LEC 06

CSE 123

LinkedIntList

Questions during Class?

Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Waffle fries, curly fries, regular fries, or tots?

Instructor: Ziao Yin

Trien Nichole Chris Packard Eeshani

Lecture Outline

- Announcements
- Revisiting the PCM (Modifying Links)
- LinkedIntList

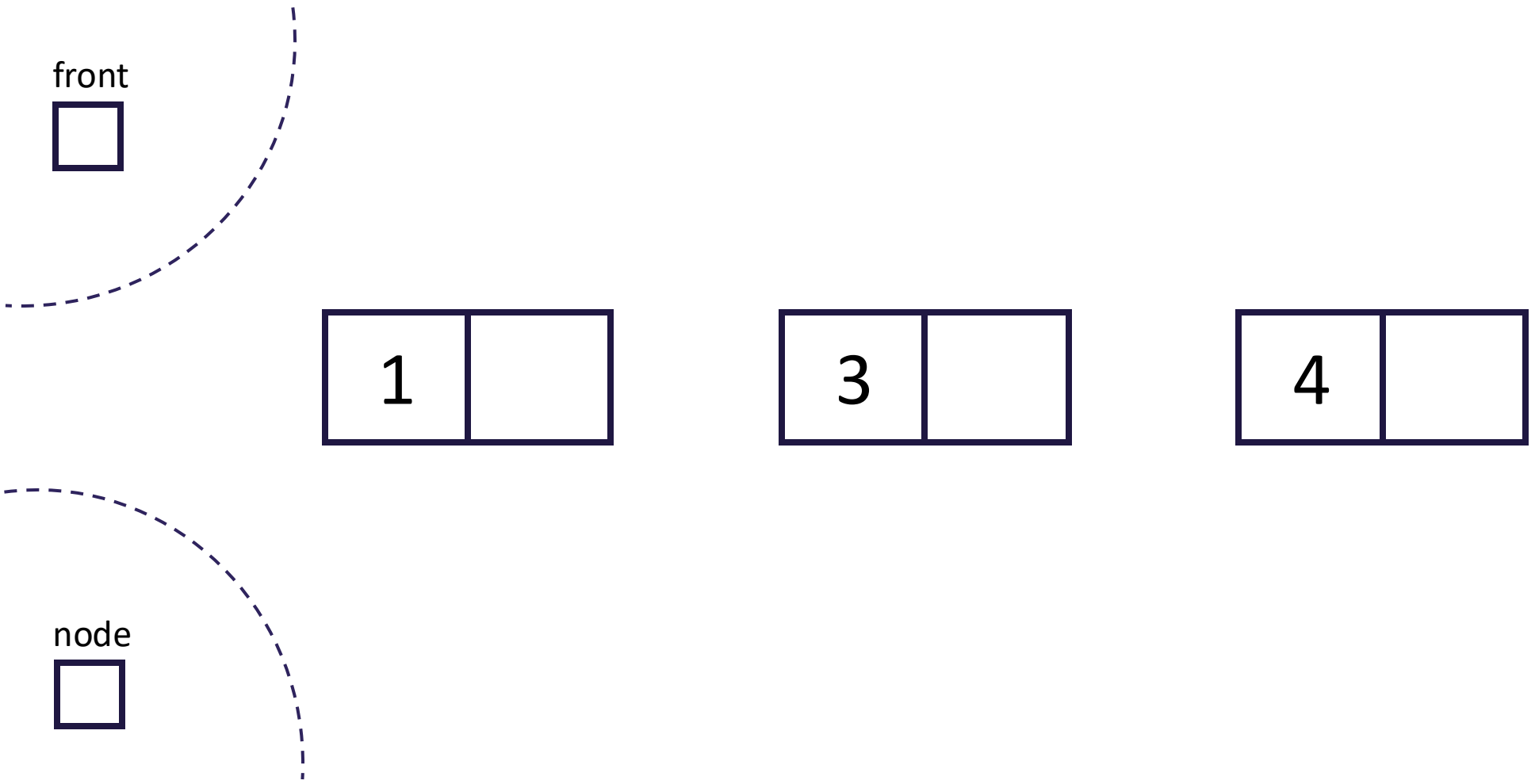
Announcements

- Quiz 0 Grades
  - Sometime later today
- R0 and P0 feedback will be released today
- Creative Project 1 due tonight at 11:59pm
  - Submit *something* so we can provide some feedback!
- Programming Project 1 releases tomorrow

