Queues

Carlos Delgado Kloos

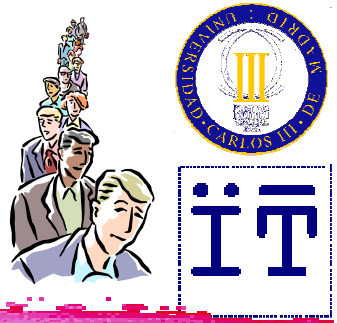
Dep. Ingeniería Telemática

Univ. Carlos III de Madrid

Example

- The queue at the bus stop
- The printer queue

Characteristics

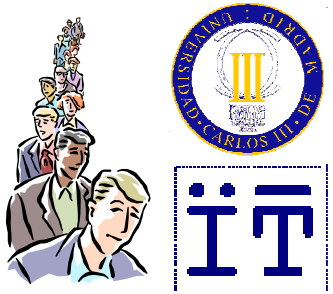


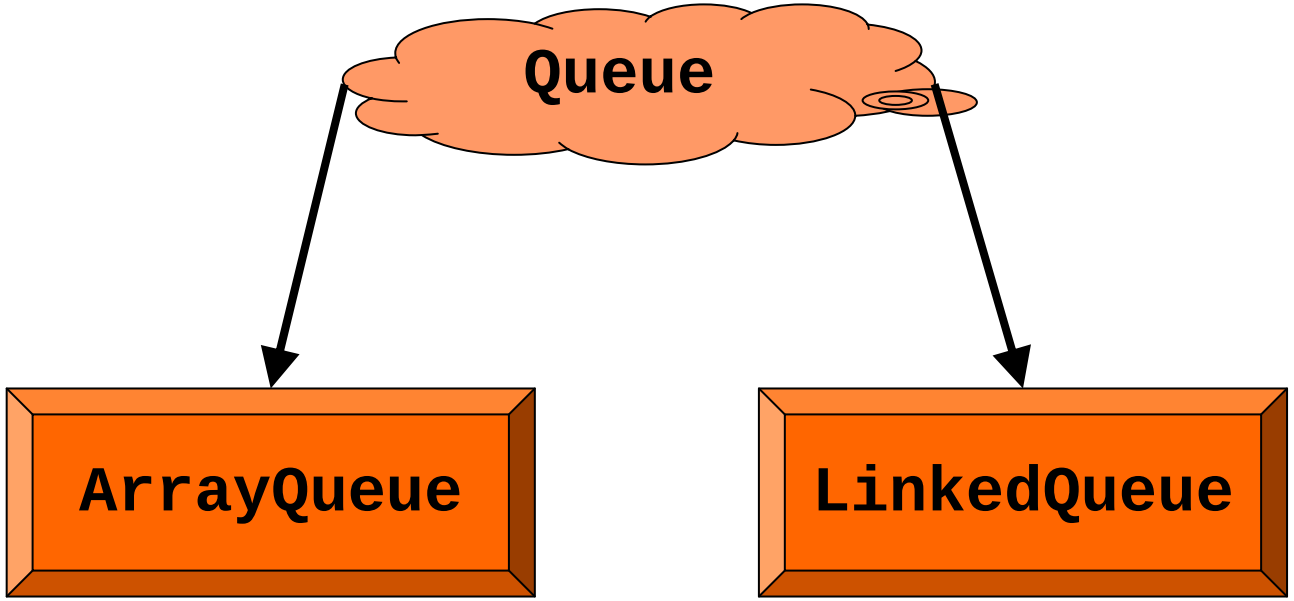
- Linear structure
- Access on one end for insertion and on the other for extraction

Queue interface

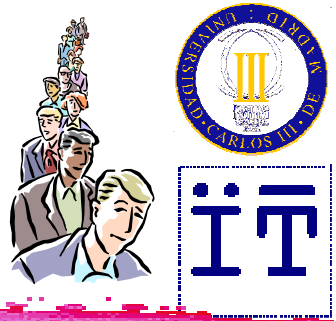
```
public interface Queue {
  public int size();
  public boolean isEmpty();
  public void enqueue(Object o)
    throws QueueOverflowException;
  public Object dequeue()
    throws EmptyQueueException;
  public Object front()
    throws EmptyQueueException;
}
```

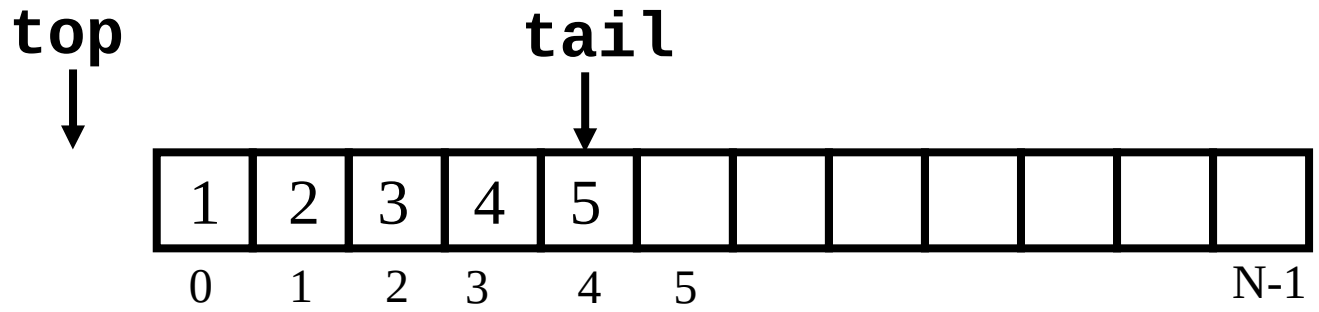
One interface and several implementations

