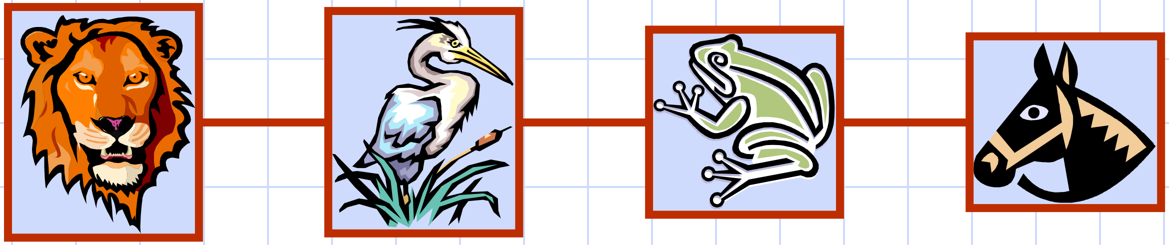
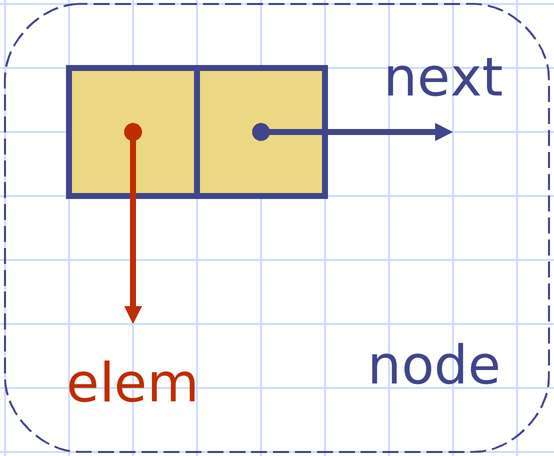
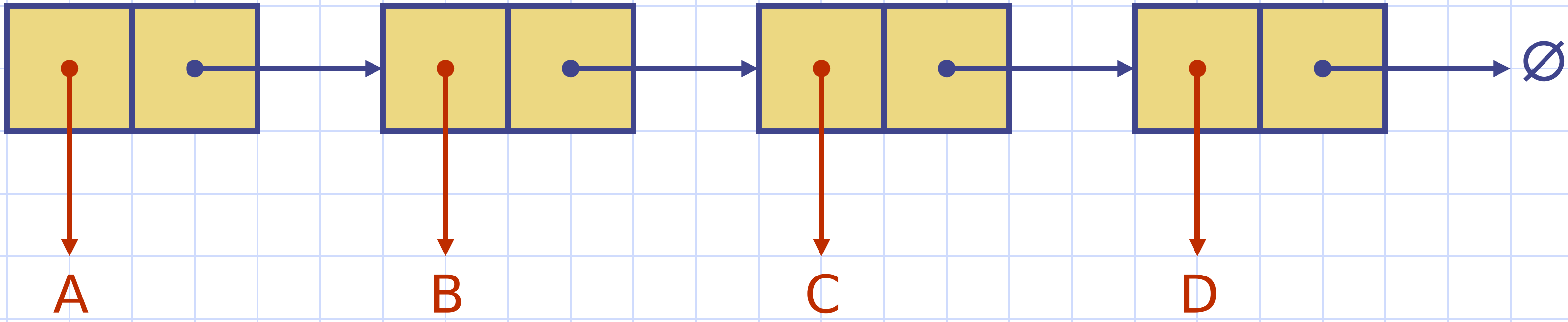
Linked Lists