Lecture Outline

- Announcements
- **Revisiting the PCM (Modifying Links)**
- LinkedIntList

Revisiting insertAfterLast



Lecture Outline

- Announcements
- Revisiting the PCM (Modifying Links)
- **LinkedIntList** 

Reminder: Implementing Data Structures

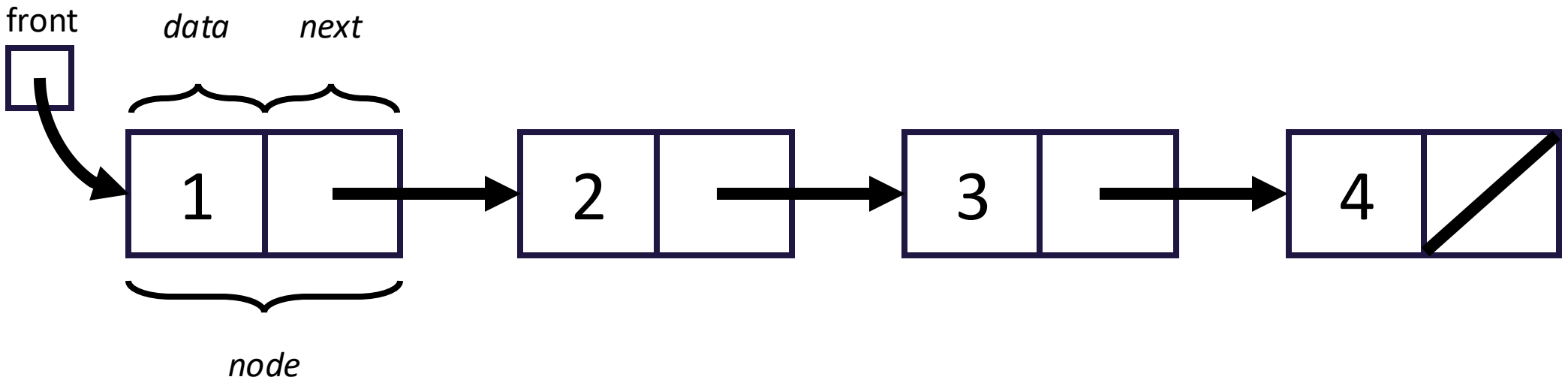
- No different from designing any other class!
  - Specified behavior (Recall the `IntList` interface):

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

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

LinkedIntList (1)

- 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`



LinkedIntList (2)

- Now that we have a `LinkedIntList` 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…

- We can make `ListNode` an inner class inside `LinkedIntList`!
  - We can still access it (just like private fields)
  - Clients don't need to worry about its existence!
  - *In the real world, we'd also make the inner static `ListNode` class `private` – we will leave it public here for ease of testing in our course environment.*

Common Cases to Consider for LinkedNodes

- Front of list
- Middle (general)
- Empty list
- End of list

Reminder: Iterating over `ListNodes`

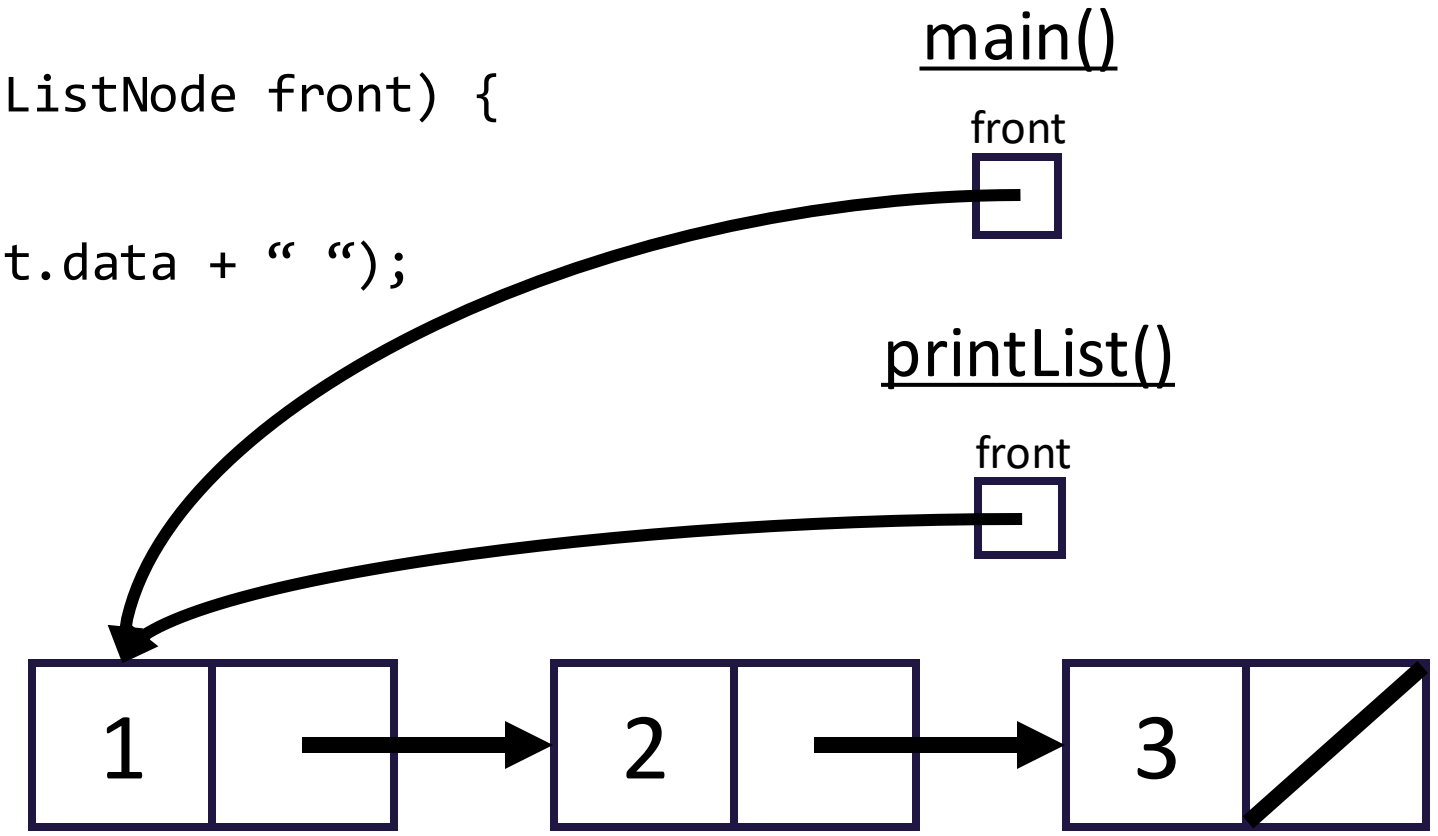
- General pattern iteration code will follow:

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

    curr = curr.next;
}
```

Why do we need a `ListNode curr`?

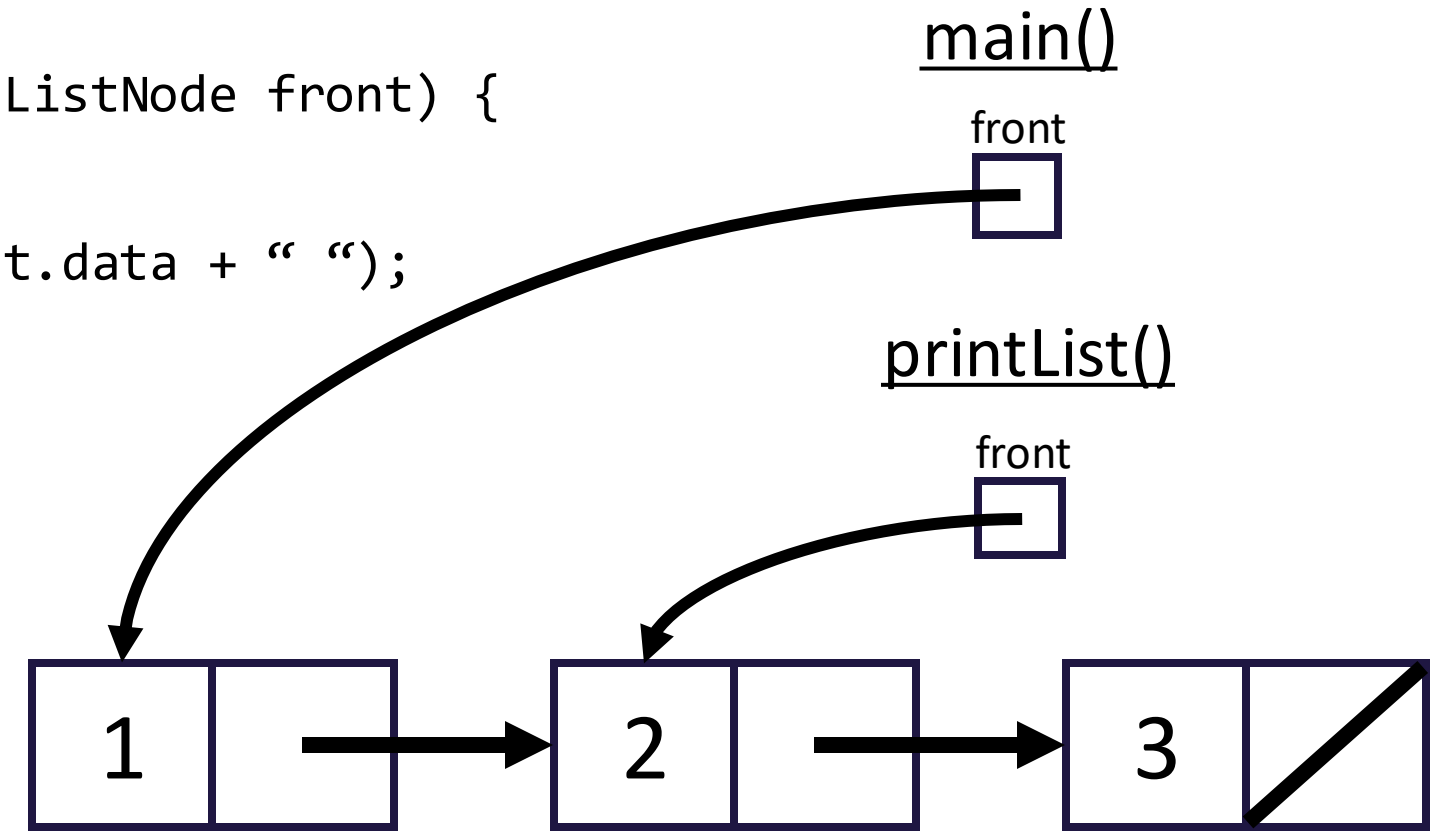
Why curr? printList(front) (1)

