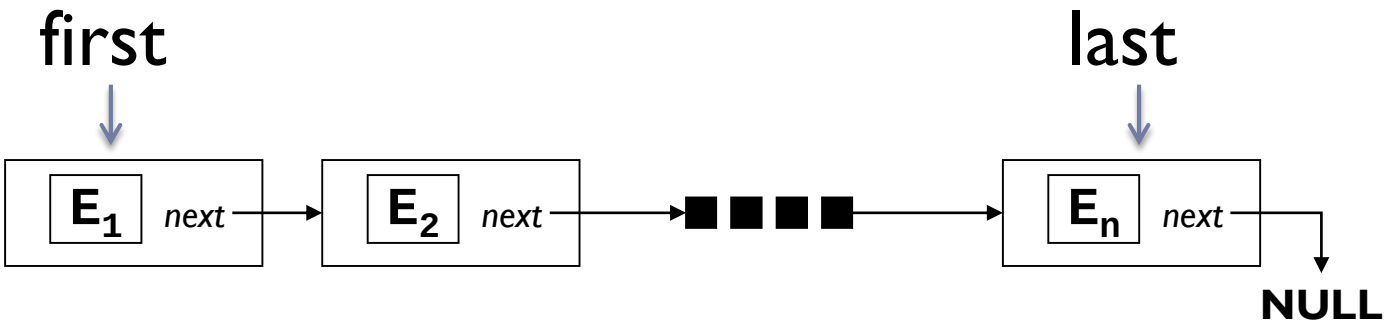
Tema 2.4.2 TAD Lineales
TAD Lista: Implementación con una Lista Doblemente Enlazada

Estructura de Datos y Algoritmos (EDA)

Outline

- 2.1. ¿Qué es un TAD Lineal?
- 2.2. TAD Pila
- 2.3. TAD Cola
- 2.4. TAD Lista
  - 2.4.1 Implementación con una Lista Simplemente Enlazada
  - **2.4.2 Implementación con una Lista Doblemente Enlazada**

Implementación con un TAD Lista Simplemente Enlazada



- Un **node** es un **Objet** que almacena una referencia (**Ei**) a un objeto (el elemento de la Lista) y una referencia (next) al node que almacena el siguiente elemento en la Lista
- **first** y **last** permiten acceder al primer y al último elemento de la Lista respectivamente
- Para visitar un elemento, debemos comenzar a visitar el first node y pasar al siguiente node (por next) de forma iterativa, hasta llegar al node que contiene el elemento buscado

Ventajas de Lista Simplemente Enlazada

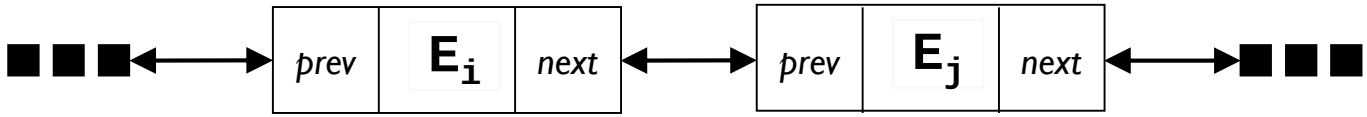
- Las inserciones y eliminaciones se pueden realizar fácilmente (no es necesario mover elementos), como sucedía en los arrays
- El tamaño de la Lista no es fijo (puede ampliarse o reducirse según los requisitos)
- Uso eficiente de la memoria (el espacio no se desperdicia)

Desventajas de Lista Simplemente Enlazada

- Si necesitamos llegar a un elemento en particular (por ejemplo, en el medio de la Lista), tenemos que pasar por todos los elementos que lo preceden
- En particular, si necesitamos un elemento, tenemos que llegar a su nodo predecesor
- Solo podemos atravesar la Lista desde el principio, nunca desde el final. Esto no es muy eficiente cuando buscas un elemento que está en las últimas posiciones

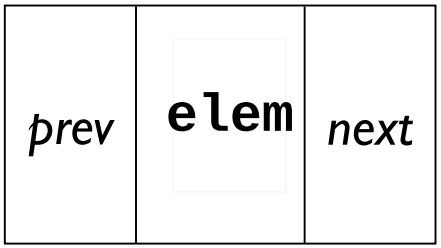
¿Cómo mejorar el acceso a los nodos? Listas Doblemente Enlazadas

- Se permite visitar la lista de izquierda a derecha, y también a la inversa
- Además de la referencia **next**, un node necesita una referencia adicional al node anterior (**prev**).



