

# Estrutura de Dados

Conjuntos – Array

**Prof<sup>a</sup>. Anna Giselle Ribeiro**

# Objetivos

- Apresentar:
  - TAD Conjunto
  - Implementação de TAD Conjunto com base em *arrays*

# CONJUNTO

# TAD Conjunto

- Definição
  - Tipo abstrato de dados que modela o conceito matemático de Conjunto
  - Tipo abstrato de dados que representa uma coleção de elementos que não admite elementos repetidos

# TAD Conjunto

- Operações:
  - `bool add(Element e)`
    - Adiciona um elemento *e* ao conjunto; retorna *true* caso o conjunto não contiver *e*, *false* caso contrário
  - `void clear()`
    - Remove todos os elementos do conjunto
  - `bool contains(Element e)`
    - Verifica se o conjunto já contém o elemento *e*
  - `bool isEmpty()`
    - Verifica se o conjunto é vazio
  - `bool remove(Element e)`
    - Remove o elemento *e* do conjunto; retorna *true* caso o conjunto contiver *e*, *false* caso contrário
  - `int size()`
    - Retorna o número de elementos contidos pelo conjunto

# Como usar um **conjunto?**

# Exemplo 1

- Suponha que uma trilha de Tweets recebeu diversas interações de diferentes usuários, e que cada usuário pode ter interagido mais de uma vez na mesma trilha. Neste contexto, quero fazer um programa que conte quantos diferentes usuários interagiram com esta trilha de Tweets.

# Exemplo 1 – Solução 1

```
int countUsers(TwitterThread twitterThread) {  
    List tweets = twitterThread.getTweets();  
    Set usersSet = new Set();  
    for( int i = 0; i < tweets.size(); ++i ) {  
        Tweet tweet = tweets.get(i);  
        User user = tweet.getUser();  
        usersSet.add(user);  
    }  
    return usersSet.size();  
}
```



# Exemplo 1 – Solução 1

```
int countUsers(TwitterThread twitterThread) {  
    List tweets = twitterThread.getTweets();  
    Set usersSet = new Set();  
    for( int i = 0; i < tweets.size(); i++)  
        Tweet tweet = tweets.get(i);  
        User user = tweet.getUser();  
        usersSet.add(user);  
    }  
    return usersSet.size();  
}
```

A operação 'add' do TAD Set abstrai pro programador a lógica de manter os usuários sem repetição

# Exemplo 1 – Solução 2

```
int countUsers(TwitterThread twitterThread) {  
    List tweets = twitterThread.getTweets();  
    Set usersSet = buildUsersSet(tweets);  
    return usersSet.size();  
}  
  
Set buildUsersSet(List tweets) {  
    Set usersSet = new Set();  
    for( int i = 0; i < tweets.size();  
        Tweet tweet = tweets.get(i);  
        User user = tweet.getUser();  
        usersSet.add(user);  
    }  
    return usersSet;  
}
```

Apenas quebrei a lógica em  
uma função auxiliar

## Exemplo 2

- Suponha agora que dadas duas trilha de Tweets, quero fazer um programa que identifique os usuários que interagiram pelo menos uma vez nas duas trilhas.

# Exemplo 2 – Solução 1

```
Set findCommonUsers( TwitterThread t1, TwitterThread t2) {  
    Set users1 = buildUsersSet( t1.getTweets() );  
    Set users2 = buildUsersSet( t2.getTweets() );  
    Set commonUsers = new Set();  
    for( User user : users1 ) {  
        if( users2.contains( user ) ) {  
            commonUsers.add( user );  
        }  
    }  
    return commonUsers;  
}
```

# Exemplo 2 – Solução 2

```
Set findCommonUsers( TwitterThread t1, TwitterThread t2) {  
    Set users1 = buildUsersSet(t1.getTweets());  
    Set users2 = buildUsersSet(t2.getTweets());  
    Set commonUsers = users1.intersection( users2 );  
  
    return commonUsers;  
}
```

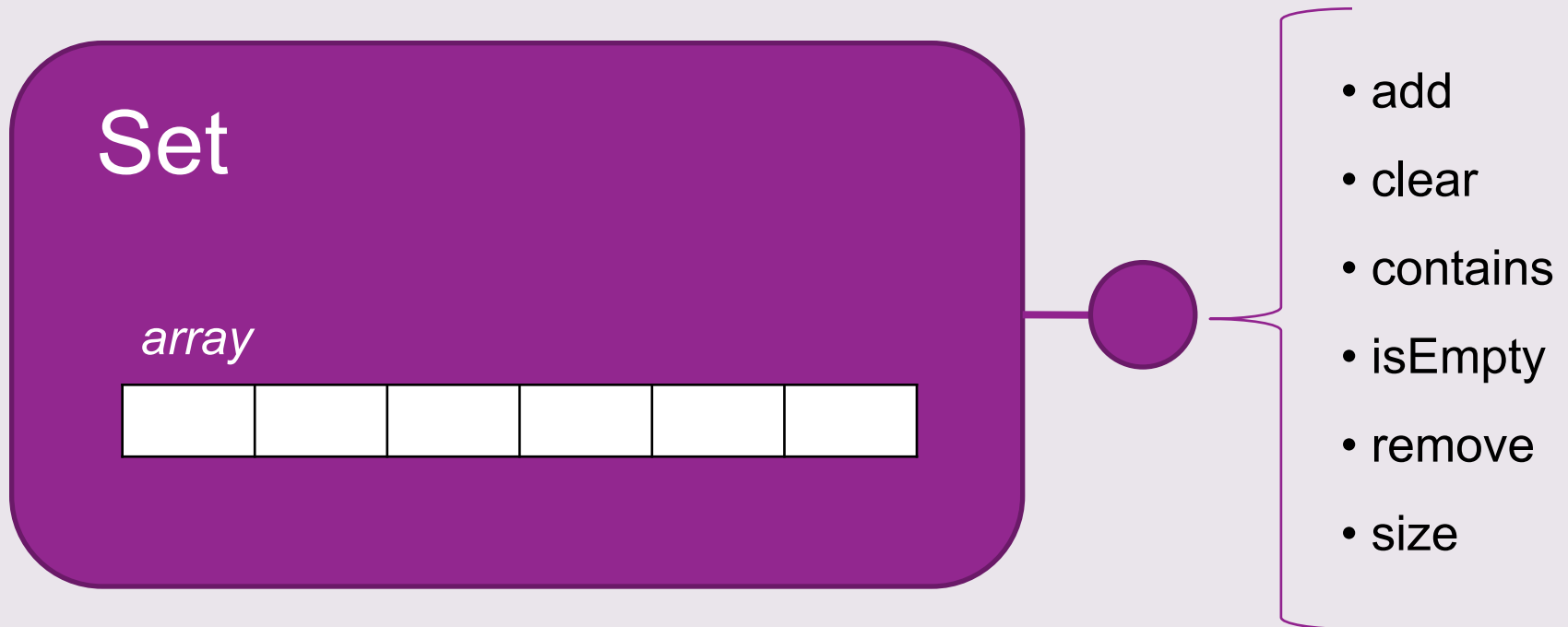
Operação 'intersection' nada mais é do que uma abstração para o que implementamos 'na mão' no slide passado

# Como implementar um **conjunto**?

Uma possibilidade: usando array.

# ArraySet

- Implementação interna do conjunto será feita com base num array



# ArraySet

```
class Set {  
private:  
    String *array;  
    ...  
public:  
    ArraySet();  
    ~ArraySet();  
    bool add(String s);  
    void clear();  
    bool contains( String s );  
    ...  
}
```



# ArraySet

- `int size()`
  - Necessário manter um atributo no objeto do tipo `ArraySet` para registrar quantos elementos estão contidos no conjunto
- `bool isEmpty()`
  - Apenas verifica se quantidade de elementos é igual a zero ou não

# ArraySet

```
class Set {  
private:  
    String *array;  
    int quantity;  
    ...  
public:  
    ArraySet();  
    ~ArraySet();  
    bool add(String s);  
    void clear();  
    bool contains( String s );  
    ...  
}
```

# ArraySet

- `bool contains( Element e )`:
  - Nada mais é do que uma busca sequencial no array interno
    - Complexidade:  $O(n)$

# ArraySet

```
bool Set::contains( std::string s ) {  
    for( int i = 0; i < this->quantity; ++i ) {  
        if ( array[i] == s ) {  
            return true;  
        }  
    }  
    return false;  
}
```

# ArraySet

- `bool add( Element e )`:
  - Percorrem-se todos os elementos do array; se encontrar um elemento igual, retorna *false*; senão, acrescenta o elemento na posição após o último elemento, incrementa a quantidade e retorna *true*
    - Necessário um atributo para registrar qual a primeira posição vazia
    - Complexidade:  $O(n)$

# ArraySet – Add

- Add
  - Add( 10 )



Quantidade = 7

# ArraySet – Add

- Add
  - Add( 10 )

Atributo fim representa  
a primeira posição  
vazia do array

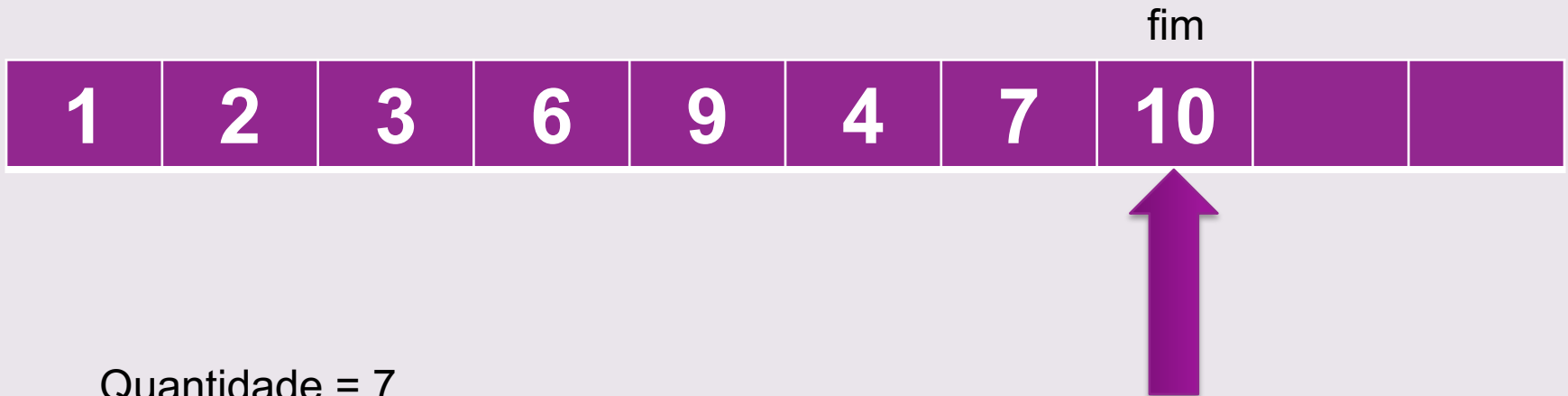
fim

1	2	3	6	9	4	7			
---	---	---	---	---	---	---	--	--	--

Quantidade = 7

# ArraySet – Add

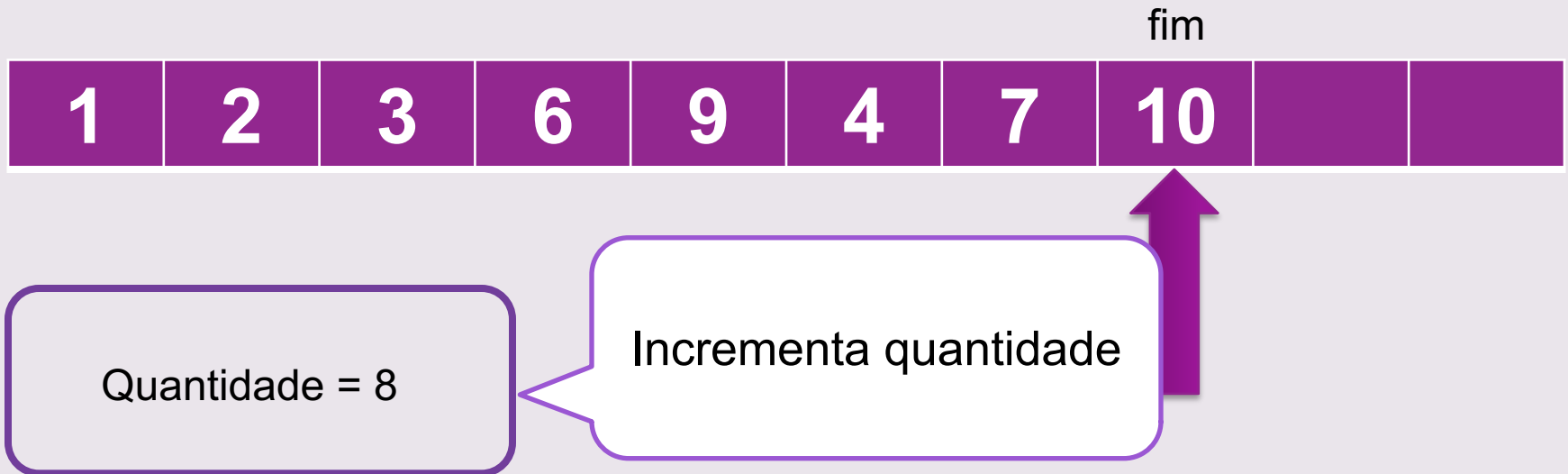
- Add
  - Add( 10 )





# ArraySet – Add

- Add
  - Add( 10 )



# ArraySet – Add

- Add

– Add( 10 )

Incrementa fim

fim



Quantidade = 8

# ArraySet – Add

- Add
  - Add( 4 )

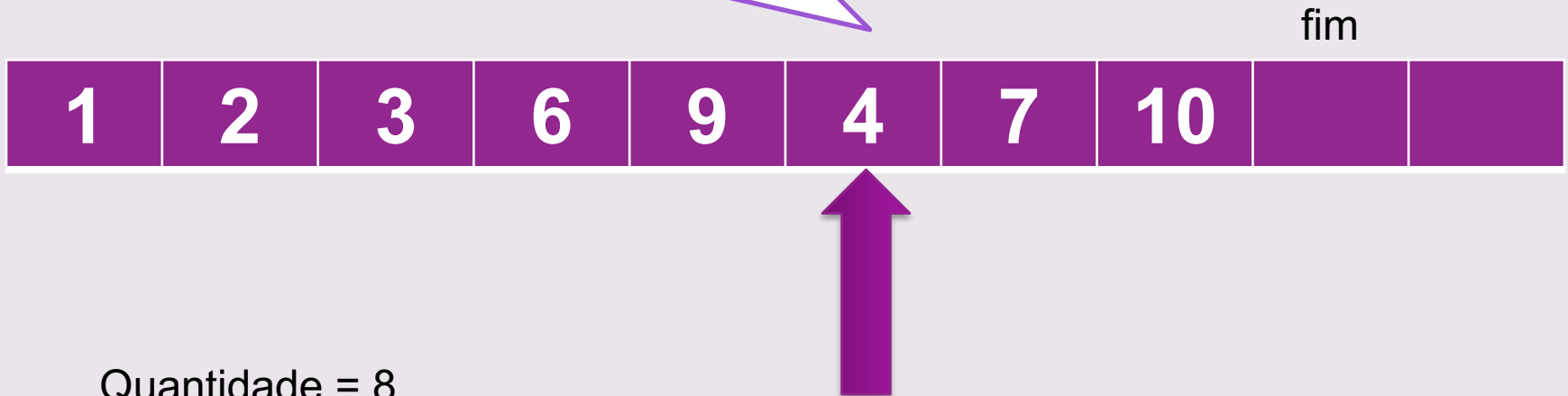


Quantidade = 8

# ArraySet – Add

- Add
  - Add( 4 )

Elemento 4 já está  
contido no conjunto;  
retorne *false*



# ArraySet – Add

- Add – E se o array estiver cheio?
  - Add( 11 )

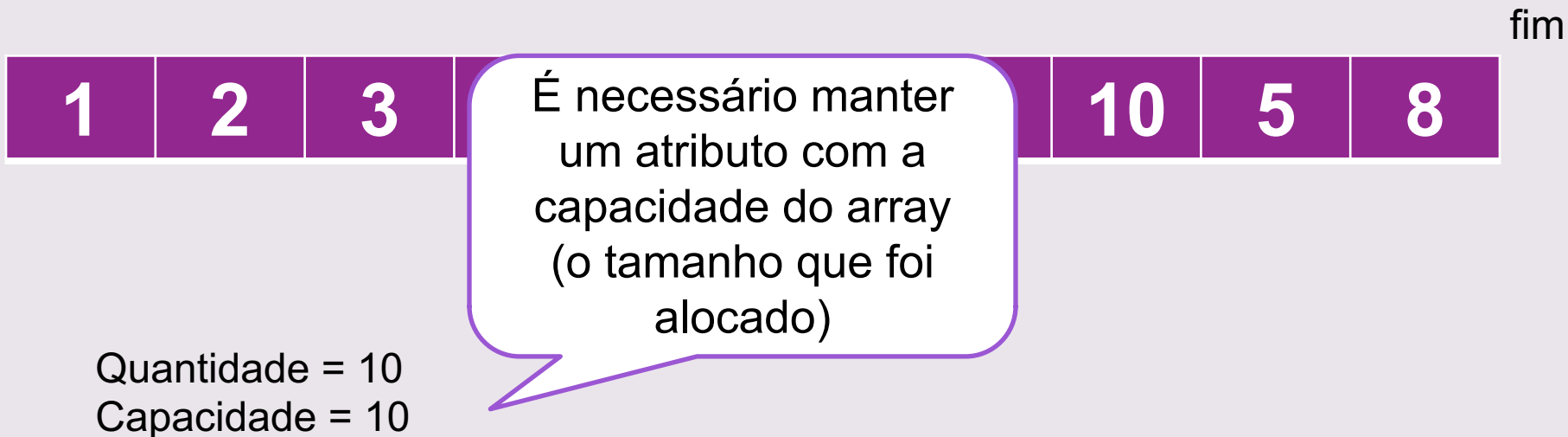
1	2	3	6	9	4	7	10	5	8
---	---	---	---	---	---	---	----	---	---

fim

Quantidade = 10

# ArraySet – Add

- Add – E se o array estiver cheio?
  - Add( 11 )



# ArraySet – Add

- Add – E se o array estiver cheio?
  - Ou não permite mais adições
  - Ou realoca um novo espaço pro array

1	2	3	6	9	4	7	10	5	8
---	---	---	---	---	---	---	----	---	---

fim

Quantidade = 10  
Capacidade = 10

# ArraySet – Remove

- Remove
  - remove ( 10 )



Quantidade = 7



# ArraySet – Remove

- Remove

– remove ( 10 )

Elemento 10 não está  
contido no conjunto, logo,  
não há o que remover

fim



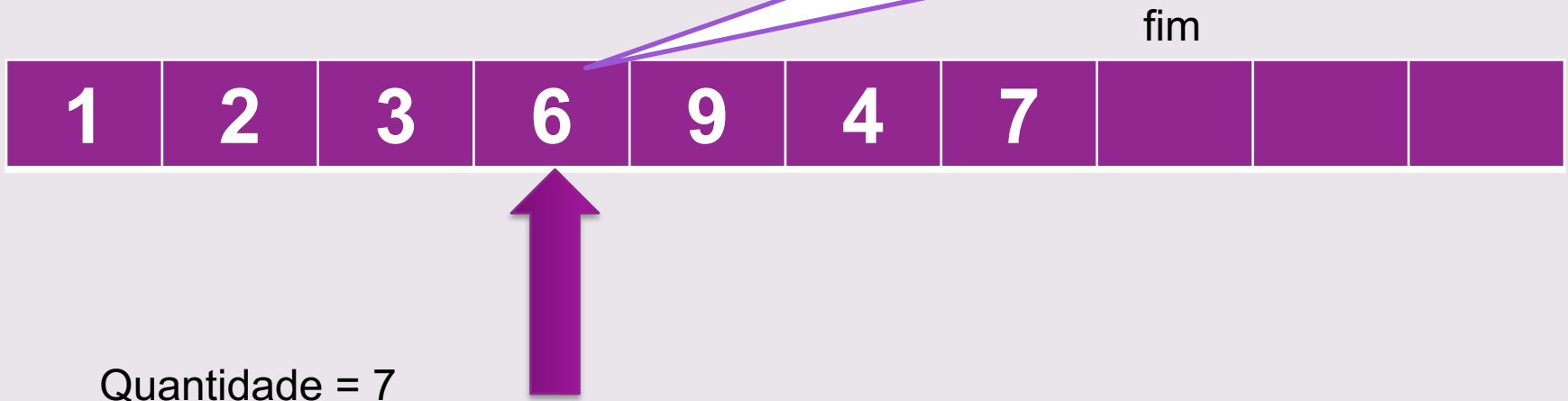
Quantidade = 7

# ArraySet – Remove

- Remove

- remove ( 6 )

Elemento 6 está contido no conjunto, logo, há o que remover. Mas como?