Singly Linked List

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





© 2006 Goodrich, Tamassia

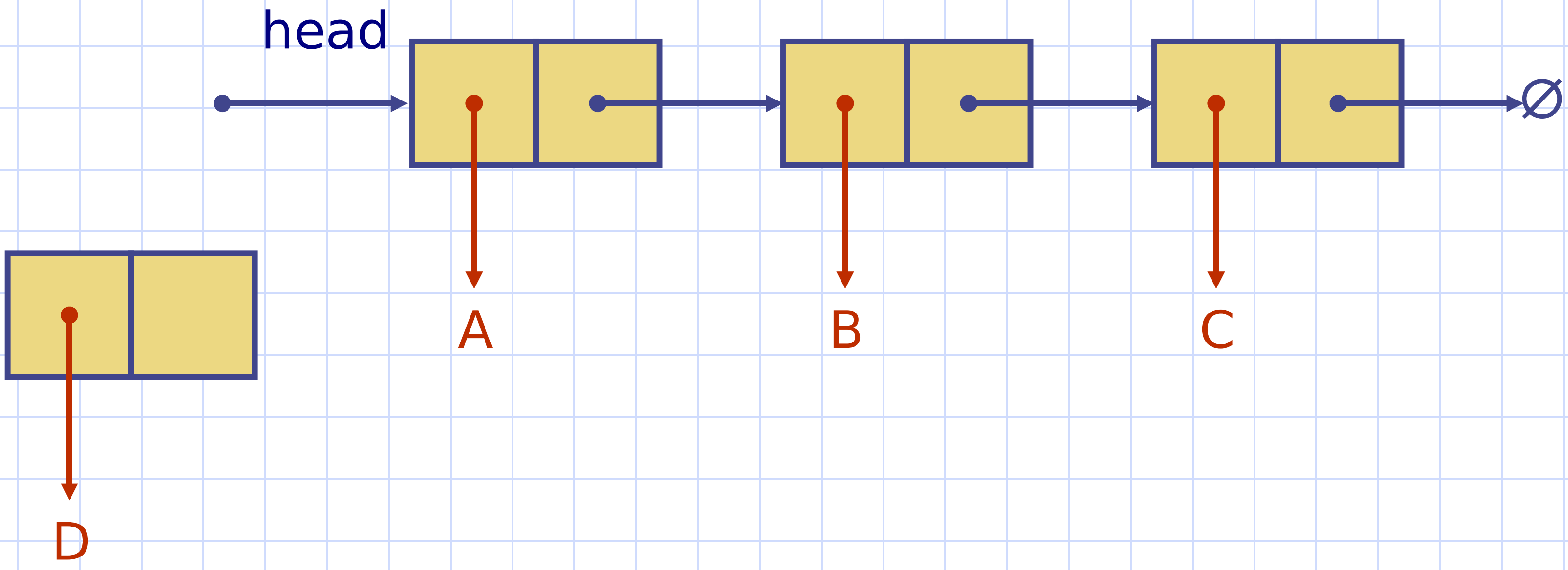
The Class for the List Nodes

```
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
 (Objectelement)  {
    this.element  =  elem;
 }
 public void setNext(Node
next)
 {
    this.next  =  next;
 }
}
```

