

LEC 06

CSE 123

LinkedList

Questions during Class?

Raise hand or send here

sli.do #cse123



BEFORE WE START

Talk to your neighbors:

What's your favorite form of potato?

Instructor: James Wilcox

Announcements

- Great job on Quiz 0!!
- R1 and P0 feedback released!
- Creative Project 1 due tonight at 11:59pm
 - Submit *something* so we can provide some feedback!
- Programming Project 1 releases tomorrow
 - One of the trickier assignments in the course
 - 2 weeks to complete this one! Feel free to take a breather if necessary but get started sooner than later

LinkedList

Reminder: Implementing Data Structures

- No different from designing any other class!

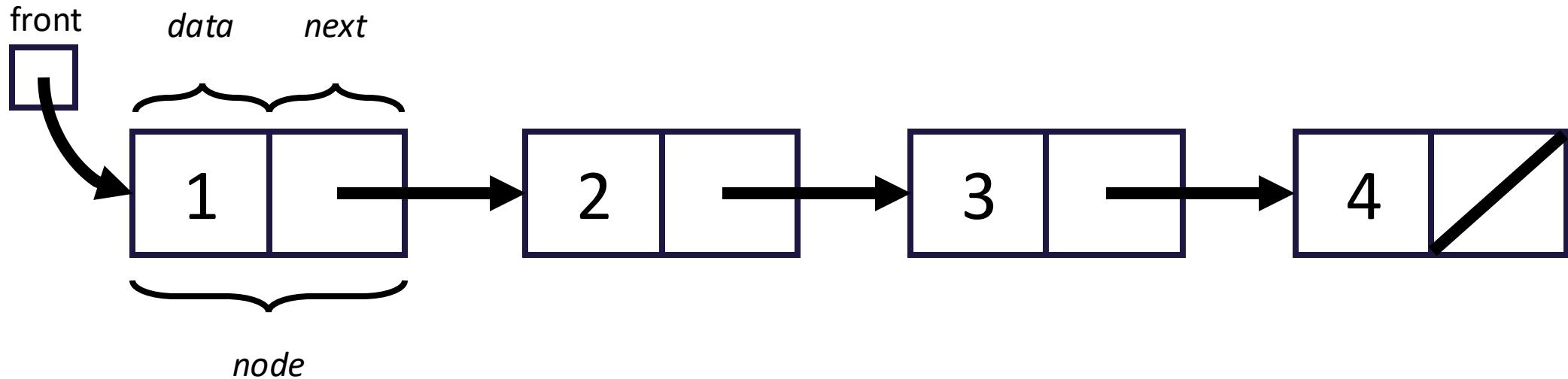
- Specified behavior (List interface):

Method	Description
<code>add(E value)</code>	Adds the given value to the end of the list
<code>add(int index, E value)</code>	Adds the given value at the given index
<code>remove(E value)</code>	Removes the given value if it exists
<code>remove(int index)</code>	Removes the value at the given index
<code>get(int index)</code>	Returns the value at the given index
<code>set(int index, int value)</code>	Updates the value at the given index to the one given
<code>size()</code>	Returns the number of elements in the list

- Choose appropriate fields based on behavior
- Just requires some thinking outside the box

LinkedList

- Goal: leverage non-contiguous memory usage
 - How? LinkedNodes!
- What field(s) do we need to keep track of?
 - `ListNode front; // First node in the chain`



LinkedList cont.

- Now that we have a `LinkedList` class, will a client ever need to interact with a `ListNode`?
 - No! Not something they should have to worry about
- How can we abstract `ListNodes` away from them?
 - Leaving them in a public file is pretty obvious...
- What if we made `ListNode` a private class inside `LinkedList`?
 - We can still access it (just like private fields)
 - Clients won't even know the class exists!
- Do fields need to be private if the entire class is private?

Reminder: Iterating over ListNodes

- General pattern iteration code will follow:

```
ListNode curr = front;  
while (curr != null) {  
    // Do something  
  
    curr = curr.next;  
}
```

Why do we need a ListNode curr?

Why curr?

```
public static void main(String[] args) {  
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));  
}
```

```
public static void printList(ListNode front) {  
    while (front != null) {  
        System.out.print(front.data + " ");  
        front = front.next;  
    }  
    System.out.println();  
}
```

The diagram illustrates the state of a singly linked list and the execution flow between the `main()` and `printList()` methods.

The linked list consists of three nodes, each represented as a rectangle divided into two parts: data and next. The nodes contain the values 1, 2, and 3 respectively. The `next` pointer of the third node is marked with a diagonal line, indicating it is null.

