

Lecture 12: Doubly Linked Lists

CS 62

Spring 2018

Alexandra Papoutsaki & William Devanny


```

public class DoublyLinkedListNode<E>{
    protected E data; // value stored in this element
    protected DoublyLinkedListNode<E> nextElement; // ref to next
    protected DoublyLinkedListNode<E> previousElement; // ref to previous

    public DoublyLinkedListNode(E v, DoublyLinkedListNode<E> next, DoublyLinkedListNode<E> previous) {
        data = v;
        nextElement = next;
        if(nextElement != null)
            nextElement.previousElement = this;
        previousElement = previous;
        if(previousElement != null)
            previousElement.nextElement = this;
    }
}

public DoublyLinkedListNode(E v) {
    this(v,null,null); // constructs a single element
}

//setters and getters
}

```

DoublyLinkedList

```
public class DoublyLinkedList<E> extends AbstractList<E> {
    protected int count; // number within list
    protected DoublyLinkedListNode<E> head; // ref. to first element
    protected DoublyLinkedListNode<E> tail; // ref. to last element

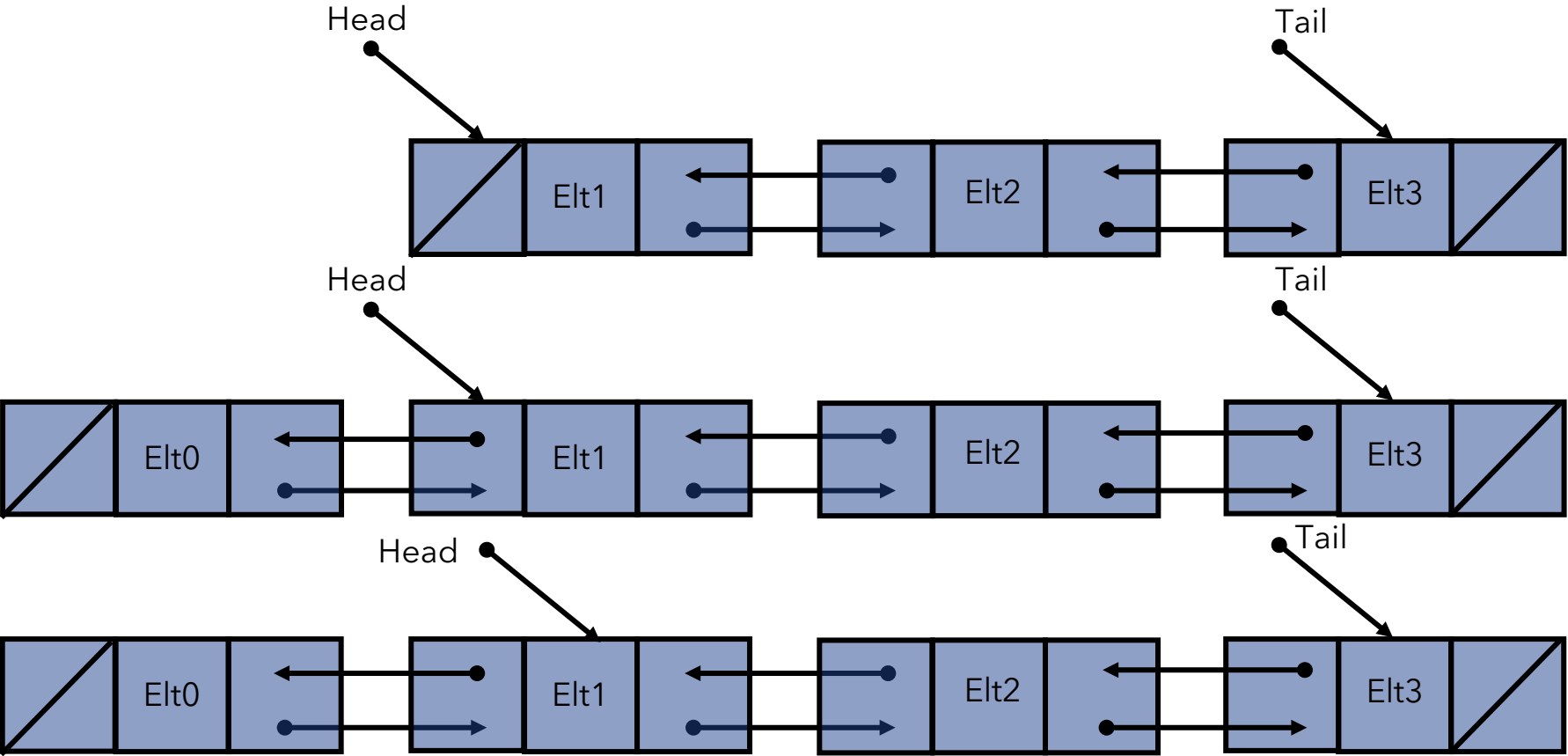
    //construct an empty list
    public DoublyLinkedList() {
        head = null;
        tail = null;
        count = 0;
    }

    public E getFirst() {
        return head.value(); //returns first value in list
    }

    public E getLast() {
        return tail.value(); //returns lastvalue in list
    }

    public int size() {
        return count;
    }
}
```

