Anatomia de um Garbage Collector Moderno

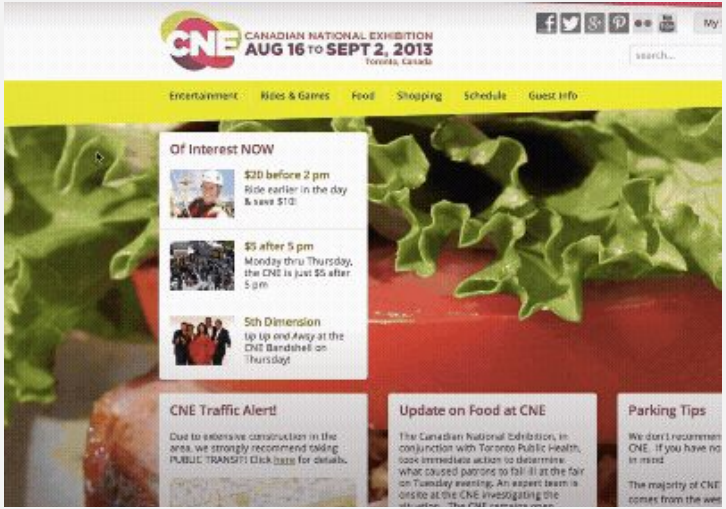
Renê Cardozo

- ➔ Formado em Ciência da Computação - USP
- ➔ 6 anos desenvolvendo em Go, Java e no que tiver.

Engine

@reneepc

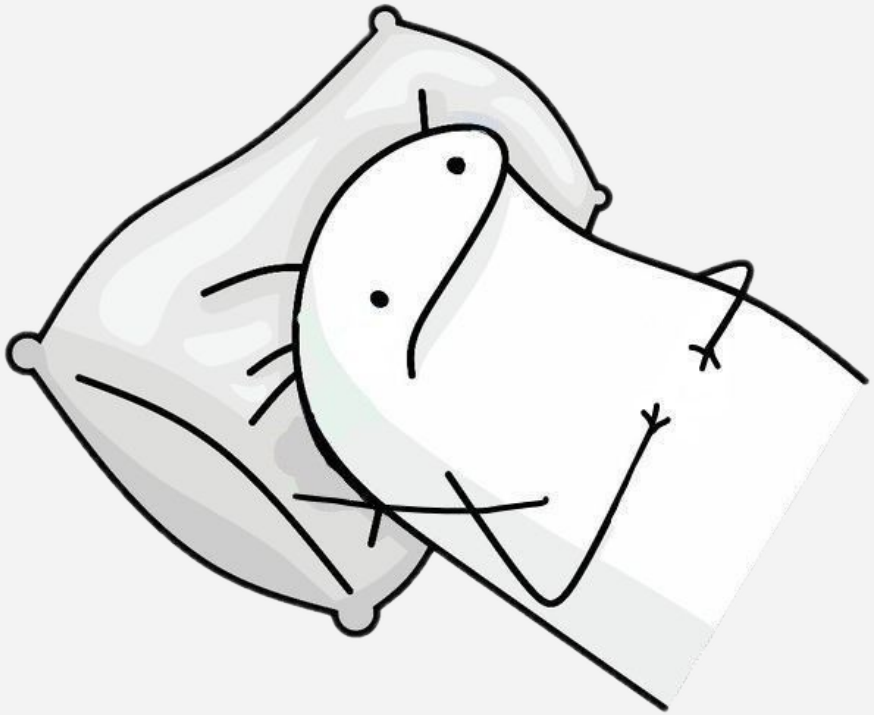
Jank



Janky

Fluído

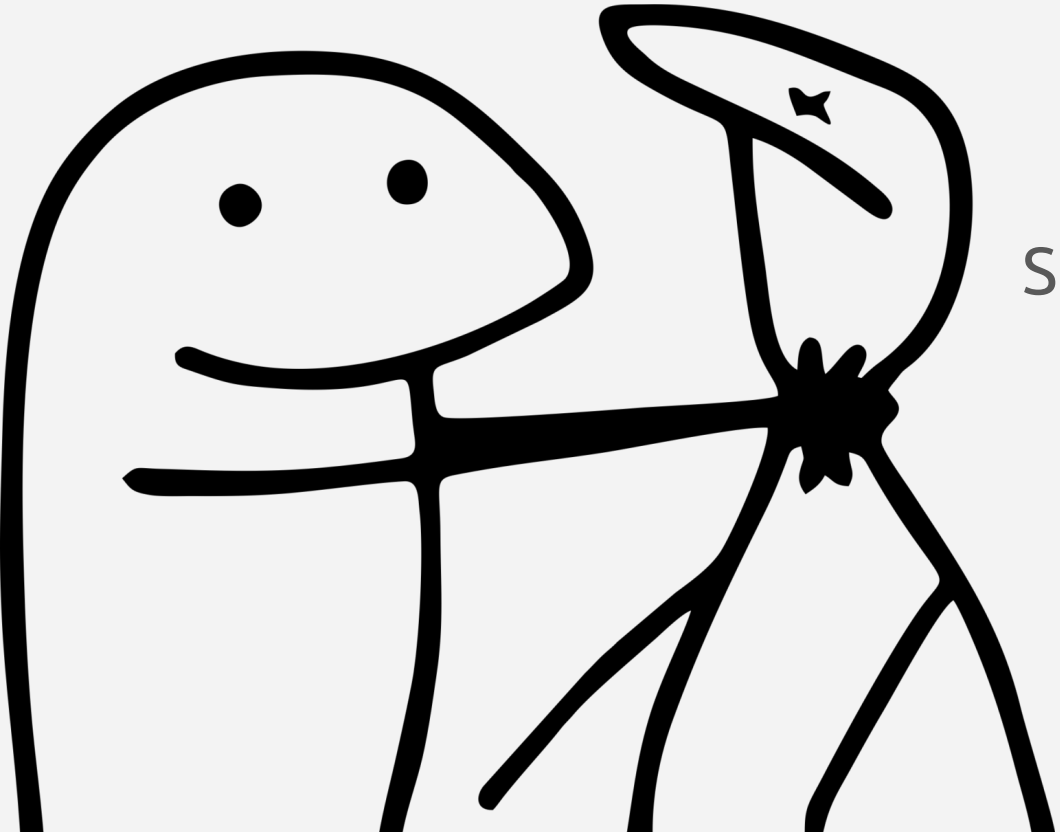
Mas o real motivo



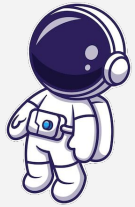
~~Ah, mano. É legal.~~

Não tinha ninguém melhor para palestrar

Se você achar ruim



Submeta sua palestra para o C4P da GolangSP e da GitTogether



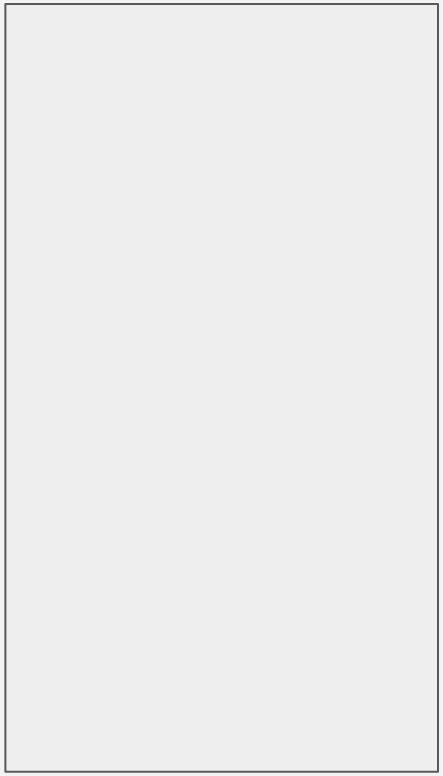
SP

Stack trace

```
Starting charge of $49.99 for user user123
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x2 addr=0x0
pc=0x100782e30]
goroutine 1 [running]:
main.(*PaymentGateway).Charge(0x0, "user123", 49.99)
    payment.go:34 +0x30

main.chargeCard("user123", 49.99)
    payment.go:26 +0xa4

main.processPayment("user123", 49.99)
    payment.go:14 +0x20

main.main()
    payment.go:6 +0x30

exit status 2
```

Stack

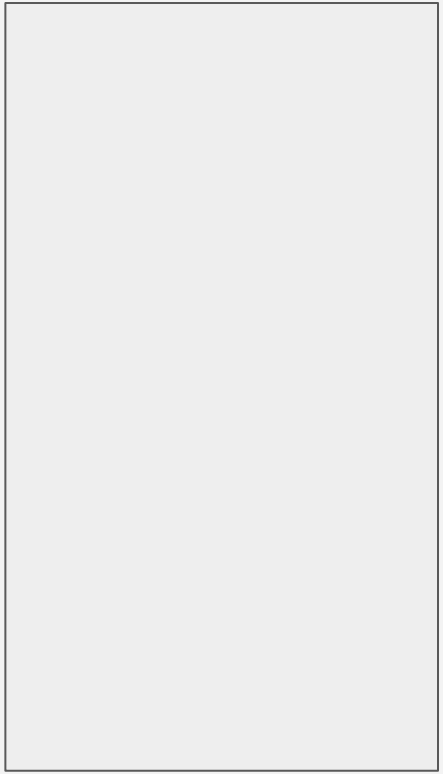
```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```

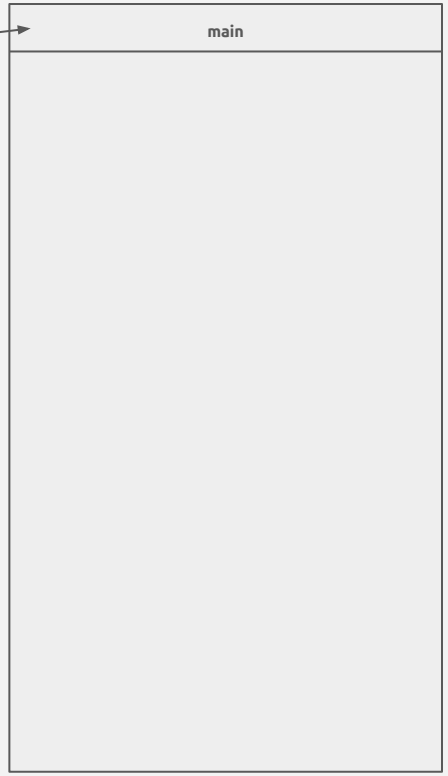
Cada thread da sua aplicação tem uma



Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```

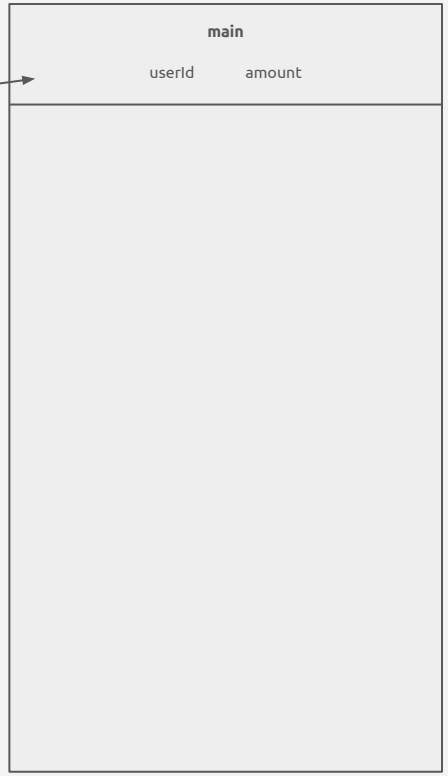
Cria um
stack frame



Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```

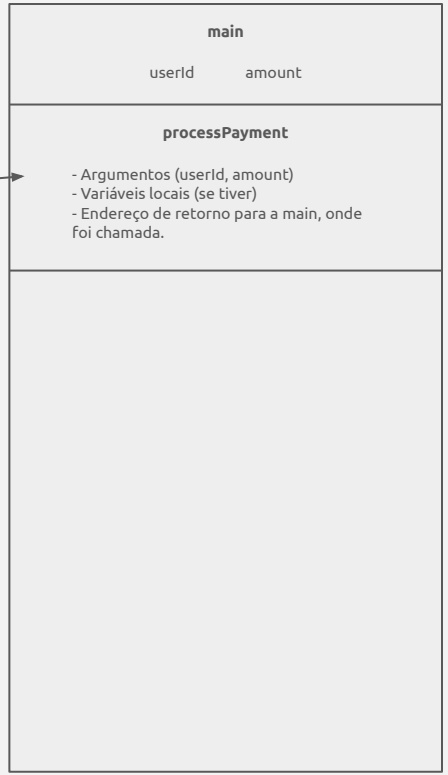
Variáveis locais são armazenadas na Stack



Stack

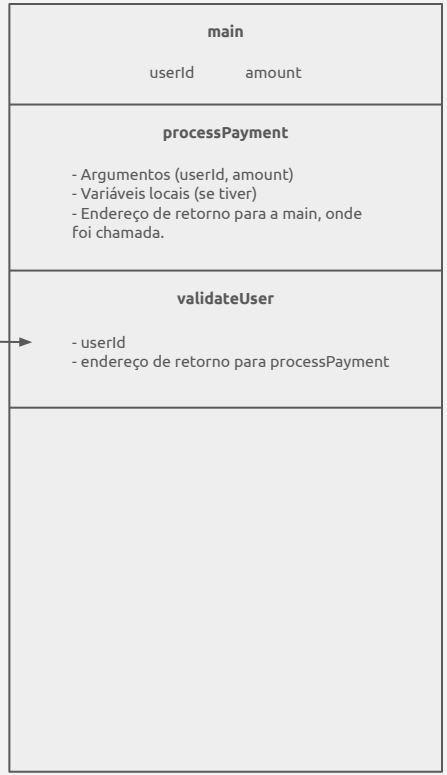
```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```

Cada stack frame mantém o **contexto de execução**.



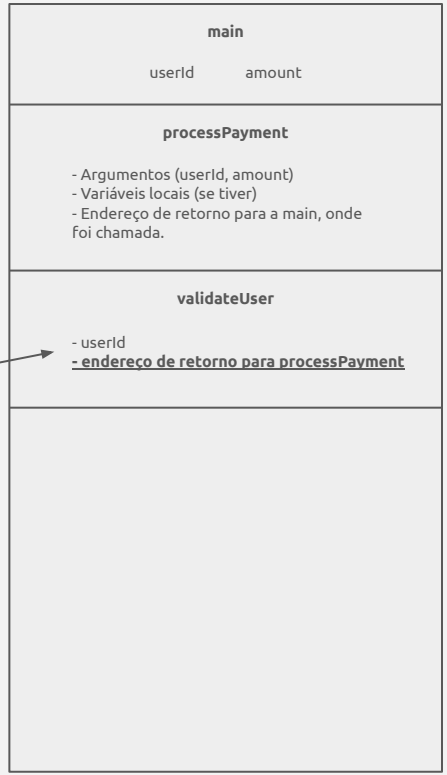
Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



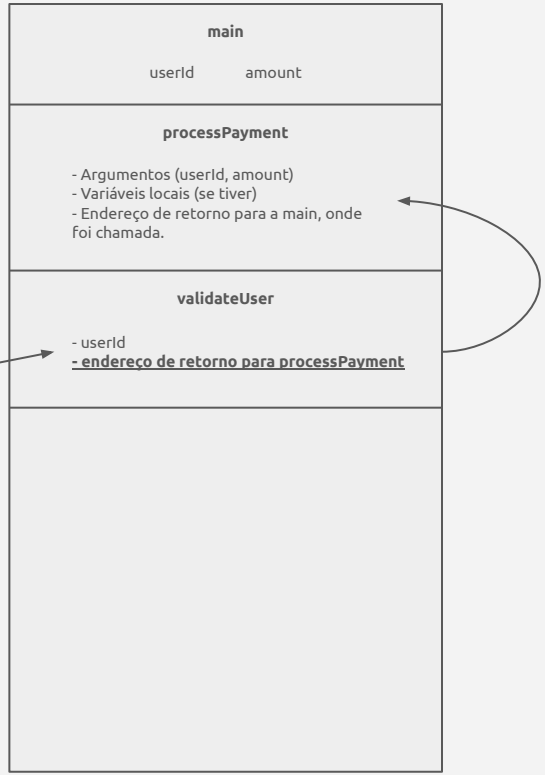
Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



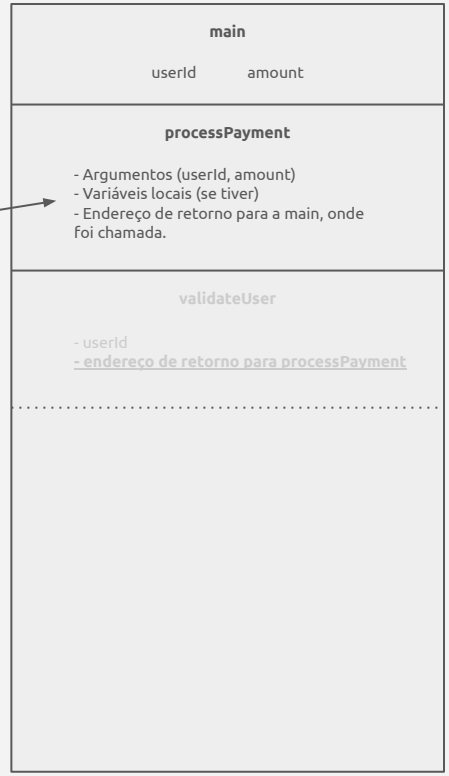
Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



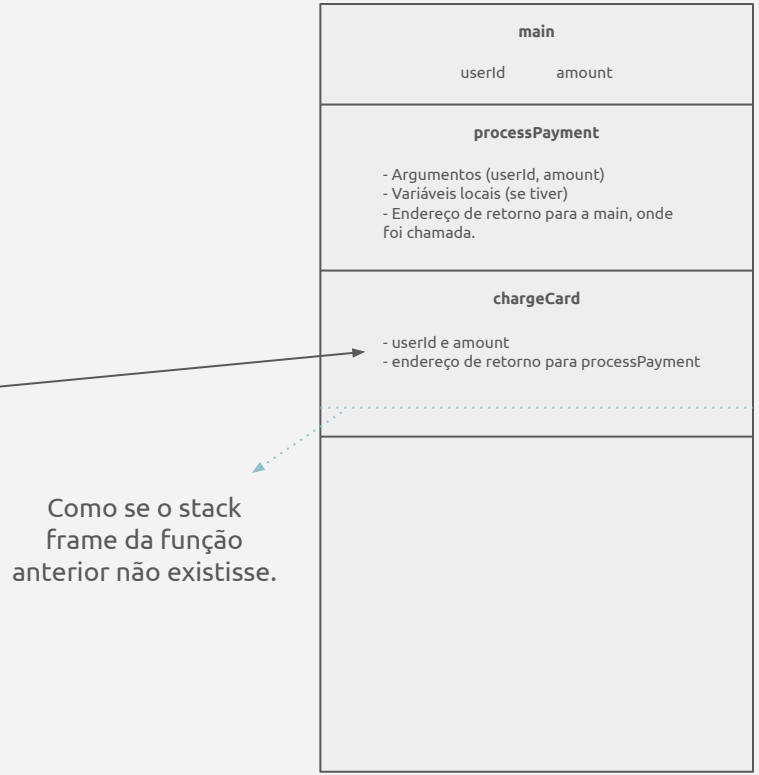
Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



Stack

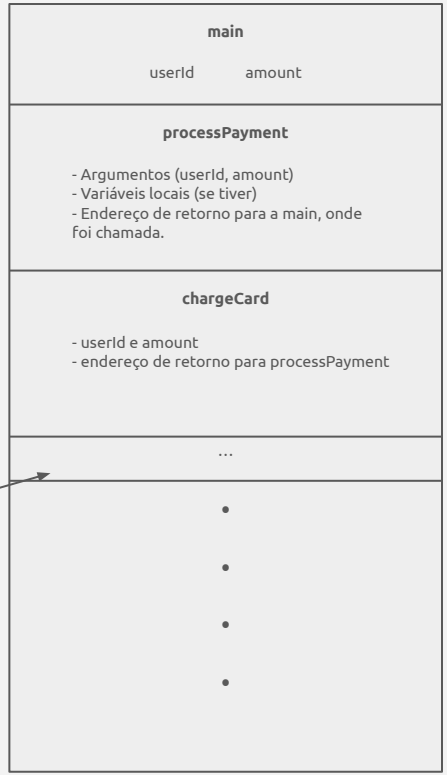
```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```



Como se o stack frame da função anterior não existisse.

Stack

```go
package main

import "fmt"

func main() {
    userId := "user123"
    amount := 49.99

    processPayment(userId, amount)
}

func processPayment(userID string, amount float64) {
    if !validateUser(userID) {
        fmt.Println("Invalid user")
        return
    }
    chargeCard(userID, amount)
}

func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

func chargeCard(userID string, amount float64) {
    fmt.Printf("Charging $%.2f to user %s\n", amount,
userID)
    // Aciona o gateway de pagamentos
}
```

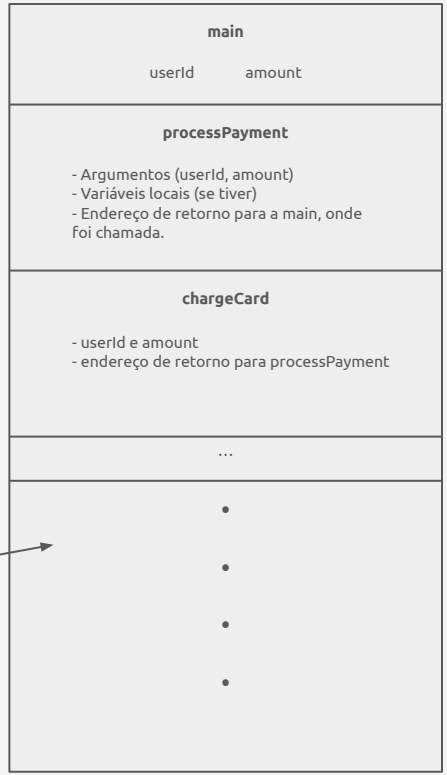


Stack

```
Starting charge of $49.99 for user user123
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x2 addr=0x0
pc=0x100782e30]
goroutine 1 [running]:
main.(*PaymentGateway).Charge(0x0, "user123", 49.99)
    payment.go:34 +0x30

main.chargeCard("user123", 49.99)
    payment.go:26 +0xa4

main.processPayment("user123", 49.99)
    payment.go:14 +0x20

main.main()
    payment.go:6 +0x30

exit status 2
```

```
userID)
    // Aciona o gateway de pagamentos
}
```

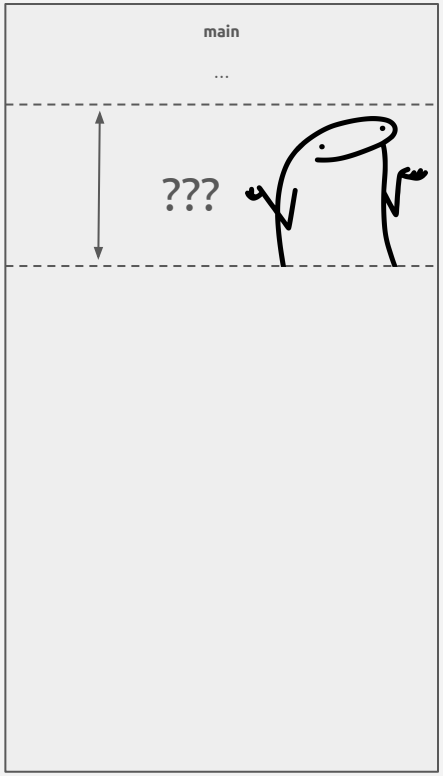
Caminha os endereços de retorno da stack para montar a stack trace.



Quando a stack não é suficiente

```go
package main

import (
    "fmt"
)

type Notification struct {
    UserID  string
    Channel string
    Message string
}

func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

func buildNotifications(userID string, amount float64) []Notification
{
    notifications := make([]Notification, 0)

    for _, channel = getUserNotificationChannels(userId) {
        notifications = append(notifications, Notification{
            UserID: userID,
            Channel: channel.Type,
            Message: channel.paymentNotification(amount)
        })
    }

    return notifications
}
```

Não é possível saber quantas notificações serão geradas, porque depende das configurações do usuário.

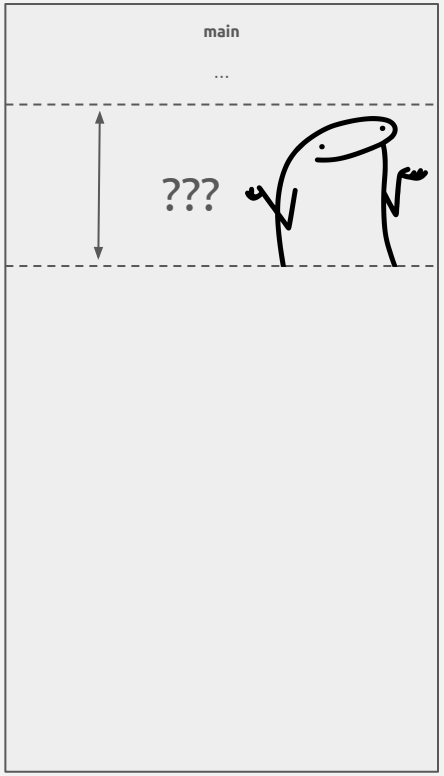


Quando a stack não é suficiente

```go
package main

import (
    "fmt"
)

type Notification struct {
    UserID  string
    Channel string
    Message string
}

func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

func buildNotifications(userID string, amount float64) []Notification
{
    notifications := make([]Notification, 0)

    for _, channel = getUserNotificationChannels(userId) {
        notifications = append(notifications, Notification{
            UserID: userID,
            Channel: channel.Type,
            Message: channel.paymentNotification(amount)
        })
    }

    return notifications
}
```

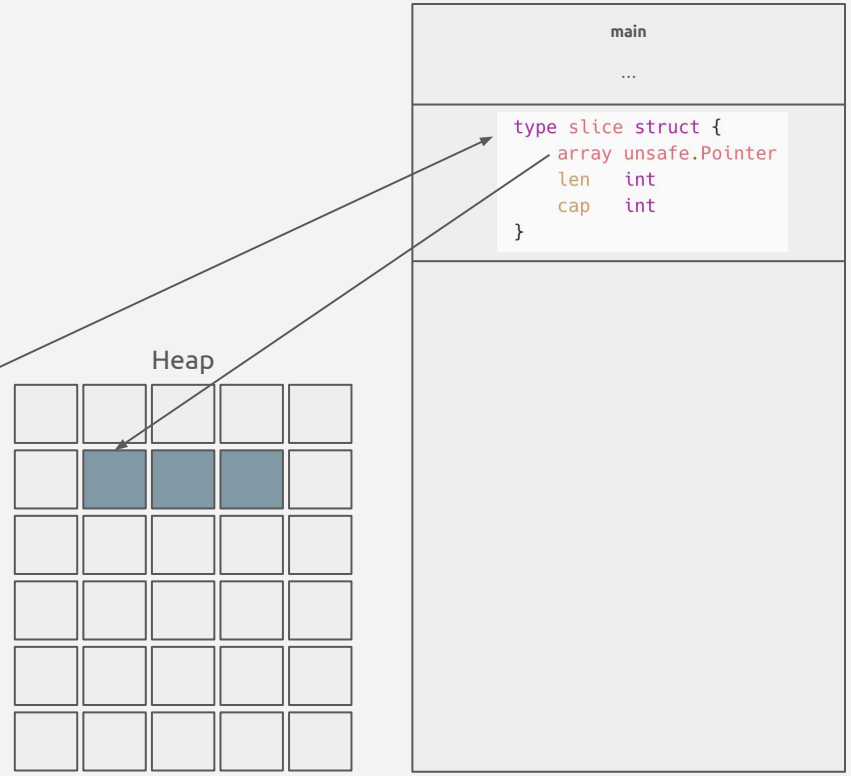
Além disso, a stack também tem tamanho limitado.

Não é possível saber quantas notificações serão geradas, porque depende das configurações do usuário.



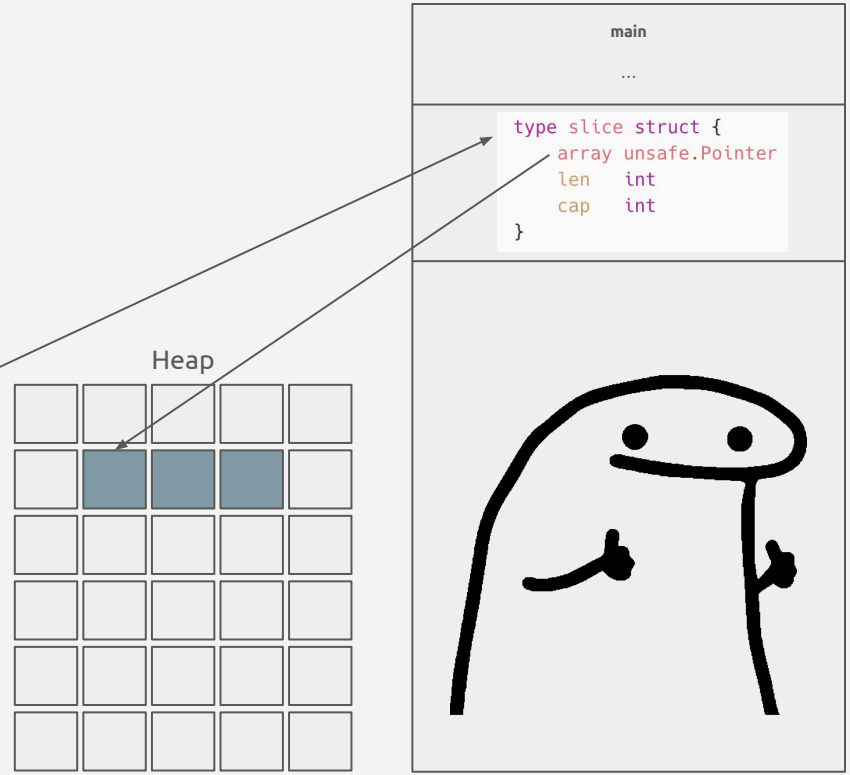
Heap

```go
package main

import (
    "fmt"
)

type Notification struct {
    UserID  string
    Channel string
    Message string
}

func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

func buildNotifications(userID string, amount float64) []Notification
{
    notifications := make([]Notification, 0)

    for _, channel = getUserNotificationChannels(userId) {
        notifications = append(notifications, Notification{
            UserID: userID,
            Channel: channel.Type,
            Message: channel.paymentNotification(amount)
        })
    }

    return notifications
}
```



Heap

```go
package main

import (
    "fmt"
)

type Notification struct {
    UserID  string
    Channel string
    Message string
}

func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

func buildNotifications(userID string, amount float64) []Notification
{
    notifications := make([]Notification, 0)

    for _, channel = getUserNotificationChannels(userId) {
        notifications = append(notifications, Notification{
            UserID: userID,
            Channel: channel.Type,
            Message: channel.paymentNotification(amount)
        })
    }

    return notifications
}
```



Gerenciamento Manual

Alocar e desalocar memória são comandos explícitos.

Gerenciamento Manual

malloc/free

Alocar e desalocar memória são comandos explícitos.

new/delete

Gerenciamento Manual

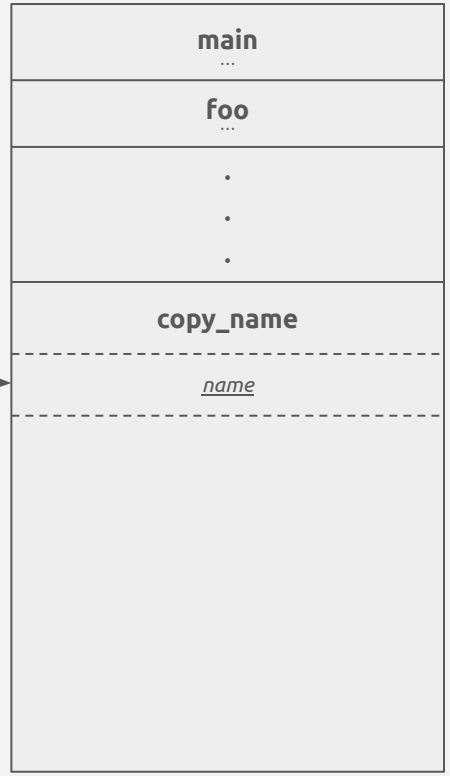
malloc/free

Alocar e desalocar memória são comandos explícitos.

new/delete

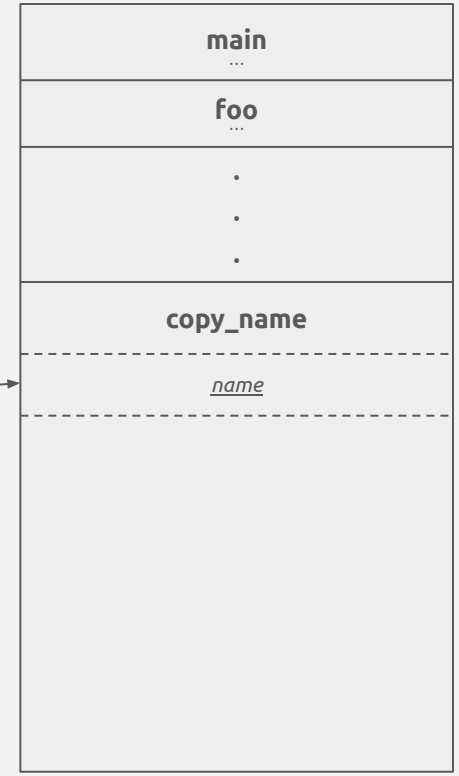
Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char name[100];
    strcpy(name, input);
    return name;
}
```



Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char name[100];
    strcpy(name, input);
    return name;
}
```

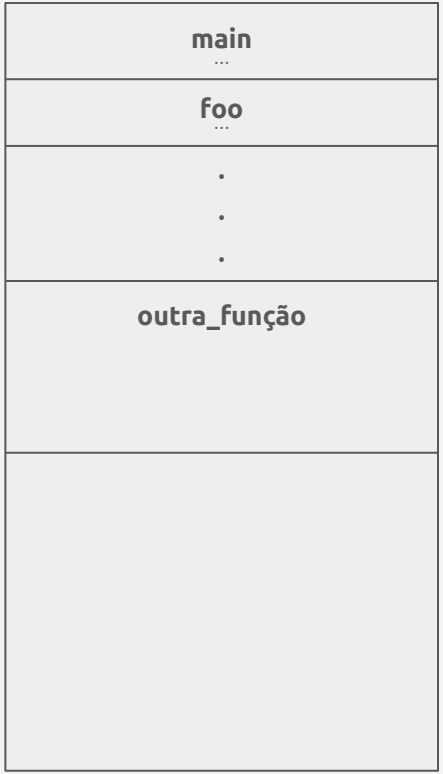


Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char name[100];
    strcpy(name, input);
    return name;
}
```

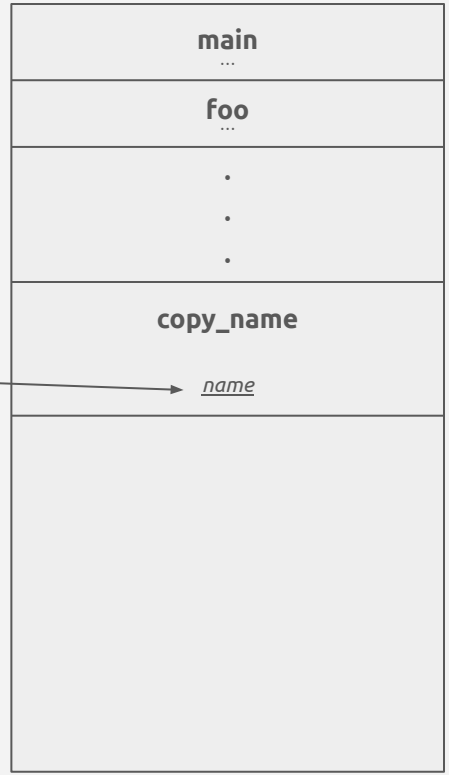
Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char name[100];
    strcpy(name, input);
    return name;
}
```



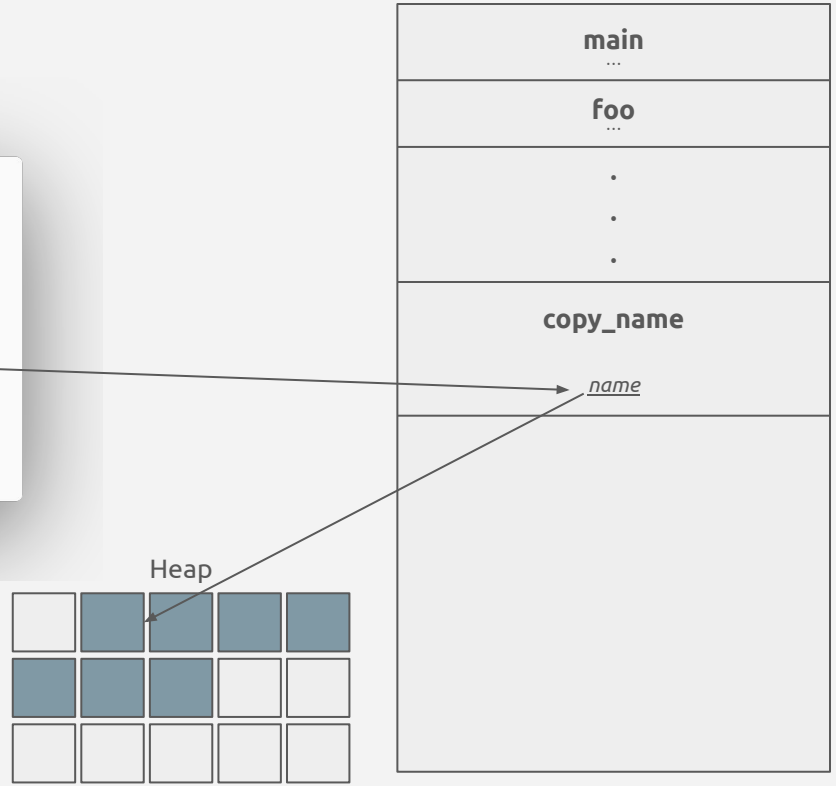
Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```



Gerenciamento Manual

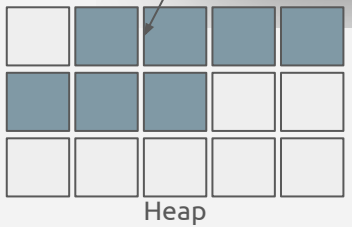
```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```



Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```

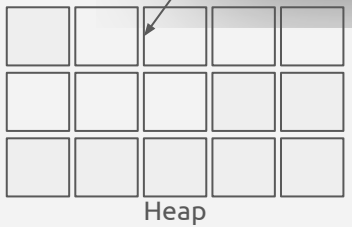
```c
#include <stdio.h>

int função_chama_copia() {
    ...
    char* myname = copy_name("Renê");
    printf("My name is %s\n", myname);
    free(myname);
    ...
}
```



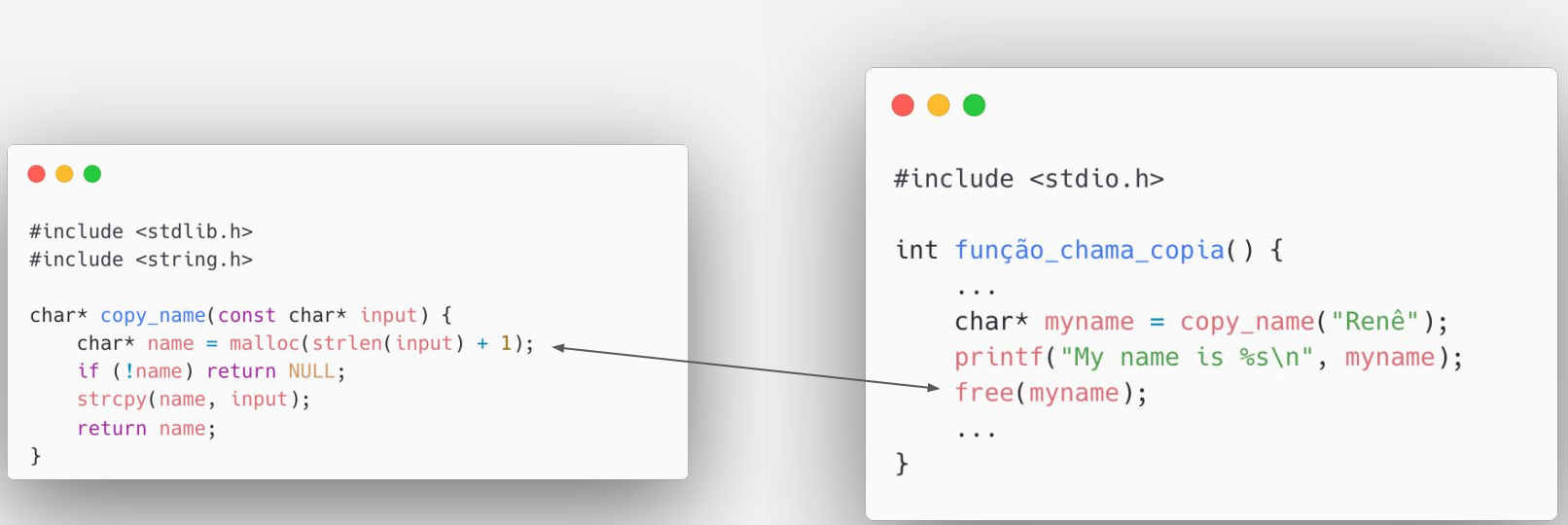
Gerenciamento Manual

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```

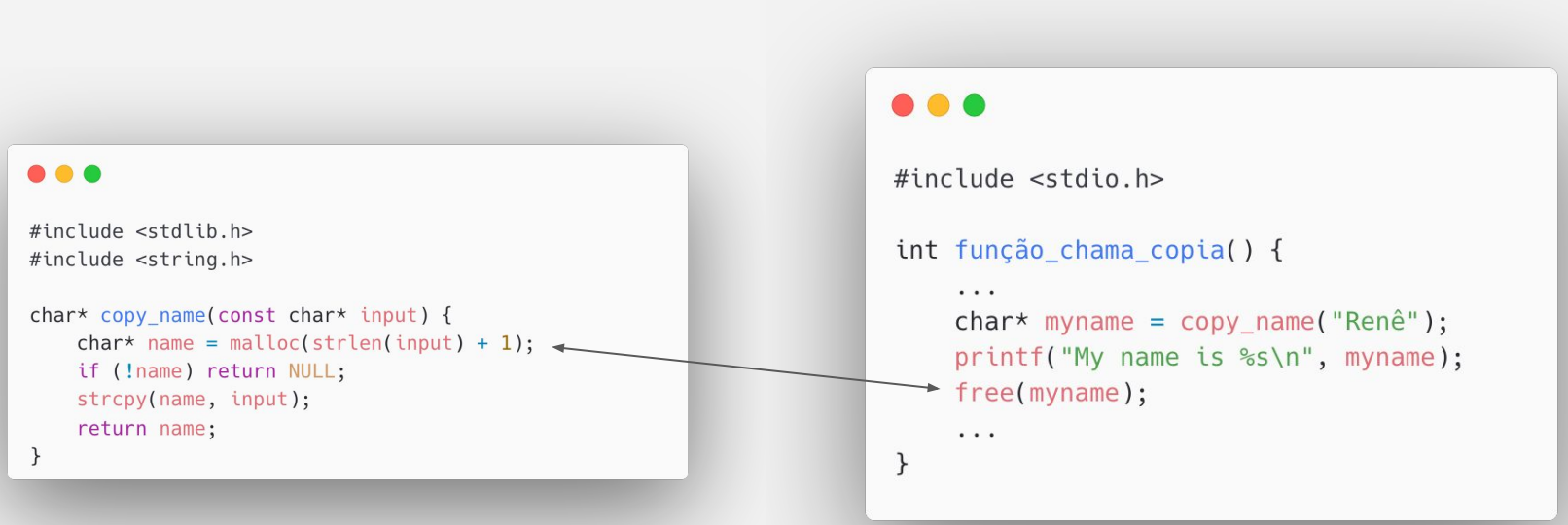
```c
#include <stdio.h>

int função_chama_copia() {
    ...
    char* myname = copy_name("Renê");
    printf("My name is %s\n", myname);
    free(myname);
    ...
}
```



Gerenciamento Manual



Caso isso não seja feito corretamente...



