

---

# Listas Encadeadas

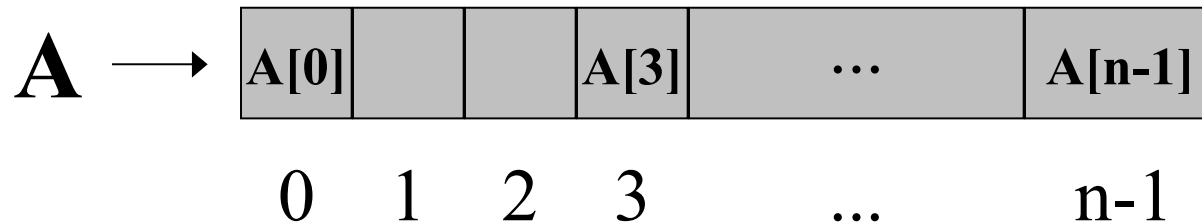
# Tipos de Estruturas de Dados

As estruturas de armazenamento de dados podem ser classificadas da seguinte maneira:

- ***Estruturas estáticas***: podem armazenar até uma quantidade fixa de elementos, que deve ser indicada quando ela é criada;
- ***Estruturas dinâmicas***: o tamanho e a capacidade variam de acordo com a demanda, a medida que o programa vai sendo executado. Em geral, são construídas com ponteiros/referências.

# Estruturas Estáticas: Arrays

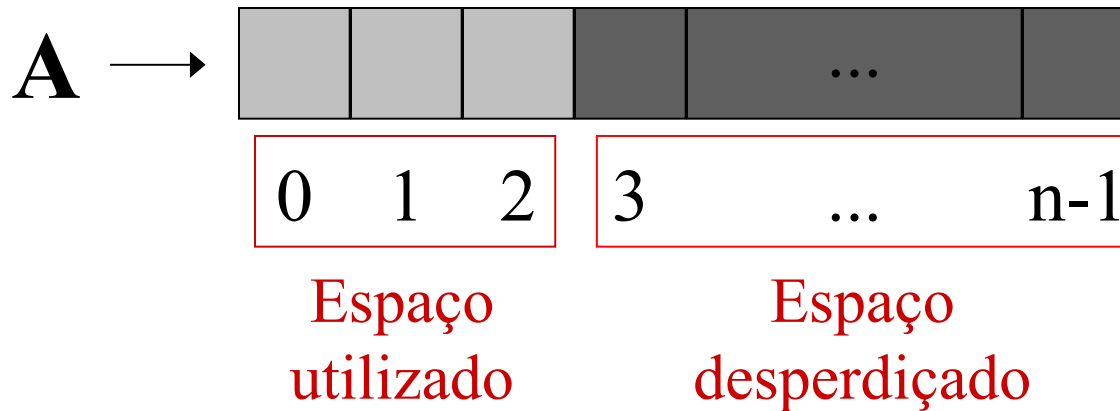
Estruturas que armazenam uma quantidade fixa de elementos do mesmo tipo. O acesso a um elemento é feito a partir do índice do elemento desejado.



Arrays não podem armazenar mais elementos do que o seu tamanho, logo, o tamanho deve ser o máximo necessário.

# Estruturas Estáticas: Arrays

Quando a quantidade de elementos é variável, o uso de arrays pode desperdiçar memória, já que nem todas as suas posições são necessariamente ocupadas.



# Estruturas Dinâmicas: Listas

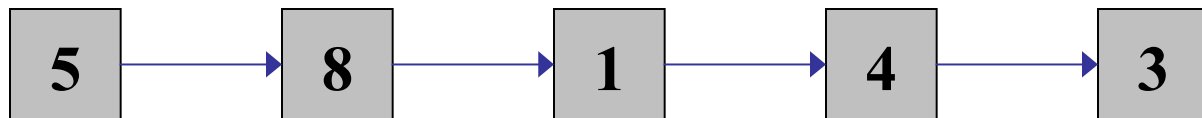
Estruturas criadas para evitar o desperdício de memória, alocando apenas o espaço necessário para seus dados.

