

Algoritmos e Estruturas de Dados I

Aula 06: Listas Lineares - Parte 1

Prof. Márcio Porto Basgalupp

créditos: Prof. Jurandy G. Almeida Jr.

Universidade Federal de São Paulo
Departamento de Ciência e Tecnologia

O que é uma lista linear?

- Uma **lista linear** é uma estrutura de dados na qual elementos de um mesmo tipo de dado estão **logicamente** organizados de maneira **sequencial**



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ou retirados ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ou retirados ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ou retirados ...



O que é uma lista linear?

- Por exemplo, pense numa lista de compras de supermercado
- Os produtos a serem comprados podem ser inseridos ou retirados em qualquer posição



O que é uma lista linear?

- O primeiro produto é o primeiro não riscado
- O segundo produto é o segundo não riscado
- ...
- O último produto é o último não riscado



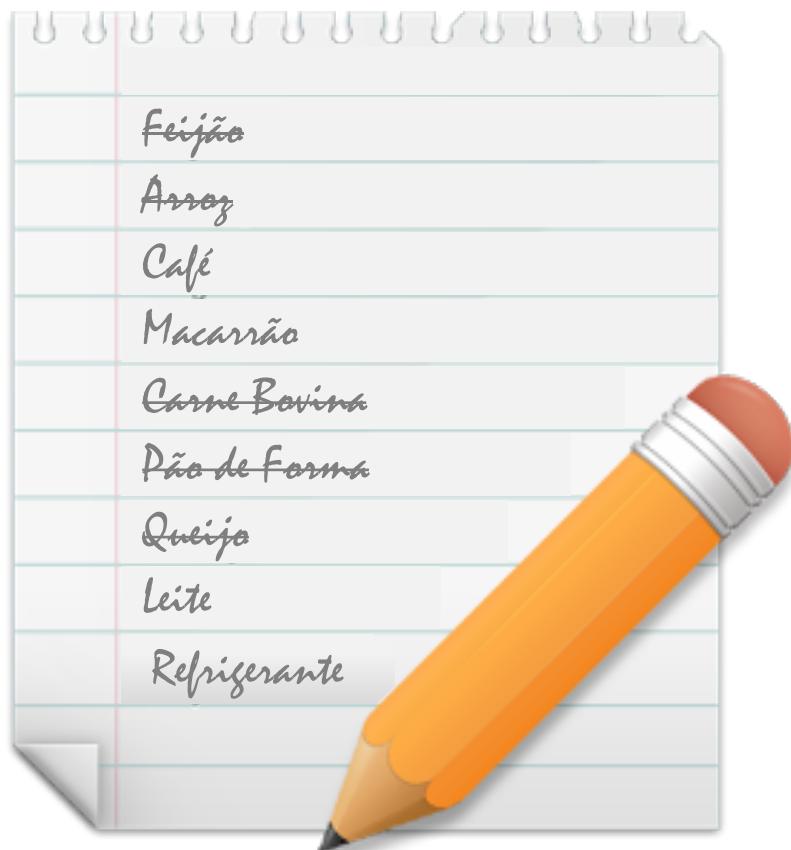
Operações Básicas

- Quando um item é adicionado numa lista linear, usa-se a operação **Inserir**



Operações Básicas

- Quando um item é adicionado numa lista linear, usa-se a operação **Inserir**
- Quando um item é retirado de uma lista linear, usa-se a operação **Retirar**



- Uma das formas mais simples de interligar os elementos de um conjunto
- Estrutura em que as operações **inserir**, **retirar** e **localizar** são definidas
- Podem **crescer** ou **diminuir** de tamanho durante a execução de um programa, de acordo com a demanda
- Itens podem ser **acessados**, **inseridos** ou **retirados** de uma lista

- **Outras operações:**
 - Concatenação de duas ou mais listas
 - Partição de lista em duas ou mais listas
- **Principais vantagens:**
 - Manipulação de quantidade imprevisível de dados
 - Manipulação de dados de formato também imprevisível
- **Exemplos de aplicação:**
 - manipulação simbólica, gerência de memória, simulações e compiladores

Definição

- **Sequência de zero ou mais itens**
 - x_1, x_2, \dots, x_n , na qual x_i é de um determinado tipo e n representa o tamanho da lista linear
- **Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão**
 - Assumindo $n \geq 1$, x_1 é o primeiro item da lista e x_n é o último item da lista
 - x_i precede x_{i+1} para $i = 1, 2, \dots, n - 1$
 - x_i sucede x_{i-1} para $i = 2, 3, \dots, n$
 - o elemento x_i é dito estar na i -ésima posição da lista

- **O que o TAD Lista deveria conter?**
 - Representação do tipo da lista
 - Conjunto de operações que atuam sobre a lista

- **Quais operações deveriam fazer parte da TAD?**

- **O que o TAD Lista deveria conter?**
 - Representação do tipo da lista
 - Conjunto de operações que atuam sobre a lista

- **Quais operações deveriam fazer parte da TAD?**

**O conjunto de operações a ser definido
depende de cada aplicação**

- Um conjunto de operações necessário a uma maioria de aplicações é:
 1. Criar uma lista linear vazia
 2. Inserir um novo item imediatamente após o i -ésimo item
 3. Retirar o i -ésimo item
 4. Localizar o i -ésimo item para examinar e/ou alterar o conteúdo de seus componentes

5. Combinar duas ou mais listas lineares em uma lista única
6. Dividir uma lista linear em duas ou mais listas
7. Fazer uma cópia da lista linear
8. Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes
9. Pesquisar a ocorrência de um item com um valor particular em algum componente

Exemplo de Protótipo

■ Exemplo de Conjunto de Operações

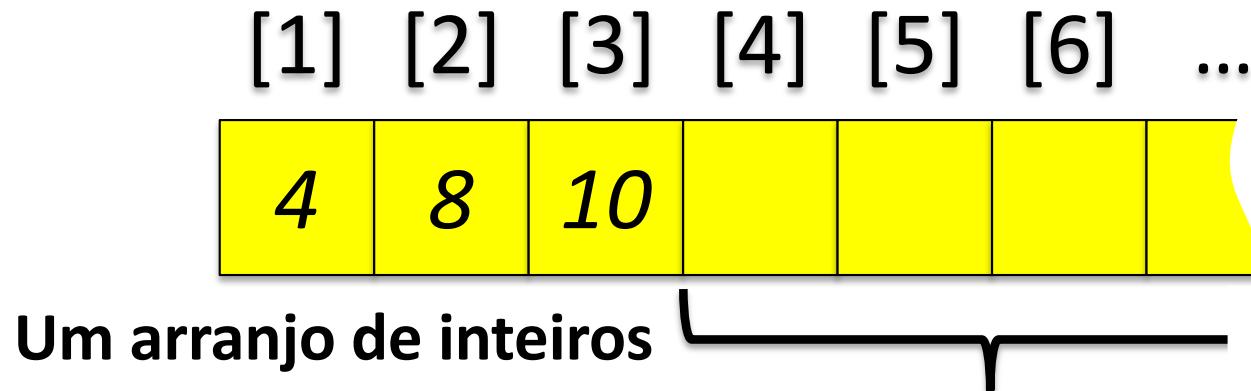
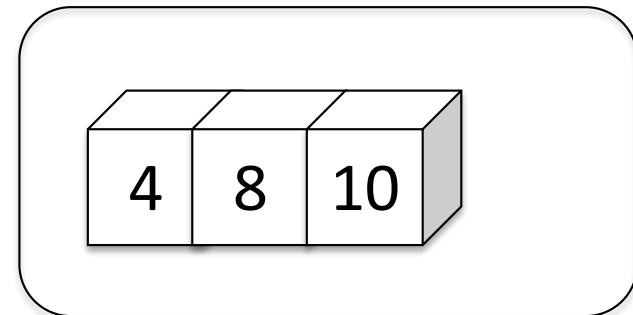
- **TLista_Inicia(Lista)**: Inicia uma lista vazia
- **TLista_EhVazia(Lista)**: Retorna *true* se a lista estiver vazia; caso contrário, retorna *false*
- **TLista_Insere(Lista, p, x)**: Insere o item *x* na posição *p* da lista, deslocando os itens a partir da posição *p* para as posições posteriores
- **TLista_Retira(Lista, p, x)**: Retorna o item *x* que está na posição *p* da lista, retirando-o da lista e deslocando os itens a partir da posição *p+1* para as posições anteriores
- **TLista_Tamanho(Lista)**: Retorna o número de itens da lista

- Existem várias opções de estruturas de dados que podem ser usadas para representar listas lineares
- As duas representações mais utilizadas são:
 - Implementação por meio de **arranjos**
 - Implementação por meio de **apontadores**

IMPLEMENTAÇÃO POR ARRANJOS

Implementação por Arranjos

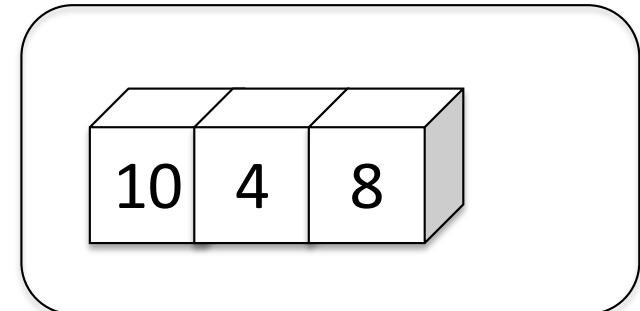
- Os itens da lista linear serão inicialmente armazenados no **início de um arranjo**, como mostra este exemplo



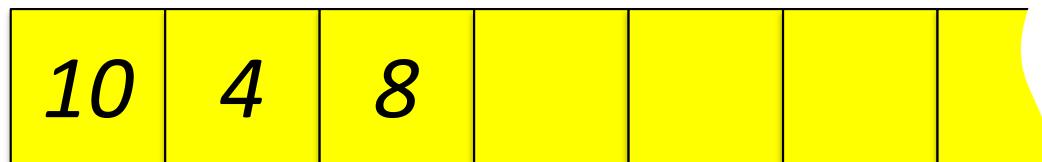
Não nos interessamos para o que está armazenado nesta parte do arranjo

Implementação por Arranjos

- Os itens possuem uma ordem.
A figura abaixo **não** representa a mesma lista que a anterior ...



[1] [2] [3] [4] [5] [6] ...

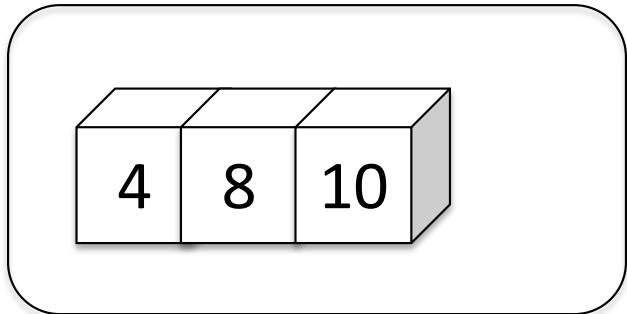


Um arranjo de inteiros

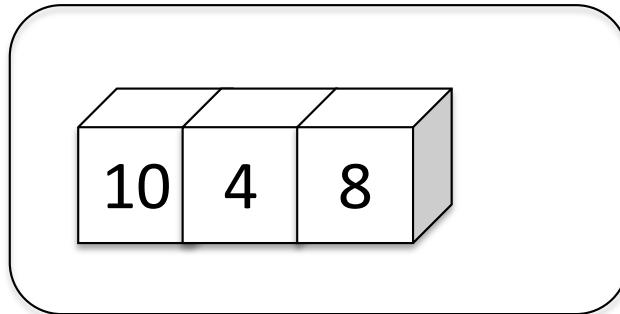
Não nos interessamos para o que está armazenado nesta parte do arranjo

Implementação por Arranjos

- Assim, ...

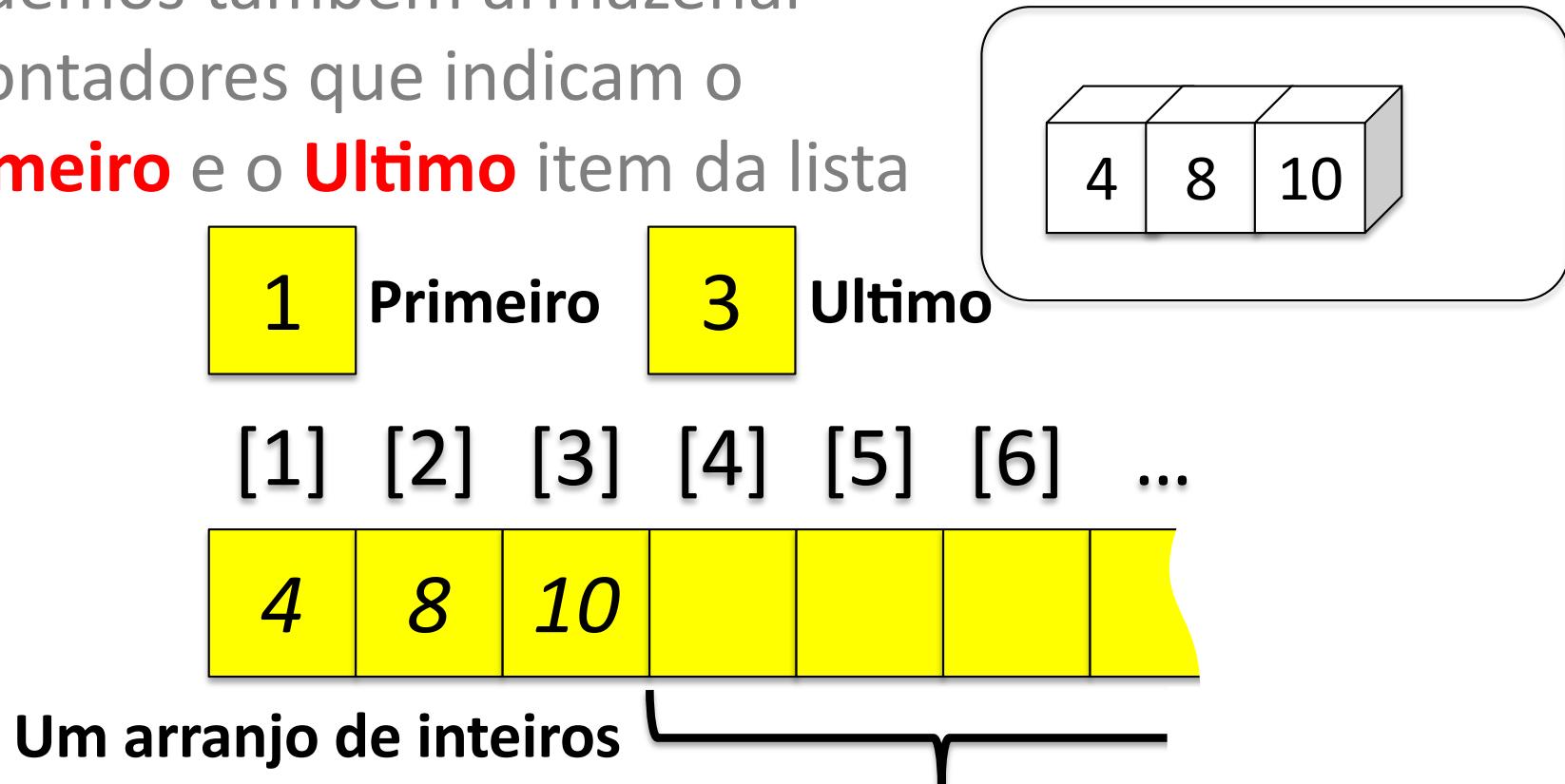


≠



Implementação por Arranjos

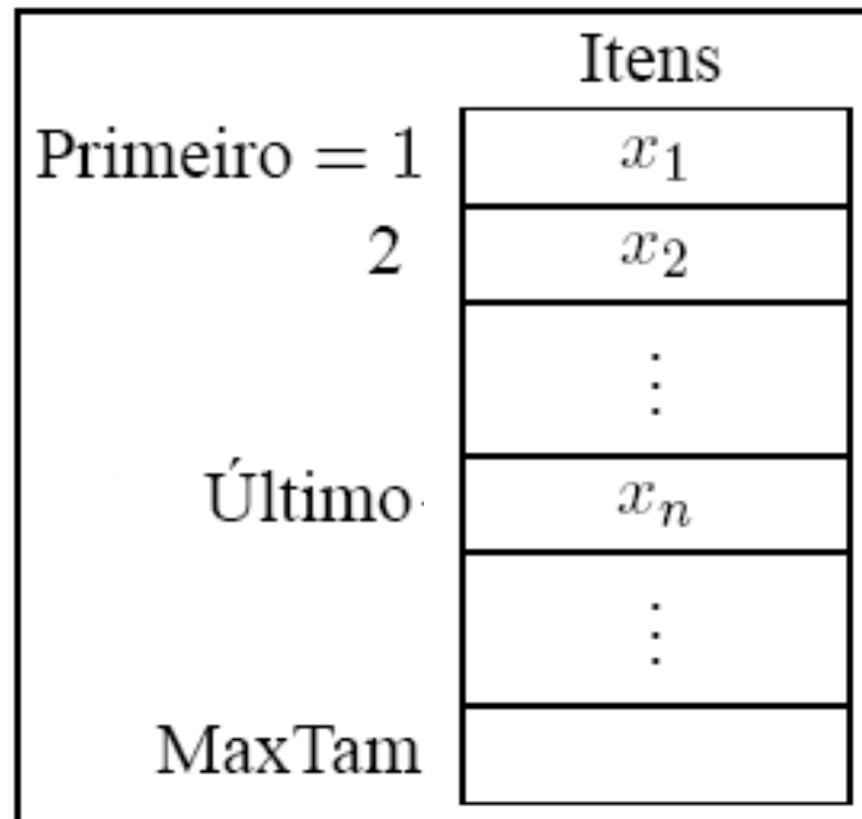
- Podemos também armazenar apontadores que indicam o **Primeiro** e o **Último** item da lista



Não nos interessamos para o que está armazenado nesta parte do arranjo

Estrutura da Lista por Arranjos

- Os itens da lista linear são armazenados em posições contíguas de memória
- A lista linear pode ser percorrida em qualquer direção e partir de qualquer posição



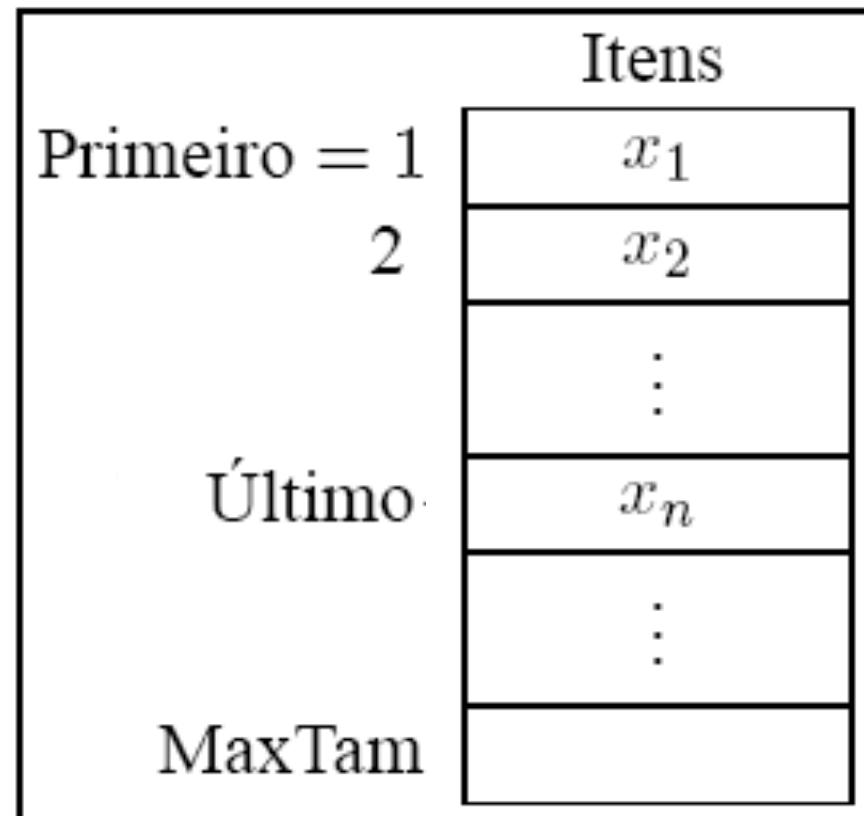
Estrutura da Lista por Arranjos

- A inserção de um novo item no meio da lista requer o deslocamento de todos os itens ...

Custo $O(n)$

- Retirar um item do meio da lista requer o deslocamento de todos os itens para preencher o espaço vazio ...

Custo $O(n)$



Estrutura da Lista por Arranjos

- Os itens são armazenados em um **arranjo** de tamanho suficiente para armazenar a lista
- O campo **Primeiro** mantém um apontador para o primeiro item da lista
- O campo **Ultimo** mantém um apontador para o último item da lista
- O i -ésimo item da lista está armazenado na i -ésima posição do arranjo, $1 \leq i \leq \text{Ultimo}$
- A constante **MaxTam** define o tamanho máximo permitido para a lista

Estrutura da Lista por Arranjos

```
#define INICIO 0
#define MAXTAM 1000

typedef int TChave;

typedef int TApontador;

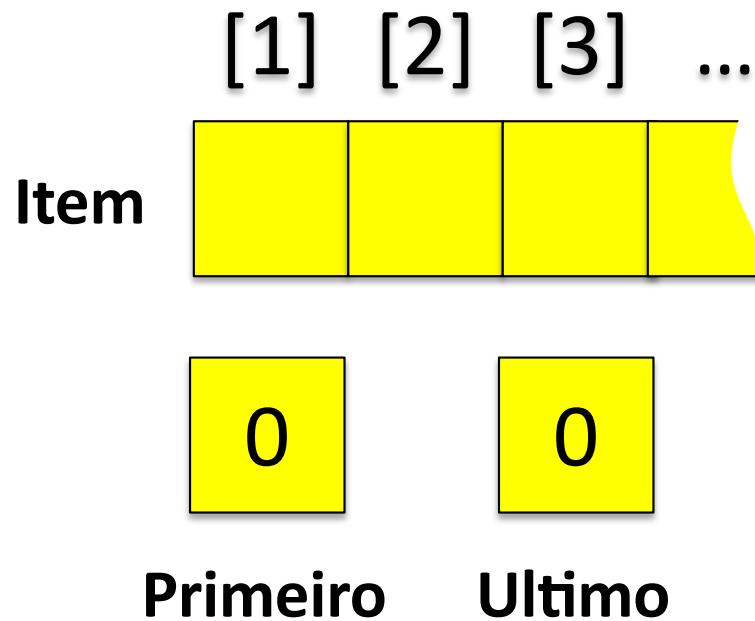
typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct {
    TItem Item[MAXTAM];
    TApontador Primeiro, Ultimo;
} TLista;
```

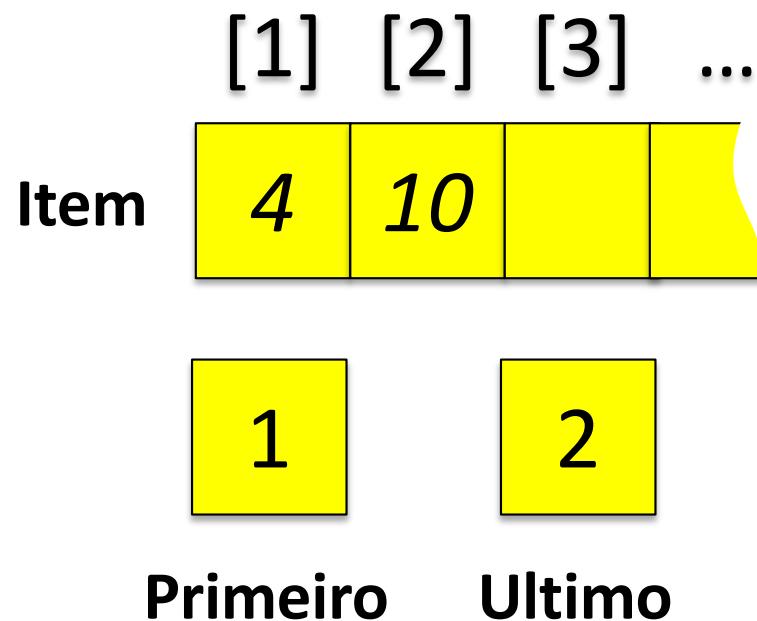
Estrutura da Lista por Arranjos

```
/* procedimentos e funções do TAD */  
  
void TLista_Inicia(TLista *pLista);  
int TLista_EhVazia(TLista *pLista);  
int TLista_Insere(TLista *pLista, TApontador p, TItem x);  
int TLista_Retira(TLista *pLista, TApontador p, TItem *pX);  
int TLista_Tamanho(TLista *pLista);
```

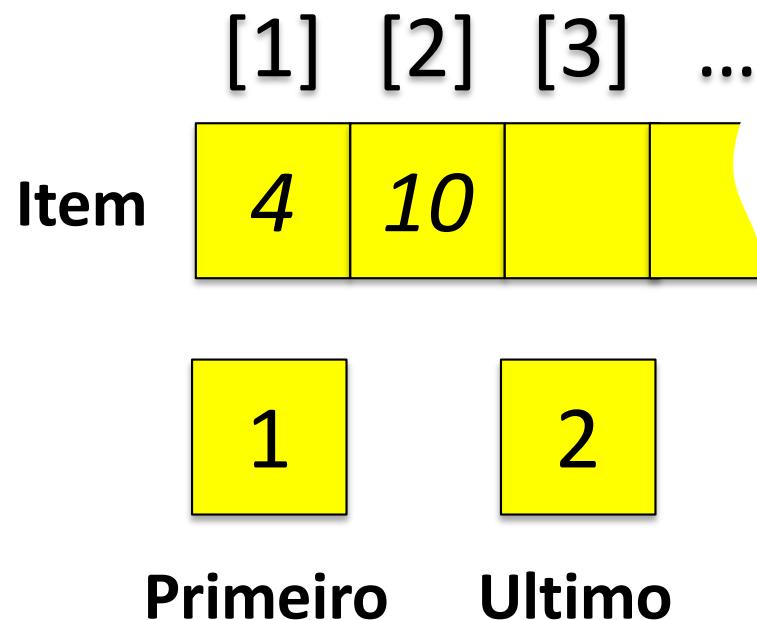
Criar Lista Vazia



Inserção de Elementos na Lista

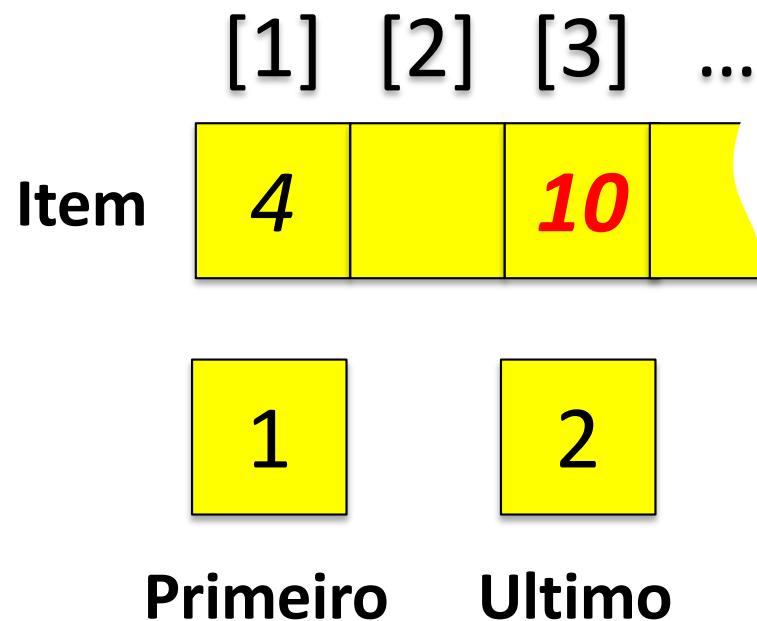


Inserção de Elementos na Lista

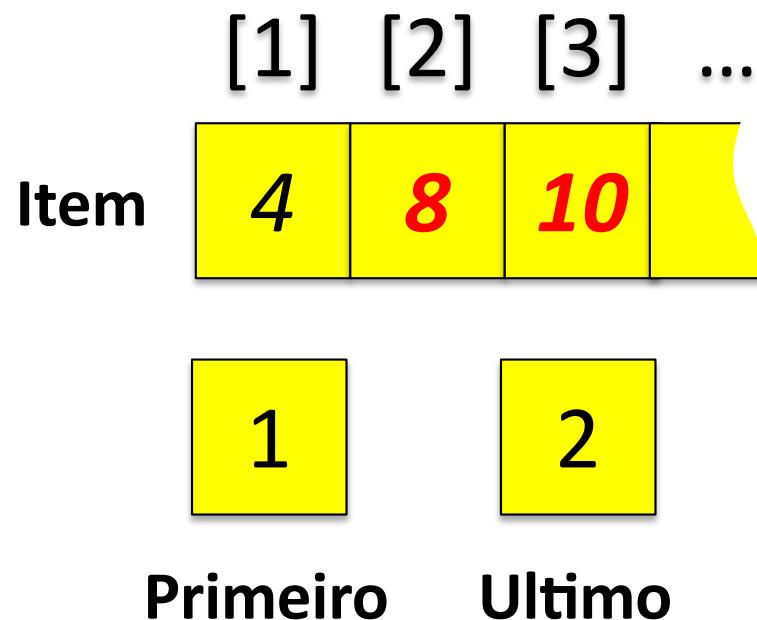


A lista está cheia?