```
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();
}
```



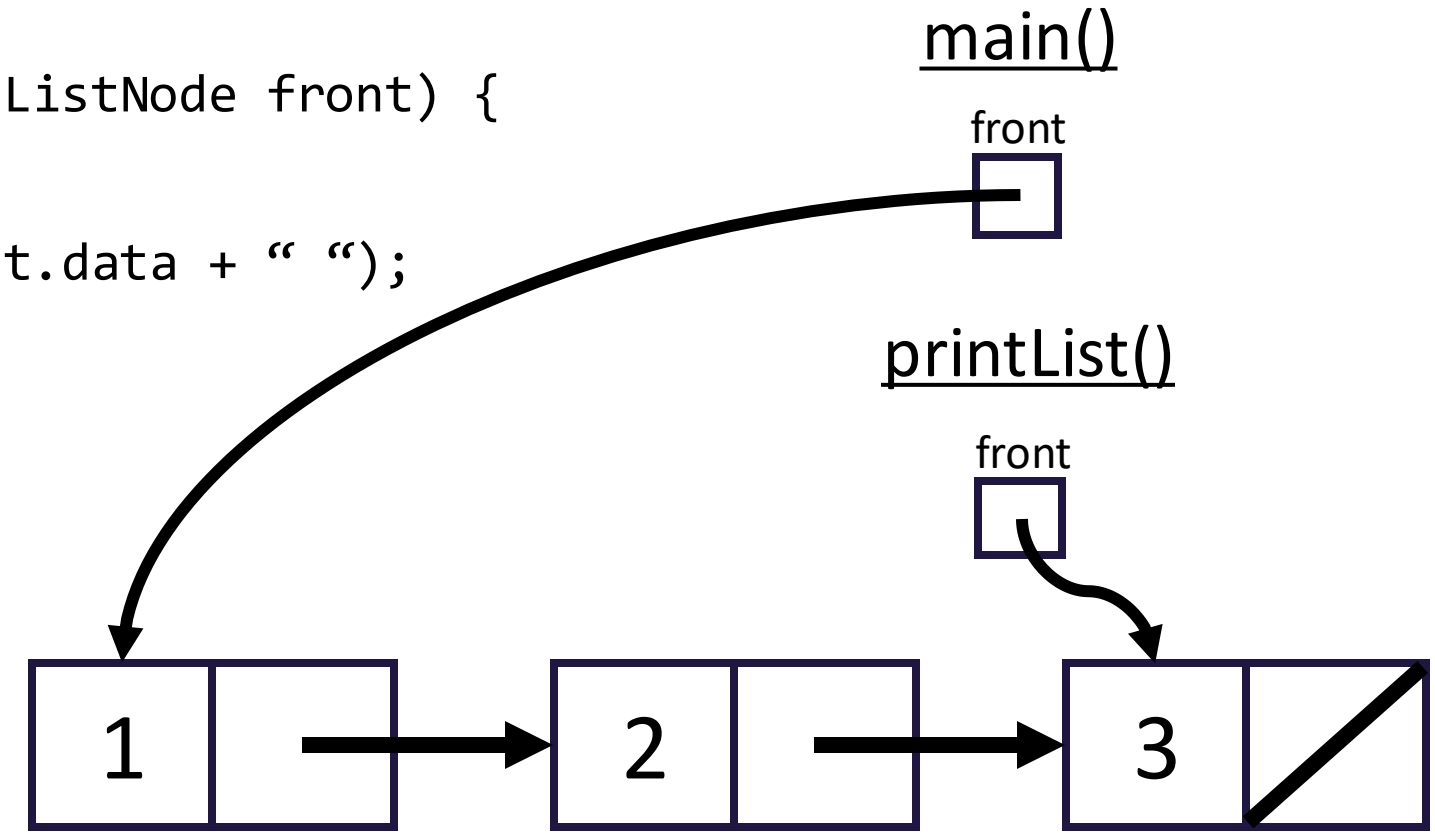
Why curr? printList(front) (2)

