159 XTENDED

A WEEKLY REVIEW

LINKED LISTS

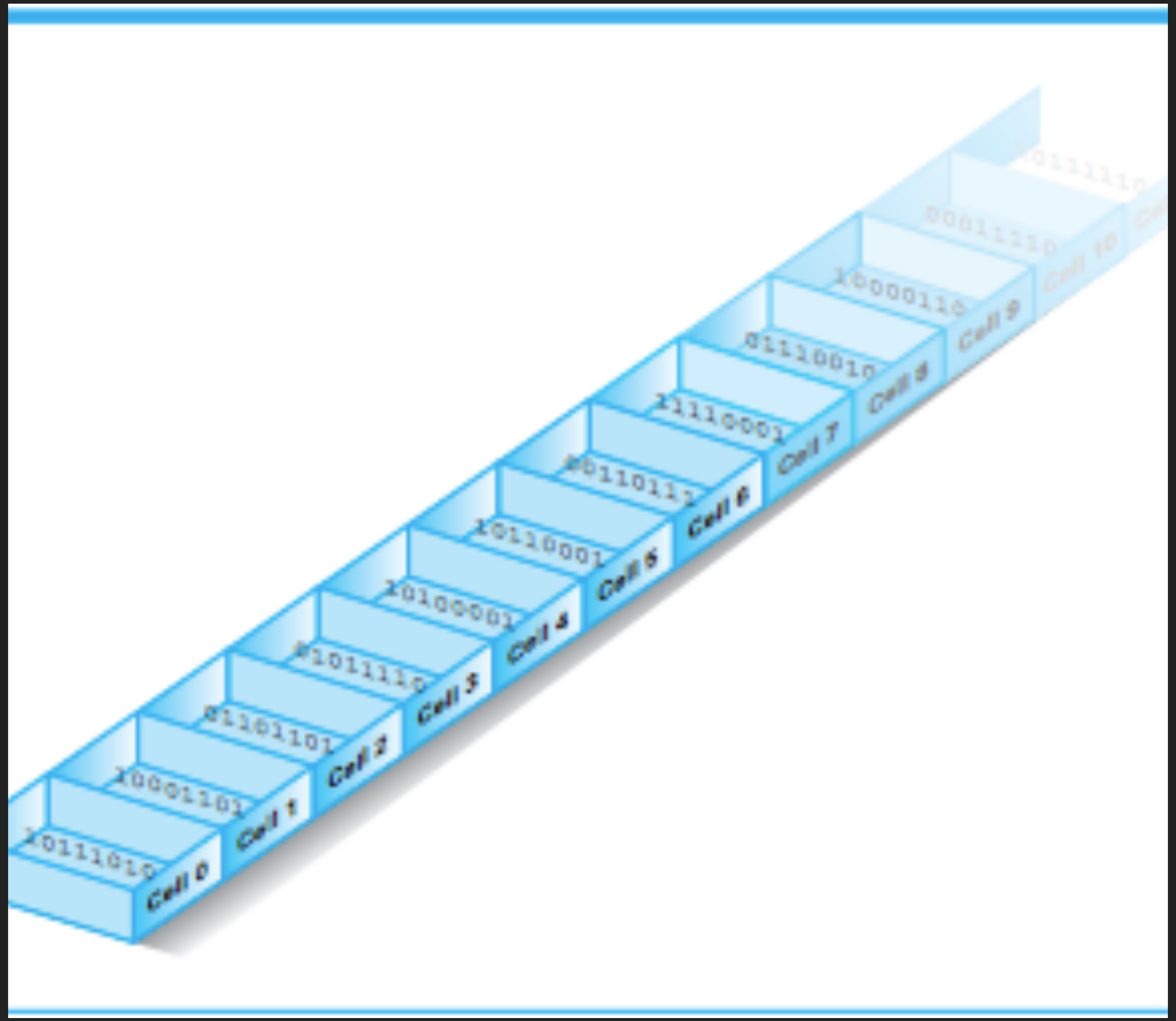
Pull up **Socrative.com**

and join room **XTEND159**

A REVIEW OF MEMORY

- A computer's memory is a sequence of bytes.
- Each byte has a unique memory address that identifies it.
- We can access and modify the individual bytes of a computer's memory in any order.
- You could think of memory as an array of bytes.

