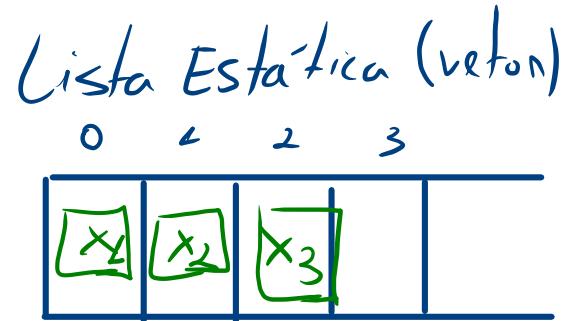


Listas Encadeadas



Listas Encadeadas

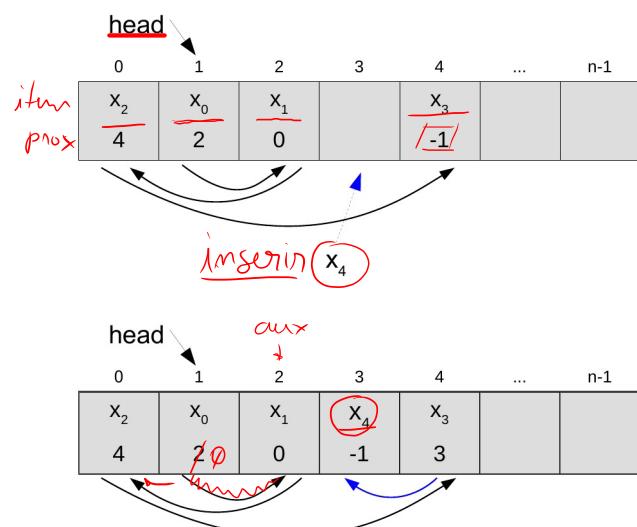
- Elementos consecutivos na lista não implica em elementos consecutivos na representação. A ordem é lógica.
- Cada item da lista contém a informação que é necessária para alcançar o próximo item
- Na implementação é necessário armazenar separadamente a informação do primeiro elemento da lista



Listas Encadeadas

- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista
- Existem duas formas de representar listas encadeadas, através de:
 - arrays, denominada lista encadeada estática
 - por referências (ponteiros), denominada lista encadeada dinâmica

Listas Encadeadas Estáticas



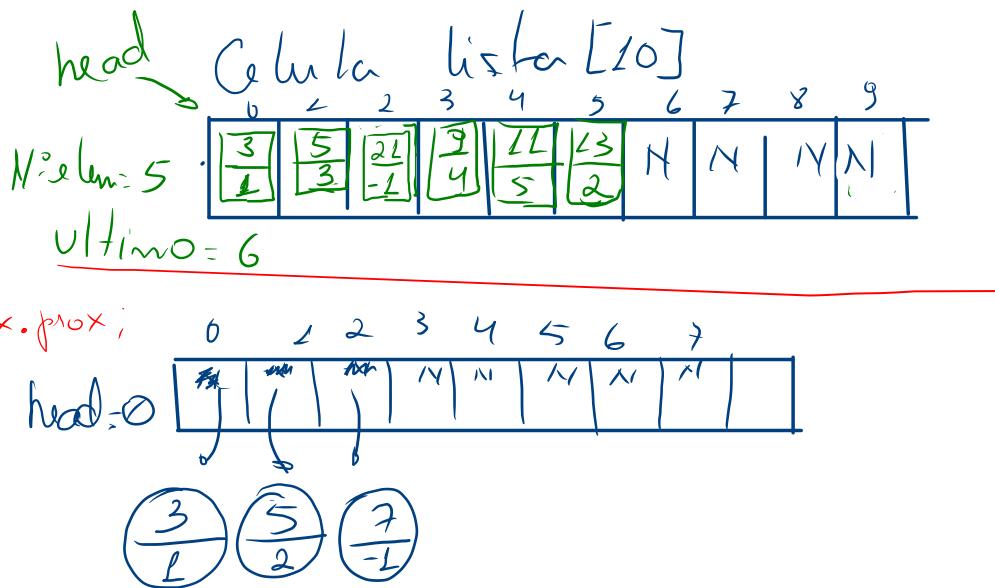
retirar x_1

$head.prox = aux.prox;$
 $aux = null;$

4

```
struct Celula {
    Elemento e;
    int prox;
};
```

linguagem C
 Prox armaz pos.vetor



Listas Encadeadas

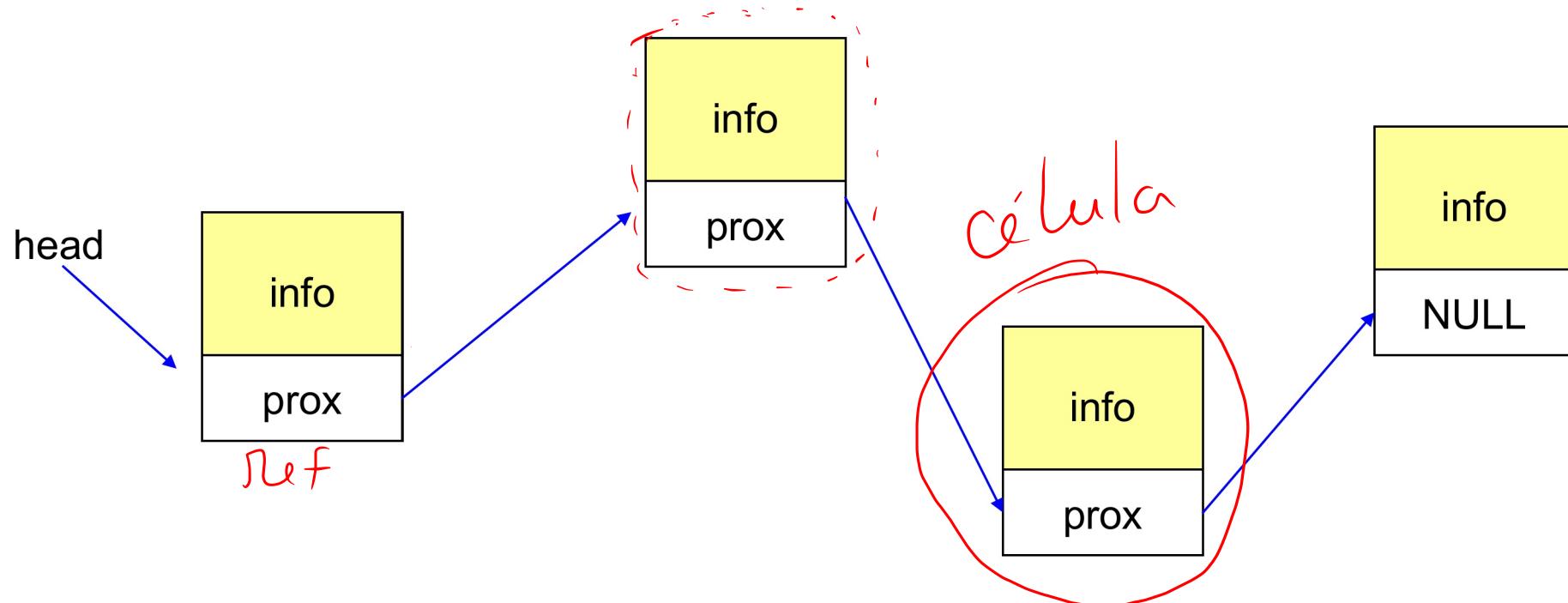
- Vantagens
 - Elimina o problema dos deslocamentos de células
 - Não é necessário saber previamente o número de elementos a serem armazenados (lista dinâmica)
- Desvantagens
 - Não se consegue acessar os elementos da lista em tempo constante
 - Mais operações para manter integridade dos dados

Listas Encadeadas Dinâmicas

- Cada item da lista é encadeado com o seguinte através de uma variável de referência (ponteiro)
- Esta implementação permite utilizar posições não contíguas de memória, sendo possível inserir e retirar elementos sem deslocar os itens da lista
- A lista é constituída de células, onde cada célula contém um item da lista e um apontador para a célula seguinte
Referência
- É conveniente utilizar listas encadeadas dinâmicas em aplicações em que não existe previsão sobre o crescimento da lista, por não ser necessário definir o tamanho da lista a priori

Listas Encadeadas Dinâmicas

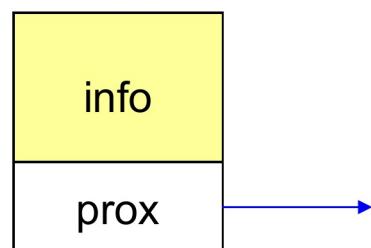
- Características:
 - Células não estão contíguas na memória



Desvantagem: utilização de memória extra para as referências

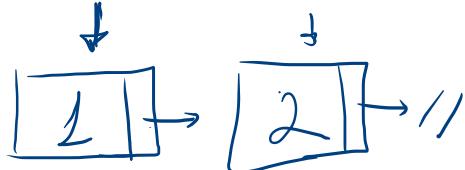
Sobre as Célula da Lista

- Célula: guarda as informações sobre cada elemento.
- Para isso define-se cada célula como uma estrutura que possui:
 - Campos de informações (*dado que será armazenado*)
 - Ponteiro (referência) para a próxima célula



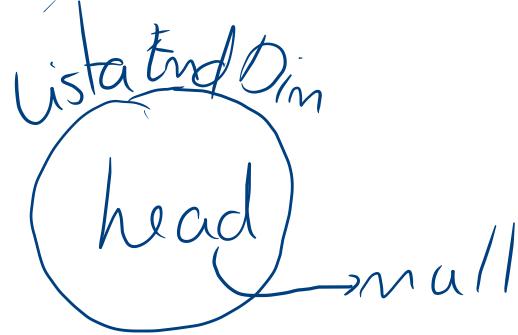
- Lista Estática (ArrayList)
 - ↳ lista dentro do vetor
- Lista Encadeada Estática
- Lista Encadeada Dinâmica (Simplesmente) (LinkedList)

head

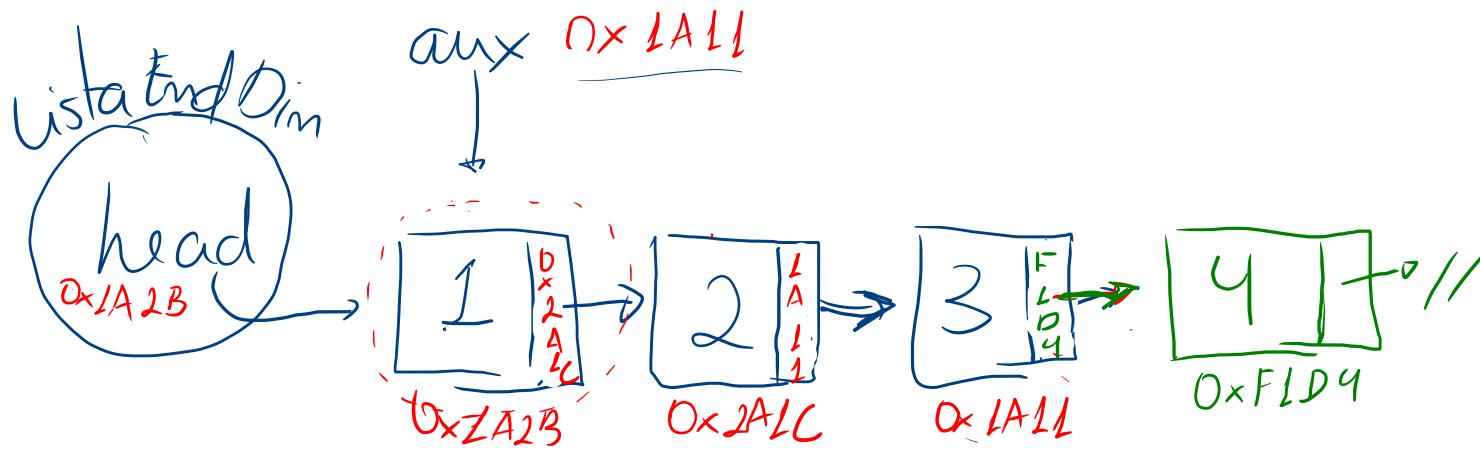
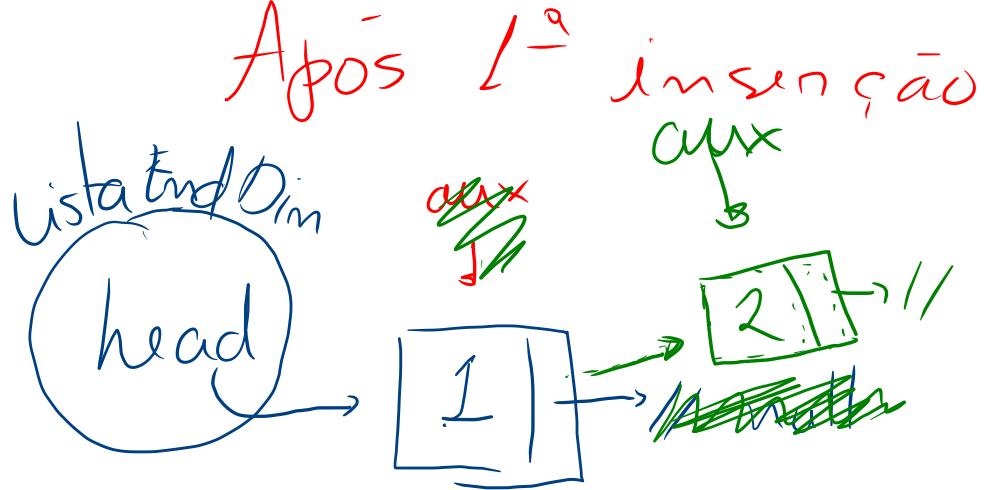


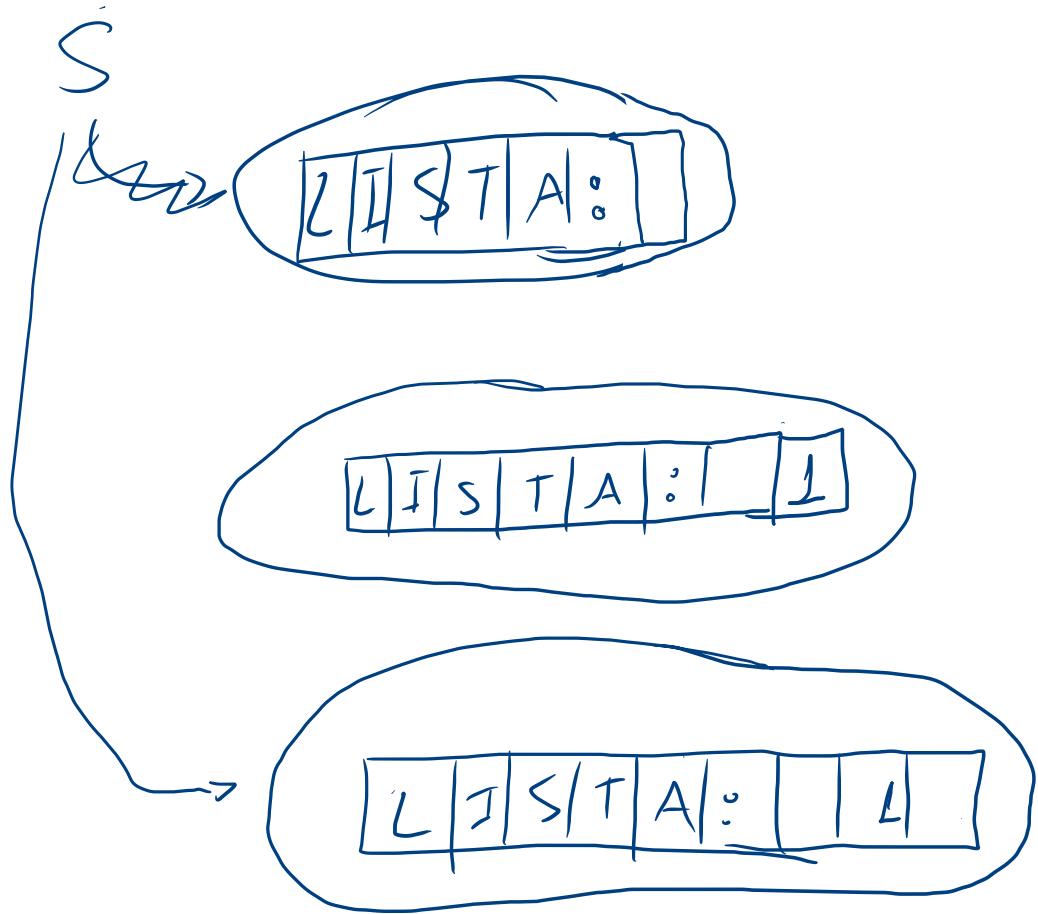
0	1	2	3
$\frac{x_1}{2}$	$\frac{x_3}{3}$	$\frac{x_2}{1}$	$\frac{x_4}{-1}$



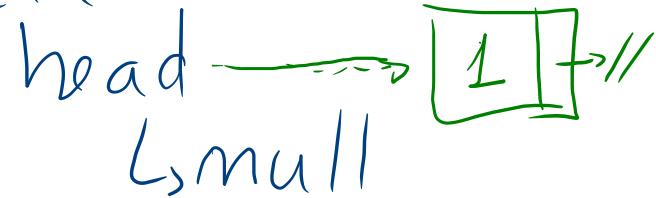


Vazio

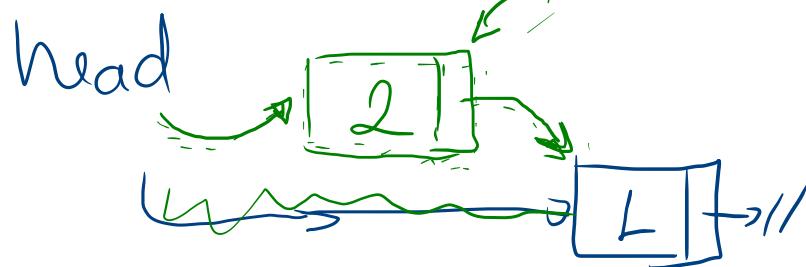




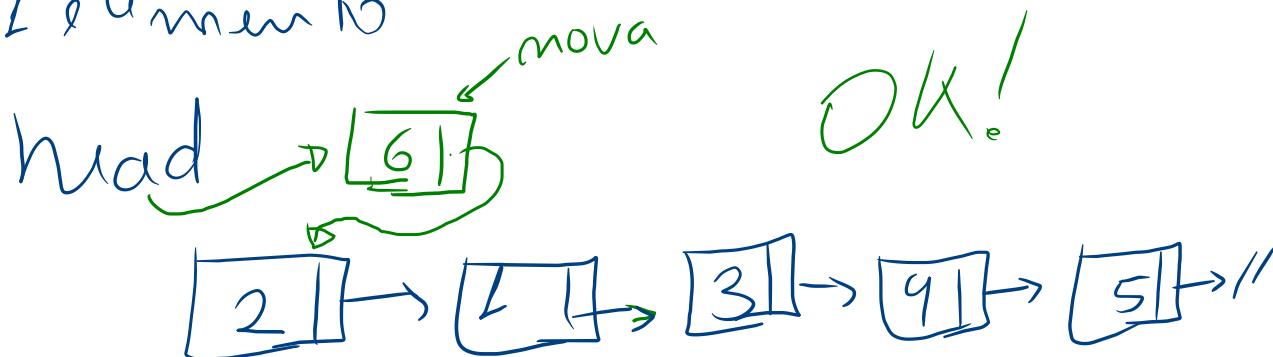
• Vazia



• 1 elemento



• + 1 elemento



Lista Enc. Dinâmica com tail (nabo)

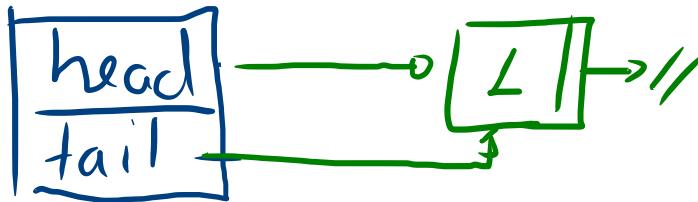
- Inserção no Final em $O(1)$

X

- Vazia

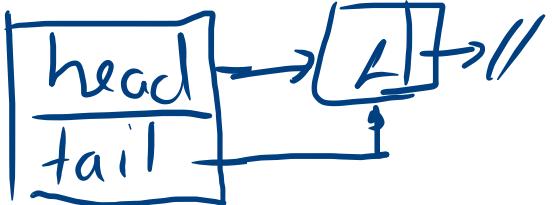


1^{a} inserção
⇒

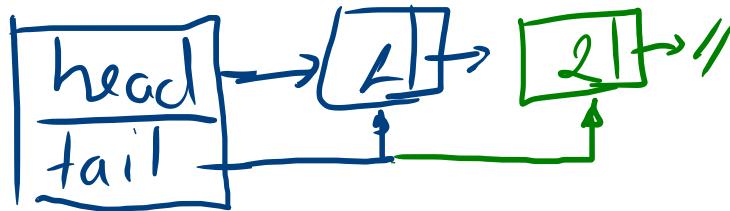


ADD

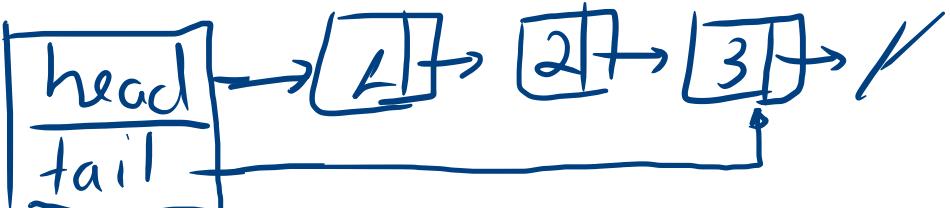
- 1 elemento



⇒



- +1 elemento



Lista Enc. Dinâmica com tail (rabo)

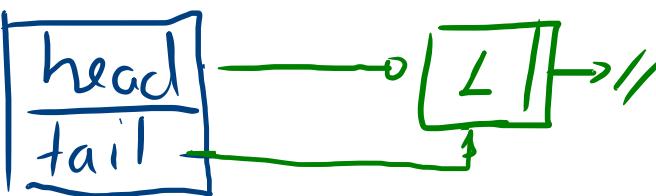
- Inserção no Final em $O(1)$

X

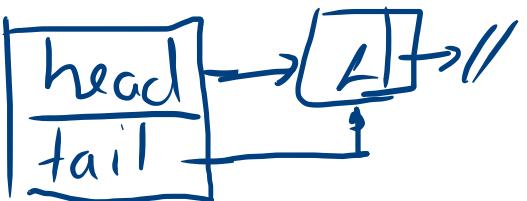
- Vazia



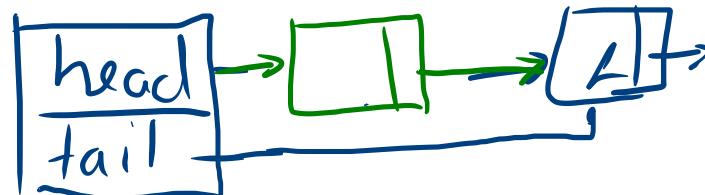
1ª inserção
⇒



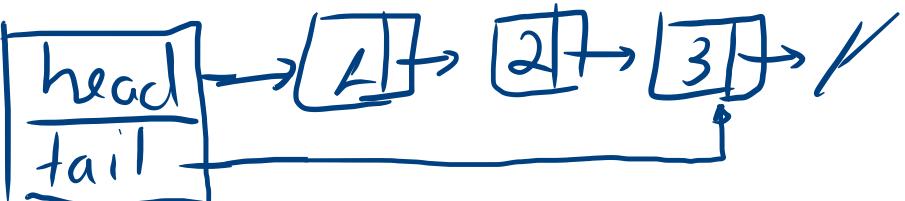
- 1 elemento



⇒



- +1 elemento

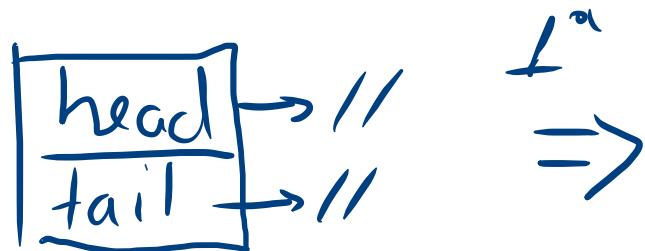


INSERT

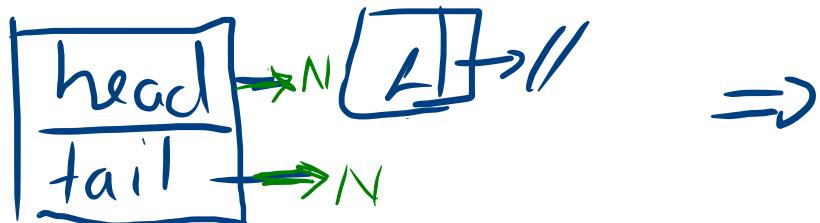
Lista Enc. Dinâmica com tail (nabo)

- Exclusão

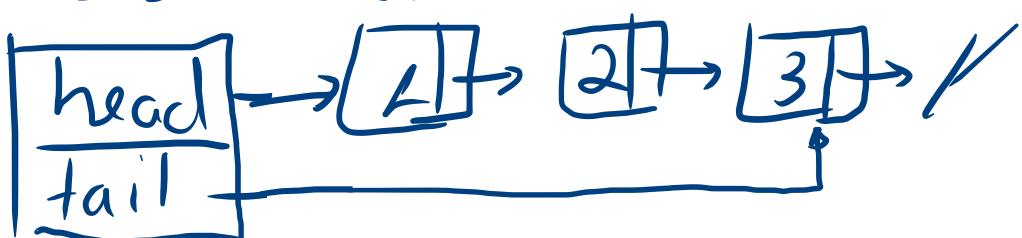
- Vazia



- 1 elemento, aux aux=N



- +1 elemento

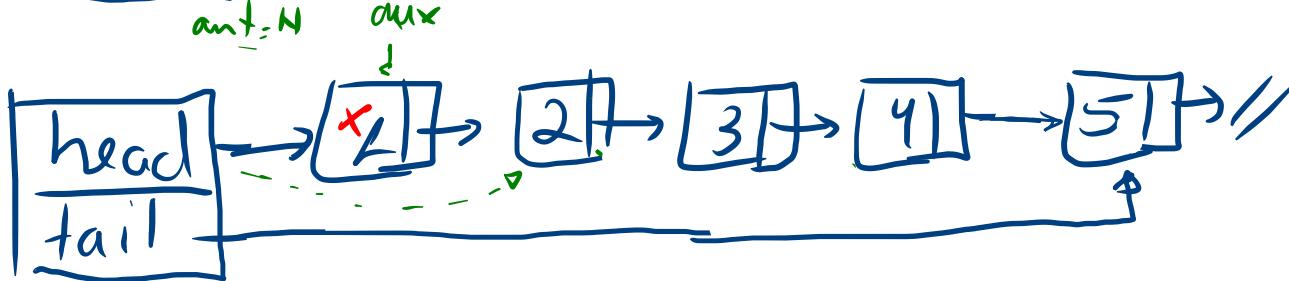
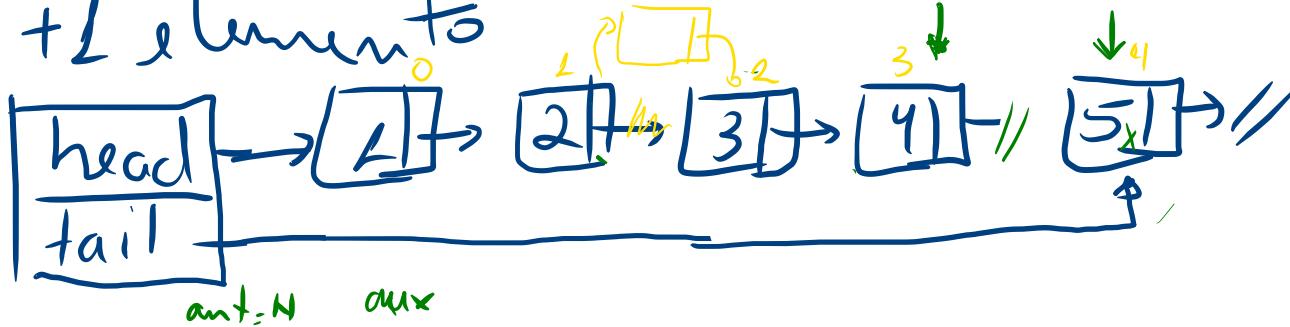


-início OK!
-meio *OK!
-Final OK!

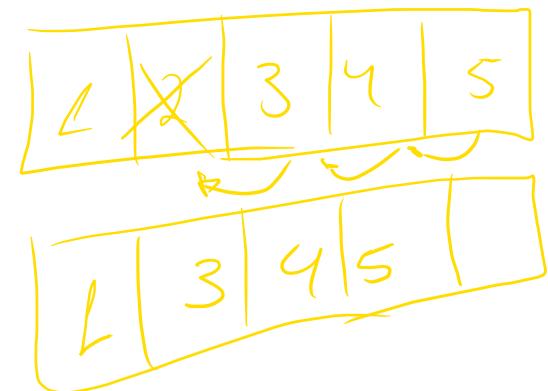
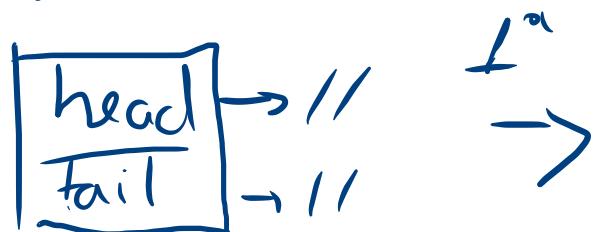
lista Enc. Dinâmica com tail (nabo)

• Exclusão

- +1 elemento



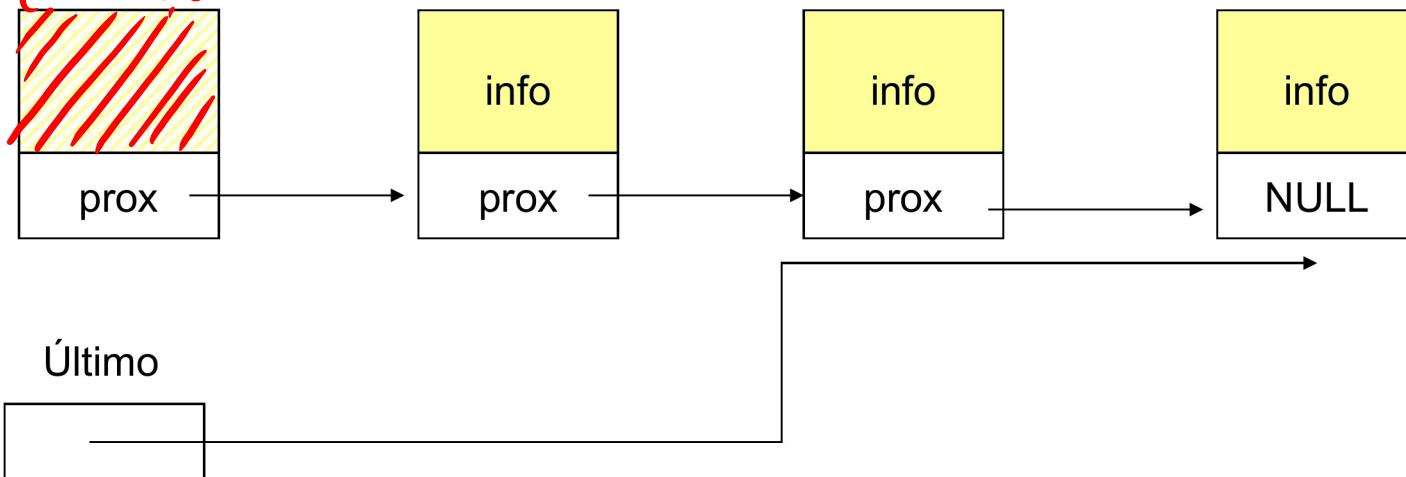
- Vazia



Sobre a Lista *com* sentinela (com cabeça) *encabeçada*

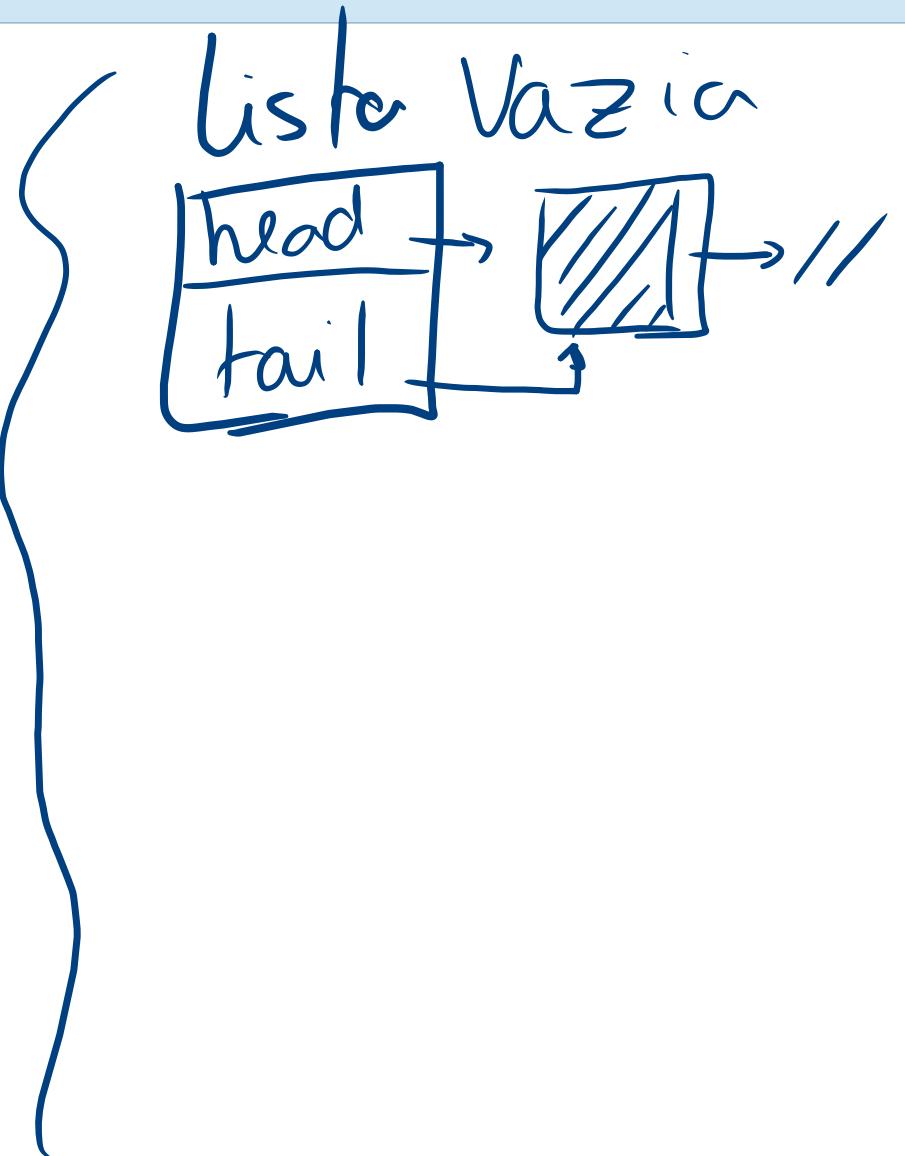
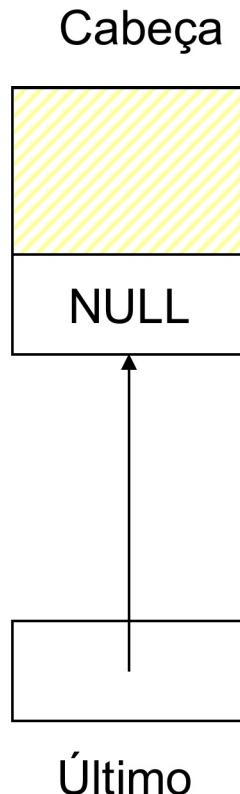
- Uma lista pode ter uma célula cabeça

*sentinela
cabeça*

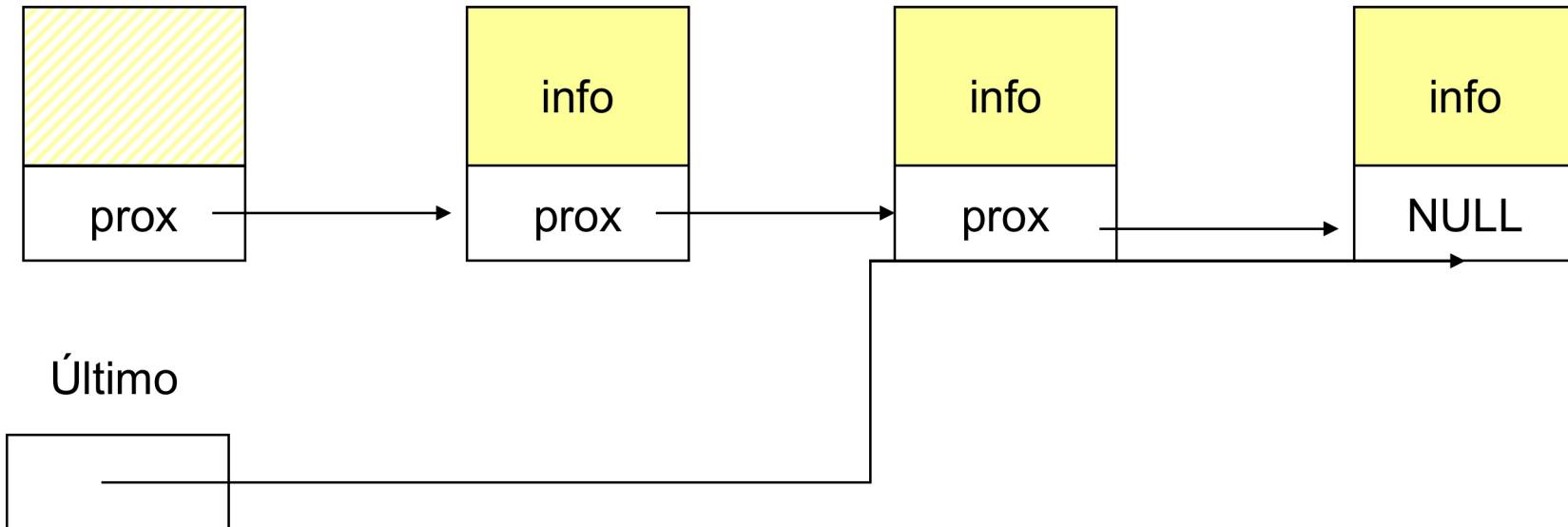


- Uma lista pode ter um apontador para o último elemento

Cria Lista Vazia



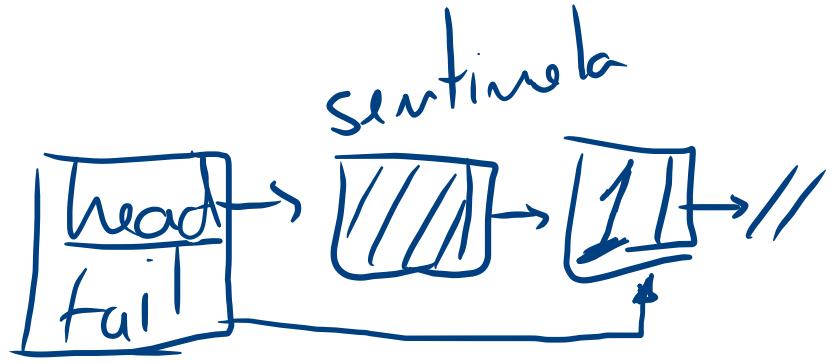
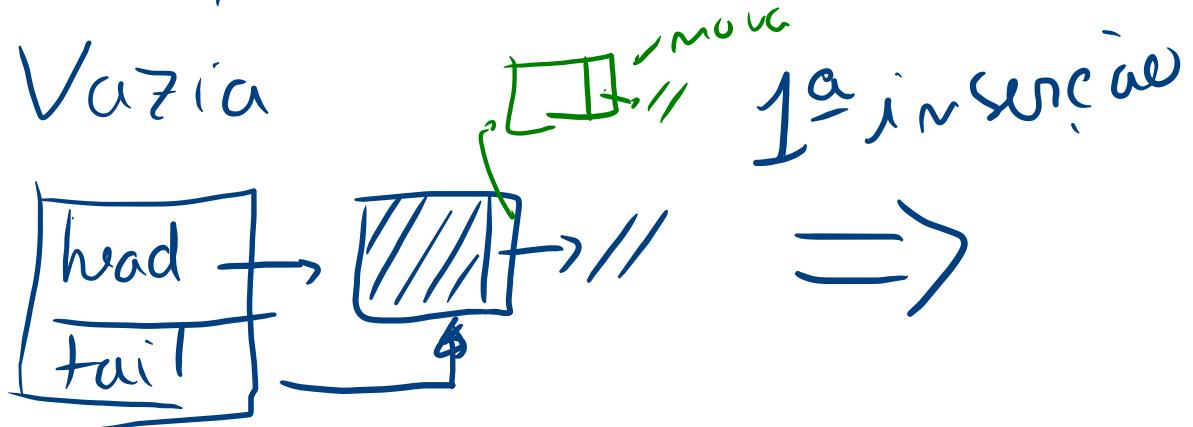
Inserção de Elementos na Lista



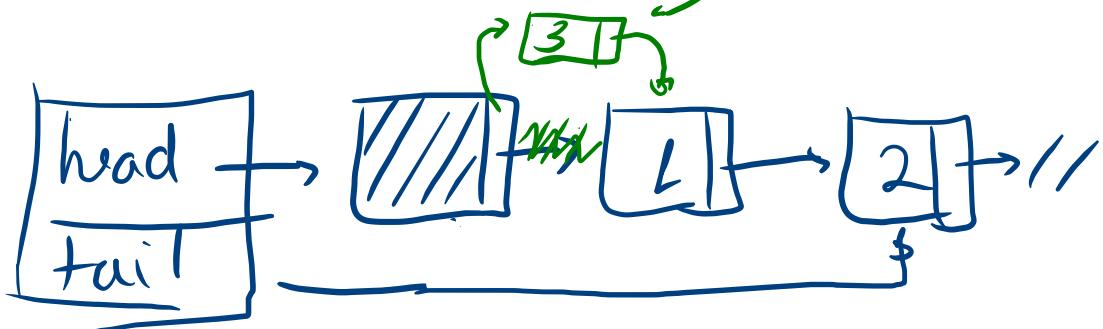
- 3 opções de posição onde pode inserir:
 - 1^a. posição
 - última posição
 - Após um elemento qualquer E

Lista c/ sentinela

- Vazia



- Inserção inicio move

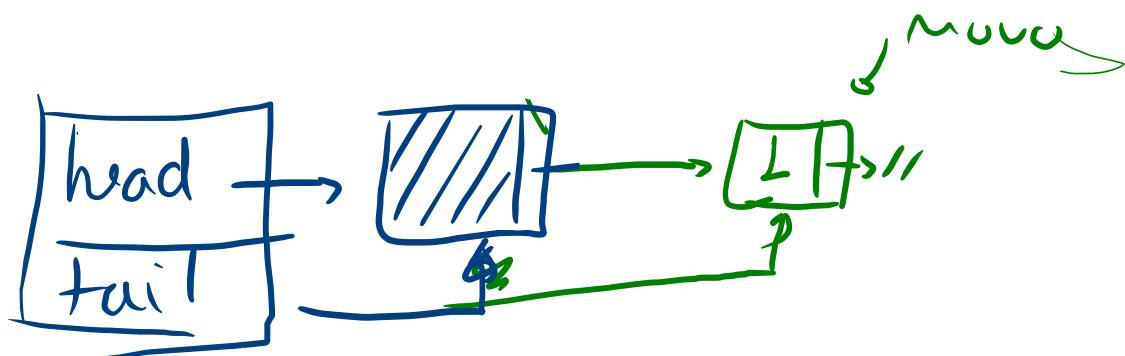
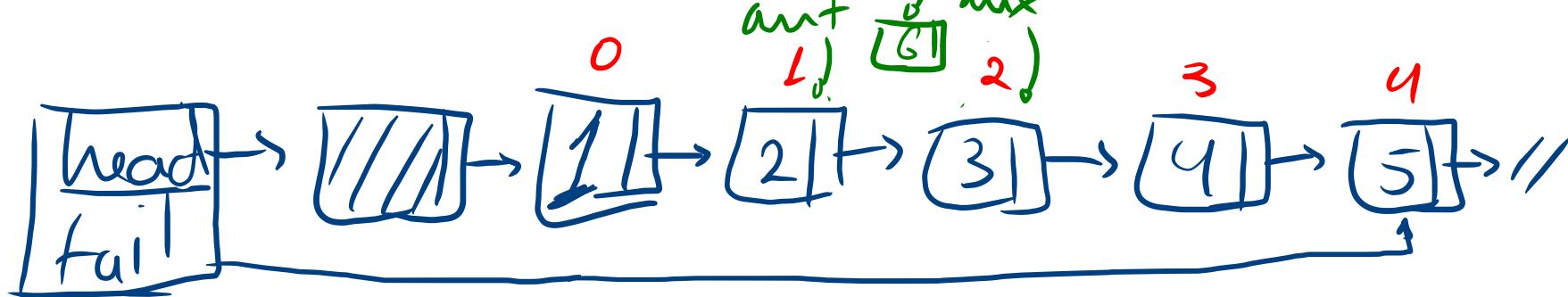


- Inserção no meio

- Inserção no Fim

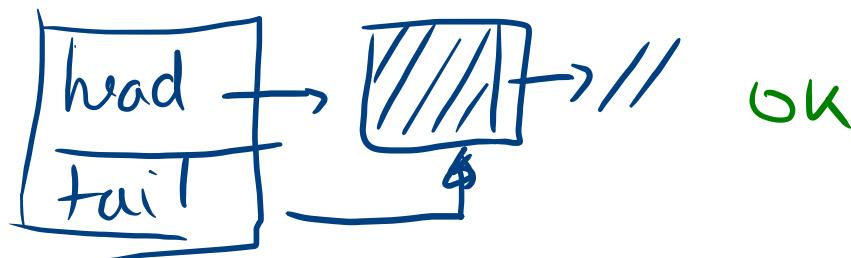
Inserção em posição nova cont = $\emptyset \neq 2$

90?

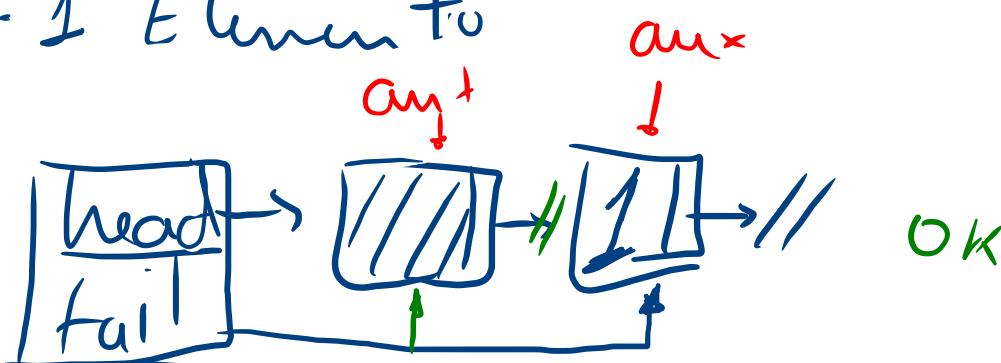


Lista c/ sentinela

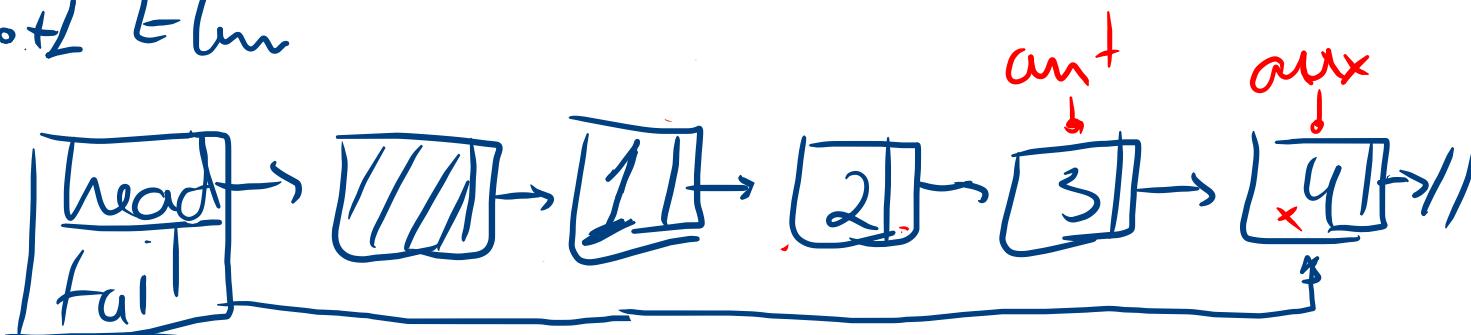
- Vazia



- 1 Elemento

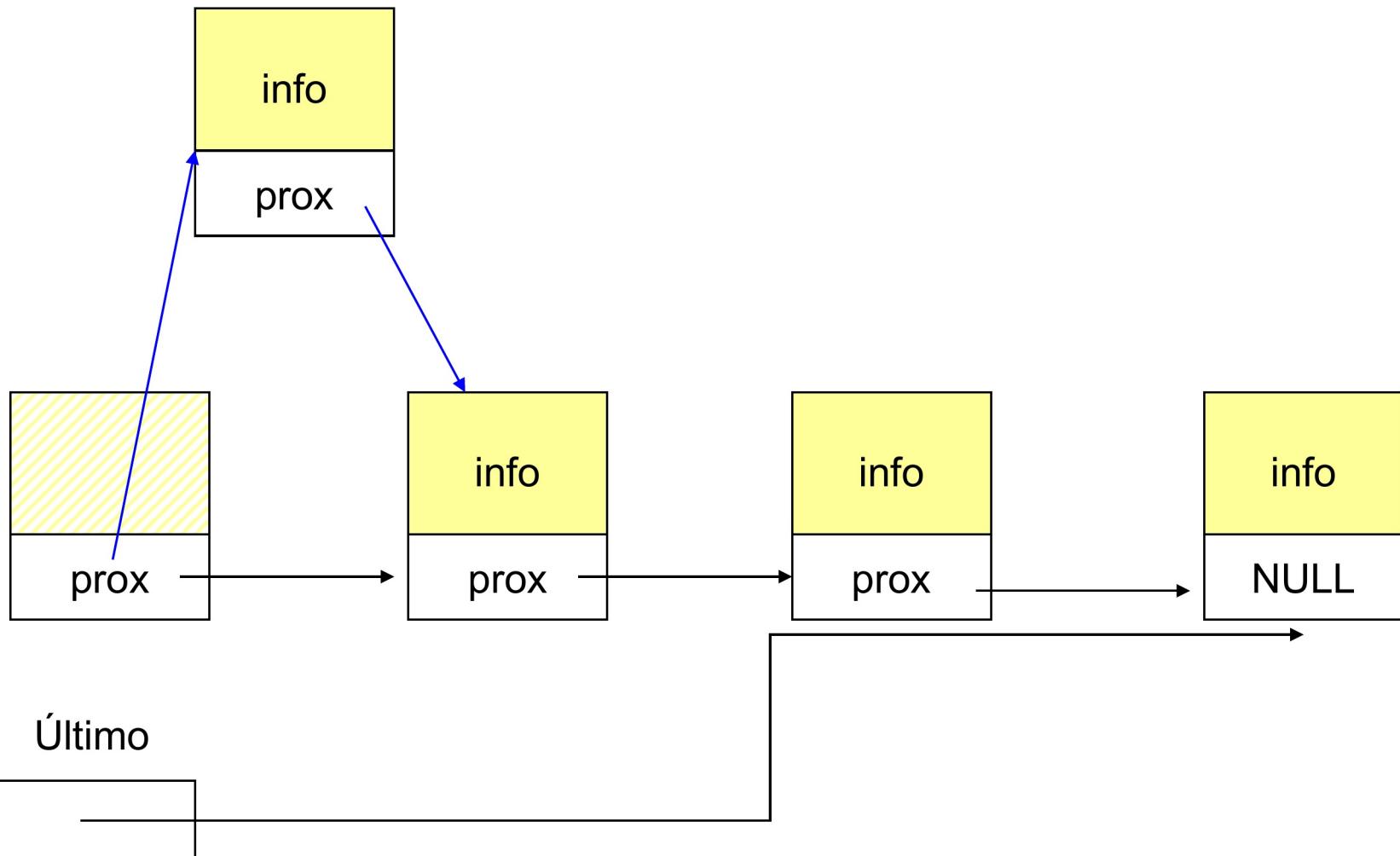


- +L Elm

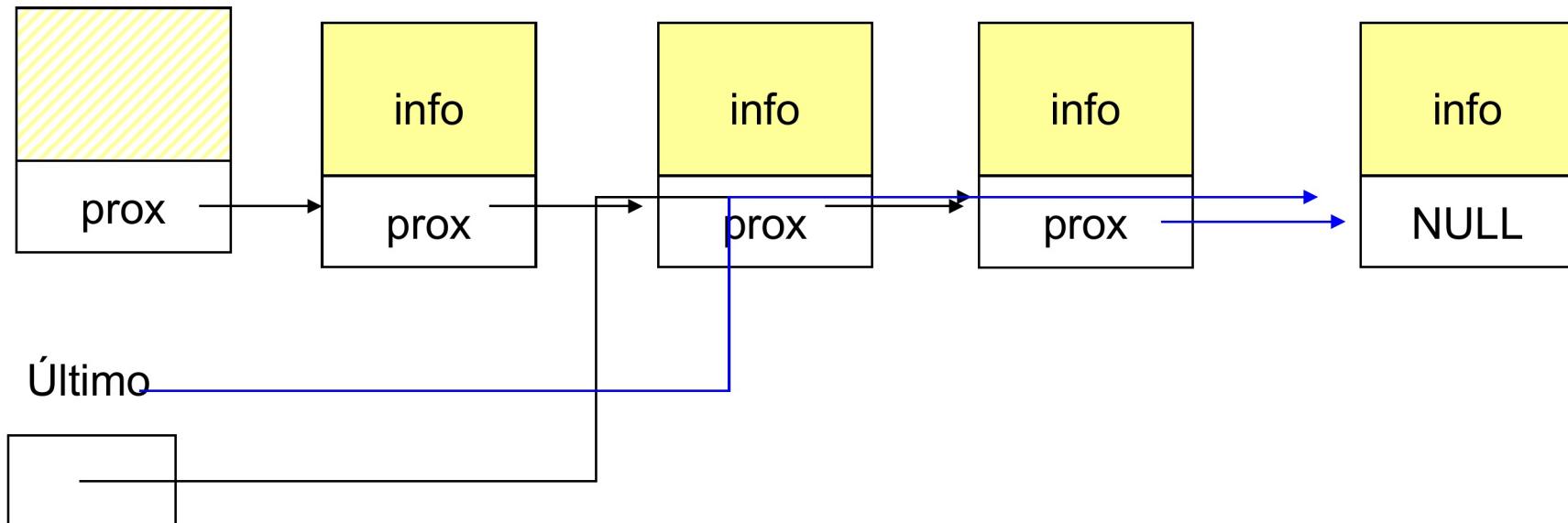


• inicio OK!
• meio OK!
• Fim OK!

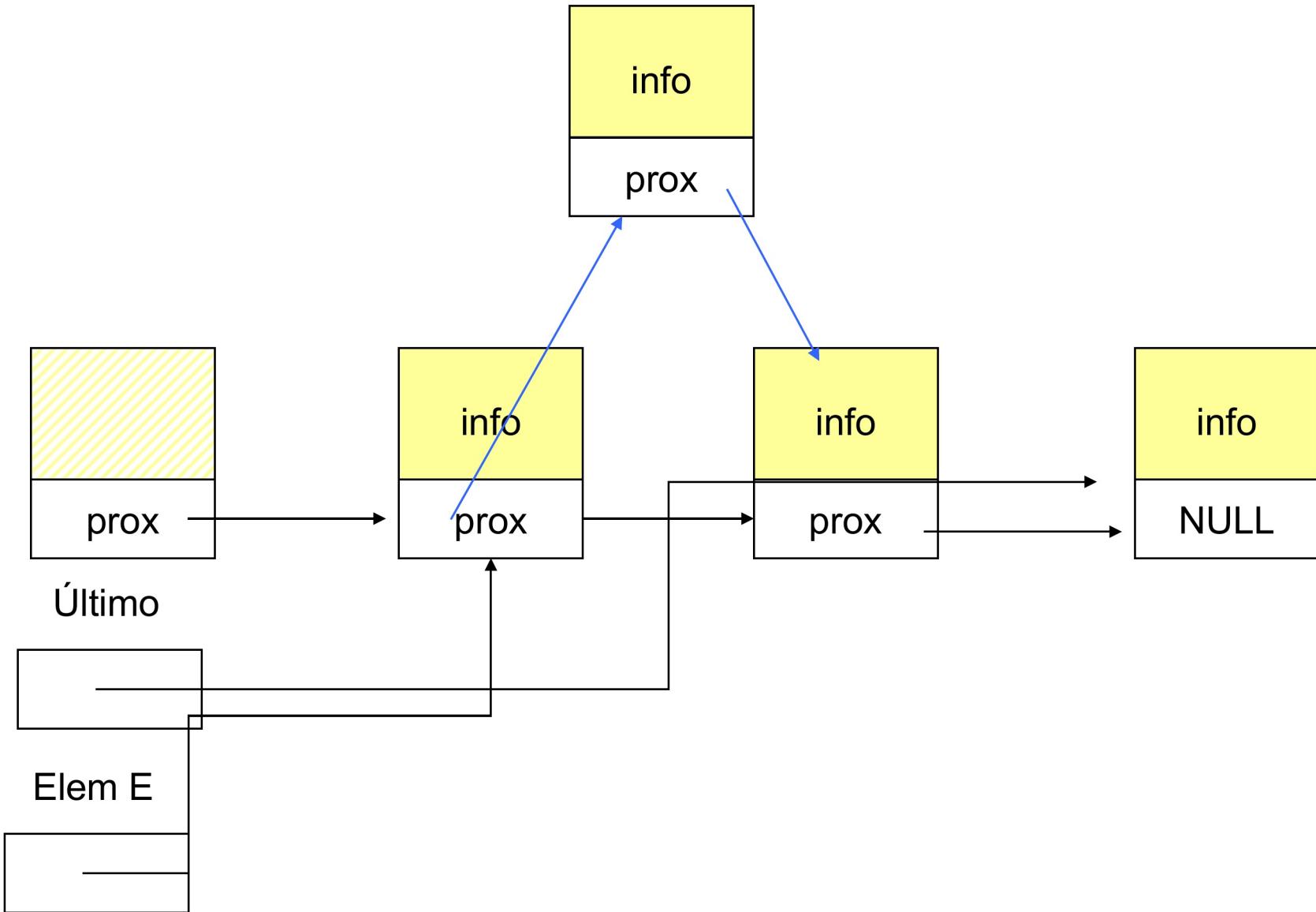
Inserção na Primeira Posição



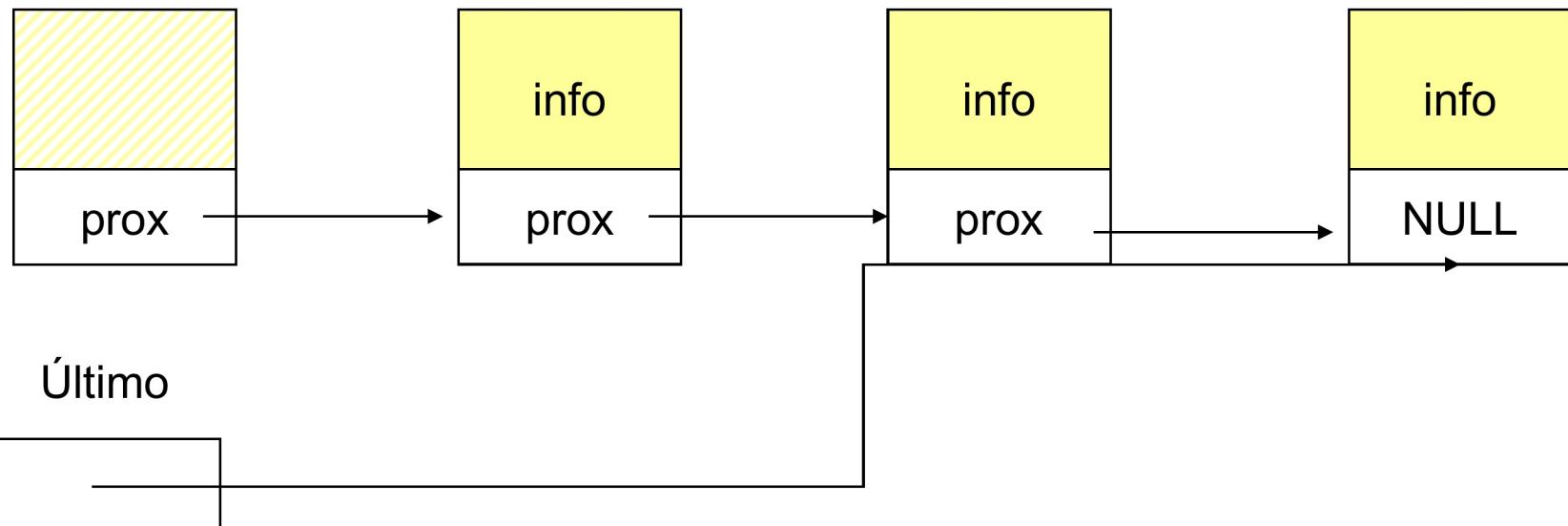
Inserção na Última Posição



Inserção Após o Elemento E

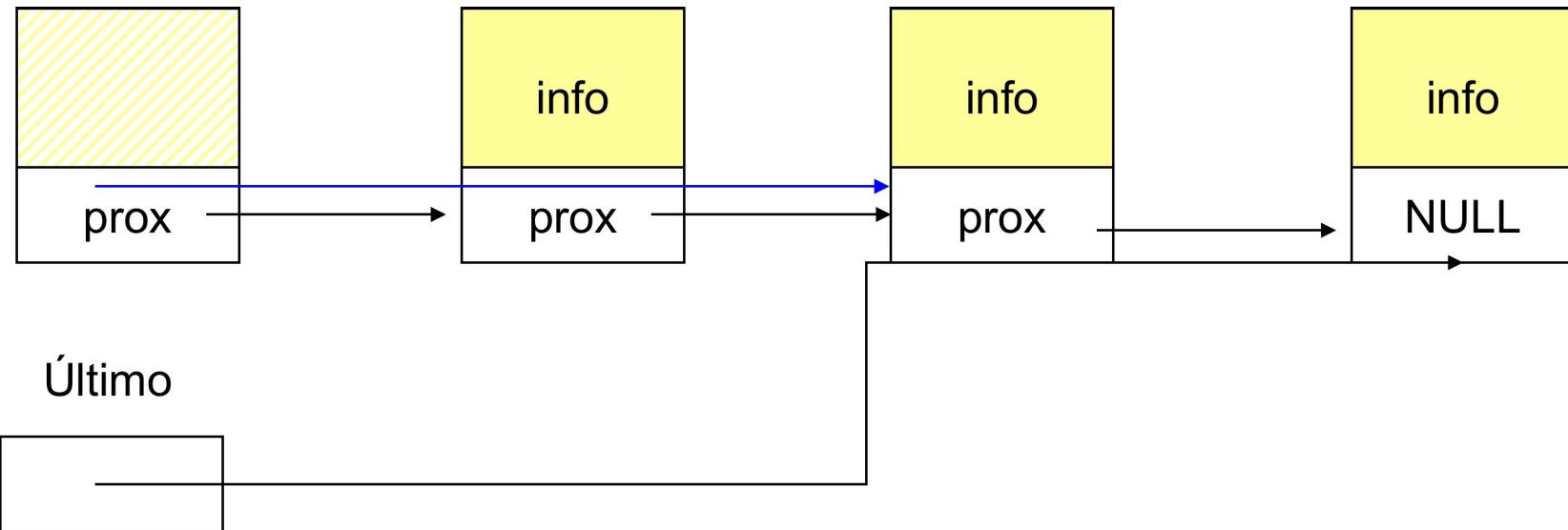


Retirada de Elementos na Lista

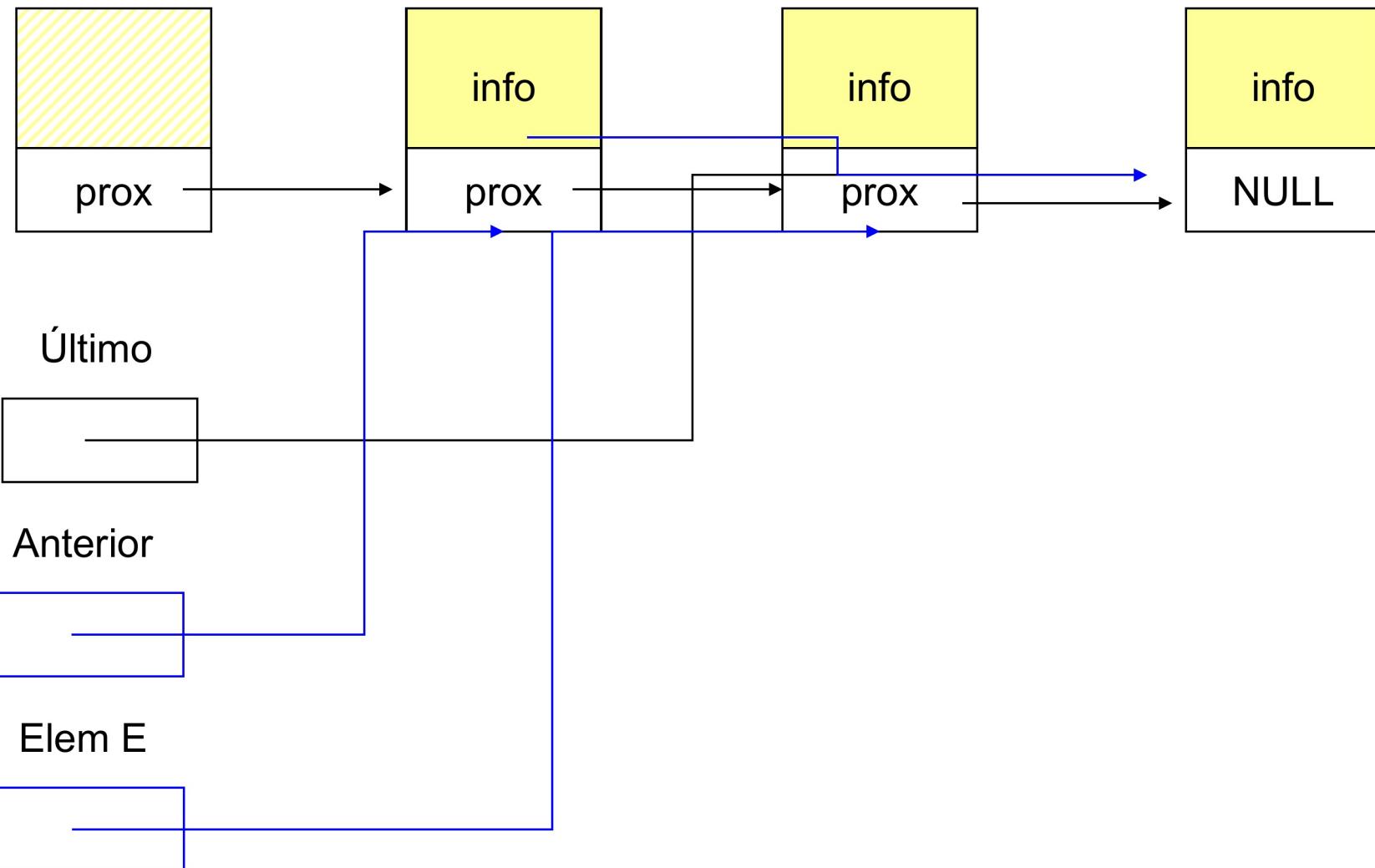


- 3 opções de posição de onde pode retirar:
 - 1^a. posição
 - última posição
 - Um elemento qualquer E

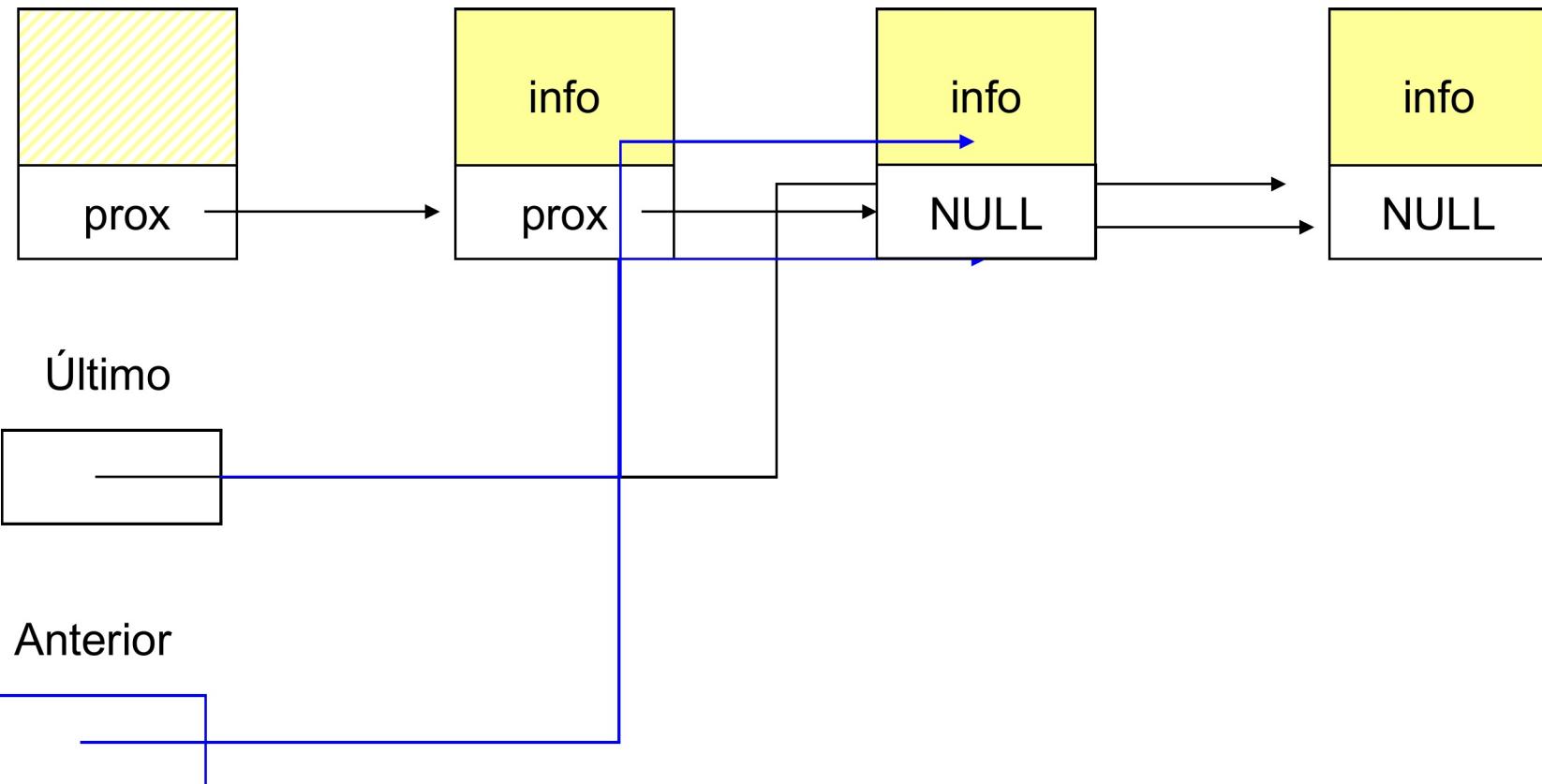
Retirada do Elemento na Primeira Posição da Lista



Retirada do Elemento E da Lista



Retirada do Último Elemento da Lista



Lista Encadeada Dinâmica

```
public class Lista {  
    private static class Celula {  
        Object item; Celula prox;  
    }  
    private Celula primeiro, ultimo, pos;  
  
    public Lista () { // Cria uma Lista vazia  
        primeiro = new Celula();  
        pos = primeiro;  
        ultimo = primeiro;  
        primeiro.prox = null;  
    }
```

Lista Encadeada Dinâmica

```
public void insere (Object x) {  
    ultimo.prox = new Celula();  
    ultimo = ultimo.prox;  
    ultimo.item = x;  
    ultimo.prox = null;  
}  
  
public boolean vazia () {  
    return (primeiro == ultimo);  
}  
  
public void imprime () {  
    Celula aux = primeiro.prox;  
    while (aux != null) {  
        System.out.println (aux.item.toString());  
        aux = aux.prox ;  
    }  
}
```

Lista Encadeada Dinâmica

```
public Object retira(Object item) throws Exception {
    if (item == null)
        throw new IllegalArgumentException("Erro: Valor invalido");
    if (vazia())
        throw new Exception("Erro: Lista vazia");
    Celula ant = primeiro, aux = primeiro.prox;
    while (aux != null && !aux.item.equals(item)) {
        ant = aux; aux = aux.prox; }
    if (aux == null)
        return null; // não encontrou o elemento a ser retirado
    else {
        ant.prox = aux.prox;
        if (aux.prox == null)
            ultimo = ant; // retirando ultimo elemento da lista
        return aux.item;
    }
}
```

Operações sobre Lista usando Referências

- 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.

Exercícios

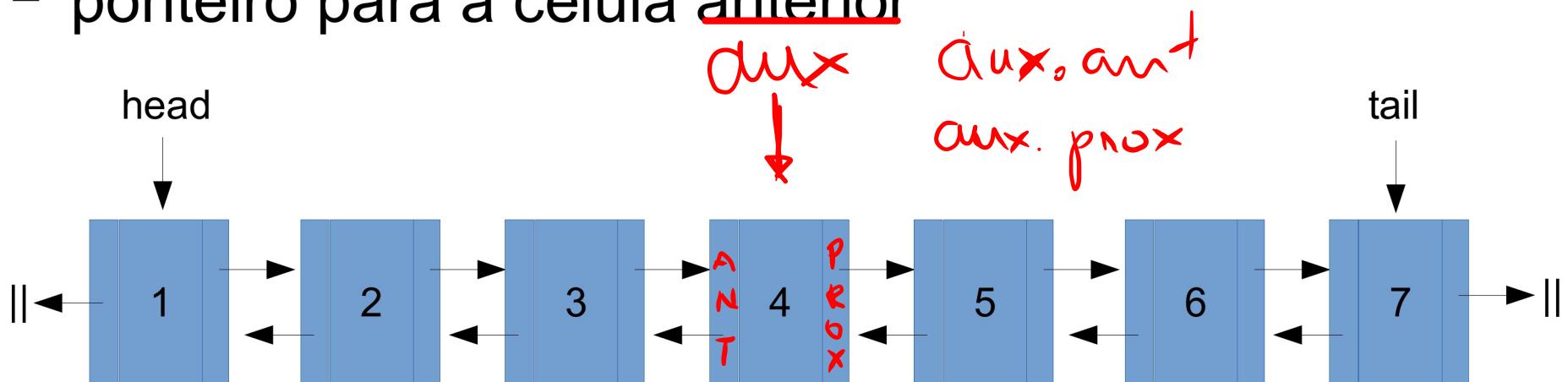
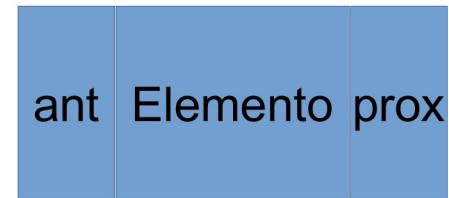
- O que precisa ser feito para criar um novo elemento para a lista?
- Escreva uma função que receba uma lista como parâmetro e retira o seu primeiro elemento, apagando-o.
- Escreva uma função que receba uma lista e um ponteiro para uma célula como parâmetros e insira a célula na primeira posição da lista.
- Imagine uma lista duplamente encadeada

Lista Duplamente Encadeadas

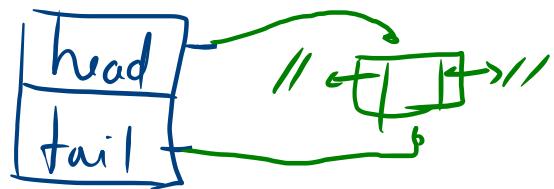
- Listas simplesmente encadeadas são ineficientes para realizar certas operações. Por exemplo:
 - Remover o último elemento ou um elemento intermediário
 - É preciso percorrer toda lista para encontrar o elemento anterior
- Muitas aplicações não demandam tais operações. P. ex:
 - Pilhas e filas podem ser implementadas eficientemente apenas removendo elementos do início de uma lista
- E quando tais operações são necessárias ?
 - Podemos utilizar uma **lista duplamente encadeada**

Lista Duplamente Encadeadas

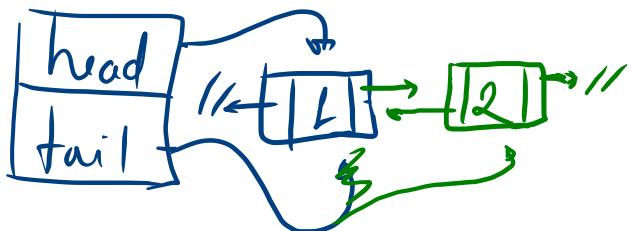
- Cada elemento da lista possui informações de quem é seu sucessor e antecessor
- Cada célula armazena:
 - elemento (registro, objeto, ...)
 - ponteiro para a próxima célula
 - ponteiro para a célula anterior



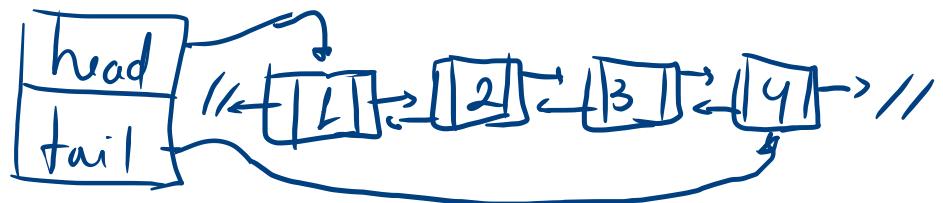
LDEP sem Sentinelas
Fazendo



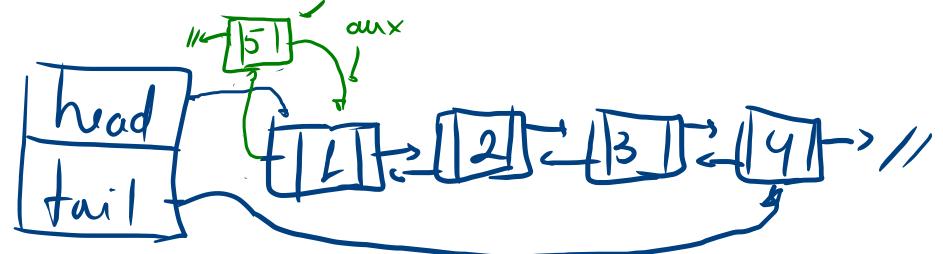
* L elementos



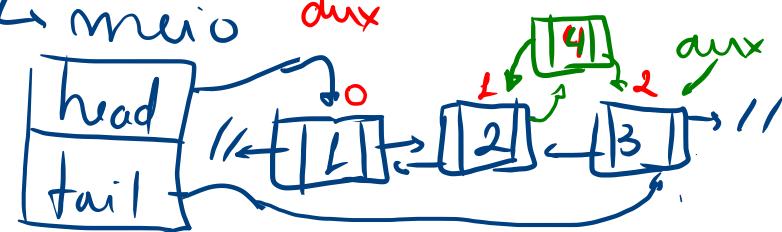
* + L elementos



↳ inicio



↳ meio

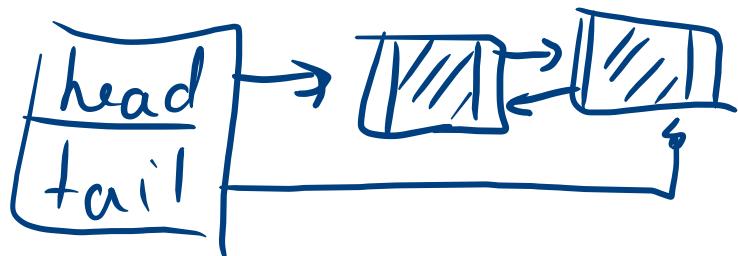


↳ Fim



LDEB com Sentinel

* vazia



* +L elementos

