

CS350: Data Structures

Linked Lists

James Moscola

Department of Engineering & Computer Science

York College of Pennsylvania



Linked Lists

- **Come in a variety of different forms**
 - singly linked lists
 - doubly linked lists
 - circular linked lists
- **Composed of a set of nodes that hold data and contain one or more pointers to neighboring nodes in the list**
 - singly linked lists contain only a pointer to the next node in the list
 - doubly linked lists contain a pointer to the next node and the previous node in the list

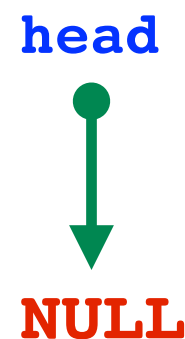


Linked List Operations

- **Basic operations include:**
 - **insert / add**
 - **remove**
- **Additional operations may include:**
 - **getFirst**
 - **getLast**
 - **find**
 - **isEmpty**
 - **makeEmpty**

Linked List Insertion

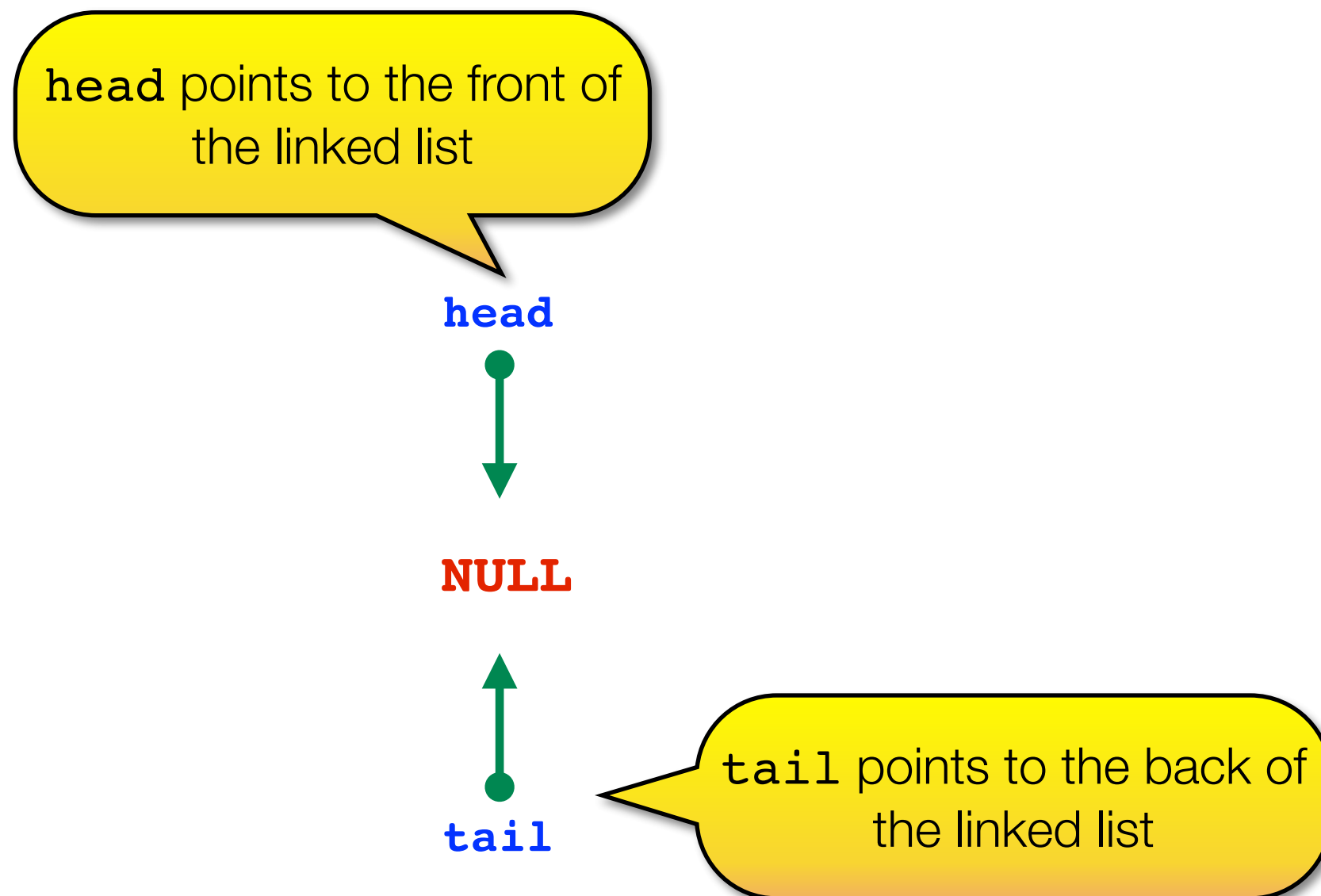
- **Basic implementation uses a **head** pointer that points to the first node in the list**
 - Points to **NULL** upon initialization when no nodes exist in the list



