Universidade do Estado do Amazonas
Escola Superior de Tecnologia - EST
Núcleo de Computação

Algoritmos e Estruturas de Dados I

Listas com implementação dinâmica

Prof. Flávio José M. Coelho

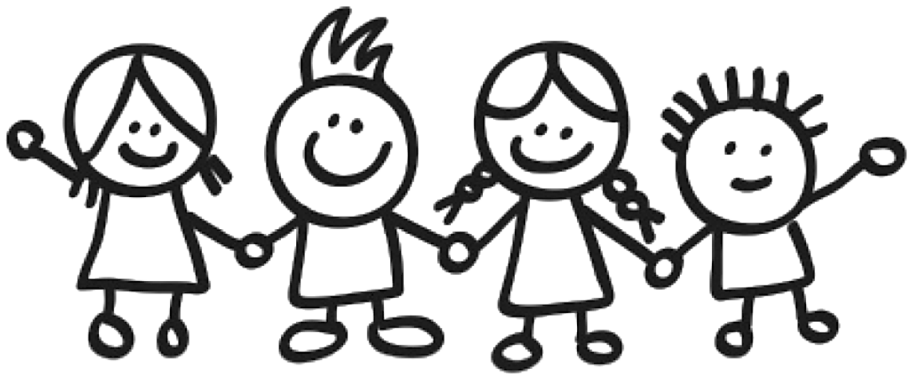
fcoelho@uea.edu.br

Objetivos

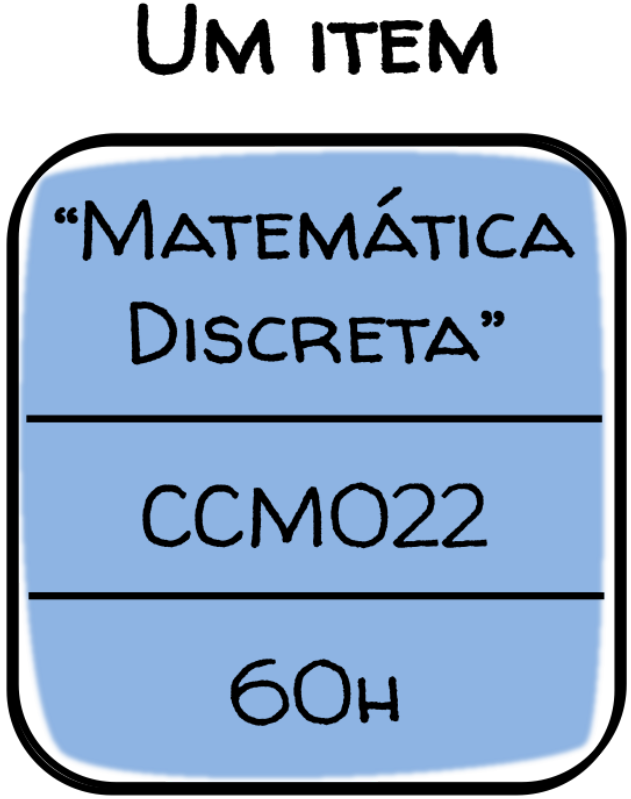
Entender o que são...

1. Listas ligadas.
2. Listas simplesmente encadeadas (**LSE**).
3. Listas duplamente encadeadas (**LDE**).
4. Entender a **implementação** de uma **LSE**.

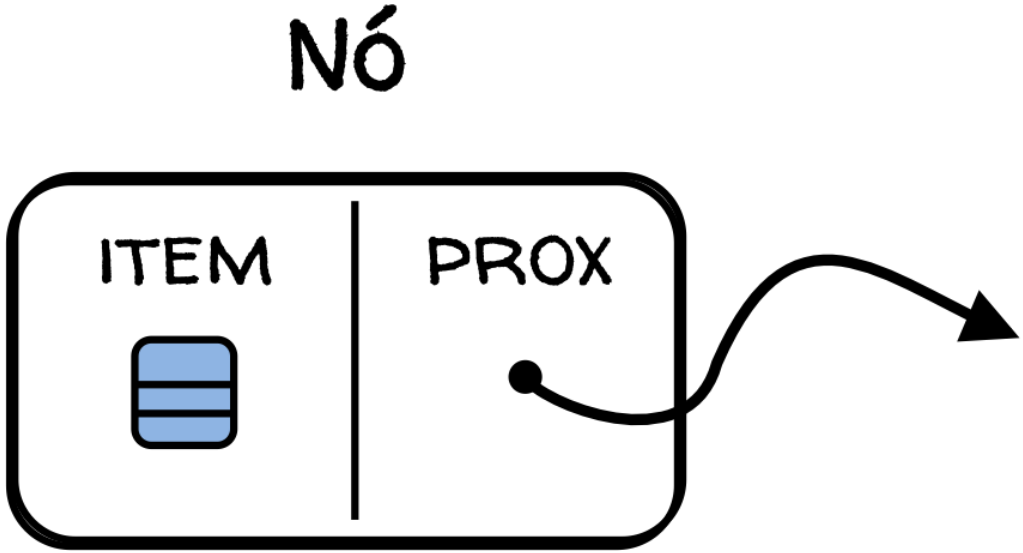
Listas ligadas



Listas ligadas

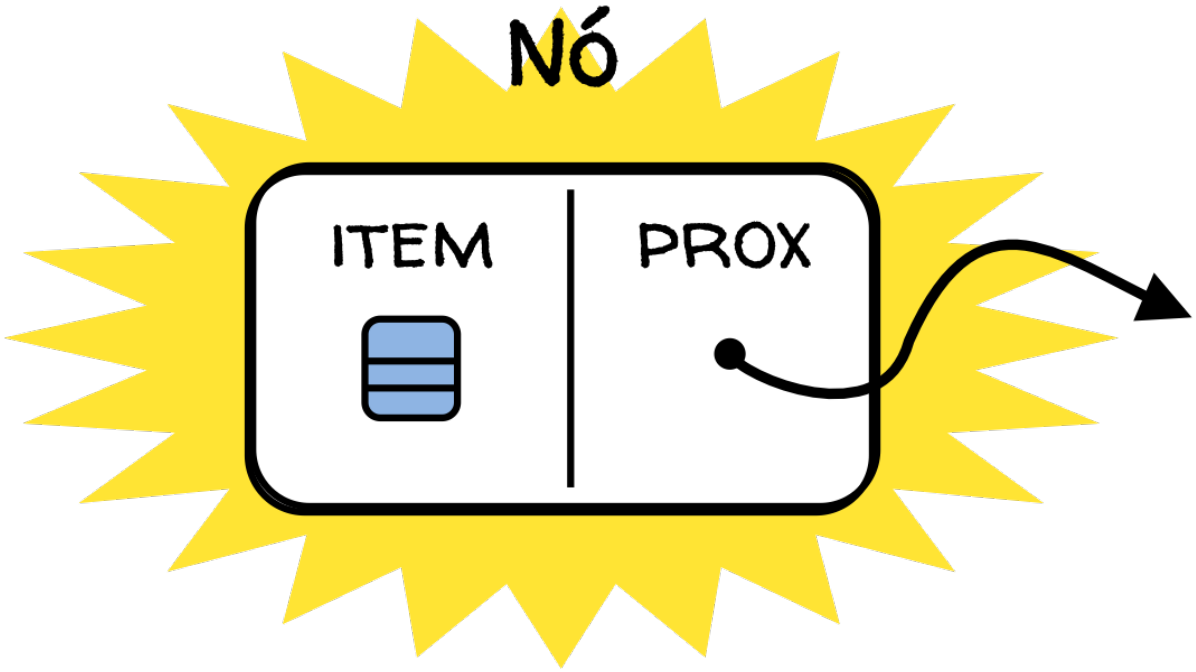


Listas ligadas



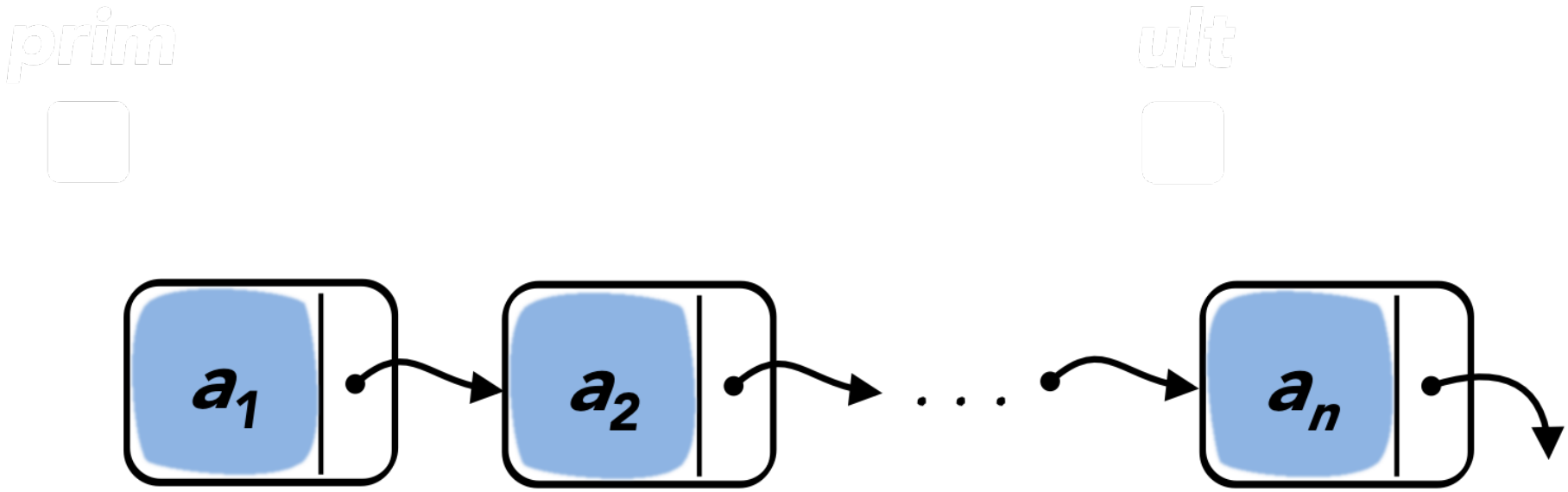
Um **nó** tem dois campos: o **item** e um **ponteiro para o próximo nó**.

Listas ligadas



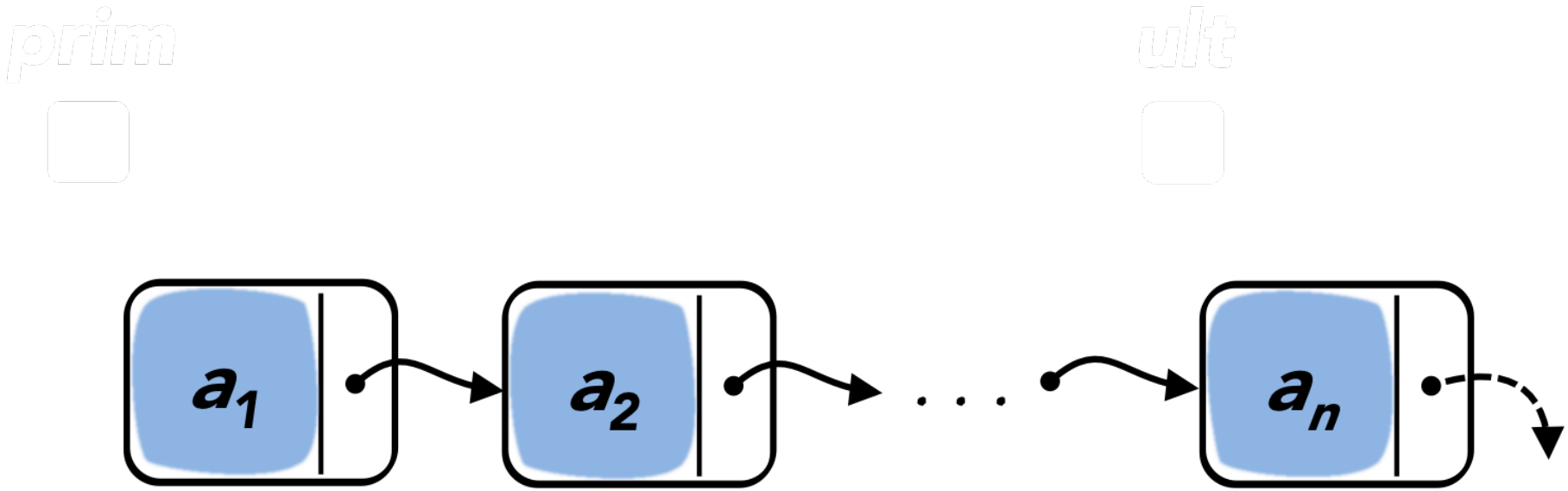
Cada nó é **alocado dinamicamente** de uma região de memória denominada **heap**.

Listas ligadas



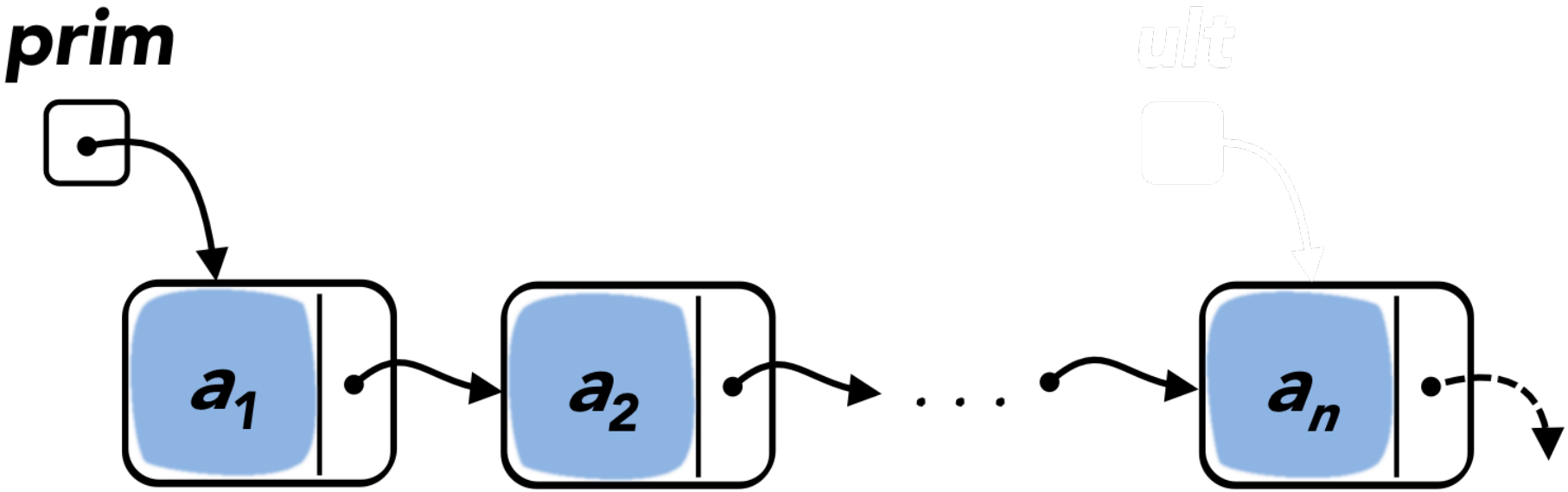
Cada nó liga-se a um próximo nó formando uma **lista ligada**.

