

Algoritmos e Estruturas de Dados I

Aula 07: Listas Lineares - Parte 2

Prof. Márcio Porto Basgalupp

créditos: Prof. Jurandy G. Almeida Jr.

Universidade Federal de São Paulo
Departamento de Ciência e Tecnologia

■ Problema:

- As operações de inserção em uma lista vazia e retirada em uma lista com um único elemento podem requerer um tratamento especial dos apontadores **Primeiro** e **Último**

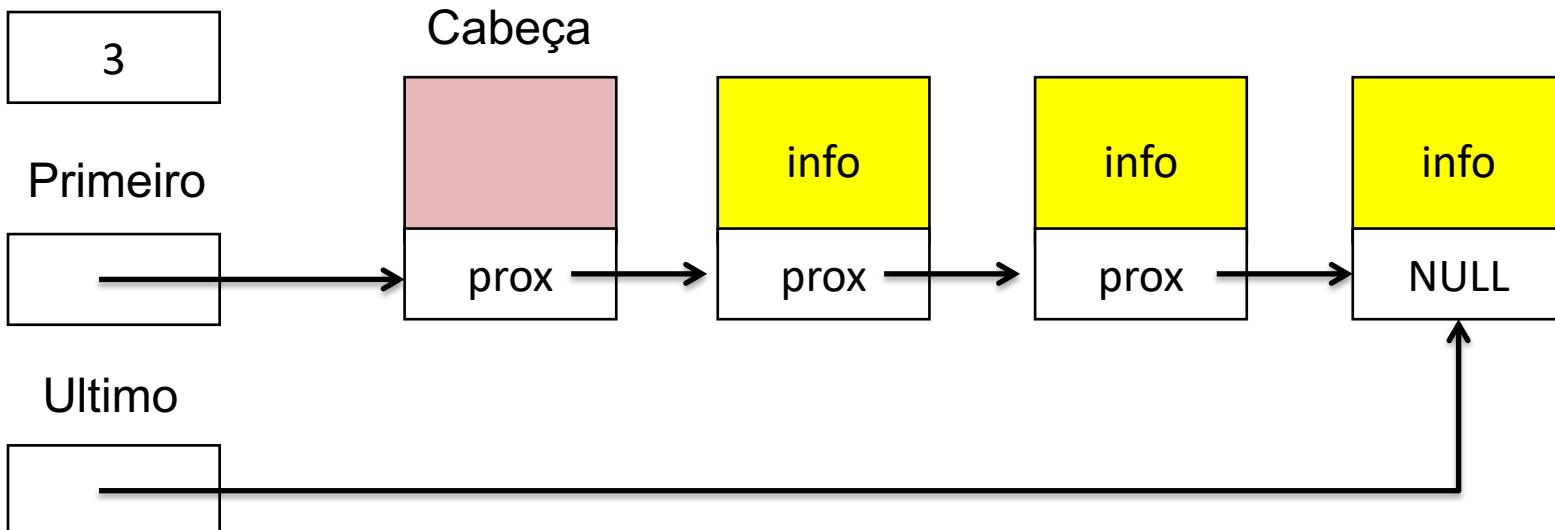
■ Solução:

- Manter uma célula adicional chamada de **célula cabeça**, que serve apenas para marcar o ínicio da lista

■ Listas com Cabeça

- Uma célula adicional é mantida no ínicio da lista
- O primeiro item da lista está na célula **Primeiro->Prox**

Tamanho



Estrutura da Lista com Cabeça

```
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;
```

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

Tamanho

0

Primeiro

Último

Cria Lista Vazia

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

Tamanho

0

Cabeça

Primeiro

—> NULL

Último

Cria Lista Vazia

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

Tamanho

0

Cabeça

Primeiro

→

NULL

→

Último

Cria Lista Vazia

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

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

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

Inserção em Lista Vazia

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

Tamanho

0

Cabeça

Primeiro

→ NULL

→ NULL

Último

Inserção em Lista Vazia

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

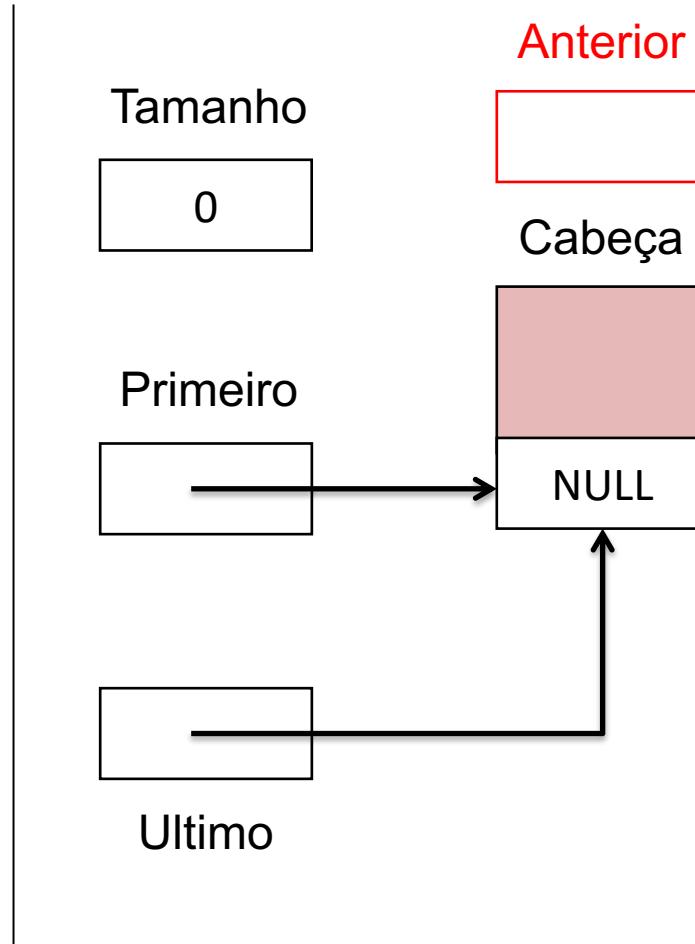
Anterior

Cabeça

Primeiro

NULL

Último



Inserção em Lista Vazia

Listas sem Cabeça

Tamanho

0

Primeiro

NULL

NULL

Último

Listas com Cabeça

Anterior



Cabeça



NULL

Primeiro



Último

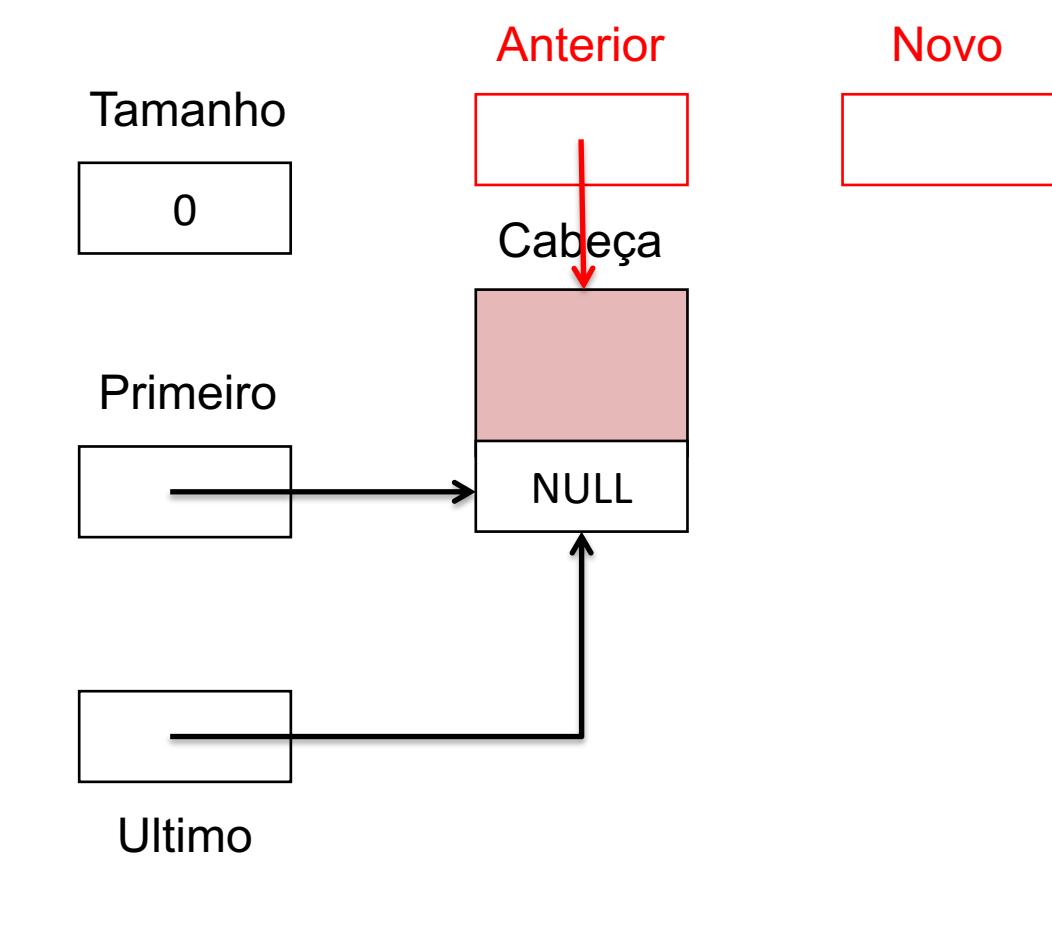


Inserção em Lista Vazia

Listas sem Cabeça

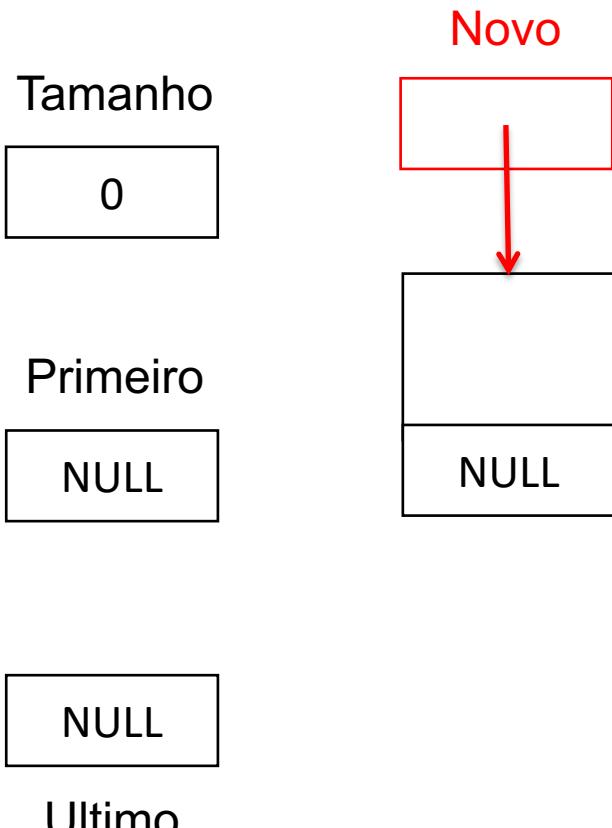
Tamanho	Novo	
		<input type="text"/>
	0	
Primeiro		
		<input type="text"/>
	NULL	
		<input type="text"/>
	NULL	
Último		
		<input type="text"/>

Listas com Cabeça

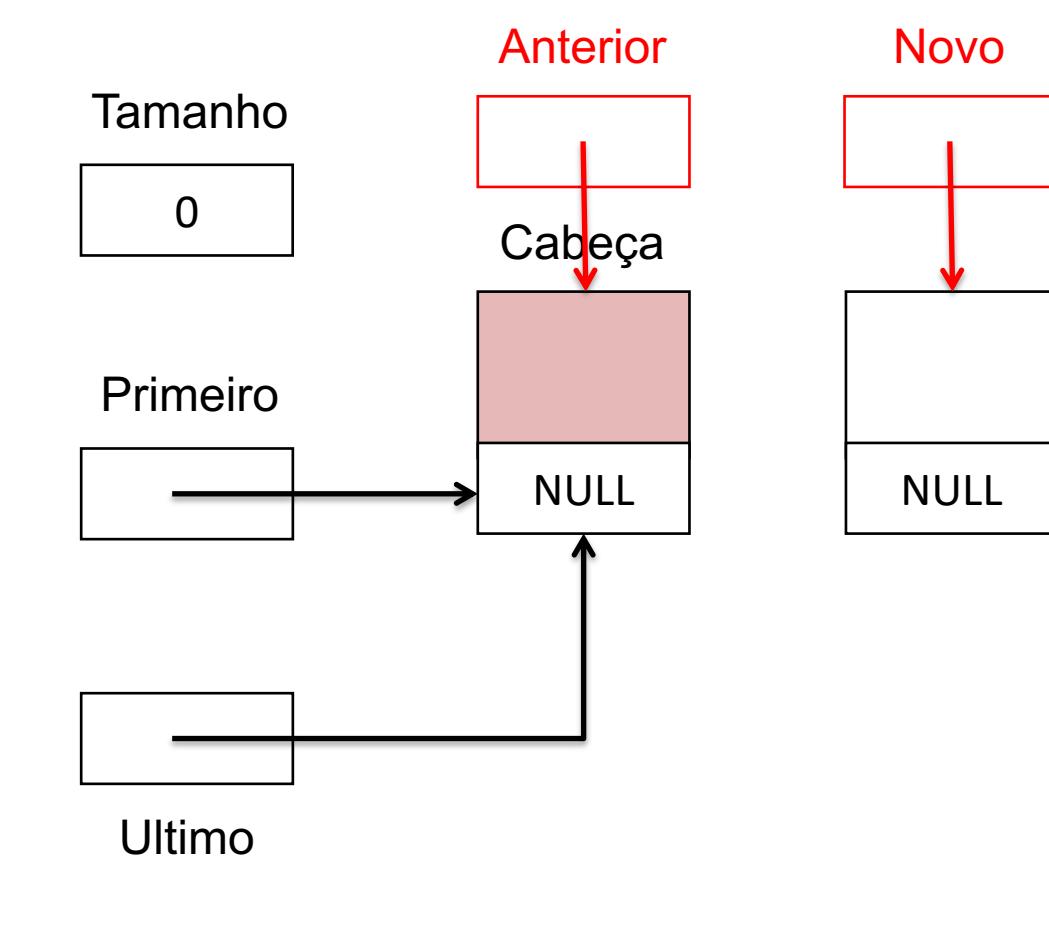


Inserção em Lista Vazia

Listas sem Cabeça

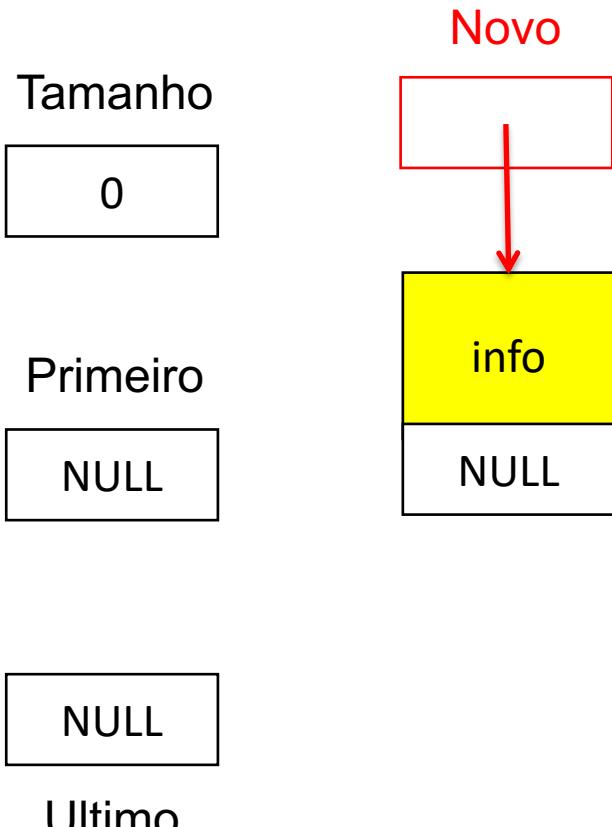


Listas com Cabeça

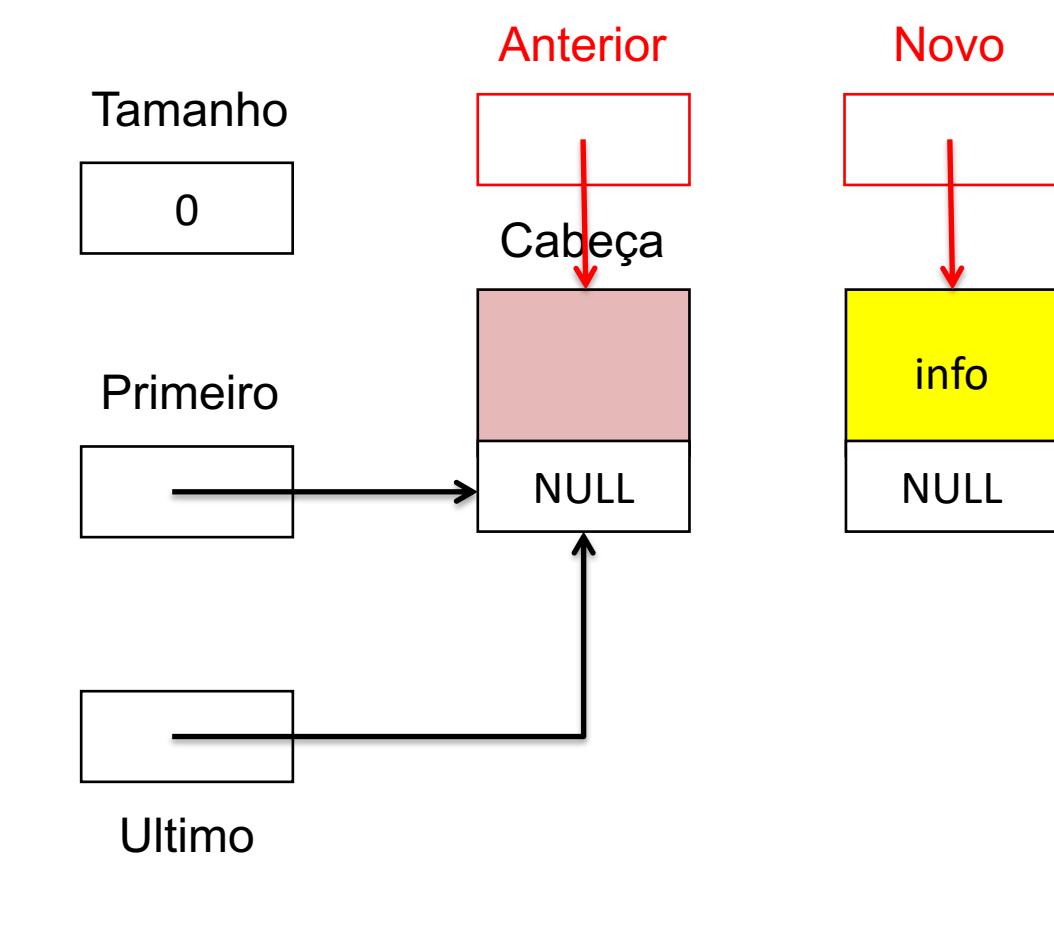


Inserção em Lista Vazia

Listas sem Cabeça

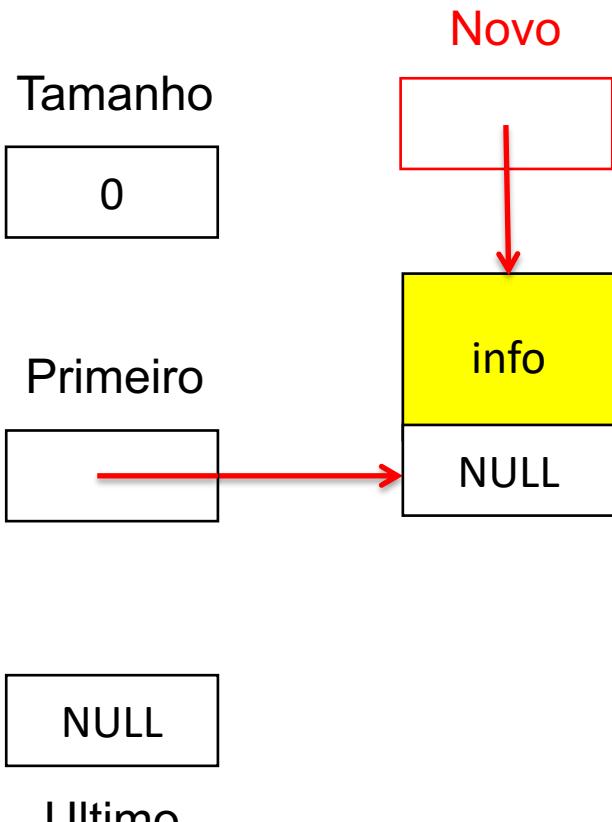


Listas com Cabeça

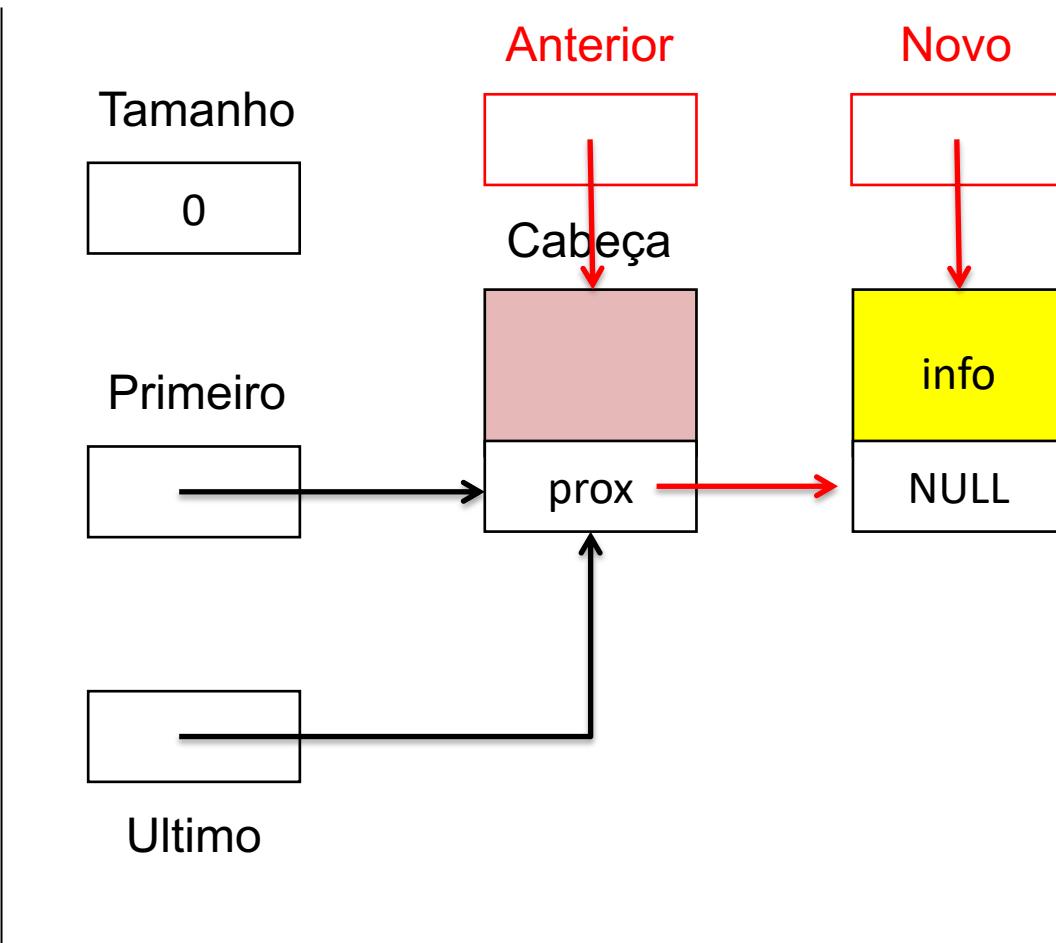


Inserção em Lista Vazia

Listas sem Cabeça

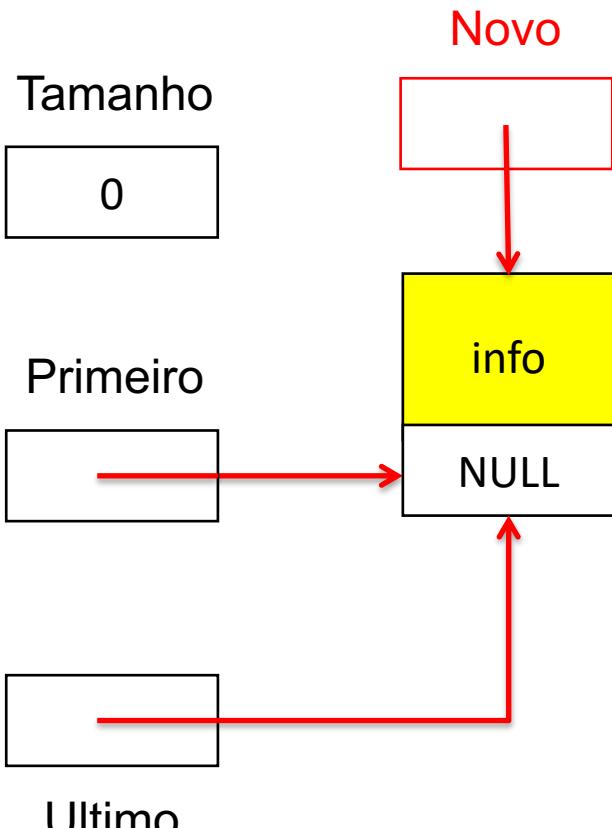


Listas com Cabeça

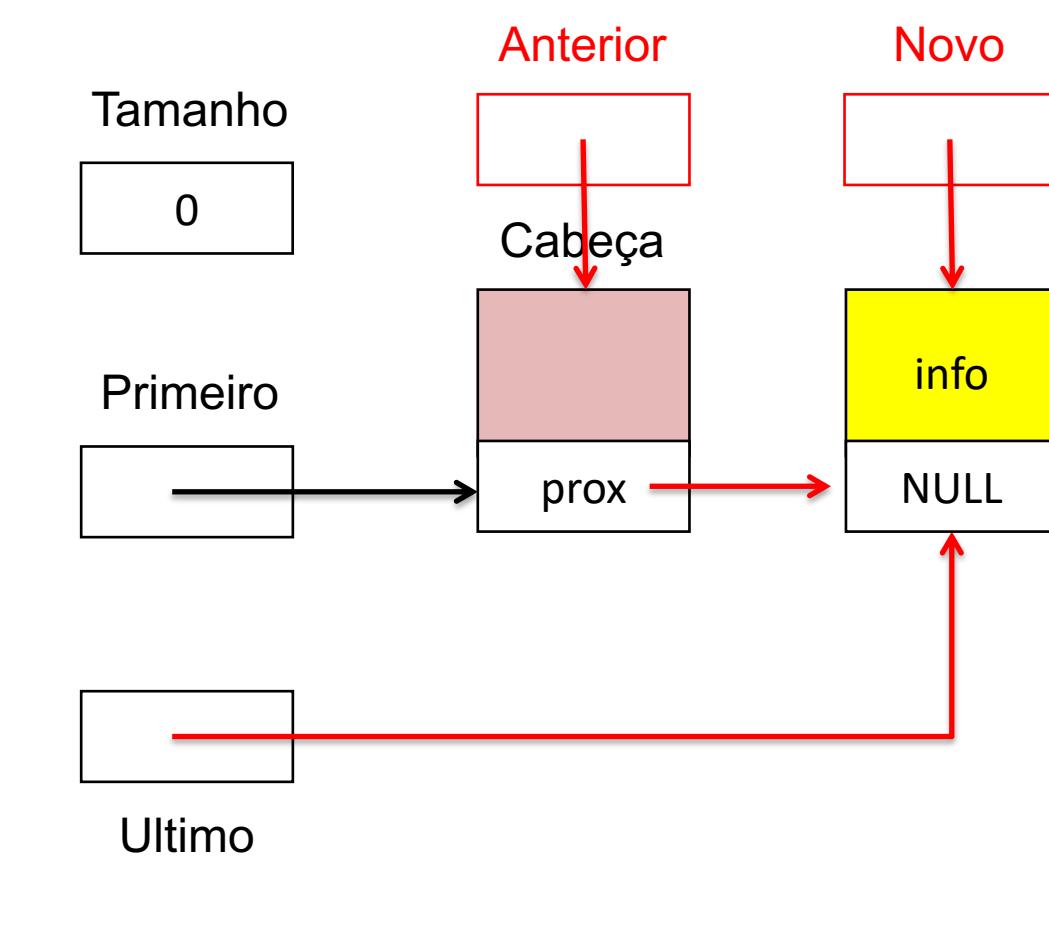


Inserção em Lista Vazia

Listas sem Cabeça

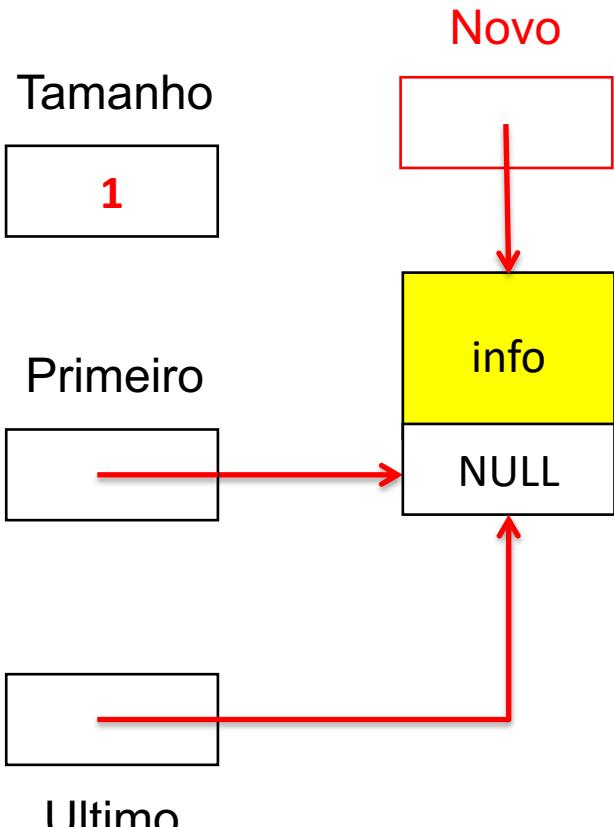


Listas com Cabeça

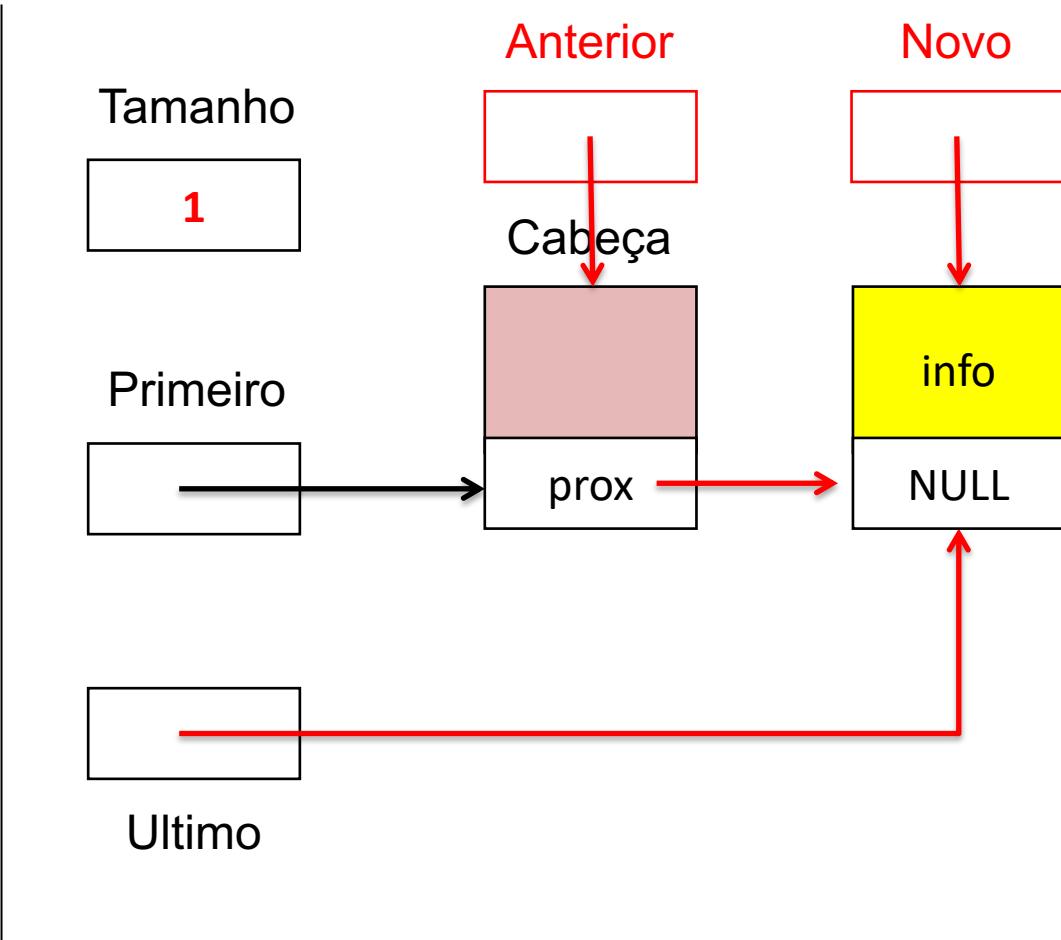


Inserção em Lista Vazia

Listas sem Cabeça



Listas com Cabeça



Inserção em Lista Vazia

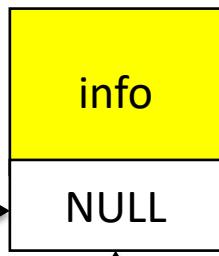
Listas sem Cabeça

Tamanho

1

Primeiro

—→



Último

Listas com Cabeça

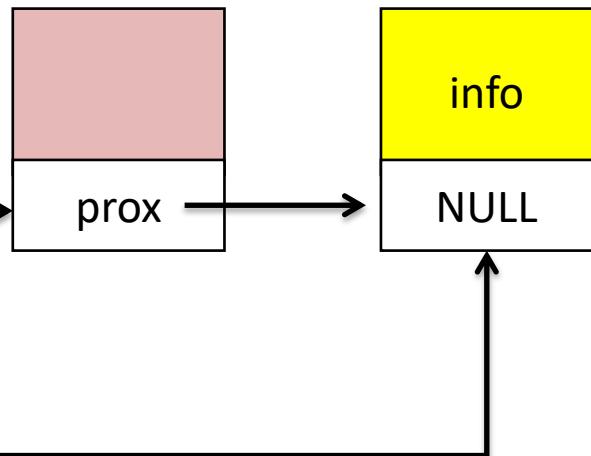
Tamanho

1

Cabeça

Primeiro

—→



Último

Inserção em Lista Vazia

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

    pAnterior = pLista->Primeiro;
    while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
        pAnterior = pAnterior->Prox;
    if (pAnterior->Prox != p)
        return 0;

