ITI 1121. Introduction to Computing II

Queue: concept

by

Marcel Turcotte

Version March 2, 2020

Preamble

Preamble

Overview

Overview

Queue: concept

We are interested in all aspects of queues in programming. We examine several examples of their use, including resource sharing, simulation algorithms, and the breadth-first search algorithm. We will see two implementations of queues: either using circular arrays or using chained elements.

General objective :

- This week, you will be able to describe, apply, and implement a queue.

Preamble

Learning objectives

Learning objectives

- **Describe** the concept of a queue in computer science.
- **Implement** a queue using linked elements.

Lectures:

- Pages 177–189 of E. Koffman and P. Wolfgang.

Preamble

Plan

Plan

Definitions

Definitions

A **queue** is a linear **abstract data type** such that adding data is done at one end, the **rear** of the queue, and removing at the other, the **front**.

These data structures are called **FIFO**: *first-in first-out*.

$$\texttt{enqueue()} \Rightarrow \text{Queue} \Rightarrow \texttt{dequeue()}$$

The two **basic operations** are:

- **enqueue:** **adding** an item to the **rear** of the queue...,
- **dequeue:** the **removal** of the **front** element to the queue.

$\Rightarrow$ The queues are therefore data structures similar to the queues at the supermarket, bank, cinema, etc.

Abstract Data Type (ADT): Queue

```java
public interface Queue<E> {
  void enqueue(E element);
  E dequeue();
  boolean isEmpty();
}
```

Applications of queues

- Managing **shared resources**:
  - Accessing the CPU;
  - Access to a disk or other peripherals, e.g. printer;
- **Algorithms** based on queues:
  - Simulations;
  - Breadth-first-search.

Example

```
public class Test {
    public static void main(String[] args) {

        Queue<Integer> q;
        q = new LinkedQueue<Integer>();

        for (int i=0; i<10; i++) {
            q.enqueue(Integer.valueOf(i));
        }

        while (! q.isEmpty()) {
            System.out.println(q.dequeue());
        }

    }
}
```

- **Prints?** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

```
q = new LinkedQueue();
q.enqueue(a);
q.enqueue(b);
q.enqueue(c);
q.dequeue();
-> a
q.dequeue();
-> b
q.enqueue(d);
q.dequeue();
-> c
q.dequeue();
-> d
```

- The elements are processed in the same order as they were inserted in the queue, here the element **a** is the first to join the queue and it is also the first to leave the queue (***first-come first-serve***).

Implementation

Implementation

Like stacks, there are two families of implementations:

- **Linked lists**;
- **With the help of an array**.

Implementation using linked elements

```
public class LinkedQueue<E> implements Queue<E> {

    public boolean isEmpty() { ... }
    public void enqueue(E o) { ... }
    public E dequeue() { ... }

}
```

Implementation

Elem

Implementation using linked elements

```
public class LinkedQueue<E> implements Queue<E> {

    private static class Elem<T> {
        private T value;
        private Elem<T> next;
        private Elem(T value, Elem<T> next ) {
            this.value = value;
            this.next = next;
        }
    }

    public boolean isEmpty() { ... }
    public void enqueue(E o ) { ... }
    public E dequeue() { ... }
}
```

Implementation

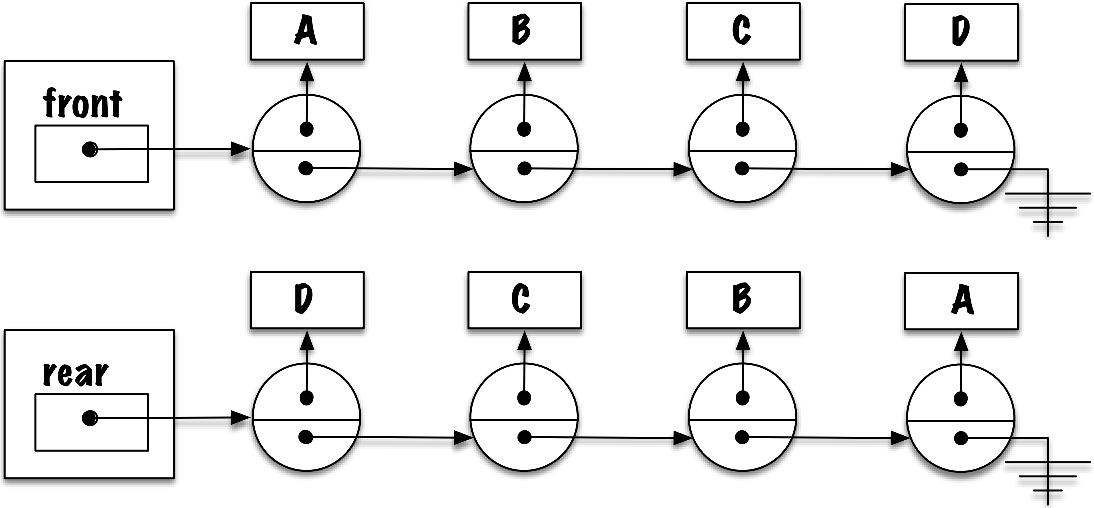
Instance variables

Implementation using linked elements

```
public class LinkedQueue<E> implements Queue<E> {

    private static class Elem<T> {
        private T value;
        private Elem<T> next;
        private Elem(T value, Elem<T> next) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem<E> front; // rear?

    public boolean isEmpty() { ... }
    public void enqueue(E o) { ... }
    public E dequeue() { ... }
}
```

Implementation using linked elements

Which representation do you think is **preferable** and **why**?



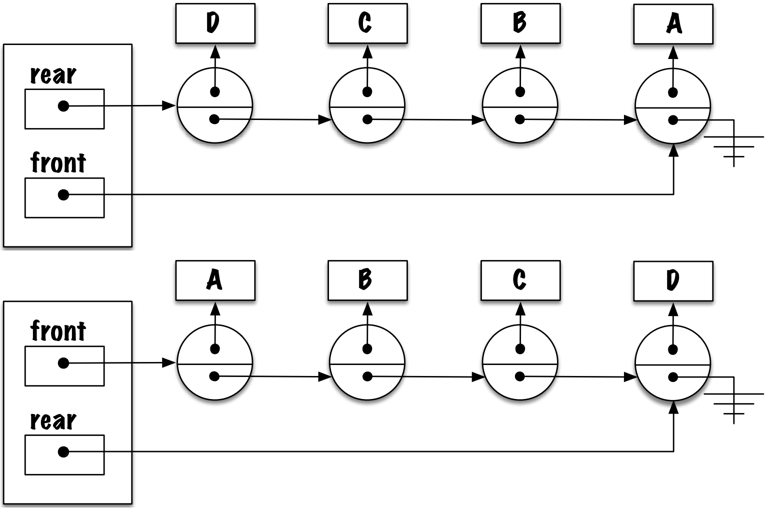
Discussion

- If we choose the first implementation, then the **removal** of an element will be easy (and **fast**) but the **adding** at the rear of the queue will be difficult (and **slow**).
- The other implementation just reverses the situation, the **removal** becomes **costly** while **adding** is **fast**.
- **Is it a dead end?**
- **What is needed to facilitate removal?**
- **What is needed to facilitate adding?**

Implementation using linked elements

Implementation using linked elements

- Will these two implementations be equally **efficient**?



