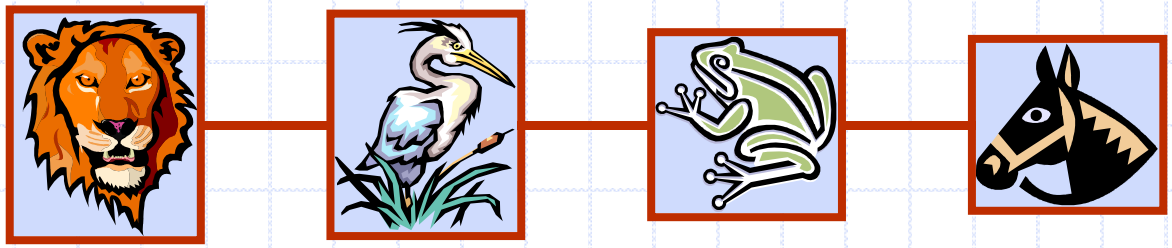
# Βασικές Δομές
# Arrays, Συνδεδεμένες Λίστες



# Πίνακες

Η πιο θεμελιώδης δομή δεδομένων που υποστηρίζεται από τις περισσότερες γλώσσες προγραμματισμού είναι ο πίνακας (array).

Ένας πίνακας αποτελεί μια σταθερή συλλογή δεδομένων ίδιου τύπου τα οποία αποθηκεύονται ακολουθιακά και είναι προσπελάσιμα με την χρήση κάπου αριθμοδείκτη.

Οι πίνακες έχουν μια ευθεία αντιστοιχία με τα συστήματα μνήμης.

Ουσιαστικά σαν δομή ο πίνακας είναι ένα όνομα και ένας δείκτης που καθορίζει την κάθε θέση του πίνακα.

Οι πίνακες έχουν ευρεία εφαρμογή επειδή έχουν άμεση αντιστοιχία με φυσικές μεθόδους οργάνωσης δεδομένων για εφαρμογές

# Ταξινόμηση με εισαγωγή

Αλγόριθμος InsertionSort(A)

Input: Ένας πίνακας A με n στοιχεία

Output: Ο πίνακας A με αναδιάταξη των στοιχείων σε μη φθίνουσα σειρά

for i←1 to n-1 do

εισαγωγή του A[i] στη σωστή του θέση σε σχέση στα A[0], A[1],…,A[i-1]

# Με μεγαλύτερη Λεπτομέρεια

Αλγόριθμος InsertionSort(A)

Input: Ένας πίνακας A με n στοιχεία

Output: Ο πίνακας A με αναδιάταξη των στοιχείων σε μη φθίνουσα σειρά

for i←1 to n-1 do

{εισαγωγή του A[i] στη σωστή του θέση σε σχέση στα A[0], A[1],…,A[i-1]}

cur ← A[i]  j ←i-1

while j≥0 and A[j] > cur do

A[j+1] ←A[j]

