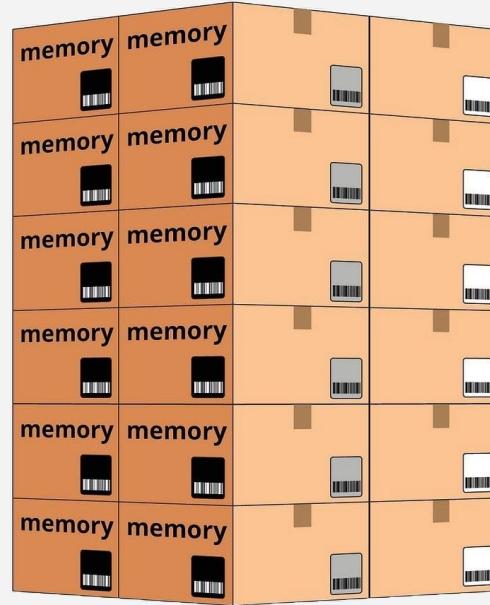


Go Memory Allocation Internals



Renê Cardozo

- Formado em Ciência da Computação - USP
- 6 anos desenvolvendo em Go



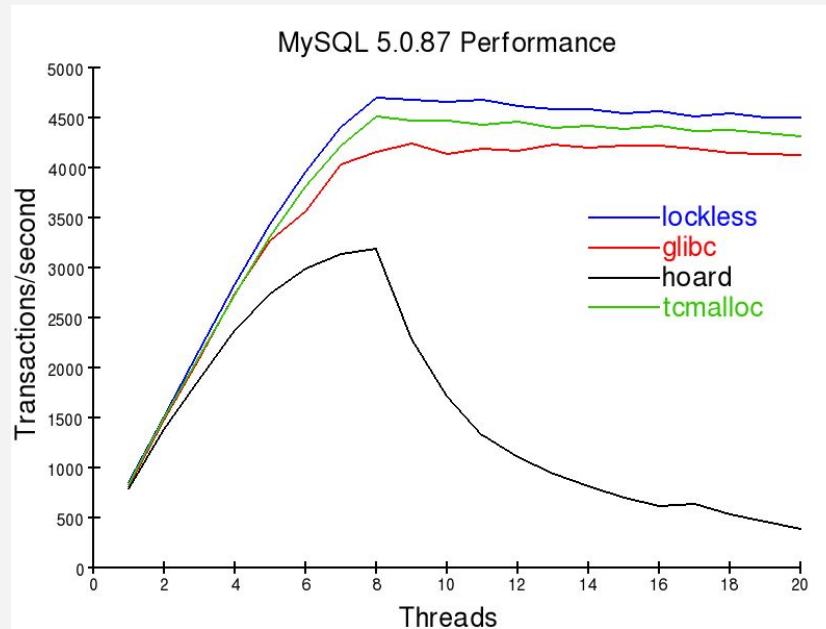
Engine



@reneepc



O algoritmo de alocação faz diferença



O algoritmo de alocação faz diferença

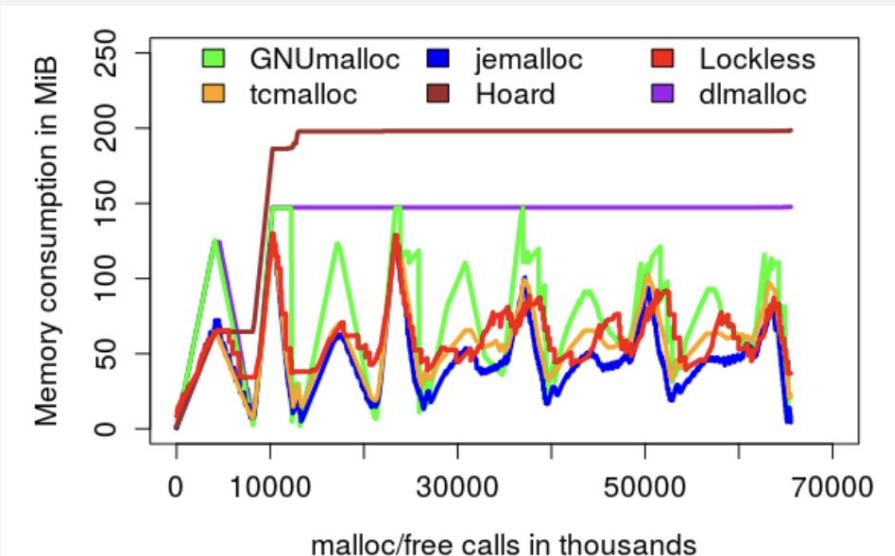


Figure 3: Memory usage over time.

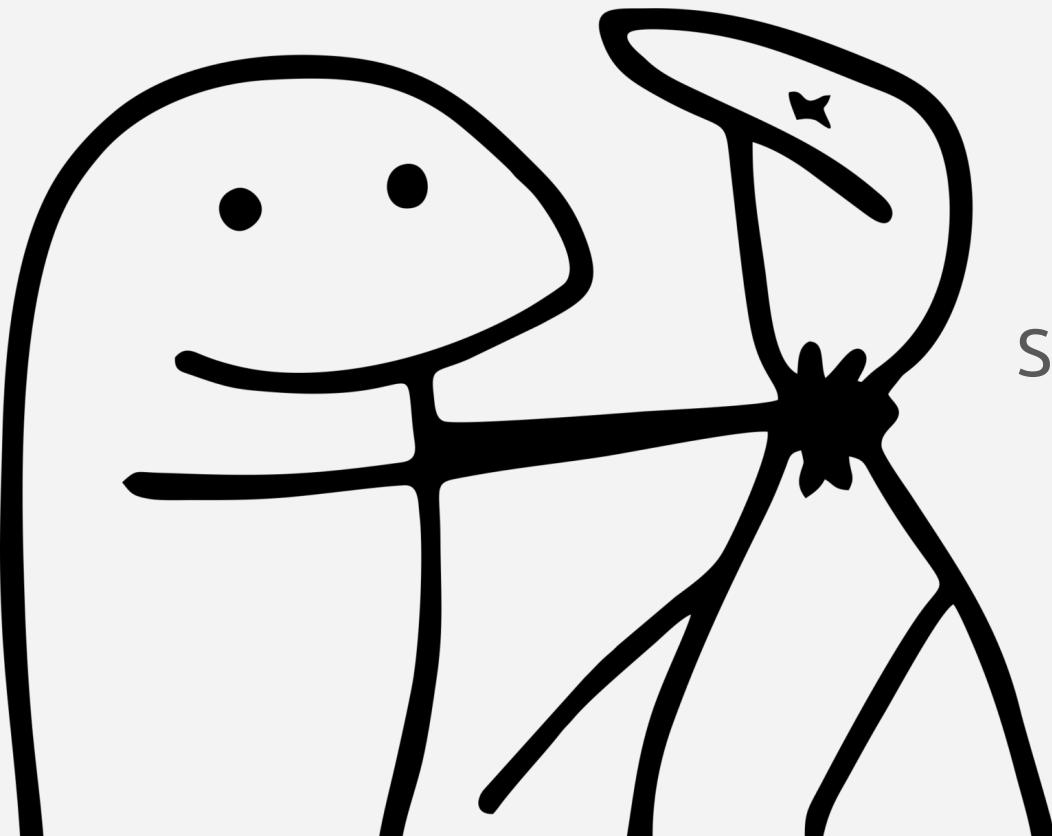
Zavrtanik, Matej & Mihelič, Jurij. (2017). Experimental Evaluation and Comparison of Memory Allocators in the GNU/Linux Operating System.

Mas o real motivo



~~Eu gosto~~
Não tinha ninguém melhor
para palestrar

Se você achar ruim



Submeta sua palestra para
o C4P da GolangSP

Panic

```
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

```
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
}
```

Para cada
Goroutine
uma
Stack

Stack

```
package main

import "fmt"

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

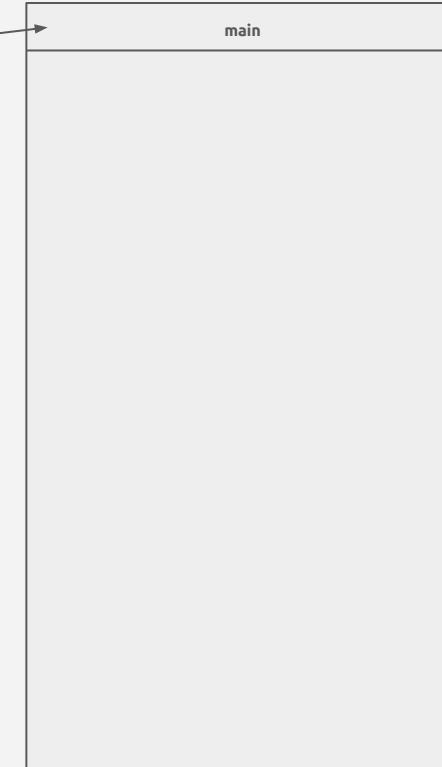
    processPayment(userID, amount)
}

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

```
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

main
userId amount

Stack

```
package main

import "fmt"

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

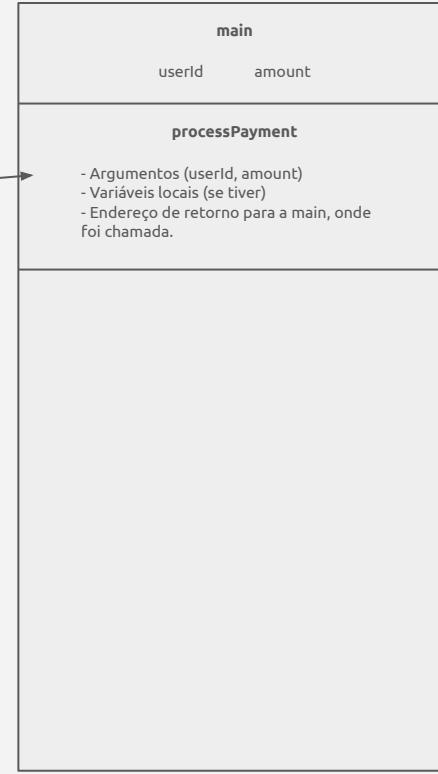
    processPayment(userId, amount)
}

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

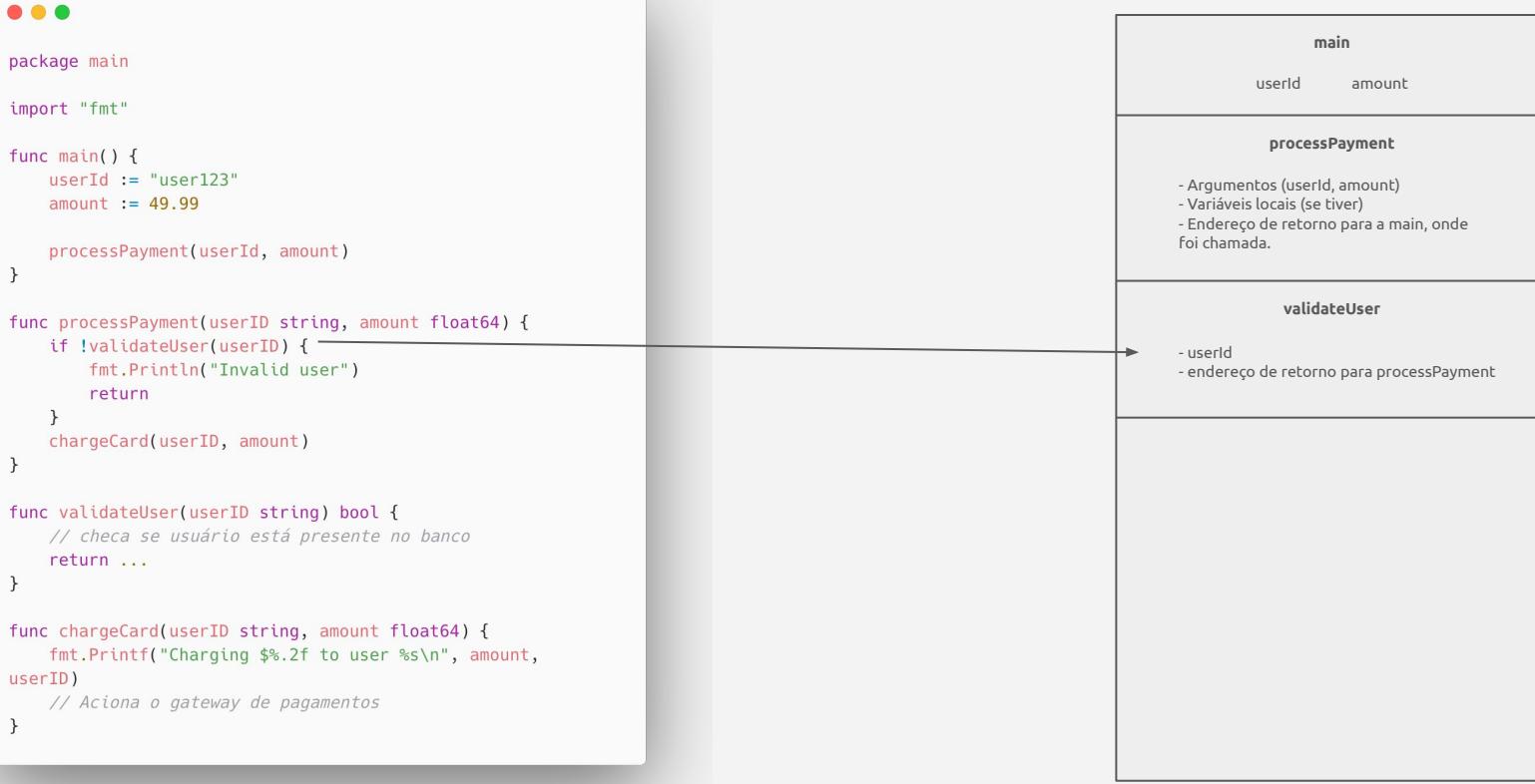
func validateUser(userID string) bool {
    // checa se usuário está presente no banco
    return ...
}

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



Stack

```
package main

import "fmt"

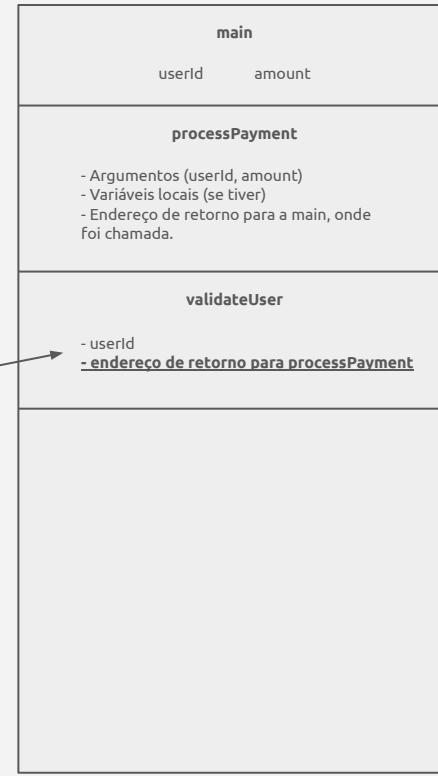
func main() {
    userId := "user123"
    amount := 49.99

    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

```
package main

import "fmt"

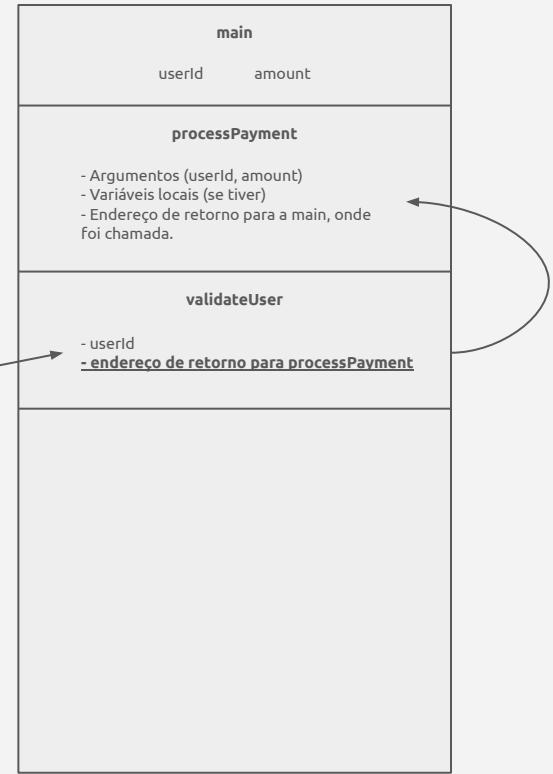
func main() {
    userId := "user123"
    amount := 49.99

    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

```
package main

import "fmt"

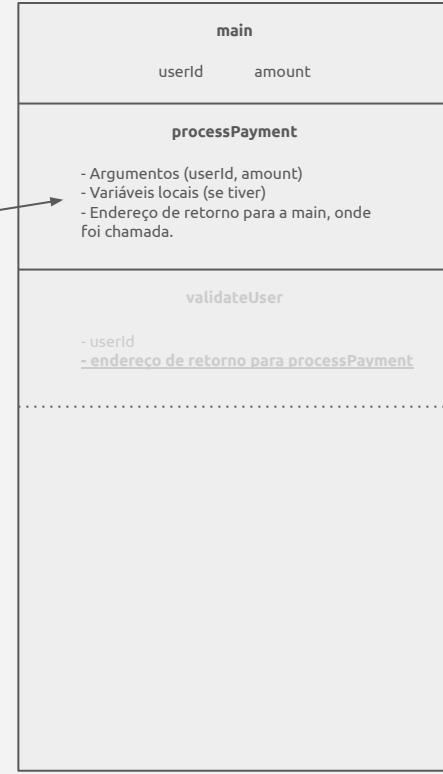
func main() {
    userId := "user123"
    amount := 49.99

    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

```
package main

import "fmt"

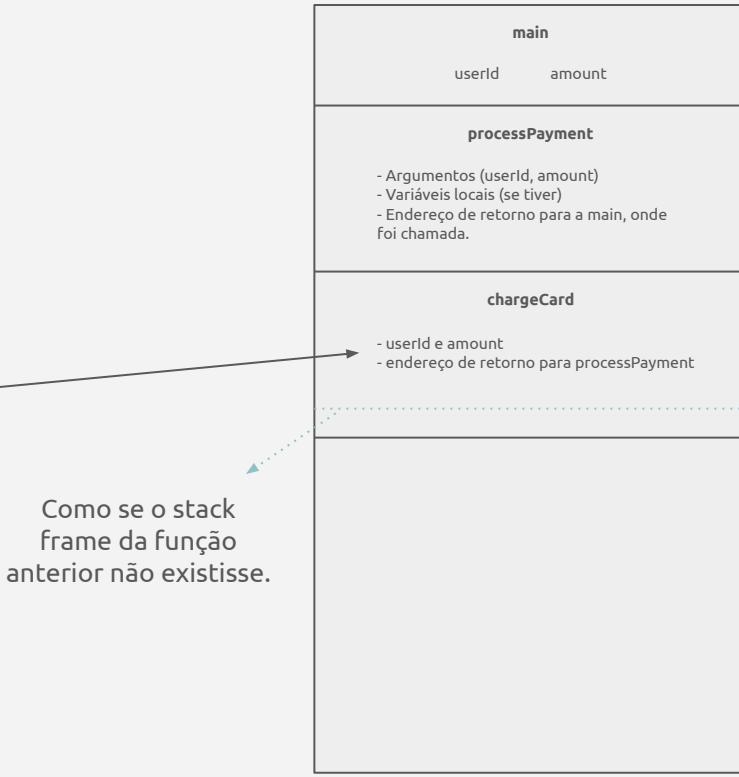
func main() {
    userId := "user123"
    amount := 49.99

    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

```
package main

import "fmt"

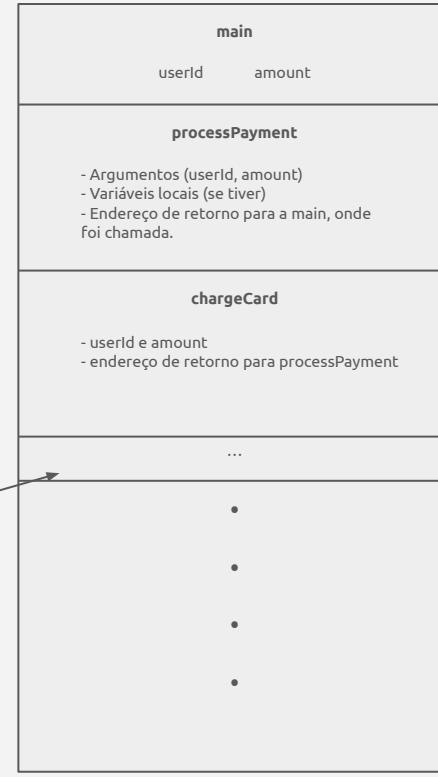
func main() {
    userId := "user123"
    amount := 49.99

    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

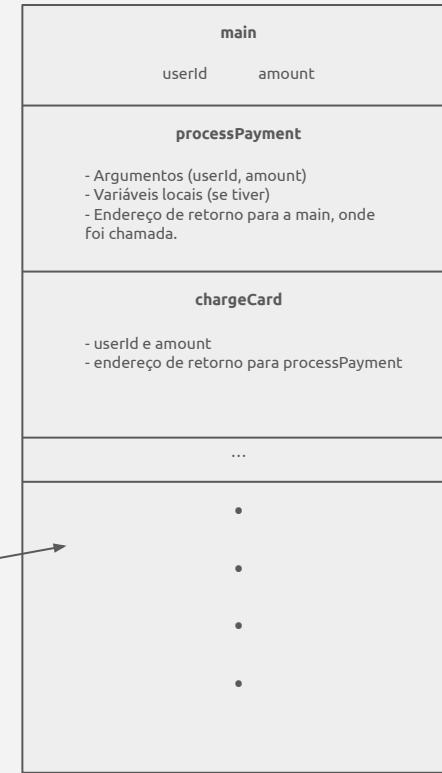
main.main()
    payment.go:6 +0x30

exit status 2

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

O runtime do Go caminha os endereços de retorno da stack para montar a stack trace.

runtime/traceback.go



Quando a stack não é suficiente

```
package main

import (
    "fmt"
)

type Notification struct {
    UserID string
    Channel string
    Message string
}

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

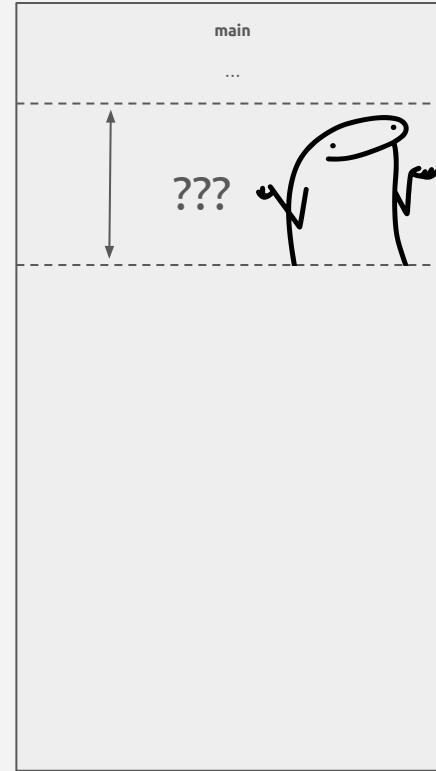
    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

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

```
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
    }
}

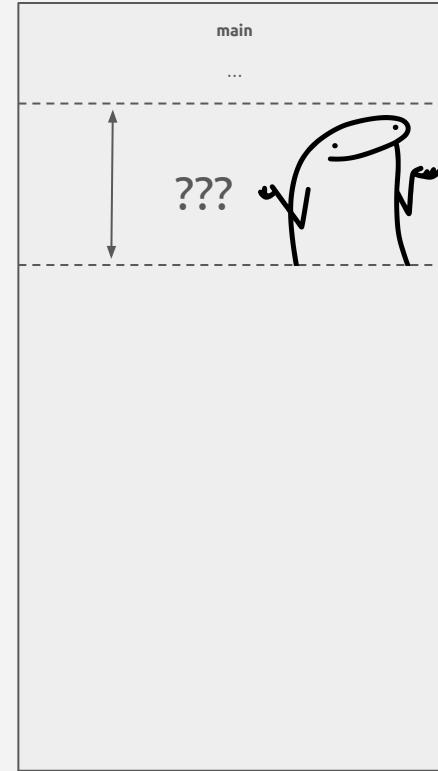
func buildNotifications(userID string, amount float64) []Notification {
    notifications := make([]Notification, 0)

    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

```
package main

import (
    "fmt"
)

type Notification struct {
    UserID string
    Channel string
    Message string
}

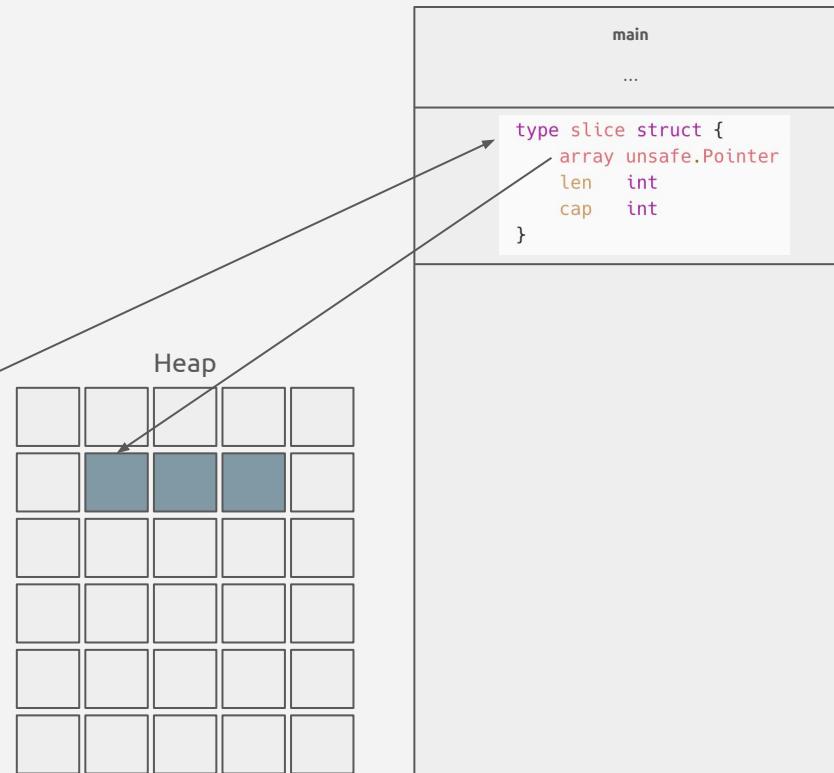
func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    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

```
package main

import (
    "fmt"
)

type Notification struct {
    UserID string
    Channel string
    Message string
}

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

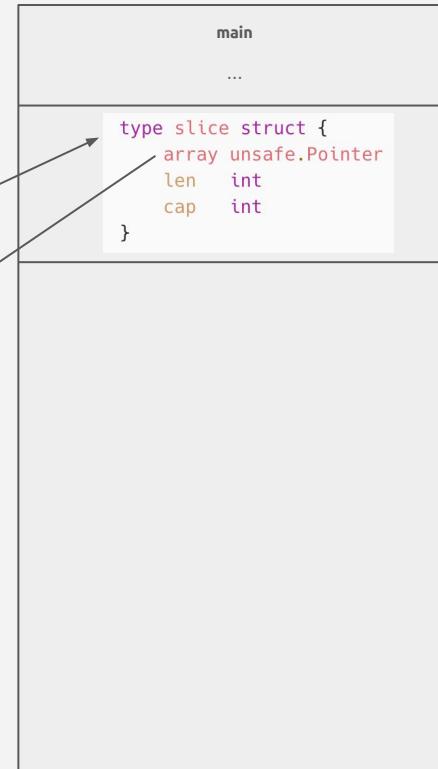
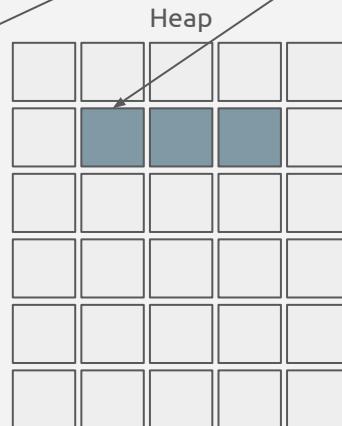
    for _, n := range notifications {
        // Persiste em outbox ou envia notificação
    }
}

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
}
```

new()
make()



Heap

```
package main

import (
    "fmt"
)

type Notification struct {
    UserID string
    Channel string
    Message string
}

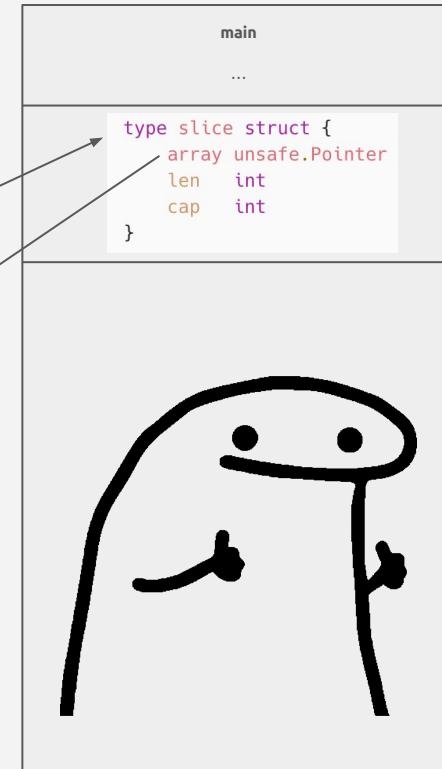
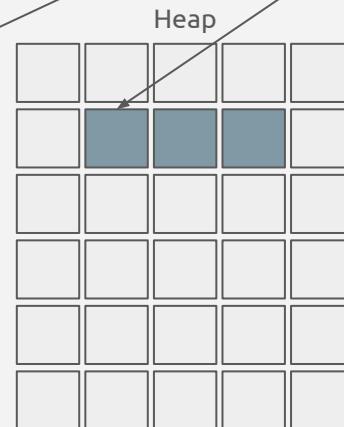
func main() {
    // ...
    // processPayment
    // ...
    notifications := buildNotifications("user123", 49.99)

    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



```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```

A heap também é usada para compartilhar referências a uma variável específica.

Heap



```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



Heap



```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



Heap

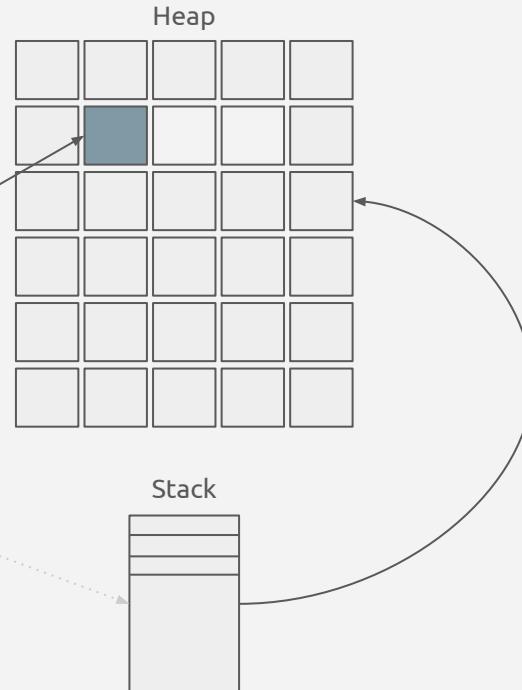


```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



O compilador
move para a heap
automaticamente

Heap

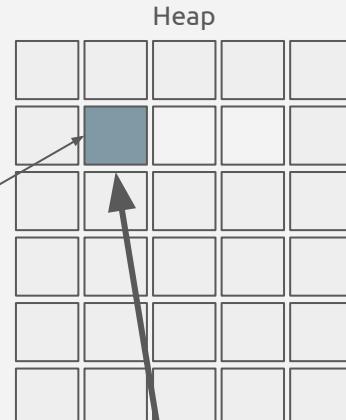


```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



Stack



Local da heap

Heap

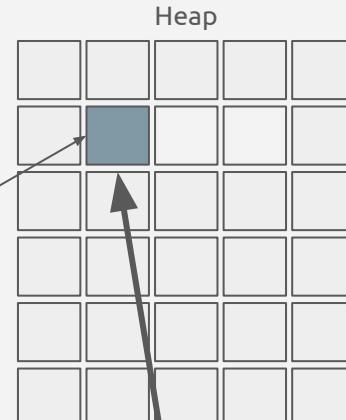


```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



Ponteiro

Escape analysis

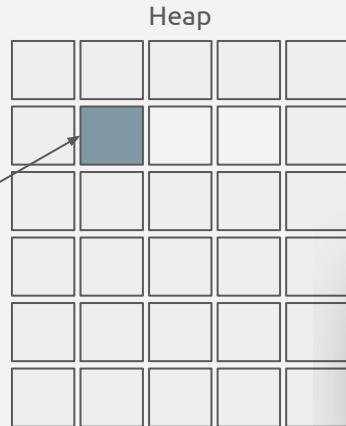


```
package amount

type Amount struct {
    Value      int
}

func NewAmountHeap(value int) *Amount
{
    amt := Amount{value}
    return &amt // causes heap escape
}

func NewAmountStack(value int) Amount
{
    amt := Amount{value}
    return amt
}
```



Heap



Stack

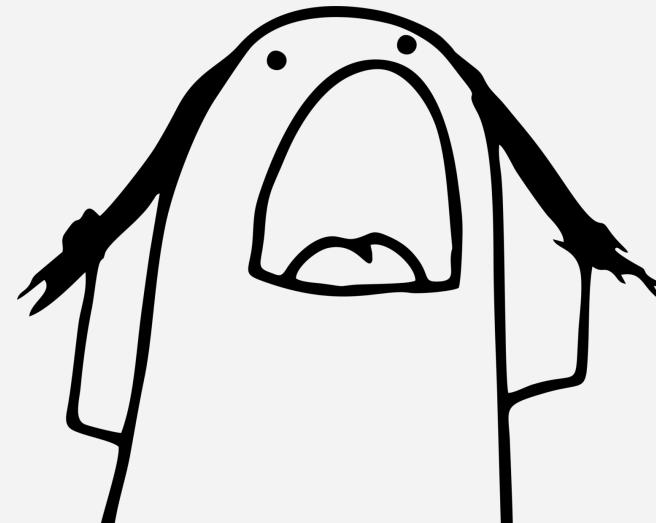


```
$> go build -gcflags="-m" main.go
# memory/amount
./amount.go:8:5: moved to heap:
amt
```

Escape analysis

```
TEXT memory.NewAmountStack(SB)
// amount.go:13
MOVD.W R30, -48(RSP)
MOVD R29, -8(RSP)
SUB $8, RSP, R29
MOVD R0, 56(RSP)
MOVD R1, 64(RSP)
MOVD R2, 72(RSP)
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)

// amount.go:14
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)
MOVD 56(RSP), R0
MOVD R0, 16(RSP)
MOVD 64(RSP), R1
MOVD 72(RSP), R2
MOVD R1, 24(RSP)
MOVD R2, 32(RSP)
ADD $40, RSP, R29
ADD $48, RSP, RSP
RET
```



Escape analysis

```
TEXT memory.NewAmountStack(SB)
// amount.go:13
MOVD.W R30, -48(RSP)
MOVD R29, -8(RSP)
SUB $8, RSP, R29
MOVD R0, 56(RSP)
MOVD R1, 64(RSP)
MOVD R2, 72(RSP)
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)

// amount.go:14
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)
MOVD 56(RSP), R0
MOVD R0, 16(RSP)
MOVD 64(RSP), R1
MOVD 72(RSP), R2
MOVD R1, 24(RSP)
MOVD R2, 32(RSP)
ADD $40, RSP, R29
ADD $48, RSP, RSP
RET
```

```
TEXT memory.NewAmountHeap(SB)
// amount.go:8
MOVD 16(R28), R16
CMP R16, RSP
BLS 39(PC)
MOVD.W R30, -80(RSP)
MOVD R29, -8(RSP)
SUB $8, RSP, R29
MOVD R0, 88(RSP)
MOVD R1, 96(RSP)
MOVD R2, 104(RSP)
MOVD ZR, 32(RSP)

// amount.go:9
ADRP type:memory.Amount, R0
ADD $0, R0, R0
CALL runtime.newobject
MOVD R0, 64(RSP)
STP (ZR, ZR), 40(RSP)
MOVD ZR, 56(RSP)
MOVD 88(RSP), R1
MOVD R1, 40(RSP)
MOVD 96(RSP), R2
MOVD 104(RSP), R3
MOVD R2, 48(RSP)
MOVD R3, 56(RSP)
MOVD 64(RSP), R4
MOVD R1, (R4)
MOVD R3, 16(R4)
ADRP runtime.writeBarrier, R22
MOVWU (R27), R1
CBZW R1, 2(PC)
JMP 2(PC)
JMP 6(PC)
CALL runtime.gcWriteBarrier
MOVD R2, (R25)
MOVD 8(R4), R1
MOVD R1, 8(R25)
JMP 1(PC)
MOVD R2, 8(R4)

// amount.go:10
MOVD 64(RSP), R0
MOVD R0, 32(RSP)
LDP -8(RSP), (R29, R30)
ADD $80, RSP, RSP
RET
MOVD R0, 8(RSP)
MOVD R1, 16(RSP)
MOVD R2, 24(RSP)
MOVD R30, R3
CALL runtime.morestack_noctxt
MOVD 8(RSP), R0
MOVD 16(RSP), R1
MOVD 24(RSP), R2
JMP memory.NewAmountHeap(SB)
```

Escape analysis

```
TEXT memory.NewAmountStack(SB)
// amount.go:13
MOVD.W R30, -48(RSP)
MOVD R29, -8(RSP)
SUB $8, RSP, R29
MOVD R0, 56(RSP)
MOVD R1, 64(RSP)
MOVD R2, 72(RSP)
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)

// amount.go:14
STP (ZR, ZR), 16(RSP)
MOVD ZR, 32(RSP)
MOVD 56(RSP), R0
MOVD R0, 16(RSP)
MOVD 64(RSP), R1
MOVD 72(RSP), R2
MOVD R1, 24(RSP)
MOVD R2, 32(RSP)
ADD $40, RSP, R29
ADD $48, RSP, RSP
RET
```

Chamada para o alocador

```
TEXT memory.NewAmountHeap(SB)
// amount.go:8
MOVD 16(R28), R16
CMP R16, RSP
BLS 39(PC)
MOVD.W R30, -80(RSP)
MOVD R29, -8(RSP)
SUB $8, RSP, R29
MOVD R0, 88(RSP)
MOVD R1, 96(RSP)
MOVD R2, 104(RSP)
MOVD ZR, 32(RSP)