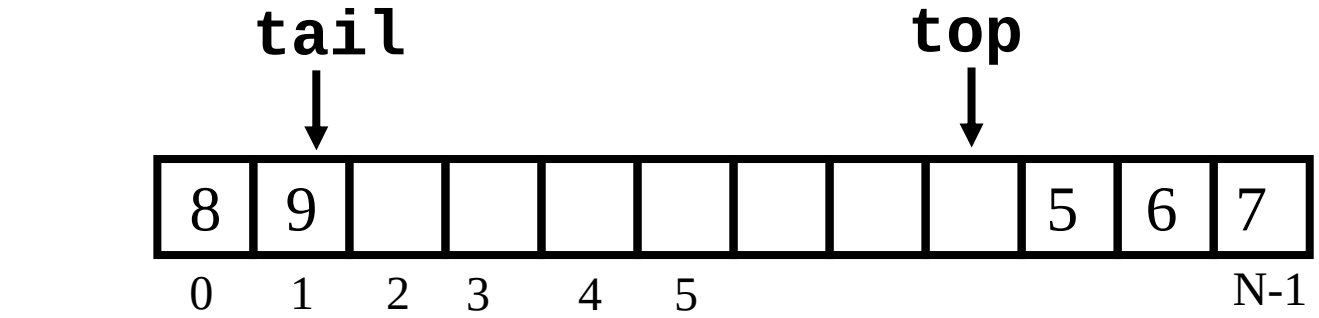




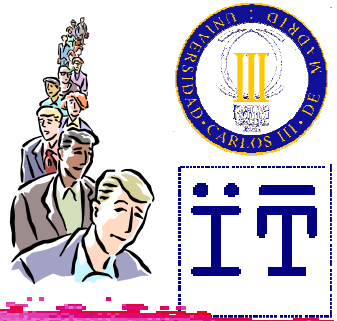
Array-based implementation

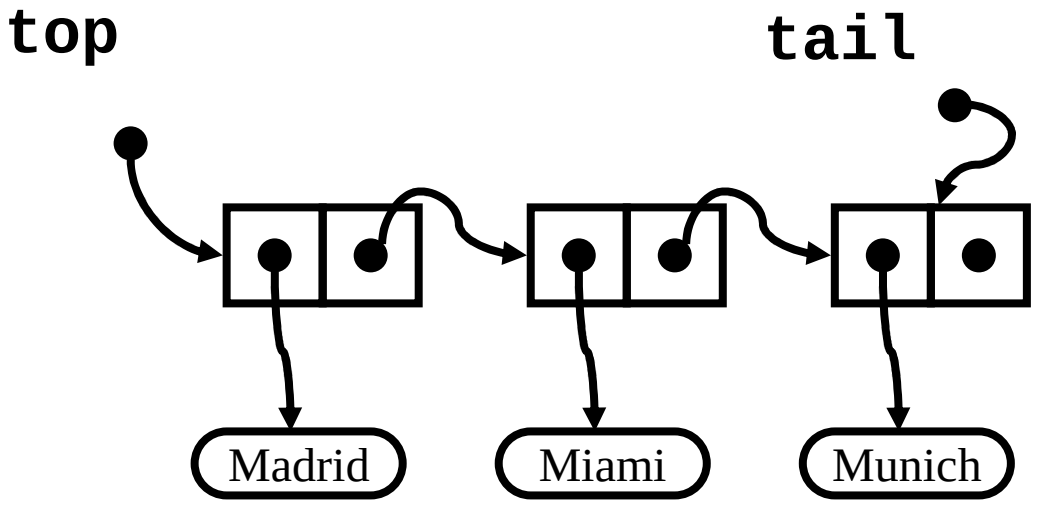




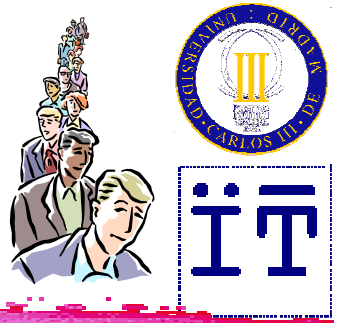


Implementation based on linked lists



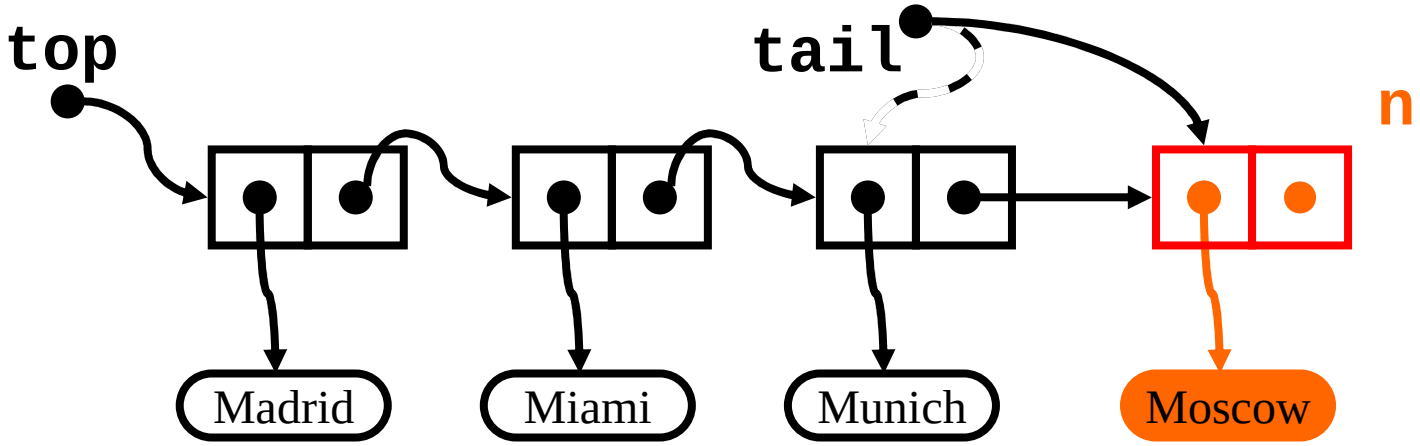


Implementation based on linked lists

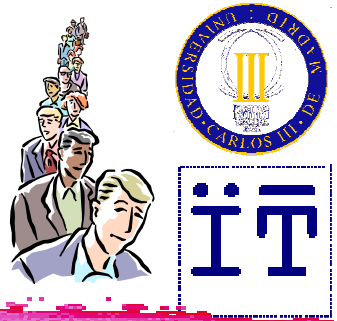


```
public class LinkedQueue implements Queue {
   private Node top = null;
   private Node tail = null;
   private int size = 0;

   (…)
}
```

Insertion (enqueue)

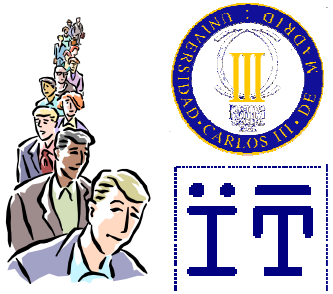


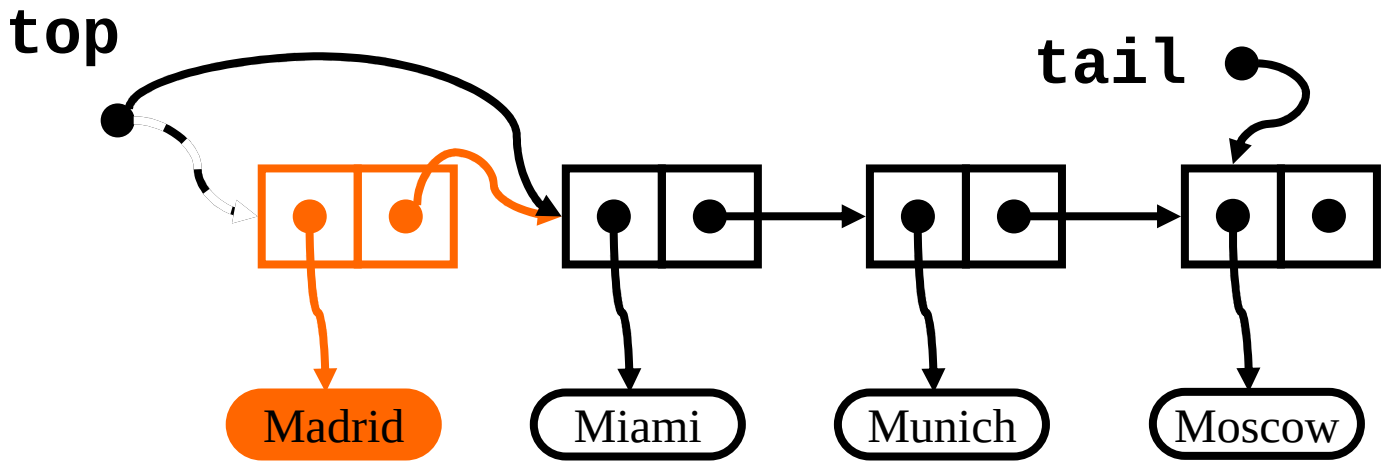
Implementation based on linked lists



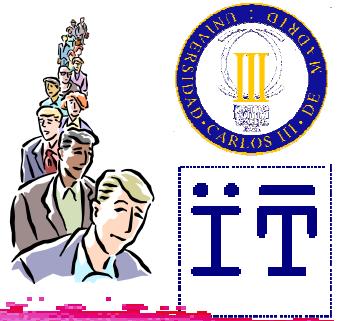
```
public void enqueue(Object info) {
  Node n = new Node(info, null);
  if (top == null)
    top = n;
  else
    tail.setNext(n);
  tail = n;
  size++;
}
```

Extraction (dequeue)





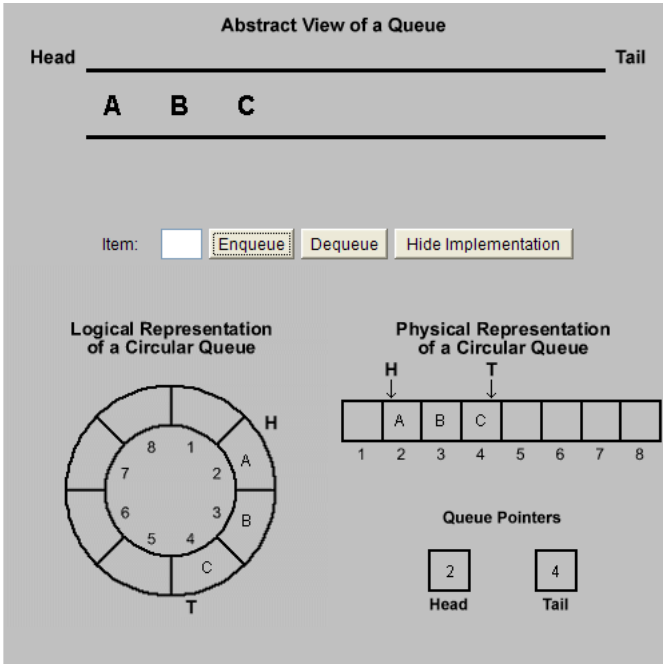
Implementation based on linked lists



```
public Object dequeue()
  throws EmptyQueueException {
  Object info;
  if (top == null)
    throw new EmptyQueueException();
  info = top.getInfo();
  top = top.getNext();
  if (top == null)
    tail = null;
  size--;
  return info;
}
```