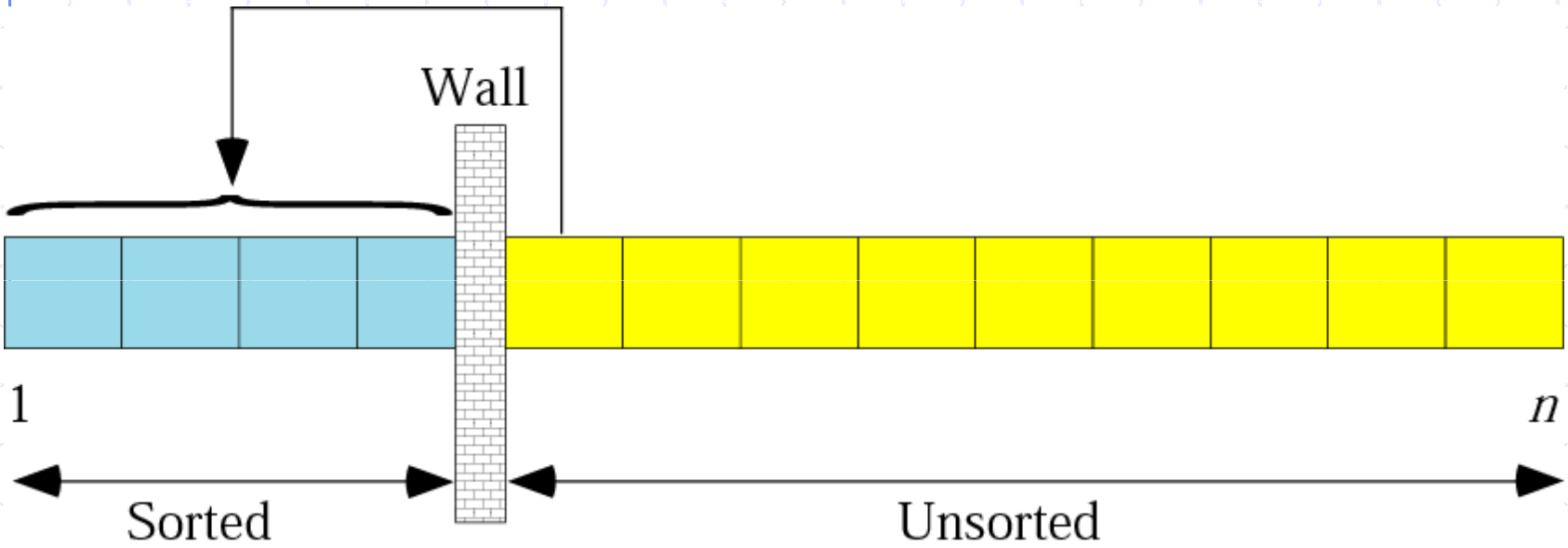
j ←j-1

A[j+1] ←cur

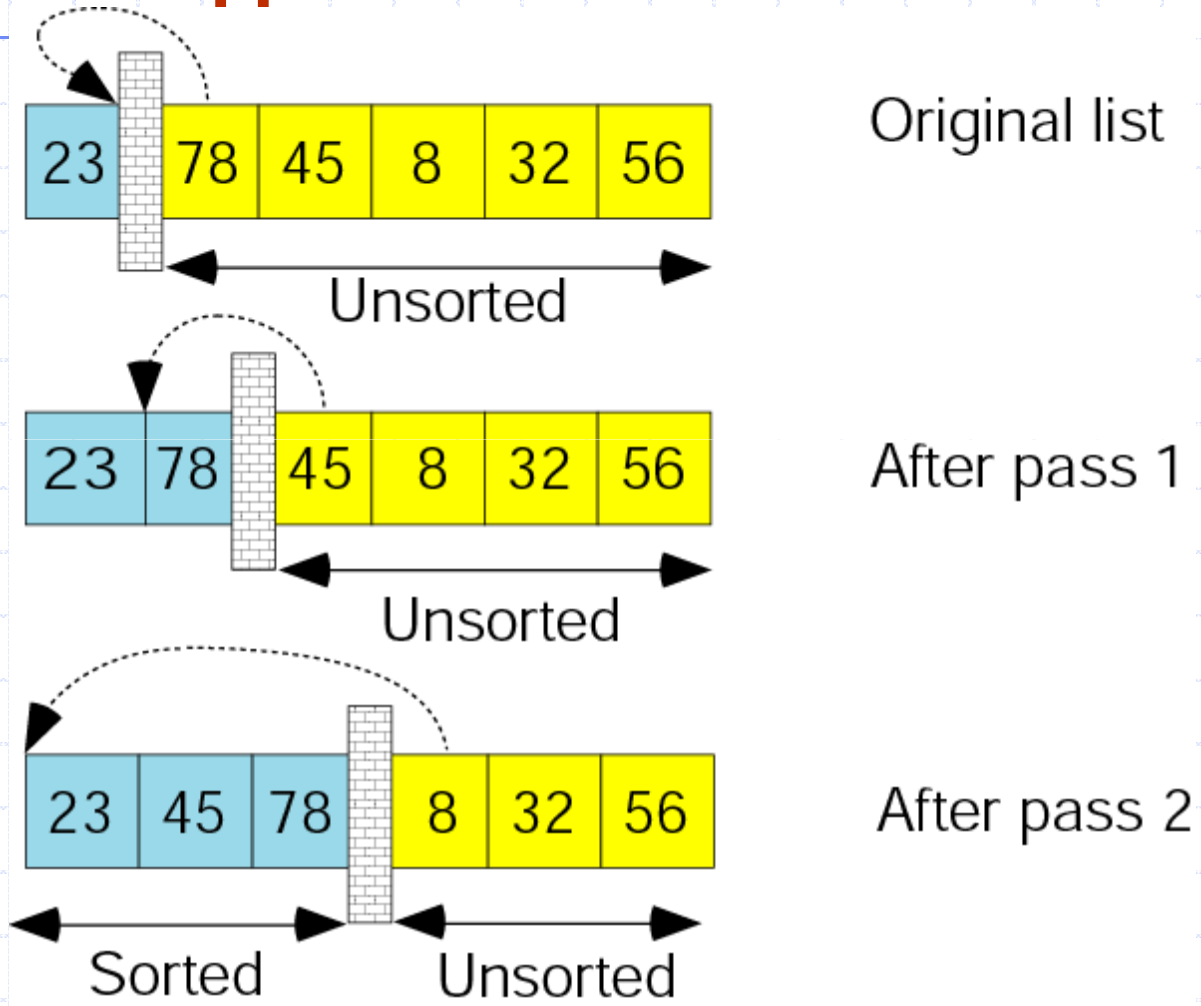
# Σε java

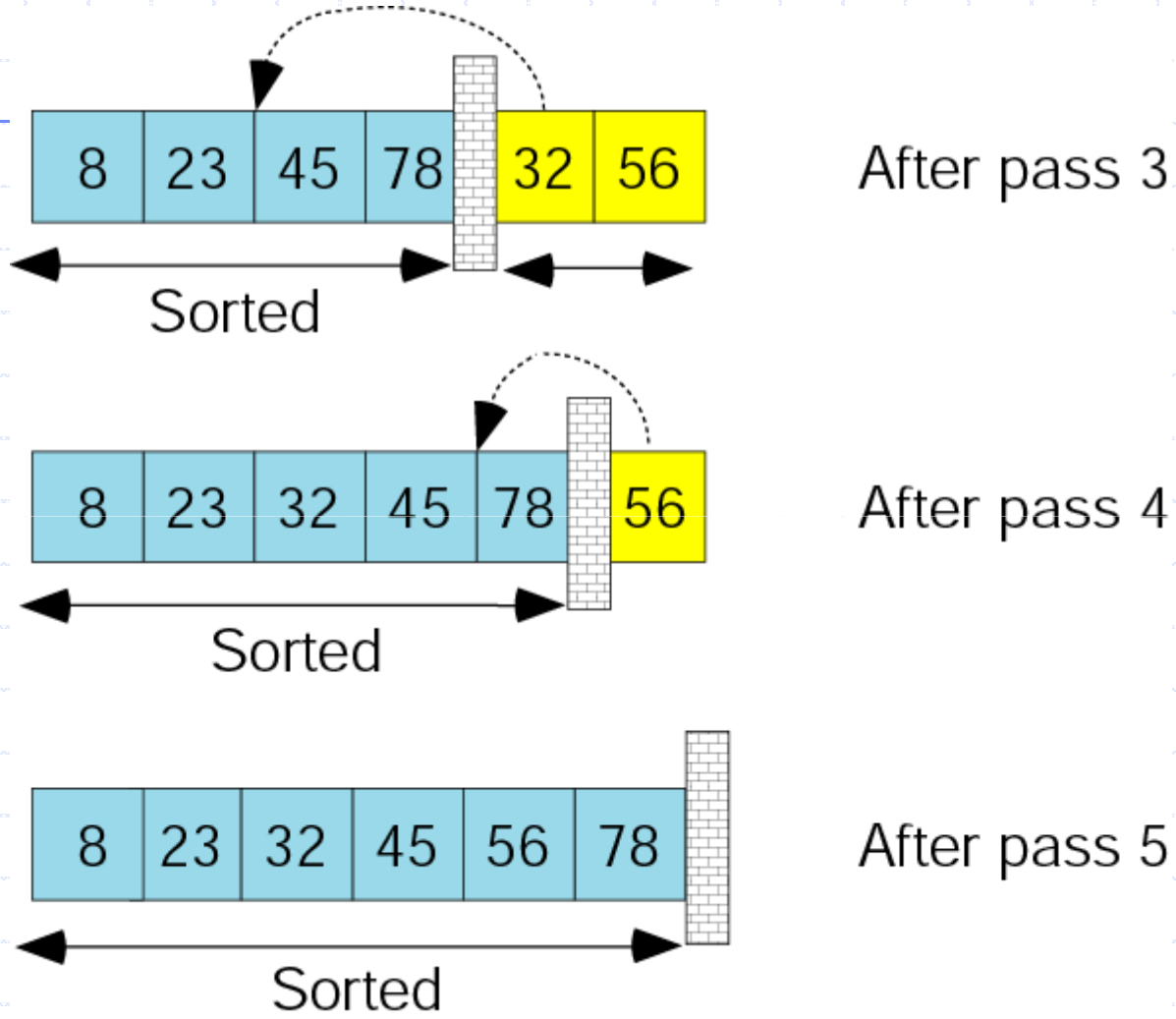
```
public static void insertionSort(char[] a) {
  int n= a.length;
  for (int i=1; i<n; i++0 {
     char cur=a[i];
      int j=i-1;
    while (j>=0) && (a[j]>cur))
      a[j+1]=a[j--];
  a[j+1]=cur;
   }
}
```

