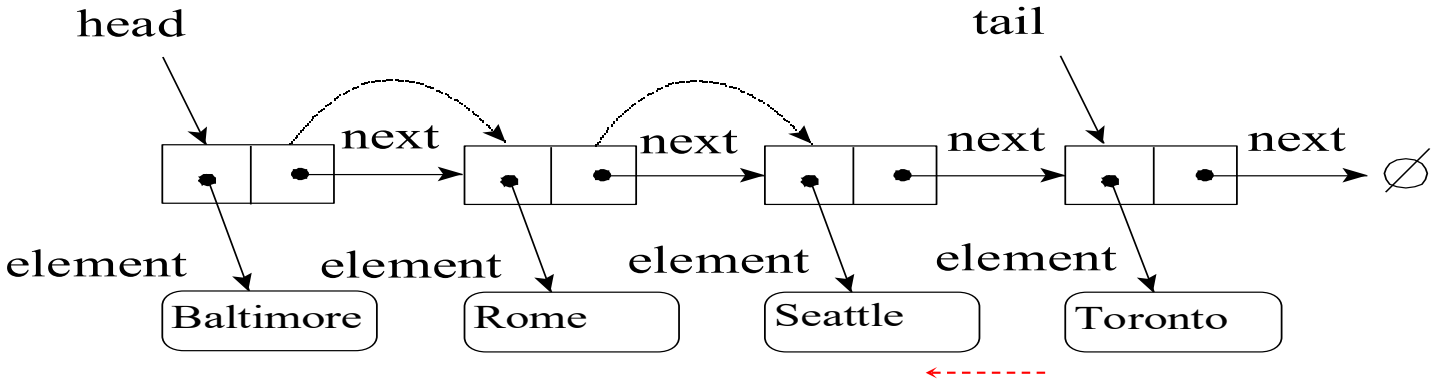
Double Linked Lists

Ye Yang
Stevens Institute of Technology

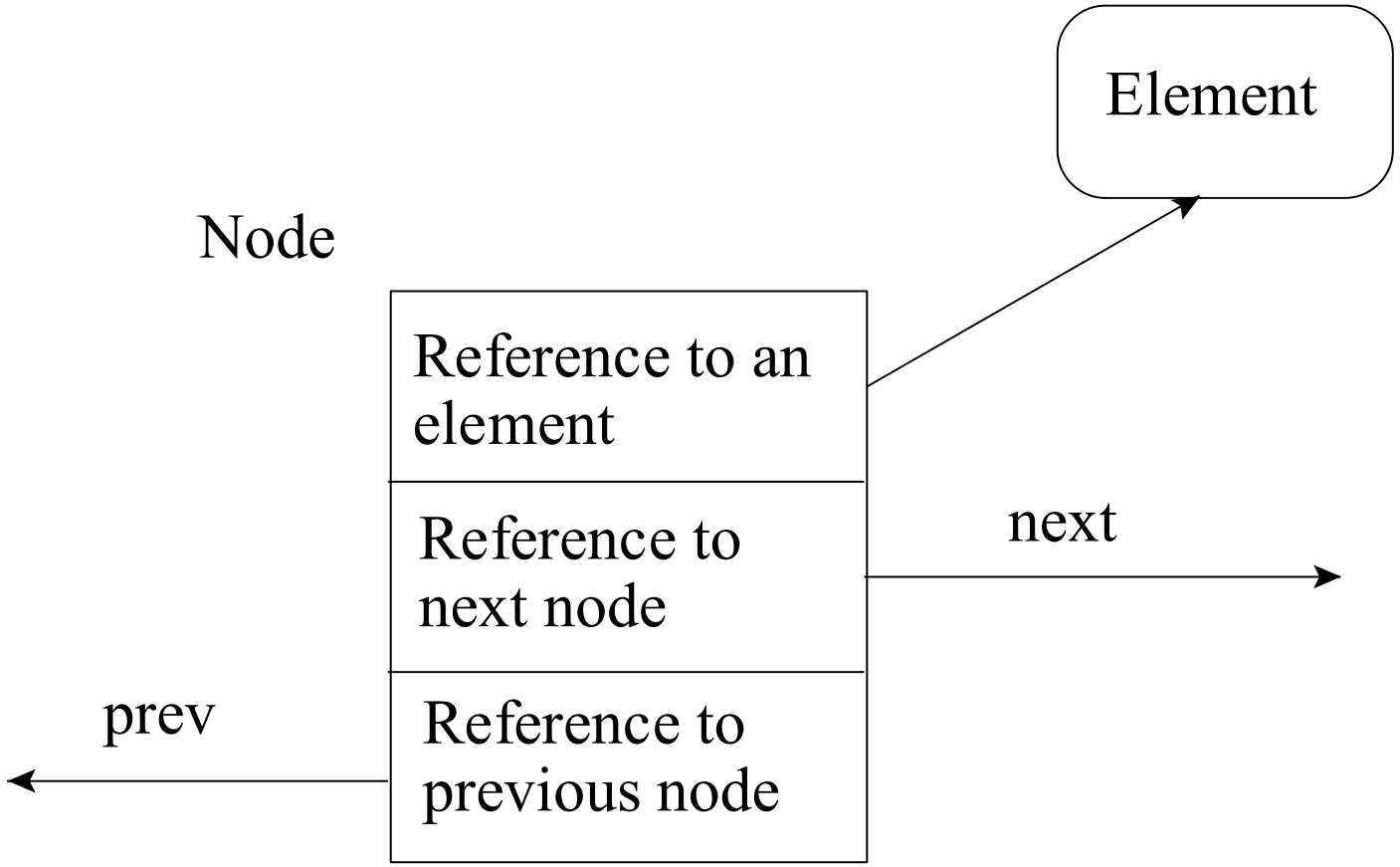
Doubly Linked List

Recall that the deletion of an element at the tail is not easy because we have to find the node before the tail (the last node) by link hopping.