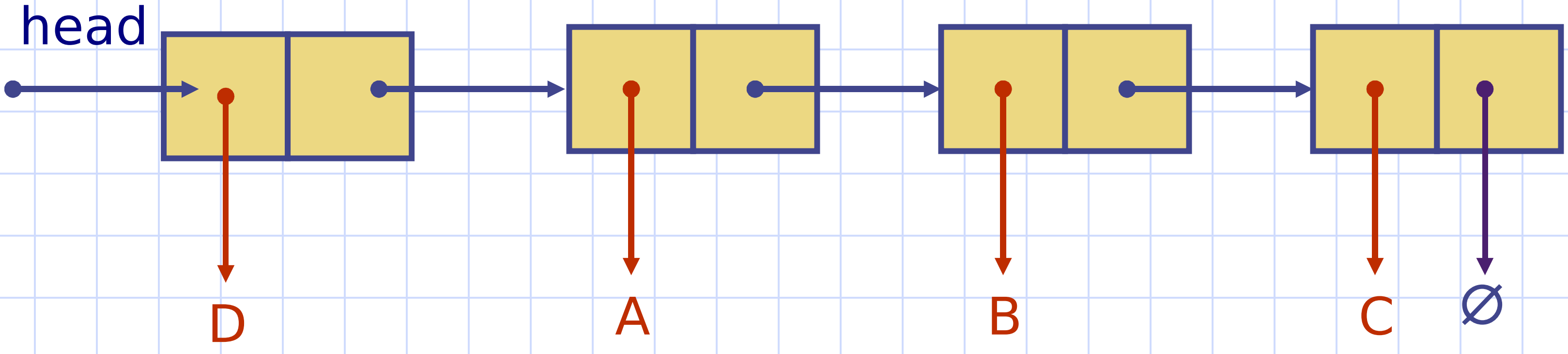
Inserting at the Head

1. Allocate a new node with a given element
2. Make the new node point to old head
3. Update head to point to new node



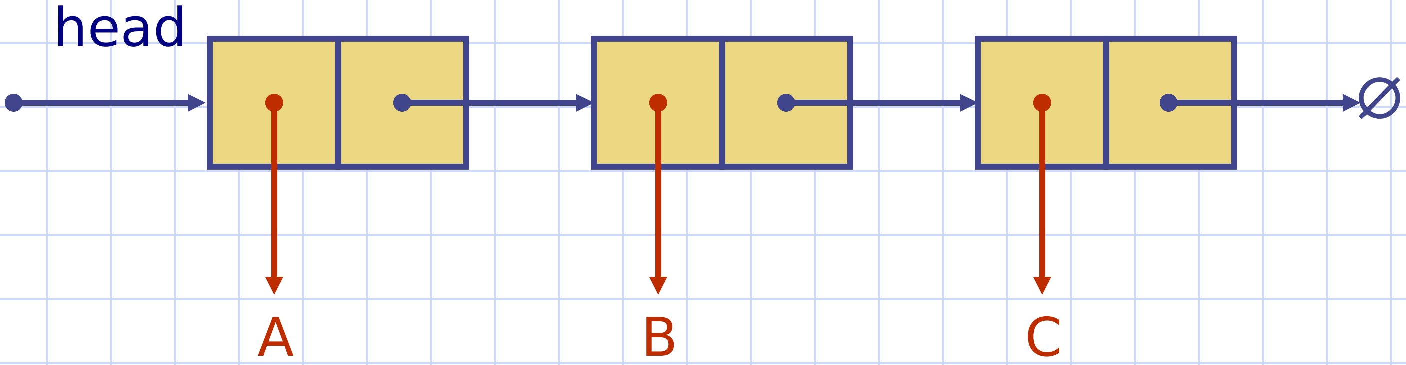
Inserting at the Head

1. Allocate a new node with a given element
2. Make the new node point to old head