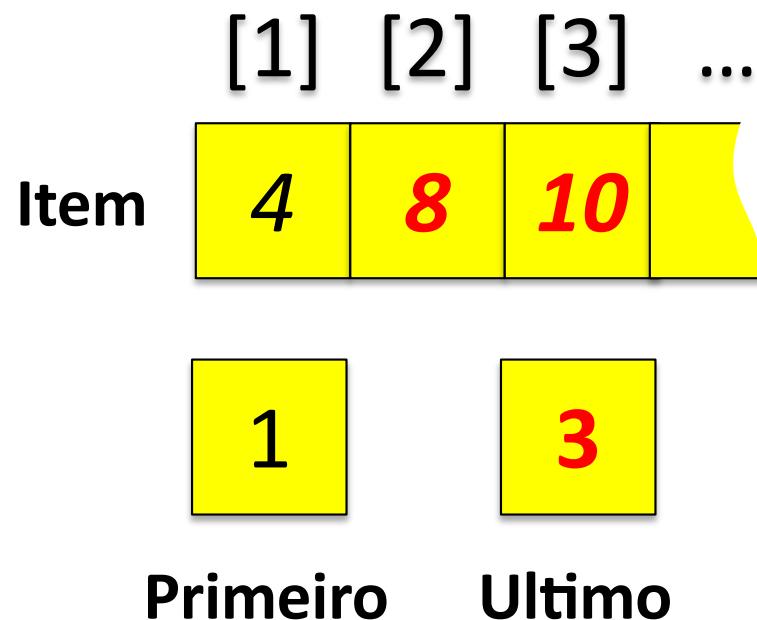
Inserção de Elementos na Lista



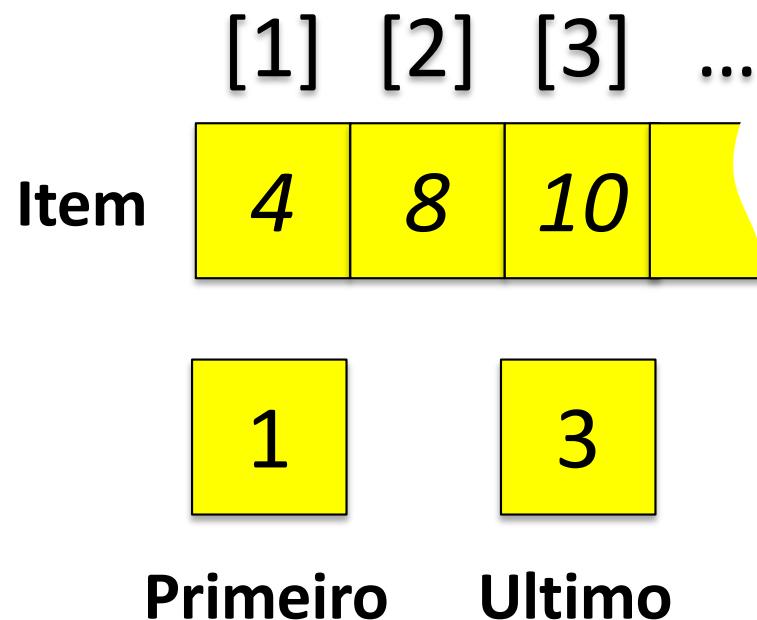
Inserção de Elementos na Lista



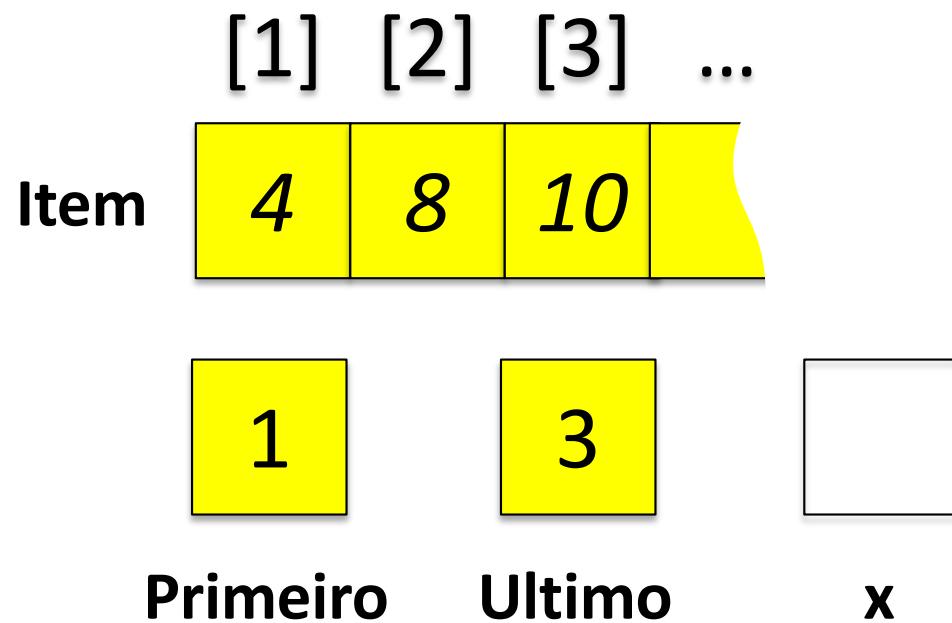
Inserção de Elementos na Lista



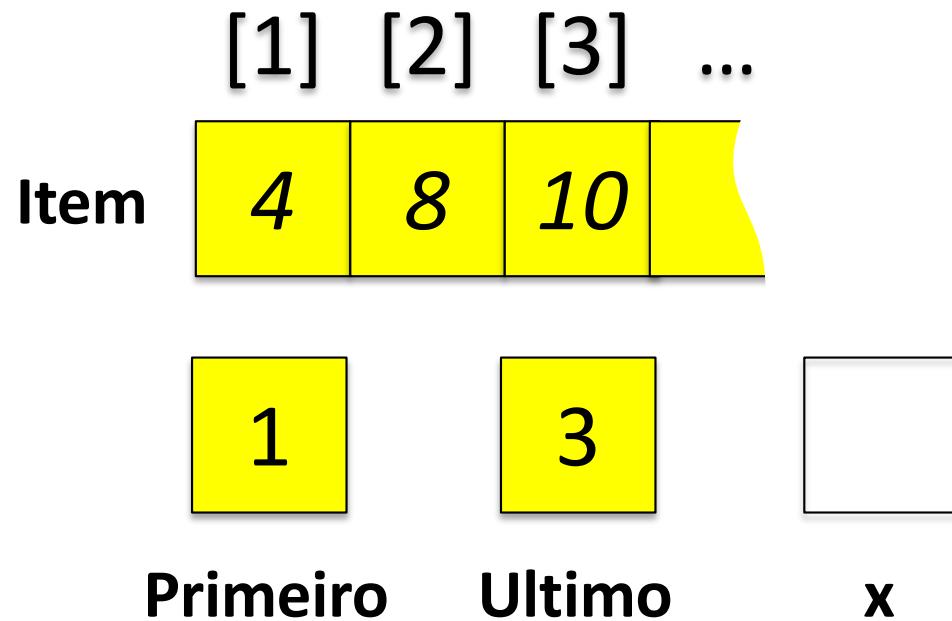
Inserção de Elementos na Lista



Retirada de Elementos na Lista

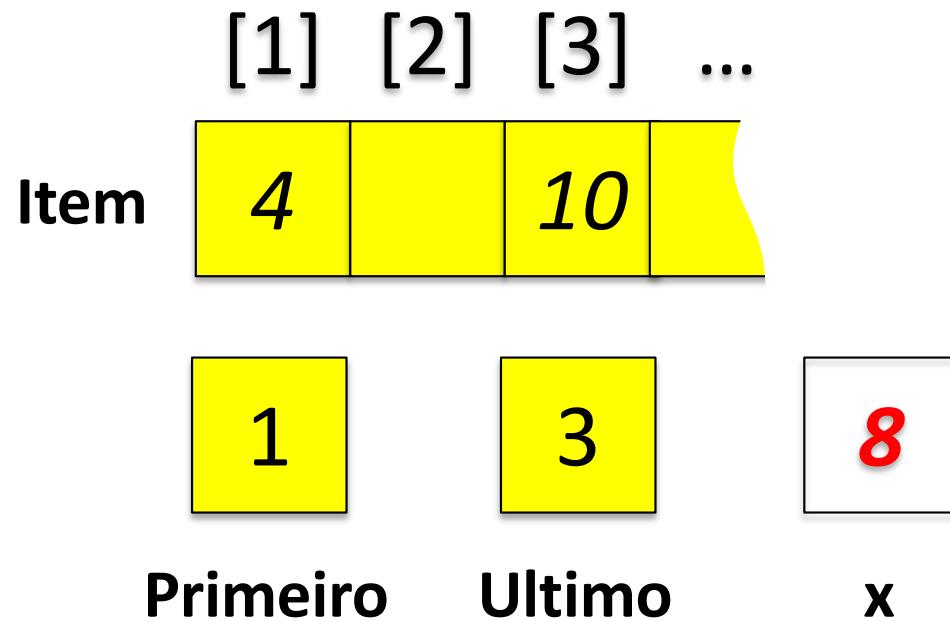


Retirada de Elementos na Lista

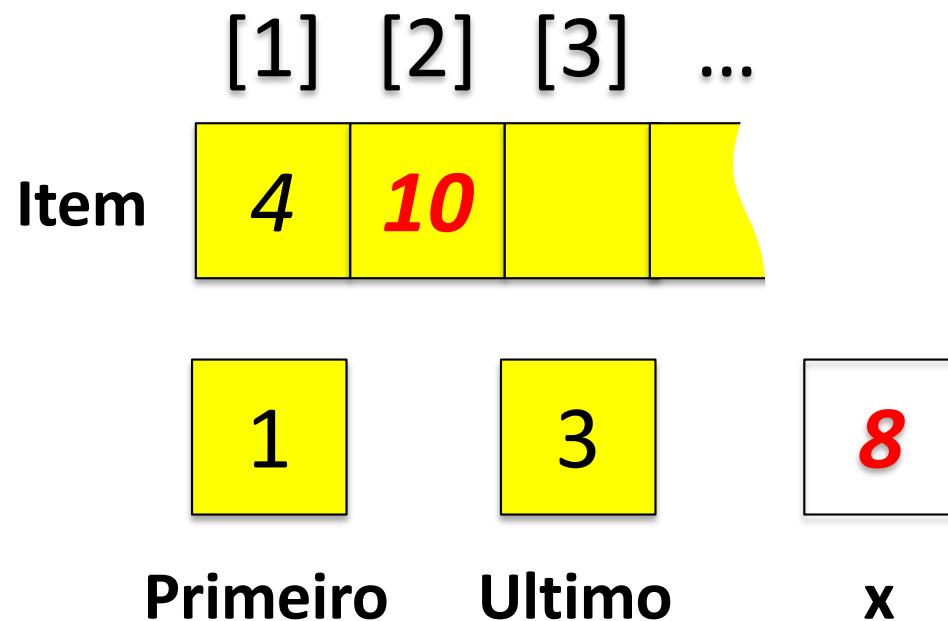


A lista está vazia?

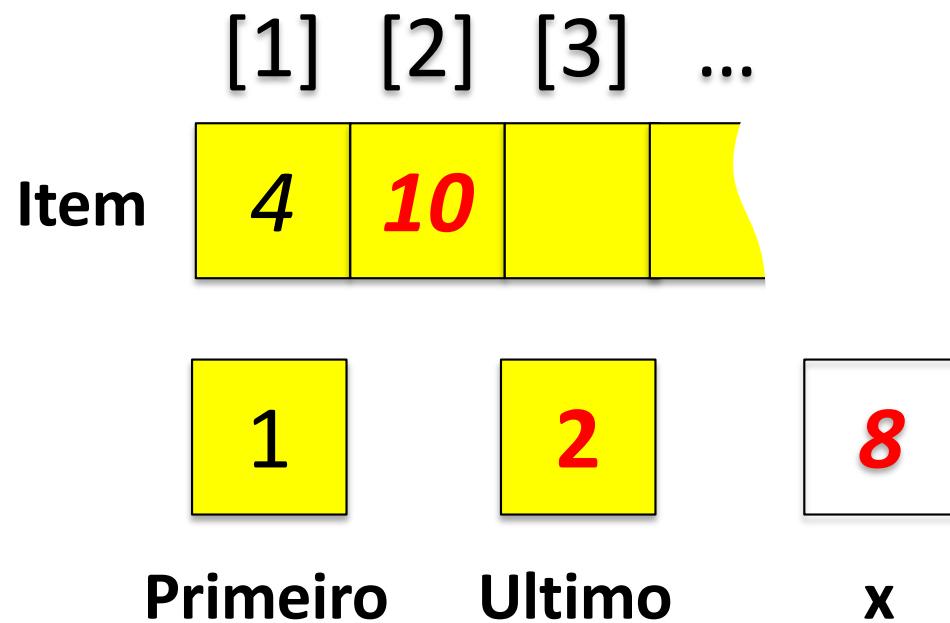
Retirada de Elementos na Lista



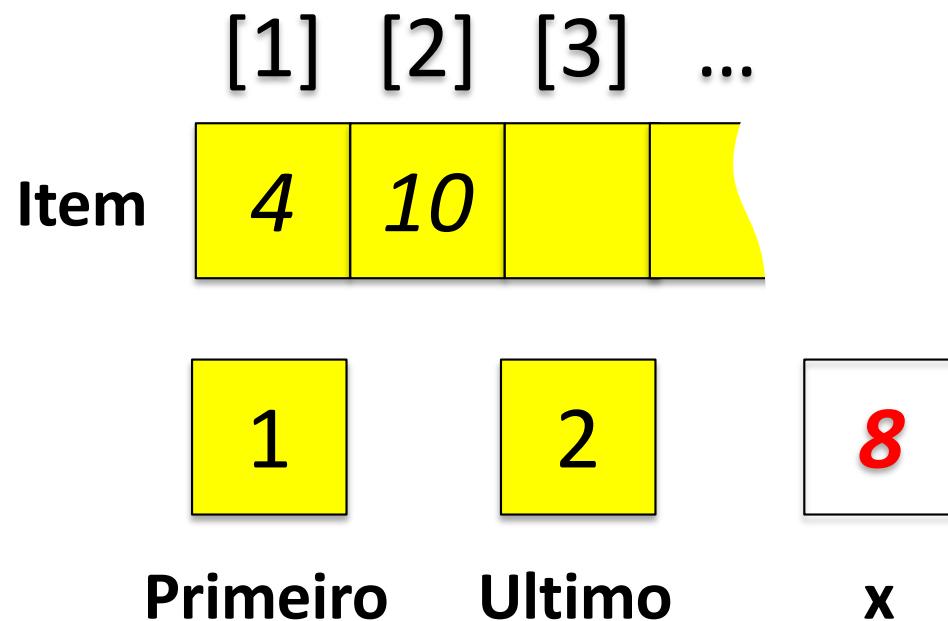
Retirada de Elementos na Lista



Retirada de Elementos na Lista



Retirada de Elementos na Lista



Operações na Lista por Arranjos

```
void TLista_Inicia(TLista *pLista)
{
    pLista->Primeiro = INICIO;
    pLista->Ultimo = pLista->Primeiro;
} /* TLista_Inicia */

int TLista_EhVazia(TLista *pLista)
{
    return (pLista->Primeiro == pLista->Ultimo);
} /* TLista_EhVazia */

int TLista_Tamanho(TLista *pLista)
{
    return (pLista->Ultimo - pLista->Primeiro);
} /* TLista_Tamanho */
```

Operações na Lista por Arranjos

```
int TLista_Insere(TLista *pLista, TApontador p, TItem x)
{
    TApontador q;

    if ((pLista->Ultimo == MAXTAM) ||
        (p < pLista->Primeiro) ||
        (p > pLista->Ultimo))
        return 0;

    for (q = pLista->Ultimo-1; q >= p; q--)
        pLista->Item[q+1] = pLista->Item[q];
    pLista->Item[p] = x;
    pLista->Ultimo++;

    return 1;
} /* TLista_Insere */
```

Operações na Lista por Arranjos

```
int TLista_Retira(TLista *pLista, TApontador p, TItem *pX)
{
    TApontador q;

    if (TLista_EhVazia(pLista) ||
        (p < pLista->Primeiro) ||
        (p >= pLista->Ultimo))
        return 0;

    *pX = pLista->Item[p];
    for (q = p+1; q < pLista->Ultimo; q++)
        pLista->Item[q-1] = pLista->Item[q];
    pLista->Ultimo--;

    return 1;
} /* TLista_Retira */
```

■ Vantagens:

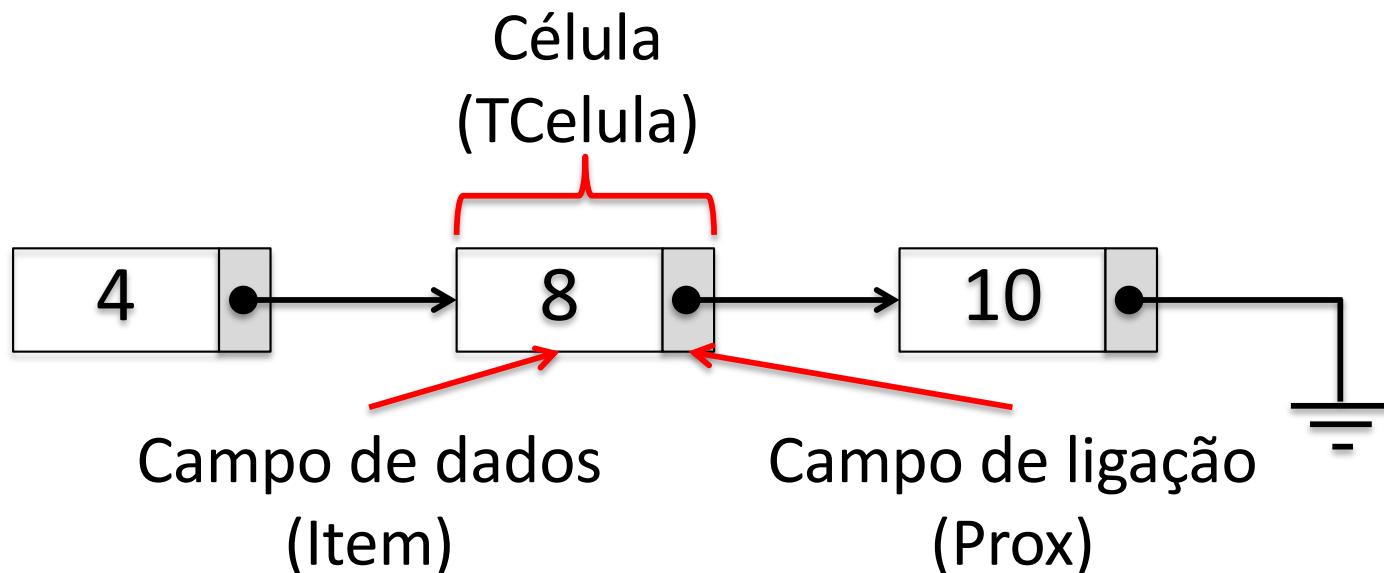
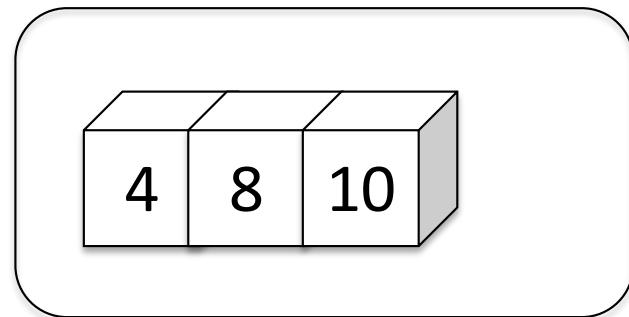
- Economia de memória***
 - Os apontadores são implícitos nessa estrutura
- Acesso a qualquer elemento da lista é feito em tempo $O(1)$

■ Desvantagens:

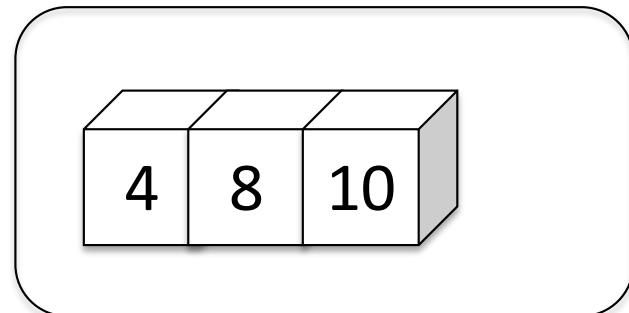
- Custo para inserir itens na lista pode ser $O(n)$
- Custo para retirar itens da lista pode ser $O(n)$
- Quando não existe previsão sobre o crescimento da lista, o arranjo que define a lista deve ser alocado dinamicamente
 - O arranjo pode precisar ser realocado

IMPLEMENTAÇÃO POR APONTADORES

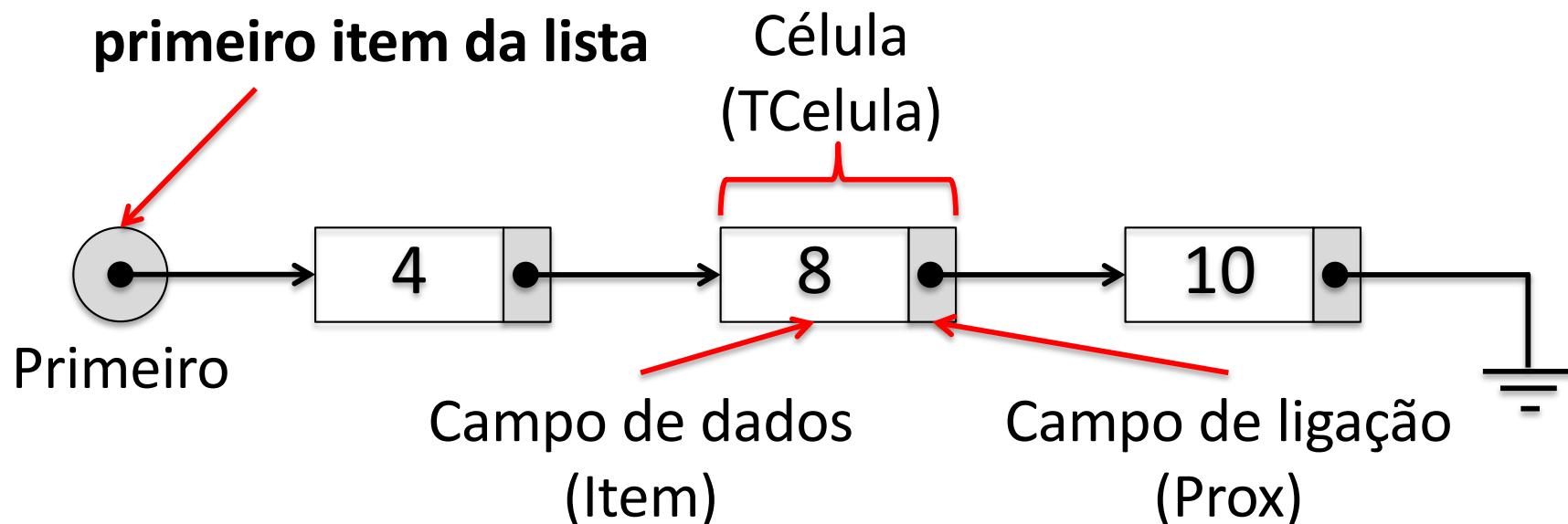
- Os itens são colocados em uma estrutura (**TCelula**) que contém um campo com um item (**Item**) e outro campo com um apontador para a próxima célula (**Prox**)



- Nós precisamos também armazenar um apontador que indica o **Primeiro** item da lista

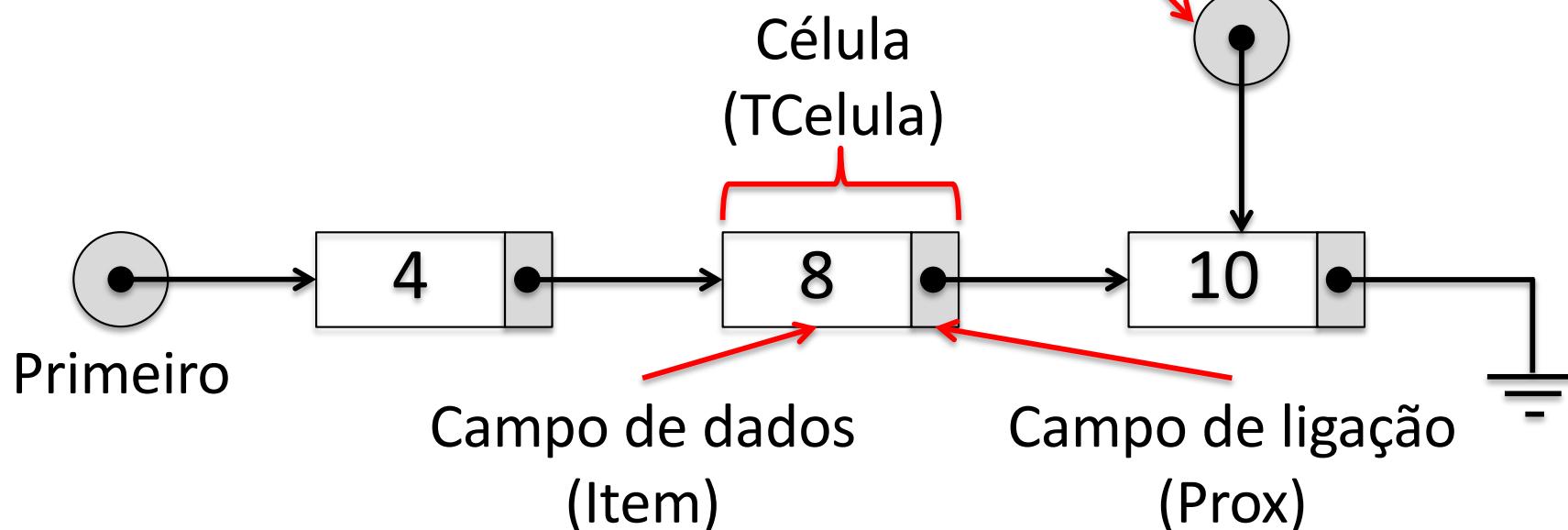


Um apontador indica o primeiro item da lista



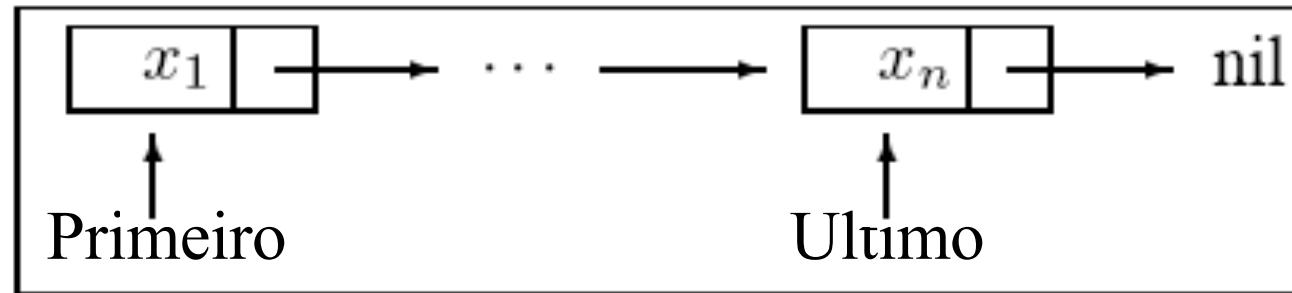
- Pode ser conveniente armazenar um apontador que indica o **Ultimo** item da lista

Outro apontador indica o último item da lista



Estrutura da Lista por Apontador

- Cada item é encadeado com o seguinte por meio de um apontador
- Permite utilizar posições não contíguas de memória



- O campo **Tamanho** evita a contagem do número de itens na função **TLista_Tamanho**
- Cada célula contém um item da lista e um apontador para outra célula
- O registro **TLista** contém um apontador para o **Primeiro** e um apontador o **Ultimo** item da lista

Estrutura da Lista por Apontador

```
typedef int TChave;

typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct SCelula *TApontador;

typedef struct SCelula {
    TItem Item;
    TApontador Prox;
} TCelula;

typedef struct {
    TApontador Primeiro, Ultimo;
    int Tamanho;
} TLista;
```

Cria Lista Vazia

Tamanho

0

Primeiro

NULL

NULL

Ultimo

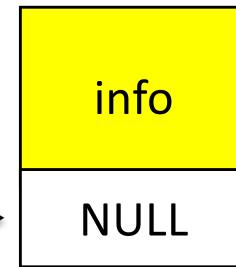
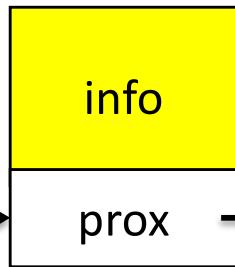
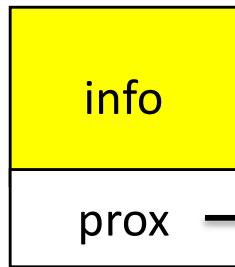
Encontrar uma Posição na Lista

Tamanho

3

Primeiro

—



Último

—

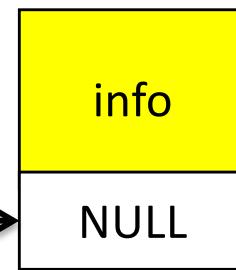
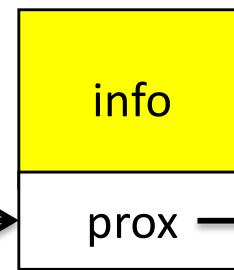
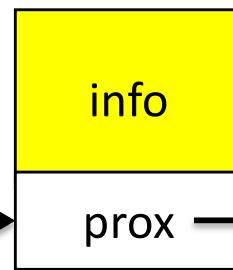
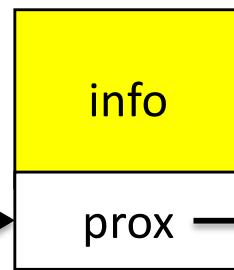
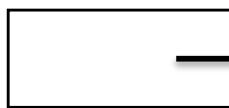
- É necessário um método que recebe como entrada a posição – um inteiro indicando um índice na lista – e retorna um apontador para a célula na lista:
 - **TLista_Retorna(Lista, p)**: Retorna um apontador para a célula que contém o item na posição p da lista

Encontrar uma Posição na Lista

Tamanho

4

Primeiro



Último



p



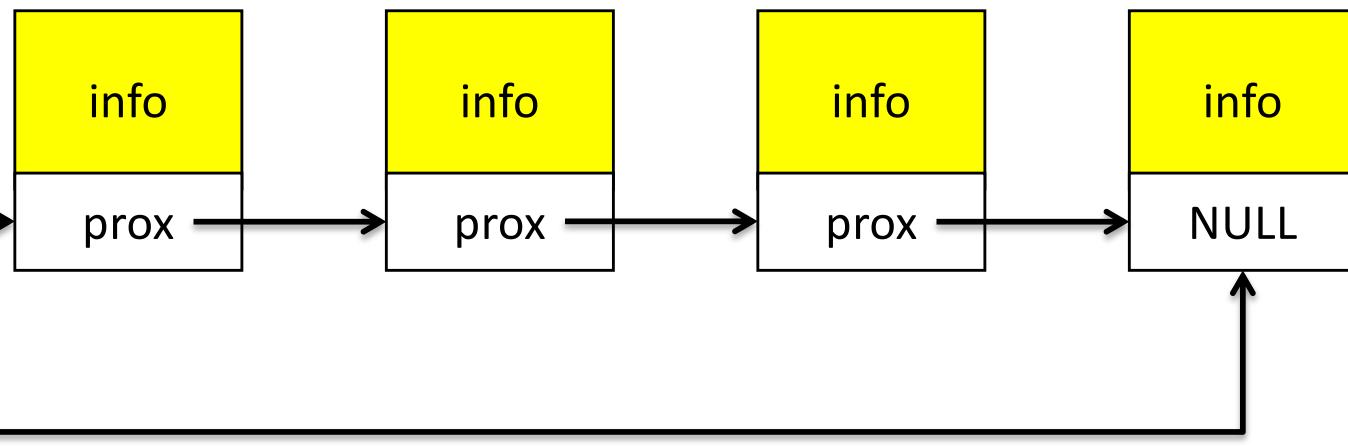
Encontrar uma Posição na Lista

Tamanho

4

Primeiro

—



Último

—

p

3

Apontador

—

Posicao

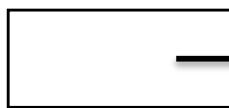
—

Encontrar uma Posição na Lista

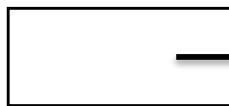
Tamanho

4

Primeiro



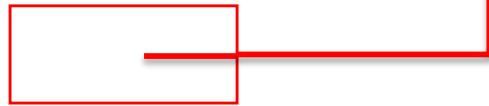
Último



p

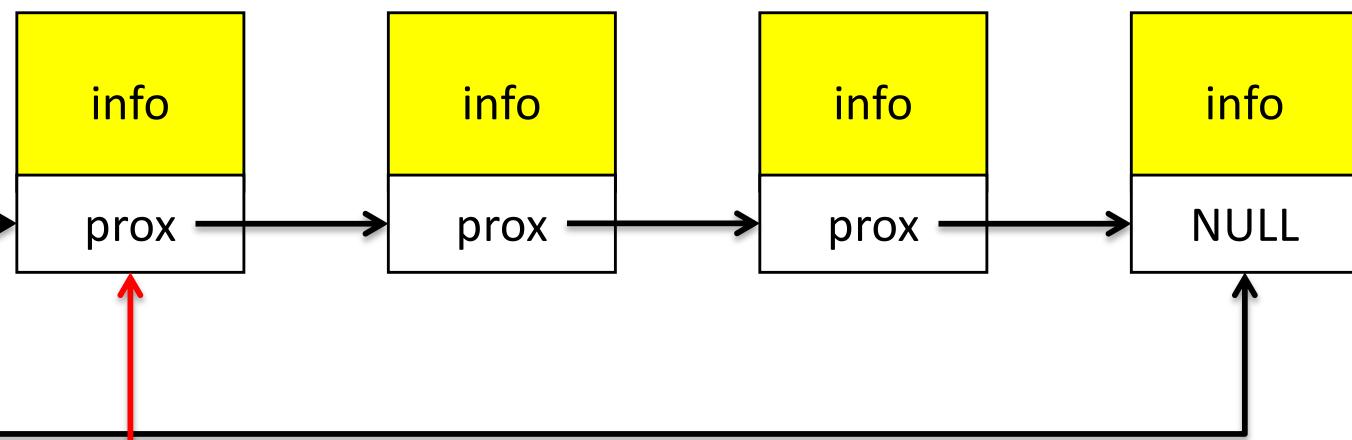
3

Apontador



Posicao

1

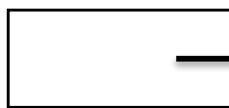


Encontrar uma Posição na Lista

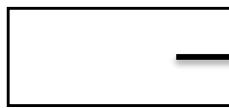
Tamanho

4

Primeiro



Último



p



Apontador



Posicao

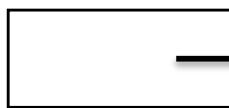
2

Encontrar uma Posição na Lista

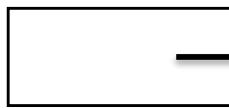
Tamanho

4

Primeiro



Último



p



Apontador



Posicao

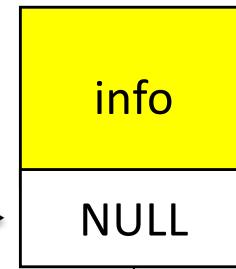
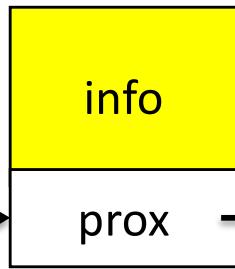
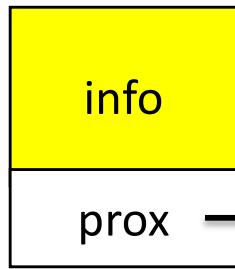
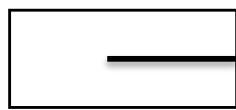
3

