

Unidade 10 – Análise de Algoritmos com Estruturas de Dados Lineares

Parte 2



Como vimos, com arrays o esforço computacional para se inserir itens terá tempo proporcional ao tamanho do array...

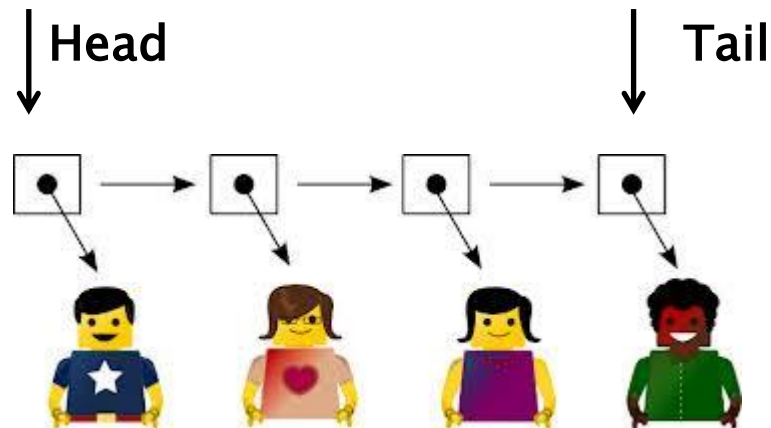


Será que há algum modo de se implementar listas de forma mais eficiente ?



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.
- O primeiro e último nó são chamados **HEAD** e **TAIL** respectivamente.



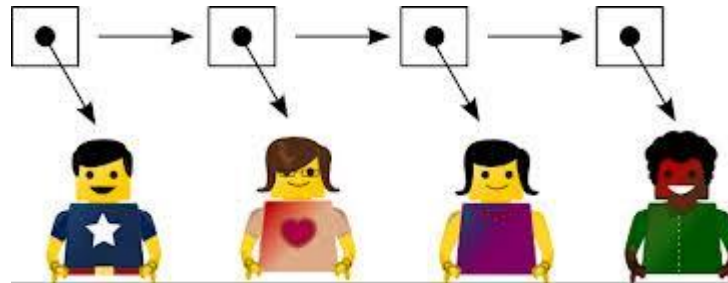
Vantagens sobre listas implementadas com arrays

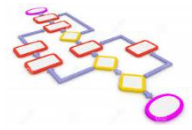
- A inserção de um item no meio da lista leva **tempo constante**, caso você tenha a referência ao prévio nó.
- Listas ligadas podem crescer até o limite de memória oferecido pela máquina virtual.



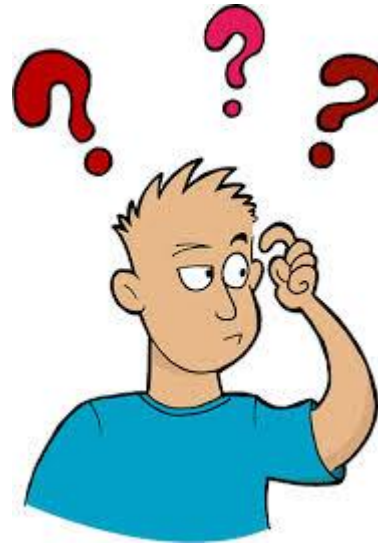
Desvantagens sobre listas implementadas com arrays

- 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).



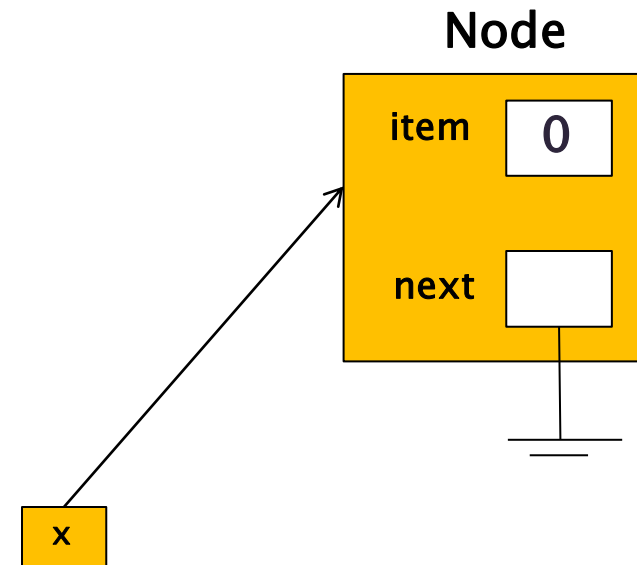


Considerando que a lista ligada é uma lista de nós,
como implementar um nó da lista ?



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;

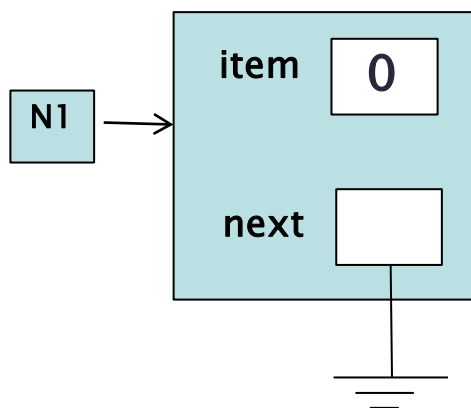
        Node N3;
        N3 = new Node();
        N3.item = 9;

    }
}
```

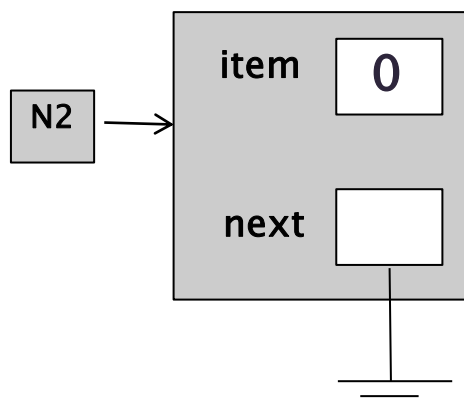


Criação de Nós

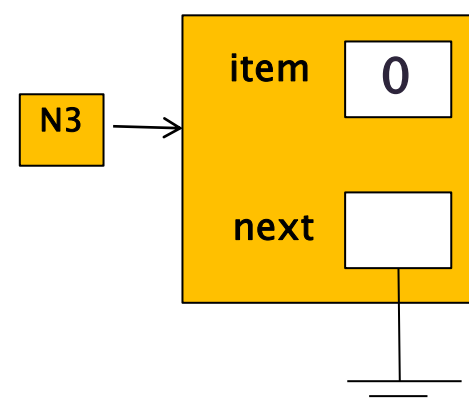
```
Node N1;  
N1 = new Node();
```



```
Node N2;  
N2 = new Node();
```

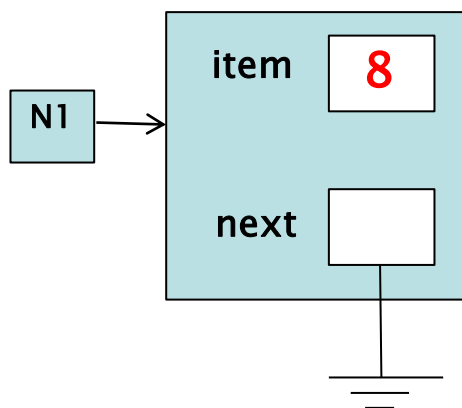


```
Node N3;  
N3 = new Node();
```

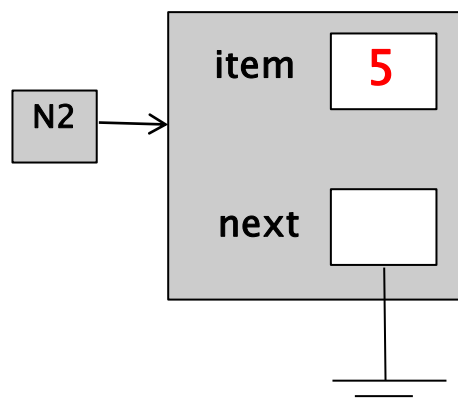


Armazenando valores nos Nós

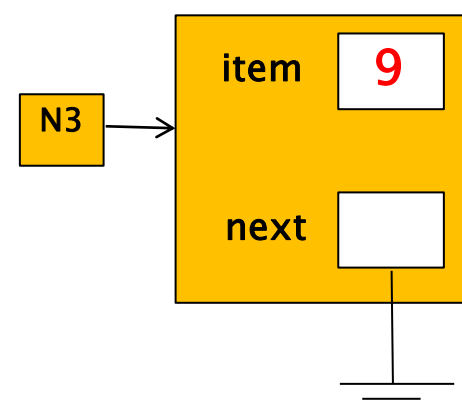
`N1.item = 8;`



`N2.item = 5;`

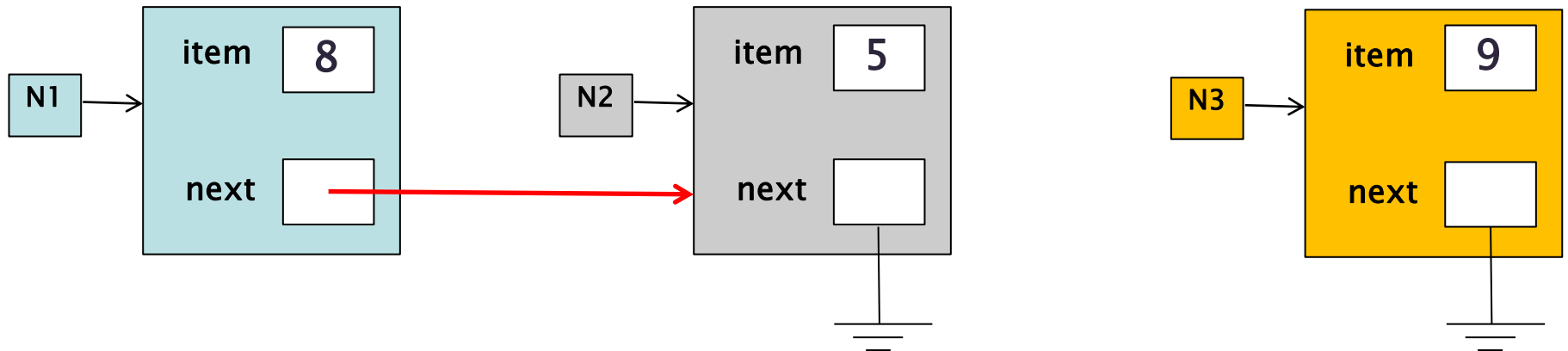


`N3.item = 9;`



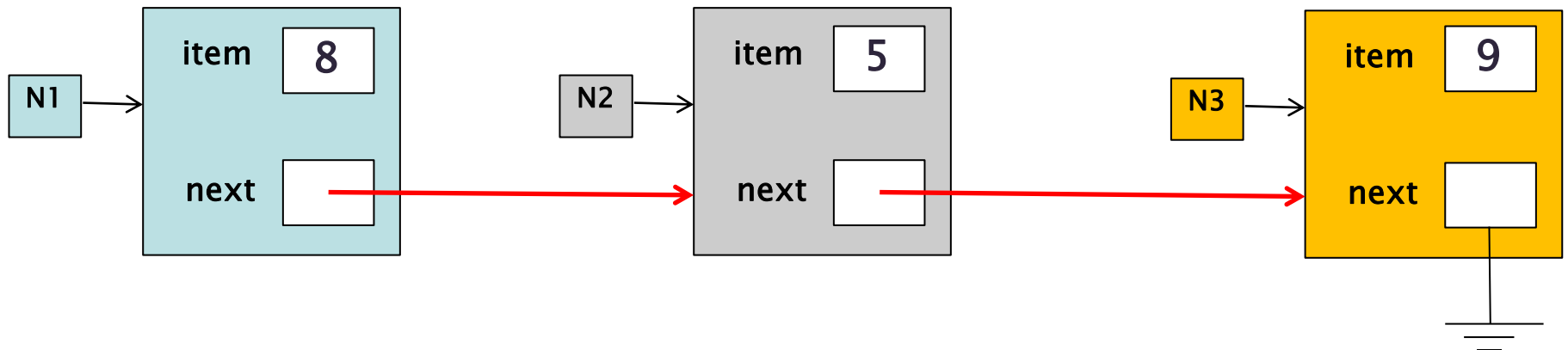
Encadeamento de Nós

N1.next = N2;



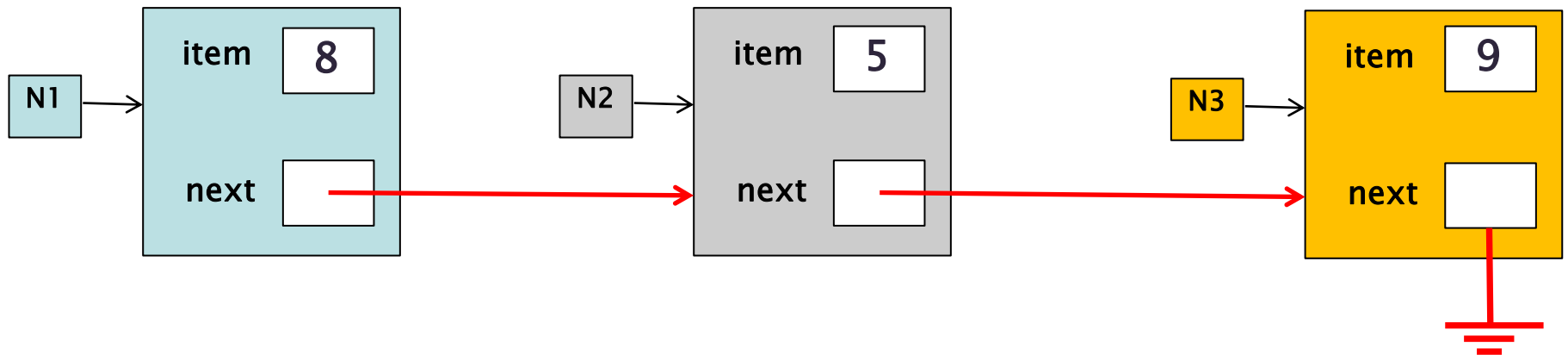
Encadeamento de Nós

N2.next = N3;



Encadeamento de Nós

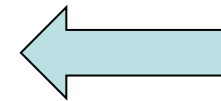
N3.next = null;



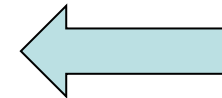
Tipo Abstrato de Dados

Node

int Node	item next
<p>Node()</p> <p>Node(int)</p> <p>Imprime_Lista()</p> <p>Insert_Item(int)</p> <p>Deleta_Proximo_Item()</p> <p>Altera_Item(int)</p>	



Dados



Operações



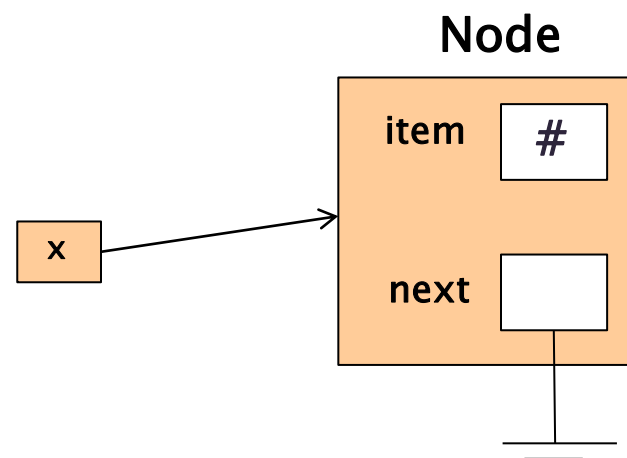
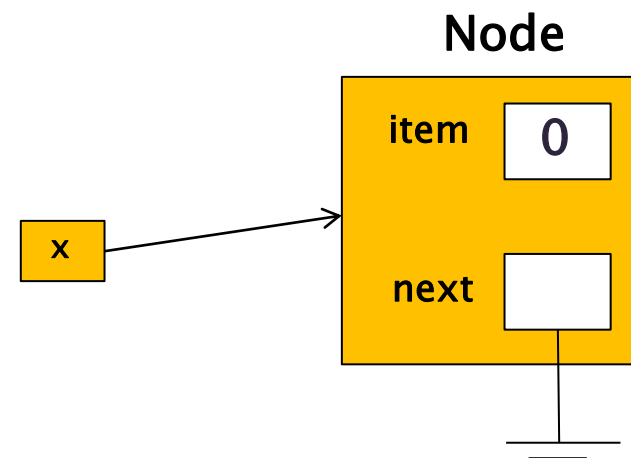
Implementação

```
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;
    }
}
```



Lista Ligada – Encadeamento de Nós

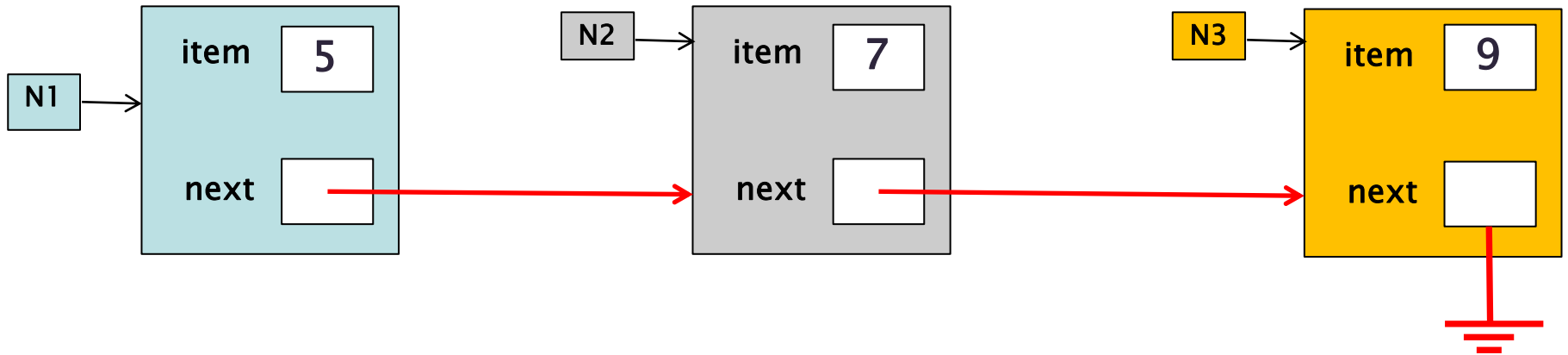
```
Node N1 = new Node(5);
```

```
Node N2 = new Node(7);
```

```
Node N3 = new Node(9);
```

```
N1.next = N2;
```

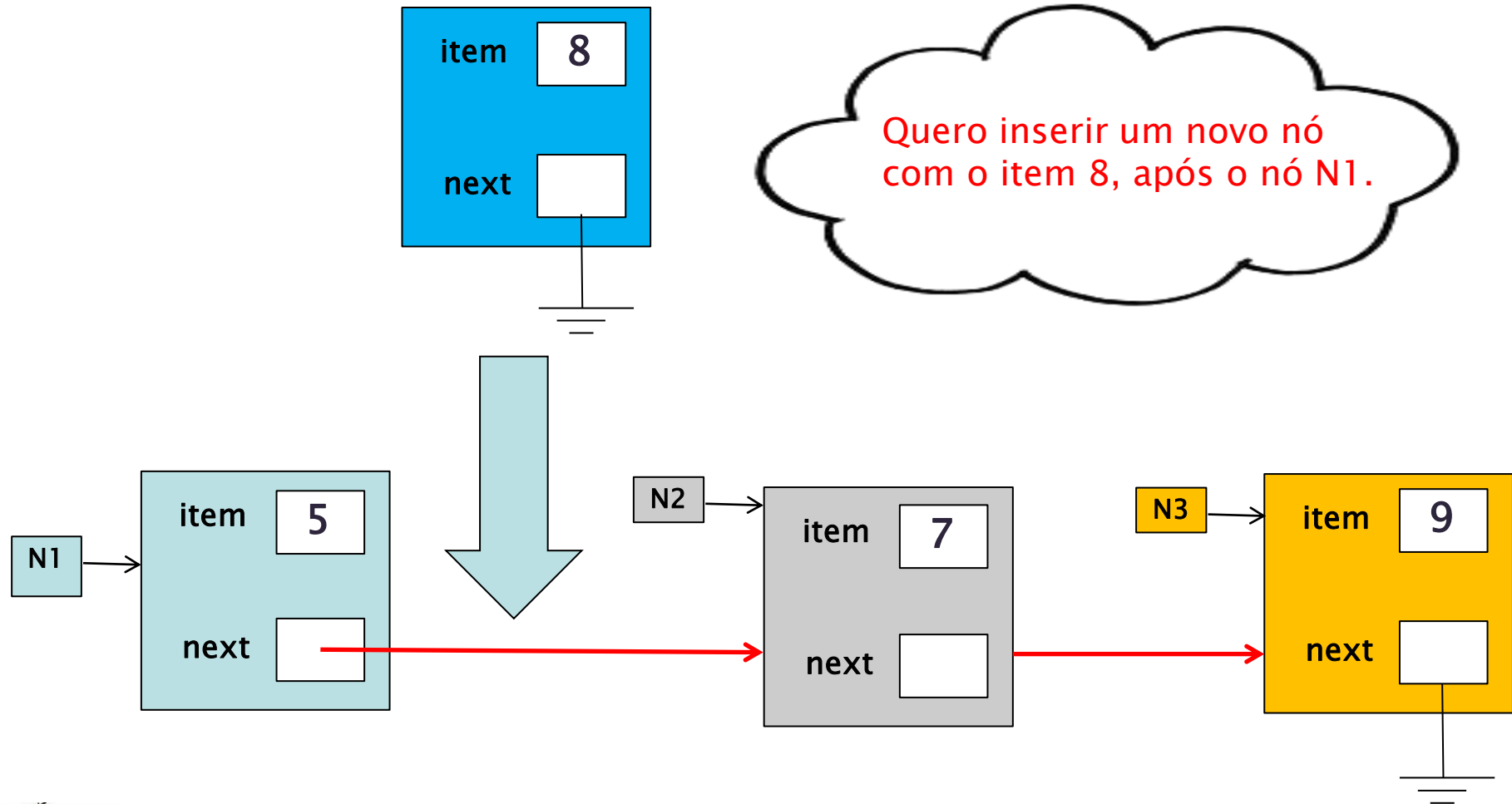
```
N2.next = N3;
```



Como inserir um novo nó ?



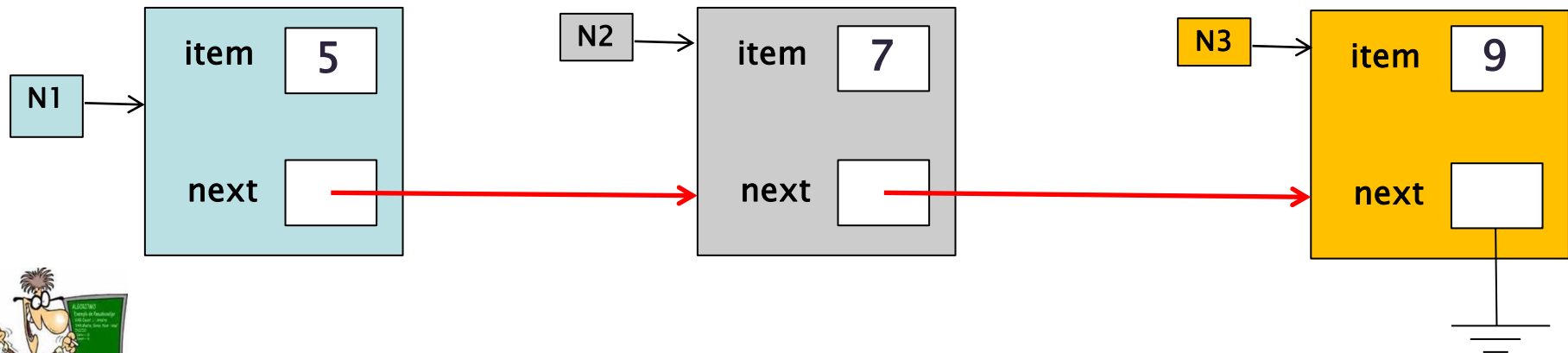
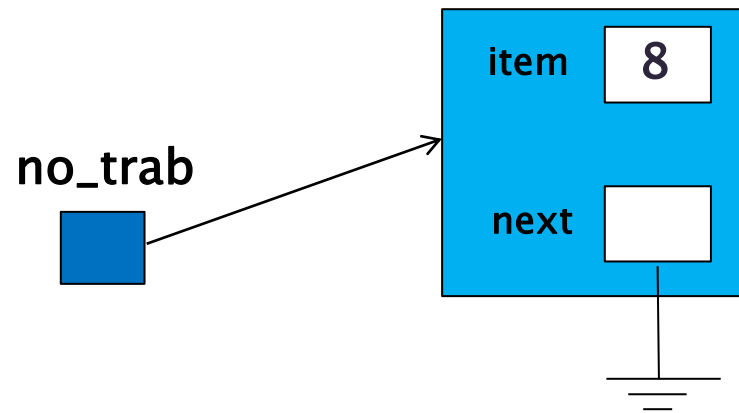
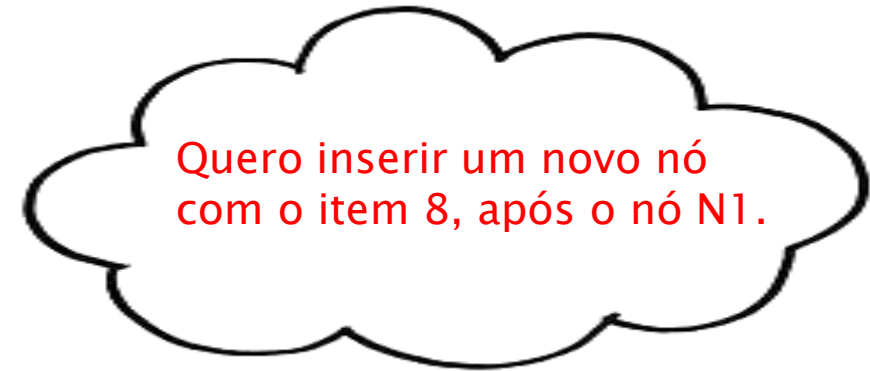
Operação de insert



1

Operação de insert

PASSO 1: Cria-se um novo nó, com o item 8

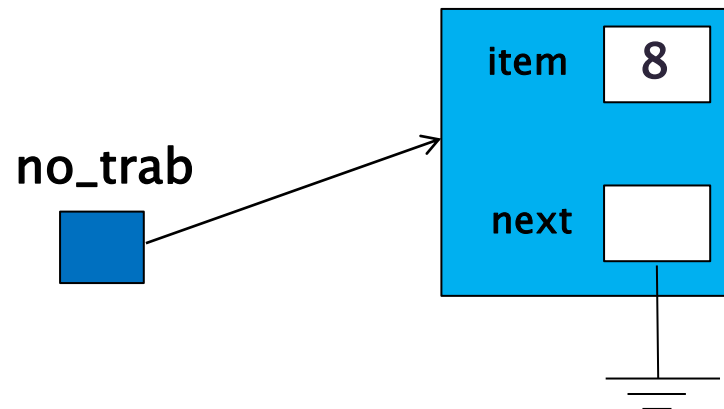


Operação de insert

1

PASSO 1: Cria-se um novo nó, com o item 8

```
Node no_trab = new Node(8);
```



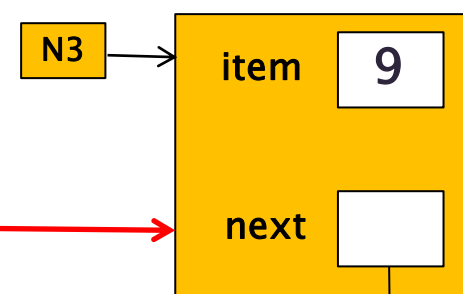
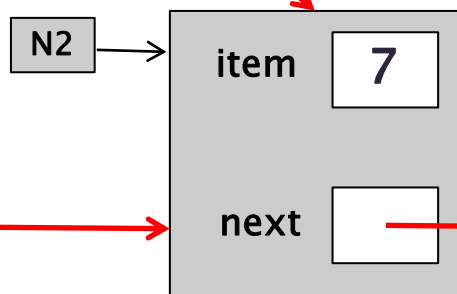
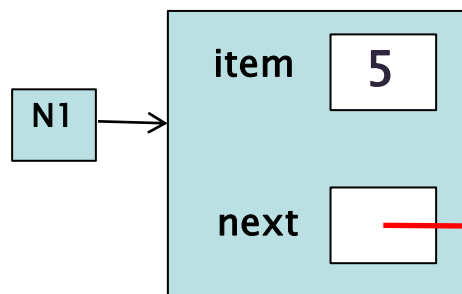
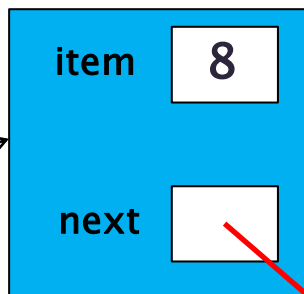
2

Operação de insert

PASSO 2: Novo nó aponta para próximo nó apontado por N1

Quero inserir um novo nó com o item 8, após o nó N1.

no_trab

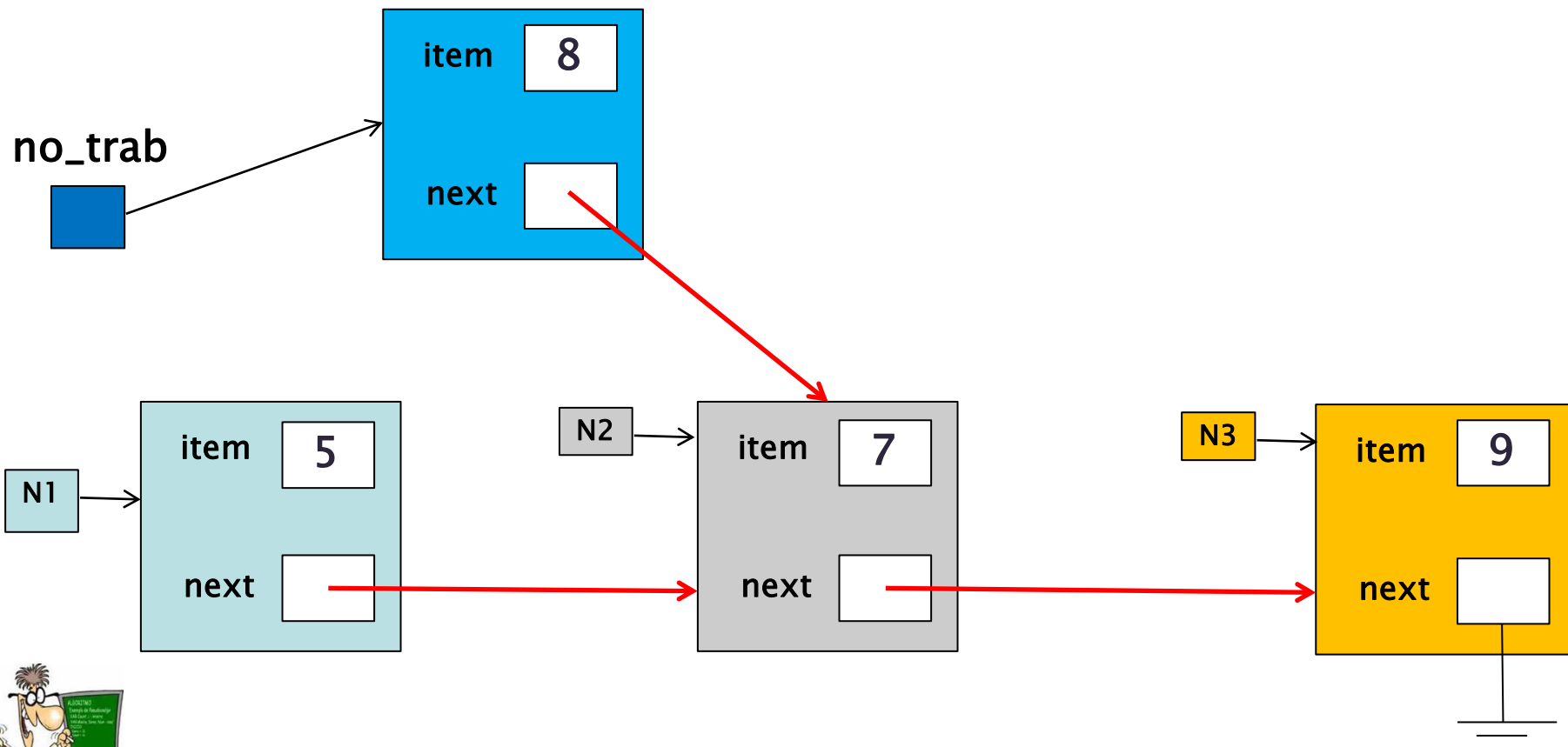


2

Operação de insert

PASSO 2: Novo nó aponta para próximo nó apontado por N1

```
no_trab.next = this.next ;
```

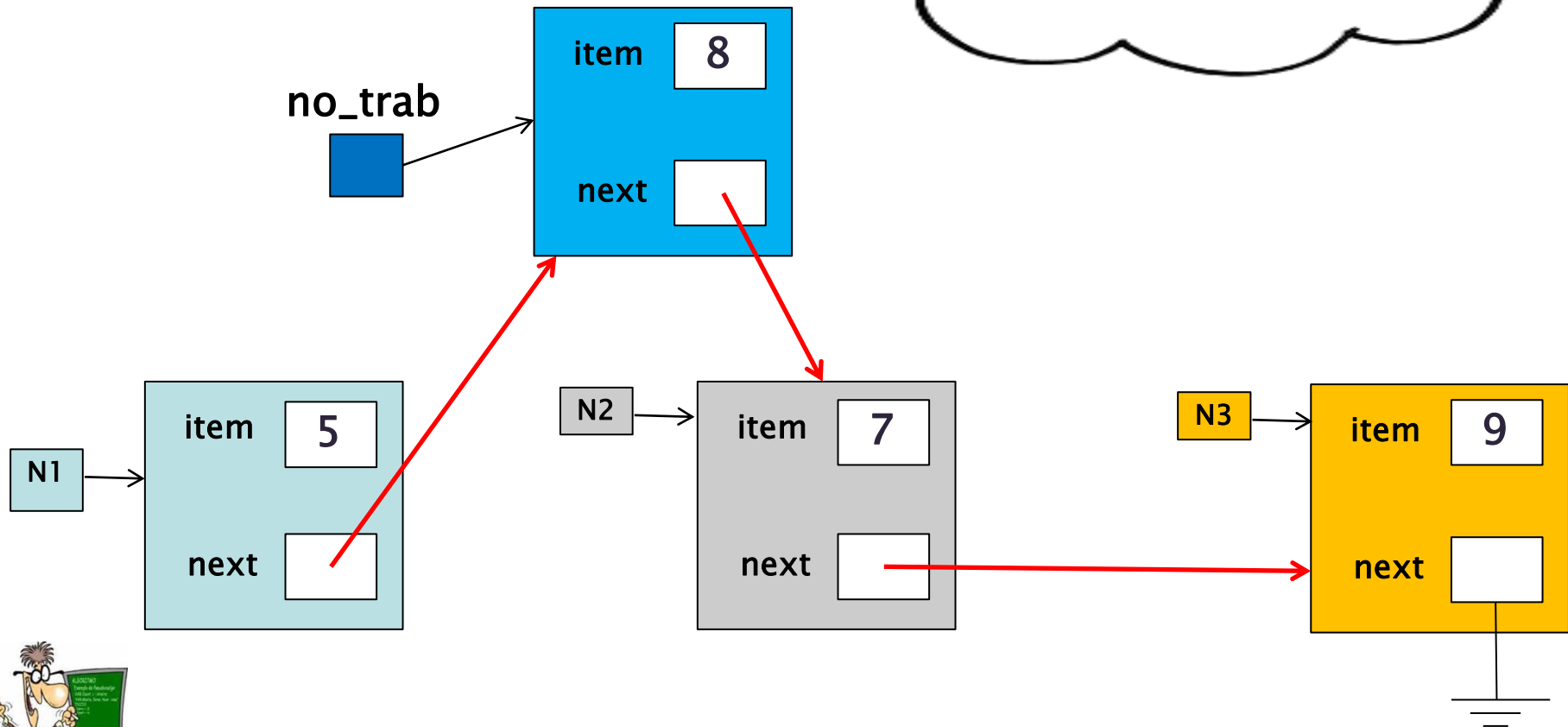


3

Operação de insert

PASSO 3: Nó N1 aponta para o novo nó para o novo nó

Quero inserir um novo nó com o item 8, após o nó N1.



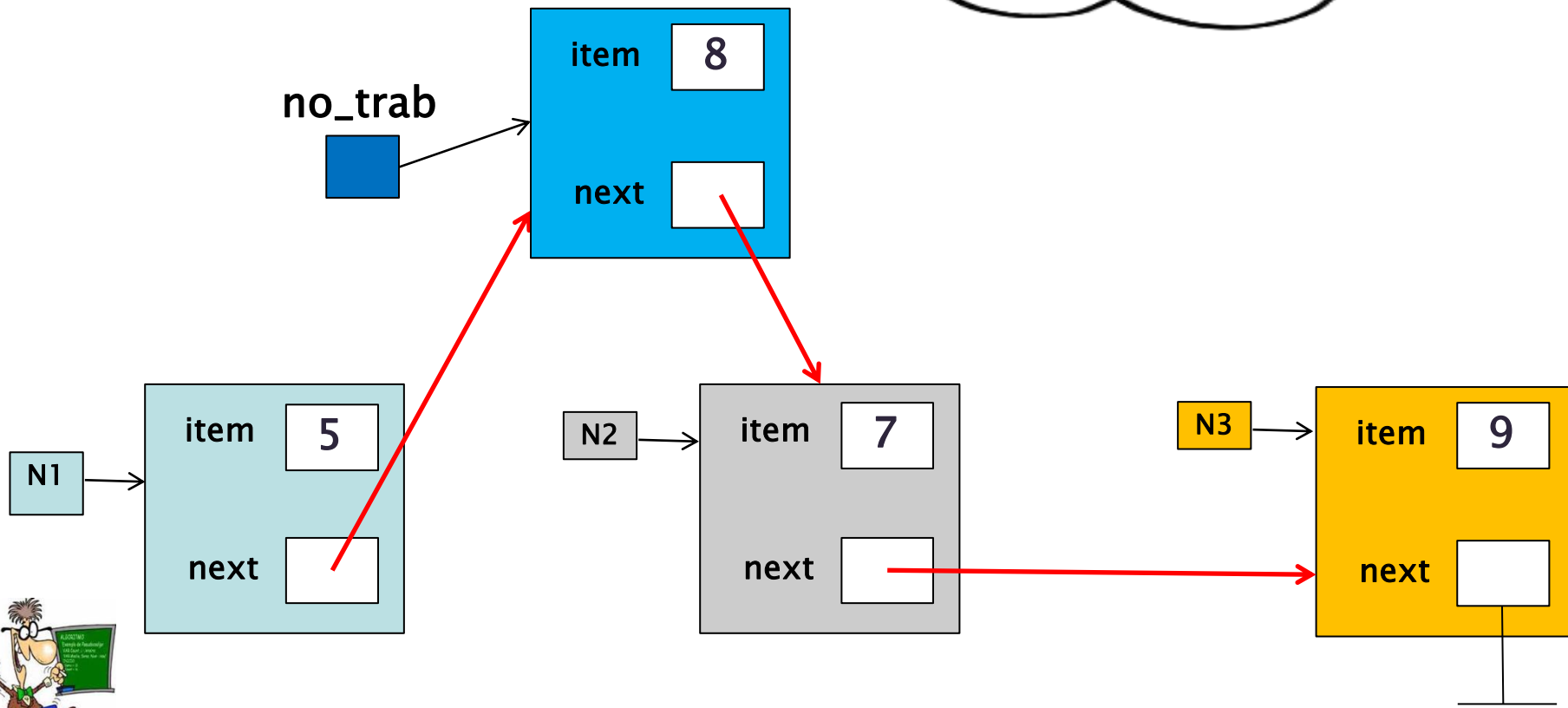
3

Operação de insert

PASSO 3: Nó N1 aponta para o novo nó

```
this.next = no_trab;
```

Quero inserir um novo nó com o item 8, após o nó N1.



Operação de insert

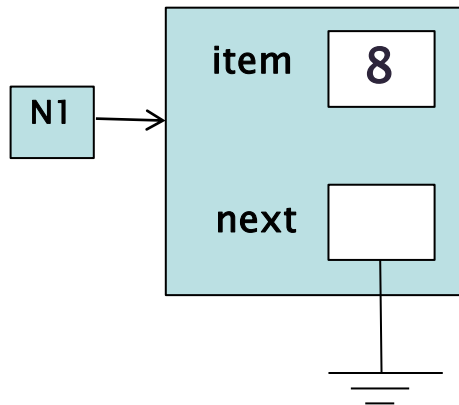
```
public void Insert_Item(int item) {  
    Node no_trab = new Node(item); ①  
    no_trab.next = this.next ;      ②  
    this.next = no_trab;            ③  
}
```

◆ Ordem de complexidade: $O(1)$.

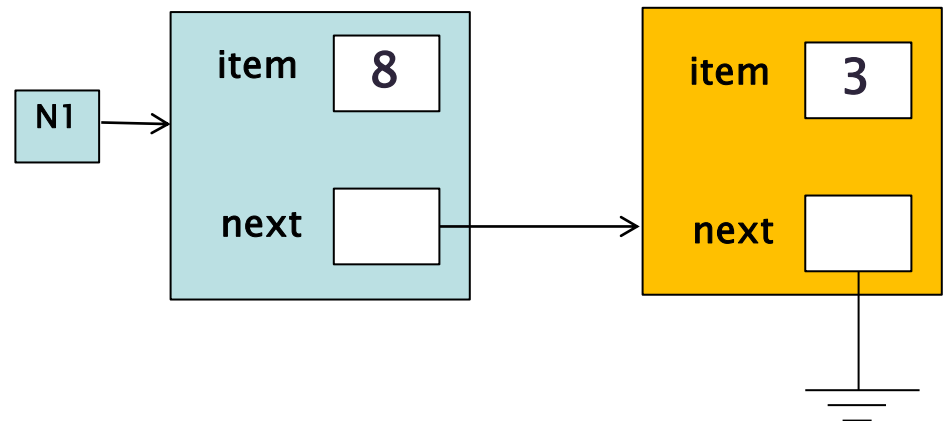


Operação de insert

Node N1 = new Node(8);



N1.Insert_Item(3);



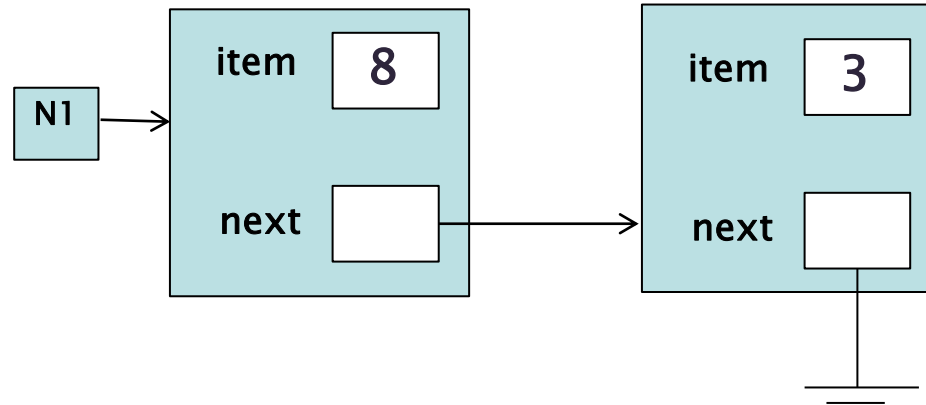
O novo nó (com o item 3) será inserido após o nó corrente (N4)



Lista Ligada – Implementação

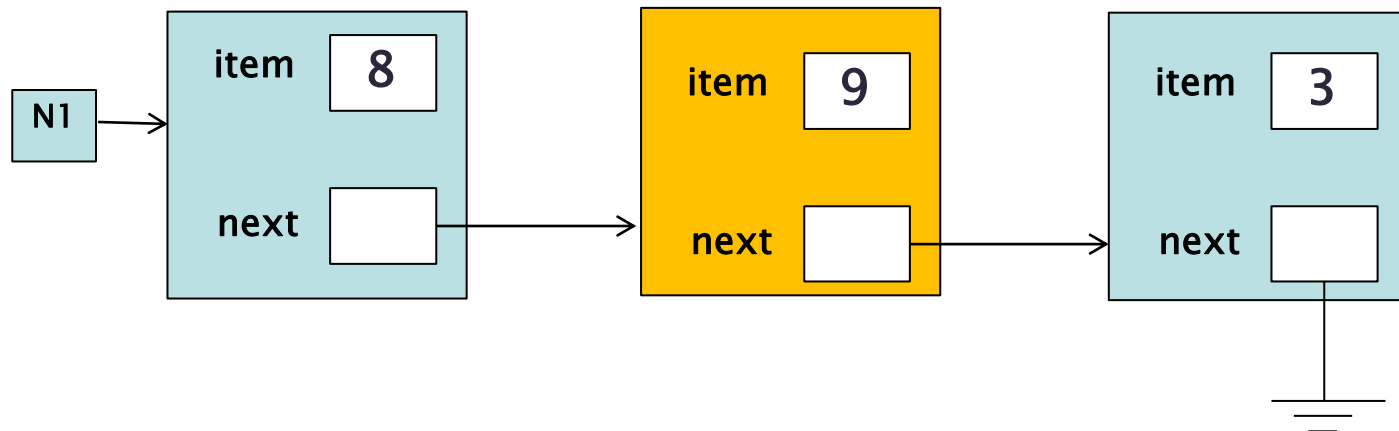
`N1.Insert_Item(9);`

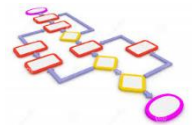
Antes



`N1.Insert_Item(9);`

Após





Como imprimir os nós da Lista ?



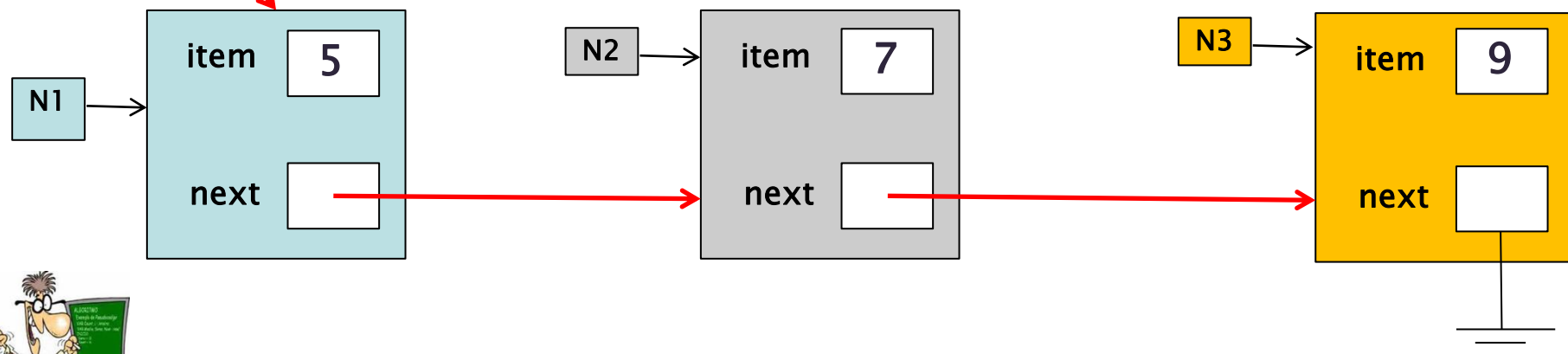
1

Imprimindo os nós da lista

PASSO 1: Declara-se um novo nó (de trabalho) apontando para o nó de início (this)

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab



1

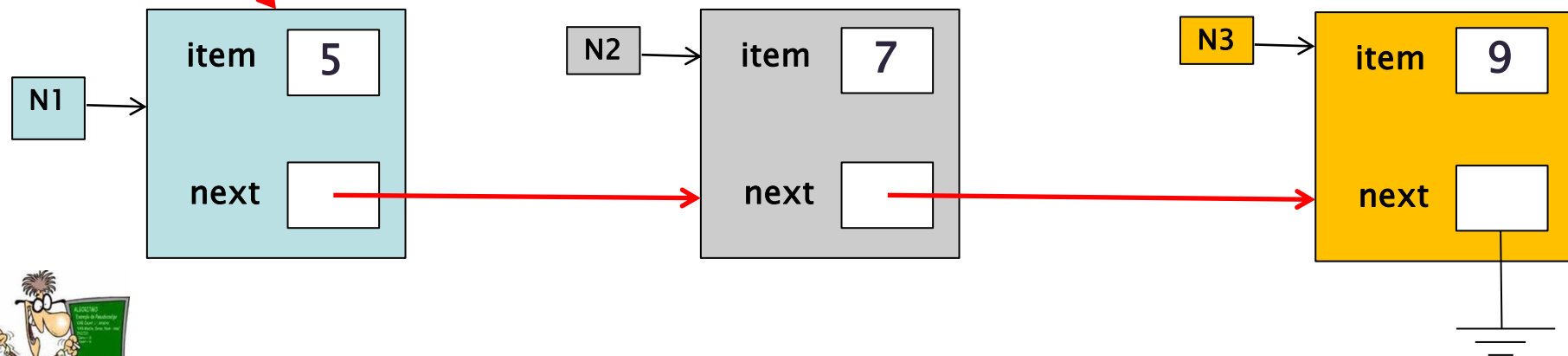
Imprimindo os nós da lista

PASSO 1: Declara-se um novo nó (de trabalho) apontando para o nó de início (this)

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab

```
Node no_trab = this;
```



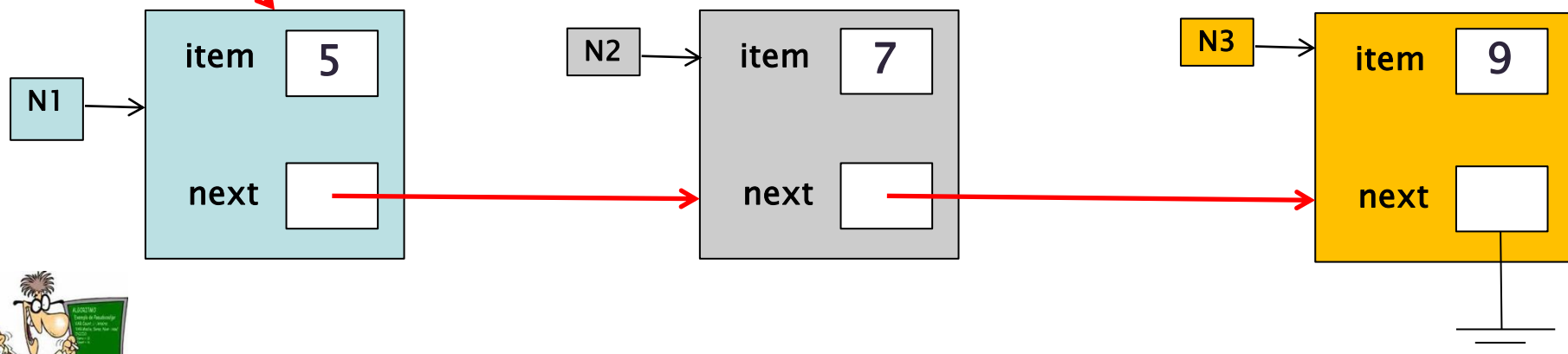
2

Imprimindo os nós da lista

PASSO 2: Executa-se um looping, verificando se **no_trab** aponta para o final da lista. Para cada iteração, imprime-se o valor do nó corrente da lista

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab



2

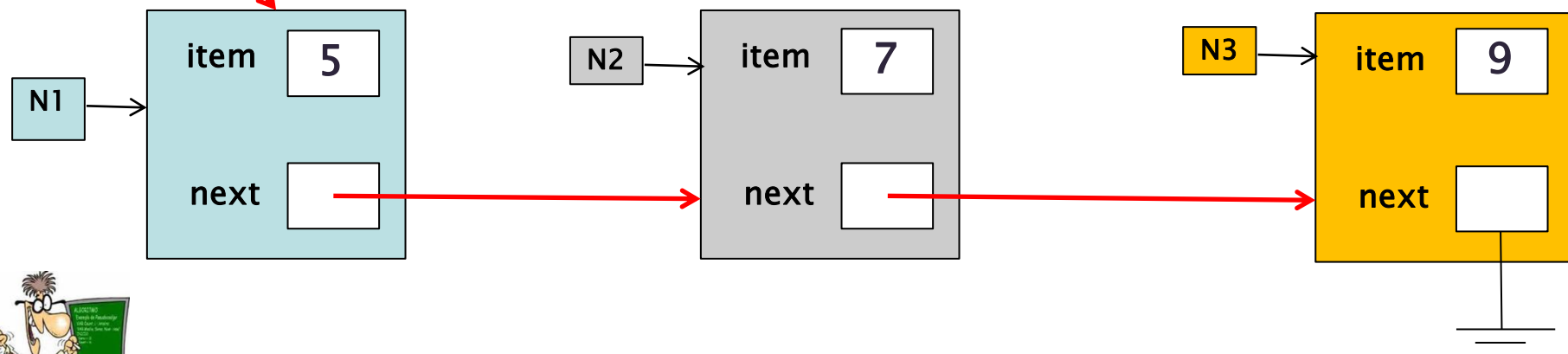
Imprimindo os nós da lista

PASSO 2: Executa-se um looping, verificando se **no_trab** aponta para o final da lista. Para cada iteração, imprime-se o valor do nó corrente da lista

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab

```
while (no_trab != null ) {  
    System.out.print("    " + no_trab.item);  
    no_trab = no_trab.next ;  
}
```



2

Imprimindo os nós da lista

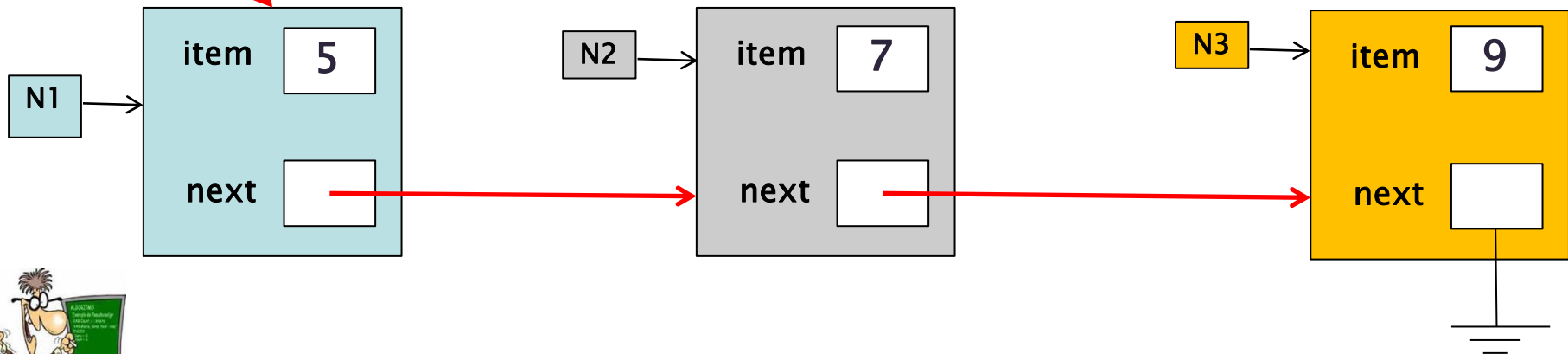
PASSO 2: Looping de busca

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab



Imprime 5



2

Imprimindo os nós da lista

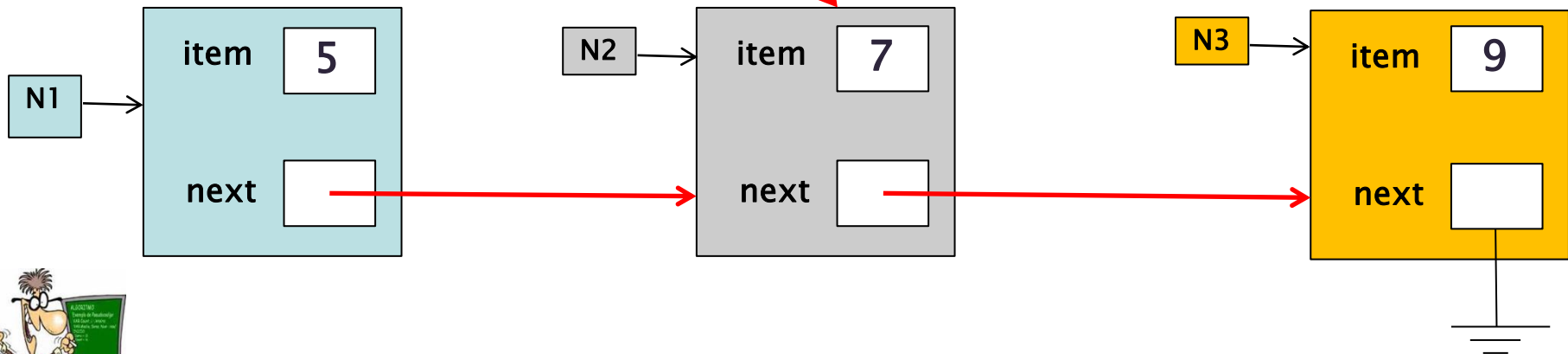
PASSO 2: Looping de busca

Quero imprimir os nós da lista, a partir de um determinado nó (N1)

no_trab



Imprime 7

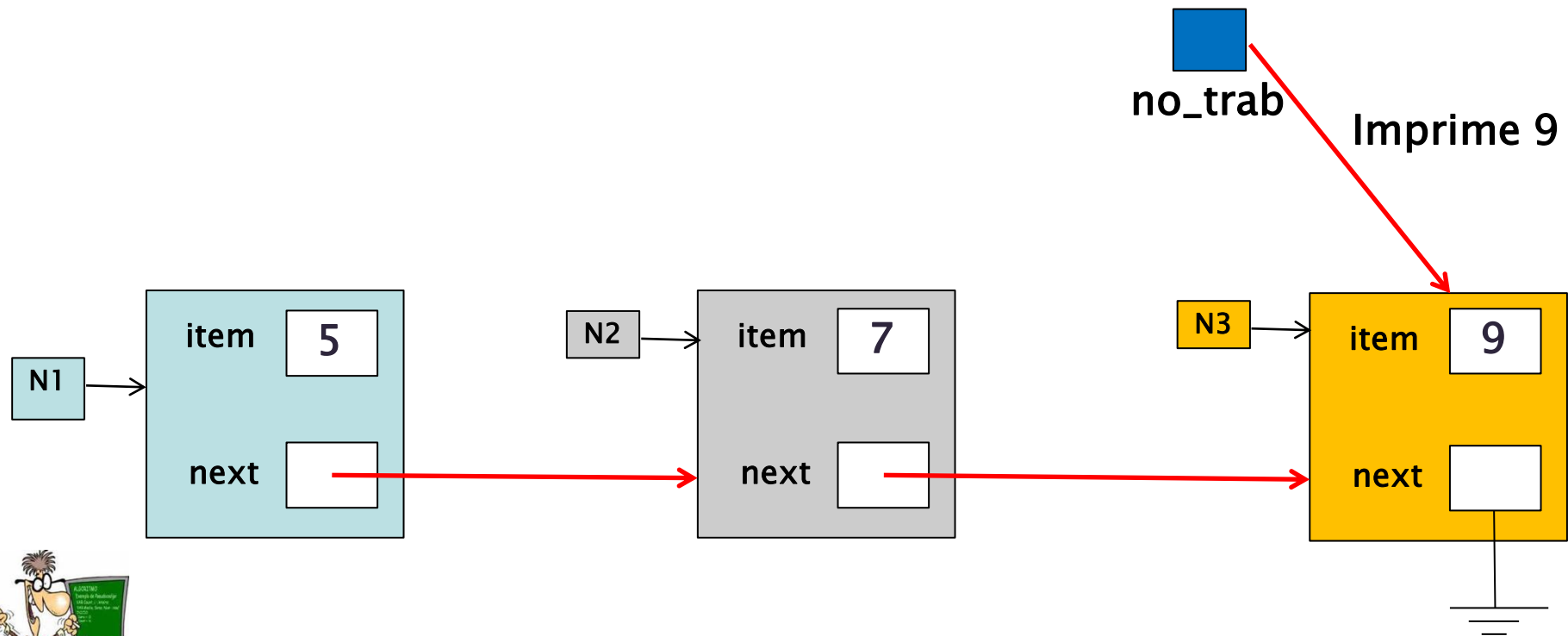


2

Imprimindo os nós da lista

PASSO 2: Looping de busca

Quero imprimir os nós da lista, a partir de um determinado nó (N1)



2

Imprimindo os nós da lista

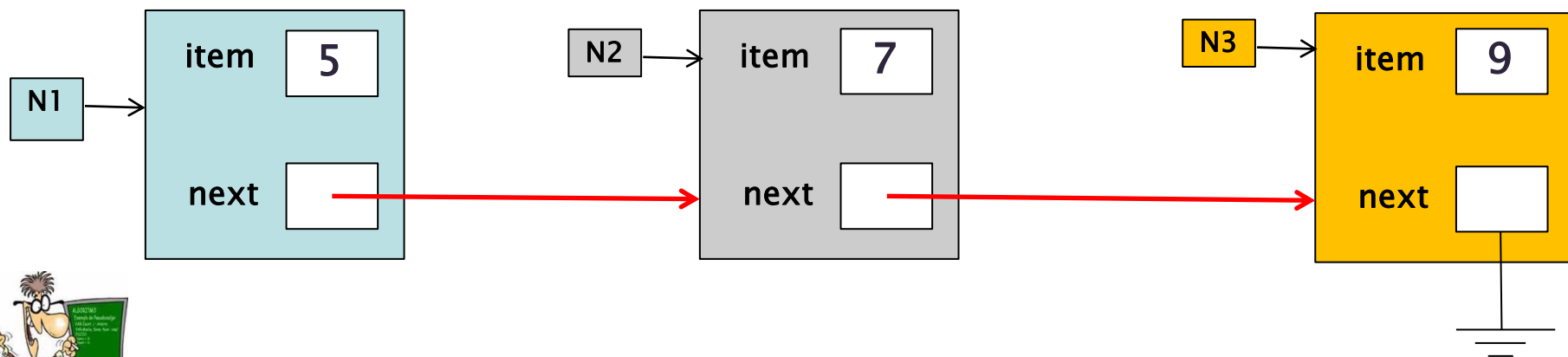
PASSO 2: Looping de busca

THE END!

Fim do Looping !!!

no_trab

Quero imprimir os nós da lista, a partir de um determinado nó (N1)



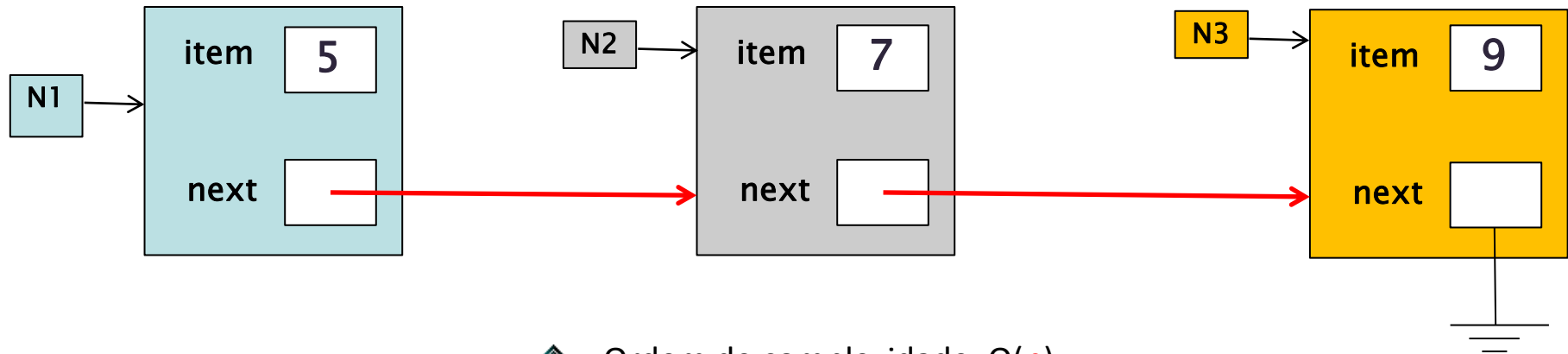
Imprimindo os nós da lista

```

public void Imprime_Lista() {
    Node no_trab = this;
    System.out.print("Lista: ");
    while (no_trab != null ) {
        System.out.print("    " + no_trab.item);
        no_trab = no_trab.next ;
    }
    System.out.println("");
}

```

Quero imprimir os nós da lista, a partir de um determinado nó (N1)



◆ Ordem de complexidade: $O(n)$.



Imprimindo os nós da lista

```
Node N1 = new Node(5);  
Node N2 = new Node(7);  
Node N3 = new Node(9);
```

```
N1.next = N2;  
N2.next = N3;
```

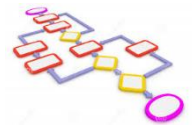
```
N1.Insert_Item(8);
```

```
N1.Imprime_Lista();
```

Quero imprimir os nós da
lista, a partir de um
determinado nó (N1)

Será impresso === > **Lista: 5 8 7 9**



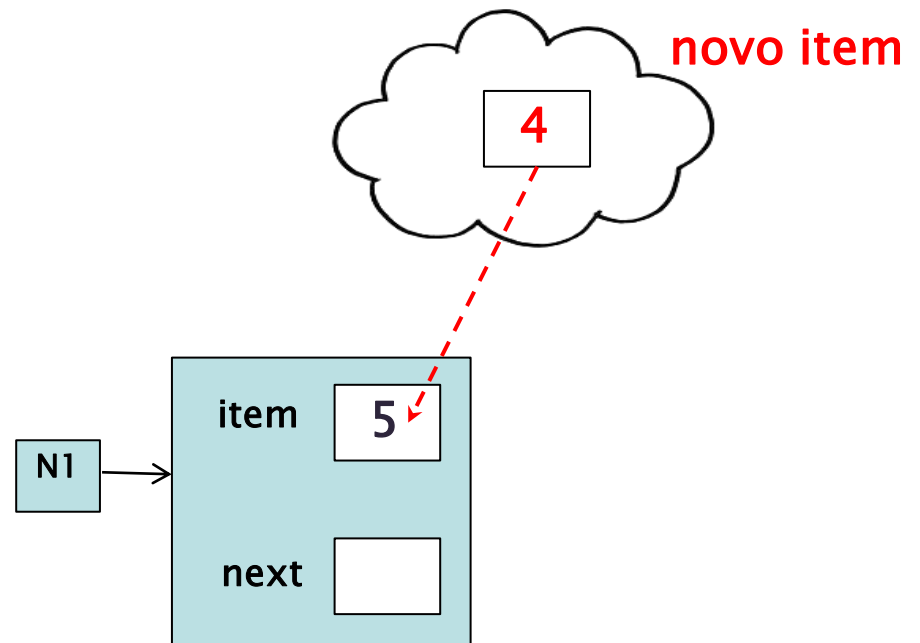


Como alterar o item de um nó ?



Alterando o item de um nó

```
public void Altera_item(int item) {  
    this.item = item;  
}
```



Executando...

```
public static void main(String[] args) {  
  
    Node n1 = new Node(5);  
    Node n2 = new Node(7);  
    Node n3 = new Node(9);  
  
    n1.next = n2;  
    n2.next = n3;  
  
    n1.Insert_Item(8);  
  
    n1.Imprime_Lista();  
  
    n1.Deleta_Proximo_Item();  
    n1.Imprime_Lista();  
  
    n1.Alterar_item(999);  
    n1.Imprime_Lista();  
  
}
```



Resultado da execução

```
Lista:      5      8      7      9
Lista:      5      7      9
Lista:    999      7      9
```



Lista de Objetos

- Referenciam qualquer objeto por meio da declaração de objetos do tipo **Object**.

S denota listas simplesmente ligadas.

```
public class SListNode {  
  
    public Object item;  
    public SListNode next;  
  
}
```



Como vimos, com listas ligadas o esforço computacional para se inserir itens terá tempo constante, uma vez que precisamos apenas implementar o encadeamento de referências (pointers) ...



Mas, será que ainda temos alguns inconvenientes
na implementação de nós encadeados ?

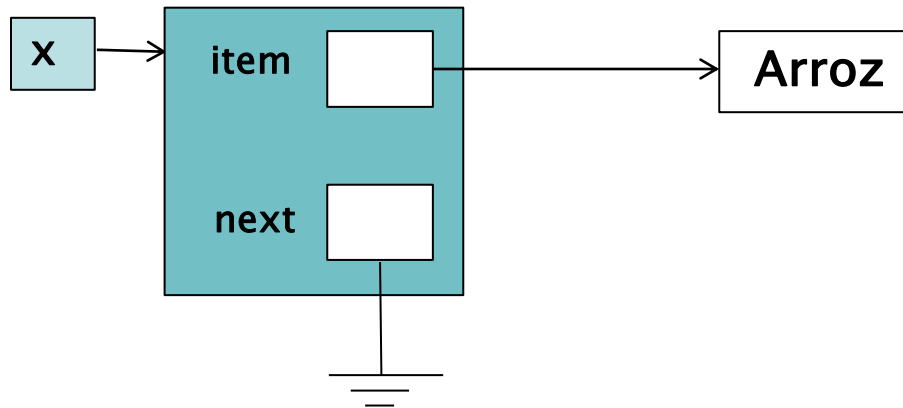


Sim, Listas ligadas por meio de encadeamento de nós ainda nos trazem alguns problemas...



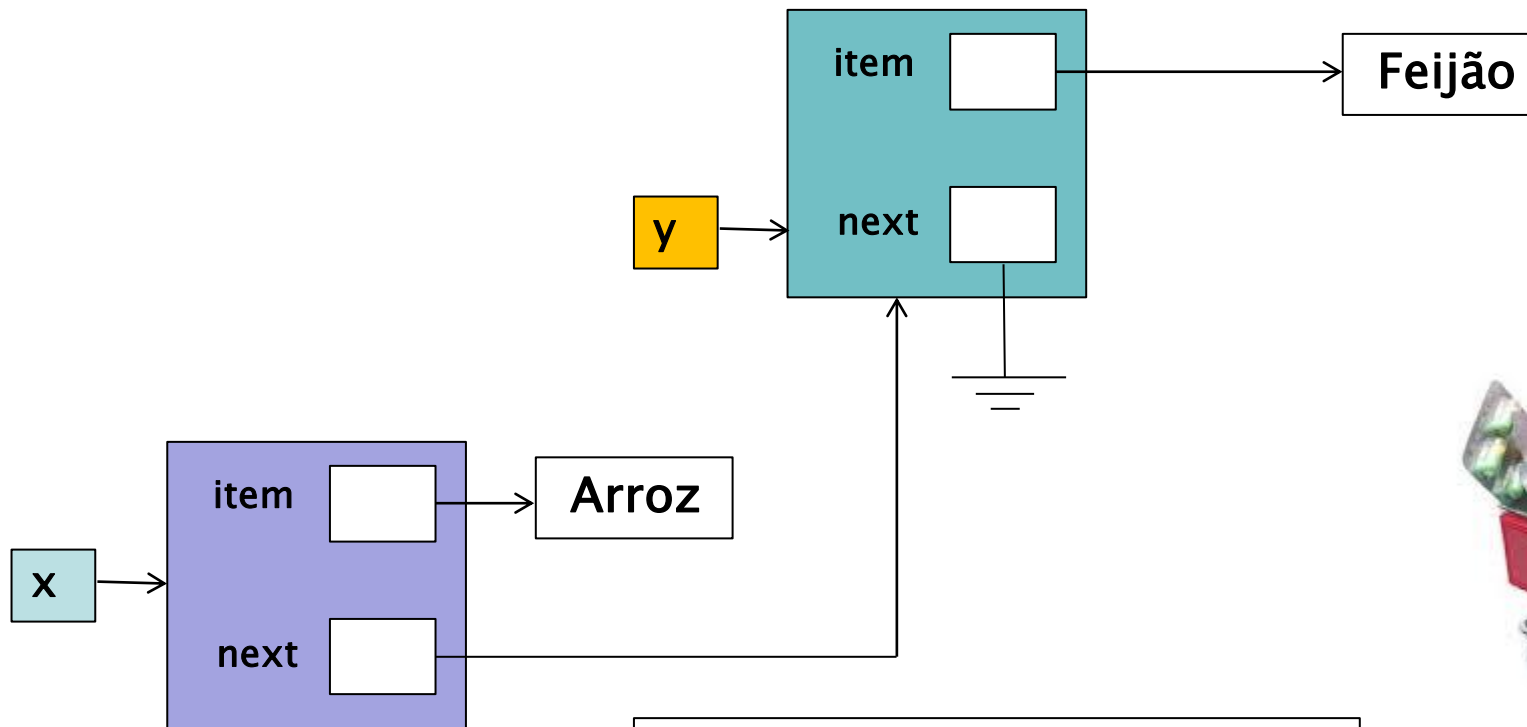
Problema 1: Node

- Seja x uma referência à um nó de uma lista ligada.



Problema 1: Node

```
Node y = new Node("Feijão");  
x.next = y;
```



Quando se acessa a lista por **y**, não se considera o item "Arroz" ...



Problema 2: Node

Como se representa uma **lista vazia** ?



Basta atribuir-se **null** à referência...



```
package maua;
```

```
public class Test_Node {
```

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

```
        Node N1;
```

```
        N1 = null;
```

```
        N1.Imprime_Lista() ; // Run-time error...
```

```
    }
```

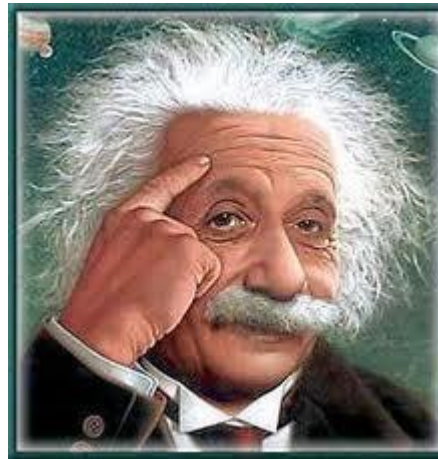
```
}
```

Exception in thread "main"
[java.lang.NullPointerException](#)
at maua.Node.main([Node.java:58](#))

**Sempre haverá erro de Run-time se você
chamar um método em um objeto null. ..**



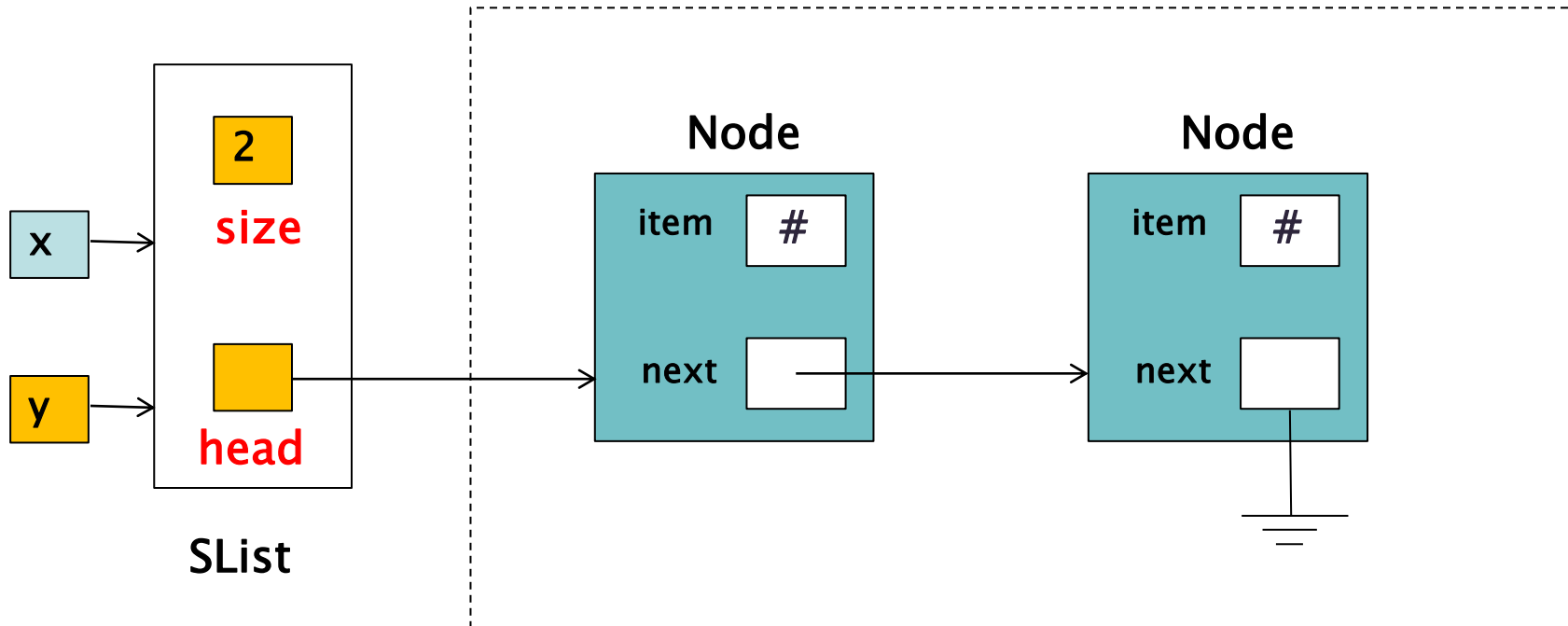
Como então resolver estes problemas ?



Criaremos uma classe separada para manter controle da lista (HEAD e SIZE).



Classe SList

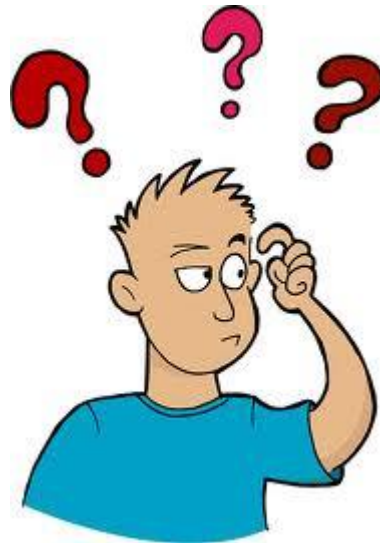


Classe SList

```
package maua;  
  
public class SList {  
  
    Node head;  
    int size;  
  
    public SList() {  
        this.head = null;  
        this.size = 0;  
    }  
}
```

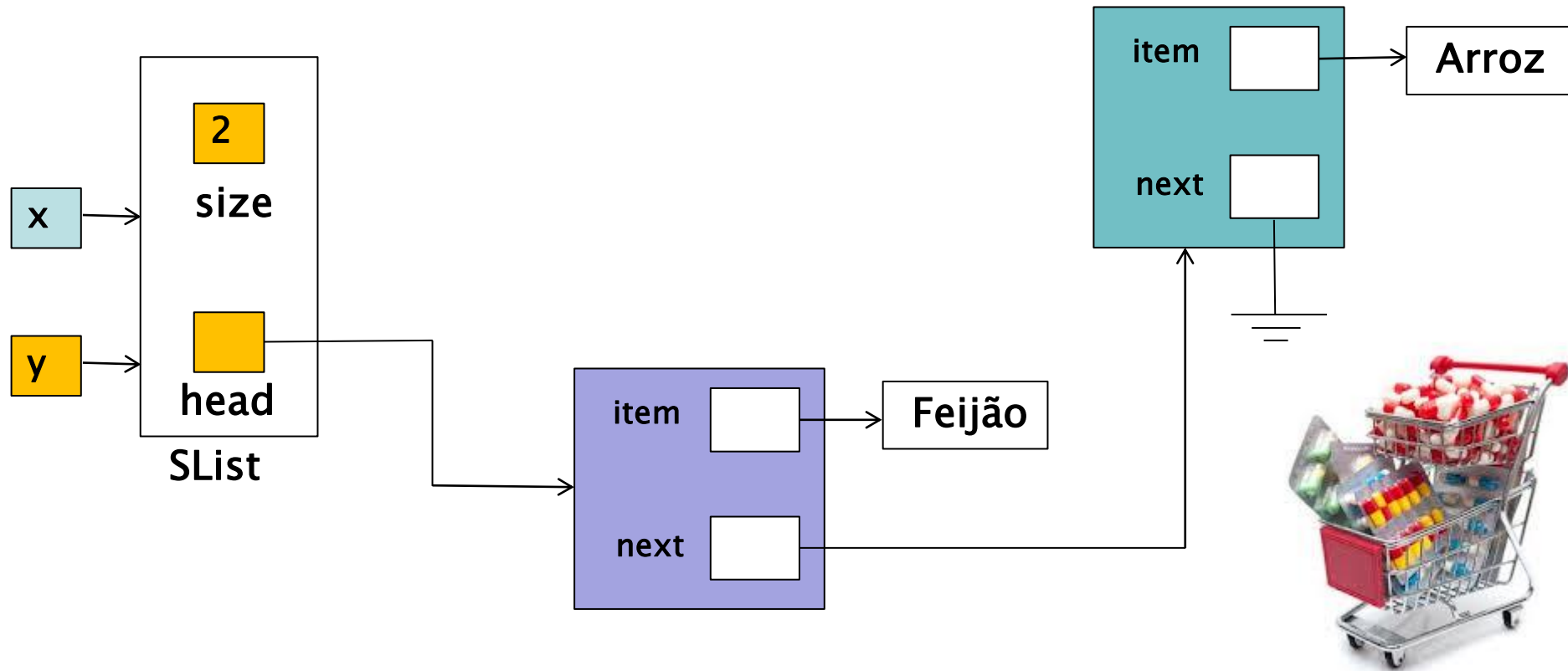


Será que os problemas 1 e 2 foram resolvidos ???



Solução do Problema 1

- Não haverá mais conflitos entre x e y ao se adicionar um novo item na lista de nós.

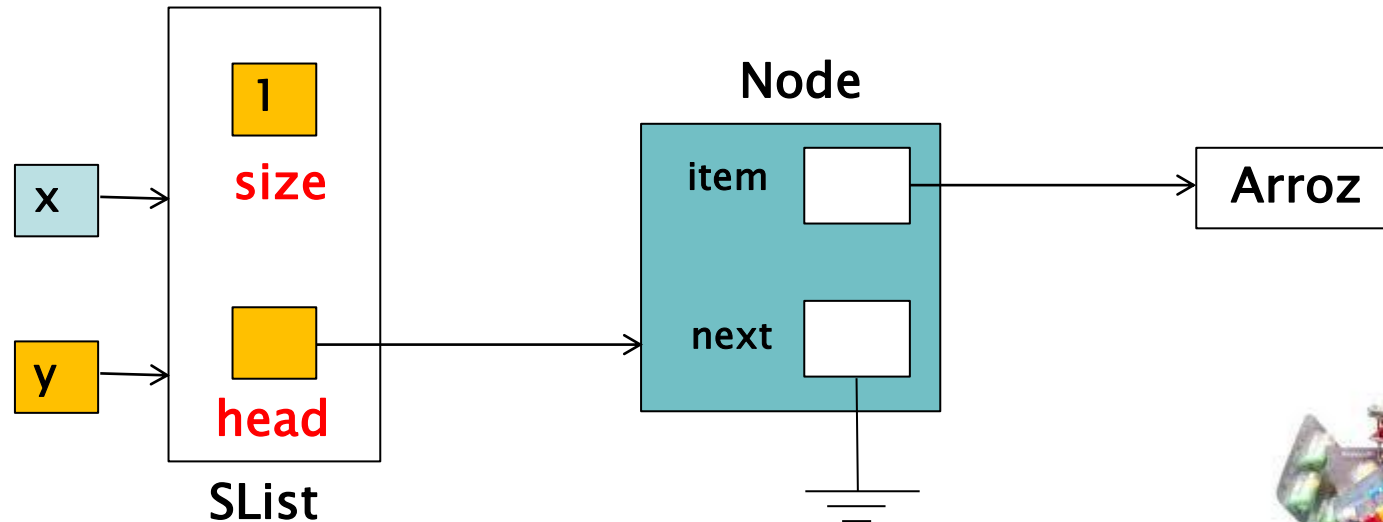


Tanto x como y apontam para todos os elementos da lista...



Solução do Problema 1

- Variáveis x e y apontam para **SList** que por sua vez aponta para a lista com apenas 1 nó.

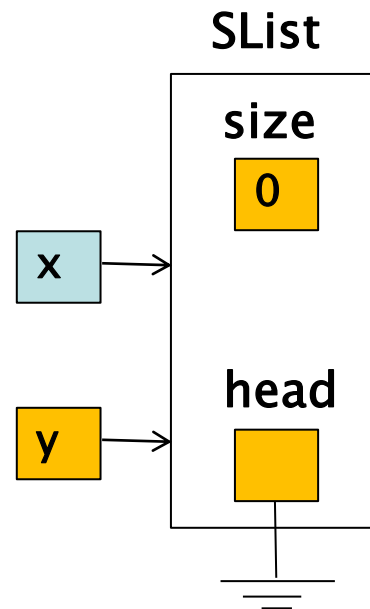


SList atua como interface para a lista de nós.



Solução do Problema 2

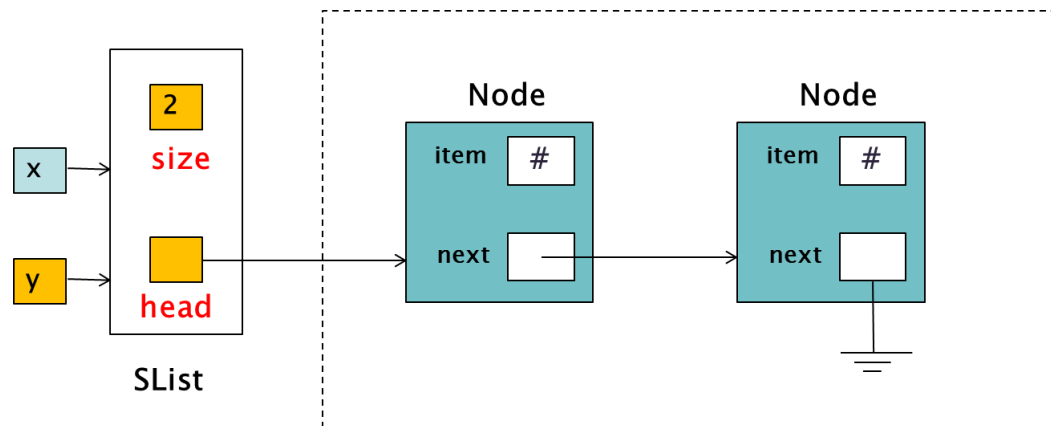
- A classe **SList** mantém em size o tamanho da lista. Para sabermos o tamanho da lista não é mais necessário varrer-se a lista para a contagem dos nós.
- Se a lista de nós está vazia, as referências x e y **NÃO** são mais nulas.



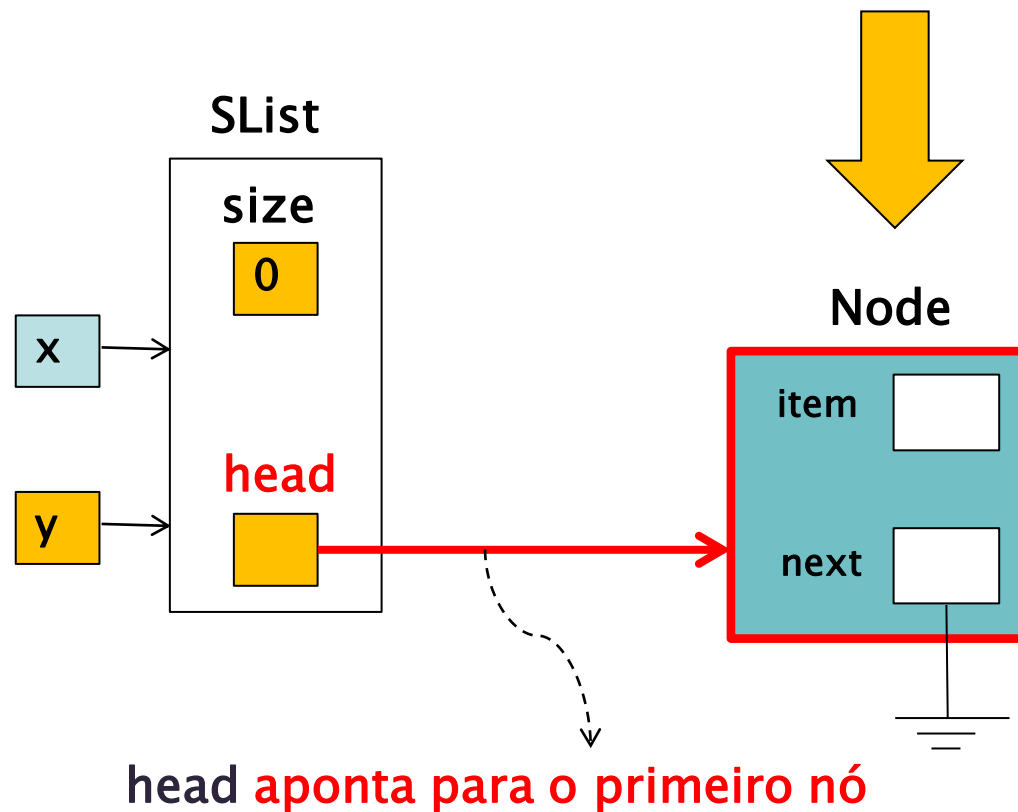
Não haverá erros de Run-time se a lista de nós estiver vazia ...



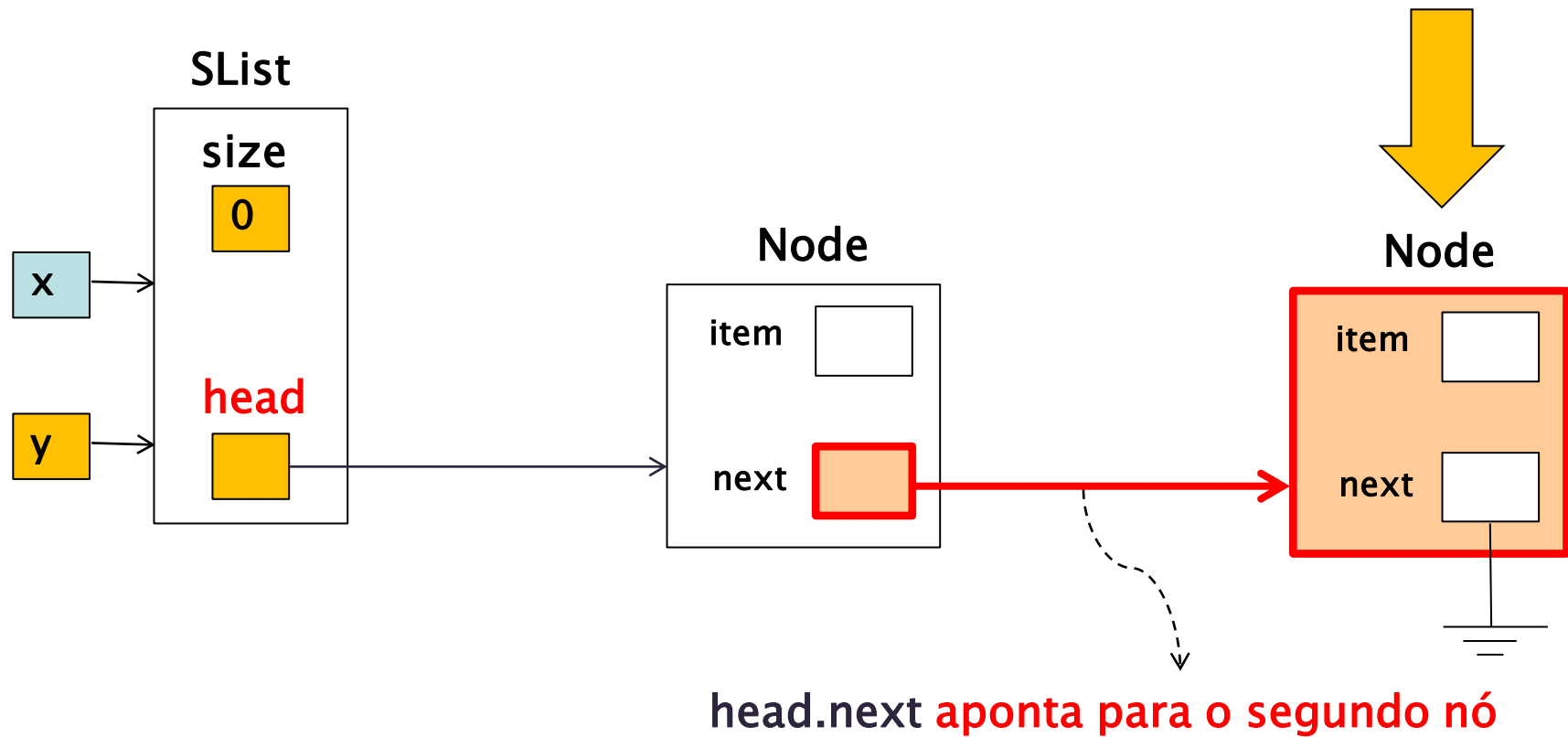
Implementação



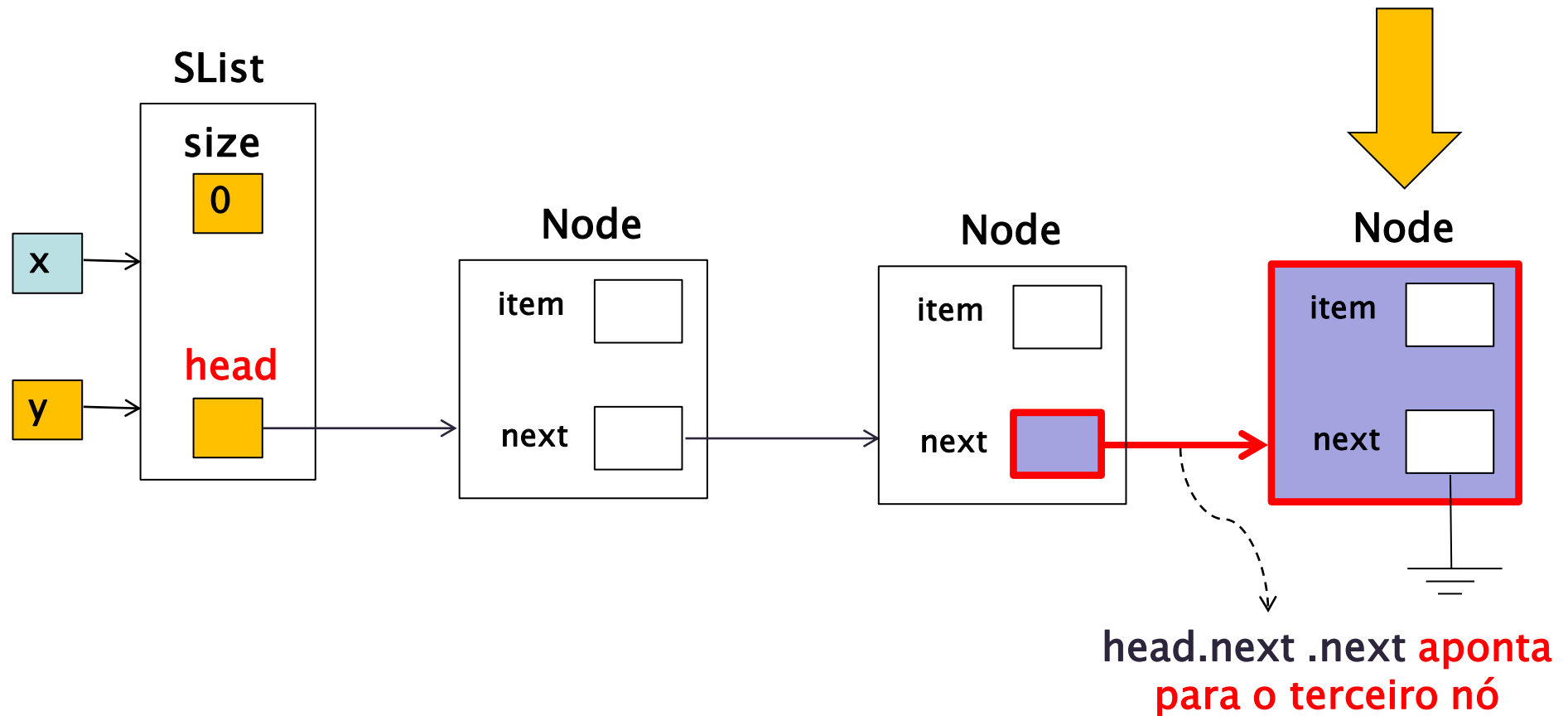
Dicas – Implementação



Dicas – Implementação



Dicas - Implementação



Classe Node

```
package maua;

public class Node {
    int item;
    Node next;

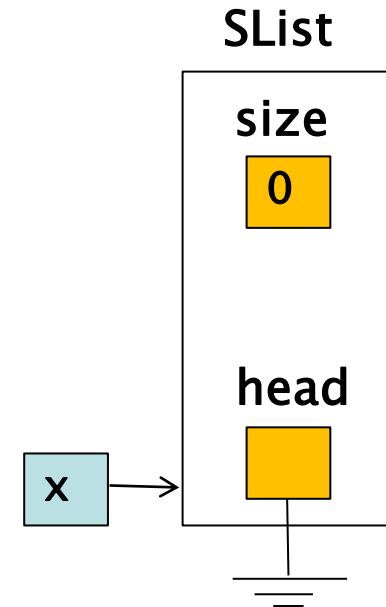
    public Node() {
        this.item=0;
        this.next=null;
    }

    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++;
    }
}
```



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++;

    }

}
```



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("");  
}
```



deleteInicio()

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



```

public void deleteFim() {
    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

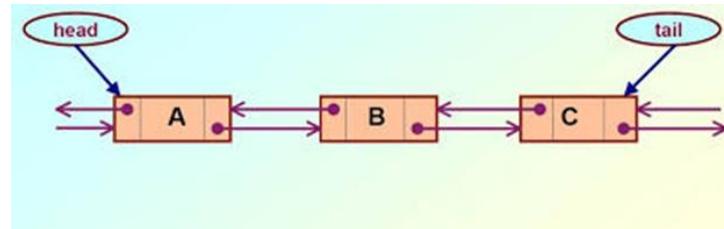
Funcao imprimeLista()

Lista: 10 99

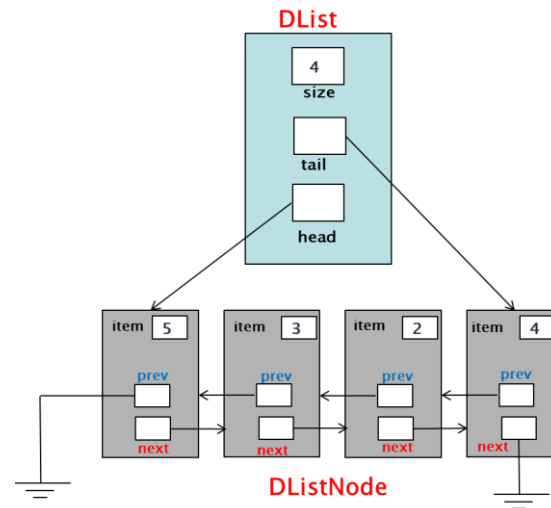
Funcao imprimeLista()

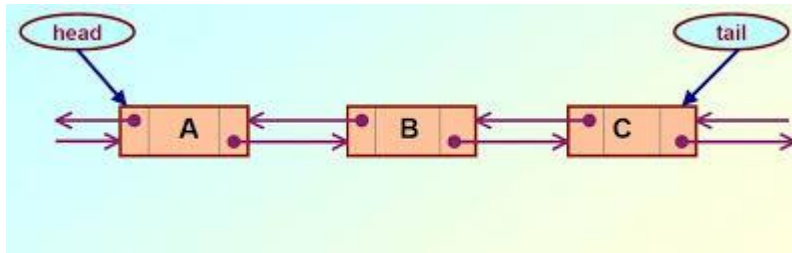
Lista: 10



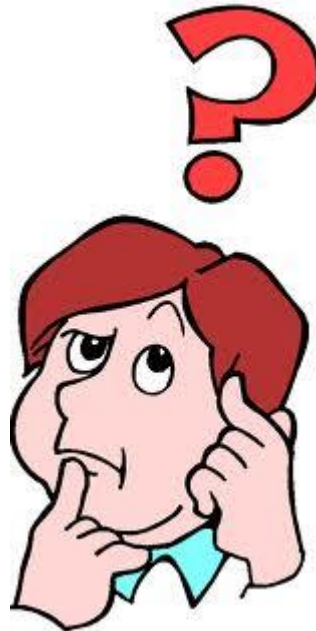


Porque listas duplamente ligadas



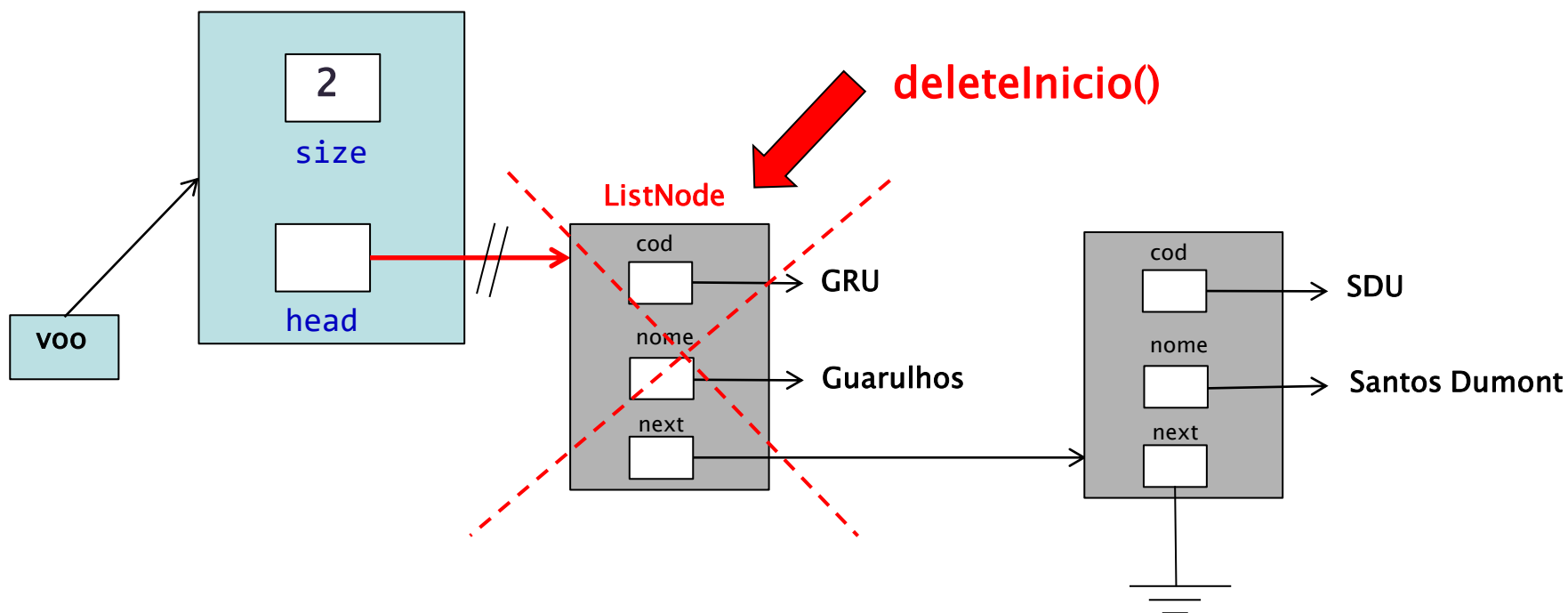


Qual a necessidade de listas duplamente ligadas ?



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.



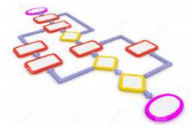
Função deleteInicio

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



O Tempo é constante e independe do tamanho da lista!



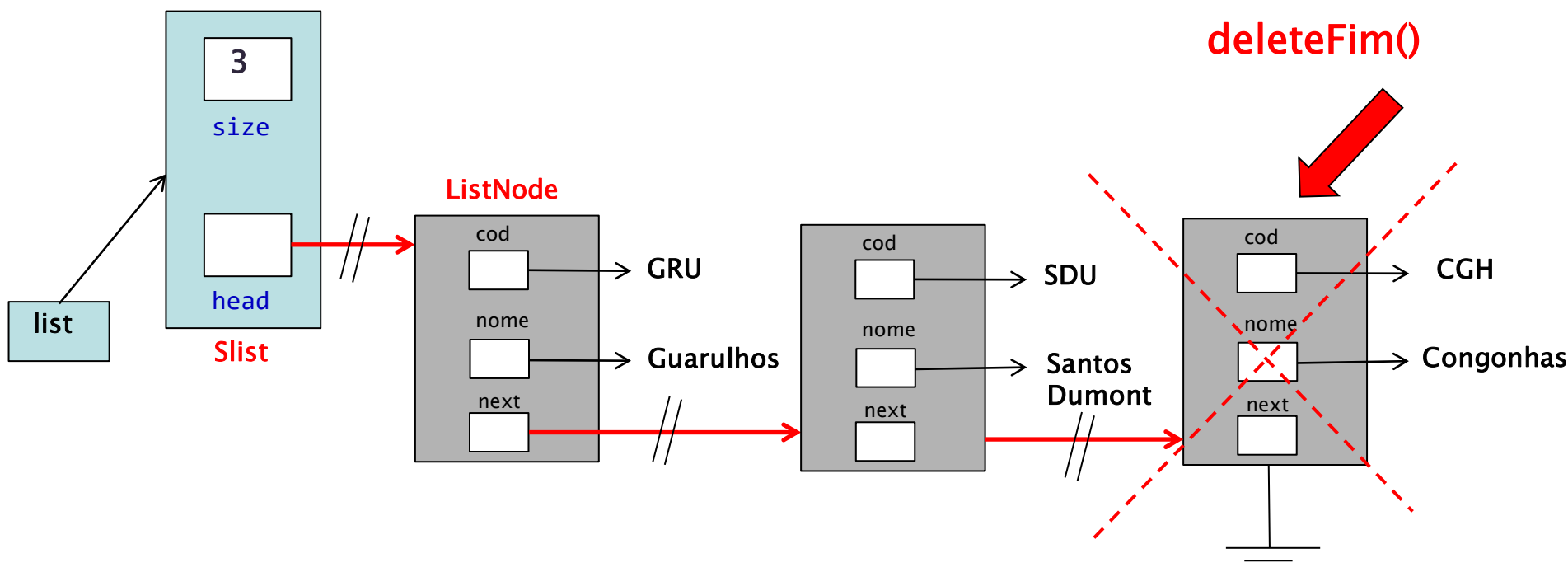


Mas, numa lista simplesmente ligada, a deleção
ou remoção no final da lista é **difícil ...**



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 !).

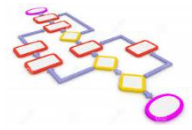


Função deleteFim()

```
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 ?

