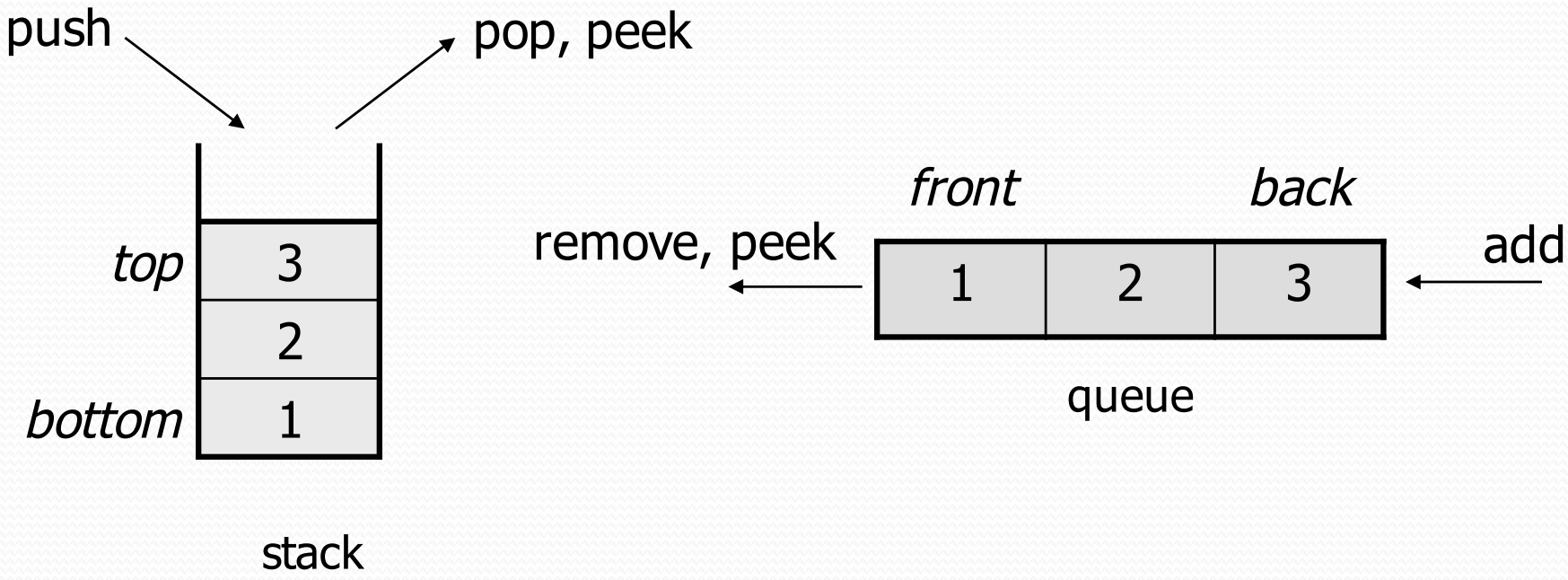
Building Java Programs

Chapter 16
Linked Nodes

reading: 16.1

Recall: stacks and queues

- **stack**: retrieves elements in reverse order as added
- **queue**: retrieves elements in same order as added



Collection efficiency

- Complexity class of various operations on collections:

Method	ArrayList	Stack	Queue
`add` (or `push`)	O(1)	O(1)	O(1)
`add(`**`index, value`**`)`	O(N)	-	-
`indexOf`	O(N)	-	-
`get`	O(1)	-	-
`remove`	O(N)	O(1)	O(1)
`set`	O(1)	-	-
`size`	O(1)	O(1)	O(1)

- Could we build lists differently to optimize other operations?

Array vs. linked structure

- All collections in this course use one of the following:
  - an **array** of all elements
    - examples: `ArrayList`, `Stack`, `HashSet`, `HashMap`

  - **linked objects** storing a value and references to other(s)
    - examples: `LinkedList`, `TreeSet`, `TreeMap`

    - First, we will learn how to create a *linked list*.
    - To understand linked lists, we must understand *references*.

Memory for a List

- Array (contiguous in memory)

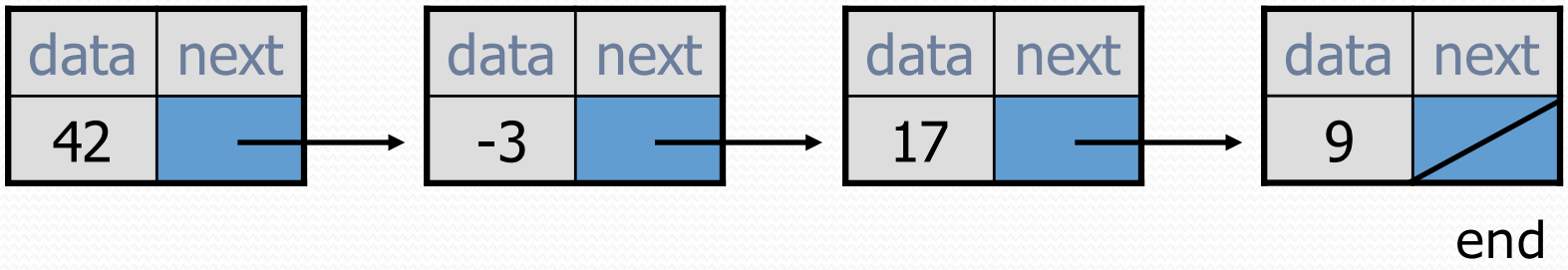
- Spread in memory

A list node class

```
public class ListNode {
    int data;
    ListNode next;
}
```

- Each list node object stores:
  - one piece of integer data
  - a reference to another list node

- `ListNode`s can be "linked" into chains to store a list of values:



References to same type

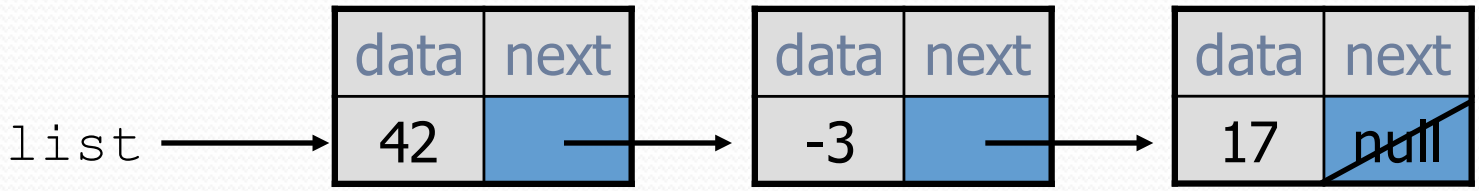
- What would happen if we had a class that declared one of its own type as a field?

```
public class Strange {
    private String name;
    private Strange other;
}
```

  - Will this compile?
    - If so, what is the behavior of the `other` field? What can it do?
    - If not, why not? What is the error and the reasoning behind it?

List node client example

```
public class ConstructList1 {
    public static void main(String[] args) {
        ListNode list = new ListNode();
        list.data = 42;
        list.next = new ListNode();
        list.next.data = -3;
        list.next.next = new ListNode();
        list.next.next.data = 17;
        list.next.next.next = null;
        System.out.println(list.data + " " + list.next.data
                           + " " + list.next.next.data);
        // 42 -3 17
    }
}
```