Discussion

- What will be the **impact** of this change?
  - The amount of extra **memory** is **negligible**.
  - The **implementation** of the methods will be **more complex**.

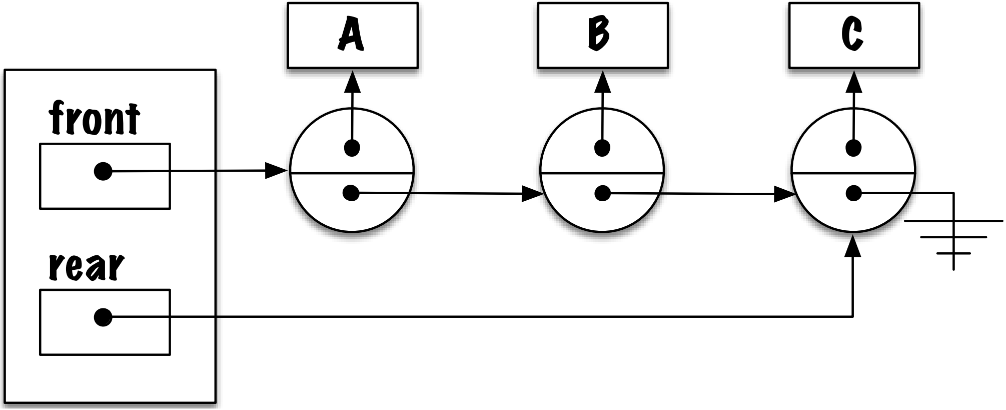
Implementation

Methodology

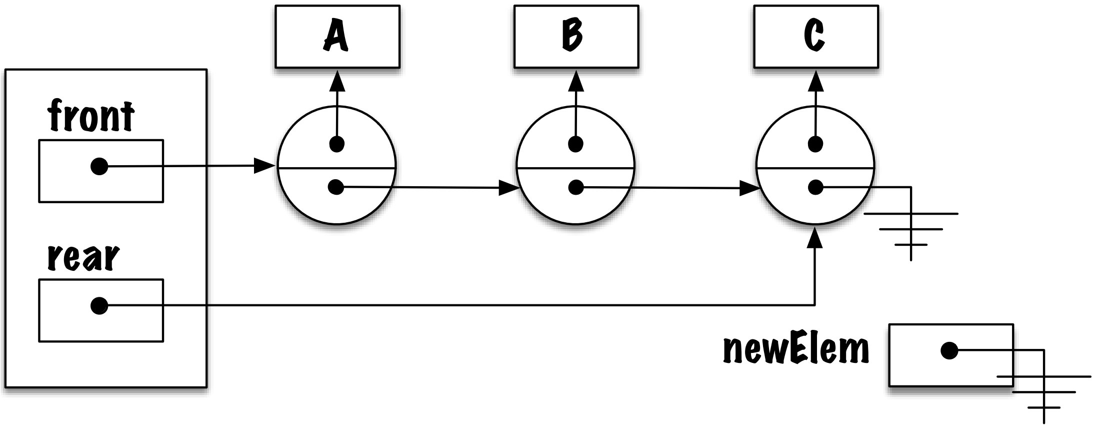
Methodology

- Identify the **general case** as well as the **special case**.
- **General case**, consider a sufficient number of elements so that it represents the majority of the cases.
- **Special cases** are cases where the strategy employed for the general case would not work.
- Queues, stacks, and lists that are **empty** or that have **an element** are often the special cases.

Adding an element (general case)

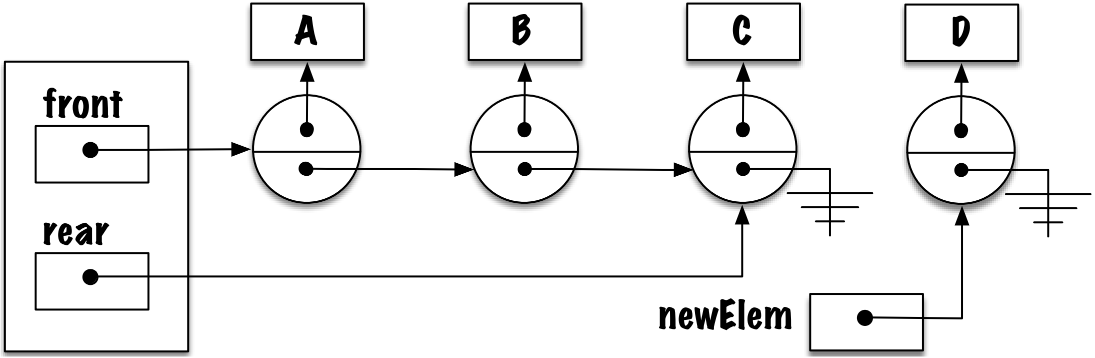


Adding an element (general case)

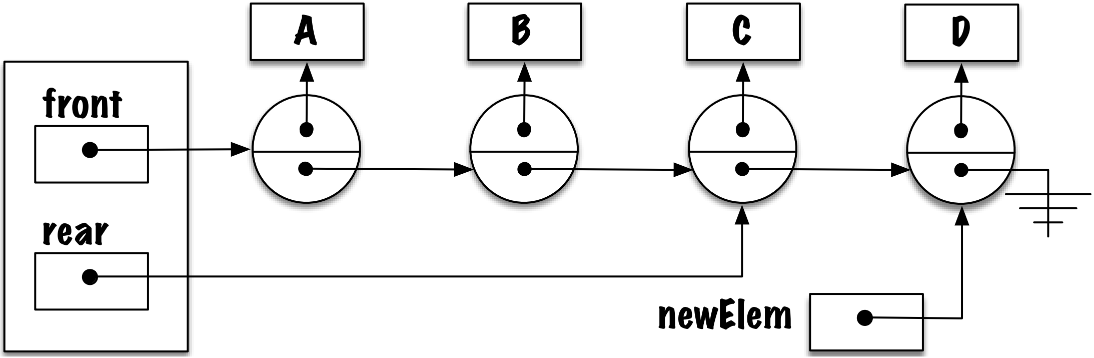


The use of a **local variable** will make the job easier.