- **Depending on implementation, insertion may take place at the head of the list, at the tail of the list, or at some other specified node**

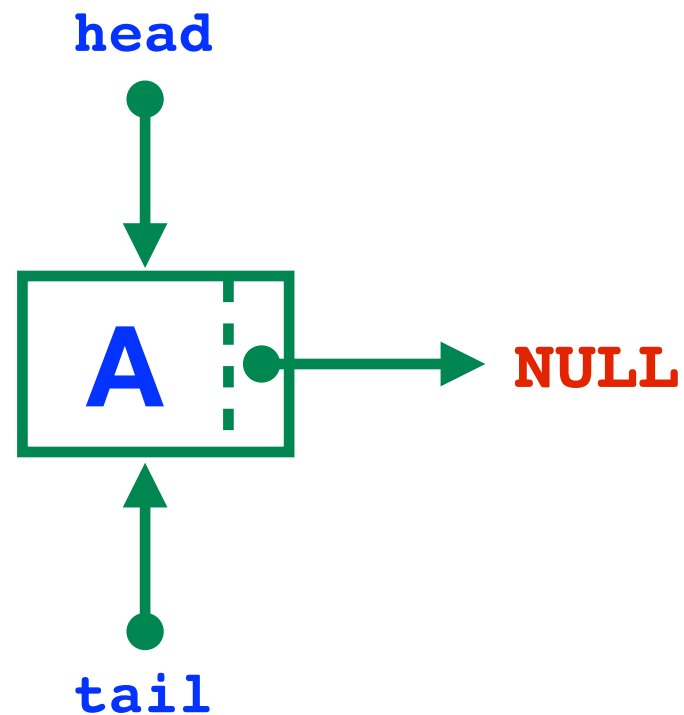
Linked List Insertion

Start with Empty List



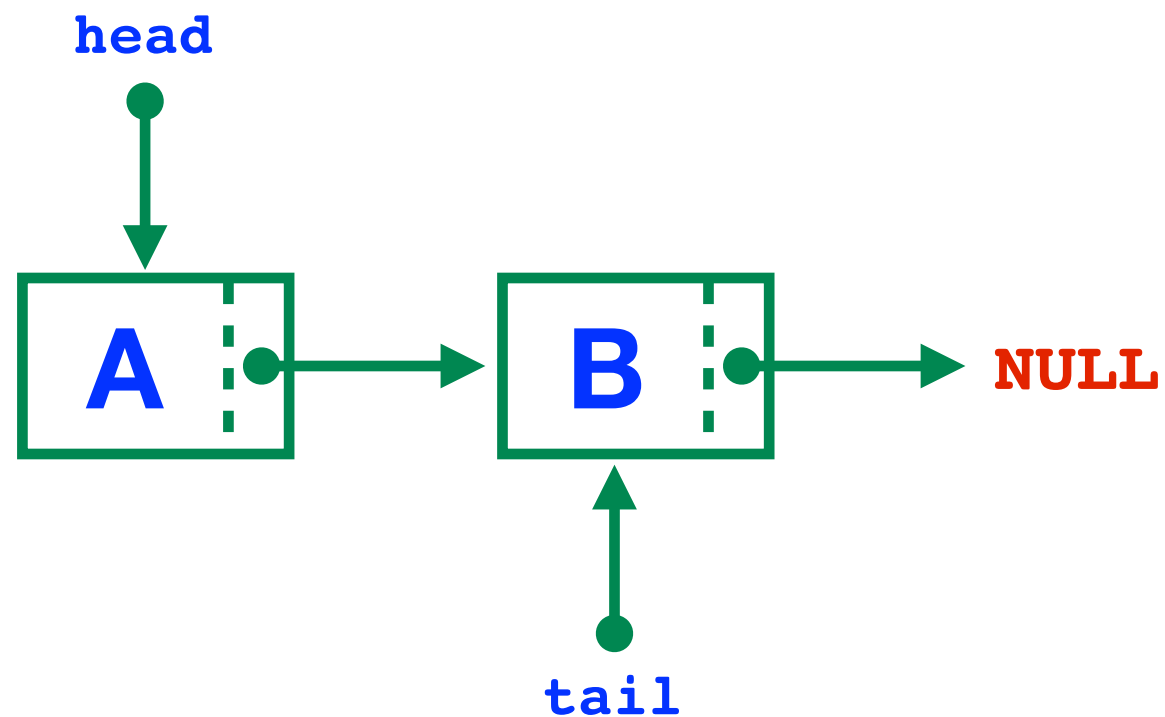
