جامعة بيرزيت
BIRZEIT UNIVERSITY

Stacks

Dr. Abdallah Karakra

Computer Science Department

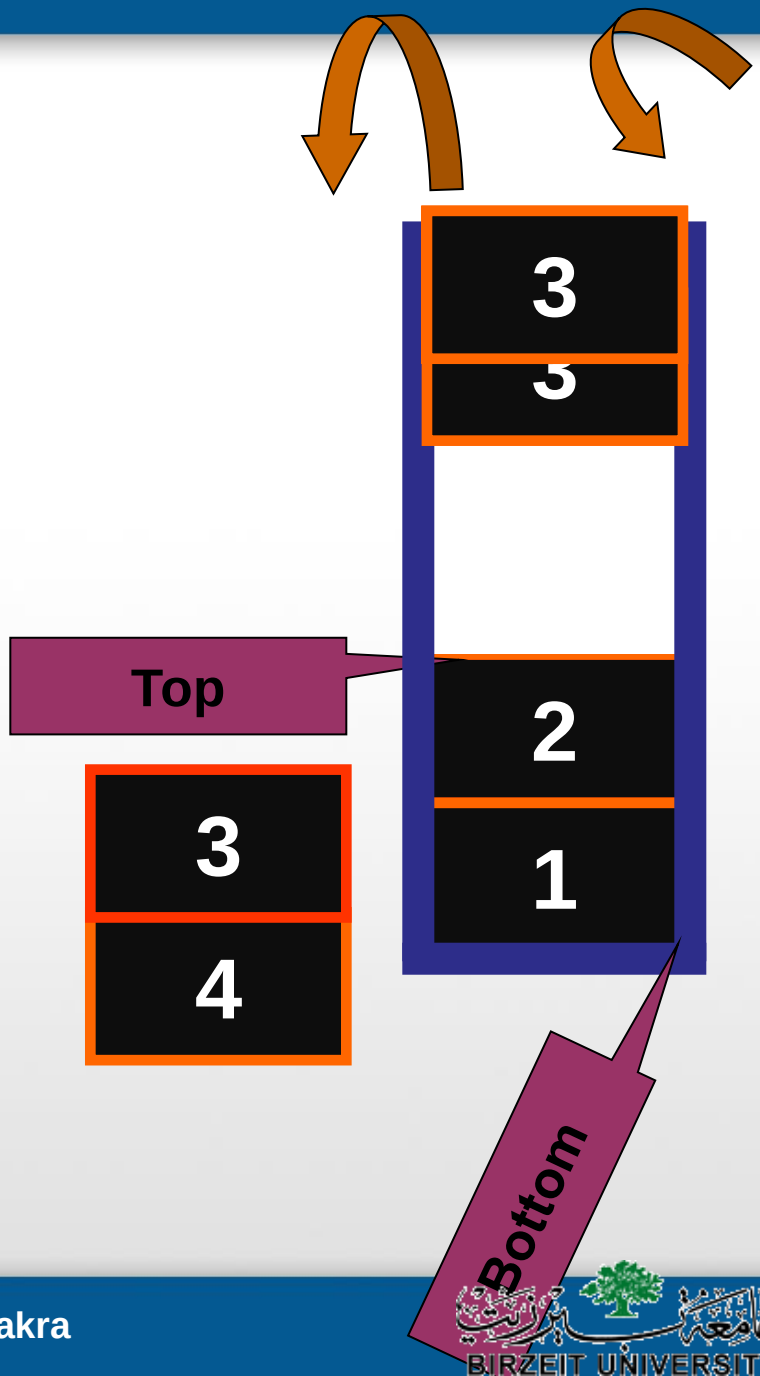
COMP242

Tuesday, March 26, 2024

Stack

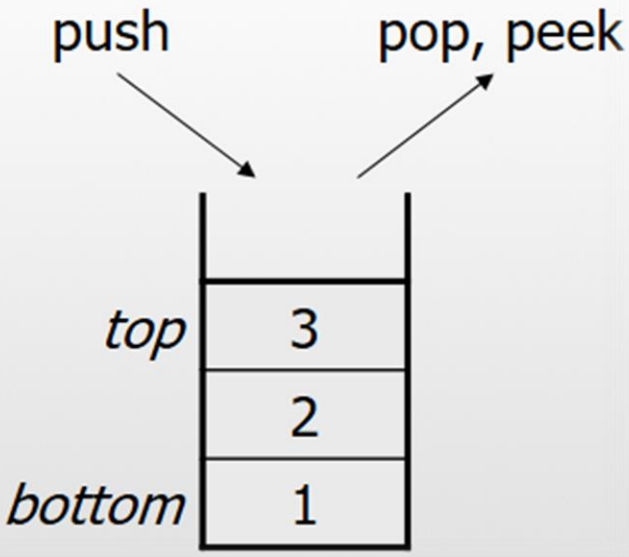
A *stack* is a data structure in which all insertions and deletions of entries are made at one end, called the *top* of the stack. The last entry which is inserted is the first one that will be removed.

