

Introduction to Computer Science II

CS112-2012S-20

Linked Lists III

David Galles

Department of Computer Science
University of San Francisco

20-0: **Linked List**

- Previous Practical Example:
 - `removeAt(int index)`
 - `remove(Object o)`

20-1: **removeAt**

- First need to get to node *before* the one we want to remove
- We can then use the next pointer of the previous node to do removal (example on board)

20-2: Special Cases

- Two special cases:
 - Removing from an empty list
 - Removing the first element in the list
- Why do we need these two special cases?

20-3: removeAt

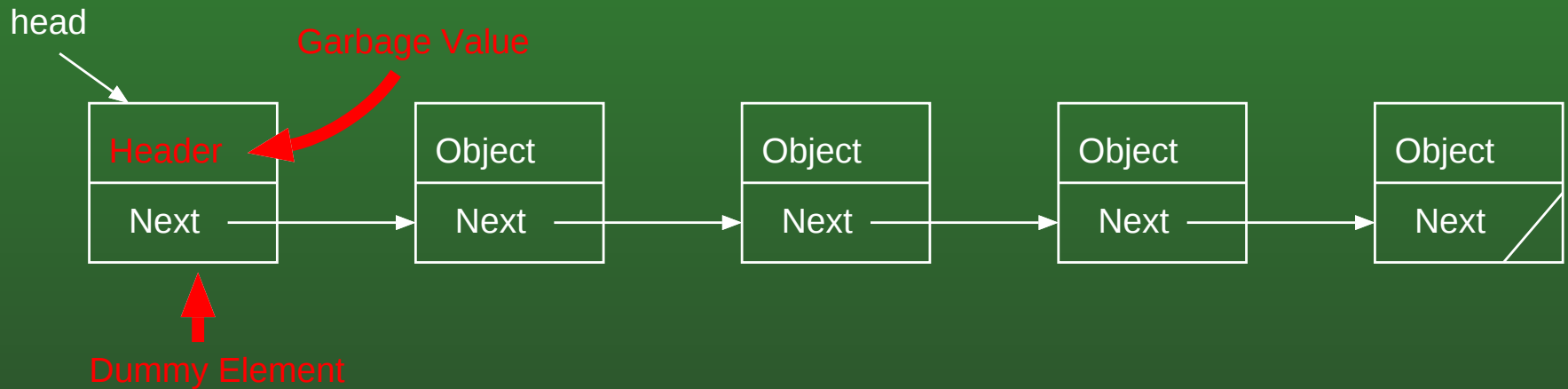
```
public class LinkedList
{
    private ListNode head;
    public void removeAt(int index)
    {
        if (head == null)
            return null; // May want to throw an exception here instead ...
        if (index == 0)
        {
            head = head.next;
        }
        else
        {
            ListNode tmp = head;
            for (int i = 0; i < index - 1; i++)
            {
                tmp = tmp.next;
            }
            tmp.next = tmp.next.next;
        }
    }
}
```

20-4: Special Cases

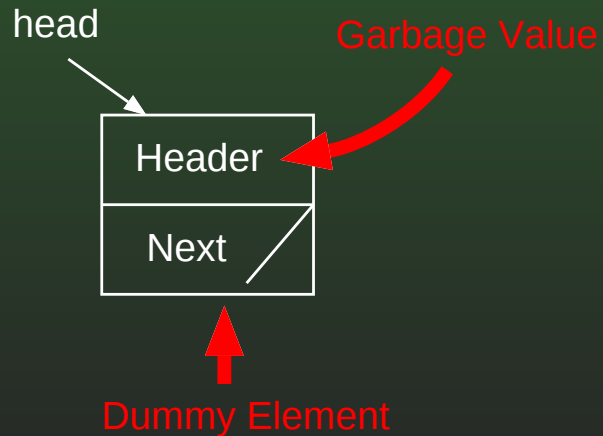
- Two special cases:
 - Removing from an empty list
 - Removing the first element in the list
- We can remove these special cases by using a dummy element

20-5: Dummy Element

Linked List with Dummy element



Empty List



20-6: Dummy Element

```
class LinkedList
{
    ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    // Other methods ...

}
```

20-7: find, Dummy Element

```
class LinkedList
{
    ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    boolean find(Object o) { ... }
}
```

20-8: find, Dummy Element

```
class LinkedList
{
    ListNode head;

    boolean find(Object o)
    {
        for (ListNode tmp = head.next; tmp != null; tmp = tmp.next)
        {
            if (tmp.data.equals(o))
            {
                return true;
            }
        }
        return false;
    }
}
```

20-9: RemoveAt, Dummy Element

```
public class LinkedList
{
    private ListNode head;

    LinkedList()
    {
        head = new ListNode(null, null);
    }

    public void removeAt(int index) { ... }

}
```

20-10: RemoveAt, Dummy Element

```
public class LinkedList
{
    private ListNode head;
    public void removeAt(int index)
    {
        ListNode tmp = head;
        for (int i = 0; i < index; i++)
        {
            tmp = tmp.next;
        }
        tmp.next = tmp.next.next;
    }
}
```

20-11: Append

- Append an element to the end of our list
- How would we do it?