Linked List Insertion

Insert Value: A



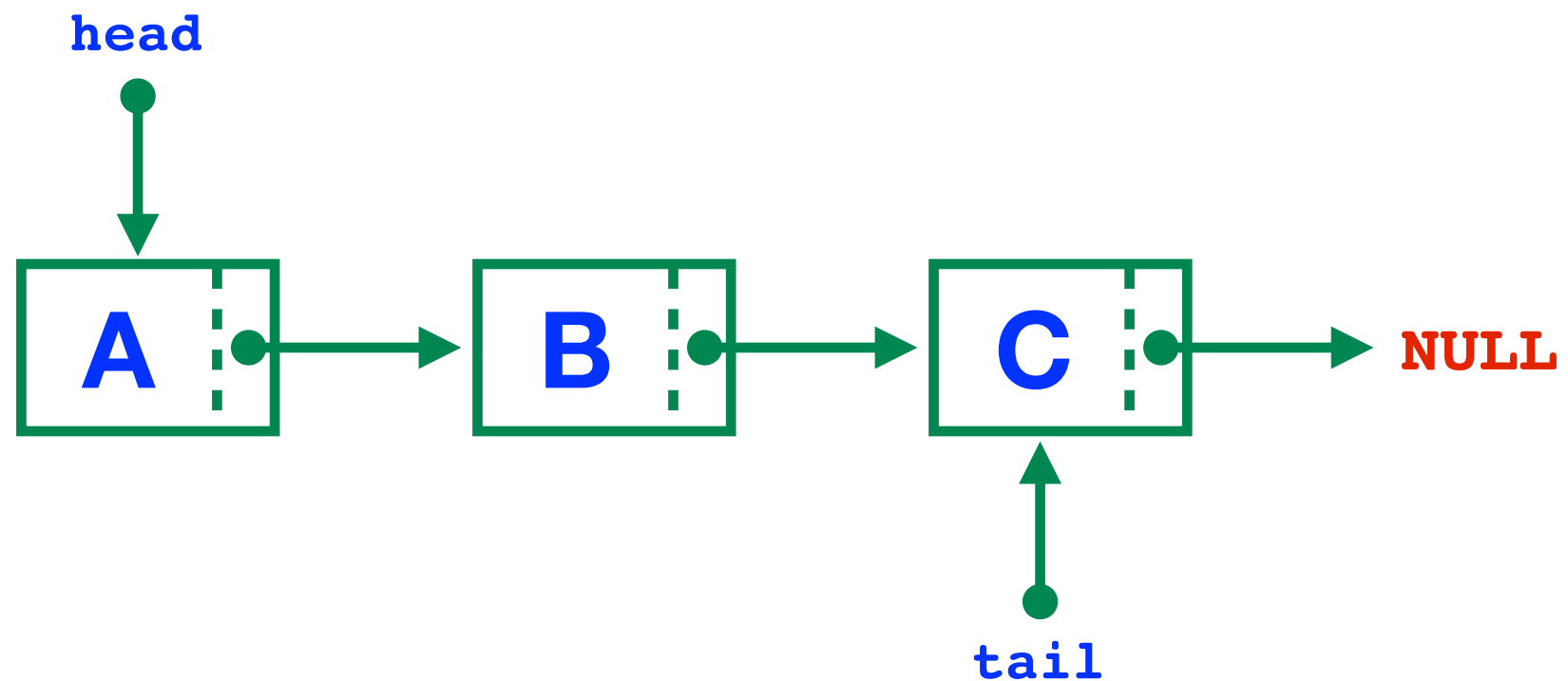
Linked List Insertion

Insert Value: B



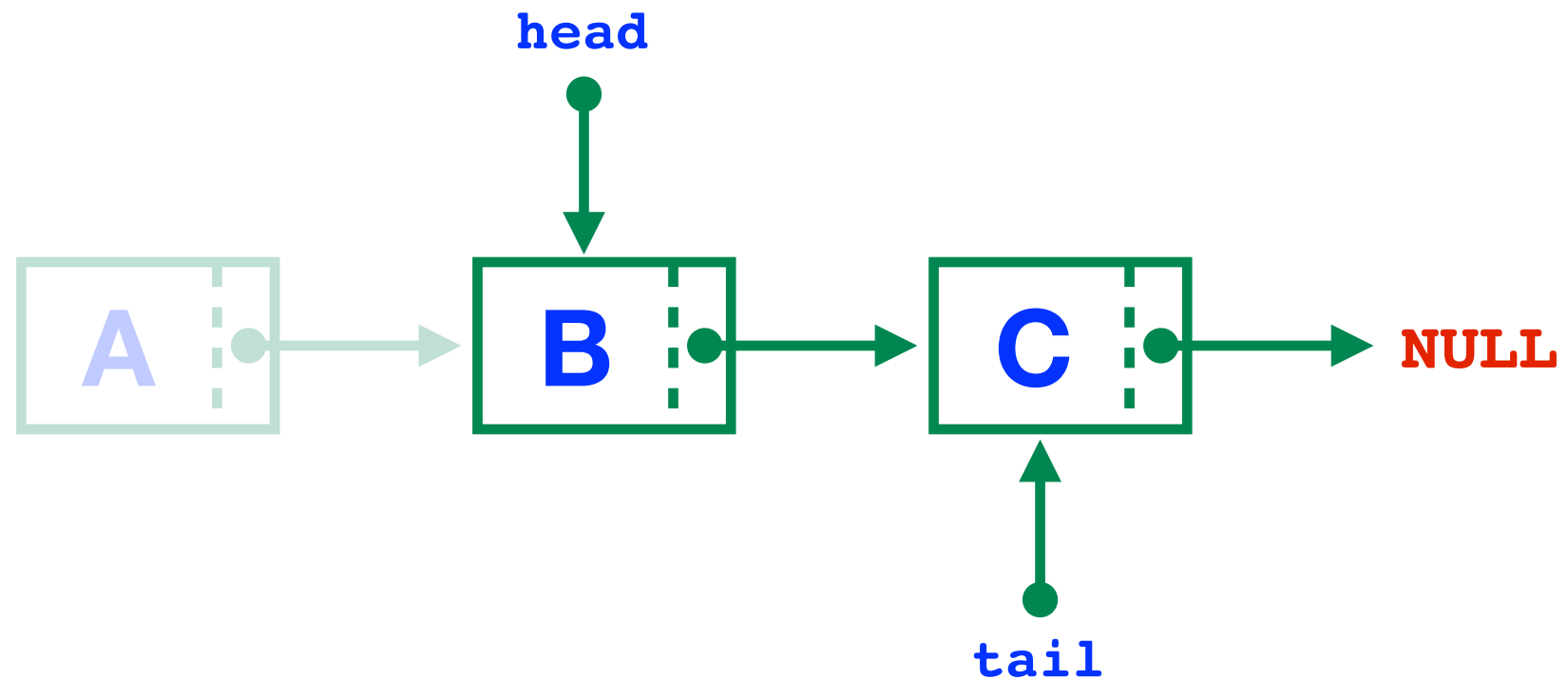