How does memory work?
See: CS 261, CS 456



REMOVING AN ELEMENT FROM AN ARRAY

1. Elements in an array are stored sequentially in memory.
2. Removing an element from the middle of a list leaves an empty space.
3. If we want to keep our array ordered, we have to individually move each element after our removed element backwards.

5	10	15	20

5	null	15	20

5	15	null	20

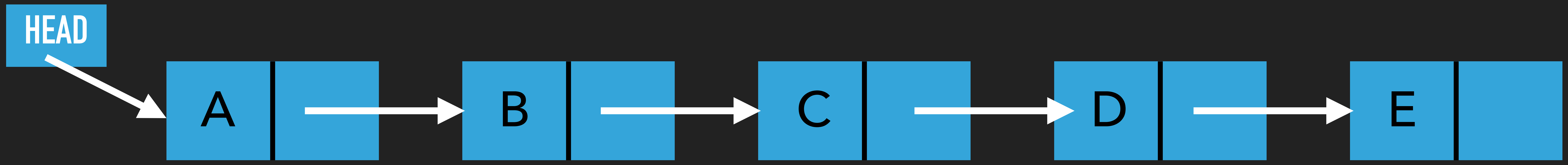
5	15	20	null

IF AN ARRAY IS 100 ELEMENTS LONG, AND WE REMOVE THE FIRST ELEMENT, HOW MANY MOVES DO WE HAVE TO MAKE TO KEEP THE ARRAY ORDERED?

Pull up Socrative.com
and join room **XTEND159**

SINGLY LINKED LISTS

- A singly linked list is a data structure that consists of a sequence of nodes.
- This sequence of nodes starts at the head node.
- Each node stores . . .
  - An element
  - A link to the next node in the list



THE NODE CLASS

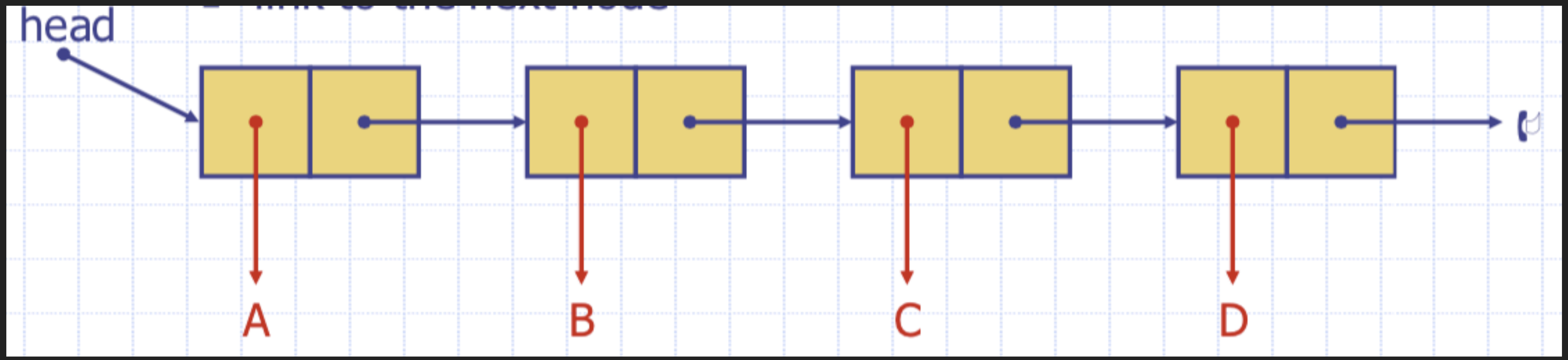
```
public class Node<E>
{
    private E element;
    private Node<E> next;

    public Node(E element, Node<E> next)
    {
        this.element = element;
        this.next = next;
    }

    // Getters and Setters for
    // element and next
}
```

- A node contains an element and a link to the next element in the list.
- Since nodes are reference types, note that the next variable is holding the address of the next node in memory.

WHAT SHOULD A SINGLY LINKED LIST CONTAIN?