# Insertion Sort



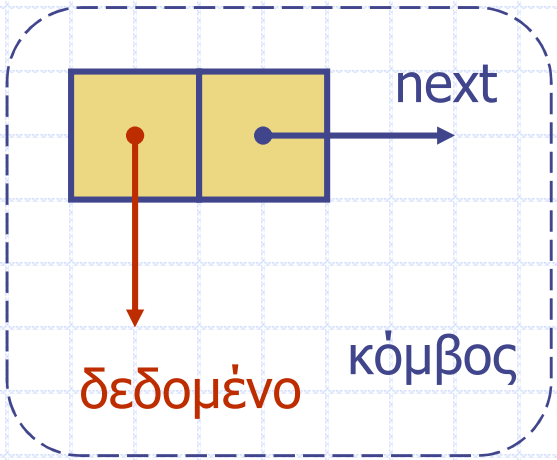
# Παράδειγμα

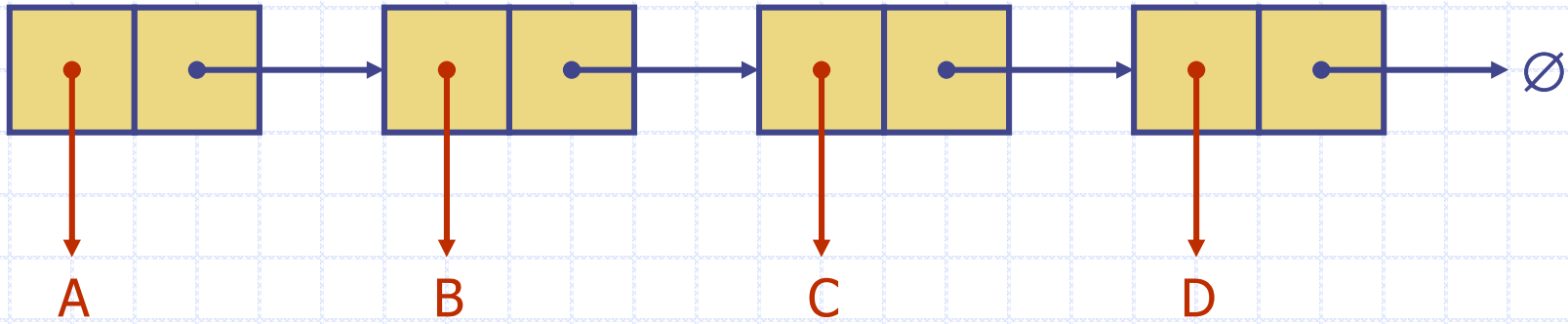

23
78
45
8
32
56
Unsorted
Original list
23
78
45
8
32
56
Unsorted
After pass 1
23
45
78
8
32
56
Sorted
Unsorted
After pass 2


8 23 45 78 32 56
Sorted
After pass 3
8 23 32 45 78 56
Sorted
After pass 4
8 23 32 45 56 78
Sorted
After pass 5

# Απλά Συνδεδεμένη Λίστα

- Μια απλά συνδεδεμένη λίστα είναι μια συμπαγής δομή δεδομένων που αποτελείται από μια ακολουθία κόμβων
- Κάθε κόμβος αποθηκεύει
  - δεδομένο
  - Σύνδεσμο στον επόμενο κόμβο