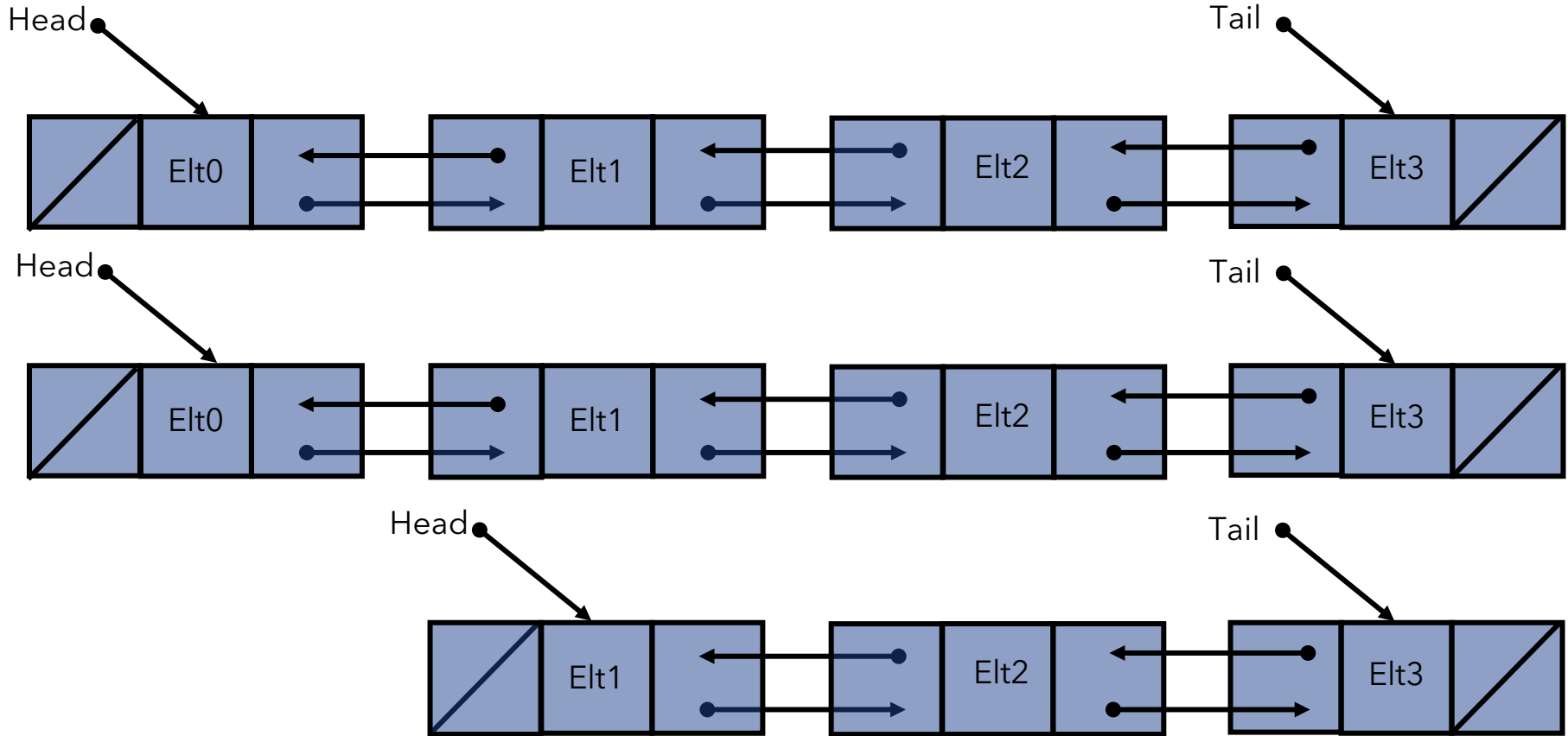
Adding at the head



Adding at the head is $O(1)$

```
public void addFirst(E value){  
    // construct a new node making it head  
    head = new DoublyLinkedListNode<E>(value, head, null);  
