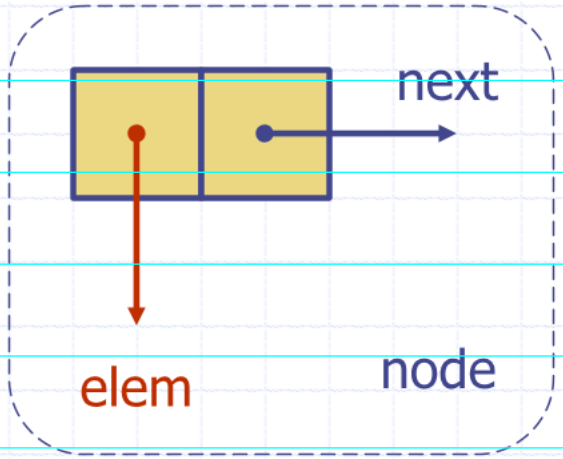
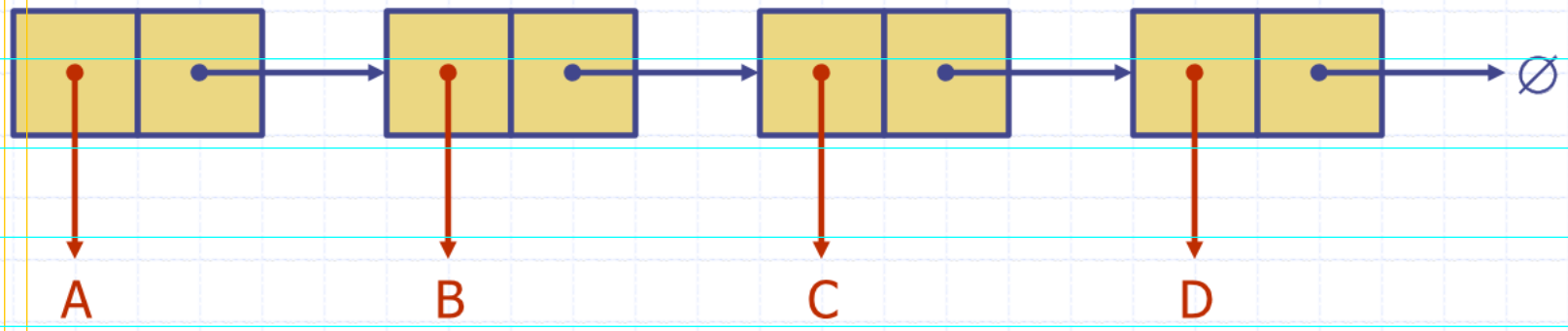
Singly Linked List (§ 4.4.1)

- A singly linked list is a concrete data structure consisting of a sequence of nodes
- Each node stores
  - element
  - link to the next node