```
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();
}
```



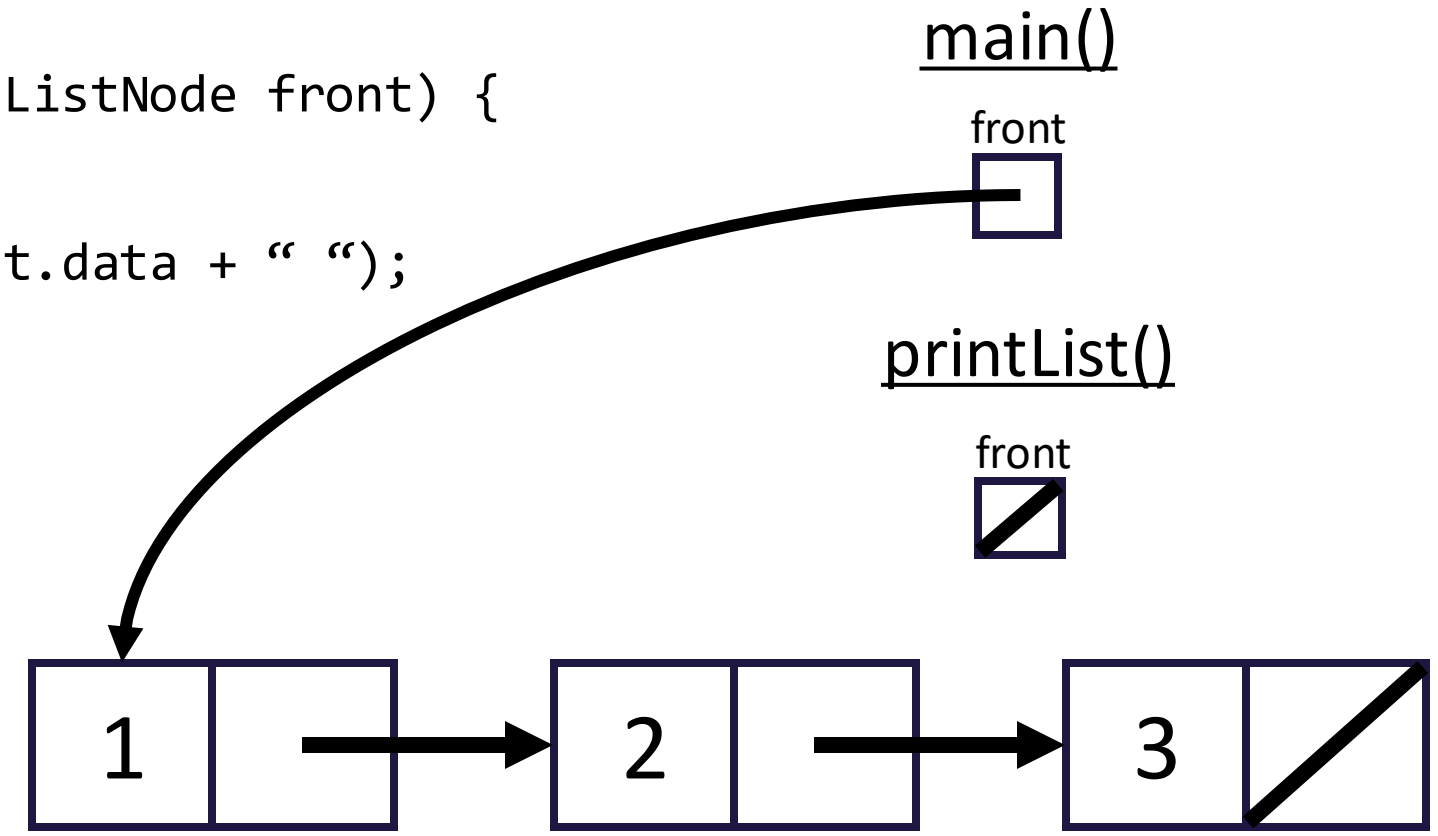
Why curr? printList(front) (3)

```
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();
}
```



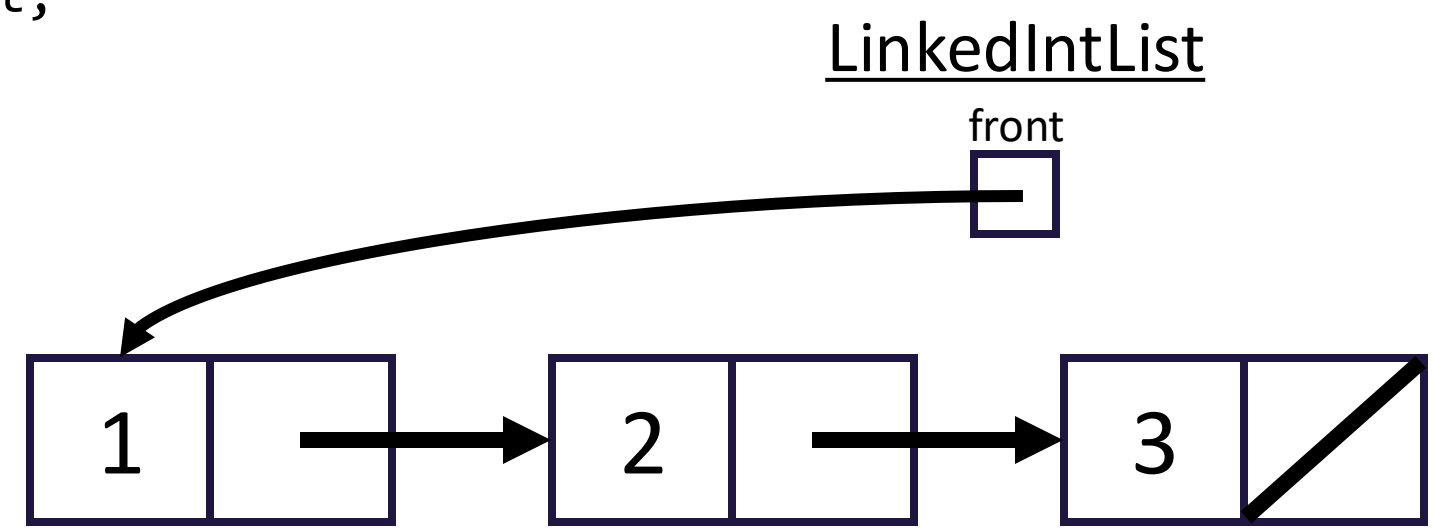
Why curr? printList(front) (4)