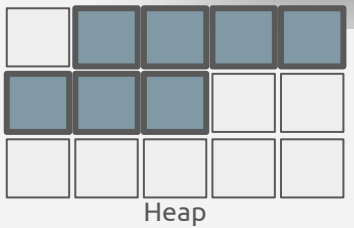
```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```

```c
#include <stdio.h>

int função_chama_copia() {
    ...
    char* myname = copy_name("Renê");
    printf("My name is %s\n", myname);
    free(myname);
    ...
}
```

Memory leak

```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```

```c
#include <stdio.h>

int função_chama_copia() {
    ...
    char* myname = copy_name("Renê");
    printf("My name is %s\n", myname);
    free(myname);
    ...
}
```

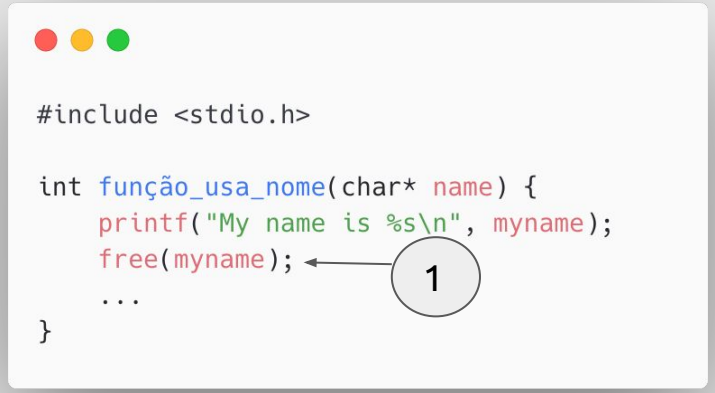


Double free

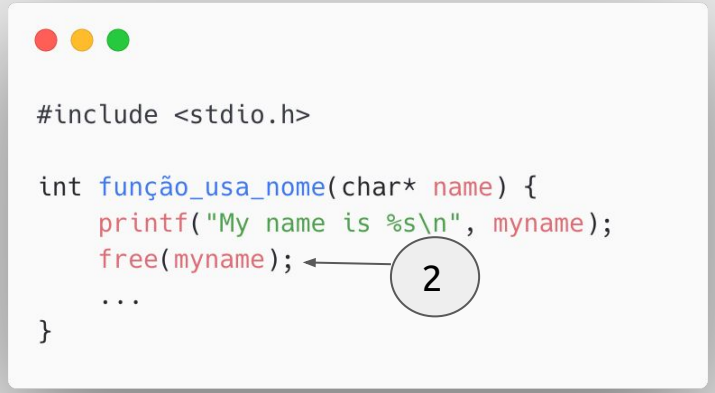
```c
#include <stdlib.h>
#include <string.h>

char* copy_name(const char* input) {
    char* name = malloc(strlen(input) + 1);
    if (!name) return NULL;
    strcpy(name, input);
    return name;
}
```

```c
#include <stdio.h>

int função_chama_copia() {
    ...
    char* myname = copy_name("Renê");
    printf("My name is %s\n", myname);
    free(myname); <-----
    ...
    ...
    free(myname); <-----
}
```

Porque alguém faria isso?

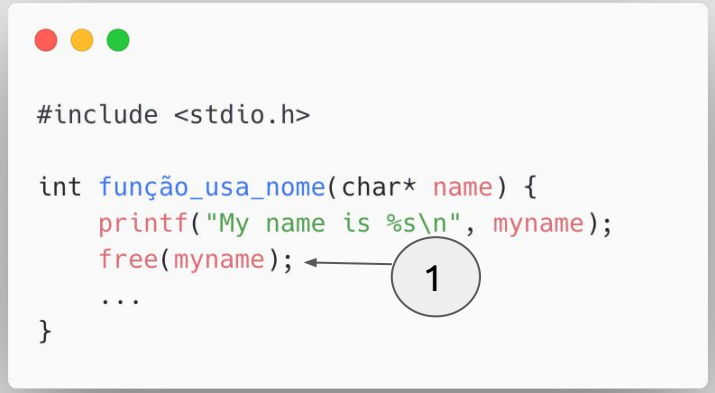


Thread 1



Thread 2

Use-after-free



Thread 1

Thread 2

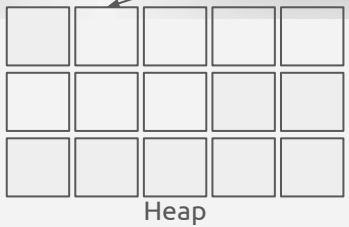
Use-after-free

```c
#include <stdio.h>

int função_usa_nome(char* name) {
    printf("My name is %s\n", myname);
    free(myname);
    ...
}
```

1

```c
#include <stdio.h>

int função_usa_nome(char* name) {
    printf("My name is %s\n", myname);
    ...
}
```

2



Thread 1

Thread 2

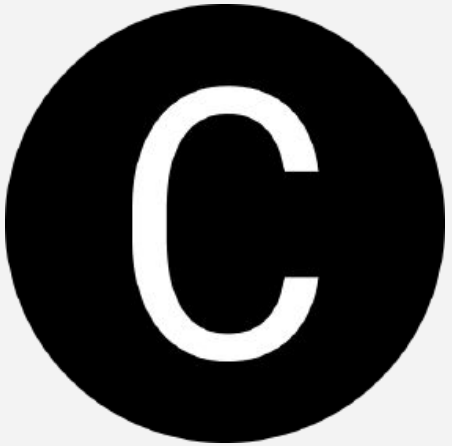
O gerenciamento de memória vira parte da API

Gerenciamento Manual

70% dos bugs são relacionados
a gerenciamento de memória.

Qual a solução?

Qual a solução?



Garbage Collection

Gerenciamento de memória automático

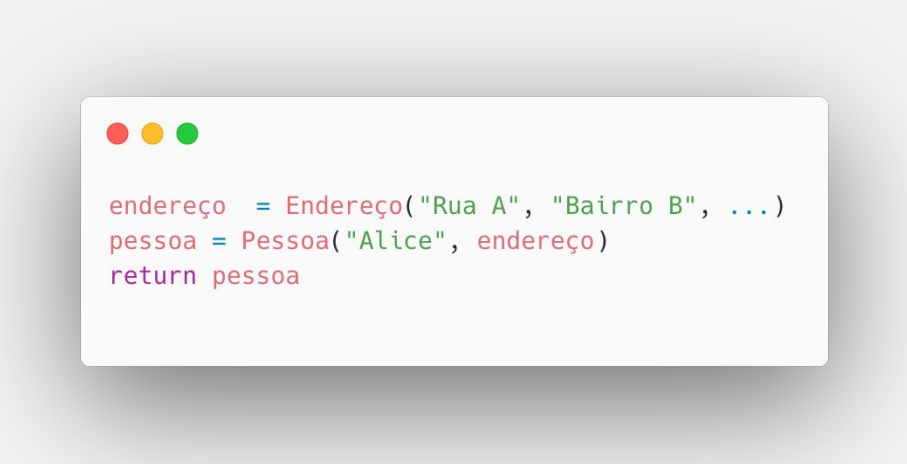
Limpar a memória não é o problema

Gerenciamento de memória automático

Limpar a memória não é o problema

O problema é achar aquilo que eu preciso remover

Contagem de referências



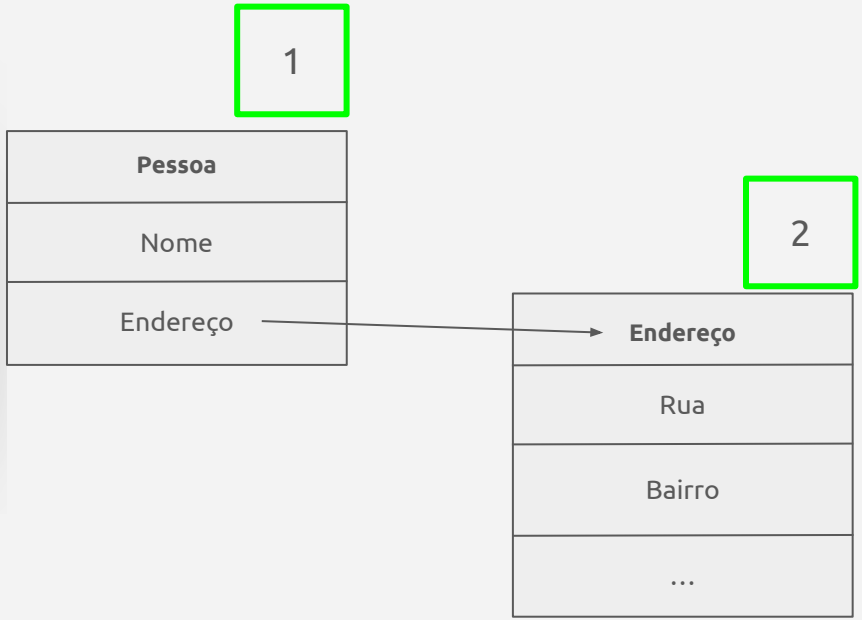
```
endereço  = Endereço("Rua A", "Bairro B", ...)
pessoa = Pessoa("Alice", endereço)
return pessoa
```

Contagem de referências

```
endereço  = Endereço("Rua A", "Bairro B", ...)
pessoa = Pessoa("Alice", endereço)
return pessoa
```

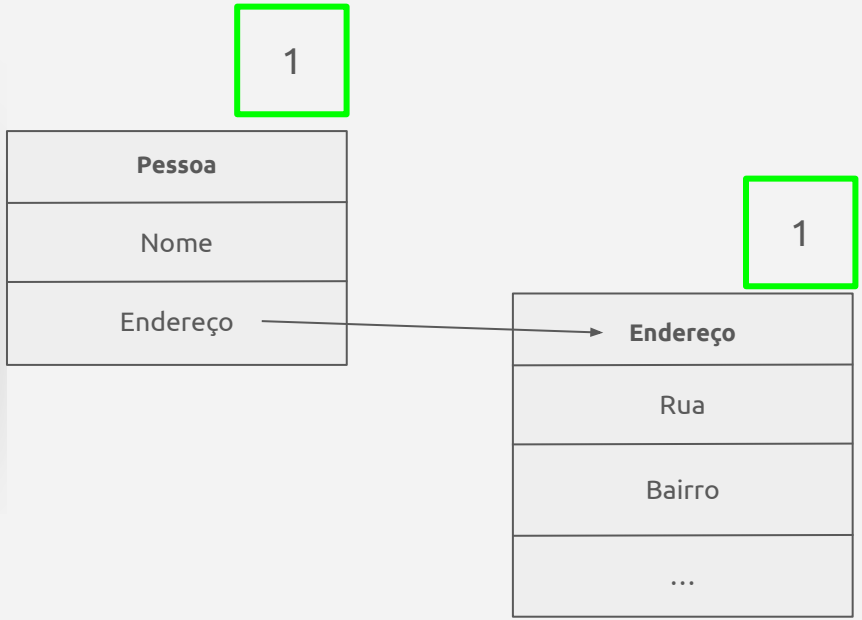
Contagem de referências

```
endereço  = Endereço("Rua A", "Bairro B", ...)
pessoa = Pessoa("Alice", endereço)
return pessoa
```



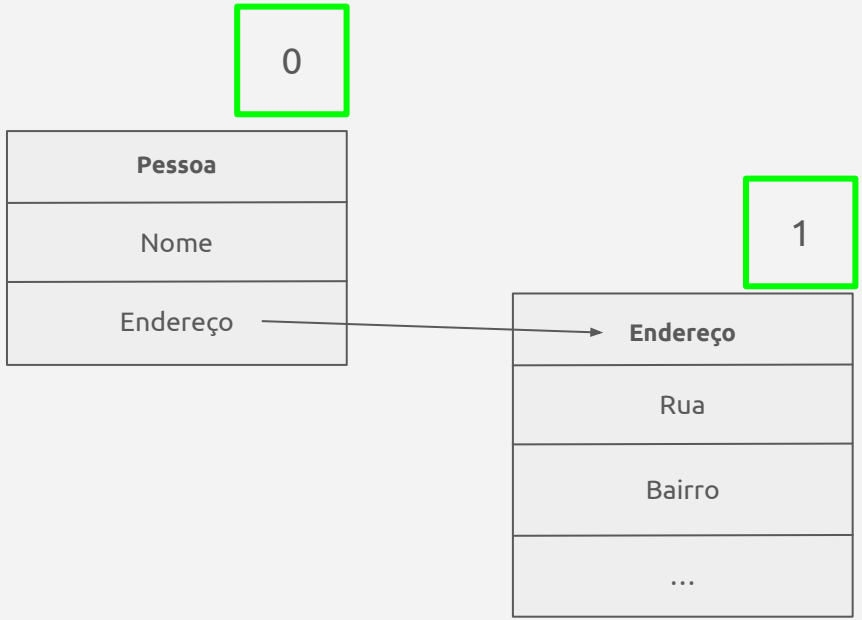
Contagem de referências

```
endereço  = Endereço("Rua A", "Bairro B", ...)
pessoa = Pessoa("Alice", endereço)
return pessoa
```



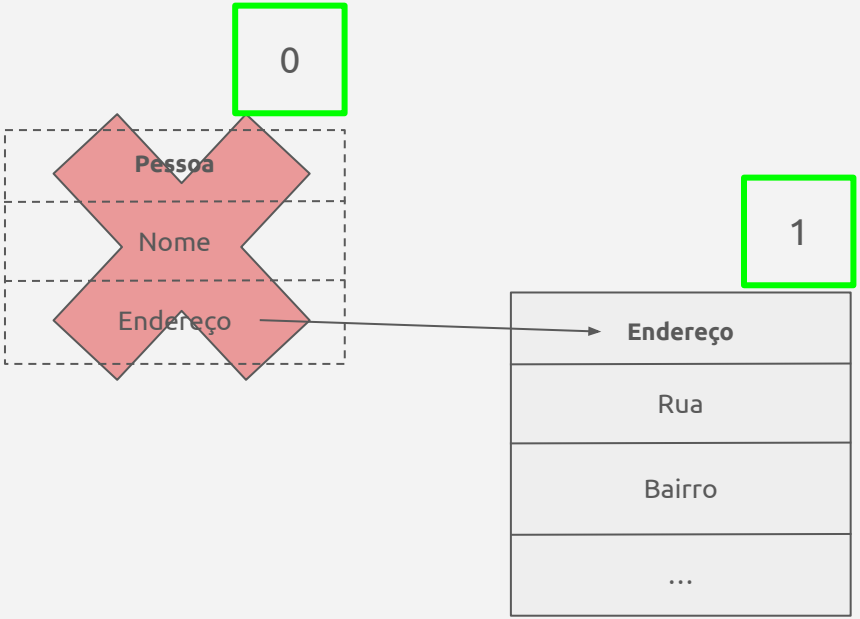
Contagem de referências

```
pessoa = None
```



Contagem de referências

```
pessoa = None
```



Contagem de referências

```
pessoa = None
```

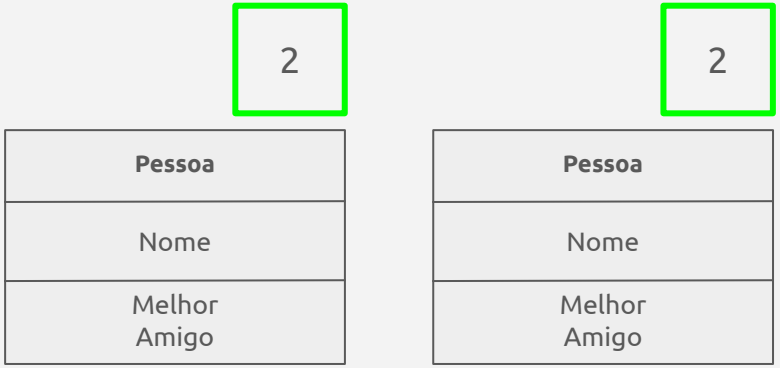
Contagem de referências

Contagem de referências

```python
def virar_melhores_amigos(p1: Pessoa, p2: Pessoa) -> None:
    p1.melhor_amigo = p2
    p2.melhor_amigo = p1


alice = Pessoa("Alice")
bob   = Pessoa("Bob")

virar_melhores_amigos(alice, bob)
```

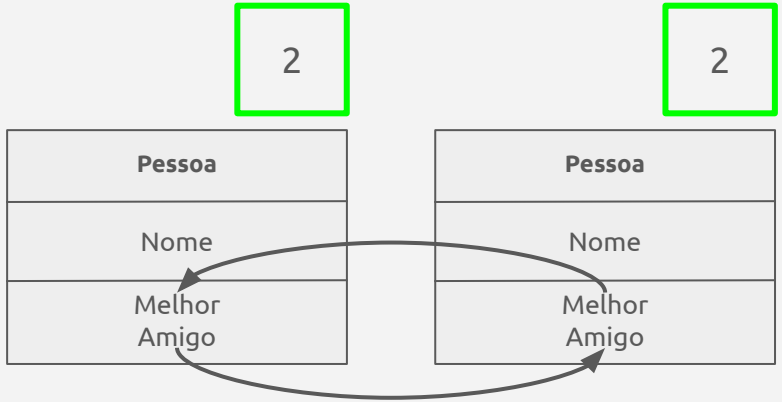
Contagem de referências

```python
def virar_melhores_amigos(p1: Pessoa, p2: Pessoa) -> None:
    p1.melhor_amigo = p2
    p2.melhor_amigo = p1


alice = Pessoa("Alice")
bob   = Pessoa("Bob")

virar_melhores_amigos(alice, bob)
```



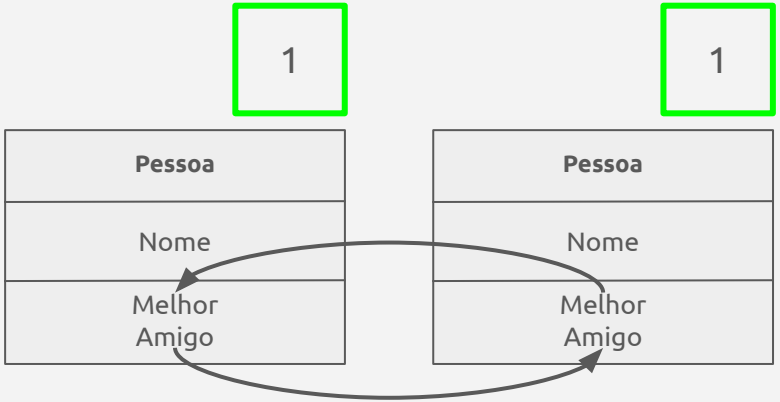
Contagem de referências

```python
def virar_melhores_amigos(p1: Pessoa, p2: Pessoa) -> None:
    p1.melhor_amigo = p2
    p2.melhor_amigo = p1


alice = Pessoa("Alice")
bob   = Pessoa("Bob")

virar_melhores_amigos(alice, bob)
```



Contagem de referências falha em ciclos

```
alice = None
bob = None
```

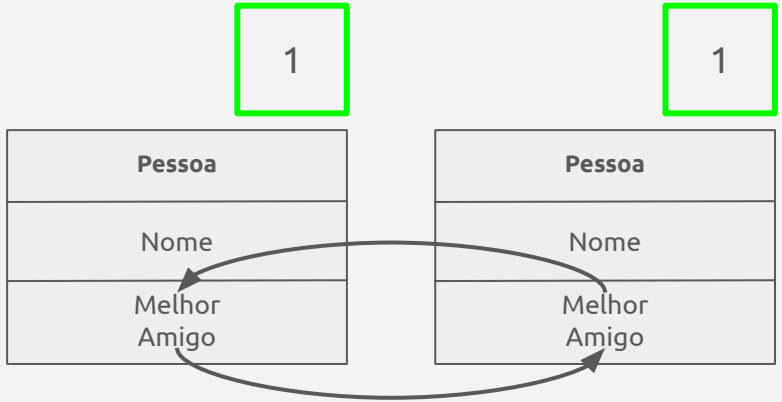


Desvantagens

Necessita de contador para cada objeto na memória

Cada atribuição tem custo

Ciclos necessitam de tratamento especial



Apesar das desvantagens

Apesar das desvantagens

Algumas libs
e frameworks

Swift

Apesar das desvantagens

Swift

A resolução de ciclos é responsabilidade do programador

Apesar das desvantagens

Swift

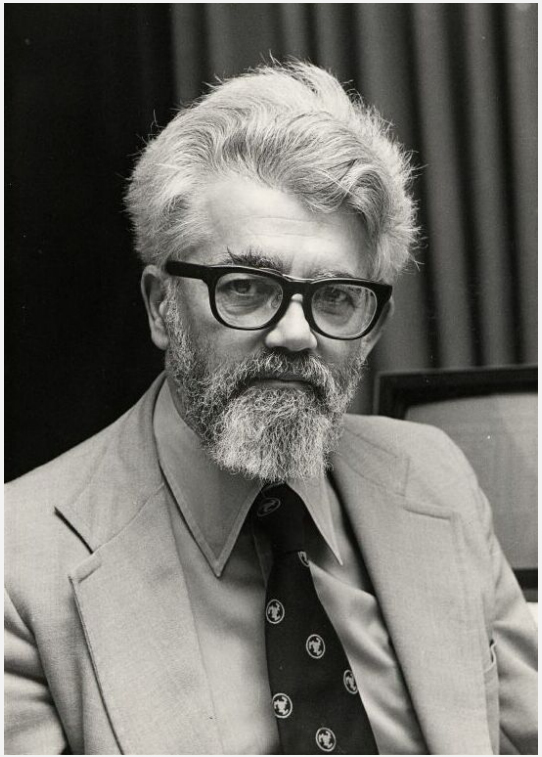
A resolução de ciclos é responsabilidade do programador

Ainda precisam de um GC para ciclos

De onde vem o Garbage Collector

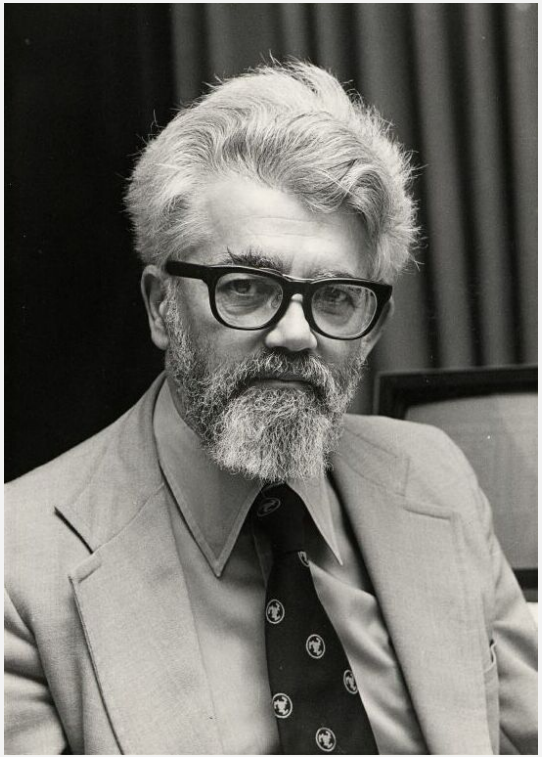
AI e Garbage Collector

AI e Garbage Collector



John McCarthy

AI e Garbage Collector

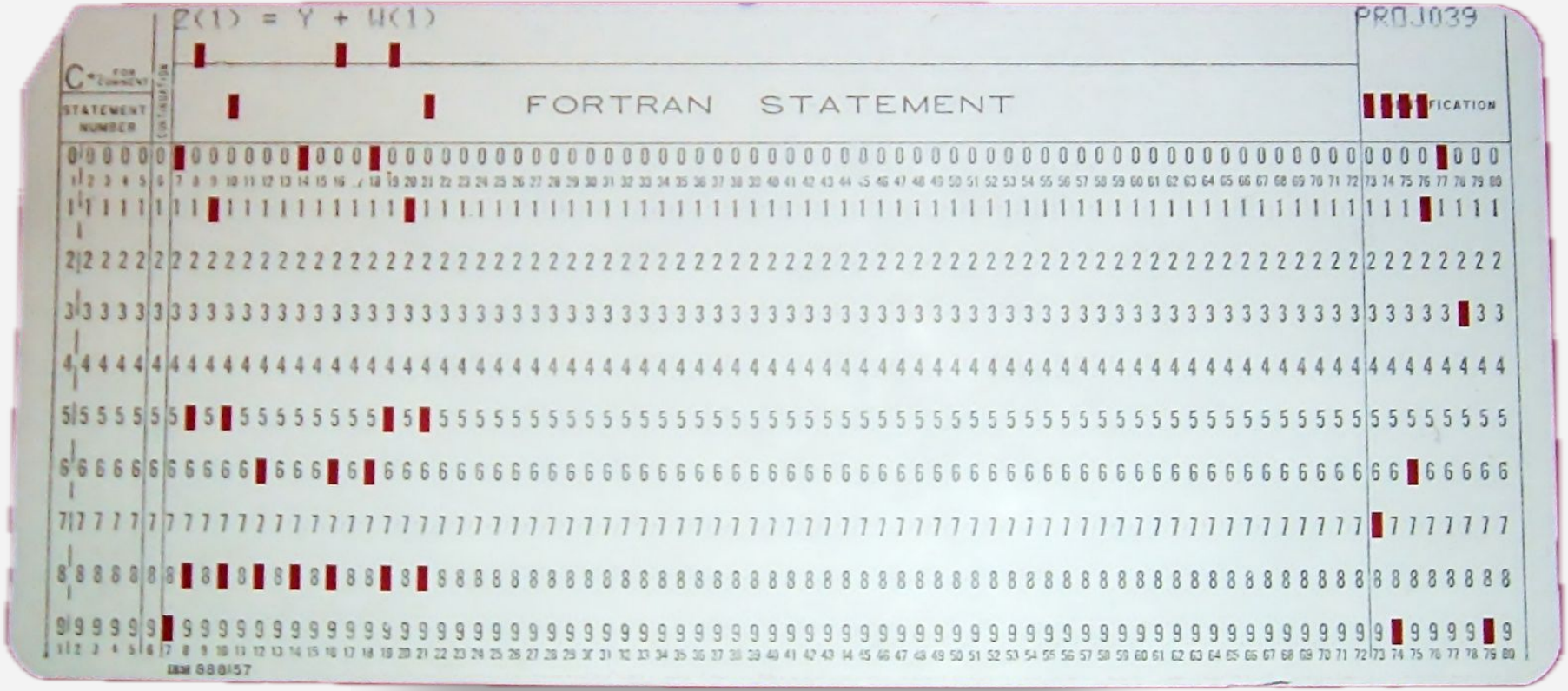


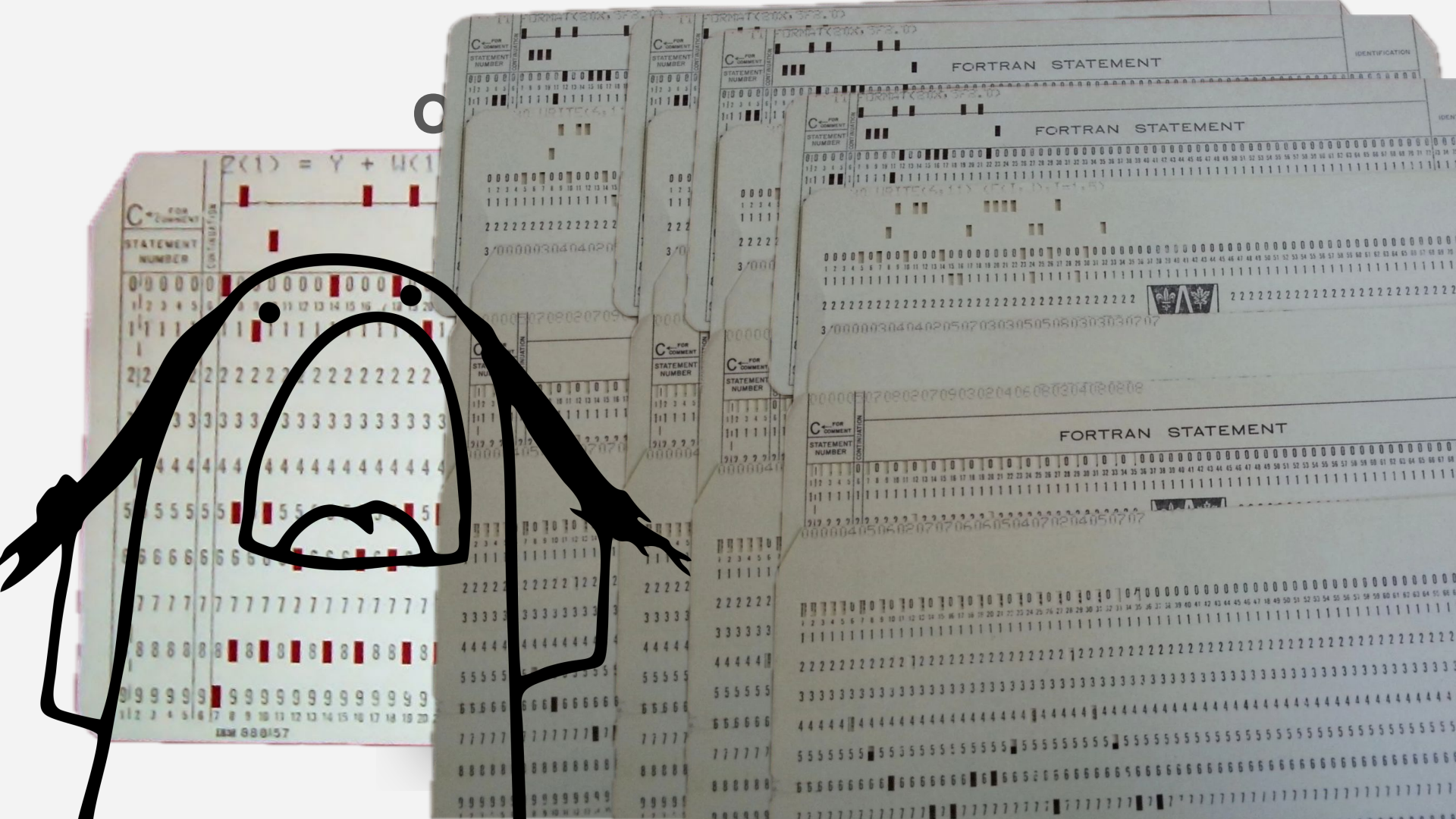
John McCarthy

O IBM704 já tinha Fortran

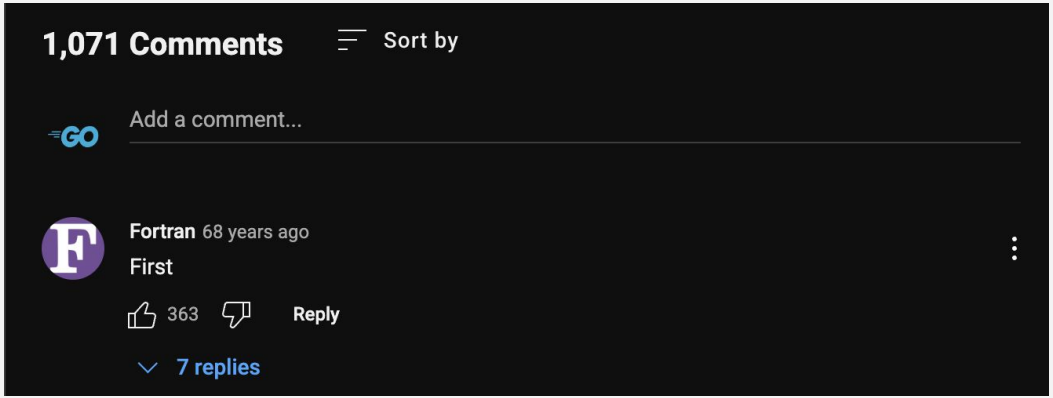
```
C  PROGRAM FOR FINDING THE LARGEST VALUE IN A SET
C  (ARRAY A CONTAINS N NUMBERS; RESULT IS LEFT IN BIGA)

      BIGA = A(1)

      DO 20 I = 2, N          C  LOOP OVER THE REST OF THE ARRAY

10    IF (BIGA - A(I)) 15, 20, 20   C  ARITHMETIC IF:
                                    C    <0  → 15
                                    C    =0  → 20
                                    C    >0  → 20

15    BIGA = A(I)             C  NEW MAXIMUM FOUND
      GOTO 20

20    CONTINUE                C  END OF LOOP

      WRITE (6,30) BIGA       C  PRINT THE ANSWER
30    FORMAT (1H , 'MAX VALUE =', 1PE14.7)

      STOP
      END
```

O IBM704 já tinha Fortran

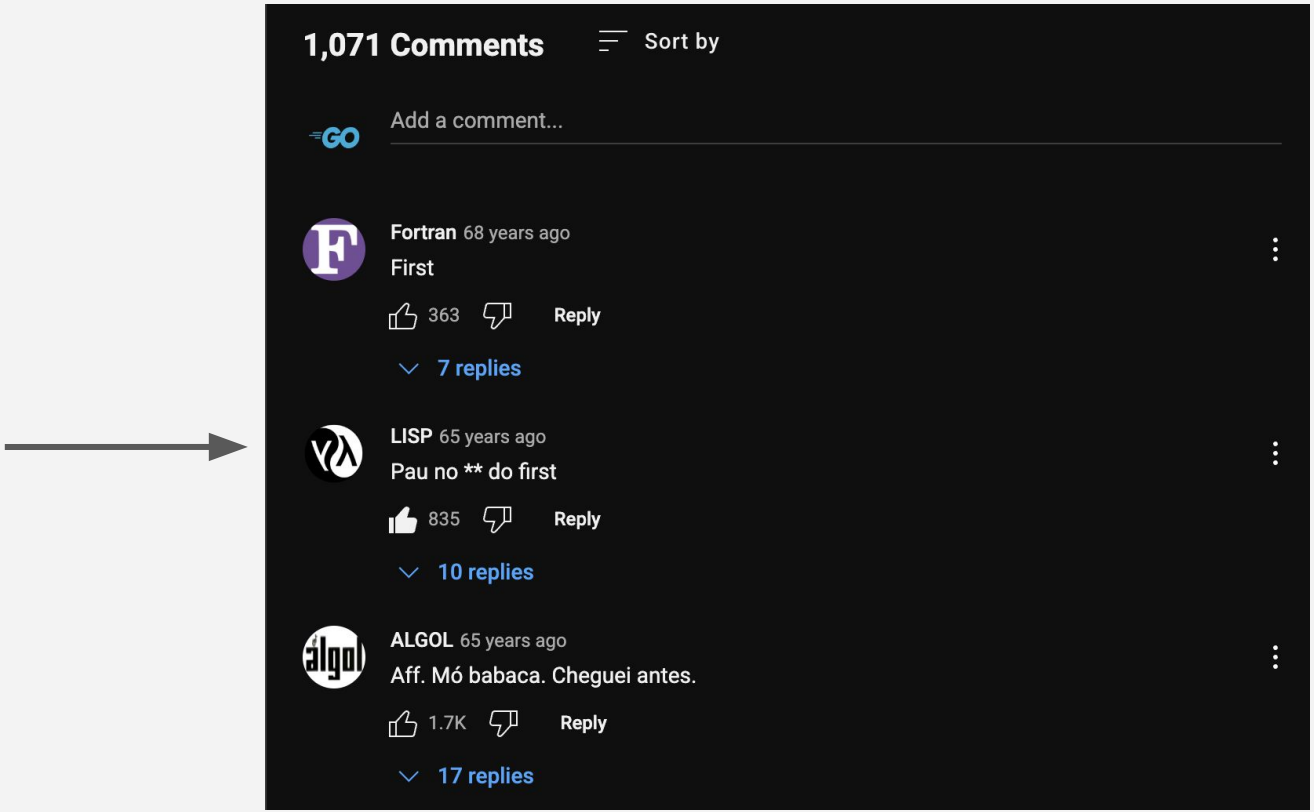



FORTRAN STATEMENT
FORTRAN STATEMENT
FORTRAN STATEMENT

Fortran utilizava apenas stack



John McCarthy decide criar uma linguagem para IA



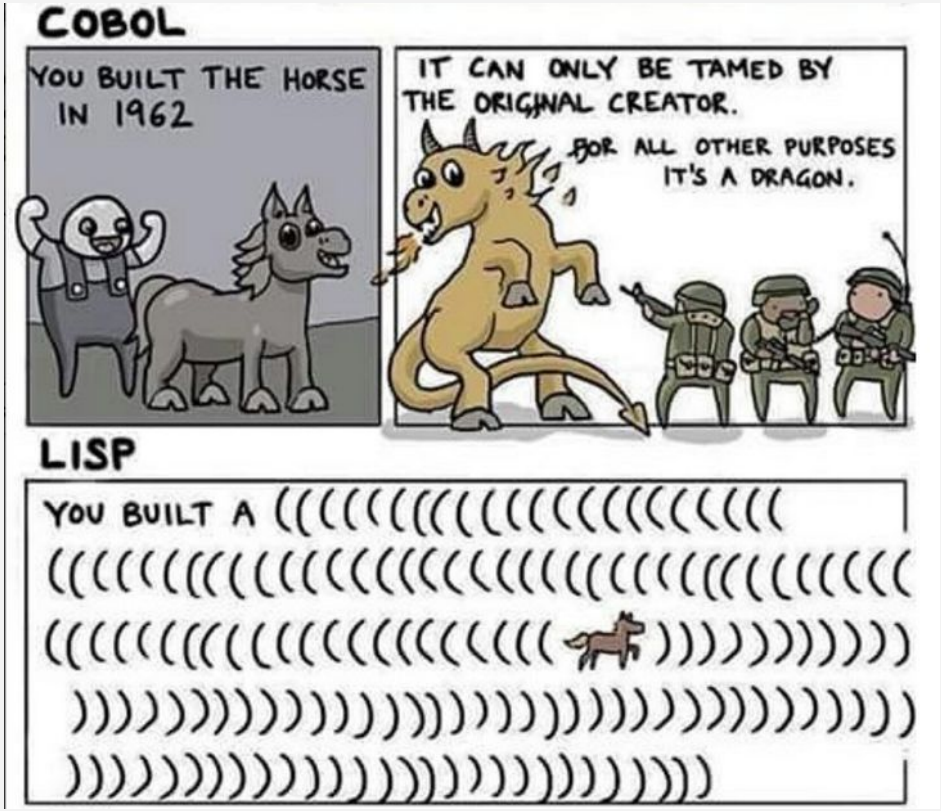
LISP

```
(DEFINE DIFF
        (LAMBDA (E X)
            (COND ((ATOM E)         ; number or variable
                   (COND ((EQ E X) 1) (T 0)))
                  ((EQ (CAR E) 'PLUS)         ; (PLUS u v)
                   (LIST 'PLUS
                         (DIFF (CADR E) X)
                         (DIFF (CADDR E) X)))
                  ((EQ (CAR E) 'TIMES)        ; (TIMES u v)
                   (LIST 'PLUS
                         (LIST 'TIMES
                               (DIFF (CADR E) X)
                               (CADDR E))
                         (LIST 'TIMES
                               (CADR E)
                               (DIFF (CADDR E) X))))
                  (T 'UNSUPPORTED))))

; Differentiate (x+3)*(x+4) with respect to x:
(DIFF '(TIMES (PLUS X 3) (PLUS X 4)) 'X)
```

LISP

```
(((((define  fact)))))
((((((((((  (= n 0))))))
((((((((((((((  t))))))
((((((((((((((fact ))))))
((((((((((((((fact 1))))))
((((((((((((((fact))))))
(((((((((((((())))))))))
((((((((((((((())))
(((((())))
```

Lisp stands for “Lots of Irritating Superfluous Parentheses”



