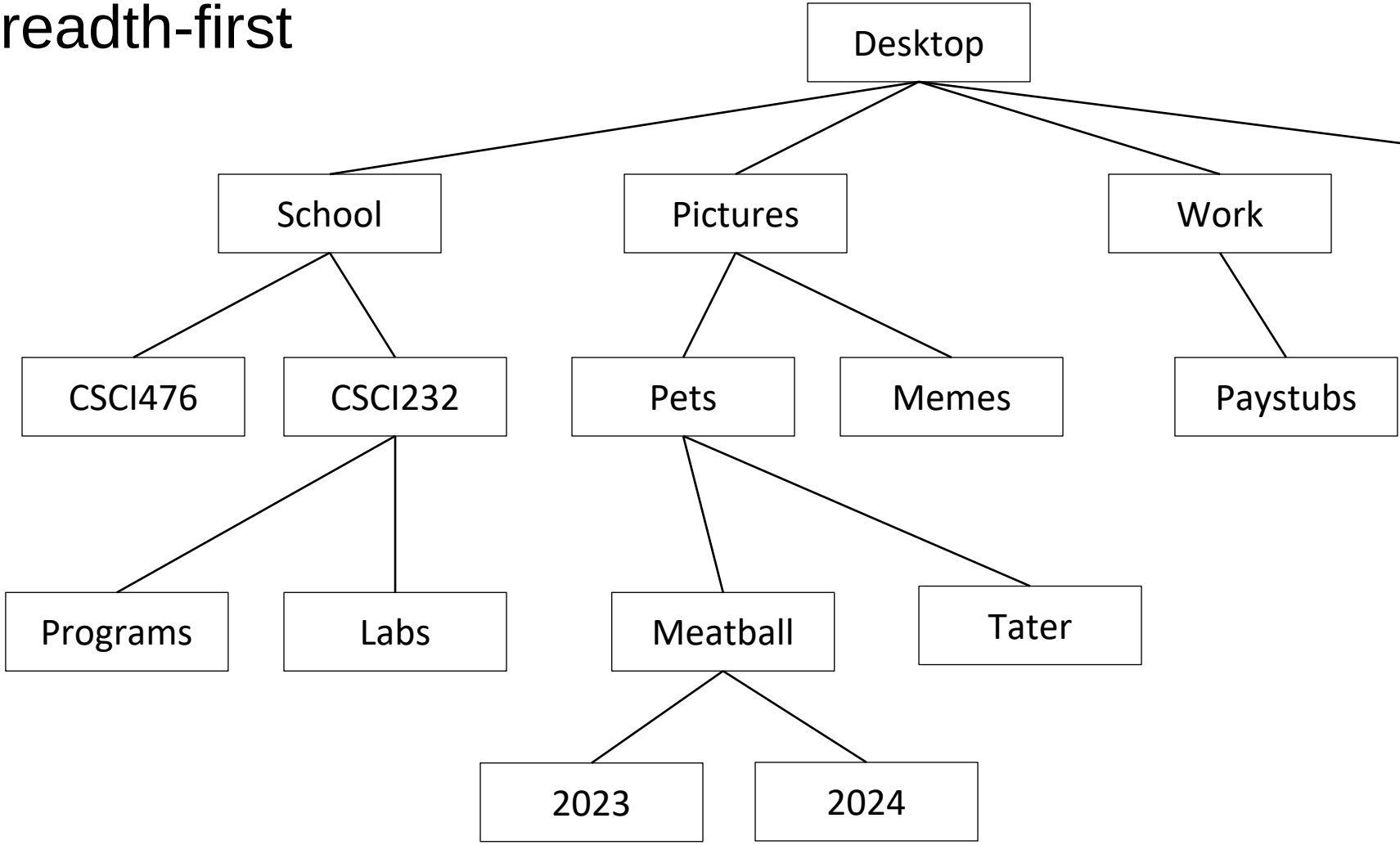
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**

Breadth-first



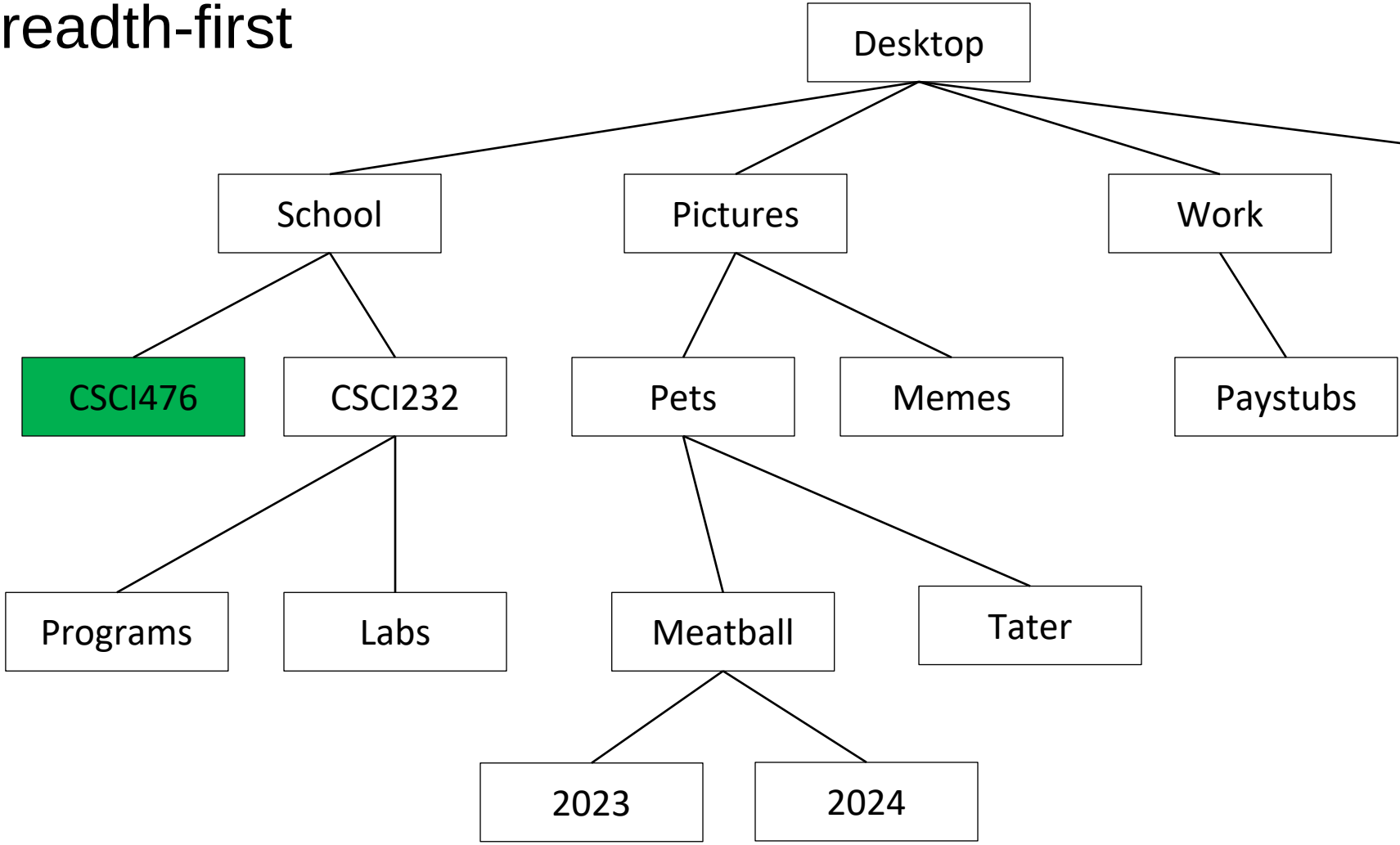
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**

Breadth-first



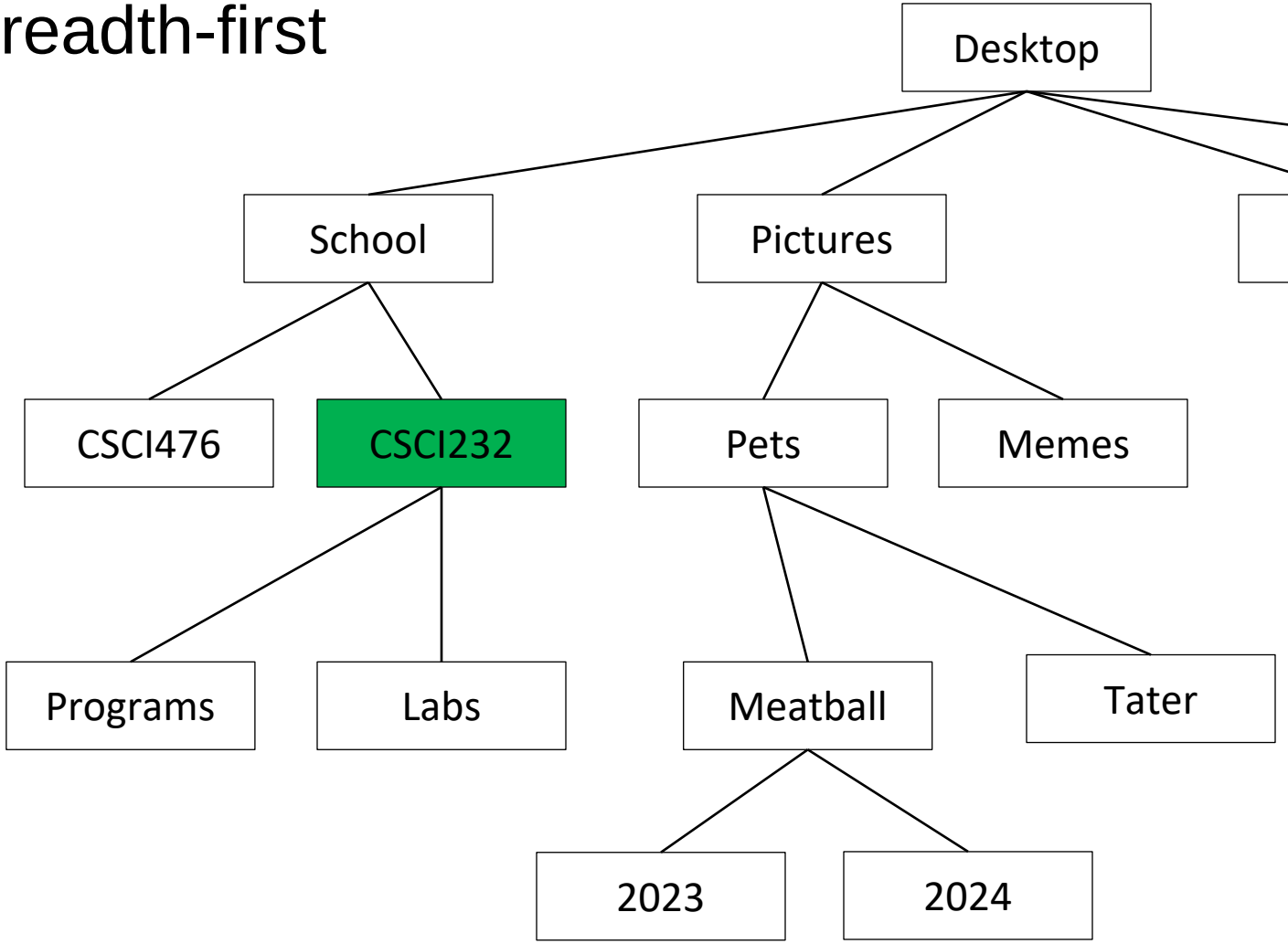
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**

Breadth-first



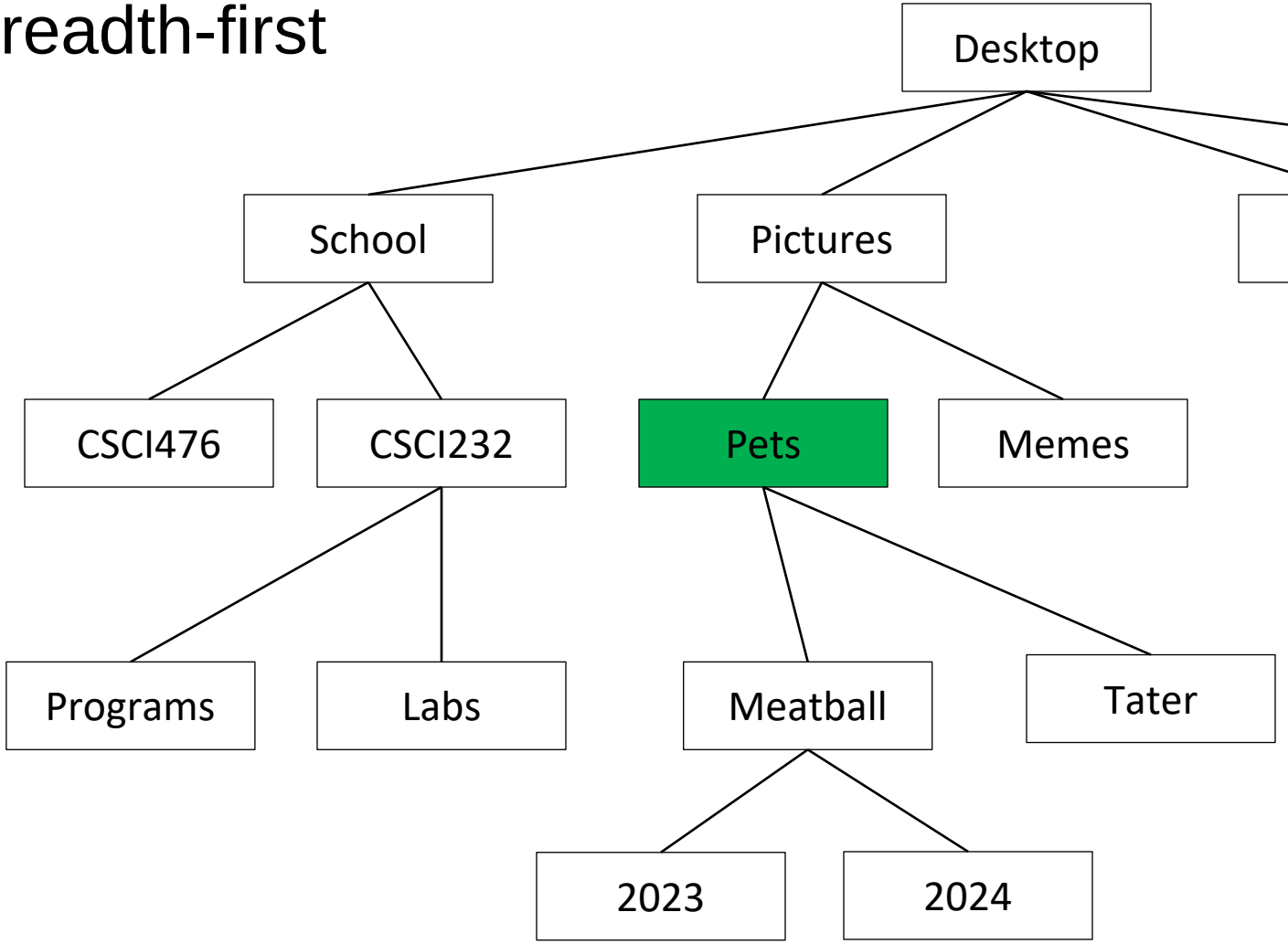
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

Breadth-first



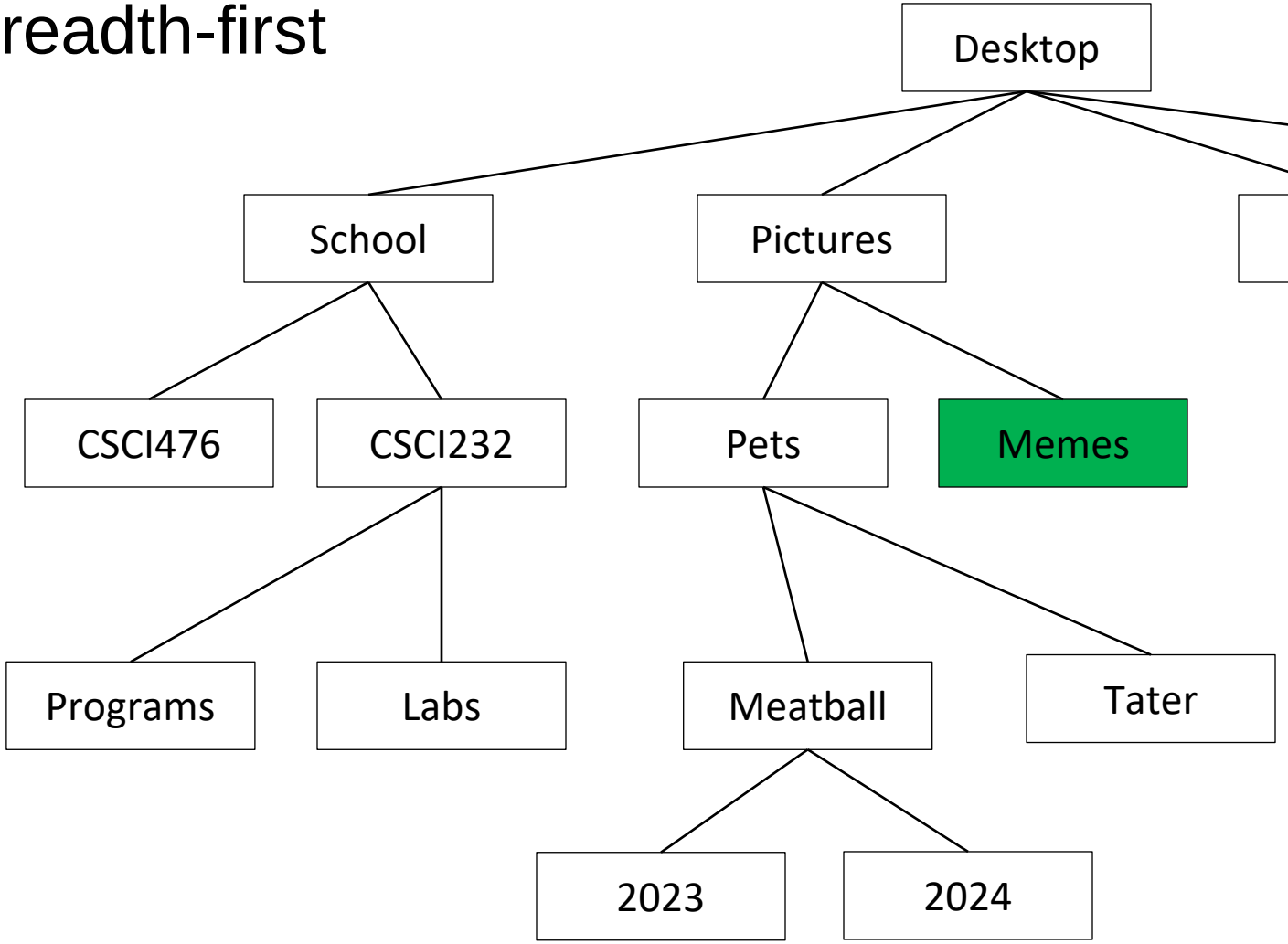
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

Breadth-first



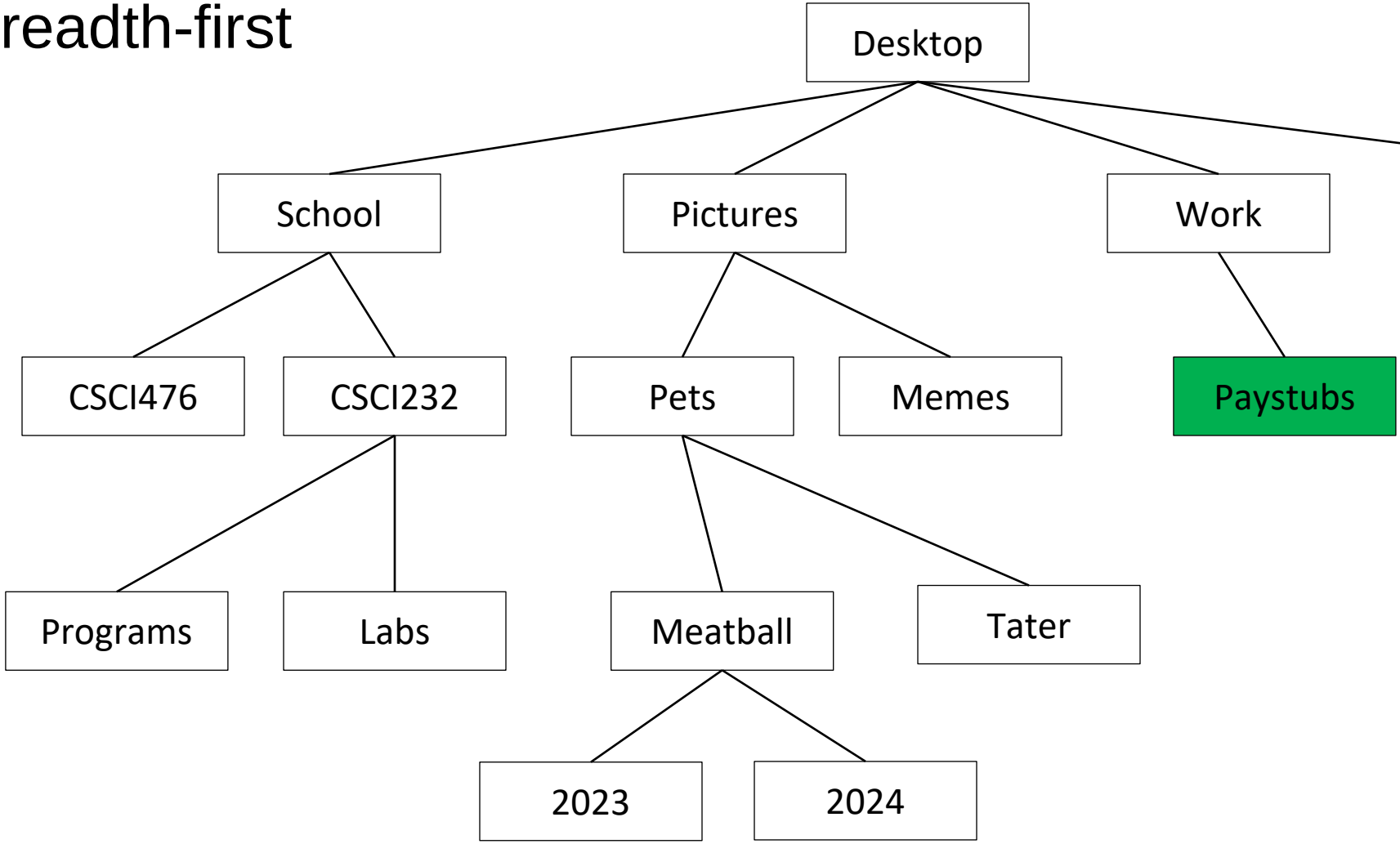
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

Breadth-first



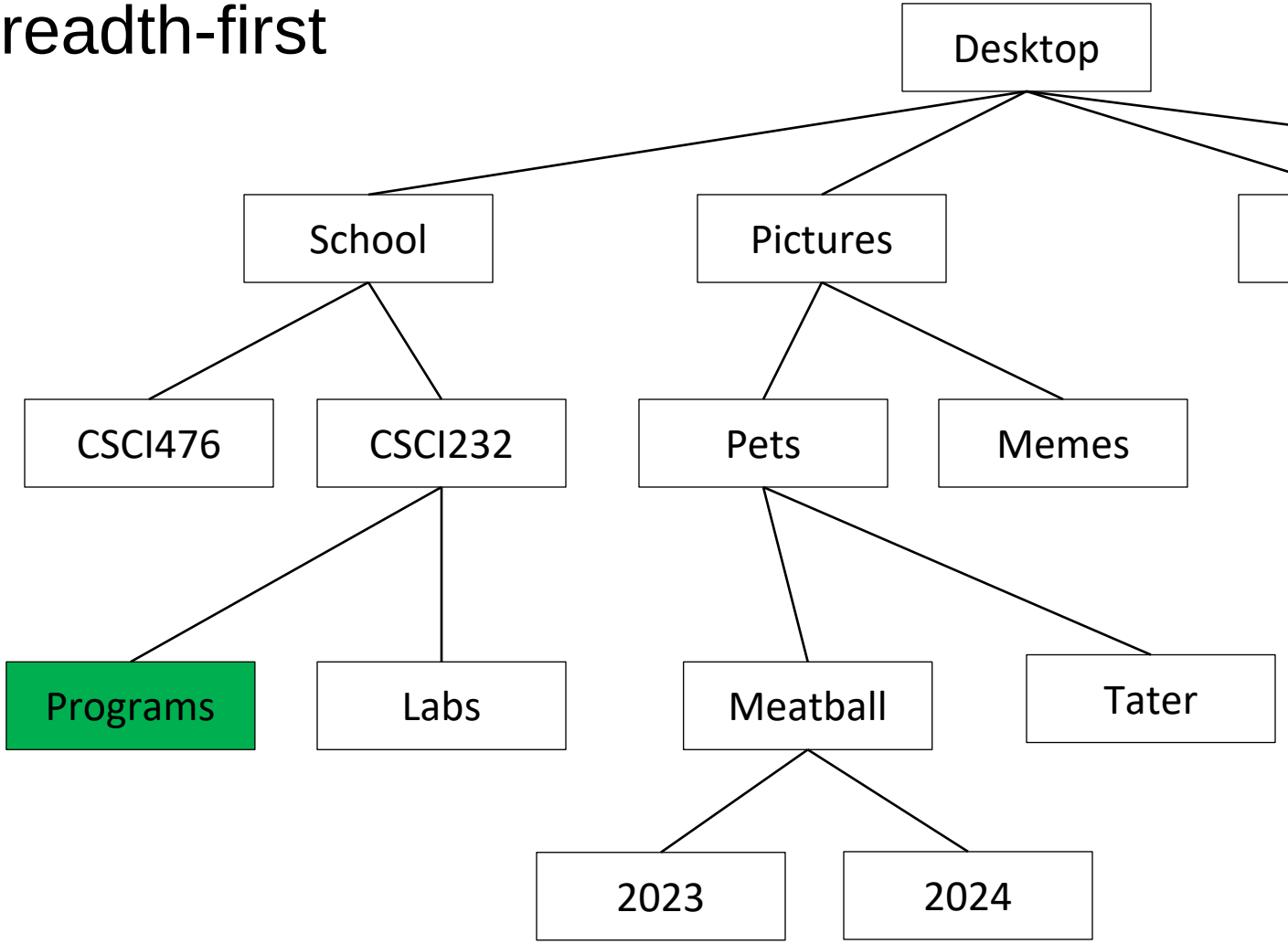
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

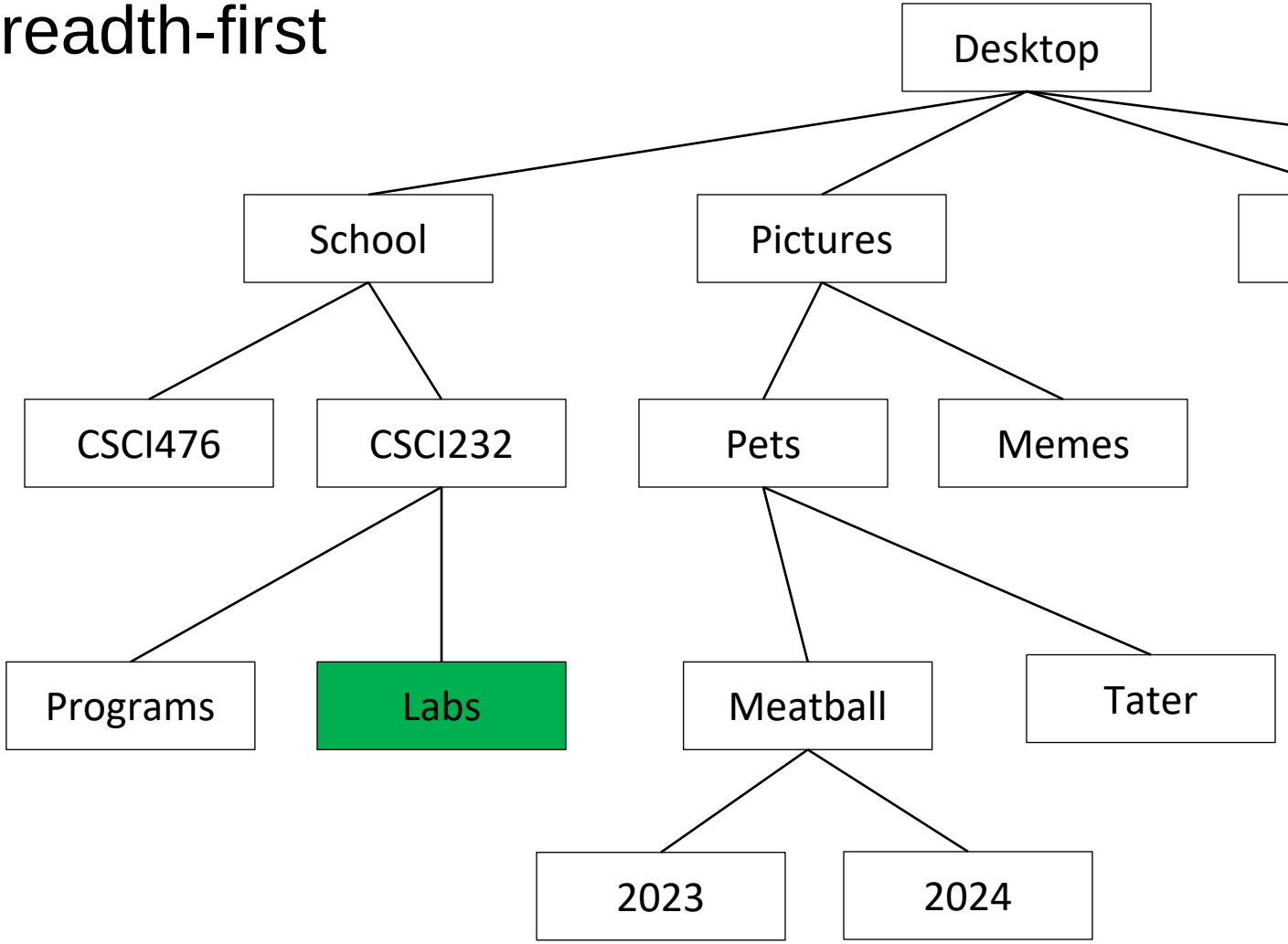
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

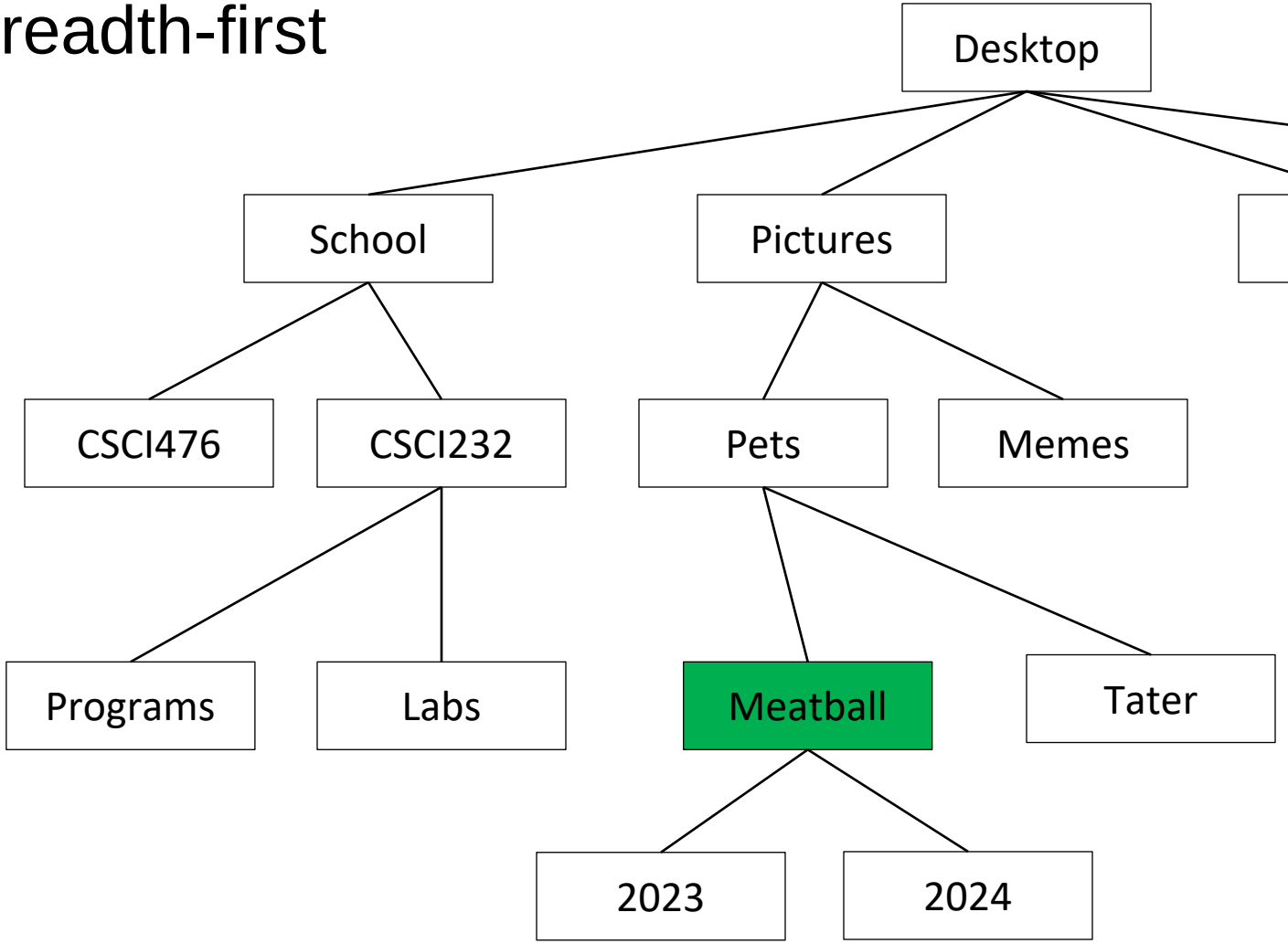
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

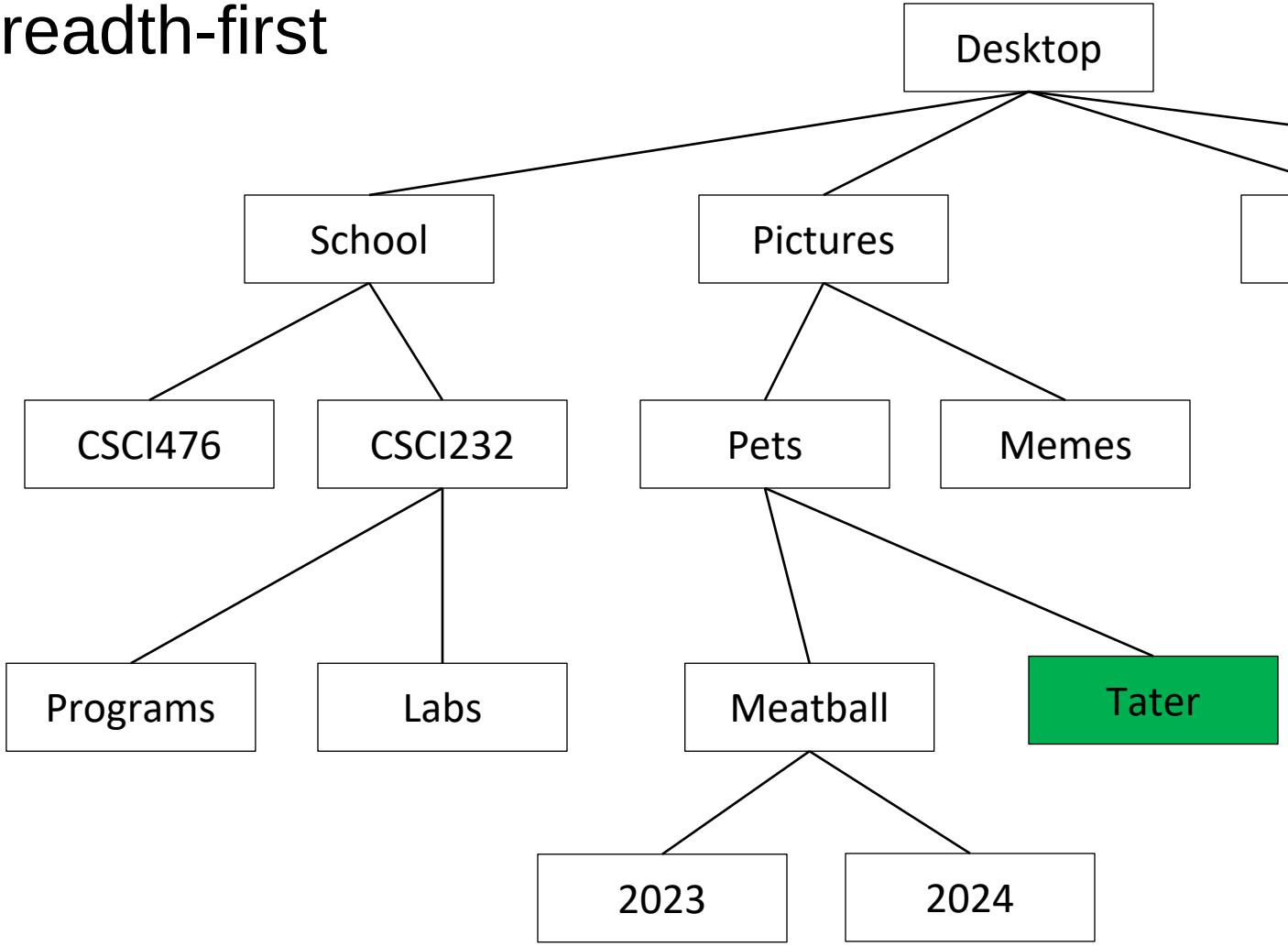
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