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

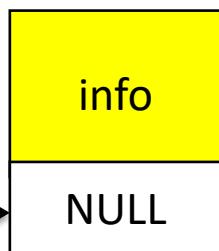
Retirada de Lista com 1 Elemento

Listas sem Cabeça

Tamanho

1

Primeiro



Último



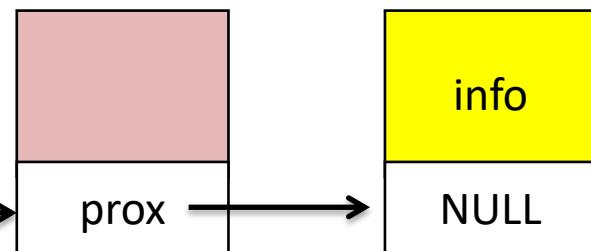
Listas com Cabeça

Tamanho

1

Cabeça

Primeiro



x



Último

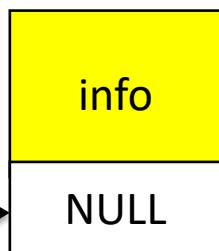
Retirada de Lista com 1 Elemento

Listas sem Cabeça

Tamanho

1

Primeiro



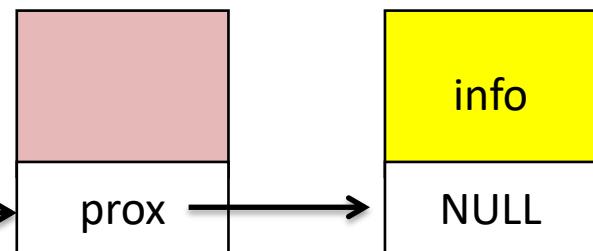
Listas com Cabeça

Tamanho

1

Cabeça

Primeiro



A lista está vazia?

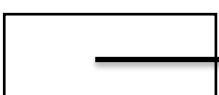
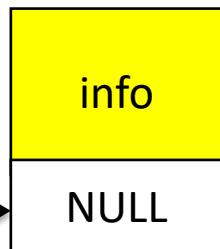
Retirada de Lista com 1 Elemento

Listas sem Cabeça

Tamanho



Primeiro



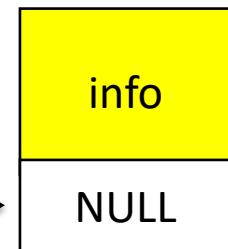
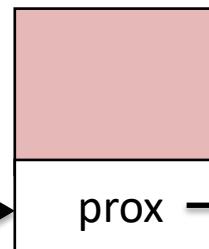
Último

