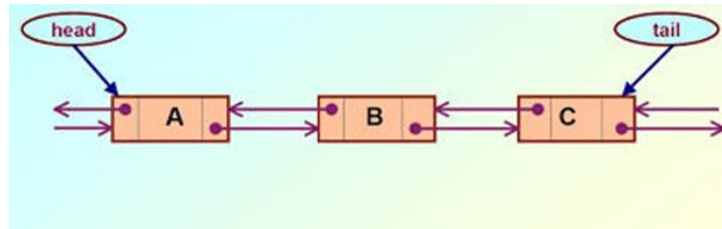
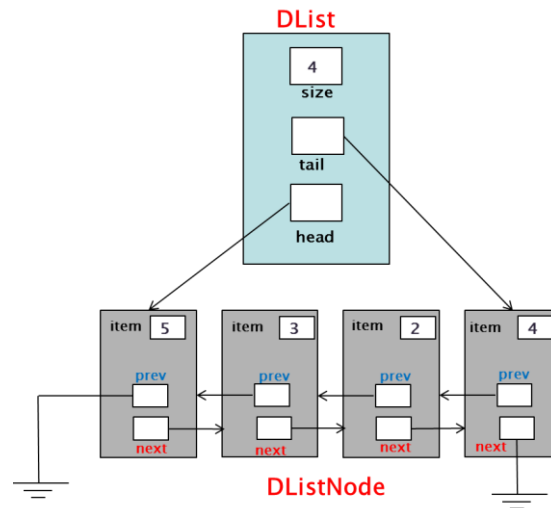


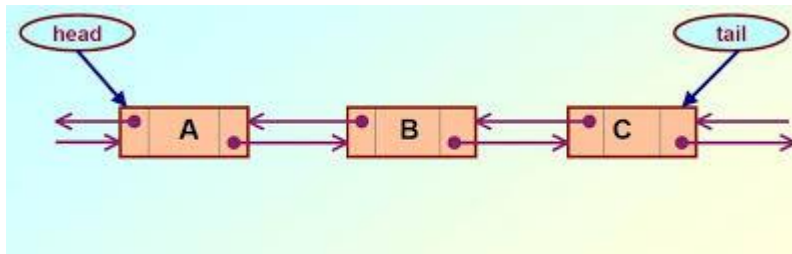
# Unidade 10 – Análise de Algoritmos com Estruturas de Dados Lineares – Parte 3





Porque listas duplamente ligadas



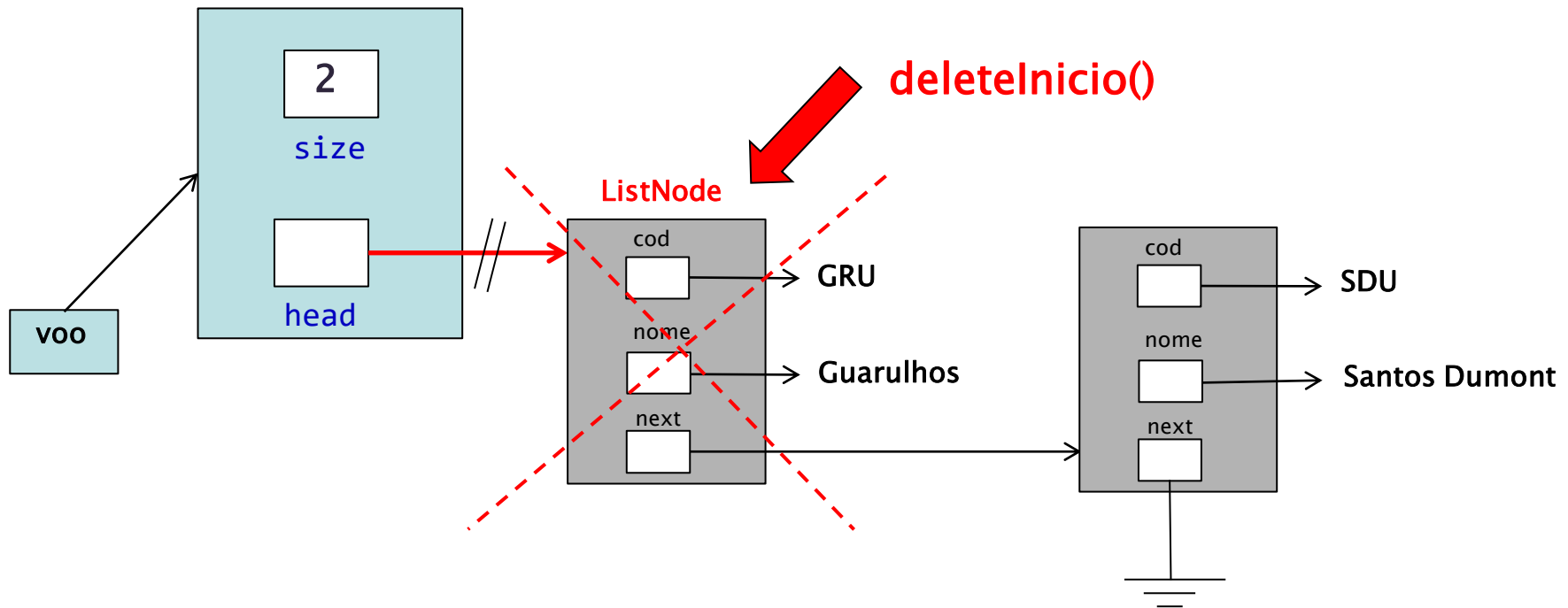


Qual a necessidade de listas duplamente ligadas ?



## Porque listas duplamente ligadas ?

- Numa lista simplesmente ligada, a **deleção** ou **inserção** do elemento da frente é muito fácil e com esforço computacional (tempo) constante.



## Função deleteInicio

```
public void deleteInicio() {  
    if (this.head == null )  
        System.out.println("Impossível deletar... Lista vazia...");  
    else {  
        this.head = this.head.next;  
        this.size--;  
    }  
}
```



**O Tempo é constante e independe do tamanho da lista!**

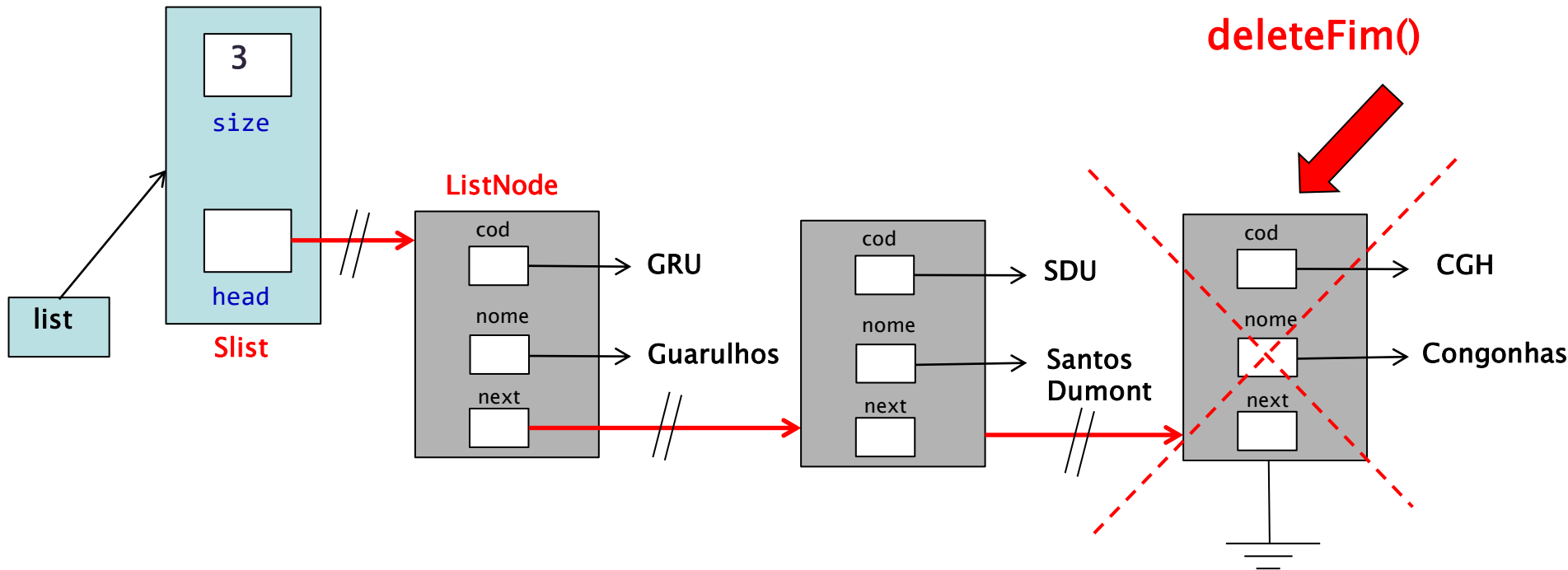


Mas, numa lista simplesmente ligada, a deleção  
ou remoção no final da lista é **difícil ...**



# Deleção em lista simplesmente ligada

- Numa lista simplesmente ligada, para se remover qualquer item (exceto o primeiro) será necessário percorrer-se toda a lista, uma vez que não se tem acesso rápido ao nó predecessor (só há pointer direto ao primeiro nó !).



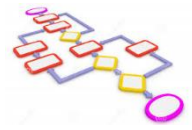
## Função deleteFim()

```
public void deleteFim() {  
  
    int contador = 1;  
  
    if ( this.size == 0 )  
        System.out.println("Erro: Lista vazia...");  
    else  
        if (this.size == 1) {  
            this.head = null;  
            this.size--;  
        }  
        else {  
  
            Node trab = this.head;  
  
            while (contador < this.size - 1) {  
                trab = trab.next;  
                contador++;  
            }  
            trab.next = null;  
            this.size--;  
        }  
    }  
}
```

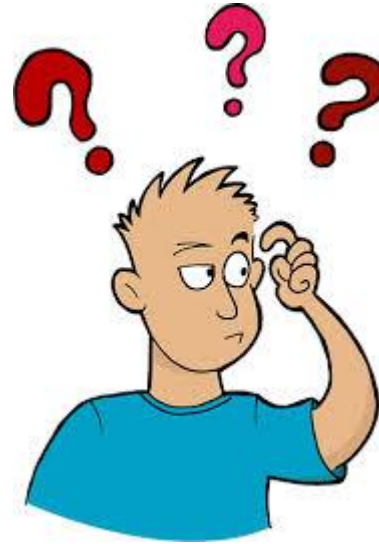
**O Tempo é proporcional ao tamanho da lista !**



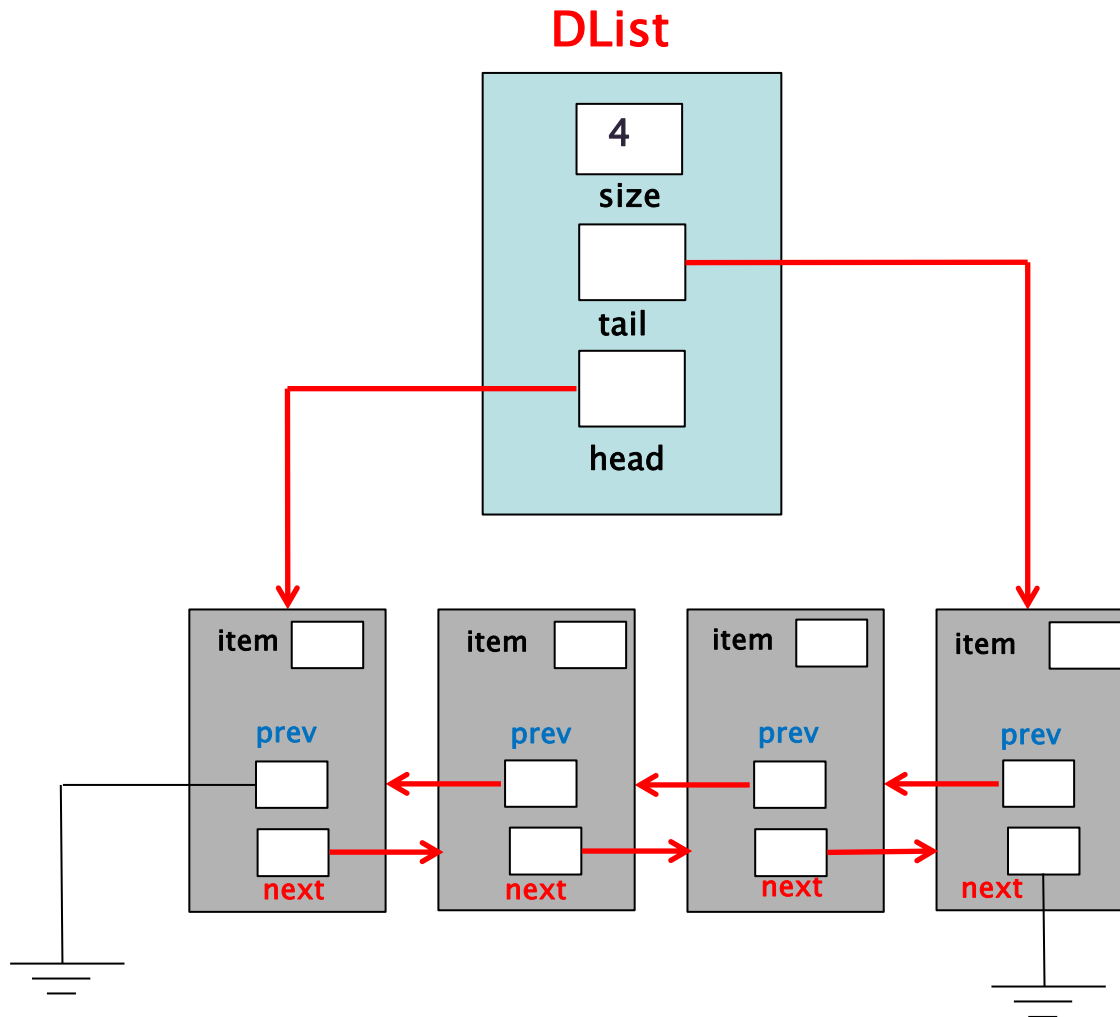




Como então melhorar essa estrutura de dados ?



# Estrutura Dlist



# Listas duplamente ligadas

```
package maua;
```

```
public class DList {
```

```
    public int size;
```

```
    public DListNode head;
```

```
    public DListNode tail;
```

```
}
```

```
package maua;
```

```
public class DListNode {
```

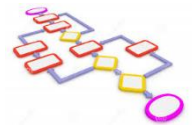
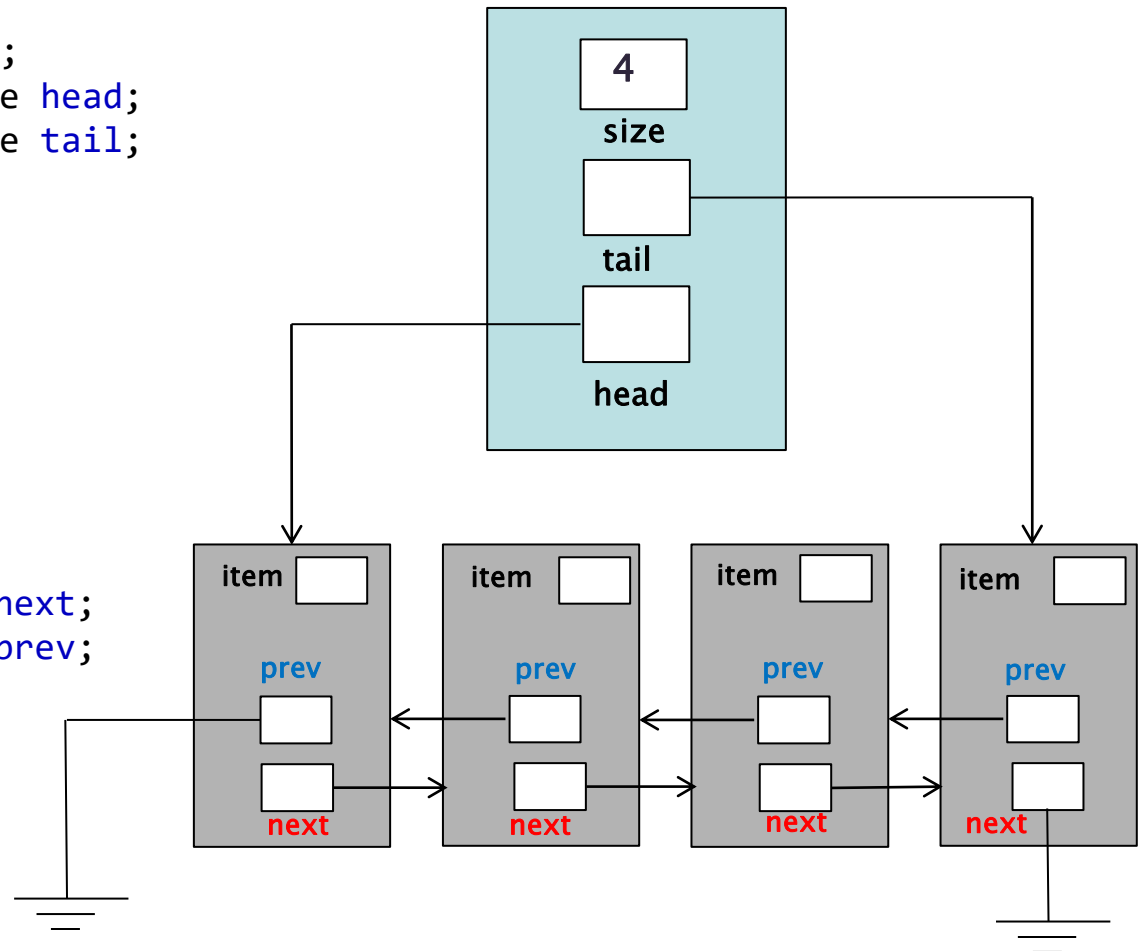
```
    public int item;
```

```
    public DListNode next;
```

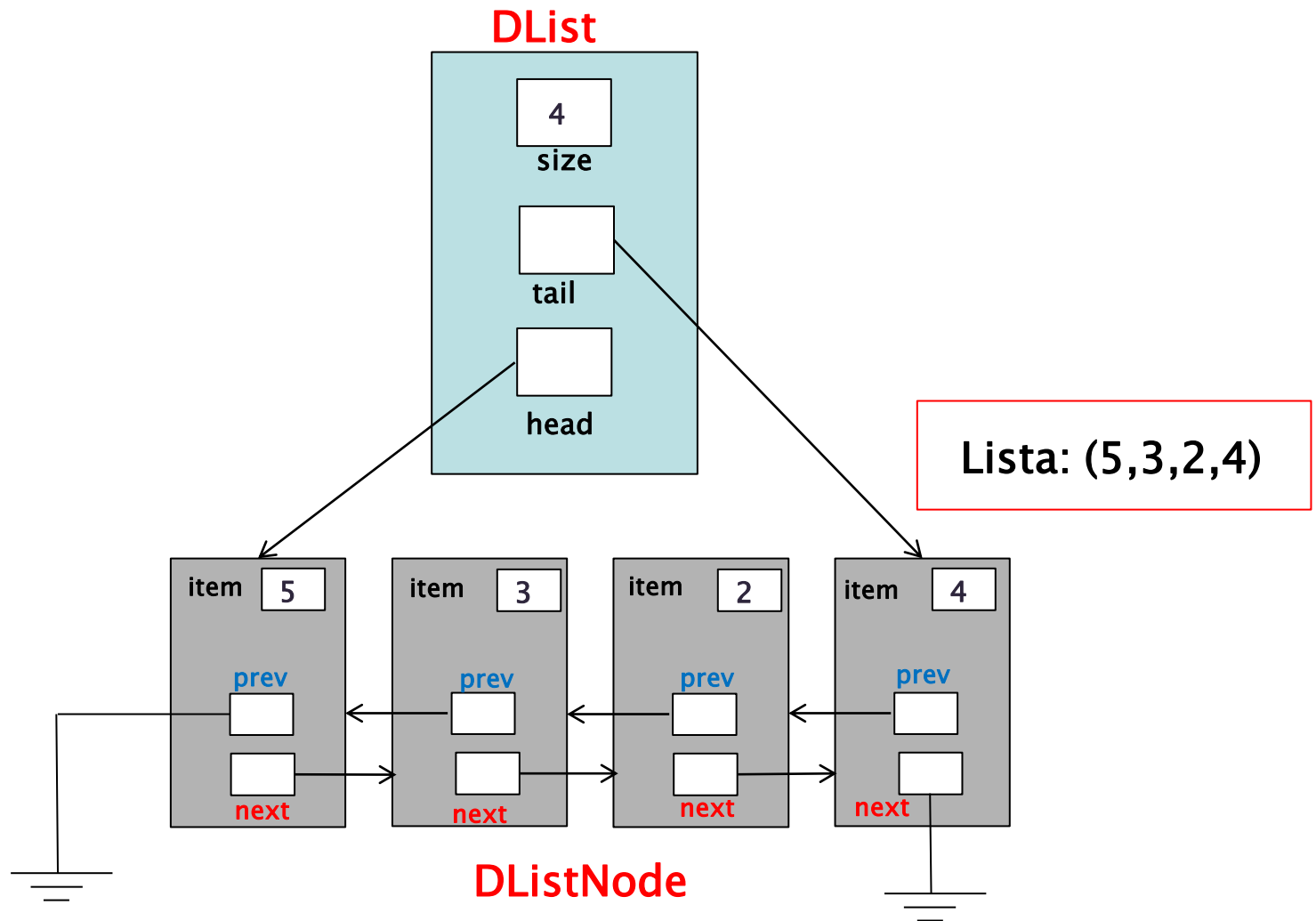
```
    public DListNode prev;
```

```
}
```

**DList**



# Exemplo: Lista Duplamente ligada



## Criando Estrutura do Nó

```
package maua;  
  
public class DListNode {  
  
    public int item;  
    public DListNode next;  
    public DListNode prev;  
}
```



## Criando construtores do nó

```
public DListNode() {  
    this.item = 0;  
    this.next = null;  
    this.prev = null;  
}
```

```
public DListNode(int item) {  
  
    this.item = item;  
    this.next = null;  
    this.prev = null;  
}
```



## Criando a estrutura de Controle da Lista Duplamente Ligada

```
package maua;  
  
public class DList {  
  
    public int size;  
    public DListNode head;  
    public DListNode tail;  
}
```



## Construtores da classe DList

```
public DList( int item) {  
  
    DListNode trab = new DListNode(item);  
  
    trab.next = null;  
    trab.prev = null;  
    this.head = trab;  
    this.tail = trab;  
    this.size = 1;  
  
}  
  
public DList() {  
  
    this.size = 0;  
    this.head = null;  
    this.tail = null;  
  
}
```





## Função **ImprimeFirst()** – Pseudocódigo

```
imprimeFirst() {  
    if (head == null)  
        Print ("Lista vazia...")  
  
    else  
        Print(head.item)  
  
}
```



## Função ImprimeFirst()

```
public void imprimeFirst() {  
  
    if (this.head == null)  
        System.out.println("Lista vazia...");  
  
    else  
        System.out.println("Primeiro item: " + this.head.item );  
  
}
```



## Função `ImprimeLast()` – Pseudocódigo

```
imprimeLast() {  
    if (tail == null)  
        Print ("Lista vazia...")  
  
    else  
        Print(tail.item)  
}
```



## Função ImprimeLast()

```
public void imprimeLast() {  
  
    if (this.tail == null)  
        System.out.println("Lista vazia...");  
  
    else  
  
        System.out.println("Último item: " + this.tail.item);  
  
}
```



## Classe para teste

```
package maua;

public class TesteDList {

    public static void main(String[] args) {

        DList x= new DList();

        x.imprimeFirst();

        x.imprimeLast();

    }

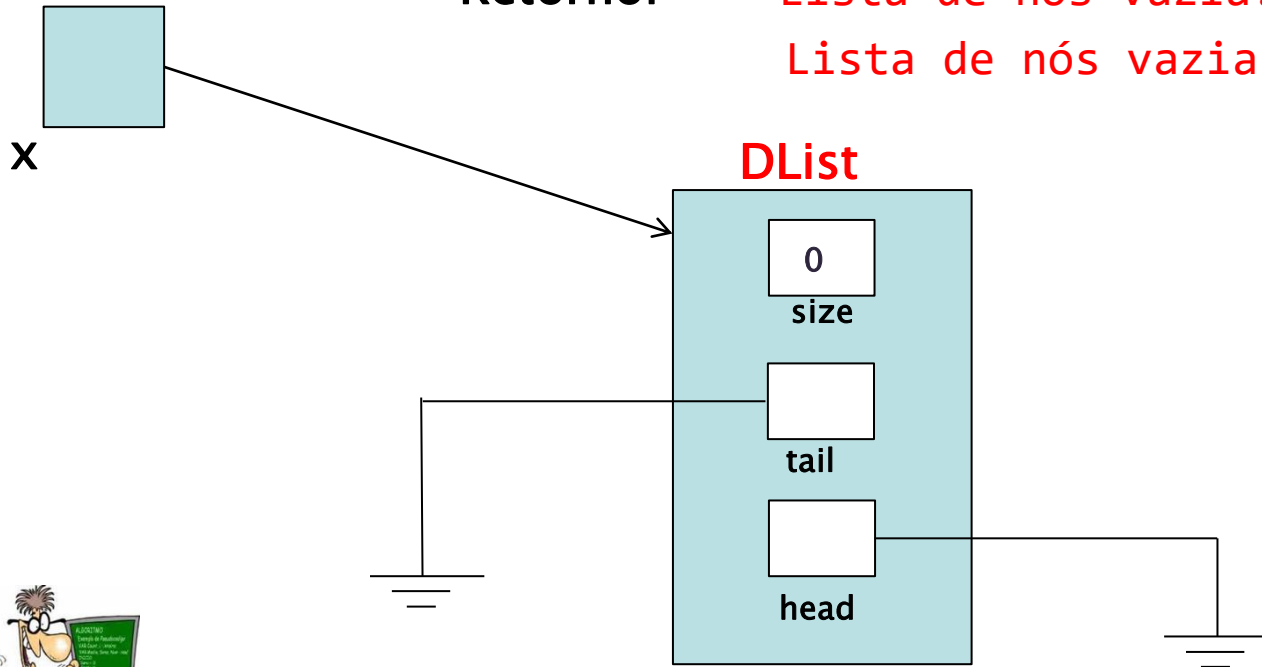
}
```

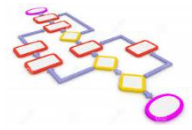


# Criando lista de nós vazia

```
DList x = new DList();  
  
x.imprimeFirst();  
  
x.imprimeLast();
```

**Retorno:**      Lista de nós vazia....  
                  Lista de nós vazia....

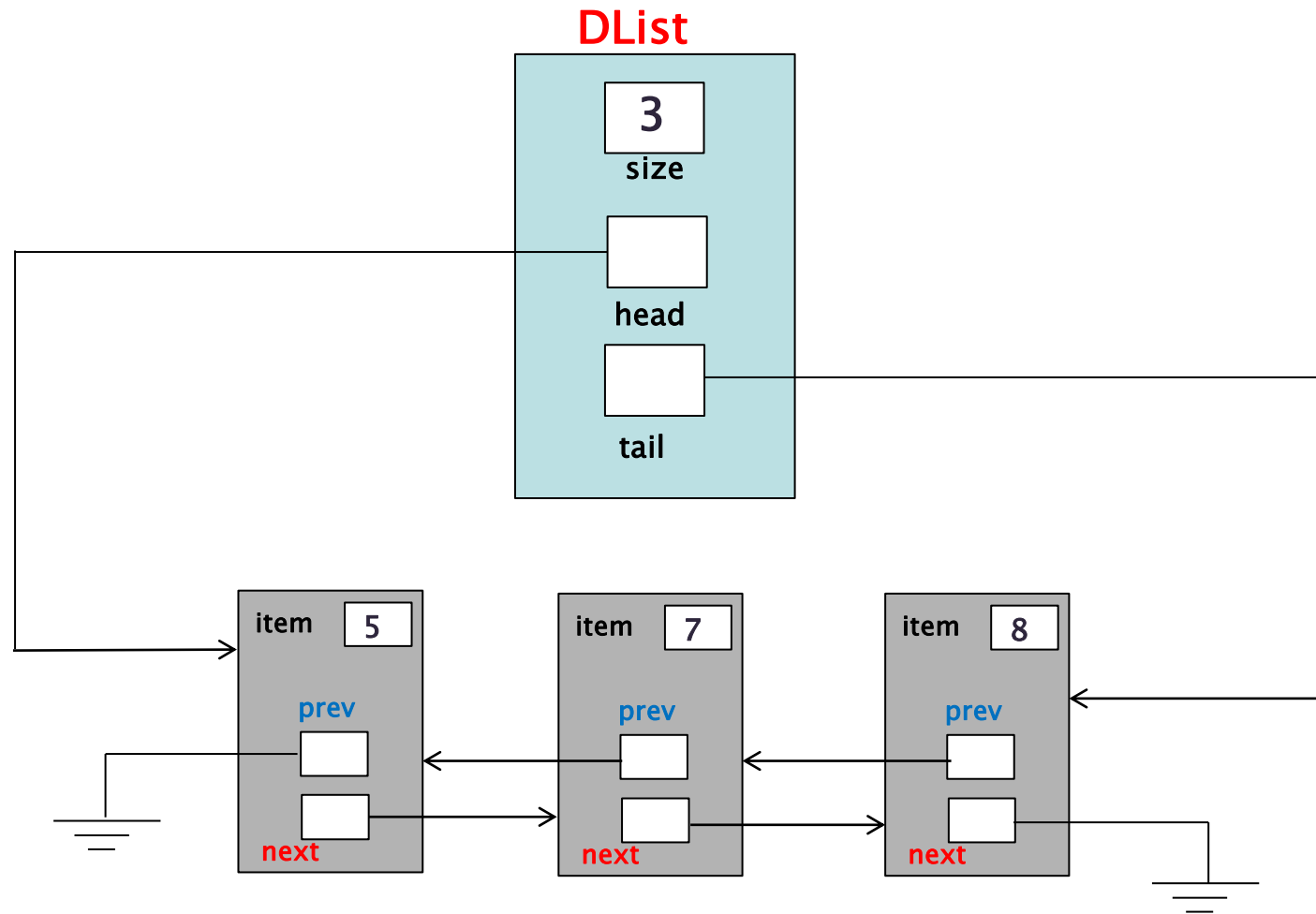




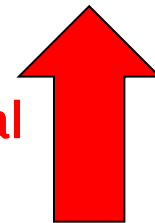
# Como inserir nós na Lista ?



# Inserindo novo nó no fim da lista



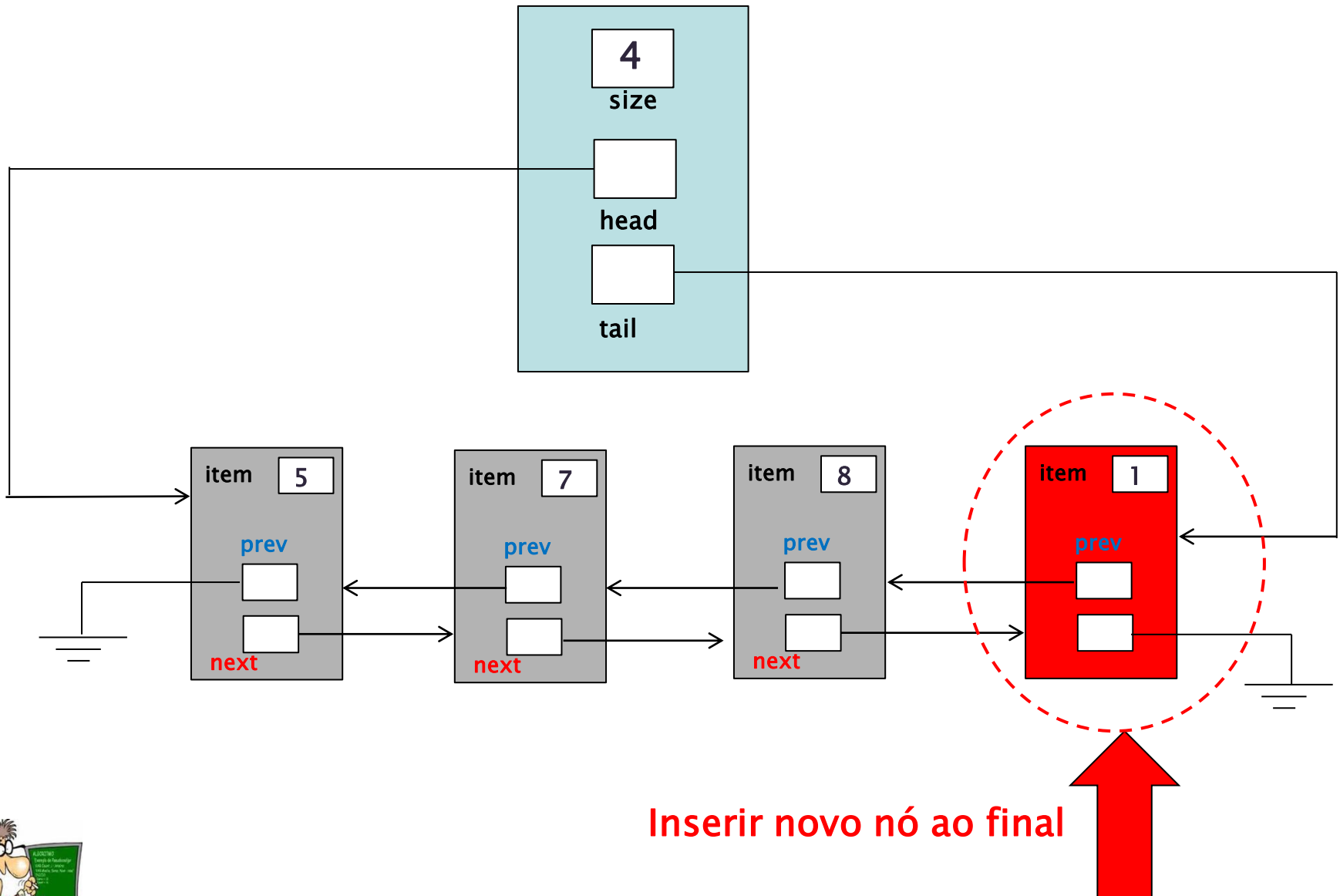
**Inserir novo nó ao final**





# Inserindo nó no fim da lista

**DList**

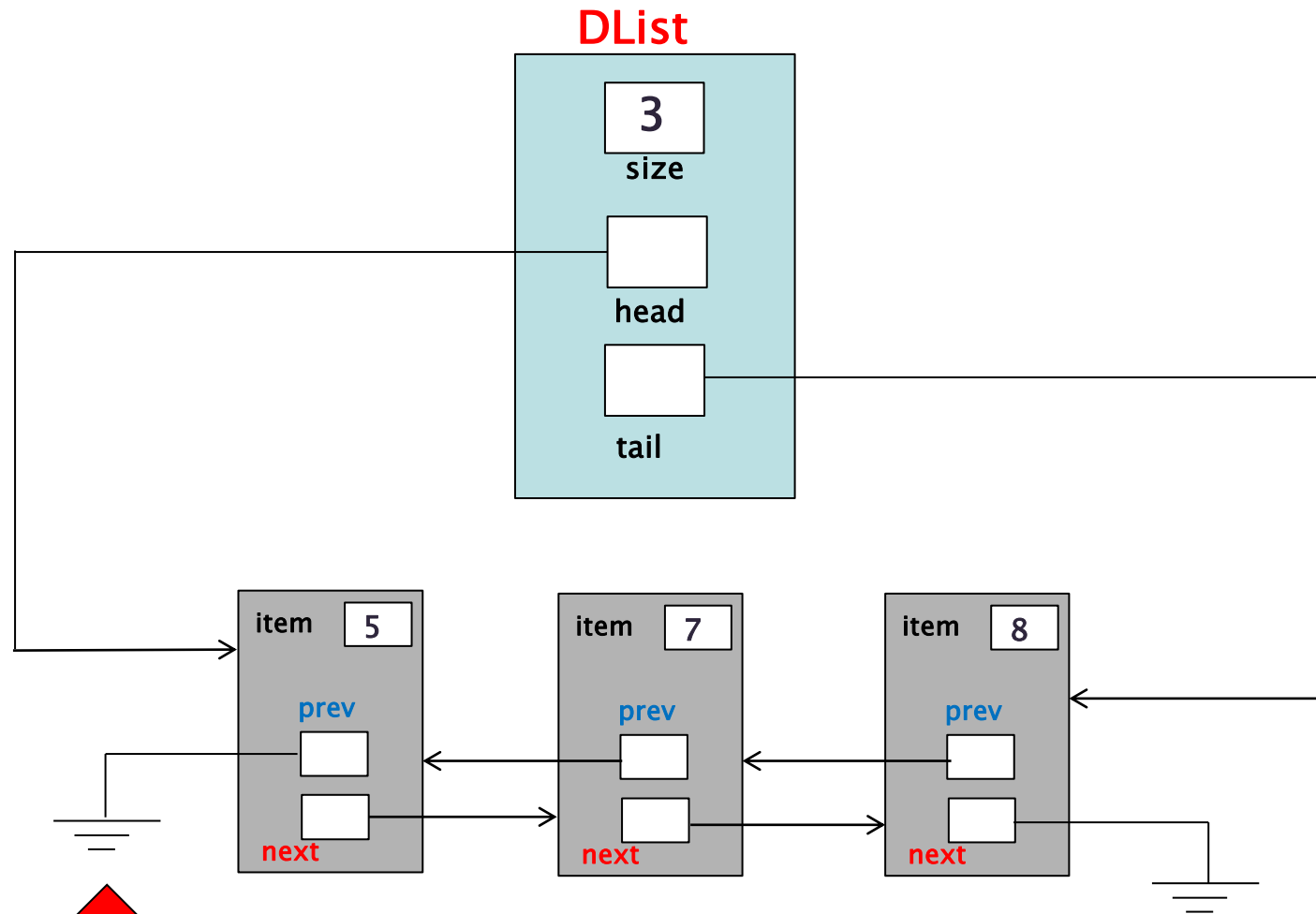


## Função `insereFim(item)`;

```
public void insereFim(int item) {  
  
    DListNode novoNode = new DListNode(item);  
  
    if (this.size == 0) {  
        this.head = novoNode;  
        this.tail = novoNode;  
        this.size++;  
    }  
    else {  
        this.tail.next = novoNode;  
        this.tail = novoNode;  
        this.size++;  
    }  
}
```



# Inserindo novo nó no início da lista



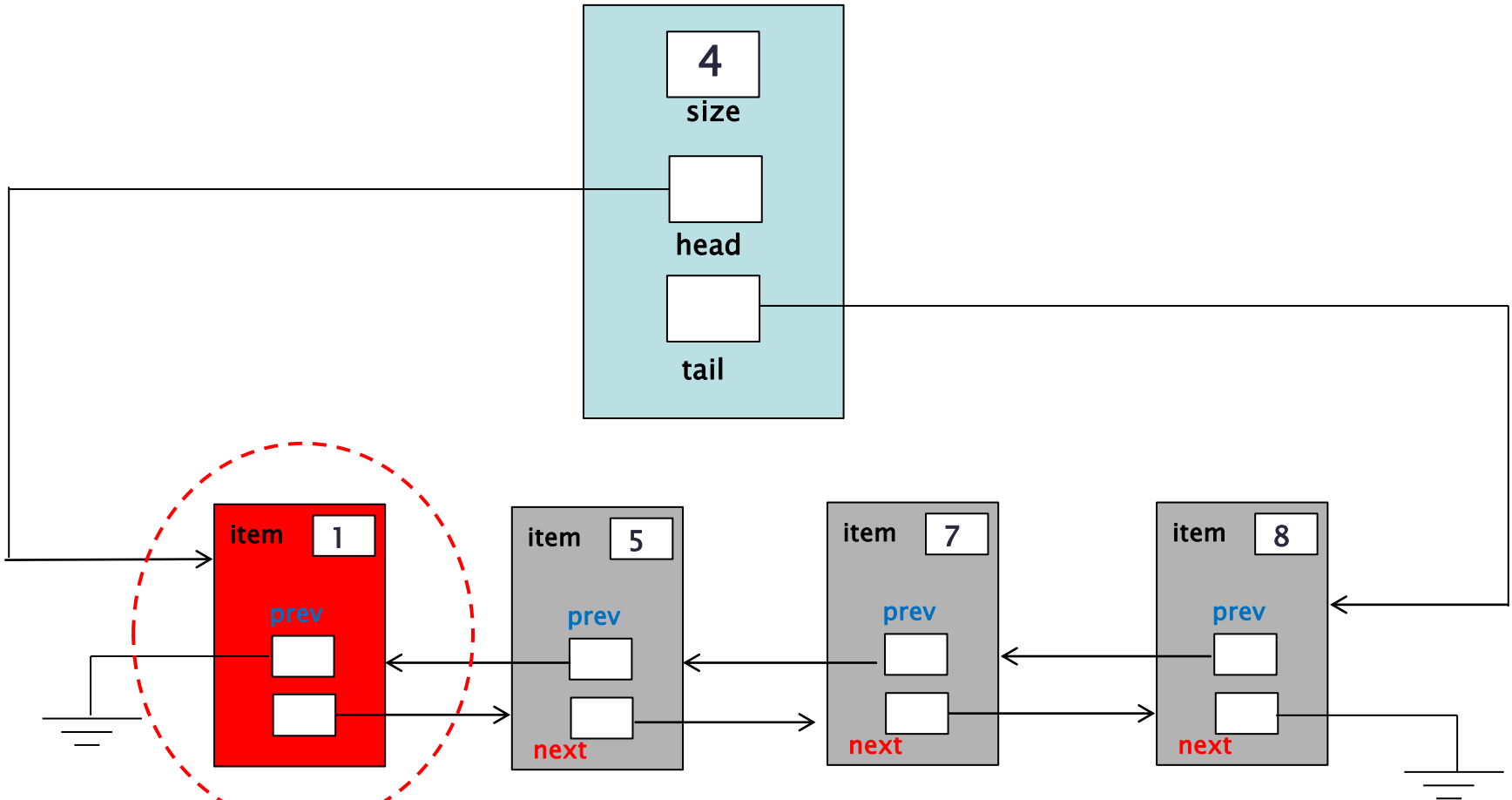
**Inserir novo nó no início**





# Inserindo nó no início da lista

## DList



**Inserir novo nó no início**

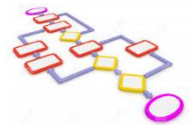


## Função `insereInicio(item)`;

```
public void insereInicio(int item) {  
  
    DListNode novoNode = new DListNode(item);  
  
    if (this.size == 0) {  
        this.head = novoNode;  
        this.tail = novoNode;  
        this.size++;  
    }  
    else {  
        this.head.prev = novoNode;  
        novoNode.next = this.head;  
        this.head = novoNode;  
        this.size++;  
    }  
}
```

◆ Ordem de complexidade:  $O(1)$ .





Como imprimir todos os nós da lista ?



## Imprimindo nós da lista – Pseudocódigo

```
imprimeLista() {
    contador ← 0

    p ← head;

    if ( size == 0 )
        print ("Lista vazia...")
    else {
        while ( p != null ) {
            contador ← contador + 1
            print (contador)
            print (p.item);
            p ← p.next
        }
    }
}
```



## Imprimindo nós da lista

```
public void imprimeLista() {  
    int contador = 0;  
  
    DListNode p;  
  
    p = this.head;  
  
    if (this.size == 0 )  
        System.out.println("Lista vazia...");  
    else {  
        while ( p != null ) {  
            System.out.print ("\nNó: " + ++contador) ;  
            System.out.print ("      Item:  " + p.item + "\n");  
            p = p.next;  
        }  
    }  
}
```

◆ Ordem de complexidade:  $O(n)$ .





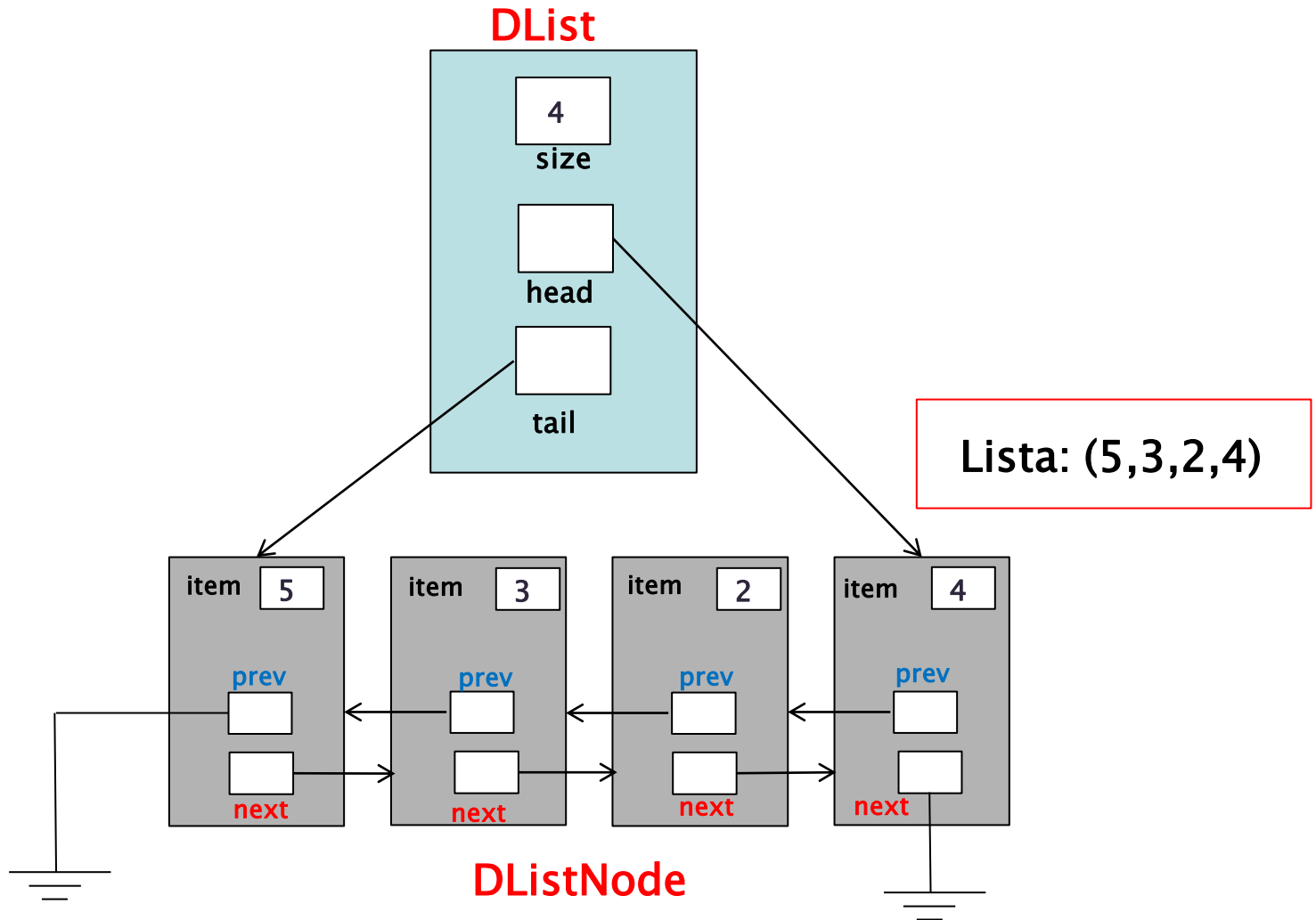
## Imprimindo nós da lista – Versão 2

```
public void imprimeLista2() {  
    int contador = 0;  
  
    DListNode p;  
  
    p = this.tail;  
  
    if (this.size == 0 )  
        System.out.println("Lista vazia...");  
    else {  
        while ( p != null ) {  
            System.out.print ("\nNó: " + ++contador) ;  
            System.out.print ("      Item: " + p.item + "\n");  
            p = p.prev;  
        }  
    }  
}
```

◆ Ordem de complexidade:  $O(n)$ .



# Gerando a lista completa



## Gerando a lista completa

```
package maua;

public class TesteDList {

    public static void main(String[] args) {

        DList x = new DList();
        x.imprimeLista();

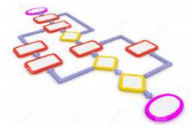
        x.insereInicio(4);
        x.insereInicio(2);
        x.insereInicio(3);
        x.insereInicio(5);

        x.imprimeLista();
        x.imprimeLista2();

    }

}
```

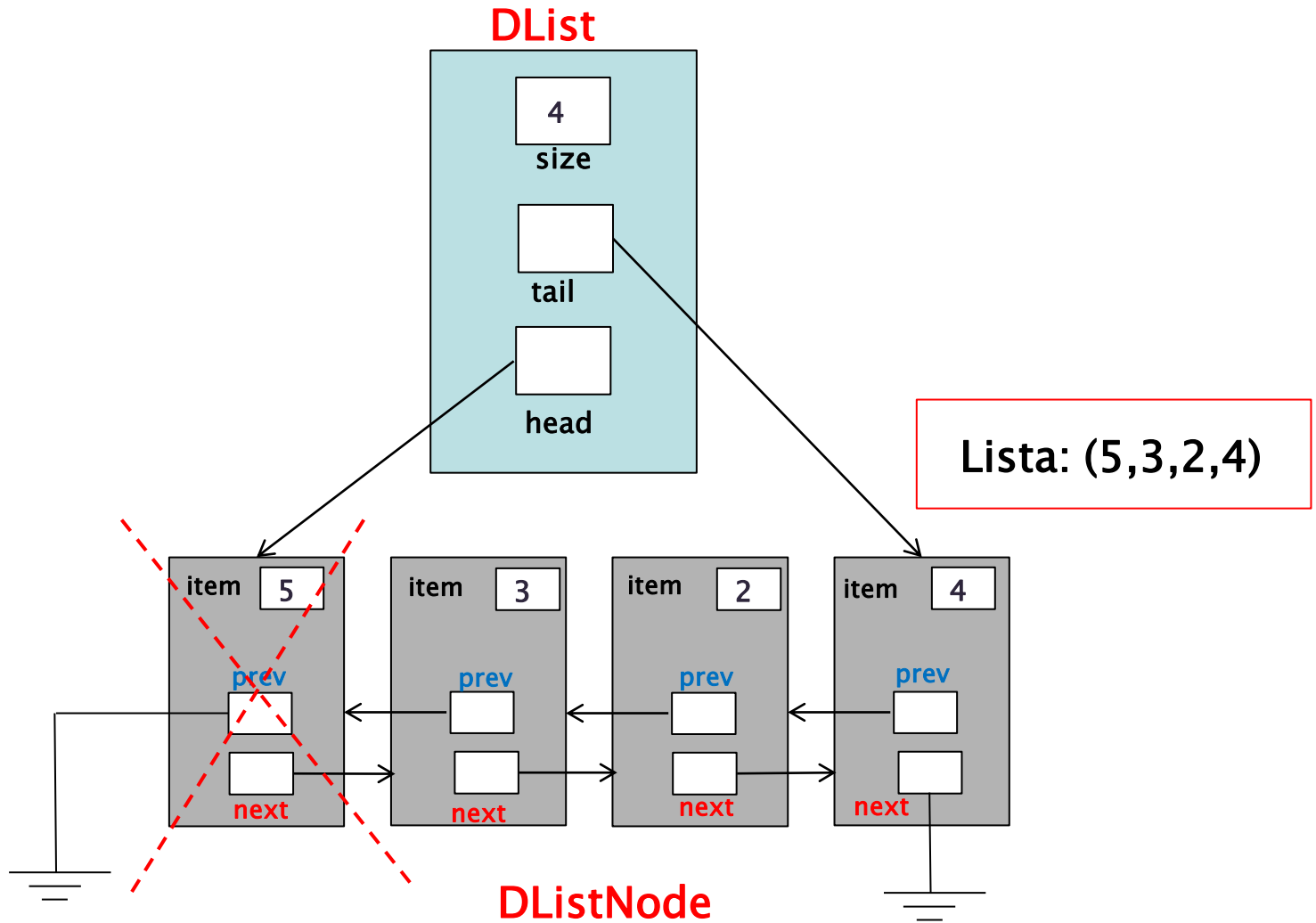




Como deletar um nó da lista ?



# Deletando um nó da lista



## Função deleteFirst() – Pseudocódigo

```
deleteFirst() {  
  
    if ( size == 0 )  
        print ("Deleção inválida ...")  
    else {  
        if (size == 1) {  
            head ← null  
            tail ← null  
            size ← size -1  
        }  
        else {  
            head ← head.next  
            head.prev ← null  
            size ← size -1  
        }  
    }  
}
```



## Função deleteFirst()

```
public void deleteFirst() {  
    if (this.size == 0)  
        System.out.println("Deleção inválida. Lista vazia...");  
    else {  
        if (this.size == 1) {  
            this.head = null;  
            this.tail = null;  
            this.size = 0;  
        }  
        else {  
            this.head = this.head.next;  
            this.head.prev = null;  
            this.size--;  
        }  
    }  
}
```

◆ Ordem de complexidade:  $O(1)$ .



## Função deleteLast() – Pseudocódigo

```
deleteLast() {  
  
    if ( size == 0 )  
        print ("Deleção inválida ...")  
    else {  
        if (size == 1) {  
            head ← null  
            tail ← null  
            size ← size -1  
        }  
        else {  
            tail ← tail.prev  
            tail.next ← null  
            size ← size -1  
        }  
    }  
}
```





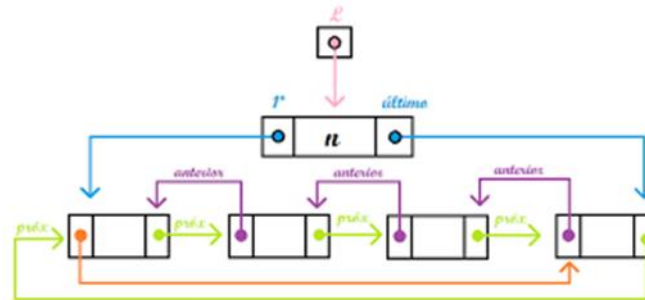
## Função deleteLast()

```
public void deleteLast() {  
    if (this.size == 0)  
        System.out.println("Deleção inválida. Lista vazia...");  
    else {  
        if (this.size == 1) {  
            this.head = null;  
            this.tail = null;  
            this.size = 0;  
        }  
        else {  
            this.tail = this.tail.prev;  
            this.tail.next = null;  
            this.size--;  
        }  
    }  
}
```

◆ Ordem de complexidade:  $O(1)$ .

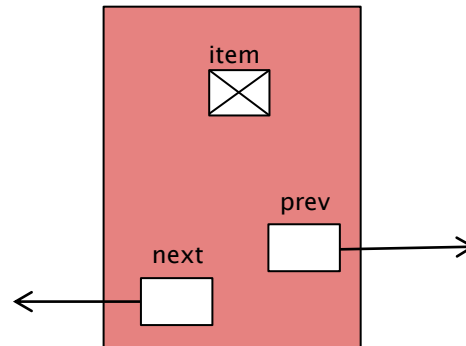


## Listas Circulares



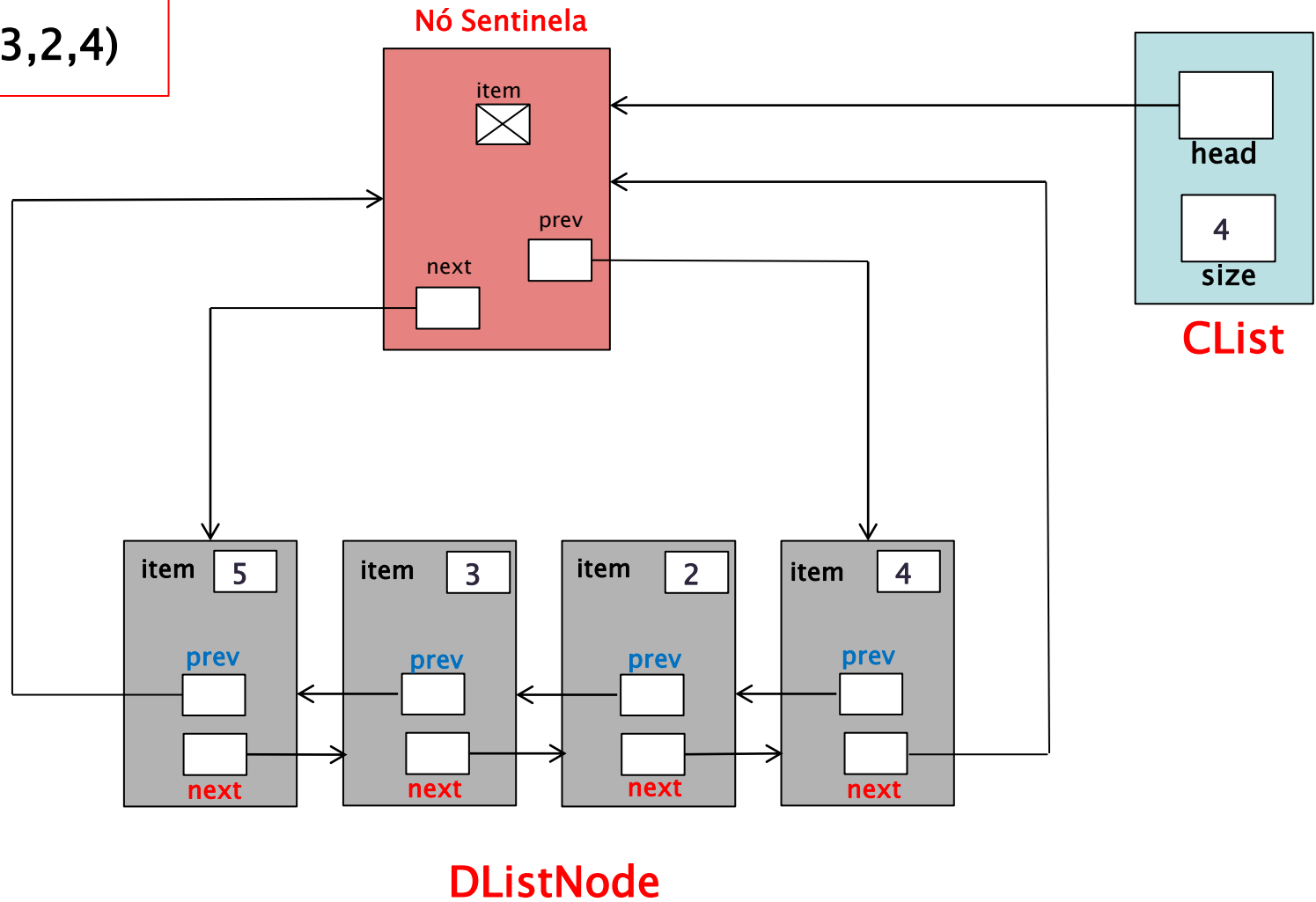
## Sentinela

- Numa lista ligada podemos definir um nó especial chamado **SENTINELA** no qual **NÃO** contém item de dados, mas apenas representando o início (head) e fim (tail) da lista ligada.



# Lista Circular

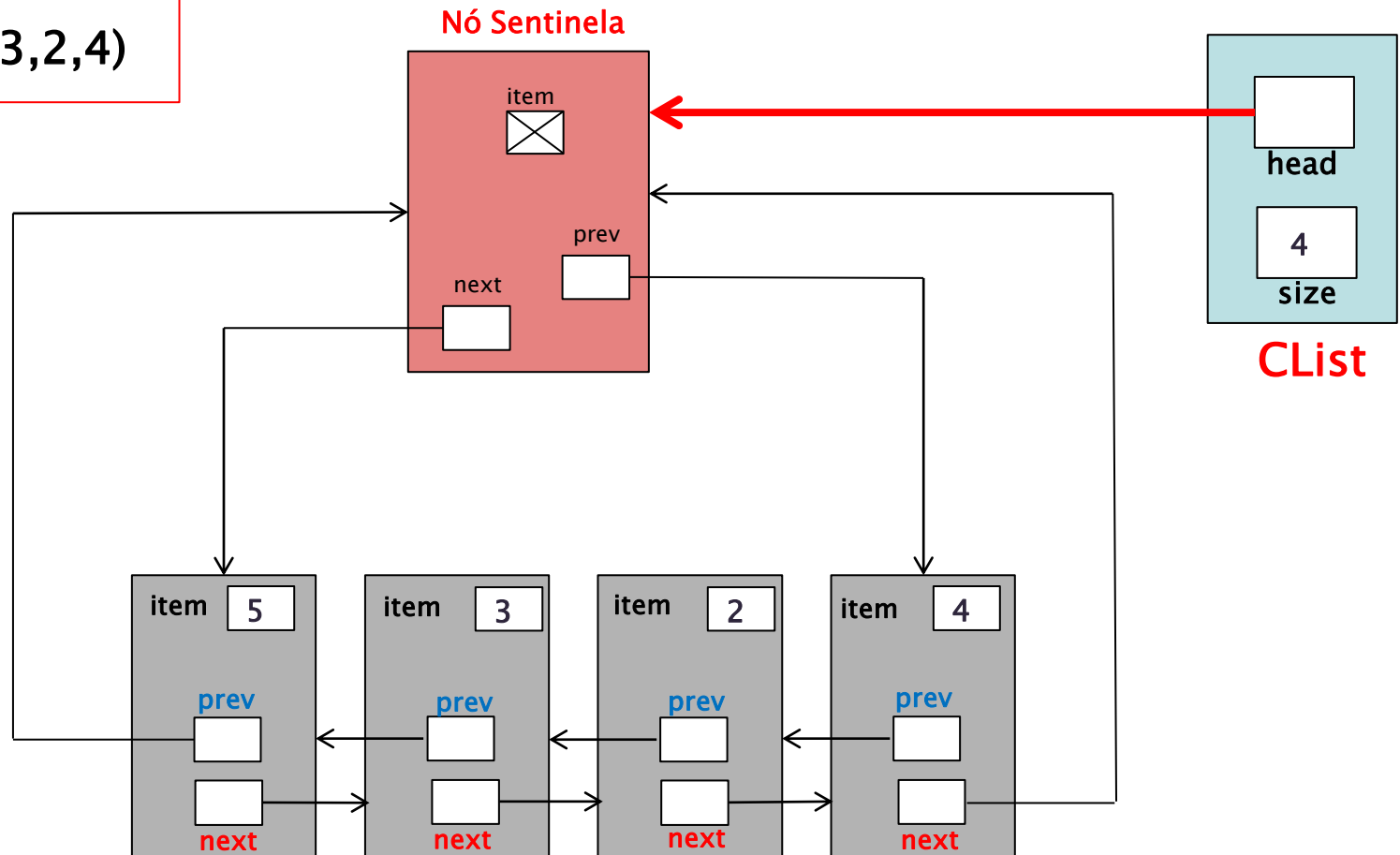
Lista: (5,3,2,4)



# Invariantes da Sentinela

▣ Para qualquer lista Clist  $c$ ,  $c.head \neq \text{null}$

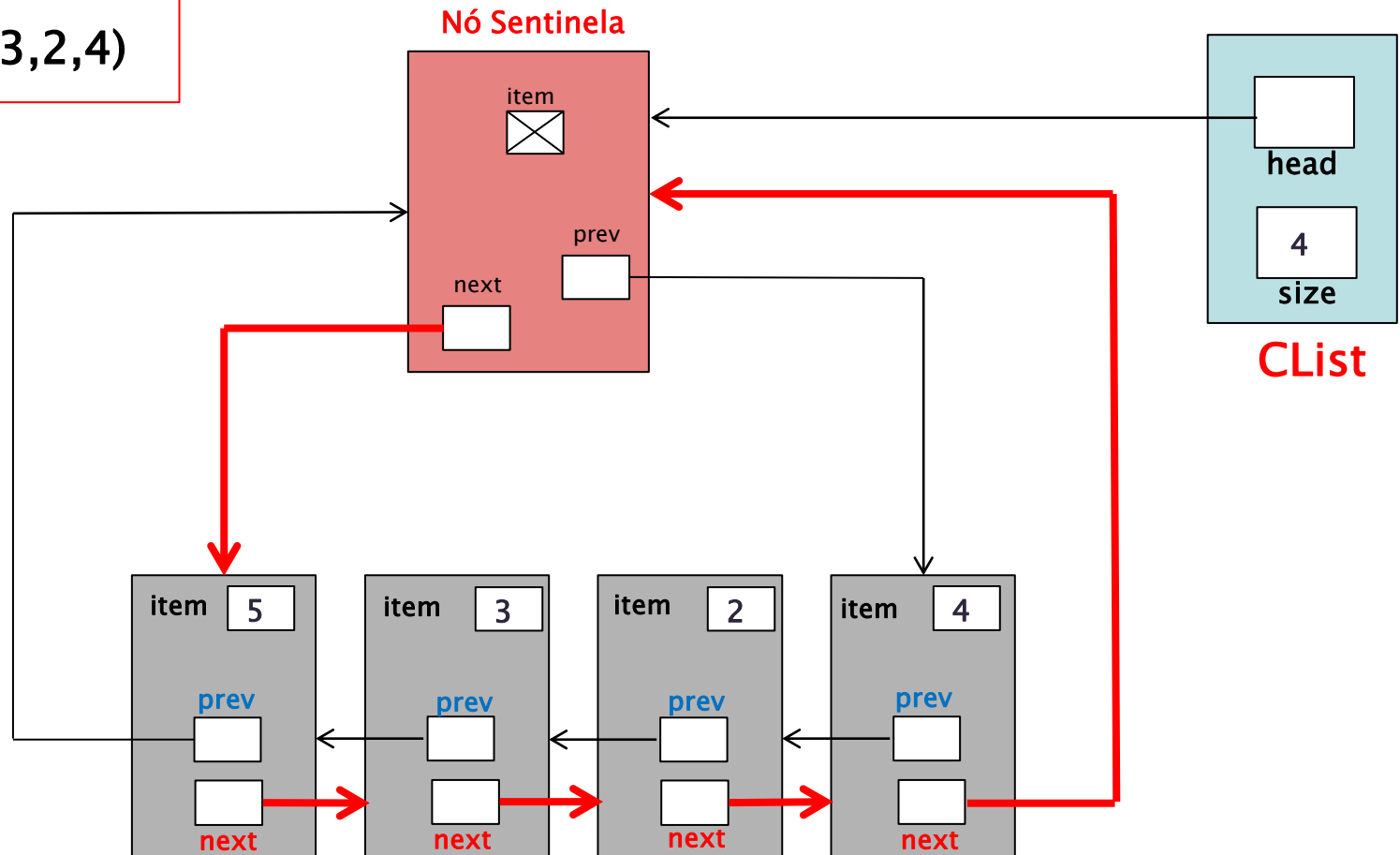
Lista: (5,3,2,4)

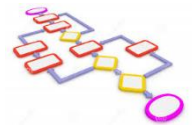


# Invariantes da Sentinela

Para qualquer nó X,  $X.next \neq null$

Lista: (5,3,2,4)

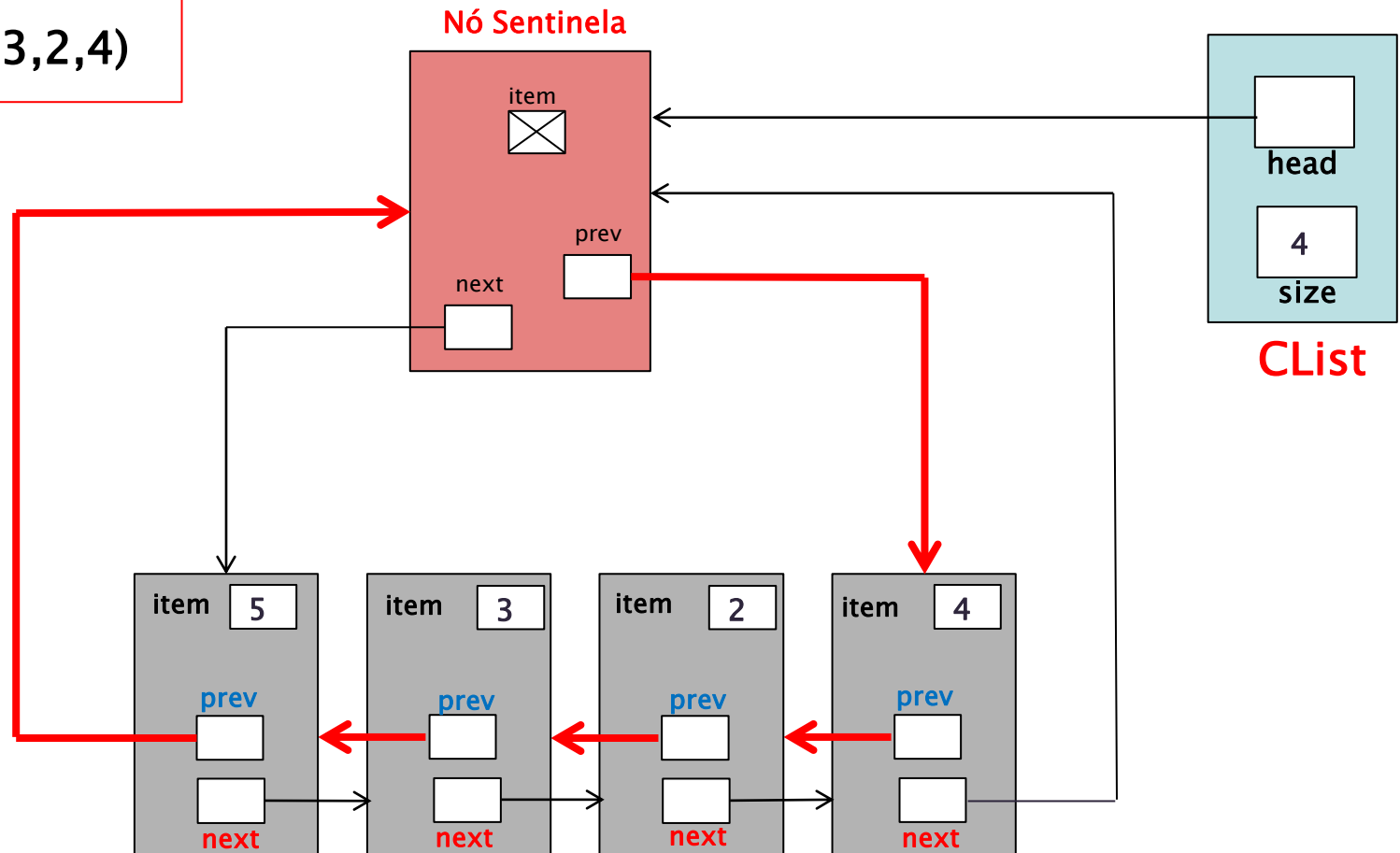


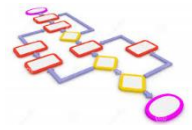


# Invariantes da Sentinela

■ Para qualquer nó X,  $X.\text{prev} \neq \text{null}$

Lista: (5,3,2,4)

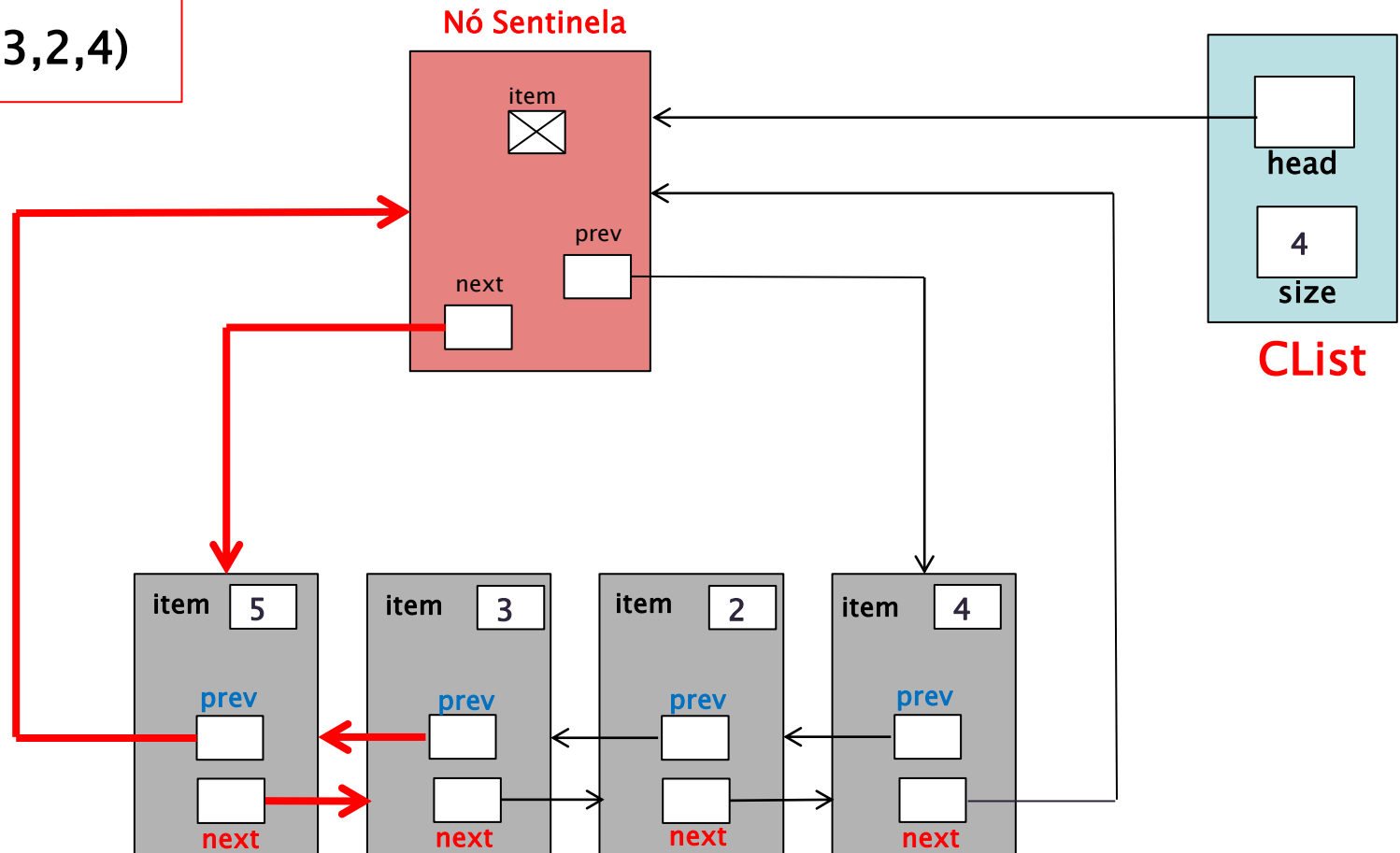




# Invariantes da Sentinela

■ Para qualquer nó  $X$ , se  $X.next == Y$ , então  $Y.prev == x$ .

Lista: (5,3,2,4)



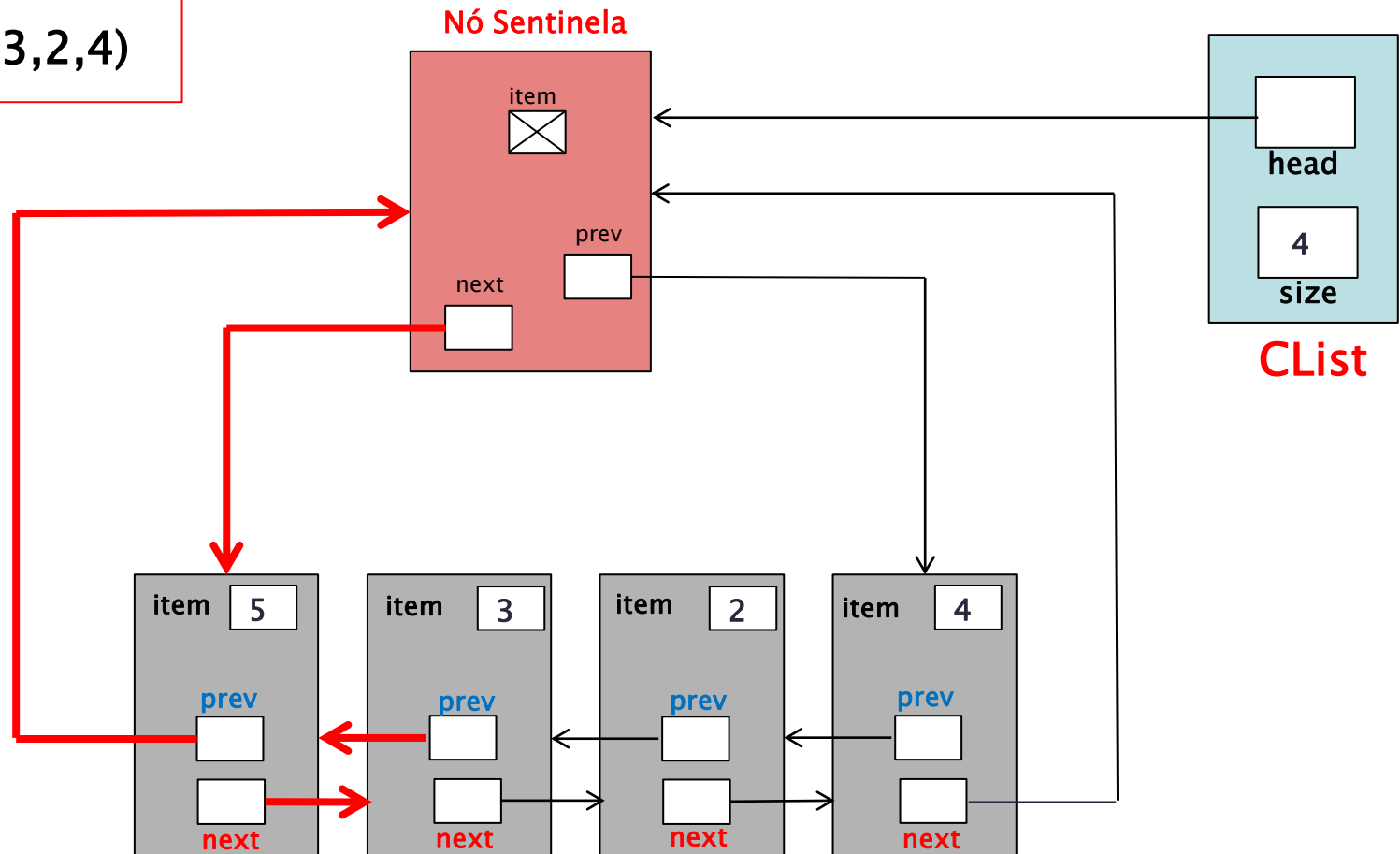


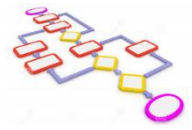


# Invariantes da Sentinela

▣ Para qualquer nó X, se  $X.\text{prev} == Y$ , então  $Y.\text{next} == x$ .

Lista: (5,3,2,4)

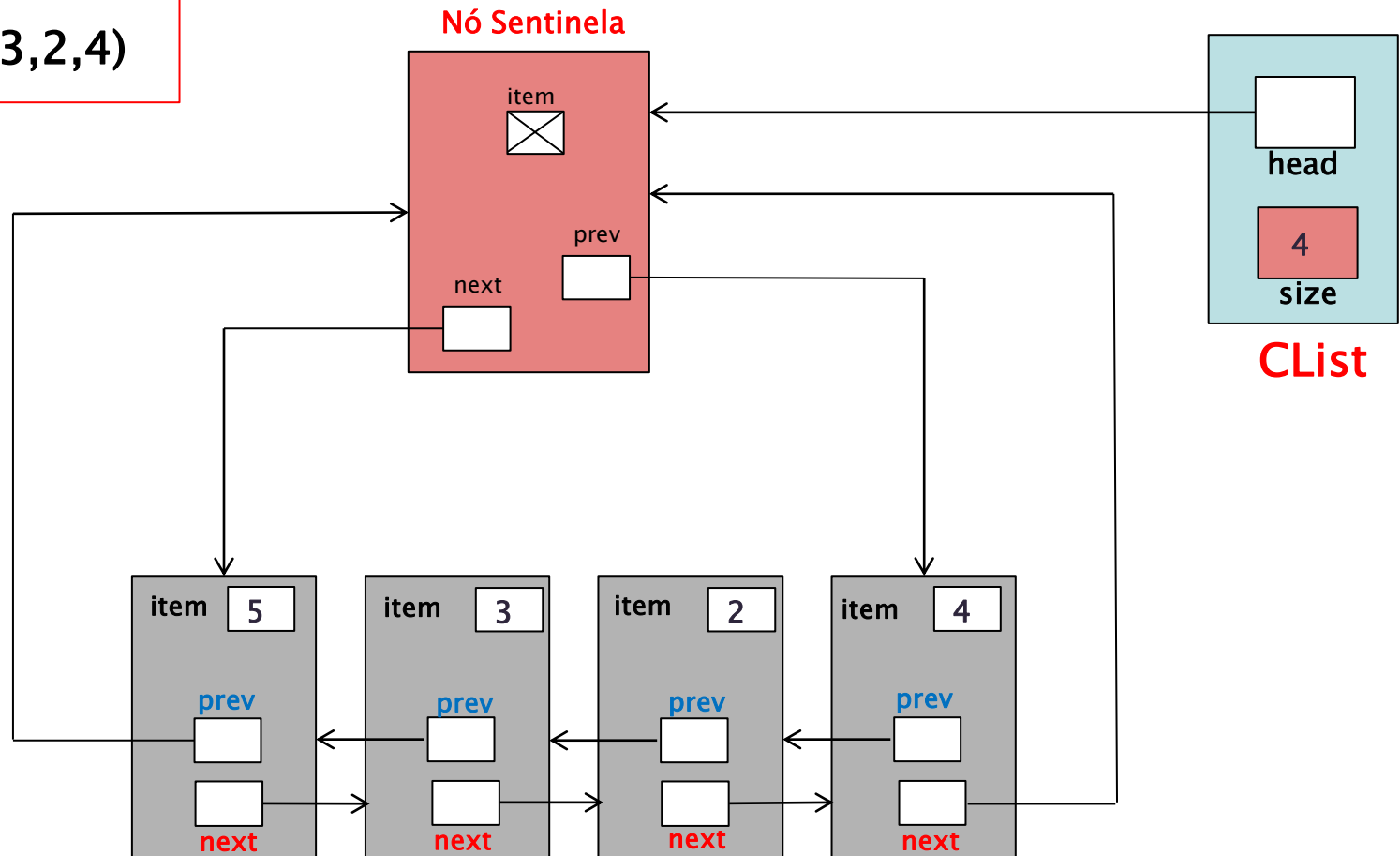




# Invariantes da Sentinela

■ A variável “size” de CList não contém o nó sentinela. (# correto)

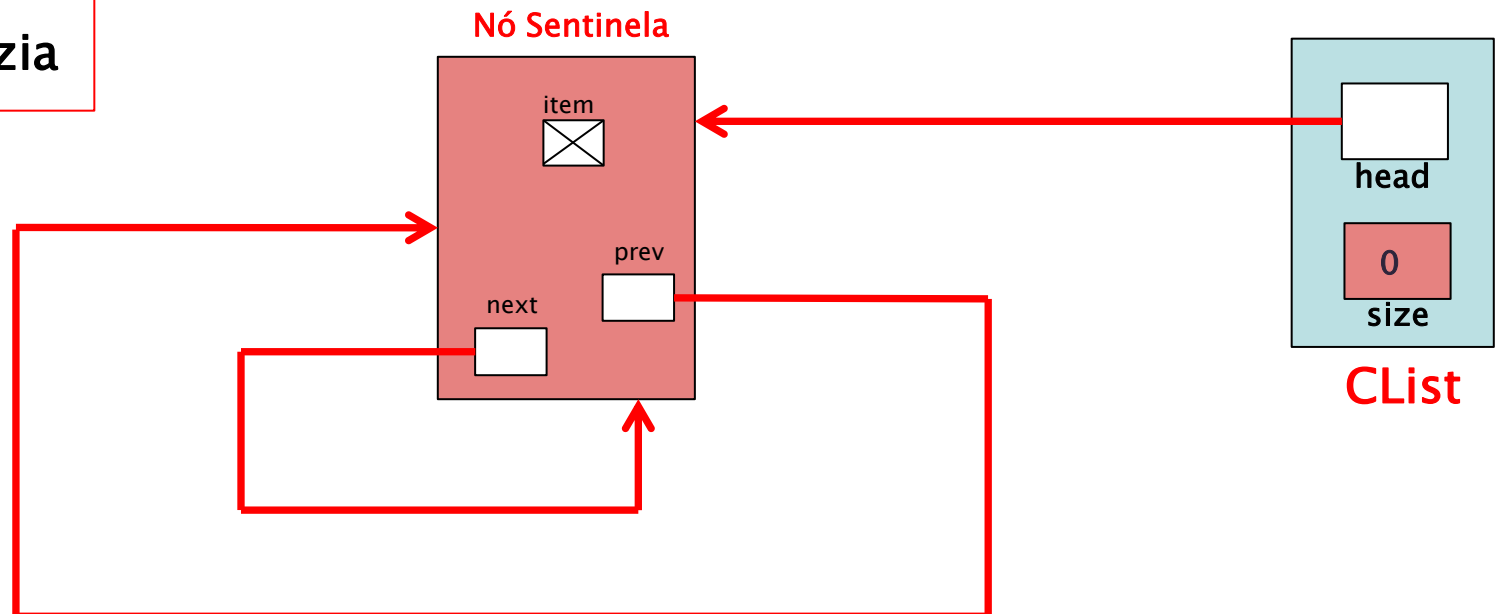
Lista: (5,3,2,4)



## Lista Vazia

Campos **next** e **prev** da sentinela apontam para a própria sentinela

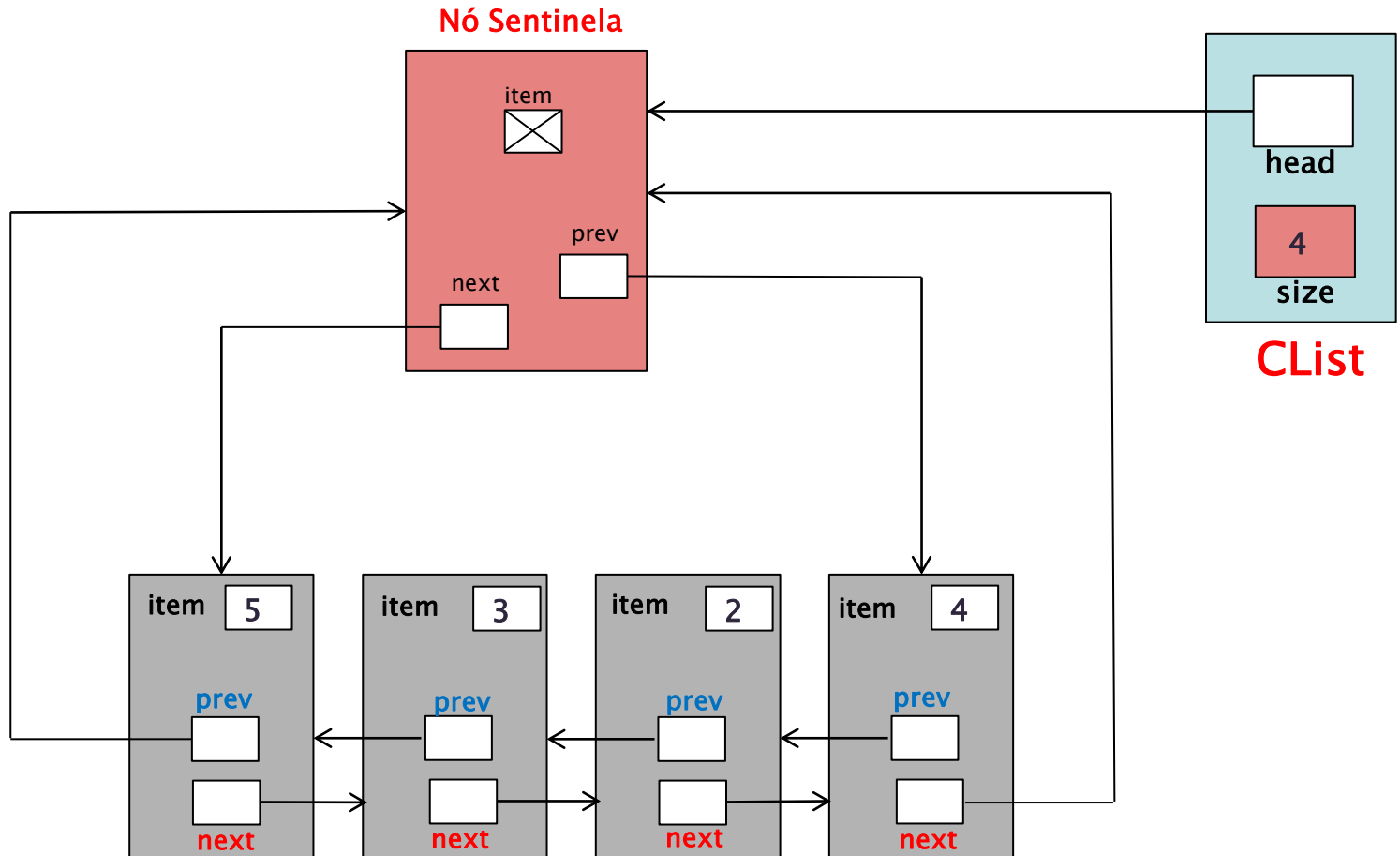
Lista: vazia





# Exemplo: Lista Circular

Lista: (5,3,2,4)



## Definição do nó da lista

```
package maua;  
  
public class DListNode {  
  
    public int item;  
  
    public DListNode next;  
  
    public DListNode prev;  
}
```



## Definindo os construtores

```
public DListNode(int item, DListNode next, DListNode prev) {  
  
    this.item = item;  
    this.next = next;  
    this.prev = prev;  
}  
  
public DListNode() {  
  
    this.item = 0;  
    this.next = null;  
    this.prev = null;  
}  
  
public DListNode(int item) {  
  
    this.item = item;  
    this.next = null;  
    this.prev = null;  
}  
}
```



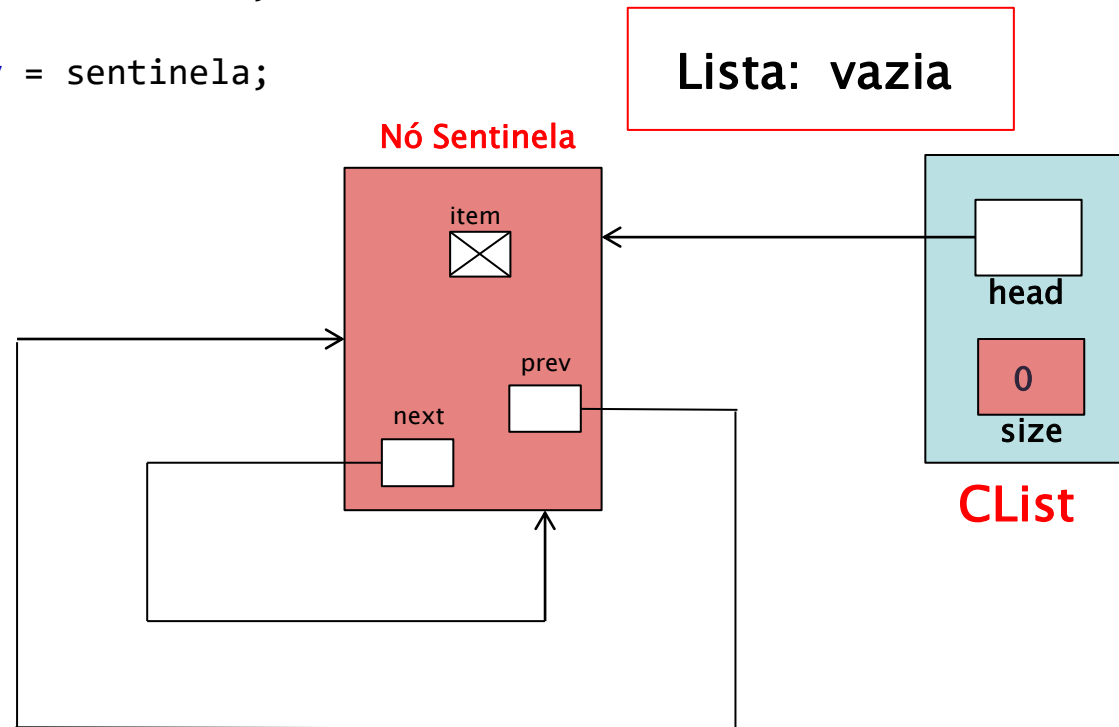
## Criando a lista Circular **CList**

```
package maua;  
  
    public class CList {  
  
        public int size;  
  
        public DListNode head;  
  
    }
```



# Construtor da classe **CList**

```
public CList() {  
  
    DListNode sentinela = new DListNode();  
  
    this.head = sentinela;  
  
    this.size = 0;  
  
    sentinela.next = sentinela;  
    sentinela.prev = sentinela;  
  
}
```





## Função `imprimeFirst()` – Pseudocódigo

```
imprimeFirst() {  
    if (size == null)  
        Print ("Lista vazia...")  
  
    else  
        Print (sentinela.next.item)  
  
}
```



## Função `imprimeFirst()`

```
public void imprimeFirst() {  
    if (this.size == 0)  
        System.out.println ("Lista vazia...");  
    else  
        System.out.println ("Primeiro item: " + this.head.next");  
}
```



## Função ImprimeLast() – Pseudocódigo

```
imprimeLast() {  
    if (size == null)  
        Print ("Lista vazia...")  
    else  
        Print (sentinela.prev.item)  
}
```



## Função Imprime\_Last()

```
public void imprimeLast() {  
    if (this.size == 0)  
        System.out.println ("Lista vazia...");  
    else  
        System.out.println("Primeiro item: " + this.head.prev);  
}
```



## Classe para teste

```
package maua;

public class Test_CList {

    public static void main(String[] args) {

        CList listaCircular = new CList();

        listaCircular.imprimeFirst();

        listaCircular.imprimeLast();

    }

}
```

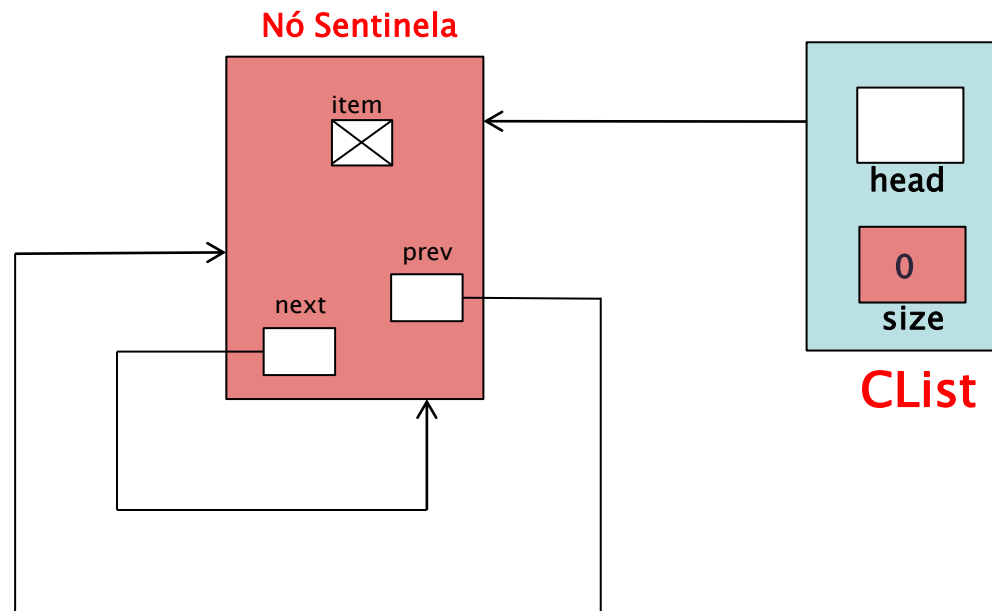


## Criando lista de nós vazia

```
CList listaCircular = new CList();  
  
listaCircular.imprimeFirst();  
  
listaCircular.imprimeLast();
```

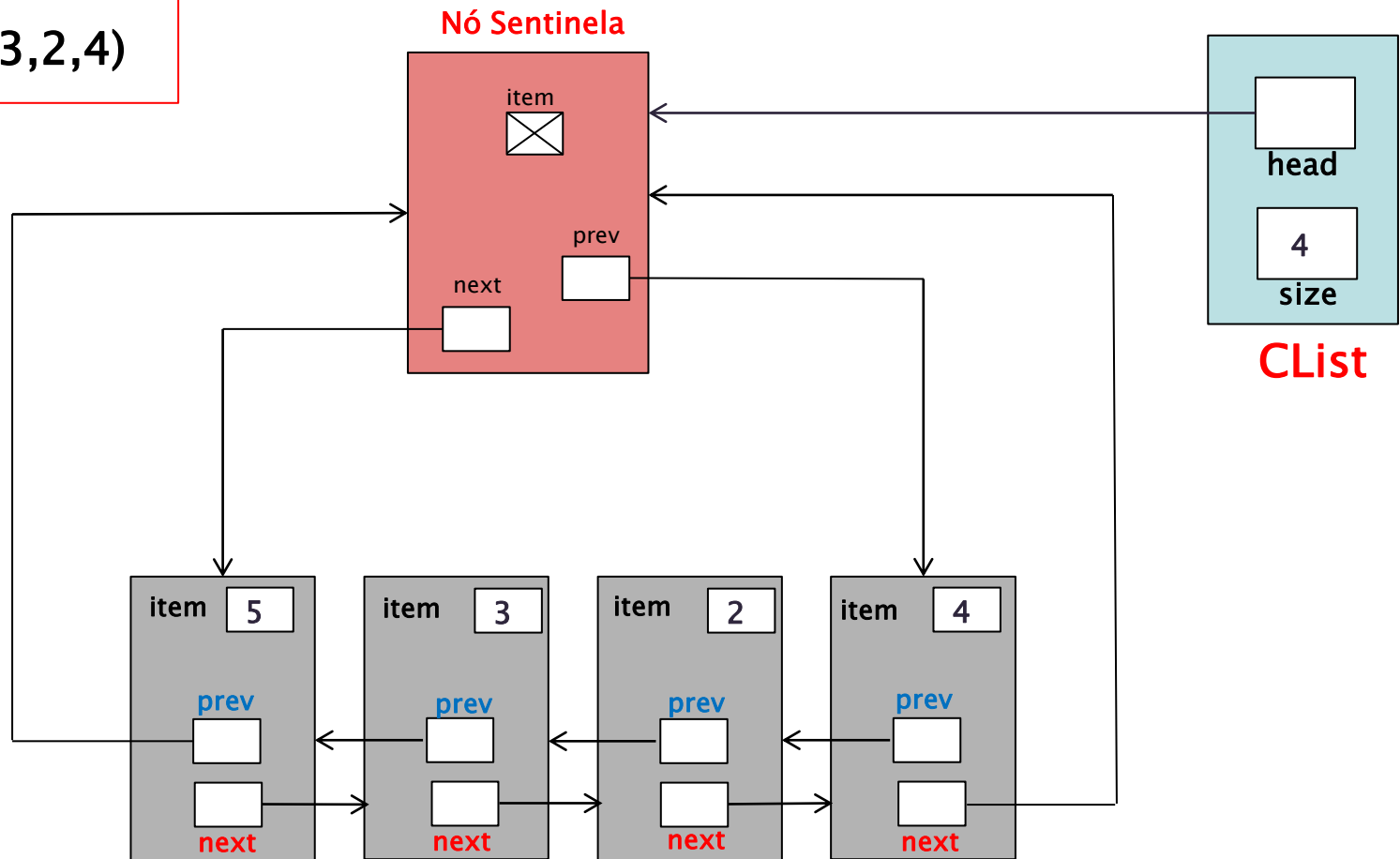
**Retorno:**      Lista de nós vazia....  
                   Lista de nós vazia....

**Lista: vazia**



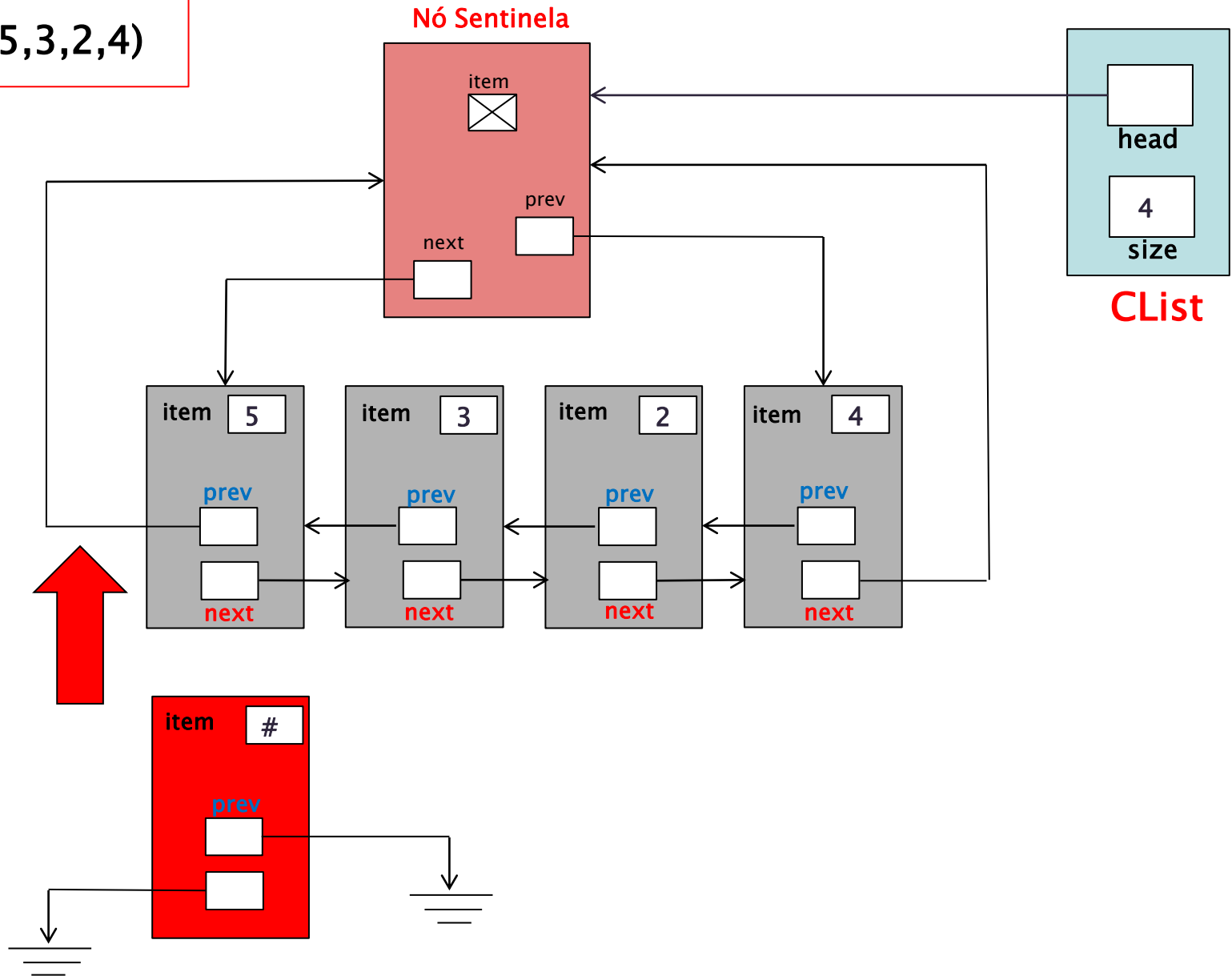
# Inserindo nós na lista de nós

Lista: (5,3,2,4)



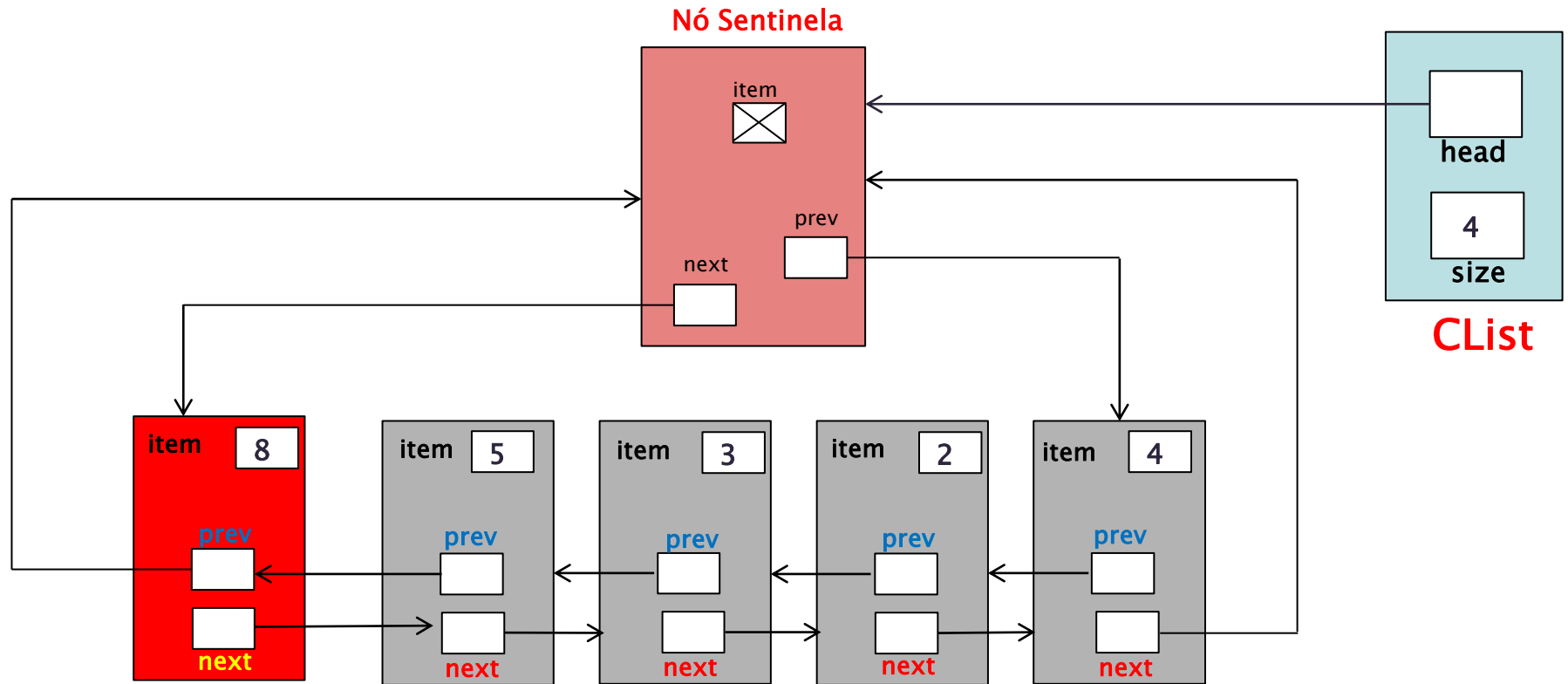
# Função insereFirst(item)

Lista: (5,3,2,4)





# Função insereFirst(item)



Lista: (8, 5,3,2,4)



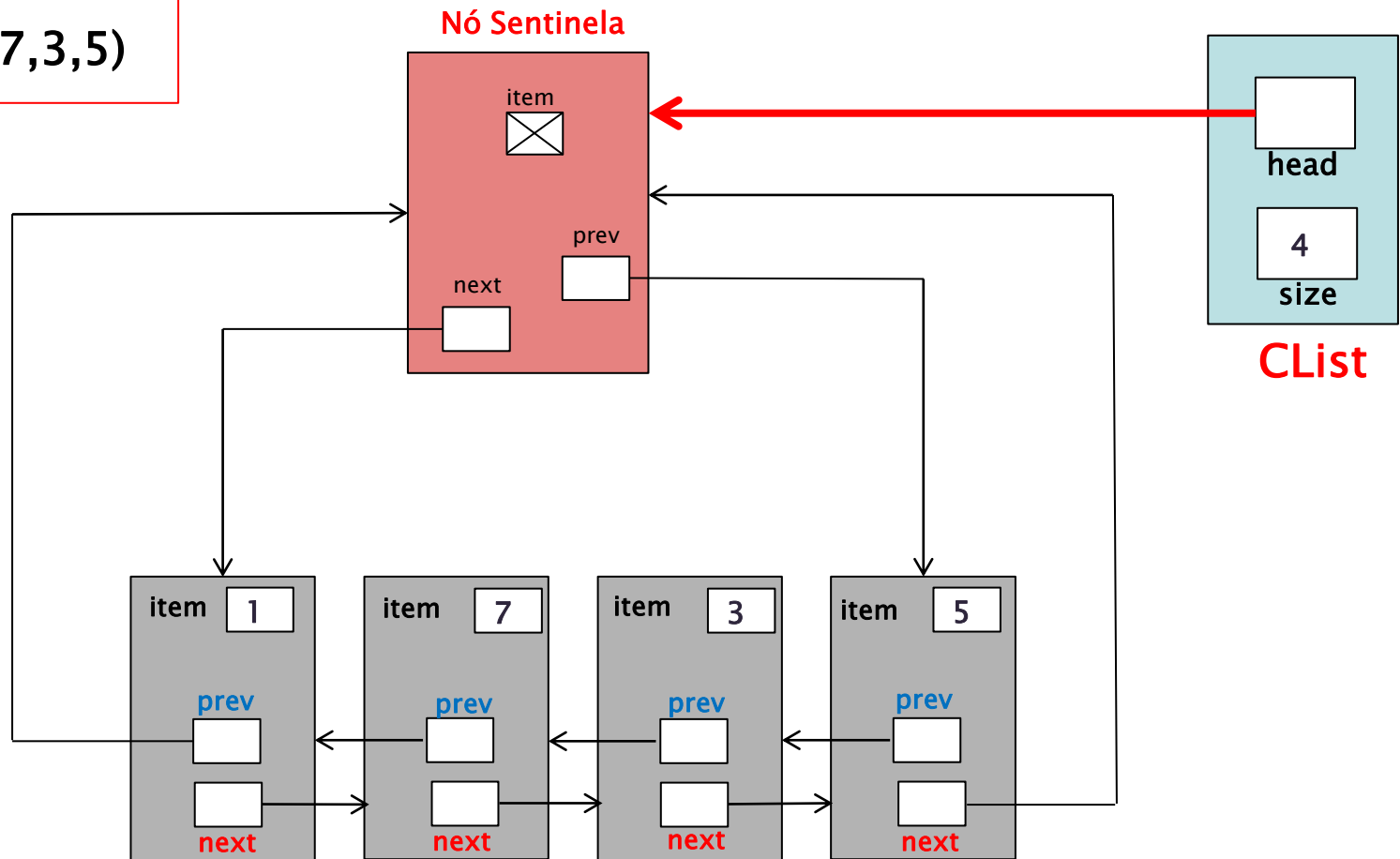
## Função insereFirst(item)

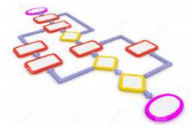
```
public void insereFirst(int item) {  
  
    DListNode novoNode = new DListNode(item);  
  
    if (this.size == 0) {  
  
        novoNode.next = this.head;  
        novoNode.prev = this.head;  
        this.head.next = novoNode;  
        this.head.prev = novoNode;  
        this.size++;  
    }  
  
    else {  
  
        novoNode.next = this.head.next;  
        novoNode.prev = this.head;  
        this.head.next.prev = novoNode;  
        this.head.next = novoNode;  
        this.size++;  
    }  
}
```



# Inserindo nós

Lista: (1,7,3,5)





## Inserindo nós

```
package maua;
```

```
public class Test_CList {
```

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

```
        CList listaCircular = new CList();
```

```
        listaCircular.imprimeFirst();
```

```
        listaCircular.imprimeLast();
```

```
        listaCircular.insereFirst(4);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();
```

```
        listaCircular.insereFirst(2);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();
```

```
        listaCircular.insereFirst(3);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();
```

```
        listaCircular.insereFirst(5);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();
```

```
        listaCircular.insereFirst(8);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();
```

```
    }
```

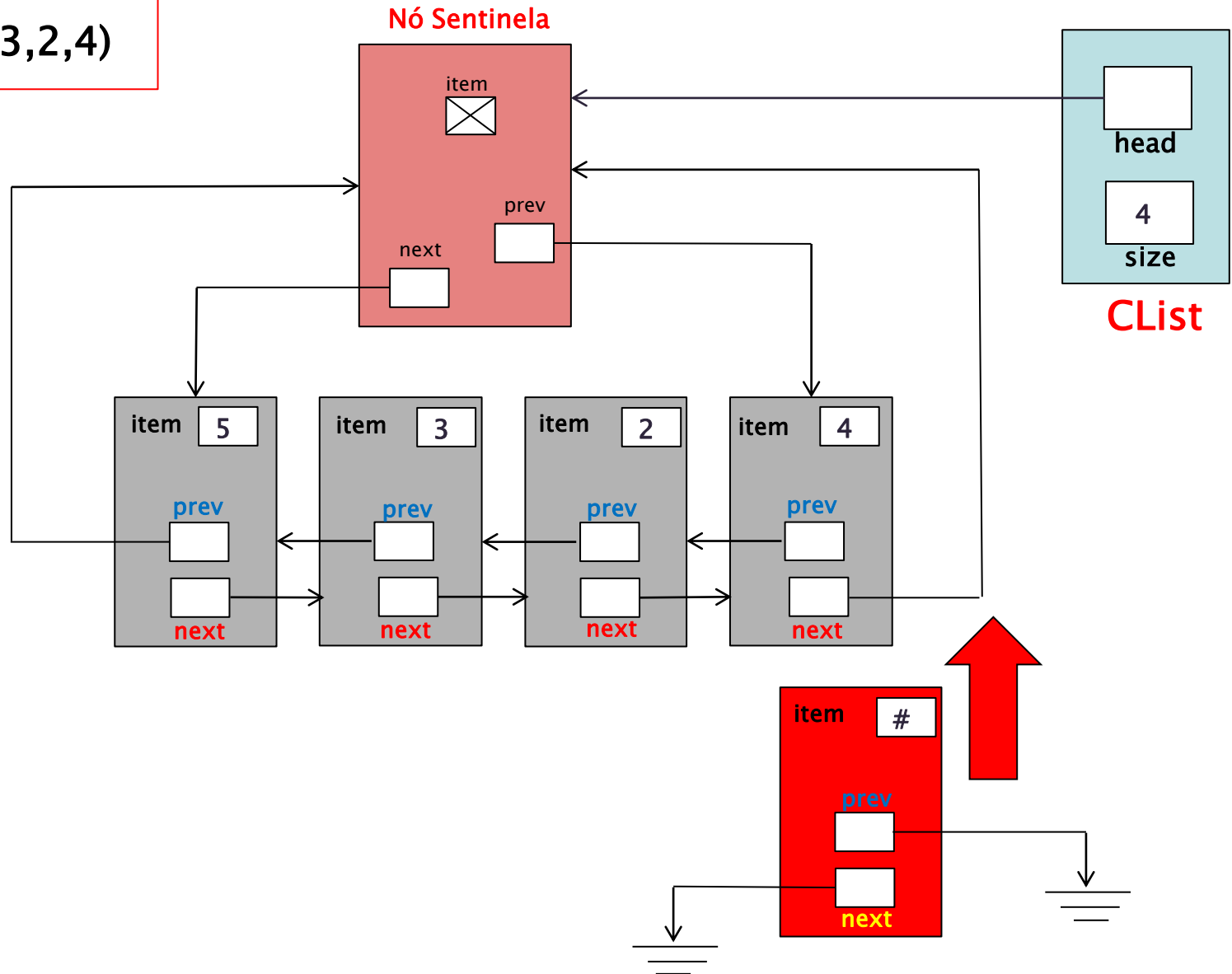
```
}
```

**Lista: (8,5,3,2,4)**

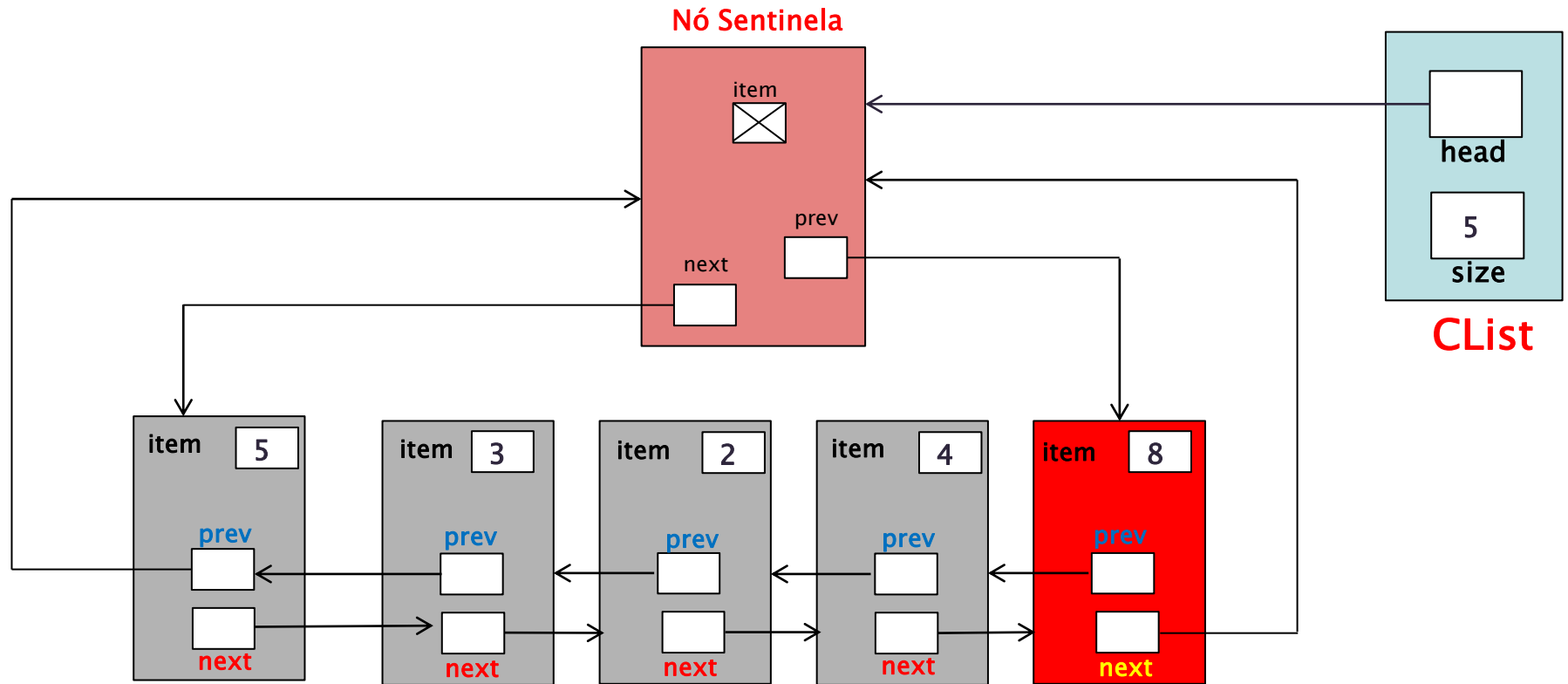


# Função insereLast(item)

Lista: (5,3,2,4)



# Função insereLast(item)



Lista: (5,3,2,4,8)



## Função `insereLast(item)`

```
public void insereLast(int item) {  
  
    DListNode novoNode = new DListNode(item);  
  
    if (this.size == 0) {  
  
        novoNode.next = this.head;  
        novoNode.prev = this.head;  
        this.head.next = novoNode;  
        this.head.prev = novoNode;  
        this.size++;  
    }  
  
    else {  
  
        novoNode.next = this.head;  
        novoNode.prev = this.head.prev;  
        this.head.prev.next = novoNode;  
        this.head.prev = novoNode;  
        this.size++;  
    }  
}
```






```
package maua;
```

## Inserindo nós

```
public class Test_CList {  
  
    public static void main(String[] args) {  
  
        CList listaCircular = new CList();  
  
        listaCircular.imprimeFirst();  
  
        listaCircular.imprimeLast();  
  
        listaCircular.inserereLast(5);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();  
  
        listaCircular.inserereLast(3);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();  
  
        listaCircular.inserereLast(2);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();  
  
        listaCircular.inserereLast(4);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();  
  
        listaCircular.inserereLast(8);  
        listaCircular.imprimeFirst();  
        listaCircular.imprimeLast();  
  
    }  
}
```



**Lista: (5,3,2,4,8)**



## Imprimindo nós da lista – Pseudocódigo

**imprimeLista()**

```
if (size == null)
    Print (“Lista vazia...”)
else
    contador = 1
    p ← head.next
    while (contador <= size)
        Print (p.item)
        p ← p.next
        contador ← contador + 1
```



## Imprimindo nós da lista

```
public void imprimeLista() {  
    if (this.size == 0)  
        System.out.println("Lista vazia...");  
    else {  
        int contador = 1;  
        DListNode p = this.head.next;  
        System.out.print("\nLista: (");  
        while (contador < this.size) {  
            System.out.print(+ p.item + ",");  
            p = p.next;  
            contador++;  
        }  
        System.out.print(p.item + ")\n");  
    }  
}
```



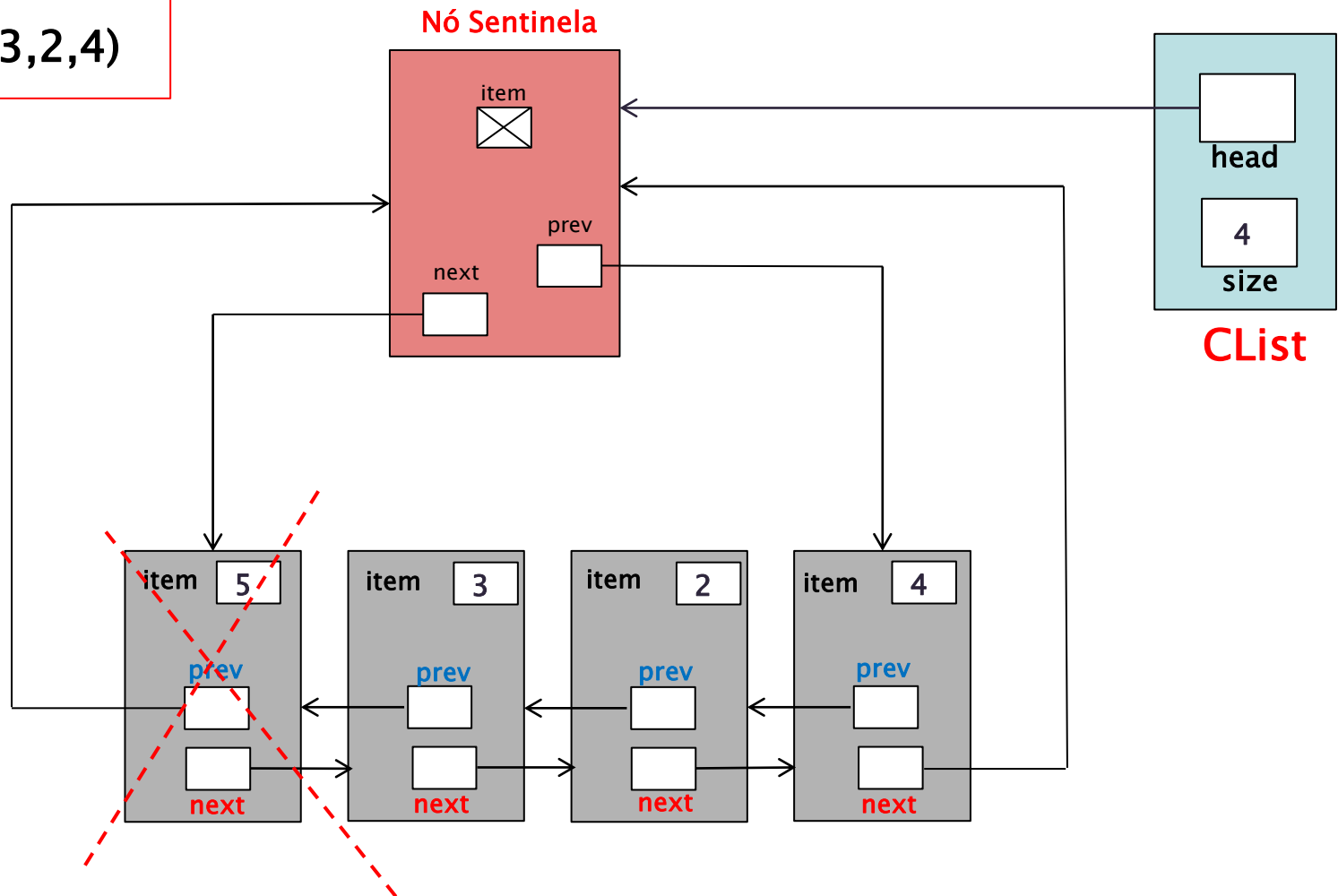
## Imprimindo nós da lista – versão 2

```
public void imprimeLista2() {  
    if (this.size == 0)  
        System.out.println("Lista vazia...");  
    else {  
        int contador = 1;  
        DListNode p = this.head.prev;  
        System.out.print("\nLista: (");  
        while (contador < this.size) {  
            System.out.print(+ p.item + ",");  
            p = p.prev;  
            contador++;  
        }  
        System.out.print(p.item + ")\n");  
    }  
}
```



# Deletando um nó da lista

Lista: (~~5~~,3,2,4)



## Função deleteFirst() – Pseudocódigo

**deleteFirst()**

```
if (size == null)
    Print ("Lista vazia...")
else
    if (size == 1)
        head.next ← head
        head.prev ← head
    else
        head.next.next.prev ← head
        head.next ← head.next.next
```



## Função deleteFirst()

```
public void deleteFirst() {  
    if (this.size == 0)  
        System.out.println("Deleção inválida... Lista Vazia...");  
    else {  
        if (this.size == 1) {  
            this.head.next = this.head;  
            this.head.prev = this.head;  
            this.size--;  
        }  
        else {  
            this.head.next.next.prev = this.head;  
            this.head.next = this.head.next.next;  
            this.size--;  
        }  
    }  
}
```



## Deletando nós da lista

```
package maua;

public class TestDList {

    public static void main(String[] args) {

        DList listaDupLigada = new DList();
        listaDupLigada.imprimeLista();

        listaDupLigada.insereInicio(4);
        listaDupLigada.insereInicio(2);
        listaDupLigada.insereInicio(3);
        listaDupLigada.insereInicio(5);

        listaDupLigada.imprimeLista();

        listaDupLigada.deleteFirst();
        listaDupLigada.imprimeLista();

        listaDupLigada.deleteFirst();
        listaDupLigada.imprimeLista();

        listaDupLigada.deleteFirst();
        listaDupLigada.imprimeLista();

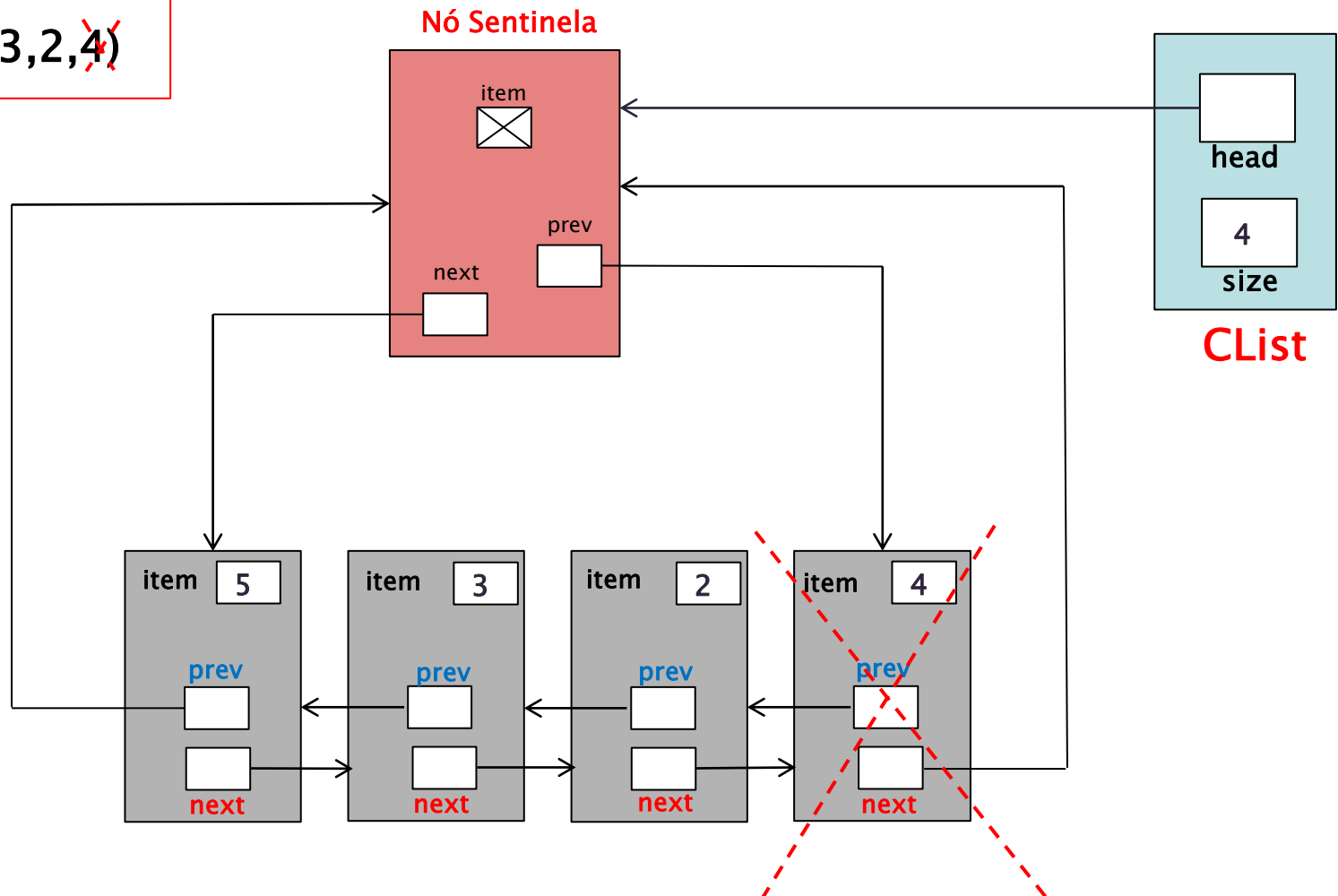
        listaDupLigada.deleteFirst();
        listaDupLigada.imprimeLista();

    }
}
```



# Deletando um nó da lista

Lista: (5,3,2,~~4~~)





## Função deleteLast() – Pseudocódigo

**deleteLast()**

```
if (size == null)
    Print (“Lista vazia...”)
else
    if (size == 1)
        head.prev ← head
        head.prev ← head
    else
        head.prev.next ← head
        head.prev ← head.prev.prev
```



## Função deleteLast()

```
public void deleteLast() {  
    if (this.size == 0)  
        System.out.println("Deleção inválida... Lista Vazia...");  
    else {  
        if (this.size == 1) {  
            this.head.next = this.head;  
            this.head.prev = this.head;  
            this.size--;  
        }  
        else {  
            this.head.prev.prev.next = this.head ;  
            this.head.prev = this.head.prev.prev ;  
            this.size--;  
        }  
    }  
}
```



## Deletando nós da lista

```
package maua;

public class Test_CList {

    public static void main(String[] args) {

        CList listaCircular = new CList();
        listaCircular.inserereLast(5);
        listaCircular.inserereLast(3);
        listaCircular.inserereLast(2);
        listaCircular.inserereLast(4);
        listaCircular.inserereLast(8);

        listaCircular.imprimeLista();

        listaCircular.deleteLast();
        listaCircular.imprimeLista();
        listaCircular.deleteLast();
        listaCircular.imprimeLista();
        listaCircular.deleteLast();
        listaCircular.imprimeLista();
        listaCircular.deleteLast();
        listaCircular.imprimeLista();
        listaCircular.deleteLast();
        listaCircular.imprimeLista();
        listaCircular.deleteLast();
        listaCircular.imprimeLista();

    }
}
```