Inserção de Elementos na Lista

Tamanho

3

Primeiro

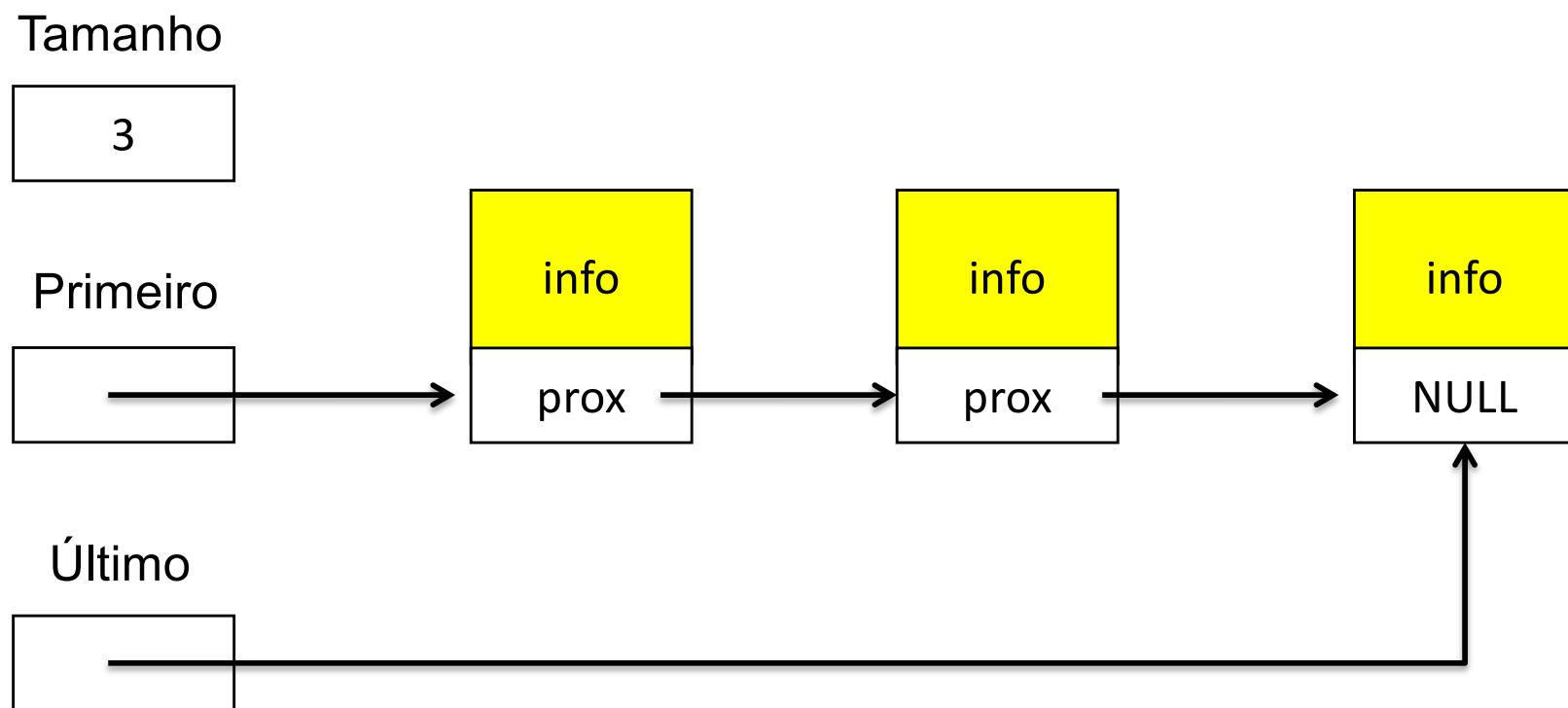


Último

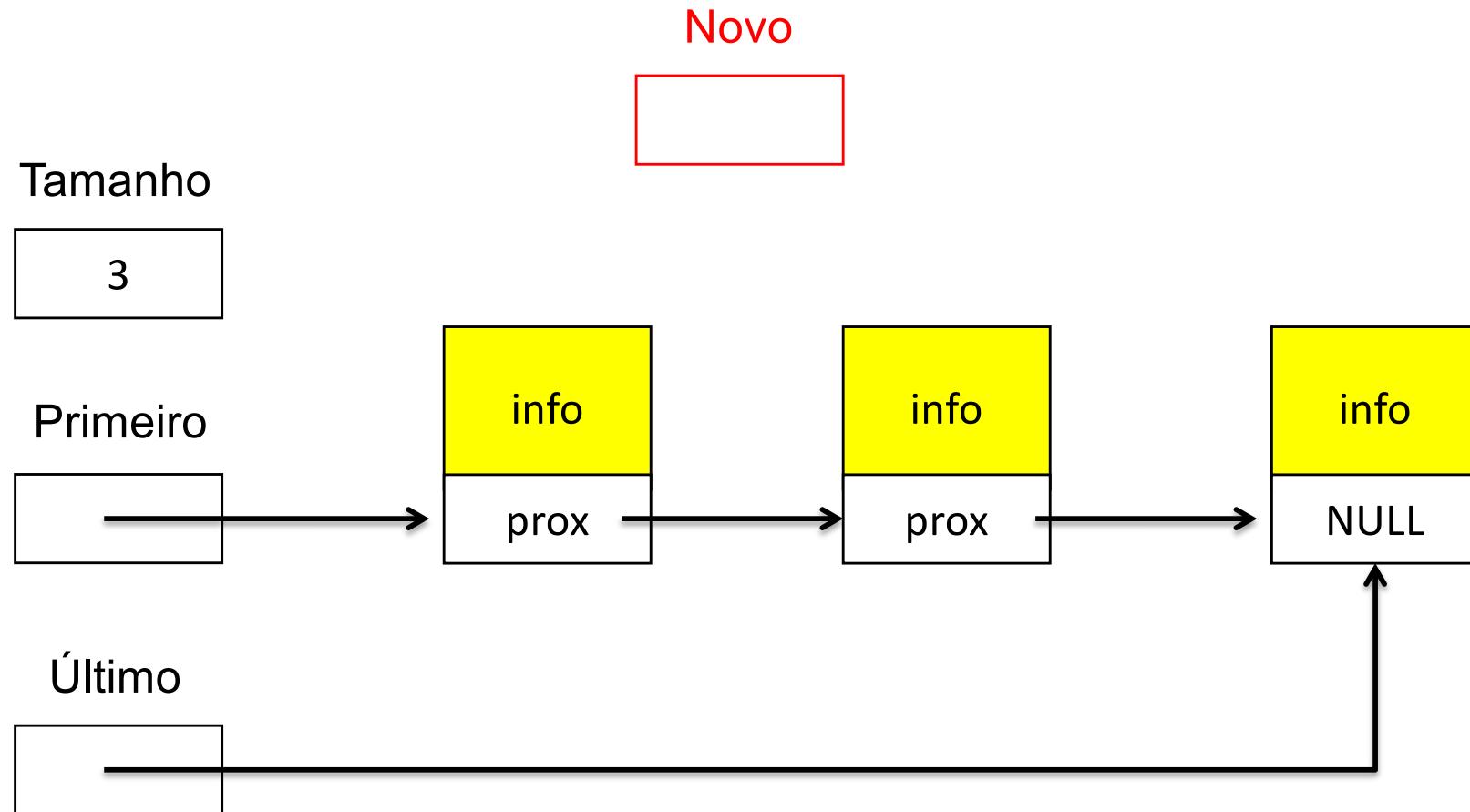


- 3 opções de posição onde pode inserir:
 - 1^a posição
 - última posição
 - uma posição qualquer