Listas com Cabeça

Anterior



Cabeça



Tamanho



Primeiro



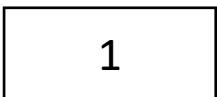
Último



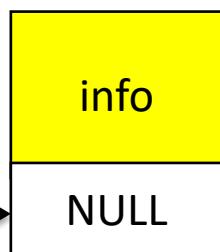
Retirada de Lista com 1 Elemento

Listas sem Cabeça

Tamanho



Primeiro



X

Último

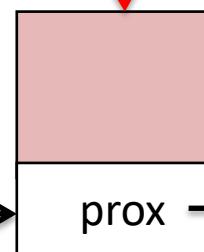


Listas com Cabeça

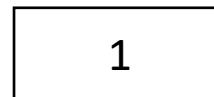
Anterior



Cabeça



Tamanho



Primeiro



info

NULL

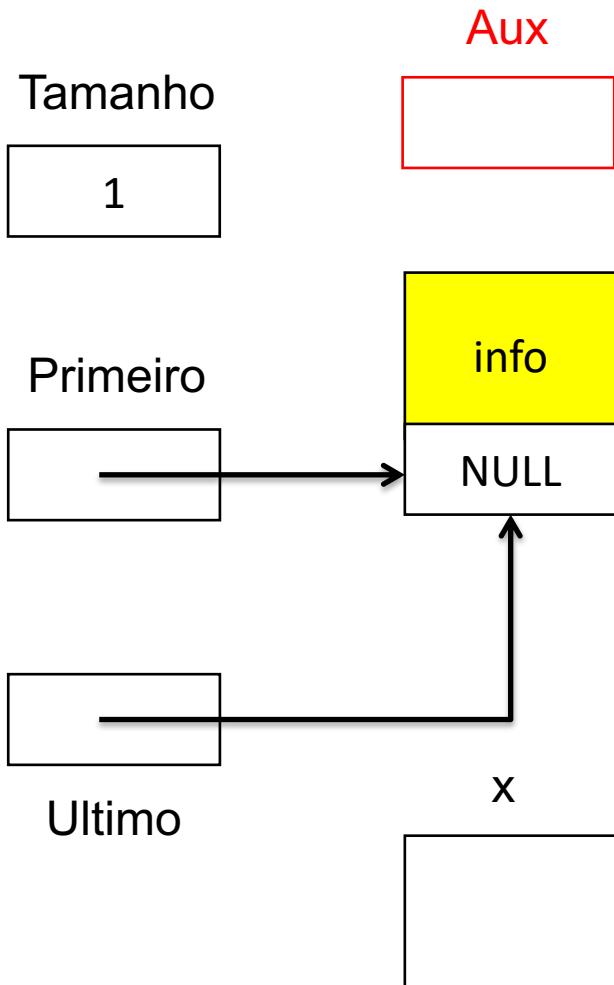


Último

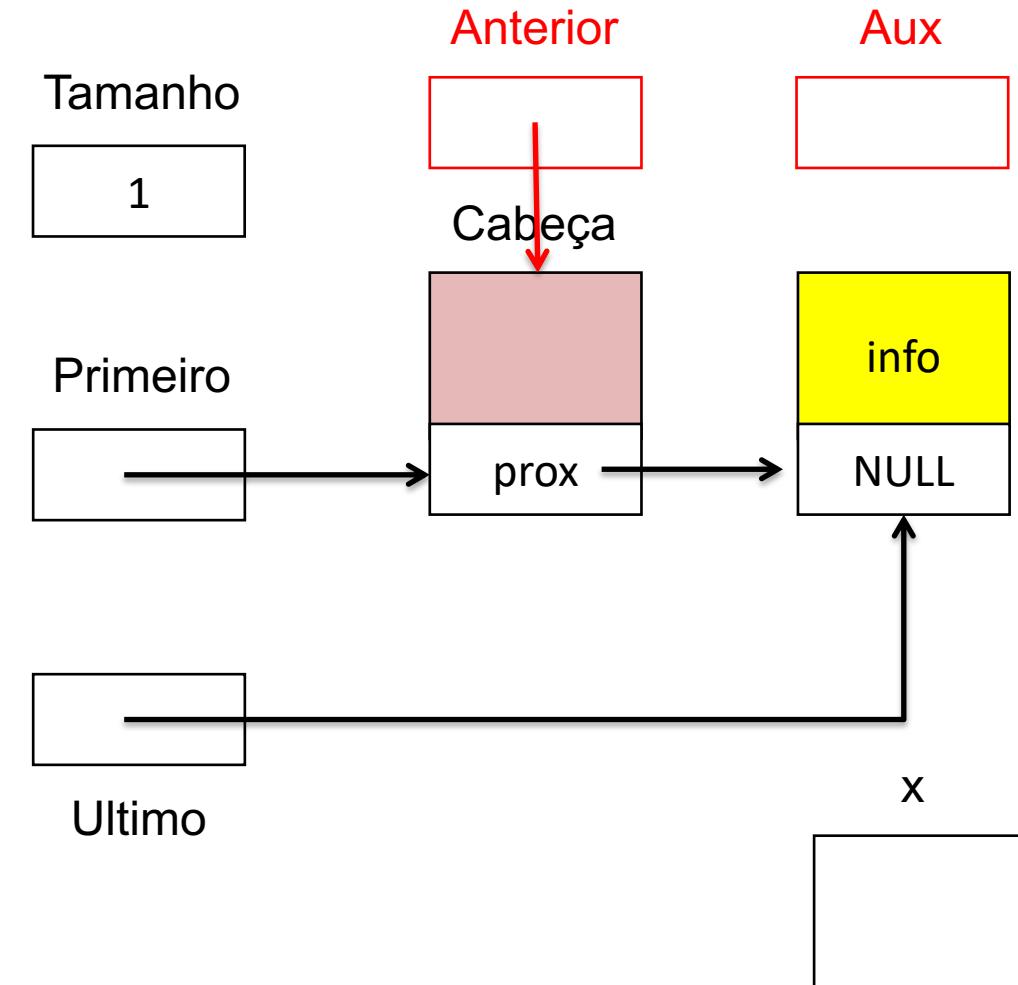


Retirada de Lista com 1 Elemento

Listas sem Cabeça

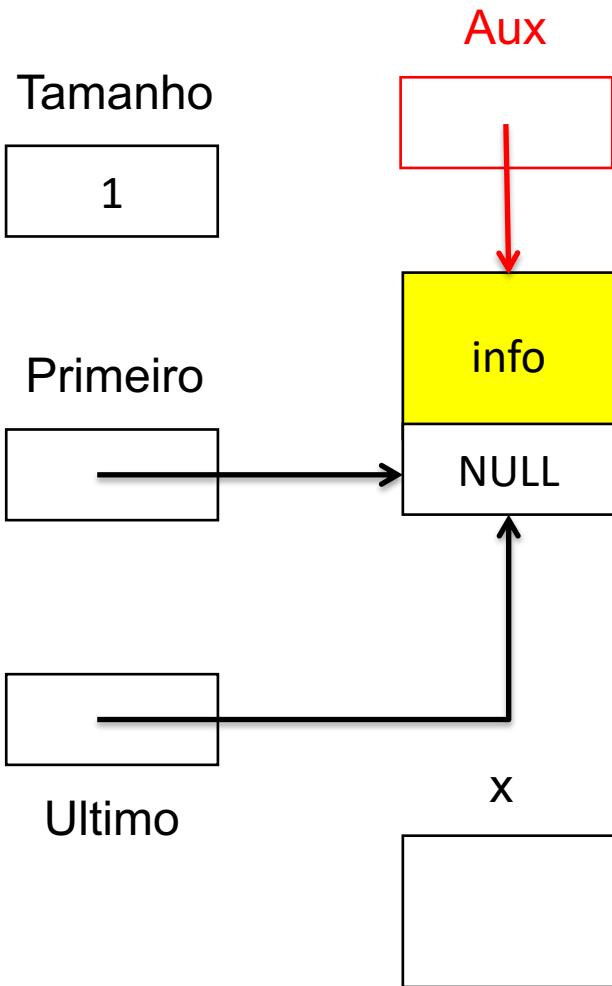


Listas com Cabeça

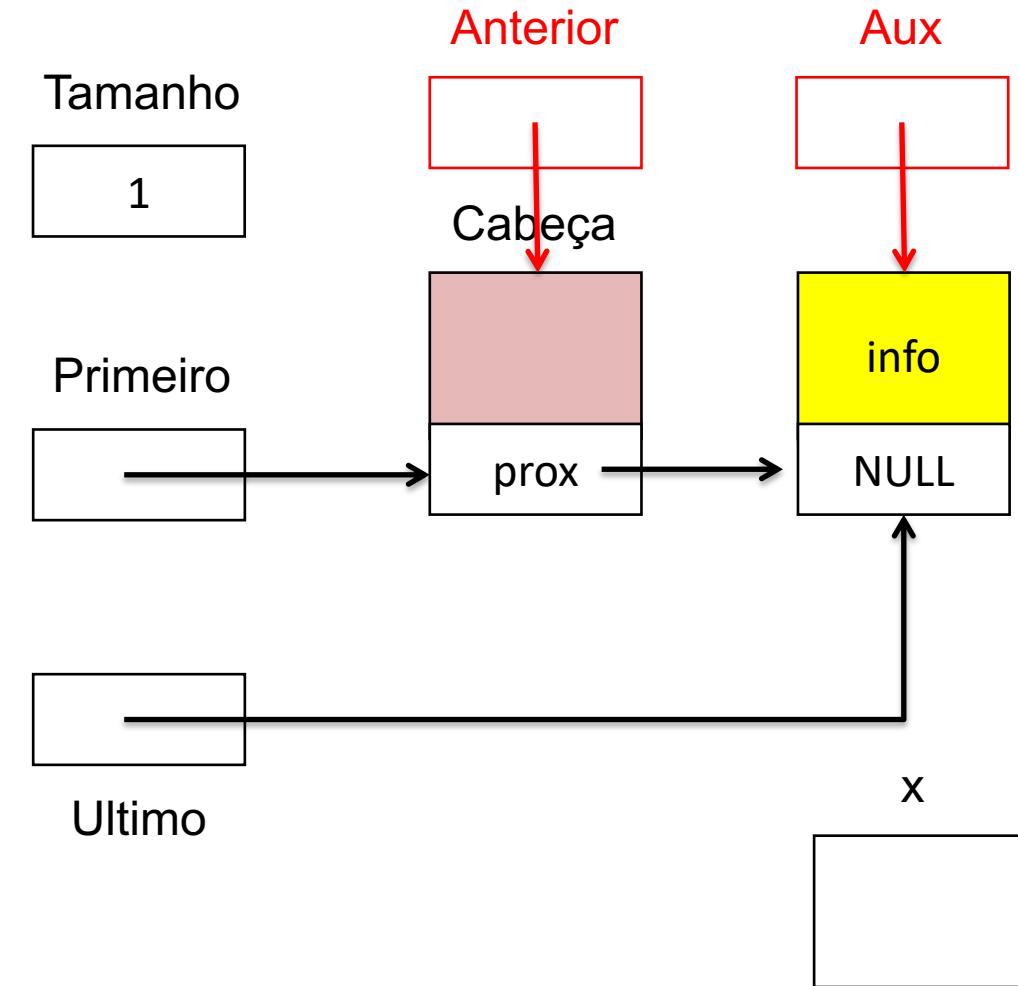


Retirada de Lista com 1 Elemento

Listas sem Cabeça

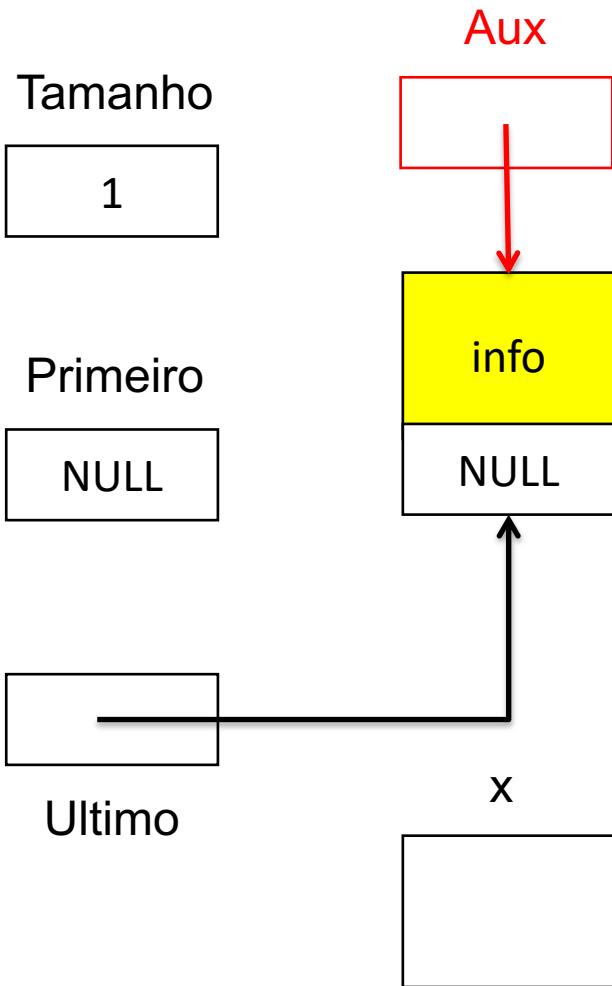


Listas com Cabeça

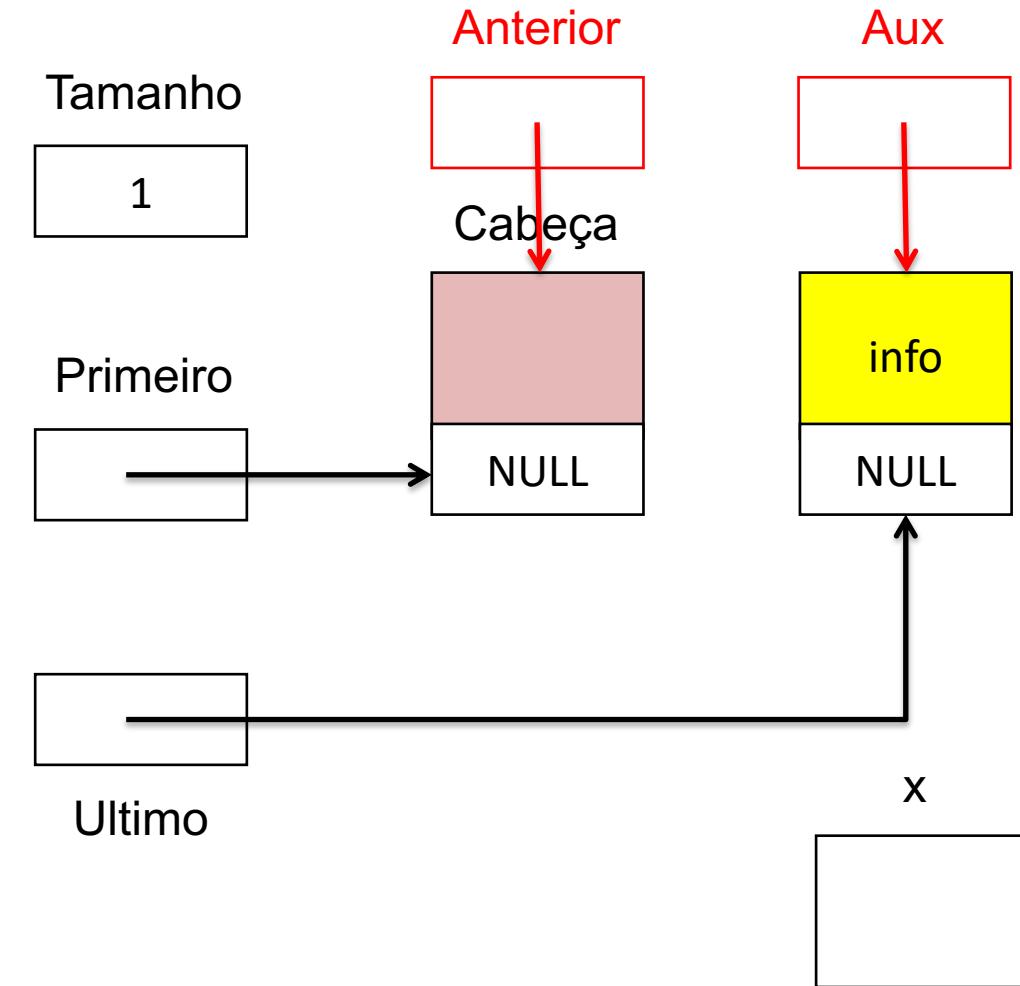


Retirada de Lista com 1 Elemento

Listas sem Cabeça

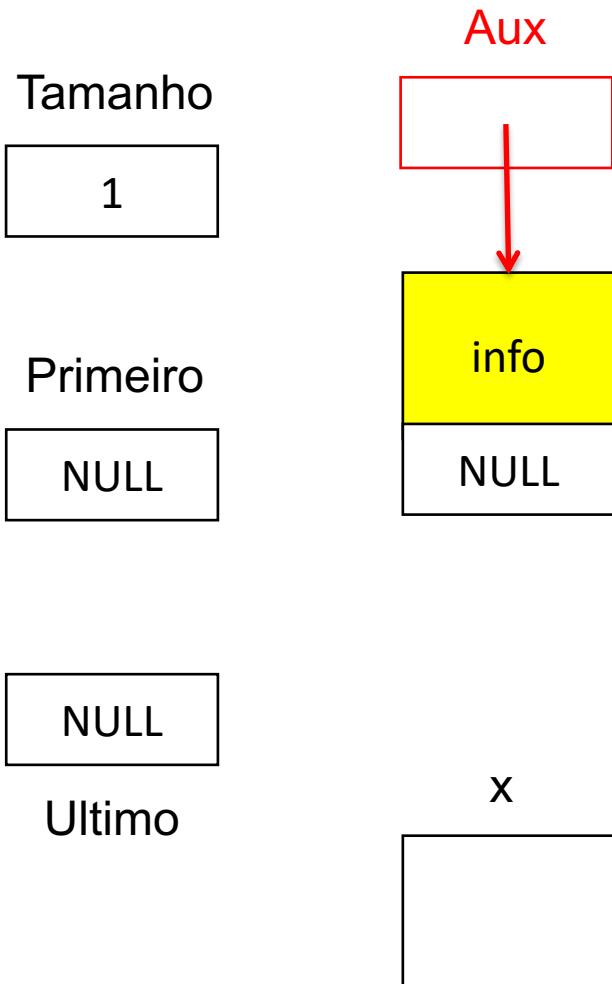


Listas com Cabeça

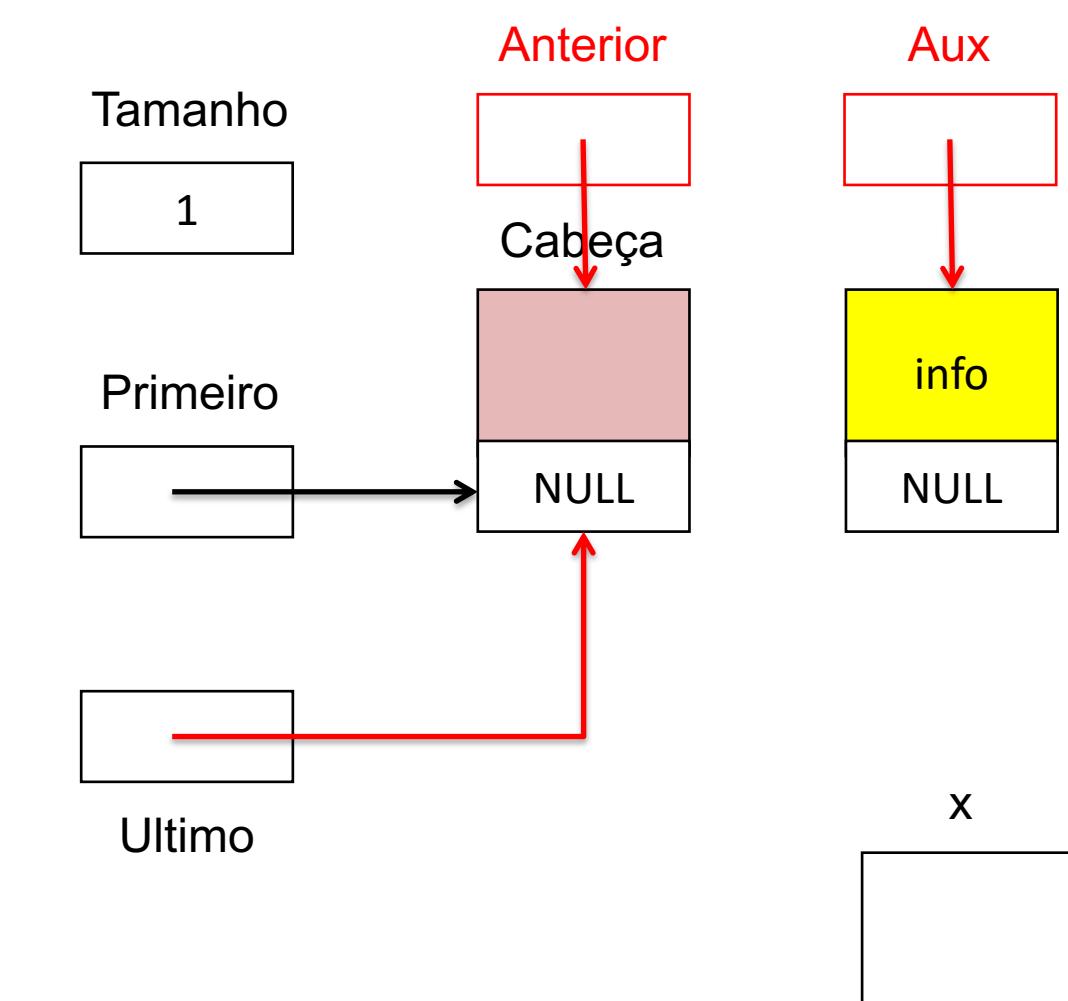


Retirada de Lista com 1 Elemento

Listas sem Cabeça

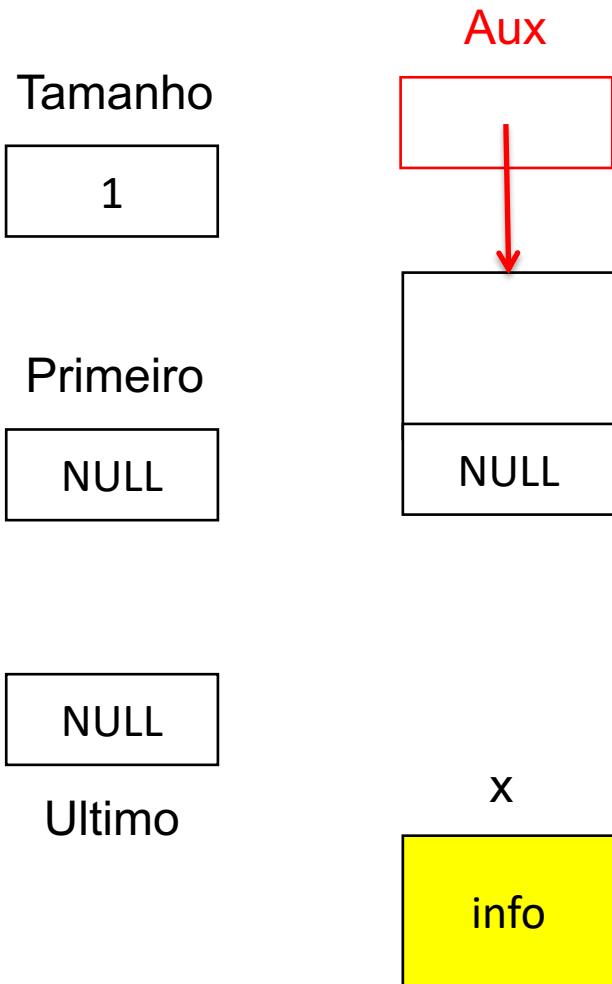


Listas com Cabeça

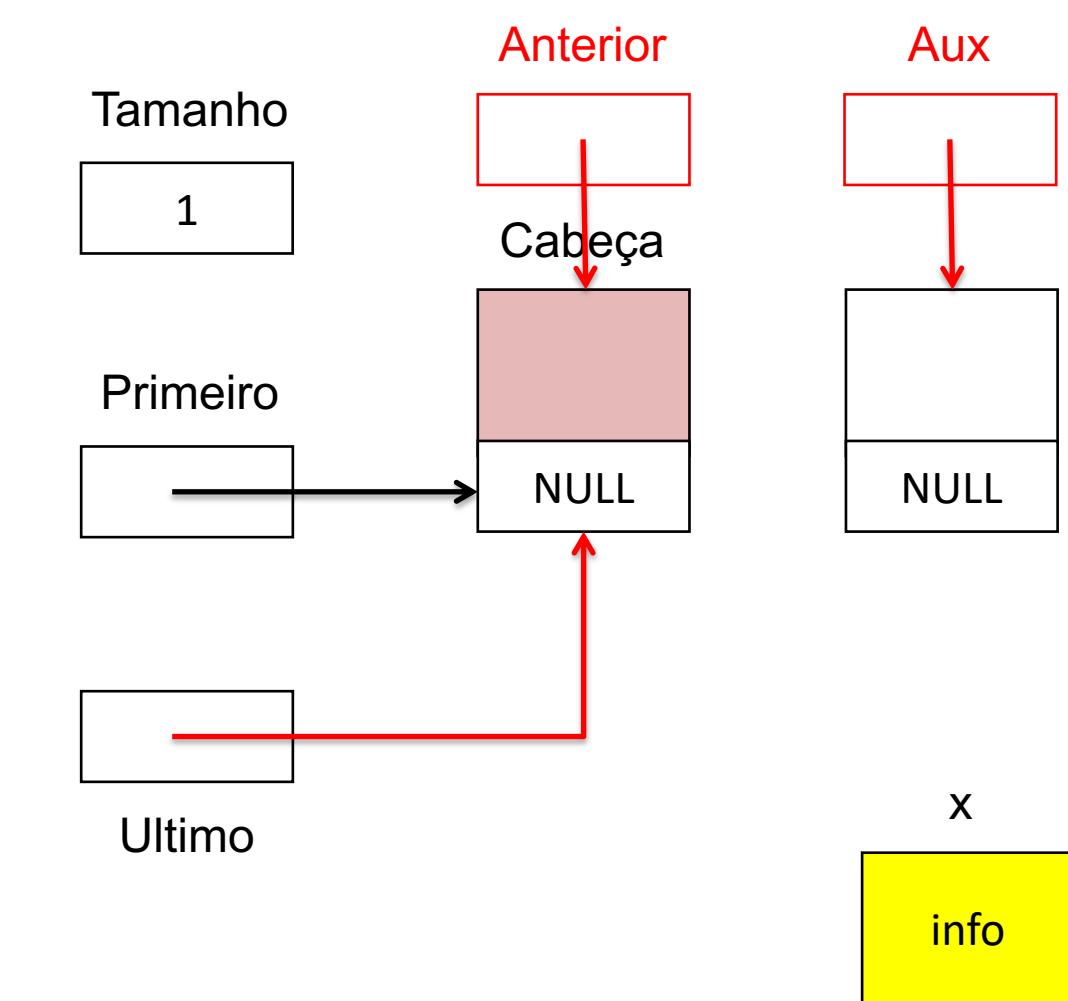


Retirada de Lista com 1 Elemento

Listas sem Cabeça

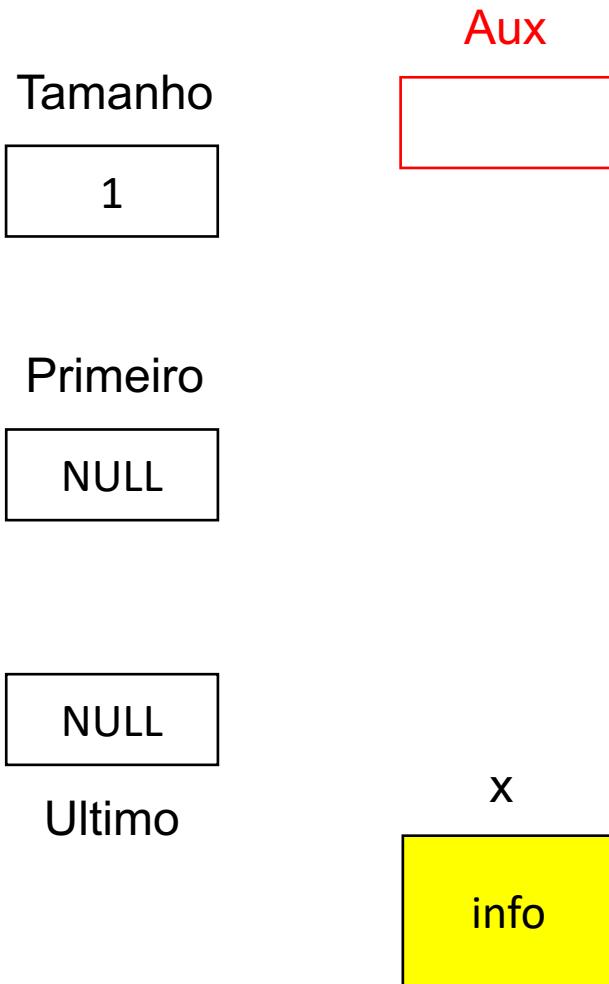


Listas com Cabeça

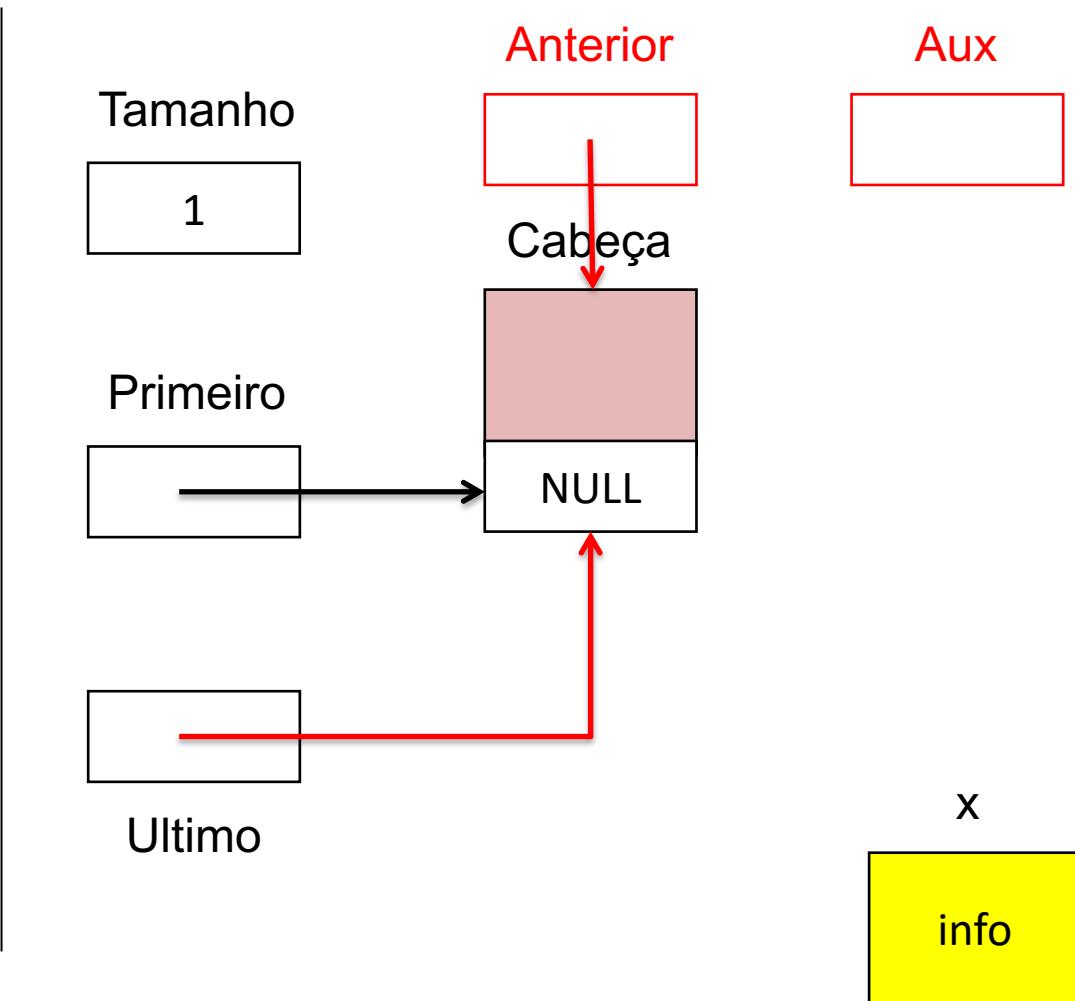


Retirada de Lista com 1 Elemento

Lista sem Cabeça

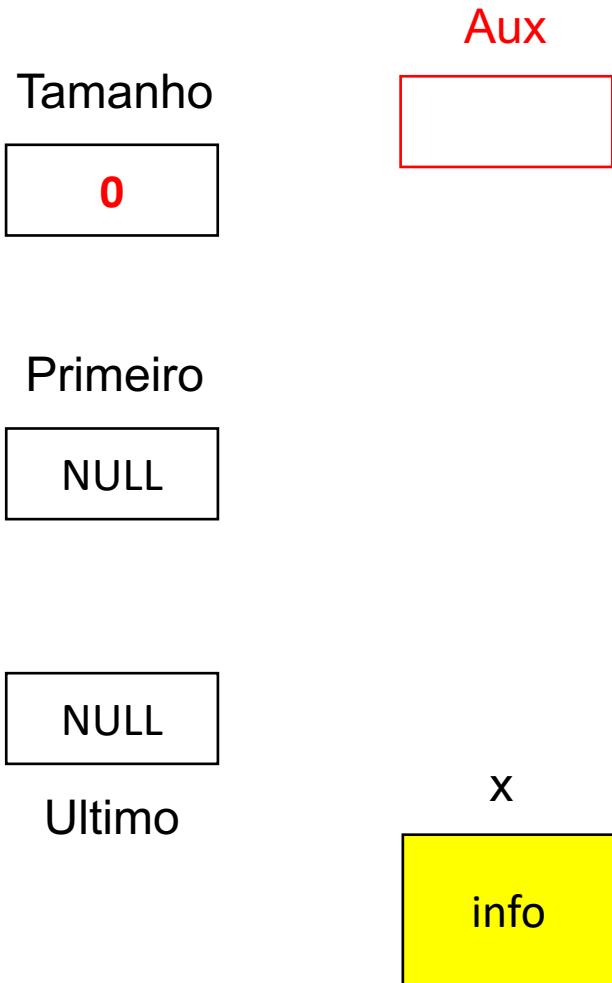


Lista com Cabeça

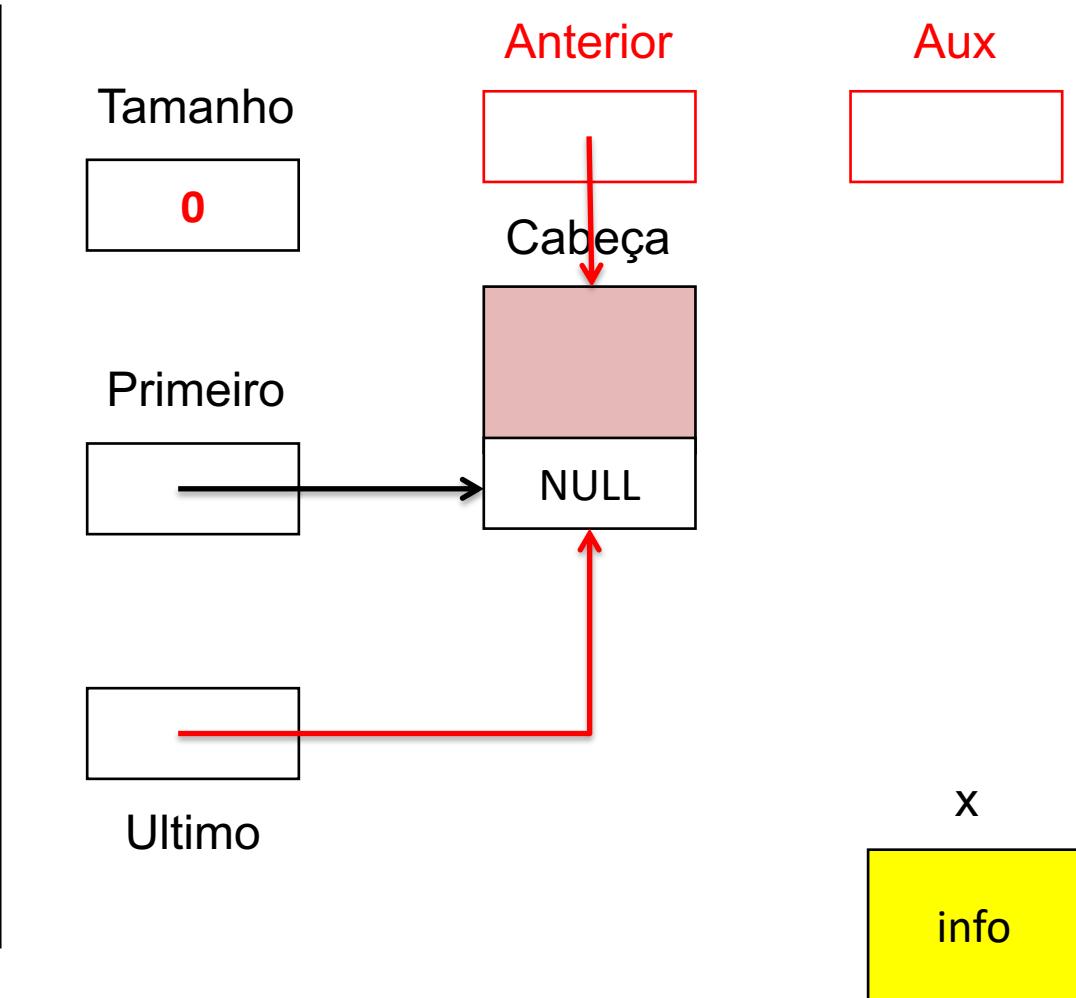


Retirada de Lista com 1 Elemento

Listas sem Cabeça



Listas com Cabeça



Retirada de Lista com 1 Elemento

Listas sem Cabeça

Tamanho

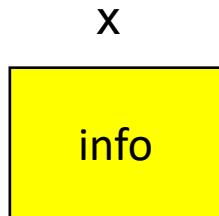
0

Primeiro

NULL

NULL

Último



Listas com Cabeça

Tamanho

0

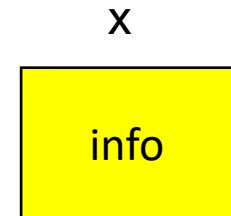
Cabeça

Primeiro

—> NULL

—> NULL

Último



Retirada de Lista com 1 Elemento

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

    pAnterior = pLista->Primeiro;
    while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
        pAnterior = pAnterior->Prox;
    if (pAnterior->Prox != p)
        return 0;

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

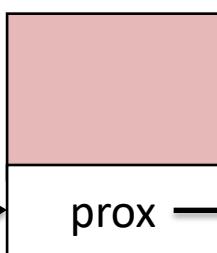
Encontrar uma Posição na Lista

Tamanho

3

Cabeça

Primeiro



Último

p

2

Encontrar uma Posição na Lista

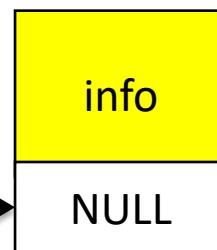
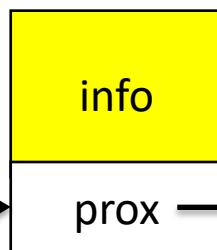
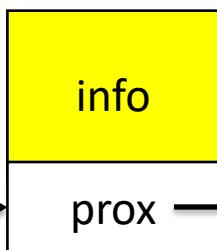
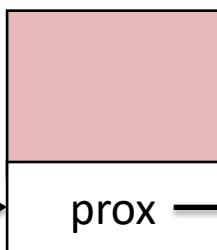
Tamanho

3

Cabeça

Primeiro

prox



Último

p

2

Apontador

Posicao

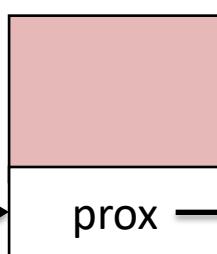
Encontrar uma Posição na Lista

Tamanho

3

Cabeça

Primeiro



Último

p

2

Apontador

Posicao

1

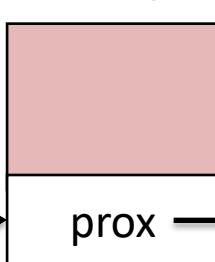
Encontrar uma Posição na Lista

Tamanho

3

Cabeça

Primeiro



Último

p

2

Apontador

Posicao

2

Encontrar uma Posição na Lista

```
/* 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->Prox;
    while ((pApontador != NULL) && (Posicao != p)) {
        pApontador = pApontador->Prox;
        Posicao++;
    }

    return pApontador;
}
```

■ Problema:

- O apontador **Primeiro** não é alterado após a inicialização
- É necessário manter um apontador para a primeira célula

■ Solução:

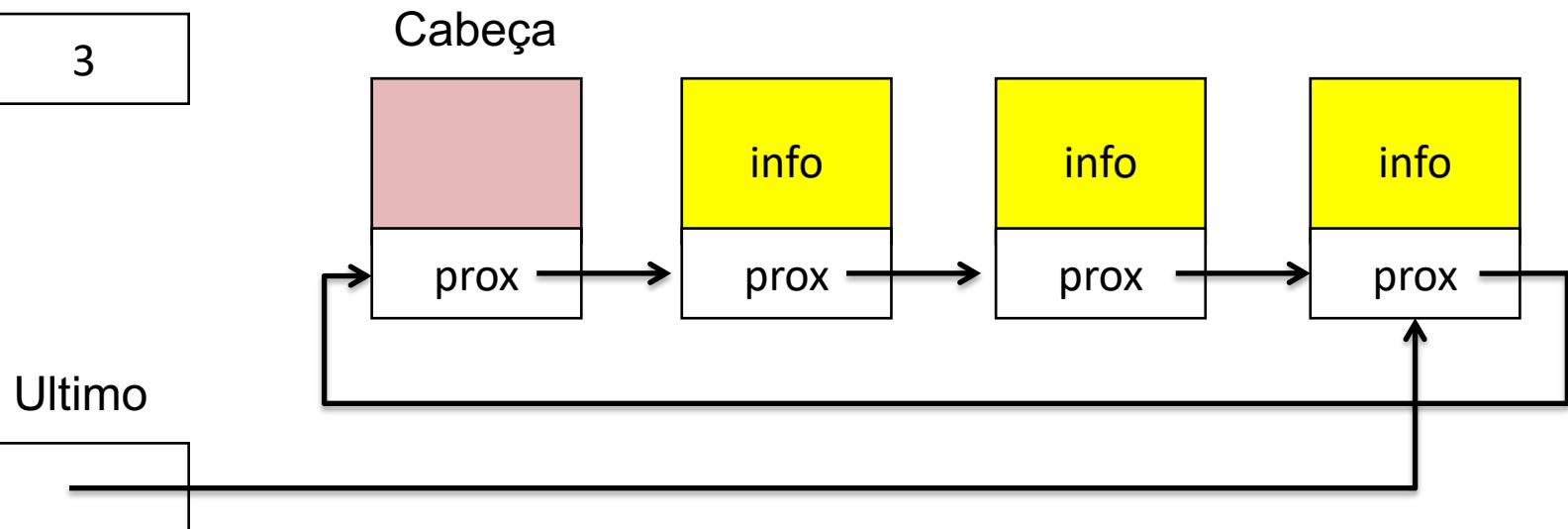
- Fazer com que o campo **Prox** da célula indicada pelo apontador **Ultimo** armazene o endereço da primeira célula da lista, em vez do endereço NULL, formando um **ciclo**

■ Listas Circulares

- A última célula mantém um apontador para primeira célula
- O primeiro item está na célula **Ultimo->Prox->Prox**

Tamanho

3



Estrutura da Lista Circular

```
typedef int TChave;

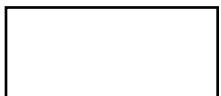
typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct SCelula *TApontador;

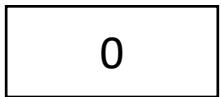
typedef struct SCelula {
    TItem Item;
    TApontador Prox;
} TCelula;

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

Cria Lista Vazia



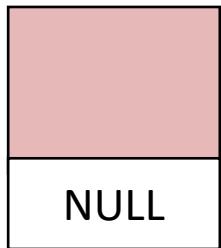
Último



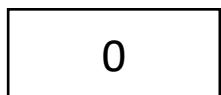
Tamanho

Cria Lista Vazia

Cabeça



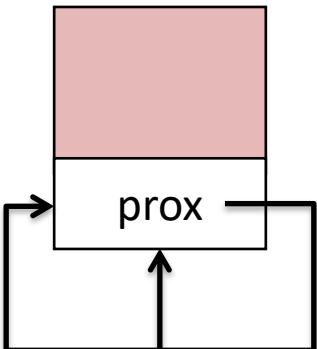
Ultimo



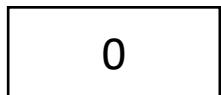
Tamanho

Cria Lista Vazia

Cabeça



Ultimo



Tamanho

Cria Lista Vazia

```
/* Inicia as variaveis da lista */
void TLista_Inicia(TLista *pLista)
{
    pLista->Ultimo = (TApontador) malloc(sizeof(TCelula));
    pLista->Ultimo->Prox = pLista->Ultimo;
    pLista->Tamanho = 0;
}

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

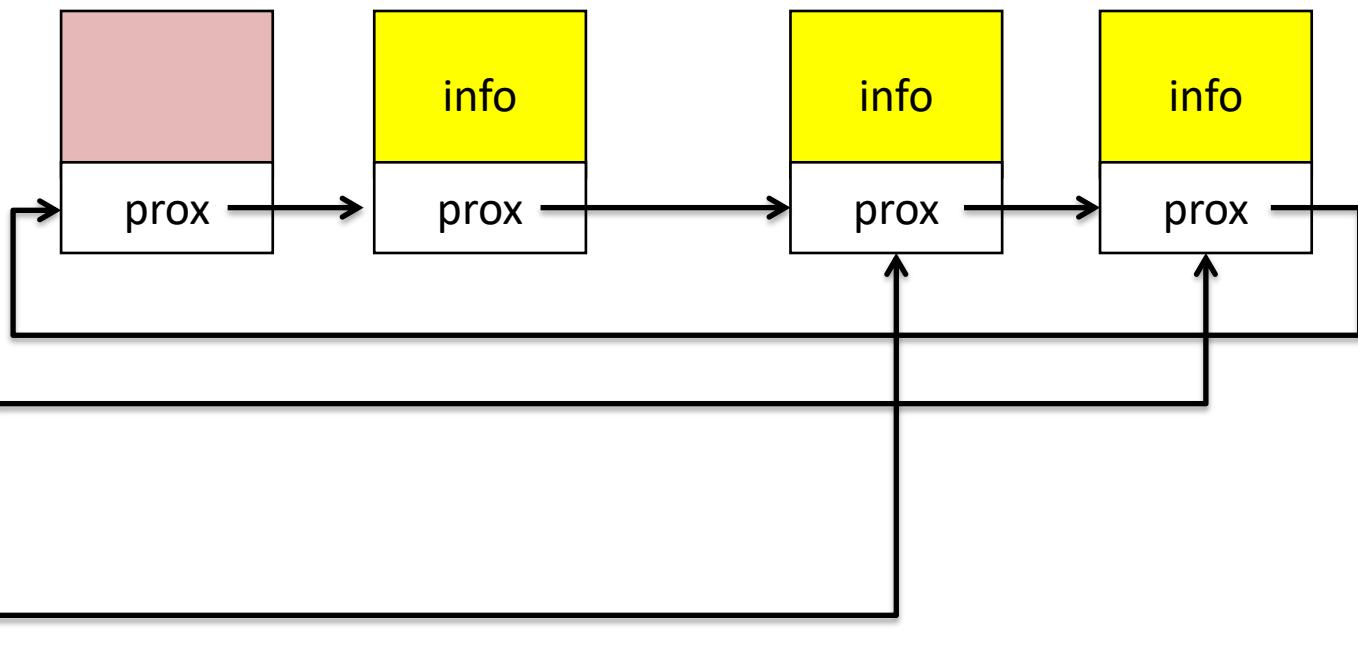
/* Retorna o tamanho da lista */
int TLista_Tamanho(TLista *pLista)
{
    return (pLista->Tamanho);
}
```

Inserção em Posição Qualquer

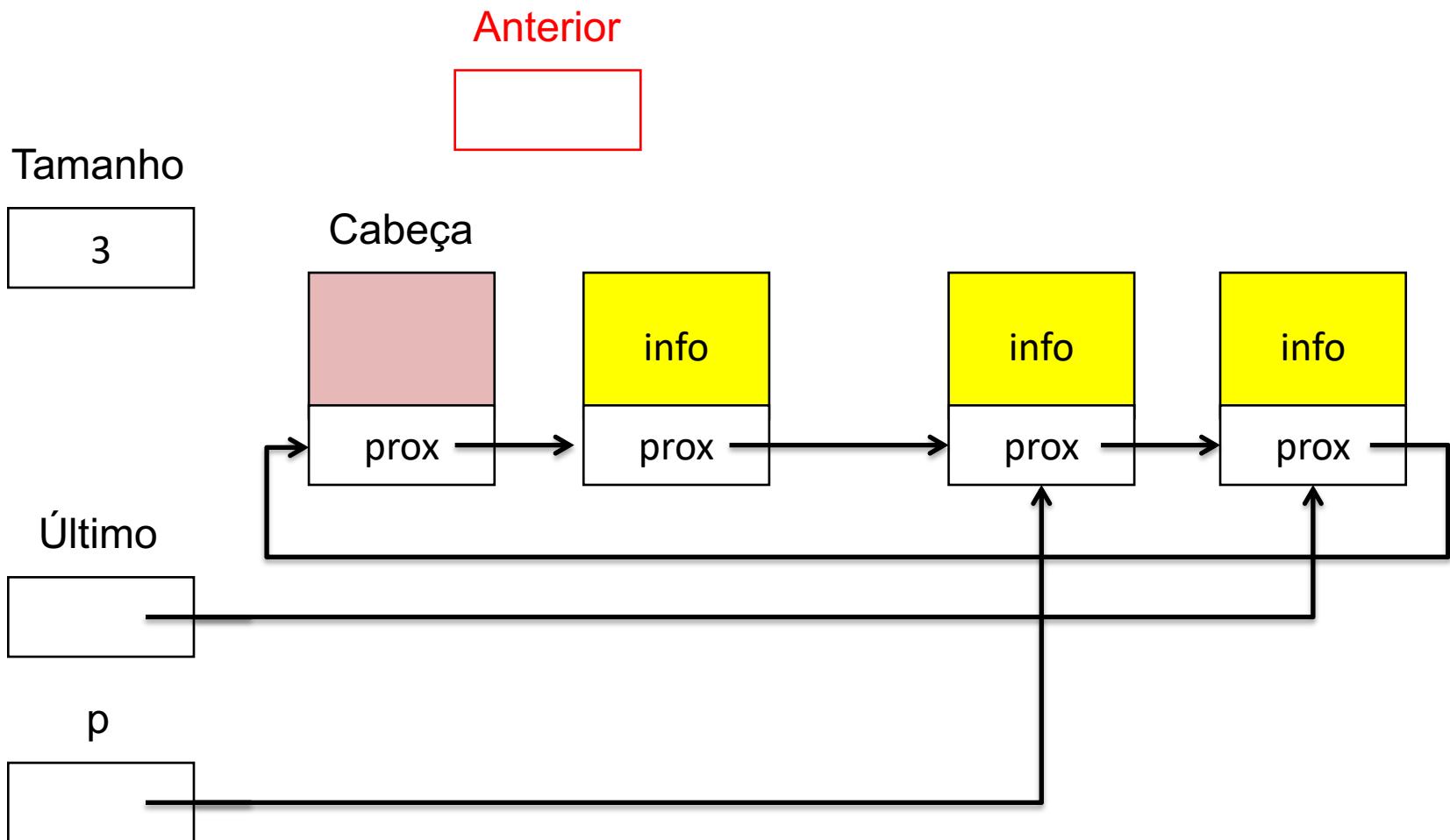
Tamanho

3

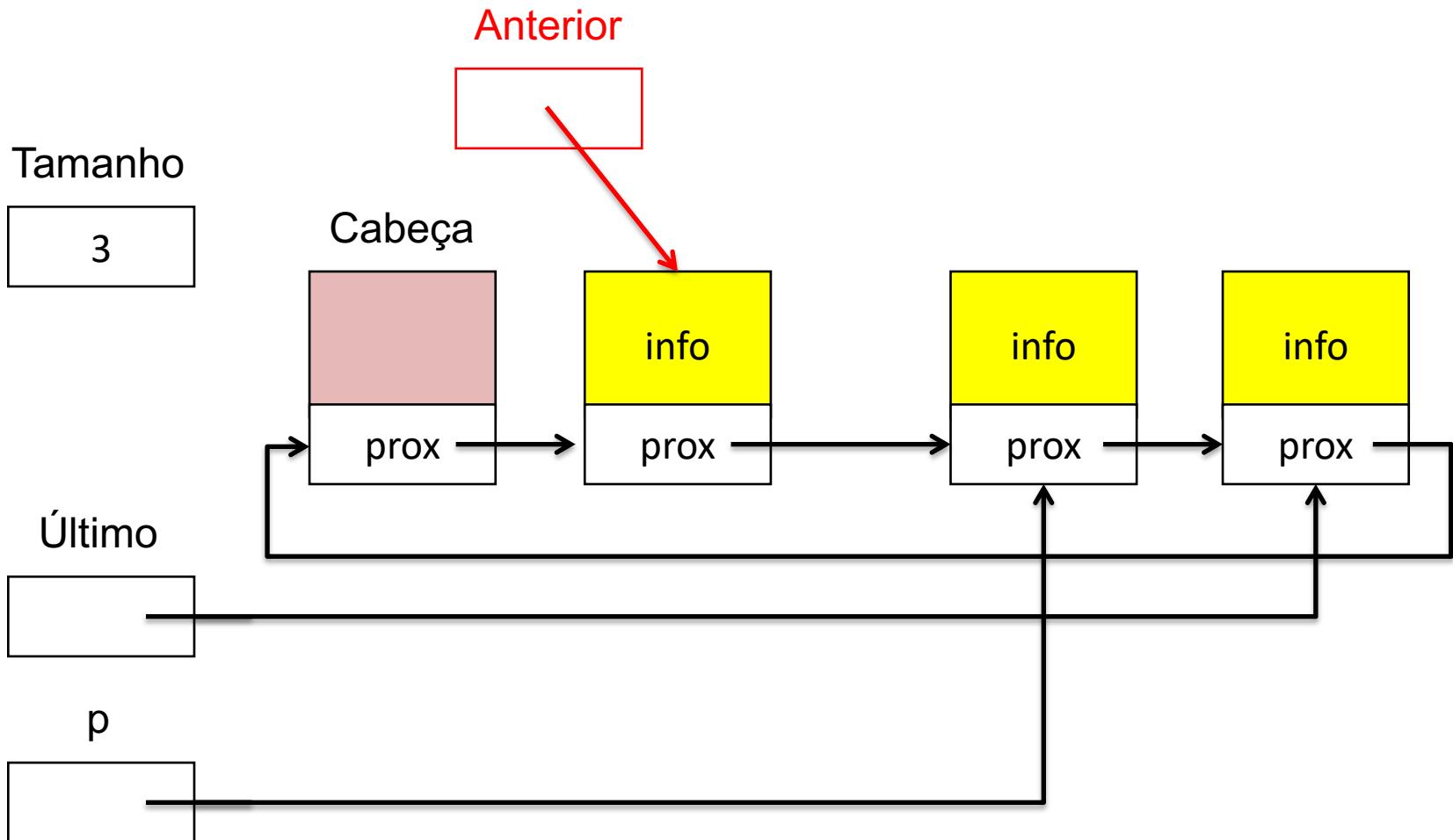
Cabeça



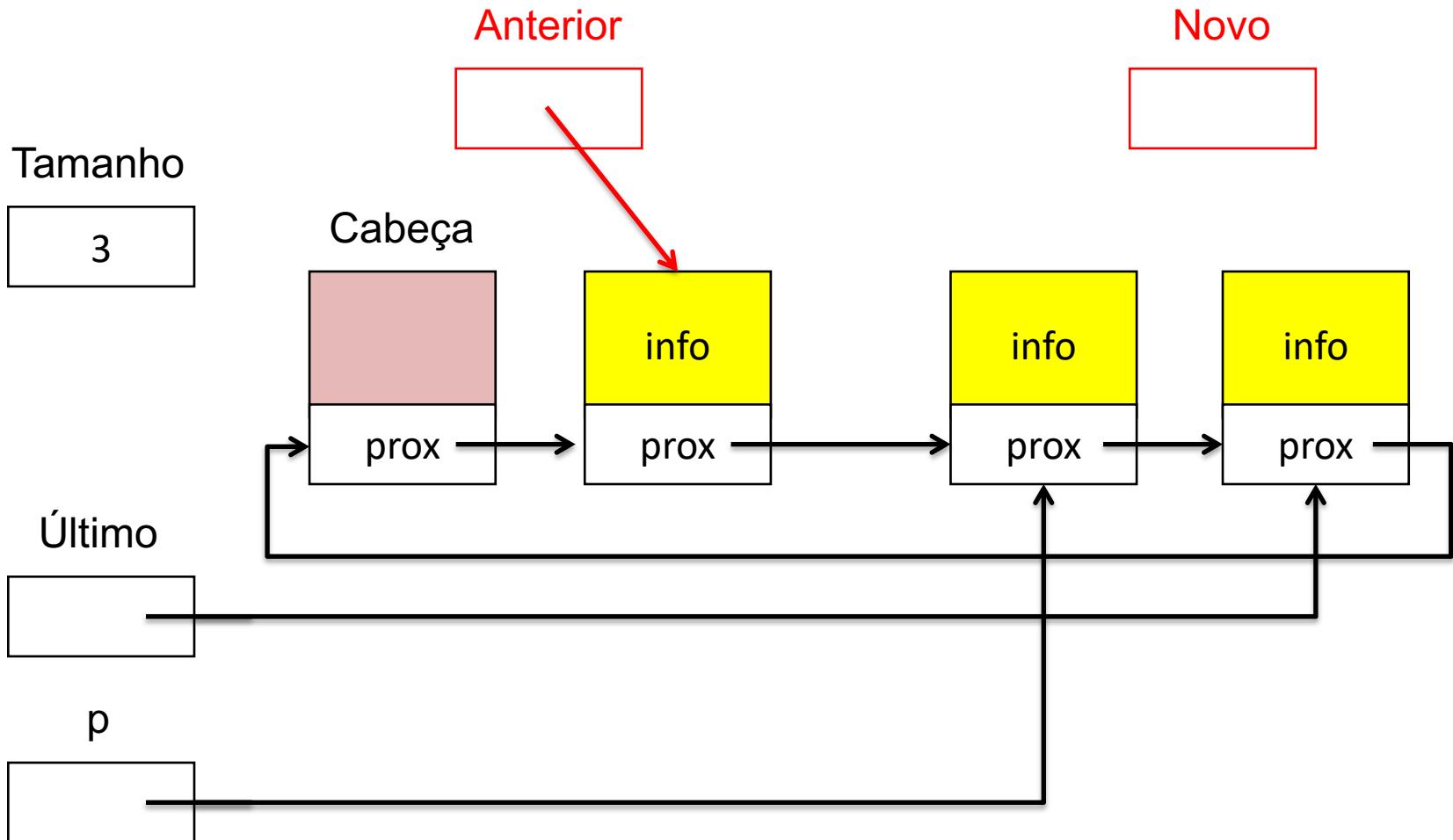
Inserção em Posição Qualquer



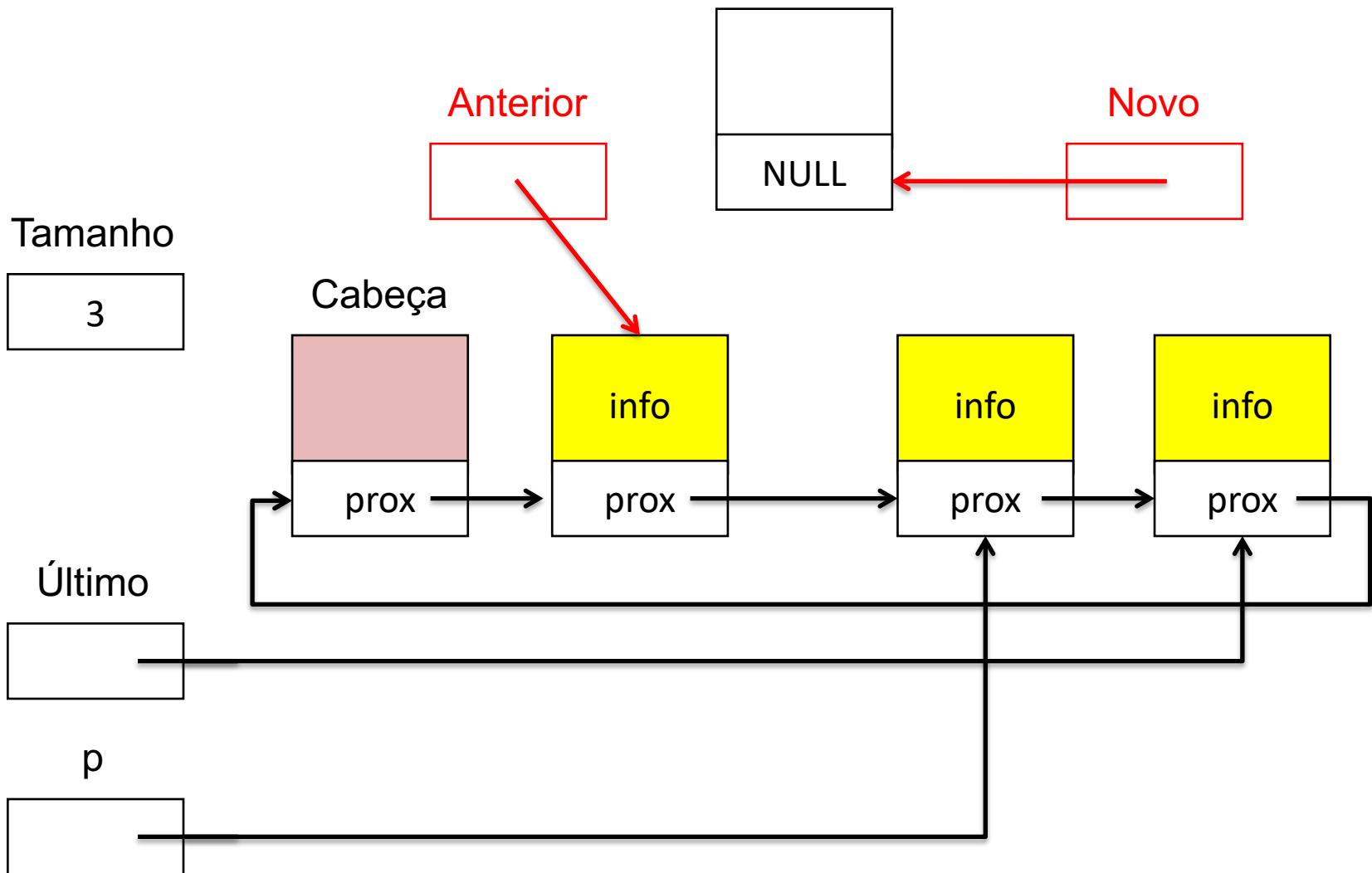
Inserção em Posição Qualquer



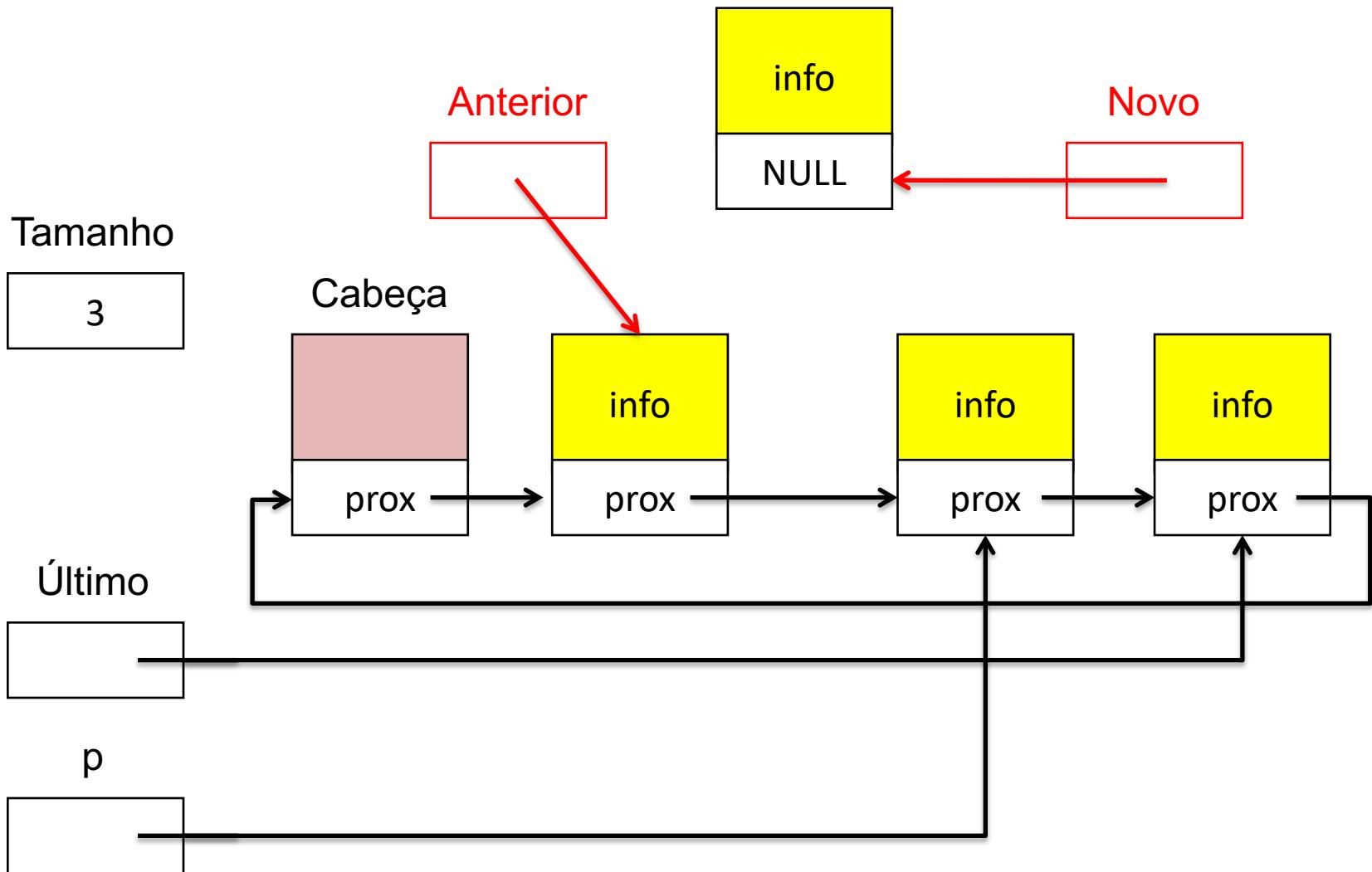
Inserção em Posição Qualquer



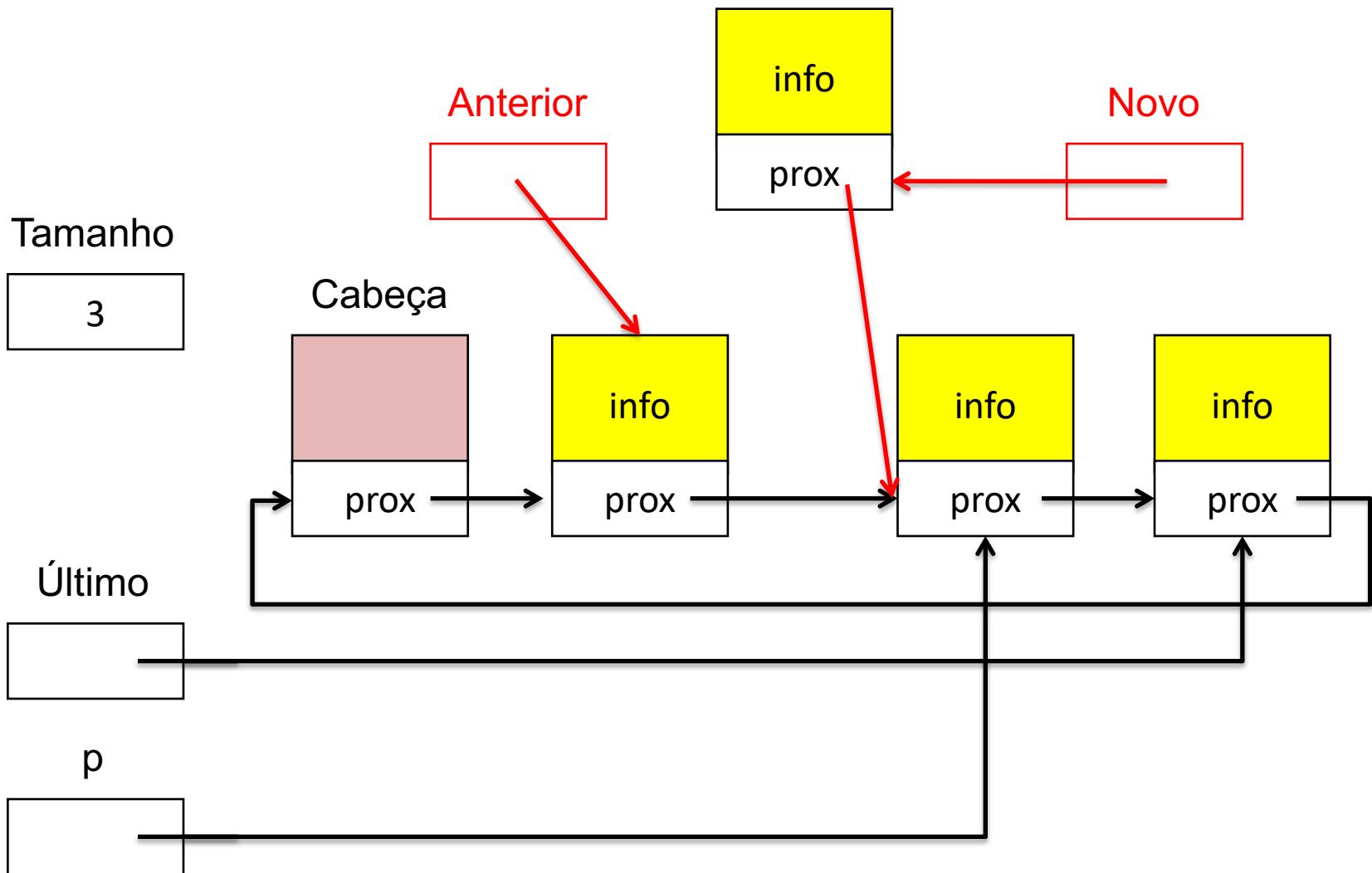
Inserção em Posição Qualquer



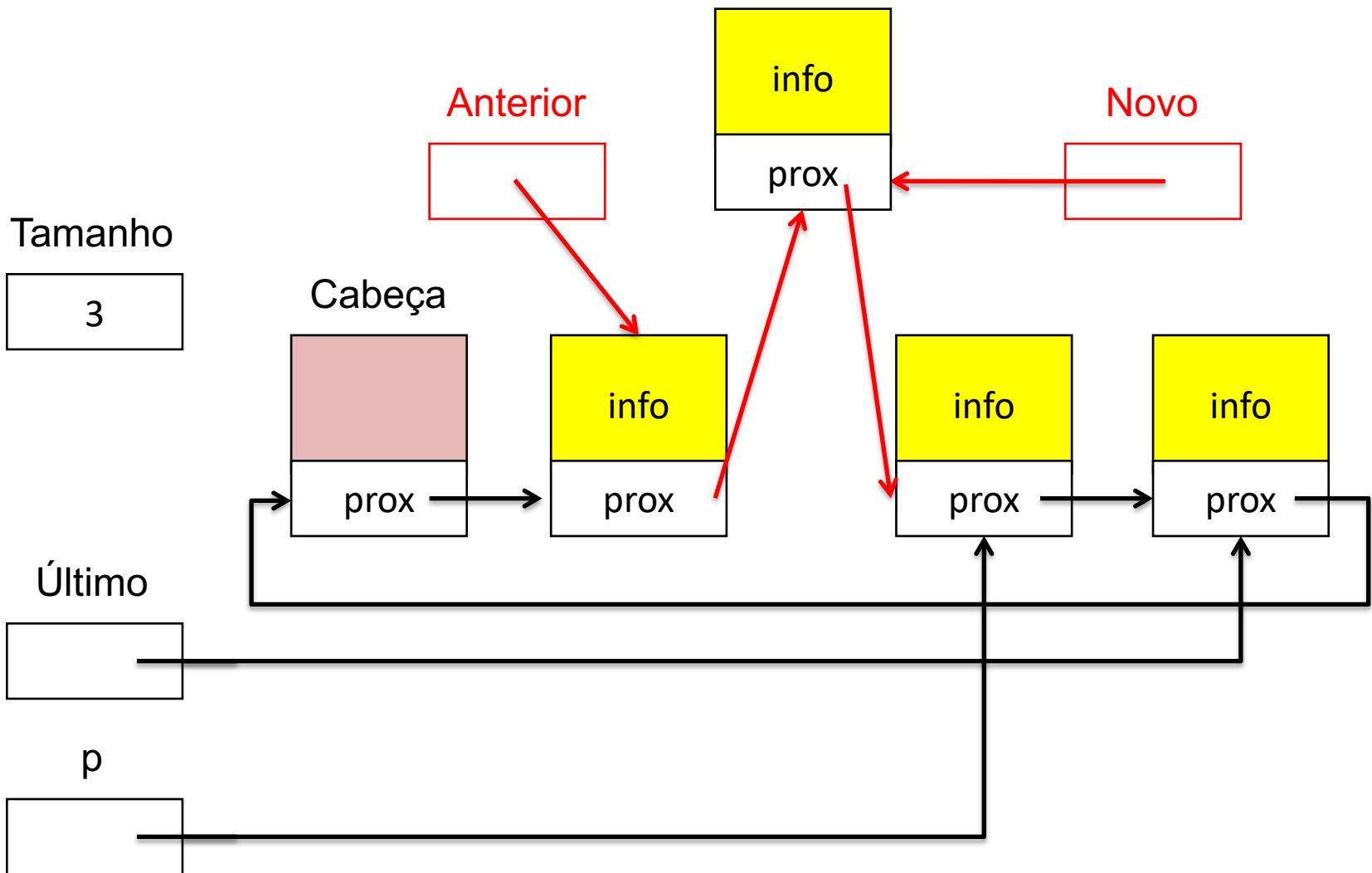
Inserção em Posição Qualquer



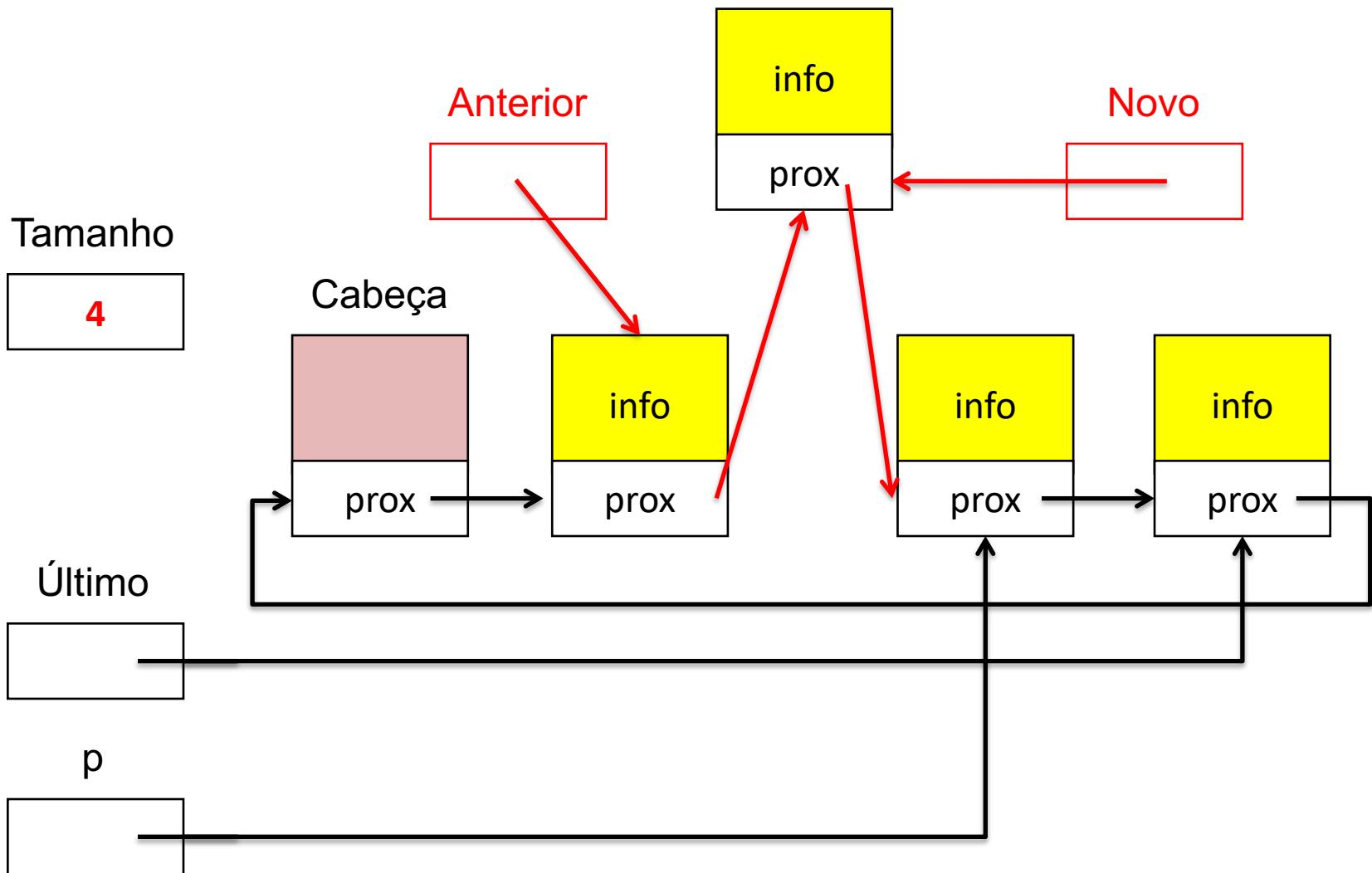
Inserção em Posição Qualquer



Inserção em Posição Qualquer



Inserção em Posição Qualquer



Inserção em Posição Qualquer

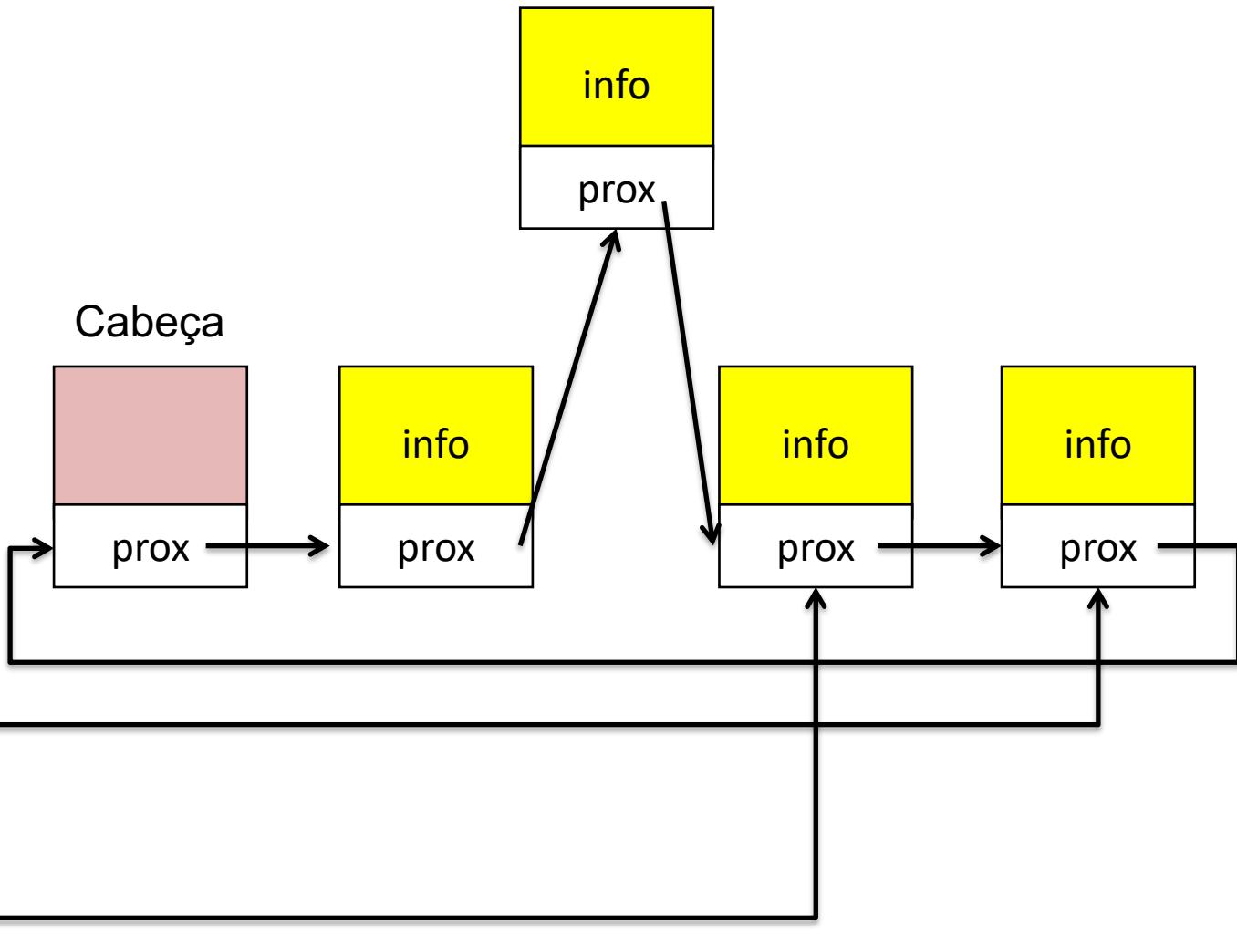
Tamanho

4

Cabeça

Último

p



Inserção em Posição Qualquer

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

    pAnterior = pLista->Ultimo->Prox;
    while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
        pAnterior = pAnterior->Prox;
    if (pAnterior->Prox != p)
        return 0;

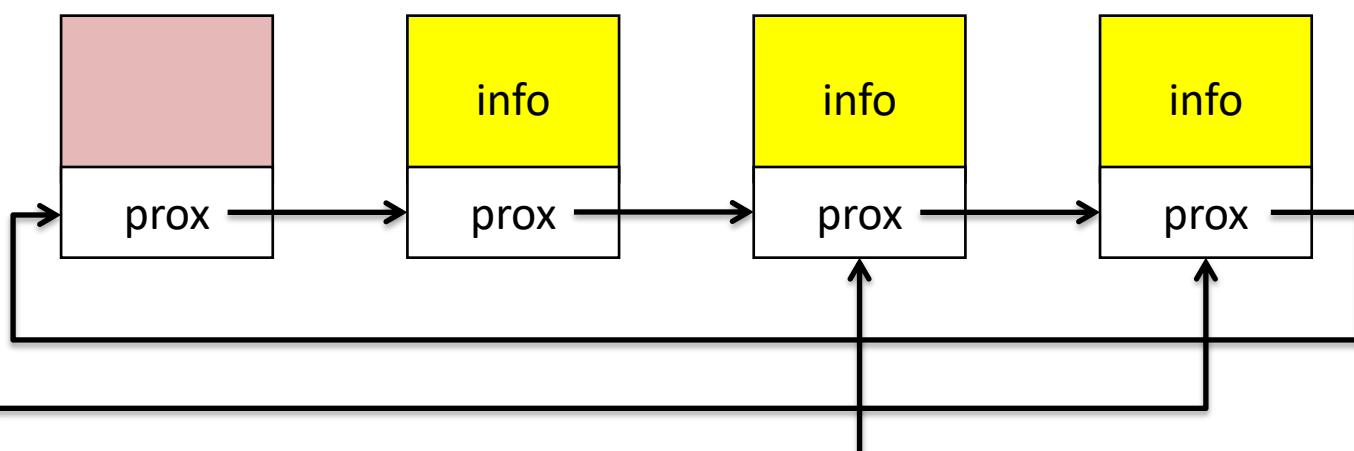
    pNovo = (TApontador) malloc(sizeof(TCelula));
    pNovo->Item = x;
    pNovo->Prox = pAnterior->Prox;
    pAnterior->Prox = pNovo;
    if (pAnterior == pLista->Ultimo)
        pLista->Ultimo = pNovo;
    pLista->Tamanho++;
    return 1;
}
```

