# Dominando Estruturas de Dados 1

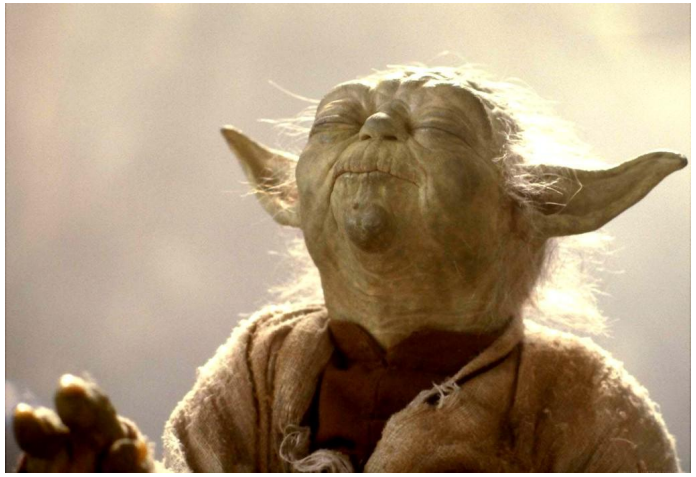
## Listas Encadeadas Simples

Prof. dr. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding

# 1º Mandamento de Estruturas de Dados

*"Sempre desenhar a estrutura de dados antes de implementá-la, você deve."*



# Um problema em vetores/arrays

- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que **a quantidade de alunos é 5**;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

# Um problema em vetores/arrays

- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que **a quantidade de alunos é 5**;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

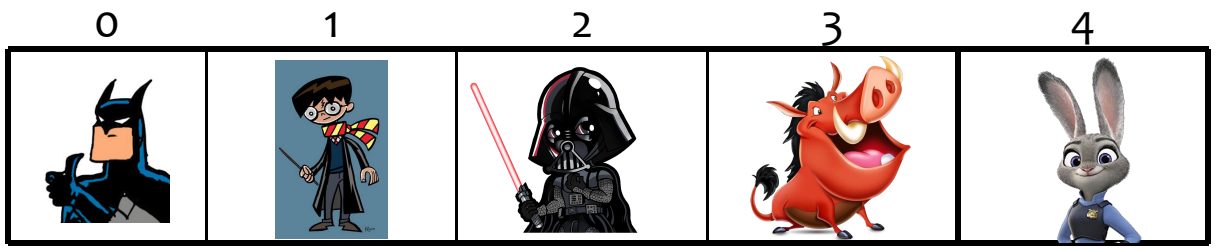
```
Aluno **alunos = (Aluno**) calloc(5, sizeof(Aluno*));
for (int i = 0; i < 5; i++) {
    alunos[i] = create_aluno();
}
```

# Um problema em vetores/arrays

- Suponha que queremos armazenar um lista de alunos em um **vetor/array**;
- Sabemos, inicialmente, que **a quantidade de alunos é 5**;
- Então, alocamos um vetor de 5 posições e inserimos os alunos;

```
Aluno **alunos = (Aluno**) calloc(5, sizeof(Aluno*));
for (int i = 0; i < 5; i++) {
    alunos[i] = create_aluno();
}
```

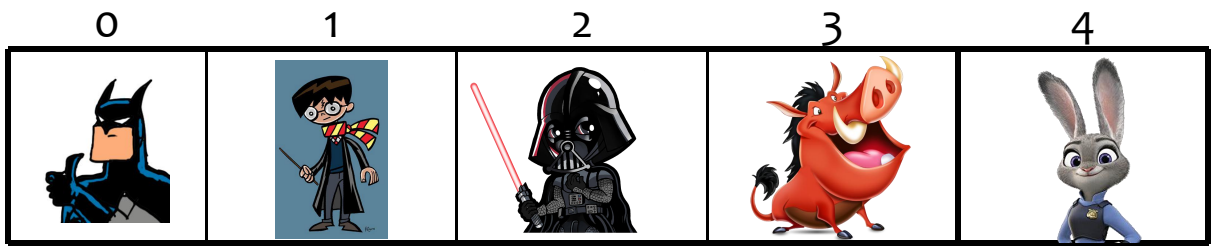
alunos =



# Um problema em vetores/arrays

- E se agora quiséssemos **adicionar** um novo aluno neste vetor?
- O vetor atual **não tem espaço suficiente**. O que fazer?

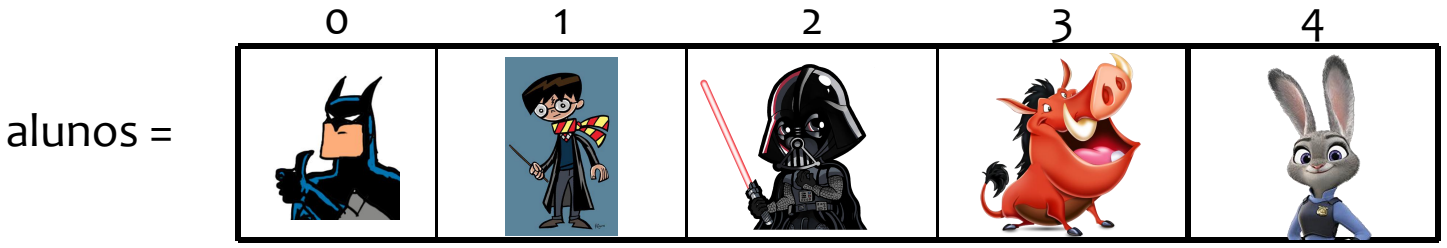
alunos =



# Um problema em vetores/arrays

**Opção 1:**

- Existe uma função em C chamada ***realloc*** que realoca um espaço de memória —> comando perigoso
- Quando mal usado, acarretará em perda de dados
- **Não indicado!**

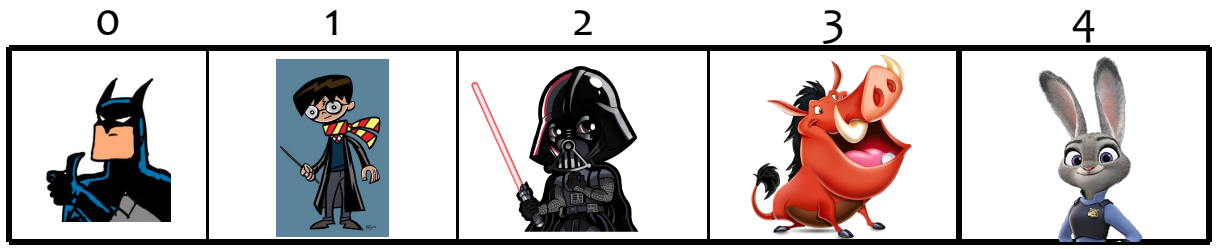


# Um problema em vetores/arrays

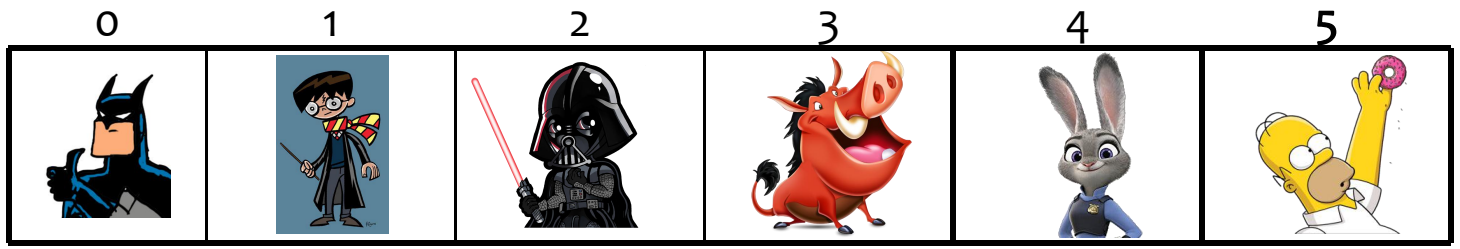
**Opção 2:**

- Criar um vetor com **6** posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;

alunos =




novos =

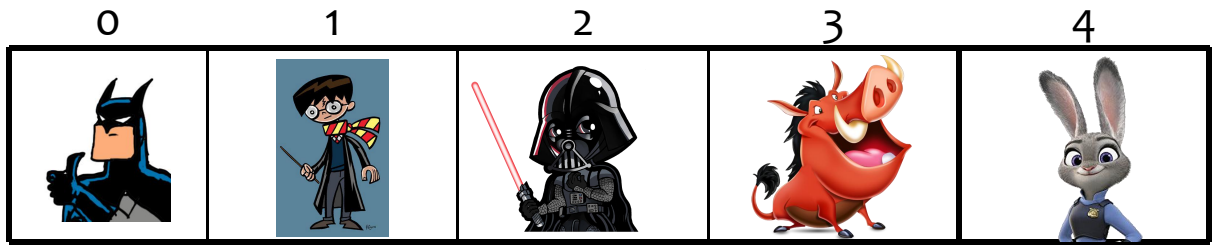


# Um problema em vetores/arrays

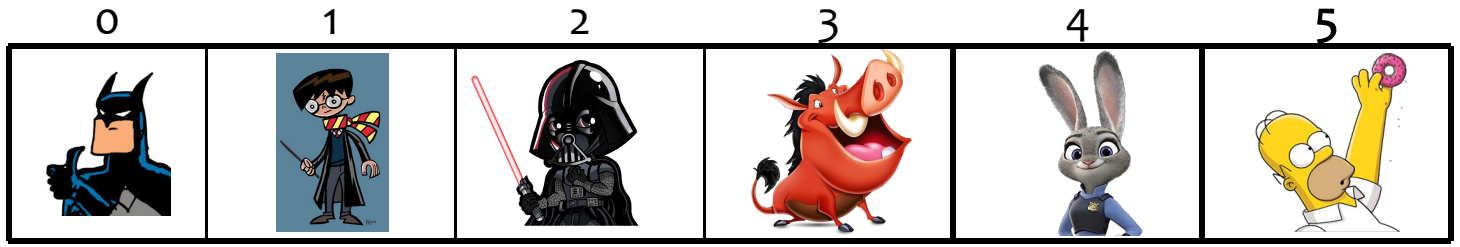
**Opção 2:**

- Criar um vetor com **6** posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;
- **Problemas?**

alunos =




novos =



# Um problema em vetores/arrays

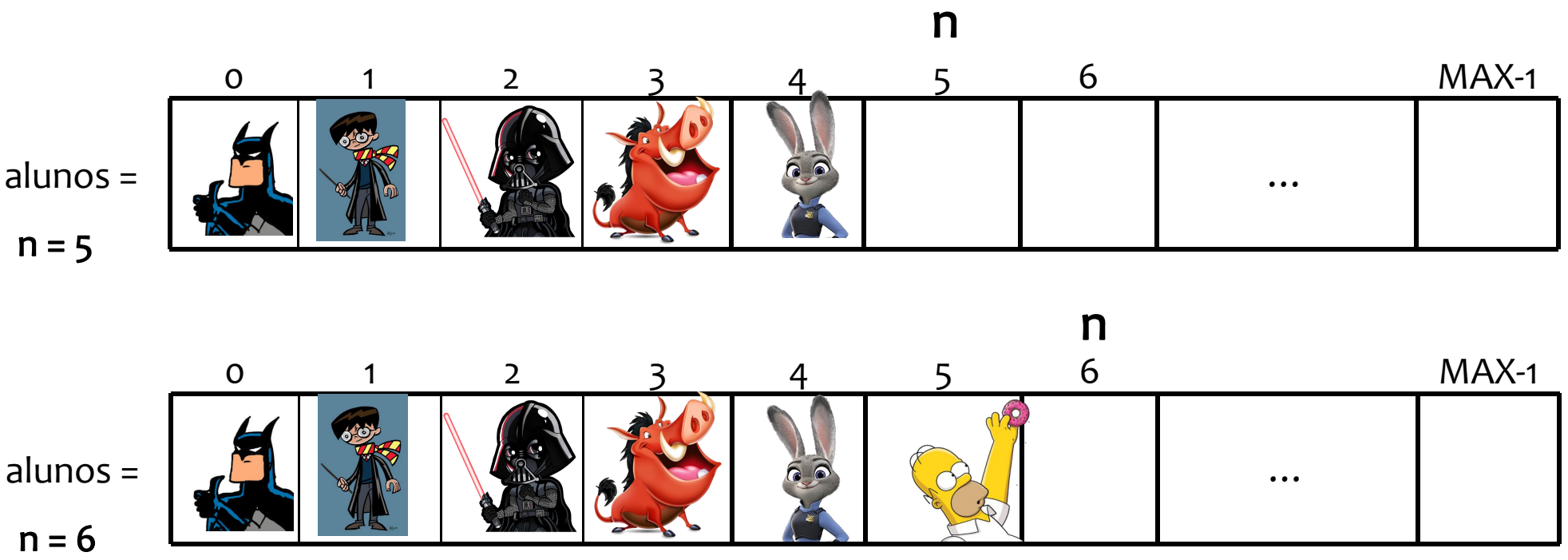
## Opção 2:

- Criar um vetor com **6** posições e **copiar todos** os valores do antigo vetor e **adicionar** o novo aluno;
- **Problemas?**
  - A cada novo aluno, precisaremos alocar um novo vetor;
  - Copiar os dados de um vetor a outro… **ineficiente**;

# Um problema em vetores/arrays

**Opção 3:**

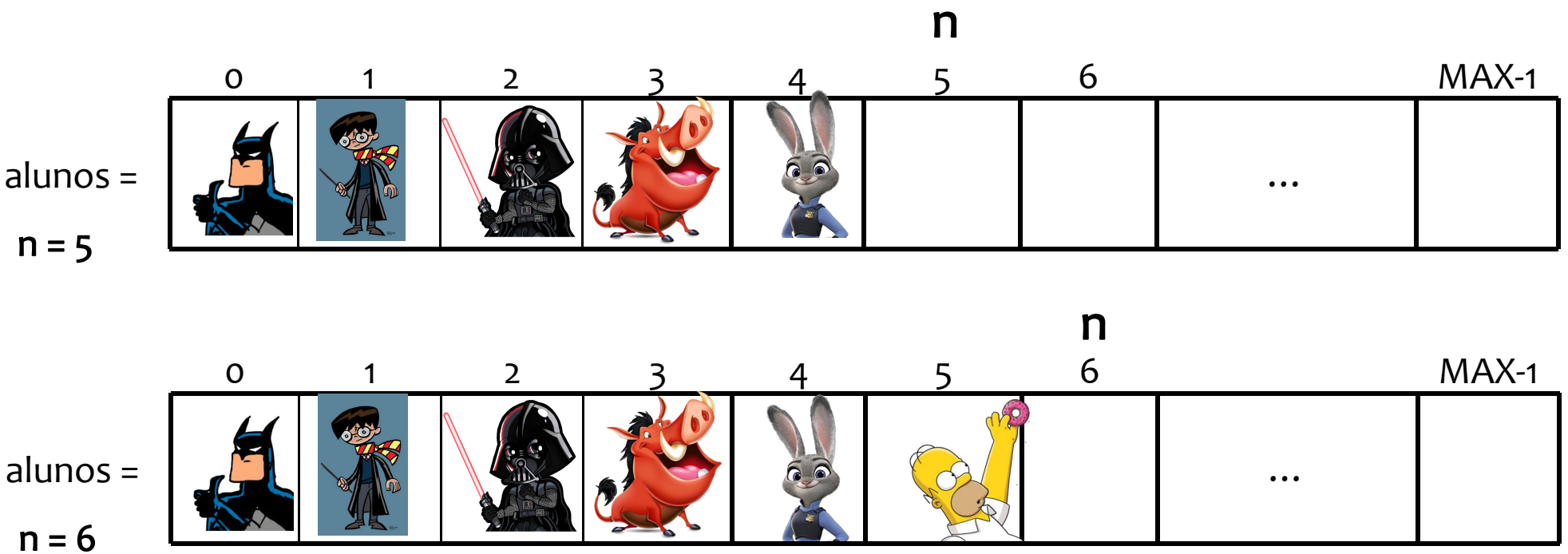
- Criamos **um vetor muito grande**, com muitas posições, de modo que **"sempre"** teremos espaço para inserir um novo aluno;