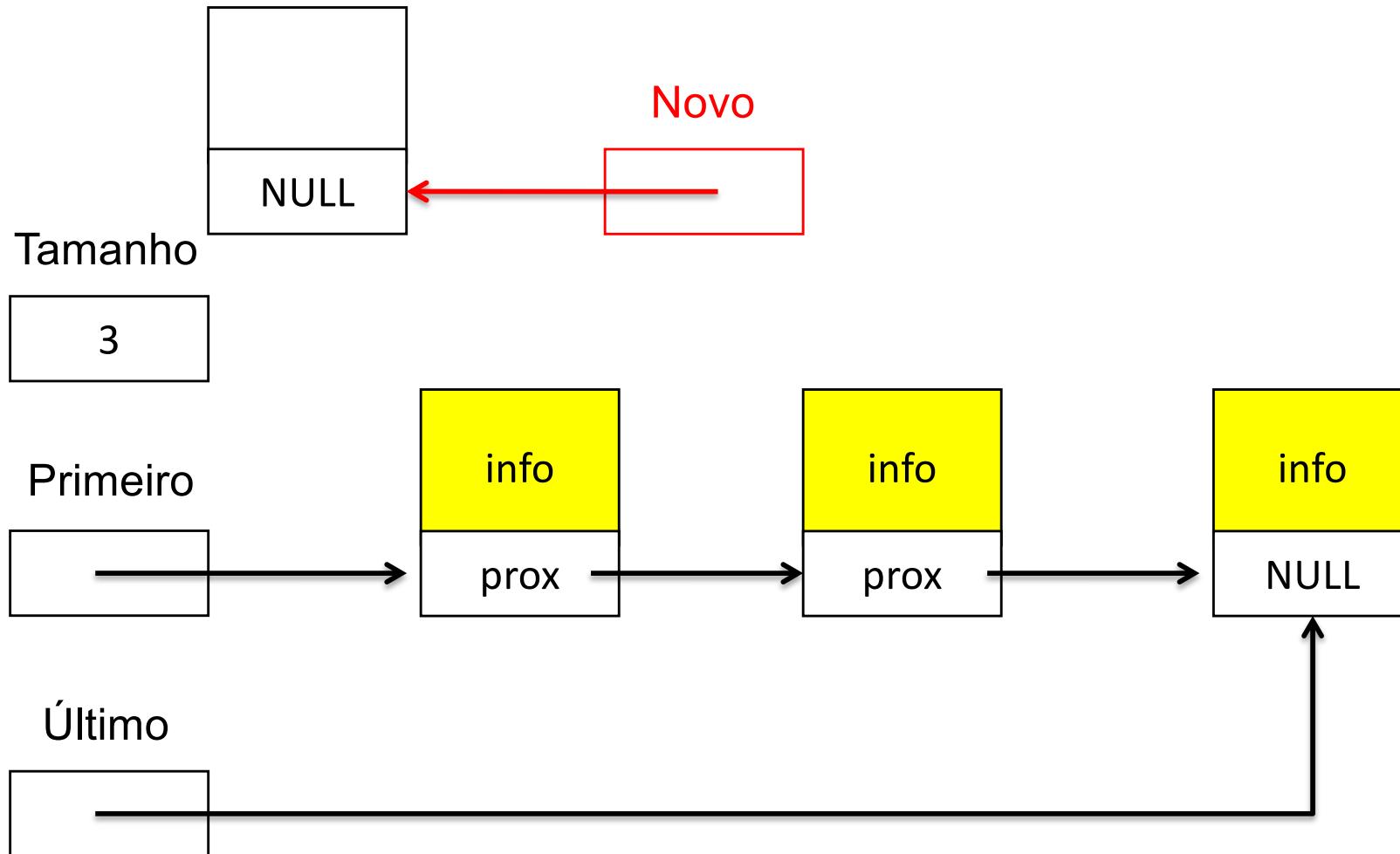
Inserção na Primeira Posição



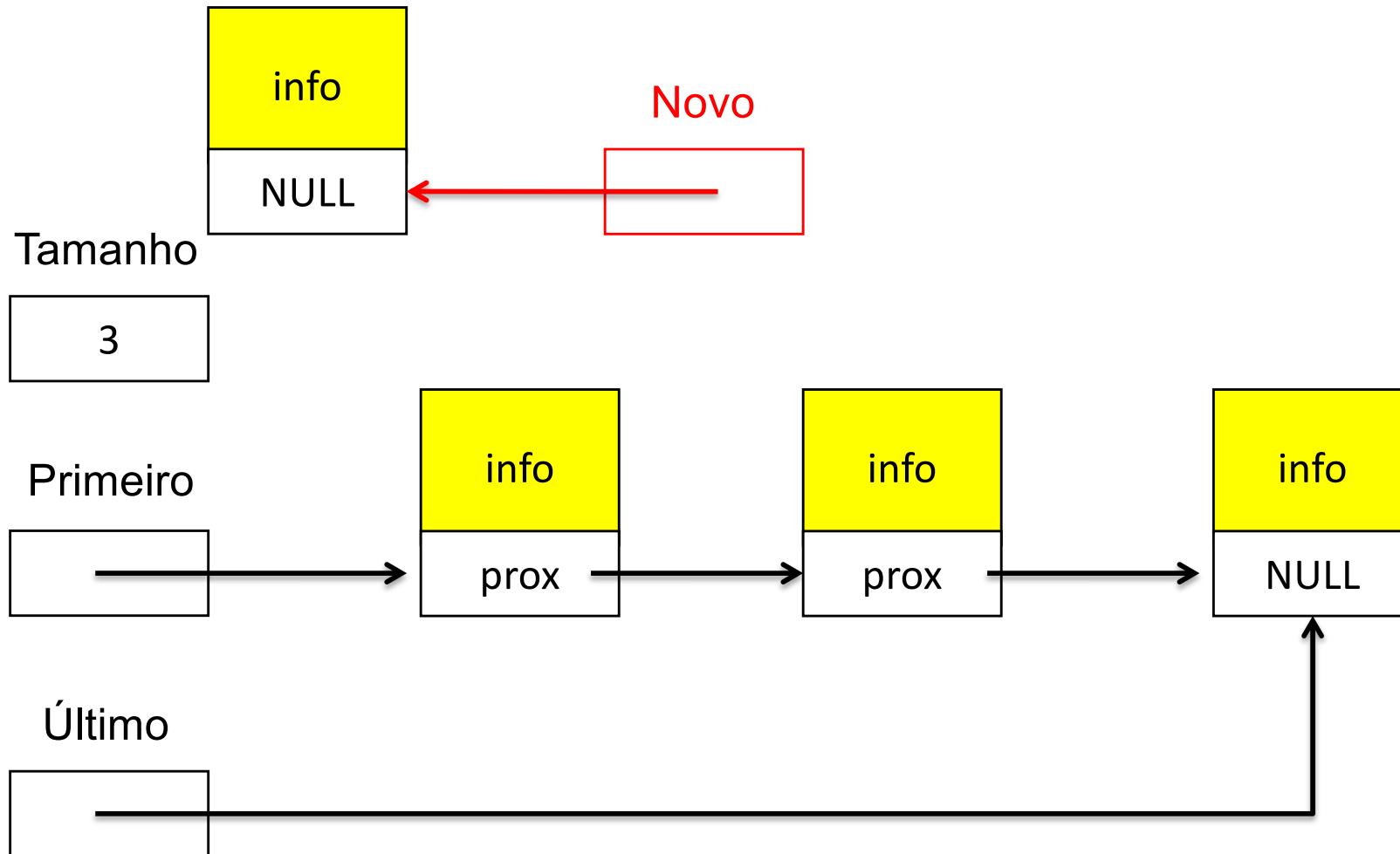
Inserção na Primeira Posição



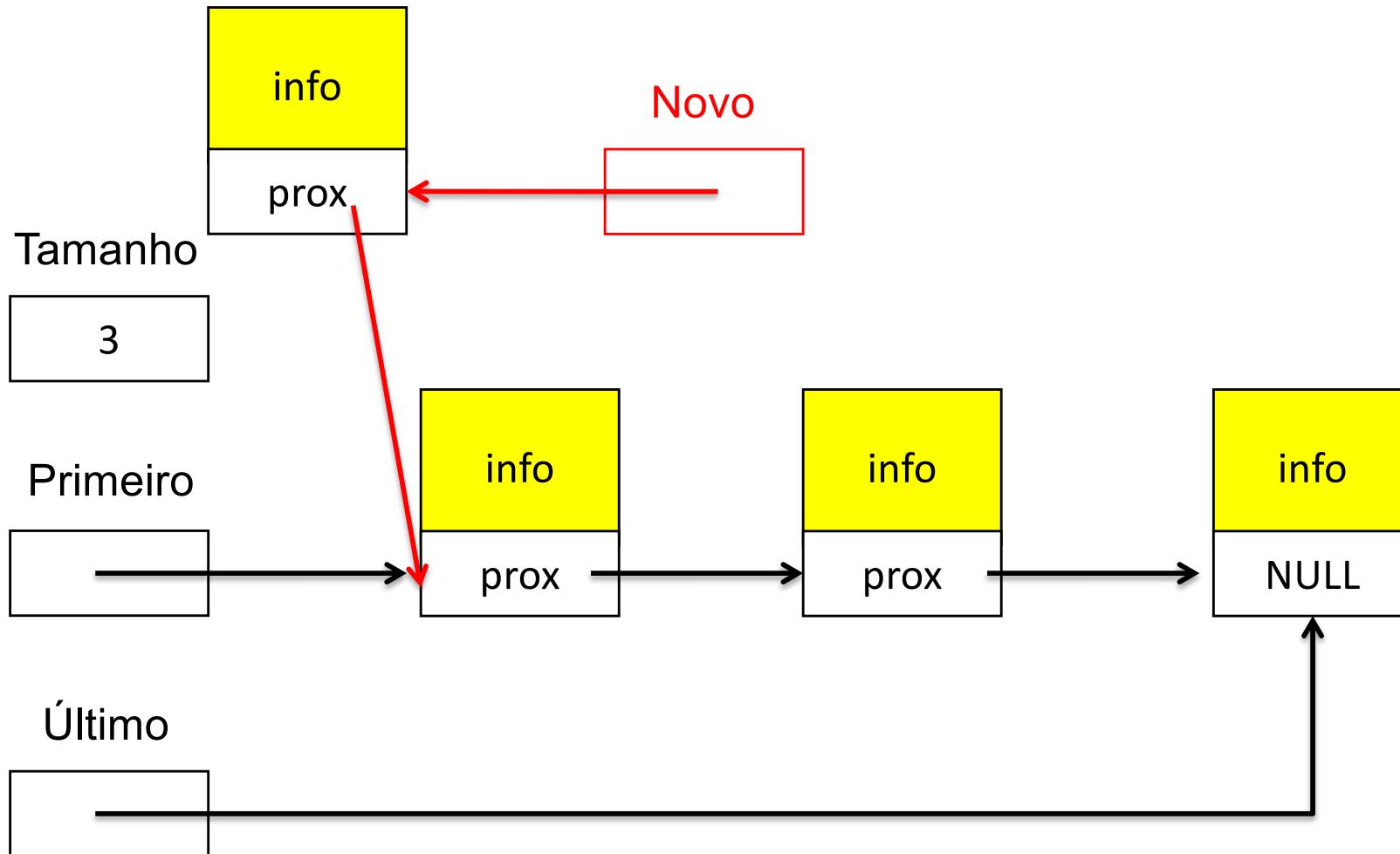
Inserção na Primeira Posição



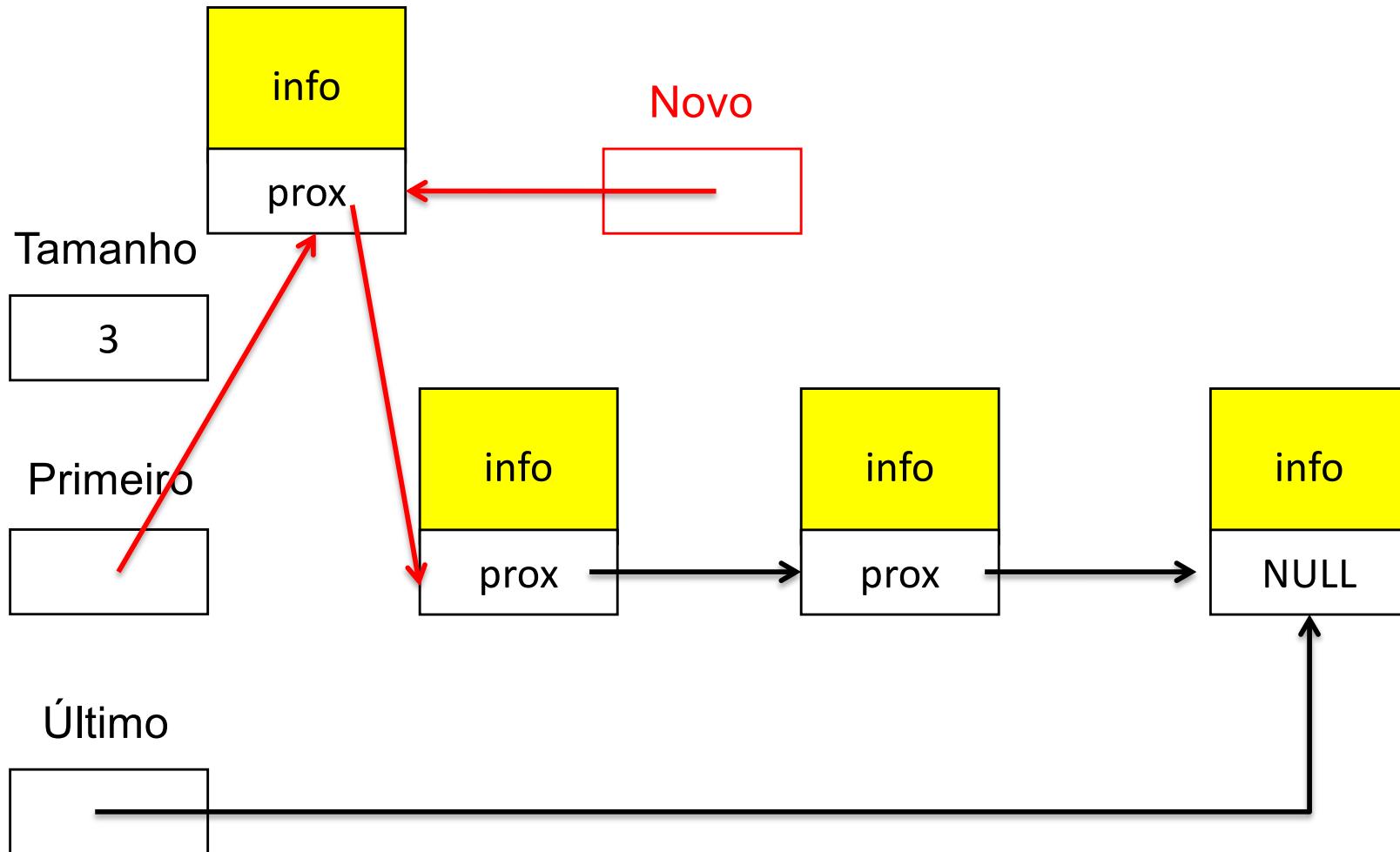
Inserção na Primeira Posição



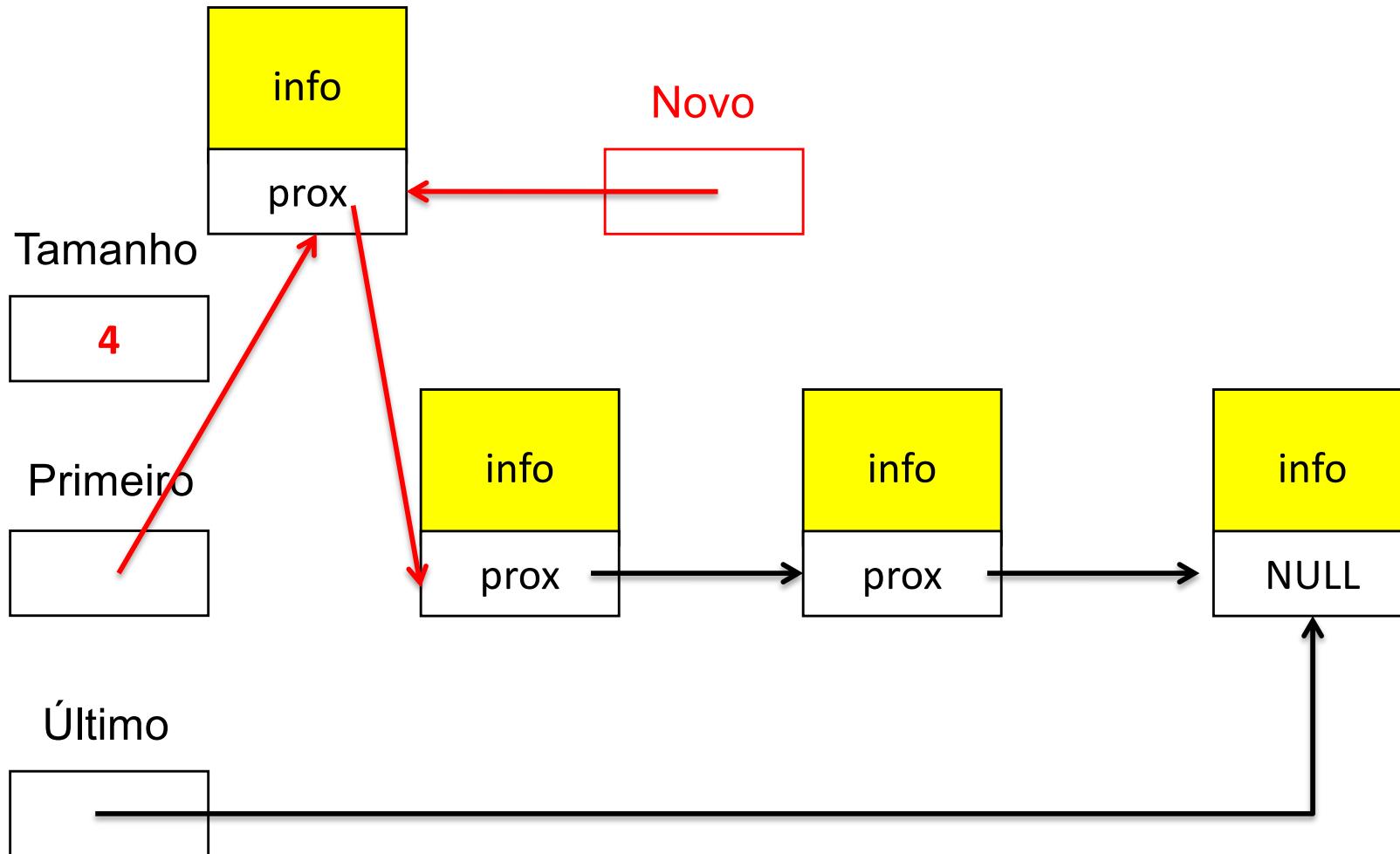
Inserção na Primeira Posição



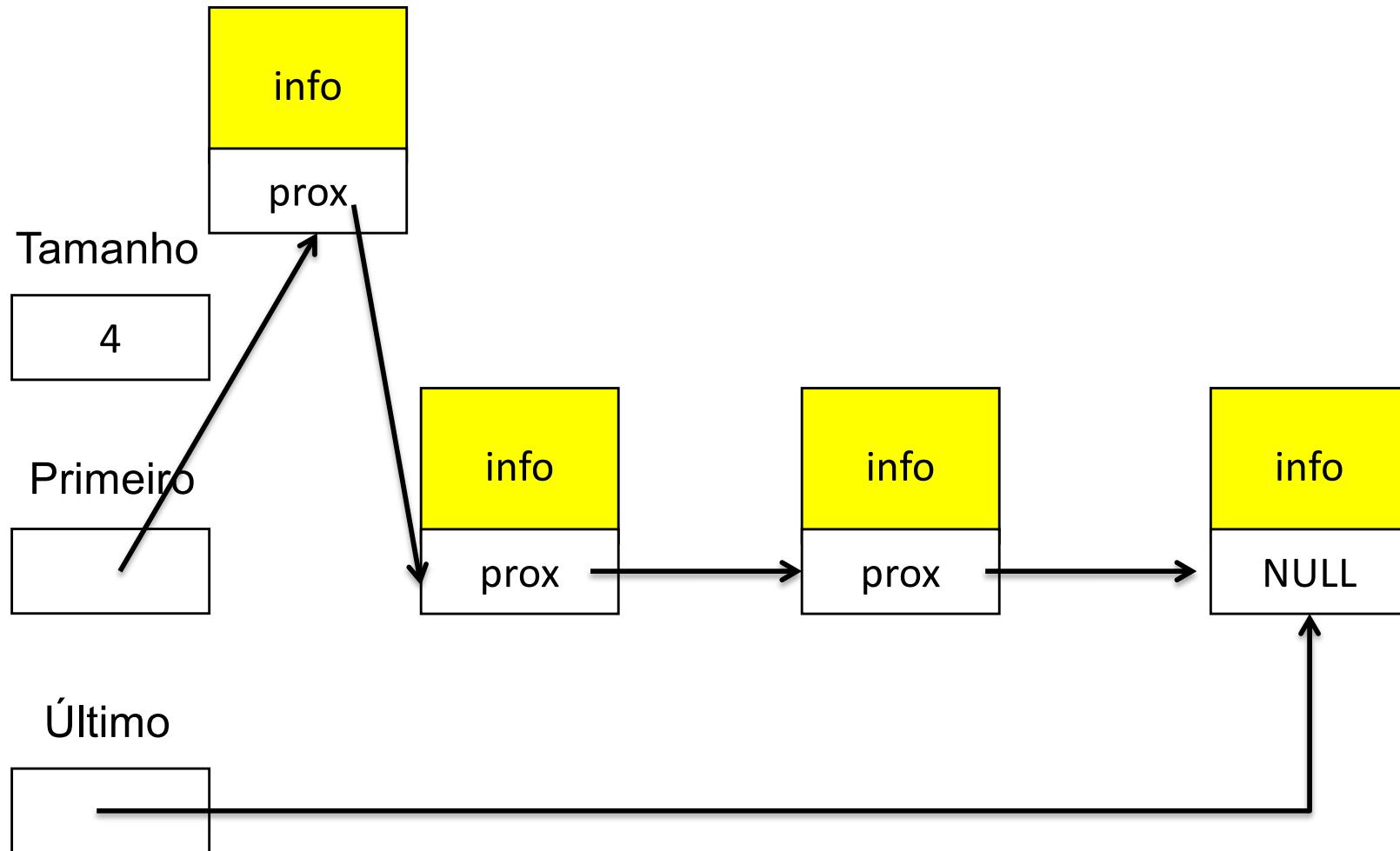
Inserção na Primeira Posição



Inserção na Primeira Posição

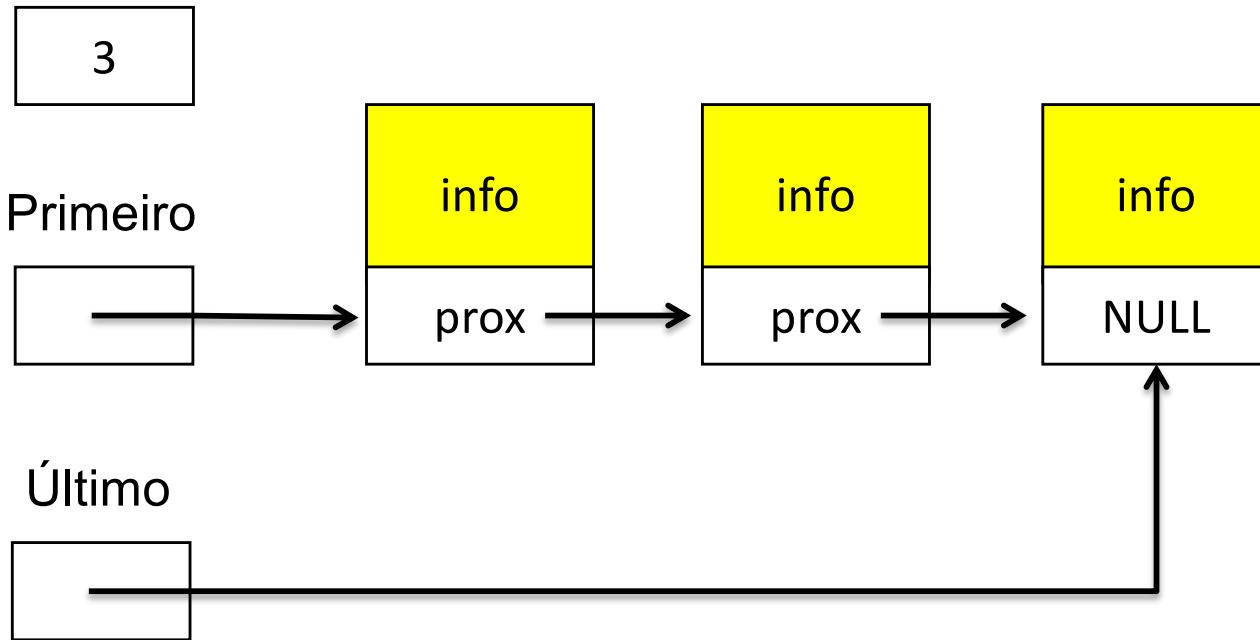


Inserção na Primeira Posição

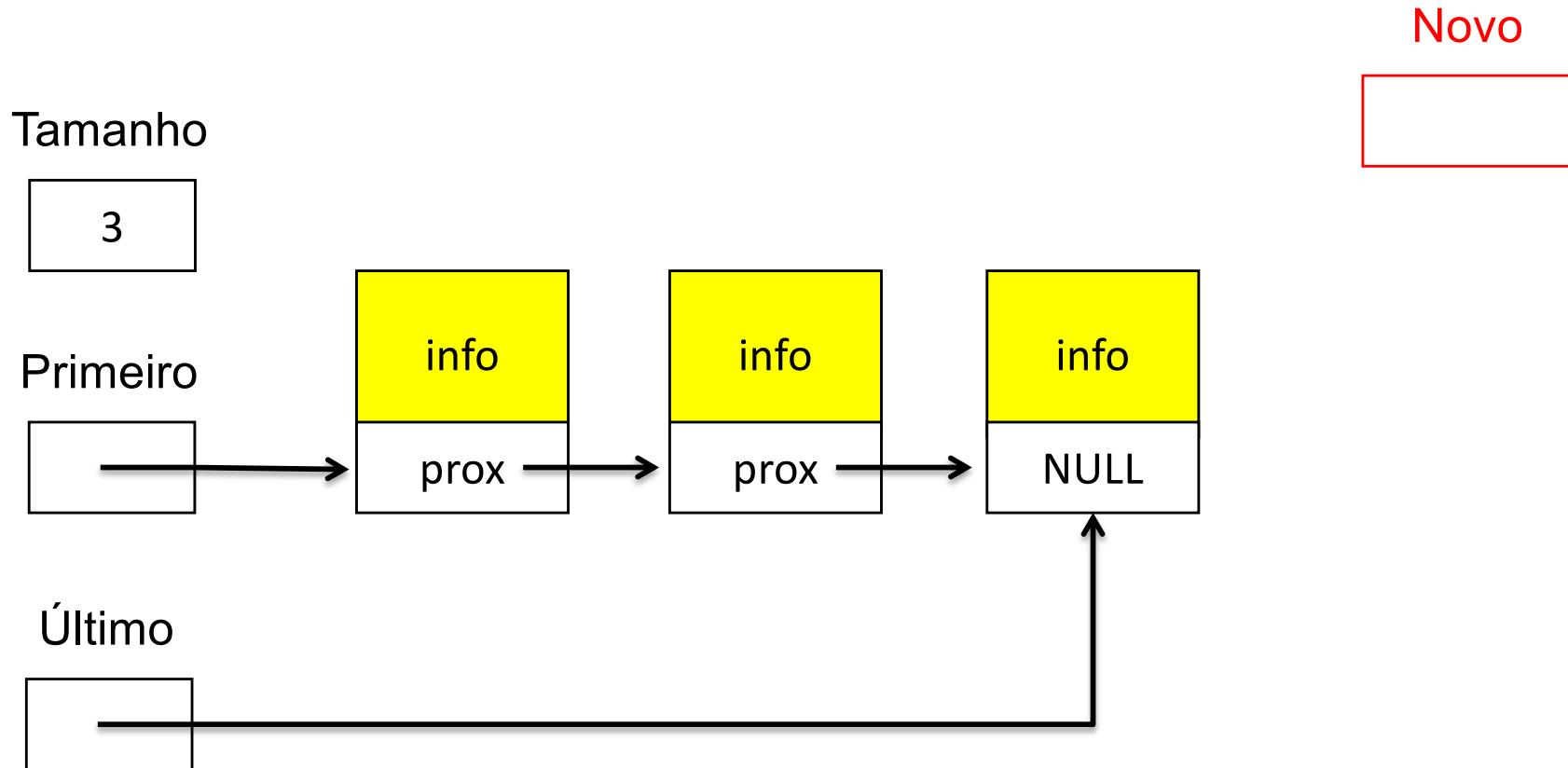


Inserção na Última Posição

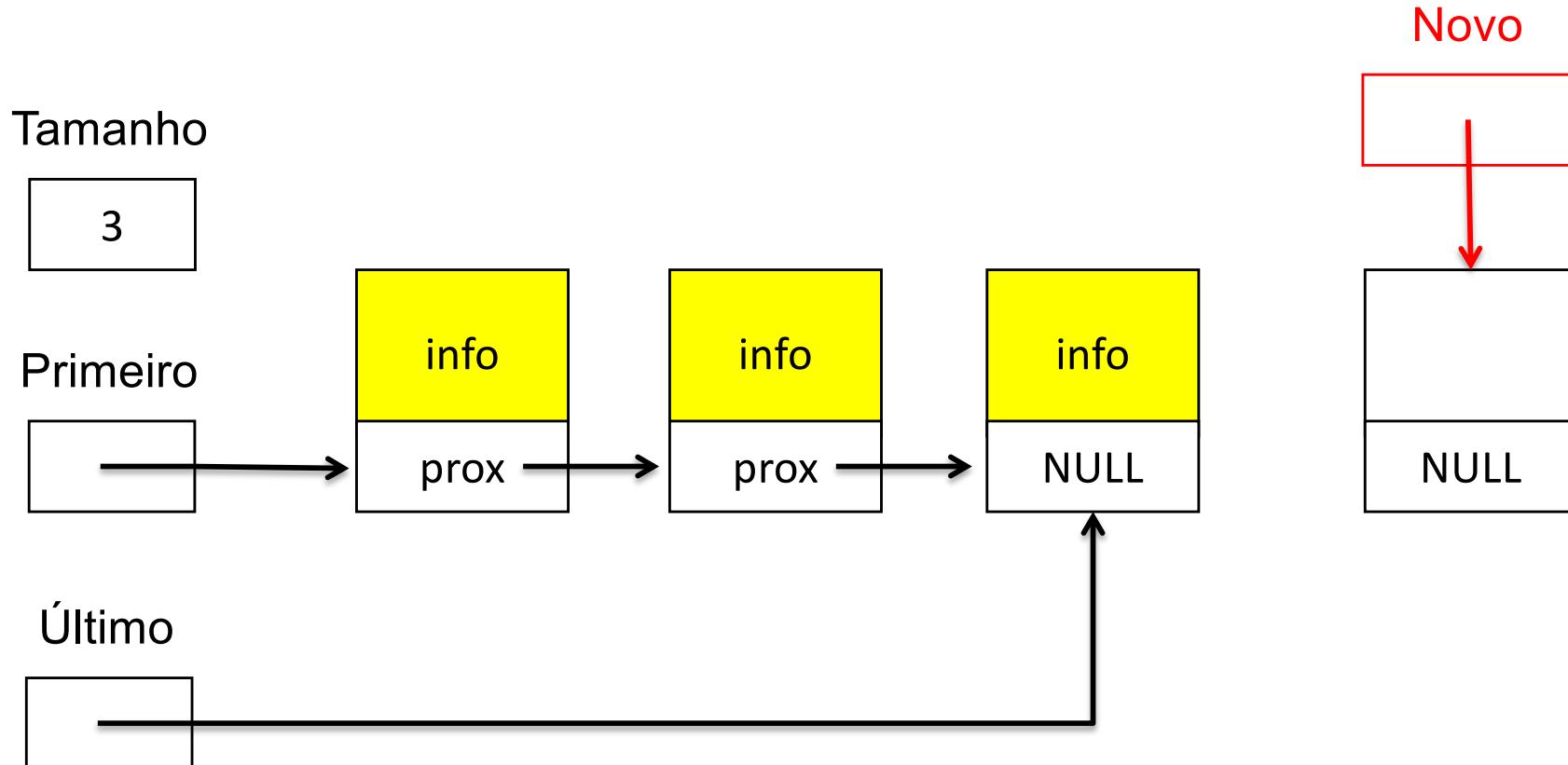
Tamanho



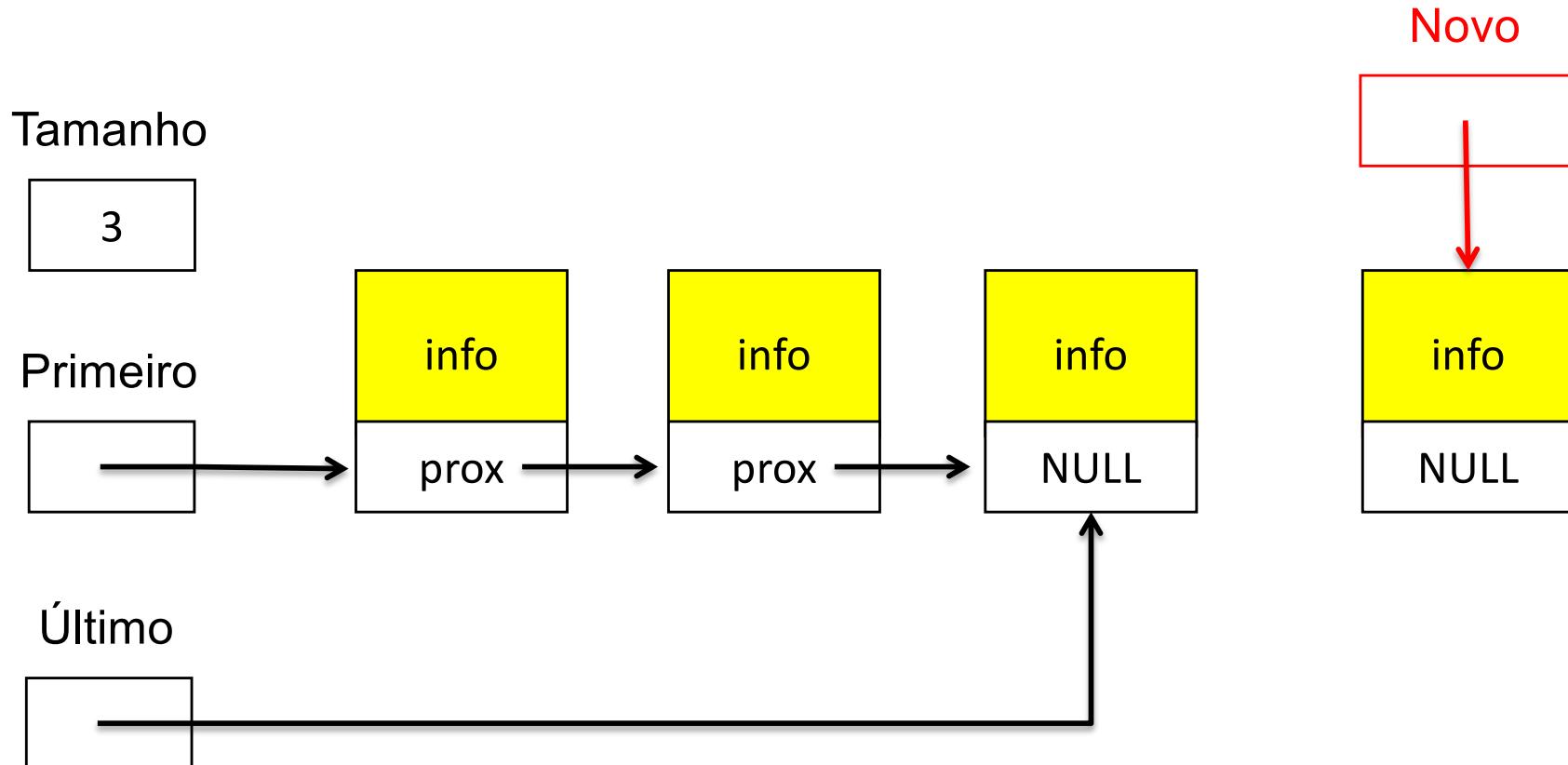
Inserção na Última Posição



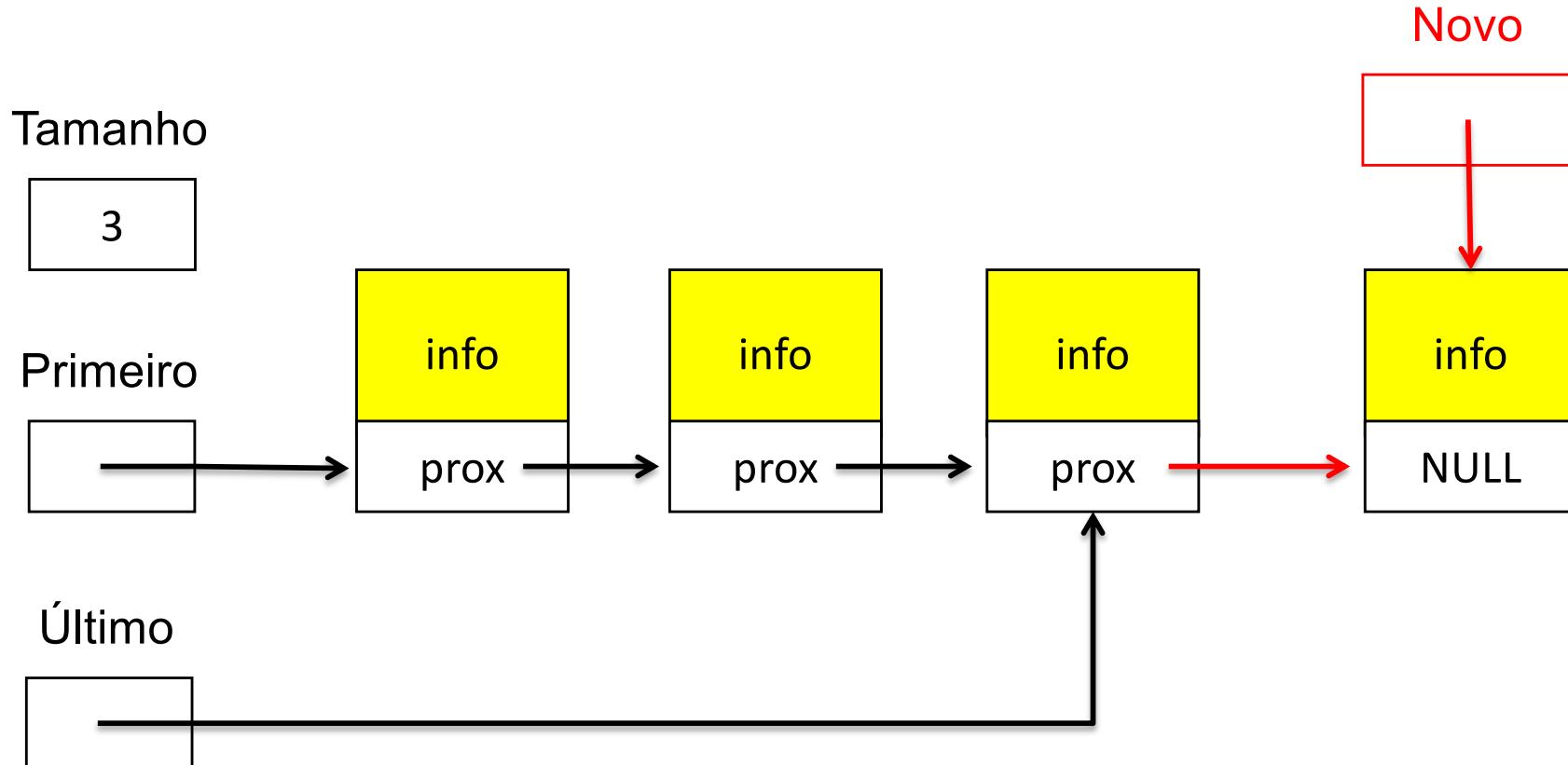
Inserção na Última Posição



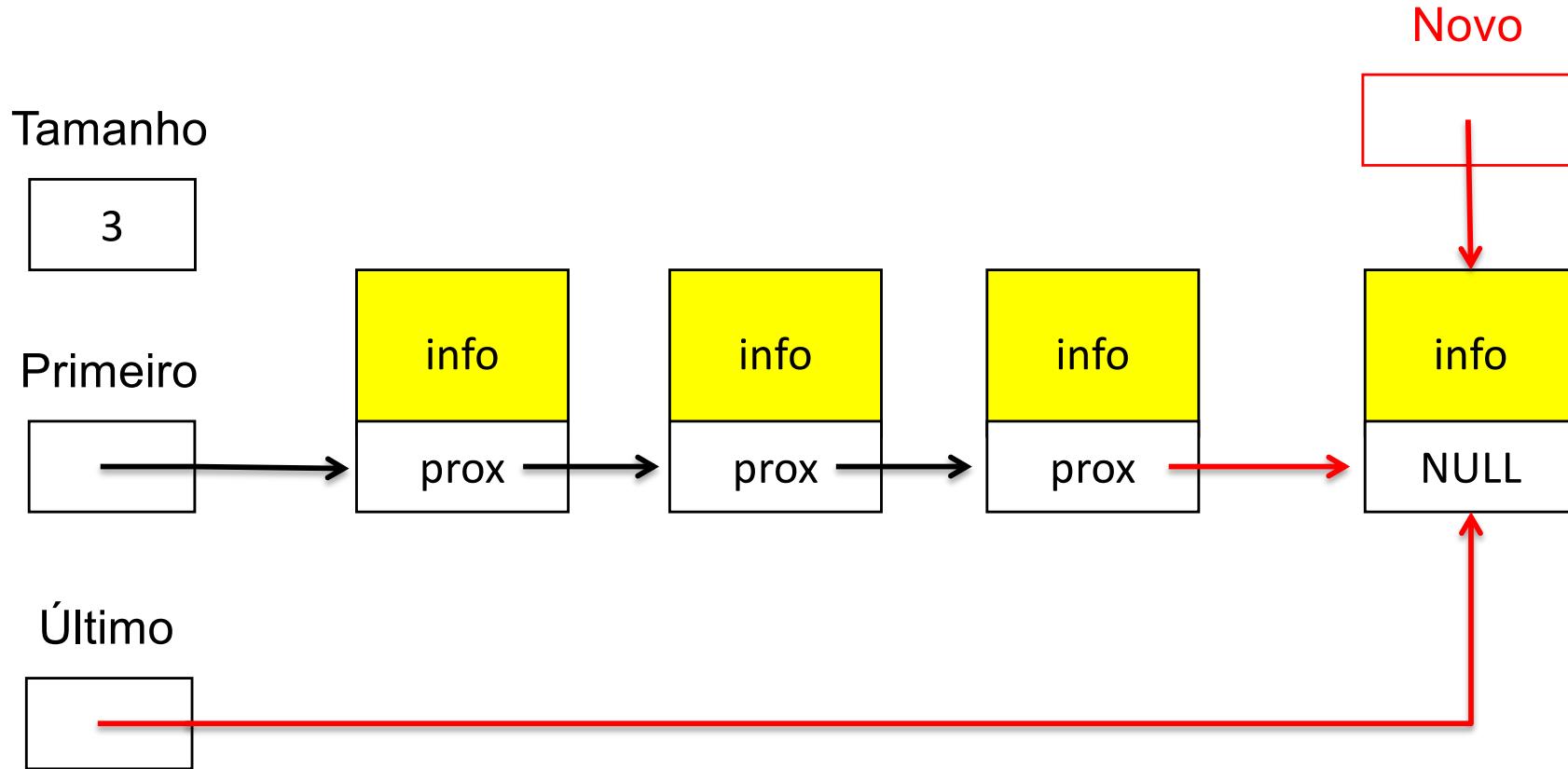
Inserção na Última Posição



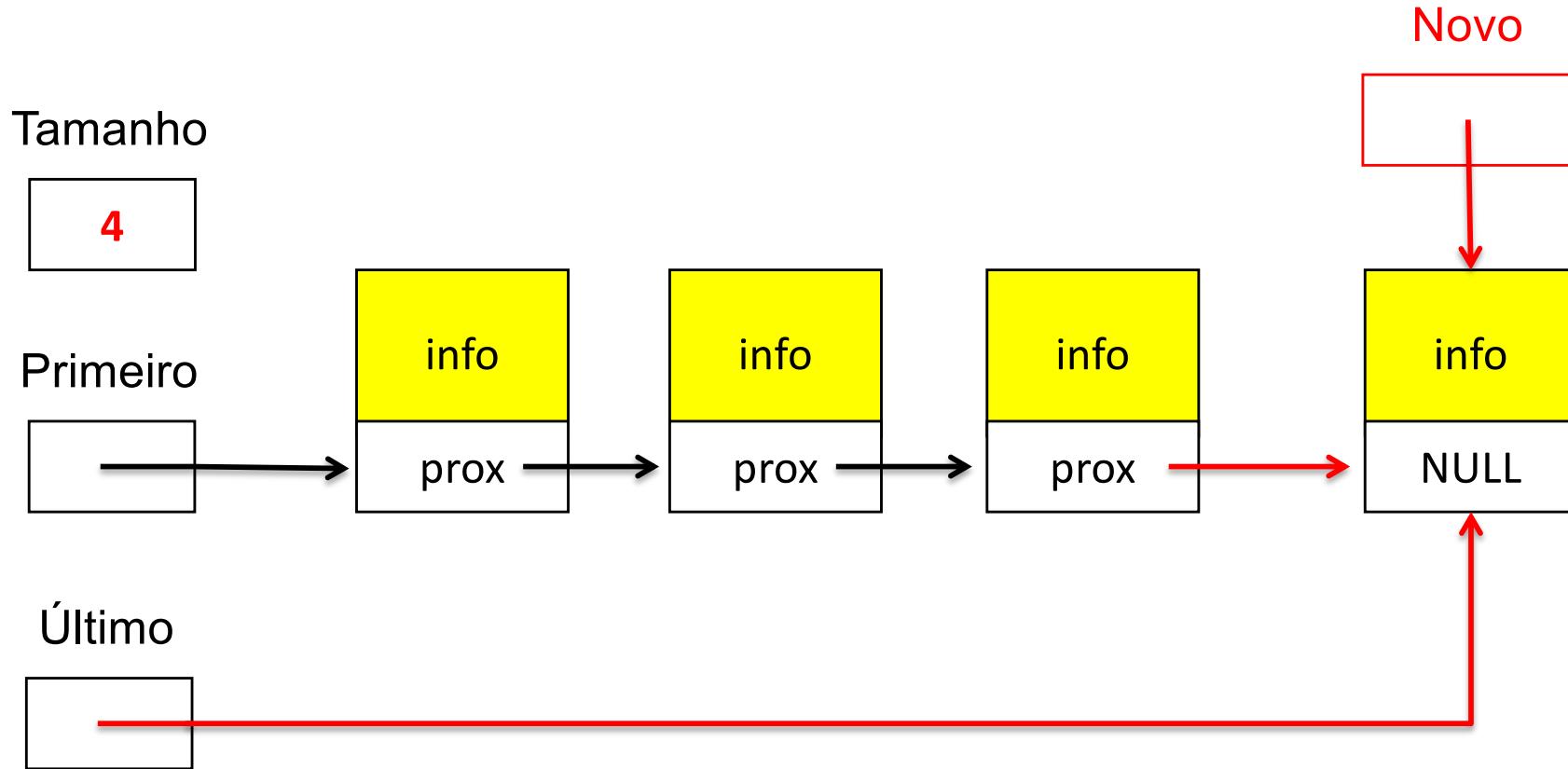
Inserção na Última Posição



Inserção na Última Posição

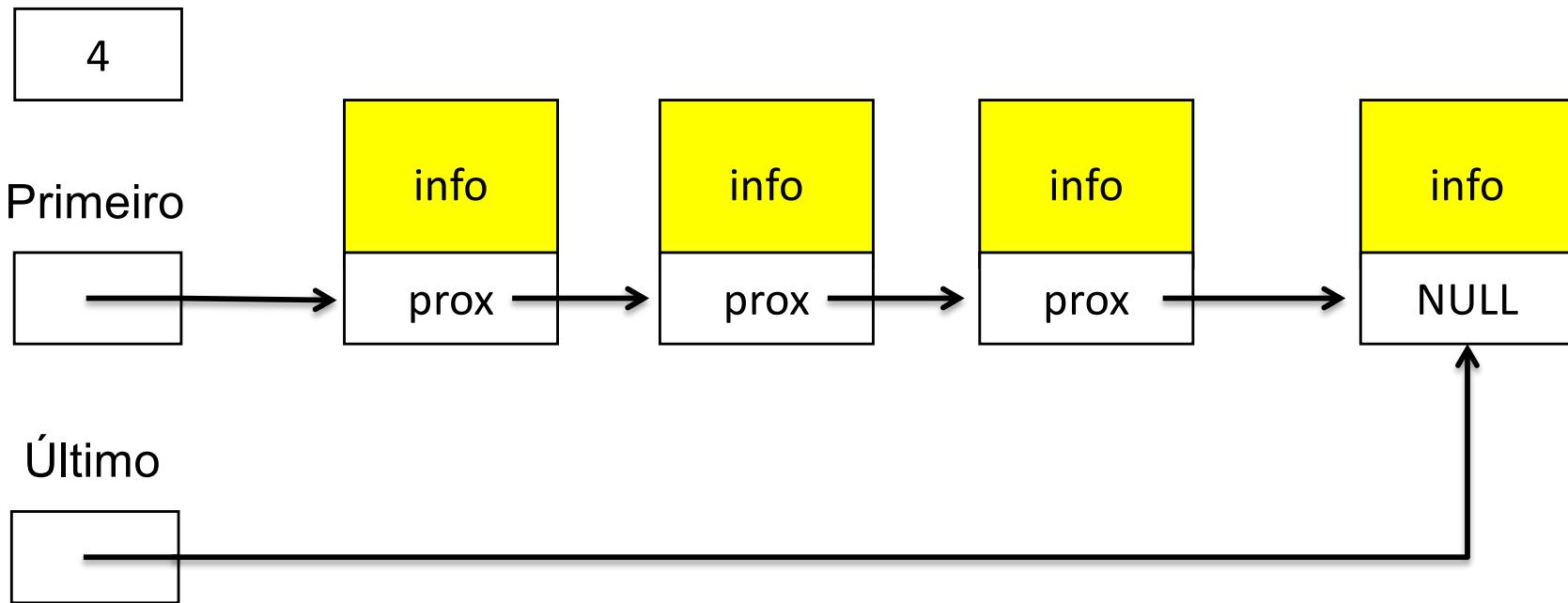


Inserção na Última Posição



Inserção na Última Posição

Tamanho

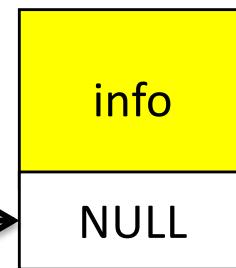
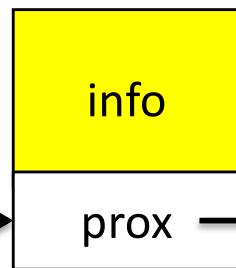
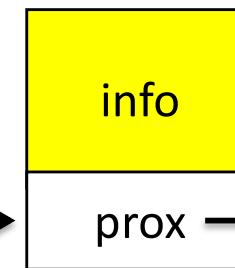
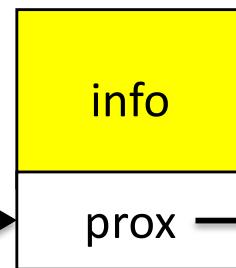