Retirada de Posição Qualquer

Tamanho

3

Cabeça



Último

—

p

—

x

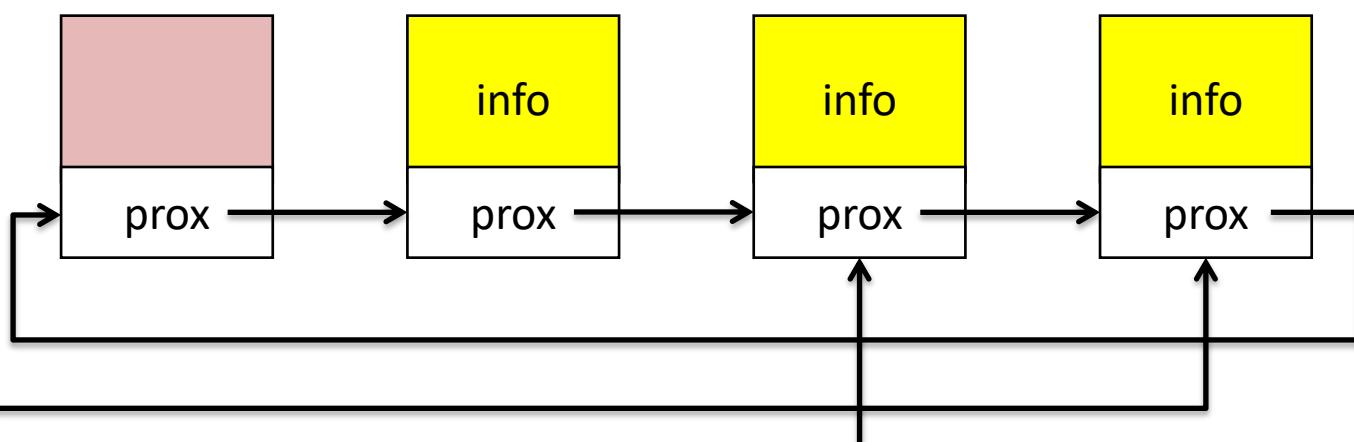
—

Retirada de Posição Qualquer

Tamanho

3

Cabeça



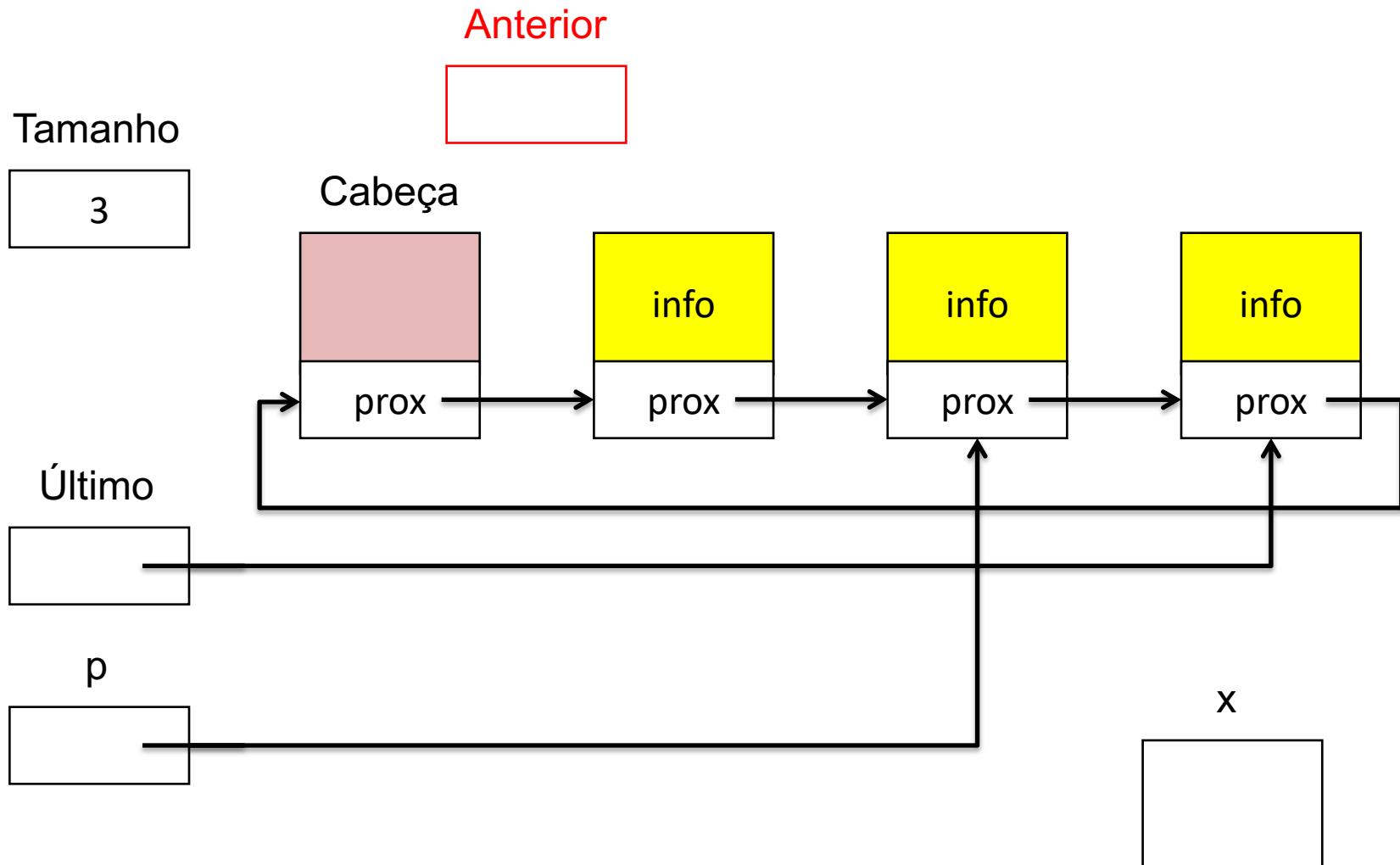
Último

p

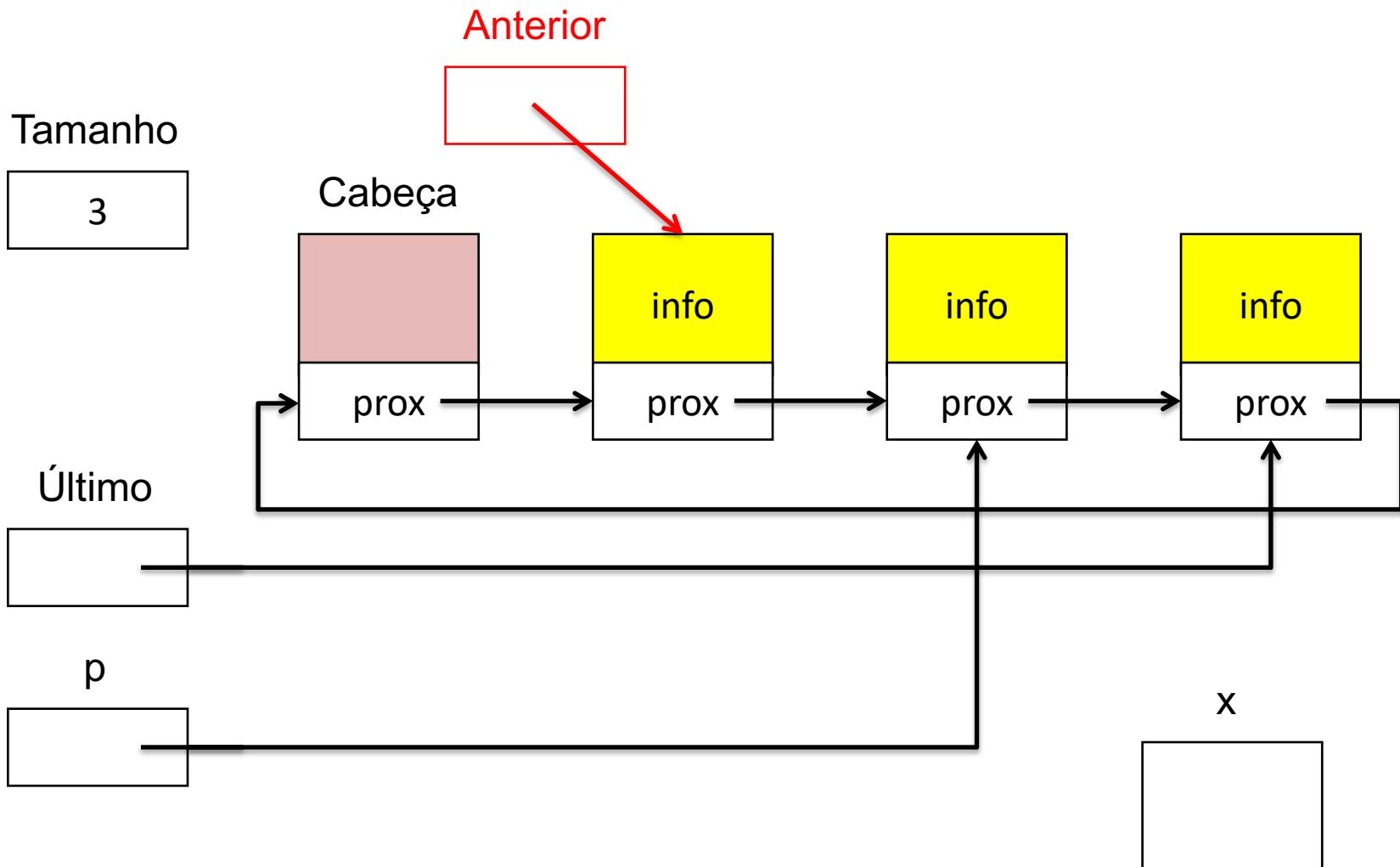
x

A lista está vazia?

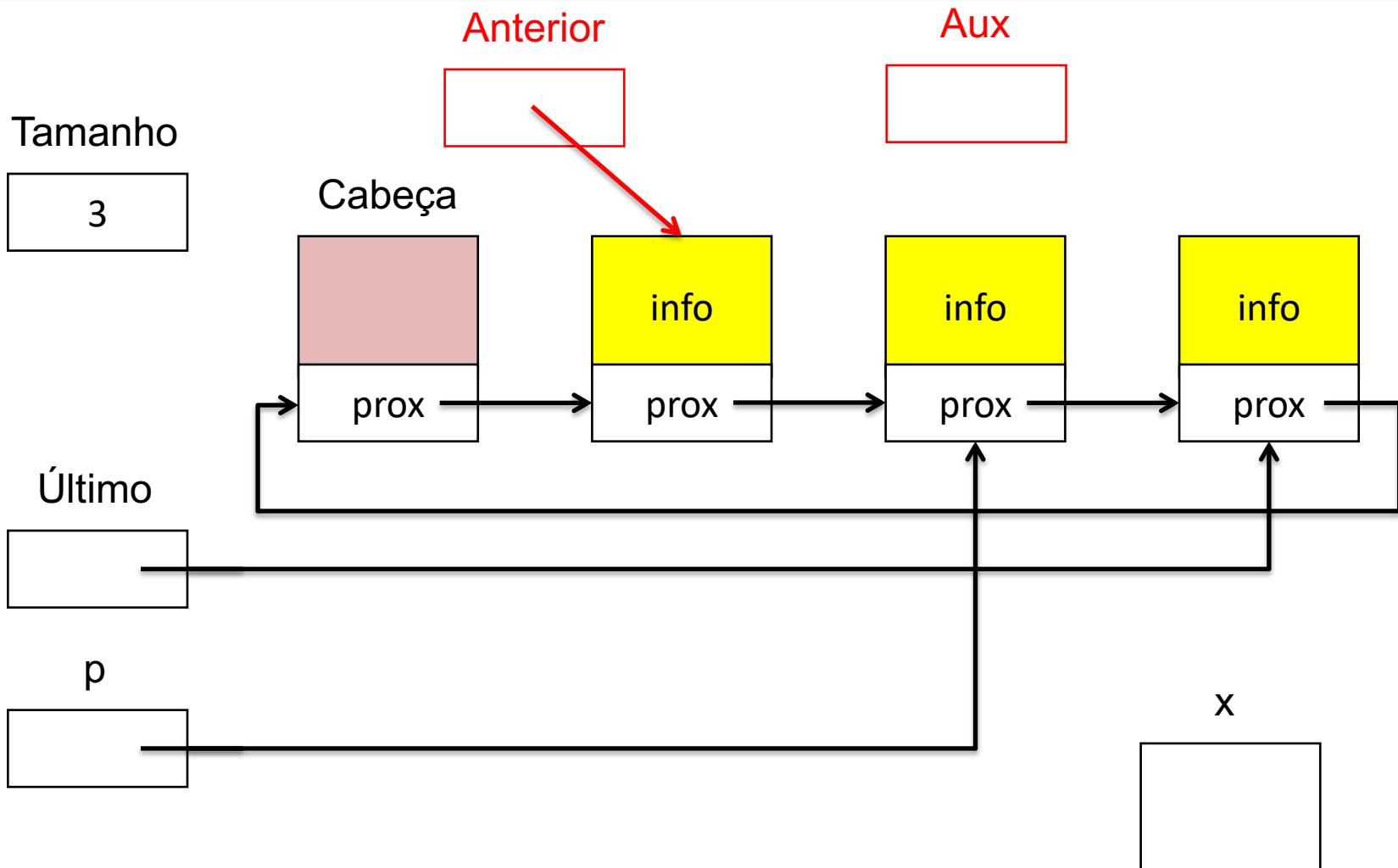
Retirada de Posição Qualquer



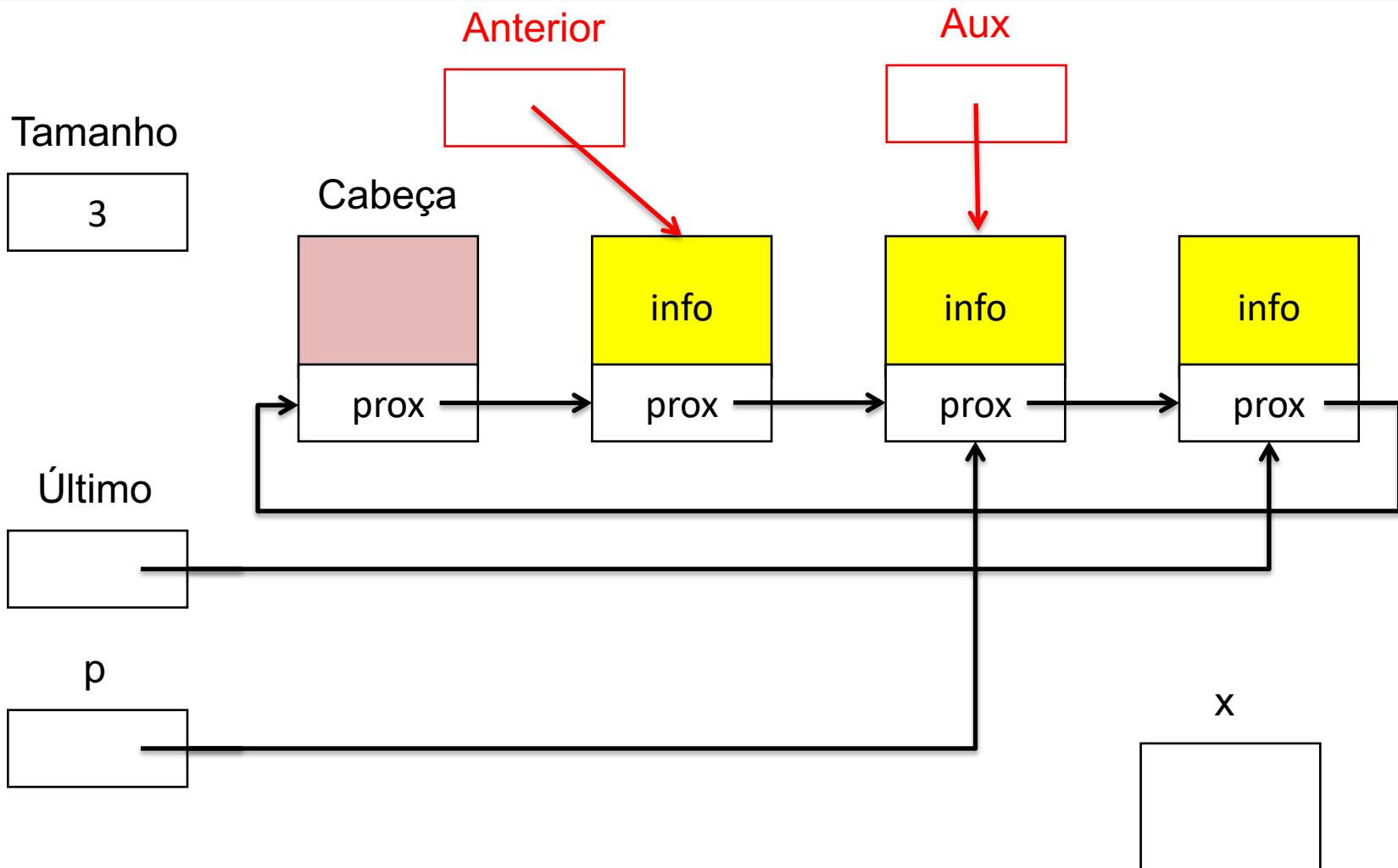
Retirada de Posição Qualquer



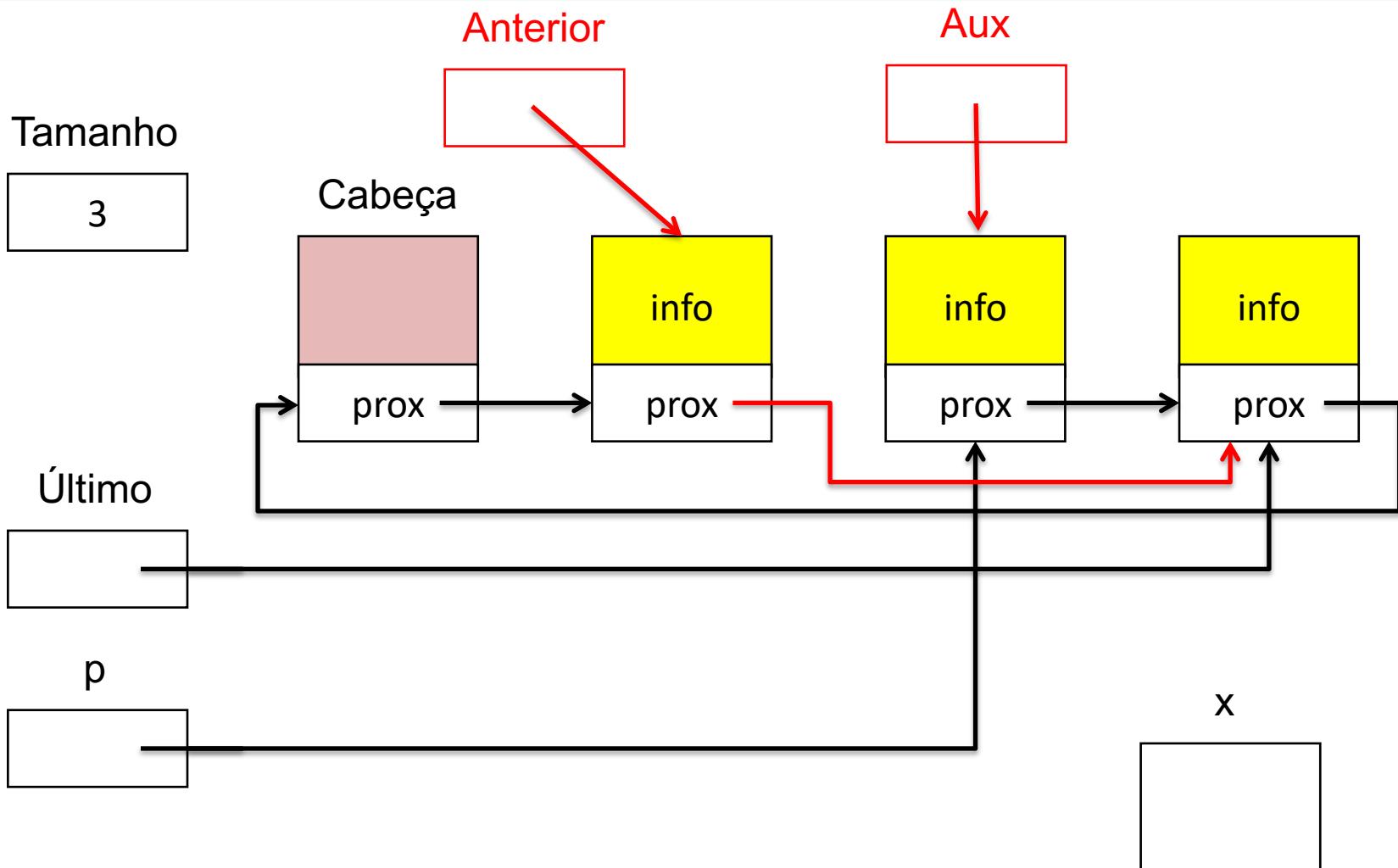
Retirada de Posição Qualquer



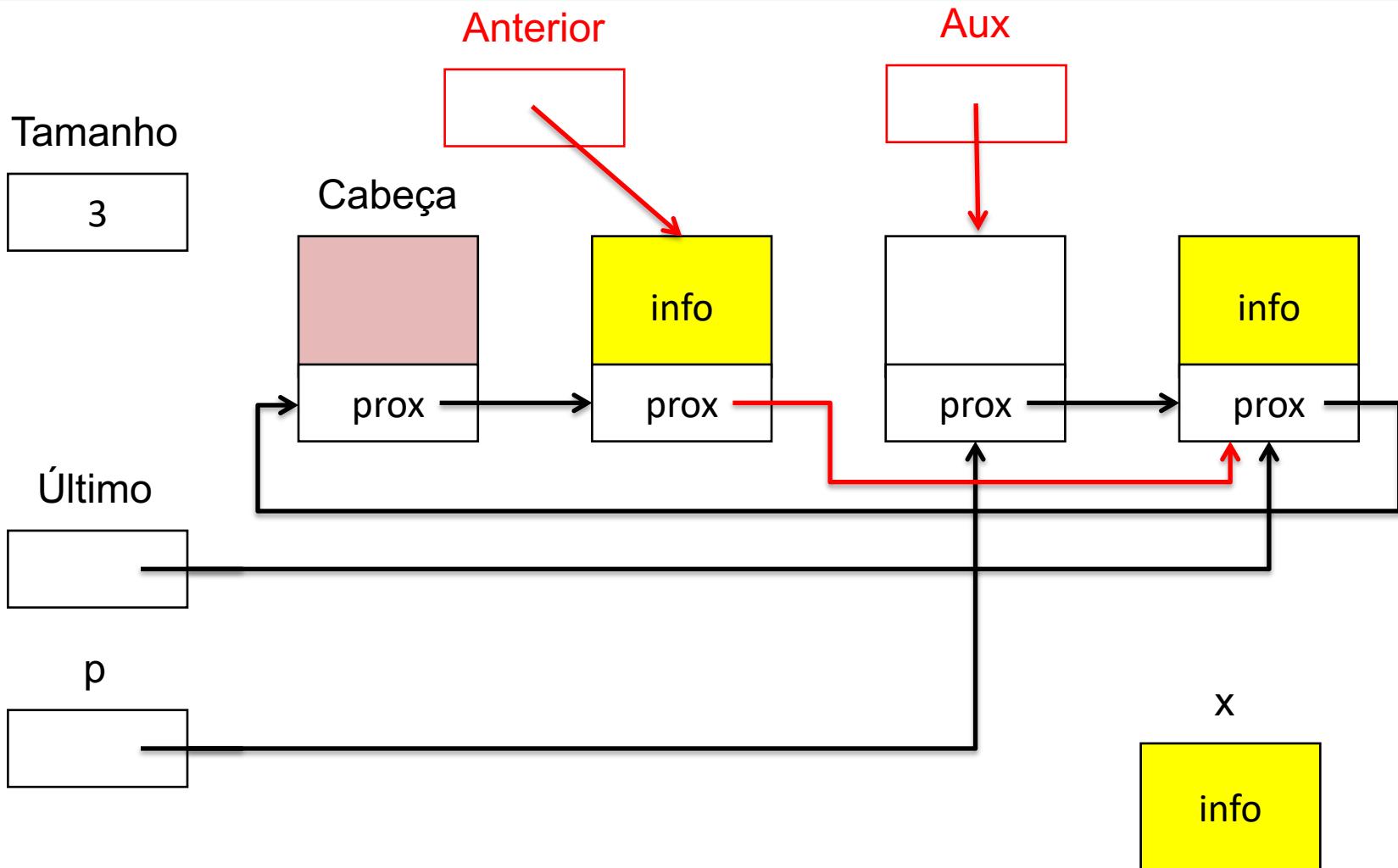
Retirada de Posição Qualquer



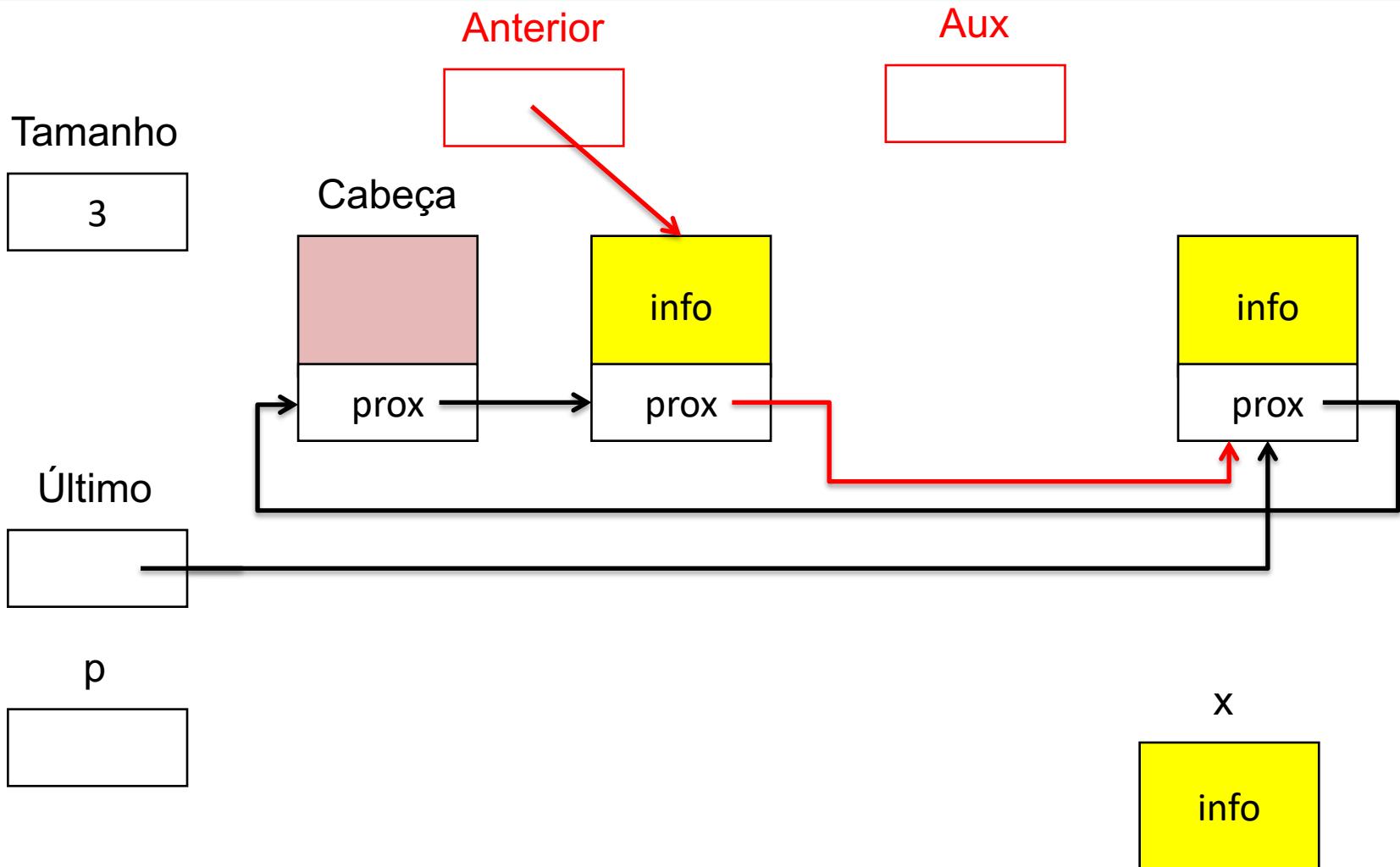
Retirada de Posição Qualquer



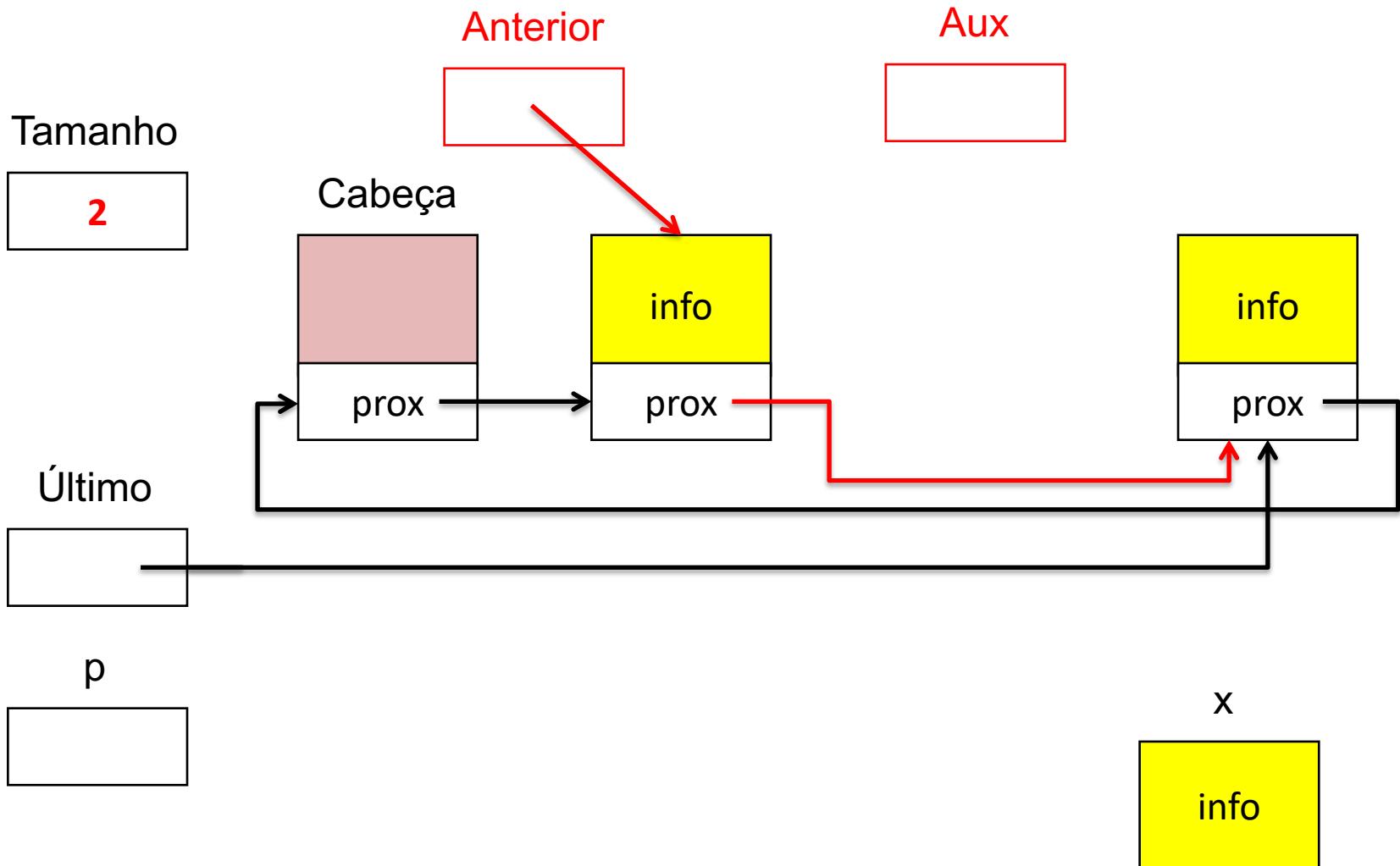
Retirada de Posição Qualquer



Retirada de Posição Qualquer



Retirada de Posição Qualquer

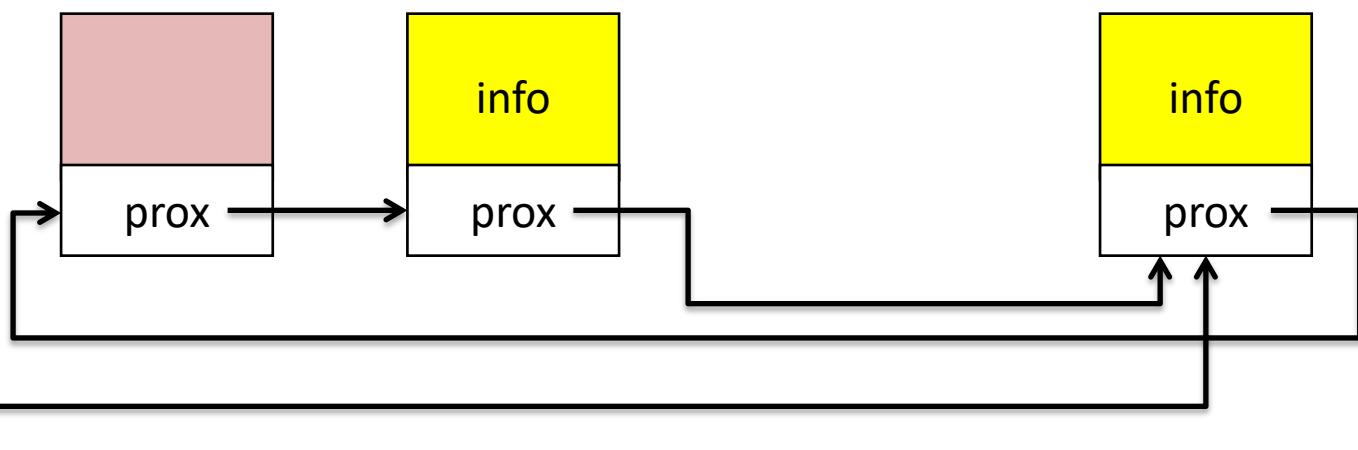


Retirada de Posição Qualquer

Tamanho

2

Cabeça



Último

p

x

info

Retirada de Posição Qualquer

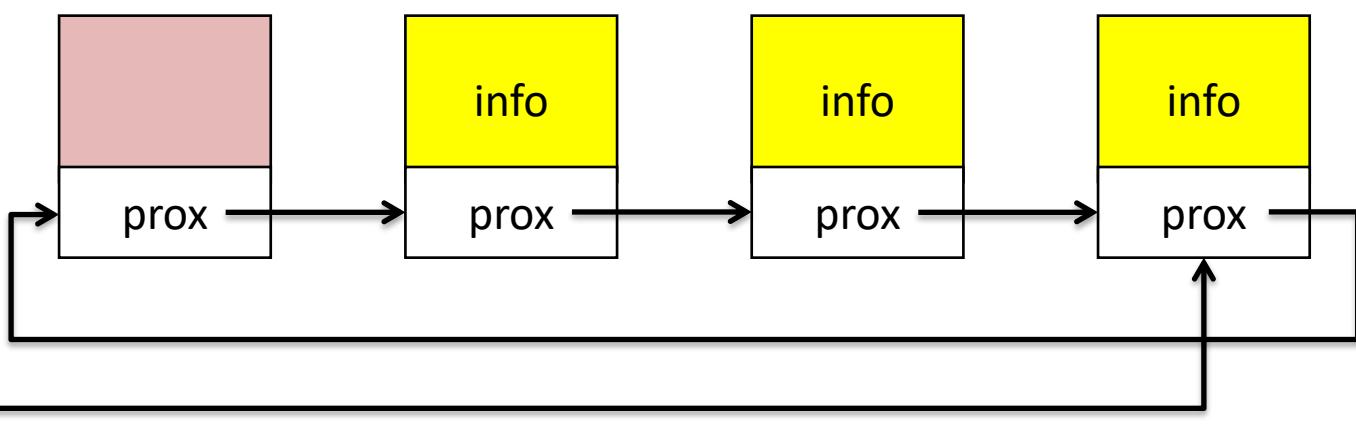
```
/* Retira o item da lista da posicao apontada por p */
int TLista_Retira(TLista *pLista, TAapontador p, TItem *pX)
{
    TAapontador pAnterior, pAux;
    if (TLista_EhVazia(pLista))
        return 0;
    pAnterior = pLista->Ultimo->Prox;
    while ((pAnterior != pLista->Ultimo) && (pAnterior->Prox != p))
        pAnterior = pAnterior->Prox;
    if (pAnterior->Prox != p)
        return 0;
    pAux = pAnterior->Prox;
    pAnterior->Prox = pAux->Prox;
    if (pAux == pLista->Ultimo)
        pLista->Ultimo = pAnterior;
    *pX = pAux->Item;
    free(pAux);
    pLista->Tamanho--;
    return 1;
}
```

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



p

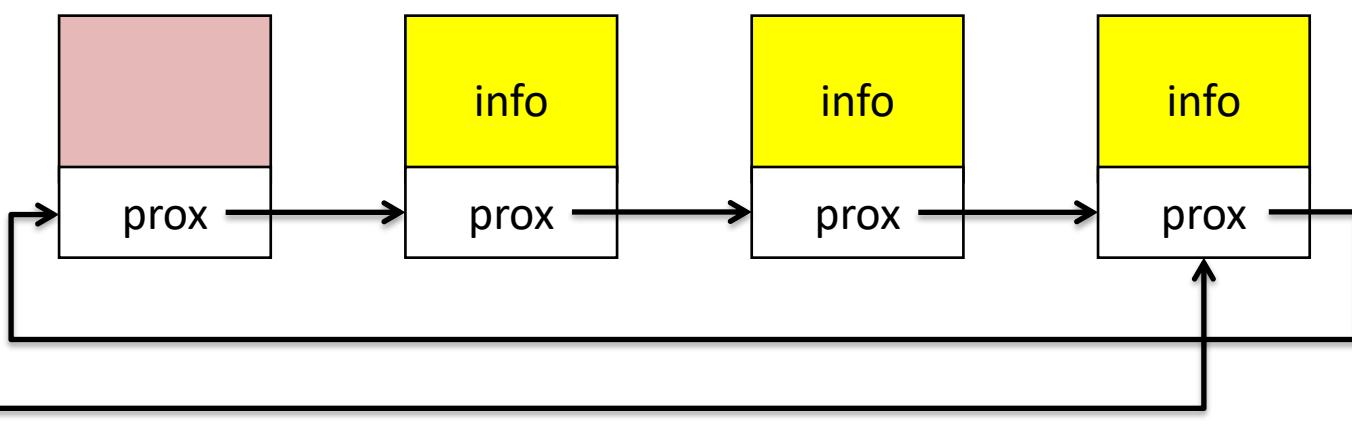
2

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



Último

[]

p

2

Apontador

[]

Posicao

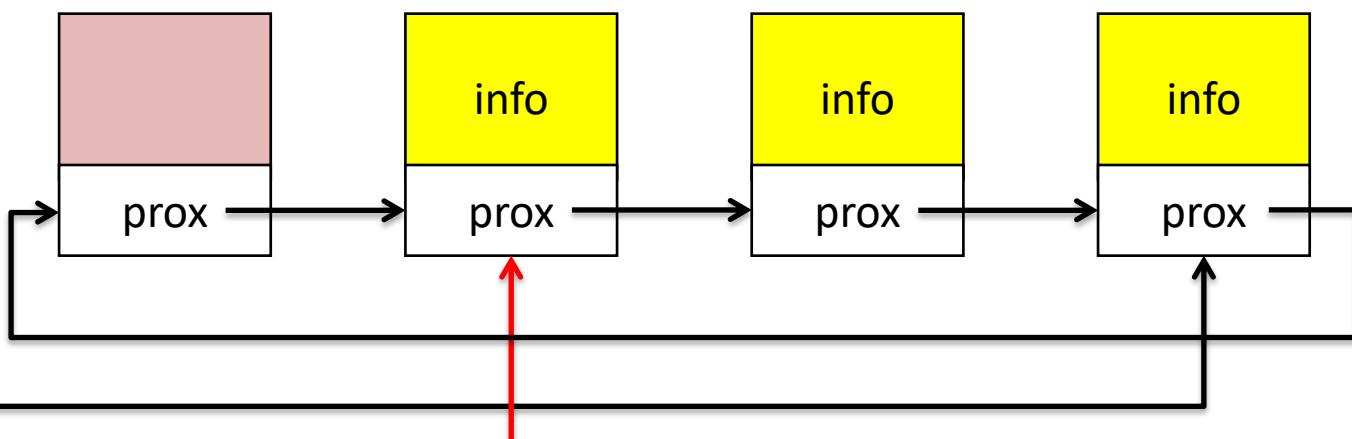
[]

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



p

2

Apontador



Posicao

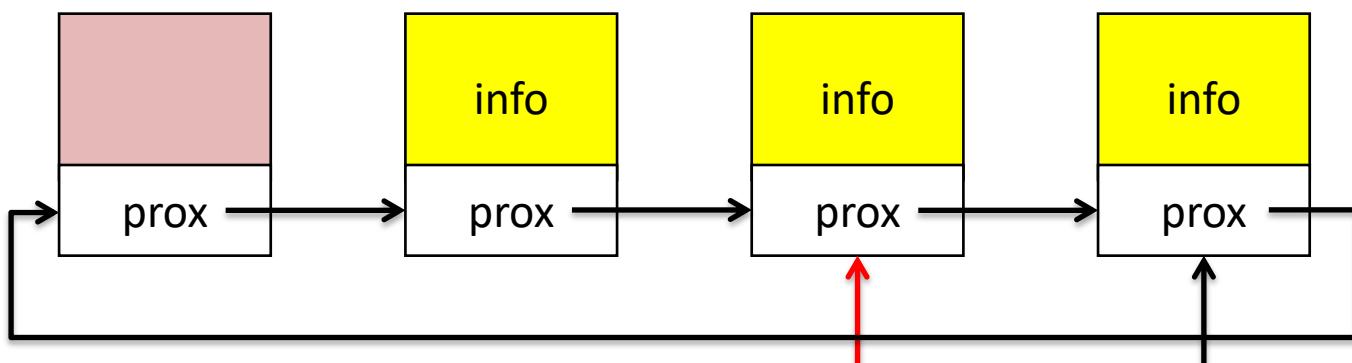
1

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



Último

[]

p

2

Apontador

[]

Posicao

2

Encontrar uma Posição na Lista

```
/* 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->Ultimo->Prox->Prox;
    while ((pApontador != pLista->Ultimo->Prox) && (Posicao != p))
    {
        pApontador = pApontador->Prox;
        Posicao++;
    }

    return pApontador;
}
```

■ Observação:

- As operações de inserção e retirada de um item do meio da lista requerem uma busca pela célula anterior
 - Percorrer a lista, procurando o i -ésimo item, tem custo $O(n)$

■ Solução:

- Adicionar um novo campo na estrutura da célula (**TCelula**) a fim de manter um apontador para a célula anterior (**Ant**)

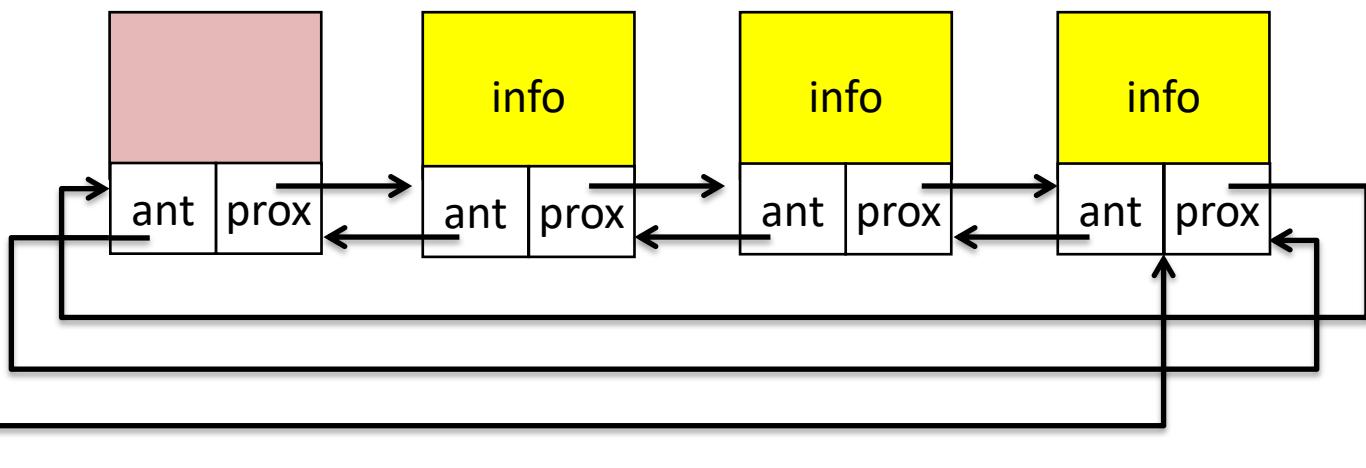
■ Listas Duplas

- Cada célula da lista mantém um apontador para a próxima célula e outro apontador para a célula anterior

Tamanho

3

Cabeça



Estrutura da Lista por Apontador

```
typedef int TChave;

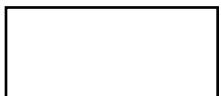
typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct SCelula *TApontador;

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

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

Cria Lista Vazia



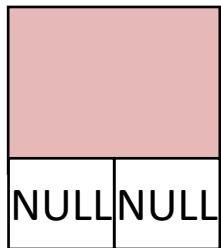
Último



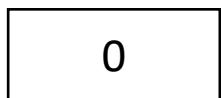
Tamanho

Cria Lista Vazia

Cabeça



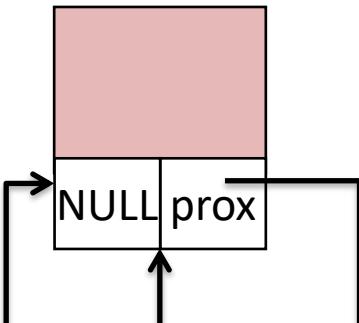
Último



Tamanho

Cria Lista Vazia

Cabeça



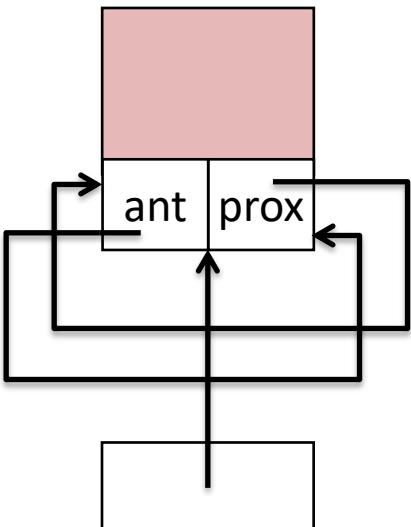
Último



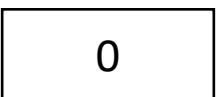
Tamanho

Cria Lista Vazia

Cabeça



Ultimo



Tamanho

Cria Lista Vazia

```
/* Inicia as variáveis da lista */
void TLista_Inicia(TLista *pLista)
{
    pLista->Ultimo = (TApontador) malloc(sizeof(TCelula));
    pLista->Ultimo->Prox = pLista->Ultimo;
    pLista->Ultimo->Ant = pLista->Ultimo;
    pLista->Tamanho = 0;
}

/* Retorna se a lista está vazia */
int TLista_EhVazia(TLista *pLista)
{
    return (pLista->Ultimo->Prox == pLista->Ultimo);
}