A construção é feita a partir de ponteiros/referências.

Antes da  
inserção:

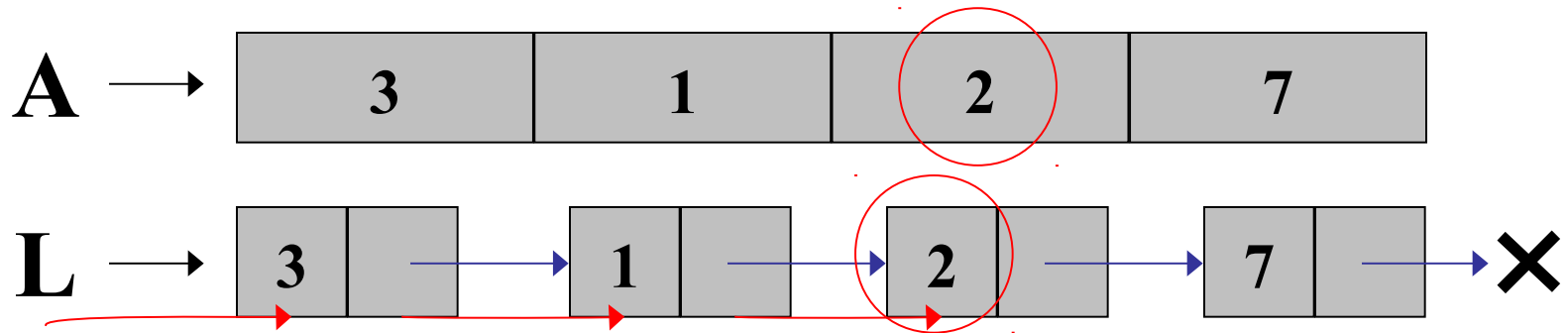


Após a  
inserção:



# Listas Encadeadas

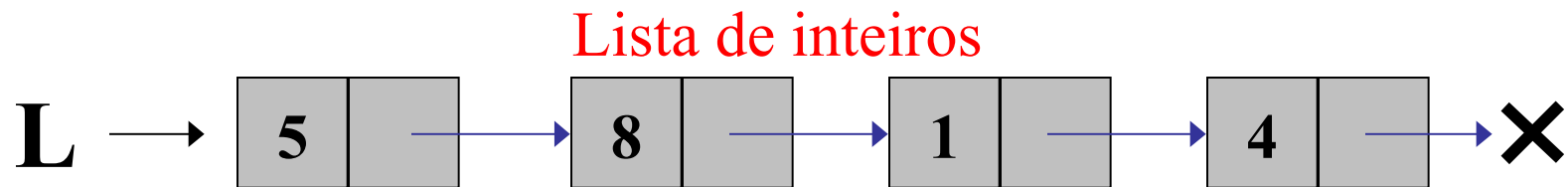
Ao contrário de um array, uma lista não pode acessar seus elementos de modo direto, e sim, de modo sequencial, ou seja, um por vez



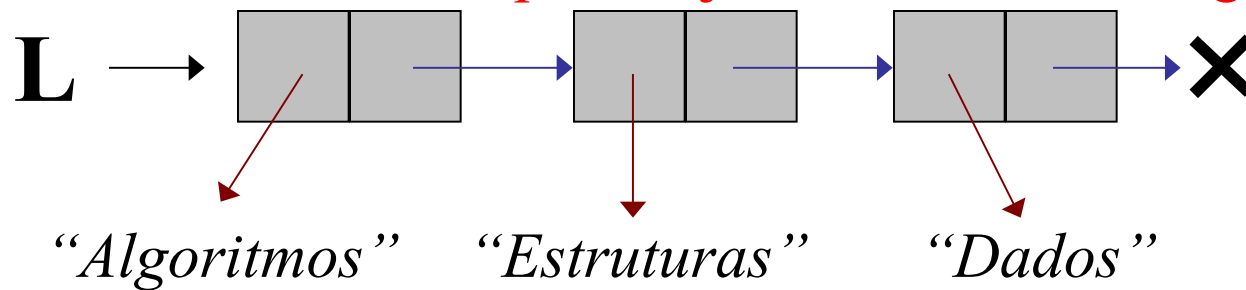
A estrutura **Lista** geralmente contém uma referência para o primeiro elemento da lista (*NoLista inicio*), a partir do qual todos os outros poderão ser acessados.

# Listas Encadeadas

Armazenam uma quantidade variável de elementos do mesmo tipo

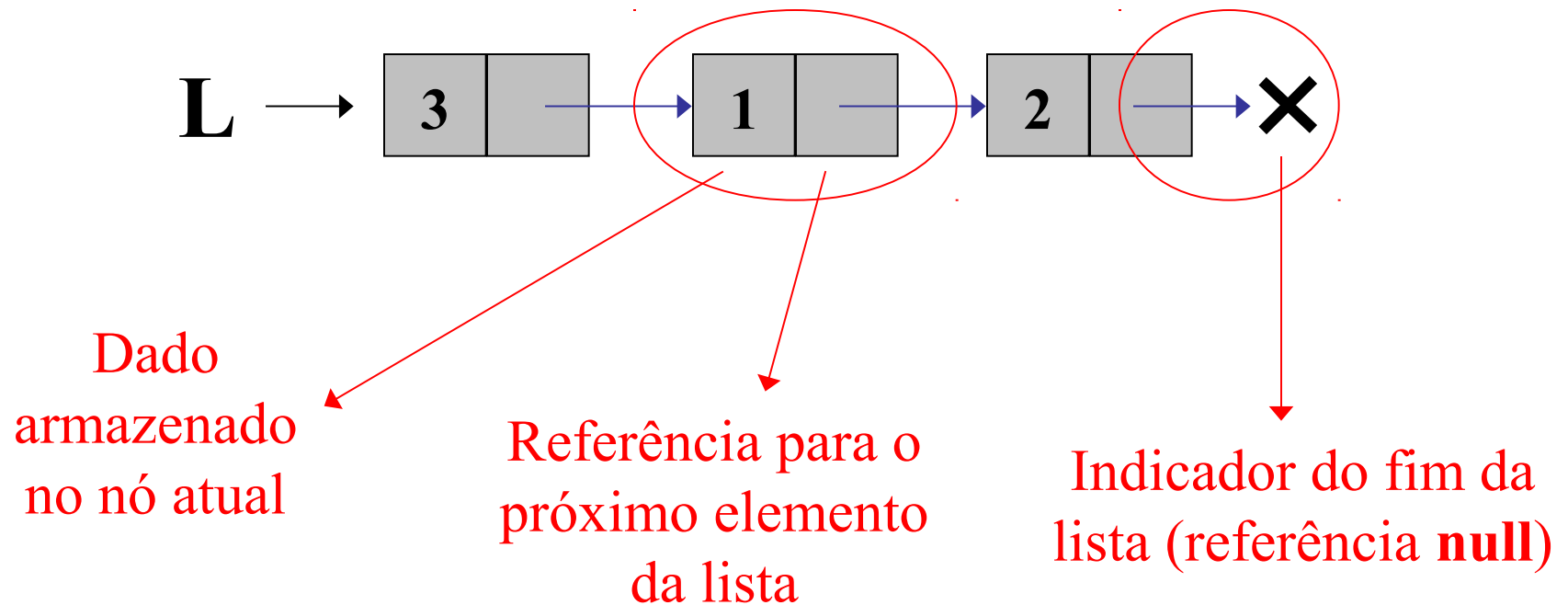


Lista de referências para objetos da classe String



# Listas Simplesmente Encadeadas

Listas são formadas por estruturas chamadas **nós**. Um nó é uma estrutura **auto-referencial**, isto é, contém uma referência para outra estrutura do mesmo tipo





# Listas Simplesmente Encadeadas

Ex: Nó de uma lista de inteiros

```
class NoLista {  
    int valor;  
    NoLista next;  
}
```

Elemento do nó

Referência para o  
nó seguinte

Ex: Nó de uma lista de objetos da classe String

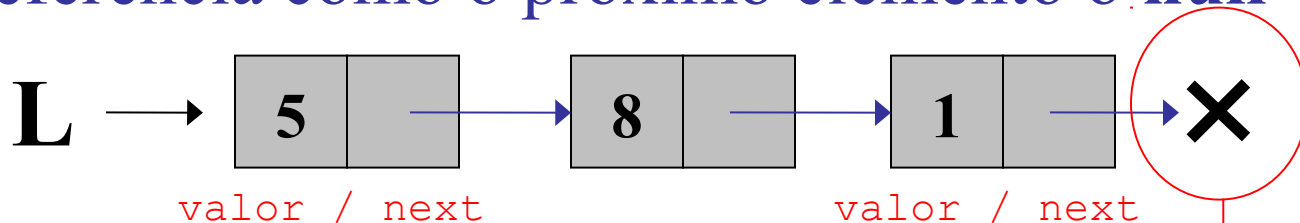
```
class NoLista {  
    String nome;  
    NoLista next;  
}
```

Elemento do nó

Referência para o  
nó seguinte

# Listas Simplesmente Encadeadas

O fim de uma lista é indicada por uma referência nula (representada por um **X**), ou seja, o último nó de uma lista referencia como o próximo elemento o **null**



```
public NoLista (int valor) {  
    this.valor = valor;  
    this.next = null;  
}
```

Indicador do fim da  
lista (referência **null**)

# Listas Simplesmente Encadeadas

Para criar a lista propriamente dita, criaremos a classe **Lista**, que manipula objetos do tipo **NoLista**

```
class Lista {
    NoLista inicio;

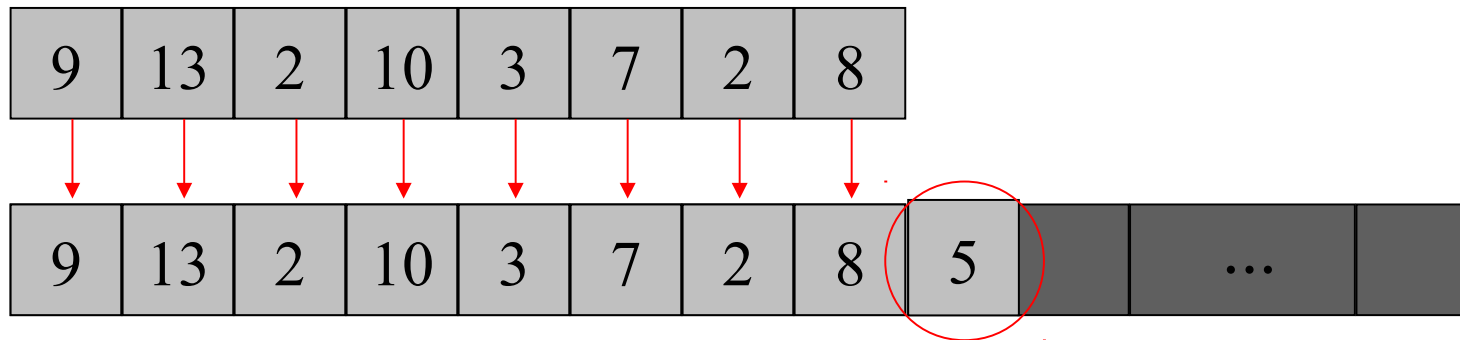
    public Lista() {
        this.inicio = null;
    }

    // insere valor no começo da lista
    public void inserir(int valor) {...}

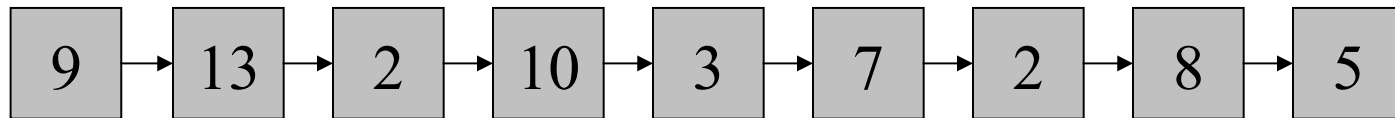
    // insere valor no fim da lista
    public void inserirNoFim(int valor) {...}
}
```

# Inserção em Listas

Para inserir um elemento em um array, pode ser necessário expandi-lo e copiar os elementos um a um:



Em uma lista, basta criar um nó, encontrar a posição desejada e inseri-lo



# Inserção em Listas

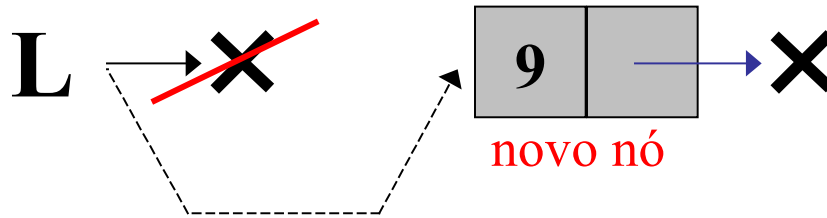
Qualquer operação em uma lista (inserção, remoção, busca, etc.) simples/duplamente encadeada é feita a partir de um NoLista armazenado na classe Lista. Se o NoLista não existir (referência nula), a lista está vazia

```
if (lista.inicio == null) {  
    ...  
}
```

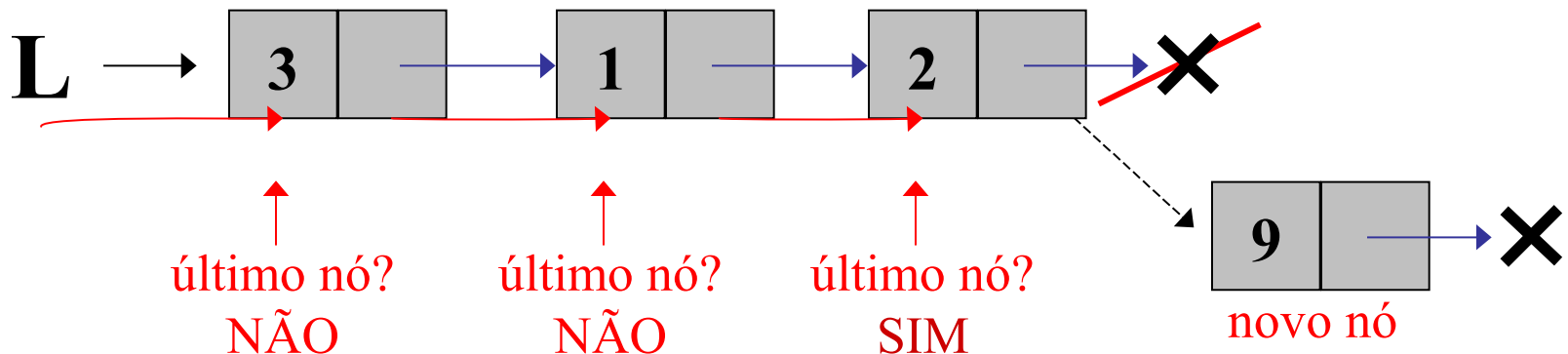
Se estiver vazia, a inserção torna-se uma tarefa fácil: basta atribuir o início ao novo nó. Senão, é preciso encontrar o local correto para inserir o novo elemento, sendo geralmente o começo ou o fim da lista.

# Inserção em Listas

Se a lista estiver vazia:



Caso contrário, inserindo no fim da lista teremos:



# Inserção em Listas

```
public void inserirNoFim(int valor) {  
    if (this.inicio == null) {  
        // lista vazia  
        this.inicio = new NoLista(valor);  
    } else {  
        // procura pelo fim da lista  
        NoLista atual = this.inicio;  
        while (atual.next != null)  
            atual = atual.next;  
        // insere o nó no fim da lista  
        atual.next = new NoLista(valor);  
    }  
}
```

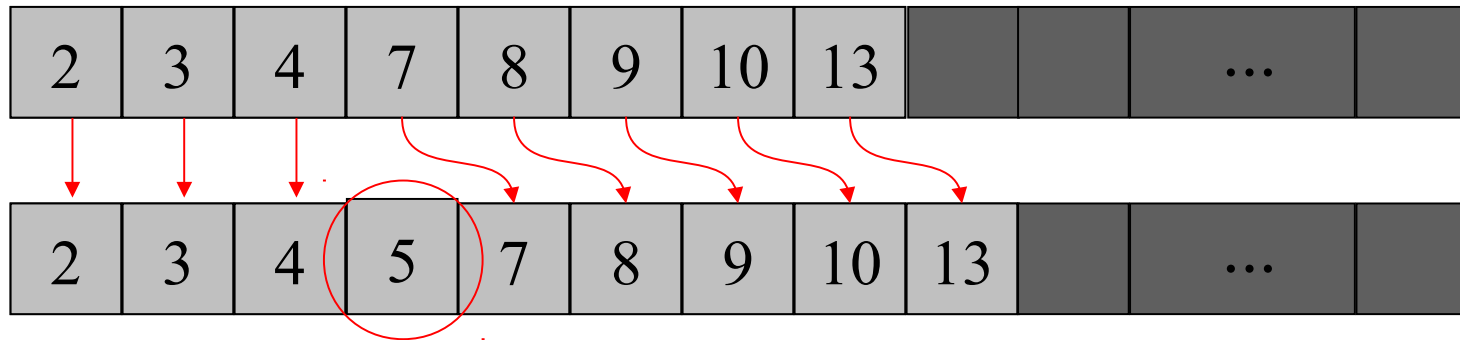
# Inserção em Listas

```
public void inserir(int valor) {  
    if (this.inicio == null) {  
        // lista vazia, então só é preciso criar o nó  
        this.inicio = new NoLista(valor);  
    } else {  
        // cria-se novo nó e atualiza o NoLista inicio  
        NoLista novoNo = new NoLista(valor);  
        novoNo.next = this.inicio;  
        this.inicio = novoNo;  
    }  
}
```

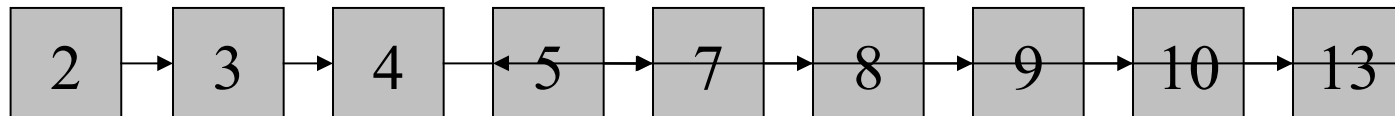


# Inserção Ordenada em Listas

Para inserir um elemento em um array em uma posição qualquer, pode ser necessário mover vários elementos:

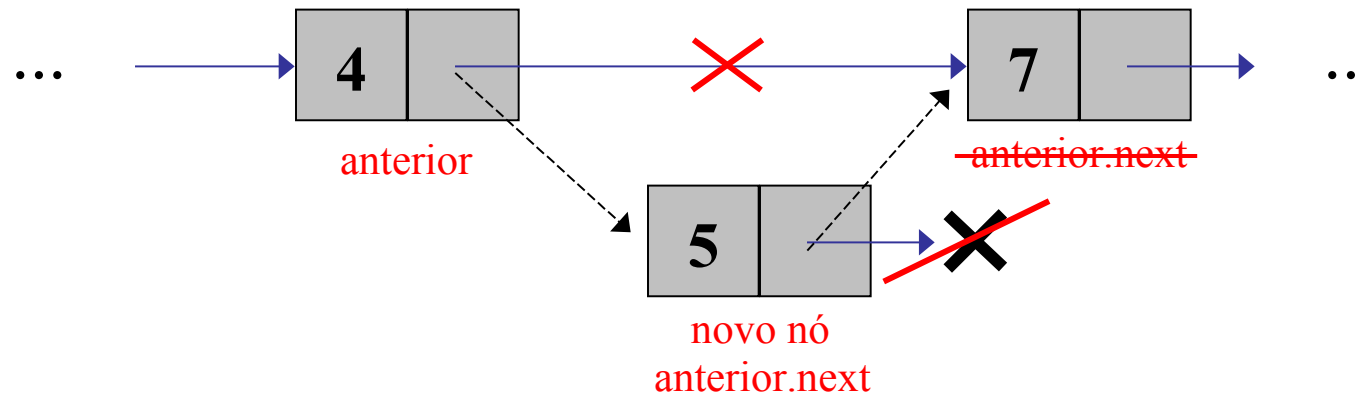


Da mesma maneira, em uma lista, basta criar um nó, encontrar a posição desejada e inseri-lo



# Inserção Ordenada em Listas

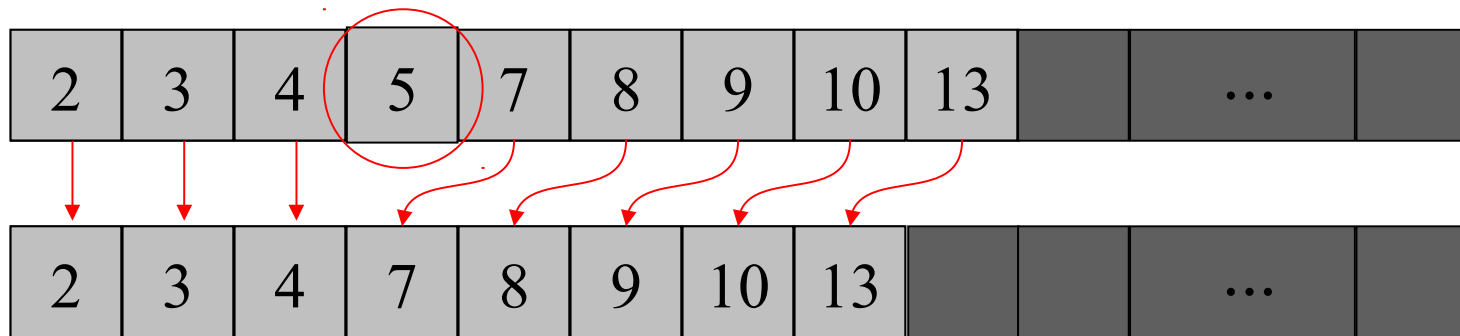
Para inserir um nó entre dois outros nós:



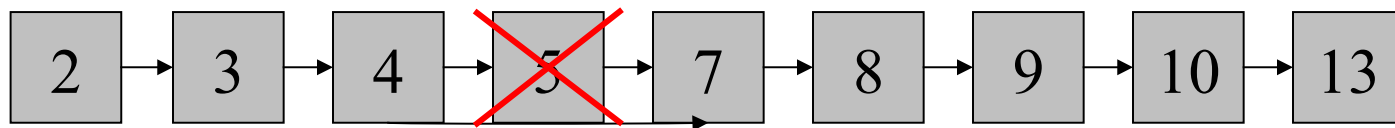
```
NoLista novoNo = new NoLista(5);  
novoNo.next = anterior.next;  
anterior.next = novoNo;
```

# Remoção em Listas

Para remover um elemento de uma posição qualquer do array, pode ser necessário mover vários elementos:

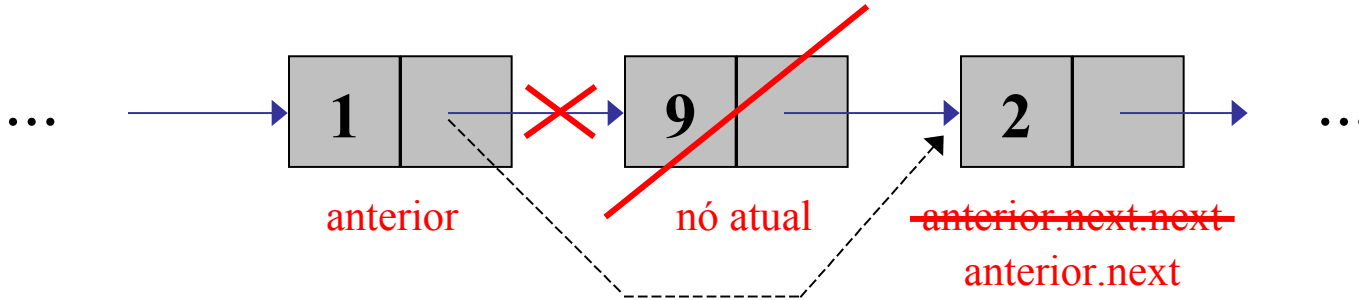


Para remover um elemento de uma lista, basta encontrar o nó correspondente e alterar os ponteiros



# Remoção em Listas

Para excluir um nó entre dois outros nós:



```
anterior.next = anterior.next.next;
```

Após isso, o nó atual não é mais utilizado, e o *garbage collector* já pode liberar a memória ocupada.

Não precisa e nem se deve chamar `System.gc()`!

# Genéricos

- Adicionado a C# 2.0 e posteriormente a Java 5
- Classes Genéricas, que utilizam o conceito de “parâmetros tipo”<..>
- Lista com Genéricos: cada lista armazena um tipo específico, sem precisar criar código novo para cada tipo

Sem Genéricos:

```
class NoListaI{  
    int valor;  
    NoLista next;  
}
```

```
class NoListaS{  
    String nome;  
    NoLista next;  
}
```

Com Genéricos:

```
class NoLista<E> {  
    E elemento;  
    NoLista<E> next;  
}
```

# Listas com Genéricos

```
class Lista<E> {  
    NoLista<E> inicio;  
  
    public Lista() {  
        this.inicio = null;  
    }  
  
    public void inserir(E elemento) {  
        if (this.inicio == null) {  
            this.inicio = new NoLista<E>(elemento);  
        } else {  
            NoLista<E> novoNo = new NoLista<E>(elemento);  
            novoNo.next = this.inicio;  
            this.inicio = novoNo;  
        }  
    }  
}
```

# Arrays vs. Listas

---

- Arrays podem ocupar espaço desnecessário na memória, mas seu acesso é feito diretamente
- Listas ocupam apenas o espaço necessário, mas é preciso espaço extra para armazenar as referências. Além disso, seu acesso é sequencial, ou seja, a busca inicia por um nó, depois vai pra outro nó, e assim por diante, até que se encontre o nó procurado.
- Listas duplamente encadeadas (dois ponteiros dentro de cada nó, um para o próximo nó e outro pro anterior) dão maior poder de implementação das funções, apesar dos custos adicionais de memória por conta do número de ponteiros.