    if(tail == null) //list was empty  
        tail = head;  
    count++;  
}
```

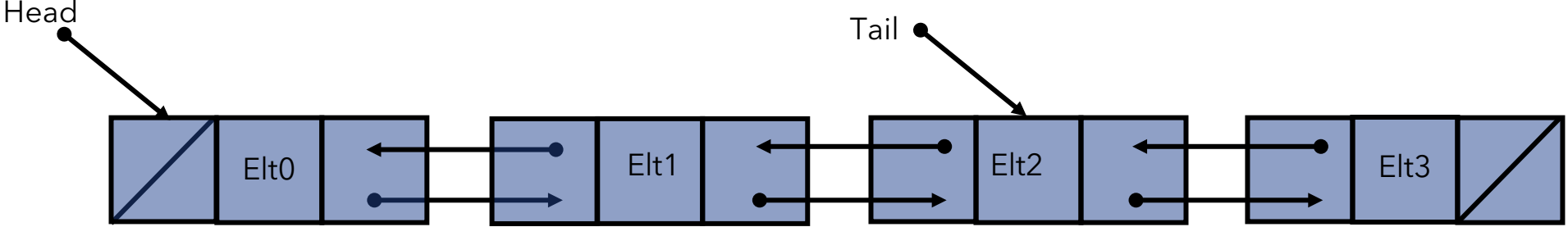
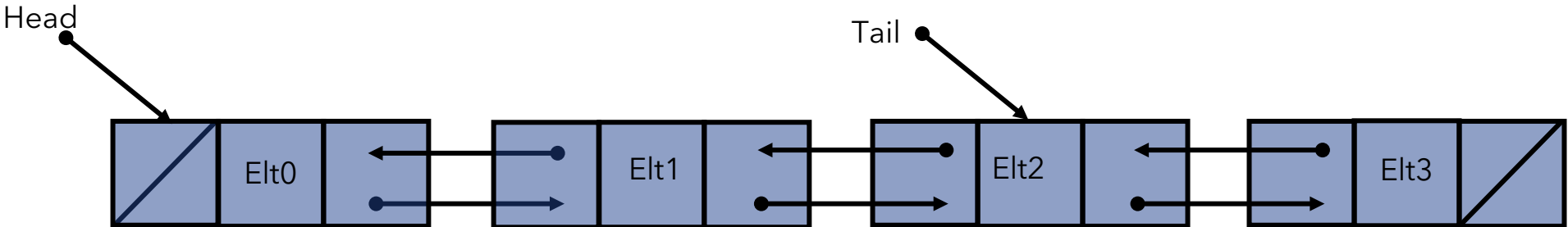
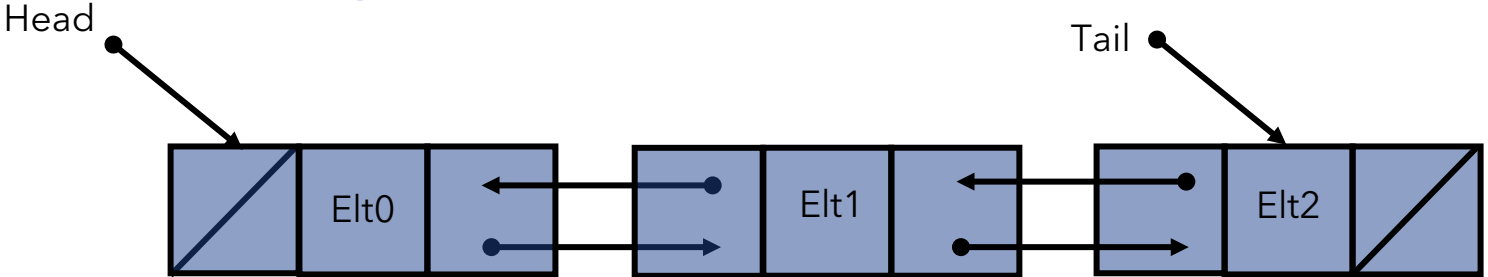
Removing at the head