Execution flow is shown by arrows:

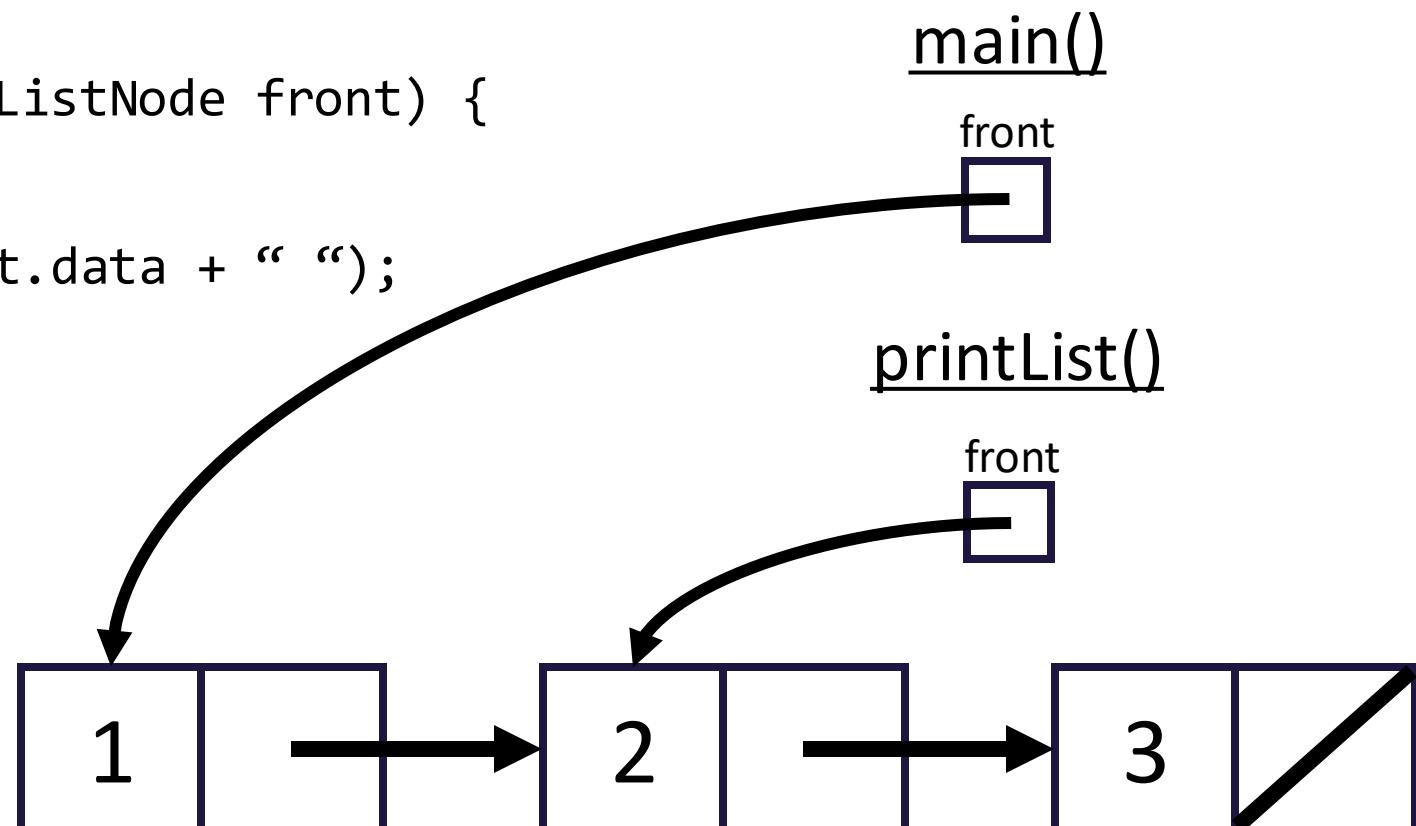
- A curved arrow originates from the `front` variable in the `main()` method and points to the first node (value 1).
- A curved arrow originates from the `front` variable in the `printList()` method and points to the same first node (value 1).
- A straight horizontal arrow points from the `next` pointer of the first node to the second node (value 2).
- A straight horizontal arrow points from the `next` pointer of the second node to the third node (value 3).
- An arrow points from the `System.out.println()` call back to the `front` variable in the `printList()` method, indicating the end of the current iteration.



Why curr?