Clojure é um dialeto de Lisp

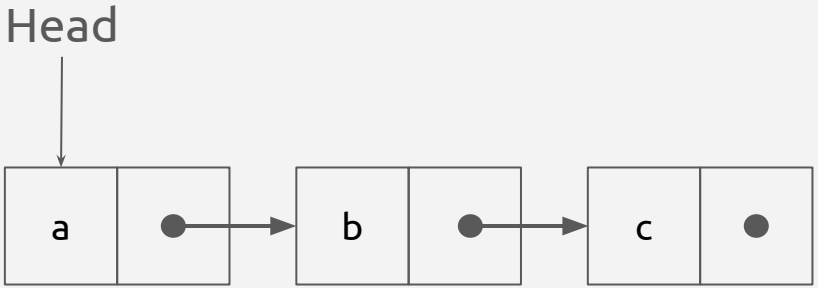
Clojure é um dialeto de Lisp

Clojure é um dialeto de Lisp

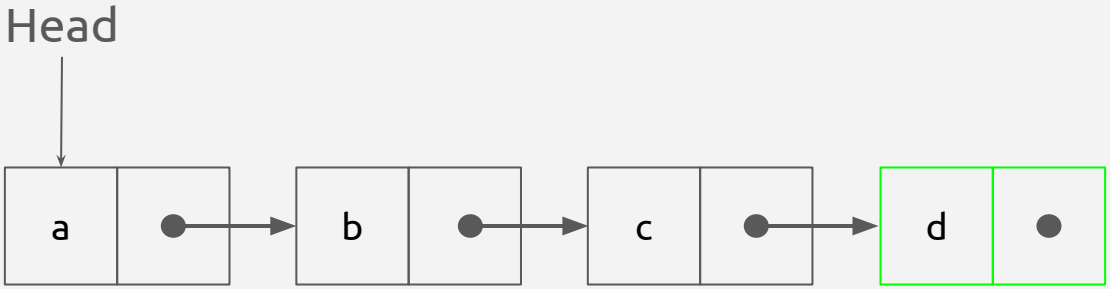
Walmart 

NETFLIX

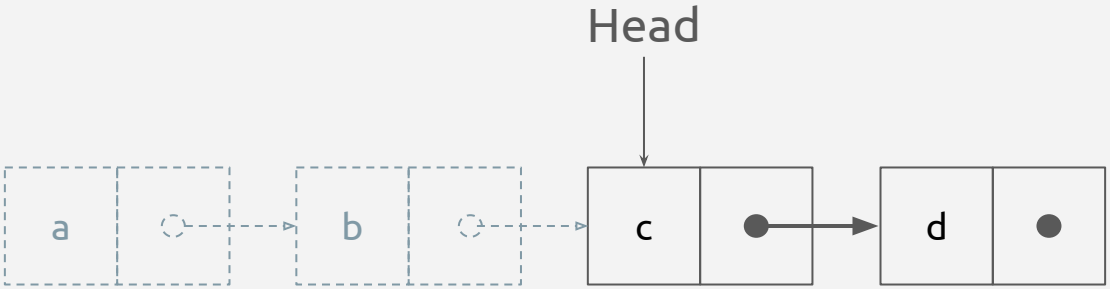
LISt Processor



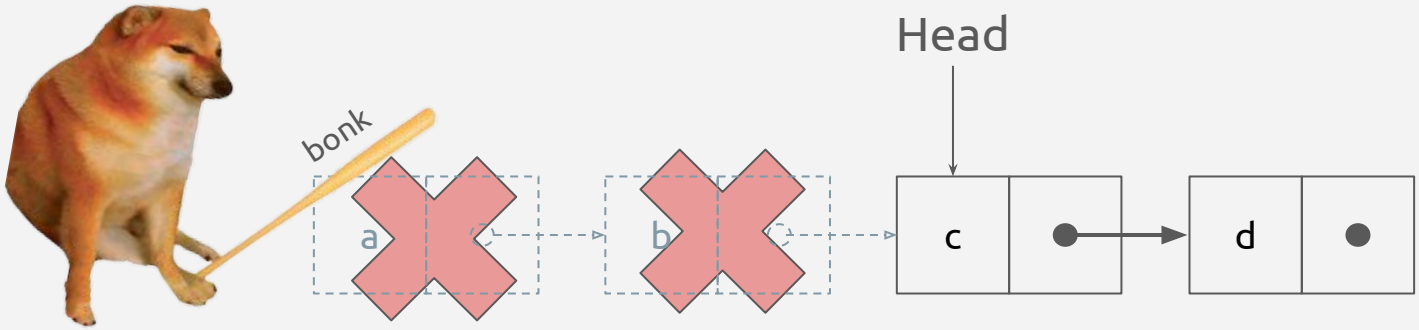
LISt Processor



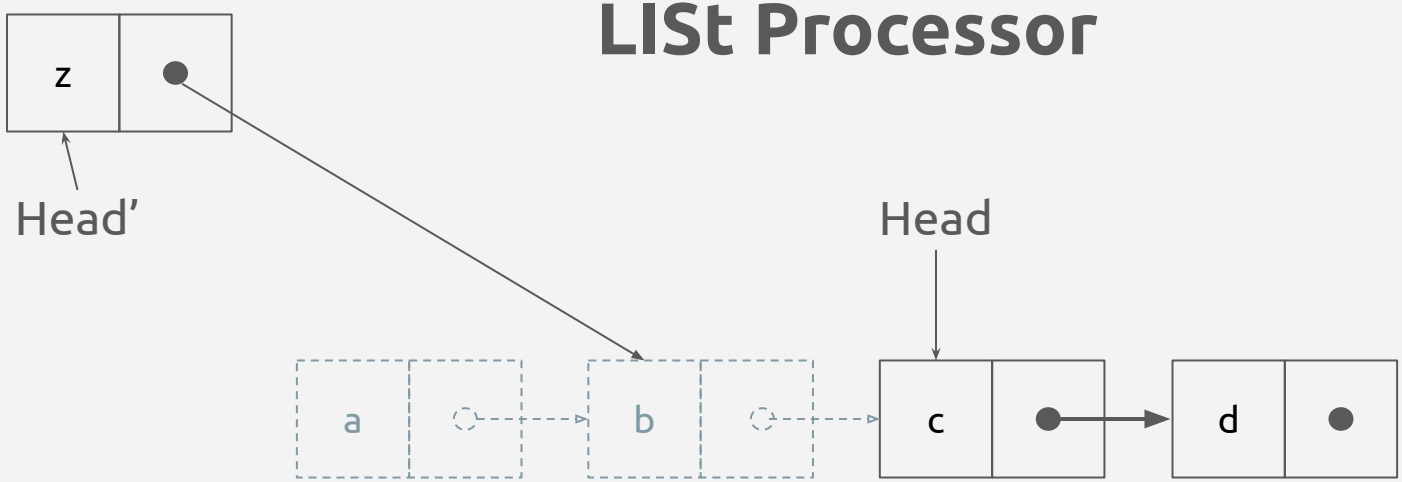
LISt Processor



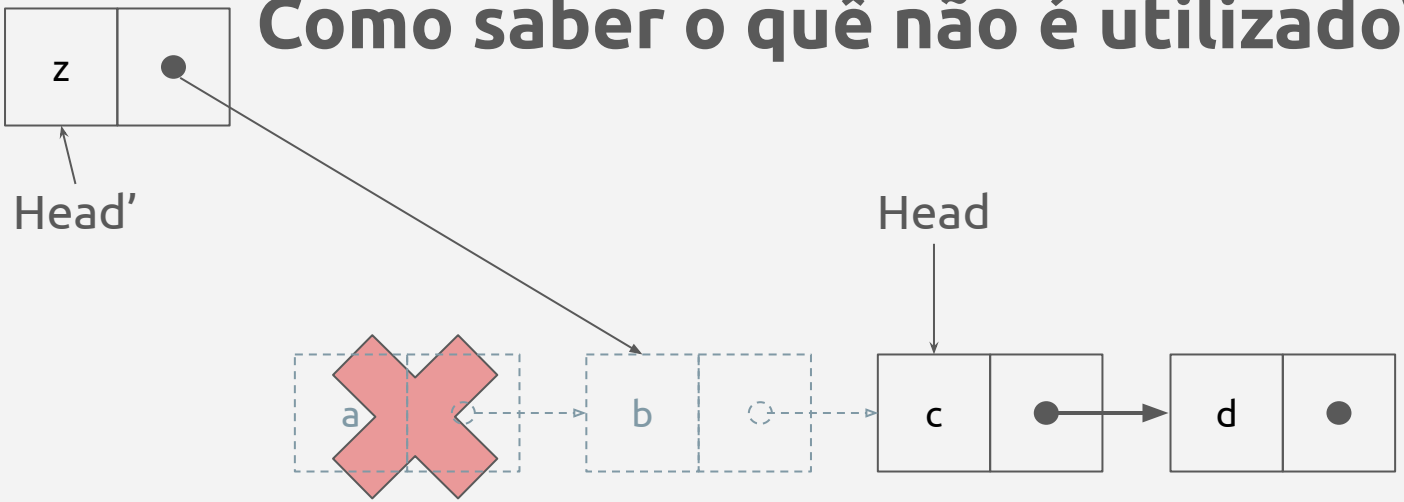
LISt Processor



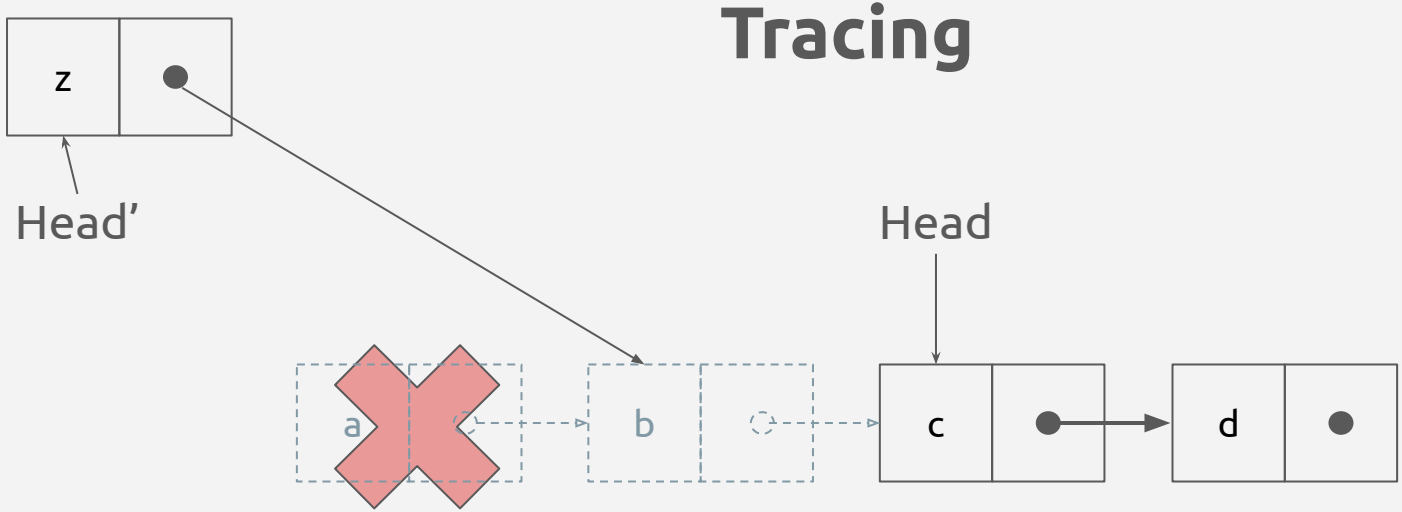
LISt Processor



Como saber o quê não é utilizado?



Tracing



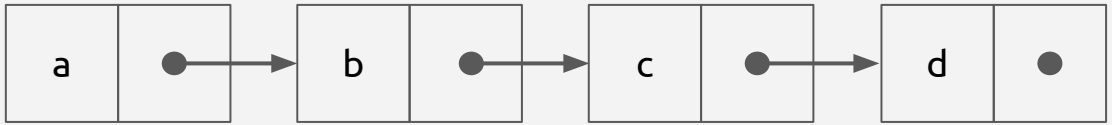
Tracing

Roots

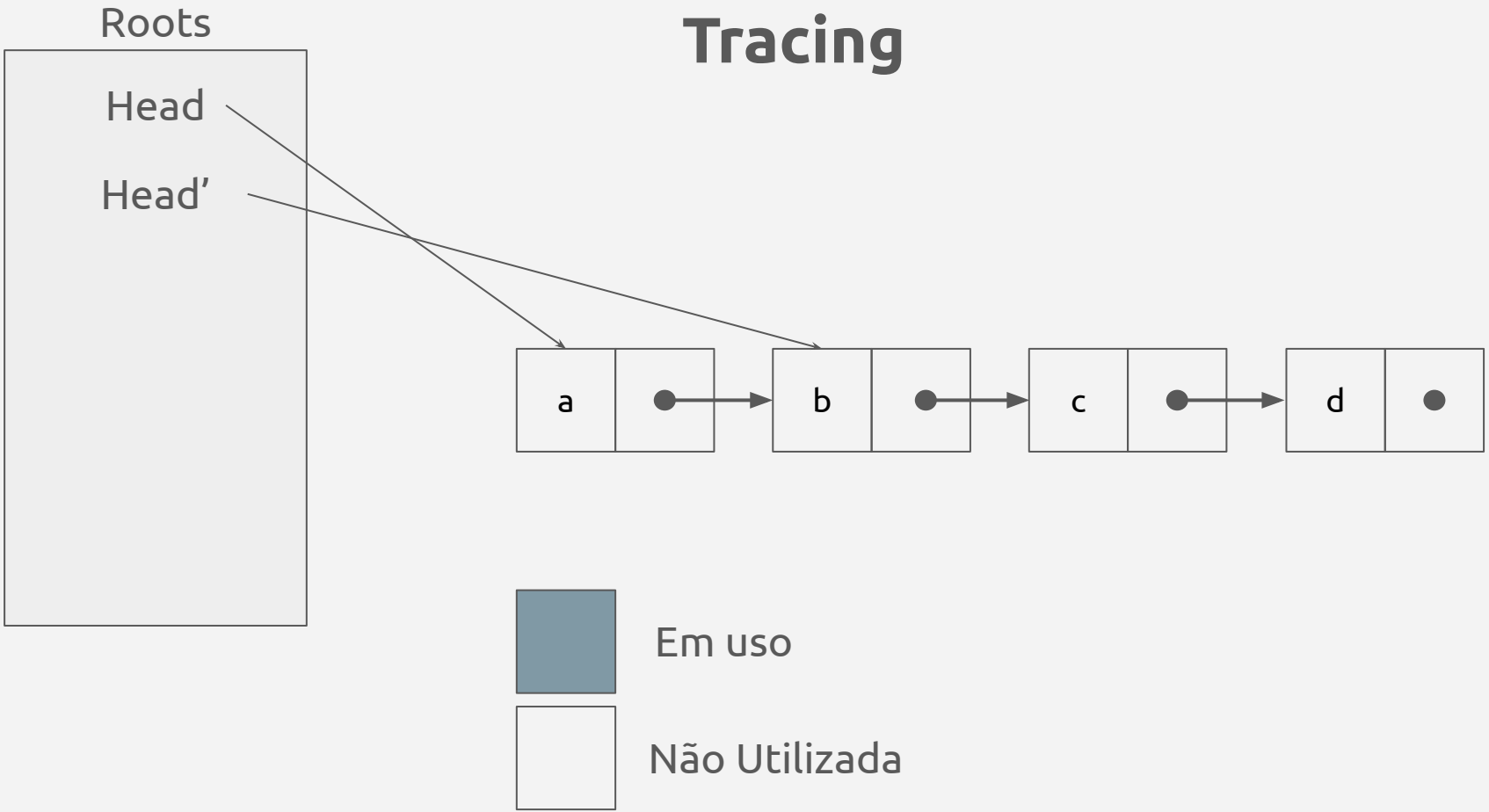
Head

Head’

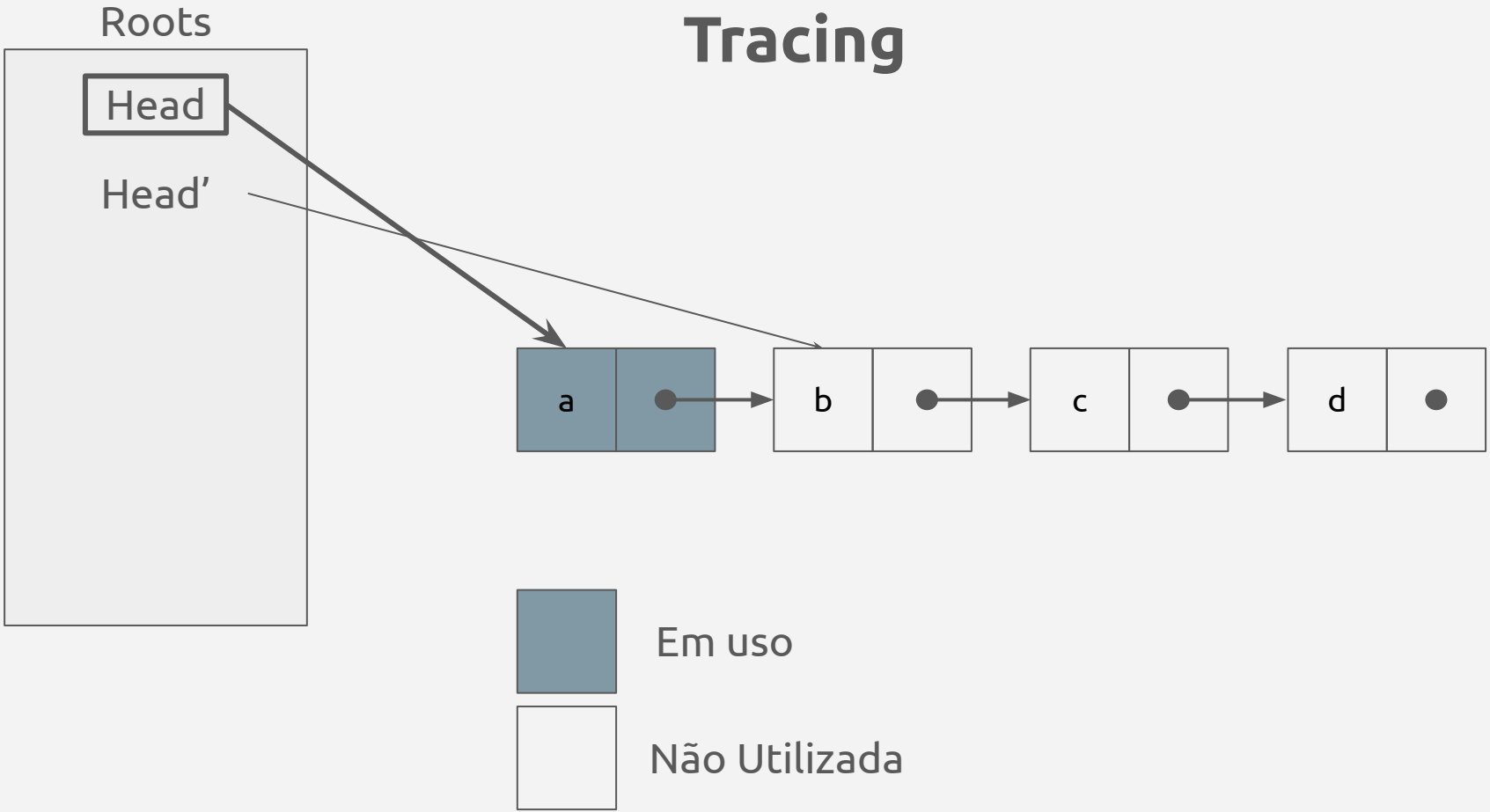
Variáveis declaradas (Stack)