3. Update head to point to new node



Removing at the Head

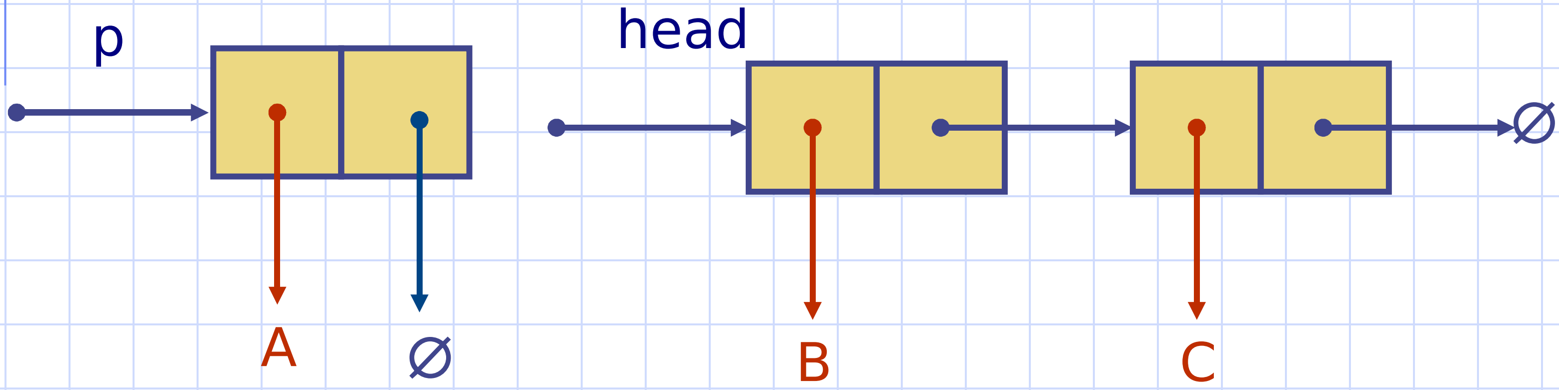
1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node



© 2006 Goodrich, Tamassia

Removing at the Head

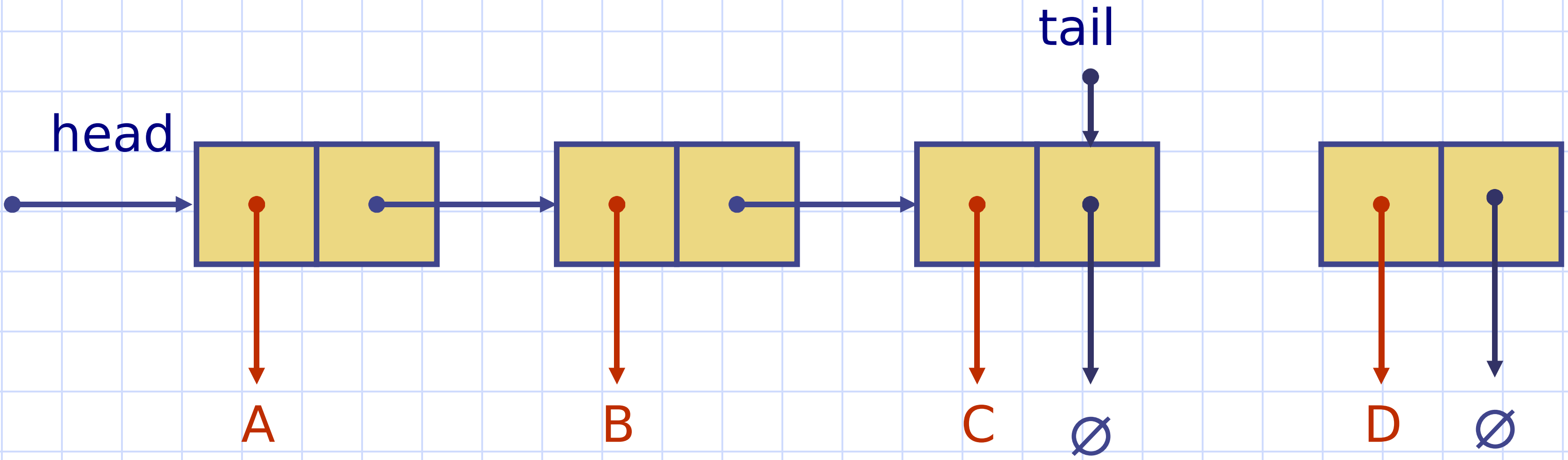
1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node



