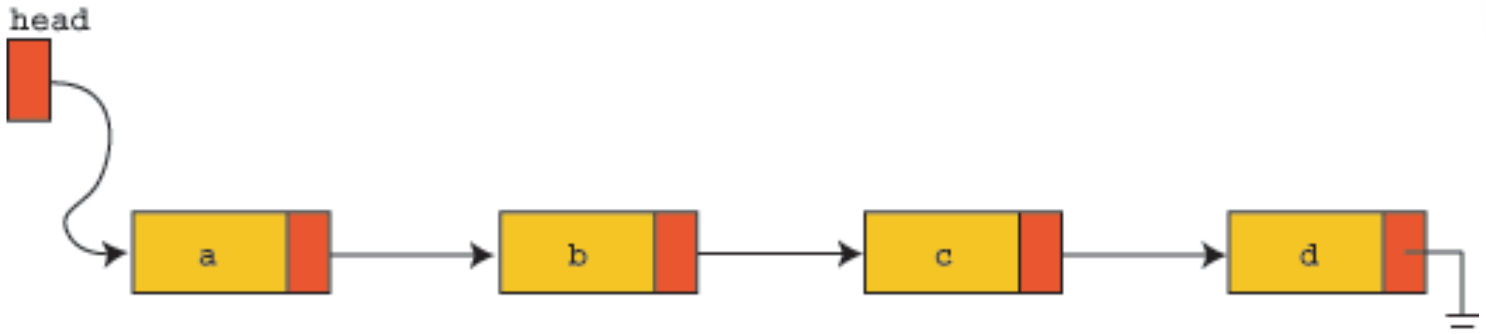
Doubly Linked Lists

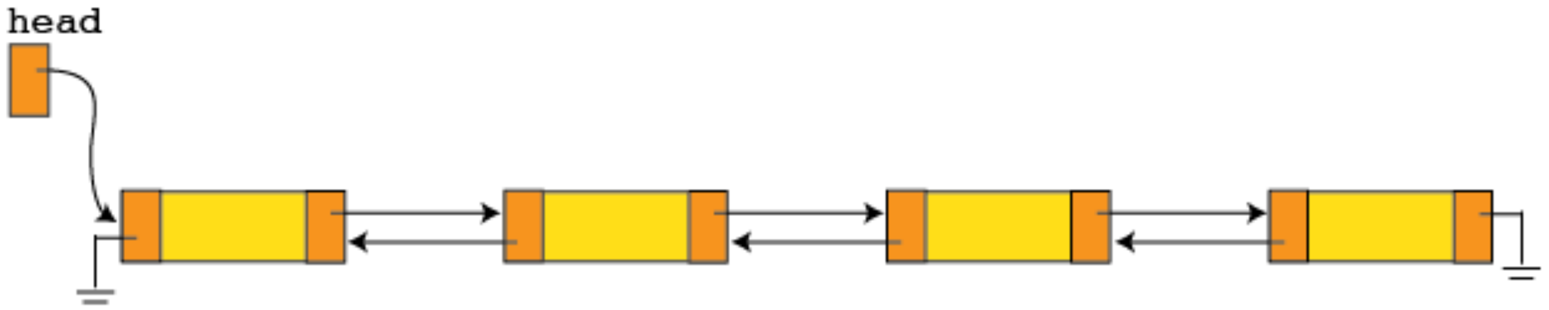
Double-ended nodes, adding to the front, removing an element, testing, list variations

What is a doubly linked list?

Linked lists we have discussed so far are **singly linked**. You can travel the list in only one direction.



With a **doubly linked list**, you can travel two directions: forward and backward.



Modify the Node class

The inner Node class for a doubly linked list has three fields, one for the data and one for each pointer.

```
private class Node{
  E data;
  Node previous;
  Node next;

  public Node(E data, Node previous, Node next);
    this.data = data;
    this.previous = previous;
    this.next = next;
}
```

Working with previous

The outer class for a doubly linked list starts out the same as for a singly linked list. But note the extra work for addToFront().

```
public class DoubleLinkedList<E> {
   private Node head;

   public DoubleLinkedList() {
      head = null;
   }

   public void addToFront(E data) {
      Node baby = new Node(data, null, head);
      if (head != null)
         head.previous = baby;
      head = baby;
   }

   // Other standard list operations go here
   // Node declaration goes here
}
```

A private removeNode( )

Private methods are nice for code that might be repeated.

```
private void removeNode(Node p) {
   if (p != null) {
      if (p == head)
         head = head.next;
      else
         p.previous.next = p.next;
      if (p.next != null)
         p.next.previous = p.previous;
   }
}
```

A public removeFirst( )

To remove the first occurrence of an item:

1. Look for the item, starting from the front of the list.
2. Remove the first Node you find where the item resides.

```
public void removeFirst(E data) {
   Node c = head;
   // Look for the first match
   while (c != null && !c.data.equals(data))
      c = c.next;
   if (c != null)
      removeNode(c);
}
```

A public removeLast( )

To remove the last occurrence of an item in the list:

1. Travel to the end of the list.
2. Look for the item, traveling backwards toward the front of the list.
3. Remove the first Node you find where the item resides.

```
public void removeLast(E data) {
   // Travel to the end of the list
   Node c = head;
   while (c != null && c.next != null)
      c = c.next;
   // Travel toward the front now
   while (c != null && !c.data.equals(data))
      c = c.previous;
   if (c != null)
      removeNode(c);
}
```

Linked list variations

There are many variations on implementations of linked lists:

- Simple singly linked lists and doubly linked lists.
- Lists that keep a count of their sizes.
- Lists that keep a pointer to the first element and another pointer to the last element.
- Circular lists: keep a pointer to the last element only, but the last element has a link to the first one.
- Lists with dummy head nodes. No need to check to see if head is null.