A stack is a container of objects that based on the principle of adding elements and retrieving them in the opposite order.



Stack

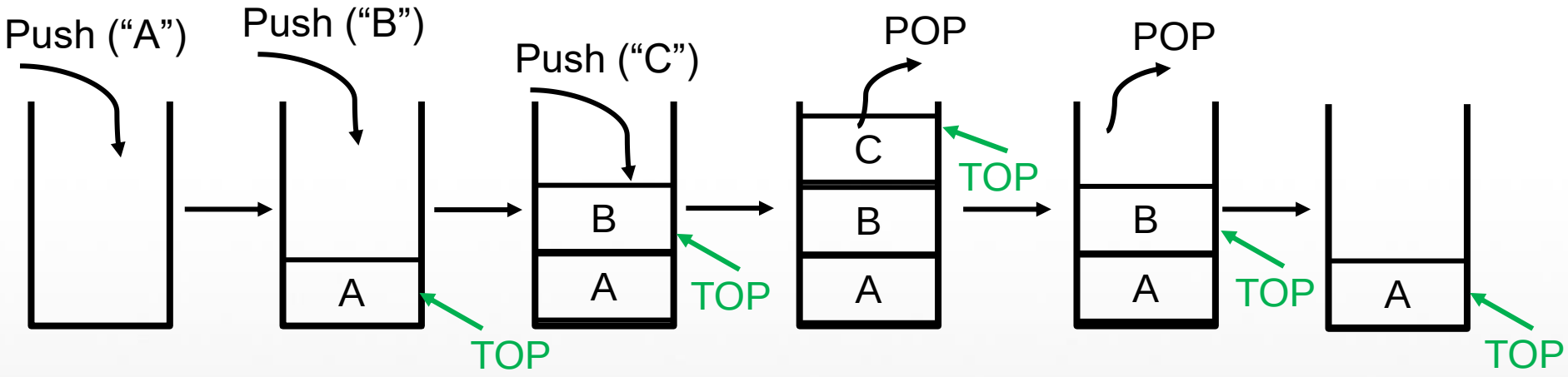
- Last-In, First-Out ("LIFO")
- Elements are stored in order of insertion.
  We do not think of them as having indexes.
- Client can only add/remove/examine the last element added (the "top").



❖ Basic stack operations:

push: Add an element to the top.
pop: Remove the top element.
peek: Examine the top element.