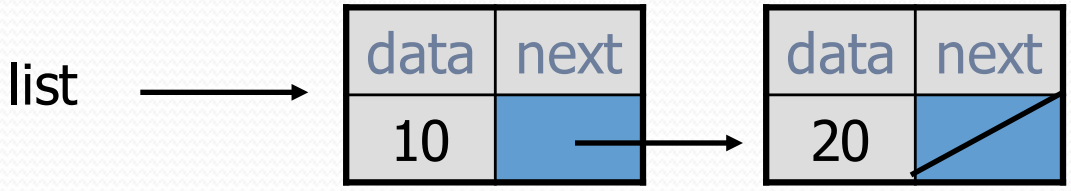
List node w/ constructor

```
public class ListNode {
    int data;
    ListNode next;

    public ListNode(int data) {
        this.data = data;
        this.next = null;
    }

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

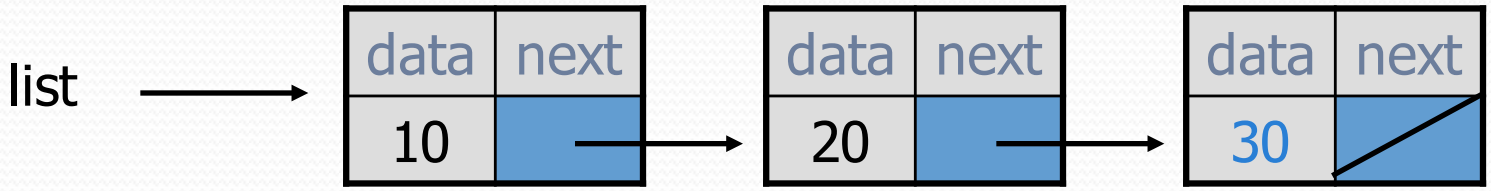
- Exercise: Modify the previous client to use these constructors.

Linked node problem 1

- What set of statements turns this picture:



- Into this?



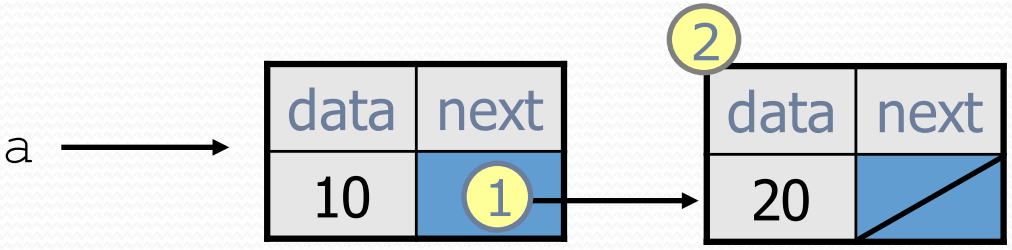
References vs. objects

variable = **value**;

a *variable* (left side of =) is an arrow (the base of an arrow)

a *value* (right side of =) is an object (a box; what an arrow points at)

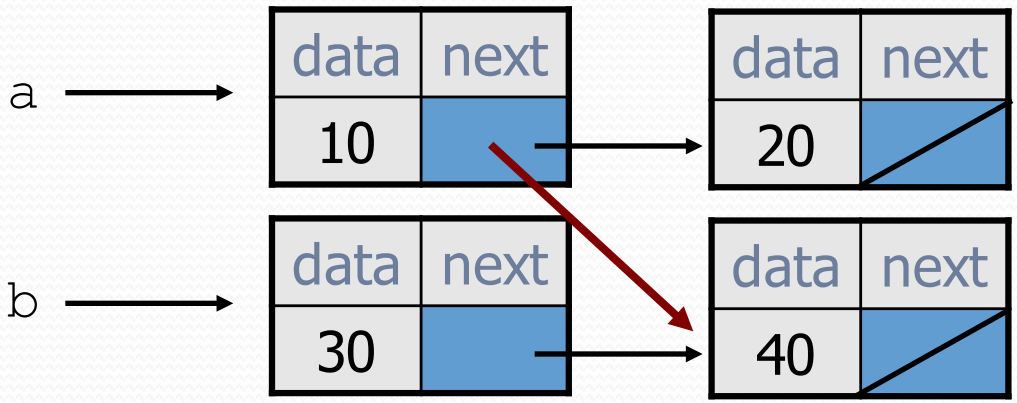
- For the list at right:
  - `a.next` = **value**;
    means to adjust where (1) points
  - **variable** = `a.next;`
    means to make **variable** point at (2)



Reassigning references

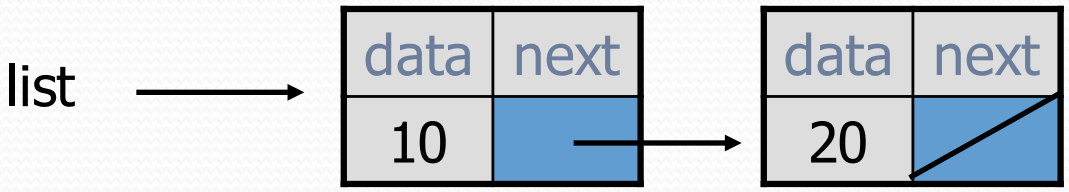
- when you say:
  - `a.next = b.next;`

- you are saying:
  - "Make *variable* `a.next` refer to the same *value* as `b.next`."