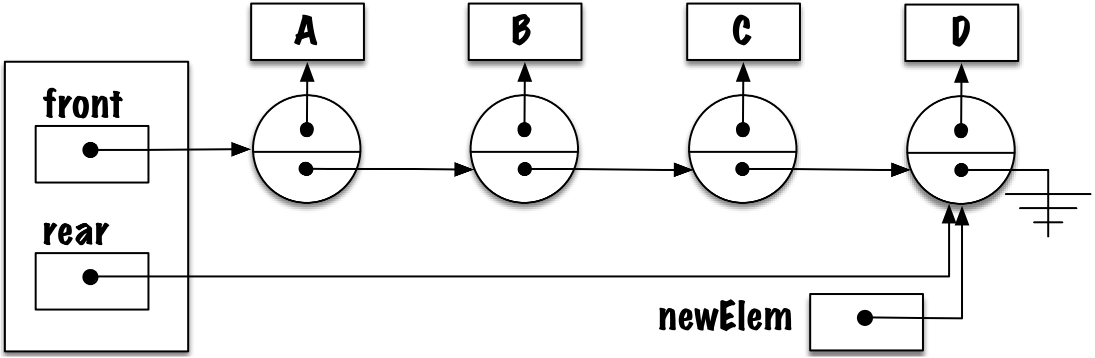
Adding an element (general case)



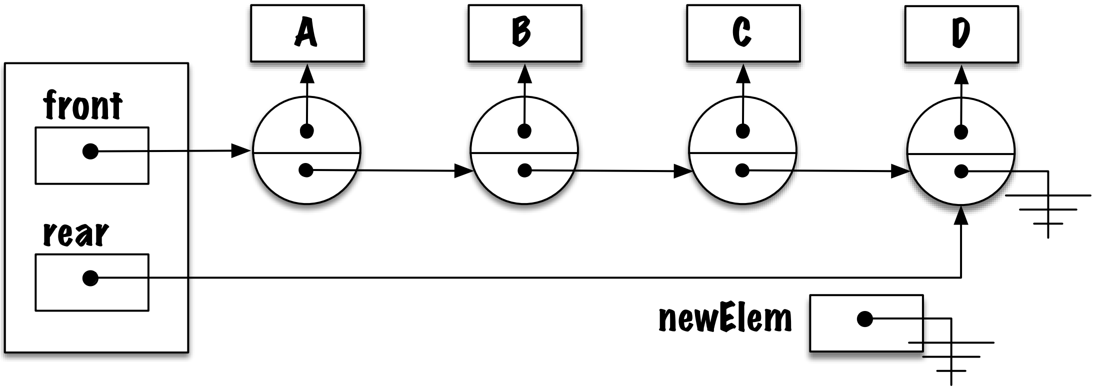
Adding an element (general case)



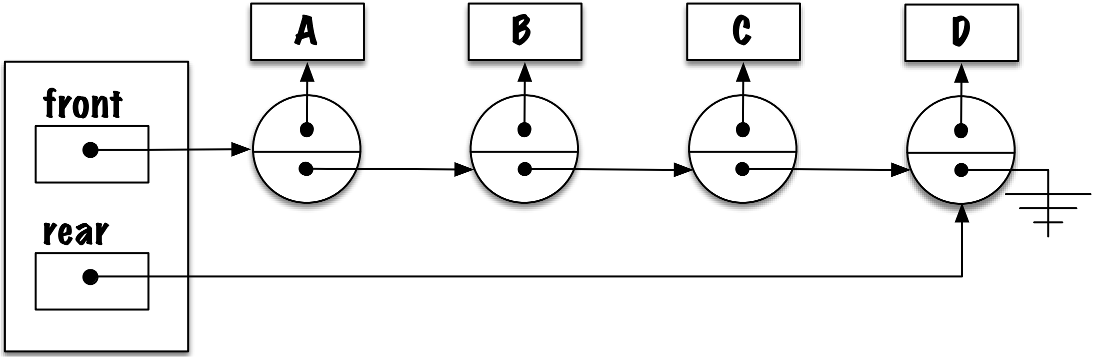
Adding an element (general case)



Adding an element (general case)



Adding an element (general case)

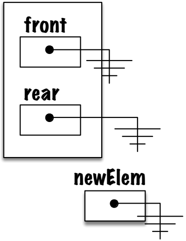


Adding an element (special case)

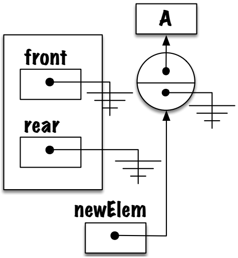
Draw the **memory diagram** representing the **empty** queue.

- What **expression** is used to identify the **empty queue**?

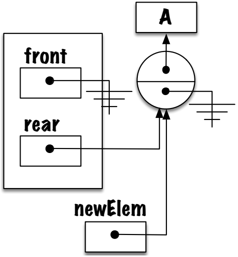
Adding an element (special case)



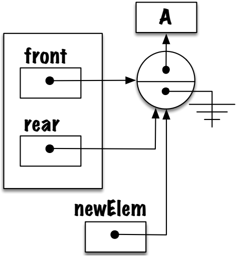
Adding an element (special case)



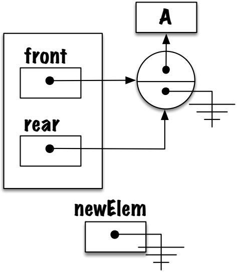
Adding an element (special case)



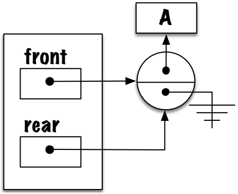
Adding an element (special case)



Adding an element (special case)