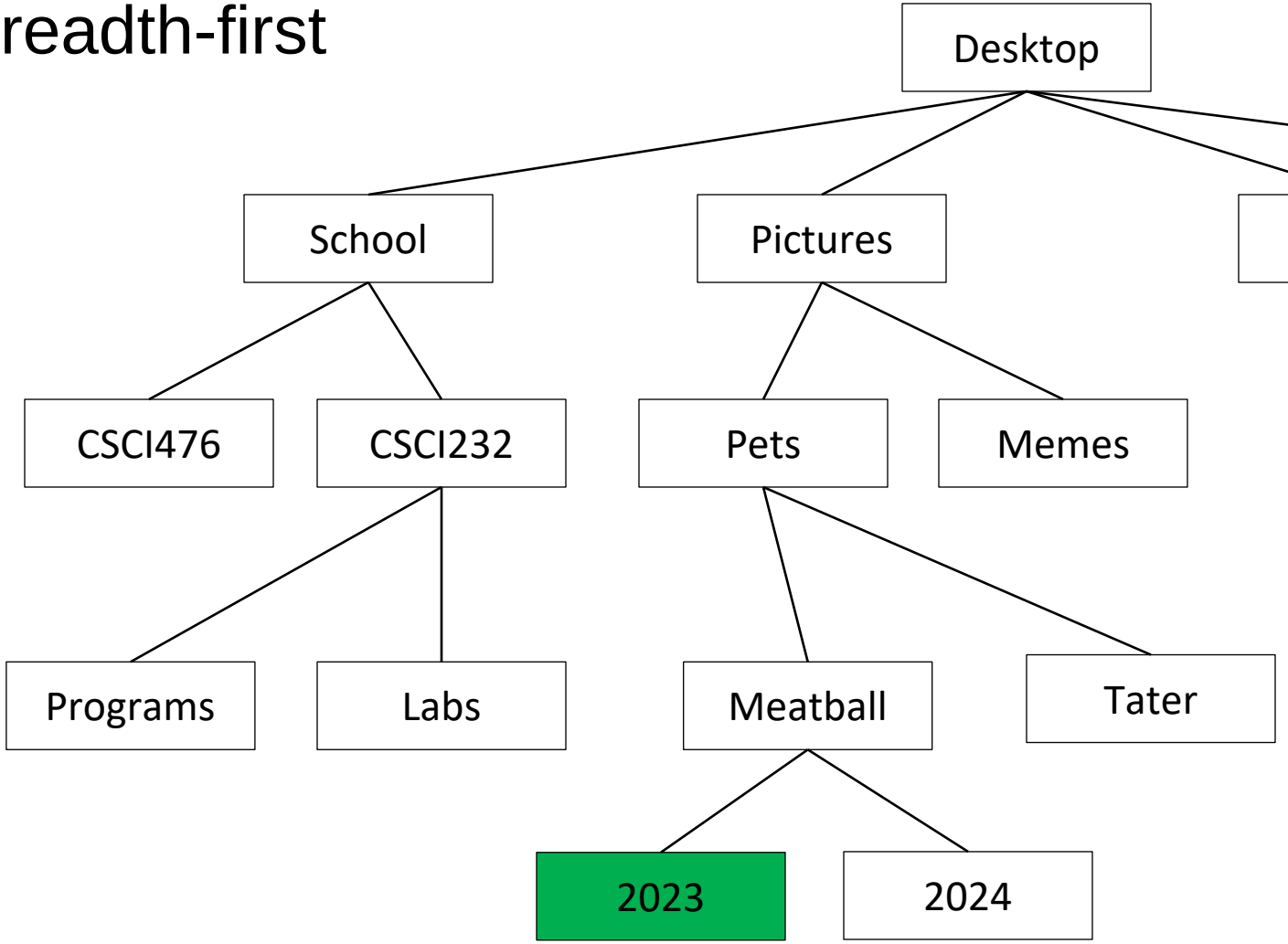
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

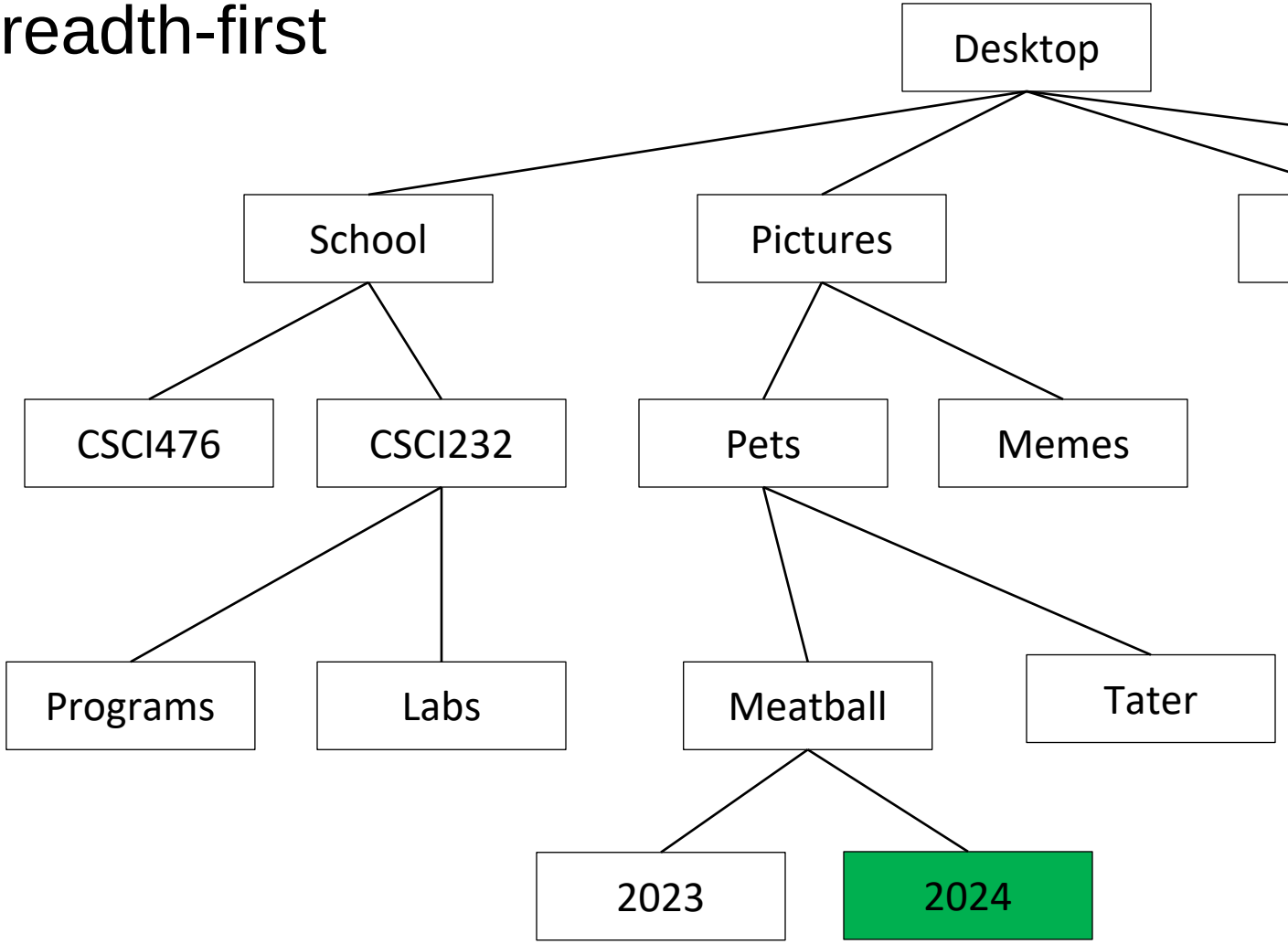
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

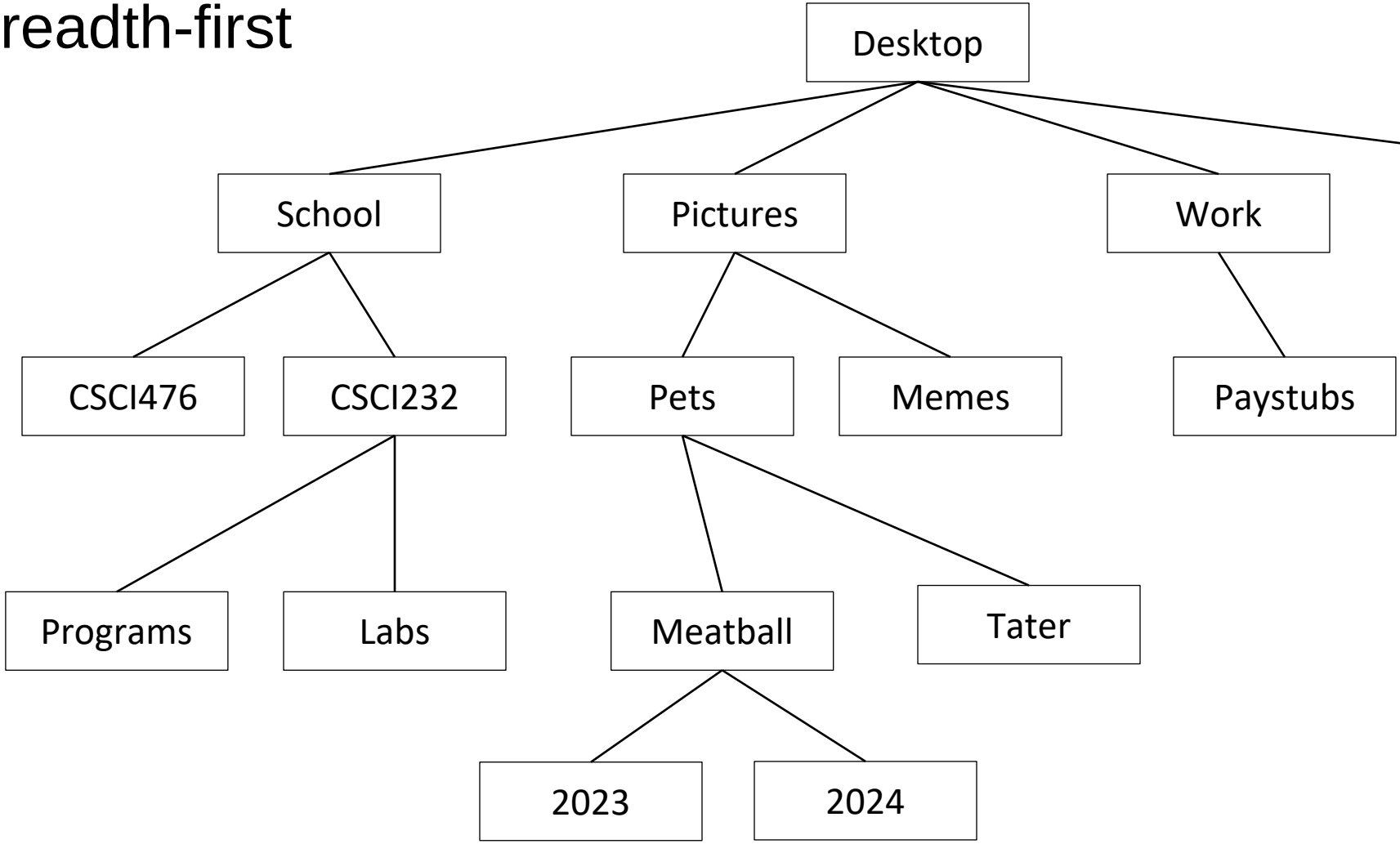
Breadth-first



1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

Breadth-first

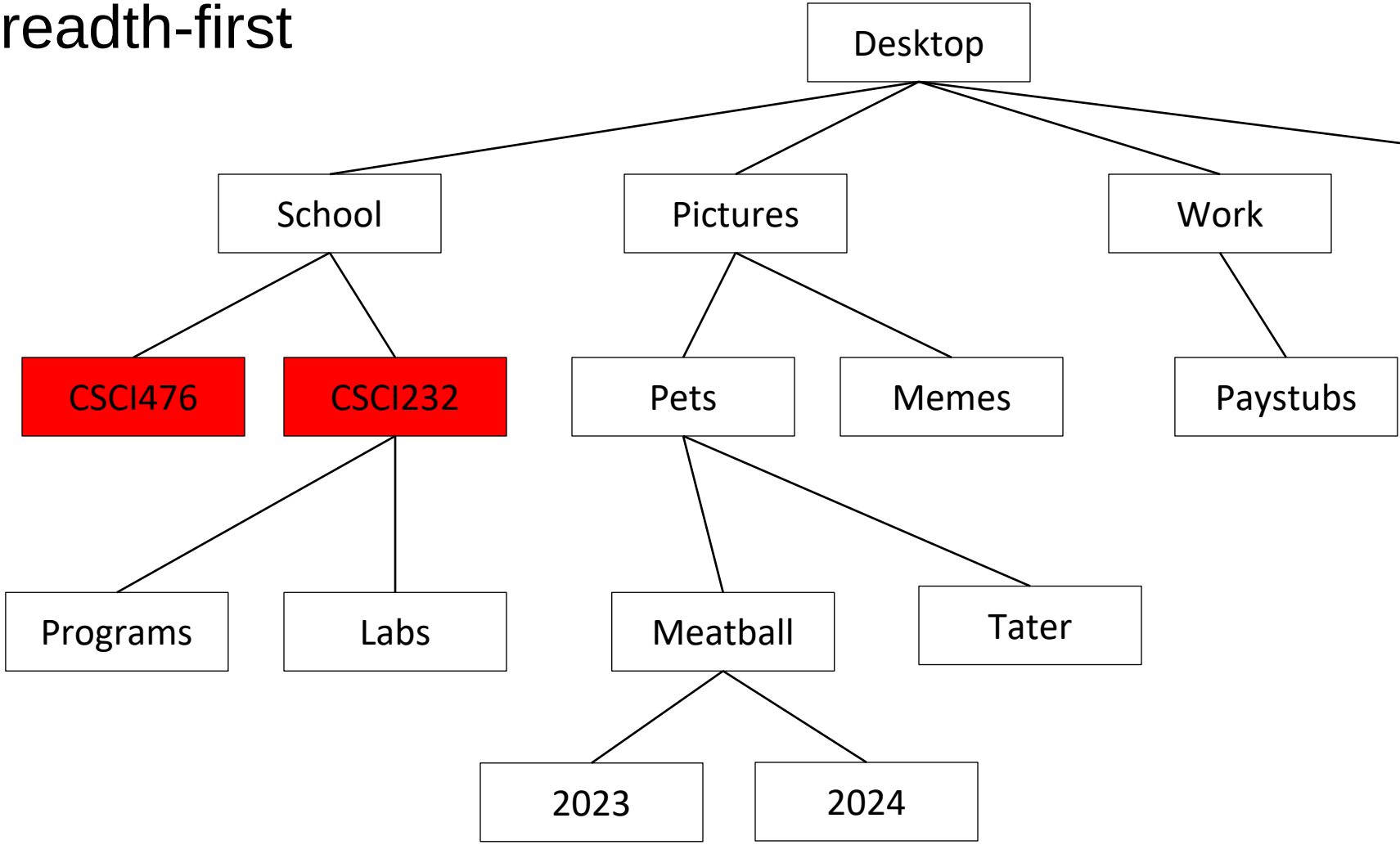


1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

….

How to implement this ?

Breadth-first



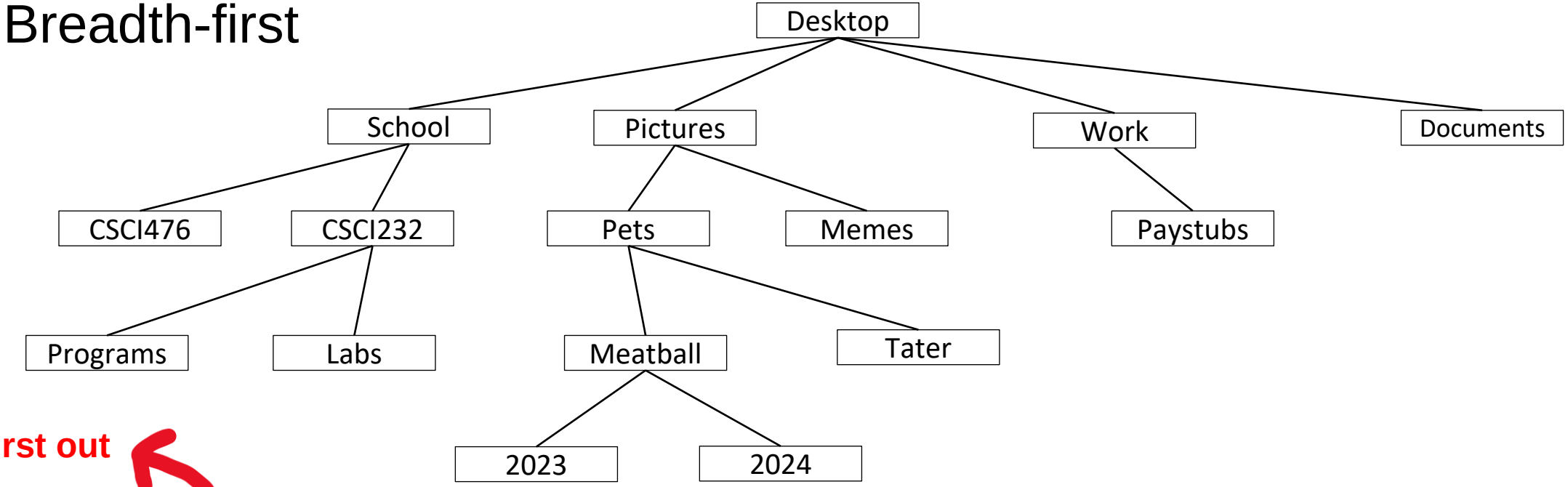
1. **Visit Root**
2. **Visit all depth 1 nodes (in order)**
3. **Visit all depth 2 nodes (in order)**

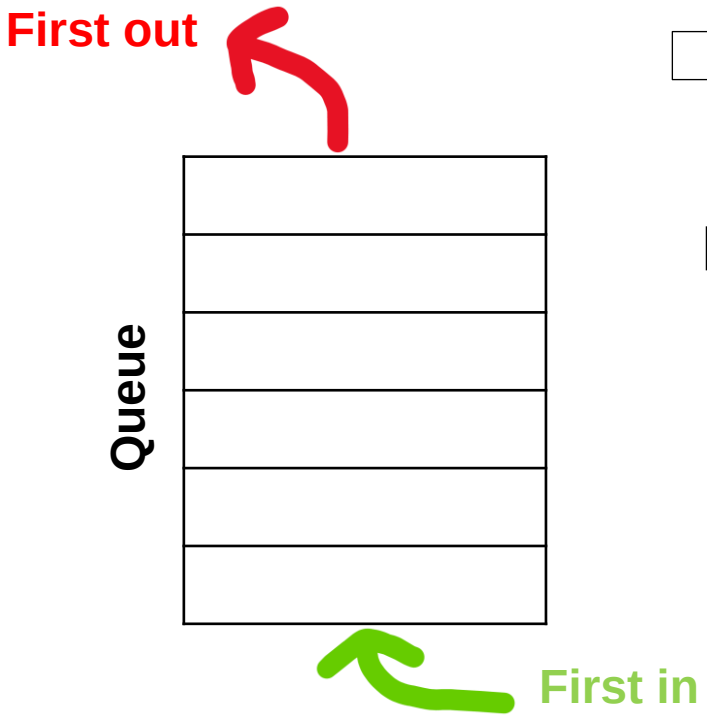
….

How to implement this ?

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

Breadth-first



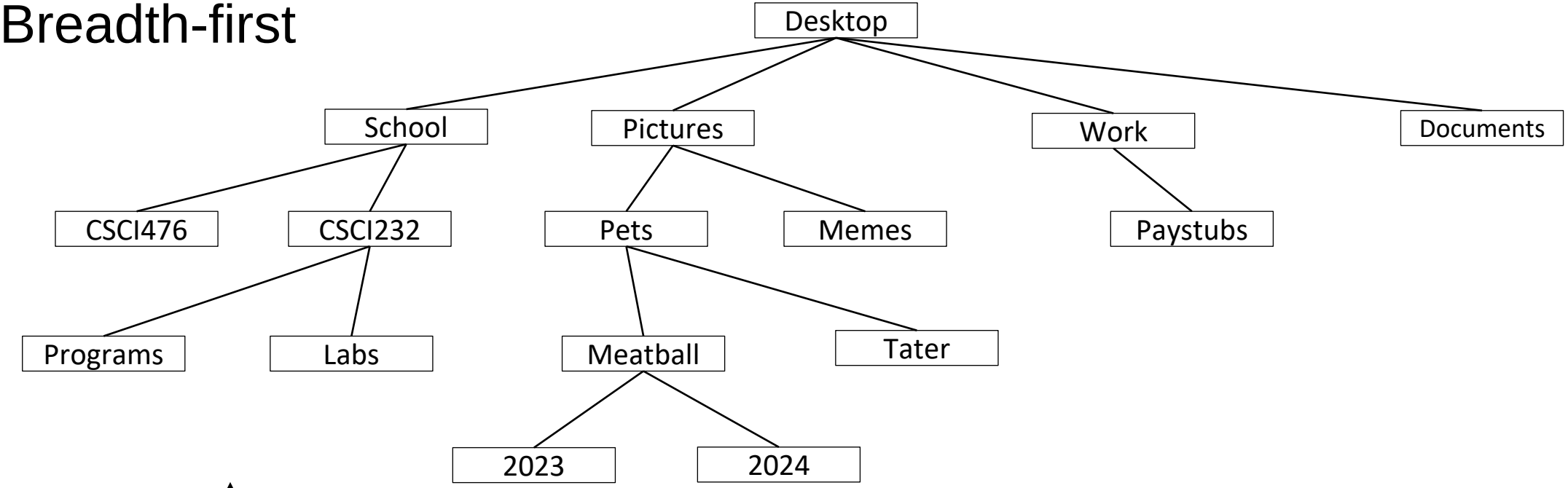


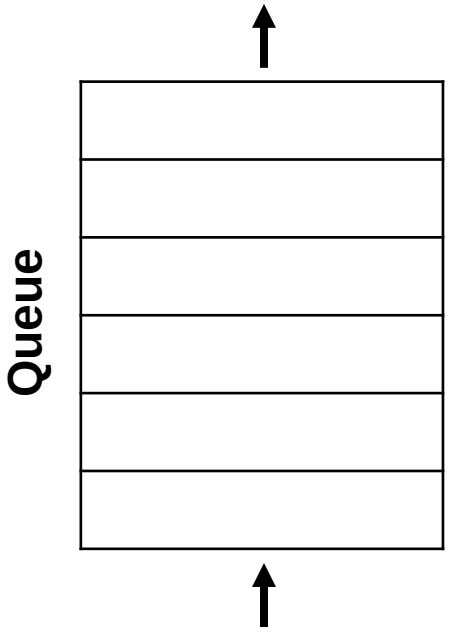
How to implement this ?

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

What if we use a **Queue** ?

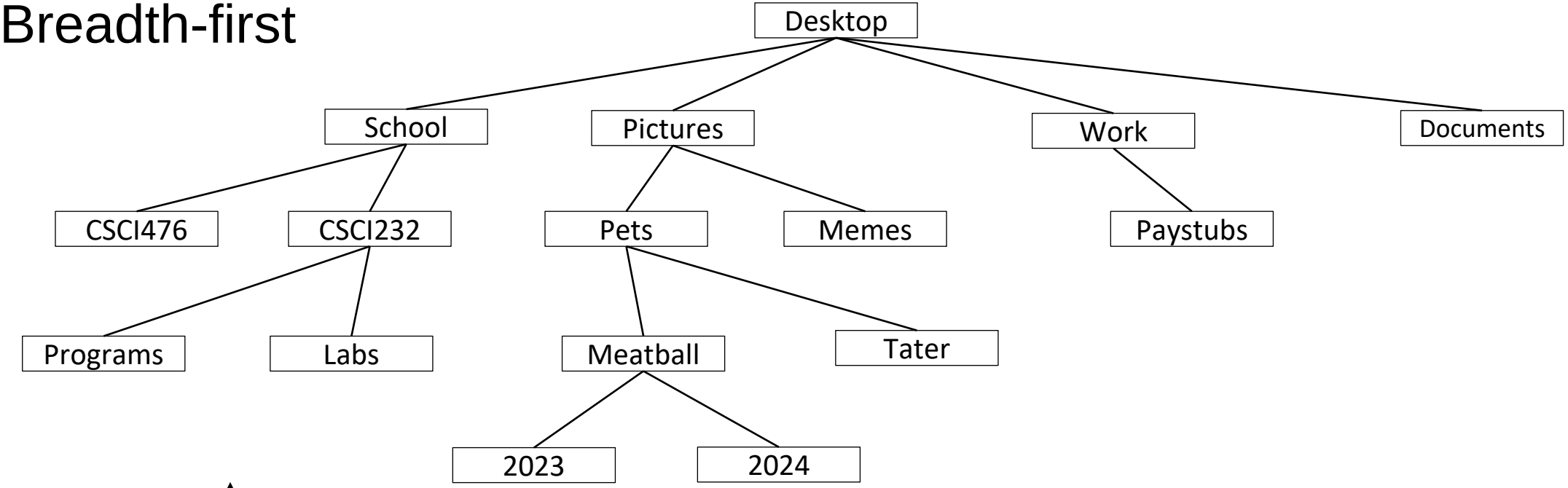
Breadth-first

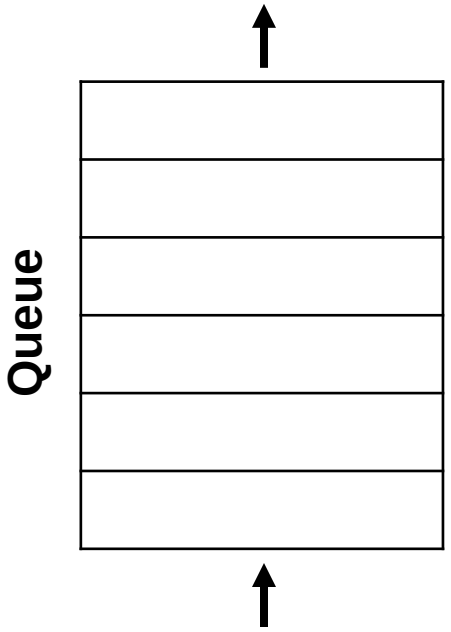




Every time we “visit” a node we:

Breadth-first

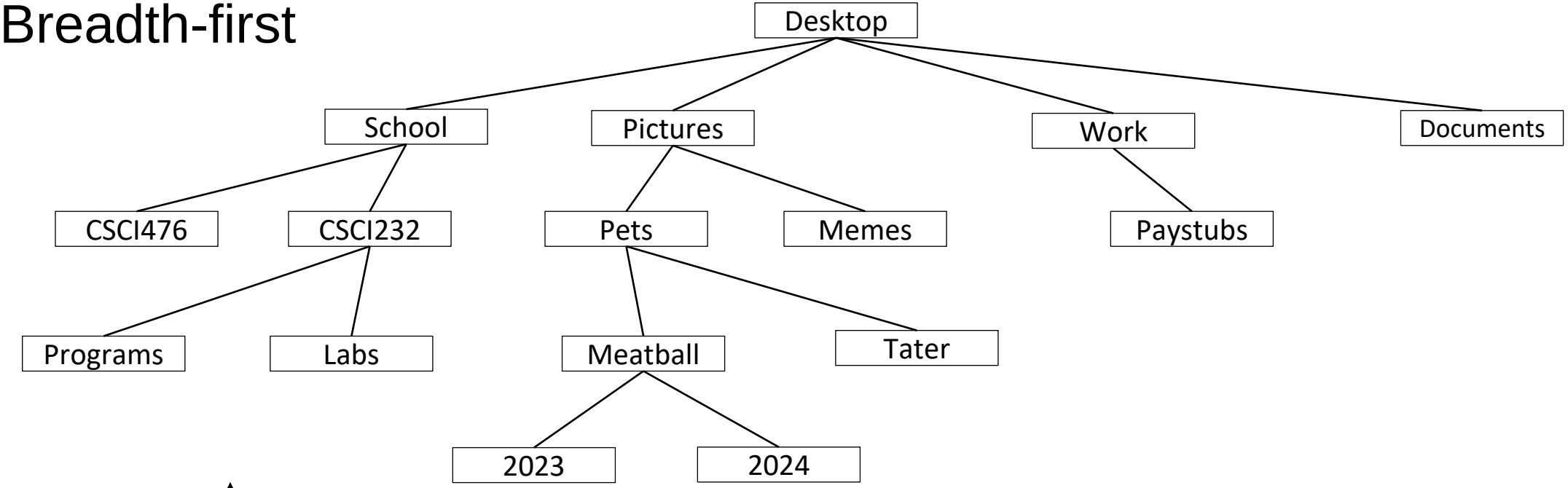


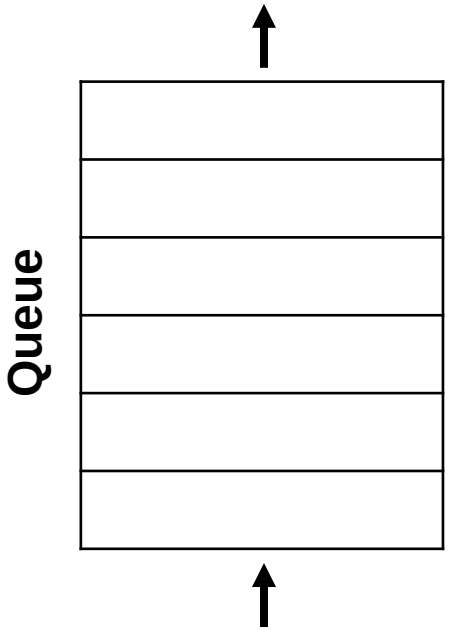


Every time we “visit” a node we:

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

Breadth-first

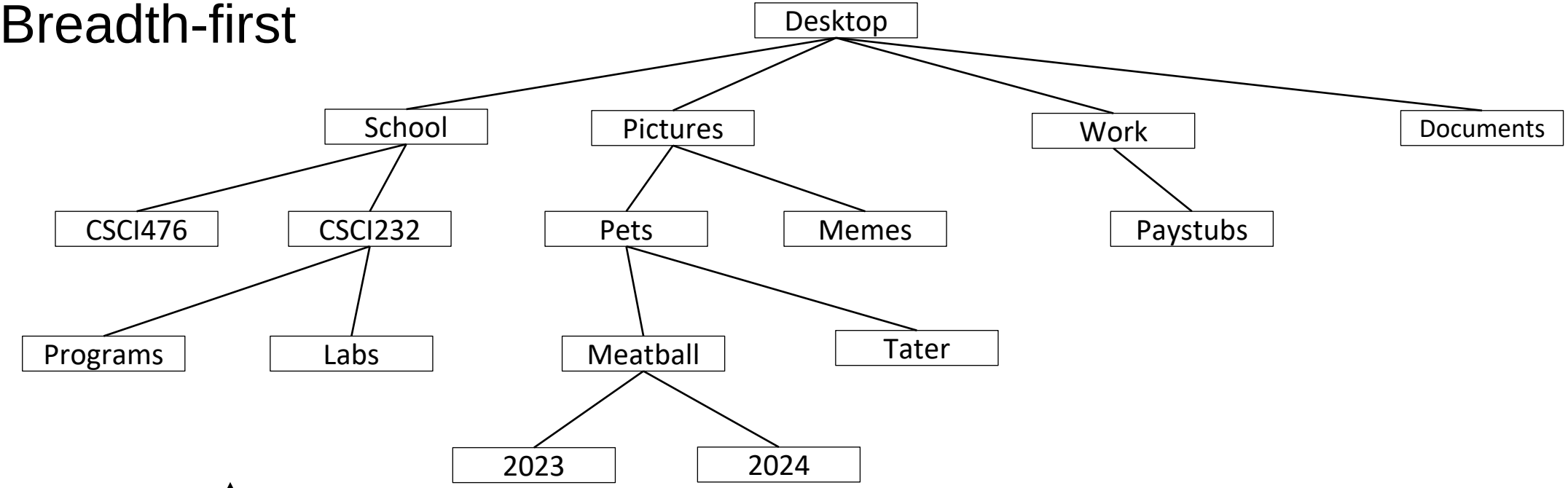


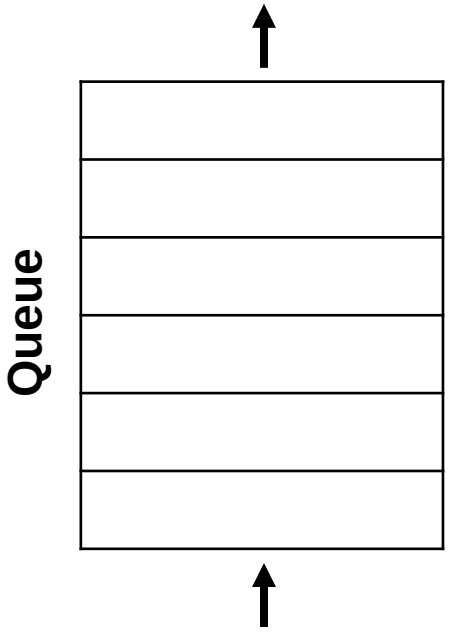


Every time we “visit” a node we:

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

Breadth-first

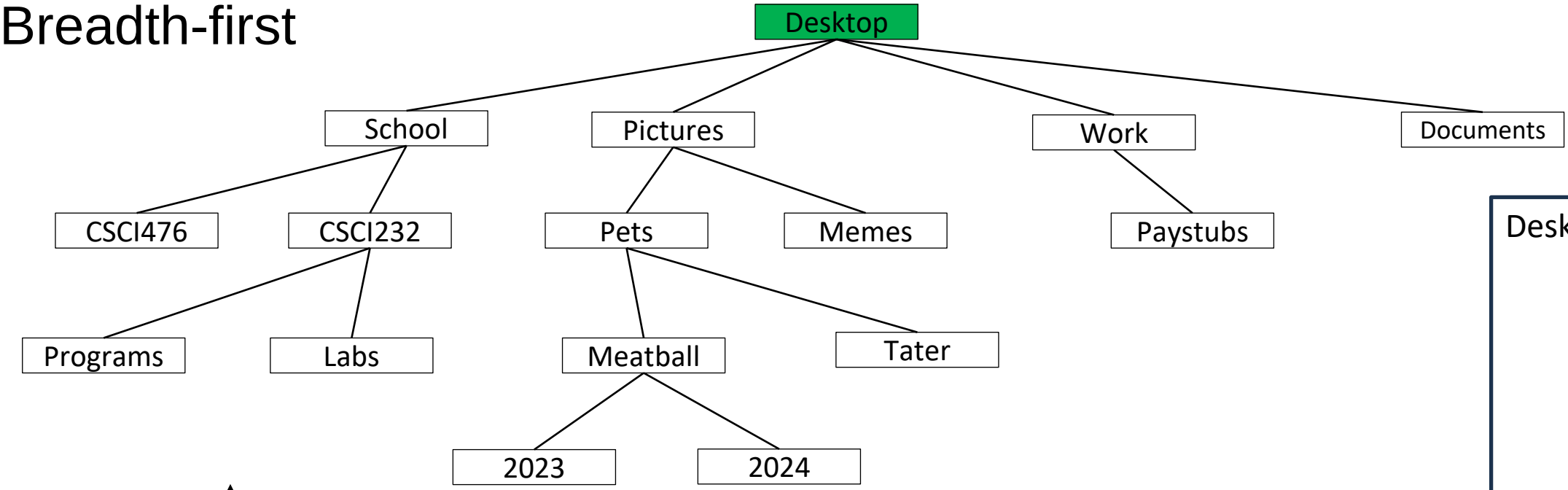




Every time we “visit” a node we:

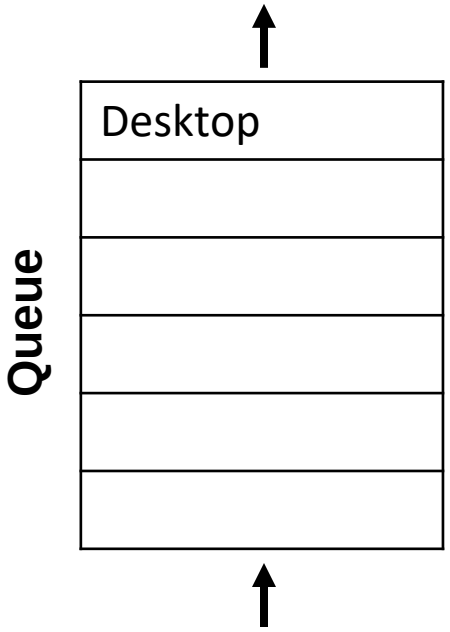
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

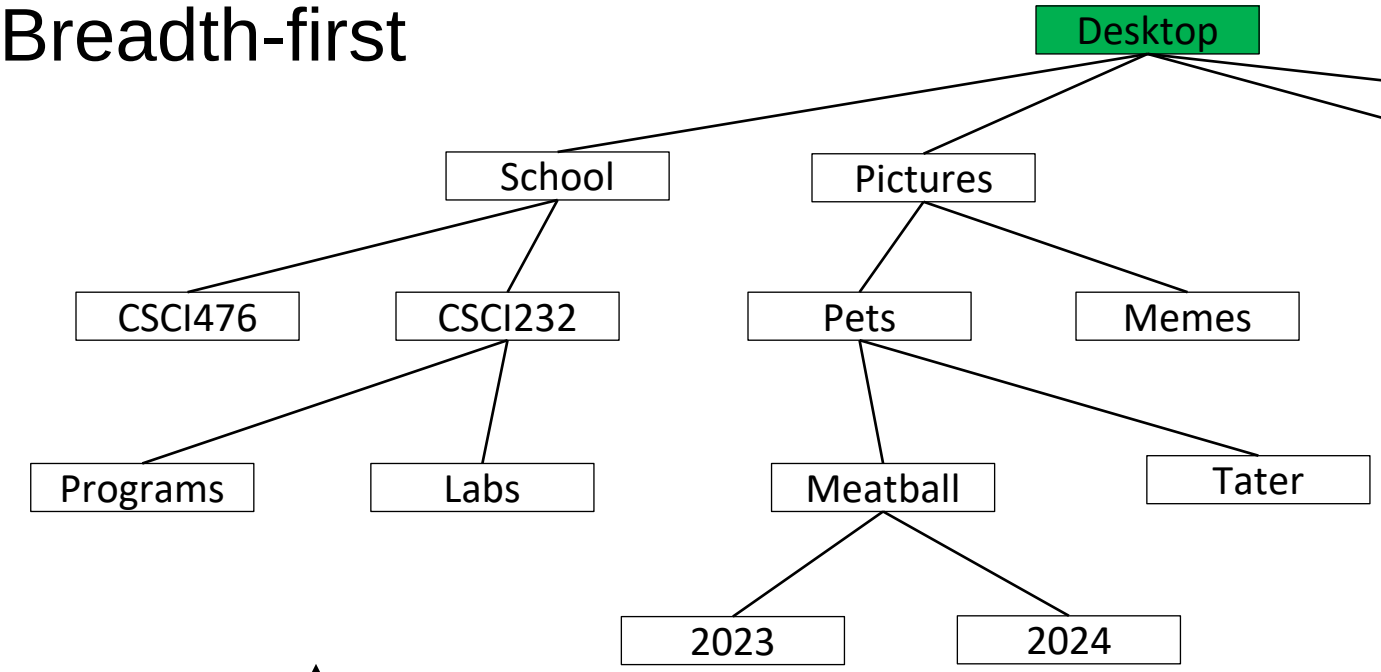
Desktop



Every time we "visit" a node we:

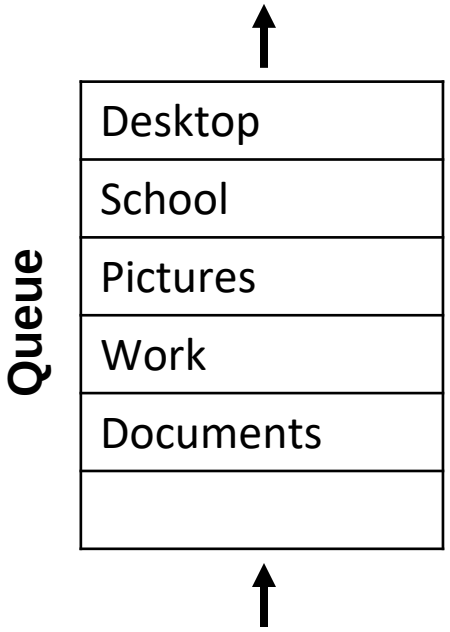
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

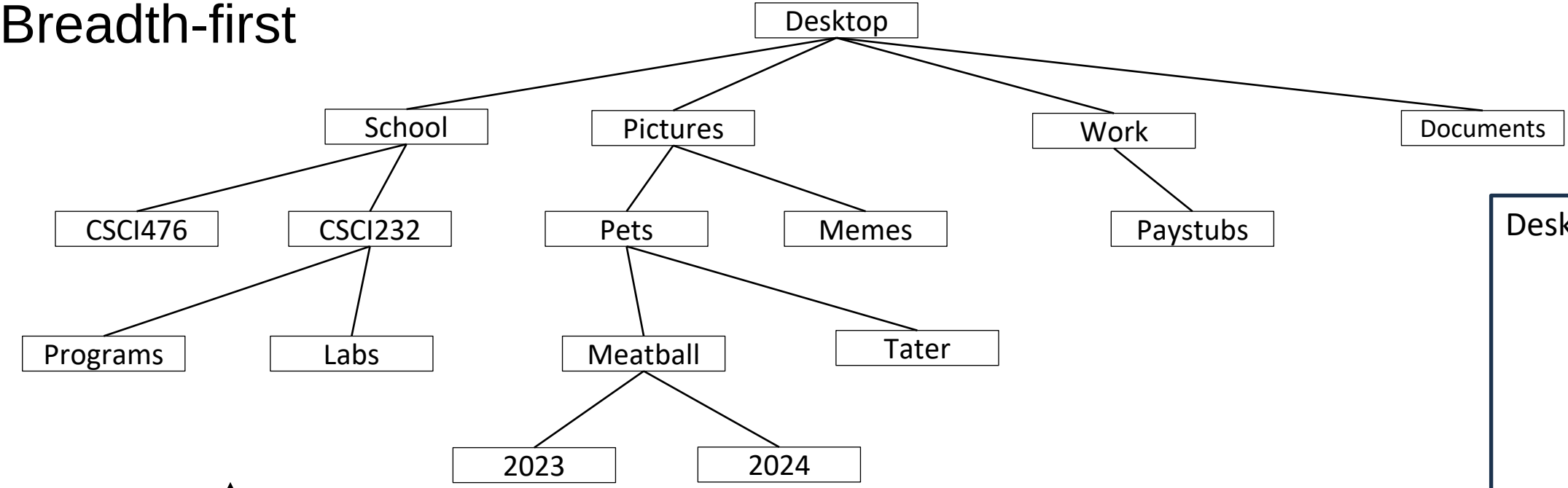
Desktop



Every time we "visit" a node we:

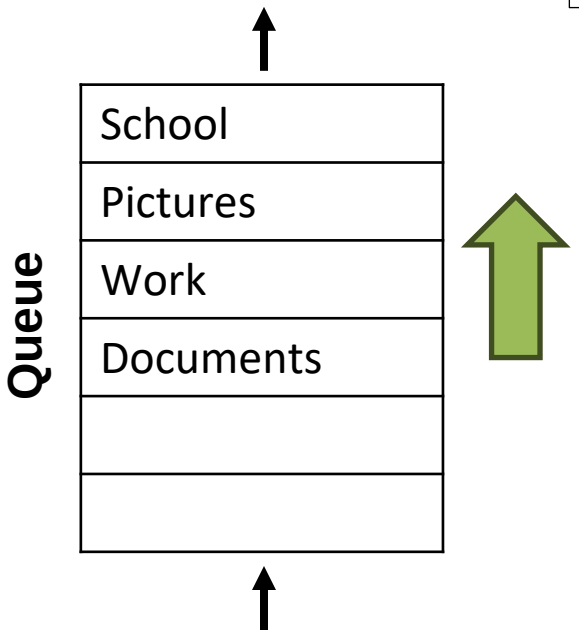
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

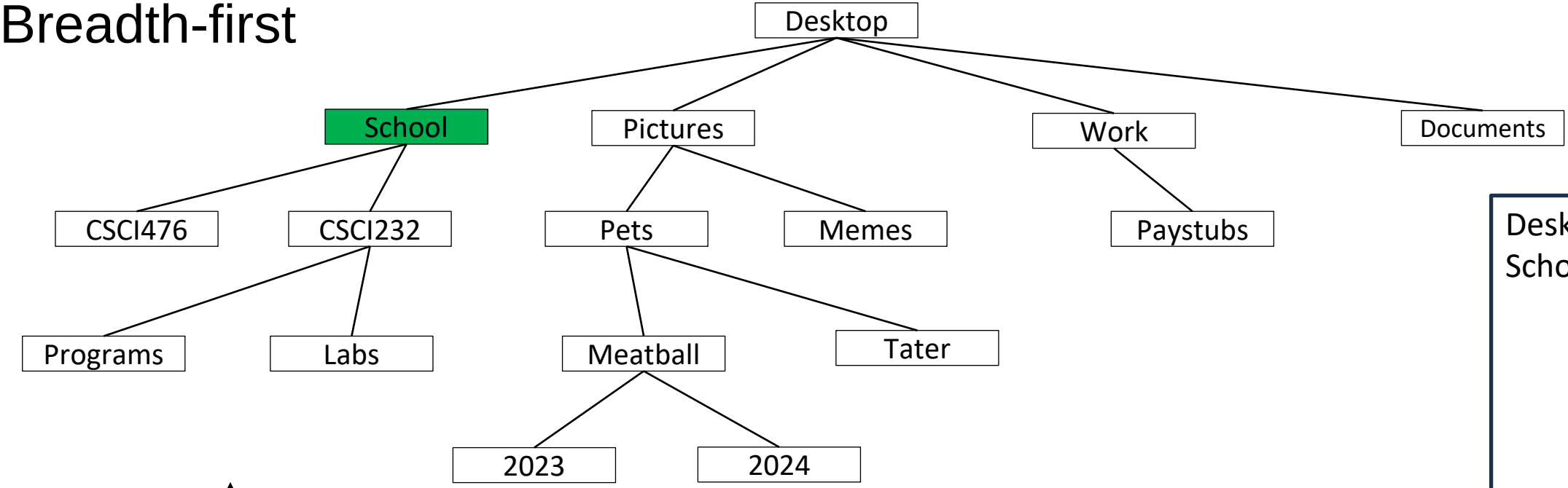
Desktop



Every time we "visit" a node we:

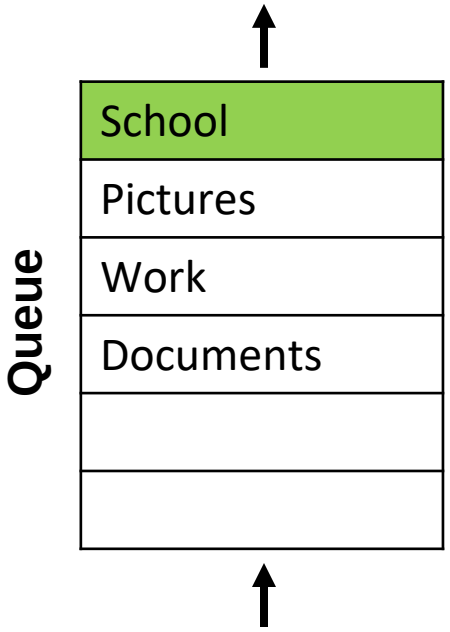
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

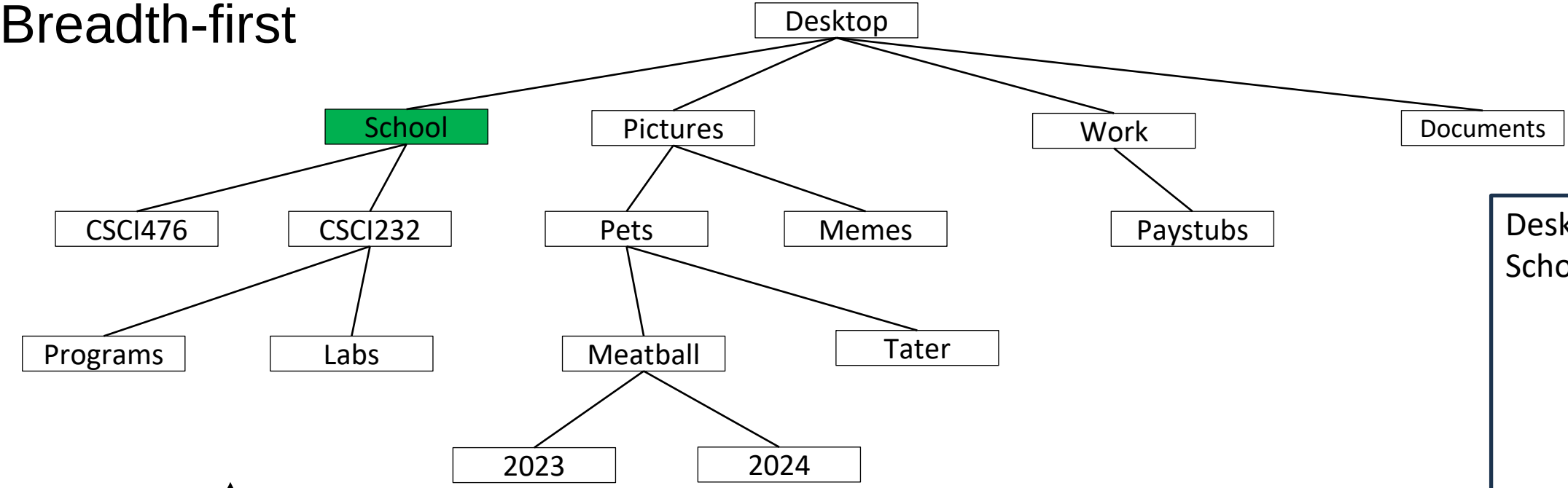
Desktop
School



Every time we "visit" a node we:

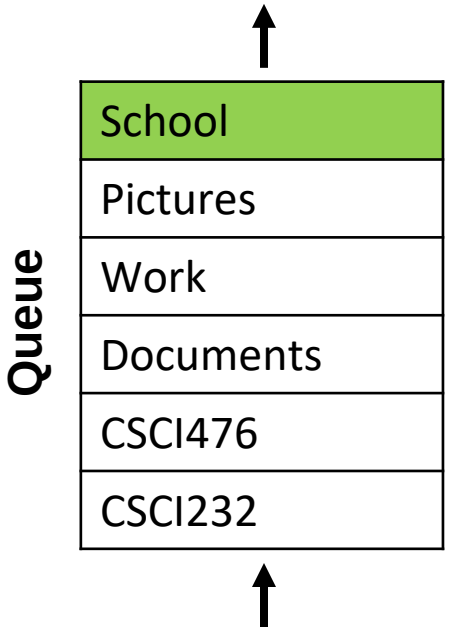
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

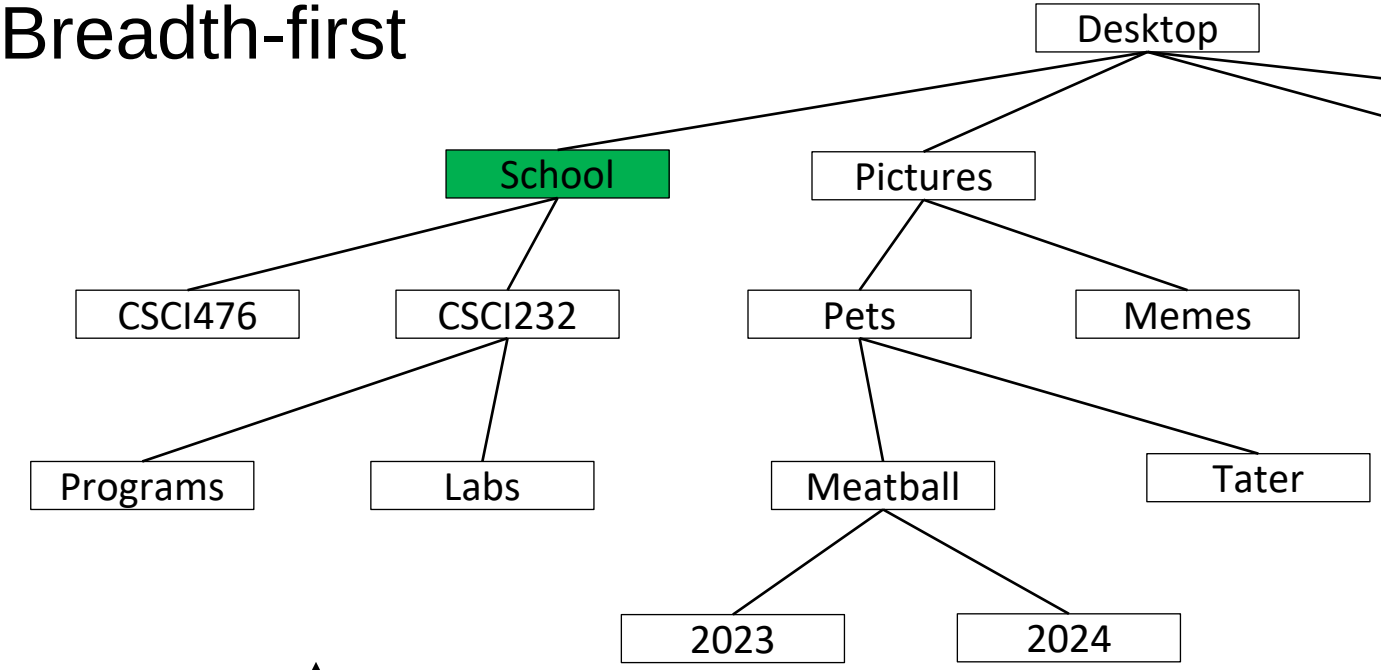
Desktop
School



Every time we "visit" a node we:

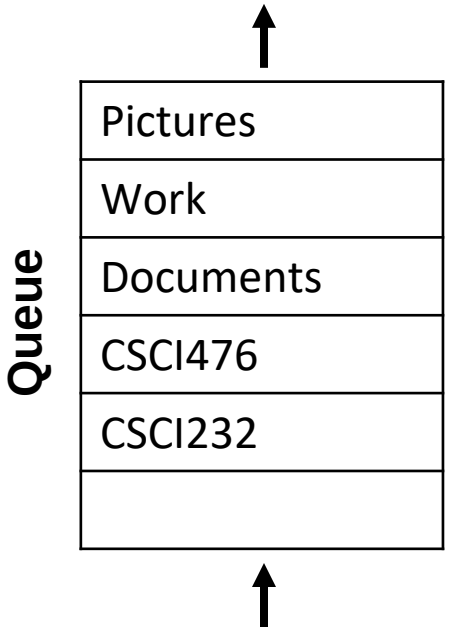
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

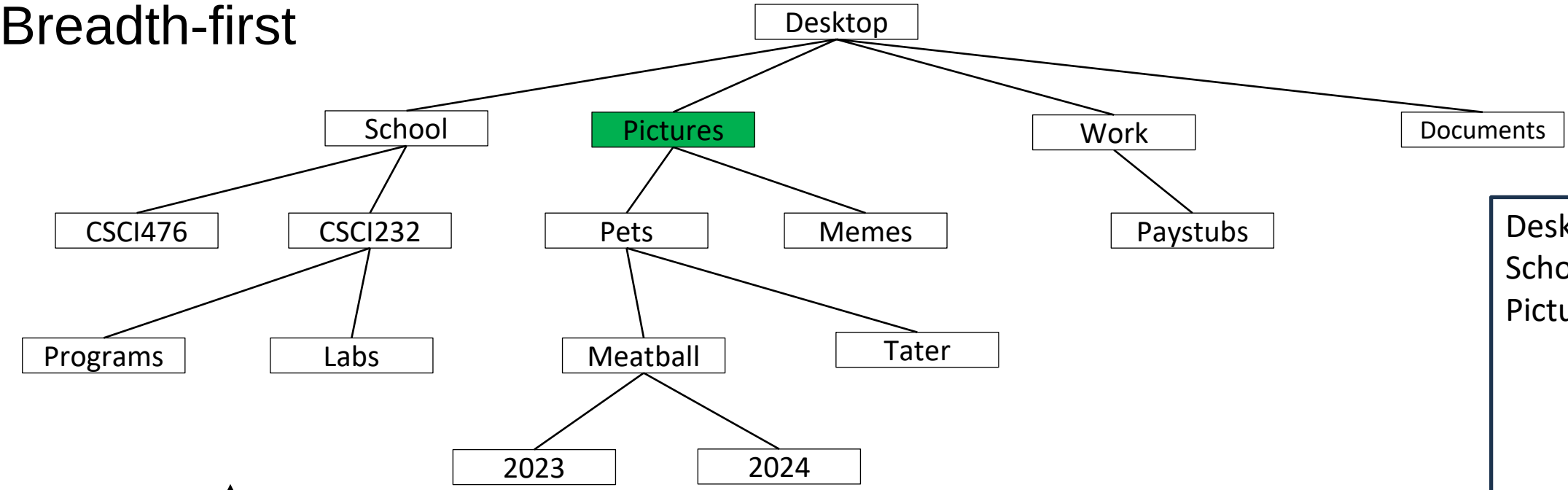
Desktop
School



Every time we "visit" a node we:

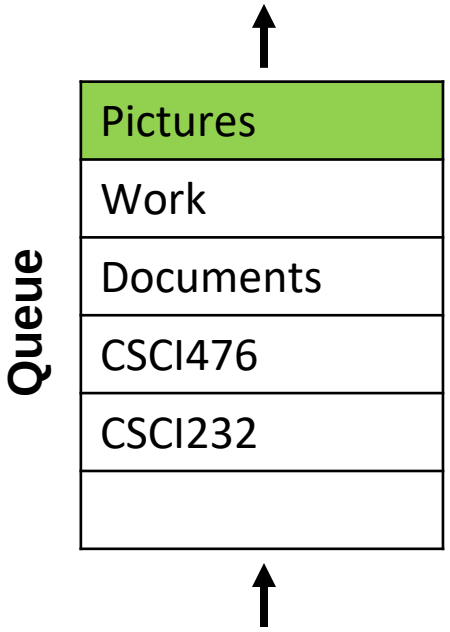
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

