

O TAD Vetor

- ◆ O TAD **Vetor** estende a noção de arranjo (*array*) armazenando sequências de objetos arbitrários
- ◆ Um elemento pode ser acessado, inserido ou removido através da especificação de sua **colocação** (*rank*)
- ◆ Uma exceção é disparada se uma colocação incorreta é especificada
- ◆ Principais operações:
 - object **elemAtRank**(integer r): retorna o elemento na colocação r, sem removê-lo
 - object **replaceAtRank**(integer r, object o): substitui o elemento na colocação r por o e retorna o antigo elemento
 - **insertAtRank**(integer r, object o): insere um novo elemento o na colocação r
 - object **removeAtRank**(integer r): remove e retorna o elemento na colocação r
- ◆ Operações adicionais **size()** e **isEmpty()**

Aplicações de vetores

◆ Aplicações diretas

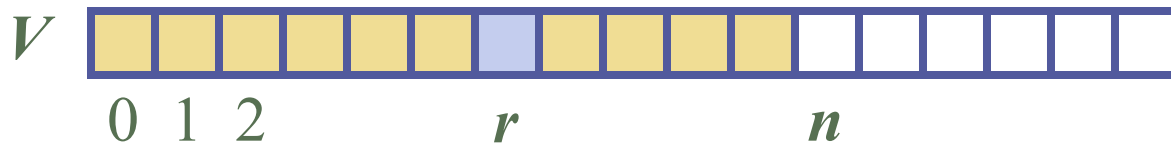
- Coleção de objetos “ordenados” (banco de dados elementar)

◆ Aplicações indiretas

- Estrutura de dados auxiliares para algoritmos
- Componentes de outras estruturas de dados

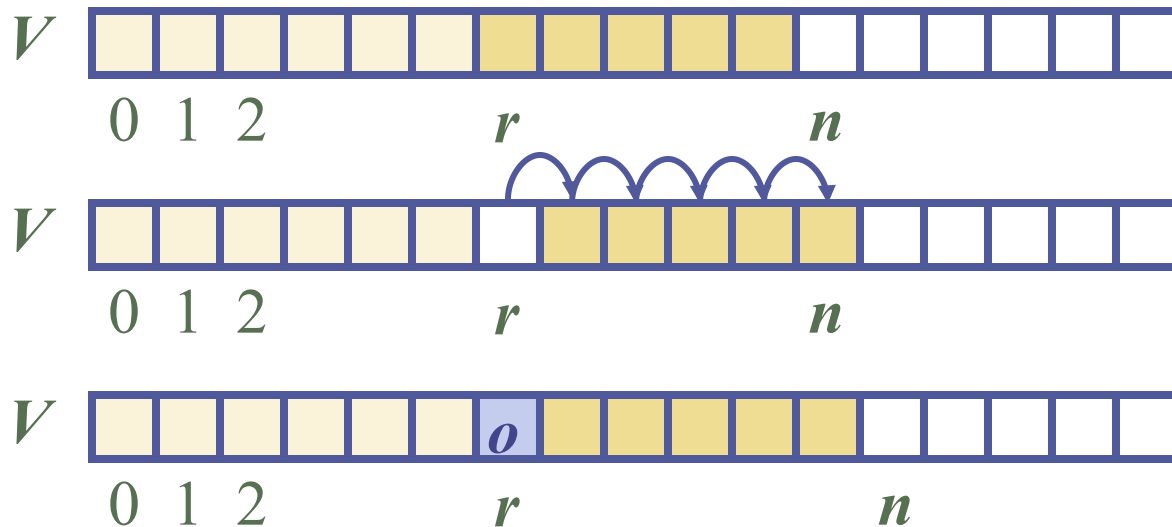
Vetor baseado em arranjo (*array*)

- ◆ Usamos um *array* V de tamanho N
- ◆ Uma variável n mantém o tamanho do vetor (número de elementos armazenados)
- ◆ A operação ***elemAtRank***(r) é implementada em tempo $O(1)$ para retornar $V[r]$



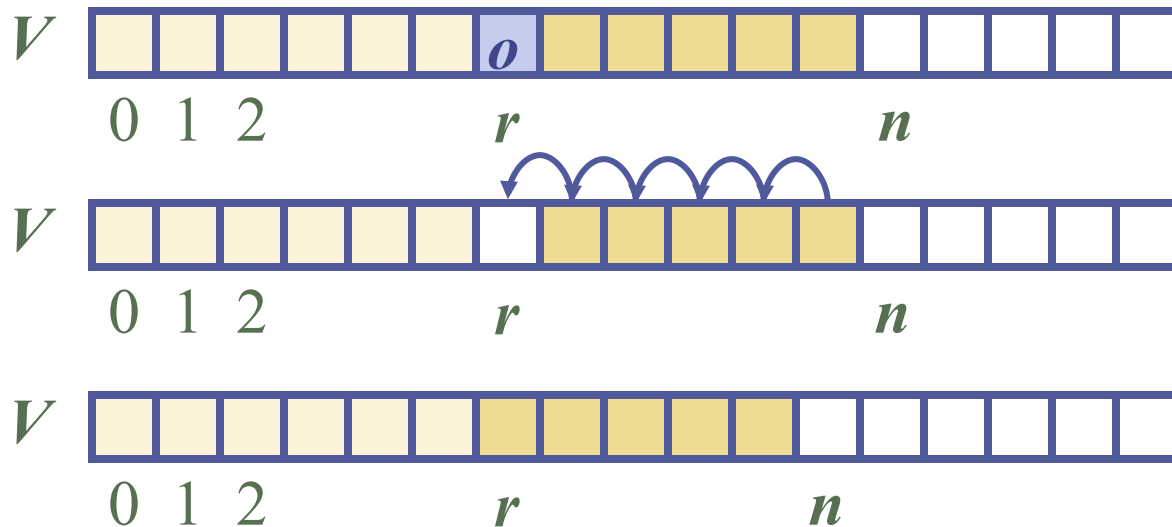
inserção

- ◆ Na operação *insertAtRank*(r, o), precisamos “arrumar espaço” para o novo elemento deslocando para a direita os $n - r$ elementos $V[r], \dots, V[n - 1]$
- ◆ No pior caso ($r = 0$), esta operação roda em tempo $O(n)$



remoção

- ◆ Na operação *removeAtRank*(r), temos que preencher o “buraco” deixado pelo elemento removido deslocando para a esquerda os $n - r - 1$ elementos $V[r + 1], \dots, V[n - 1]$
- ◆ No pior caso ($r = 0$), esta operação roda em tempo $O(n)$



Desempenho

- ◆ Em um Vetor implementado com *arrays*:
 - O espaço usado pela estrutura de dados é $O(n)$
 - *size*, *isEmpty*, *elemAtRank* e *replaceAtRank* rodam em tempo $O(1)$
 - *insertAtRank* e *removeAtRank* roda em tempo $O(n)$
- ◆ Se usarmos um *array* circular, *insertAtRank*(0,o) e *removeAtRank*(0) rodam em tempo $O(1)$
- ◆ Na operação *insertAtRank*, quando o array está cheio, ao invés de disparar uma exceção, podemos substituí-lo por um maior

JAVA

◆ JAVA possui duas classes que fornecem as funcionalidade de vetores:

- `java.util.Vector`
- `java.util.ArrayList`

◆ Correspondência de métodos:

TAD vetor	java
<code>size()</code>	<code>size()</code>
<code>isEmpty()</code>	<code>isEmpty()</code>
<code>elemAtRank(r)</code>	<code>get(r)</code>
<code>replaceAtRank(r,e)</code>	<code>Set(r,e)</code>
<code>insertAtRank(r)</code>	<code>add(r,e)</code>
<code>removeAtRank(r)</code>	<code>remove(r)</code>

TAD Lista

- ◆ O TAD **Lista** modela um sequência de posições armazenando objetos quaisquer
- ◆ Ele estabelece uma relação antes/depois entre posições
- ◆ Métodos genéricos
 - **size()**, **isEmpty()**
- ◆ Métodos de fila:
 - **isFirst(n)**, **isLast(n)**

Métodos para acessar:

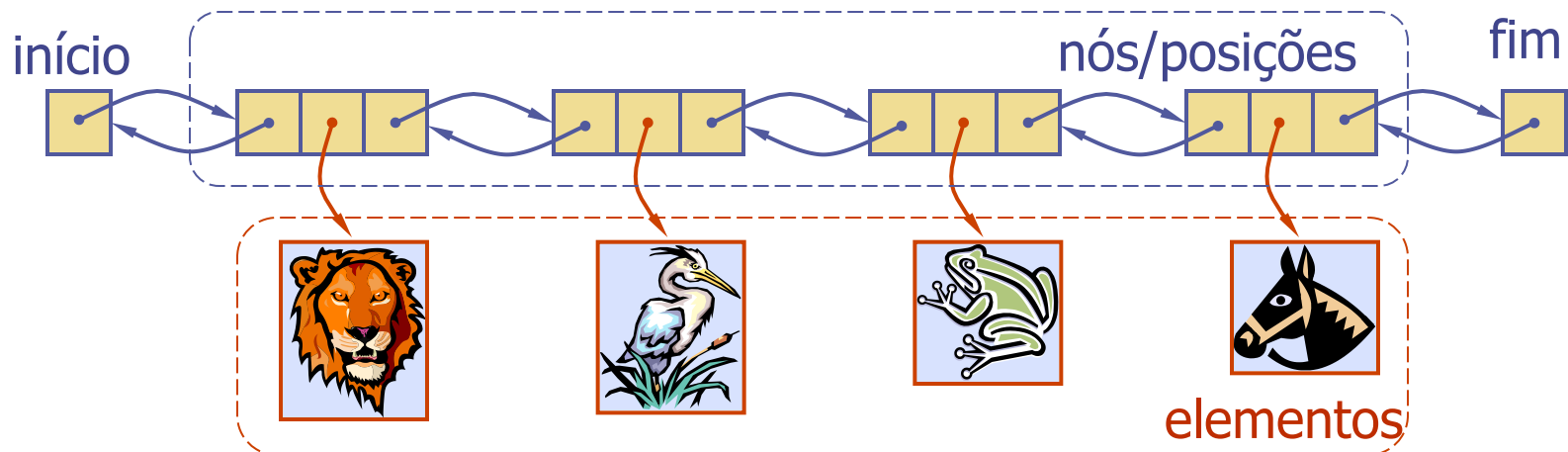
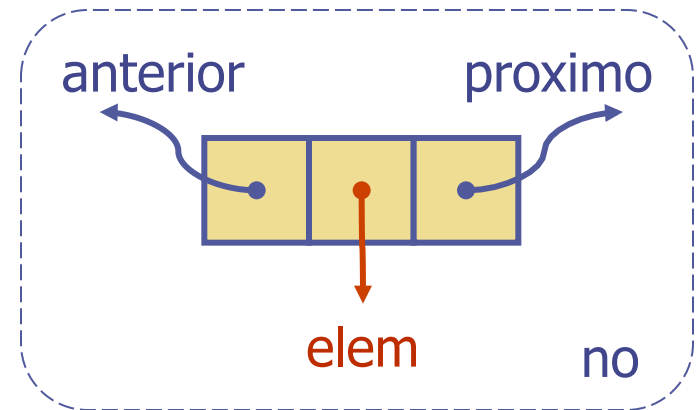
- **first()**, **last()**
- **before(p)**, **after(p)**

◆ Métodos para atualizar:

- **replaceElement(n, o)**, **swapElements(n, q)**
- **insertBefore(n, o)**, **insertAfter(n, o)**,
- **insertFirst(o)**, **insertLast(o)**
- **remove(n)**

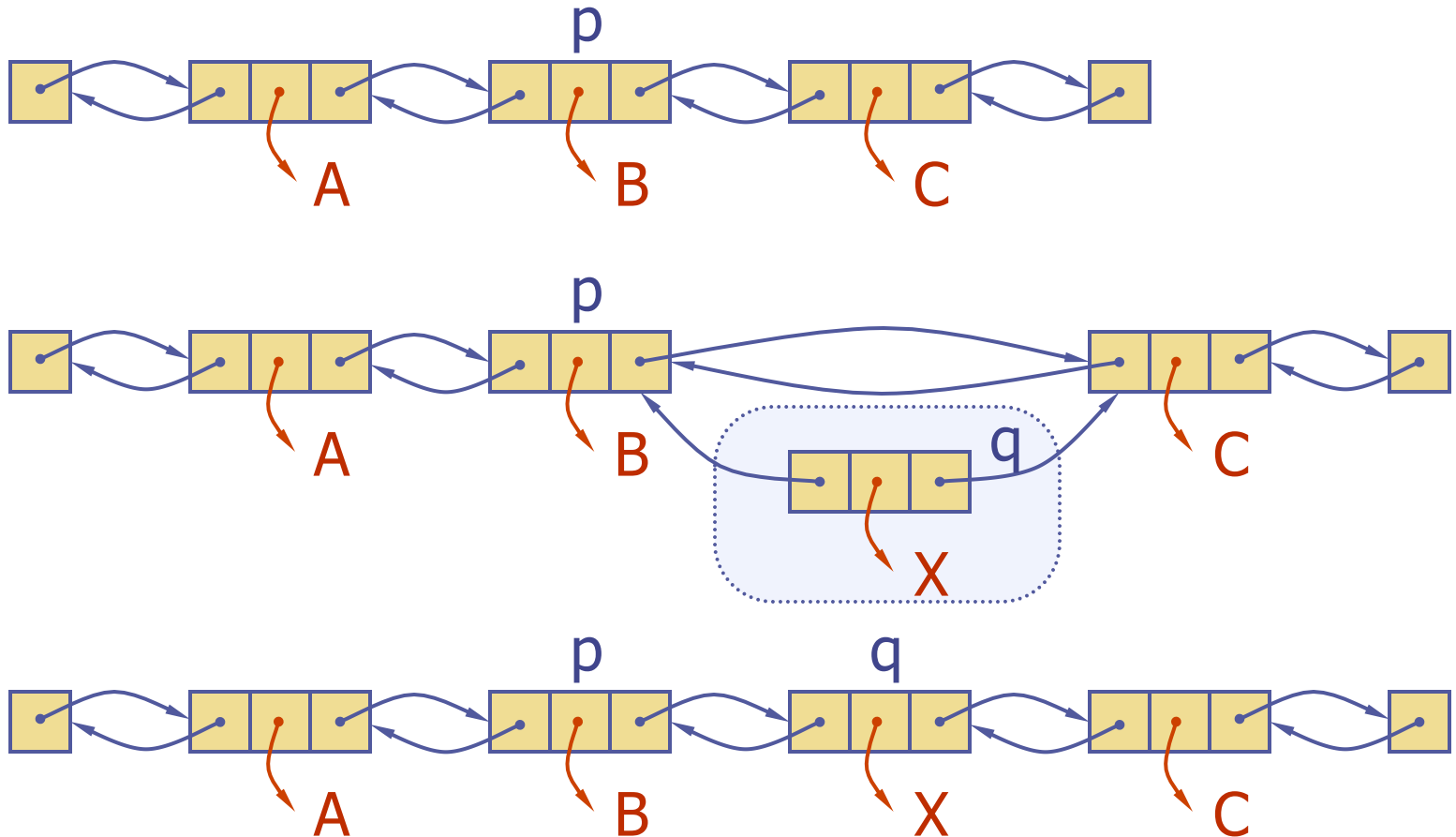
Listas duplamente encadeadas

- ◆ Uma lista duplamente encadeada provê uma implementação natural do TAD Lista
- ◆ Nós implementam Posições e armazenam:
 - elemento
 - referência ao nó anterior
 - referência ao nó posterior
- ◆ Nós especiais para início e fim



Inserção

- ◆ A operação **insertAfter**(p, X), que retorna um posição q



Algoritmo de inserção

Algoritmo insertAfter(p, e):

Criar novo nó q

$q.setElement(e)$

$q.setPrev(p)$ $\{v \text{ referencia seu anterior}\}$

$q.setNext(p.getNext())$ $\{v \text{ referencia}$
seu posterior}

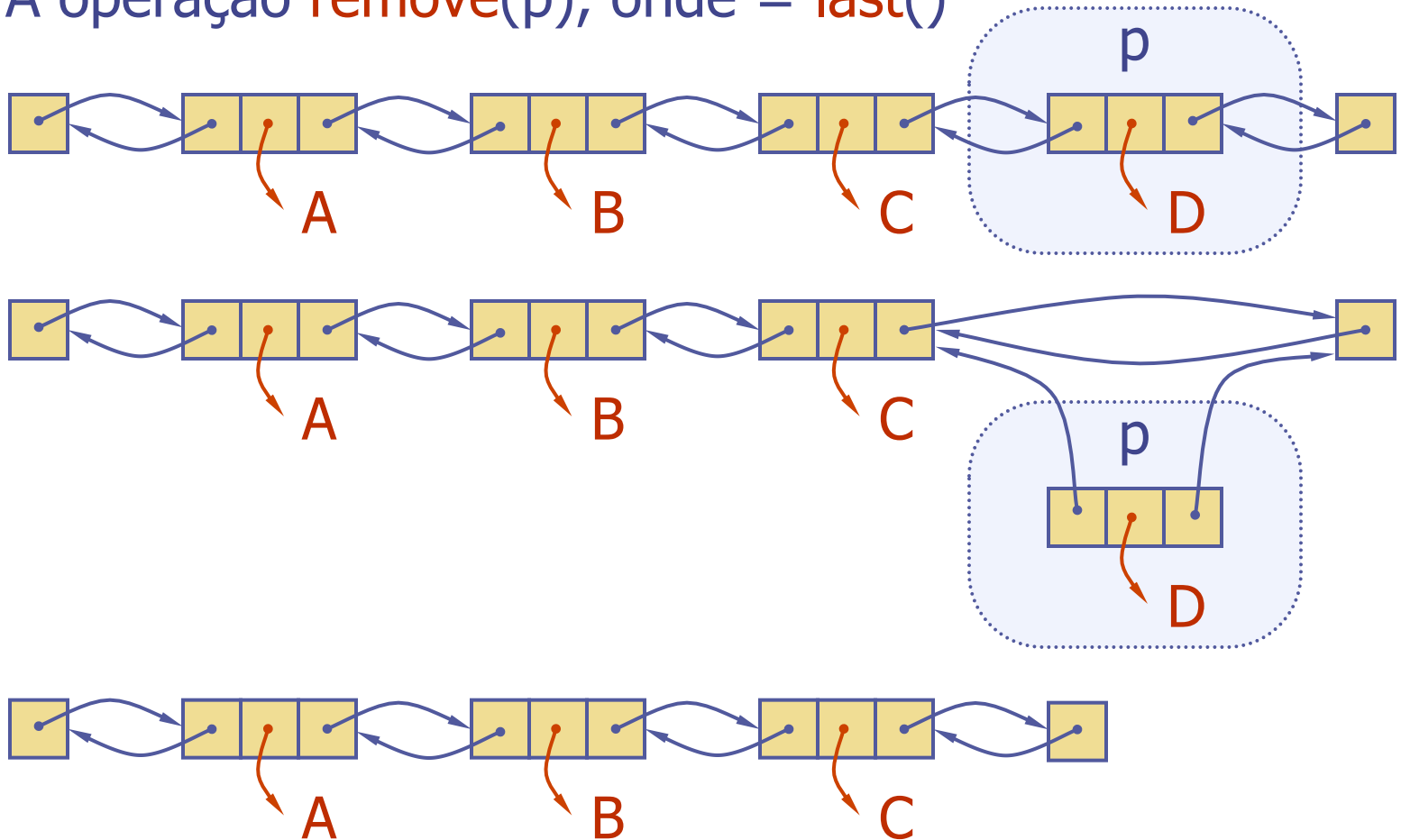
$(p.getNext()).setPrev(q)$ $\{\text{anterior do próximo}$
de p agora é $v\}$

$p.setNext(q)$ $\{\text{próximo de } p \text{ é o novo nó } v\}$

return q $\{A \text{ posição do elemento } e\}$

Remoção

◆ A operação **remove(p)**, onde **p = last()**



Algoritmo de remoção

Algoritmo remove(p):

$t = n.\text{element}$ {Variável temporária para
armazenar valor de retorno}

$(p.\text{getPrev}()).\text{setNext}(p.\text{getNext}())$
 {“desreferenciando” n }

$(p.\text{getNext}()).\text{setPrev}(p.\text{getPrev}())$

$p.\text{setPrev}(\text{null})$ {invalidando em n }

$p.\text{setNext}(\text{null})$

return t

Desempenho

- ◆ A implementação do TAD Lista usando uma lista duplamente encadeada:
 - O espaço usado pela lista com n elementos é $O(n)$
 - O espaço usado por cada posição na lista é $O(1)$
 - Todas as operações na lista são executadas em tempo $O(1)$
 - A operação `element()` do TAD Posição executa em tempo $O(1)$

TAD sequência

- ◆ O TAD **Sequencia** é a união de Vetor e Lista
- ◆ Elementos podem ser acessados por:
 - colocação, ou
 - posição
- ◆ Métodos genéricos:
 - **size()**, **isEmpty()**
- ◆ Métodos de Vetor:
 - **elemAtRank(r)**,
replaceAtRank(r, o),
insertAtRank(r, o),
removeAtRank(r)

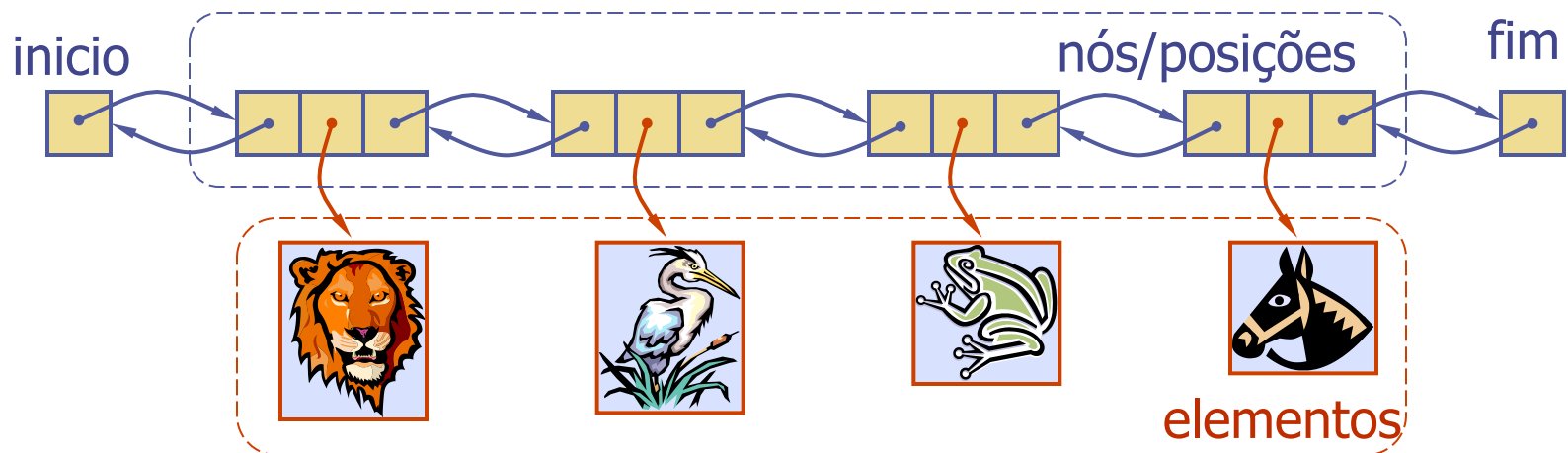
- ◆ Métodos de Lista:
 - **first()**, **last()**,
before(n), **after(n)**,
replaceElement(n, o),
swapElements(n, q),
insertBefore(n, o),
insertAfter(n, o),
insertFirst(o),
insertLast(o),
remove(n)
- ◆ Métodos “ponte”:
 - **atRank(r)**, **rankOf(n)**

Aplicações de Sequências

- ◆ O TAD sequência é uma estrutura de dados básica de propósito geral para armazenar um coleção ordenada de elementos
- ◆ Aplicações diretas
 - Substituto genérico para Pilha, Fila, Deque, Vetor ou Lista
 - Pequenos Bancos de dados (e.g., Agenda de endereços)
- ◆ Aplicações indiretas:
 - Blocos de construção para estruturas de dados mais complexas

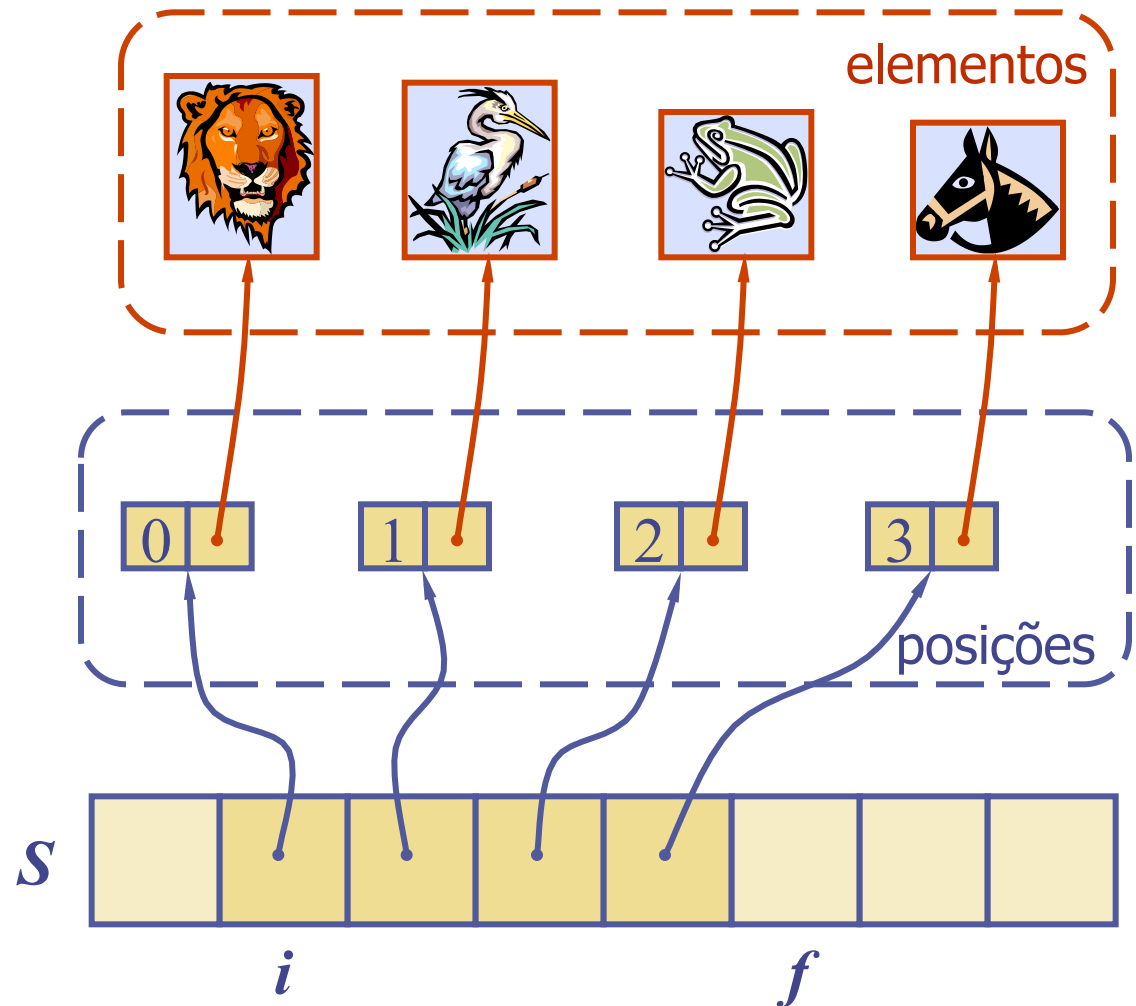
Implementação com Lista encadeada

- ◆ Uma lista duplamente ligada provê uma implementação razoável do TAD Sequência
- ◆ Nós implementam Nós e armazenam:
 - elemento
 - referência ao nó anterior
 - referência ao nó posterior
- ◆ Nós especiais de início e fim
- ◆ Position-based methods run in constant time
- ◆ Rank-based methods require searching from header or trailer while keeping track of ranks; hence, run in linear time



Implementação baseada em *array*

- ◆ Usamos um *array* circular para armazenar posições
- ◆ Uma posição armazena:
 - Elemento
 - Colocação (rank)
- ◆ Índices i and f armazenam as primeira e última posições



atRank(r) – com Lista encadeada

```
public Nó AtRank (int rank) {  
    Nó node;  
    If (rank <= size()/2) {  
        node = header.getNext();  
        for(int i=0; i < rank; i++)  
            node = node.getNext();  
    }else{  
        node = trailer.getPrev();  
        for(int i=0; i < size()-rank-1 ; i++)  
            node = node.getPrev();  
    }  
    return node;  
}
```

rankOf(n)-com Lista encadeada

```
public int rankOf(Nó no) {  
    Nó n =header.getNext();  
    int r = 0;  
    while(n != no && n != trailer) {  
        n = n.getNext();  
        r++;  
    }  
    return r;  
}
```

rankOf(n)-com Lista encadeada

```
public int rankOf(Nó no) {  
    Nó n =header.getNext();  
    int r = 0;  
    while(n != no && n != trailer) {  
        n = n.getNext();  
        r++;  
    }  
    return r;  
}
```

Implementação de Sequência

Operação	Array	Lista
size, isEmpty	1	1
atRank, rankOf, elemAtRank	1	<i>n</i>
first, last, before, after	1	1
replaceElement, swapElements	1	1
replaceAtRank	1	<i>n</i>
insertAtRank, removeAtRank	<i>n</i>	<i>n</i>
insertFirst, insertLast	1	1
insertAfter, insertBefore	<i>n</i>	1
remove	<i>n</i>	1

Coleções e iteradores

- ◆ JAVA possui uma API específica para coleções de objetos
 - Vector e ArrayList são exemplos de implementações desta API
- ◆ Um iterador (*iterator*) abstrai o processo de percorrer uma coleção de elementos
- ◆ Métodos do TAD ObjectIterator:
 - object **object()**
 - boolean **hasNext()**
 - object **nextObject()**
 - **reset()**
- ◆ Estende o conceito de posição adicionando a capacidade de travessia
- ◆ Implementação com *array* ou lista ligada
- ◆ Um iterador é, tipicamente, associado com outra estrutura de dados
- ◆ Podemos aumentar os TAD Pilha, Fila, Deque, Vetor, Lista e Sequencia com o método
 - ObjectIterator **elements()**
- ◆ Duas noções de iteradores:
 - estático: congela o conteúdo da estrutura de dados em um dado momento
 - Dinâmico segue as mudanças da estrutura de dados