Basic stack Operations



Top= -1
Size= 0

Top= 0
Size= 1
Peek=A

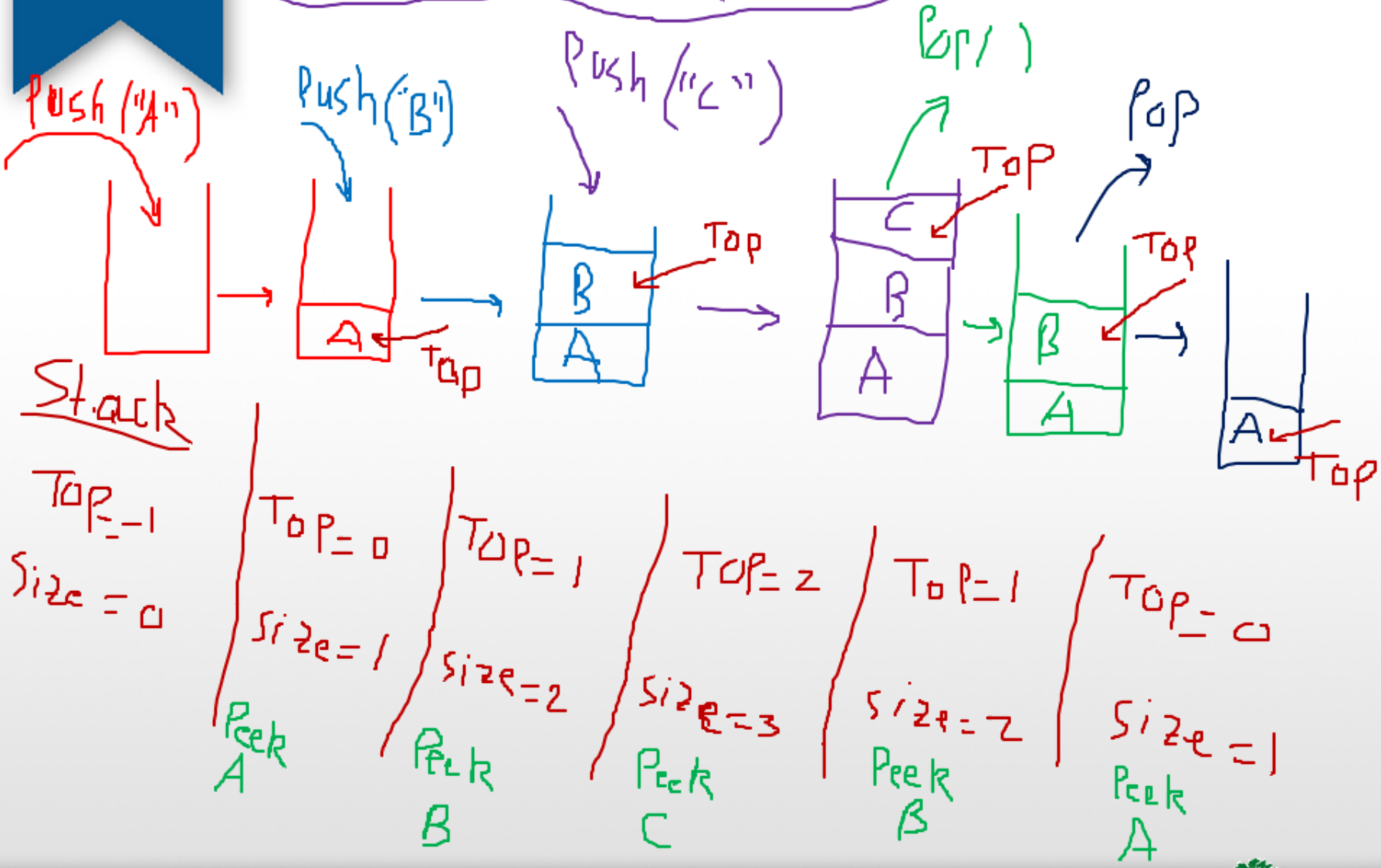
Top= 1
Size= 2
Peek=B

Top= 2
Size= 3
Peek=C

Top= 1
Size= 2
Peek=B

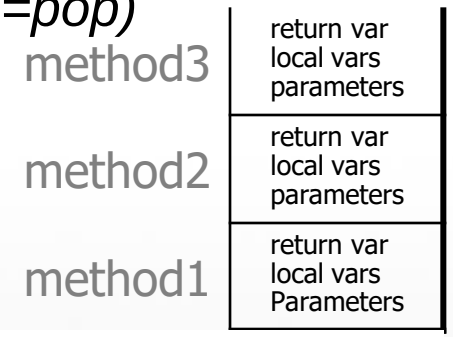
Top= 0
Size= 1
Peek=A

Basic Stack Operations

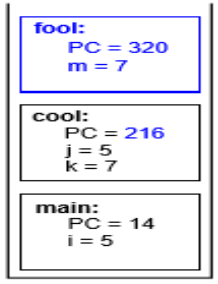


Stacks in Computer Science

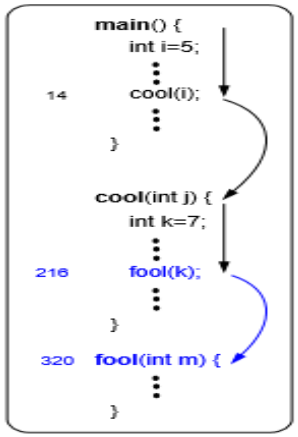
- Programming languages and compilers:
  - method calls are placed onto a stack *(call=push, return=pop)*
  - compilers use stacks to evaluate expressions



- Matching up related pairs of things:
  - find out whether a string is a palindrome
  - examine a file to see if its braces { } match
  - convert "infix" expressions to pre/postfix



Java Stack



Java Program

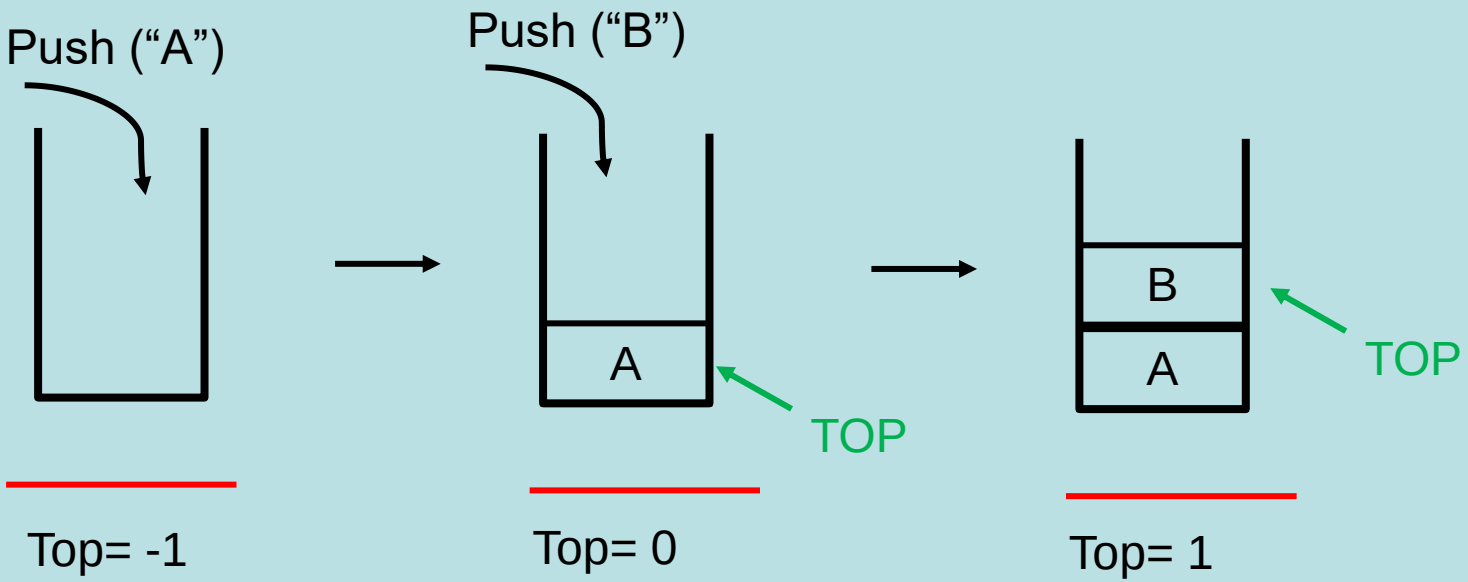
- Sophisticated algorithms:
  - many programs use an "undo stack" of previous operations

Stack: Array-Based Implementation

```
public class Stack {

   private int maxSize;
   private Object[] stackArray;// Holds the elements
   private int top;

   public Stack(int maxSize) {

      this.maxSize = maxSize;
      stackArray = new Object[maxSize];
      top = -1; //Empty stack
   }
       /* Methods go here */

}
```

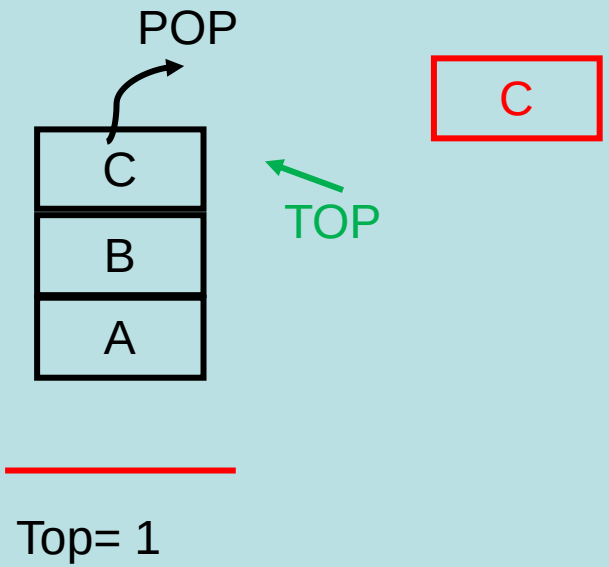
Stack: Array-Based Implementation

```
public void push(Object element){
  // Adds a new element to the  top of the stack
    stackArray[++top] = element;
}
```



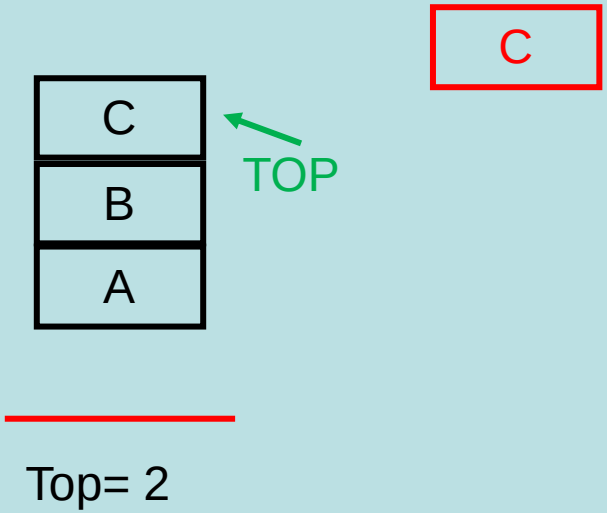
Stack: Array-Based Implementation

```
public Object pop(){
  // Removes and returns the stack's top element
  if (!isEmpty())
      return stackArray[top--];
  return null; //Empty stack
}
```



Stack: Array-Based Implementation

```
public Object peek(){
    // Return the top element without changing the stack
   if (!isEmpty())
      return stackArray[top];
   return null; //Empty stack
}
```