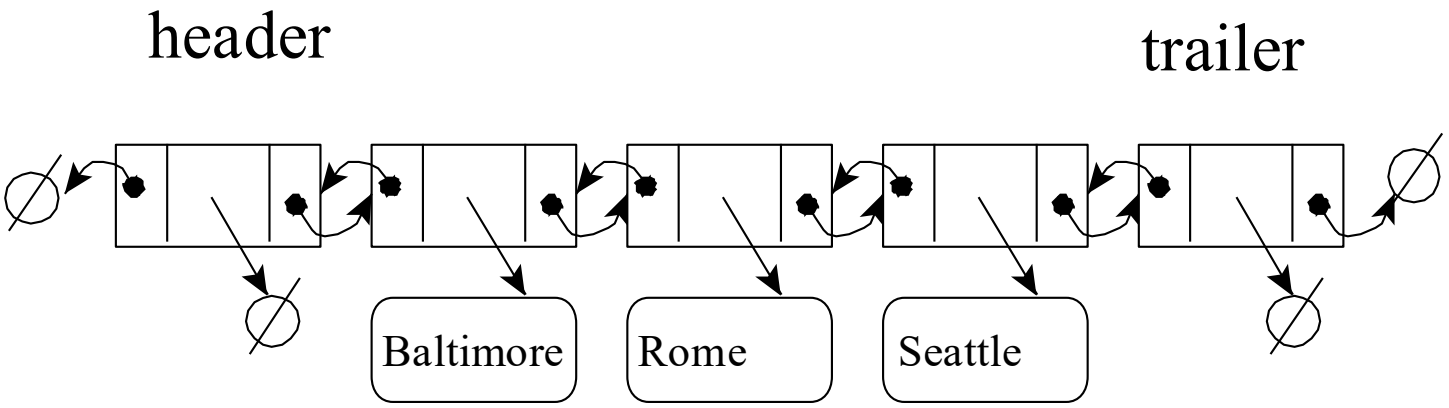


This problem can be easily solved by using the double linked list.

A **node** in a doubly linked list: A compound object that stores a reference to an element and two references, called **next** and **prev**, to the next and previous nodes, respectively.



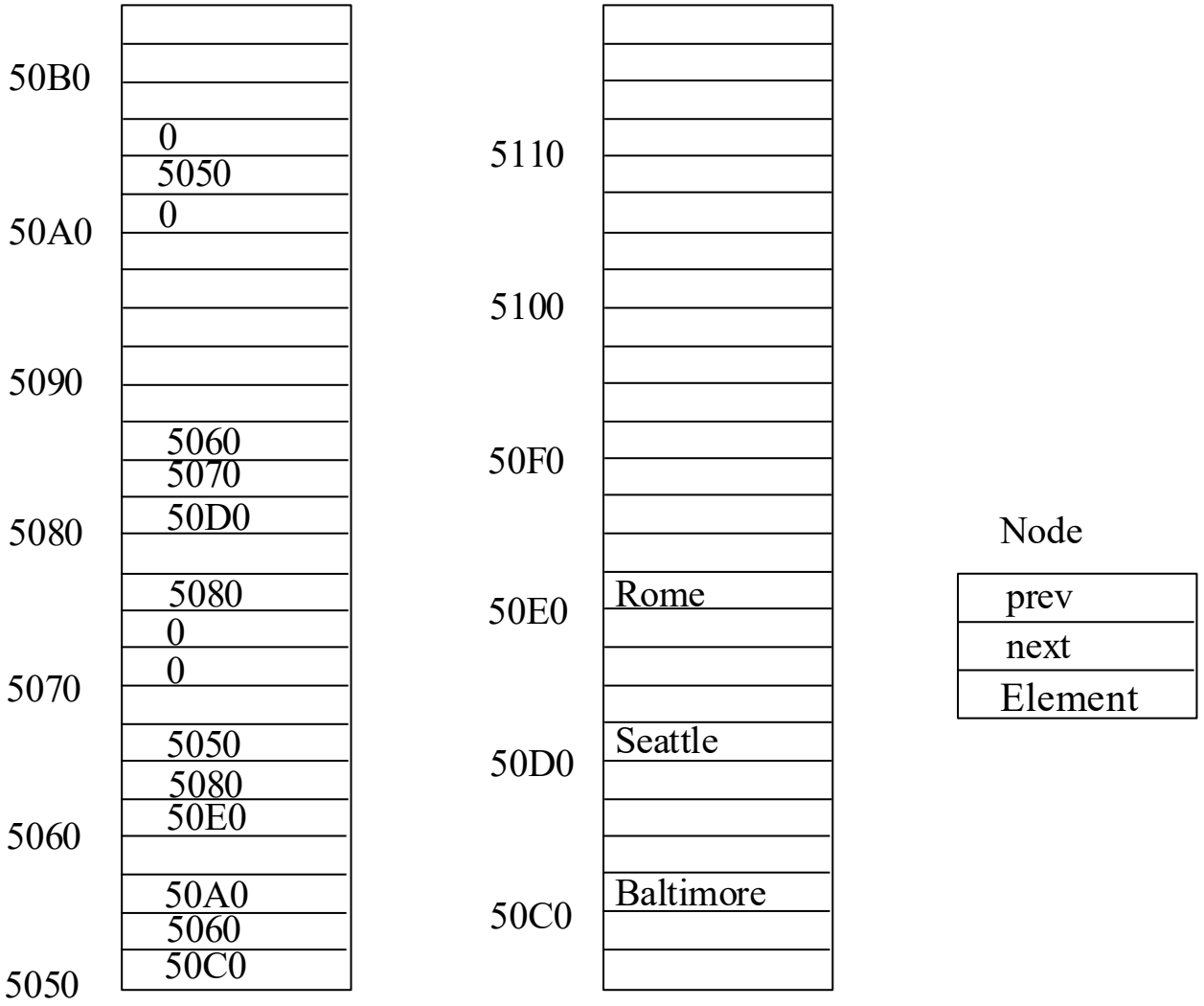
For convenience, a doubly linked list has a **header** node and a **trailer** node. They are also called **sentinel** nodes, indicating both the ends of a list.

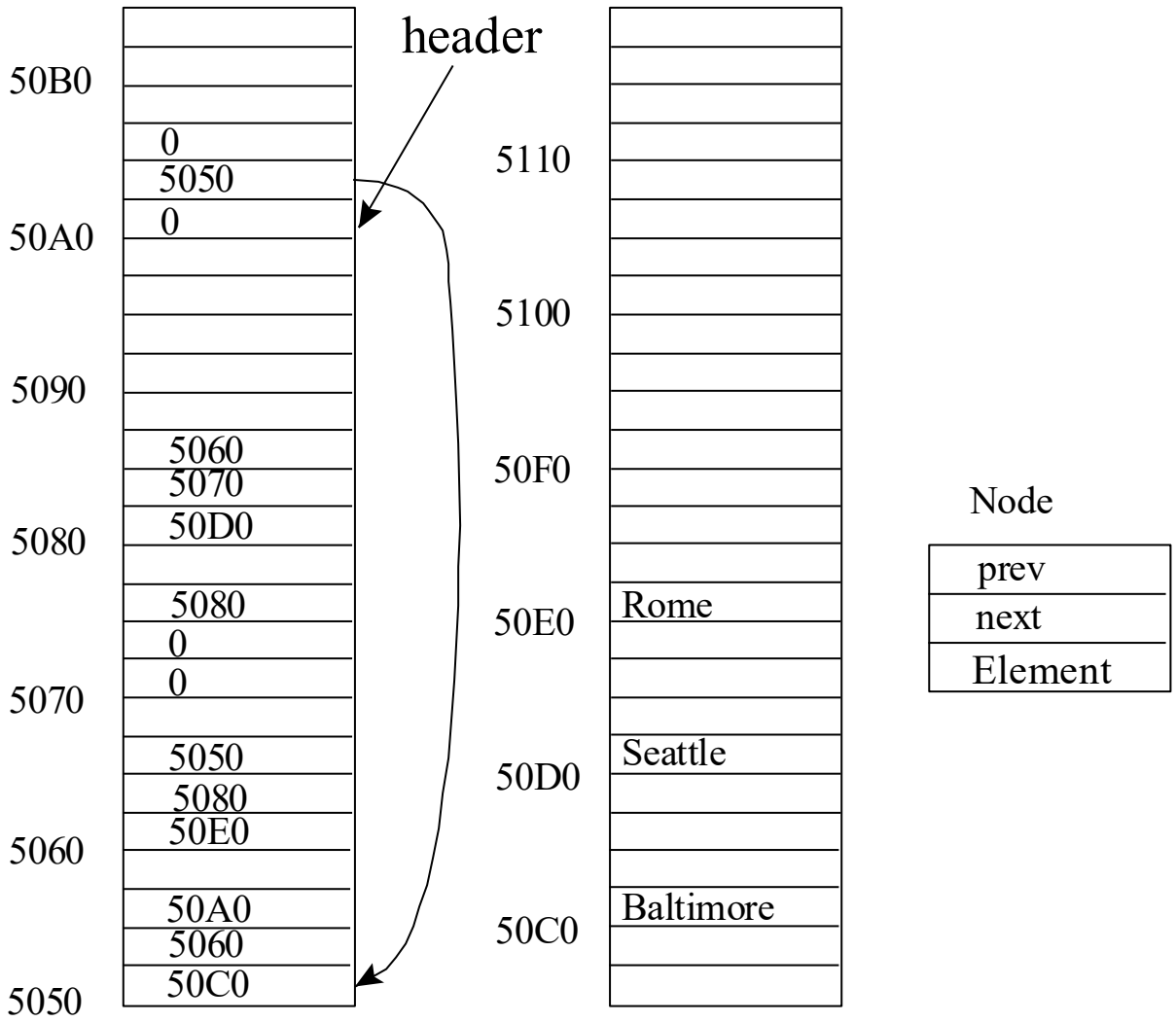


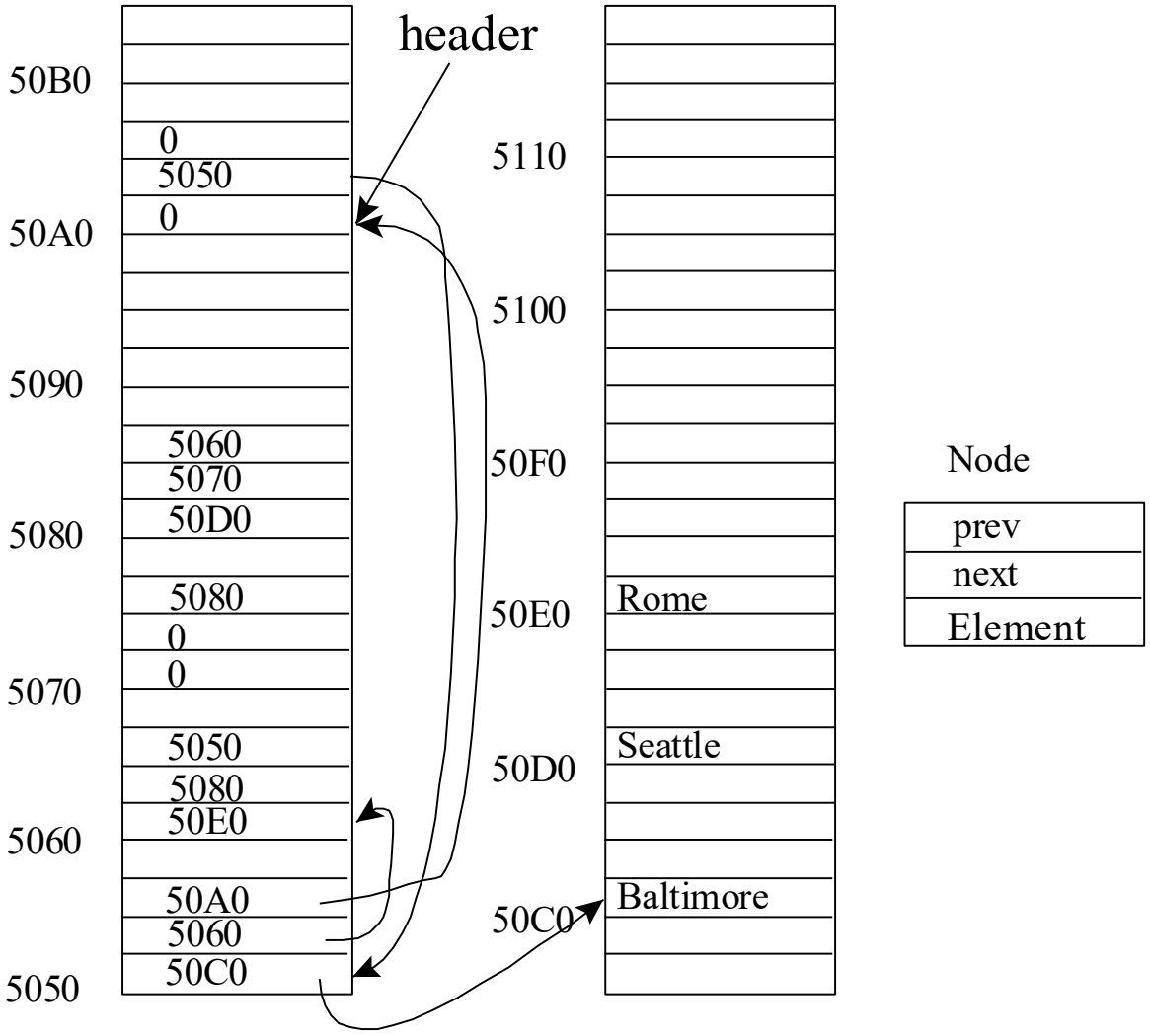
Difference from singly linked lists:

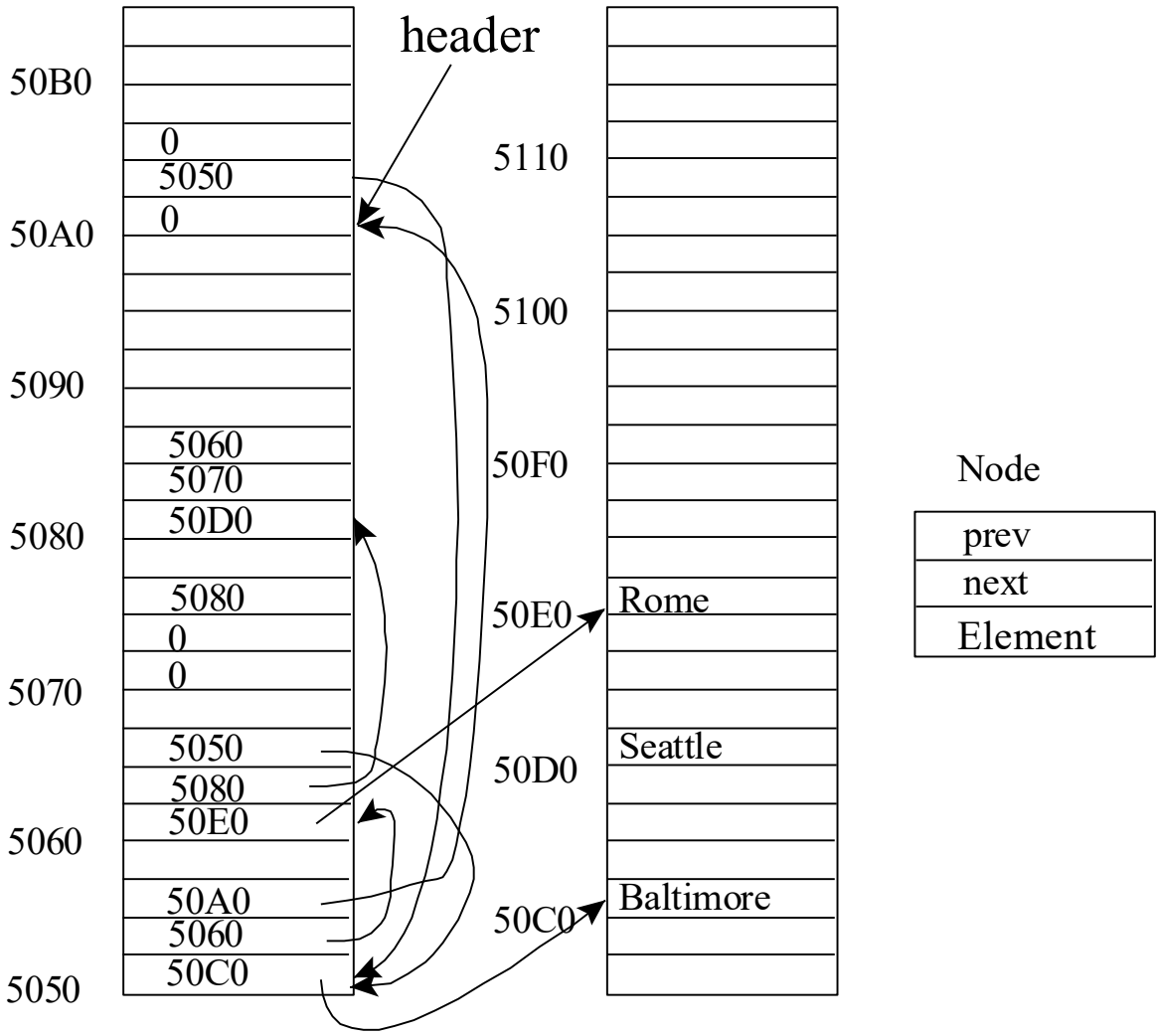
- each node contains two links.
- two extra nodes: header and trailer, which contain no elements.

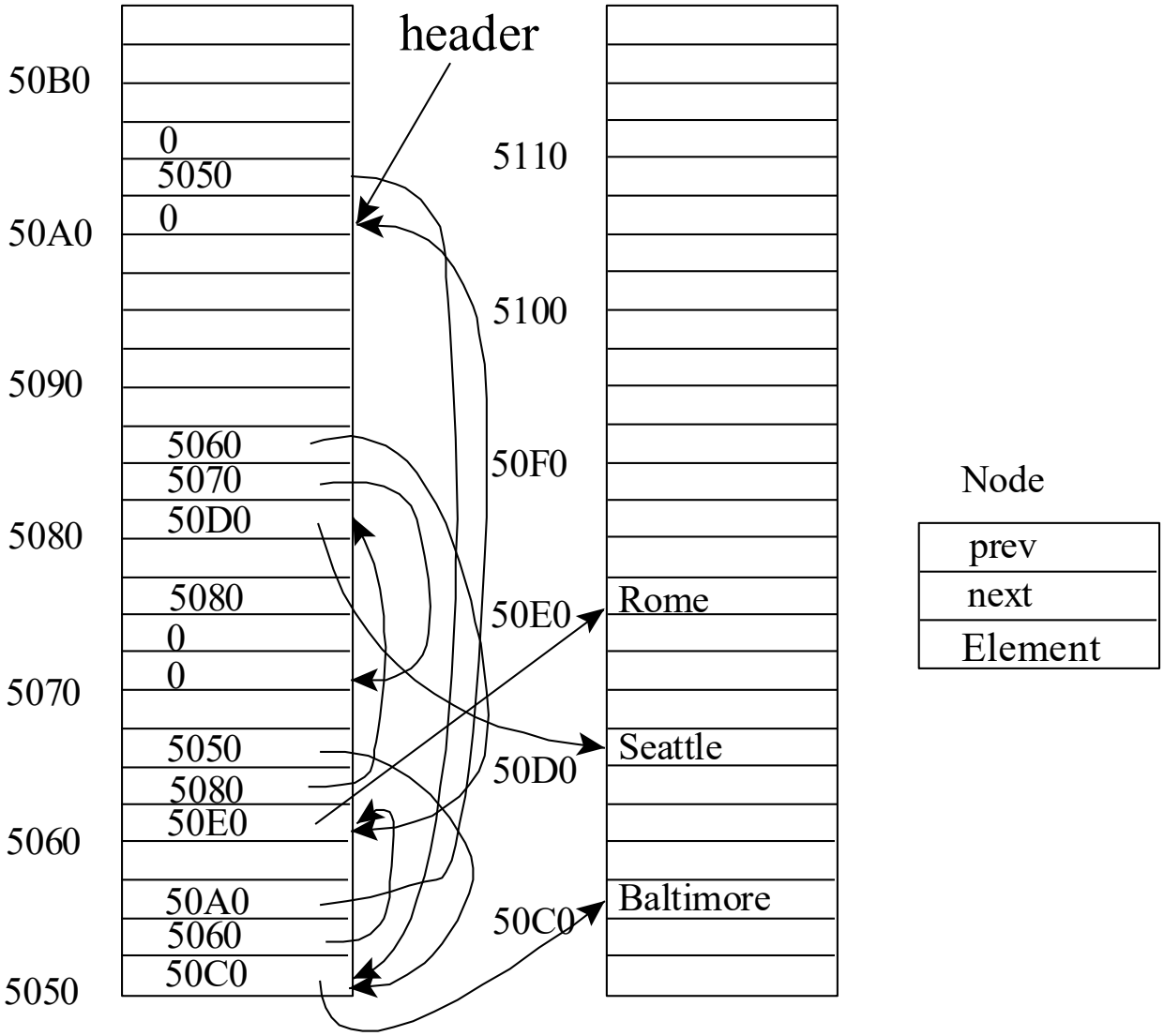
Example:

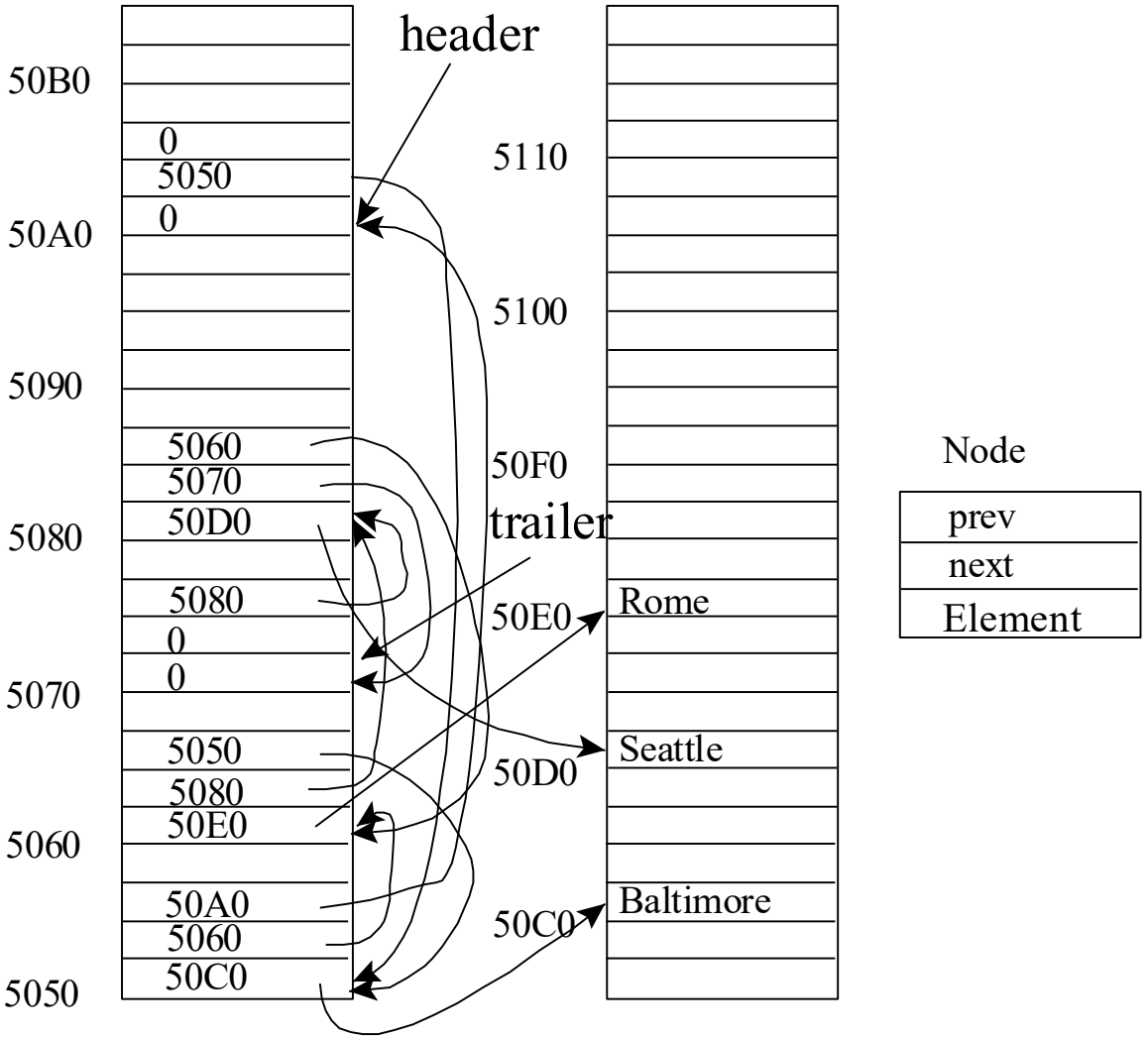

50B0
0
5050
50A0
0
5090
5060
5070
5080
50D0
5080
0
5070
0
5050
5080
5060
50E0
50A0
5060
5050
50C0
5110
5100
50F0
50E0
Rome
50D0
Seattle
50C0
Baltimore
Node
prev
next
Element


header
50B0
0
5050
50A0
0
5090
5060
5070
50D0
5080
5080
0
5070
0
5050
5080
50E0
5060
50A0
5060
50C0
5050
5110
5100
50F0
50E0
Rome
50D0
Seattle
50C0
Baltimore
Node
prev
next
Element


header
50B0
0
5050
50A0
0
5090
5060
5070
50D0
5080
5080
0
5070
0
5050
5080
50E0
5060
50A0
5060
50C0
5050
5110
5100
50F0
50E0
Rome
50D0
Seattle
Baltimore
50C0
Node
prev
next
Element


header
50B0
0
5050
50A0
0
5090
5060
5070
5080
50D0
5080
0
5070
0
5050
5080
50E0
5060
50A0
5060
50C0
5050
5110
5100
50F0
50E0
Rome
50D0
Seattle
50C0
Baltimore
Node
prev
next
Element


header
50B0
0
5050
50A0
0
5090
5060
5070
50D0
5080
5080
0
0
5070
5050
5080
50E0
5060
50A0
5060
50C0
5050
5110
5100
50F0
50E0
50D0
50C0
Rome
Seattle
Baltimore
Node
prev
next
Element


header
50B0
0
5050
0
50A0
5090
5060
5070
50D0
5080
5080
0
0
5070
5050
5080
50E0
5060
50A0
5060
50C0
5050
5110
5100
50F0
trailer
50E0
Rome
50D0
Seattle
50C0
Baltimore
Node
prev
next
Element

Class DLNode

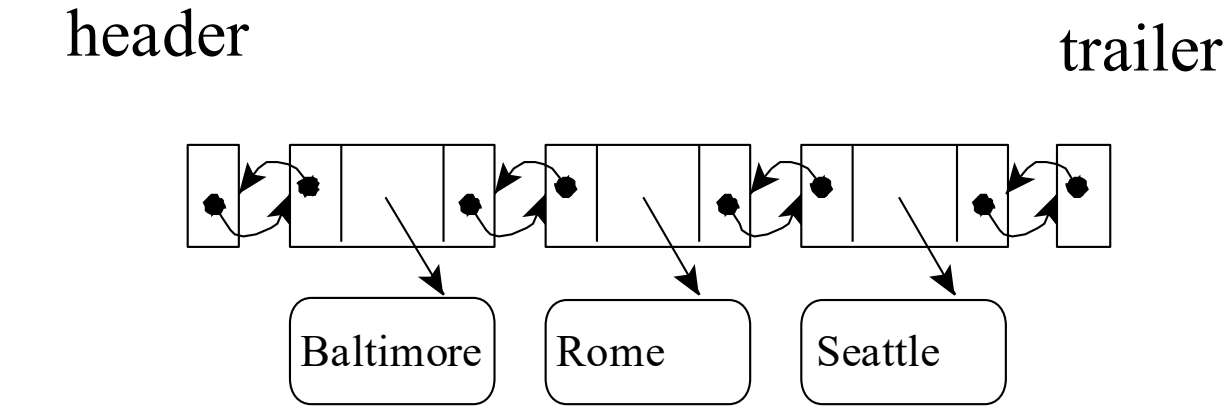
Here is an implementation of nodes for doubly linked lists in Java:

```
public class DLNode {
    private Object element;
    private DLNode next, prev;

    public DLNode() {
        this( null, null, null );
    }

    public DLNode( Object e, DLNode p, DLNode n
) {
        element = e;
        next = n;
        prev = p;
    }
```

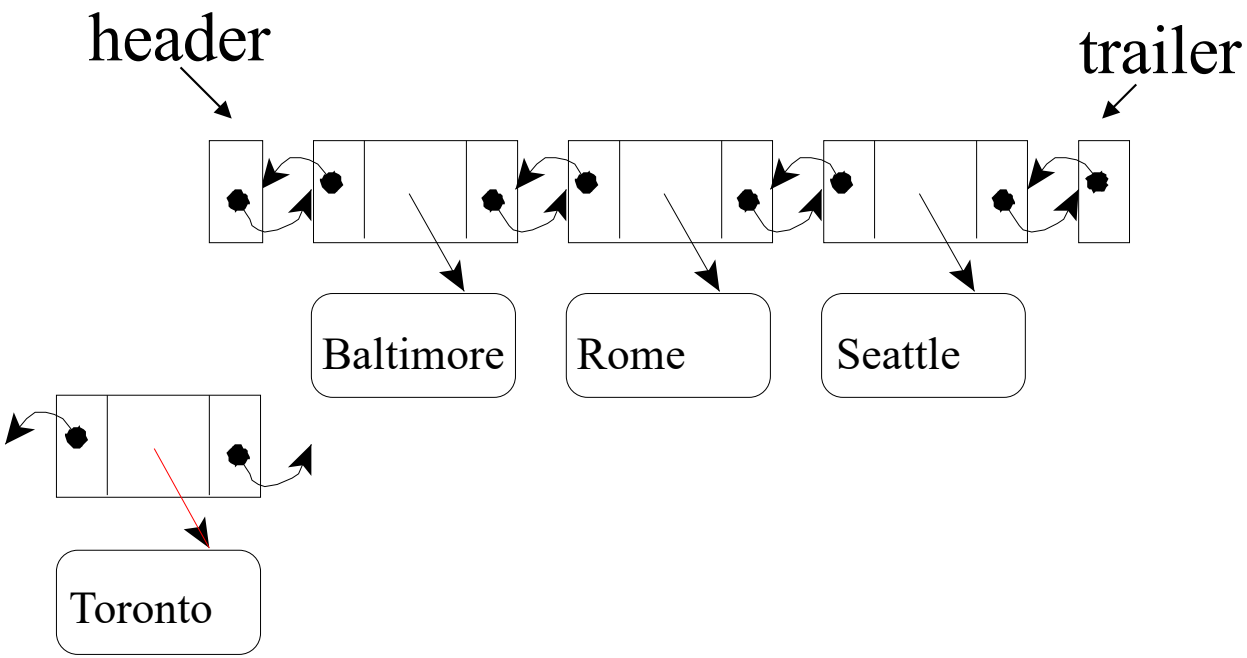
```
    void setElement( Object newElem ) {
        element = newElem;
    }

    void setNext( DLNode newNext ) {
        next = newNext;
    }

    void setPrev( DLNode newPrev ) {
        prev = newPrev;
    }

    Object getElement() {
        return element;
    }

    DLNode getNext() {
        return next;
    }

    DLNode getPrev() {
        return prev;
    }
}
```