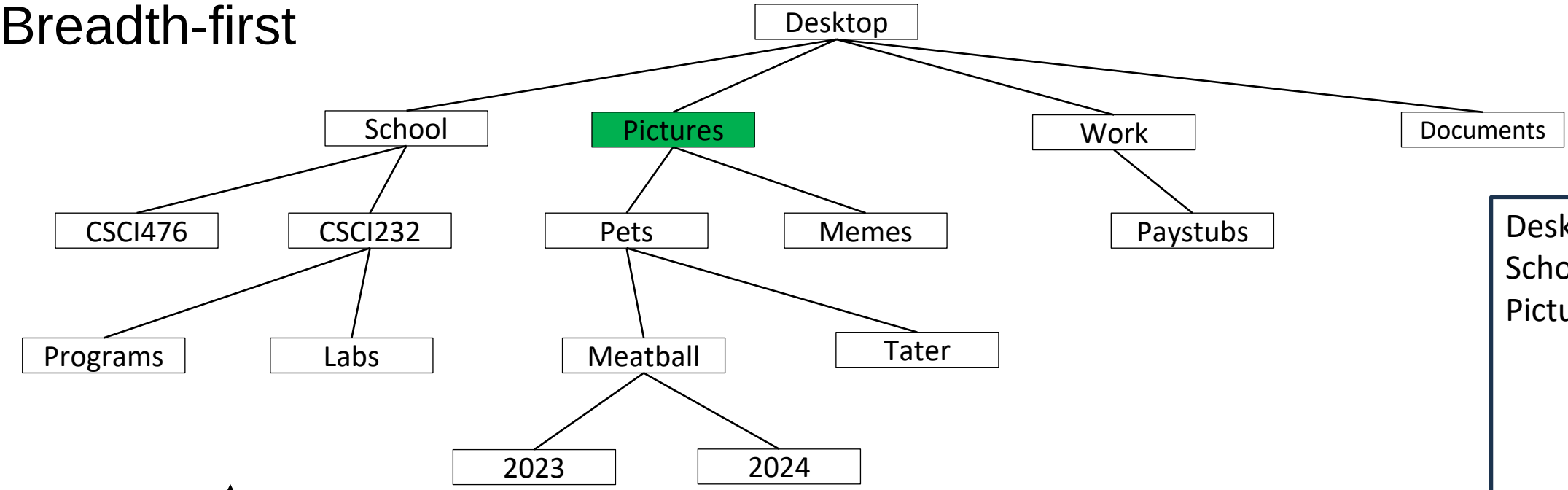
Desktop
School
Pictures



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 the queue

Breadth-first



Output

Desktop
School
Pictures

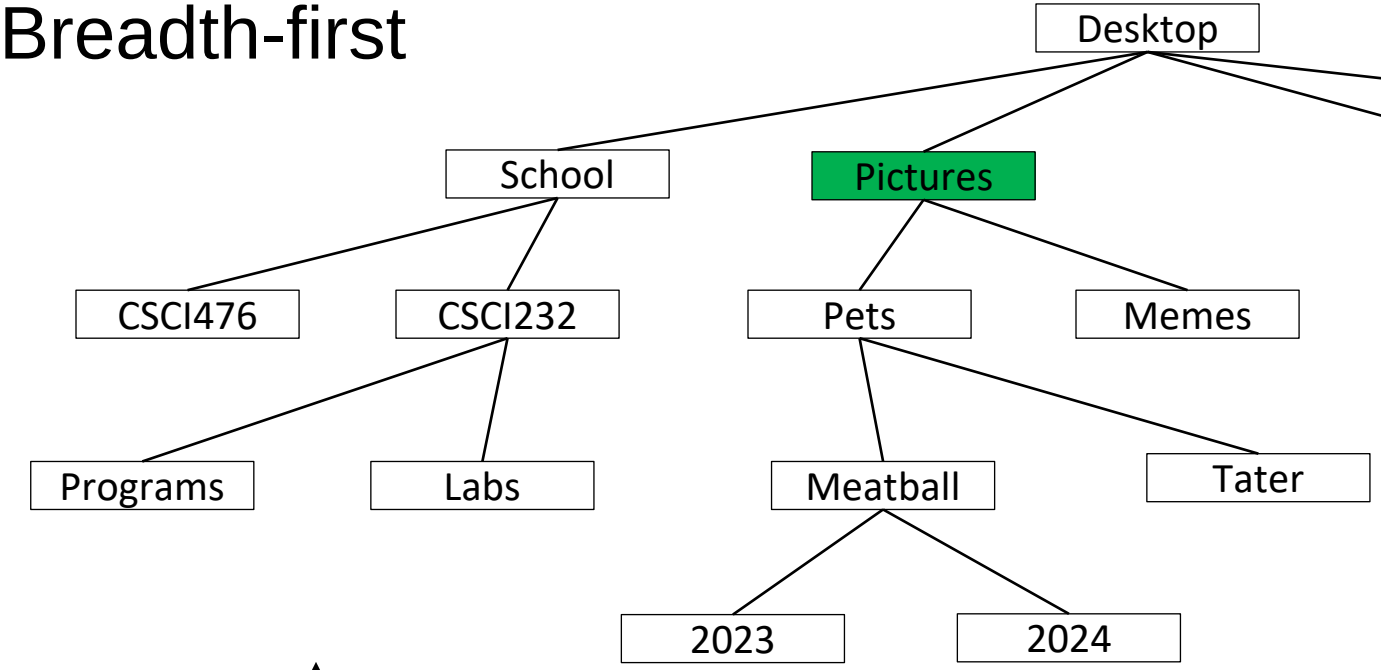
Queue

Pictures
Work
Documents
CSCI476
CSCI232
Pets
Memes

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 the queue

Breadth-first



Output

Desktop
School
Pictures

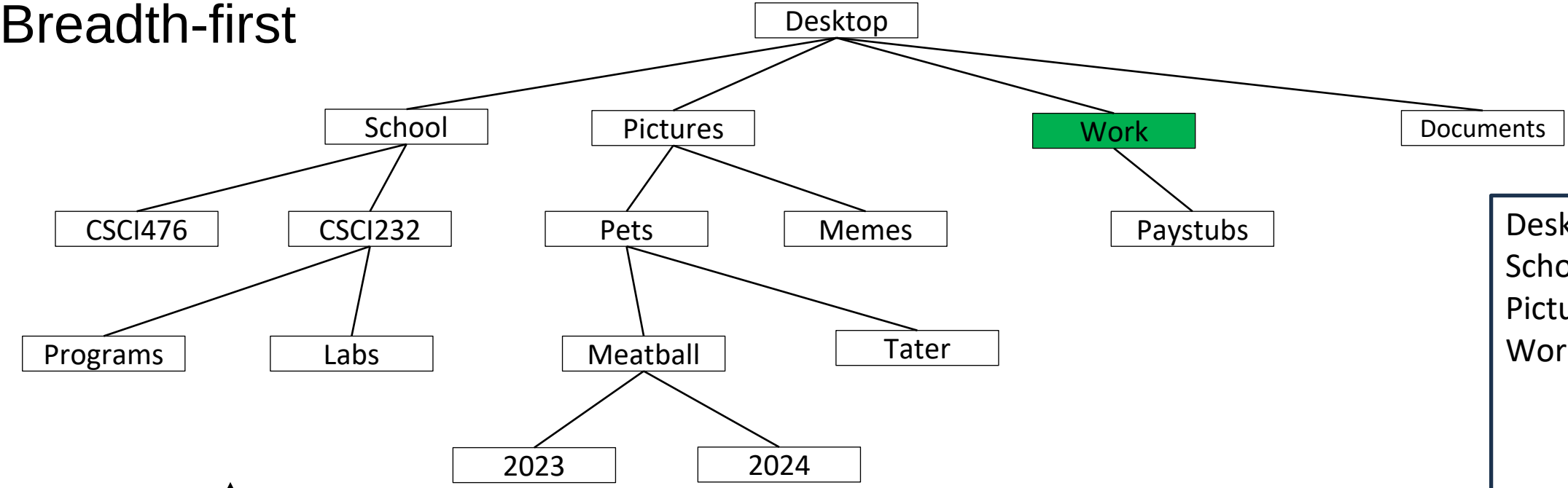
Queue

Work
Documents
CSCI476
CSCI232
Pets
Memes

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 the queue

Breadth-first



Output

Desktop
School
Pictures
Work

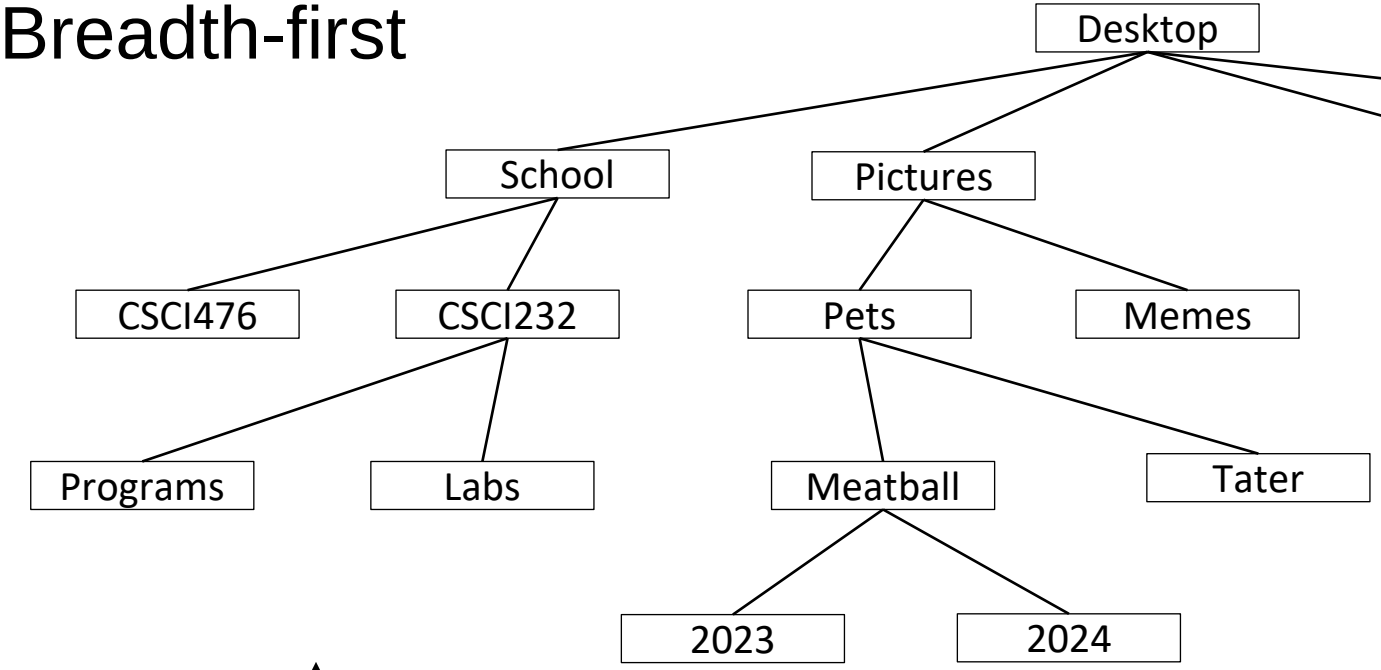
Queue

Work
Documents
CSCI476
CSCI232
Pets
Memes

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 the queue

Breadth-first



Output

Desktop
School
Pictures
Work

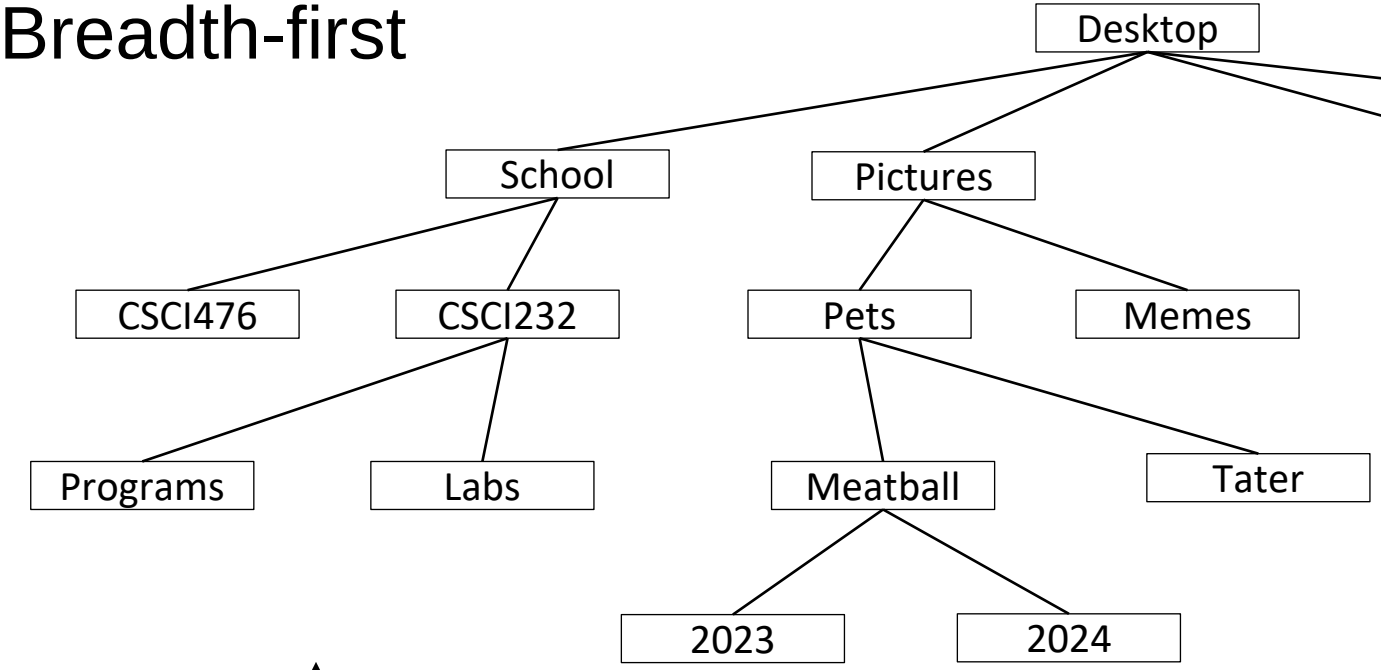
Queue

Work
Documents
CSCI476
CSCI232
Pets
Memes
Paystubs

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 the queue

Breadth-first



Output

Desktop
School
Pictures
Work

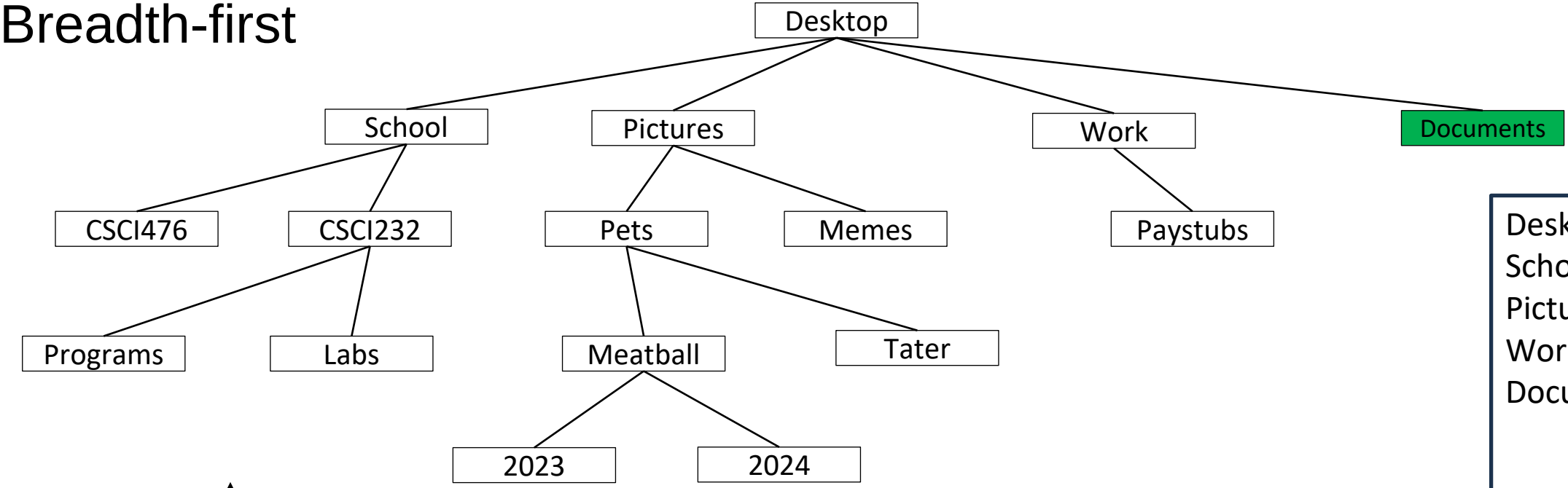
Queue

Documents
CSCI476
CSCI232
Pets
Memes
Paystubs

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 the queue

Breadth-first



Output

Desktop
School
Pictures
Work
Documents

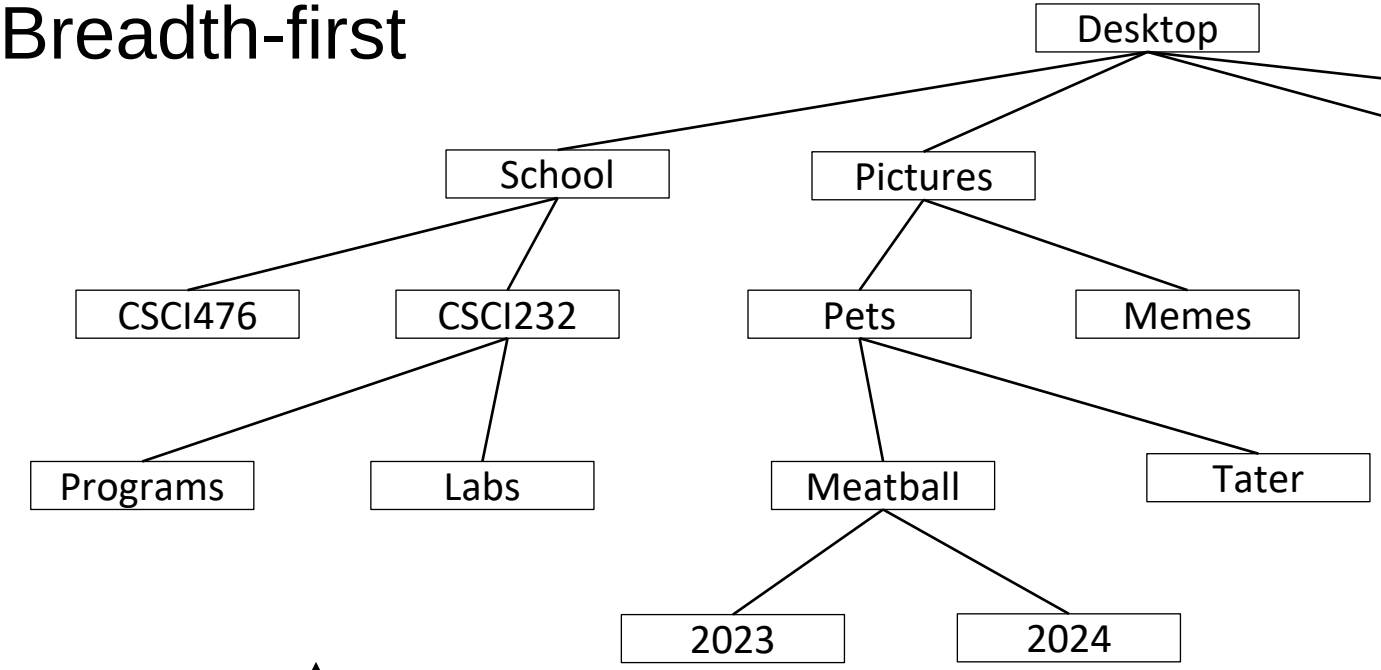
Queue

Documents
CSCI476
CSCI232
Pets
Memes
Paystubs

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 the queue

Breadth-first



Output

Desktop
School
Pictures
Work
Documents

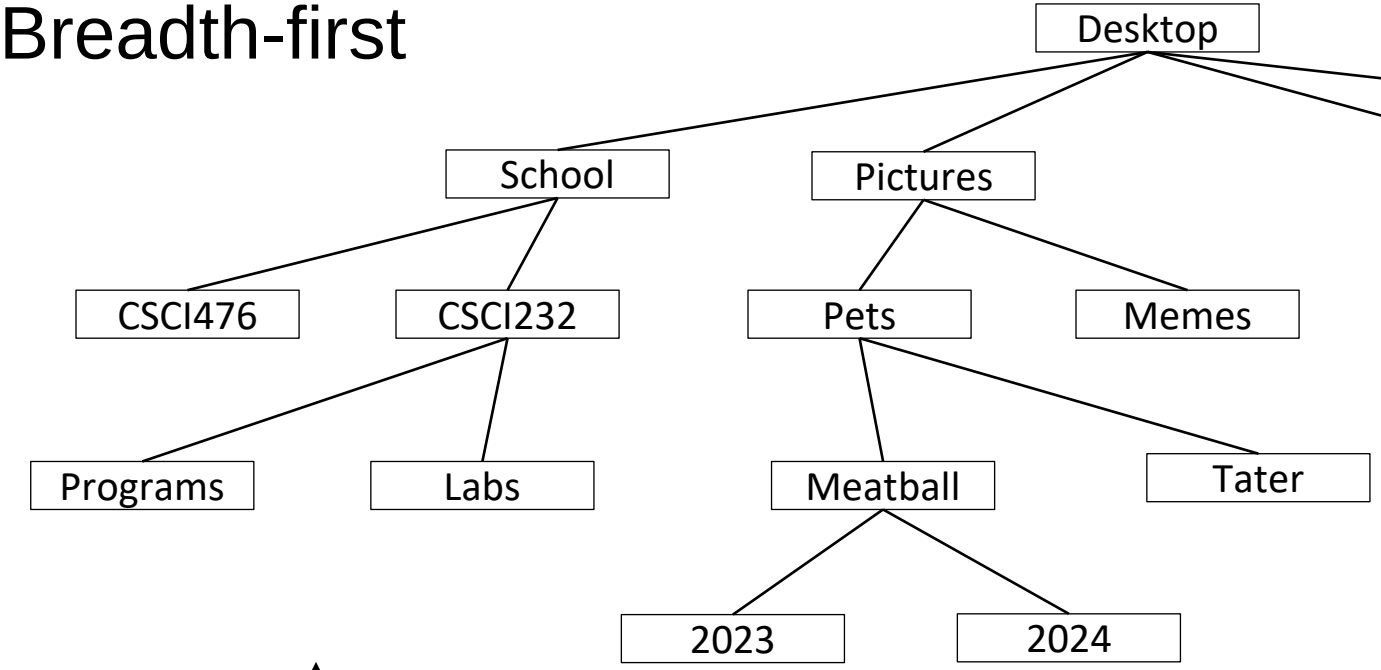
Queue

Documents
CSCI476
CSCI232
Pets
Memes
Paystubs

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 the queue

Breadth-first



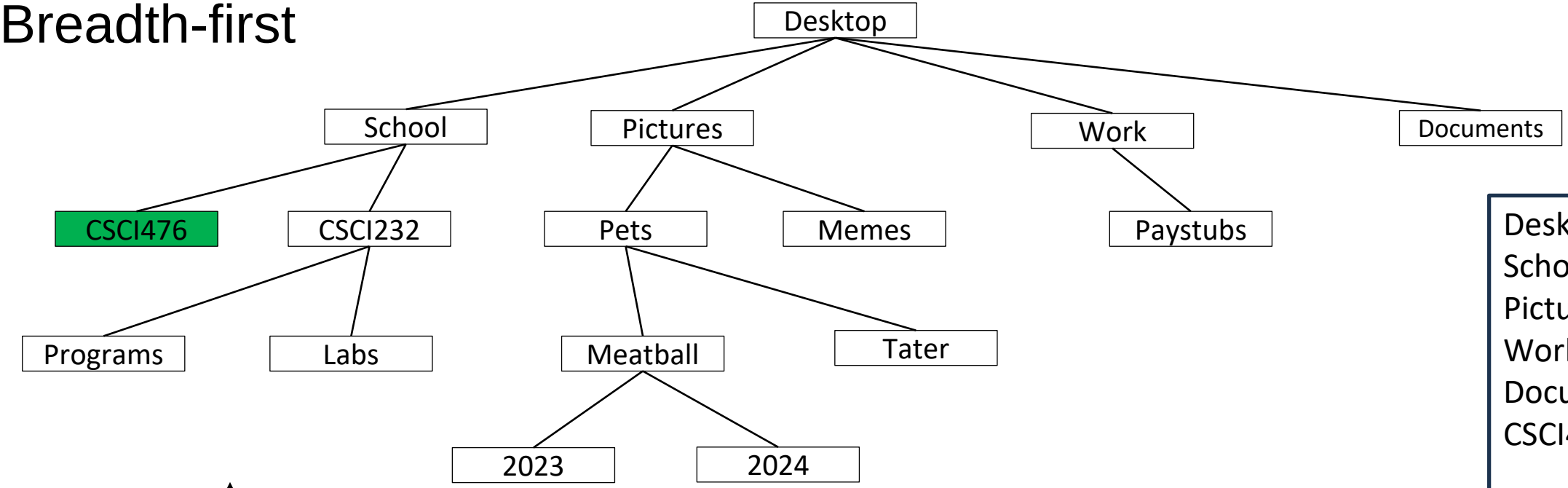
Output

Desktop
School
Pictures
Work
Documents

Every time we "visit" a node we:

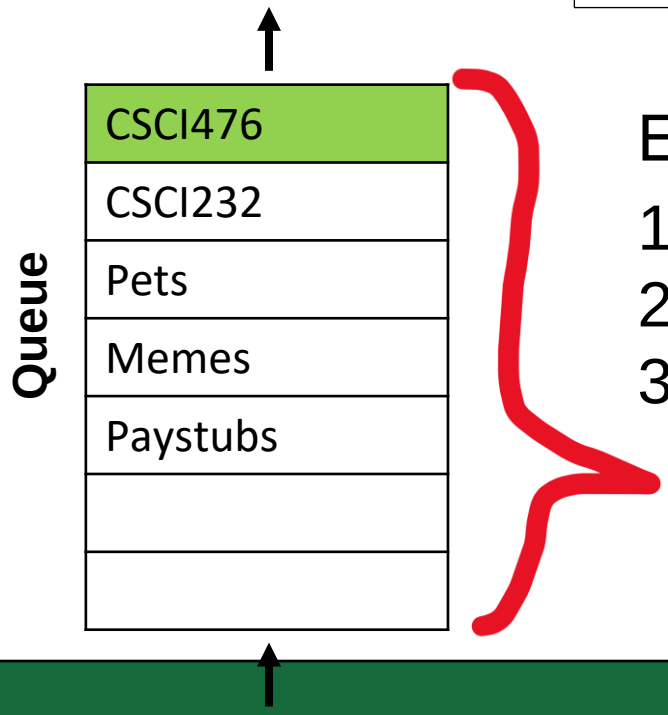
1. Execute the action (e.g., print, compare, …)
2. Add all of its children to the queue
3. Remove visited node from the queue

Breadth-first



Output

Desktop
School
Pictures
Work
Documents
CSCI476



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 the queue

Because of step 2, we can visit nodes in the proper order in layer two !

Breadth-first

```
public void breadthFirst(){




}
```

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 the queue

Breadth-first

```
public void breadthFirst(){
        Queue<???> = new ???<???>();









}
```

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 the queue

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();









}
```

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 the queue

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();









}
```

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 the queue

Where do we start at ?

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)









        }
}
```

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 the queue

Where do we start at ? **THE ROOT**

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)









        }
}
```

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 the queue

How long to loop for?

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){









                }

        }
}
```

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 the queue

How long to loop for? As long as our queue as unvisited nodes inside of it

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){


                        Node node = queue.remove()









                }

        }
}
```

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 the queue

In order to execute the Node action, I need to retrieve the next node. However, I am going to retrieve and remove it in the same step

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){


                        Node node = queue.remove()

                        System.out.println(node.get???)









                }

        }
}
```

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 the queue

In order to execute the Node action, I need to retrieve the next node. However, I am going to retrieve and remove it in the same step

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){


                        Node node = queue.remove()

                        System.out.println(node.get???)

                        for(Node n: node.getChildren()){


                        }

                }

        }
}
```

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 the queue

Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){


                        Node node = queue.remove()

                        System.out.println(node.get???)

                        for(Node n: node.getChildren()){
                                queue.add(n);
                        }

                }

        }
}
```

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 the queue

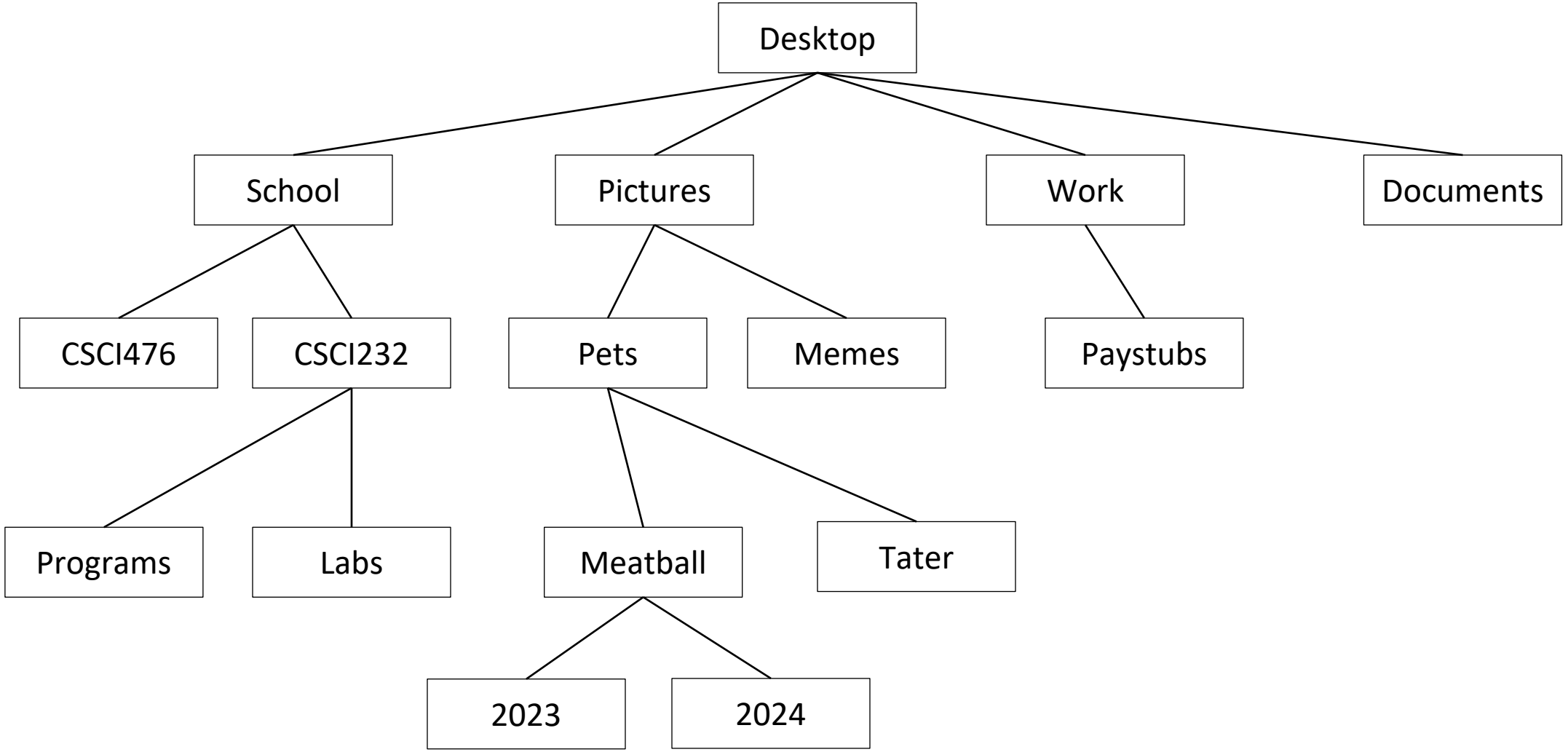
Breadth-first

```
public void breadthFirst(){
        Queue<Node> = new LinkedList<Node>();
        if( root != null){
                queue.add(root)
                while( !queue.isEmpty() ){


                        Node node = queue.remove()

                        System.out.println(node.get???)

                        for(Node n: node.getChildren()){
                                queue.add(n);
                        }