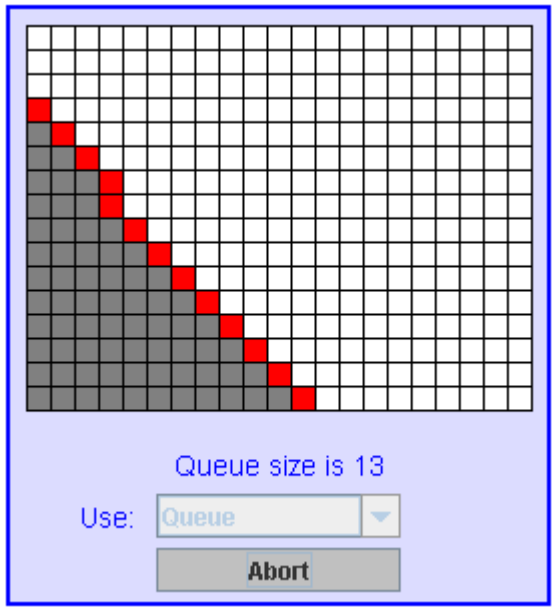
Activity

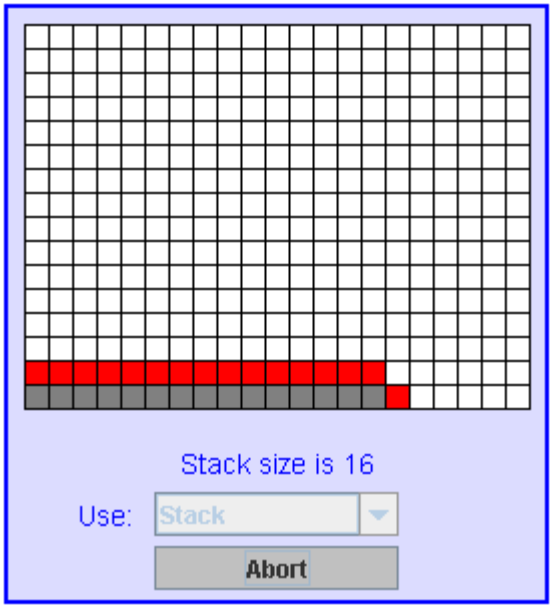
- View queue animations:
- **http://courses.cs.vt.edu/csonline/DataStructures/Lessons/QueuesImplementationView/applet.html**



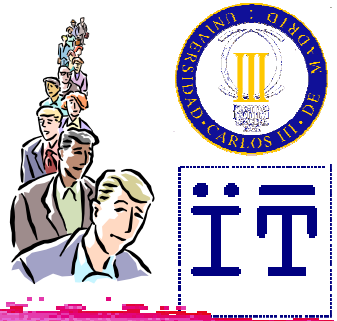
Activity

- Try the applet **`DepthBreadth.java`** that can be found here:
  `http://www.faqs.org/docs/javap/c11/s3.html`



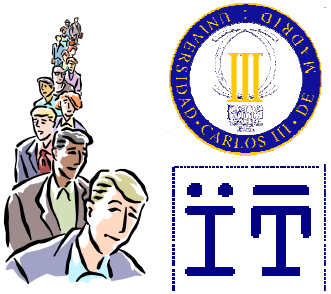


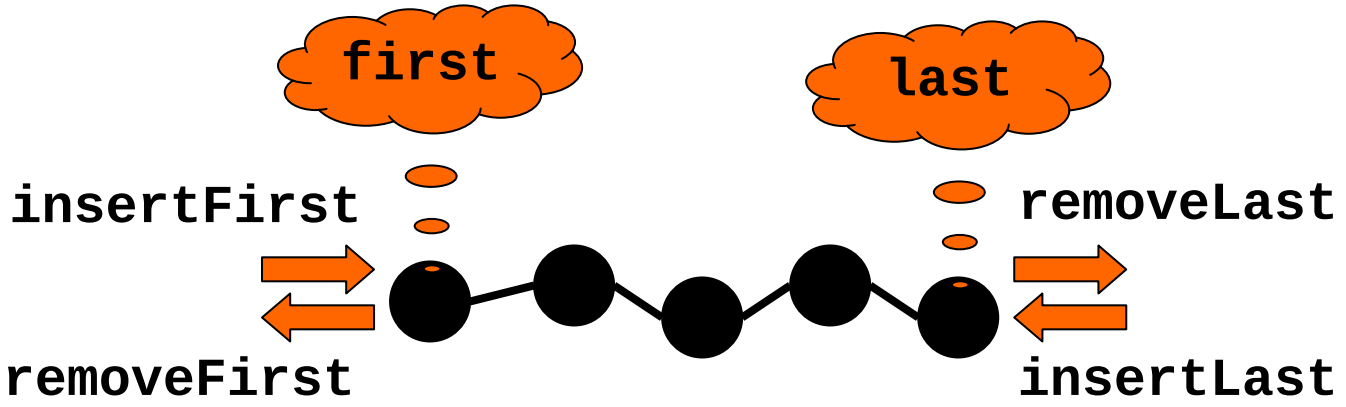
Other kinds of queues (not queues any more)



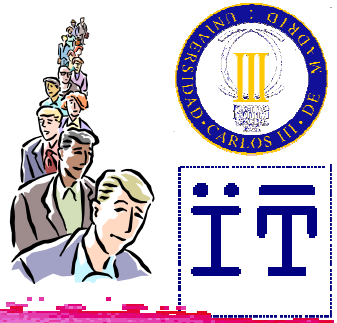
- Double-ended queues
- Priority queues

Deques
(Double-ended queues)





Interface for deques



```
public interface Deque {

   public int size();
   public boolean isEmpty();

   public void insertFirst(Object info);
   public void insertLast(Object info);
```

Interface for deques

```
public Object removeFirst()
  throws EmptyDequeException;
public Object removeLast()
  throws EmptyDequeException;

public Object first()
  throws EmptyDequeException;
public Object last()
  throws EmptyDequeException;
```

Stacks and queues as deques

Stack	Deque
`size()`	`size()`
`isEmpty()`	`isEmpty()`
`top()`	`last()`
`push(o)`	`insertLast(o)`
`pop()`	`removeLast()`

Queue	Deque
`size()`	`size()`
`isEmpty()`	`isEmpty()`
`front()`	`first()`
`enqueue(o)`	`insertLast(o)`
`dequeue()`	`removeFirst()`

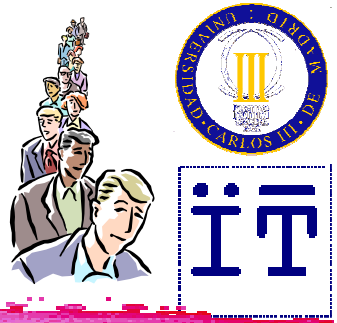
Definition of stacks from deques

```
public class DequeStack implements Stack {
  private Deque deque;
  public DequeStack() {
    deque = new Deque();
  }
  public int size() {
    return deque.size();
  }
  public boolean isEmpty() {
    return deque.isEmpty();
  }
```

Definition of stacks from deques

```
public void push(Object info) {
  deque.insertLast(info);
}
public Object pop()
  throws EmptyStackException {
  try {
    return deque.removeLast();
  } catch (EmptyDequeException ede) {
    throw new EmptyStackException();
  }
}
```

Definition of stacks from deques



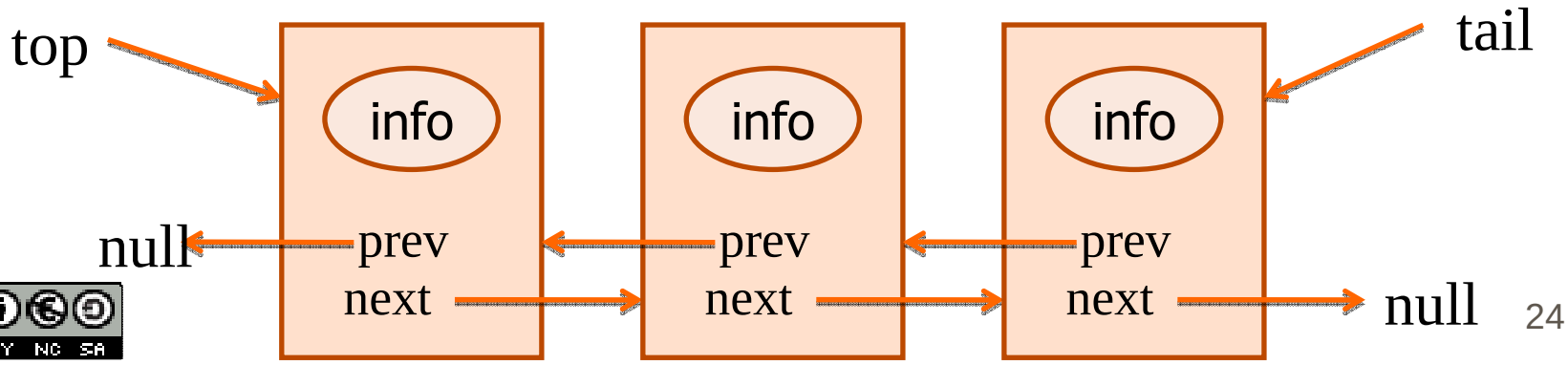
```
public Object top()
   throws EmptyStackException {
   try {
      return deque.last();
   } catch (EmptyDequeException ede) {
      throw new EmptyStackException();
   }
}
```

Implementation of deques based on lists

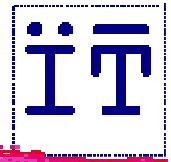
- Singly-linked lists are not appropriate because `removeLast` requires the whole list to be traversed, in order to get the reference of the last-but-one node
- Solution: doubly-linked lists

Doubly Linked Lists

- Linked lists in which each node has an additional reference pointing to the previous node in the list.
  - Can be traversed both from the beginning to the end and vice-versa
  - `RemoveLast` does not need the whole list to be traversed