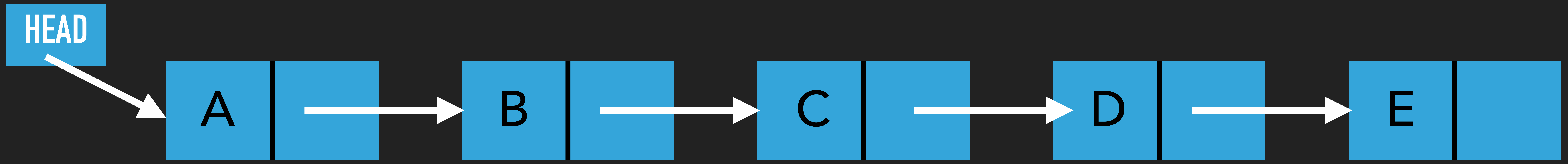
- A head node and a tail node (pointers to front and back of list)
- Getters and setters for the head and tail
- Getter method for an element at any arbitrary index in the list
- Methods for adding and removing elements from the list
- Generally, the class should follow the List abstract data type.

WHAT STEPS WOULD YOU TAKE TO ADD A NEW NODE TO THE FRONT OF A SINGLY LINKED LIST?

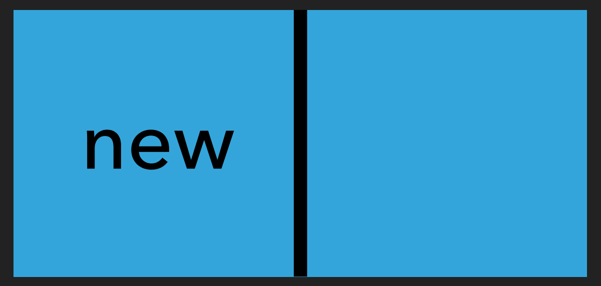


Pull up Socrative.com
and join room **XTEND159**

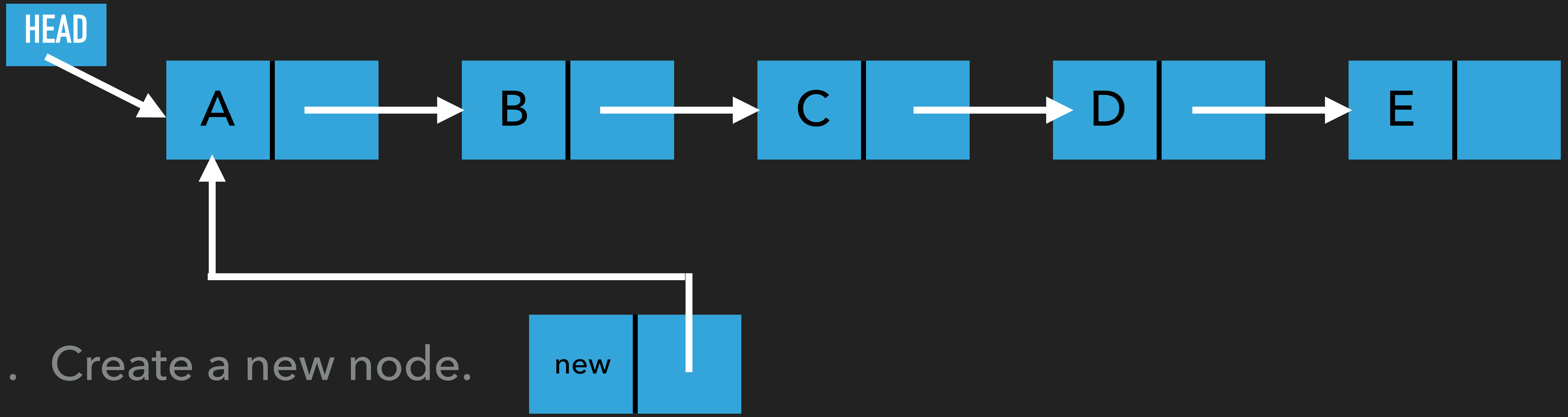
ADDING A NODE TO THE FRONT