Linked List Insertion

Insert Value: C



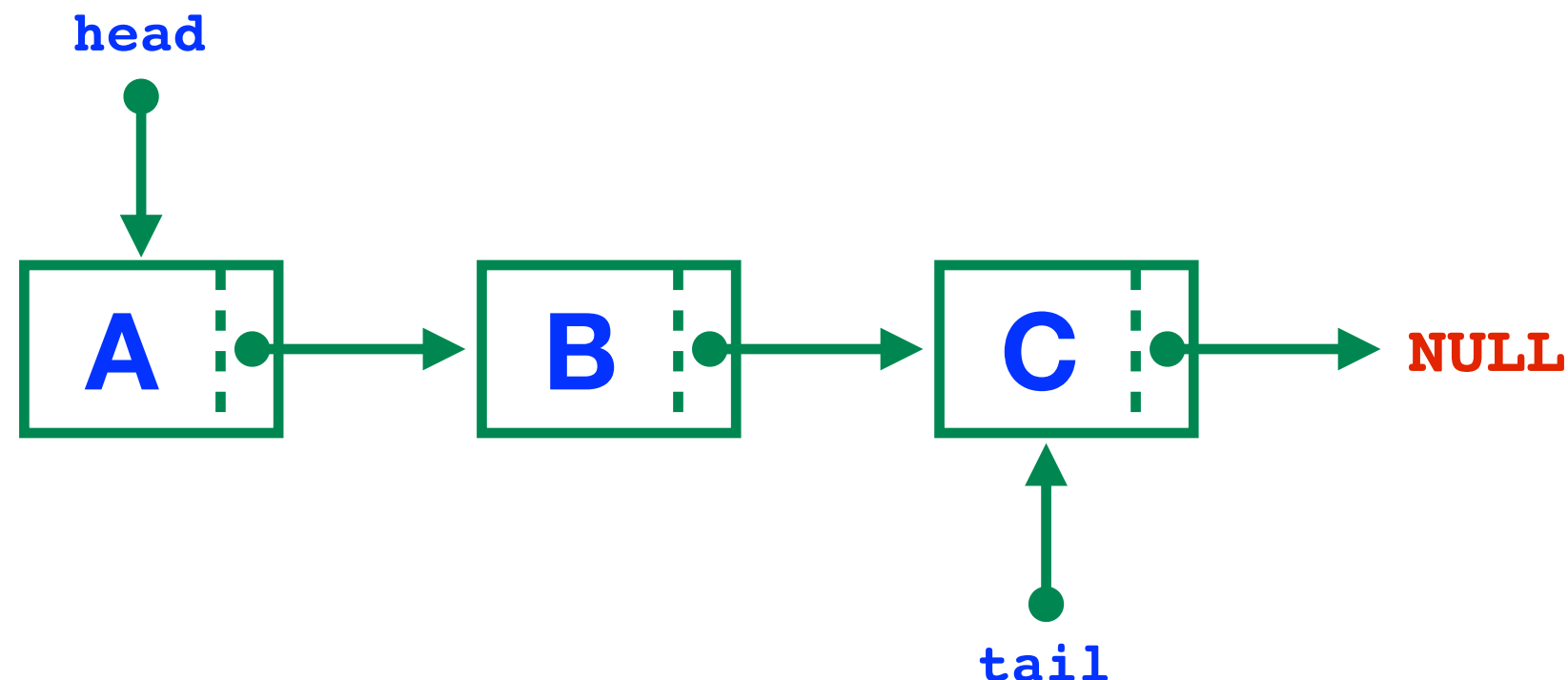
Linked List Removal

Remove First Value:



Linked List Removal

- **Removing last value is not very efficient when using singly linked lists**
 - Want to make next-to-last node in list the last node
 - Must traverse entire list, starting from the **head**, to find the next-to-last node in the list
 - $O(N)$



Linked List Implementation

```
public class LinkedListNode<E> {  
    public E data;  
    public LinkedListNode<E> next;  
}
```

Linked List Implementation

```
// Inserts at the tail of the list

public void insert (E data) {
    LinkedListNode<E> newNode = new LinkedListNode<E>( );
    newNode.data = data;    // assign data to newNode
    tail.next = newNode;
    tail = newNode;
}
```

This method is oversimplified, what happens if this is called when the list is empty?

Linked List Implementation

Fixed insert method

```
// Inserts at the tail of the list

public void insert (E data) {
    LinkedListNode<E> newNode = new LinkedListNode<E>();
    newNode.data = data; // assign data to newNode
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail.next = newNode;
        tail = newNode;
    }
}
```

Linked List Implementation

```
// Inserts at the head of the list

public void insertAtHead (E data) {
    LinkedListNode<E> newNode = new LinkedListNode<E>();
    newNode.data = data;    // assign data to newNode
    newNode.next = head;
    head = newNode;
}
```

This method is oversimplified, what happens if this is called when the list is empty?

Linked List Implementation

```
// Inserts at the head of the list

public void insertAtHead (E data) {
    LinkedListNode<E> newNode = new LinkedListNode<E>();
    newNode.data = data;    // assign data to newNode
    newNode.next = head;

    if (!isEmpty()) {
        head = newNode;
    } else {
        head = tail = newNode;
    }
}
```

Linked List Implementation

```
// Removes node from head of list and returns its value
```

```
public E remove() {  
    if (head != null) {  
        E nodeData = head.data;  
        head = head.next;  
        return nodeData;  
    } else {  
        return null;  
    }  
}
```