```
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();
}
```



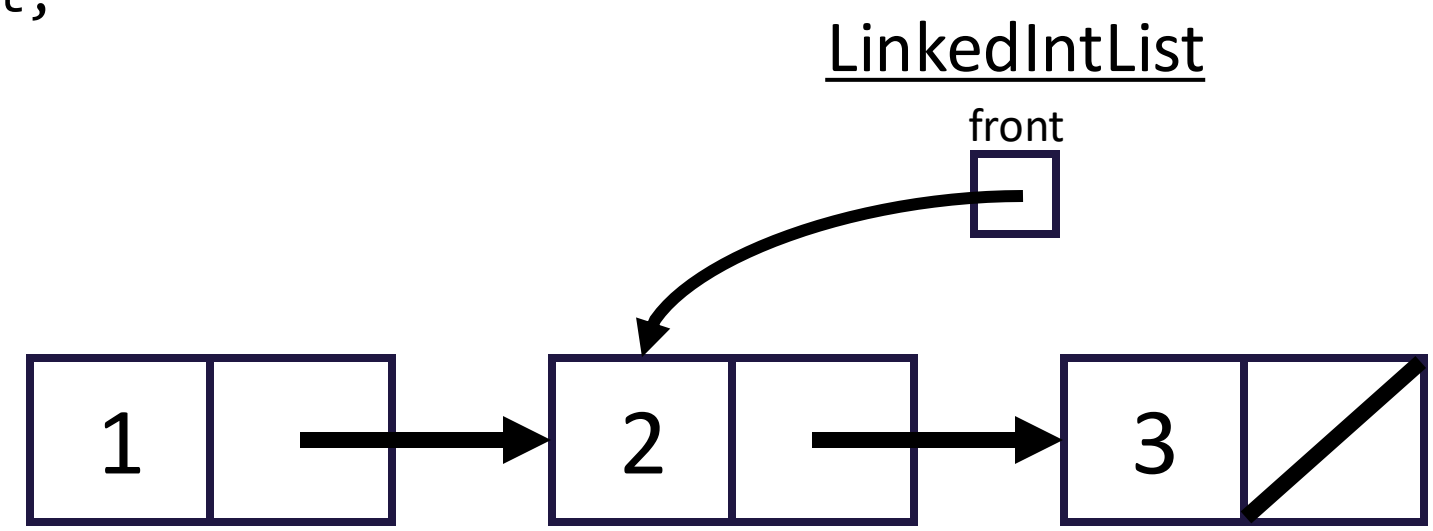
Why curr? LinkedIntList (1)

```
public class LinkedIntList {
    private ListNode front;

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



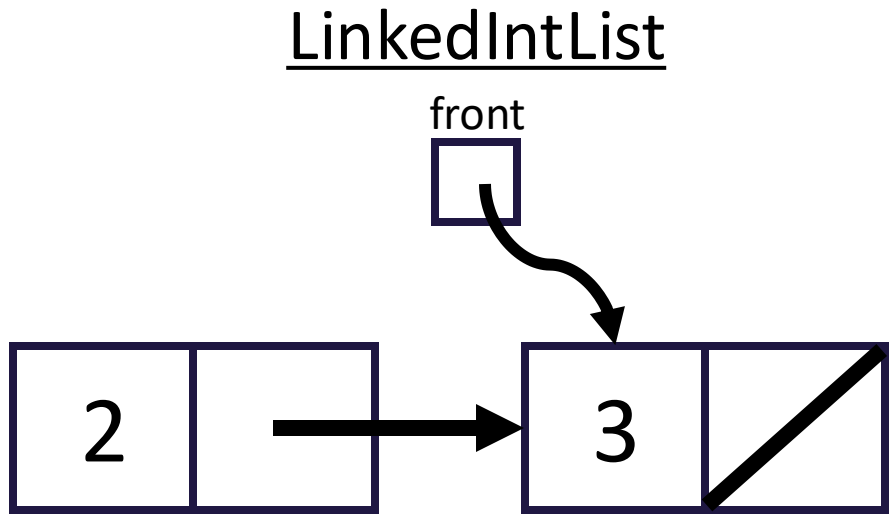
Why curr? LinkedIntList (2)

```
public class LinkedIntList {
    private ListNode front;

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



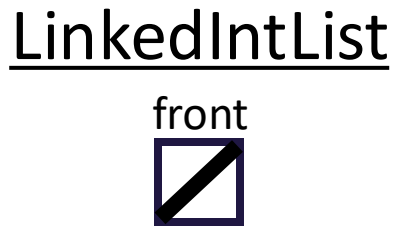
Why curr? LinkedIntList (3)

```
public class LinkedIntList {
    private ListNode front;

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

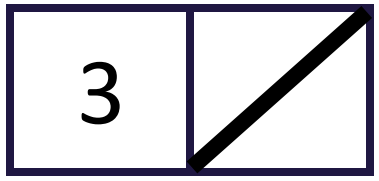


Why curr? LinkedIntList (4)

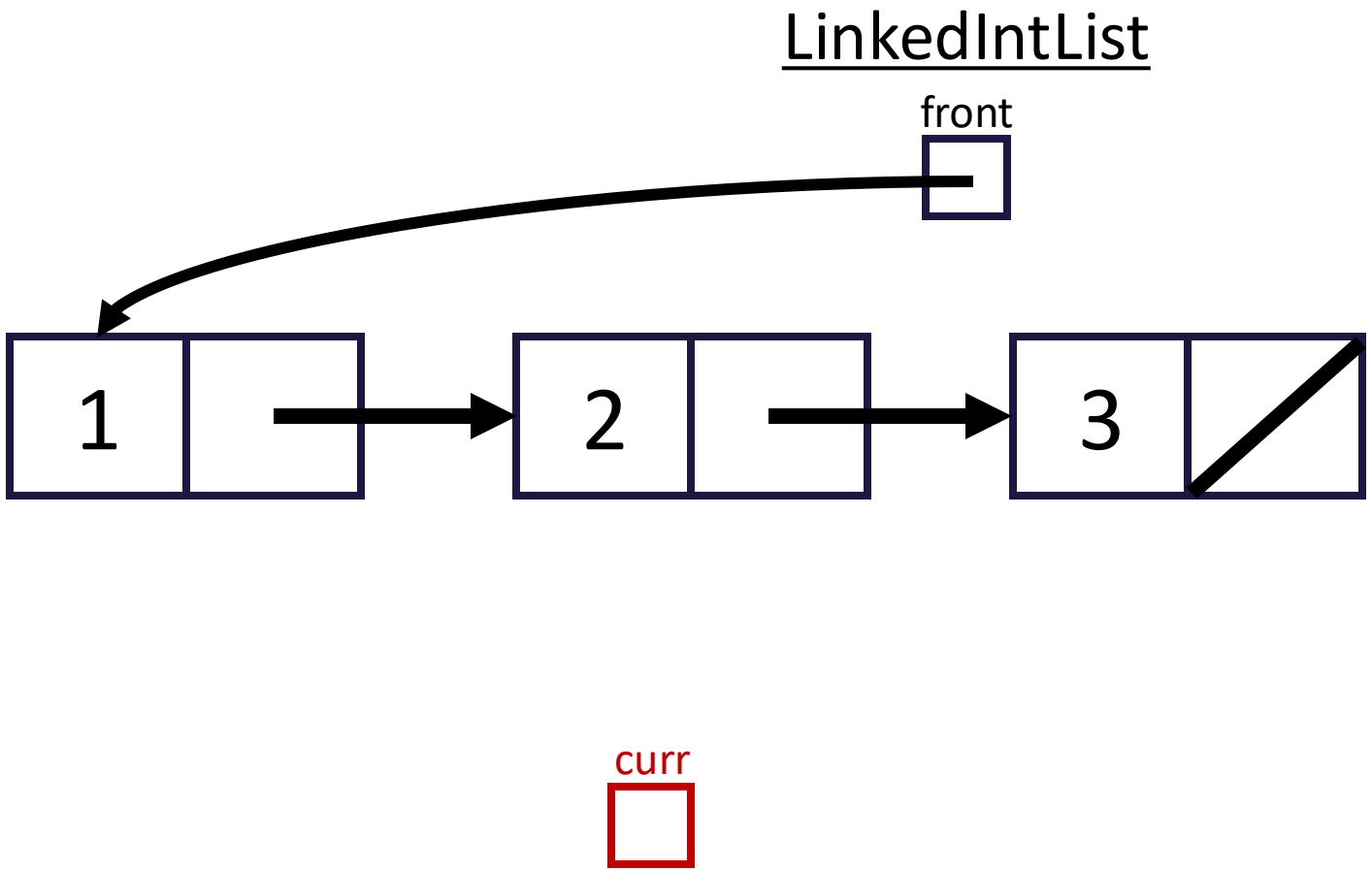
```
public class LinkedIntList {
    private ListNode front;

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



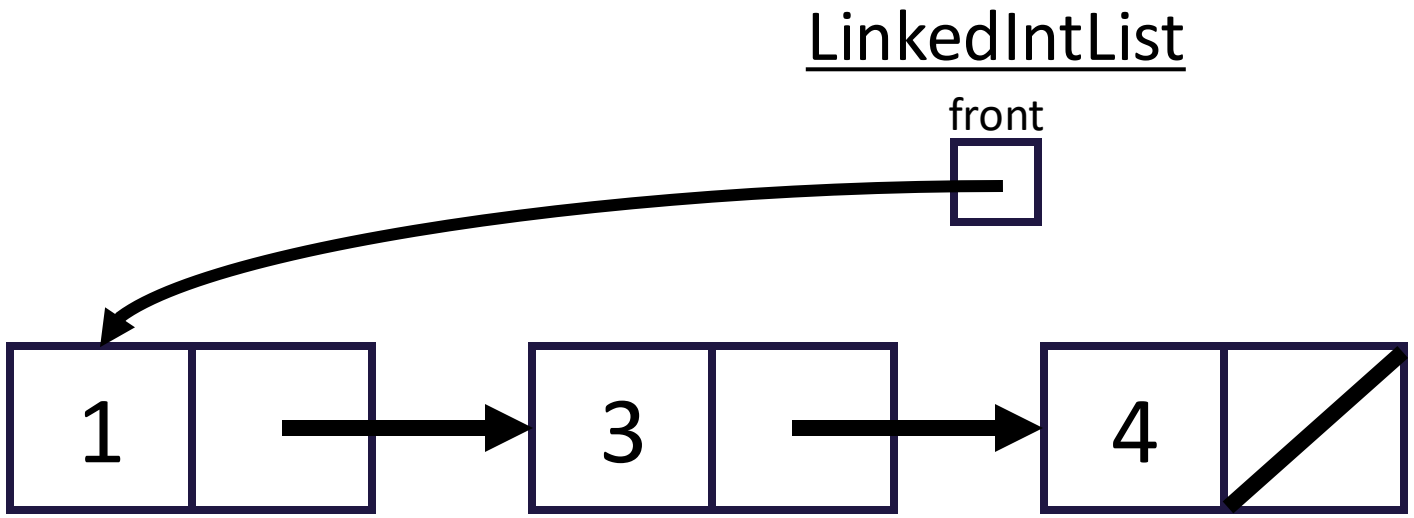
Modifying front now modifies the list!



Considering LinkedIntList printList()



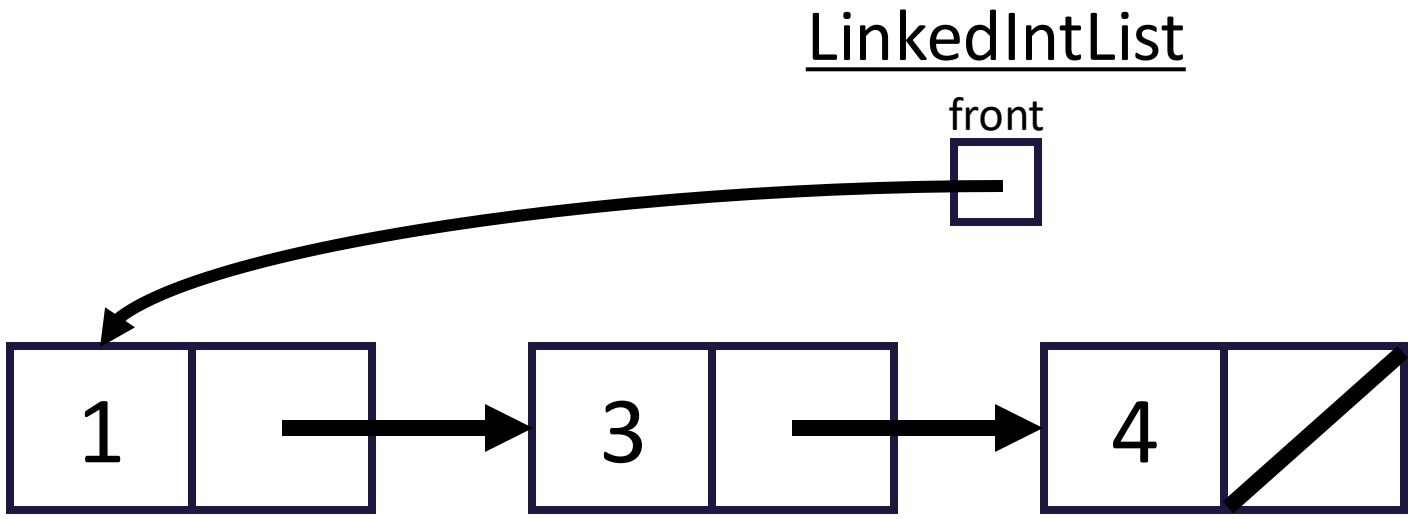
remove(value) v1



curr

```
if (curr.data == value) {
    // remove curr from the list...
}
```

remove(value) v2



curr

```
if (curr.next.data == value) {
    // remove curr from the list...
}
```

Modifying LinkedLists

- Remember: using a `curr` variable to iterate over nodes

- 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 (edge) 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?