Insertion of an Element at the Head

Before the insertion:

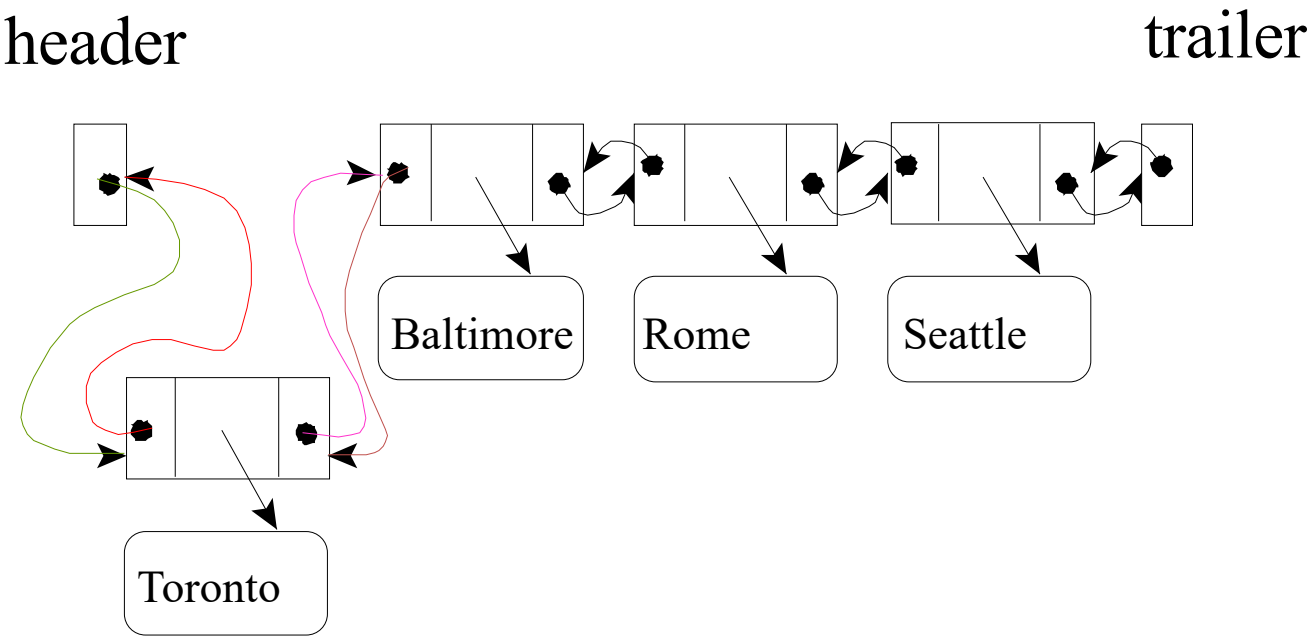


Have a new node:




```
DLNode x = new DLNode();
x.setElement(new String("Toronto"));
(x.element = new String("Toronto"))
```

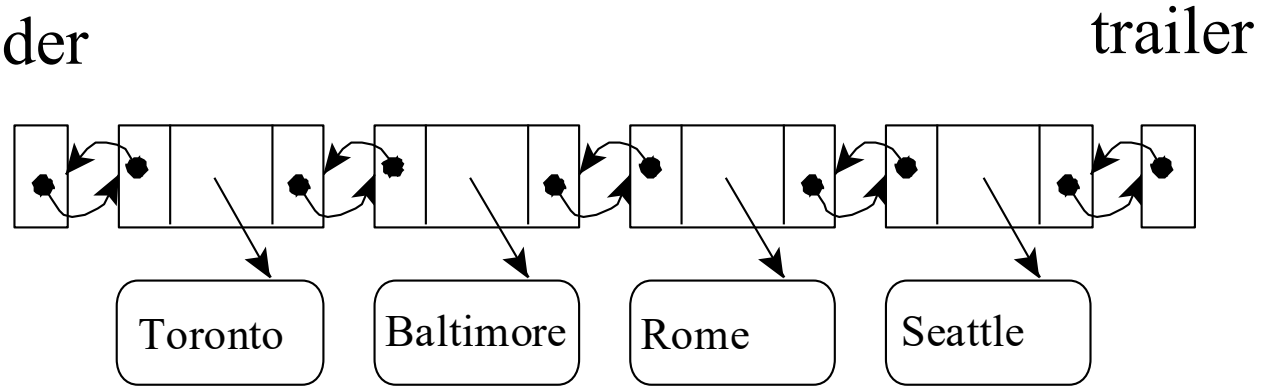
Update the links:




```
x.setPrev(header);
x.setNext(header.getNext());
(header.getNext()).setPrev(x);
header.setNext(x);
```

