Estruturas de Dados

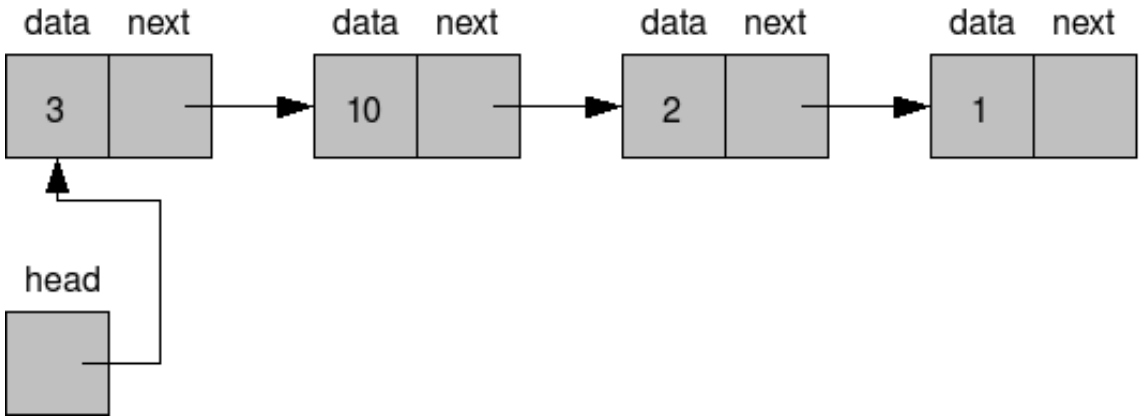
Listas

Universidade Federal do Rio Grande do Norte

Lista

Definição

Listas são conjuntos de dados em que os objetos estão **organizados de maneira linear**. Diferentemente de arrays, onde a ordem linear é determinada por um índice, nas listas a ordem é determinada por um **ponteiro em cada objeto**.



FONTE: https://commons.wikimedia.org/wiki/File:C_language_linked_list.png

Dado	Link para o próximo elemento.

Lista

```
class Node {
    constructor(dado) {
        this.dado = dado;
        this.proximo = null;
    }
}
```

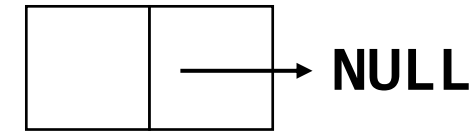
Nó de uma lista

Dado	Link para o próximo elemento.

Lista

Lista Vazia

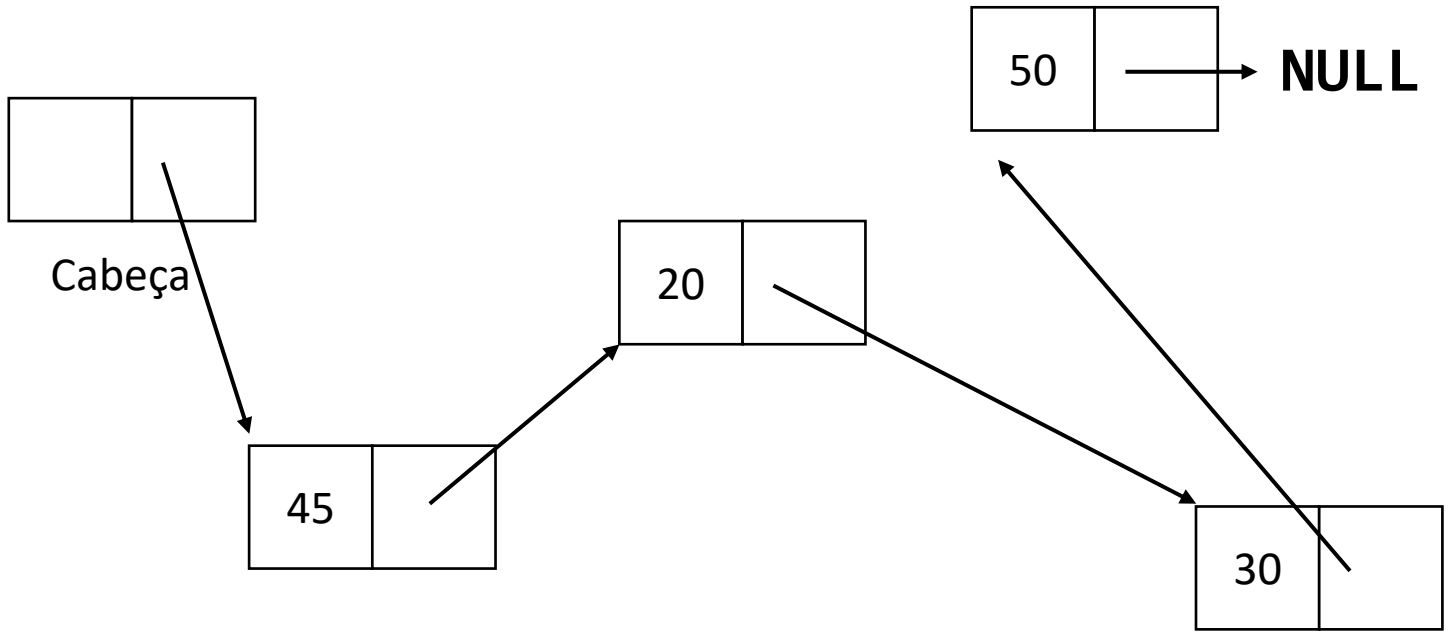
A lista pode ser considerada vazia quando o próximo elemento apontado pela **cabeça da lista** é NULL.



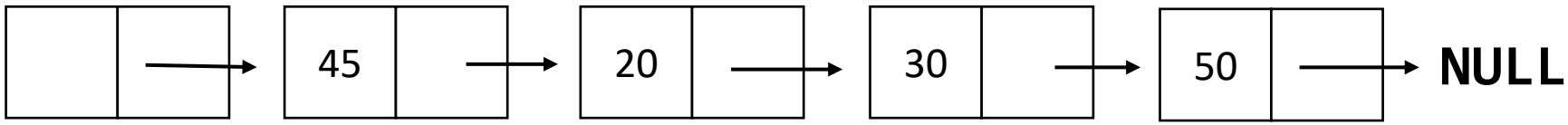
Cabeça da lista

Exemplo de uma lista “na memória”

- Como a lista não é implementada com um array, ela não será armazenada de maneira contígua na memória.



Representação de uma lista simplesmente encadeada



Cabeça

Primeiro elemento

Último elemento

Operações para Listas

add (dado)

Aloca um espaço de memória para um nó e adiciona um dado no início da lista.

append (dado)

Aloca um espaço de memória para um nó e adiciona um dado no final da lista.

addAt (dado, posição)

Aloca um espaço de memória para um nó e adiciona um dado em uma posição específica da lista.

Operações para Listas

removeFirst ()

Remove um nó do início da lista e libera o espaço de memória usado pelo nó.

removeLast ()

Remove um nó do fim da lista e libera o espaço de memória usado pelo nó.

removeAt (dado, posição)

Remove um nó de uma posição específica da lista e libera o espaço de memória usado pelo nó.

Outras Operações

isEmpty ()

Verifica se a lista está vazia.

search (data)

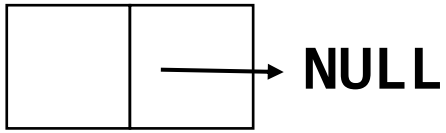
Pesquisa se um dado está presente em algum nó da lista.

length()

Informa a quantidade de elementos na lista.

Implementação - Adicionar Elemento no Inicio

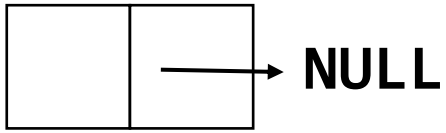
add(20)



Cabeça

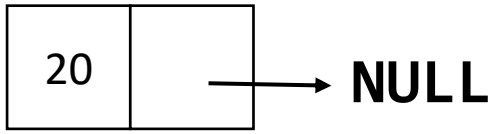
Implementação - Adicionar Elemento no Inicio

add(20)



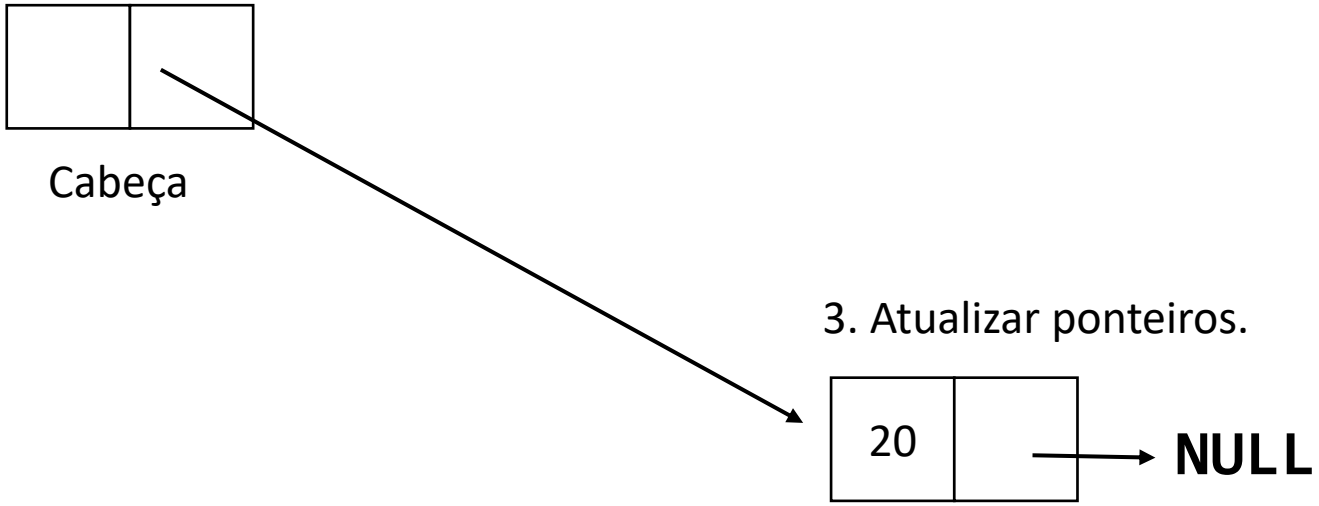
Cabeça

1. Instanciar o nó.



Implementação - Adicionar Elemento no Inicio

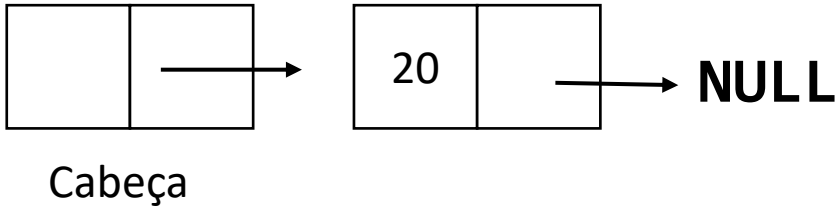
add(20)




```
novo_no->proximo = head->proximo
head->proximo = novo_no
```

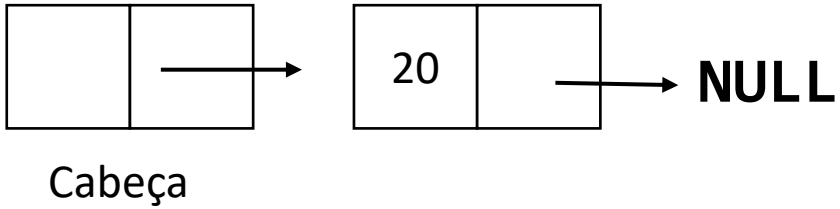
Implementação - Adicionar Elemento no Inicio

add(20)



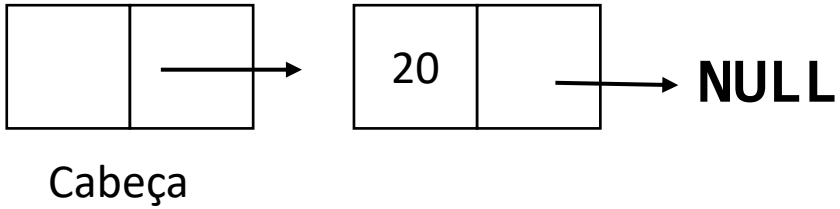
Implementação - Adicionar Elemento no Inicio

add(45)

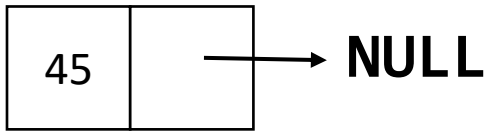


Implementação - Adicionar Elemento no Inicio

add(45)

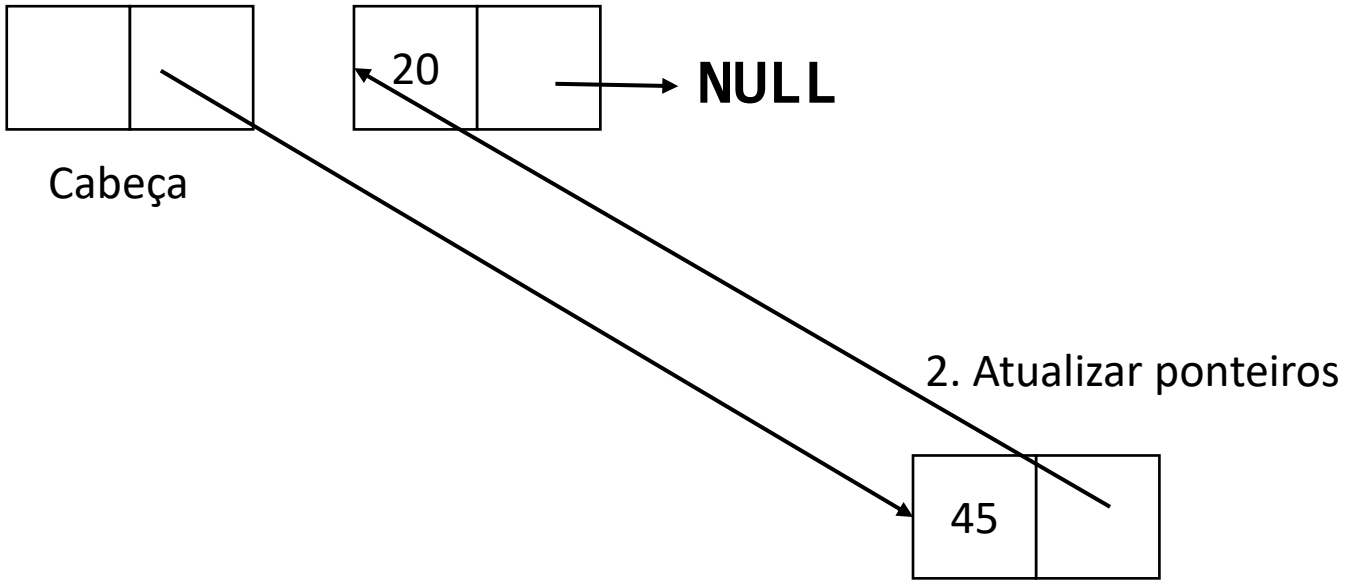


1. Instanciar o nó



Implementação - Adicionar Elemento no Inicio

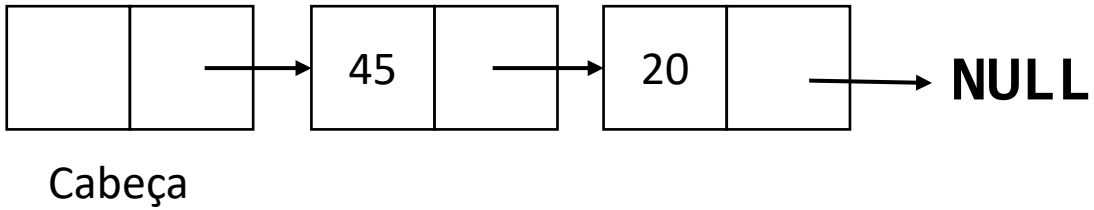
add(45)




```
novo_no->proximo = head->proximo
head->proximo = novo_no
```