20-12: Append

- Append an element to the end of our list
- How would we do it?
 - Get a pointer to the last element of the list
 - (start from the front, advance until last element reached)
 - Append element using this pointer

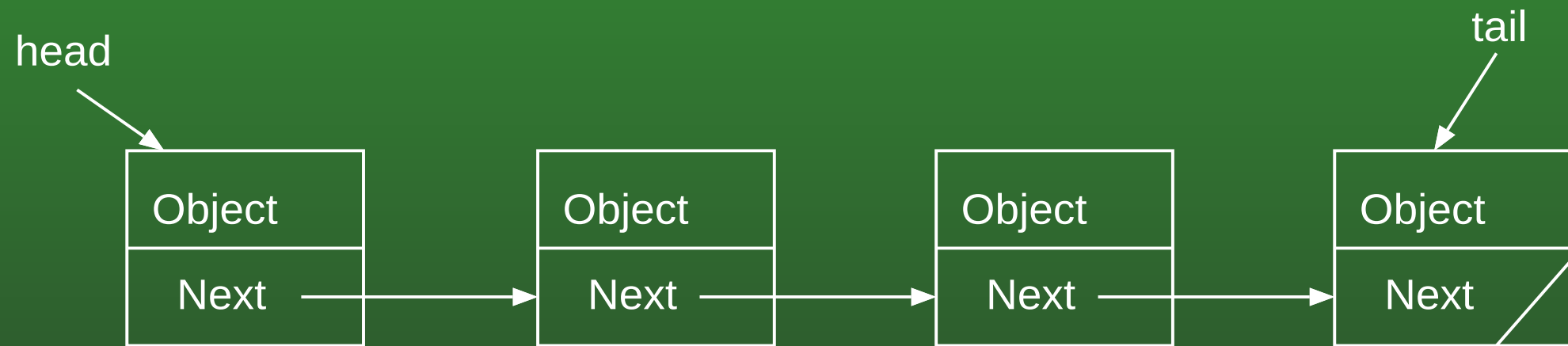
20-13: Append

```
public class LinkedList
{
    private ListNode head;
    public void append(Object obj)
    {
        ListNode last = head;
        while (last.next != null)
        {
            last = last.next;
        }
        last.next = new ListNode(obj, null);
    }
}
```

20-14: Speed up Appending

- To append an element to the end of a linked list, need to move a temp pointer to the end of the list
- How could we do this more quickly
 - We can add things to our data structure ...

20-15: Tail Pointer



20-16: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void insert(Object o) { ... }
}
```

20-17: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void insert(Object o)
    {
        head = new ListNode(o, head);
        if (tail == null)
            tail = head;
    }
}
```

20-18: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public LinkedList()
    {
        head = null;
        tail = null;
    }

    public void append(Object o) { ... }
}
```

20-19: Tail Pointer

```
public class LinkedList
{
    private ListNode head;
    private ListNode tail;

    public void append(Object o)
    {
        if (tail != null)
        {
            tail.next = new ListNode(o);
            tail = tail.next
        }
        else
        {
            head = new ListNode(o, head);
            tail = head;
        }
    }
}
```

20-20: Doubly Linked List

- Deleting from (and inserting into!) a linked list can be challenging because you need to find the node *before* the node you are looking for
- Once you've found the node, it's too late – can't follow pointers backwards to get to the previous node

20-21: Doubly Linked List

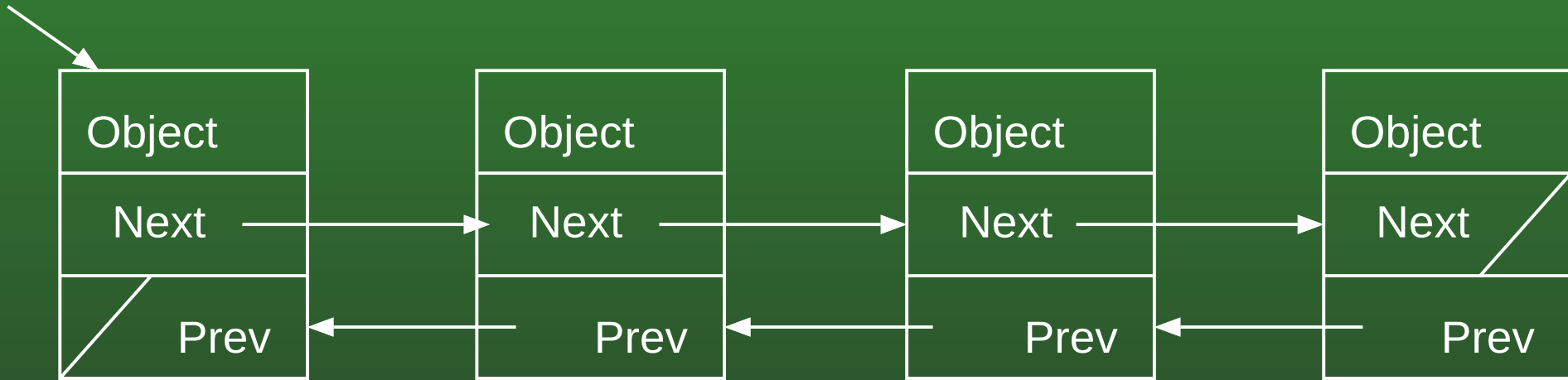
- Deleting from (and inserting into!) a linked list can be challenging because you need to find the node *before* the node you are looking for
- Once you've found the node, it's too late – can't follow pointers backwards to get to the previous node
 - ... unless you keep a pointer to the previous node in the list, too!

20-22: Doubly Linked List Node

