

CSE 220

Data Structures

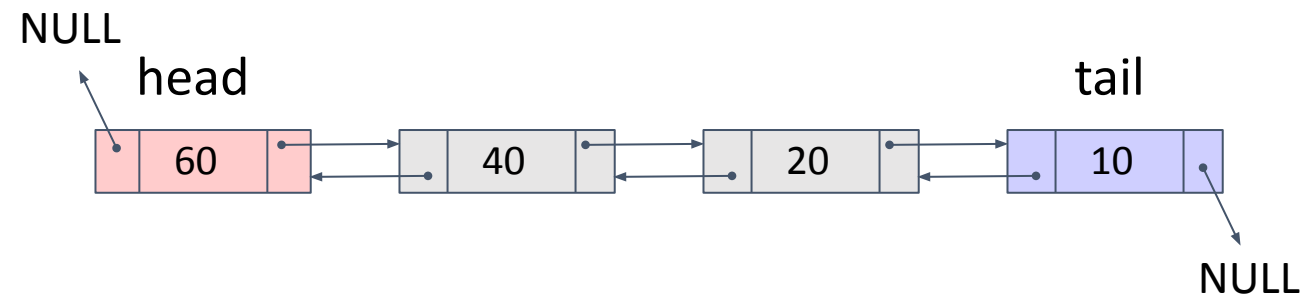
Lecture 05: Dummy Headed Circular DLLs

Anwarul Bashir Shuaib [AWBS]
Lecturer
Department of Computer Science and Engineering
BRAC University



Review: Doubly Linked Lists

- Each node contains **three** things:
 - Data
 - A reference to the **next** node
 - A reference to the **previous** node
- This version is also known as “Doubly Linked Lists (DLL)” as each node contains the reference to both the **next node and the previous node**.

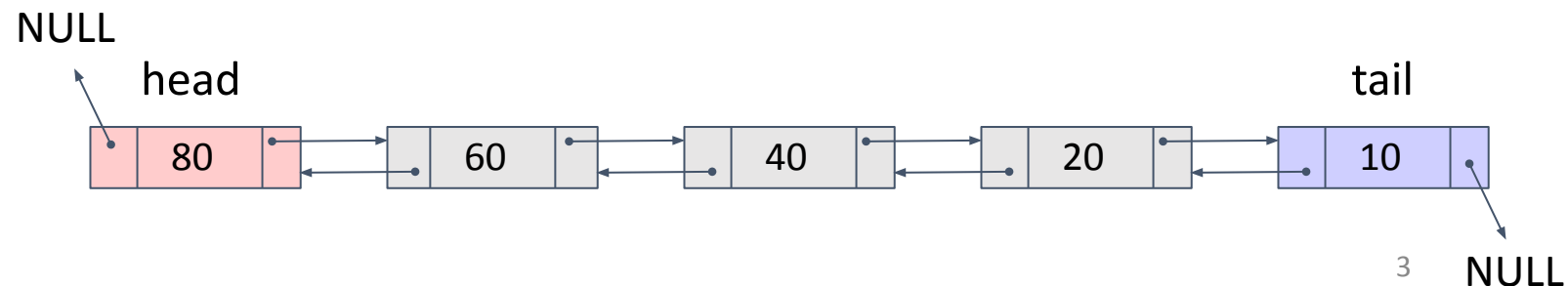


Review: Element Insertion (At the beginning)

Prepend 80 to the list

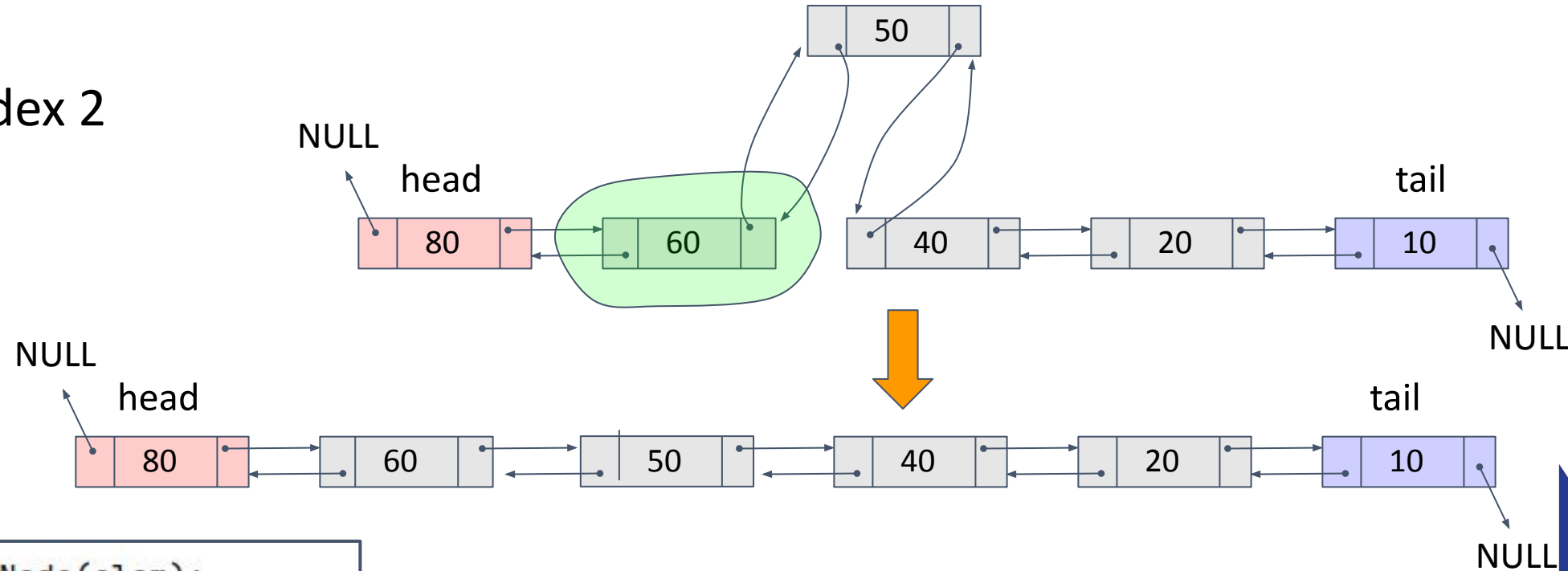
```
// Insert at the beginning
Node newNode = new Node(elem: 80);
newNode.next = head;
head.prev = newNode;
head = newNode;
```

$O(1)$



Review: Element Insertion (Middle)

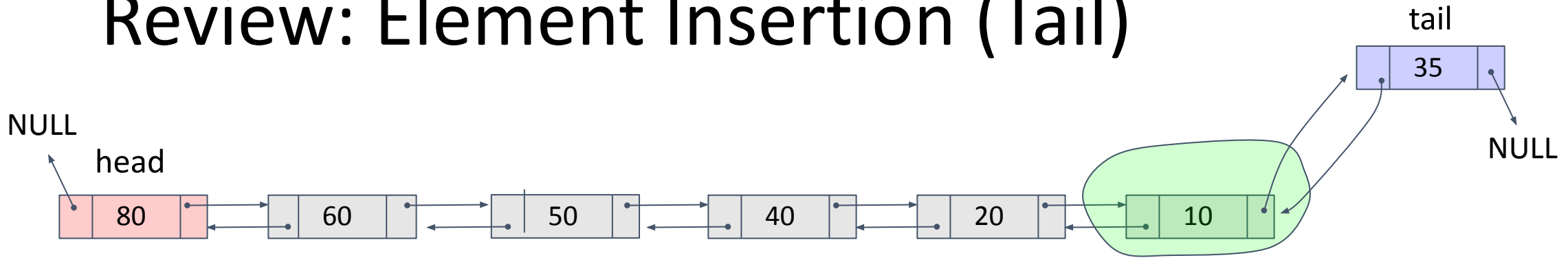
Insert 50 at index 2



```
Node newNode = new Node(elem);
Node prev = getNode(index: index - 1);
newNode.next = prev.next;
newNode.prev = prev;
prev.next = newNode;
newNode.next.prev = newNode;
```

