Stack and Queue

Stack and Queue

- Data structures with specific purpose
  - Restricted features
- All operations are very efficient
  - Inefficient operations are restricted
- We'll a stack and queue using linked lists
- *Scala has builtin Stack and Queue classes

Stack

- LIFO
  - Last in First out
  - The last element pushed onto the stack is the first element to be popped off the stack
- Only the element on the top of the stack can be accessed

Stack Methods

- Push
  - Add an element to the top of the stack
- Pop
  - Remove the top element of the stack

Stack Implementation

- Implement a Stack class by wrapping a linked list
- Stack uses the linked list and adapts its methods to implement push and pop

```
class Stack[A] {

  var top: LinkedListNode[A] = null

  def push(a: A): Unit = {
    this.top = new LinkedListNode[A](a, this.top)
  }

  def pop(): A = {
    val toReturn = this.top.value
    this.top = this.top.next
    toReturn
  }

}
```

Stack Usage

- Create a new empty Stack
- Call push to add an element to the top
- Call pop to remove an element
- Same exact usage when using Scala's builtin Stack

```scala
val stack = new Stack[Int]()
stack.push(3)
stack.push(7)
stack.push(2)
stack.push(-5)

val element = stack.pop()
```

Stack Usage

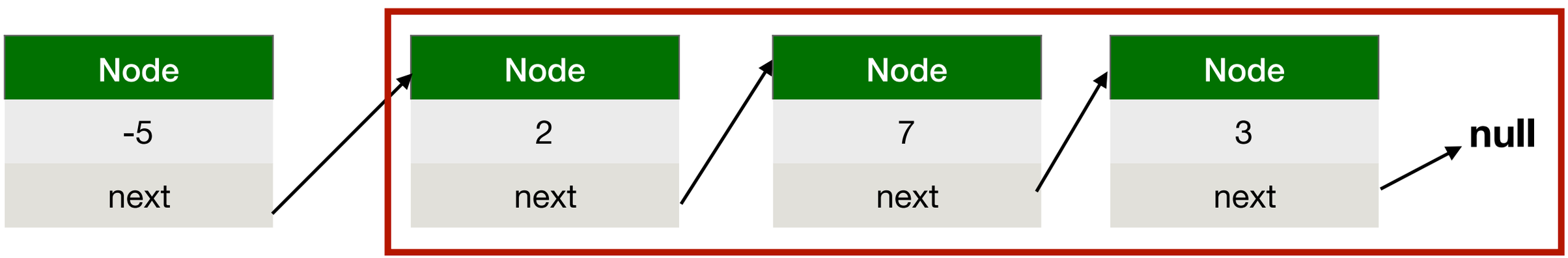
- We can use Scala's list as a Stack
  - The preferred way to use the concept of a stack in practice
- This is very efficient!
- But wait.. doesn't this create a new list each time an element is pushed or popped since List is immutable (cannot change)?
  - No.. well, kind of

```
var stack = List[Int]()
stack = 3 :: stack
stack = 7 :: stack
stack = 2 :: stack
stack = -5 :: stack

val element = stack.head
stack = stack.drop(1)
```

Stack Usage

- Before -5 is pushed, the stack is equal to nodes in the red box
- After pushing -5, the red box is unchanged
- A new List **is** returned, but it reuses the old List
- No need to recreate the entire List

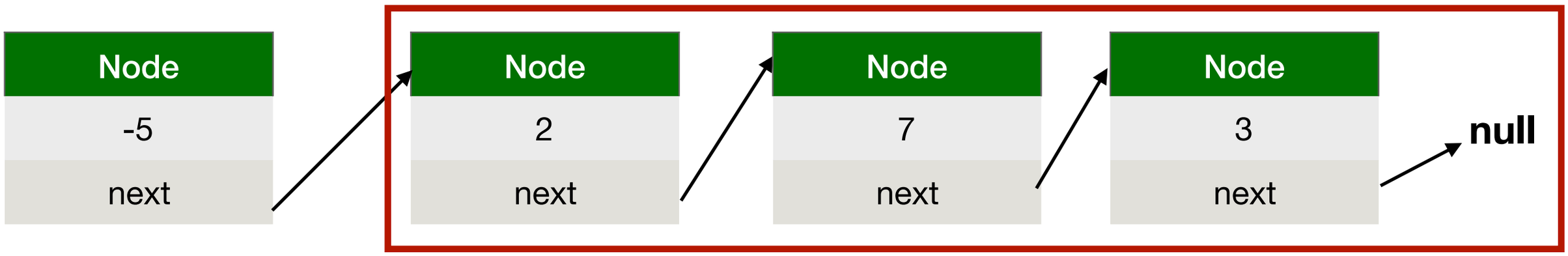
```
var stack = List[Int]()
stack = 3 :: stack
stack = 7 :: stack
stack = 2 :: stack
stack = -5 :: stack

val element = stack.head
stack = stack.drop(1)
```



Stack Usage

- Same efficiency when -5 is popped
- The red box never changed, but we update the reference stored in the stack variable
- Other parts of the program can share parts of a List without having their changes affect each other

```
var stack = List[Int]()
stack = 3 :: stack
stack = 7 :: stack
stack = 2 :: stack
stack = -5 :: stack

val element = stack.head
stack = stack.drop(1)
```



Queue

- FIFO
  - First in First out
  - The first element enqueued into the queue is the first element to be dequeued out of the queue
- Elements can only be added to the end of the queue
- Only the element at the front of the queue can be accessed

Queue Methods

- Enqueue
  - Add an element to the end of the queue
- Dequeue
  - Remove the front element in the queue

Queue Implementation

- Implement a Queue class by wrapping a linked list
- Queue needs a reference to the first and last element

```
class Queue[A] {

  var front: LinkedListNode[A] = null
  var back: LinkedListNode[A] = null

  def enqueue(a: A): Unit = {
    if (back == null) {
      this.back = new LinkedListNode[A](a, null)
      this.front = this.back
    } else {
      this.back.next = new LinkedListNode[A](a, null)
      this.back = this.back.next
    }
  }

  def dequeue(): A = {
    val toReturn = this.front.value
    this.front = this.front.next
    if(this.front == null){
      this.back = null
    }
    toReturn
  }

}
```

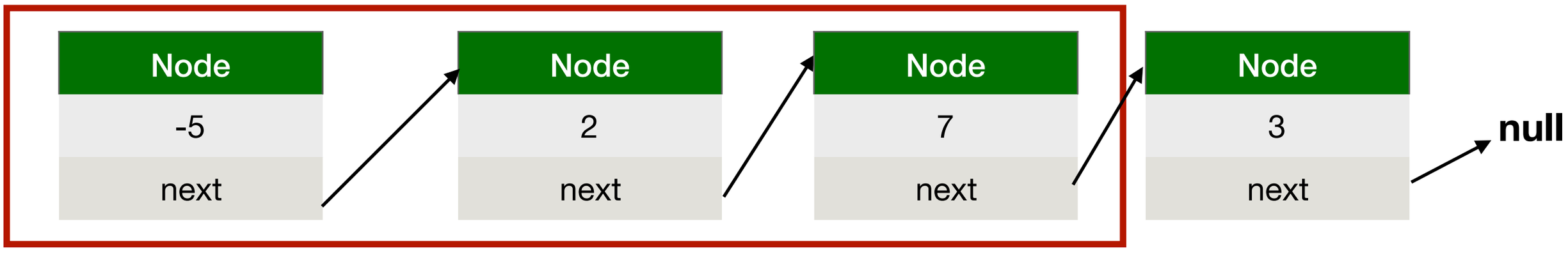
Queue Usage

- Create a new empty Queue
- Call enqueue to add an element to the back
- Call dequeue to remove the element at the front
- Same exact usage when using Scala's builtin Queue
  - [based on mutable List just like our implementation]

```
val queue = new Queue[Int]()
queue.enqueue(3)
queue.enqueue(7)
queue.enqueue(2)
queue.enqueue(-5)

val element = queue.dequeue()
```

Queue Usage

- No efficient way to use an immutable List as a queue
- The enqueue 3 the list in the red box must change
  - The next reference of the node containing 7 has to be updated
  - This List cannot be [should not be] used by other parts of the program since the List is changing



Lecture Question

Task: Free

- Submit anything

- You should study and practice using stacks and queues

* This question will be open until midnight