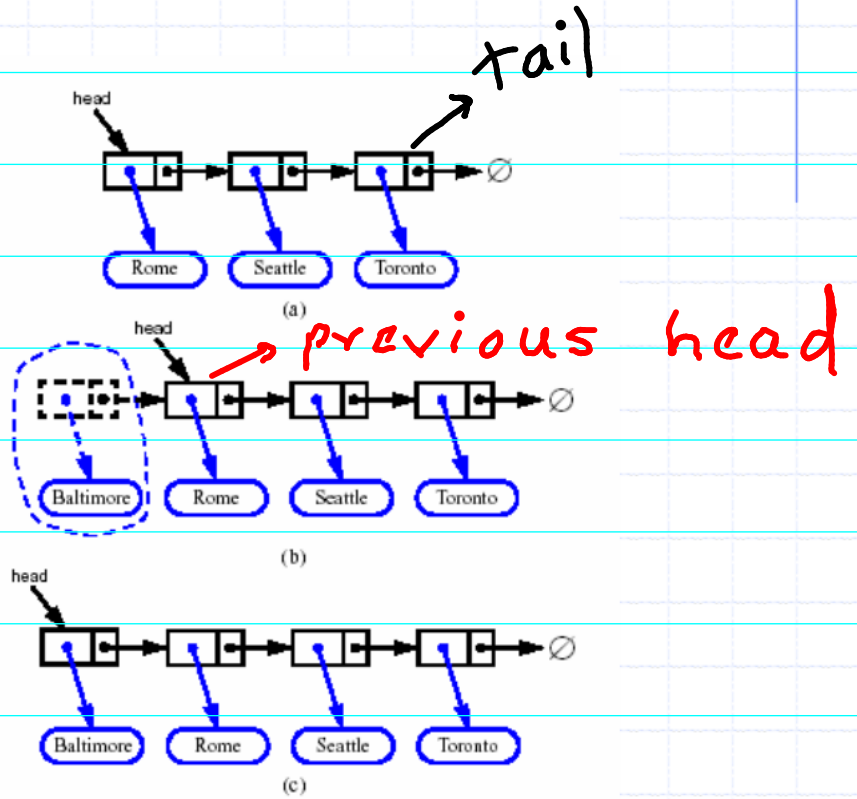


The Node Class for List Nodes

```
public class Node {
  // Instance variables:
  private Object element;
  private Node next;
  /** Creates a node with null references to its element and next node. */
  public Node() {
    this(null, null);
  }
  /** Creates a node with the given element and next node. */
  public Node(Object e, Node n) {
      element = e;
      next = n;
  }
  // Accessor methods:
  public Object getElement() {
    return element;
  }
  public Node getNext() {
    return next;
  }
  // Modifier methods:
  public void setElement(Object newElem) {
      element = newElem;
  }
  public void setNext(Node newNext) {
      next = newNext;
  }
}
```

Inserting at the Head

1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node

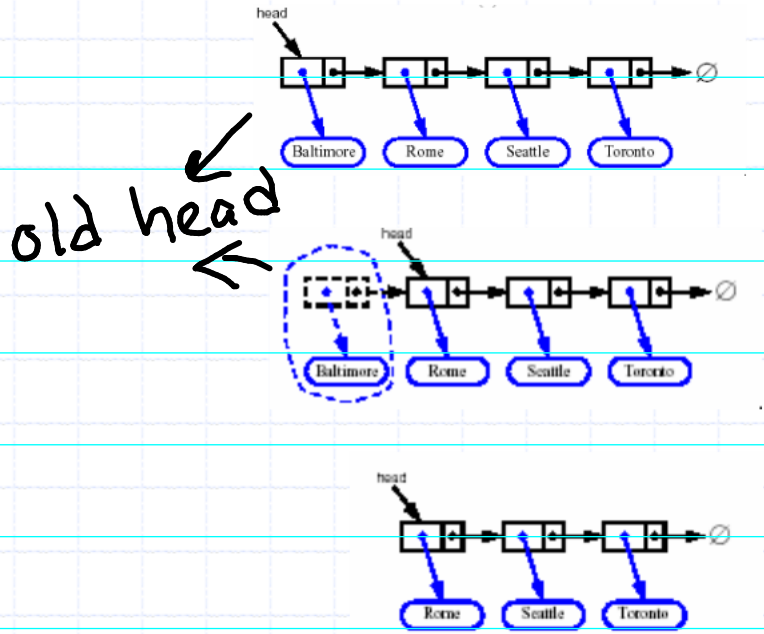


head : first element

O(1)

Removing at the Head

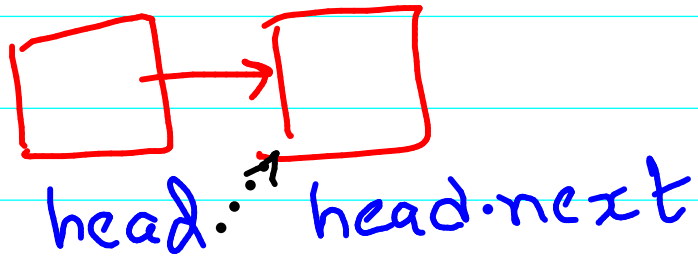
1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node



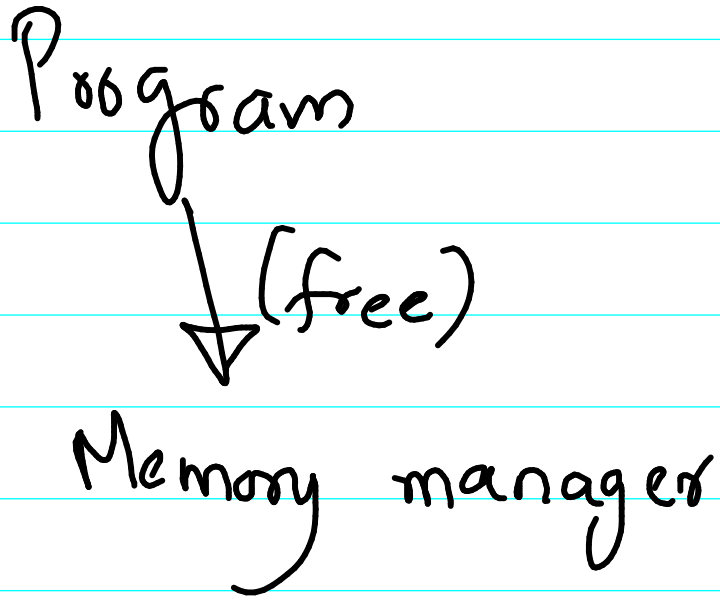
Delete method for linked list

```
public void deletehead( ) {
    head = head.getNext( );
}
```

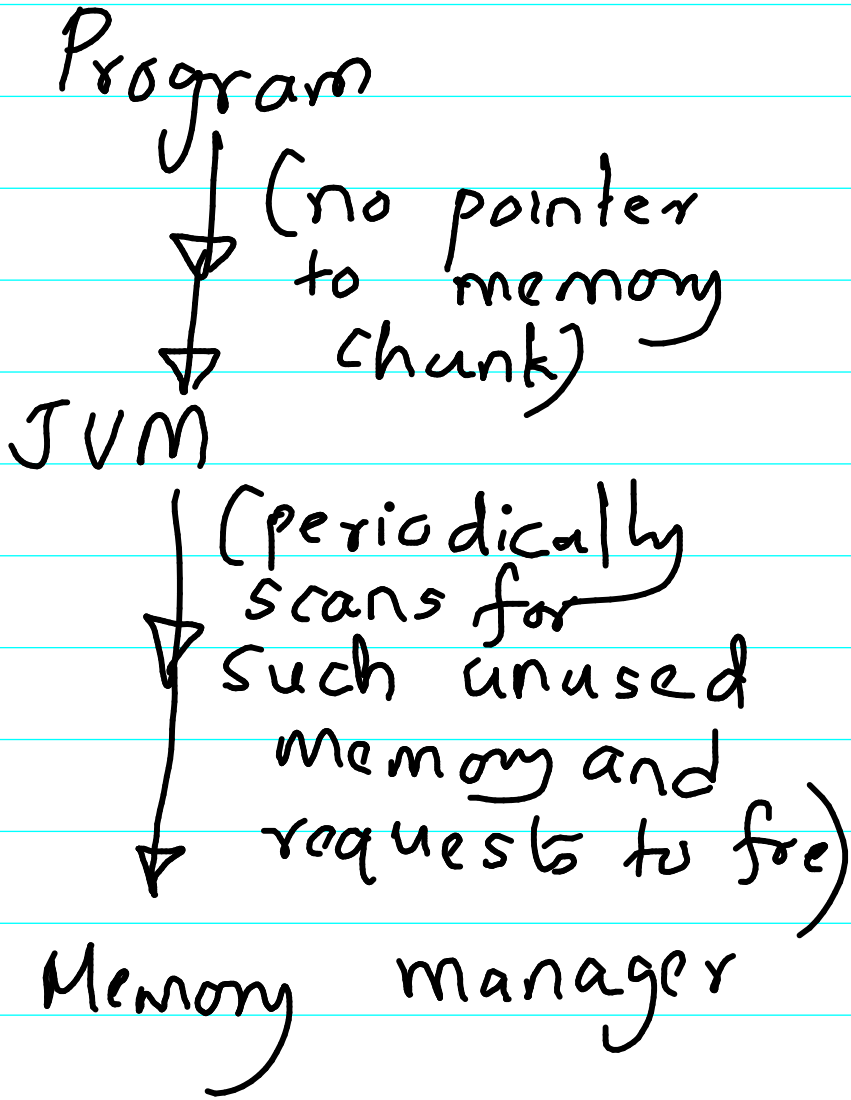
In C++ you would next "free" head
In Java, garbage collector, run periodically by the virtual M/c (JVM) does this freeing



C++


Program
(free)
Memory manager

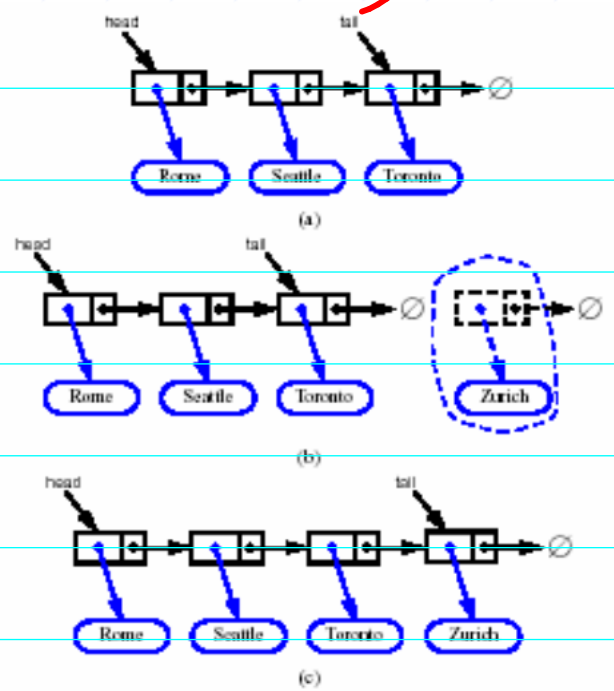
Java


Program
(no pointer to memory chunk)
JVM
(periodically scans for such unused memory and requests to free)
Memory manager

Inserting at the Tail

(Assuming you have a pointer to tail)

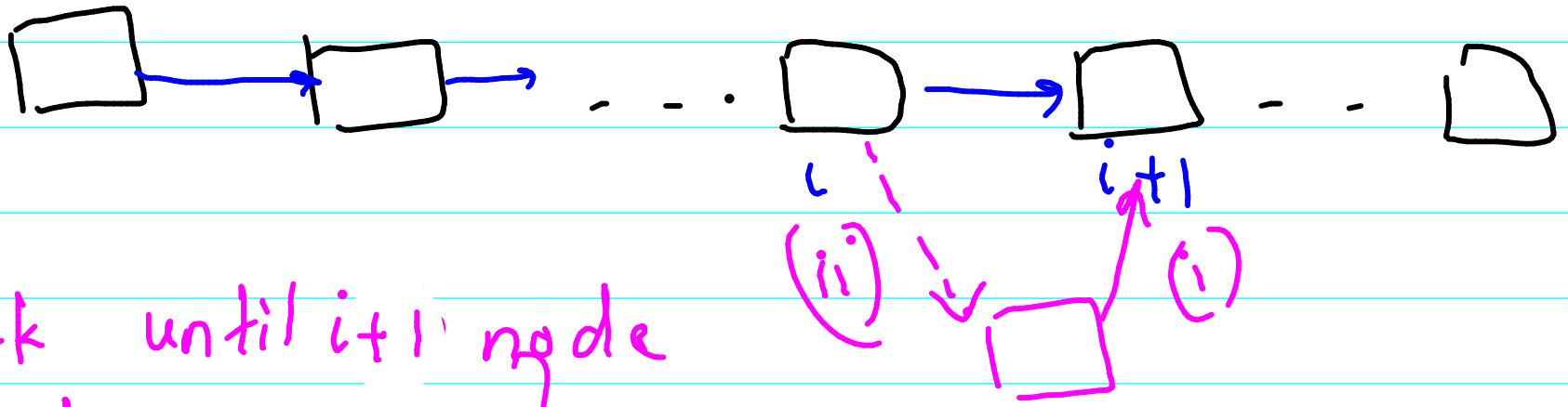
1. Allocate a new node
2. Insert new element
3. Have new node point to null
4. Have old last node point to new node
5. Update tail to point to new node



O(1)

O(N) if you dont keep track of tail.

Element Insertion between i^{th} node & $i+1^{th}$ node



(1) Seek until i+1 node

(2) Create a new node for new element

(3) Set "next" pointers of i^{th} node & new node

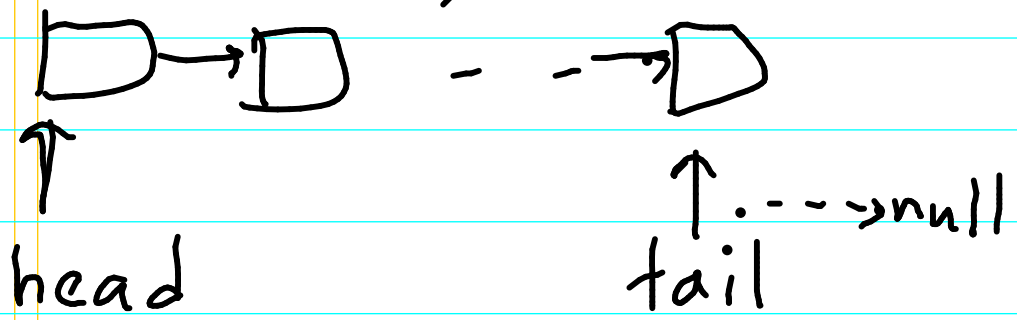
Approx i+3 operations

Complexity specified in terms of input (size) N = O(N) since i can be N-1 in worst case

Removing at tail

① tail = null

(NO!)



O(N)

② Traverse till

a) node.getNext().

getNext()

= null

b) node.setNext(null)

c) tail : node

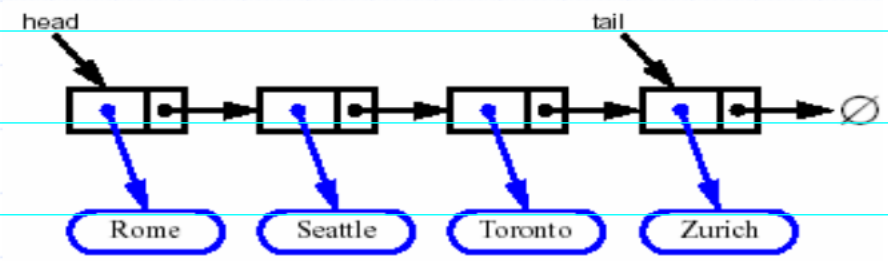
(YES)

First check for corner cases: ie head = tail

(list containing single element)

Removing at the Tail

- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node



Q: Suppose you want to implement a "list", not as Linked list but as an array of Nodes, i.e

Node[ ] nodeList

so that I could (a) insert at head
(b) insert at tail
(c) delete at head
(d) delete at tail.

Design a data structure that uses Node[ ] to implement above