1. Create a new node.

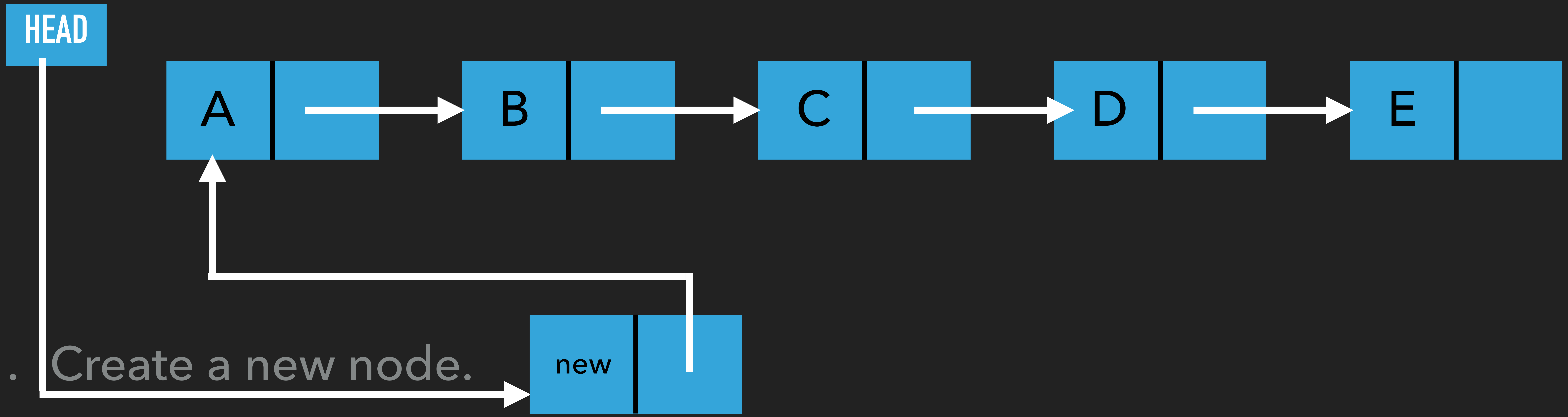


ADDING A NODE TO THE FRONT



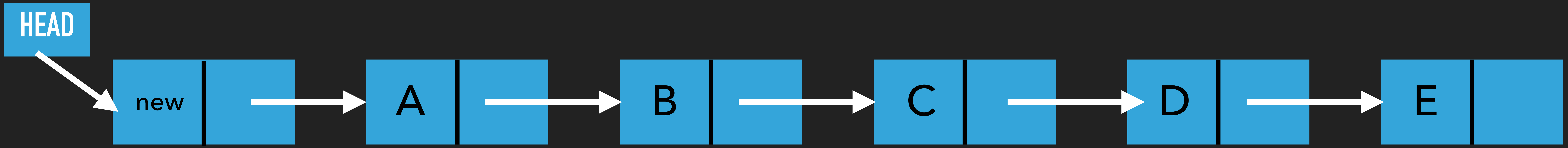
1. Create a new node.
2. Set the link in the new node to the current head.

ADDING A NODE TO THE FRONT



1. Create a new node.
2. Set the link in the new node to the current head.
3. Set the head link to the newly created node.

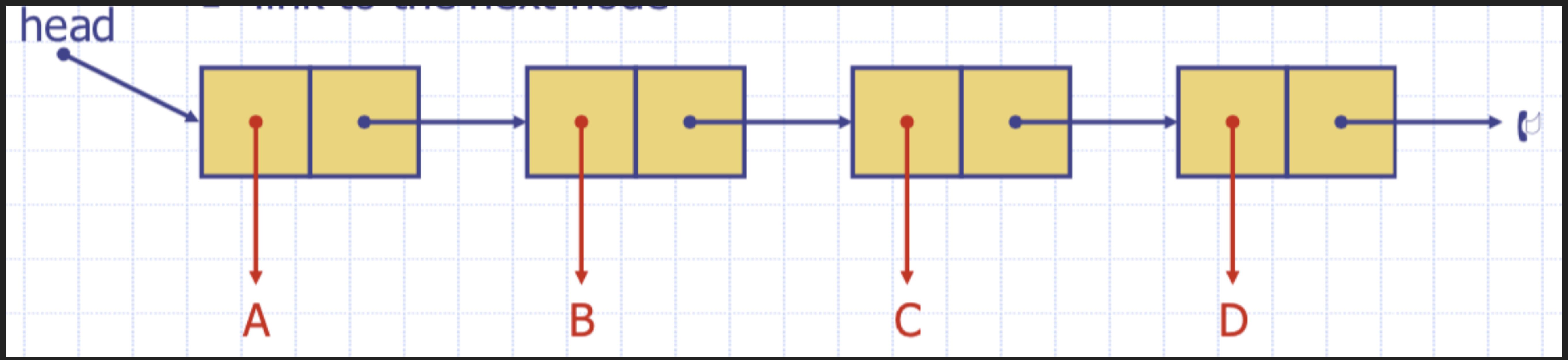
ADDING A NODE TO THE FRONT



1. Create a new node.
2. Set the link in the new node to the current head.
3. Set the head link to the newly created node.