Removing at the head is $O(1)$

```
public E removeFirst(){
    //check that list is not empty
    DoublyLinkedListNode<E> temp = head;
    head = head.next(); // move head down list
    if (head != null)
        head.setPrevious(null);
    else
        tail = null; //remove final value
    count--;
    return temp.value();
}
```

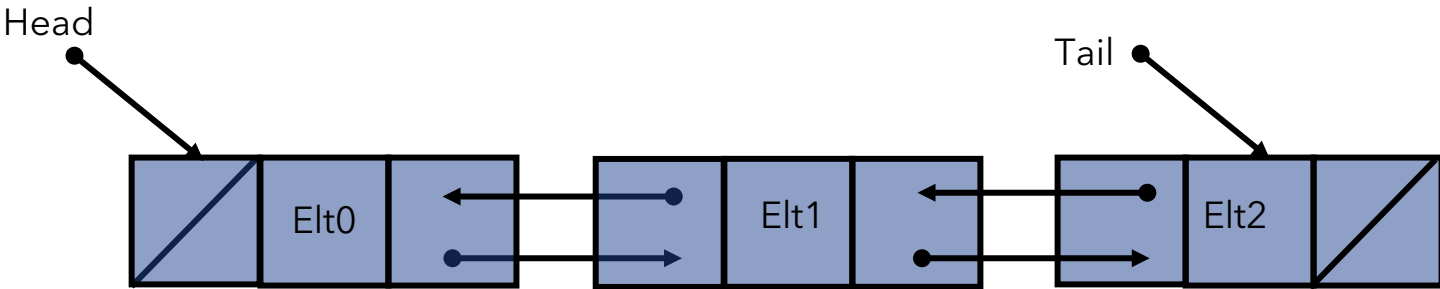
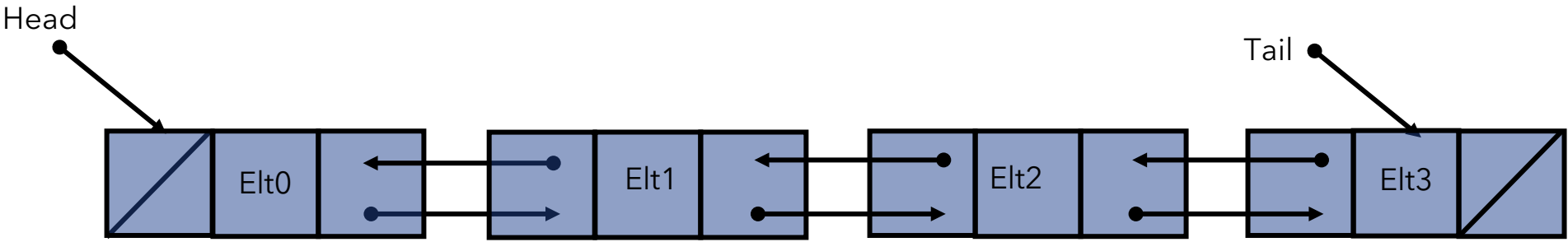
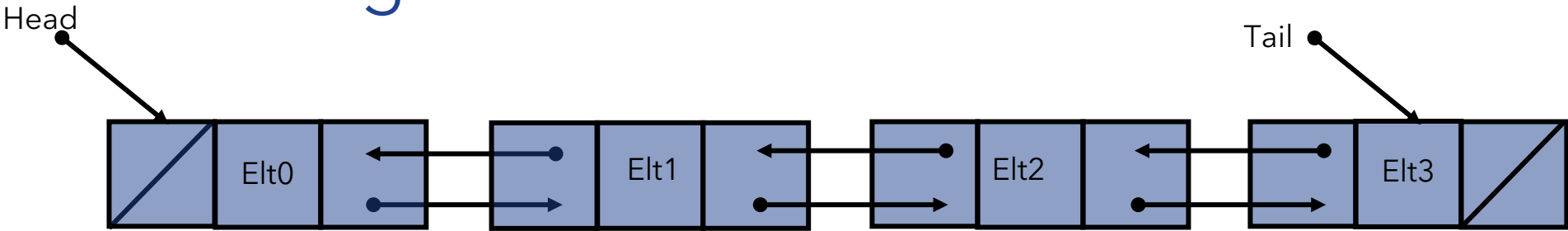