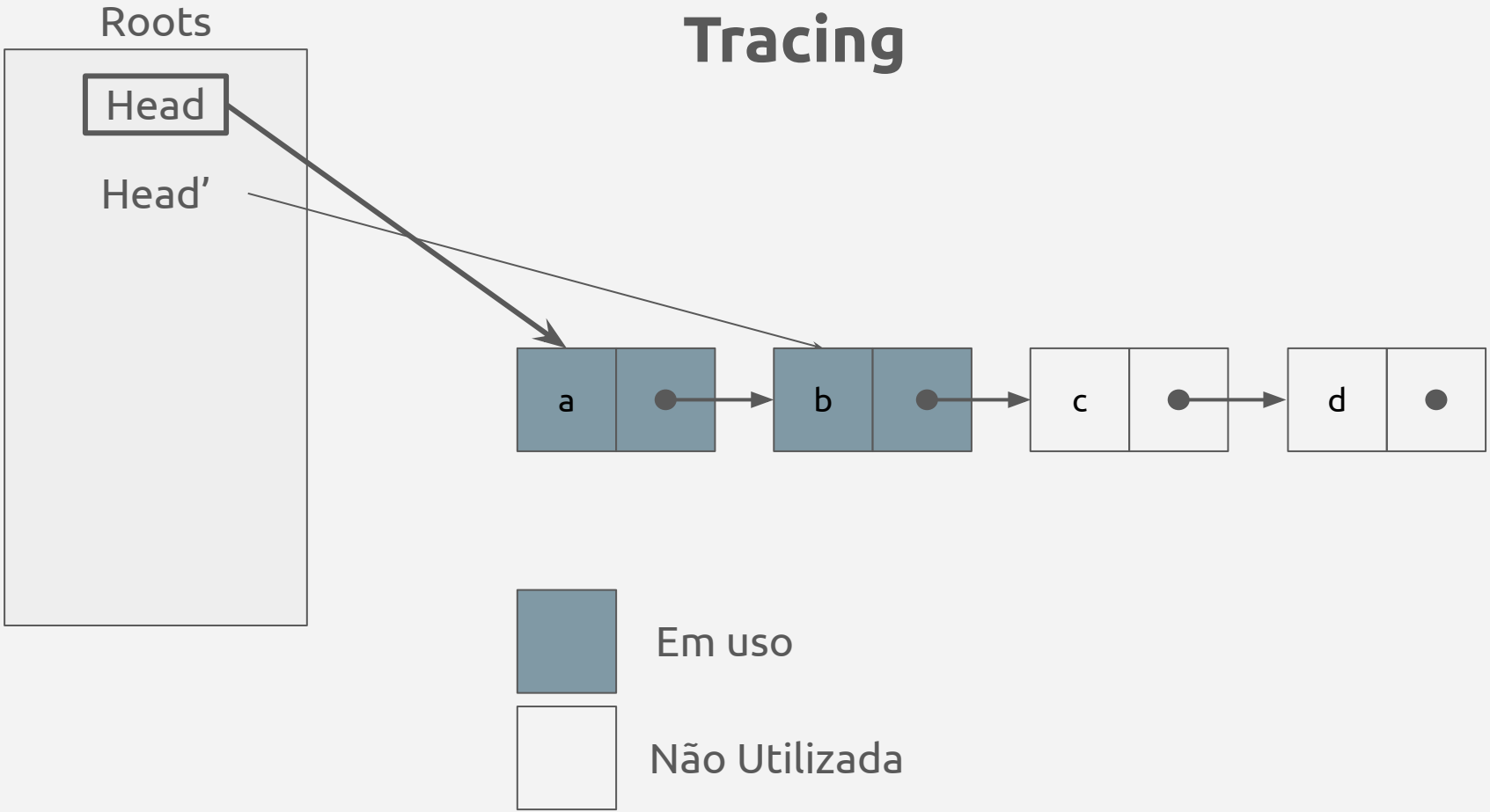
Tracing



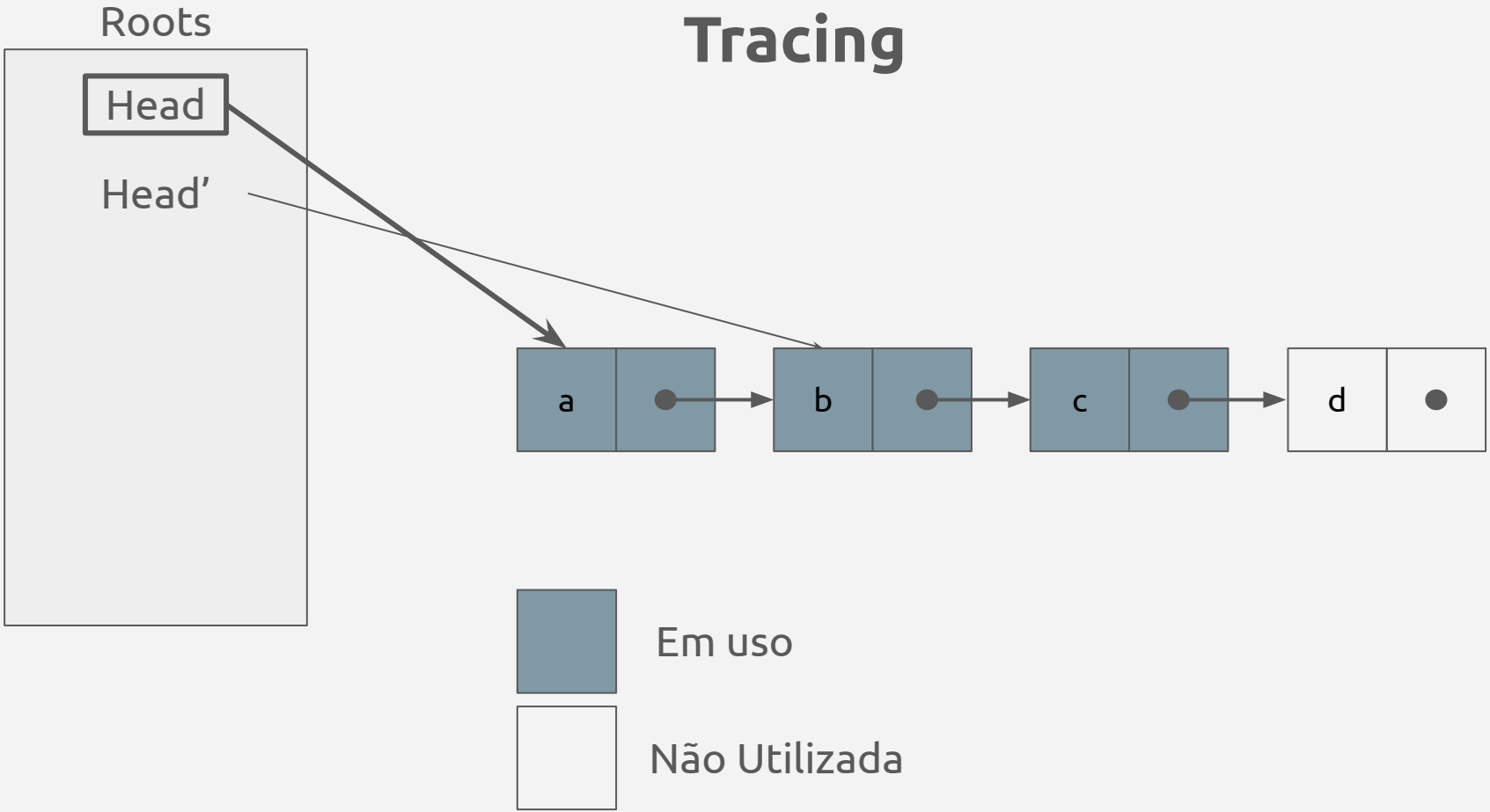
Tracing



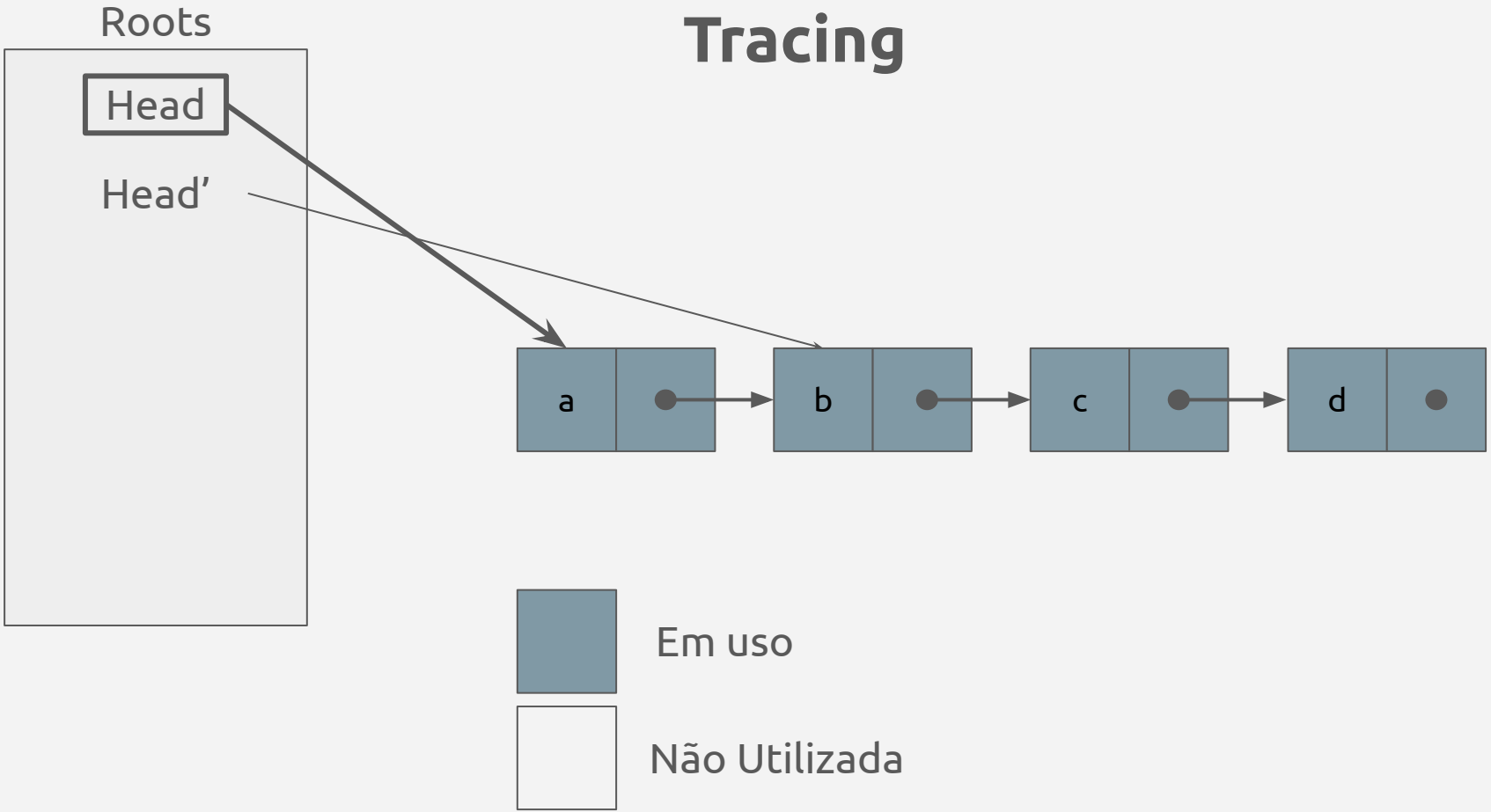
Tracing



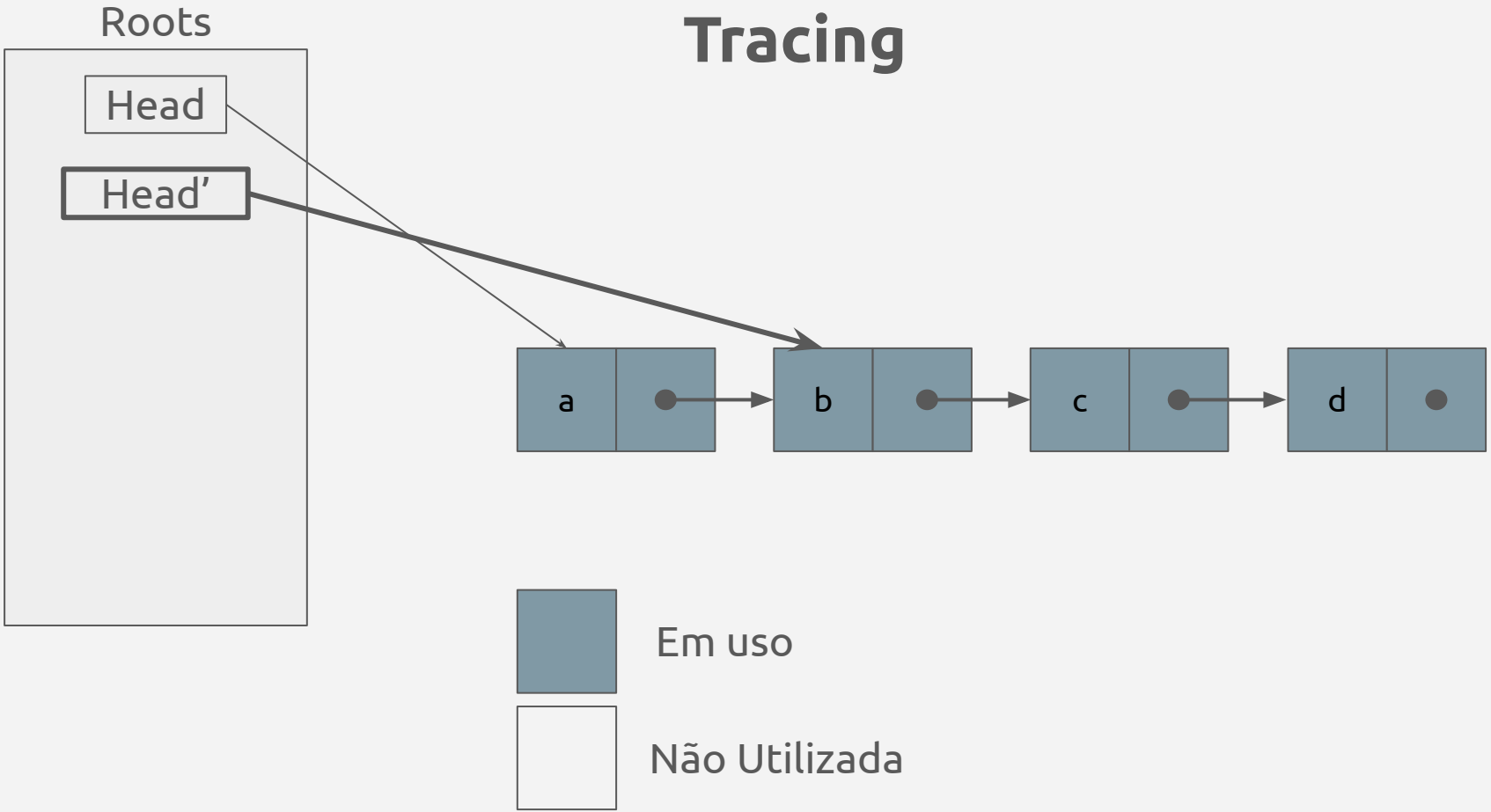
Tracing



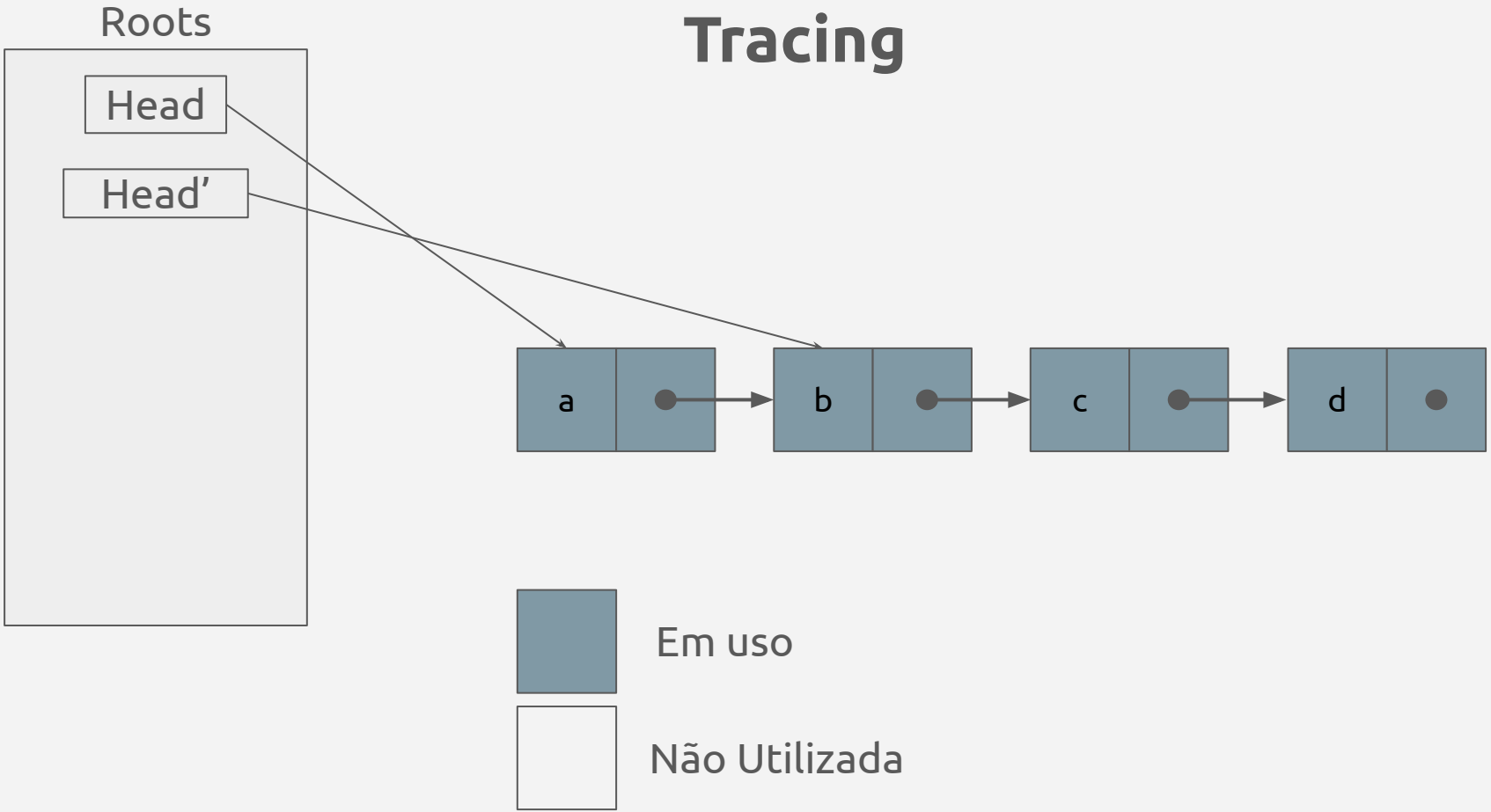
Tracing



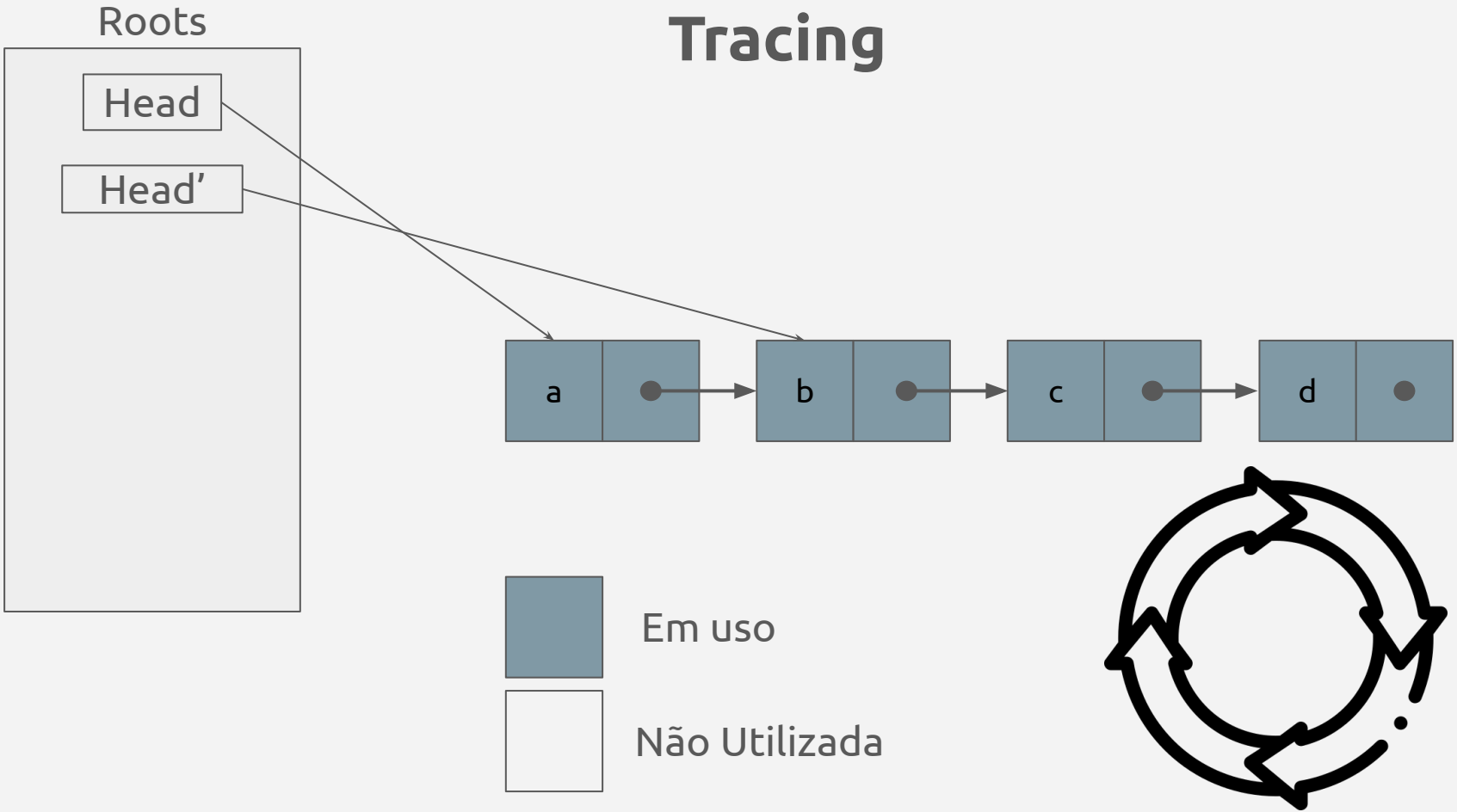
Tracing



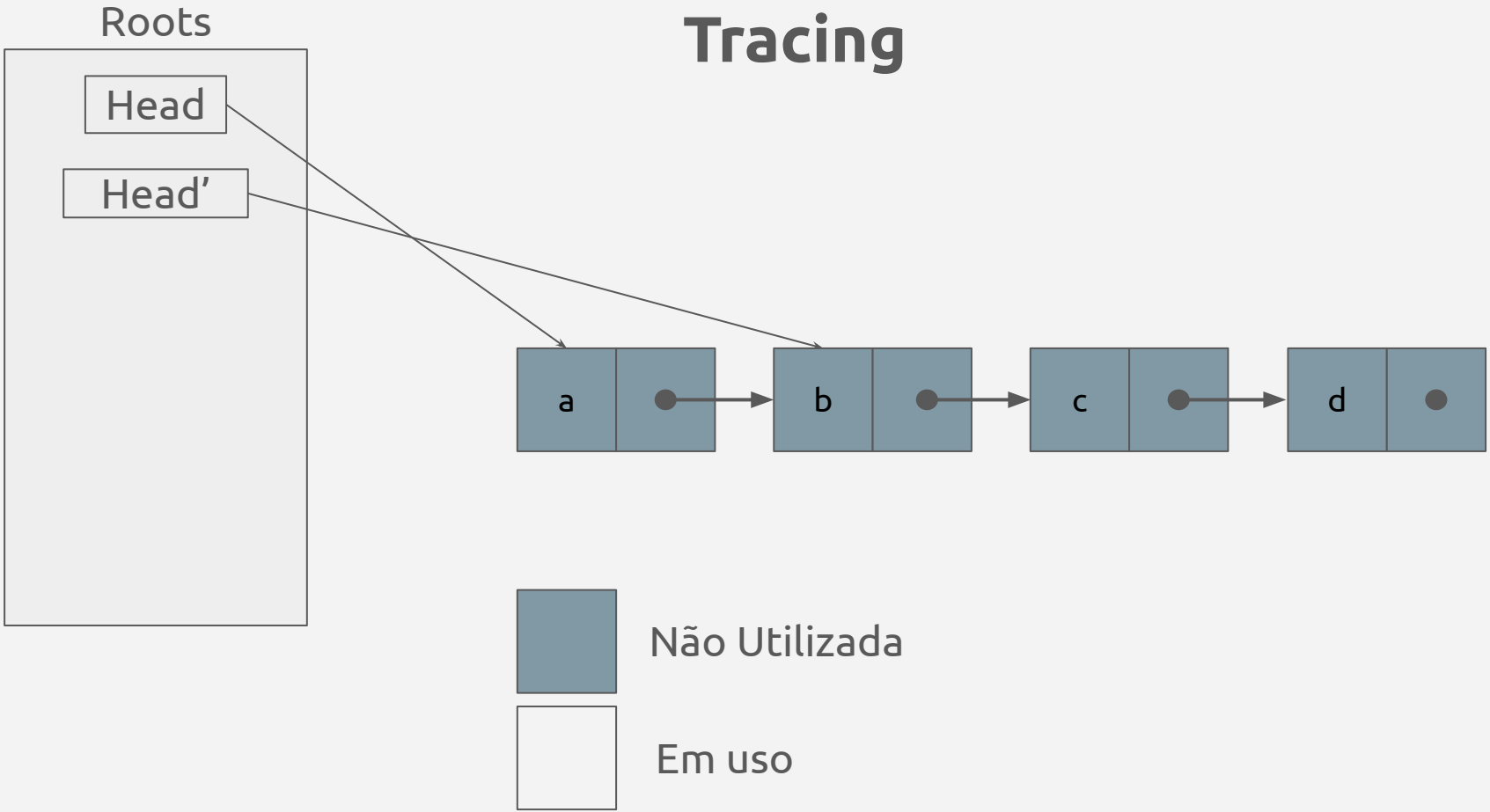
Tracing



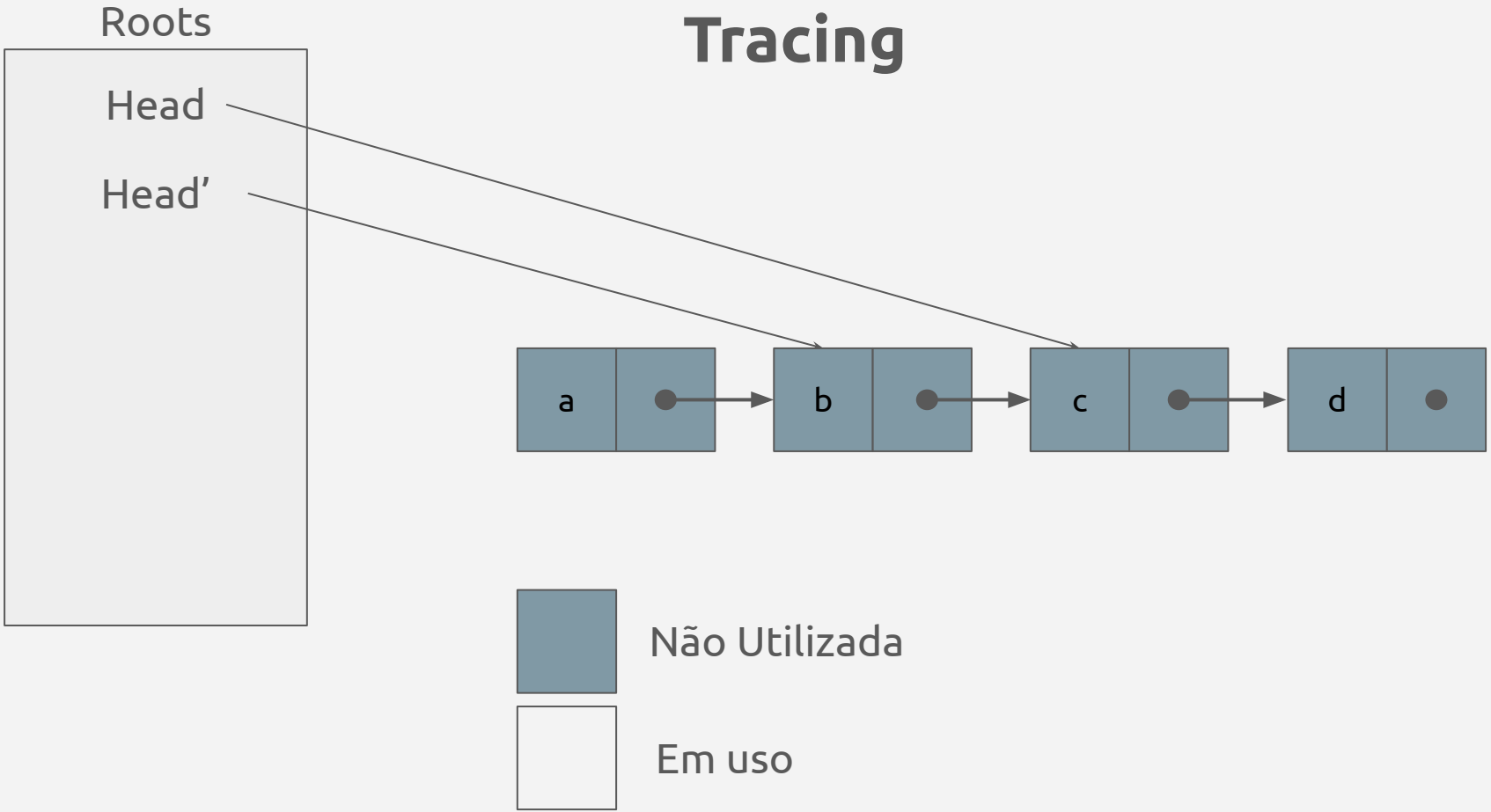
Tracing



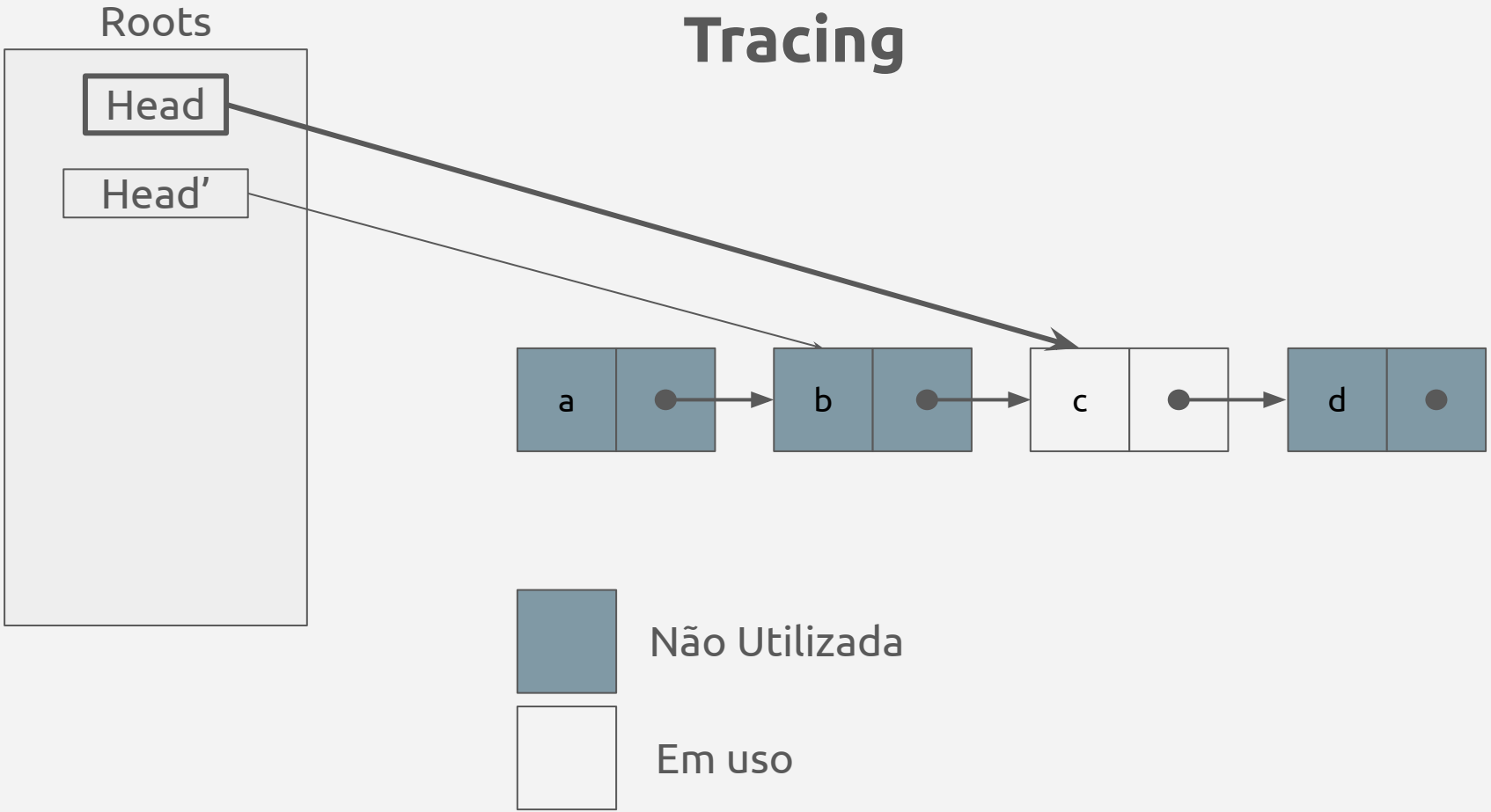
Tracing



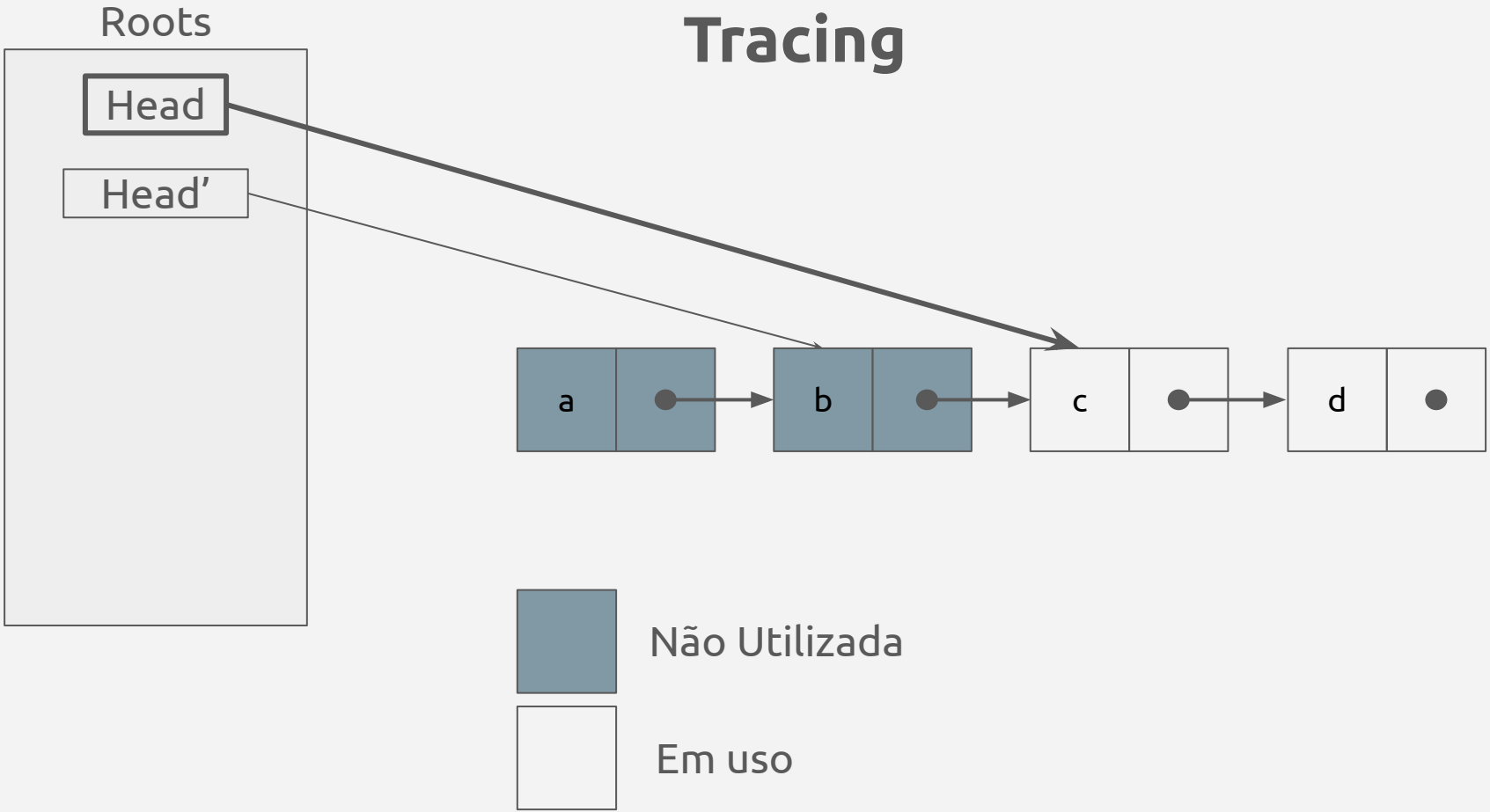
Tracing



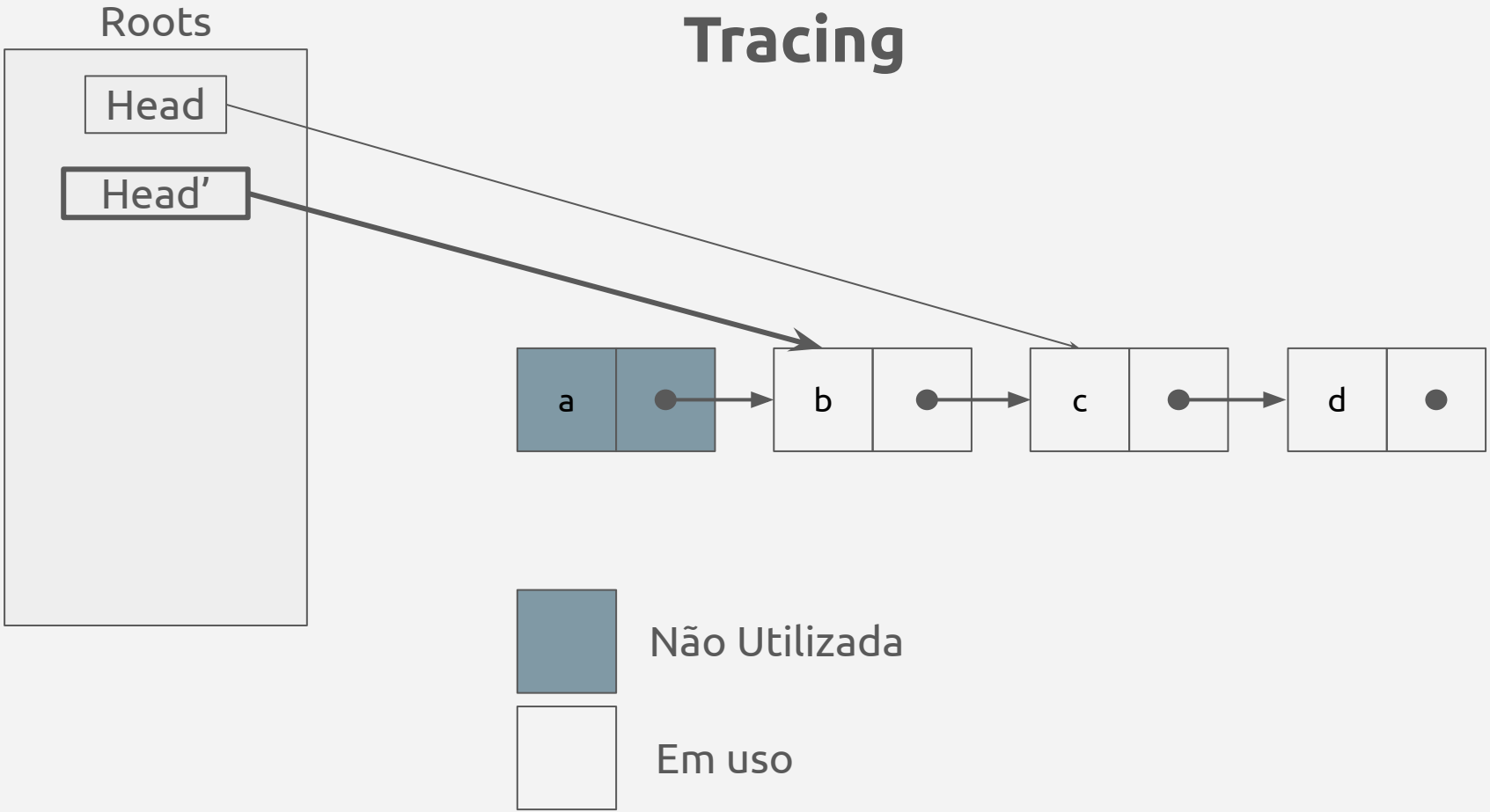
Tracing



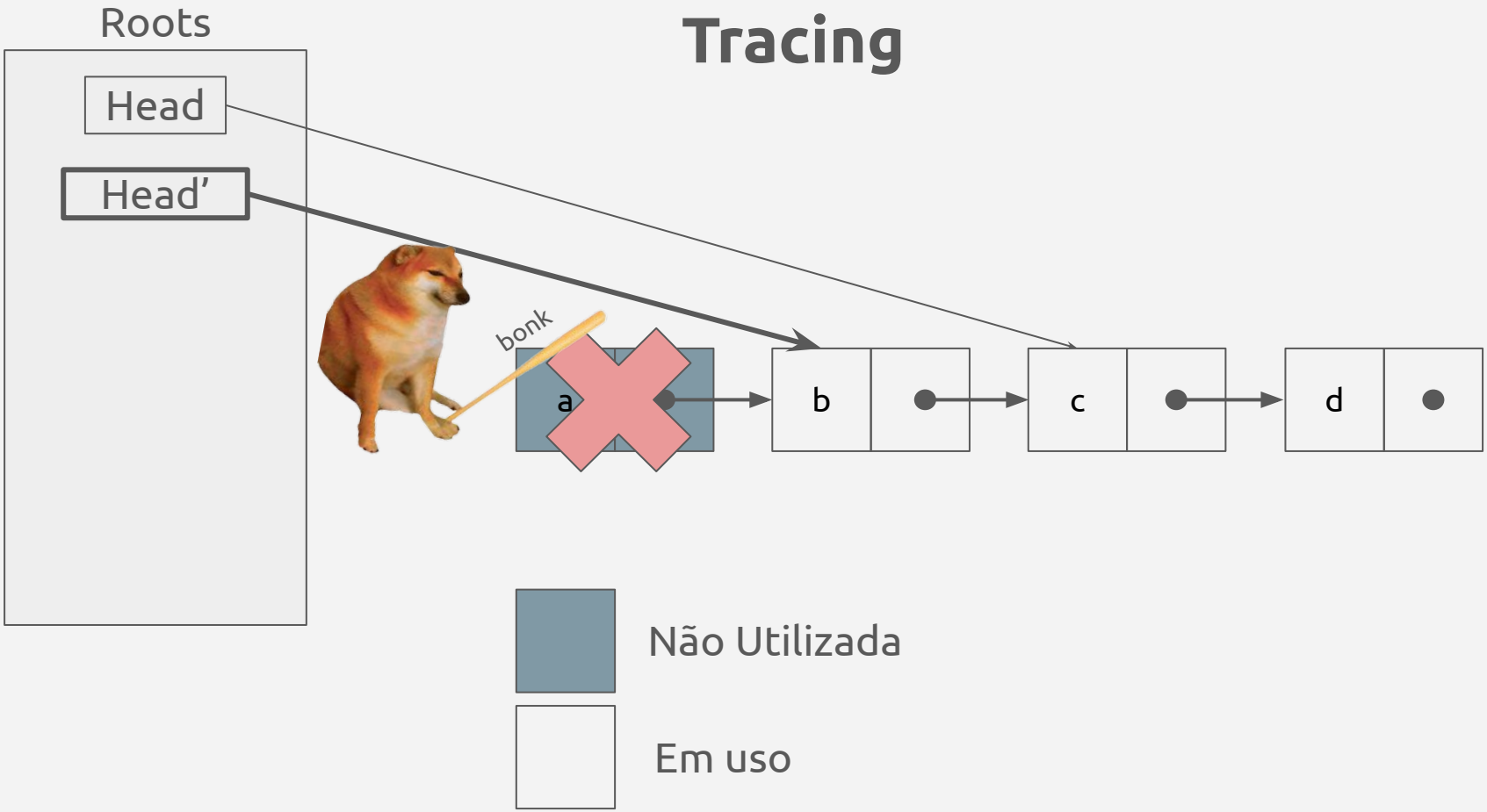
Tracing



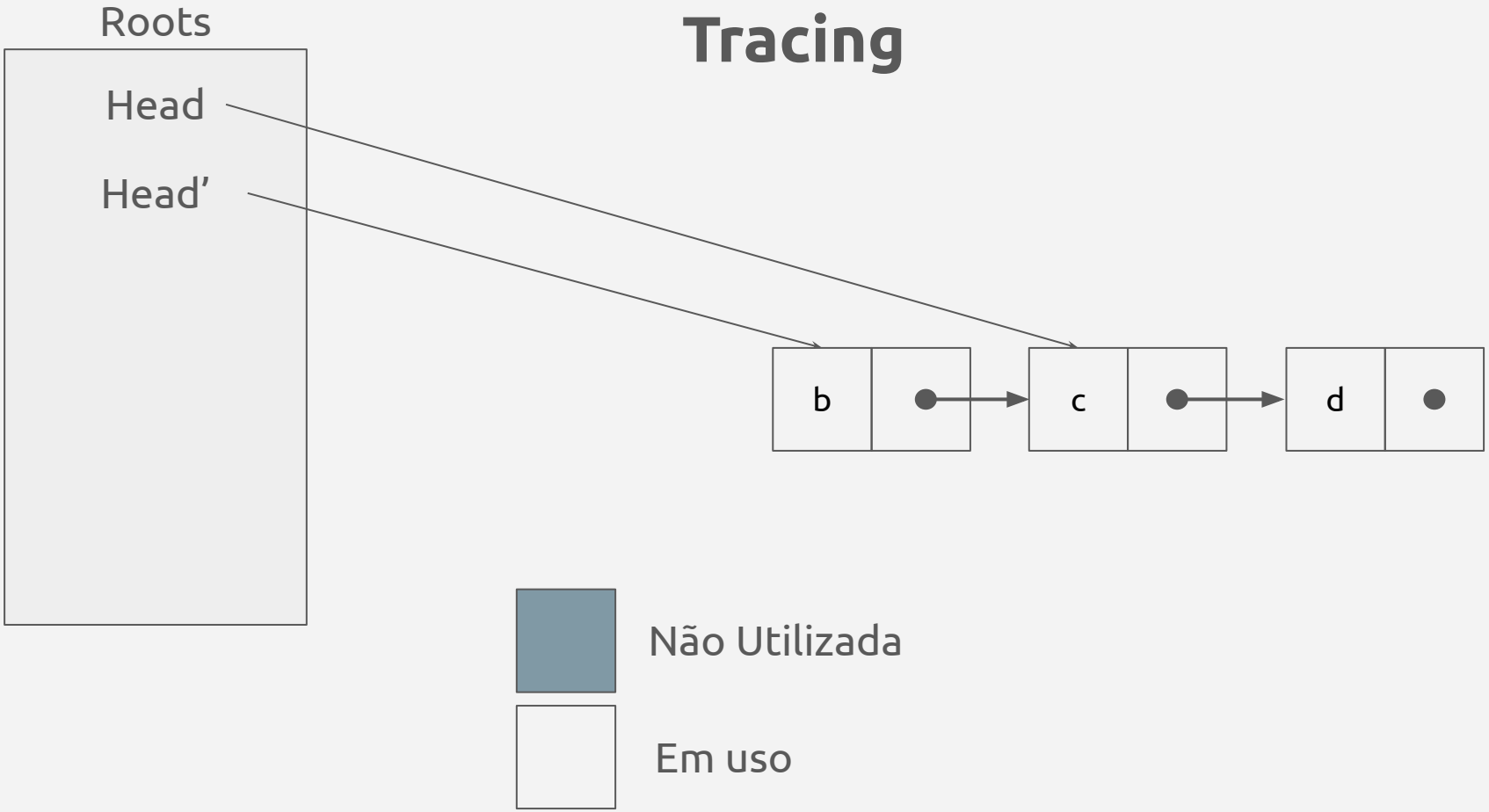
Tracing



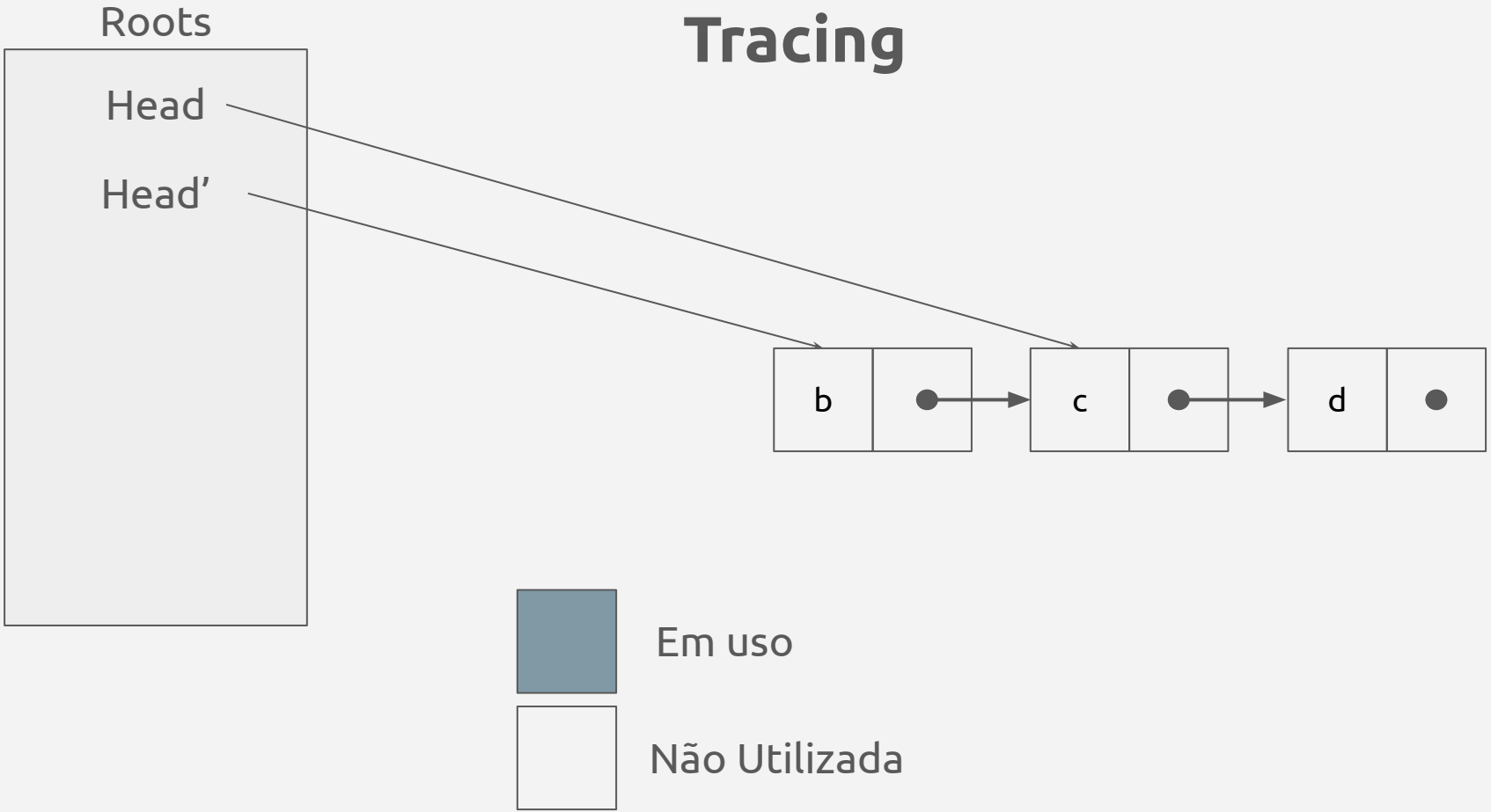
Tracing



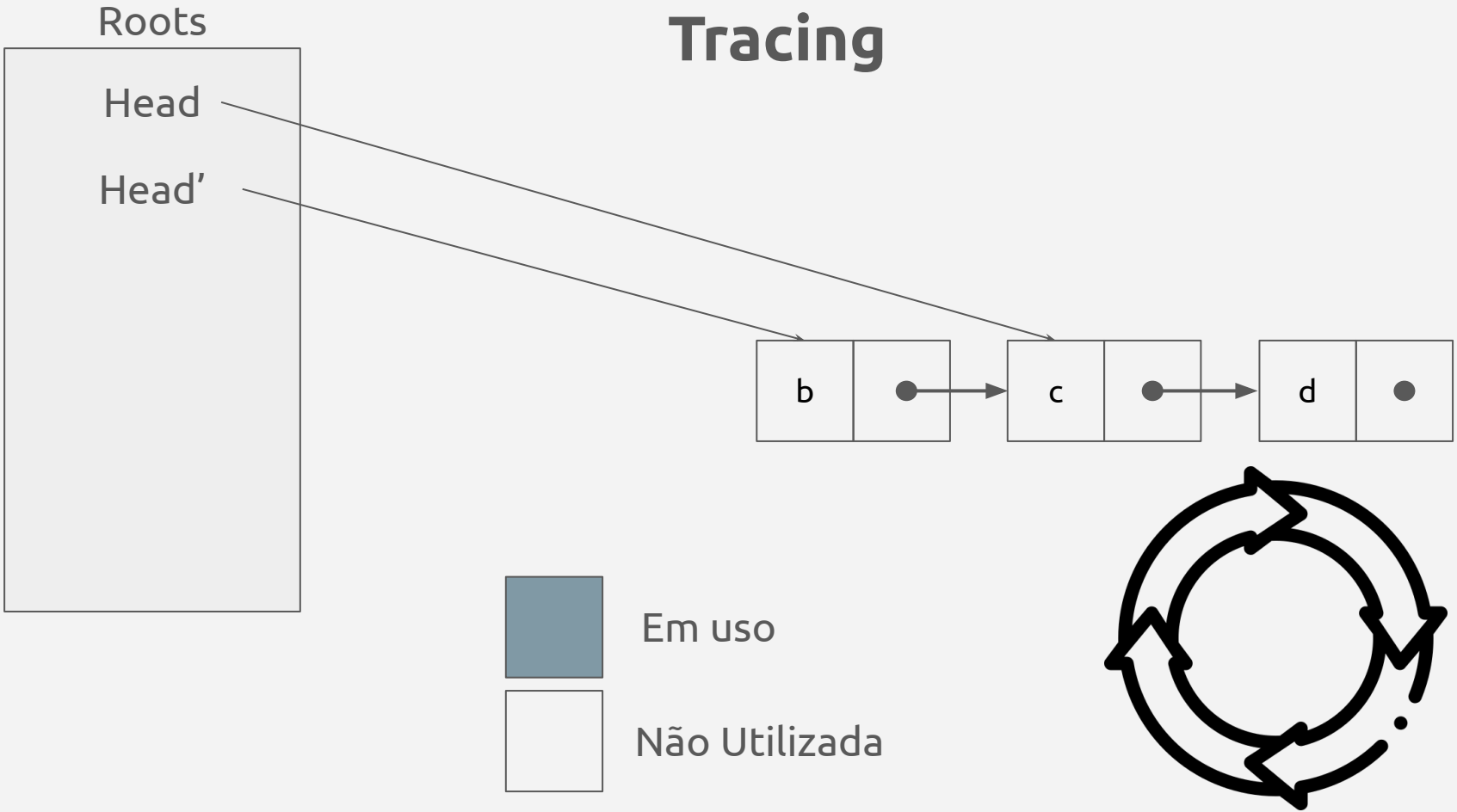
Tracing



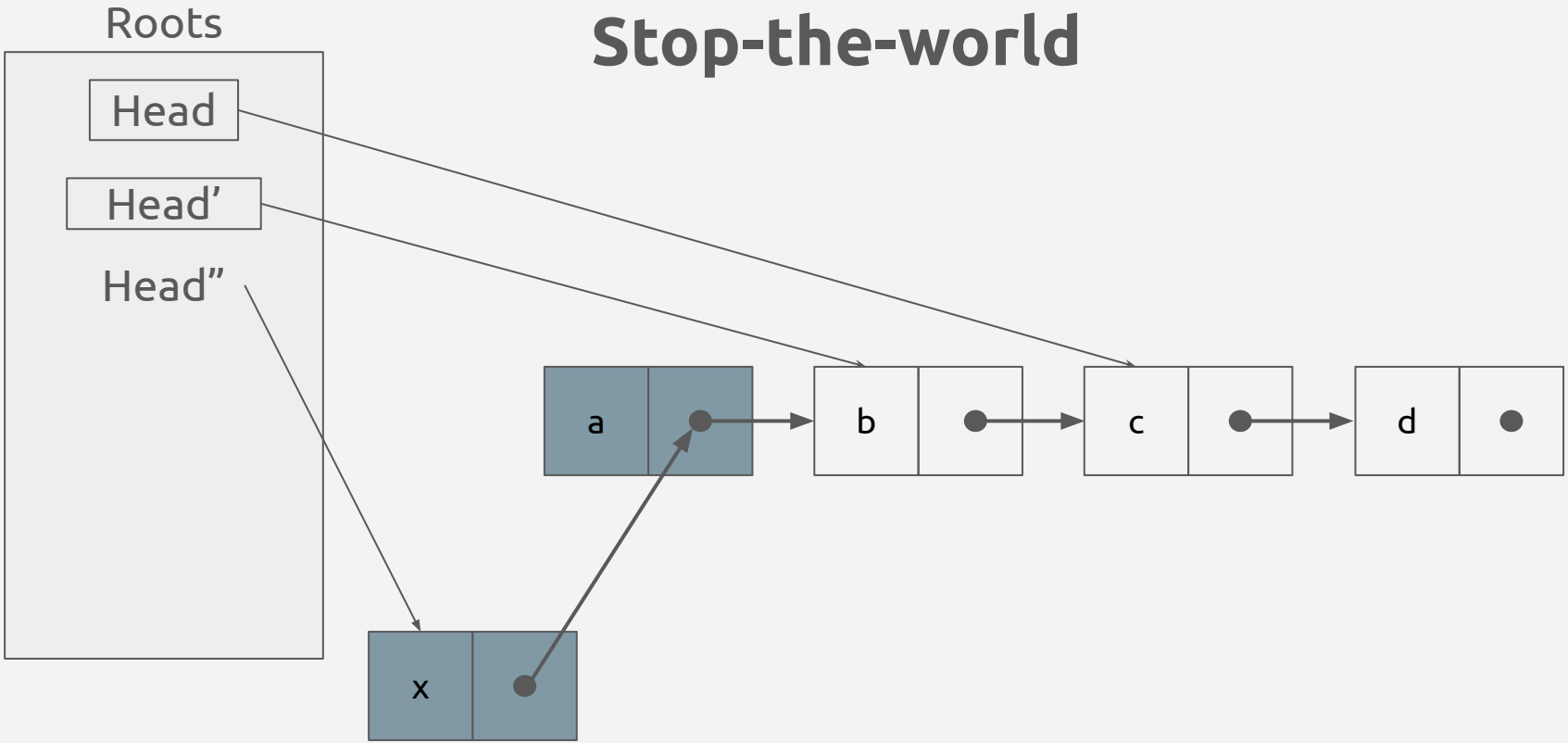
Tracing



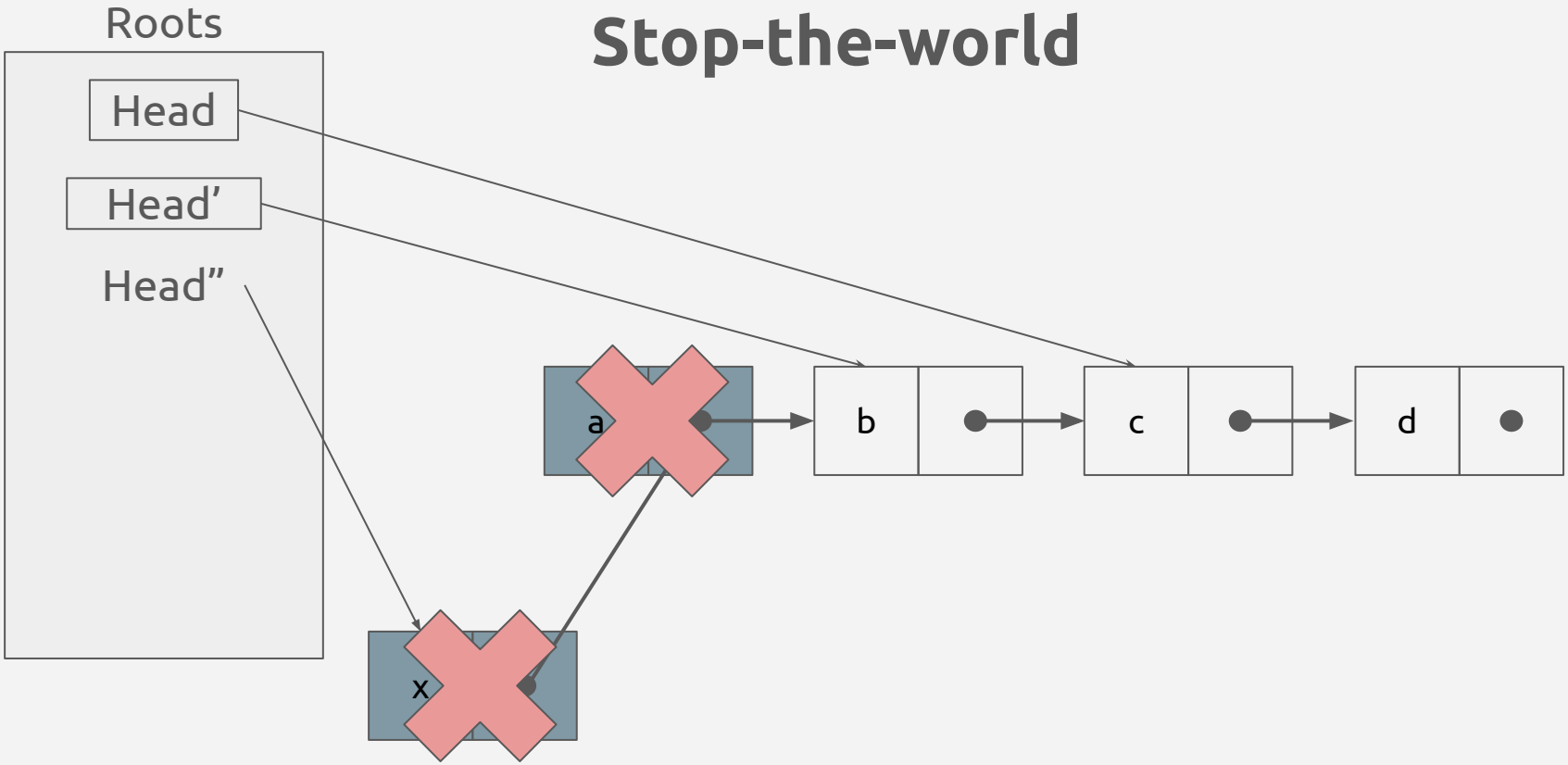
Tracing



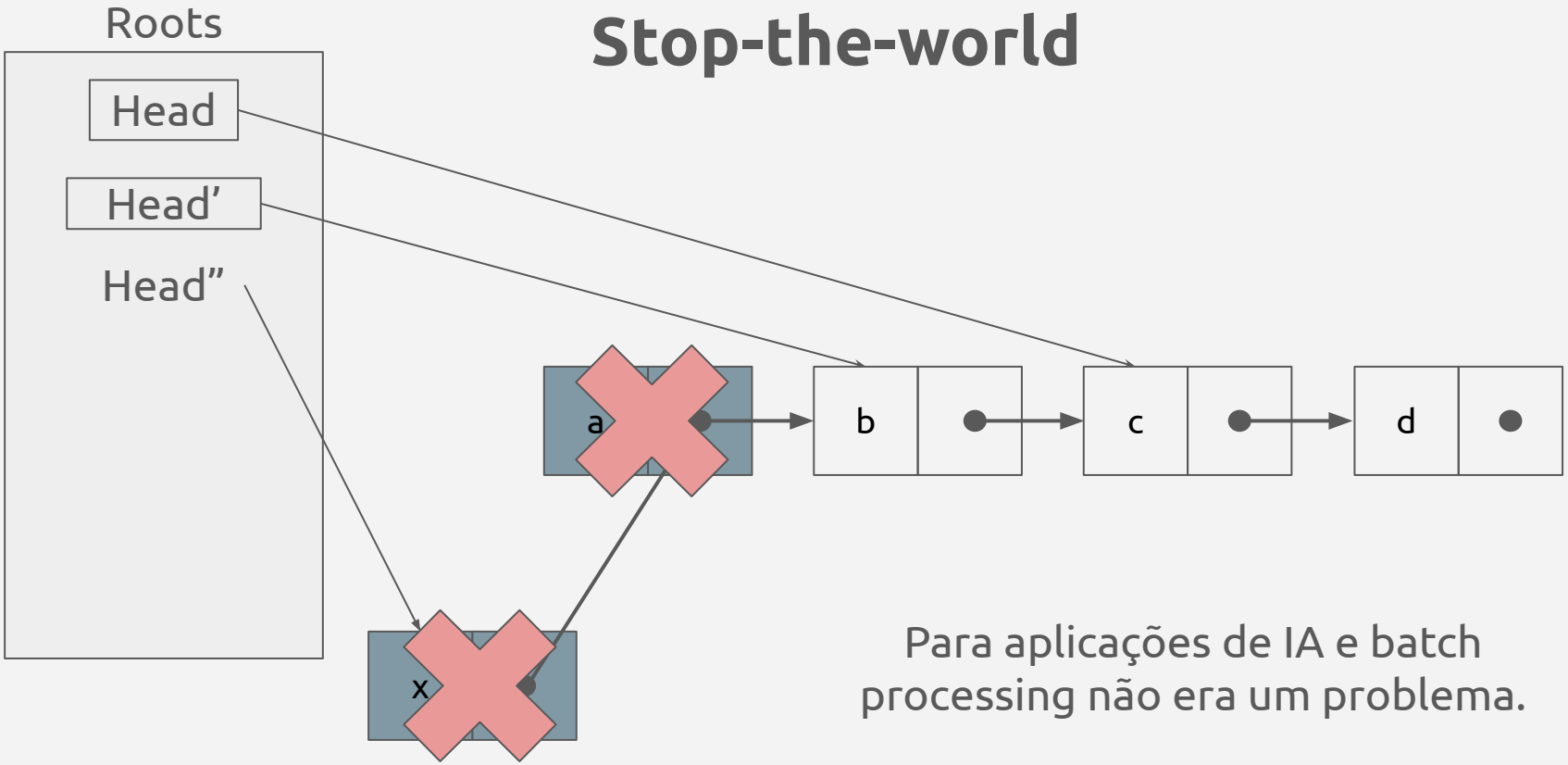
Stop-the-world



Stop-the-world

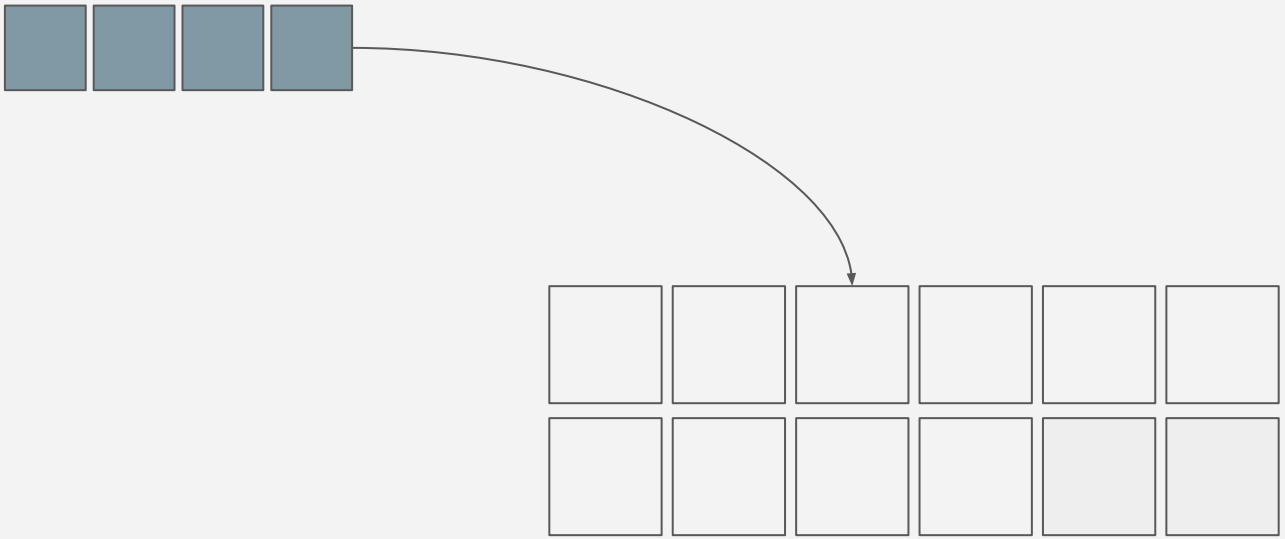


Stop-the-world



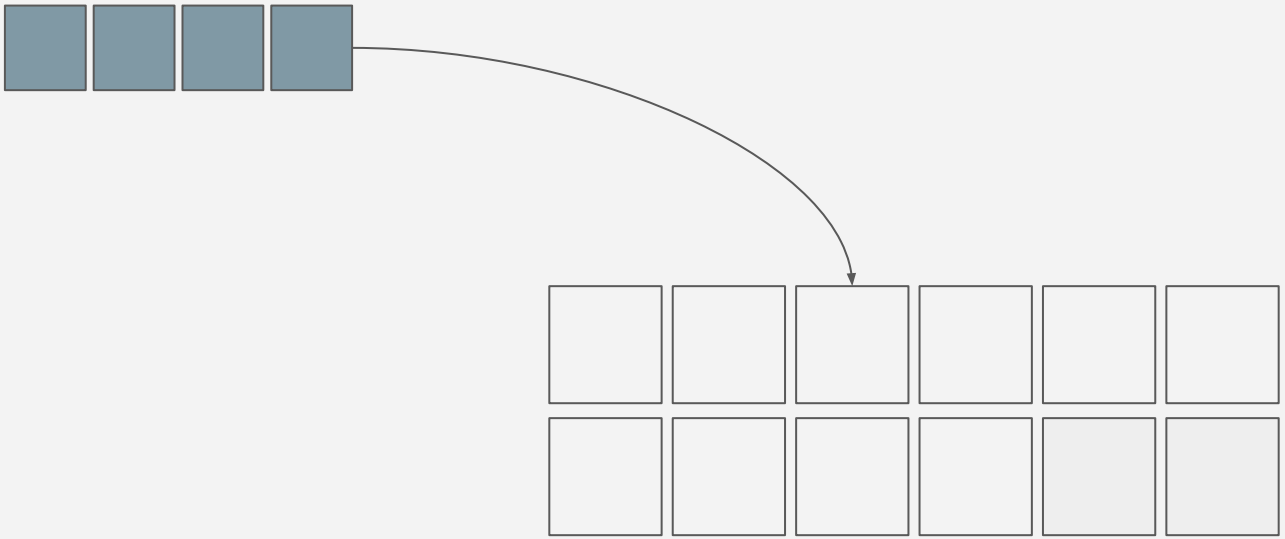
Para aplicações de IA e batch processing não era um problema.

