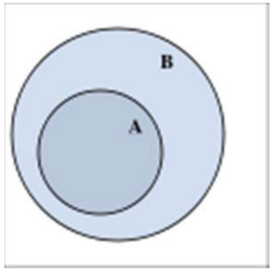
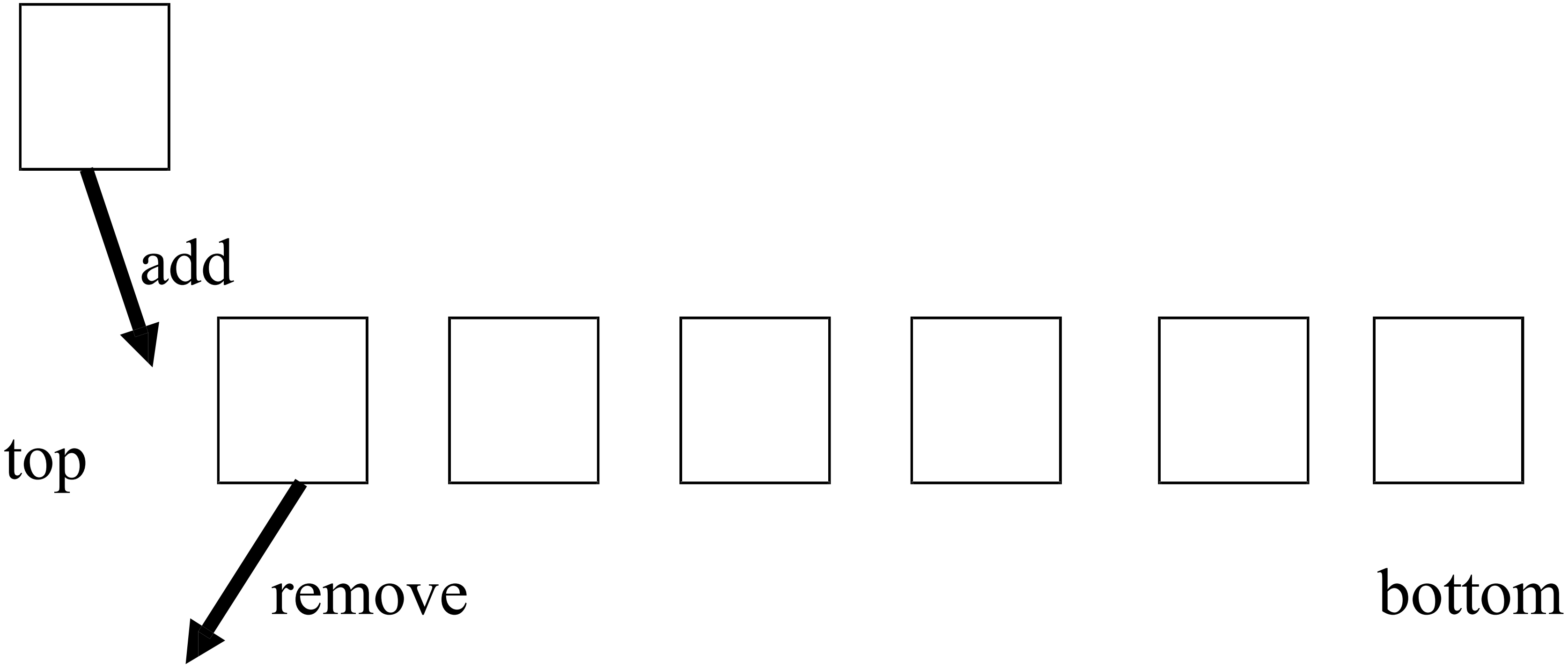
SCC120 Fundamentals of Computer Science
Unit 2: Stacks



Jidong Yuan
yuanjd@bjtu.edu.cn

“Stack” Abstract Data Type

The Stack ADT



The Stack ADT

- A stack is
  - a dynamic collection in which, when we remove an element, we remove the one which was most recently added
  - a ***last in first out*** structure
  - a linear structure in which items are added and removed at the same end - called the *top* of the stack
  - you are not supposed to remove things from the middle or bottom of a stack
  - like a stack of books or plates

Applications of the Stack ADT

- Stacks are widely used in computer science
- Consider subroutines or methods calling each other
  - main calls A calls B calls C
  - when method C is being executed, we need to remember
  - where we came from (the return address) in B
  - and where we came from in A
  - and where we came from in main
- This is naturally held in a stack, the *run-time* stack