Inserção em Posição Qualquer

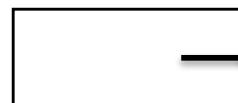
Tamanho

4

Primeiro



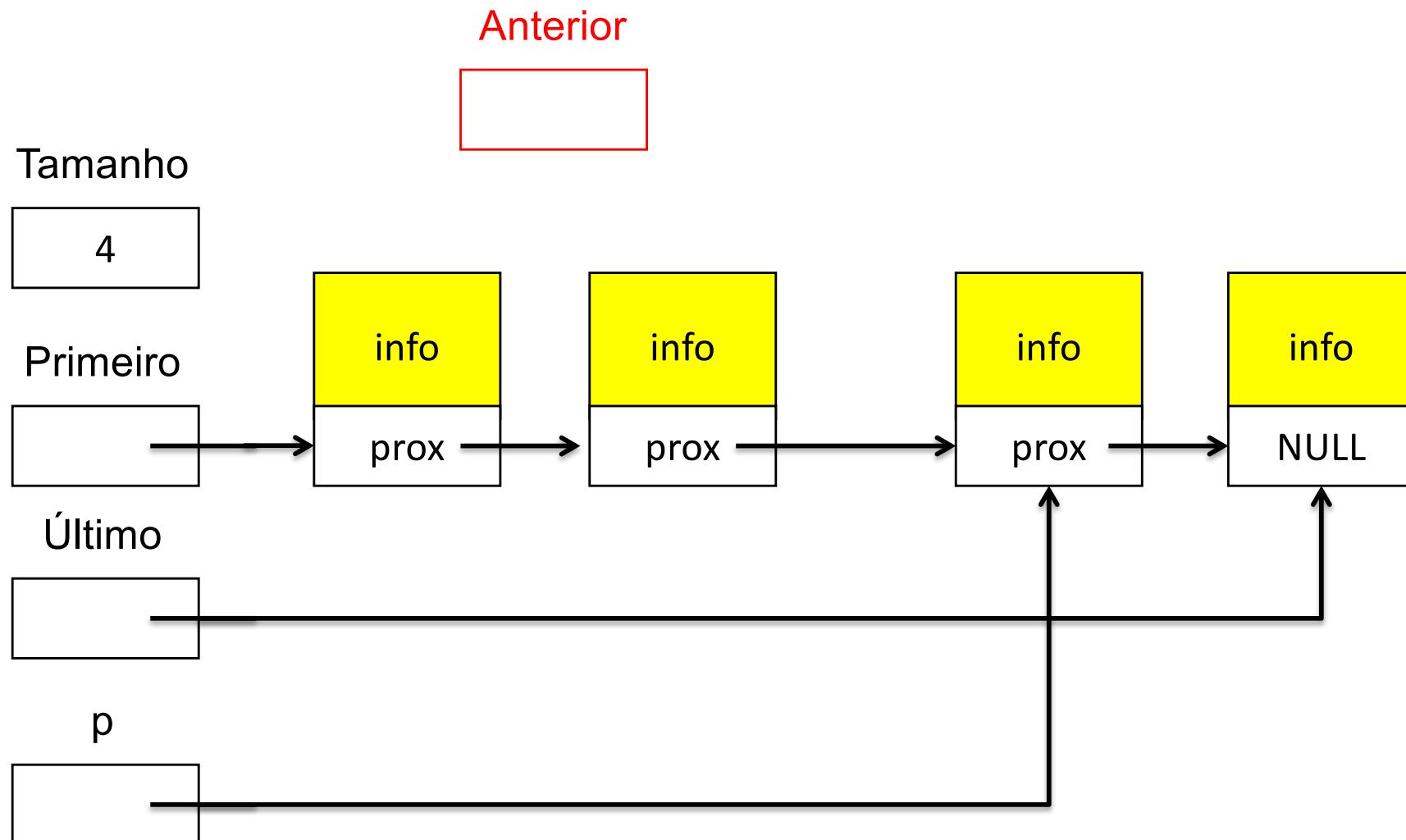
Último



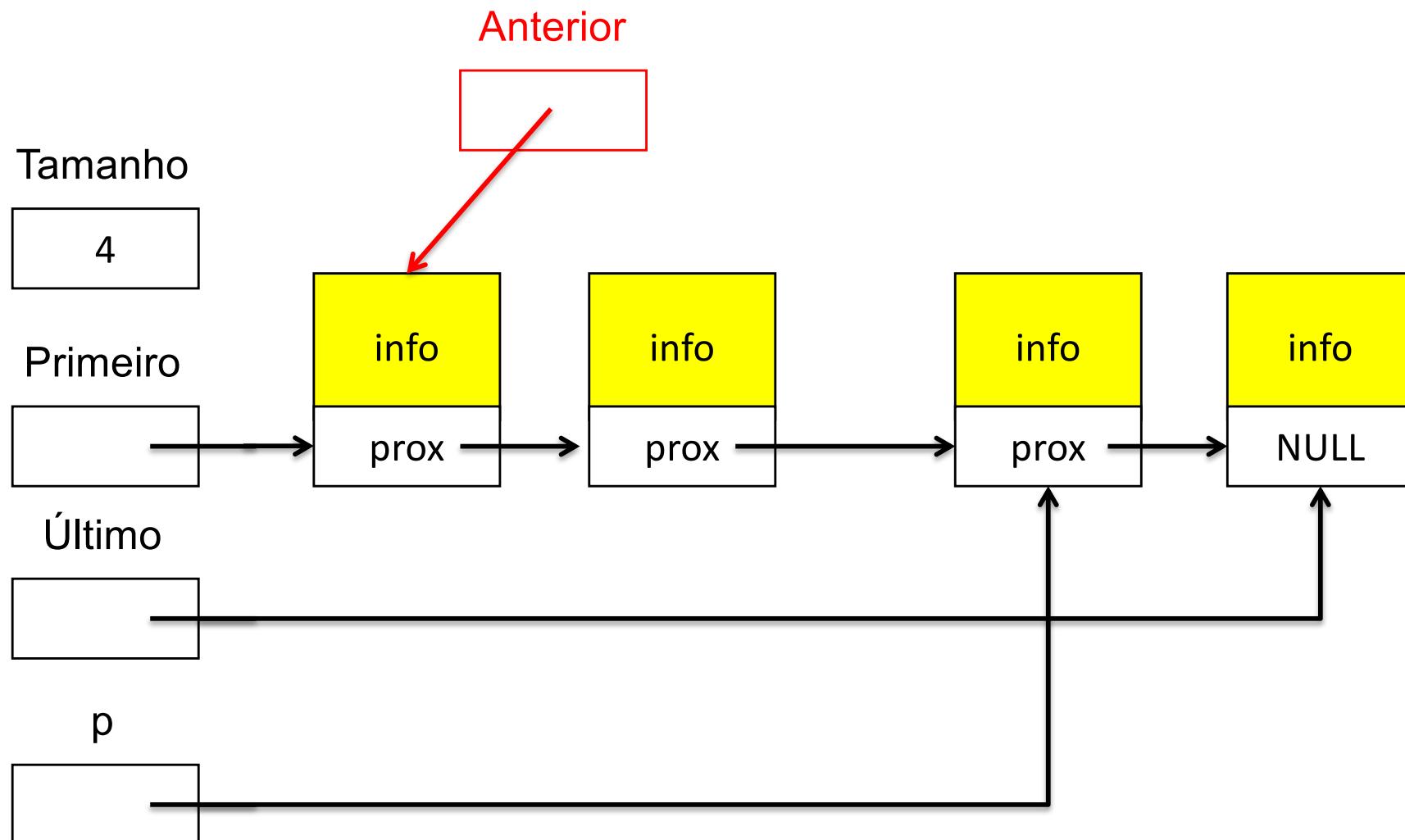
p



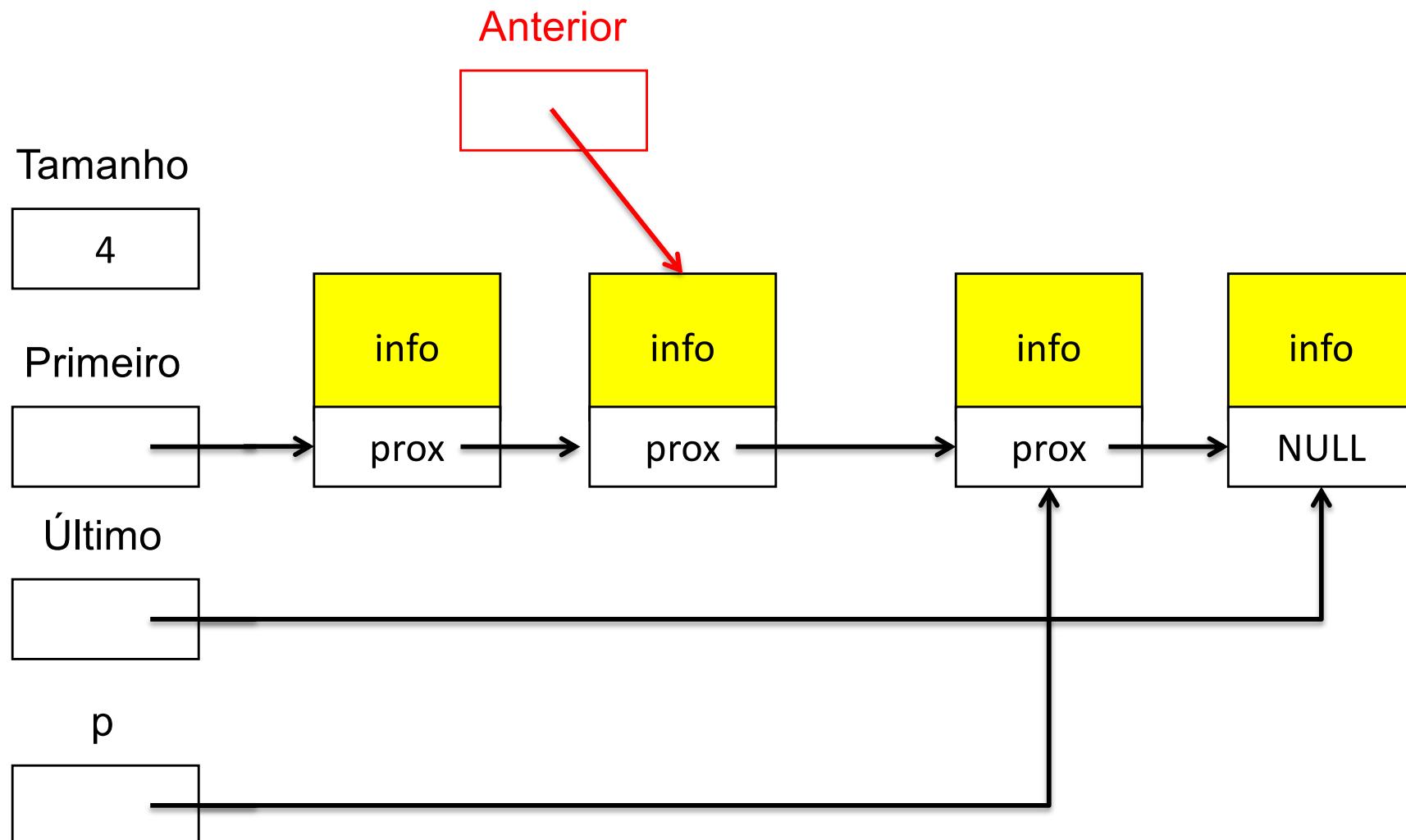
Inserção em Posição Qualquer



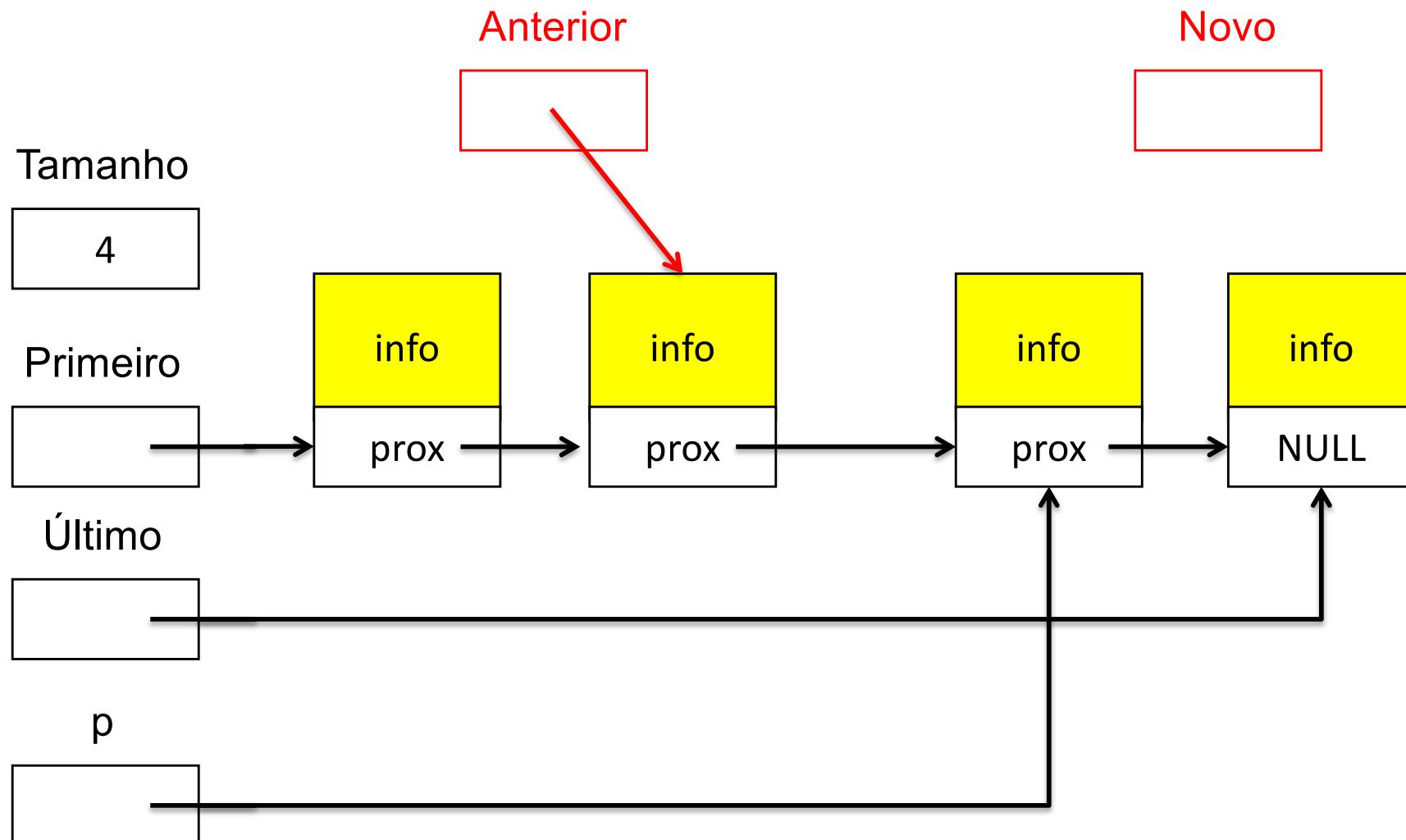
Inserção em Posição Qualquer



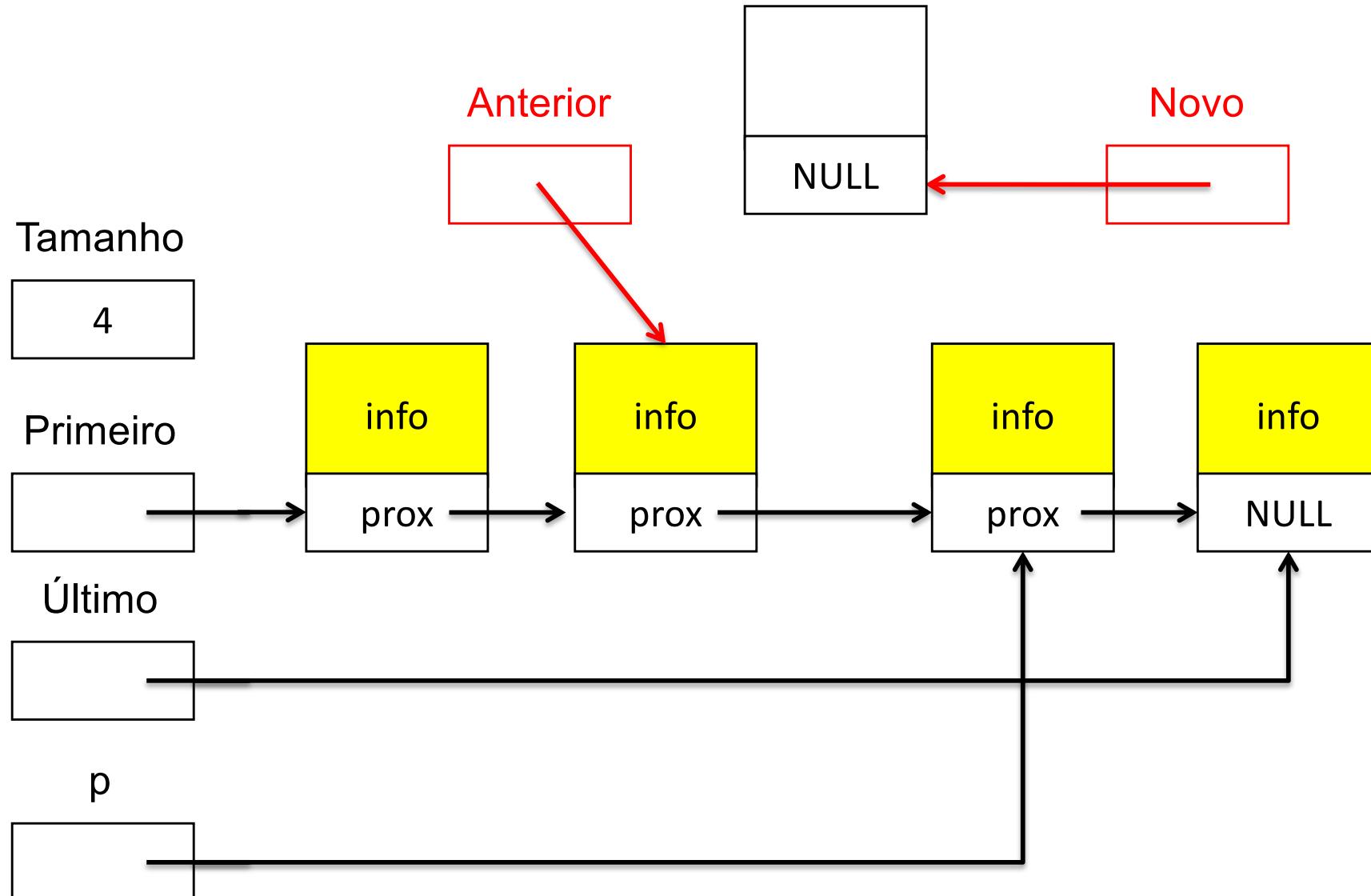
Inserção em Posição Qualquer



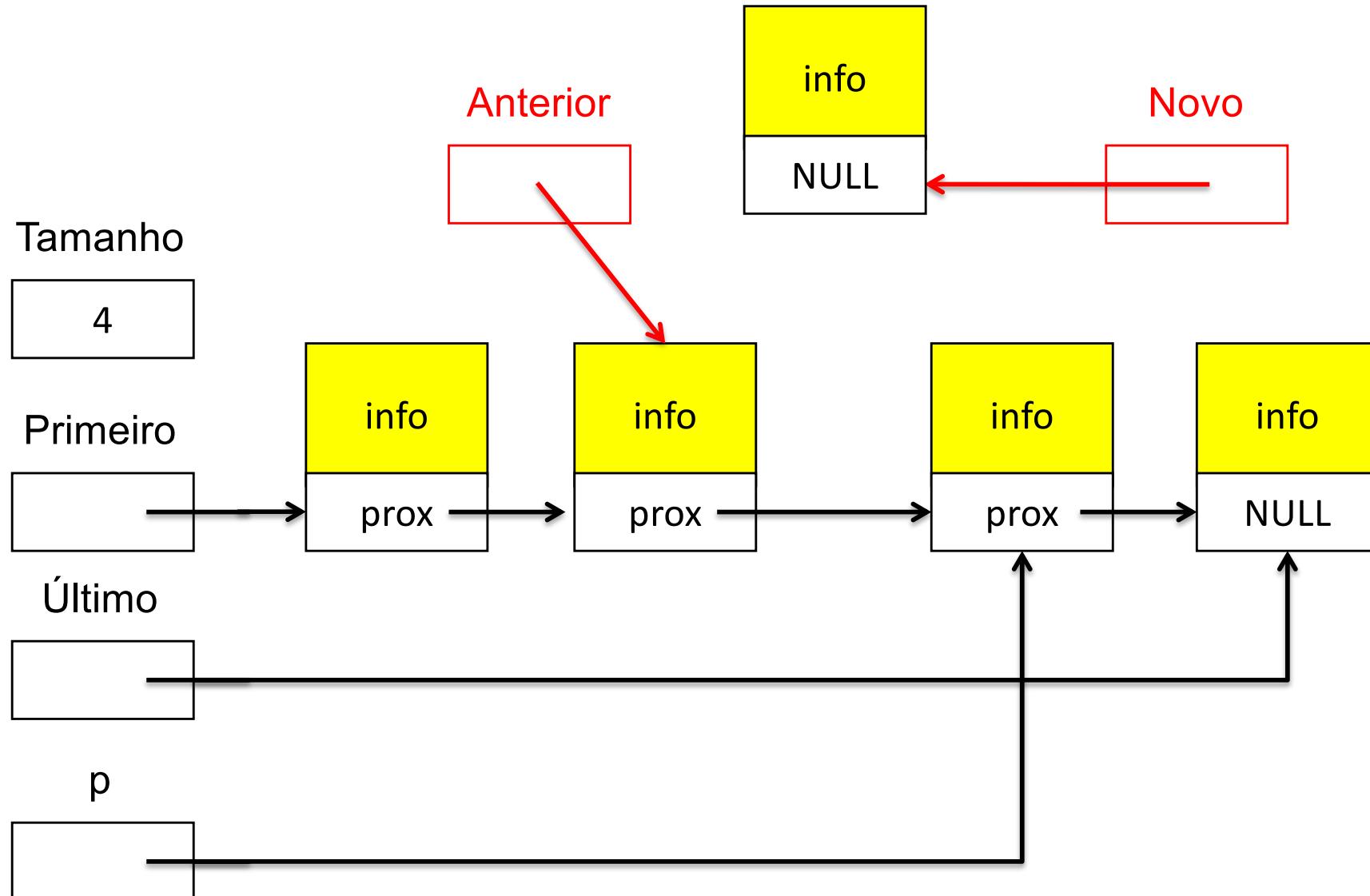
Inserção em Posição Qualquer



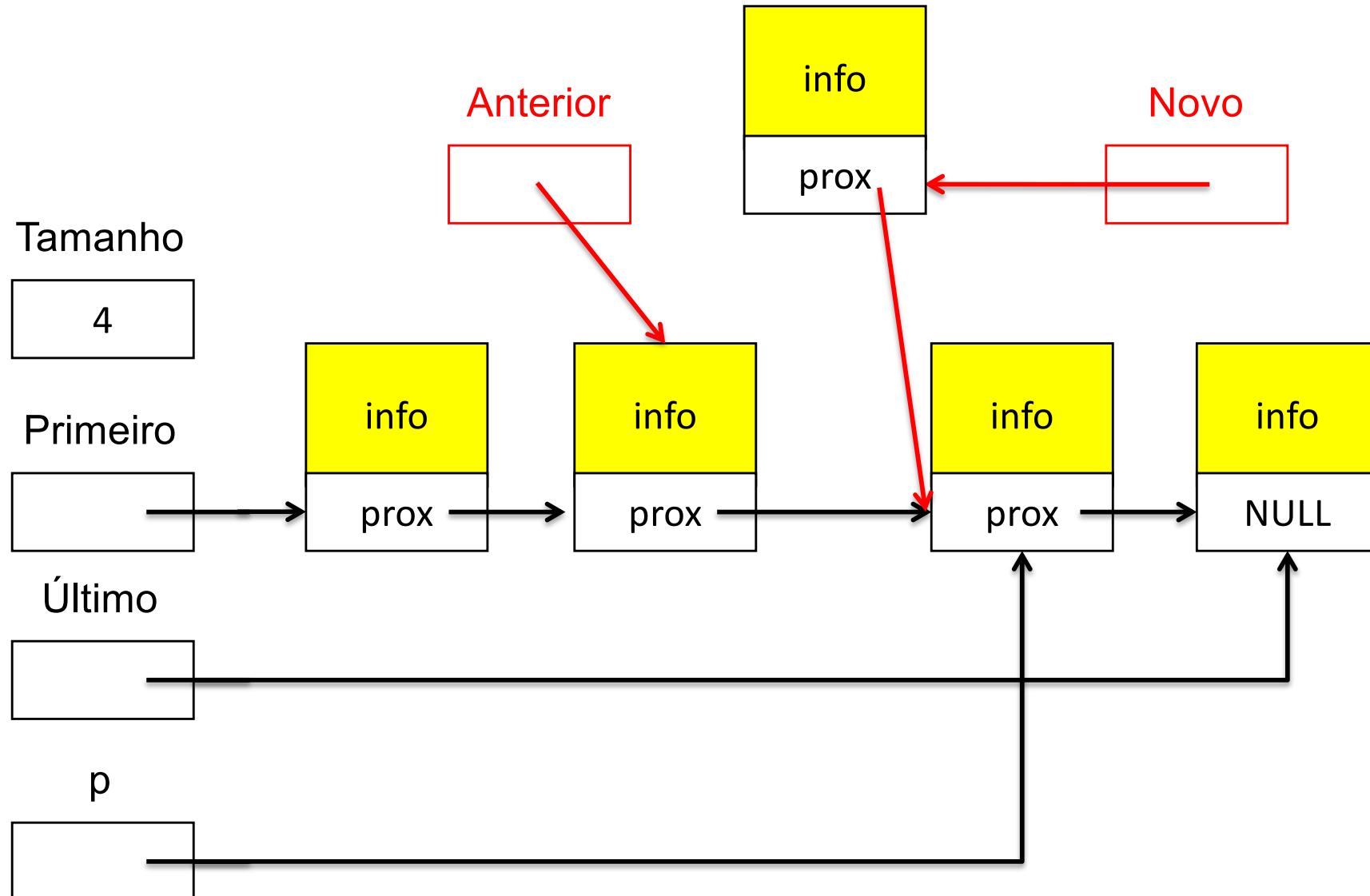
Inserção em Posição Qualquer



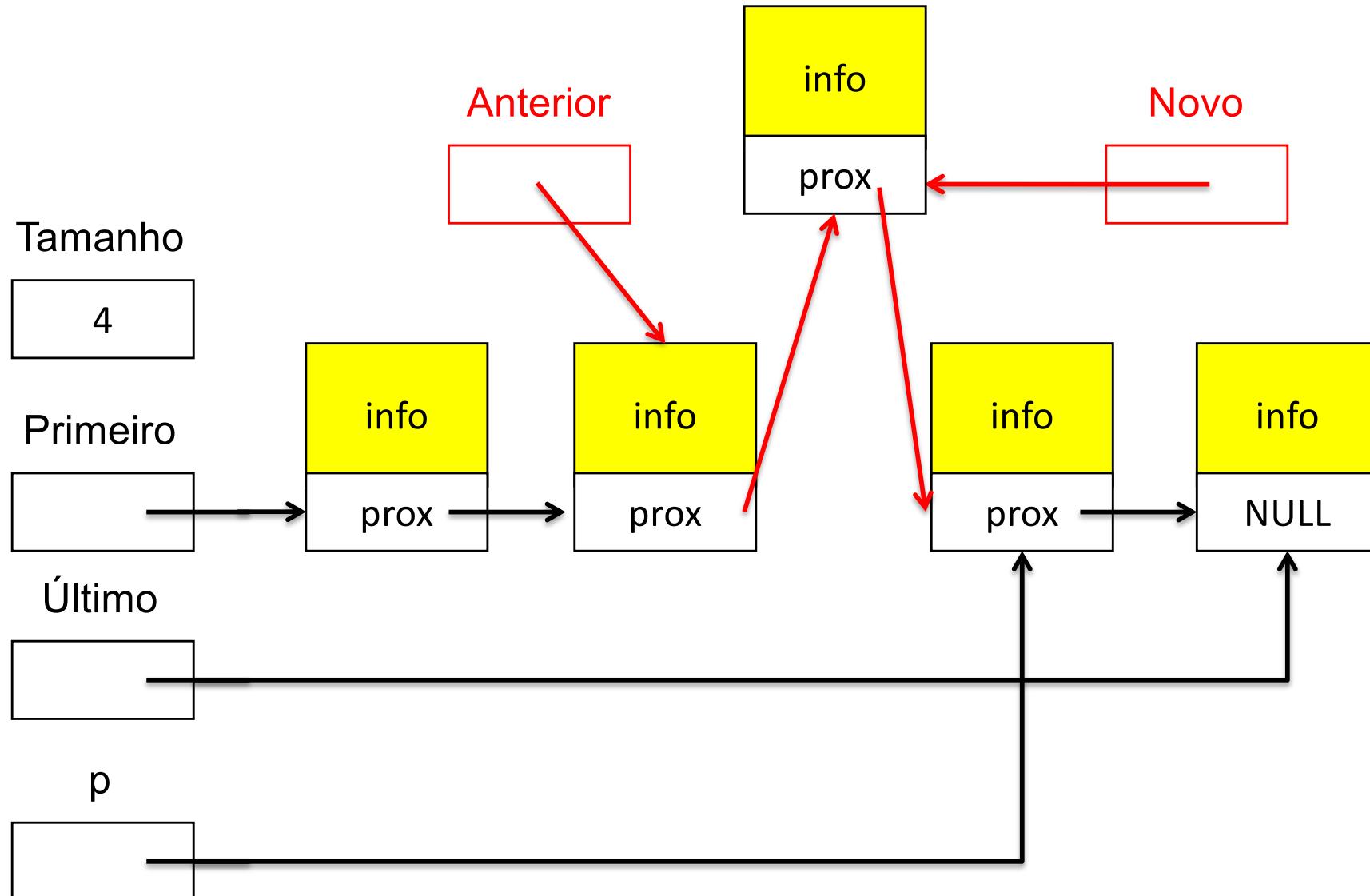
Inserção em Posição Qualquer



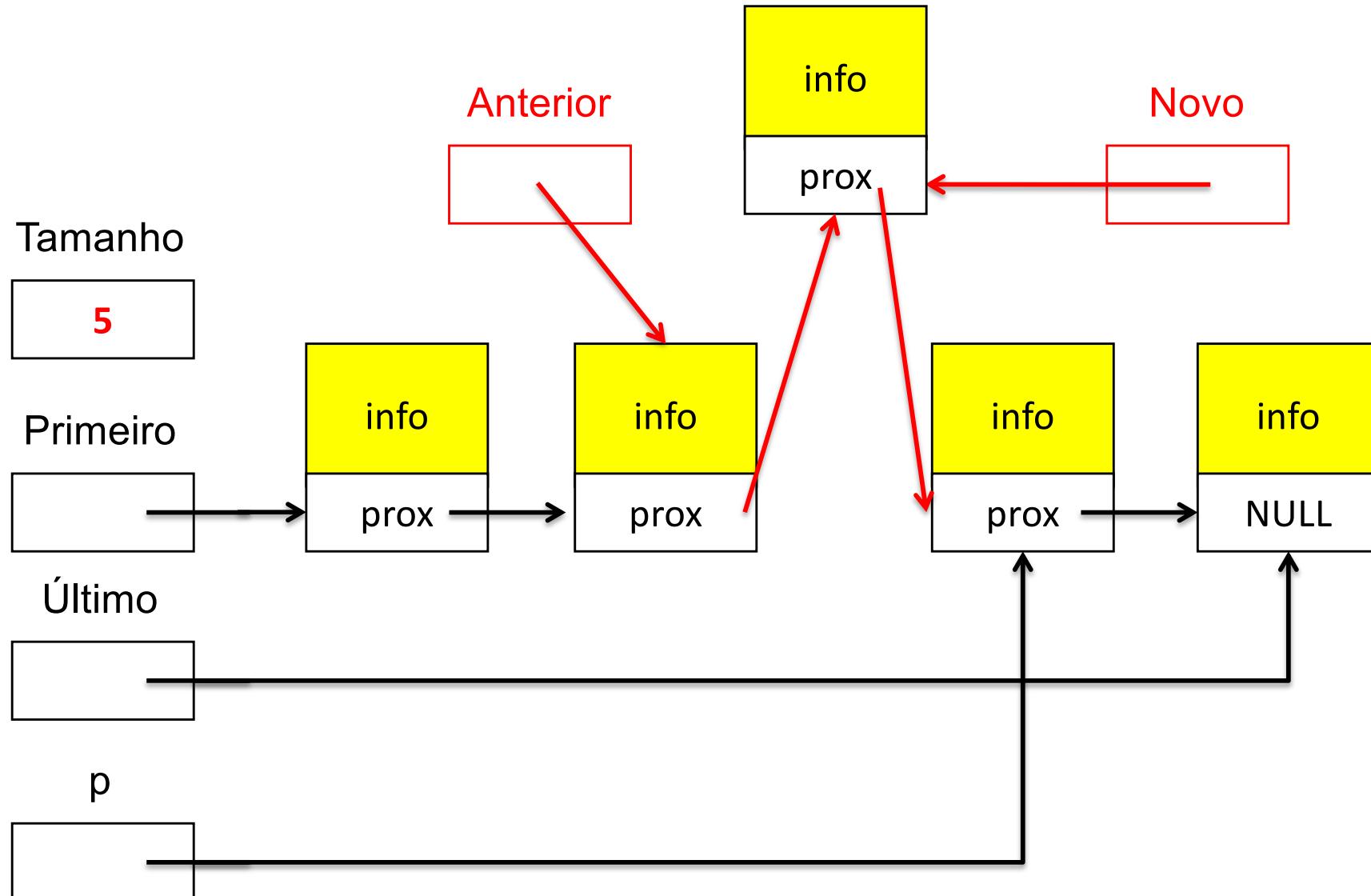
Inserção em Posição Qualquer



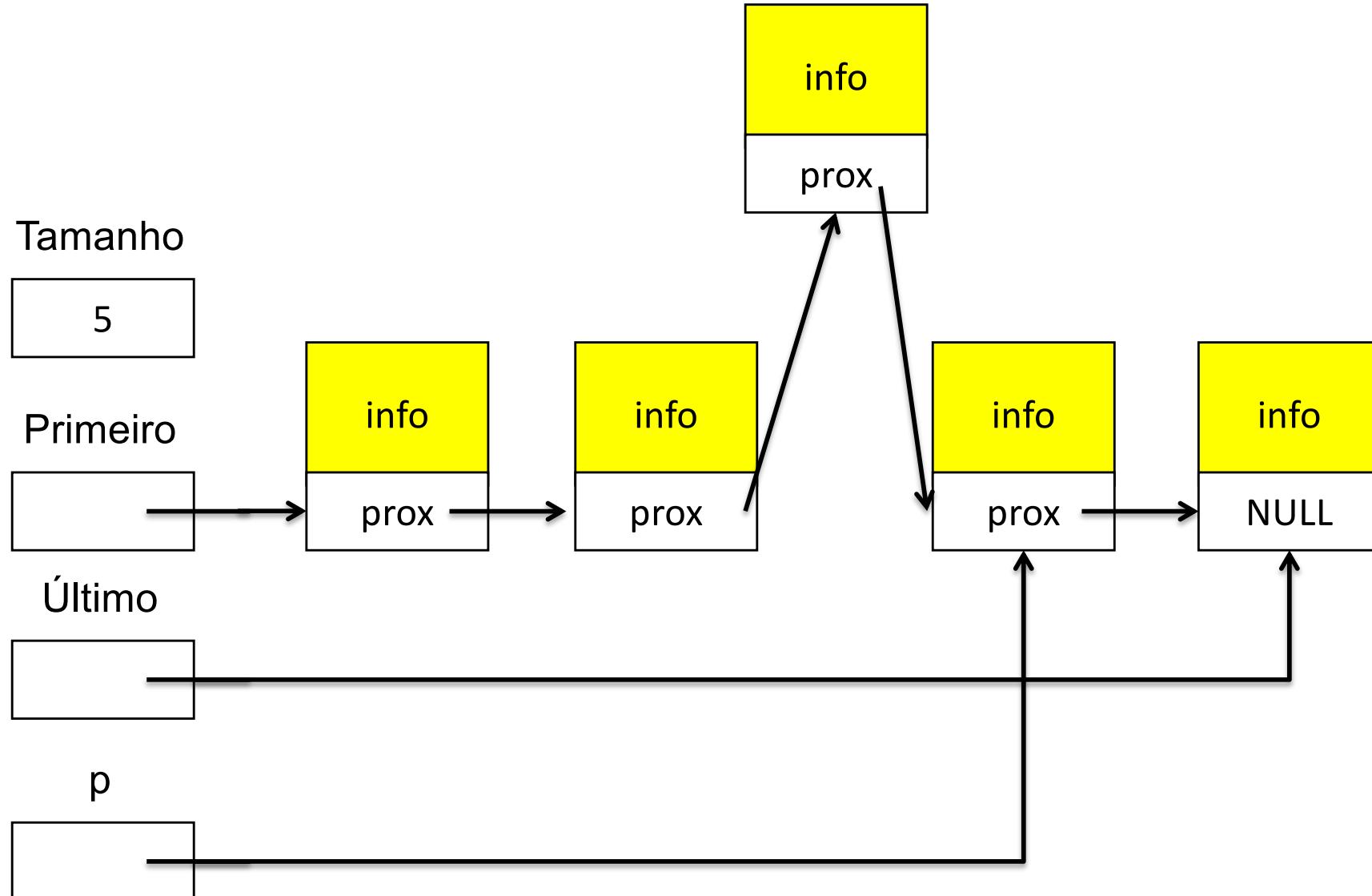
Inserção em Posição Qualquer



Inserção em Posição Qualquer



Inserção em Posição Qualquer



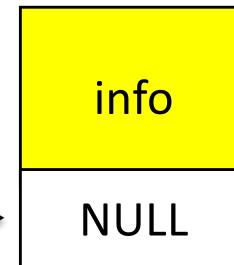
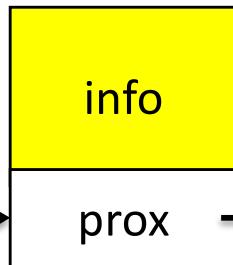
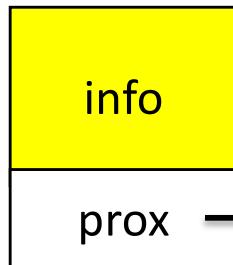
Retirada de Elementos da Lista

Tamanho

3

Primeiro

—



Último

—

- 3 opções de posição onde pode retirar:
 - 1^a posição
 - última posição
 - uma posição qualquer

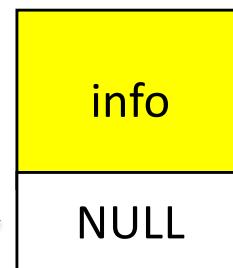
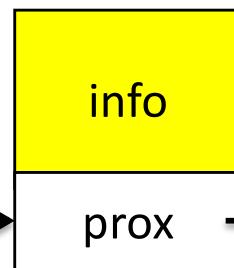
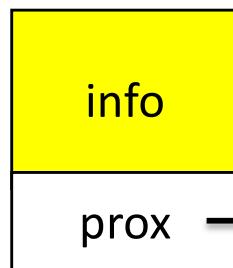
Retirada da Primeira Posição

Tamanho

3

Primeiro

—→



Último

—→

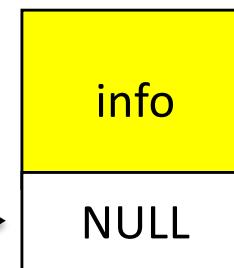
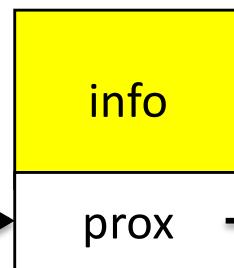
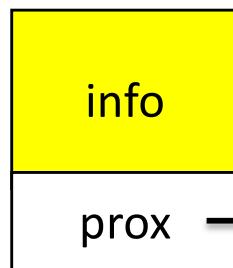
x

Retirada da Primeira Posição

Tamanho

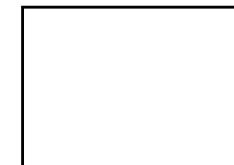
3

Primeiro

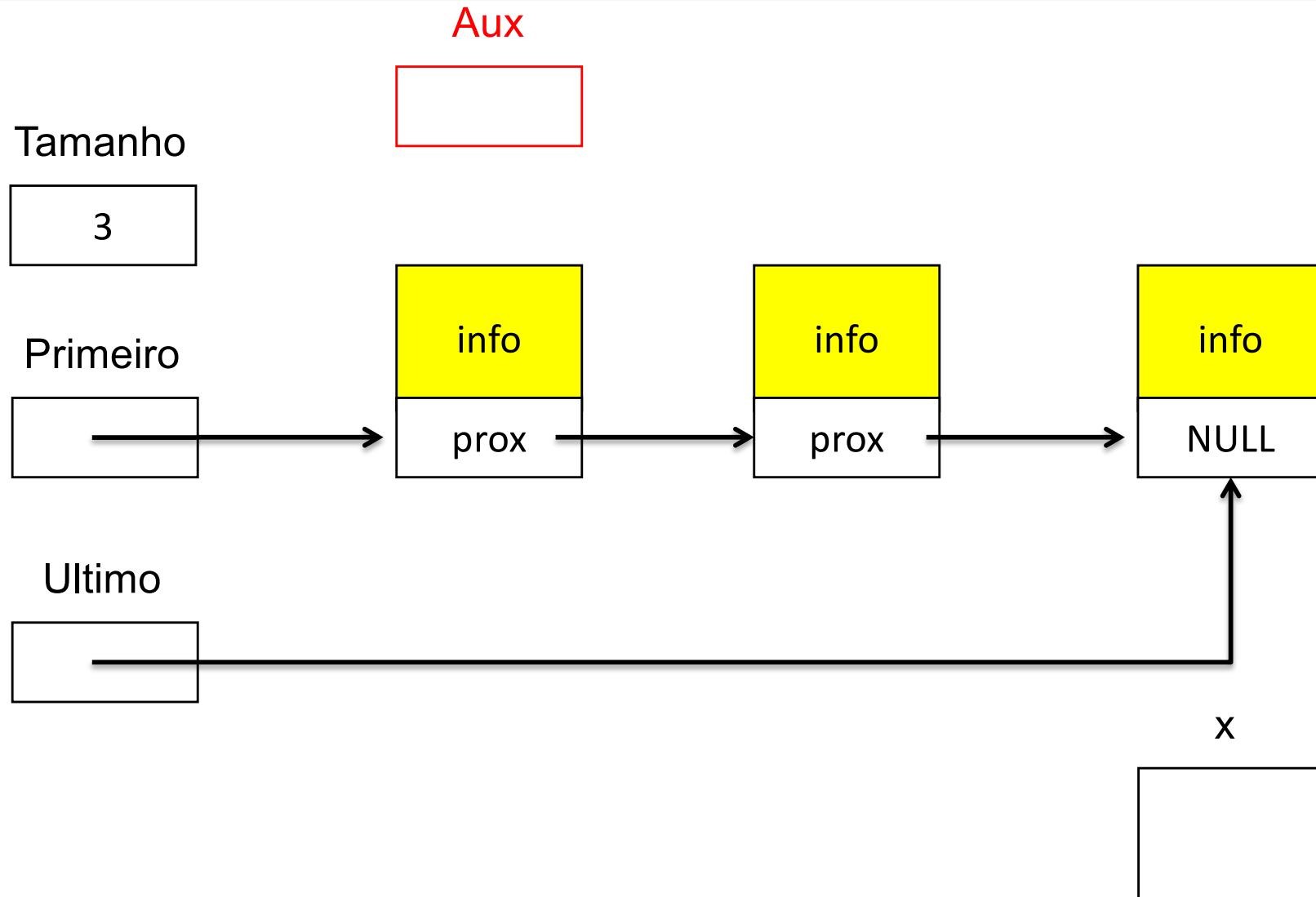


Último

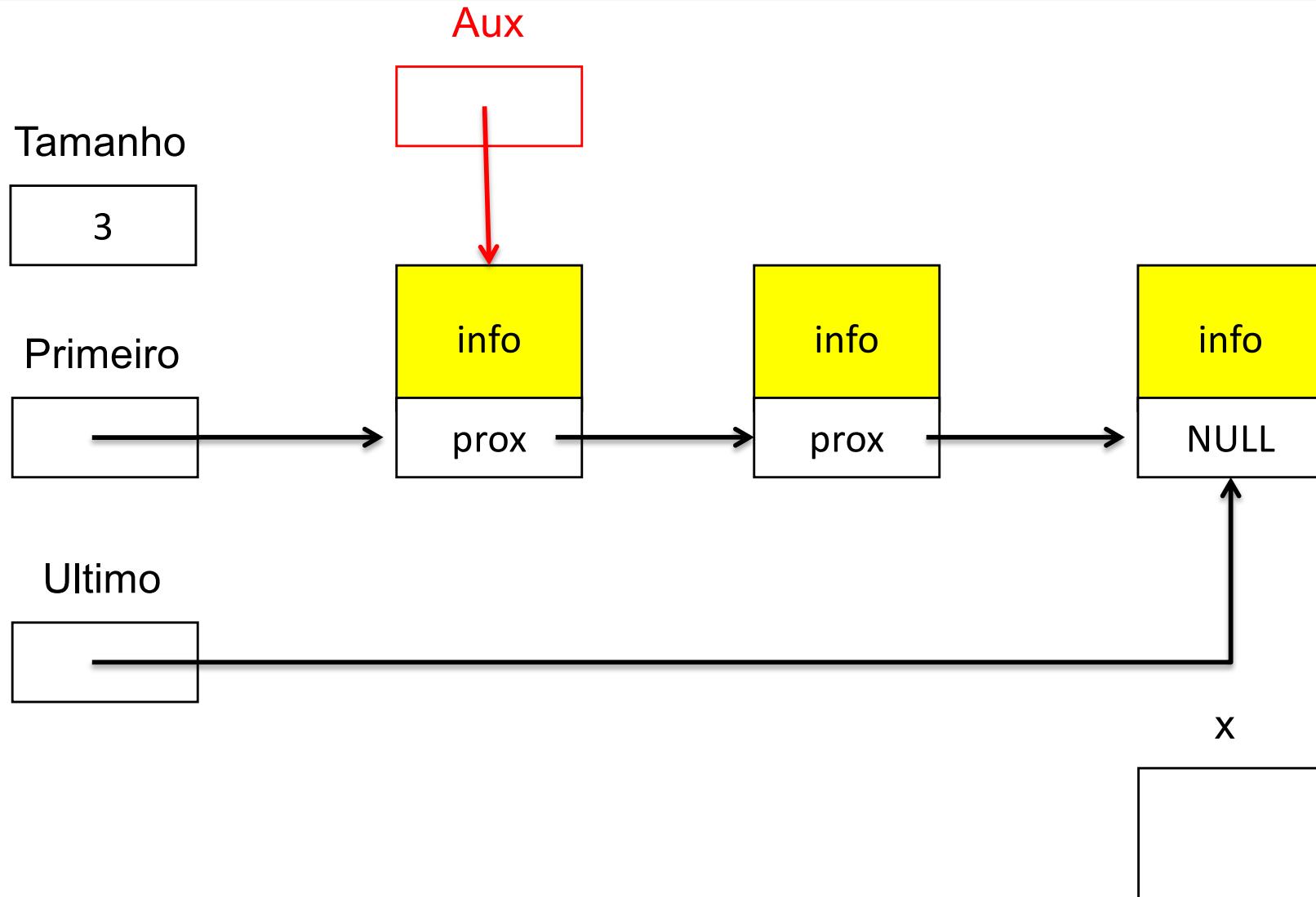
A lista está vazia?