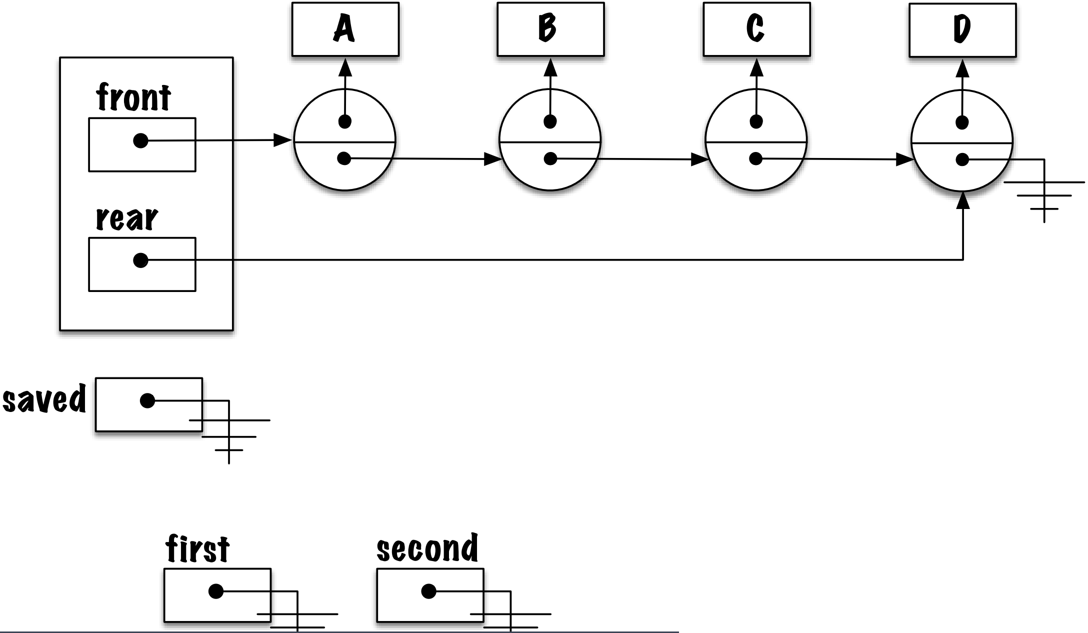
Adding an element (special case)



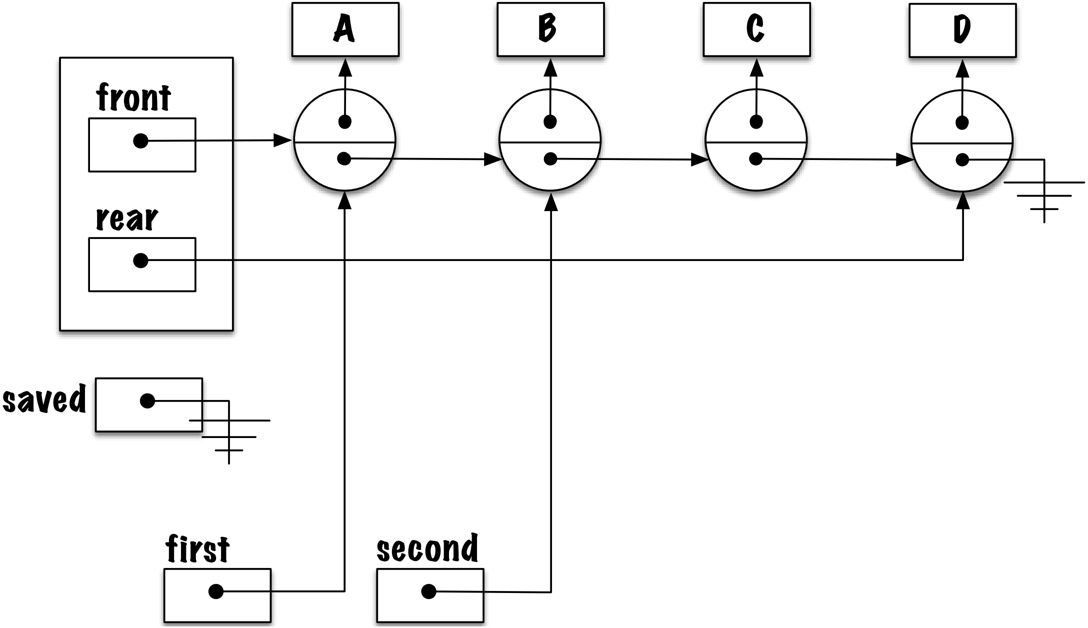
Methodology: removing an element

- Identify the **general case** as well as the **special case(s)**.
- Is the **empty queue** a **special case**?
  - **No**, this is a **illegal** case, for which we'll need to **throw an exception**.
- The queue containing **only one element** is the **special case**.

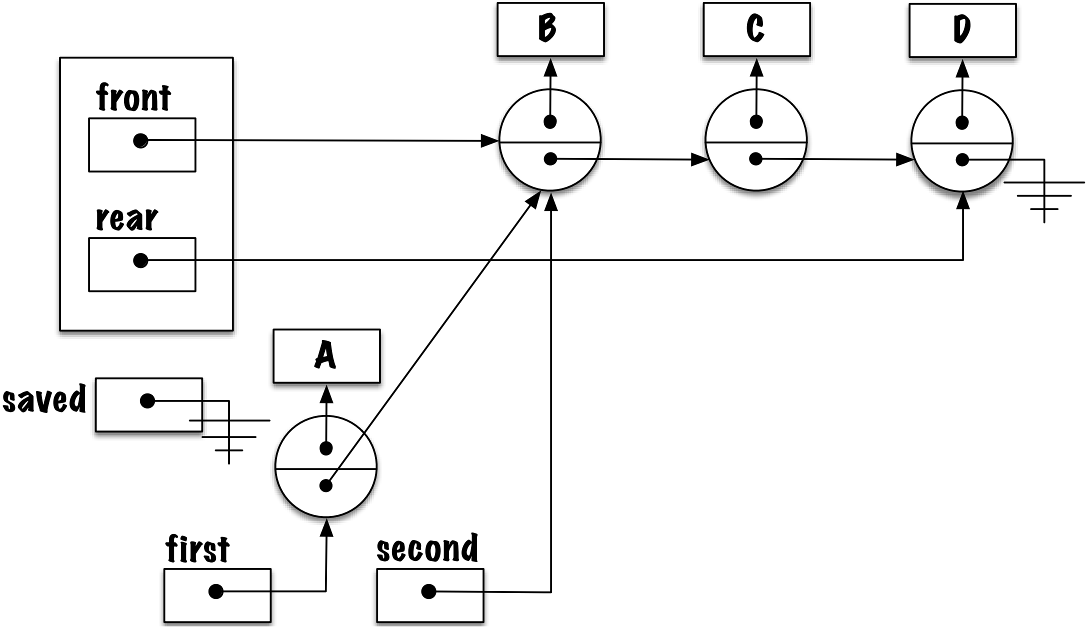
Removal of an element (general case)