// amount.go:9
ADRP type:memory.Amount, R0
ADD $0, R0, R0
CALL runtime.newobject
MOVD R0, 64(RSP)
STP (ZR, ZR), 40(RSP)
MOVD ZR, 56(RSP)
MOVD 88(RSP), R1
MOVD R1, 40(RSP)
MOVD 96(RSP), R2
MOVD 104(RSP), R3
MOVD R2, 48(RSP)
MOVD R3, 56(RSP)
MOVD 64(RSP), R4
MOVD R1, (R4)
MOUD R3, 16(B4)
ADRP runtime.writeBarrier, R27
MOVWU (R27), R1
CBZW R1, 2(PC)
JMP 2(PC)
JMP 6(PC)
CALL runtime.gcWriteBarrier2
MOVD R2, (R25)
MOVD 8(R4), R1
MOVD R1, 8(R25)
JMP 1(PC)
MOVD R2, 8(R4)

// amount.go:10
MOVD 64(RSP), R0
MOVD R0, 32(RSP)
LDP -8(RSP), (R29, R30)
ADD $80, RSP, RSP
RET
MOVD R0, 8(RSP)
MOVD R1, 16(RSP)
MOVD R2, 24(RSP)
MOVD R30, R3
CALL runtime.morestack_noctxt
MOVD 8(RSP), R0
MOVD 16(RSP), R1
MOVD 24(RSP), R2
JMP memory.NewAmountHeap(SB)
```

Tracking pelo garbage collector

**If the closure outlives the stack frame of
the variables it references, the variable
must be allocated on the heap**

Functions that return interfaces also escape to the heap.

O trabalho do alocação

- Como lidar com a paginação do SO
- Como lidar com a fragmentação

O trabalho do alocador

RAM



96KB

Hierarquia de memória

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

Hierarquia de memória

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

Hierarquia de memória

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

Hierarquia de memória

40x mais rápido que
um Pentium (1995)
100Mhz

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

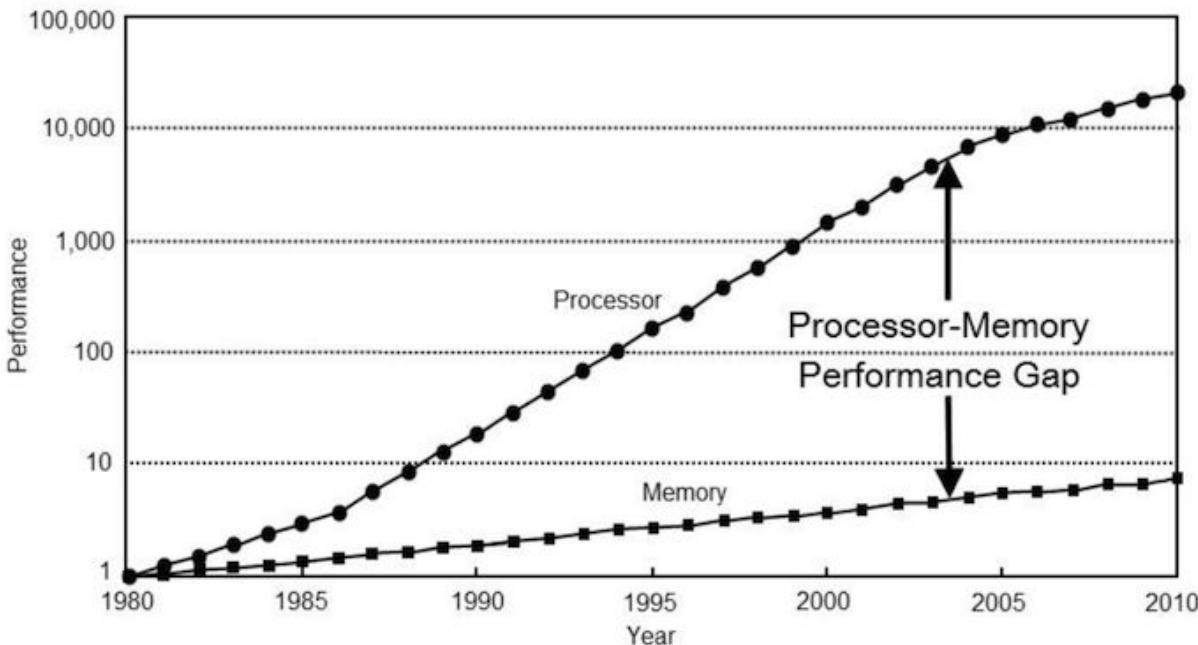
Hierarquia de memória

40x mais rápido que
um Pentium (1995)
100Mhz

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

Melhorou menos de 2x-3x nos últimos 30 anos

Memory Wall



Computer Architecture: A Quantitative Approach by John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

Paginação



mmap(8 bytes)



RAM



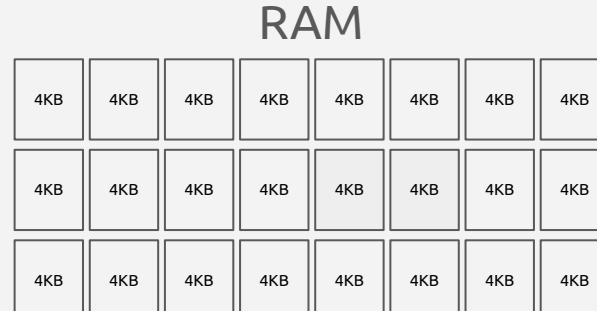
Paginação

mmap(8 bytes)



=GO

4KB



Paginação

=GO

mmap(5KB)



RAM

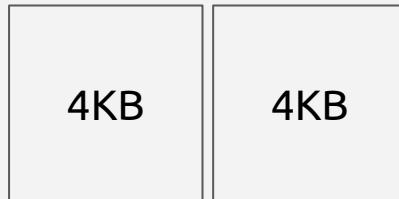


Paginação

mmap(5KB)



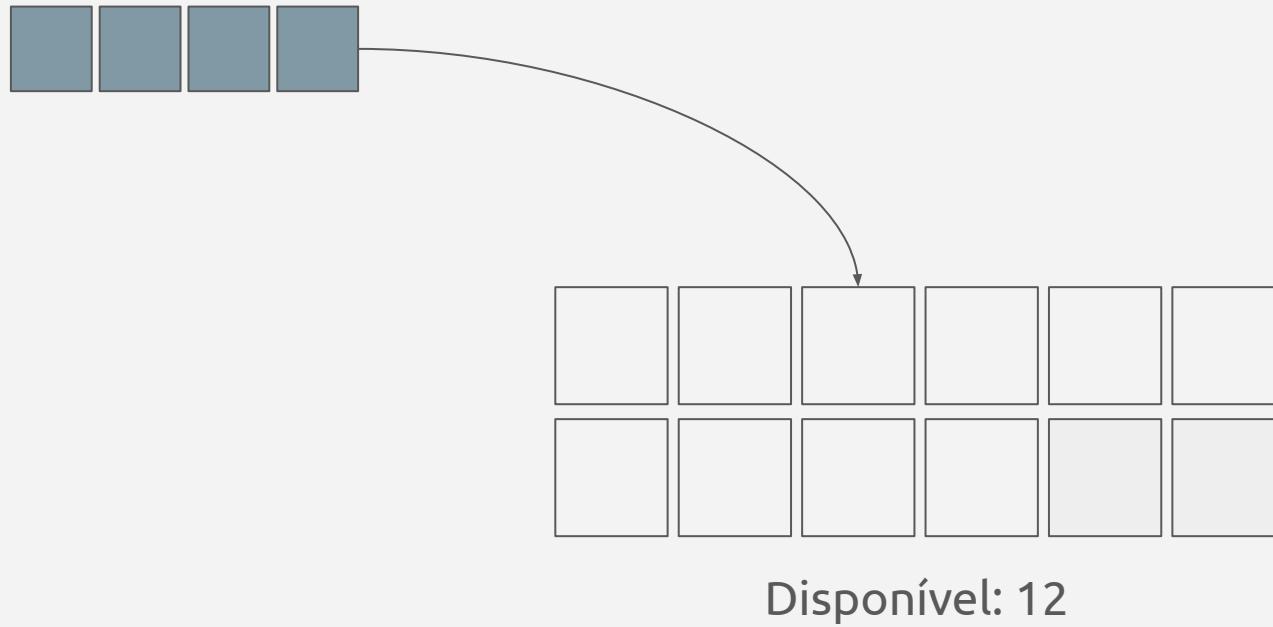
=GO



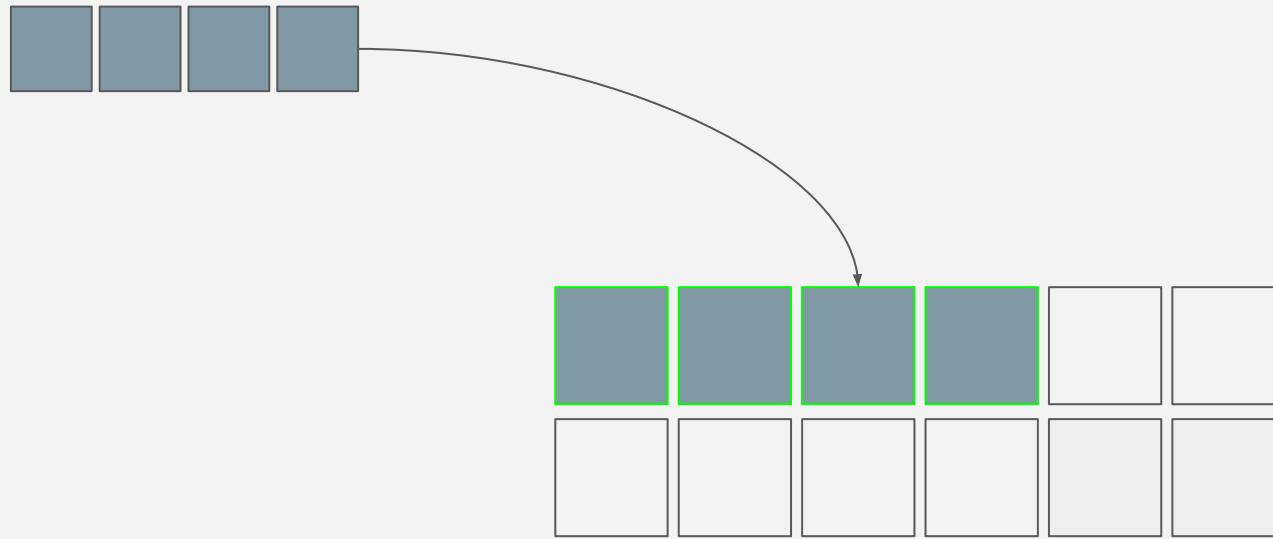
RAM



O trabalho do alocador

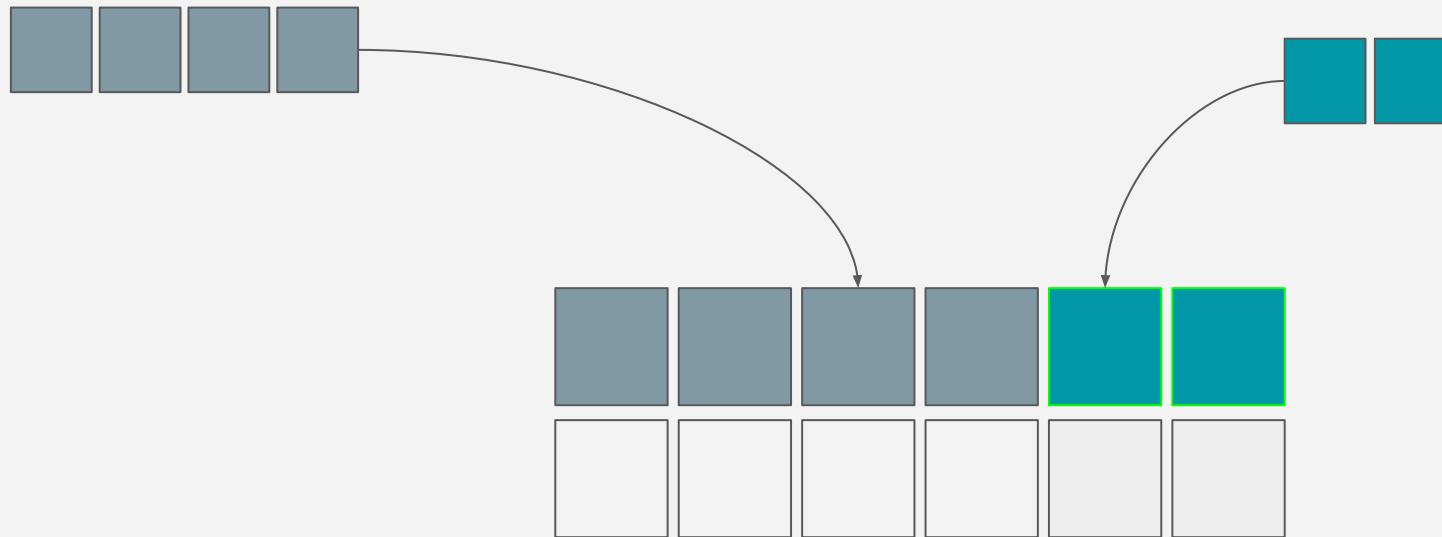


O trabalho do alocador



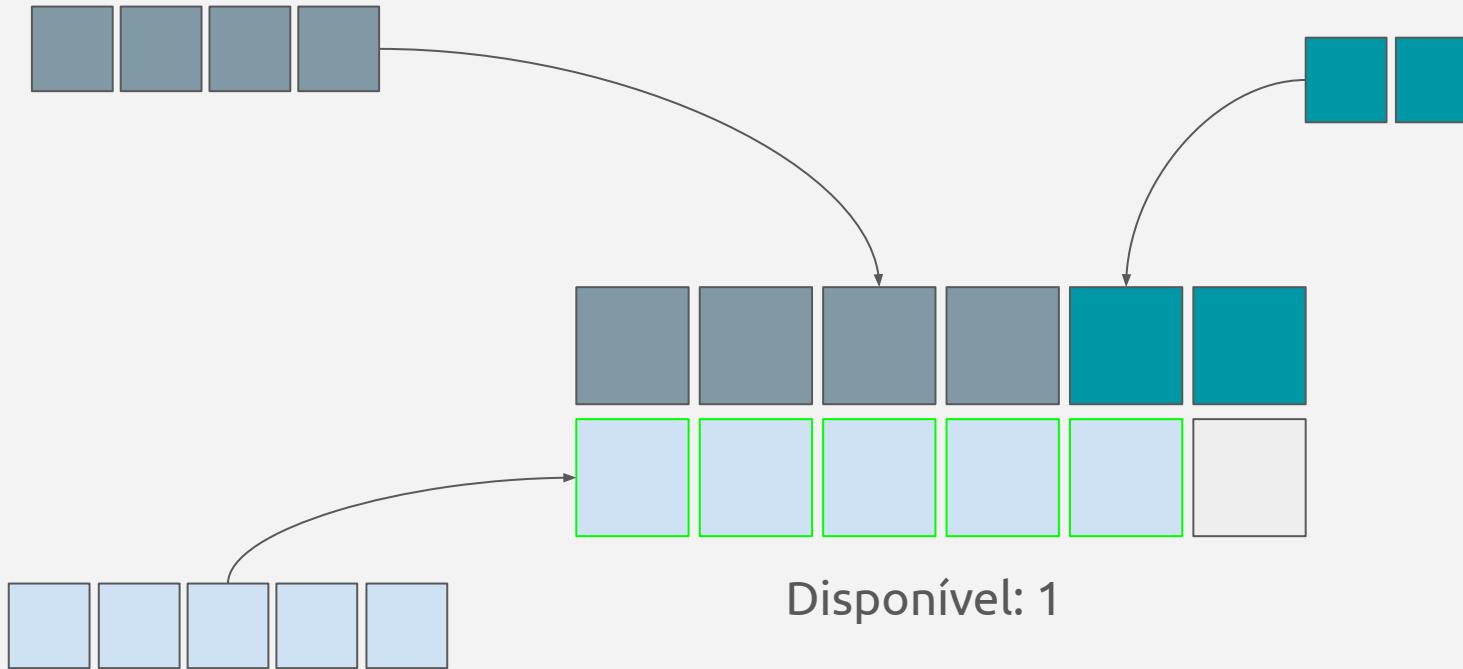
Disponível: 8

O trabalho do alocador

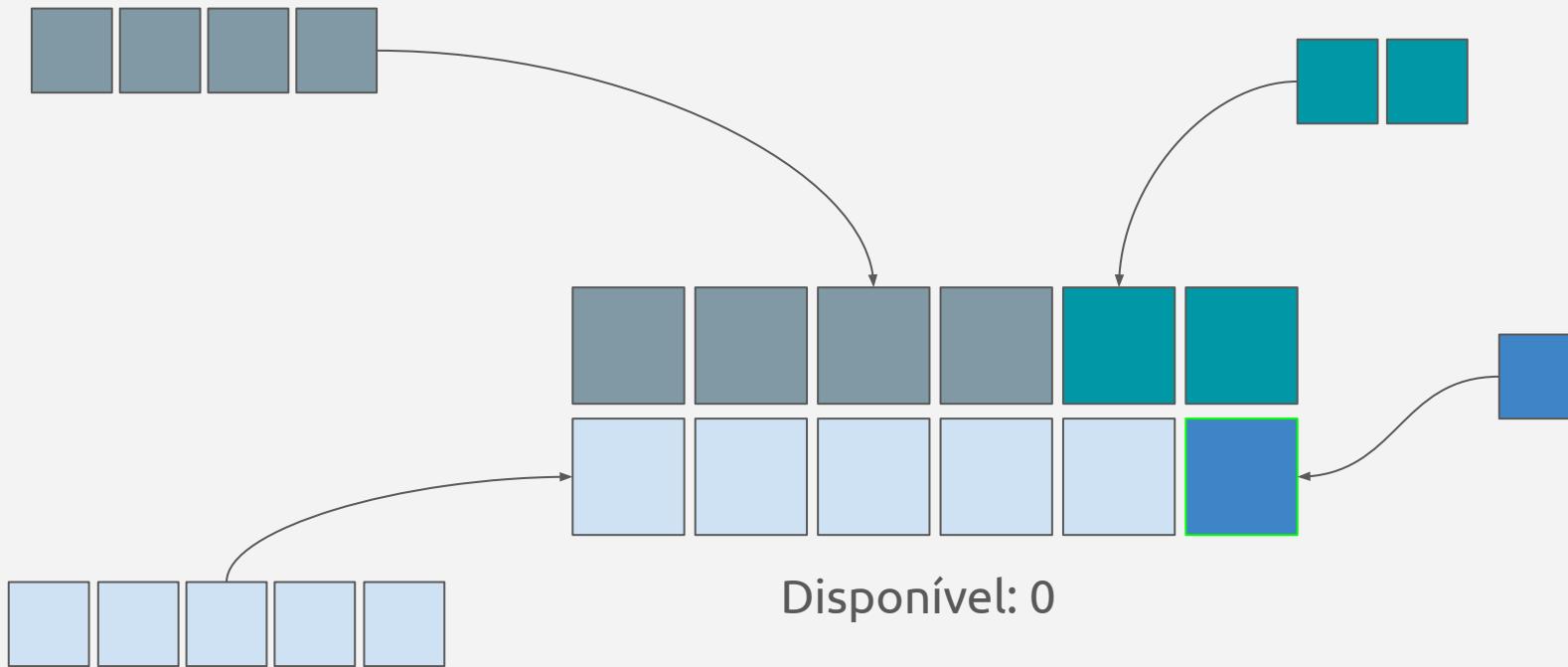


Disponível: 6

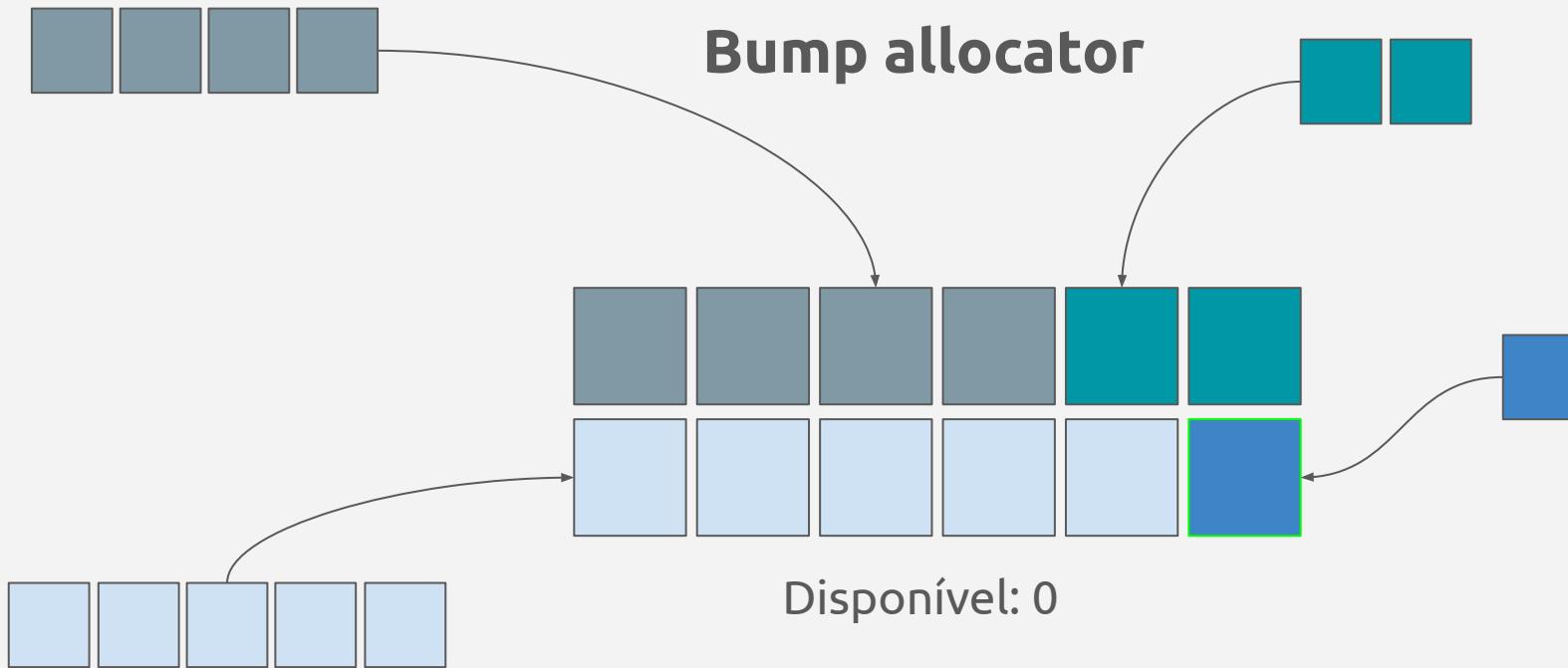
O trabalho do alocador



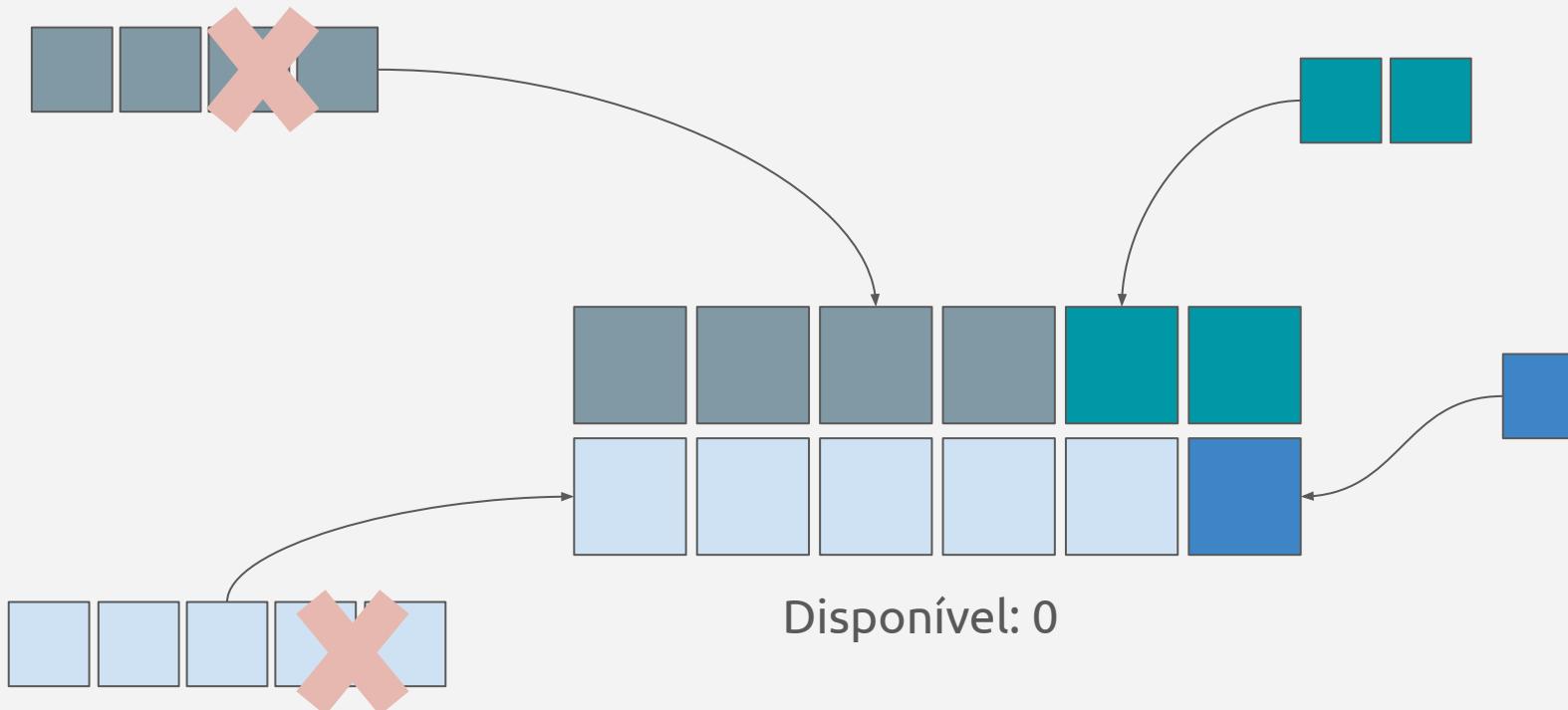
O trabalho do alocador



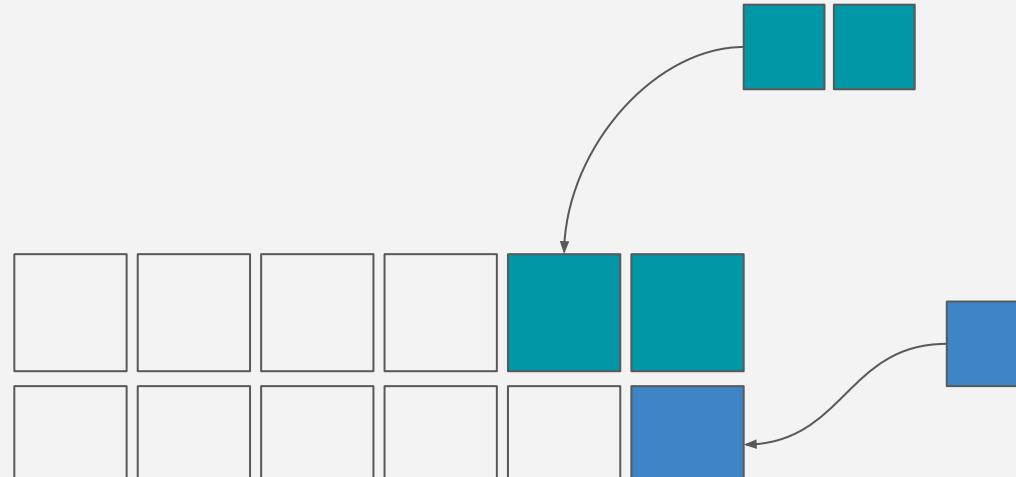
O trabalho do alocador



O trabalho do alocador

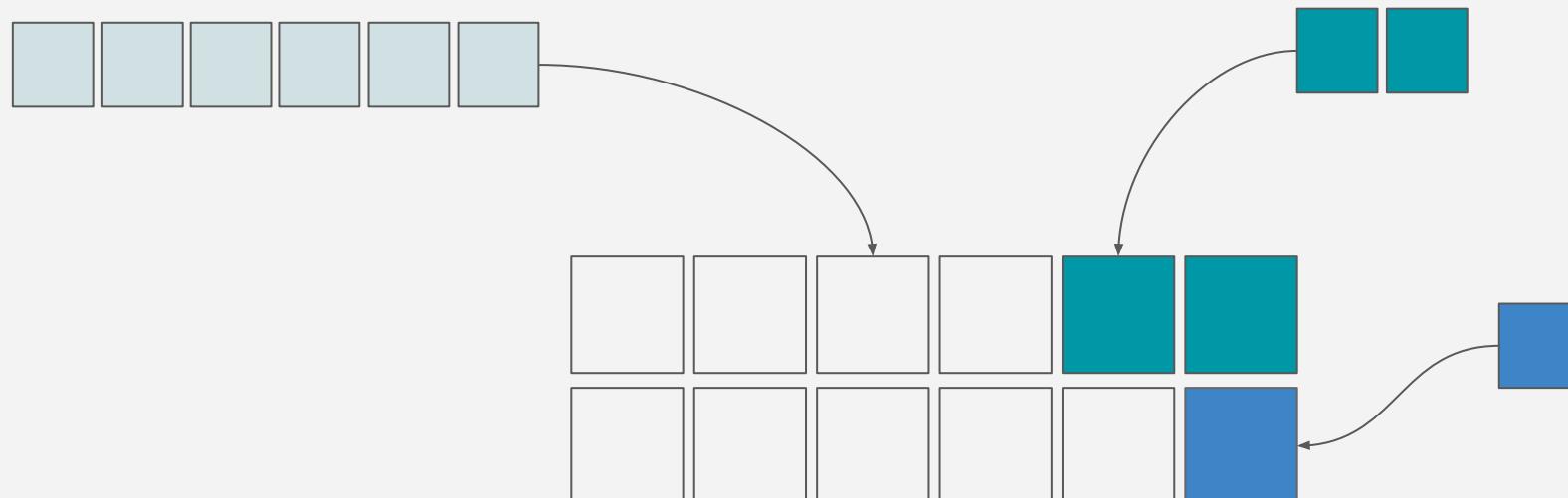


O trabalho do alocador



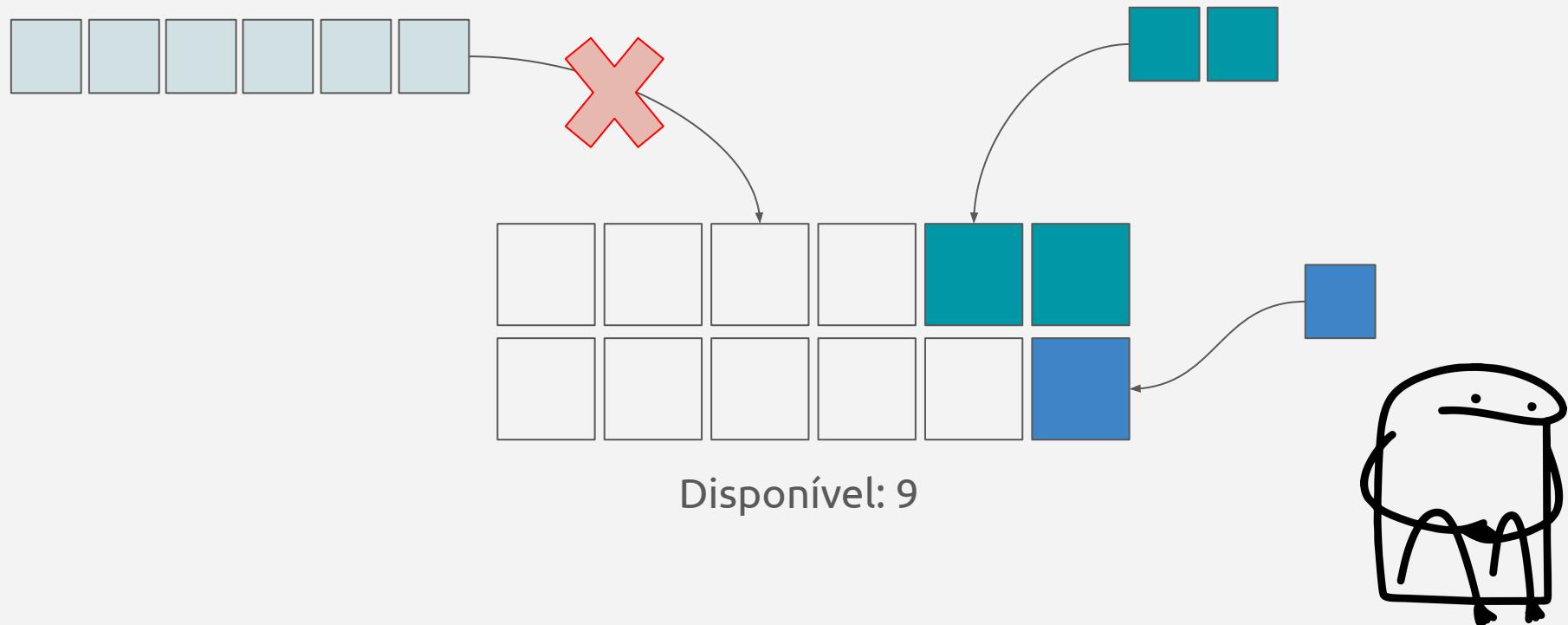
Disponível: 9

O trabalho do alocador

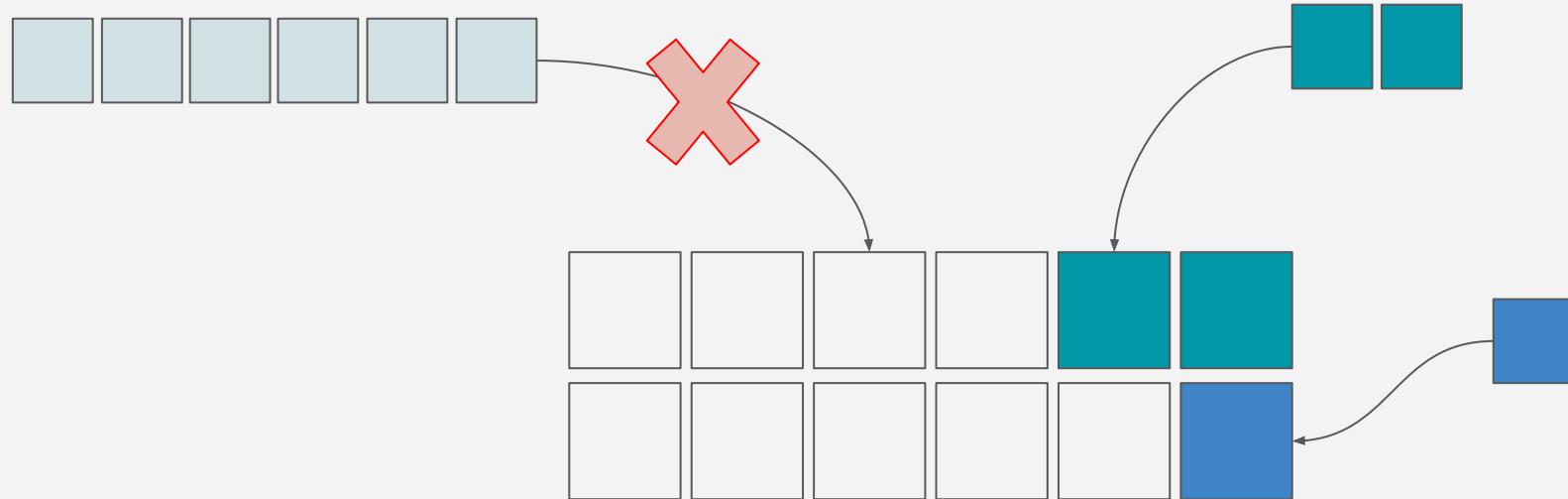


Disponível: 9

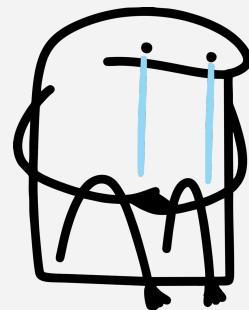
O trabalho do alocador



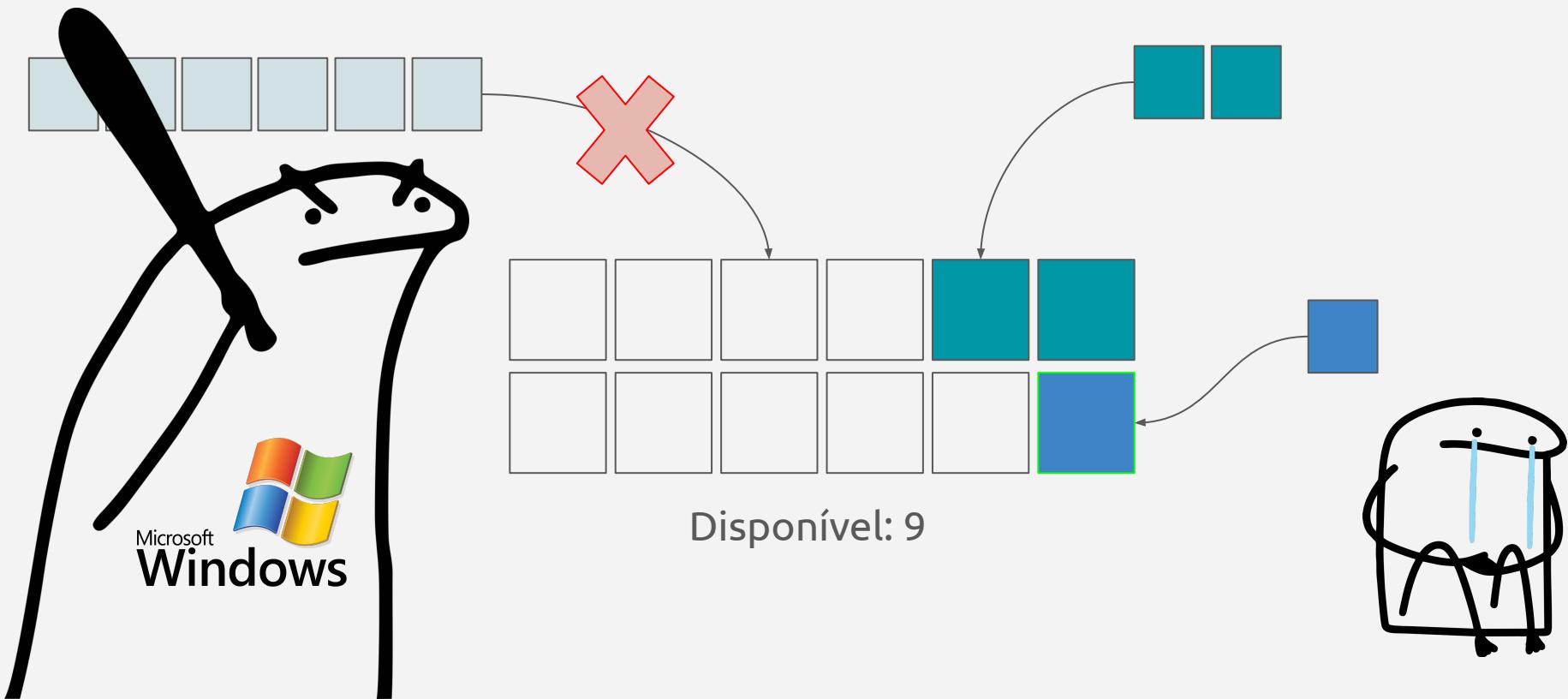
Fragmentação



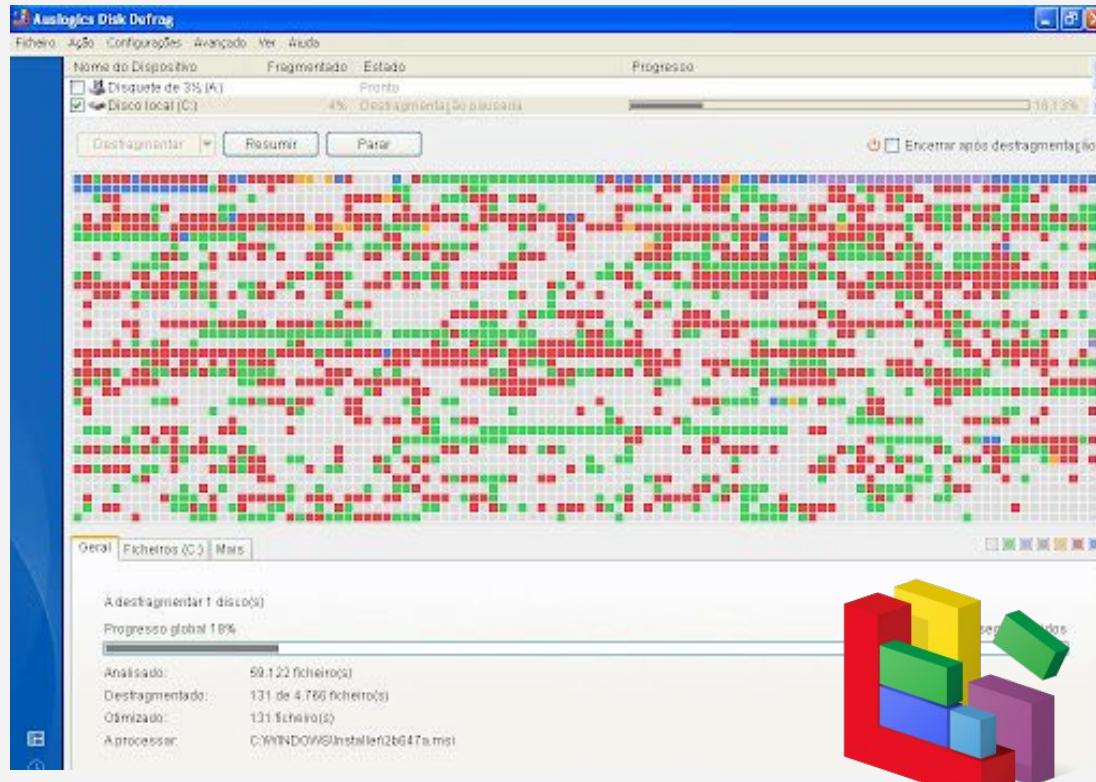
Disponível: 9



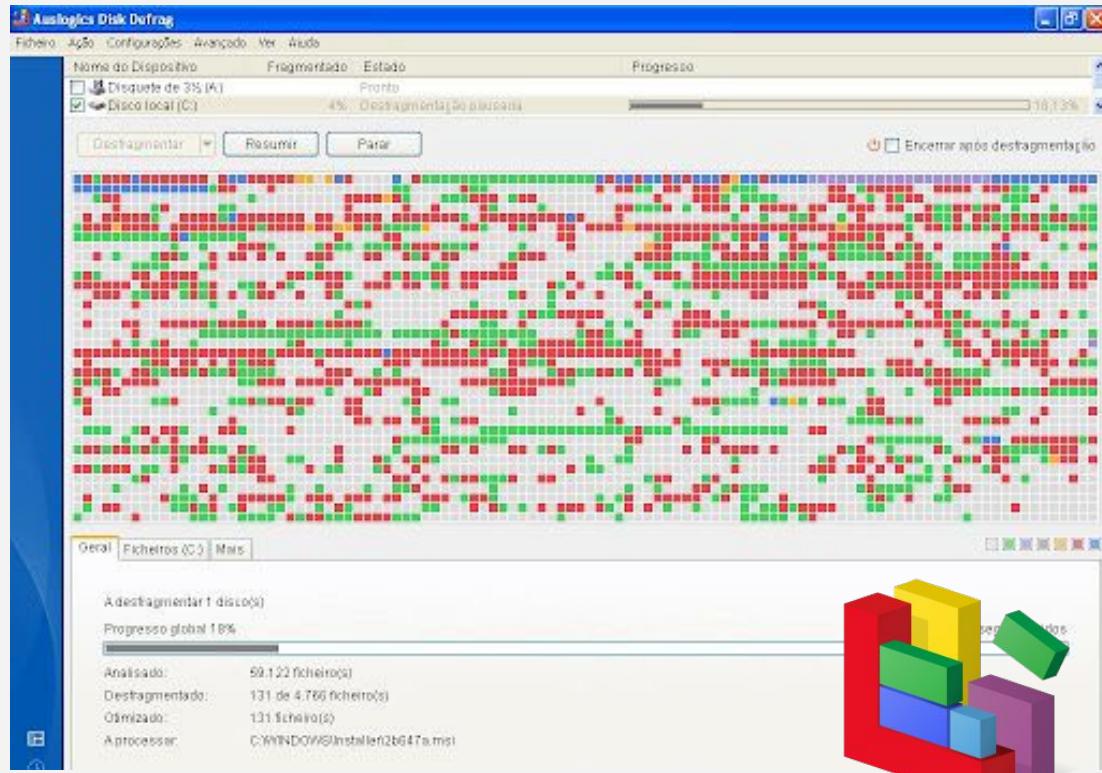
Fragmentação



(des)Fragmentação



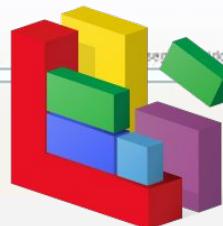
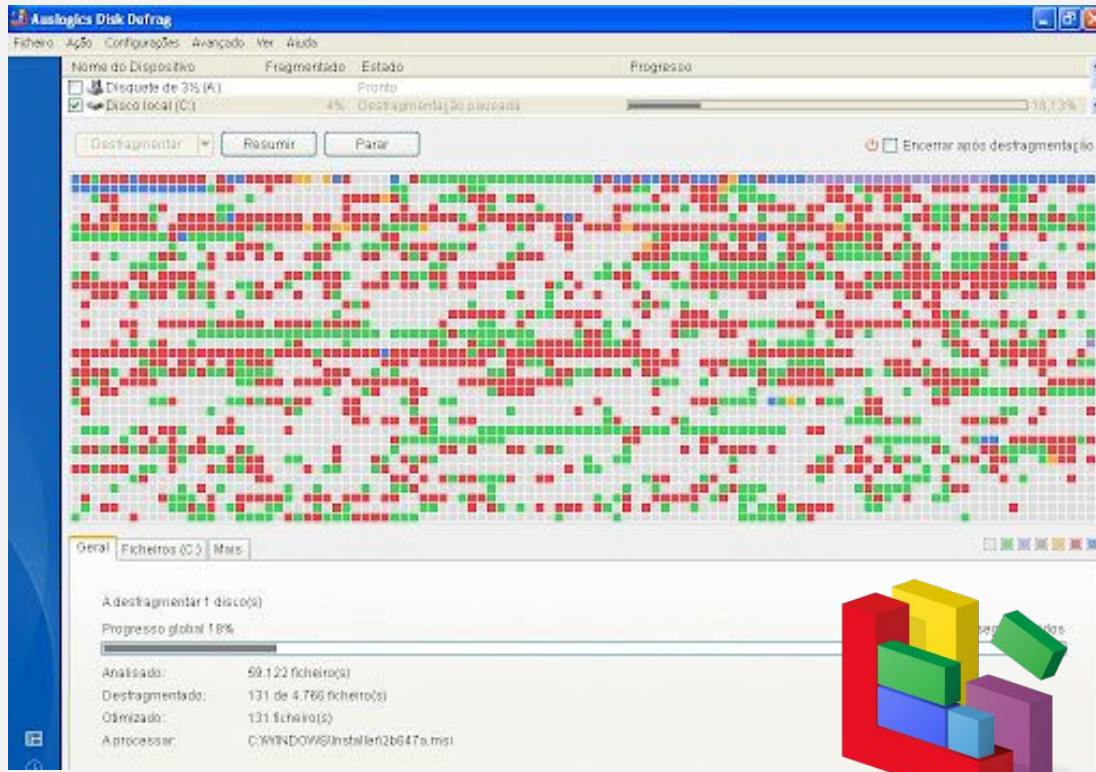
(des)Fragmentação



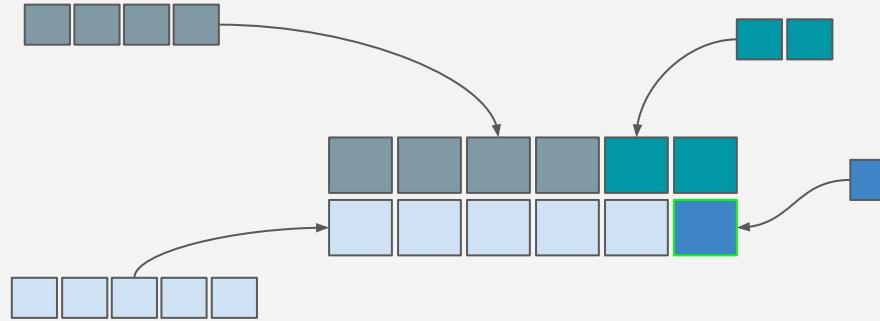
Move os dados para
regiões contíguas de
memória e elimina
espaços vazios.

Compactação

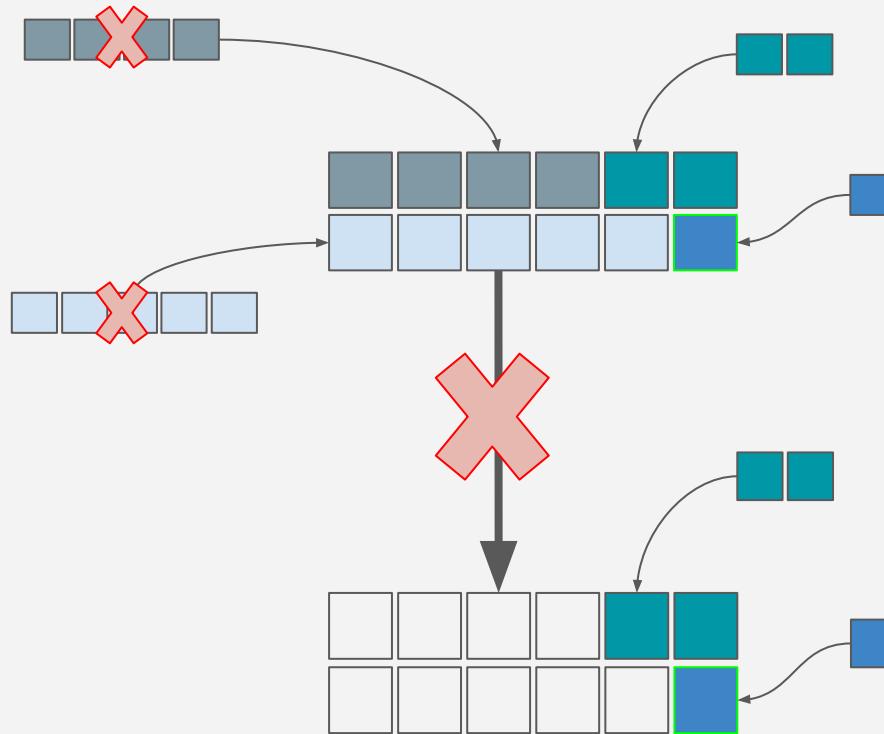
(des)Fragmentação



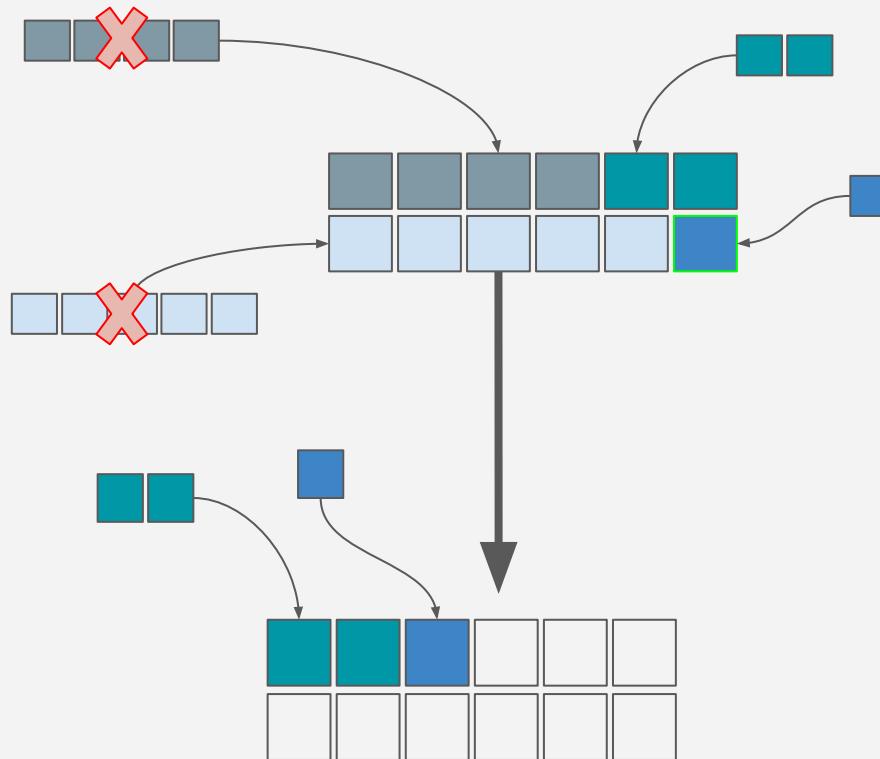
GC Geracional



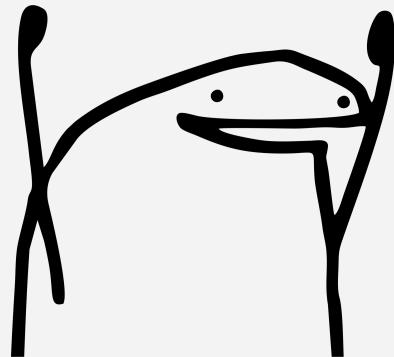
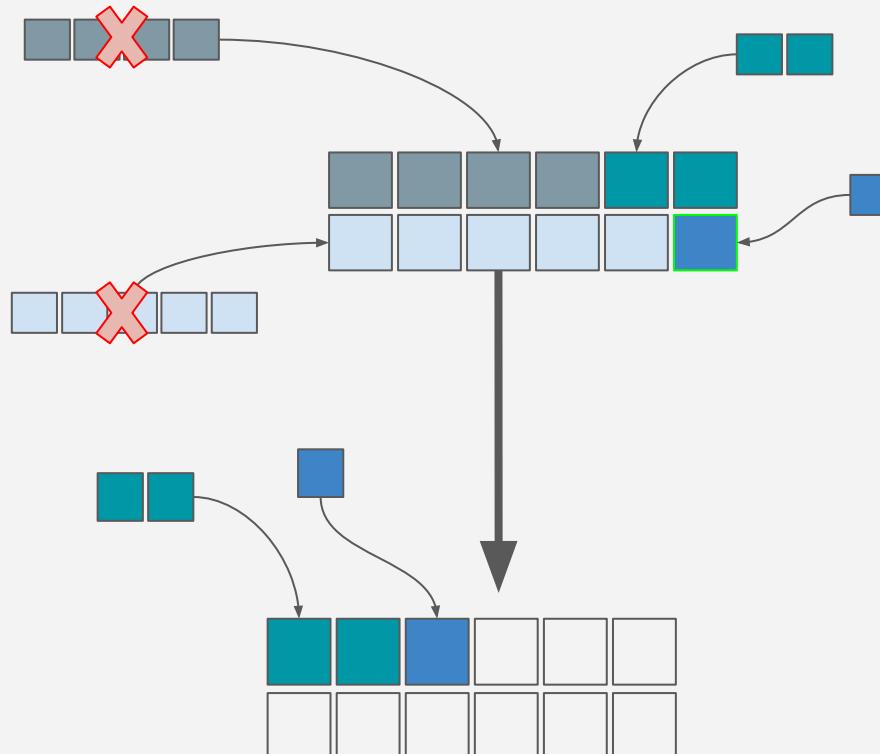
GC Geracional



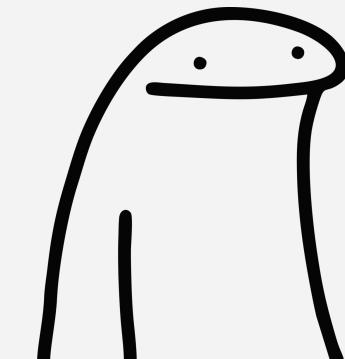