# ArraySet – Remove

- Remove

– remove ( 6 )

Copiando os elementos  
na posição i para a  
posição i-1

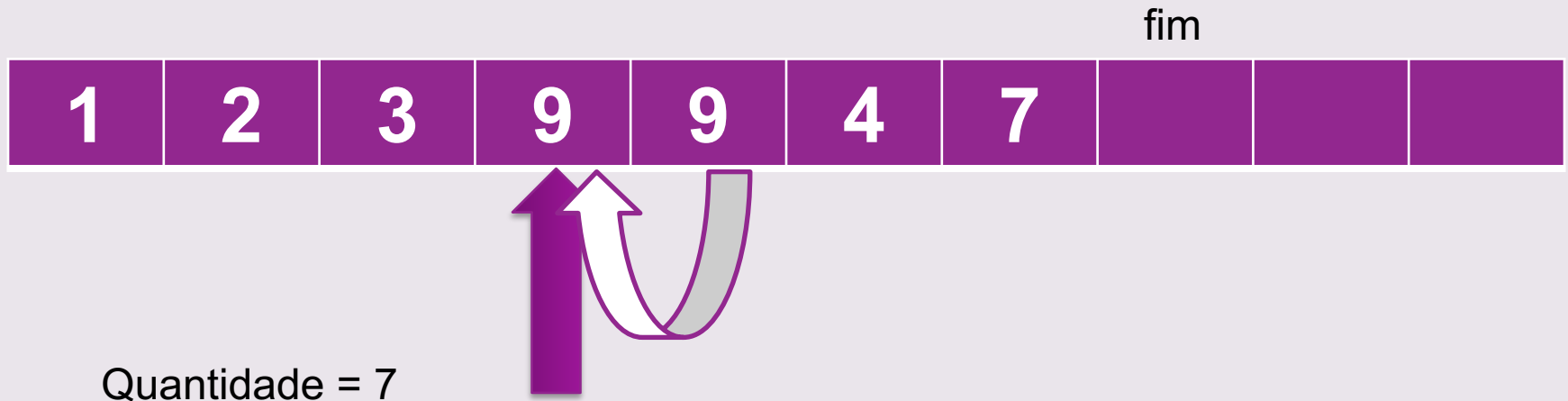
fim



Quantidade = 7

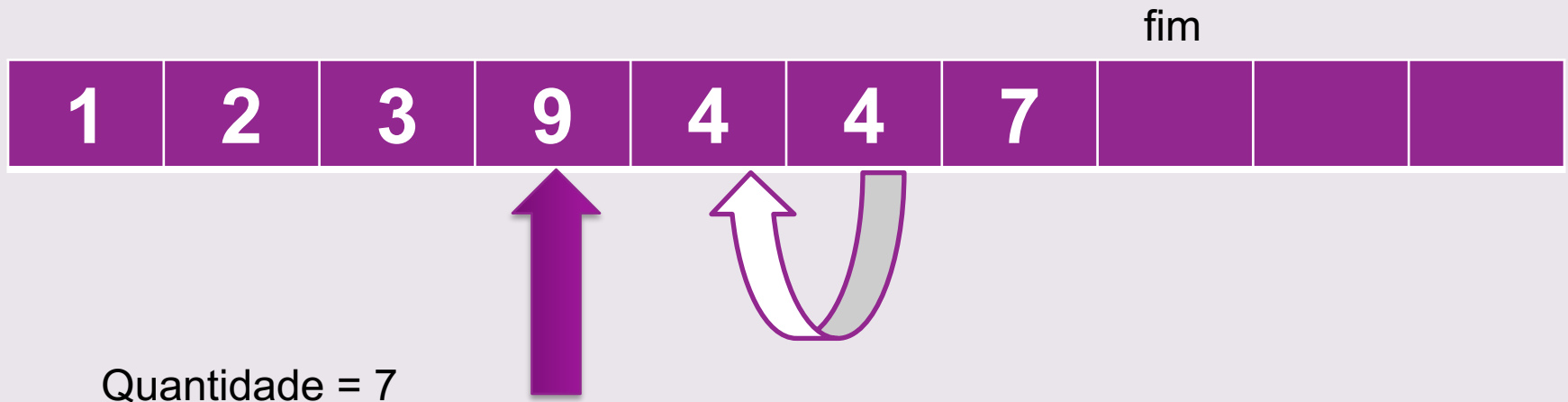
# ArraySet – Remove

- Remove
  - remove ( 6 )



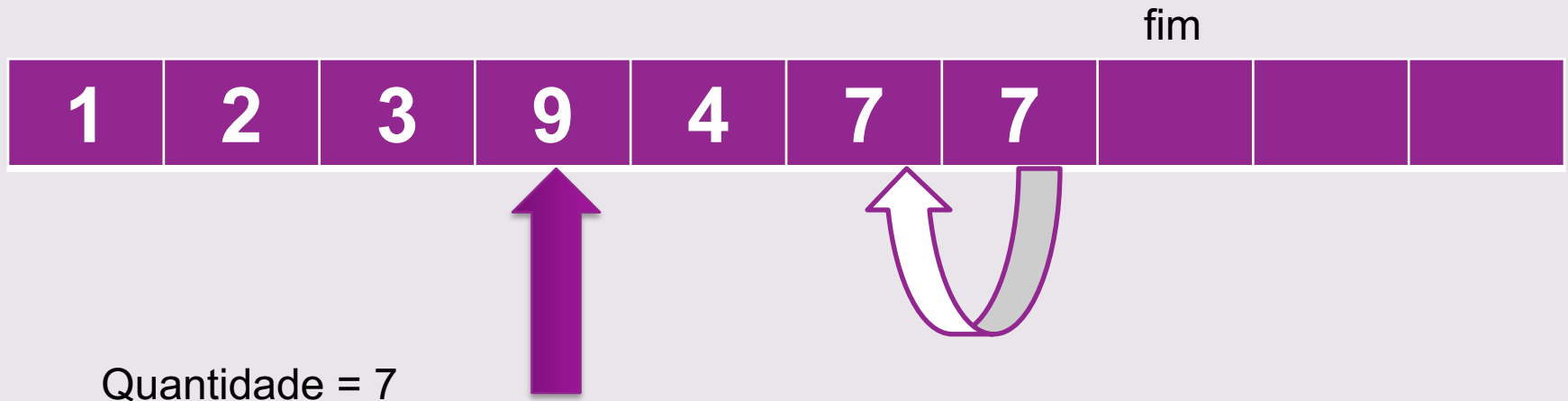
# ArraySet – Remove

- Remove
  - remove ( 6 )



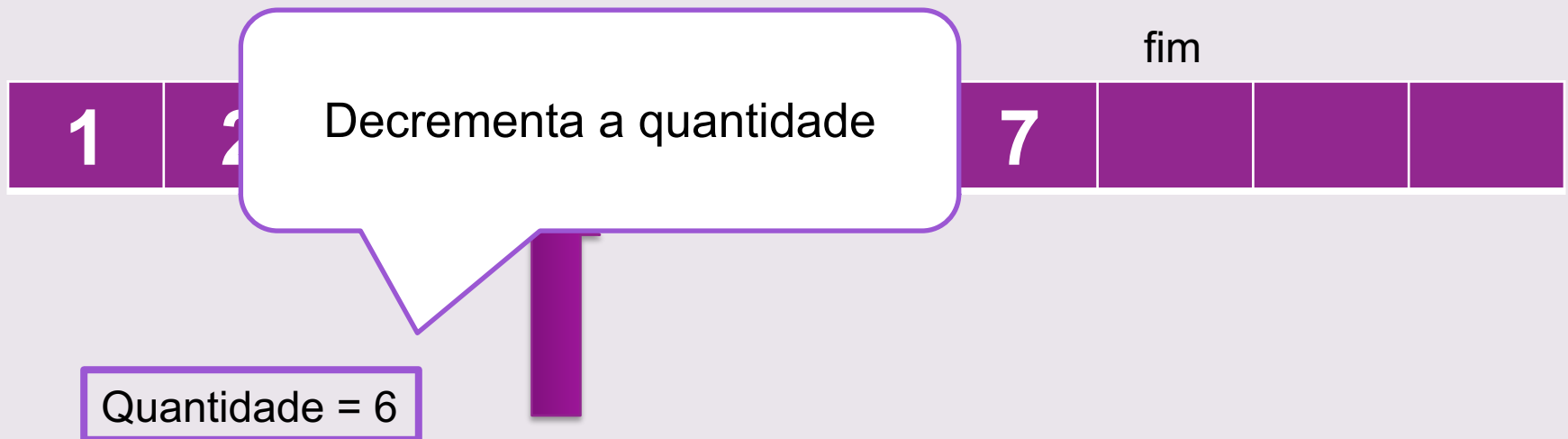
# ArraySet – Remove

- Remove
  - remove ( 6 )



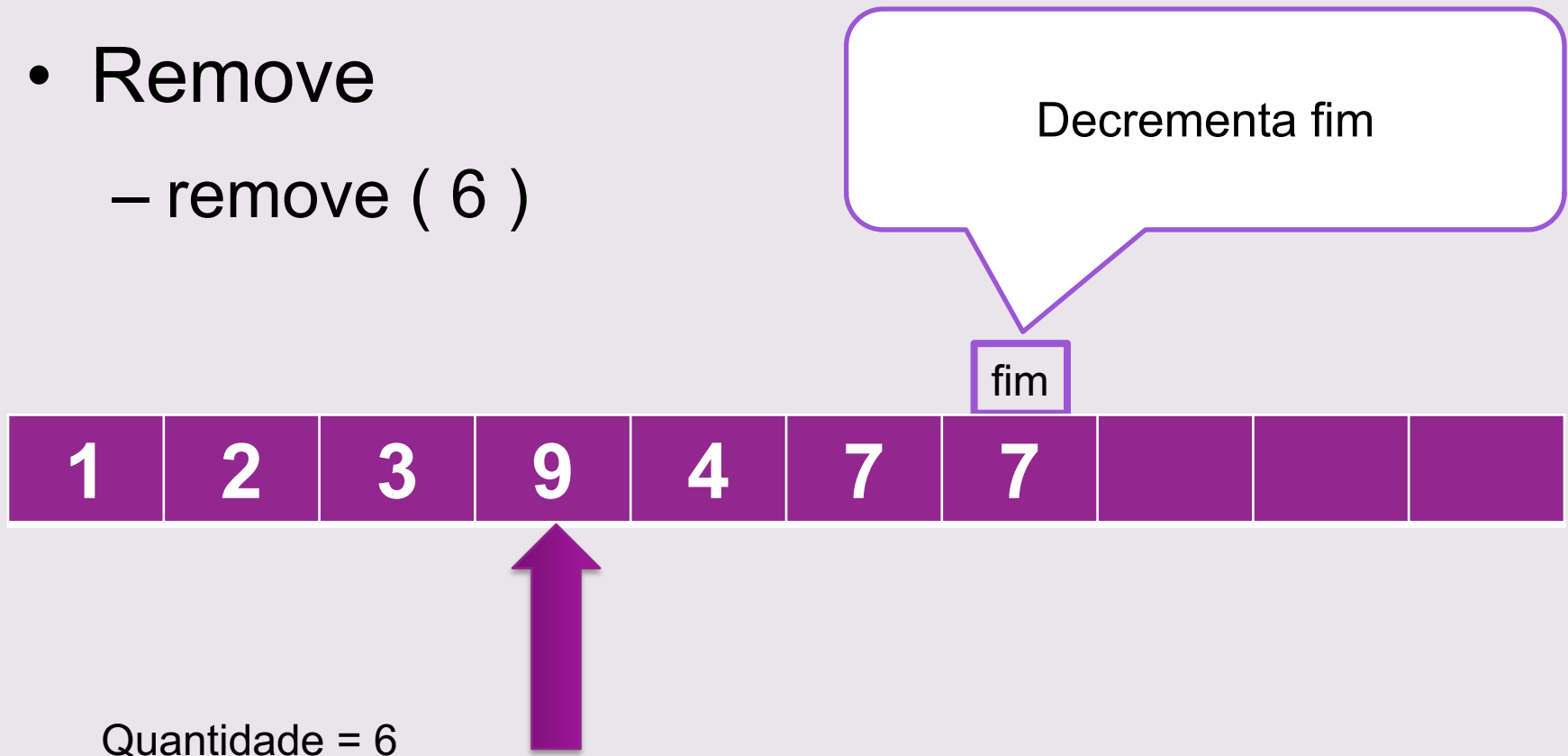
# ArraySet – Remove

- Remove
  - remove ( 6 )



# ArraySet – Remove

- Remove  
– remove ( 6 )

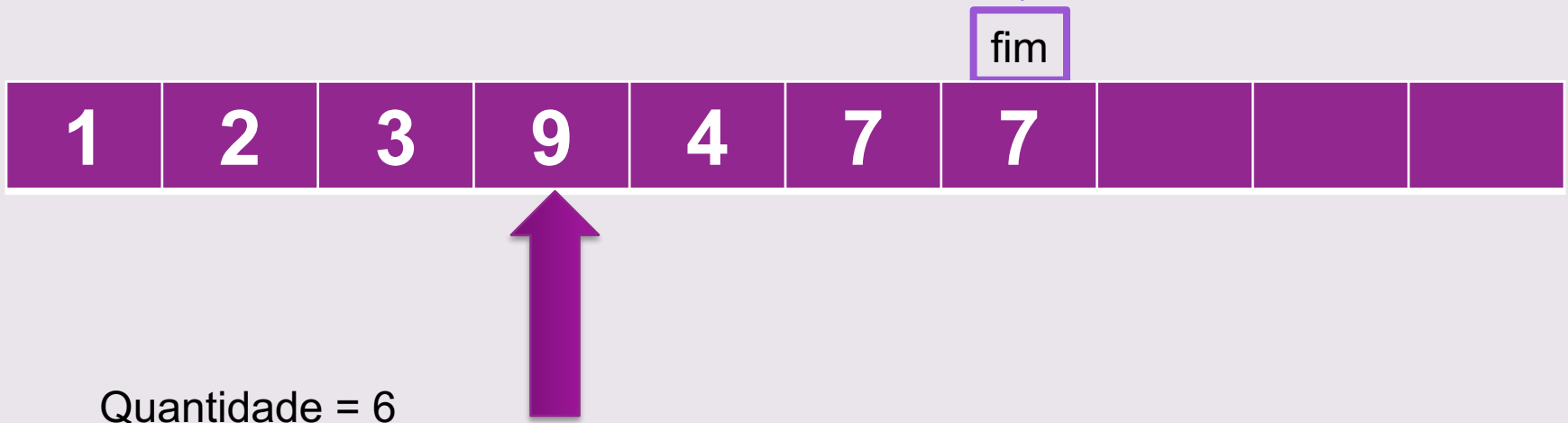




# ArraySet – Remove

- Remove
  - remove ( 6 )

Como fim indica a posição 'vazia', na próxima adição o elemento será sobrescrito



# Prática

# SEQUÊNCIA

# TAD Sequência

- Definição
  - Tipo abstrato de dados que representa uma coleção finita de elementos em sequência, admitindo elementos repetidos

# TAD Sequência

- Operações:
  - `bool add(Element e, int position)`
    - Adiciona um elemento 'e' à sequência na posição '*position*'
  - `void clear()`
    - Remove todos os elementos da sequência
  - `bool contains(Element e)`
    - Verifica se a sequência já contém o elemento *e*
  - `bool isEmpty()`
    - Verifica se a sequência está vazia
  - `bool remove(Element e)`
    - Remove o elemento *e* da sequência; retorna *true* caso a sequência contiver *e*, *false* caso contrário
  - `int size()`
    - Retorna o número de elementos contidos pelo conjunto
  - `Element get(int i)`
    - Retorna o elemento que está na *i*-ésima posição da Sequência

# Como implementar uma **sequência?**

Uma possibilidade: usando array.