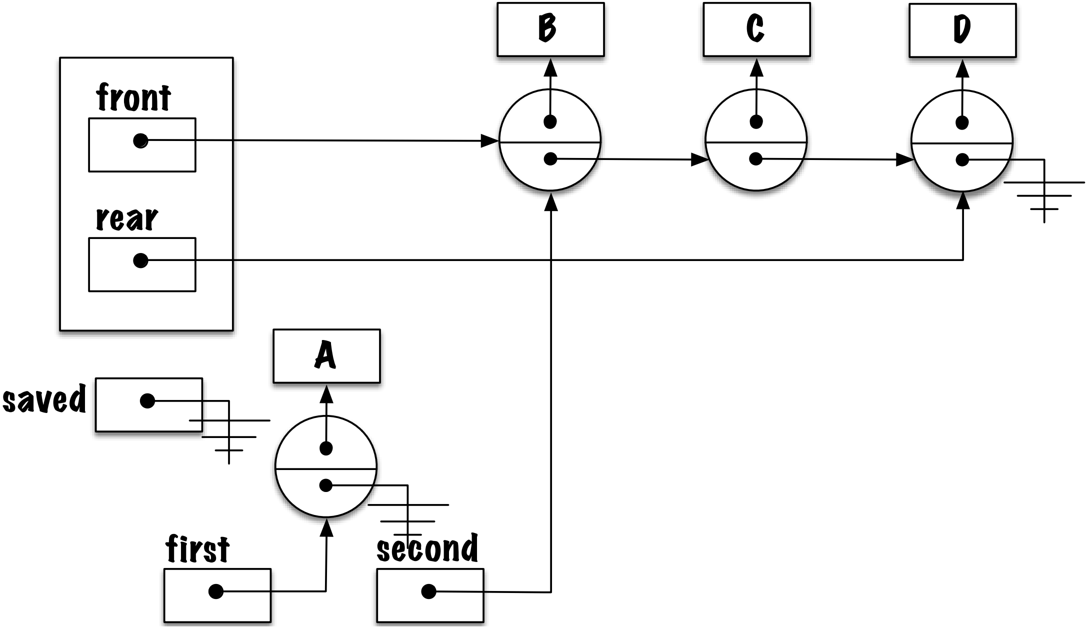
Removal of an element (general case)



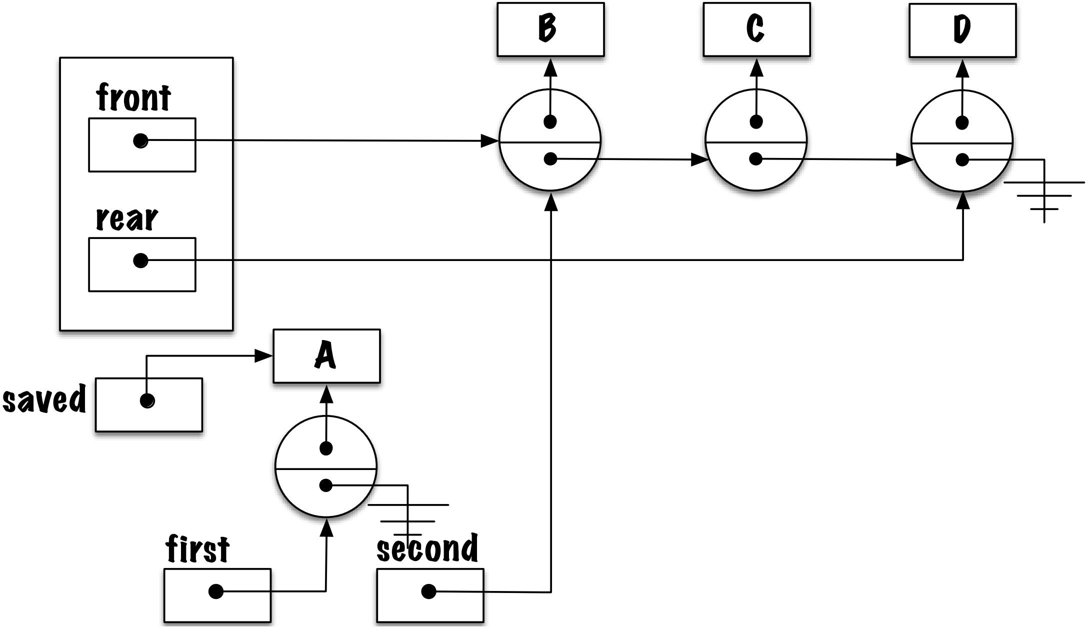
Removal of an element (general case)



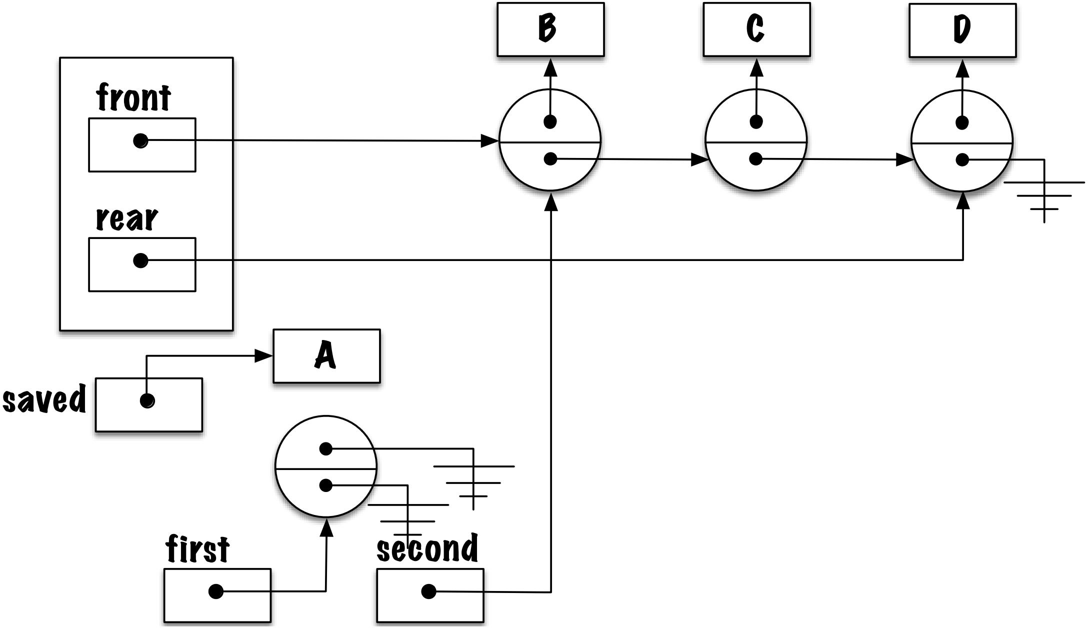
Removal of an element (general case)