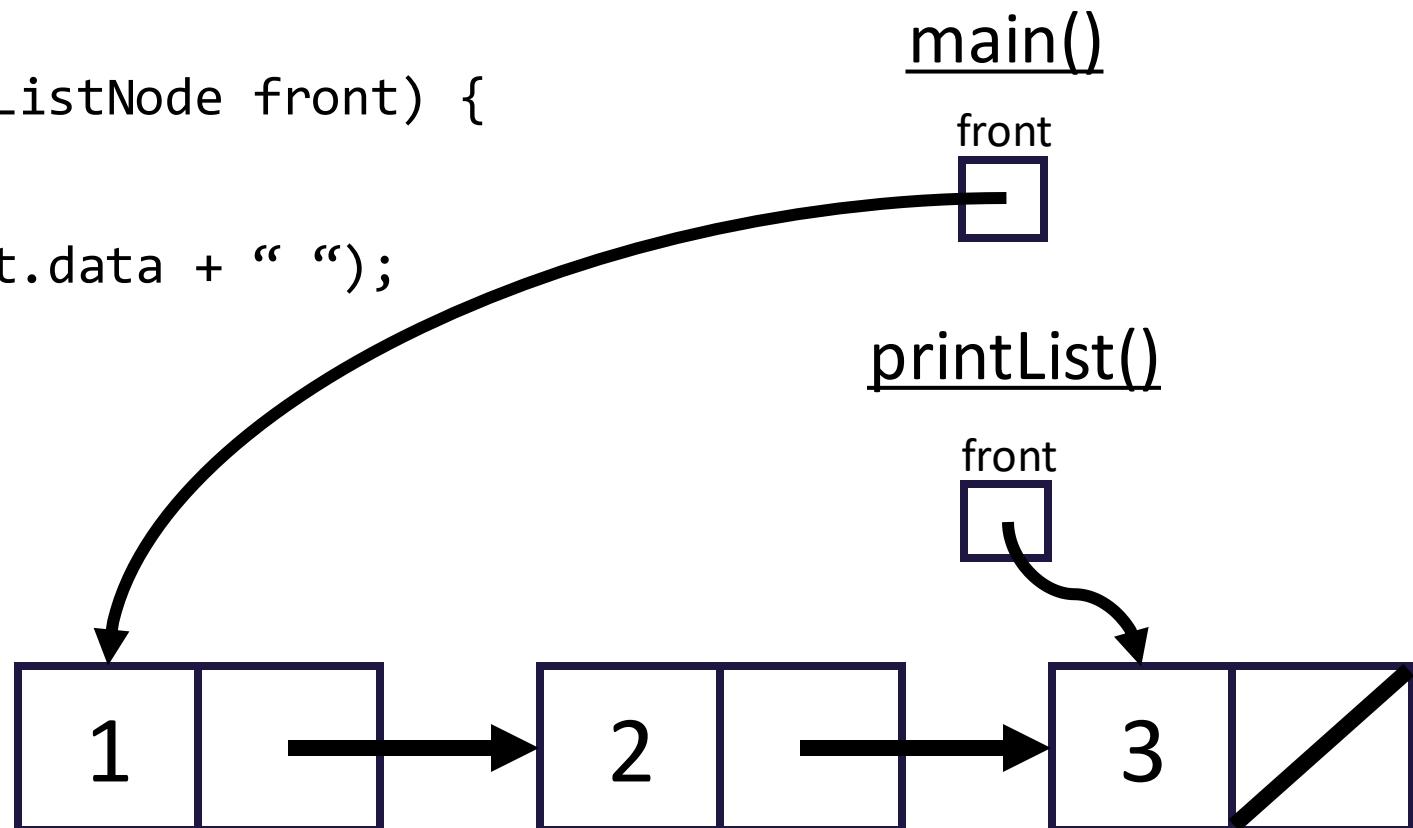
```
public static void main(String[] args) {  
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));  
}
```

```
public static void printList(ListNode front) {  
    while (front != null) {  
        System.out.print(front.data + " ");  
        front = front.next;  
    }  
    System.out.println();  
}
```



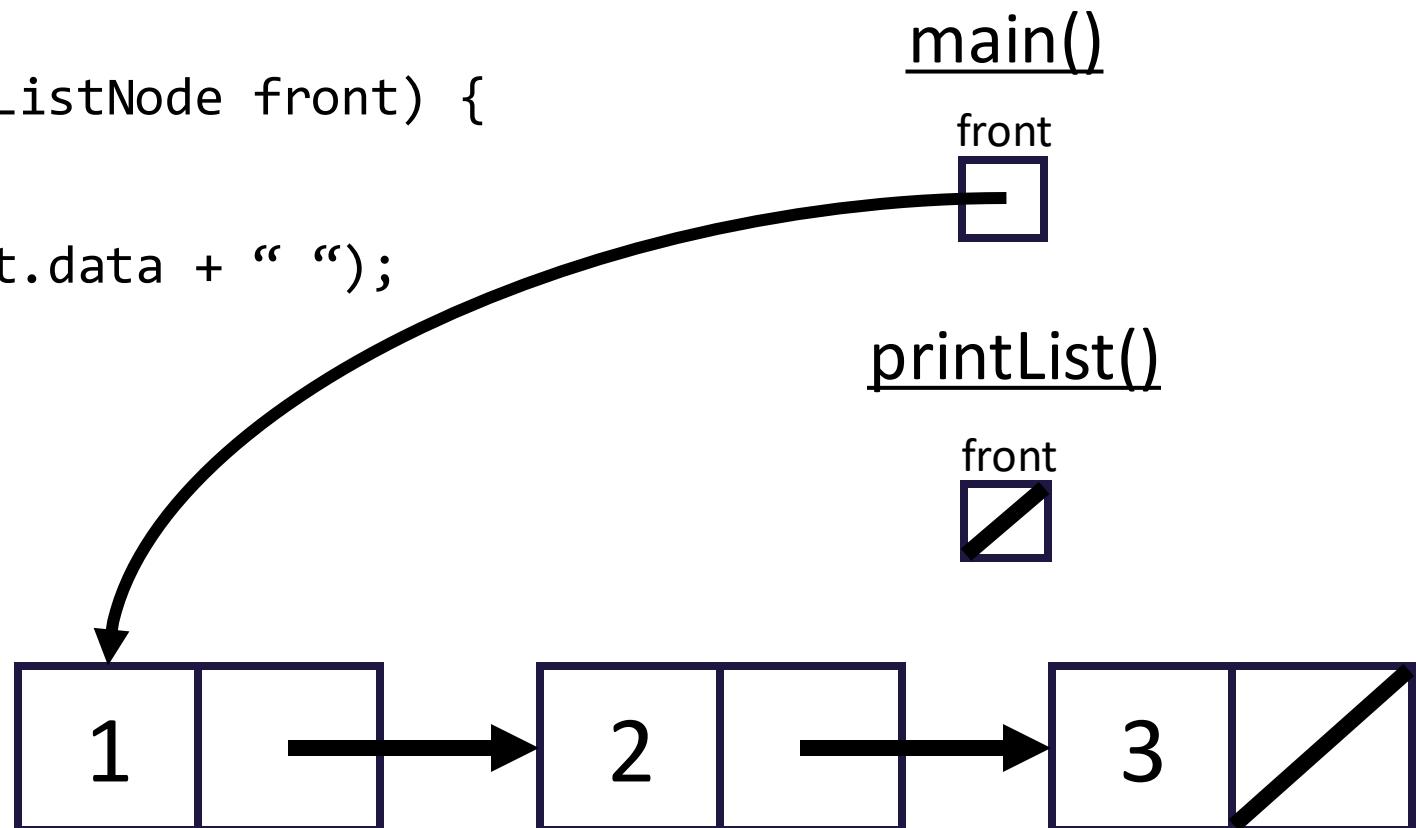
Why curr?

```
public static void main(String[] args) {  
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));  
}  
  
public static void printList(ListNode front) {  
    while (front != null) {  
        System.out.print(front.data + " ");  
        front = front.next;  
    }  
    System.out.println();  
}
```