Outro problema…



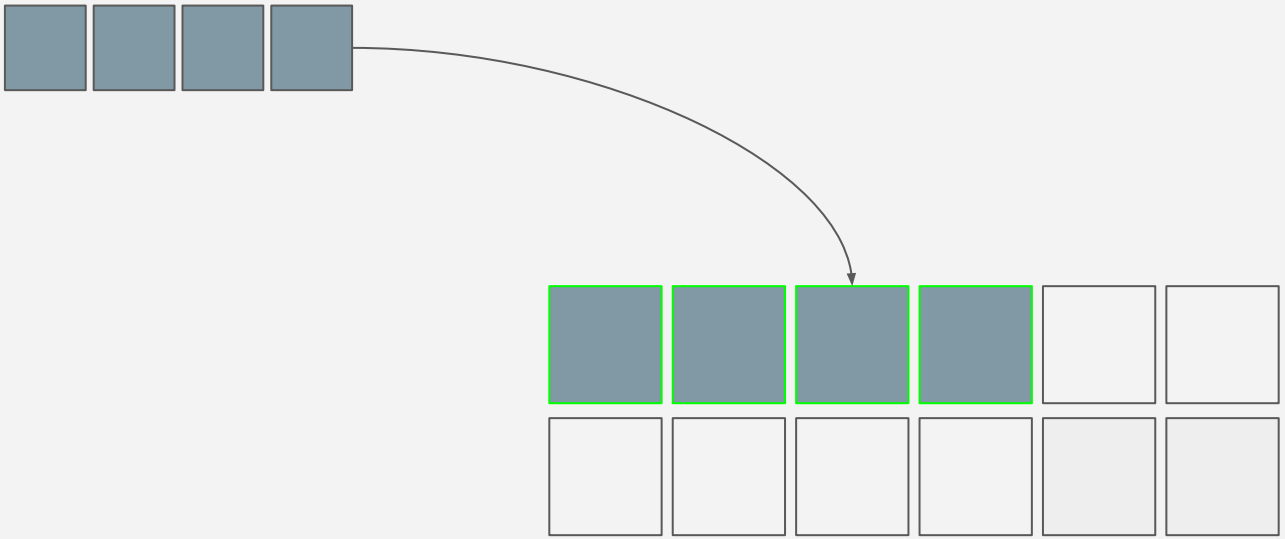
Disponível: 12

Alocando memória



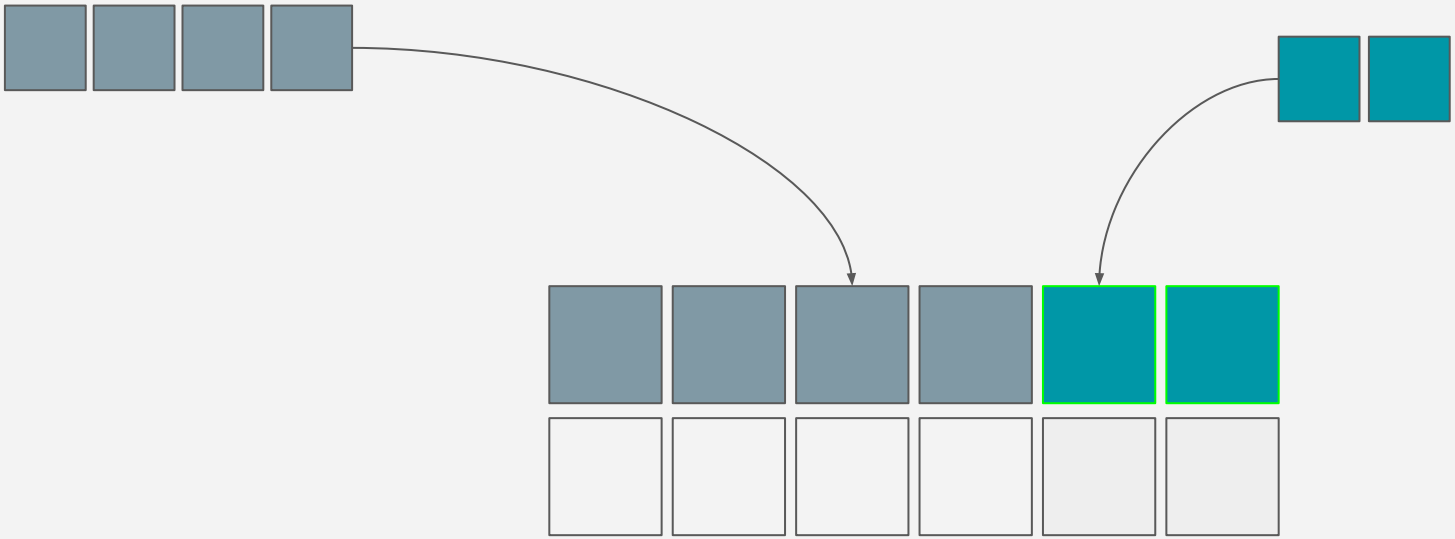
Disponível: 12

Alocando memória



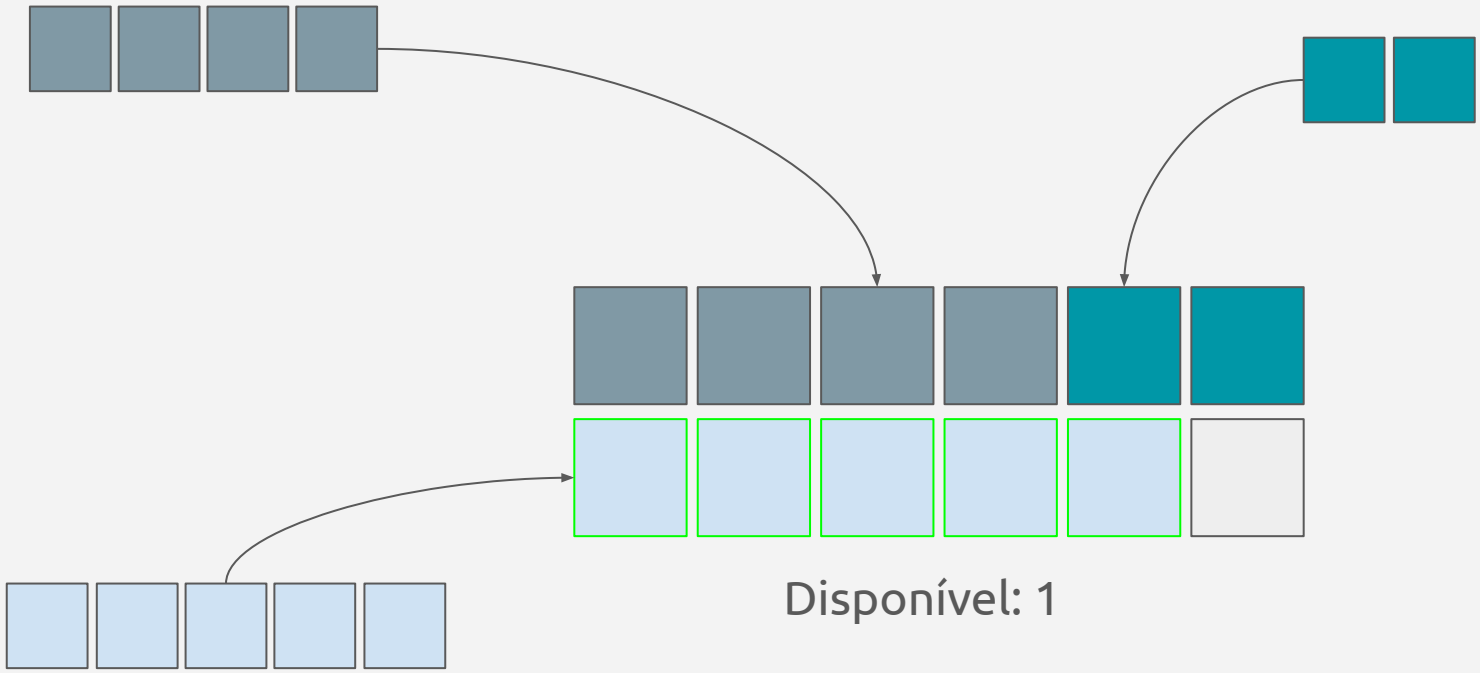
Disponível: 8

Alocando memória



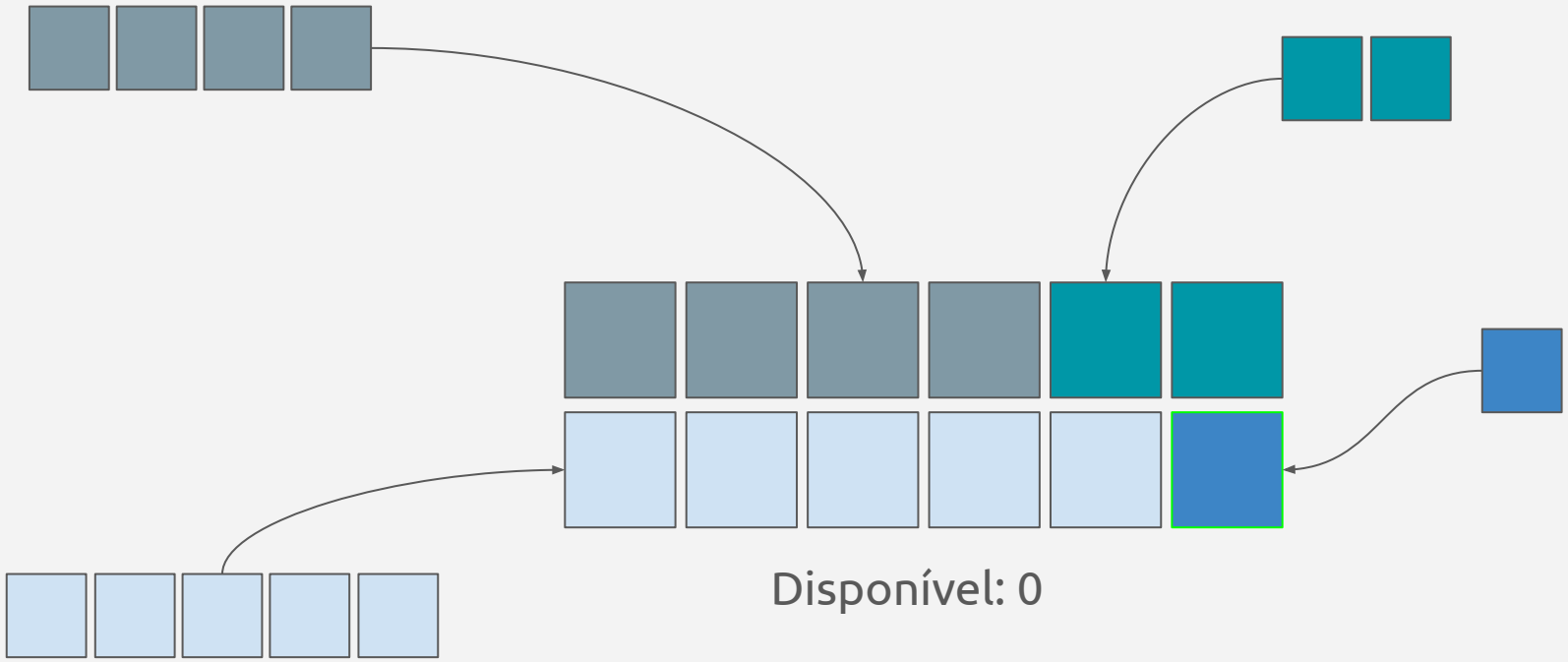
Disponível: 6

Alocando memória



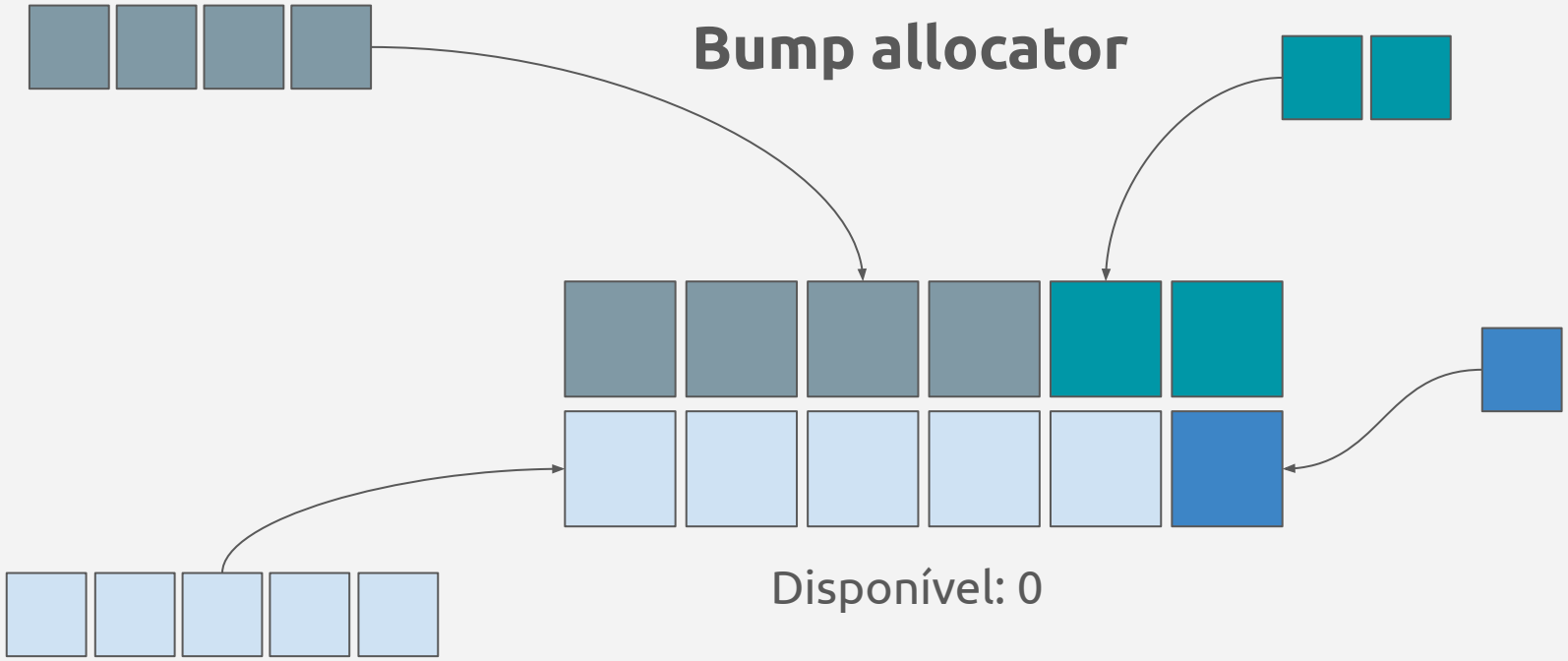
Disponível: 1

Alocando memória

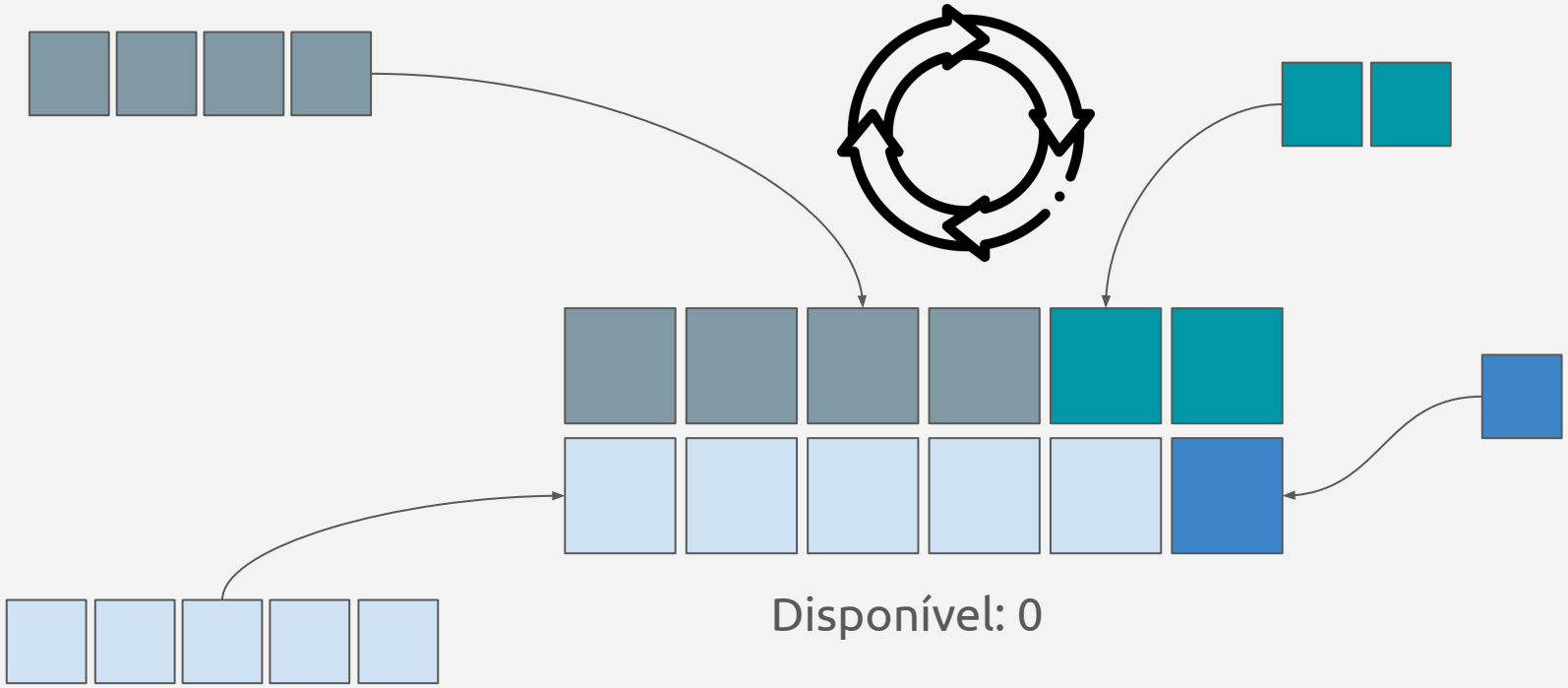


Disponível: 0

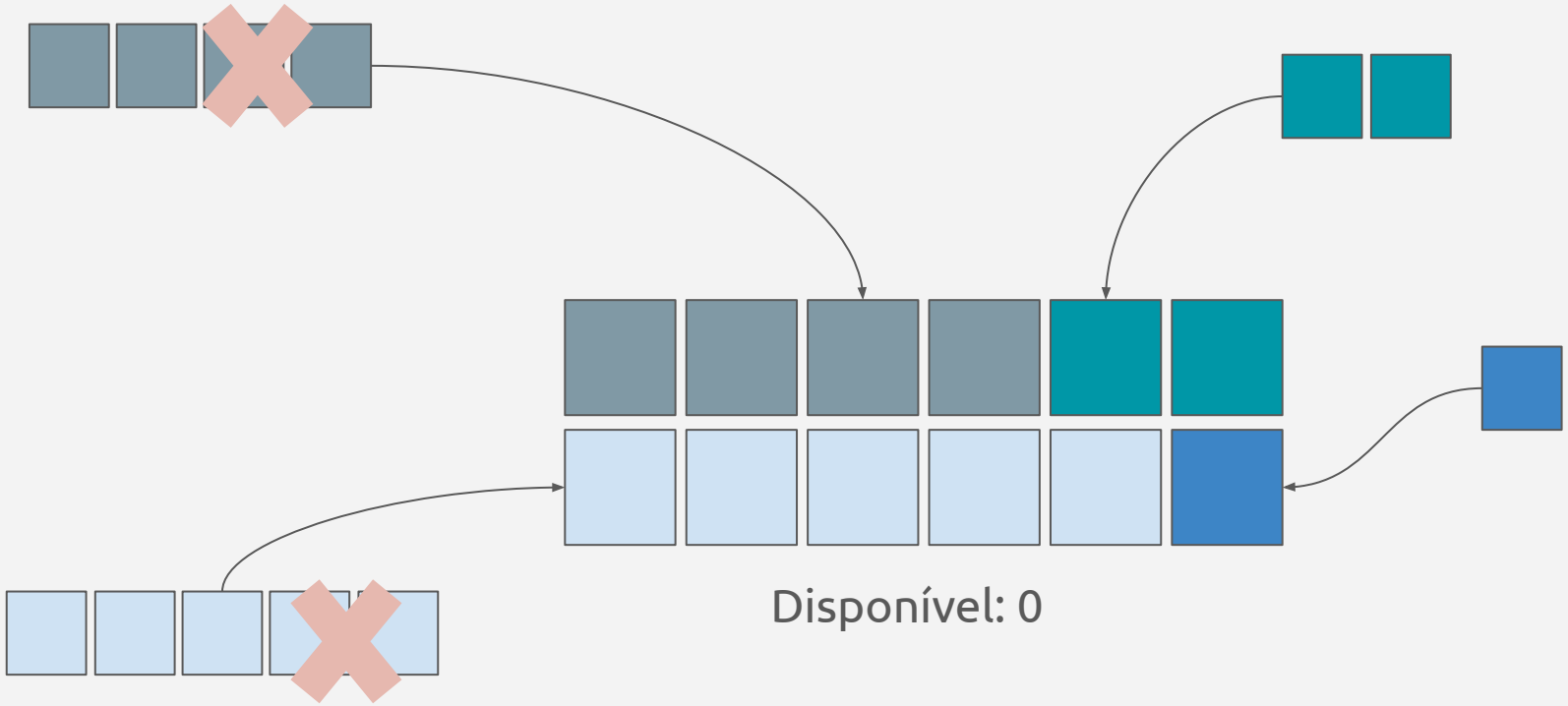
Alocando memória



Coletando a memória

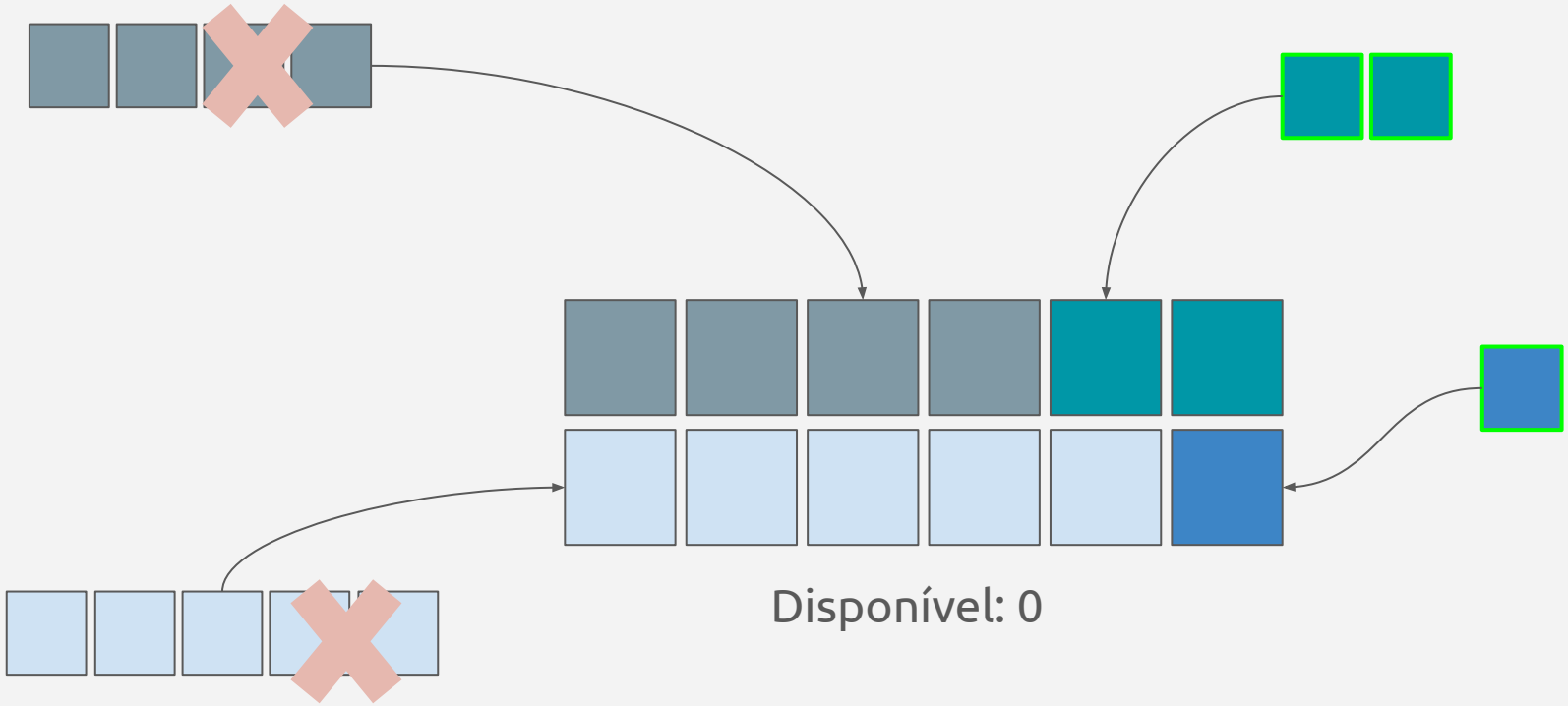


Coletando a memória

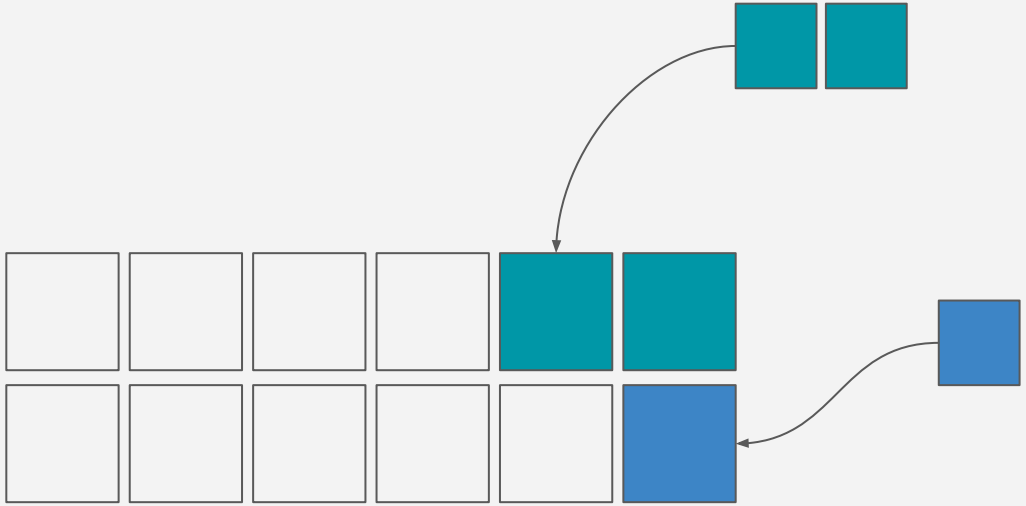


Disponível: 0

Mark

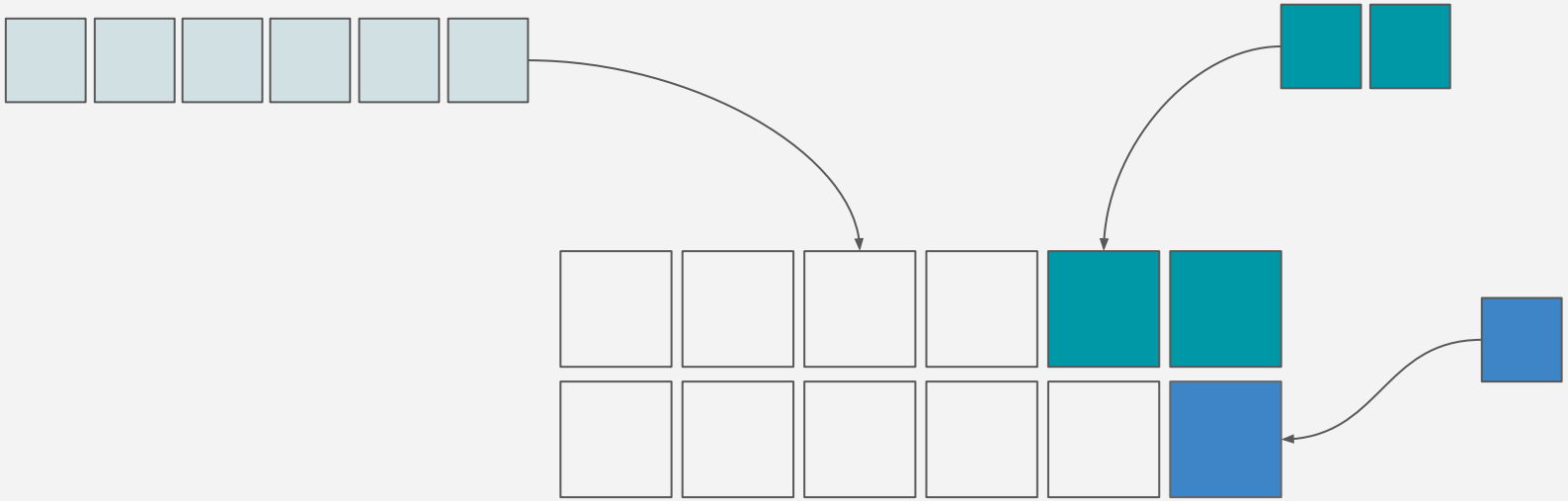


Sweep



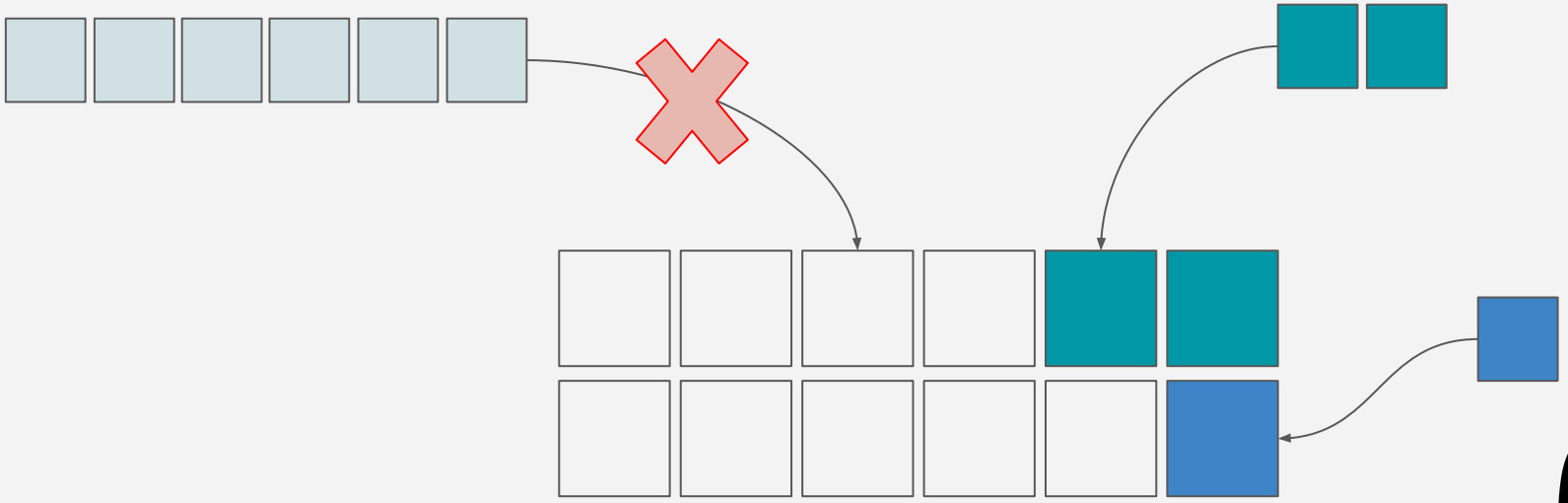
Disponível: 9

Alocando um novo valor

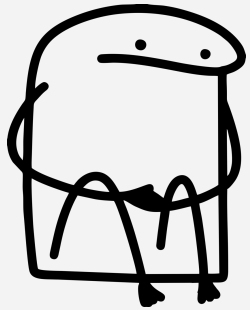


Disponível: 9

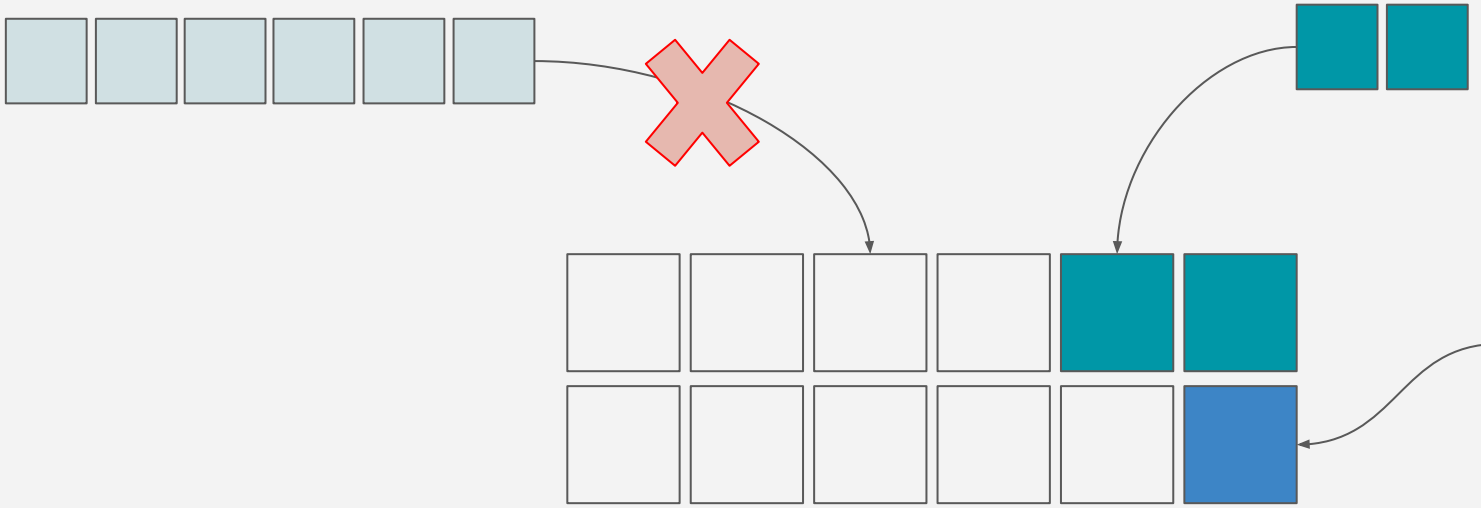
Alocando um novo valor



Disponível: 9

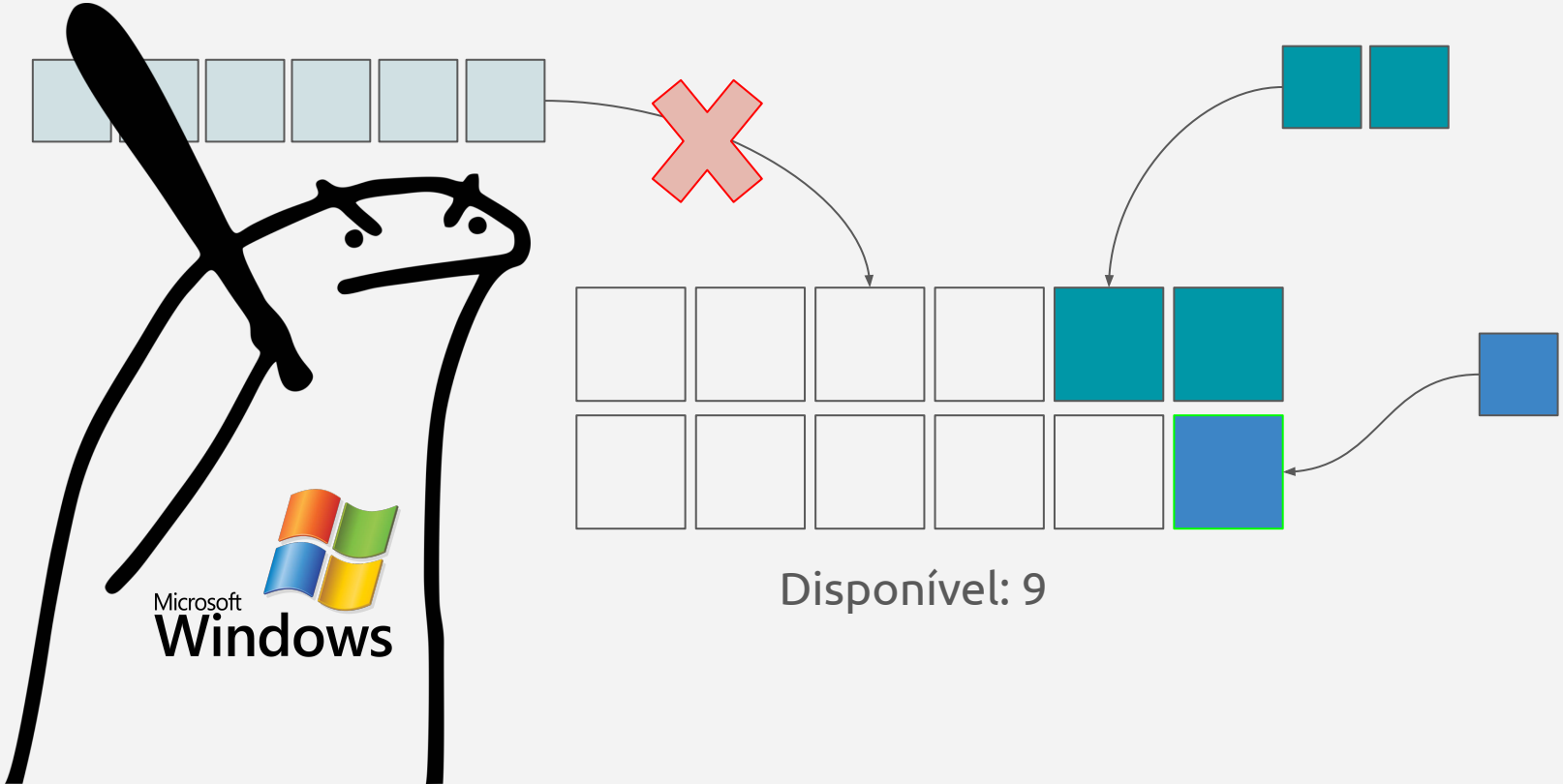


Fragmentação



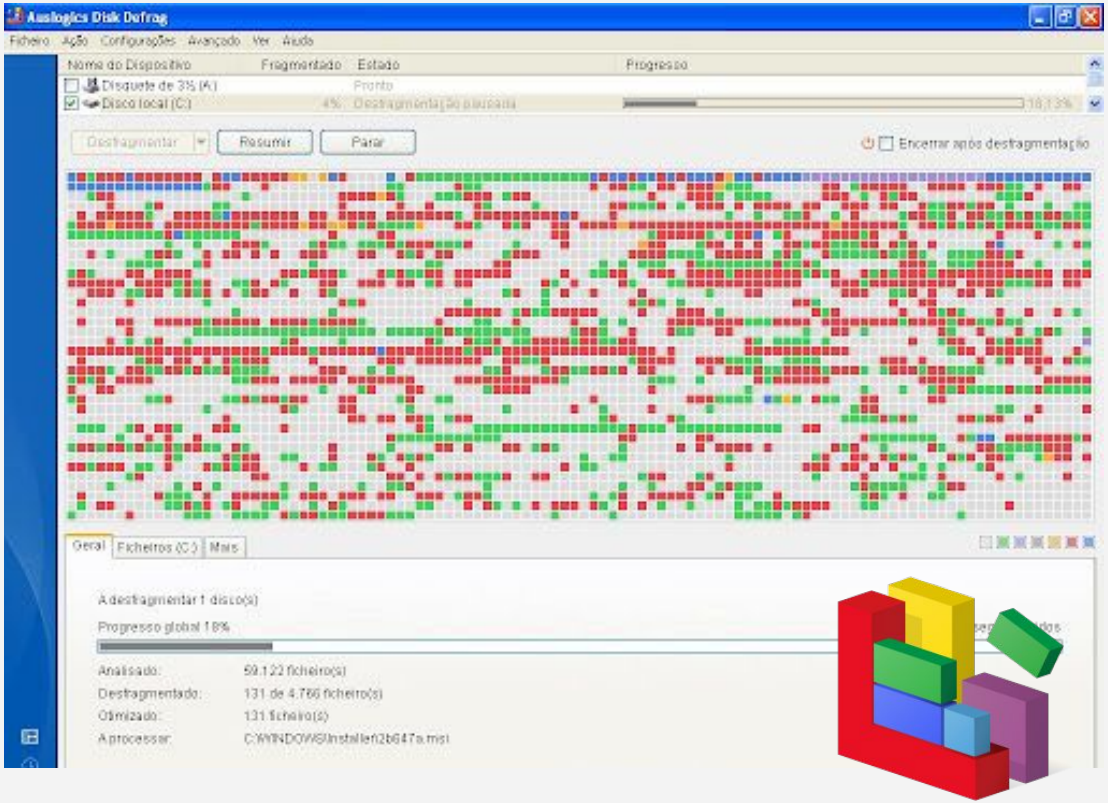
Disponível: 9

Fragmentação

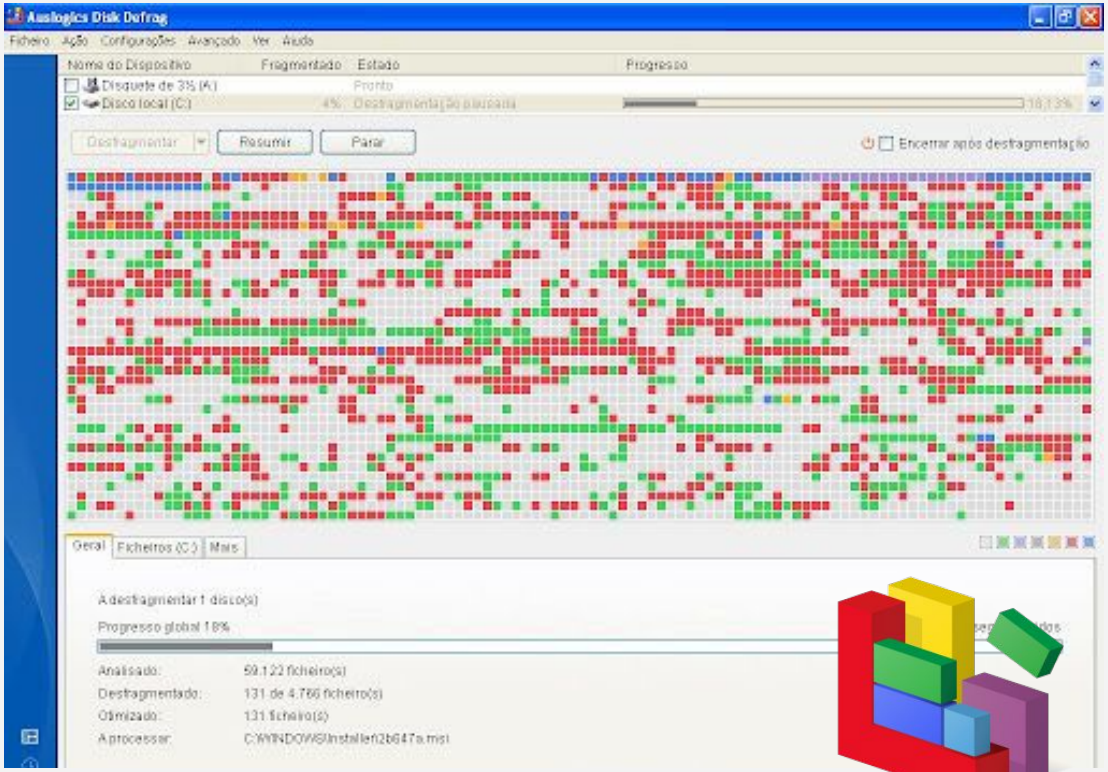


Disponível: 9

(des)Fragmentação



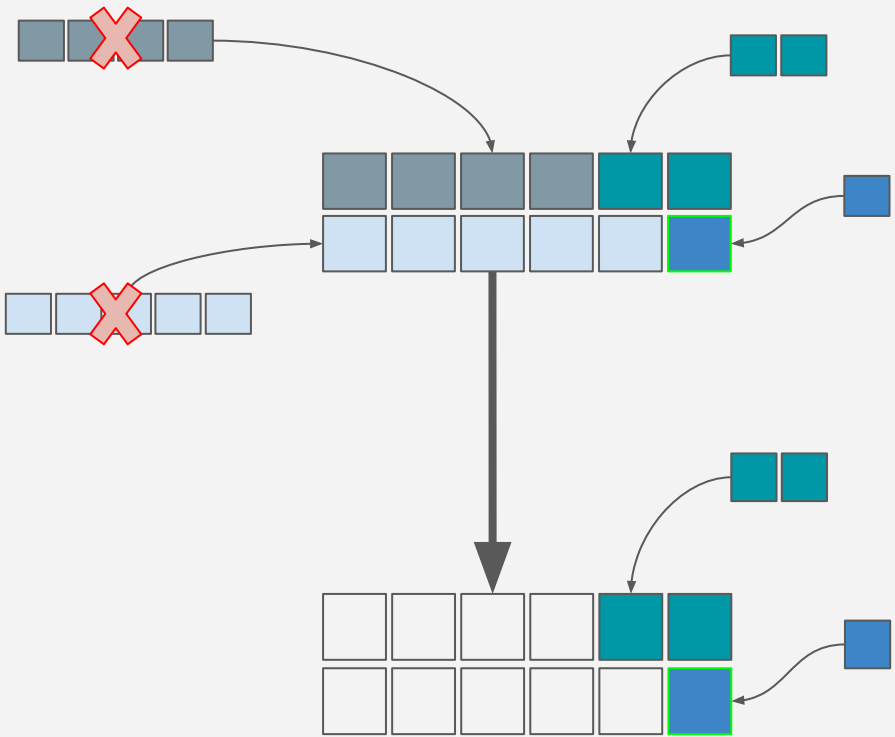
(des)Fragmentação



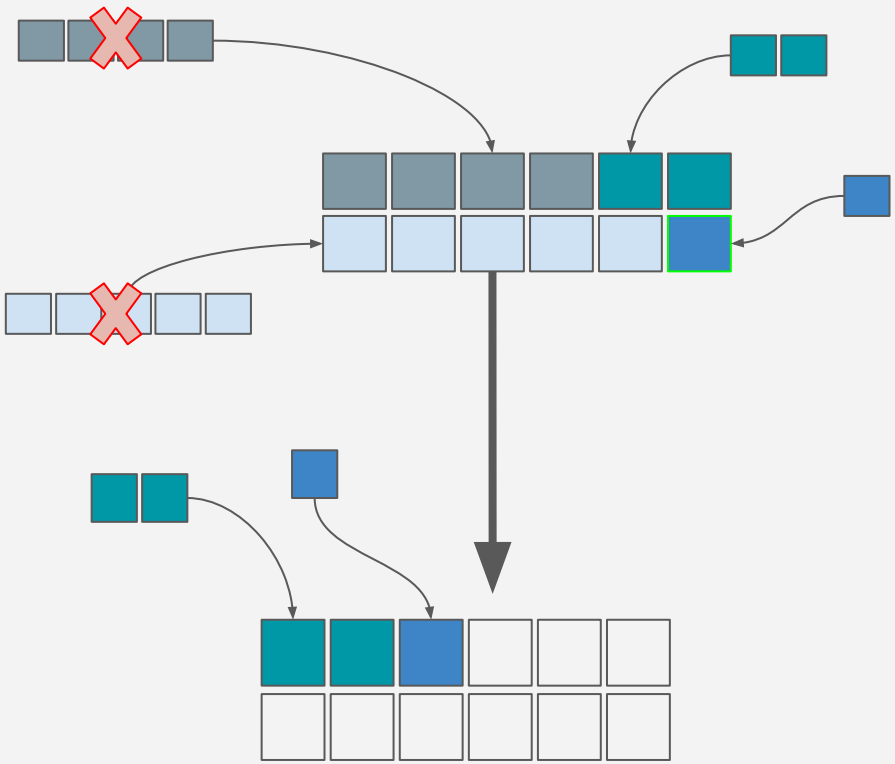
Move os dados para regiões contíguas de memória e elimina espaços vazios.

Compactação

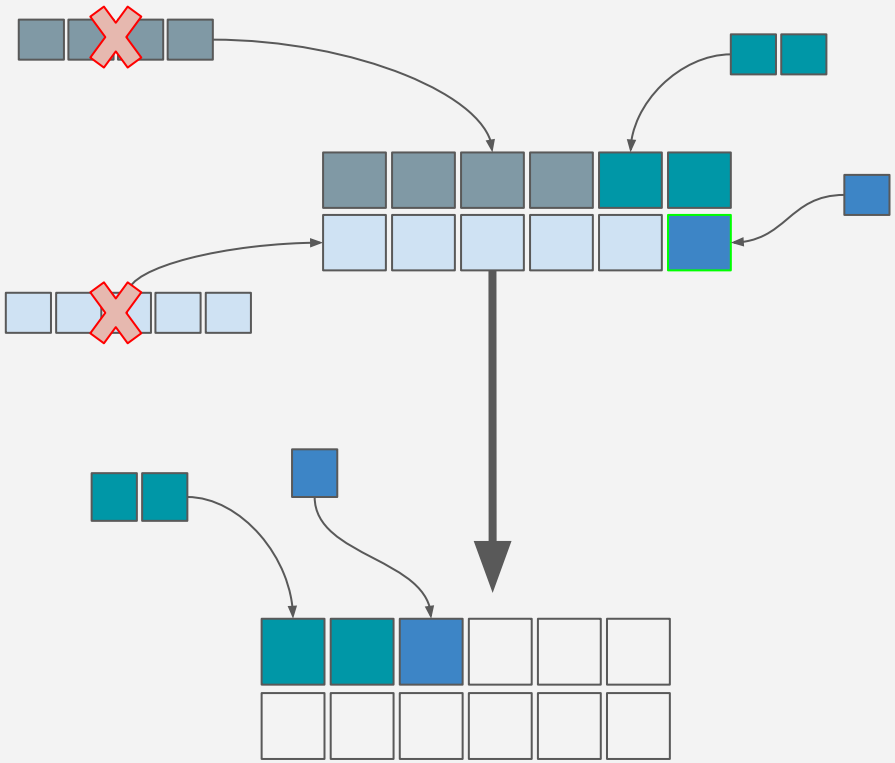
Compactação

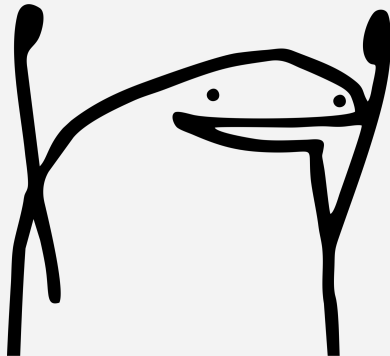


Compactação

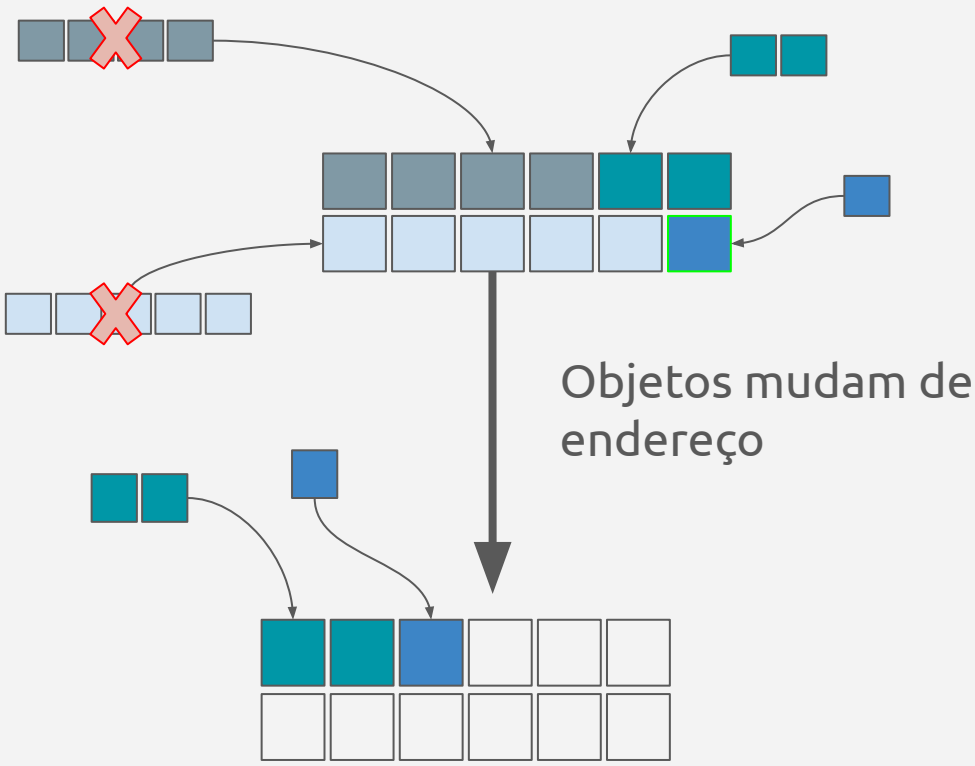


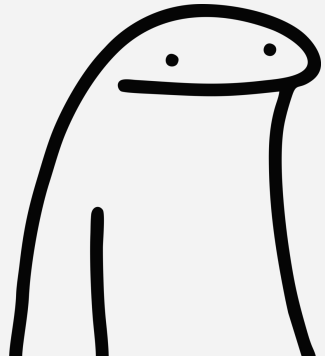
Compactação



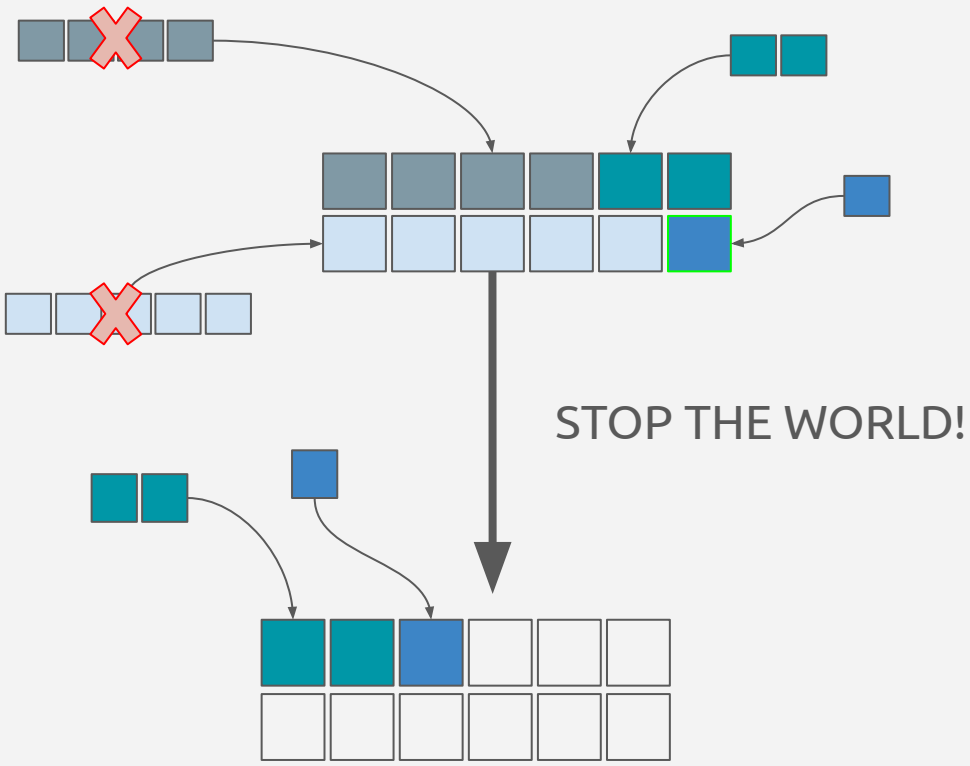


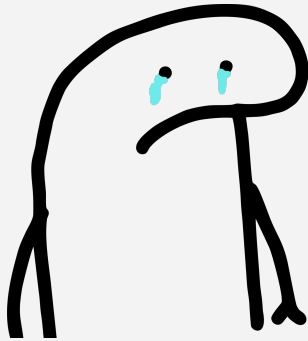
Compactação





Compactação





Como tornar a compactação mais eficiente

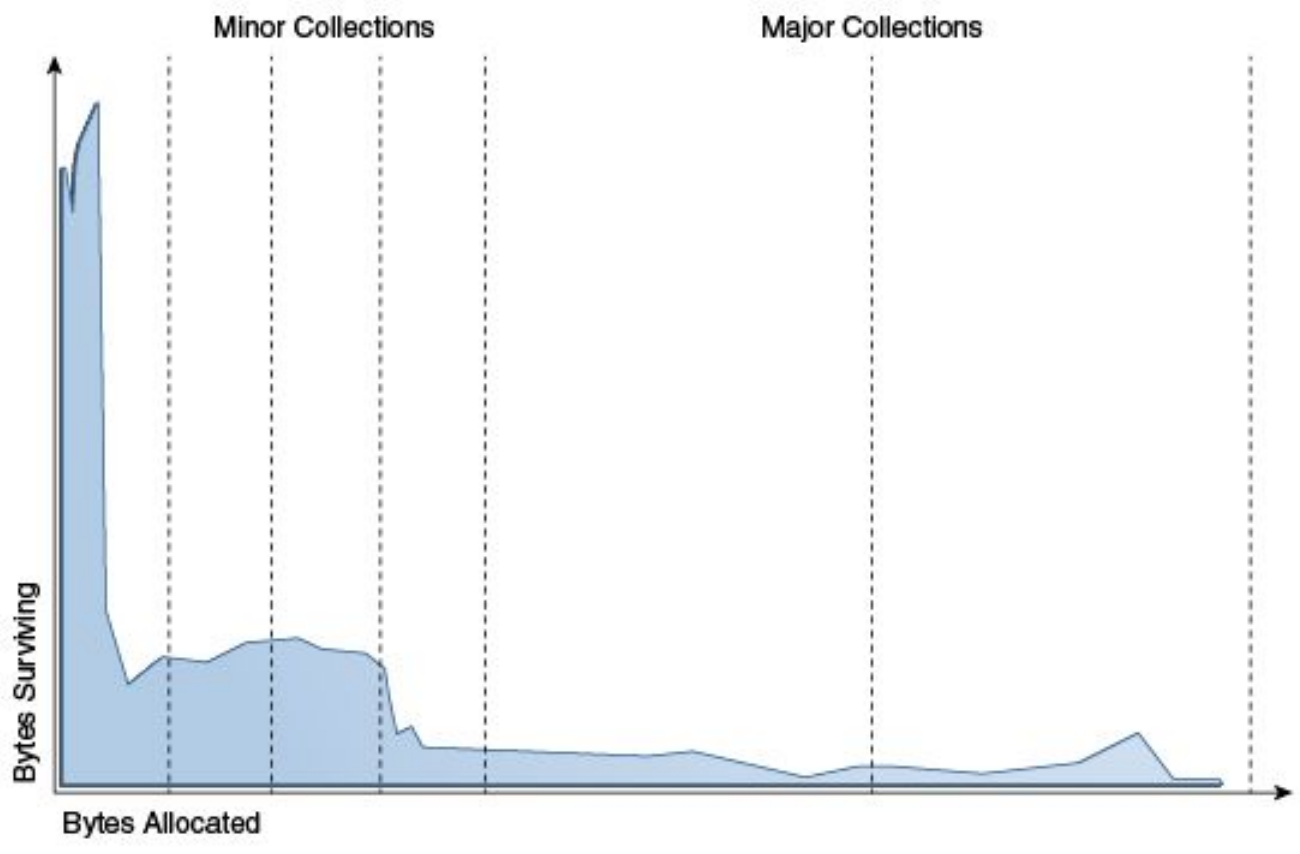
Weak Generational Hypothesis

Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm

David Ungar

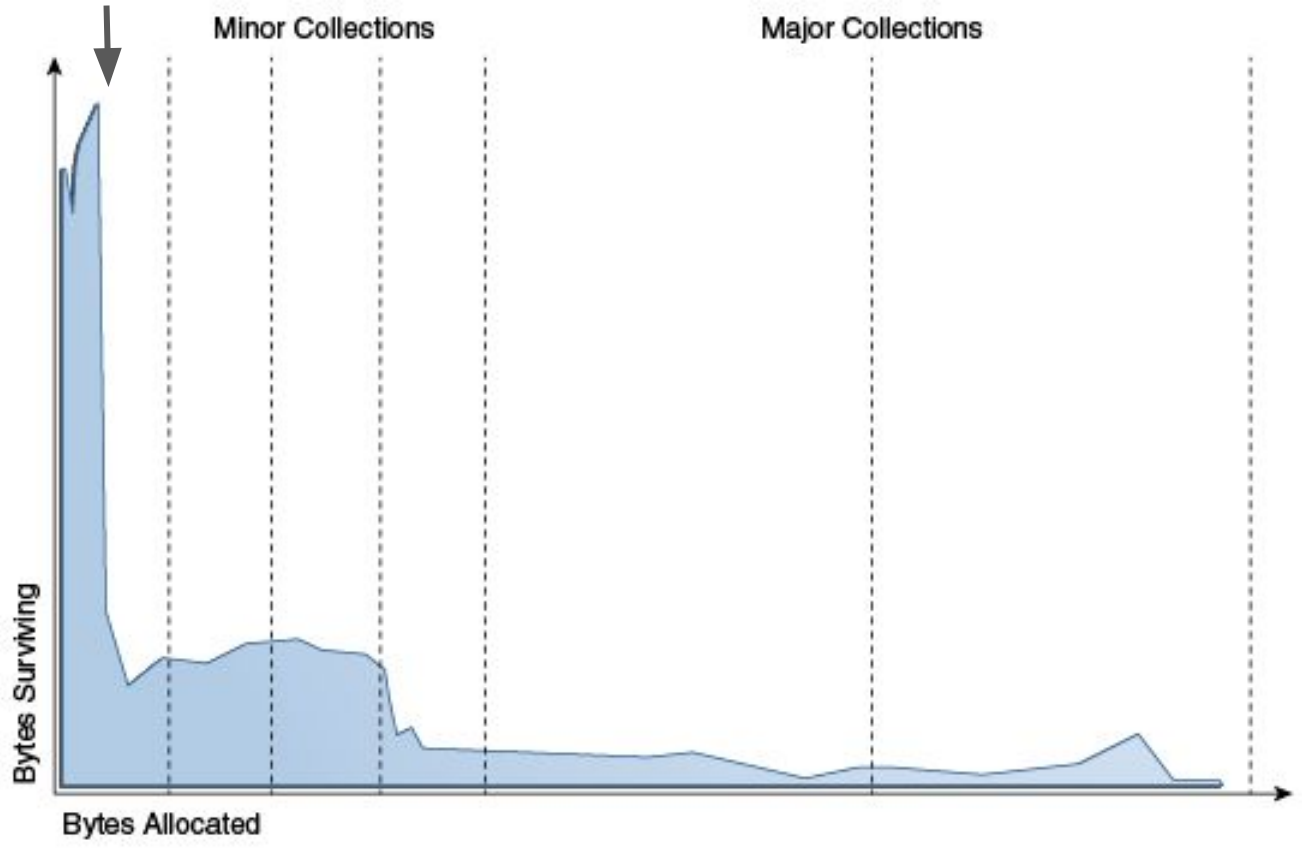
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