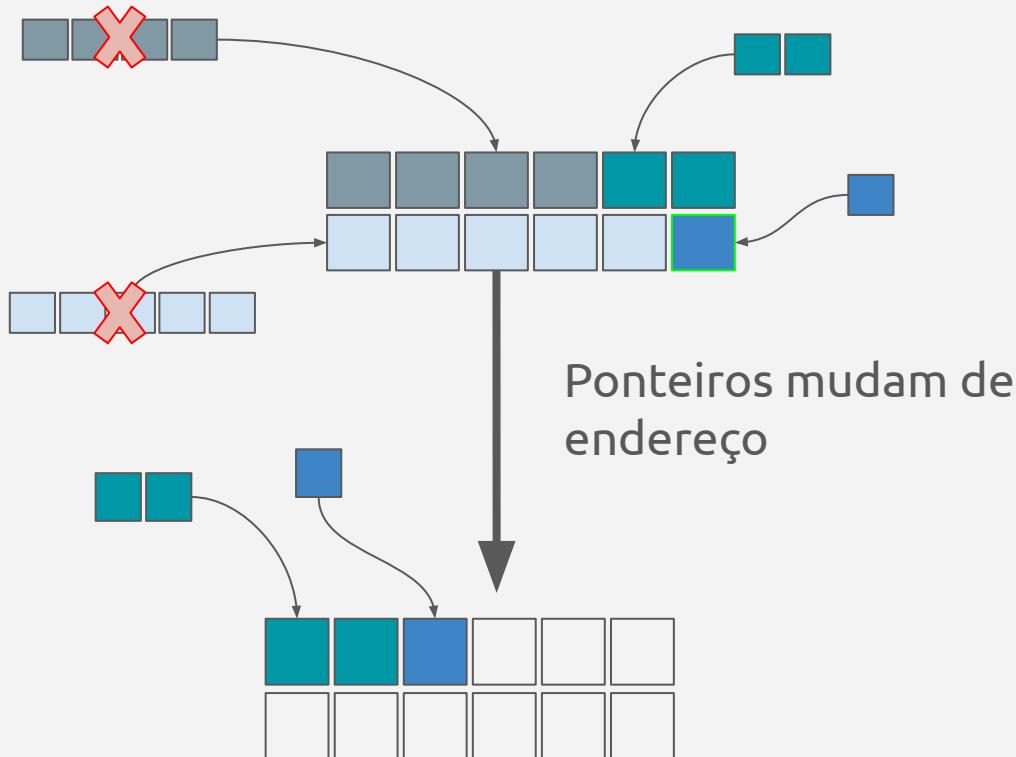
GC Geracional



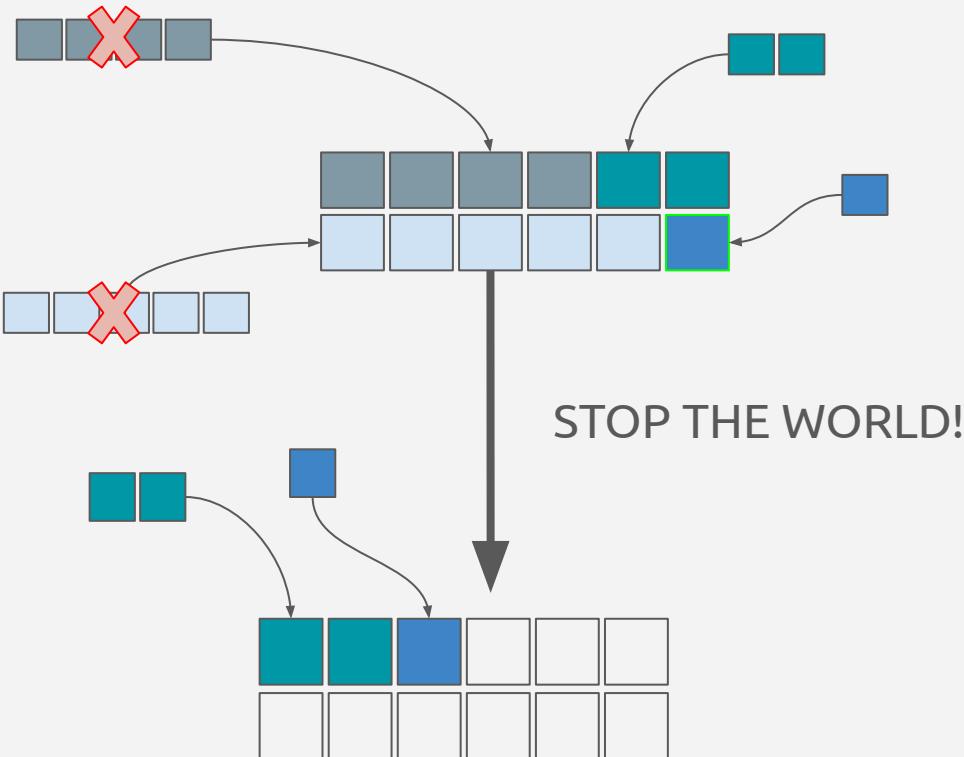
GC Geracional



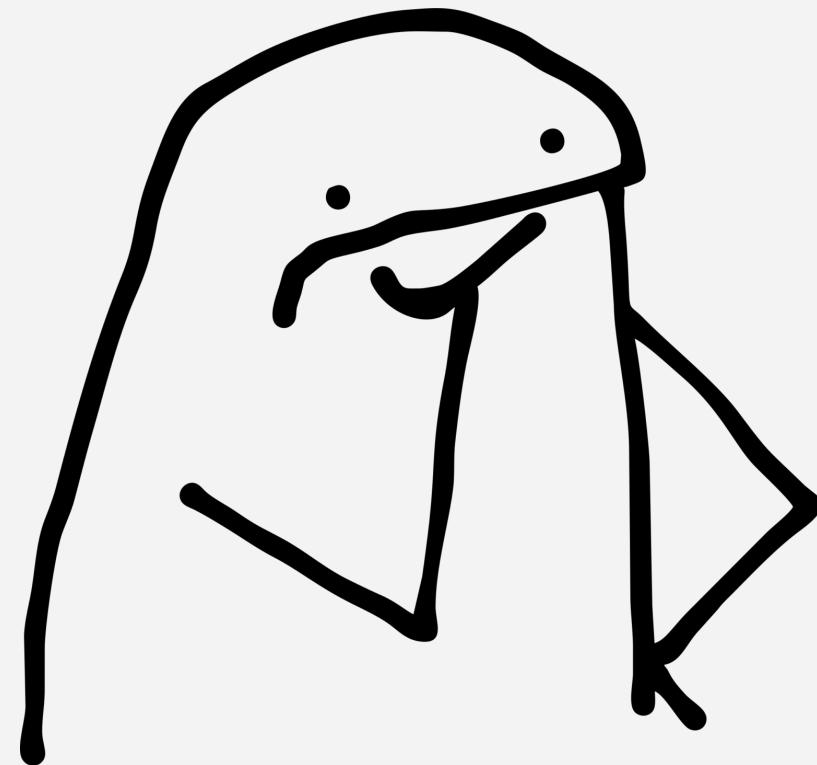
GC Geracional



GC Geracional



Como eliminar a fragmentação?



A fragmentação é inevitável

An Estimate of the Store Size Necessary for Dynamic Storage Allocation

J. M. ROBSON

Oxford University, Oxford, England*

ABSTRACT. Dynamic storage allocation using fixed blocks is usually inefficient in its use of store. The amount of store needed depends on the al for any strategy the amount of store needed is boun arithmically with the size of blocks used. A certain st factor of at most $13/4$. The exact amount of store need sizes 1 and 2 only.

Worst case fragmentation of first fit and best fit storage allocation strategies

J. M. Robson

Department of Computer Studies, University of Lancaster, Bailrigg, Lancaster LA1 4YN

The worst possible storage fragmentation is analysed for two commonly used allocation strategies. In the case of the first fit system, fragmentation is not much worse than is inevitable but for the best fit system, it is almost as bad as it could be for any system.

(Received January 1975)

A fragmentação é inevitável

An Estimate of the Store Size Necessary for Dynamic Storage Allocation

J. M. ROBSON

Oxford University, Oxford, England*

ABSTRACT. Dynamic storage allocation using fixed blocks is usually inefficient in its use of store. The amount of store needed depends on the al for any strategy the amount of store needed is boun arithmically with the size of blocks used. A certain sti factor of at most 13/4. The exact amount of store need sizes 1 and 2 only.

Realizar uma alocação ótima de memória é um problema NP-hard.

Worst case fragmentation of first fit and best fit storage allocation strategies

J. M. Robson

Department of Computer Studies, University of Lancaster, Bailrigg, Lancaster LA1 4YN

The worst possible storage fragmentation is analysed for two commonly used allocation strategies. In the case of the first fit system, fragmentation is not much worse than is inevitable but for the best fit system, it is almost as bad as it could be for any system.

(Received January 1975)

Heurísticas

- Um palpite fundamentado

Heurísticas

- Um palpite fundamentado
- Troca a perfeição ou o algoritmos ótimo por velocidade

Heurísticas relacionadas a alocação de memória

- 1. Objetos pequenos são alocados mais frequentemente do que objetos grandes.**

Heurísticas relacionadas a alocação de memória

1. **Objetos pequenos são alocados mais frequentemente** do que objetos grandes.
2. O tempo de vida de um objeto geralmente é **proporcional ao tamanho** do mesmo.
 - a. Objetos menores tem ciclos de vida menores.
 - b. Objetos maiores tendem a permanecer mais tempo na memória.

Heurísticas relacionadas a alocação de memória

1. **Objetos pequenos são alocados mais frequentemente** do que objetos grandes.
2. O tempo de vida de um objeto geralmente é **proporcional ao tamanho** do mesmo.
 - a. Objetos menores tem ciclos de vida menores.
 - b. Objetos maiores tendem a permanecer mais tempo na memória.
3. Objetos do mesmo tamanho tendem a ser **alocados e desalocados juntos**.

Heurísticas relacionadas a alocação de memória

1. **Objetos pequenos são alocados mais frequentemente** do que objetos grandes.
2. O tempo de vida de um objeto geralmente é **proporcional ao tamanho** do mesmo.
 - a. Objetos menores tem ciclos de vida menores.
 - b. Objetos maiores tendem a permanecer mais tempo na memória.