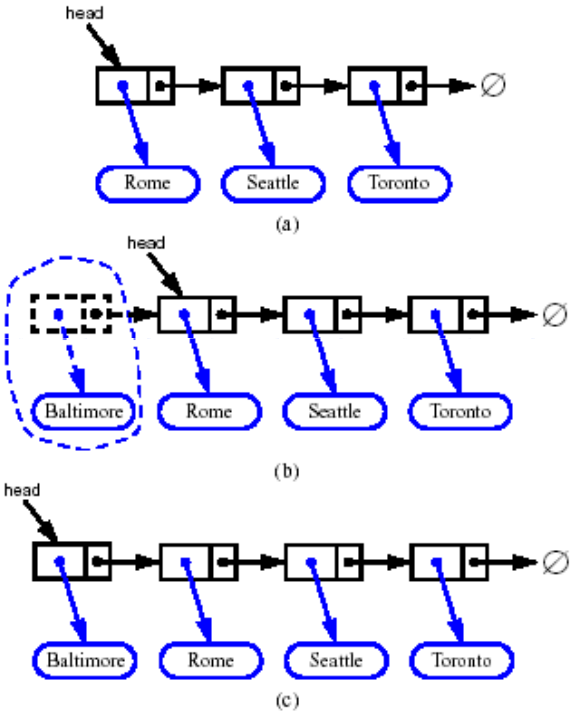



# Η κλάση κόμβος για κόμβους της λίστας

```
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;
  }
}
```

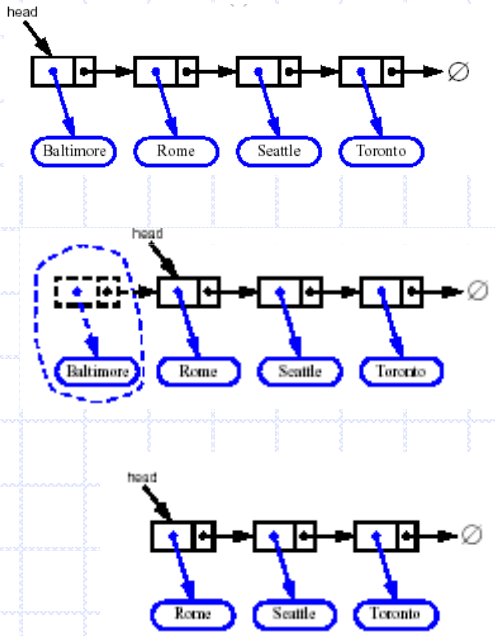
# Εισαγωγή στην αρχή

1. Παροχή ενός νέου κόμβου
2. Εισαγωγή νέου στοιχείου
3. Ο νέος κόμβος δείχνει την παλαιά κεφακλή
4. Εημέρωση του  head να δείχνει το νέο κόμβο



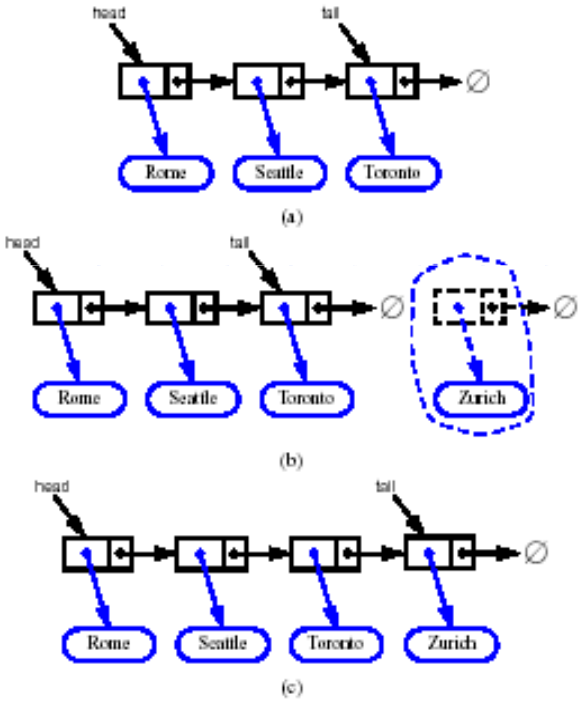
# Διαγραφή από την κεφαλή

1. Τροποποίηση της κεφαλής να δείχνει στον επόμενο κόμβο της λίστας
2. Εοιστροφή του κόμβου που διαγράφεται στους διαθέσιμους (στη Java αυτόματα)



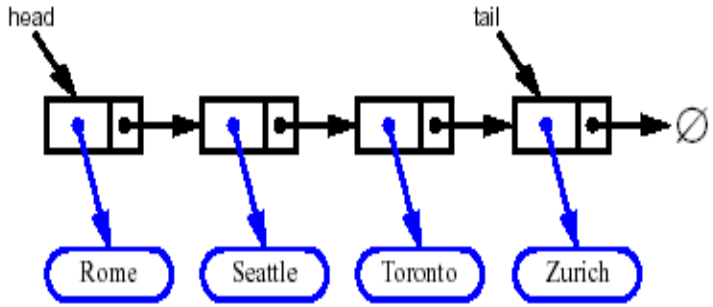
# Εισαγωγή στο τέλος

1. Δημιουργία του νέου κόμβο
2. Εισαγωγή των δεδομένων
3. Ο νέος κόμβος να δείχνει null
4. Ο παλαιός τελευταίος να δείχνει τον νέο κόμβο