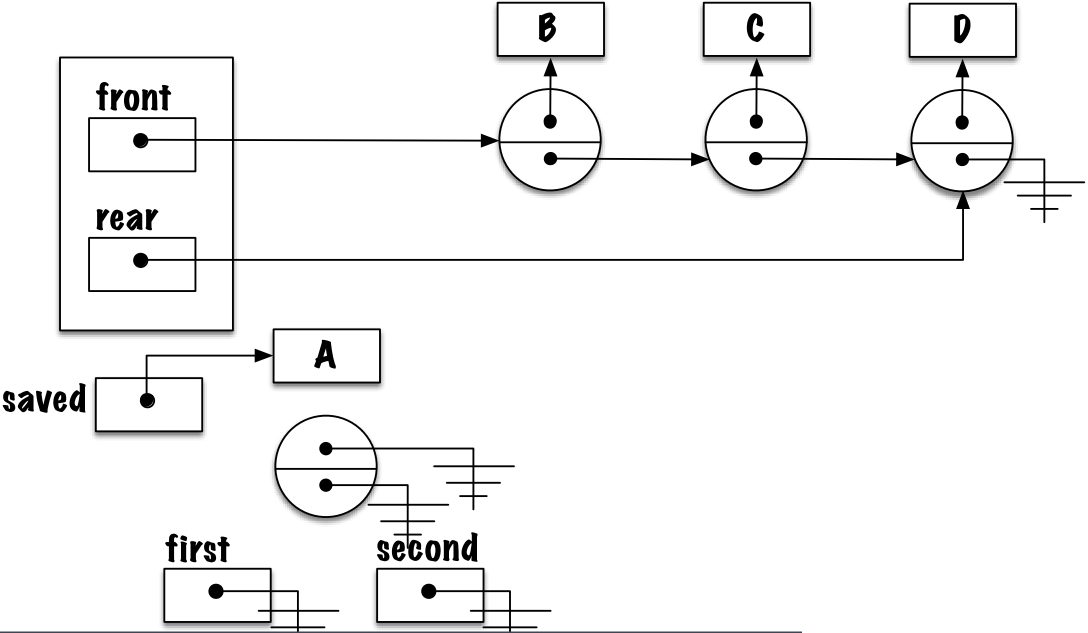
Removal of an element (general case)



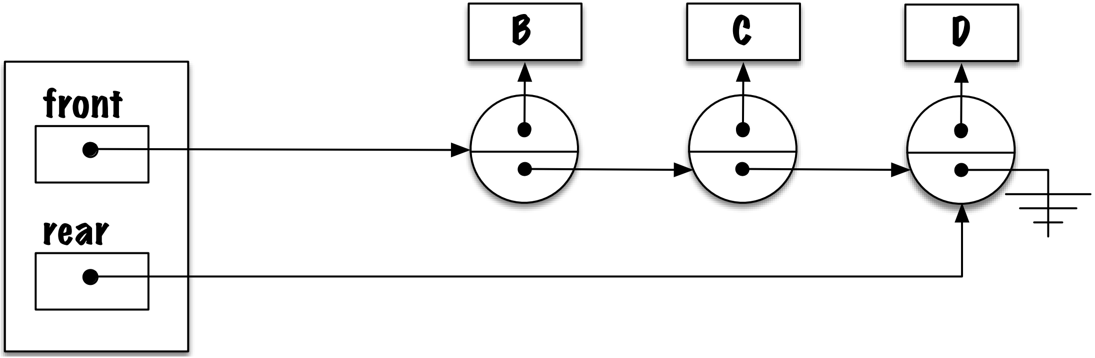
Removal of an element (general case)



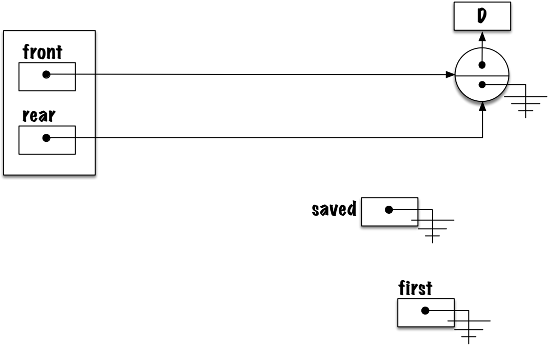
Removal of an element (general case)



Removal of an element (general case)



Removal of an element (special case)

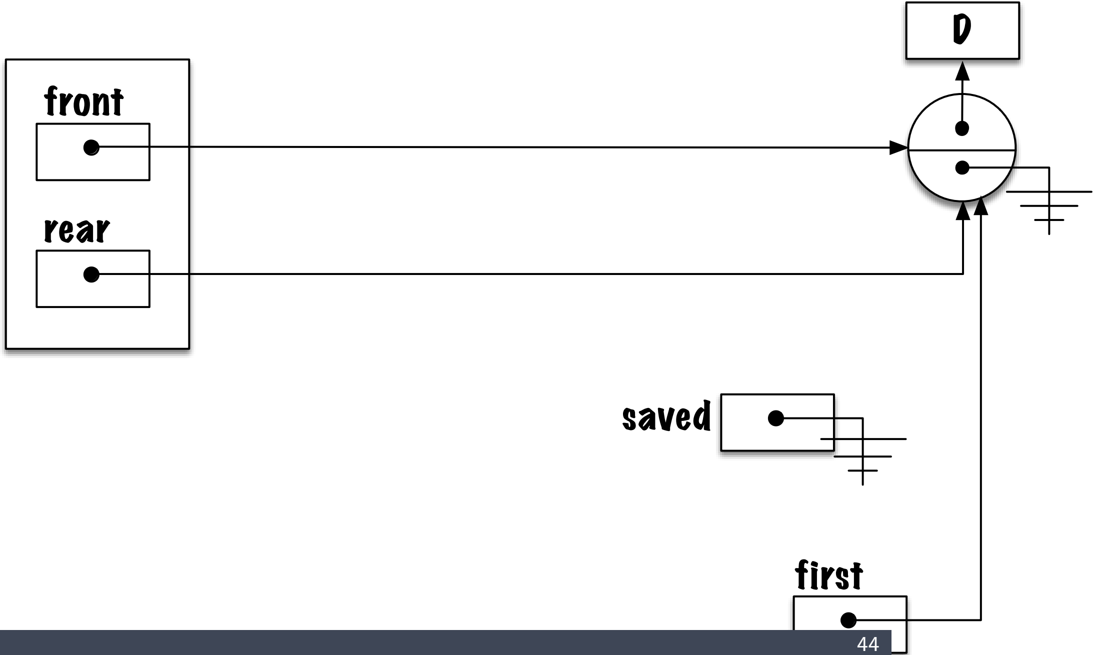


- What **expression** can be used to recognize a queue containing **only one element**?
  - **what do you think** of the following?