Applications of the Stack ADT

- Apart from the return address, we may hold other information in the stack
  - arguments, local variables, etc.
- Note in particular the use of the stack for recursive method calls

Applications of the Stack ADT

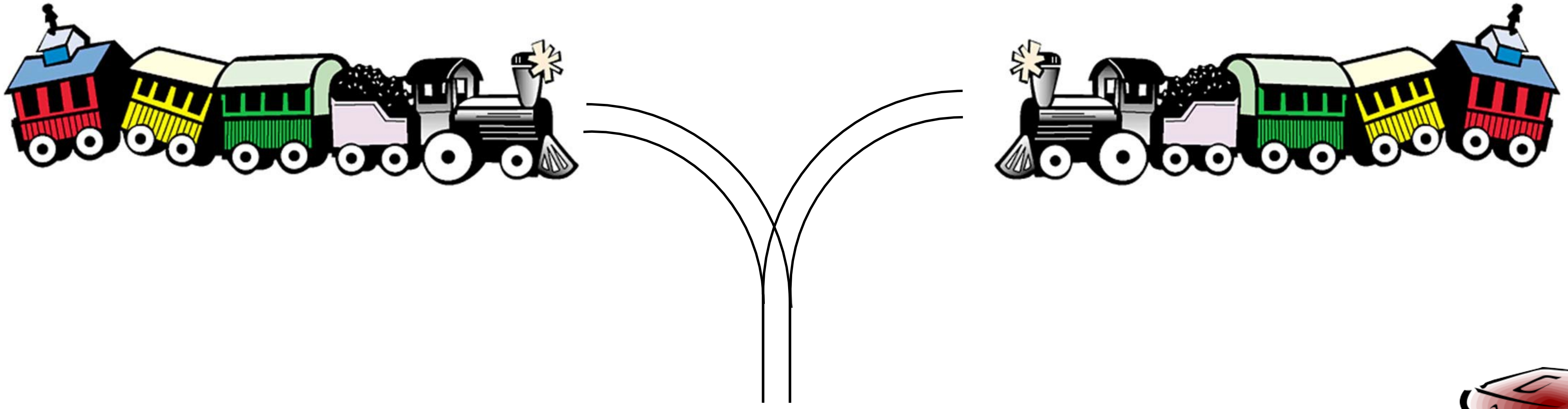
- Suppose we are searching, say for a way out of a maze
  - we reach a point where there are several alternatives…

Applications of the Stack ADT

- Suppose we are searching, say for a way out of a maze
  - we reach a point where there are several alternatives
  - we make a choice of one path to pursue and we follow that
  - on this path we reach another point where there are several alternatives
  - we make a choice of one path to pursue and we follow that

Applications of the Stack ADT

- a stack can be used to reverse a set of values
- put all the values in the stack one-by-one
- then remove them all one-by-one



Stack Operations

- Add the specified element onto the top of the stack
  - called *push*
  - unlike a set, the same element can be added more than once
- Remove the top element from the stack
  - called *pop*
  - fails if the stack is empty

A Stack ADT Using a Linear Array

stack

22	44	22	33		
0	1	2	3	4	5

top

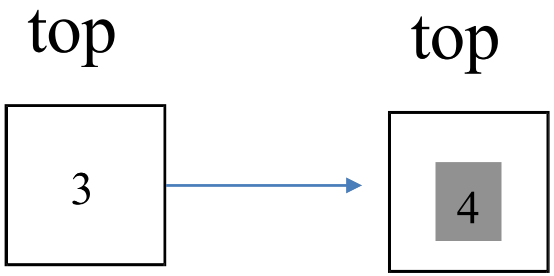
3

initially top is -1
(no elements in the stack)

After push(66)