$O(n)$

Review: Element Insertion (Tail)

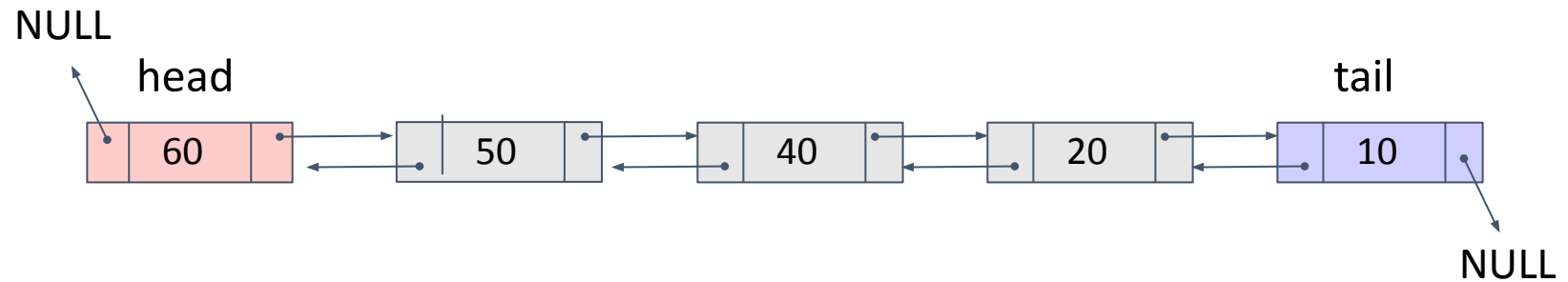


Can we optimize tail insertions?

```
Node newNode = new Node(elem);
Node prev = tail;
newNode.prev = prev;
prev.next = newNode;
tail = newNode;
```

$O(1)$

Review: Node Removal (Head)



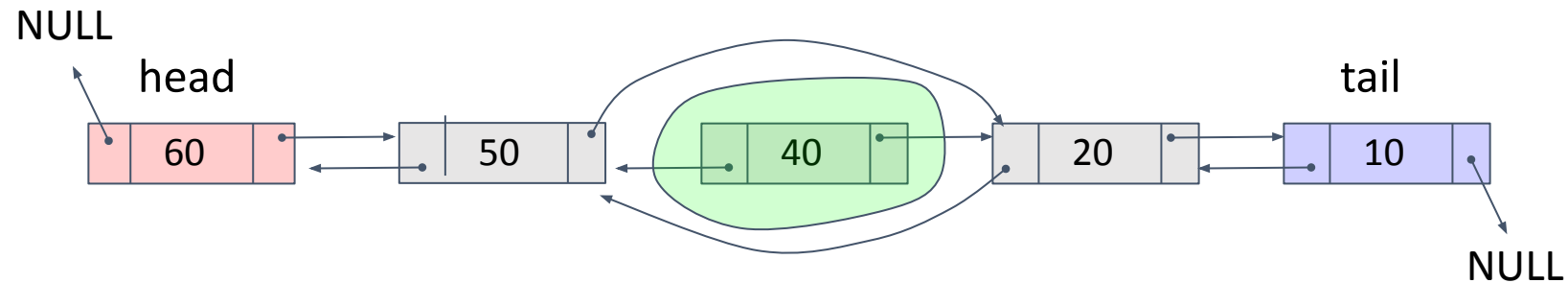
```
if (head == null) return;
head = head.next;
if (head == null) {
    tail = null;
    return;
}
head.prev = null;
```

Special case when list becomes empty

$O(1)$

Review: Node Removal (Middle)

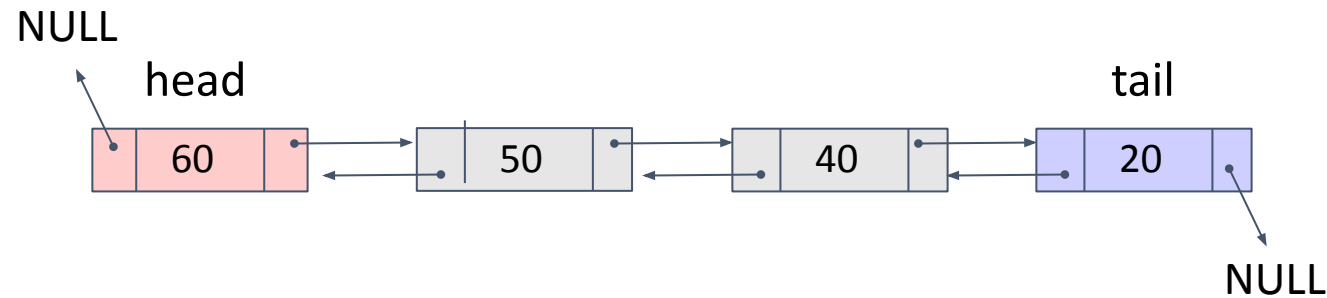
Delete the node at index 2



Garbage collected

```
Node node = getNode(index);  
node.prev.next = node.next;  
node.next.prev = node.prev;
```

Review: Node Removal (Tail)



Can we optimize tail deletions?

```
Node node = getNode(index);
if (node.next == null) { // Deleting the tail
    tail = node.prev;
    tail.next = null;
}
```

$O(n)$



```
tail = tail.prev;
tail.next = null;
```

$O(1)$

Doubly Linked Lists – Continued

- DLLs provide efficient node insertions and deletions on the head and tail.

But...

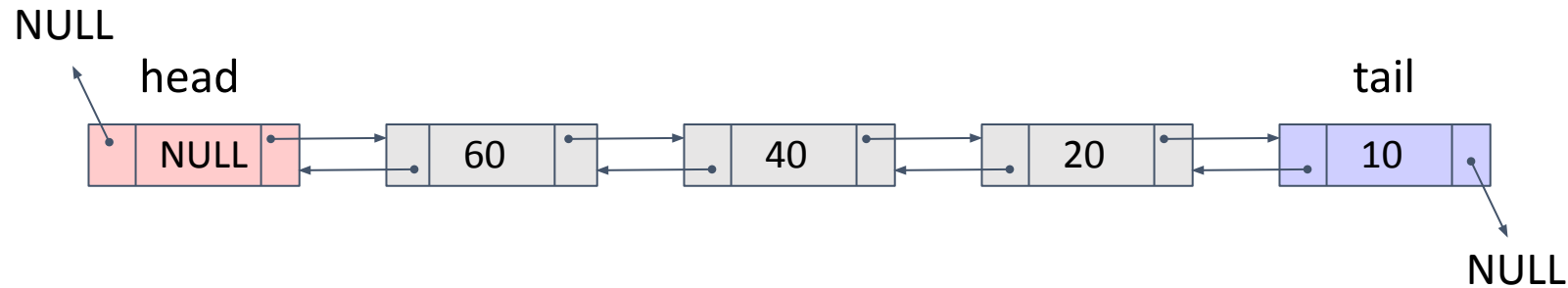
Problem in Traditional DLLs:

- If `head` is `null`, insertions and deletions require extra handling.
- When inserting at the beginning, `head` must be updated explicitly.
- Edge cases exist when deleting the first or last node.



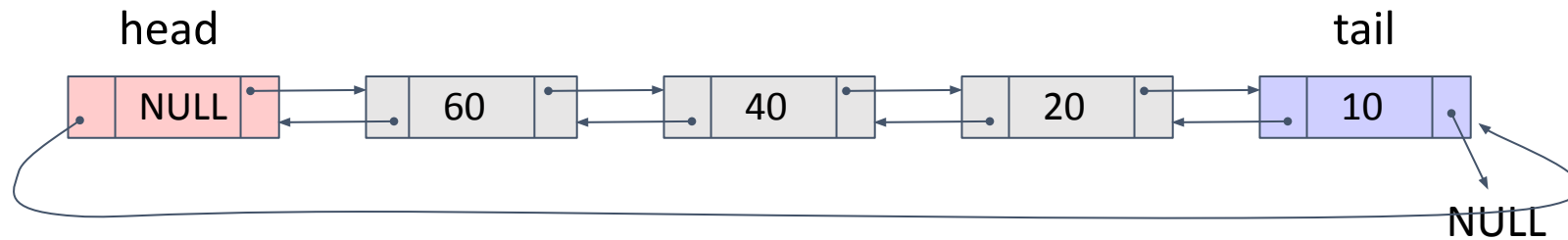
Dummy Headed Circular DLLs

- Almost same as DLLs, except:
 - The DLL has a dummy head (a head containing null as the element)
 - `head.prev` points to tail, and `tail.next` points to head.



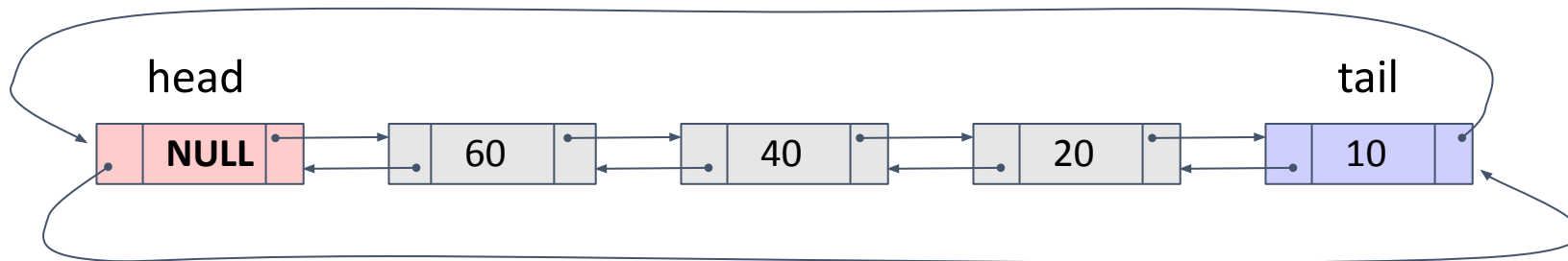
Dummy Headed Circular DLLs

- Almost same as DLLs, except:
 - The DLL has a dummy head (a head containing null as the element)
 - `head.prev` points to tail, and `tail.next` points to head.



Dummy Headed Circular DLLs

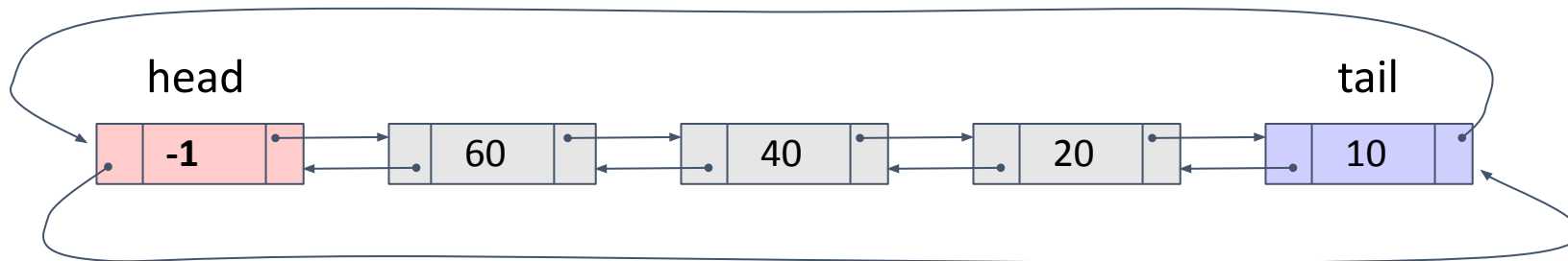
- Almost same as DLLs, except:
 - The DLL has a dummy head (a head containing null as the element)
 - `head.prev` points to tail, and `tail.next` points to head.



Some implementations use -1 instead of NULL as the dummy head element

Dummy Headed Circular DLLs

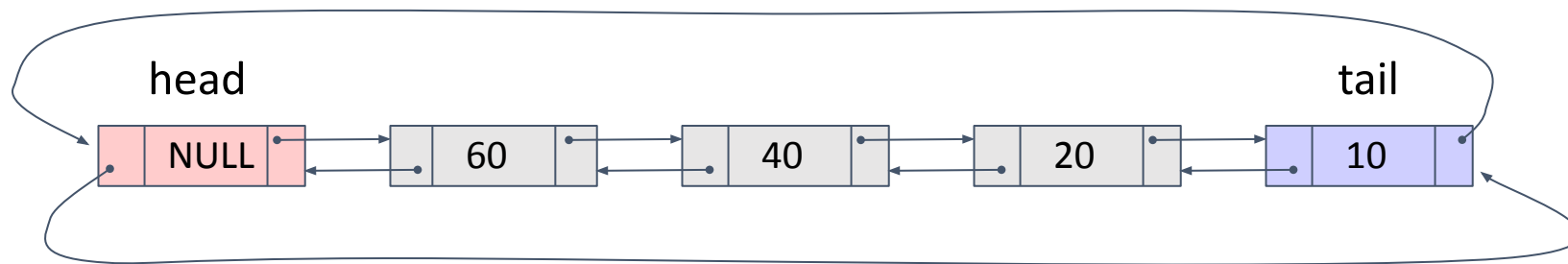
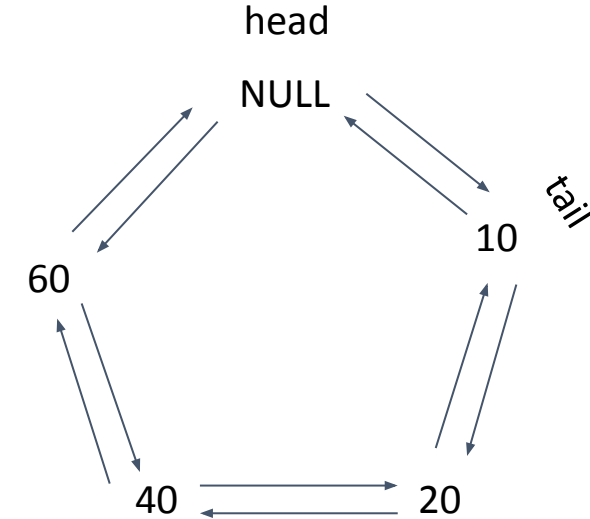
- Almost same as DLLs, except:
 - The DLL has a dummy head (a head containing null as the element)
 - `head.prev` points to tail, and `tail.next` points to head.



Some implementations use -1 instead of NULL as the dummy head element

Dummy Headed Circular DLLs

- Why called “Circular”?



Dummy Headed Circular DLLs

Advantage of Dummy Head:

- The dummy node **always exists**, so **head** is never **null**.
- Insertions at the start don't need to update **head** separately.
- Deletions are uniform—no need to check if the node is the first or last.



Dummy Headed DLL Class

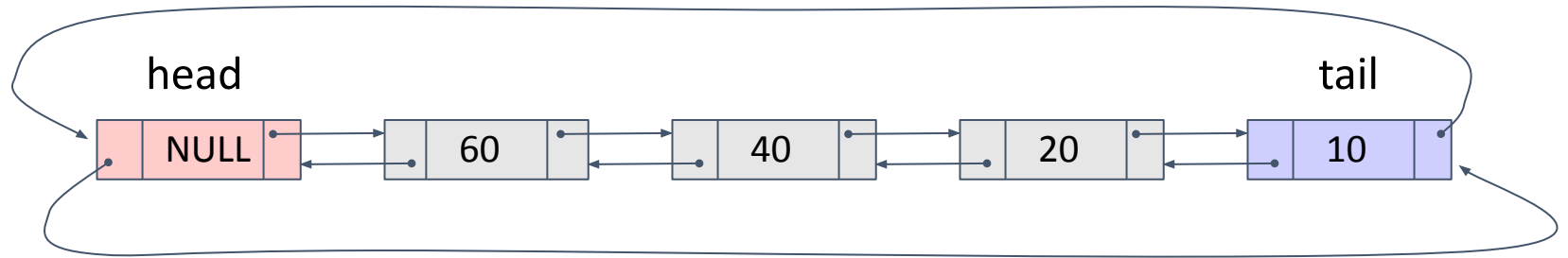
DLL Class

```
private static class Node {  
    int elem;  
    Node next;  
    Node prev;  
  
    Node(int elem) {  
        this.elem = elem;  
        this.next = null;  
        this.prev = null;  
    }  
}  
  
private Node head;  
private Node tail;
```

Optional

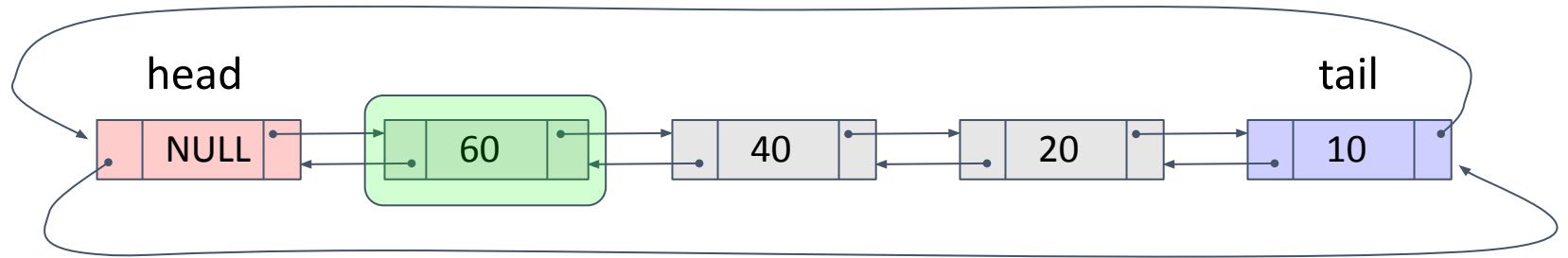
```
public DoublyCircularLinkedList() {  
    head = new Node(elem: -1); // Dummy node  
    head.next = head;  
    head.prev = head;  
}
```


Iteration



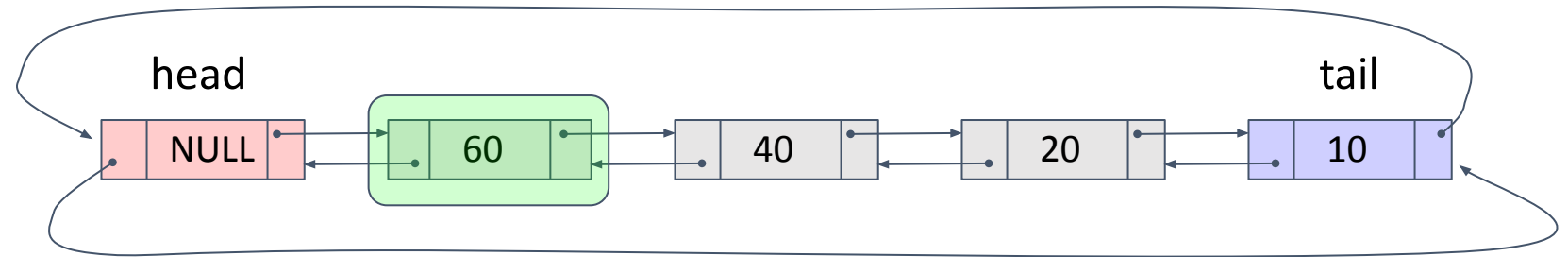
```
// 2. Iteration of the linked list  
public void iterate() {  
  
  
  
  
  
  
  
  
  
}
```

Iteration



```
// 2. Iteration of the linked list  
public void iterate() {  
    Node current = head.next;  
  
}
```

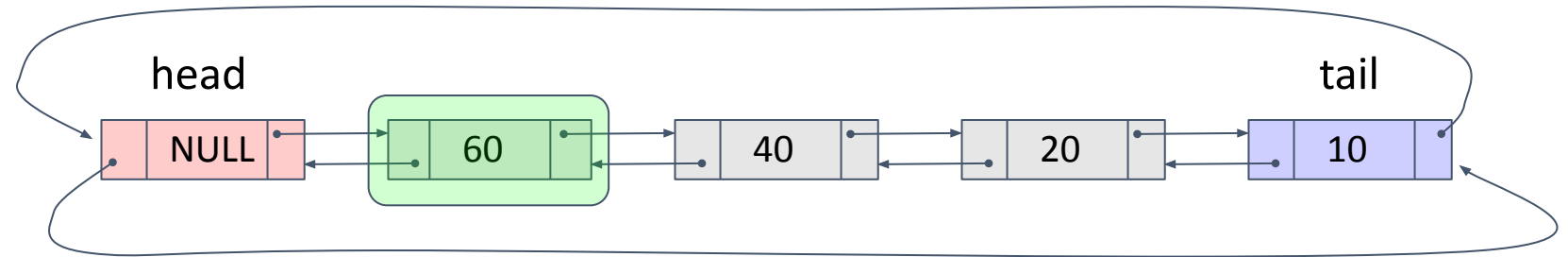
Iteration



```
// 2. Iteration of the linked list
public void iterate() {
    Node current = head.next;
    while (current != head) {

    }
}
```

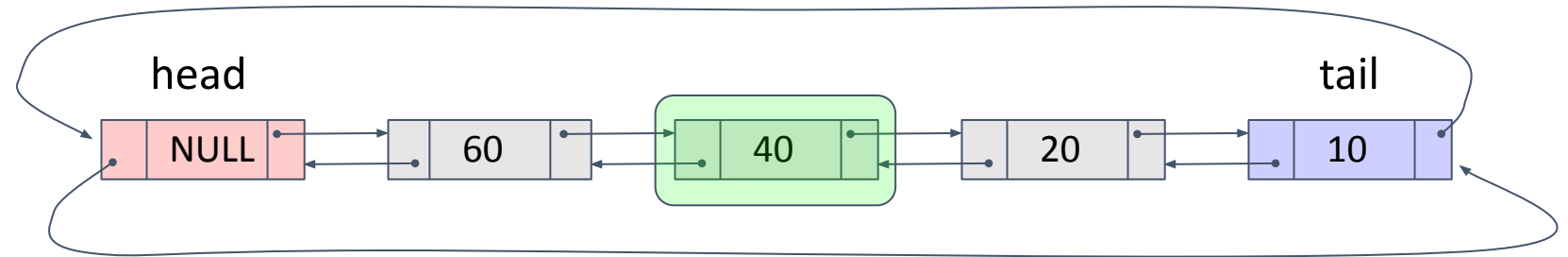
Iteration



```
// 2. Iteration of the linked list
public void iterate() {
    Node current = head.next;
    while (current != head) {
        System.out.print(current.elem + " ");
    }
}
```

60

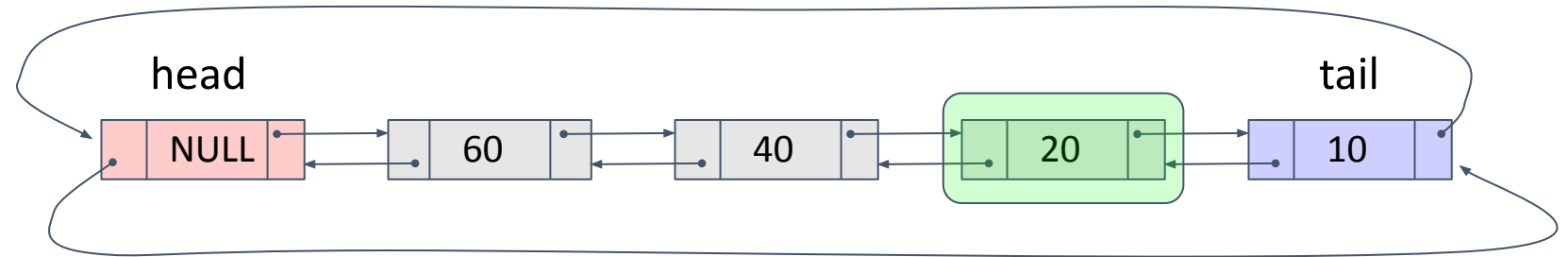
Iteration



```
// 2. Iteration of the linked list
public void iterate() {
    Node current = head.next;
    while (current != head) {
        System.out.print(current.elem + " ");
        current = current.next;
    }
}
```

60 40

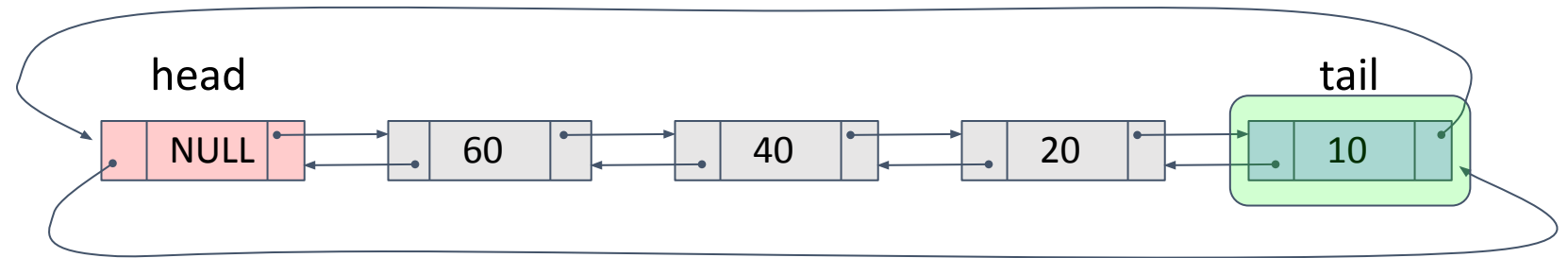
Iteration



```
// 2. Iteration of the linked list
public void iterate() {
    Node current = head.next;
    while (current != head) {
        System.out.print(current.elem + " ");
        current = current.next;
    }
}
```

60 40 20

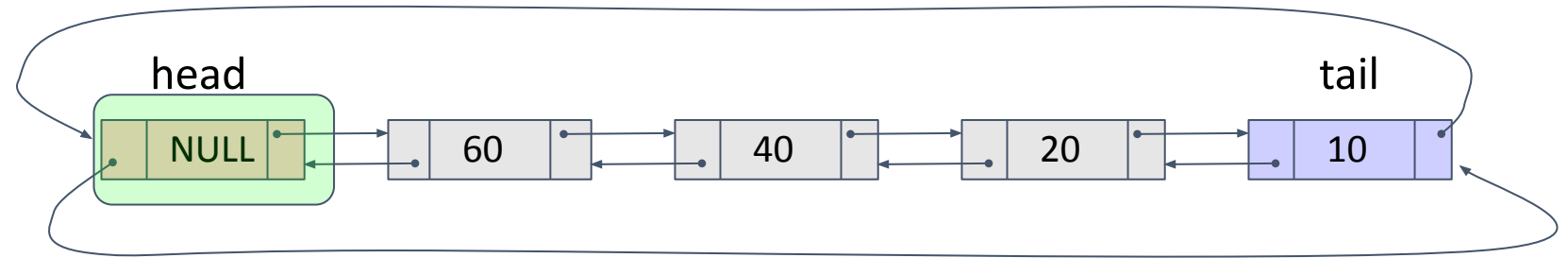
Iteration



```
// 2. Iteration of the linked list
public void iterate() {
    Node current = head.next;
    while (current != head) {
        System.out.print(current.elem + " ");
        current = current.next;
    }
}
```

60 40 20 10

Iteration



// 2. Iteration of the linked list

```
public void iterate() {
```

```
    Node current = head.next;
```

```
    while (current != head) {
```

```
        System.out.print(current.elem + " ");
```

```
        current = current.next;
```

```
    }
```

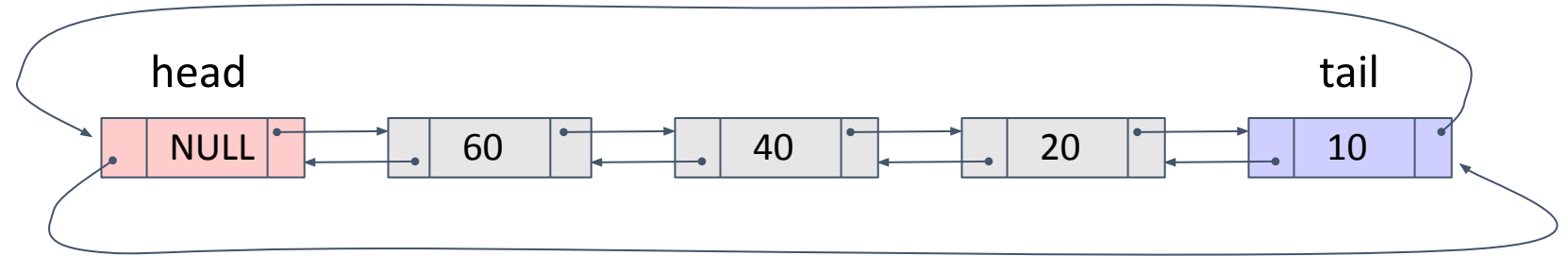
```
}
```

Termination

60 40 20 10

InsertAtStart

Insert 30 at start

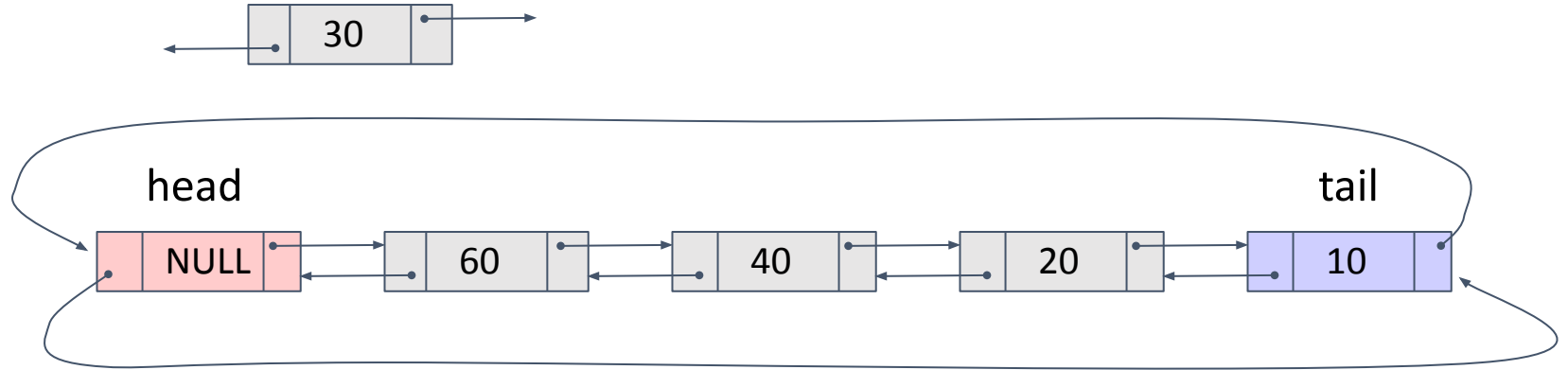


```
public void insertAtStart(int elem) {
```

}

InsertAtStart

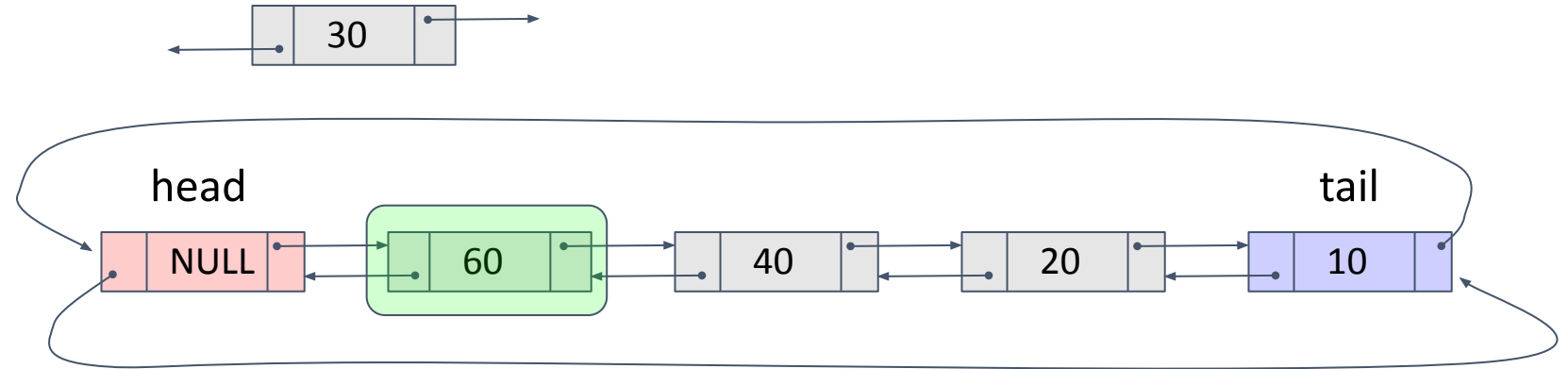
Insert 30 at start



```
public void insertAtStart(int elem) {  
    Node newNode = new Node(elem);  
  
    }  
}
```

InsertAtStart

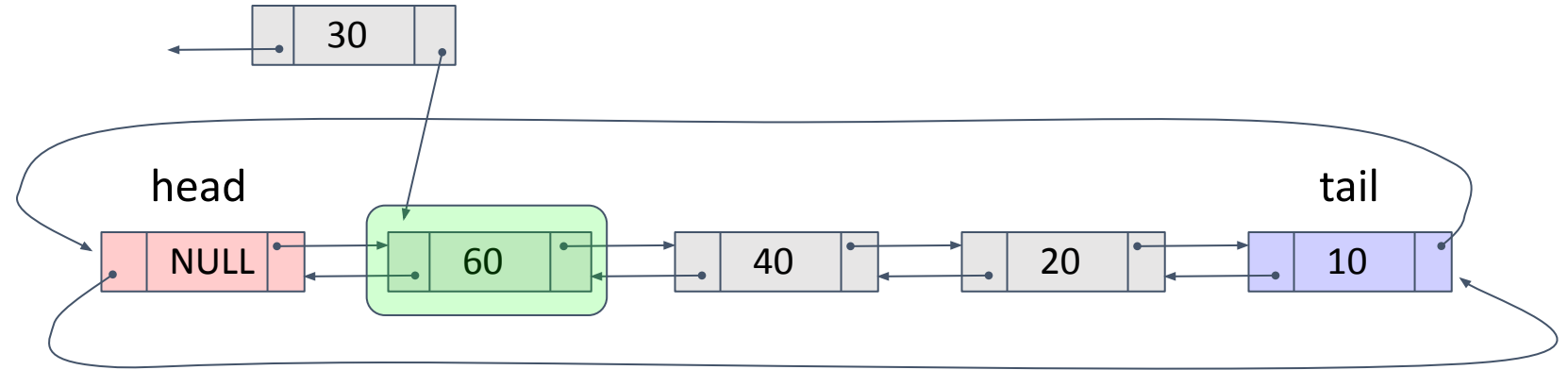
Insert 30 at start



```
public void insertAtStart(int elem) {
    Node newNode = new Node(elem);
    Node first = head.next;
}
```

InsertAtStart

Insert 30 at start



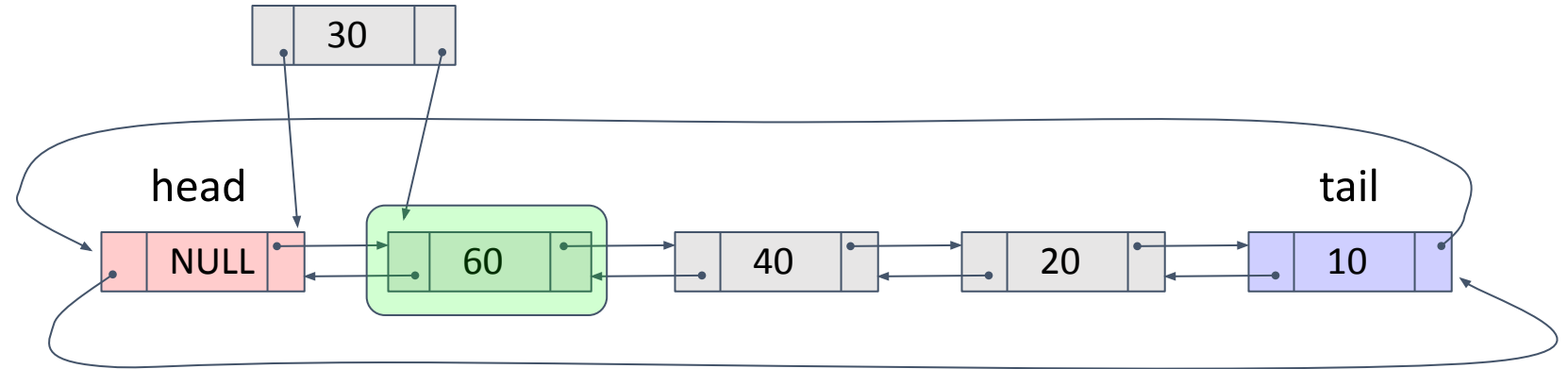
```
public void insertAtStart(int elem) {
    Node newNode = new Node(elem);
    Node first = head.next;

    newNode.next = first;

}
```

InsertAtStart

Insert 30 at start

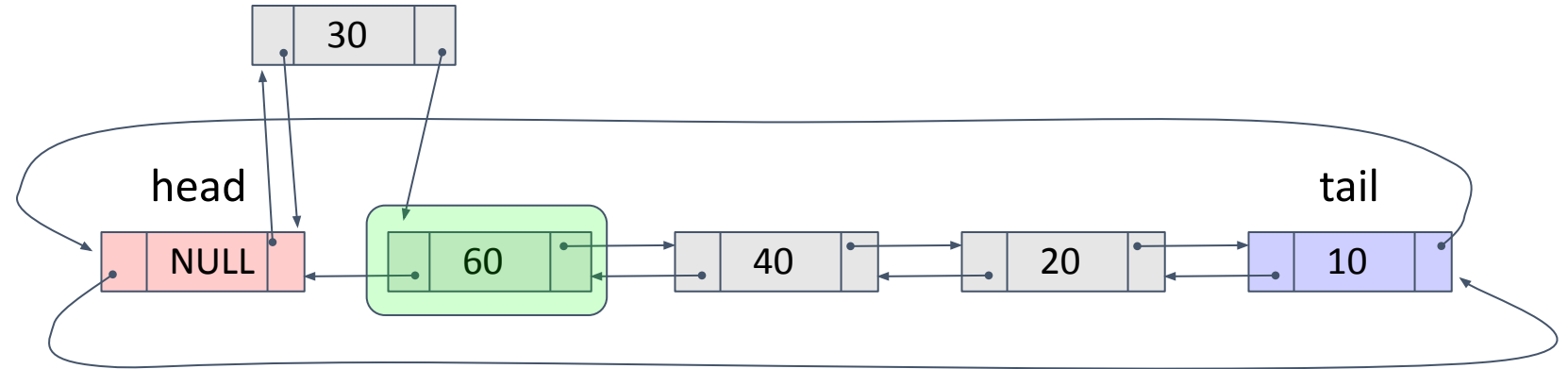


```
public void insertAtStart(int elem) {
    Node newNode = new Node(elem);
    Node first = head.next;

    newNode.next = first;
    newNode.prev = head;
}
```

InsertAtStart

Insert 30 at start

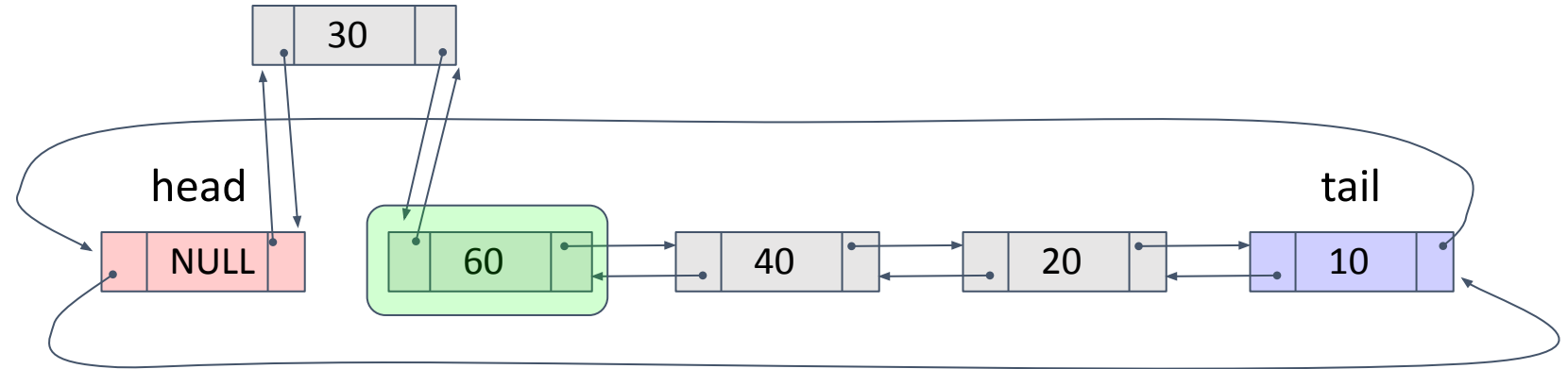


```
public void insertAtStart(int elem) {
    Node newNode = new Node(elem);
    Node first = head.next;

    newNode.next = first;
    newNode.prev = head;
    head.next = newNode;
}
```

InsertAtStart

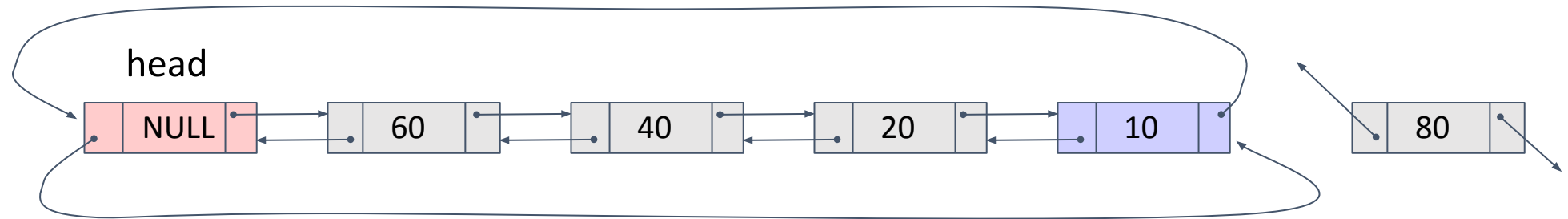
Insert 30 at start



```
public void insertAtStart(int elem) {
    Node newNode = new Node(elem);
    Node first = head.next;

    newNode.next = first;
    newNode.prev = head;
    head.next = newNode;
    first.prev = newNode;
}
```

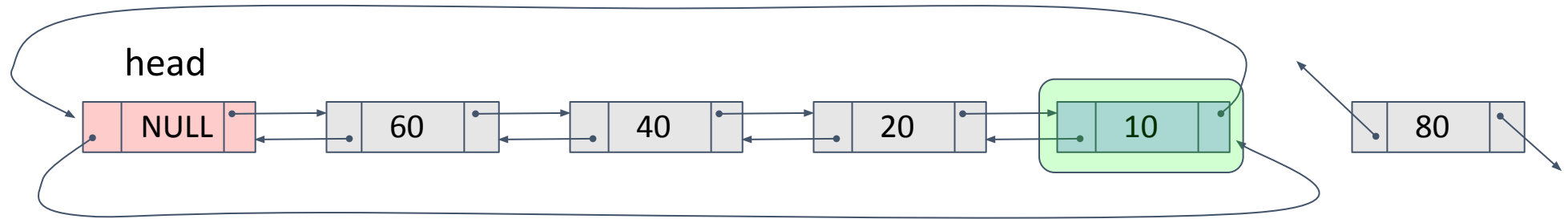
InsertAtEnd



```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    // ... (rest of the code)
}
```

Insert 80 at end

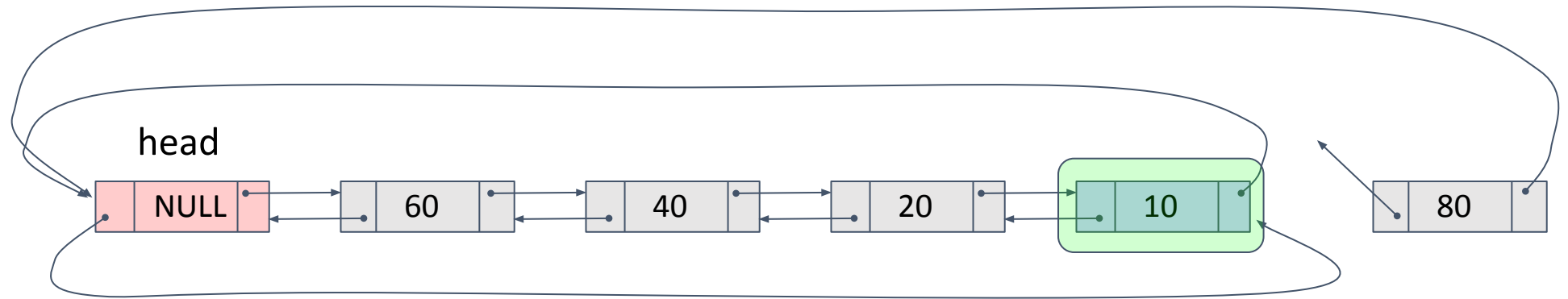
InsertAtEnd



```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    Node last = head.prev;
}
```

Insert 80 at end

InsertAtEnd

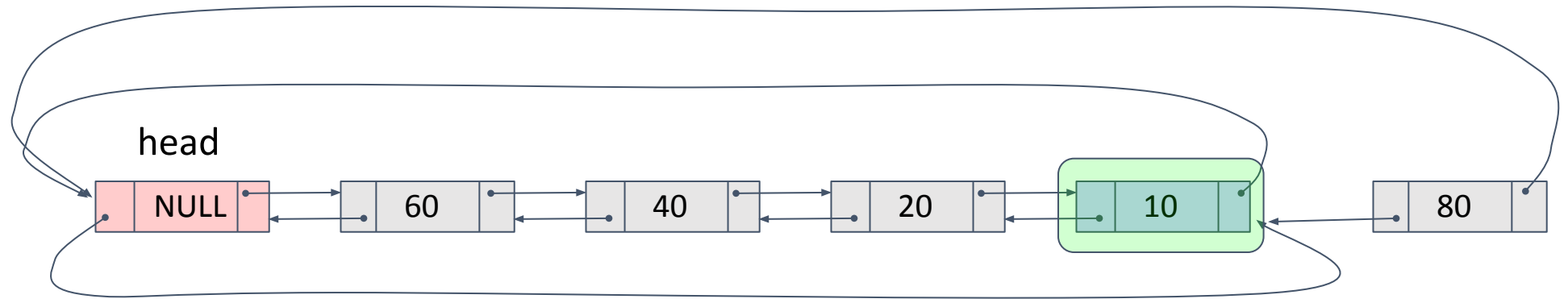


```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    Node last = head.prev;

    newNode.next = head;
}
```

Insert 80 at end

InsertAtEnd

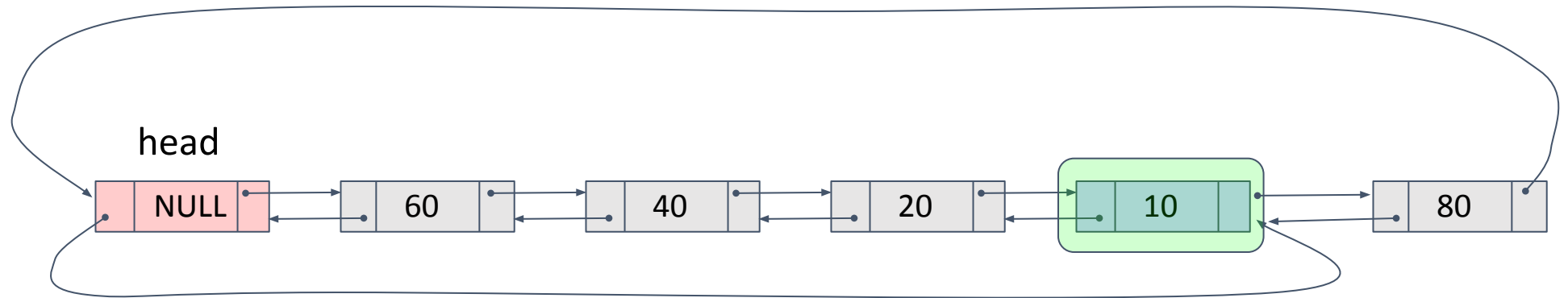


```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    Node last = head.prev;

    newNode.next = head;
    newNode.prev = last;
}
```

Insert 80 at end

InsertAtEnd

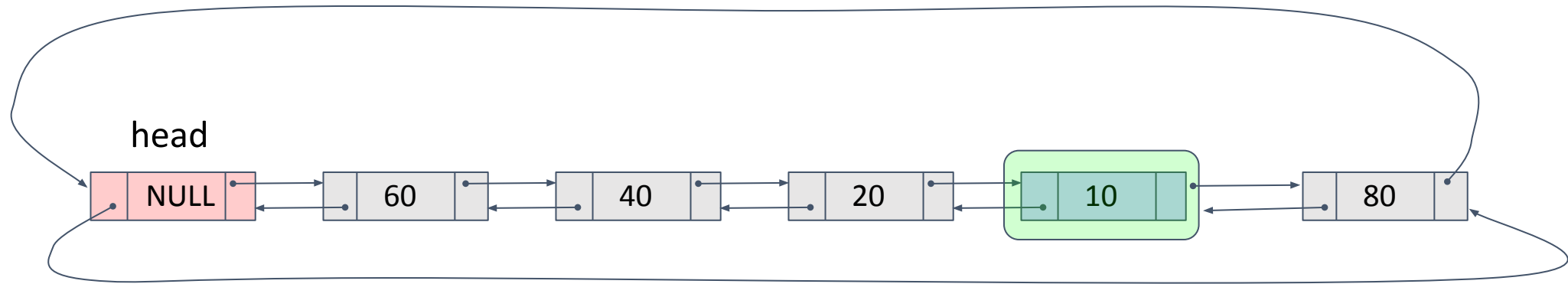


```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    Node last = head.prev;

    newNode.next = head;
    newNode.prev = last;
    last.next = newNode;
}
```

Insert 80 at end

InsertAtEnd



```
public void insertAtEnd(int elem) {
    Node newNode = new Node(elem);
    Node last = head.prev;

    newNode.next = head;
    newNode.prev = last;
    last.next = newNode;
    head.prev = newNode;
}
```

Insert 80 at end

Insert Anywhere

```
public void insertAtIndex(int index, int elem) {  
    if (index < 0) return;  
  
    if (index == 0) {  
        insertAtStart(elem);  
        return;  
    }  
  
    Node current = head.next;  
    int currentIndex = 0;  
  
    while (current != head) {  
        if (currentIndex == index) {  
            Node newNode = new Node(elem);  
            Node prevNode = current.prev;  
  
            newNode.next = current;  
            newNode.prev = prevNode;  
            prevNode.next = newNode;  
            current.prev = newNode;  
  
            return;  
        }  
        currentIndex++;  
        current = current.next;  
    }  
  
    if (currentIndex == index) {  
        insertAtEnd(elem);  
    }  
}
```

Insert Anywhere

Similar to DLLs

```
public void insertAtIndex(int index, int elem) {  
    if (index < 0) return;  
  
    if (index == 0) {  
        insertAtStart(elem);  
        return;  
    }  
  
    Node current = head.next;  
    int currentIndex = 0;  
  
    while (current != head) {  
        if (currentIndex == index) {  
            Node newNode = new Node(elem);  
            Node prevNode = current.prev;  
  
            newNode.next = current;  
            newNode.prev = prevNode;  
            prevNode.next = newNode;  
            current.prev = newNode;  
  
            return;  
        }  
        currentIndex++;  
        current = current.next;  
    }  
  
    if (currentIndex == index) {  
        insertAtEnd(elem);  
    }  
}
```

Creation

```
public void createFromArray(int[] arr) {  
    for (int elem : arr) {  
        insertAtEnd(elem);  
    }  
}
```



Removal

```
public void remove(int index) {  
    if (index < 0) return;  
  
    Node current = head.next;  
    int currentIndex = 0;  
  
    while (current != head) {  
        if (currentIndex == index) {  
            Node prevNode = current.prev;  
            Node nextNode = current.next;  
  
            prevNode.next = nextNode;  
            nextNode.prev = prevNode;  
  
            return;  
        }  
        currentIndex++;  
        current = current.next;  
    }  
    System.out.println("Index out of bounds");  
}
```



Exercise

- Reverse a dummy headed circular DLL
- Detect cycles (hint: hare and tortoise algorithm)

Practice Problems

- Browser history design: <https://leetcode.com/problems/design-browser-history>
- Authentication manager: <https://leetcode.com/problems/design-authentication-manager>