© 2006 Goodrich, Tamassia

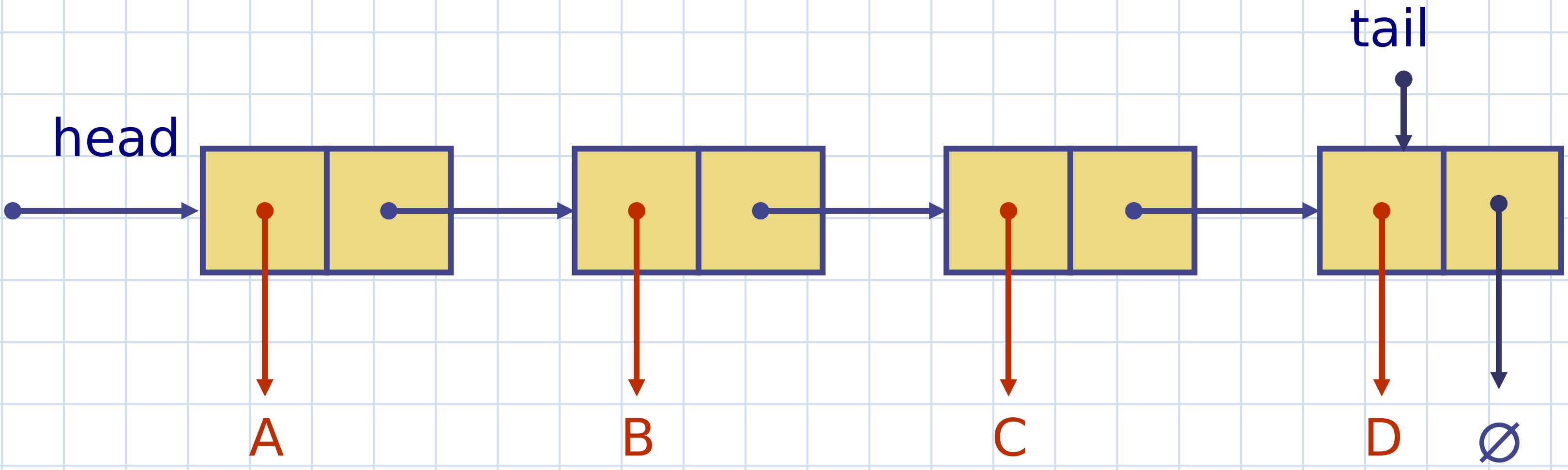
Inserting at the Tail

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



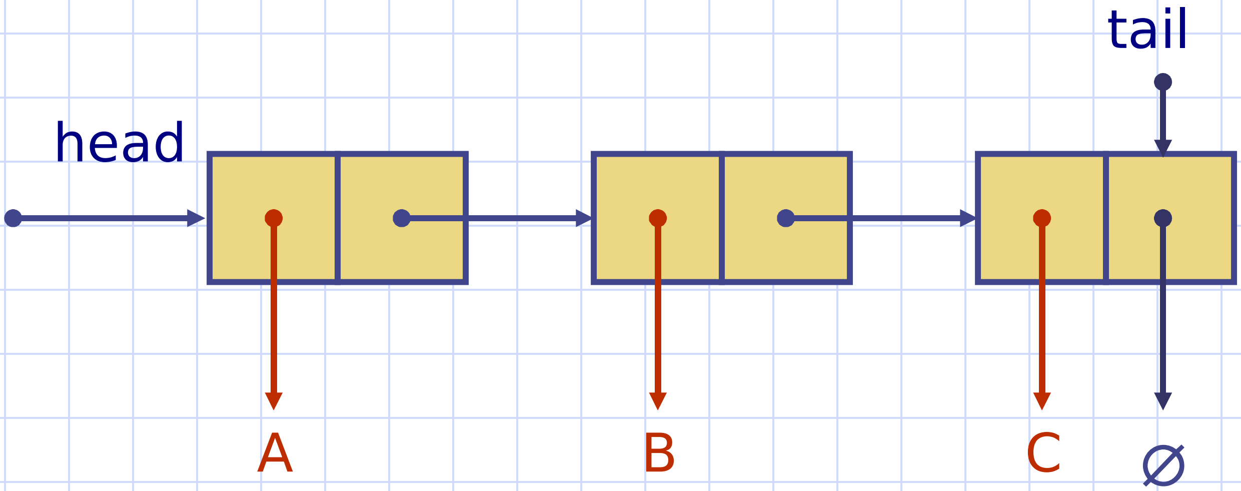
Inserting at the Tail

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



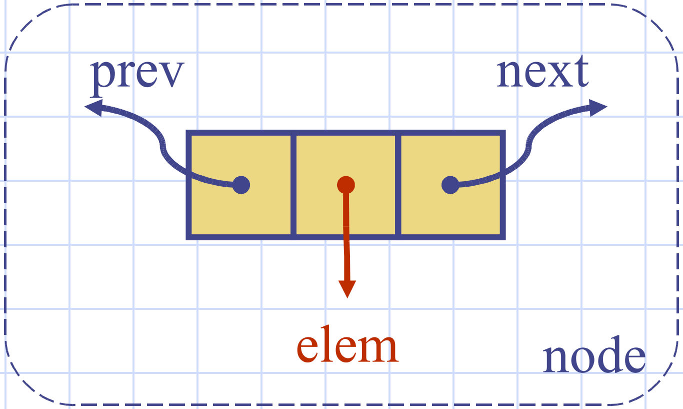
Removing at the Tail

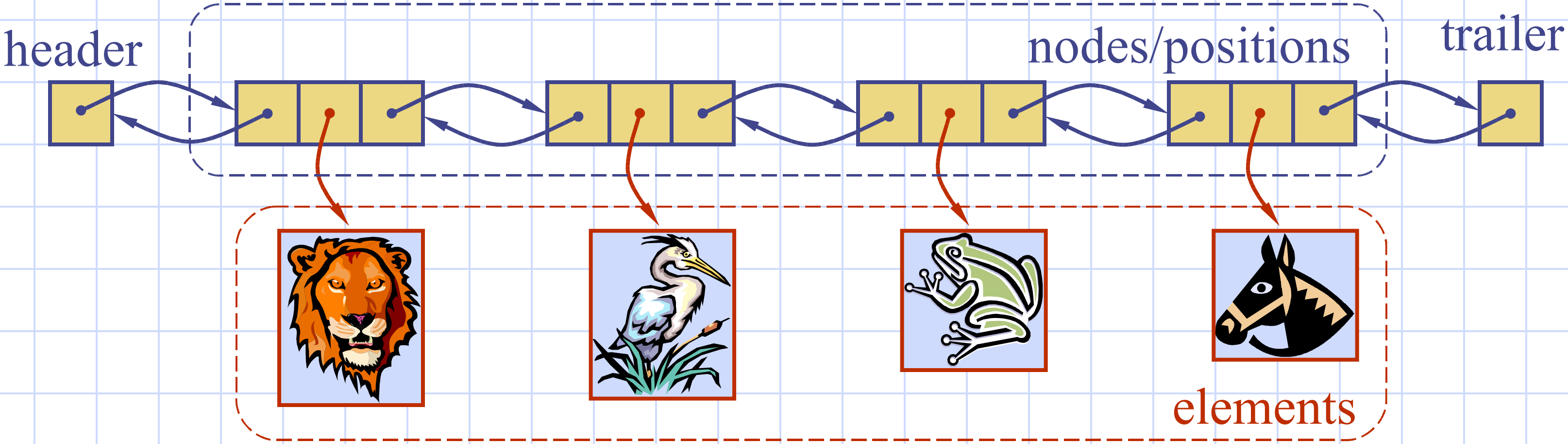
- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node