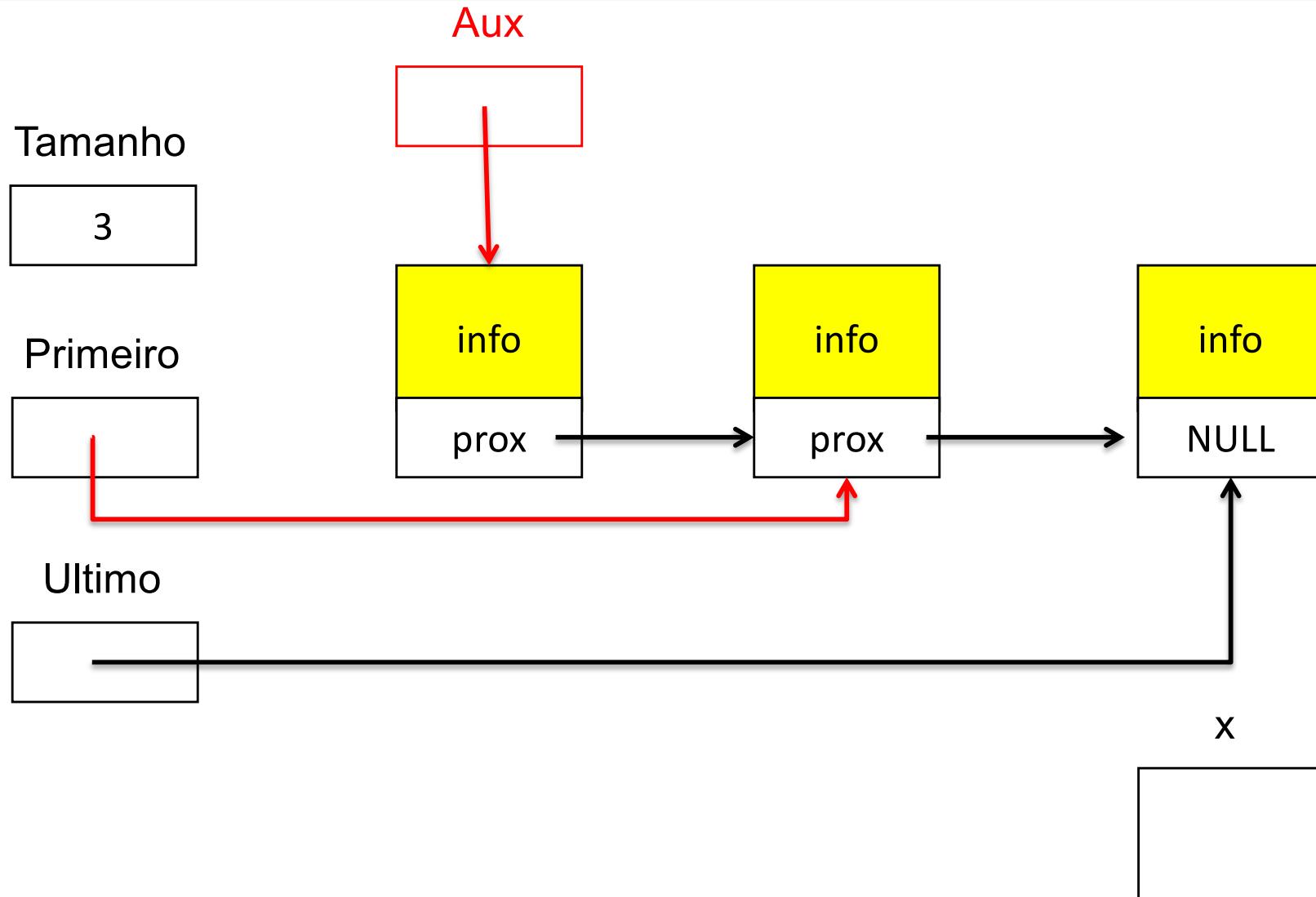
Retirada da Primeira Posição



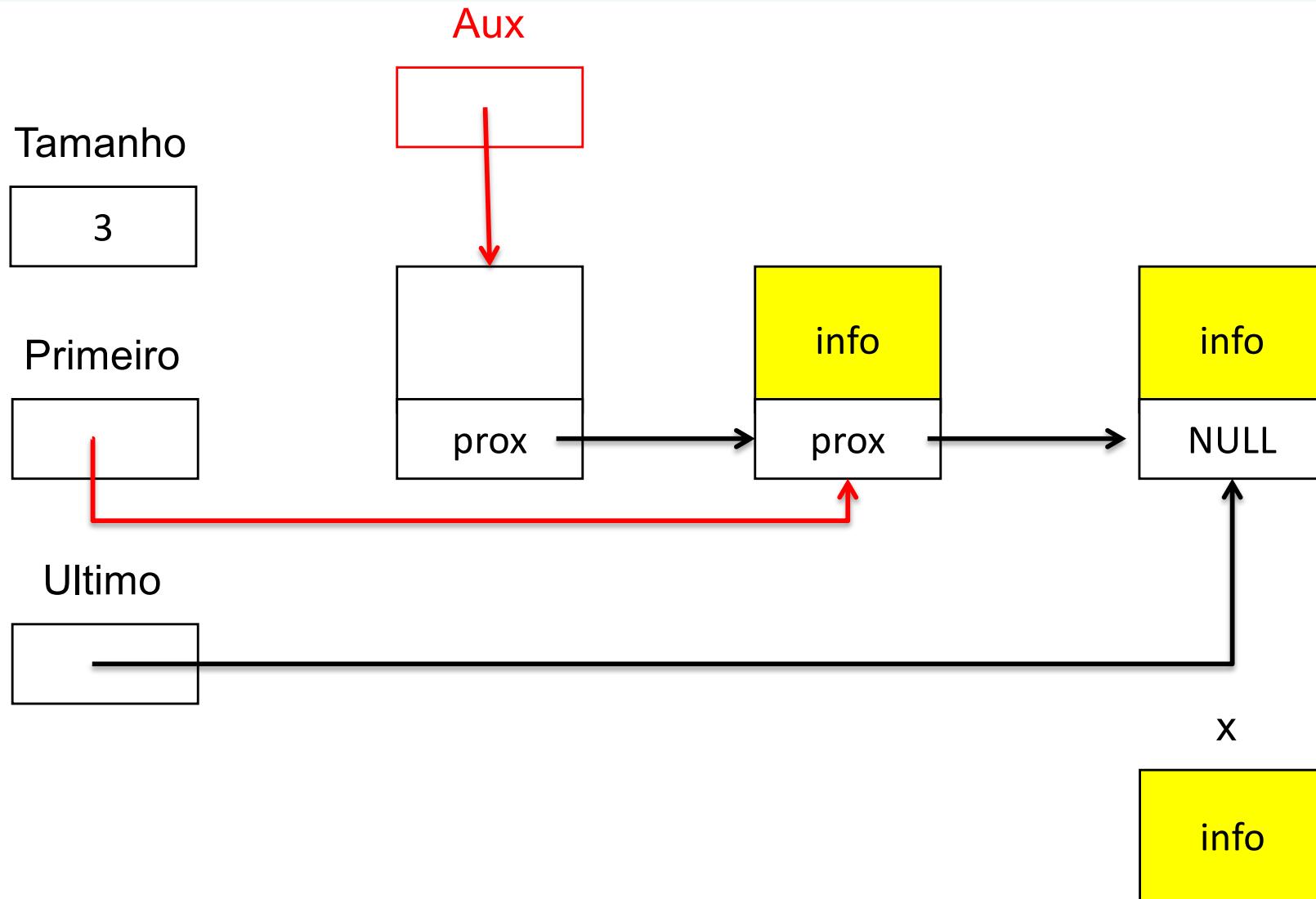
Retirada da Primeira Posição



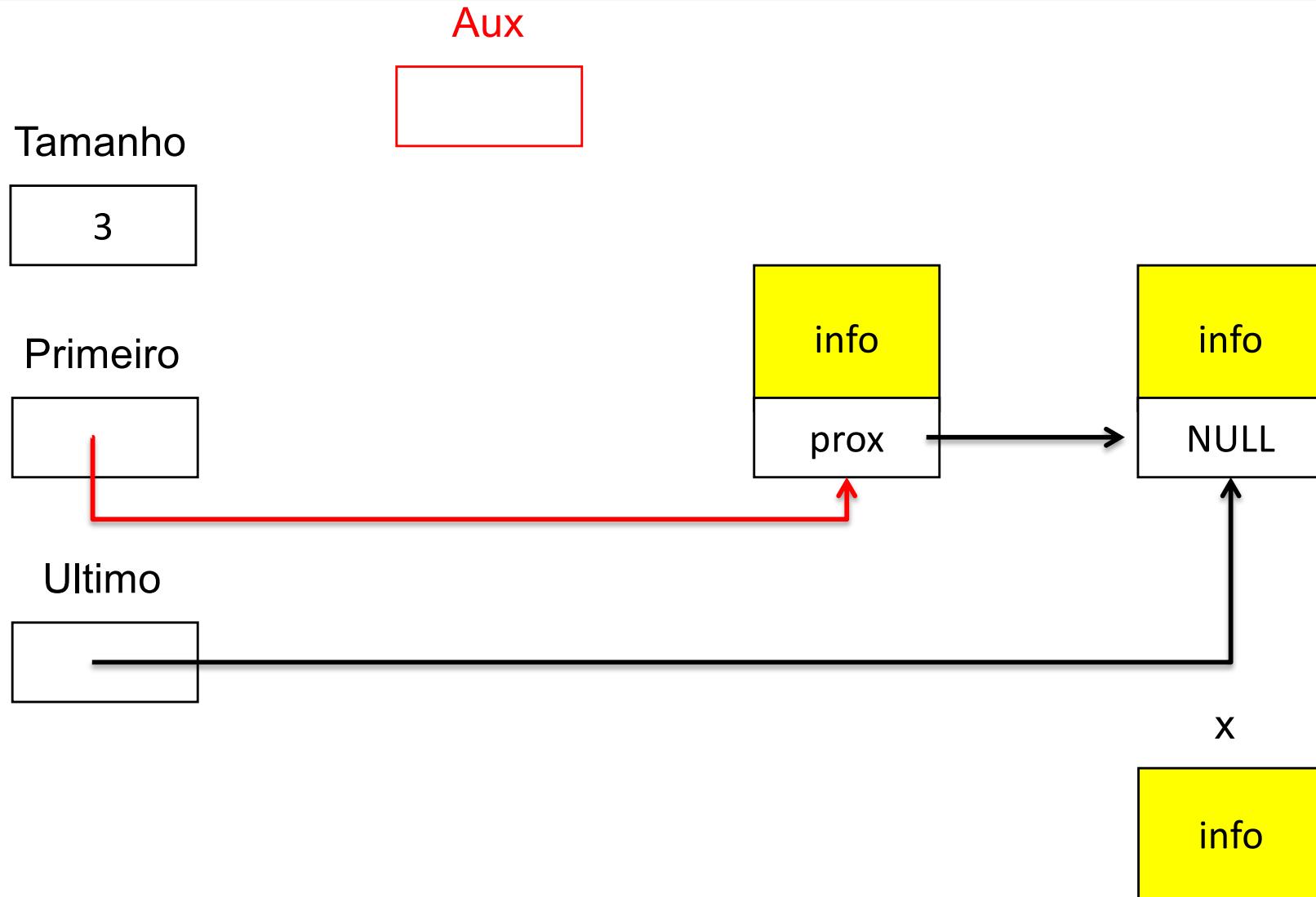
Retirada da Primeira Posição



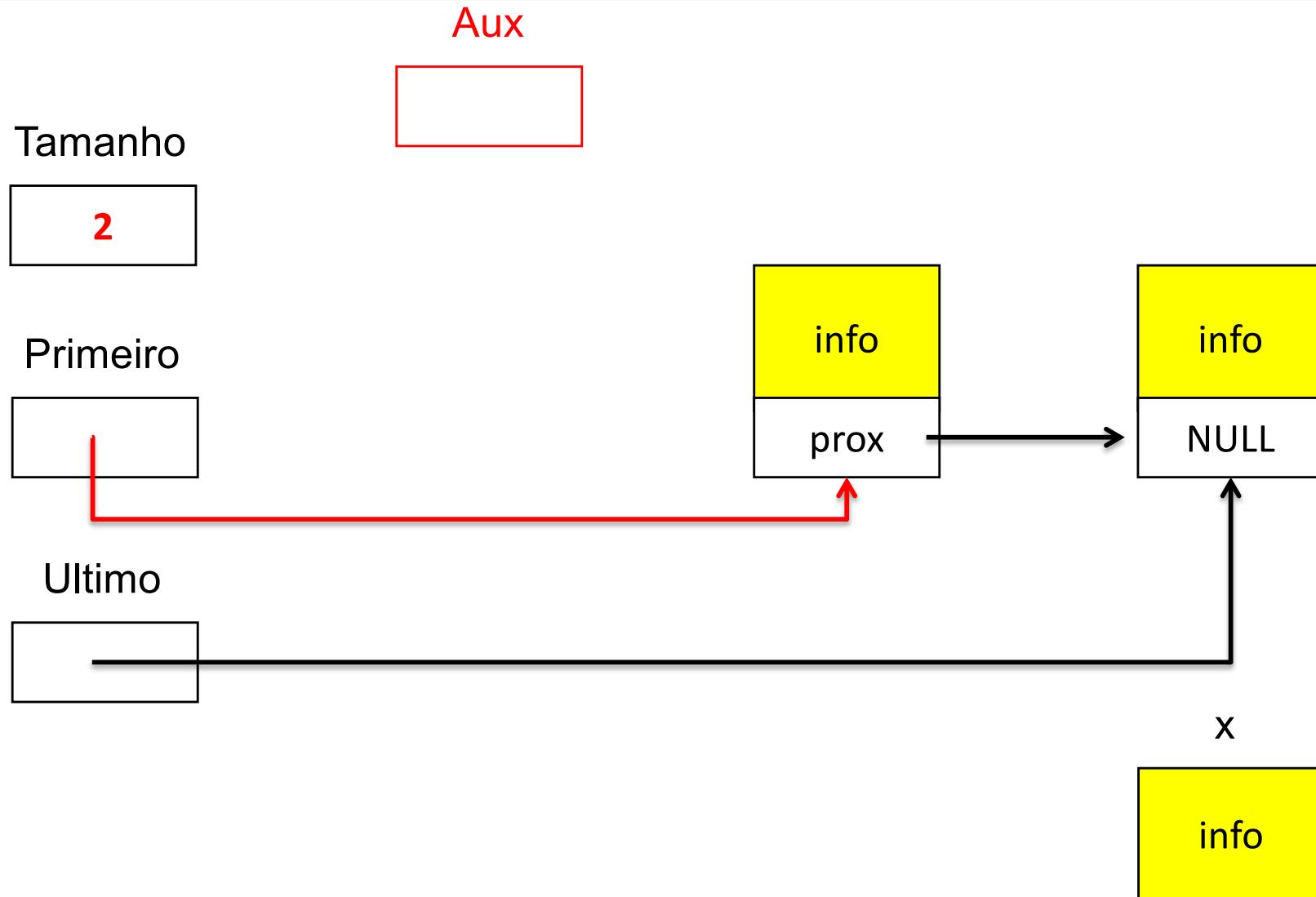
Retirada da Primeira Posição



Retirada da Primeira Posição



Retirada da Primeira Posição

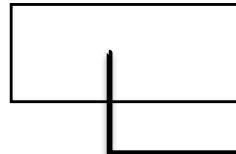


Retirada da Primeira Posição

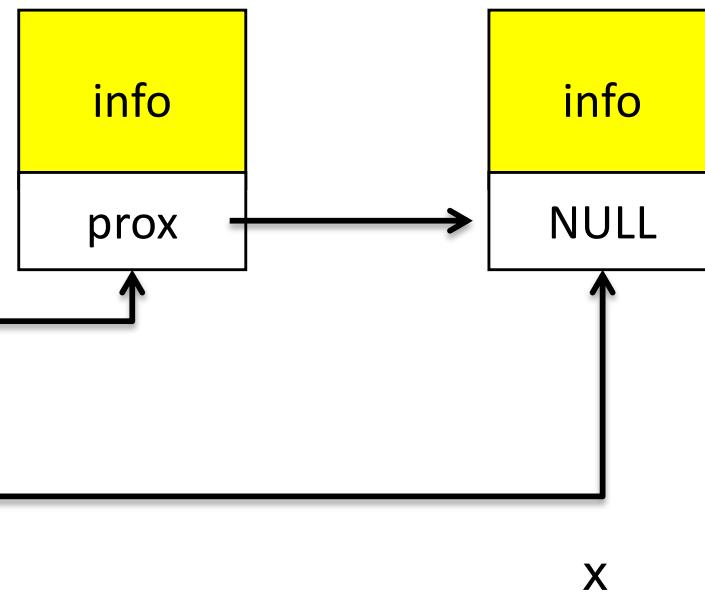
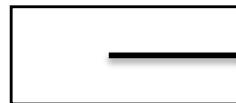
Tamanho

2

Primeiro



Último

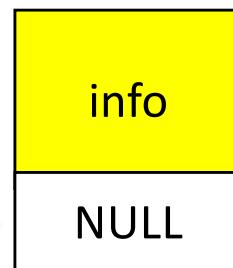
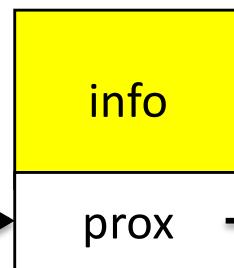
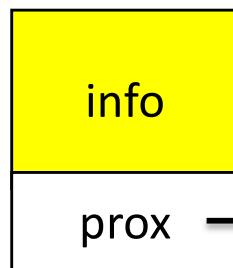


Retirada da Última Posição

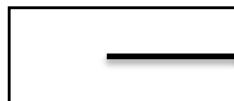
Tamanho

3

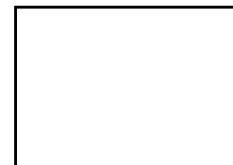
Primeiro



Último



X

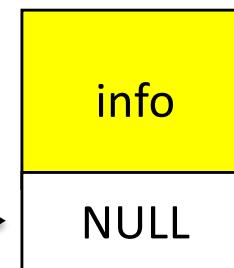
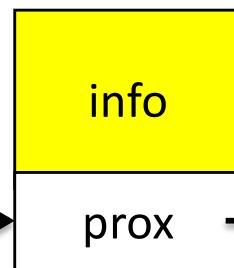
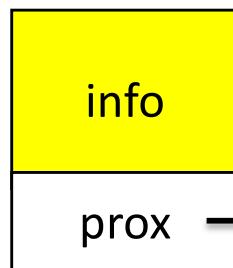


Retirada da Última Posição

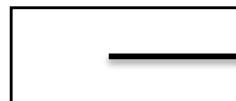
Tamanho

3

Primeiro

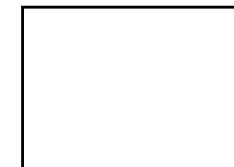


Último



x

A lista está vazia?

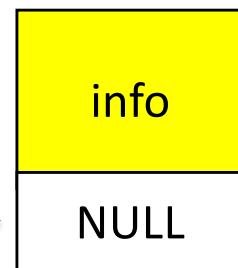
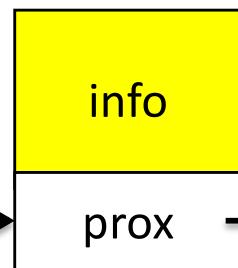
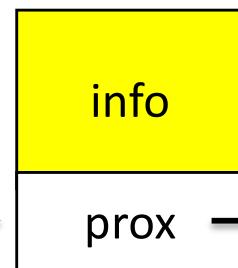


Retirada da Última Posição

Tamanho

3

Primeiro



Último

Anterior

x

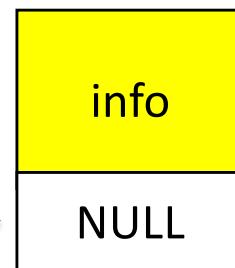
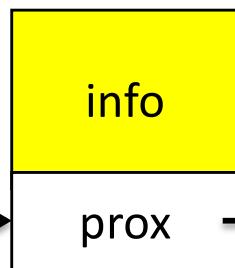
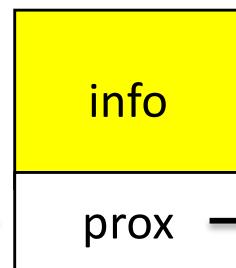
Retirada da Última Posição

Tamanho

3

Primeiro

—→



Último

—→



Anterior

—→

x

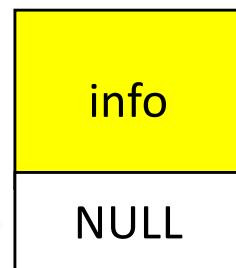
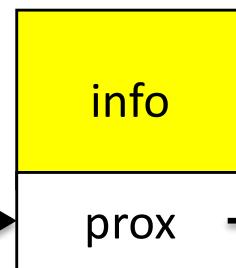
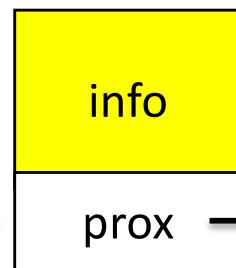
Retirada da Última Posição