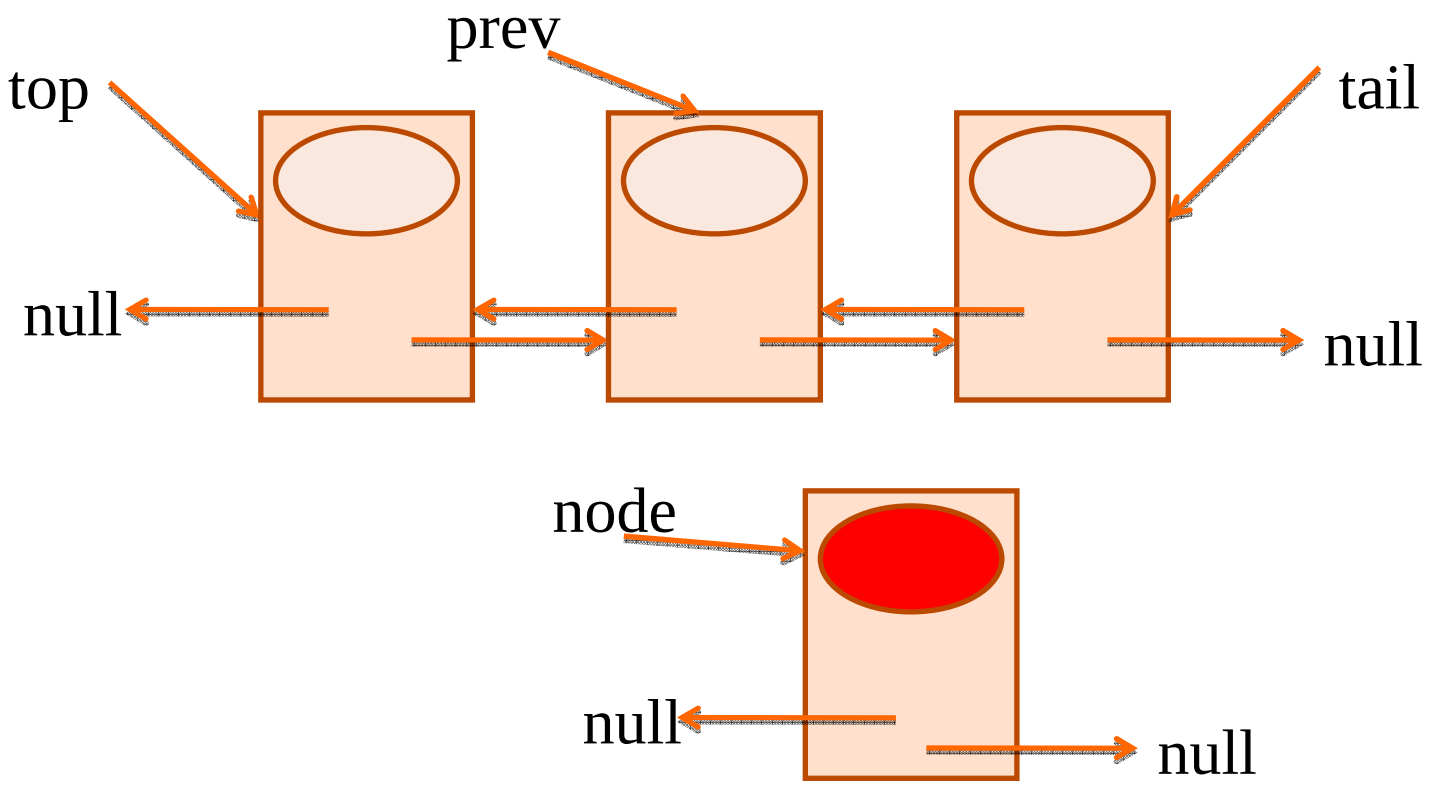


The DLNode class



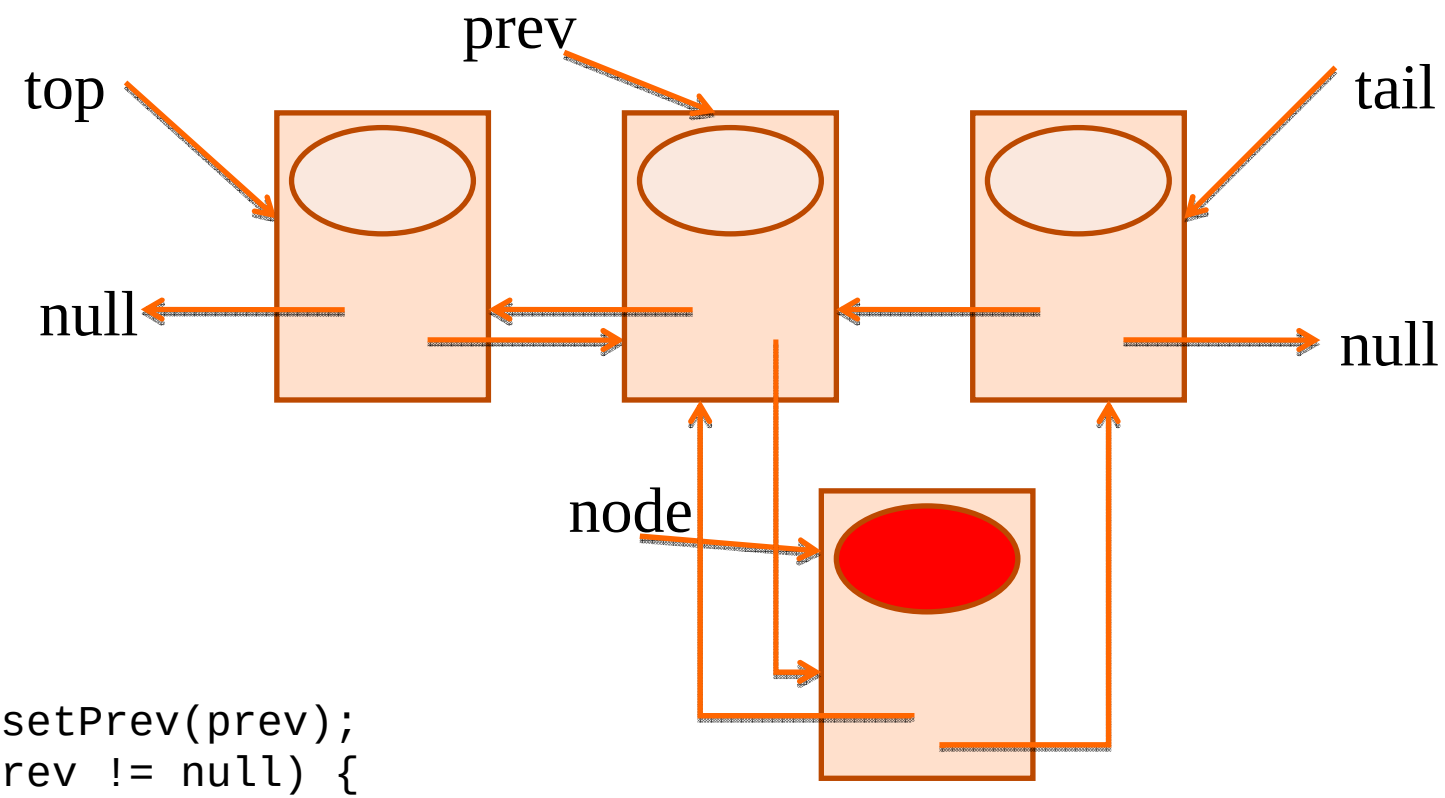
```
Public class DLNode {
    private Object info;
    private DLNode next;
    private DLNode prev;

    public DLNode(Object info) {…}
    public DLNode(Object info, DLNode prev, DLNode next) {…}

    public DLNode getNext() {…}
    public void setNext(DLNode next) {…}
    public DLNode getPrev() {…}
    public void setPrev(DLNode prev) {…}
    public Object getInfo() {…}
    public void setInfo(Object info) {…}
}
```