stack

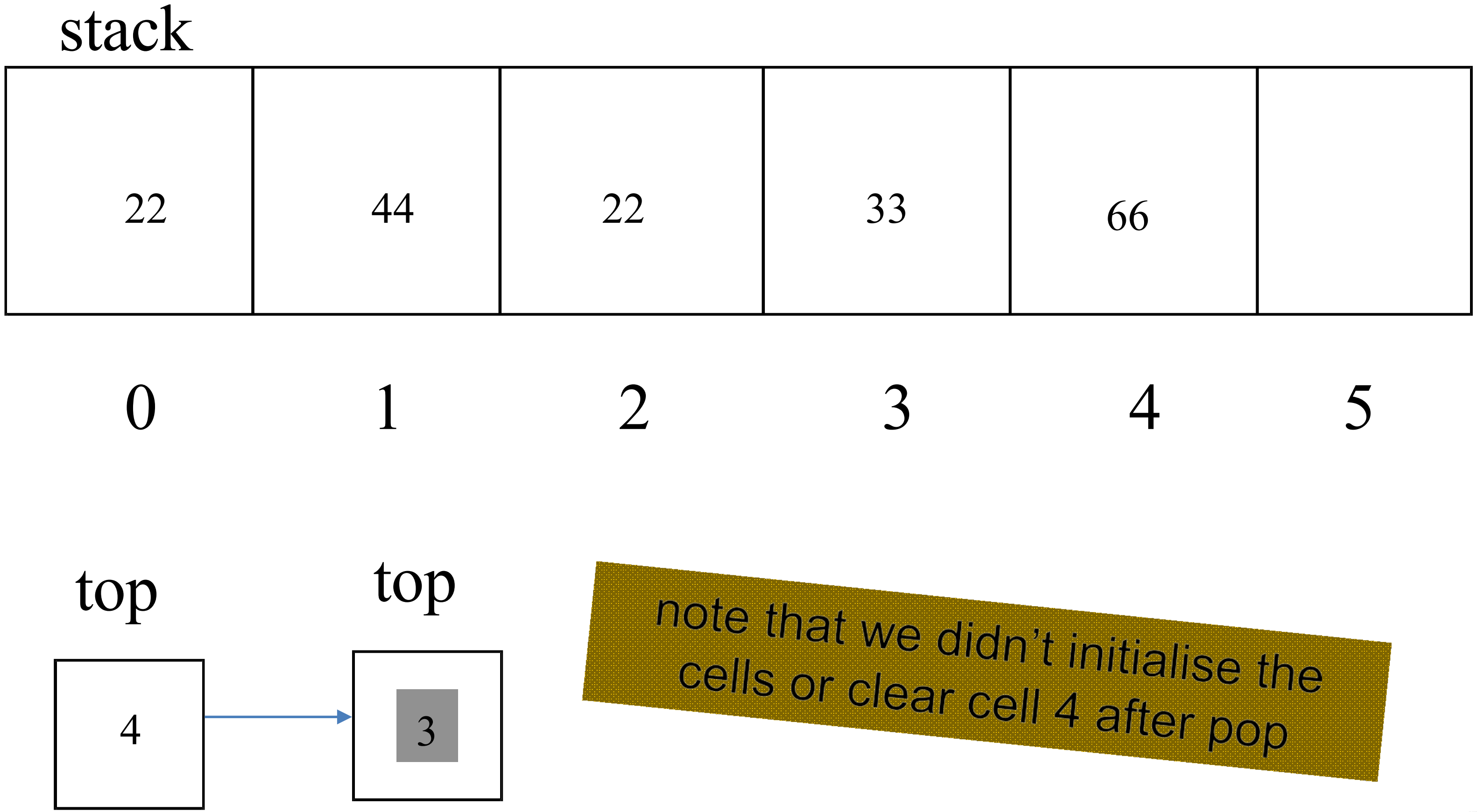
22	44	22	33	66	
0	1	2	3	4	5



The push Method

```
if (top == limit - 1)
    PROBLEM - STACK FULL
else
{
    top++ ;
    stack[top] = X ;
}
```

After pop
[returns 66]