Tamanho

3

Primeiro

—→



Último

—→

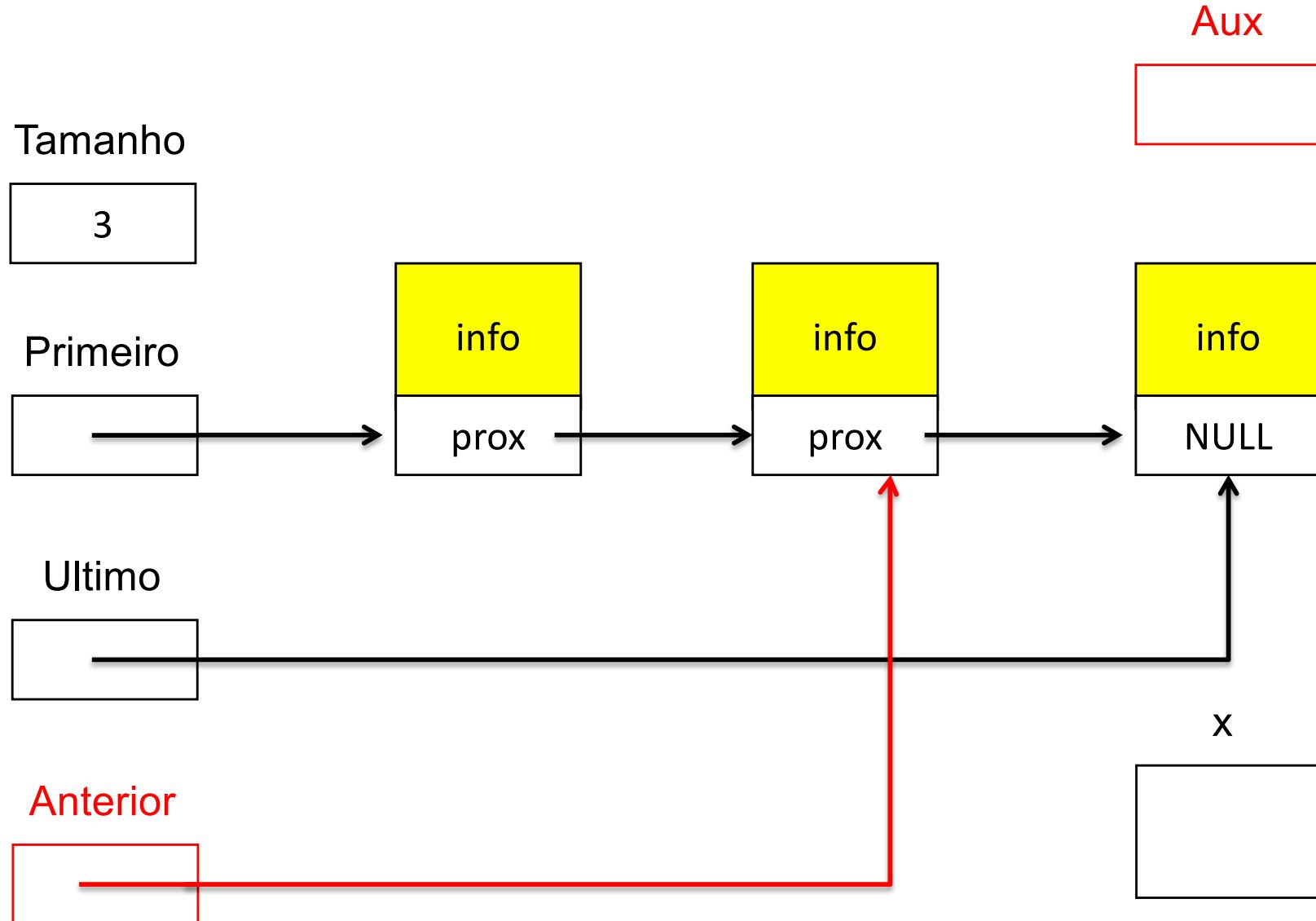
x

Anterior

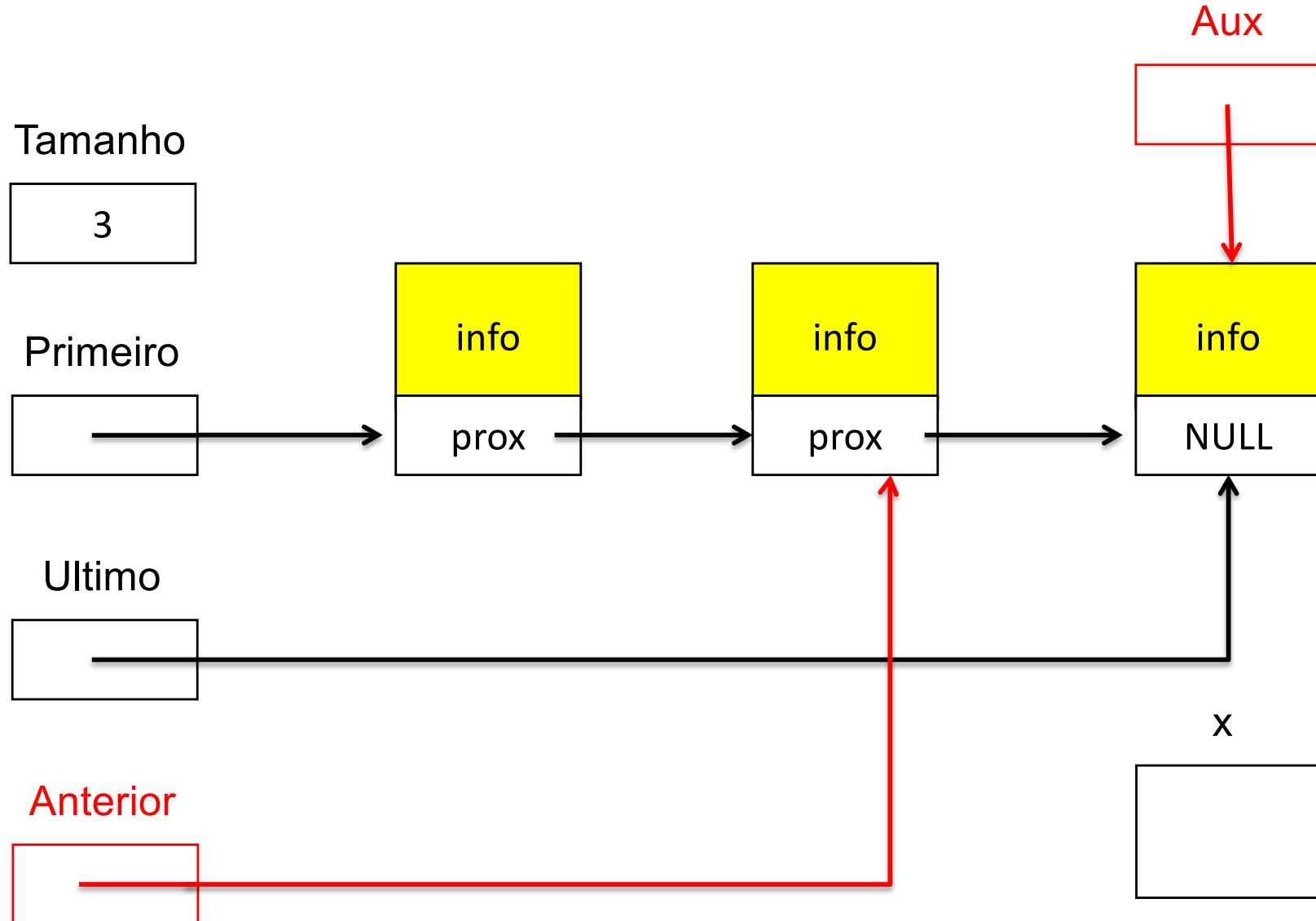
—→

X

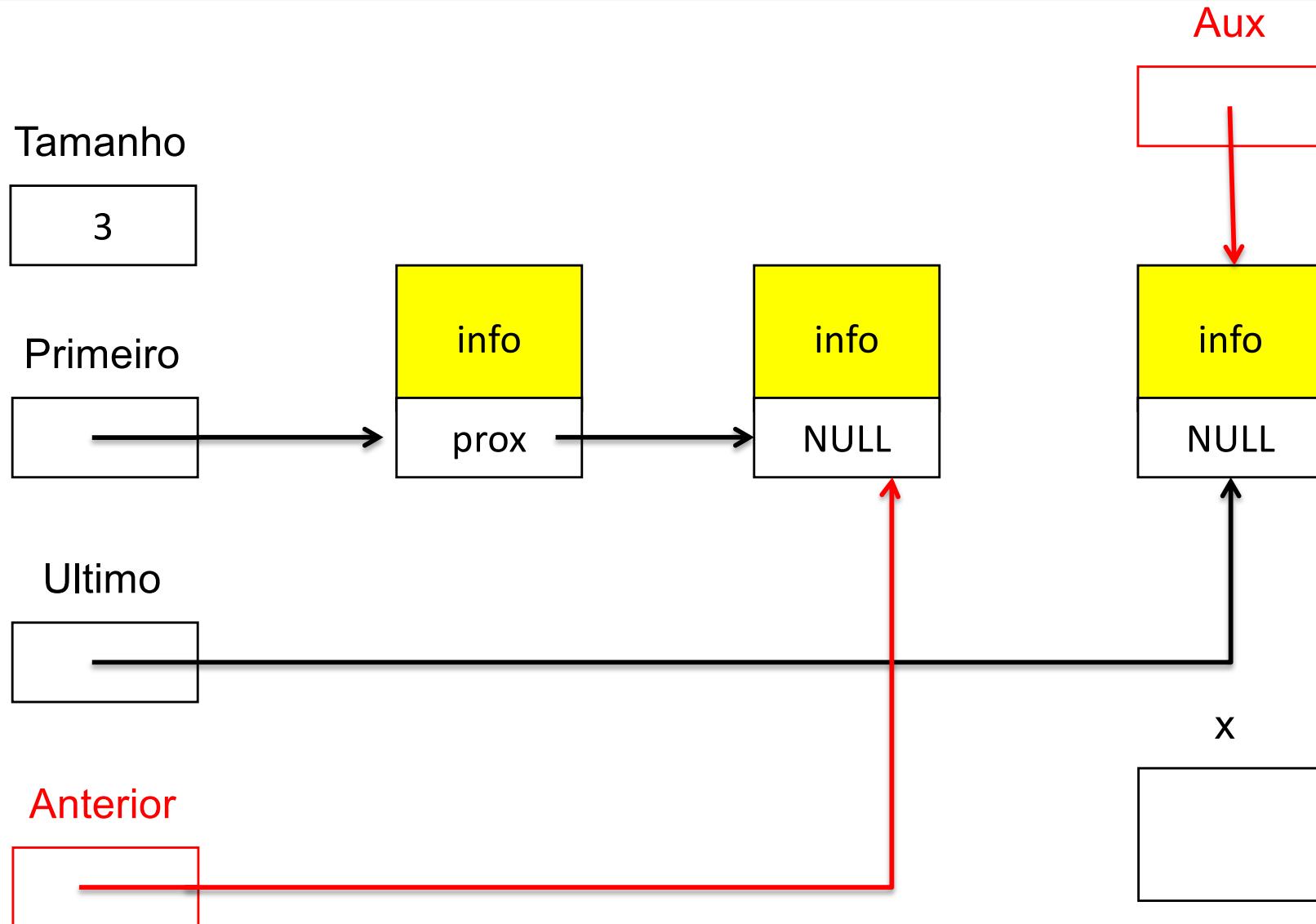
Retirada da Última Posição



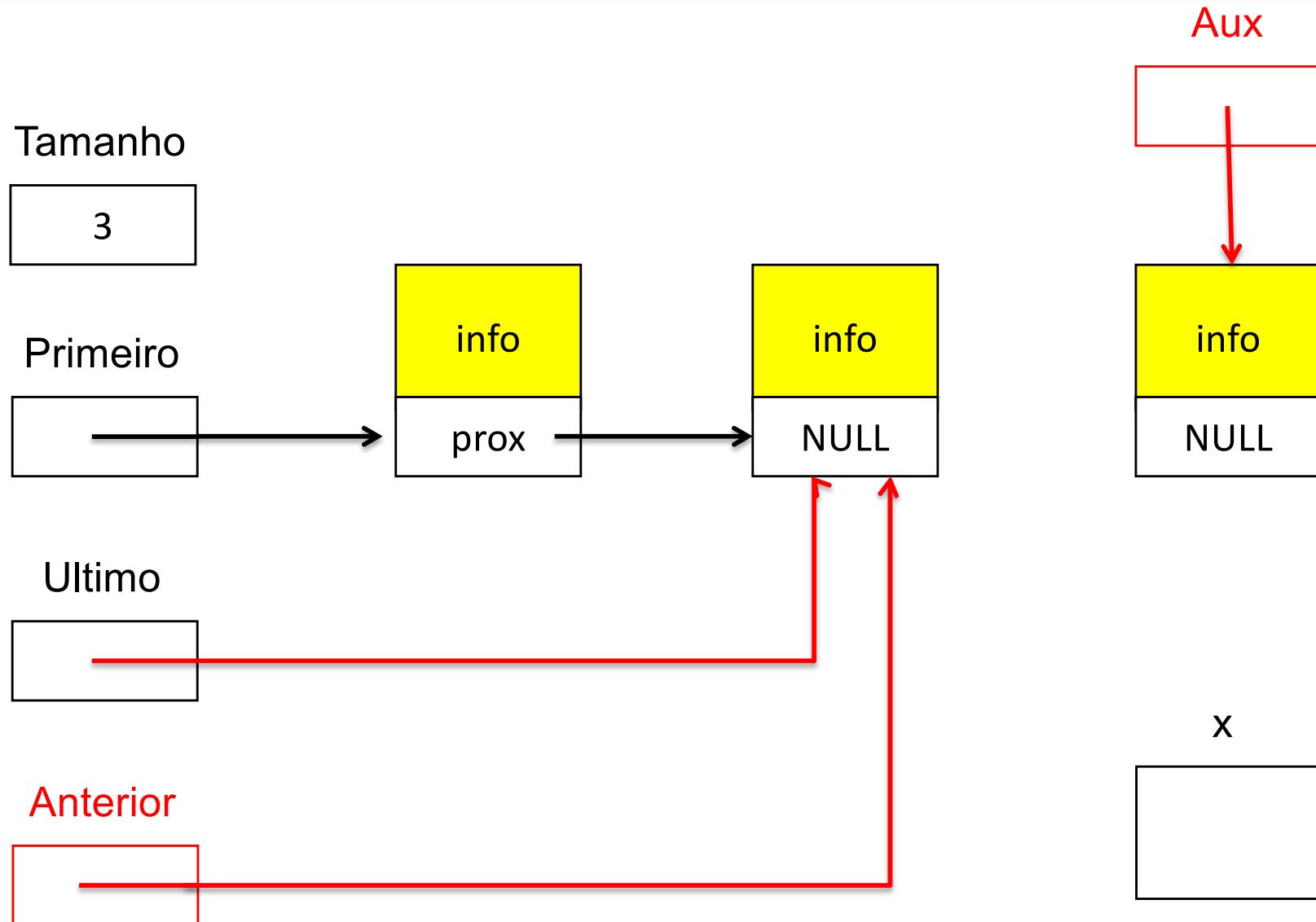
Retirada da Última Posição



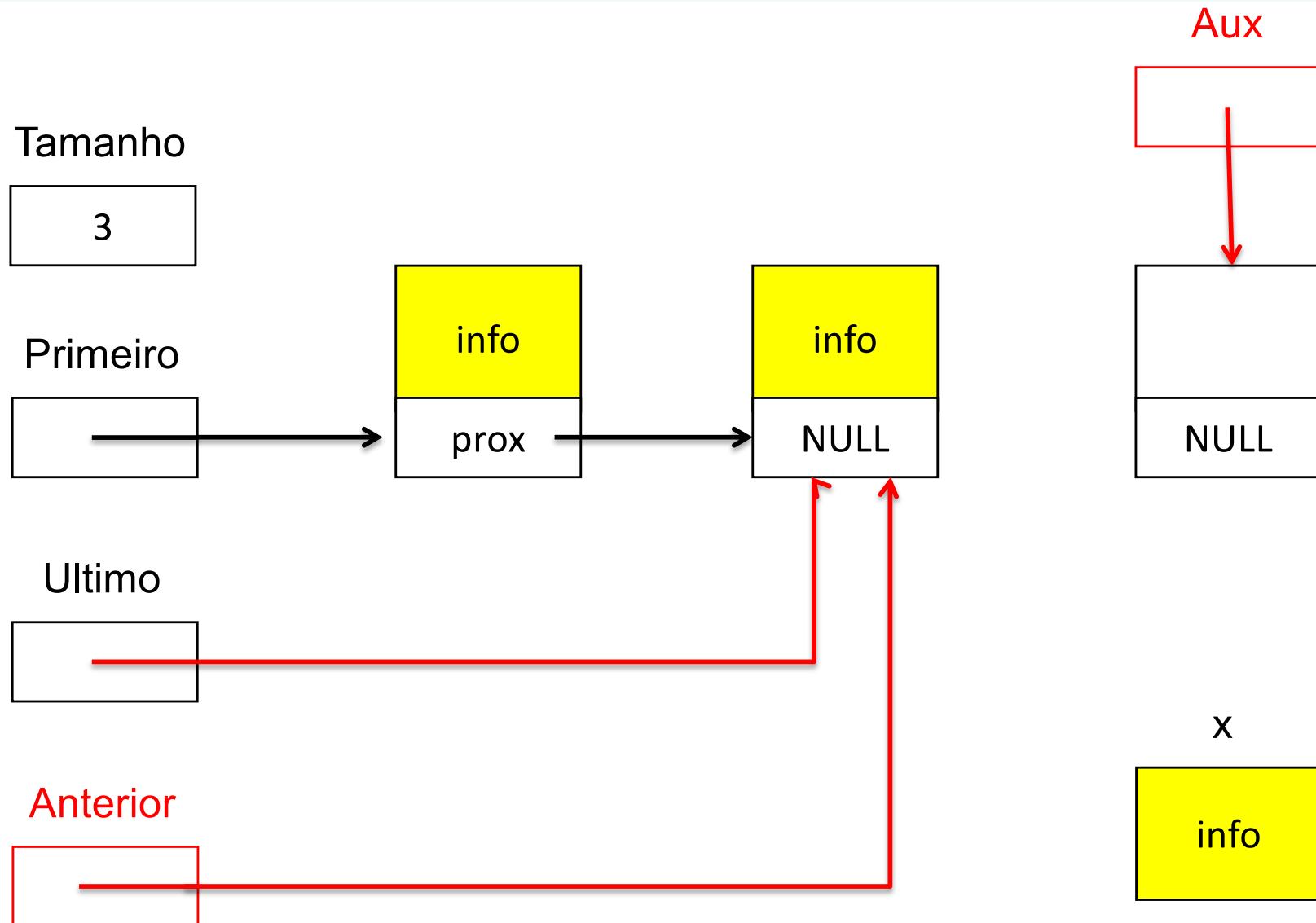
Retirada da Última Posição



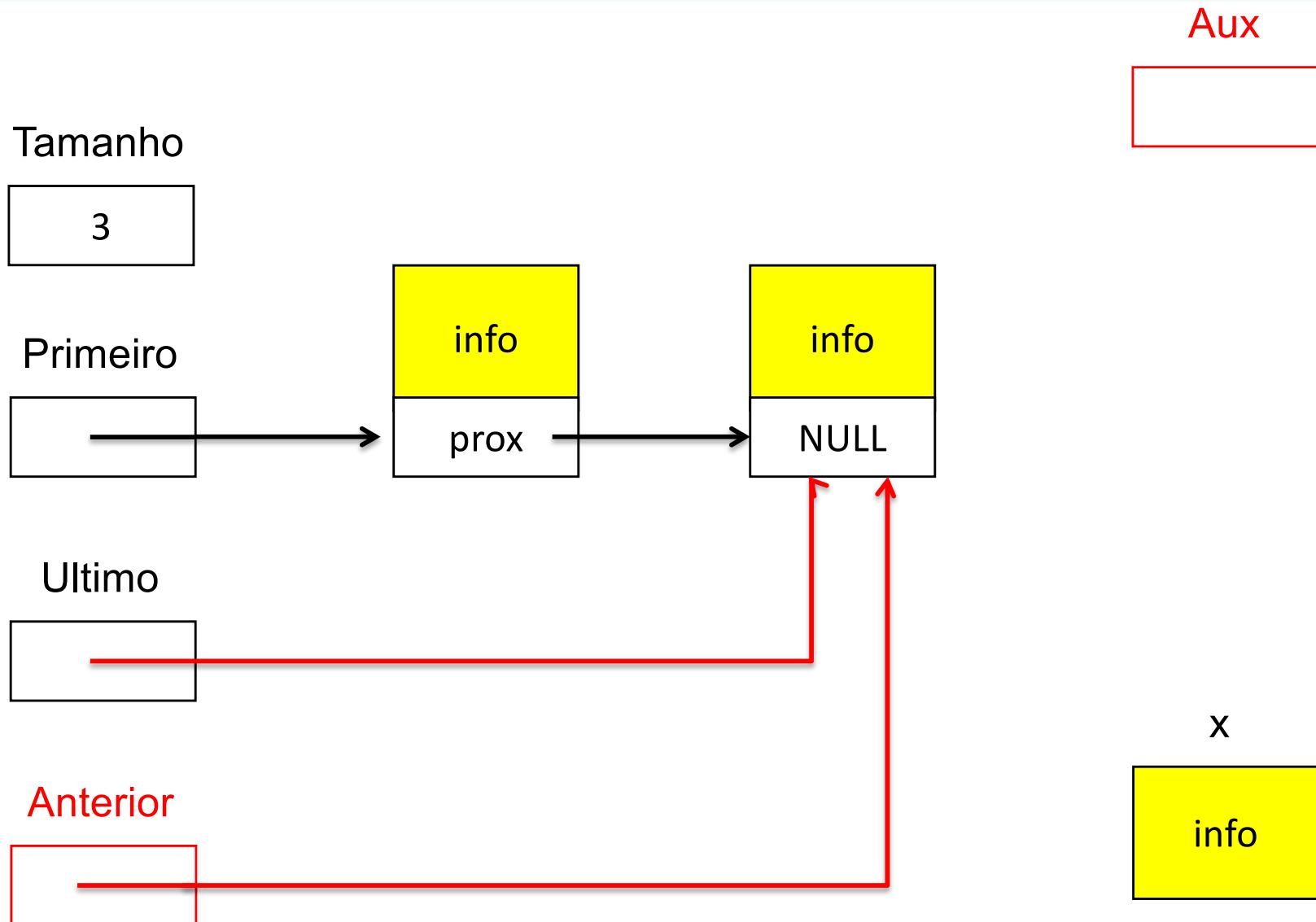
Retirada da Última Posição



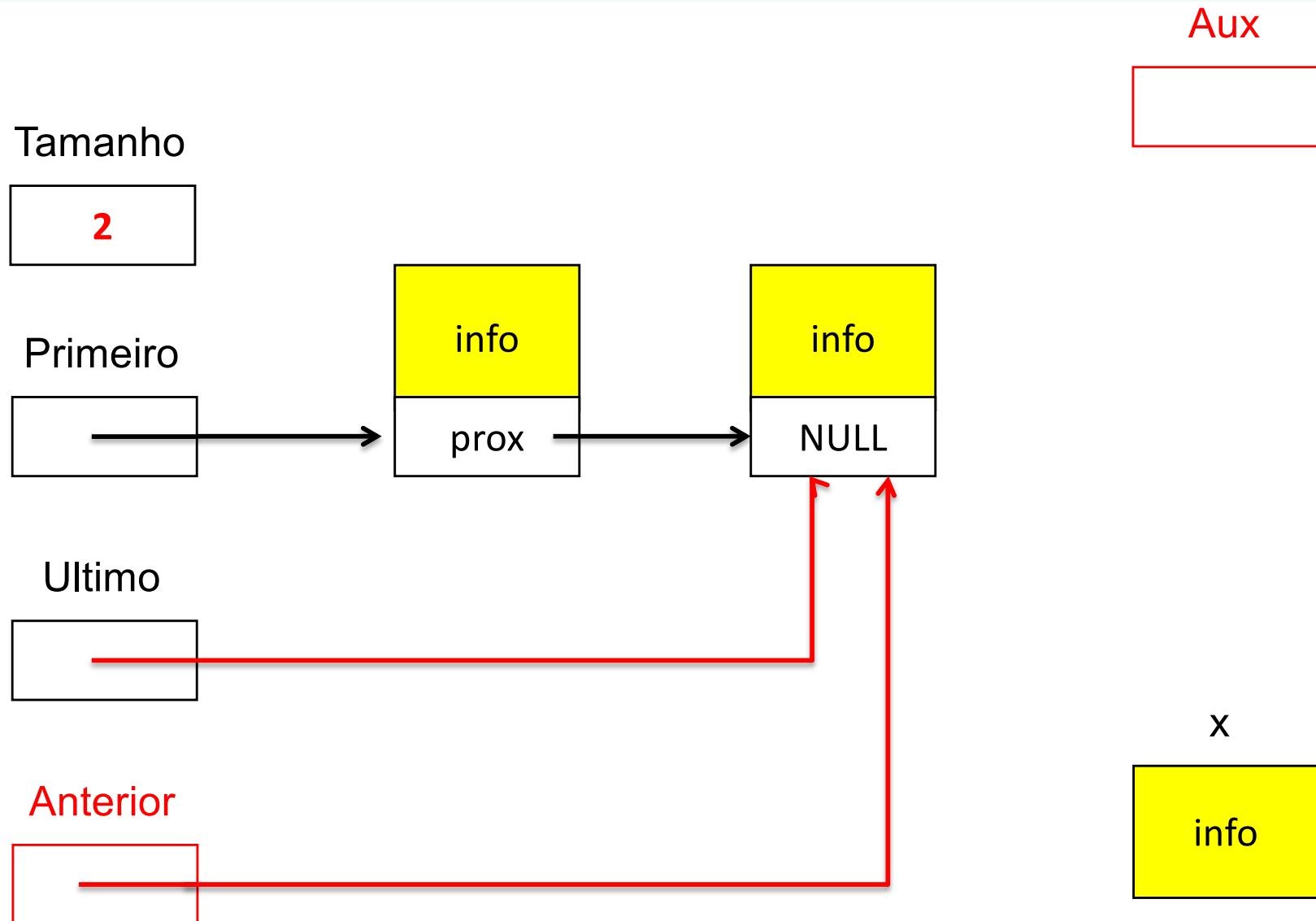
Retirada da Última Posição



Retirada da Última Posição



Retirada da Última Posição



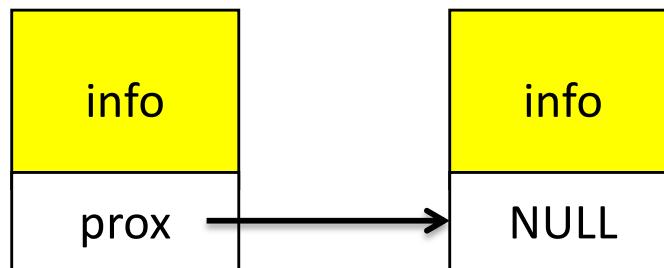
Retirada da Última Posição

Tamanho

2

Primeiro

—→



Último

—→

X

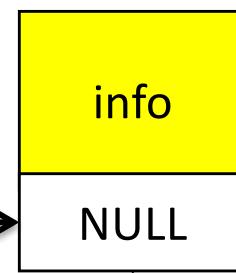
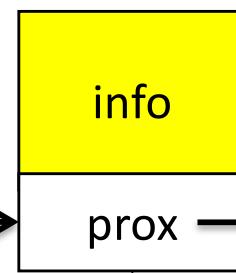
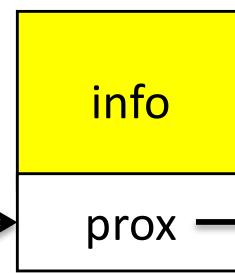
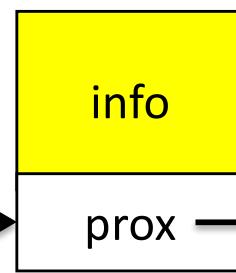
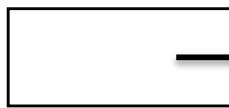
info

Retirada de Posição Qualquer

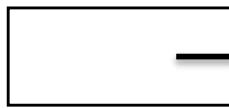
Tamanho

4

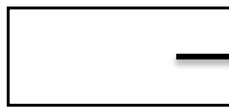
Primeiro



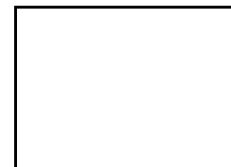
Último



p



x

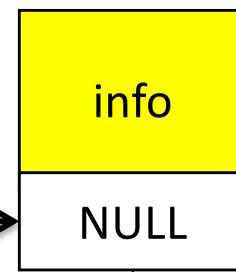
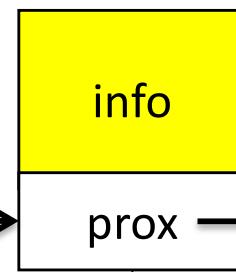
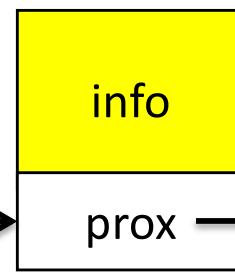
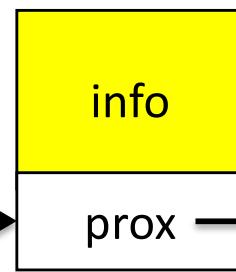
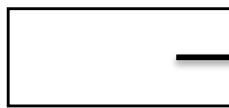


Retirada de Posição Qualquer

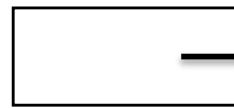
Tamanho

4

Primeiro



Último



p

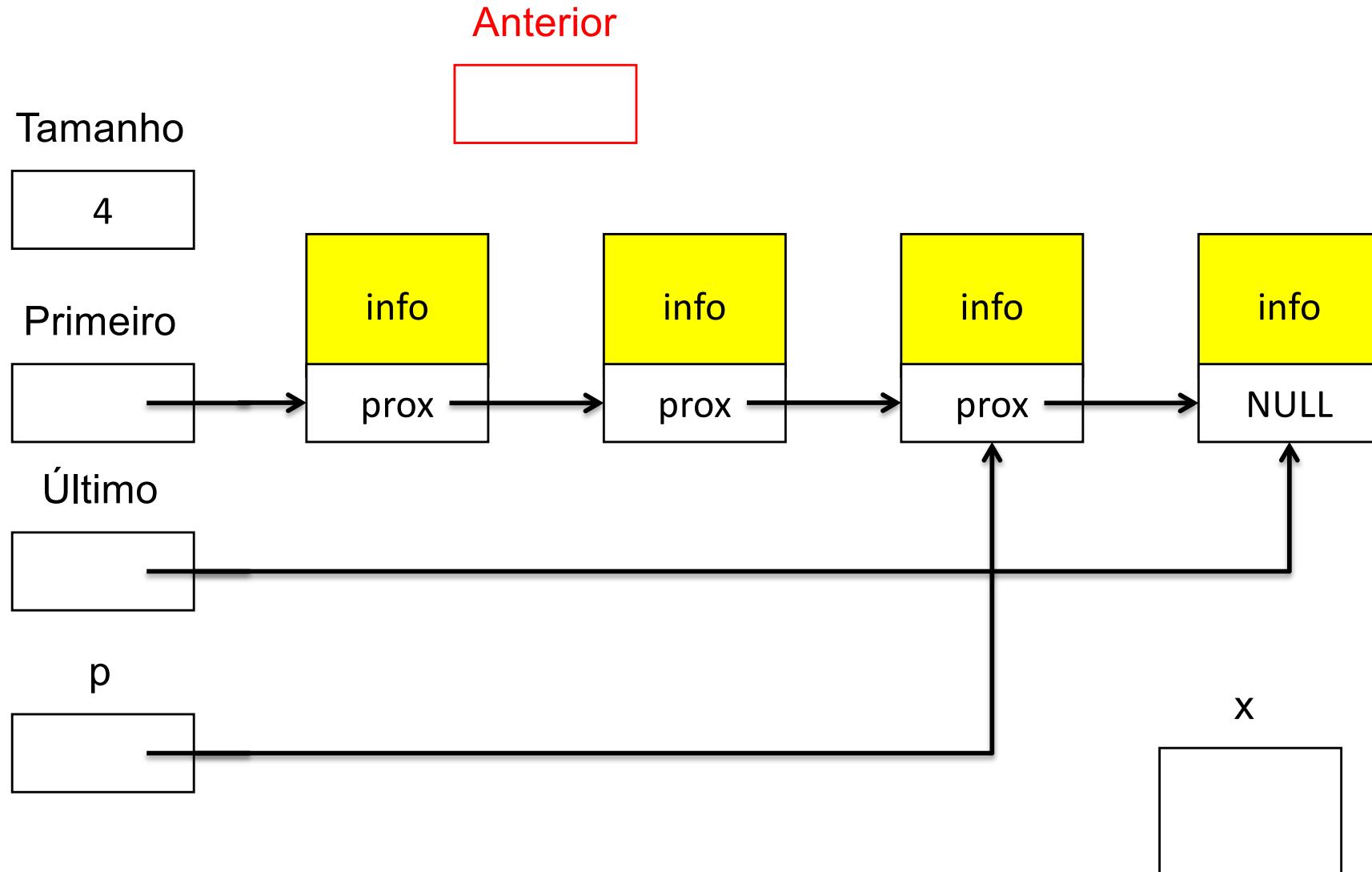


x

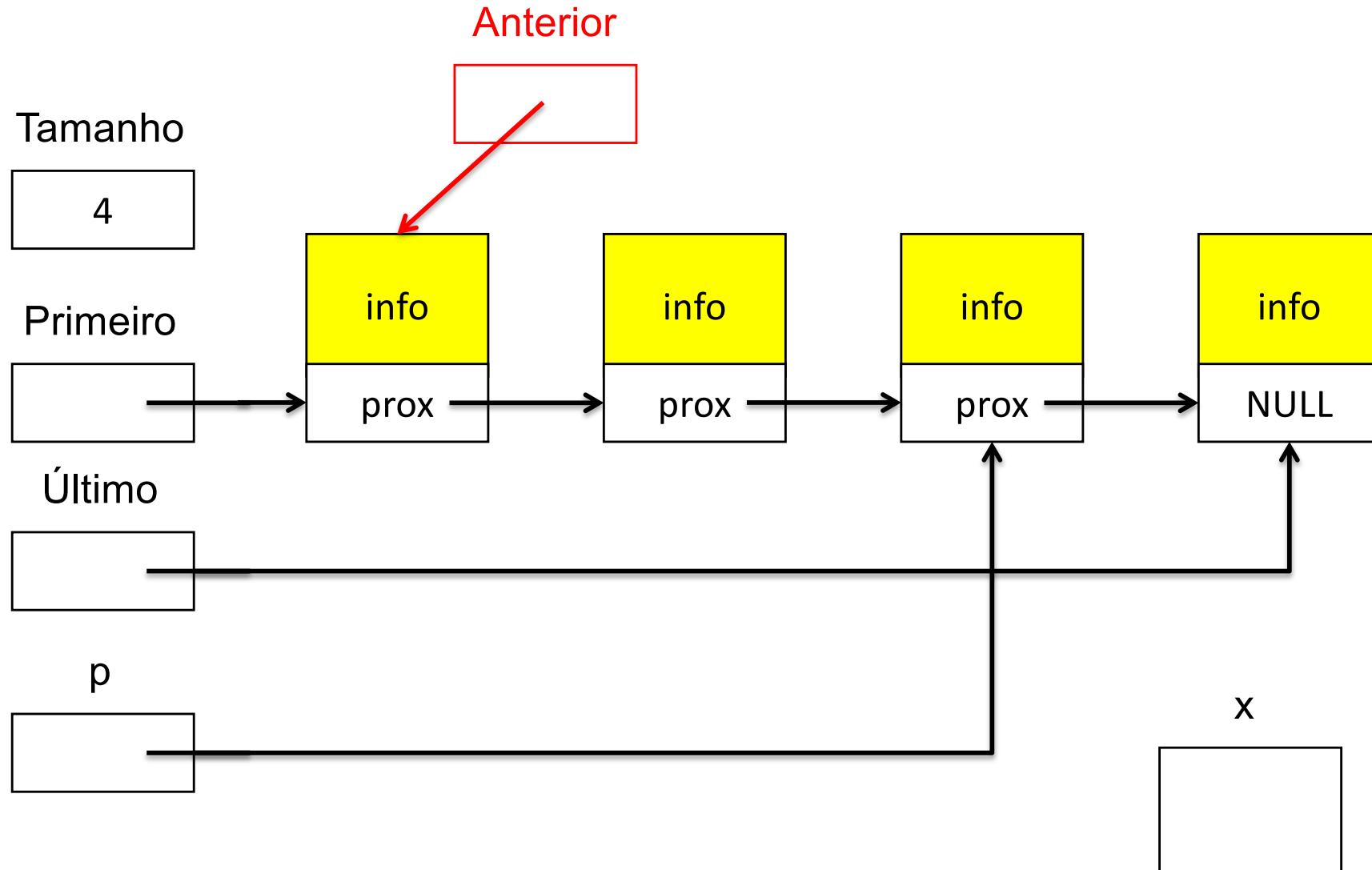


A lista está vazia?