# Um problema em vetores/arrays

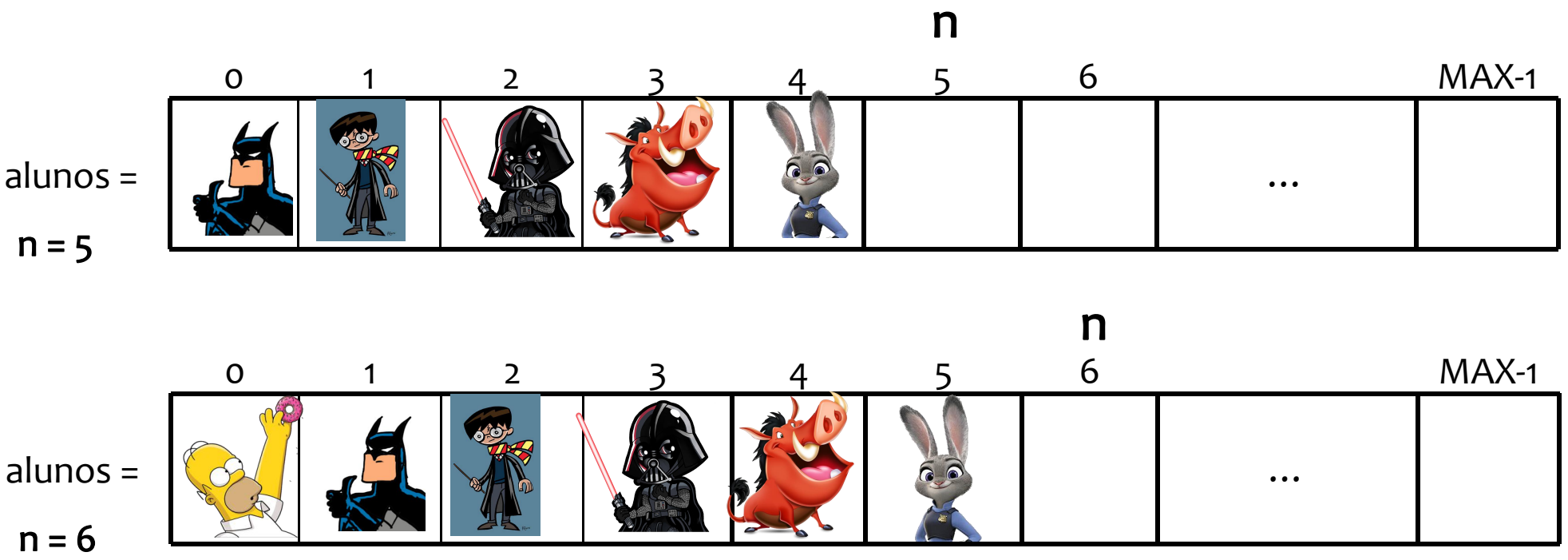
**Opção 3:**

- Criamos **um vetor muito grande**, com muitas posições, de modo que **"sempre"** teremos espaço para inserir um novo aluno;
- Problemas?



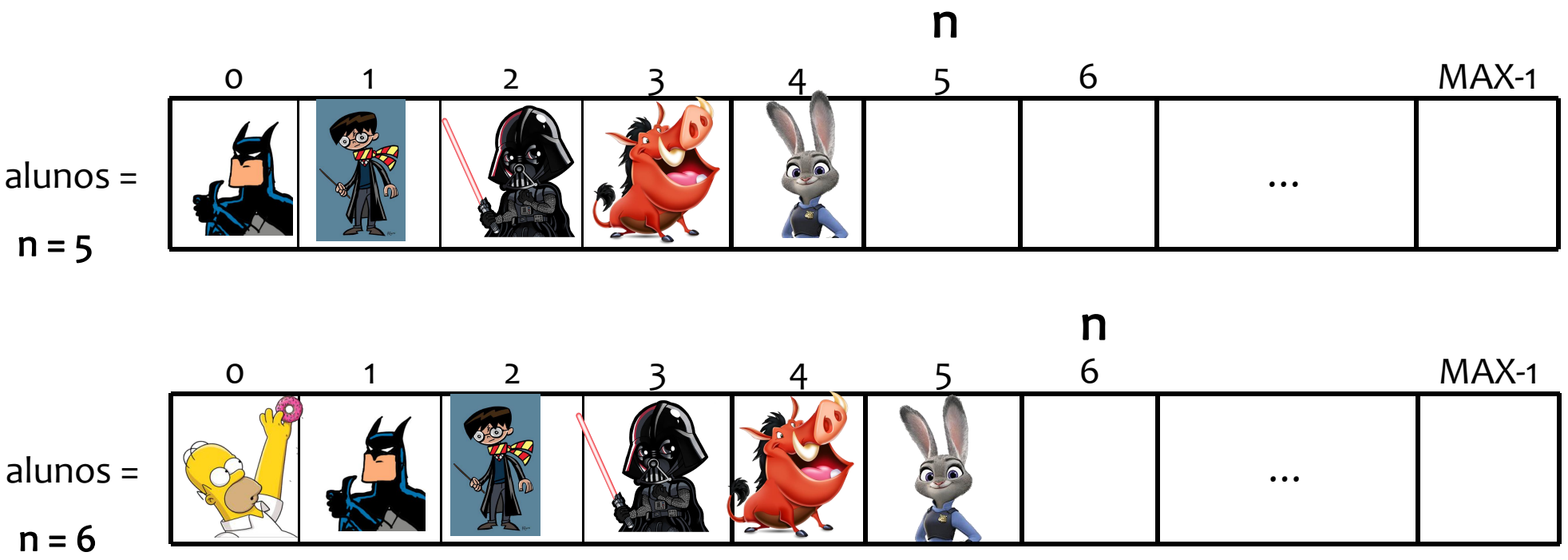
# Outro problema em vetores/arrays

- Suponha que adotamos a **opção 3** para a implementação;
- Suponha que um novo aluno **sempre** deva ser inserido no **começo** do vetor;
- O que fazer?



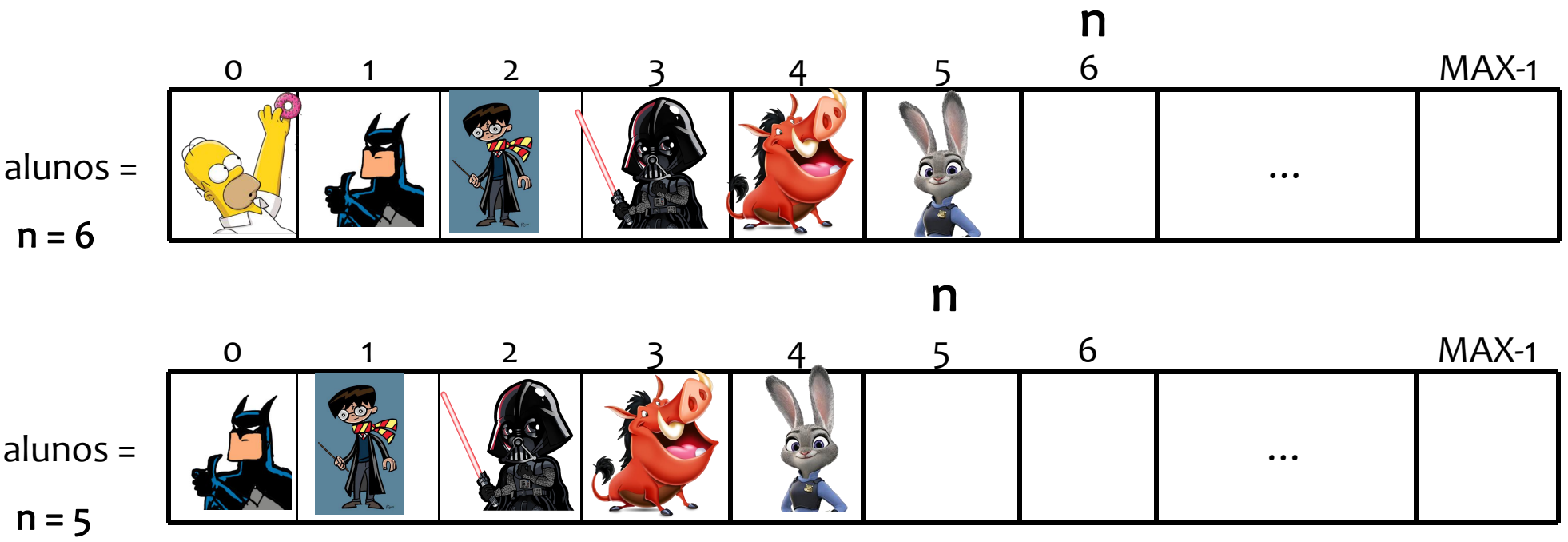
# Outro problema em vetores/arrays

- Suponha que adotamos a **opção 3** para a implementação;
- Suponha que um novo aluno **sempre** deva ser inserido no **começo** do vetor;
- O que fazer?
- **Problemas?**



# Outro problema em vetores/arrays

- O mesmo problema acontecerá se quisermos **remover** um aluno, mas quisermos manter a ordem a qual eles foram inseridos no vetor;



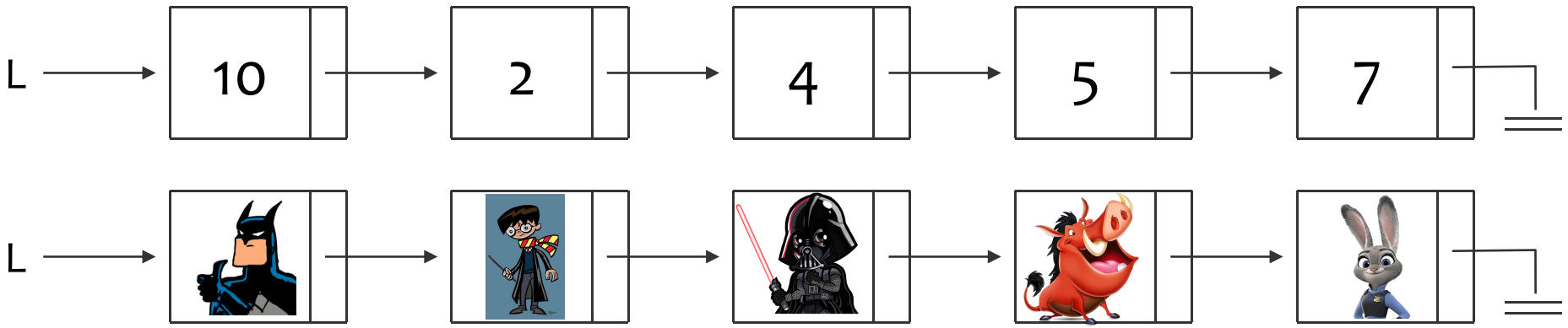
# Listas Encadeadas/Ligadas

# Lista Encadeada

Uma **lista encadeada (ligada)** é uma representação de uma **sequência de elementos/objetos** na memória do computador;

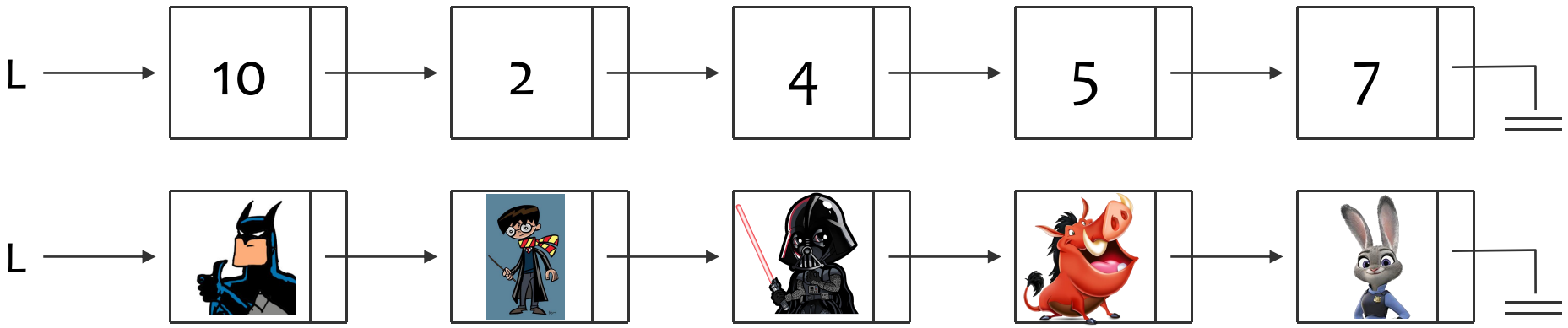
Os elementos, **nós da lista,** são armazenados em **posições quaisquer da memória** e são ligados por **ponteiros**;

- Logo, **elementos consecutivos** da lista **não** ficam necessariamente em **posições consecutivas na memória**;



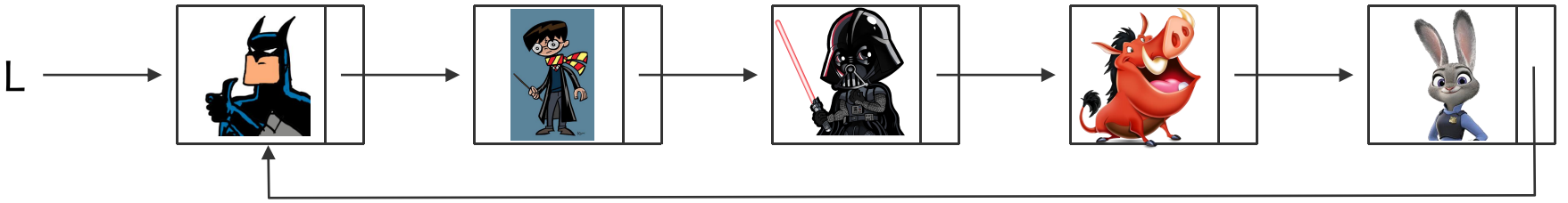
# Lista Encadeada

- Cada **nó** contém um **elemento/objeto** de determinado tipo e um **ponteiro** para o **próximo elemento da lista** --> Lista Encadeada Simples;
- No caso do **último nó**, este ponteiro aponta para **NULL**;



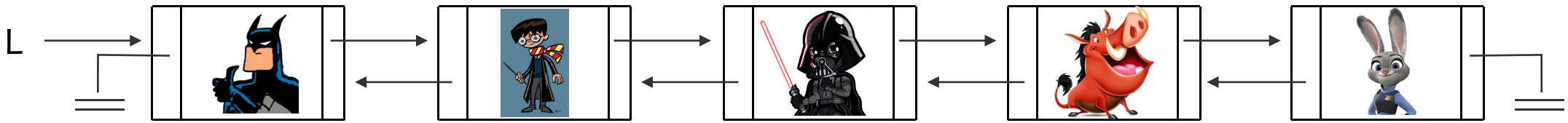
# Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
  - Listas Circulares;



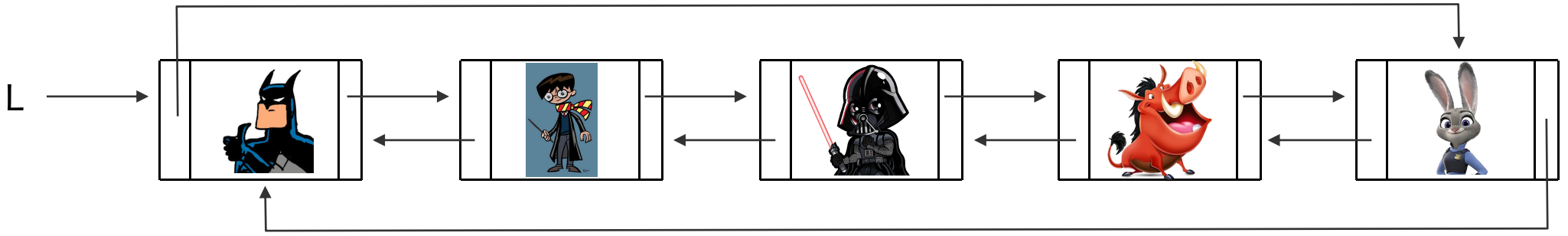
# Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
  - Listas Circulares;
  - **Listas Duplamente Encadeadas;**



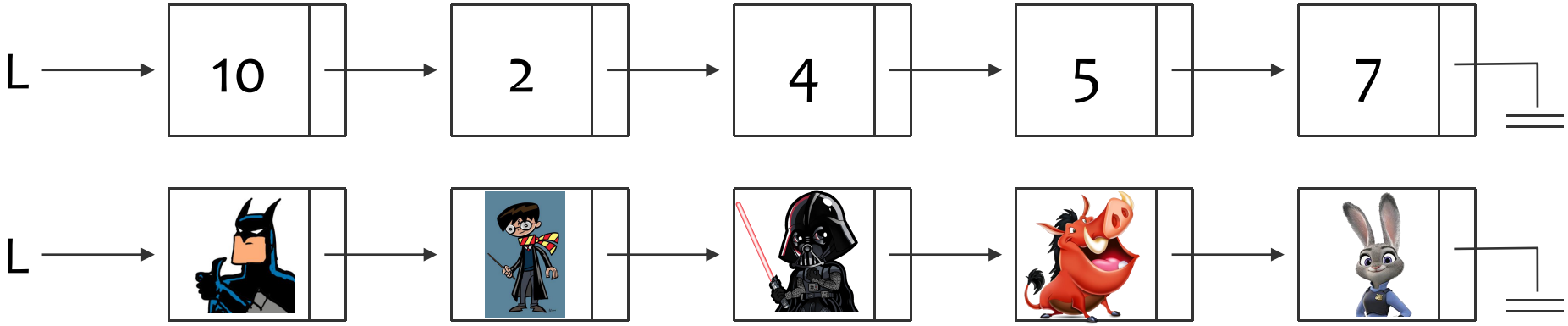
# Lista Encadeada

- Podemos ainda ter outros tipos de listas (que detalharemos depois):
  - Listas Circulares;
  - Listas Duplamente Encadeadas;
  - **Listas Circulares Duplamente Encadeadas;**