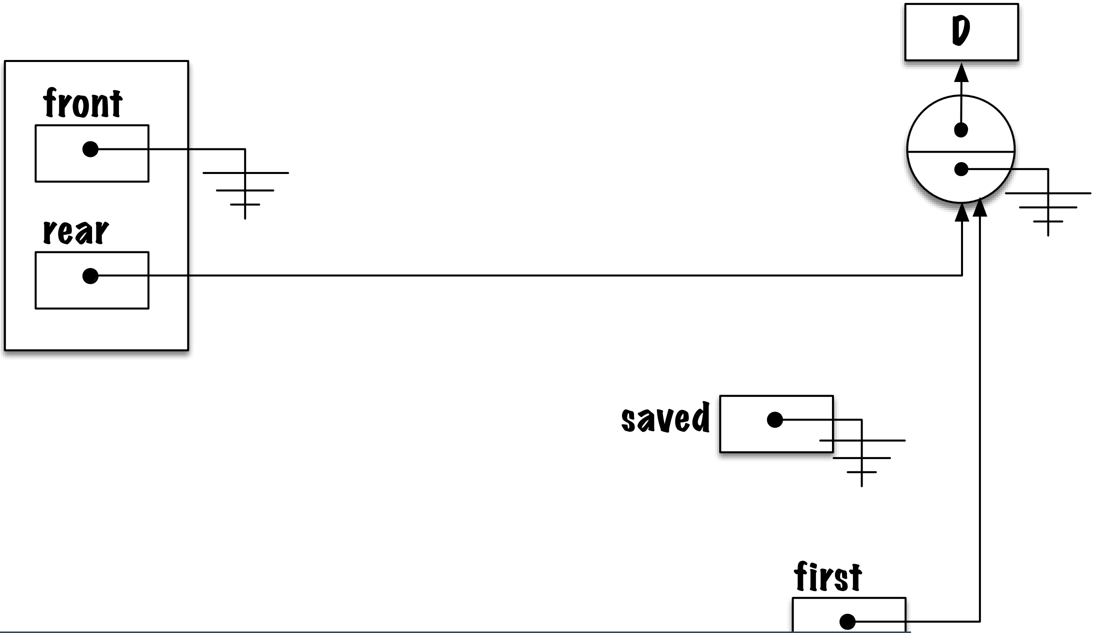
```
front == rear
```

Removal of an element (special case)

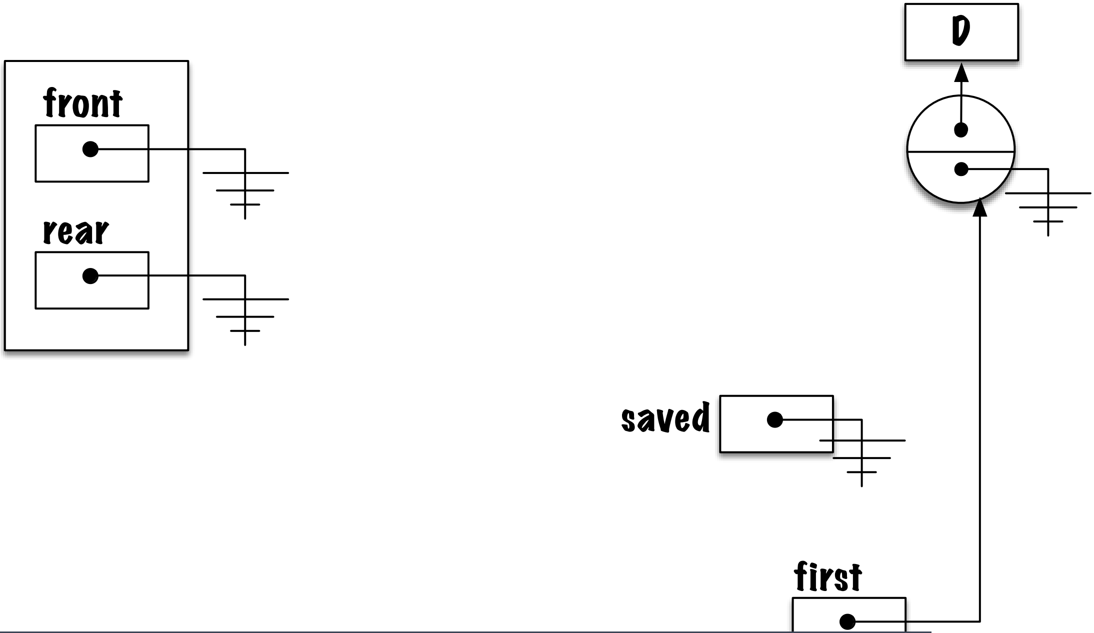
Removal of an element (special case)