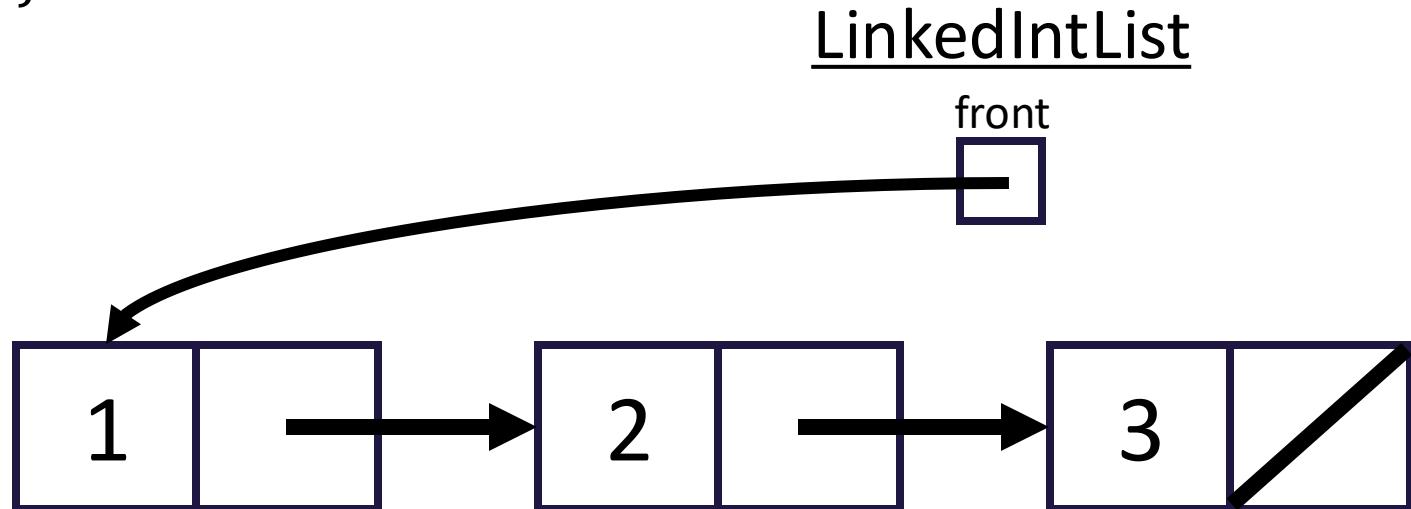
Why curr?

```
public static void main(String[] args) {  
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));  
}  
  
public static void printList(ListNode front) {  
    while (front != null) {  
        System.out.print(front.data + " ");  
        front = front.next;  
    }  
    System.out.println();  
}
```



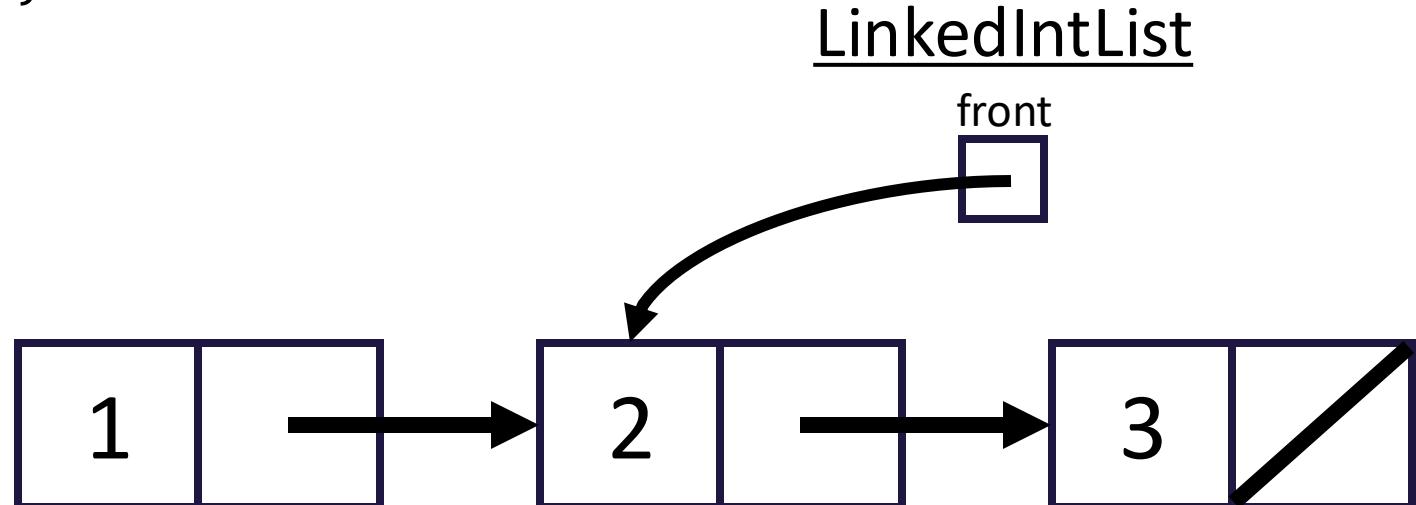
Why curr?

```
public class LinkedList {  
    private ListNode front;  
  
    public void printList() {  
        while (front != null) {  
            System.out.print(front.data + " ");  
            front = front.next;  
        }  
    }  
}
```



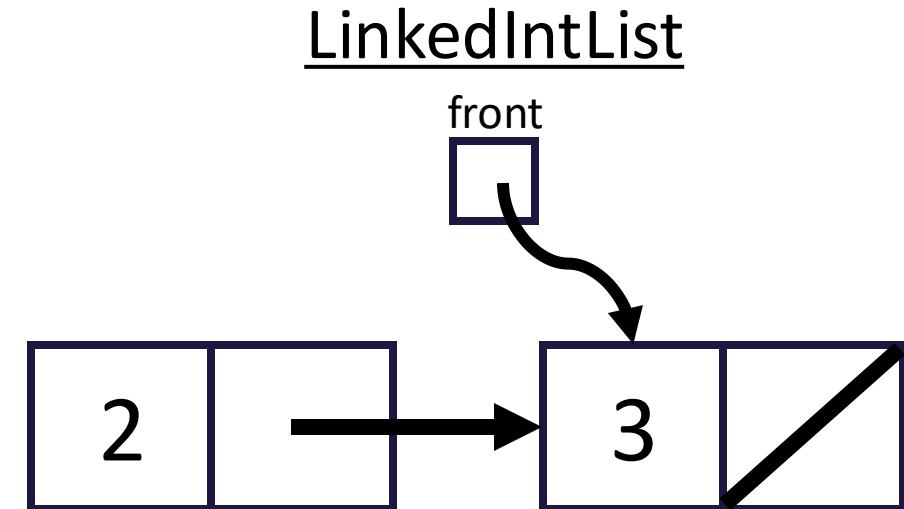
Why curr?

```
public class LinkedList {  
    private ListNode front;  
  
    public void printList() {  
        while (front != null) {  
            System.out.print(front.data + " ");  
            front = front.next;  
        }  
    }  
}
```



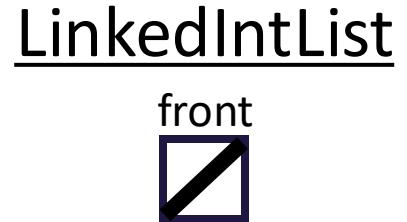
Why curr?

```
public class LinkedList {  
    private ListNode front;  
  
    public void printList() {  
        while (front != null) {  
            System.out.print(front.data + " ");  
            front = front.next;  
        }  
    }  
}
```



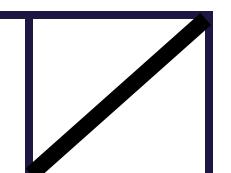
Why curr?

```
public class LinkedList {  
    private ListNode front;  
  
    public void printList() {  
        while (front != null) {  
            System.out.print(front.data + " ");  
            front = front.next;  
        }  
    }  
}
```



