

Algoritmos e Programação II

Prof. MSc. Samuel Benjoino Ferraz Aquino



Módulo 1 – Unidade 2

Variáveis compostas heterogêneas (registros)

Prof. MSc. Samuel Benjoino Ferraz Aquino



Roteiro

- Variável simples heterogênea
 - Criando um tipo heterogêneo
 - Declarando e acessando uma variável heterogênea
 - Ponteiro
- Variável composta heterogênea

Parte 1

- Variável simples heterogênea
 - Criando um tipo heterogêneo
 - Declarando e acessando uma variável heterogênea
 - Ponteiro



Variável Simples Heterogênea

- Variável capaz de armazenar uma única informação
- Podem ser de dois tipos:
 - *Homogênea*
 - *Heterogênea*

Variável Simples Heterogênea

- Variável capaz de armazenar uma **única informação**, que pode ser dividida em “pedaços” de **diferentes tipos**.
- Antes de declarar uma variável simples heterogênea, é necessário definir quais serão os “pedaços” dessa variável.

Criando um Tipo Heterogêneo

- Em *Python*, para criarmos uma variável simples heterogênea precisamos criar uma **classe**
- **Classe**
 - Define quais os “pedaços” da sua variável heterogênea
 - Usaremos de maneira simplificada nesse módulo

Criando um Tipo Heterogêneo

- Exemplo: uma variável chamada *n* que precisa armazenar o *nome*, a *nota 1* e a *nota 2* de um aluno.
- Criaremos uma **classe** para definir o que a variável *n* deve conter

<i>class</i>				<i>Nota:</i>
<i>def</i>	<i>__init__(self,</i>	<i>nome,</i>	<i>nota1,</i>	<i>nota2):</i>
	<i>self.nome</i>	<i>=</i>		<i>nome</i>
	<i>self.nota1</i>	<i>=</i>		<i>nota1</i>
	<i>self.nota2 = nota2</i>			

Criando um Tipo Heterogêneo

class

```
def __init__(self, nome, nota1,
    self.nome = nome
    self.nota1 = nota1
    self.nota2 = nota2
```

Nota:
nota2):
nome
nota1

- **class:** Palavra-chave usada para a criação de qualquer class

Criando um Tipo Heterogêneo

```
class
    def      __init__(self,      nome,      nota1,      nota2):
        self.nome      =      nome
        self.nota1      =      nota1
        self.nota2 = nota2
```

- **Nota:** Nome da sua classe. Deve sempre aparecer após a palavra-chave *class*.

Criando um Tipo Heterogêneo

```
class
    def __init__(self, nome, nota1, nota2):
        self.nome = nome
        self.nota1 = nota1
        self.nota2 = nota2
```

- **def __init__**: Define um **método** (equivalente a uma função) que dirá quais os “pedaços” da variável nota.

Criando um Tipo Heterogêneo

<i>class</i>				<i>Nota:</i>
<i>def</i>	<i>__init__</i>	(<i>self</i> ,	<i>nome,</i>	<i>nota1,</i>
				<i>nota2):</i>
	<i>self.nome</i>		<i>=</i>	<i>nome</i>
	<i>self.nota1</i>		<i>=</i>	<i>nota1</i>
	<i>self.nota2 = nota2</i>			

- ***self***: Deve sempre ser o primeiro parâmetro de qualquer método. Vocês entenderão em mais detalhes quando estudarem sobre Orientação a Objetos.

Criando um Tipo Heterogêneo

<i>class</i>				<i>Nota:</i>
<i>def</i>	<i>__init__(self,</i>	<i>nome,</i>	<i>nota1,</i>	<i>nota2):</i>
	<i>self.nome</i>	<i>=</i>		<i>nome</i>
	<i>self.nota1</i>	<i>=</i>		<i>nota1</i>
	<i>self.nota2 = nota2</i>			

- ***nome, nota1, nota2***: Os nomes dos “pedaços” da sua variável, separados por vírgula.
 - Esses “pedaços” são chamados de **atributos**.

Criando um Tipo Heterogêneo

```
class
    def      __init__(self,      nome,      nota1,      Nota:
        self.nome                =      nome
        self.nota1               =      nota1
        self.nota2 = nota2
```

- Essas três linhas “declaram” os atributos de uma Nota.
 - Vocês entenderão em mais detalhes quando estudarem sobre Orientação a Objetos.

Criando um Tipo Heterogêneo

```
class
    def      __init__(self,      nome,      nota1,      nota2):
        self.nome      =      nome
        self.nota1      =      nota1
        self.nota2 = nota2
```

- Pronto, já definimos o que uma variável **Nota** deve ter.
- Entretanto, ainda não declaramos a variável:
 - Apenas definimos o seu “esqueleto”.

Declarando e Acessando uma Variável Heterogênea

- Mãos à obra!

Ponteiro

- O que acontece quando declaramos uma variável de um tipo heterogêneo
- Exemplo:
 - $n = \text{Nota}(\text{"Ziguifrido"}, 3.5, 4.2)$
- O que a variável n armazena?

Ponteiro

- Toda variável heterogênea armazena o **endereço** dos seus atributos na memória
- Uma variável que armazena um endereço é chamada de **ponteiro**
- Exemplo:
 - $n = \text{Nota}(\text{"Ziguifrido"}, 3.5, 4.2)$

- Exemplo:

- $n = \text{Nota}(\text{"Ziguifrido"}, 3.5, 4.2)$

Nome

Nota 1

Nota 2

Endereço

Conteúdo

0	
1	
2	"Ziguifrido"
3	3.5
4	4.2
5	
6	
7	

- Exemplo:
 - $n = \text{Nota}(\text{"Ziguifrido"}, 3.5, 4.2)$
- Onde está a variável n na memória?!

Nome

Nota 1

Nota 2

Endereço

Conteúdo

0	
1	
2	"Ziguifrido"
3	3.5
4	4.2
5	
6	
7	

- Exemplo:
 - n = Nota (“Ziguifrido”, 3.5, 4.2)
- Onde está a variável n na memória?

Nome

Nota 1

Nota 2

n

Endereço

Conteúdo

0	
1	
2	“Ziguifrido”
3	3.5
4	4.2
5	
6	
7	

- Exemplo:
 - $n = \text{Nota}(\text{"Ziguifrido"}, 3.5, 4.2)$
- Onde está a variável n na memória?
 - O que ela contém?

Nome

Nota 1

Nota 2

n

Endereço

Conteúdo

0	
1	
2	"Ziguifrido"
3	3.5
4	4.2
5	
6	??
7	

- Exemplo:
 - n = Nota (“Ziguifrido”, 3.5, 4.2)
- Onde está a variável n na memória?
 - O que ela contém?
 - O endereço de início do seu conteúdo!