3. Objetos do mesmo tamanho tendem a ser **alocados e desalocados juntos**.



Alocações pequenas são fundamentalmente diferentes de alocações grandes.

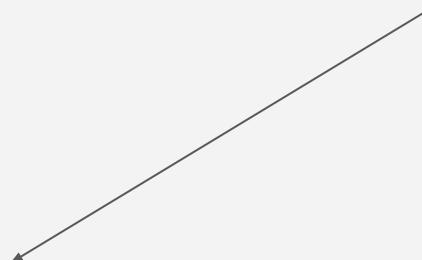
Heurísticas relacionadas a alocação de memória

1. **Objetos pequenos são alocados mais frequentemente do que objetos grandes.**
2. O tempo de vida de um objeto geralmente é **proporcional ao tamanho** do mesmo.
 - a. Objetos menores tem ciclos de vida menores.
 - b. Objetos maiores tendem a permanecer mais tempo na memória.
3. Objetos do mesmo tamanho tendem a ser **alocados e desalocados juntos**.

Alocações pequenas são fundamentalmente diferentes de alocações grandes.

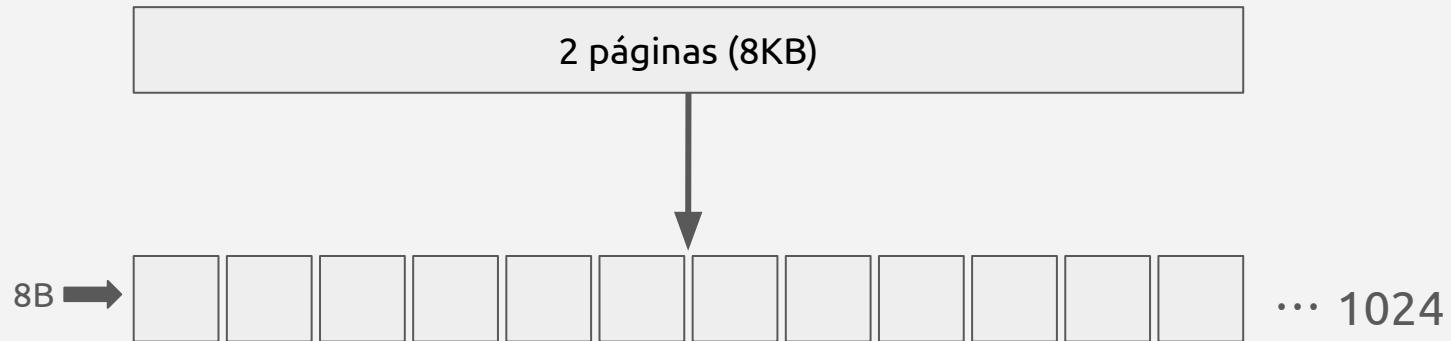
Merecem diferentes estratégias

Estratégia para alocações pequenas

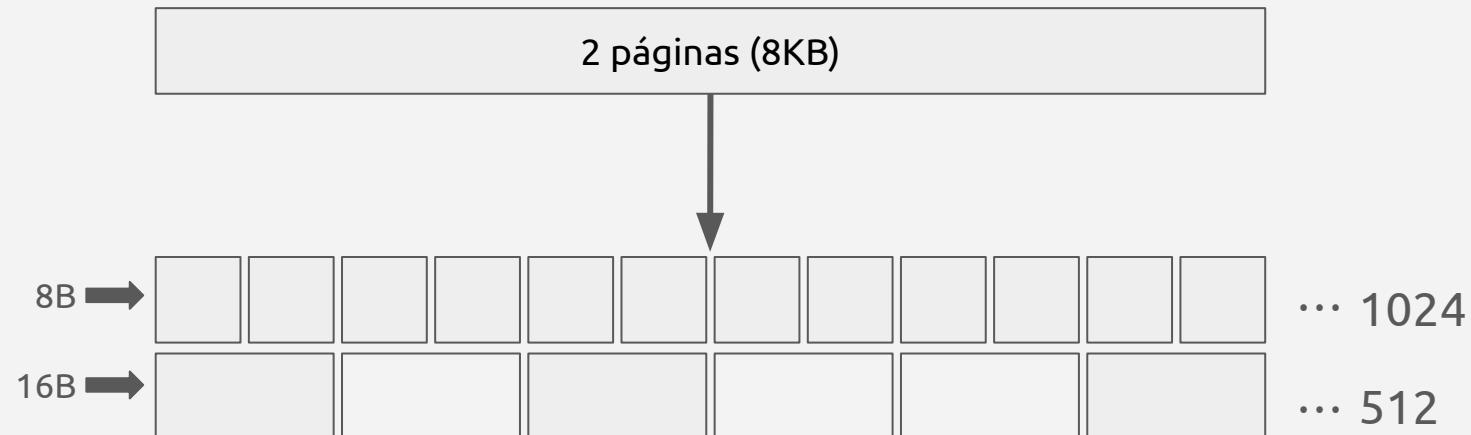


De 1B à 32KB

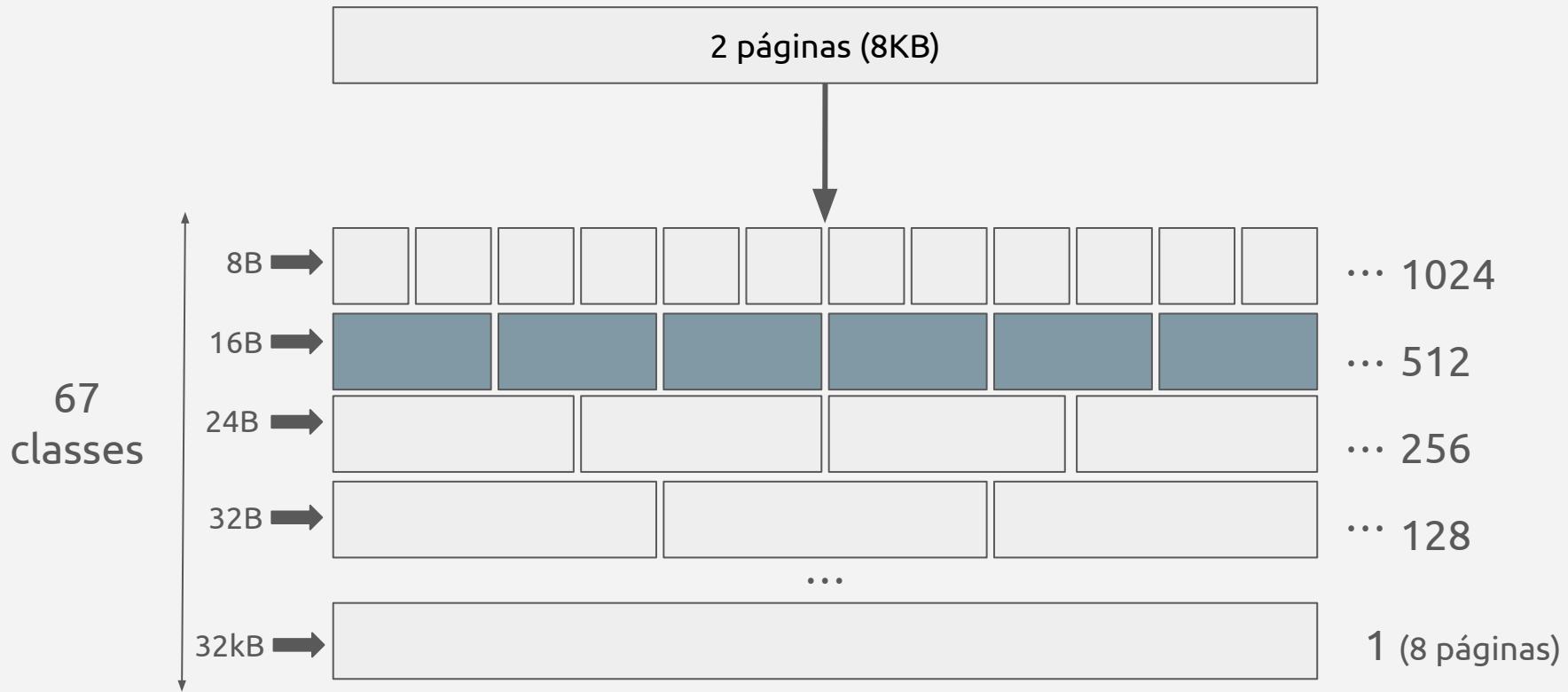
Estratégia para alocações pequenas



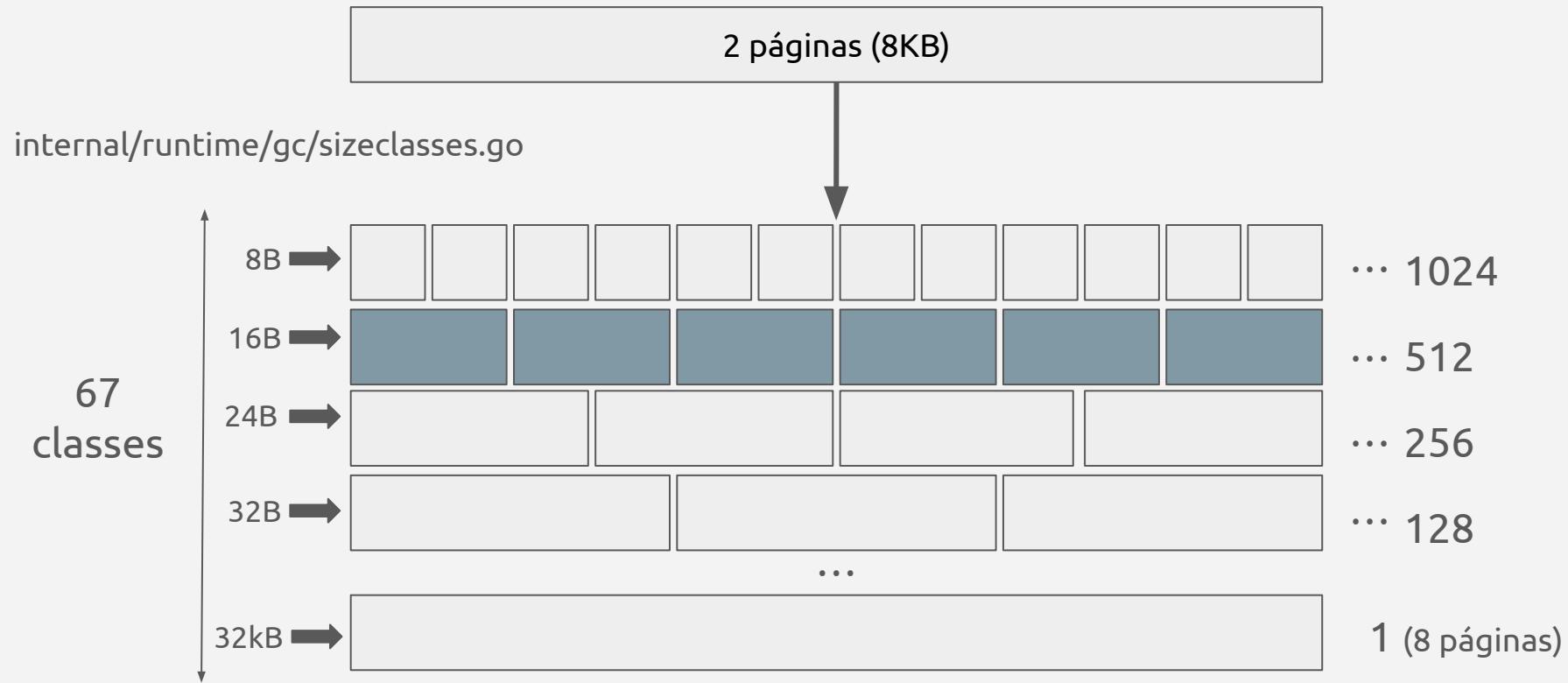
Estratégia para alocações pequenas



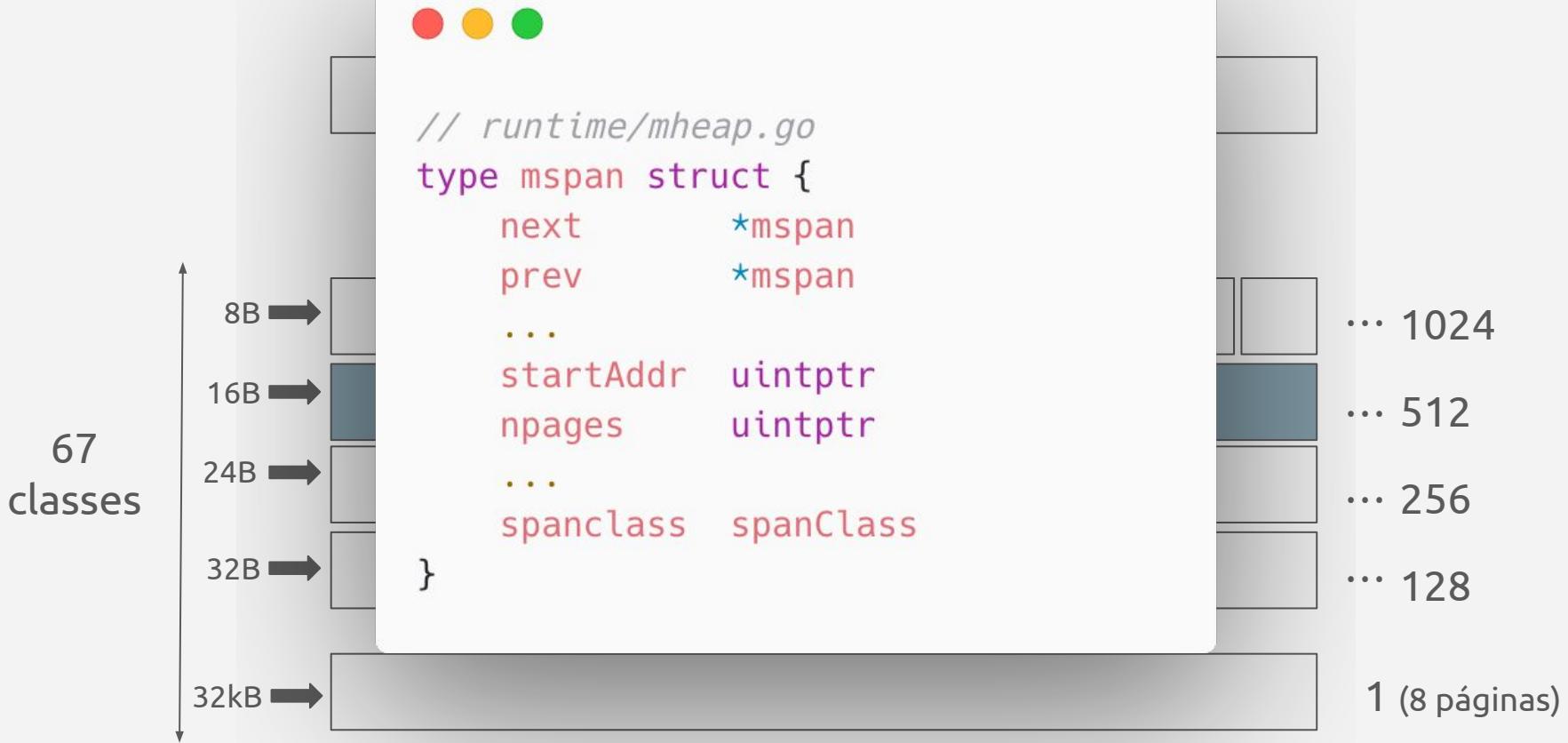
mspan



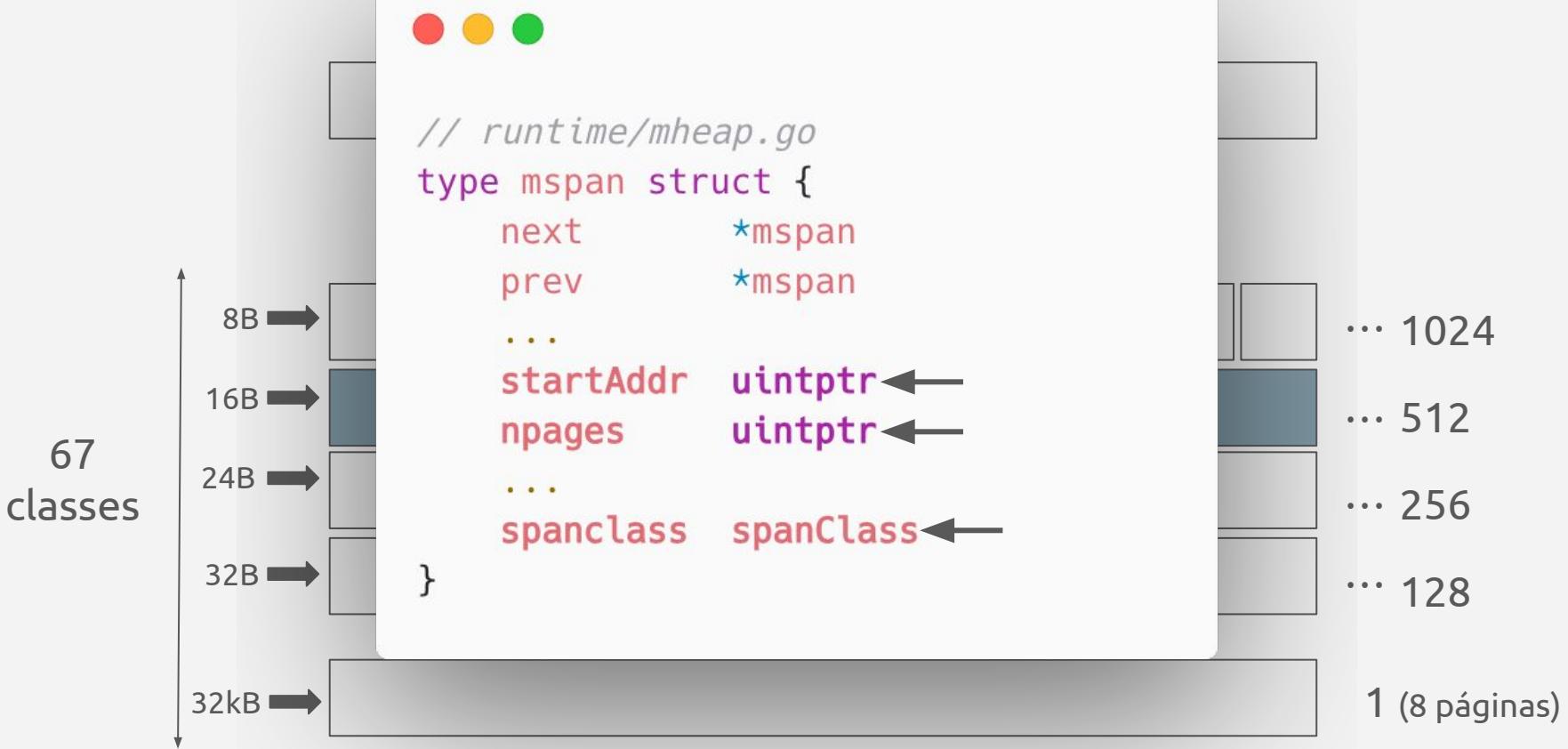
mspan



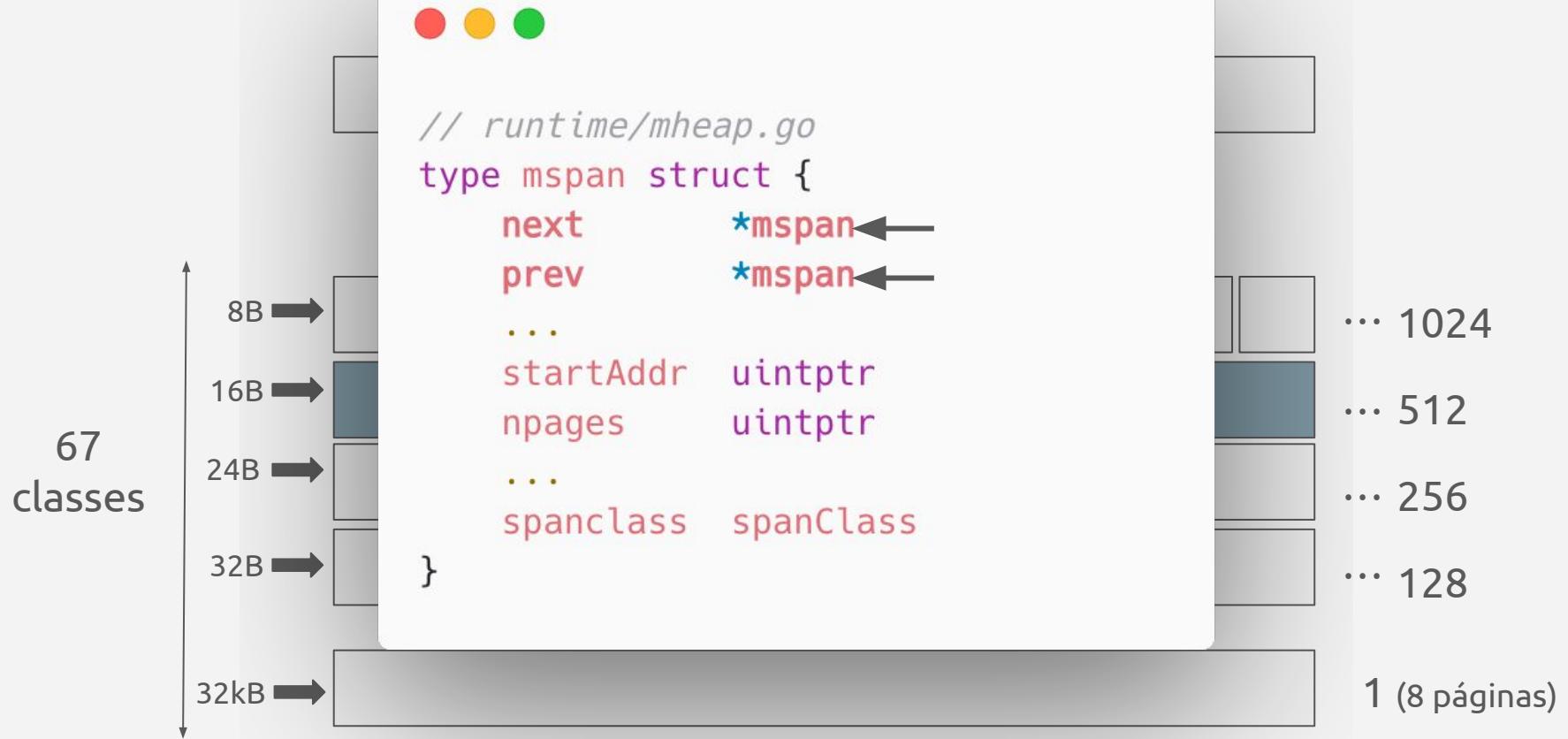
mspan



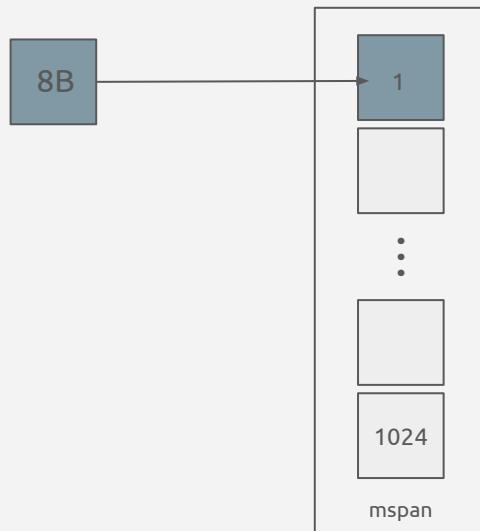
mspan



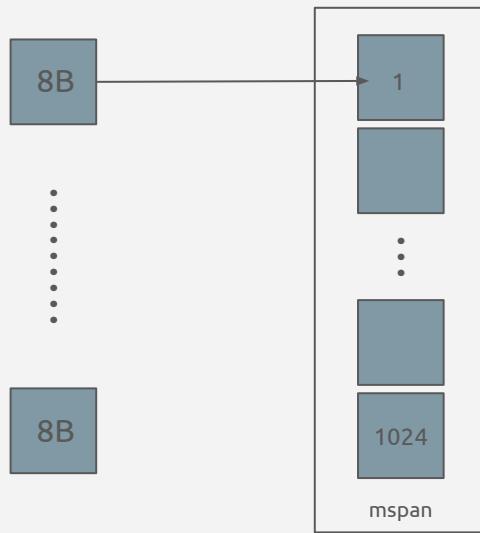
mspan



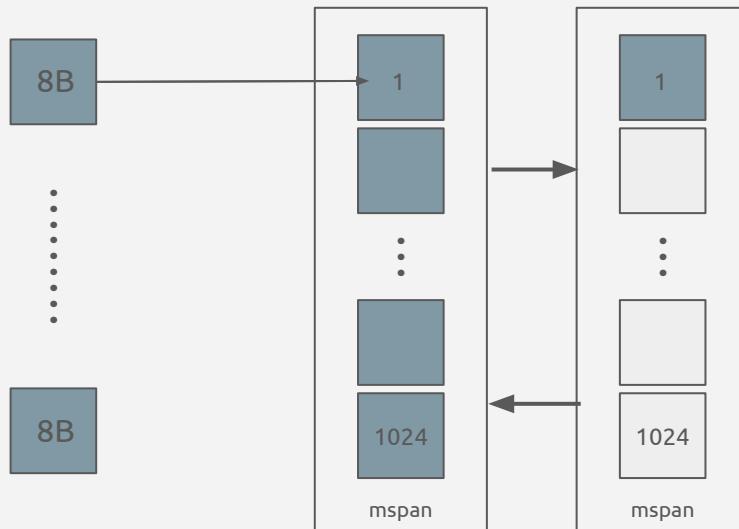
Para cada classe, uma lista de mspans



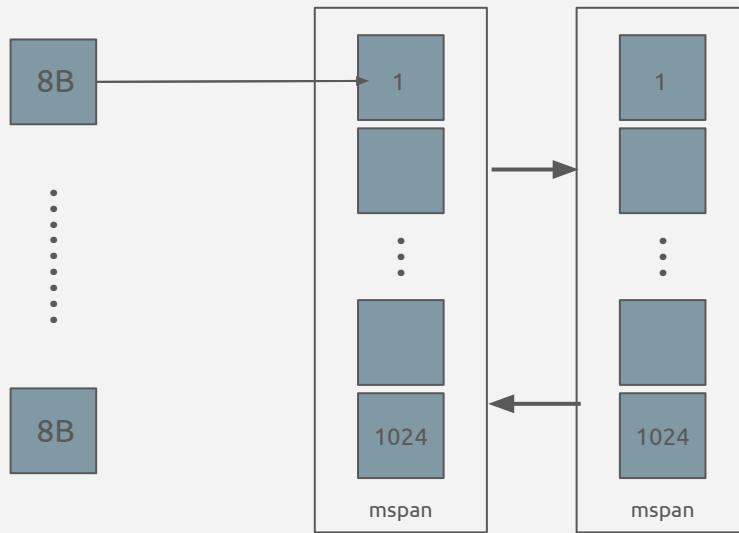
Para cada classe, uma lista de mspans



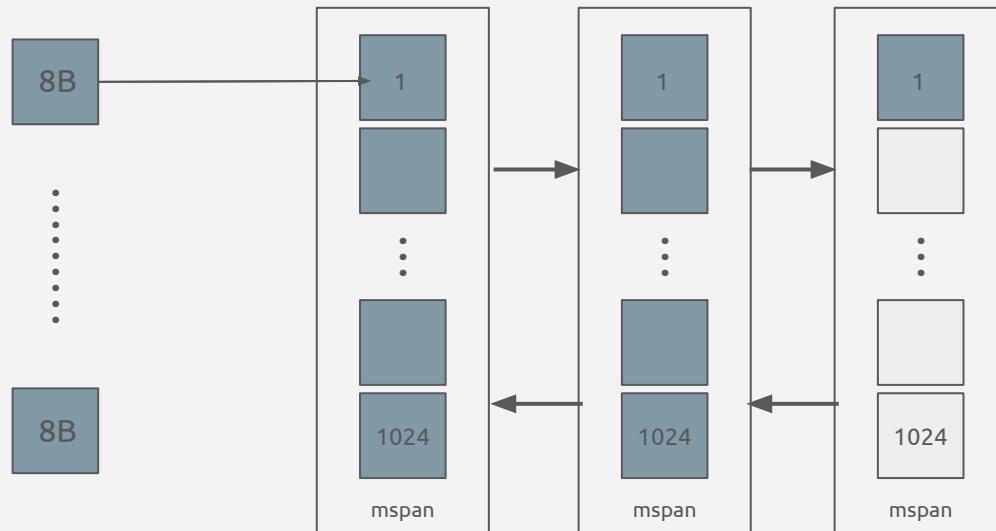
Para cada classe, uma lista de mspans



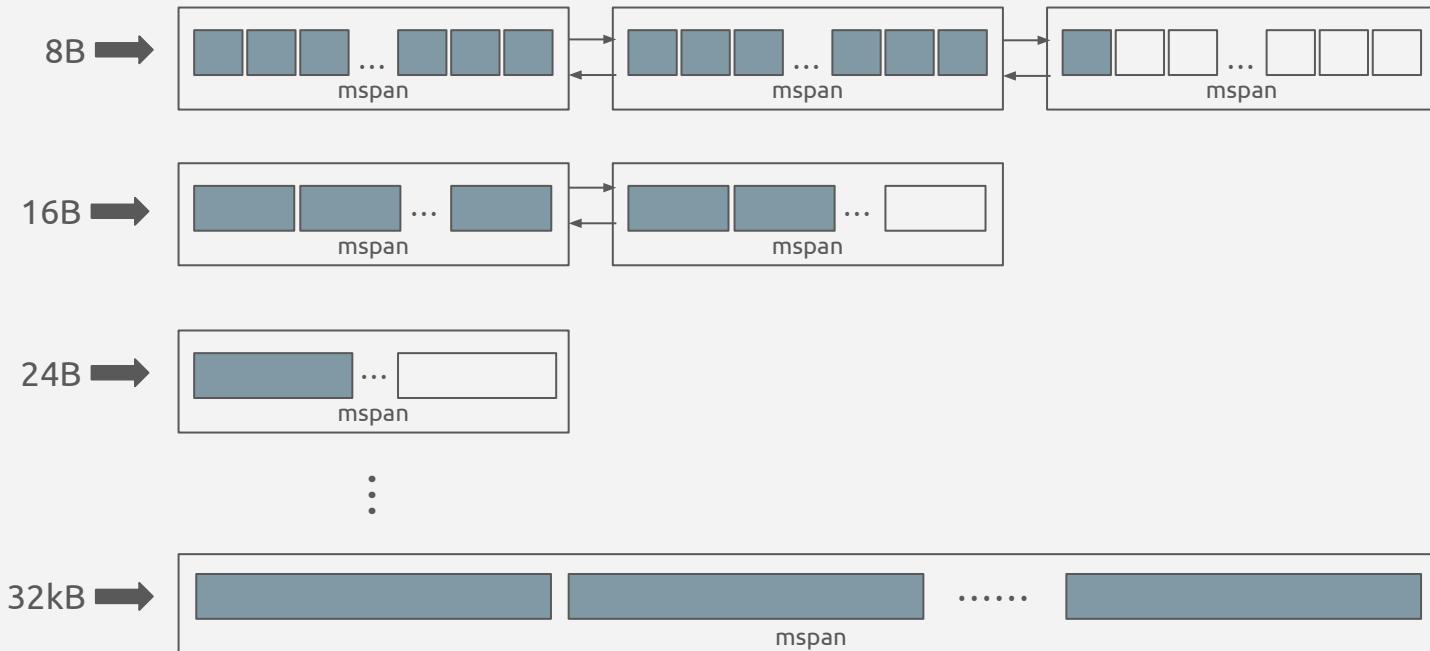
Para cada classe, uma lista de mspans



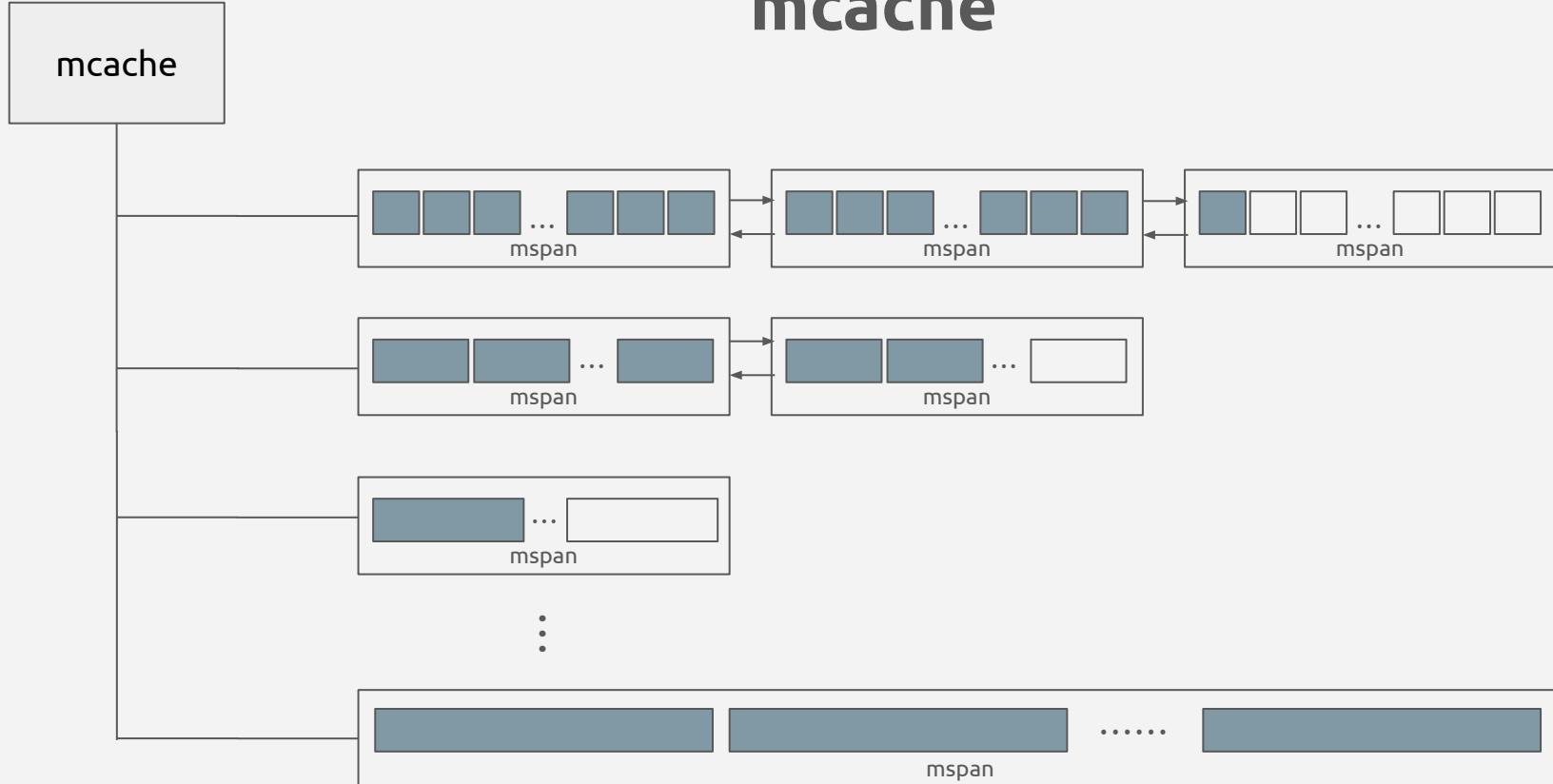
Para cada classe, uma lista de mspans



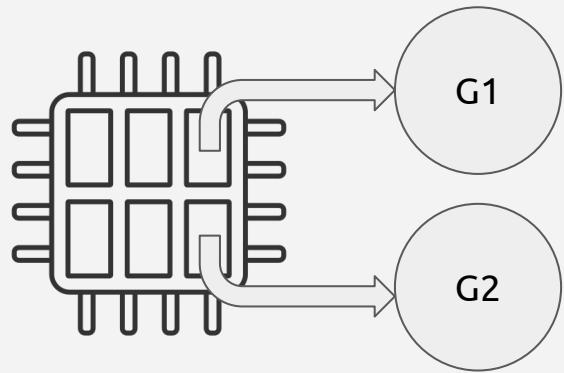
Para cada classe, uma lista de mspans



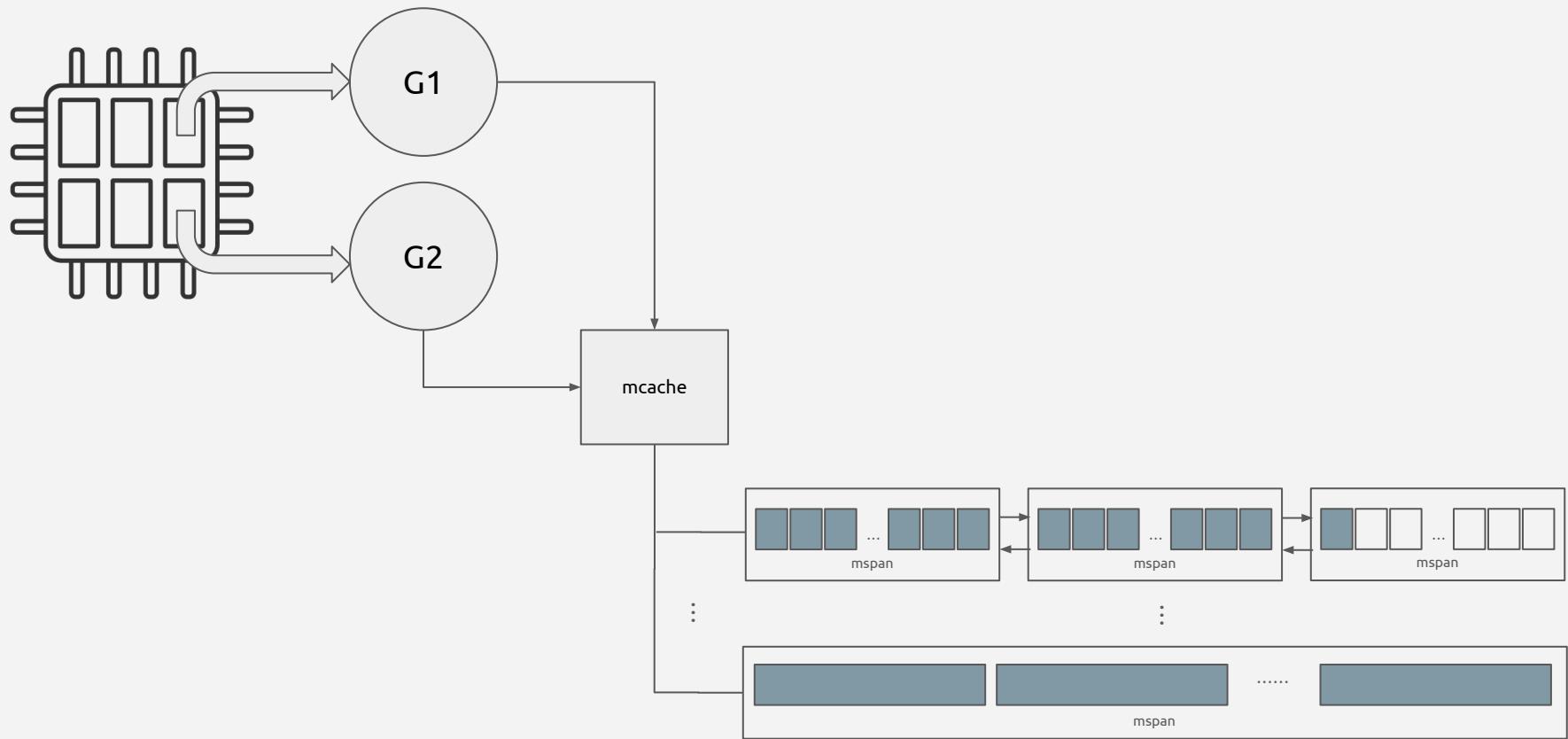
mcache



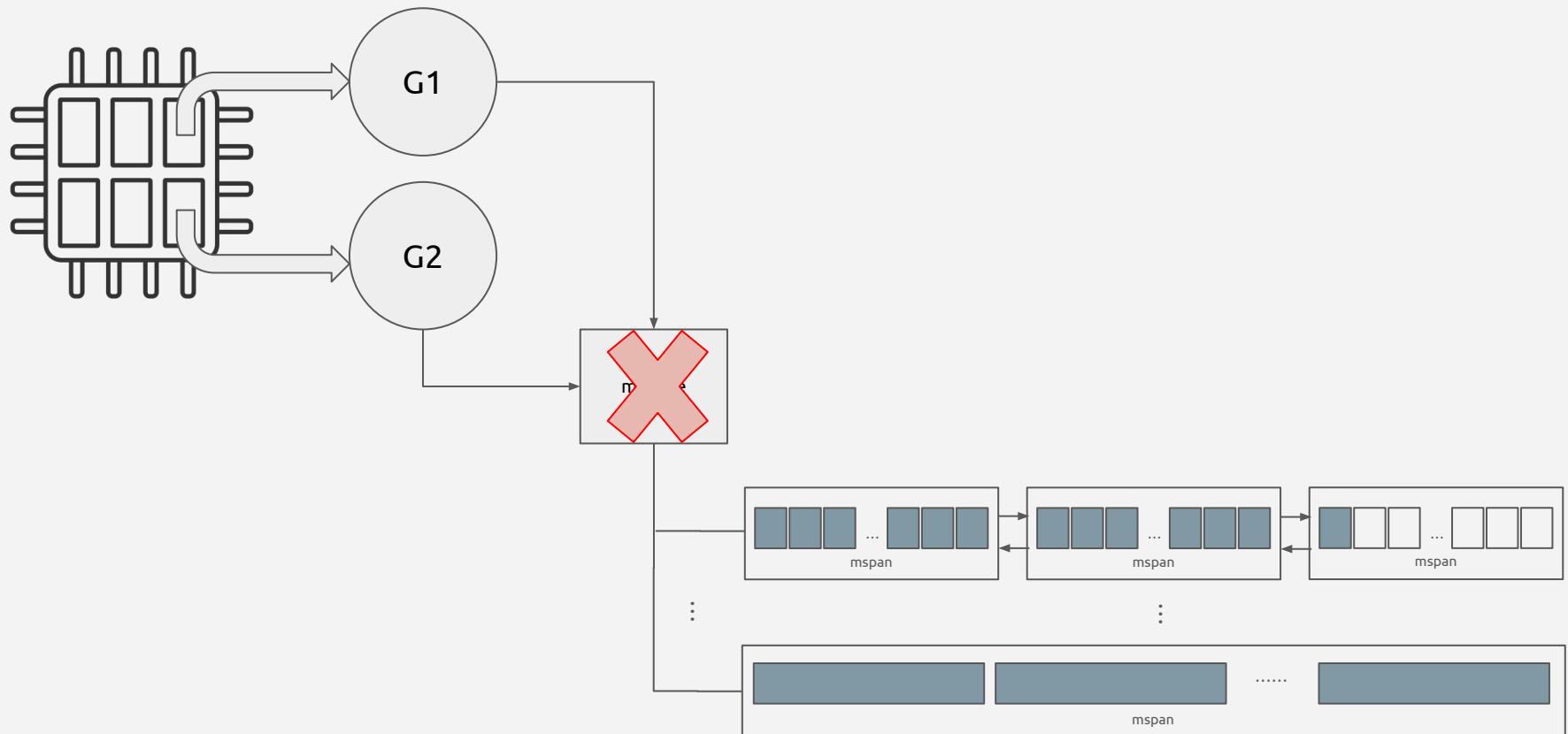
Concorrência



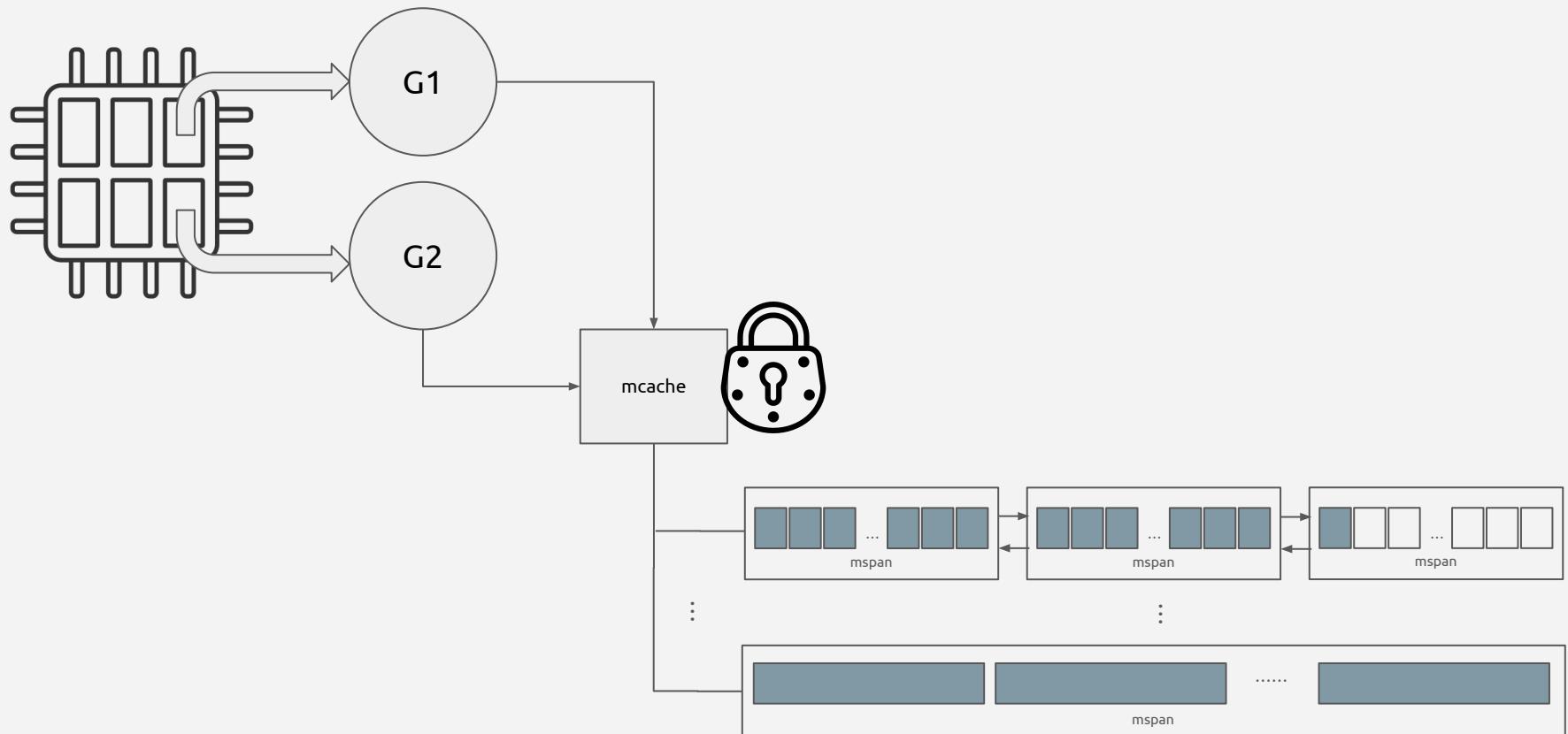
Concorrência



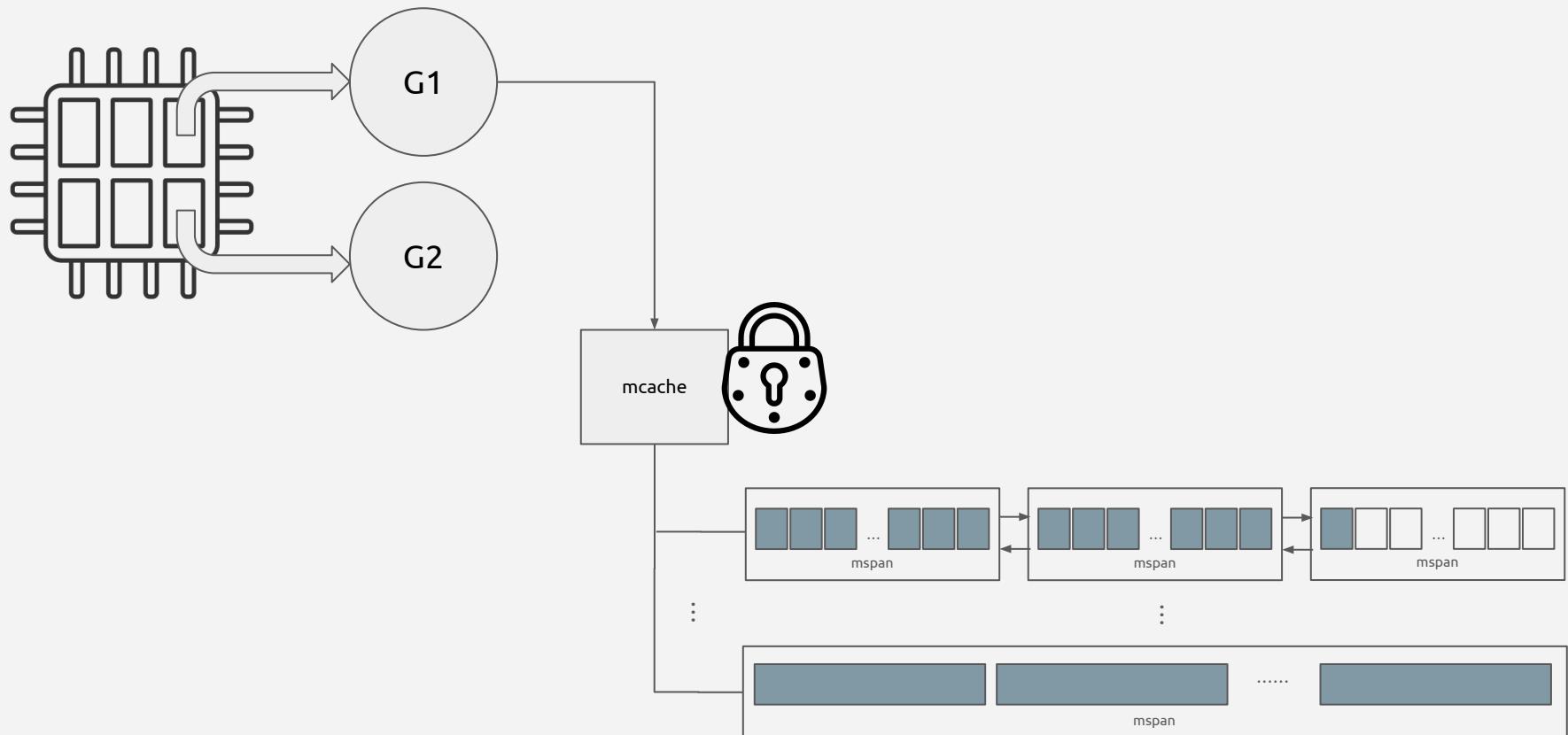
Concorrência



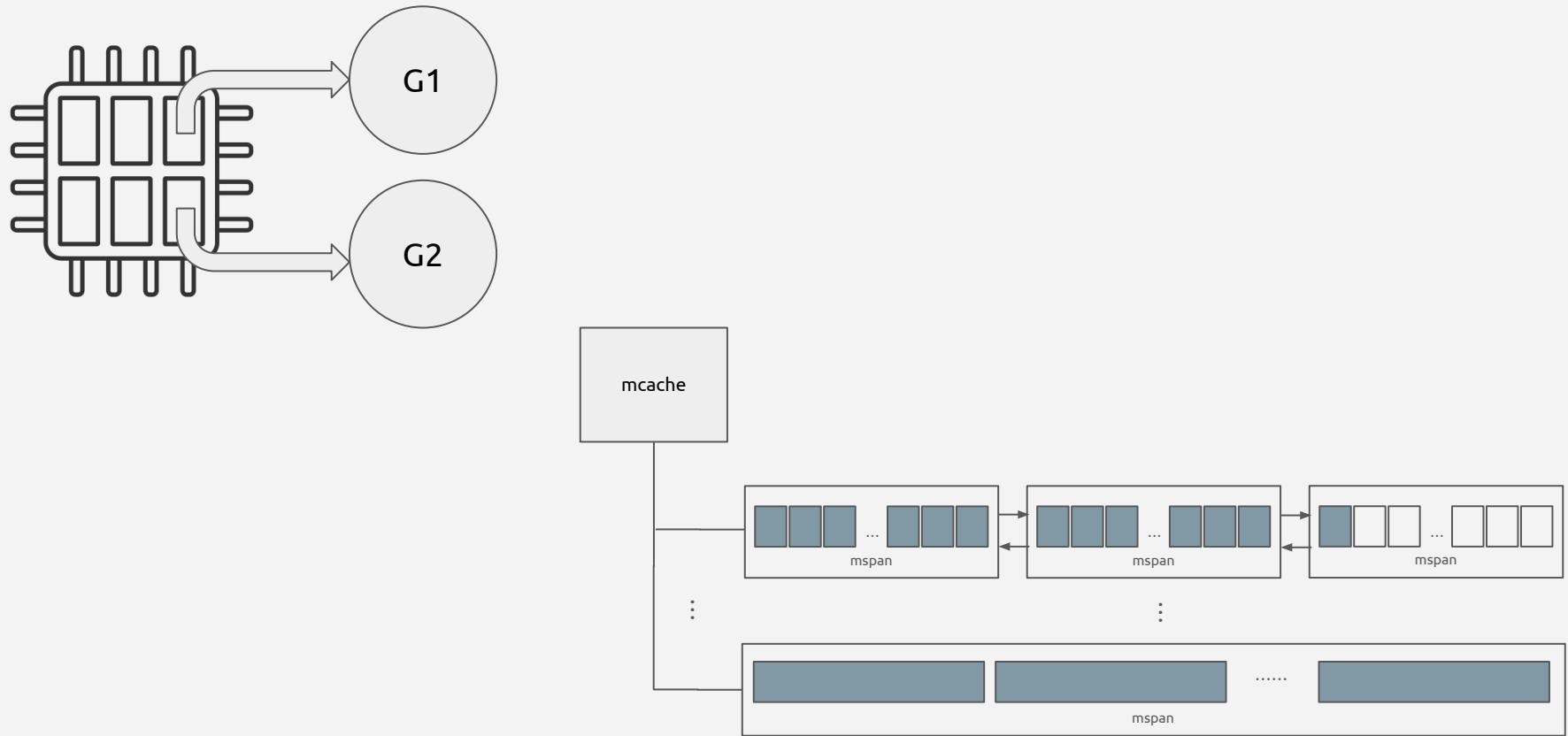
Concorrência



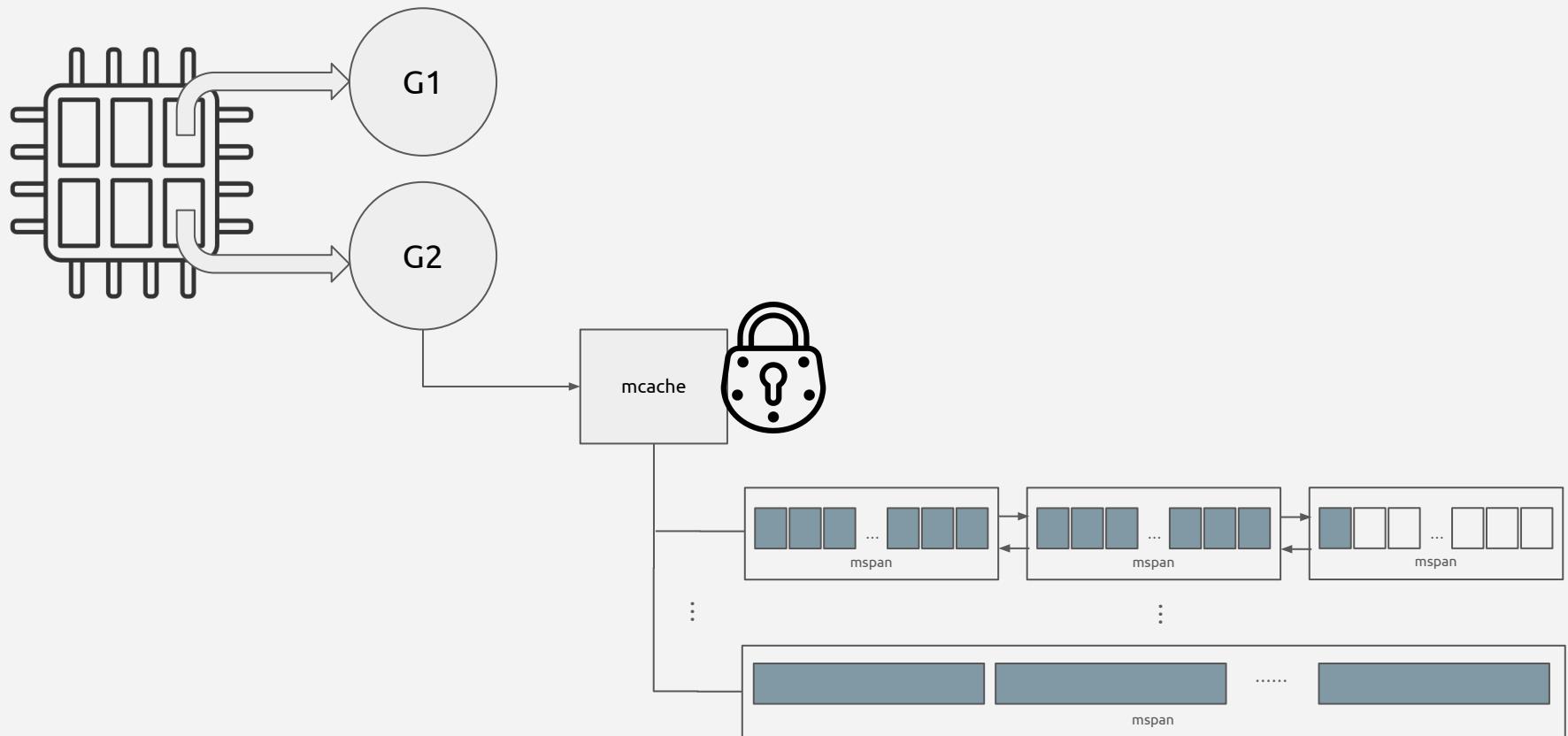
Concorrência



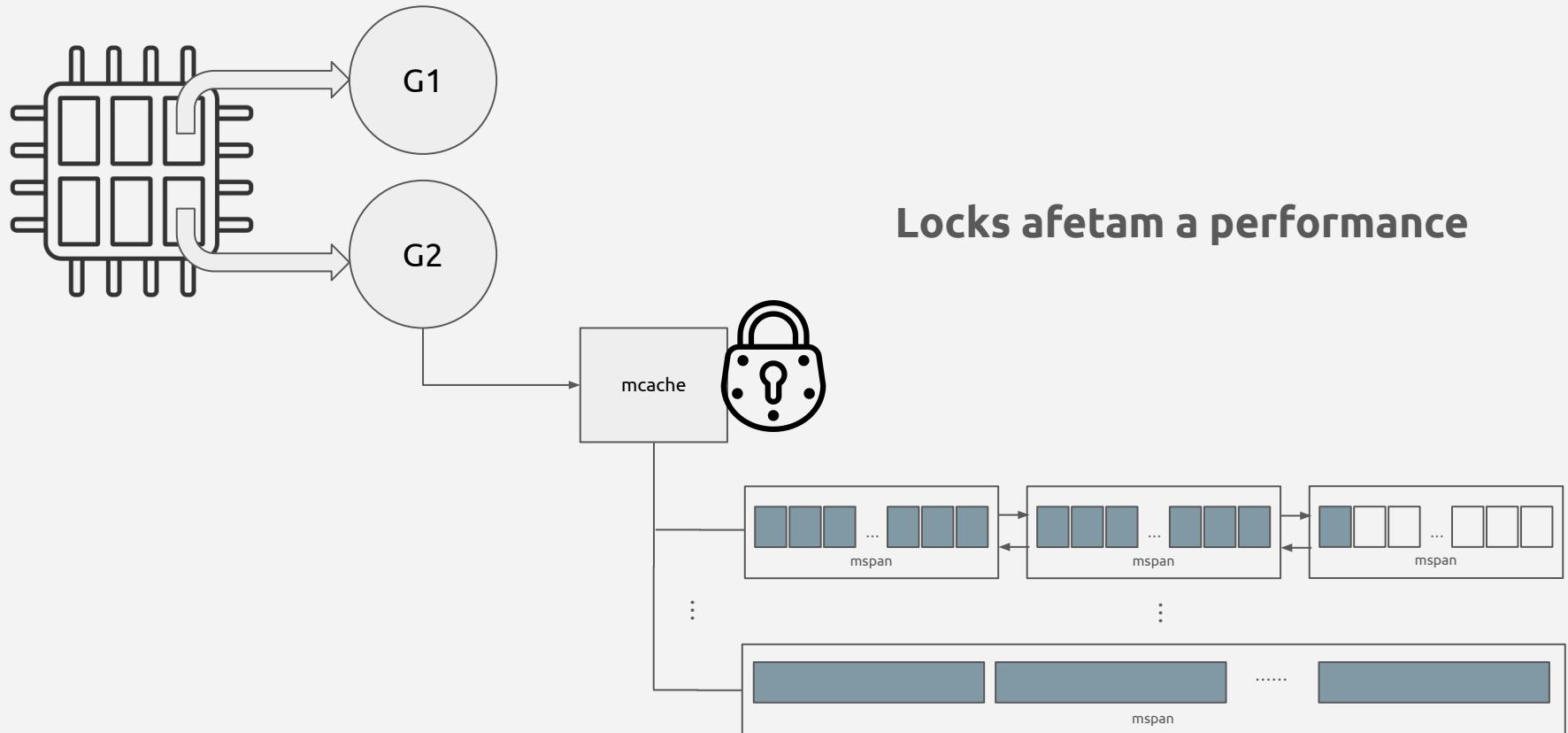
Concorrência



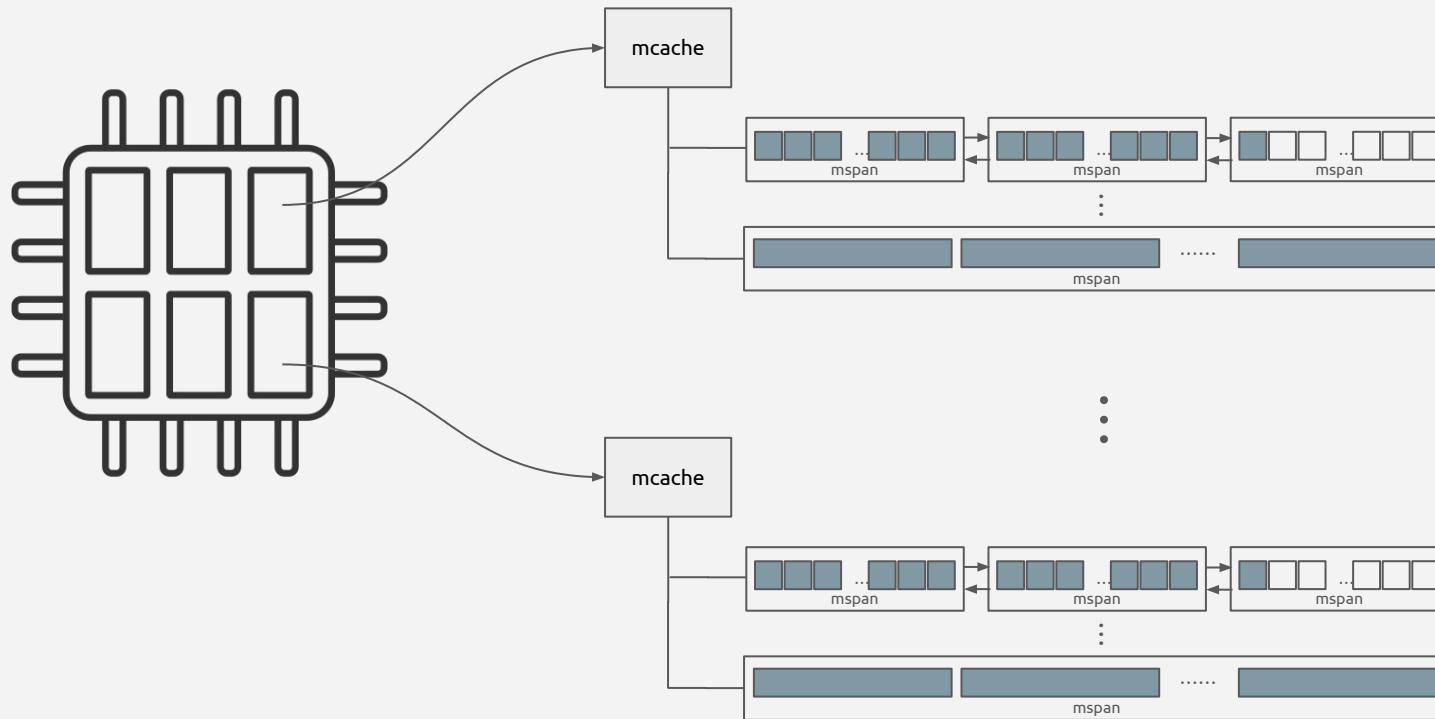
Concorrência



Concorrência

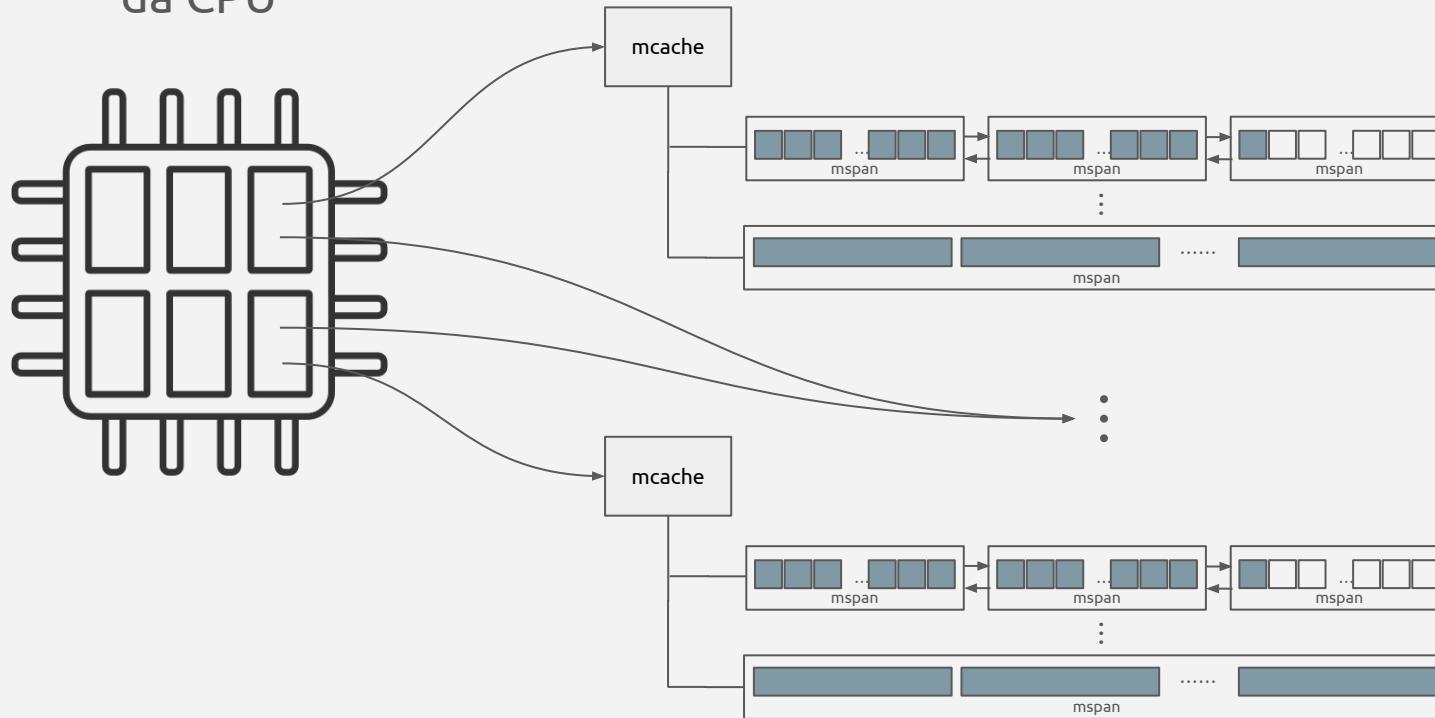


mcache



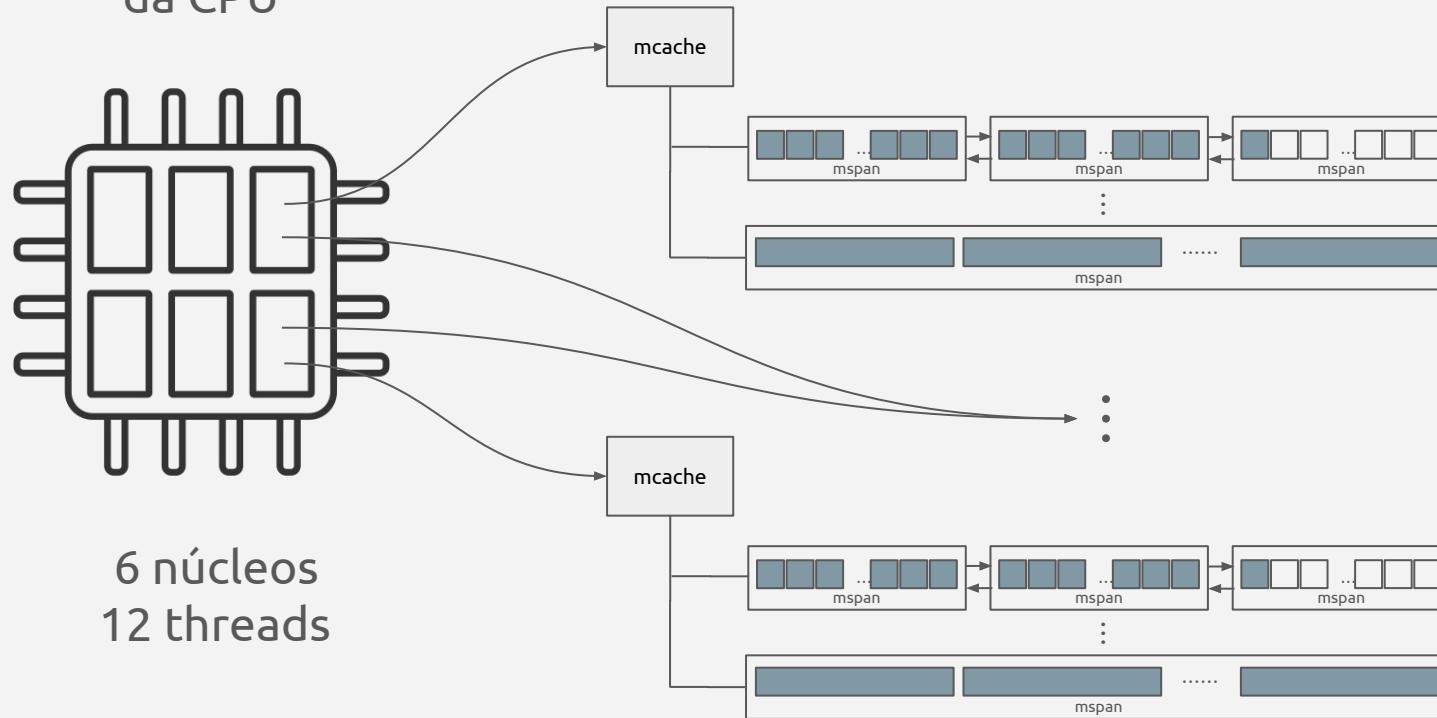
mcache

Thread de execução
da CPU



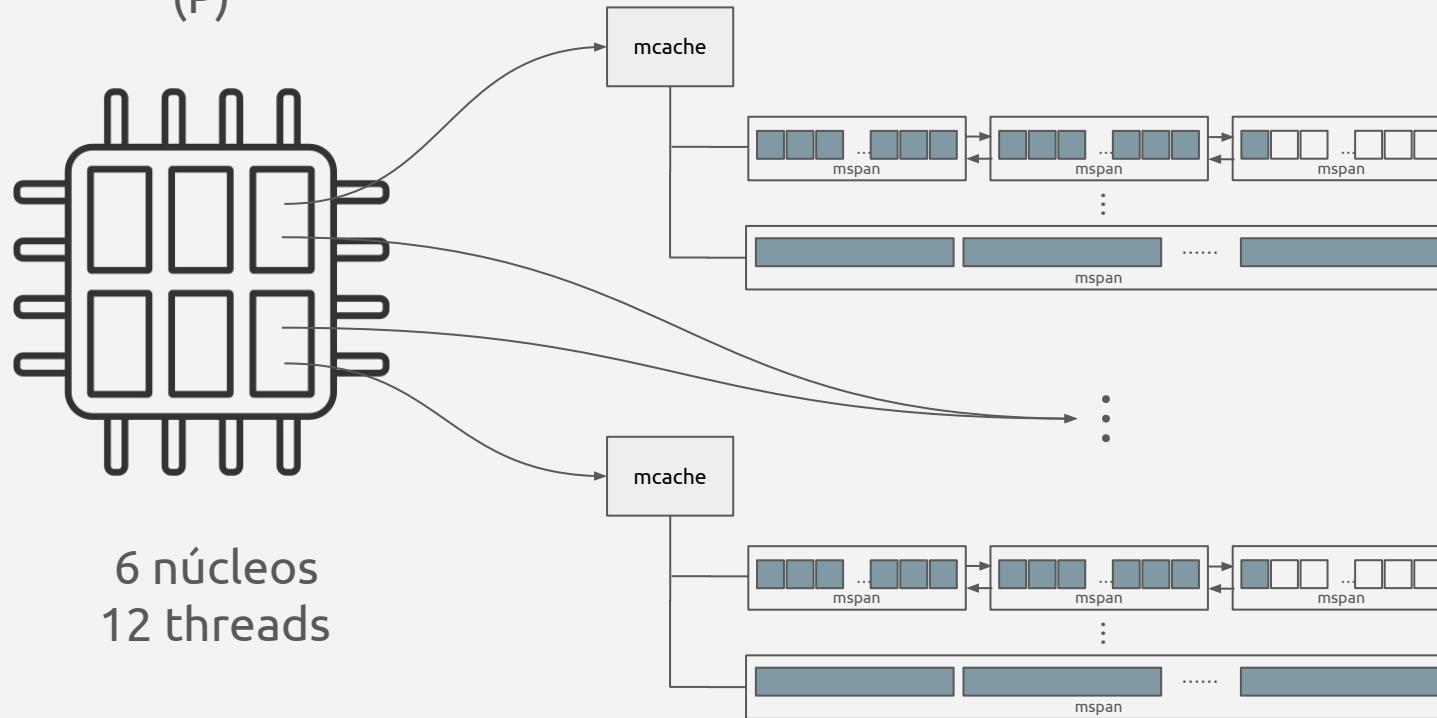
mcache

Thread de execução
da CPU

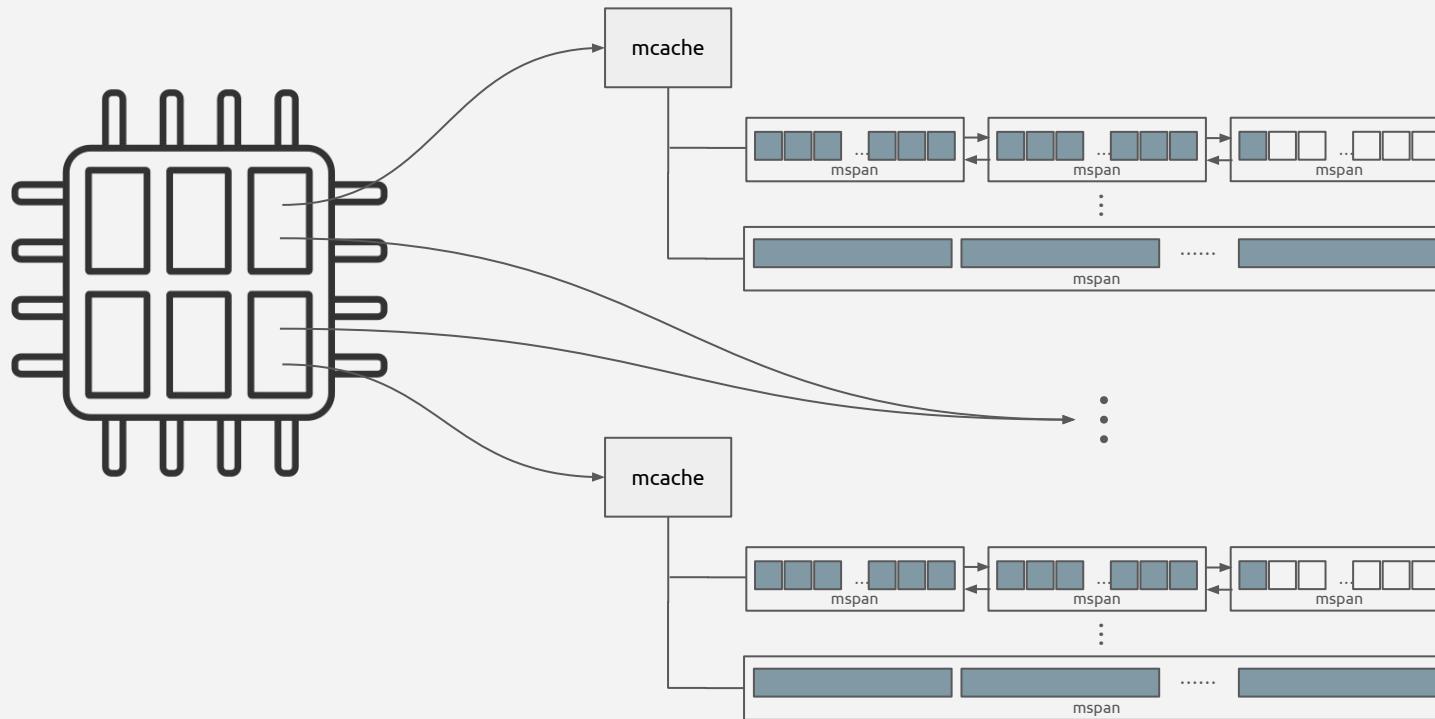


mcache

Processador lógico
(P)



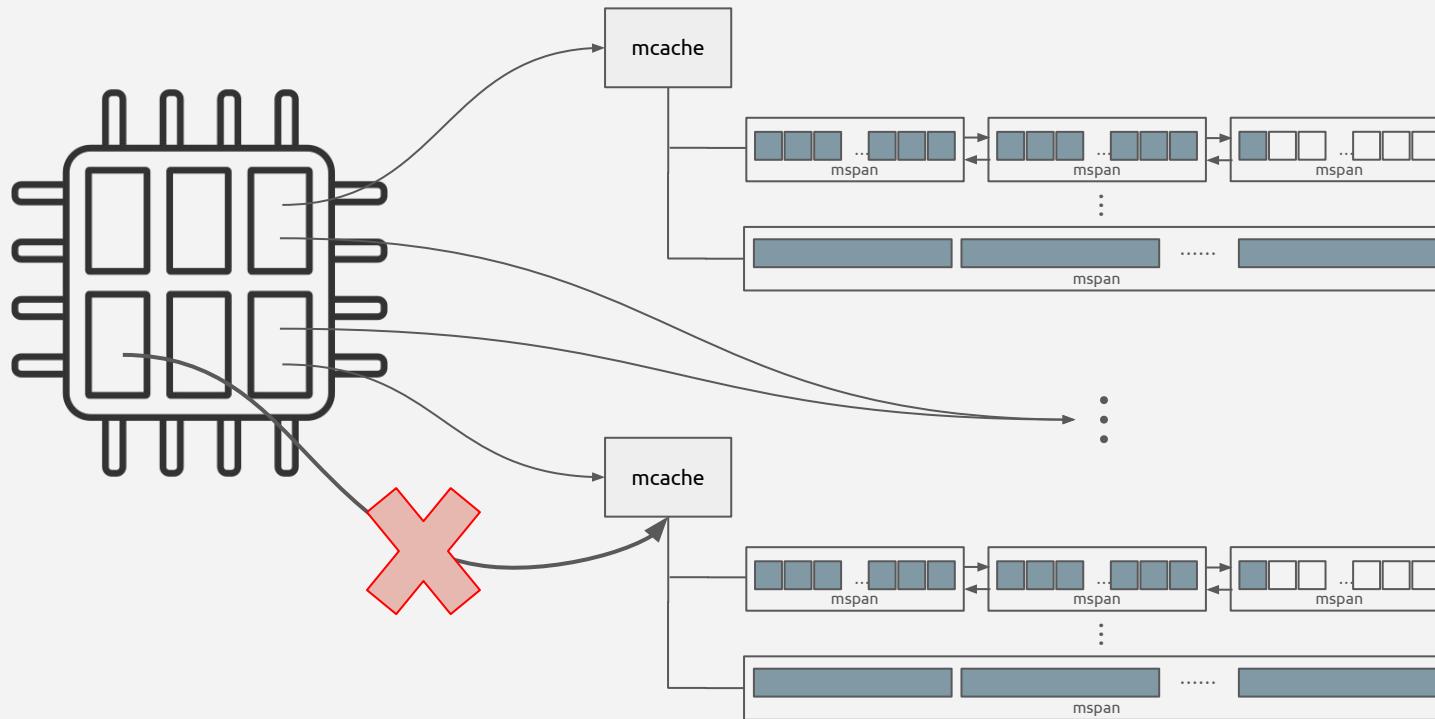
mcache



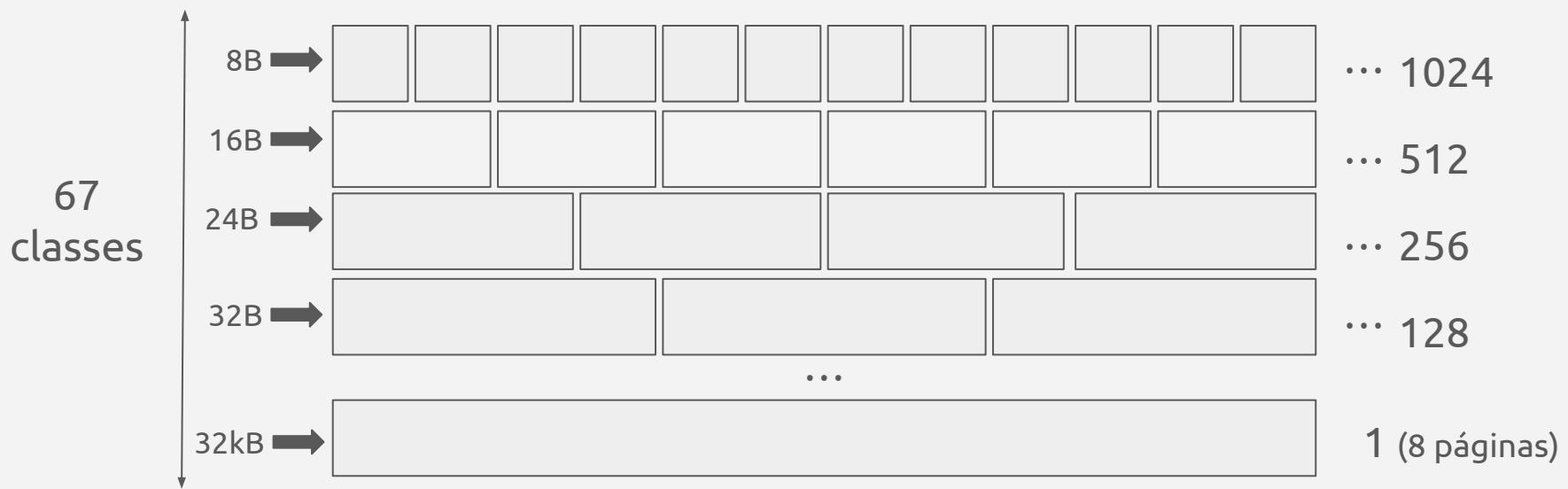
Preserva os valores no cache da CPU

Operação	Latência
Clock da CPU (4GHz)	0.25ns
Acesso Cache L1	0,75ns
Acesso Cache L2	7ns
Lock/Unlock de um Mutex	25ns
Acesso à memória RAM	100ns
Leitura de 4KB do SSD	150.000ns

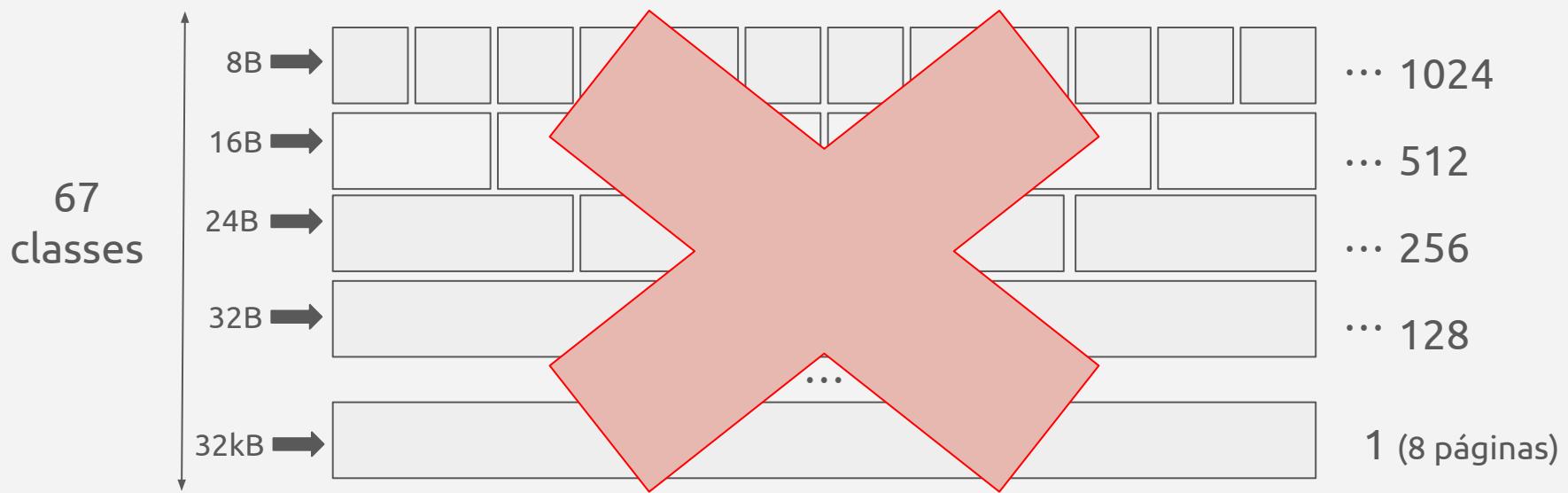
mcache



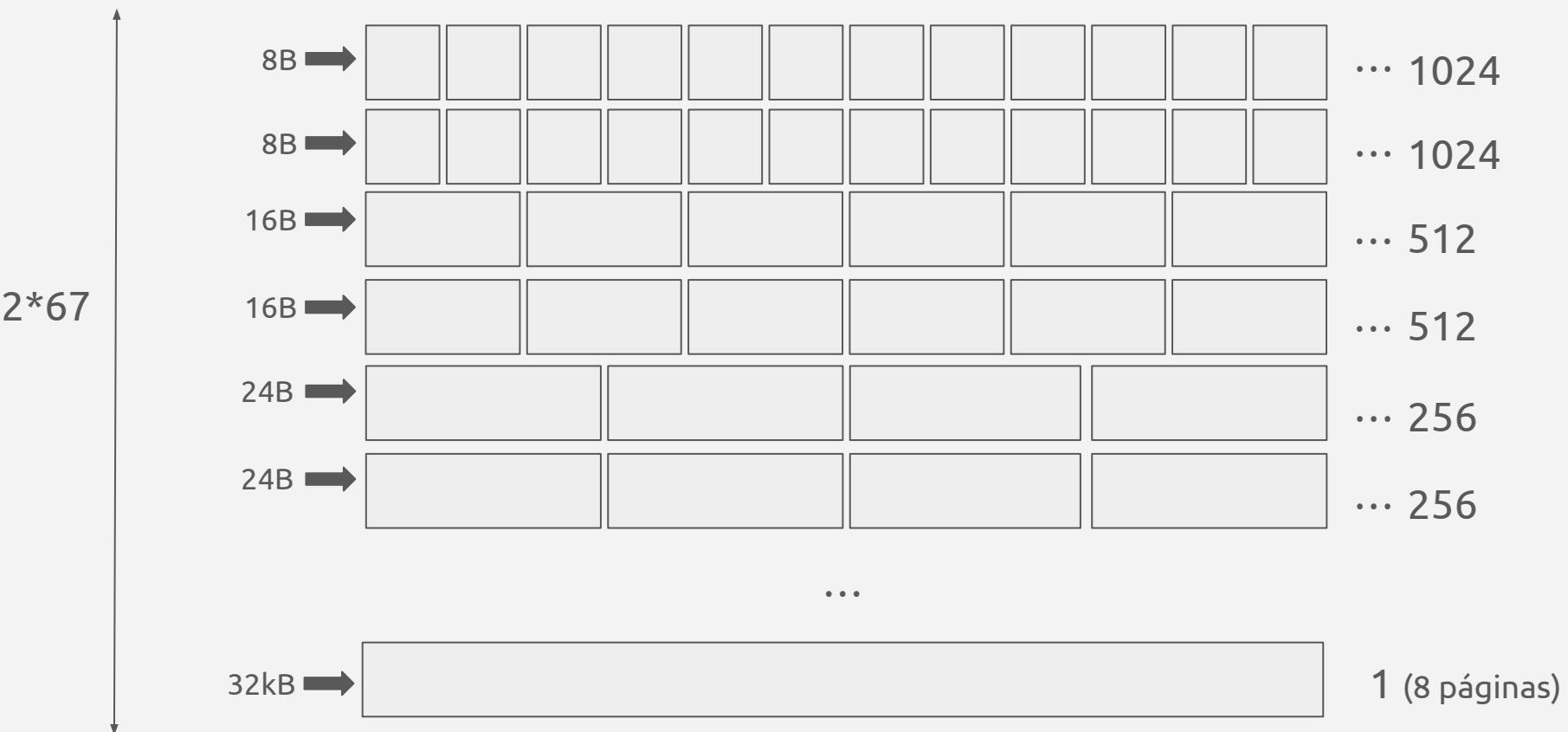
Cada mcache tem 67 classes



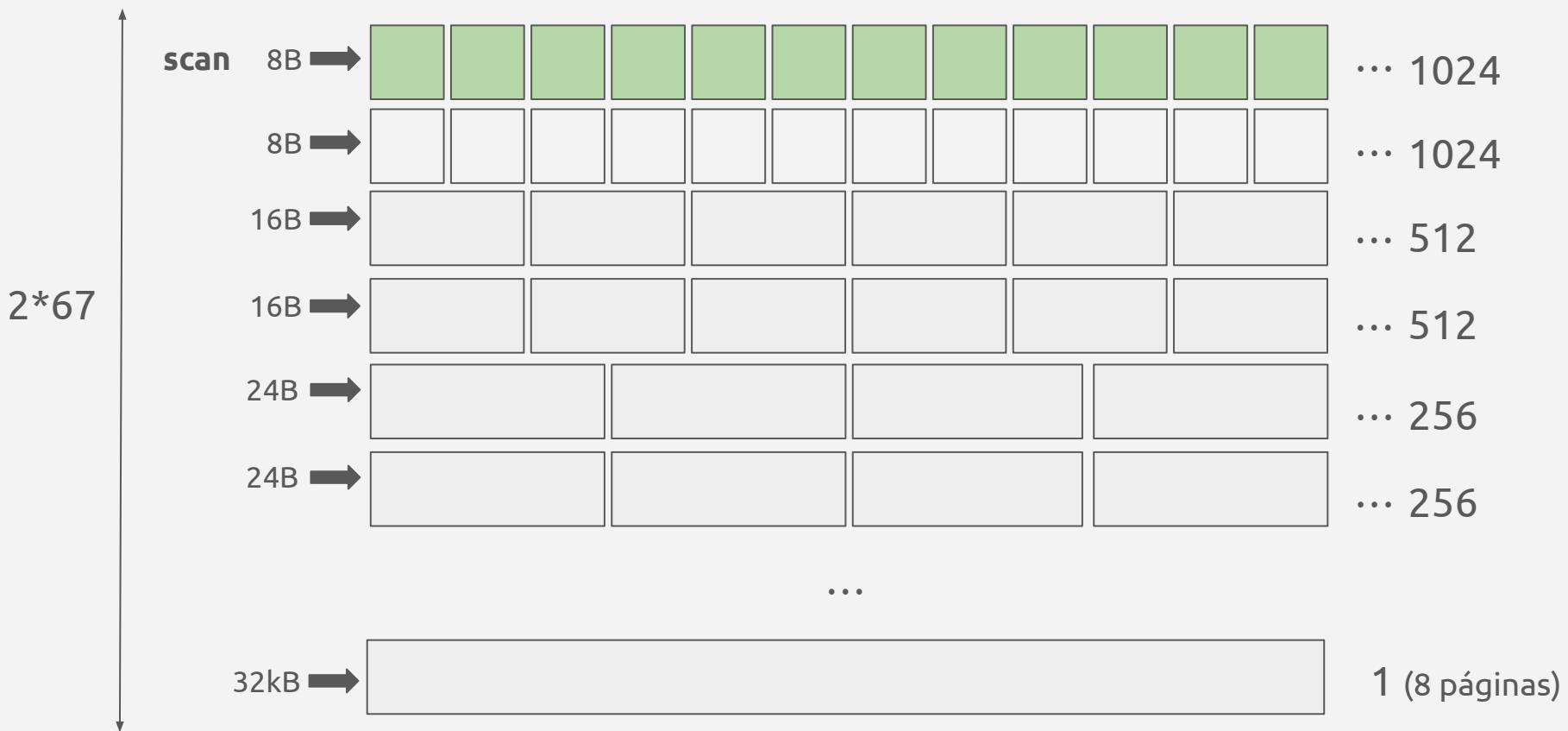
Cada mcache tem 67 classes



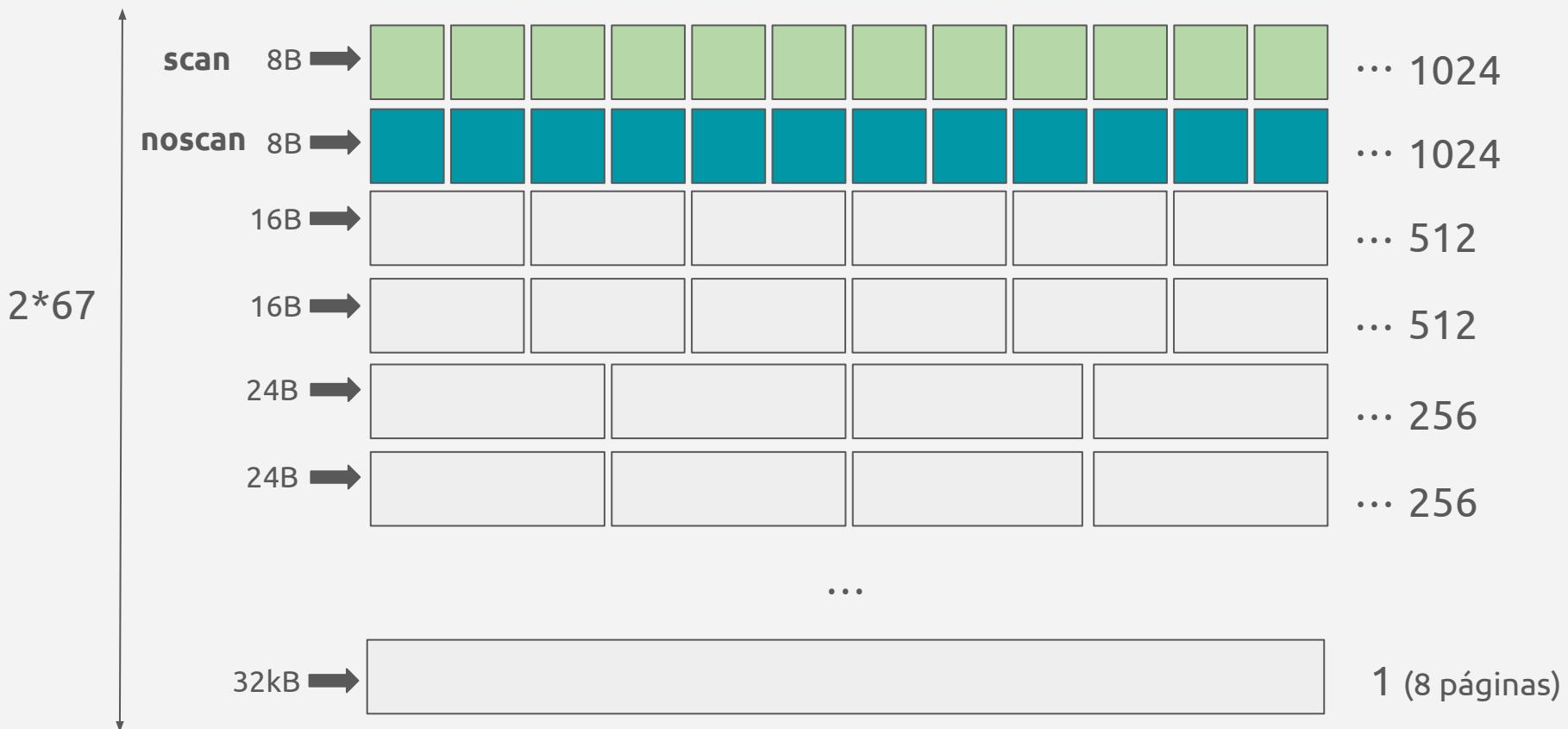
Cada mcache tem 134 classes



Cada mcache tem 134 classes



Cada mcache tem 134 classes

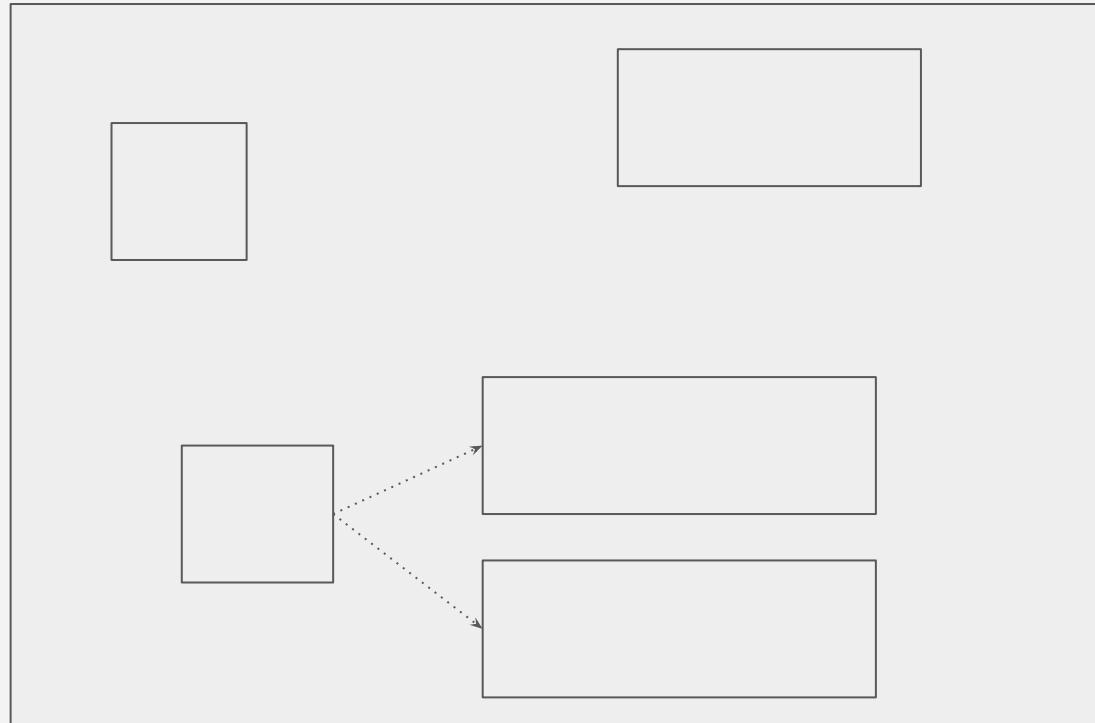


Como o GC funciona

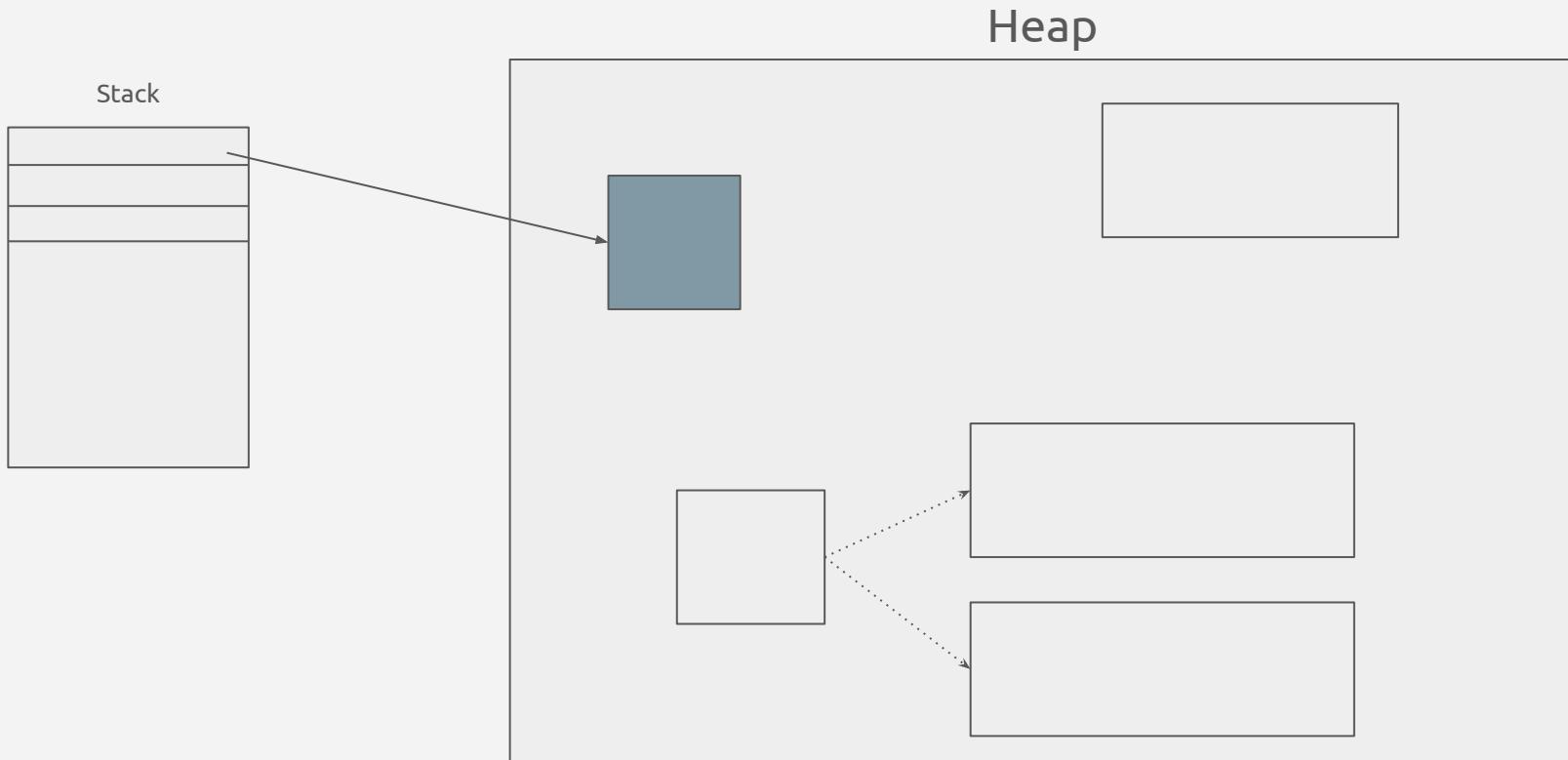
Stack



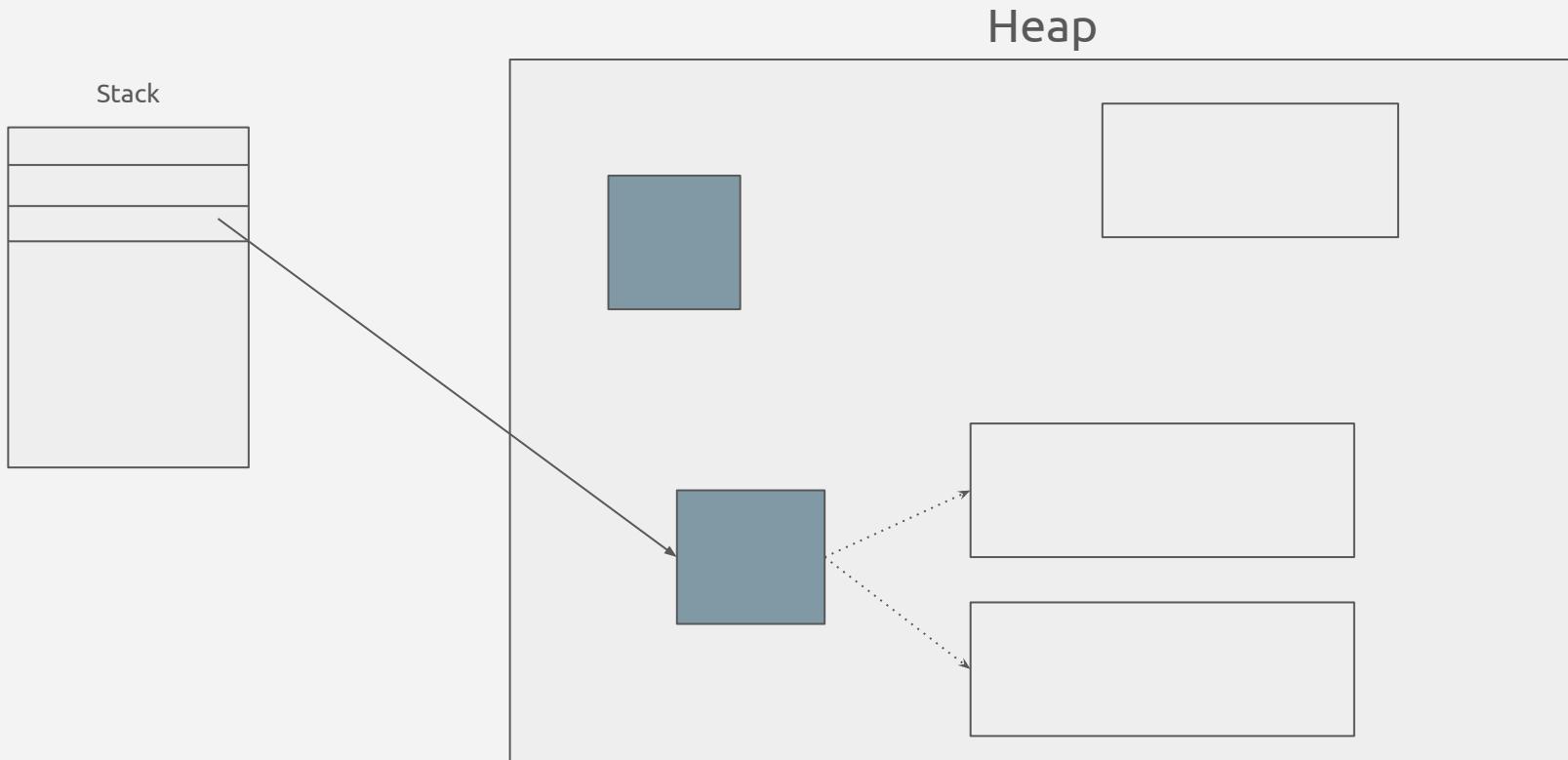
Heap



Como o GC funciona

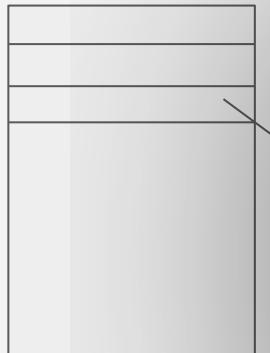


Como o GC funciona



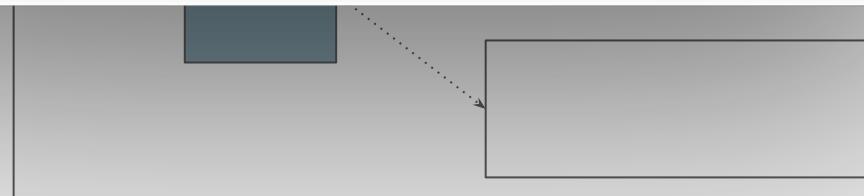
Se a variável tem ponteiros...

Stack

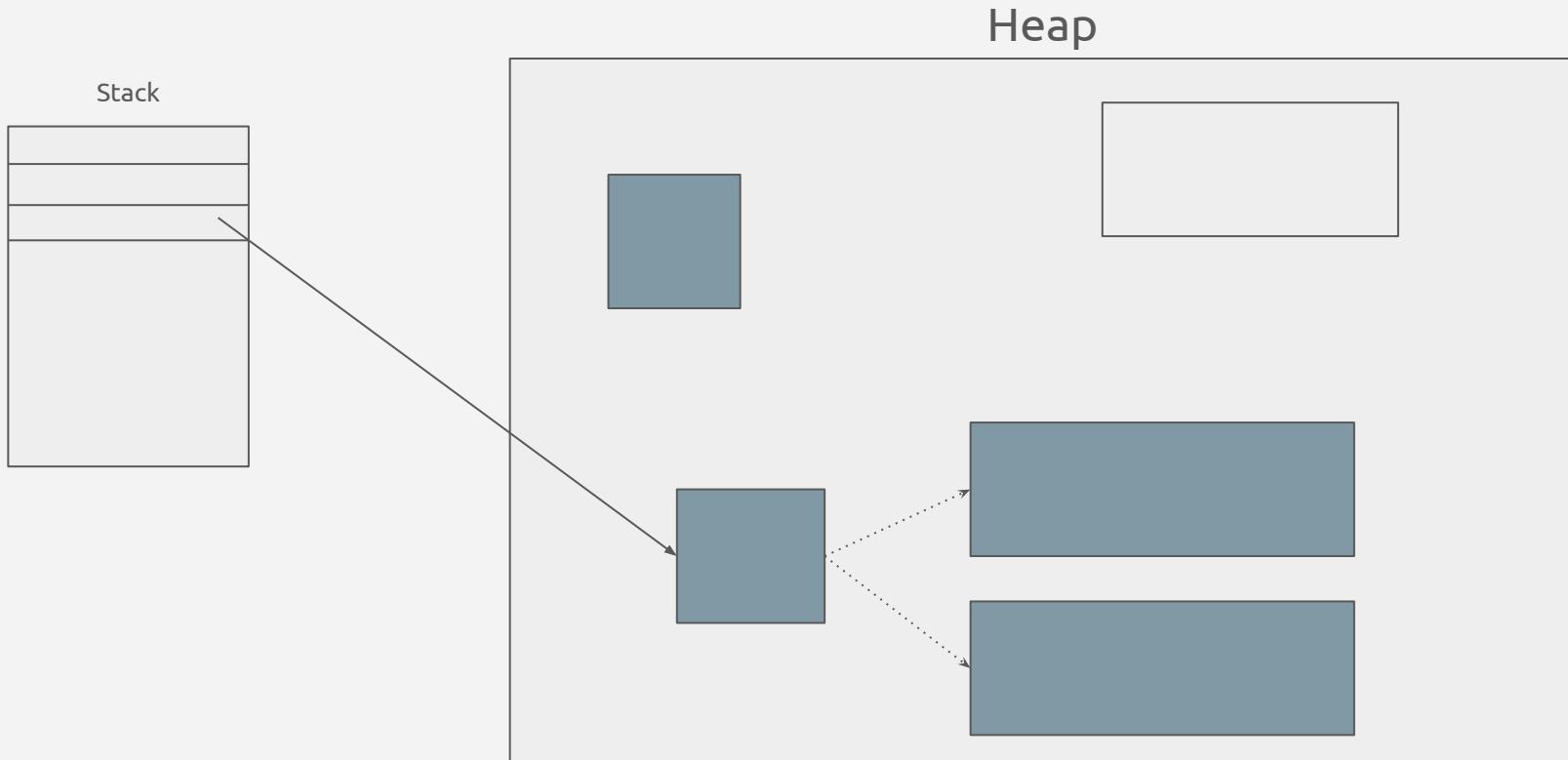


