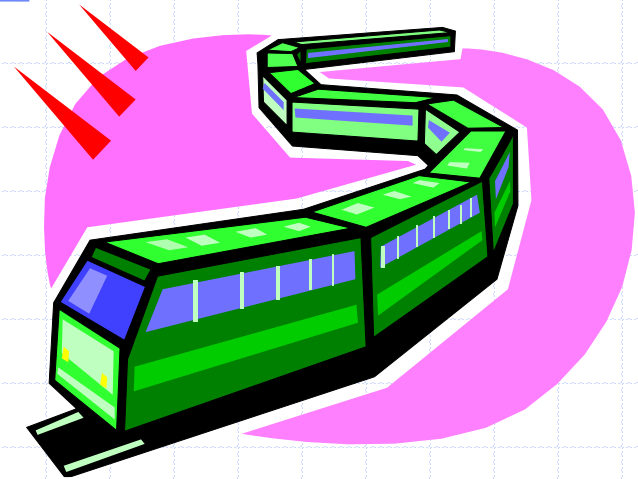
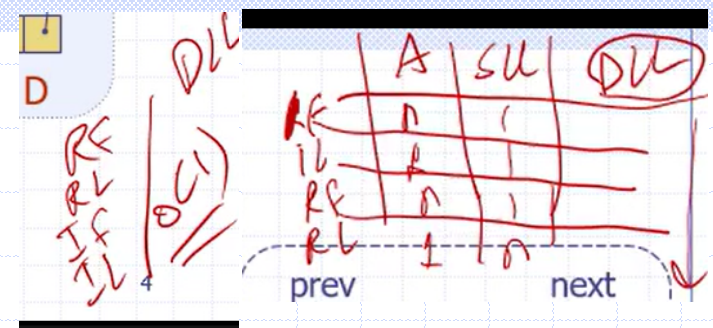


Presentation for use with the textbook **Data Structures and Algorithms in Java, 6th edition**, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

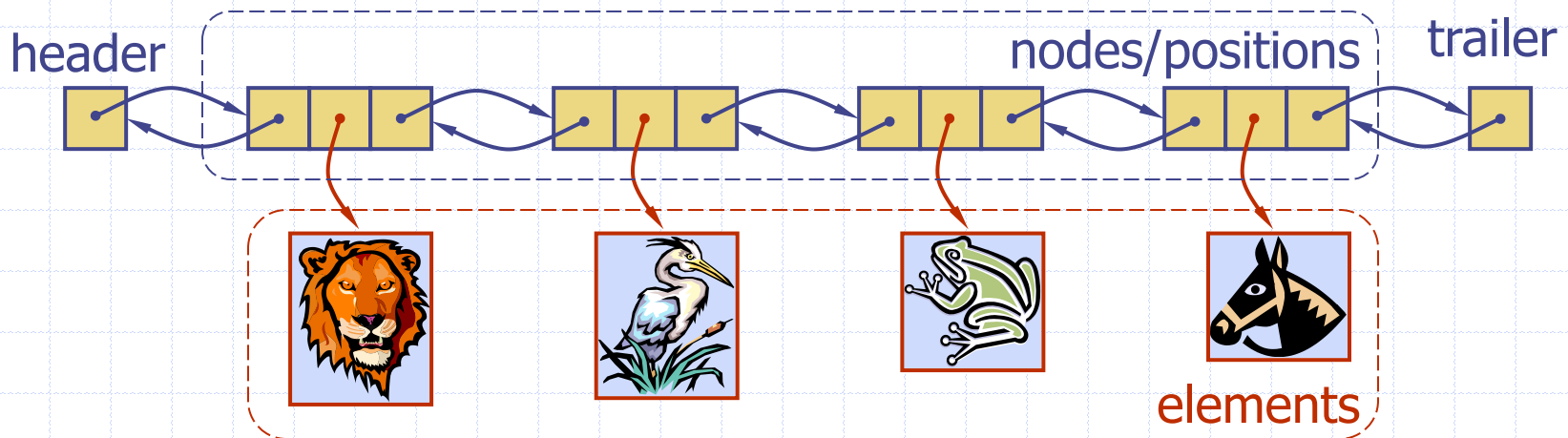
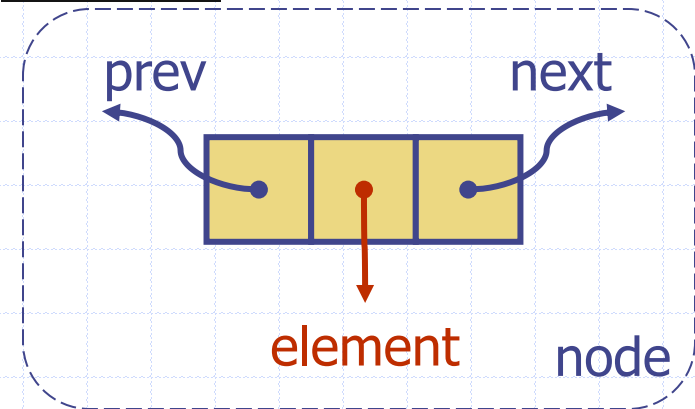
Doubly Linked Lists