stack

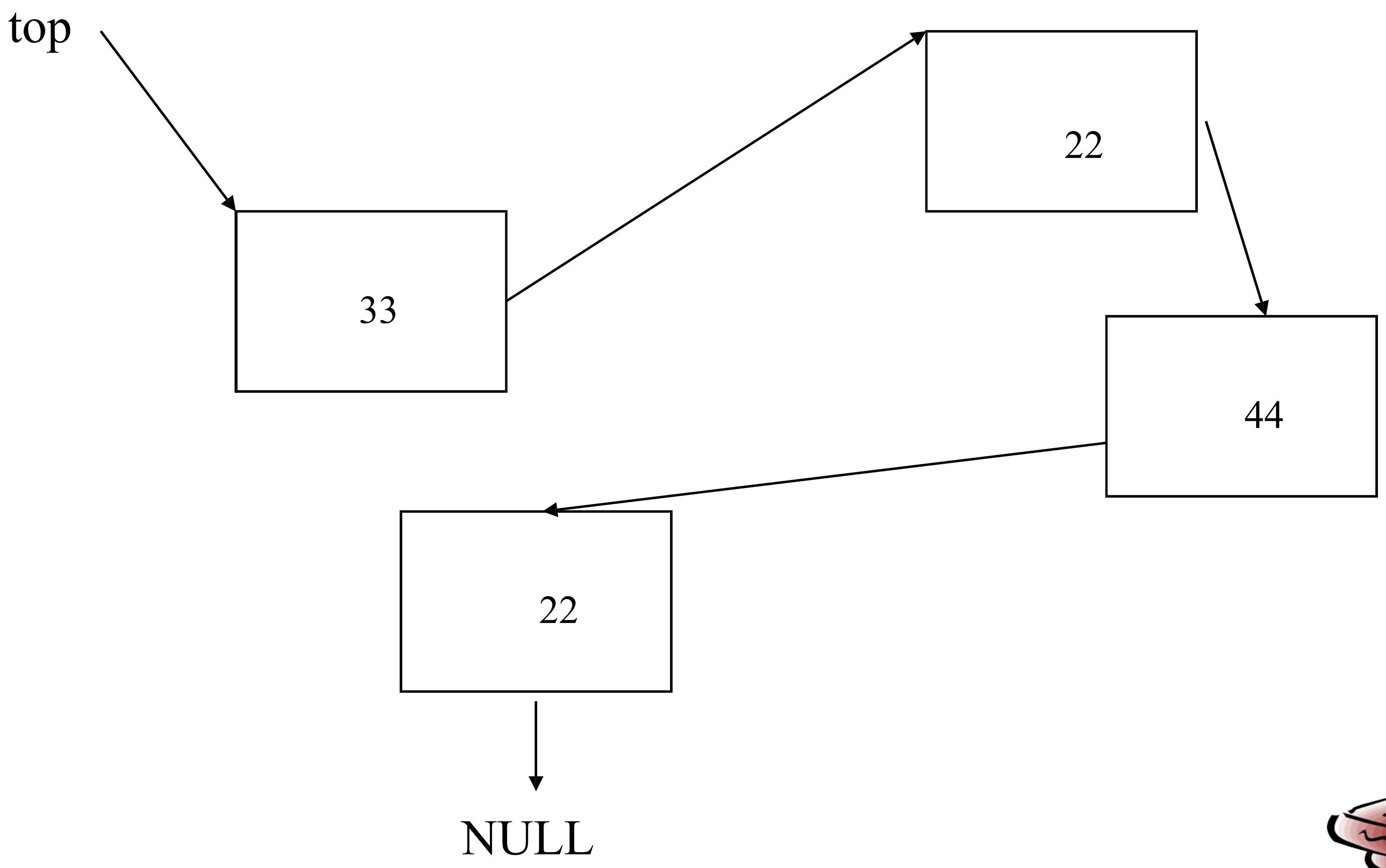
0	1	2	3	4	5
22	44	22	33	66	

note that we didn't initialise the cells or clear cell 4 after pop

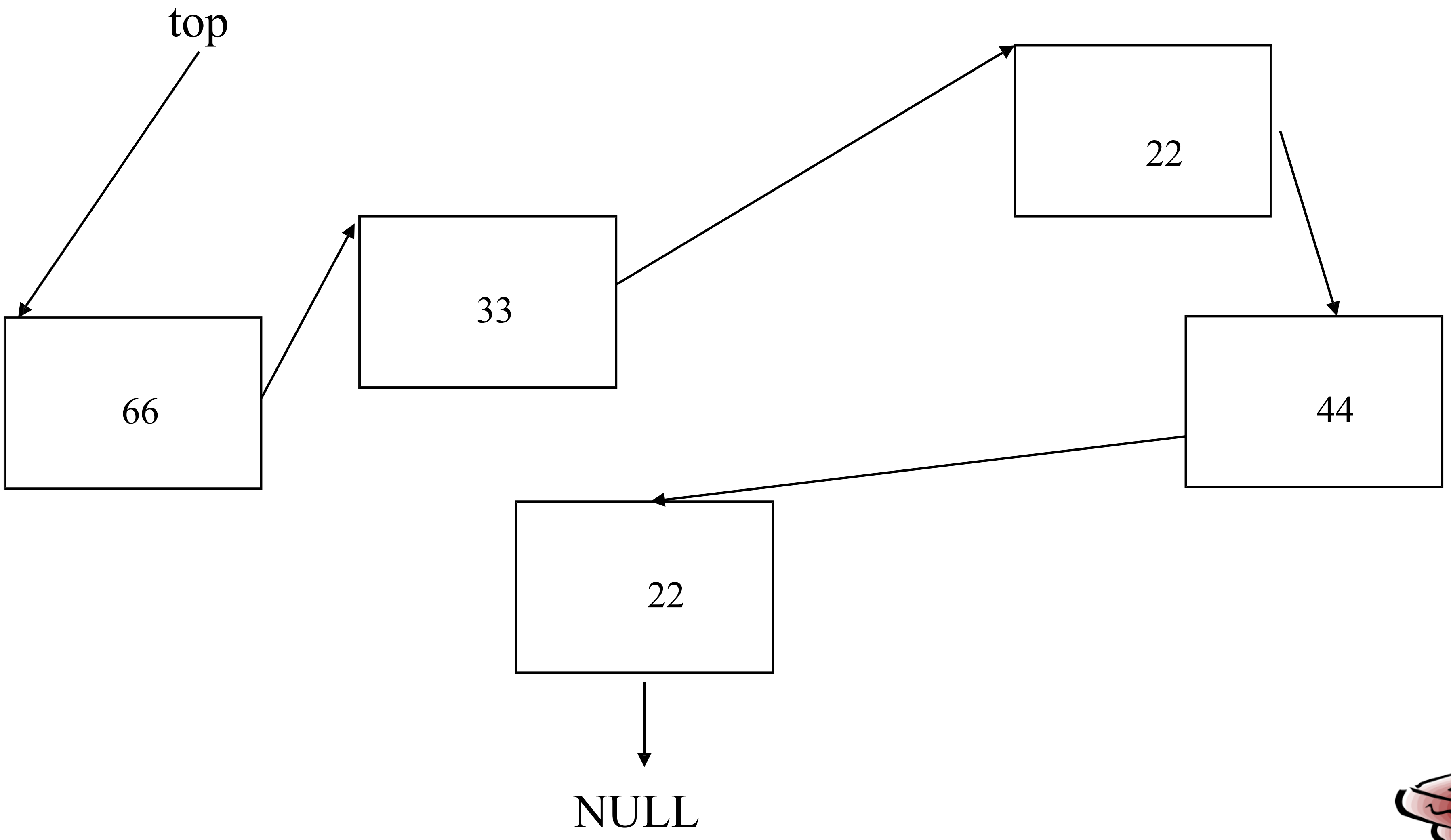
The pop Method

```
if (top == -1)
    PROBLEM - STACK EMPTY
else
{
    temp = stack[top] ;
    top-- ;
    return temp ;
}
```

A Stack ADT Using a Linked List



After push(66)

