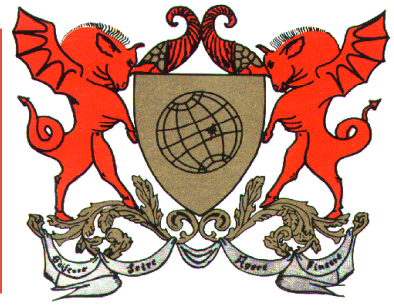


Universidade Federal de Viçosa
Campus de Florestal

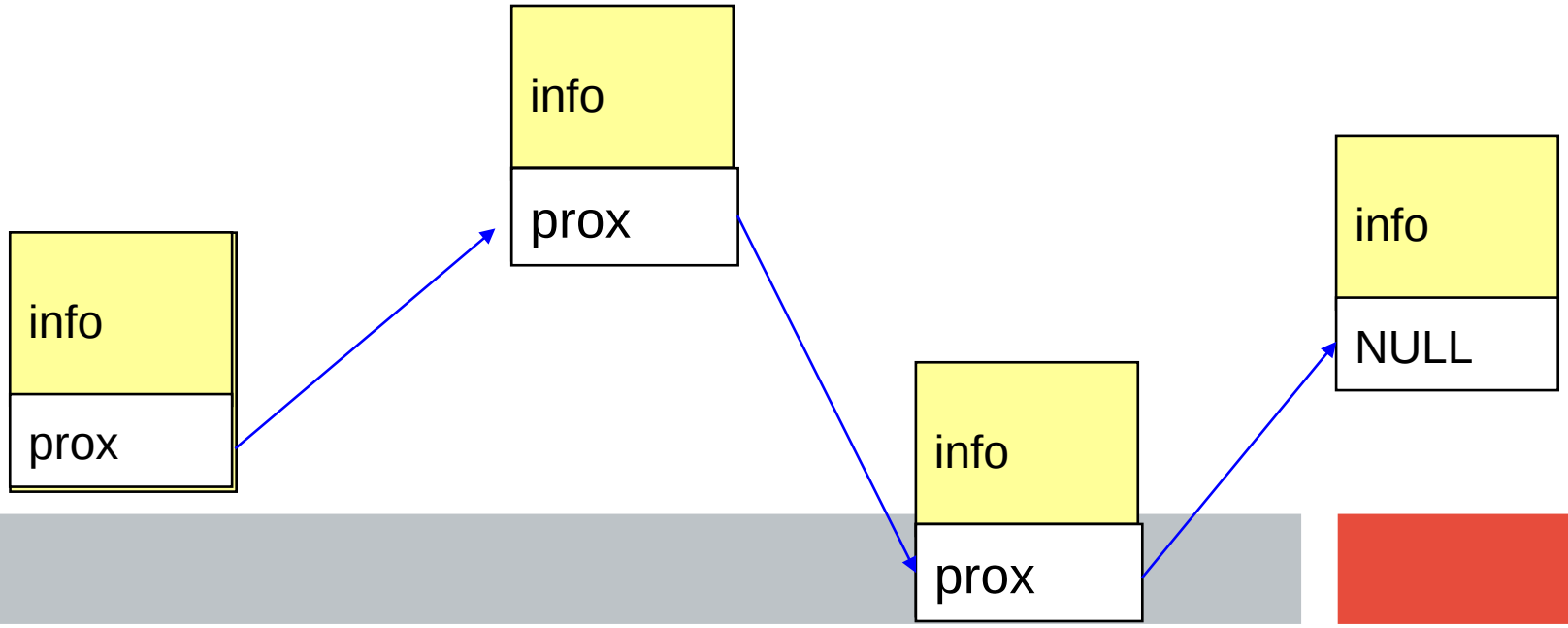
Algoritmos e Estruturas de Dados I (CCF 211)

Listas Encadeadas (Cap03 – Seção 3.1 - Ziviani)

Profa.Thais R. M. Braga Silva
<thais.braga@ufv.br>

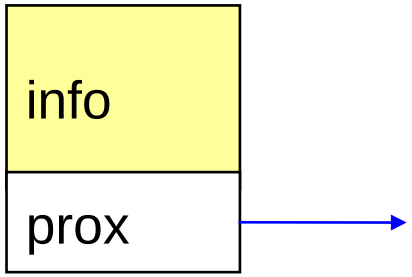
Listas Encadeadas

- Características:
  - Tamanho da lista não é pré-definido
  - Cada elemento guarda quem é o próximo (apontador)
  - Elementos não estão contíguos na memória (alocação dinâmica)
  - É possível fazer inserção e remoção sem necessidade de deslocamento de itens na memória