Doubly Linked List

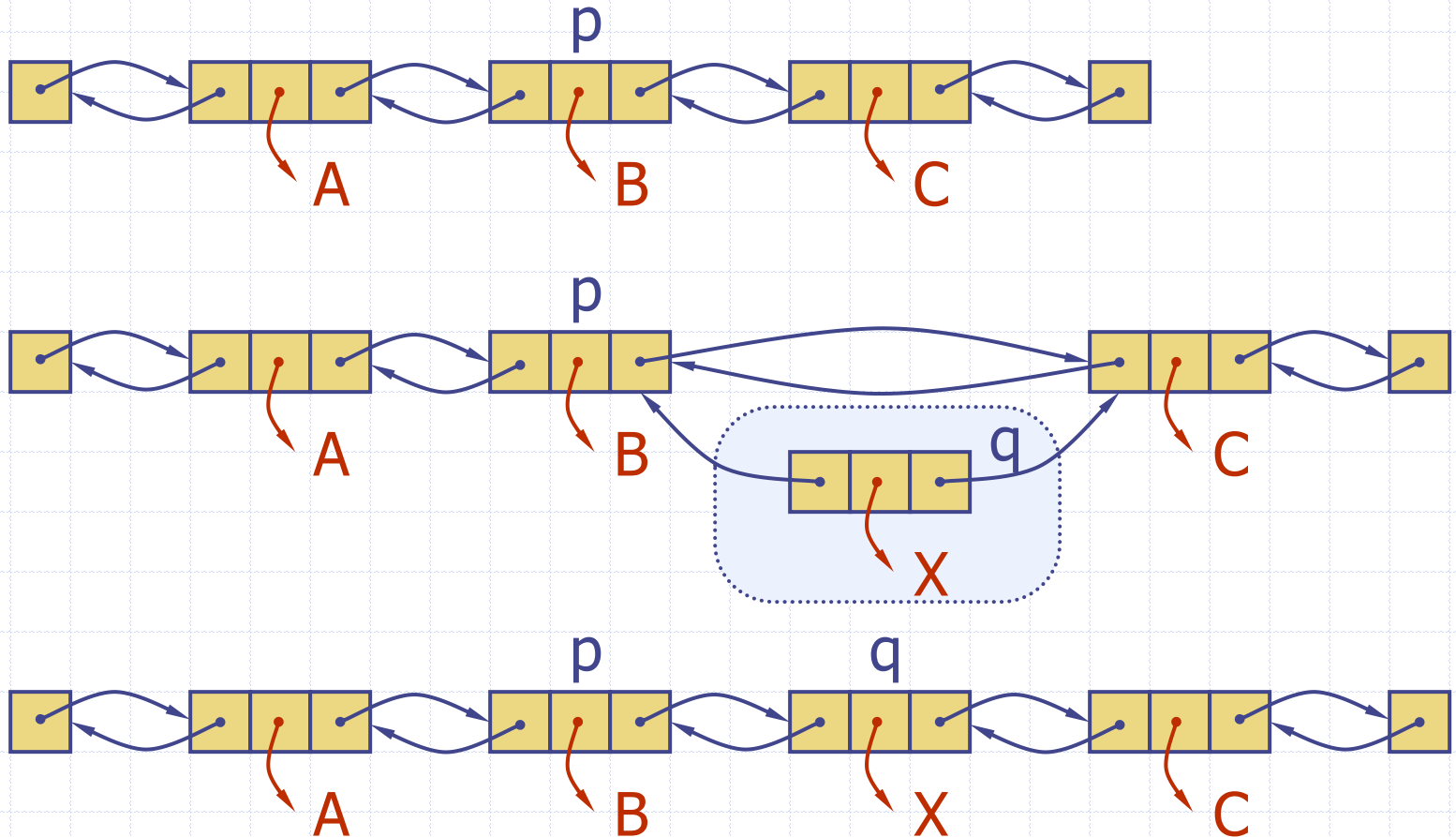


- ❑ A doubly linked list can be traversed forward and backward
- ❑ Nodes store:
 - element
 - link to the previous node
 - link to the next node
- ❑ Special trailer and header nodes