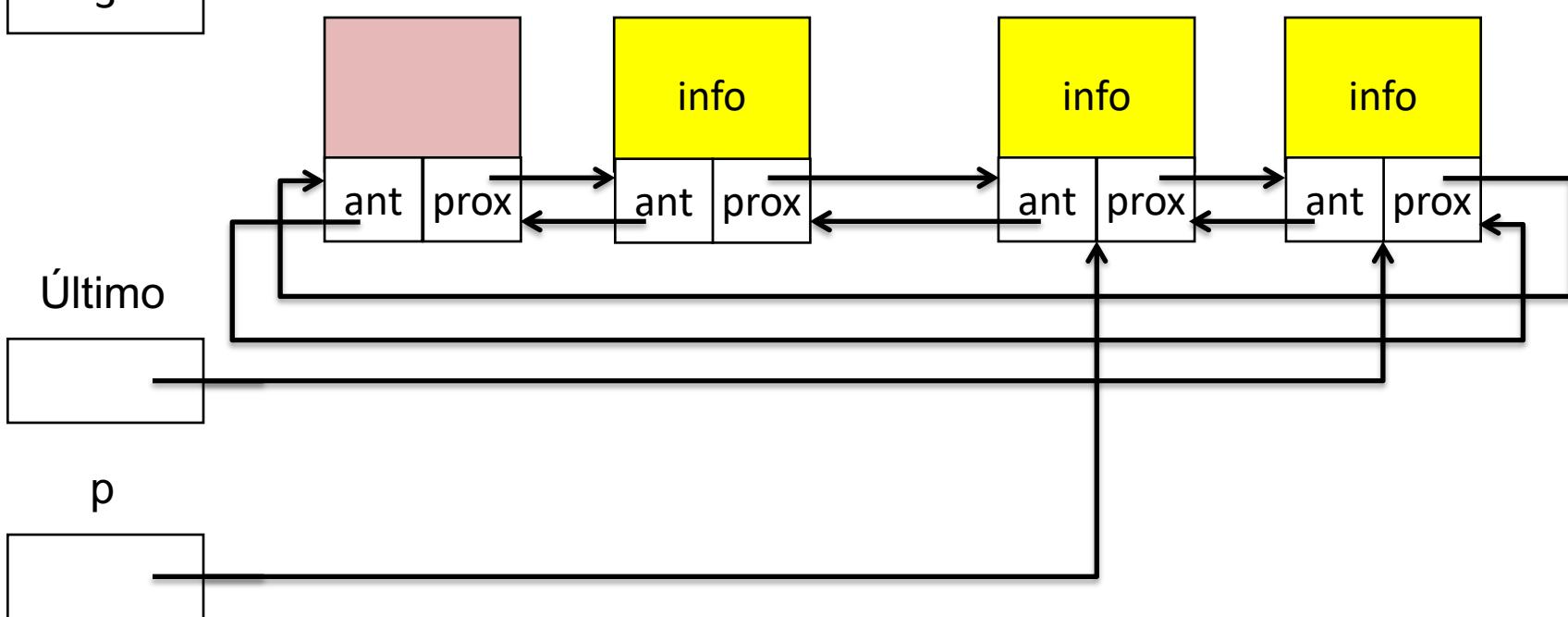
/* Retorna o tamanho da lista */
int TLista_Tamanho(TLista *pLista)
{
    return (pLista->Tamanho);
}
```

Inserção em Posição Qualquer

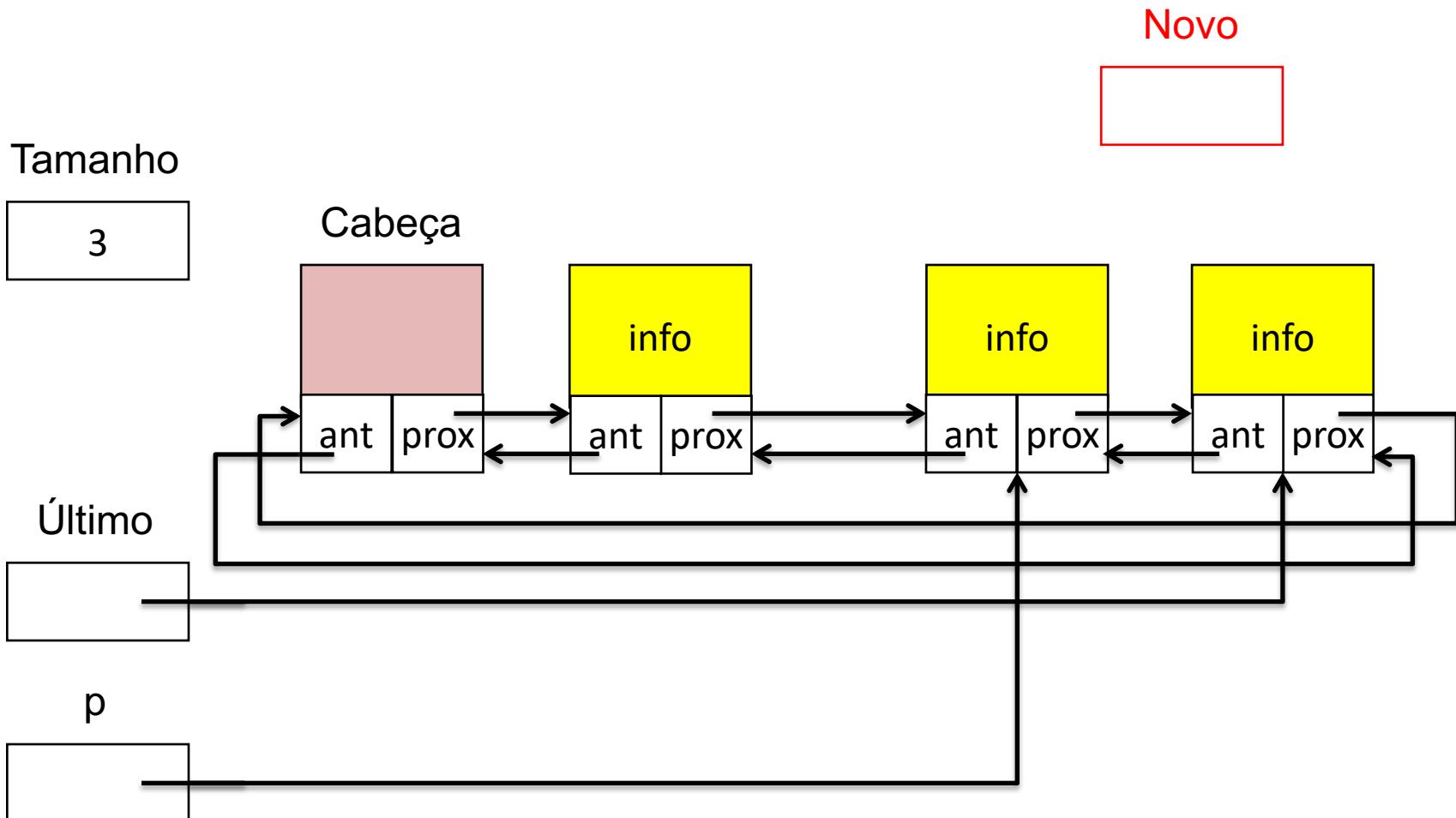
Tamanho

3

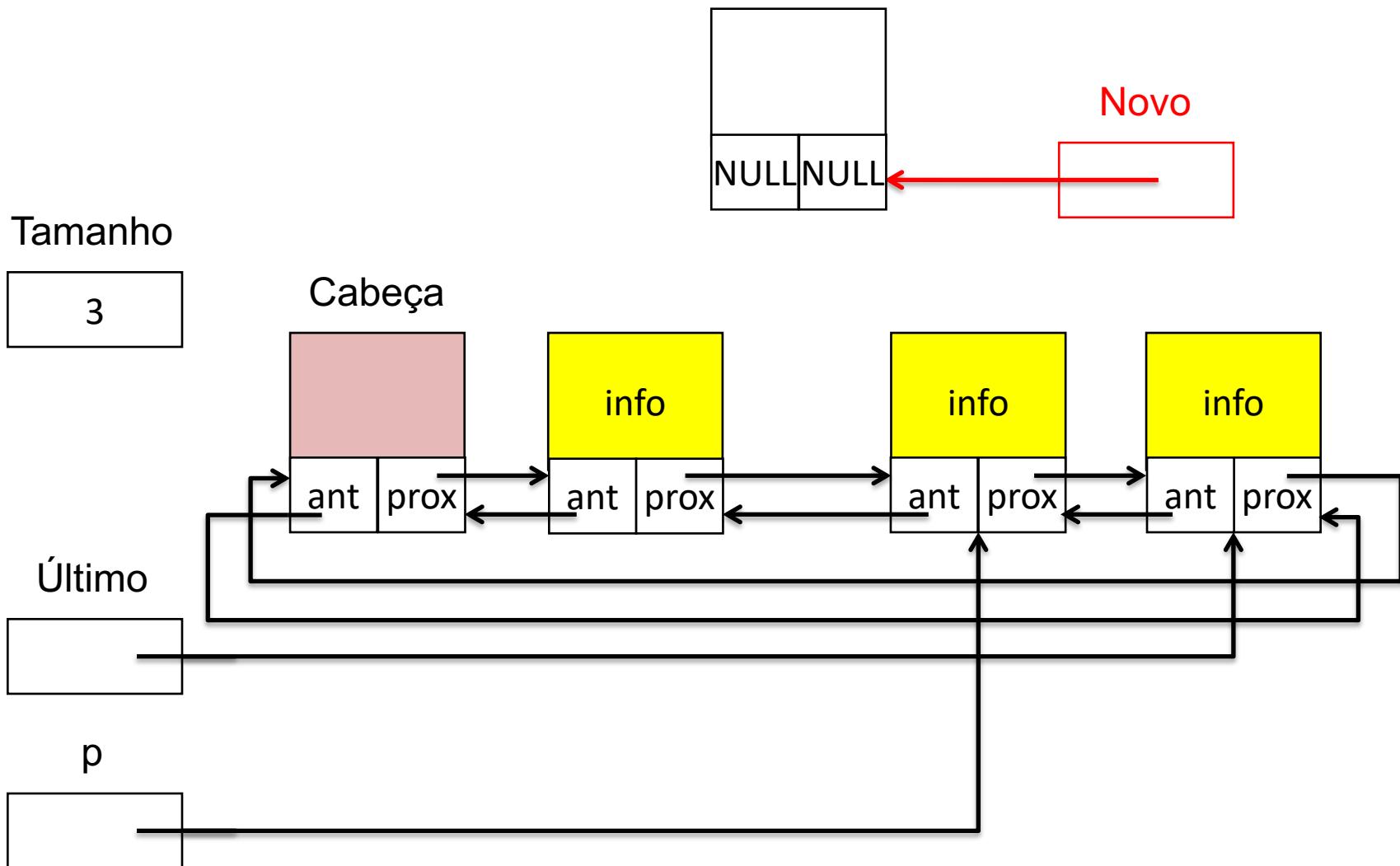
Cabeça



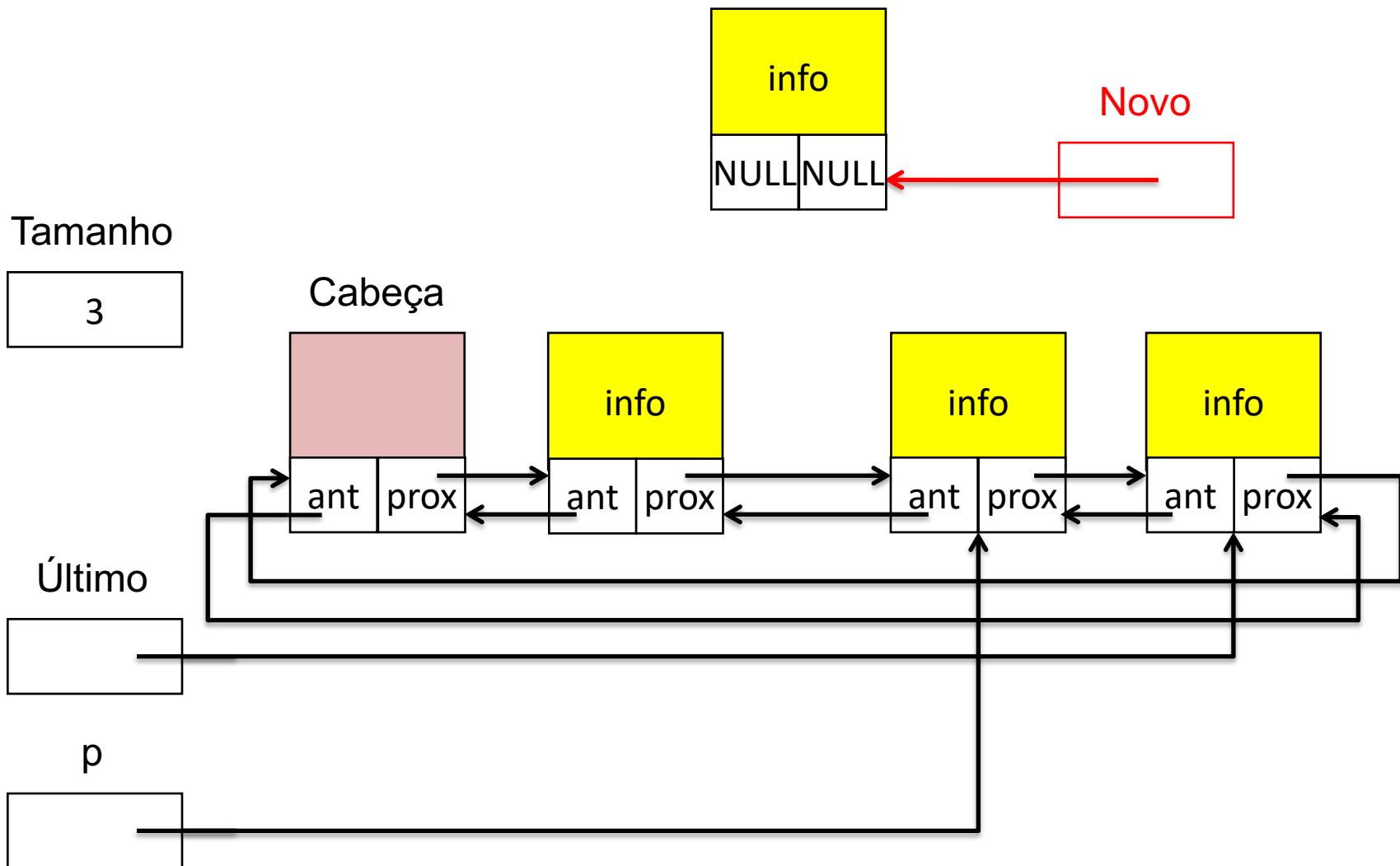
Inserção em Posição Qualquer



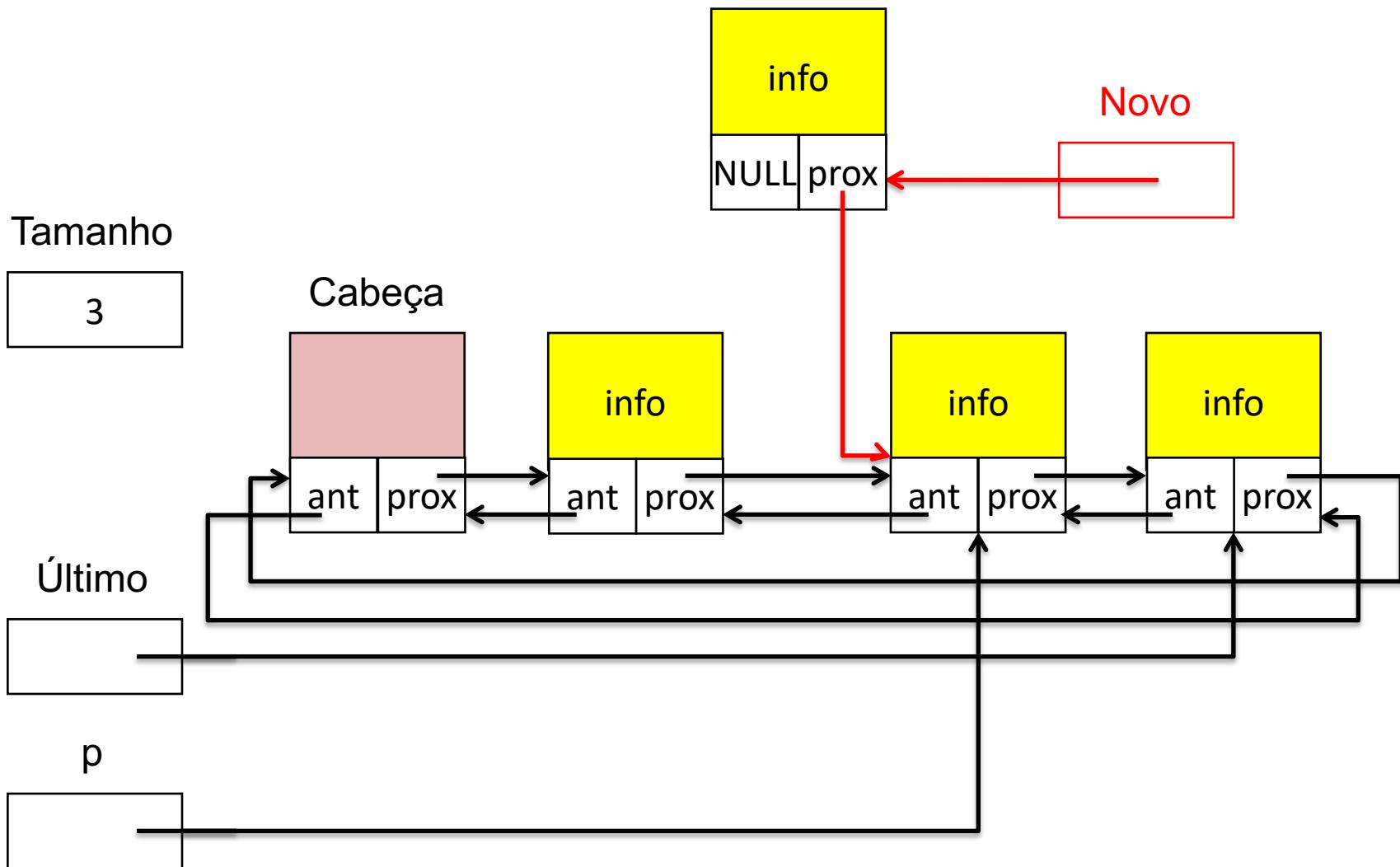
Inserção em Posição Qualquer



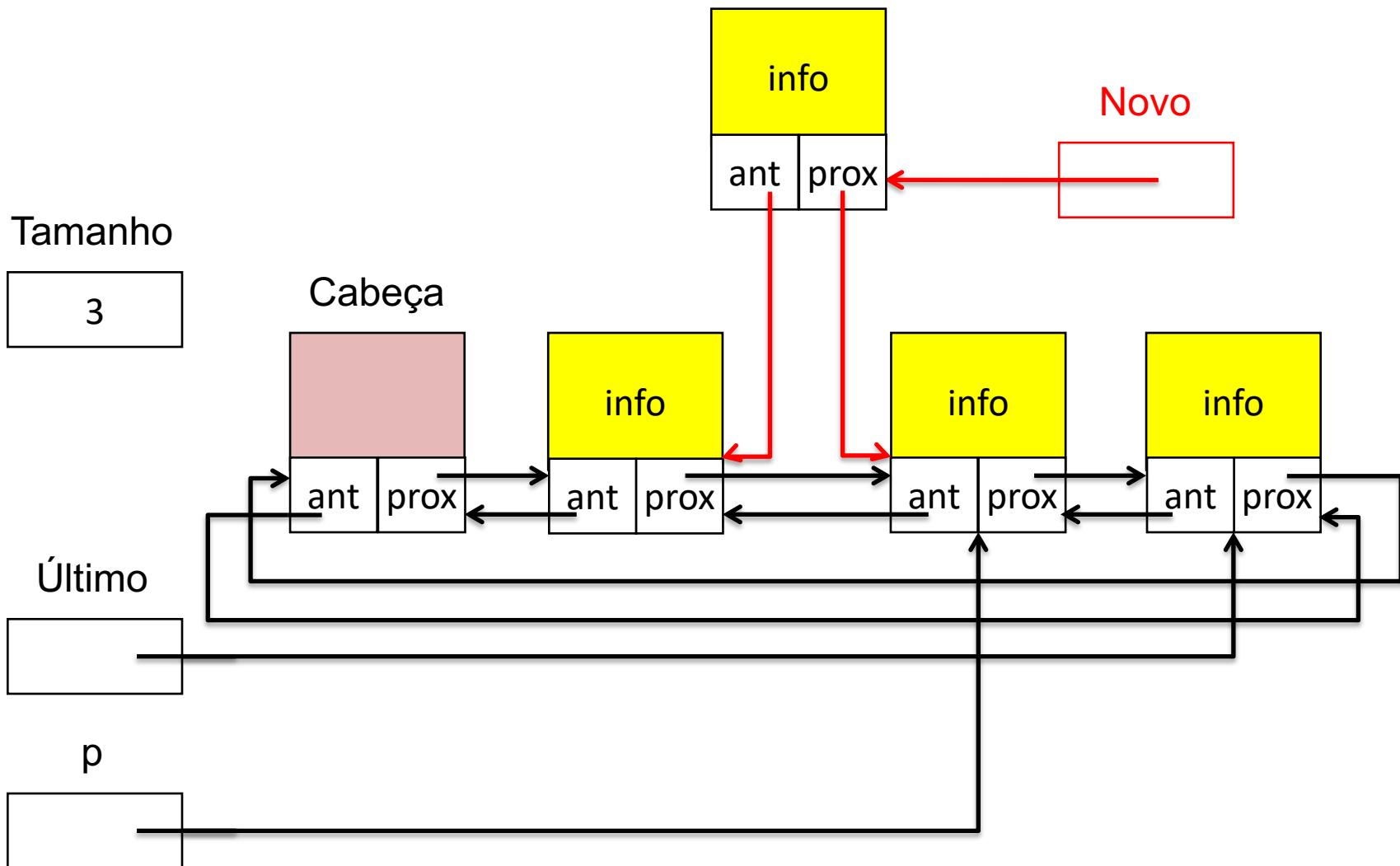
Inserção em Posição Qualquer



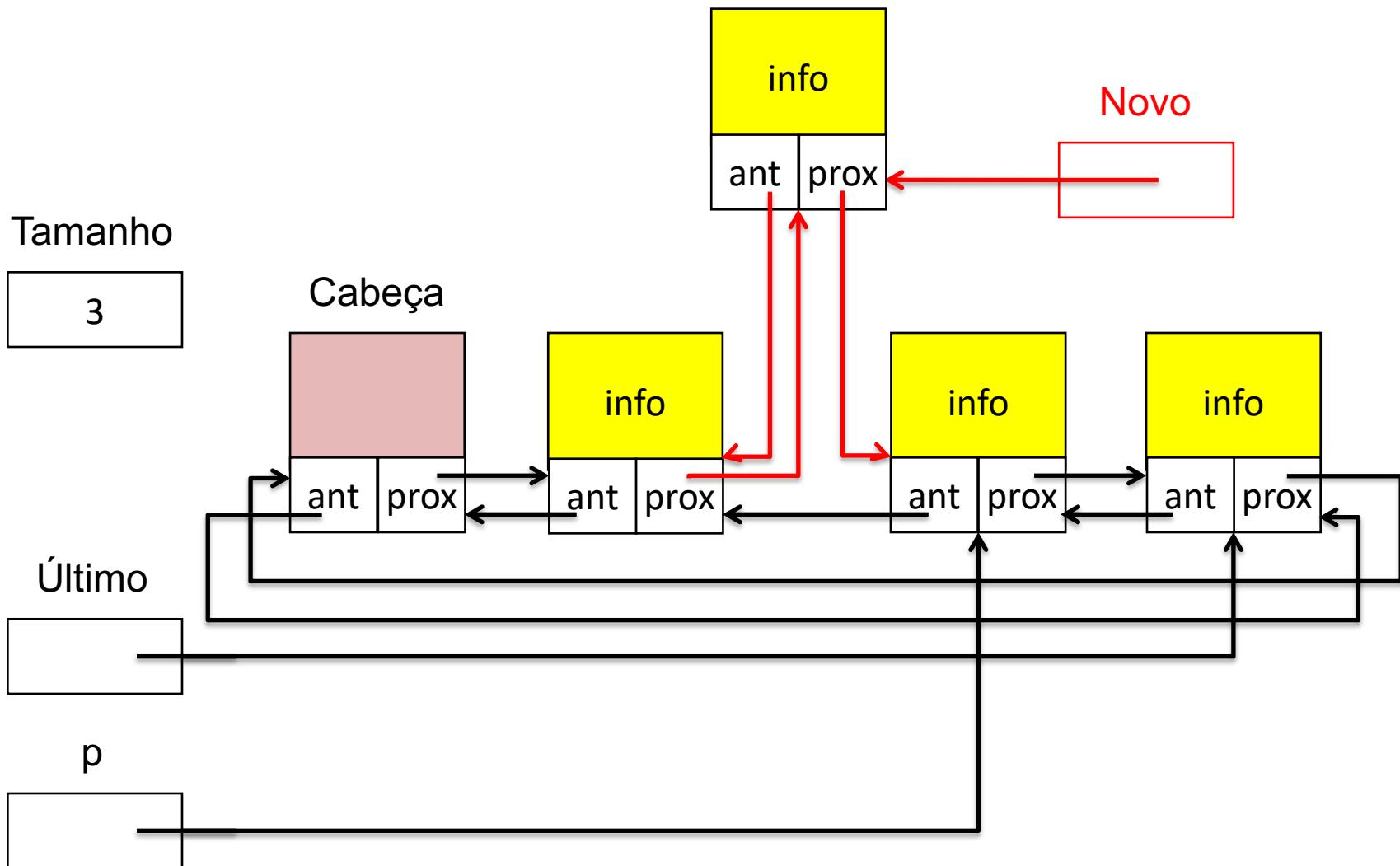
Inserção em Posição Qualquer



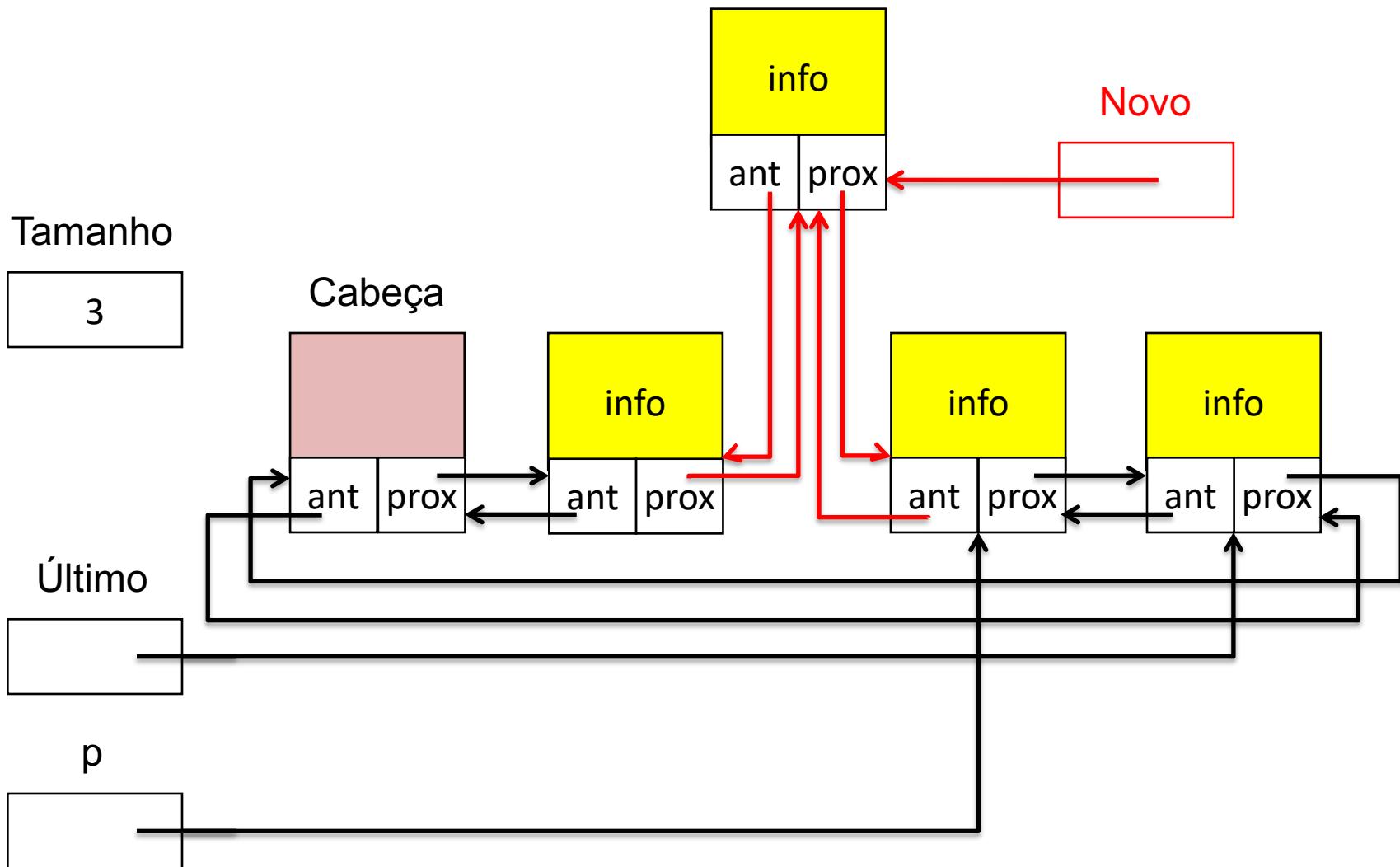
Inserção em Posição Qualquer



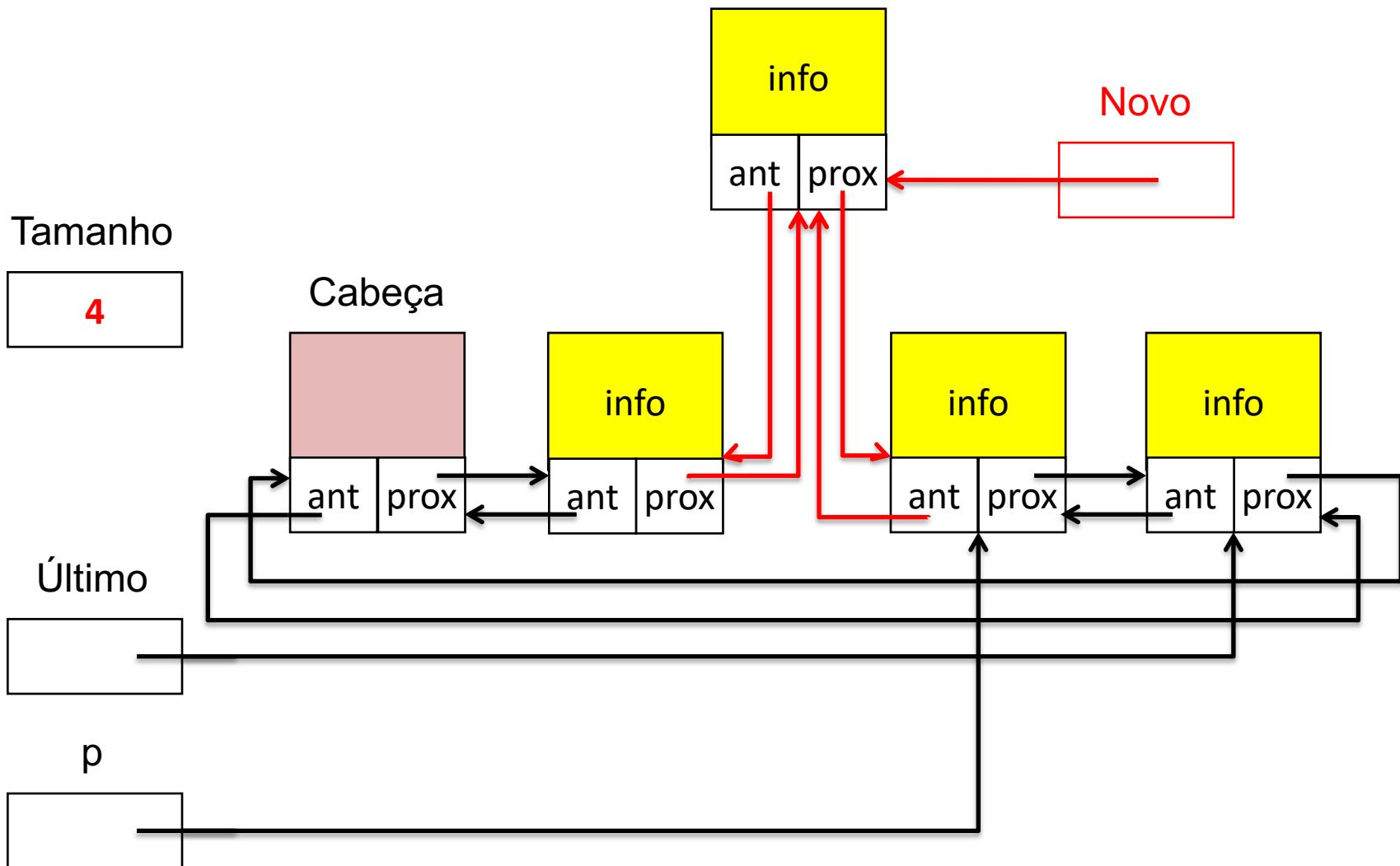
Inserção em Posição Qualquer



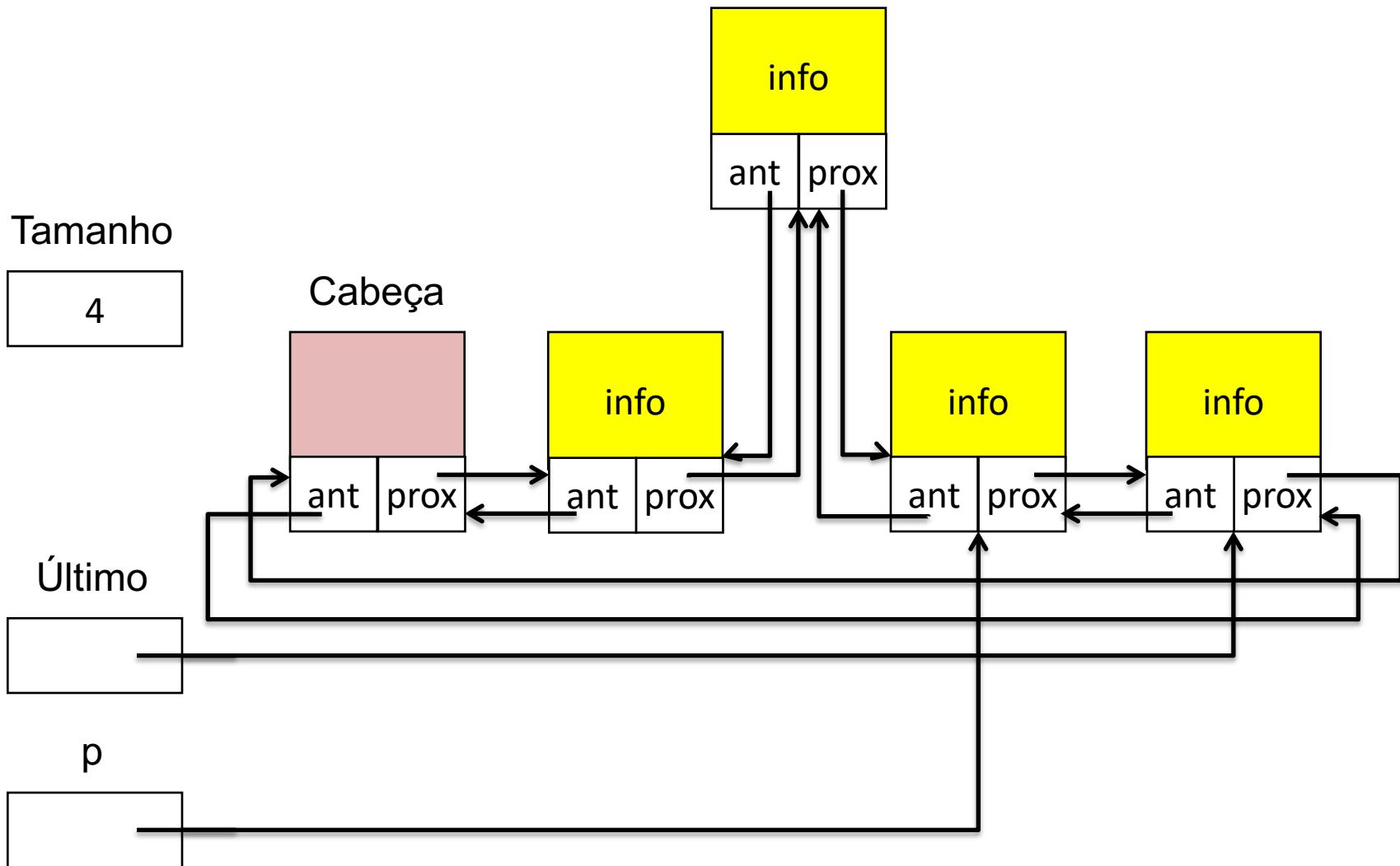
Inserção em Posição Qualquer



Inserção em Posição Qualquer



Inserção em Posição Qualquer



Inserção em Posição Qualquer

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

    if (p == NULL)
        return 0;

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

Retirada de Posição Qualquer

Tamanho

3

Cabeça

ant prox

info

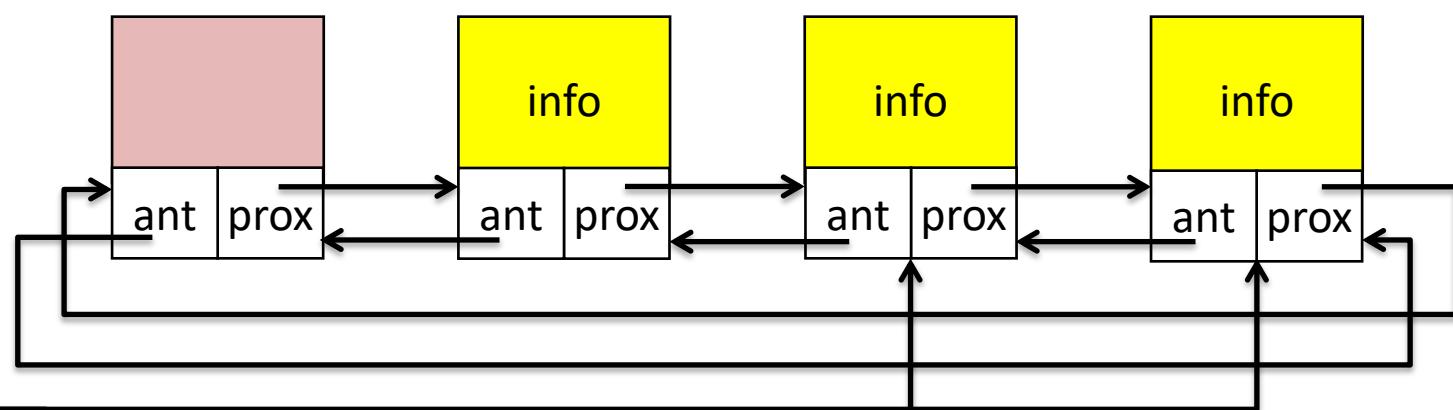
info

info

Último

p

x

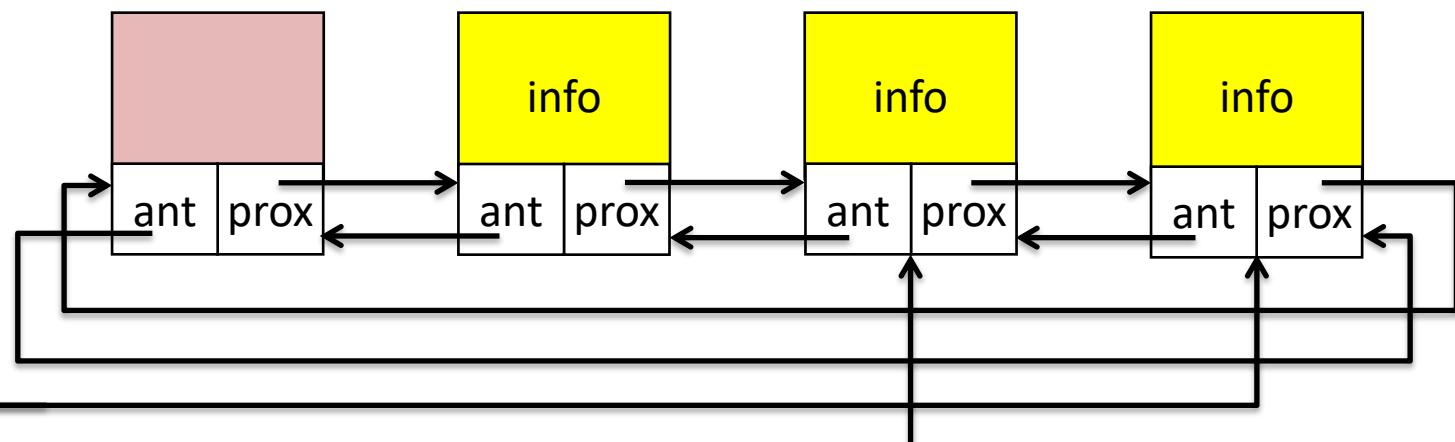


Retirada de Posição Qualquer

Tamanho

3

Cabeça



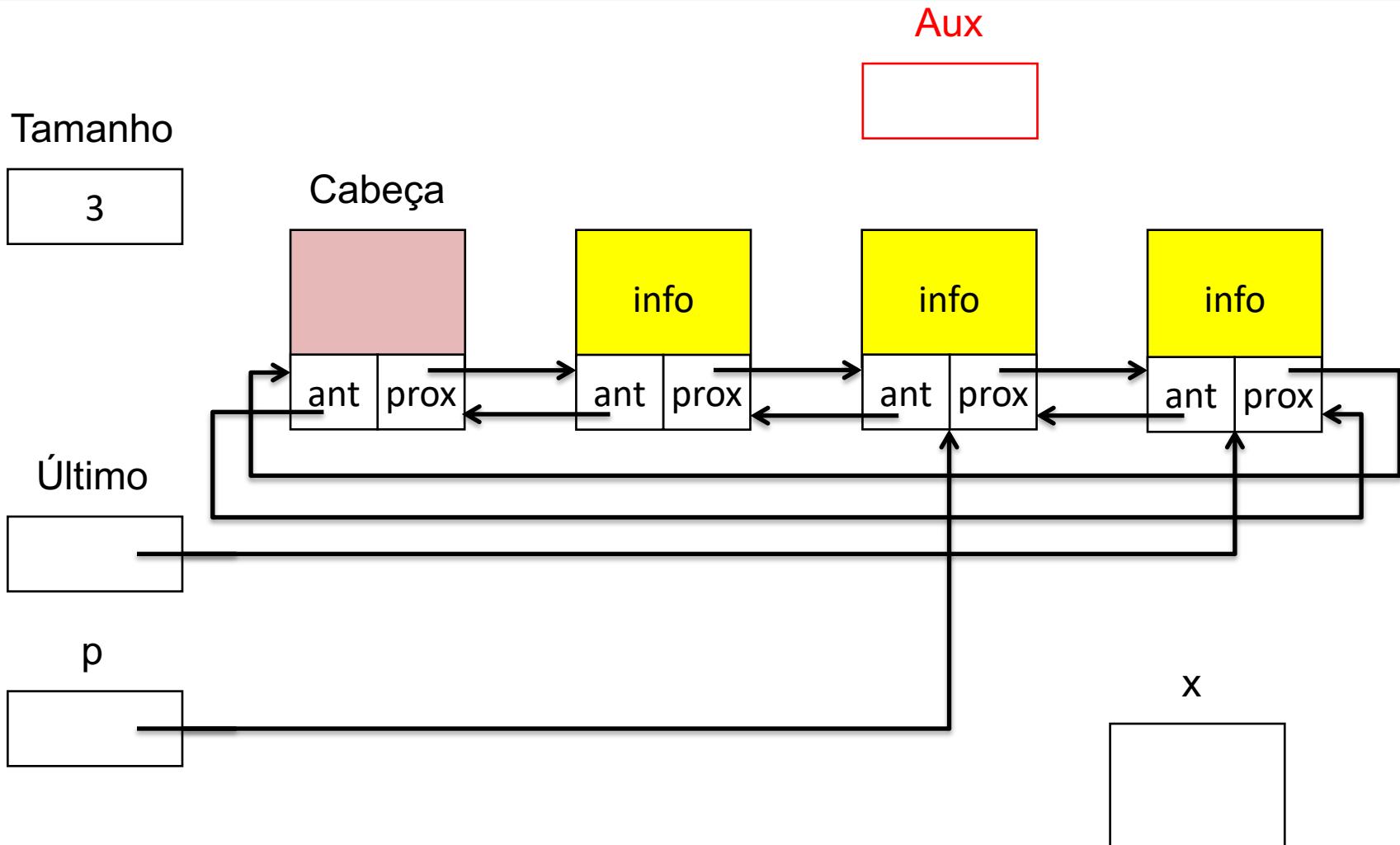
Último

p

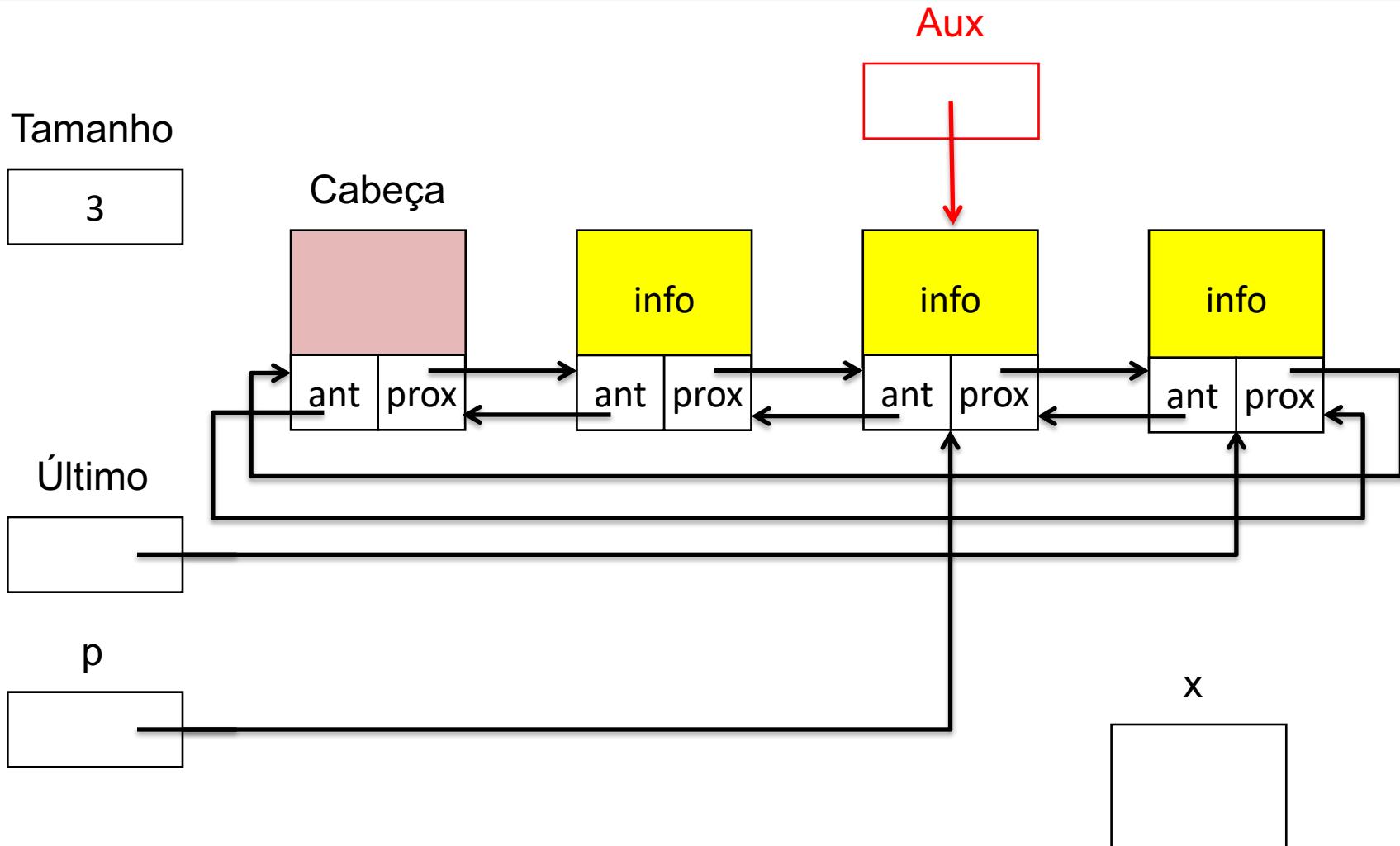
x

A lista está vazia?

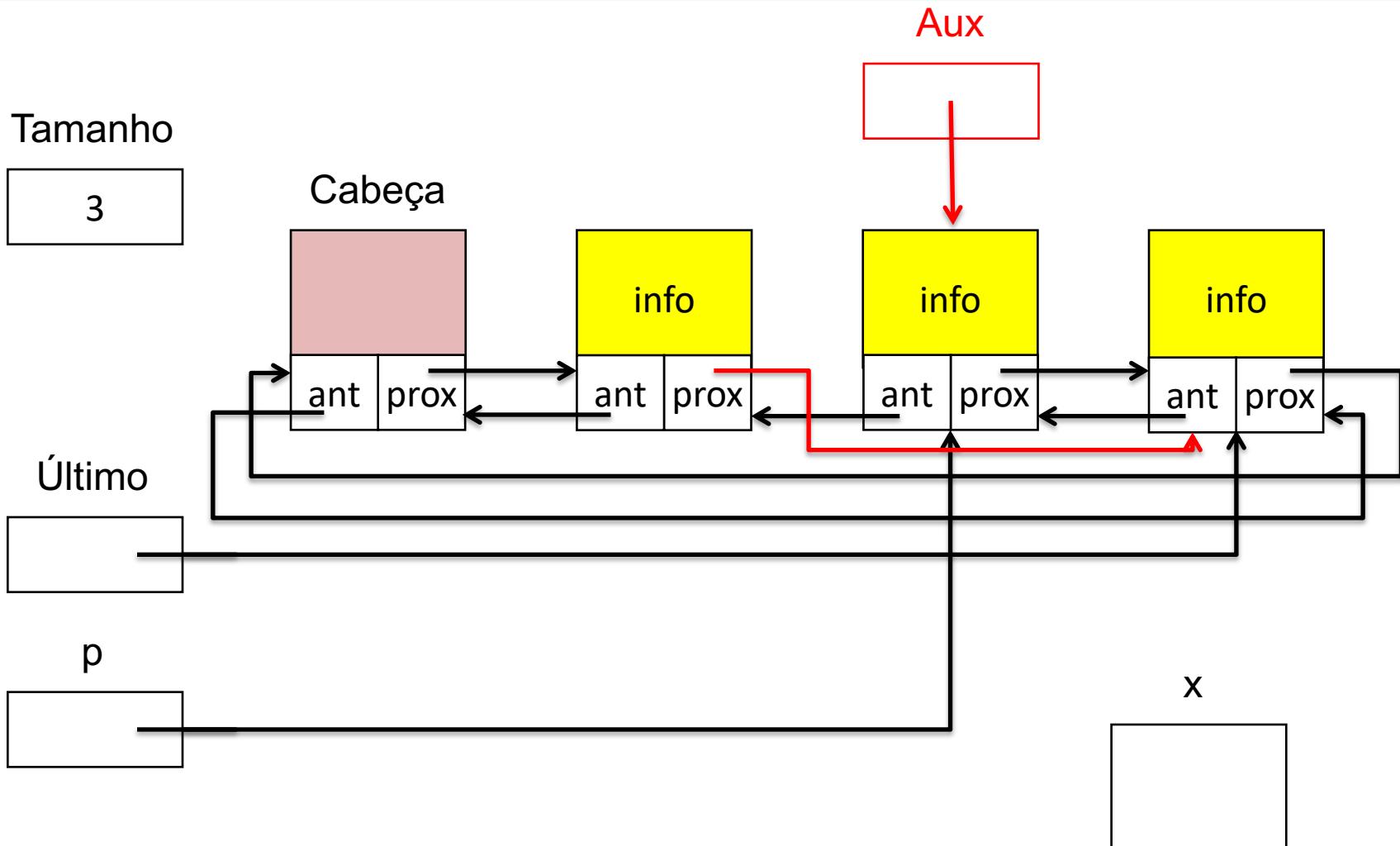
Retirada de Posição Qualquer



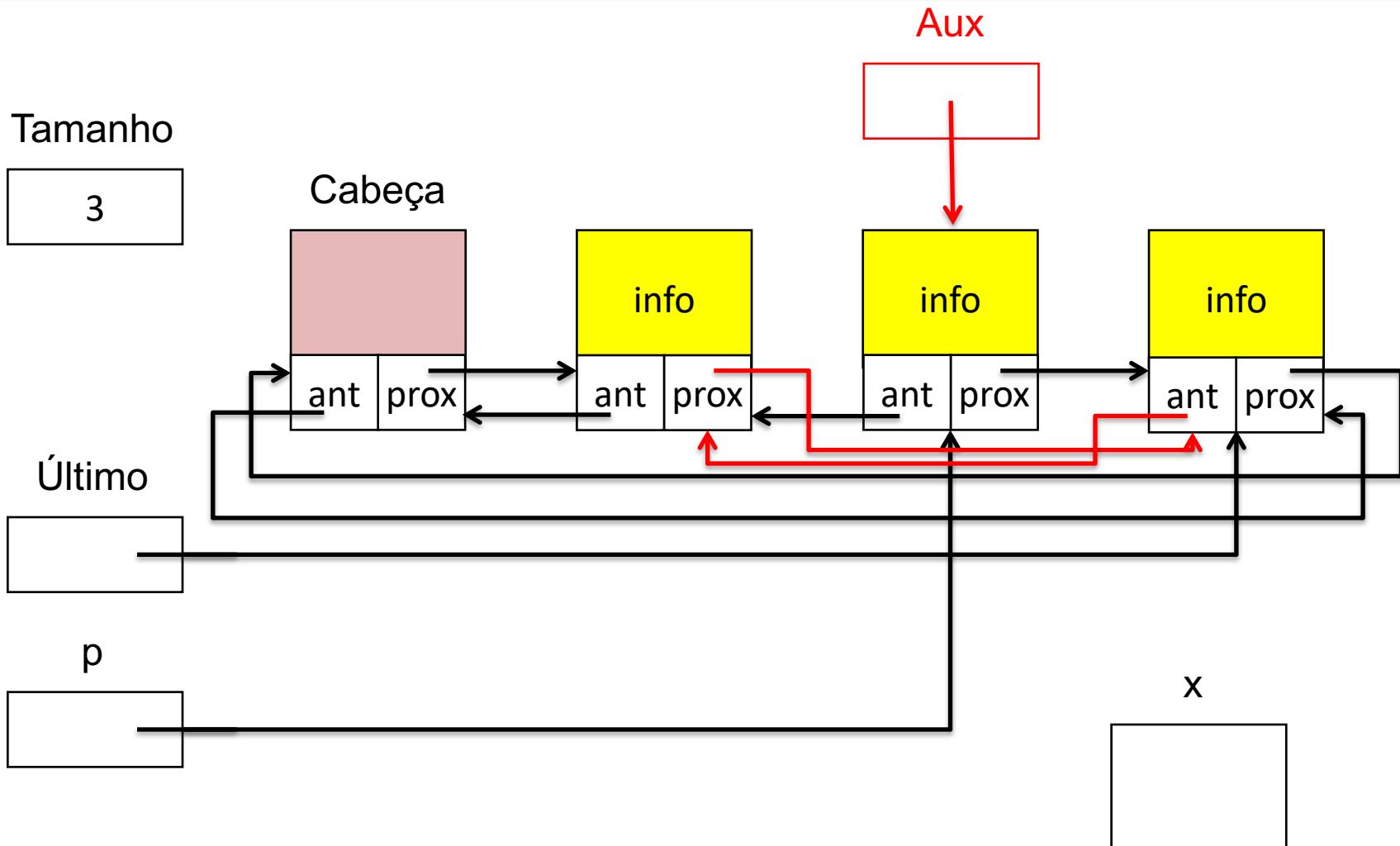
Retirada de Posição Qualquer



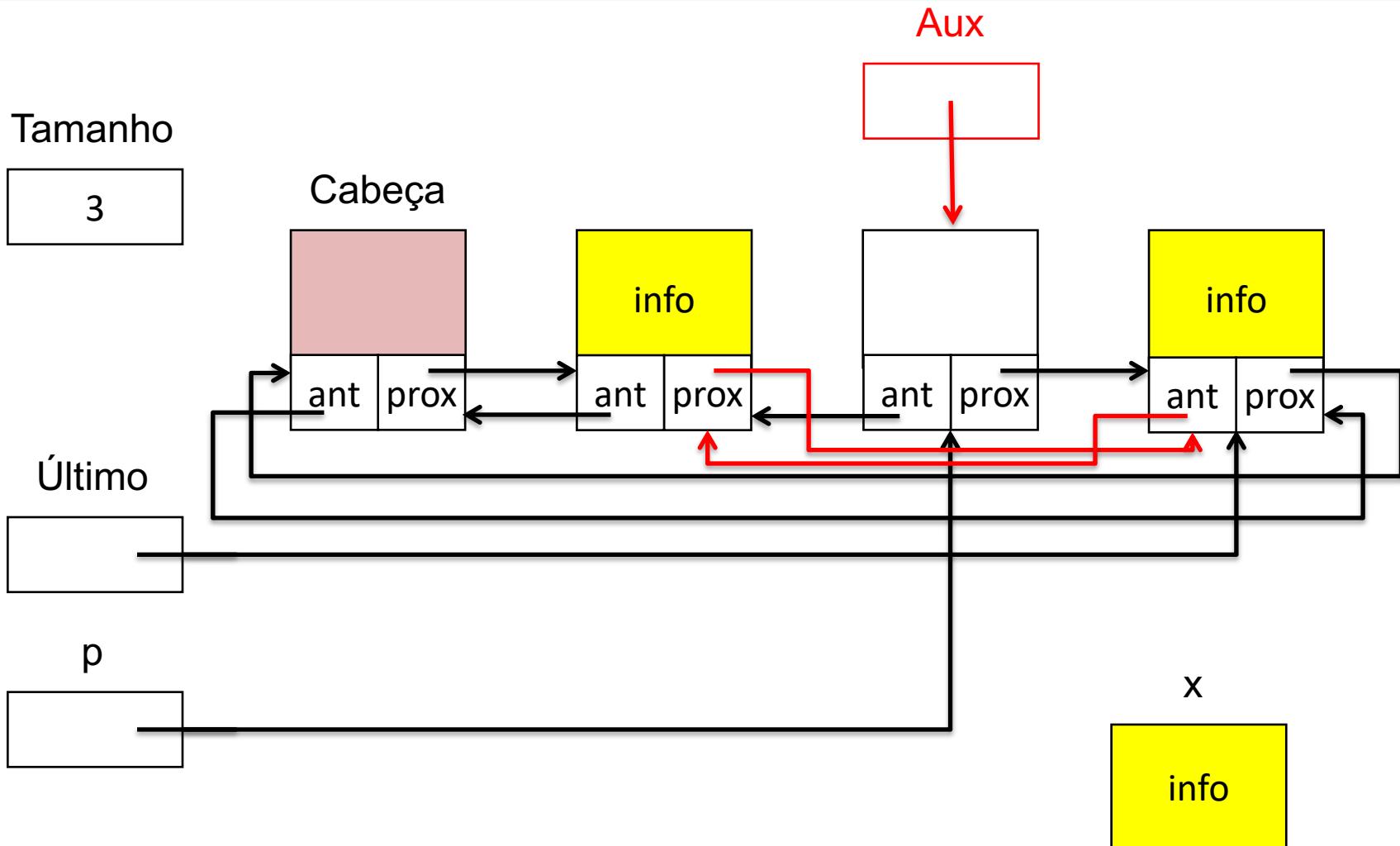
Retirada de Posição Qualquer



Retirada de Posição Qualquer



Retirada de Posição Qualquer



Retirada de Posição Qualquer

Tamanho

3

Cabeça

ant prox

info

Último

—

p

—

Aux

—

info

ant prox

x

info

Retirada de Posição Qualquer

Tamanho

2

Cabeça

ant prox

info

Último

—

p

—

Aux

—

info

ant prox

x

info

Retirada de Posição Qualquer

Tamanho

2

Cabeça

Último

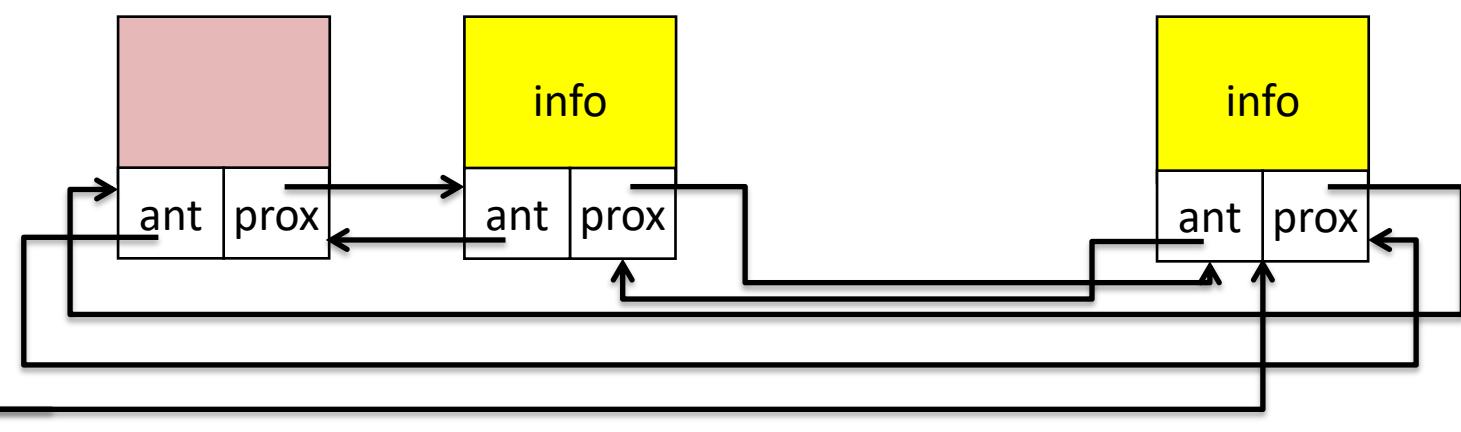
—

p

—

x

info



Retirada de Posição Qualquer

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

    if (TLista_EhVazia(pLista))
        return 0;

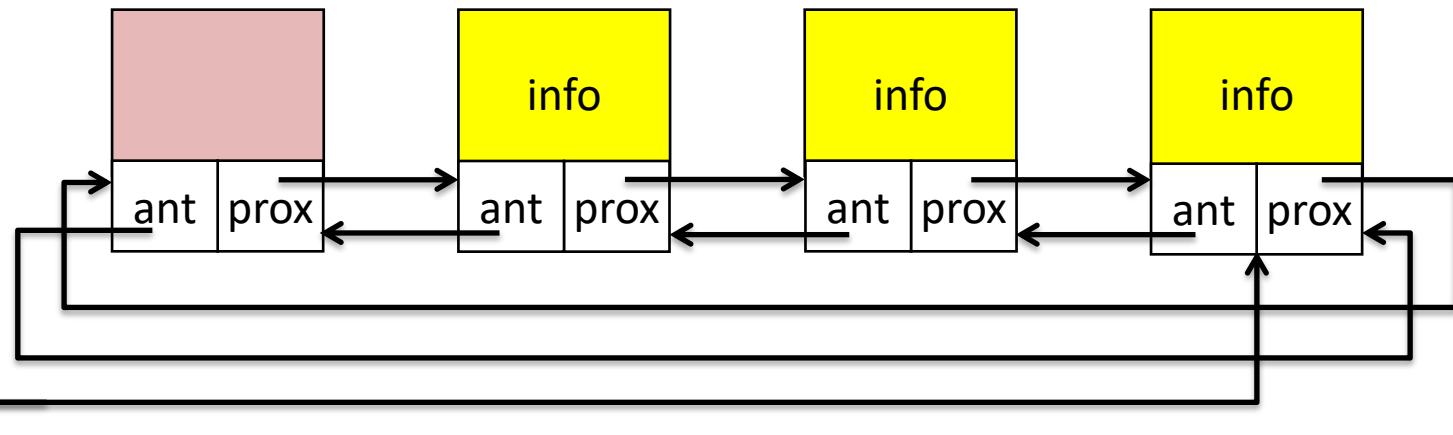
    pAux = p;
    p->Ant->Prox = p->Prox;
    p->Prox->Ant = p->Ant;
    if (pAux == pLista->Ultimo)
        pLista->Ultimo = p->Ant;
    *pX = pAux->Item;
    free(pAux);
    pLista->Tamanho--;
    return 1;
}
```

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



Último

p

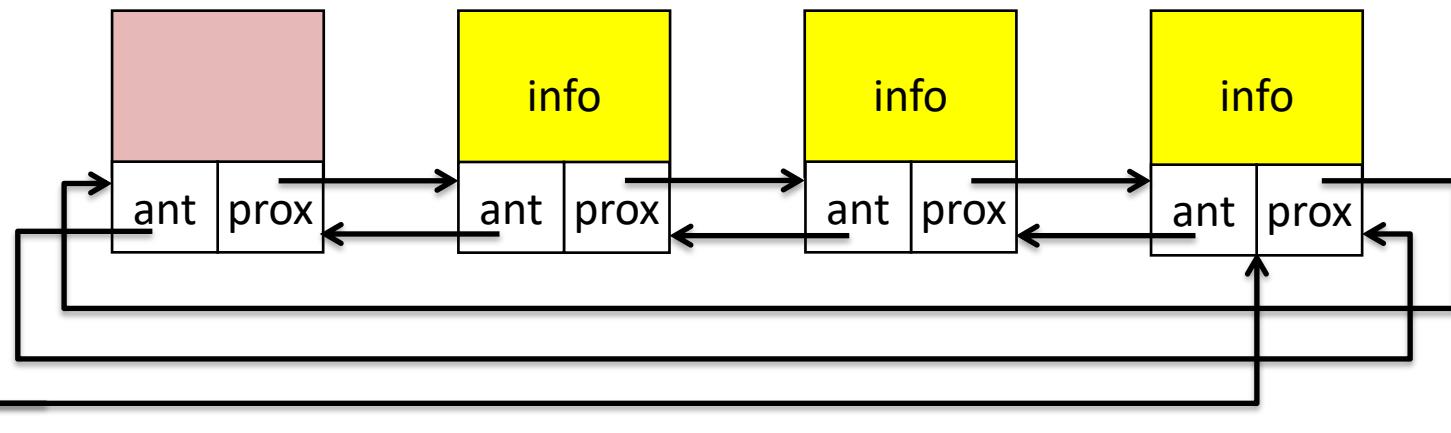
2

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



Último

p

2

Apontador

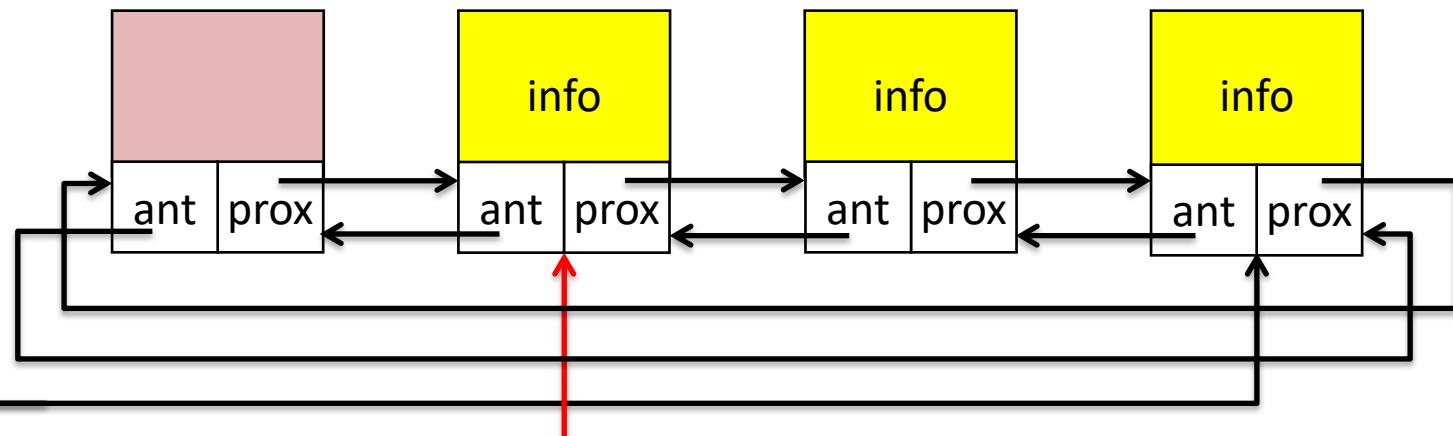
Posicao

Encontrar uma Posição na Lista

Tamanho

3

Cabeça



Último

p

2

Apontador



Posicao

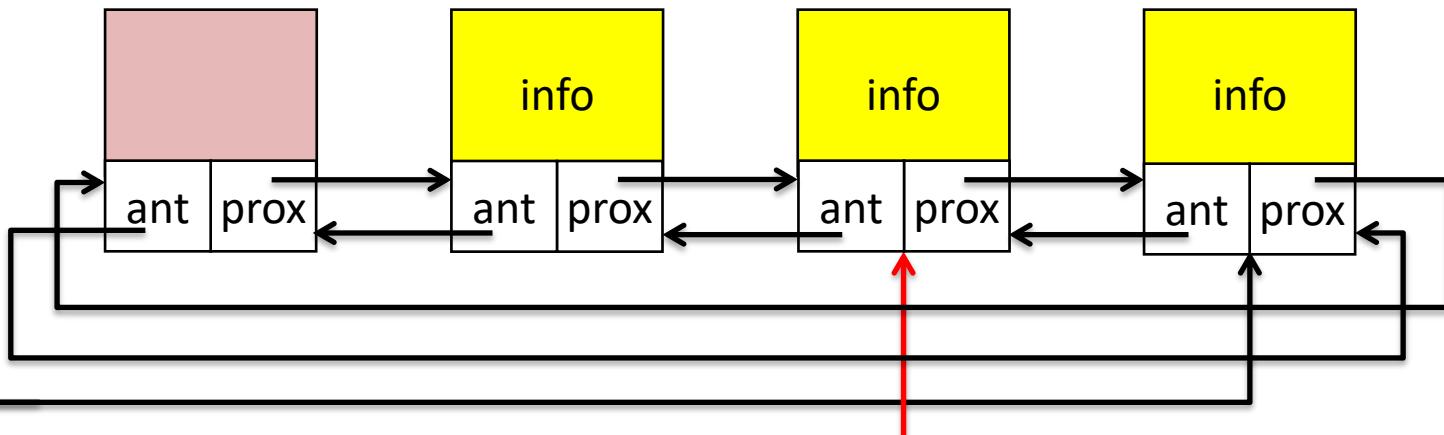
1

Encontrar uma Posição na Lista

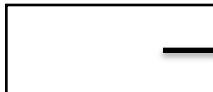
Tamanho

3

Cabeça



Último



p

2

Apontador



Posicao

2

Encontrar uma Posição na Lista

```
/* 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->Ultimo->Prox->Prox;
    while ((pApontador != pLista->Ultimo->Prox) && (Posicao != p))
    {
        pApontador = pApontador->Prox;
        Posicao++;
    }

    return pApontador;
}
```

EXERCÍCIO

```
/* Retorna um apontador para o p-esimo item da lista */
```

EXERCÍCIO: ALTERAR A FUNÇÃO PARA FAZER A BUSCA COMEÇANDO PELO ÚLTIMO QUANDO A POSIÇÃO P ESTIVER MAIS PRÓXIMA DO FIM DO QUE DO INÍCIO DA LISTA.

```
TApontador TLista_Retorna(TLista *pLista, int p)
{
    TApontador pApontador;
    int Posicao;

    Posicao = 0;
    pApontador = pLista->Ultimo->Prox->Prox;
    while ((pApontador != pLista->Ultimo->Prox) && (Posicao != p)) {
        pApontador = pApontador->Prox;
        Posicao++;
    }
    return pApontador;
}
```

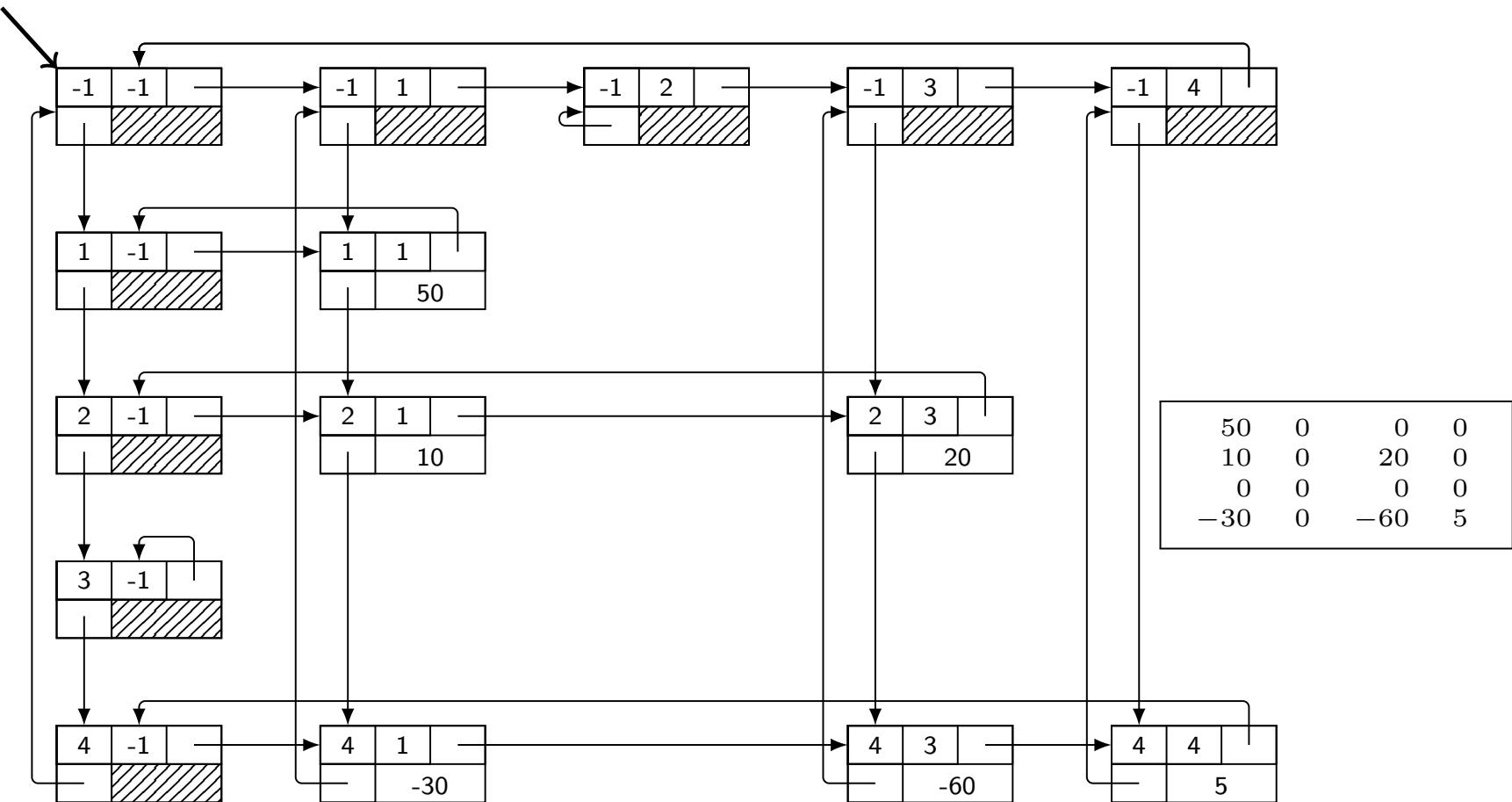
Exemplos de Aplicação

- Manipulação de matrizes esparsas:

50	0	0	0
10	0	20	0
0	0	0	0
-30	0	-60	5

- A solução a seguir usa **listas ortogonais** com célula cabeça, a fim de utilizar a técnica de sentinelas
- Considere que:
 - as linhas e as colunas são numeradas a partir de 1
 - o nó cabeça tem valor de linha e/ou coluna igual a -1

Exemplos de Aplicação



Exemplos de Aplicação

```
typedef float TItem;

typedef struct SCelula *TApontador;

typedef struct SCelula {
    int Linha, Coluna;
    TItem Item;
    TApontador Abaixo, Direita;
} TCelula;

typedef struct {
    TApontador Cabeca;
    int NLinhas, NColunas;
} TMatriz;

int TMatriz_NLinhas(TMatriz *pMatriz)
{
    return (pMatriz->NLinhas);
}

int TMatriz_NColunas(TMatriz *pMatriz)
{
    return (pMatriz->NColunas);
}
```

Exemplos de Aplicação

```
/* Insere uma celula cabeca no inicio da matriz*/
void TMatriz_InsereCabecaInicio(TMatriz *pMatriz, int Linha, int Coluna)
{
    TApontador pNovo;

    pNovo = (TApontador) malloc(sizeof(TCelula));
    pNovo->Linha = Linha;
    pNovo->Coluna = Coluna;
    if (Linha != -1) {
        pNovo->Abaixo = pMatriz->Cabeca->Abaixo;
        pNovo->Direita = pNovo;
        pMatriz->Cabeca->Abaixo = pNovo;
    }
    else if (Coluna != -1) {
        pNovo->Abaixo = pNovo;
        pNovo->Direita = pMatriz->Cabeca->Direita;
        pMatriz->Cabeca->Direita = pNovo;
    }
    else {
        pNovo->Abaixo = pNovo;
        pNovo->Direita = pNovo;
        pMatriz->Cabeca = pNovo;
    }
}
```

Exemplos de Aplicação

```
/* Retorna um apontador para a celula na posicao (Linha, Coluna) da matriz */
TApontador TMatriz_Retorna(TMatriz *pMatriz, int Linha, int Coluna)
{
    TApontador pCabeca, pCelula;

    pCabeca = pMatriz->Cabeca;
    while ((pCabeca->Abaixo != pMatriz->Cabeca) && (pCabeca->Linha < Linha))
        pCabeca = pCabeca->Abaixo;
    if (pCabeca->Linha == Linha) {
        pCelula = pCabeca;
        while ((pCelula->Direita != pCabeca) && (pCelula->Coluna < Coluna))
            pCelula = pCelula->Direita;
        if (pCelula->Coluna == Coluna)
            return pCelula;
        else
            return NULL;
    }
    else
        return NULL;
}
```

Exemplos de Aplicação

```
/* Inicia as variaveis da matriz */
void TMatriz_Inicia(TMatriz *pMatriz, int NLinhas, int NColunas)
{
    int Linha, Coluna;

    pMatriz->NLinhas = NLinhas;
    pMatriz->NColunas = NColunas;

    TMatriz_InsereCabecaInicio(pMatriz, -1, -1);
    for (Linha = NLinhas; Linha > 0; Linha--)
        TMatriz_InsereCabecaInicio(pMatriz, Linha, -1);
    for (Coluna = NColunas; Coluna > 0; Coluna--)
        TMatriz_InsereCabecaInicio(pMatriz, -1, Coluna);
}

/* Le o valor da posicao (Linha, Coluna) da matriz */
TItem TMatriz_Le(TMatriz *pMatriz, int Linha, int Coluna)
{
    TA pontador pCelula;

    pCelula = TMatriz_Retorna(pMatriz, Linha, Coluna);
    if (! pCelula)
        return 0.0;
    else
        return pCelula->Item;
}
```

Exemplos de Aplicação

```
int TMatriz_Insere(TMatriz *pMatriz, TApontador pEsquerda, TApontador pAcima,
                     int Linha, int Coluna, TItem x)
{
    TApontador pNovo;
    pNovo = (TApontador) malloc(sizeof(TCelula));
    if (! pNovo)
        return 0;

    pNovo->Linha = Linha;
    pNovo->Coluna = Coluna;
    pNovo->Item = x;
    pNovo->Direita = pEsquerda->Direita;
    pEsquerda->Direita = pNovo;
    pNovo->Abaixo = pAcima->Abaixo;
    pAcima->Abaixo = pNovo;
    return 1;
}

int TMatriz_Retira(TMatriz *pMatriz, TApontador pEsquerda, TApontador pAcima,
                    TApontador pCelula)
{
    pEsquerda->Direita = pCelula->Direita;
    pAcima->Abaixo = pCelula->Abaixo;
    free(pCelula);
    return 1;
}
```

Exemplos de Aplicação

```
/* Escreve o valor da posicao (Linha, Coluna) da matriz */
int TMatriz_Escreve(TMatriz *pMatriz, int Linha, int Coluna, TItem x)
{
    TA pontador pCabecaLinha, pCabecaColuna, pEsquerda, pAcima, pCelula;

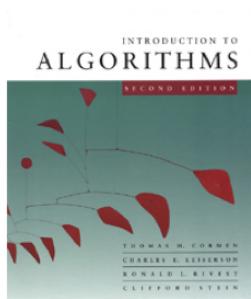
    pCabecaLinha = TMatriz_Retorna(pMatriz, Linha, -1);
    pEsquerda = pCabecaLinha->Direita;
    while ((pEsquerda->Direita != pCabecaLinha) &&
            (pEsquerda->Direita->Coluna < Coluna))
        pEsquerda = pEsquerda->Direita;

    pCabecaColuna = TMatriz_Retorna(pMatriz, -1, Coluna);
    pAcima = pCabecaColuna->Abaixo;
    while ((pAcima->Abaixo != pCabecaColuna) && (pAcima->Abaixo->Linha < Linha))
        pAcima = pAcima->Abaixo;

    if ((pAcima->Abaixo->Linha == Linha) &&
          (pEsquerda->Direita->Coluna == Coluna)) {
        pCelula = pAcima->Abaixo;
        if (x != 0.0) pCelula->Item = x;
        else return TMatriz_Retira(pMatriz, pEsquerda, pAcima, pCelula);
    }
    else if (x != 0.0)
        return TMatriz_Insere(pMatriz, pEsquerda, pAcima, Linha, Coluna, x);
    return 1;
}
```

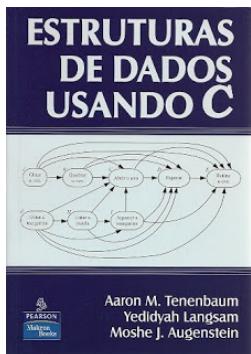
Exercício

- Escreva uma função que recebe como parâmetro uma lista linear implementada por apontadores usando encadeamento circular duplo e com célula cabeça e, sem contar explicitamente o número de itens da lista, retorna um apontador para a célula que contém o item que esteja o mais próximo possível do meio da lista, ou seja, o item na posição mediana da lista.



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