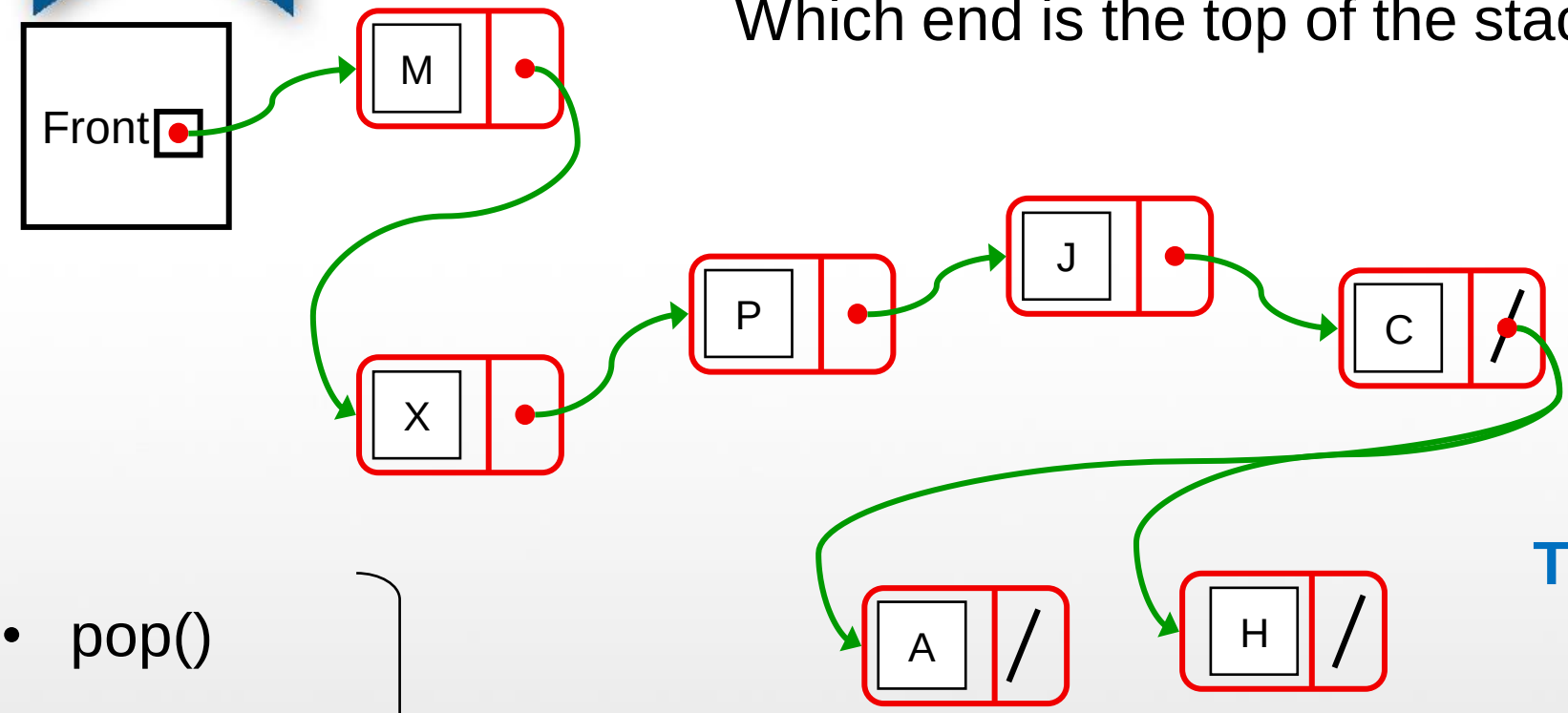
Stack: Array-Based Implementation

```
public void push(Object element){
  // Adds a new element to the  top of the stack
    stackArray[++top] = element;
}

public Object pop(){
  // Removes and returns the stack's top element
  if (!isEmpty())
      return stackArray[top--];
  return null; //Empty stack
}

public Object peek(){
    // Return the top element without changing the stack
   if (!isEmpty())
      return stackArray[top];
   return null; //Empty stack
}
```

Stack: Array-Based Implementation

```
public boolean isEmpty() {

    return (top == -1); // Returns true if stack is empty
}

public boolean isFull() {

   return (top == maxSize-1);  // Returns true if stack is full
}


public int size() {

    return top + 1; // returns number of elements inside stack
}
```

Stack: Linked List-Based Implementation



Which end is the top of the stack?