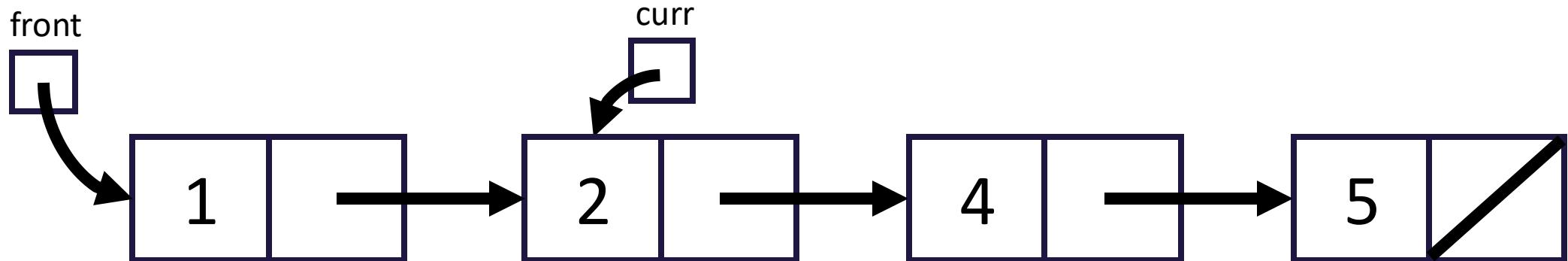
Modifying front now modifies the list!

3



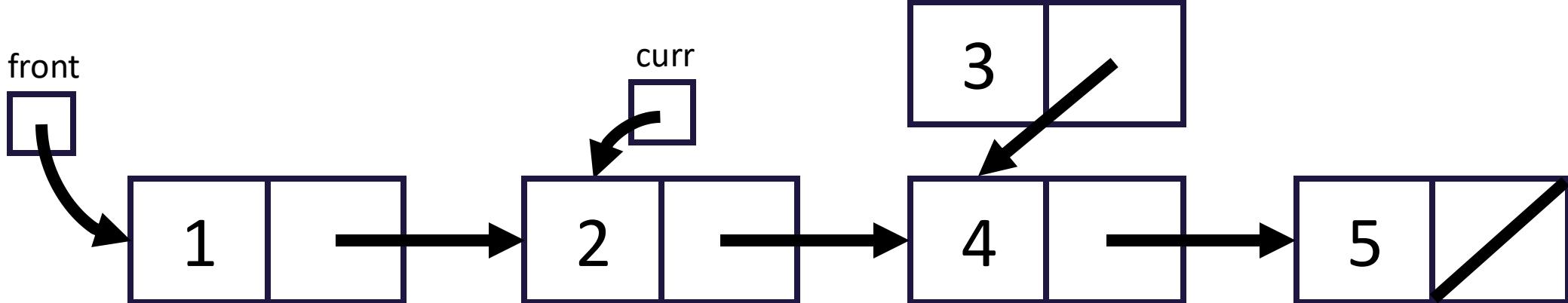
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - What will?



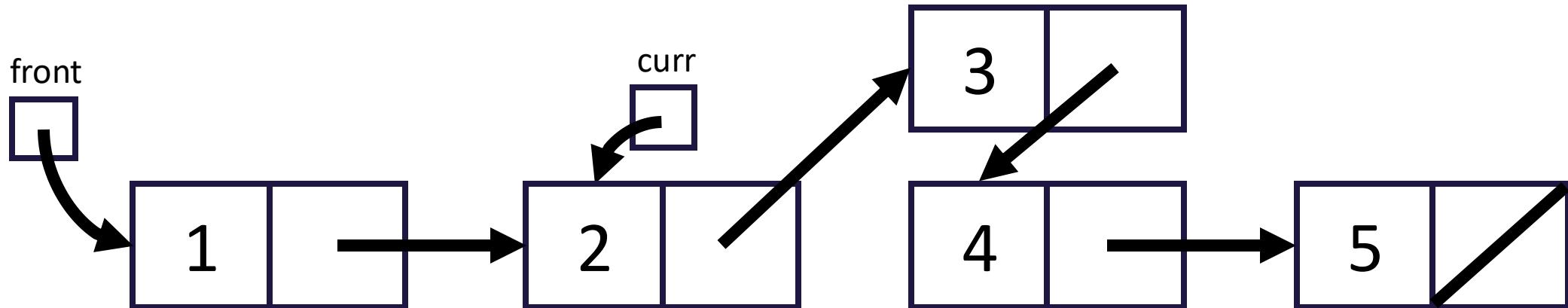
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - 1. Make a new node storing 3 pointing to 4