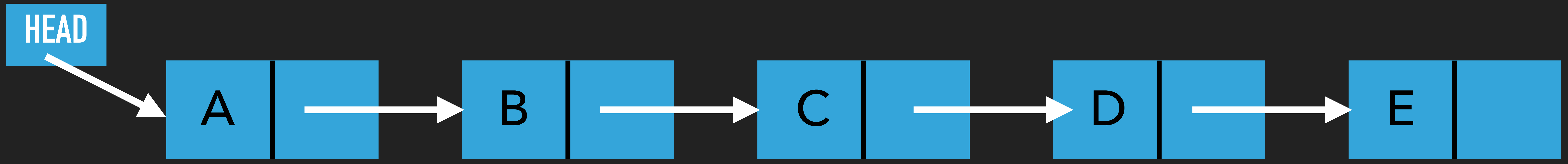
We have successfully added a new node to the front of the list!

WHAT STEPS WOULD YOU TAKE TO REMOVE NODE B FROM THIS SINGLY LINKED LIST?



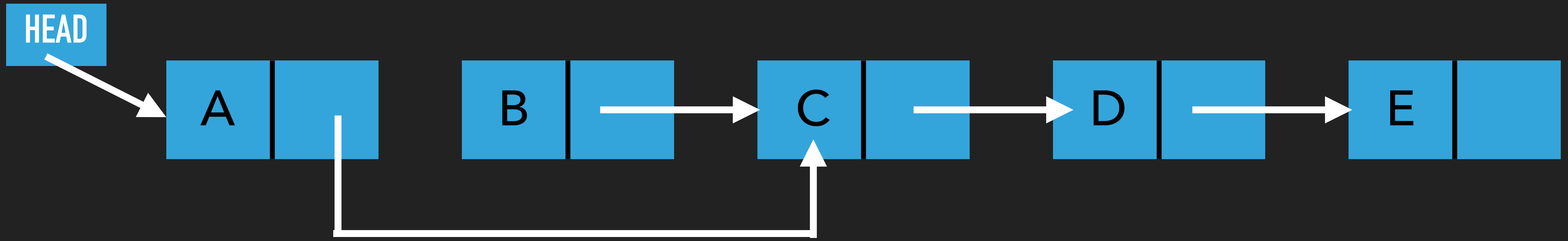
Pull up Socrative.com and join room **XTEND159**

REMOVING A NODE FROM THE LIST



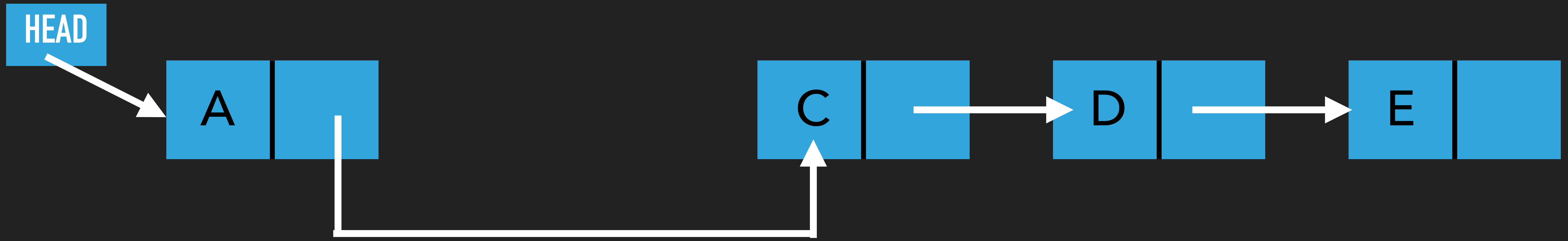
1. Set the link of the node before the one you want to delete to the node that is after the one you want to delete.