# ArraySequence

- Já vimos como implementar um conjunto com base em arrays. Implementação da Sequência é similar.
- Algumas diferenças:
  - Add:
    - No Conjunto, é necessário checar se o elemento já existe; na Sequência não
    - Na Sequência, é definido uma posição de onde deve ser inserido o elemento; no Conjunto não
  - Get:
    - Na Sequência, é possível acessar um elemento que está numa determinada posição, no Conjunto não

# ArraySequence – Get

- No caso de acesso a um determinado índice:  $O(1)$
- Atenção: `get( 1 )` quer dizer que queremos o primeiro elemento da sequência, e não o elemento que está no índice 1 do array interno; lembre-se que o usuário do TAD Sequência não conhece os detalhes de implementação, logo, não sabe que existe um *array* internamente



# ArraySequence – Add

- Add( 10, 8 )
  - “Adicione o elemento 10 de modo que ele seja o oitavo elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 8 )
  - “Adicione o elemento 10 de modo que ele seja o oitavo elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 8 )
  - “Adicione o elemento 10 de modo que ele seja o oitavo elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 8 )
  - “Adicione o elemento 10 de modo que ele seja o oitavo elemento da sequência”



Quantidade = 8

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

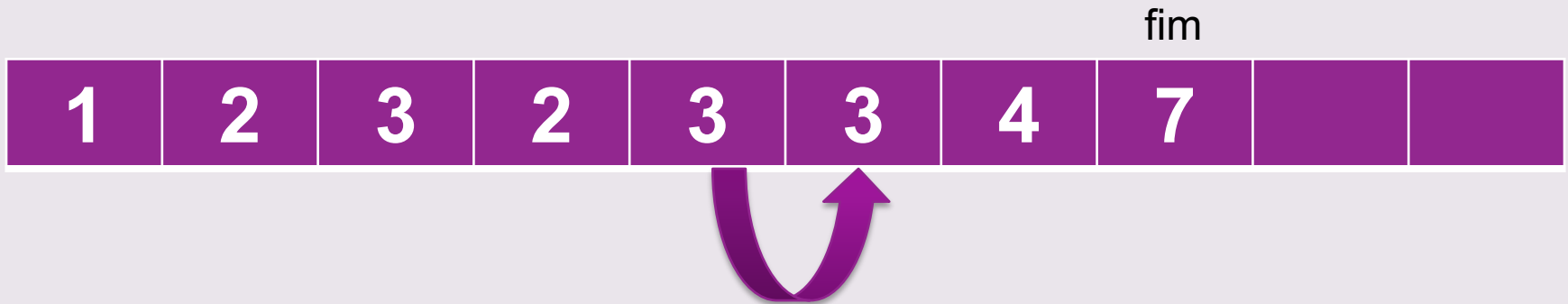
- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”

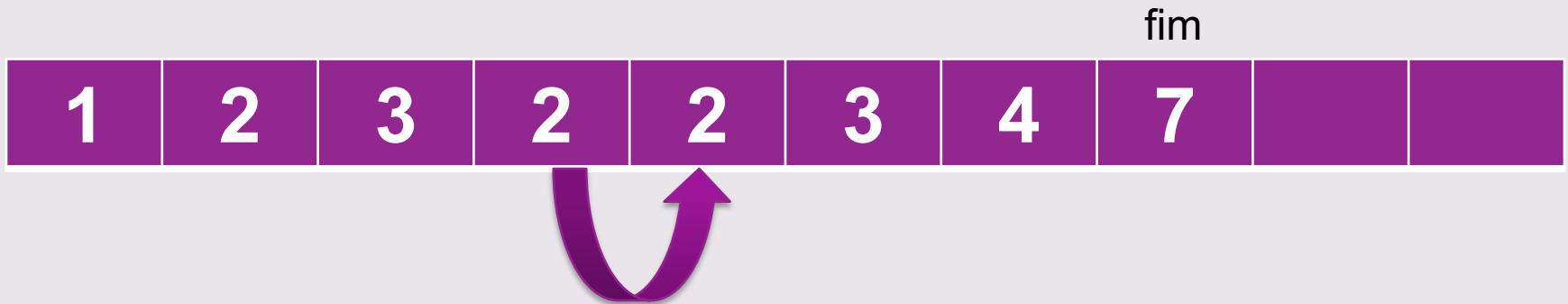


Quantidade = 7



# ArraySequence – Add

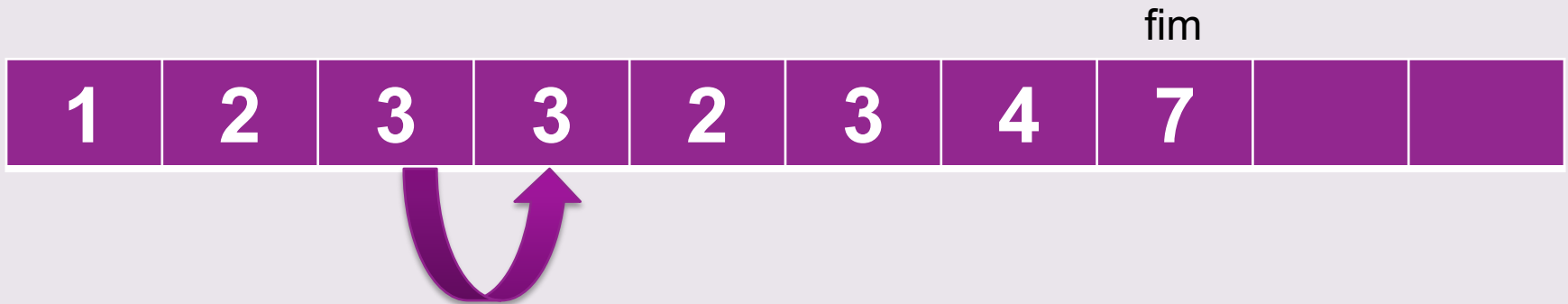
- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

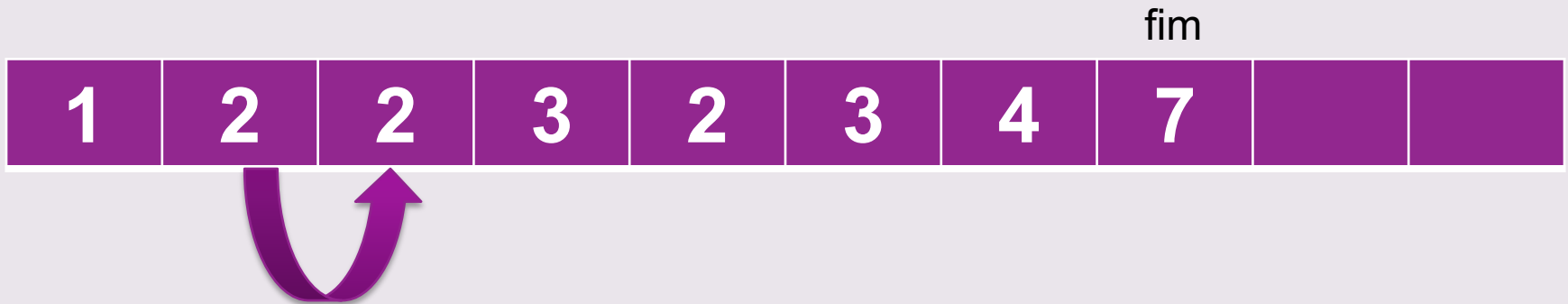
- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

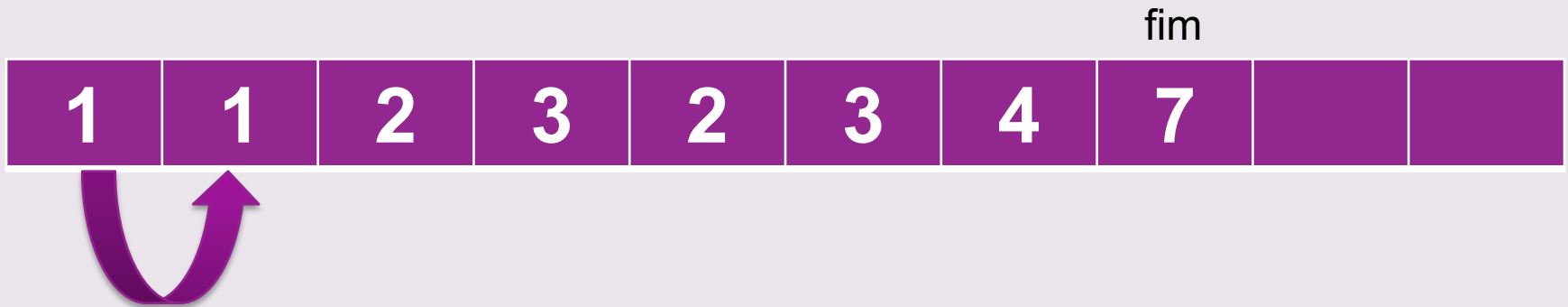
- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 7

# ArraySequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da sequência”



Quantidade = 8

# Qual seria uma outra forma de implementar uma Sequência?

Outra possibilidade: usando “nós” encadeados.  
Neste caso, chamamos de lista encadeada.

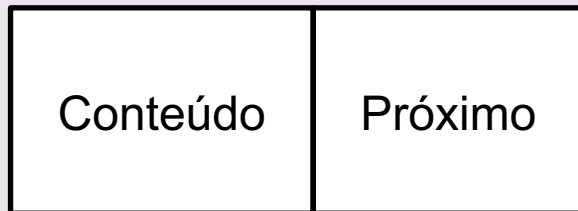
```
Estrutura No{  
    conteúdo;  
    No próximo;  
}
```

```
Estrutura Lista{  
    No início;  
    quantidade;  
}
```



# Lista encadeada – Estrutura Nó

Nó



# Lista encadeada – Estrutura Lista

## Lista

Início

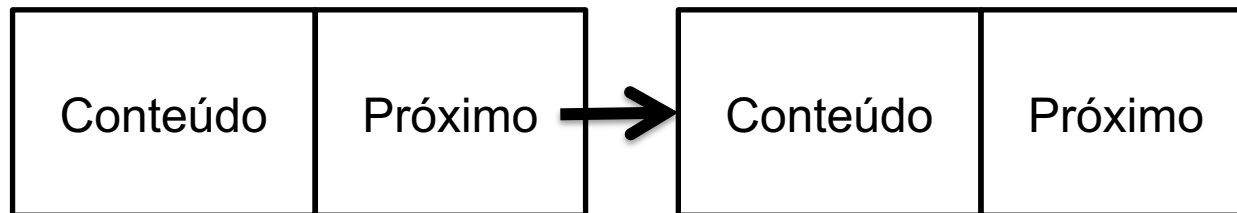
Conteúdo	Próximo
----------	---------

Quantidade = 1

# Lista encadeada – Estrutura Lista

## Lista

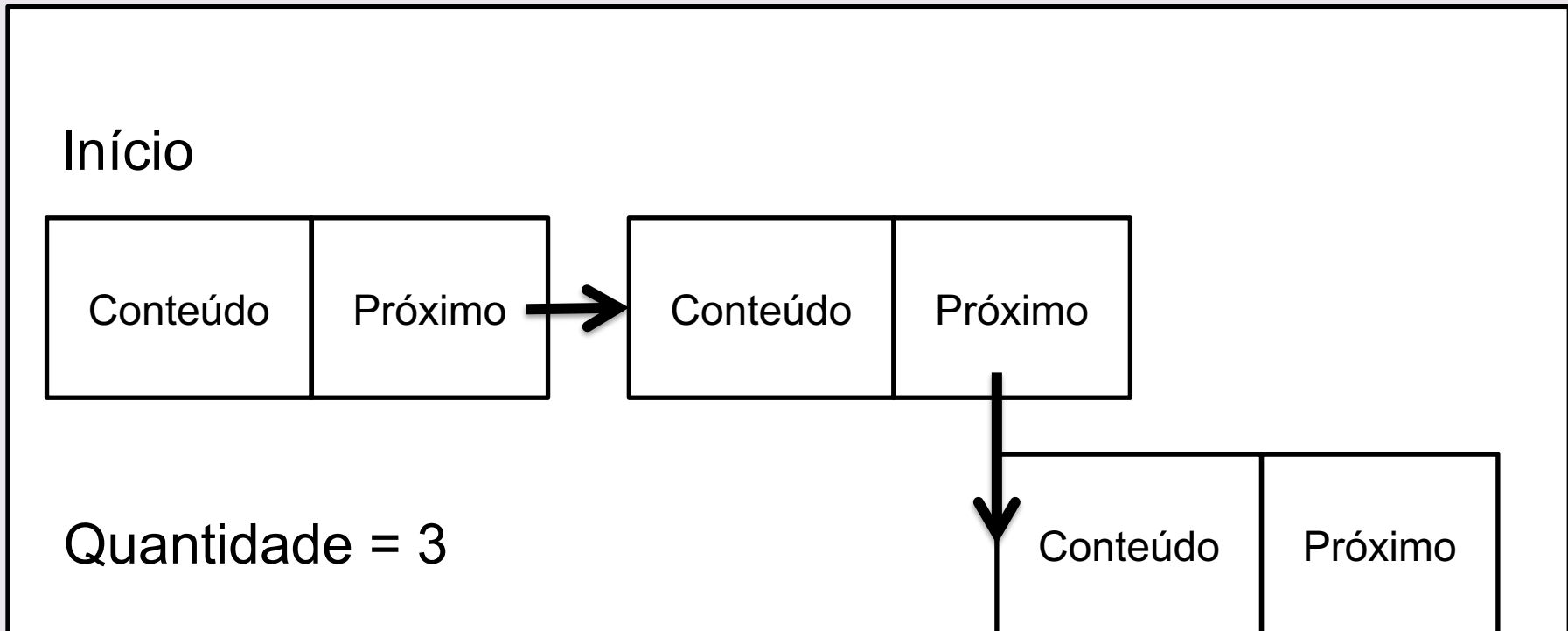
Início



Quantidade = 2

# Lista encadeada – Estrutura Lista

## Lista



# Lista sequencial X Lista encadeada

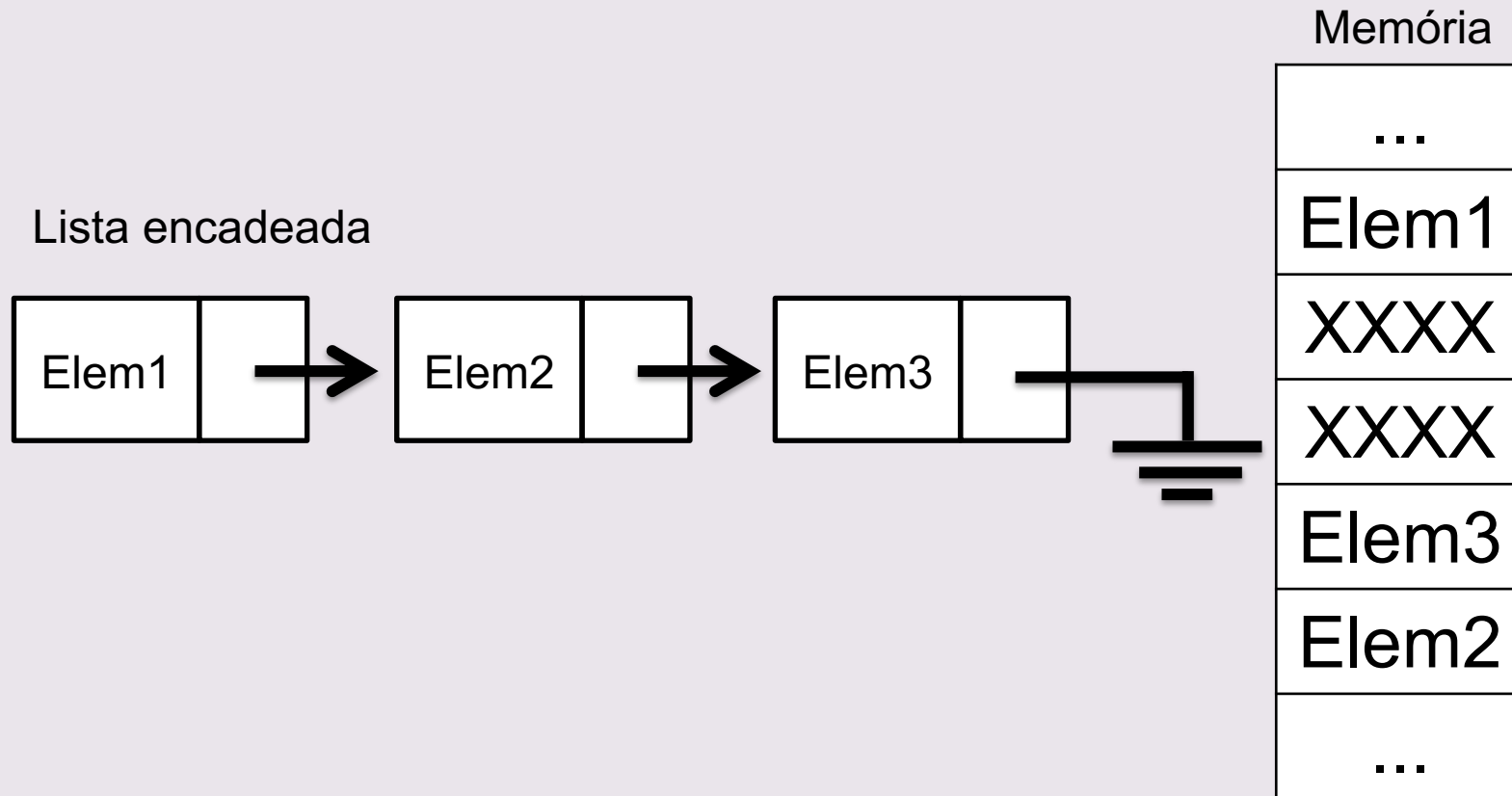
Lista sequencial

Elem1	Elem2	Elem3	Elem4
-------	-------	-------	-------

Memória

...
Elem1
Elem2
Elem3
Elem4
XXXX
...

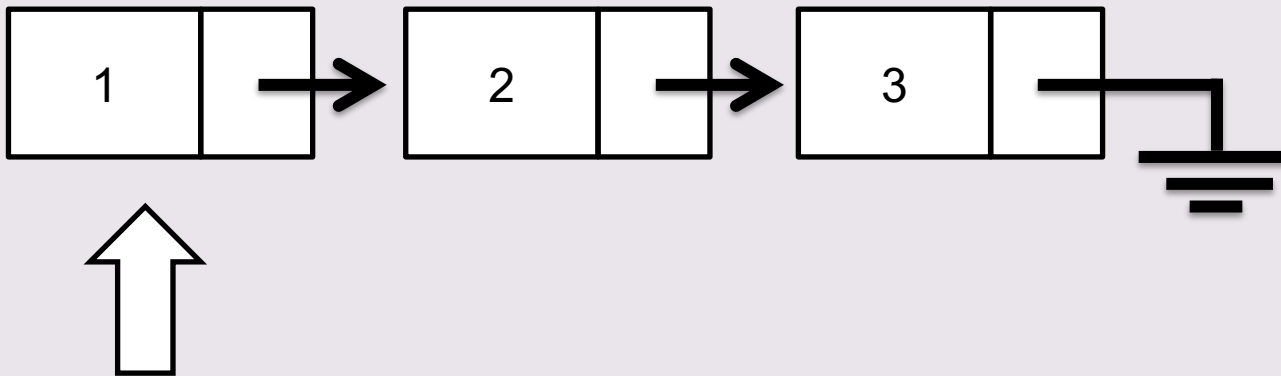
# Lista sequencial X Lista encadeada



# ListSequence – Add

- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da sequência”

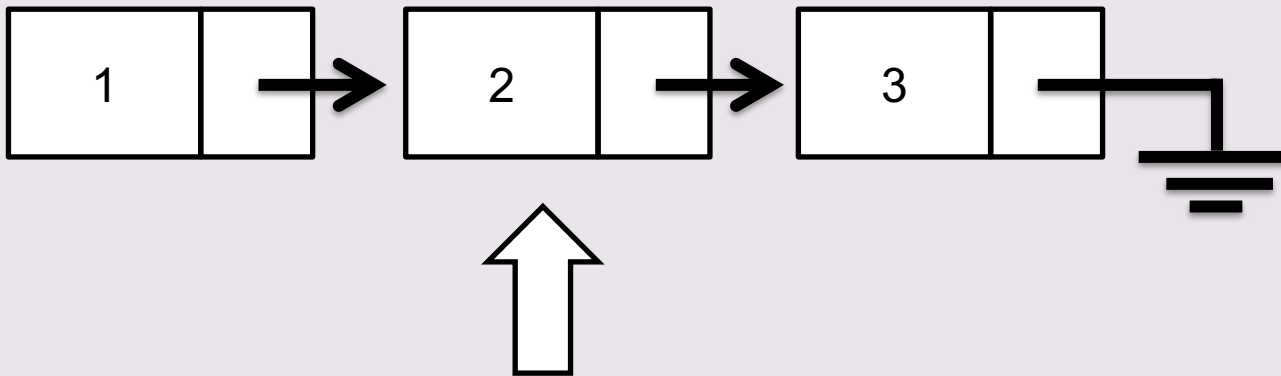
Início



# ListSequence – Add

- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da sequência”

Início

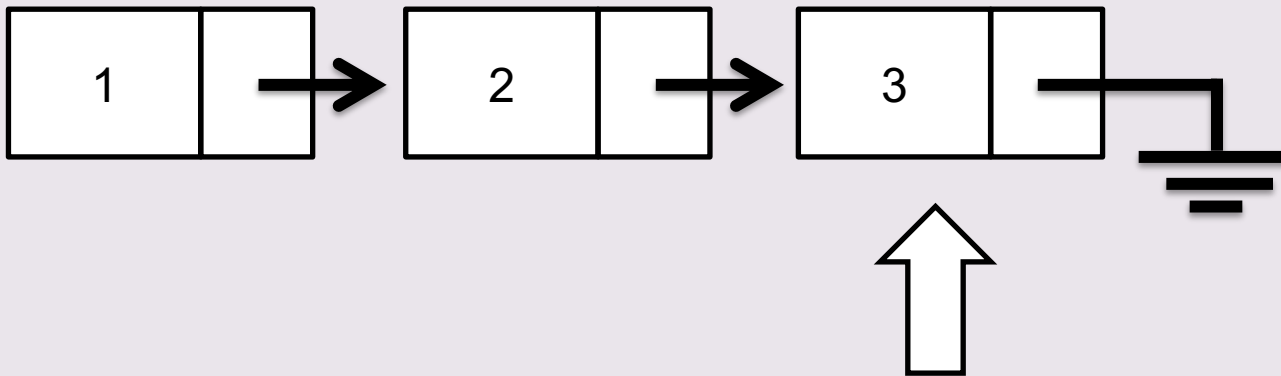




# ListSequence – Add

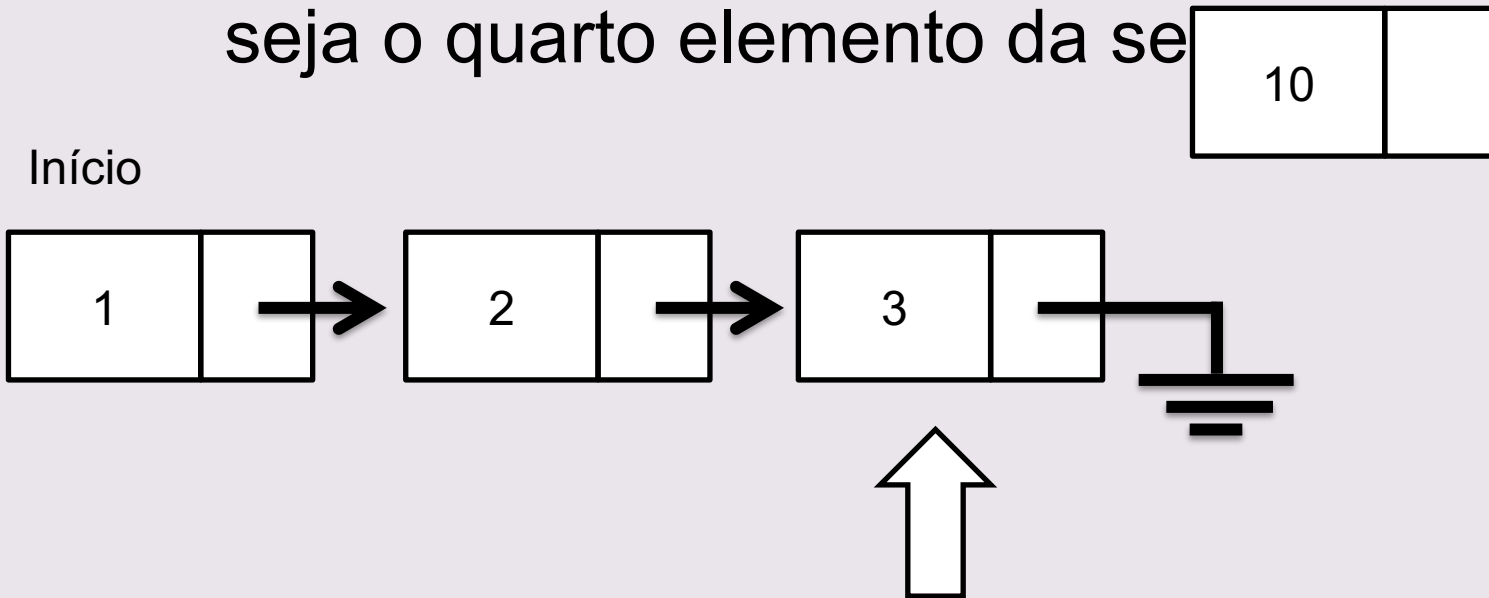
- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da sequência”

Início



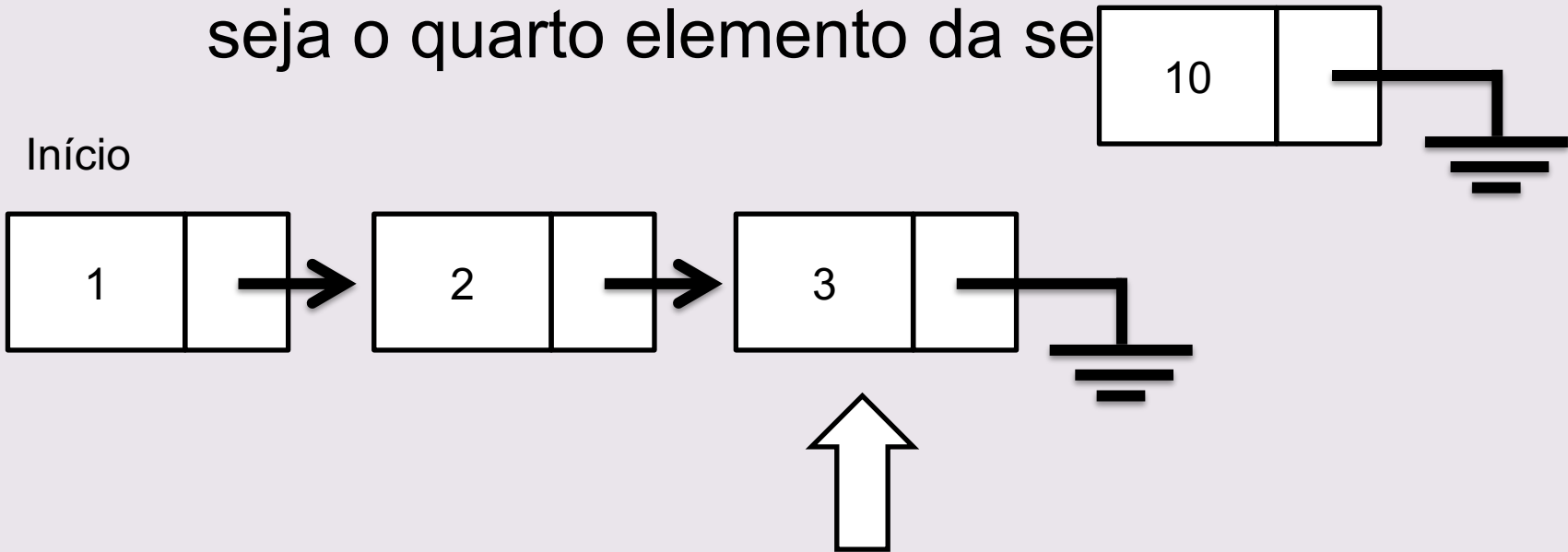
# ListSequence – Add

- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da se



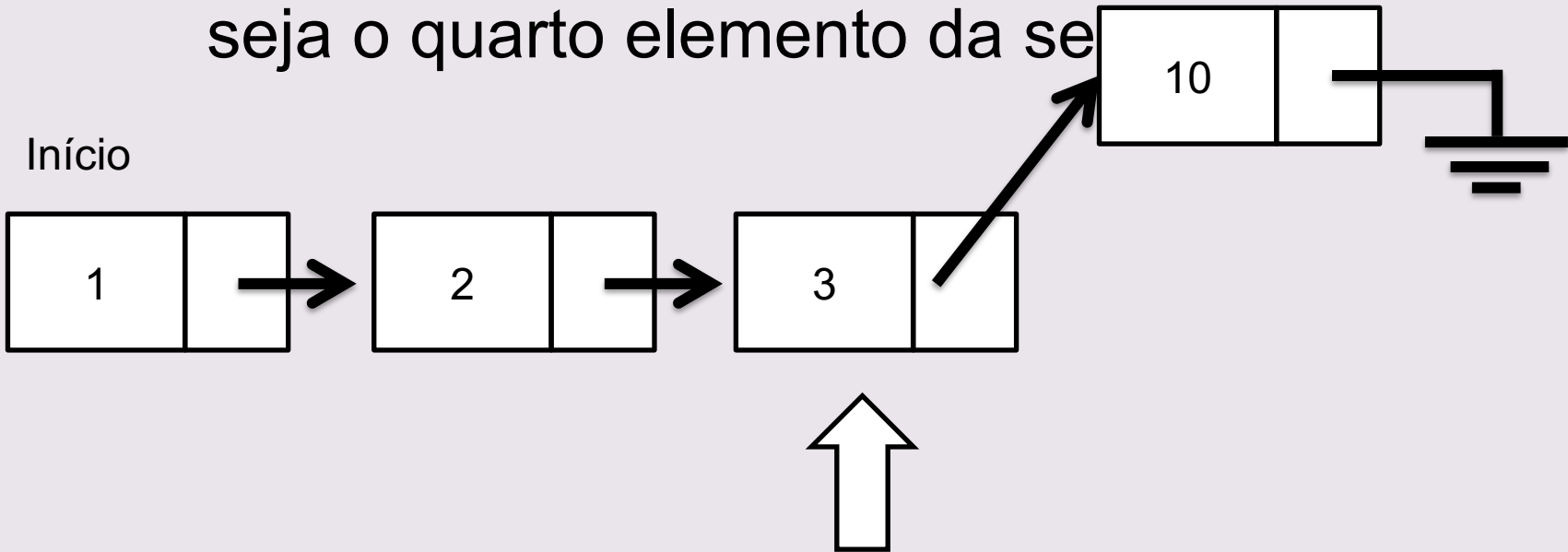
# ListSequence – Add

- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da se



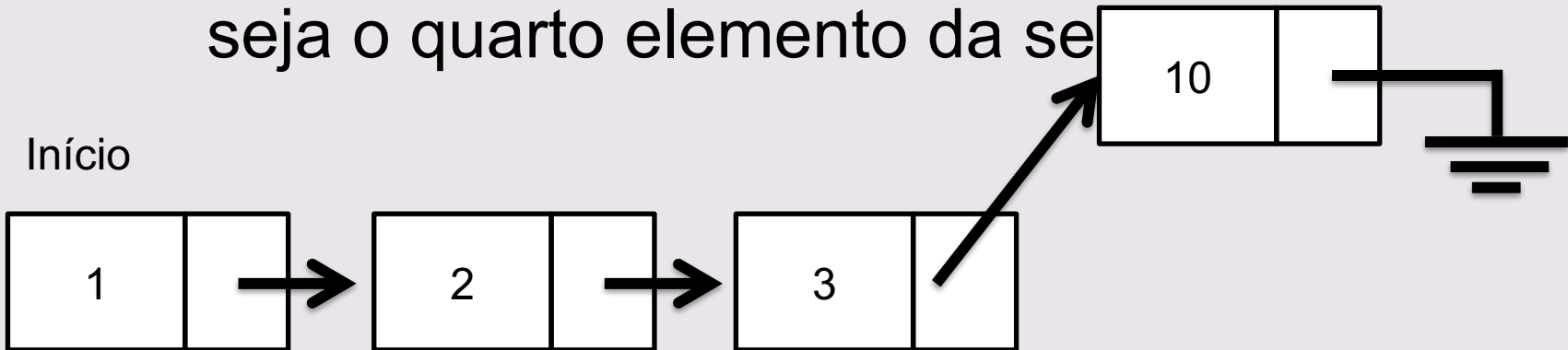
# ListSequence – Add

- Add( 10, 4 )
  - “Adicione o elemento 10 de modo que ele seja o quarto elemento da sequência”



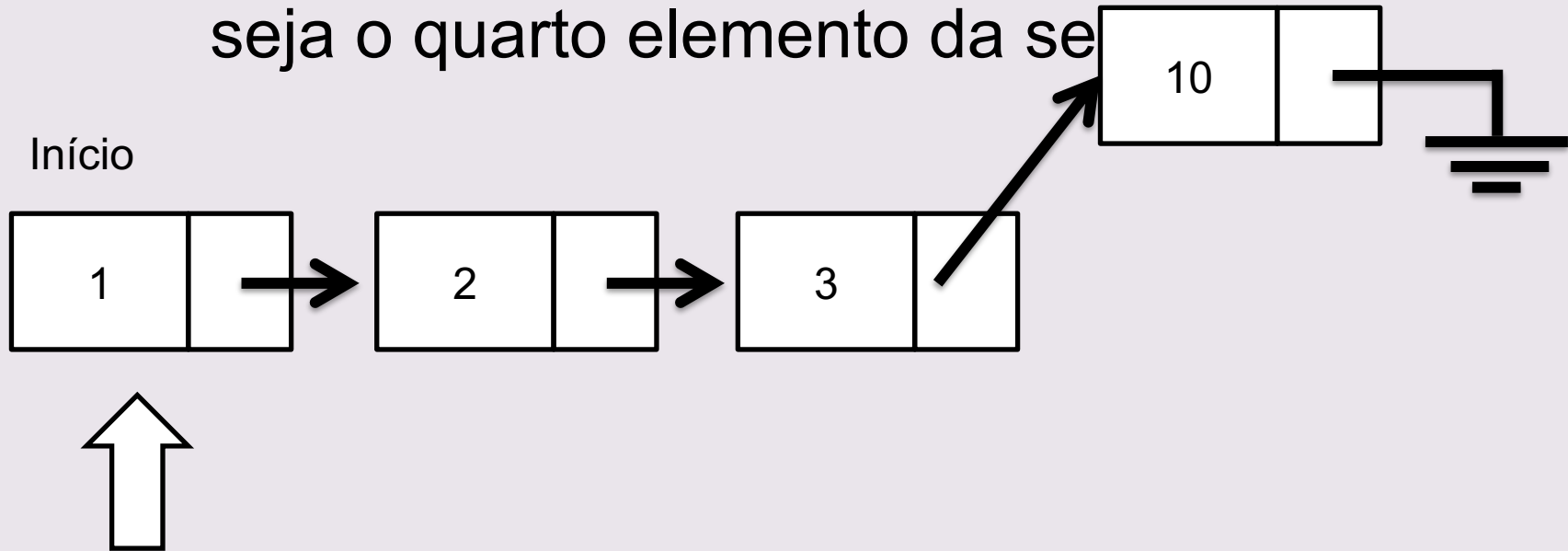
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



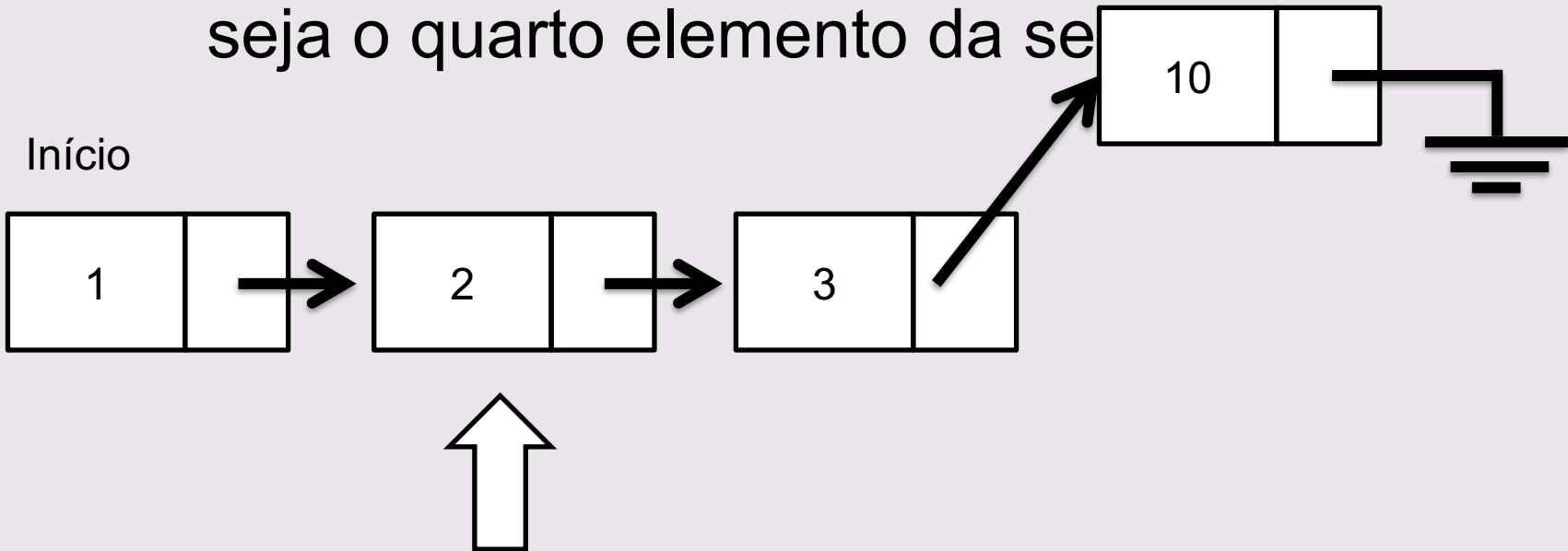
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