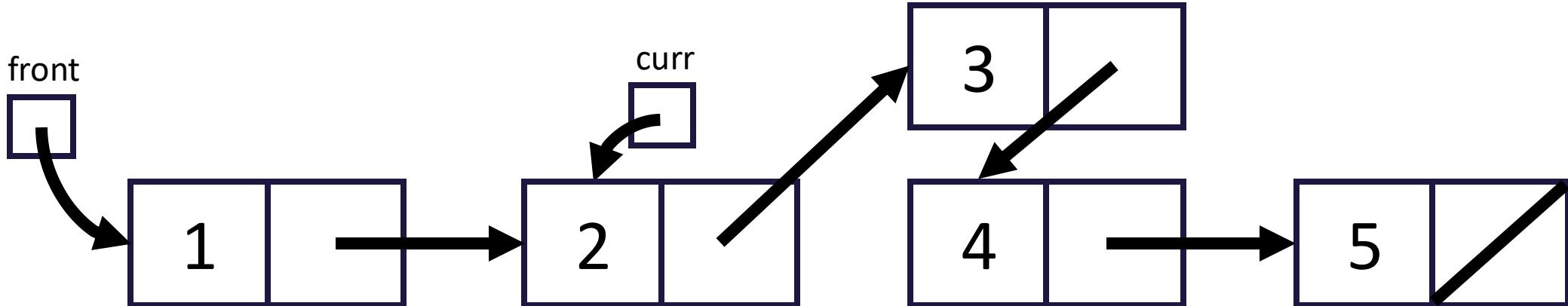
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - 1. Make a new node storing 3 pointing to 4
 - 2. Make 2 point to 3



Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - `curr.next = new ListNode(3, curr.next);`

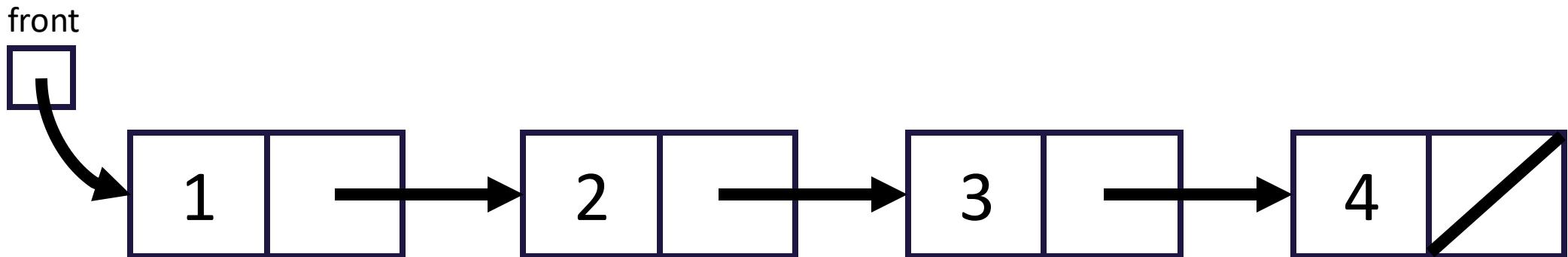


Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - What will? Changing `curr.next`

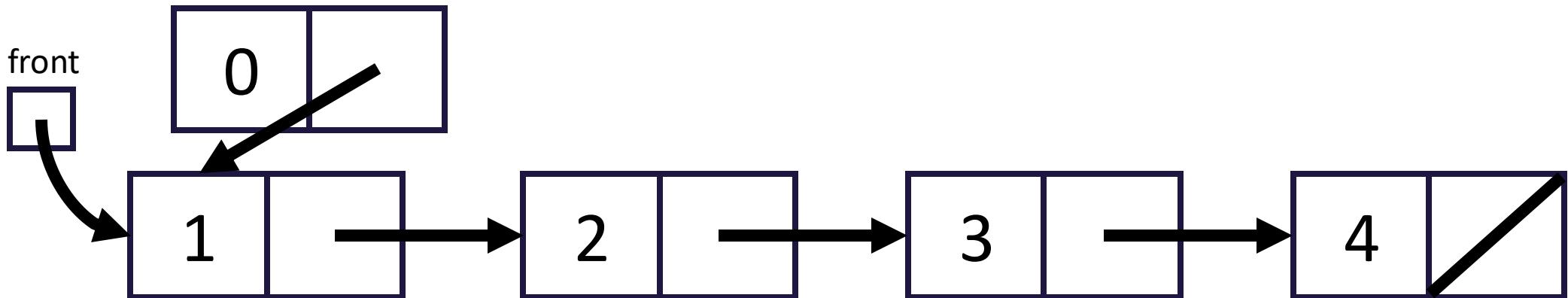
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 0 node before 1
- Is there anyway for us to do this with `curr`?
 - No!



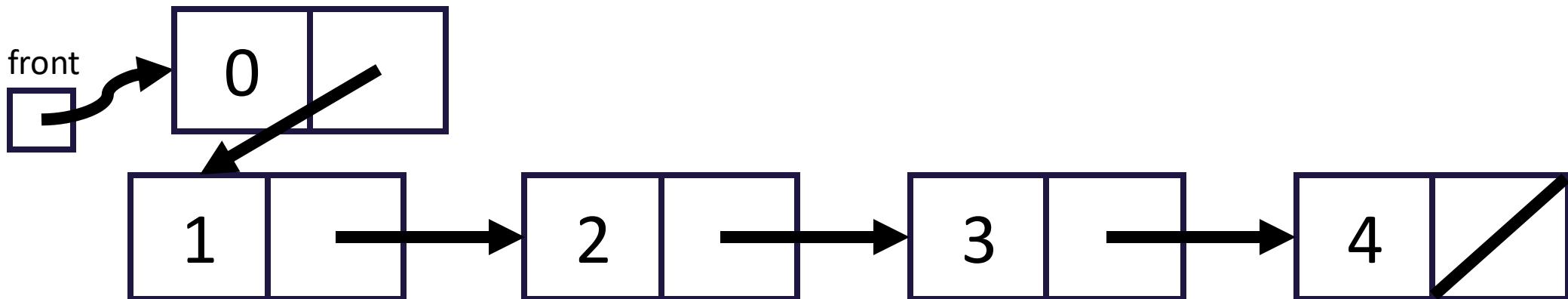
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 0 node before 1
- Is there anyway for us to do this with `curr`?
 - 1. Make a new node storing 0 pointing to 1