Clase DNode

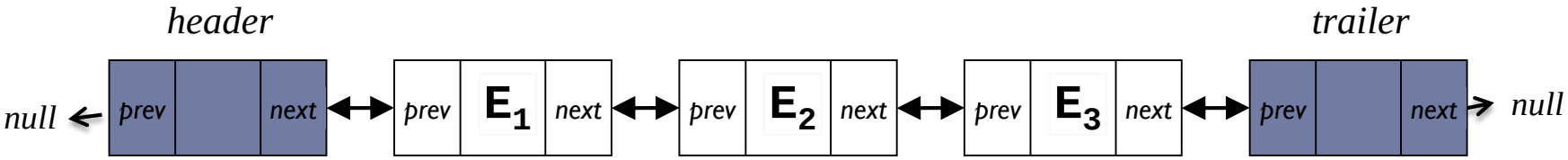
```
public class DNode {

    Object elem;
    DNode prev;
    DNode next;

    public DNode(Object elem) {
        this.elem = elem;
    }
}
```



Recuerde que elem puede pertenecer a cualquier tipo de datos: String, Integer, Point, etc

Nodos Centinelas

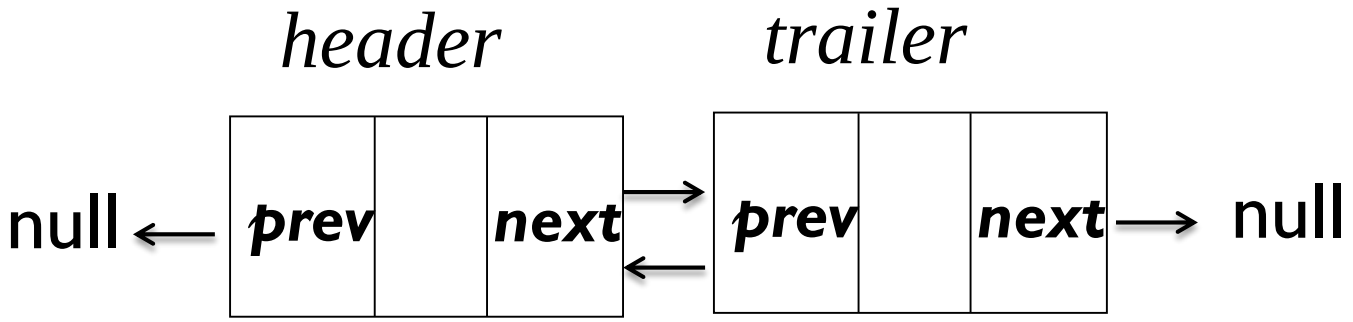
- No almacenan ningún elemento
- Situado en ambos extremos de una Lista Doblemente Enlazada.
  - **header** node justo antes de la cabeza de la Lista
  - **trailer** node justo después de la cola de la Lista