REMOVING A NODE FROM THE LIST



1. Set the link of the node before the one you want to delete to the node that is after the one you want to delete.

REMOVING A NODE FROM THE LIST



1. Set the link of the node before the one you want to delete to the node that is after the one you want to delete.

2. Eventually, Java's garbage collector will come and delete node B.

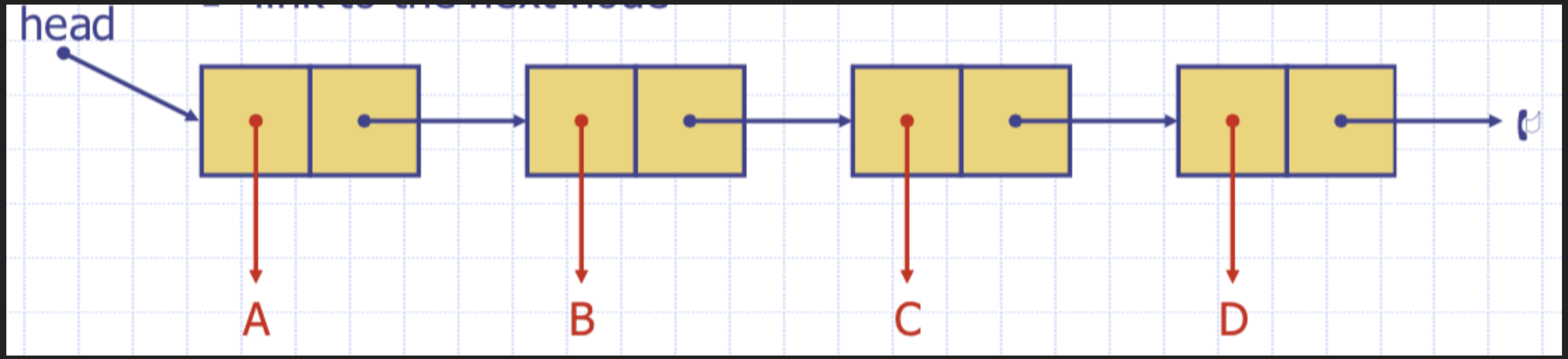
The memory that was used for node B is automatically "deallocated" by Java's garbage collector. For more on memory management, see CS 261.

WHY WOULD WE USE A LINKED LIST OVER AN ARRAY?

- Elements can be added or removed from a link list without reorganization of the entire list.
- Nodes in a link list do **not** need to be stored contiguously in memory. They can be stored anywhere.
- Removing an element from an array with 100 elements may take as many as 100 operations. With a linked list, there are only a few moves that have to be made.
- This makes adding and removing from a linked list less computationally expensive than adding or removing from an array.

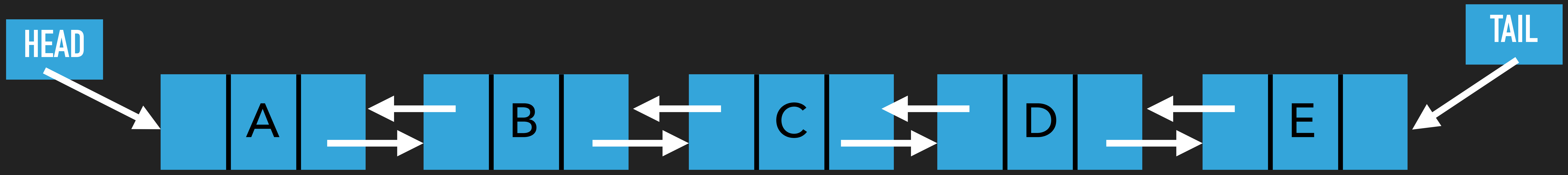
For more on algorithmic efficiency, see CS 240 and CS 452.

HOW WOULD YOU REMOVE AN ELEMENT FROM THE TAIL OF A SINGLY LINKED LIST?



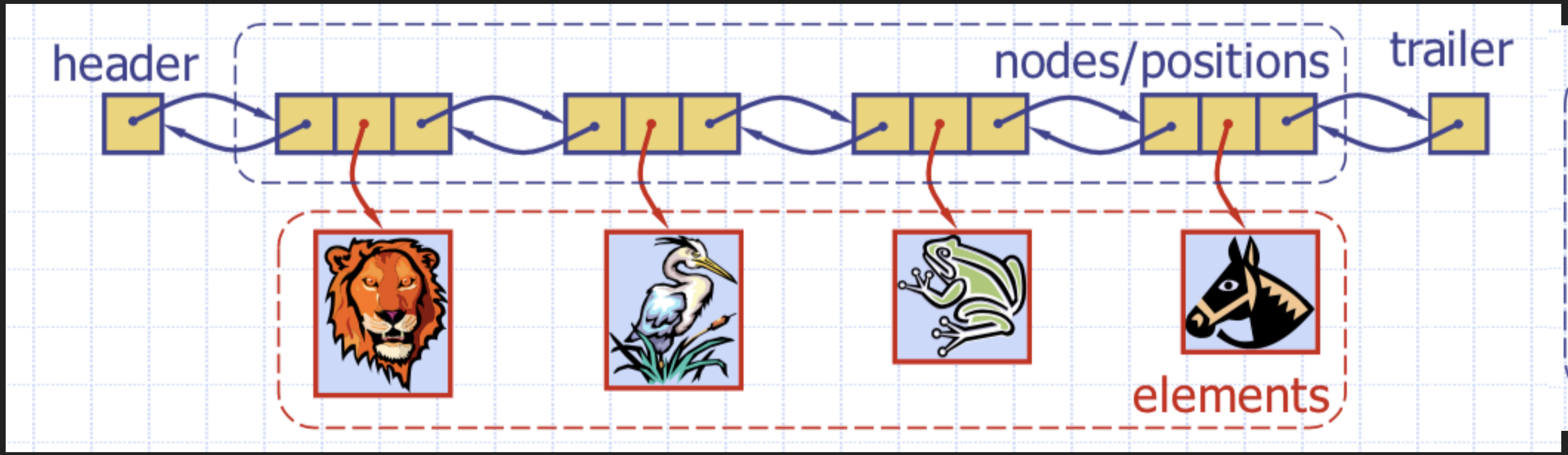
Pull up Socrative.com and join room **XTEND159**

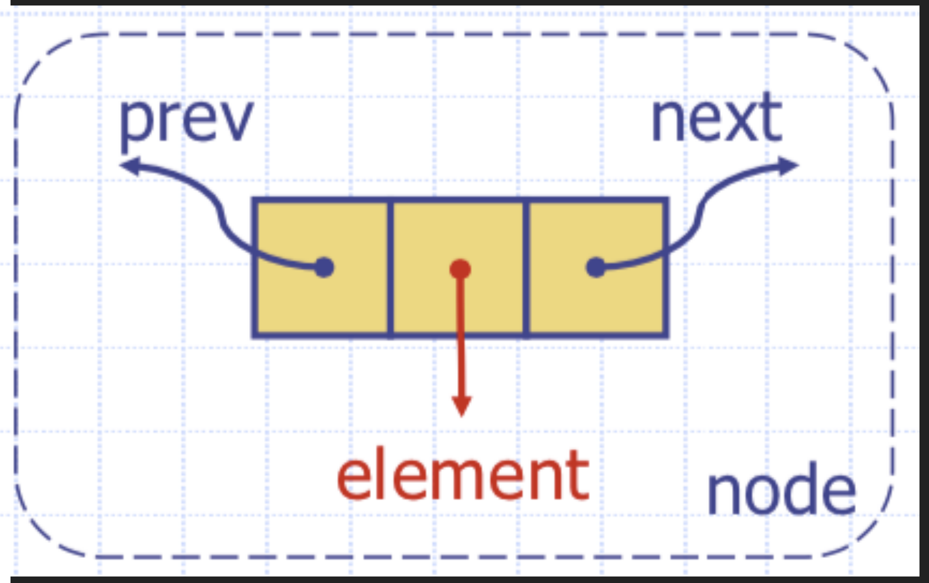
DOUBLY LINKED LISTS



- In previous examples, our list can only be traversed forwards.
- By adding a previous pointer in addition to our next pointer, our list can be traversed both forwards and backwards.
- All that we have to do is add a previous link to each node.

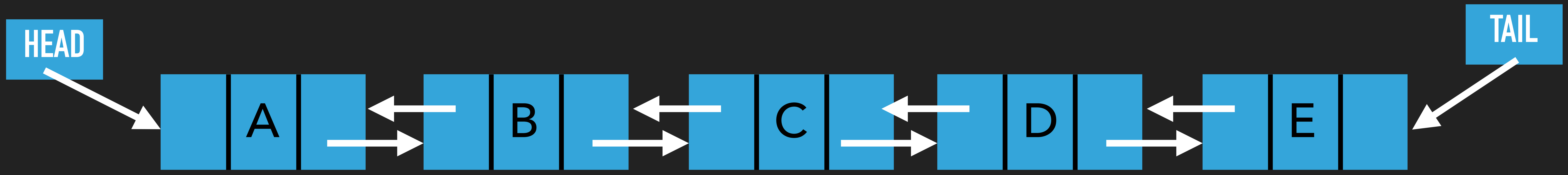
HOW DO WE REMOVE AN ELEMENT FROM THE TAIL OF A DOUBLY LINKED LIST?





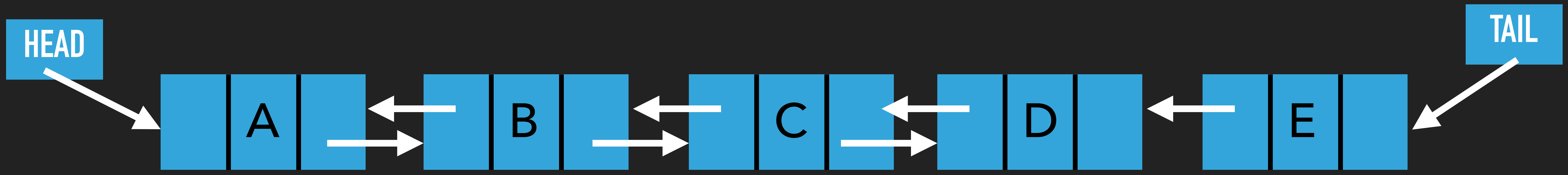
Pull up Socrative.com
and join room **XTEND159**

REMOVING AN ELEMENT FROM THE TAIL OF A DOUBLY LINKED LIST



1. Follow the tail pointer to the last node in the list, and follow the previous pointer in the last node to the second to last node in the list.

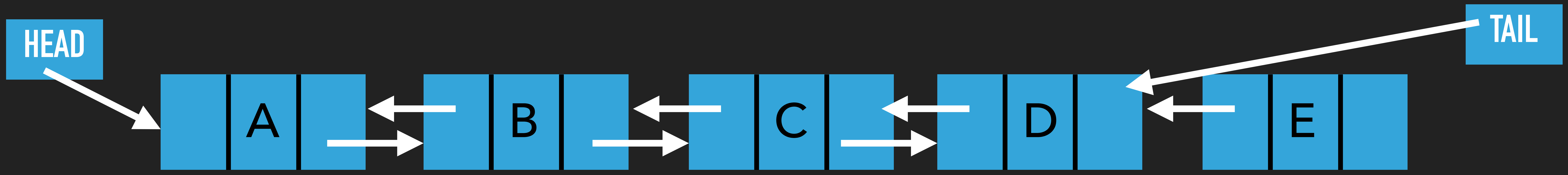
REMOVING AN ELEMENT FROM THE TAIL OF A DOUBLY LINKED LIST



1. Follow the tail pointer to the last node in the list, and follow the previous pointer in the last node to the second to last node in the list.

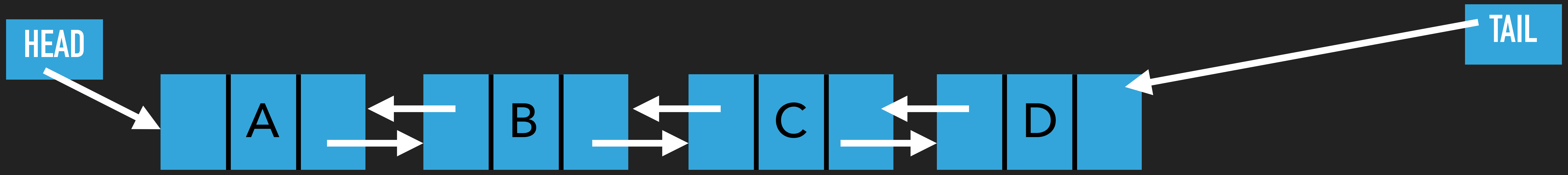
2. Make the next pointer in this node null. (Note: D no longer has a next node)

REMOVING AN ELEMENT FROM THE TAIL OF A DOUBLY LINKED LIST



1. Follow the tail pointer to the last node in the list, and follow the previous pointer in the last node to the second to last node in the list.
2. Make the next pointer in this node null. (Note: D no longer has a next node)
3. Set the tail pointer to the second to last element.

REMOVING AN ELEMENT FROM THE TAIL OF A DOUBLY LINKED LIST



1. Follow the tail pointer to the last node in the list, and follow the previous pointer in the last node to the second to last node in the list.
2. Make the next pointer in this node null. (Note: D no longer has a next node)
3. Set the tail pointer to the second to last element.
4. Eventually, Java's garbage collector will delete the last node.