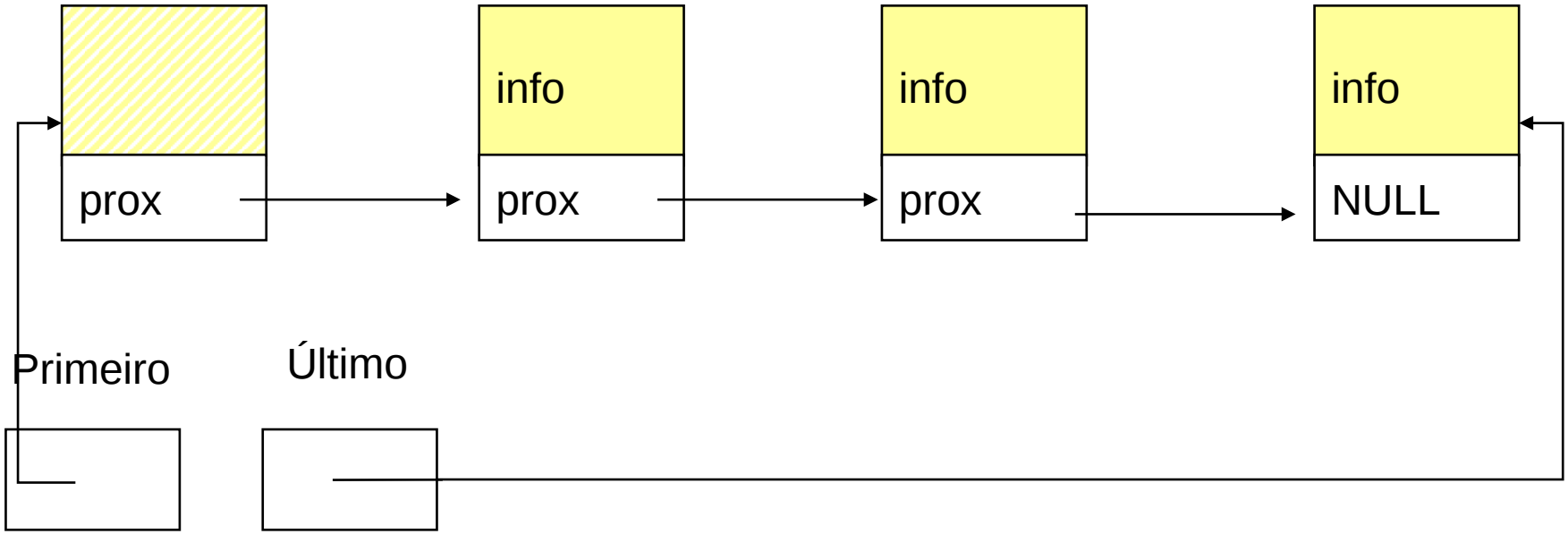
Elementos da Lista Encadeada

- Elemento: guarda as informações relacionadas
- Também é chamado de **célula**
- Para isso define-se cada elemento como uma estrutura que possui:
  - campos de informações
  - ponteiro para o próximo elemento (apontador)



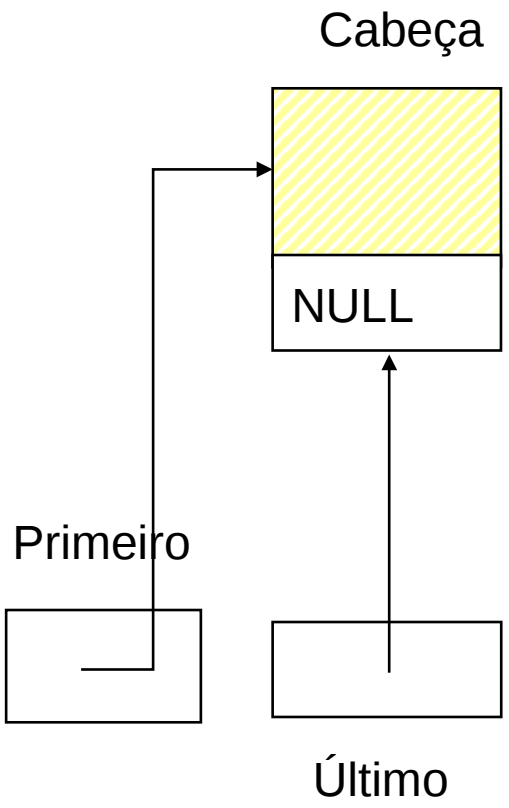
Lista Encadeada

- Uma lista **pode** ter uma célula cabeça
  - Simplifica operações sobre a lista