Dlist class

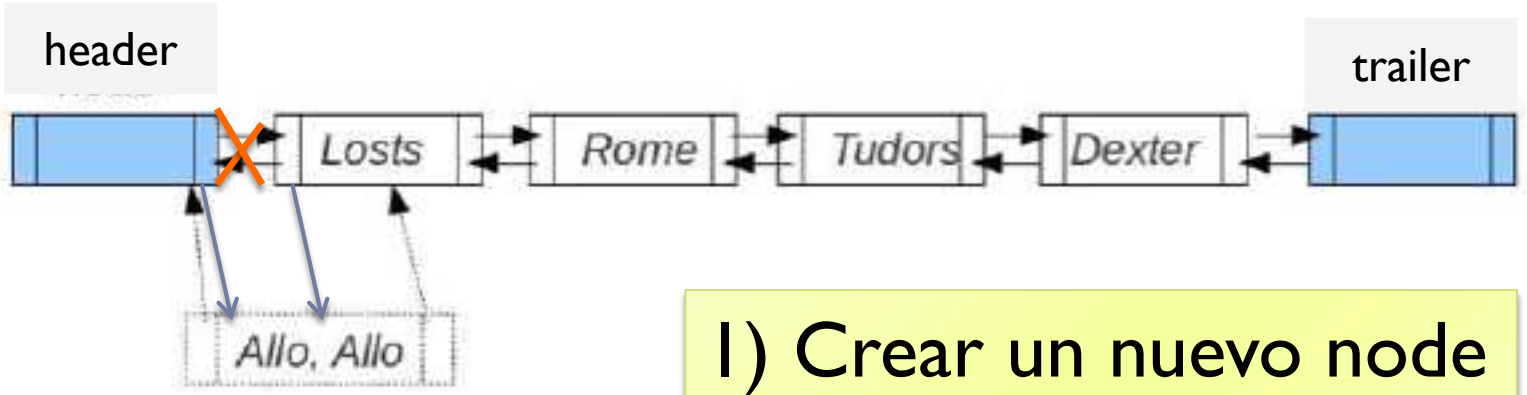
```
/**
 * A double-linked list class with sentinel nodes
 */
public class DList implements IList {

    DNode header;
    DNode trailer;
    int size=0;
```

¿Cómo crear una Lista vacía?

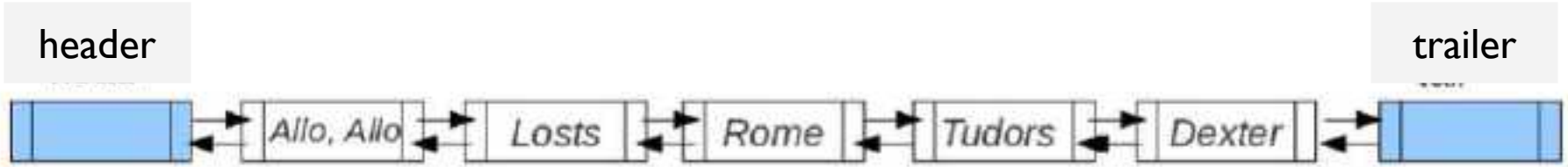



```
public DList() {
    header = new DNode(null);
    trailer = new DNode(null);
    header.next = trailer;
    trailer.prev= header;
}
```

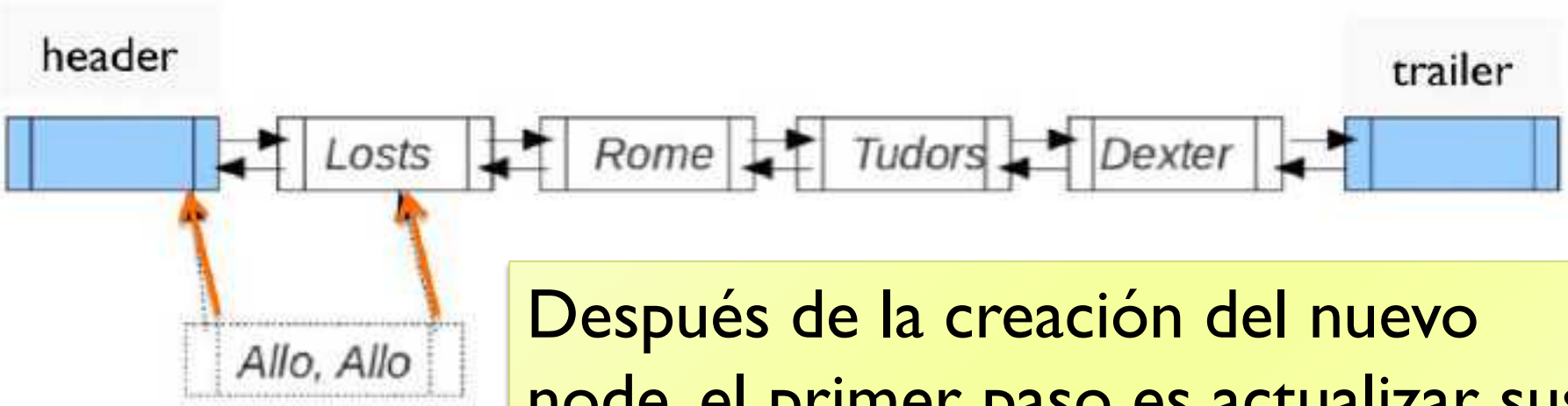
Implementación con un TAD Lista Doblemente Enlazada (addFirst)



1) Crear un nuevo node
2) Enlazar a la lista



Implementación con un TAD Lista Doblemente Enlazada (addFirst)

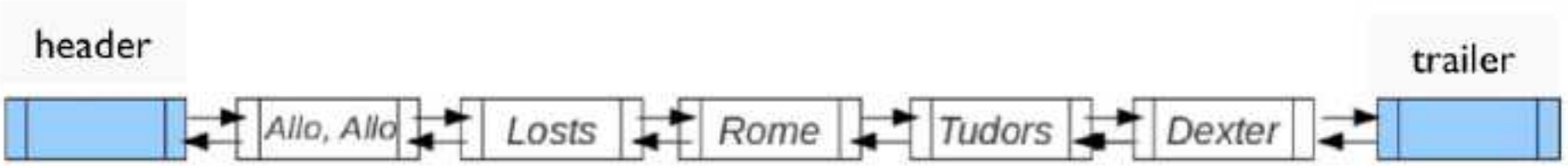


Después de la creación del nuevo node, el primer paso es actualizar sus referencias next y prev

DNode newNode=new DNode ("Allo,Allo");
newNode.next=header.next;
newNode.prev=header;

Implementación con un TAD Lista Doblemente Enlazada (addFirst)

Finalmente, tenemos que actualizar las referencias de la lista para apuntar al nuevo node



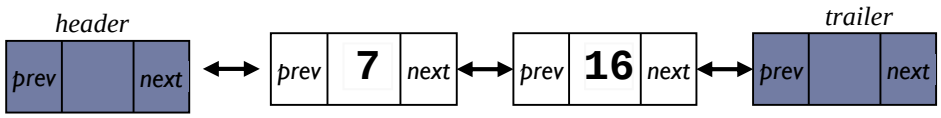
DNode newNode=new DNode("Allo,Allo");
newNode.next=header.next;
newNode.prev=header;
header.next.prev=newNode;
header.next=newNode;

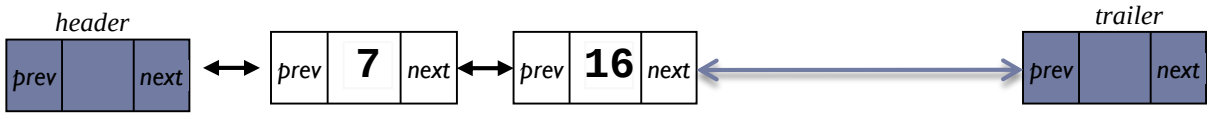
Implementación con un TAD Lista Doblemente Enlazada (addFirst)

```
public void addFirst(String elem) {
    DNode newNode = new DNode(elem);
    newNode.next = header.next;
    newNode.prev= header;
    header.next.prev= newNode;
    header.next = newNode;
    size++;
}
```

¿Qué sucede si cambiamos el orden de estas instrucciones?

Implementación con un TAD Lista Doblemente Enlazada (addLast)

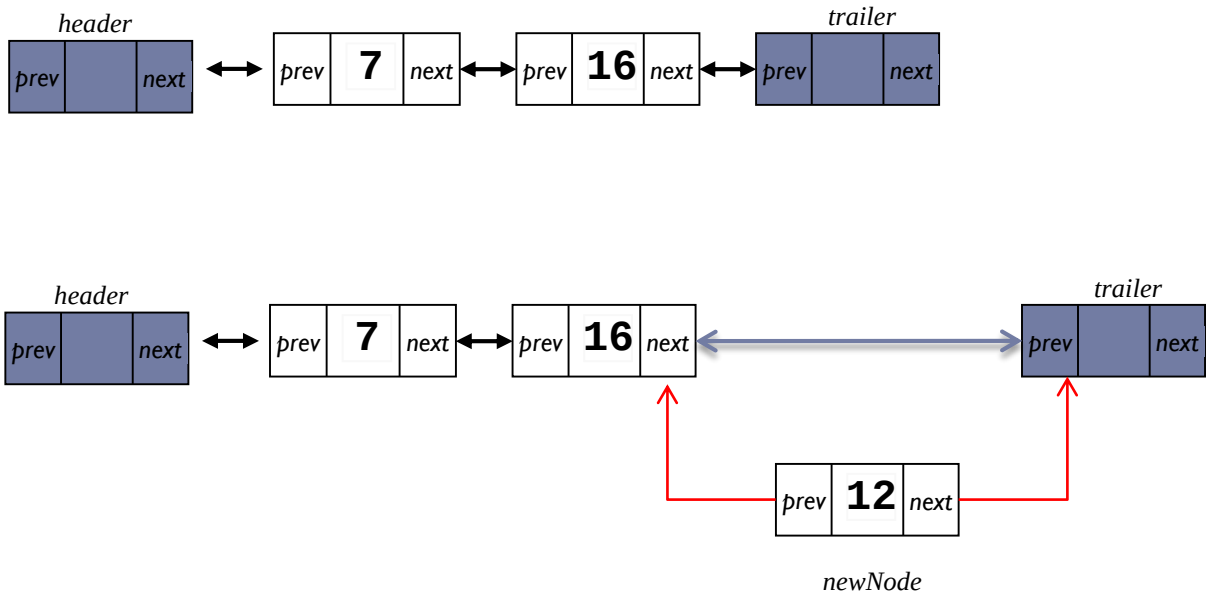




newNode

DNode newNode=new DNode (12);

Implementación con un TAD Lista Doblemente Enlazada (addLast)

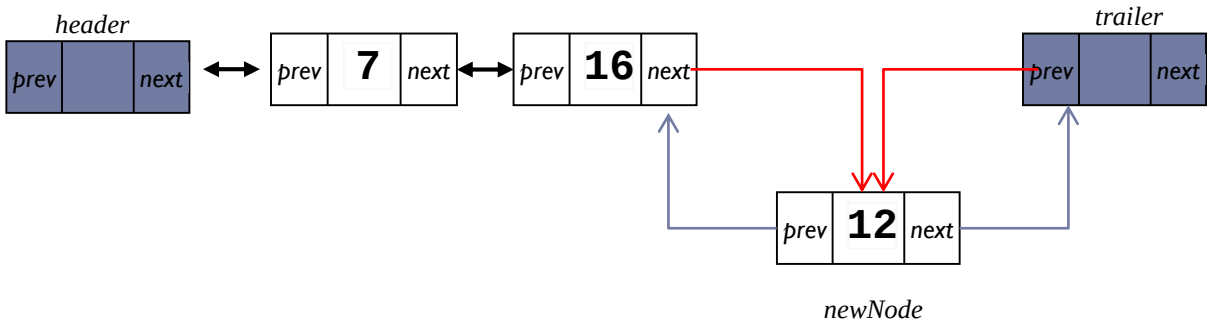


DNode newNode=new DNode (12);

newNode.next=tailer;

newNode.prev=tailer.prev;

Implementación con un TAD Lista Doblemente Enlazada (addLast)



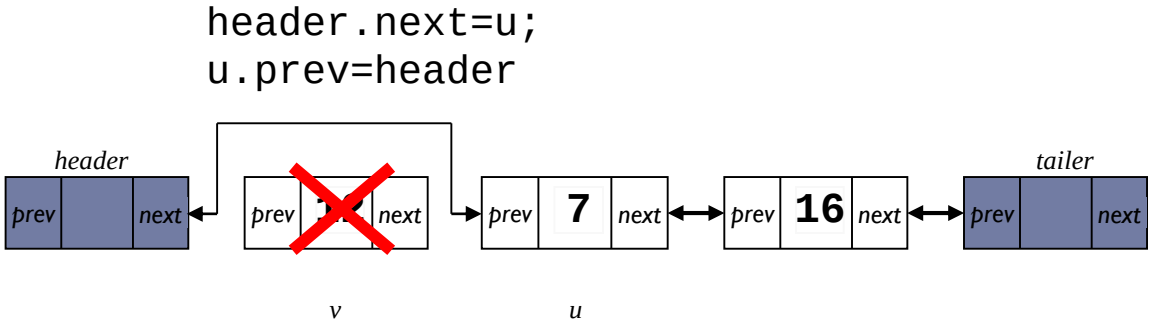
Dnode newNode=new Dnode(12);
newNode.next=tailer;
newNode.prev=tailer.prev;
tailer.prev.next=newNode;
tailer.prev=newNode;

Implementación con un TAD Lista Doblemente Enlazada (addLast)

```
public void addLast(String elem) {
      DNode newNode = new DNode(elem);
      newNode.next = trailer;
      newNode.prev= trailer.prev;
      trailer.prev.next = newNode;
      trailer.prev= newNode;
      size++;
}
```

¿Funciona cuando la lista está vacía?

Implementación con un TAD Lista Doblemente Enlazada (removeFirst)



header.next=header.next.next;
header.next.prev=header;

Implementación con un TAD Lista Doblemente Enlazada (removeFirst)

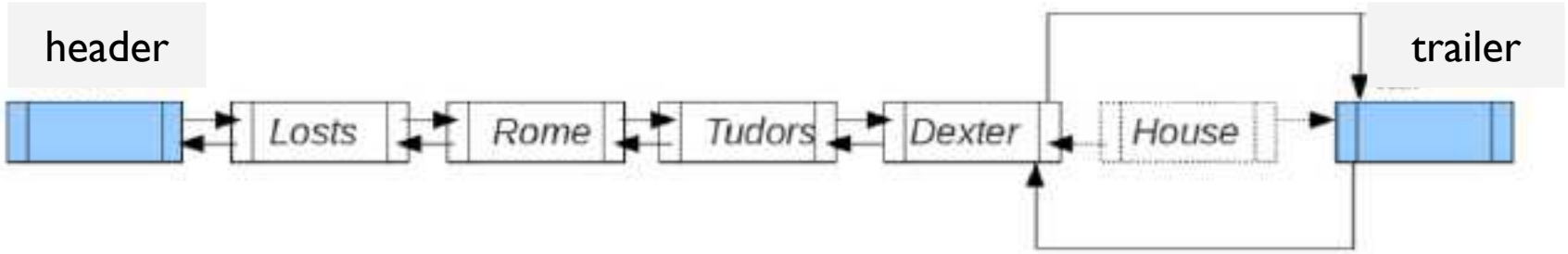
Encuentra el error en el método:

```
public void removeFirst() {
    header.next=header.next.next;
    header.next.prev=header;
}
```

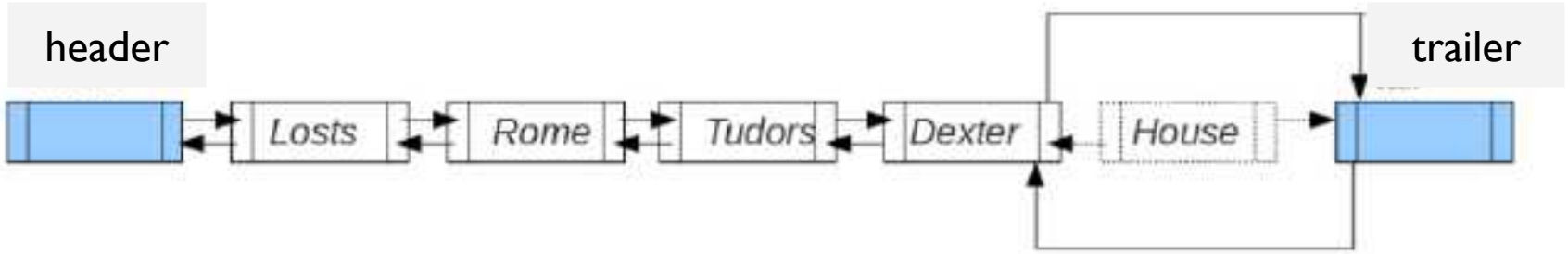
Implementación con un TAD Lista Doblemente Enlazada (removeFirst)

```
public void removeFirst() {
    if (isEmpty()) {
        System.out.println("DList: List is empty");
        return;
    }
    header.next = header.next.next;
    header.next.prev= header;
    size--;
}
```

Implementación con un TAD Lista Doblemente Enlazada (removeLast)

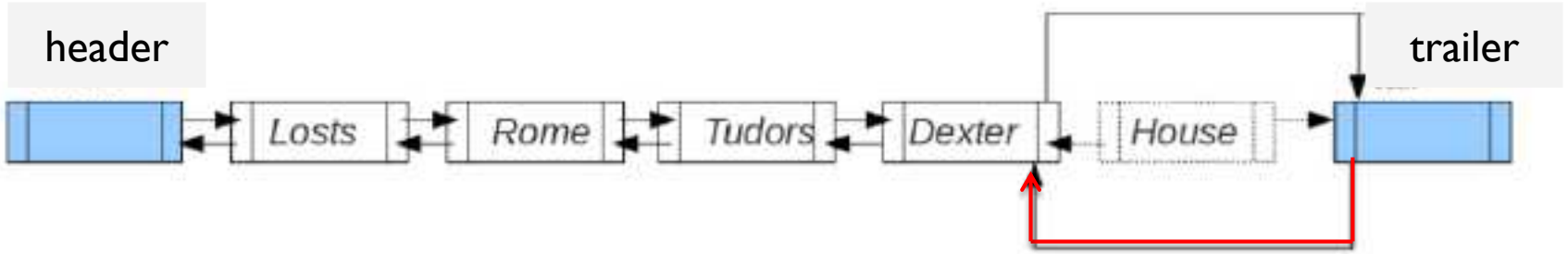


Implementación con un TAD Lista Doblemente Enlazada (removeLast)



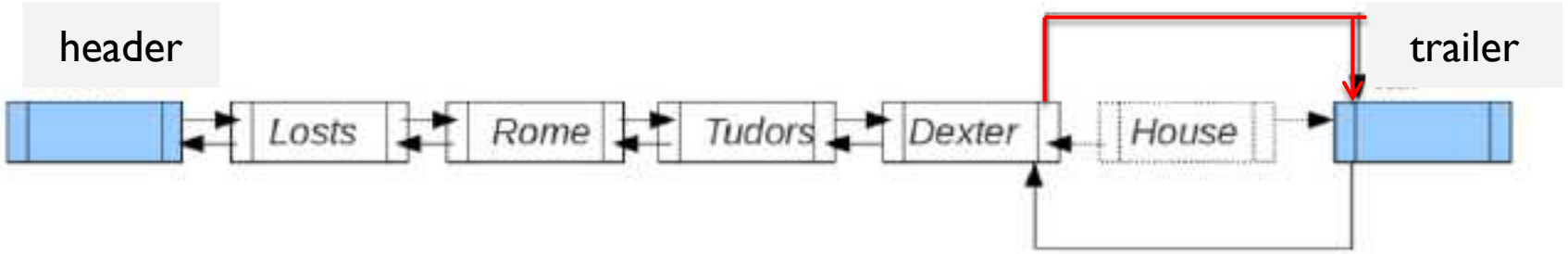
¿Tienes que recorrer la lista para encontrar el penúltimo nodo?

Implementación con un TAD Lista Doblemente Enlazada (removeLast)




```
trailer.prev=trailer.prev.prev;
```

Implementación con un TAD Lista Doblemente Enlazada (removeLast)



trailer.prev=tailer.prev.prev;

trailer.prev.next=trailer;

Implementación con un TAD Lista Doblemente Enlazada (removeLast)

Encuentra el error en el método:

```
public void removeLast() {
    tailer.prev=tailer.prev.prev;
    tailer.prev.next=tailer;
}
```

Implementación con un TAD Lista Doblemente Enlazada (removeLast)

```
public void removeLast() {
    if (isEmpty()) {
        System.out.println("DList: List is empty");
        return;
    }
    trailer.prev= trailer.prev.prev;
    trailer.prev.next = trailer;
    size--;
}
```

Implementar el resto de los métodos

- int getSize()
- void insertAt(int index, Object elem)
- Object getAt(int index)
- void removeAt()
- boolean isEmpty()
- void removeAll(Object elem)