Inserting a node



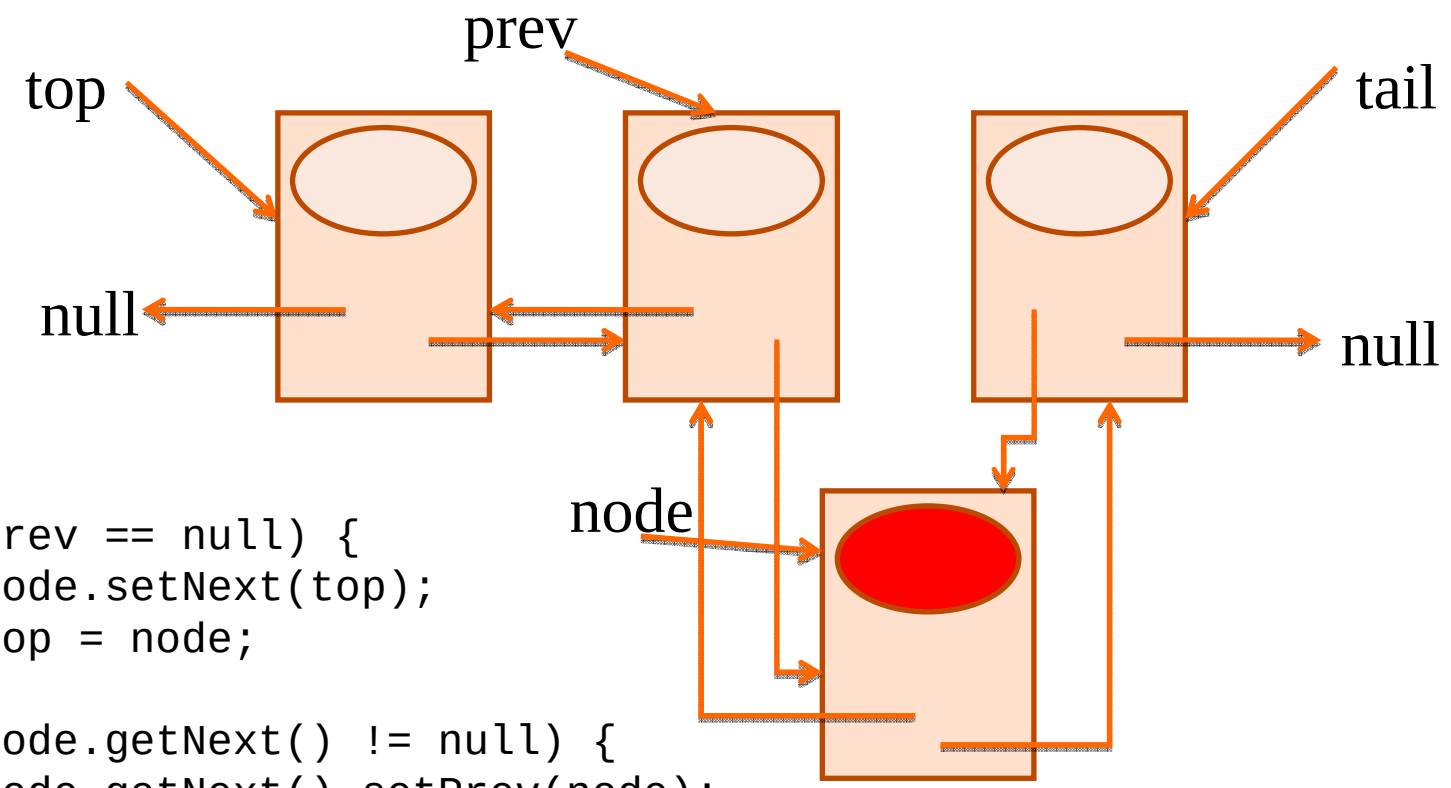

```
DLNode node = new DLNode(data);
```

Inserting a node



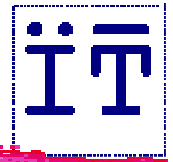

```
node.setPrev(prev);
if (prev != null) {
    node.setNext(prev.getNext());
    prev.setNext(node);
}
```

Inserting a node



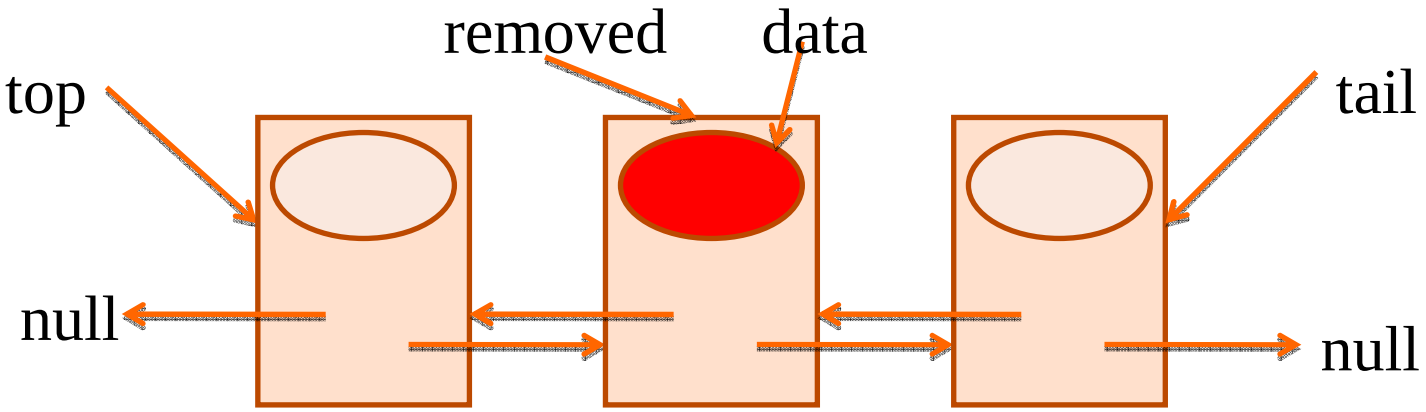

```
if (prev == null) {
    node.setNext(top);
    top = node;
}
if (node.getNext() != null) {
    node.getNext().setPrev(node);
} else {
    tail = node;
}
```

Inserting a node



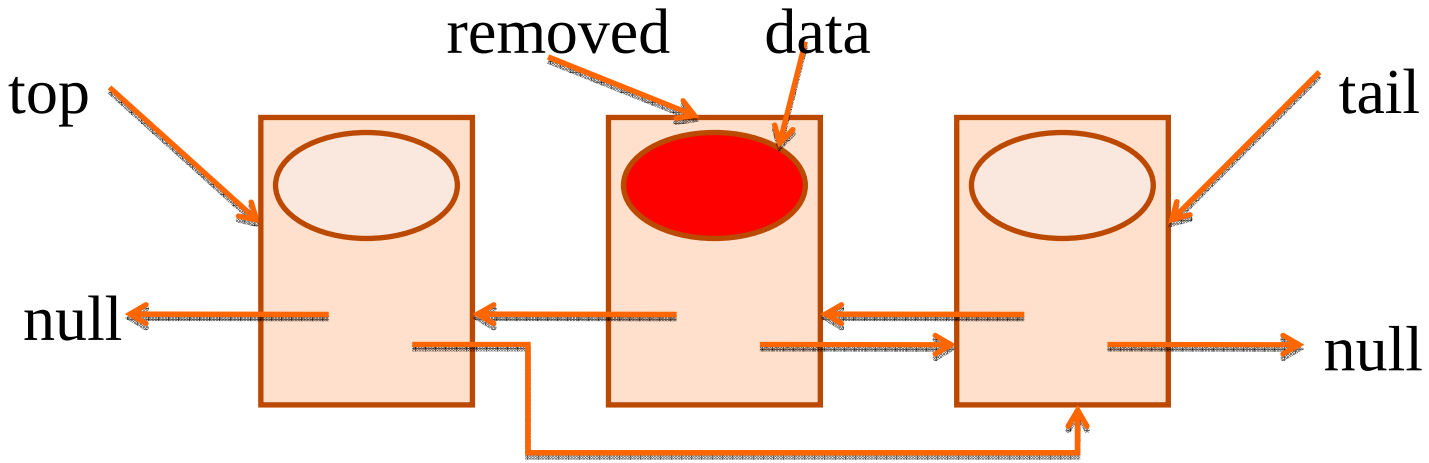
```
/**
  * Inserts 'data' after the 'prev' node. If 'prev'
  * is null, 'data' is inserted at the first position of
  * the list.
  */
public void insert(DLNode prev, Object data) {
    DLNode node = new DLNode(data);
    node.setPrev(prev);
    if (prev != null) {
        node.setNext(prev.getNext());
        prev.setNext(node);
    } else {
        node.setNext(top);
        top = node;
    }
    if (node.getNext() != null) {
        node.getNext().setPrev(node);
    } else {
        tail = node;
    }
}
```

Removing a node



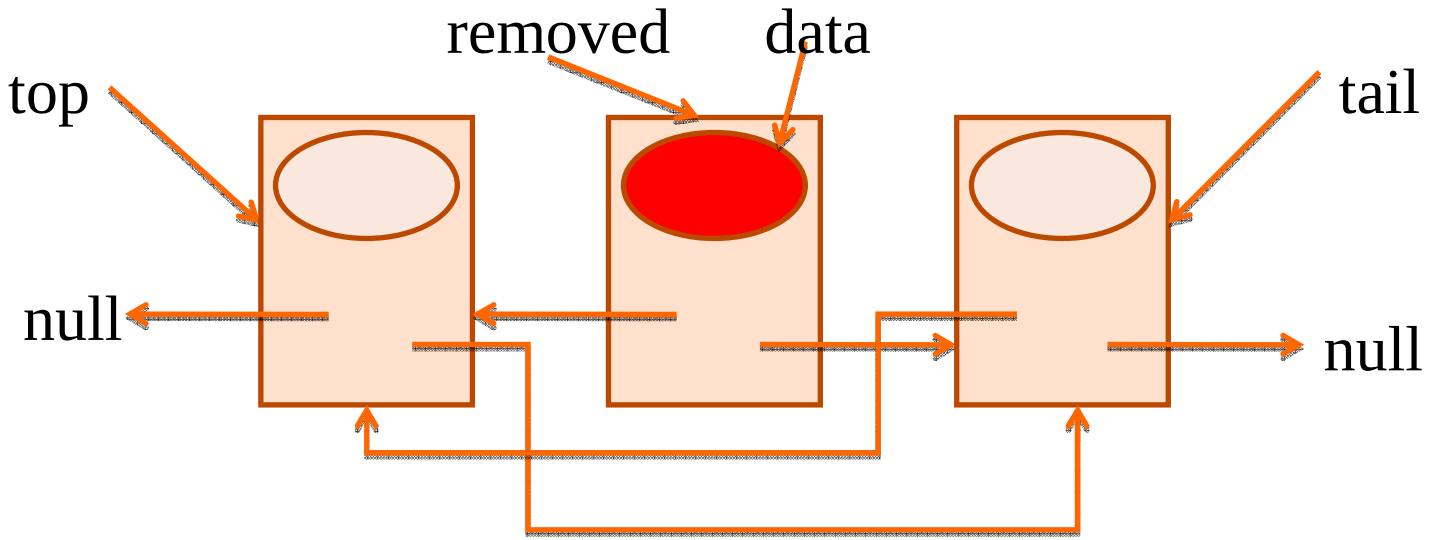

```
Object data = removed.getInfo();
```