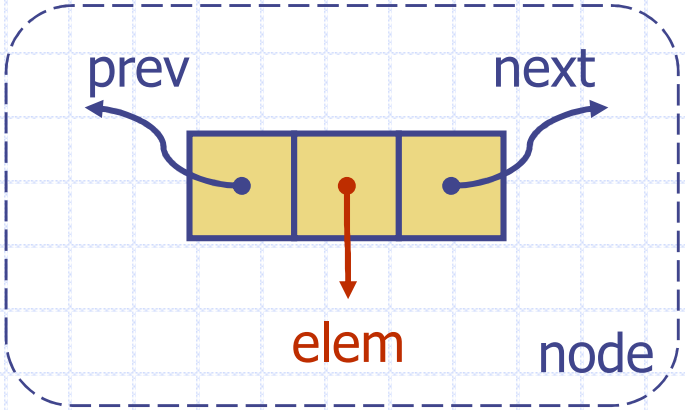
# Διαγραφή από το τέλος

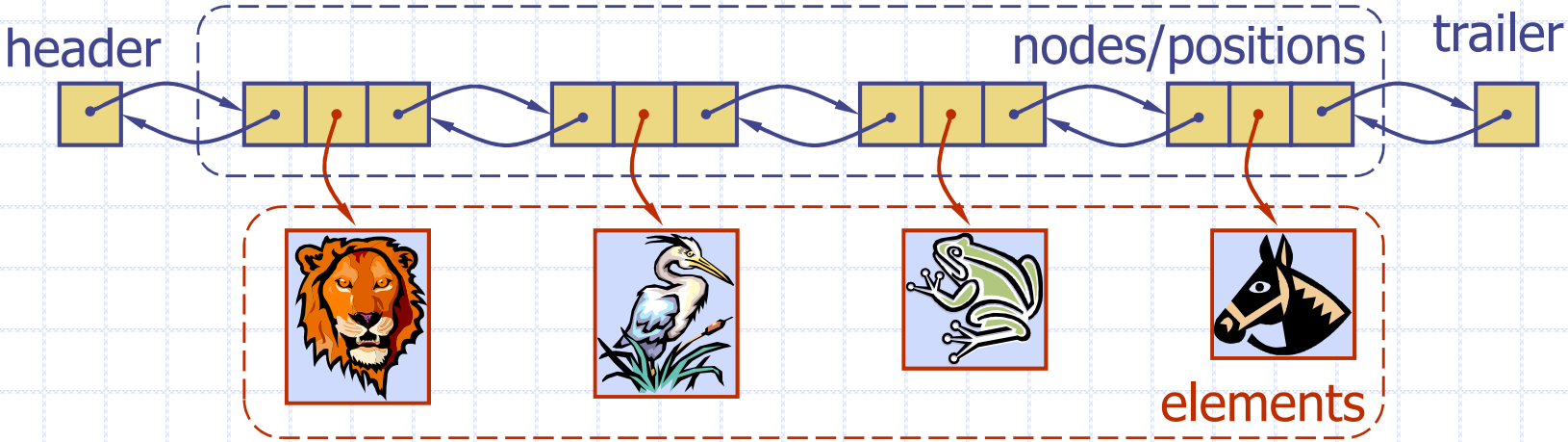
- Η πράξη της διαγραφής από το τέλος δεν είναι αποτελεσματική!
- Δεν υπάρχει τρόπος σε σταθερό χρόνο να αλλάζει το τέλος να δείχνει στον προηγούμενο κόμβο.



