2-Listas Encadeadas

BCC - Estruturas de Dados

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br

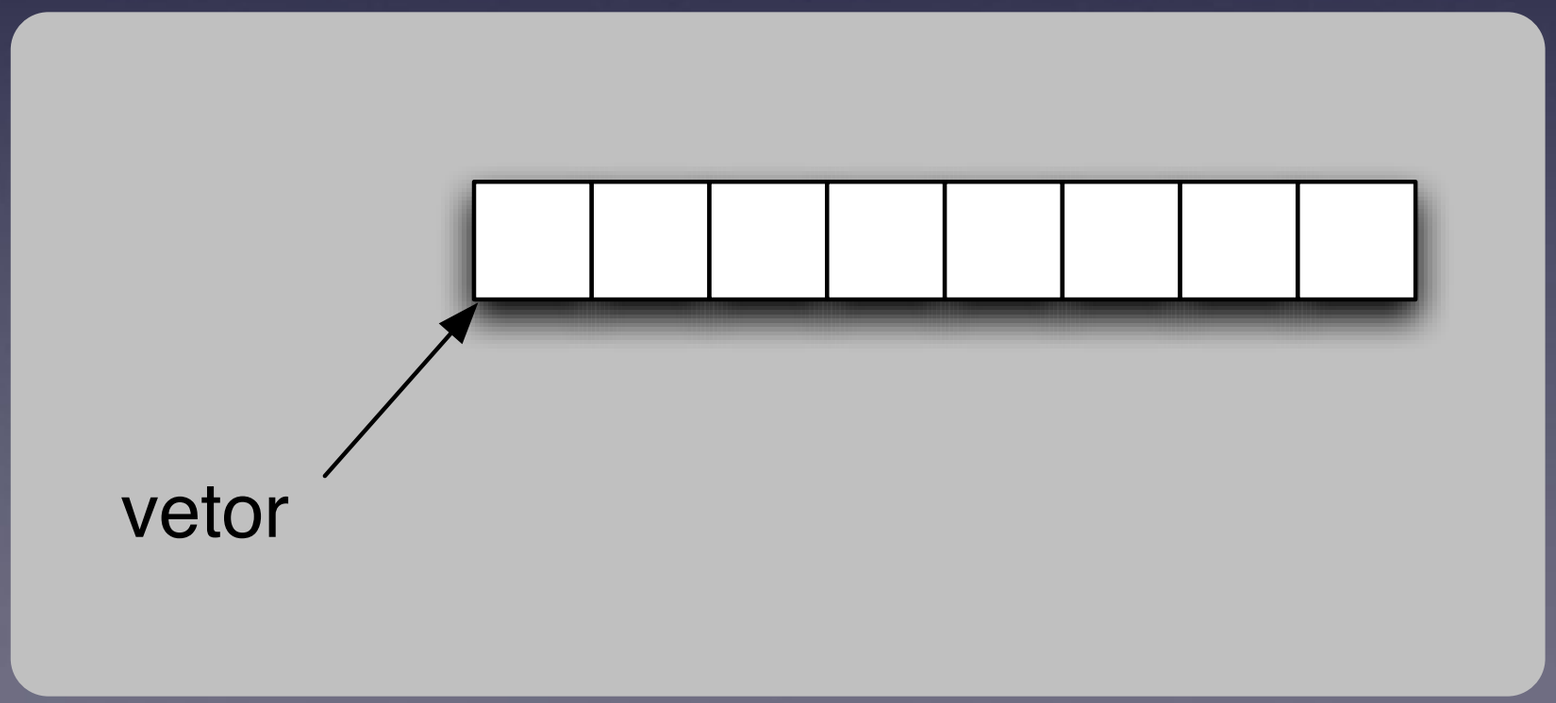
Blumenau, 2022

Tópicos

- Motivação
- Listas encadeadas
- Implementações recursivas
- Listas circulares
- Listas duplamente encadeadas

Motivação

- Vetor
  - ocupa espaço contíguo de memória
  - permite acesso randômico aos elementos
  - deve ser dimensionado com número máximo de elementos

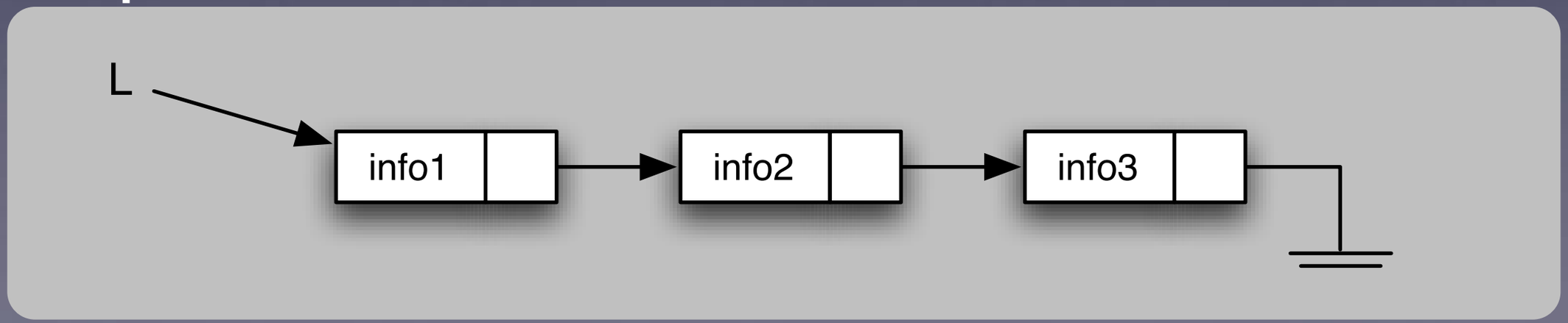


Motivação

- Estruturas de dados dinâmicas
  - crescem (ou decrescem) à medida que elementos são inseridos ou removidos
  - Exemplo: listas encadeadas - amplamente usadas para implementar outras estruturas de dados

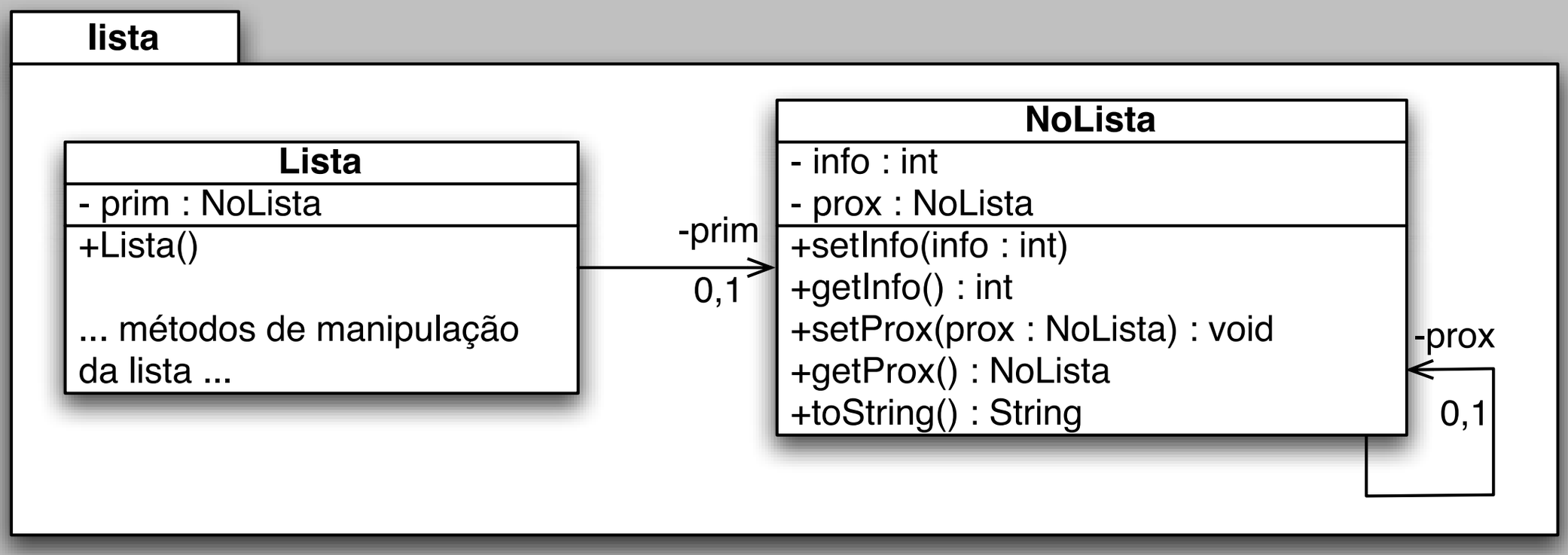
Listas encadeadas

- seqüência encadeada de elementos, chamados de *nós da lista*
- *nó da lista* é representado por dois campos:
  - a informação armazenada e
  - o ponteiro para o próximo elemento da lista
- a lista é representada por um ponteiro para o primeiro nó
- o ponteiro do último elemento é null



Listas encadeadas

Exemplo: lista encadeada armazenando valores inteiros



Listas encadeadas

Exemplo: lista encadeada armazenando valores inteiros

- Classe NoLista: representa cada nó

```
public class NoLista {
    private int info;
    private NoLista prox;
}
```

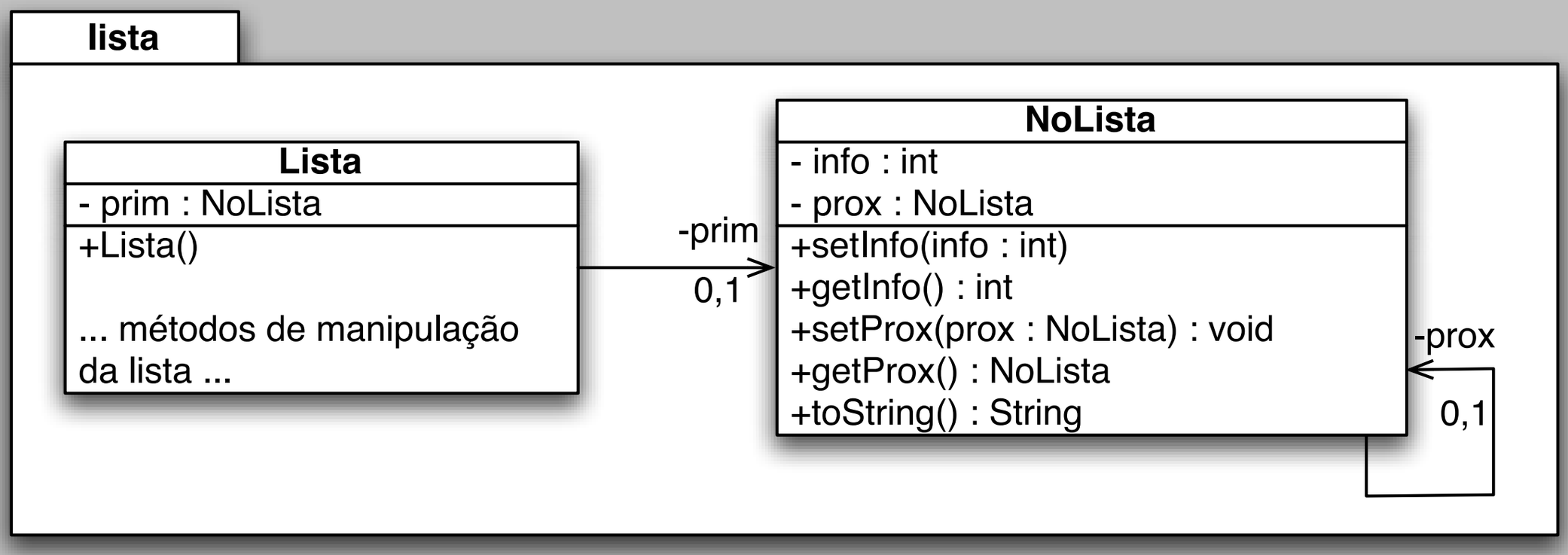
Obs: NoLista é uma classe auto-referenciada, pois o atributo *prox* é uma referência para uma próxima instância da mesma classe

- Classe NoLista: setters, getters e toString

```
public void setInfo(int info){
    this.info = info;
}

public int getInfo(){
    return this.info;
}

public void setProx(NoLista prox){
    this.prox = prox;
}

public NoLista getProx(){
    return this.prox;
}

public String toString(){
    return (getInfo()+" ");
}
```

Listas encadeadas

Exemplo: lista encadeada armazenando valores inteiros



Listas encadeadas

- Classe Lista: referência para o primeiro elemento e métodos de manipulação da lista

```
public class Lista {
    private NoLista prim;
}
```

Obs: uma lista encadeada é representada pela referência para o seu primeiro elemento, que é um objeto da classe NoLista

Listas encadeadas

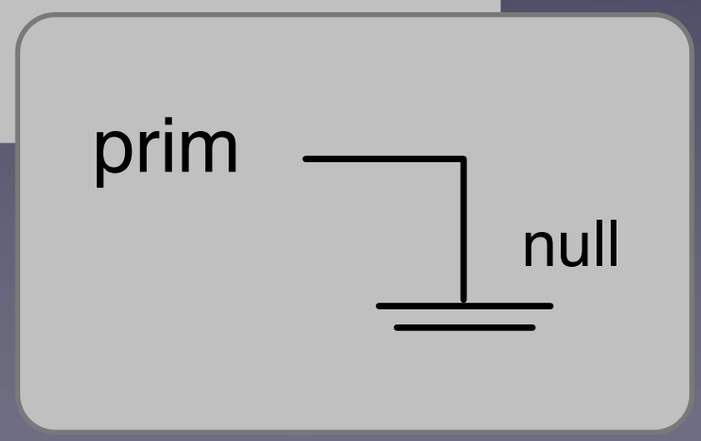
Exemplo: Método construtor da lista

- cria uma lista vazia, representada pela referência para *null*

```
// método construtor, inicializa uma lista vazia
```

Algoritmo: Lista()

$prim \leftarrow null;$

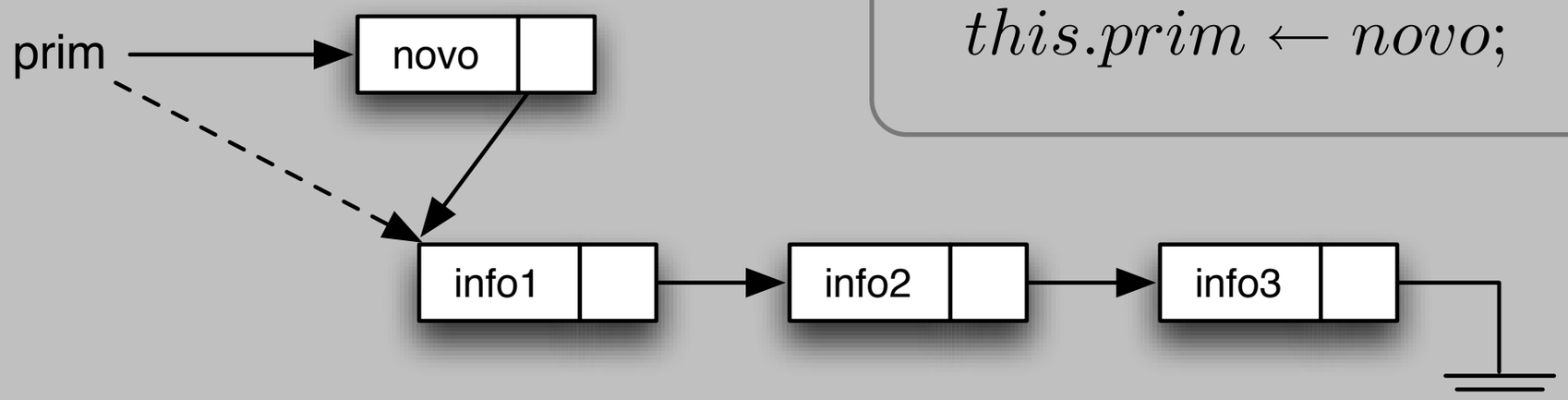


Exemplo: método de inserção

- instancia novo objeto NoLista
- encadeia o elemento no início da lista existente

Algoritmo: insere(int info)

$NoLista\ novo \leftarrow new\ NoLista();$
$novo.info \leftarrow info;$
$novo.prox \leftarrow prim;$
$this.prim \leftarrow novo;$



Exemplo: método para imprimir valores da lista

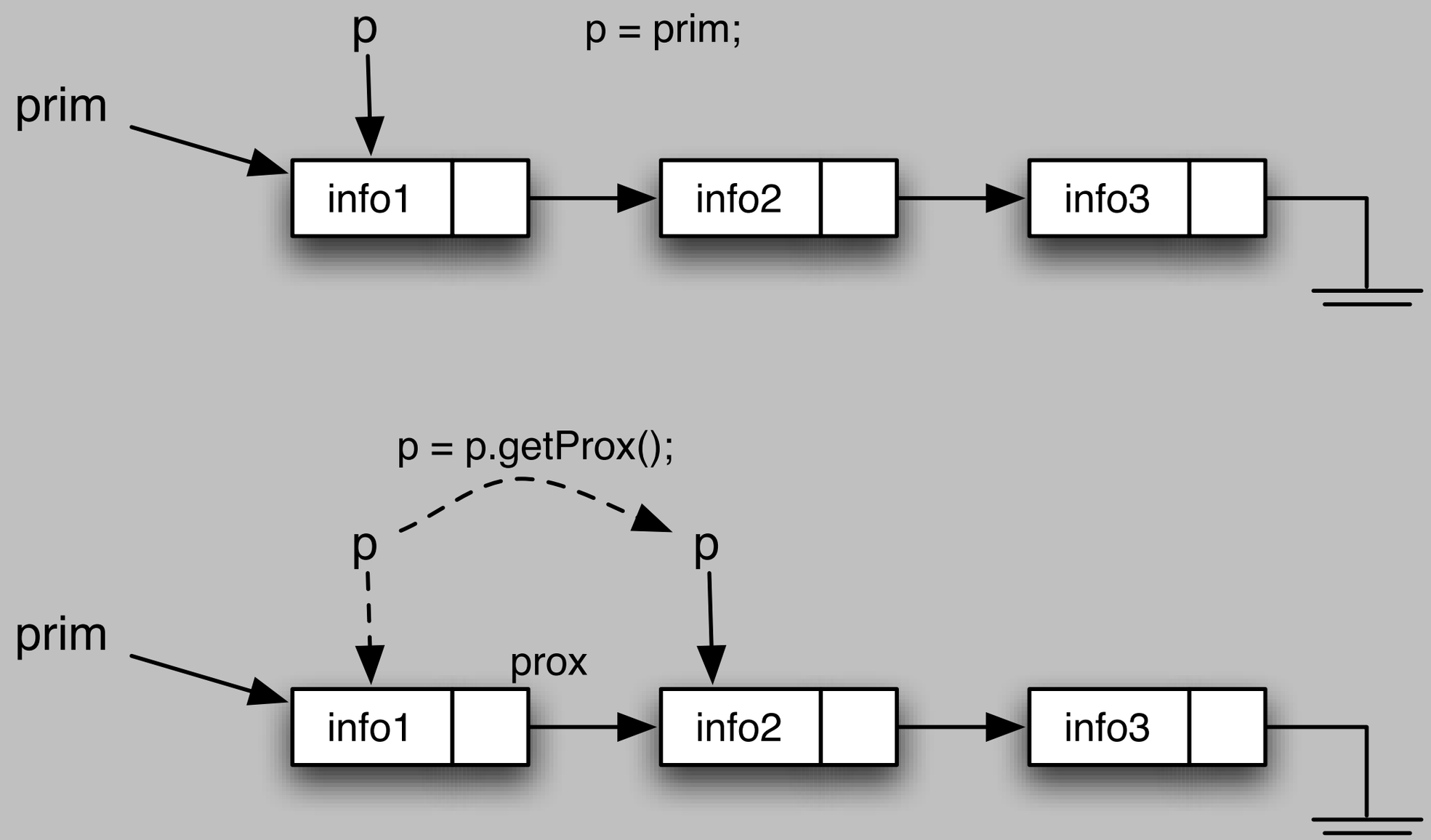
- imprime os valores armazenados nos nós

Algoritmo: imprime()

$NoLista\ p \leftarrow prim;$
enquanto $p \neq null$ **faça**
    $print(p.info);$
    $p \leftarrow p.prox;$

variável auxiliar p: referência para armazenar o endereço de cada elemento. Dentro do loop, aponta para cada um dos elementos da lista

Variável auxiliar p



Exemplo: método para verificar se a lista está vazia

- retorna *true* se estiver vazia, caso contrário retorna *false*

```
public boolean vazia(){}
```

Algoritmo: vazia()

se $prim == null$ **então**
  retorna $verdadeiro$;
senão
  retorna $falso$;

Exemplo: método buscar elemento na lista

- recebe a informação referente ao elemento a pesquisar
- retorna o endereço do nó da lista que armazena o valor, ou null, caso o elemento não seja encontrado na lista

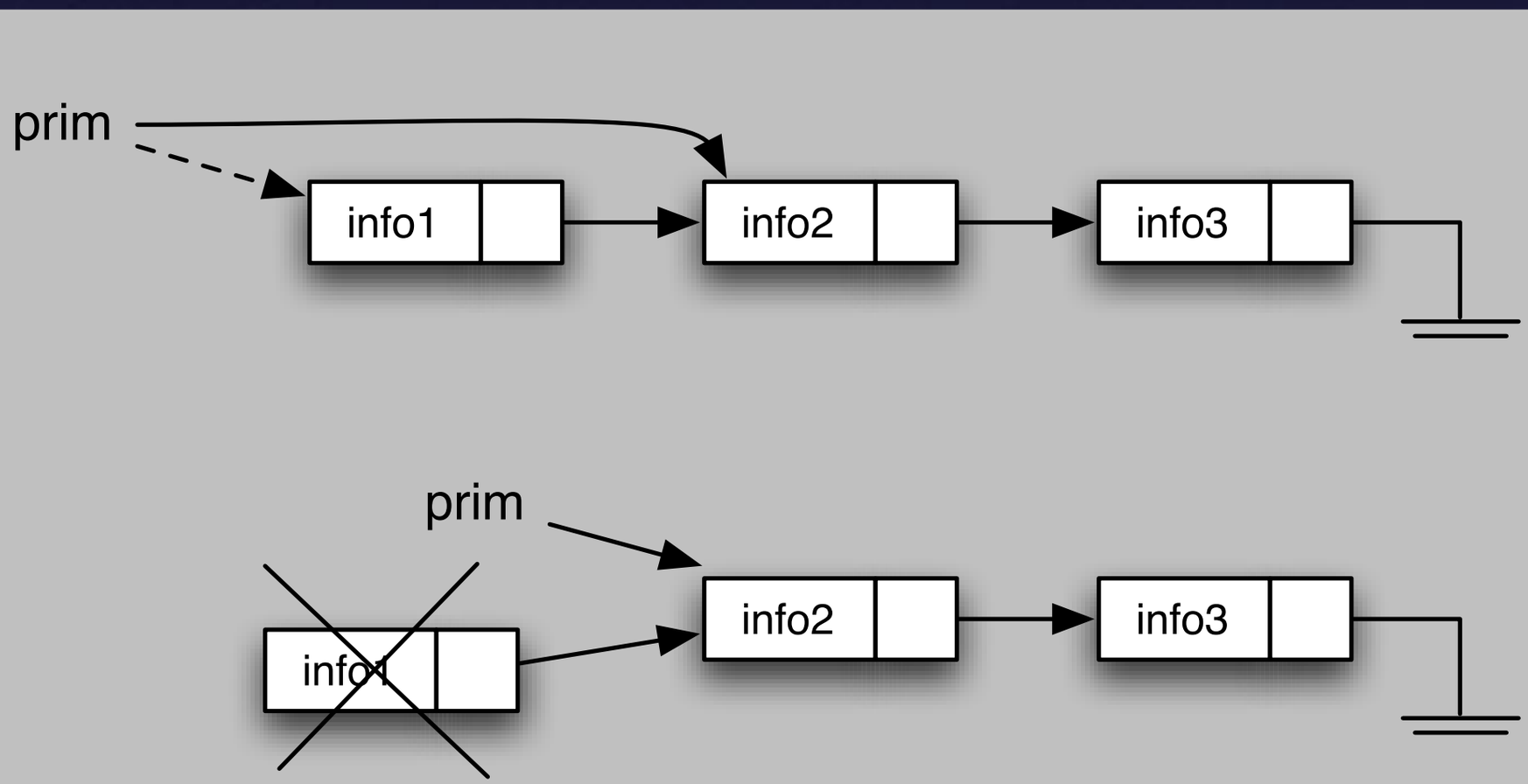
```
public NoLista busca(int v) {}
```

Algoritmo: busca(int v)

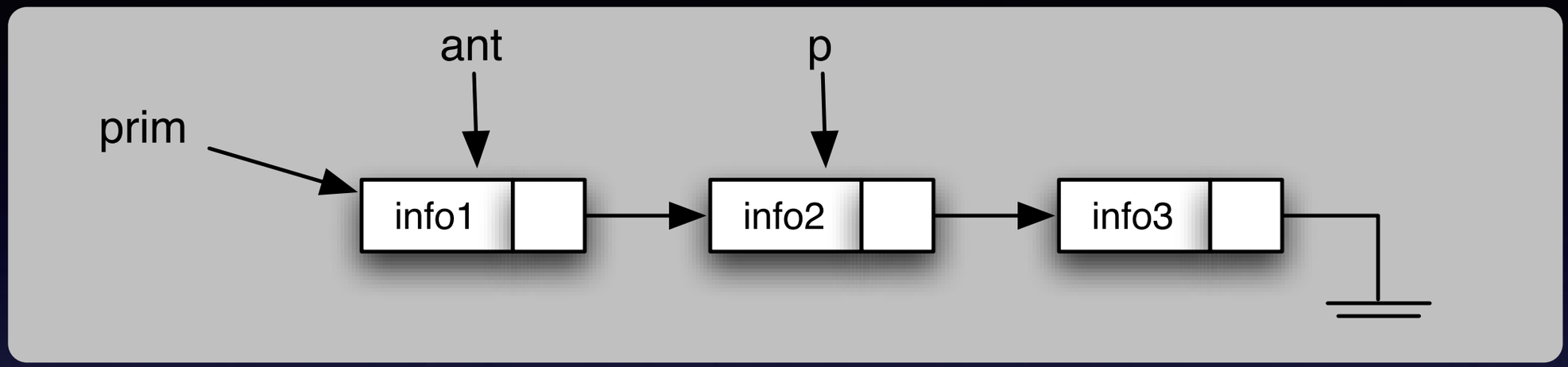
NoLista $p \leftarrow prim$;
enquanto $p \neq null$ **faça**
  se $p.info == v$ **então**
    retorna p;
  $p \leftarrow p.prox$;
retorna *null*;

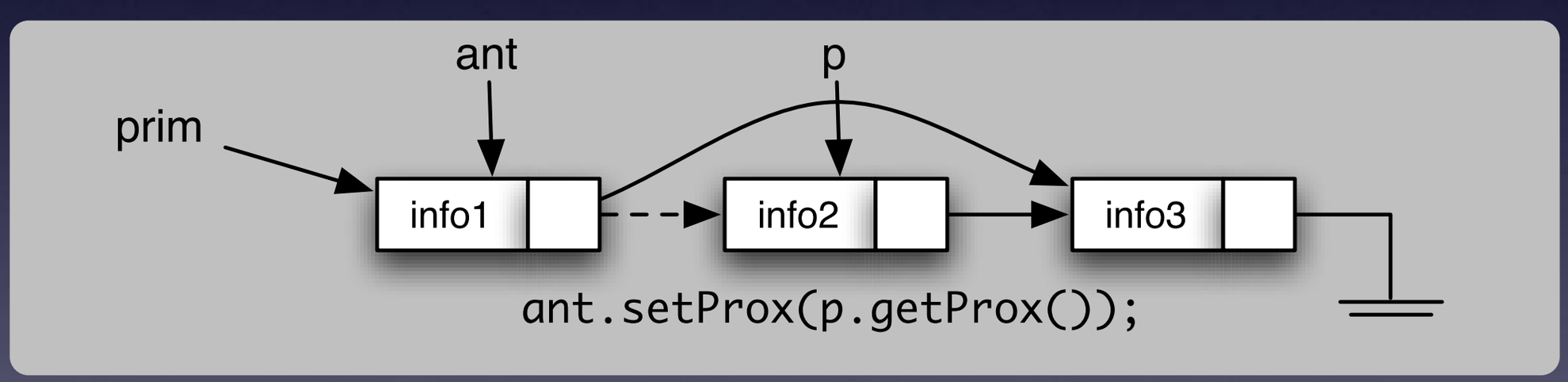
Exemplo: método para retirar um elemento

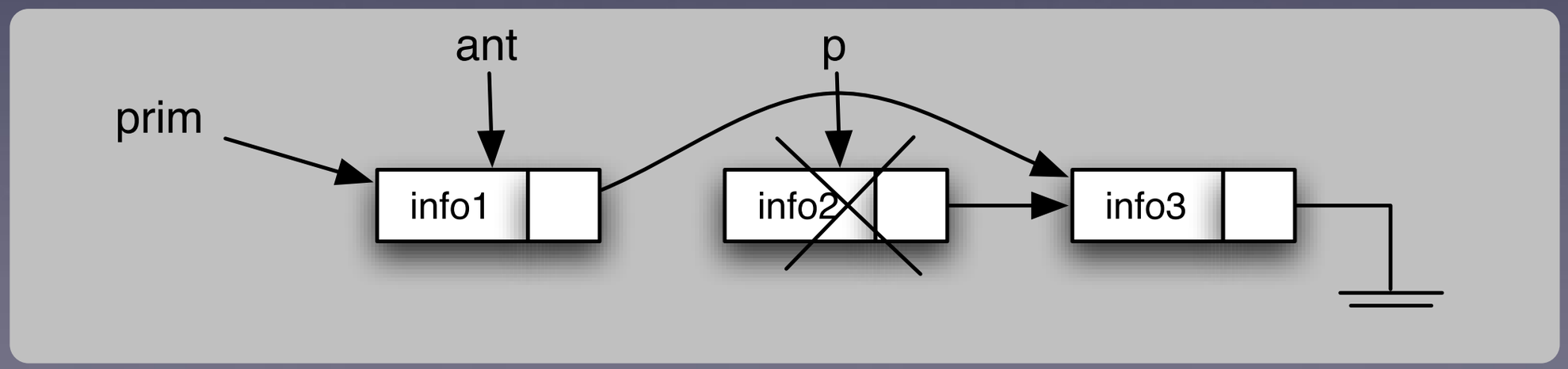
- recebe como entrada o valor do elemento a retirar
- se o elemento a ser removido é o primeiro, é necessário atualizar *prim*



- caso contrário, apenas remove o elemento da lista







```java
public void retira(int v) {}
```

Algoritmo: retira(int v)

$NoLista\ ant \leftarrow null;$
$NoLista\ p \leftarrow prim;$

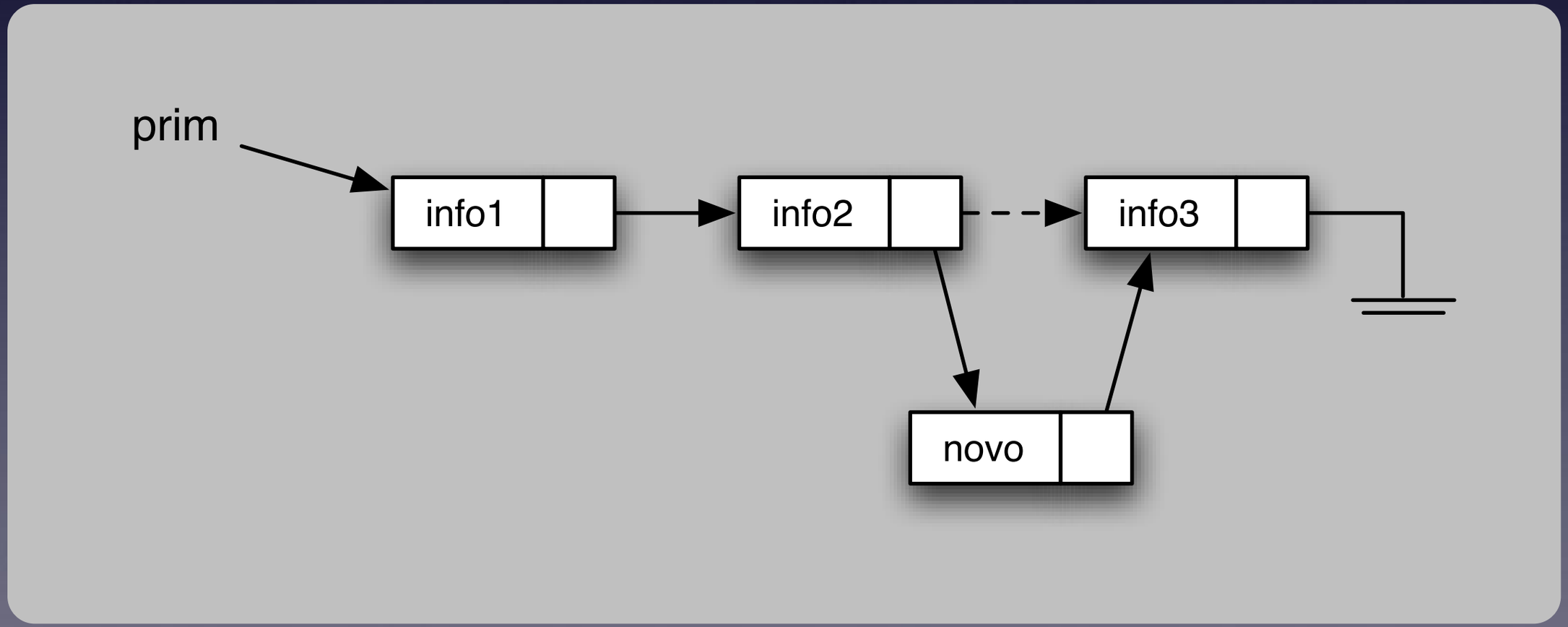
// procura elem, guardando anterior
enquanto $(p \neq null)$ **e** $(p.info \neq v)$ **faça**
  $ant \leftarrow p;$
  $p \leftarrow p.prox;$

// caso não achou elem...
se $p == null$ **então**
  retorna;

// retira elemento
se $ant == null$ **então**
  $this.prim \leftarrow p.prox;$
senão
  $ant.prox \leftarrow p.prox;$

Manutenção da lista ordenada

- o método de inserção percorre os elementos da lista até encontrara posição correta para inserção do novo nó



```java
// insere elementos em ordem crescente
public void insereOrdenado(int v) {
```

Algoritmo: insereOrdenado(int v)

$NoLista\ ant \leftarrow null;$
$NoLista\ novo;$
$NoLista\ p \leftarrow prim;$

// procura posicao correta, guardando anterior
enquanto $(p \neq null)$ **e** $(p.info < v)$ **faça**
  $ant \leftarrow p;$
  $p \leftarrow p.prox;$

$novo \leftarrow new\ NoLista();$
$novo.info \leftarrow v;$

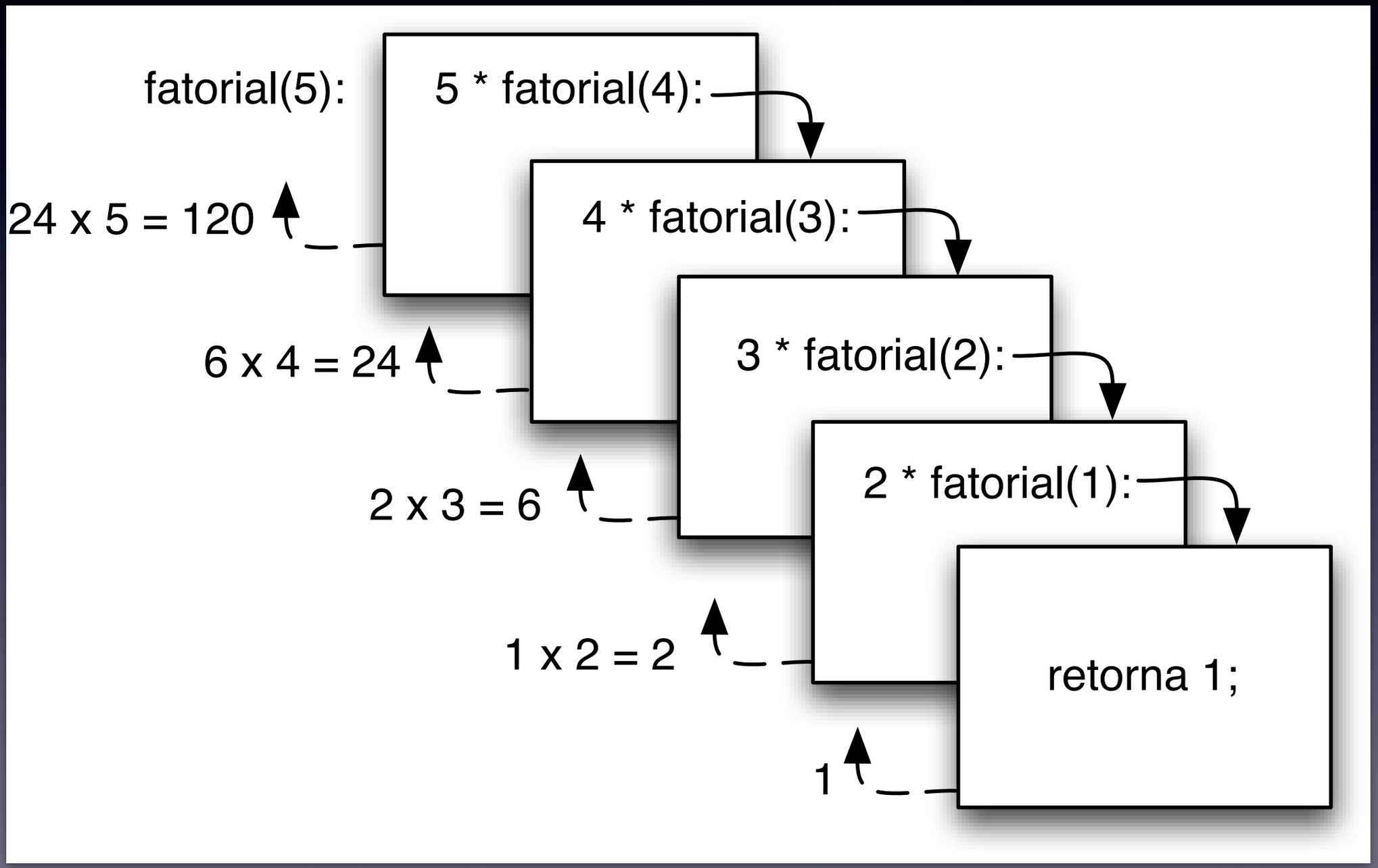
se $ant == null$ **então**
  $novo.prox \leftarrow prim;$
  $prim \leftarrow novo;$
senão
  $novo.prox \leftarrow ant.prox;$
  $ant.prox \leftarrow novo;$

Recursividade

- Uma definição recursiva é usada para definir um objeto em termos de si próprio (Aczel 1977).
- Uma definição recursiva de uma função define valores das funções para algumas entradas em termos dos valores da mesma função para outras entradas
- Exemplo: Função Fatorial
    - 0! = 1
    - n! = n.(n-1)!

P. Aczel (1977), "An introduction to inductive definitions", Handbook of Mathematical Logic, J. Barwise (ed.), ISBN 0-444-86388-5

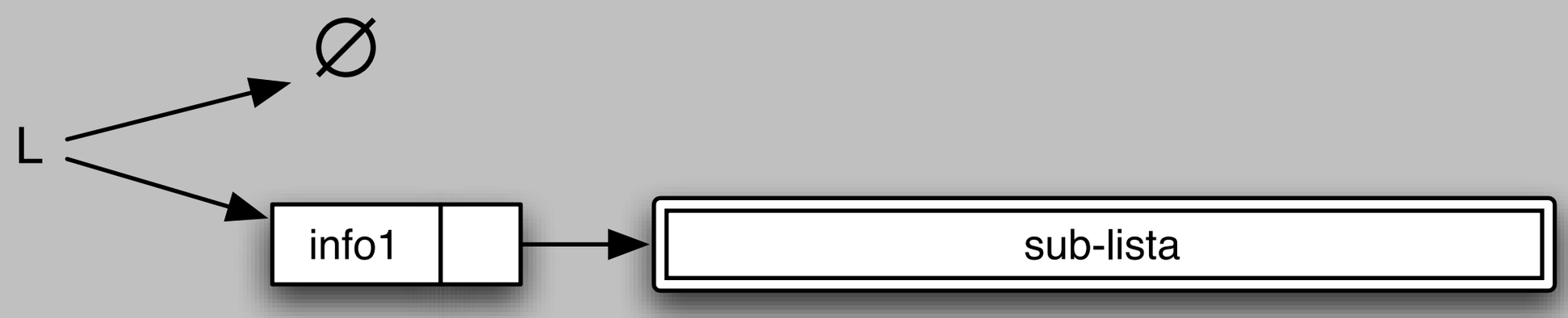
Recursividade



Implementação recursiva

Definição recursiva: uma lista é...

- uma lista vazia; ou
- um nó seguido de uma (sub-)lista



Implementação recursiva

Exemplo: imprimir lista

- se a lista for vazia, não imprima nada;
- caso contrário,
  - imprima a informação associada ao primeiro nó, dada por `no.getInfo()`;
  - imprima a sub-lista, dada por `no.prox` chamando recursivamente a função

Exemplo: método imprimeRecursivo

```
public void imprimeRecursivo (){ }
private void imprimeRecursivoAux(NoLista l){ }
```

Algoritmo: public void imprimeRecursivo()

$imprimeRecursivoAux(prim)$;

Algoritmo: private void imprimeRecursivoAux(NoLista l)

se $l \neq null$ **então**
  $imprime(l.info)$;
  $imprimeRecursivoAux(l.prox)$;

Implementação recursiva

Exemplo: retirar elemento

- retire o elemento, se ele for o primeiro da lista (ou da sub-lista);
- caso contrário, chame o método recursivamente para retirar o elemento da sub-lista

```
public void retiraRecursivo(int v){ }

private NoLista retiraRecursivoAux(NoLista l, int v) { }
```

Algoritmo: public void retiraRecursivo(int v)

$prim \leftarrow retiraRecursivoAux(prim, v)$;

Algoritmo 1.10: Retira recursivo

Algoritmo: private NoLista retiraRecursivoAux(NoLista l, int v)

se $l \neq null$ **então**
  se $l.info == v$ **então**
    $l \leftarrow l.prox$;
  senão
    $l.prox \leftarrow retiraRecursivoAux(l.prox, v)$;
retorna l;

Algoritmo 1.11: Retira recursivo auxiliar

Implementação recursiva

Exemplo: igualdade de duas listas (não recursiva)

- percorre as dias listas usando duas referências auxiliares, se duas informações forem diferentes, as listas são diferentes;
- ao terminar de percorrer uma das listas (ou as duas), se duas referências auxiliares são nulas, as duas listas tem o mesmo número de elementos e são iguais

Exemplo: método igual

Algoritmo: public boolean igual(Lista l)

NoLista p1 $\leftarrow$ *this.prim*;
NoLista p2 $\leftarrow$ *l.prim*;

enquanto $(p1 \neq null)$ **e** $(p2 \neq null)$ **faça**
  se $(p1.info \neq p2.info)$ **então**
    retorna *falso*;
  $p1 \leftarrow p1.prox$;
  $p2 \leftarrow p2.prox$;

se $p1 == p2$ **então**
  retorna *verdadeiro*;
senão
  retorna *falso*;

Algoritmo 1.13: Testa igualdade de duas listas (iterativo)

Implementação recursiva

Exemplo: igualdade de duas listas

- se as duas listas dadas são vazias, então são iguais;
- se não forem ambas vazias, mas uma delas é vazia, então são diferentes;
- se ambas não forem vazias, teste:
  - se as informações associadas aos primeiros nós são iguais e
  - se as sub-listas são iguais

Exemplo: método igual recursivo

Algoritmo: public boolean igualRecursivo(Lista l)

retorna *igualRecursivoAux(this.prim, l.prim)*;

Algoritmo 1.14: Igualdade recursivo

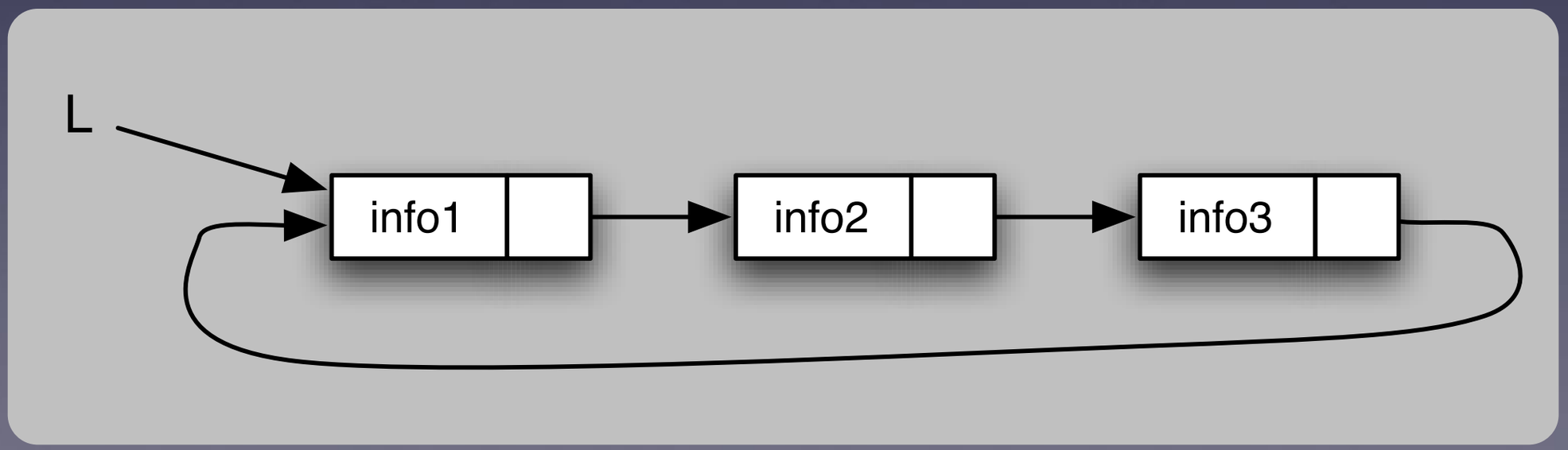
Algoritmo: private boolean igualRecursivoAux(NoLista l1, NoLista l2)

```
se (l1 == null) e (l2 == null) então
    retorna verdadeiro;
senão
    se (l1 == null) ou (l2 == null) então
        retorna falso;
    senão
        retorna ((l1.info == l2.info) e
        igualRecursivoAux(l1.prox, l2.prox));
```

Algoritmo 1.15: Igualdade recursivo (auxiliar)

Listas Circulares

- o último elemento tem como seu próximo o primeiro elemento da lista, formando um ciclo
- a lista pode ser representada por uma referência para um nó inicial qualquer da lista



Listas circulares

Exemplo: método para imprimir

- visita todos os nós a partir da referência para o inicial até alcançar novamente este mesmo elemento
- se a lista é vazia, a referência para o nó inicial é nula

Exemplo: método imprime lista circular

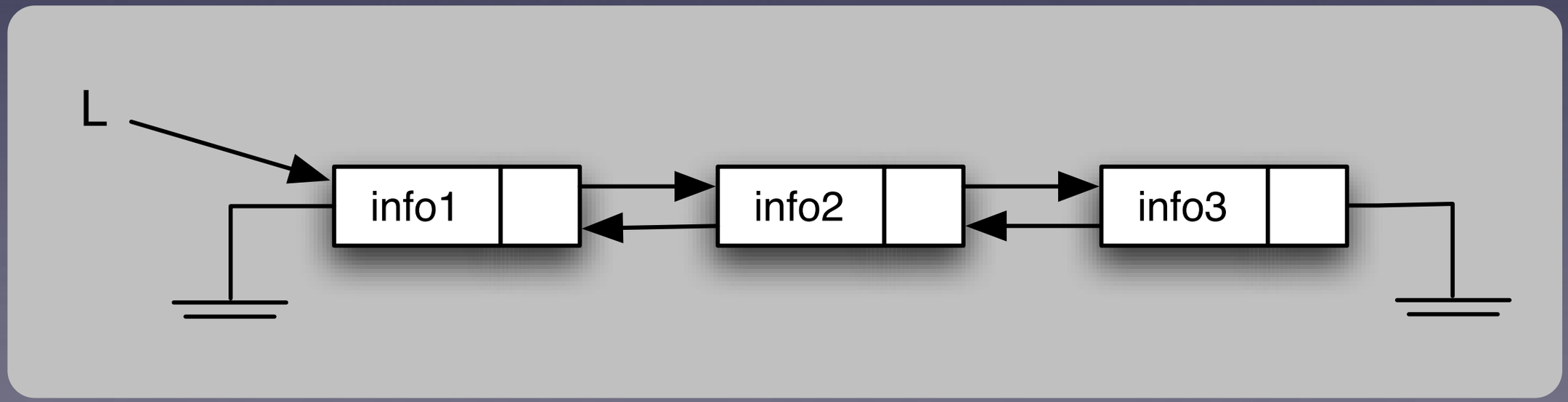
Algoritmo: imprimeCircular()

```
NoLista p ← prim;
se p ≠ null então
    repita
        print(p.info);
        p ← p.prox;
    até p == this.prim ;
```

Algoritmo 1.16: Imprime lista circular

Listas Duplamente Encadeadas

- cada nó possui uma referência para o próximo nó e para o nó anterior;
- dado um nó, é possível acessar diretamente seu próximo e seu anterior;
- dada uma referência para o último elemento da lista, é possível percorrer a lista em ordem inversa;



Listas Duplamente Encadeadas

Exemplo: lista duplamente encadeada armazenando valores inteiros

- Classe NoListaDupla: representa cada nó

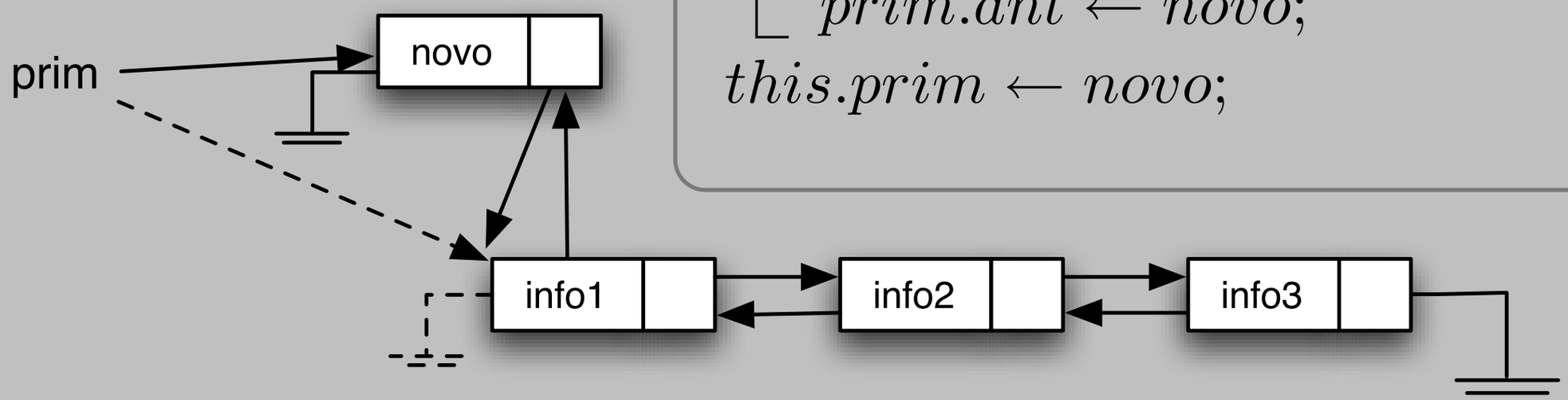
```
public class NoListaDupla {
    private int info;
    private NoListaDupla prox;
    private NoListaDupla ant;
}
```

Exemplo: método de inserção no início

- instancia novo objeto NoLista
- encadeia o elemento no início da lista existente

Algoritmo: insere(int info)

$NoListaDupla\ novo \leftarrow new\ NoListaDupla();$
$novo.info \leftarrow info;$
$novo.prox \leftarrow prim;$
$novo.ant \leftarrow null;$
se $prim \neq null$ **então**
  $prim.ant \leftarrow novo;$
$this.prim \leftarrow novo;$



Exemplo: método buscar elemento na lista

- recebe a informação referente ao elemento a pesquisar
- retorna o referência para o nó da lista que armazena o valor, ou null, caso o elemento não seja encontrado na lista

Algoritmo: busca(int v)

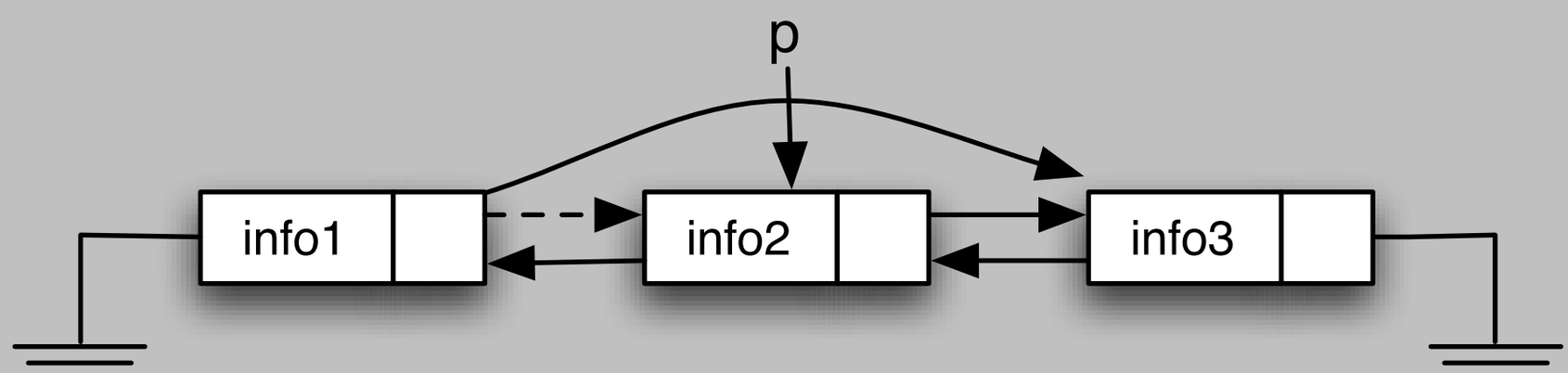
NoLista $p \leftarrow prim$;
enquanto $p \neq null$ **faça**
  se $p.info == v$ **então**
    retorna p;
  $p \leftarrow p.prox$;
retorna *null*;

Algoritmo 1.18: Busca um elemento na lista dupla

Listas Duplamente Encadeadas

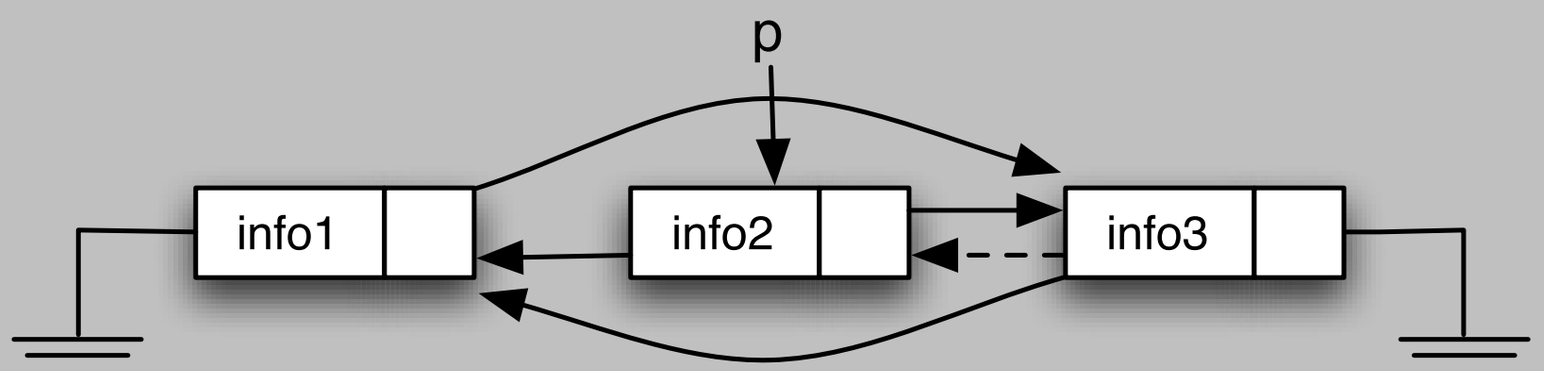
Método para retirar nó da lista

- p aponta para o nó a retirar
- Se p aponta para um nó no meio da lista:

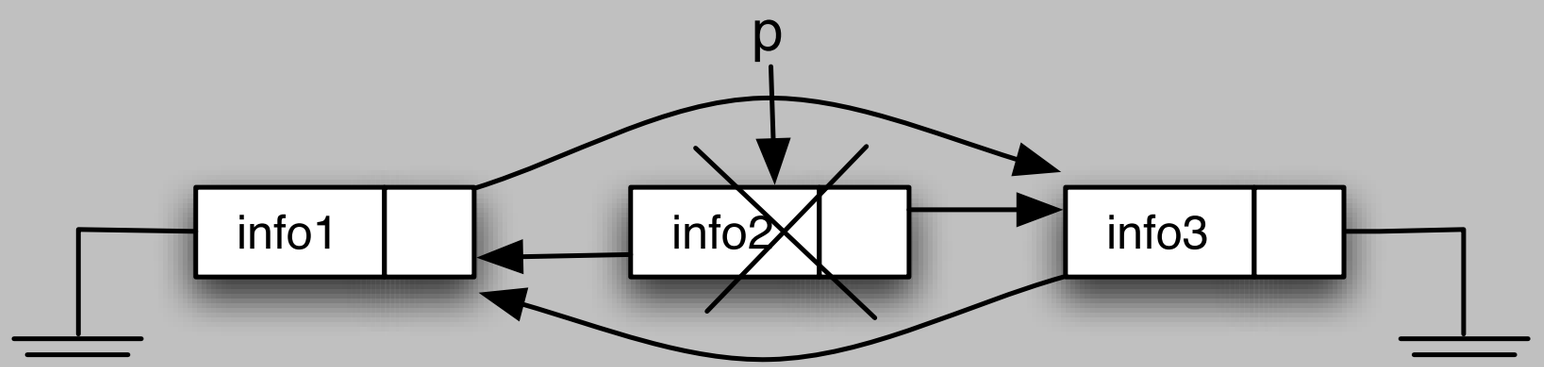


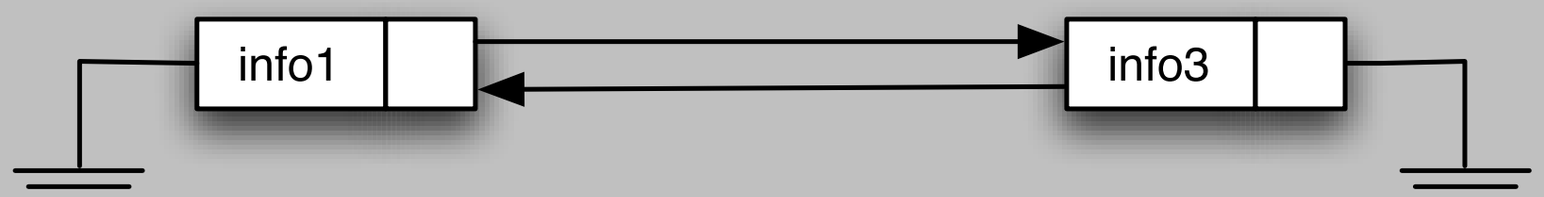

```
p.getAnt().setProx(p.getProx());
```

- Se p aponta para um nó no meio da lista:




```
p.getProx().setAnt(p.getAnt());
```

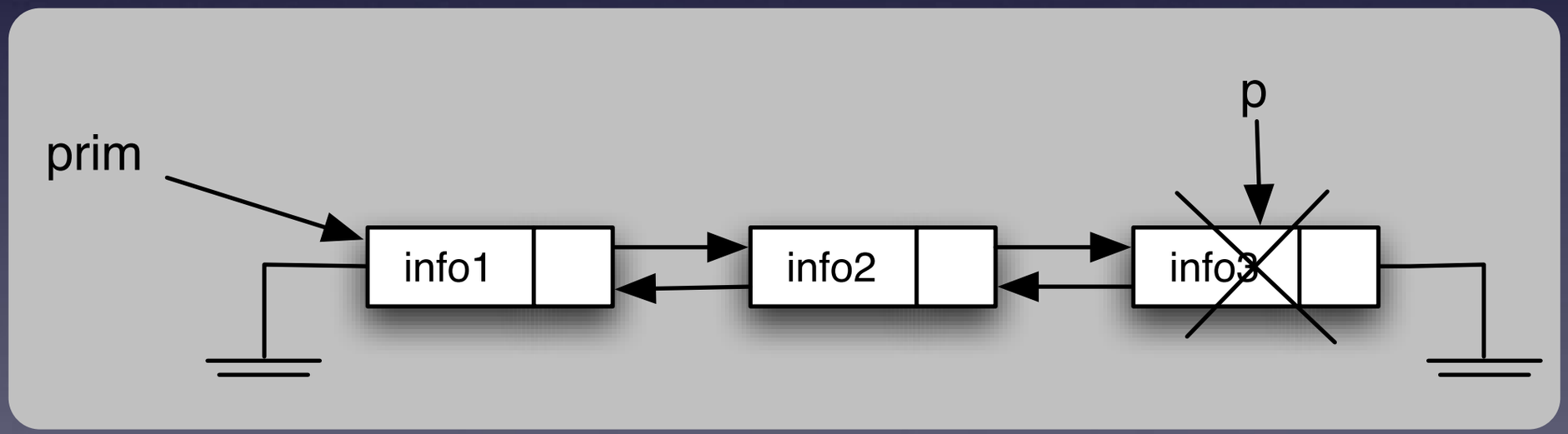




- Se p aponta para o último nó da lista, não é possível escrever

  p.getProx().setAnt(p.ant);

  porque p.prox é igual a null;

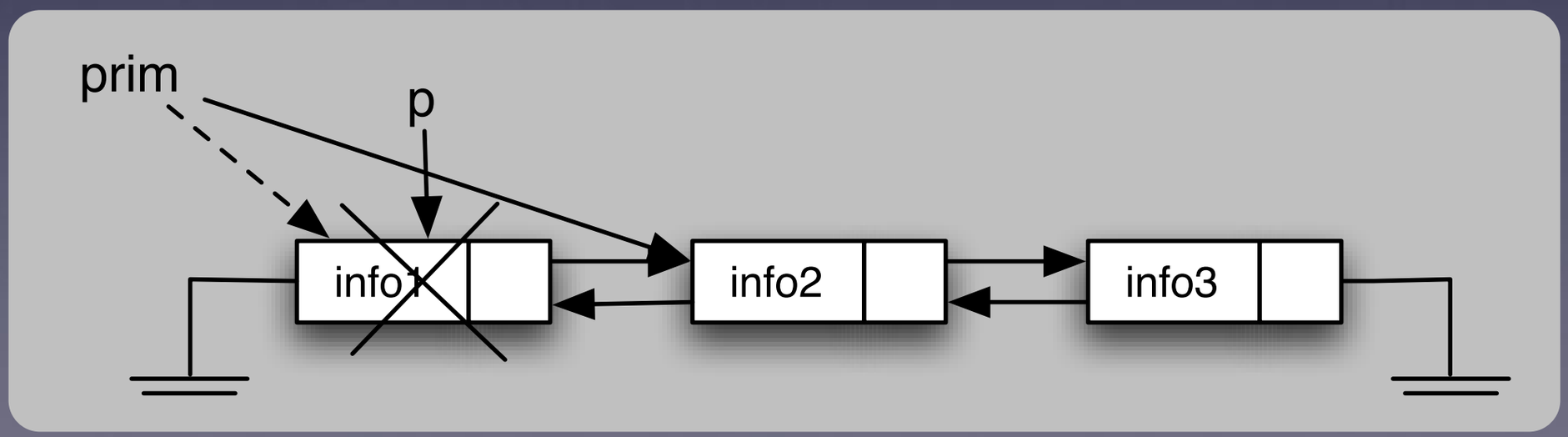


- Se p aponta para o primeiro nó da lista, não é possível escrever

  p.ant.prox ← p.prox;

  porque p.ant é igual a null;

- é necessário atualizar a referência para o primeiro nó da lista this.prim



Exemplo: método retira

Algoritmo: retira(int v)

NoListaDupla $p \leftarrow$ *busca(v)*;

```
// caso não achou elem...
```
se $p == null$ **então**
  retorna;

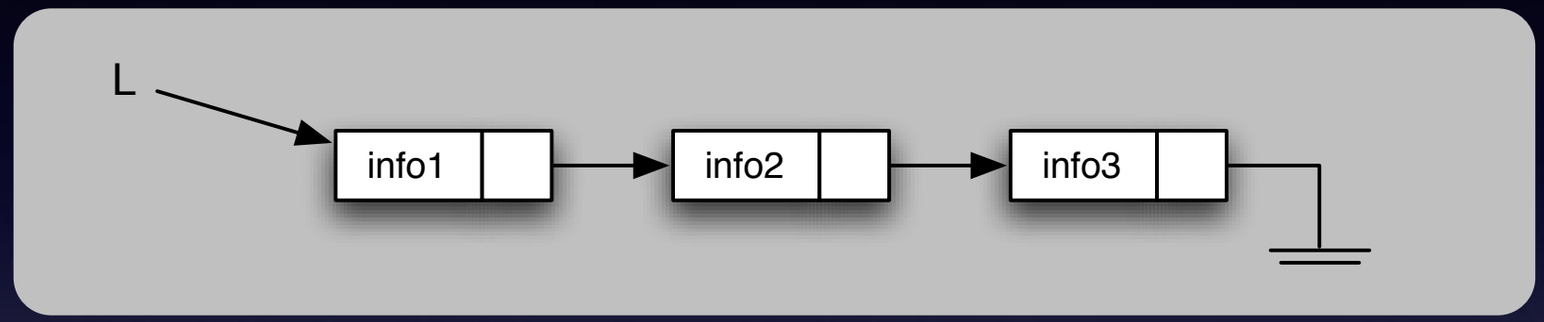
```
// testa se é o primeiro nó
```
se $prim == p$ **então**
  $prim \leftarrow p.prox$;
senão
  $p.ant.prox \leftarrow p.prox$;

```
// testa se não é o último
```
se $p.prox \neq null$ **então**
  $p.prox.ant \leftarrow p.ant$;

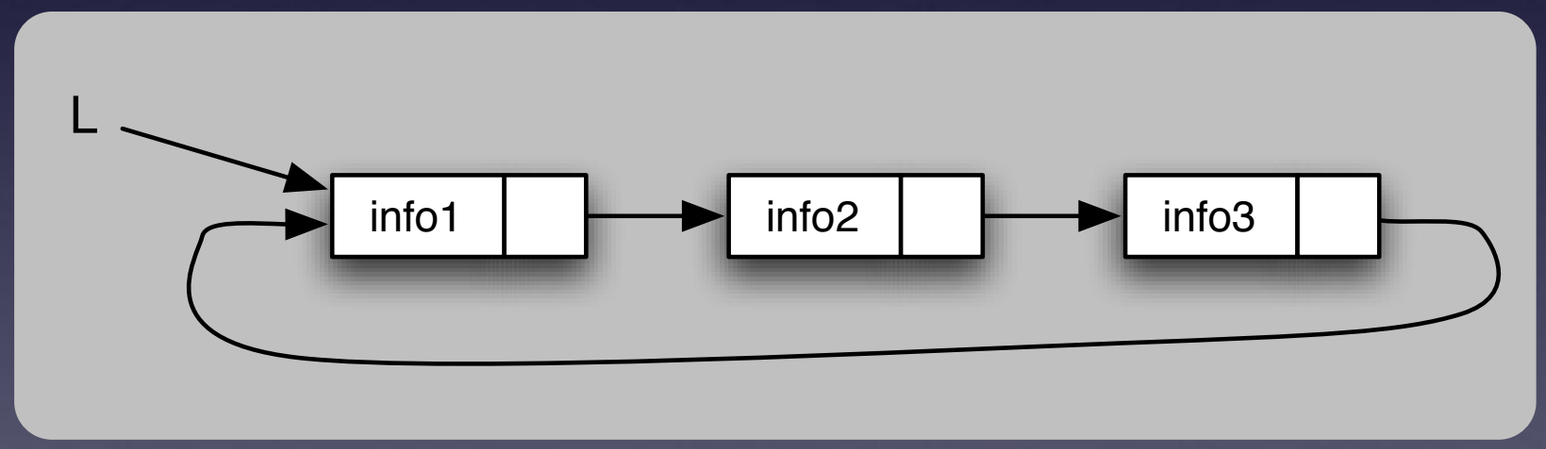
Algoritmo 1.19: Retira um elemento da lista duplamente encadeada

Resumo

- Listas simplesmente encadeadas



- Listas circulares



- Listas duplamente encadeadas