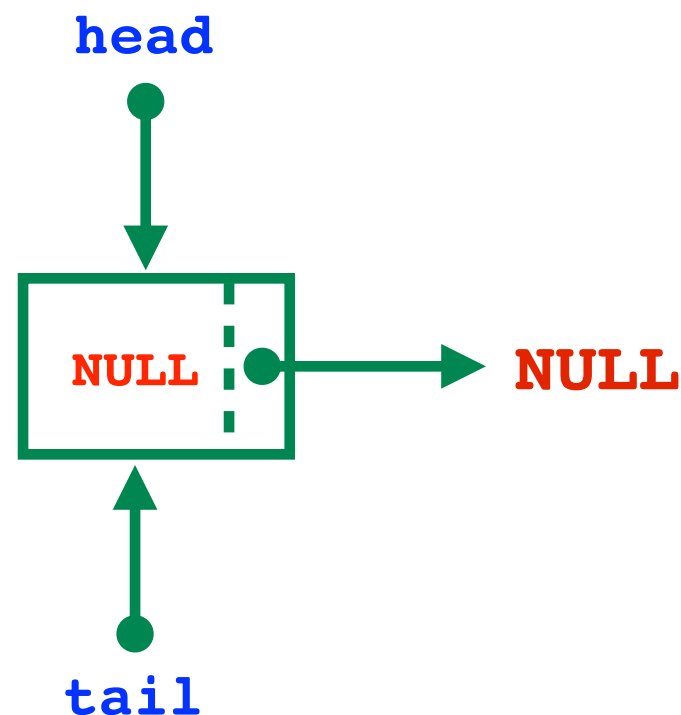

Considerations for Linked List Implementation

- Implementation as previously shown requires error checking in the **insert** and **remove** methods to check for edge cases (i.e. checking for an empty list)
- To improve the speed of the Linked List operations, it is possible to remove these tests
 - Tradeoff: speedup comes at the expense of one additional ‘dummy’ node in the Linked List
- Idea: create a dummy/sentinel node that exists in the linked list at ALL times ... created as part of the list serves as a **head** node
 - Eliminates the need to always check for **NULL**
 - Generalized the **insert** and **remove** methods

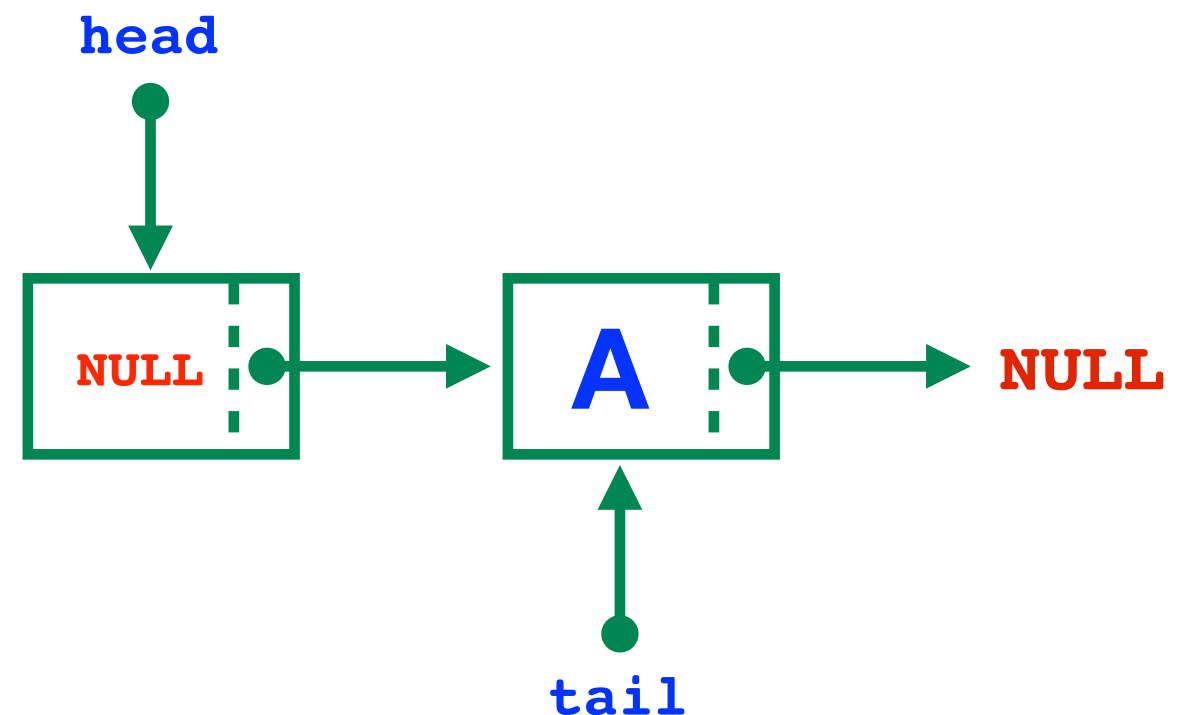
Linked List with Dummy/Sentinel Node

- Implementation as previously shown requires error checking in the **insert** and **remove** methods to check for edge cases (i.e. checking for an empty list)

Empty list



List with single node



Linked List Implementation

```
// Inserts at the tail of the list
// When using dummy header node, no need to test for null

public void insert (E data) {
    LinkedListNode<E> newNode = new LinkedListNode<E>();
    newNode.data = data;    // assign data to newNode
    tail.next = newNode;
    tail = newNode;
}
```