```
type Node struct {  
    Value      string  
    Left, Right *Node  
}
```

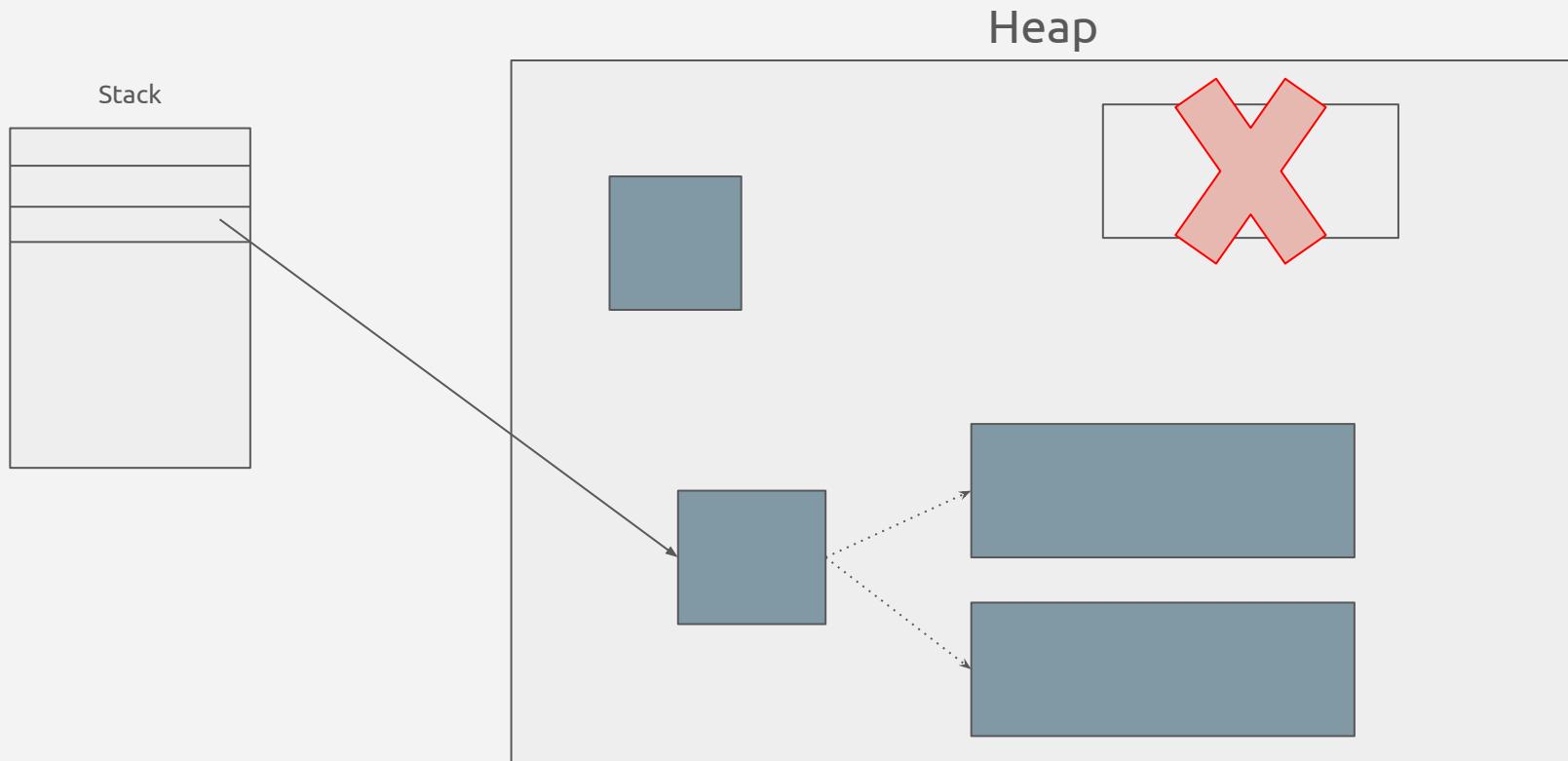
Heap



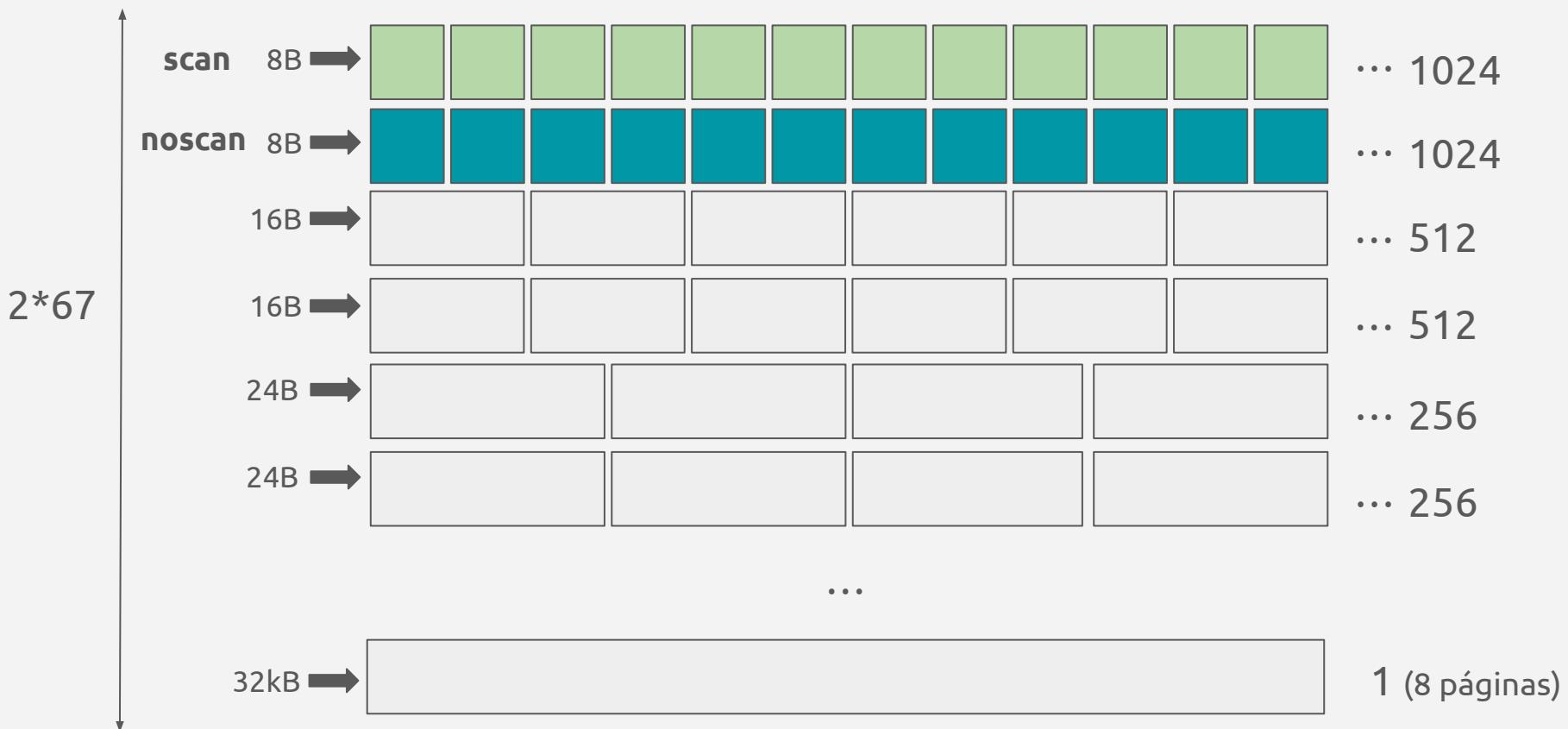
Persegue os ponteiros



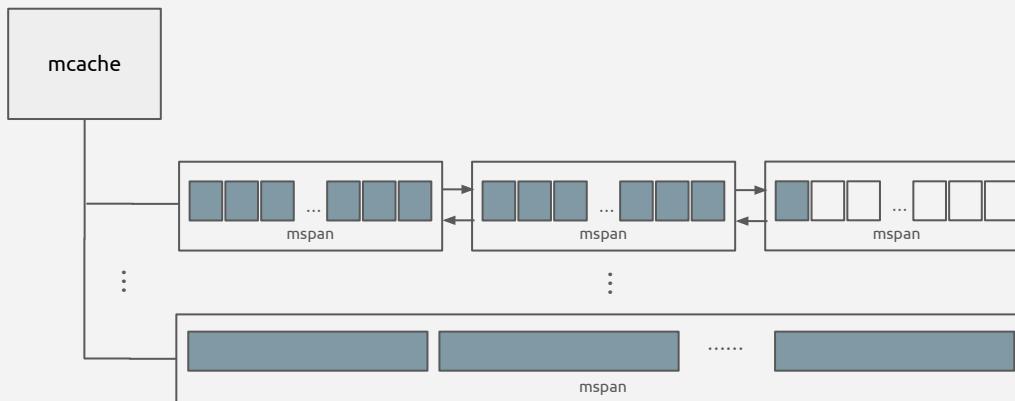
Elimina o que não foi marcado



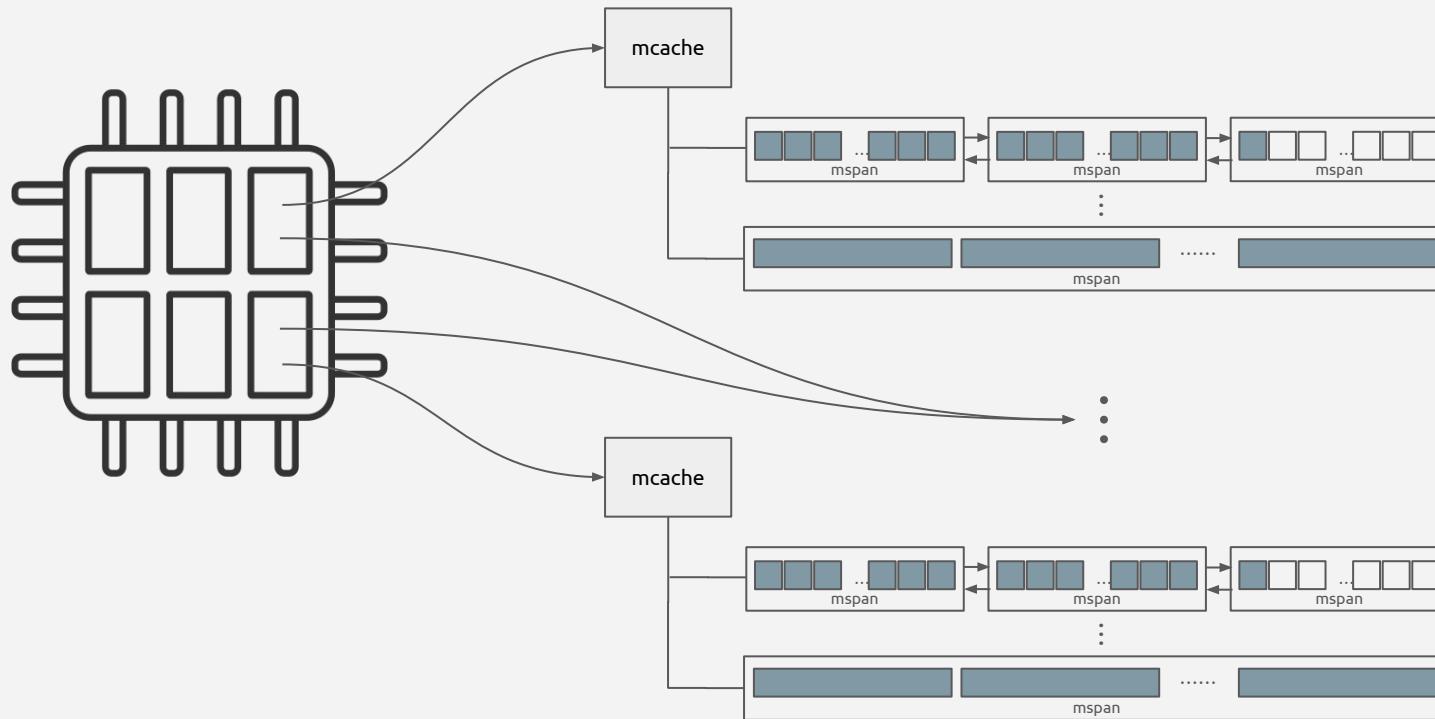
Cada mcache tem 134 classes



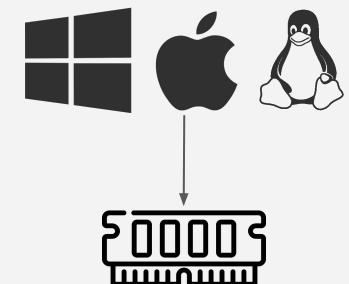
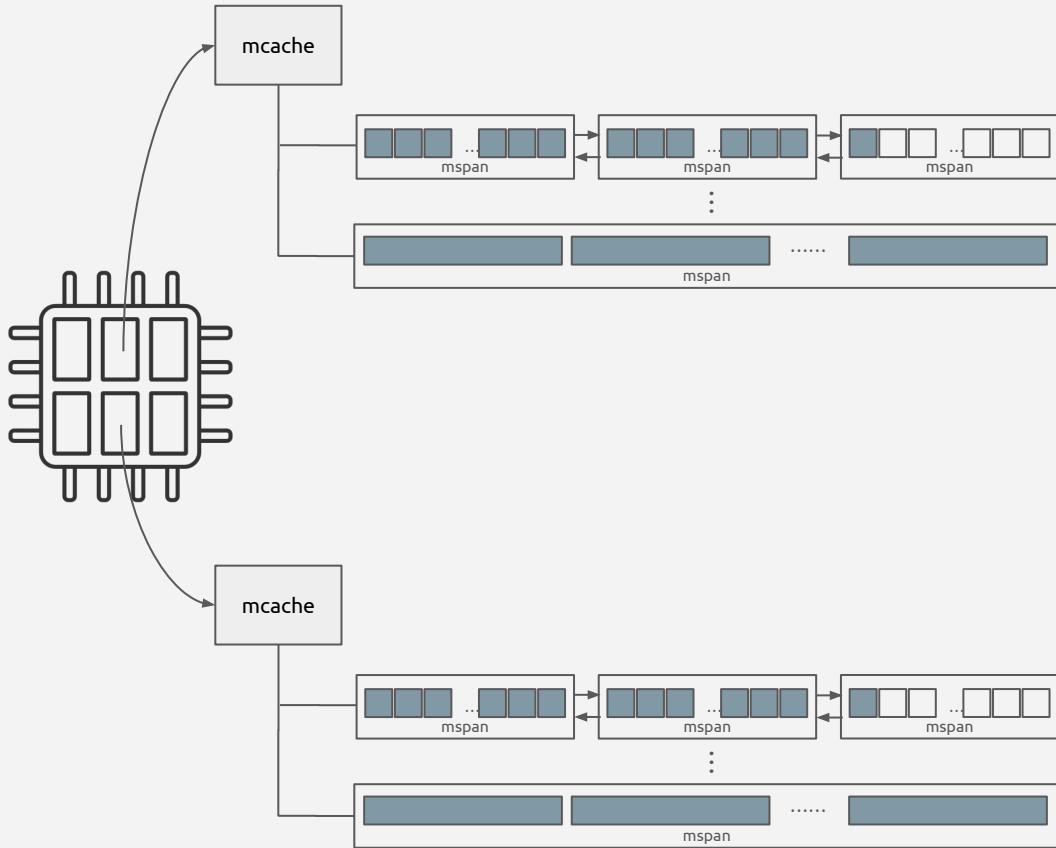
mcache



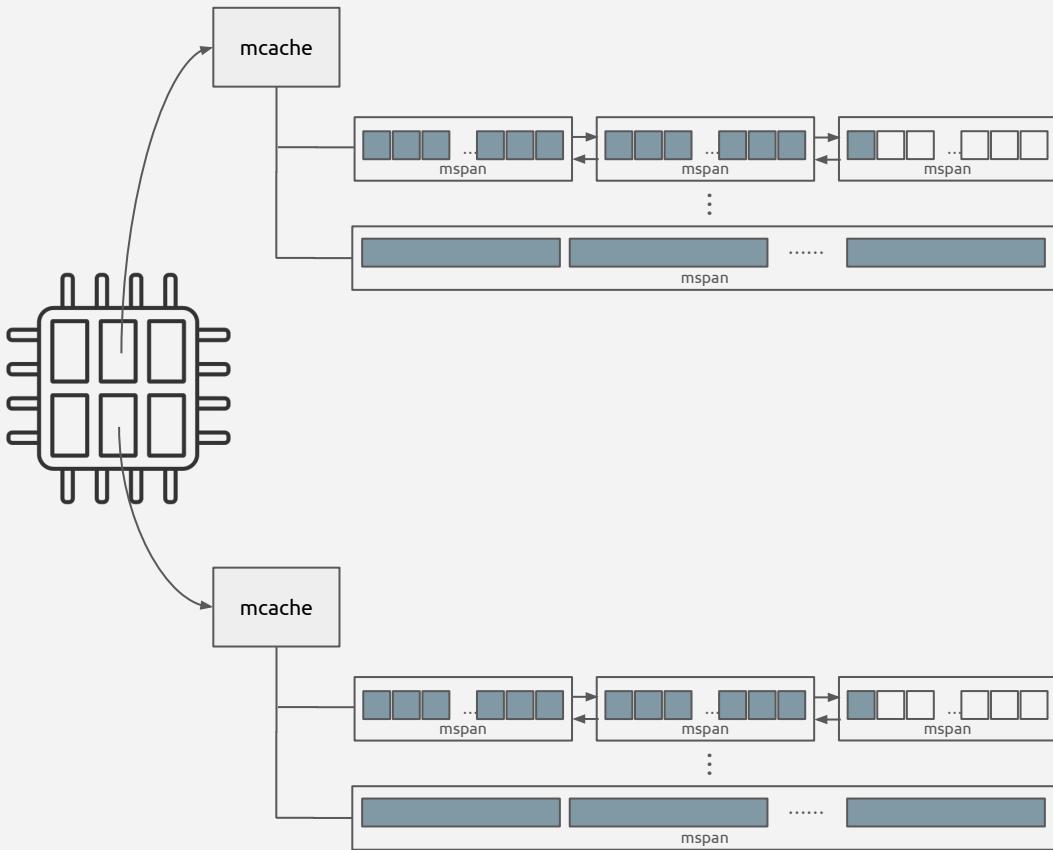
mcache para cada processador lógico



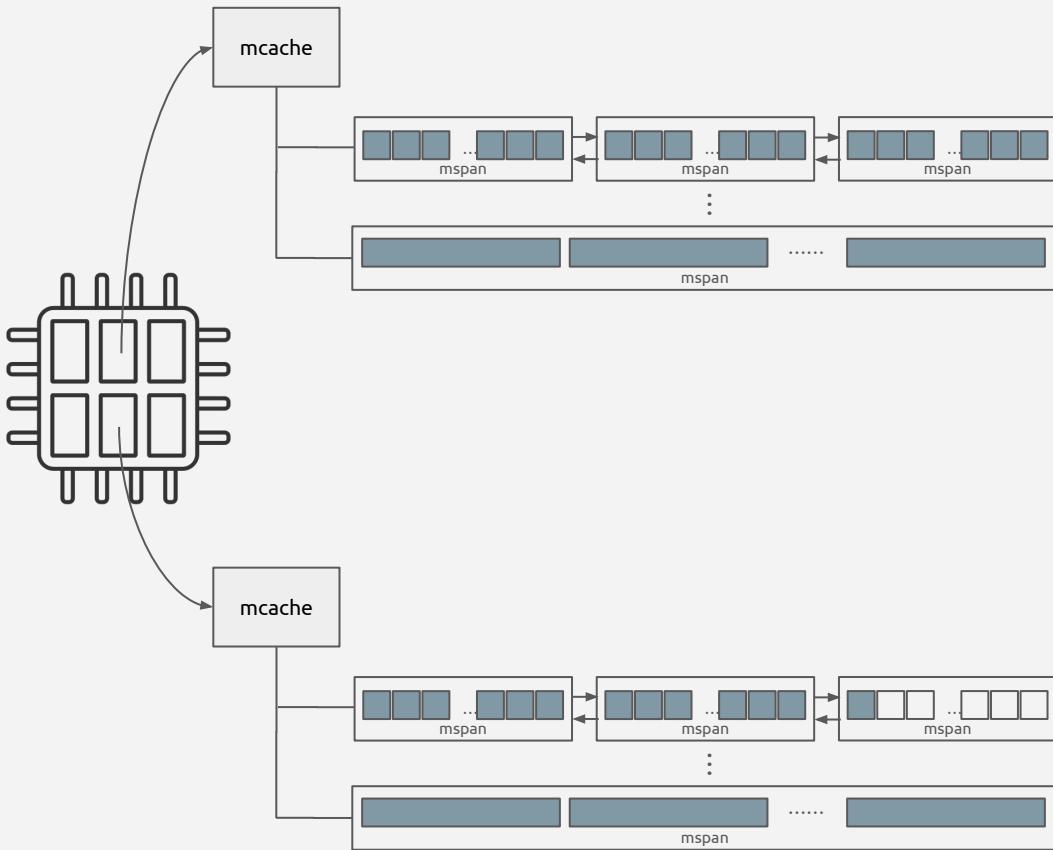
No fim, a memória é uma só



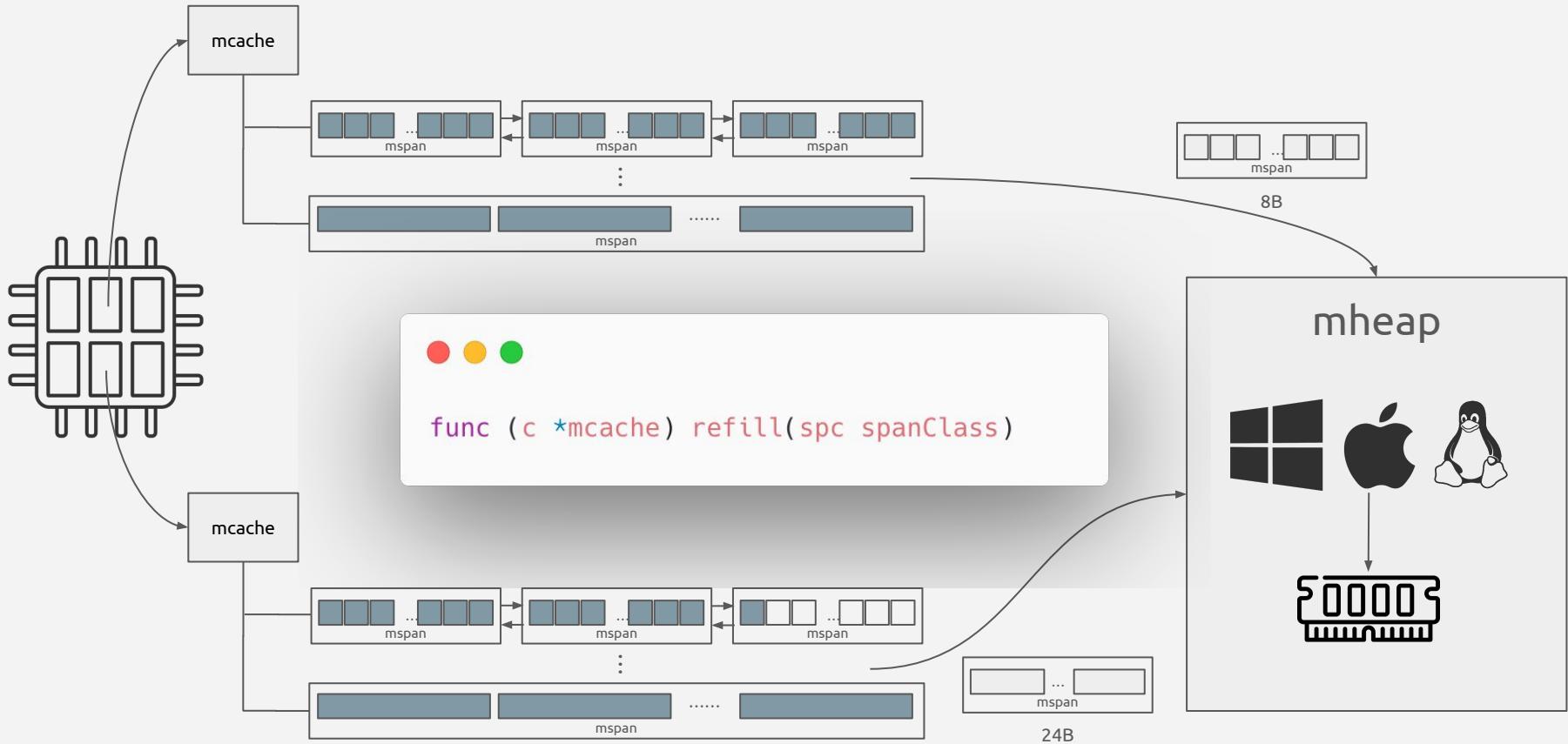
mheap



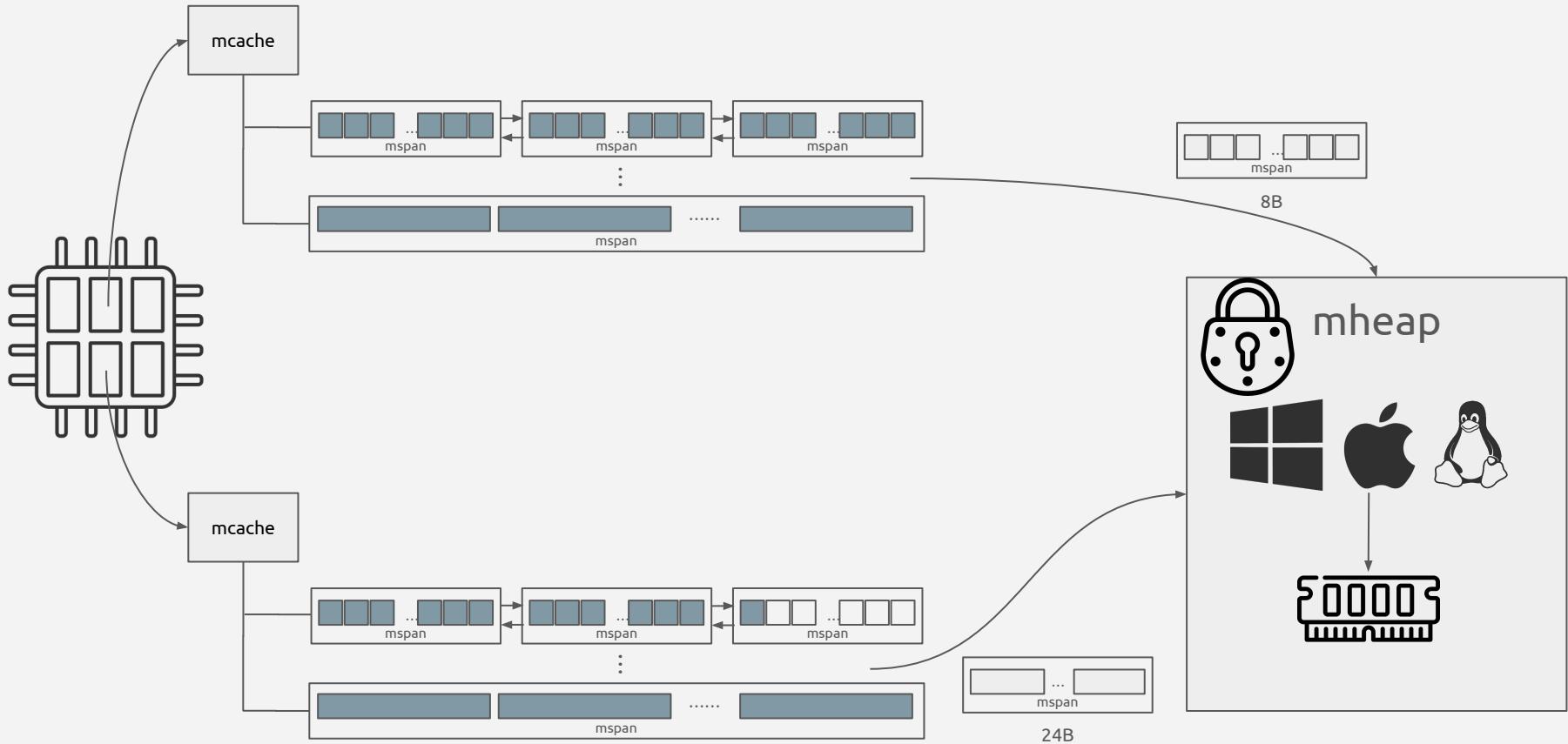
mheap



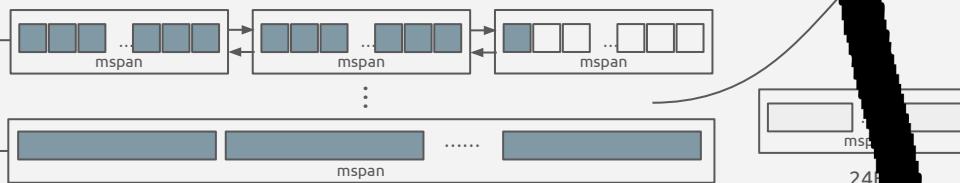
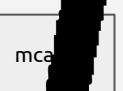
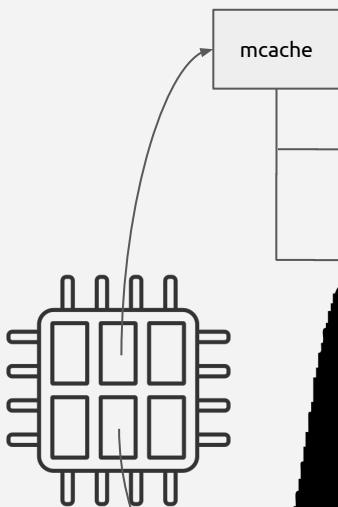
mheap



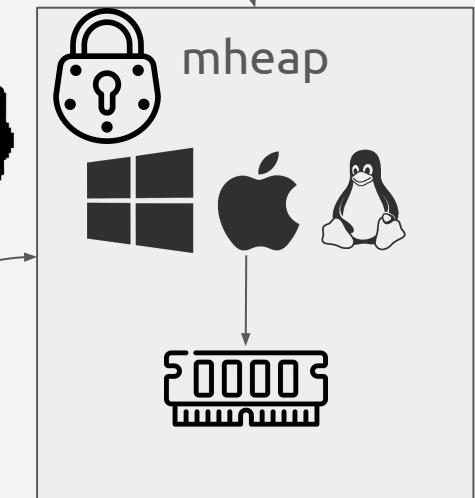
mheap



mheap



8B



mheap



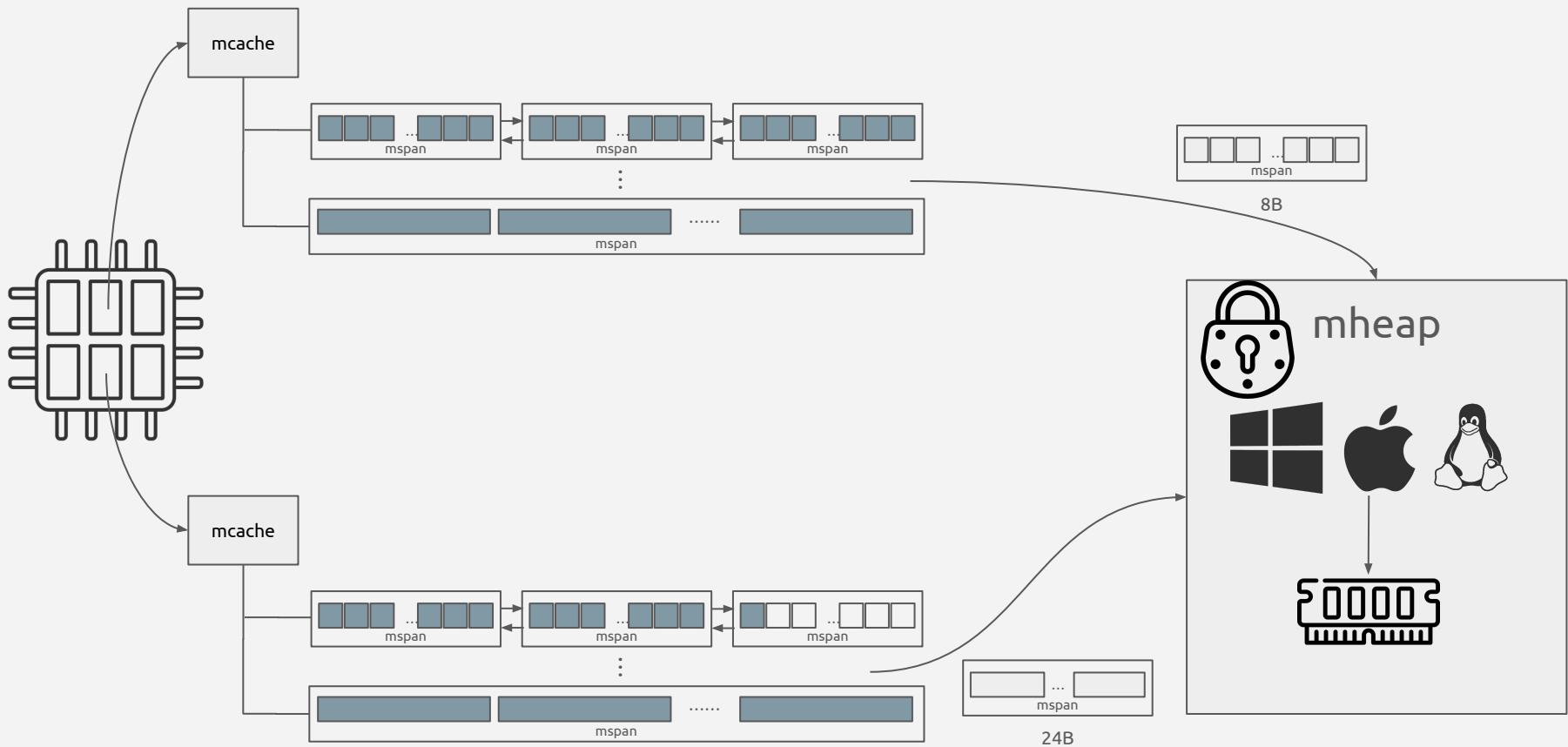
```
// runtime/mheap.go
type mheap struct {
    ...
    lock      mutex ←
    pages     pageAlloc
    ...
    allspans [ ]*mspan
}
```

mheap



```
// runtime/mheap.go
type mheap struct {
    ...
    lock      mutex
    pages     pageAlloc
    ...
    allspans []*mspan ←
}
```

Nem todas as alocações na heap são iguais



mcentral

mheap

mcentral



mspan

mspan



mspan

mcentral



mspan

mspan

mspan

mcentral



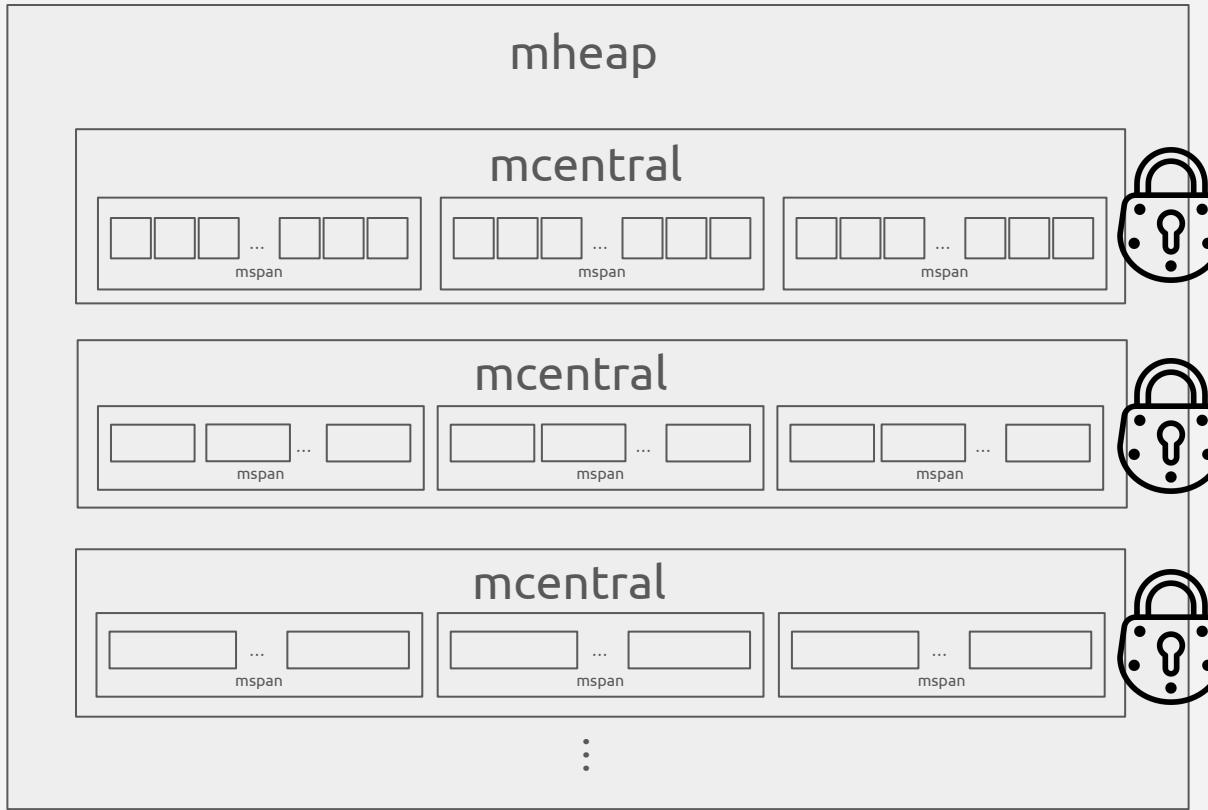
mspan

mspan

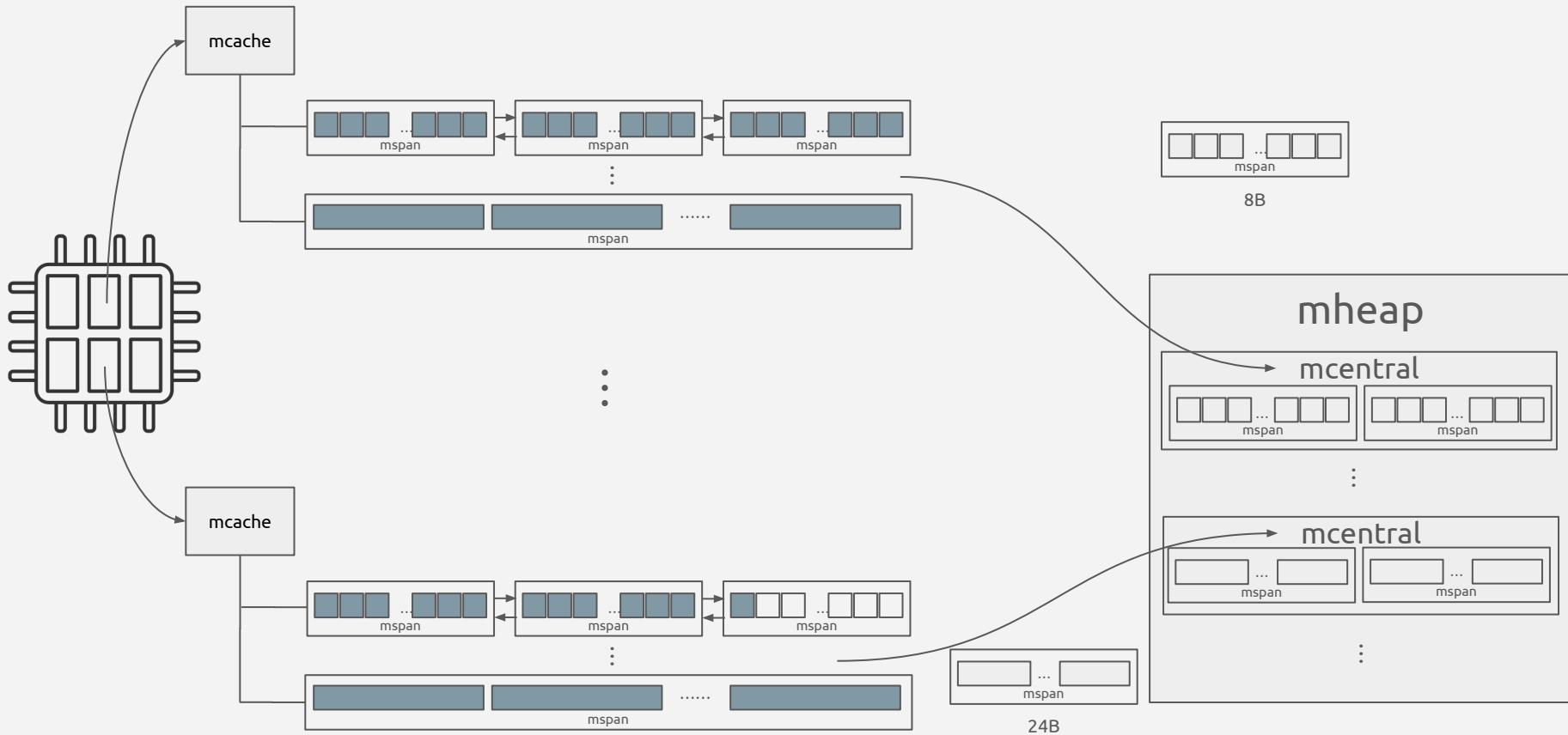
mspan

⋮

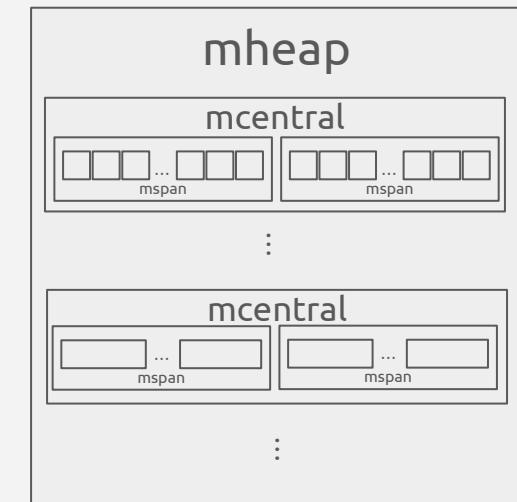
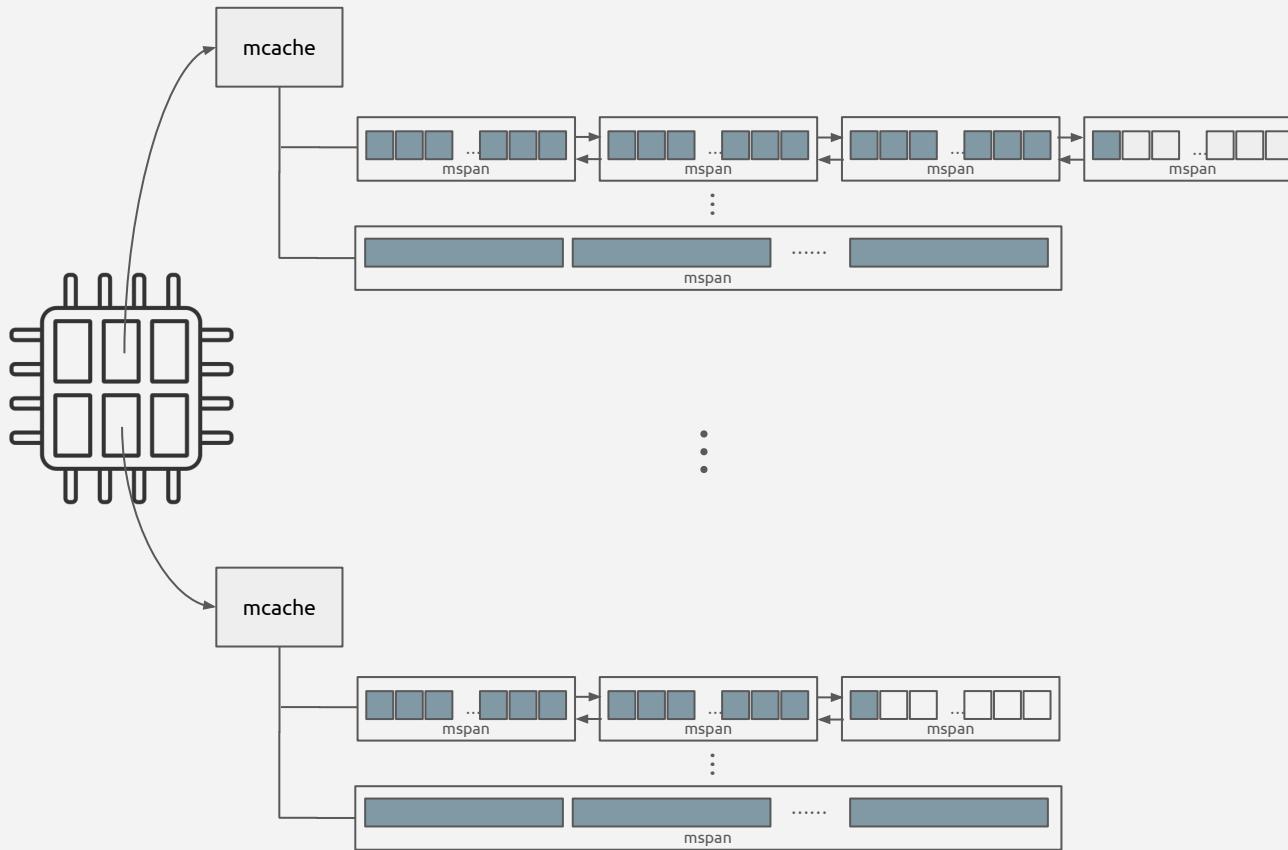
mcentral



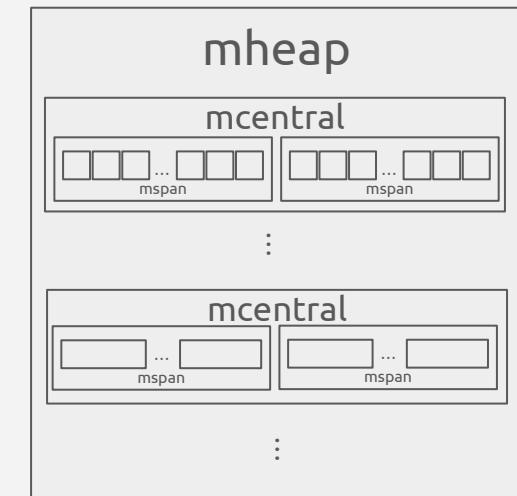
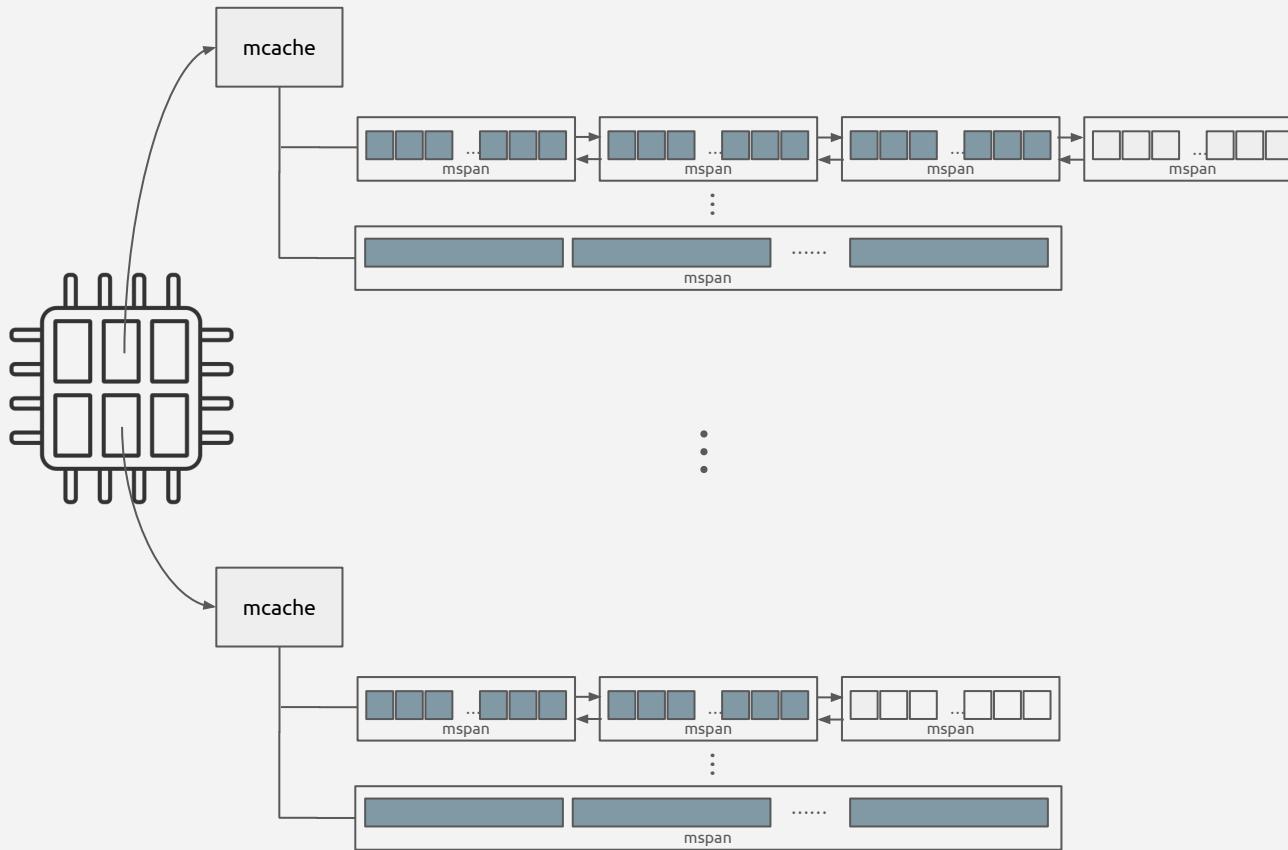
Layout final para alocações pequenas



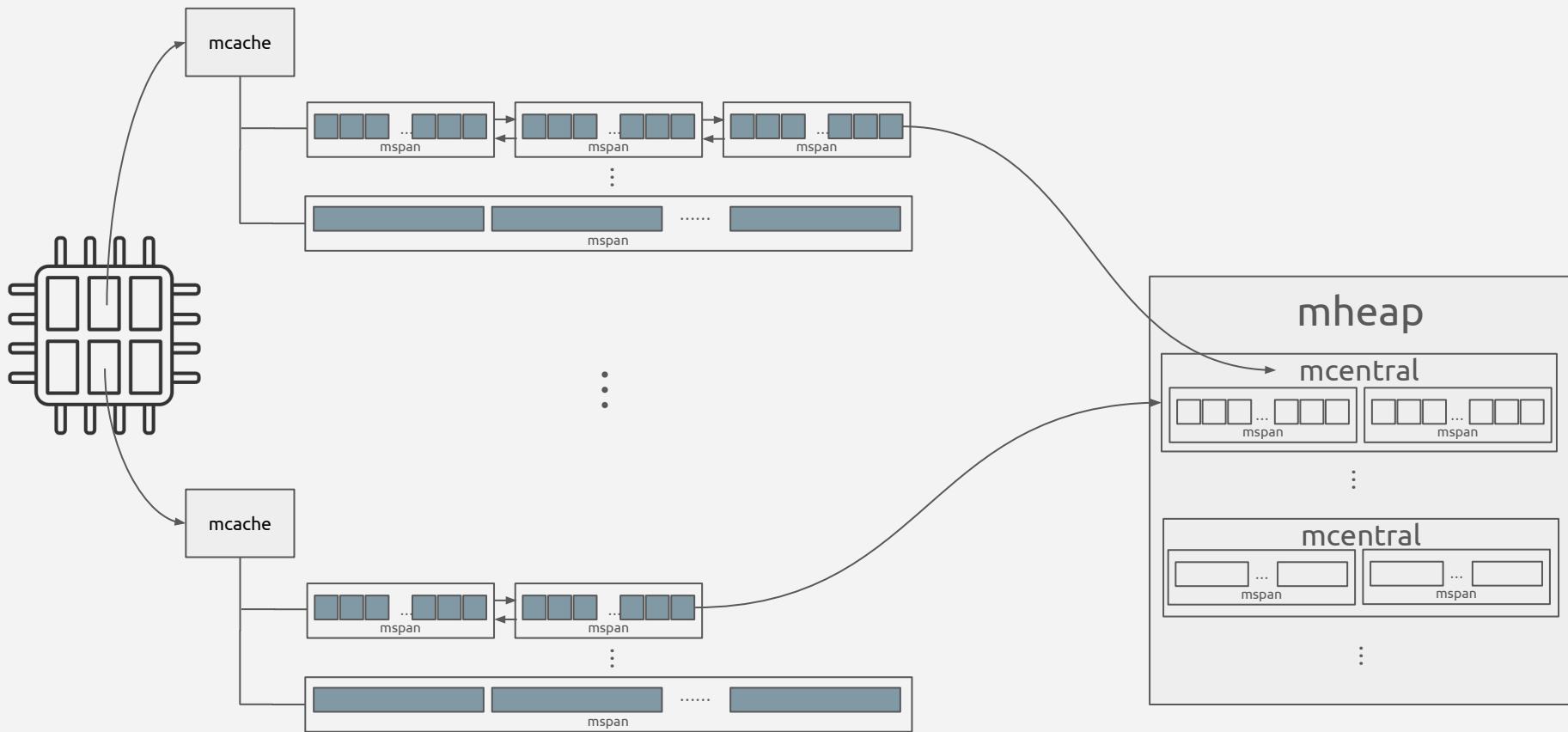
Layout final para alocações pequenas



Spans desalocados retornam para mcentral



Spans desalocados retornam para mcentral



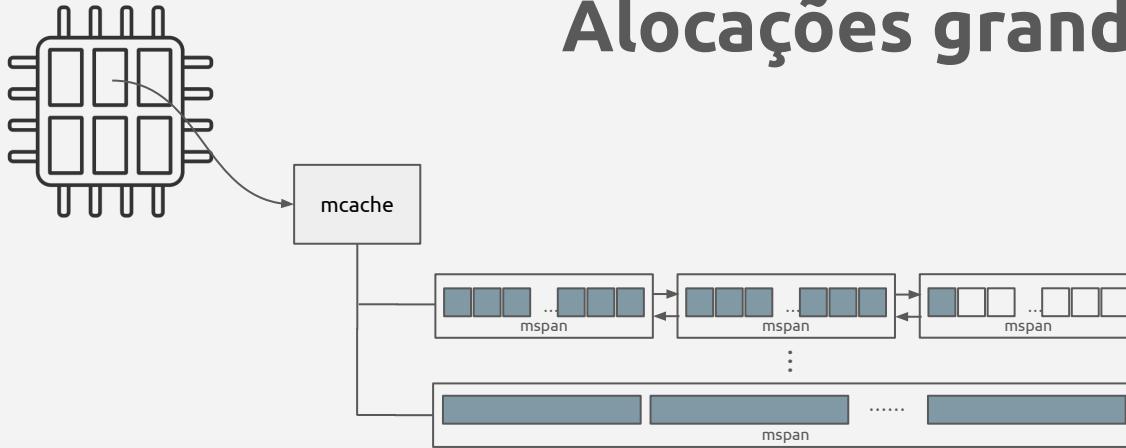
Alocações grandes



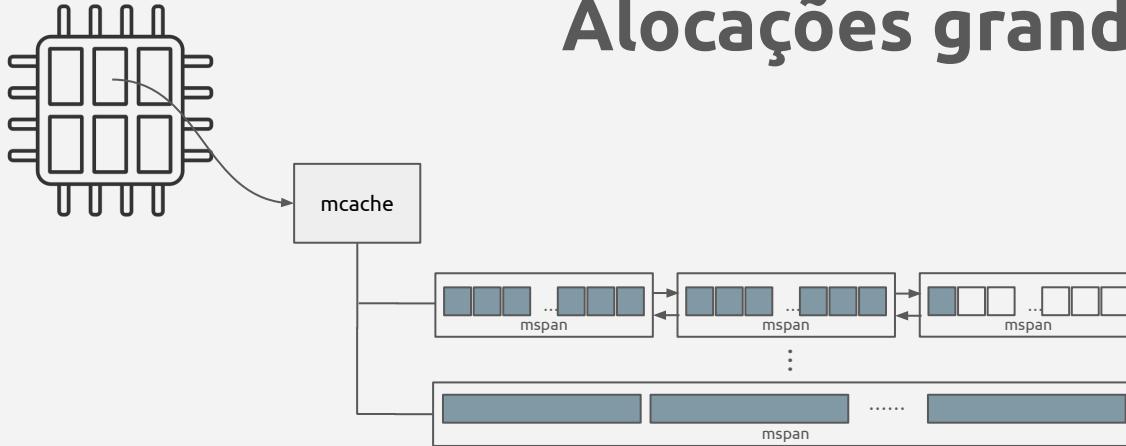
```
big1 := make([]byte, 10*1024*1024)
```

10MB

Alocações grandes

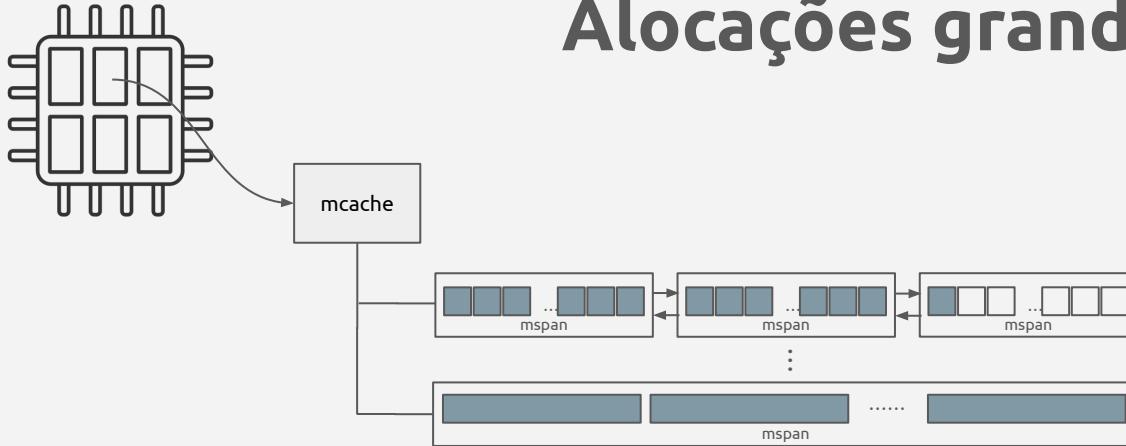


Alocações grandes



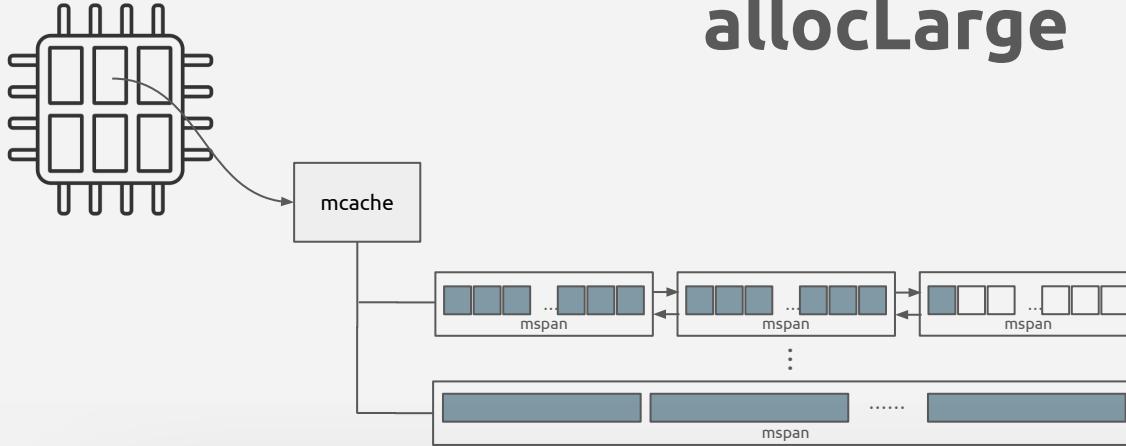
```
func (c *mcache) refill(spc spanClass)
```

Alocações grandes



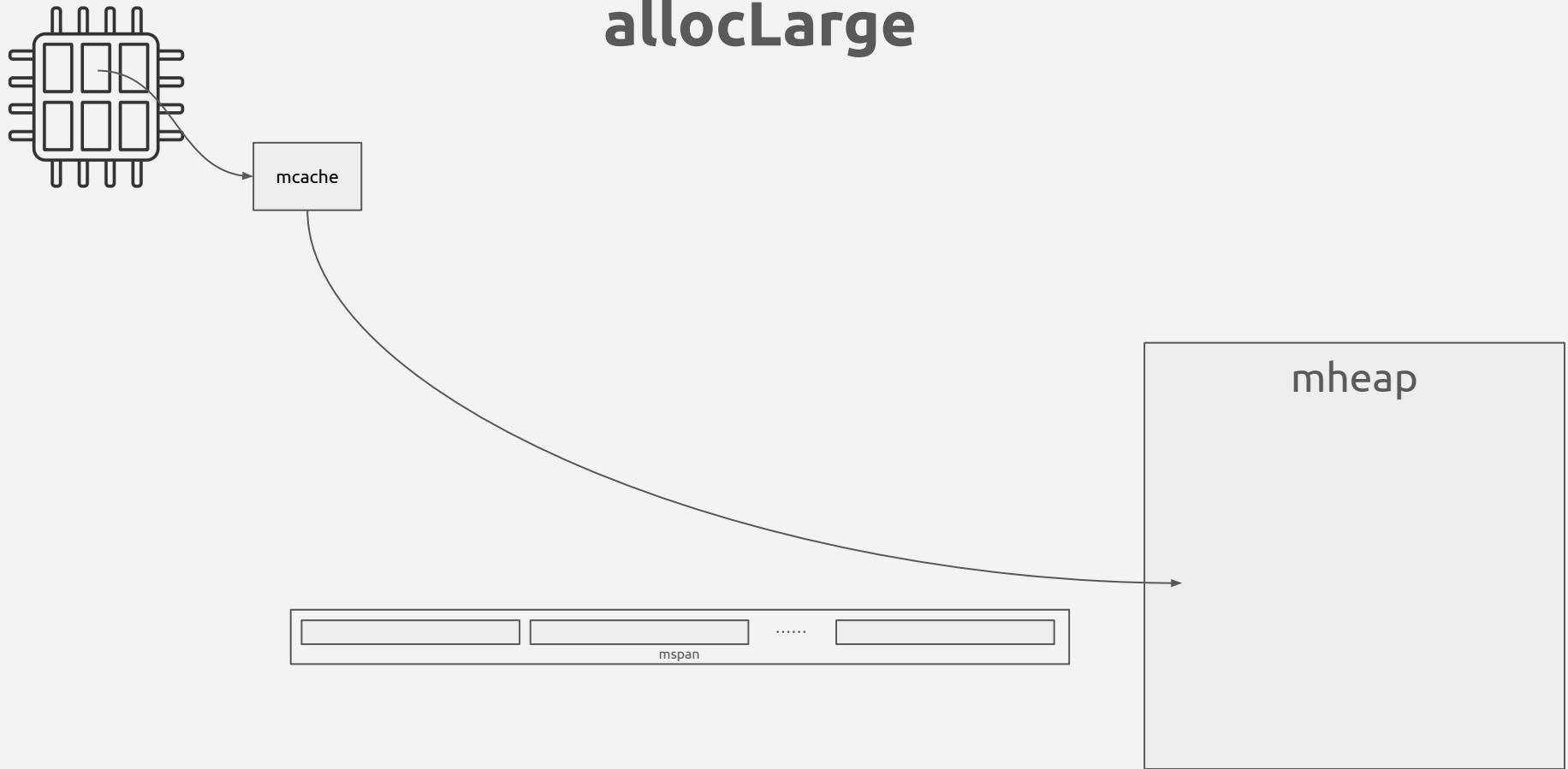
func (c *mcache) GetMSpan(c spanClass)

allocLarge

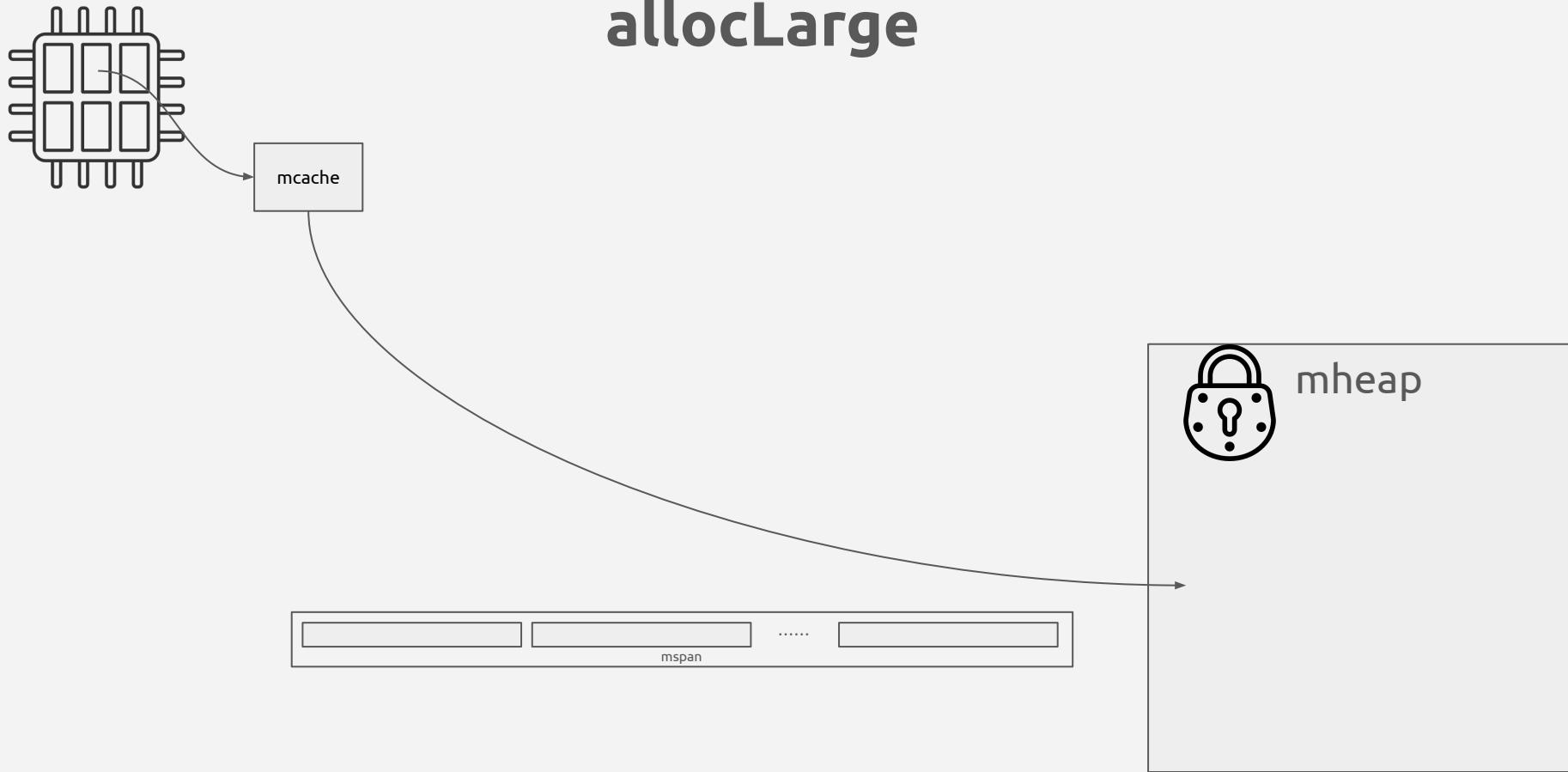


```
func (c *mcache) allocLarge(size uintptr, noscan bool) *mspan
```

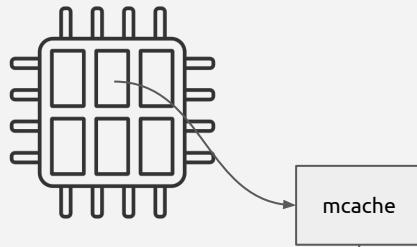
allocLarge



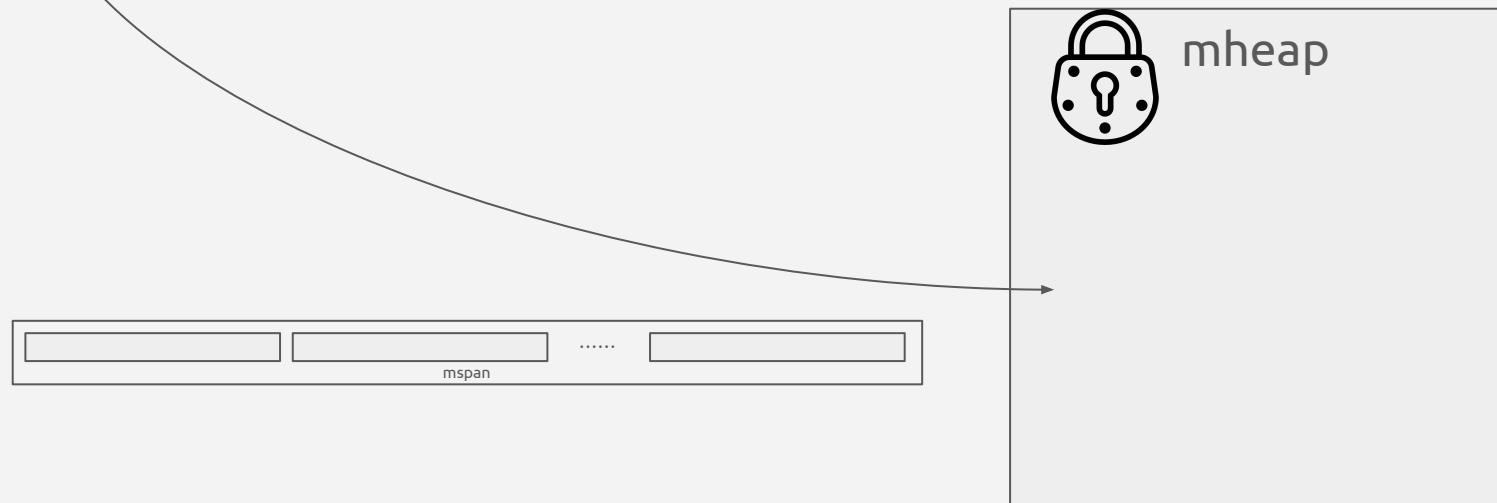
allocLarge



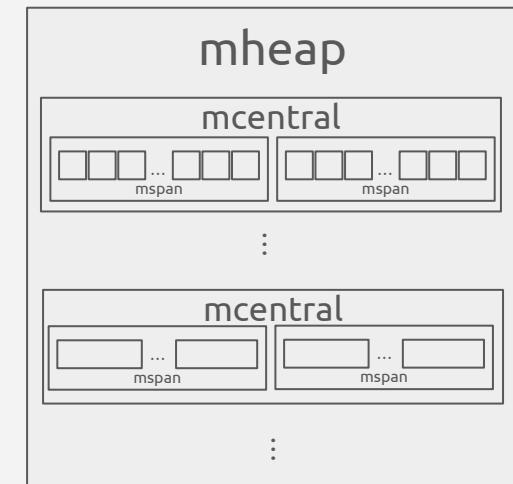
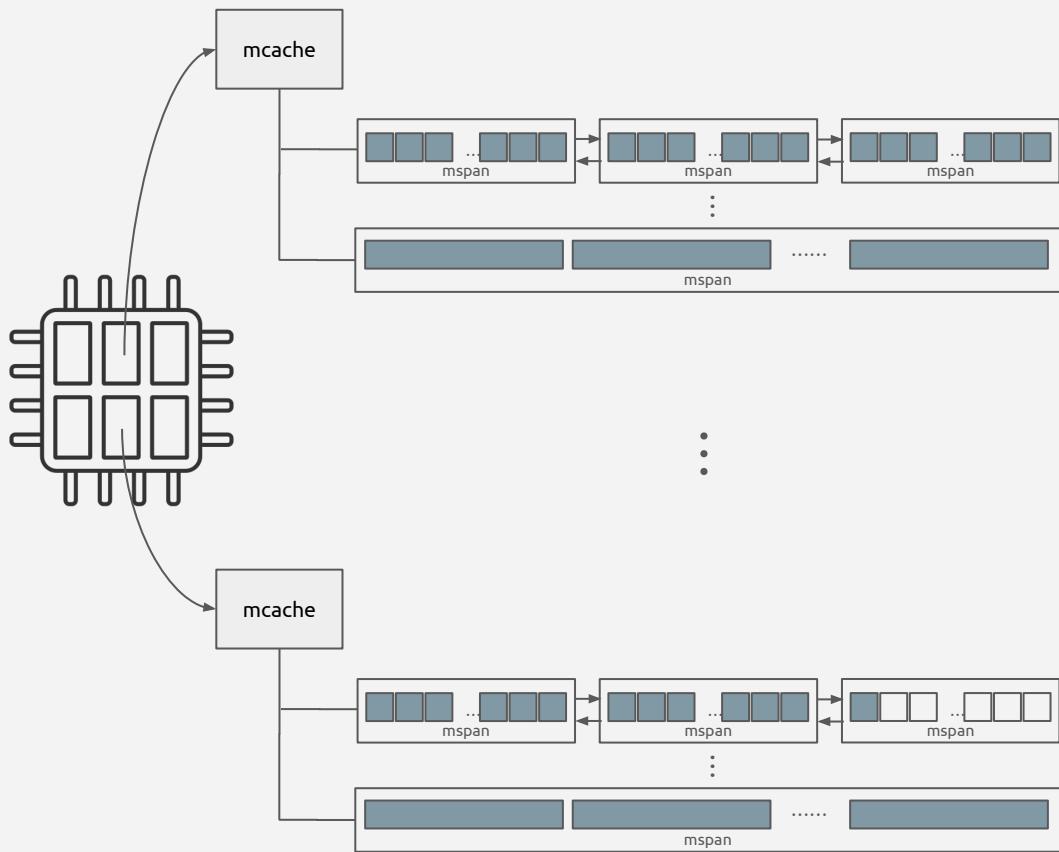
allocLarge



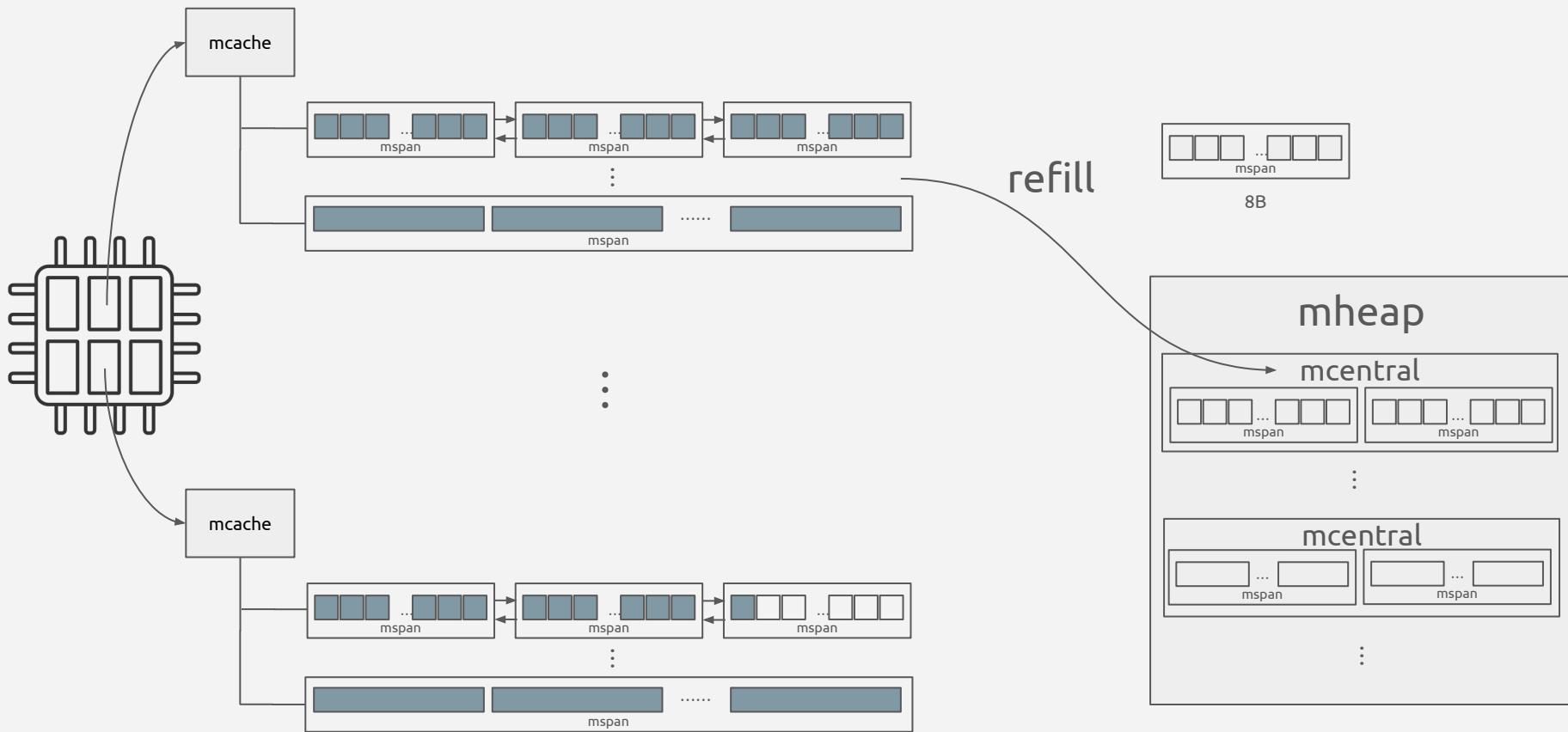
Menos frequentes = menos contenção



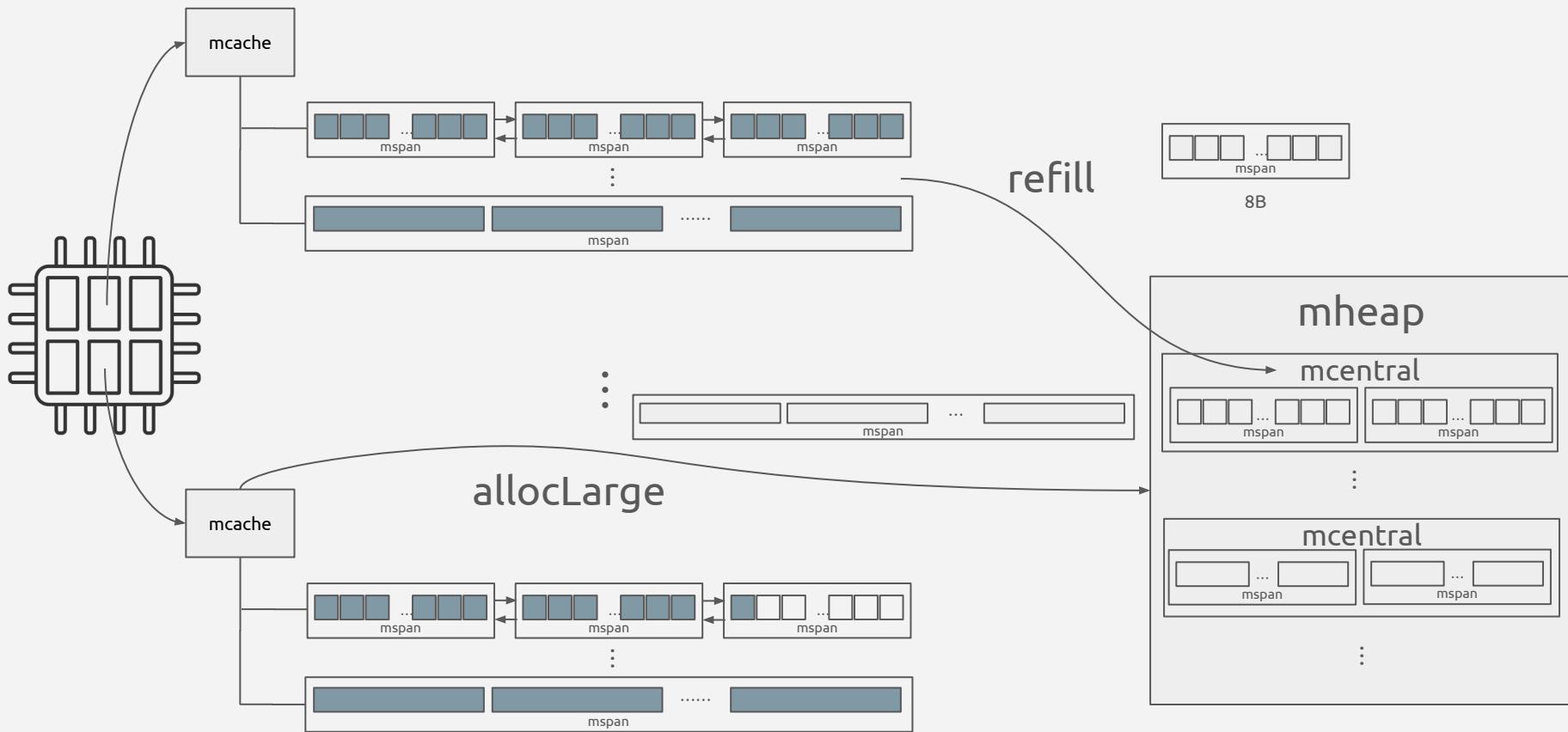
Memory Zoo



Memory Zoo



Memory Zoo



Arenas



