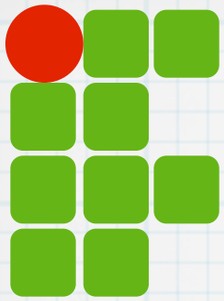
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

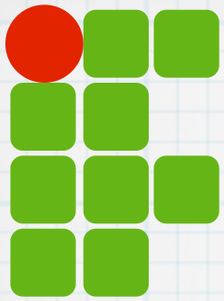
Algoritmos

Lista duplamente ligada



Agenda

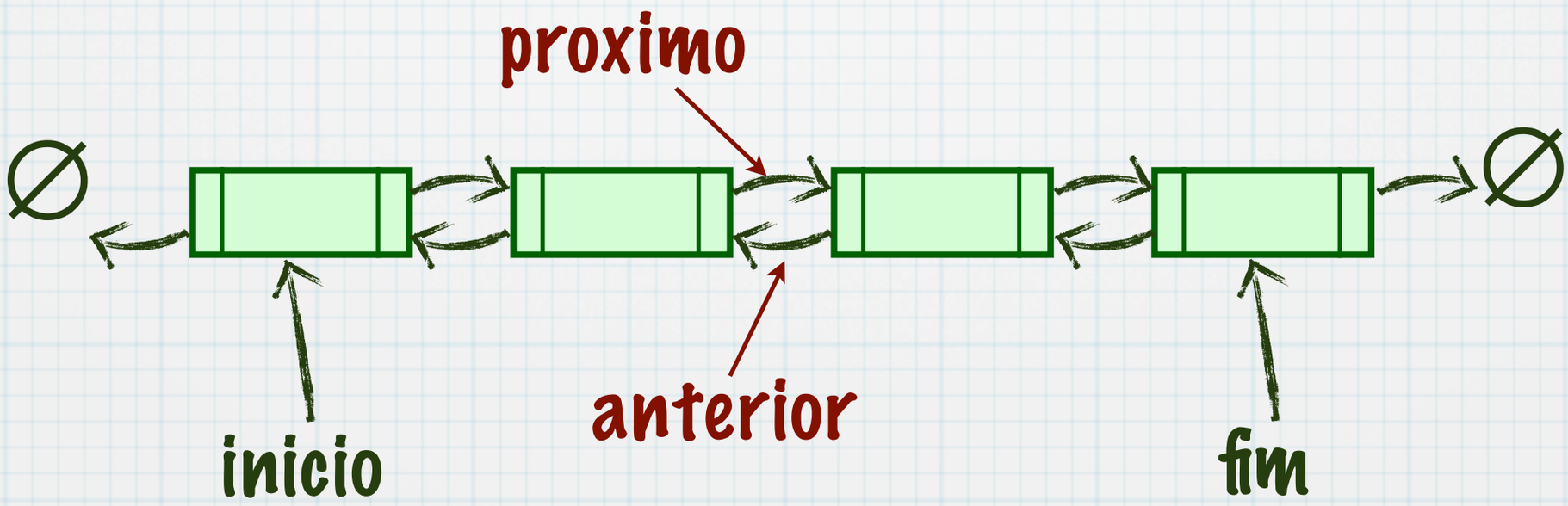
* Lista duplamente ligada
* Listas ligadas vs arrays
* Exercícios



Lista ligada

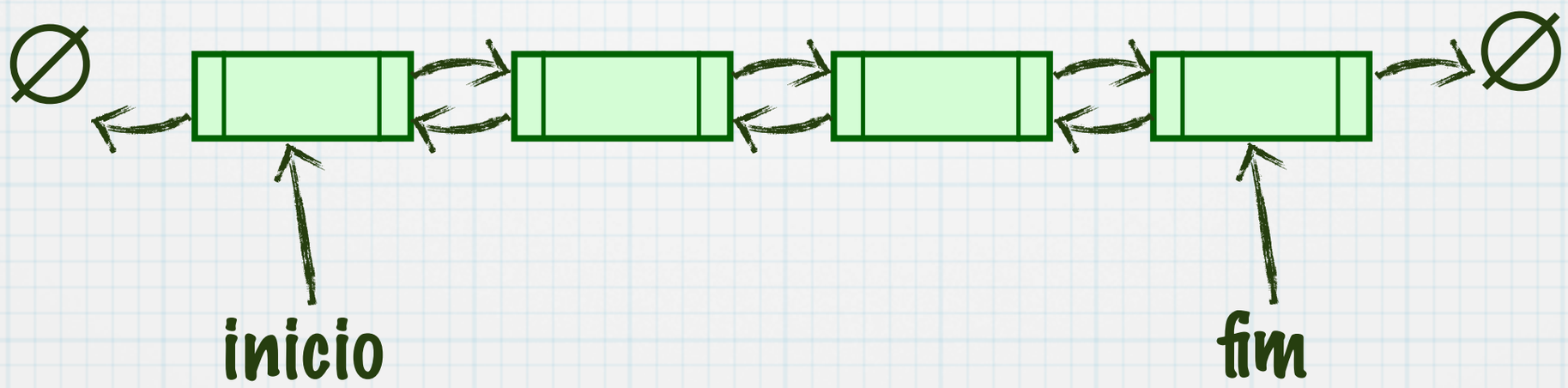
* Lista duplamente ligada
  * Cada nó conhece seu sucessor e também seu antecessor

```c
struct ListaDuplaLigada{
    /* Dados do nó */
    struct ListaDuplaLigada *proximo,*anterior;
};
```



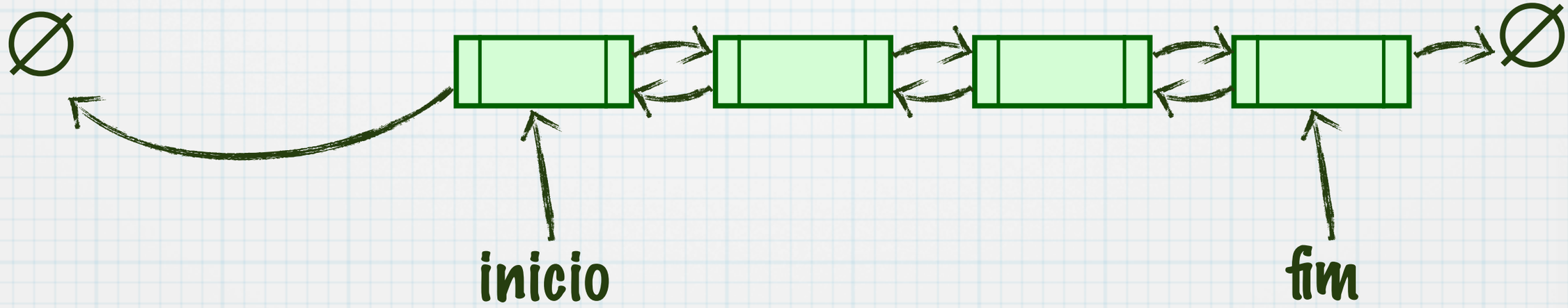
Observações

* Percorre em ambos sentidos
  * Pode otimizar alguns algoritmos
* Usa mais memória
  * Ponteiro ocupa 32 ou 64 bits



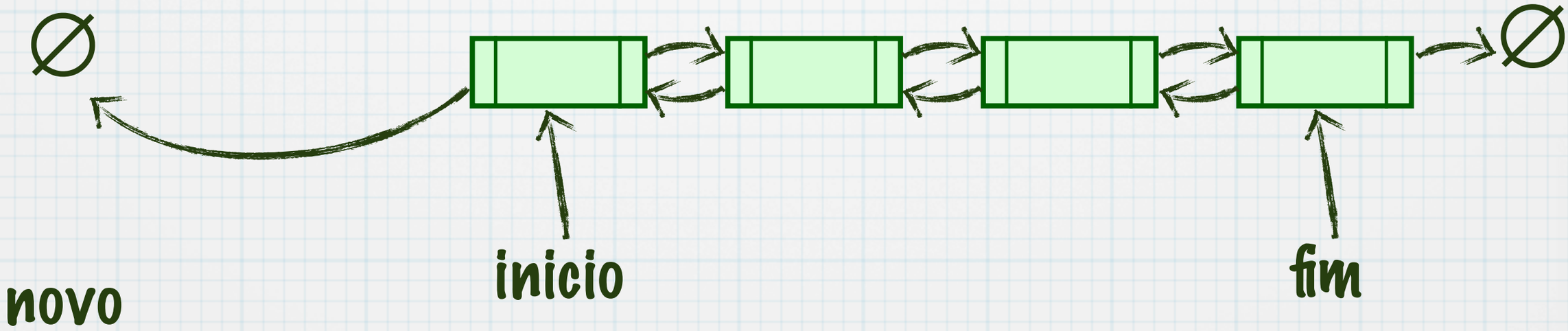
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



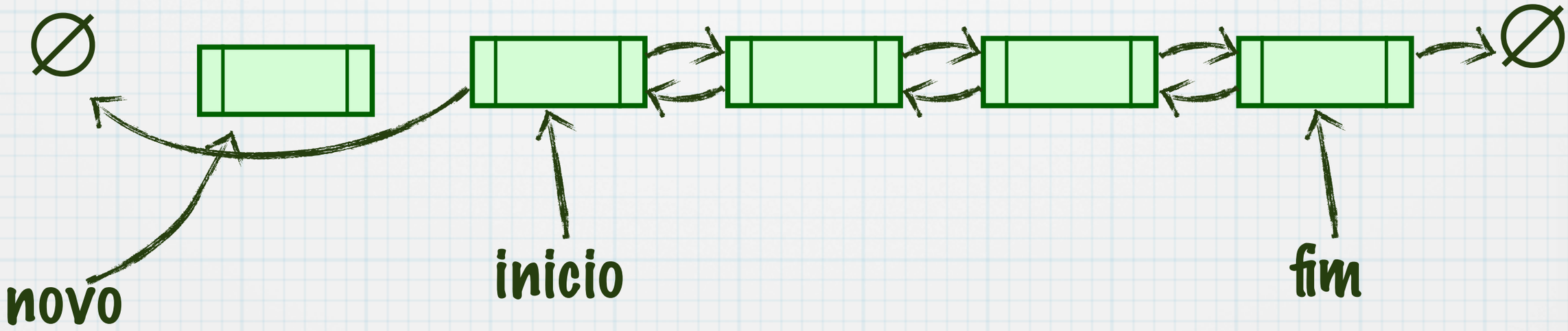
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



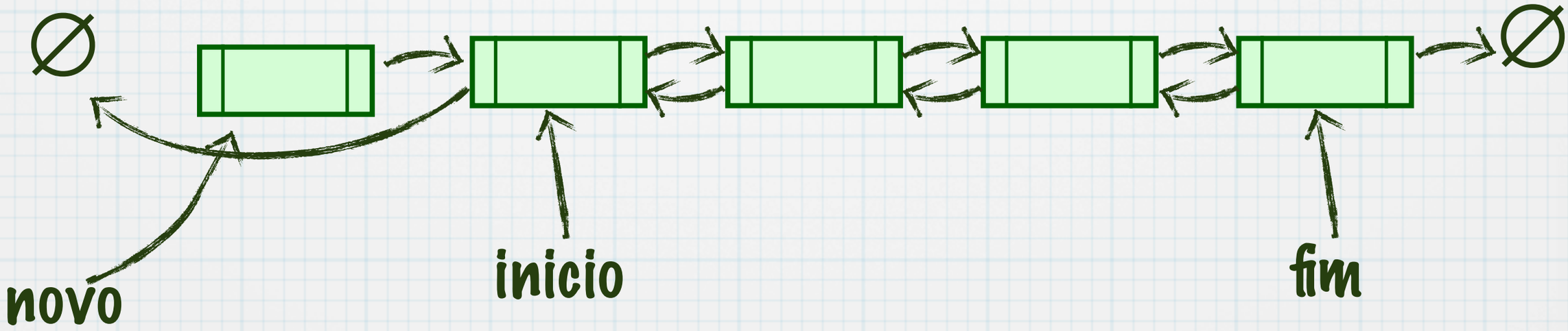
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



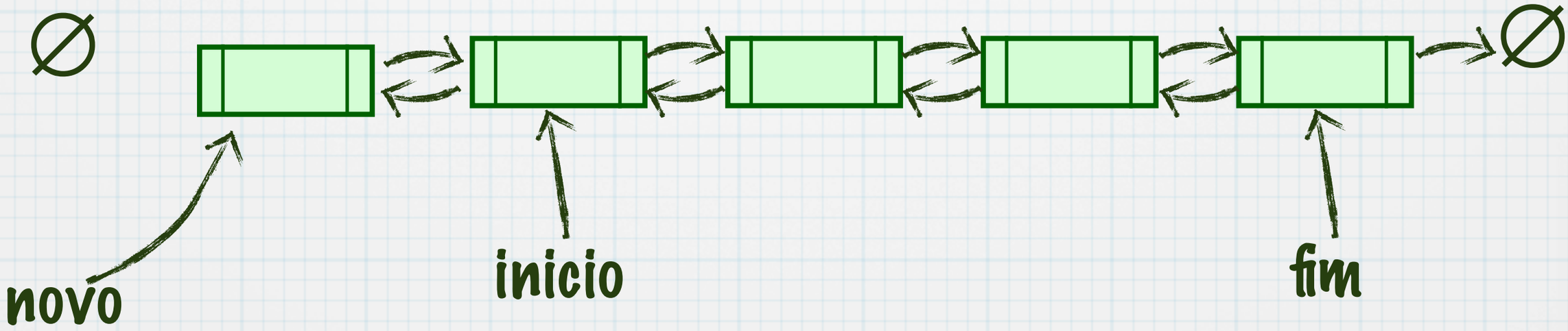
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



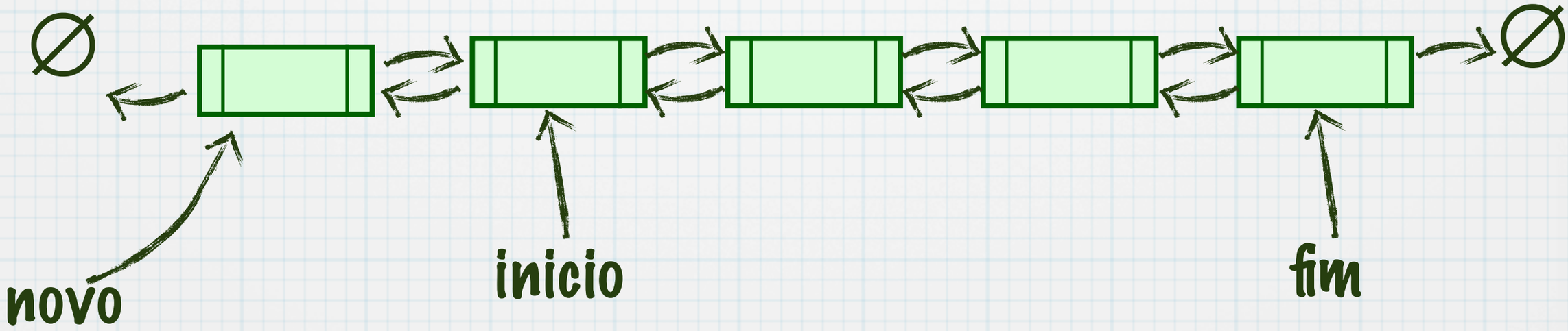
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



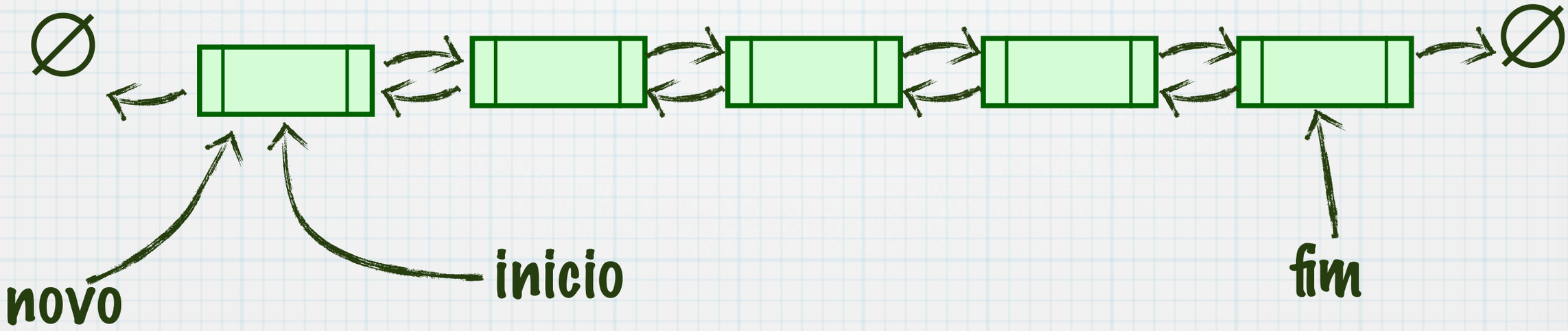
Inserir início

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



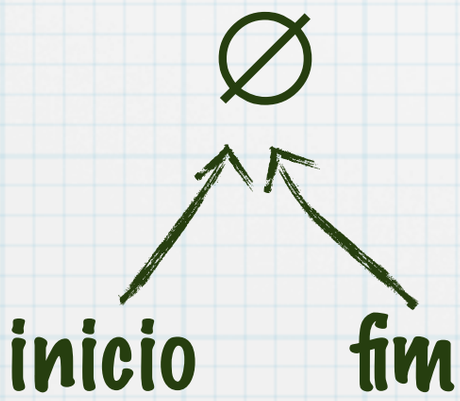
Inserir início

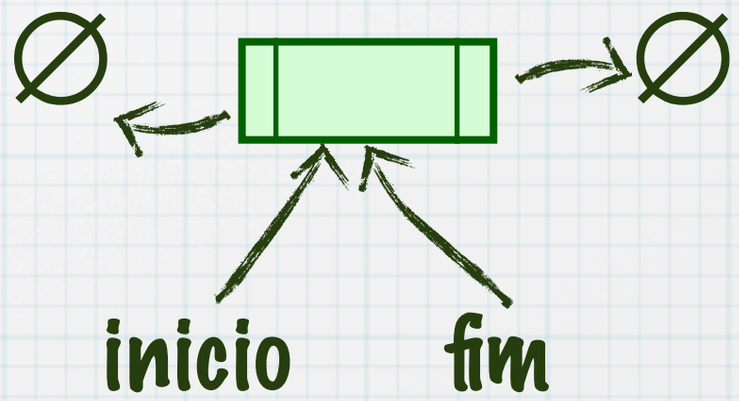
```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = inicio;
inicio->anterior = novo;
novo->anterior = NULL;
inicio=novo;
```



Inserir início

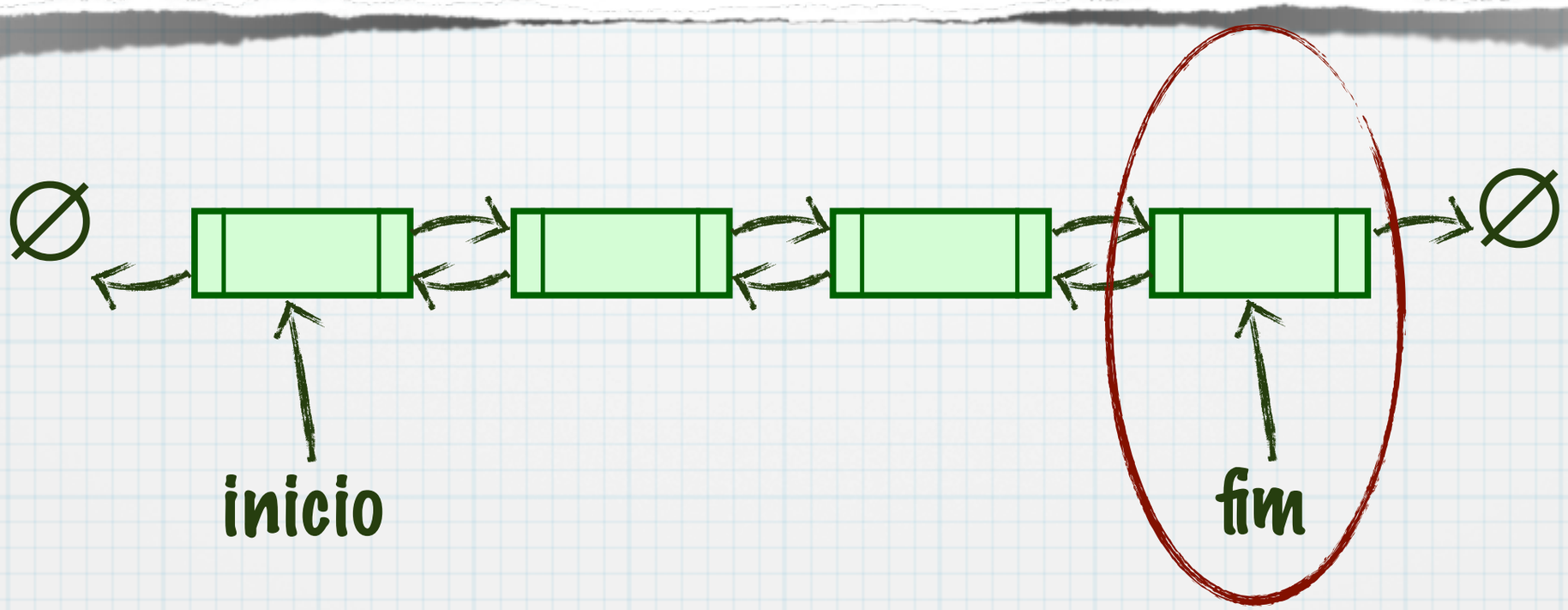
* Cuidado com o ponteiro para o fim da lista
  * Lista pode ser vazia
  * Lista pode conter apenas um elemento





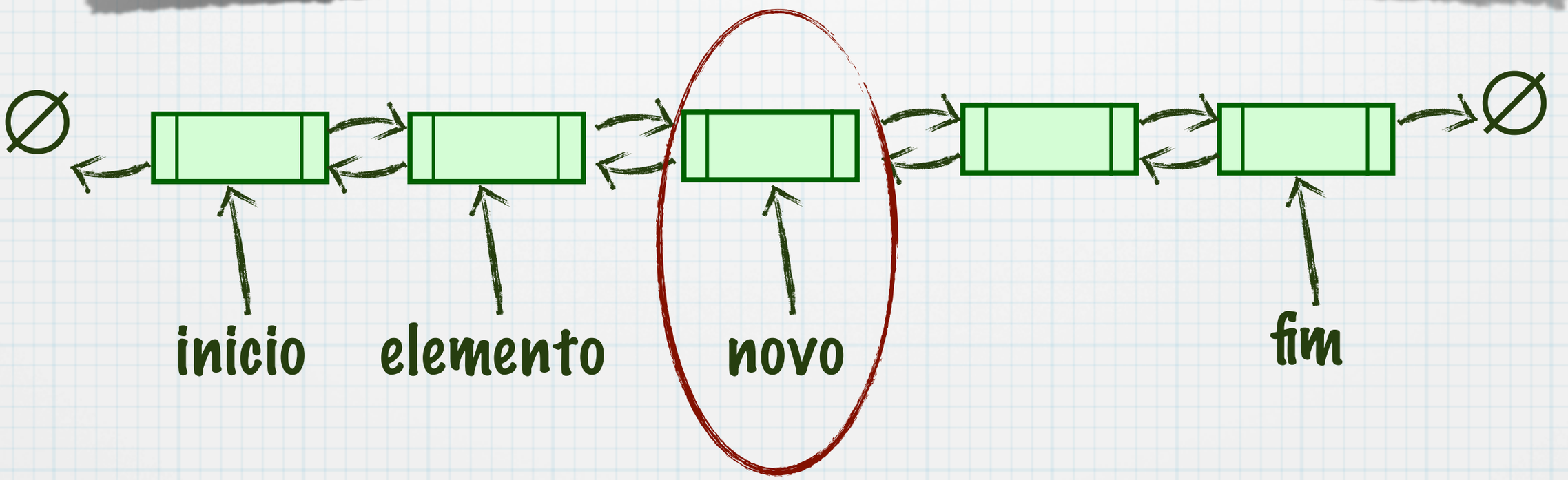
Inserir fim

```
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = NULL;
fim->proxmo = novo;
novo->anterior = fim;
fim=novo;
```



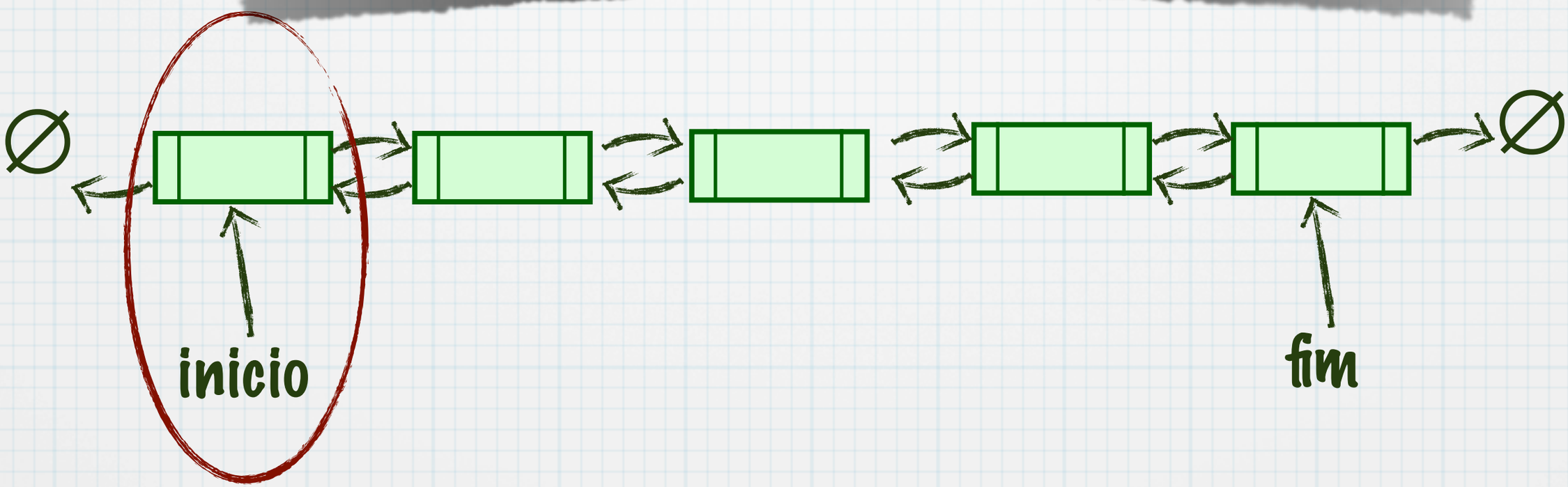
Inserir após

```
/* Buscar elemento */
ListaDuplaLigada *novo;
novo = (ListaDuplaLigada*)malloc(sizeof(ListaDuplaLigada));
novo->proximo = elemento->proximo;
novo->anterior = elemento;
elemento->proximo->anterior = novo;
elemento->proximo = novo;
```



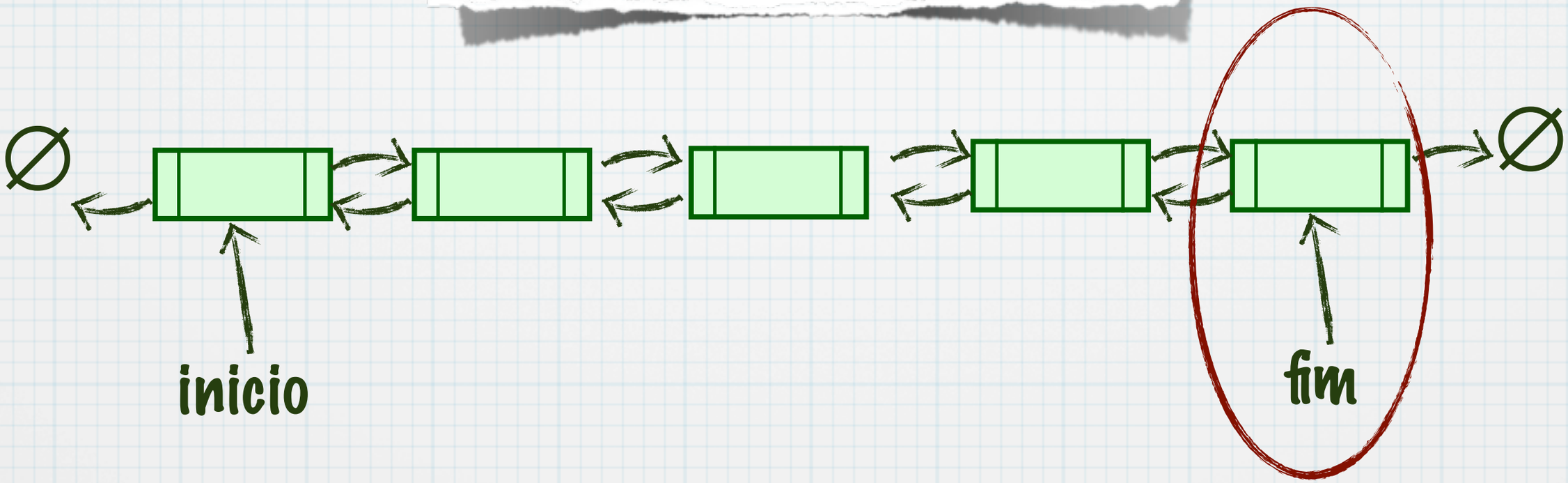
Remover início

```
ListaDuplaLigada *velho;
velho = inicio;
inicio = inicio->proximo;
inicio->anterior = NULL;
free(velho);
```



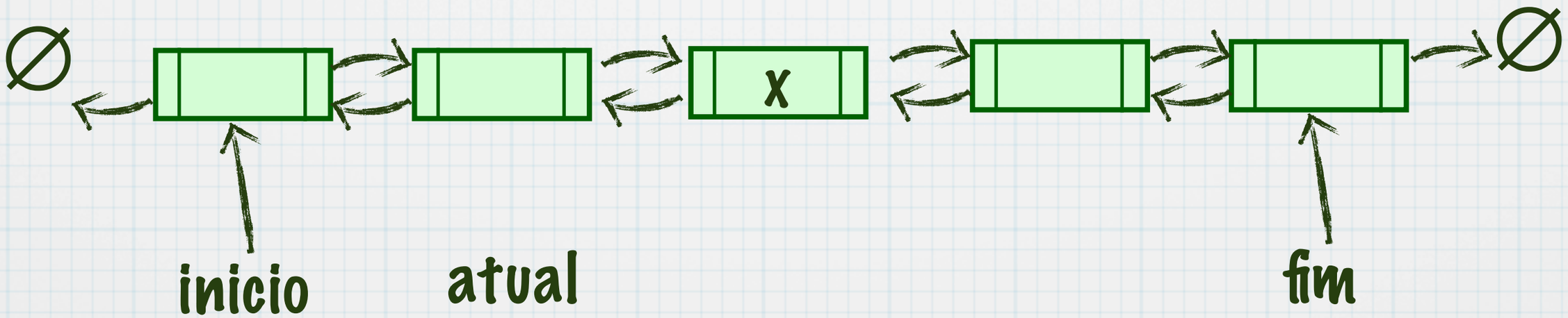
Remover fim

```
ListaDuplaLigada *velho;
velho = fim;
fim = fim->anterior;
fim->proximo = NULL;
free(velho);
```



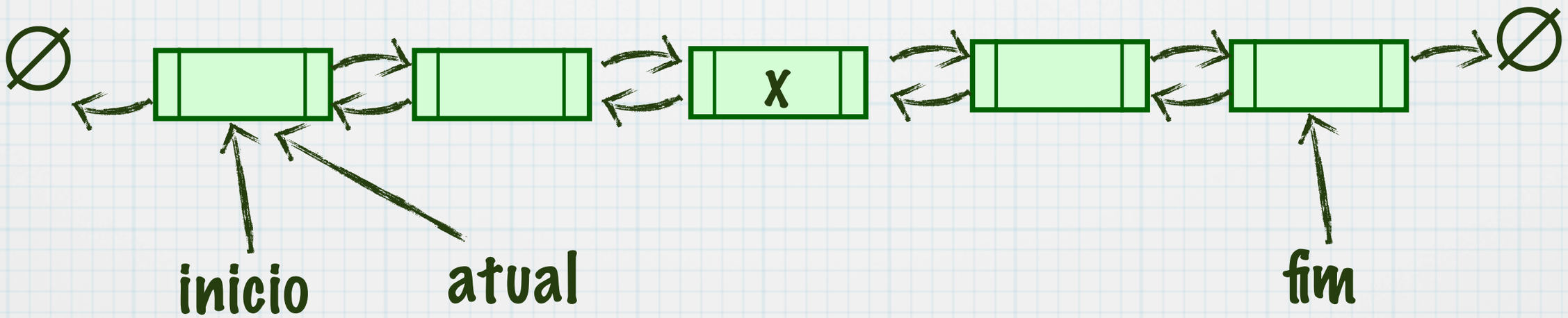
Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



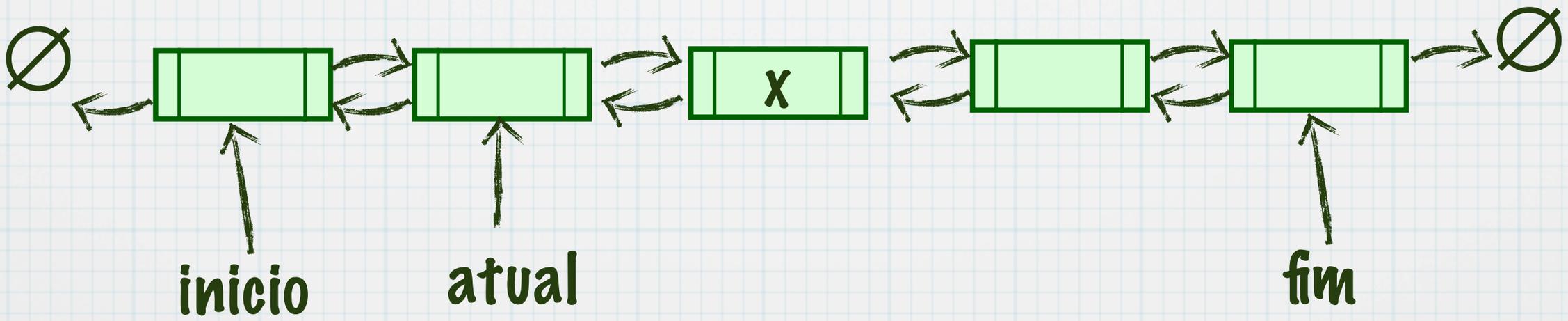
Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



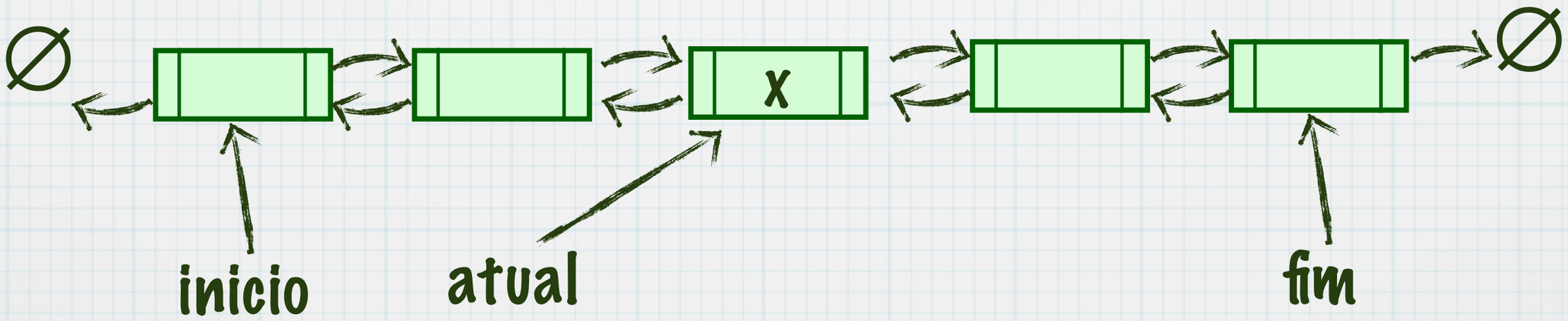
Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



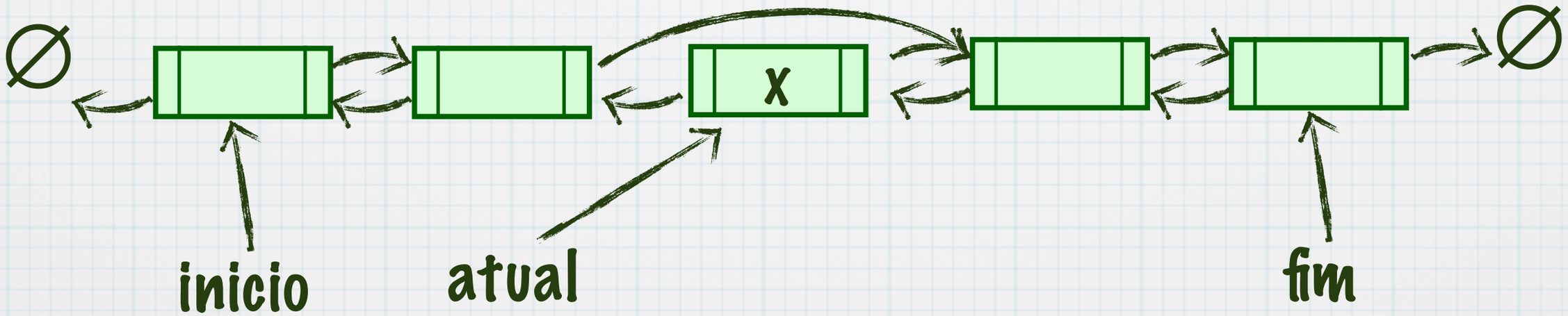
Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



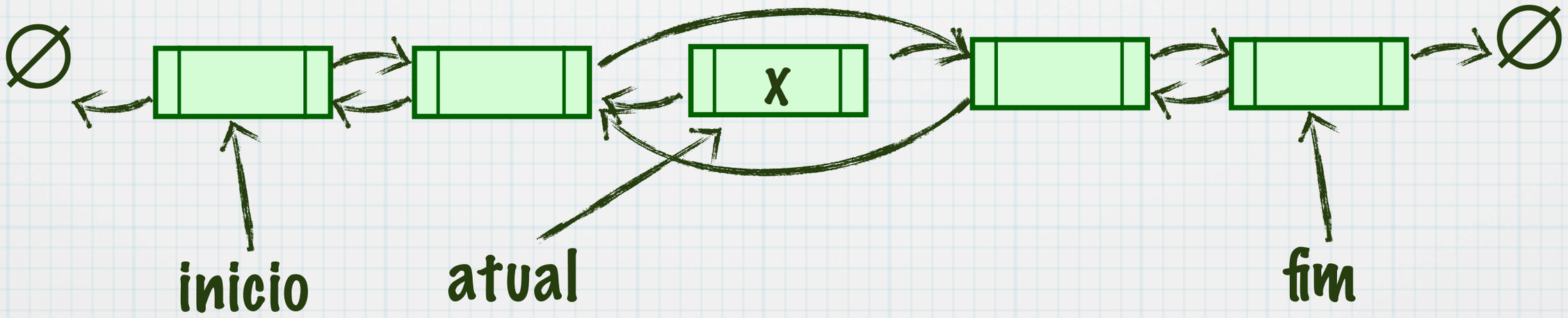
Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



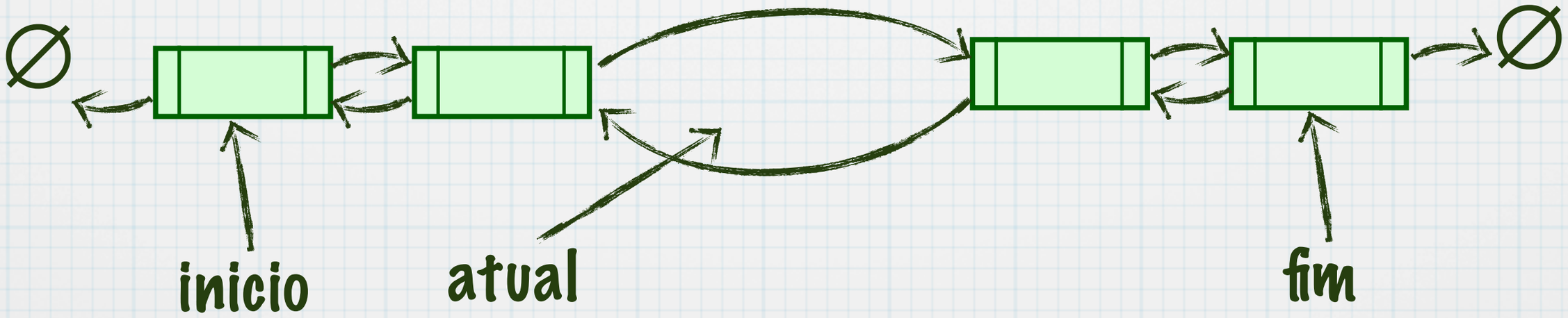
Remover x

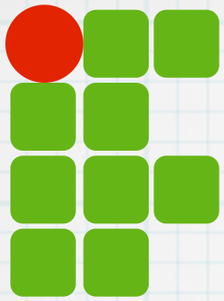
```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```



Remover x

```
ListaDuplaLigada *atual;
atual = inicio;
while (atual != NULL) {
    if (atual->elemento == x) {
        atual->anterior->proximo = atual->proximo;
        atual->proximo->anterior = atual->anterior;
        free(atual);
        break;
    } else {
        atual = atual->proximo;
    }
}
```





Listas vs arrays

* Listas ocupam mais memória (ponteiro)
* Arrays com pouco uso desperdiçam memória
* Modelo de acesso pode ser fundamental para decidir arrays ou listas
  * Acesso pelo índice
  * Percorrer a partir do primeiro/último elemento

Dúvidas?