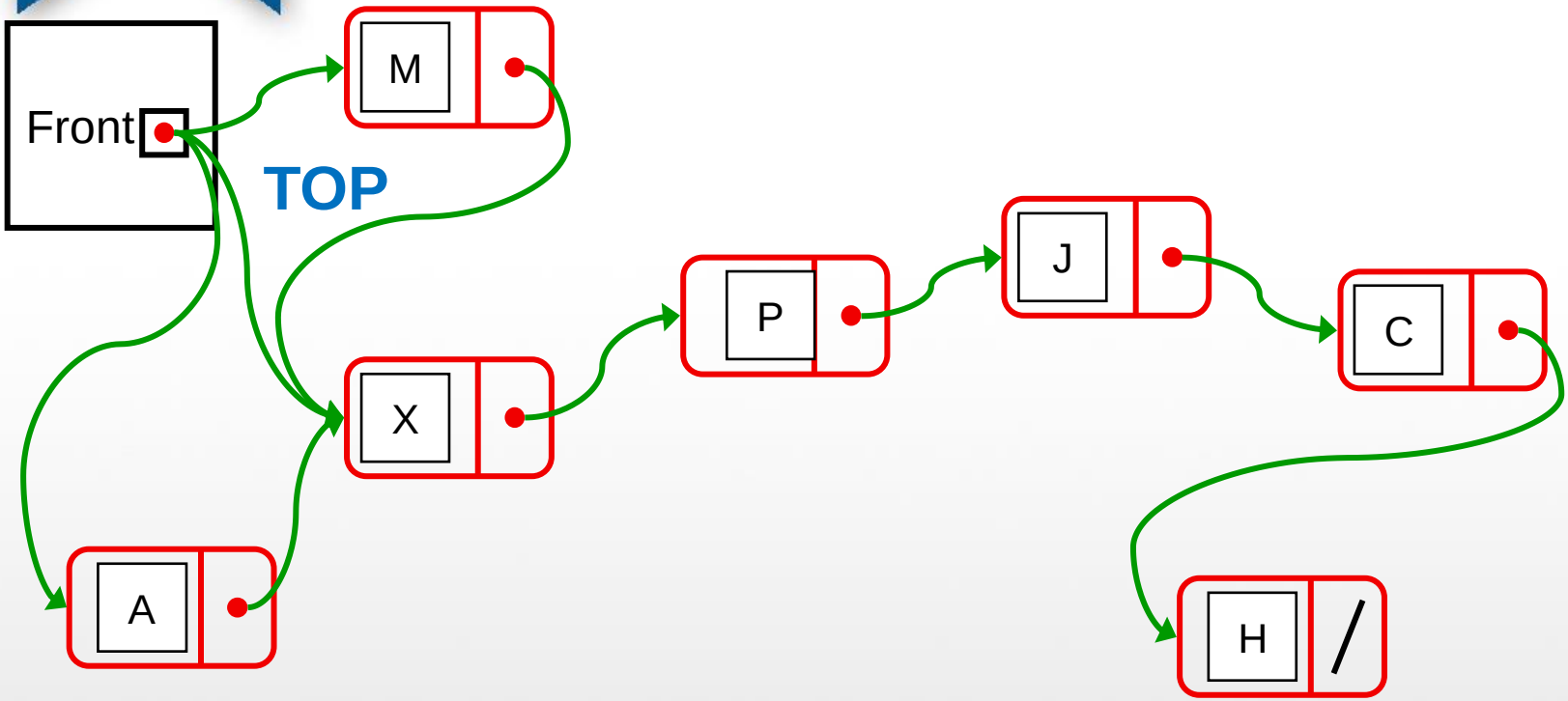
- pop()
- push("A")

O(n)

Why is this a bad idea?

Stack: Linked List-Based Implementation

- Make the top of the Stack be the head of the list.



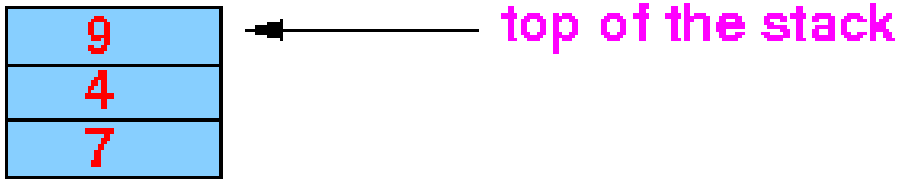
- pop()
- push("A")

Stack: Linked List-Based Implementation

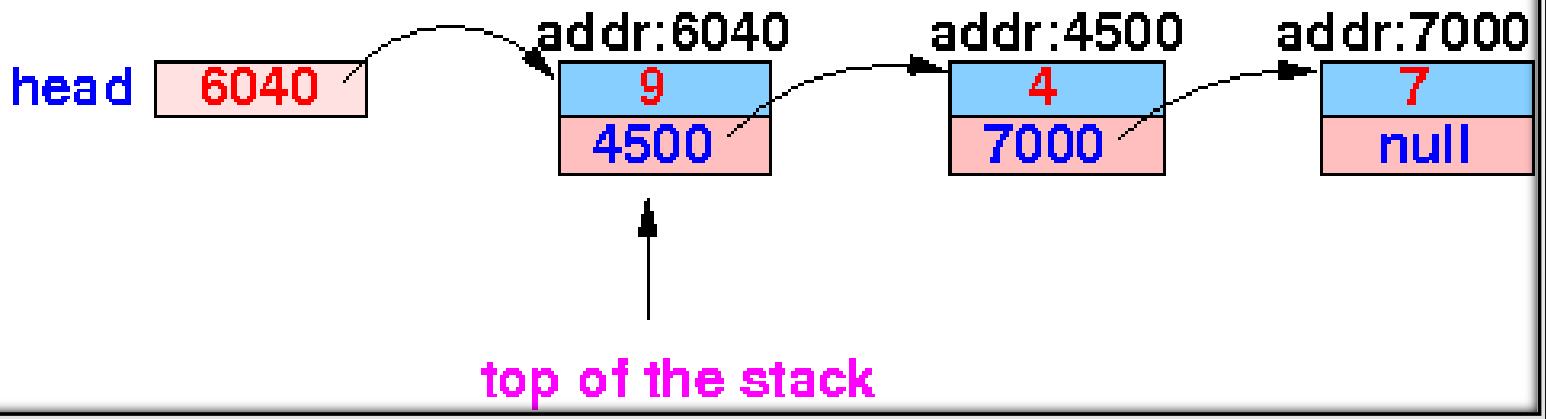
Representing a stack with a list:

7, 4, 9

Stack:



List:



Stack: Linked List-Based Implementation

Node Class

```
public class Node {

    public Object element;
    public Node next;

    public Node(Object element) {
      this(element, null);
    }

   public Node(Object element, Node next) {
     this.element = element;
     this.next = next;
   }
}
```

Stack: Linked List-Based Implementation

Stack Class

```
public class Stack {

    private int size; //number of elements in the stack
    private Node Front; // pointer to the top node

    public Stack (){
        //empty stack
      Front=null;
      size=0;
    }
    /* Methods go here */
}
```

Stack: Linked List-Based Implementation

```
 // Adds a new element to the top of the stack
public void push(Object element){

    Node newNode;
    newNode=new Node(element);

    newNode.next=Front;
    Front=newNode;

    size++;// update size
}
```

Stack: Linked List-Based Implementation

```
// Removes and returns the stack's top element
public Object pop() {

 if (!isEmpty()) {

    Node top = Front; //save reference
    Front = Front.next;// Remove first node
    size--;
    return top.element;//Return the element from the saved ref
 }
 else
  return null;

}
```

Stack: Linked List-Based Implementation

```
public Object peek(){
// Return the top element without changing the stack
  if (!isEmpty())
     return Front.element;
  else
    return null;
}

public int Size(){
   return size;
}

public boolean isEmpty(){

   return (Front==null); // return true if the stack is empty
}
```