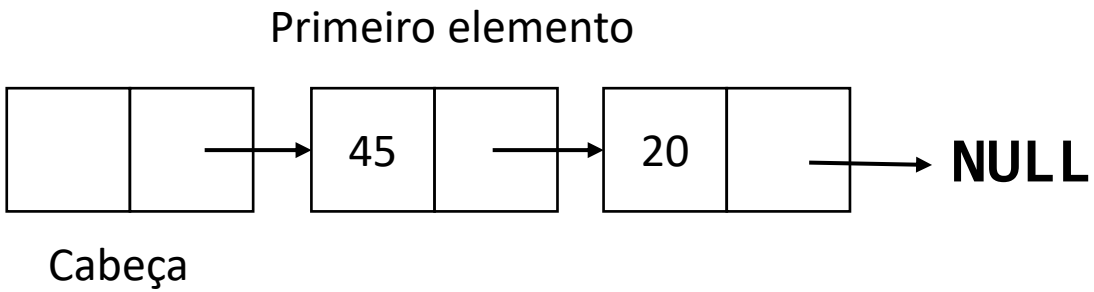
Implementação - Adicionar Elemento no Inicio

add(45)



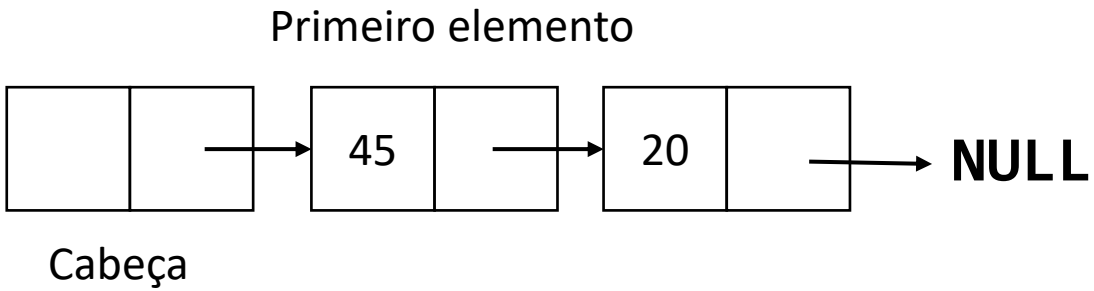
Implementação - Adicionar Elemento no Final

append(30)

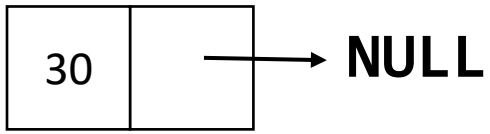


Implementação - Adicionar Elemento no Final

append(30)



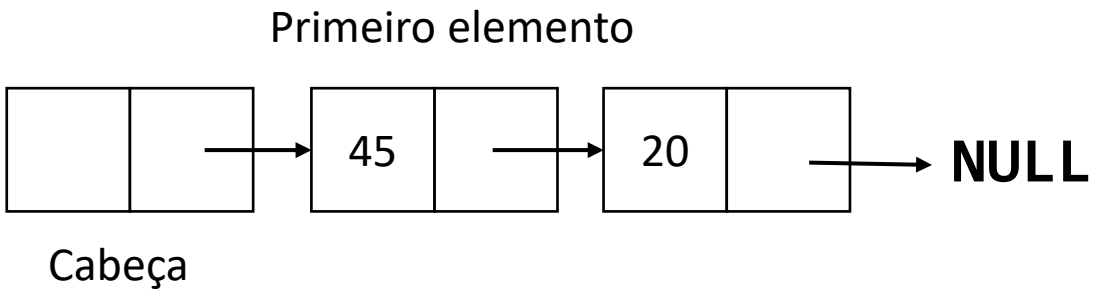
1. Instanciar o nó



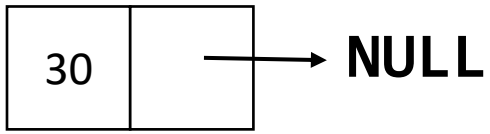
Implementação - Adicionar Elemento no Final

append(30)

```
Se a lista estiver vazia{
            head->proximo=novo_no;
}
```

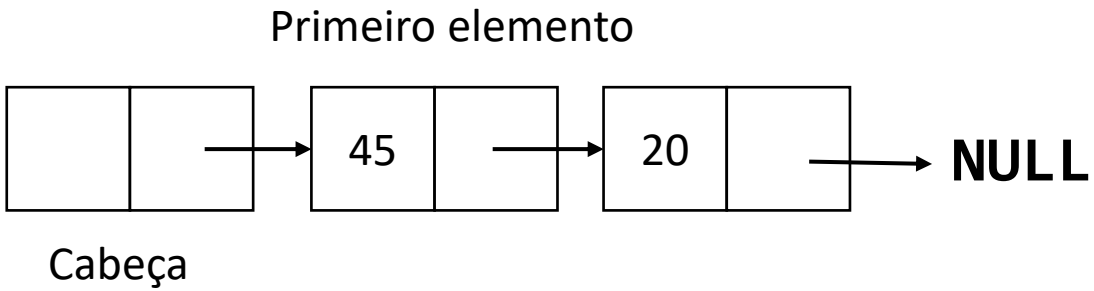


2. Atualizar ponteiros



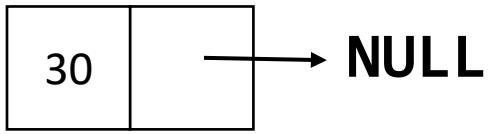
Implementação - Adicionar Elemento no Final

append(30)



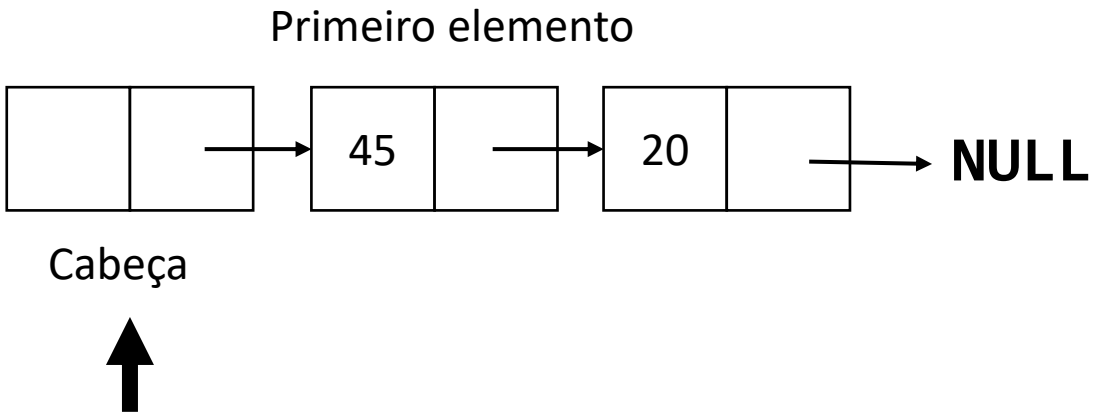
Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

2. Atualizar ponteiros



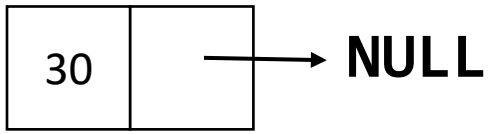
Implementação - Adicionar Elemento no Final

append(30)



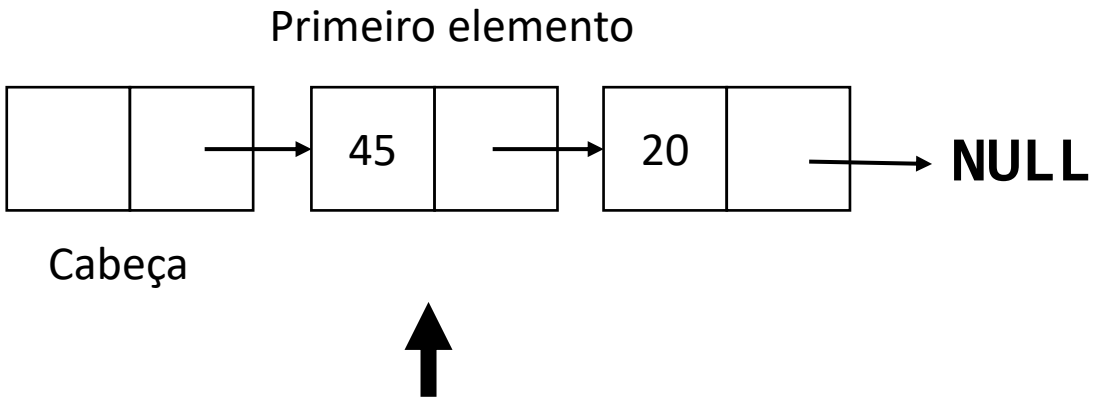
Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

2. Atualizar ponteiros



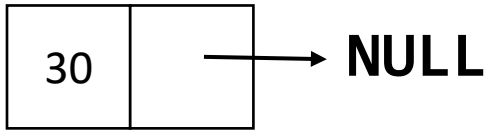
Implementação - Adicionar Elemento no Final

append(30)



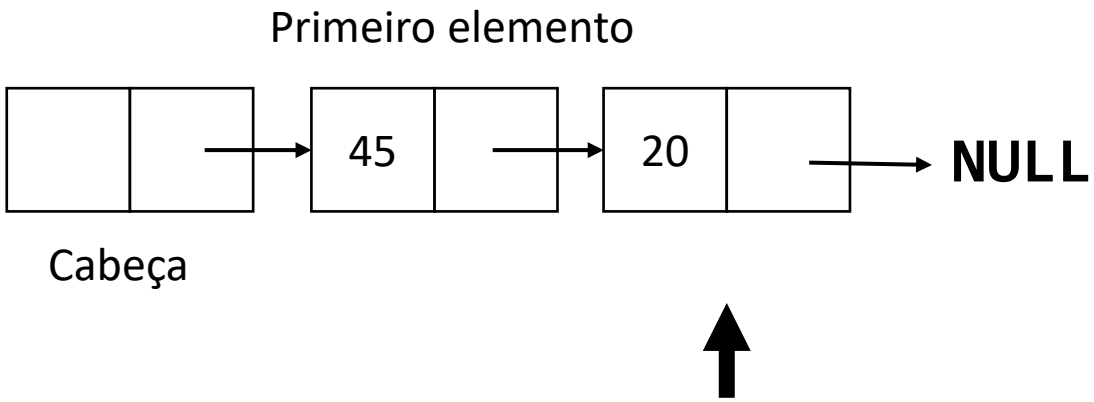
Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

2. Atualizar ponteiros



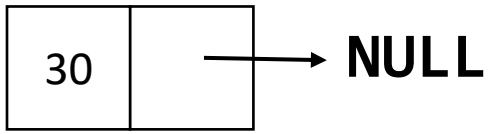
Implementação - Adicionar Elemento no Final

append(30)



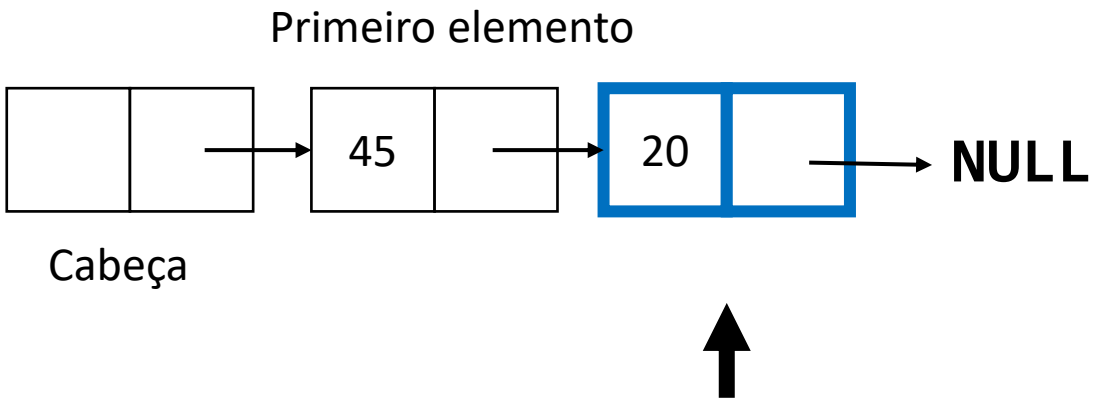
Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

2. Atualizar ponteiros



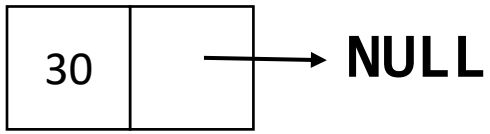
Implementação - Adicionar Elemento no Final

append(30)



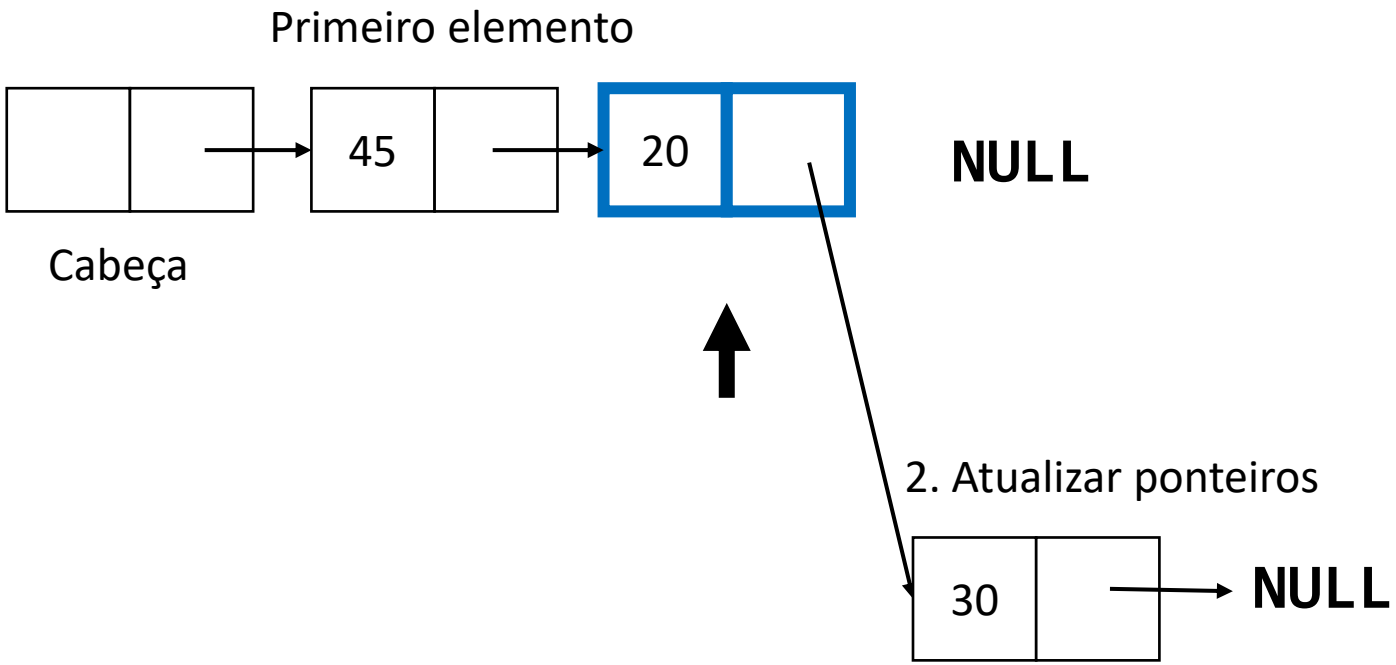
Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

2. Atualizar ponteiros



Implementação - Adicionar Elemento no Final

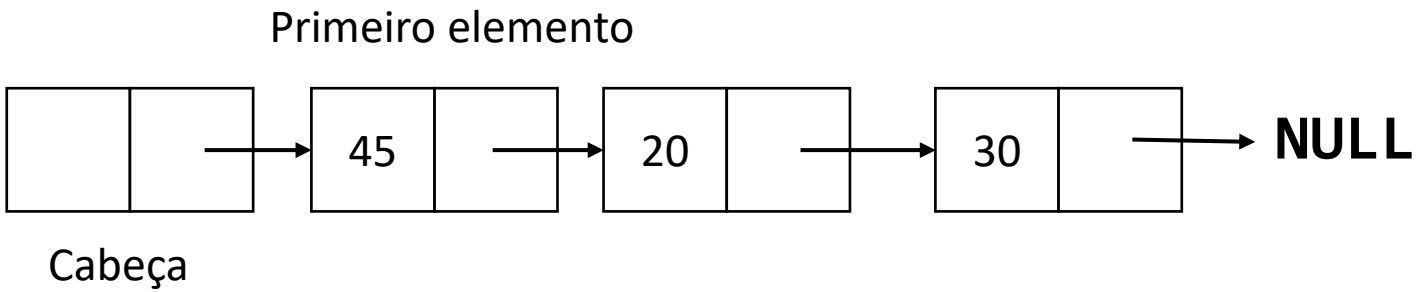
append(30)



Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

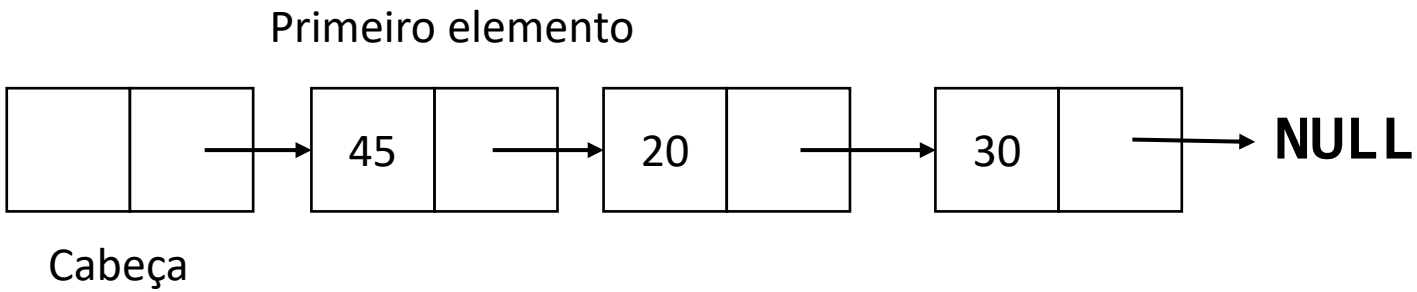
Implementação - Adicionar Elemento no Final

append(30)



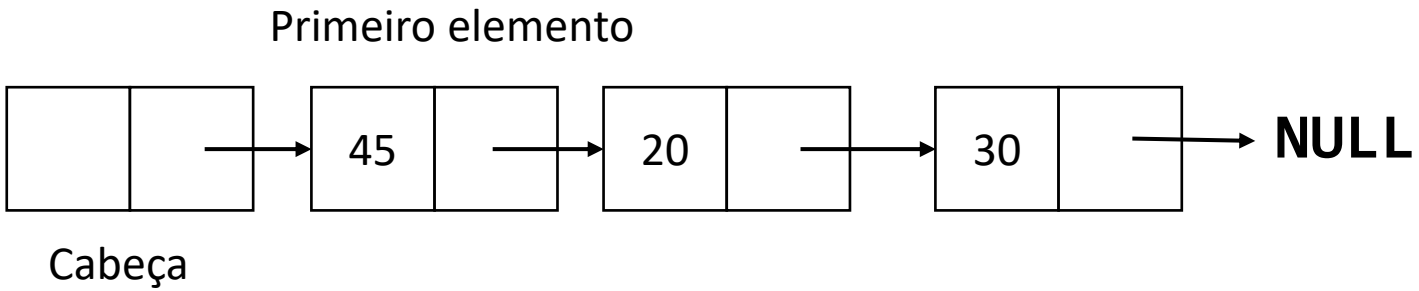
Implementação - Adicionar Elemento no Final

append(50)

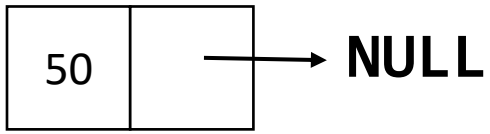


Implementação - Adicionar Elemento no Final

append(50)



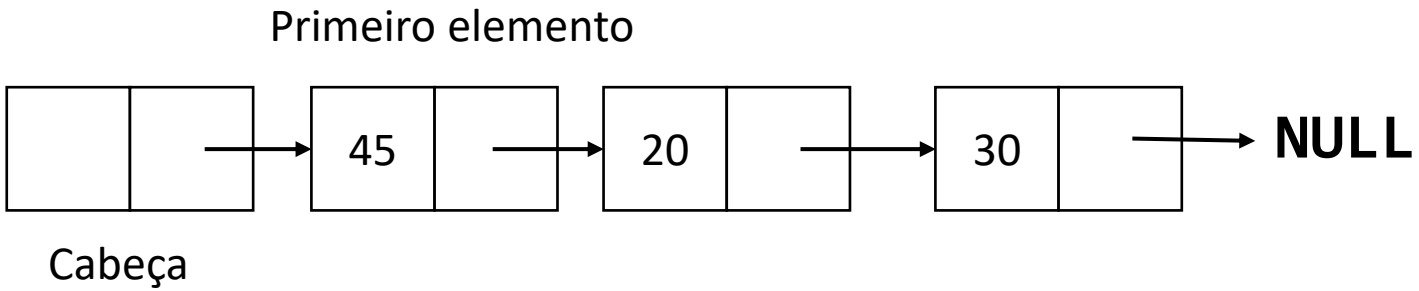
1. Instanciar o nó



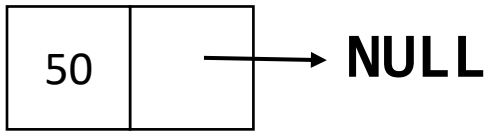
Implementação - Adicionar Elemento no Final

append(50)

```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



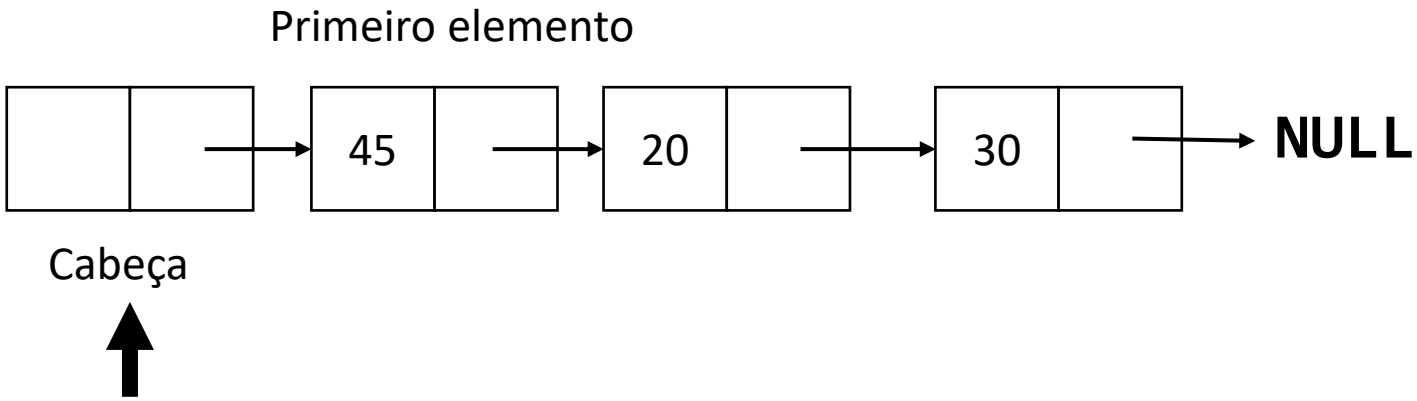
2. Atualizar ponteiros



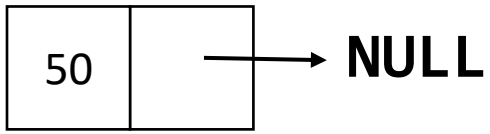
Implementação - Adicionar Elemento no Final

append(50)

```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



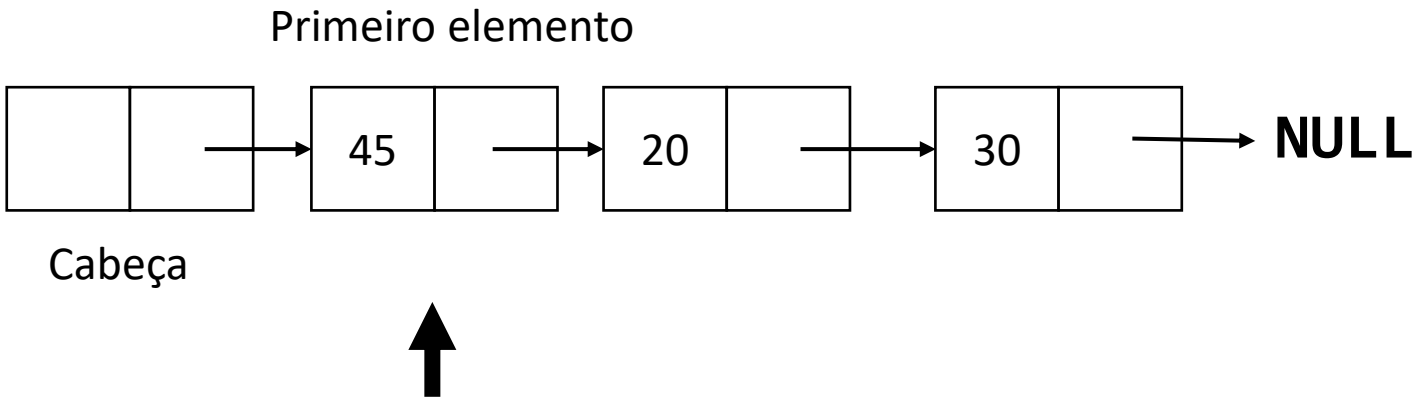
2. Atualizar ponteiros



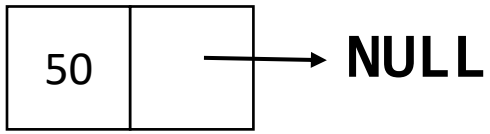
Implementação - Adicionar Elemento no Final

append(50)

```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



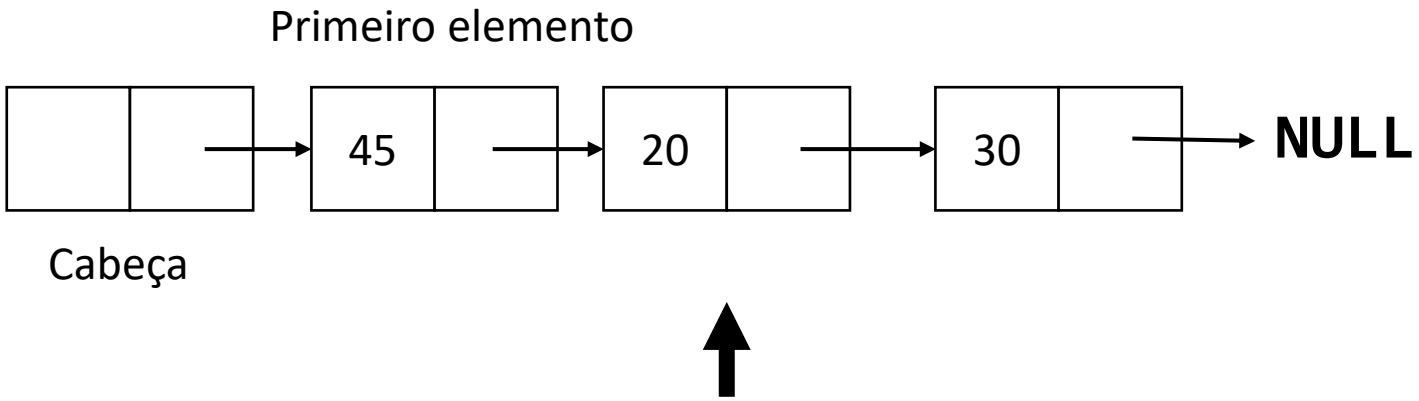
2. Atualizar ponteiros



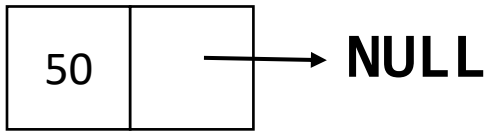
Implementação - Adicionar Elemento no Final

append(50)

```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



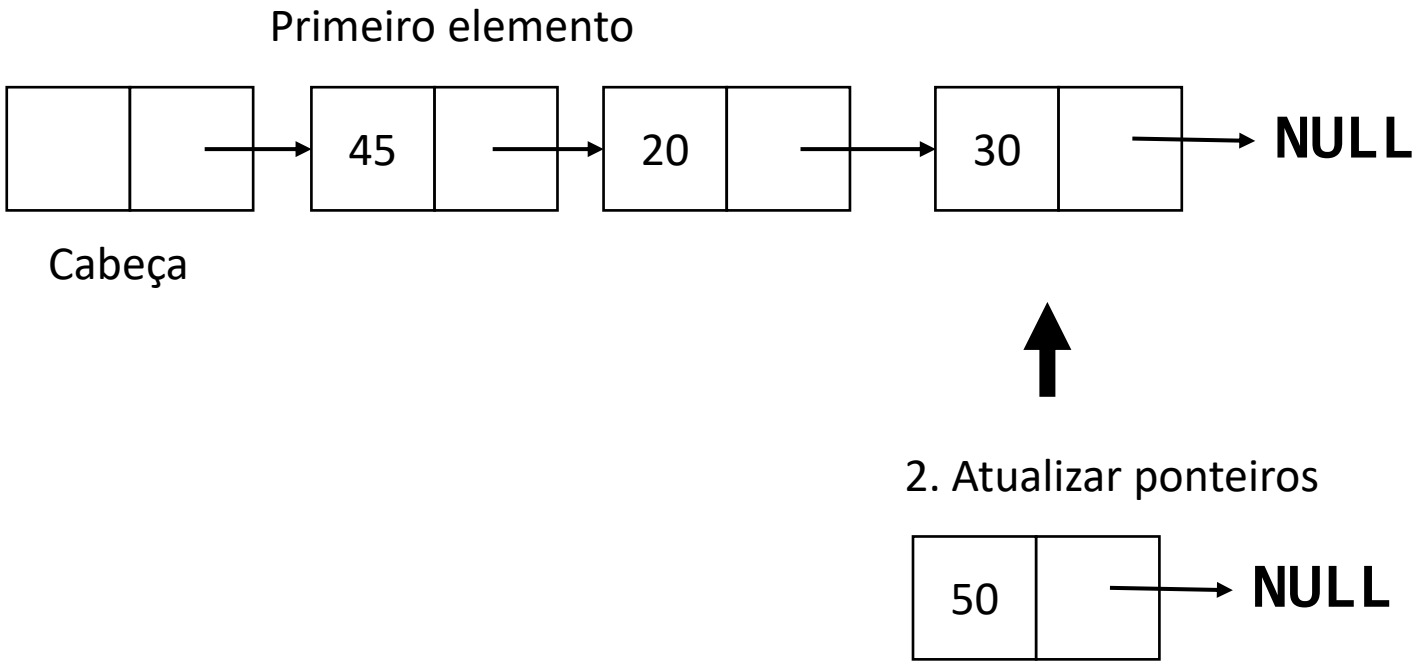
2. Atualizar ponteiros



Implementação - Adicionar Elemento no Final

append(50)

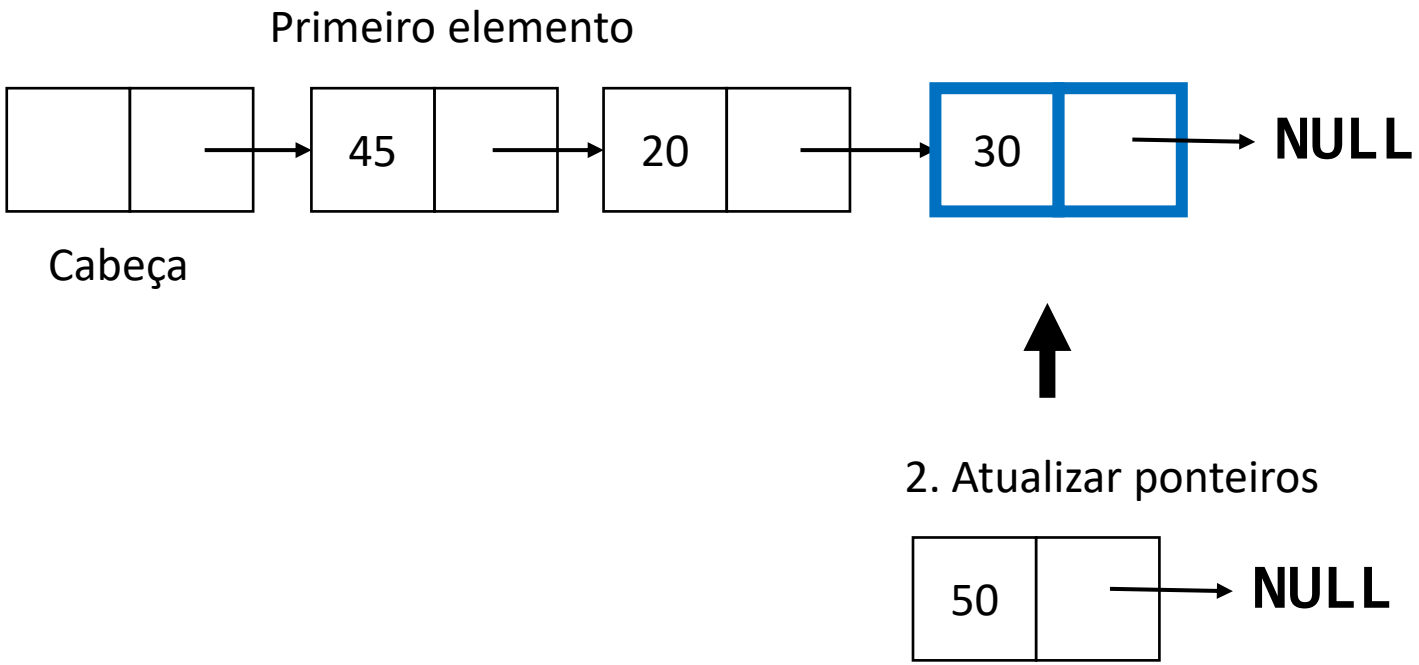
```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



Implementação - Adicionar Elemento no Final

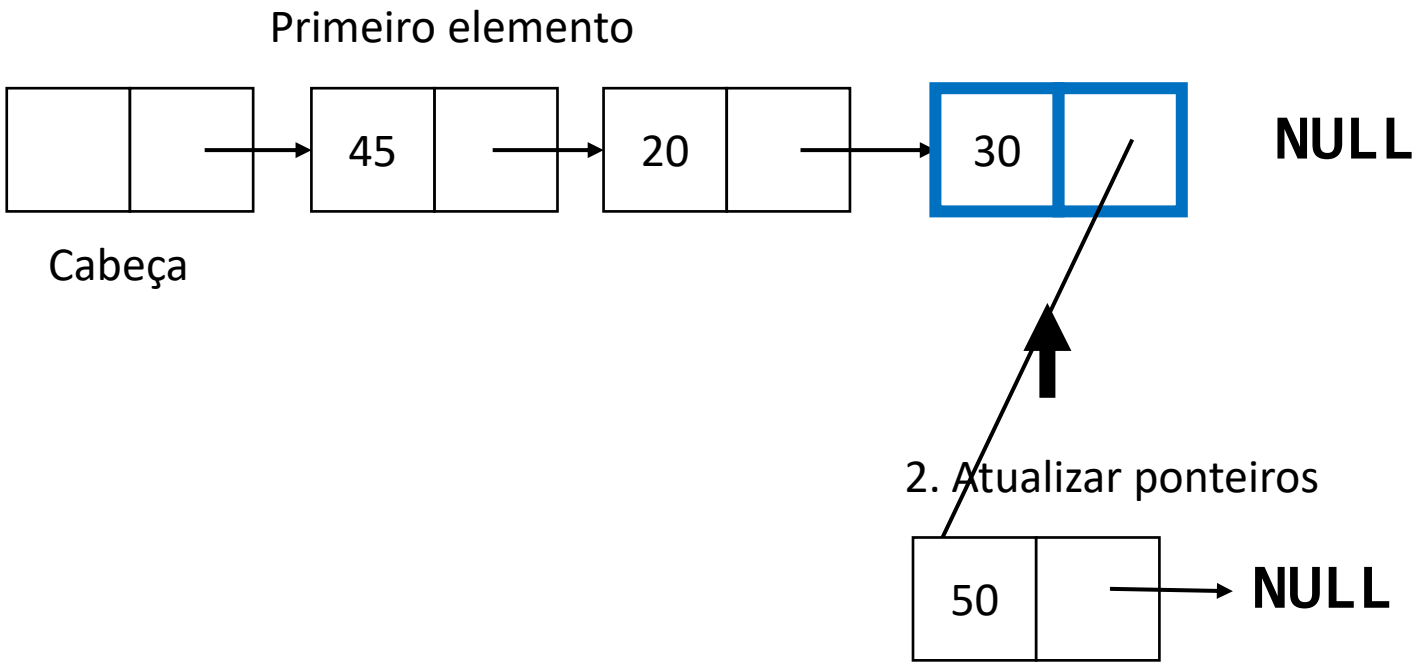
append(50)

```
Se a lista não está vazia, varra a
lista do início até descobrir qual
nó aponta para NULL, esse nó agora
deverá apontar para novo_no.
```



Implementação - Adicionar Elemento no Final

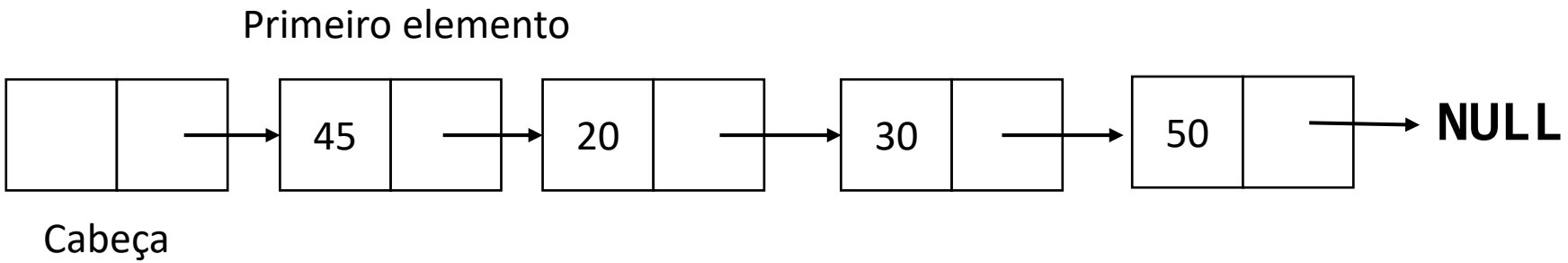
append(50)



Se a lista não está vazia, varra a lista do início até descobrir qual nó aponta para NULL, esse nó agora deverá apontar para novo_no.

Implementação - Adicionar Elemento no Final

append(50)

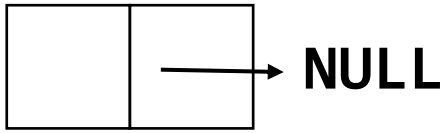


Implementação - Remover um elemento do início

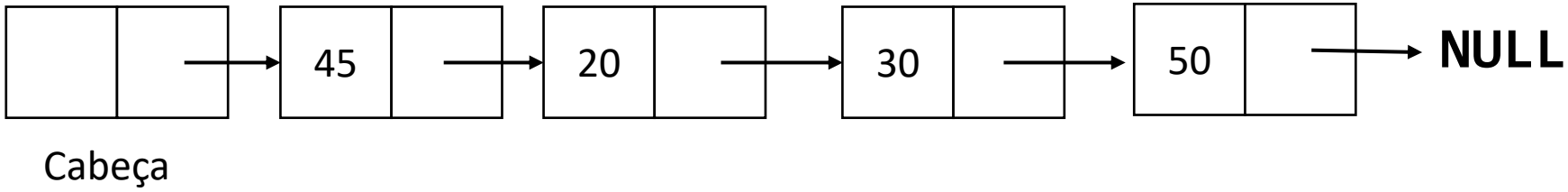
removeFirst()

CASO 1: Se a lista está vazia não há nada a ser feito.

CASO 1



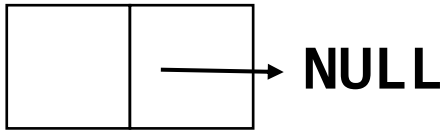
CASO 2



Implementação - Remover um elemento do início

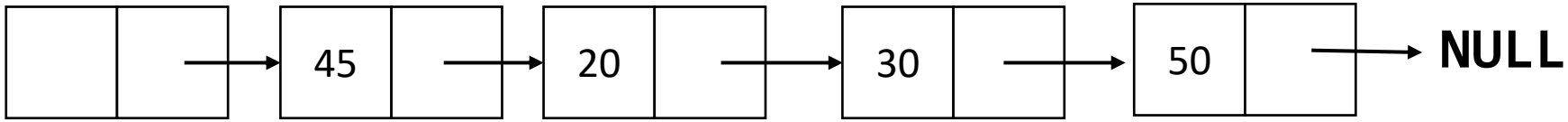
removeFirst()

CASO 1



Cabeça

CASO 2



Cabeça

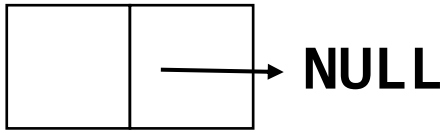
CASO 2: Se a lista possui elementos,

1) aux = head.próximo
2) head.próximo = aux.proximo

Implementação - Remover um elemento do início

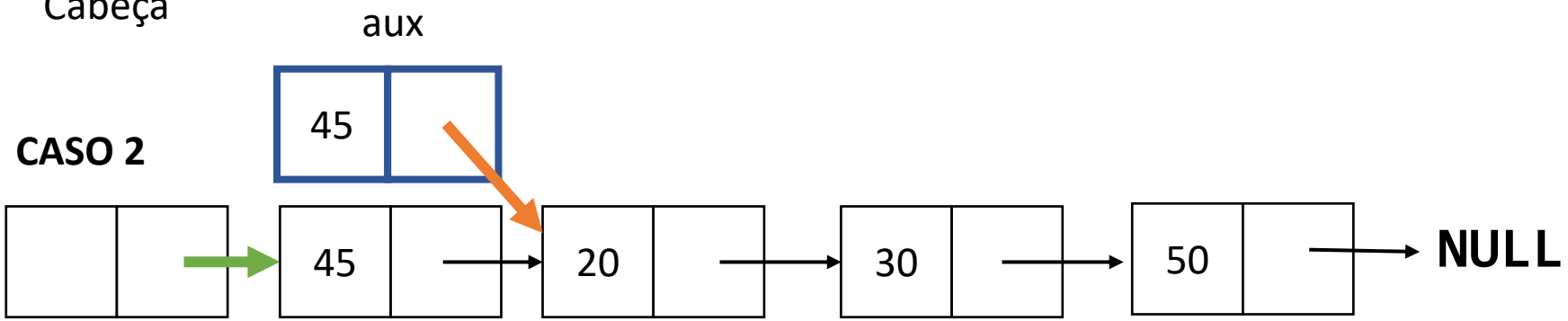
removeFirst()

CASO 1



Cabeça

CASO 2



Cabeça

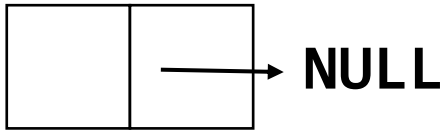
CASO 2: Se a lista possui elementos,

1) aux = head.próximo
2) head.próximo = aux.proximo.

Implementação - Remover um elemento do início

removeFirst()

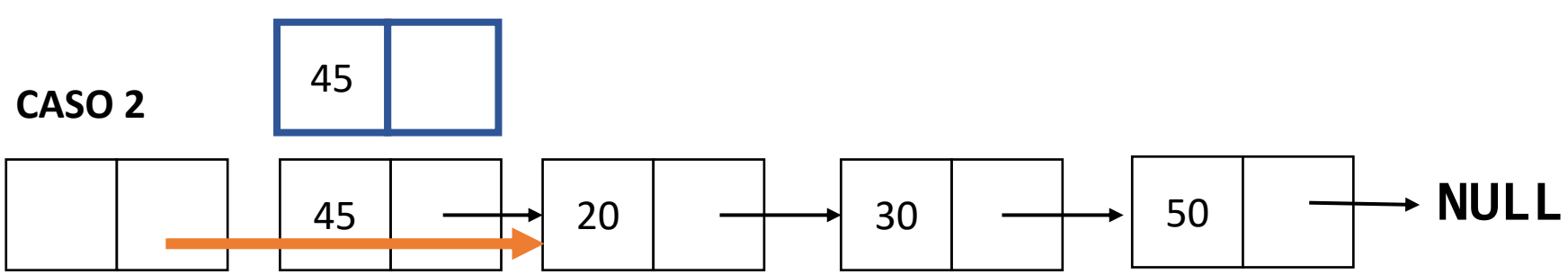
CASO 1



Cabeça

aux

CASO 2



Cabeça

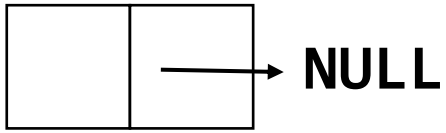
CASO 2: Se a lista possui elementos,

1) aux = head.proximo
2) head.proximo = aux.proximo.

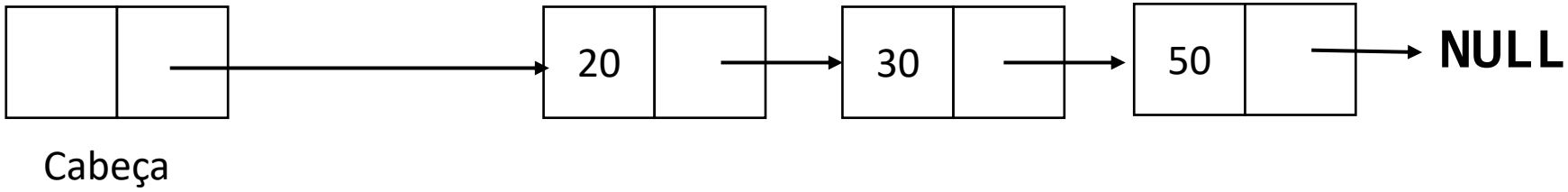
Implementação - Remover um elemento do início

removeFirst()

CASO 1



CASO 2



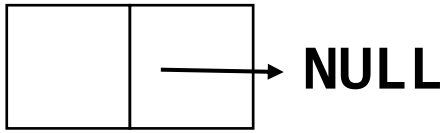
CASO 2: Se a lista possui elementos,

1) aux = head.proximo
2) head.proximo = aux.proximo.

Implementação - Remover um elemento do início

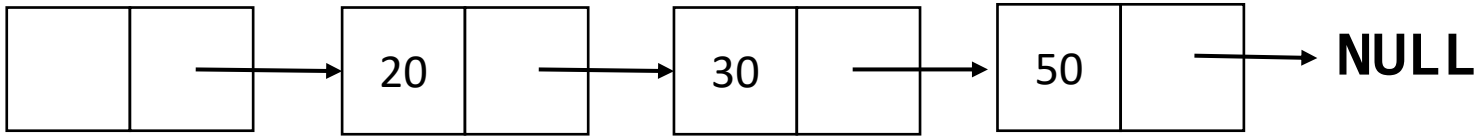
removeFirst()

CASO 1



Cabeça

CASO 2



Cabeça

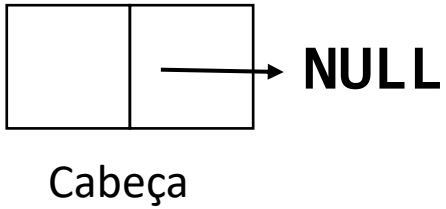
CASO 2: Se a lista possui elementos,

1) aux = head.proximo
2) head.proximo = aux.proximo.

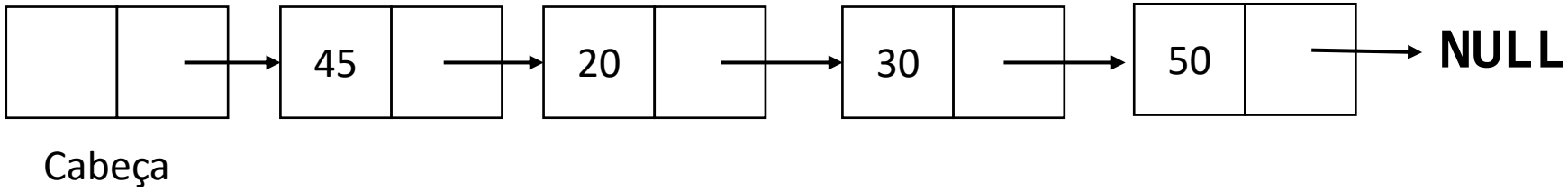
Implementação - Remover um elemento do final

removeLast()

CASO 1



CASO 2

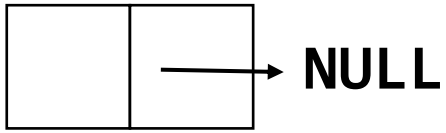


Implementação - Remover um elemento do final

removeLast()

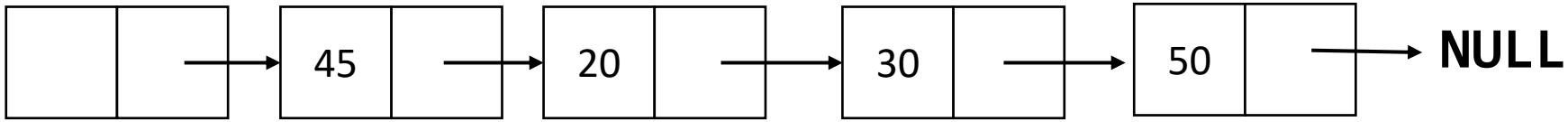
CASO 1: Se a lista está vazia não há nada a ser feito.

CASO 1



Cabeça

CASO 2

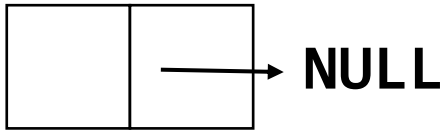


Cabeça

Implementação - Remover um elemento do final

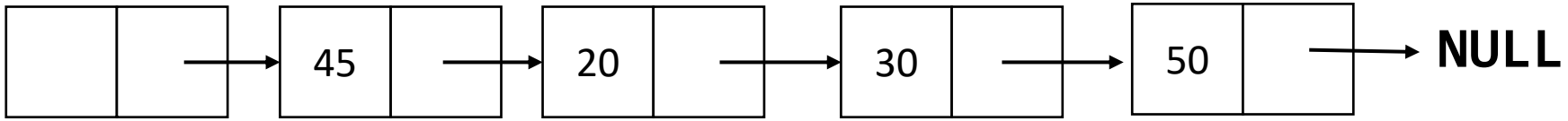
removeLast()

CASO 1



Cabeça

CASO 2



Cabeça

CASO 2: Se a lista não está vazia:

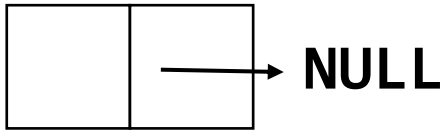
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ **NULL** faça
    1) node_a = node_b
    2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = **NULL**

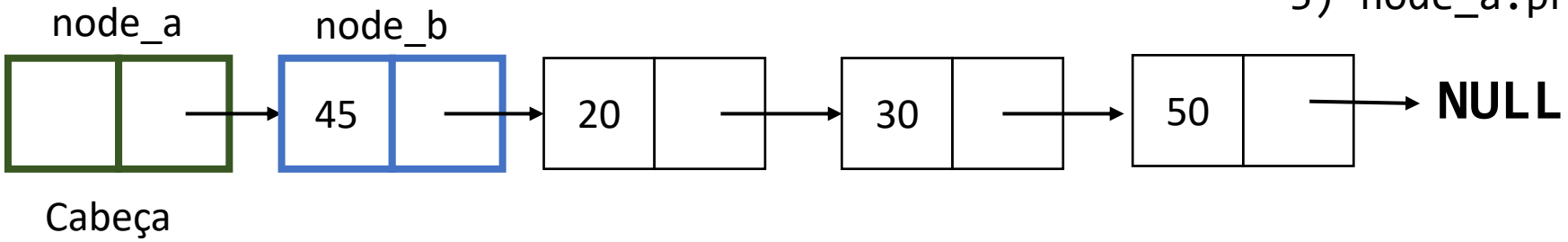
Implementação - Remover um elemento do final

removeLast()

CASO 1



CASO 2



CASO 2: Se a lista não está vazia:

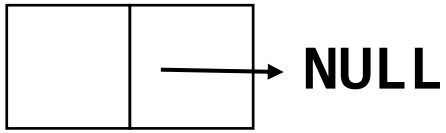
```
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ NULL faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = NULL
```

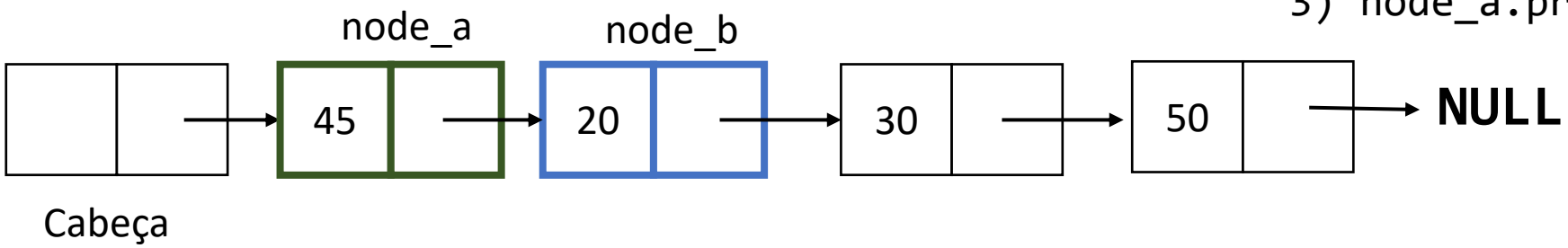
Implementação - Remover um elemento do final

removeLast()

CASO 1



CASO 2



CASO 2: Se a lista não está vazia:

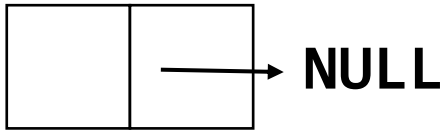
```
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ NULL faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = NULL
```

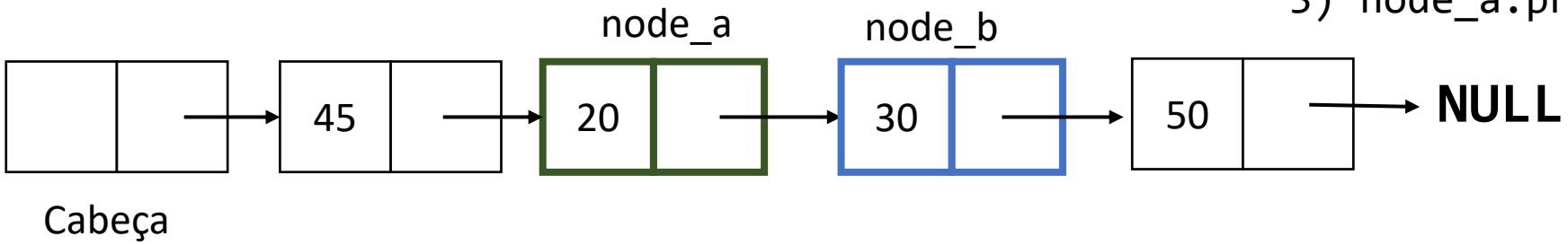
Implementação - Remover um elemento do final

removeLast()

CASO 1



CASO 2



CASO 2: Se a lista não está vazia:

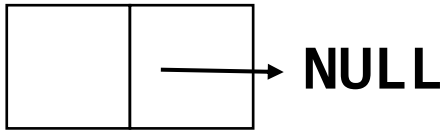
```
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ NULL faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = NULL
```

Implementação - Remover um elemento do final

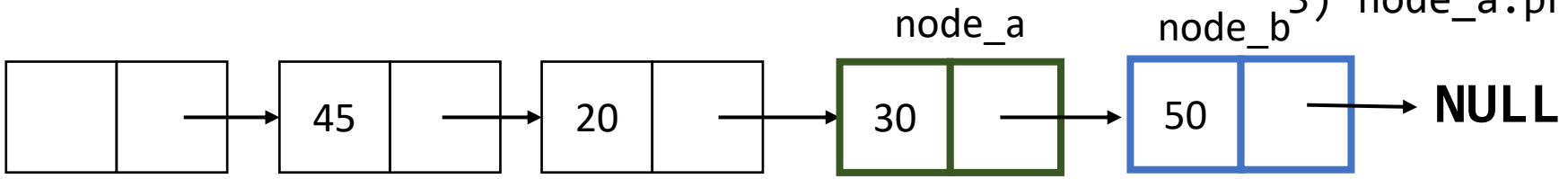
removeLast()

CASO 1



Cabeça

CASO 2



Cabeça

CASO 2: Se a lista não está vazia:

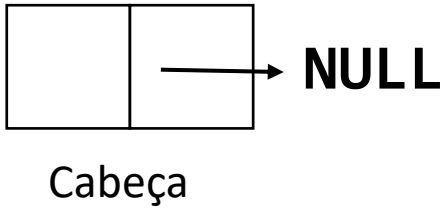
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ **NULL** faça
    1) node_a = node_b
    2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = **NULL**

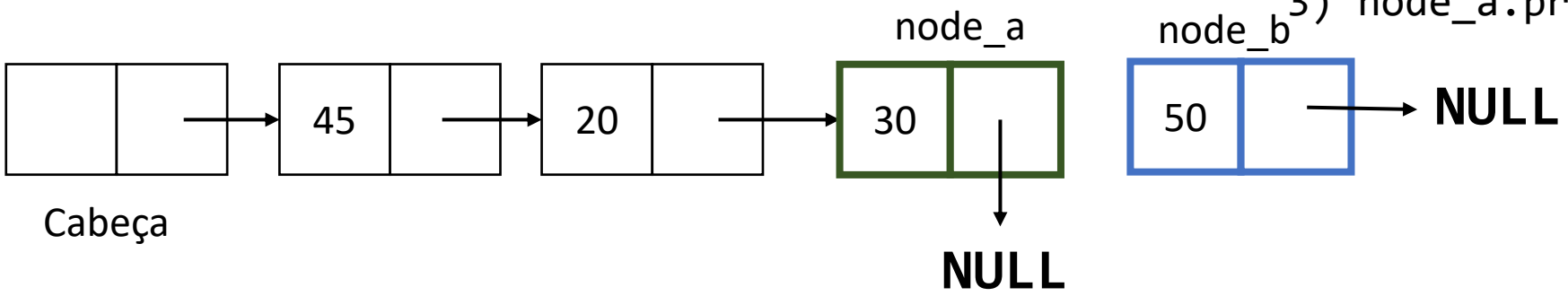
Implementação - Remover um elemento do final

removeLast()

CASO 1



CASO 2



CASO 2: Se a lista não está vazia:

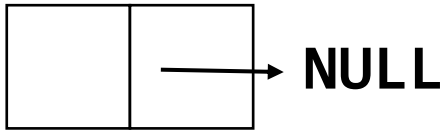
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ **NULL** faça
    1) node_a = node_b
    2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = **NULL**

Implementação - Remover um elemento do final

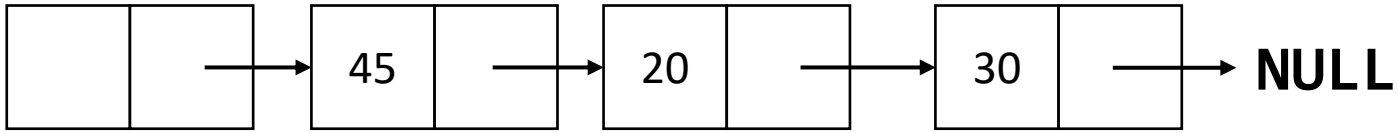
removeLast()

CASO 1



Cabeça

CASO 2



Cabeça

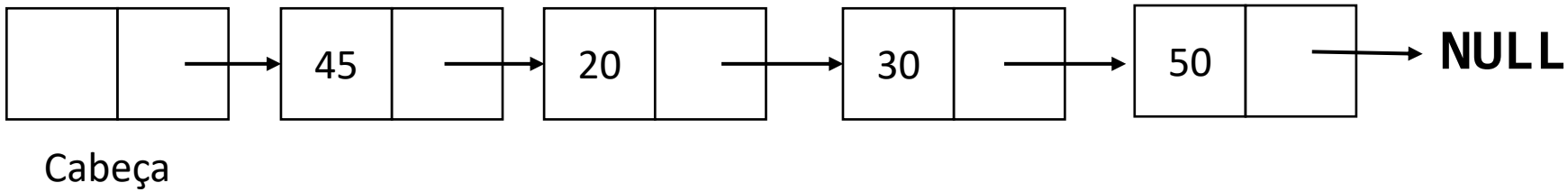
CASO 2: Se a lista não está vazia:

```
1) node_a = head
2) node_b = head.próximo

enquanto node_b.próximo ≠ NULL faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.próximo = NULL
```

Implementação – Adicionar entre elementos

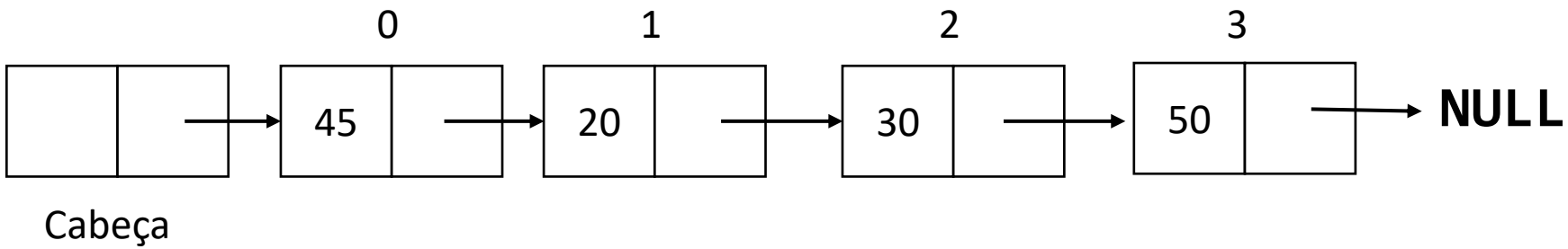
addAt(50, pos)



- Em algumas situações precisamos adicionar um elemento entre outros elementos. Exemplos:
  - Quando há a necessidade de manter os elementos ordenados;
  - Quando queremos colocar o novo elemento em uma posição específica;

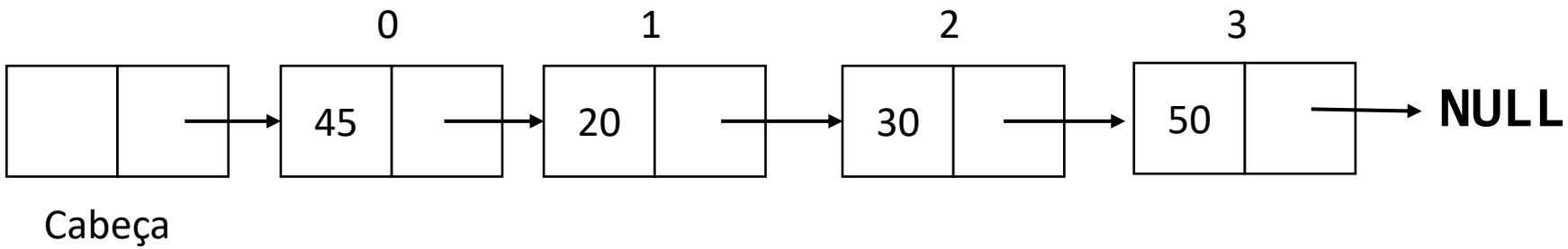
Implementação – Adicionar entre elementos

addAt(50, 2)

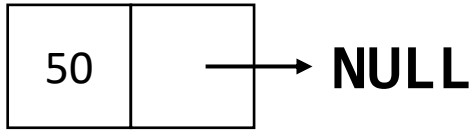


Implementação – Adicionar entre elementos

addAt(50, 2)

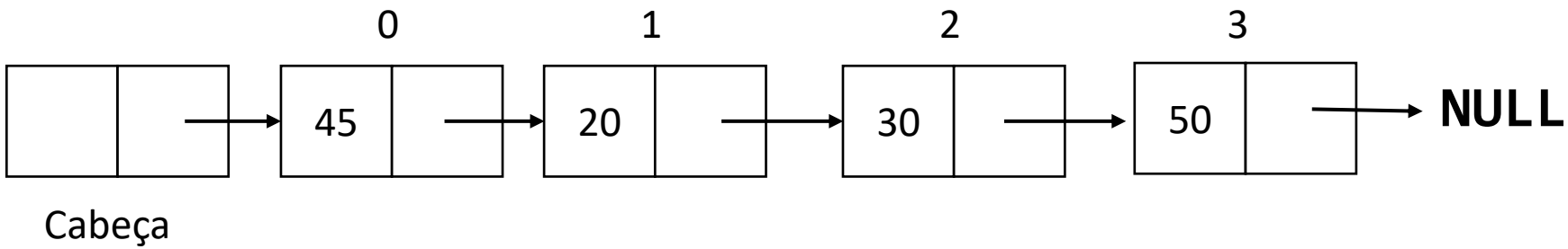


1. Instanciar o nó

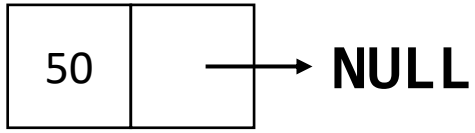


Implementação – Adicionar entre elementos

addAt(50, 2)

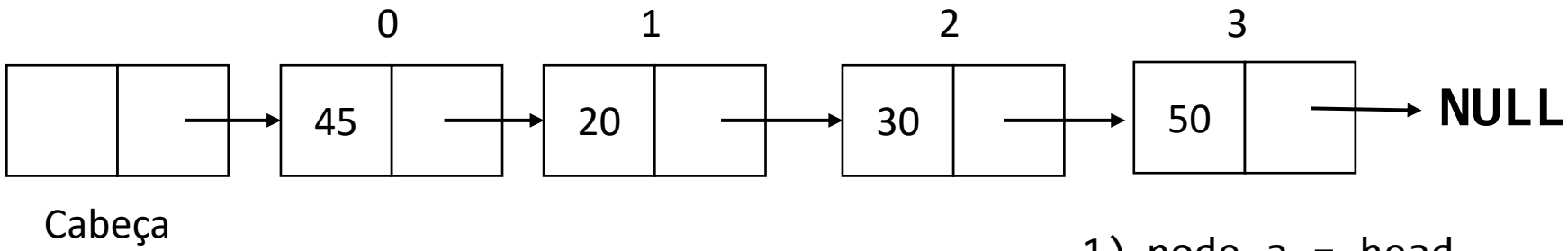


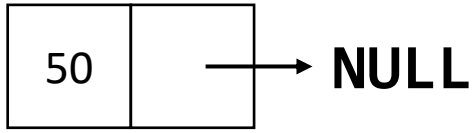
2) Iterar sobre a lista para encontrar a posição onde o elemento será adicionado.



Implementação – Adicionar entre elementos

addAt(50, 2)



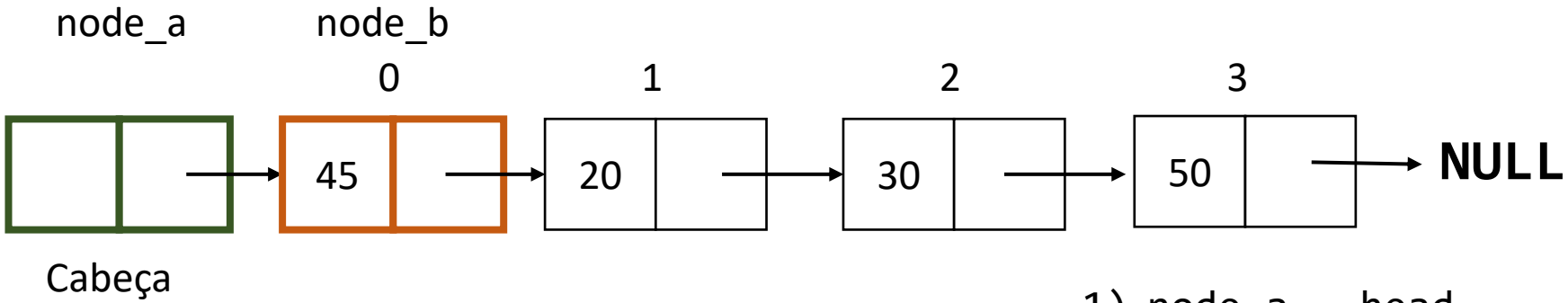


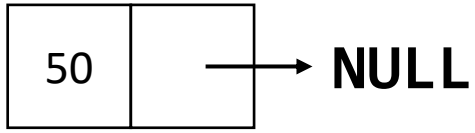

```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

Implementação – Adicionar entre elementos

addAt(50, 2)



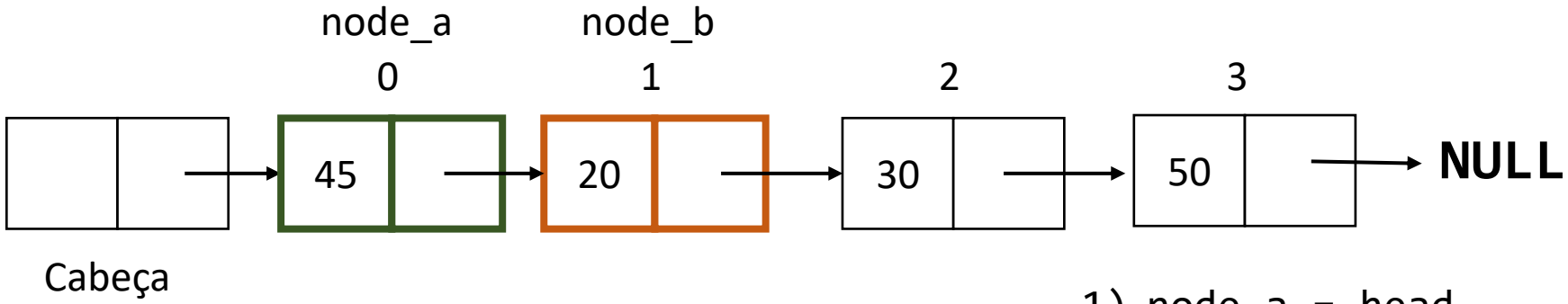


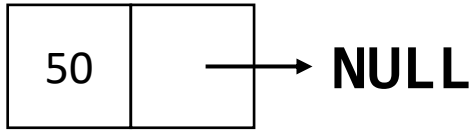

```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

Implementação – Adicionar entre elementos

addAt(50, 2)



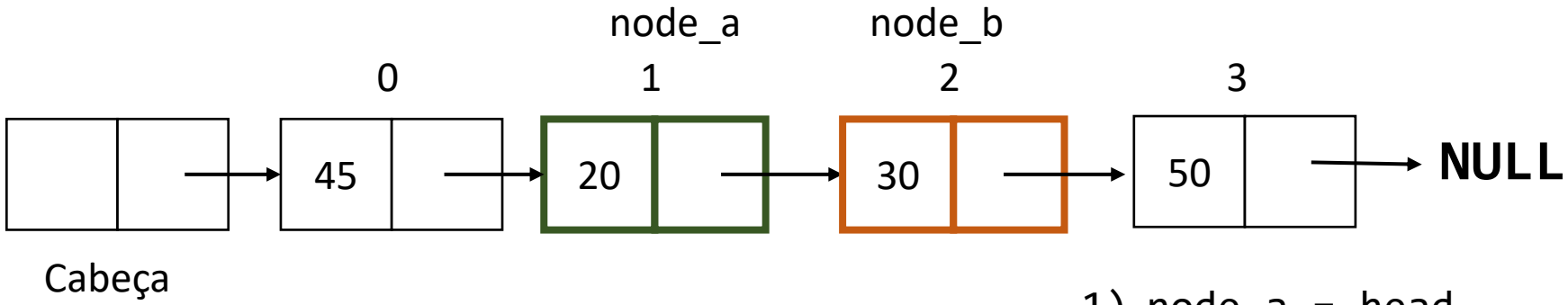


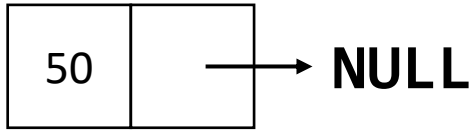

```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

Implementação – Adicionar entre elementos

addAt(50, 2)



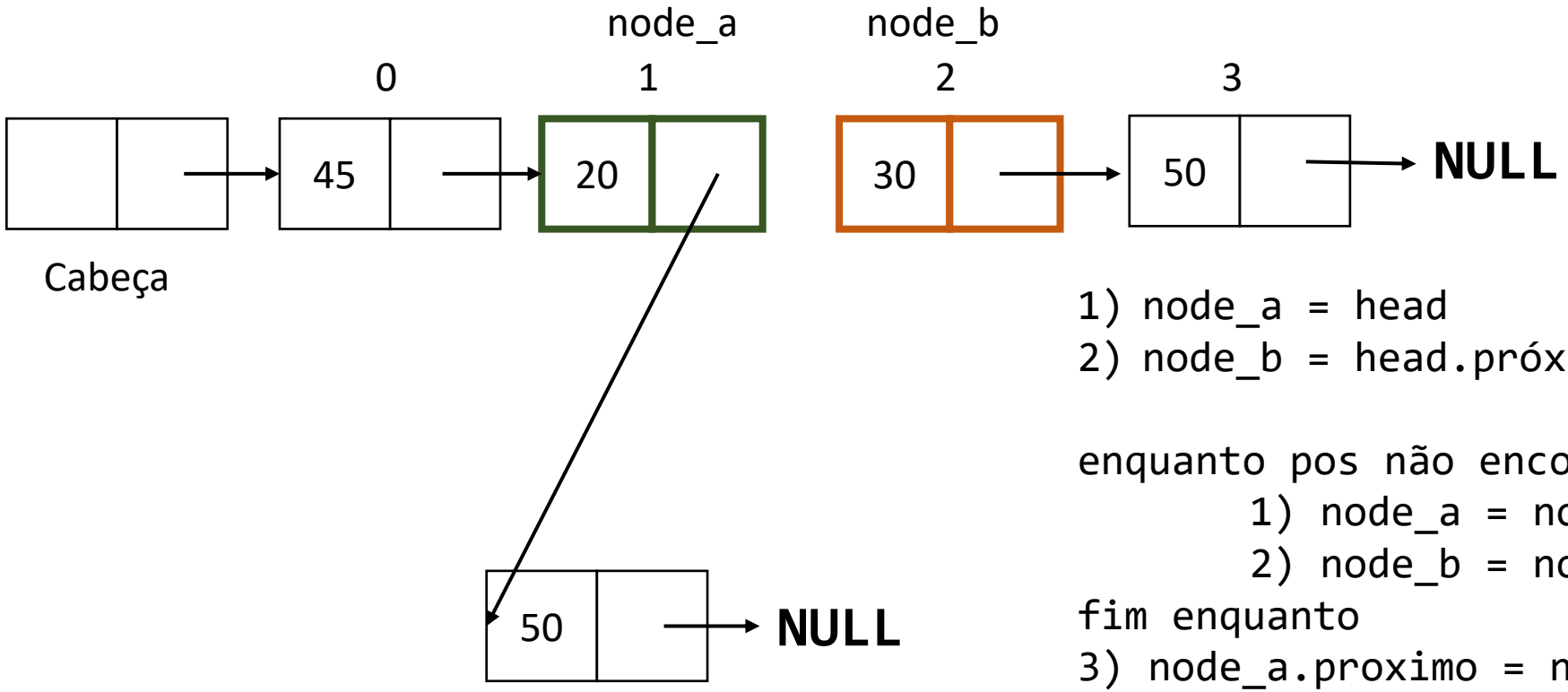



```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

Implementação – Adicionar entre elementos

addAt(50, 2)

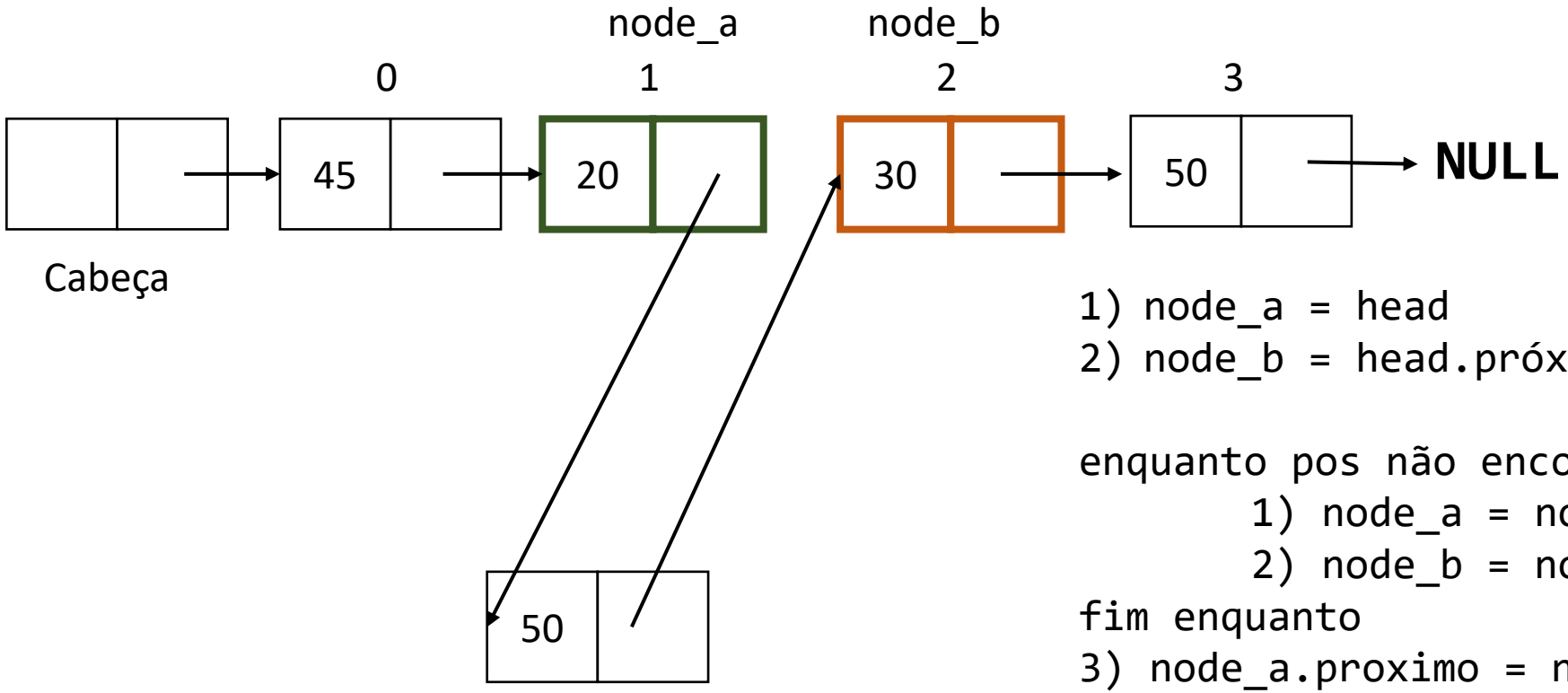



```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

Implementação – Adicionar entre elementos

addAt(50, 2)

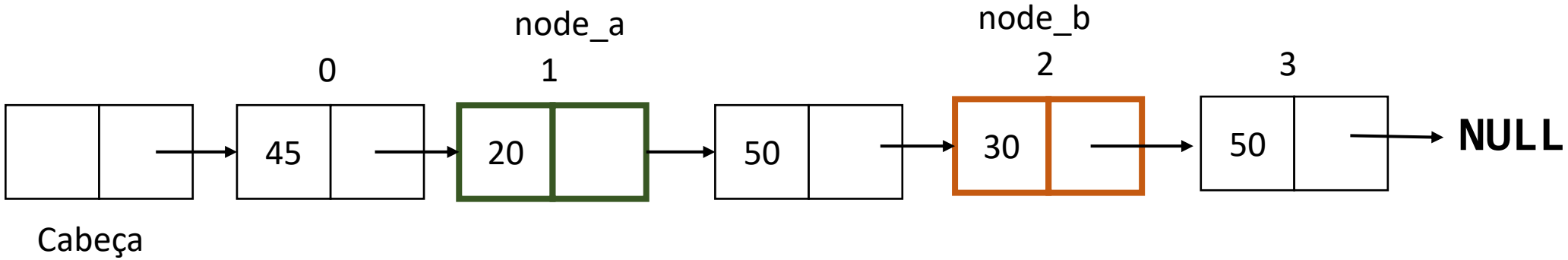



```
1) node_a = head
2) node_b = head.próximo

enquanto pos não encontrada faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = novo_no
4) novo_no.próximo = node_b
```

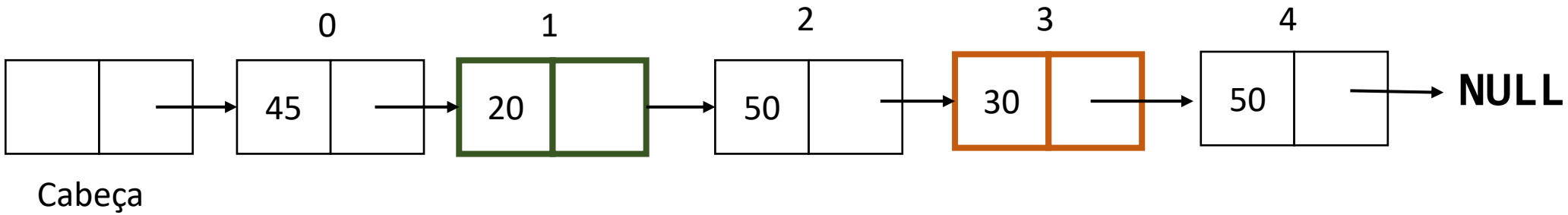
Implementação – Adicionar entre elementos

addAt(50, 2)



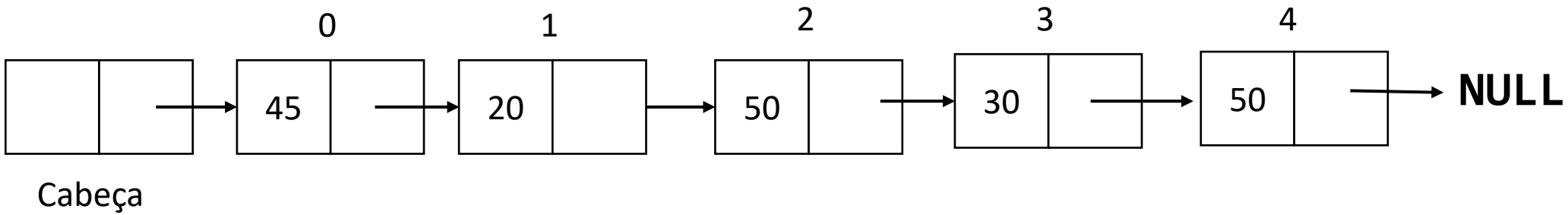
Implementação – Adicionar entre elementos

addAt(50, 2)



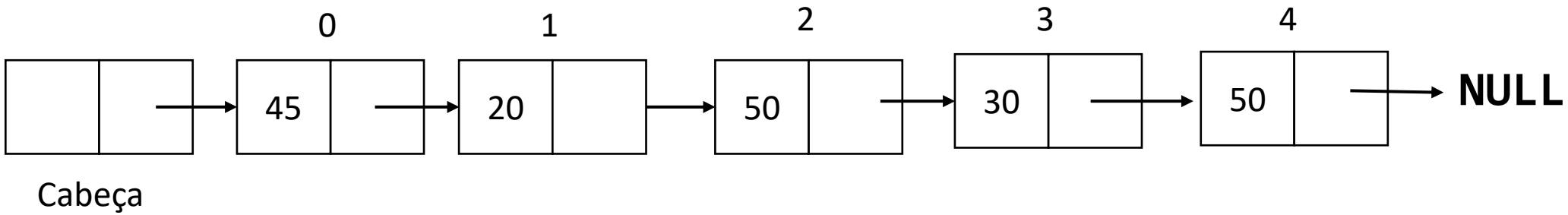
Implementação – Adicionar entre elementos

addAt(50, 2)



Implementação – Remover entre elementos

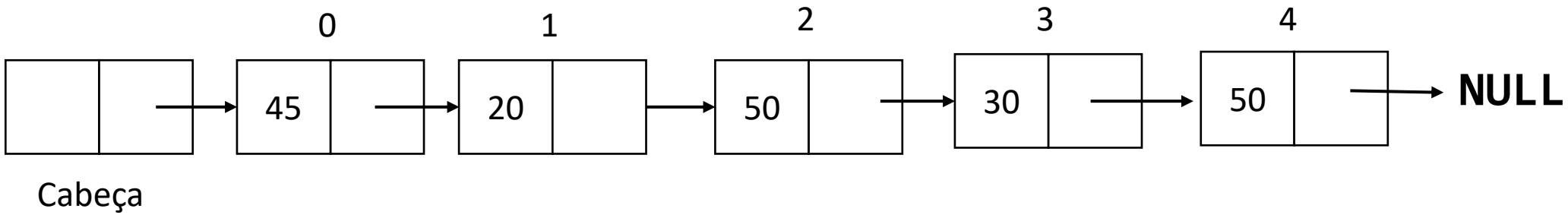
remove(50)



- Em algumas situações precisamos remover um elemento entre outros elementos.

Implementação – Remover entre elementos

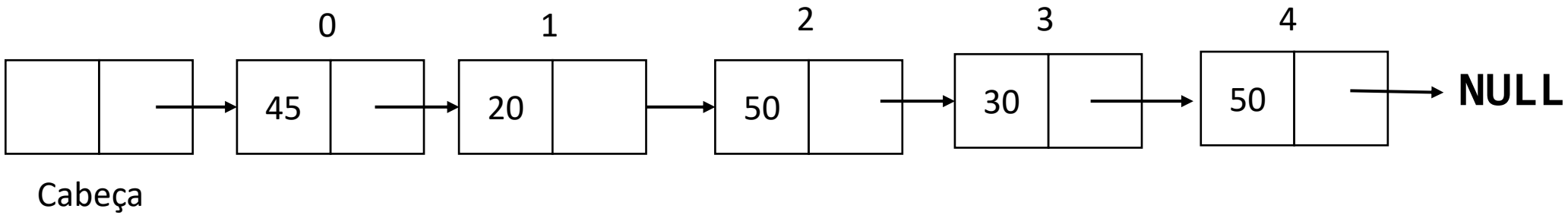
remove(50)



- Em algumas situações precisamos remover um elemento entre outros elementos.

Implementação – Remover entre elementos

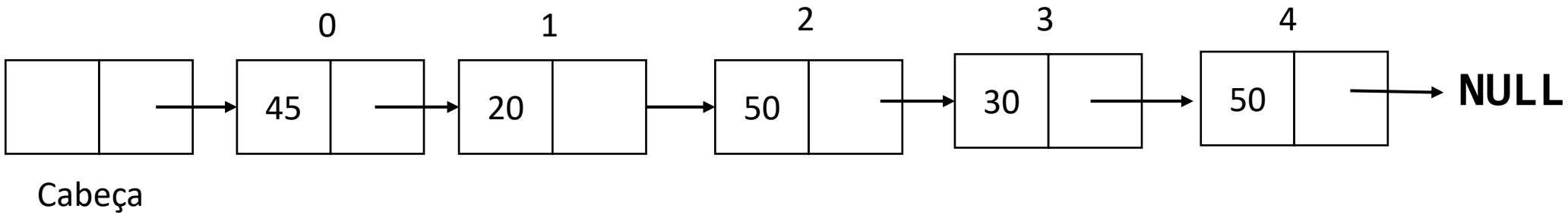
remove(50)



1) Iterar sobre a lista para encontrar a posição onde o elemento será removido e atualizar os ponteiros

Implementação – Remover entre elementos

remove(50)

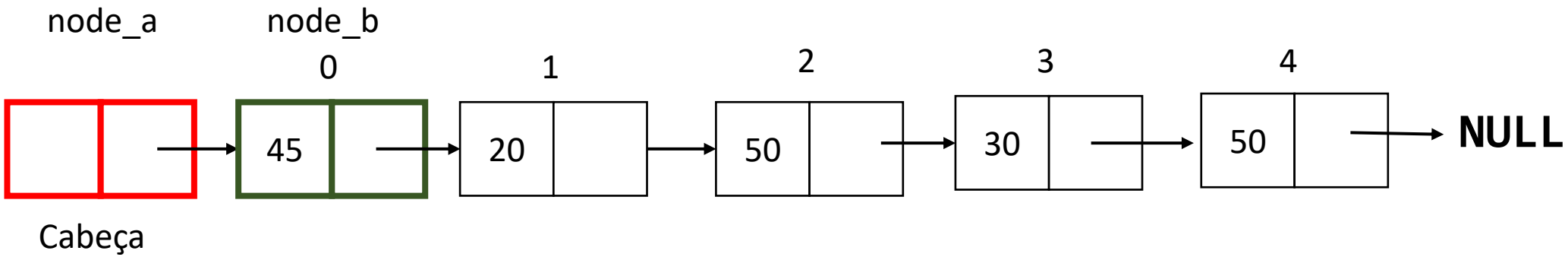



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)

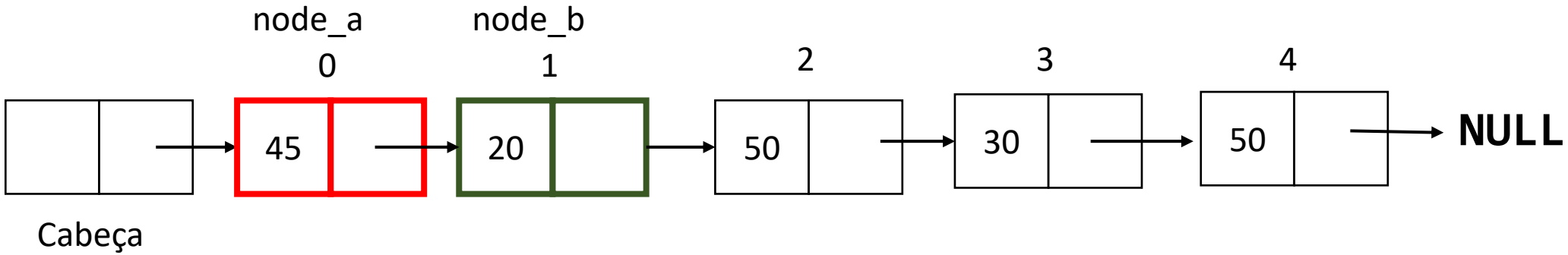



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)

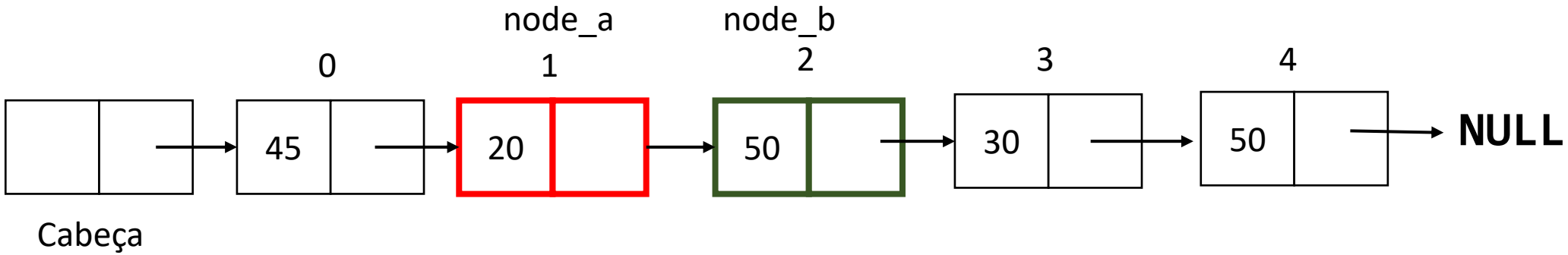



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)

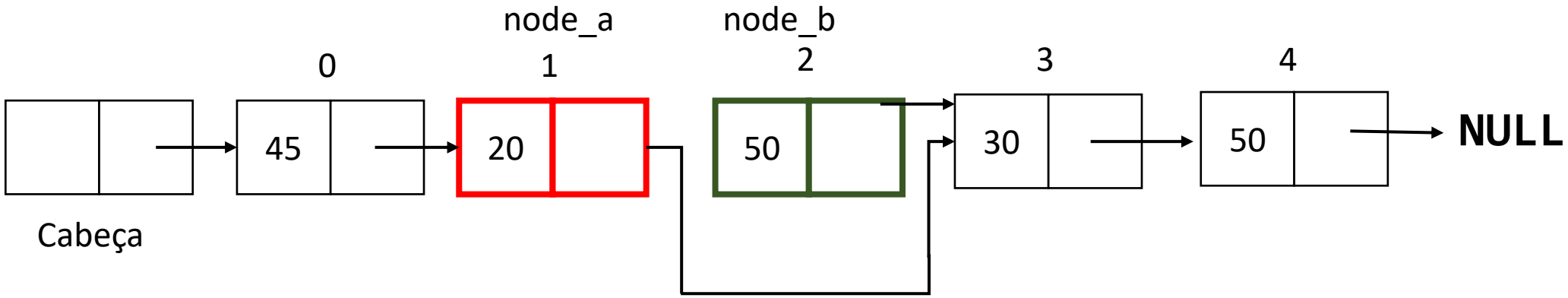



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)

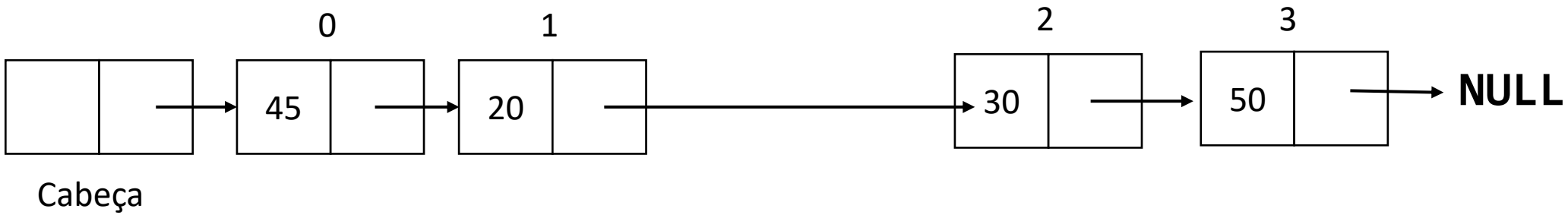



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)

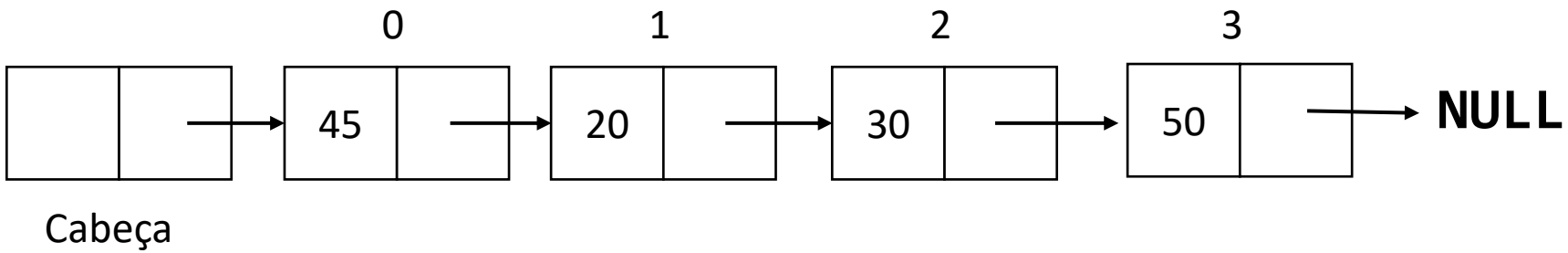



```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Implementação – Remover entre elementos

remove(50)




```
1) node_a = head
2) node_b = head.próximo

enquanto não encontrado faça
        1) node_a = node_b
        2) node_b = node_b.próximo
fim enquanto
3) node_a.proximo = node_b.proximo
```

Bibliografia Básica

- CORMEN, Thomas H et al. **Algoritmos: teoria e prática.** Rio de Janeiro: Elsevier, 2012. 926 p. ISBN: 9788535236996.
- ASCENCIO, Ana Fernanda Gomes. **Estruturas de dados: algoritmos, análise da complexidade e implementações em Java e C/C++.** São Paulo: Pearson, c2010. 432 p. ISBN: 9788576052216, 97885760581 6.
- PIVA JÚNIOR, Dilermando (et al). **Estrutura de dados e técnicas de programação**. 1. ed. Rio de Janeiro, RJ: Campus, 2014. 399 p. ISBN: 9788535274370.

Bibliografia Complementar

- FERRARI, Roberto et al. **Estruturas de dados com jogos**. 1. ed. Rio de Janeiro: Elsevier, 2014. 259p. ISBN: 9788535278040.
- GRONER, Loiane. **Estruturas de dados e algoritmos em Javascript**: aperfeiçoe suas habilidades conhecendo estruturas de dados e algoritmos clássicos em JavaScript. São Paulo: Novatec, 2017. 302 p. ISBN: 9788575225530.
- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro: LTC, 2010. xv, 302 p. ISBN: 9788521617501.
- GOODRICH, Michael T; TAMASSIA, Roberto. **Estruturas de dados e algoritmos em Java**. 5. ed. Porto Alegre: Bookman, 2013. xxii, 713 p. ISBN: 9788582600184.
- GUIMARÃES, Ângelo M. **Algoritmos e estruturas de dados**. LTC, 1994.