  - Or, "Make `a.next` point to the same place that `b.next` points."

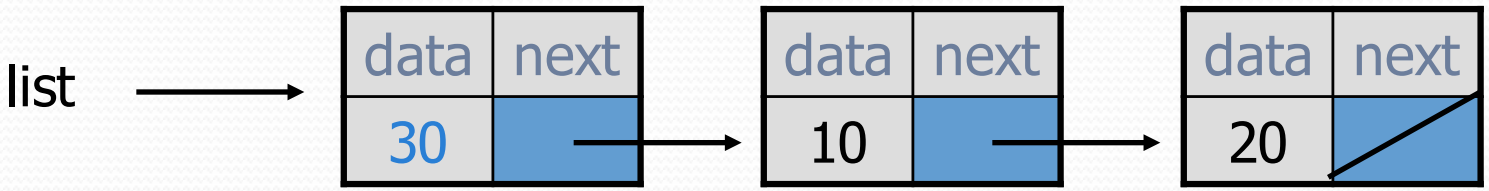


Linked node problem 2

- What set of statements turns this picture:

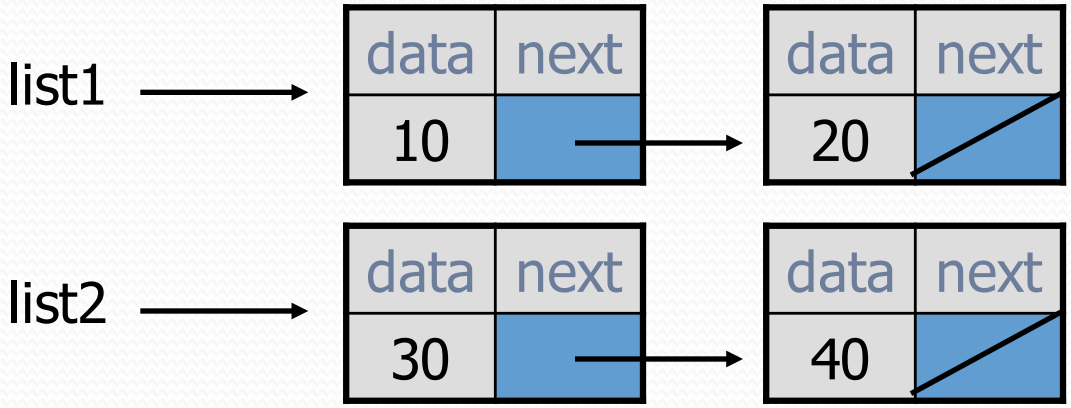


- Into this?

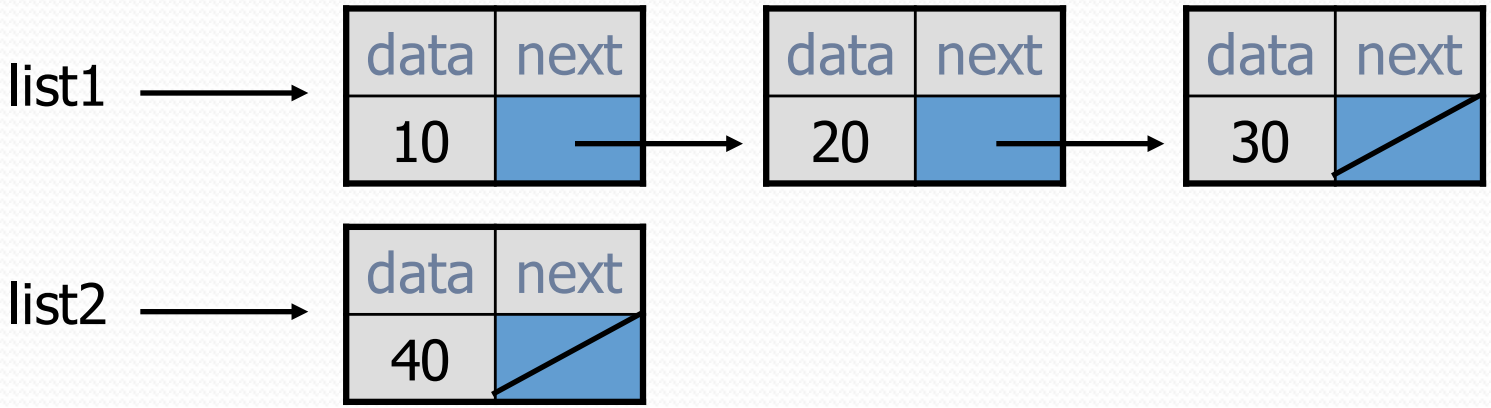


Linked node problem 3

- What set of statements turns this picture:

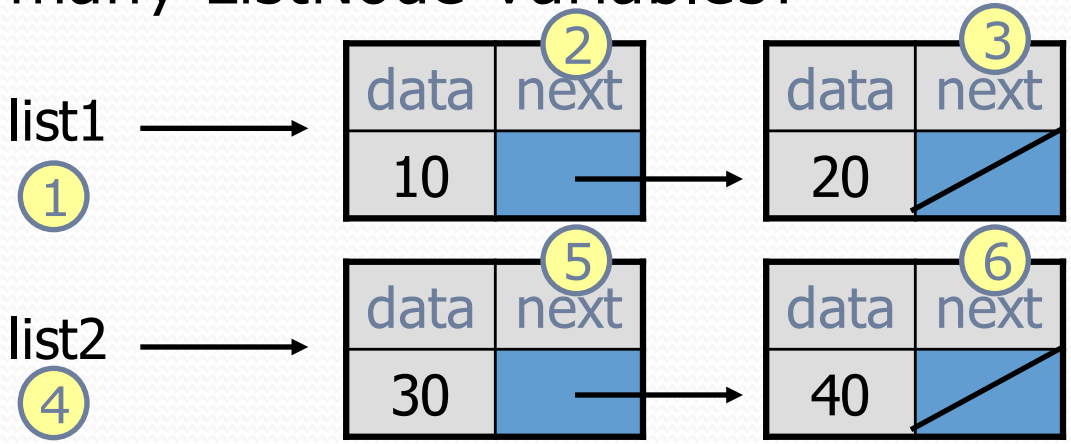


- Into this?

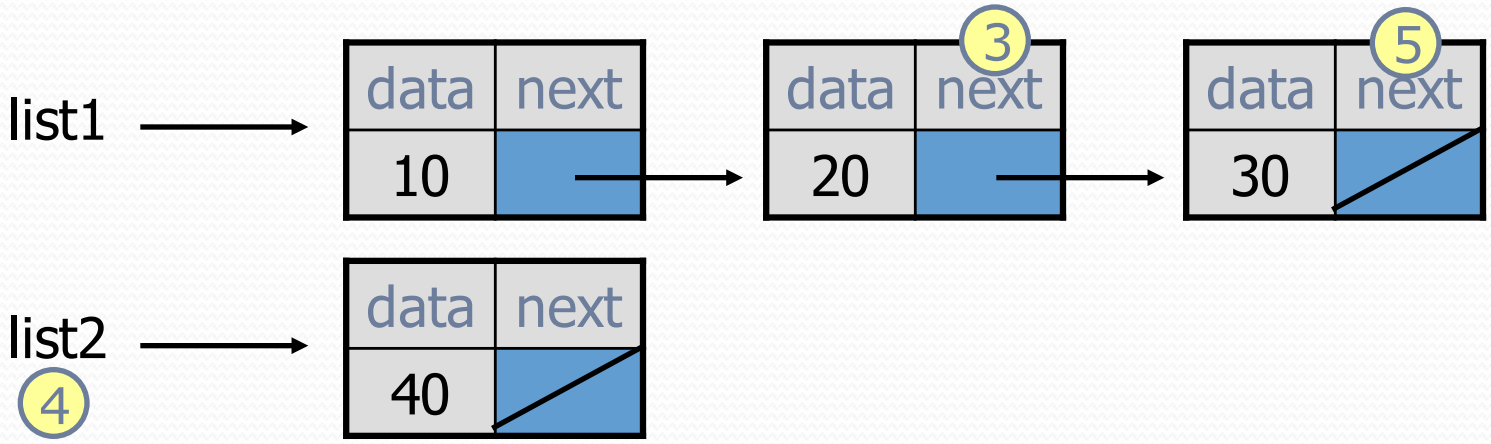


Linked node problem 3

- How many ListNode variables?

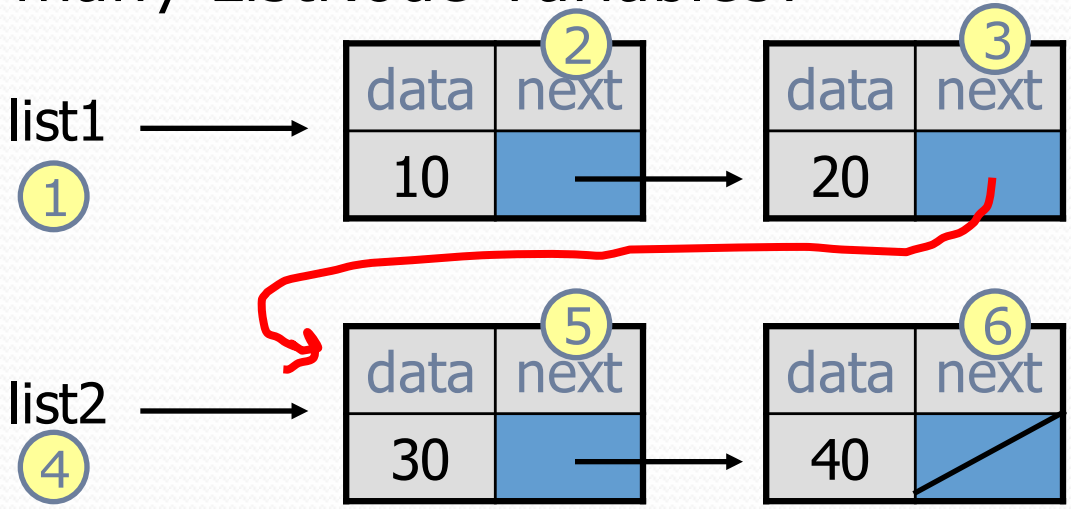


- Which variables change?



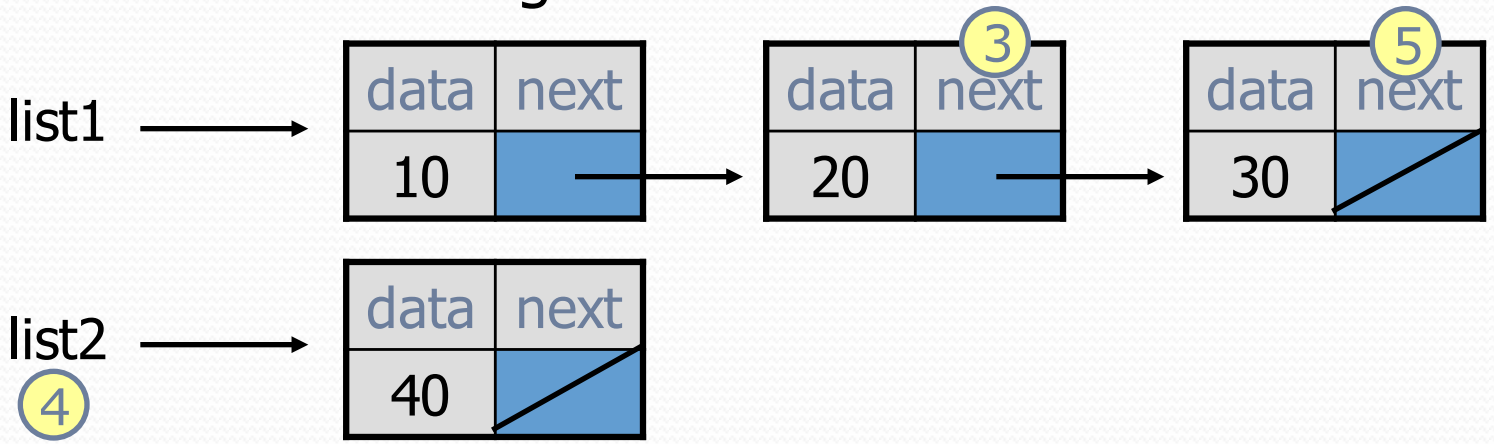
Linked node problem 3

- How many ListNode variables?



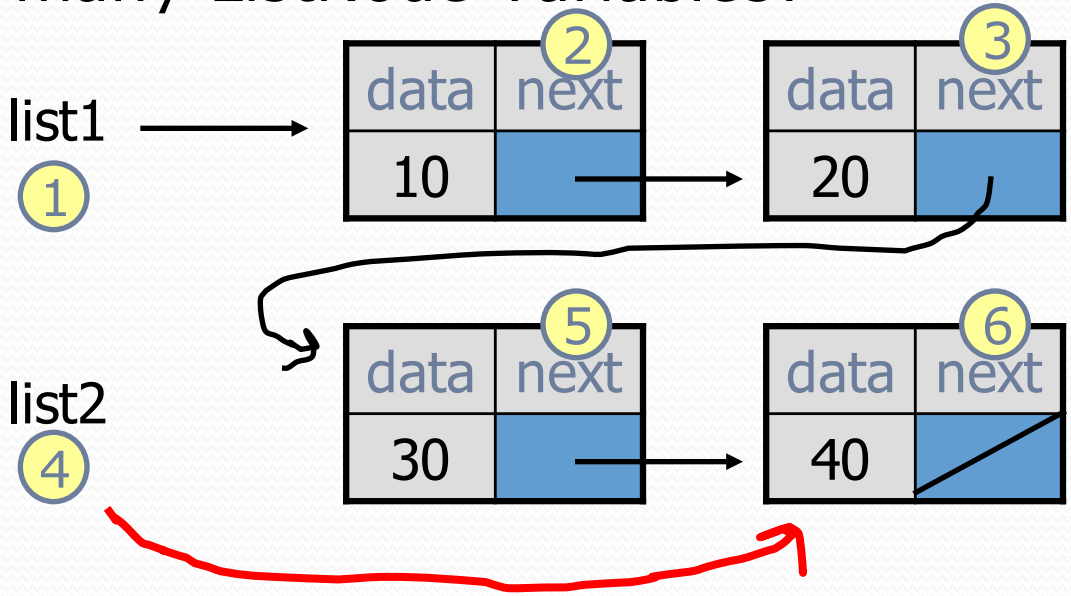
list1.next.next = list2

- Which variables change?



Linked node problem 3

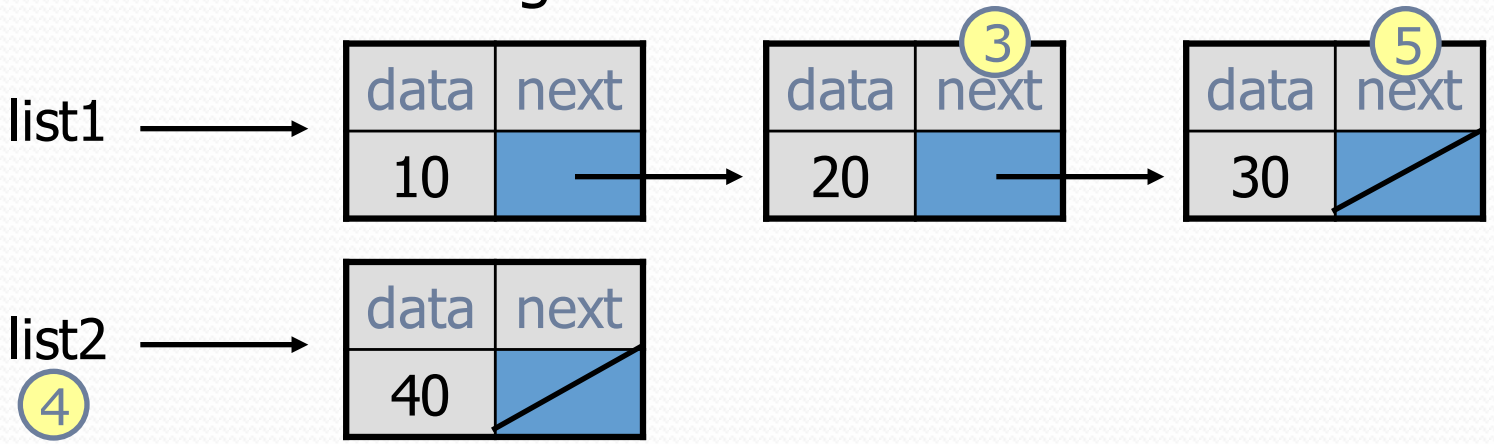
- How many ListNode variables?



list1.next.next = list2

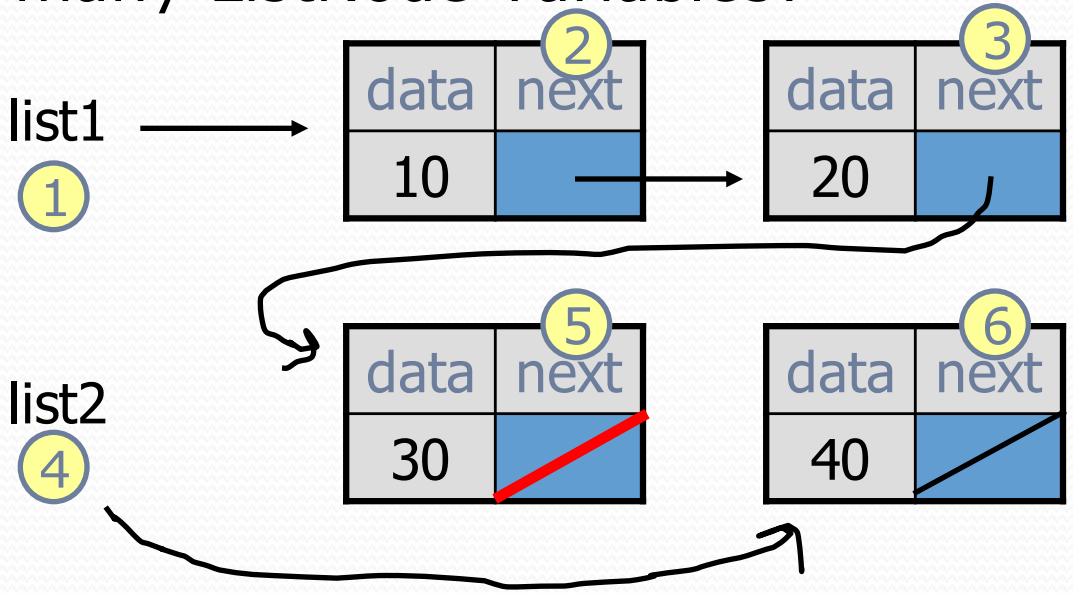
list2 = list2.next

- Which variables change?



Linked node problem 3

- How many ListNode variables?

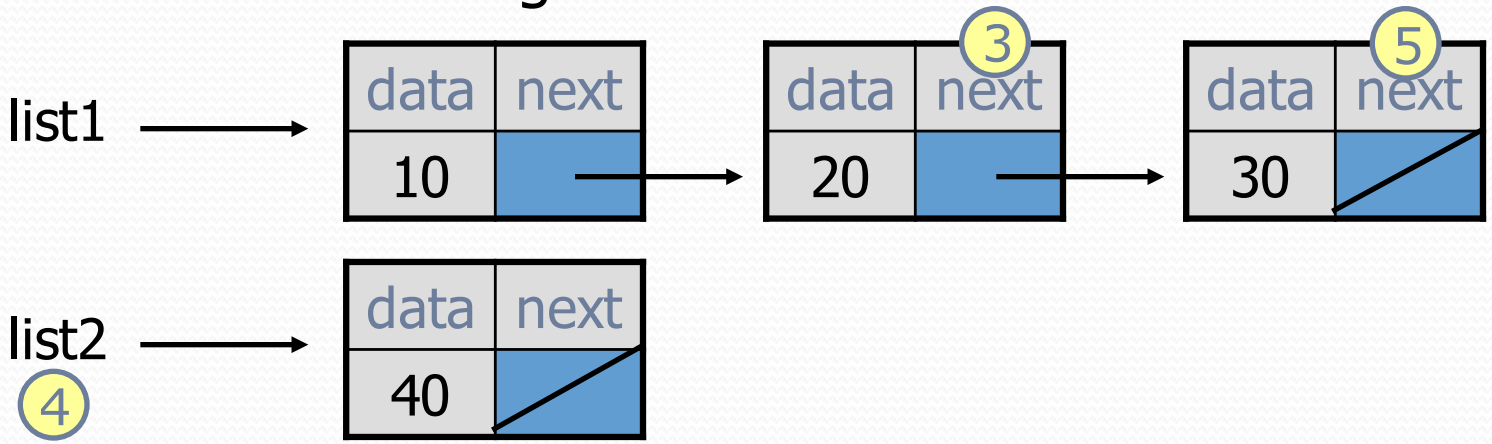


list1.next.next = list2

list2 = list2.next

list1.next.next.next = null

- Which variables change?

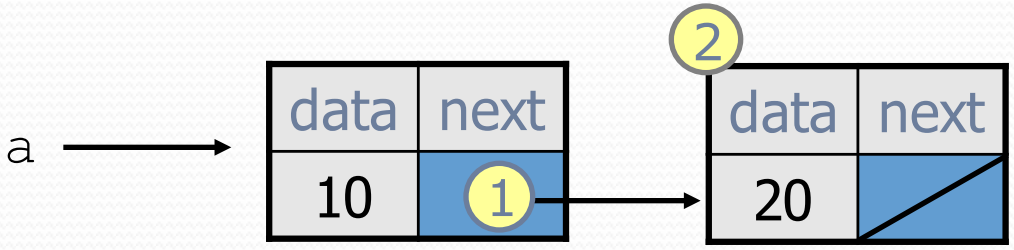


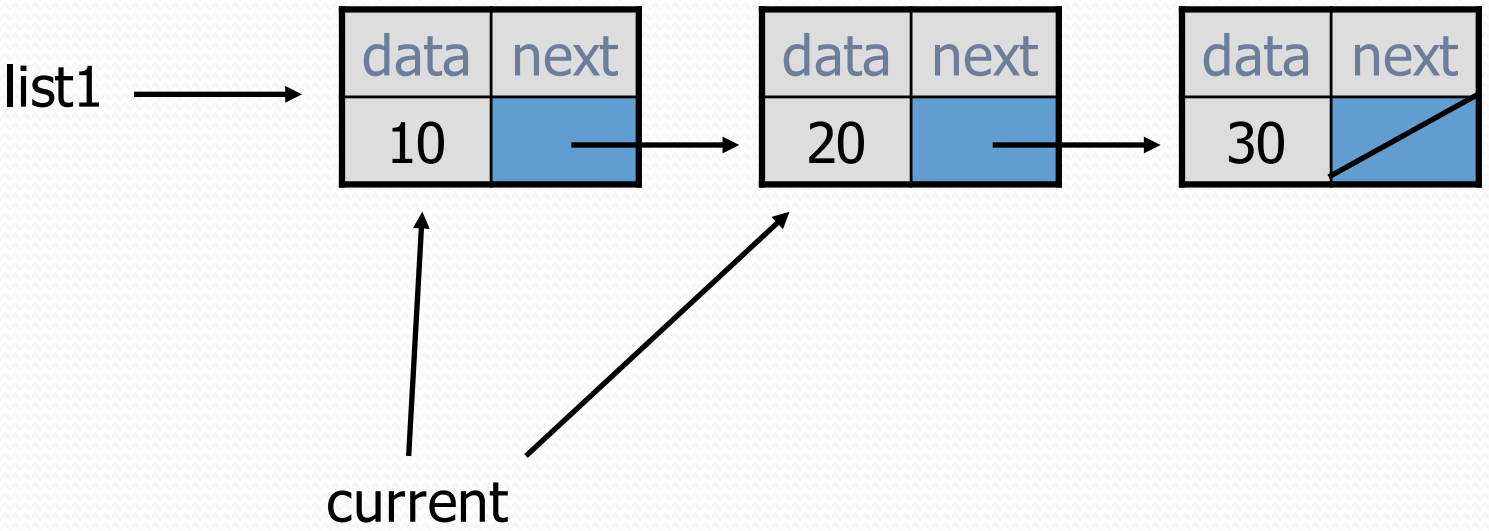
References vs. objects

variable = **value**;

a *variable* (left side of =) is an arrow (the base of an arrow)
a *value* (right side of =) is an object (a box; what an arrow points at)

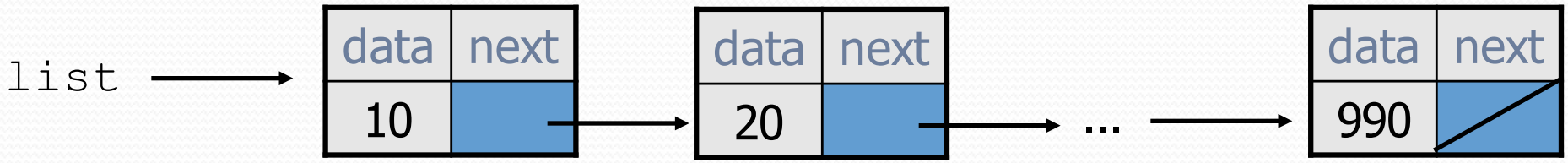
- For the list at right:
  - `a.next` = **value**;
    means to adjust where (1) points
  - **variable** = `a.next`;
    means to make **variable** point at (2)




list1
data
next
10
data
next
20
data
next
30
current

Linked node question

- Suppose we have a long chain of list nodes:



  - We don't know exactly how long the chain is.

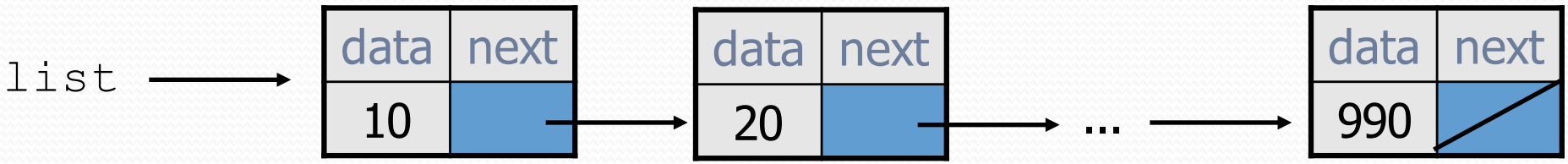
- How would we print the data values in all the nodes?

Algorithm pseudocode

- Start at the **front** of the list.
- While (there are more nodes to print):
  - Print the current node's **data**.
  - Go to the **next** node.

- How do we walk through the nodes of the list?