Listas ligadas



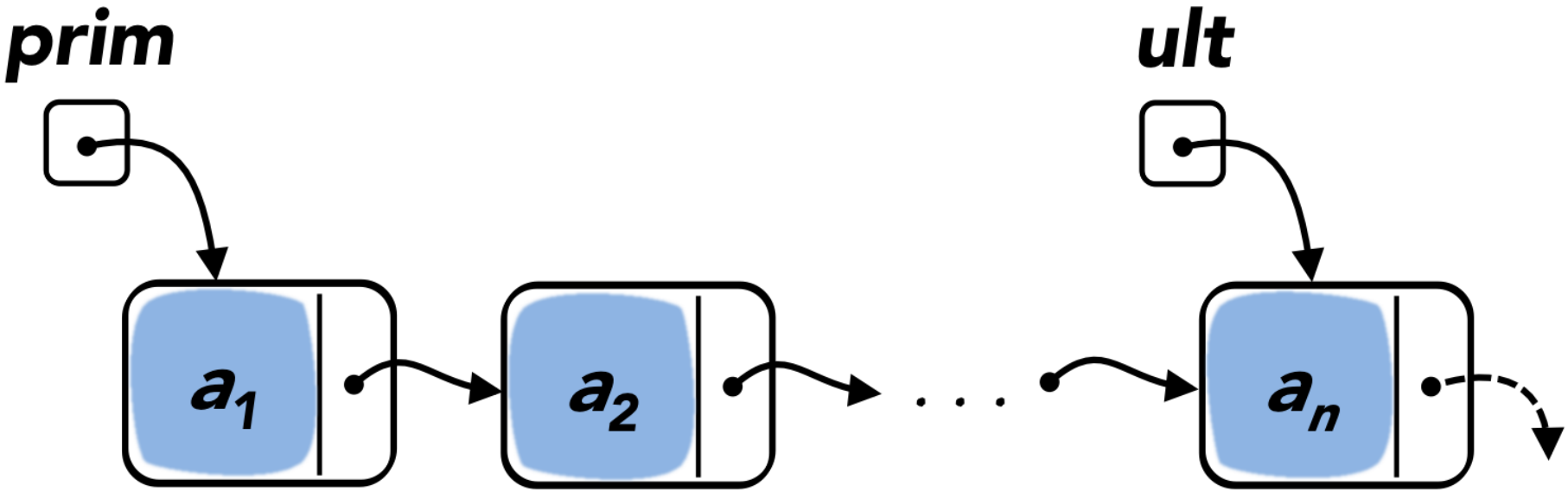
O ponteiro do último nó é NIL pois não há próximo nó.

Listas ligadas



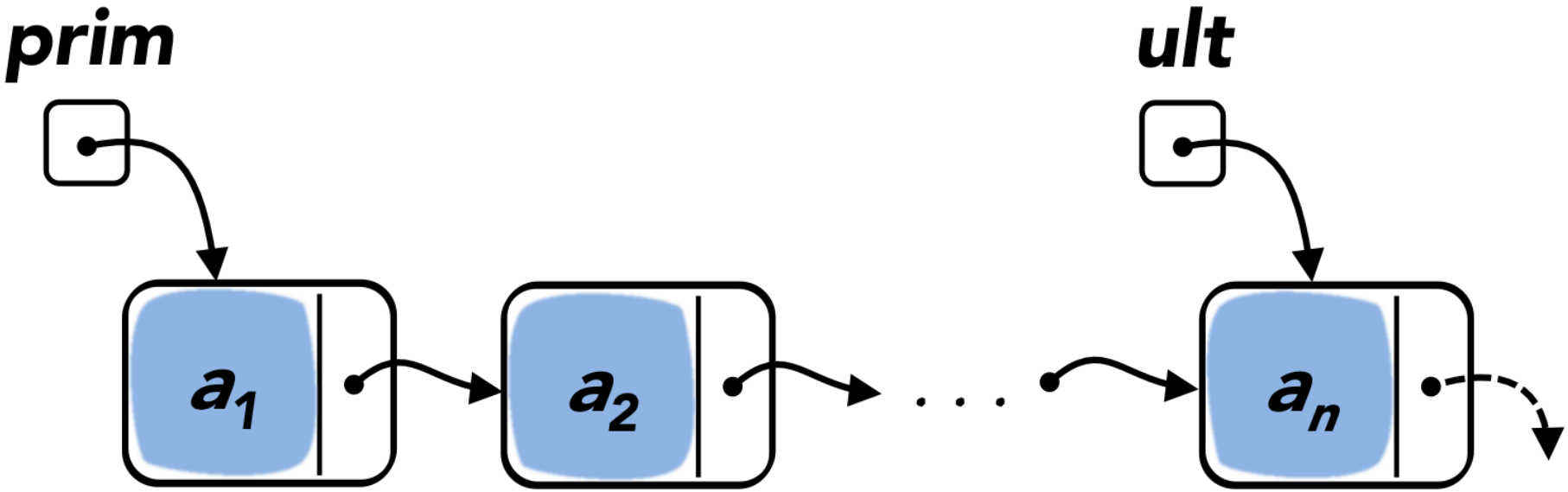
Um ponteiro (**prim**) aponta para o primeiro nó da lista, e dá acesso ao **início da lista**.

Listas ligadas



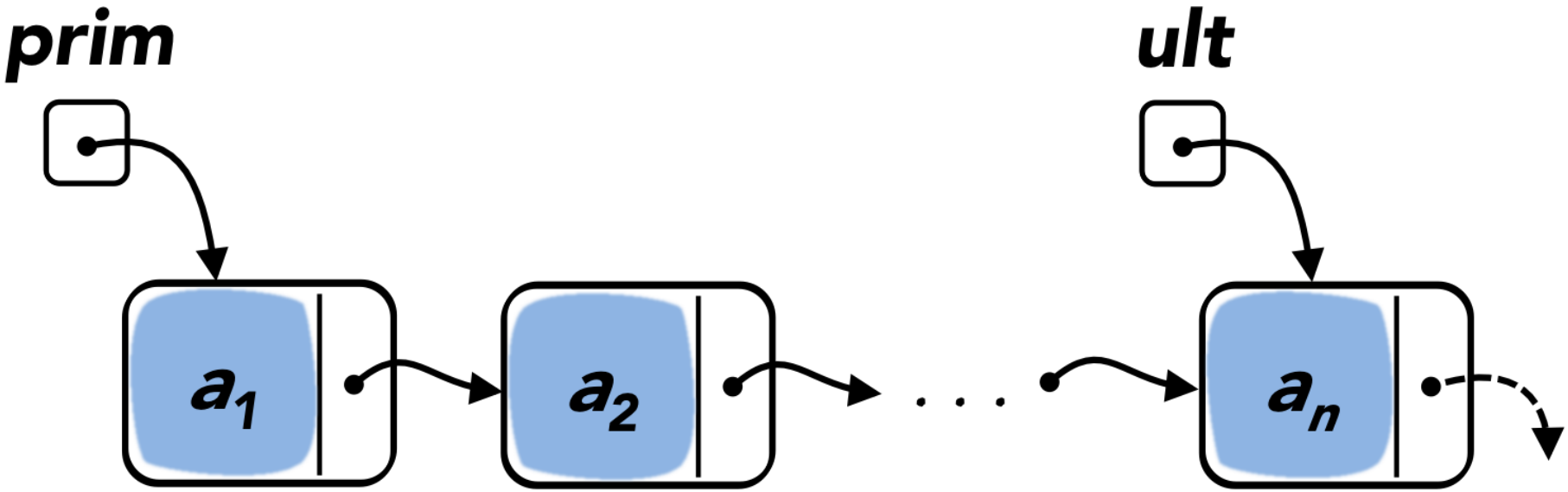
Outro ponteiro (**ult**) aponta para o último nó, e ajuda operações no final da lista.

Listas ligadas



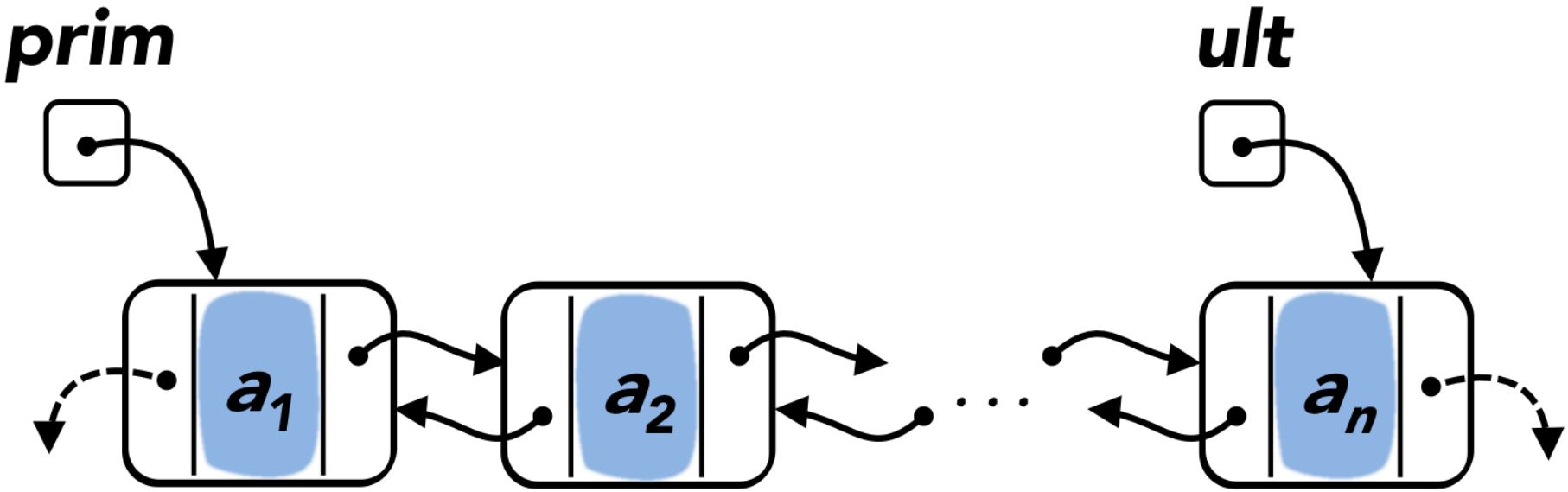
A lista é **simplesmente encadeada** (**LSE**), se cada nó se liga somente ao próximo nó.

Listas ligadas



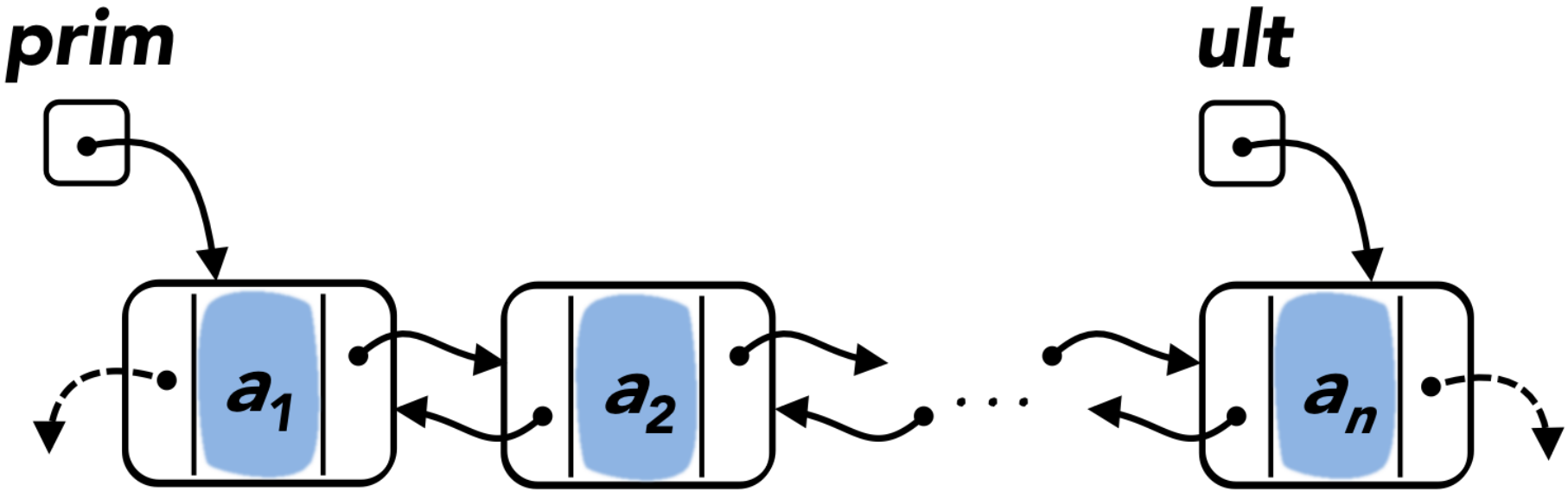
Em uma **LSE** somente é possível navegar pelos itens no sentido **início-fim** da lista.

Listas ligadas



A lista é **duplamente encadeada** (**LDE**), se cada nó se liga ao próximo e ao nó anterior.

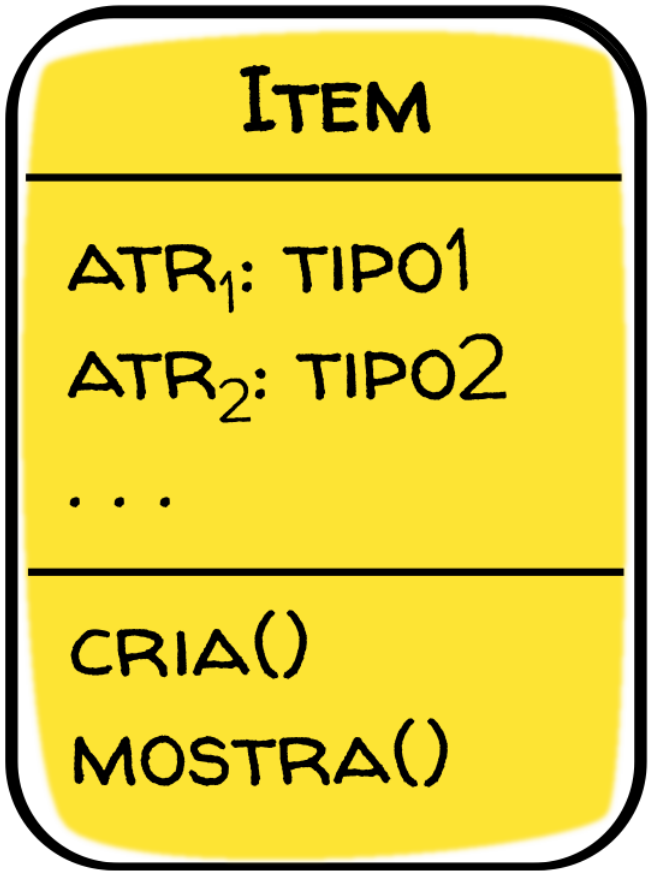
Listas ligadas



Uma **LDE** permite navegação nos sentidos **início-fim** e **fim-início**.

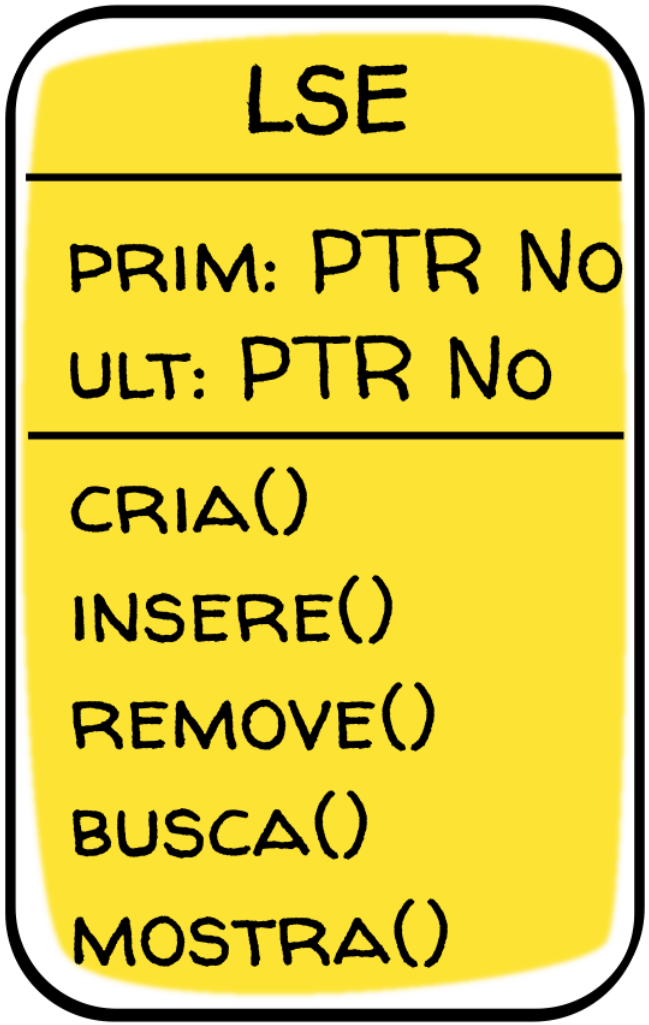
TAD Lista ligada

TAD Lista Ligada

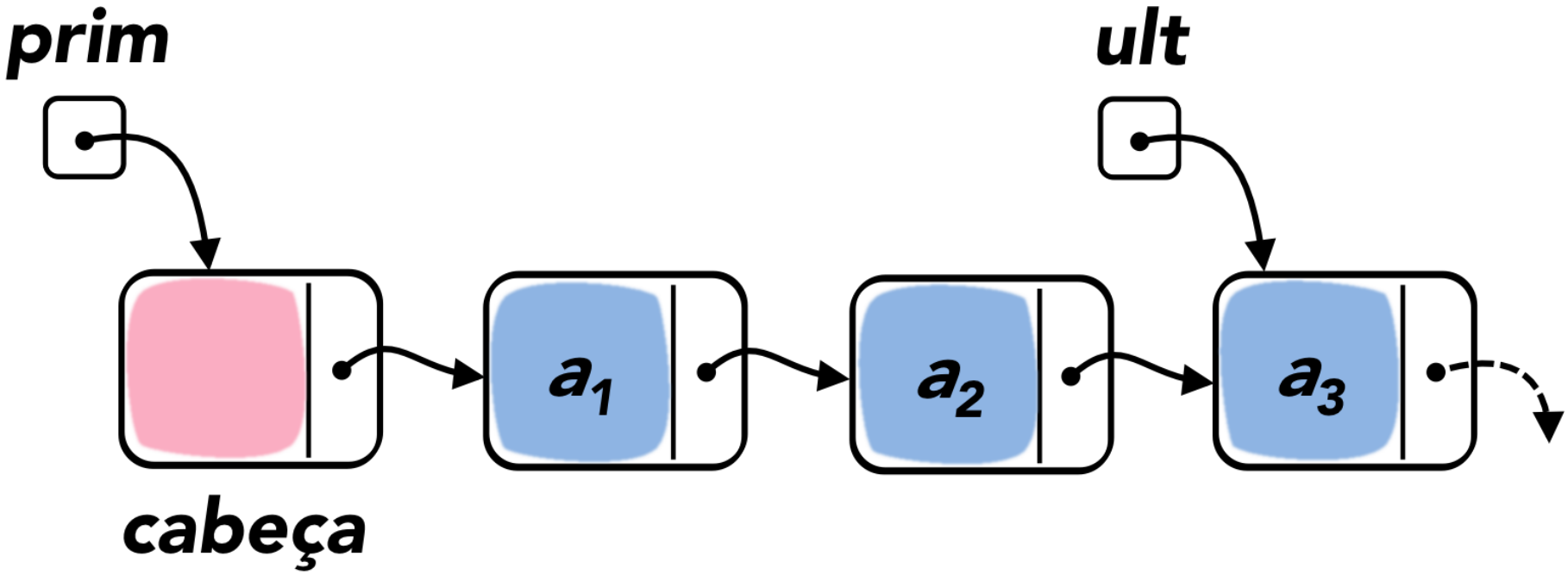


TAD Lista Ligada

TAD Lista Ligada



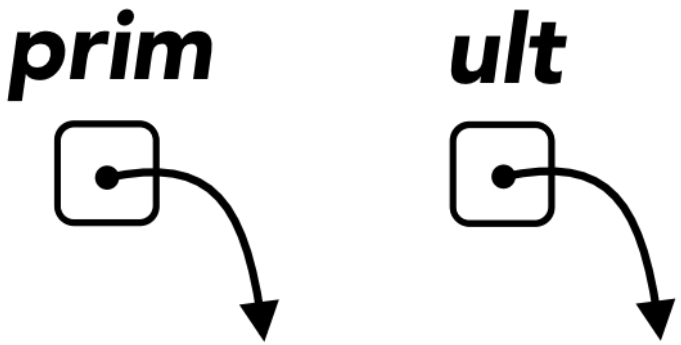
Operações



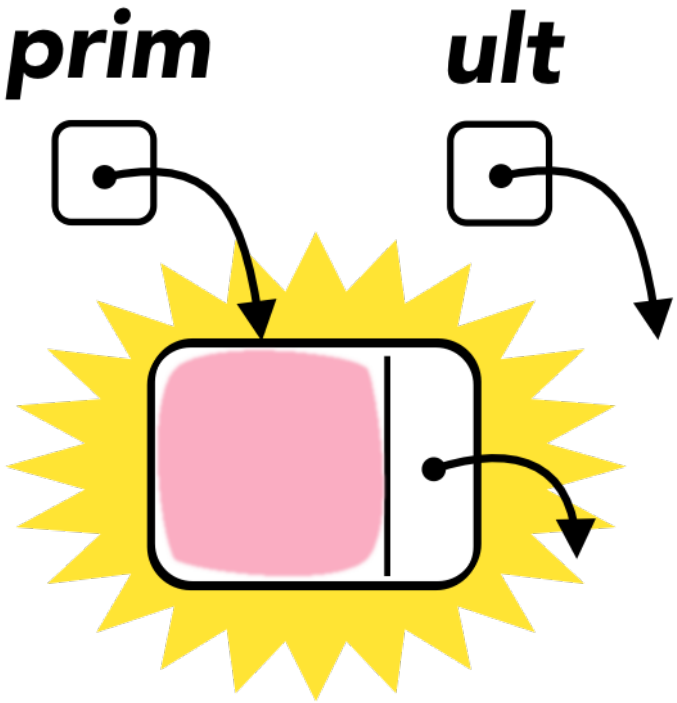
Um **nó-cabeça** (item vazio) antes do início da lista facilita inserções e remoções.

```
CRIA(L)
1   L.prim = aloque novo No
2   L.prim.prox = NIL
3   L.ult = L.prim
```

CRIA(L)

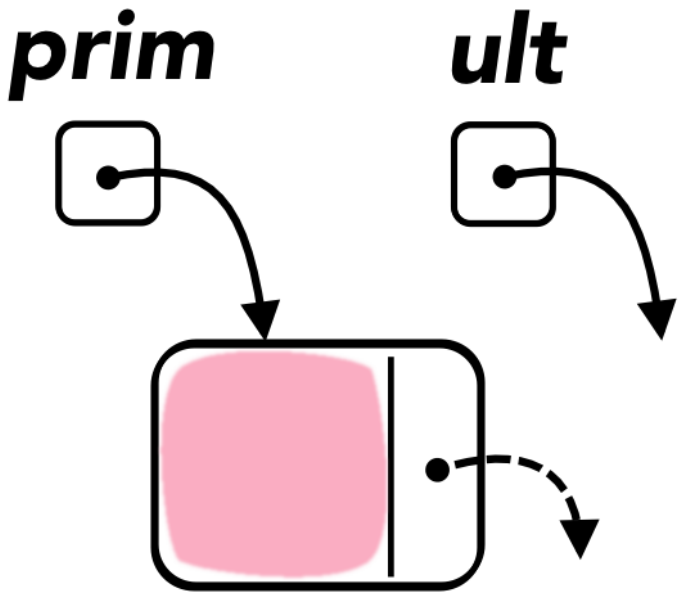


CRIA(L)



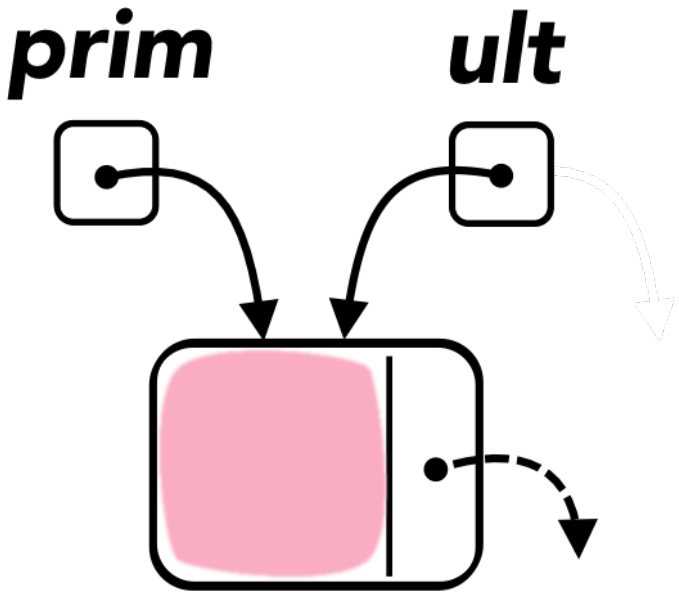
(1) L.prim = aloque novo **NO**

CRIA(L)



(2) L.prim.prox = NIL

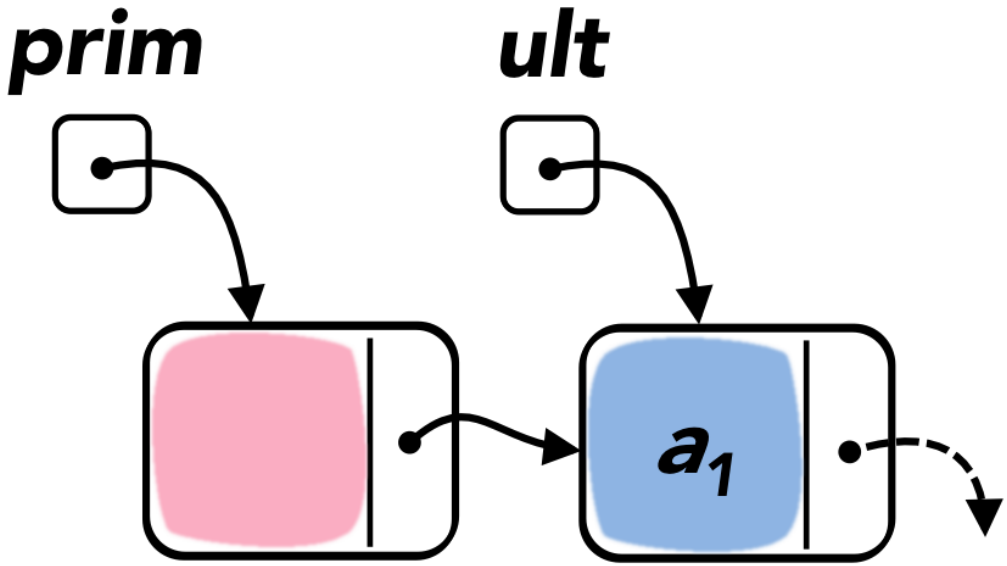
CRIA(L)



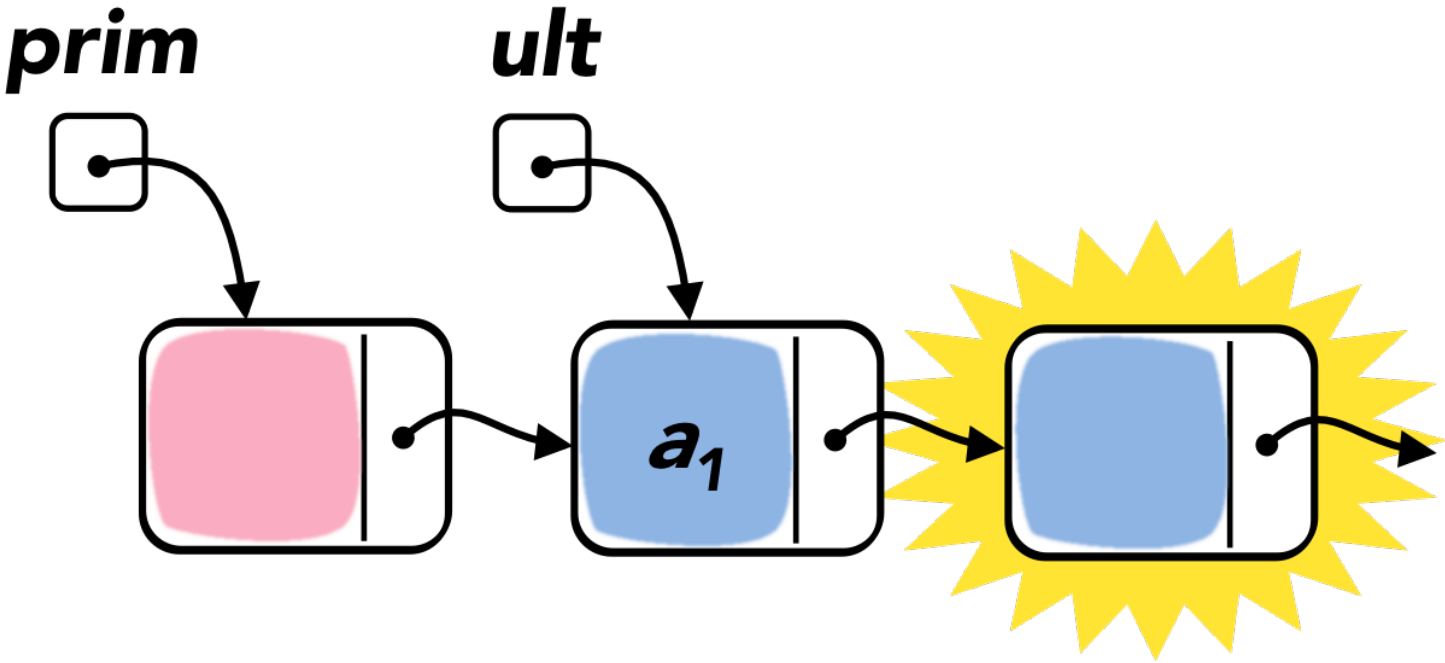
(3) L.ult = L.prim

```
INSERE(L, item)
1    L.ult.prox = aloque novo No
2    L.ult = L.ult.prox
3    L.ult.prox = NIL
4    L.ult.item = item
```

INSERE(L, item)

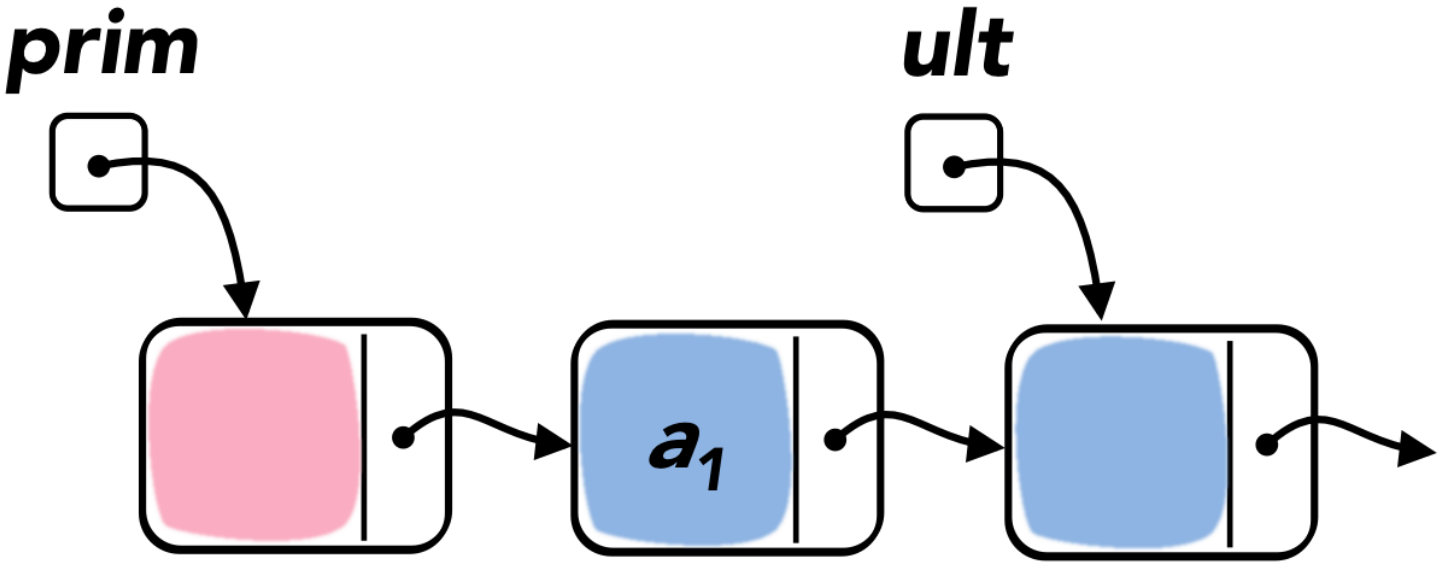


INSERE(L, item)



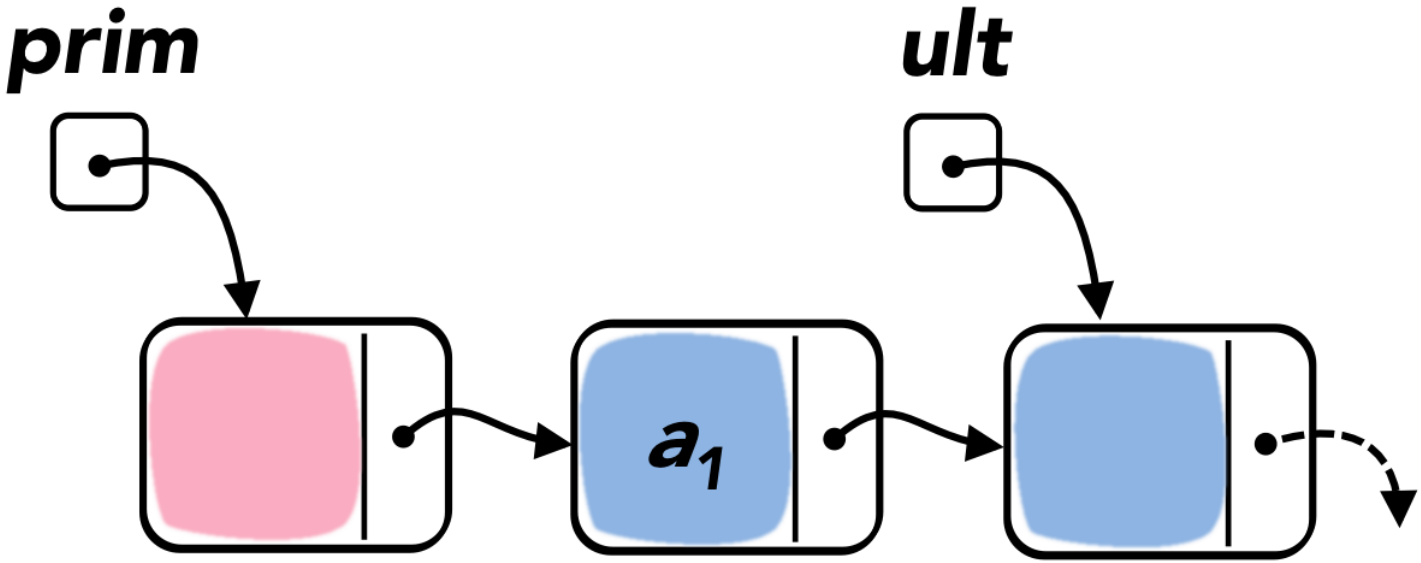
(1) L.ult.prox = aloque novo **No**.

INSERE(L, item)



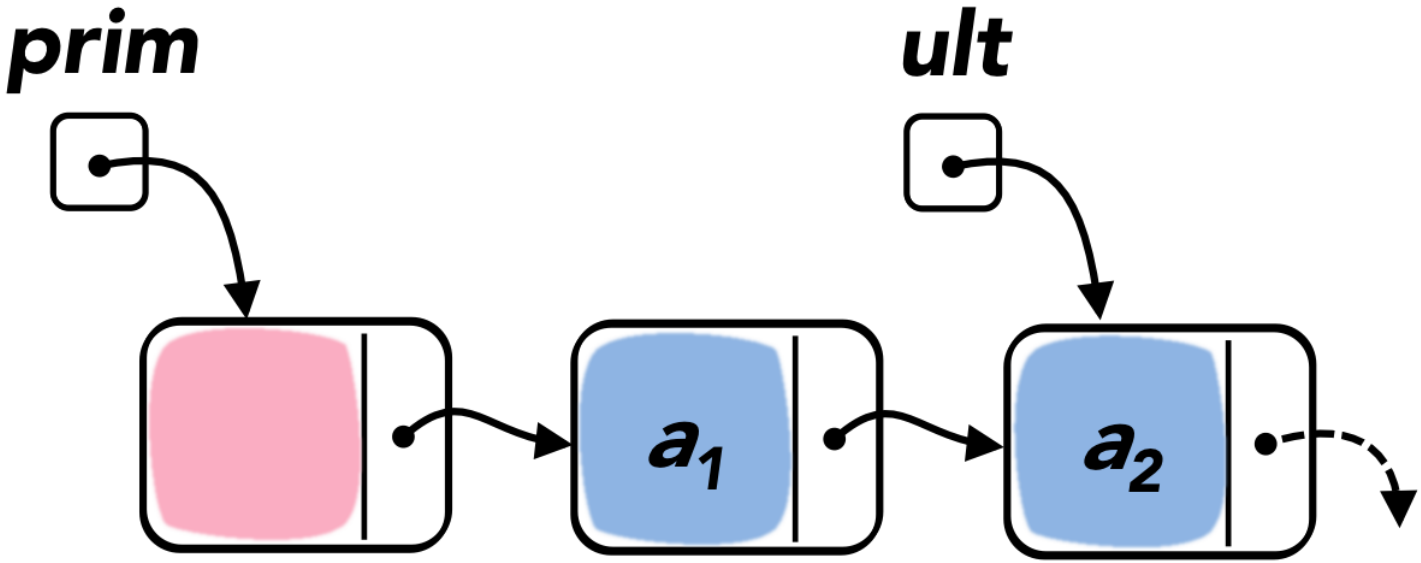
(2) L.ult = L.ult.prox

INSERE(L, item)



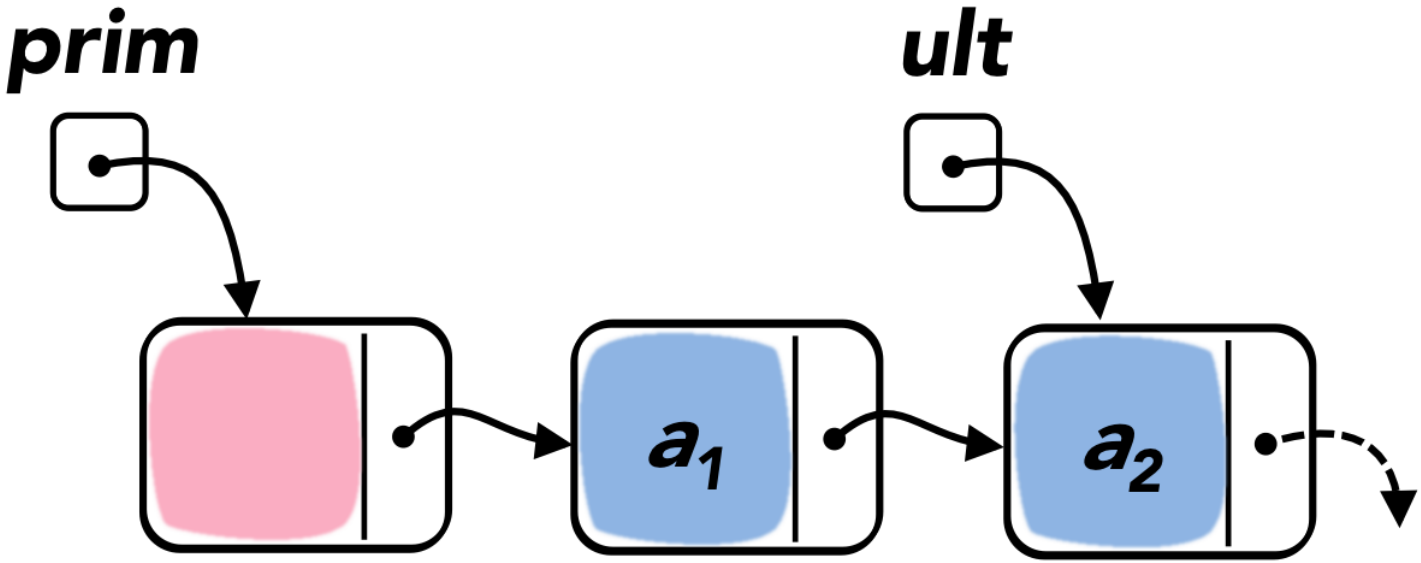
(3) L.ult.prox = NIL

INSERE(L, item)



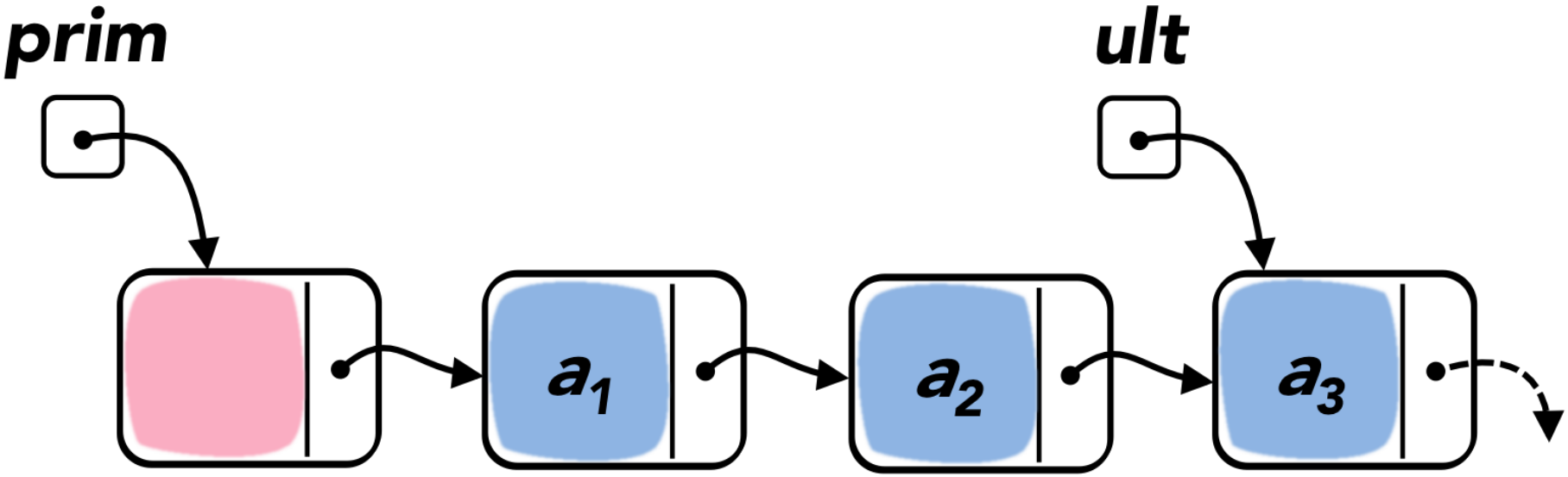
(4) L.ult.item = item

INSERE(L, item)

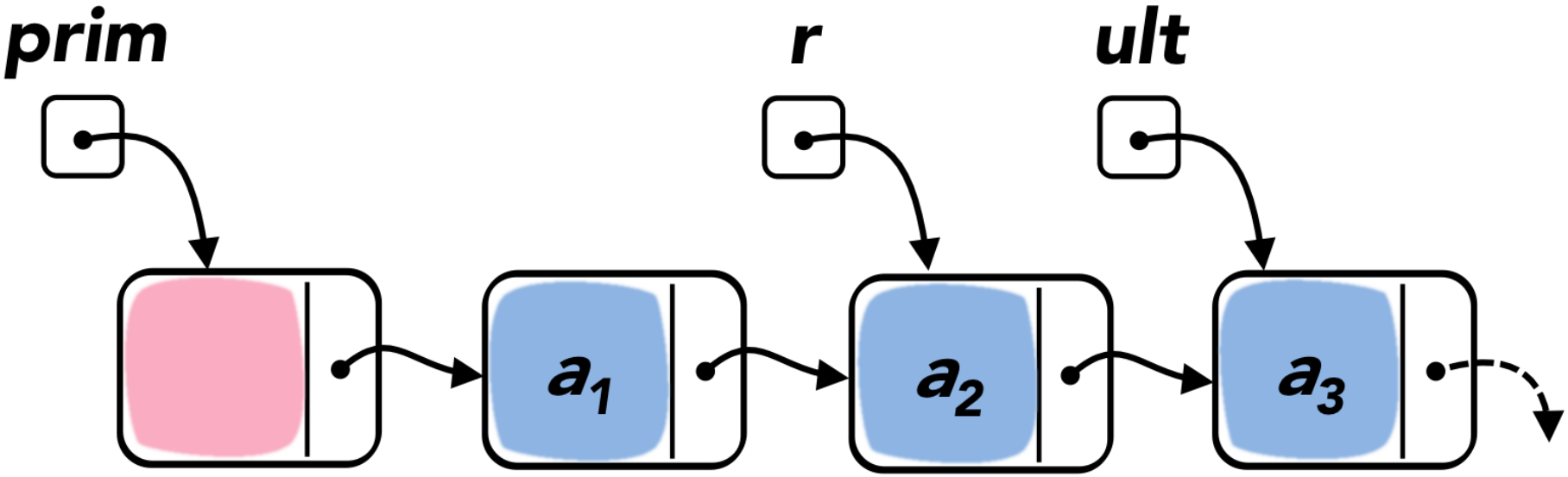


```
REMOVE(L, r, item)
1    se |L| == 0 ou r == NIL ou r ==prim
2        impossível remoção
3    senão
4        item = r.item
5        p = PREDECESSOR(L, r)
6        p.prox = r.prox
7        se p.prox == NIL  L.ult = p
8        desaloque r
```

REMOVE(L, r, item)

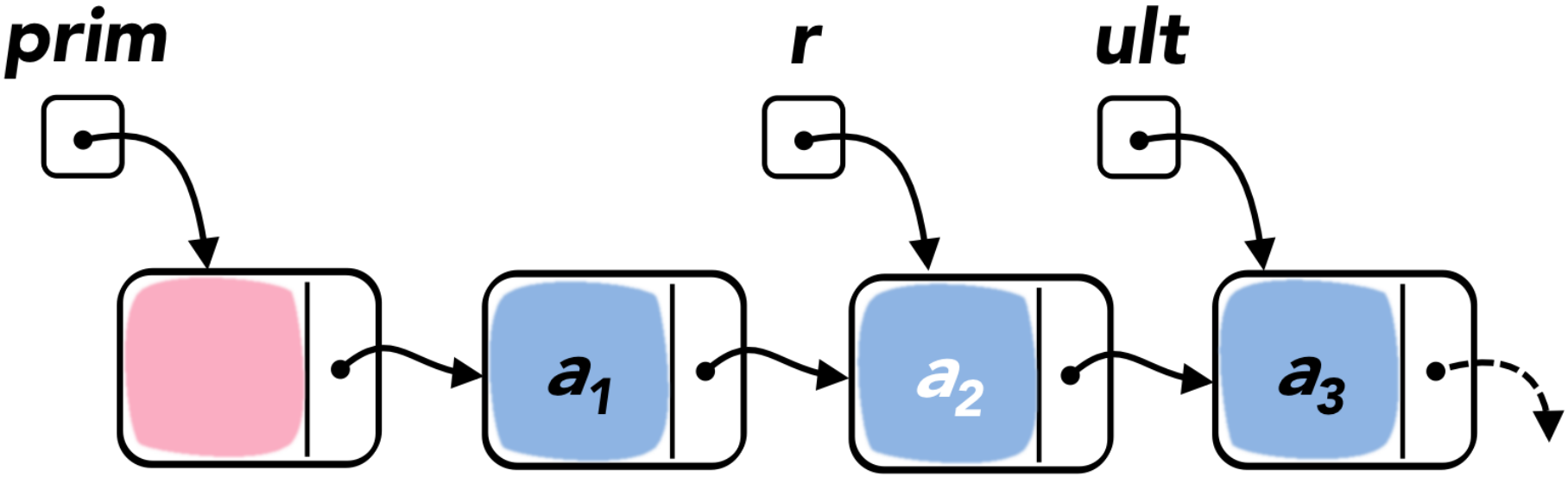


REMOVE(L, r, item)



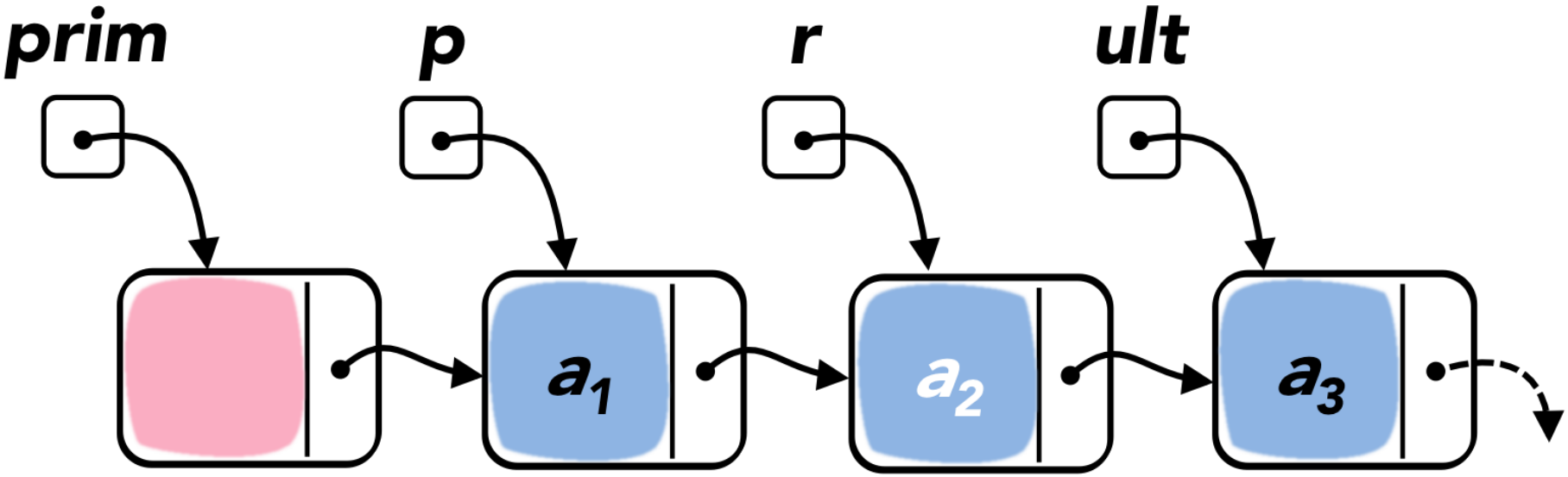
Parâmetro **r** aponta para o item a ser removido.

REMOVE(L, r, item)



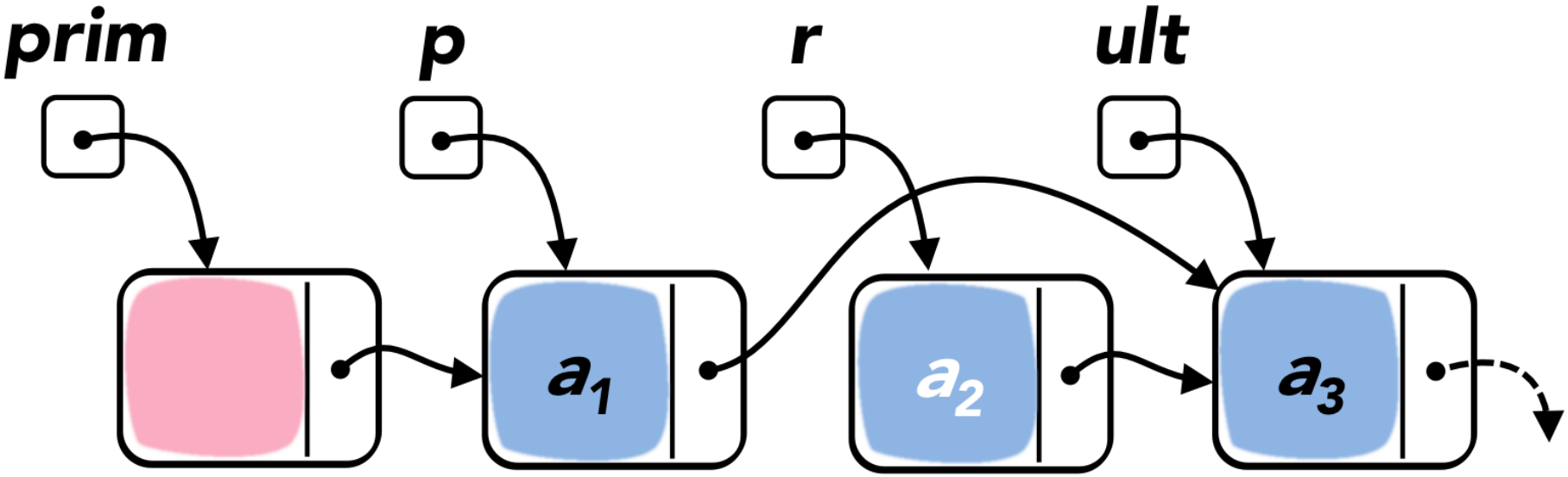
(4) item = r.item

REMOVE(L, r, item)



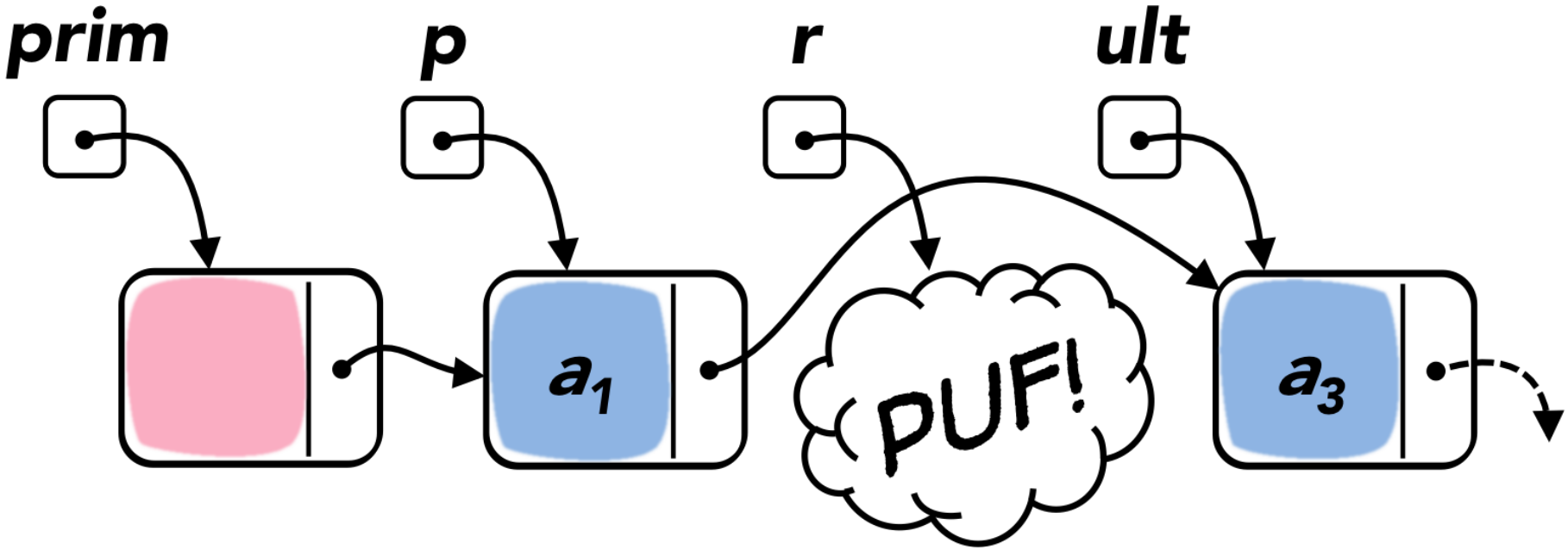
(5) p = PREDECESSOR(L, r)

REMOVE(L, r, item)



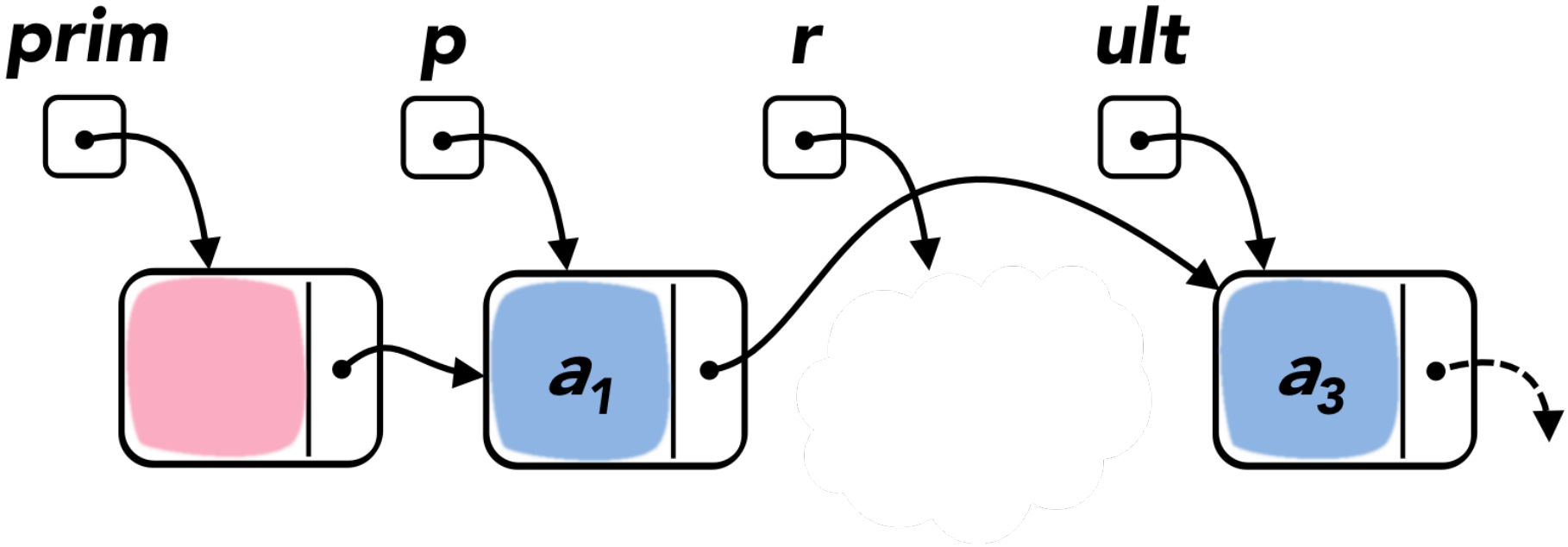
(6) p.prox = r.prox

REMOVE(L, r, item)



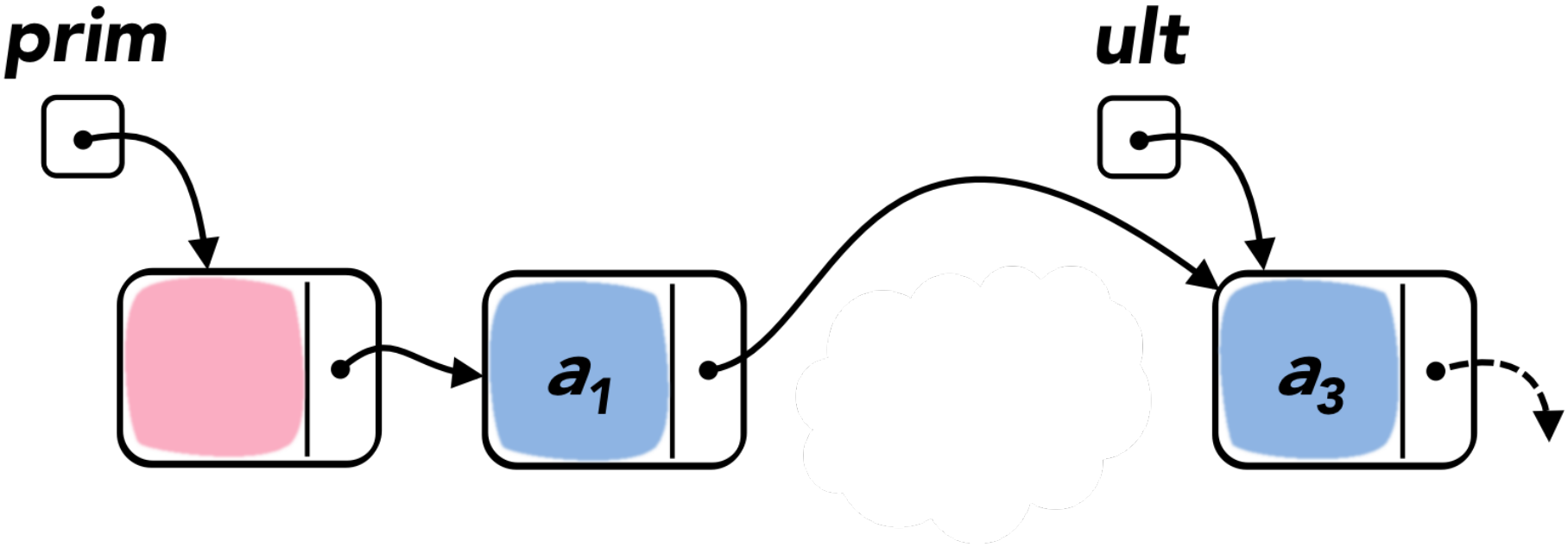
(8) desaloque q

REMOVE(L, r, item)



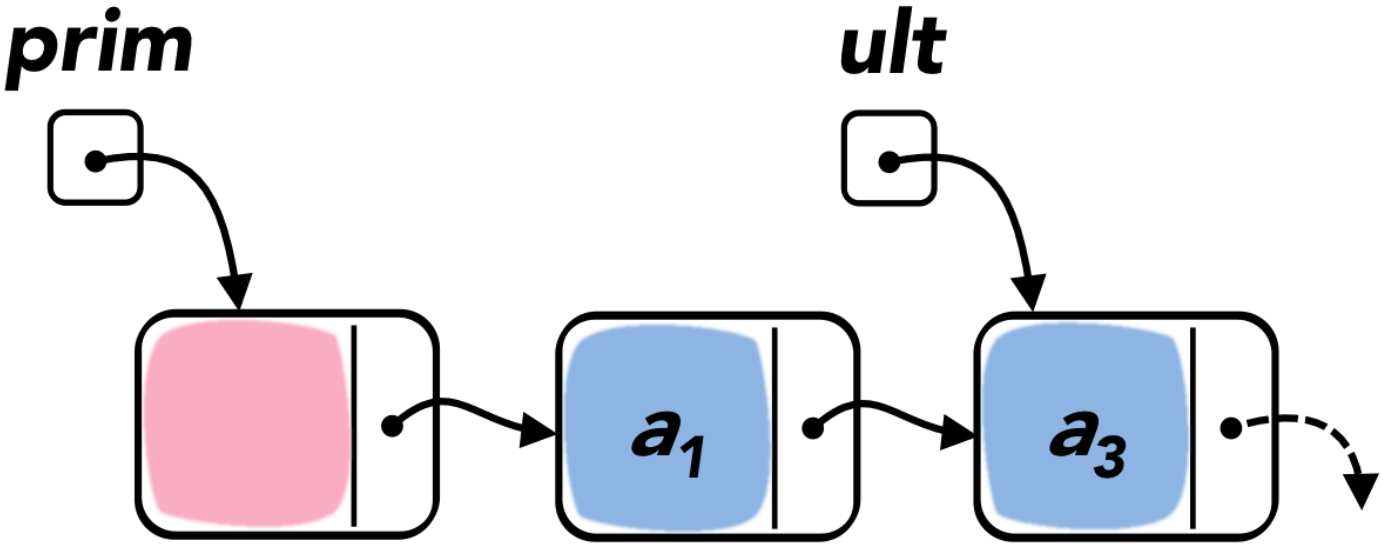
(8) desaloque q

REMOVE(L, r, item)



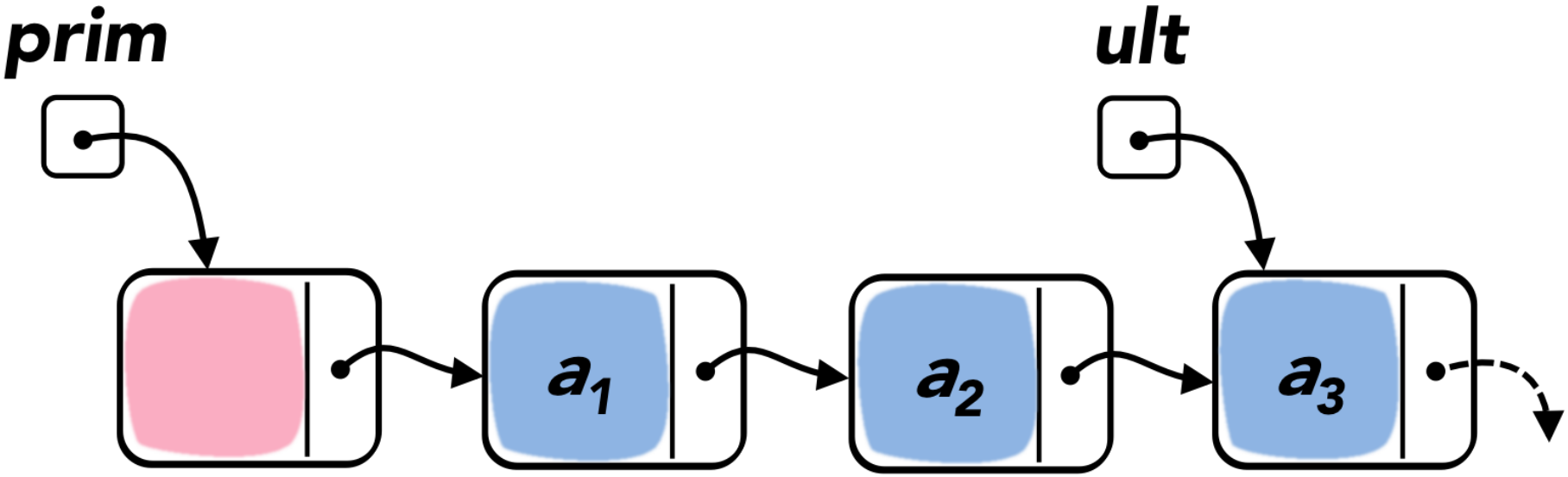
Com o término da operação, **p** e **r** são destruídos.

REMOVE(L, r, item)

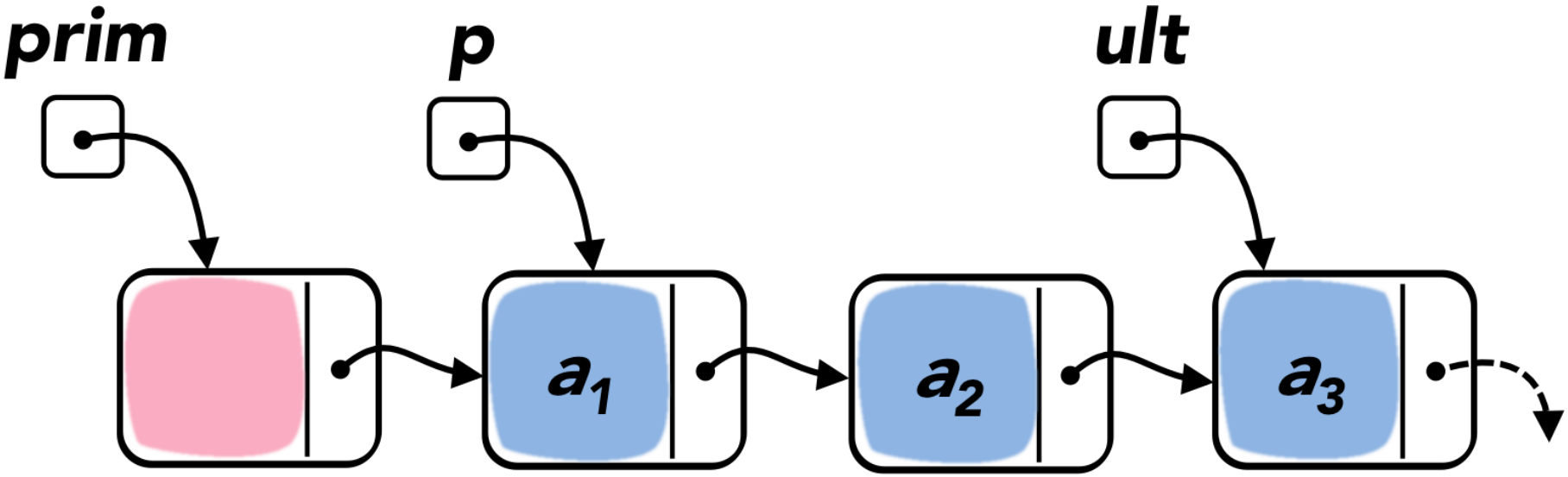


```
BUSCA(L, k)
1    p = L.prim.prox
2    enquanto p ≠ NIL e p.chave ≠ k
3        p = p.prox
4    retorne p
```

BUSCA(L, k)

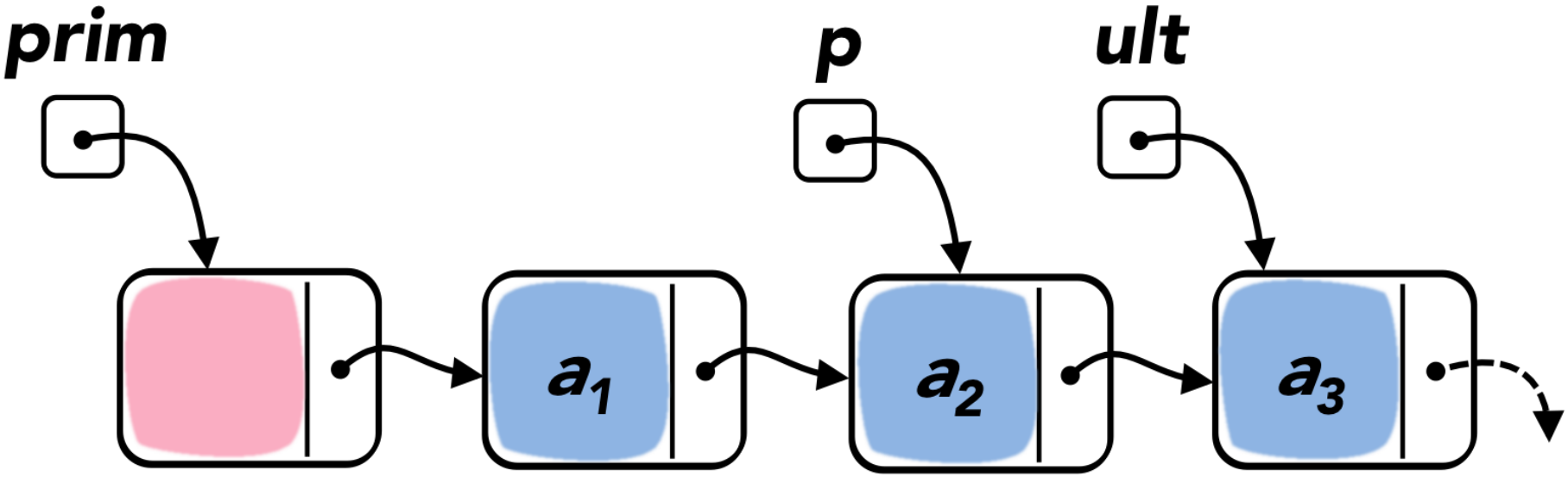


BUSCA(L, k)



(1) p = L.prim.prox

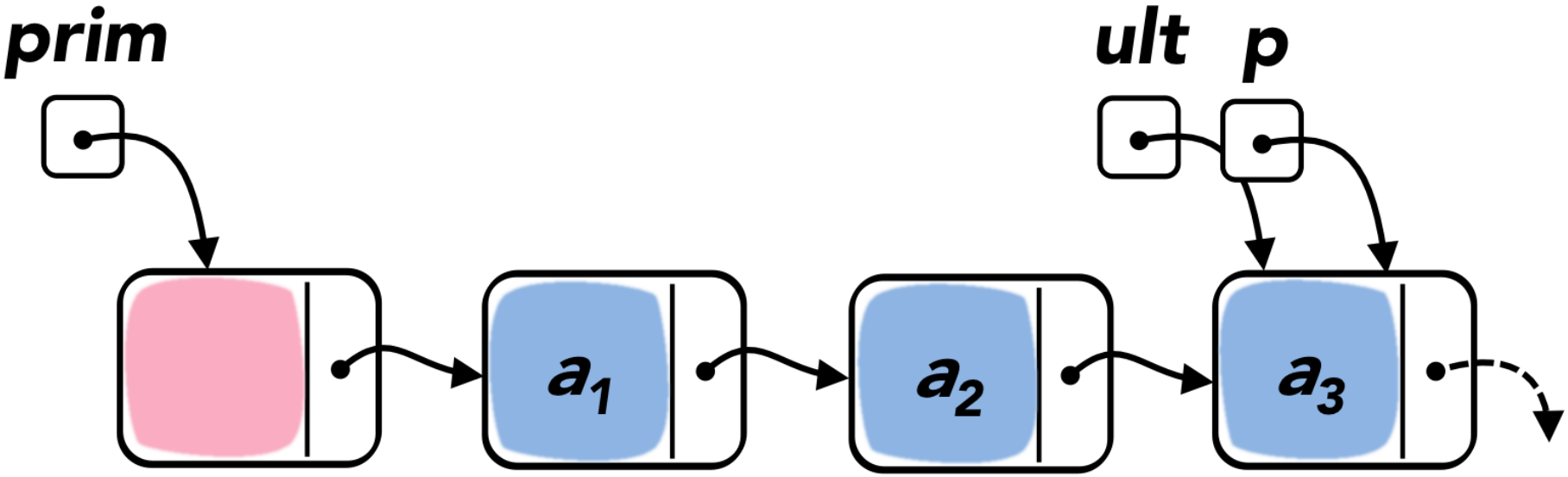
BUSCA(L, k)



(2) **enquanto** p $\neq$ NIL **e** p.chave $\neq$ k

(3) p = p.prox

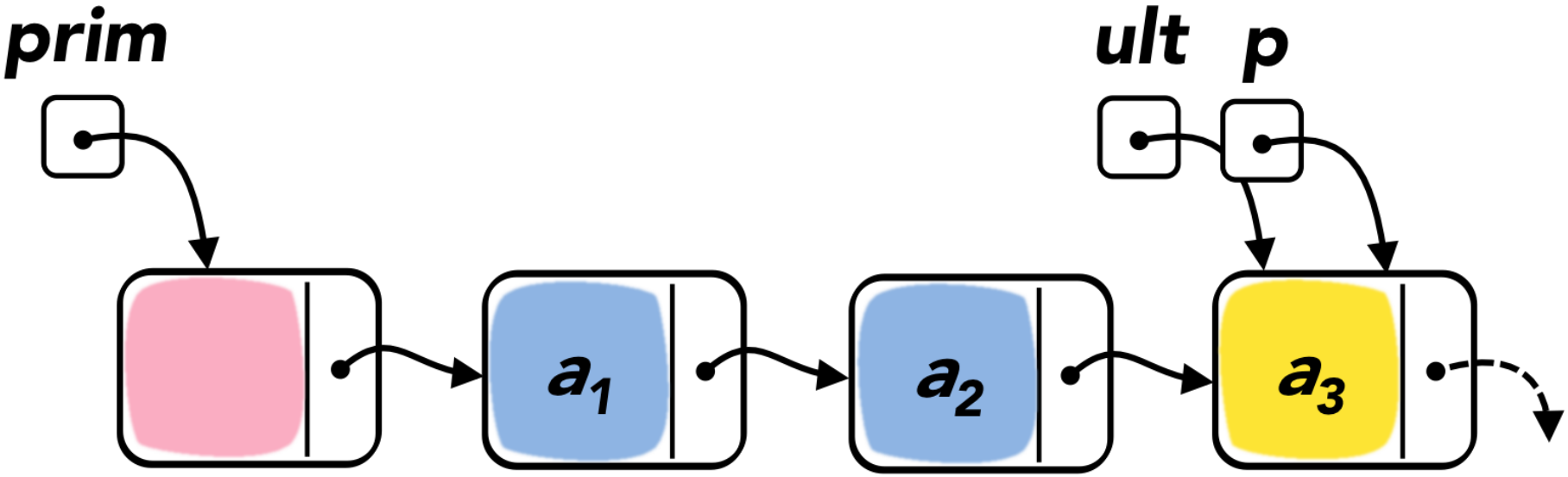
BUSCA(L, k)



(2) **enquanto** p $\neq$ NIL **e** p.chave $\neq$ k

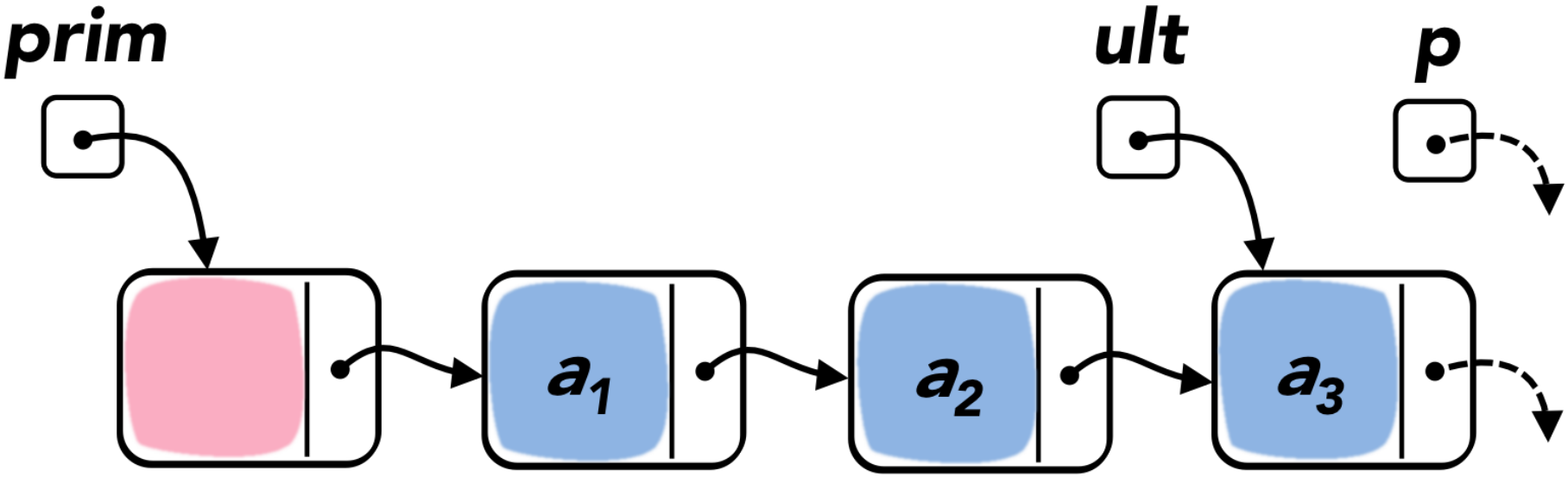
(3) p = p.prox

BUSCA(L, k)



(4) Acha o item: **retorne** p $= a_3$

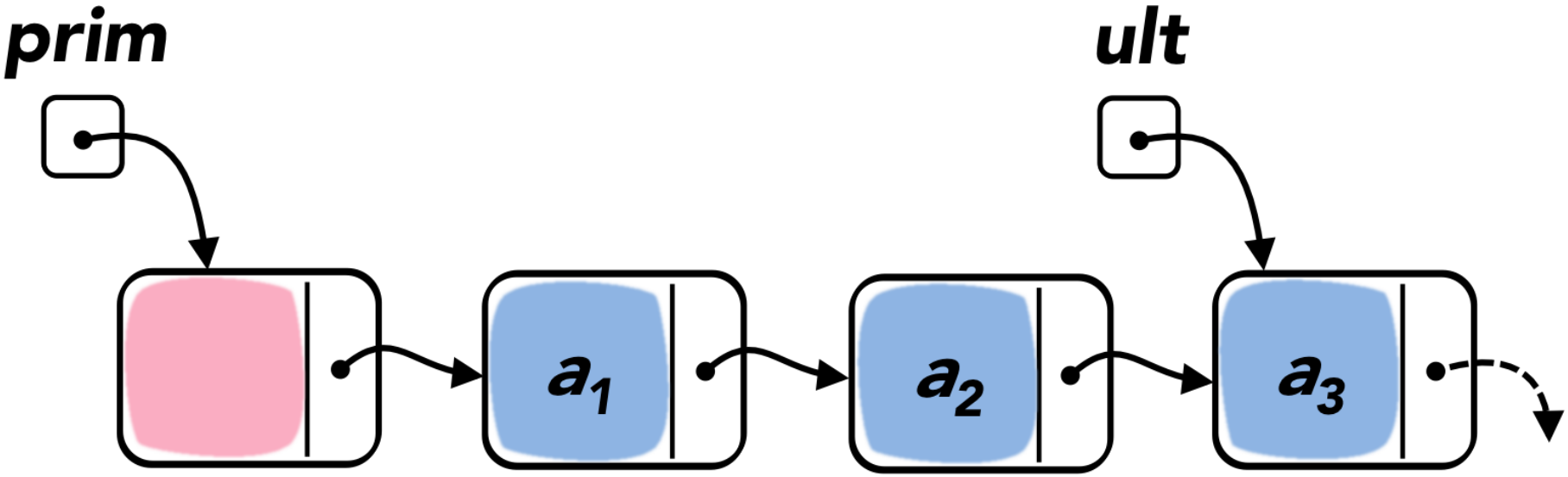
BUSCA(L, k)



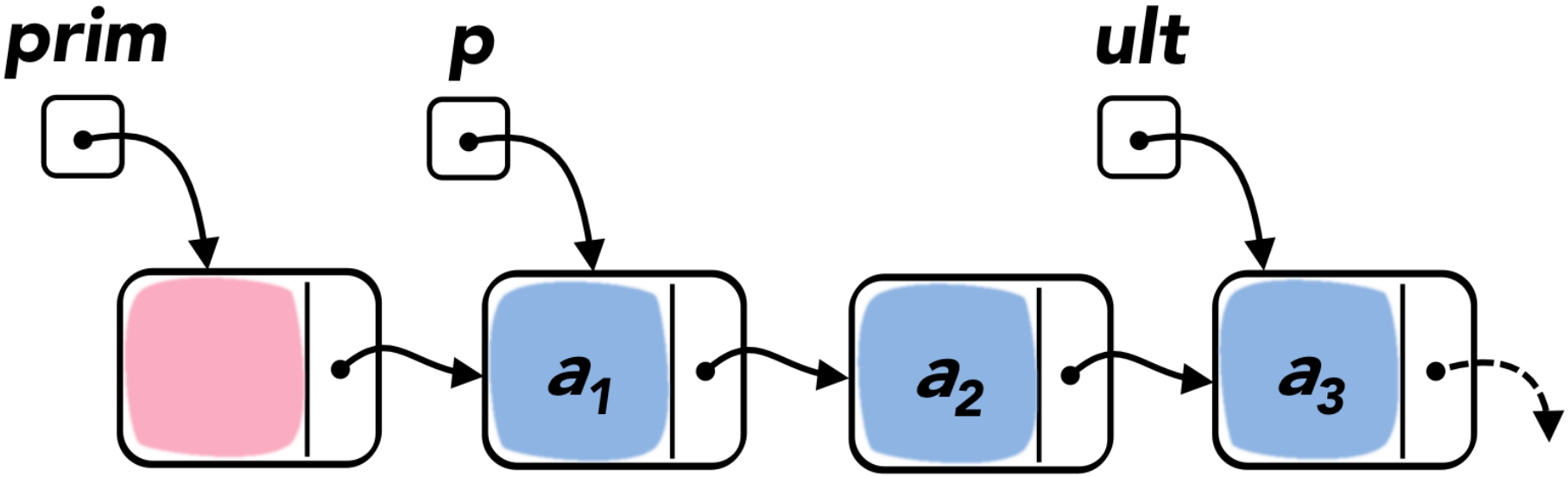
(4) Não acha item: **retorne** p = NIL

```
MOSTRE(L)
1    p = L.prim.prox
2    enquanto p ≠ NIL
3        mostre p.item
4        p = p.prox
```

MOSTRE(L)

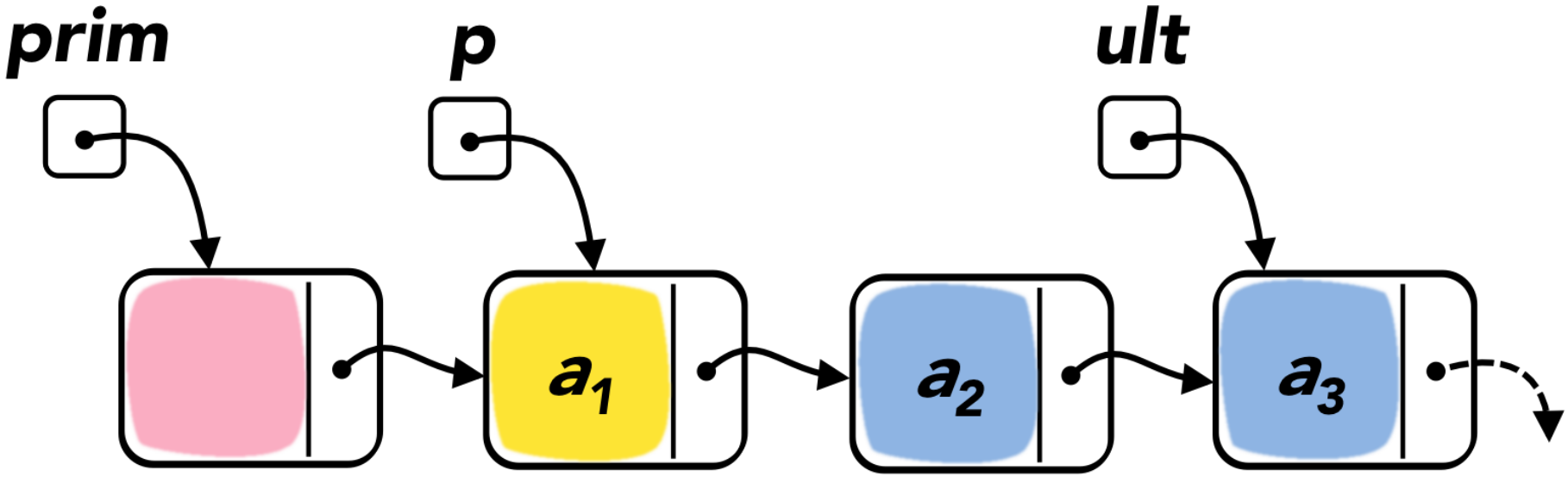


MOSTRE(L)



(1) p = L.prim.prox

MOSTRE(L)

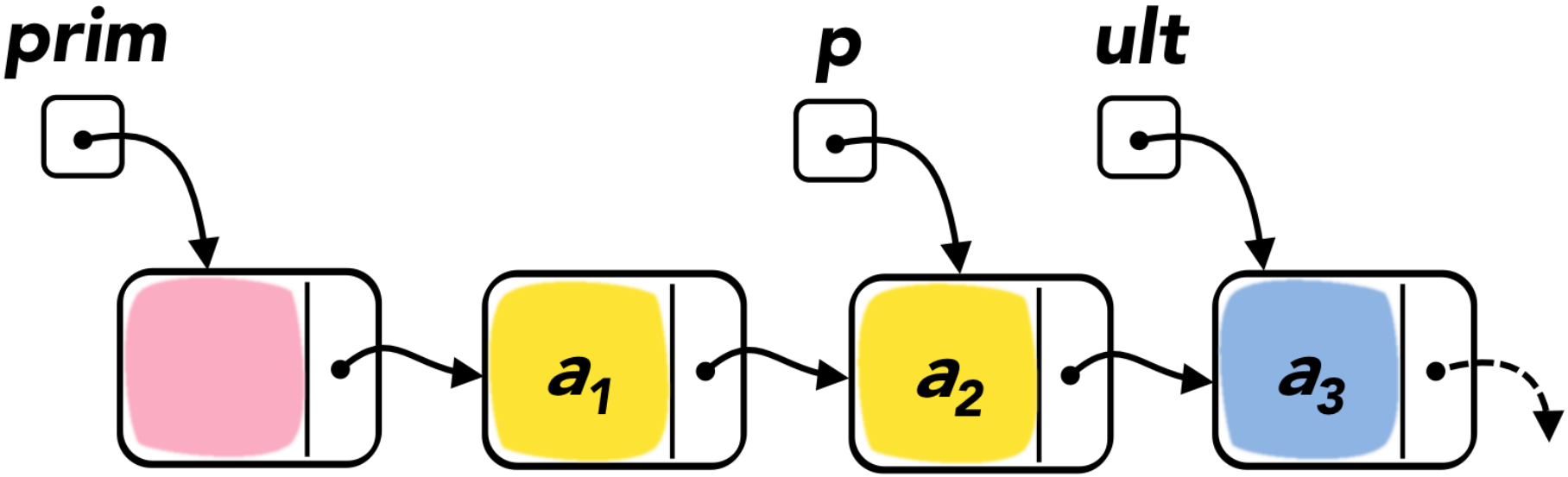


(2) **enquanto** p $\neq$ NIL

(3) **mostre** p.item

(4) p = p.prox

MOSTRE(L)

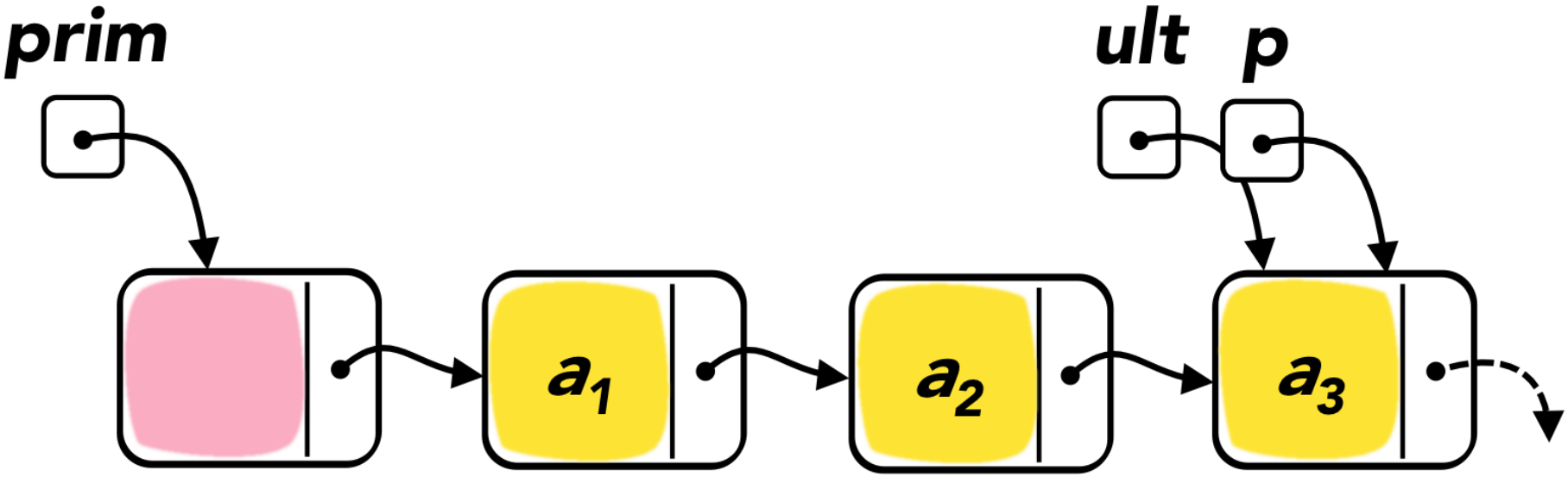


(2) **enquanto** p $\neq$ NIL

(3) **mostre** p.item

(4) p = p.prox

MOSTRE(L)

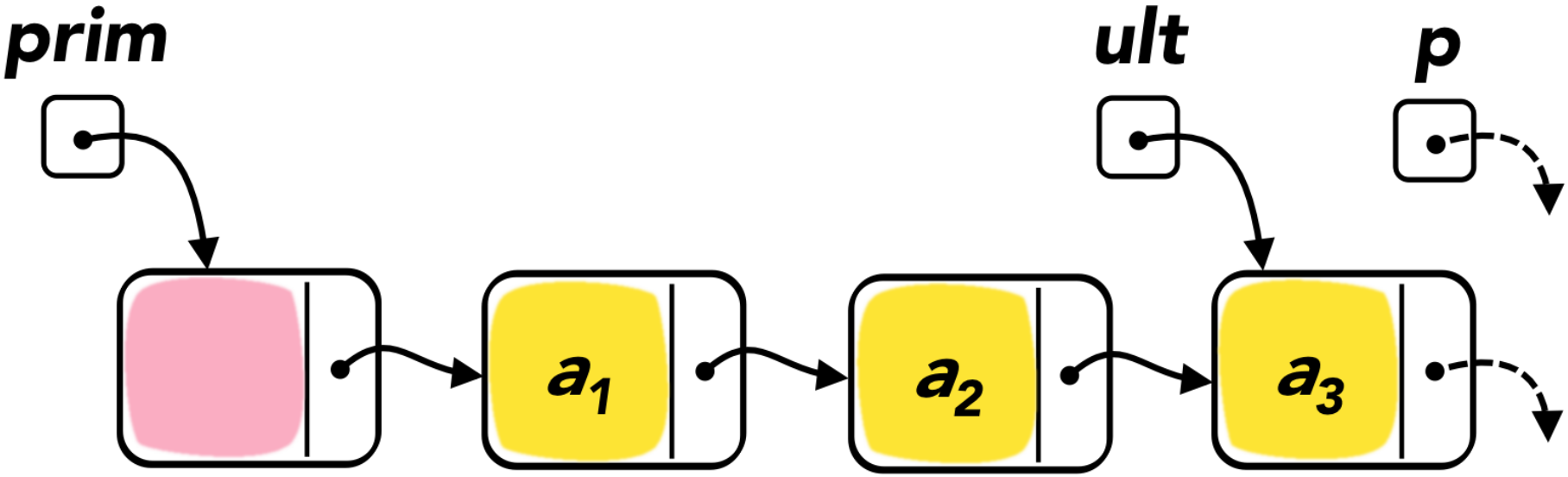


(2) **enquanto** p $\neq$ NIL

(3) **mostre** p.item

(4) p = p.prox

MOSTRE(L)

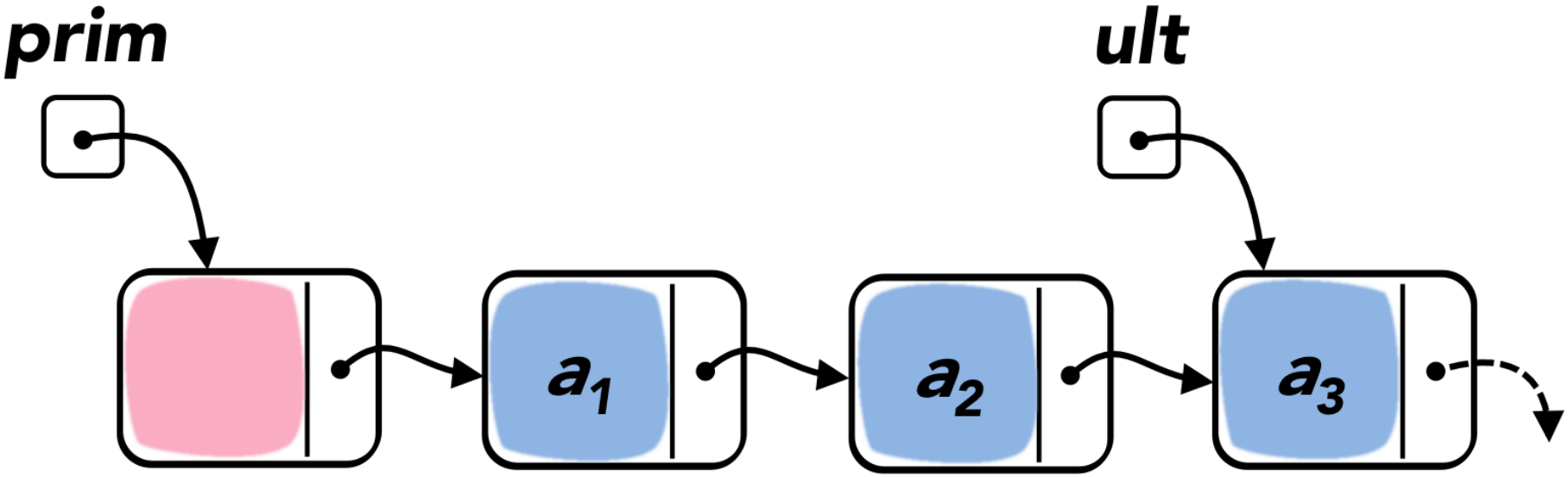


(2) **enquanto** p $\neq$ NIL

(3) **mostre** p.item

(4) p = p.prox

MOSTRE(L)



Conclusão

#1

Lista ligada tem **tamanho dinâmico**, mas é **mais complexa** de se implementar do que lista estática.

Conclusão

#2

Busca, **mostra** são $O(n)$. **Remoção** é $O(n)$, ou $O(1)$ se a lista for **LDE**. **Inserção** é $O(1)$.

Referências

- T. H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2010
- N. Ziviani. Projeto de Algoritmos com Implementação em Pascal C. Cengage Learning, 2012.

Onde obter este material:

est.uea.edu.br/fcoelho