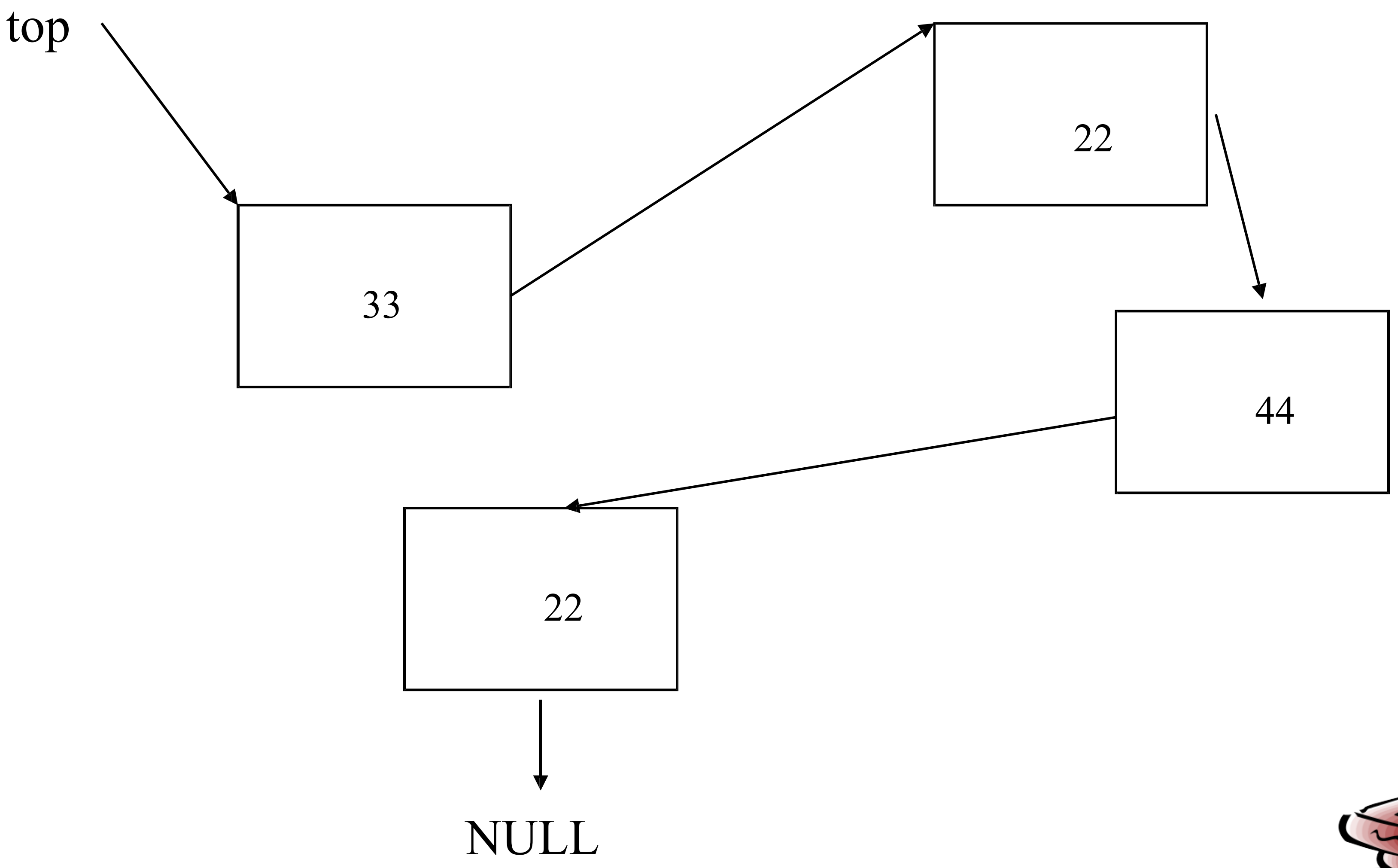


The push Method

```
StackCell temp = new StackCell(X, null) ;
temp.next = top ;
top = temp ;
```

- Unlike the array implementation, there is no size restriction
  - the stack can keep growing (until the available memory runs out)

After pop
[returns 66]



top

33

22

44

22

NULL

The pop Method

```
if (top == null)
    PROBLEM - STACK EMPTY
else
{
    int X = top.data ;
    top = top.next ;
    return X ;
}
```

note that we didn't clear StackCell after pop

Efficiency of the Stack Implementations

- In contrast to the Set ADT, none of the push/pop operations involves doing a scan
  - they are all fast, constant-time, O(1) operations

Efficiency of the Stack Implementations

- a *size* operation in the array implementation is also fast, O(1)
  - just return the value of “top + 1”
- a *size* operation in the linked list implementation requires a scan, and so takes linear O(N) time
  - but we could record the size as well as the “top”, and update this when we push or pop
  - in this case, *size* can be constant or O(1) as well

A Stack Class

```
public class Stack
{
    public Stack() ;
    public void push(Element X) ;
    public Element pop() ;
    public boolean isEmpty() ;
    public int size() ;
    public Element top() ;
}
```

A Stack Class

- the first four methods must be available
  - a constructor to set up the stack; *push* and *pop*; *isEmpty* to check it is safe to do pop
- possible further methods
  - *size*; and *top* to return the top element without popping it
- *pop* and *top* should return some error message if stack is empty

A Stack Class

- the implementation is hidden
  - we can’t tell if it is an array, linked list, or something else

Interesting point on Set ADT vs. Stack ADT

- To delete an element from a Set ADT, we have to specify which element to remove
  - so the remove operation requires an argument specifying the value to remove
- To pop an element from a Stack ADT, we do not specify the element to remove because there is no alternative
  - we can remove only the top element (if there is one)
  - so pop has no argument

SCC120 ADT (weeks 5-10)

- Week 5 Abstractions; Set

  Stack (push and pop operations, implementations with arrays or linked lists)
- Week 6
- Week 7
- Week 8
- Week 9
- Week 10