Doubly Linked Lists

cs206

Lec 19

void addLast(J c);
void addFirst(J c);

```
private Node<J> nextToLastNode() {
    Node<J> n = head;
    if (n.next == null)
        return null;
    while (n.next.next != null) {
        n = n.next;
    }
    return n;
}
```

Assumes that caller confirmed that head!=null

!!!!

```
public void addLast(J c) {
    Node<J> newnode = new Node<>(c);
    if (head == null) {
        head = newnode;
        return;
    }
    Node<J> n = nextToLastNode();
    if (n == null) {
        head.next = newnode;
        return;
    }
    n.next = newnode;
}
```

AddFirst

```
public void addFirst(J c) {
    Node<J> newnode = new Node<>(c);
    if (head == null) {
        head = newnode;
        return;
    }
    newnode.next = head;
    head = newnode;
}
```

Lab

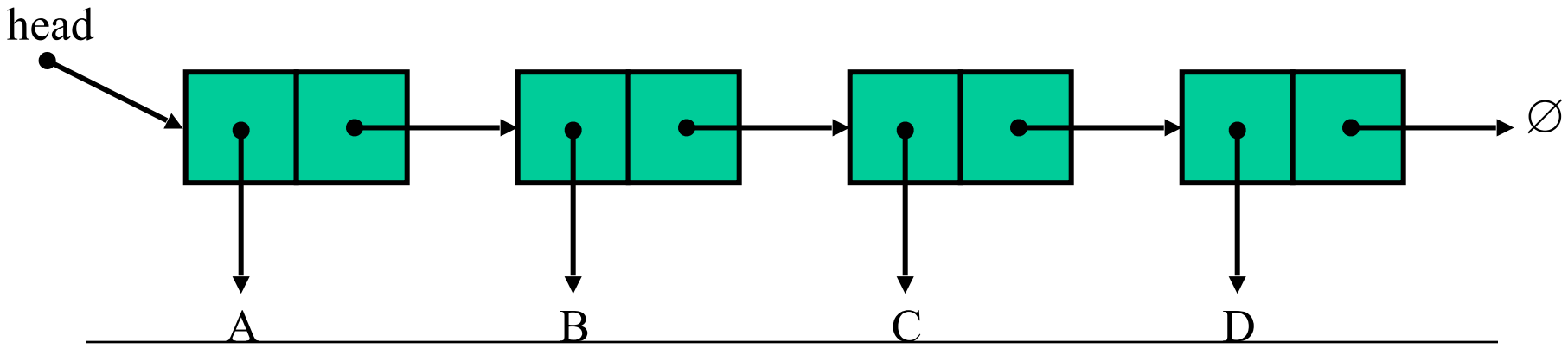
- `public boolean contains(J jj)`

- This method should search through its linked list for a node containing the object r (use ==).

```
public boolean contains(J jj) {
        Node<J> n = head;
        while (n != null) {
            if (n.data == jj)
                return true;
            n = n.next;
        }
        return false;
    }
```

removeLast()

- Problem
  - How do you remove the last
  - Problem, knowing the last node is not enough?
    - To remove D we need to do things to C
    - Cannot go backwards!!
- So, need to search forward in list to find the node before the last node
- Happily, I already had a function to do that



Remove Last

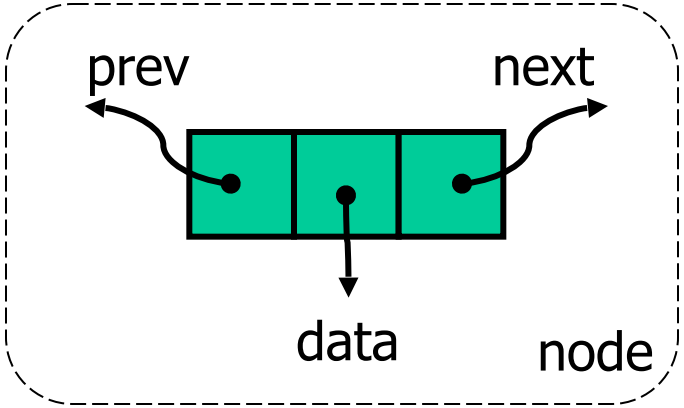
```
public J removeLast() {
    if (head == null)
        return null;
    if (head.next == null) {
        J tmp = head.data;
        head = null;
        return tmp;
    }
    Node<J> n2last = nextToLastNode();
    J tmp = n2last.next.data;
    n2last.next = null;
    return tmp;
}
```

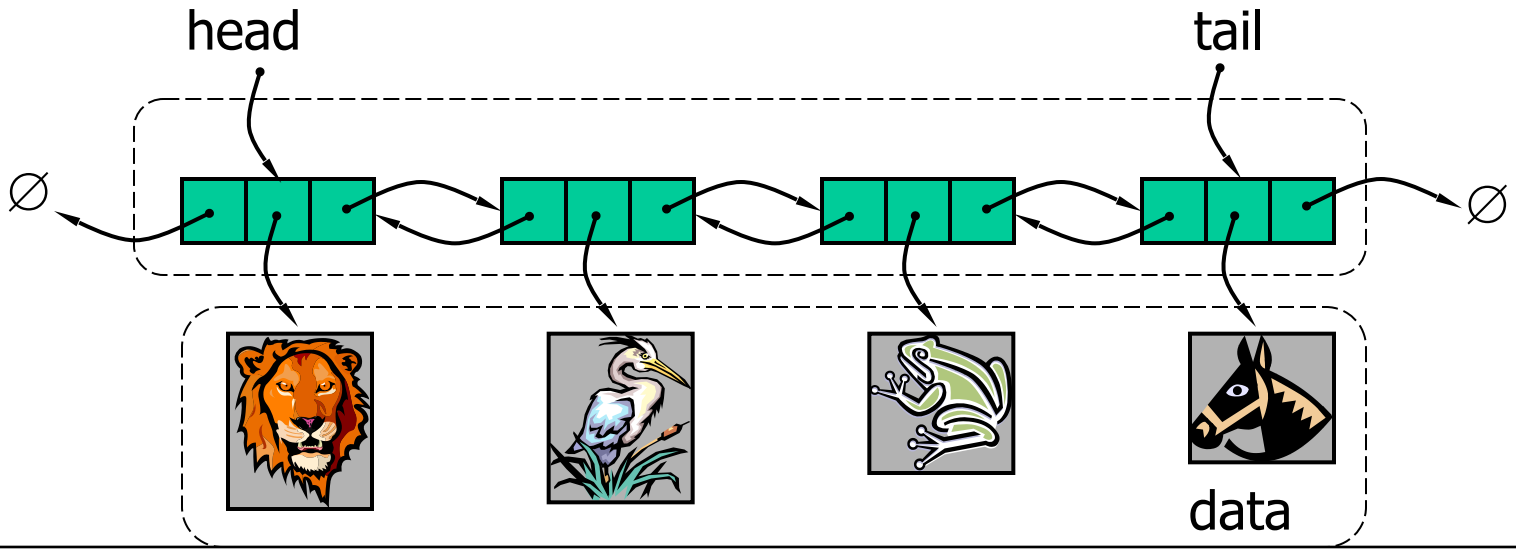
Musings on singly linked lists

- The whole remove last method is a pain
- Not knowing how to go backward is a pain
  - Linked lists are a pain
- Can’t do anything about linked lists be ing a pain

Doubly Linked List

- Can be traversed forward and backward
- Nodes store an extra reference





Double Linked List interface

```
public interface LinkedListInterfaceComp<E extends Comparable<E>> {
    int size();
    boolean isEmpty();
    Comparable<E> first();
    Comparable<E> last();
    void addLast(Comparable<E> c);
    void addFirst(Comparable<E> c);
    Comparable<E> removeFirst();
    Comparable<E> removeLast();
    Comparable<E> remove(Comparable<E> r);
    boolean contains(Comparable<E> iD);
}
```

This could also be applied to a single linked list (or an array list) or …

Node & DLL start

```
public class DoubleLinkedList<T extends Comparable<T>> implements
LinkedListInterfaceComp<T> {
    protected class Node<V extends Comparable<V>> {
       public Comparable<V> data;
       public Node<V> next;
       public Node<V> prev;
       public Node(Comparable<V> data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }
   private Node<T> head = null;
   private Node<T> tail = null;
   private int size = 0;
```

Basics

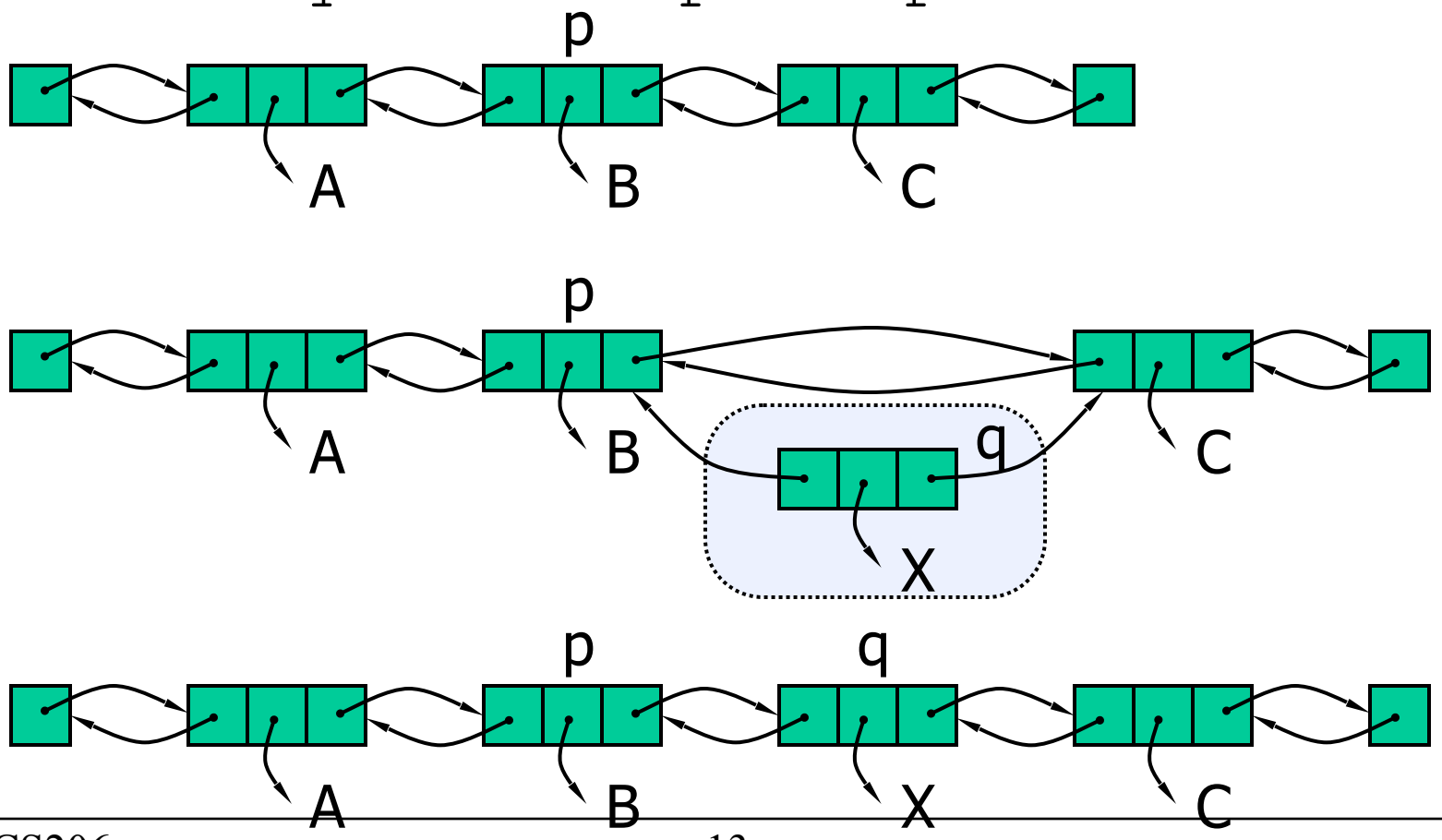
```
@Override
public int size() {
     return size;
 }
@Override
 public boolean isEmpty() {
     return size == 0;
 }

@Override
    public Comparable<T> first() {
     if (head == null)
         return null;
     return head.data;
 }
@Override
 public Comparable<T> last() {
     if (head == null)
         return null;
     return tail.data;
 }
```

Insertion: AddFirst, AddLast

Add Between

- Insert `q` between `p` and `p.next`



Add Between

```
public void addBtw(T c, Node prev, Node next) {
    Node newest = new Node(c);
    prev.next = newest;
    next.prev = newest;
     newest.prev = prev;
     newest.next = next;
    size++;
}
```

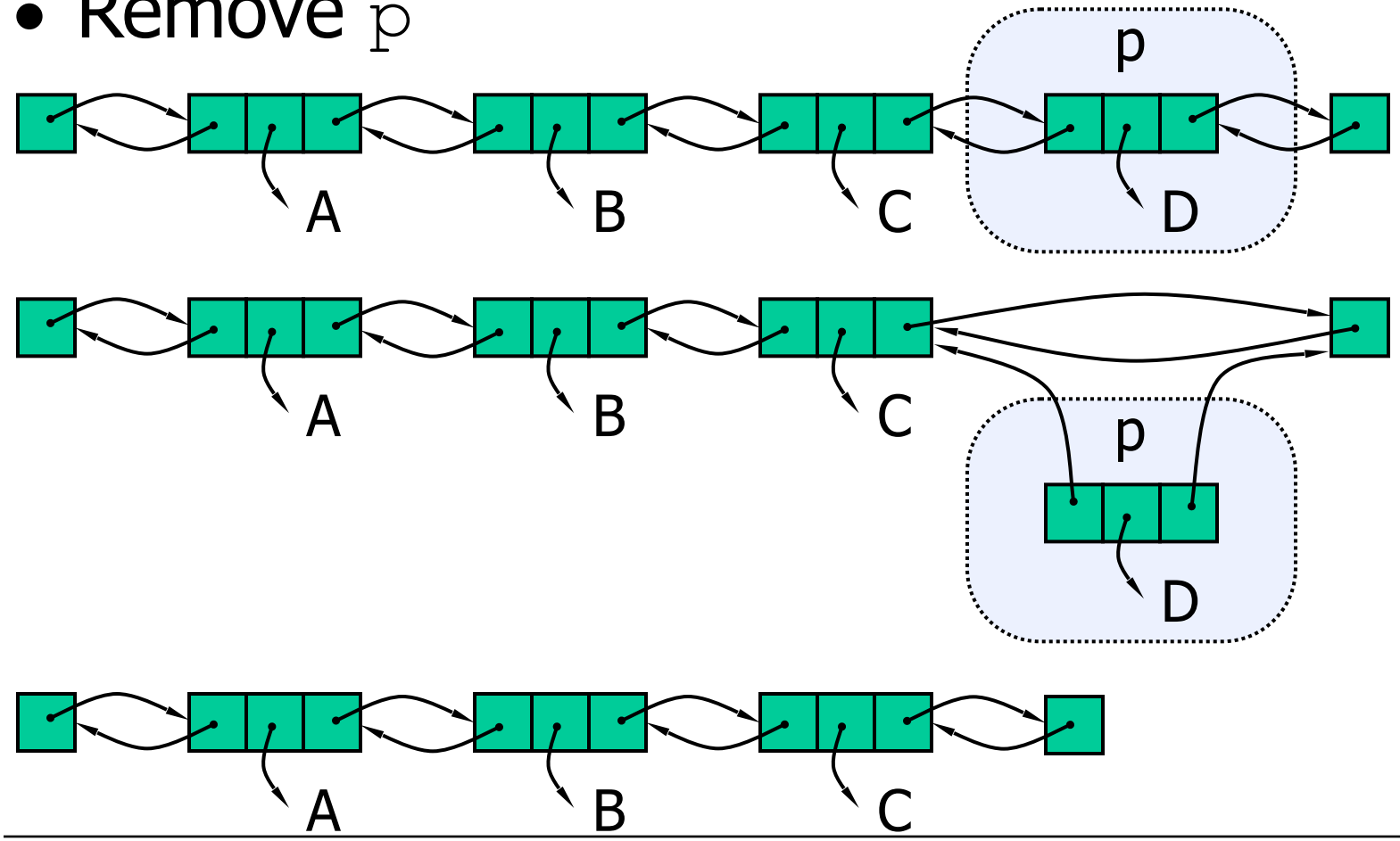
Problems??

Deletion — first element

```
public Comparable<T> removeLast() {
        if (head == null)
            return null;
        Comparable<T> rtn = tail.data;
        if (head == tail) {
            head = null;
            size = 0;
            tail = null;
            return (T) rtn;
        }
        tail = tail.prev;
        tail.next = null;
        size--;
        return rtn;
    }
```

Deletion

- Remove `p`



Deletion and contains

```
    Comparable<E> remove(Comparable<E> r);
    boolean contains(Comparable<E> iD);
```

First write a private utility function

```
    private Node<T> find(Comparable<E> look4)
```

that returns a node containing look4 (or null)
then use this function in remove and contains

Sorted Linked Lists

```
public class SortedDLL<T extends Comparable<T>> extends
DoubleLinkedList<T> {
     public void addSorted(Comparable<T> t) {
      // lots of thought here
      // feels a lot like addBetween except you have to figure
      // out where
      }
```

- If you have a sorted linked list, should you have
  - addFirst
  - addLast
  - removeFirst
  - removeLast