

CSCI 132:

Basic Data Structures and Algorithms

Linked Lists (Part 2)
Doubly Linked List

Reese Pearsall
Spring 2023

Lab 6 due Tuesday @ 11:59 PM
(Circular Linked Lists)

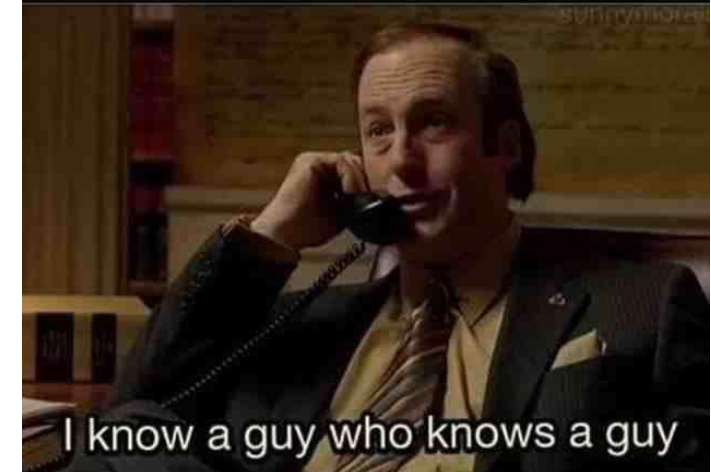
Program 2 due Friday 3/10 @ 11:59 PM
(Circular Linked Lists)

- We will try to talk about it on Monday

Next week we are covering some important stuff 😊

- *(Not a good week to ignore the class)*

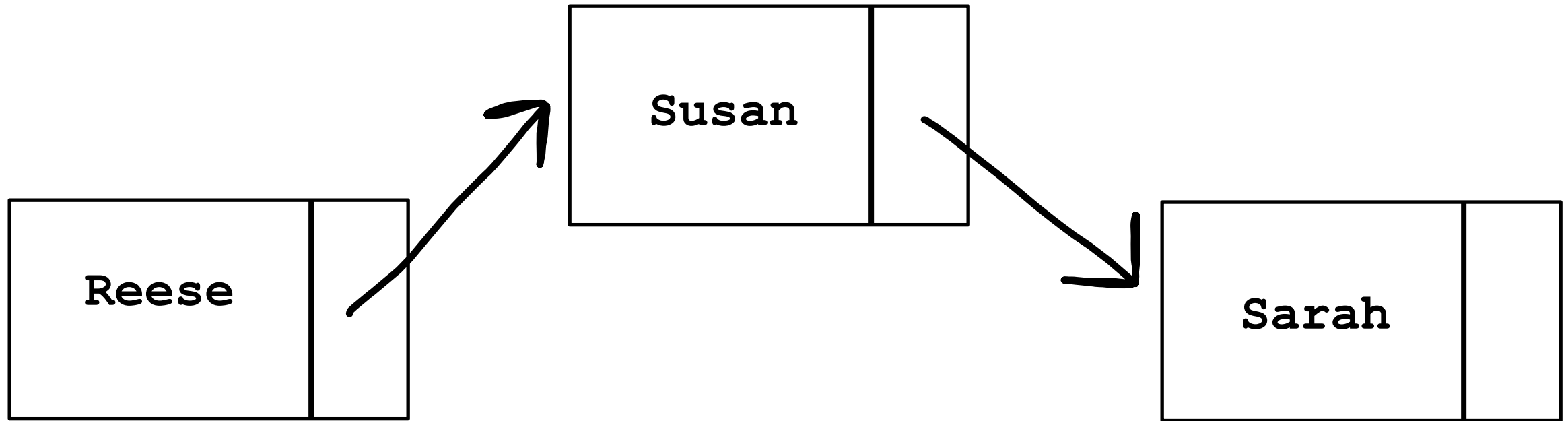
Linked List data structures be like:



when you ask stack overflow how to get the first element in a linked list

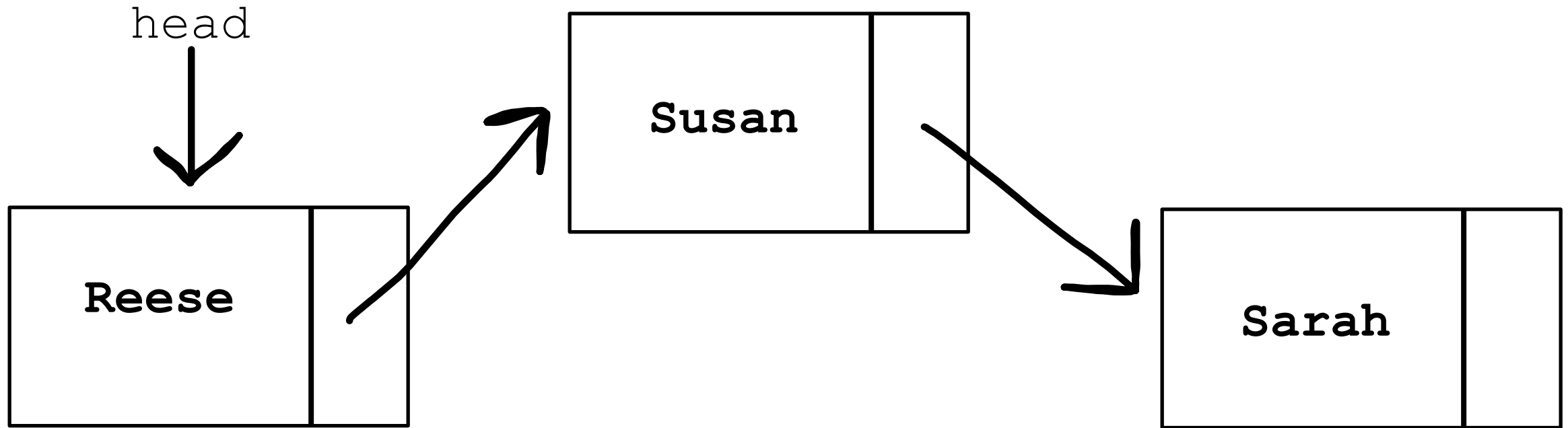


A **Linked List** is a data structure that consists of a collection of connected nodes



Nodes consists of **data** (String, int, array, etc) and a **pointer to the next node**

A **Linked List** is a data structure that consists of a collection of connected nodes



Nodes consists of **data** (String, int, array, etc) and a **pointer to the next node**

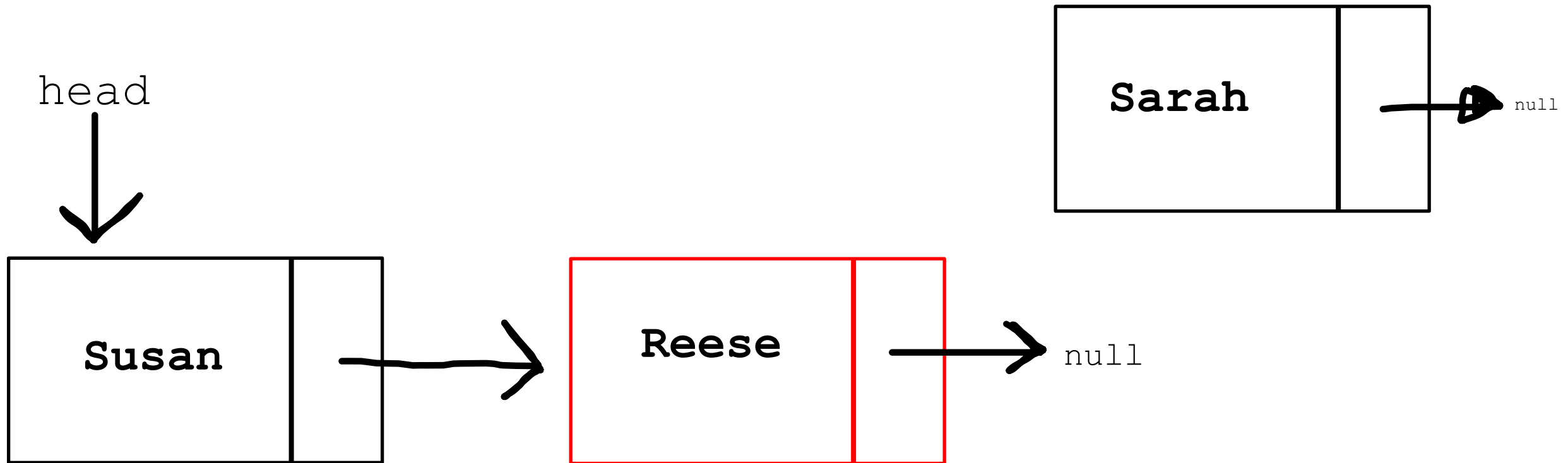
A Linked List also has a pointer to the start of the Linked List (`head`)

Linked List Methods

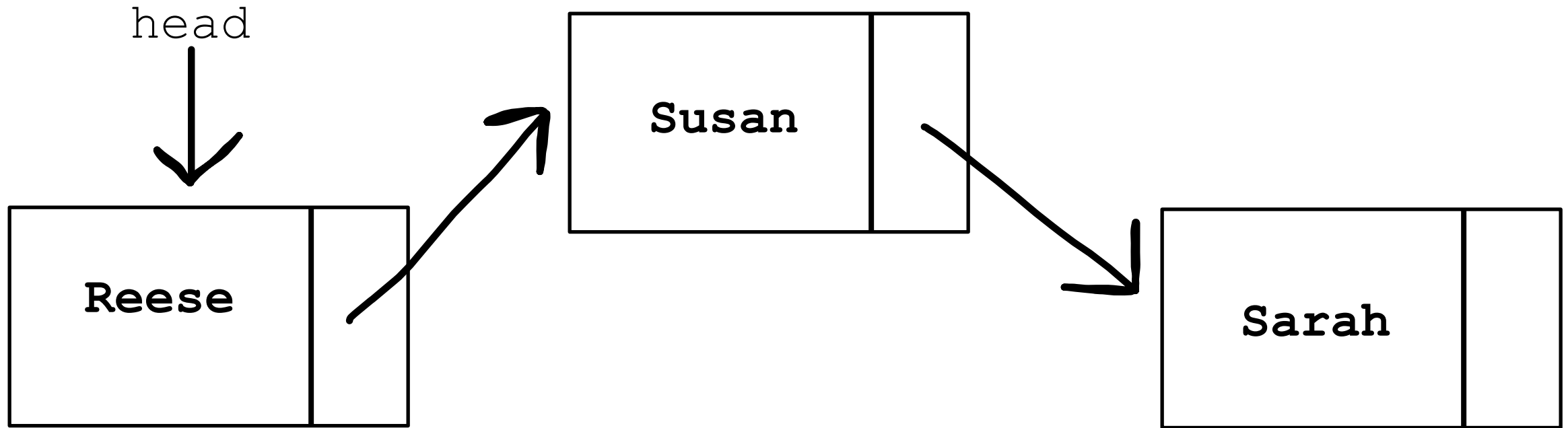
- `removeLast()` – removes last node of LL

1. Find the **second to last node**
2. Set that node's next value to null

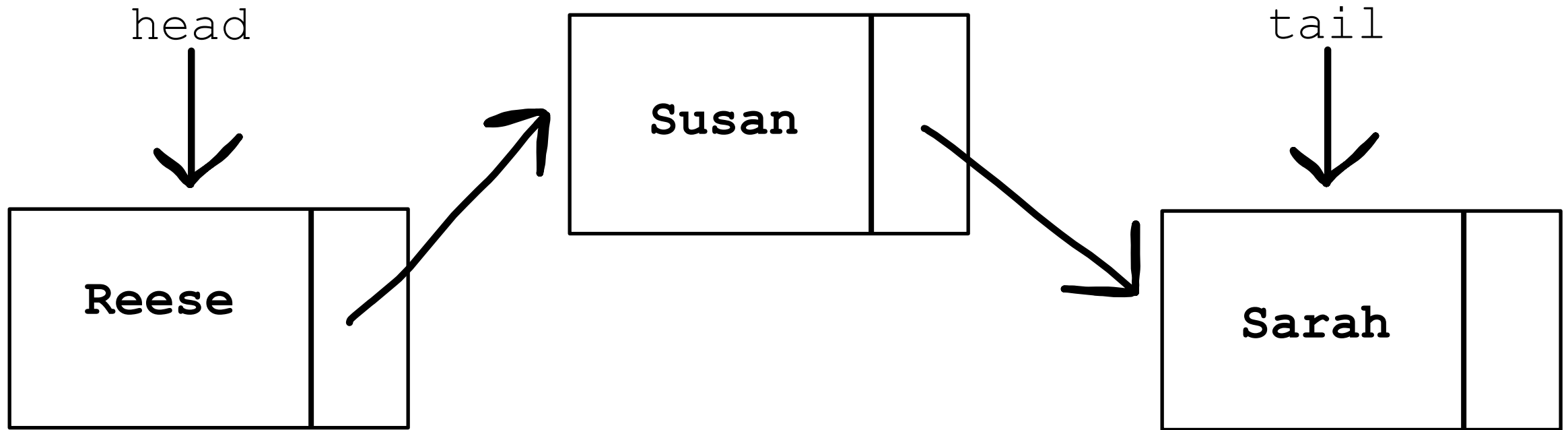
```
public void removeLast() {  
  
    Node current = head;  
    while(current.getNext().getNext() != null) {  
        current = current.getNext();  
    }  
    current.setNext(null);  
}
```