Weak Generational Hypothesis



docs.oracle.com

Weak Generational Hypothesis

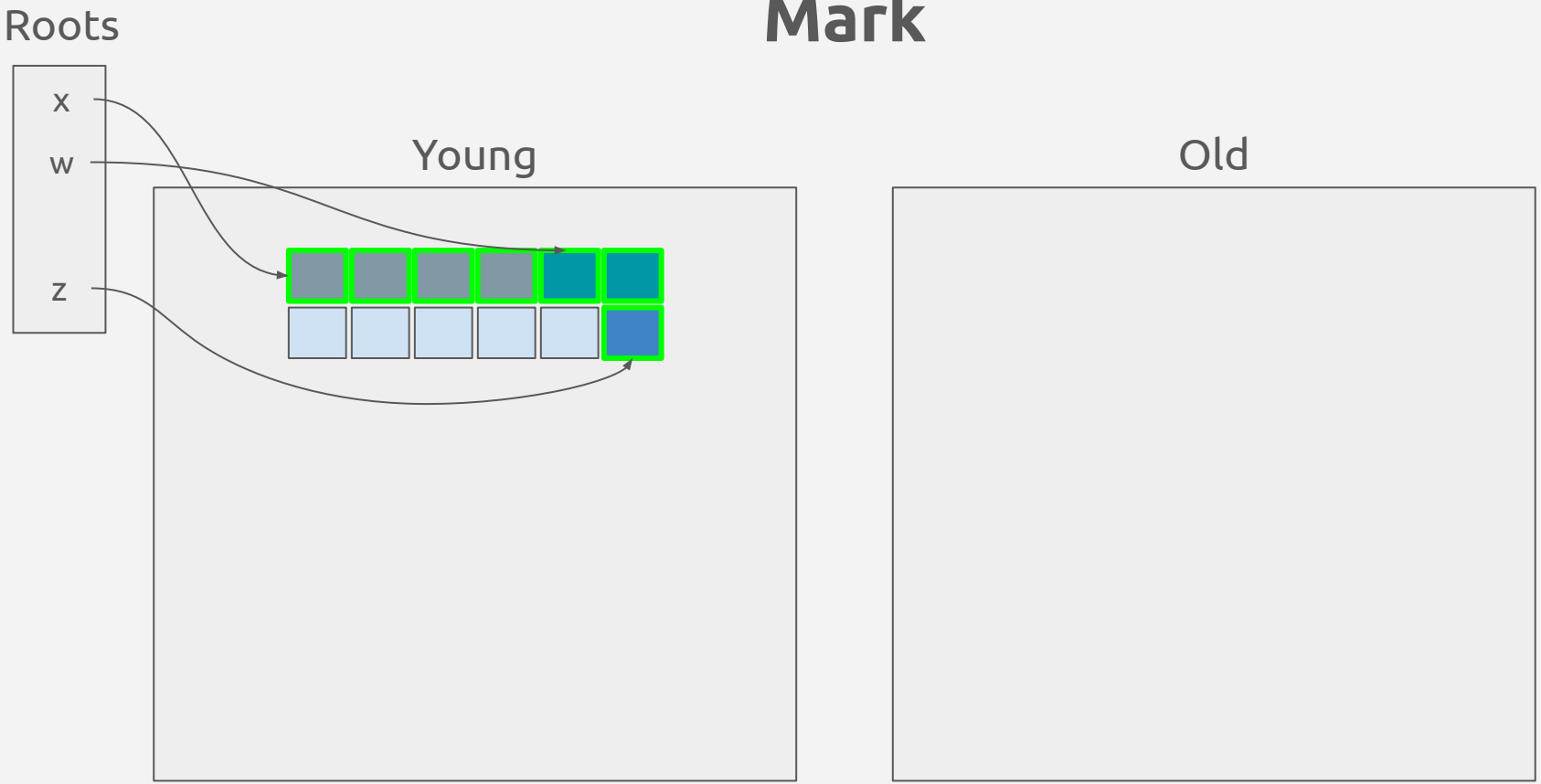


docs.oracle.com

GC geracional

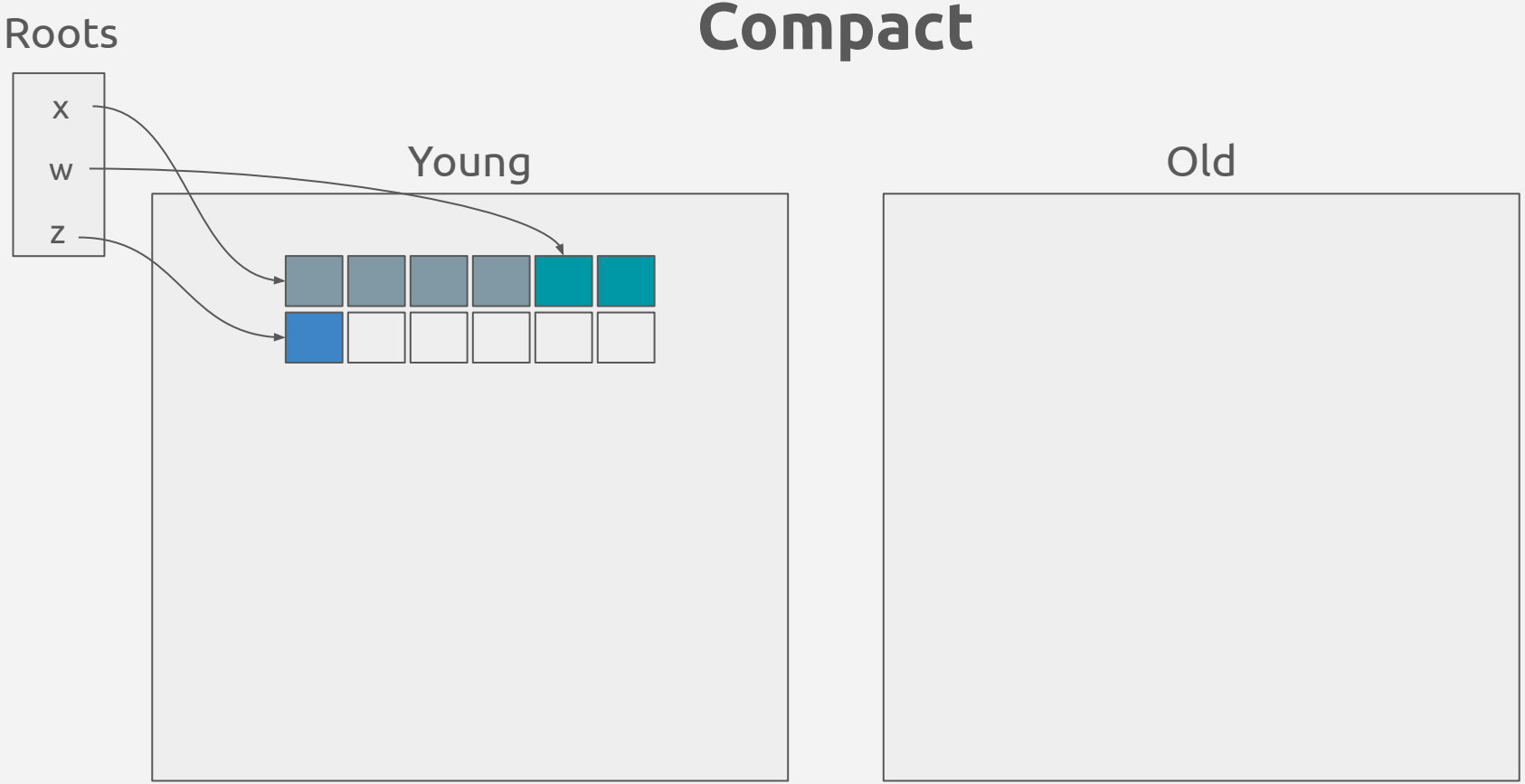
Variável y sai de escopo

Mark



Sweep

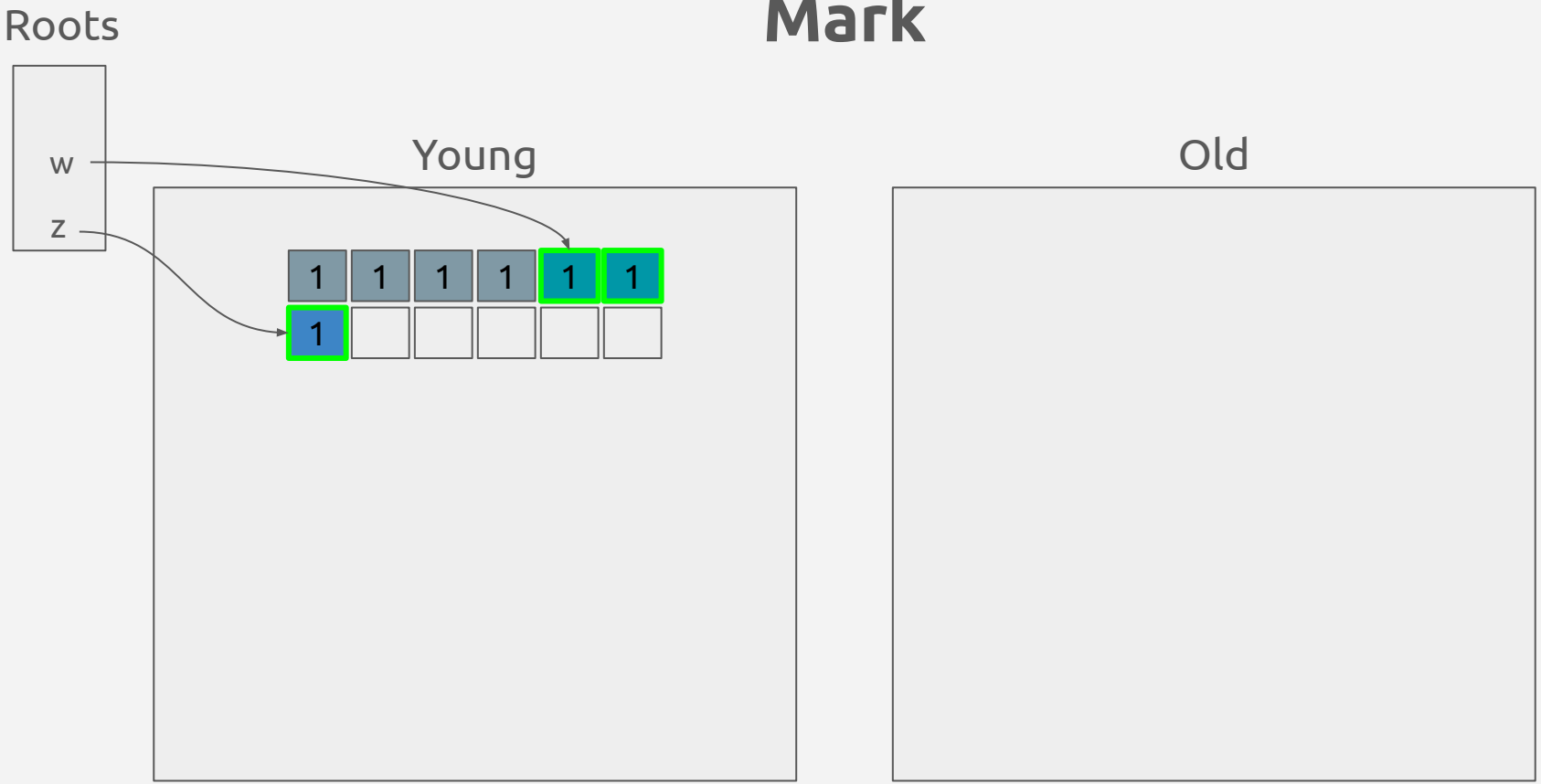
Compact



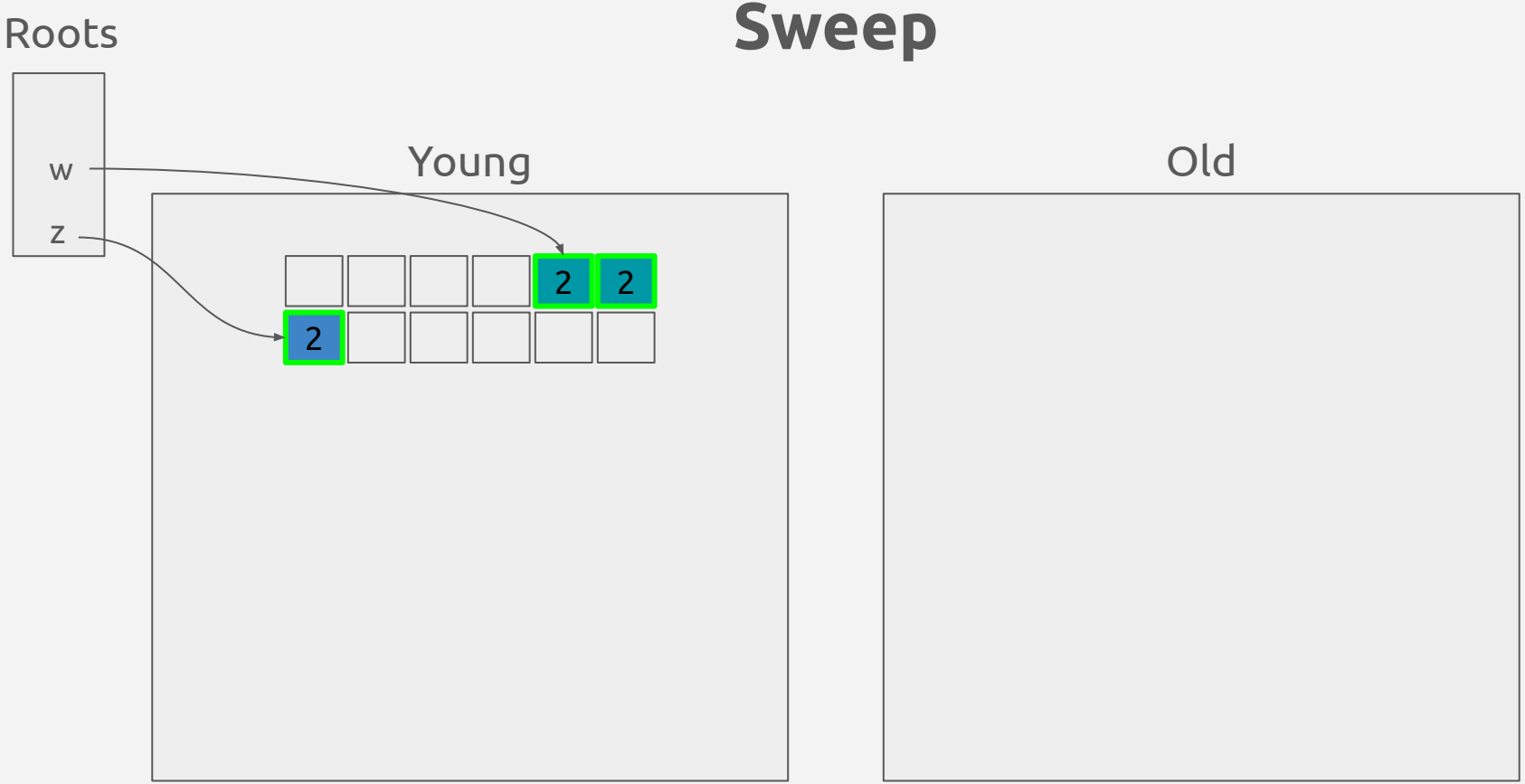
Marca geração

Variável x sai do escopo

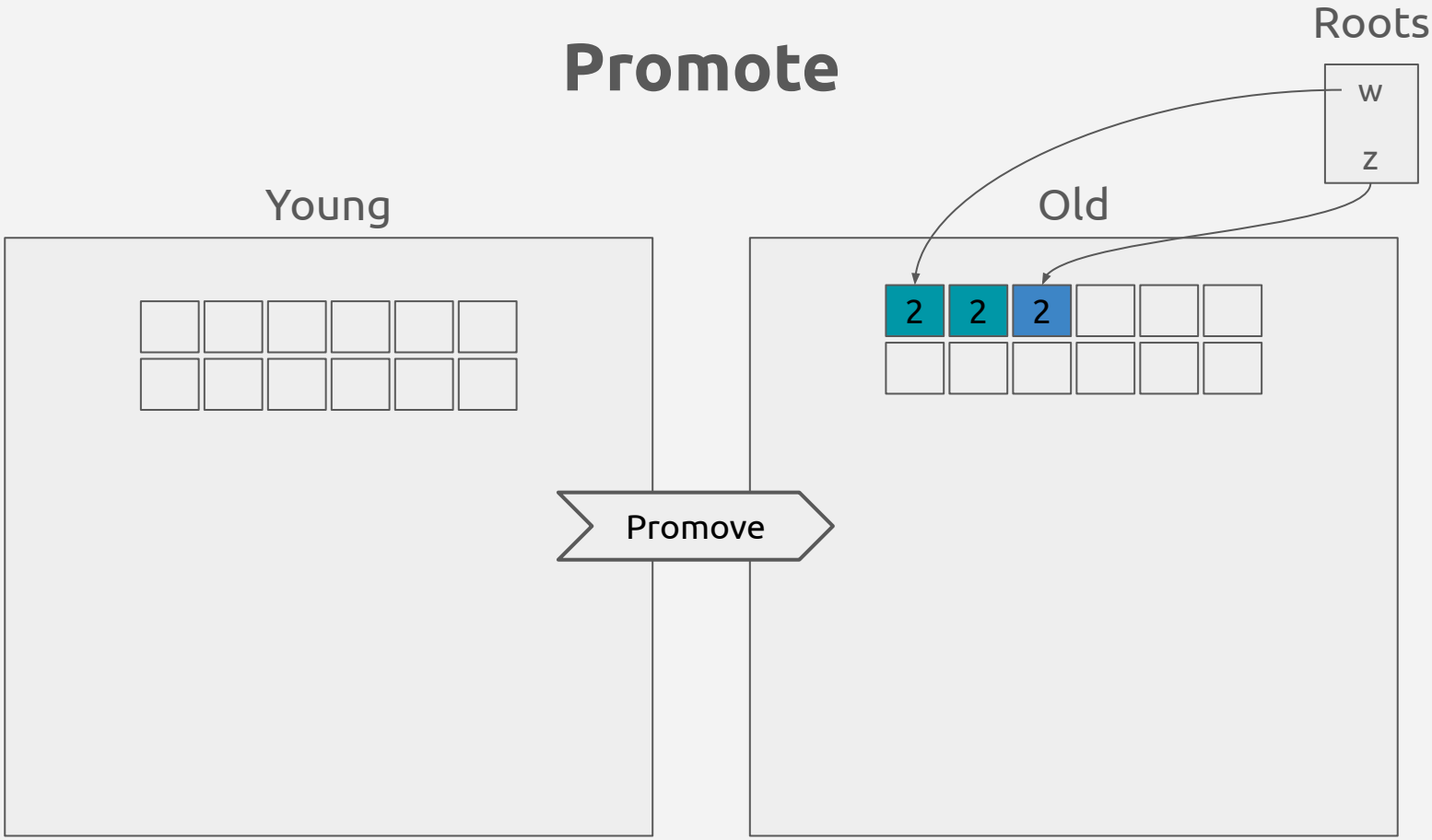
Mark



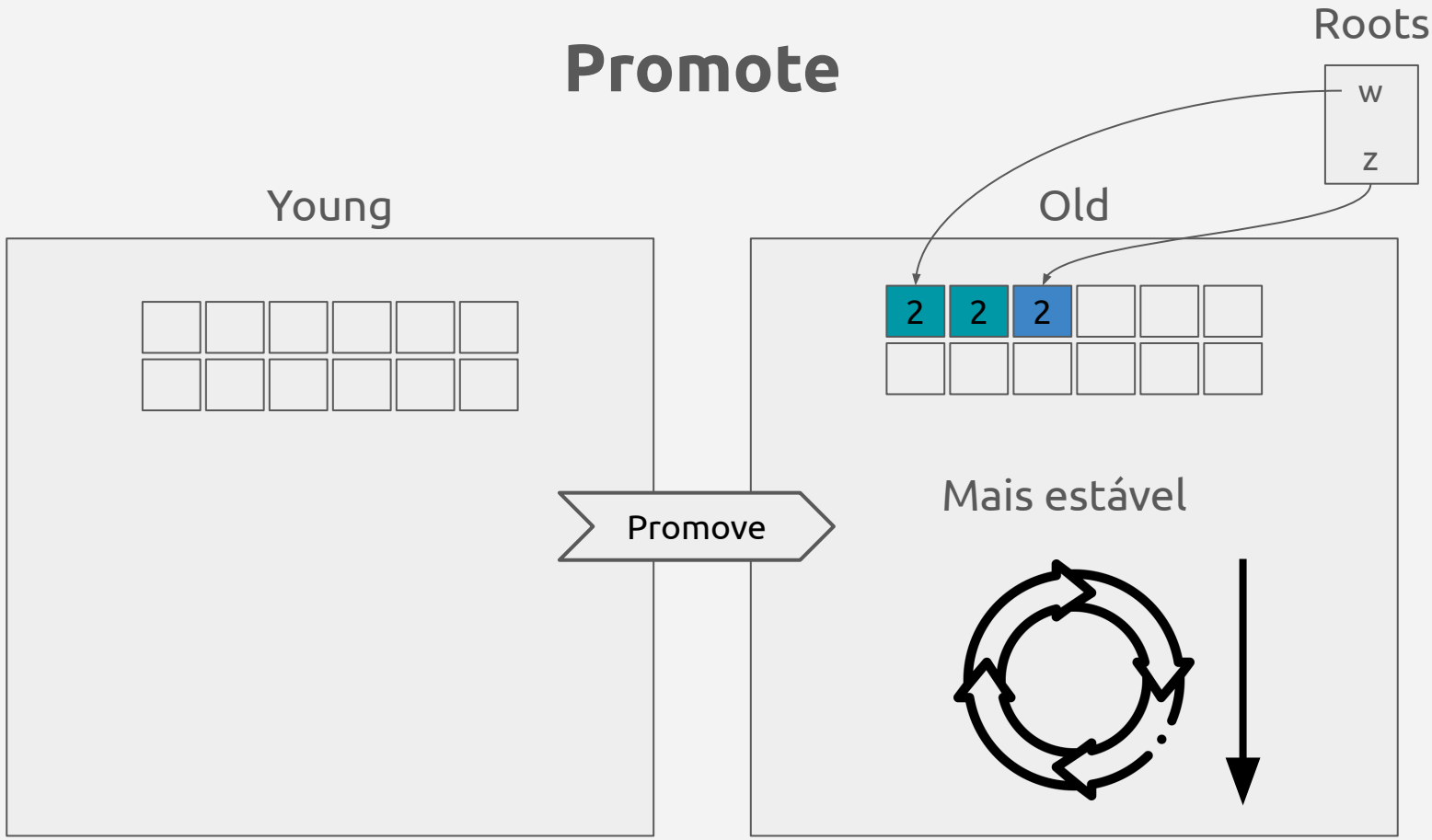
Sweep



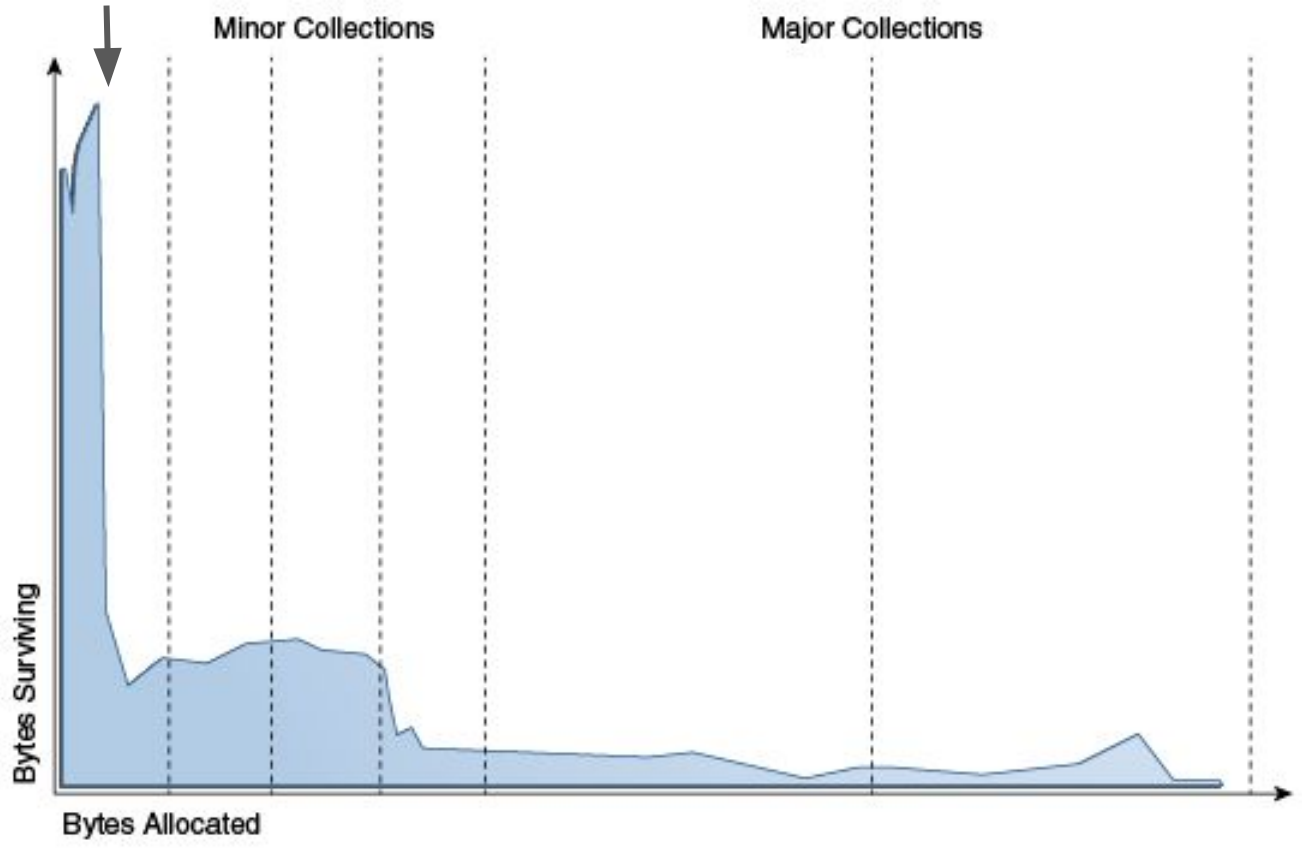
Promote



Promote

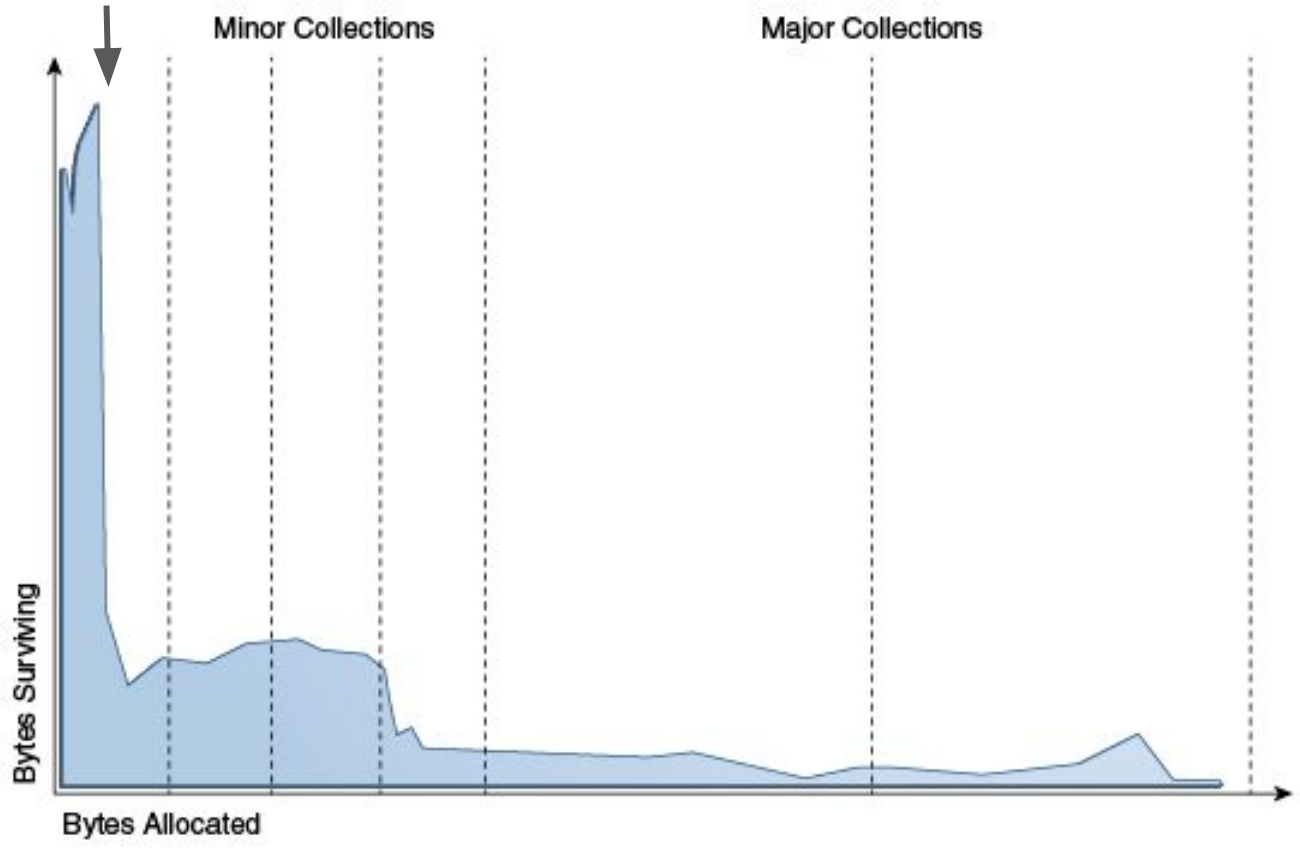


Objetos novos vivem pouco



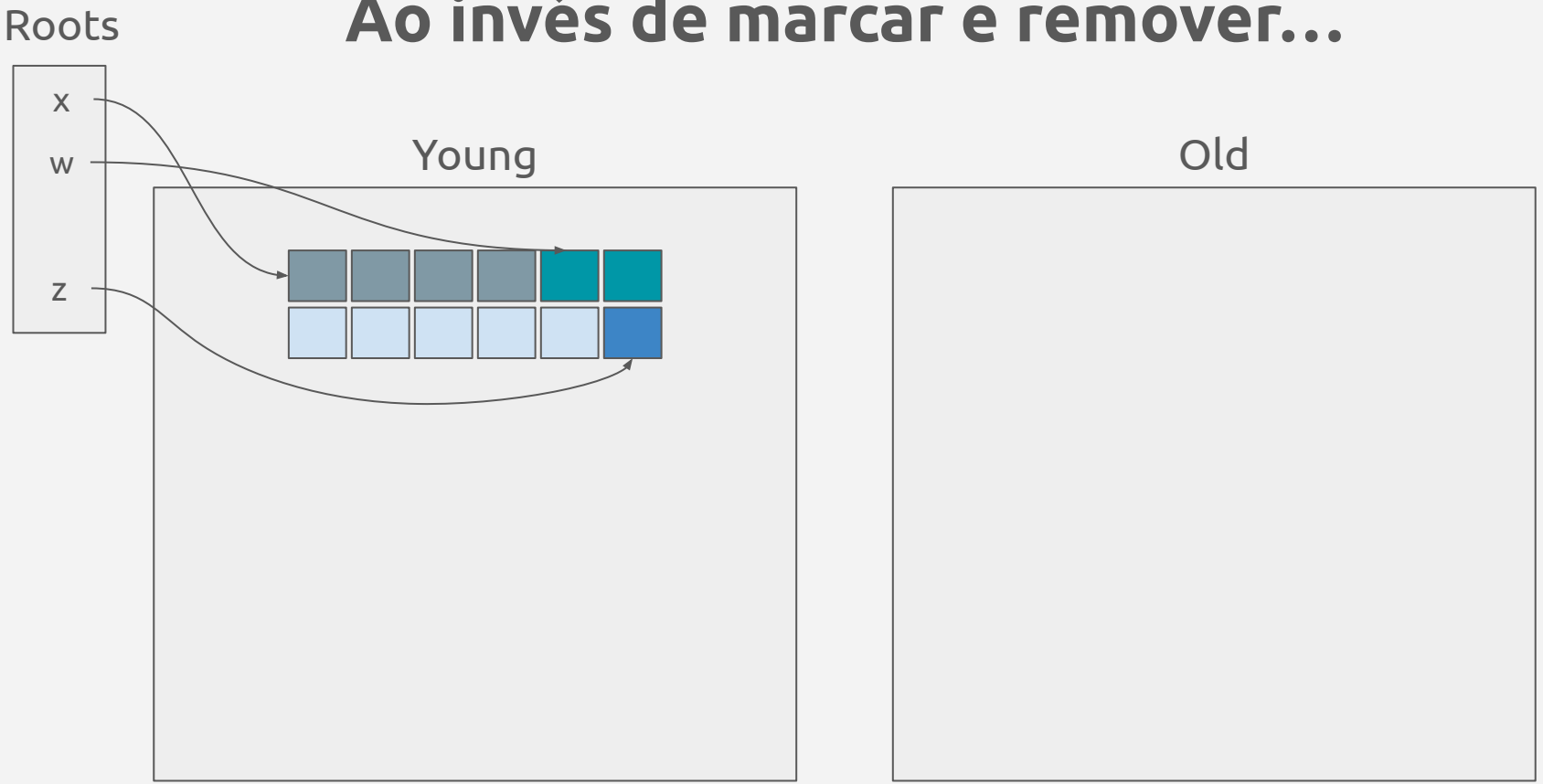
docs.oracle.com

Mais removidos do que mantidos

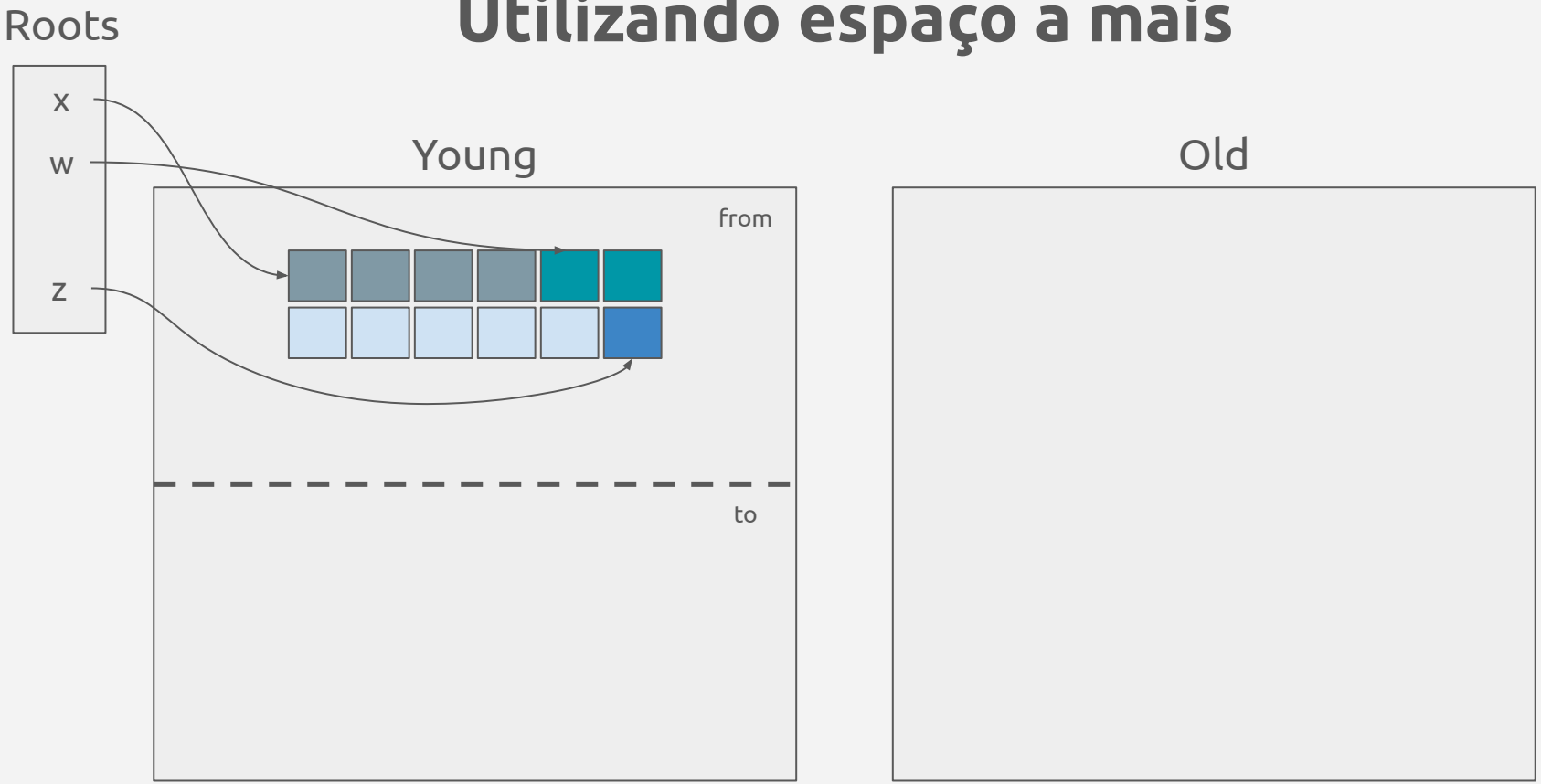


docs.oracle.com

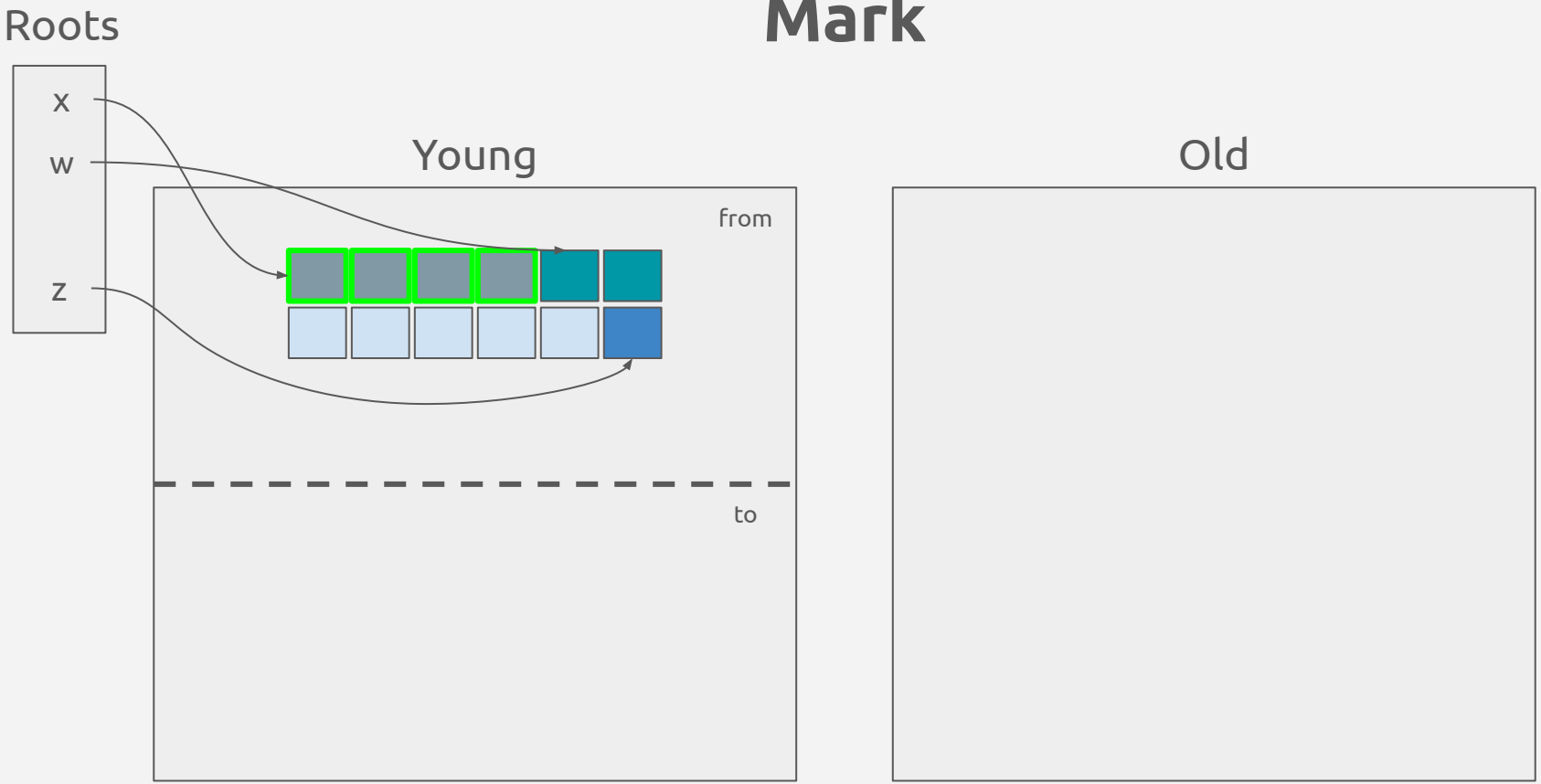
Ao invés de marcar e remover...



Utilizando espaço a mais

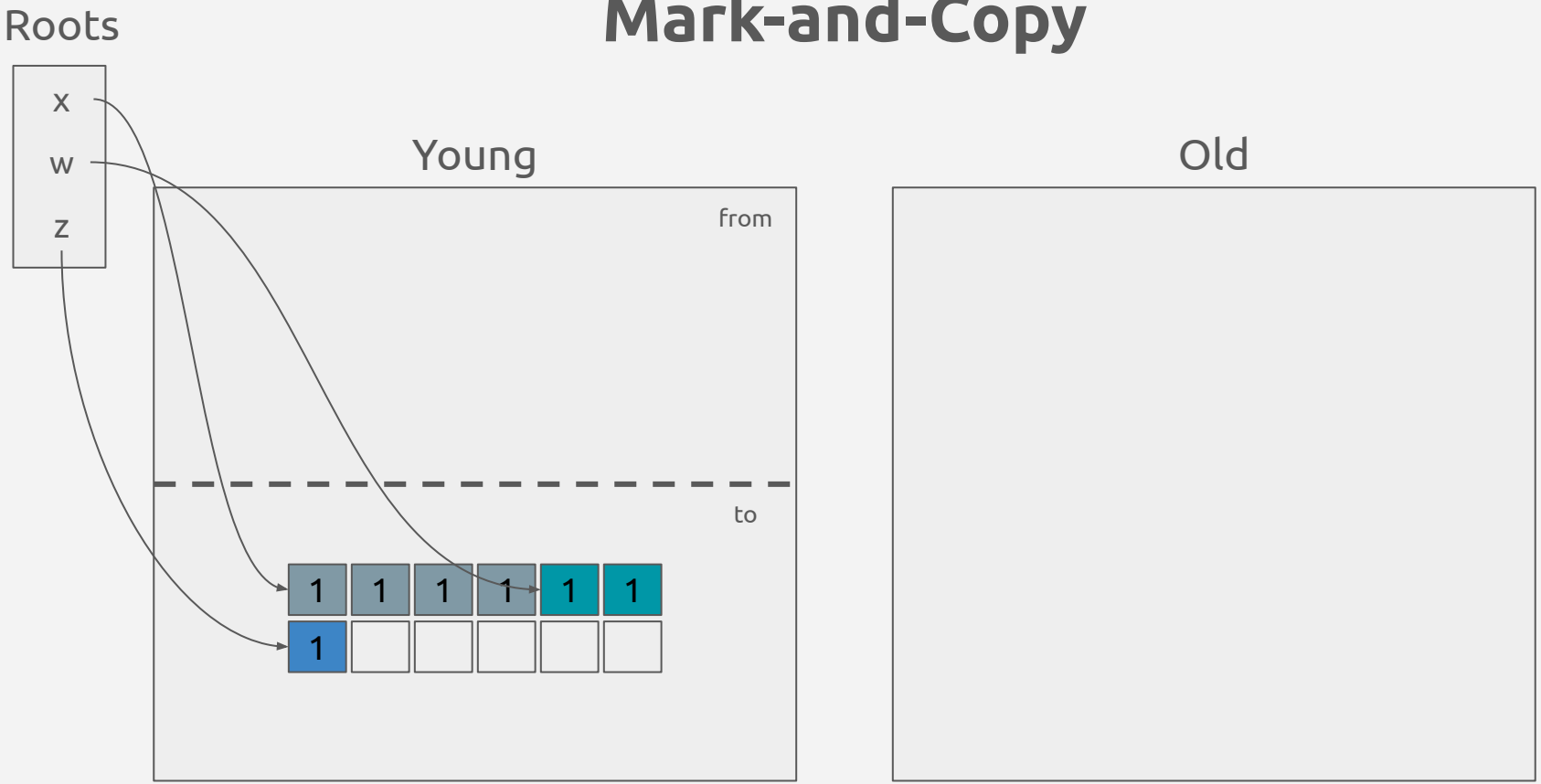


Mark

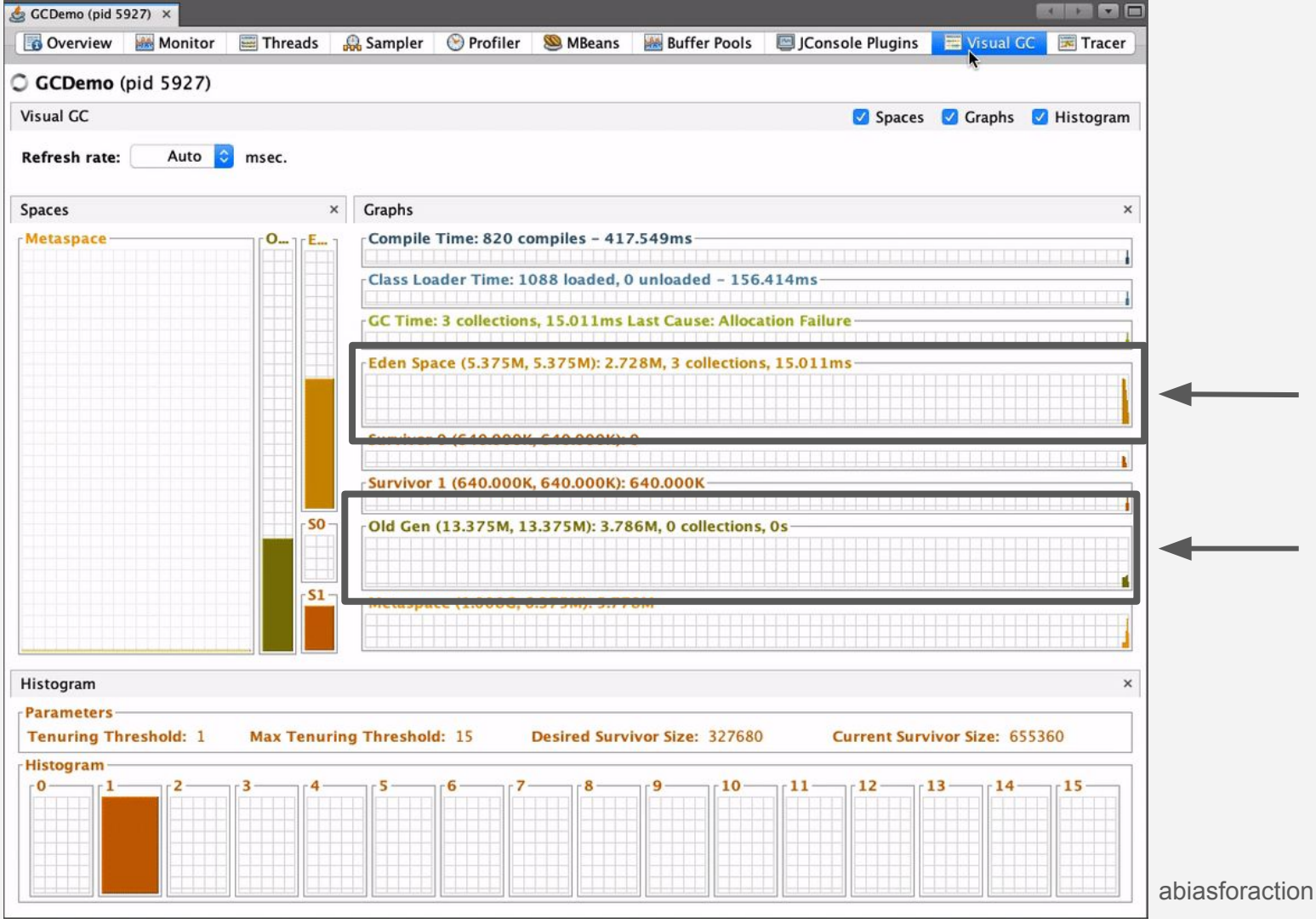


and… Copy

Mark-and-Copy



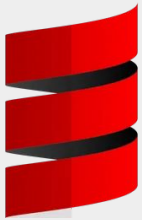

Visual GC
Spaces
Graphs
Histogram
Refresh rate:
Auto
msec.
Spaces
Graphs
Perm
Old
Eden
S0
S1
Compile Time: 668 compiles - 9.071s
Class Loader Time: 1775 loaded, 0 unloaded - 1.009s
GC Time: 202 collections, 333.156ms Last Cause: unknown GCCause
Eden Space (25.625M, 16.625M): 6.180M, 202 collections, 333.156ms
Survivor 1 (3.188M, 2.062M): 0
Old Gen (224.000M, 62.375M): 2.789M, 0 collections, 0s
202
0
Histogram
Parameters
Tenuring Threshold: 4
Max Tenuring Threshold: 4
Desired Survivor S... 1081344
Current Survivor S... 2162688
Histogram
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



Generational GC

JVM






CLR





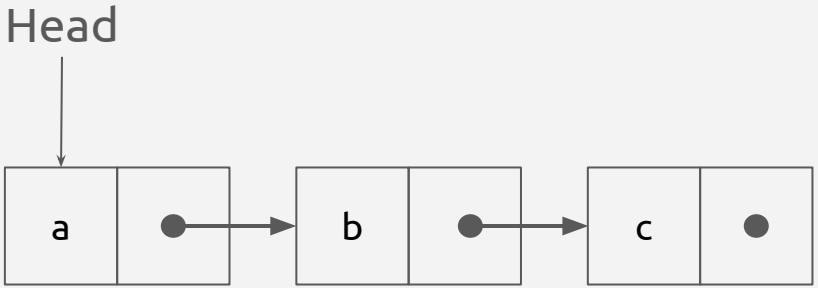




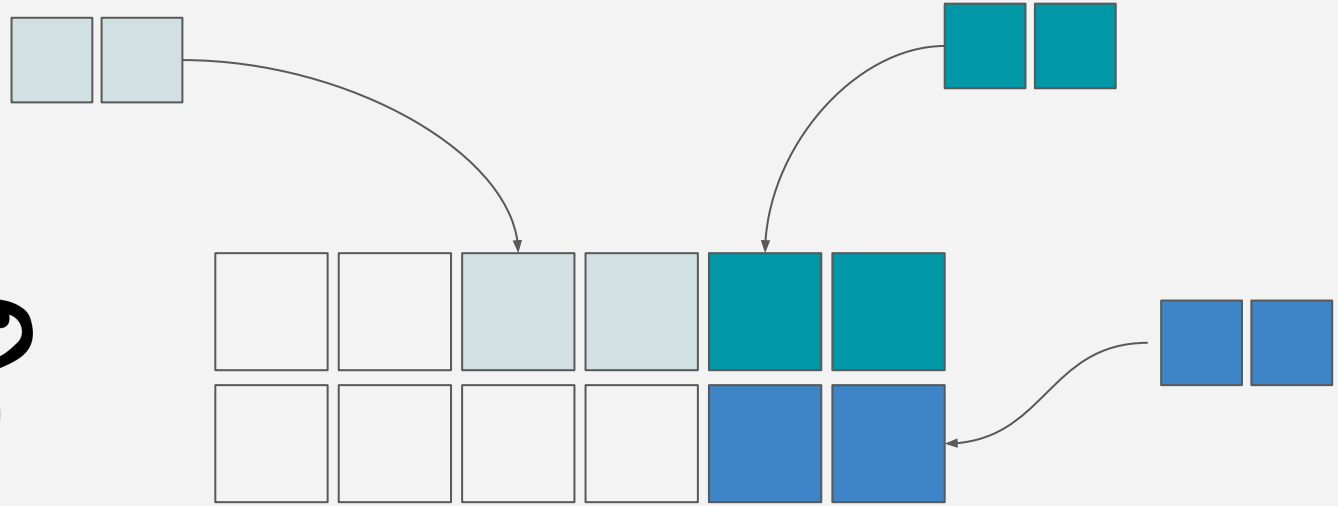
Mark-and-Copy (young)/Mark-and-compact (old) Generational GC

Cópias e compactação não são as únicas maneiras de lidar com fragmentação

Não havia fragmentação significativa em LISP

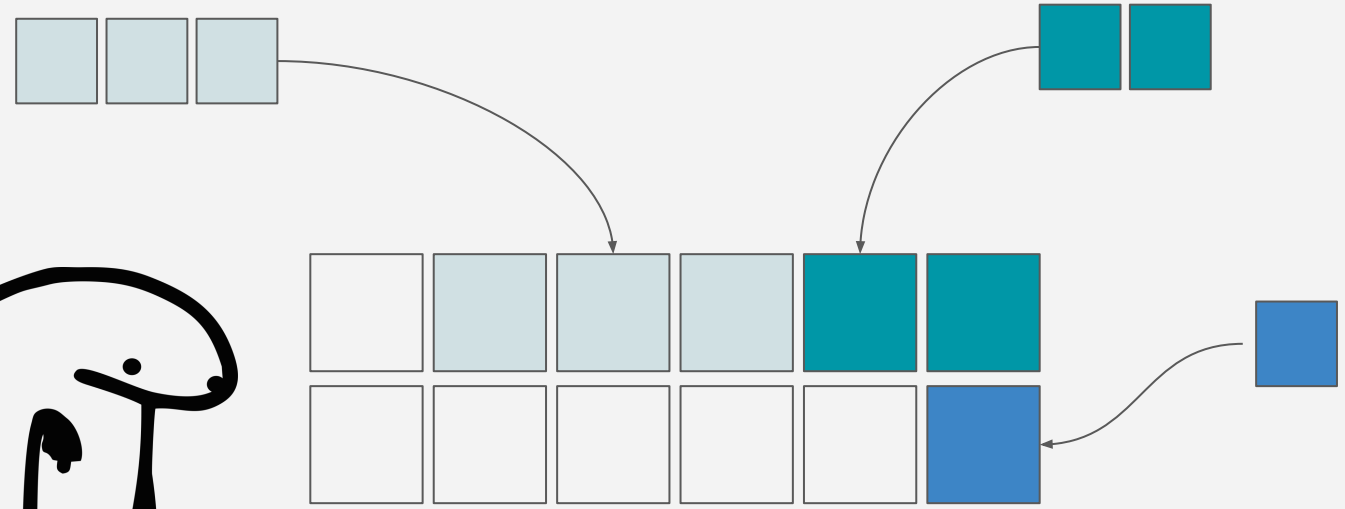


Alocações iguais == sem fragmentação



Disponível: 6

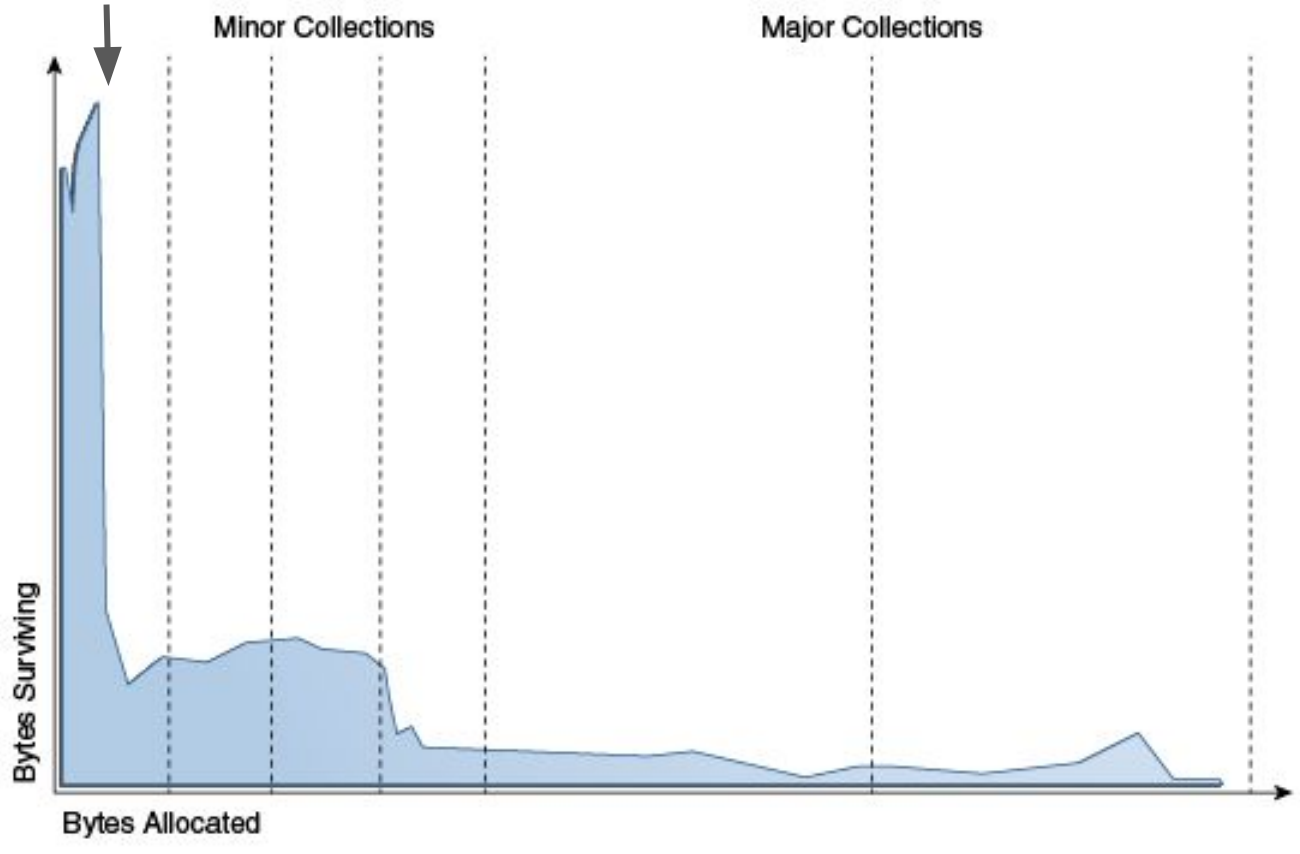
Mas o mundo real é diferente



Disponível: 6

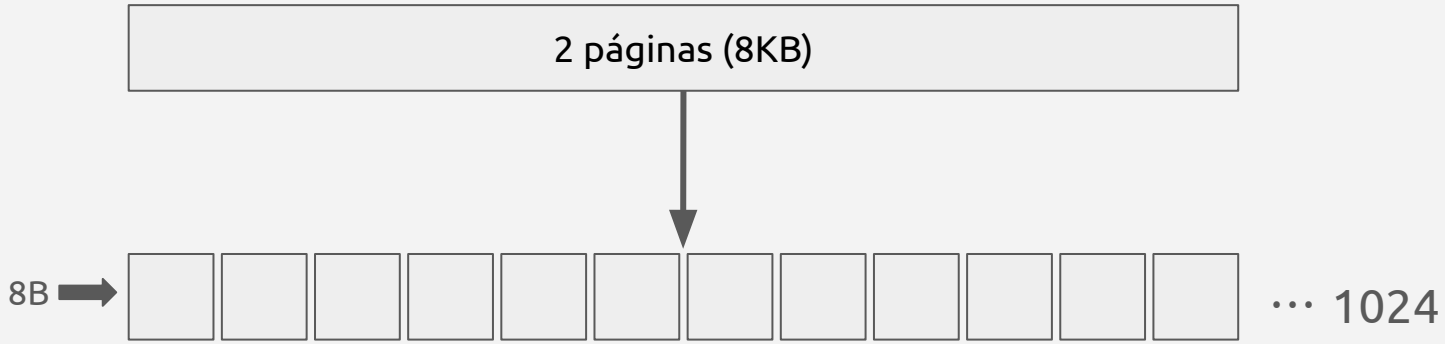
Modelo de memória do Go

Novamente...