# Listas Encadeadas Simples

# Lista Encadeada Simples

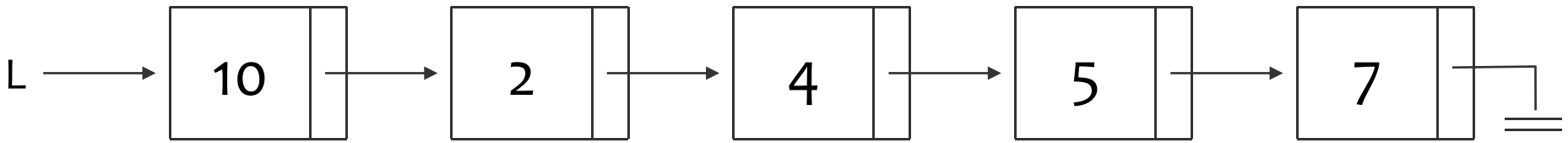


# Lista Encadeada Simples

**Implementação:**

- Uma Lista Encadeada é uma **sequência** de **Nós**;
- Se o **ponteiro L** aponta para **o primeiro nó da Lista**, podemos dizer que **L é uma Lista**;

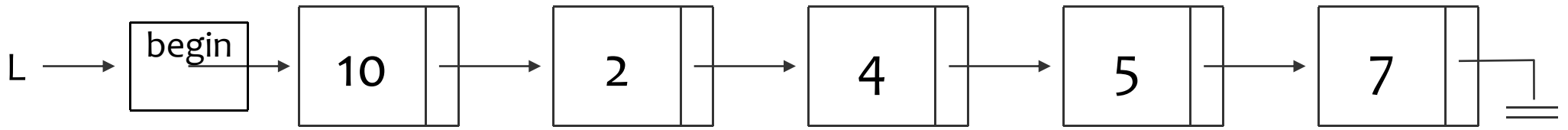
```
typedef struct _node {
    int val;
    struct _node *next;
} Node;
```



# Lista Encadeada Simples

**Outra Implementação:**

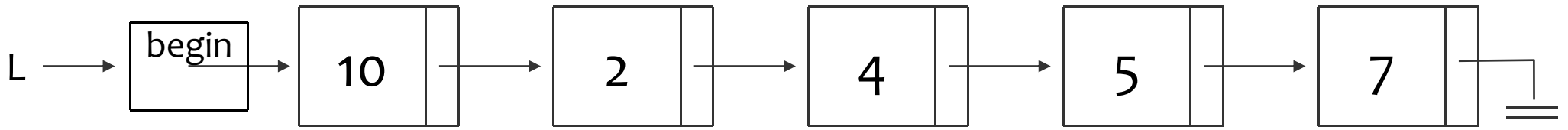
- Temos um **tipo próprio** para a Lista, que guardará um ponteiro para seu **início**;



# Lista Encadeada Simples

**Outra Implementação:**

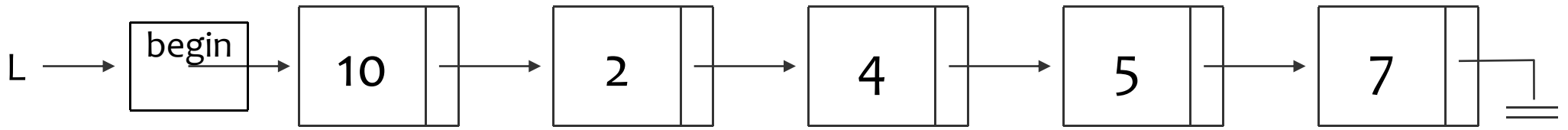
- Temos um **tipo próprio** para a Lista, que guardará um ponteiro para seu **início**;
- Este novo tipo nos dá a liberdade para **guardamos outras informações**, como o número de nós da Lista, outros ponteiros, etc...



# Lista Encadeada Simples

**Outra Implementação:**

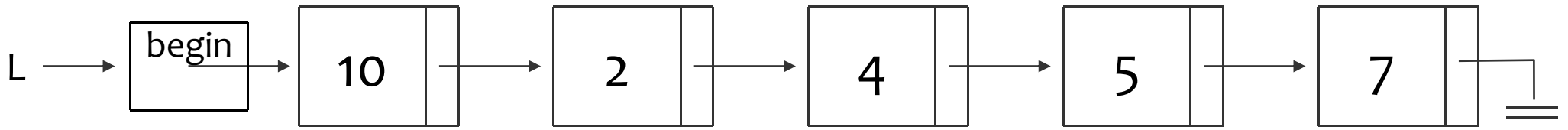
- Temos um **tipo próprio** para a Lista, que guardará um ponteiro para seu **início**;
- Este novo tipo nos dá a liberdade para **guardamos outras informações**, como o número de nós da Lista, outros ponteiros, etc...
- Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas (veremos jajá);



# Lista Encadeada Simples

**Outra Implementação:**

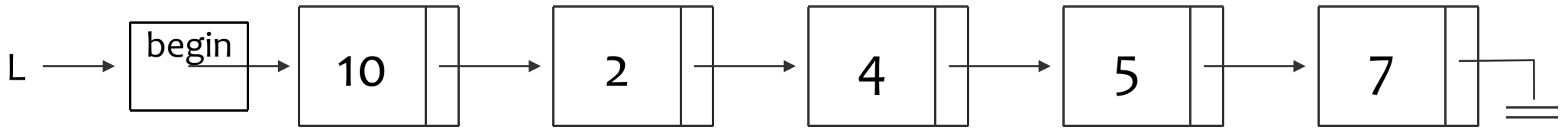
- Temos um **tipo próprio** para a Lista, que guardará um ponteiro para seu **início**;
- Este novo tipo nos dá a liberdade para **guardamos outras informações**, como o número de nós da Lista, outros ponteiros, etc...
- Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas (veremos jajá);
- Esta solução é parecida com a Lista Encadeada **com Cabeça**, porém **mais robusta**;



# Lista Encadeada Simples

**Outra Implementação:**

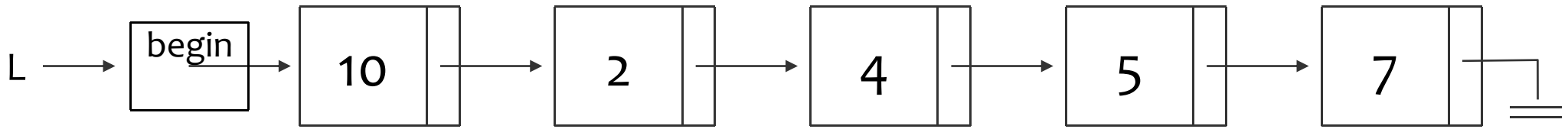
- Temos um **tipo próprio** para a Lista, que guardará um ponteiro para seu **início**;
- Este novo tipo nos dá a liberdade para **guardamos outras informações**, como o número de nós da Lista, outros ponteiros, etc...
- Com isso, podemos **otimizar** certas operações comuns de Listas Encadeadas (veremos jajá);
- Esta solução é parecida com a Lista Encadeada **com Cabeça**, porém **mais robusta**;
- Utilizaremos essa versão;



# Lista Encadeada Simples

```c
typedef struct _node {
    int val;
    struct _node *next;
} Node;
```

```c
typedef struct _linked_list {
    Node *begin;
} LinkedList;
```



# Lista Encadeada Simples

Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

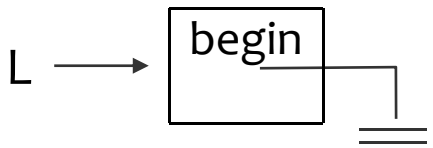
# Lista Encadeada Simples

Diversos tipos de operações:

- **Inserção na cabeça (início) da lista;**
- Impressão dos Elementos da Lista
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Inserção na cabeça (início) da lista

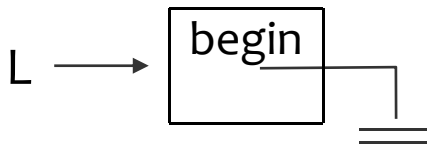
**Caso 1:** Lista está vazia



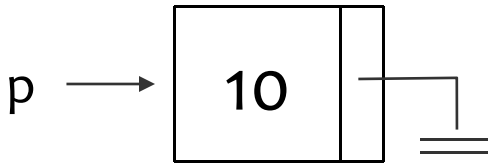

inserir **10**

# Inserção na cabeça (início) da lista

**Caso 1:** Lista está vazia



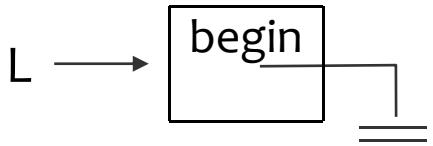

inserir **10**



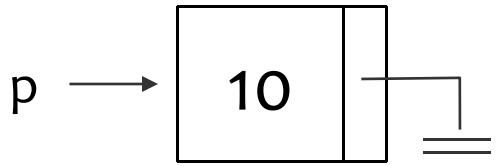

cria nó **p**

# Inserção na cabeça (início) da lista

**Caso 1:** Lista está vazia

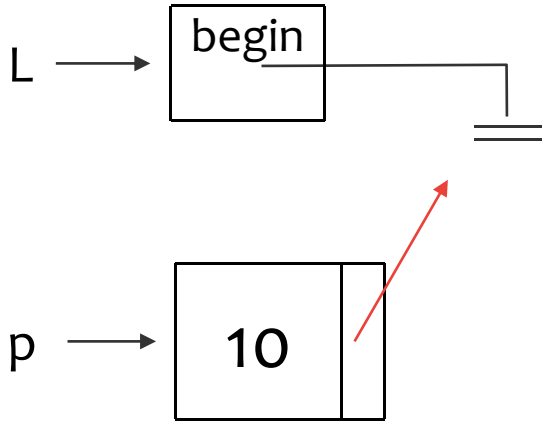



inserir **10**



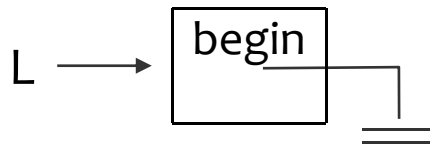

cria nó **p**

o **próximo** de **p** aponta para onde o **início** de L aponta

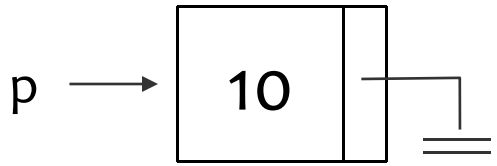


# Inserção na cabeça (início) da lista

**Caso 1:** Lista está vazia

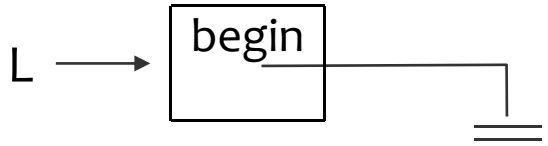


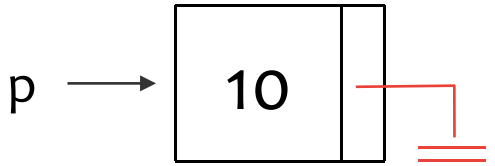

inserir **10**



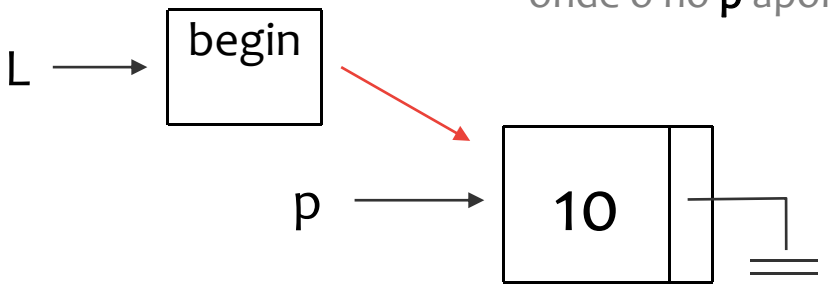

cria nó **p**

o **próximo** de **p** aponta para onde o **início** de L aponta



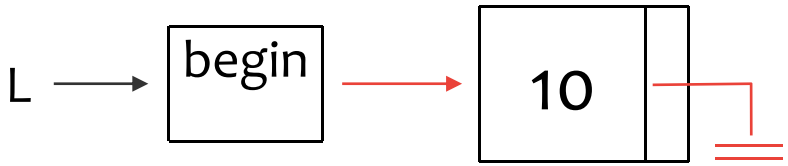




o **início** da lista L aponta para onde o nó **p** aponta



# Inserção na cabeça (início) da lista

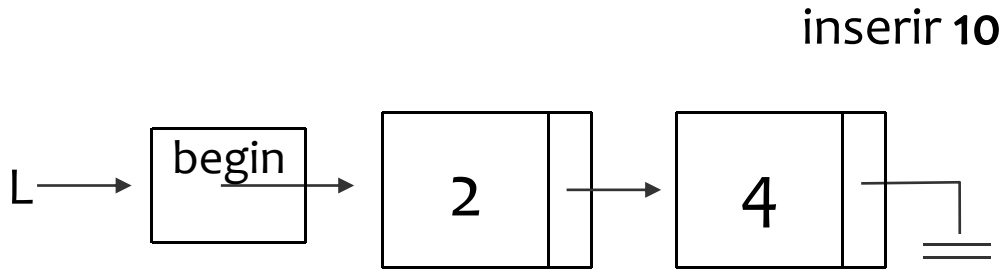
**Caso 1:** Lista está vazia




configuração final da lista
após a inserção

# Inserção na cabeça (início) da lista

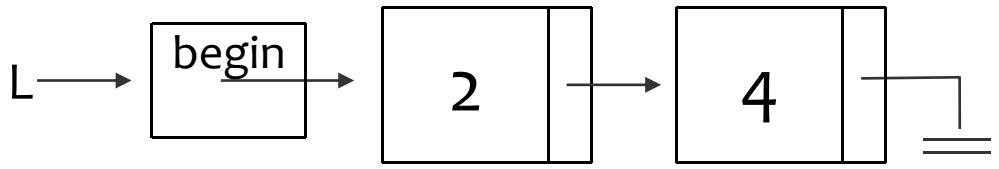
**Caso 2:** Lista possui elementos

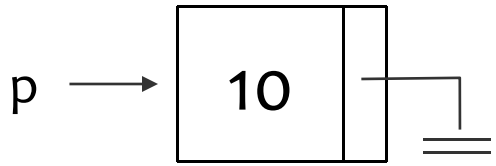


# Inserção na cabeça (início) da lista

**Caso 2:** Lista possui elementos

inserir **10**



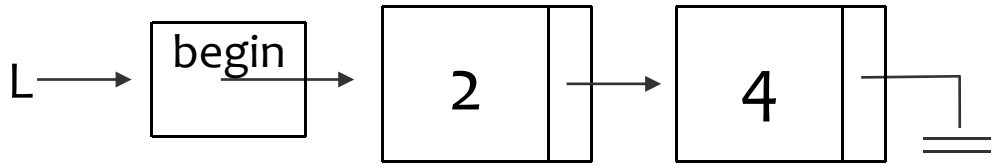



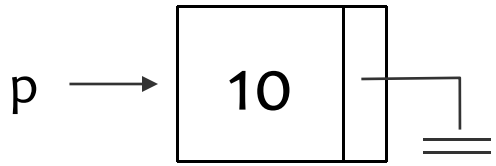

cria nó **p**

# Inserção na cabeça (início) da lista

**Caso 2:** Lista possui elementos

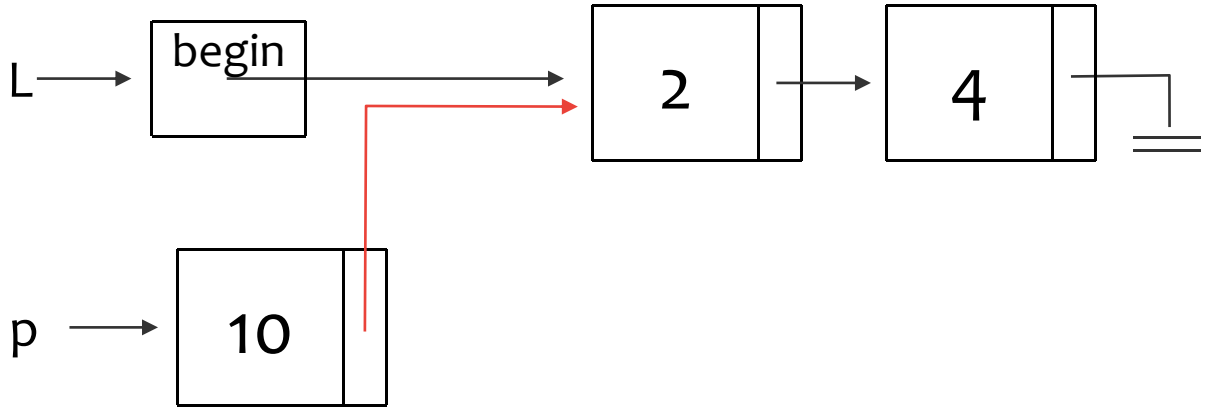
inserir **10**




cria nó **p**

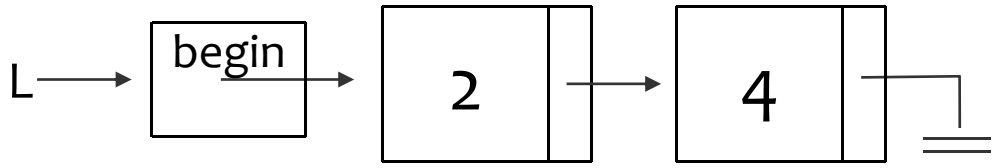
o **próximo** de **p** aponta para onde o **início** de L aponta