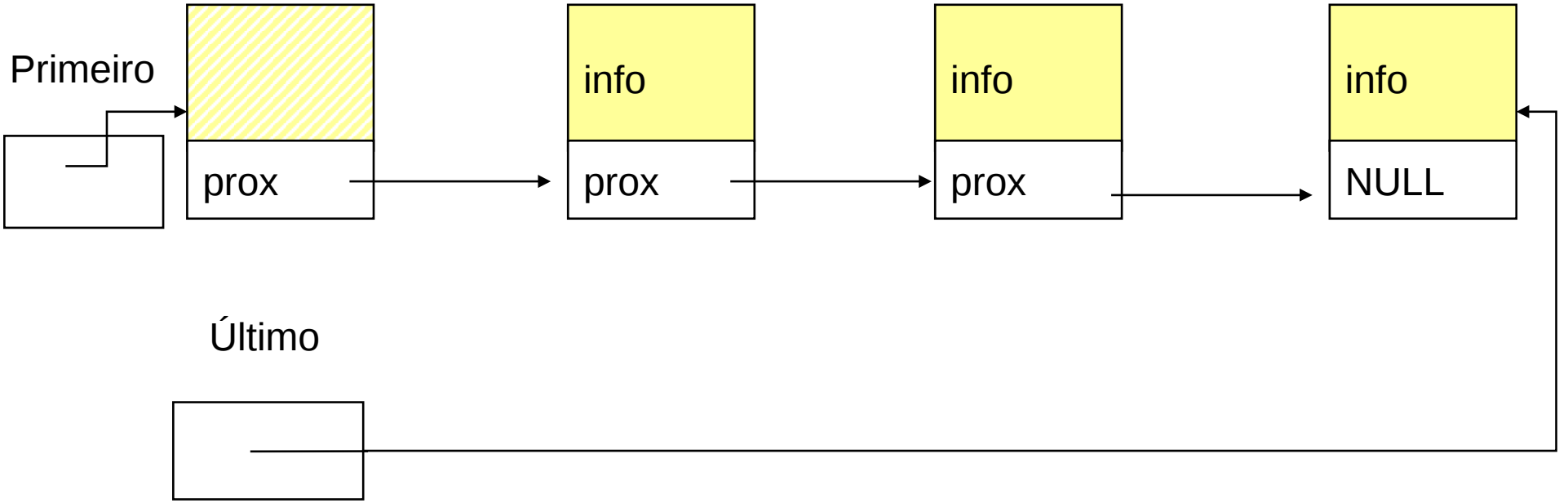


- Uma lista **pode** ter um apontador para o último elemento

Cria Lista Vazia

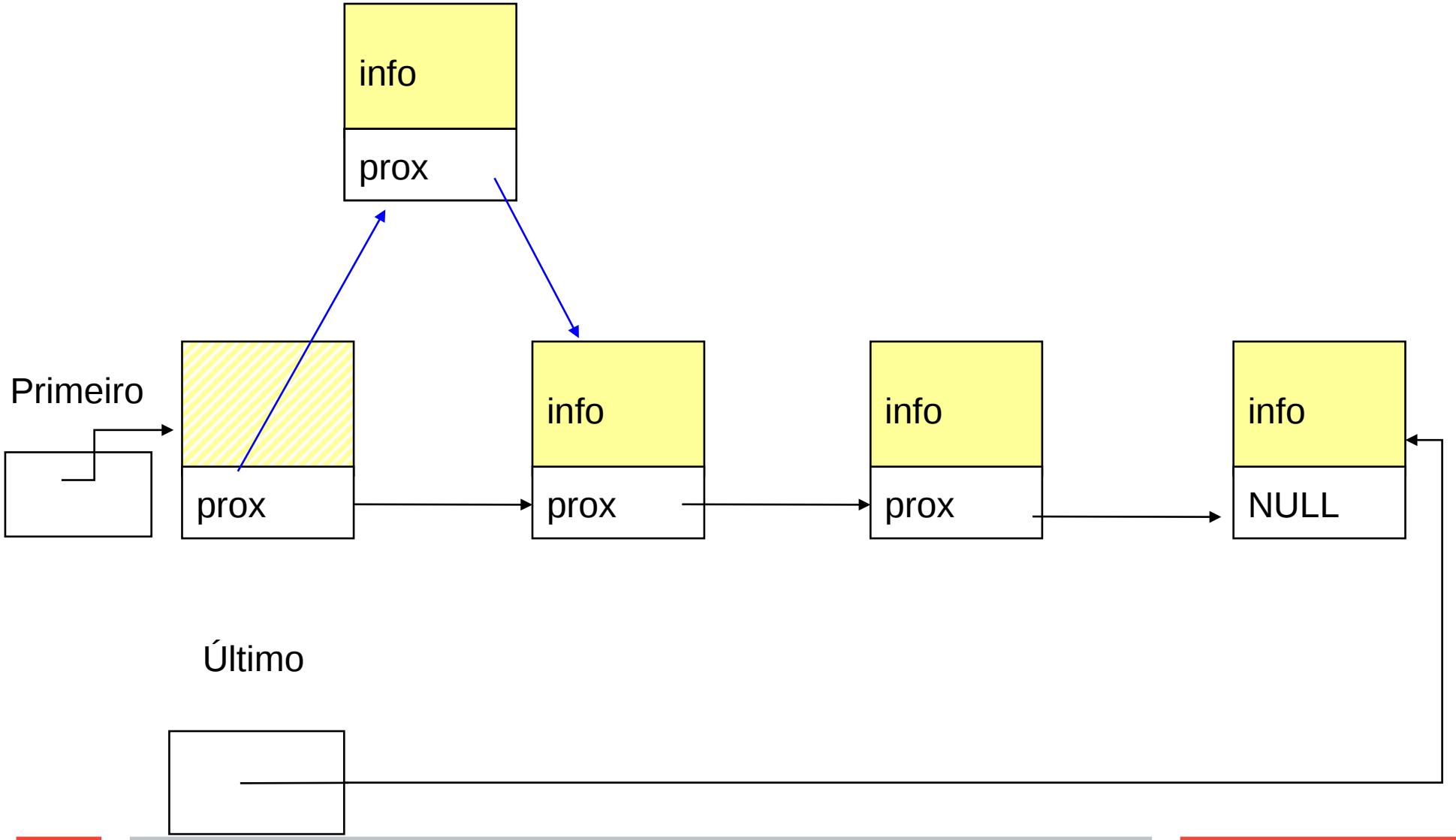


Inserção de Elementos na Lista

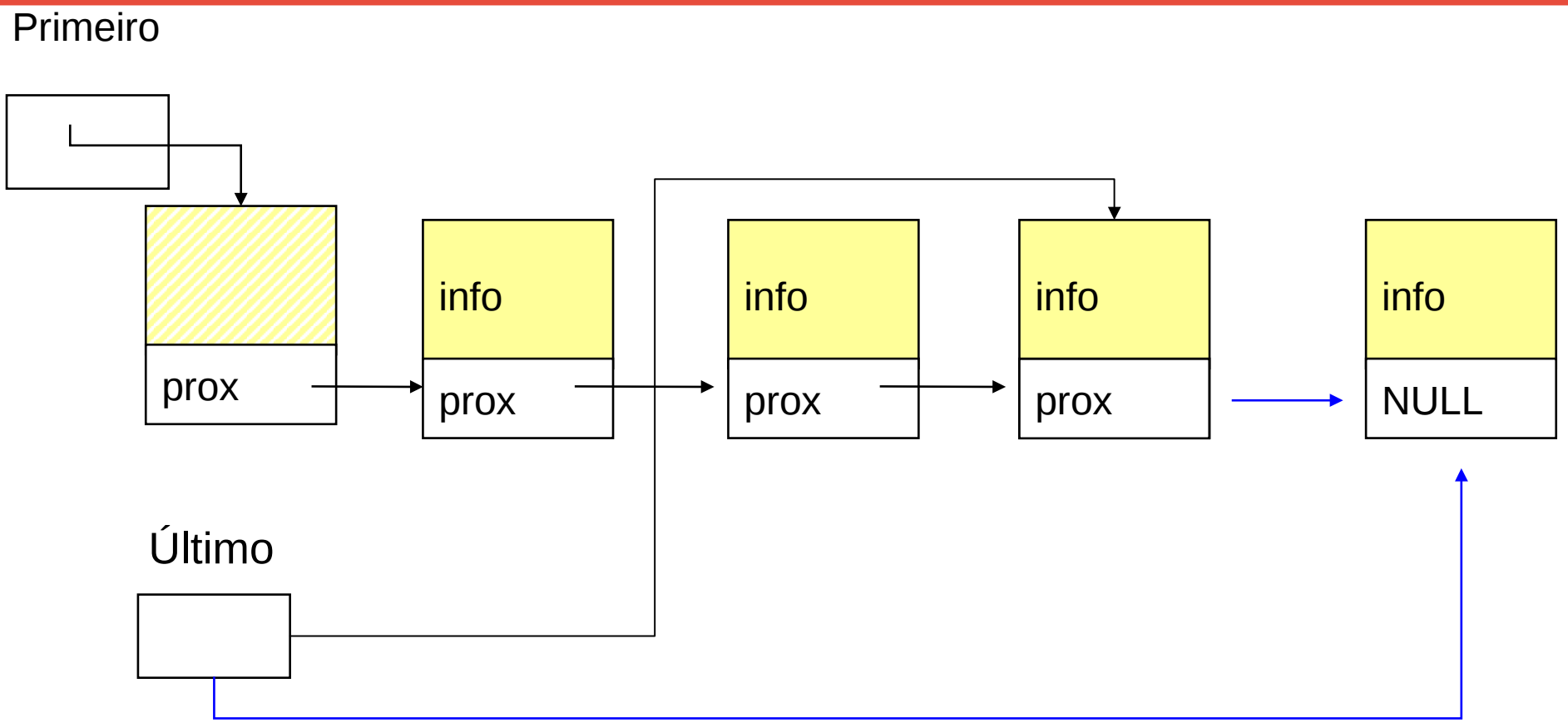


- 3 opções para inserção:
  - 1ª posição
  - última posição
  - após um elemento qualquer E

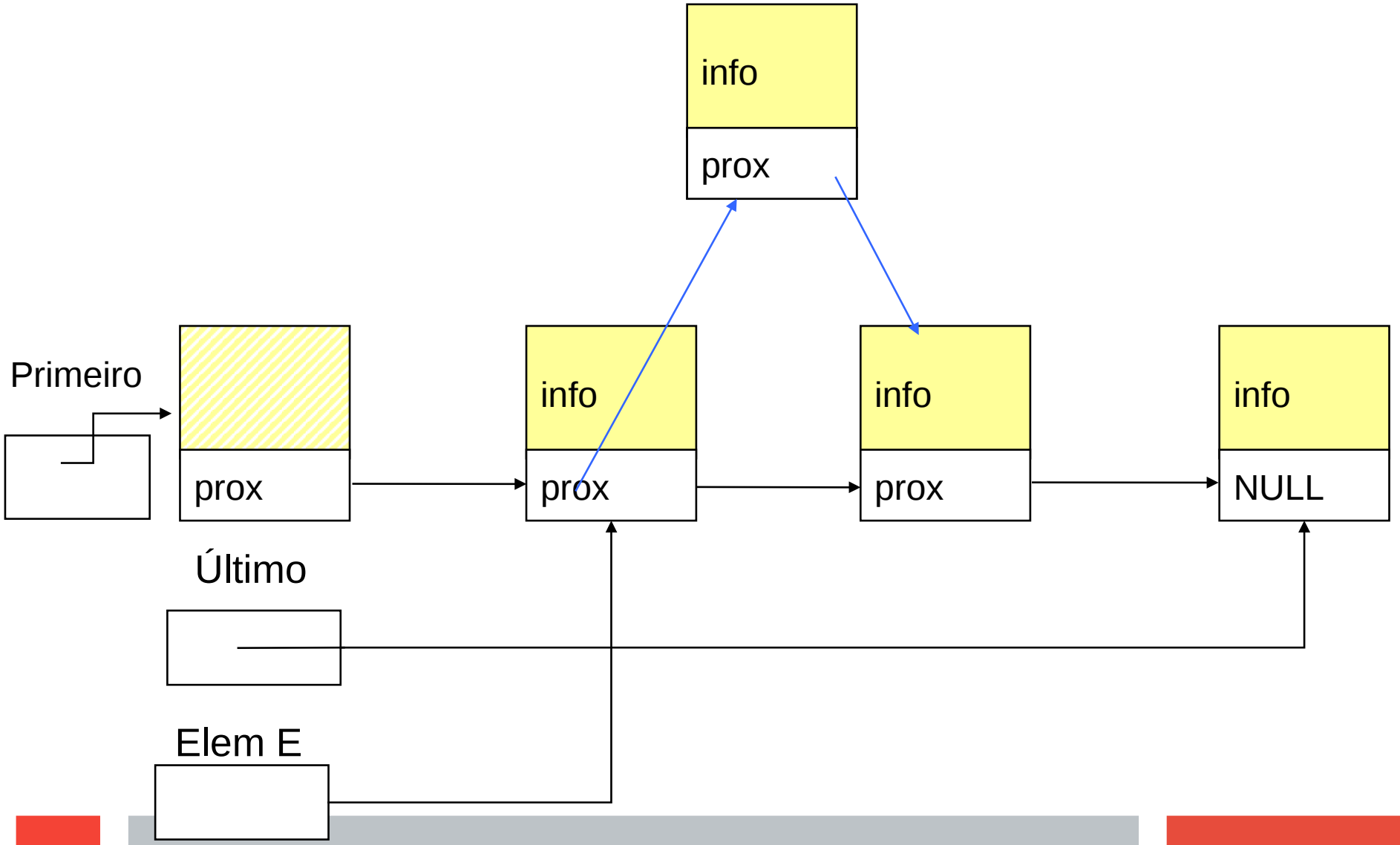
Inserção na Primeira Posição



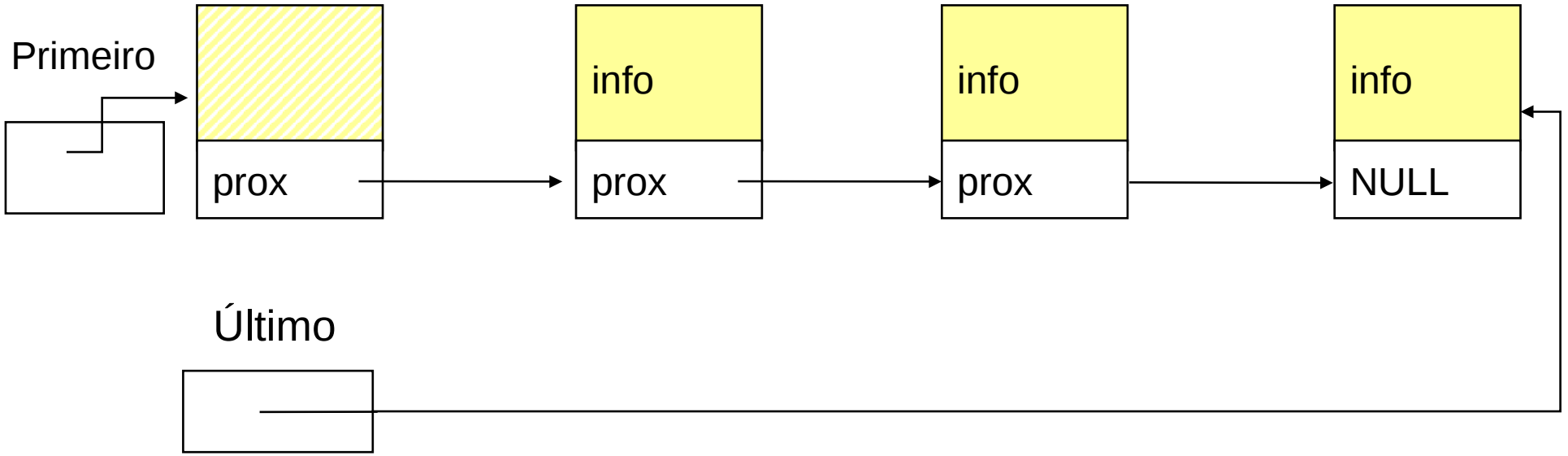
Inserção na Última Posição



Inserção Após o Elemento E

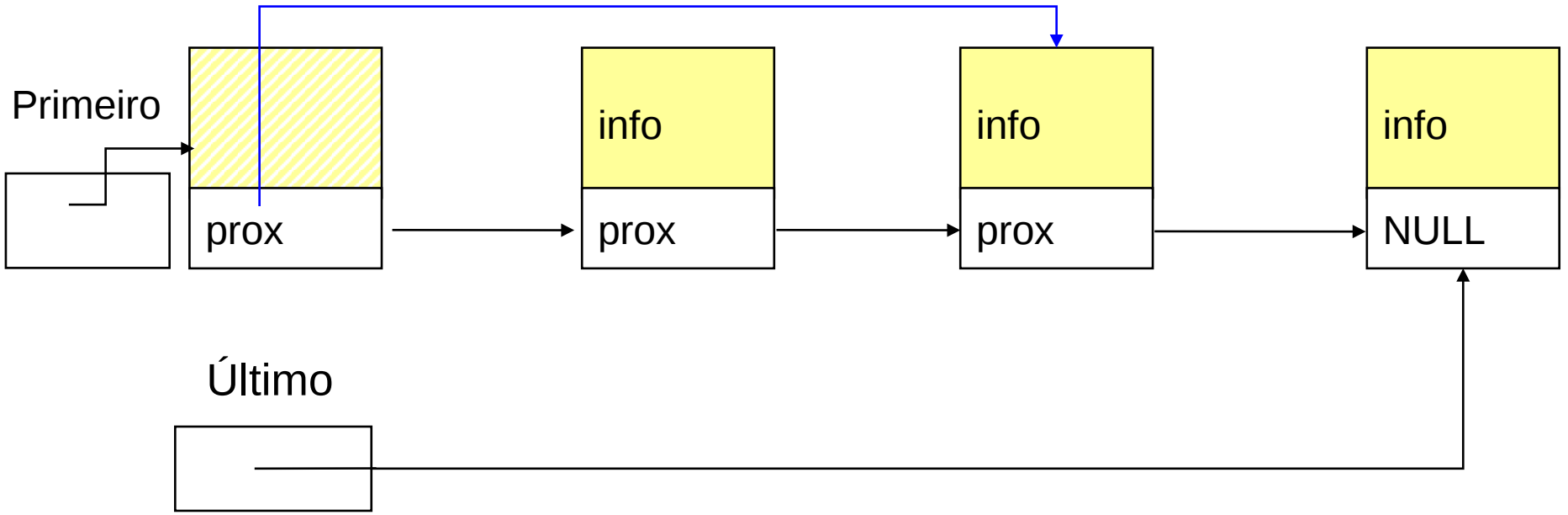


Retirada de Elementos na Lista

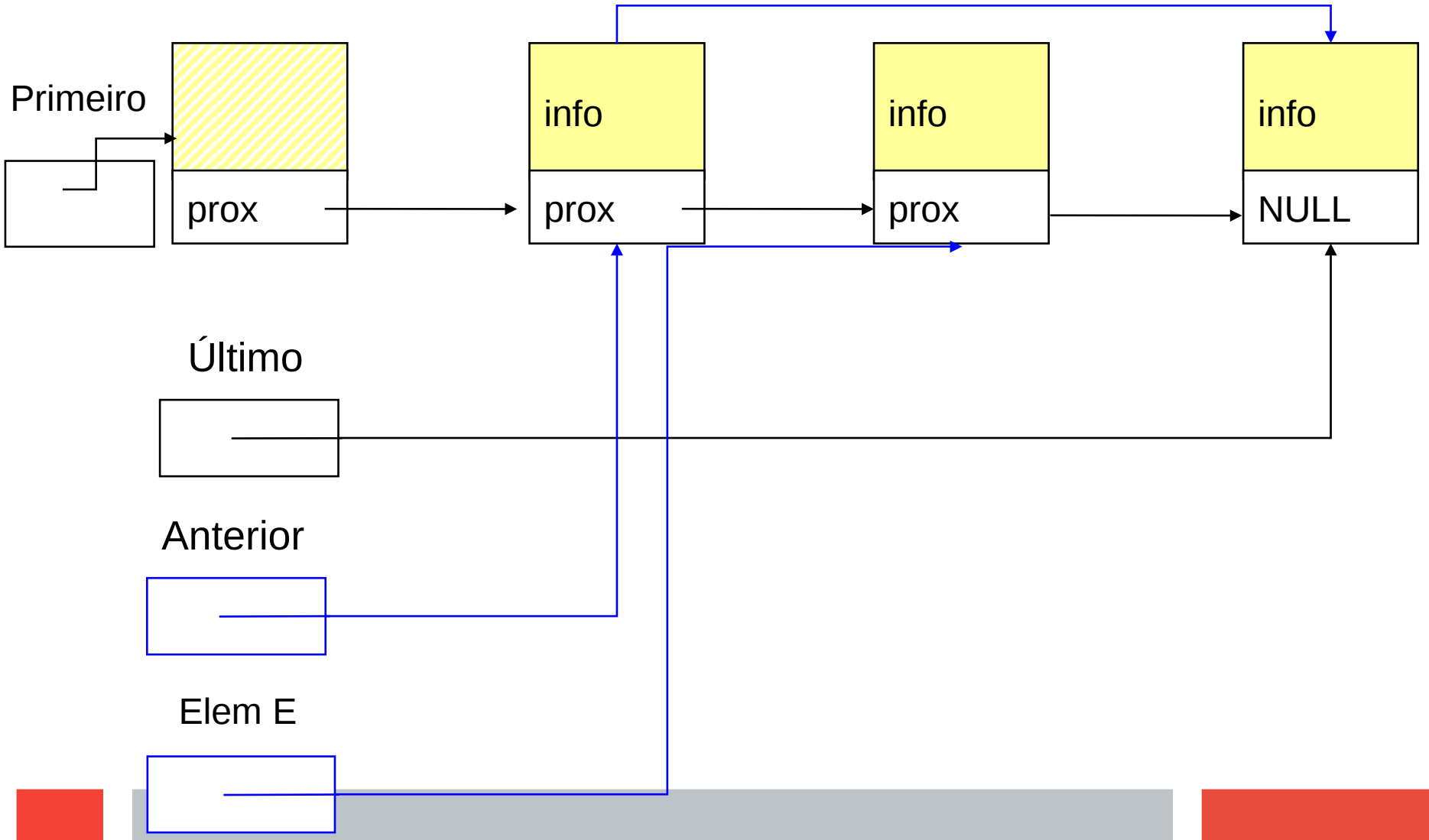


- 3 opções de remoção:
  - 1ª. posição
  - última posição
  - um elemento qualquer E

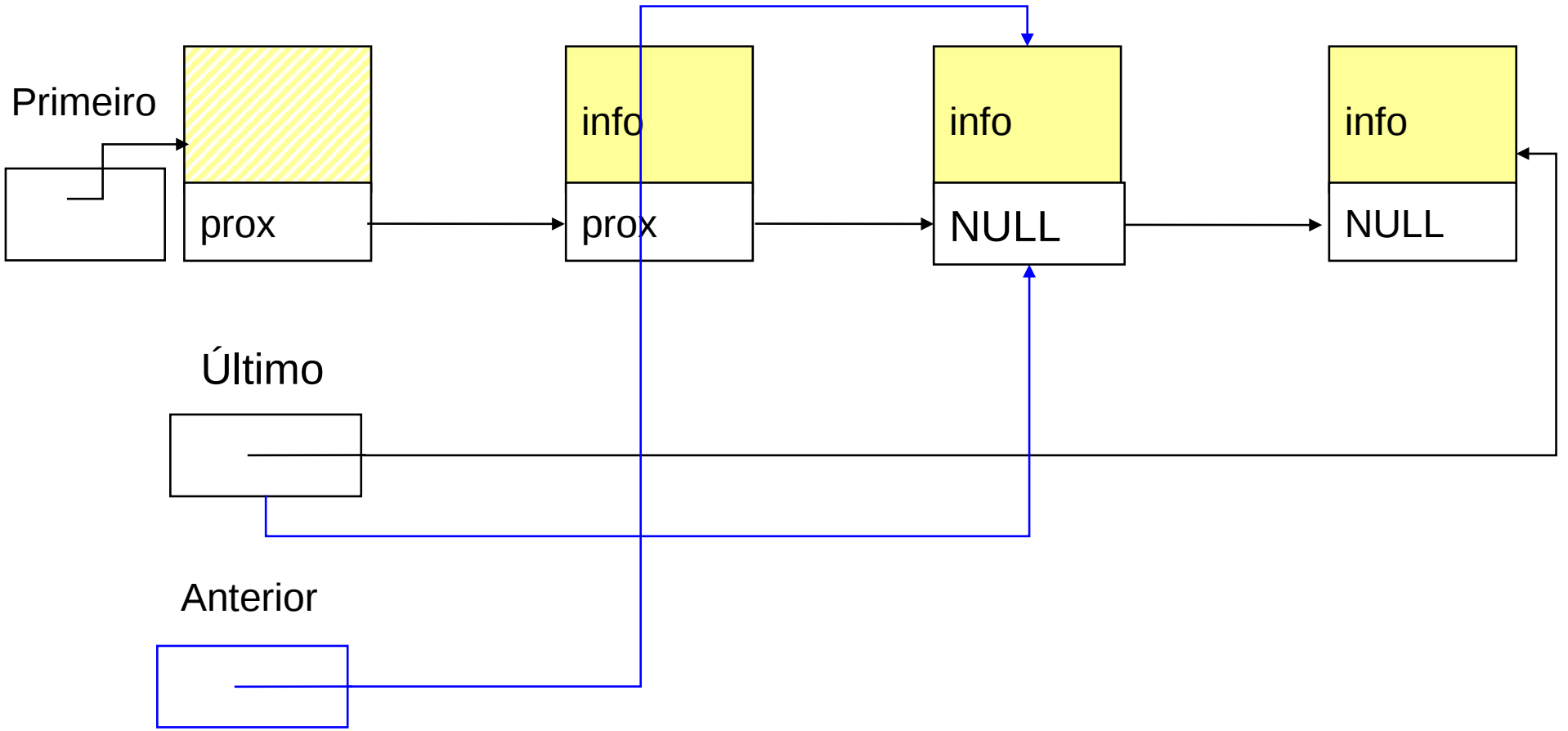
Retirada do Elemento na Primeira Posição da Lista



Retirada do Elemento E da Lista



Retirada do Último Elemento da Lista



Estrutura da Lista Usando Apontadores

arquivo.h

```
typedef int TipoChave;
typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TItem;
```

← TAD TItem

```
typedef struct Celula* Apontador;
typedef struct Celula {
  TItem Item;
  struct Celula* pProx; /* Apontador pProx; */
} TCelula;
```

← Célula

```
typedef struct {
  Apontador pPrimeiro;
  Apontador pUltimo;
} TLista;
```

← TAD TLista

Estrutura da Lista Usando Apontadores

arquivo.h

```
void FLVazia(TLista* pLista);

int LEhVazia(TLista* pLista);

int LInsere(TLista* pLista, TItem *pItem);

int LRetira(TLista* pLista, TItem *pItem);

void LImprime(TLista* pLista);
```

Operações sobre Lista Usando Apontadores (com Cabeça)

```
void FLVazia(TLista* pLista)
{
  pLista->pPrimeiro = (Apontador) malloc(sizeof(TCelula));
  pLista->pUltimo = pLista->pPrimeiro;
  pLista->pPrimeiro->pProx = NULL;
}

int LEhVazia(TLista* pLista)
{
  return (pLista->pPrimeiro == pLista->pUltimo);
}

void LInsere(TLista *pLista,TItem* pItem)
{
  pLista->pUltimo->pProx = (Apontador) malloc(sizeof(TCelula));
  pLista->pUltimo = pLista->pUltimo->pProx;
  pLista->pUltimo->Item = *pItem;
  pLista->pUltimo->pProx = NULL;
}
```

Operações sobre Lista Usando Apontadores (sem Cabeça)

```
void FLVazia(TLista *pLista){
  pLista->pPrimeiro = NULL;
  pLista->pUltimo   = NULL;
}

int LEhVazia(TLista* pLista)
{ return (pLista->pUltimo == NULL); }

void LInsere(TLista* pLista,TItem* pItem){
  if (pLista->pUltimo == NULL)
  { pLista->pUltimo = (Apontador) malloc(sizeof(TCelula));
    pLista->pPrimeiro = pLista->pUltimo; }
  else
  { pLista->pUltimo->pProx = (Apontador) malloc(sizeof(TCelula));
    pLista->pUltimo = pLista->pUltimo->pProx;
  }
  pLista->pUltimo->Item = *pItem;
  pLista->pUltimo->pProx = NULL;
}
```

Operações sobre Lista Usando Apontadores (com Cabeça)

```
int LRetira(TLista* pLista, TItem* pItem)
{
  TCelula* pAux;
  if (LEhVazia(pLista))
    return 0;
  *pItem = pLista->pPrimeiro->pProx->Item;
  pAux = pLista->pPrimeiro;
  pLista->pPrimeiro = pLista->pPrimeiro->pProx;
  free(pAux);
  return 1;
}
```

Operações sobre Lista Usando Apontadores (sem Cabeça)

```
int LRetira(TLista* pLista, TItem* pItem)
{
  TCelula* pAux;
  if (LEhVazia(pLista))
    return 0;
  *pItem = pLista->pPrimeiro->Item;
  pAux = pLista->pPrimeiro;
  pLista->pPrimeiro = pLista->pPrimeiro->pProx;
  free(pAux);
  if (pLista->pPrimeiro == NULL)
    pLista->pUltimo = NULL; /* lista vazia */
  return 1;
}
```

Operações sobre Lista Usando Apontadores (com Cabeça)

```
void LImprime(TLista* pLista)
{
  Apontador pAux;
  pAux = pLista->pPrimeiro->pProx;
  while (pAux != NULL)
  {
    printf("%d\n", pAux->Item.Chave);
    pAux = pAux->pProx; /* próxima célula */
  }
}
```

Operações sobre Lista Usando Apontadores (sem Cabeça)

```
void LImprime(TLista* pLista)
{
  Apontador pAux;
  pAux = pLista->pPrimeiro;
  while (pAux != NULL)
  {
    printf("%d\n", pAux->Item.Chave);
    pAux = pAux->pProx; /* próxima célula */
  }
}
```

Operações sobre Lista Usando Apontadores

- **Vantagens:**
  - Permite inserir ou retirar itens do meio da lista a um custo constante (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).

- **Desvantagem:**
  - Utilização de memória extra para armazenar os apontadores.
  - Percorrer a lista, procurando pelo *i*-ésimo elemento.

Exercícios

- Escreva uma função que receba uma lista como parâmetro e retira seu último elemento
- Escreva uma função que receba uma lista e um ponteiro para uma célula como parâmetros e insira a célula na primeira posição da lista