Adding at the tail



Adding at the tail is $O(1)$

```
public void addLast(E value) {  
    tail = new DoublyLinkedListNode<E>(value, null, tail);  
    if (head == null)  
        head = tail; //fix up head, it's the first node  
    count++;  
}
```

Removing from the tail

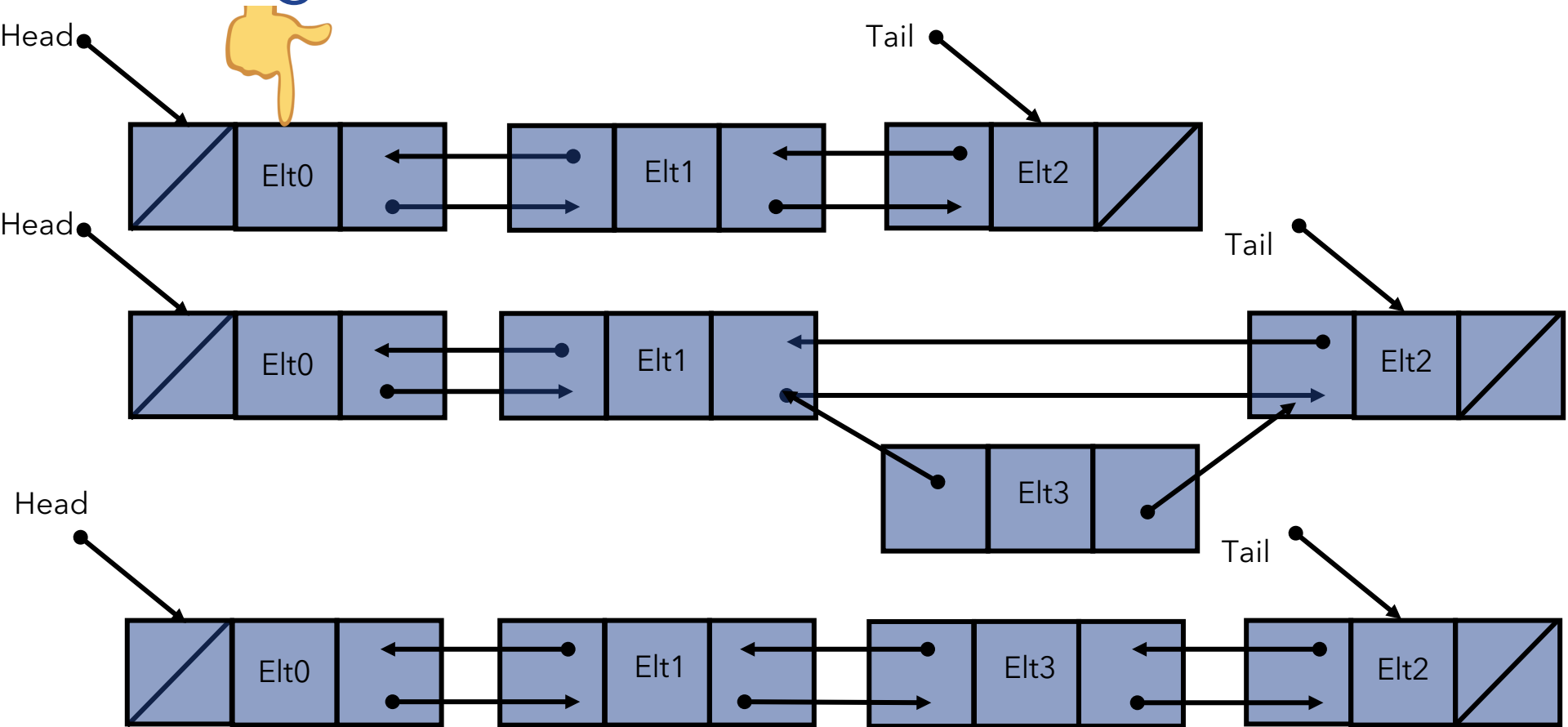


Removing from the end is $O(1)$

```
public E removeLast() {  
    //check that the list is not empty  
    DoublyLinkedListNode<E> temp= tail;  
    tail = tail.previous();  
    if (tail == null)  
        head = null; //empty list  
    else  
        tail.setNext(null);  
    count--;  
    return temp.value();  
}
```