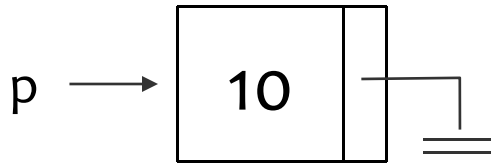


# Inserção na cabeça (início) da lista

**Caso 2:** Lista possui elementos

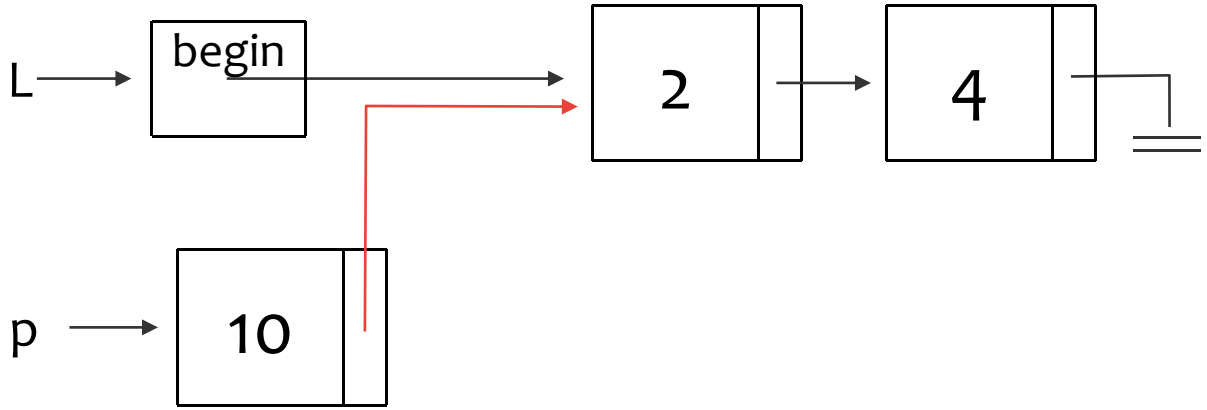
inserir **10**

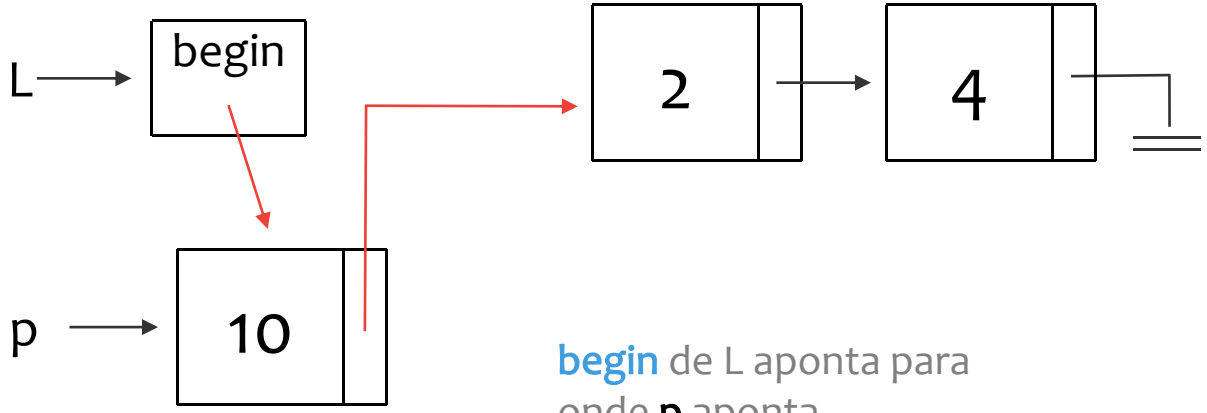



cria nó **p**

o **próximo** de **p** aponta para onde o **início** de L aponta

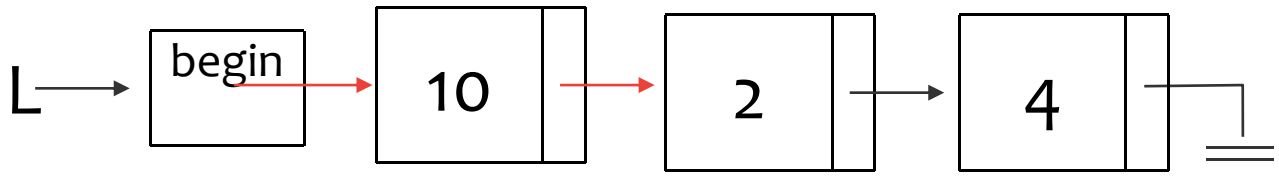



**begin** de L aponta para onde **p** aponta

# Inserção na cabeça (início) da lista

**Caso 2:** Lista possui elementos




configuração final da lista
após a inserção

# Lista Encadeada Simples

Diversos tipos de operações:

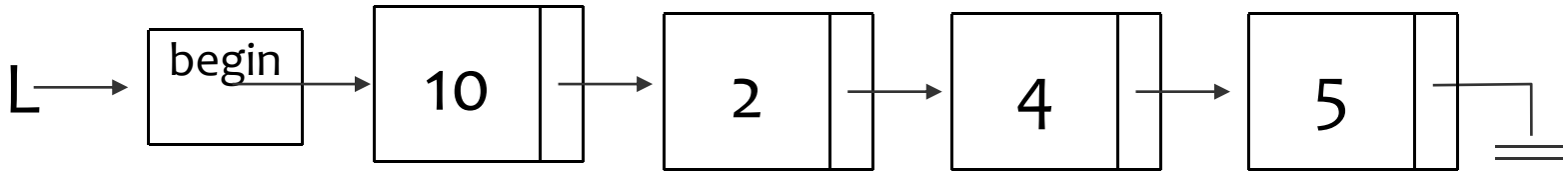
- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Lista Encadeada Simples

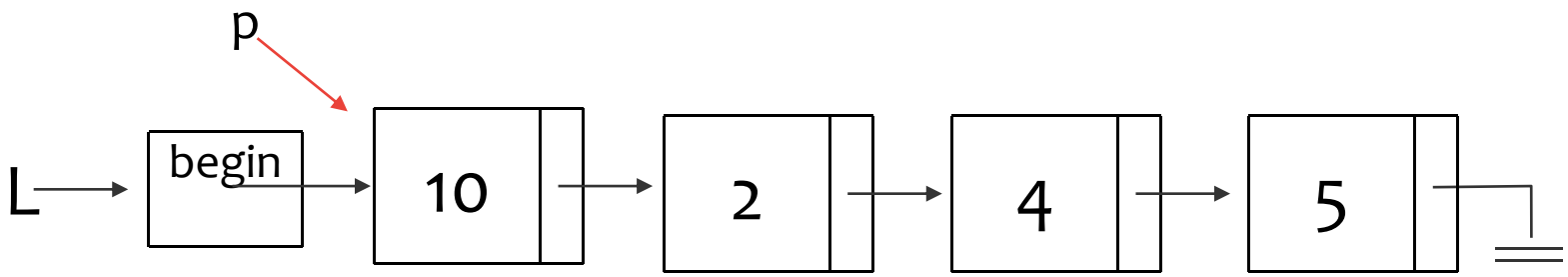
Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- **Impressão dos Elementos da Lista**
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Impressão dos Elementos da Lista

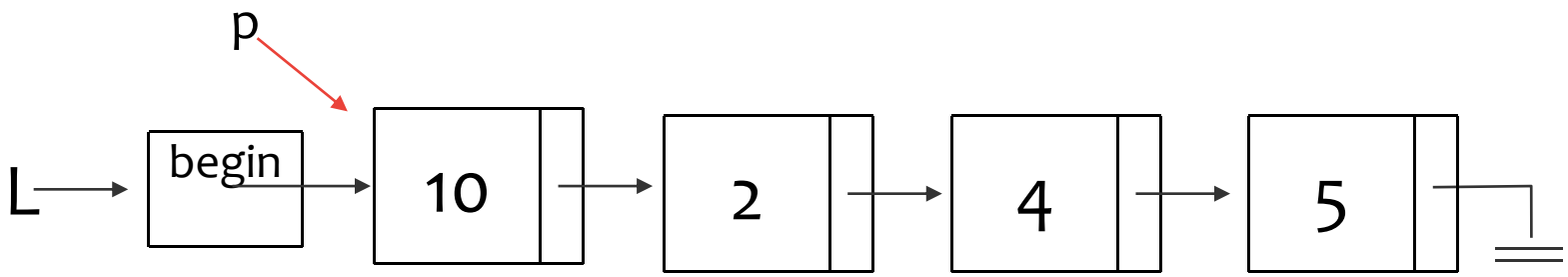


# Impressão dos Elementos da Lista



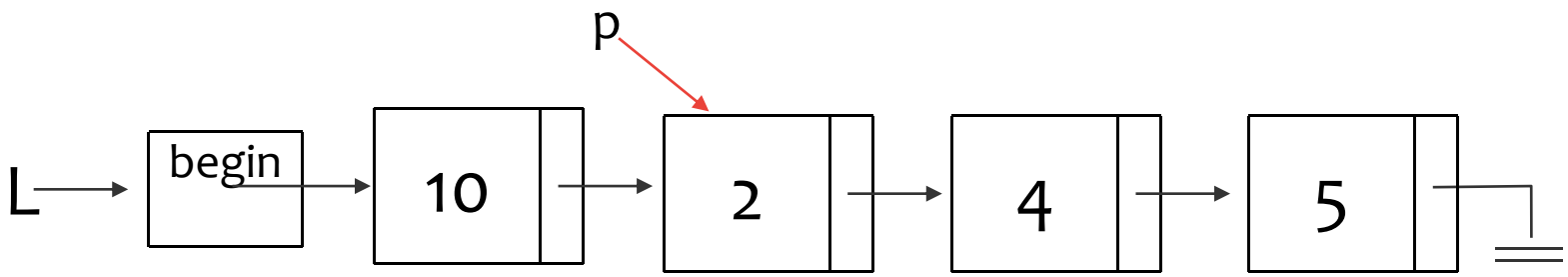

crie um ponteiro **p** e faça-o
apontar para o **início da lista**

# Impressão dos Elementos da Lista



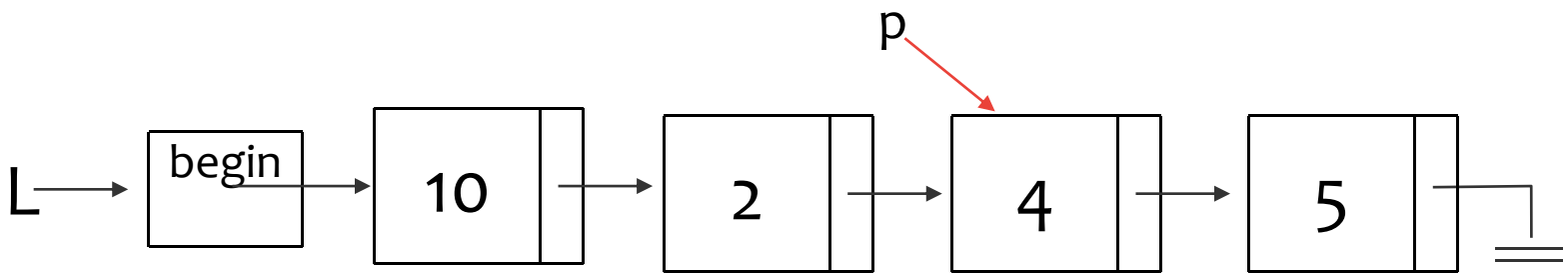

enquanto a lista **não chegou ao fim** (NULL):
imprima os dados apontado por **p** e
mova **p** para o **próximo elemento** da lista: **p = p->prox;**

# Impressão dos Elementos da Lista



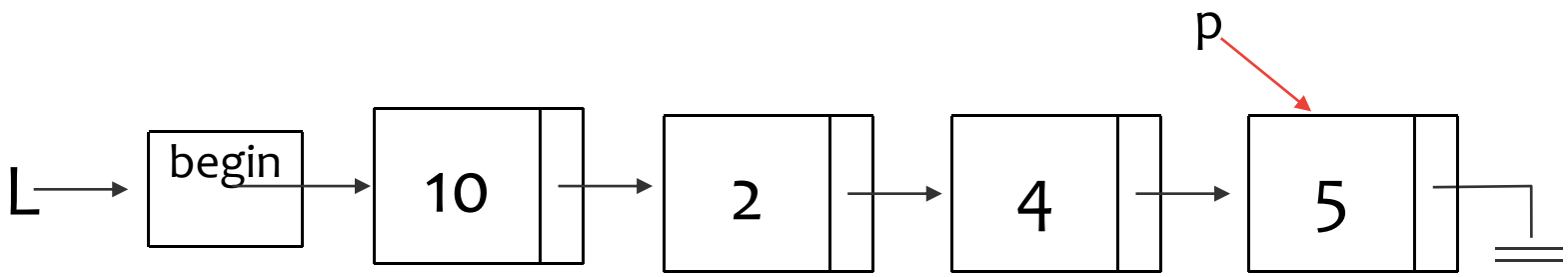

enquanto a lista **não chegou ao fim** (NULL):
imprima os dados apontado por **p** e
mova **p** para o **próximo elemento** da lista: **p = p->prox;**

# Impressão dos Elementos da Lista



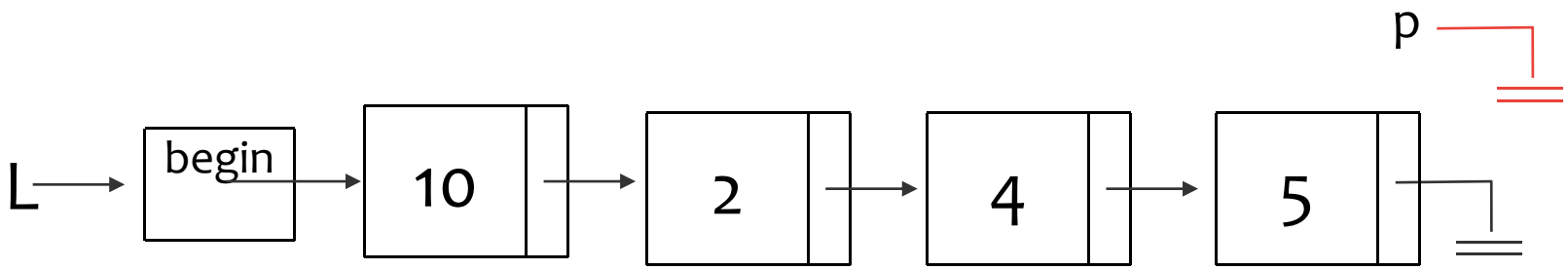

enquanto a lista **não chegou ao fim** (NULL):
imprima os dados apontado por **p** e
mova **p** para o **próximo elemento** da lista: **p = p->prox;**

# Impressão dos Elementos da Lista




enquanto a lista **não chegou ao fim** (NULL):
imprima os dados apontado por **p** e
mova **p** para o **próximo elemento** da lista: **p = p->prox;**

# Impressão dos Elementos da Lista




enquanto a lista **não chegou ao fim** (NULL):
imprima os dados apontado por **p** e
mova **p** para o **próximo elemento** da lista: **p = p->prox;**

# Lista Encadeada Simples

Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- **Impressão dos Elementos da Lista**
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

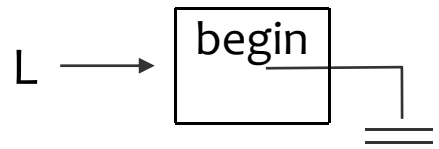
# Lista Encadeada Simples

Diversos tipos de operações:

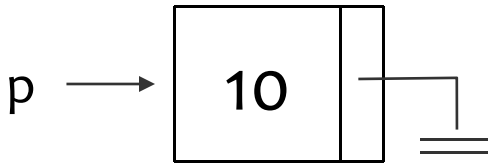
- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- **Inserção na cauda (fim) da lista;**
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Inserção na cauda (fim) da lista

**Caso 1:** Lista está vazia

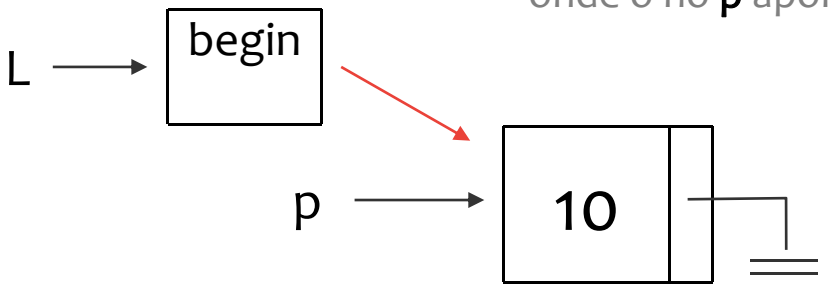



inserir **10**




cria nó **p**

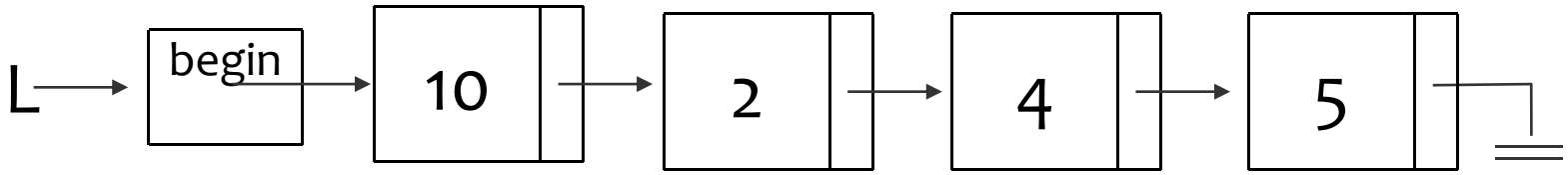
o **início** da lista L aponta para onde o nó **p** aponta



# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

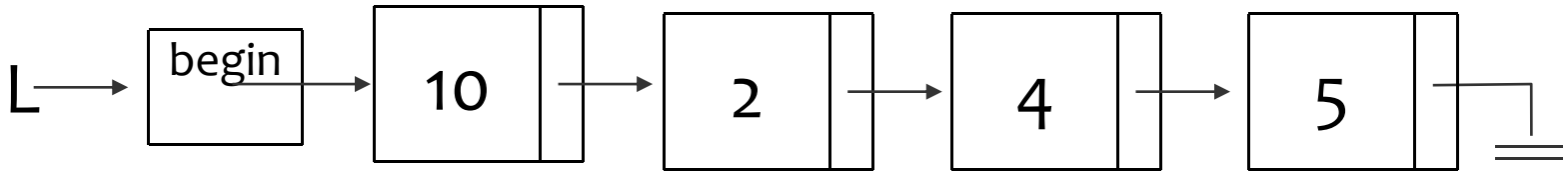
inserir **7**



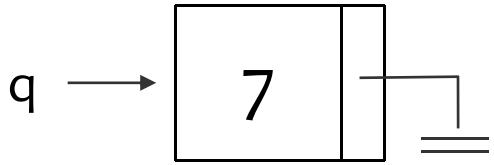
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

inserir **7**



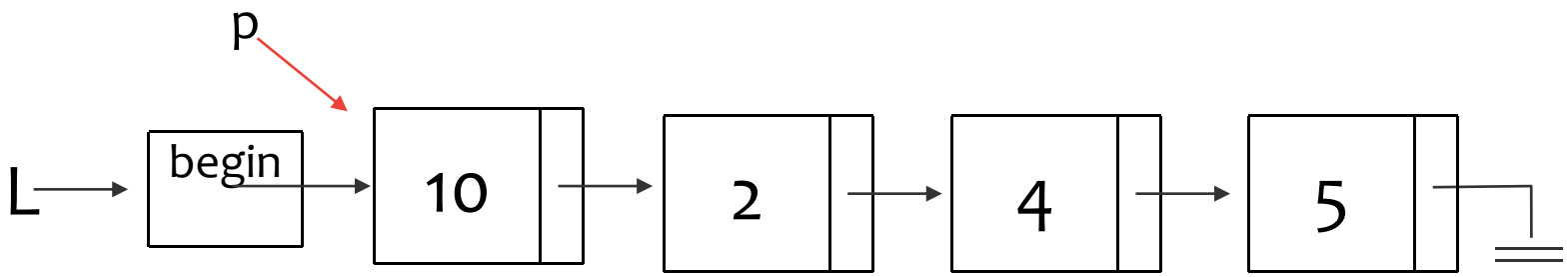

cria nó **q**



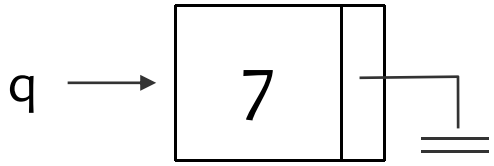
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

inserir **7**




cria nó **q**

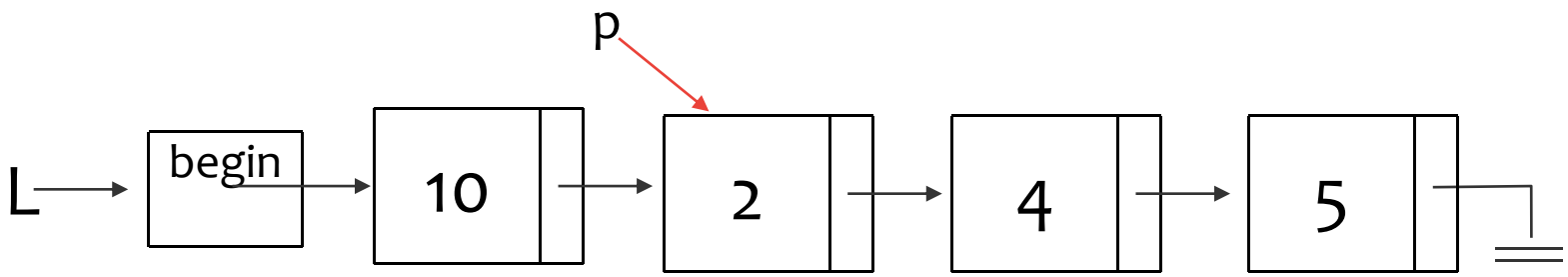



O ponrteiro **p** aponta, inicialmente, para o **nó inicial da lista**. Queremos posicionar **p** no **último** nós para então inserir o novo depois (p->next = q)

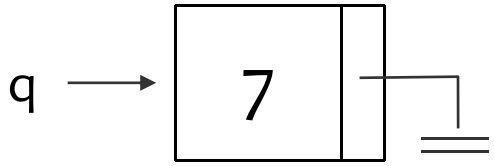
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

inserir **7**




cria nó **q**

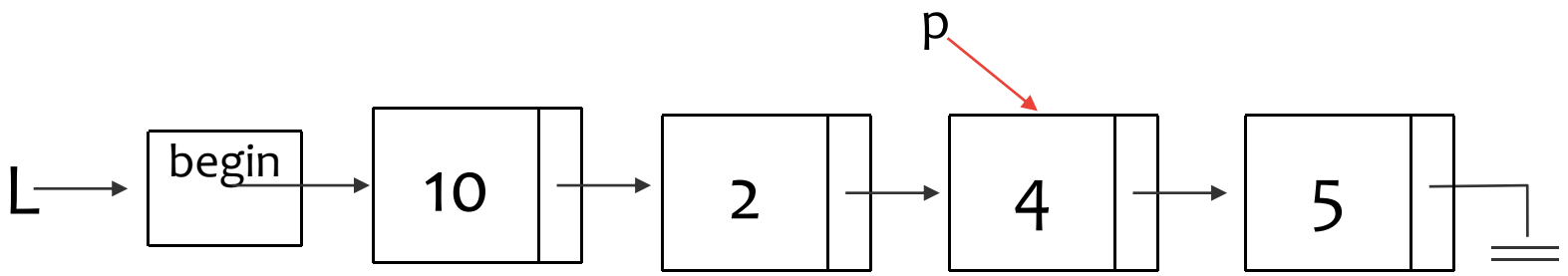



O ponrteiro **p** aponta, inicialmente, para o **nó inicial da lista**. Queremos posicionar **p** no **último** nós para então inserir o novo depois (p->next = q)

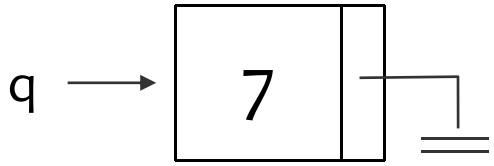
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

inserir **7**




cria nó **q**

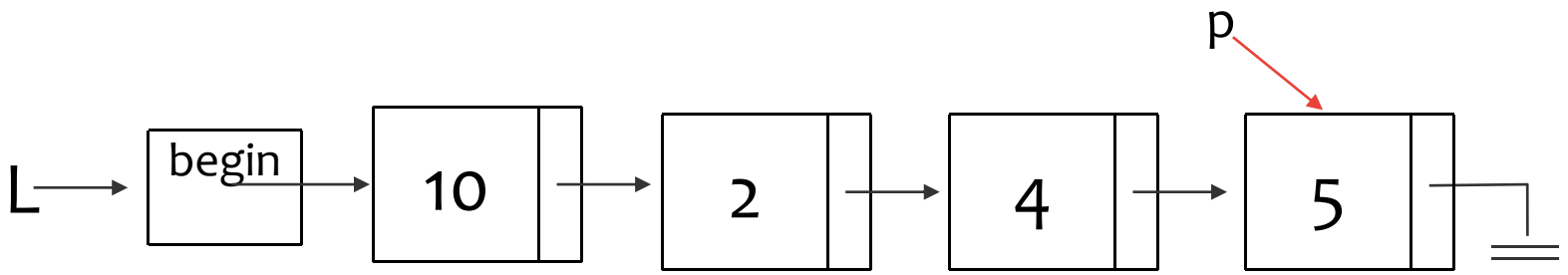



O ponrteiro **p** aponta, inicialmente, para o **nó inicial da lista**. Queremos posicionar **p** no **último** nós para então inserir o novo depois (p->next = q)

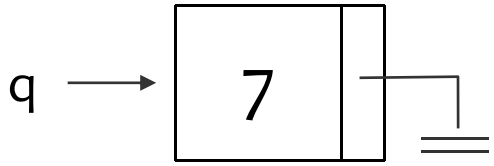
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

inserir **7**




cria nó **q**



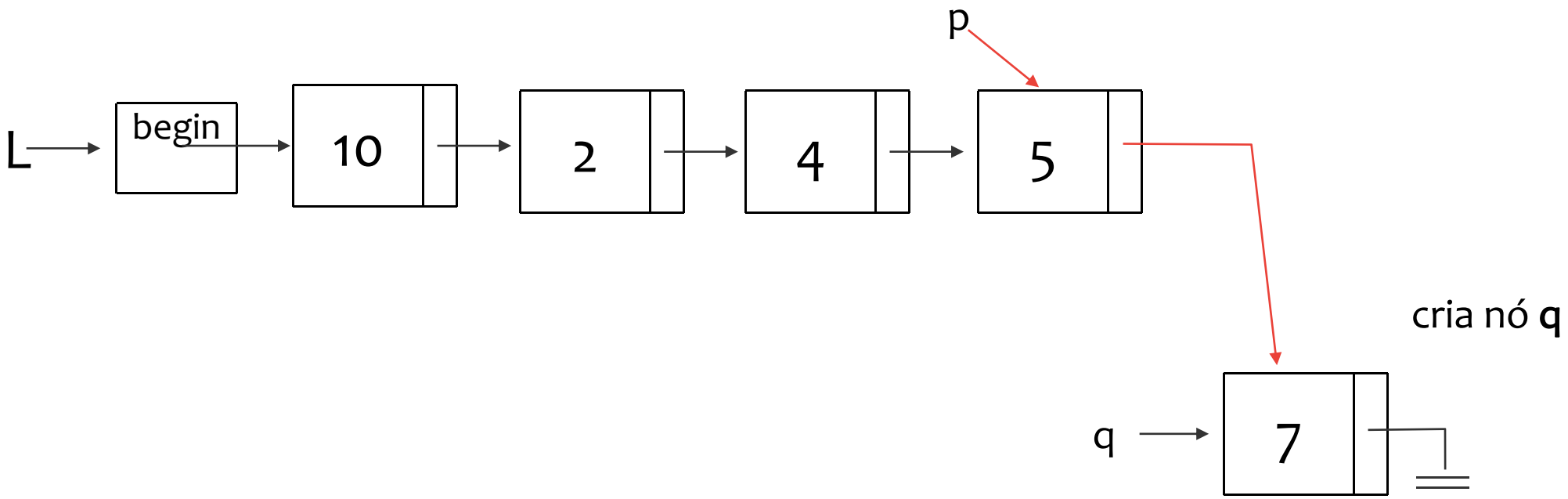

O ponrteiro **p** aponta, inicialmente, para o **nó inicial da lista**. Queremos posicionar **p** no **último** nós para então inserir o novo depois (p->next = q)

# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

o próximo nó de **p** é **NULL**,
chegamos no nó final

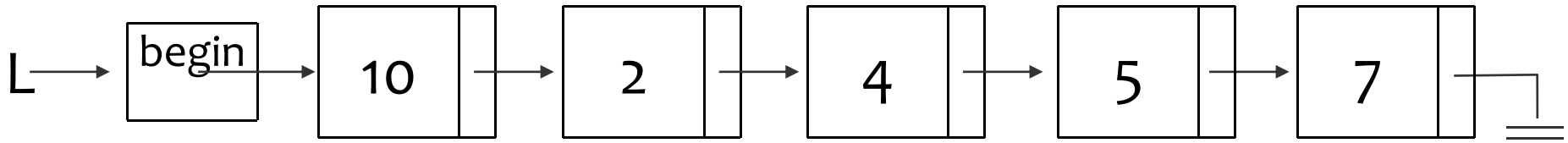
inserir **7**




o **próximo** de **p** aponta para **q**

# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos



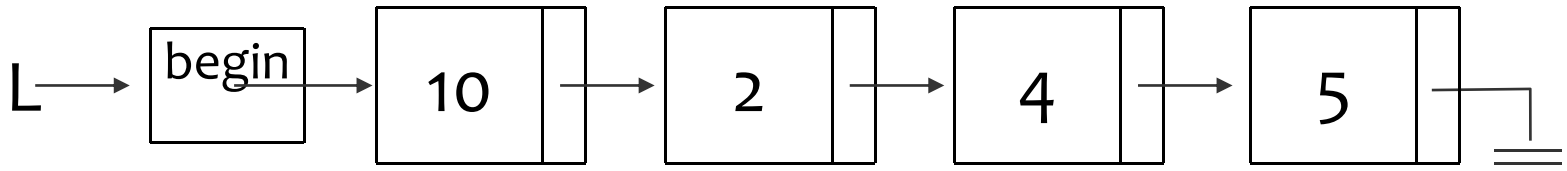

configuração final da lista
após a inserção

# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**



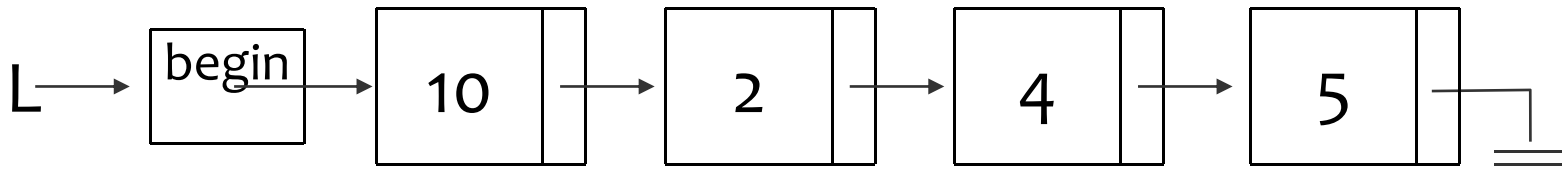
# Inserção na cauda (fim) da lista

**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**

## Solução?



# Inserção na cauda (fim) da lista

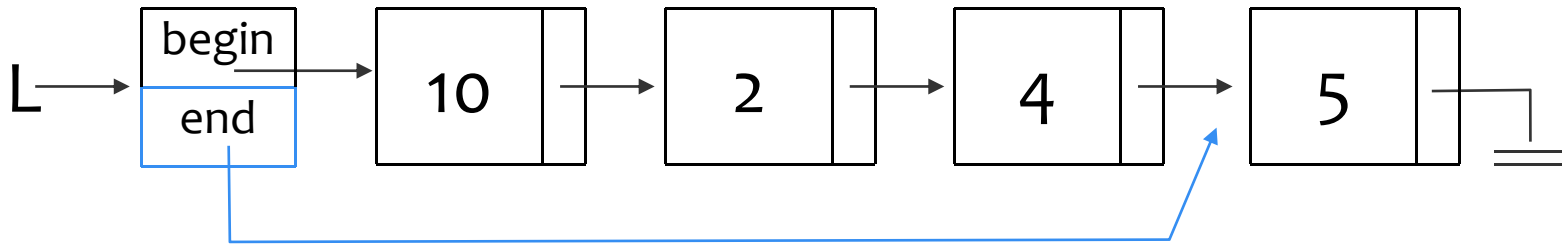
**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**