# Διπλά συνδεδμένη λίστα

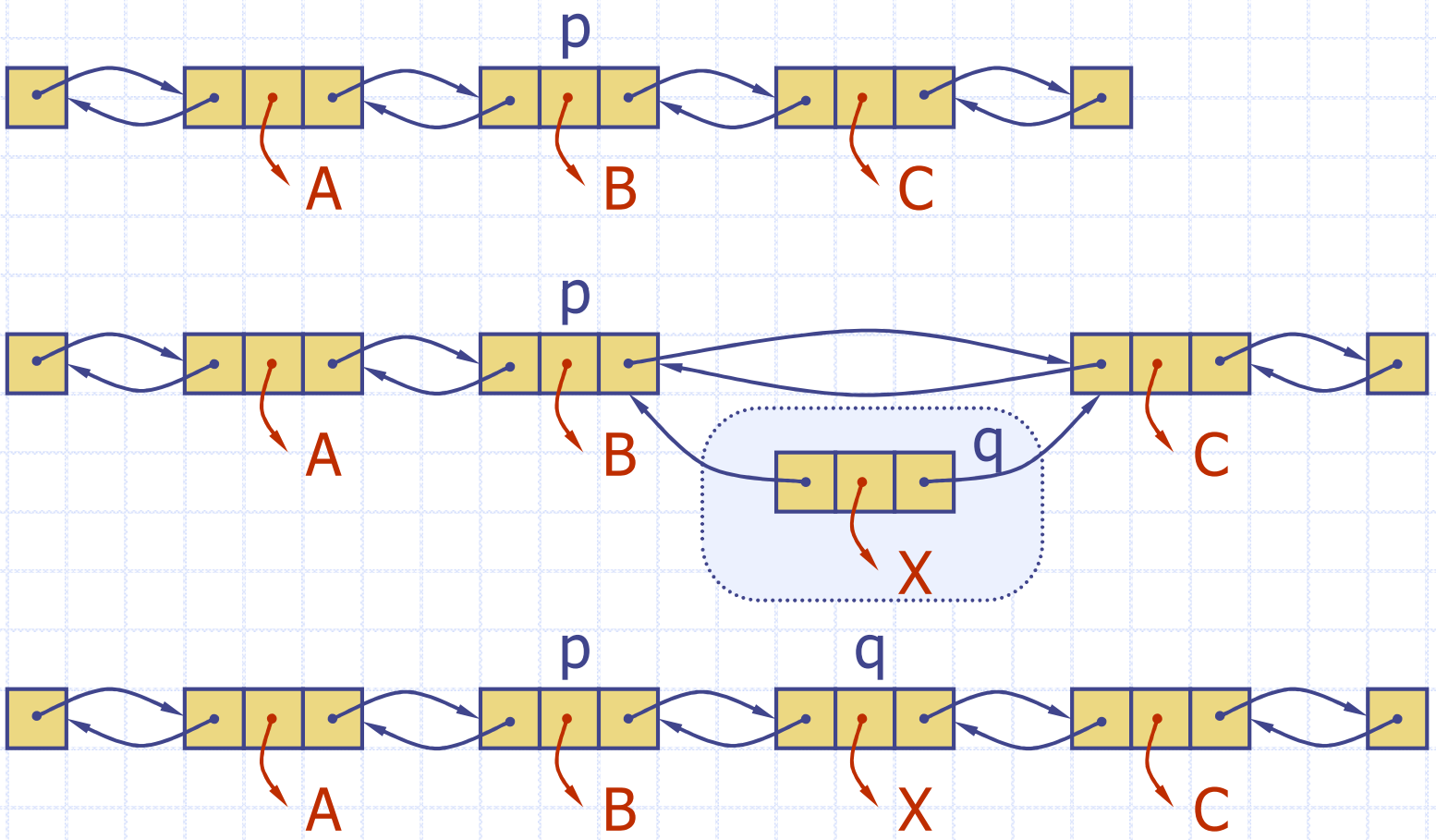
- Μια διπλά συνδεδεμένη λίστα αποτελεί τη φυσική υλοποίηση του ΑΤΔ λιστας κομβων
- Οι κόμβοι υλοποιούν τη θέση και αποθηκευουν:
  - στοιχείο
  - σύνδεσμο στον προηγούμενο κόμβο
  - Σύνδεσμο στον επόμενο κόμβοτο
- Ειδικούς κόμβους αρχής και τέλους



# Εισαγωγή

- Οπτικοποίησση της εισαγωγής insertAfter(p, X), που επιστρέφει τη θέση q



# Αλγόριθμος Εισαγωγής

**Algorithm** addAfter(p,e):

Create a new node v

v.setElement(e)