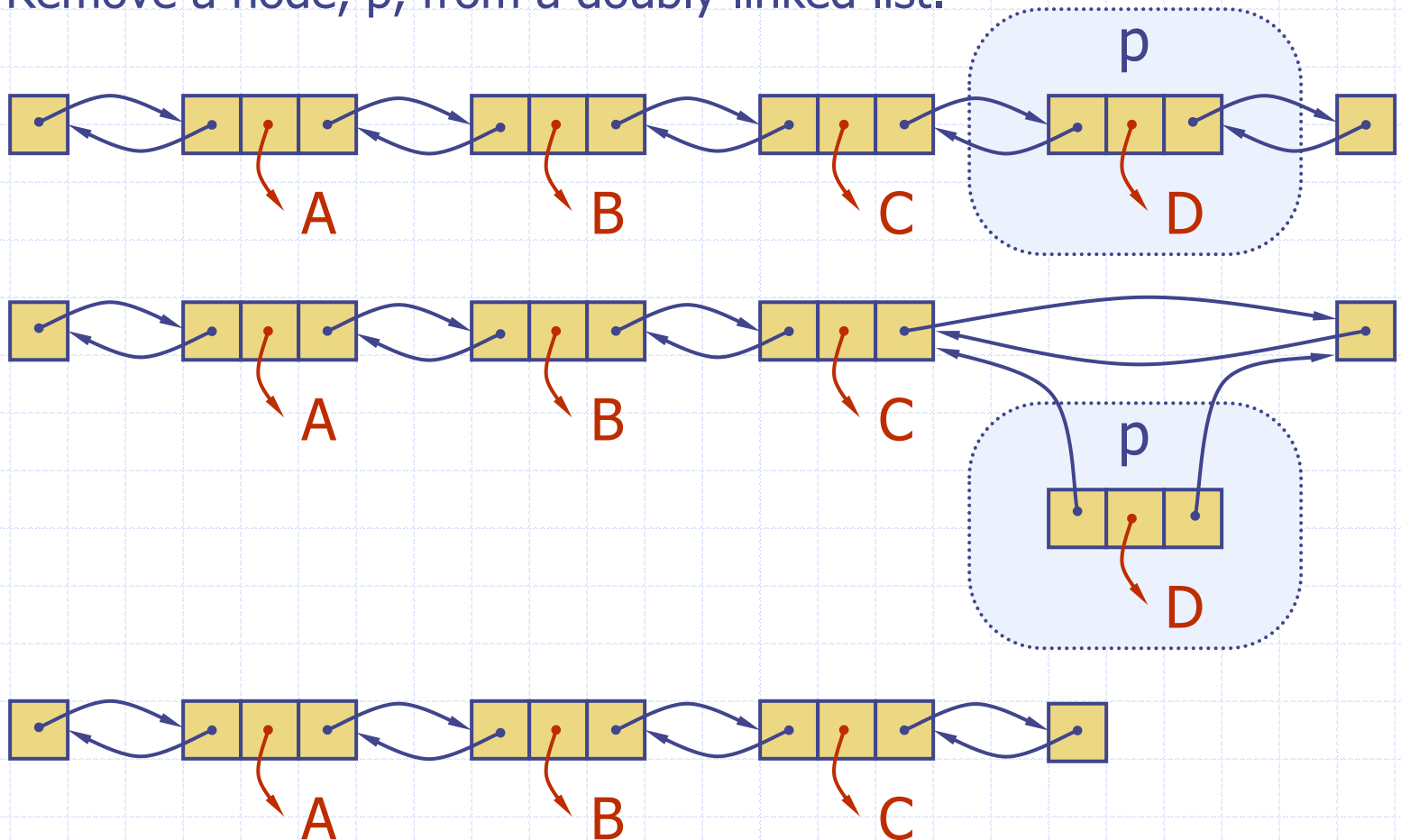
Insertion

- Insert a new node, q , between p and its successor.



Deletion

- Remove a node, p , from a doubly linked list.



Doubly-Linked List in Java

```
1  /** A basic doubly linked list implementation. */
2  public class DoublyLinkedList<E> {
3      //----- nested Node class -----
4      private static class Node<E> {
5          private E element;           // reference to the element stored at this node
6          private Node<E> prev;        // reference to the previous node in the list
7          private Node<E> next;        // reference to the subsequent node in the list
8          public Node(E e, Node<E> p, Node<E> n) {
9              element = e;
10             prev = p;
11             next = n;
12         }
13         public E getElement() { return element; }
14         public Node<E> getPrev() { return prev; }
15         public Node<E> getNext() { return next; }
16         public void setPrev(Node<E> p) { prev = p; }
17         public void setNext(Node<E> n) { next = n; }
18     } //----- end of nested Node class -----
19 }
```

Doubly-Linked List in Java, 2

```
21 private Node<E> header;           // header sentinel
22 private Node<E> trailer;          // trailer sentinel
23 private int size = 0;              // number of elements in the list
24 /** Constructs a new empty list. */
25 public DoublyLinkedList() {
26     header = new Node<>(null, null, null); // create header
27     trailer = new Node<>(null, header, null); // trailer is preceded by header
28     header.setNext(trailer);           // header is followed by trailer
29 }
30 /** Returns the number of elements in the linked list. */
31 public int size() { return size; }
32 /** Tests whether the linked list is empty. */
33 public boolean isEmpty() { return size == 0; }
34 /** Returns (but does not remove) the first element of the list. */
35 public E first() {
36     if (isEmpty()) return null;