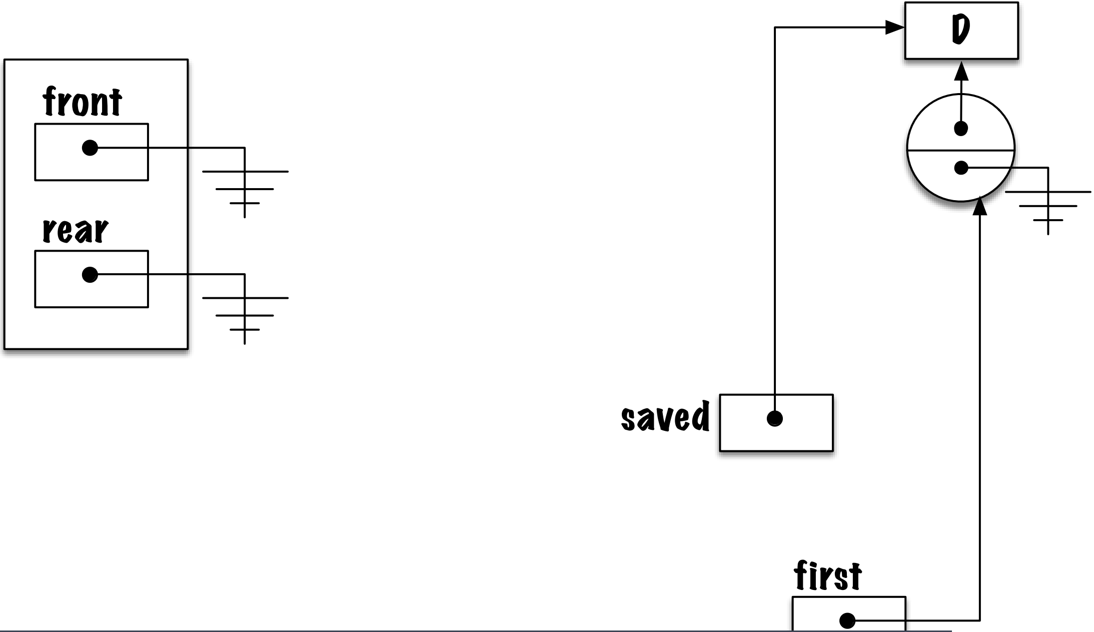
Removal of an element (special case)



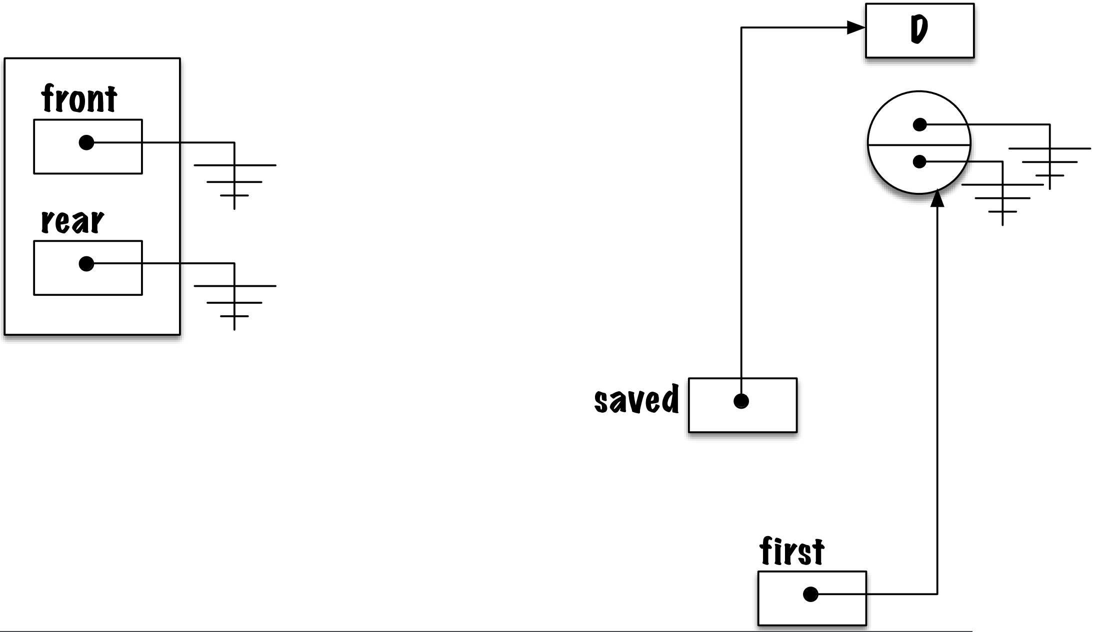
Removal of an element (special case)



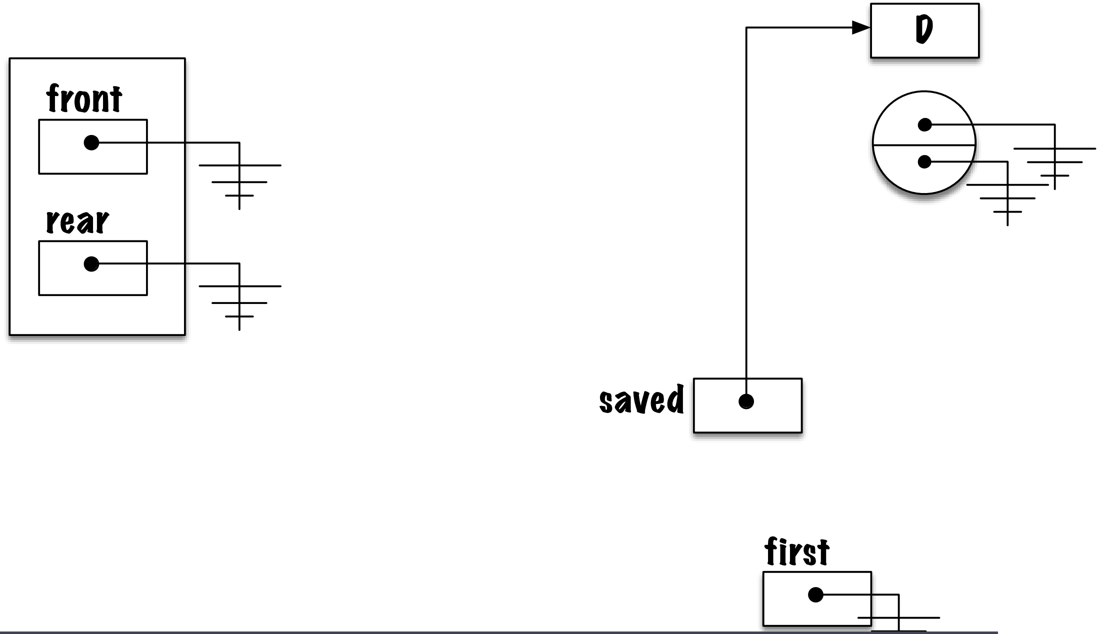
Removal of an element (special case)



Removal of an element (special case)



Removal of an element (special case)



Piège

Pitfall!

- Here's a common problem. The **link to the rear element hasn't been severed**.
- What **kinds of problems** might arise?
- **What will happen** if we use the following expression to detect the empty queue? `front == null && rear == null`.

Prologue

Summary

- A **queue** is a linear **abstract data type** such that adding data is done at one end, the **rear** of the queue, and removing at the other, the **front**.
- The **linked implementation** requires a reference to the **front** element as well as the **rear** element.

Next module

- **Queue** : applications

References I

E. B. Koffman and Wolfgang P. A. T.
***Data Structures: Abstraction and Design Using Java*.**
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**
Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa