Software Development 3 (F27SG)

Lecture 7

Linked Lists operations

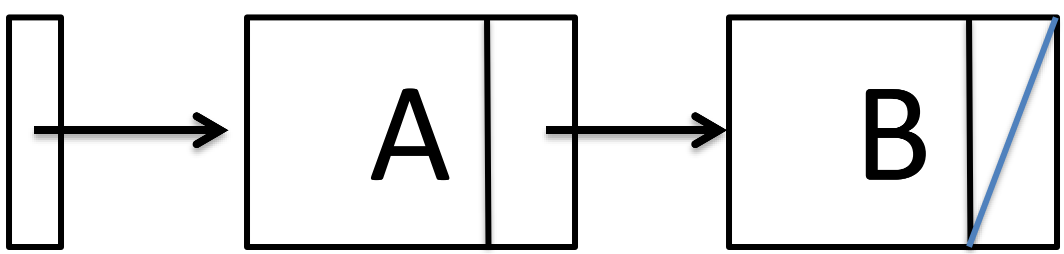
Rob Stewart)

Overview

- Adding and Removing Nodes from a Linked List
- Implementing Linked Lists

Linked Lists

- A Linked List is a **Linear** ordered sequence of **Nodes**
- Each **Node** contains
  - one **data item**
  - a **reference** to another Node of the same type
- We can traverse through the chain of nodes to visit each one in turn
  - No random access (unlike arrays)



Operations on Linked List

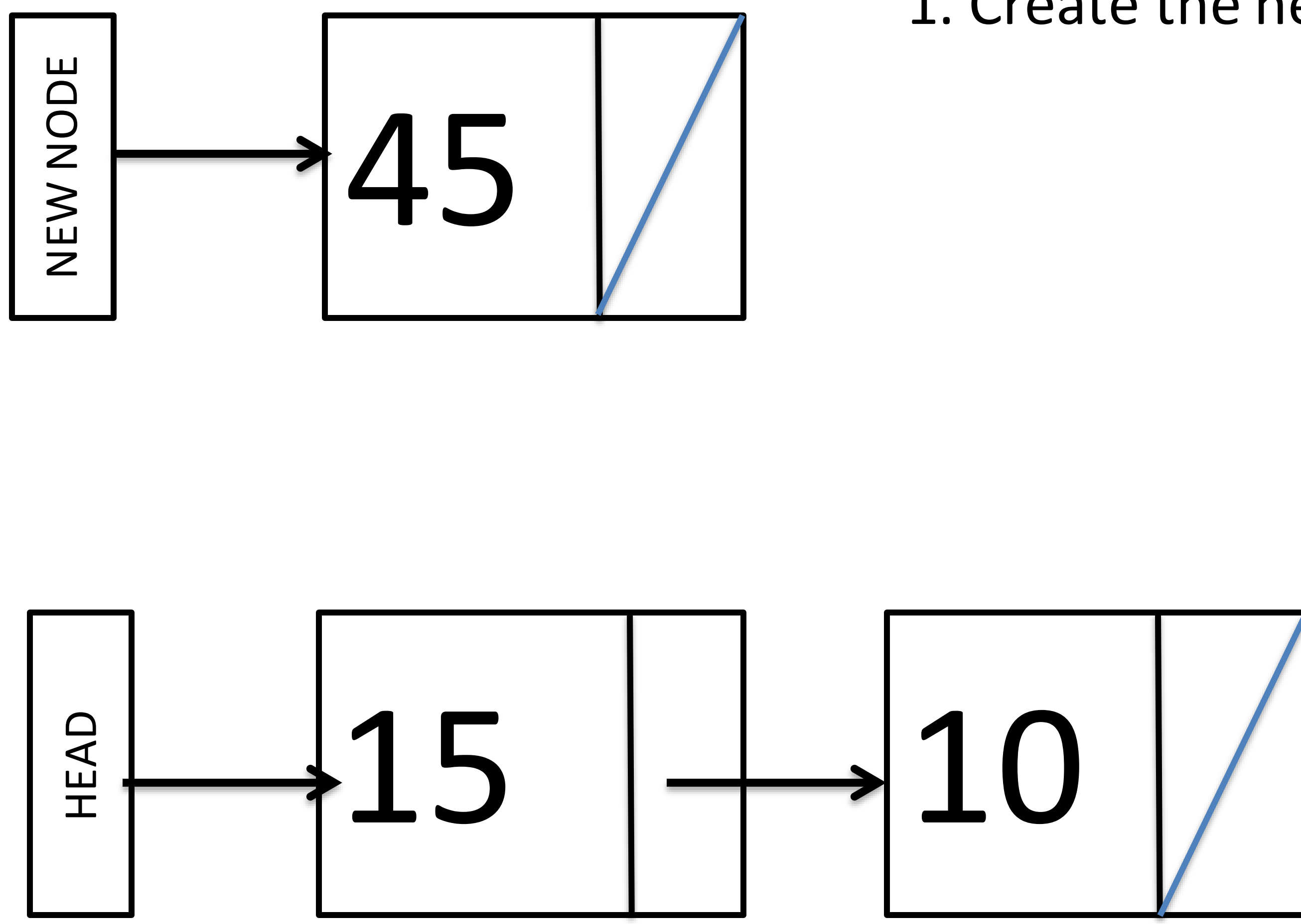
- Inserting a Node
  - At Head
  - At Tail
  - In Middle
- Removing a Node
  - At Head
  - At Tail
  - In Middle

Inserting at Head

- Easiest of all cases

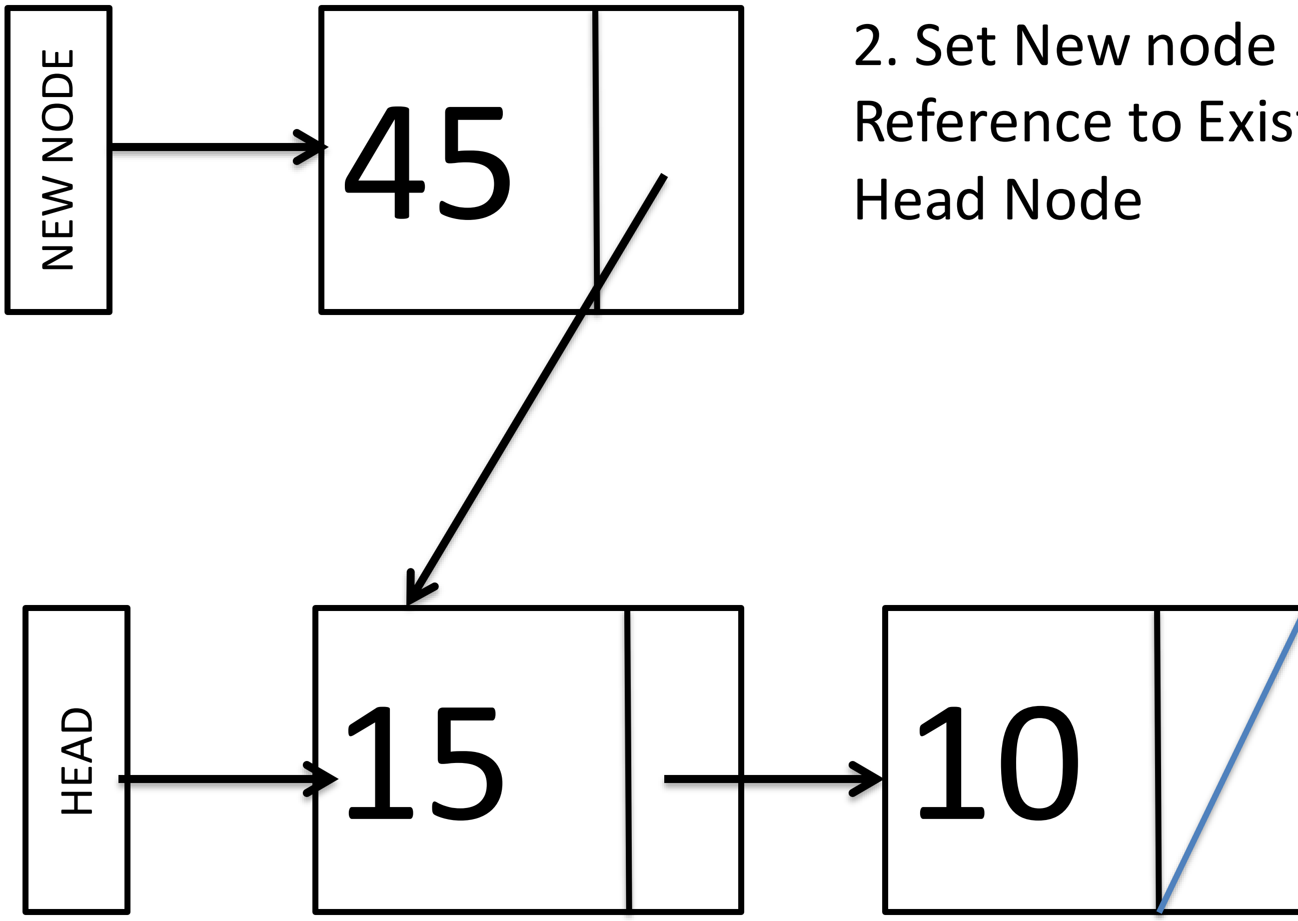
1. Create the New Node
2. Set its reference to be the existing Head Node
3. Set the New Node to be the Head Node

Insert at Head



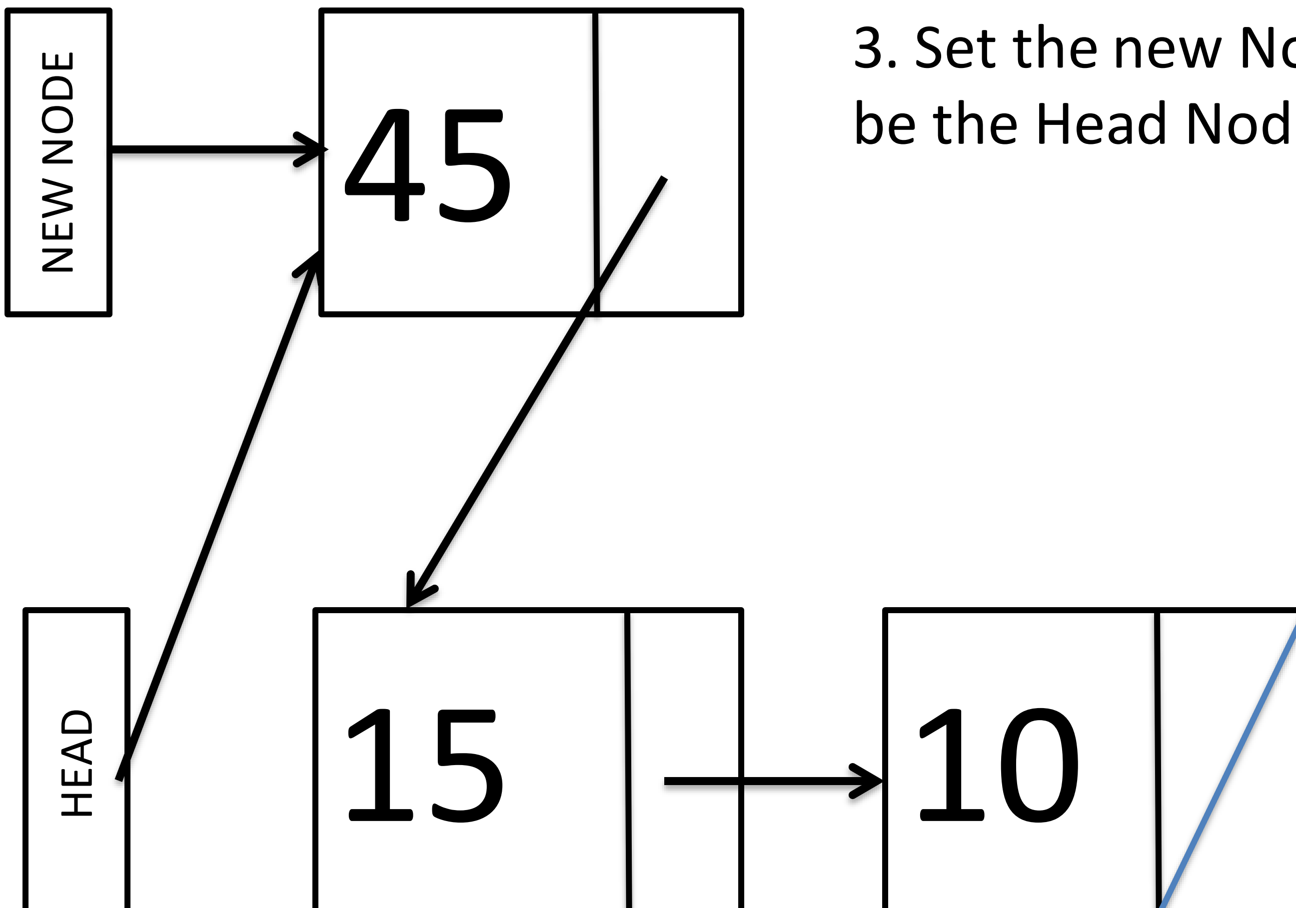
1. Create the new node

Insert at Head



2. Set New node Reference to Existing Head Node

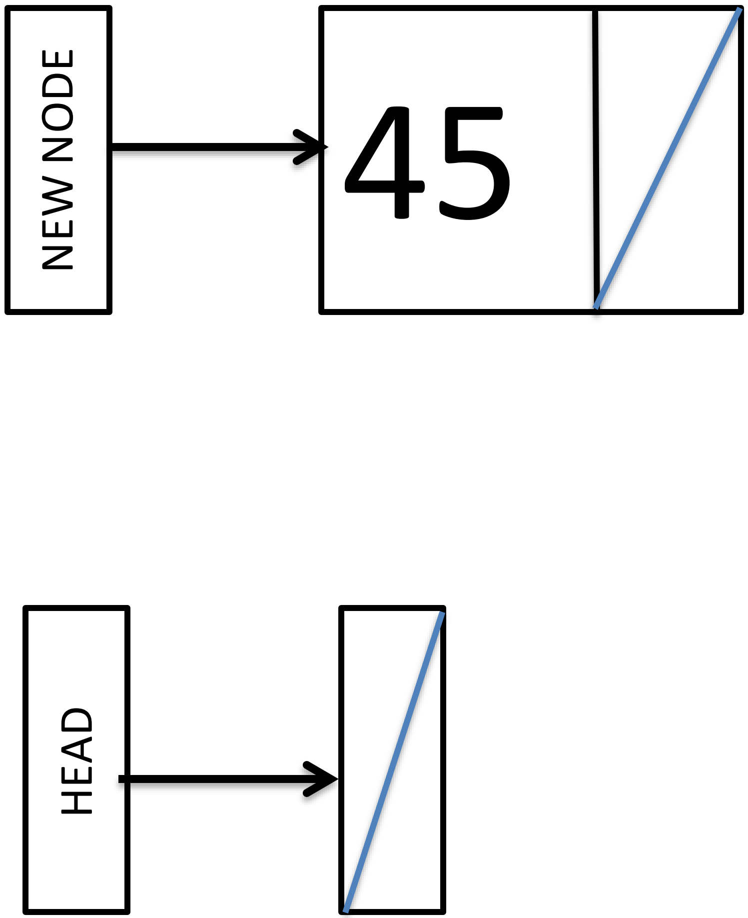
Insert at Head



3. Set the new Node to be the Head Node

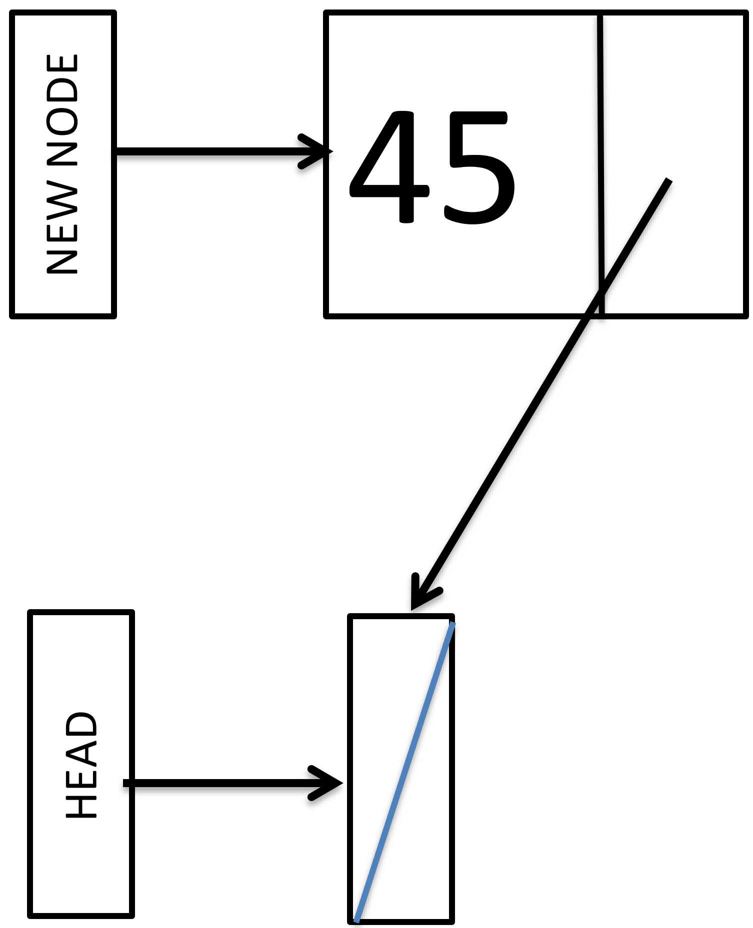
What if the list is Empty?

Insert at Head (Empty List)



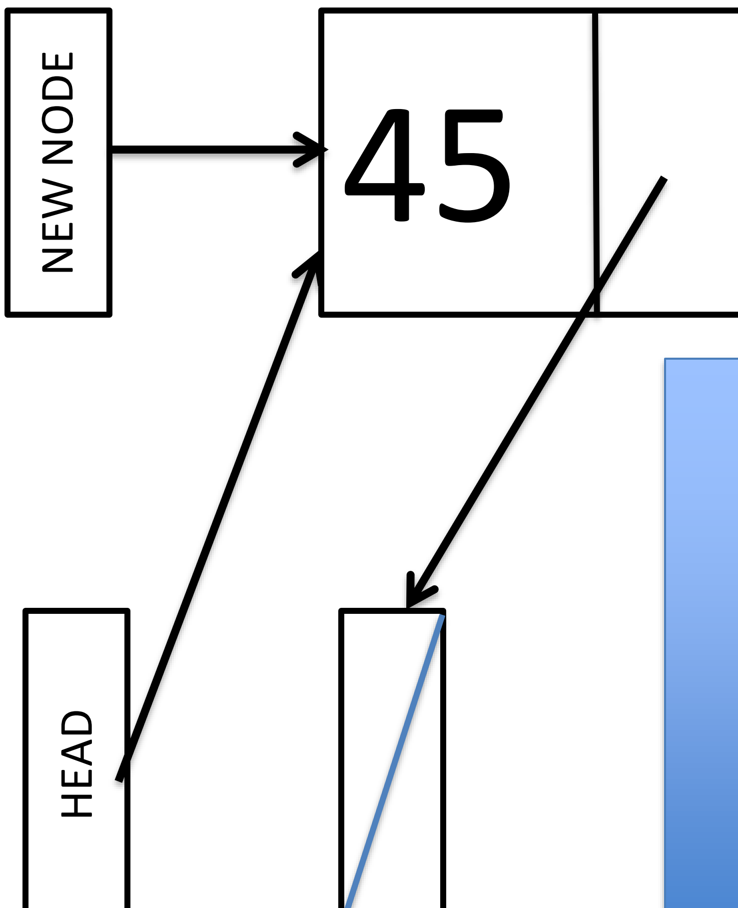
1. Create the new node

Insert at Head



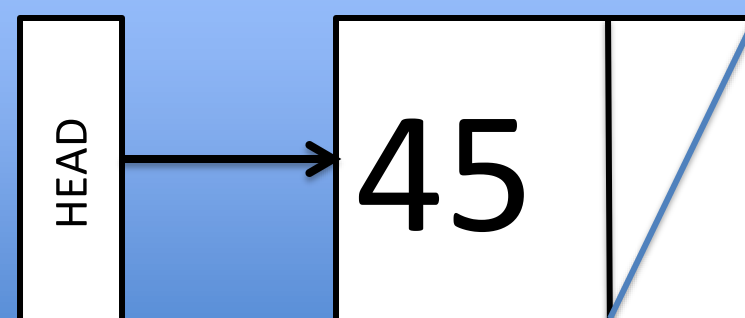
2. Set New node Reference to Existing Head Node

Insert at Head



3. Set the new Node to be the Head Node

Which is the same as:



Complexity

- What is the growth rate of insert at head
  A. O(1)
  B. O(N)

www.socrative.com / `student login' / room: SD32019

Complexity

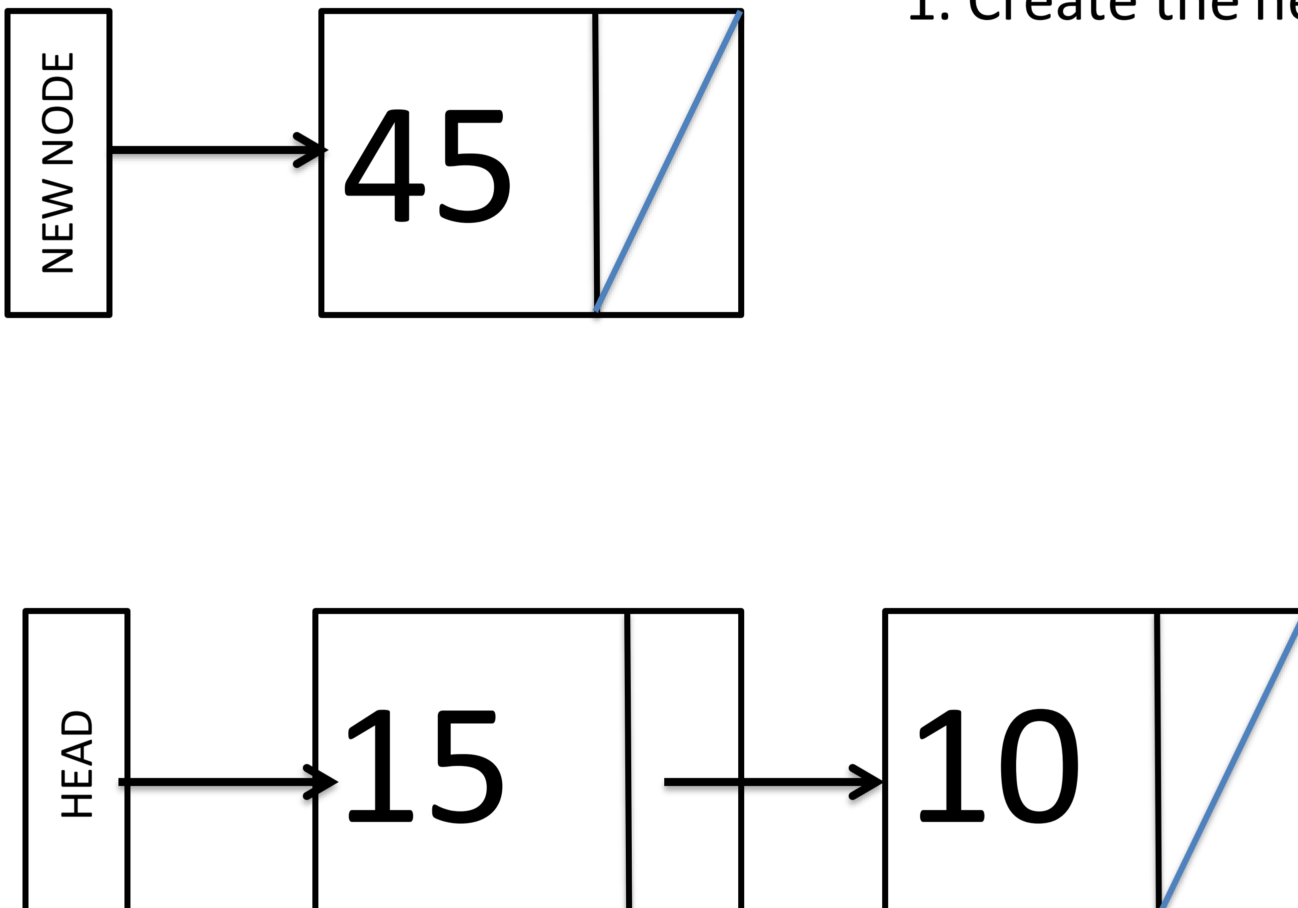
- What is the growth rate of insert at head
  A. O(1)
  B. O(N)

Solution: A – O(1)
No need to iterate the linked list, and the number of operations are the same regardless of the size of the list

Insert at Tail

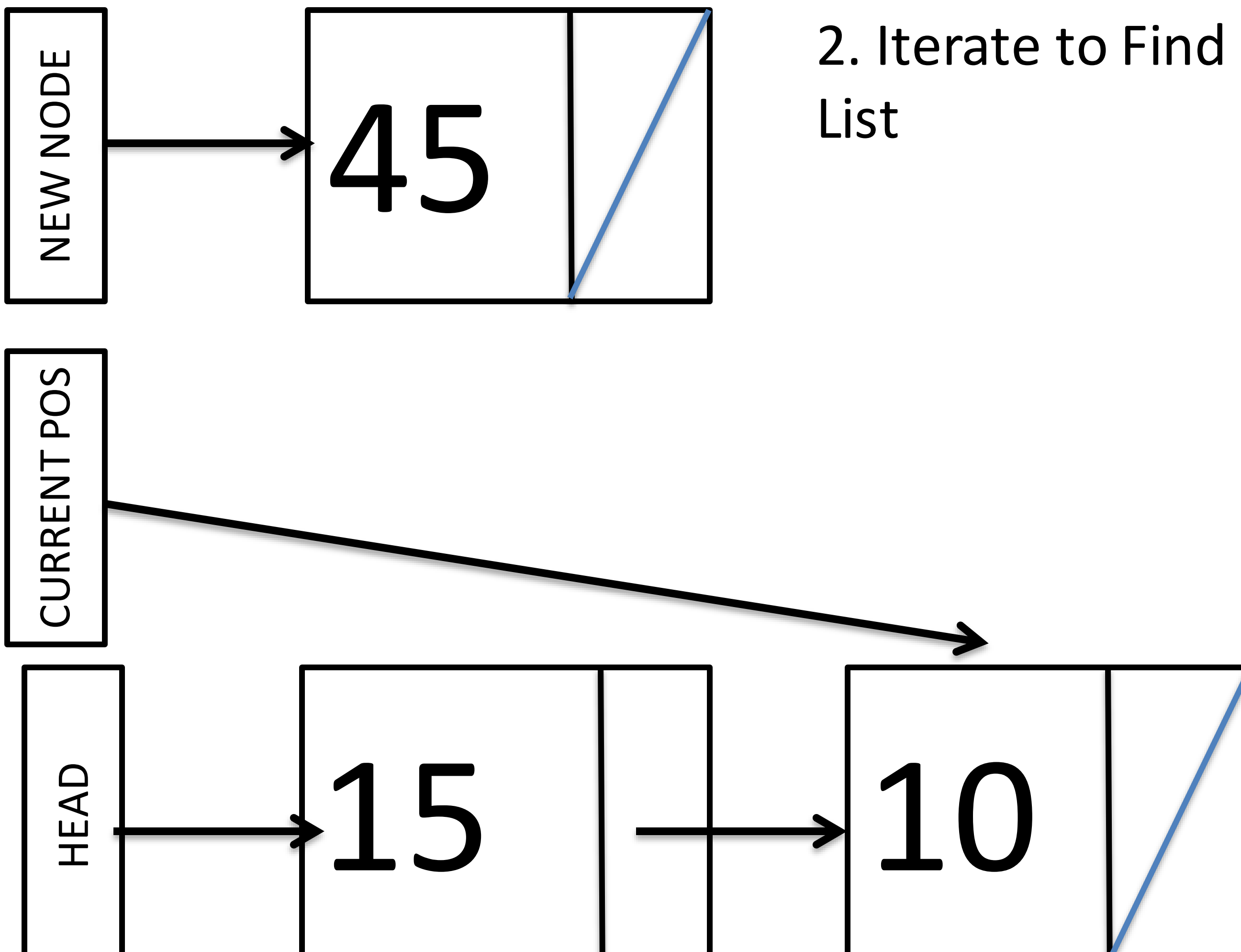
- This means adding the New Node to **the End of the List**
- Need to find the existing end of the list first
  - Remember linked-list are not random access
- So
  - Create the New Node
  - Iterate over the List until we reach the End
  - Set the New Node to be the Last Node’s Reference

Insert at Tail



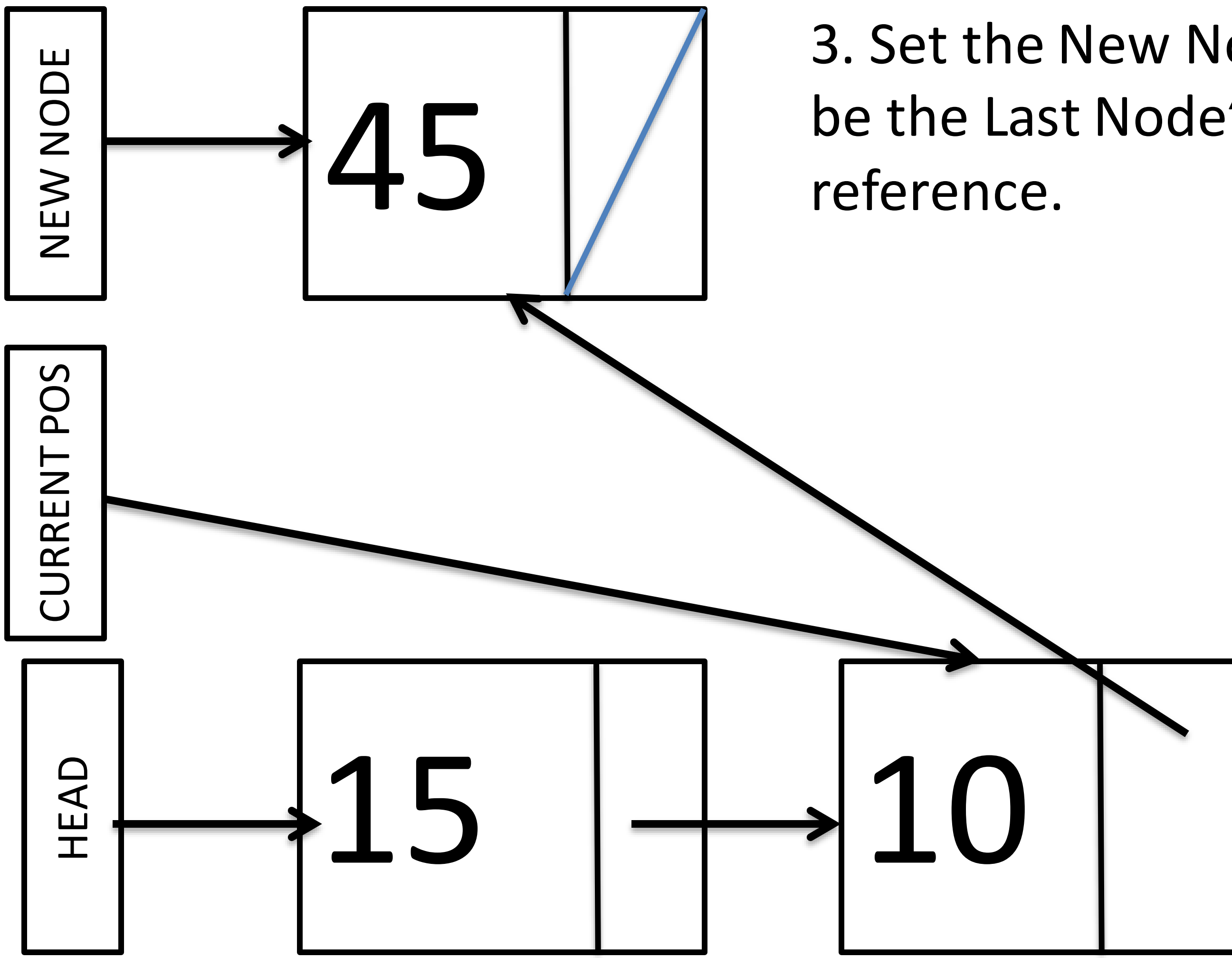
1. Create the new node

Insert at Tail



2. Iterate to Find End of List

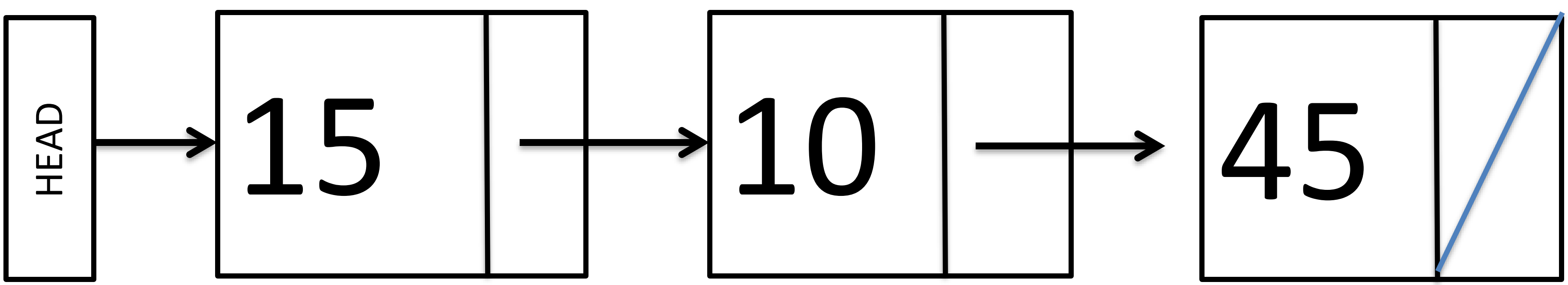
Insert at Tail



3. Set the New Node to be the Last Node's reference.

Insert at Tail

- Which Leaves us with this..
- Note because we add at end there is no final fix-up step



- What if the List is empty ?

Complexity

- What is the growth rate of insert at tail
  A. O(1)
  B. O(N)

Complexity

- What is the growth rate of insert at tail
  A. O(1)
  B. O(N)

Solution: B – O(N)
Need to iterate the end of linked list, which is linear to the size, which constant size operations for each step.

Insert in the middle

- Sometimes want to add in the “middle” of the list
- Need to Determine where in the “Middle” we want the new node
- Not a basic operation on Linked-Lists
  - Add at beginning and Add at end are
  - But this is a more generic replacement for Add at End

Insert in the middle

- We could find an index and insert there

```
insertAt(Node n, int position)
```

- Alternatively, we find the Node that should be immediately prior to the New Node

```
insertAfter(Node n, String s)
```

```
insertMiddle(Node n)
```

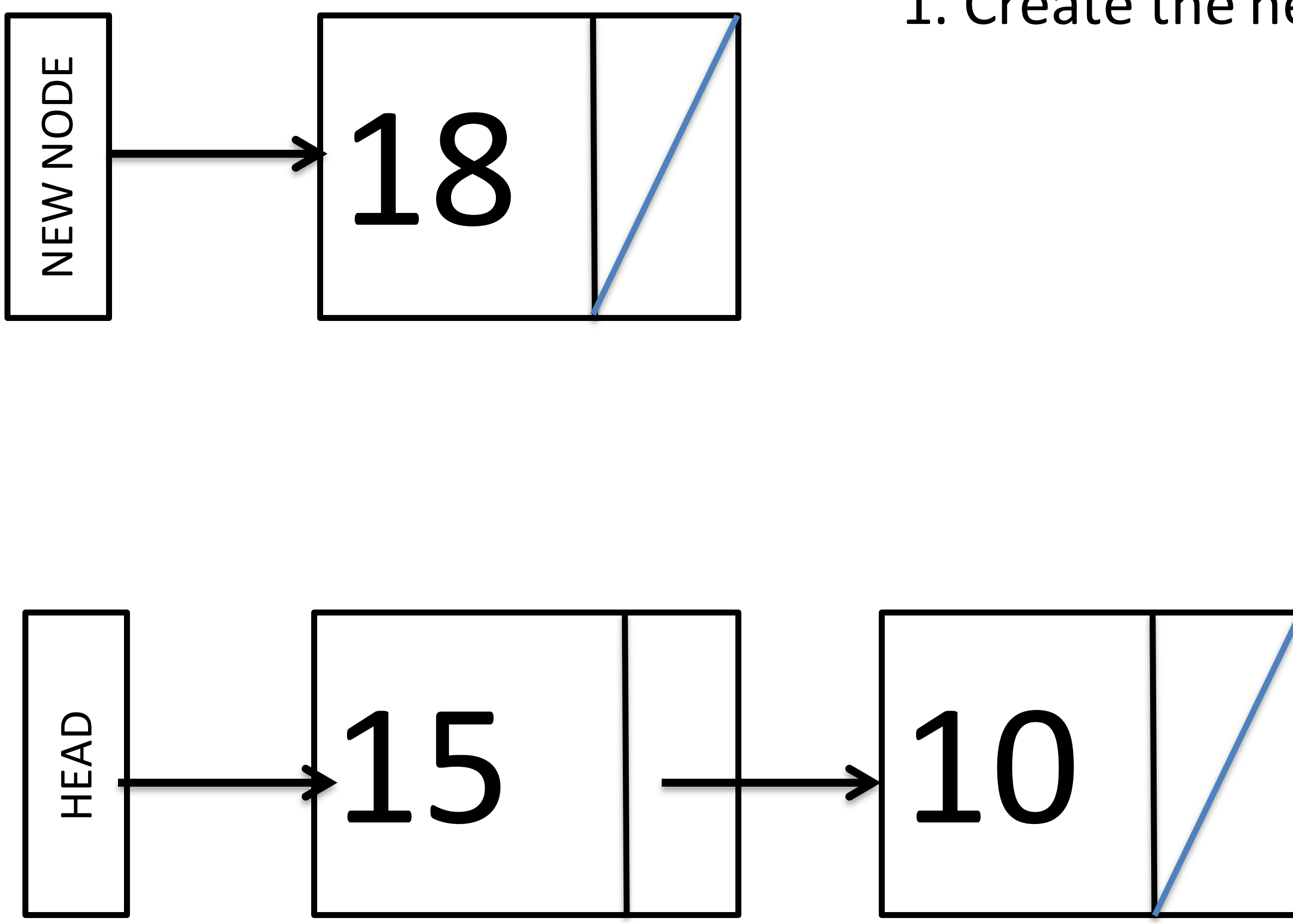
Insert in the middle

- So..
  1. Create the New Node (Nn)
  2. Find the immediately Prior Node (Np)
  3. Set the tail of Np as the tail of Nn
  4. Add Nn as the tail node of Np

- You need to figure out how to get Np depending on the application area

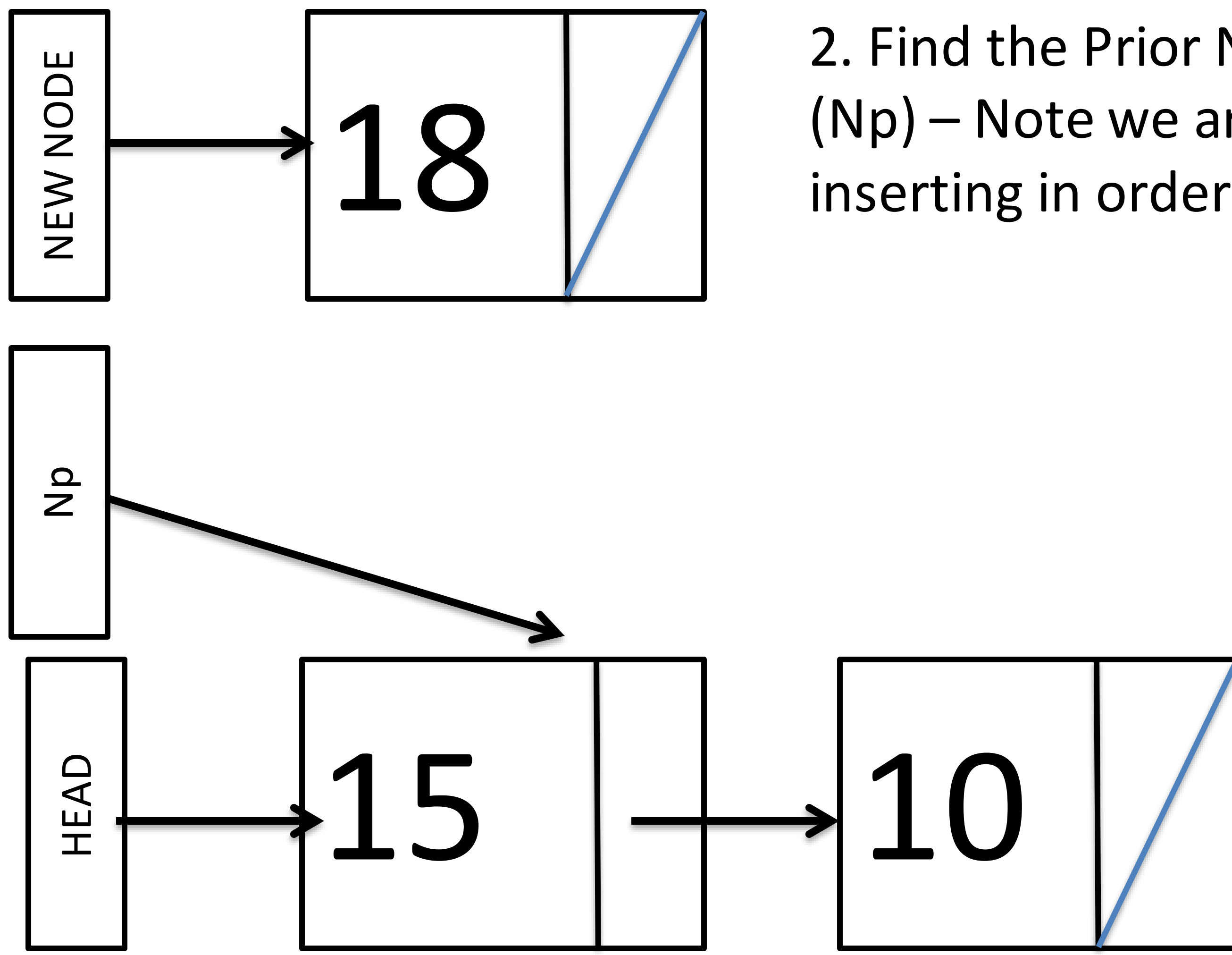
Insert in the middle

1. Create the new node

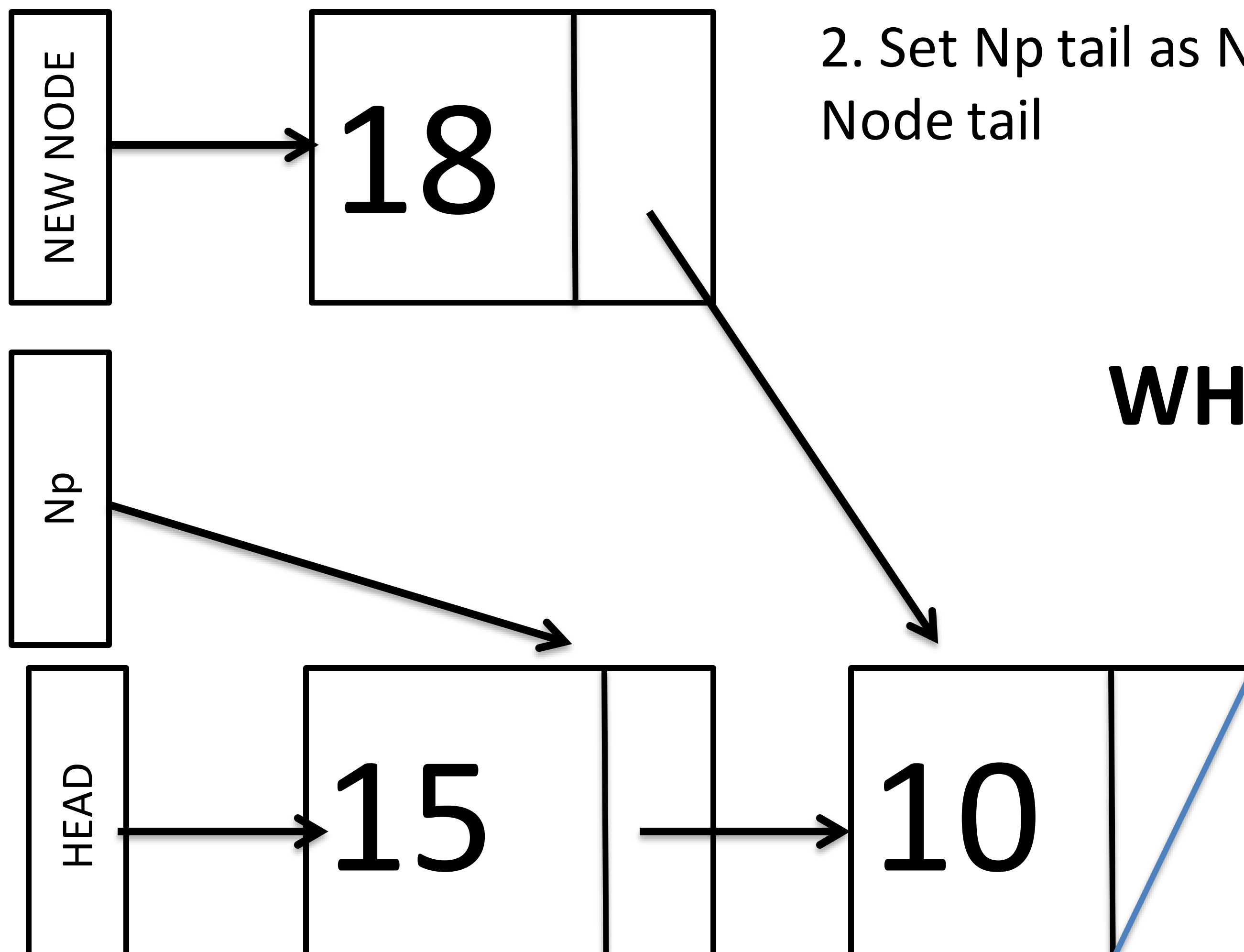


Insert in the middle

2. Find the Prior Node (Np) – Note we are inserting in order here



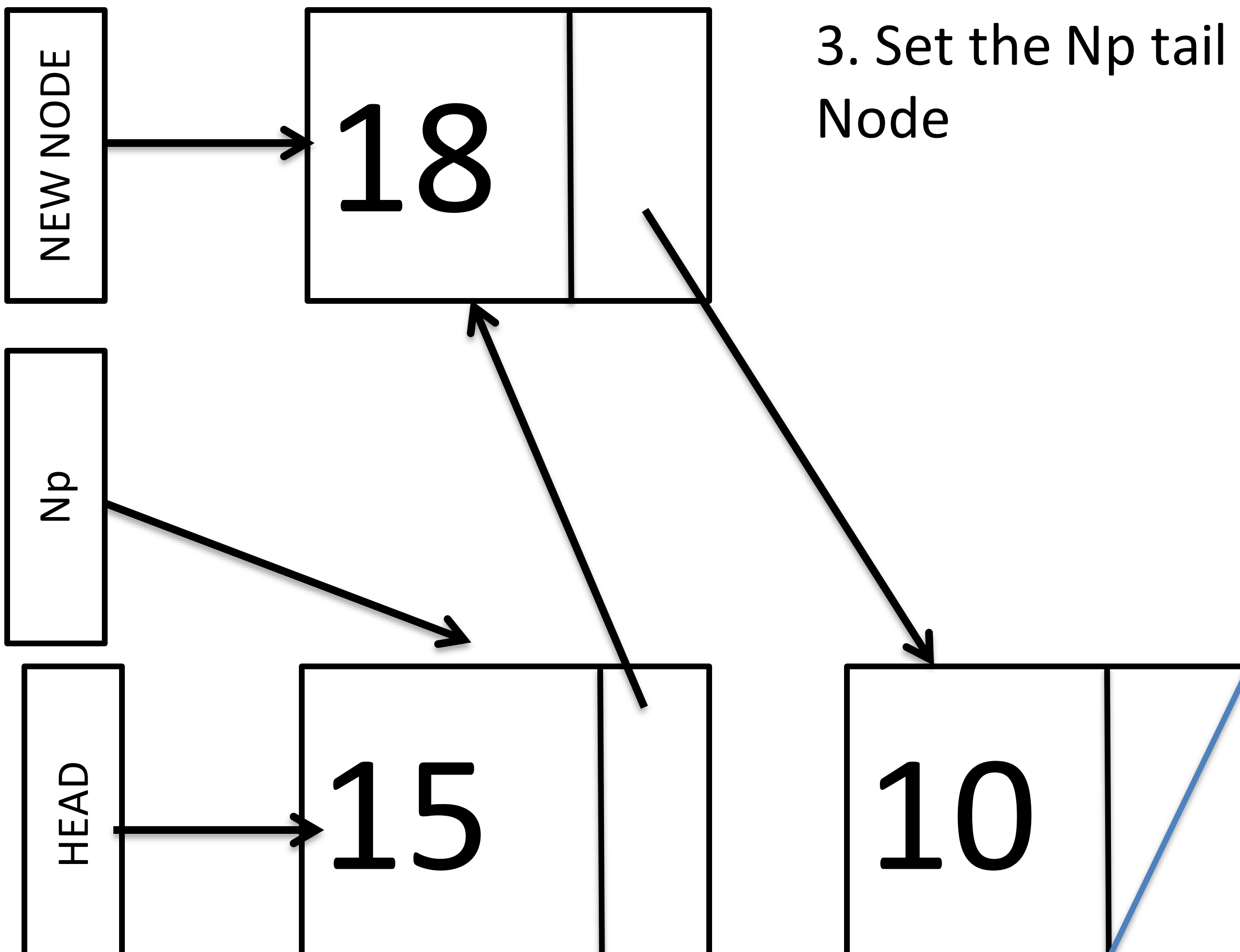
Insert in the middle



2. Set Np tail as New Node tail

WHY?

Insert in the middle



3. Set the Np tail to New Node

Complexity

- What is the growth rate of insert in middle
  A. O(1)
  B. O(N)

Complexity

- What is the growth rate of insert in middle
  A. O(1)
  B. O(N)

Solution: B – O(N)
As we don't know where the middle is it can be the same as end, so same argument as with insert at end

www.socrative.com / `student login' / room: SD32019

Deleting from the List

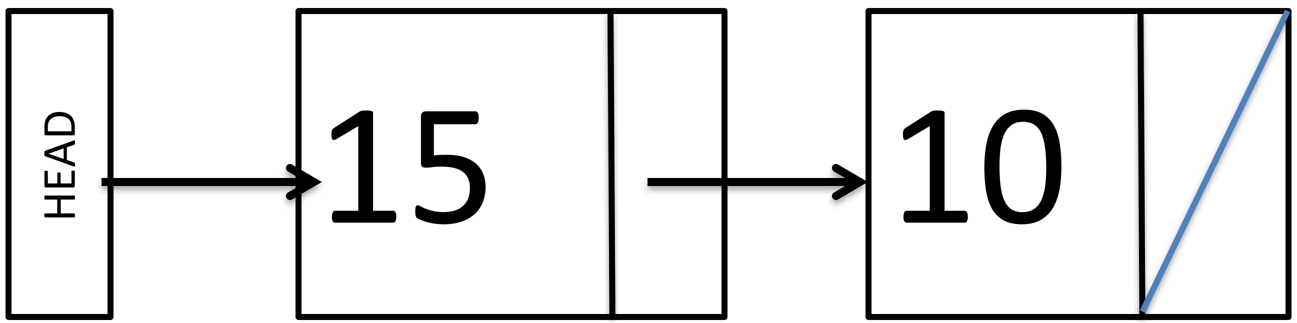
- We may also need to remove them
- Again – 3 Ways
  - Remove at Head
  - Remove at Tail
  - Remove Named Node
    - i.e. determine the node we want removed
    - E.g. The Node with the value **50**
  - Only **Remove at Head** is easy

Remove at Head

- Even easier than adding a node
  - Check that list is not empty
    - Can’t remove something if it isn’t there
    - Next step will crash the program if the list is null
  - Make the List’s Head Node the Head’s Tail Node

Delete from Head

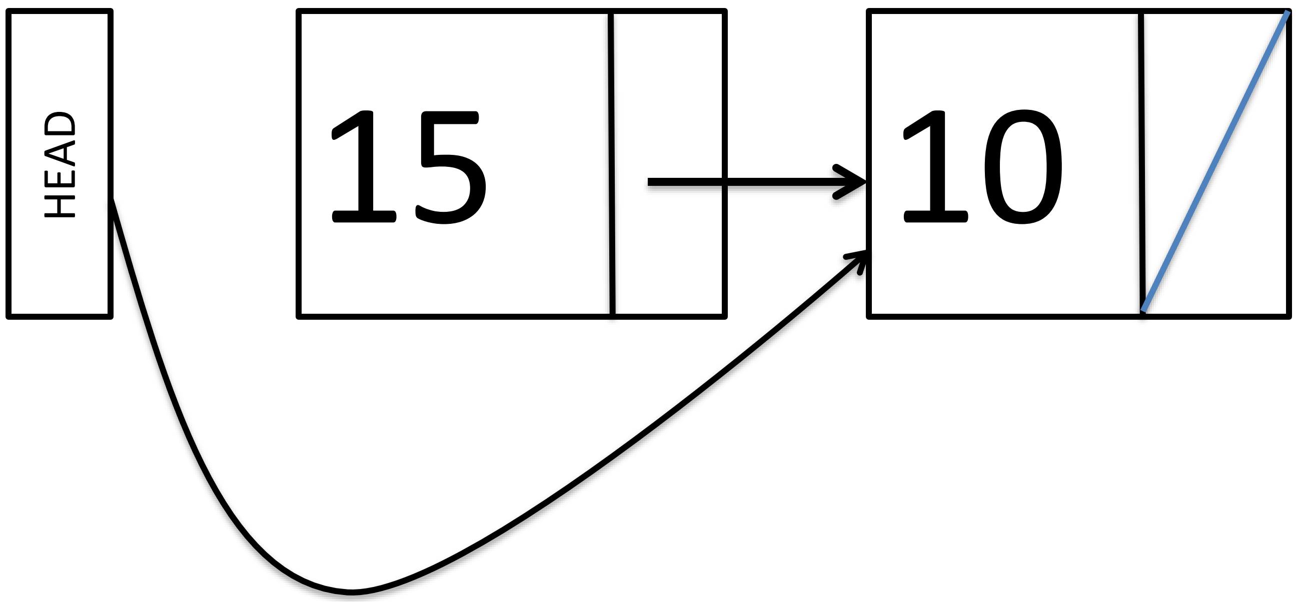
1. Check the List isn't Null



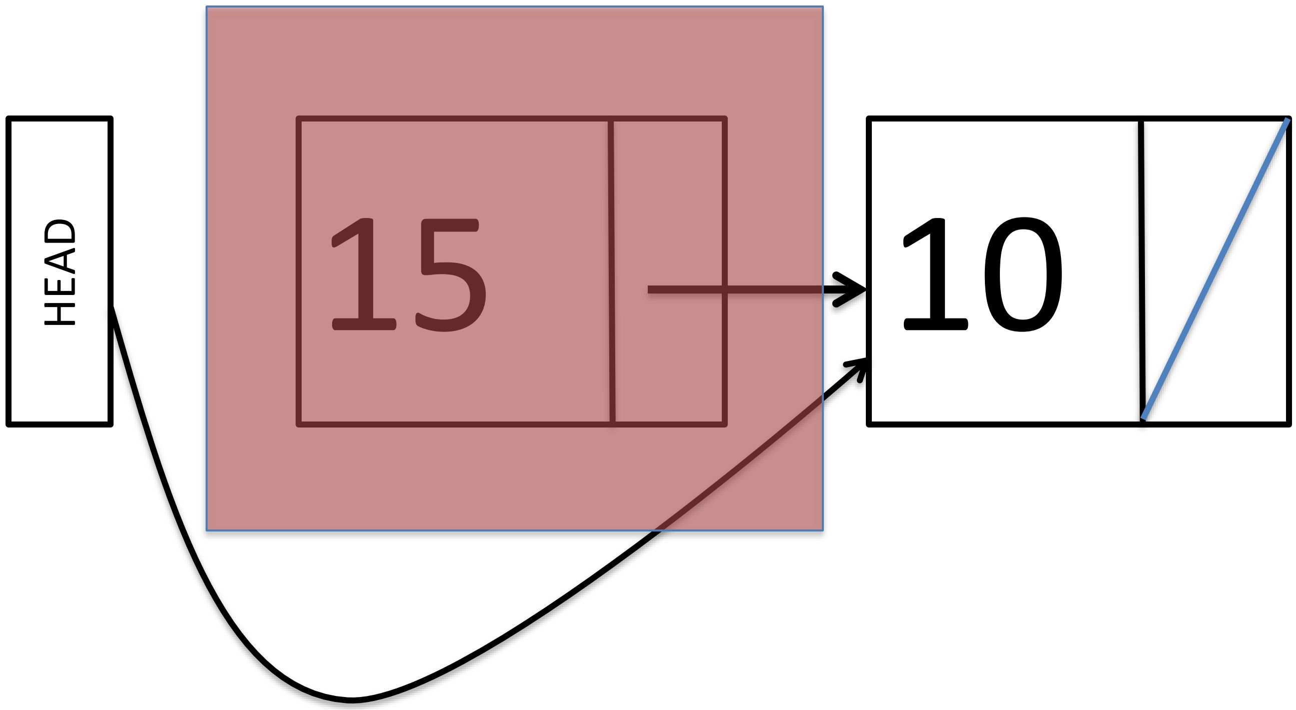
HEAD == NULL returns FALSE

Delete From Head

2. Set the Head to the First Node's Tail

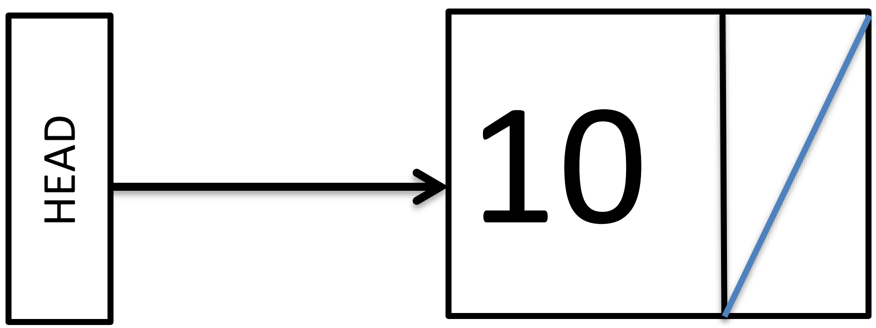


Delete From Head



What Happens to this Node?

Delete From Head



Complexity

- What is the growth rate of deleting from head?
  A. O(1)
  B. O(N)

www.socrative.com / `student login’ / room: SD32018

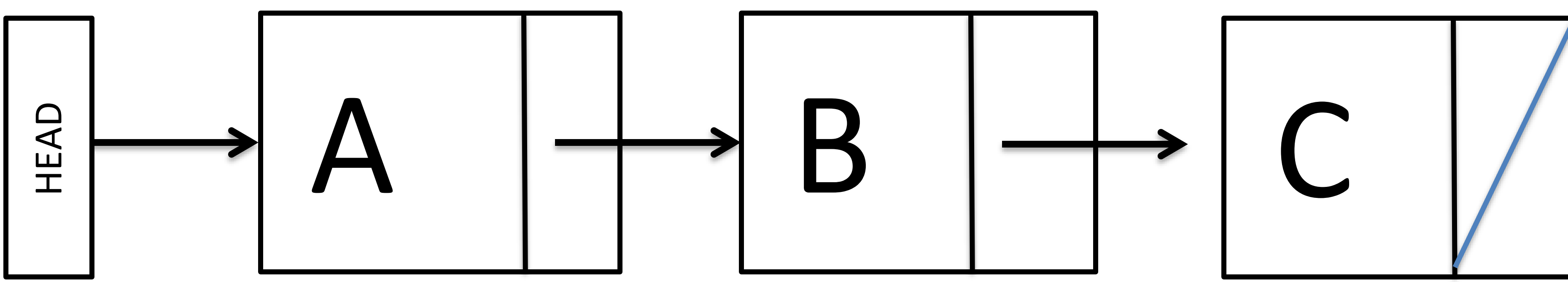
Complexity

- What is the growth rate of deleting from head?
  A. O(1)
  B. O(N)

Solution: A – O(1)
Similar to insert at head

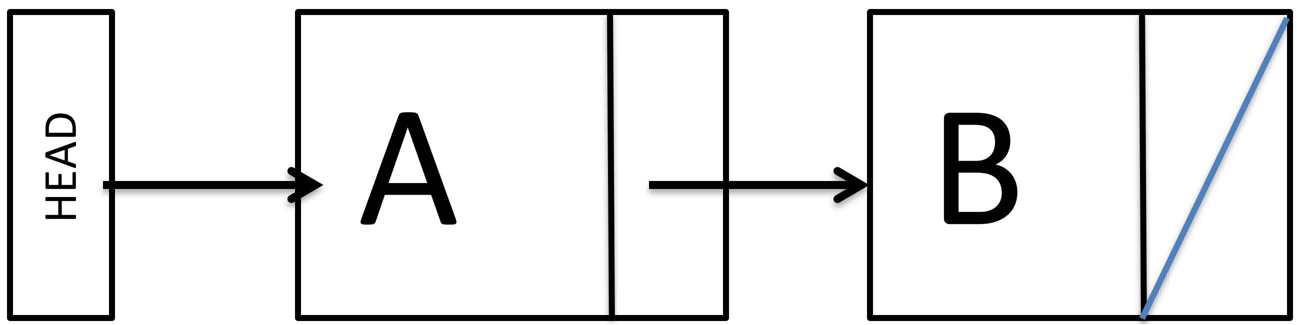
Exercise

- Describe how to
  - Remove B
  - Remove C
- Remember you can only access the linked list via HEAD and can only get to the next element



Middle and At End Deletion

- Removal at the End or middle is hard
  - We cannot go back.
  - If we have reference to B we cannot get A from that



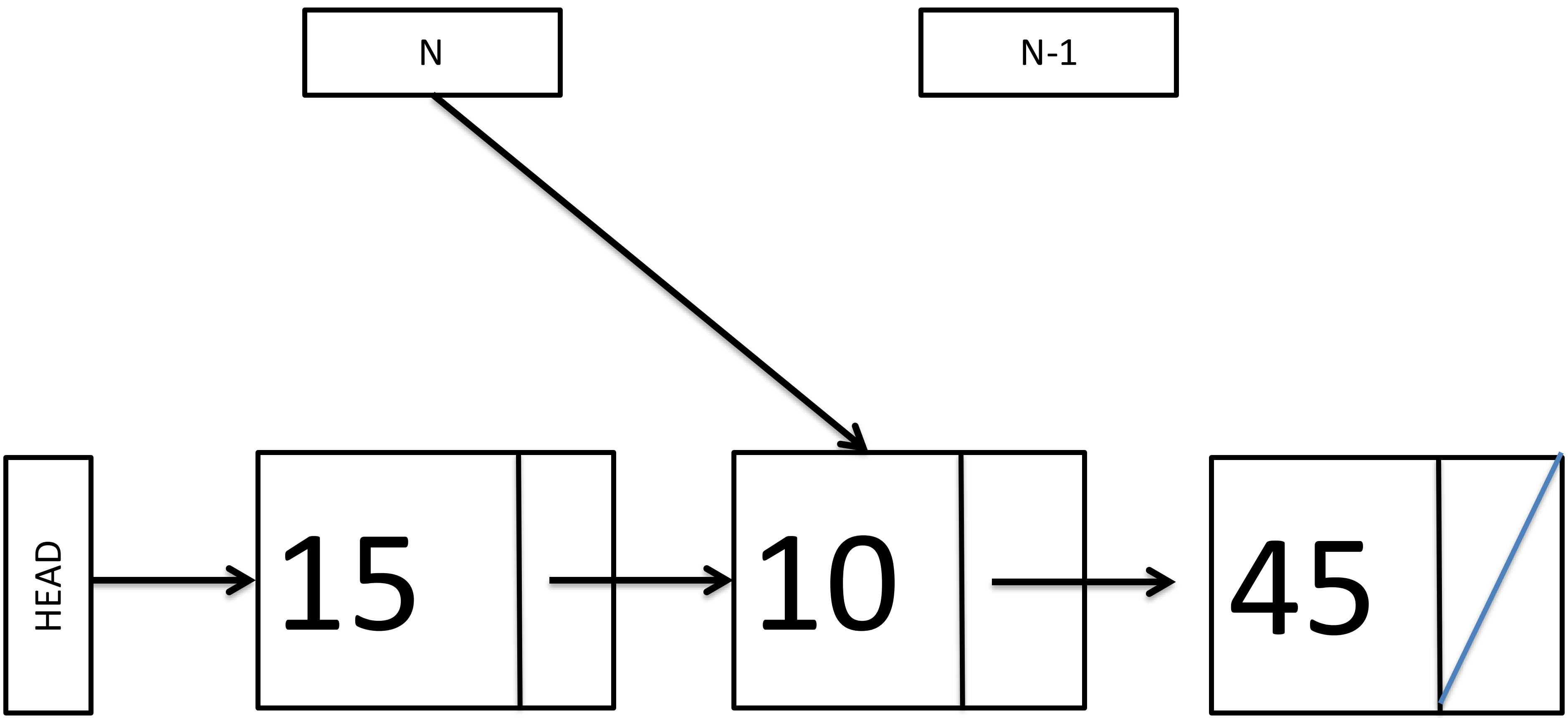
- To Delete B from the list, you need to do it from A
  - You need to access the Node **before** the one you want to delete

Middle and At End Deletions

- So the steps to delete node N are
  1. Find the node (N) to delete
  2. Find the node **prior** (N-1) to the one you want to delete (N)
     - If you want the last Node removed you want the node prior to the Node that has **null** as a Reference
  3. Set N-1’s nextNode reference to N’s nextNode reference
     - Effectively cut it out of the list

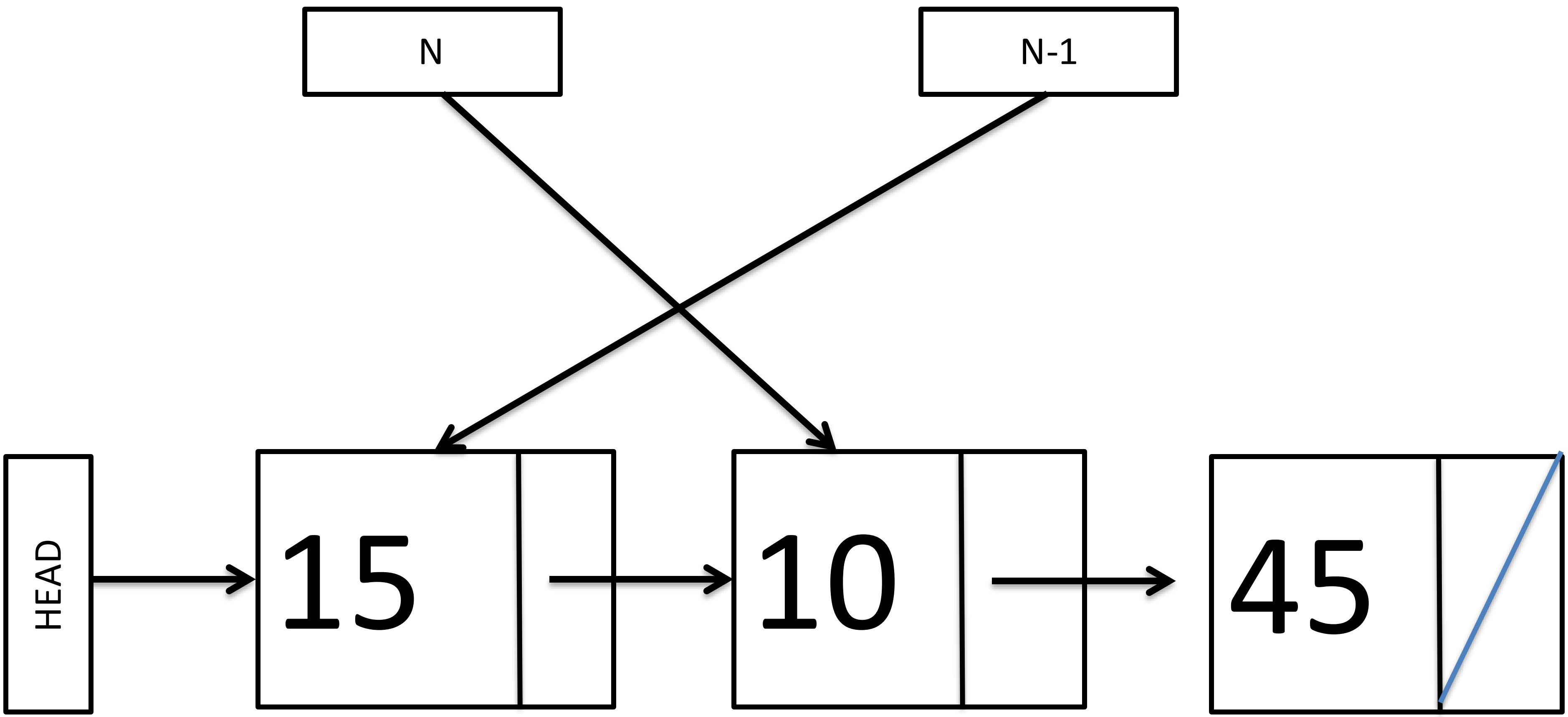
Delete Node 10

1. Find Node N



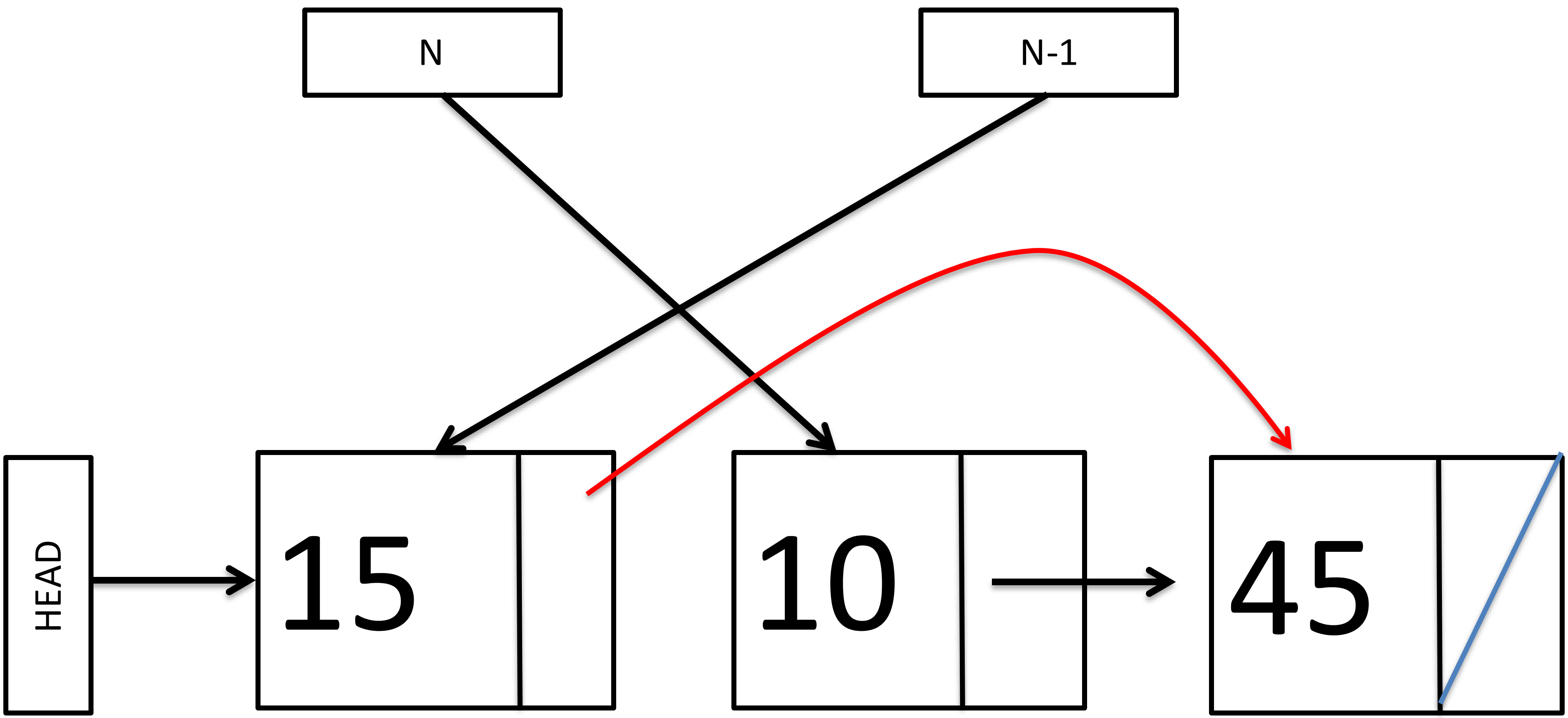
Delete Node 10

2. Find Node Prior to N (N-1)



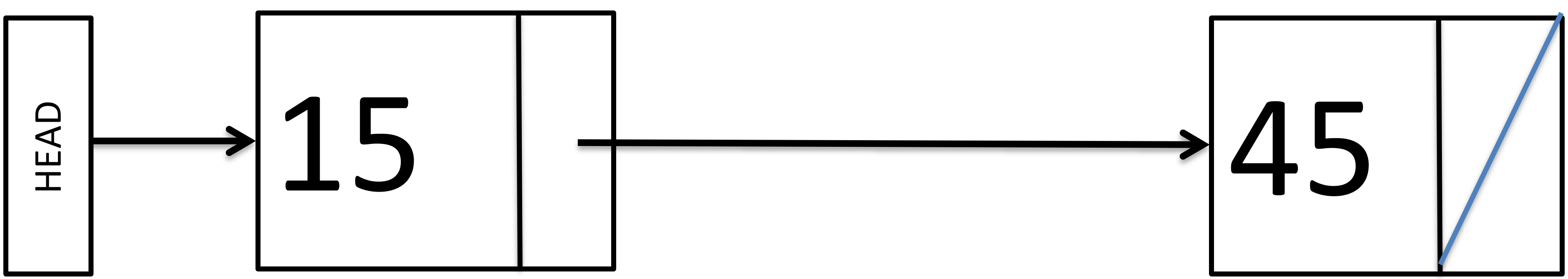
Delete Node 10

3. Make Node N-1 point to N+1



Delete Node 10

Variables N and N-1 are temp so after garbage collection



Complexity

- What is the growth rate of deleting from middle/end?
    - A. O(1)
    - B. O(N)

www.socrative.com / `student login' / room: SD32019

Complexity

- What is the growth rate of deleting from middle/end?
  A. O(1)
  B. O(N)

Solution: B – O(N)
Similar argument to insert at middle/end

Implementing Linked Lists

- Best Implemented as an ADT
  - Usually have Node class (internal)
  - A Linked List class (public facing)
  - Functionality is split between both
- Functionality can be iterative or recursive
  - We will look at both
- You will be implementing some additional functionality in the Lab

Iterative Linked Lists

```
public class LinkedList{

// The Node is represented as a private inner
//  class
// This helps keep the implementation cleaner.
// Especially when we start using recursion
private class Node{
	private Object value;
	private Node nextNode;

	public Node (Object v){
		value = v;
		//good practice to be explicit
		// about making this null;
		nextNode = null;
	}

	public Object getValue(){
		return value;
	}

	//Returns the Node that this Node
	//  links to - this may return null
	public Node getNextNode(){
		return nextNode;
	}

	//Sets the NextNode to the given Node
	public void setNextNode(Node n){
		nextNode = n;
	}
}

// Holds a reference to the head of the list
private Node headNode;

public LinkedList(){
	headNode = null;
}
```

Iterative Linked Lists

```
//Adds the given Object at the Head of the list
public void addAtHead(Object o){
        Node newNode = new Node(o); //create the node
        newNode.setNextNode(headNode); //add the headNode to the tail of the newNode
                                                          //note how it doesn't matter if the list (headNode) is null
        headNode = newNode; // set the New Node to be the new head Node
}

//adds the given Object to the Tail of the list
public void addAtTail(Object o){
        Node newNode = new Node(o);
        // we need to check if the list is null
        if(headNode == null){
              // if it is, then we just add the newNode as the headNode;
              headNode = newNode;
        }else{
               //we need to find the current last node in the list
               Node tempNode = headNode;
               while (tempNode.getNextNode() != null){
                      tempNode = tempNode.getNextNode();
               }
               tempNode.setNextNode(newNode);
        }
}
```

```
//removes and returns the object at the head of the list
//note, this may return null if the list is empty
public Object removeAtHead(){
        if(headNode == null){
                return null;
        }else{
                Node returnedNode = headNode;
                headNode = headNode.getNextNode();
                return returnedNode.getValue();
        }
}
```

```
public Object removeAtTail(){
  if(headNode == null){
        return null;
  }else{
        Node prevNode = null; //N-1
        Node currentNode = headNode; //N

        while (currentNode.getNextNode()!= null){
                prevNode = currentNode;
                currentNode = currentNode.getNextNode();
        }

        if(prevNode == null){
         // Need to be careful here.
         // If there is only 1 element of the list
                headNode = null;
        }else{
                prevNode.setNextNode(null);
        }
        return currentNode.getValue();
    }
}
```

Recursive Linked Lists

- Node class data the same
  - but we add some operations
- The Linked list is really all in the Node class
  - so that is where the recursion goes
- This is not something we want to leave open to the rest of the program
- So the LinkedList Class and Node class work together for best results

```
//adds Node n to the tail of the list
public void addNodeAtTail(Node n){
        if(this.nextNode == null){
                this.nextNode = n;
        }else{
                this.nextNode.addNodeAtTail(n);
        }
}
```

Summary

- We have covered
  - Adding and removing Nodes from a Linked List
  - Diagrammatically and programmatically
- Attendance sheet
- Next lecture: **linked lists** to implement a **stack**
  - These stacks aren't restricted to fixed capacity