Nome

Nota 1

Nota 2

n

Endereço

Conteúdo

0	
1	
2	“Ziguifrido”
3	3.5
4	4.2
5	
6	2
7	

Resumo

- Variáveis heterogêneas são divididas em “pedaços”
- O conteúdo de uma variável heterogênea é definido em uma **classe**
- Os “pedaços” de uma variável heterogênea são chamados de **atributos**
- Uma variável heterogênea armazena um **endereço**

Parte 2

- Variável composta heterogênea



Variável Composta

- Variável capaz de armazenar um **conjunto** de informações
- Exemplo: vetores de inteiros

v = *[* *]*

v.append(10)
v.append(15)
v.append(8)

Variável Composta Heterogênea

- Variável capaz de armazenar um **conjunto** de informações **heterogêneas**
- Exemplo: vetores de notas

```
v = [  
v.append(Nota("Samuel",6,  
v.append(Nota("Wallace",7, 8))7))
```

Variável Composta Heterogênea

- Exemplo: vetores de notas

```
v =  
v.append(Nota("Samuel",4,  
v.append(Nota("Wallace",7,
```

- O que v armazena?

Endereço	Conteúdo
0	
1	
2	
3	
4	
5	
6	
7	

[5))
8))]

Variável Composta Heterogênea

- Exemplo: vetores de notas

v

v =

```
v.append(Nota("Samuel",4,  
v.append(Nota("Wallace",7,
```

- O que *v* armazena?
 - O endereço do início do vetor!
 - v* é um **ponteiro!**

Endereço

Conteúdo

0	2
1	
2	
3	
4	
5	
6	
7	

[5))
8))

Variável Composta Heterogênea

- Exemplo: vetores de notas

v =

*$v.append(Nota("Samuel",4,$
 $v.append(Nota("Wallace",7,$*

v

$v[0]$

- O que v armazena?
 - Um conjunto de ponteiros!

Endereço

Conteúdo

0	2
1	
2	5
3	
4	
5	"Samuel"
6	4
7	5

$[$ $5))$
 $8))$ $]$

Variável Composta Heterogênea

- Exemplo: vetores de notas

v =

`v.append(Nota("Samuel",4,
v.append(Nota("Wallace",7,`

v

$v[0]$

$v[1]$

- O que v armazena?
 - Um conjunto de ponteiros!

Endereço

Conteúdo

0	2
1	
2	5
3	25
4	
5	"Samuel"
6	4
7	5

[5))
8))

Resumo

- Variáveis compostas heterogêneas são conjuntos de variáveis heterogêneas.
- Uma variável composta (vetor) também é um ponteiro.
- Uma variável composta heterogênea armazena um conjunto de ponteiros.

Listas lineares

- Uma **lista linear** é um conjunto de $n \geq 0$ **nós/elementos** organizados de acordo com as suas **posições** dentro da lista.

"Alberto"	10	10.5	João	45
0	1	2	3	4

Listas lineares

- Principais operações em uma lista linear
 - **Busca**
 - **Inserção**
 - **Remoção**
- Custo dessas operações depende da implementação da lista linear
 - Lista linear com **alocação sequencial**
 - Lista linear com **alocação encadeada**

Lista linear com alocação encadeada

- Alocar as posições da lista **sob demanda** e de maneira **esparsa** na memória física.
- Ou seja, dada uma lista linear **L** com alocação encadeada, os seus elementos estão **espalhados** na memória física.
- Implementaremos uma **lista simplesmente encadeada**.

Lista simplesmente encadeada

- Uma posição de uma lista simplesmente encadeada é chamada de **nó**
- Cada **nó** contém:
 - Conteúdo (*chave*)
 - Endereço da próxima posição (*prox*)
- Um **nó** é uma **variável simples heterogênea**

Lista simplesmente encadeada

- Exemplo: lista simplesmente encadeada **L** com 3 elementos

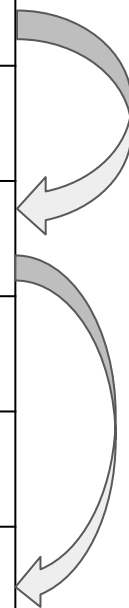
L = []

L.append(10)

L.append(20)

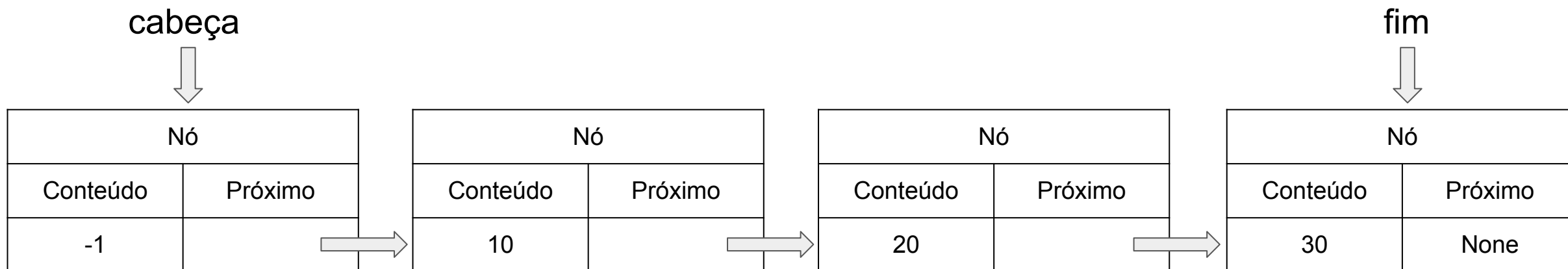
L.append(30)

	<i>Endereço</i>	<i>Conteúdo</i>
<i>L</i>	0	10
	1	
	2	20
	3	
	4	
	5	30



Lista simplesmente encadeada

- Como esperarmos uma lista na memória?



Lista simplesmente encadeada

- Declarando um nó

class No:

```
def __init__(self, conteudo, proximo):  
    self.conteudo = conteudo  
    self.proximo = proximo
```

Nó	
Conteúdo	Próximo

Lista simplesmente encadeada

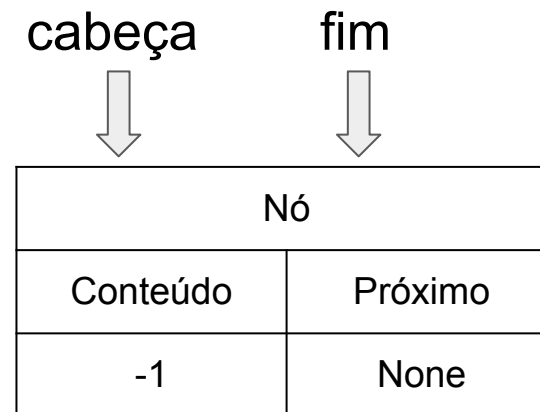
- Declarando uma lista simplesmente encadeada

class Lista:

def __init__(self):

self.cabeca = No(-1, None)

self.fim = self.cabeca

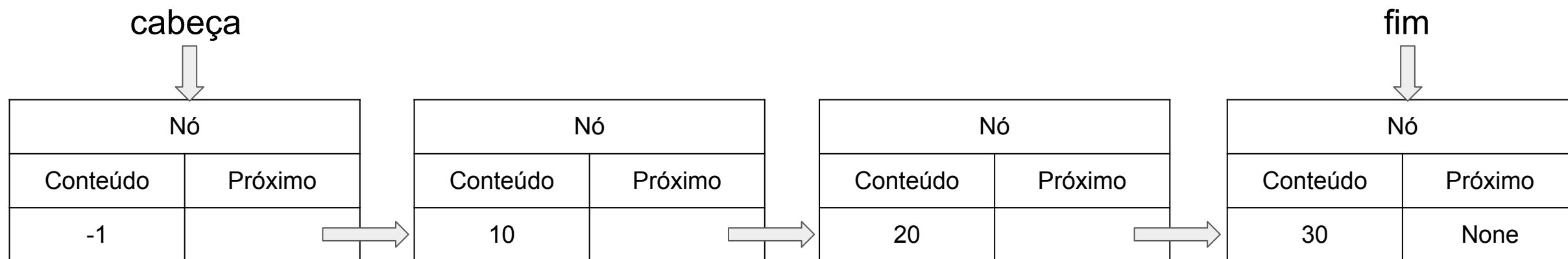


Lista linear com alocação encadeada

- Considere as seguintes operações em uma lista linear **L** com alocação encadeada:
 - **Imprimir** o conteúdo da lista
 - **Inserir** um elemento **x** no final da lista
 - **Buscar** um elemento **x**
 - **Remover** um elemento **x**

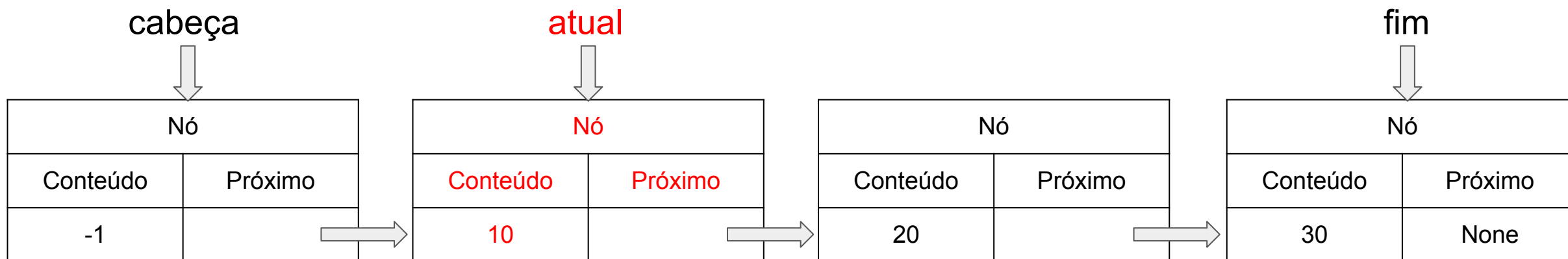
Lista simplesmente encadeada

- Imprimir o conteúdo da lista



Lista simplesmente encadeada

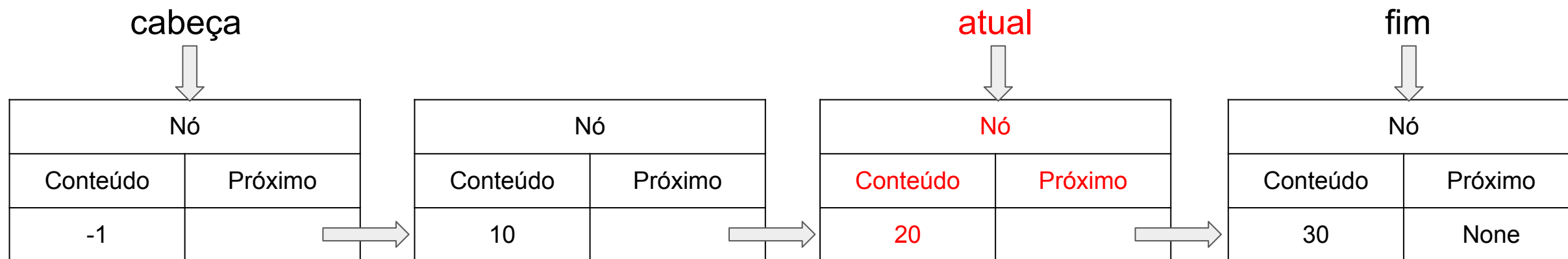
- Imprimir o conteúdo da lista



atual = self.cabeça.proximo

Lista simplesmente encadeada

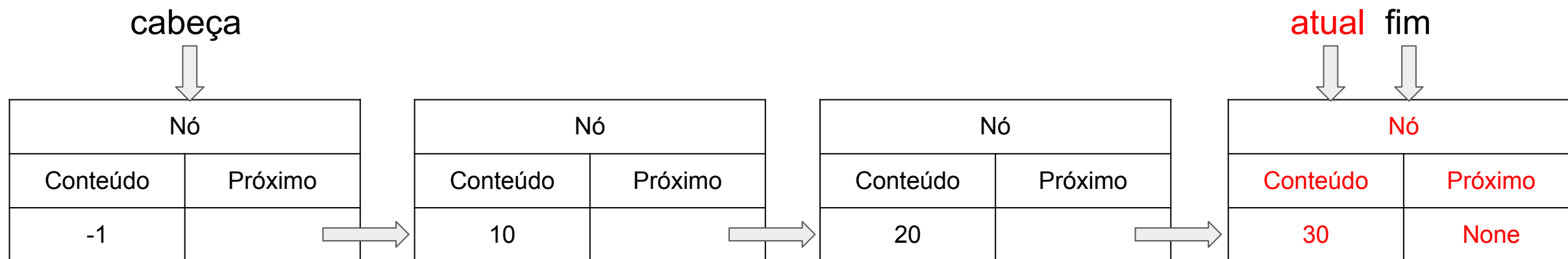
- Imprimir o conteúdo da lista



atual = atual.proximo

Lista simplesmente encadeada

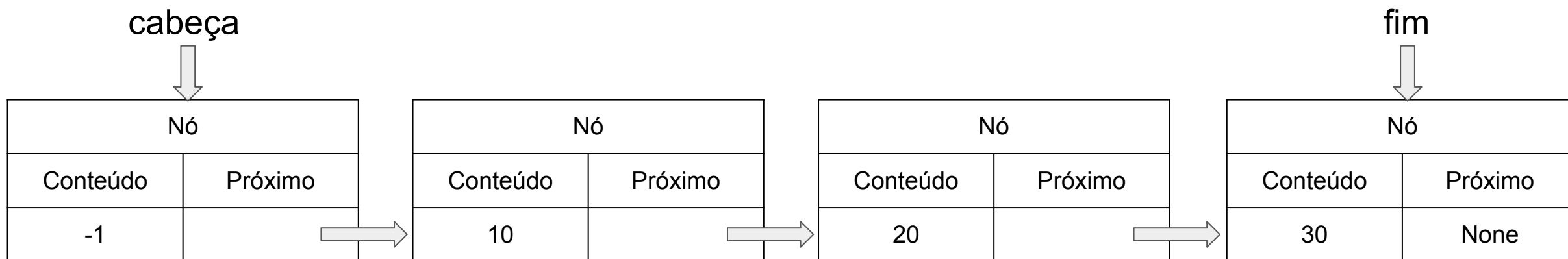
- Imprimir o conteúdo da lista



atual = atual.proximo

Lista simplesmente encadeada

- Imprimir o conteúdo da lista



atual = atual.proximo

atual
↓
None

Lista simplesmente encadeada

- **Imprimir** o conteúdo da lista

```
class Lista:
```

```
...
```

```
def imprime(self):
```

```
    atual = self.cabeca.proximo
```

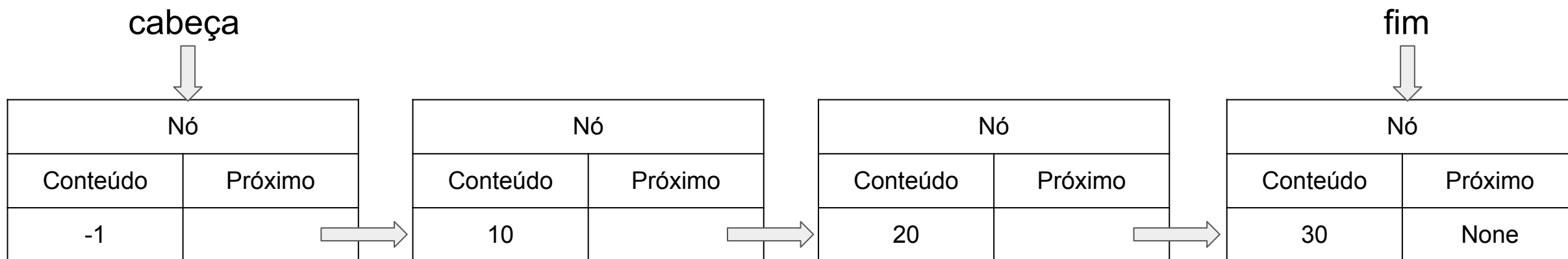
```
    while(atual != None):
```

```
        print(atual.conteudo)
```

```
        atual = atual.proximo
```

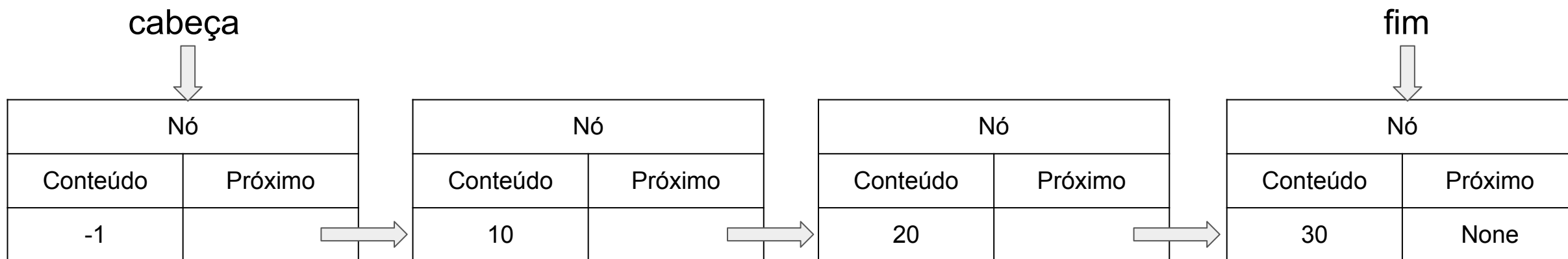
Lista simplesmente encadeada

- Inserir um elemento **x** no final da lista

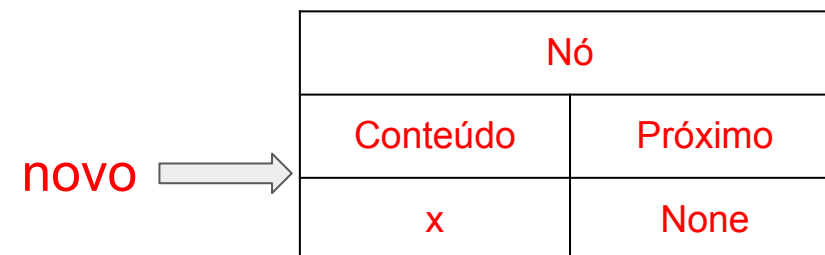


Lista simplesmente encadeada

- Inserir um elemento x no final da lista

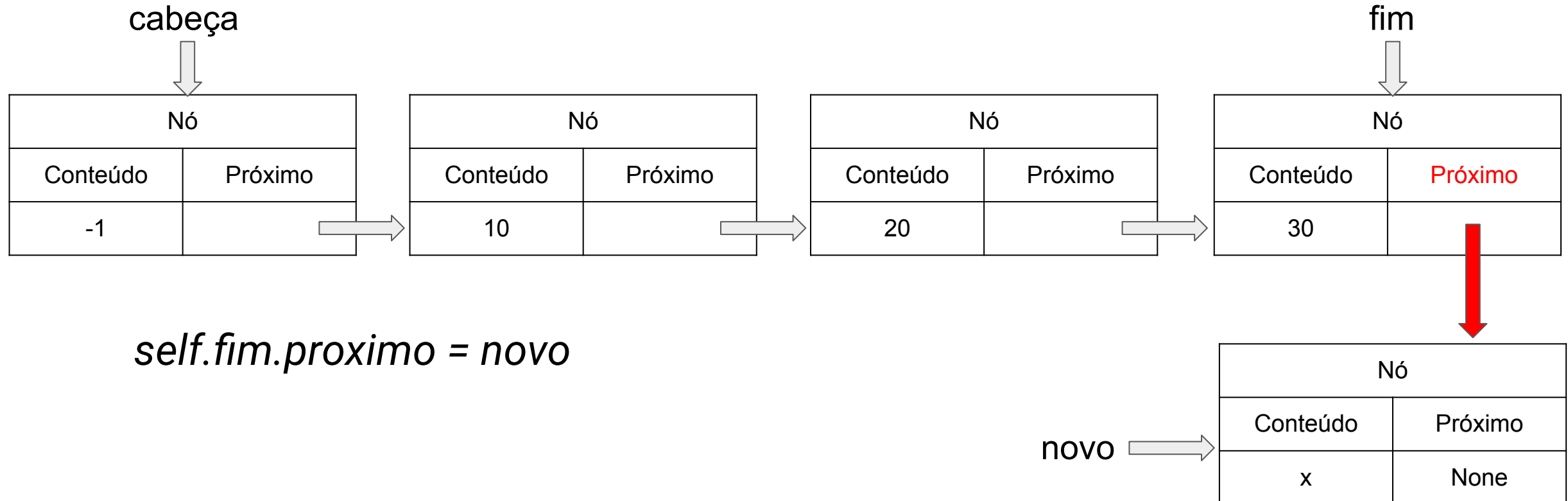


$novo = No(x, None)$



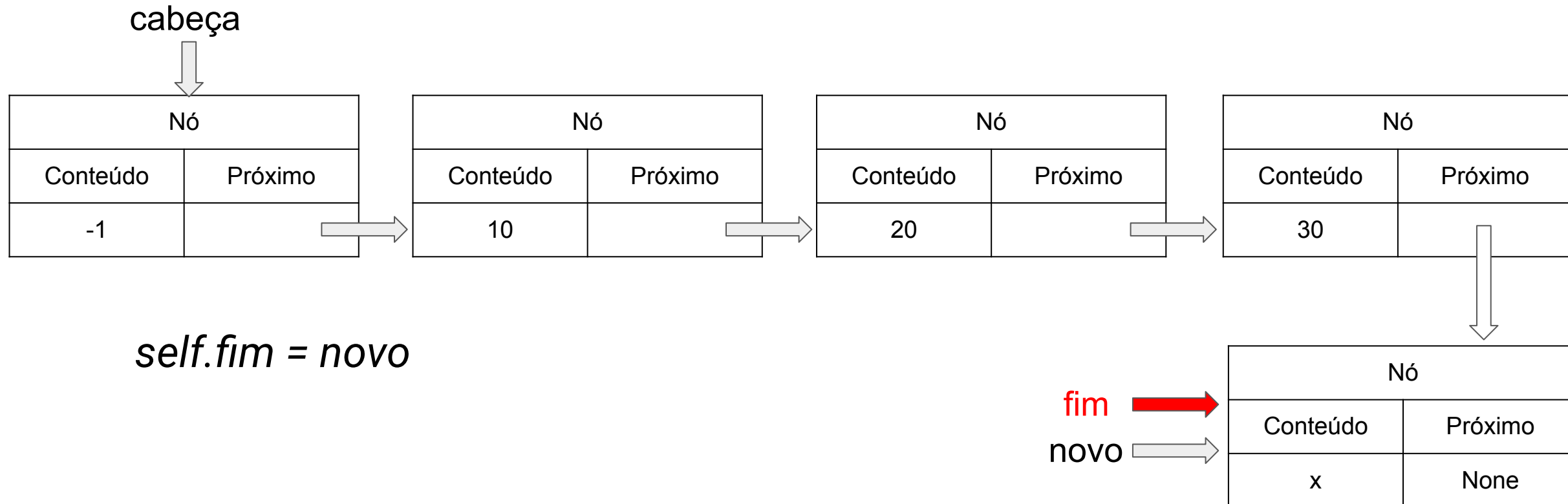
Lista simplesmente encadeada

- Inserir um elemento **x** no final da lista



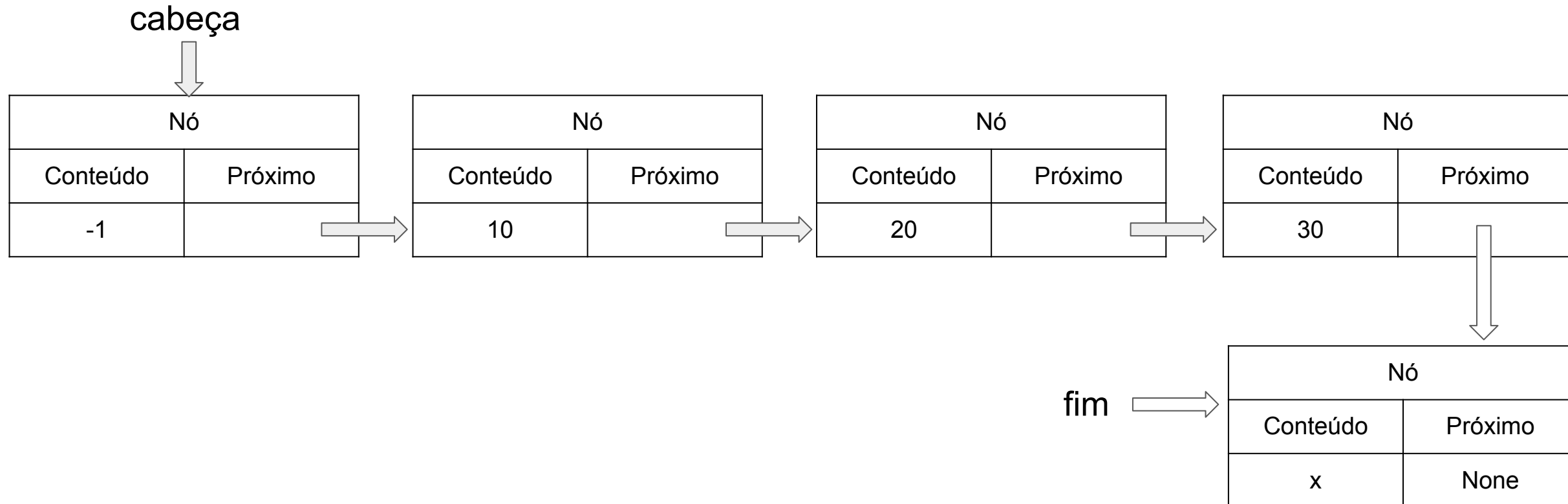
Lista simplesmente encadeada

- Inserir um elemento **x** no final da lista



Lista simplesmente encadeada

- Inserir um elemento **x** no final da lista



Lista simplesmente encadeada

- **Inserir** um elemento **x** no final da lista

```
class Lista:
```

```
...
```

```
def insereNoFim(self, x):
```

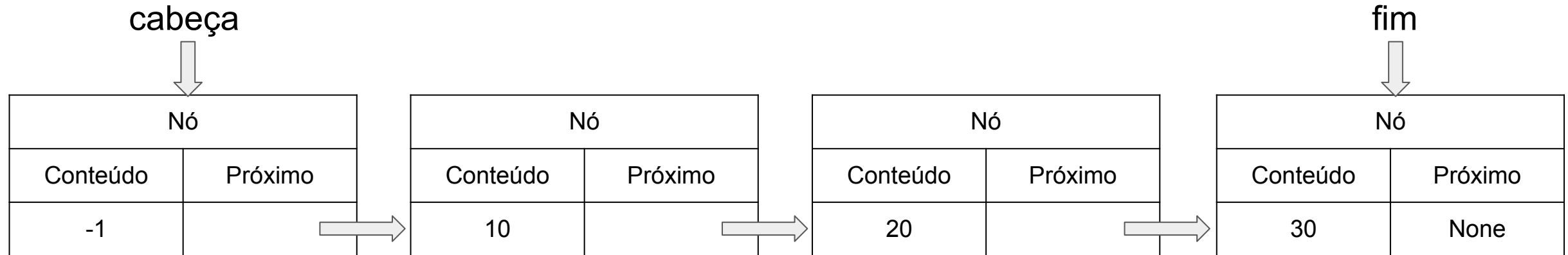
```
    novo = No(x, None)
```

```
    self.fim.proximo = novo
```

```
    self.fim = novo
```

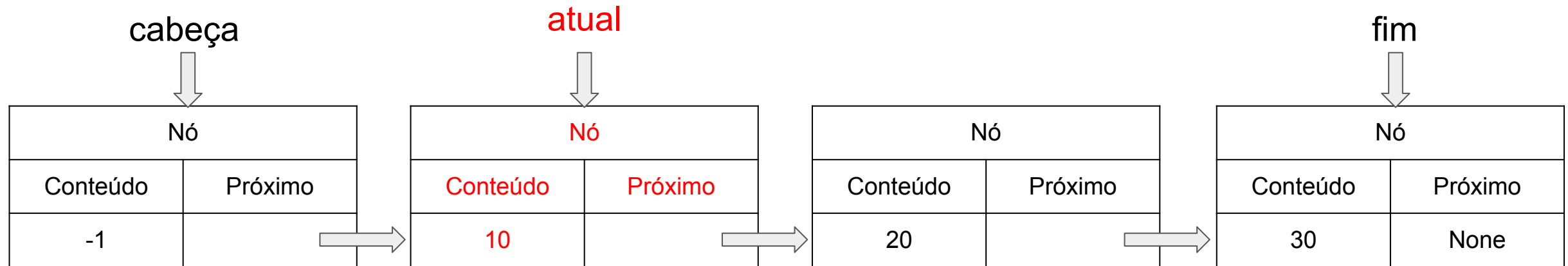
Lista simplesmente encadeada

- **Buscar um elemento x**



Lista simplesmente encadeada

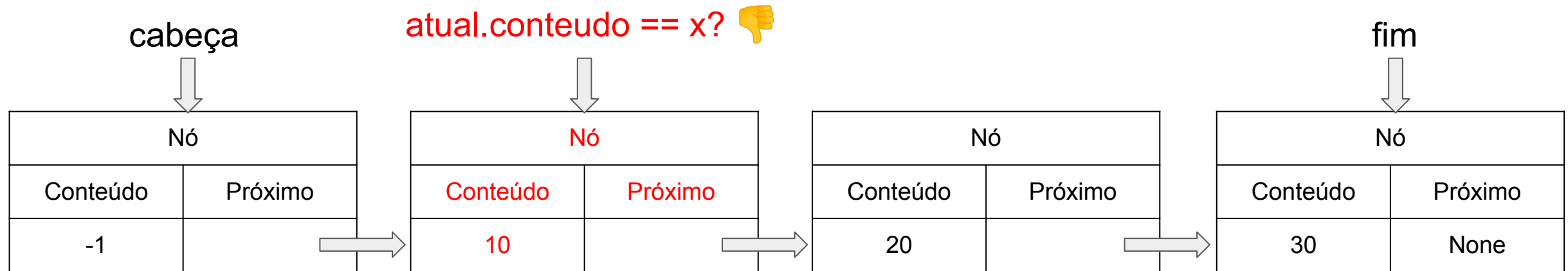
- **Buscar um elemento x**



atual = self.cabeça.proximo

Lista simplesmente encadeada

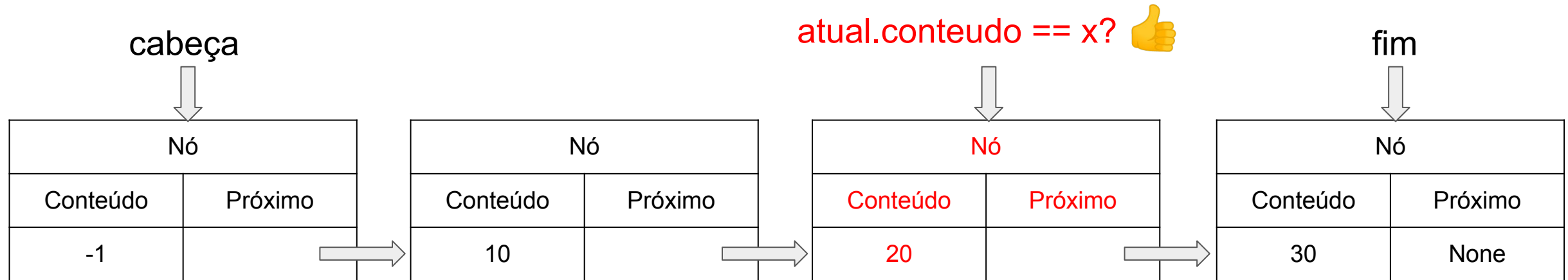
- **Buscar um elemento x**



atual = atual.proximo

Lista simplesmente encadeada

- **Buscar um elemento x**



Lista simplesmente encadeada

- **Buscar** um elemento **x**

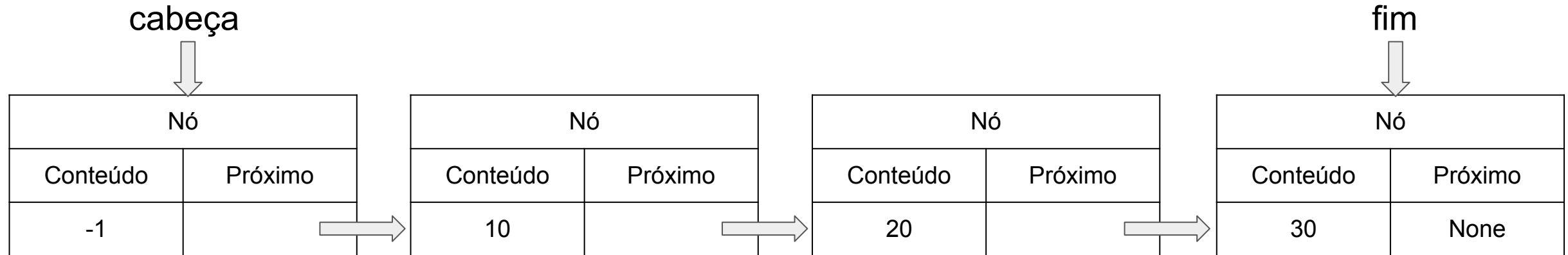
```
class Lista:
```

```
...
```

```
    def buscar(self, x):  
        atual = self.cabeca.proximo  
        while(atual != None):  
            if atual.conteudo == x:  
                return True  
            atual = atual.proximo  
        return False
```

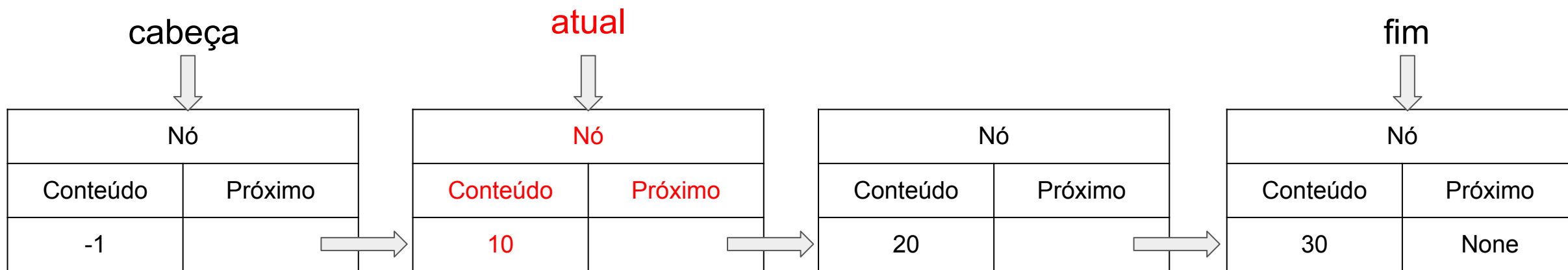
Lista simplesmente encadeada

- **Remover um elemento x (20)**



Lista simplesmente encadeada

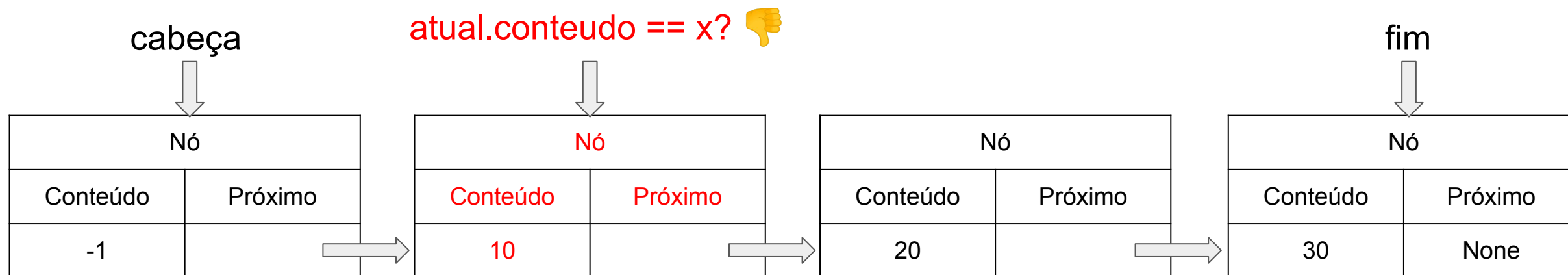
- Remover um elemento x (20)



atual = self.cabeça.proximo

Lista simplesmente encadeada

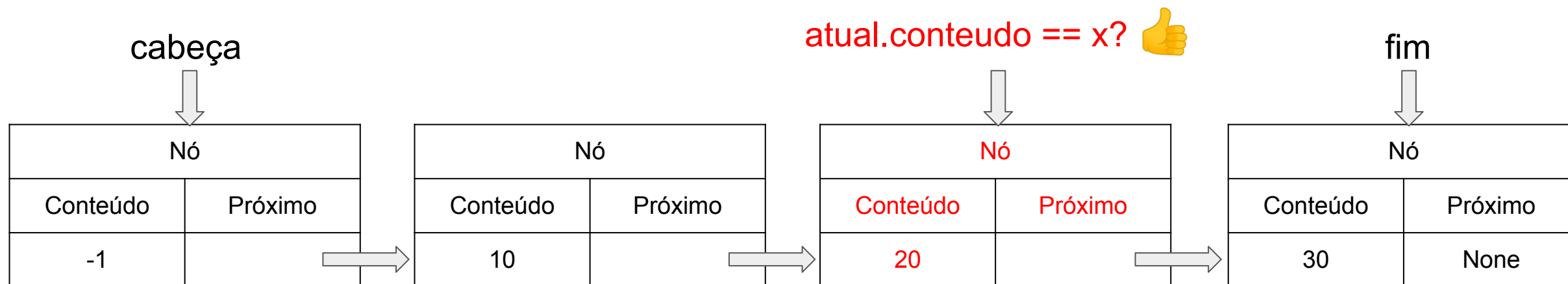
- Remover um elemento x (20)



atual = atual.proximo

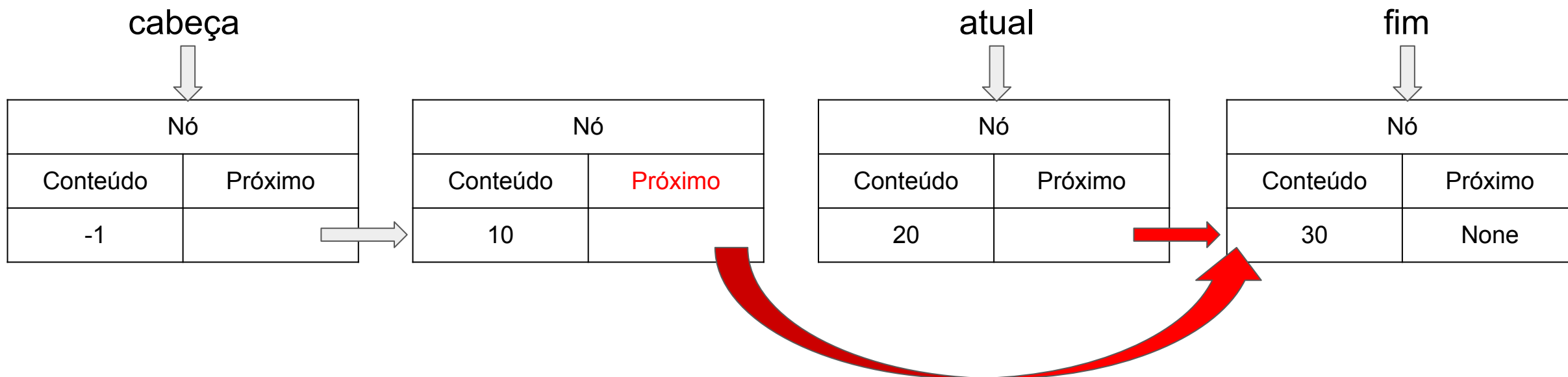
Lista simplesmente encadeada

- Remover um elemento x (20)



Lista simplesmente encadeada

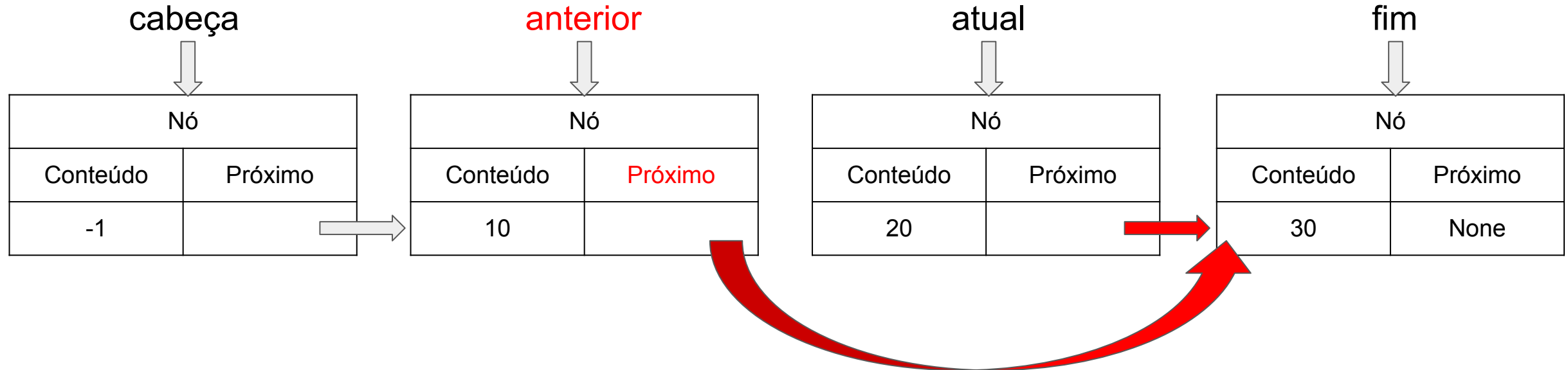
- Remover um elemento x (20)



anterior.proximo = atual.proximo

Lista simplesmente encadeada

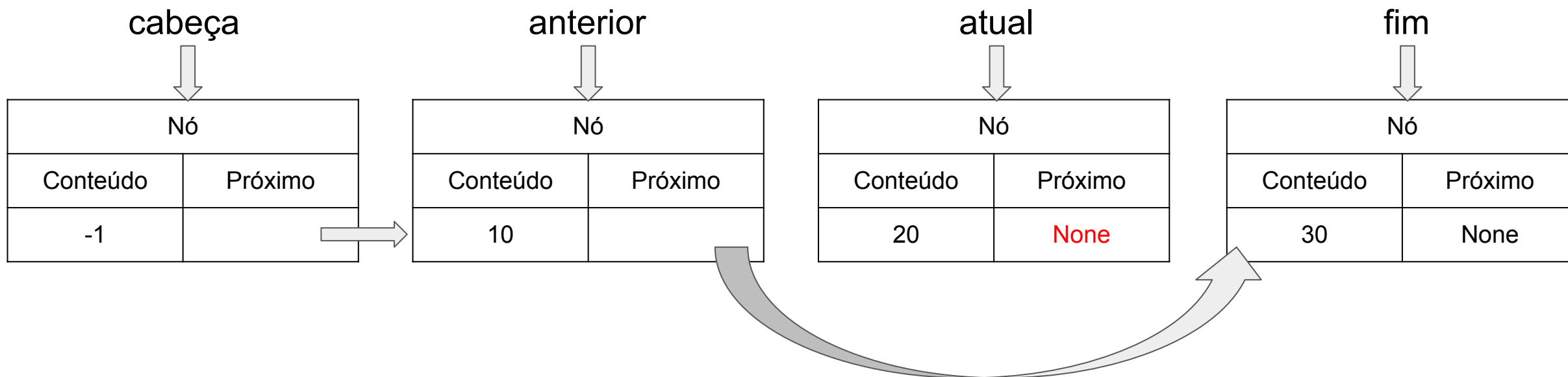
- Remover um elemento x (20)



anterior.proximo = atual.proximo

Lista simplesmente encadeada

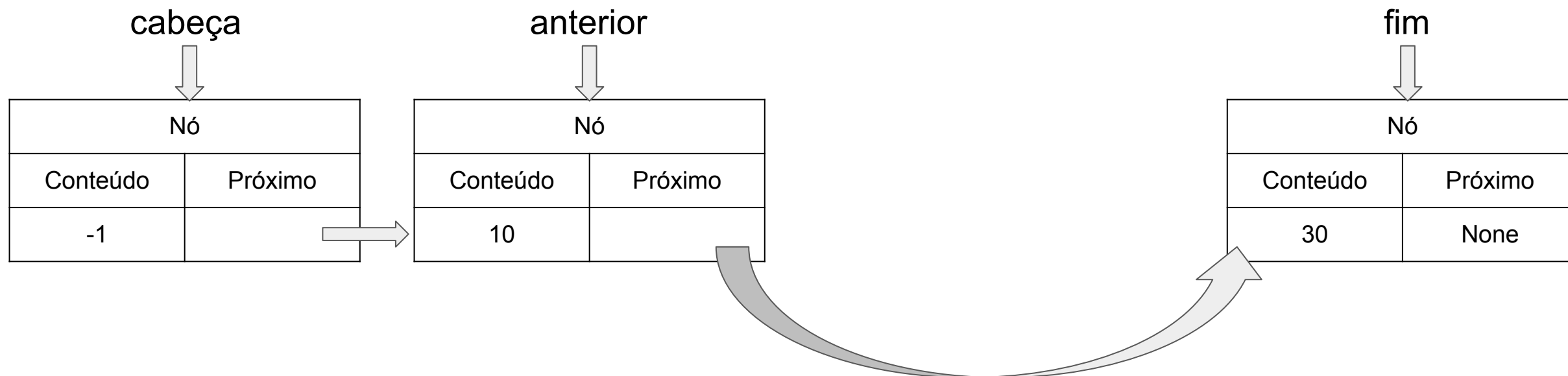
- Remover um elemento x (20)



atual.proximo = None

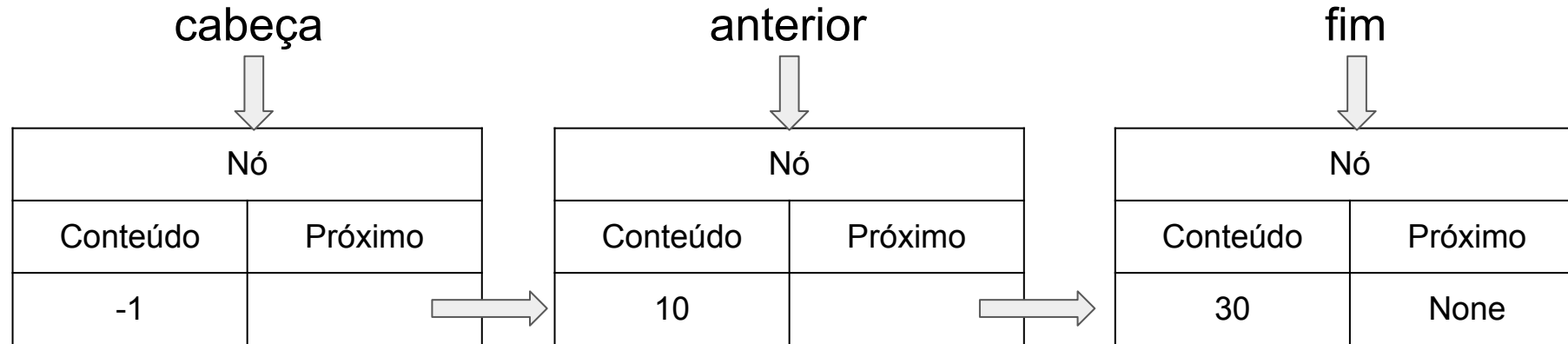
Lista simplesmente encadeada

- Remover um elemento x (20)



Lista simplesmente encadeada

- **Remover um elemento x (20)**



Lista simplesmente encadeada

- **Remover um elemento x (20)**

```
class Lista:
```

```
...
```

```
    anterior = self.cabeca
```

```
    atual = self.cabeca.proximo
```

```
    while(atual != None):
```

```
        if atual.conteudo == x:
```

```
            anterior.proximo = atual.proximo
```

```
            atual.proximo = None
```

```
            break
```

```
        anterior = atual
```

```
        atual = atual.proximo
```

```
    if self.fim.conteudo == x:
```

```
        self.fim = anterior
```

Resumo

- Lista simplesmente encadeada
 - Impressão
 - Inserção
 - Busca
 - Remoção
- Vantagens e desvantagens