Doubly Linked List

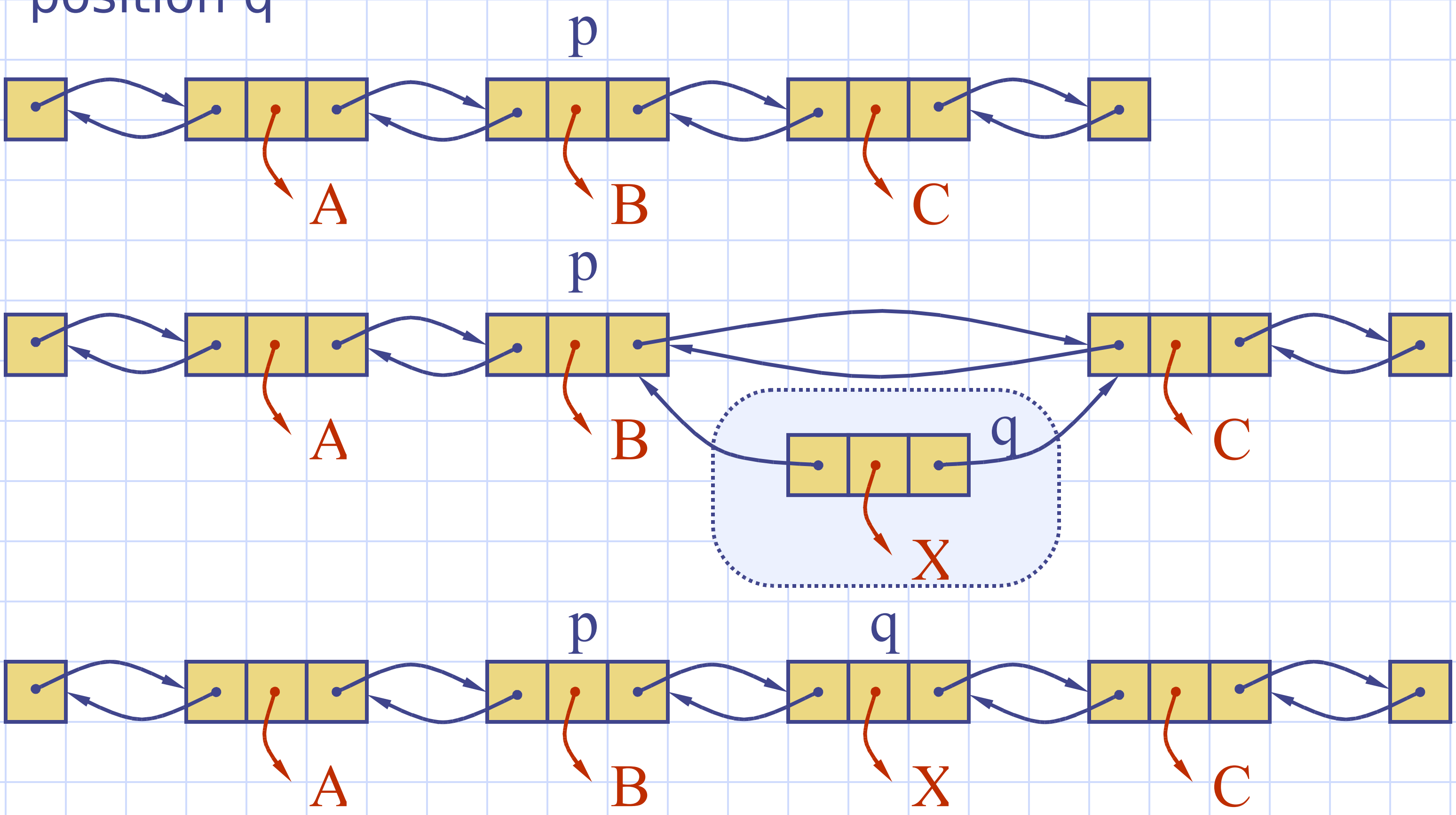
- A doubly linked list provides a natural implementation of the Node List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





Insertion

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

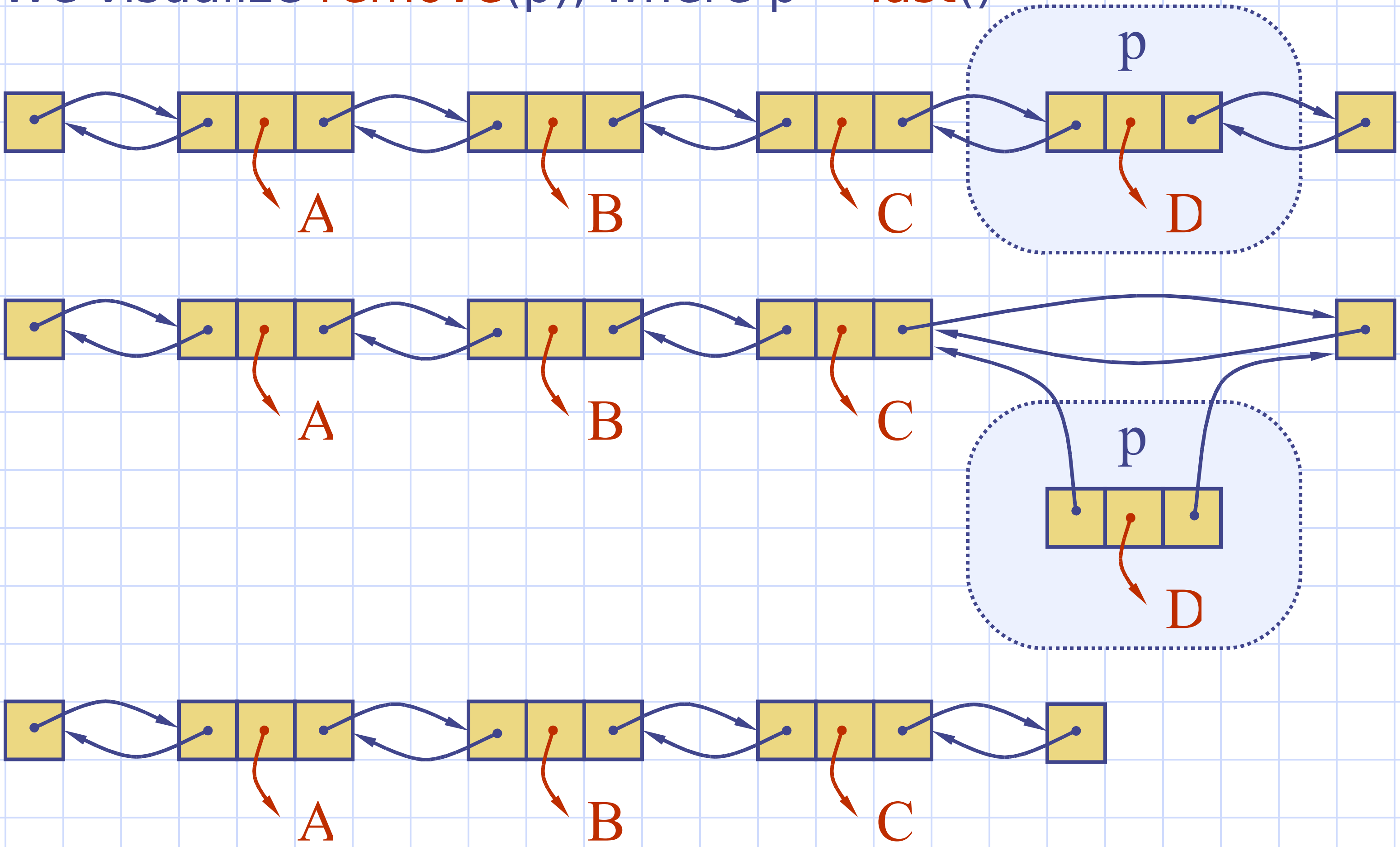


Insertion Algorithm

```
Algorithm addAfter(p,e):
    Create a new node cur;
    next = p.getNext();
    cur.setElement(e);
    cur.setPrev(p);              // link cur to its predecessor
    cur.setNext(next);           // link cur to its successor
    next.setPrev(cur)            // link next  (the old p successor
                                    to cur
    p.setNext(cur)               // link p to its new successor,cur
    return v                     // the position for the element e
```

Deletion

We visualize remove(p), where p = last()



Deletion Algorithm

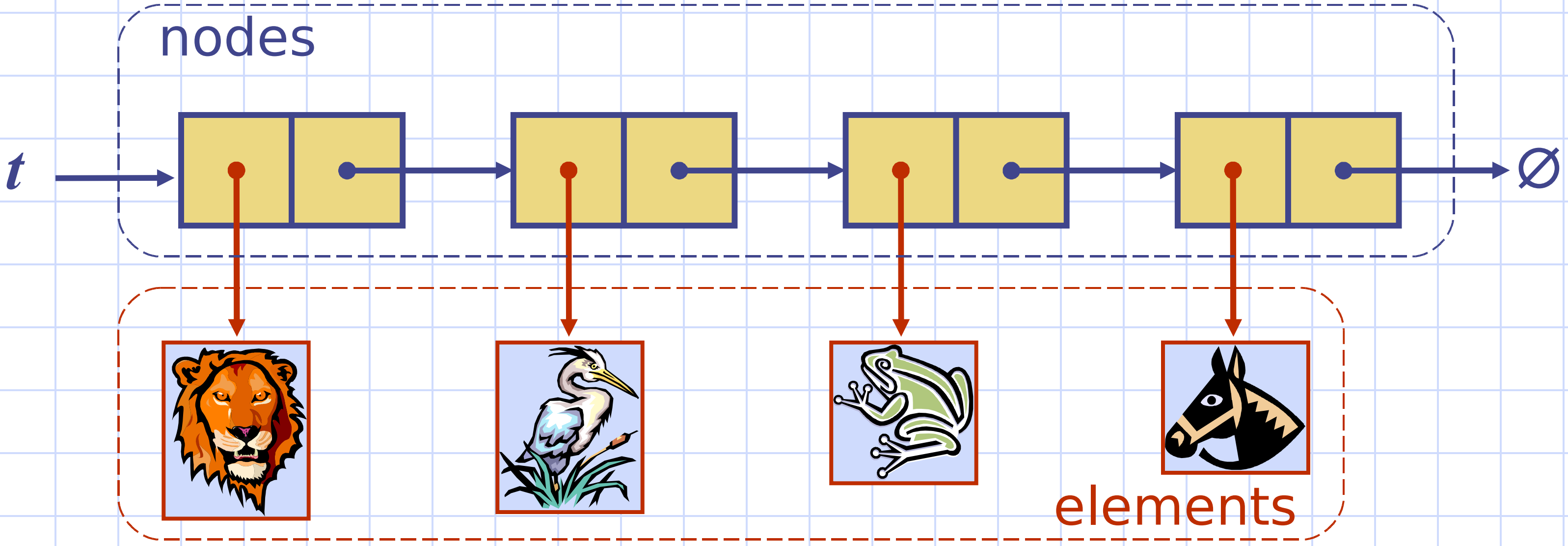
```
Algorithm remove(p):
    t = p.element              // a temporary variable
    prev = p.getPrev();
    next = p.getNext();
    prev.setNext(next)      // linking out p
    next.setPrev(prev)

    p.setPrev(null)            // invalidating the position p}
    p.setNext(null)
    return t
```

Performance

- In the implementation of the List ADT by means of a doubly linked list
    - The space used by a list with n elements is $O(n)$
    - The space used by each position of the list is $O(1)$
    - All the operations of the List ADT run in $O(1)$ time
    - Operation element() of the Position ADT runs in $O(1)$ time

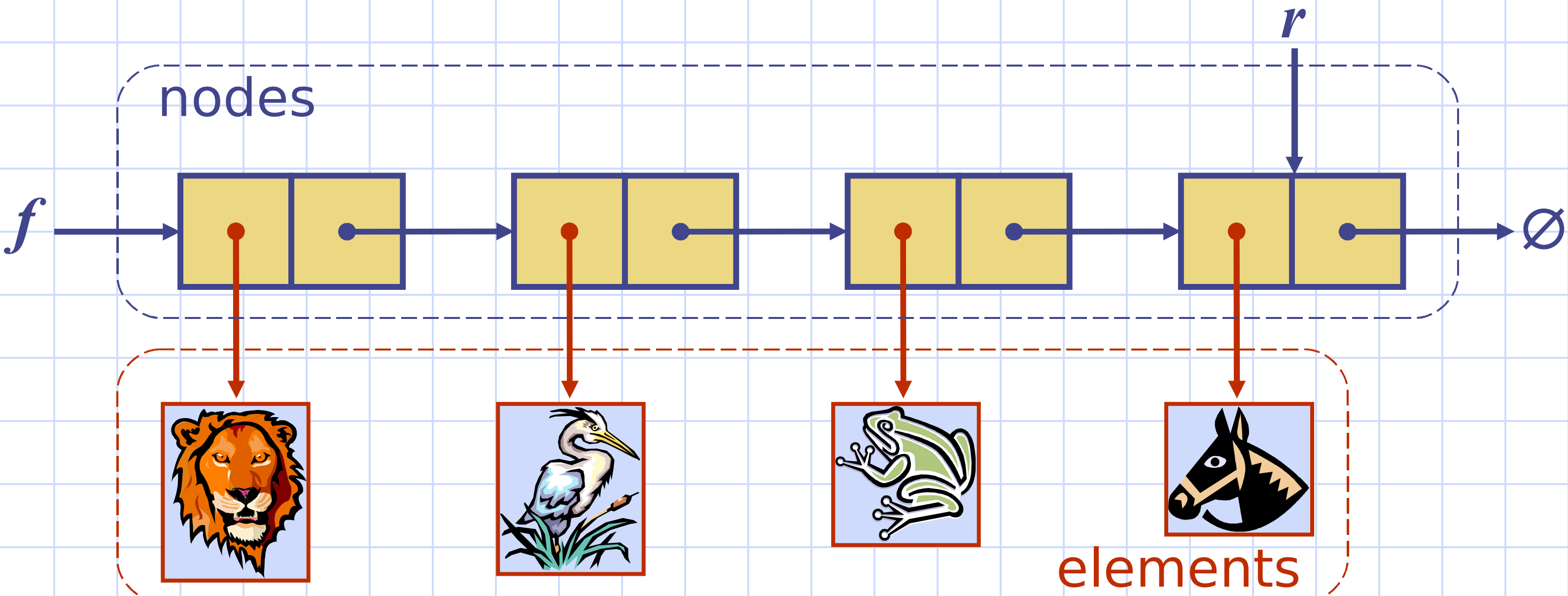
Stack as a Linked List

- We can implement a stack with a singly linked list
- The top element is stored at the first node of the list
- The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time



Queue as a Linked List

- We can implement a queue with a singly linked list (front = first element, rear the last one)
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time


r
nodes
f
∅
elements