v.setPrev(p) {σύνδεση του v με τον προηγούμενό του}

v.setNext(p.getNext()) {σύνδεση του v με τον επόμενο του}

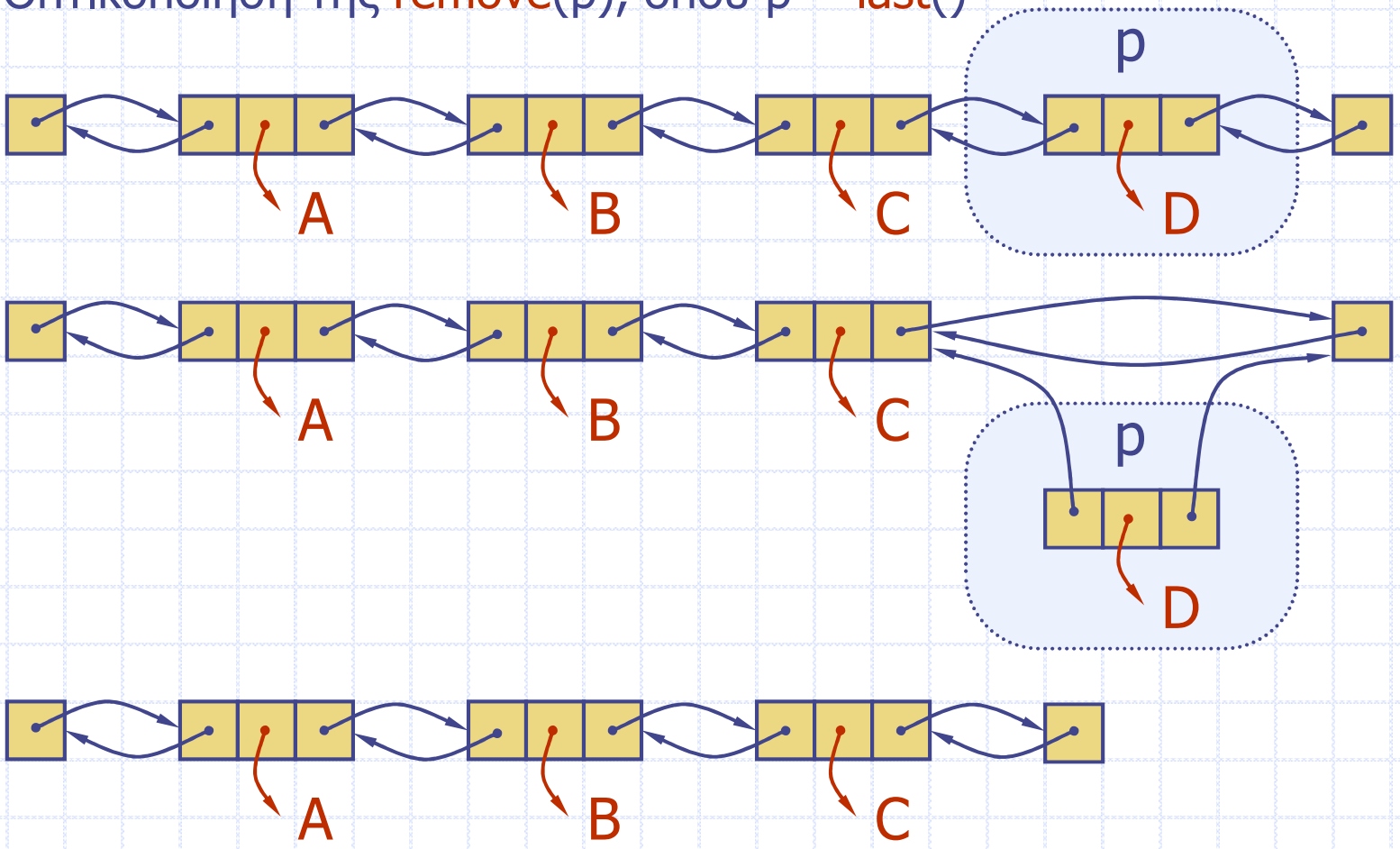
(p.getNext()).setPrev(v) {σύνδεση του παλαιού διαδόχου του p με το παλαιό v}

p.setNext(v) {σύνδεση του p με τον νέο διάδοχό του , v}

**return** v {τη θέση του στοιχείου e}

# Διαγραφή

- Οπτικοποίηση της remove(p), όπου p = last()



# Αλγόριθμος Διαγραφής

```
Algorithm remove(p):
    t = p.element      {μια προσωρινή μεταβλητή για την
    τιμή που επιστρέφεται }
    (p.getPrev()).setNext(p.getNext()) {σύνδεση με το p}
    (p.getNext()).setPrev(p.getPrev())
    p.setPrev(null)    {ακύρωση της θέσης p}
    p.setNext(null)
    return t
```

# Απόδοση

- Στην υλοποίηση του ΑΤΔ λίστας με χρήση διπλά συνδεδεμένης λίστας
  - Ο χώρος που απαιτεί μια λίστα με $n$ στοιχεία είναι $O(n)$
  - Ο χώρος που απαιτεί κάθε θέση της λίστας είναι $O(1)$
  - Όλες οι πράξεις του ΑΤΔ λίστας τρέχουν σε χρόνο $O(1)$
  - Η πράξη element() του ΑΤΔ τρέχει σε $O(1)$ χρόνο

# Η ουρά σαν συνδεδεμένη λίστα

- Μπορούμε να υλοποιήσουμε μια ουρά με μια απλά σνδεδεμάνη λίστα
  - Το στοιχείο της αρχής αποθηκελυεται στον πρώτο κόμβο
  - Το στοιχείο του τέλους πάει στον τελευταίο κόμβο
- Ο απαιτούμενος χώρος είναι $O(n)$ ακαι κάθε πράξη της ουράς απαιτεί χρόνο $O(1)$