```
list = list.next;   // is this a good idea?
```

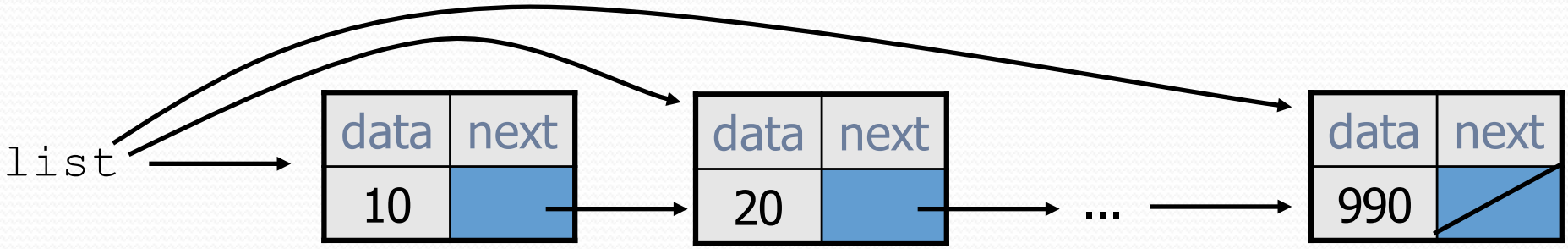


Traversing a list?

- One (bad) way to print every value in the list:

```
while (list != null) {
    System.out.println(list.data);
    list = list.next;     // move to next node
}
```

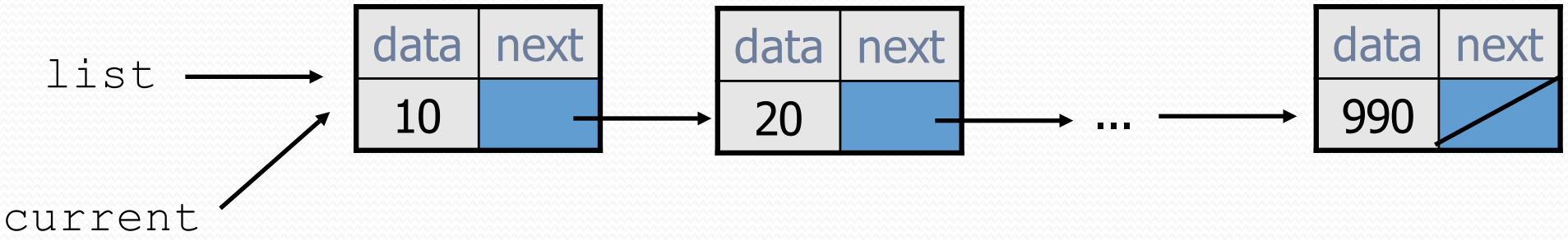
  - What's wrong with this approach?
    - (It loses the linked list as it prints it!)



A `current` reference

- Don't change `list`. Make another variable, and change it.
  - A `ListNode` variable is NOT a `ListNode` object