37     return header.getNext().getElement(); // first element is beyond header
38 }
39 /** Returns (but does not remove) the last element of the list. */
40 public E last() {
41     if (isEmpty()) return null;
42     return trailer.getPrev().getElement(); // last element is before trailer
43 }
```

Doubly-Linked List in Java, 3

```
44 // public update methods
45 /** Adds element e to the front of the list. */
46 public void addFirst(E e) {
47     addBetween(e, header, header.getNext());    // place just after the header
48 }
49 /** Adds element e to the end of the list. */
50 public void addLast(E e) {
51     addBetween(e, trailer.getPrev(), trailer);    // place just before the trailer
52 }
53 /** Removes and returns the first element of the list. */
54 public E removeFirst() {
55     if (isEmpty()) return null;    // nothing to remove
56     return remove(header.getNext());    // first element is beyond header
57 }
58 /** Removes and returns the last element of the list. */
59 public E removeLast() {
60     if (isEmpty()) return null;    // nothing to remove
61     return remove(trailer.getPrev());    // last element is before trailer
62 }
```

Doubly-Linked List in Java, 4

```
64 // private update methods
65 /** Adds element e to the linked list in between the given nodes. */
66 private void addBetween(E e, Node<E> predecessor, Node<E> successor) {
67     // create and link a new node
68     Node<E> newest = new Node<>(e, predecessor, successor);
69     predecessor.setNext(newest);
70     successor.setPrev(newest);
71     size++;
72 }
73 /** Removes the given node from the list and returns its element. */
74 private E remove(Node<E> node) {
75     Node<E> predecessor = node.getPrev();
76     Node<E> successor = node.getNext();
77     predecessor.setNext(successor);
78     successor.setPrev(predecessor);
79     size--;
80     return node.getElement();
81 }
82 } //----- end of DoublyLinkedList class -----
```