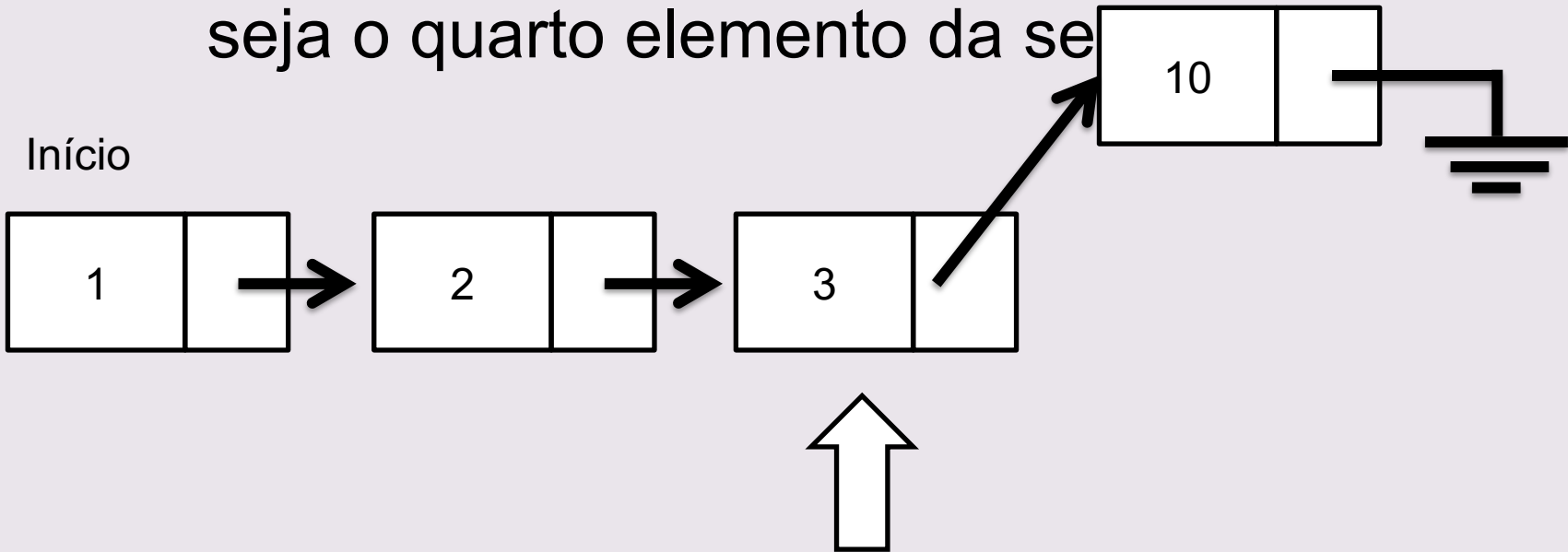
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



# ListSequence – Add

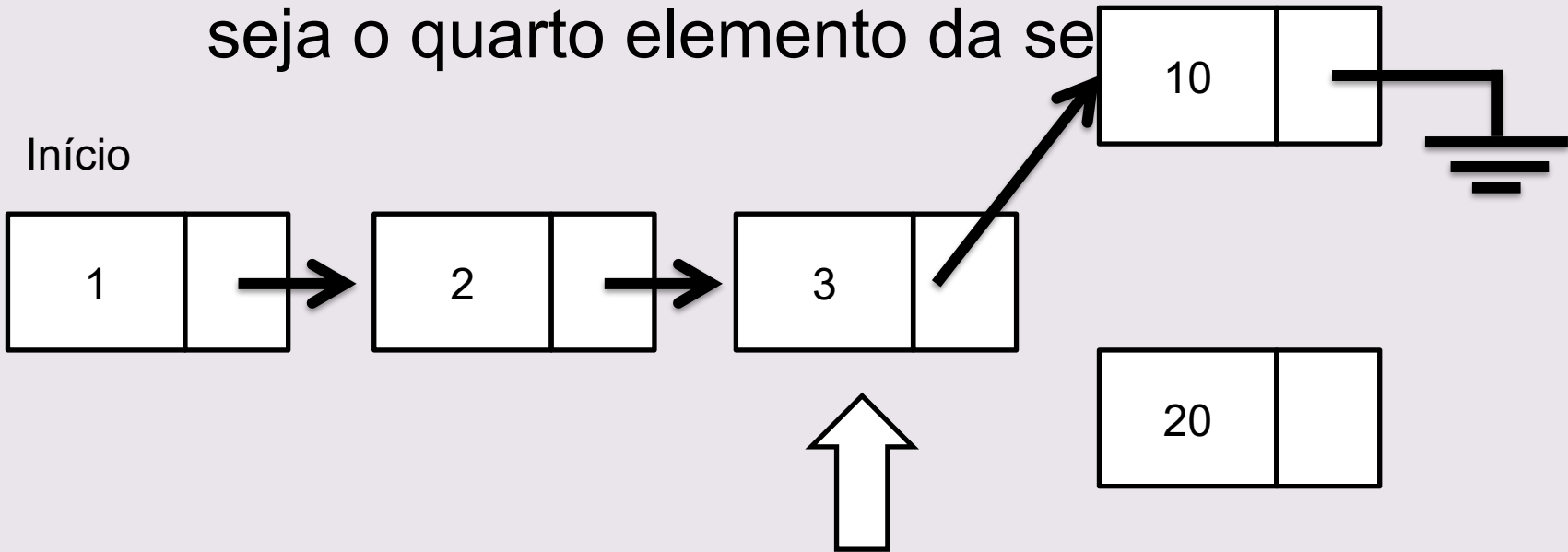
- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”





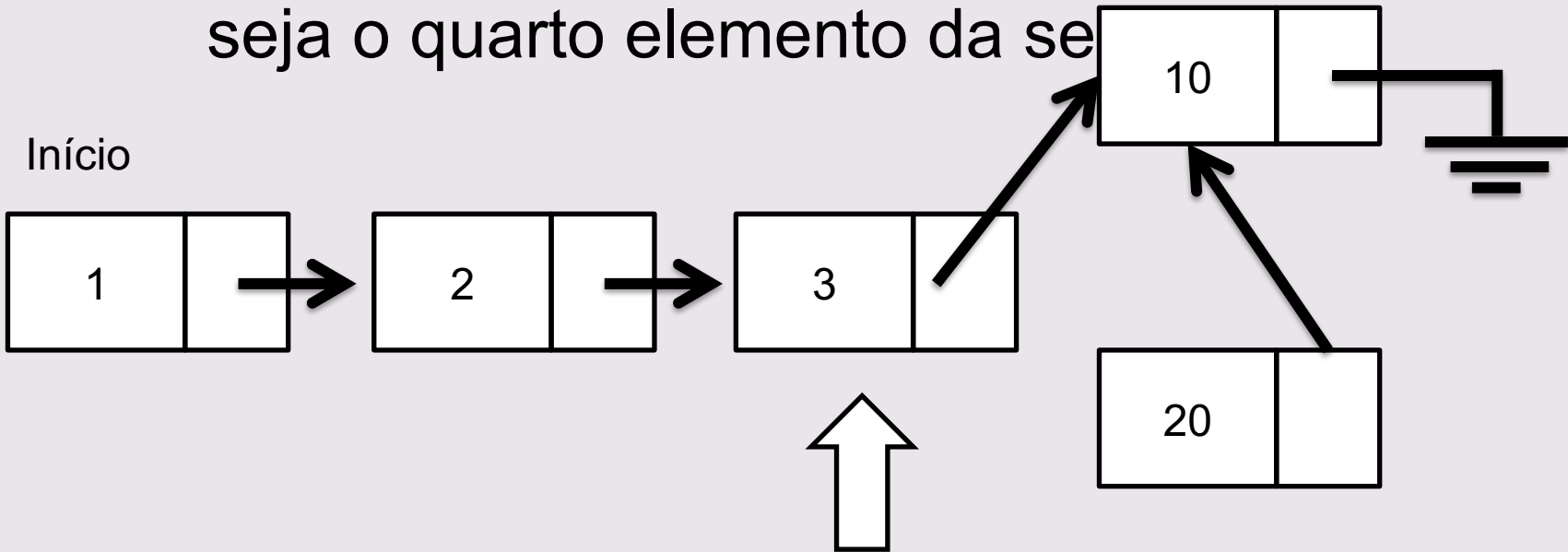
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



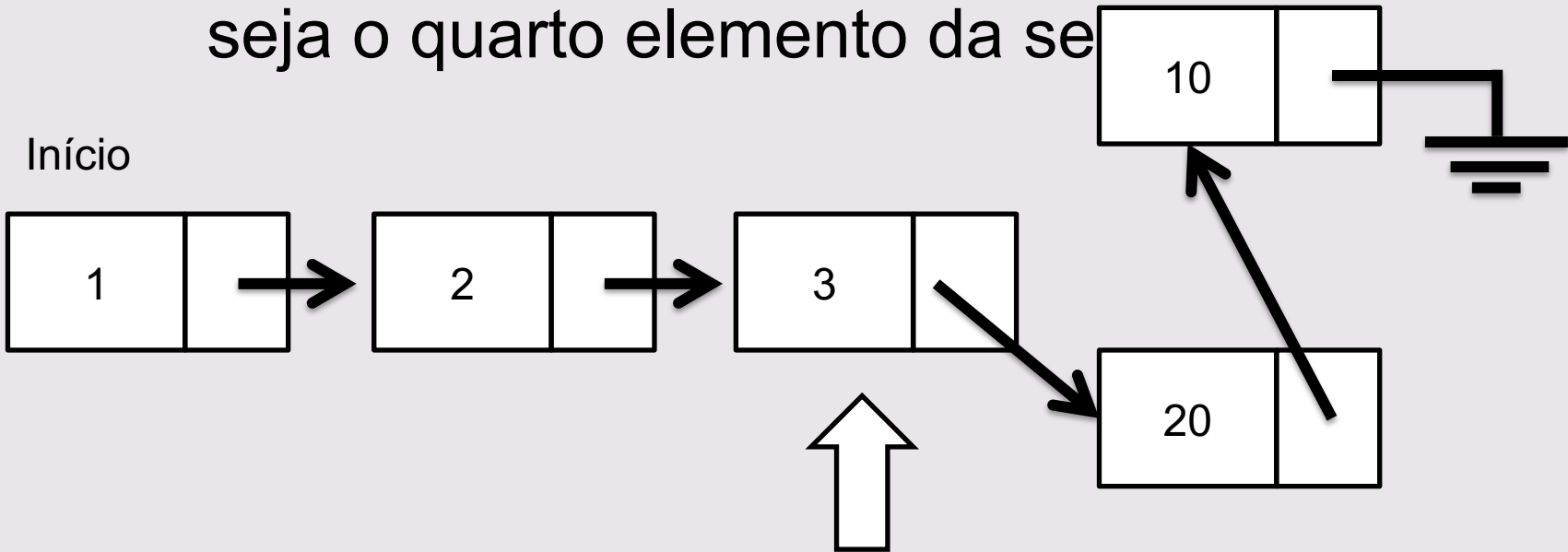
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



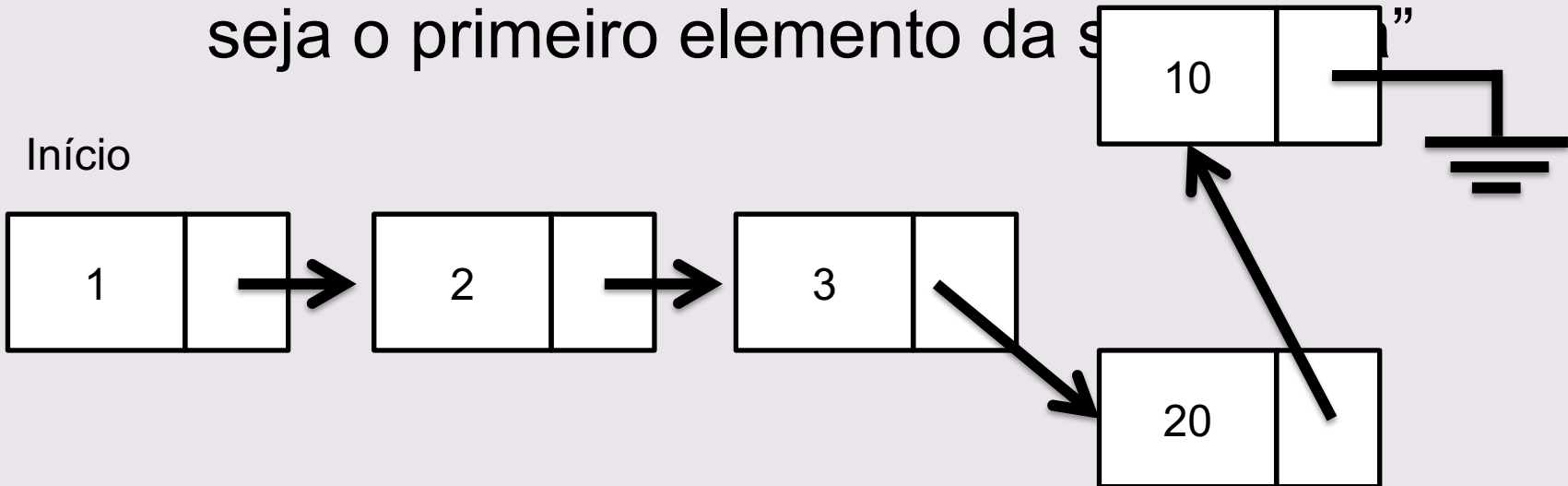
# ListSequence – Add

- Add( 20, 4 )
  - “Adicione o elemento 20 de modo que ele seja o quarto elemento da sequência”



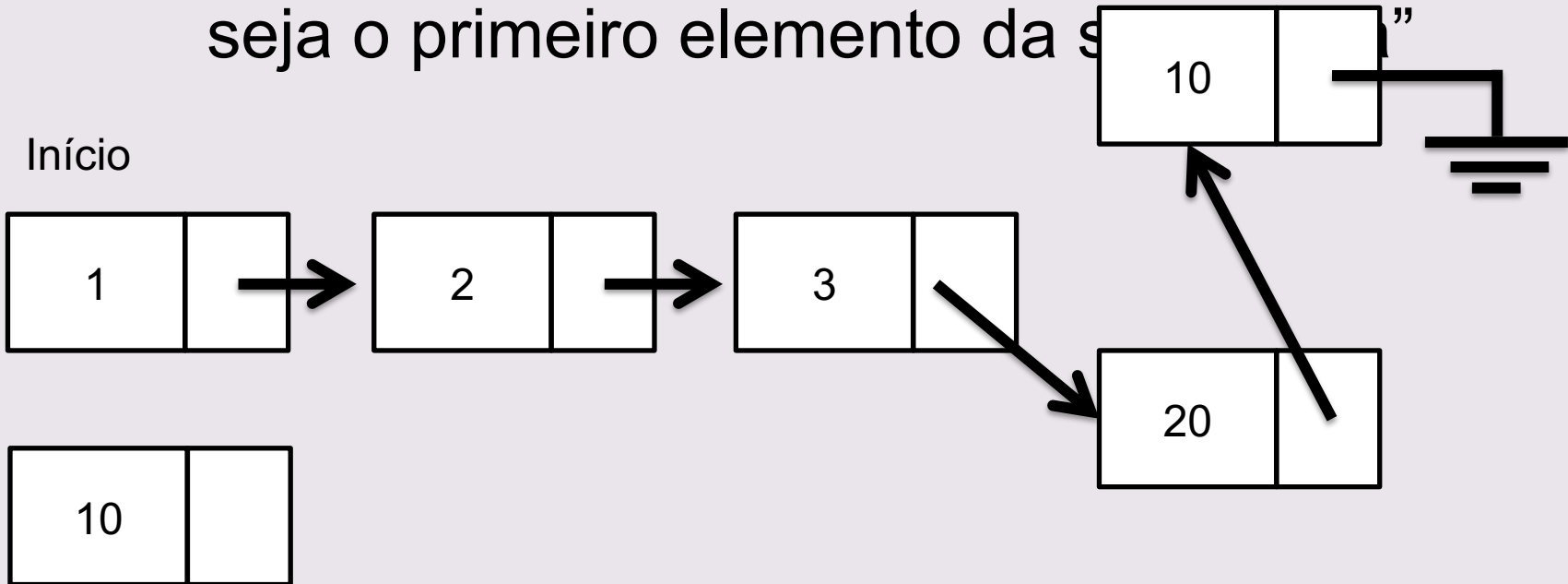
# ListSequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da s