A **Singly Linked List** only keeps track of the next node

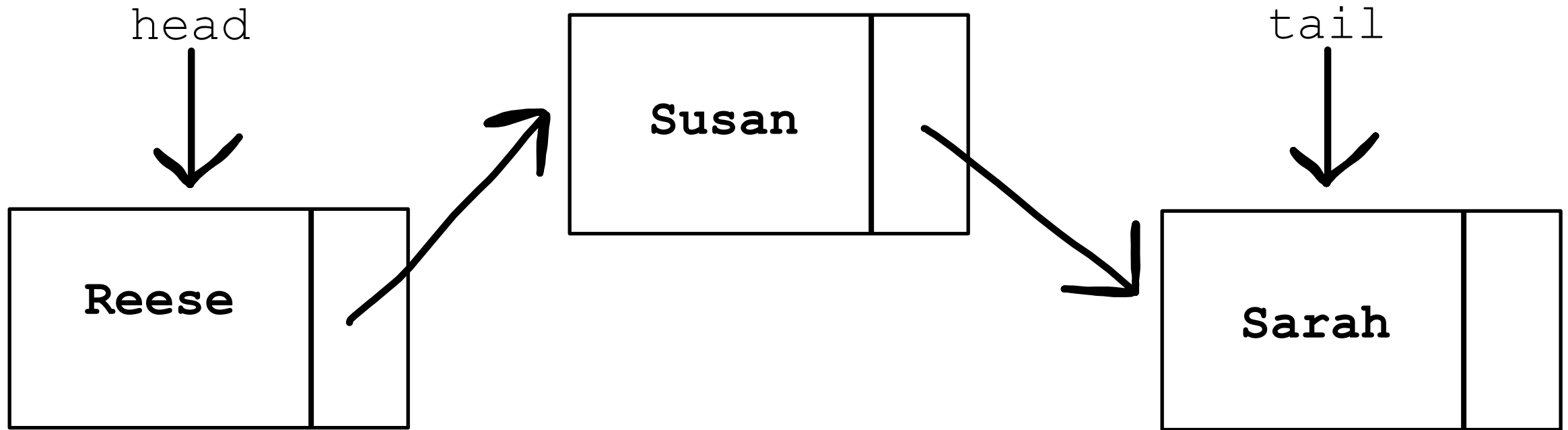


A **Singly Linked List** only keeps track of the next node



The `tail` of a linked list is a pointer to the last node

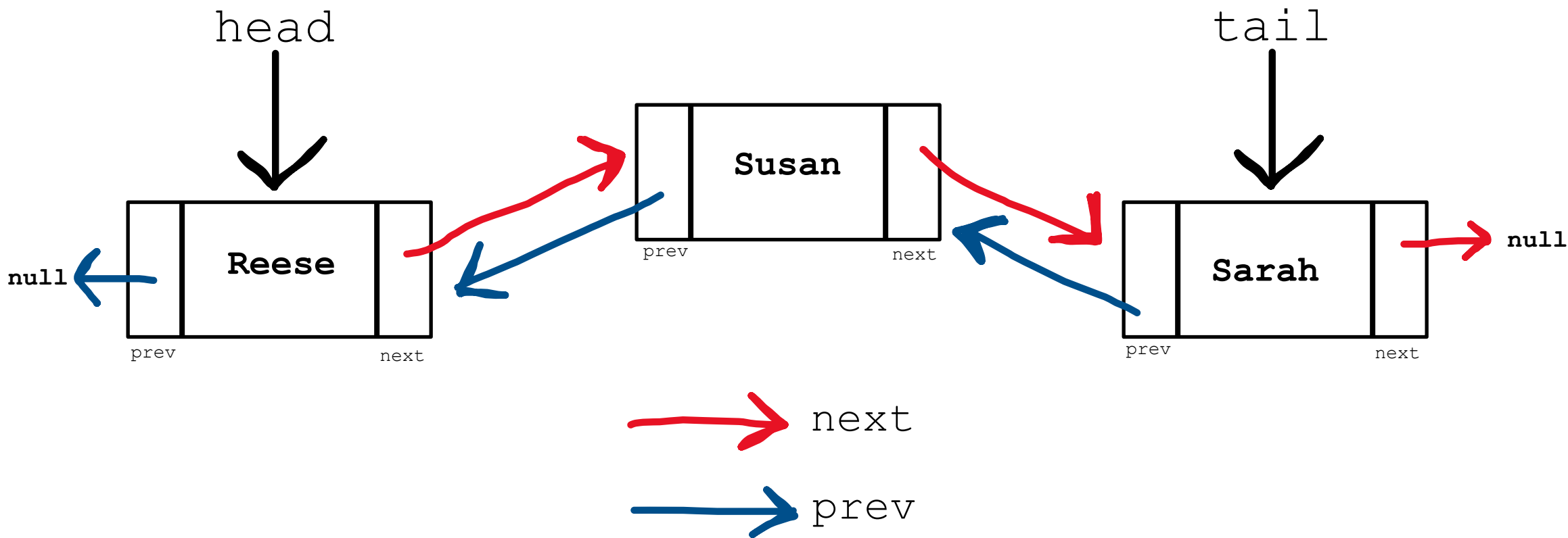
A **Singly Linked List** only keeps track of the next node



The `tail` of a linked list is a pointer to the last node

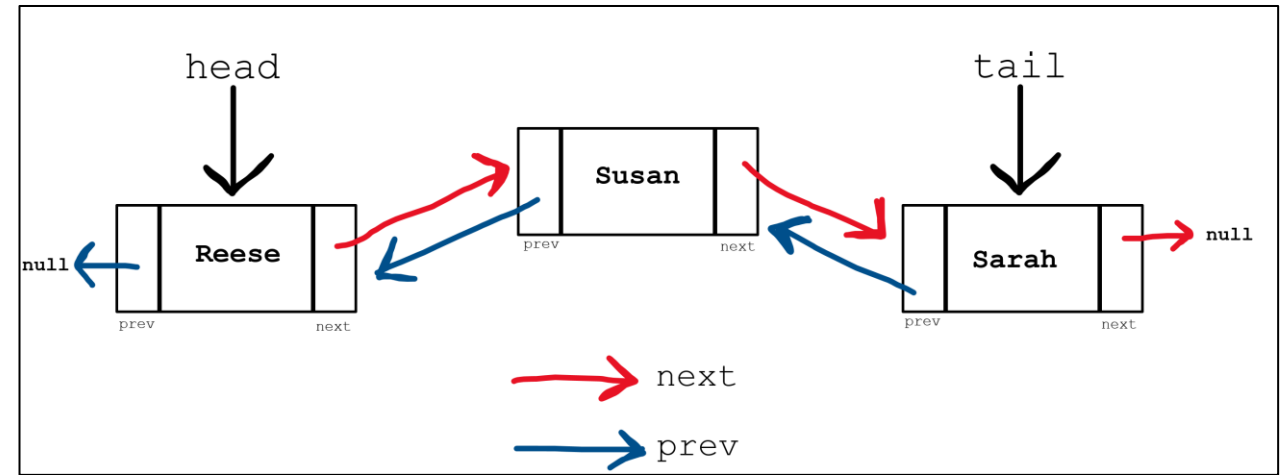
This makes adding to/removing from the end of a linked list easier

A **Doubly Linked List** keeps track of the next node and the previous node



Doubly Linked List Methods

- `insert(newNode, N)` — Insert new node at spot N
- `remove(name)` — Remove node by name
- `remove(N)` — Remove node by Spot #
- `printReverse()` — Prints LL in reverse order



Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

LAX, Los Angeles

SEA, Seattle

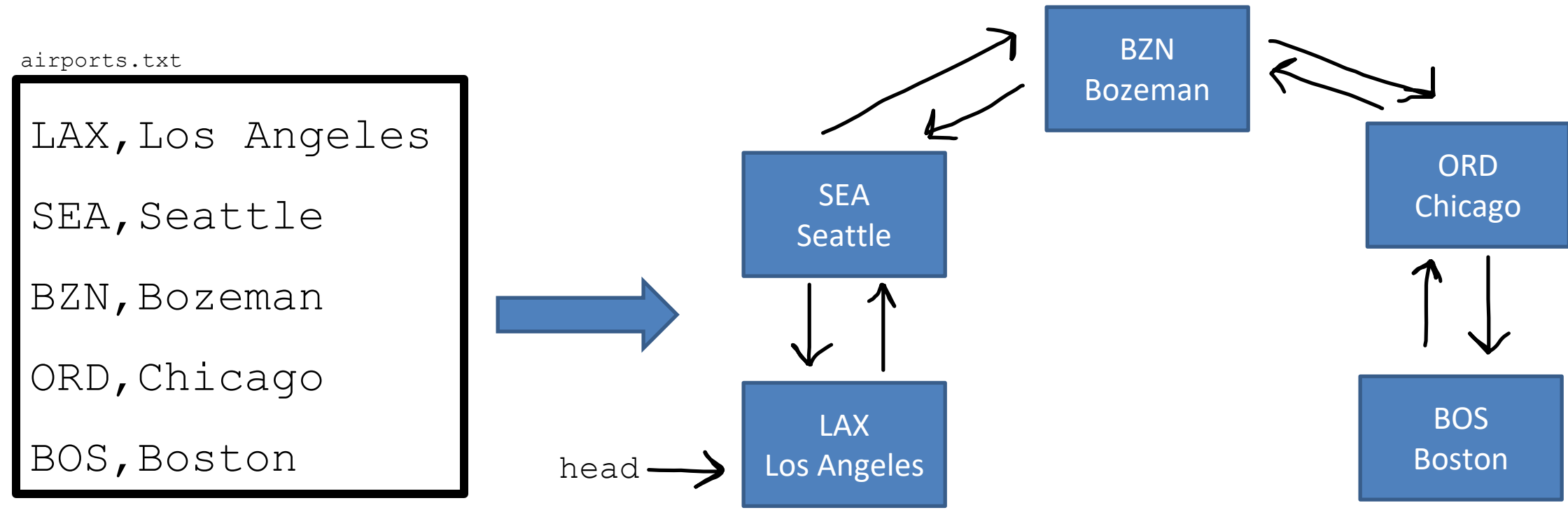
BZN, Bozeman

ORD, Chicago

BOS, Boston

Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library



Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

```
LAX, Los Angeles  
SEA, Seattle  
BZN, Bozeman  
ORD, Chicago  
BOS, Boston
```

1. Iterate through each line of the file

```
BufferedReader br = new BufferedReader(new FileReader(filename));  
String line = "";  
while( (line=br.readLine()) != null){  
  
}
```

Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

```
LAX, Los Angeles  
SEA, Seattle  
BZN, Bozeman  
ORD, Chicago  
BOS, Boston
```

1. Iterate through each line of the file

```
BufferedReader br = new BufferedReader(new FileReader(filename));  
String line = "";  
while( (line=br.readLine()) != null){  
  
}
```

“Iterate through each line in the file until we reach the end”

Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

```
LAX, Los Angeles  
SEA, Seattle  
BZN, Bozeman  
ORD, Chicago  
BOS, Boston
```

1. Iterate through each line of the file
2. Parse each line using `.split()`

```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");
```

"LAX, Los Angeles" → vals =

0	1
LAX	Los Angeles

`.split(",")` will "split" the string everything it sees a comma, returns an array of the splitted string

Let's read in node information **from a file**

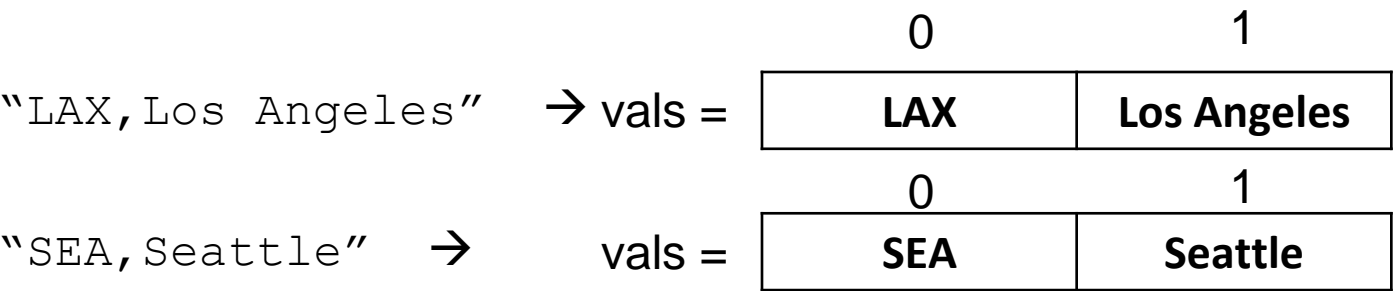
There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

LAX, Los Angeles
SEA, Seattle
BZN, Bozeman
ORD, Chicago
BOS, Boston

- 1. Iterate through each line of the file
- 2. Parse each line using `.split()`

```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");
```



Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

```
LAX, Los Angeles  
SEA, Seattle  
BZN, Bozeman  
ORD, Chicago  
BOS, Boston
```

1. Iterate through each line of the file
2. Parse each line using `.split()`
3. Create Node object using information from file

```
1 while( (line=br.readLine()) != null){  
    2 String[] vals = line.split(",");  
    3 {  
        String code = vals[0];  
        String location = vals[1];  
        Node n = new Node(code, location);  
        insert(n, size+1);  
    }  
}
```

Let's read in node information **from a file**

There are tons of way to read from a file in Java. We will use the BufferedReader library

airports.txt

```
LAX, Los Angeles  
SEA, Seattle  
BZN, Bozeman  
ORD, Chicago  
BOS, Boston
```

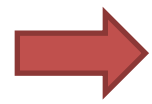
1. Iterate through each line of the file