```
public class DoubleLinkedListNode
{
    Object data;
    DoubleLinkedListNode next;
    DoubleLinkedListNode previous;
    public DoubleLinkedListNode (int data)
    {
        this.data = data;
        this.next = null;
        this.previous = null;
    }
    public DoubleLinkedListNode (int data, DoubleLinedListNode next)
    {
        this.data = data;
        this.next = next;
        this.previous = null;
    }
    public DoubleLinkedListNode (int data, DoubleLinkedListNode next,
                                DoubleLinkedListNode previous)
    {
        this.data = data;
        this.next = next;
        this.previous = previous;
    }
}
```

20-23: Doubly Linked List

head



20-24: Doubly Linked List

- Advantages
 - Don't need to be “one off” to do deletions
 - Don't need as many special cases (with some modifications)
- Disadvantages
 - Uses more space (two pointers per node)
 - Need to update both previous/next when changing list

20-25: Linked Lists

```
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public void insert(Object o)
    {
        head = new DoubleLinkedListNode(o, head);
        if (head.next != null)
        {
            head.next.previous = head;
        }
    }
}
```

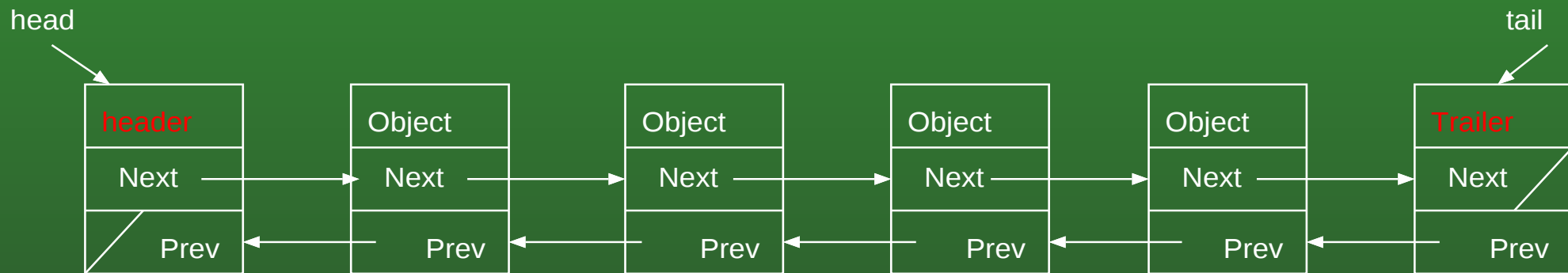
20-26: remove, Doubly Linked

```
public void remove(Object o)
{
    DoubleLinkedListNode tmp = head;
    while (tmp != null && !tmp.data.equals(o))
        tmp = tmp.next;
    if (tmp != null)
    {
        if (tmp.previous != null) // {} omitted to fit on slide!
            tmp.previous.next = tmp;
        else
            head = tmp;
        if (tmp.next != null)
        {
            tmp.next.previous = tmp;
        }
        return true;
    }
    return false;
}
```

20-27: Dummy Element II

- We still needed special cases for deleting from beginning / end of the list
- We could avoid these special cases by adding two dummy elements – to front and end of list

20-28: Doubly Linked List



20-29: Dummy Element

```
class LinkedList
{
    DoublyLinkedListNode head;
    DoublyLinkedListNode tail;

    LinkedList()
    {
        head = new ListNode(null, null, null); // Dummy header
        tail = new ListNode(null, null, head); // Dummy trailer
        head.next = tail;
    }

    // Other methods ...
}
```

20-30: Dummy Element II

- We need to change the traversal a bit
 - Don't want to compare anything to the dummy element

20-31: remove, Doubly Linked

```
public class LinkedList
{
    private DoubleLinkedListNode head;
    public void remove(Object o)
    {
        DoubleLinkedListNode tmp = head;
        while (tmp.next != null && !tmp.data.equals(o))
        {
            tmp = tmp.next;
        }
        if (tmp.next != null)
        {
            tmp.previous.next = tmp;
            tmp.next.previous = tmp;
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

20-32: **Next Project**

- Go over next project / lab on website