EECS 114:
Engineering Data Structures and Algorithms
Lecture 3

Instructor: Ryan Rusich
E-mail: rusichr@uci.edu
Office: EH 2204

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lists

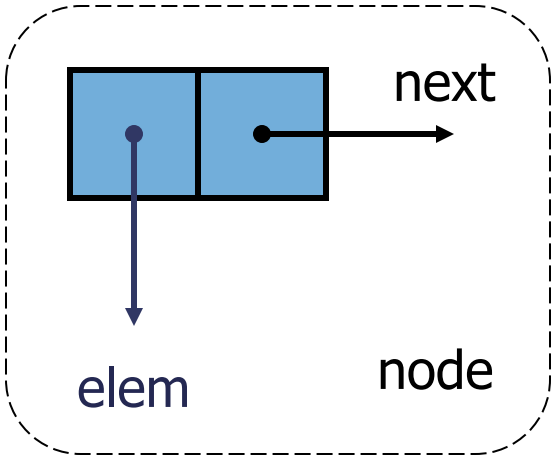
- Lists
  - List of students
  - List of games
  - List of assignments to complete
  - Etc.
- Is a collection of elements.
- One of most fundamental/simple data structures.
- Implementation:
  - Array-based
  - Node-based

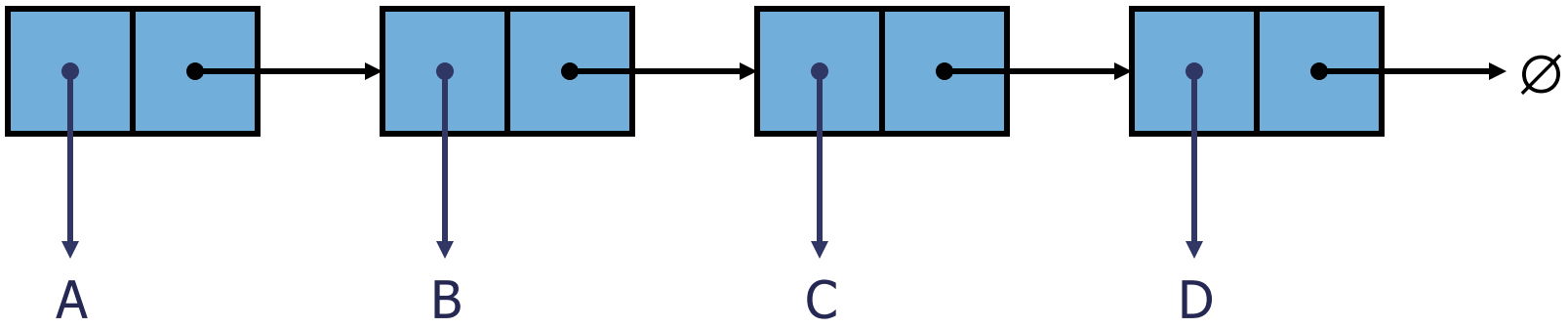
Node-based Linked List

- Composed of nodes.
- Each node holds data (item, value, element).
- Each node has a pointer (next) that connects it to the next node in the list.
- Special pointe r(head) to beginning of list.
- Size of the list is flexible.
  - List grows and shrinks with each insert/delete.
- Actions performed on a list:
  - Insert element
  - Remove element
  - Access/modify element
  - Traversel (search, print)

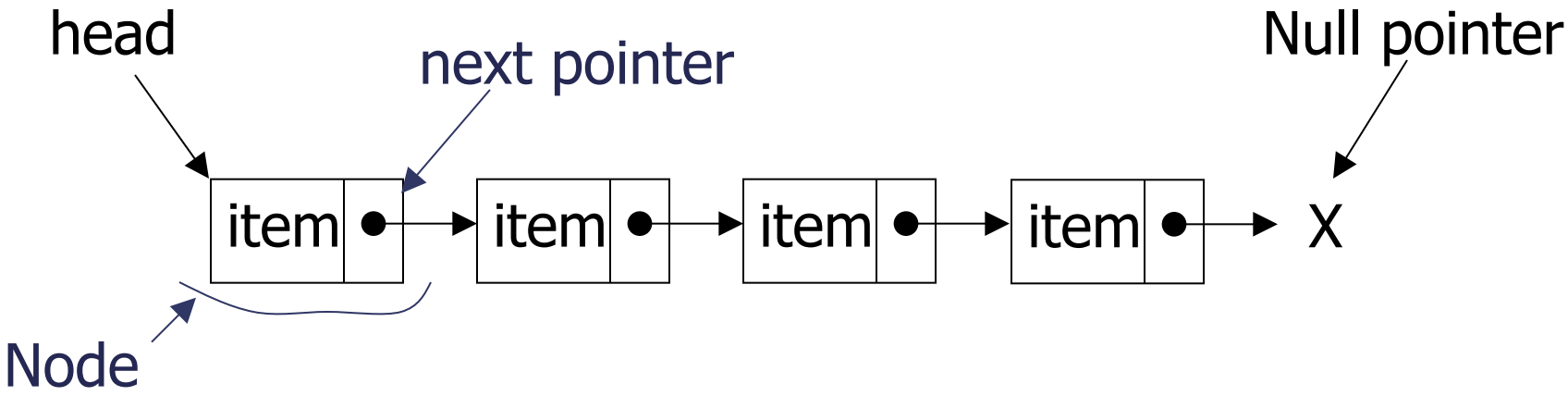
Singly Linked Lists

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

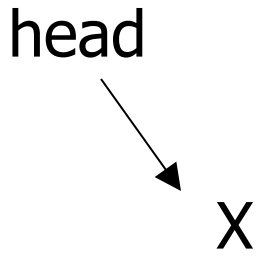




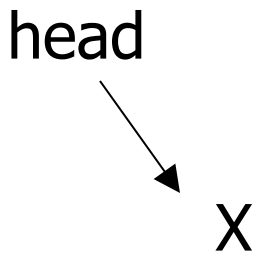
Linked List



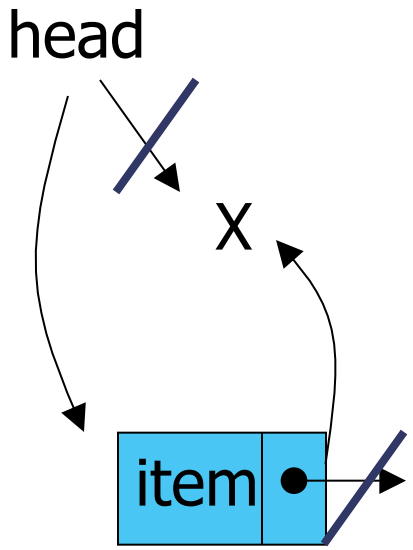
Empty list:



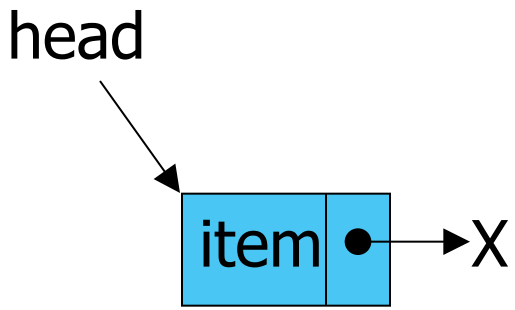
Linked list - Inserting Nodes



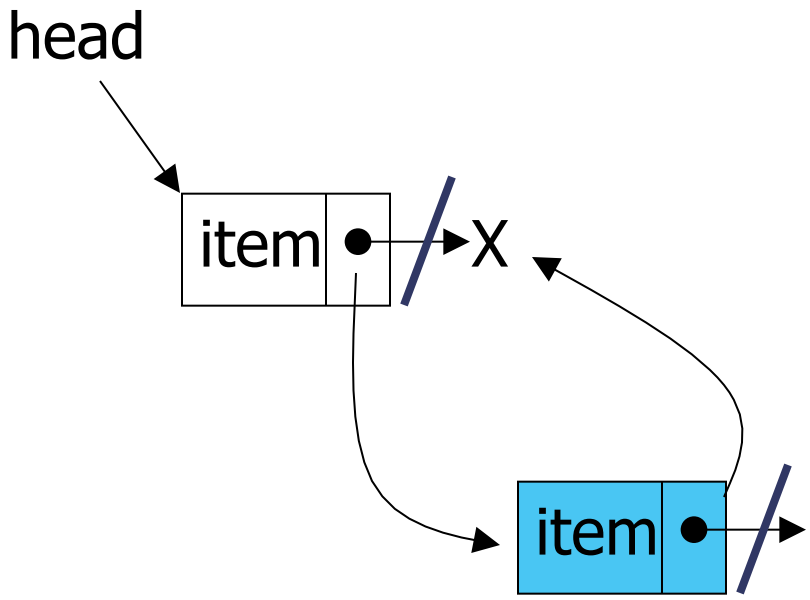
Linked List – Inserting Nodes



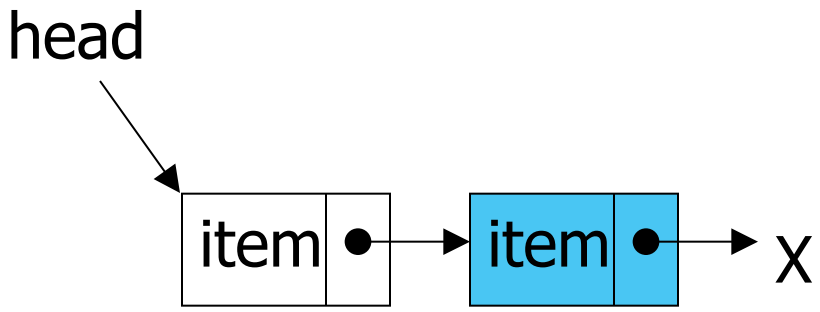
Linked List – Inserting Nodes



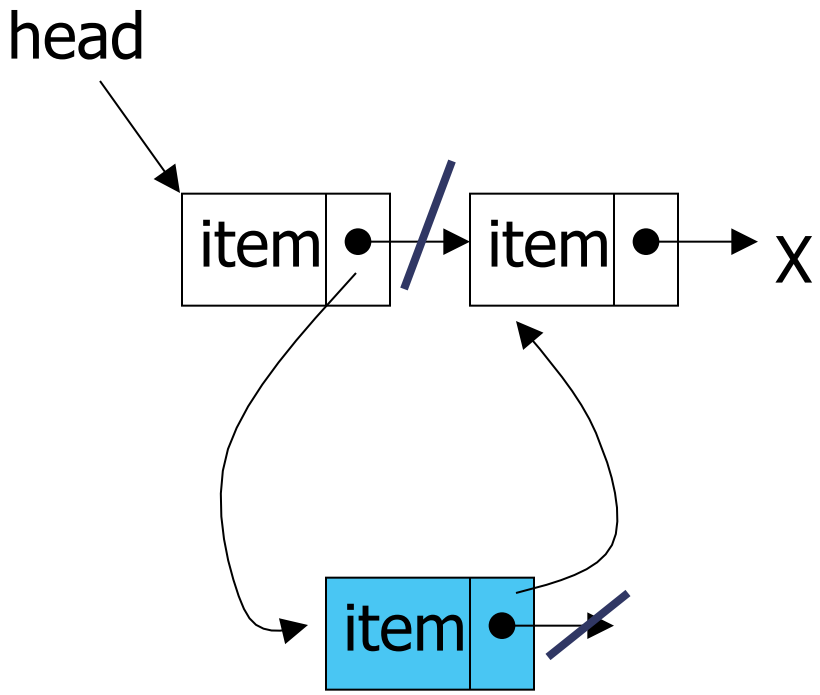
Linked List – Inserting Nodes



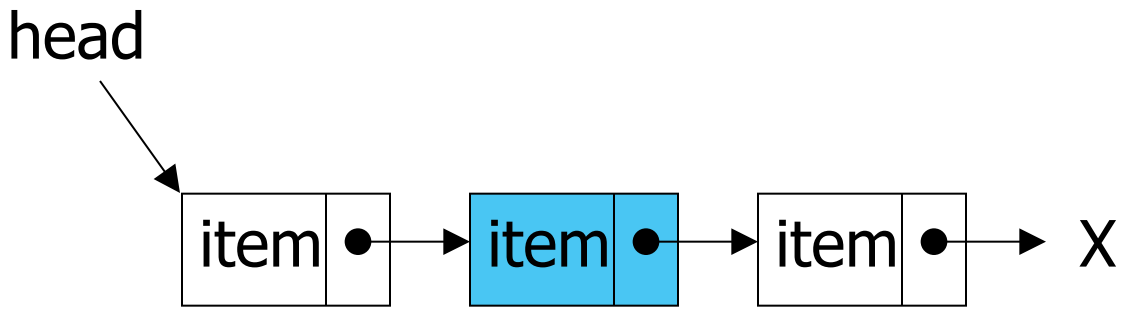
Linked List – Inserting Nodes



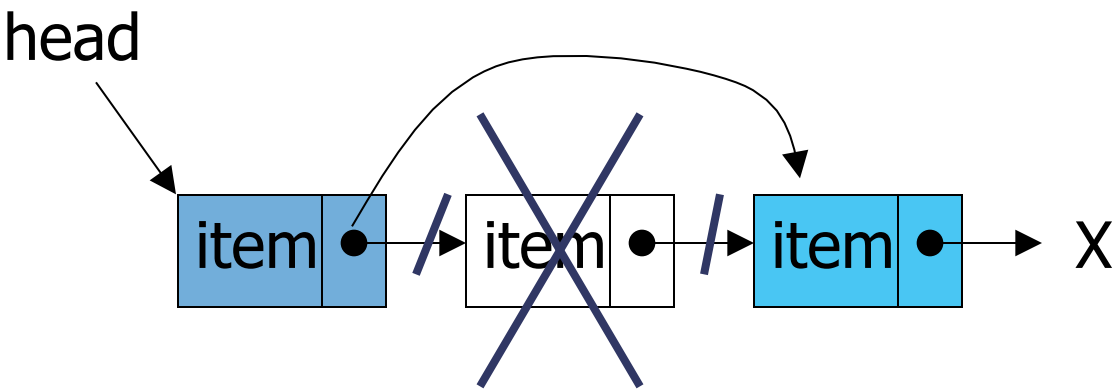
Linked List – Inserting Nodes



Linked List – Inserting Nodes



Linked List – Deleting Nodes



Node Class – Singly Linked List

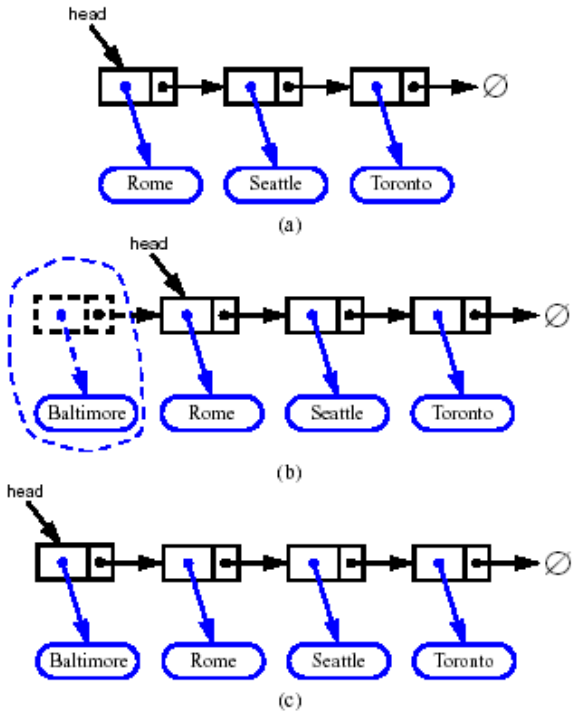
```
public class Node {
  private  Object element;
  private  Node next;
  public  Node() { this(null, null); }
  public  Node(Object e, Node n) {
    element = e;
    next = n;
  }
  // Accessor methods:
  public Object getElement() { return  element; }
  public Node getNext() { return  next; }
  // Modifier methods:
  public void setElement(Object newElem) { element = newElem; }
  public void setNext(Node newNext) { next = newNext; }
}
```

List Class – Singly Linked List

```
public class SLinkedList{
   protected Node head; // head node of the list
   /** Default constructor that creates an empty list
   */
   public SLinkedList() {
       head = null;
   }
   // ... Accessor, modifier, and search methods
   would go here ...
}
```

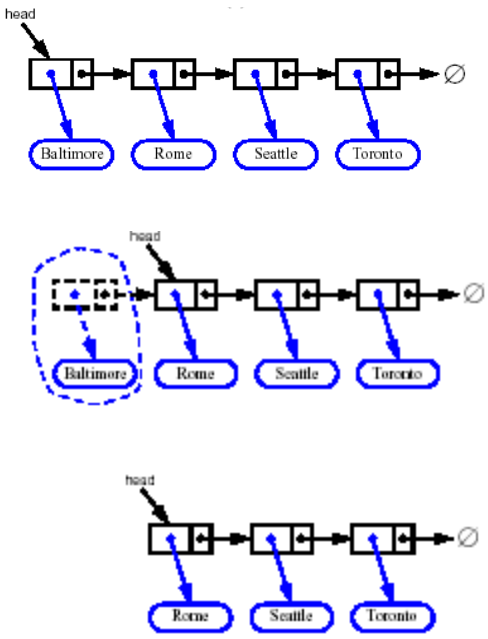
Inserting at the Head

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



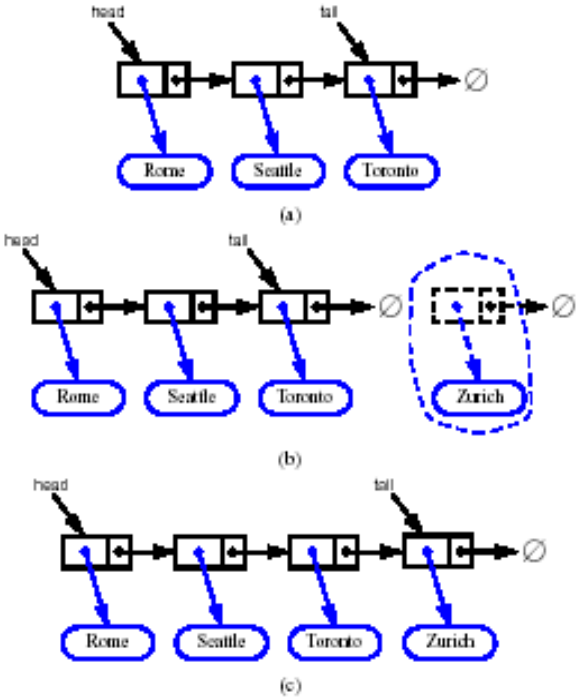
Removing at the Head

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

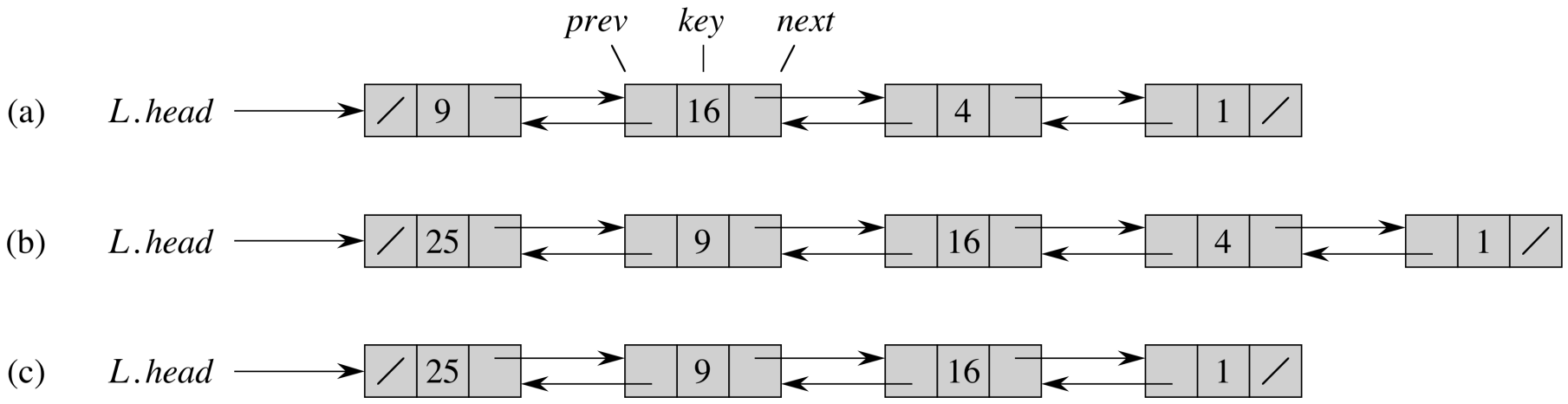


Inserting at the Tail

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



Doubly Linked List



Doubly Linked List – insertAfter(p, X)

- We visualize operation insertAfter(p, X), which returns position q