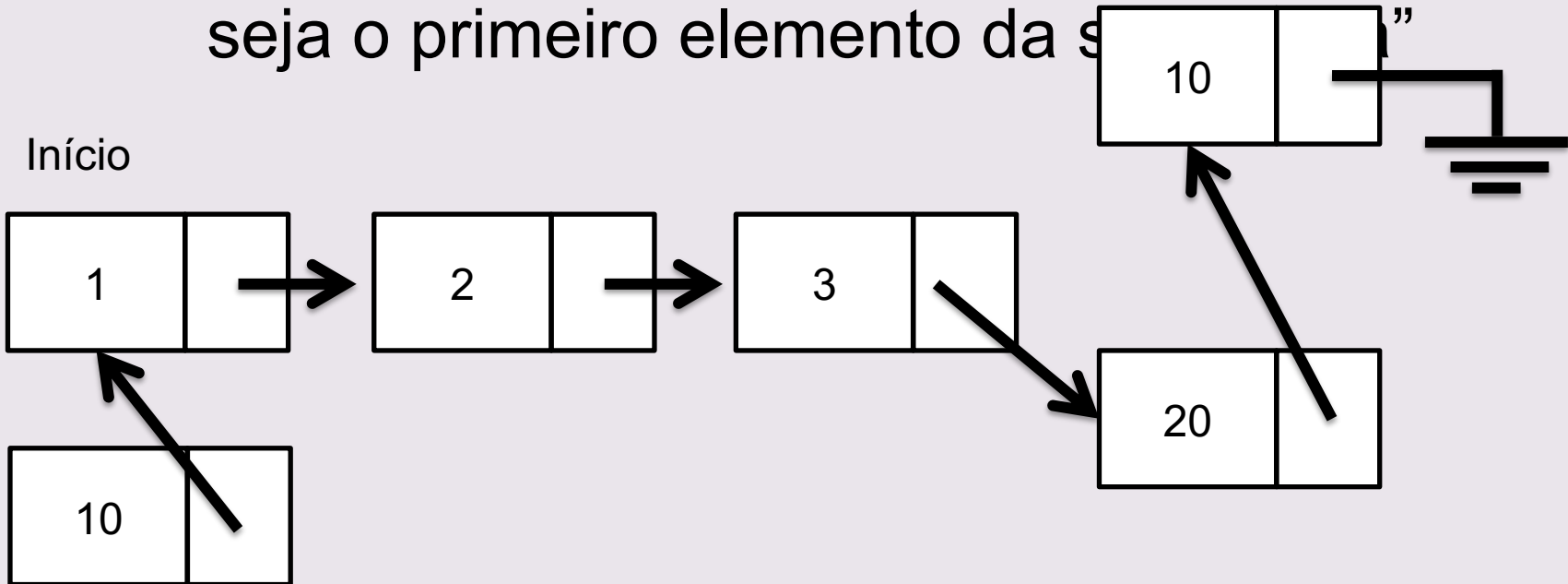
# ListSequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da s



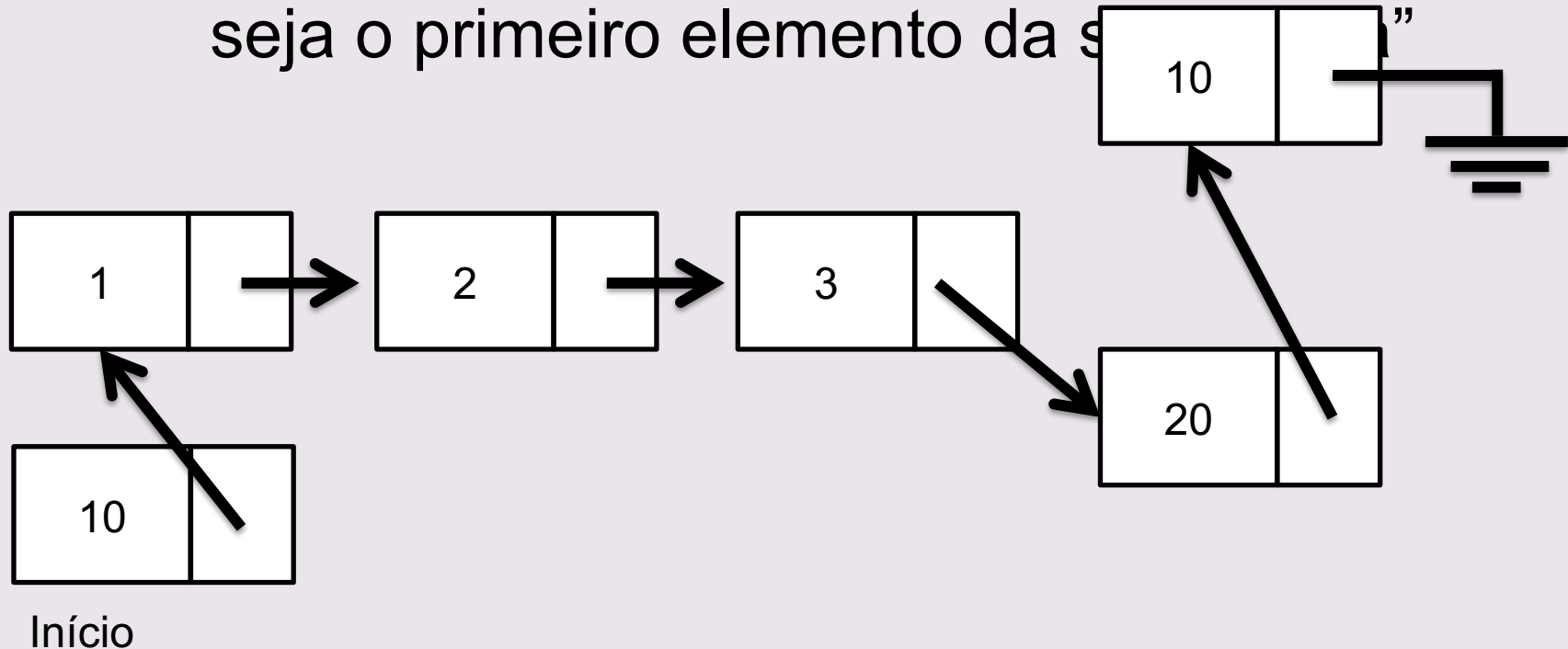
# ListSequence – Add

- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da s



# ListSequence – Add

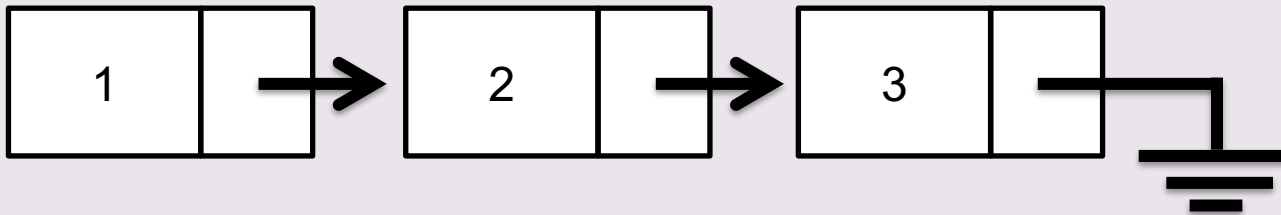
- Add( 10, 1 )
  - “Adicione o elemento 10 de modo que ele seja o primeiro elemento da s



# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

Início

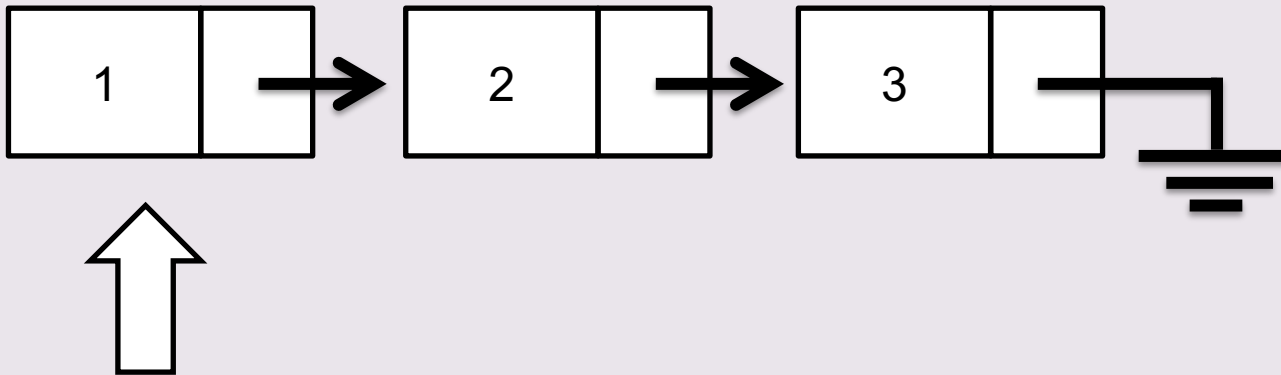




# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

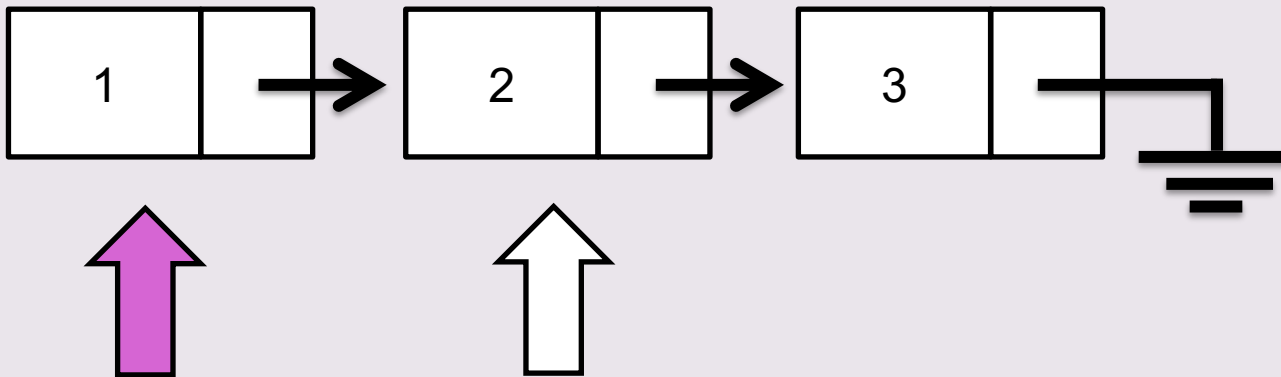
Início



# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

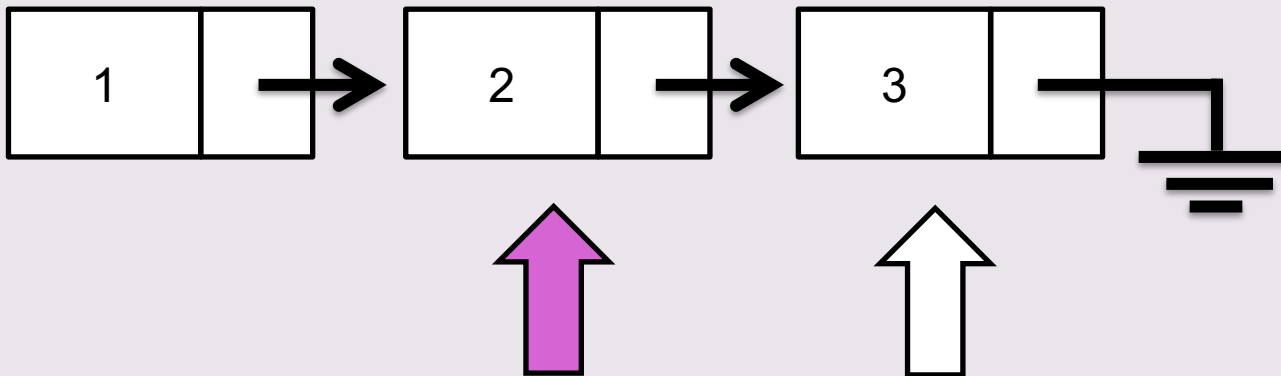
Início



# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

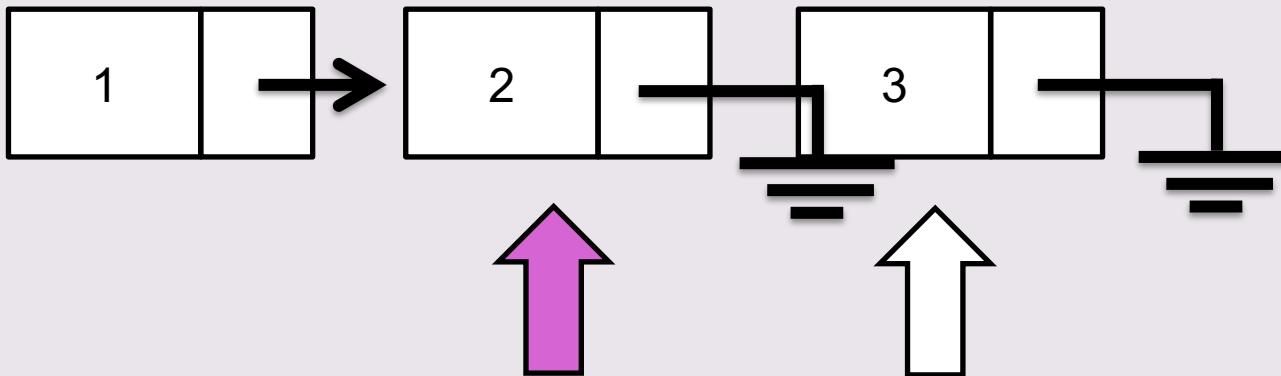
Início



# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

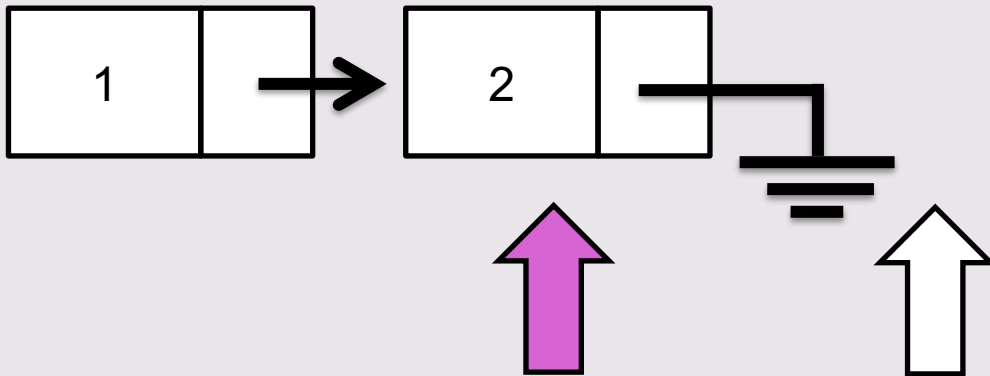
Início



# ListSequence – Remove

- Remove( 3 )
  - Remova o terceiro elemento da sequência

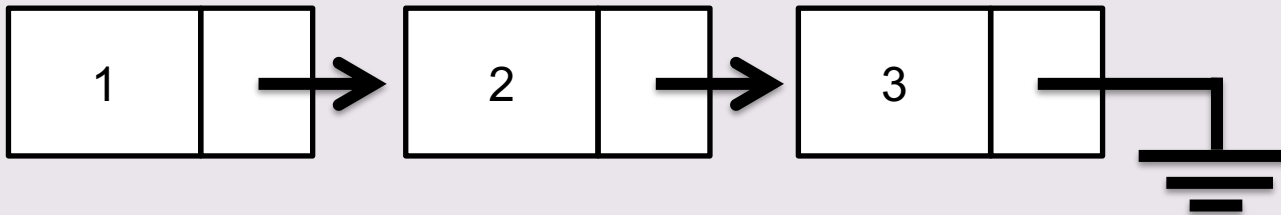
Início



# ListSequence – Remove

- Remove( 2 )
  - Remova o segundo elemento da sequência

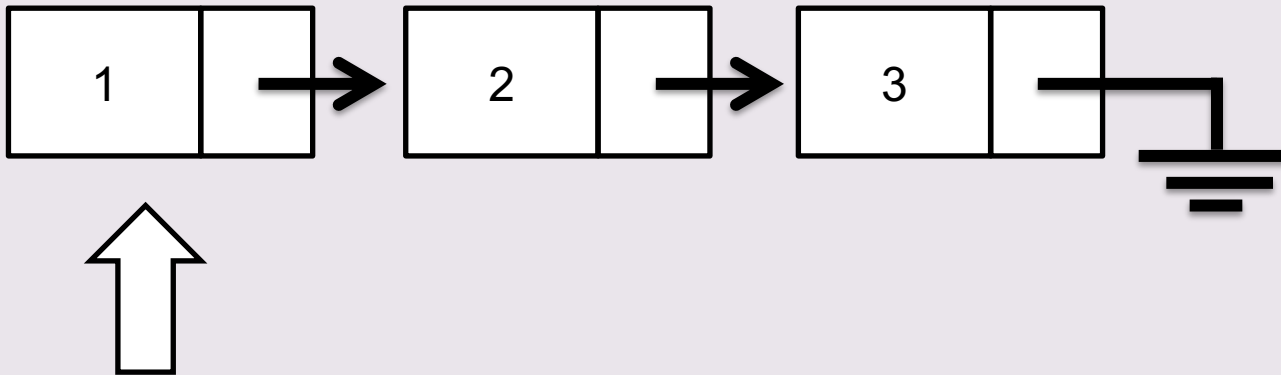
Início



# ListSequence – Remove

- Remove( 2 )
  - Remova o segundo elemento da sequência

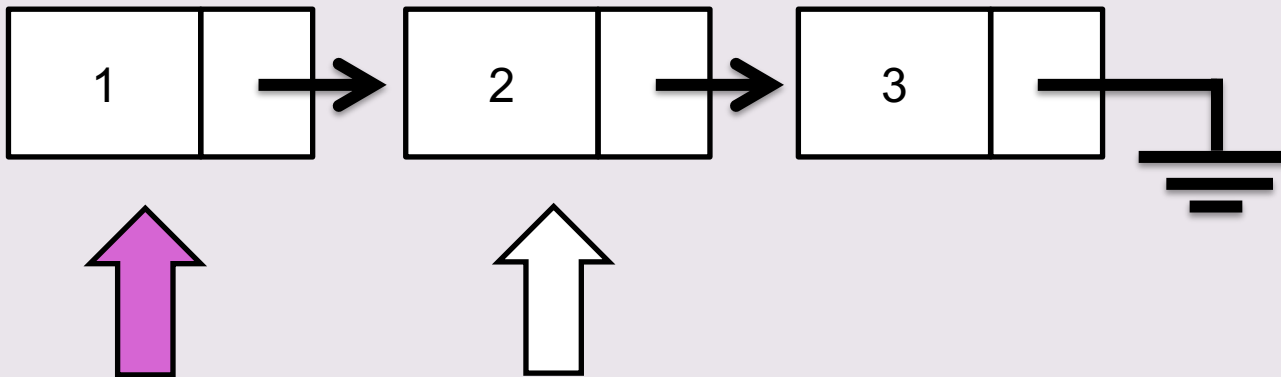
Início



# ListSequence – Remove

- Remove( 2 )
  - Remova o segundo elemento da sequência

Início

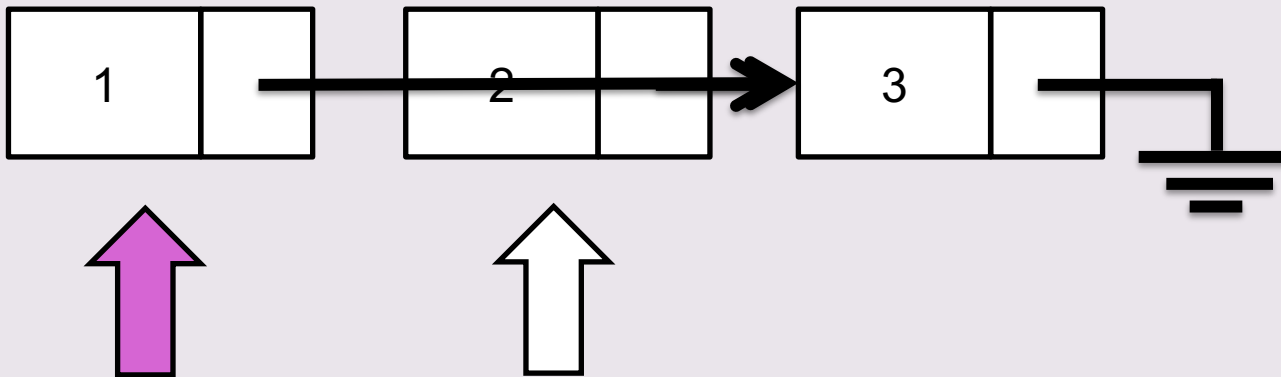




# ListSequence – Remove

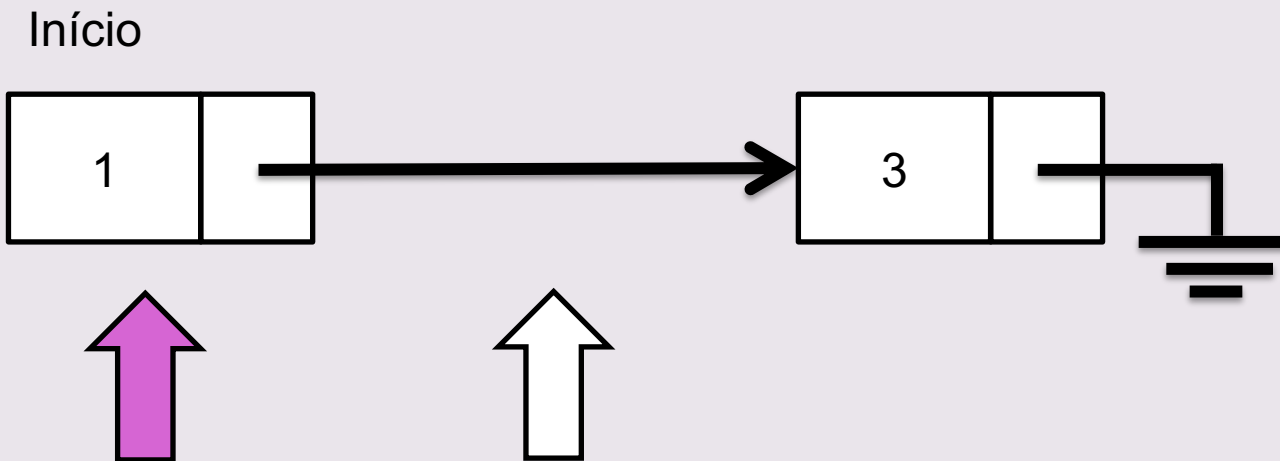
- Remove( 2 )
  - Remova o segundo elemento da sequência

Início



# ListSequence – Remove

- Remove( 2 )
  - Remova o segundo elemento da sequência



Qual a relação entre  
**Pilha, Fila e Deque**  
com Sequência?

# Casos especiais de uma Sequência

- TAD Pilha pode ser visto como um caso especial do TAD Sequência:
  - Inserção e remoção são feitas só no início, ou só no fim da lista/array
    - Push  $\equiv$  InserirInício, Pop  $\equiv$  RemoverInício  
ou
    - Push  $\equiv$  InserirFim e Pop  $\equiv$  RemoverFim

# Casos especiais de uma Sequência

- TAD Fila pode ser visto como um caso especial do TAD Sequência:
  - Inserção e remoção são feitas nos extremos opostos da lista/array
    - Queue  $\equiv$  InserirInício e Dequeue  $\equiv$  RemoverFim}  
ou
    - Queue  $\equiv$  InserirFim e Dequeue  $\equiv$  RemoverInício

# Casos especiais de uma Sequência

- TAD Deque pode ser visto como um caso especial do TAD Sequência:
  - Inserção e remoção são feitas nos dois extremos da lista/array
    - PushBack  $\equiv$  InserirFim
    - PushFront  $\equiv$  InserirInício
    - PopBack  $\equiv$  RemoverFim
    - PopFront  $\equiv$  RemoverInício

# Exercício

- Dadas duas listas simplesmente encadeadas com apenas um ponteiro para o primeiro nó da lista, projete algoritmos **iterativos** para as seguintes funções:
- List Union( List L1, List L2 ) – Retorna uma nova lista representando a união das listas L1 e L2
- List Intersection( List L1, List L2 ) – Retorna uma nova lista representando a interseção das listas L1 e L2
- int Complement( List L1, List L2 ) – Retorna uma nova lista representando o complemento de L1 em relação a L2

# Como eu poderia melhorar as operações no fim de uma Lista?

Numa remoção no fim da lista, é necessário saber quem é o nó anterior ao último nó da lista. Para isso, podemos acrescentar um ponteiro 'anterior' ao nó.

```
Estrutura Lista{  
    No início;  
    No fim;  
    tamanho;  
}
```

```
Estrutura No{  
    conteúdo;  
    No próximo;  
    No anterior;  
}
```



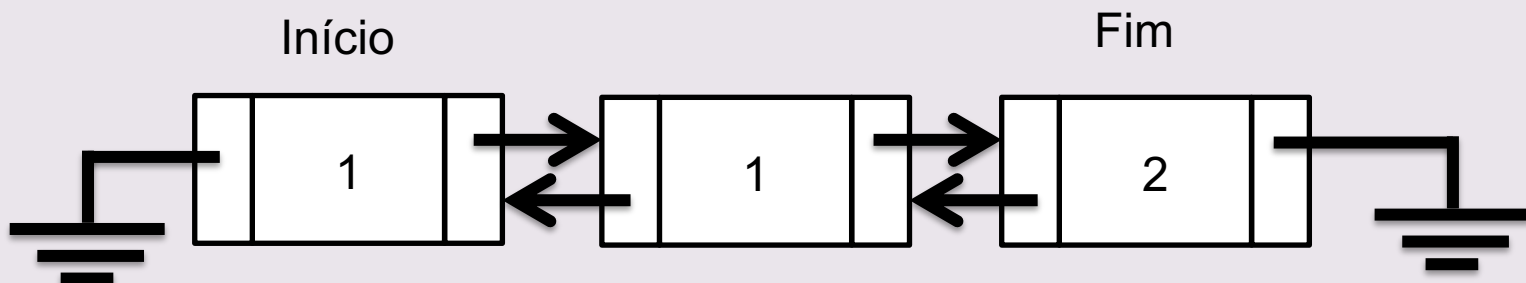
# LISTA DUPLAMENTE ENCADEADA

# Lista duplamente encadeada

## Estrutura No

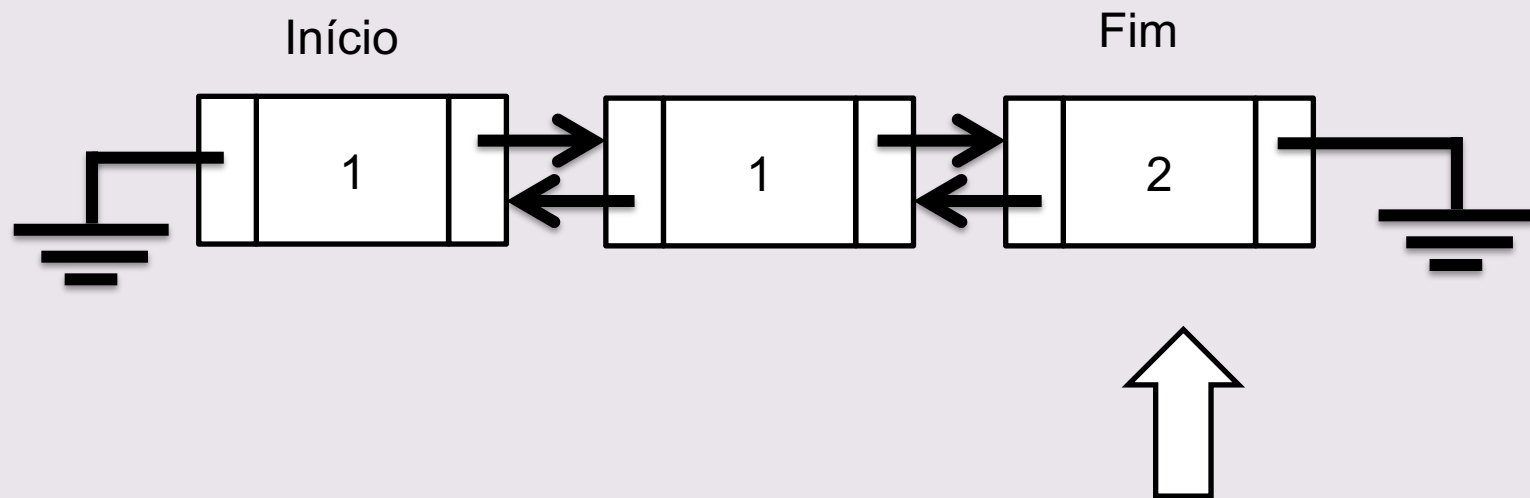


## Estrutura Lista



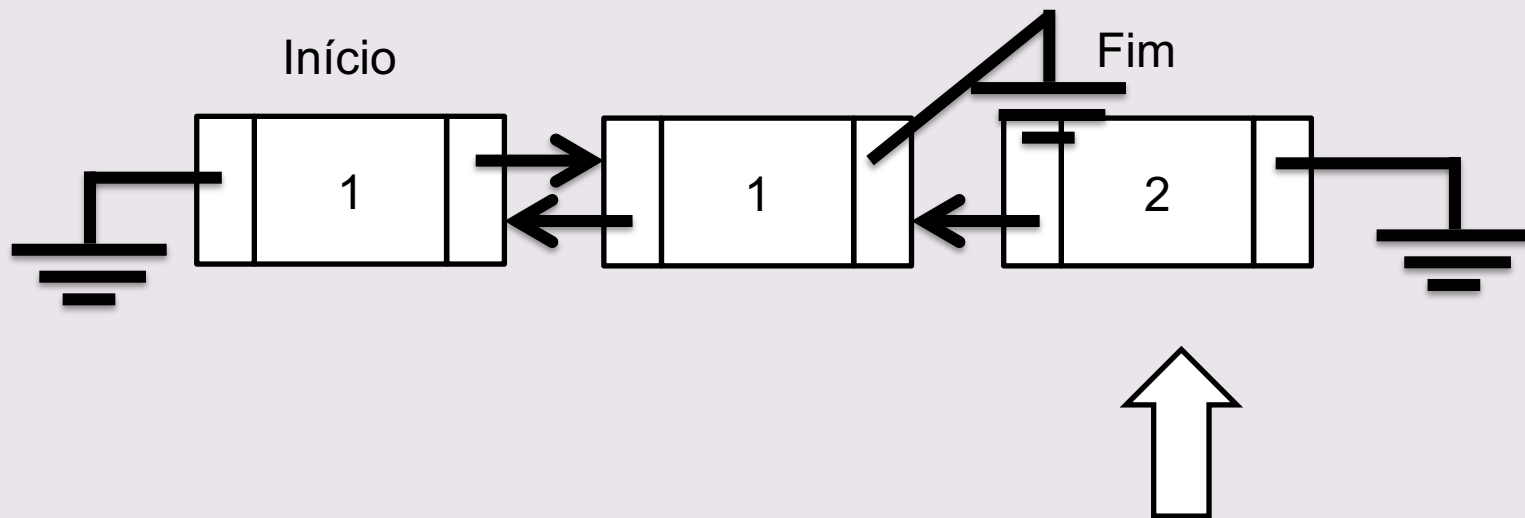
# Lista duplamente encadeada – Remoção

- Remover no fim
  - RemoveFim( )



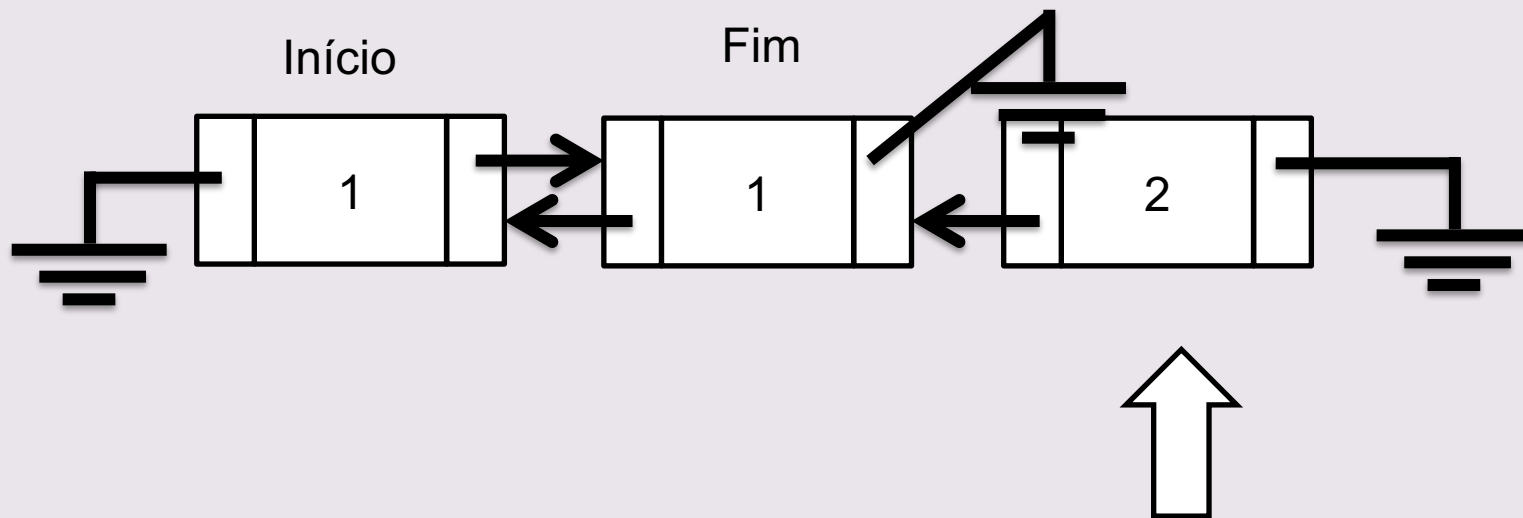
# Lista duplamente encadeada – Remoção

- Remover no fim
  - RemoveFim( )



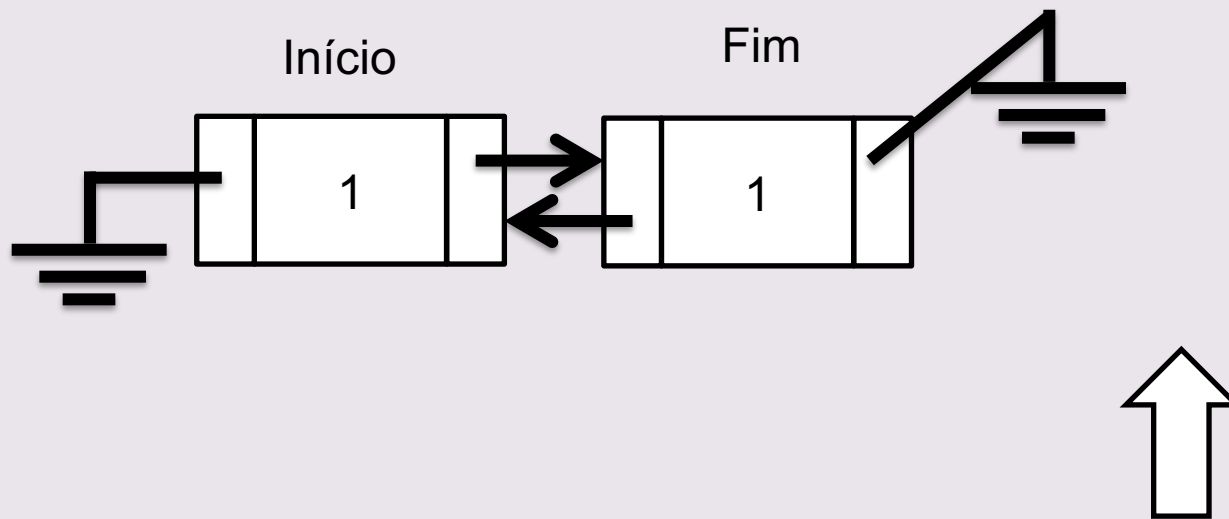
# Lista duplamente encadeada – Remoção

- Remover no fim
  - RemoveFim( )



# Lista duplamente encadeada – Remoção

- Remover no fim
  - RemoveFim( )



# Lista encadeada

## Simplex

- Inserir no fim –  $O(n)$
- Inserir no início –  $O(1)$
- **Remover no fim –  $O(n)$**
- Remover no início –  $O(1)$

## Dupla

- Inserir no fim –  $O(1)$
- Inserir no início –  $O(1)$
- **Remover no fim –  $O(1)$**
- Remover no início –  $O(1)$

Há ganho de eficiência na remoção no fim.  
As outras operações ficam um pouco mais complexas,  
pois é necessário atualizar o ponteiro 'anterior'.

# LISTA ENCADEADA COM SENTINELAS

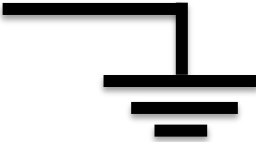


# Lista encadeada com sentinelas

- Sentinelas
  - São nós que não armazenam informações
  - Servem para delimitar extremos da lista, evitando usar NULO
  - Nunca devem ser modificados!
    - Atribuição aos nós sentinelas ocorrem única e exclusivamente no momento da criação da lista

# Lista encadeada com sentinelas

## Lista sem sentinelas

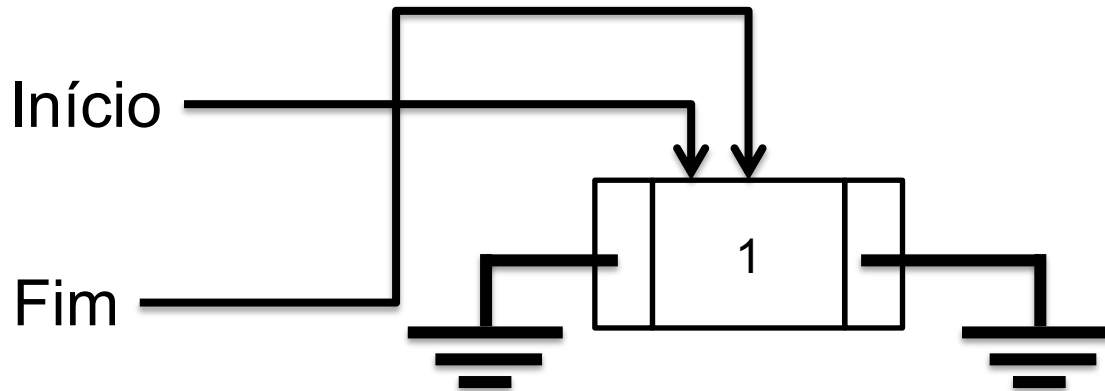
Início 

Fim 

Tamanho = 0

# Lista encadeada com sentinelas

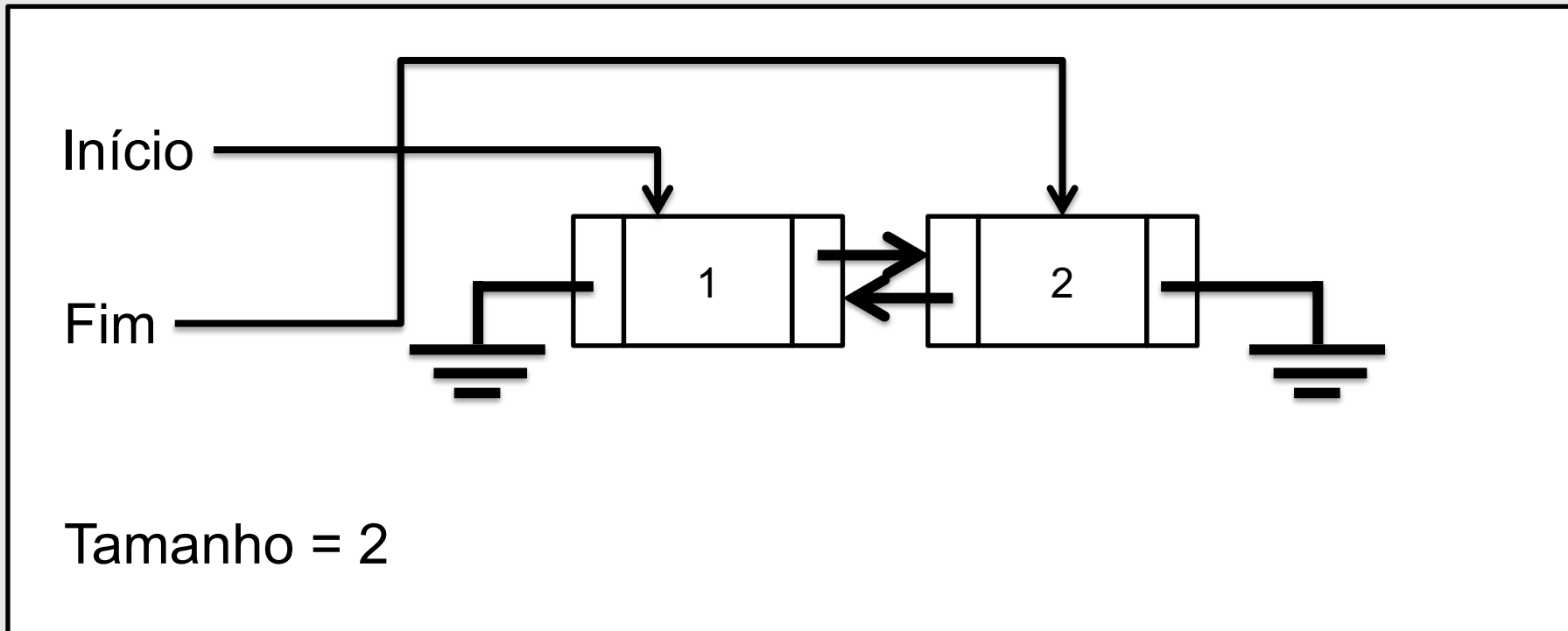
## Lista sem sentinelas



Tamanho = 1

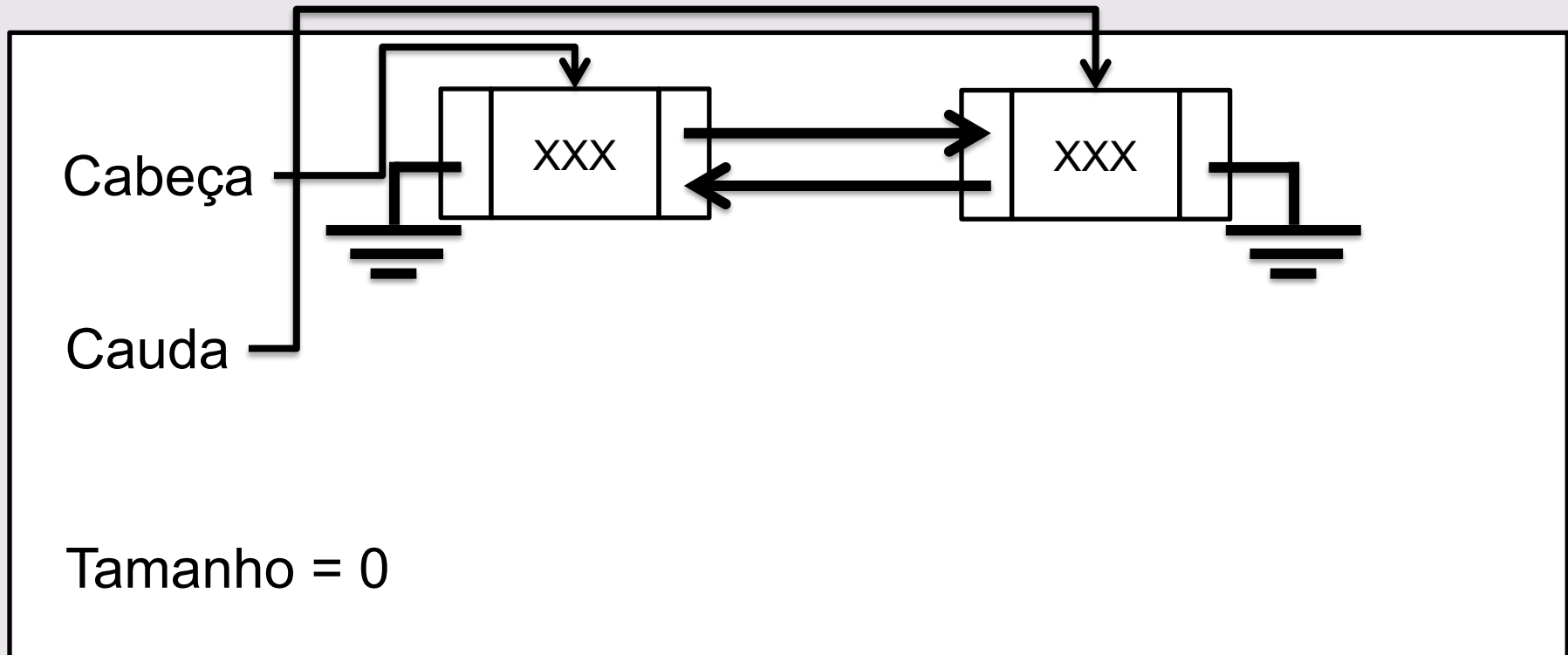
# Lista encadeada com sentinelas

## Lista sem sentinelas



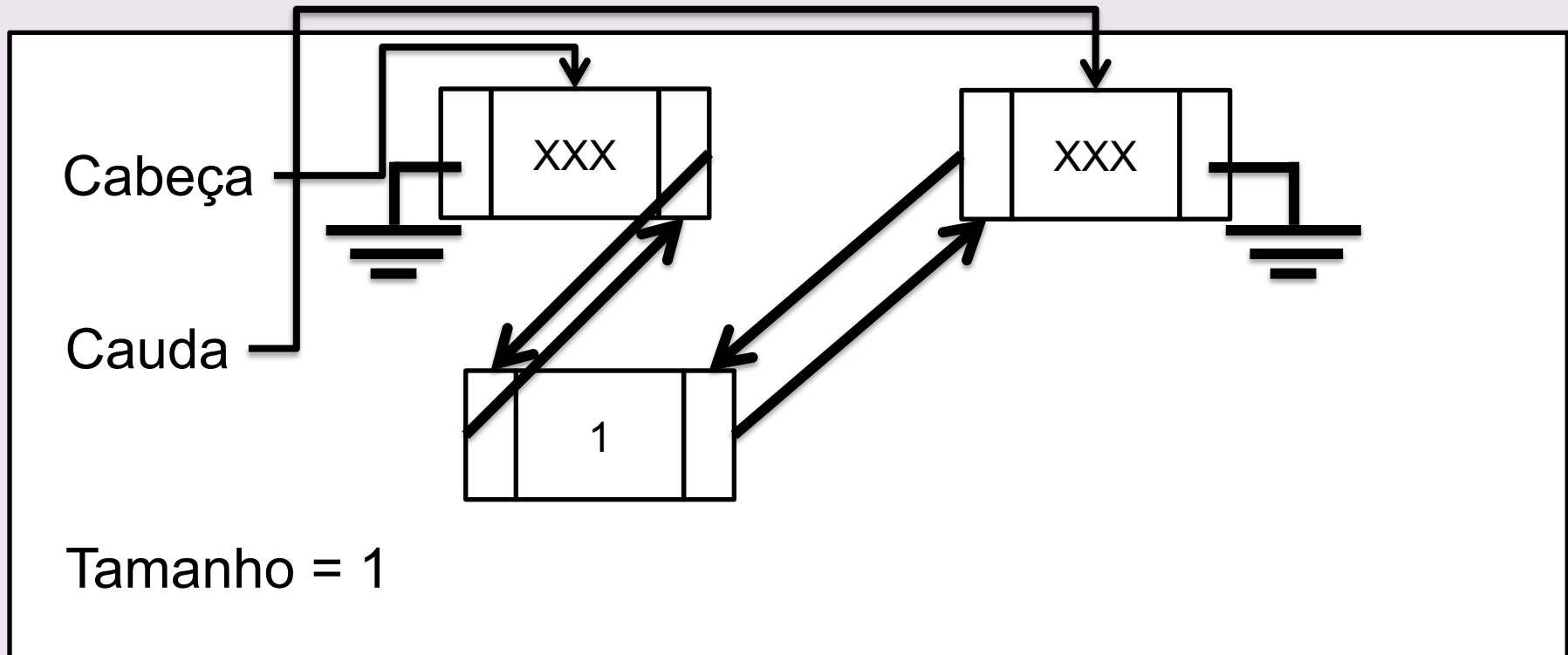
# Lista encadeada com sentinelas

Lista com sentinelas



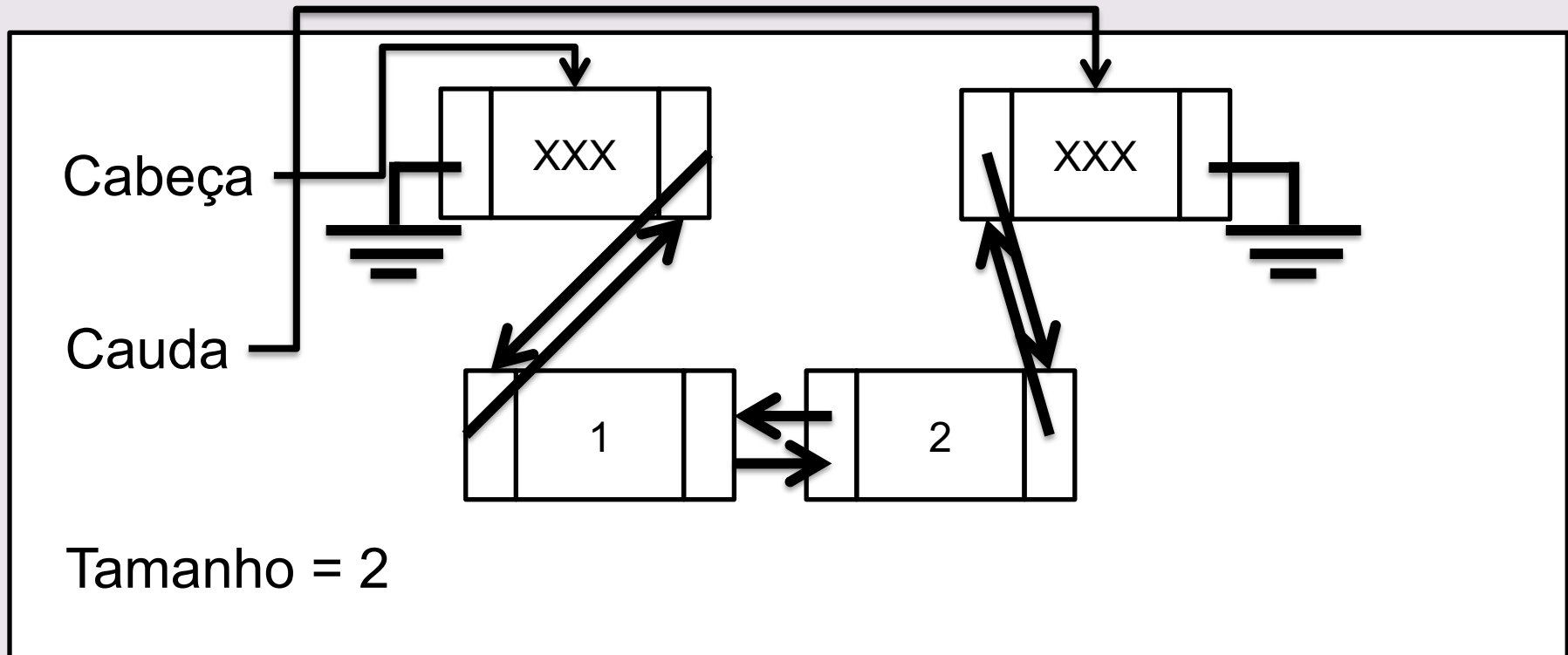
# Lista encadeada com sentinelas

Lista com sentinelas



# Lista encadeada com sentinelas

Lista com sentinelas



# Lista encadeada com sentinelas

```
Estrutura Lista{  
    No cabeça;  
    No cauda;  
    tamanho;  
}
```

```
Estrutura No{  
    conteúdo;  
    No próximo;  
    No anterior;  
}
```



# Lista encadeada com sentinelas

- Criar lista

```
CriarLista( ):  
    Lista lista = Alocar Memória Para Lista  
    lista.cabeça = CriarNo( valor-qualquer )  
    lista.cauda  = CriarNo( valor-qualquer )  
  
    lista.cabeça.próximo = lista.cauda  
    lista.cauda.anterior = lista.cabeça  
  
    lista.cabeça.anterior = NULO  
    lista.cauda.próximo  = NULO  
  
    lista.tamanho = 0
```

FIM

# Lista encadeada com sentinelas

- Percorrer lista

Evita usar NULO

```
PercorrerLista( lista ):  
  No atual = lista.cabeça.próximo  
  ENQUANTO atual != lista.cauda FAÇA  
    // Faça alguma operação com nó da lista  
    atual = atual.próximo  
  FIM_ENQUANTO  
FIM
```

# Lista encadeada com sentinelas

- Inserir no início

```
InserirInício( lista, valor ):  
  No nó = CriarNo( valor )  
  No cabeça = lista.cabeça  
  
  nó.próximo = cabeça.próximo  
  nó.anterior = cabeça  
  
  nó.próximo.anterior = nó  
  nó.anterior.próximo = nó  
  
  lista.tamanho = lista.tamanho+1  
FIM
```

Não precisa checar  
se é NULO

# Lista encadeada com sentinelas

- Inserir no Fim

```
InserirFim( lista, valor ):  
    No nó = CriarNo( valor )  
    No cauda = lista.cauda  
  
    nó.próximo = cauda  
    nó.anterior = cauda.anterior  
  
    nó.próximo.anterior = nó  
    nó.anterior.próximo = nó  
  
    lista.tamanho = lista.tamanho+1  
FIM
```

Não precisa checar  
se é NULO

# Lista encadeada com sentinelas

- Remover no início

```
RemoverInício( lista ):  
    SE lista está vazia ENTÃO  
        RETORNE ERRO-ListaVazia  
    FIM_SE
```

```
No aRemover = lista.cabeça.próximo  
valor = aRemover.conteúdo
```

```
nó.anterior.próximo = nó.próximo  
nó.próximo.anterior = nó.anterior  
delete nó
```

```
lista.tamanho = lista.tamanho-1  
FIM
```

# Lista encadeada com sentinelas

- Remover no fim

```
RemoverFim( lista ):  
    SE lista está vazia ENTÃO  
        RETORNE ERRO-ListaVazia  
    FIM_SE
```

```
No aRemover = lista.cauda.anterior  
valor = aRemover.conteúdo
```

```
nó.anterior.próximo = nó.próximo  
nó.próximo.anterior = nó.anterior  
delete nó
```

```
lista.tamanho = lista.tamanho-1  
FIM
```

1. Faça uma função que recebe uma lista duplamente encadeada e remover os nós com valores duplicados.
2. Dadas duas listas duplamente encadeadas com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

List Merge(List L1, List L2) – Retorna uma nova lista contendo todos os elementos de L1 e L2 em ordem crescente. Pode assumir que as listas L1 e L2 já estão ordenadas em ordem decrescente.

*Dica: Esta solução é apenas uma versão do algoritmo merge usado no MergeSort. No MergeSort visto em sala de aula, nós operamos sobre arrays. Para esta solução, deveremos operar sobre listas duplamente encadeadas.*

3. Dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

void RemoveRepeated(List L1) – Remove os elementos repetidos da lista L1, retornando quantos elementos foram removidos.

*Obs.: Esta função não deverá remover todos os elementos cujos valores se repetem; ele deverá deixar o primeiro elemento na lista, removendo os seguintes. Ex.: A lista { 7 ↔ 2 ↔ 4 ↔ 2 ↔ 3 ↔ 9 ↔ 9 ↔ 3 }, após a execução da função deverá ficar igual a { 7 ↔ 2 ↔ 4 ↔ 3 ↔ 9 }.*

# Estrutura de Dados

Conjuntos – Array

**Prof<sup>a</sup>. Anna Giselle Ribeiro**