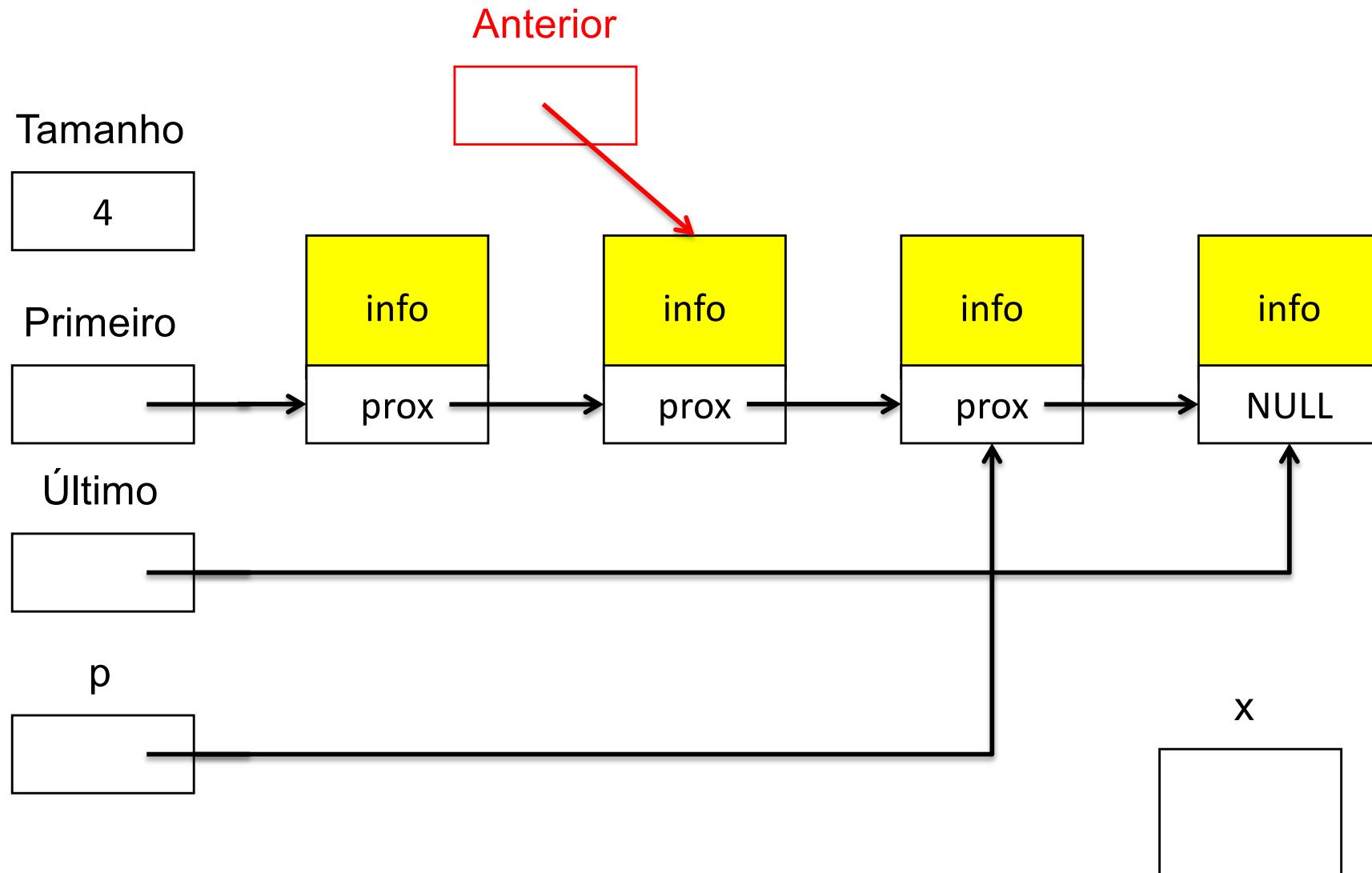
Retirada de Posição Qualquer



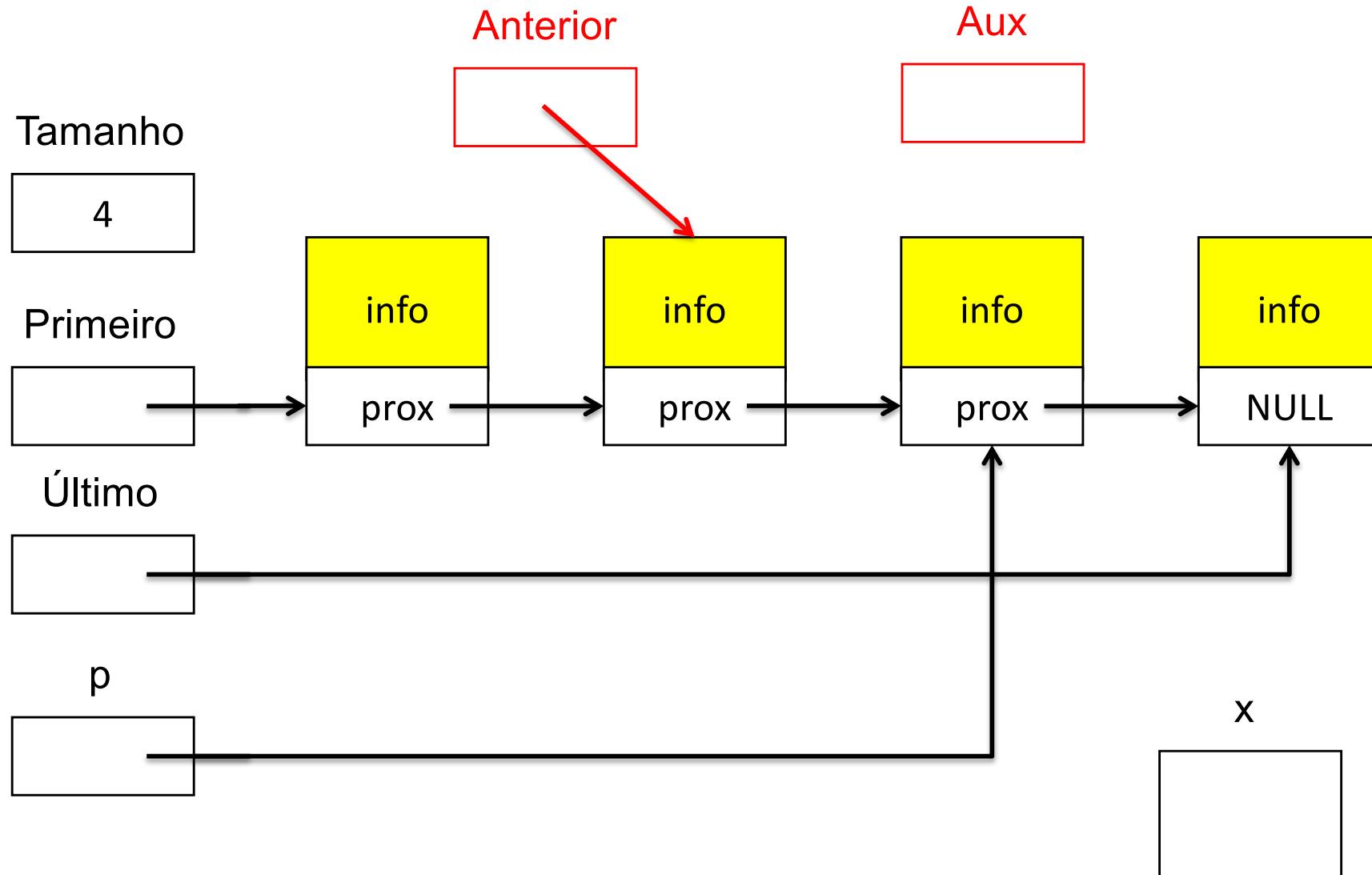
Retirada de Posição Qualquer



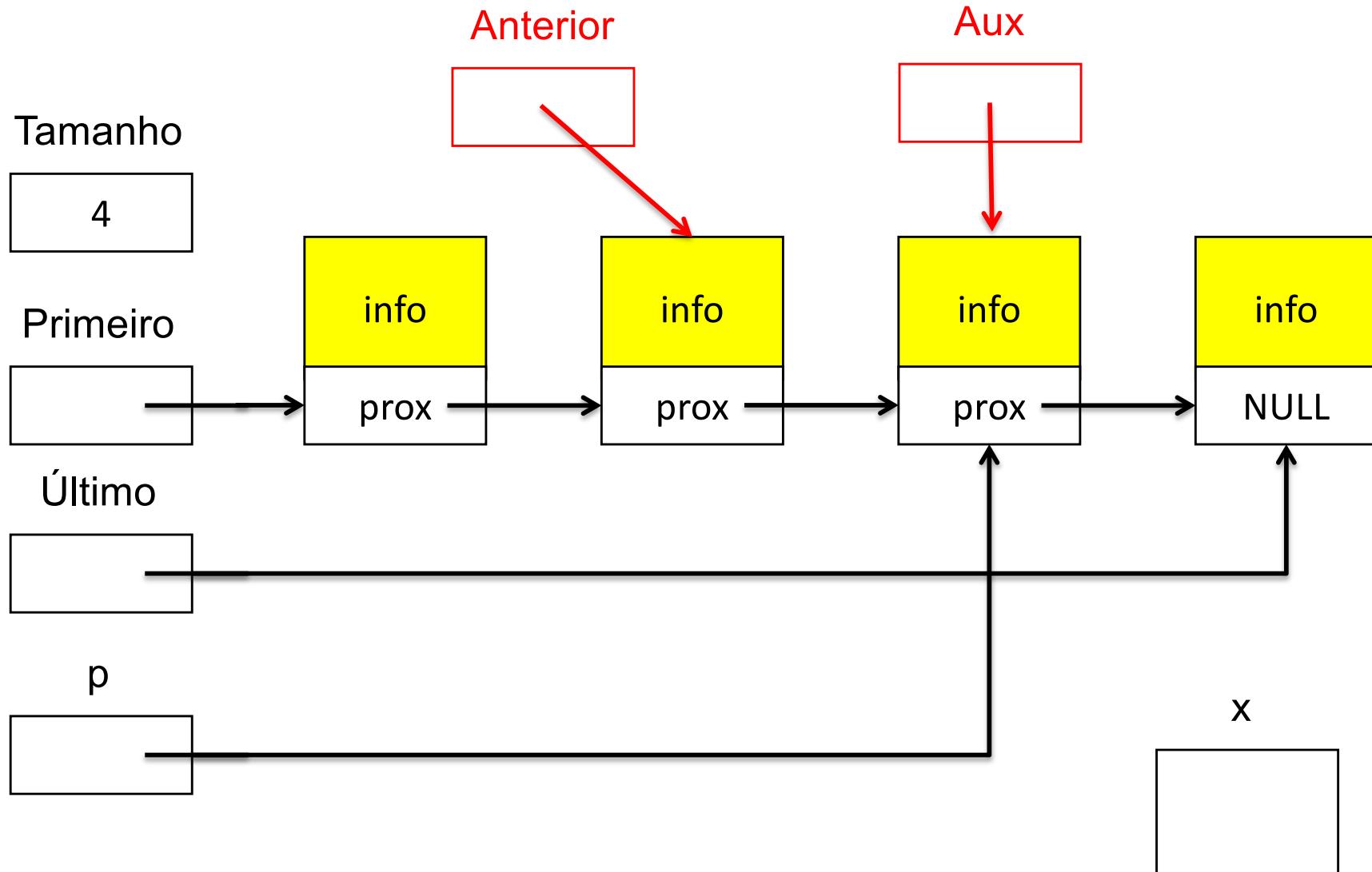
Retirada de Posição Qualquer



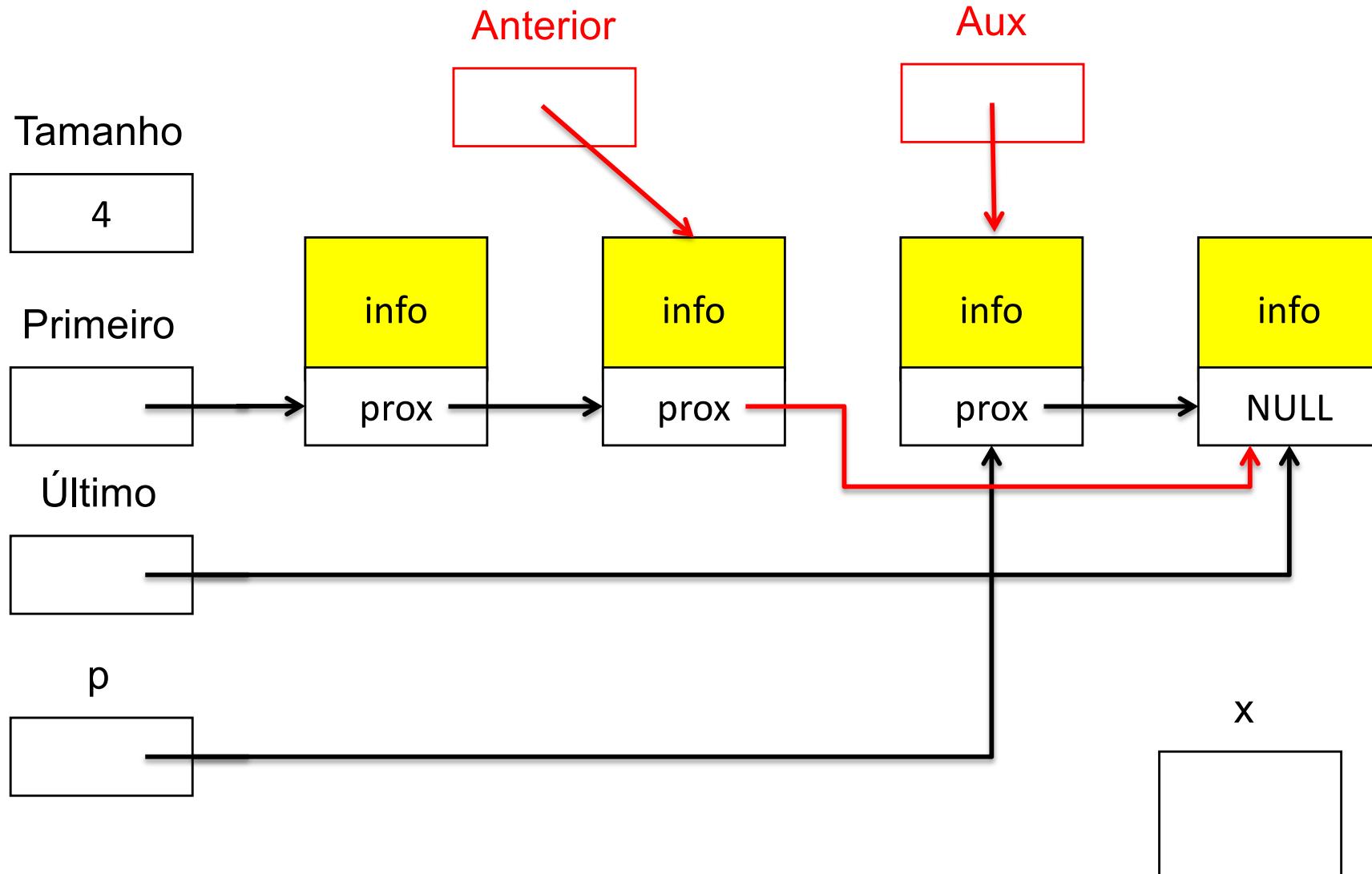
Retirada de Posição Qualquer



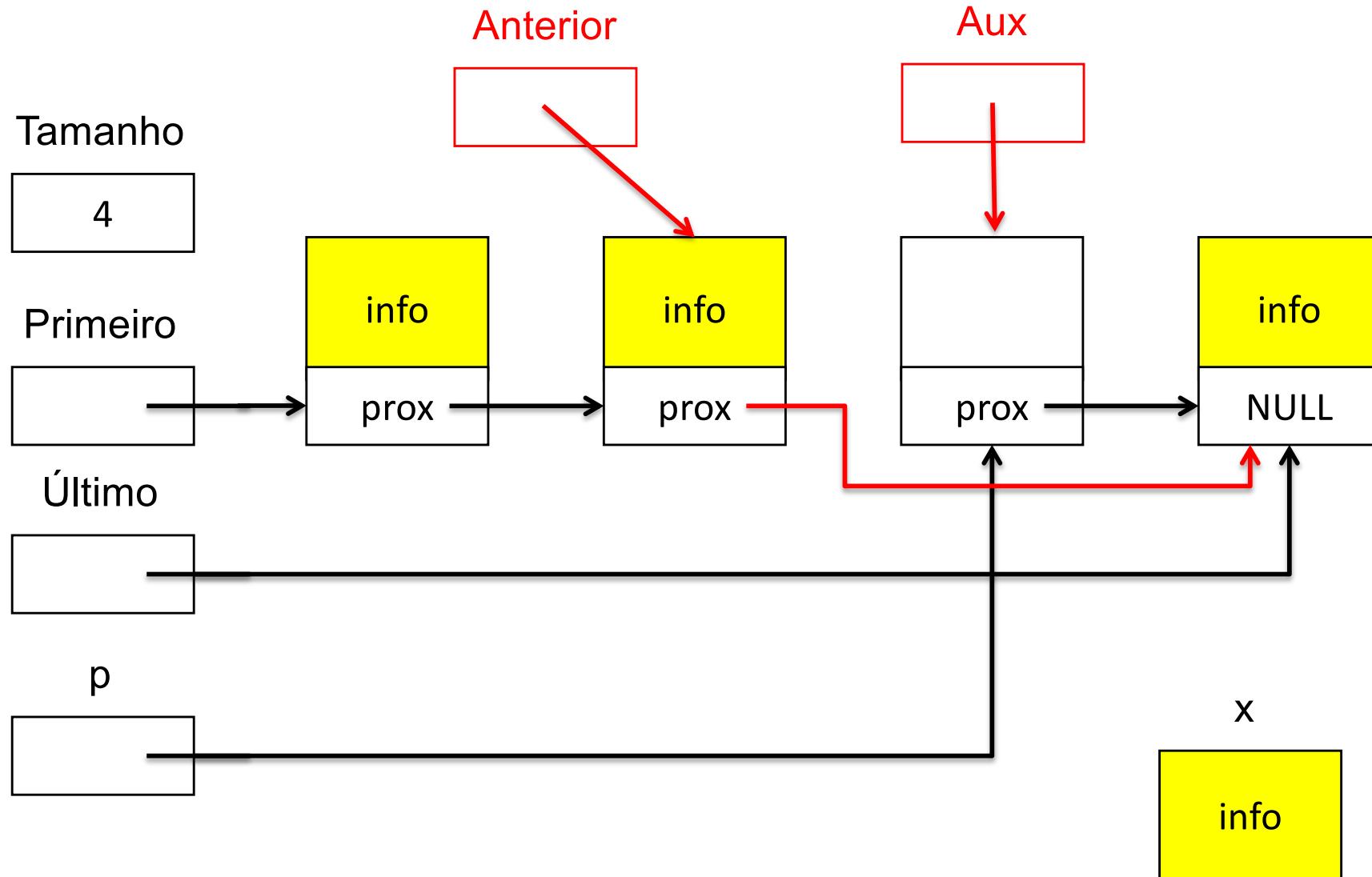
Retirada de Posição Qualquer



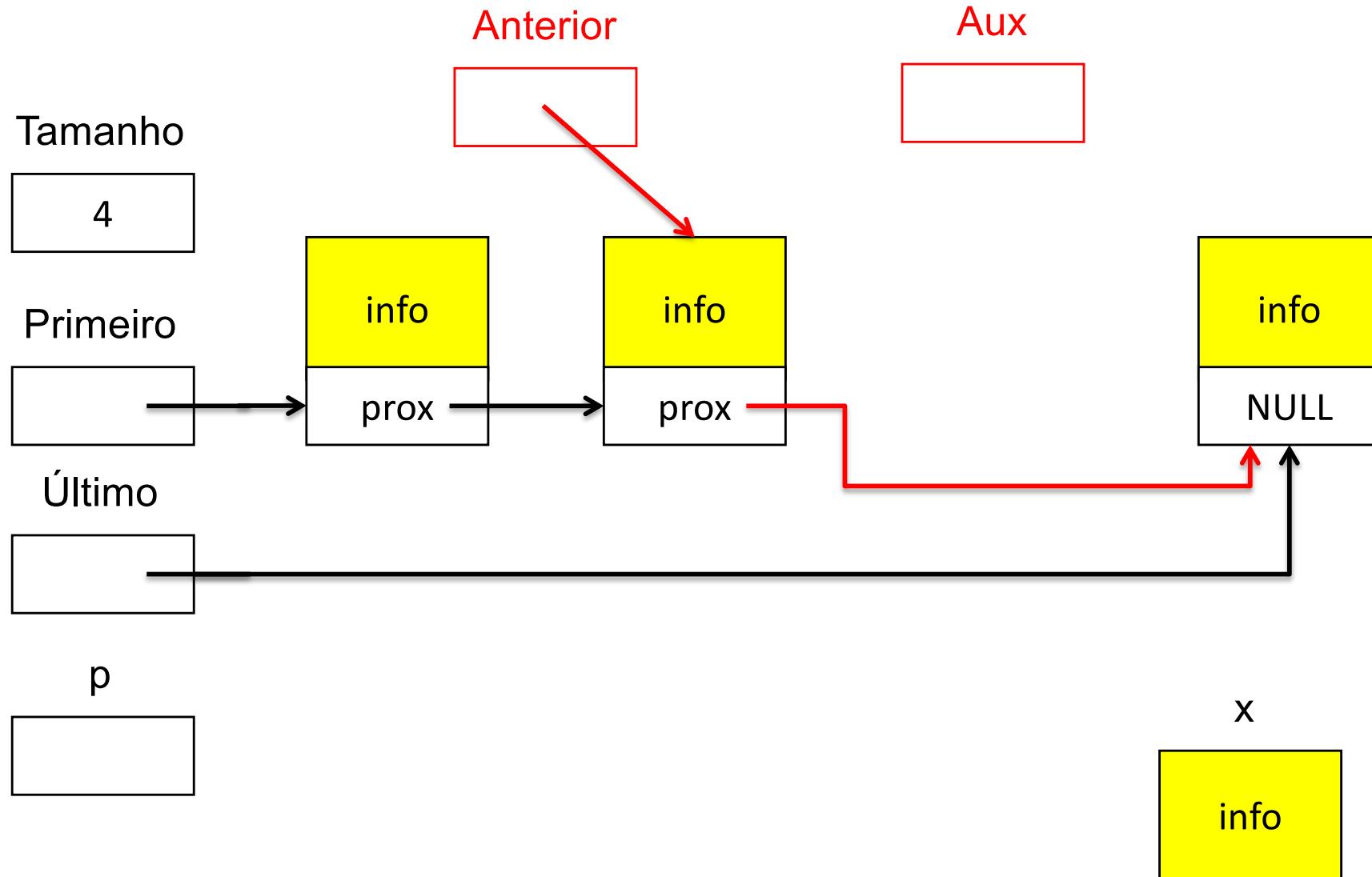
Retirada de Posição Qualquer



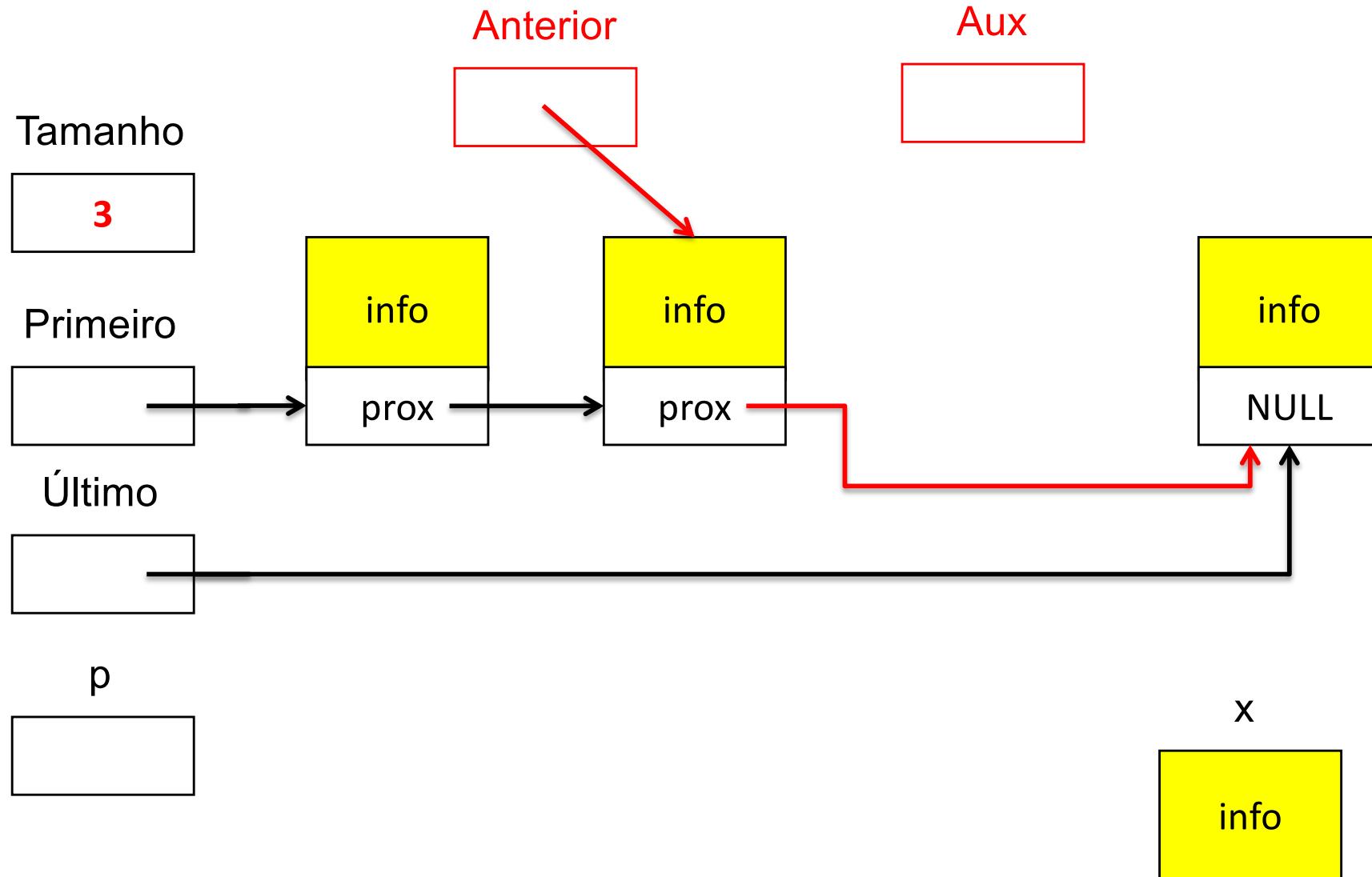
Retirada de Posição Qualquer



Retirada de Posição Qualquer



Retirada de Posição Qualquer

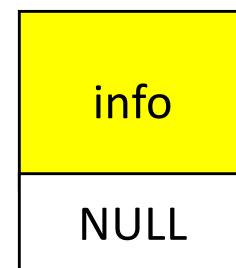
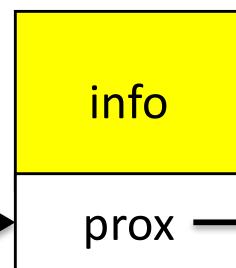
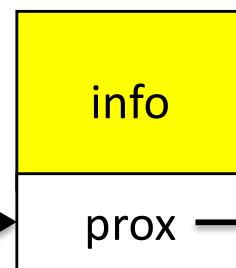


Retirada de Posição Qualquer

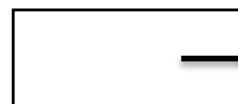
Tamanho

3

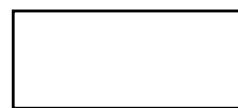
Primeiro



Último



p



x



Operações em Listas

```
/* Inicia as variaveis da lista */
void TLista_Inicia(TLista *pLista)
{
    pLista->Primeiro = NULL;
    pLista->Ultimo = pLista->Primeiro;
    pLista->Tamanho = 0;
}

/* Retorna se a lista esta vazia */
int TLista_EhVazia(TLista *pLista)
{
    return (pLista->Primeiro == NULL);
}

/* Retorna o tamanho da lista */
int TLista_Tamanho(TLista *pLista)
{
    return (pLista->Tamanho);
}
```

Operações em Listas

```
/* Retorna um apontador para o p-esimo item da lista */
TApontador TLista_Retorna(TLista *pLista, int p)
{
    TApontador pApontador;
    int Posicao;

    Posicao = 0;
    pApontador = pLista->Primeiro;
    while ((pApontador != NULL) && (Posicao != p)) {
        pApontador = pApontador->Prox;
        Posicao++;
    }

    return pApontador;
}
```

Operações em Listas

```
/* Insere um item na primeira posicao da lista */
int TLista_InserePrimeiro(TLista *pLista, TItem x)
{
    TApontador pNovo;

    pNovo = (TApontador) malloc(sizeof(TCelula));
    pNovo->Item = x;
    pNovo->Prox = pLista->Primeiro;
    if (TLista_EhVazia(pLista))
        pLista->Ultimo = pNovo;
    pLista->Primeiro = pNovo;
    pLista->Tamanho++;
    return 1;
}
```

Operações em Listas

```
/* Insere um item na ultima posicao da lista */
int TLista_InsereUltimo(TLista *pLista, TItem x)
{
    TApontador pNovo;

    pNovo = (TApontador) malloc(sizeof(TCelula));
    pNovo->Item = x;
    pNovo->Prox = NULL;

    if (TLista_EhVazia(pLista))
        pLista->Primeiro = pNovo;
    else
        pLista->Ultimo->Prox = pNovo;
    pLista->Ultimo = pNovo;
    pLista->Tamanho++;
    return 1;
}
```

Operações em Listas

```

/* Insere um item na lista na posicao apontada por p */
int TLista_Insere(TLista *pLista, TApontador p, TItem x)
{
    TApontador pAnterior, pNovo;

    if (p == pLista->Primeiro)
        return TLista_InserePrimeiro(pLista, x);
    else if (p == NULL)
        return TLista_InsereUltimo(pLista, x);
    else {
        pAnterior = pLista->Primeiro;
        while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
            pAnterior = pAnterior->Prox;
        if (pAnterior == pLista->Ultimo)
            return 0;

        pNovo = (TApontador) malloc(sizeof(TCelula));
        pNovo->Item = x;
        pNovo->Prox = pAnterior->Prox;
        pAnterior->Prox = pNovo;
        pLista->Tamanho++;
        return 1;
    }
}

```

Operações em Listas (Alternativa)

```

// p = p-ésima posição da lista
int TLista_Insere2(TLista *pLista, int p, TItem x)
{
    TApontador pAnterior, pNovo;
    int i;

    if (p == 1)
        return TLista_InserePrimeiro(pLista, x);
    else if (p == TLista_Tamanho(pLista))
        return TLista_InsereUltimo(pLista, x);
    else if ((p < 0) || (p > TLista_Tamanho(pLista)))
        return 0;
    else {
        i = 2;
        pAnterior = pLista->Primeiro;
        while (i != p) {
            pAnterior = pAnterior->Prox;
            i++;
        }

        pNovo = (TApontador) malloc(sizeof(TCelula));
        pNovo->Item = x;
        pNovo->Prox = pAnterior->Prox;
        pAnterior->Prox = pNovo;
        pLista->Tamanho++;
        return 1;
    }
}
  
```

Operações em Listas

```
/* Retira o item da primeira posicao da lista */
int TLista_RetiraPrimeiro(TLista *pLista, TItem *pX)
{
    TA pontador pAux;

    if (TLista_EhVazia(pLista))
        return 0;

    pAux = pLista->Primeiro;
    pLista->Primeiro = pAux->Prox;
    *pX = pAux->Item;
    free(pAux);
    pLista->Tamanho--;

    return 1;
}
```

Operações em Listas

```
/* Retira o item da ultima posicao da lista */
int TLista_RetiraUltimo(TLista *pLista, TItem *pX)
{
    TA pontador pAnterior, pAux;

    pAnterior = pLista->Primeiro;
    if (pAnterior == pLista->Ultimo)
        return TLista_RetiraPrimeiro(pLista, pX);
    else {
        while (pAnterior->Prox != pLista->Ultimo)
            pAnterior = pAnterior->Prox;

        pAux = pLista->Ultimo;
        pAnterior->Prox = NULL;
        pLista->Ultimo = pAnterior;
        *pX = pAux->Item;
        free(pAux);
        pLista->Tamanho--;

        return 1;
    }
}
```

Operações em Listas

```
/* Retira o item da lista da posicao apontada por p */
int TLista_Retira(TLista *pLista, TApontador p, TItem *pX)
{
    TApontador pAnterior, pAux;

    if (p == pLista->Primeiro)
        return TLista_RetiraPrimeiro(pLista, pX);
    else if (p == pLista->Ultimo)
        return TLista_RetiraUltimo(pLista, pX);
    else {
        pAnterior = pLista->Primeiro;
        while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
            pAnterior = pAnterior->Prox;
        if (pAnterior == pLista->Ultimo)
            return 0;

        pAux = pAnterior->Prox;
        pAnterior->Prox = pAux->Prox;
        *pX = pAux->Item;
        free(pAux);
        pLista->Tamanho--;
        return 1;
    }
}
```

■ Vantagens:

- ~~Permite inserir ou retirar itens do meio da lista a custo $O(1)$~~
 - Importante quando a lista tem de ser mantida em ordem
- Bom para aplicações em que não existe previsão sobre o crescimento da lista
 - O tamanho máximo da lista não precisa ser definido a priori

■ Desvantagens:

- Utilização de memória extra para armazenar os apontadores
- Percorrer a lista, procurando o i -ésimo item, tem custo $O(n)$

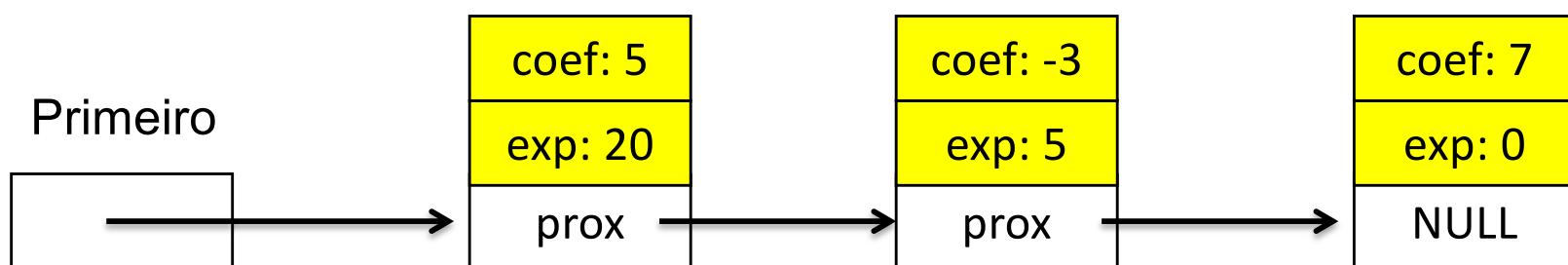
Exemplos de Aplicação

- Manipulação de polinômios de grau $n \geq 0$:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 x^0$$

- A solução a seguir usa listas por apontadores:
 - cada nó representa um termo com coeficiente não nulo
 - os termos estão em ordem decrescente dos expoentes

$$P(x) = 5 x^{20} - 3 x^5 + 7$$



Exemplos de Aplicação

```
typedef struct {
    int Expoente;
    float Coeficiente;
} TItem;

/* Retorna o p-esimo item da lista */
TItem TLista_LeItem(TLista *pLista, int p)
{
    TA pontador q;
    q = TLista_Retorna(pLista, p);
    return q->Item;
}
```

Exemplos de Aplicação

```
int InsereTermo(TLista *pLista, TItem x)
{
    TApontador p;
    TItem Item;
    int i;

    for (i = 0; i < TLista_Tamanho(pLista); i++) {
        Item = TLista_LeItem(pLista, i);
        if (Item.Expoente < x.Expoente)
            break;
    }

    p = TLista_Retorna(pLista, i);
    return TLista_Insere(pLista, p, x);
}
```

Exemplos de Aplicação

```
void IniciaPolinomio(TLista *pLista, char *expr)
{
    TItem t;
    int r, n, nn = 0;
    TLista_Inicia(pLista);
    do /* Formato expr: "5.0*x^4 + 7.5*x^2 + 3.0*x^1" */
        r = sscanf(expr+nn, " %f*x^%d%n", &t.Coefficiente,
                   &t.Expoente, &n);
        if (r == EOF || r == 0) break;
        nn += n;
        InsereTermo(pLista, t);
        r = sscanf(expr+nn, " + %n", &n);
        if (r == EOF) break;
        nn += n;
    } while (1);
}
```

Exemplos de Aplicação

```
float CalculaTermo(TItem t, float x)
{
    return t.Coefficiente * pow(x, t.Expoente);
}

float CalculaPolinomio(TLista *pLista, float x)
{
    TItem Item;
    float soma;
    int i;

    soma = 0.0;
    for (i = 0; i < TLista_Tamanho(pLista); i++) {
        Item = TLista_LeItem(pLista, i);
        soma += CalculaTermo(Item, x);
    }

    return soma;
}
```

Exemplos de Aplicação

```
void ImprimeTermo(TItem t)
{
    printf("%f*x^%d", t.Coefficiente, t.Expoente);
}

void ImprimePolinomio(TLista *pLista)
{
    TItem Item;
    int i;

    if (TLista_EhVazia(pLista))
        return;

    Item = TLista_LeItem(pLista, 0);
    ImprimeTermo(Item);
    for (i = 1; i < TLista_Tamanho(pLista); i++) {
        printf(" + ");
        Item = TLista_LeItem(pLista, i);
        ImprimeTermo(Item);
    }
    printf("\n");
}
```

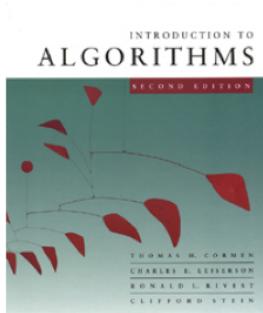
Exercício

- Imagine que você descobriu uma biblioteca que implementa o TAD Lista de maneira diferente, mas ainda oferece suporte as seguintes operações:

```
void TLista_Inicia(TLista *pLista);
int TLista_EhVazia(TLista *pLista);
int TLista_Insere(TLista *pLista, TApontador p, TItem x);
int TLista_Retira(TPilha *pLista, TApontador p, TItem *pX);
TApontador TLista_Retorna(TLista *pLista, int p);
int TLista_Tamanho(TLista *pLista);
```

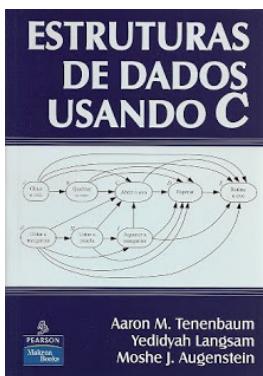
- Você possui apenas os códigos em binário dessa biblioteca e, portanto, não tem conhecimento de como esse TAD Lista foi implementado. Usando esse TAD Lista, implemente o TAD Pilha e o TAD Fila

Leitura Recomendada



CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms.** 3^a Edição. MIT Press, 2009. **Seção 10.2**

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C.** 3^a Edição. Cengage Learning, 2010. **Seção 3.1**



TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de Dados usando C.** Pearson Makron Books, 2008.
Capítulo 4