```
import "arena"

func main() {
    a := arena.New()
    defer a.Free()

    x := a.New((*MyType)(nil)).(*MyType)
    s := a.NewSlice((*[]int)(nil), 100).([]int)
}
```

Go 1.20

Feature
experimental

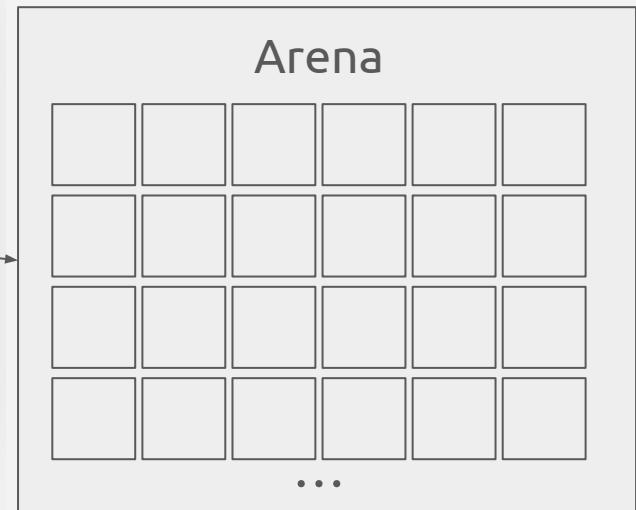
Arenas



```
import "arena"

func main() {
    a := arena.New()
    defer a.Free()

    x := a.New((*MyType)(nil)).(*MyType)
    s := a.NewSlice(([]int)(nil), 100).([]int)
}
```



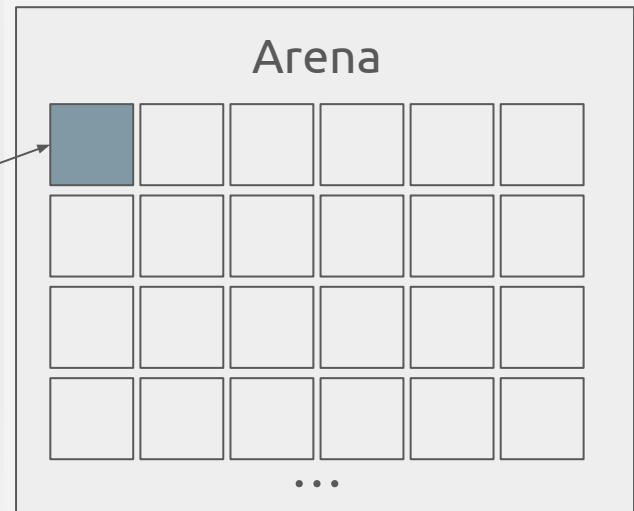
Arenas



```
import "arena"

func main() {
    a := arena.New()
    defer a.Free()

    x := a.New((*MyType)(nil)).(*MyType)
    s := a.NewSlice((*[]int)(nil), 100).([]int)
}
```



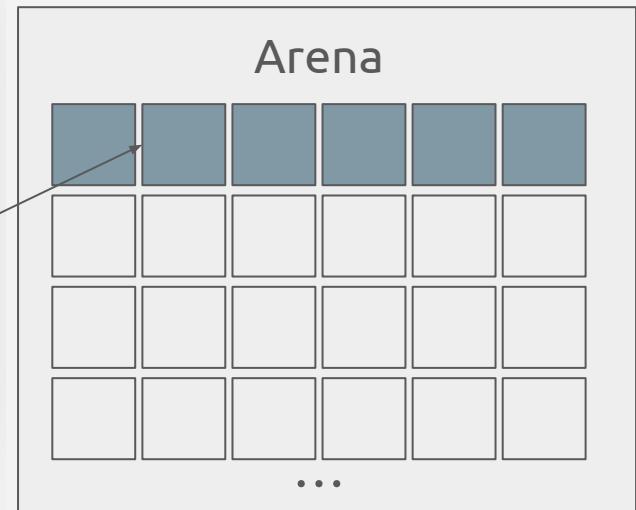
Arenas



```
import "arena"

func main() {
    a := arena.New()
    defer a.Free()

    x := a.New((*MyType)(nil)).(*MyType)
    s := a.NewSlice((*[]int)(nil), 100).([]int)
}
```



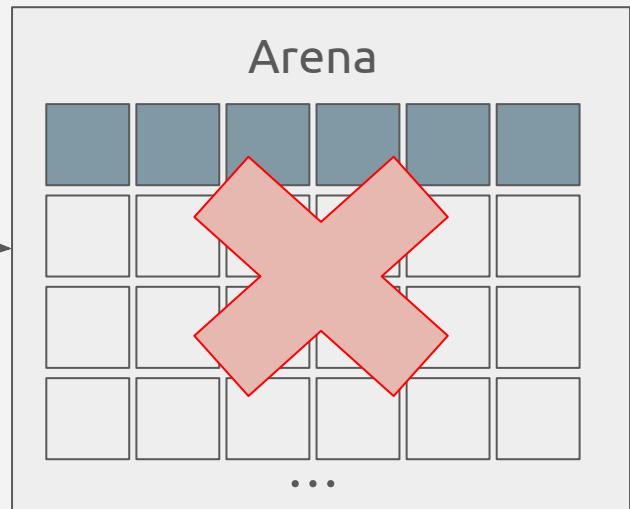
Arenas



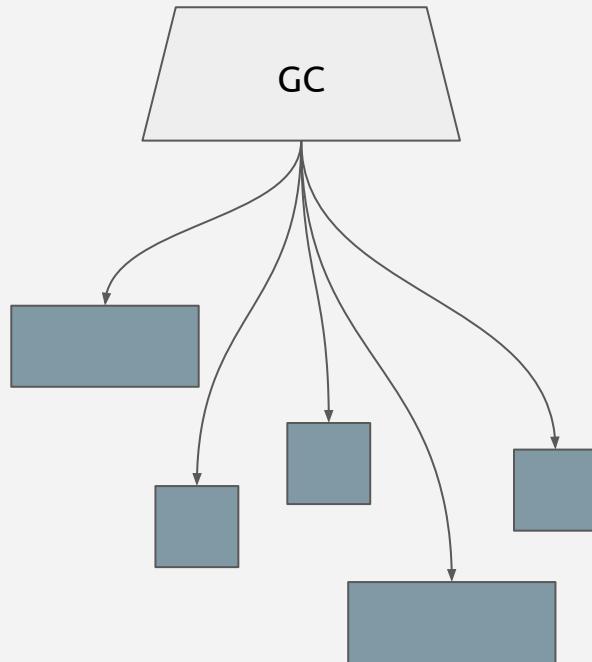
```
import "arena"

func main() {
    a := arena.New()
    defer a.Free() →

    x := a.New((*MyType)(nil)).(*MyType)
    s := a.NewSlice((*[]int)(nil), 100).([]int)
}
```

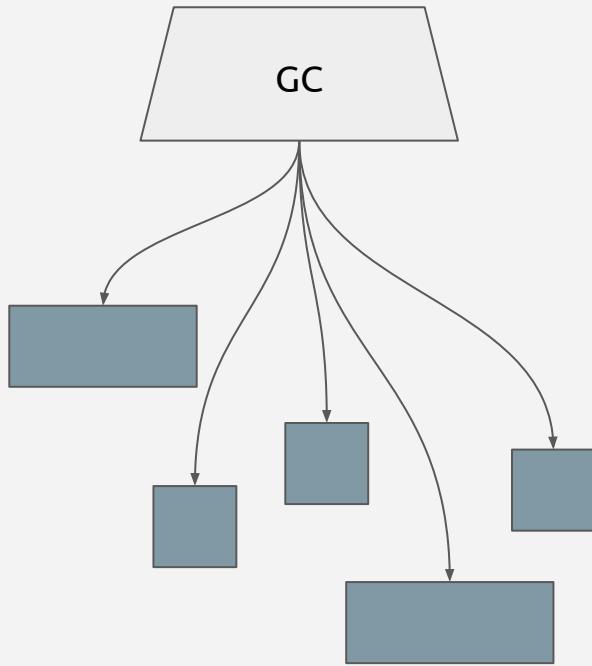


Alivia o garbage collector

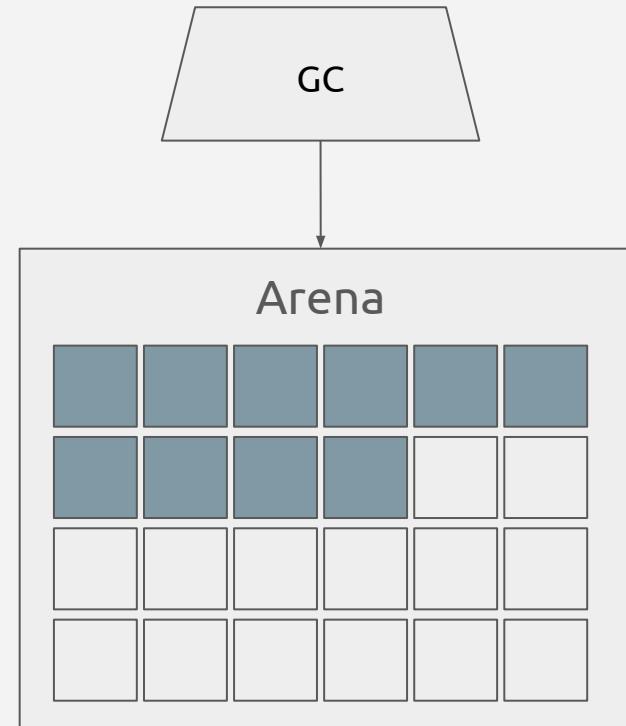


Sem arenas

Alivia o garbage collector

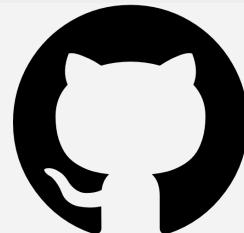


Sem arenas



Com arenas

Redes sociais



[reneepc](#)



[/in/reneepc](#)