```
ListNode current = list;
```



- What happens to the picture above when we write:

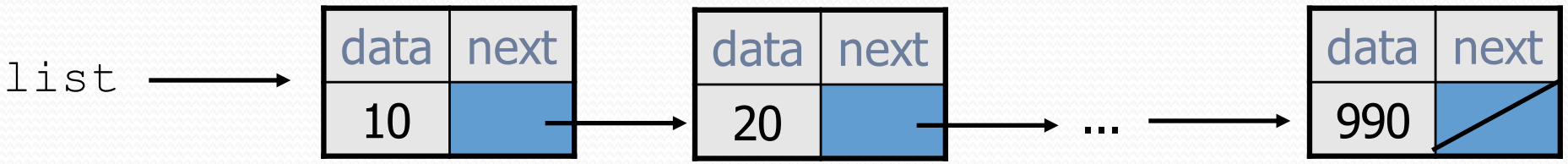
```
current = current.next;
```

Traversing a list correctly

- The correct way to print every value in the list:

```
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;  // move to next node
}
```

  - Changing `current` does not damage the list.



Linked List vs. Array

- Print list values:

```
ListNode list= ...;

ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

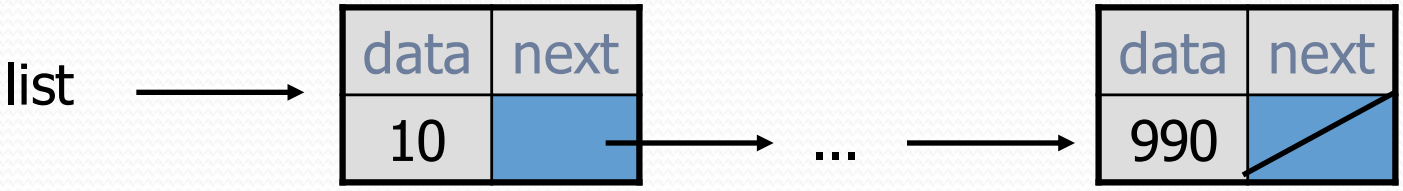
- Similar to array code:

```
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i++;
}
```

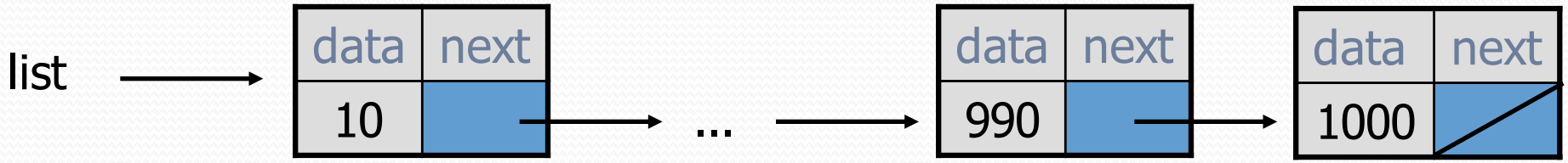
Description	Array Code	Linked List Code
Go to front of list	`int i = 0;`	`ListNode current = list;`
Test for more elements	`i < size`	`current != null`
Current value	`elementData[i]`	`current.data`
Go to next element	`i++;`	`current = current.next;`

Linked node problem 4

- What set of statements turns this picture:



- Into this?



Arrays vs. linked lists

- Array advantages
  - Random access: can quickly retrieve any value
- Array disadvantages
  - Adding/removing in middle is O(n)
  - Expanding requires creating a new array and copying elements
- Linked list advantages
  - Adding/removing in middle is O(1)
  - Expanding is O(1) (just add a node)
- Linked list disadvantages
  - Sequential access: can't directly retrieve any value