## Solução?

- Adicionamos, na struct de Lista, um ponteiro **end** para o **nó final da lista**;



# Inserção na cauda (fim) da lista

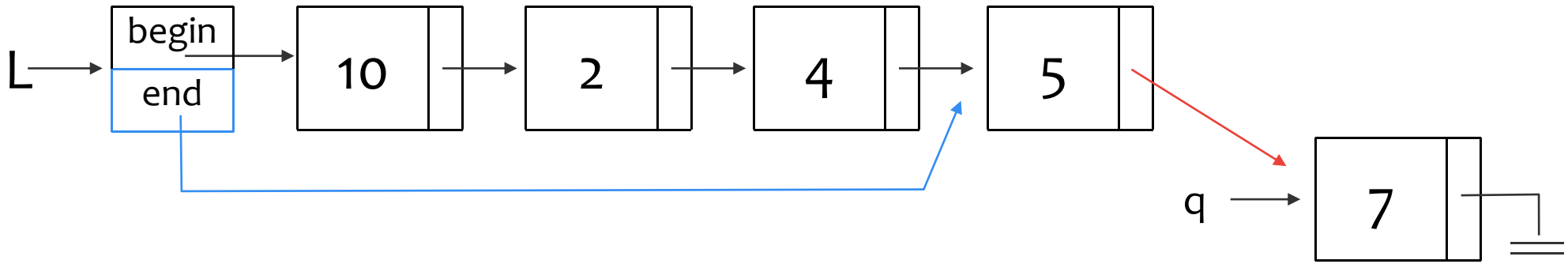
**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**

## Solução?

- Adicionamos, na struct de Lista, um ponteiro **end** para o **nó final da lista**;
- Assim, conseguimos adicionar o novo nó diretamente no final da lista



# Inserção na cauda (fim) da lista

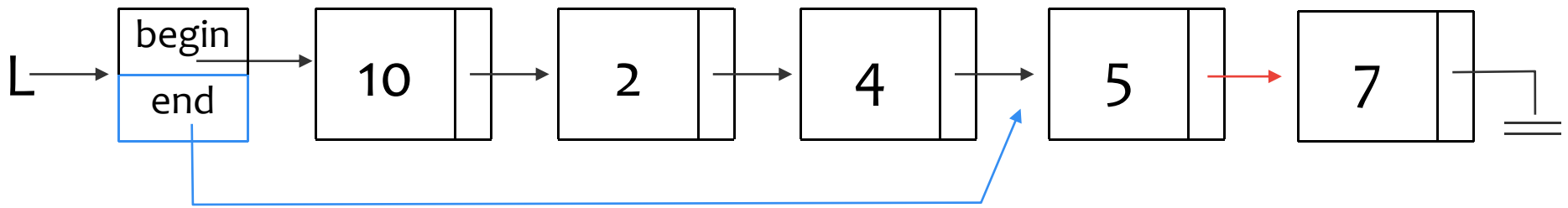
**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**

## Solução?

- Adicionamos, na struct de Lista, um ponteiro **end** para o **nó final da lista**;
- Assim, conseguimos adicionar o novo nó diretamente no final da lista



# Inserção na cauda (fim) da lista

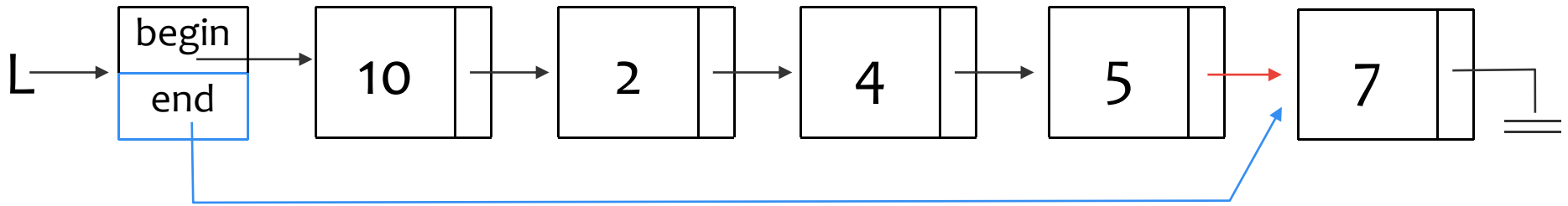
**Caso 2:** Lista possui elementos

**Problemas?**

- Precisamos percorrer todos os nós da lista até chegar no último nó —> **ineficiente**

## Solução?

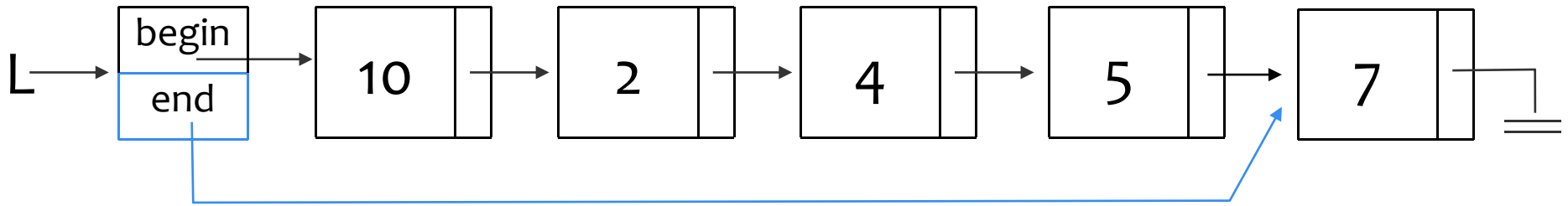
- Adicionamos, na struct de Lista, um ponteiro **end** para o **nó final da lista**;
- Assim, conseguimos adicionar o novo nó diretamente no final da lista
- Teremos que adaptar o **caso 1** da função de **inserção no início** da lista



# Inserção na cauda (fim) da lista

```c
typedef struct _node {
    int val;
    struct _node *next;
} Node;
```

```c
typedef struct _linked_list {
    Node *begin;
    Node *end;
} LinkedList;
```



# Lista Encadeada Simples

Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- **Inserção na cauda (fim) da lista;**
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Lista Encadeada Simples

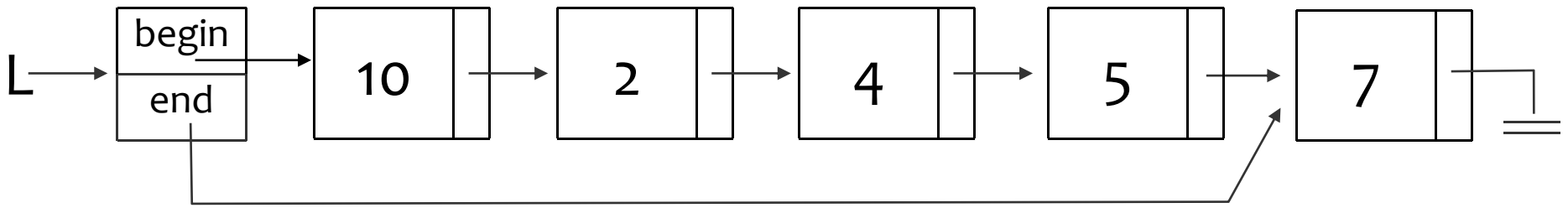
Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- Inserção na cauda (fim) da lista;
- **Remover elementos da lista;**
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i

# Remoção de Elementos da Lista

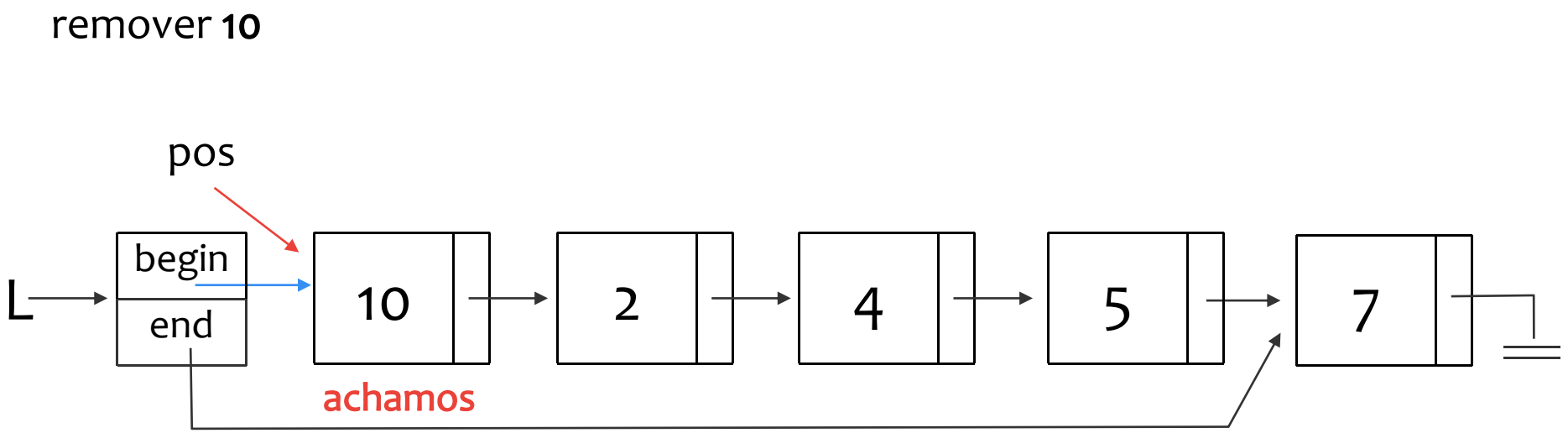
**Caso 1:** Elemento está na **cabeça (início)** da lista

remover **10**



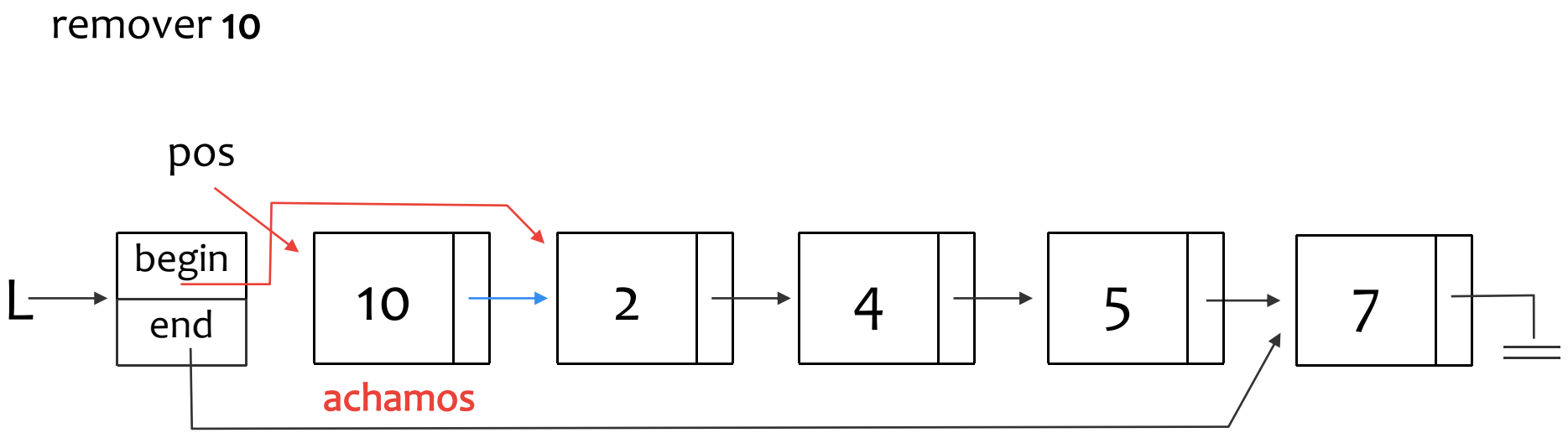
# Remoção de Elementos da Lista

**Caso 1:** Elemento está na **cabeça (início)** da lista



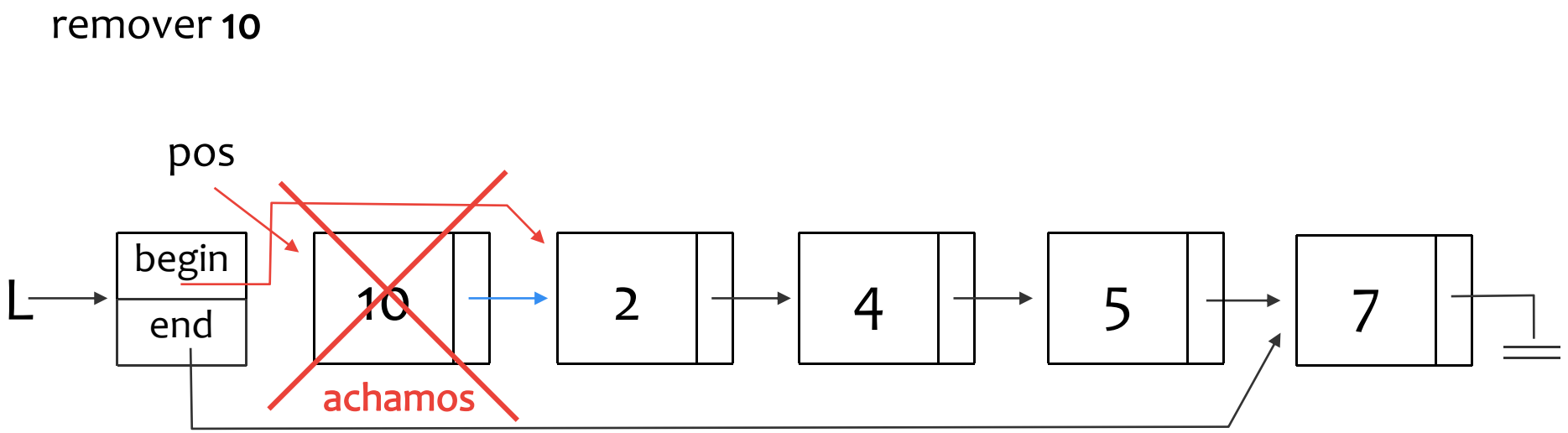
# Remoção de Elementos da Lista

**Caso 1:** Elemento está na **cabeça (início)** da lista



# Remoção de Elementos da Lista

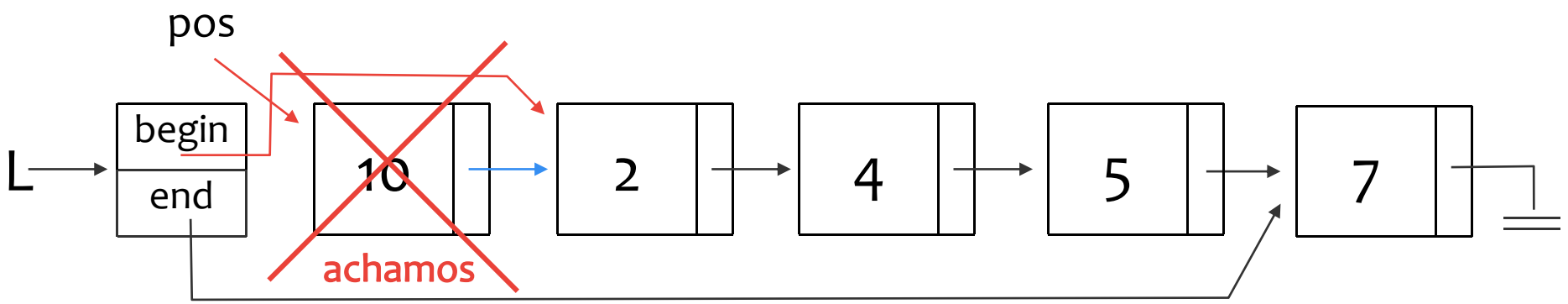
**Caso 1:** Elemento está na **cabeça (início)** da lista



# Remoção de Elementos da Lista

**Caso 1:** Elemento está na **cabeça (início)** da lista

remover **10**

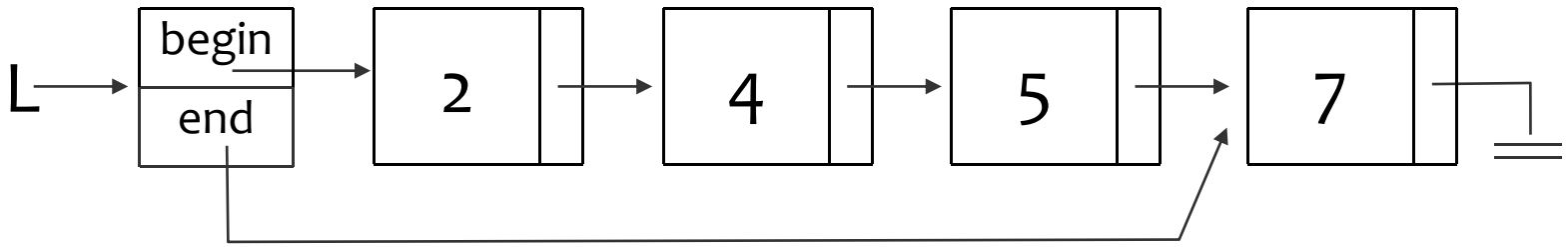



poderíamos continuar percorrendo
a lista para remover outros
possíveis nós **10**

# Remoção de Elementos da Lista

**Caso 1:** Elemento está na **cabeça (início)** da lista

remover **10**

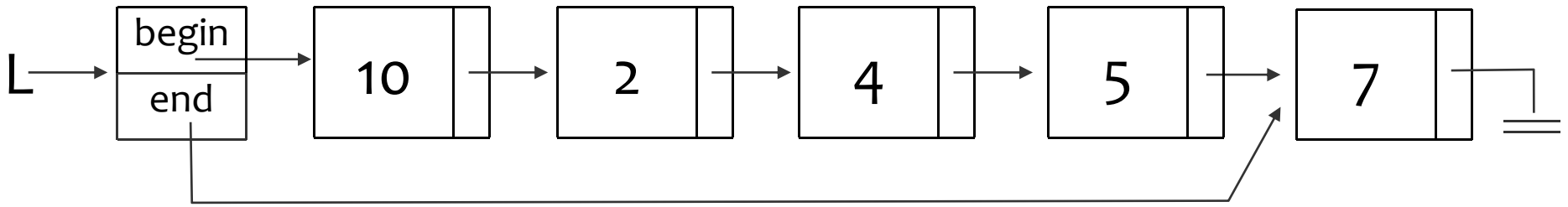



configuração final da lista
após a inserção

# Remoção de Elementos da Lista

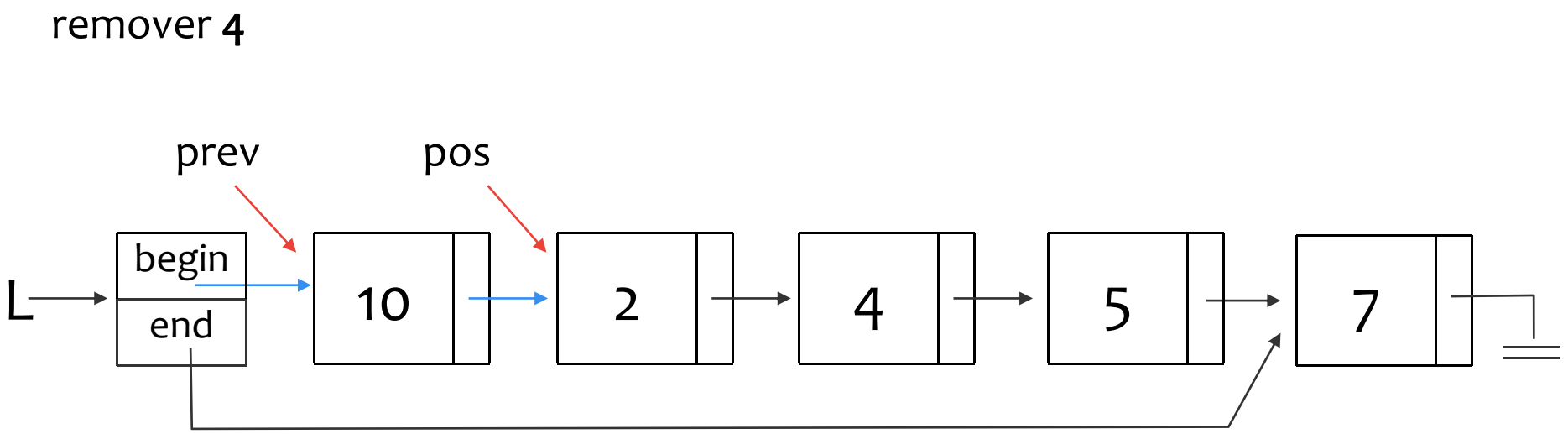
**Caso 2:** Elemento está no **meio** da lista

remover **4**



# Remoção de Elementos da Lista

**Caso 2:** Elemento está no **meio** da lista



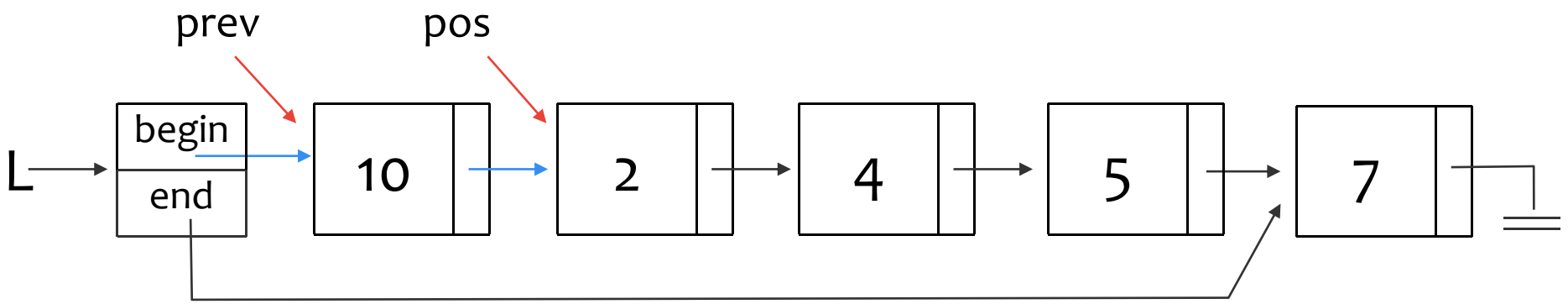

```
prev = L->inicio;
pos = L->inicio->prox;
```

além do ponteiro **pos** que checa o **elemento atual**, precisaremos de um outro ponteiro **prev** apontando para o **nó anterior**

# Remoção de Elementos da Lista

**Caso 2:** Elemento está no **meio** da lista

remover **4**



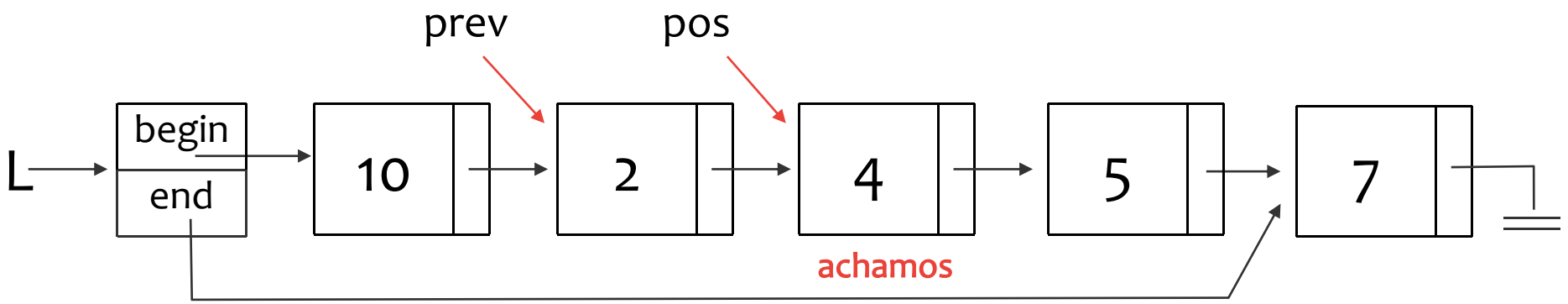

enquanto **não achou** o elemento e a lista **não chegou ao fim** (pos == NULL),
mova os dois ponteiros para os próximos nós

```
prev = L->inicio;
pos = L->inicio->prox;
```

# Remoção de Elementos da Lista

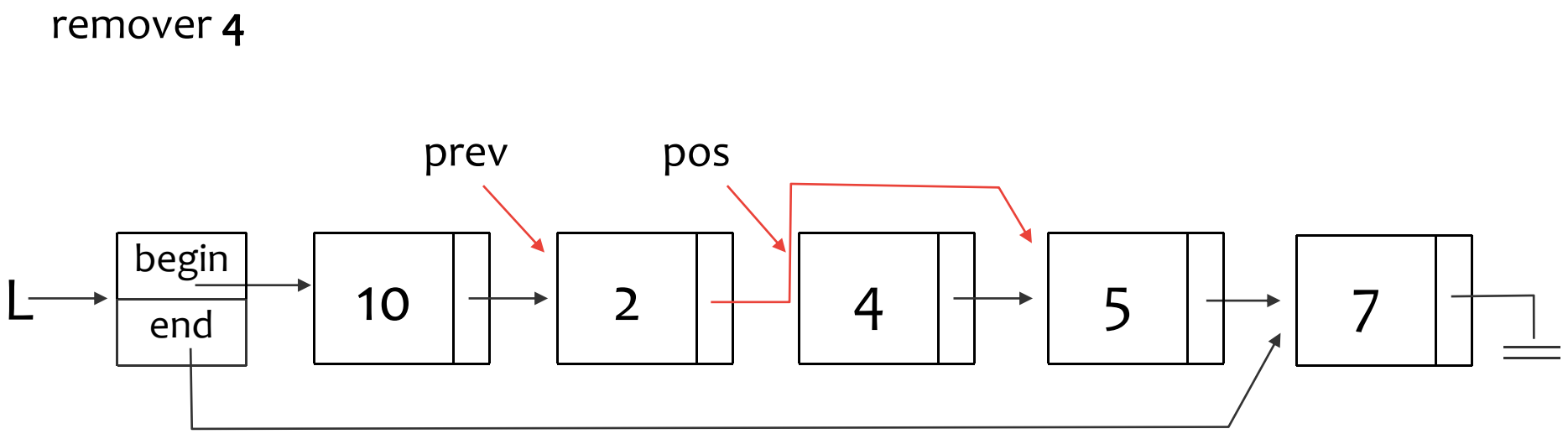
**Caso 2:** Elemento está no **meio** da lista

remover **4**



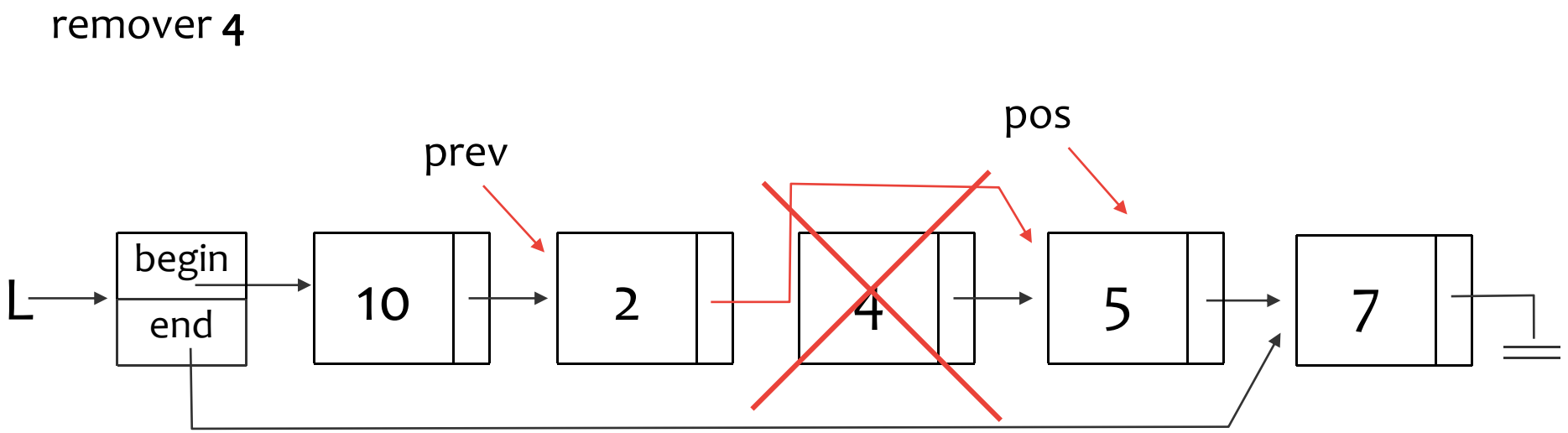
# Remoção de Elementos da Lista

**Caso 2:** Elemento está no **meio** da lista



# Remoção de Elementos da Lista

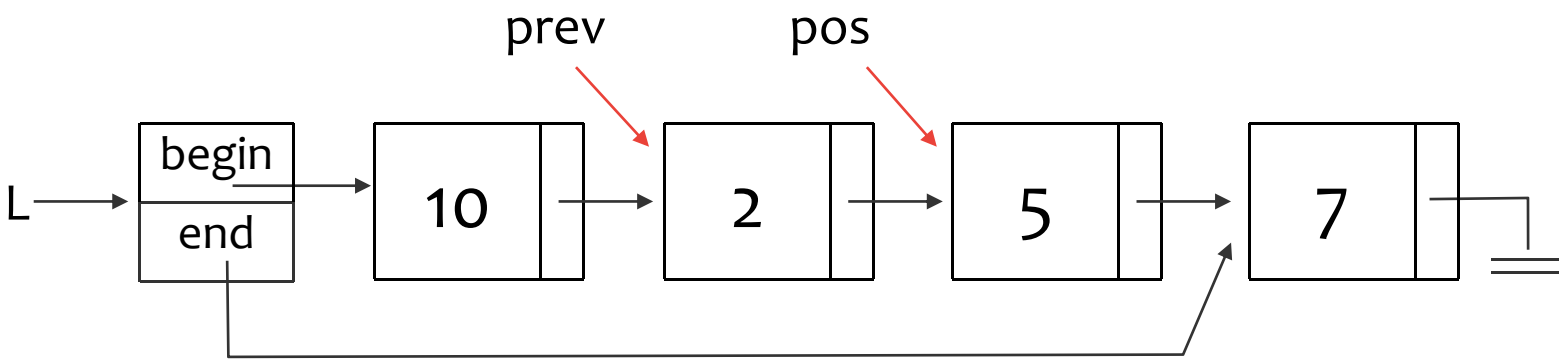
**Caso 2:** Elemento está no **meio** da lista