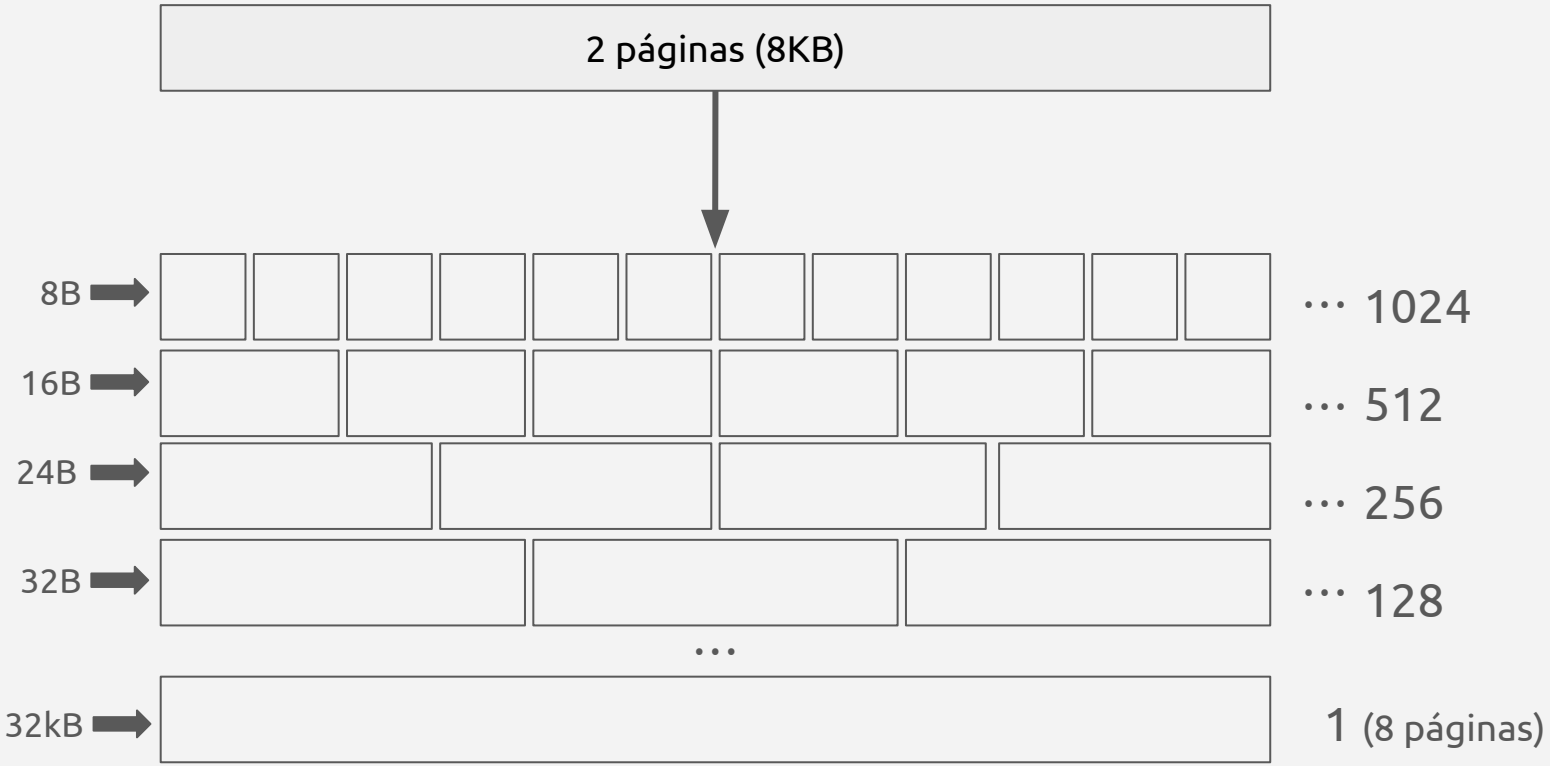


docs.oracle.com

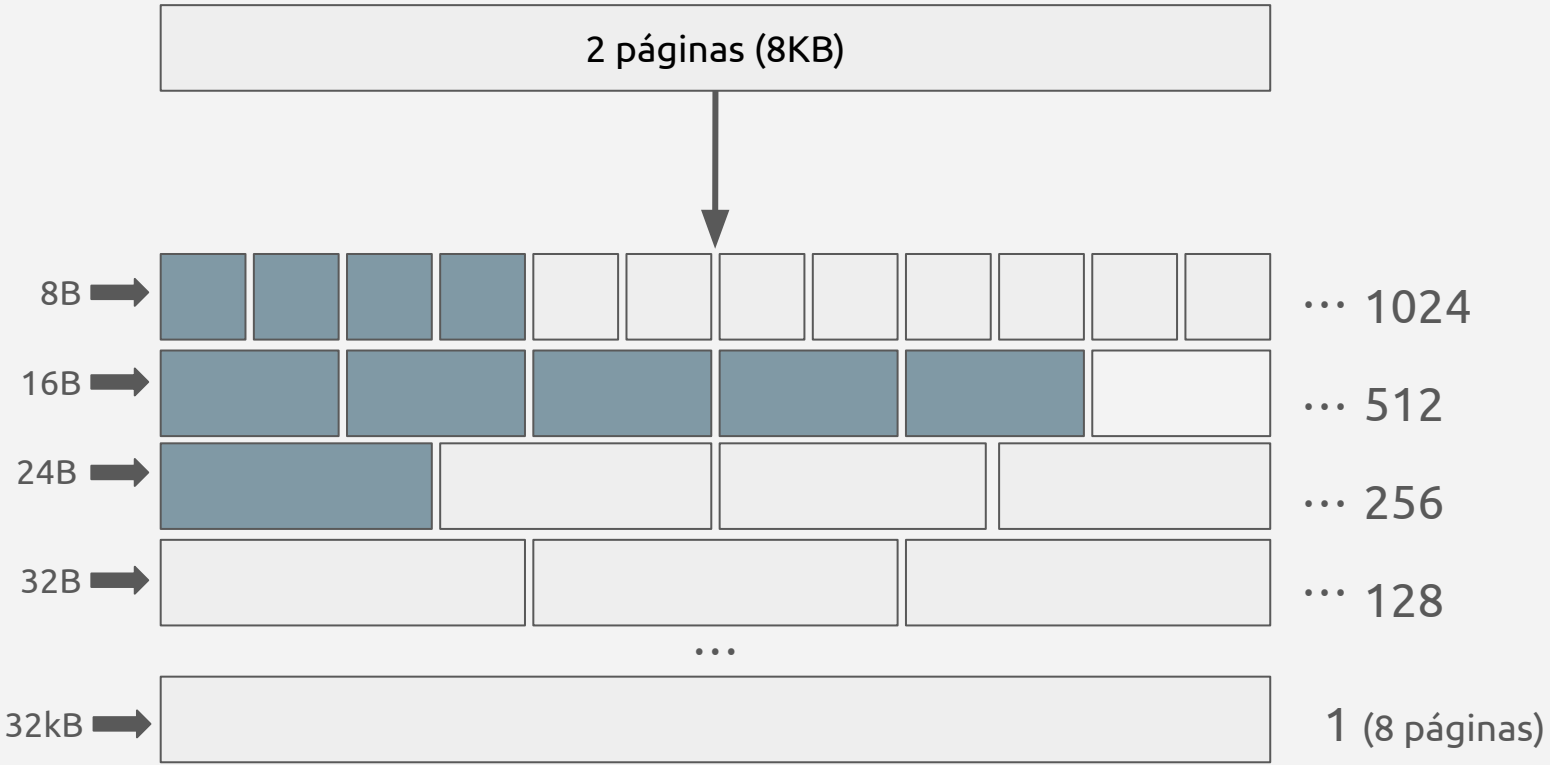
Quebra porções de memória em blocos iguais (classes)



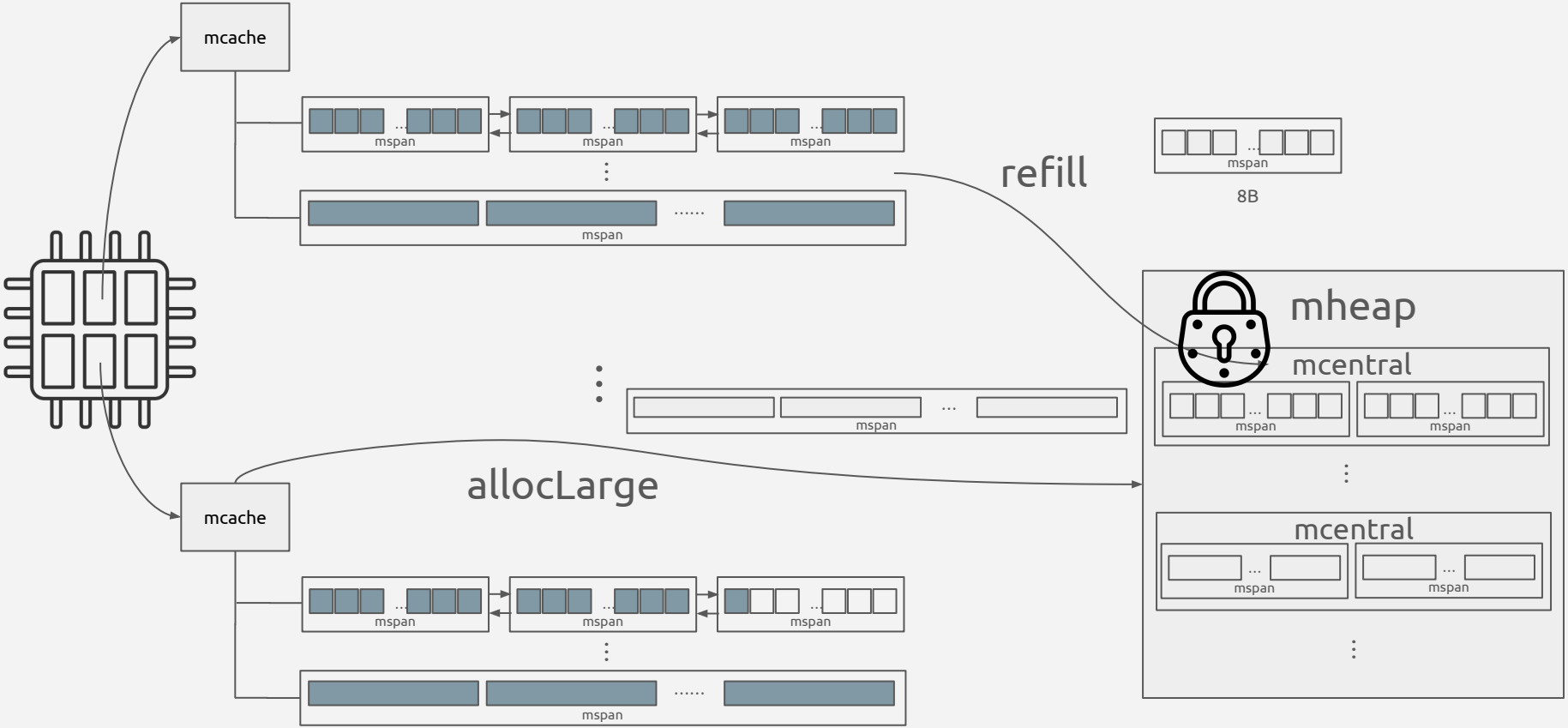
Cada classe de tamanho tem uma lista



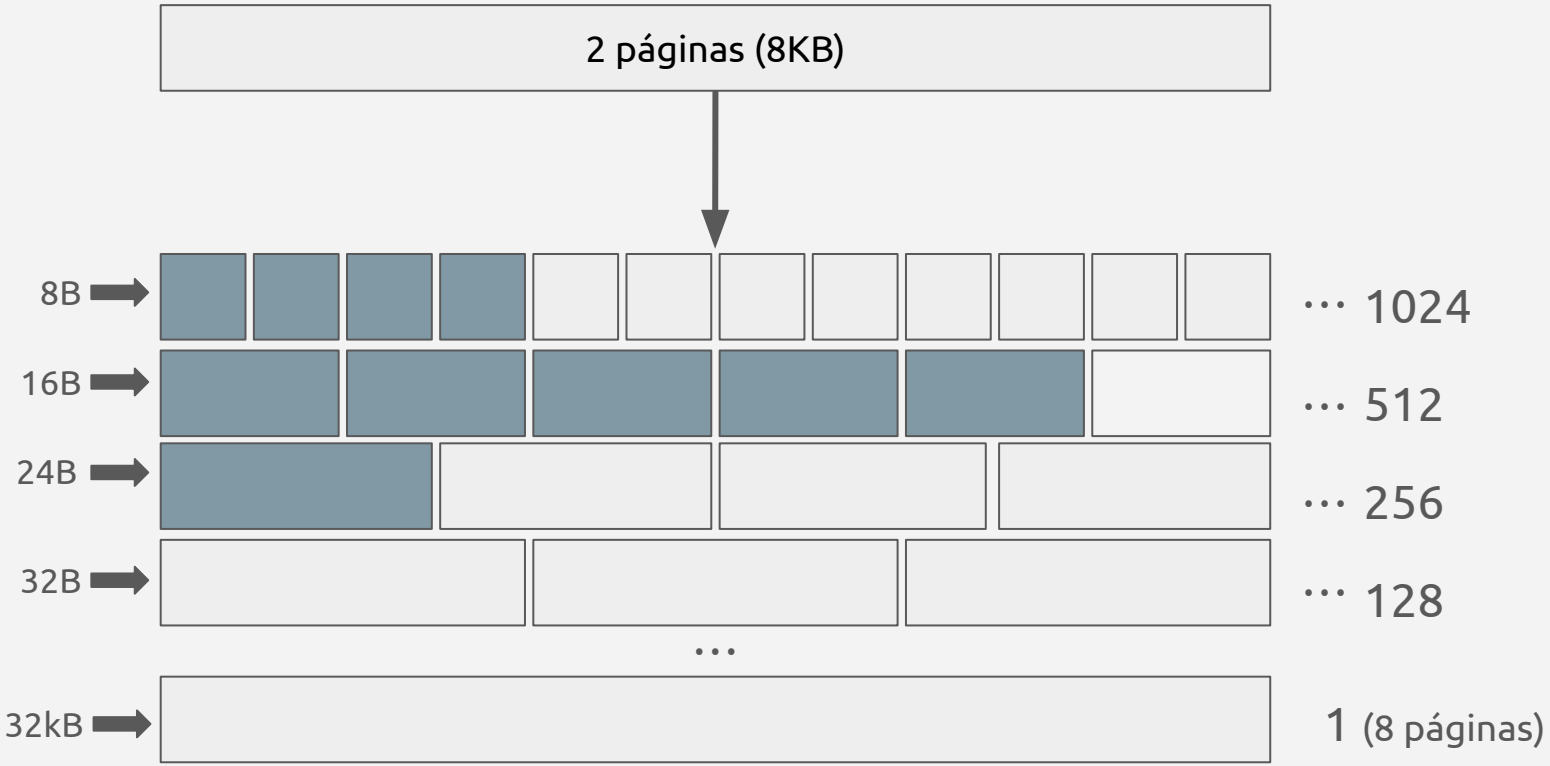
Objetos de tamanho similar são alocados e removidos juntos



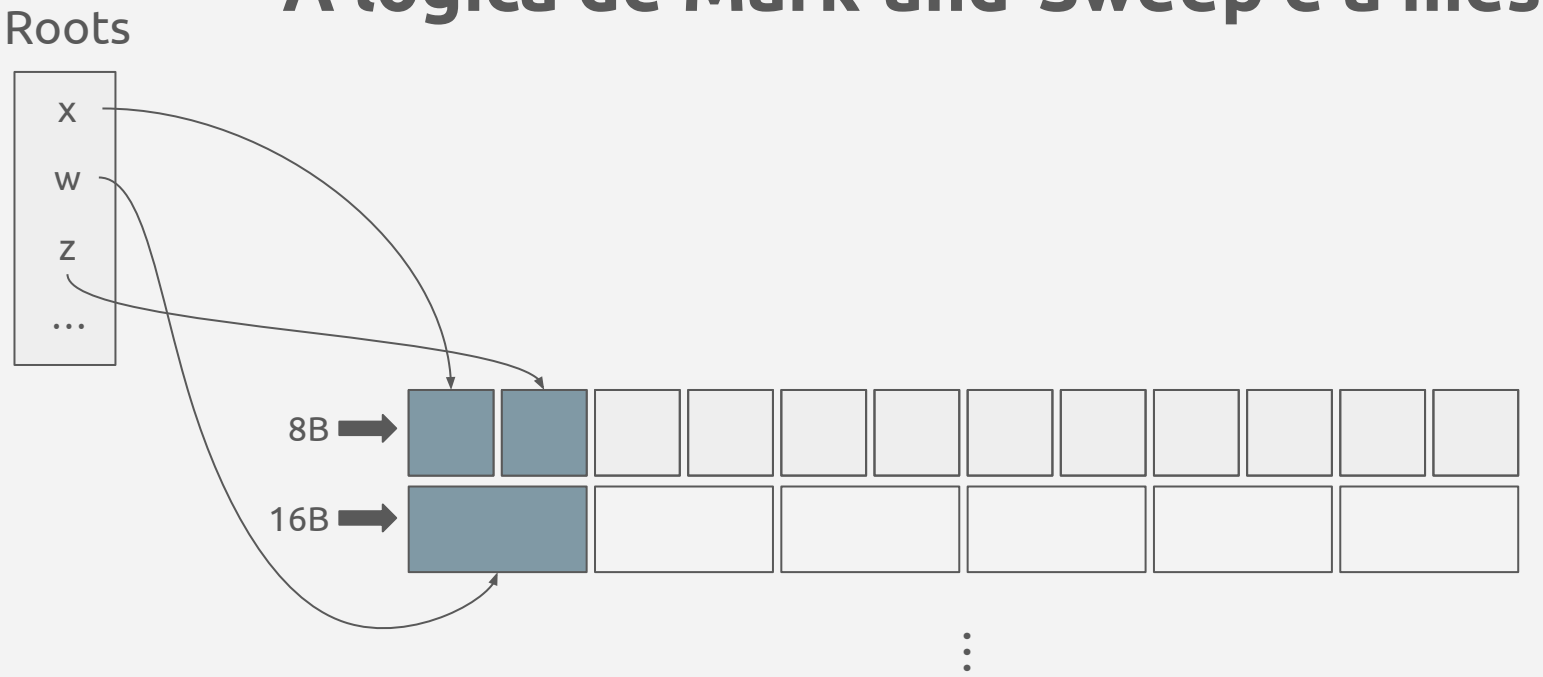
Na realidade é bem mais complicado



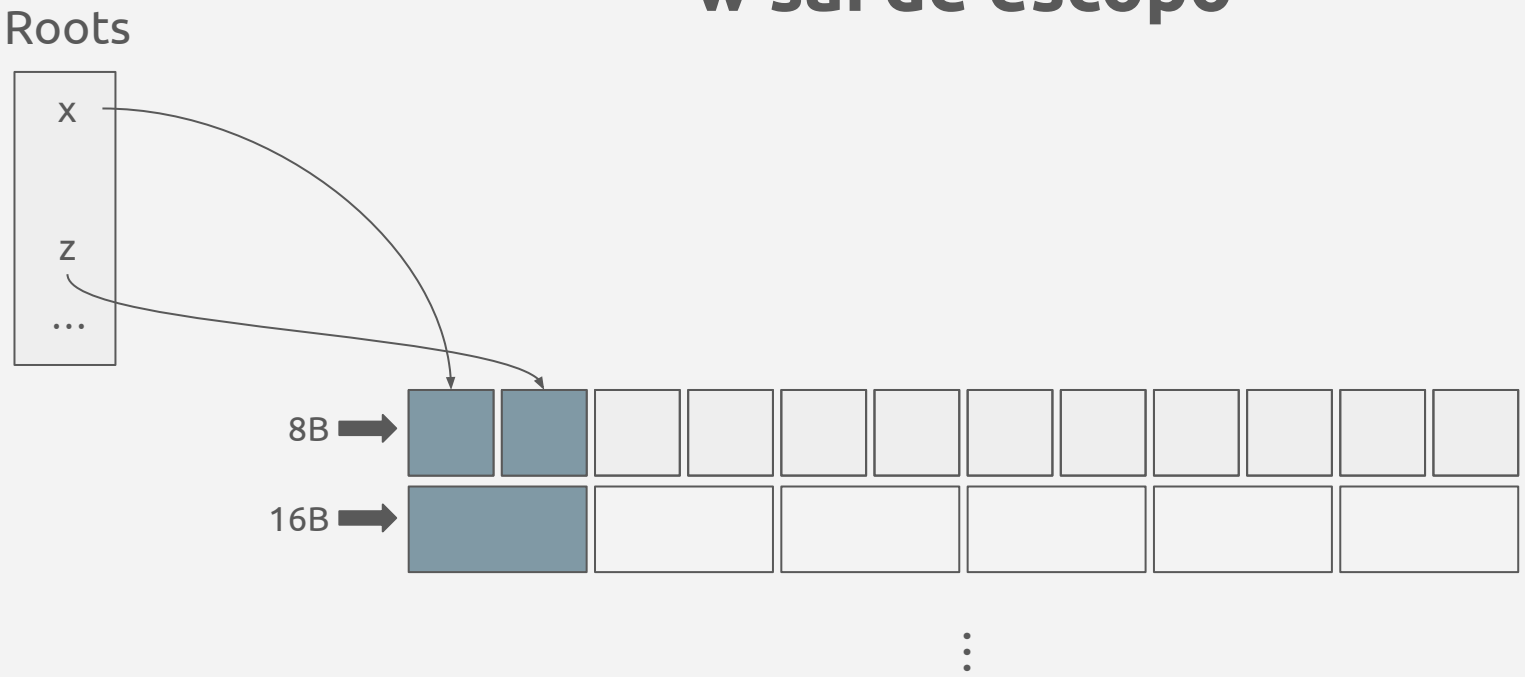
Mas a ideia original é a mesma



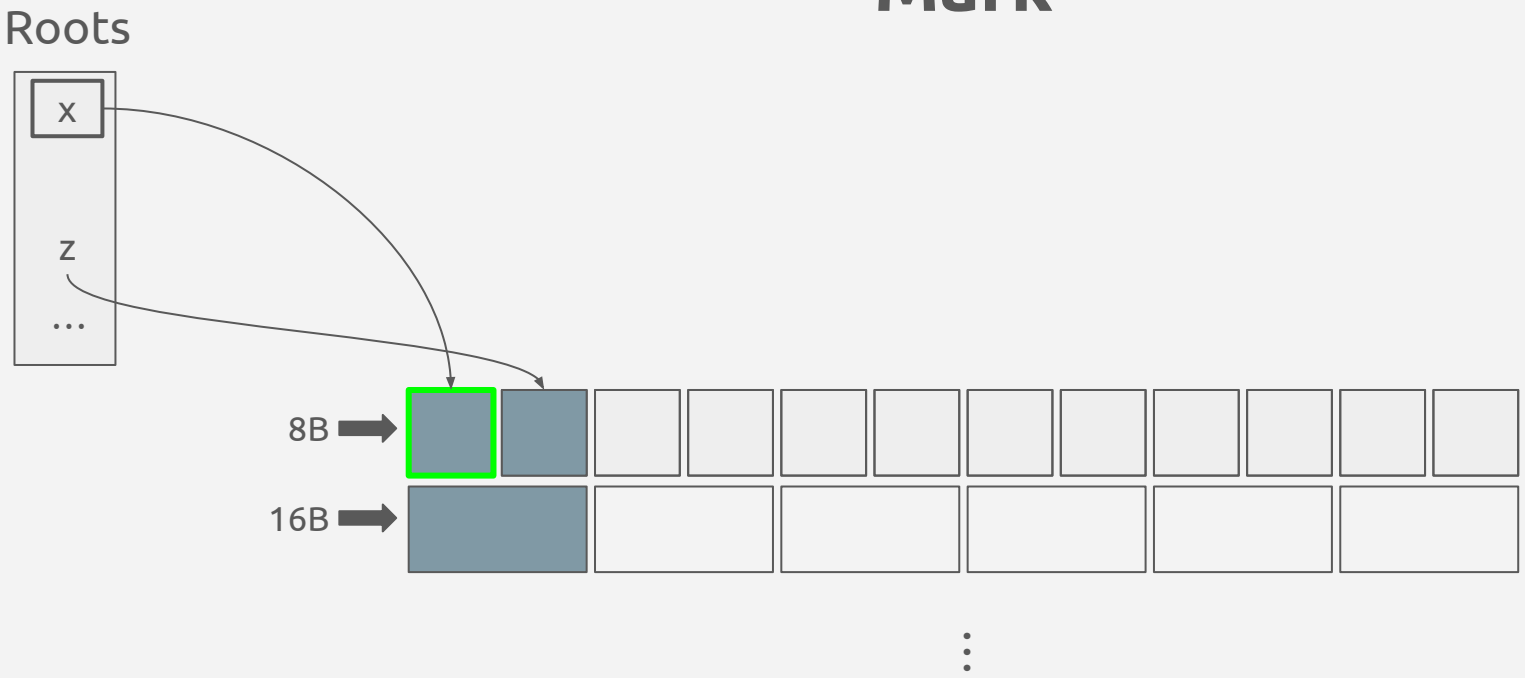
A lógica de Mark-and-Sweep é a mesma



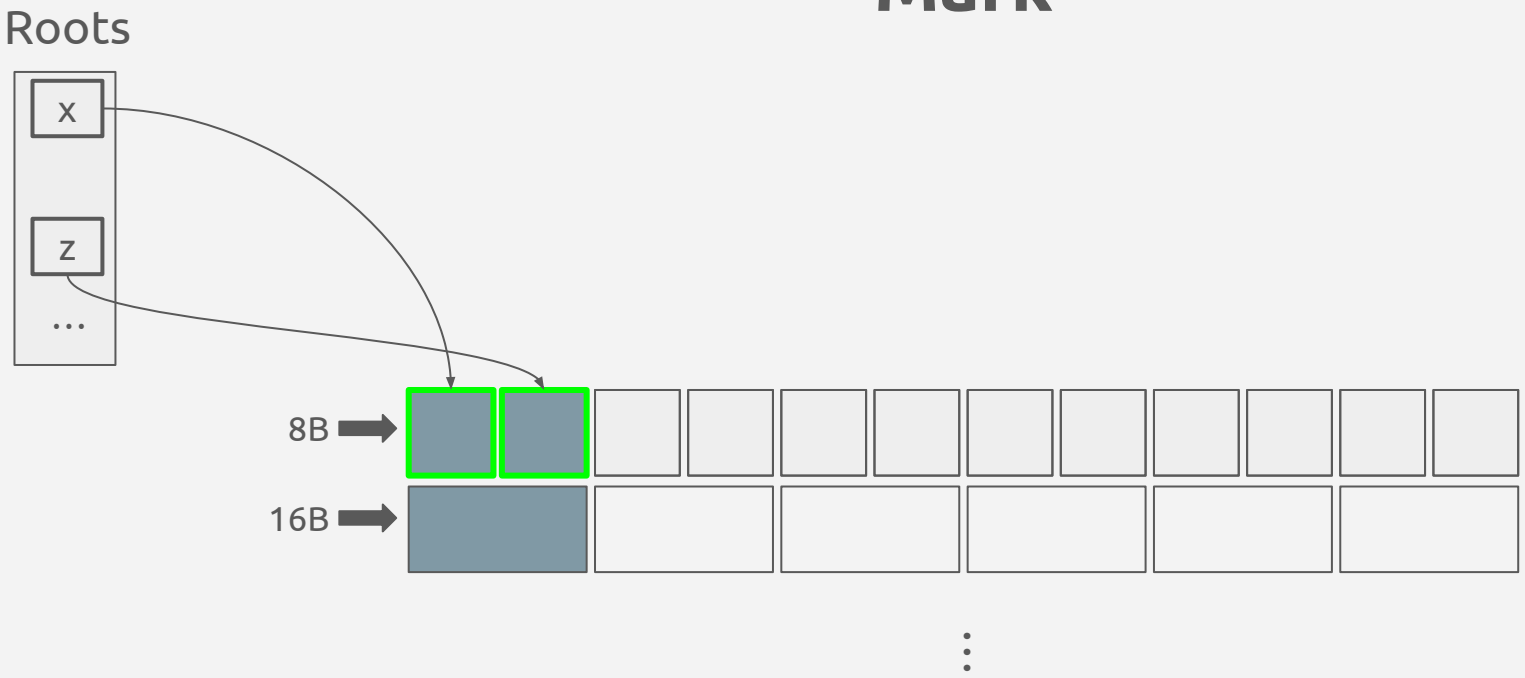
w sai de escopo



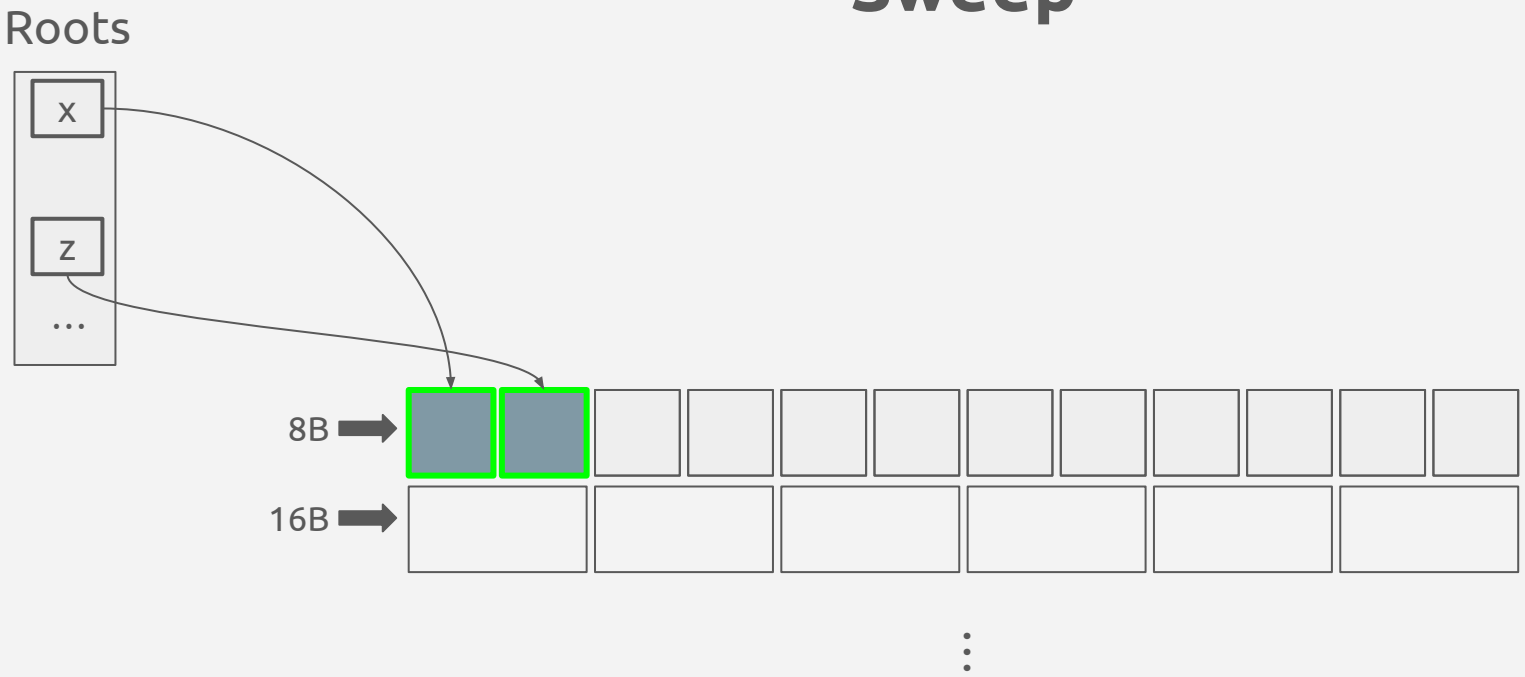
Mark



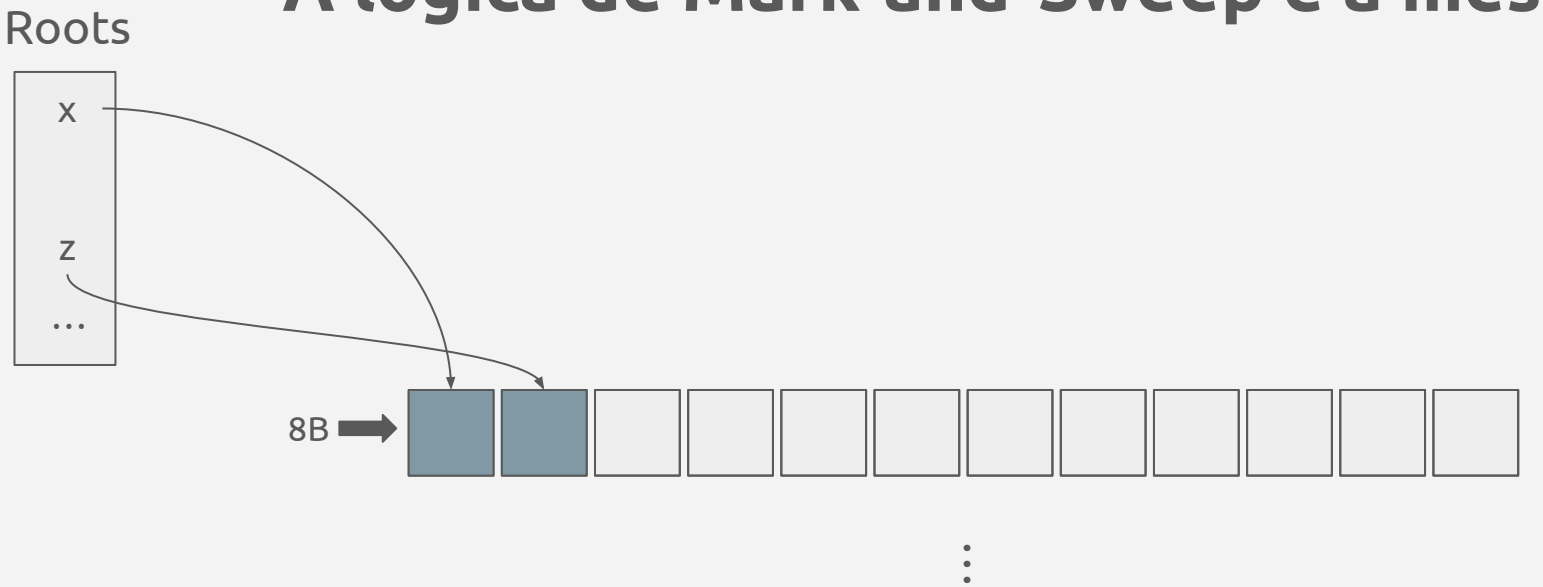
Mark



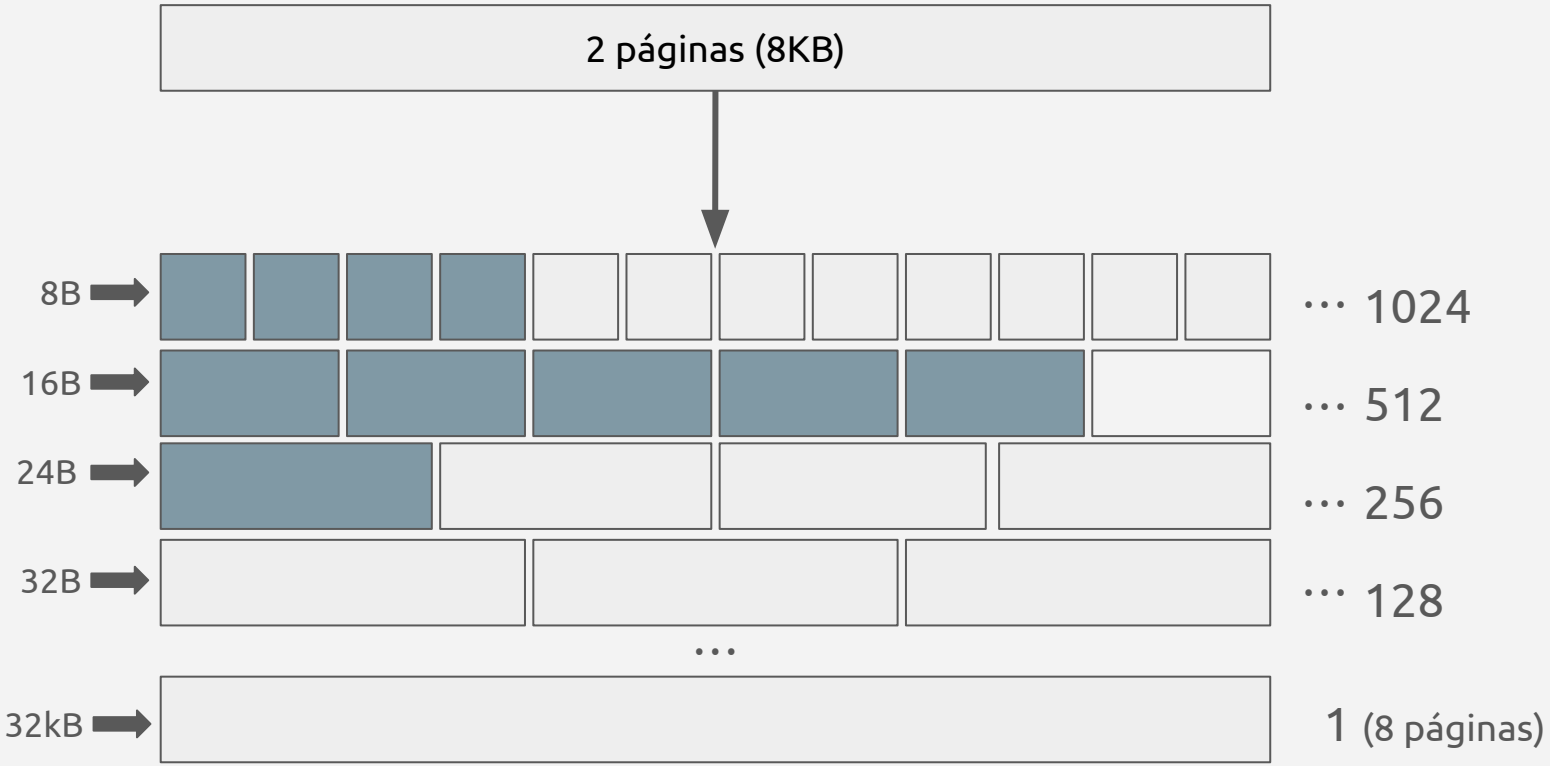
Sweep



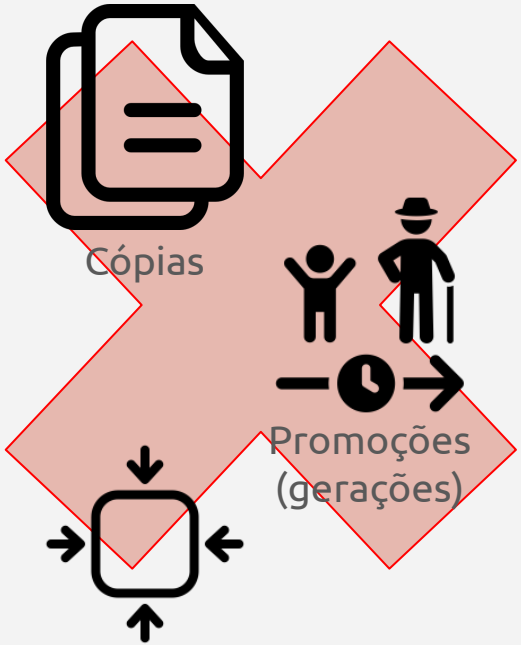
A lógica de Mark-and-Sweep é a mesma



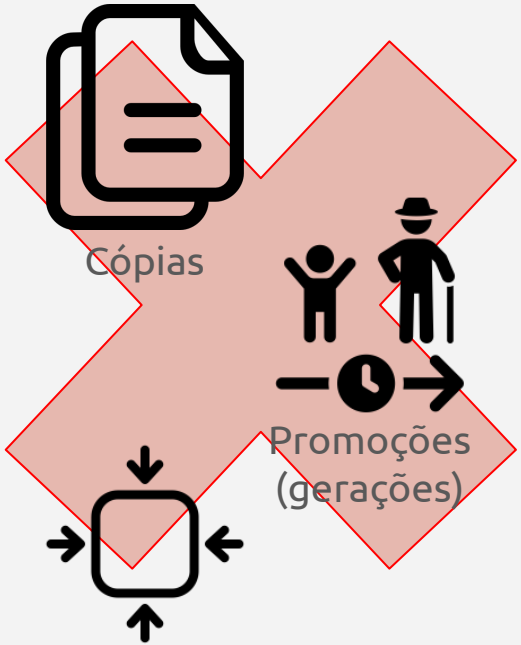
A diferença é como lidar com a fragmentação

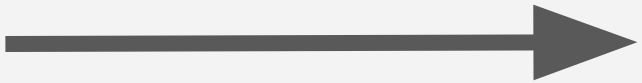


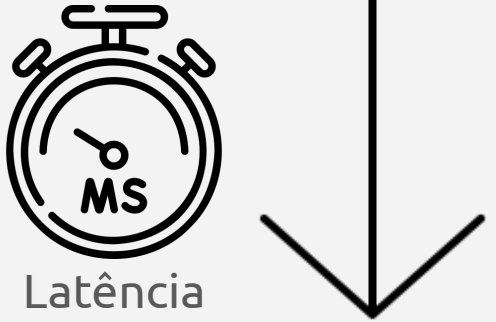
Vantagens



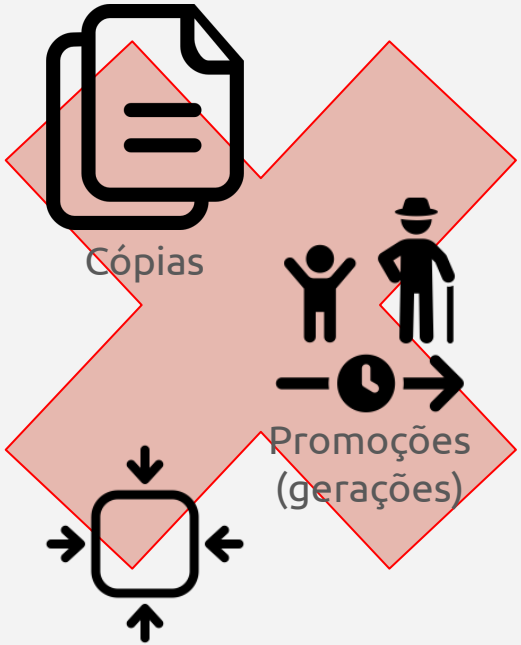
Vantagens

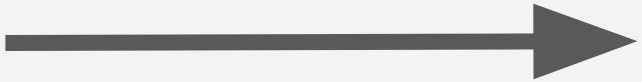




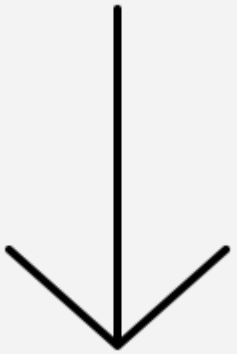


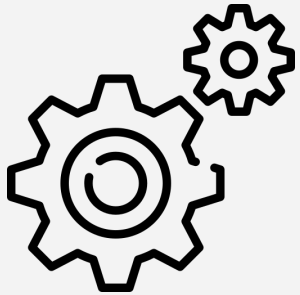
Vantagens





Latência





Mais Fácil

Otimização

Qual é melhor?

Depende

Go vs Java

Característica	Go	Java
Latência	<1ms	~5-100ms (depende da JDK)

Go vs Java

Característica	Go	Java
Latência	<1ms	~5-100ms (depende da JDK)
Throughput (% CPU gasta em GC)	Moderado (~80-90%)	Alta (~95-98%)

Go vs Java

Característica	Go	Java
Latência	<1ms	~5-100ms (depende da JDK)
Throughput (% CPU gasta em GC)	Moderado (~80-90%)	Alta (~95-98%)
Previsibilidade (consistência)	Alta (Não geracional)	Menor (Depende da geração)

Go vs Java

Característica	Go	Java
Latência	<1ms	~5-100ms (depende da JDK)
Throughput (% CPU gasta em GC)	Moderado (~80-90%)	Alta (~95-98%)
Previsibilidade (consistência)	Alta (Não geracional)	Menor (Depende da geração)
Velocidade de alocação	Menor (Avita fragmentação na alocação)	Maior (Bump allocator)

Go vs Java

Característica	Go	Java
Latência	<1ms	~5-100ms (depende da JDK)
Throughput (% CPU gasta em GC)	Moderado (~80-90%)	Alta (~95-98%)
Previsibilidade (consistência)	Alta (Não geracional)	Menor (Depende da geração)
Velocidade de alocação	Menor (Avita fragmentação na alocação)	Maior (Bump allocator)

Os dois são concorrentes: Algoritmo Tri-Color

Otimização

Depende

Otimização

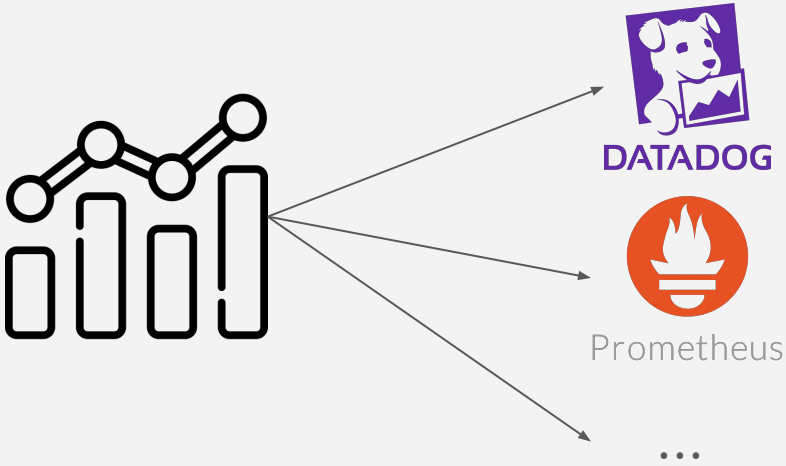
Entender como o GC funciona não é somente uma curiosidade acadêmica

Otimização

Cada combinação de workload com algoritmo de GC gera algo diferente

Otimização

Cada combinação de workload com algoritmo de GC gera algo diferente



Como então bolsas de valores usam Java?

Existem JDKs especializadas em Real Time

Zing

WebsphereRT

JDKs comerciais

Zing

WebsphereRT

Redes sociais

reneepc

/in/reneepc