                }

        }
}
```

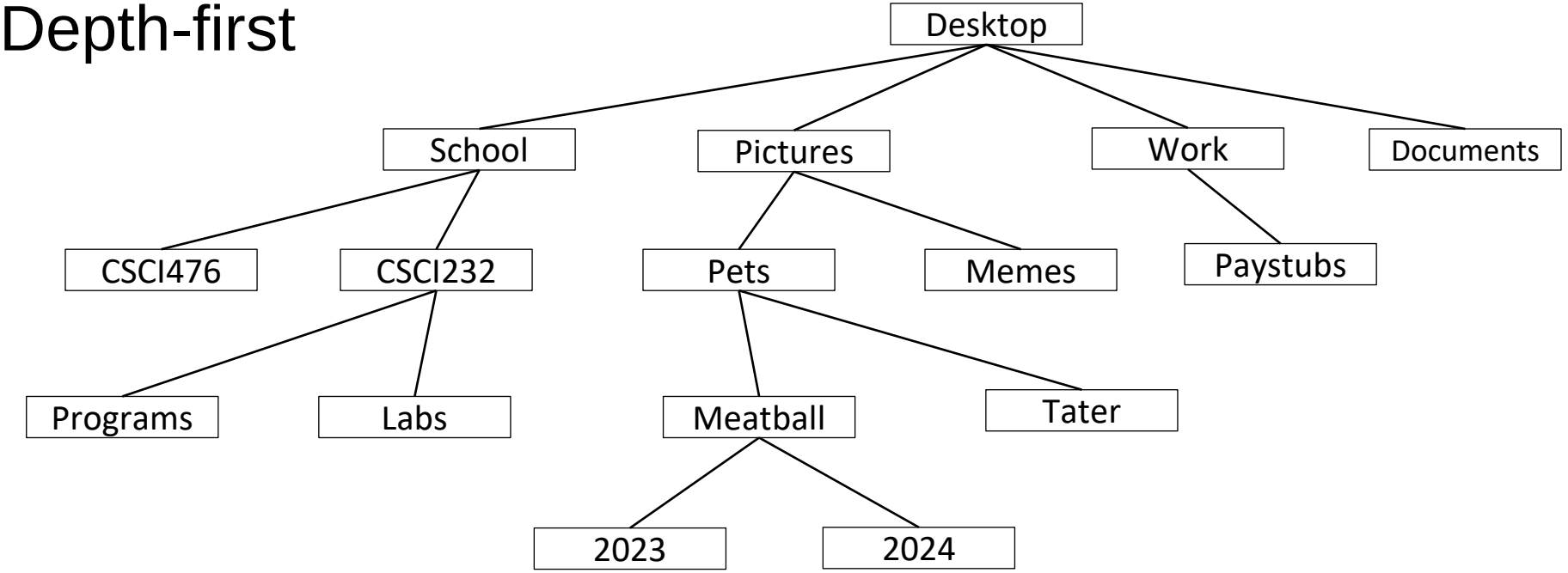
Every time we "visit" a node we:

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

Let's code this!

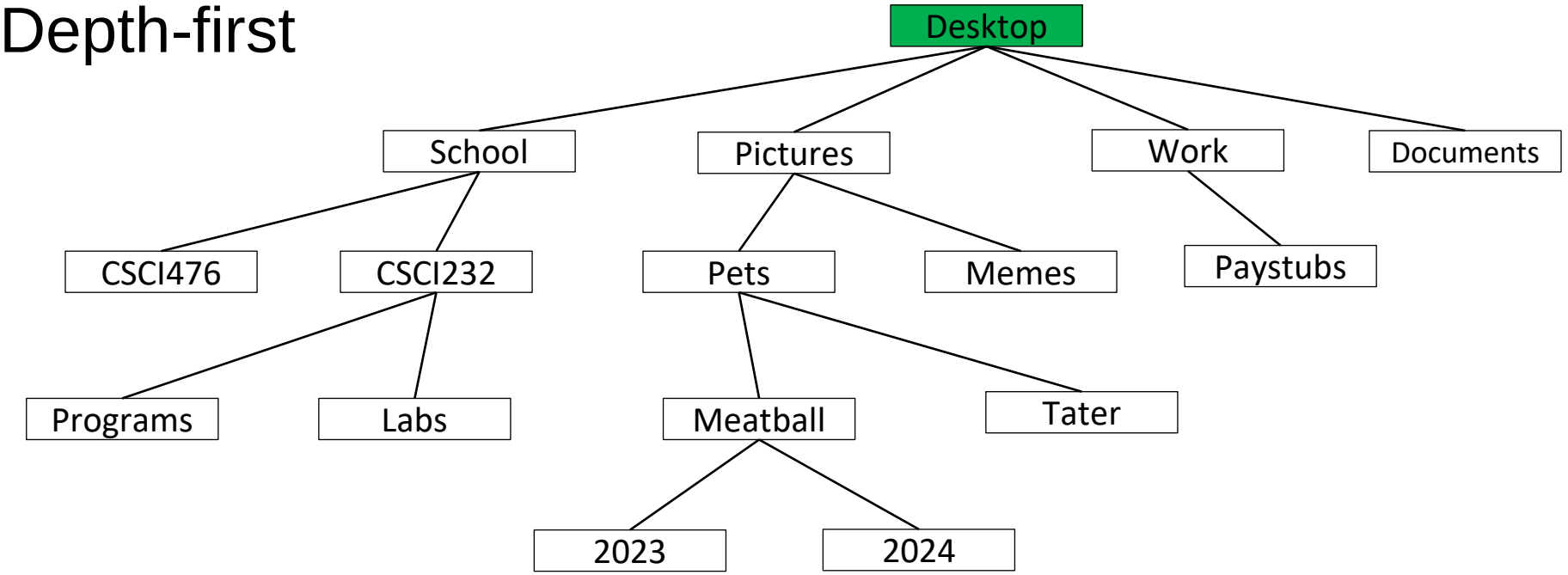

Desktop
School
Pictures
Work
Documents
CSCI476
CSCI232
Pets
Memes
Paystubs
Programs
Labs
Meatball
Tater
2023
2024

Depth-first



Output

Depth-first

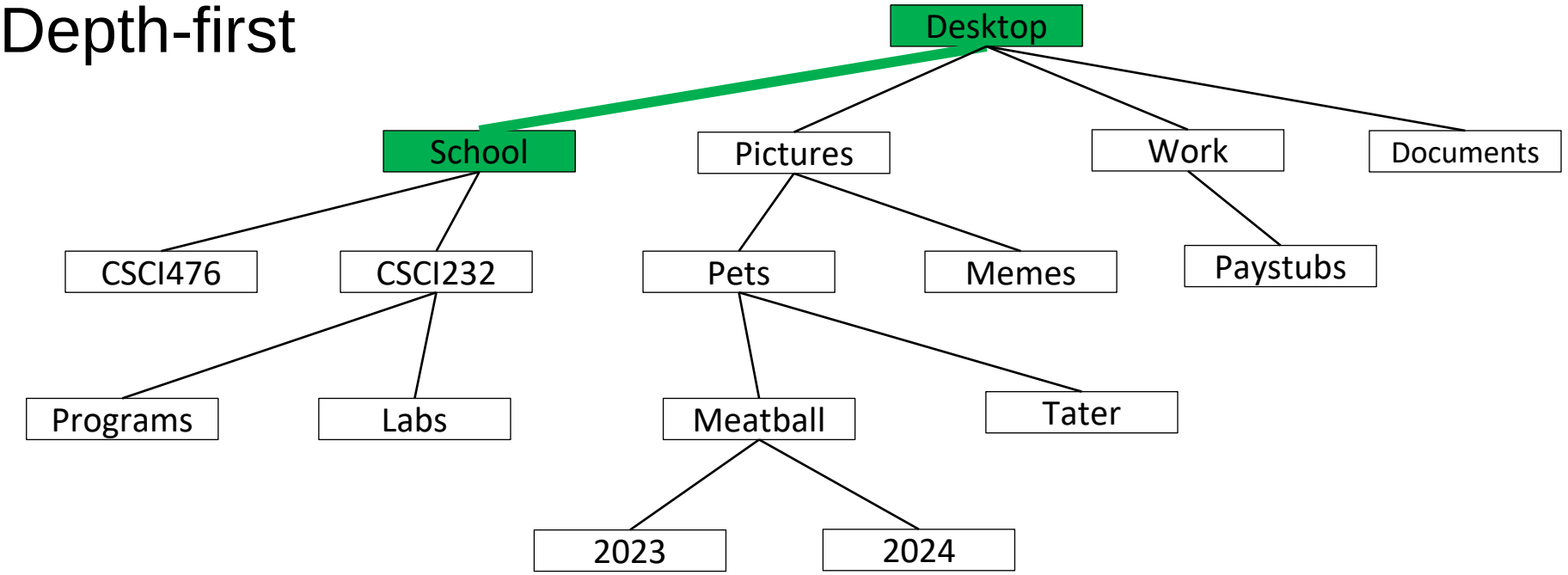


Output

Desktop

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

Depth-first

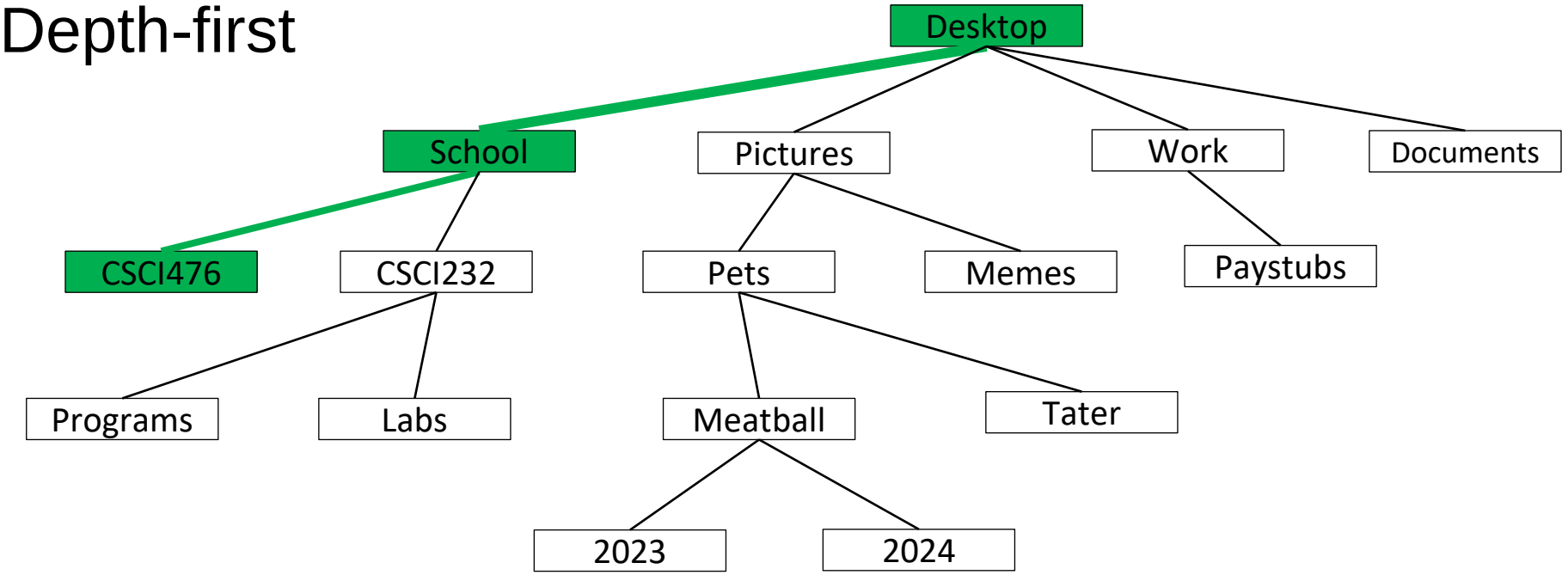


Output

Desktop
School

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

Depth-first

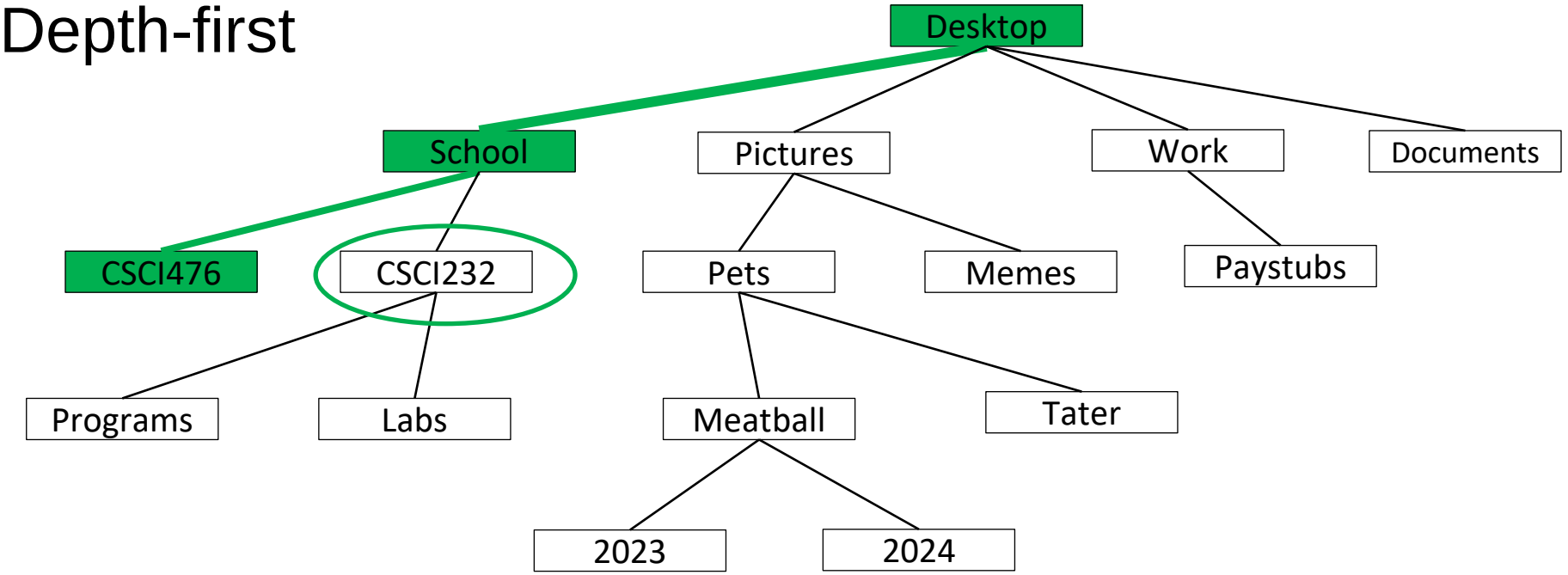


Output

Desktop
School
CSCI476

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

Depth-first

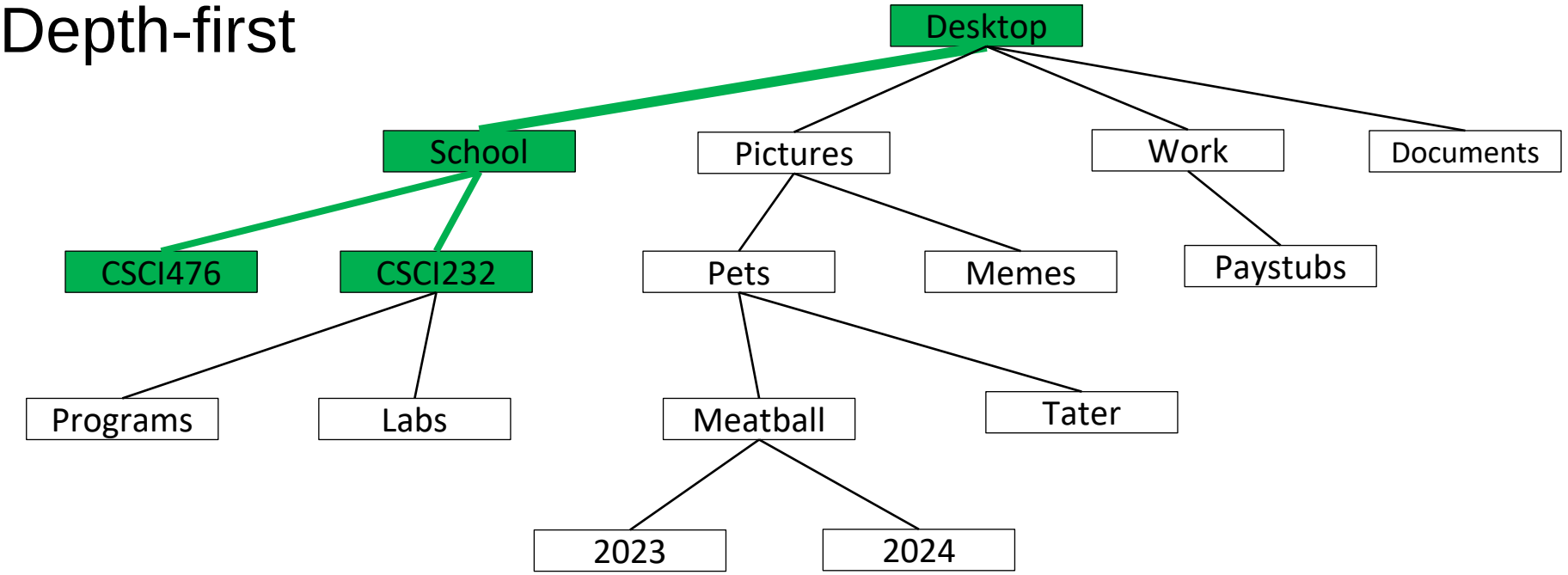


Output

Desktop
School
CSCI476

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

Depth-first

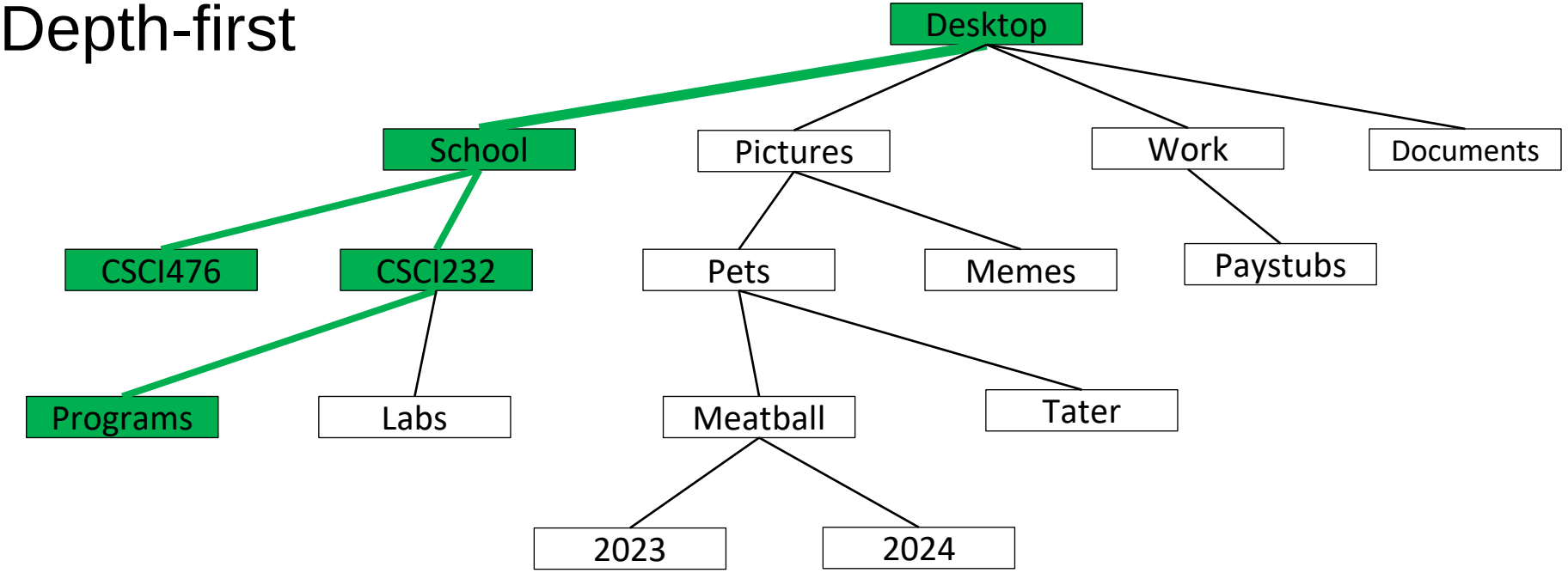


Output

Desktop
School
CSCI476
CSCI 232

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

Depth-first

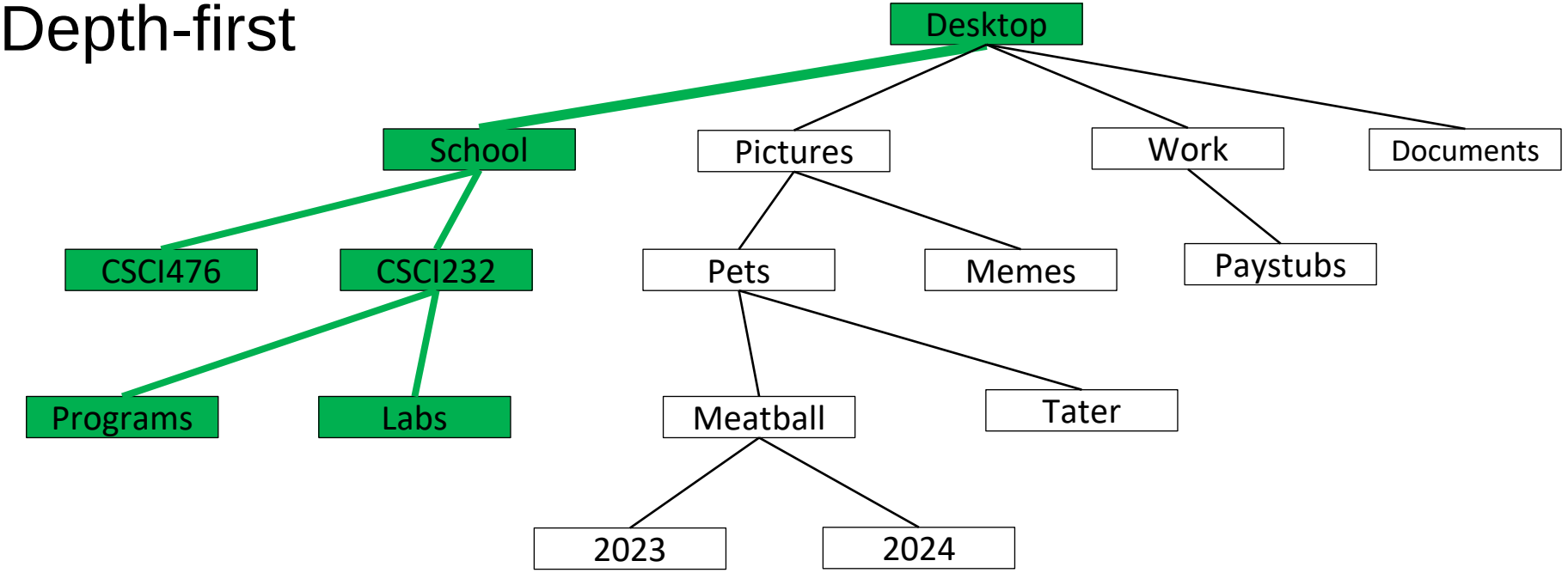


Output

Desktop
School
CSCI476
CSCI 232
Programs

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

Depth-first

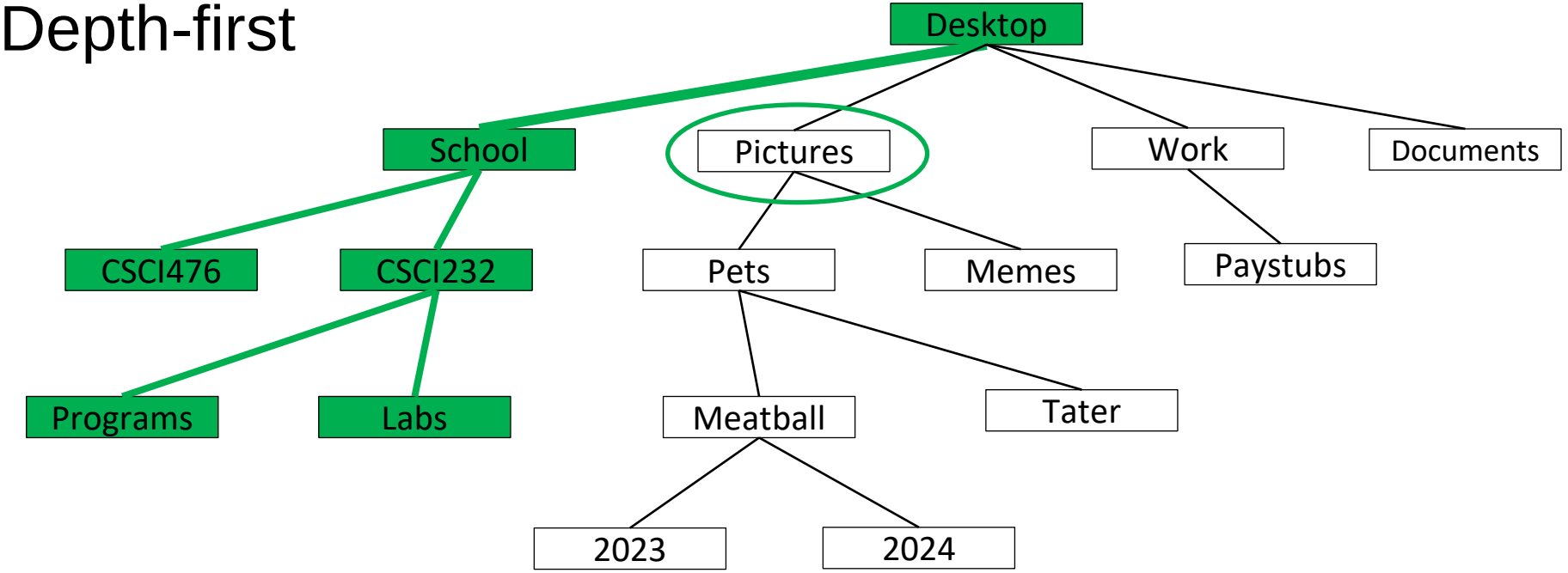


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs

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

Depth-first

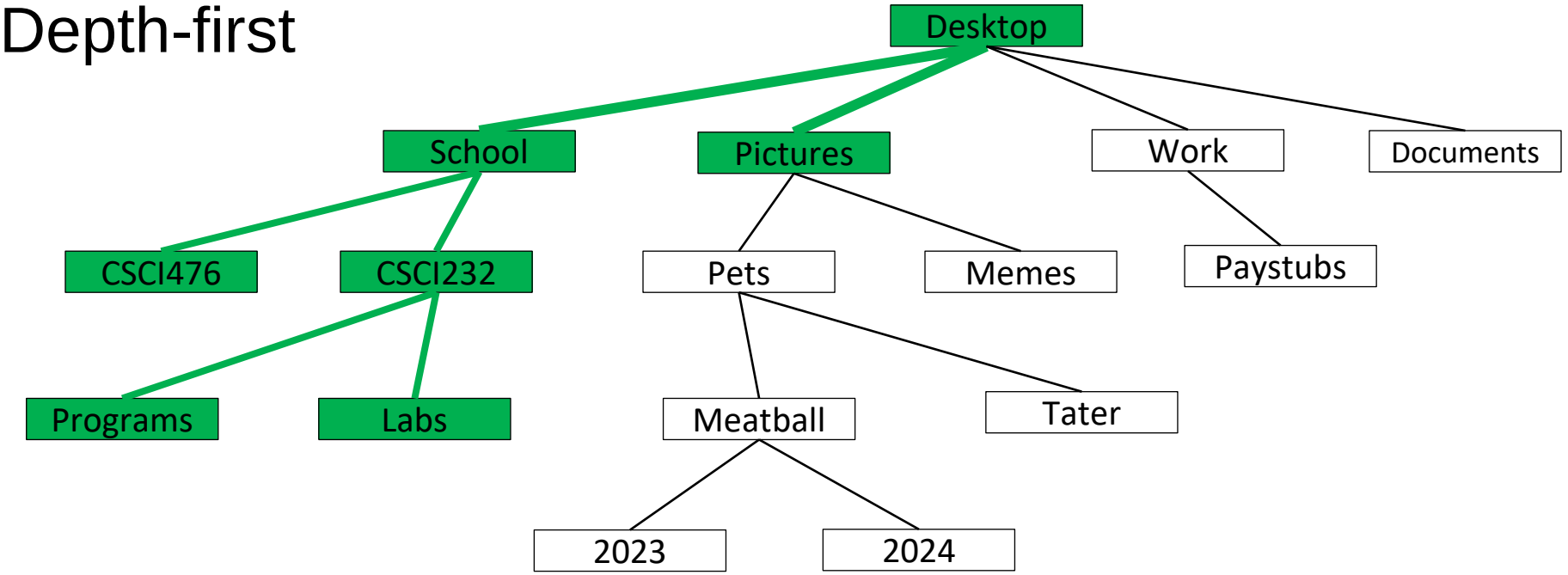


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs

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

Depth-first

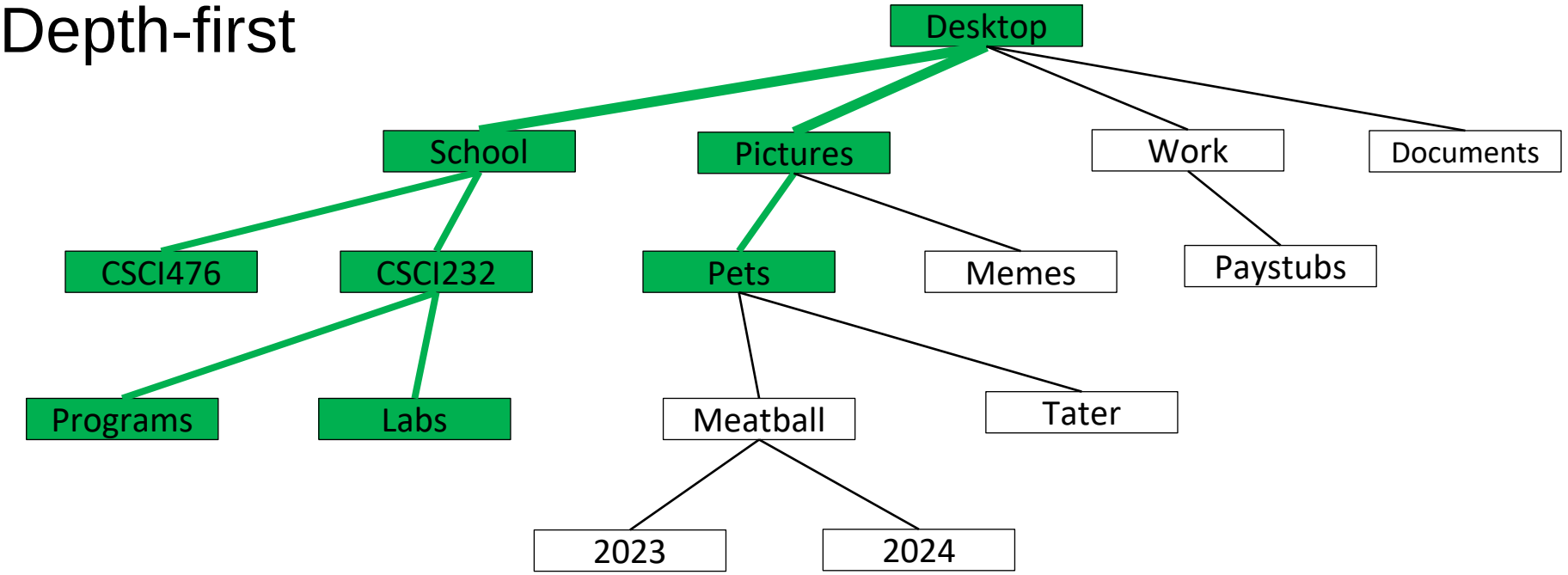


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures

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

Depth-first

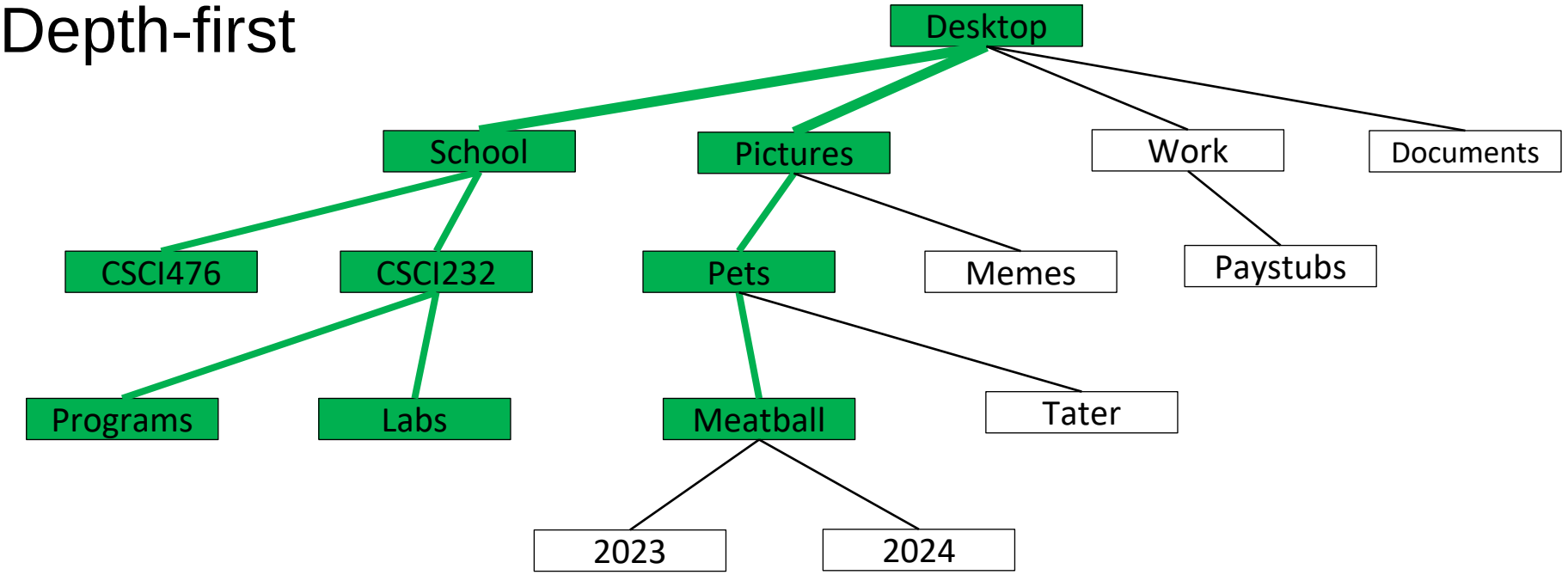


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets

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

Depth-first

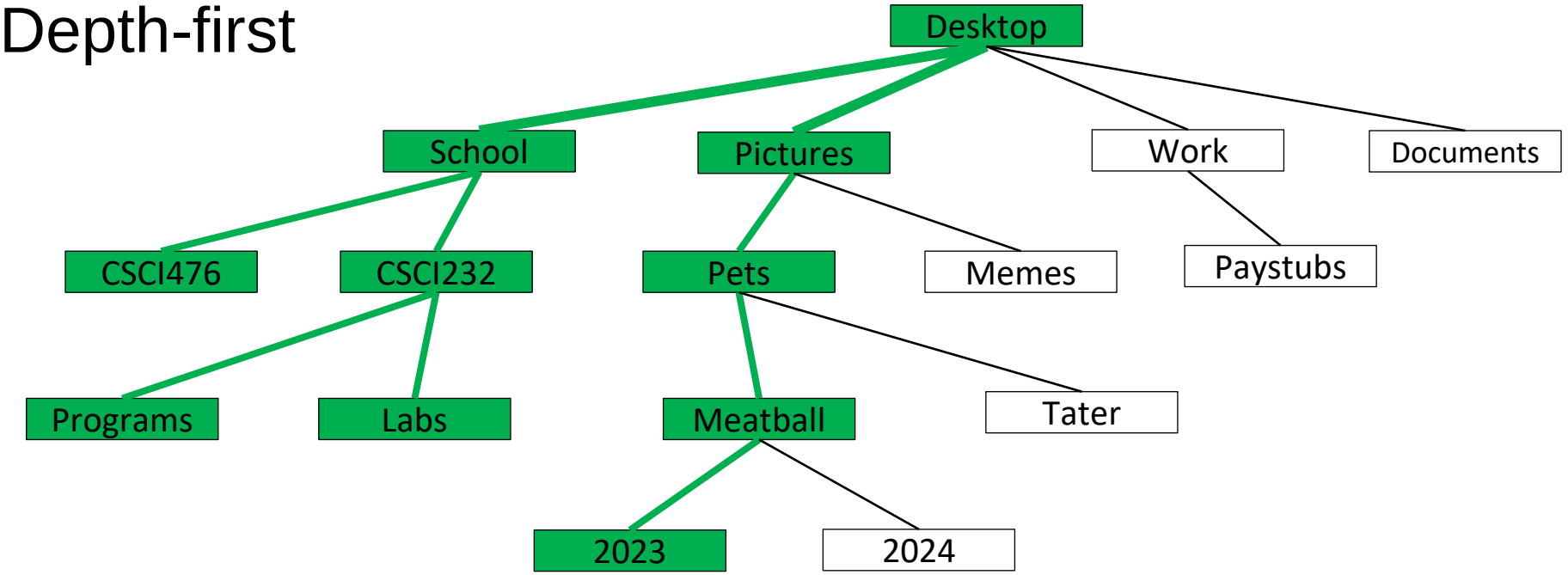


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball

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

Depth-first

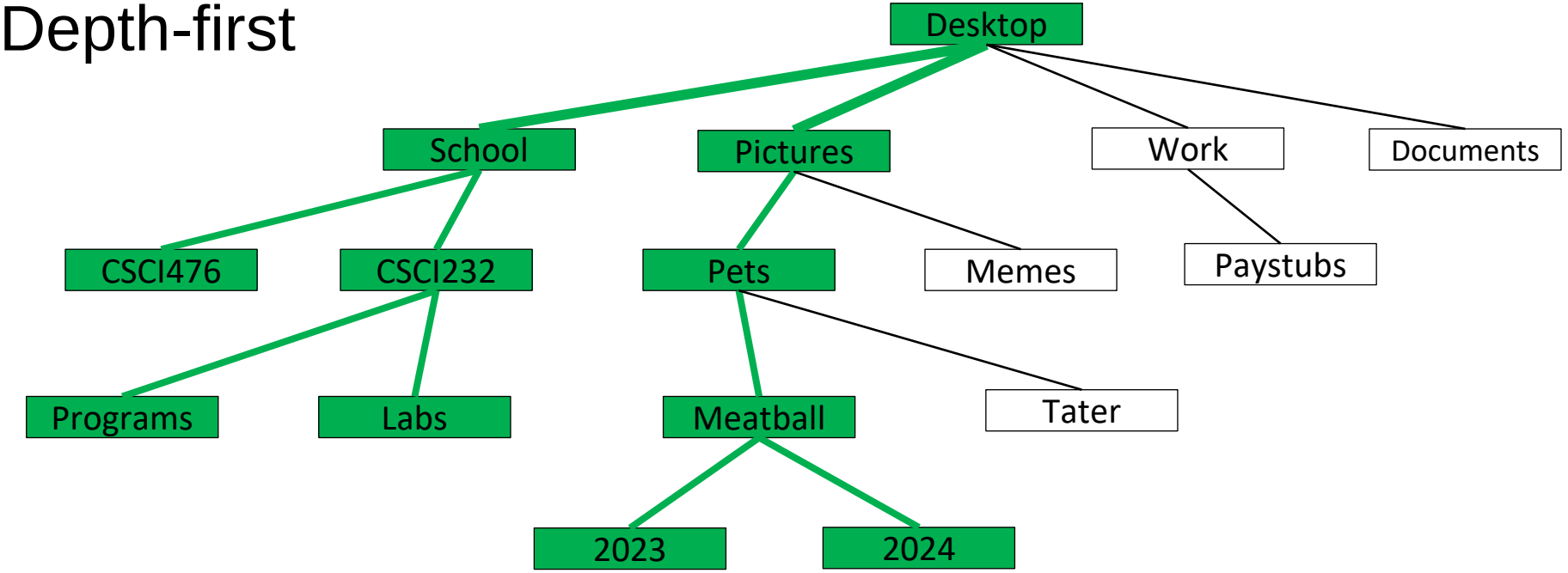


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023

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

Depth-first

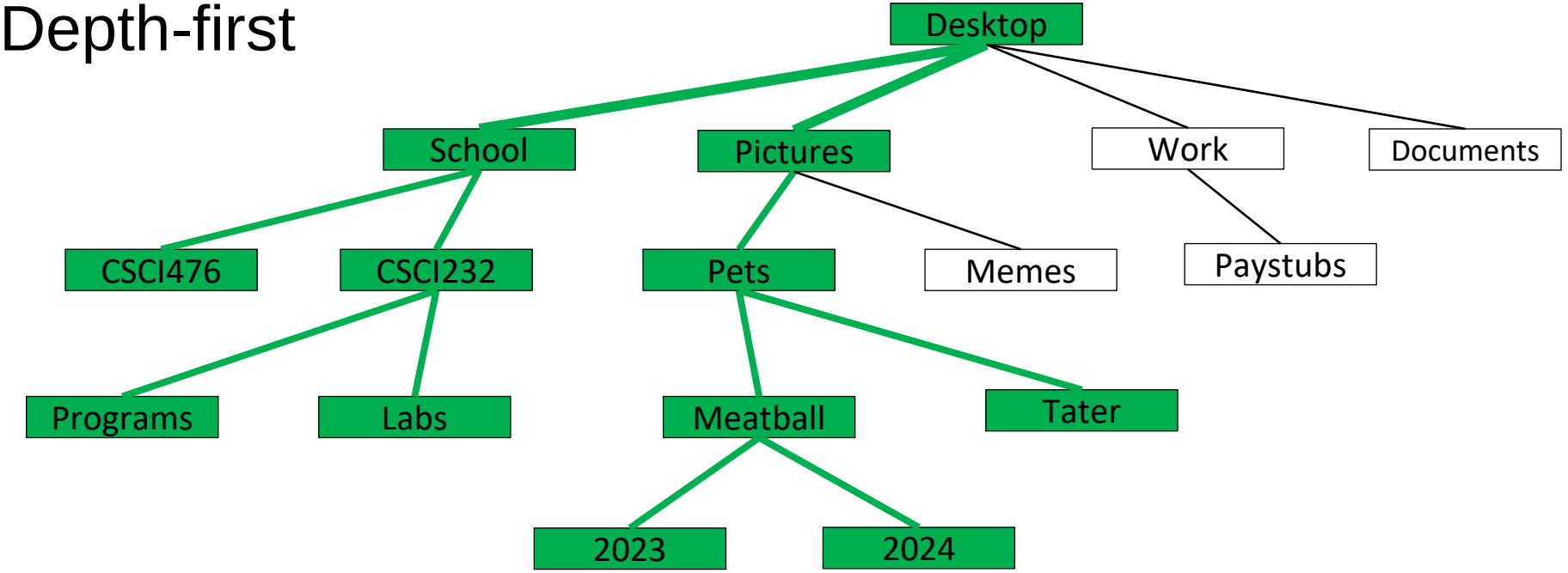


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024

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

Depth-first

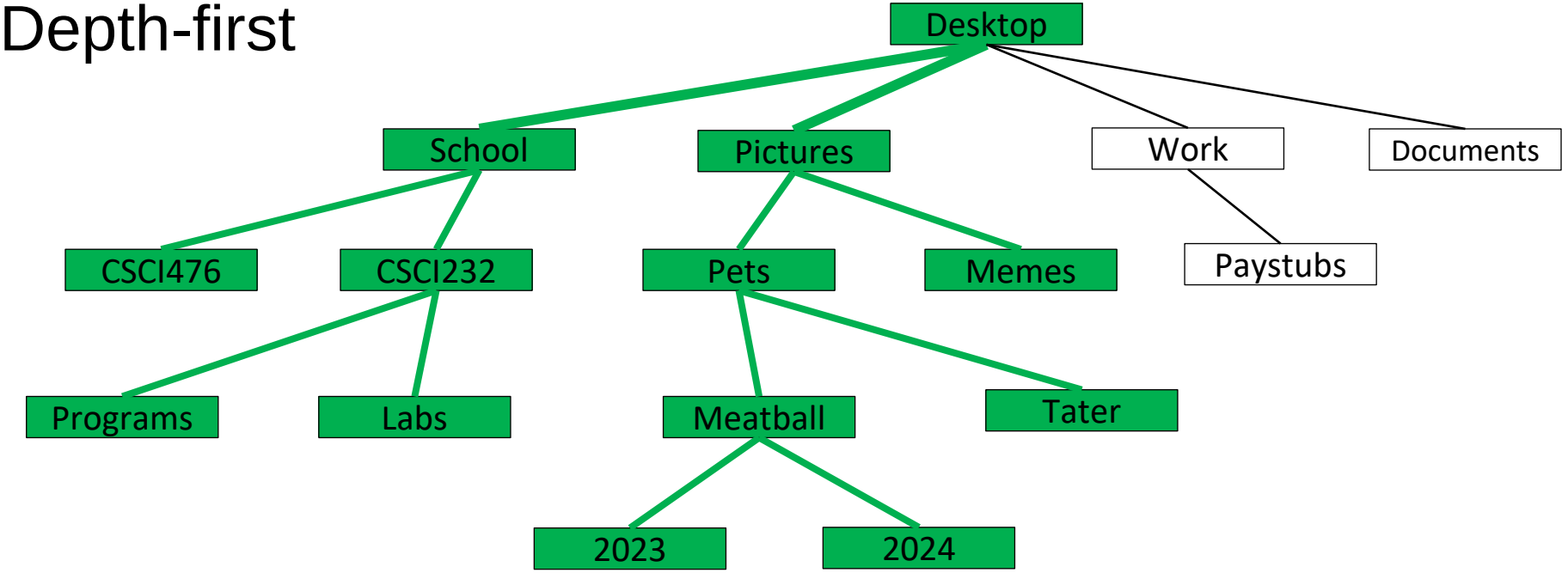


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater

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

Depth-first

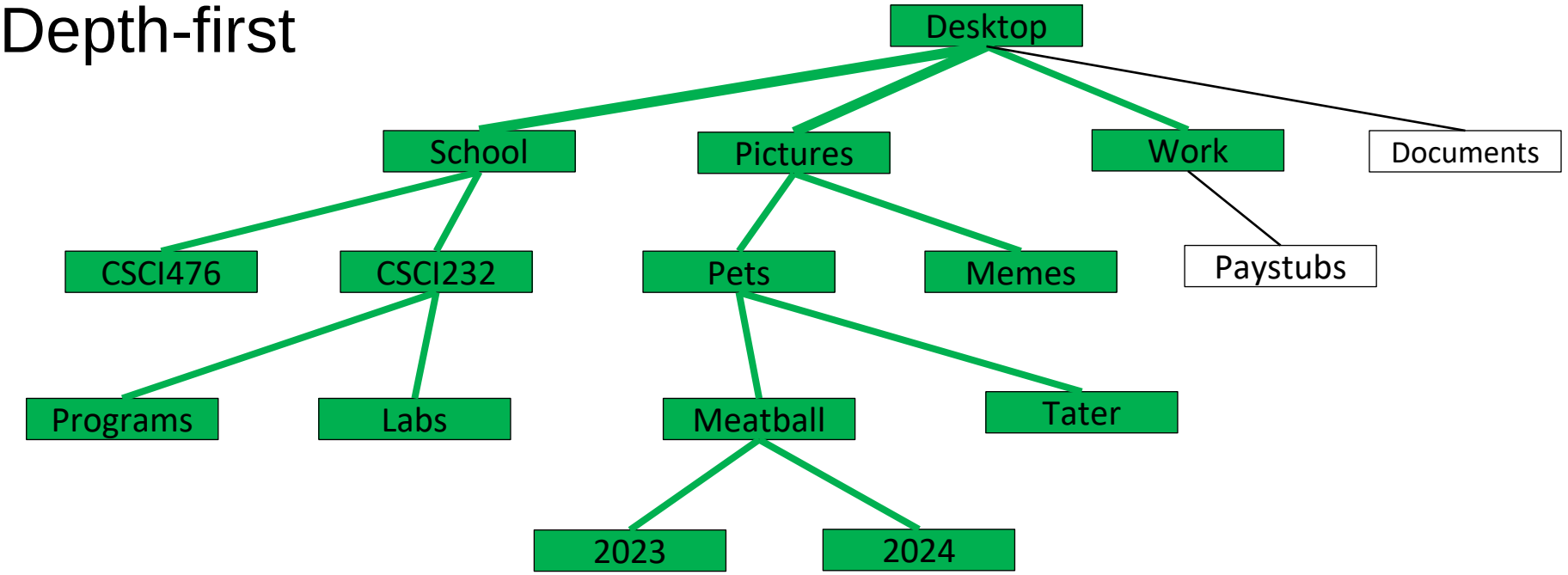


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater
Memes

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

Depth-first

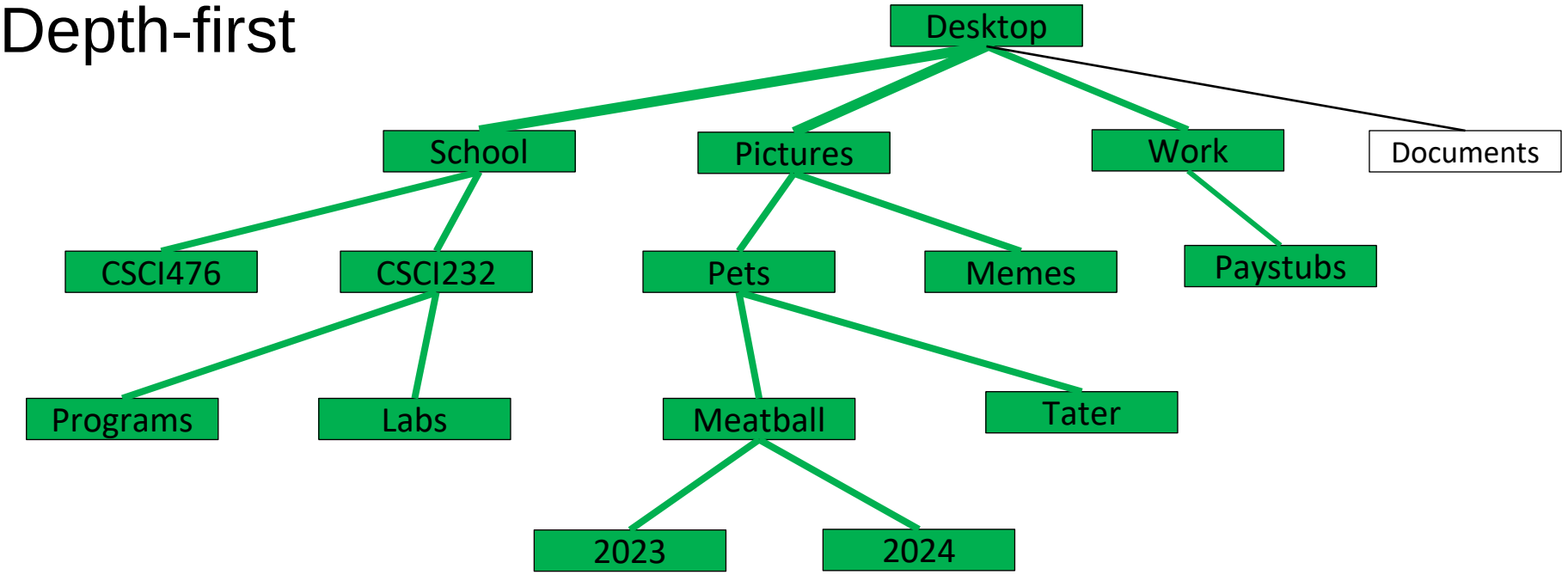


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater
Memes
Work

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

Depth-first

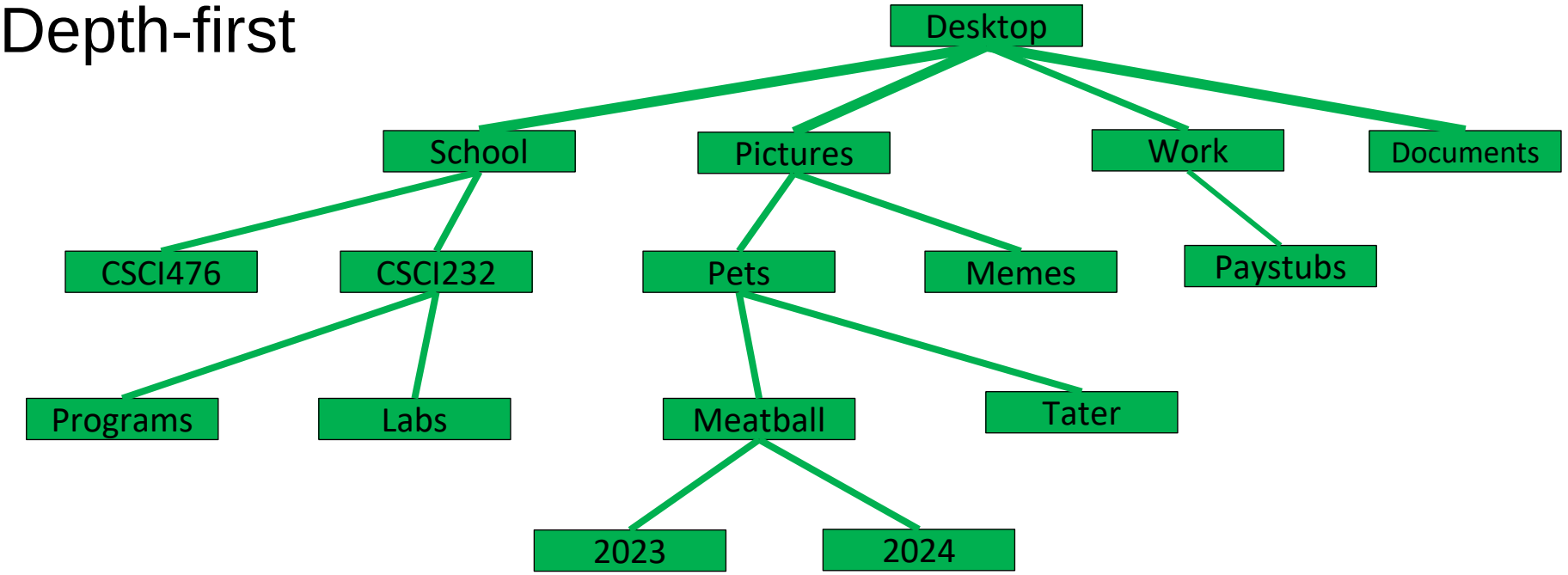


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater
Memes
Work
Paystubs

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

Depth-first

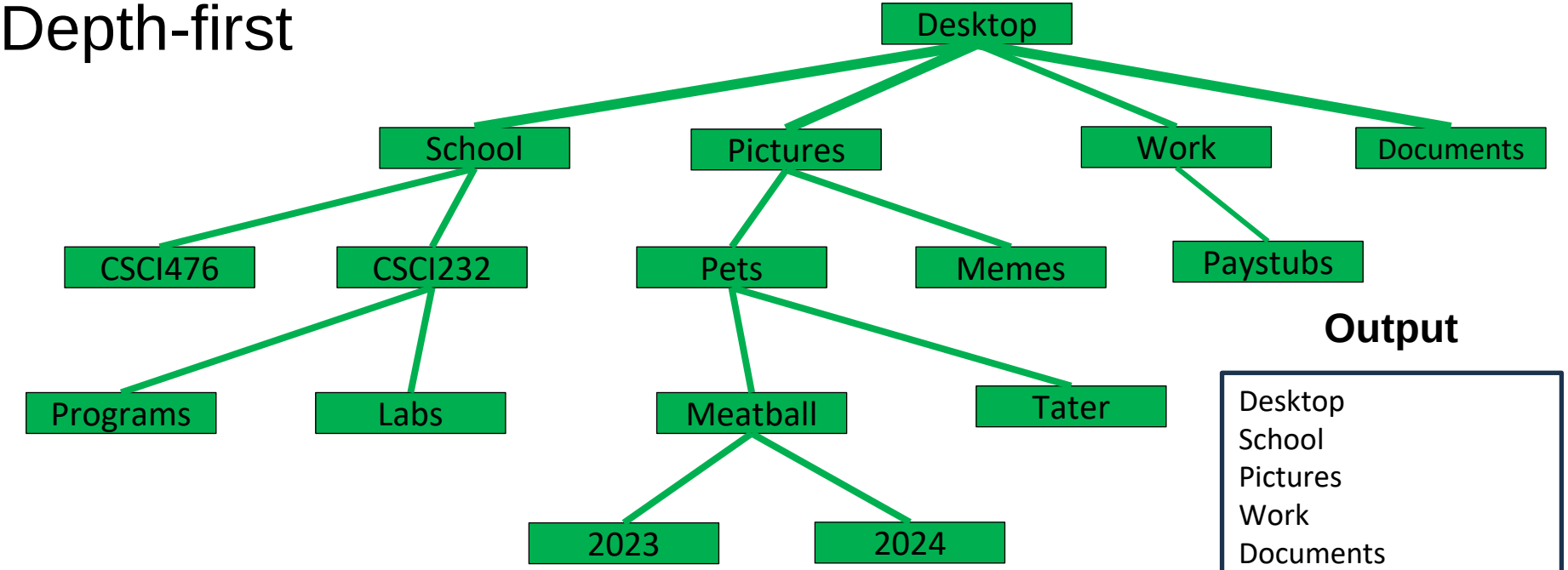


Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater
Memes
Work
Paystubs
Documents

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

Depth-first



Output

Desktop
School
Pictures
Work
Documents
CSCI476
CSCI232
Pets
Memes
Paystubs
Programs
Labs
Meatball
Tater
2023
2024

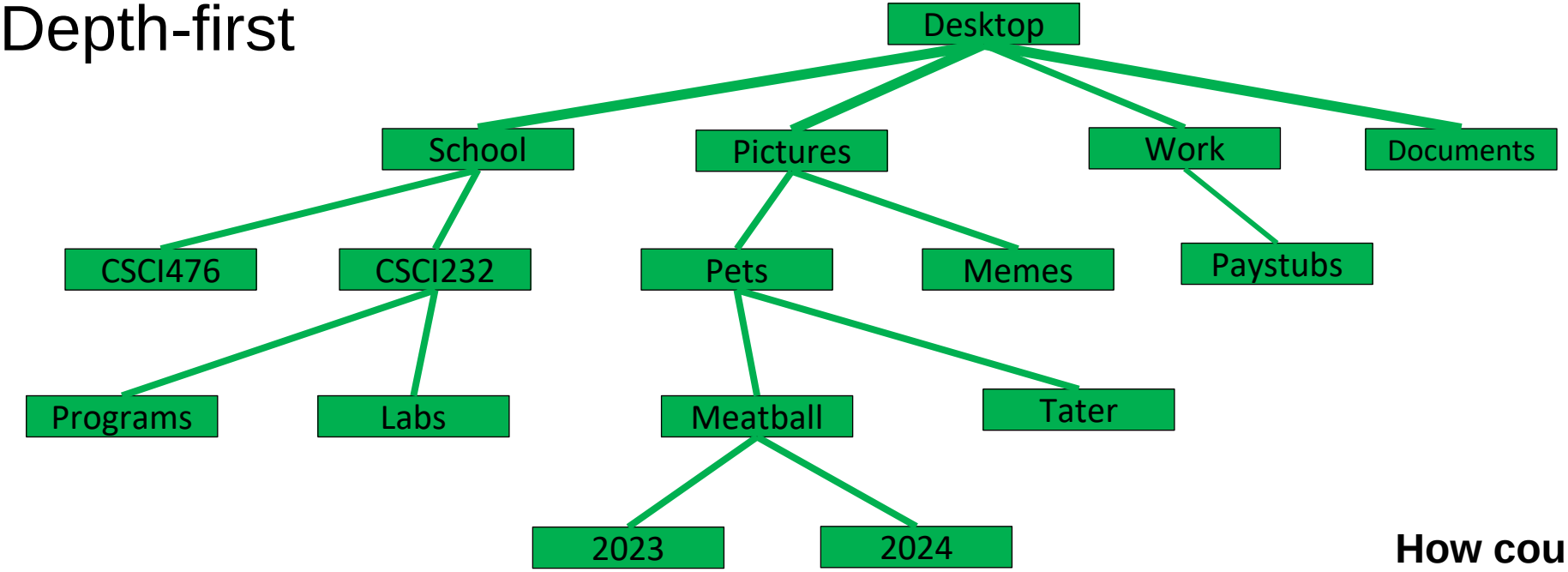
Breadth First

Output

Desktop
School
CSCI476
CSCI 232
Programs
Labs
Pictures
Pets
Meatball
2023
2024
Tater
Memes
Work
Paystubs
Documents

Depth First

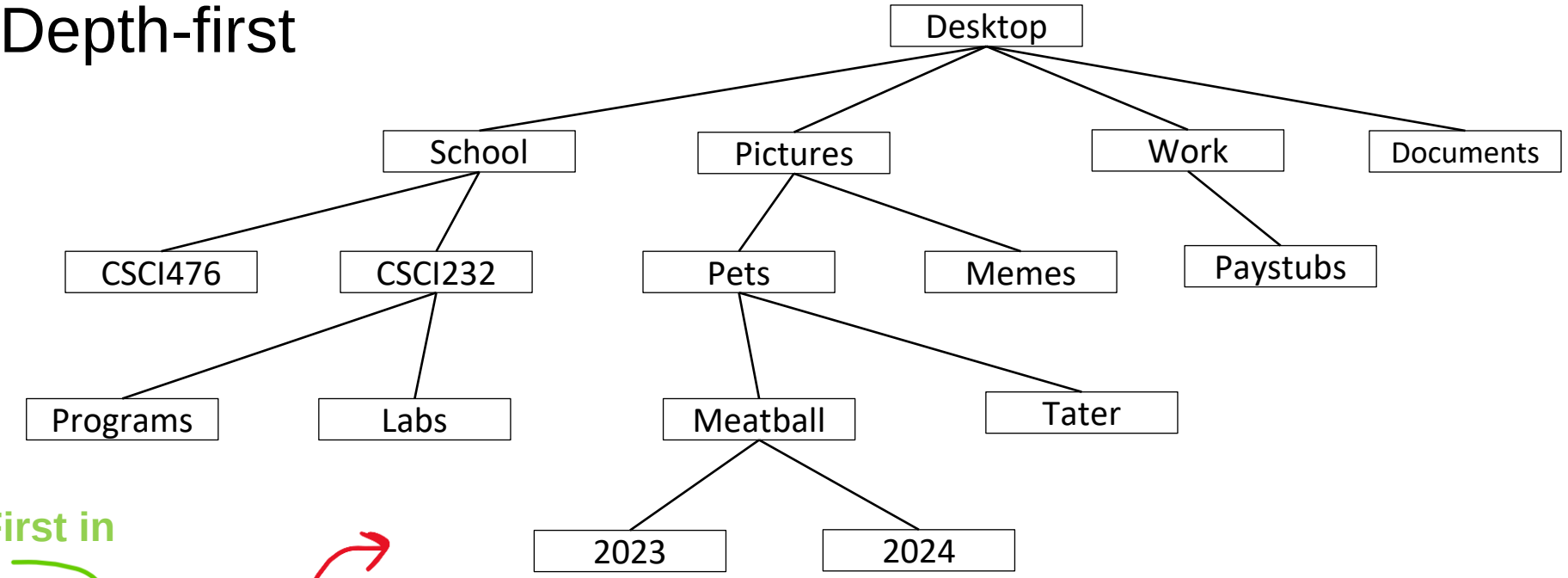
Depth-first



How could we implement this?

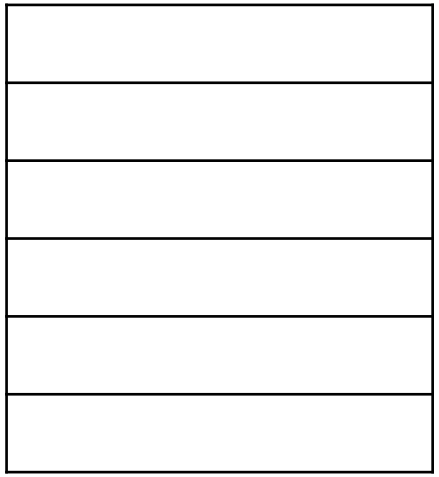
1. Go all the way down the “first” leaf
2. Backtrack until unvisited child is encountered
3. Repeat

Depth-first



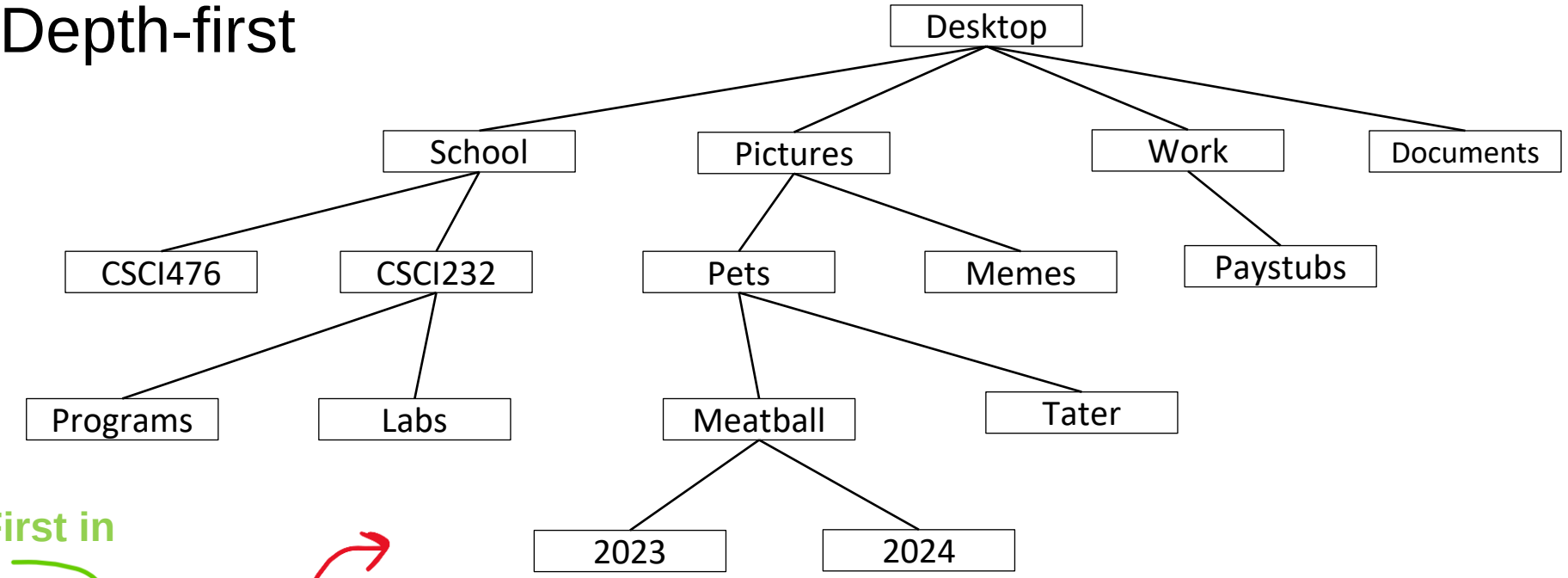
First in

Last out



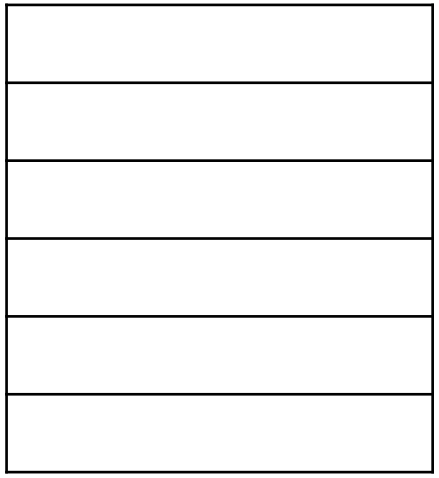
Stack

Depth-first



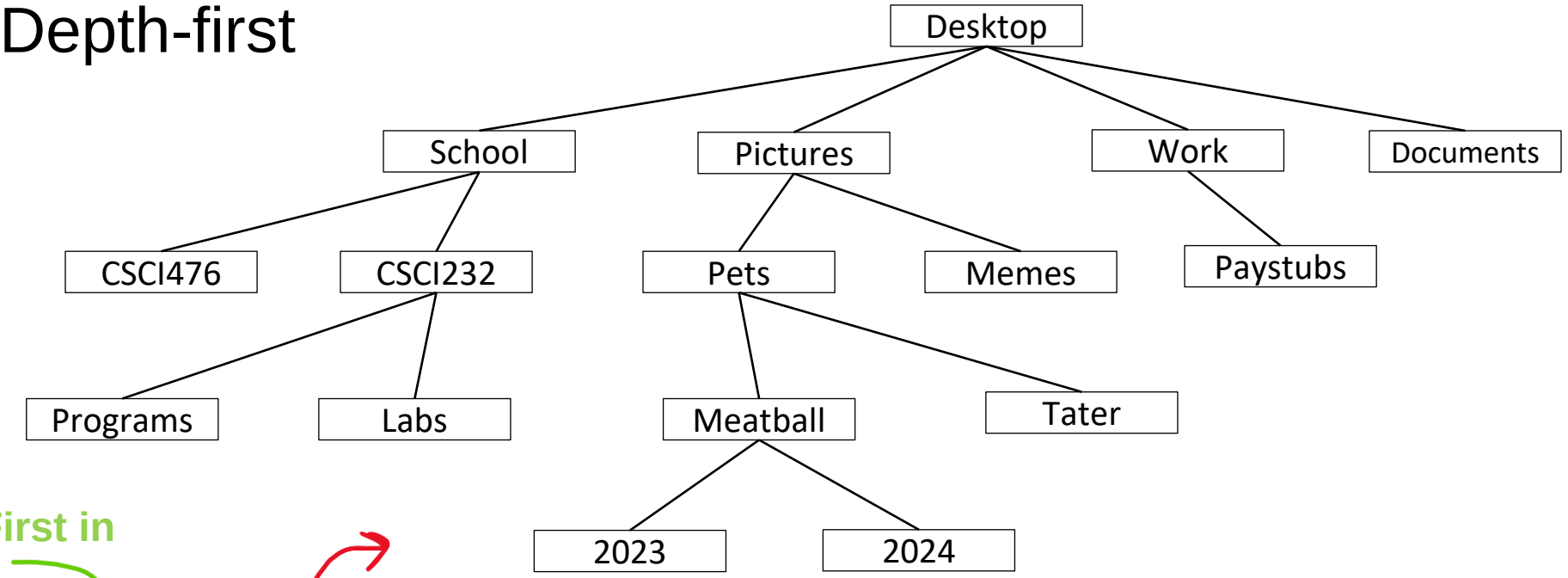
First in

Last out



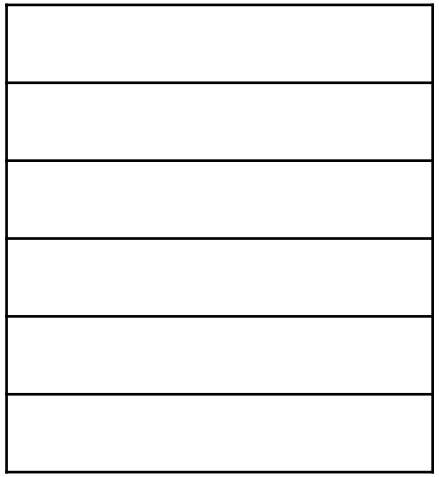
Every time we “visit” a node we:

Depth-first



First in

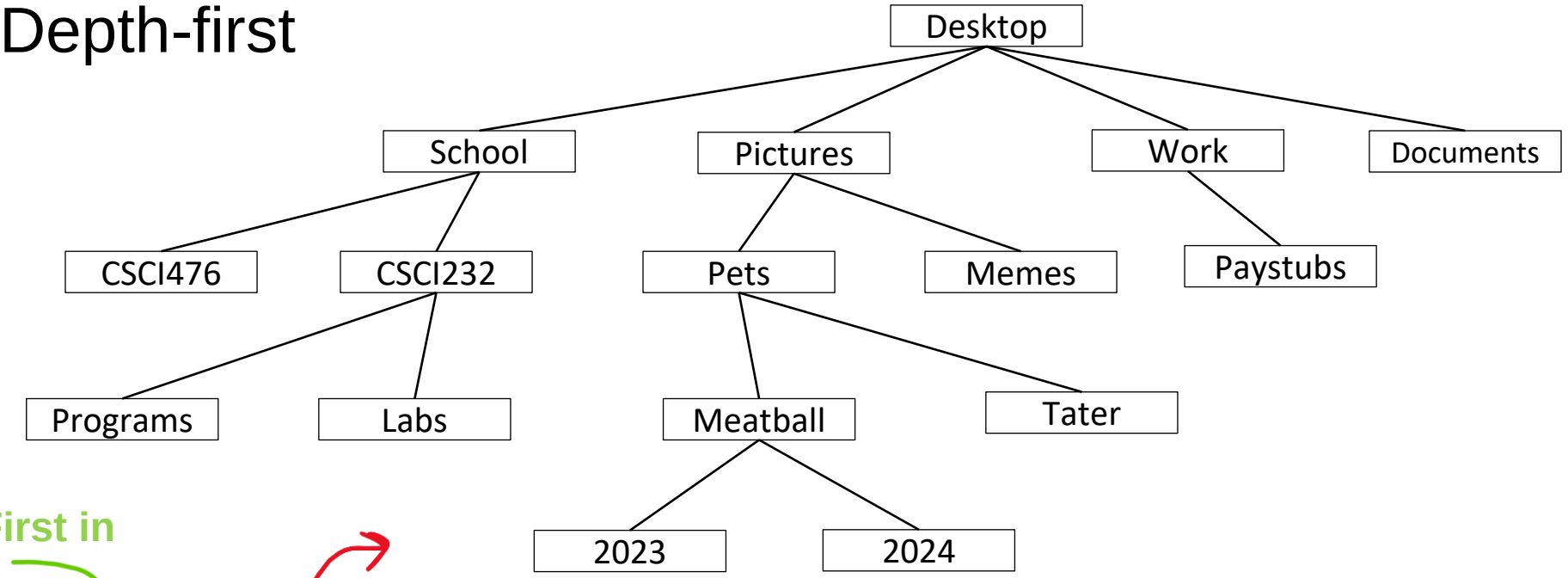
Last out



Every time we “visit” a node we:

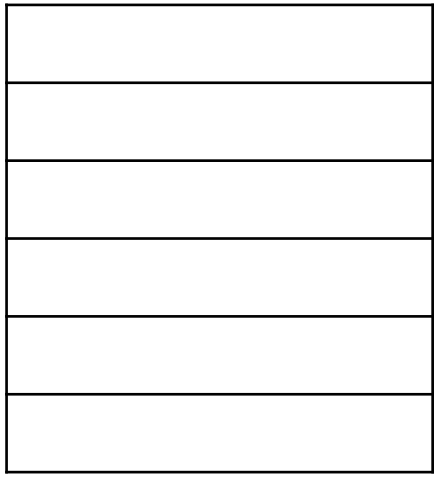
1. Remove node from stack

Depth-first



First in

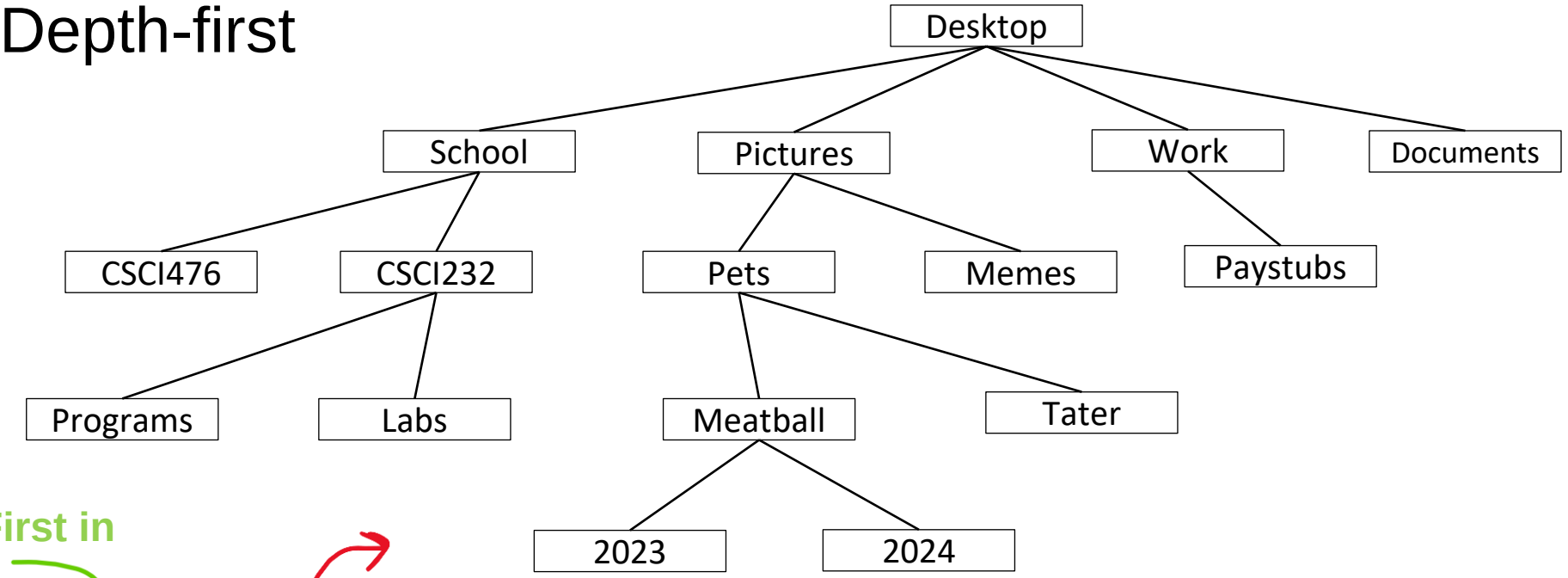
Last out



Every time we “visit” a node we:

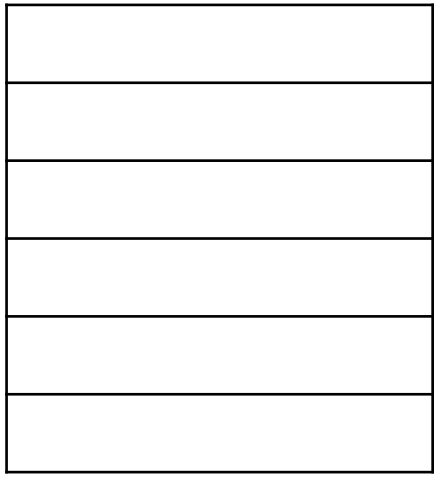
1. Remove node from stack
2. Execute the action (print, compare, etc)

Depth-first



First in

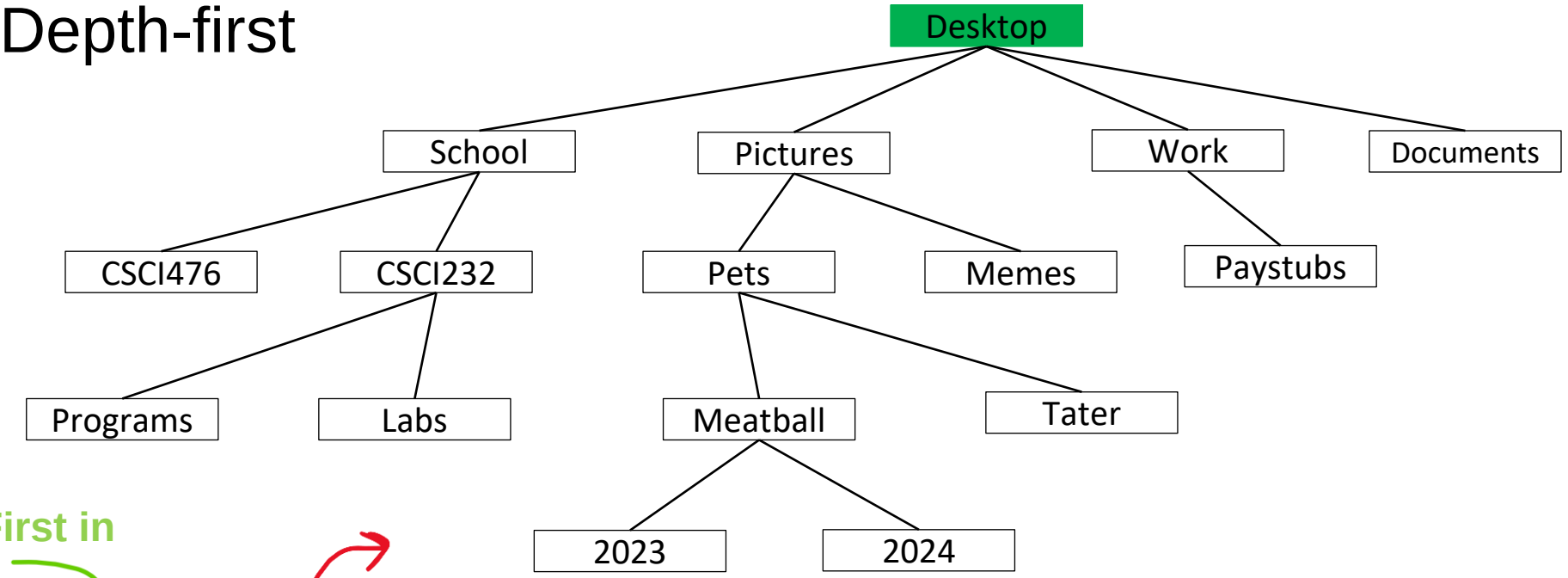
Last out



Every time we “visit” a node we:

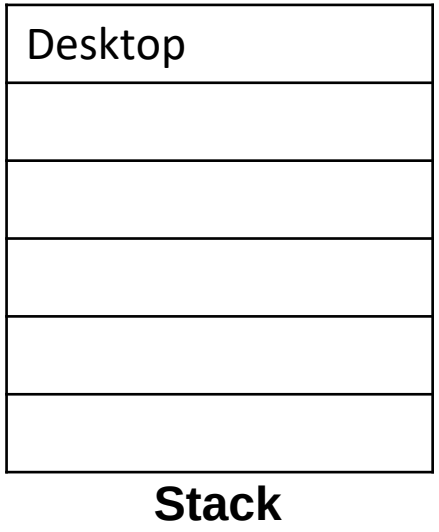
1. Remove node from stack
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



First in

Last out

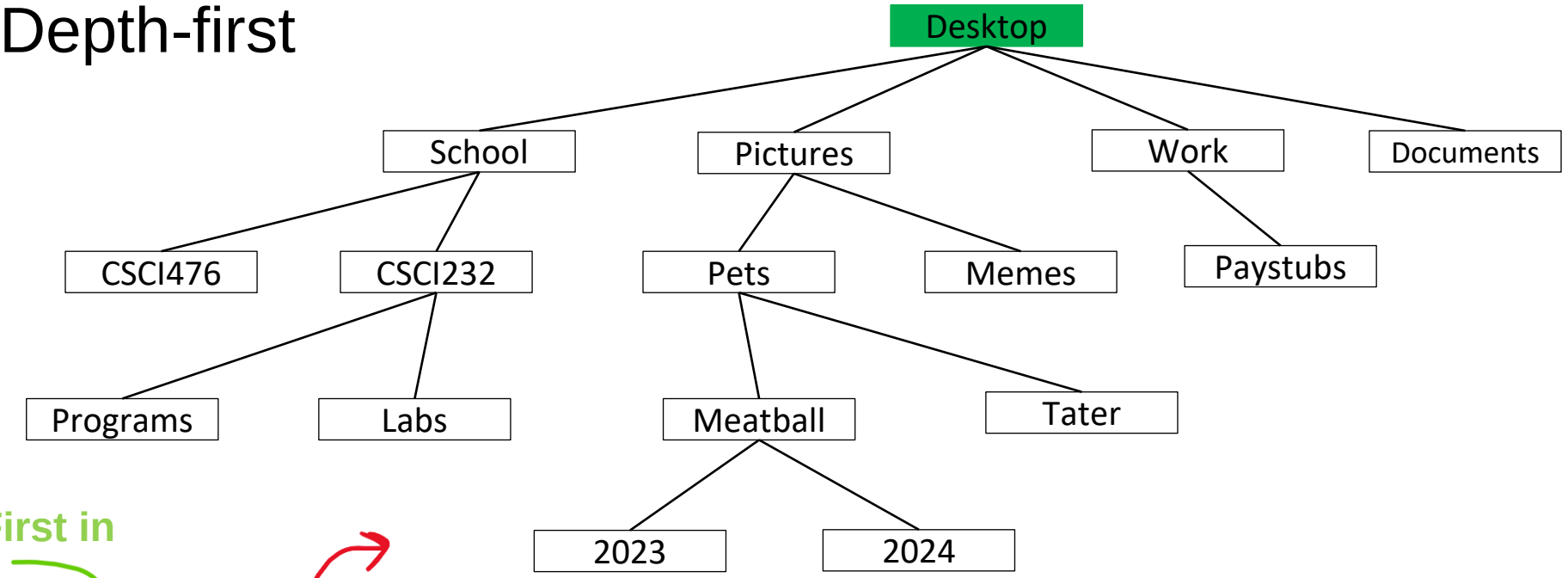


Stack

Every time we “visit” a node we:

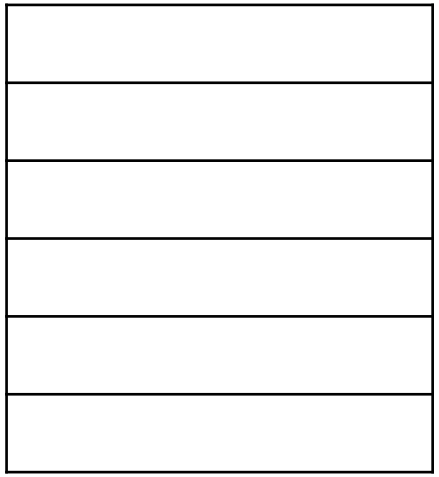
1. Remove node from stack
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



First in

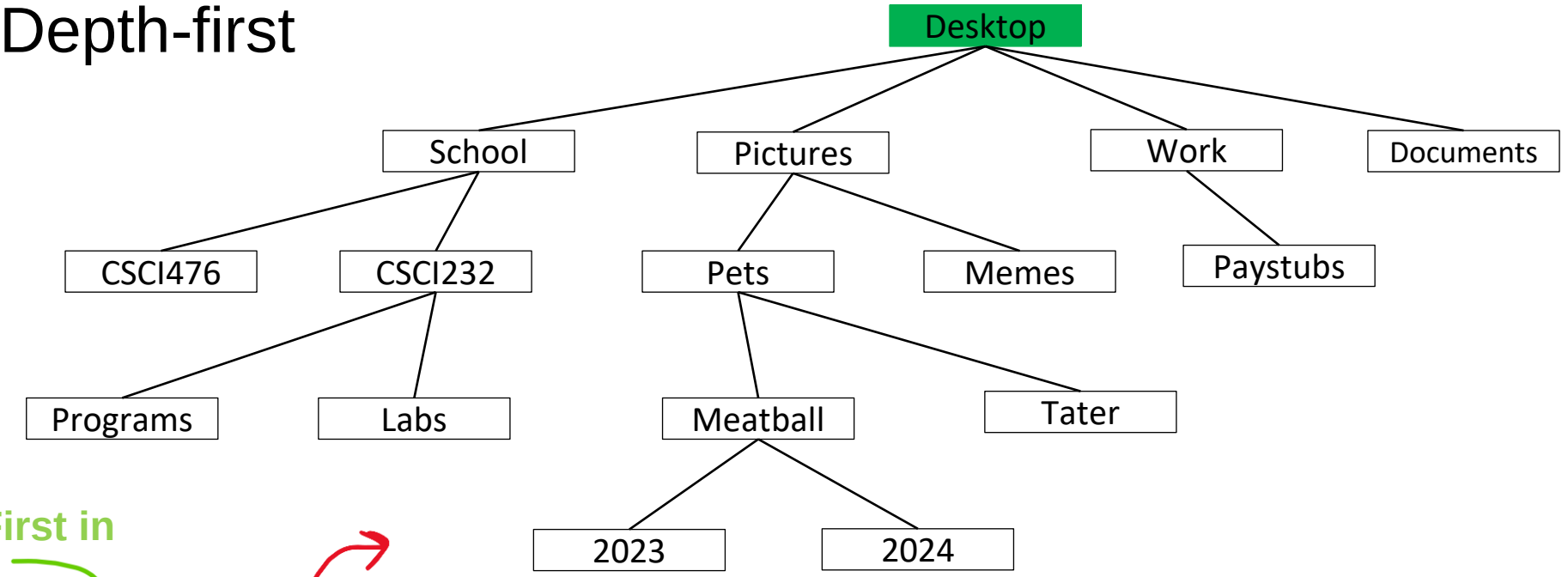
Last out



Every time we “visit” a node we:

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

Depth-first

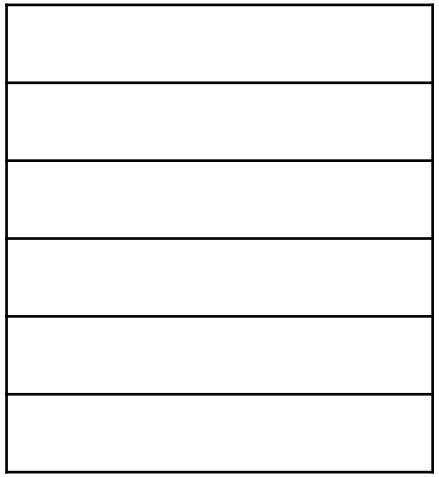


Output

Desktop

First in

Last out

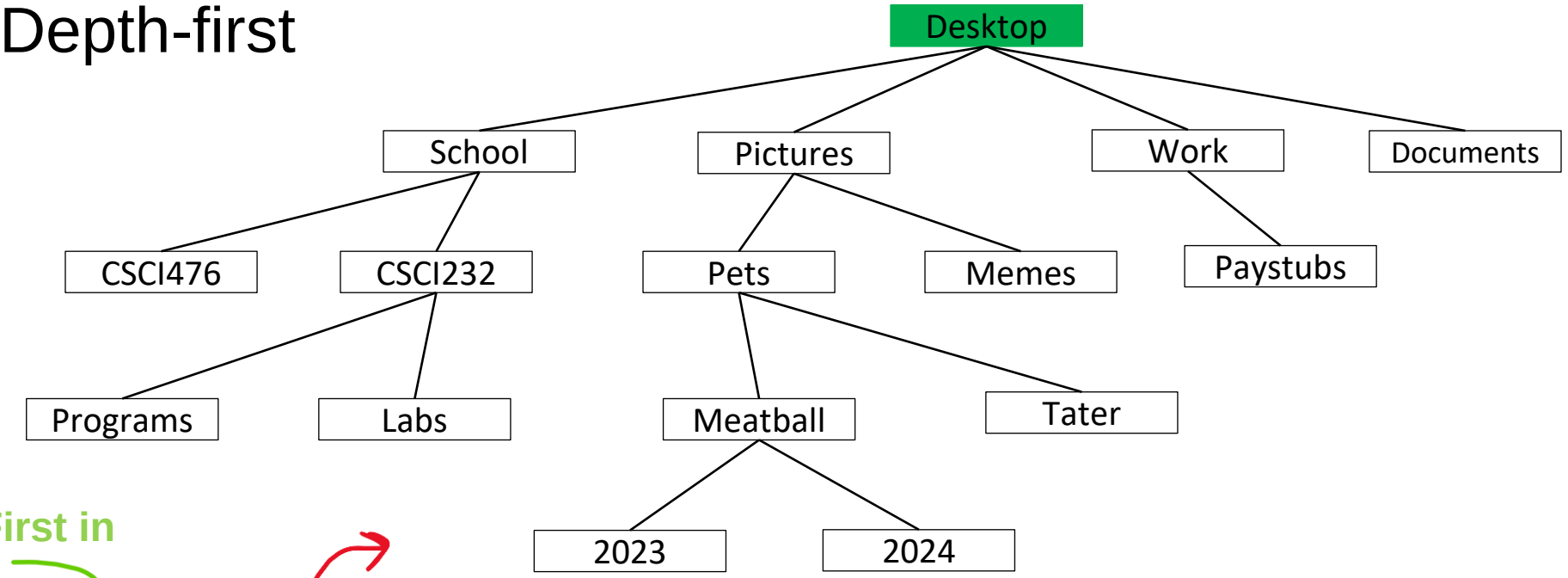


Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop

First in

Last out

!!

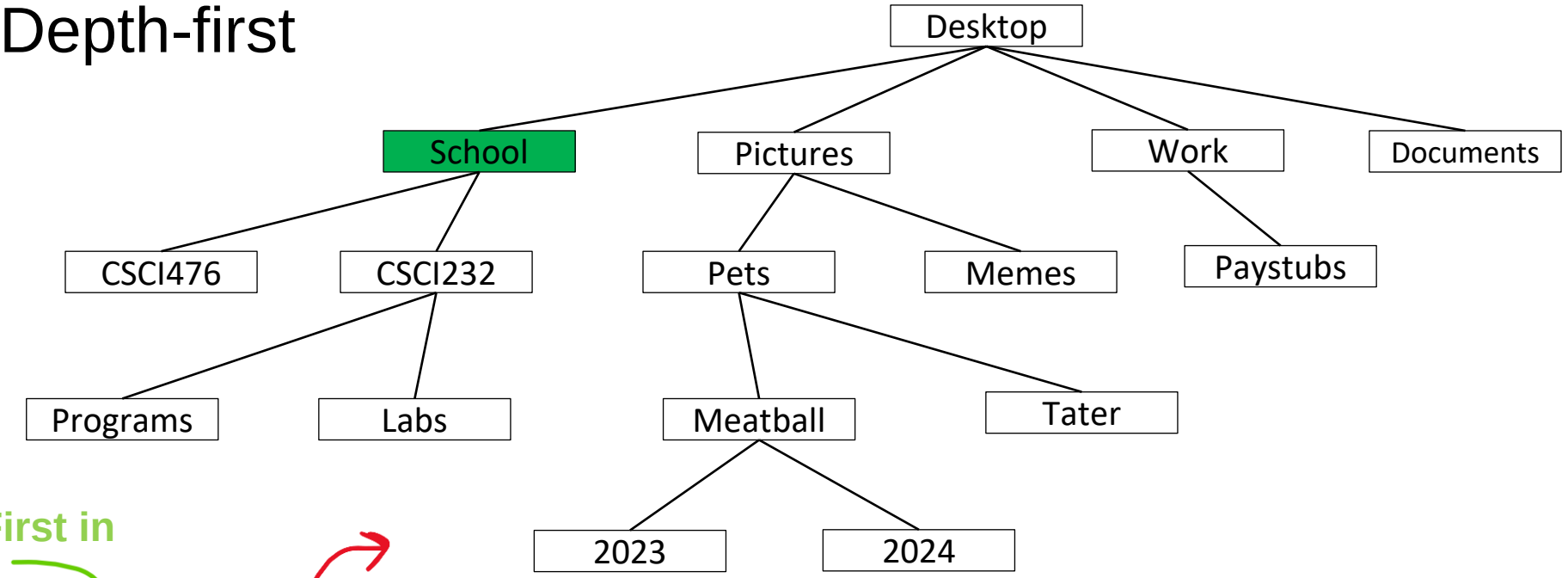
School
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop

First in

Last out

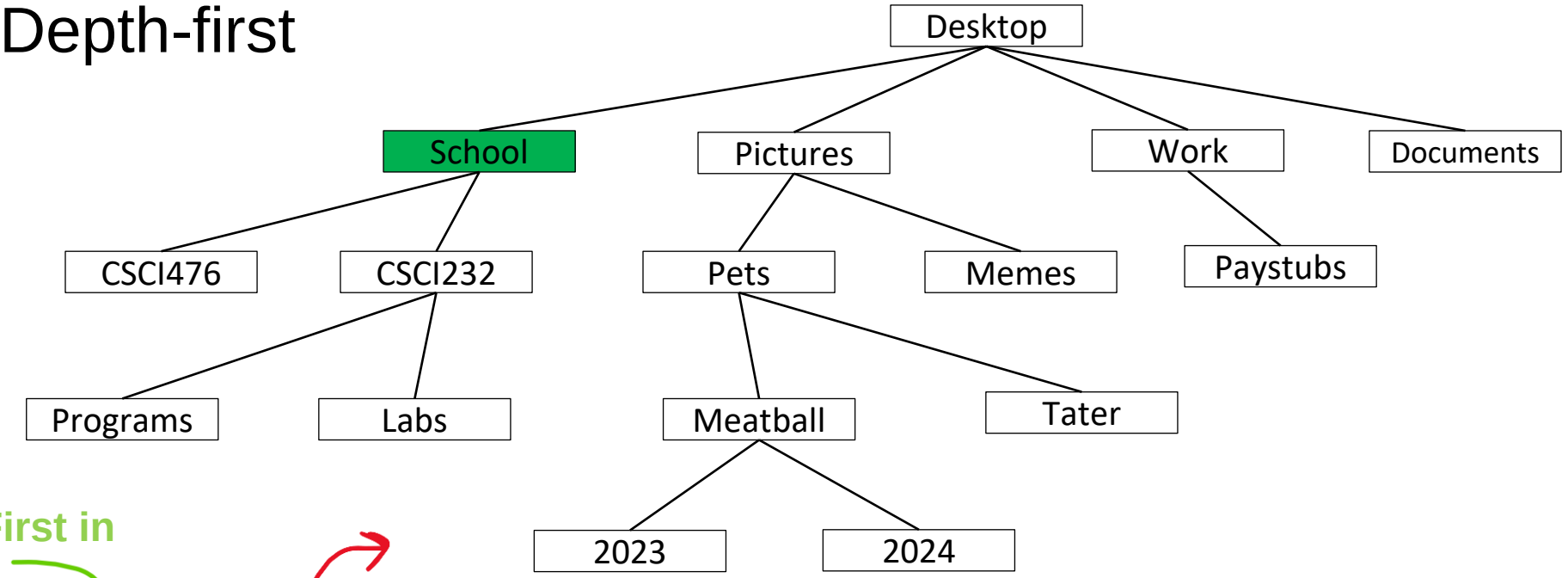
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School

First in

Last out

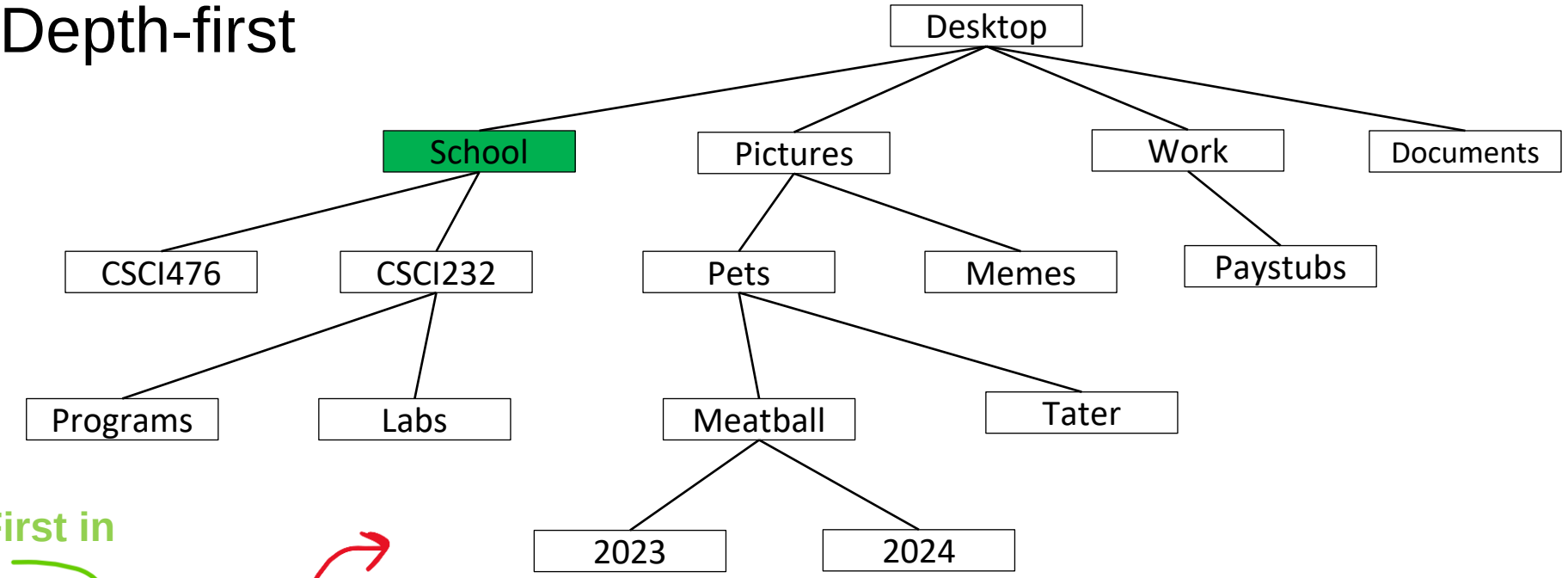
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School

First in

Last out

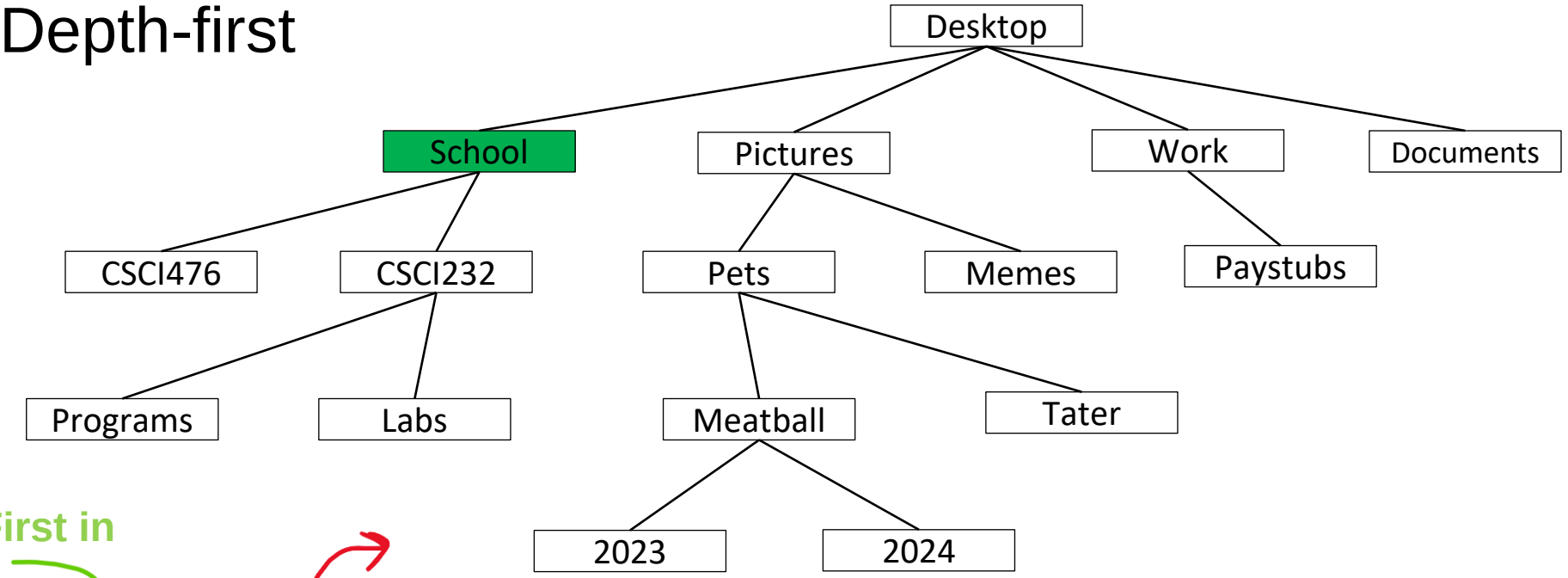
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School

First in

Last out

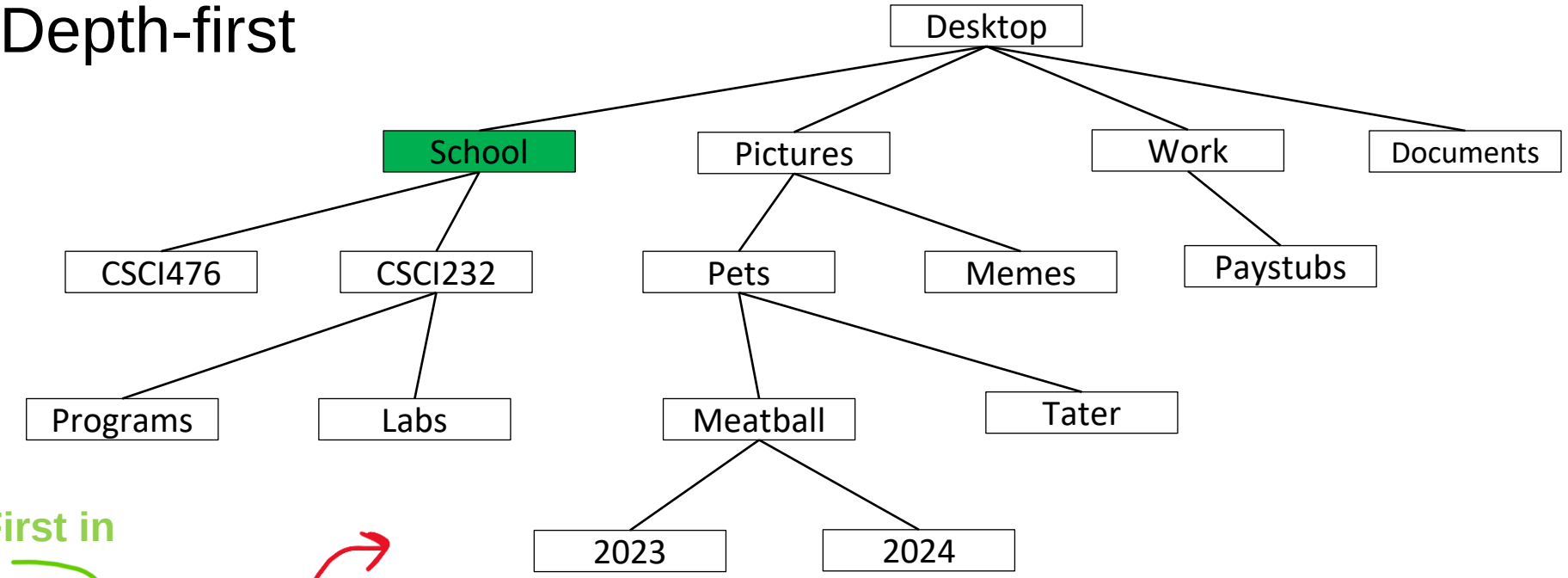
CSCI476
CSCI232
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School

First in

Last out

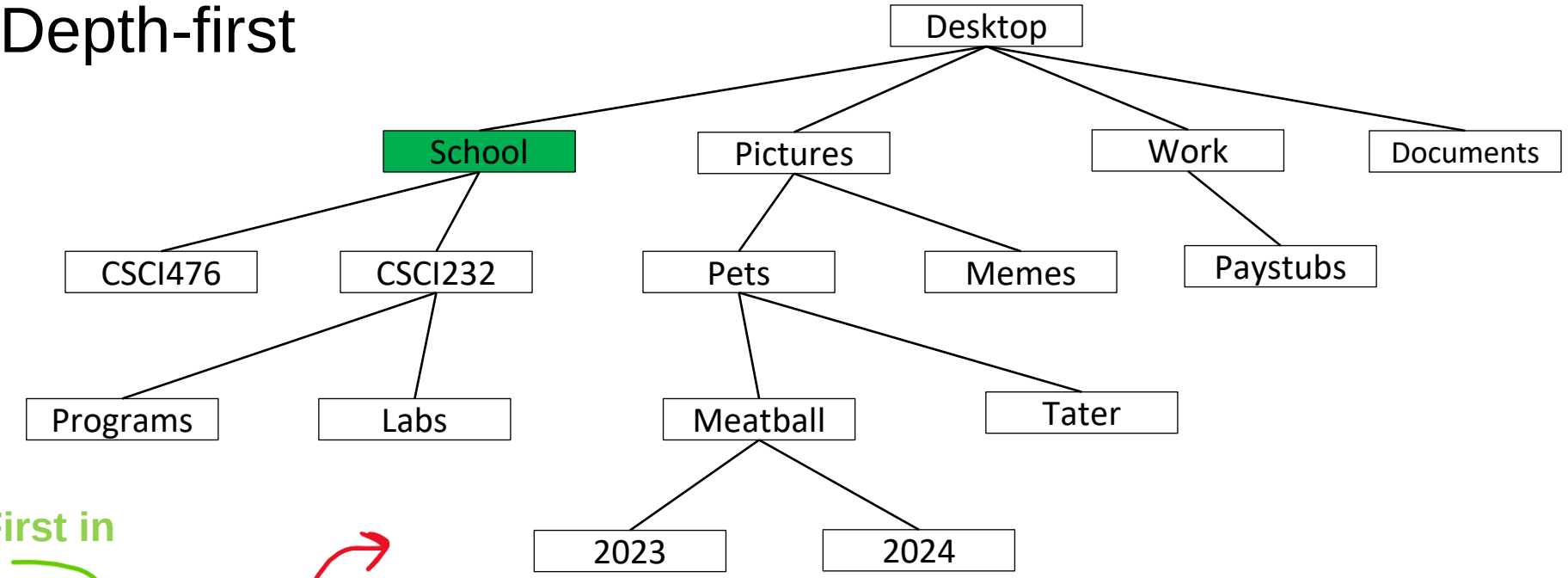
CSCI476
CSCI232
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School

First in

Last out

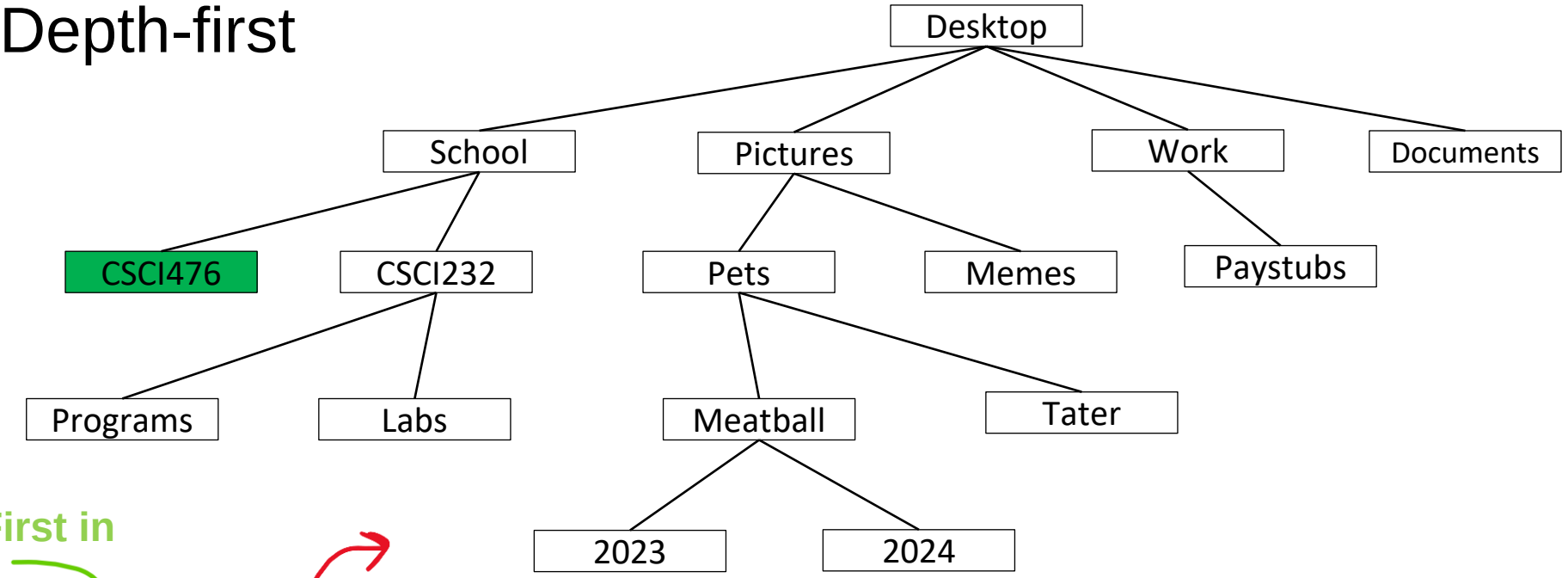
CSCI232
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

1. Remove node from stack CSCI476
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476

First in

Last out

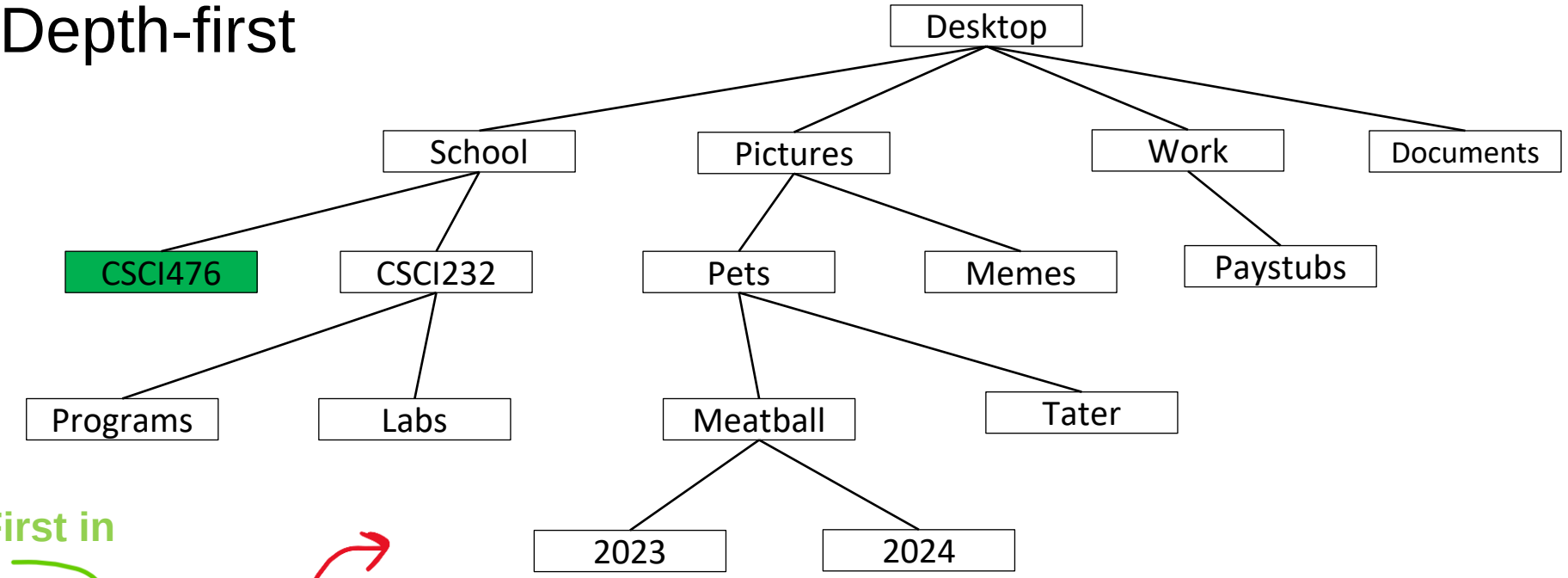
CSCI232
Pictures
Work
Documents

Stack

Every time we “visit” a node we:

1. Remove node from stack CSCI476
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476

First in

Last out

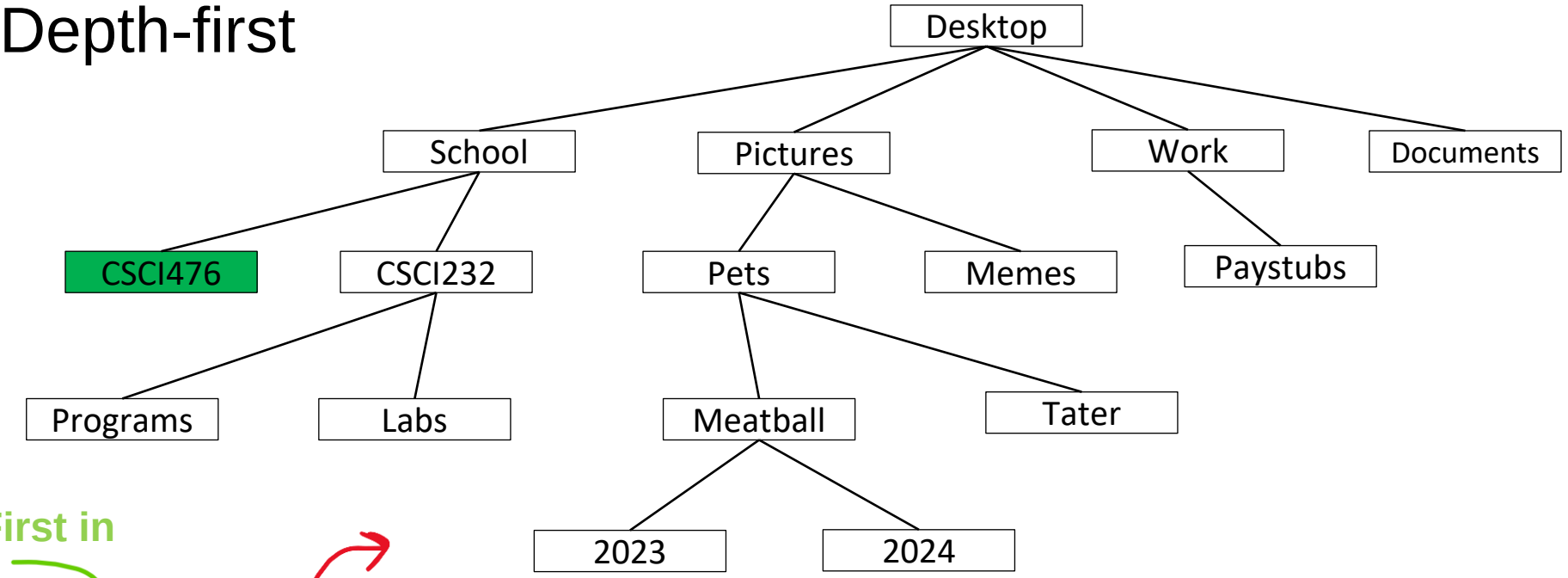
CSCI232
Pictures
Work
Documents

Stack

Every time we “visit” a node we:

1. Remove node from stack CSCI476
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476

First in

Last out

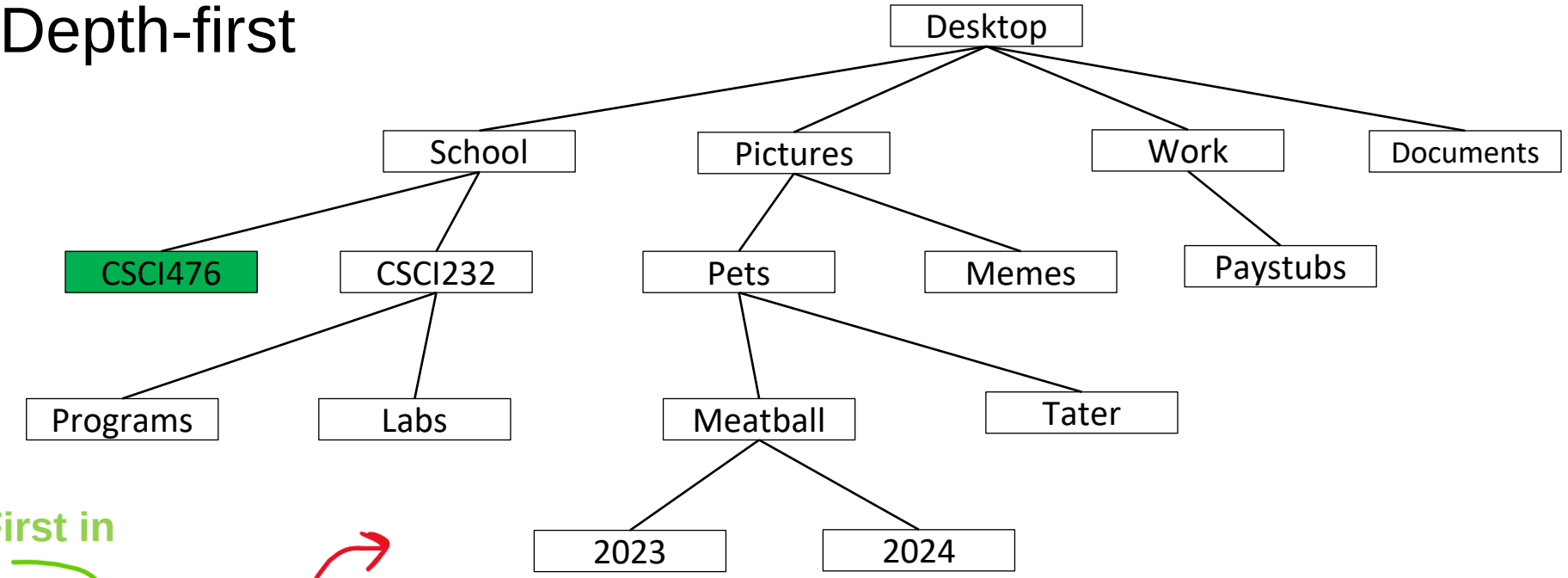
CSCI232
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476

First in

Last out

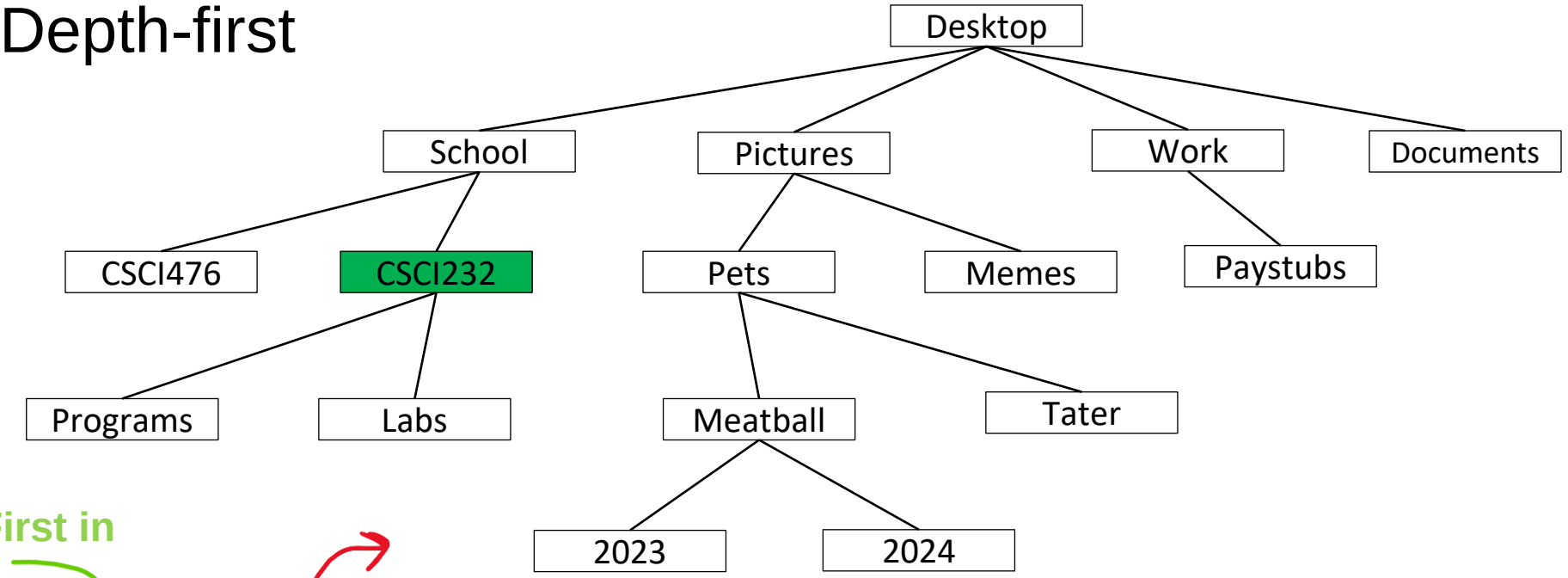
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

1. Remove node from stack CSCI232
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476
CSCI232

First in

Last out

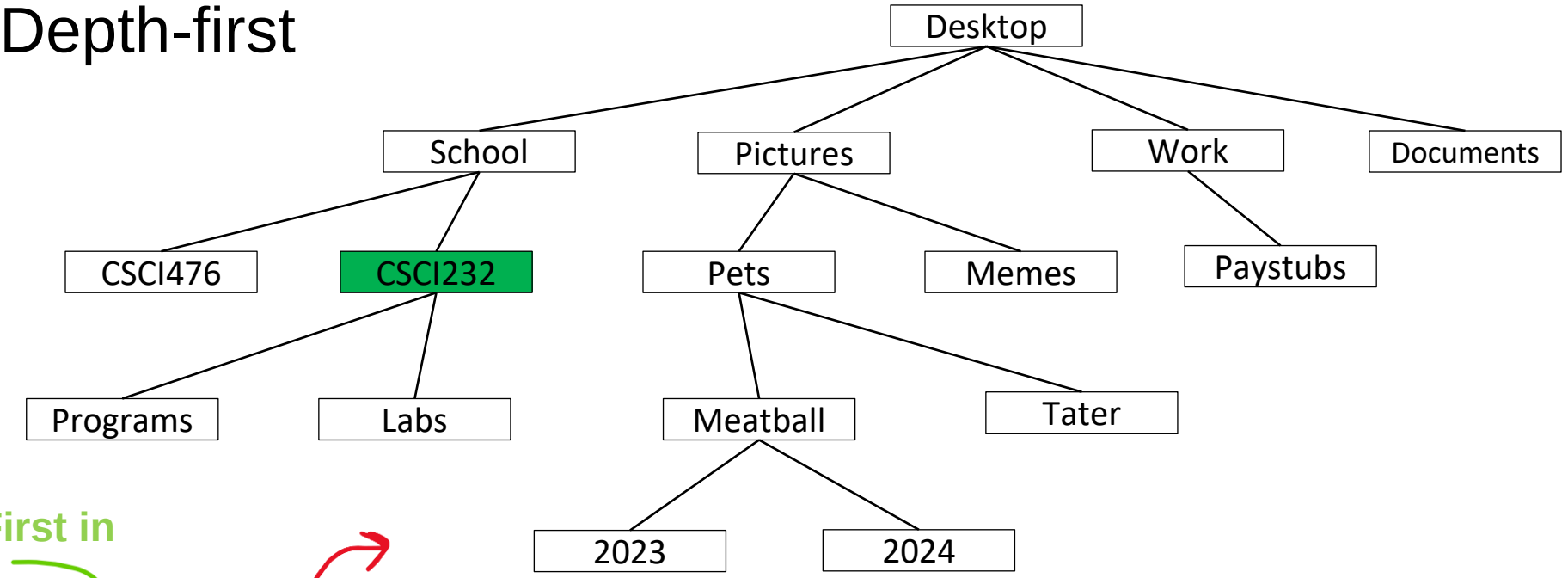
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

1. Remove node from stack CSCI232
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476
CSCI232

First in

Last out

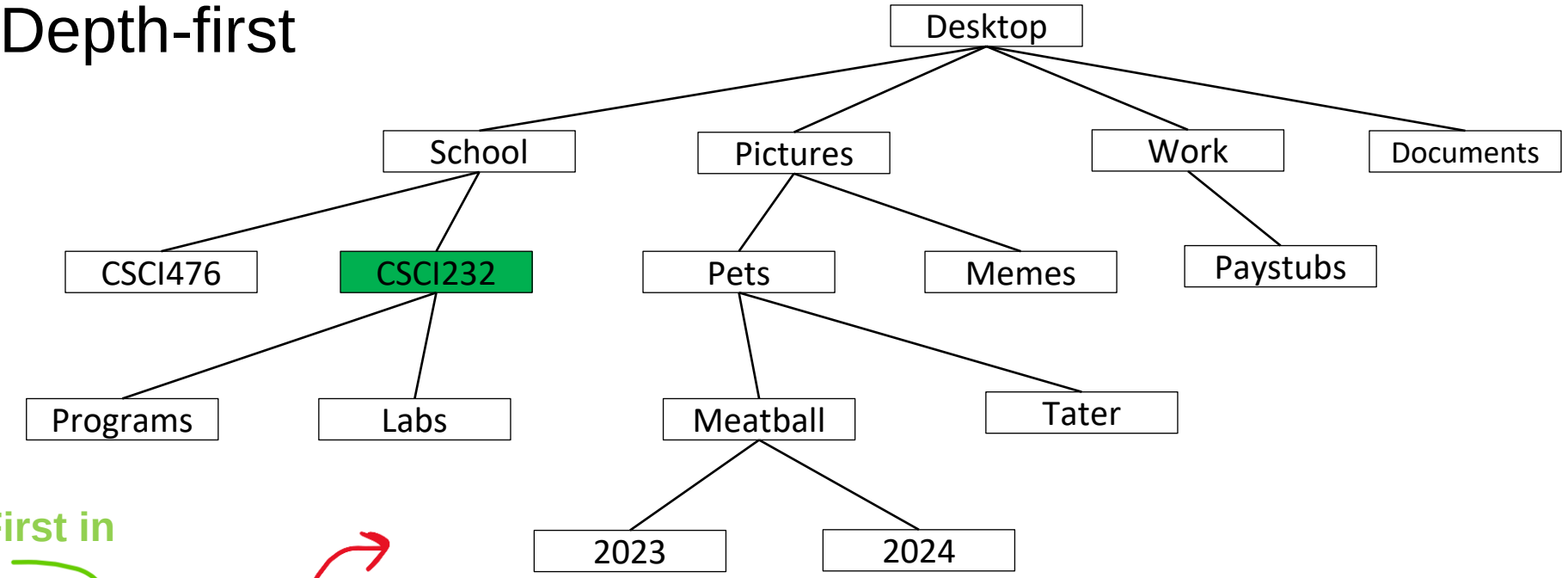
Programs
Labs
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

1. Remove node from stack CSCI232
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Depth-first



Output

Desktop
School
CSCI476
CSCI232

First in

Last out

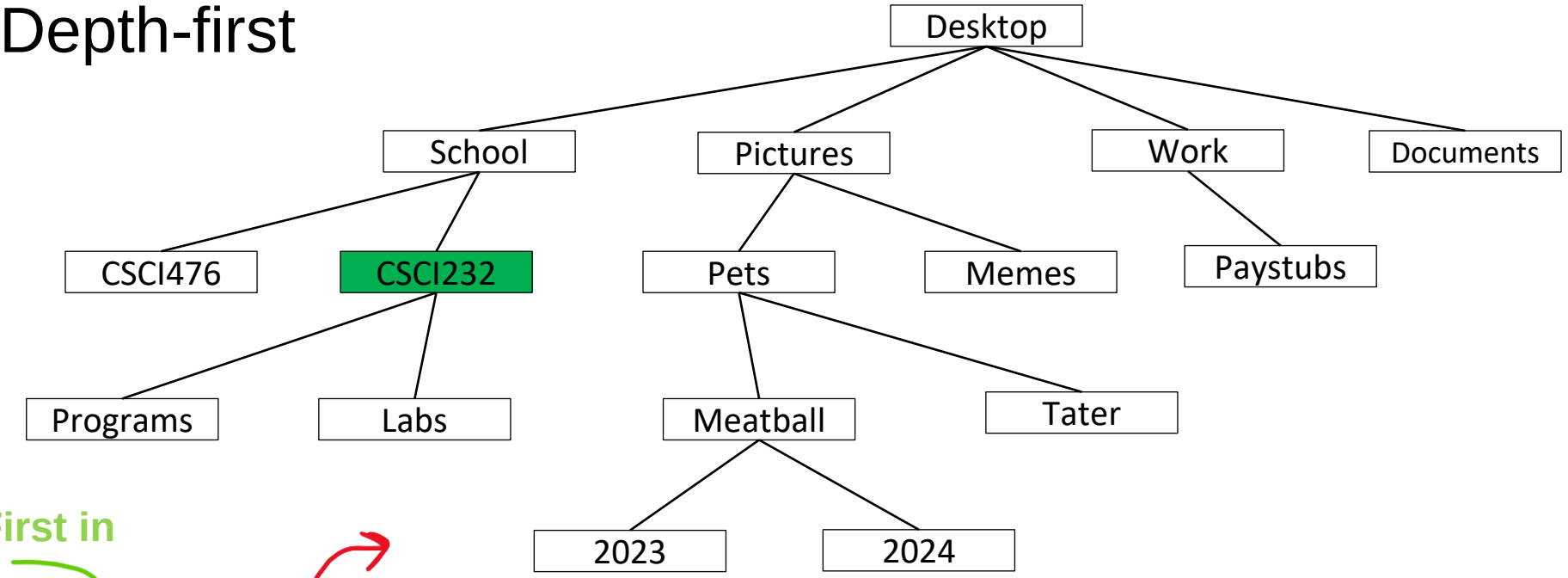
Programs
Labs
Pictures
Work
Documents

Stack

Every time we “visit” a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232

First in

Last out

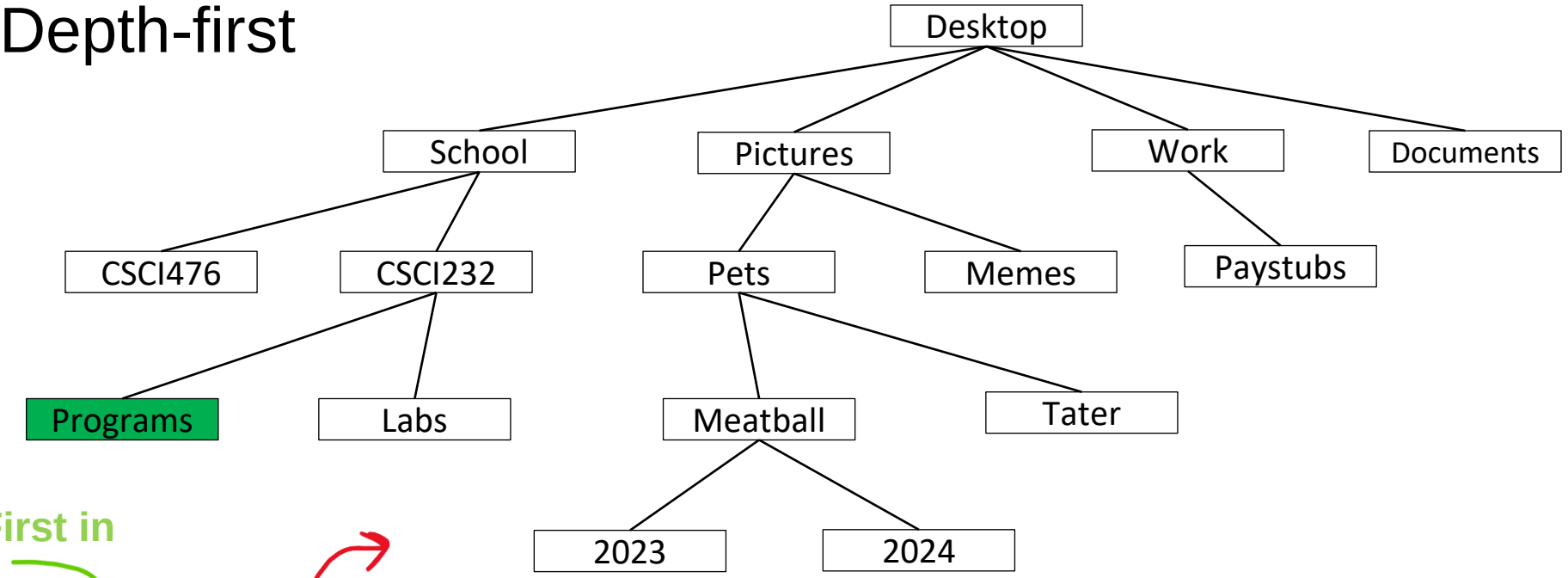
Labs
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs

First in

Last out

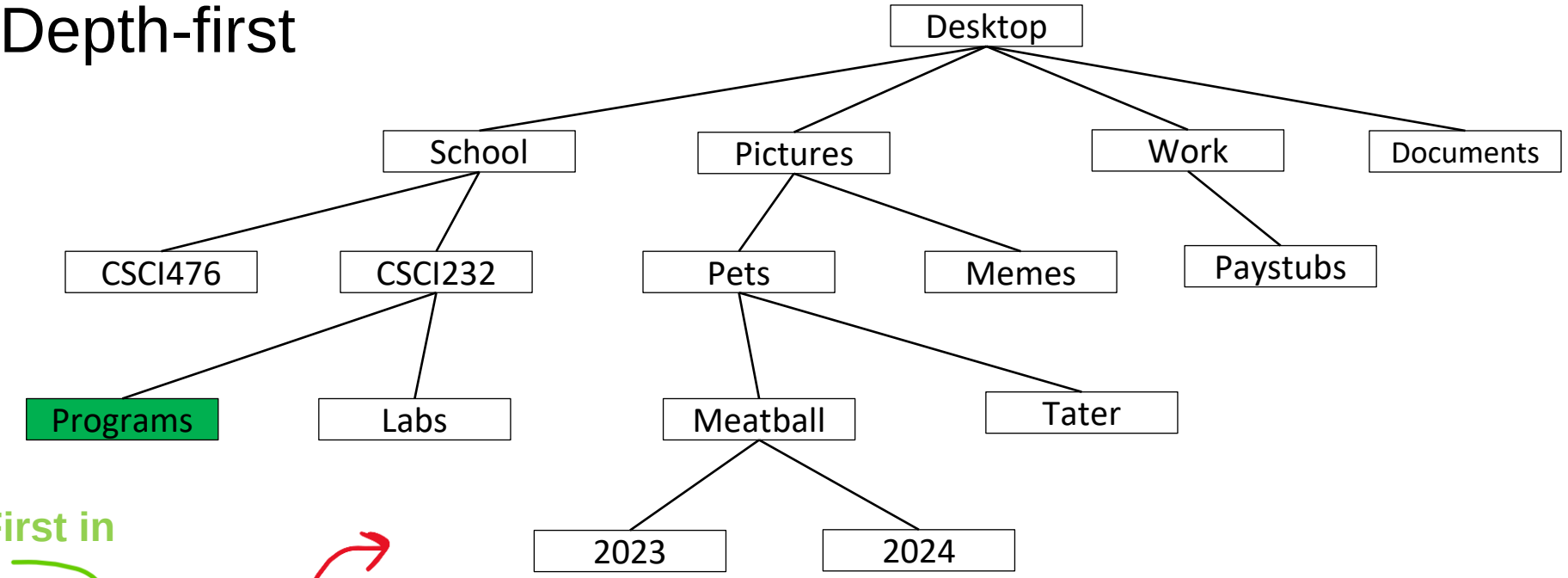
Labs
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs

First in

Last out

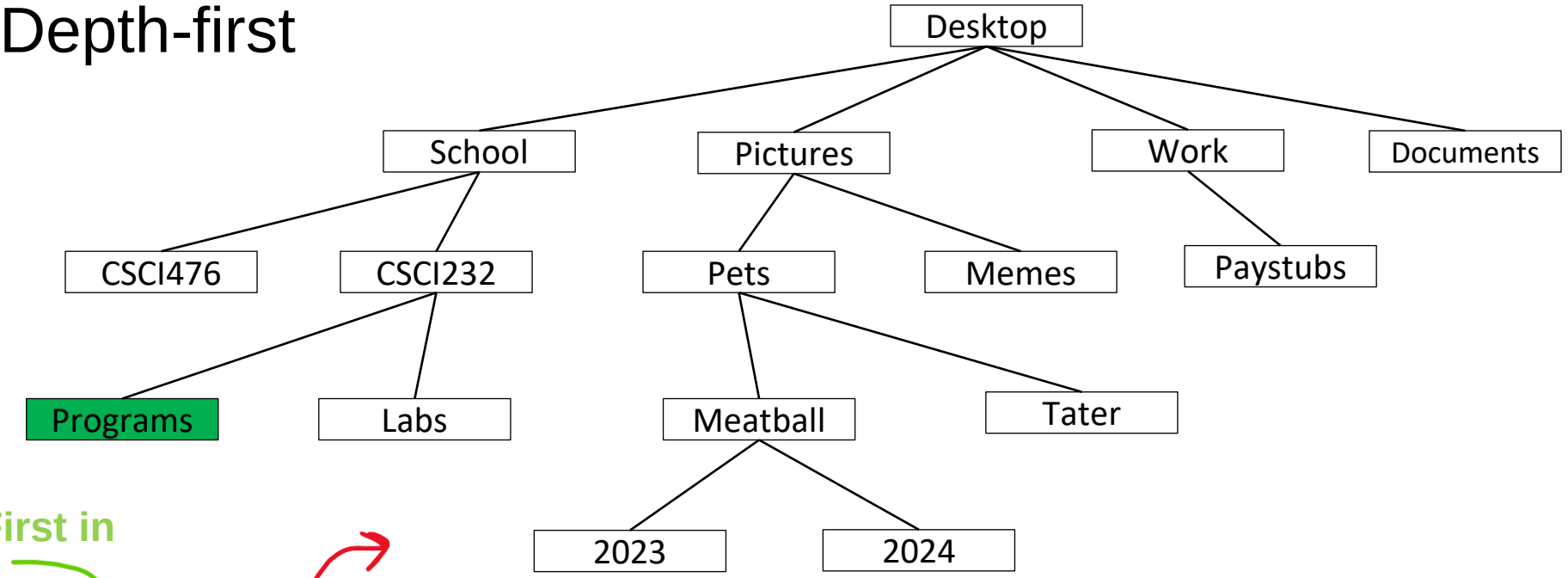
Labs
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs

First in

Last out

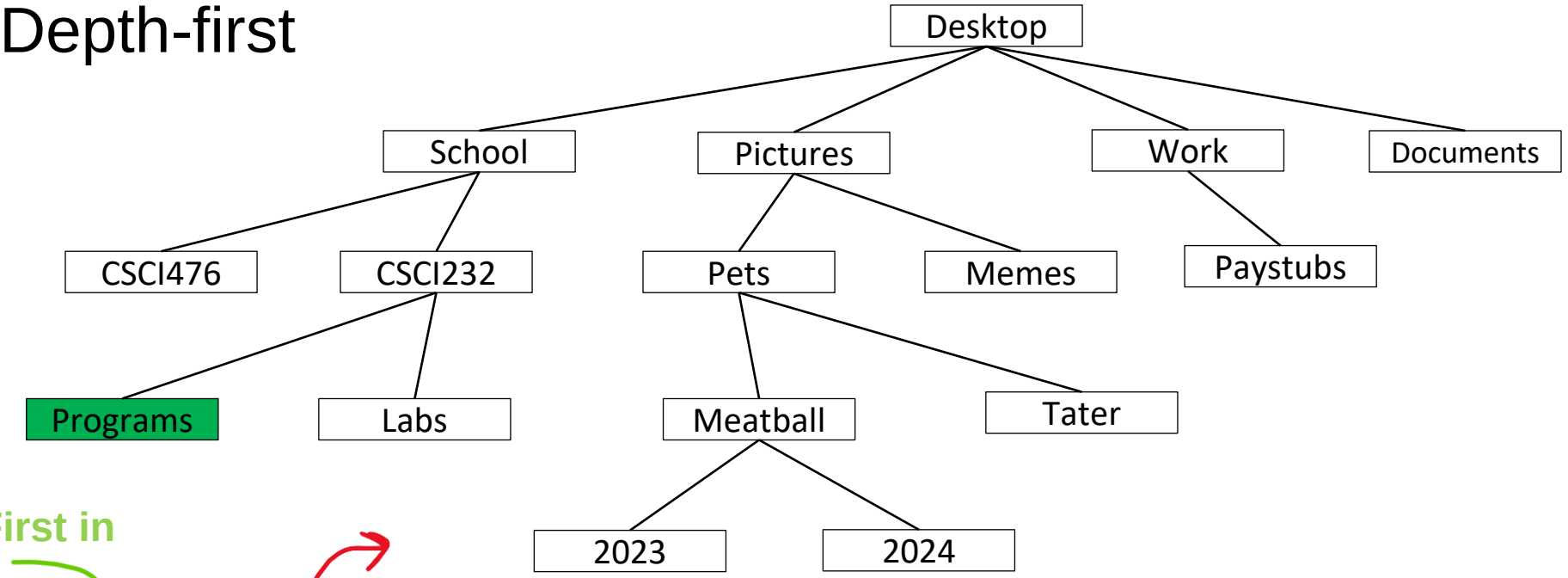
Labs
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs

First in

Last out

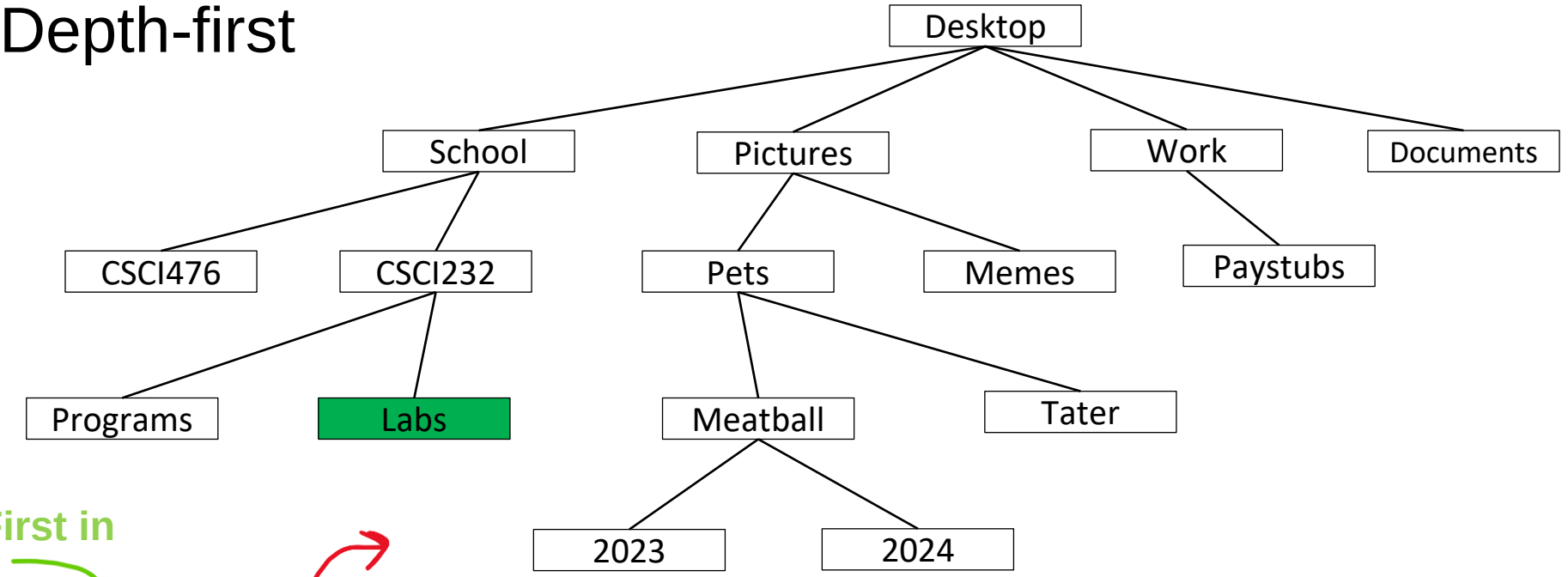
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs
Labs

First in

Last out

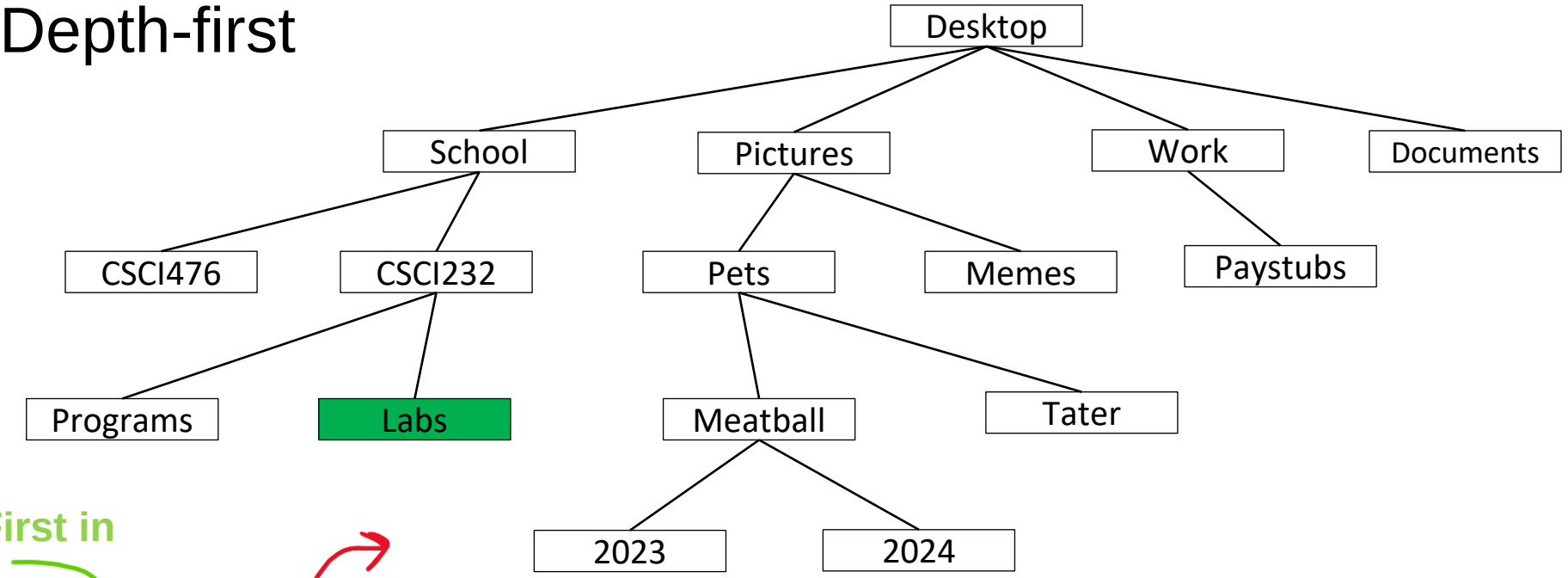
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs
Labs

First in

Last out

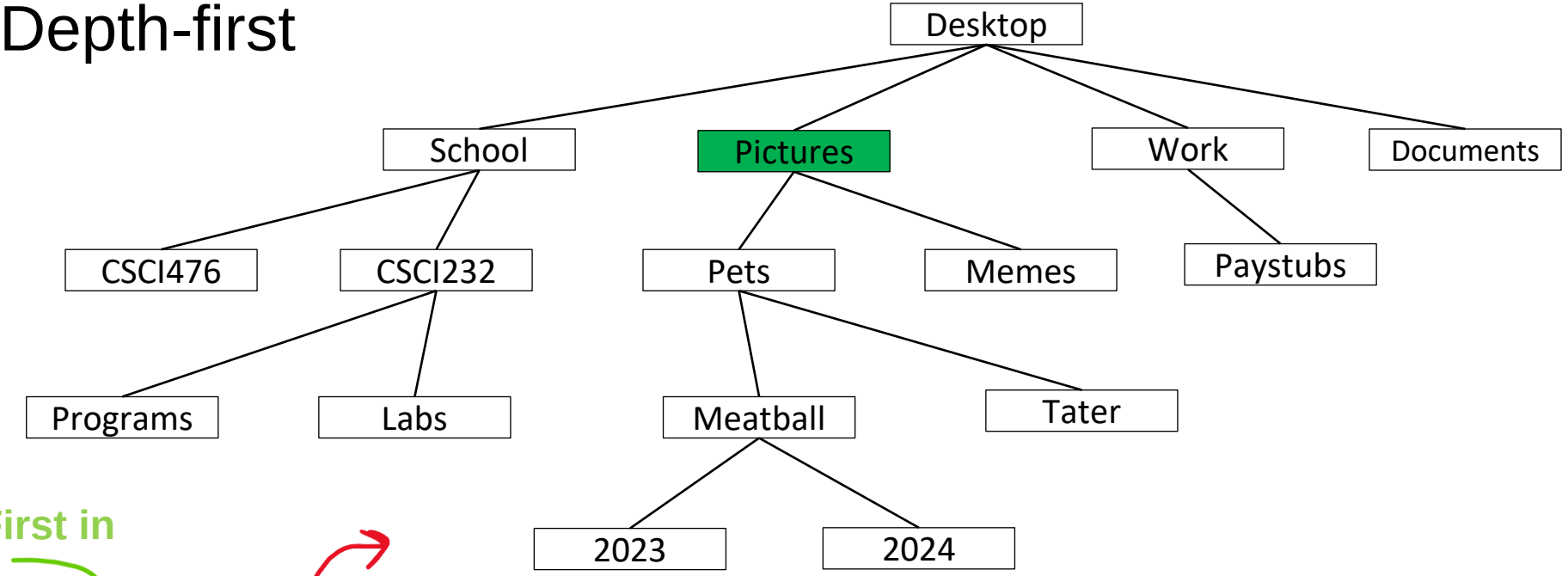
Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

Depth-first



Output

Desktop
School
CSCI476
CSCI232
Programs
Labs

First in

Last out

Pictures
Work
Documents

Stack

Every time we "visit" a node we:

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

… and so on

Depth First

```
public void depthFirst(){




}
```

Every time we “visit” a node we:

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

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();









}
```

Every time we “visit” a node we:

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

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);









    }

}
```

Every time we “visit” a node we:

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

Start at the root node

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){









                }

    }

}
```

Every time we “visit” a node we:

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

Keep looping as long as we have unvisited nodes in our stack

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){
                        Node remove = stack.pop()




                }

    }

}
```

Every time we “visit” a node we:

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

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){
                        Node remove = stack.pop();
                        System.out.println(……);



                }

    }

}
```

Every time we “visit” a node we:

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

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){
                        Node remove = stack.pop();
                        System.out.println(……);



                }

    }

}
```

Every time we “visit” a node we:

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

Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){
                        Node remove = stack.pop();
                        System.out.println(……);
                        for(Node c: remove.getChildren()){

                        }
                }

    }

}
```

Every time we “visit” a node we:

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

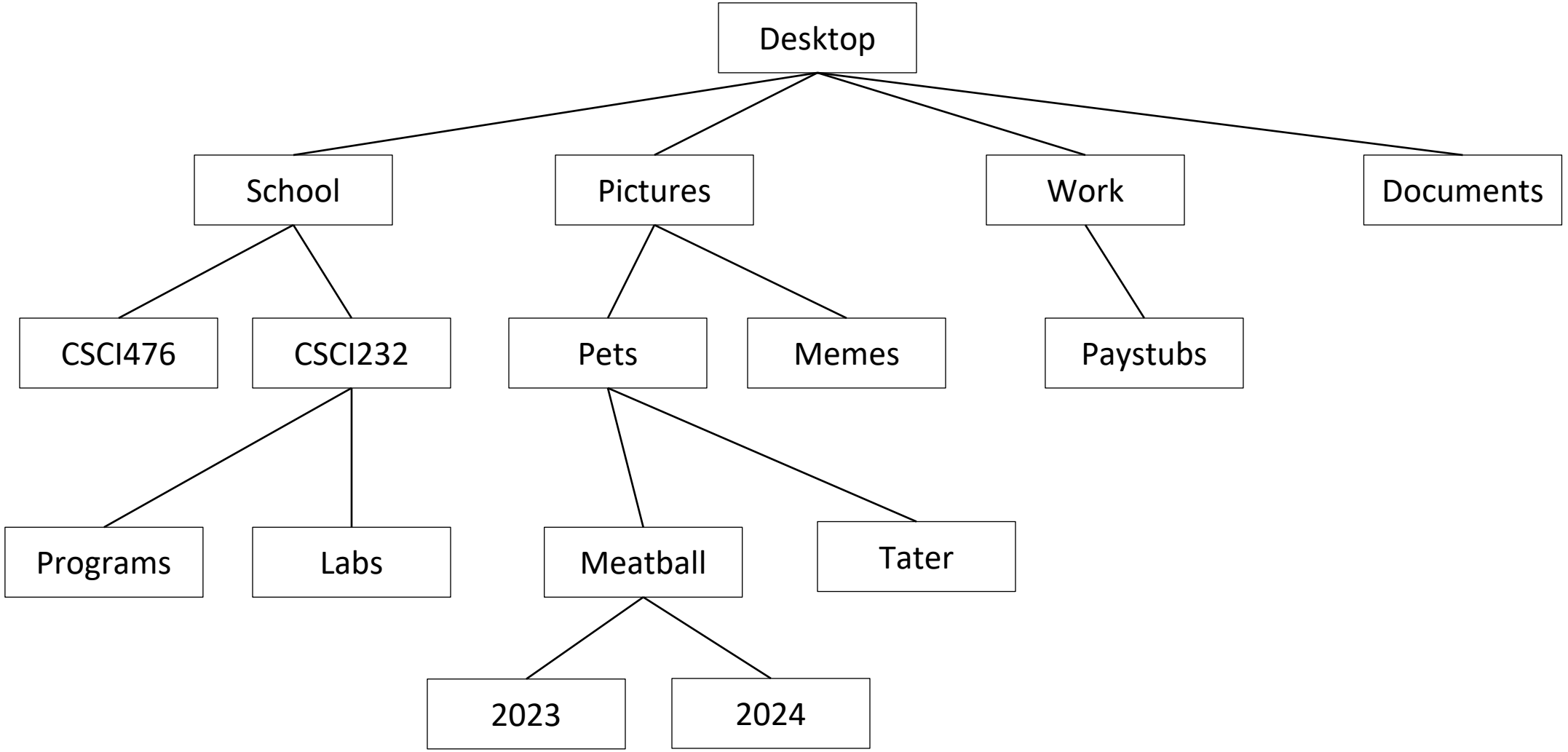
Depth First

```
public void depthFirst(){

        Stack<Node> stack = new Stack<Node>();

        if ( root != null){

                stack.add(root);

                while (!stack.isEmpty()){
                        Node remove = stack.pop();
                        System.out.println(……);
                        for(Node c: remove.getChildren()){
                                stack.push(c);
                        }
                }

    }

}
```

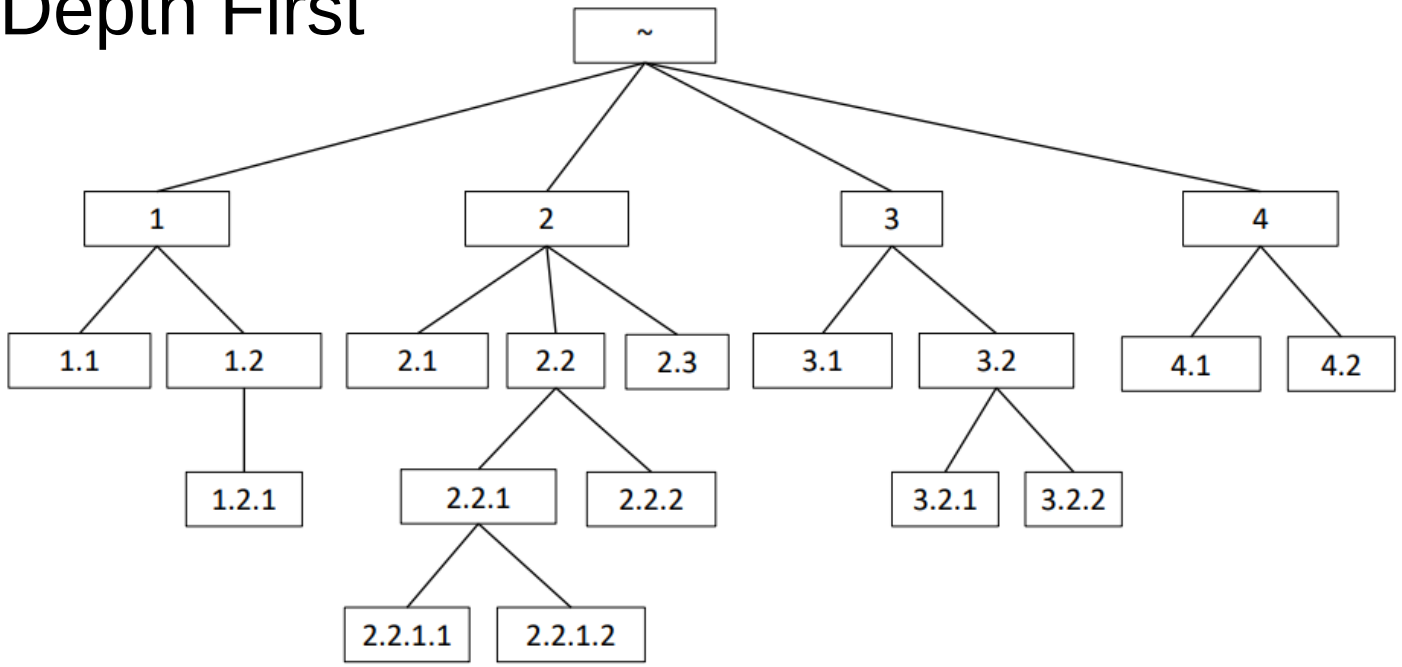
Every time we “visit” a node we:

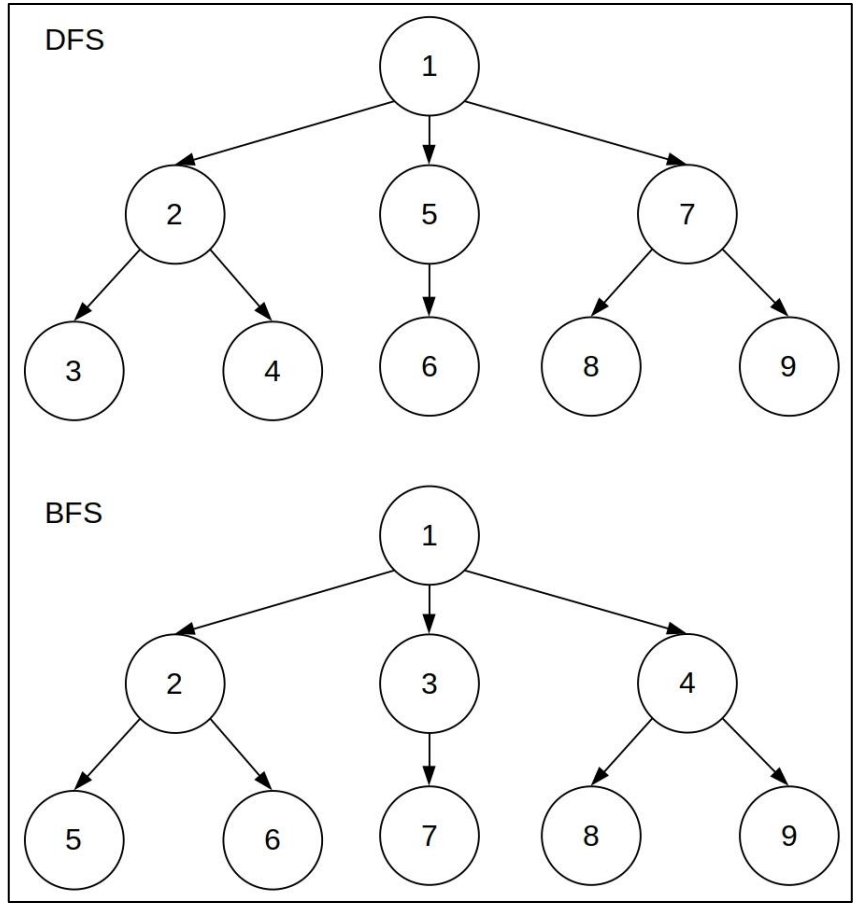
1. Remove node from stack
2. Execute the action (print, compare, etc)
3. Push all children to the stack

Let's code this!


Desktop
School
Pictures
Work
Documents
CSCI476
CSCI232
Pets
Memes
Paystubs
Programs
Labs
Meatball
Tater
2023
2024

Depth First


~
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


DFS
1
2
5
7
3
4
6
8
9
BFS
1
2
3
4
5
6
7
8
9