Removing a node



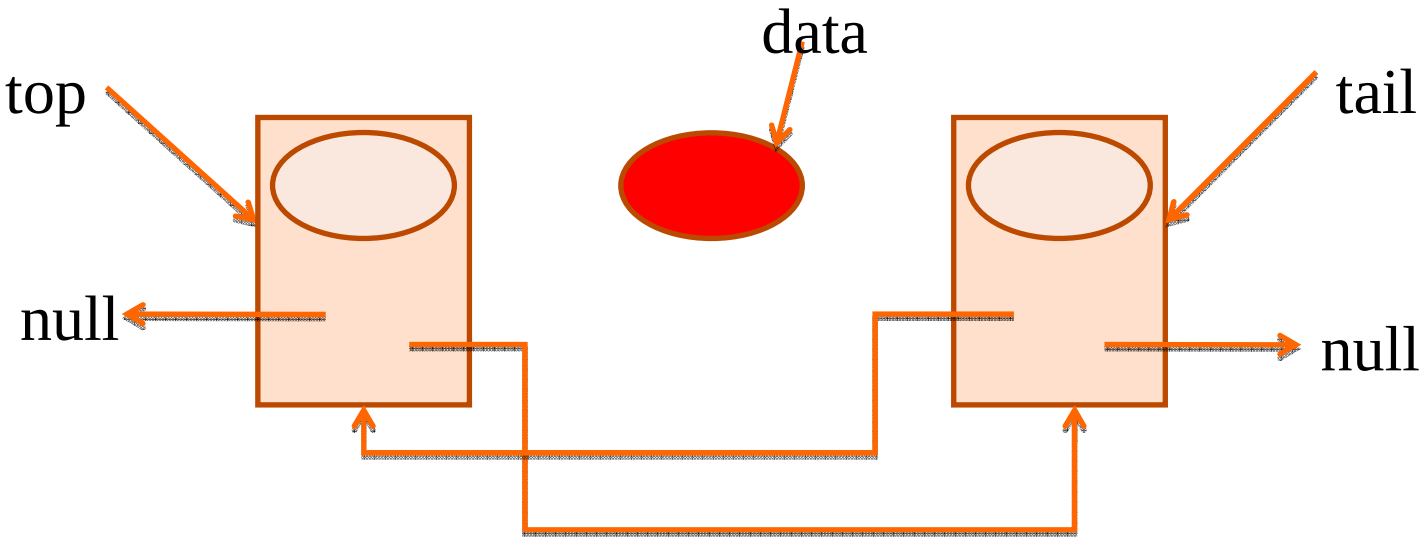

```
if (removed.getPrev() != null) {
    removed.getPrev().setNext(removed.getNext());
} else {
    top = removed.getNext();
}
```

Removing a node




```
if (removed.getNext() != null) {
    removed.getNext().setPrev(removed.getPrev());
} else {
    tail = removed.getPrev();
}
```

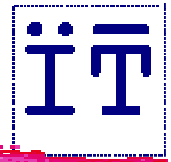
Removing a node



Removing a node

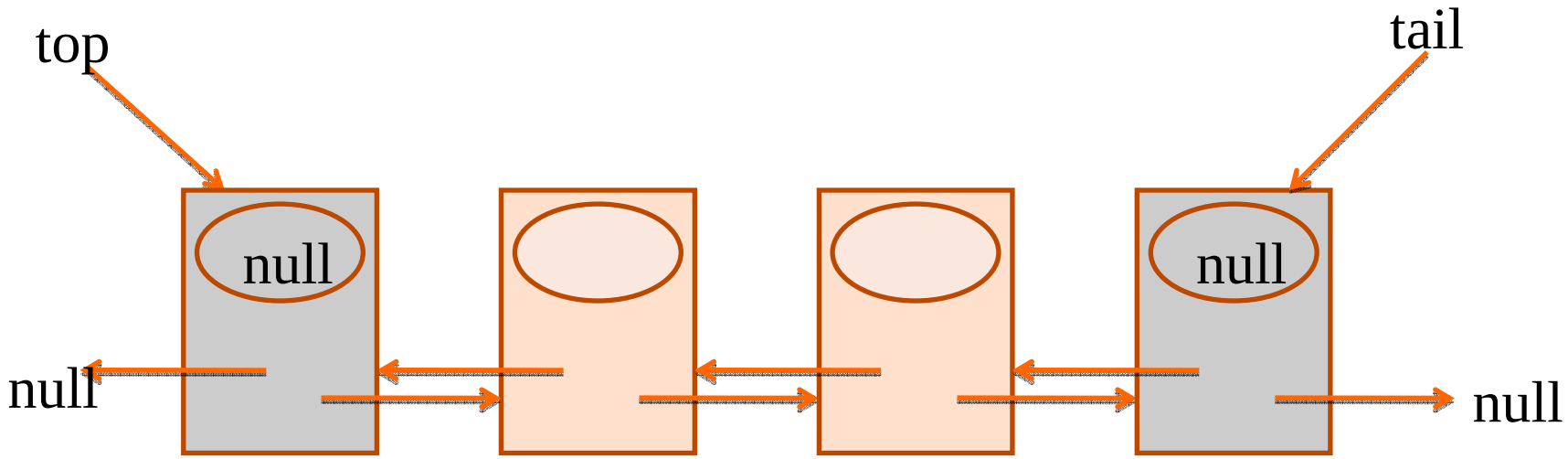
```
/**
  * Removes a node from the list and returns
  * the information it holds.
  */
public Object remove(DLNode removed) {
    Object data = removed.getInfo();
    if (removed.getPrev() != null) {
        removed.getPrev().setNext(removed.getNext());
    } else {
        top = removed.getNext();
    }
    if (removed.getNext() != null) {
        removed.getNext().setPrev(removed.getPrev());
    } else {
        tail = removed.getPrev();
    }
    return data;
}
```

Alternate implementation

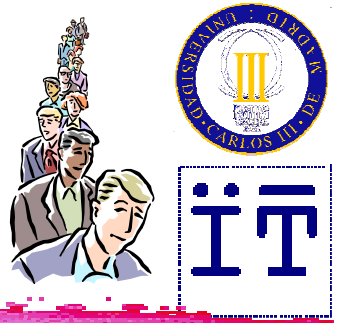


- Checking that the previous and next nodes are not null makes the previous implementation complex and error-prone
- Possible simplification:
  - Create two special nodes, without associated info, so that one is always at the beginning of the list and the other one is always at the end:
    - An empty list contains only those two empty nodes.
    - For insertions and removals, it is guaranteed that the previous and next nodes exists, so there is no need to check them.
    - References `top` and `tail` do not need to be updated

Alternate implementation

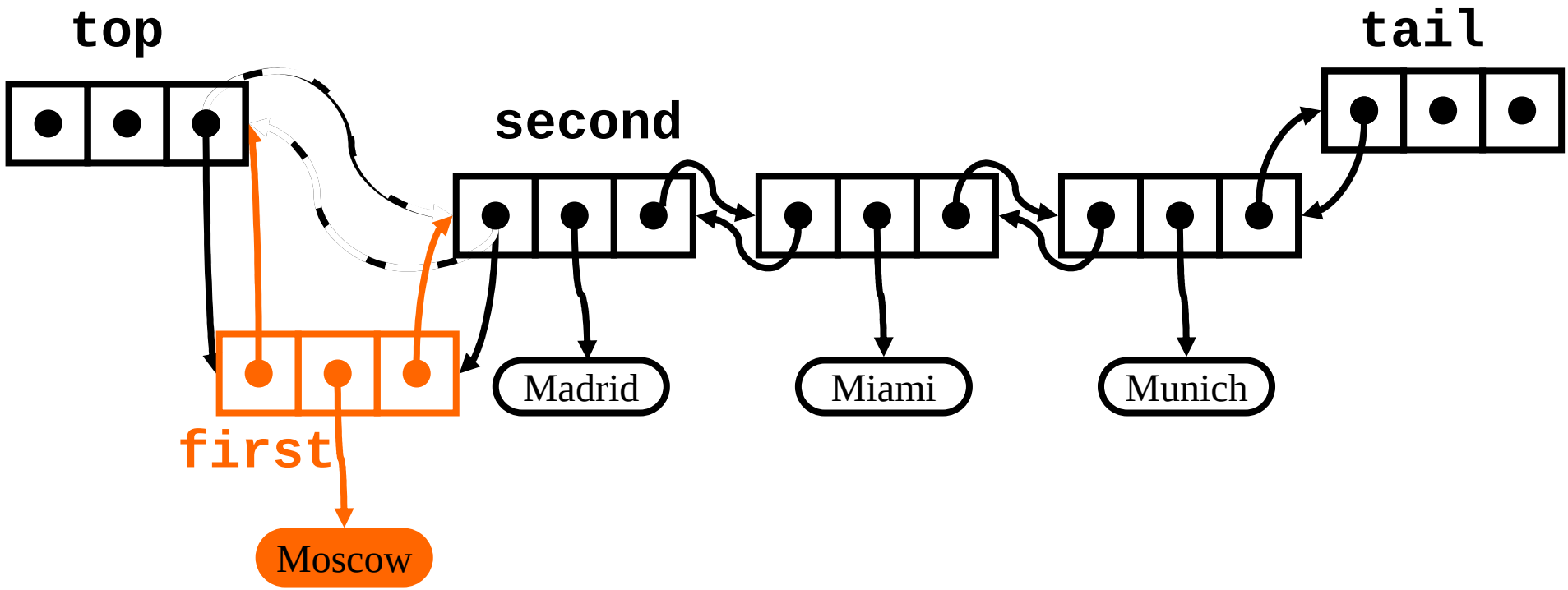


Implementation based on lists

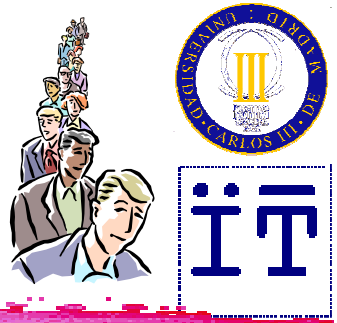


```
public class DLDeque implements Deque {
  private DLNode top, tail;
  private int size;
  public DLDeque() {
    top = new DLNode();
    tail = new DLNode();
    tail.setPrev(top);
    top.setNext(tail);
    size = 0;
  }
```

Insertion

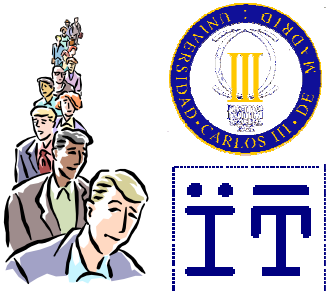


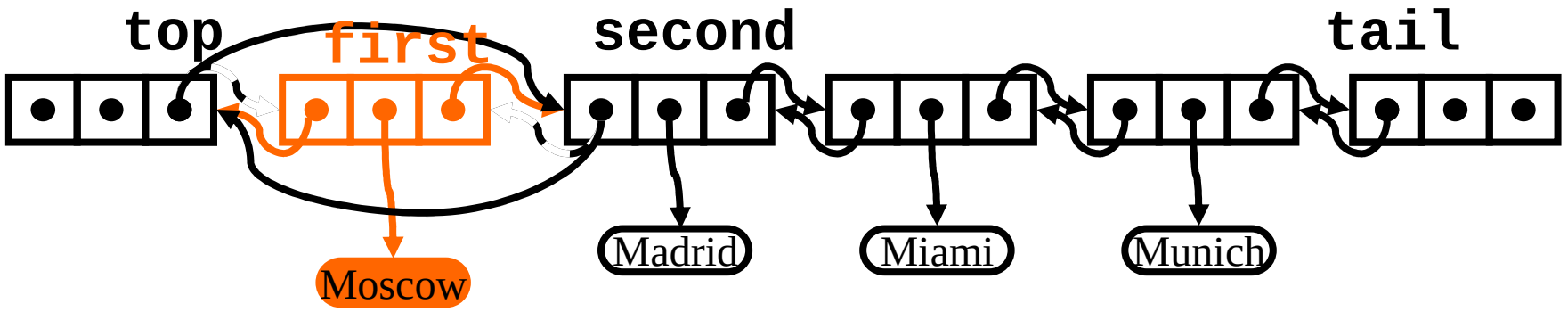
Implementation based on lists



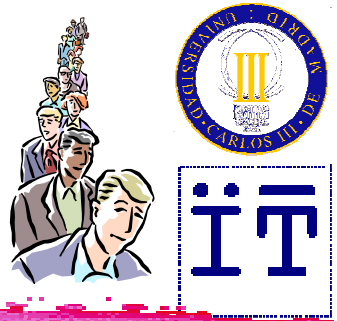
```
public void insertFirst(Object info) {
  DLNode second = top.getNext();
  DLNode first = new DLNode(info, top, second);
  second.setPrev(first);
  top.setNext(first);
  size++;
}
```

Extraction





Implementation based on lists



```
public Object removeFirst()
  throws EmptyDequeException {
  if (top.getNext() == tail)
    throw new EmptyDequeException();
  DLNode first = top.getNext();
  Object info = first.getInfo();
  DLNode second = first.getNext();
  top.setNext(second);
  second.setPrev(top);
  size--;
  return info;
```

Activity

- Review how "queues" are implemented in
  - **http://java.sun.com/docs/books/tutorial/collections/interfaces/queue.html**
  - **http://java.sun.com/javase/6/docs/api/java/util/Queue.html**

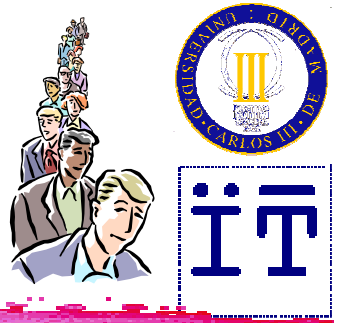
Method Summary	
boolean	add(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
E	element() Retrieves, but does not remove, the head of this queue.
boolean	offer(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll() Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	remove() Retrieves and removes the head of this queue.

Methods inherited from interface java.util.Collection
addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Priority queue

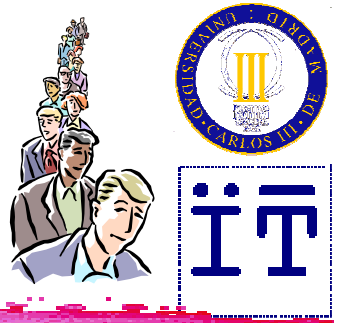
- A priority queue is a linear data structure where elements are retuned according to a value associated to them (priority) (and not necessarily to the order of insertion).
- The priority might be the value of the element itself, but it might also differ from it.

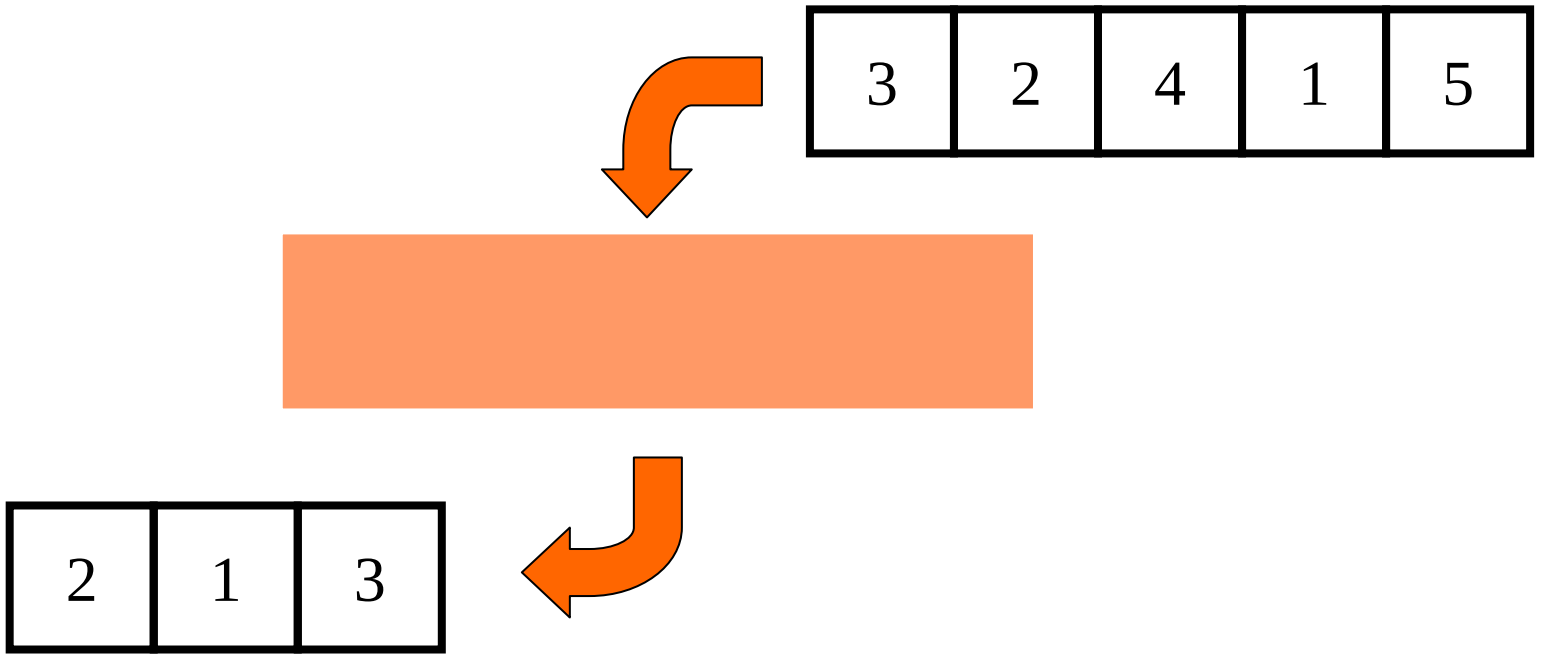
Interface for priority queues



```
public interface PriorityQueue {
  public int size();
  public boolean isEmpty();
  public void insertItem(Comparable priority,
                         Object info);
  public Object minElem()
    throws EmptyPriorityQueueException;
  public Object removeMinElem()
    throws EmptyPriorityQueueException;
  public Object minKey()
    throws EmptyPriorityQueueException;
}
```

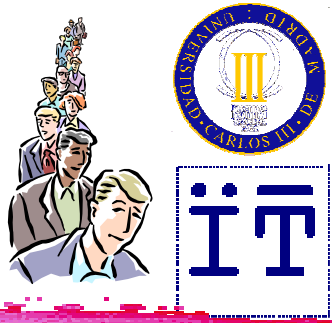
Example

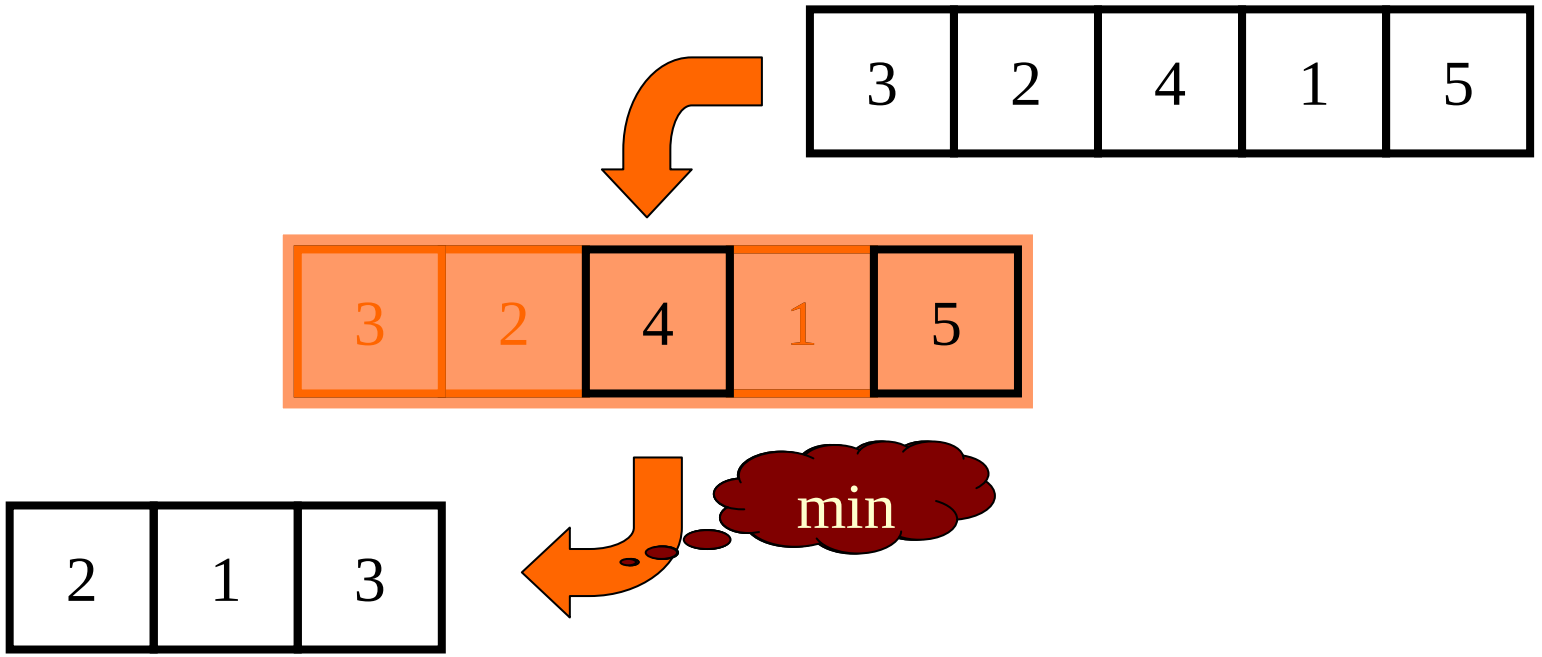




+ + + – + – + –

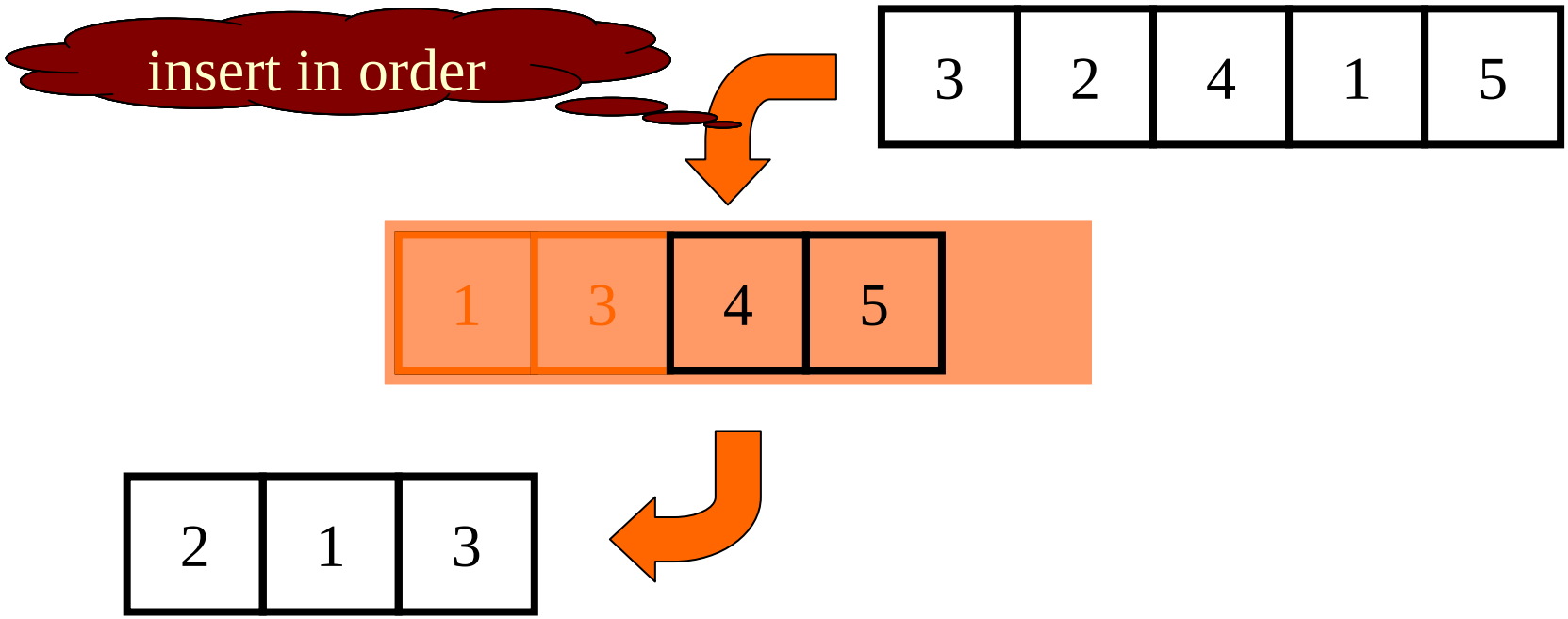
Example





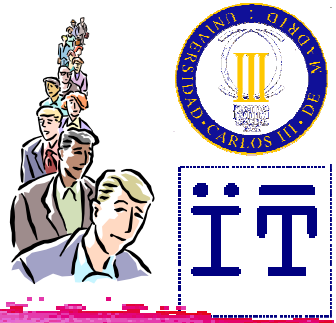
+ + + – + – + –

Example



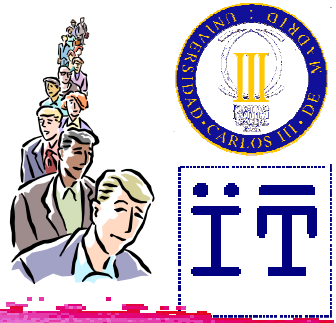
+ + + – + – + –

Implementations



- With an unsorted sequence
  - Easy insertion
  - Comparison needed for extraction
- With a sorted sequence
  - Comparison needed for insertion
  - Easy extraction

Activity



- Try

http://www.akira.ruc.dk/~keld/algoritmik_e99/Applets/Chap11/PriorityQ/PriorityQ.html