2. Parse each line using `.split()`

3. Create Node object using information from file

4. Insert new node at the end of the linked list

```
1 while( (line=br.readLine()) != null){  
    2 String[] vals = line.split(",");  
    3 {  
        String code = vals[0];  
        String location = vals[1];  
    4 Node n = new Node(code, location);  
        insert(n, size+1);  
    }  
}
```



```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");  
  
    String code = vals[0];  
    String location = vals[1];  
  
    Node n = new Node(code, location);  
    insert(n,size+1);  
}
```

airports.txt

LAX, Los Angeles

SEA, Seattle

BZN, Bozeman

ORD, Chicago

BOS, Boston

```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");  
  
    String code = vals[0];  
    String location = vals[1];  
  
    Node n = new Node(code, location);  
    insert(n,size+1);  
}
```

airports.txt

LAX, Los Angeles

SEA, Seattle

BZN, Bozeman

ORD, Chicago

BOS, Boston

line = "LAX, Los Angeles"

```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");  
  
    String code = vals[0];  
    String location = vals[1];  
  
    Node n = new Node(code, location);  
    insert(n,size+1);  
}
```

airports.txt

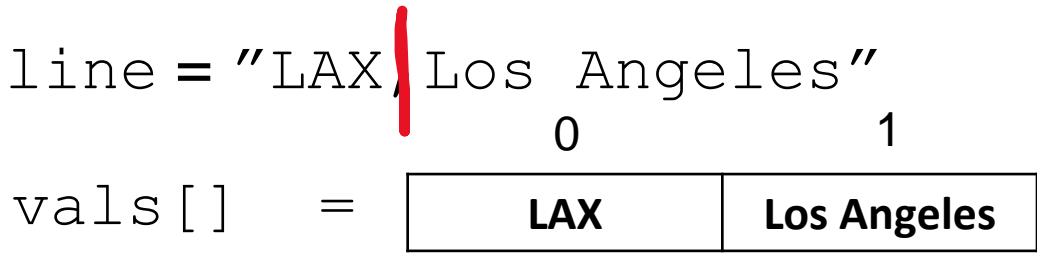
LAX, Los Angeles

SEA, Seattle

BZN, Bozeman

ORD, Chicago

BOS, Boston



```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");  
  
    String code = vals[0];  
    String location = vals[1];  
  
    Node n = new Node(code, location);  
    insert(n,size+1);  
}
```

airports.txt

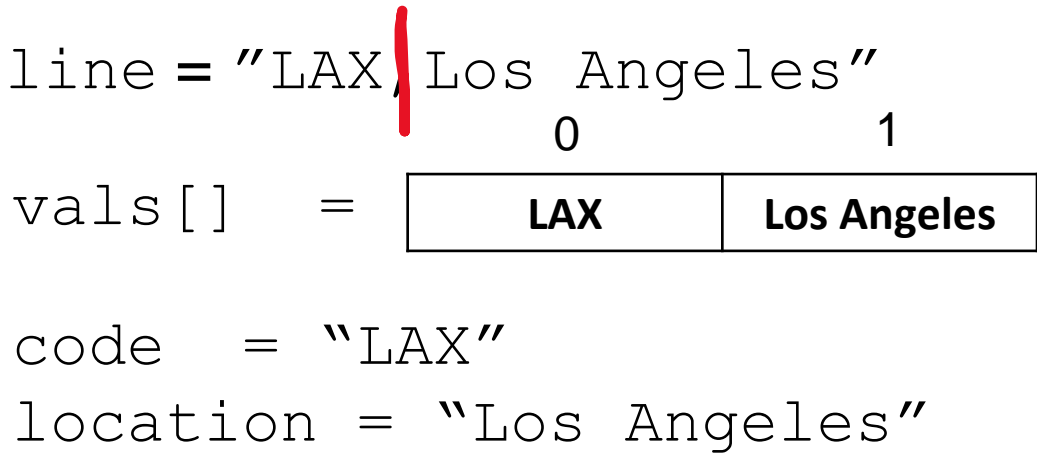
LAX, Los Angeles

SEA, Seattle

BZN, Bozeman

ORD, Chicago

BOS, Boston



airports.txt

LAX, Los Angeles

SEA, Seattle

BZN, Bozeman

ORD, Chicago

BOS, Boston

```
while( (line=br.readLine()) != null){  
    String[] vals = line.split(",");
```



```
String code = vals[0];  
String location = vals[1];
```

```
Node n = new Node(code, location);  
insert(n, size+1);
```

```
}
```

```
line = "LAX, Los Angeles"  
           0           1
```

vals[]	=	LAX	Los Angeles
--------	---	-----	-------------

```
code = "LAX"
```

```
location = "Los Angeles"
```

airports.txt

LAX, Los Angeles

SEA, Seattle


BZN, Bozeman

ORD, Chicago

BOS, Boston

```
while( (line=br.readLine()) != null){
    String[] vals = line.split(",");
```

```
String code = vals[0];  
String location = vals[1];
```



```
Node n = new Node(code, location);
insert(n, size+1);
}
```

```
line = "LAX, Los Angeles"
```

vals[]	=	<table><tr><td>LAX</td><td>Los Angeles</td></tr></table>	LAX	Los Angeles
LAX	Los Angeles			

```
code    = "LAX"
location = "Los Angeles"
```

$$n =$$

LAX
Los Angeles


```
BufferedReader br = new BufferedReader(new FileReader(filename));
String line = "";
while( (line=br.readLine()) != null){
    String[] vals = line.split(",");

    String code = vals[0];
    String location = vals[1];

    Node n = new Node(code, location);
    insert(n,size+1);
}
```

- **insert(newNode, N)** — Insert new node (newNode) at spot N

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

Case 2: The user is inserting a node at the very beginning ($N = 1$)

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

Case 2: The user is inserting a node at the very beginning ($N = 1$)

Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 1: The Linked List is Empty

Case 2: The user is inserting a node at the very beginning ($N = 1$)

Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)

Case 4: The user is inserting a node somewhere in the middle of the LL

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

How do we know if the linked list is empty?

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

How do we know if the linked list is empty?

If the head and tail are null

If the size is 0

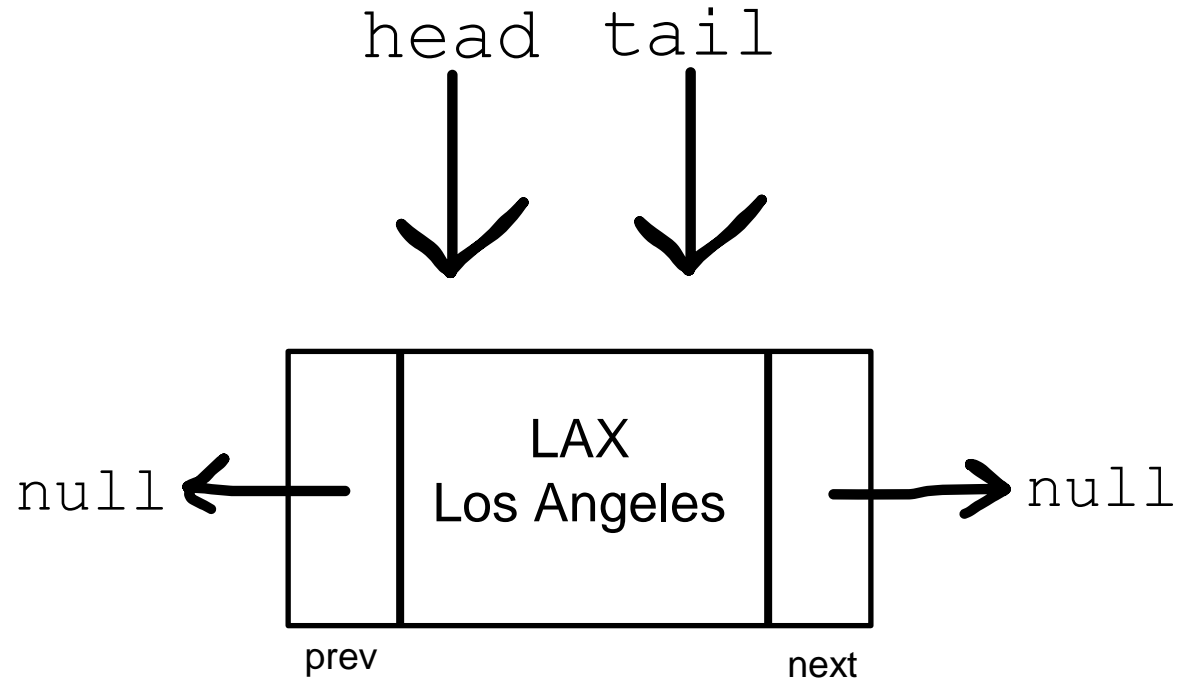
- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

Case 1: The Linked List is Empty

???

- **insert(newNode, N)** — Insert new node (newNode) at spot **N**

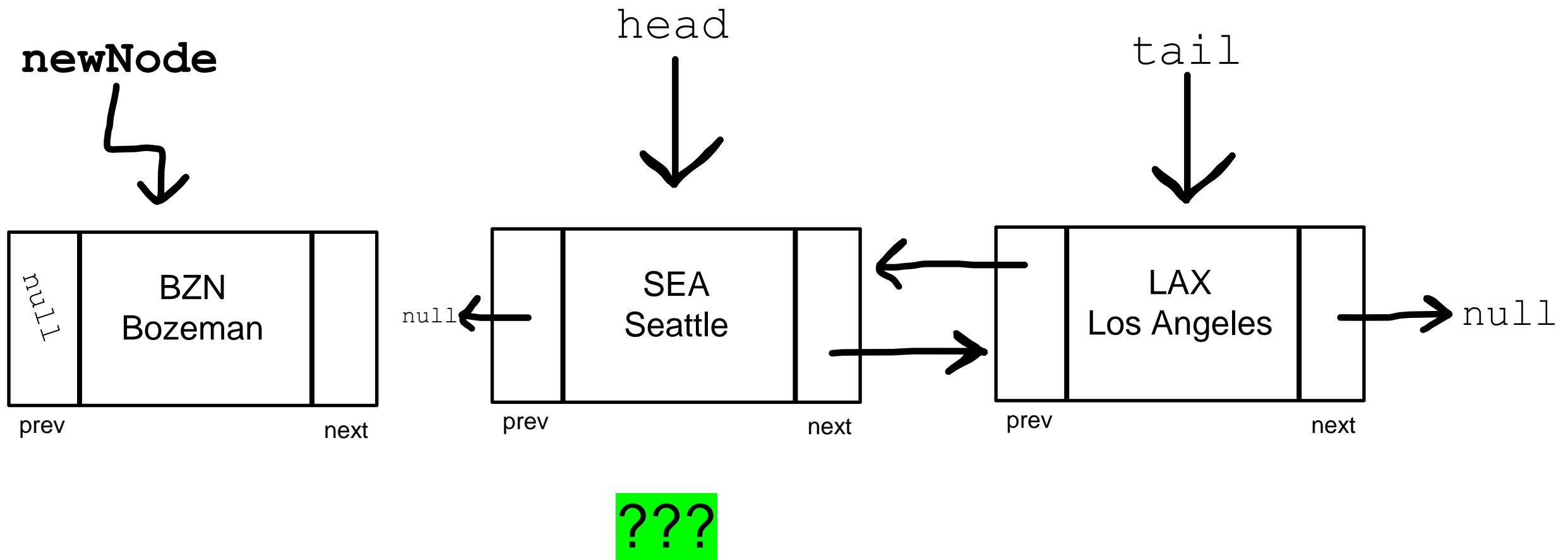
Case 1: The Linked List is Empty



Set the `tail` and `head` to be the `newNode`

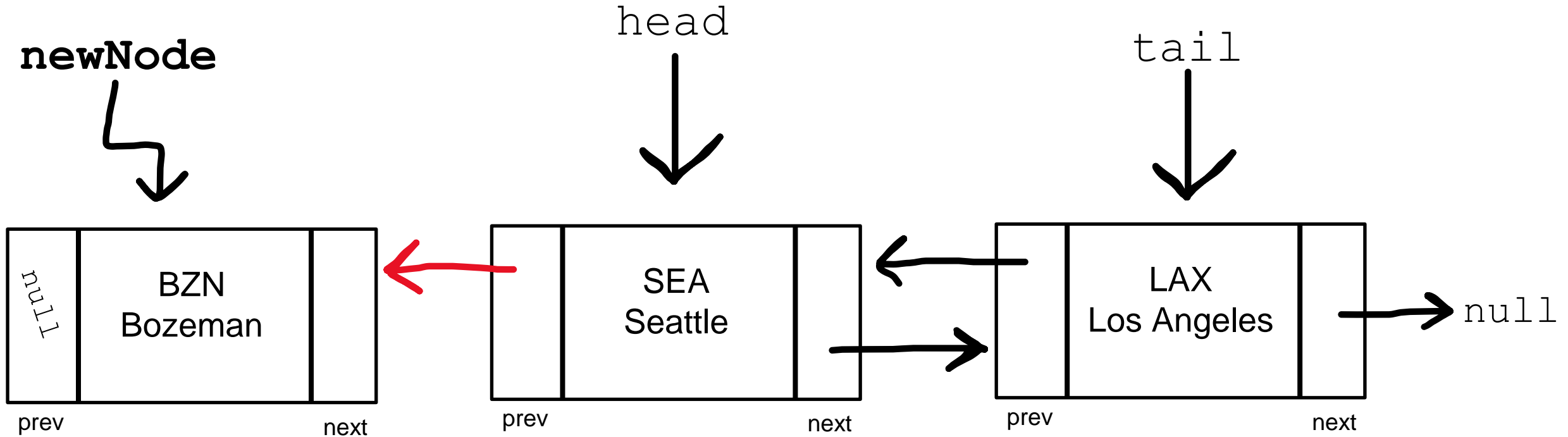
- insert(newNode, N)** — Insert new node (newNode) at spot N

Case 2: The user is inserting a node at the very beginning (N = 1)



- **insert(newNode, N)** — Insert new node (newNode) at spot N

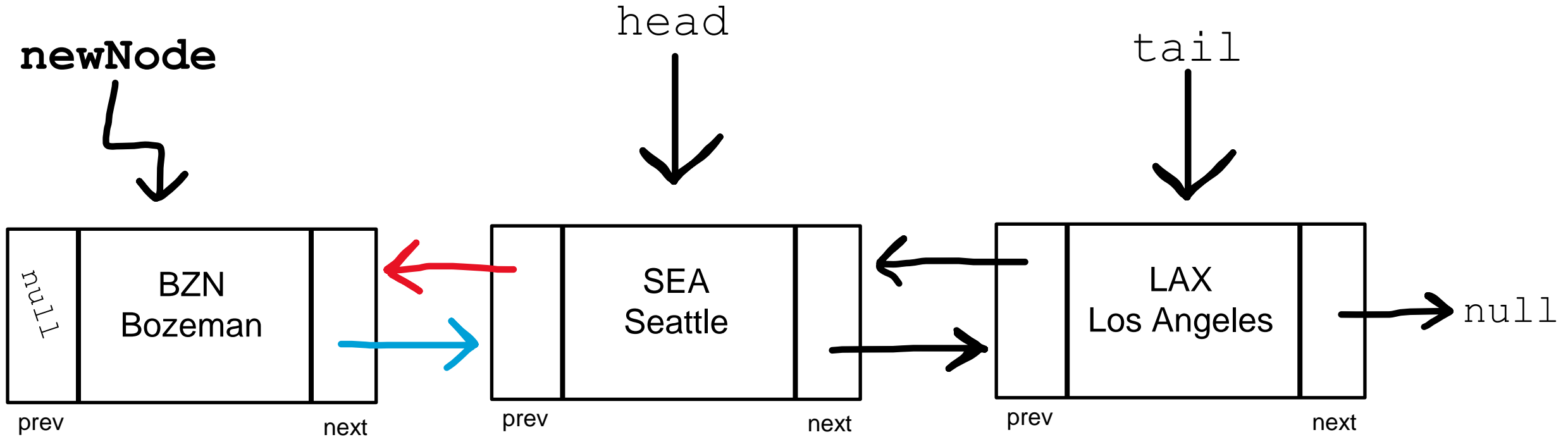
Case 2: The user is inserting a node at the very beginning (N = 1)



Update the head node prev value to newNode

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 2: The user is inserting a node at the very beginning (N = 1)

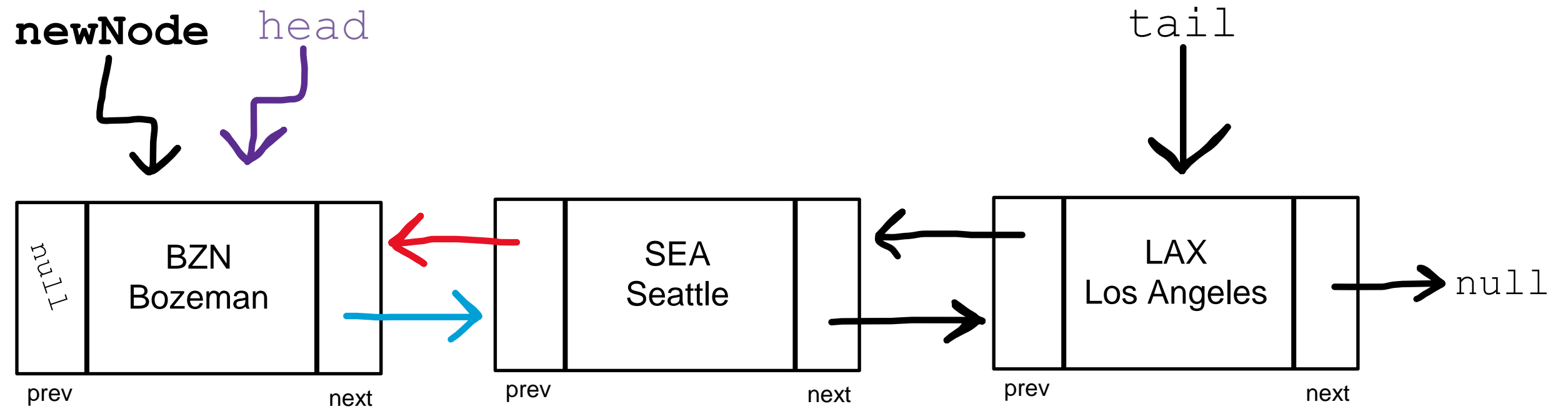


Update the head node prev value to newNode

Update the newNode's next value to be the current head node

- `insert(newNode, N)` — Insert new node (`newNode`) at spot `N`

Case 2: The user is inserting a node at the very beginning (`N = 1`)



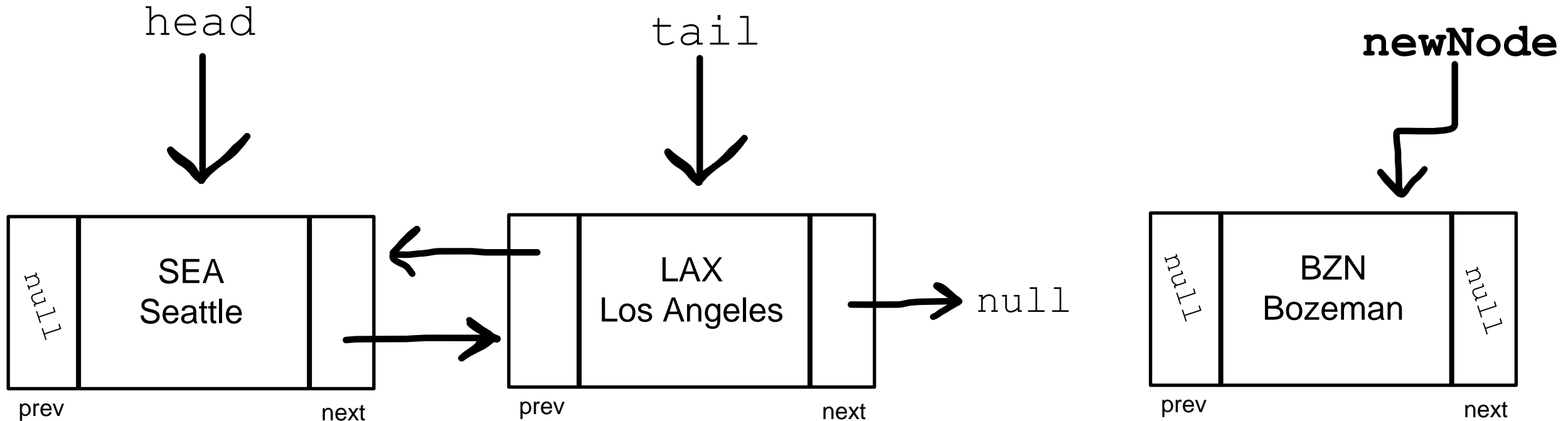
Update the head node prev value to newNode

Update the newNode's next value to be the current head node

Update the head node to be the newNode

- **insert(newNode, N)** — Insert new node (newNode) at spot N

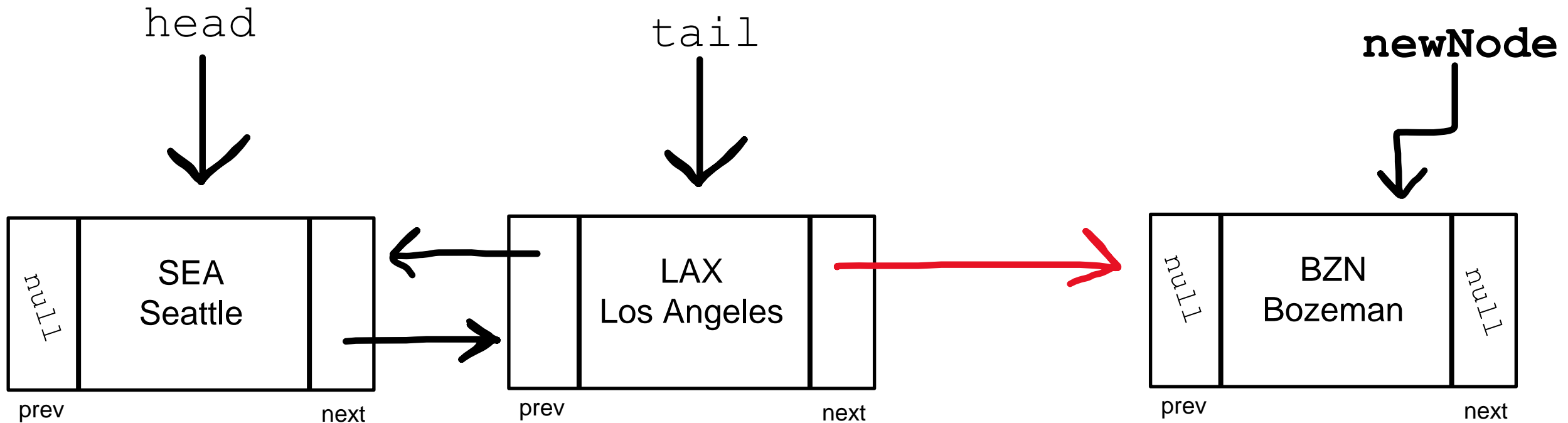
Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)



`insert(newNode, 3)`

- **insert(newNode, N)** — Insert new node (newNode) at spot N

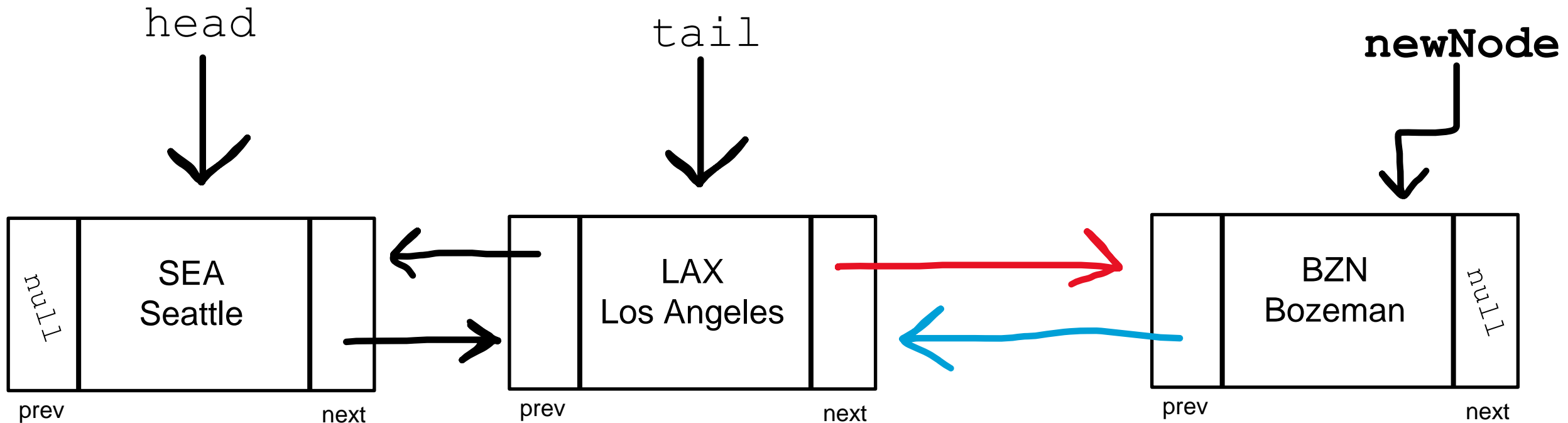
Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)



Update the tail node next value to newNode

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)

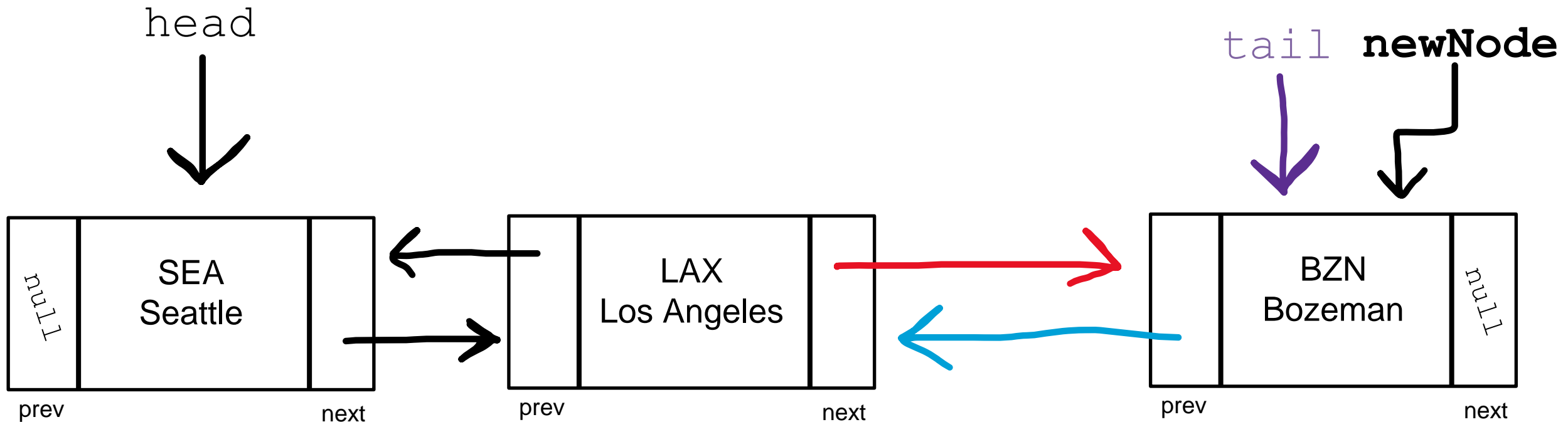


Update the **tail node next value** to newNode

Update the **newNode's prev value** to be the current **tail node**

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)



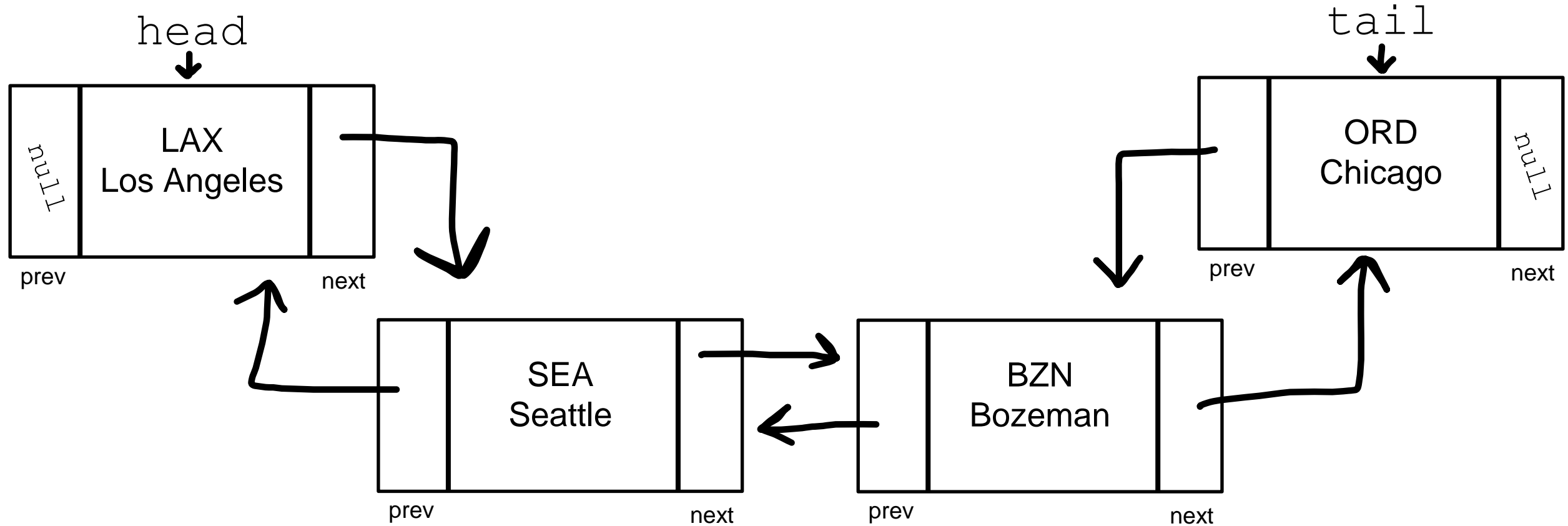
Update the **tail node** next value to newNode

Update the newNode's prev value to be the current tail node

Update the tail node to be the newNode

- **insert(newNode, N)** — Insert new node (newNode) at spot N

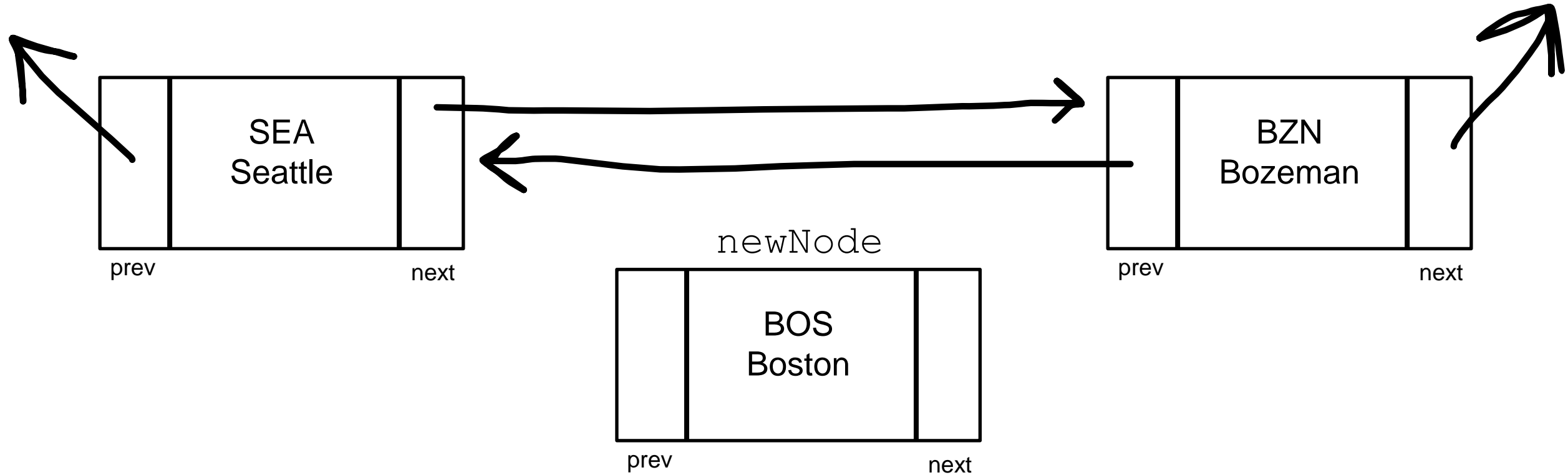
Case 4: The user is inserting a node somewhere in the middle of the LL



`insert(newNode, 3)`

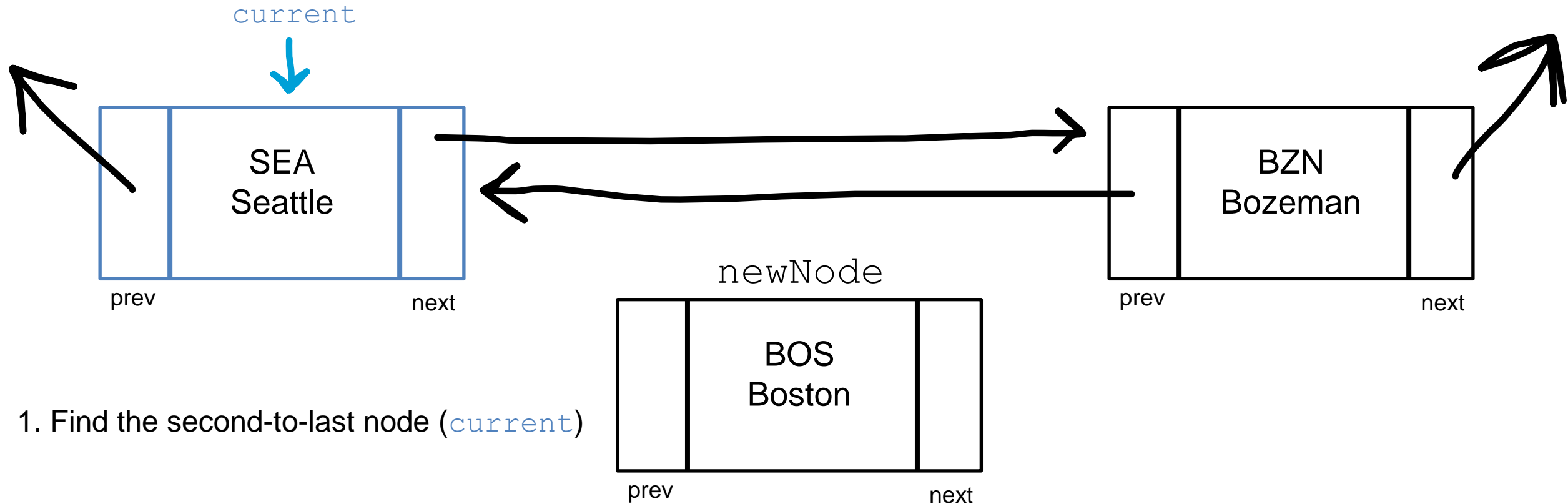
- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL



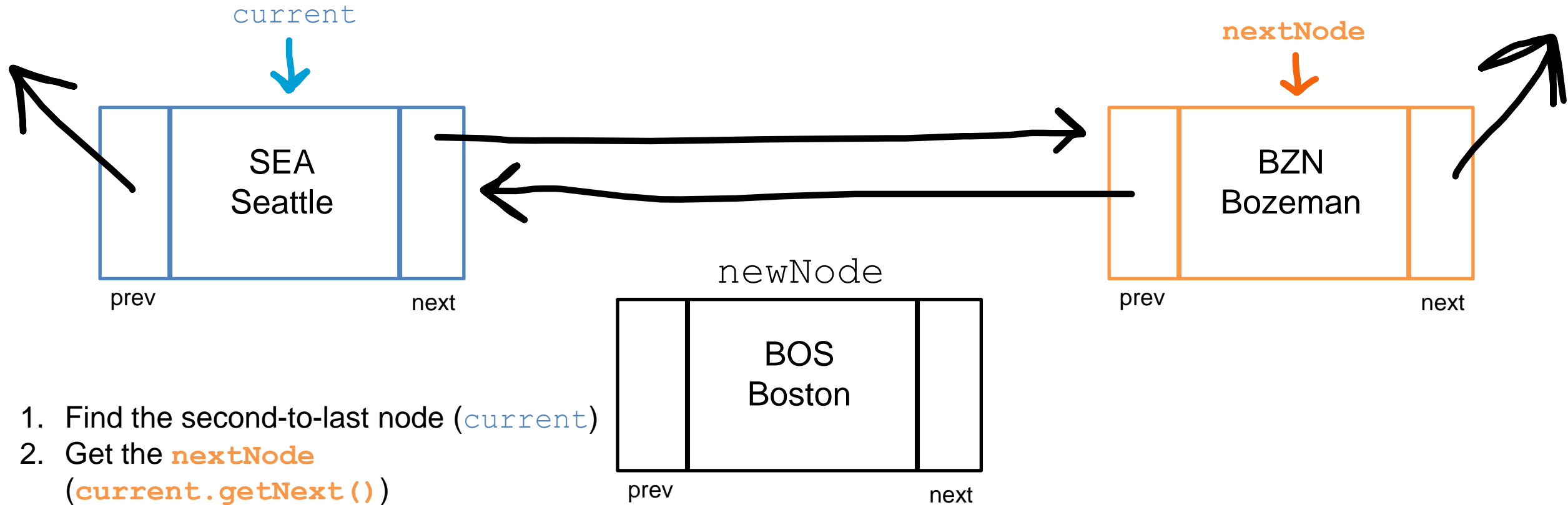
- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL



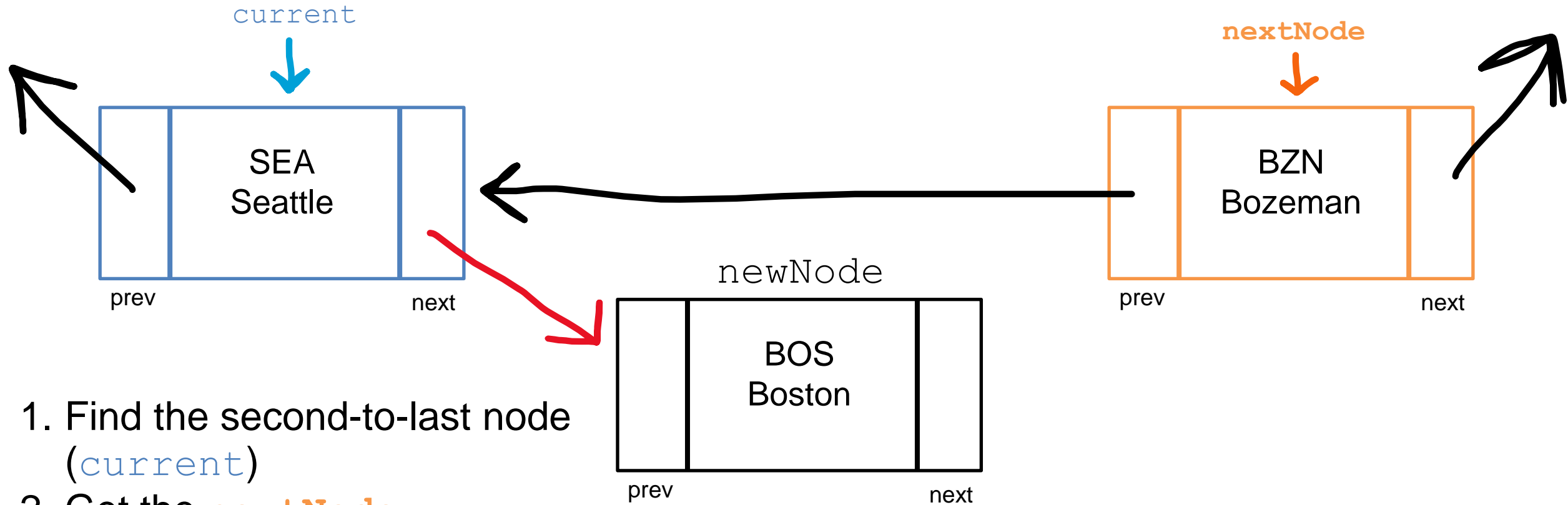
- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL



- **insert(newNode, N)** — Insert new node (newNode) at spot N

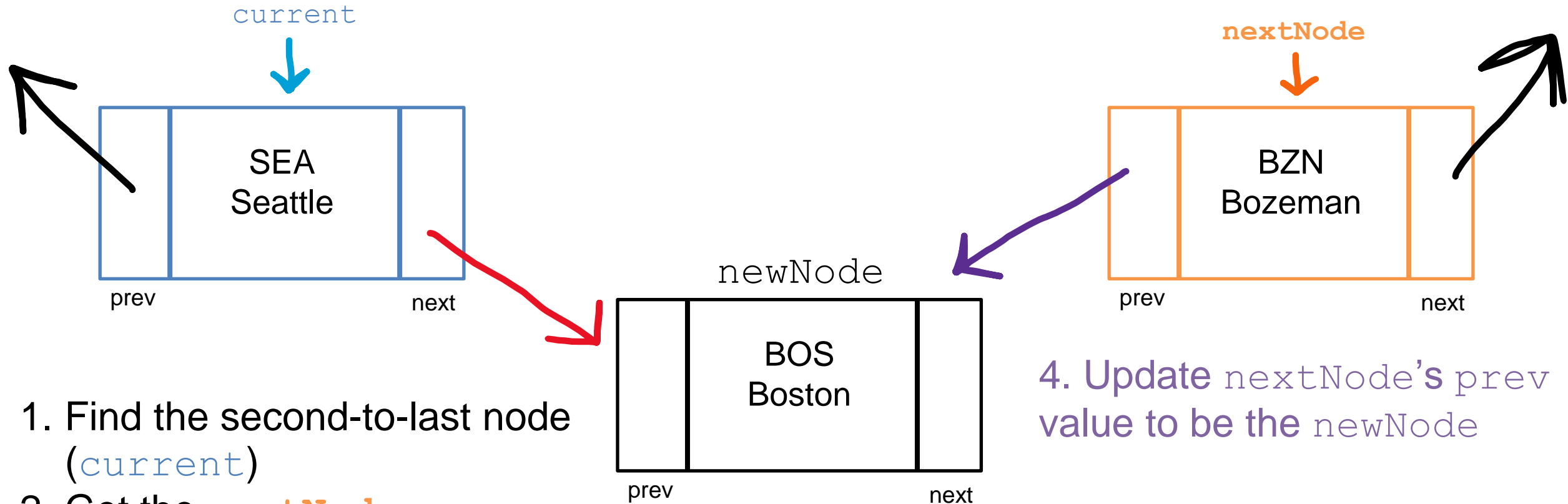
Case 4: The user is inserting a node somewhere in the middle of the LL



1. Find the second-to-last node (`current`)
2. Get the `nextNode` (`current.getNext()`)
3. Update `current's` next value to the `newNode`

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL

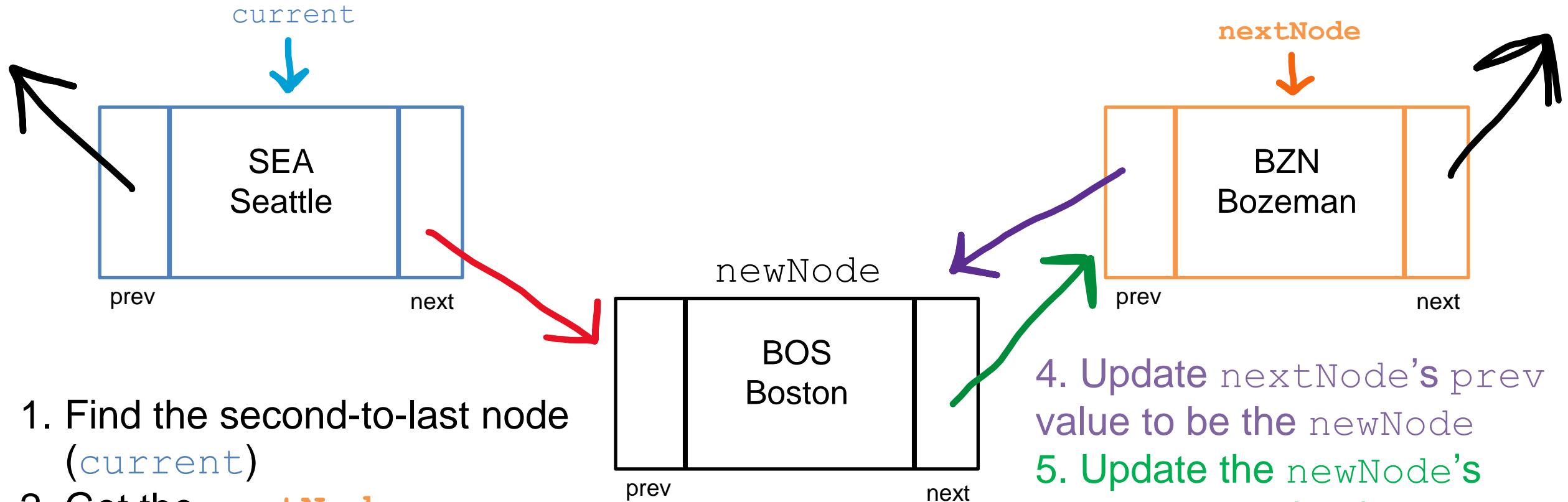


1. Find the second-to-last node (`current`)
2. Get the `nextNode` (`current.getNext()`)
3. Update `current's` next value to the `newNode`

4. Update `nextNode's` prev value to be the `newNode`

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL

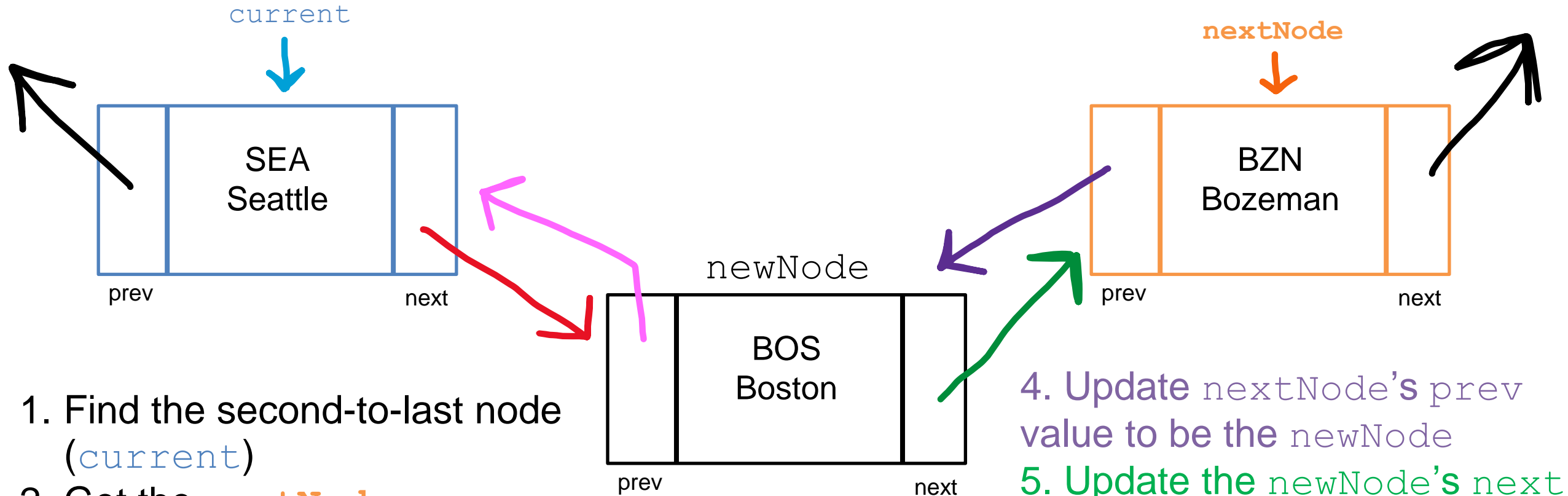


1. Find the second-to-last node (`current`)
2. Get the `nextNode` (`current.getNext()`)
3. Update `current`'s next value to the `newNode`

4. Update `nextNode`'s prev value to be the `newNode`
5. Update the `newNode`'s next value to be the `nextNode`

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL

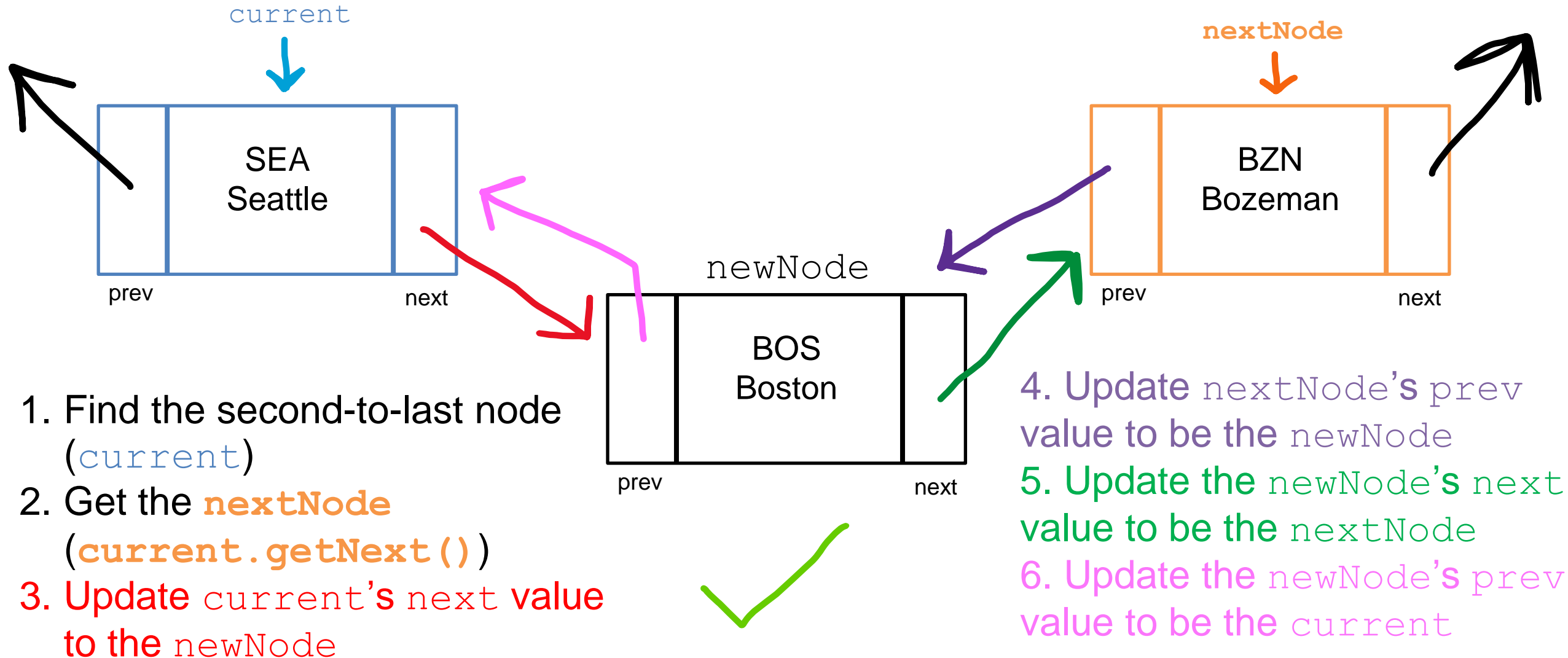


1. Find the second-to-last node (`current`)
2. Get the `nextNode` (`current.getNext()`)
3. Update `current's` next value to the `newNode`

4. Update `nextNode's` prev value to be the `newNode`
5. Update the `newNode's` next value to be the `nextNode`
6. Update the `newNode's` prev value to be the `current`

- **insert(newNode, N)** — Insert new node (newNode) at spot N

Case 4: The user is inserting a node somewhere in the middle of the LL



- **insert(newNode, N)** — Insert new node (newNode) at spot N `public void insert(Node newNode, int n) {`

Case 1: The Linked List is Empty

```
//Case #1 Linked List is empty
if(this.size == 0) {
    this.head = newNode;
    this.tail = newNode;
}
```

Case 2: The user is inserting a node at the very beginning (N = 1)

```
//Case #2 Insert at the beginning
else if(n == 1) {
    this.head.setPrev(newNode);
    newNode.setNext(this.head);
    this.head = newNode;
}
```

- **insert(newNode, N)** — Insert new node (newNode) at spot N `public void insert(Node newNode, int n) {`

Case 3: The user is inserting a node at the very end ($N = \text{getSize}() + 1$)

```
//Case #3 Insert at the end
else if(n == this.size+1) {

    this.tail.setNext(newNode);
    newNode.setPrev(this.tail);
    this.tail = newNode;

}
..
```

Case 4: The user is inserting a node somewhere in the middle of the LL

```
//Case #4 Insert in the middle
else {
    int counter = 1;
    Node current = this.head;
    while(current.getNext() != null) {

        if(counter == n-1) {

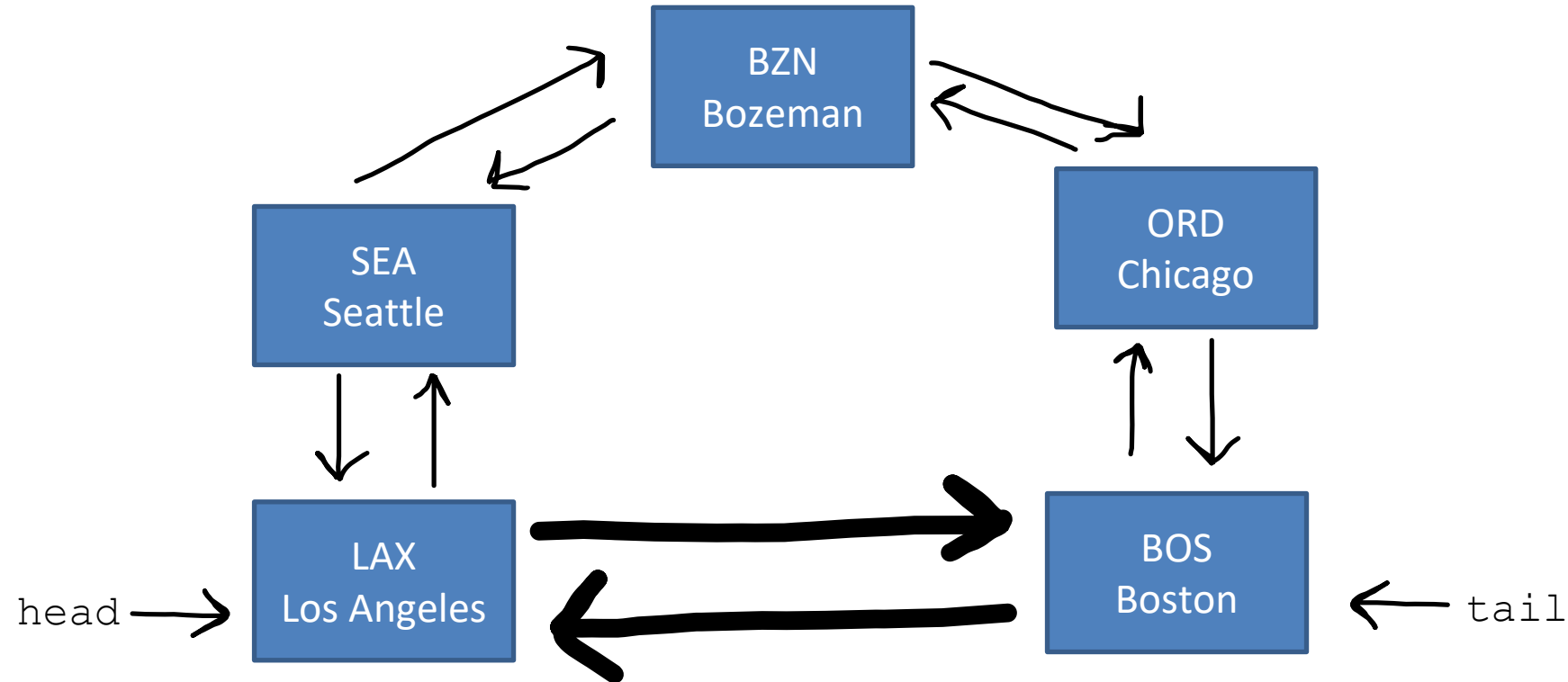
            Node nextNode = current.getNext();

            current.setNext(newNode);
            newNode.setPrev(current);

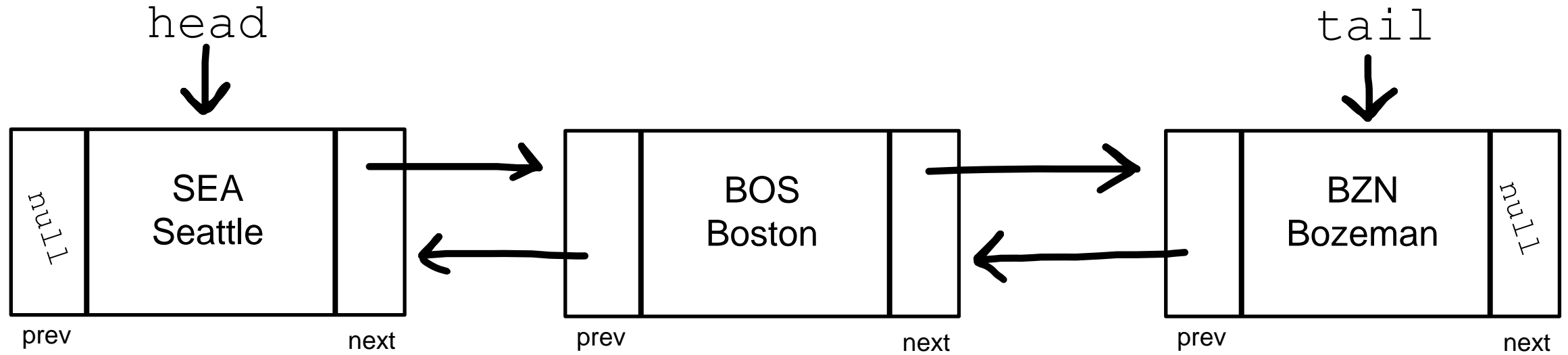
            nextNode.setPrev(newNode);
            newNode.setNext(nextNode);

        }
        current = current.getNext();
        counter++;
    }
}
```

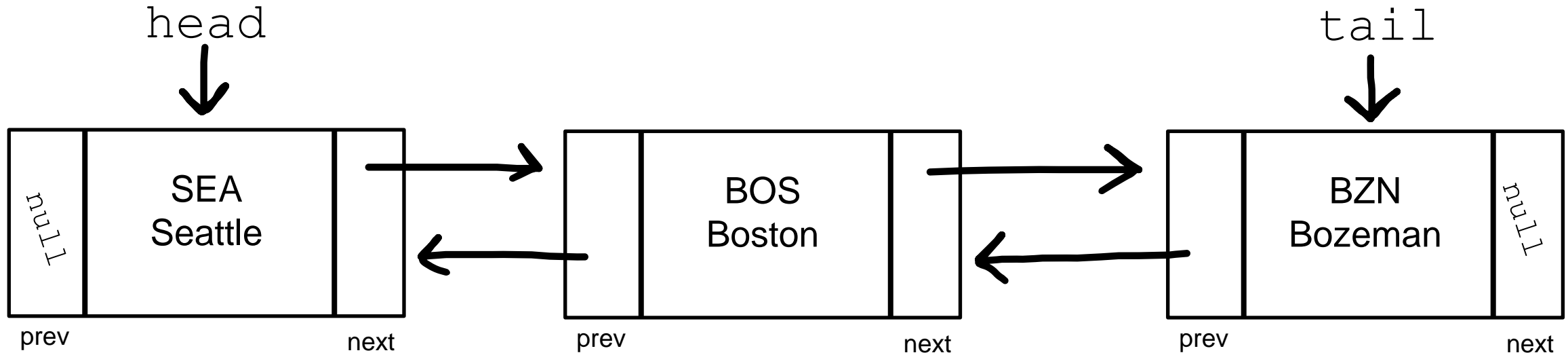
A **Circular Linked List** is a linked list where the first and last node are connected, which creates a circle



- `remove(name)` — Remove node by name

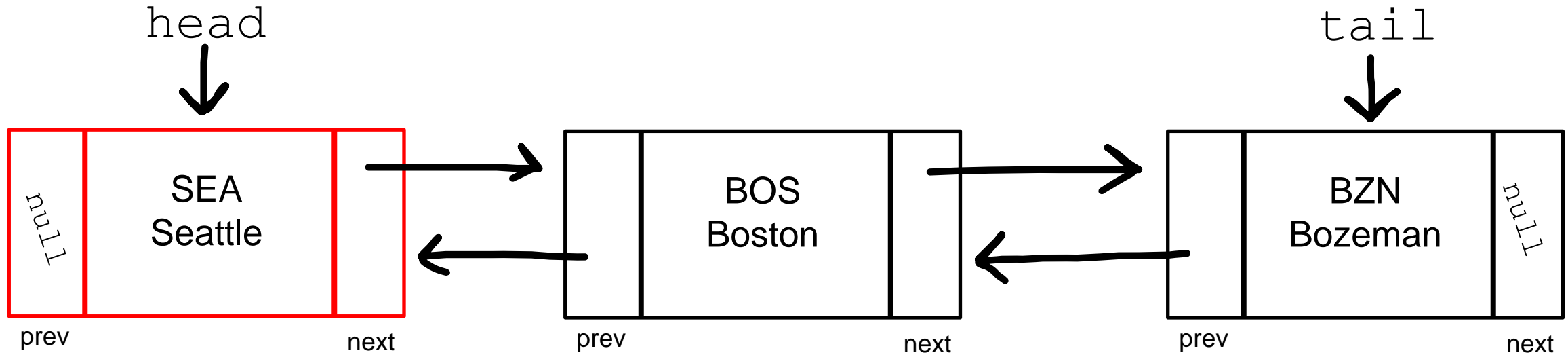


- `remove(name)` — Remove node by name



1. Traverse the Linked List and look for a match

- `remove(name)` — Remove node by name

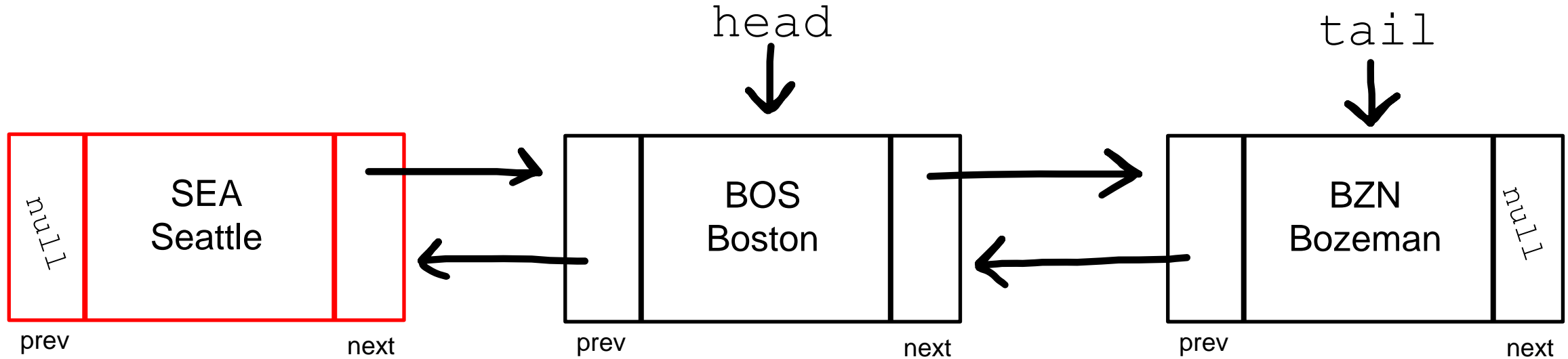


1. Traverse the Linked List and look for a match

`remove("SEA")`

What if the removed node is the head?

- `remove(name)` — Remove node by name



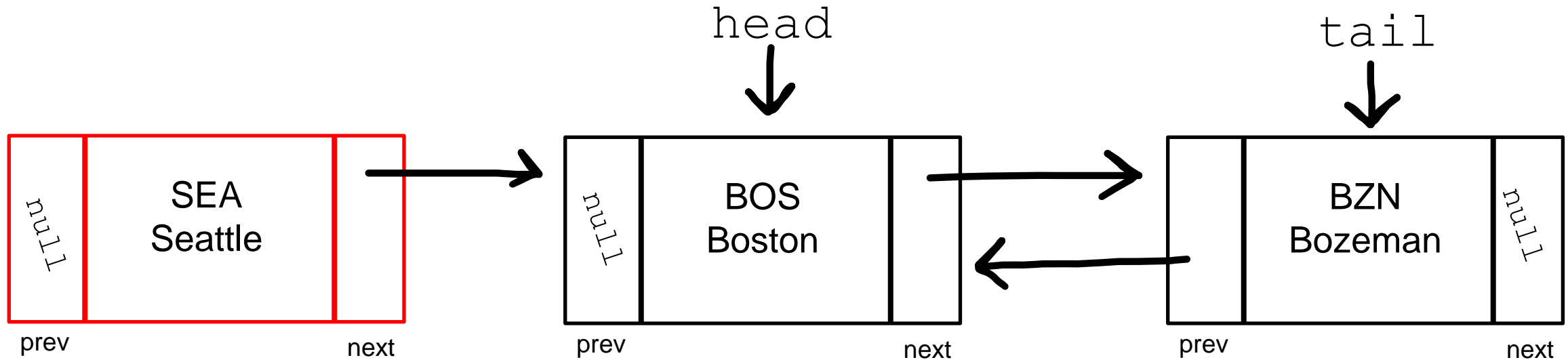
1. Traverse the Linked List and look for a match

`remove("SEA")`

What if the removed node is the head?

2. Update the `head` to be the next node

- `remove(name)` — Remove node by name



1. Traverse the Linked List and look for a match

`remove("SEA")`

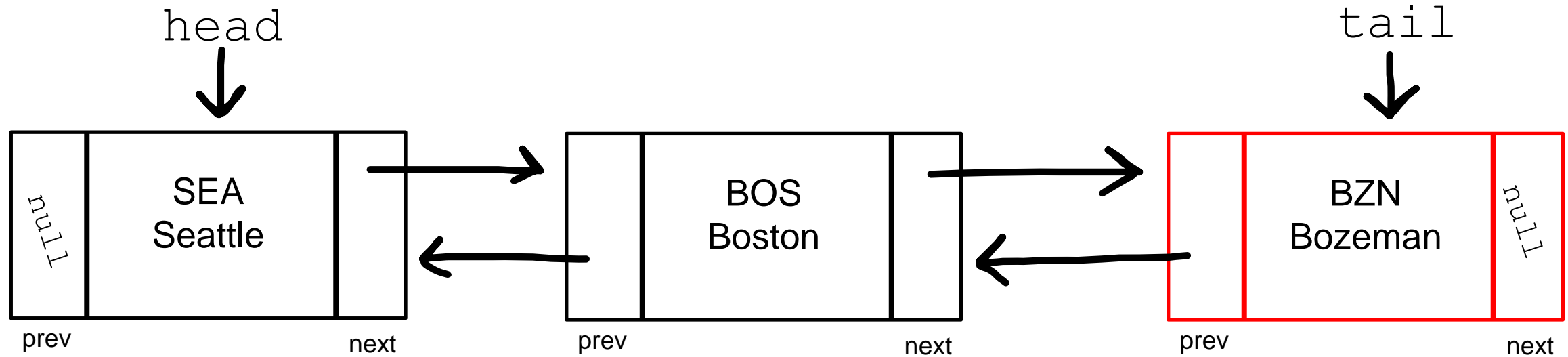
What if the removed node is the head?

2. Update the head to be the next node

3. Update the new head's prev value to be null

We can no longer reach the SEA node from the head node, so it is effectively removed

- `remove(name)` — Remove node by name

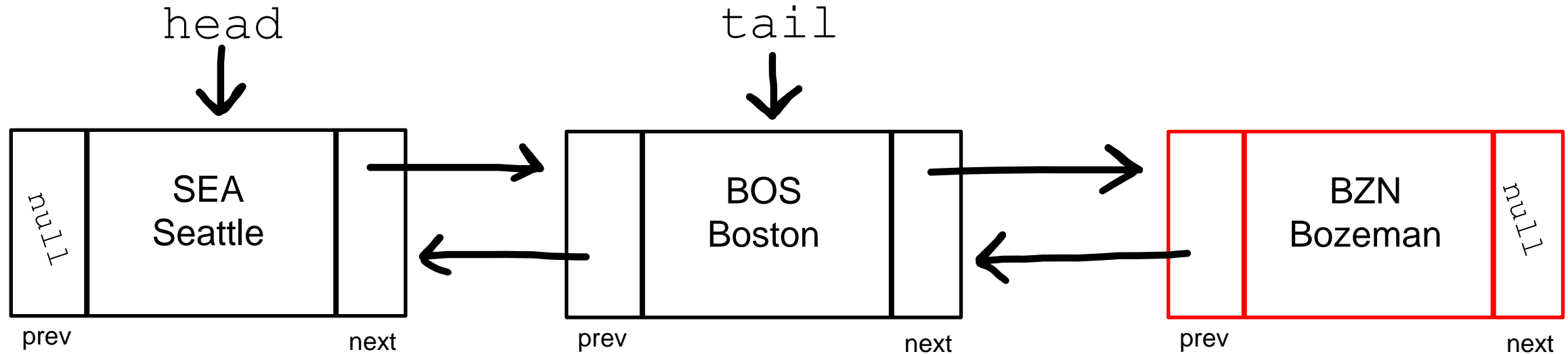


1. Traverse the Linked List and look for a match

`remove("BZN")`

What if the removed node is the tail?

- `remove(name)` — Remove node by name



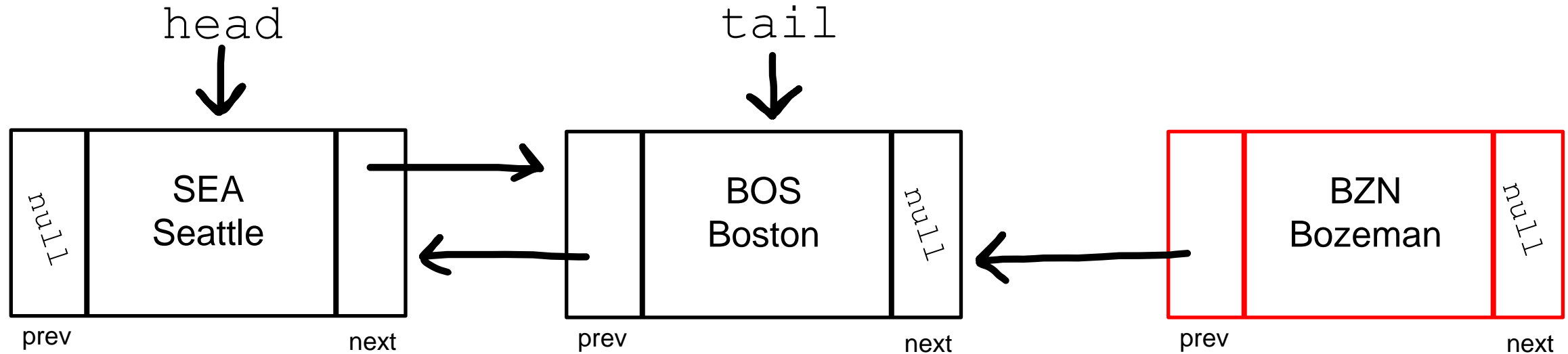
1. Traverse the Linked List and look for a match

`remove("BZN")`

What if the removed node is the tail?

2. Update the `tail` to be the previous node

- `remove(name)` — Remove node by name



1. Traverse the Linked List and look for a match

`remove("BZN")`

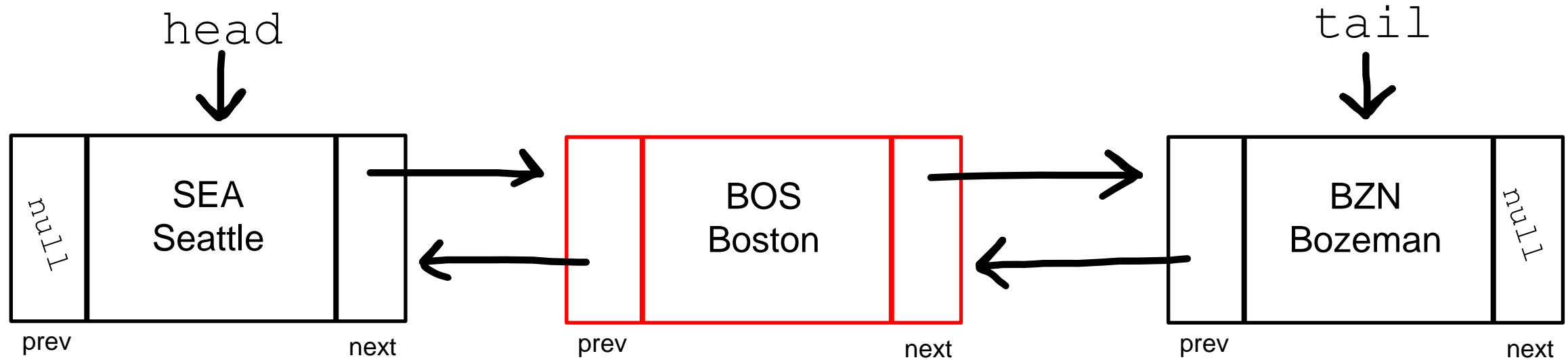
What if the removed node is the tail?

2. Update the `tail` to be the previous node

3. Update the new `tail`'s `next` value to be null

We can no longer reach the BZN node from the head node, so it is effectively removed

- `remove(name)` — Remove node by name

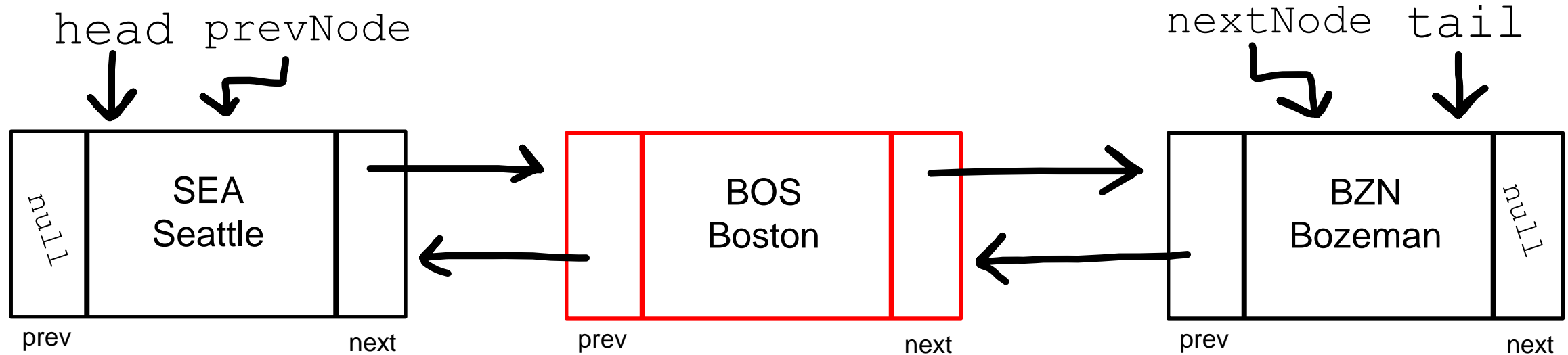


1. Traverse the Linked List and look for a match

`remove("BOS")`

What if the removed node is somewhere in the middle?

- `remove(name)` — Remove node by name



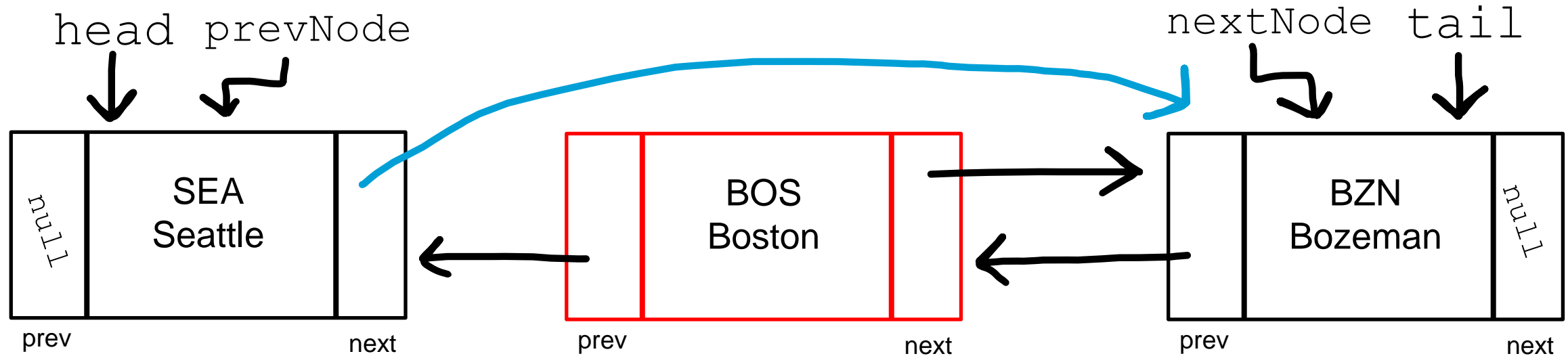
1. Traverse the Linked List and look for a match

`remove("BOS")`

What if the removed node is somewhere in the middle?

2. Retrieve the previous node and next node of the to-be-removed node

- **remove(name)** — Remove node by name



1. Traverse the Linked List and look for a match

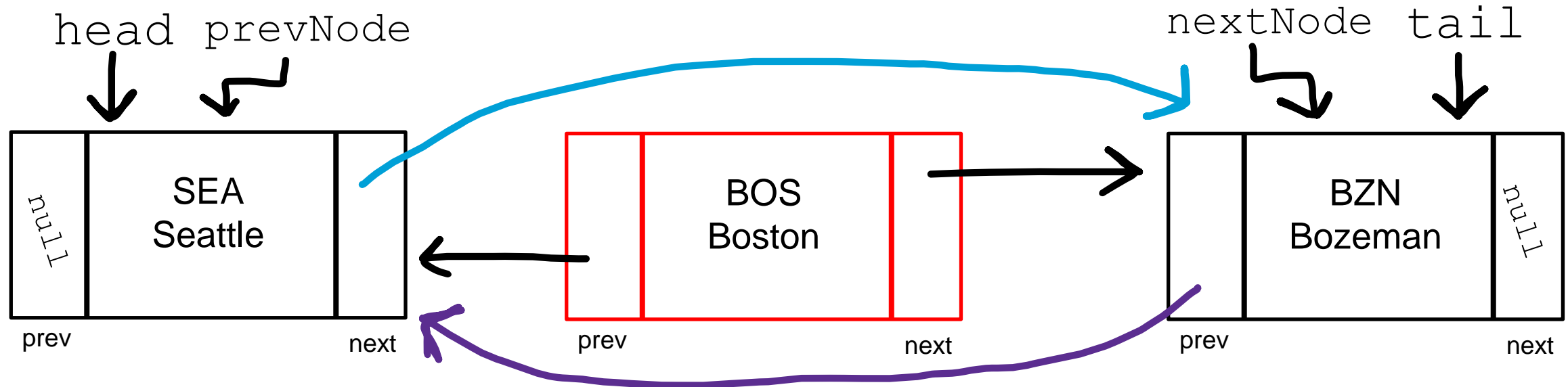
`remove ("BOS")`

What if the removed node is somewhere in the middle?

2. Retrieve the previous node and next node of the to-be-removed node

3. Update `prevNode's next` value to be the `nextNode`

- **remove(name)** — Remove node by name



1. Traverse the Linked List and look for a match

`remove("BOS")`

What if the removed node is somewhere in the middle?

2. Retrieve the previous node and next node of the to-be-removed node

3. Update `prevNode`'s `next` value to be the `nextNode`

4. Update `nextNode`'s `prev` value to be `prevNode`