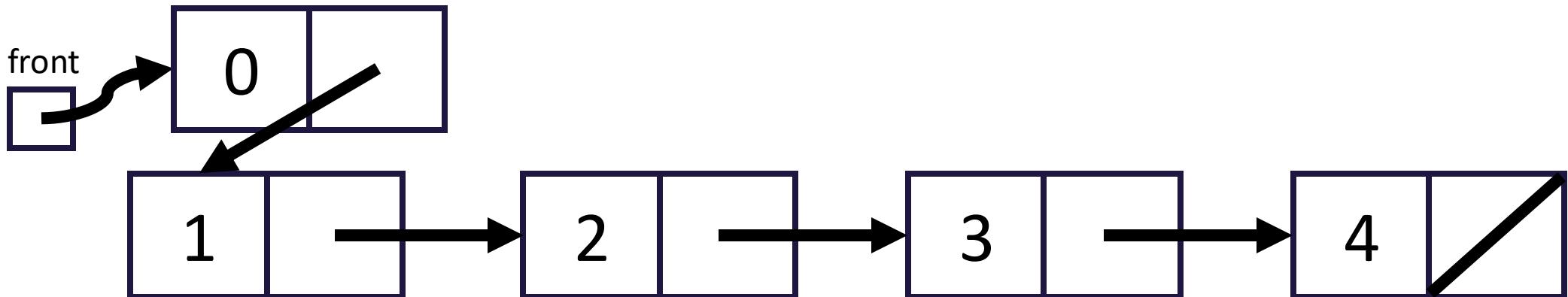
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 0 node before 1
- Is there anyway for us to do this with `curr`?
 - 1. Make a new node storing 0 pointing to 1
 - 2. Make `front` point to 0



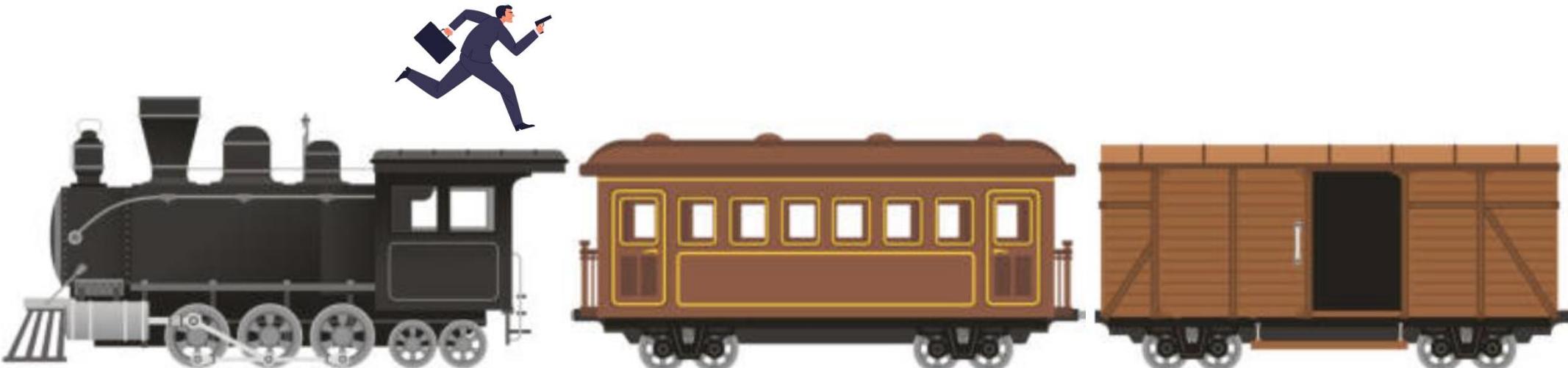
Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 0 node before 1
- Is there anyway for us to do this with `curr`?
 - `this.front = new ListNode(0, this.front);`



Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 0 node before 1
- So, what will actually change our list?
 - Changing `curr.next`, changing `front`
 - Need to **stop one early** to make changes



Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes
- We want to insert a 3 node between 2 and 4
- Does changing `curr` actually update our chain?
 - What will? Changing `curr.next`, changing `front`
 - Need to **stop one early** to make changes
- Often a number of cases to watch out for:
 - M(iddle) – Modifying node in the middle of the list (general)
 - F(ront) – Modifying the first node
 - E(mpty) – What if the list is empty?
 - E(nd) – Rare, do we need to do something with the end of the list?