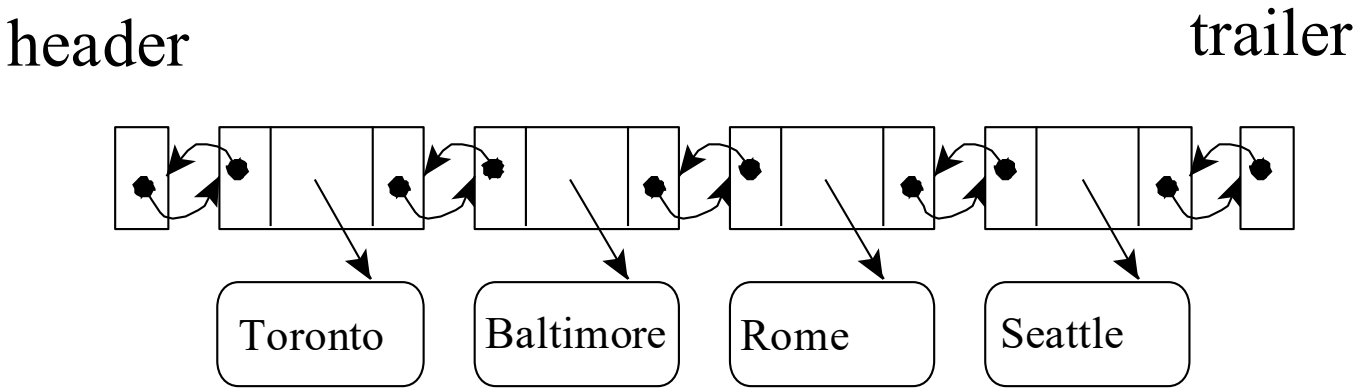
```
x.prev ← header;
x.next ← header.next;
header.next.prev ← x;
header.next ← x;
```

After the insertion:

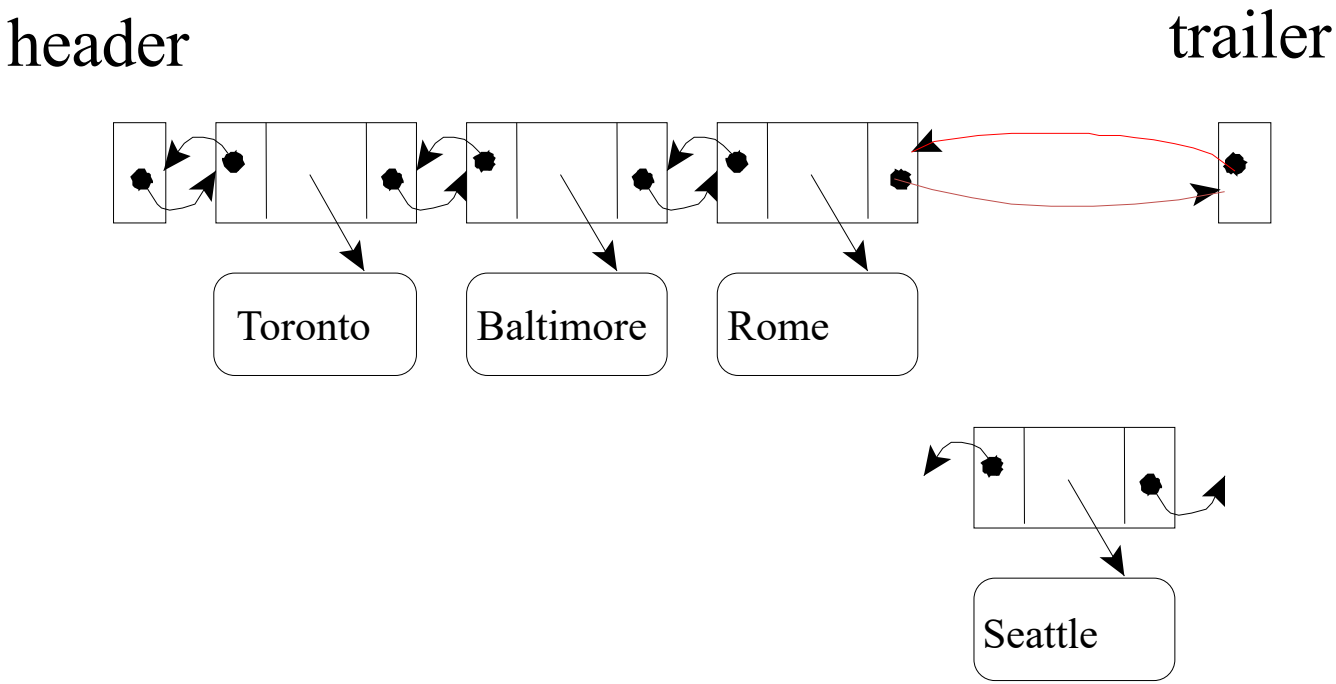


Deleting an Element at the Tail

Before the deletion:



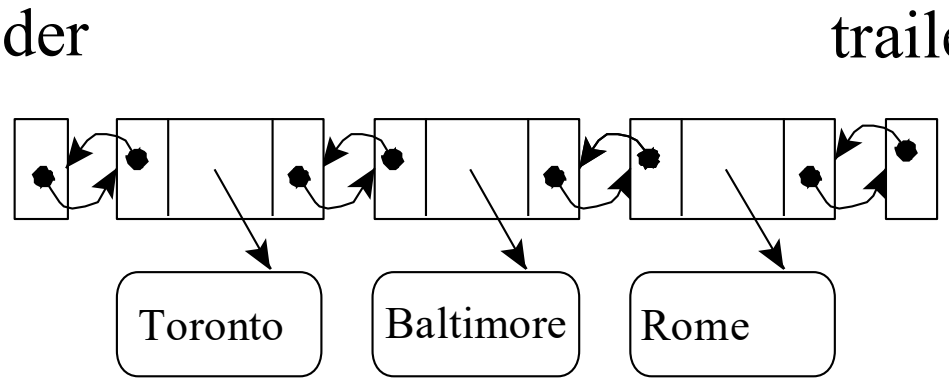
Update the links:




```
((trailer.getPrev()).getPrev()).setNext(trailer);
trailer.setPrev((trailer.getPrev()).getPrev());
```

```
trailer.prev.prev.next ← trailer;
trailer.prev ← trailer.prev.prev;
```

After the deletion:



Deleting an element at the head is similar to deleting an element at the tail.

Inserting an element at the tail is similar to inserting an element at the head.

No link hopping is needed in any of the operations.

However, for inserting a node into the middle of a double linked list or deleting a node in the middle, link hopping is always needed.