Adding between nodes

Example: `add(2, Elt3)`

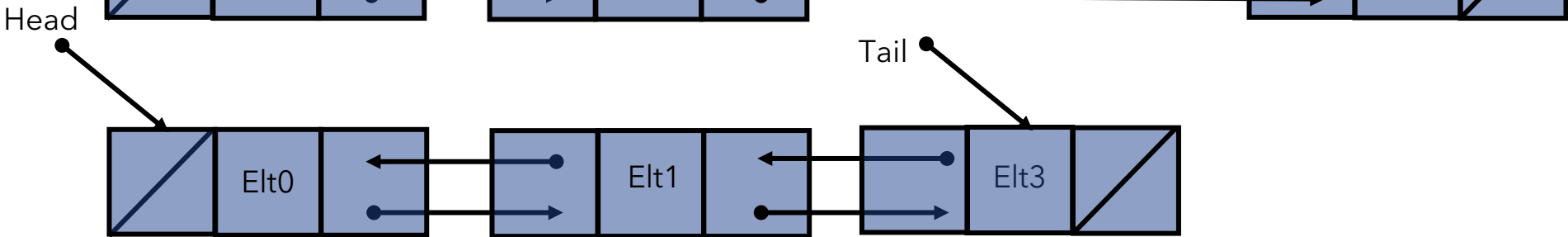
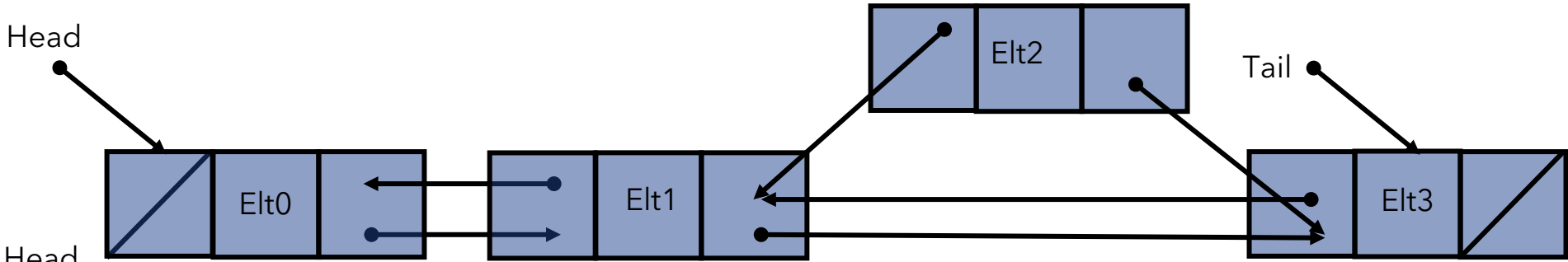
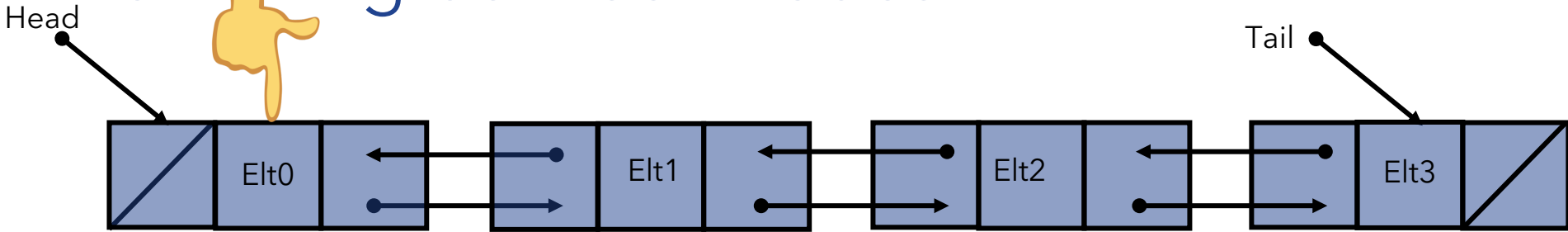


Adding between nodes is $O(n)$

```
public void add(int i, E o) {  
    //check that i in range. If i==0 call addFirst,  
    //if i==size() call addLast()  
    DoublyLinkedListNode<E> before = null;  
    DoublyLinkedListNode<E> after = head;  
    while (i > 0) { // search for ith position, or end of list  
        before = after;  
        after = after.next();  
        i--;  
    }  
    // create new value to insert in correct position  
    DoublyLinkedListNode<E> current = new DoublyLinkedListNode<E>(o,after,before);  
    count++; // make after and before value point to new value  
    before.setNext(current);  
    after.setPrevious(current);  
}
```

Removing between nodes

Example: `remove(2)`



Removing between nodes is $O(n)$

```
public E remove(int i) {  
    //check that i in range. If i==0 call removeFirst,  
    //if i==size() call removeLast()  
    DoublyLinkedListNode<E> previous = null;  
    DoublyLinkedListNode<E> finger= head;  
    while (i > 0) { // search for ith position, or end of list  
        previous = finger;  
        finger = finger.next();  
        i--;  
    }  
    previous.setNext(finger.next());  
    finger.next().setPrevious(previous);  
    count--;  
    return finger.value();  
}
```


More on Linked Lists

`structure5` provides iterators both for singly and doubly linked lists → Check the code!

[java.util.LinkedList](#) implements a Doubly Linked List

Doubly linked lists are often represented as circular:

