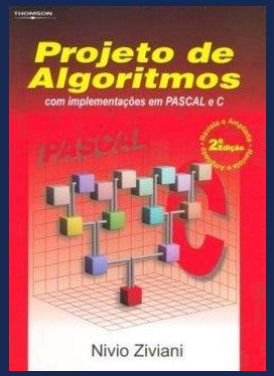
QualitSys

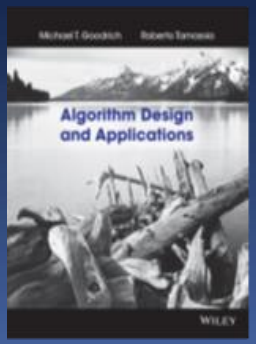
Unidade 8 - Análise de Algoritmos com Estruturas Lineares

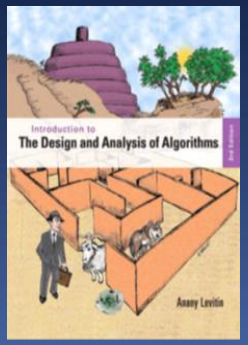
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

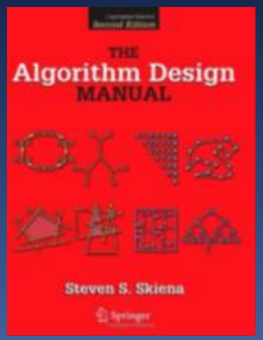
Bibliografia

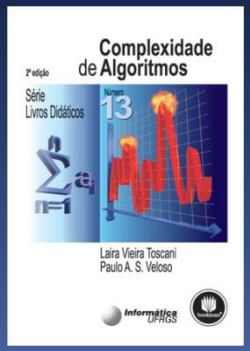
- **Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002**
- **Projeto de Algoritmos – Nivio Ziviani – Pioneira Informática - 1993**
- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS
- Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press – 2015
- Notas de Aulas – Prof. Dr. Marcelo Henriques de Carvalho – UFMS – FACOM

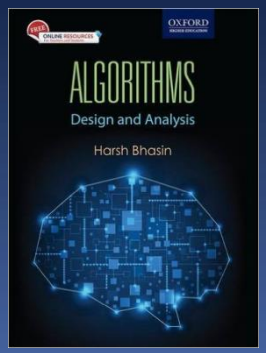












Imprimindo os elementos do array

```
package maua;
public class ArrayPrint {

        public static void main(String[] args) {
                Integer[] x = new Integer[50];

                for (int i=0 ; i < x.length ; i++)
                        x[i]=i;

                imprimeArray(x);
        }
        public static void imprimeArray (Integer[] array )  {

                Integer n = array.length;

                for (int i = 0 ; i < n ; i++)
                        System.out.println(array[i]);

        }

}
```

```
public static void imprimeArray (Integer[] array )  {

        Integer n = array.length;

        for (int i = 0 ; i < n ; i++)
                System.out.println(array[i]);

}
```

Qual o ordem de Complexidade da Função imprimeArray?

Qual o ordem de Complexidade da Função imprimeArray

```
public static void imprimeArray (Integer[] array )  {

                Integer n = array.length;

                for (int i = 0 ; i < n ; i++)
                        System.out.println(array[i]);

        }
```

Ordem de Complexidade: **O(n)**

Busca Linear

```
package maua;

public class BuscaLinear {

        public static void main(String[] args) {
                Integer[] x = new Integer[50];

                for (int i=0 ; i < x.length ; i++)
                        x[i]=i+1;

                Integer argumento = 50;

                Integer n = (BuscaLinearArray(x,argumento) ) ;

                if ( n == -1)
                        System.out.println("Valor não existente no
                                array...");
                else
                        System.out.println("Valor " + argumento + "
                                encontrado na posição: " + n );
}
```

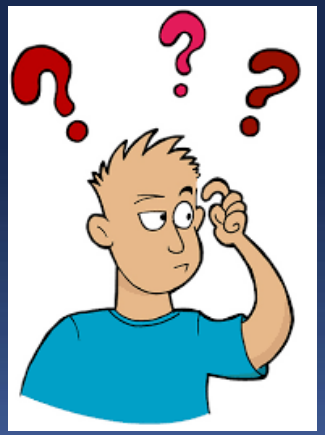
Busca Linear

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {

        Integer n = array.length;

        for (int i = 0 ; i < n ; i++)
                if (array[i] == argumento)
                        return i;
        return -1;

    }

}
```

Busca Linear

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {

        Integer n = array.length;

        for (int i = 0 ; i < n ; i++)
                if (array[i] == argumento)
                        return i;
        return -1;

   }

}
```

Qual o ordem de Complexidade da Função BuscaLinearArray?



Ordem de Complexidade da Função BuscaLinearArray

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {

        Integer n = array.length;

        for (int i = 0 ; i < n ; i++)
                if (array[i] == argumento)
                        return i;
        return -1;

   }

}
```

- Se o elemento estiver presente logo na primeira posição do array, então a ordem de complexidade seria **O(1)**;

- No caso extremo, quando o elemento não estiver presente no array, todos os n elementos do array deverão ser visitados, o que corresponde a complexidade **O(n)**.

Revertendo a ordem dos elementos de um array

```
package maua;

import java.util.Arrays;

        public class RevertArray {

                public static void main(String[] args) {
                        Integer[] x = new Integer[9];

                        for (int i=0 ; i < x.length ; i++)
                                x[i]=i+1;

                        System.out.println(Arrays.toString(x));

                        System.out.println("\n");

                        System.out.println(Arrays.toString(Reorder(x)));

}
```

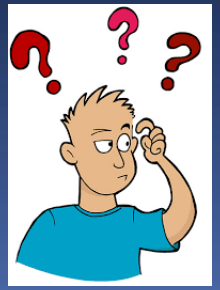
Revertendo a ordem dos elementos de um array

```
public static Integer[]  Reorder (Integer[] array )  {

    Integer n = array.length;

    Integer j = (n/2), temp;

    for (int i = 0; i < j ; i++) {
            temp = array[i];
            array[i] = array[n-1-i];
            array[n-1-i] = temp;
    }
    return array;
  }
}
```

Revertendo a ordem dos elementos de um array

```
public static Integer[]  Reorder (Integer[] array )  {

      Integer n = array.length;

      Integer j = (n/2), temp;

      for (int i = 0; i < j ; i++) {
            temp = array[i];
            array[i] = array[n-1-i];
            array[n-1-i] = temp;
      }
      return array;
   }
}
```

Qual o ordem de Complexidade da Função Reorder?



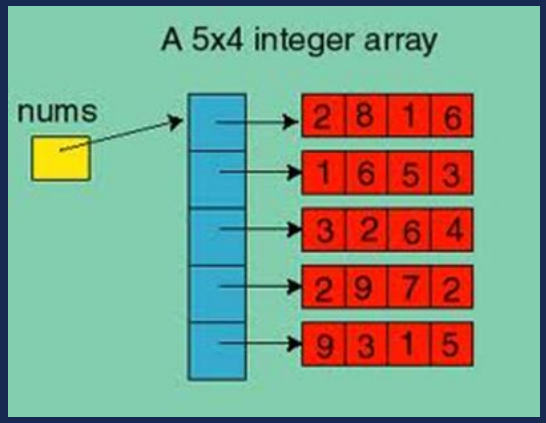
Ordem de Complexidade da Função Reorder

```
public static Integer[]  Reorder (Integer[] array )  {

     Integer n = array.length;

     Integer j = (n/2), temp;

     for (int i = 0; i < j ; i++) {
          temp = array[i];
          array[i] = array[n-1-i];
          array[n-1-i] = temp;
     }
     return array;
   }
}
```

- Todas as **operações** internas loop são executadas **n/2** vezes;

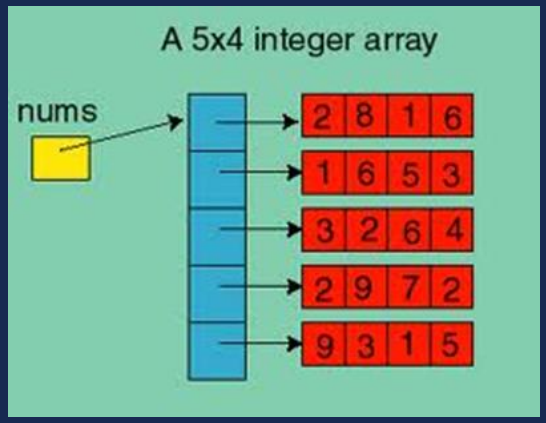
- Assim, a ordem de complexidade da função Reorder é **O(n)**.

Carga de Array de arrays




```
package maua;

public class ArrayArray {

        public static void main(String[] args) {

                int[][] nums  = new int[5][4];

                for (int r=0; r < nums.length; r++) {

                        for (int c=0; c < nums[r].length; c++) {
                                nums [r][c] = (int) (Math.random() * 10);
                                System.out.print(" " + nums[r][c]);
                        }
                        System.out.println("");
                }
        }
}
```

Carga de Array de arrays




```
package maua;

public class ArrayArray {

        public static void main(String[] args) {

                int[][] nums  = new int[5][4];

                for (int r=0; r < nums.length; r++) {

                        for (int c=0; c < nums[r].length; c++) {
                                nums [r][c] = (int) (Math.random() * 10);
                                System.out.print(" " + nums[r][c]);
                        }
                        System.out.println("");
                }
        }
}
```

Ordem de complexidade do algoritmo => $O(n^2)$.

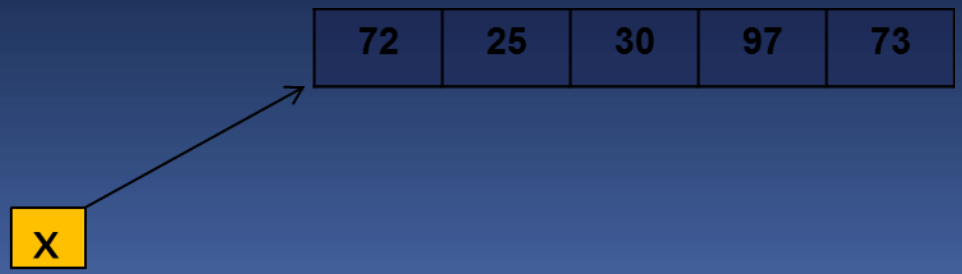
Estruturas do Tipo Lista

- Uma lista ou sequência é uma estrutura de dados abstrata que implementa uma **coleção ordenada de valores**, onde o mesmo valor pode ocorrer mais de uma vez.

- Uma lista é um **tipo abstrato de dados** (especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados) .

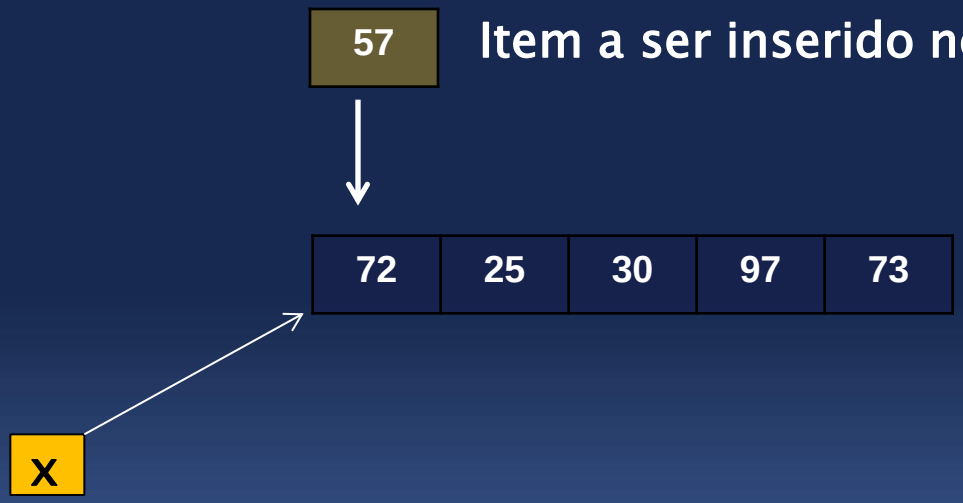
Lista de 5 valores aleatórios entre 0 e 100

```
package maua;

public class Lista_Aleatoria {

        public static void main(String[] args) {

                int[] x = new int[5];
                for (int i=0; i<x.length ; i++) {
                        x[i] = (int)(100.0 * Math.random());
                        System.out.print(" " + x[i]);
                }
        }

}
```

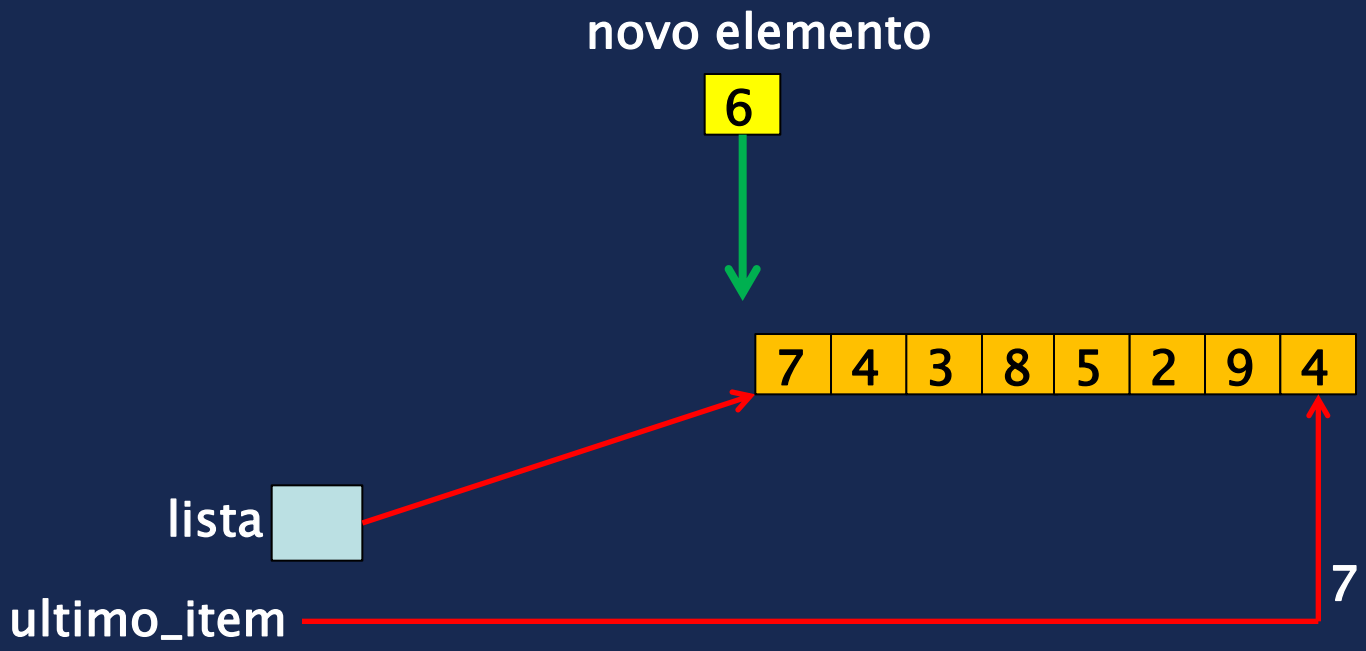


Mas, há alguns inconvenientes em se implementar listas com arrays...

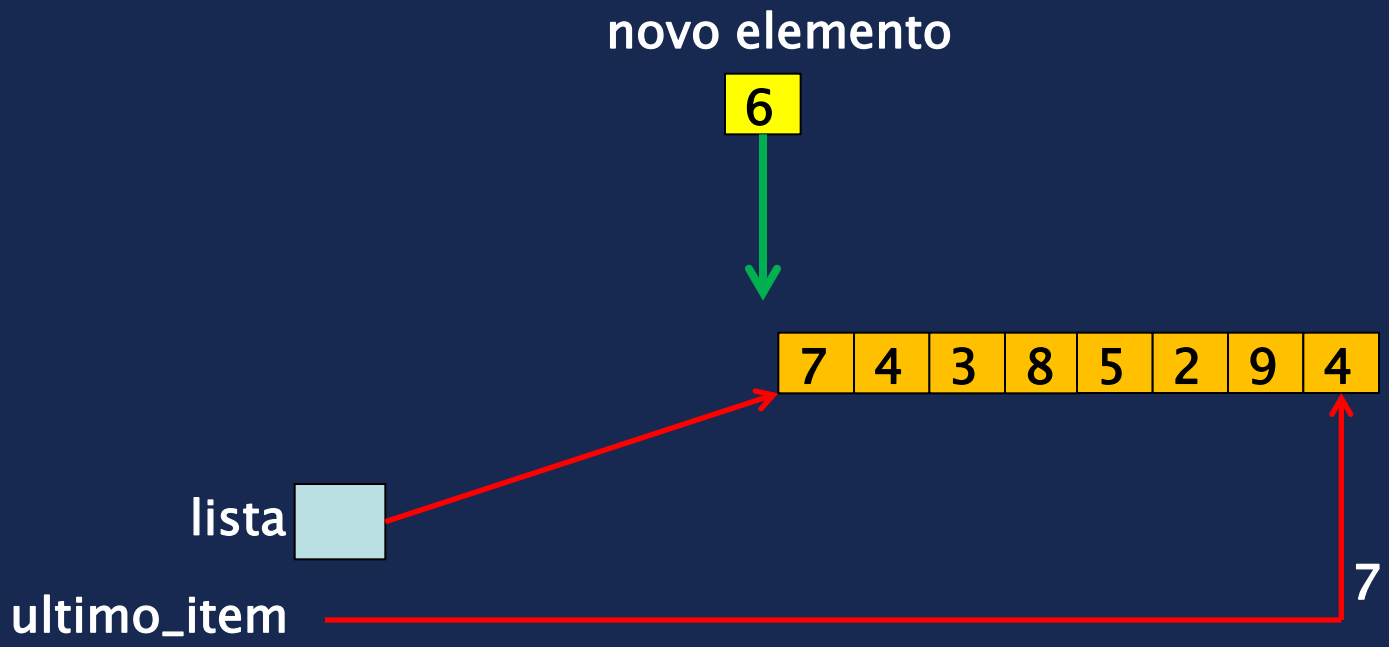
Considere a necessidade de se inserir um item no início ou metade da lista ...



Inserção no início da lista

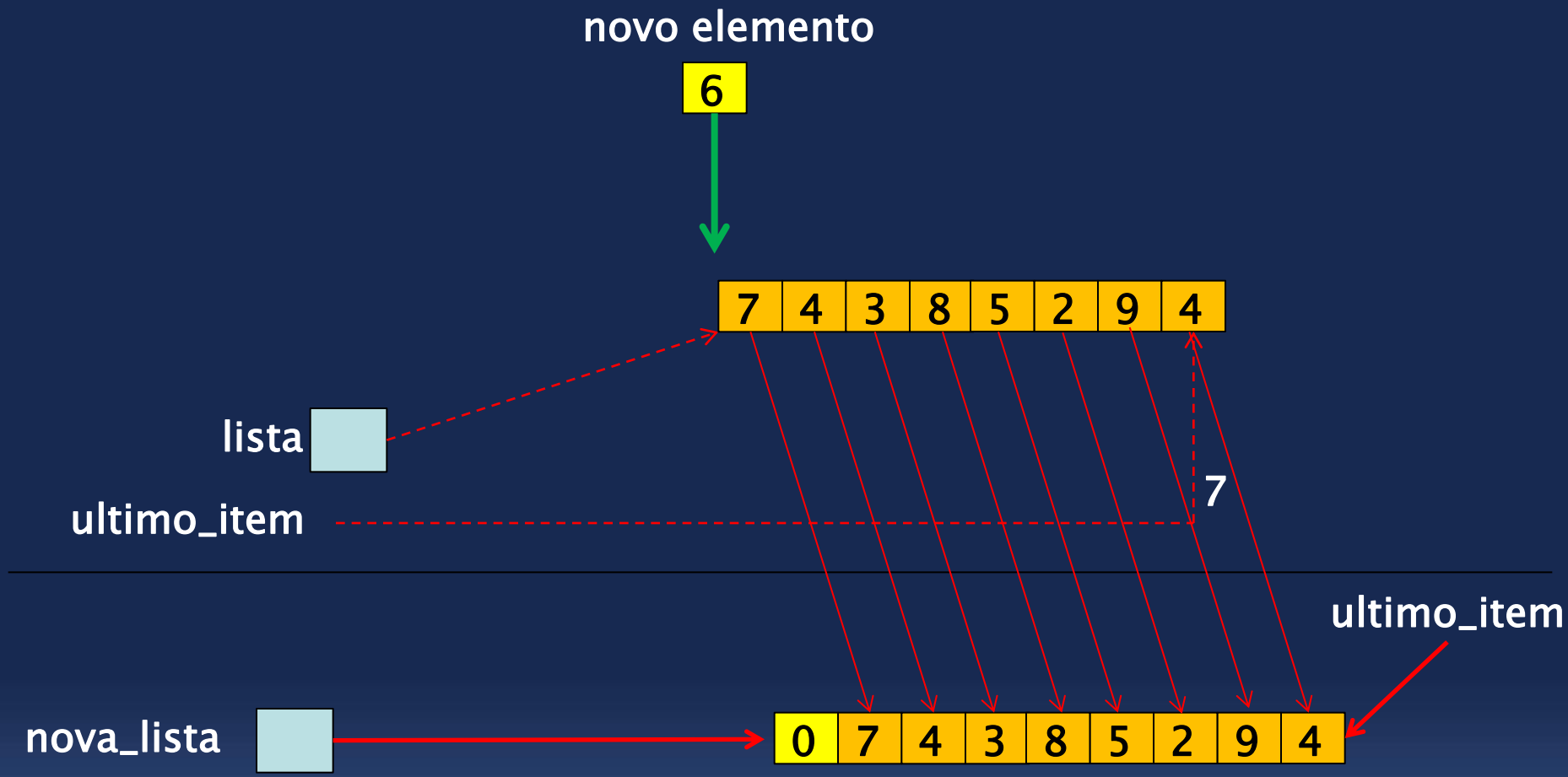


Inserção no início da lista



Primeiro: cria-se um novo array com um elemento a mais...

Inserção no início da lista



Segundo: copia-se os elementos da lista antiga para a lista nova

Implementação

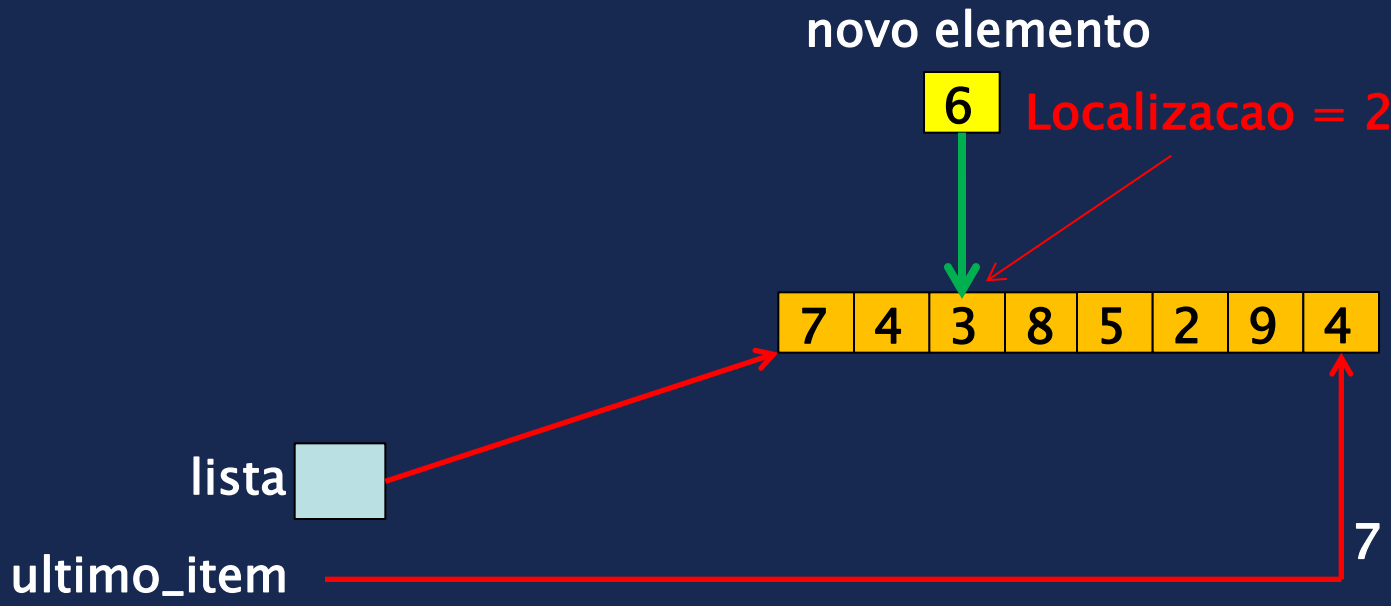
```
public void insereItem_Inicio(int novo_item) {

        int[] nova_lista = new int[lista.length+1];

        for (int i=0; i<lista.length; i++)
                nova_lista[i+1]=lista[i];

        nova_lista[0] = novo_item;
        lista = nova_lista;
        ultimo_item = lista.length-1;

}
```

Ordem de complexidade: O(**n**).

Como inserir um elemento numa posição qualquer da lista ?

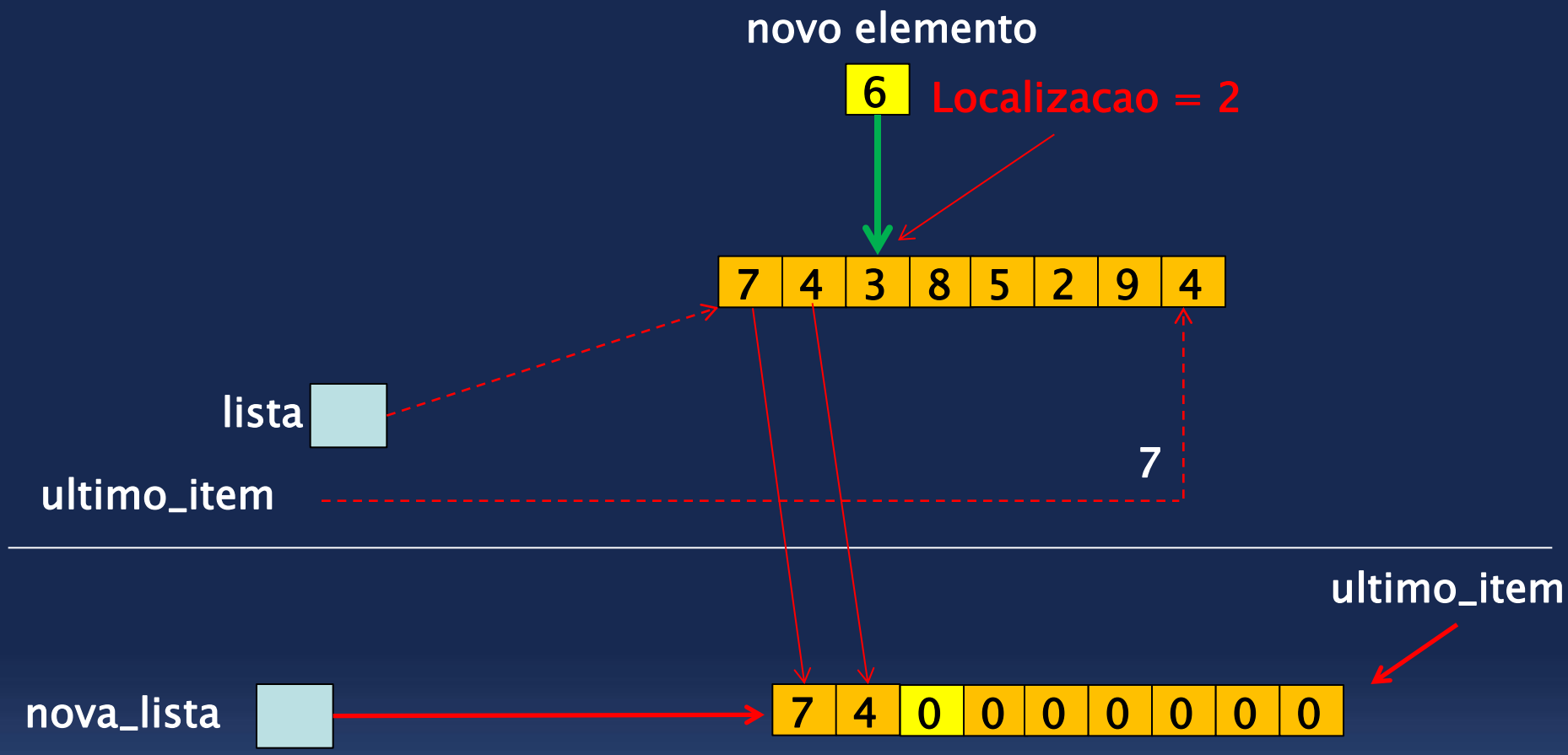
Inserção numa posição qualquer da lista

Inserção numa posição qualquer da lista



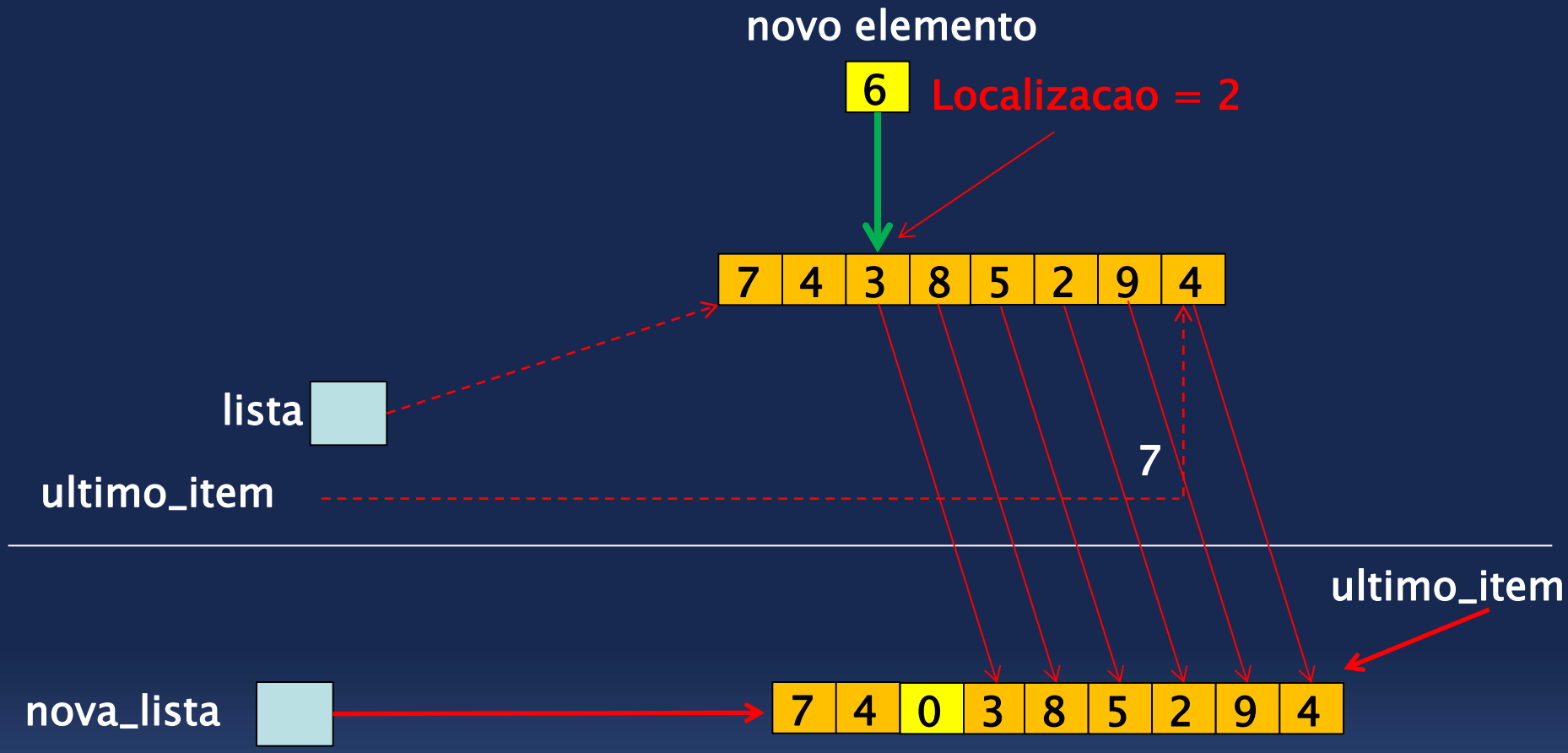
Primeiro: cria-se um novo array com um elemento a mais...

Inserção numa posição qualquer da lista



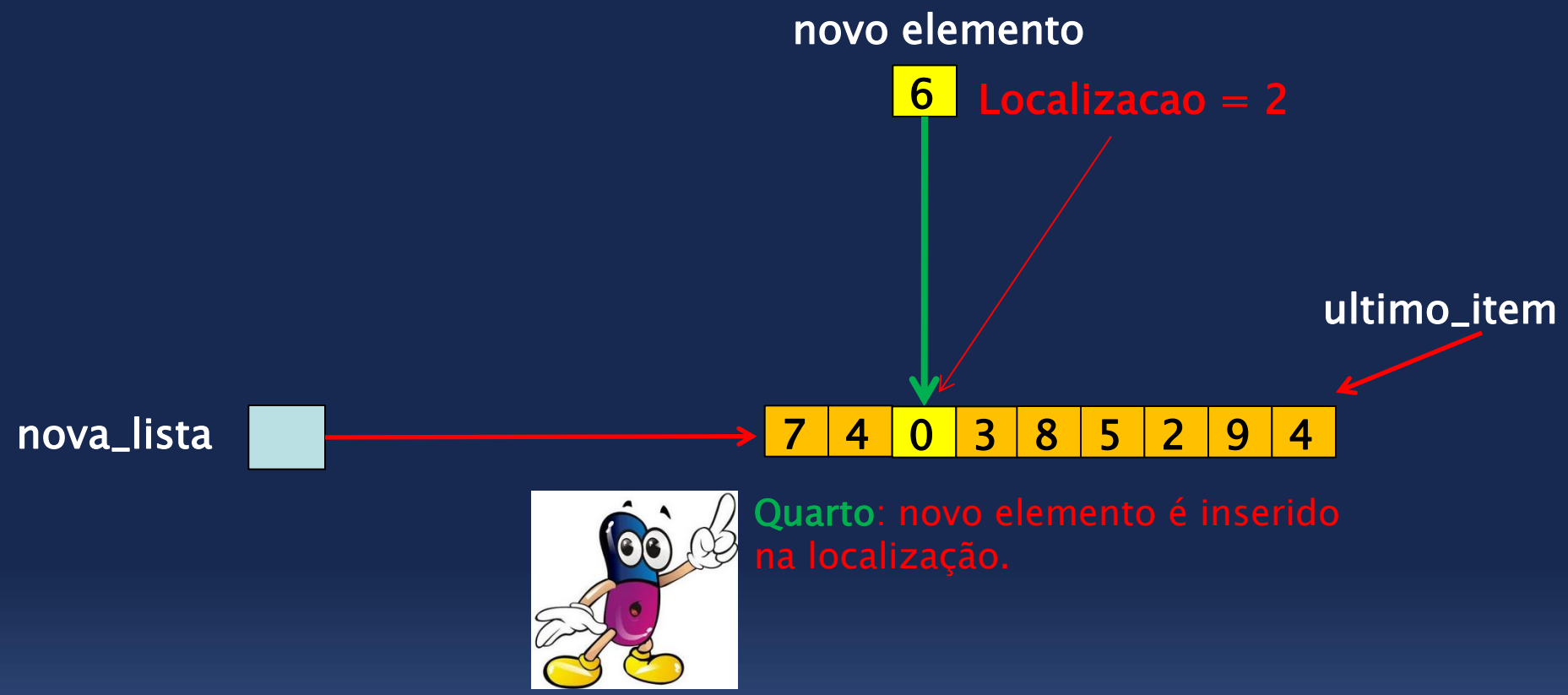
Segundo: copia-se os elementos anteriores à localização da lista antiga para a lista nova

Inserção numa posição qualquer da lista



Terceiro: copia-se os elementos posteriores à localização da lista antiga para a lista nova

Inserção numa posição qualquer da lista



Quarto: novo elemento é inserido na localização.

Implementação

```
public void insereItem(int novo_item, int localizacao) {

        if (localizacao < 0 || localizacao > this.ultimo_item)
                System.out.println("*** ERRO: localização inválida...");
        else {
                int[] nova_lista = new int[lista.length+1];

                for (int i =0; i < localizacao ; i++ )
                nova_lista[i] = lista[i];

                for(int i=localizacao; i<lista.length; i++)
                        nova_lista[i+1] = lista[i];

                nova_lista[localizacao] = novo_item;
                lista = nova_lista;
                ultimo_item = lista.length-1;
        }
}
```

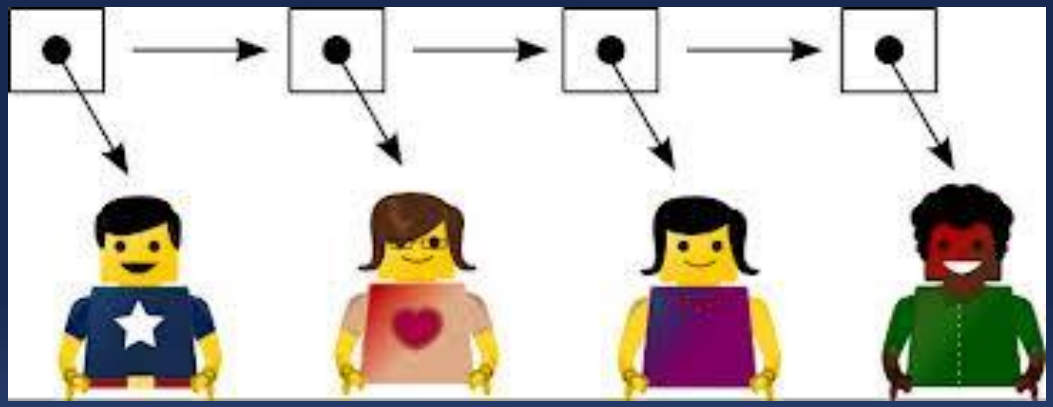
Ordem de complexidade: O(n).

```
package maua;

public class Test_List_Array {

        public static void main(String[] args) {

                List_Array x = new List_Array();
                x.Imprime_Lista();
                x.imprime_Primeiro();
                x.imprime_Ultimo();

                x.insereItem_Inicio(7);
                x.Imprime_Lista();
                x.imprime_Primeiro();
                x.imprime_Ultimo();

                x.insereItem_Fim(99);
                x.Imprime_Lista();
                x.imprime_Primeiro();
                x.imprime_Ultimo();
        }
}
```

Será que há algum modo de se implementar listas de forma mais eficiente ?

Listas Ligadas

- É um conjunto de nós que são definidos de forma **recursiva**.
- Cada nó tem um item de dado e uma **referência** ao próximo nó.

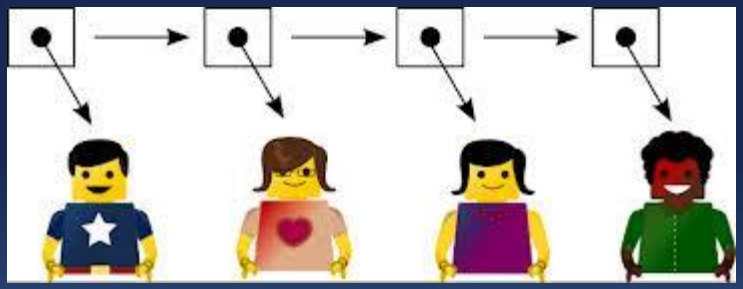


Vantagens sobre listas implementadas com arrays

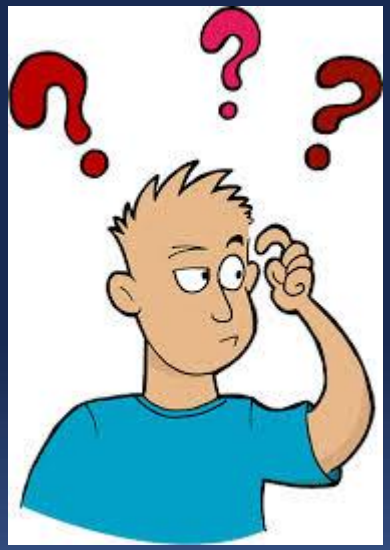
- A inserção de um item no meio da lista leva **tempo constante**, caso você tenha a referência ao prévio nó (O(1)).
- Listas ligadas podem crescer até o limite de memória oferecido pela máquina virtual.

Desvantagens sobre listas implementadas com arrays

- A busca do n^{th} elemento de um array é de tempo constante (**índice**).
- A busca do n^{th} elemento de uma lista ligada é proporcional a n, sendo n o tamanho da lista. (A pesquisa se inicia a partir do HEAD até se encontrar de forma exaustiva o item procurado).

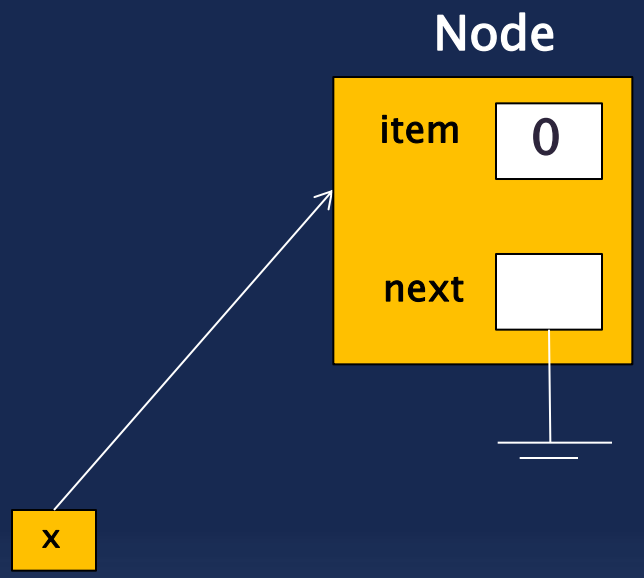


Considerando que a lista ligada é uma lista de nós, como implementar um nó da lista ?



Implementação de Nós

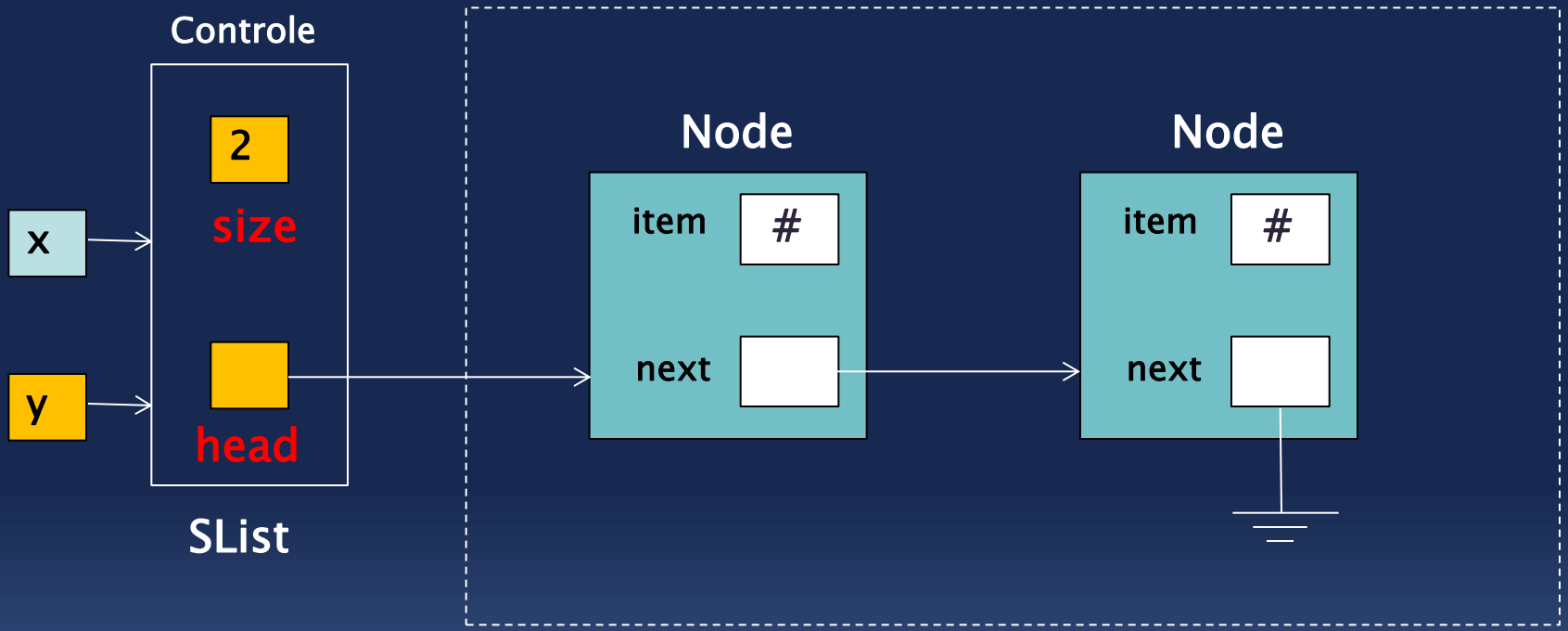
```
package maua;

public class Node {

        int item;
        Node next;

}
```



Criação de Nós

```
package maua;

public class Test_ListNode {

        public static void main(String[] args) {

                Node N1;
                N1 = new Node();
                N1.item = 8;

                Node N2;
                N2 = new Node();
                N2.item = 5;

                Node N3;
                N3 = new Node();
                N3.item = 9;

        }
}
```

Lista Simplesmente Ligada – Classe SList

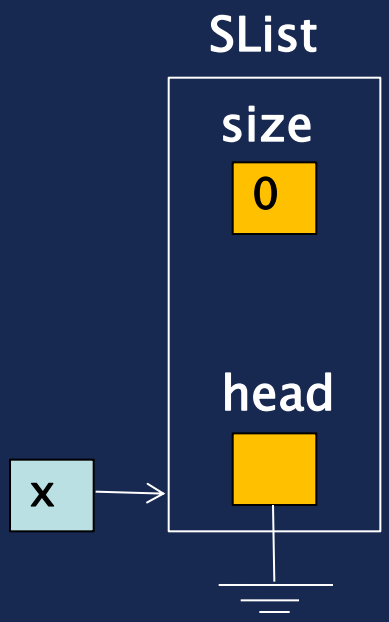


Classe Node

```
package maua;

public class Node {
        int item;
        Node next;

public Node() {
        this.item=0;
        this.next=null;
}

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

Classe SList

```
package maua;

public class SList {

        Node head;
        int size;

public SList() {
        this.head = null;
        this.size = 0;
}
```



insereInicio()

```
public void insereInicio (int item) { // Insere no inicio da lista
	Node x = new Node(item);
	if (this.size == 0 ) {
		this.head = x;
		this.size++;
	}
	else {
		x.next = this.head ;
		this.head = x;
		this.size++;
	}
}
```

O(1)

insereFim ()

```
public void insereFim(int item) { // Insere no fim da lista

        Node no_novo = new Node(item);

        if (this.size == 0) {

                this.head = no_novo;
                this.size++;

        }
        else {
                int contador = 1;
                Node no_trab = this.head;

                while (contador < this.size) {
                        no_trab = no_trab.next ;
                        contador++;
                }
                no_trab.next = no_novo;
                this.size++;
        }
}
```

O(n)

imprimeLista()

```
public void imprimeLista() {
        System.out.println("Funcao imprimeLista() ......");
        System.out.print("Lista: ");
        if (this.size == 0)
                System.out.print(" vazia...");
        else {
                int contador = 1;
                Node no_trab = this.head;
                while (contador <= this.size) {
                        System.out.println("contador = " + contador );
                        System.out.print (" " + no_trab.item);
                        no_trab = no_trab.next;
                        contador++;
                }
        }
        System.out.println("");
}
```

O(n)

deleteInicio()

```
public void deleteInicio() {

        if (this.head == null )
                System.out.println("Impossivel deletar... Lista vazia...");

        else {
                this.head = this.head.next;

                this.size--;
        }

}
```

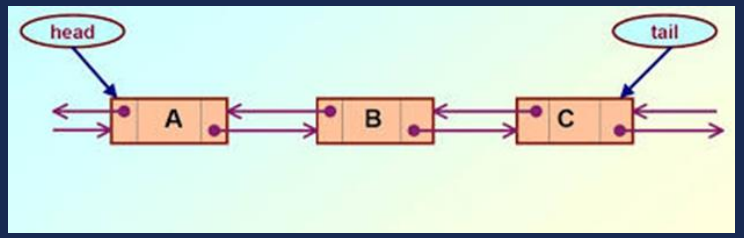
deleteFim()

```
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--;
            }

  }
```

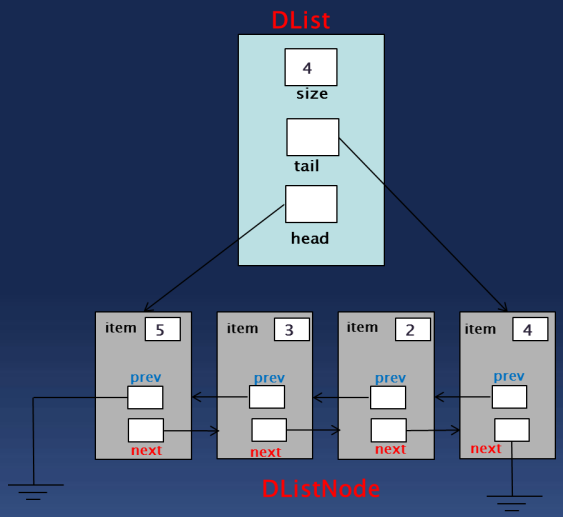
Classe TestSList

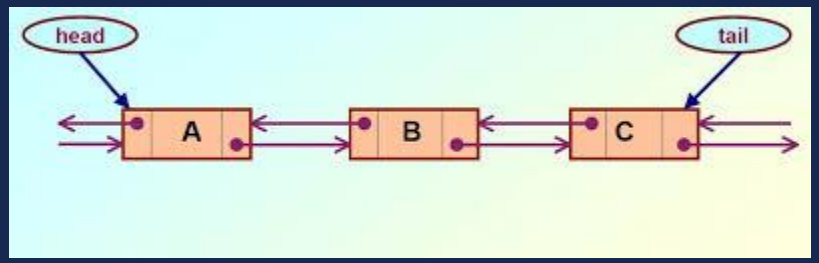
```
package maua;

public class TesteSList {

        public static void main(String[] args) {

                SList S1 = new SList();
                S1.imprimeLista();

                S1.insereFim(10);
                S1.imprimeLista();

                S1.insereFim(99);
                S1.imprimeLista();

                S1.insereFim(33);
                S1.imprimeLista();

                S1.insereInicio(44);
                S1.imprimeLista();

                S1.deleteFim();
                S1.imprimeLista();

                S1.deleteInicio();
                S1.imprimeLista();

                S1.deleteFim();
                S1.imprimeLista();

        }

}
```

```
Funcao imprimeLista() ......
Lista:  vazia...
Funcao imprimeLista() ......
Lista:  10
Funcao imprimeLista() ......
Lista:  10 99
Funcao imprimeLista() ......
Lista:  10 99 33
Funcao imprimeLista() ......
Lista:  44 10 99 33
Funcao imprimeLista() ......
Lista:  44 10 99
Funcao imprimeLista() ......
Lista:  10 99
Funcao imprimeLista() ......
Lista:  10
```



Porque listas duplamente ligadas

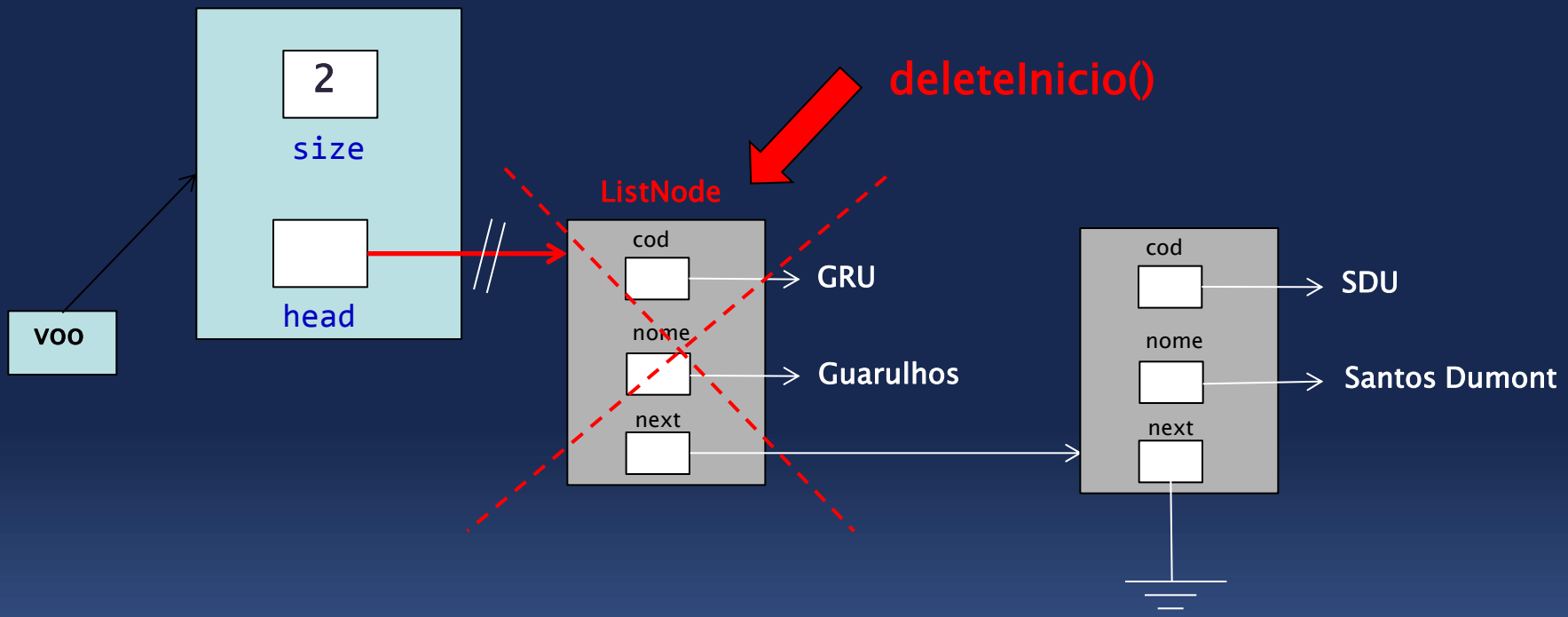




Qual a necessidade de listas duplamente ligadas ?

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

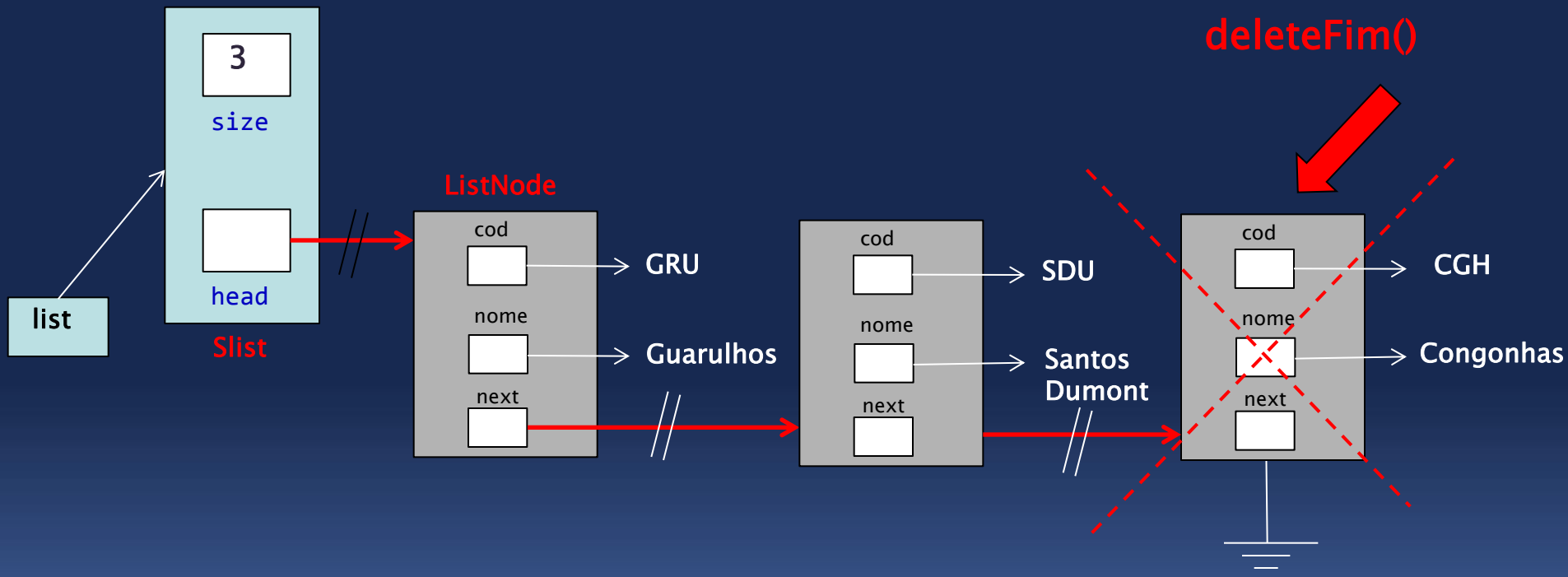
```
public void deleteInicio() {

        if (this.head == null )
                System.out.println("Impossivel 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 !).

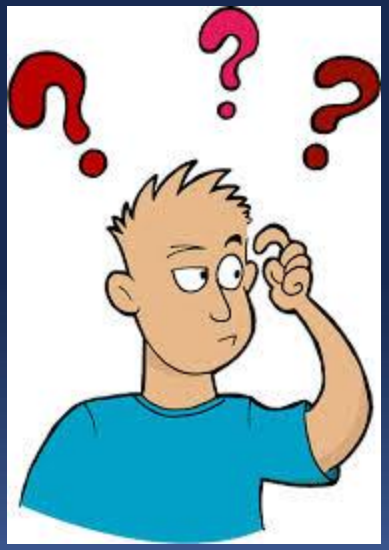


Função deleteFim()

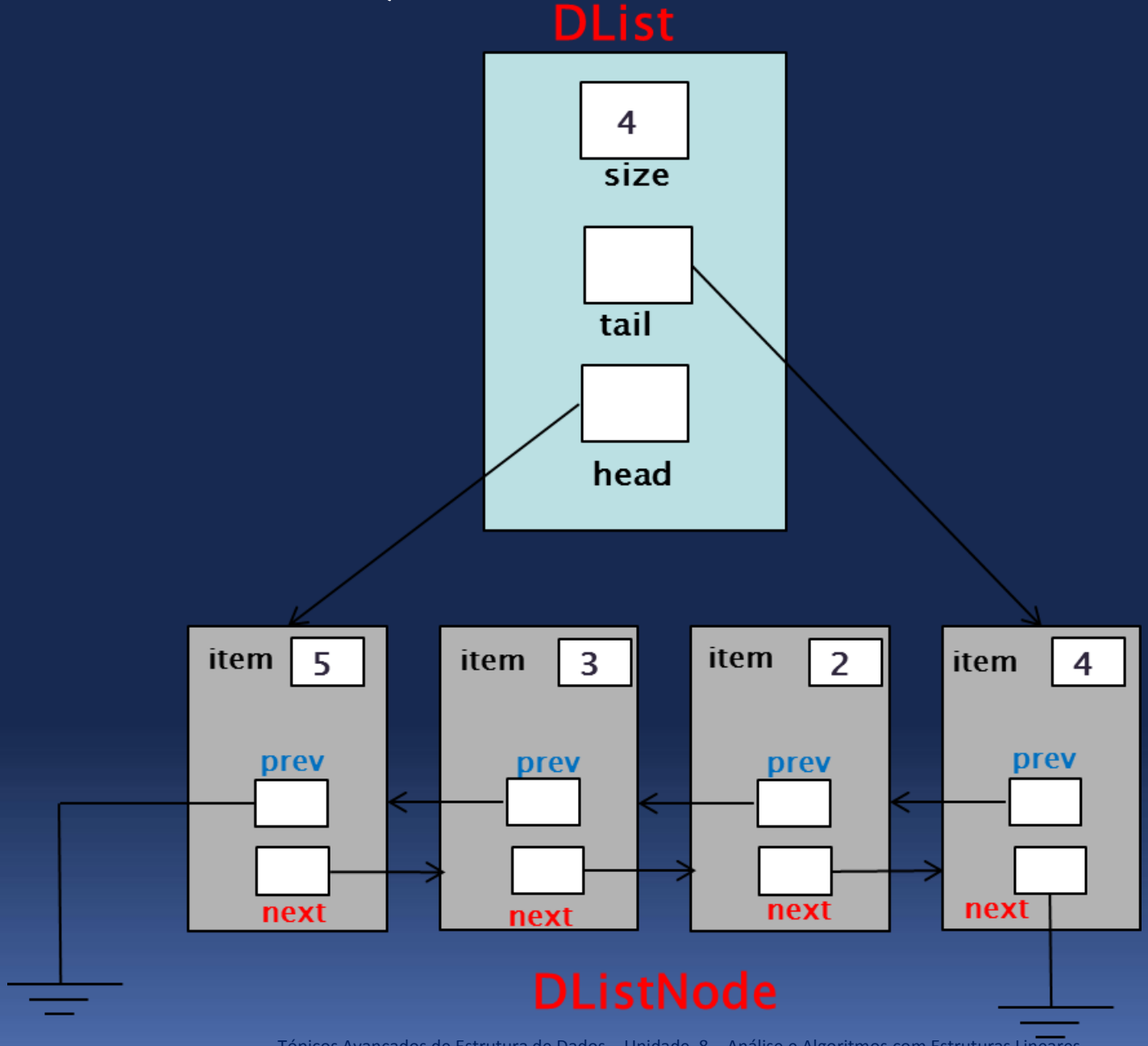
```
public void deleteFim() {

    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--;
            }
  }
```

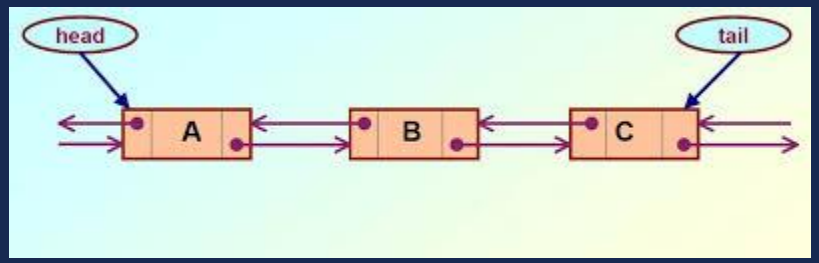
O Tempo é proporcional ao tamanho da lista !

Como então melhorar essa estrutura de dados ?



Porque listas duplamente ligadas

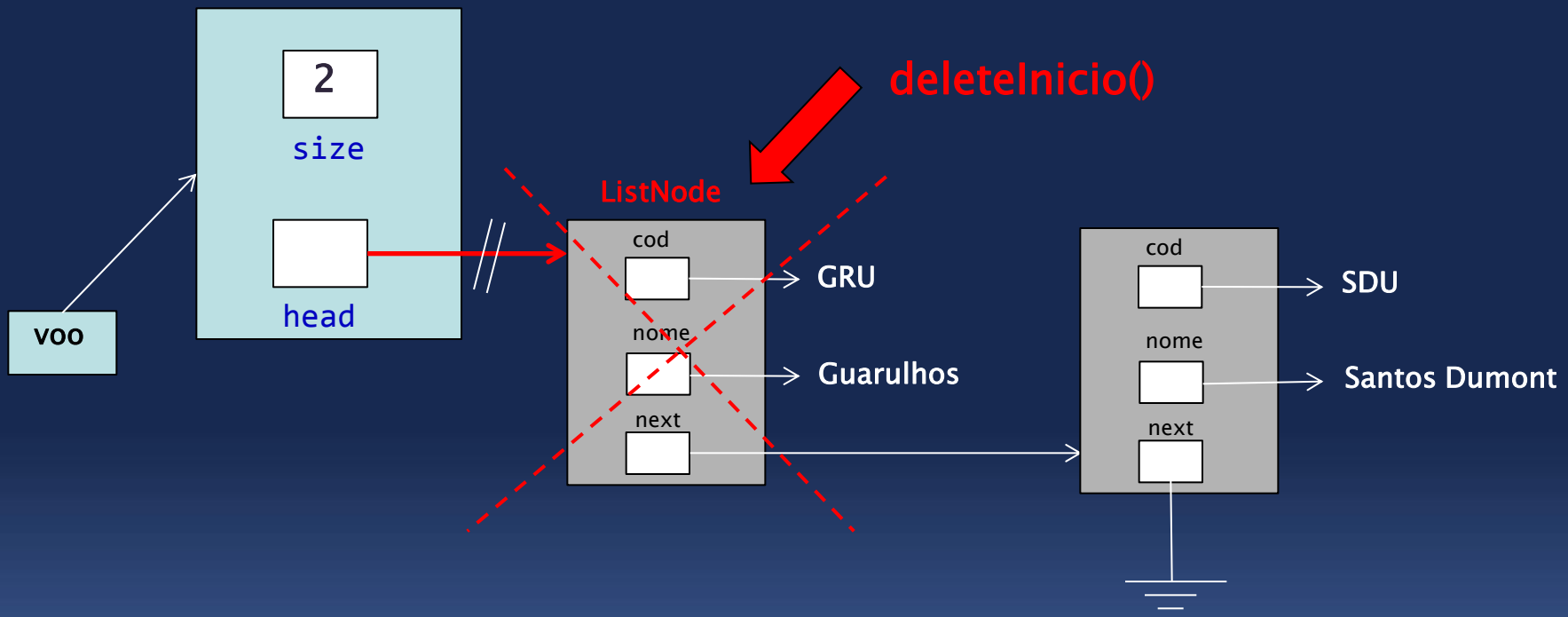




Qual a necessidade de listas duplamente ligadas ?

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

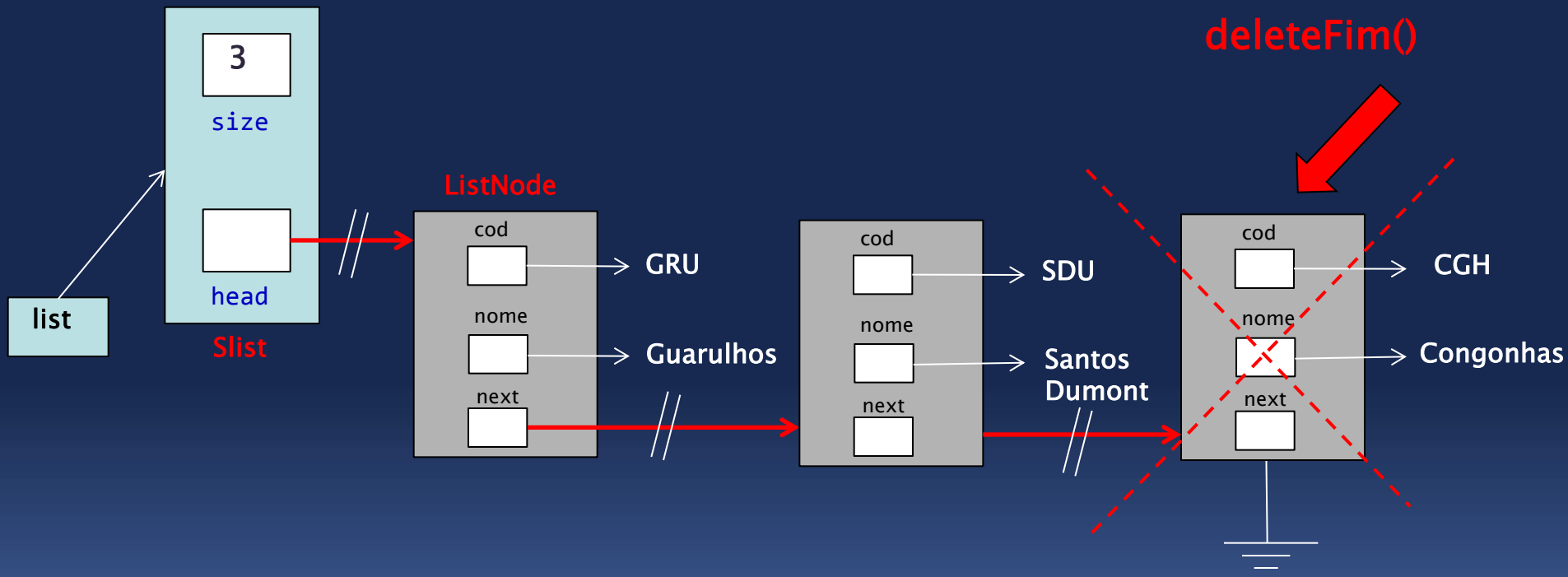
```
public void deleteInicio() {

        if (this.head == null )
                System.out.println("Impossivel 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 !).

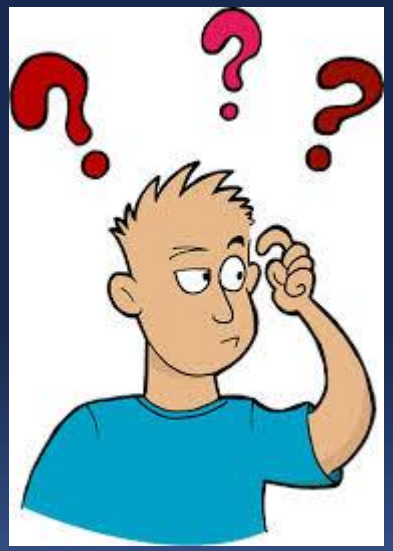


Função deleteFim()

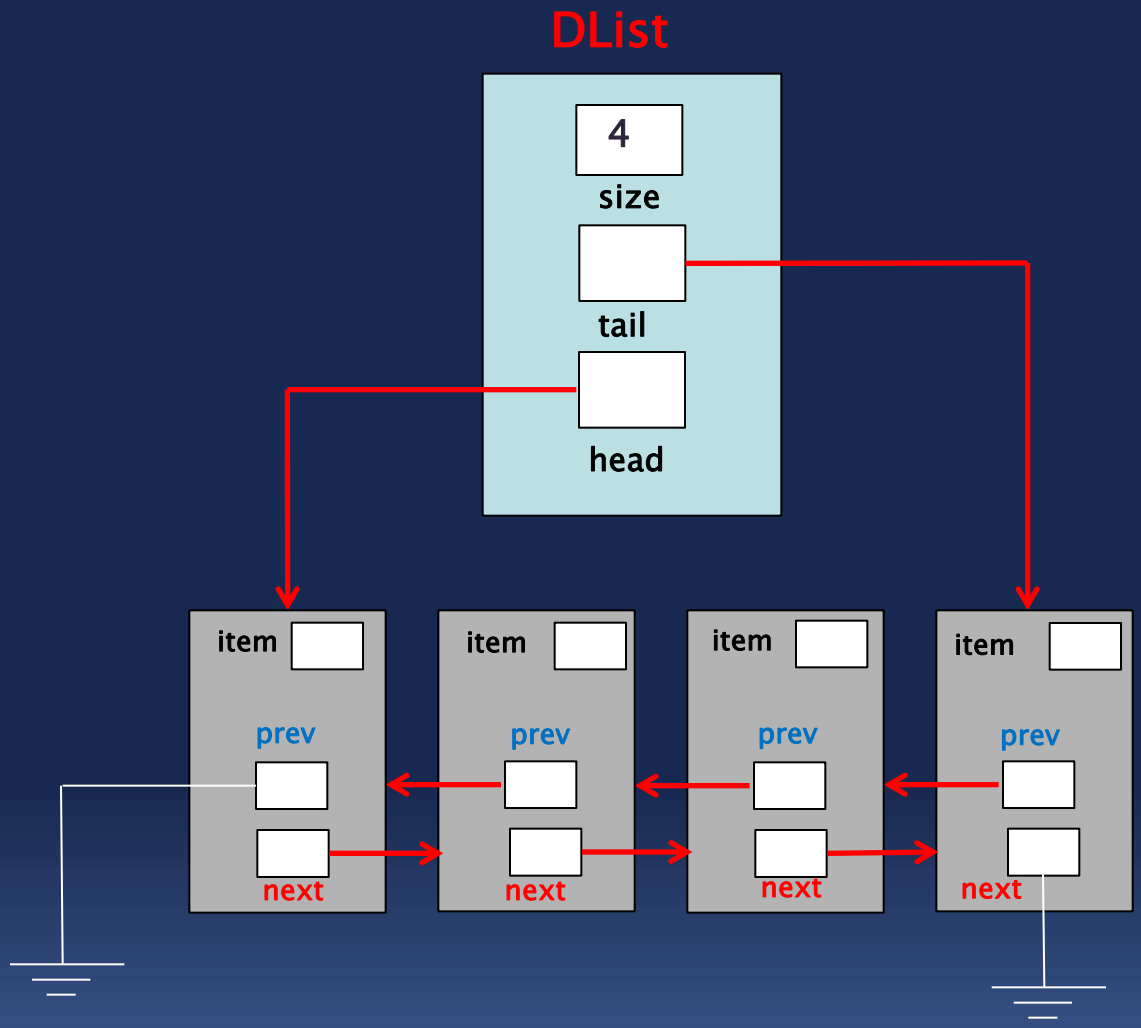
```
public void deleteFim() {

    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 ?



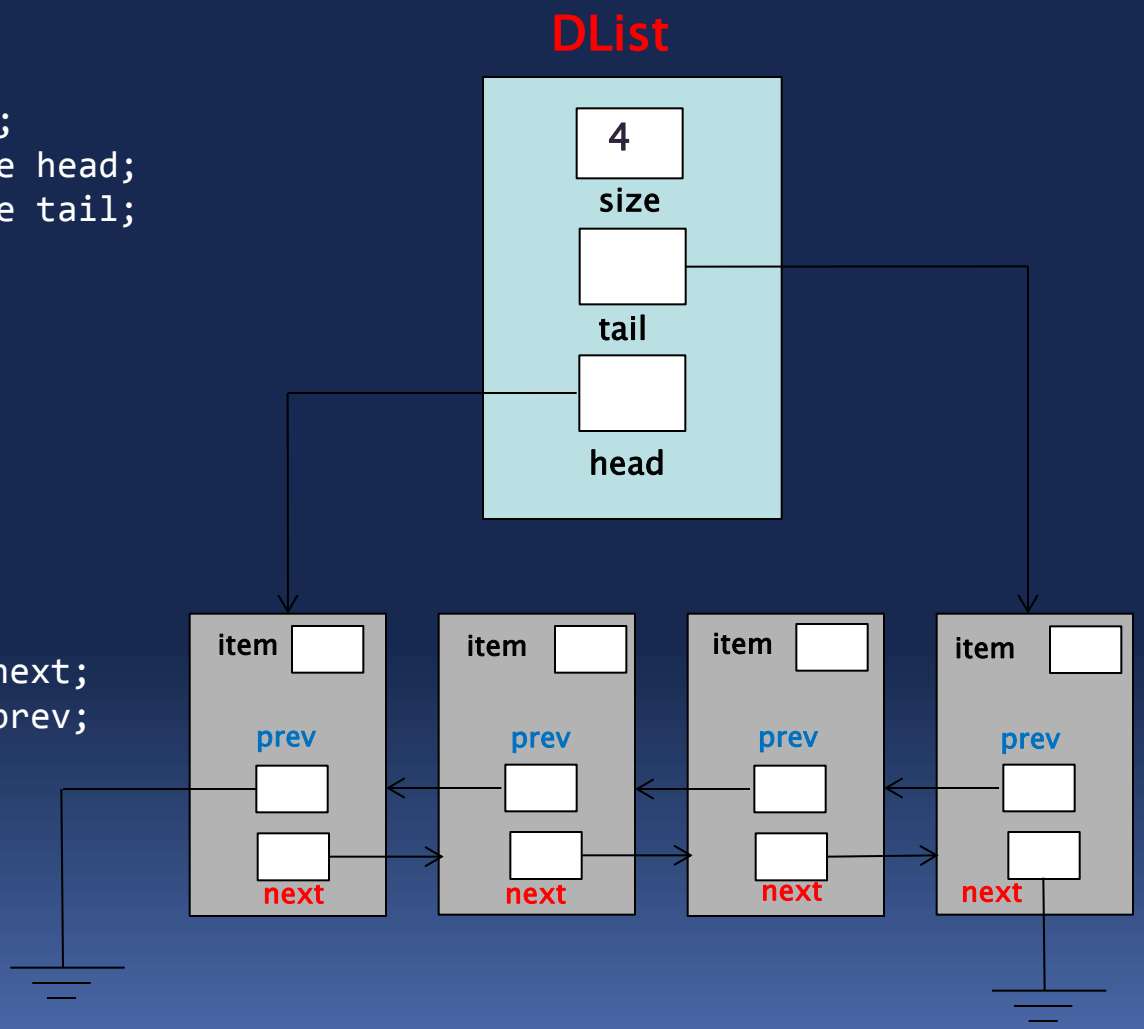
Estrutura Dlist



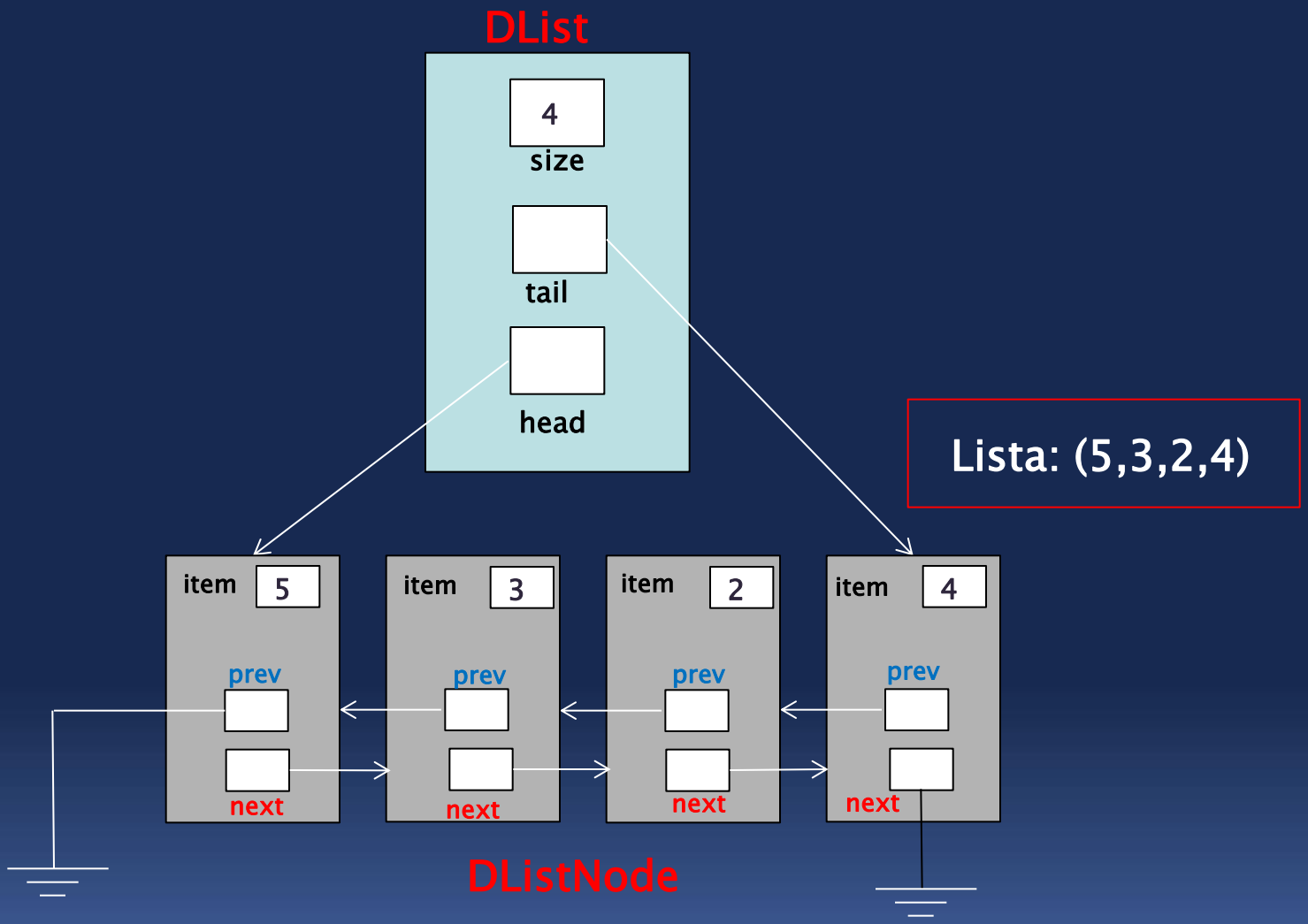
Listas duplamente ligadas

```
package maua;

public class DList {

        public   int size;
        public   DListNode head;
        public   DListNode tail;

}
```

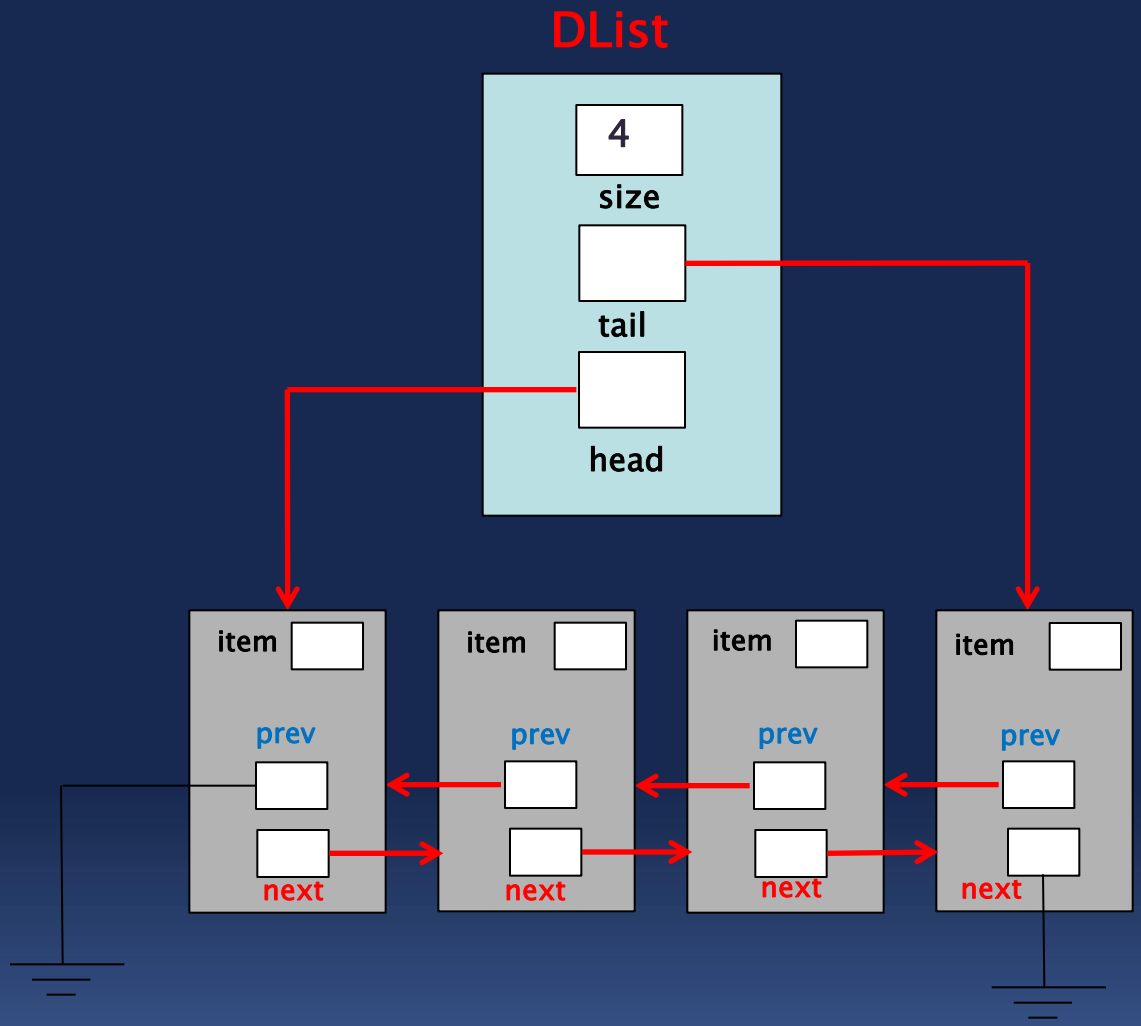
```
package maua;

public class DListNode {

        public int item;
        public DListNode next;
        public DListNode prev;
}
```



Exemplo: Lista Duplamente ligada



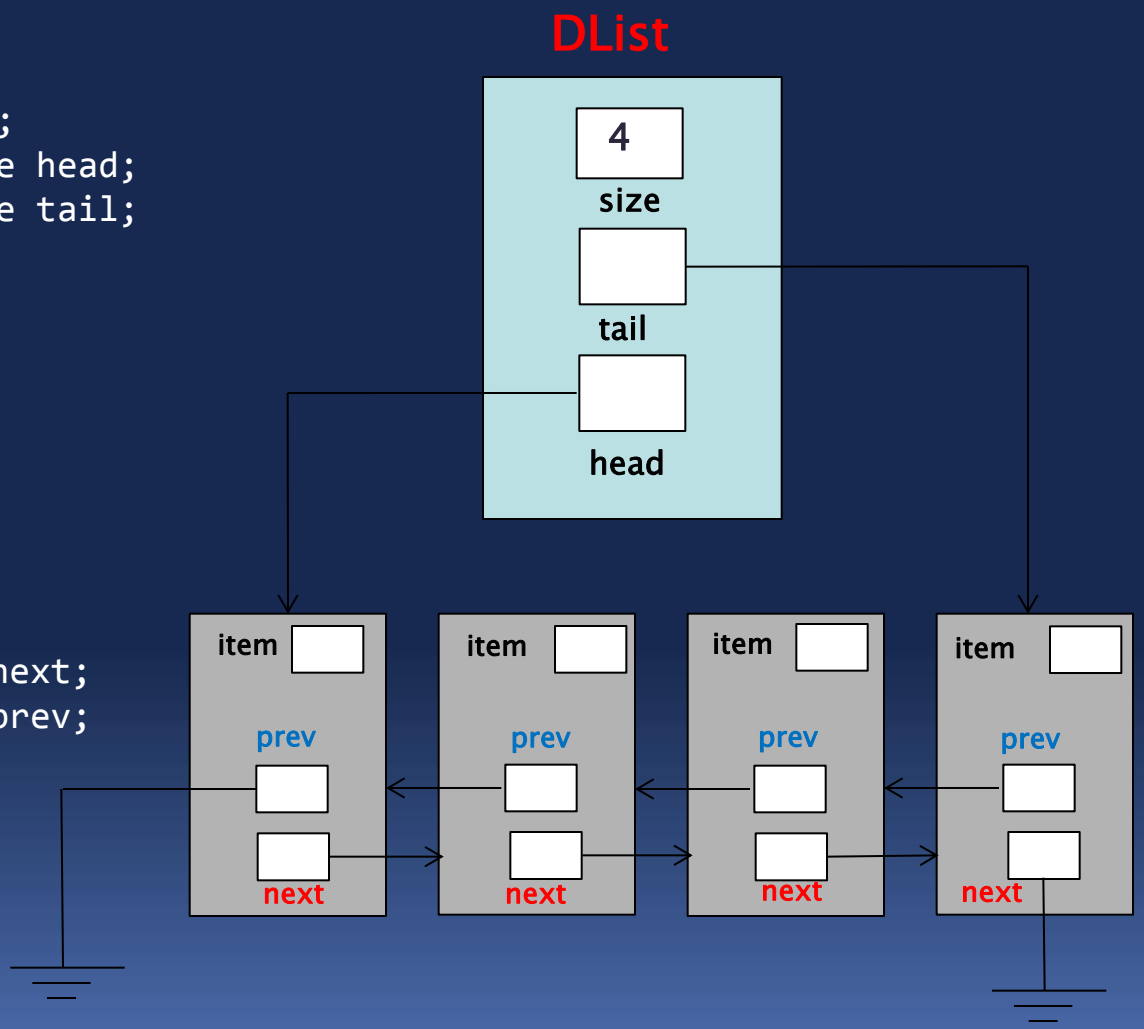
Estrutura Dlist



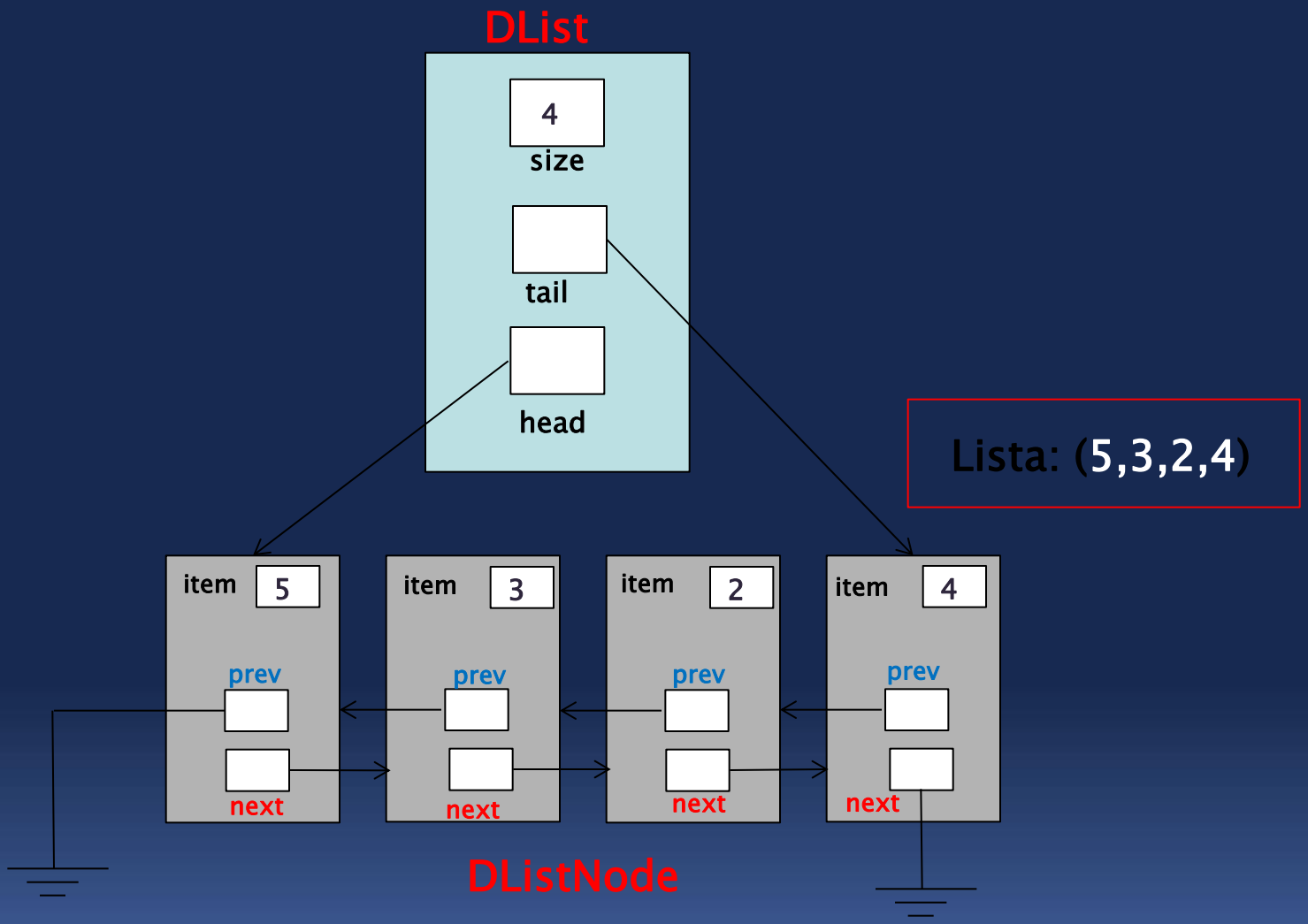
Listas duplamente ligadas

```
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;
}
```



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

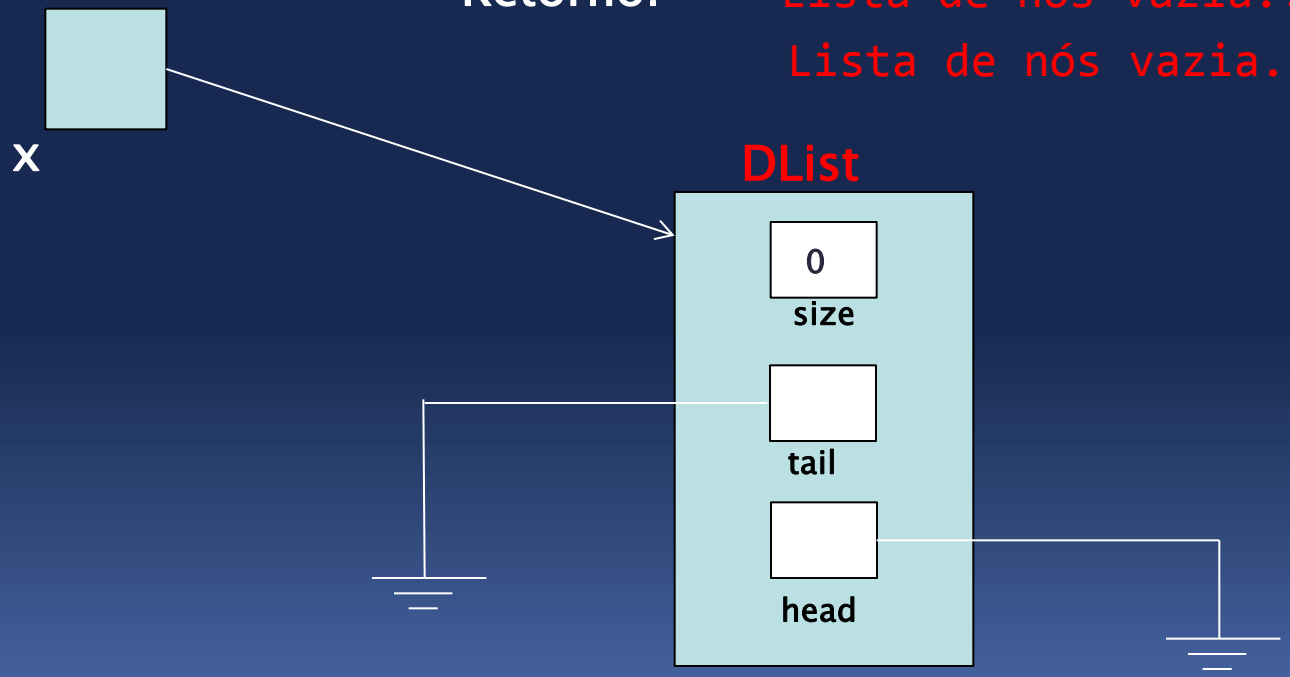
```
package maua;

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();
```

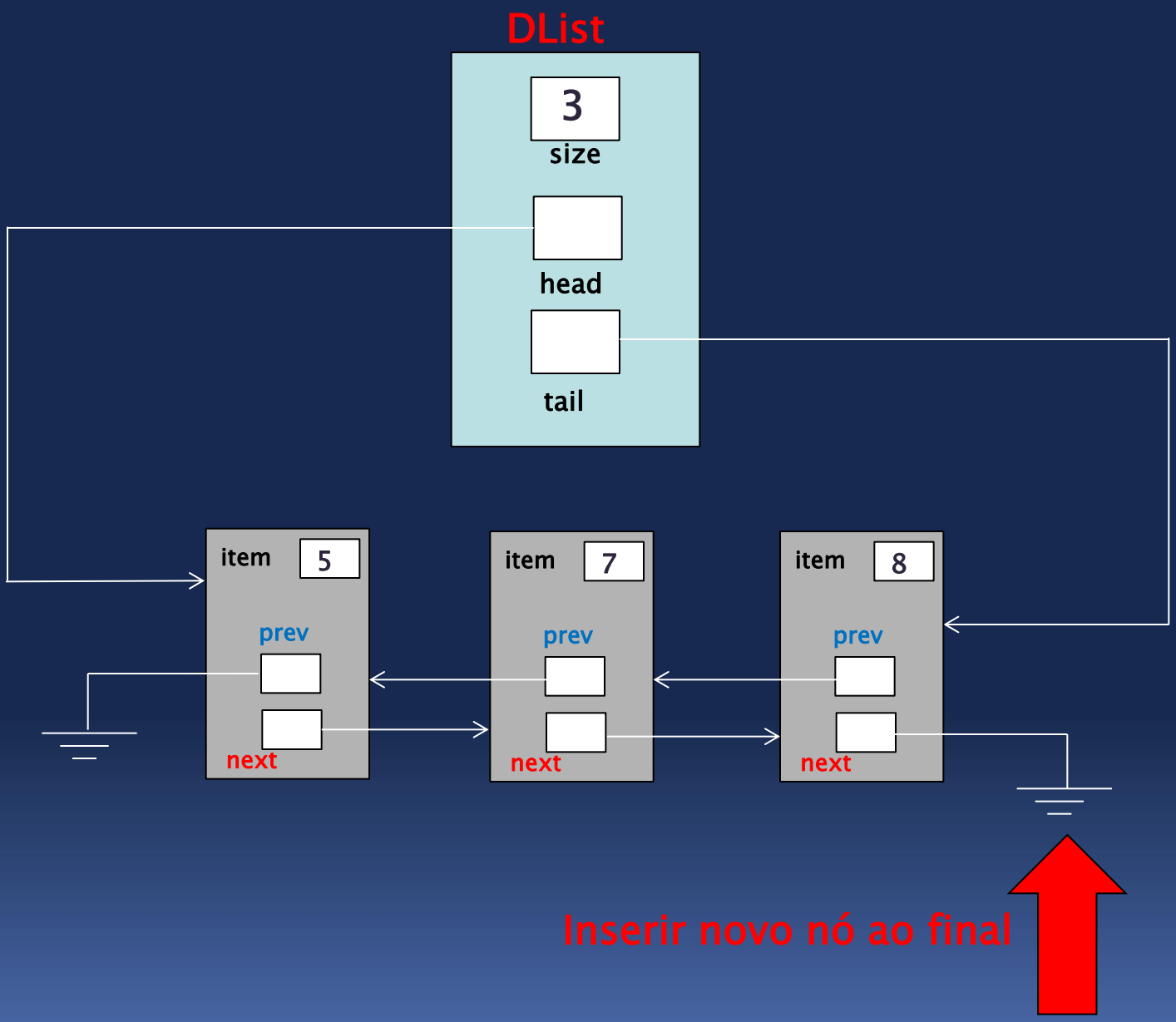
Retorno:

```
Lista de nós vazia....
Lista de nós vazia....
```

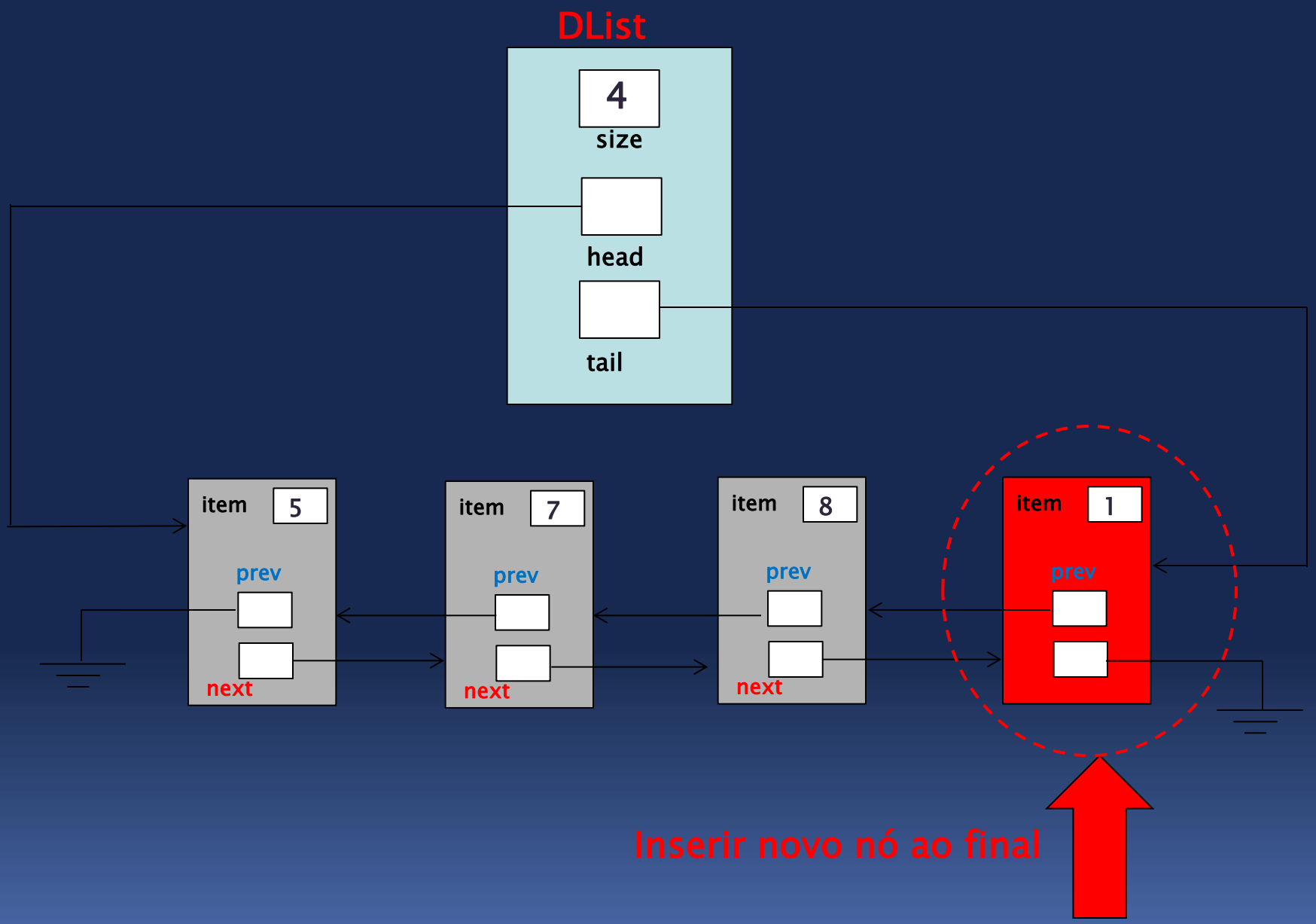


Como inserir nós na Lista ?

Inserindo novo nó no fim da lista



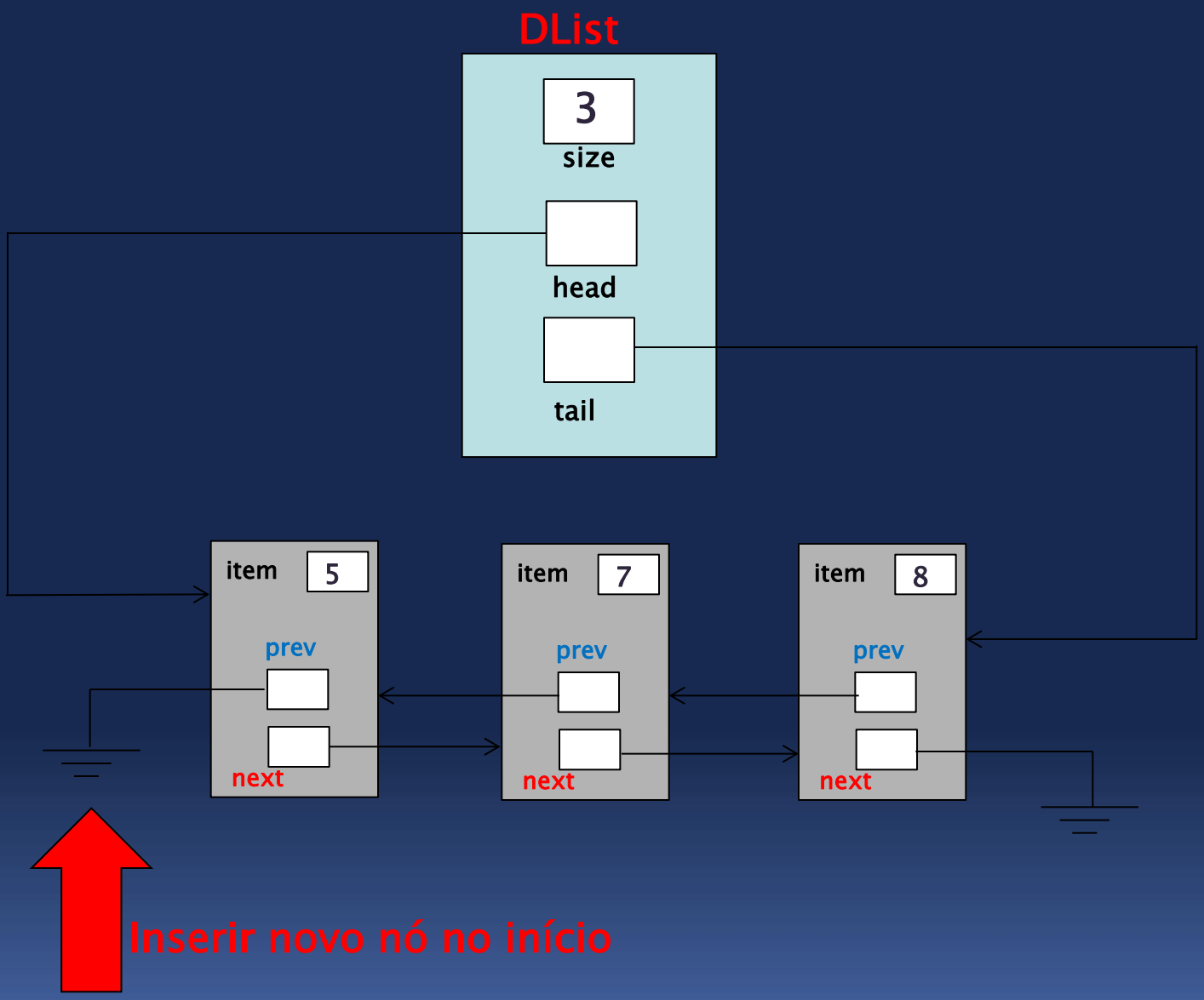
Inserindo nó no fim da lista



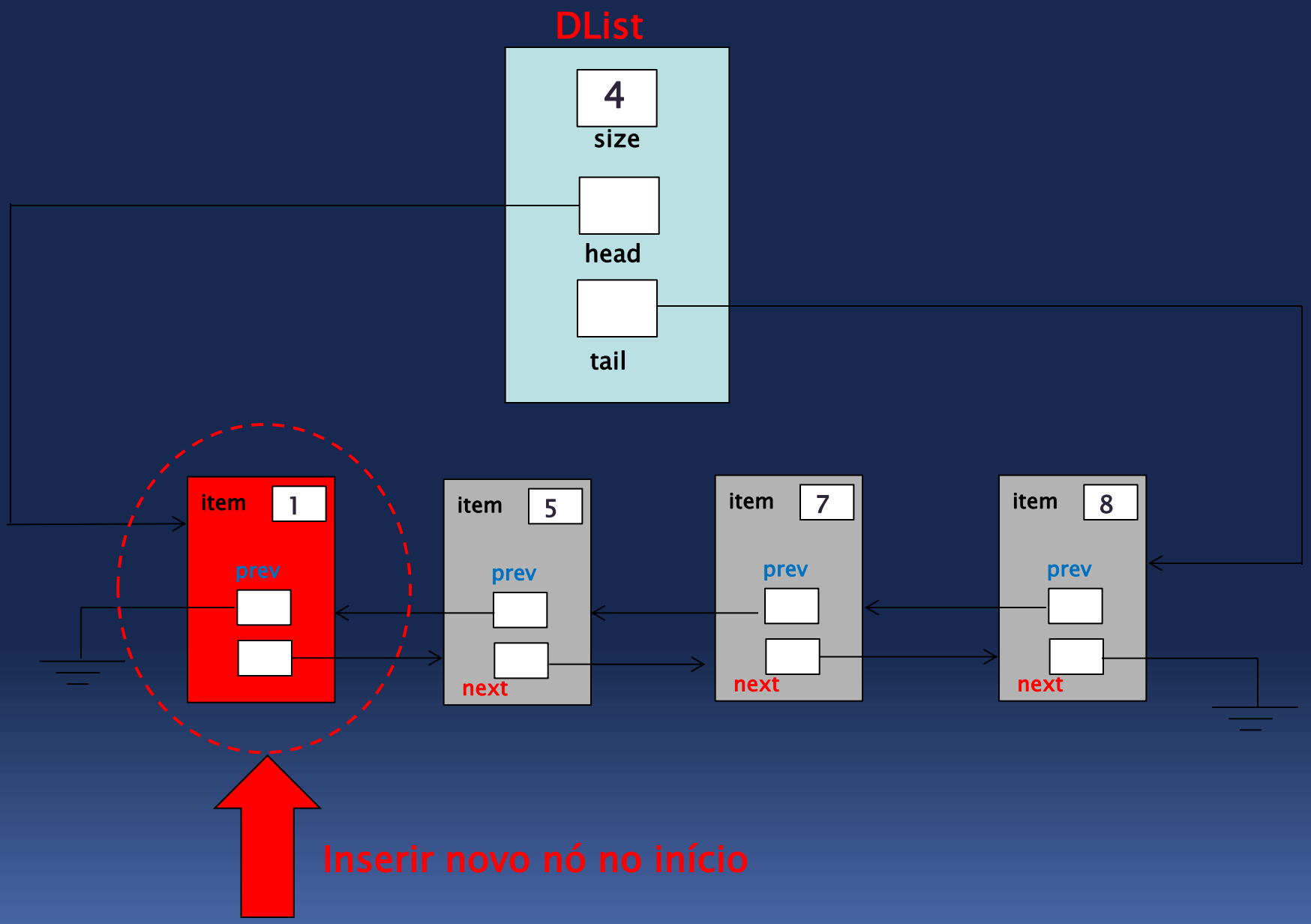
Função insereFim(item);

```
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



Inserindo nó no início da lista



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;
                }
        }

}
```

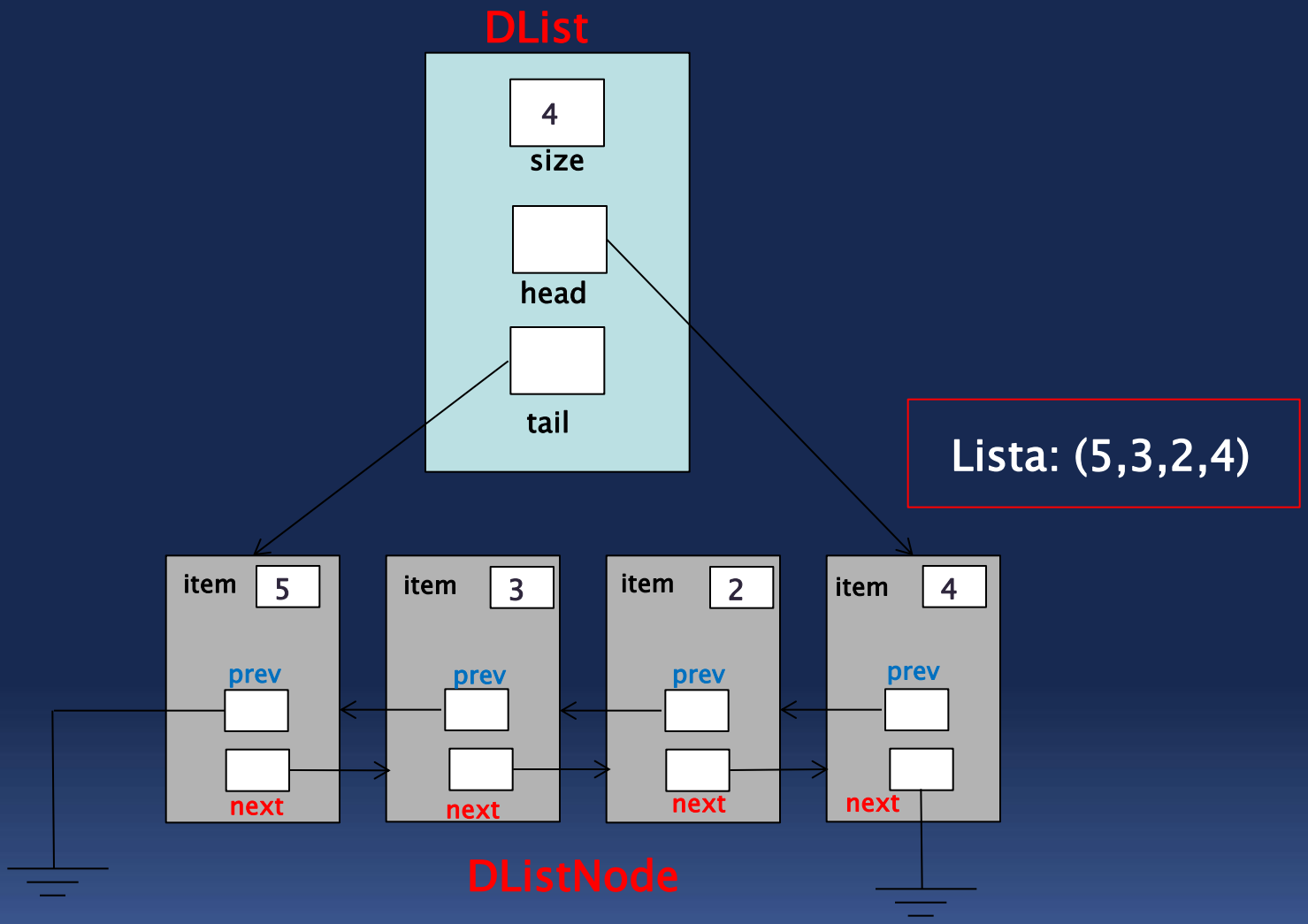
- Ordem de complexidade: O(**n**).

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

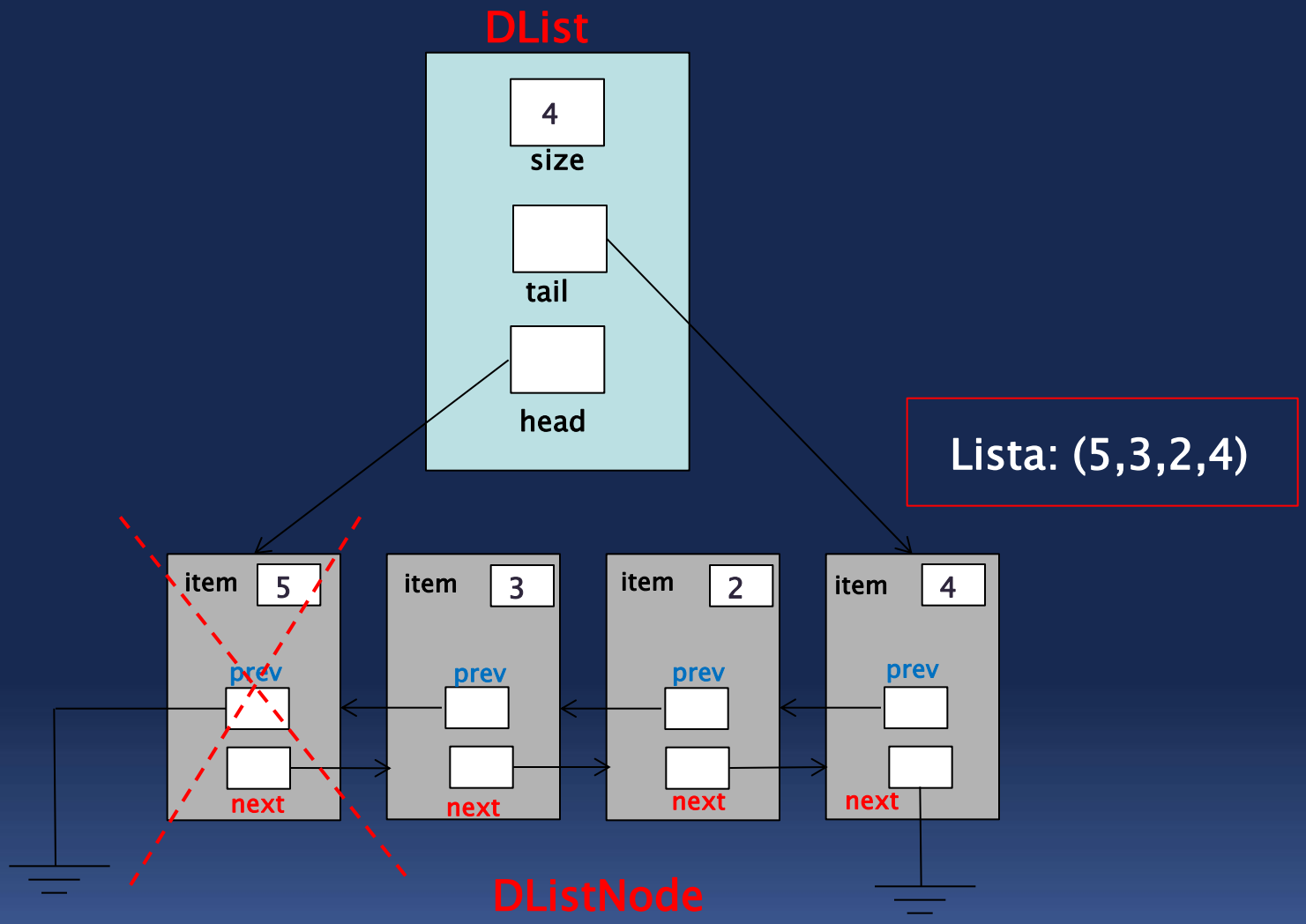


Gerando a lista completa

```
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() - Pseucódigo

```
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() - Pseucódigo

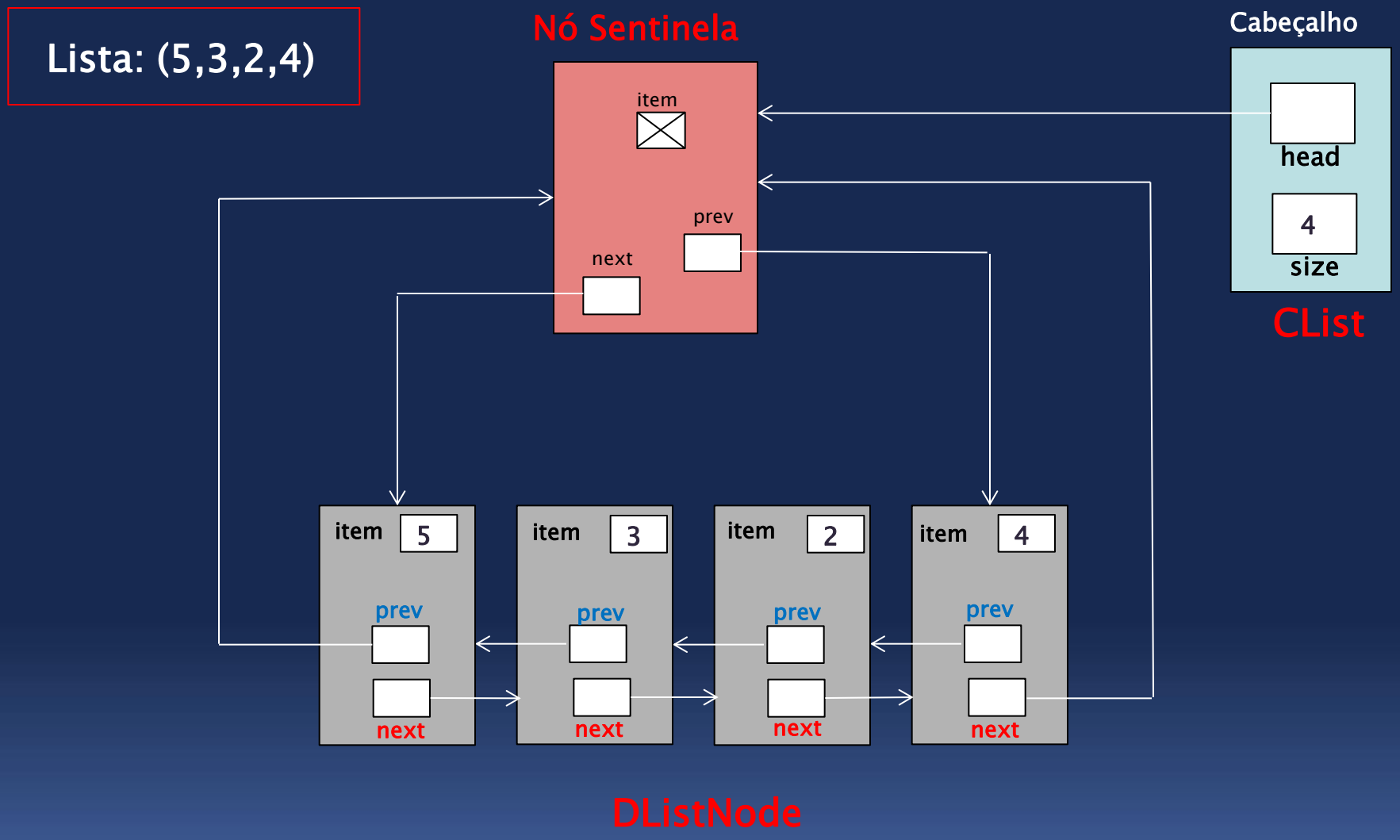
```
deleteLast() {

        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**).

Lista Circular



Definição do nó da lista

```
package maua;

public class DListNode {

        public int item;

        public DListNode next;

        public DListNode prev;
```

Definindo os construtores

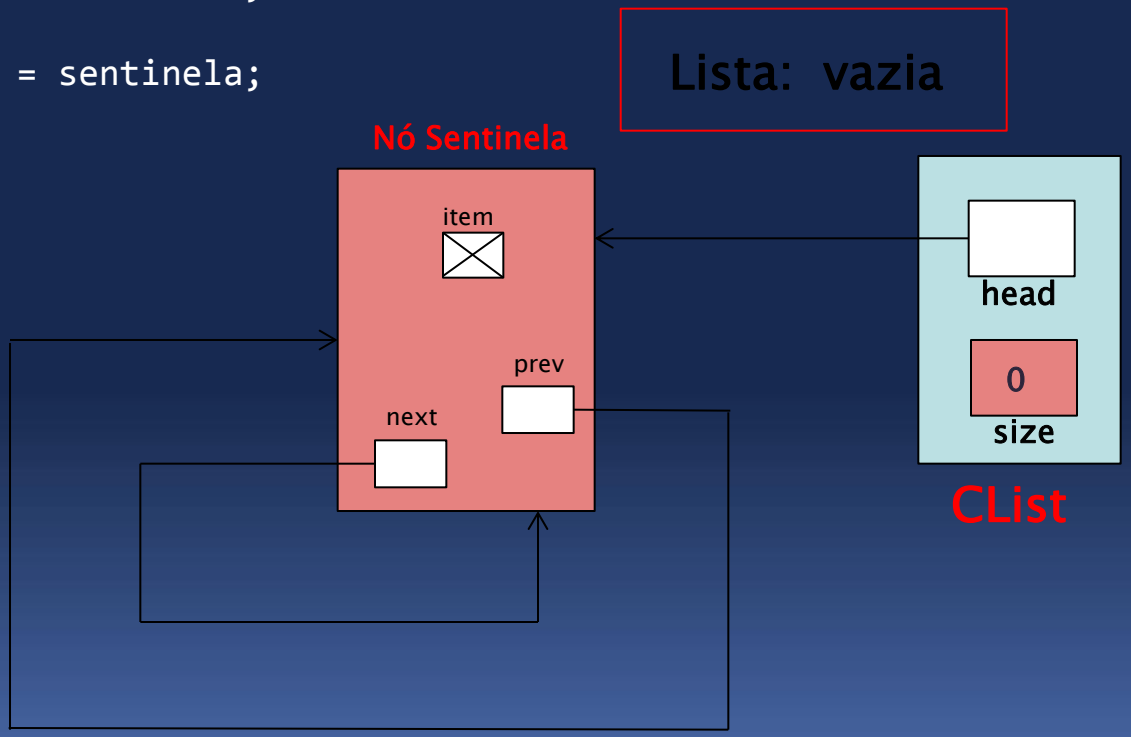
```
    public DListNode(int item, DListNode next, DListNode prev) {

            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();

        }

}
```

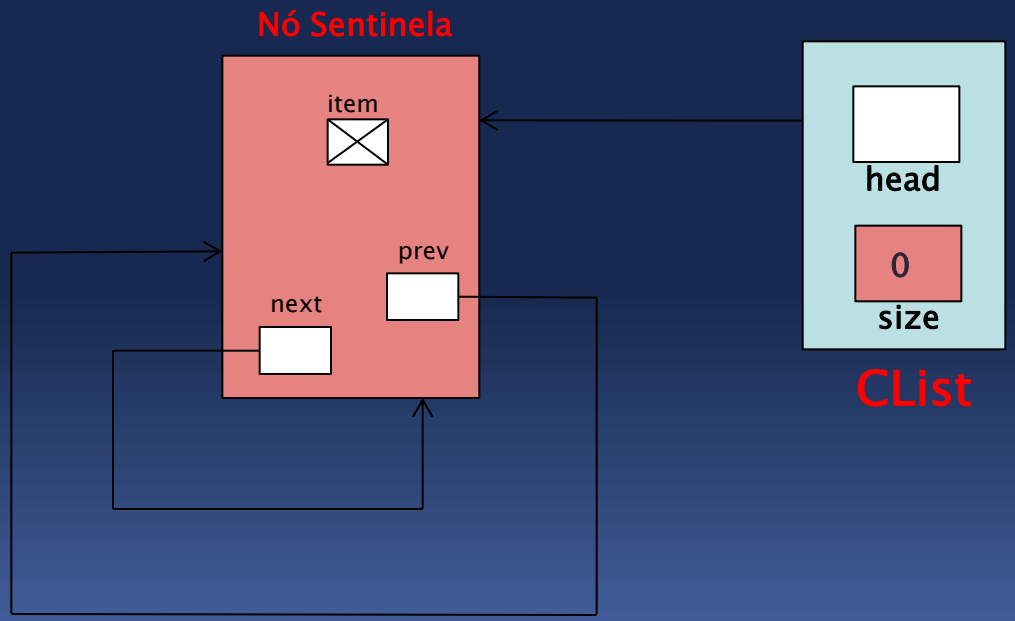
Criando lista de nós vazia

```
CList listaCircular = new CList();

listaCircular.imprimeFirst();

listaCircular.imprimeLast();
```

Retorno:

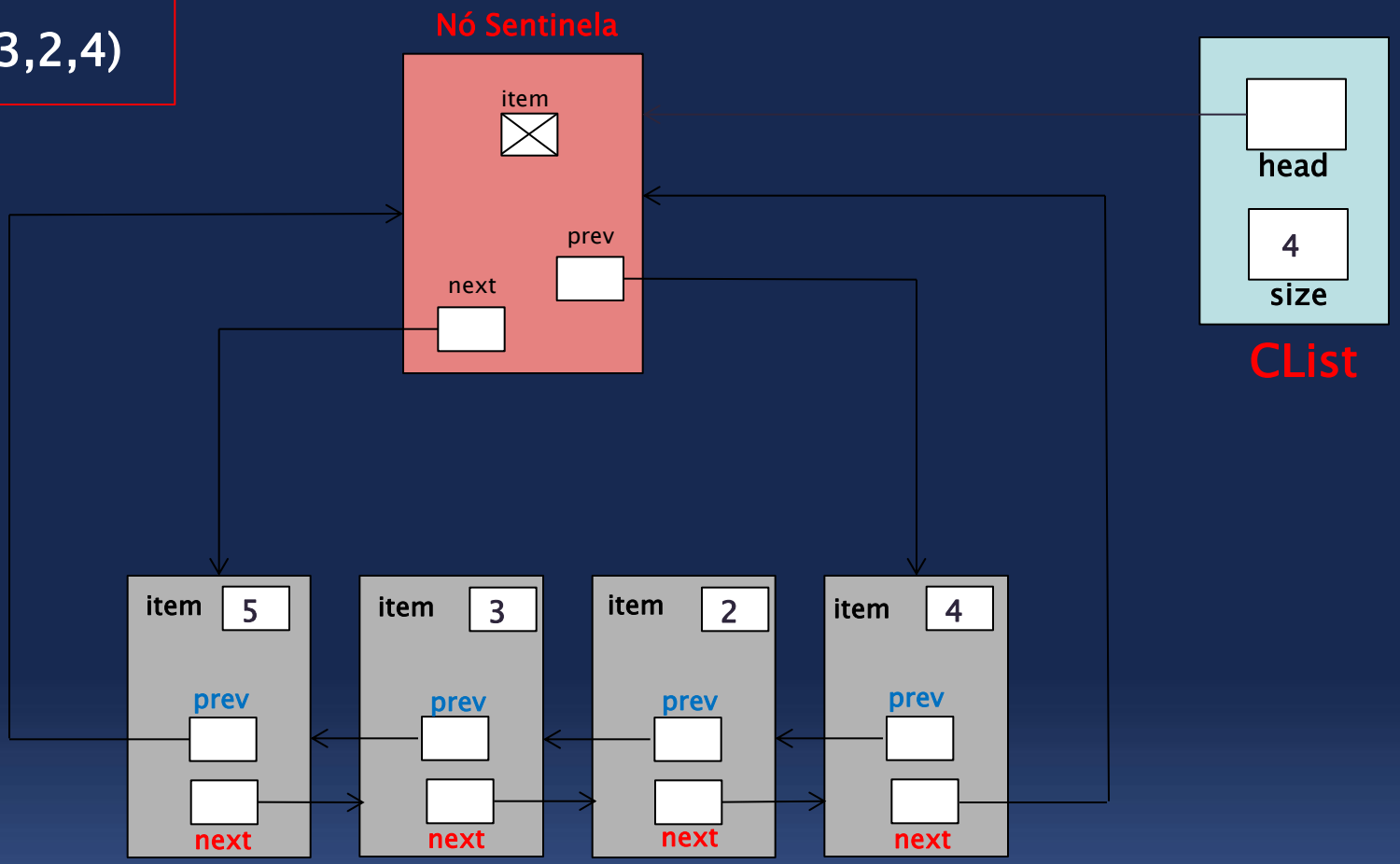
```
Lista de nós vazia....
Lista de nós vazia....
```

Lista: vazia



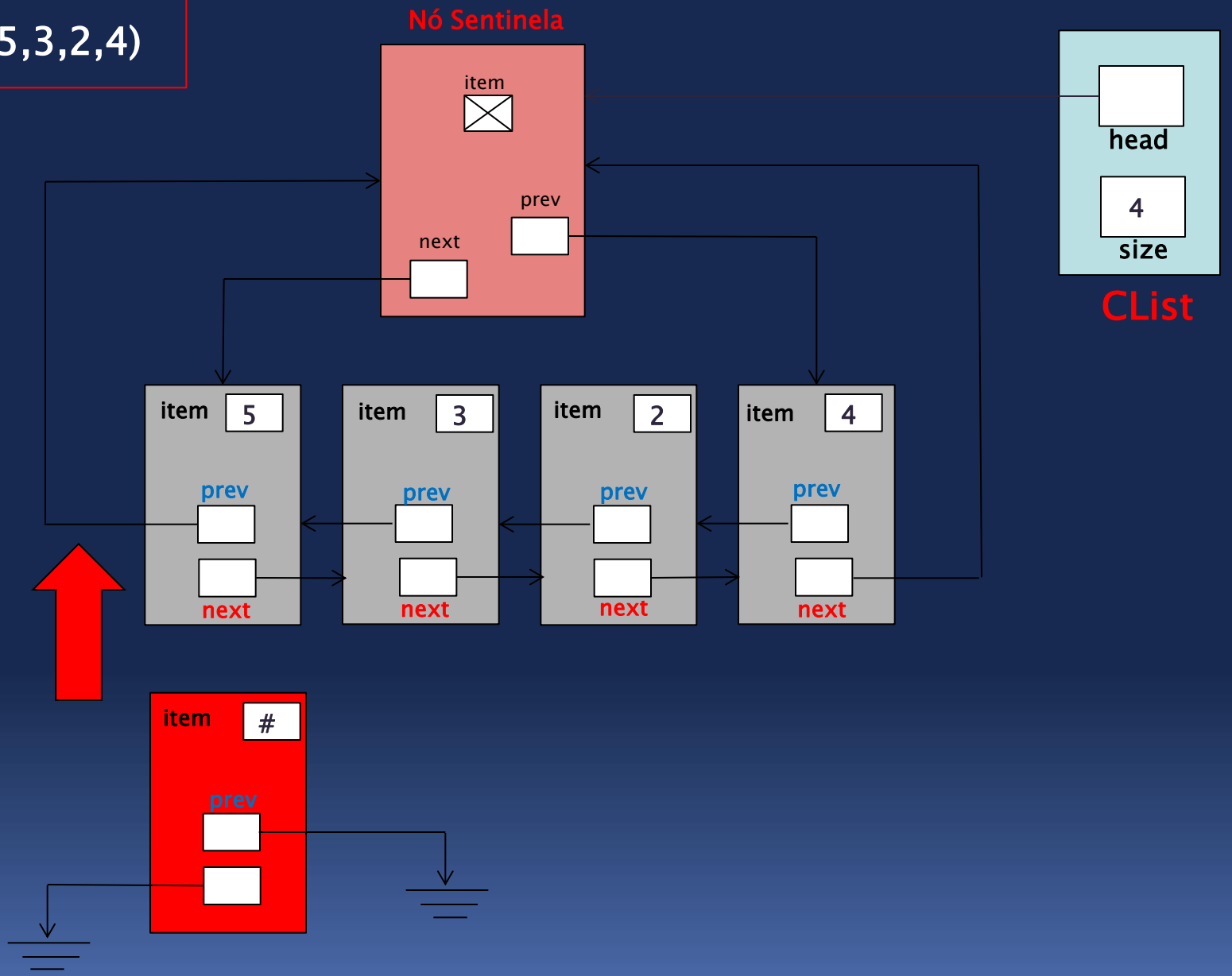
Inserindo nós na lista de nós

Lista: (5,3,2,4)

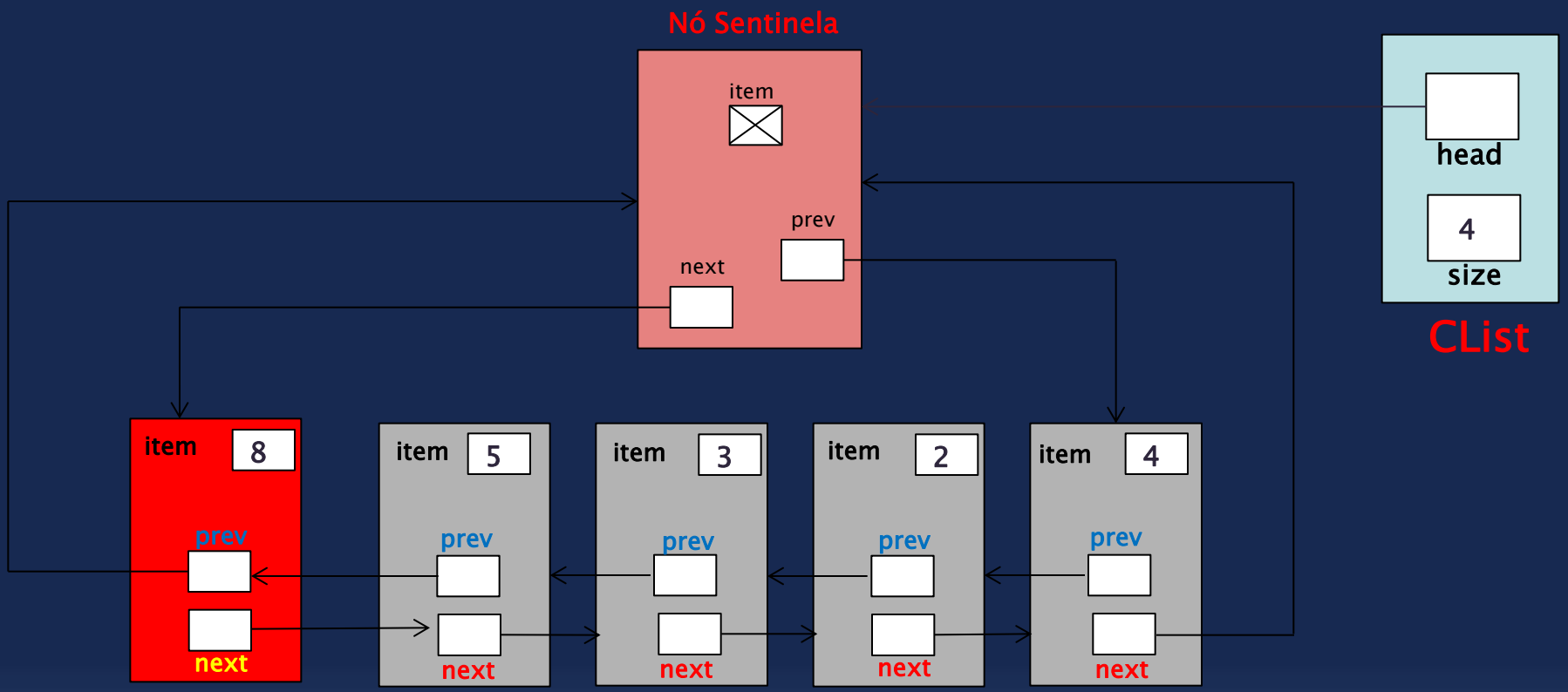


Função insereFirst(item)

Lista: (5,3,2,4)



Função insereFirst(item)



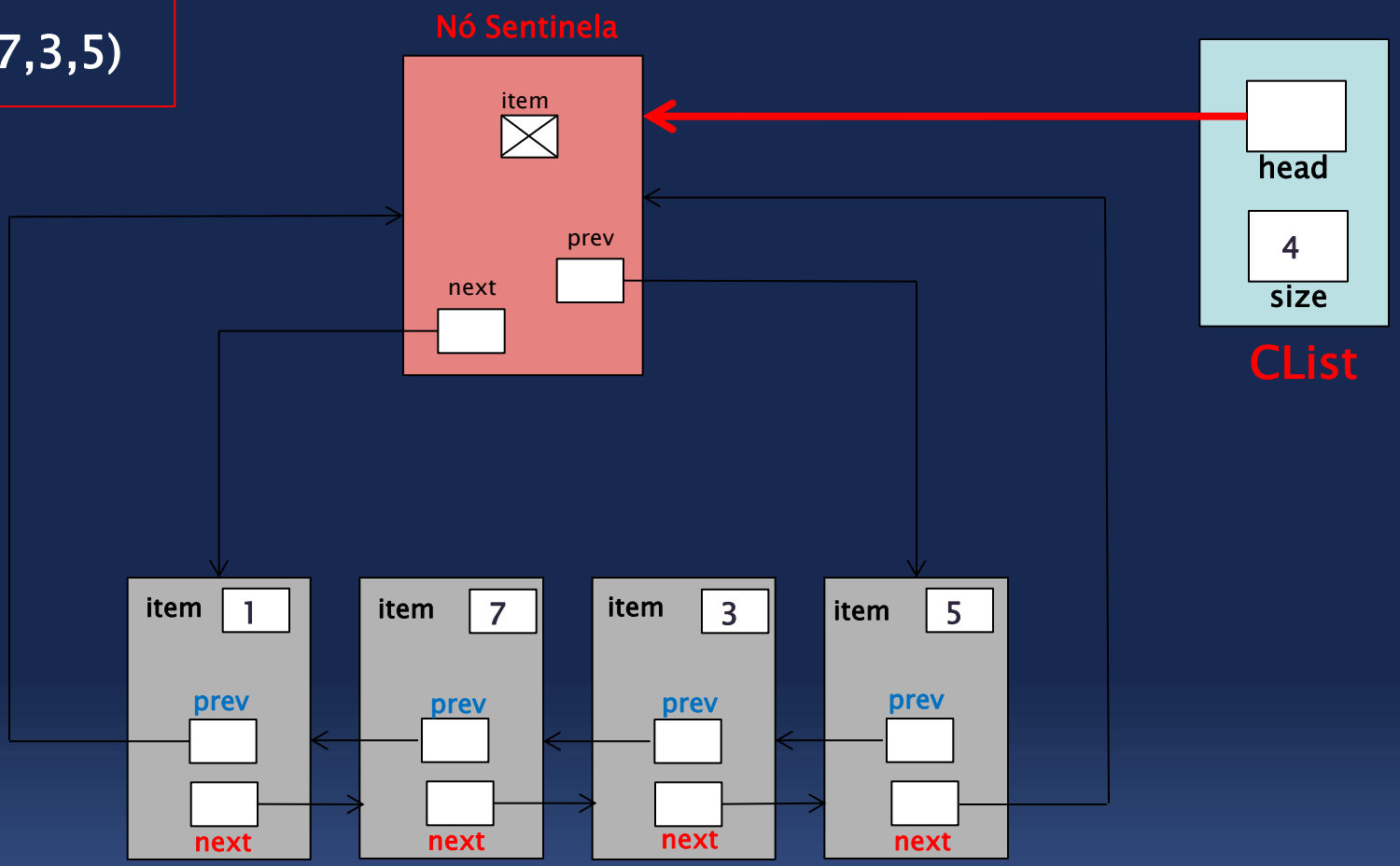
Lista: (8, 5,3,2,4)

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)



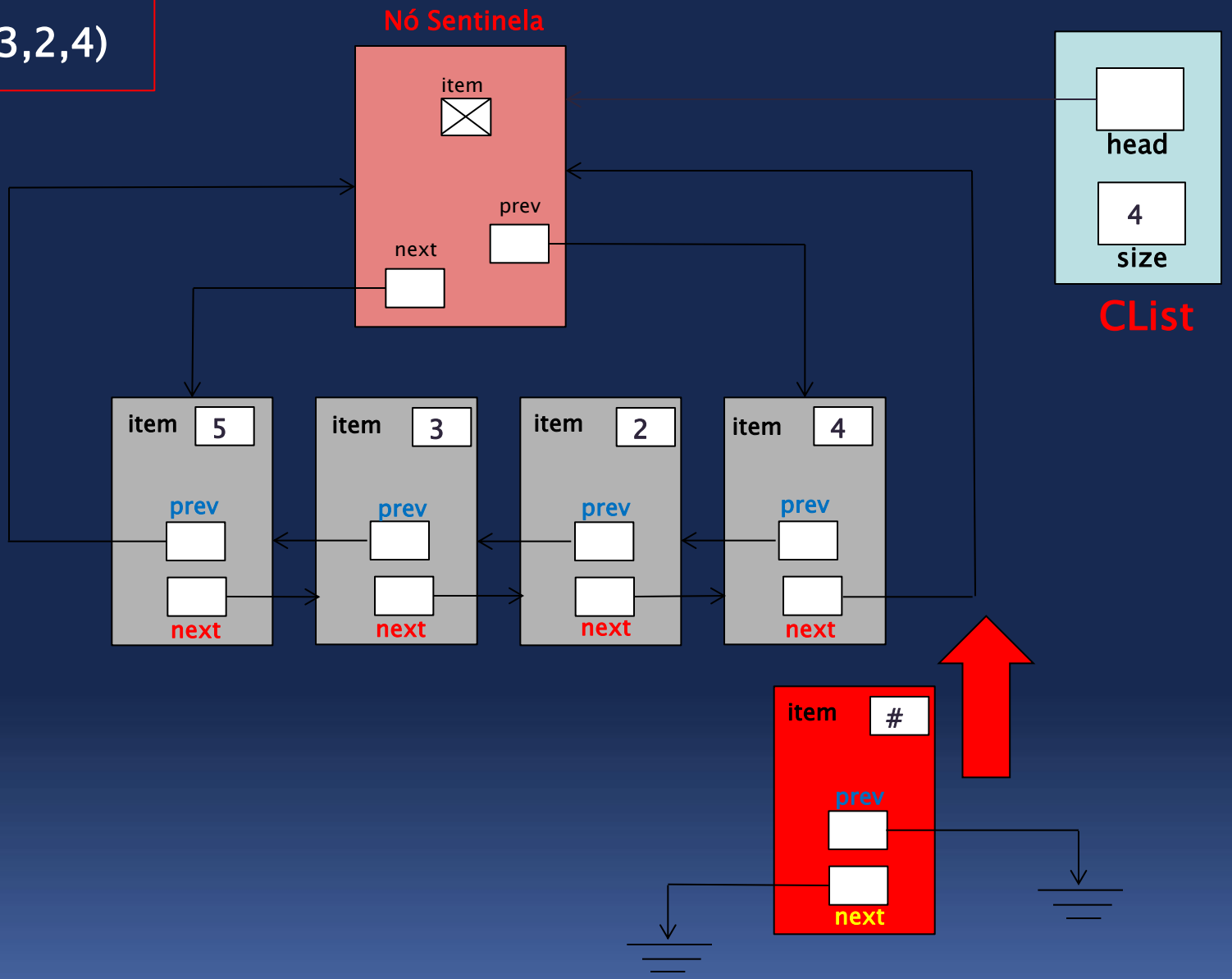
Inserindo nós

```
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();
        }

}
```

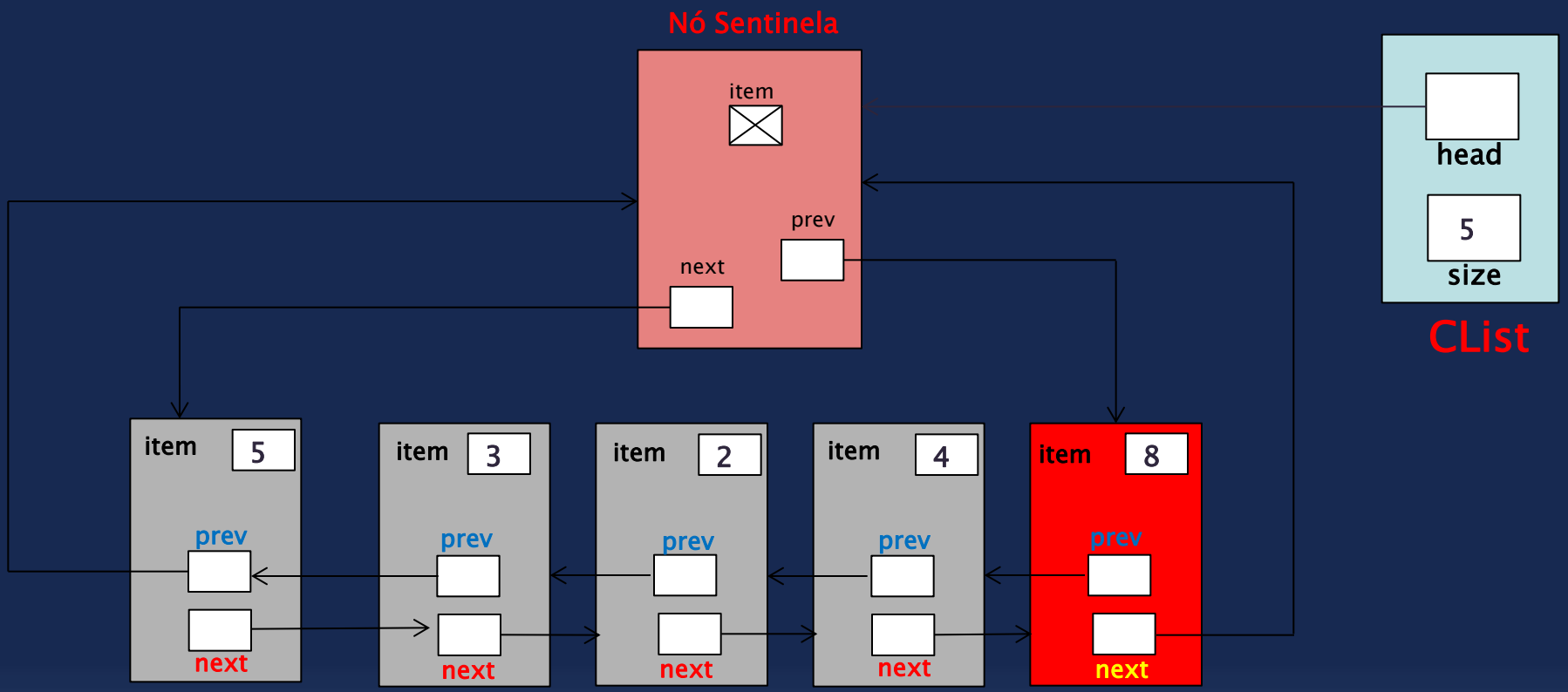
Lista: (8,5,3,2,4)

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++;

                }
}
```

Inserindo nós

```
package maua;

public class Test_CList {

        public static void main(String[] args) {

                CList listaCircular = new CList();

                listaCircular.imprimeFirst();

                listaCircular.imprimeLast();

                listaCircular.insereLast(5);
                listaCircular.imprimeFirst();
                listaCircular.imprimeLast();

                listaCircular.insereLast(3);
                listaCircular.imprimeFirst();
                listaCircular.imprimeLast();

                listaCircular.insereLast(2);
                listaCircular.imprimeFirst();
                listaCircular.imprimeLast();

                listaCircular.insereLast(4);
                listaCircular.imprimeFirst();
                listaCircular.imprimeLast();

                listaCircular.insereLast(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
```

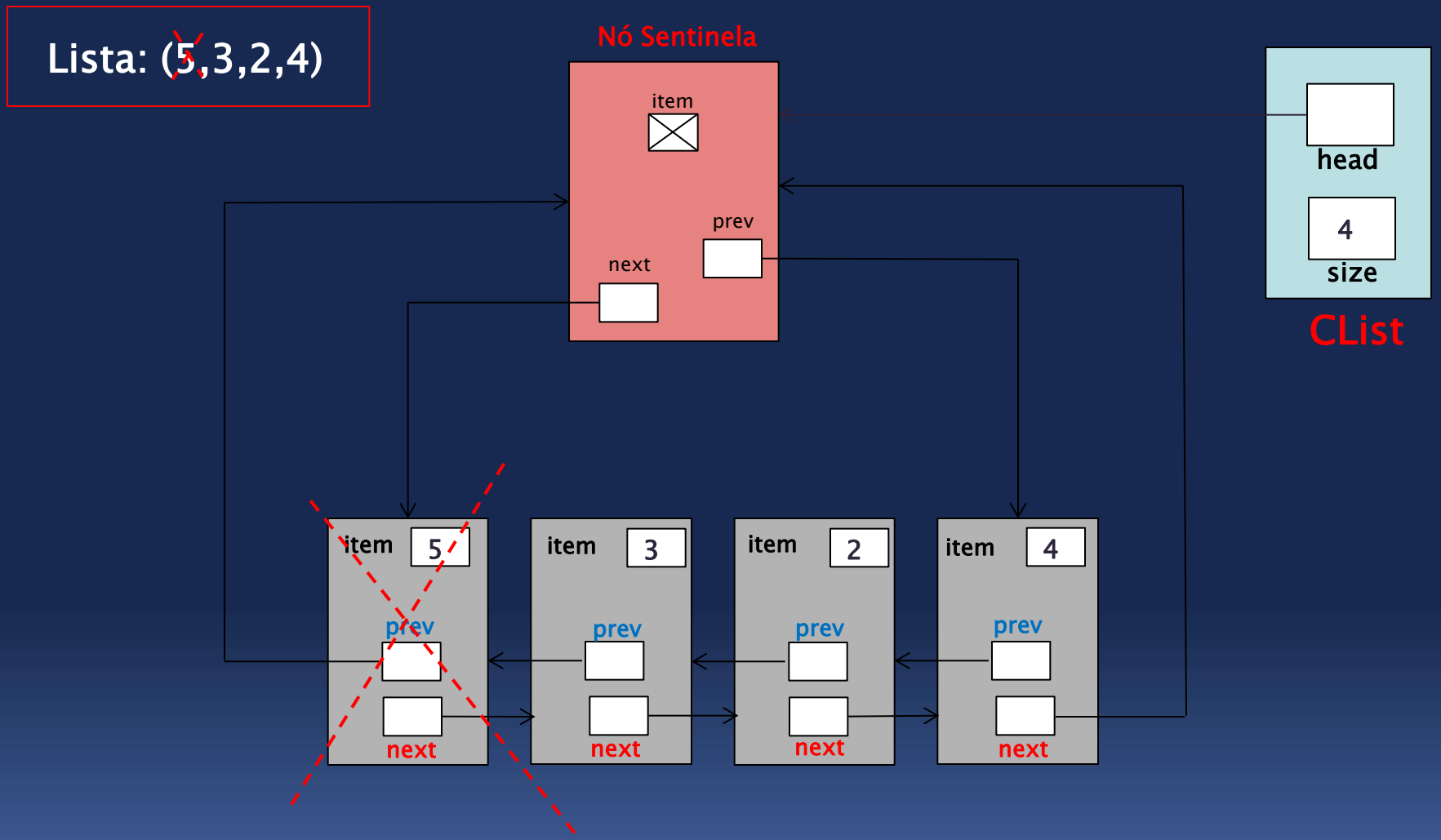
Imprimindo nós da lista

```
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



Função deleteFirst() - Pseucódigo

```
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()

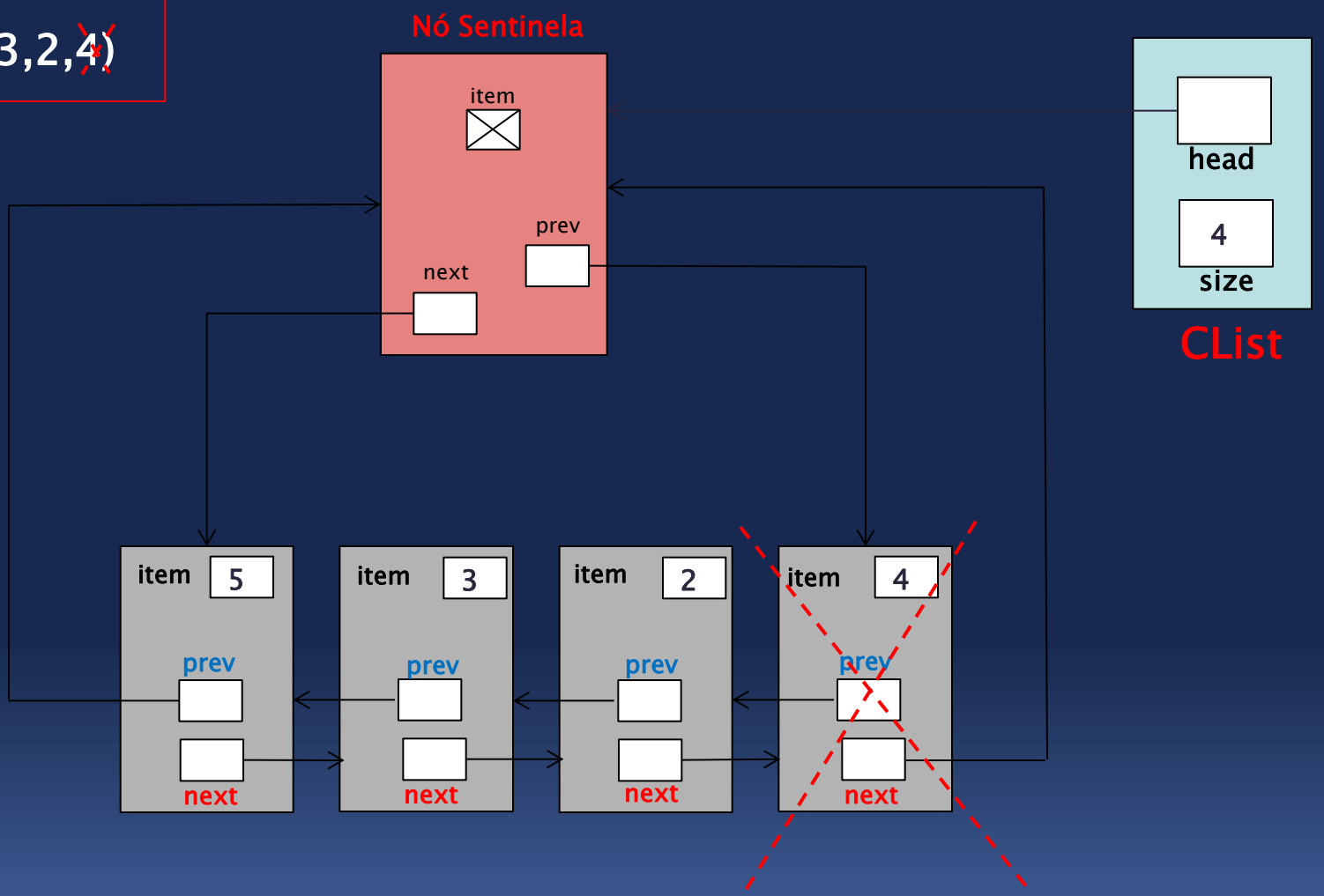
```
public void deleteFirst() {
        if (this.size == 0)
                System.out.println("Deleção inválida... Lista Vazia...");
        else {
                if (this.size == 1) {
                        this.head.next = this.head;
                        this.head.prev = this.head;
         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

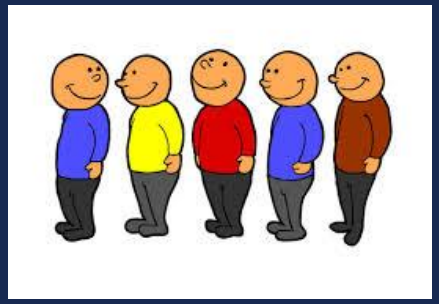
```
deleteLast()

        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()

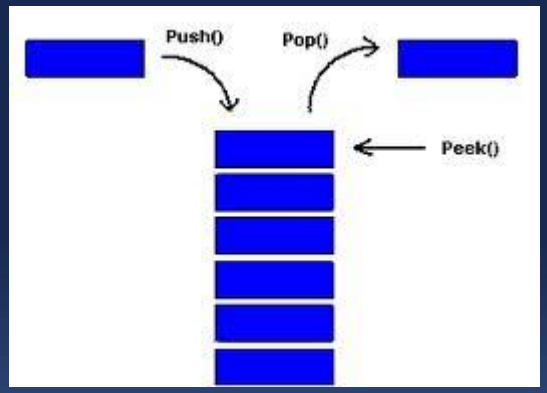
```
public void deleteLast() {
        if (this.size == 0)
                System.out.println("Deleção inválida... Lista Vazia...");
        else {
                if (this.size == 1) {
                        this.head.next = this.head;
                        this.head.prev = this.head;
                        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.insereLast(5);
           listaCircular.insereLast(3);
           listaCircular.insereLast(2);
           listaCircular.insereLast(4);
           listaCircular.insereLast(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();

   }

}
```

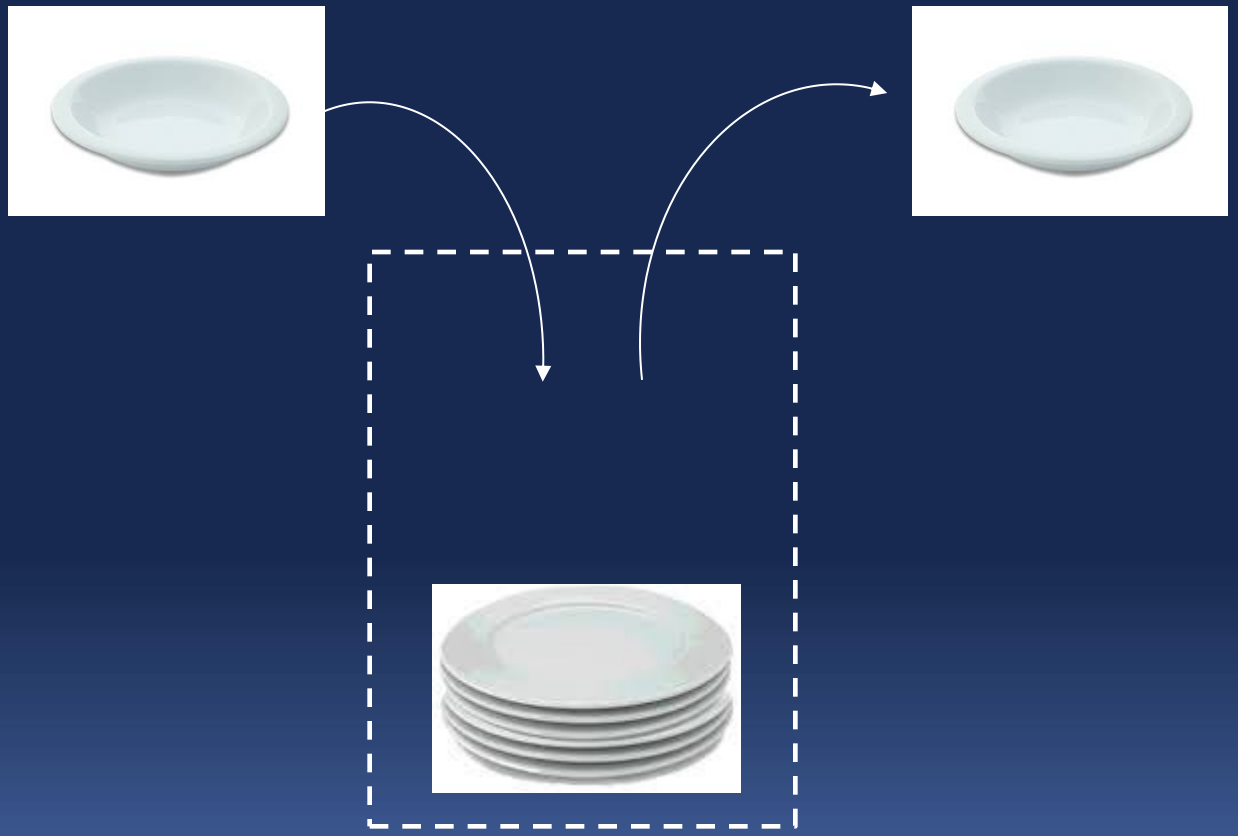


Pilhas e Filas



Pilha

- Estrutura de Dados que implementa uma lista **LIFO** (Last Input First Output).



Fila

- Estrutura de Dados que implementa uma lista **FIFO** (First In, First Out).