# Remoção de Elementos da Lista

**Caso 2:** Elemento está no **meio** da lista

remover **4**

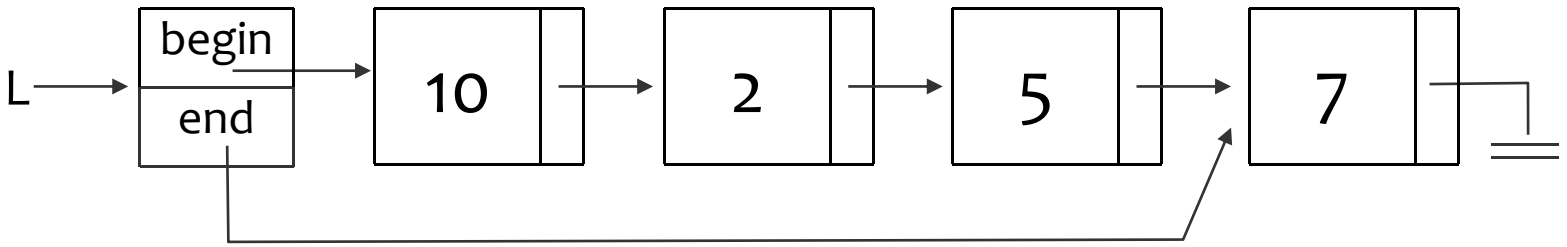



poderíamos continuar percorrendo
a lista para remover outros
possíveis nós **4**

# Remoção de Elementos da Lista

**Caso 2:** Elemento está no **meio** da lista

remover **4**

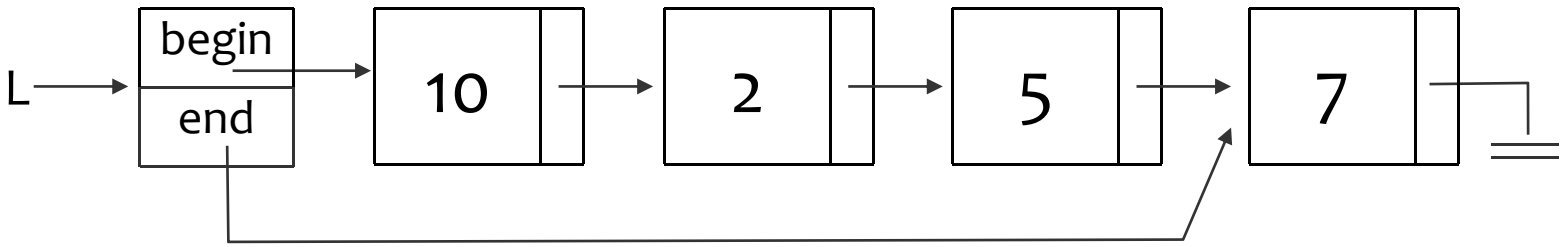



configuração final da lista
após a inserção

# Remoção de Elementos da Lista

**Caso 3:** Elemento está na **cauda (final)** da lista

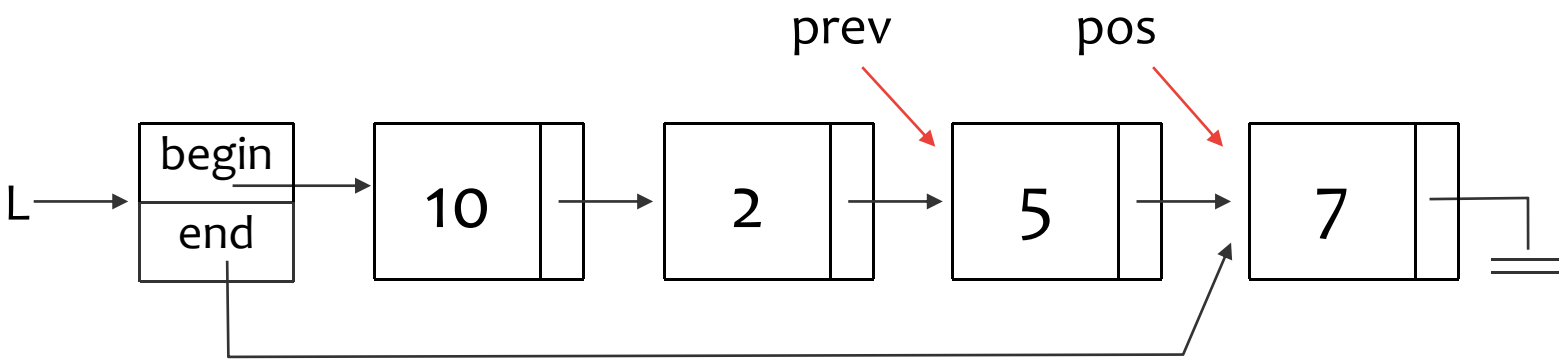
remover **7**



# Remoção de Elementos da Lista

**Caso 3:** Elemento está na **cauda (final)** da lista

remover **7**

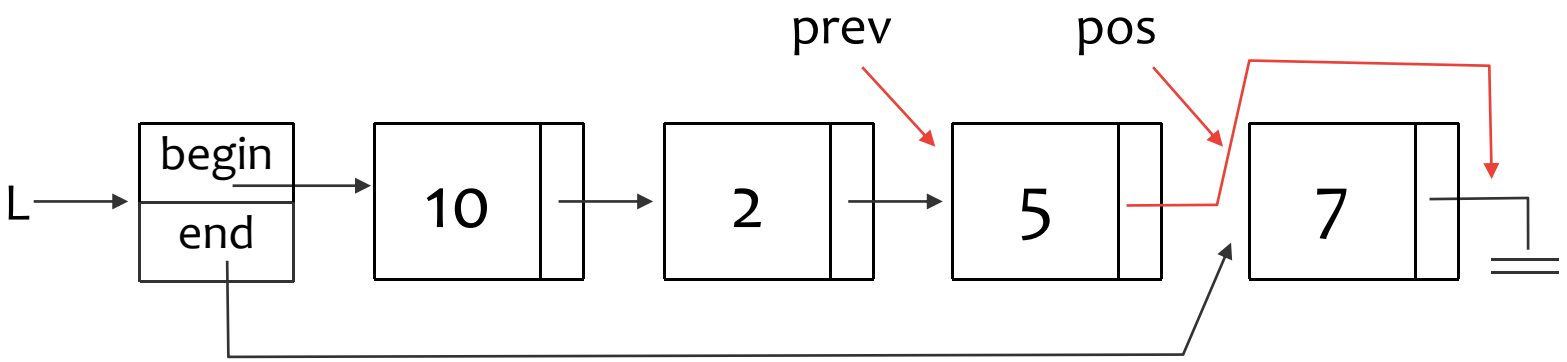



Suponha que já percorremos a lista inteira e chegamos no nó final

# Remoção de Elementos da Lista

**Caso 3:** Elemento está na **cauda (final)** da lista

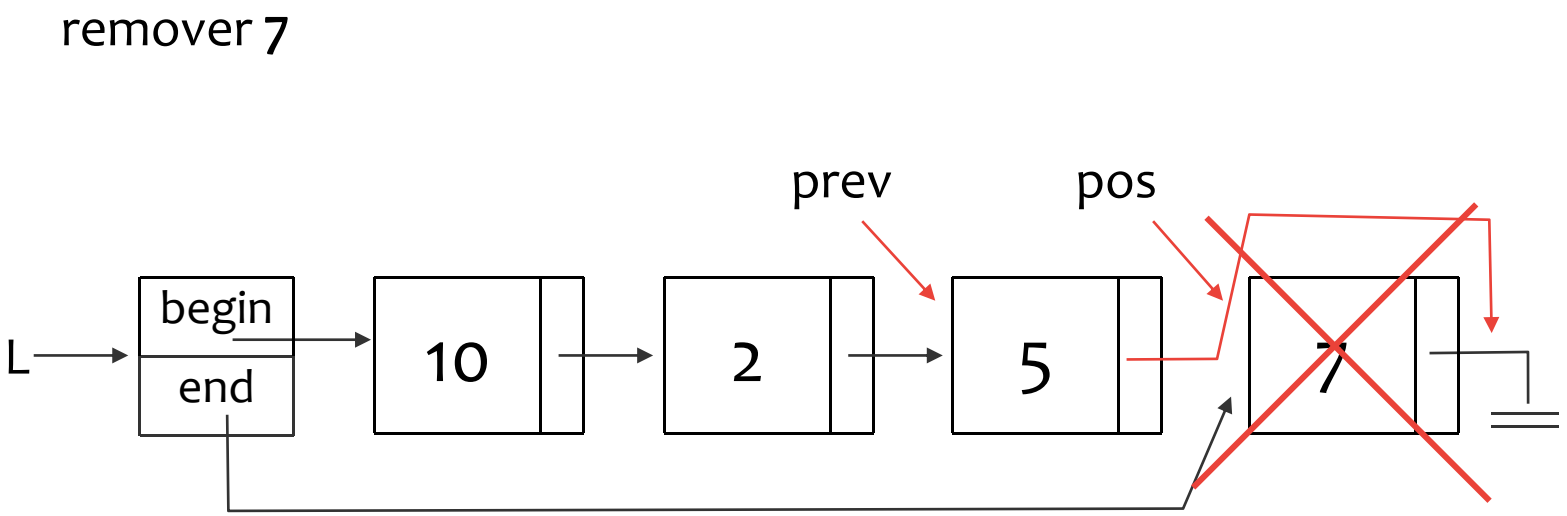
remover **7**




Suponha que já percorremos a lista inteira e chegamos no nó final

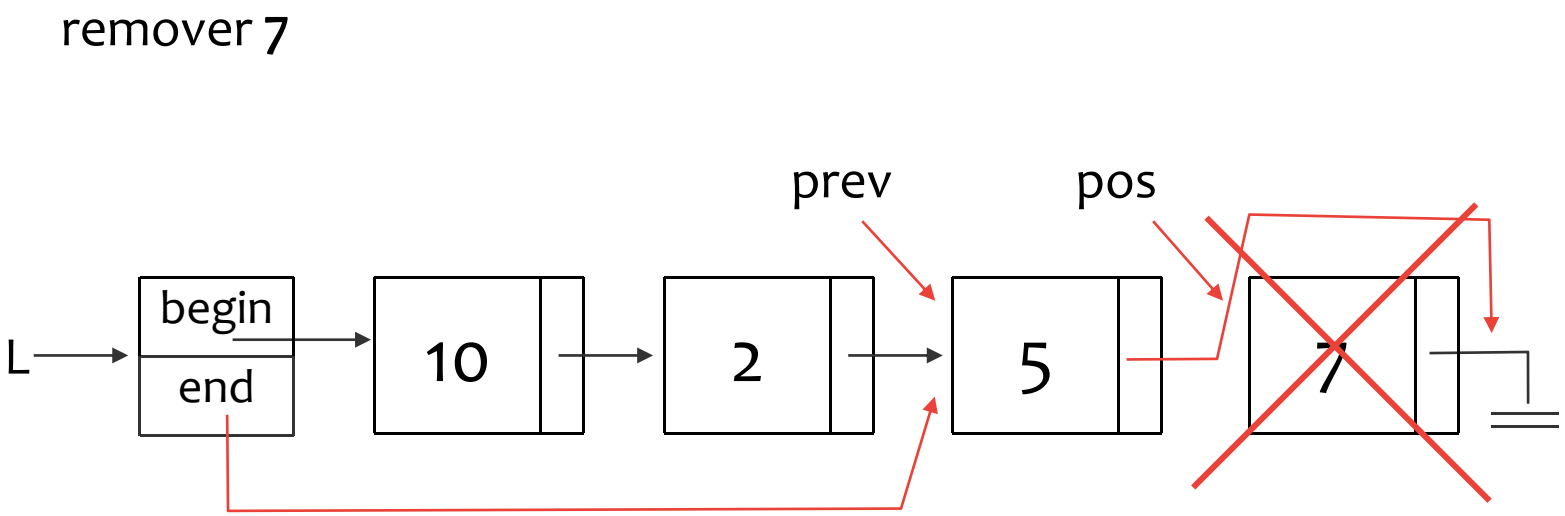
# Remoção de Elementos da Lista

**Caso 3:** Elemento está na **cauda (final)** da lista



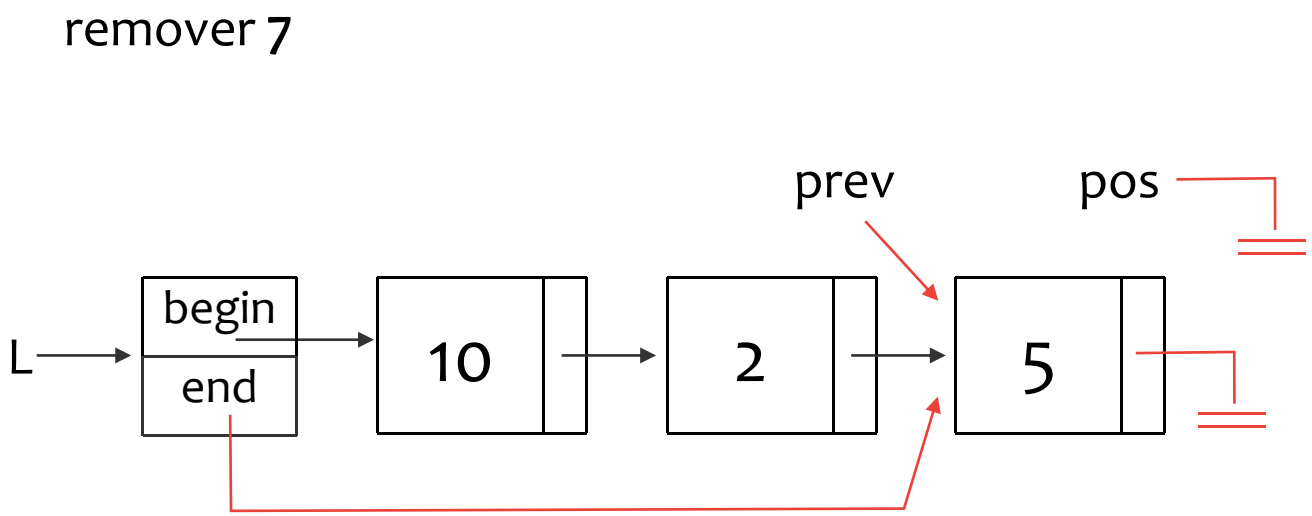
# Remoção de Elementos da Lista

**Caso 3:** Elemento está na **cauda (final)** da lista



# Remoção de Elementos da Lista

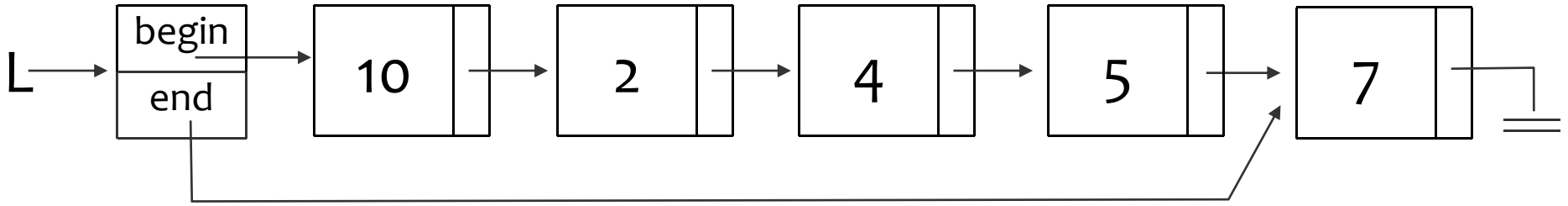
**Caso 3:** Elemento está na **cauda (final)** da lista



# Lista Encadeada Simples

Diversos tipos de operações:

- Inserção na cabeça (início) da lista;
- Impressão dos Elementos da Lista
- Inserção na cauda (fim) da lista;
- Remover elementos da lista;
- Contar o número de elementos da Lista;
- Verificar se a lista está vazia e retornar verdadeiro/falso
- Retornar o primeiro elemento;
- Retornar o último elemento;
- Retornar um elemento na posição i


L
begin
end
10
2
4
5
7

# Exercícios

- Contar o número de elementos da Lista;
- Inserir um elemento em uma dada posição
- Inverter uma lista
- Copiar/clonar uma lista;
- Apagar todos os elementos da lista
- Concatenar de duas listas;
- Ordenação de uma lista;
- Buscar um dado elemento na lista e retornar seu ponteiro;
- Inserção ordenada;

# Array x Lista Encadeada

## Arrays

### Vantagens

- Acesso direto (v[i]): **rápido**
- Dados **contíguos** na memória
- Percorrer os elementos do array é **rápido**

### Desvantagens

- Tamanho do array é fixado (não é flexível)

## Listas Encadeadas

### Vantagens

- Tamanho é flexível;

### Desvantagens

- Acesso de um elemento é **sequencial**;
  - É necessário comparar elemento por elemento;
- Dados não contíguos na memória;
- Percorrer a lista inteira é **lento**;

# Dominando Estruturas de Dados 1

## Listas Encadeadas Simples

Prof. dr. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding