

# Fundamentos de Processamento Paralelo e Distribuído

canais

Referência: Principles of Concurrent and Distributed Programming  
(Second Edition) Addison-Wesley, 2006. Mordechai (Moti) Ben-Ari

Fernando Luís Dotti





- Comunicação entre processos  
—?

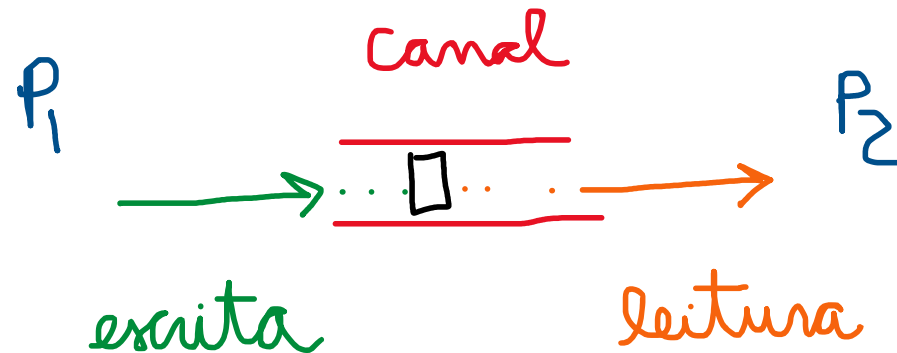
- Comunicação entre processos
  - canais (troca de mensagens)
  - memória compartilhada

# Canais de Comunicação

# Canais

## Processos

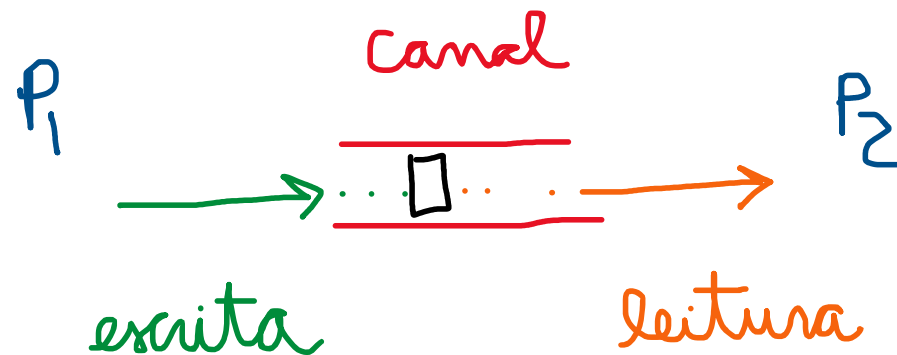
- têm canais comuns
- comunicam através deles:
  - escrita no canal; leitura do canal



# Canais

## Processos

- troca de mensagens:
  - escrita no canal ~ envio de mensagem
  - leitura do canal ~ recebimento de mensagem
  - mensagem ~ item escrito/lido



# Canais

## Características

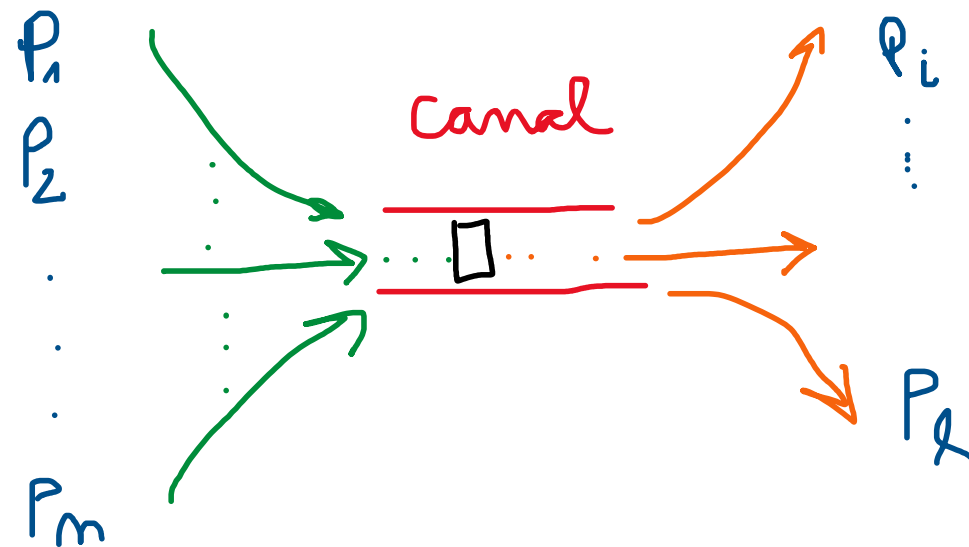
- tipados:  
int, string, ..., ou um record criado
- podem ser atomicamente  
escritos e lidos por processos
- semântica FIFO  
escrita e leitura preservam ordem
- síncrono ou assíncrono  
atenção!!!
- não determinismo



# Canais

- são atomicamente  
escritos e lidos por processos
  - diversos processos podem ler/escrever em um canal
  - escritas e leituras podem ser concorrentes
  - canal preserva consistência
  - operações concorrentes tem efeito em alguma ordem sequencial sobre o canal (interleaving)
  - não-determinismo das operações concorrentes

# Canais



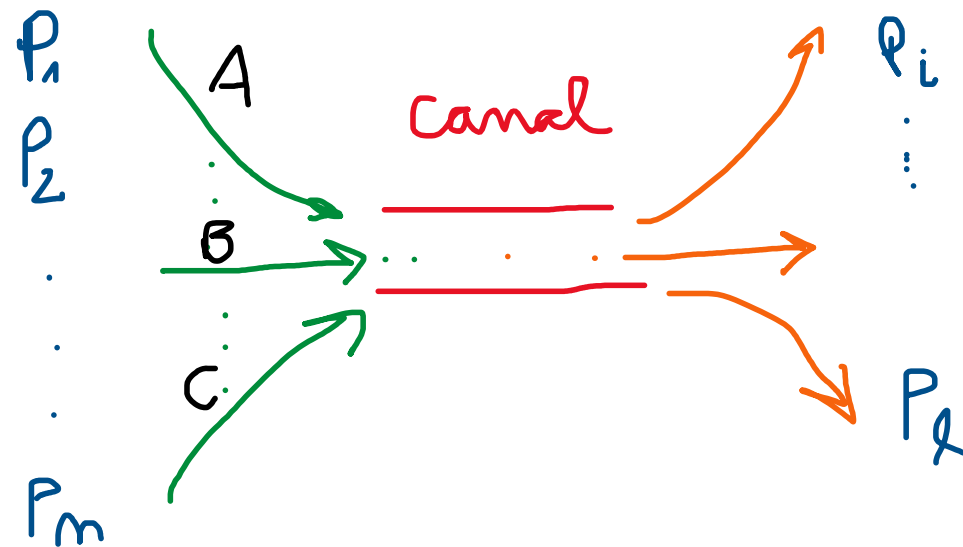
# Canais

## Assíncronos

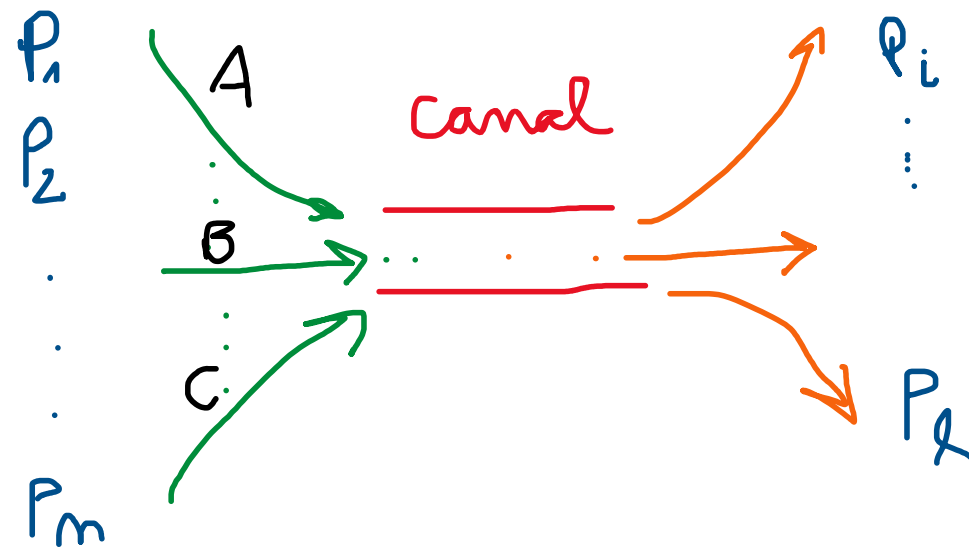
- bufferizados
  - leitor e escritor independentes, exceto
  - vazio: leitura depende de escrita
  - cheio: escrita depende de leitura

# Representação

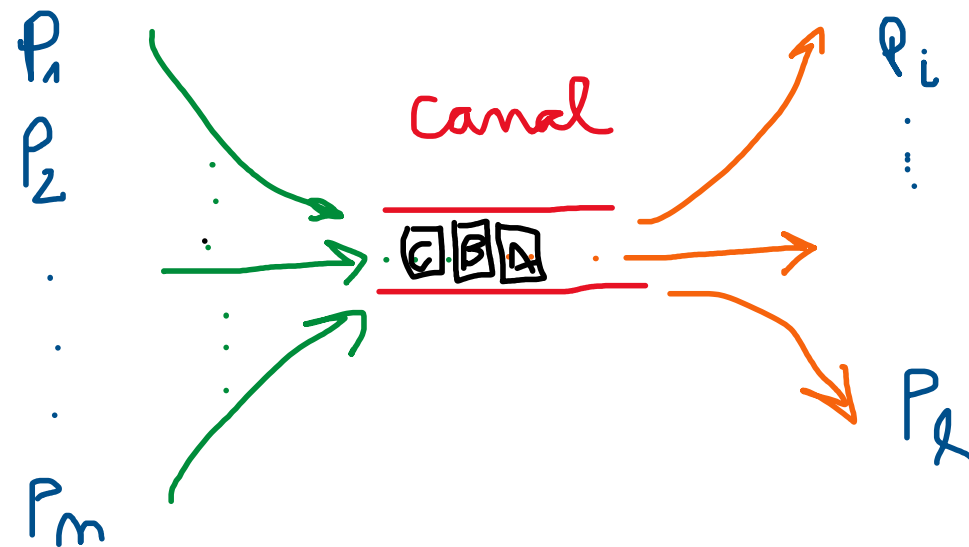
Assíncrono, tem buffer (N)



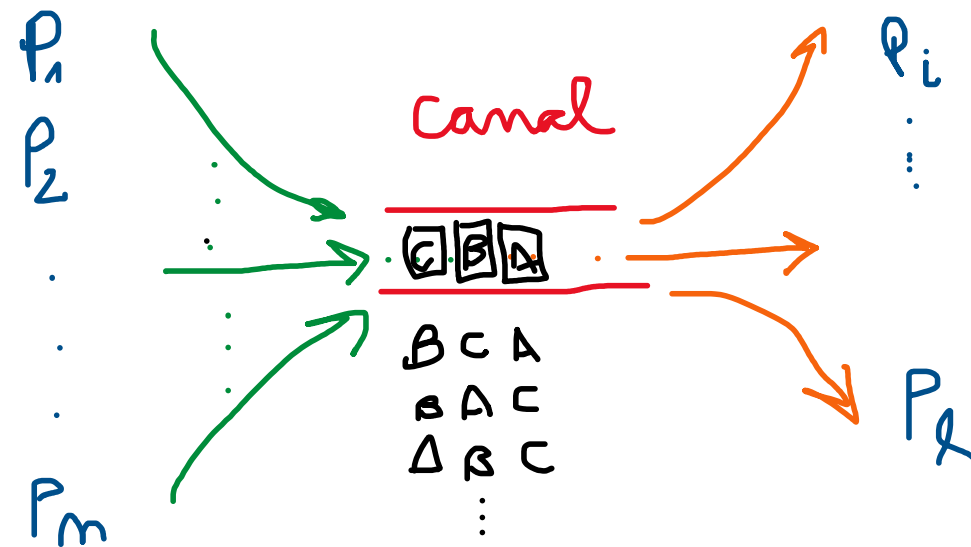
# Canais



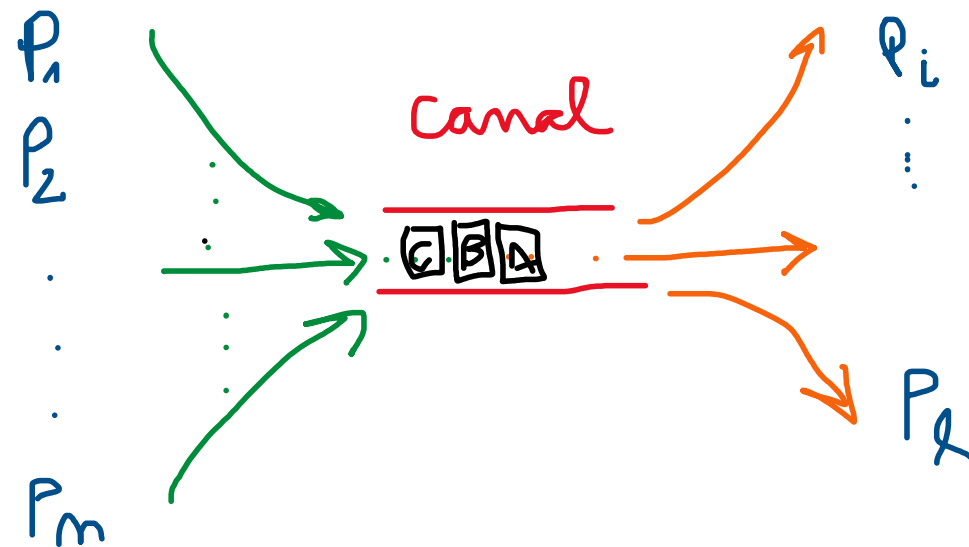
# Canais



# Canais

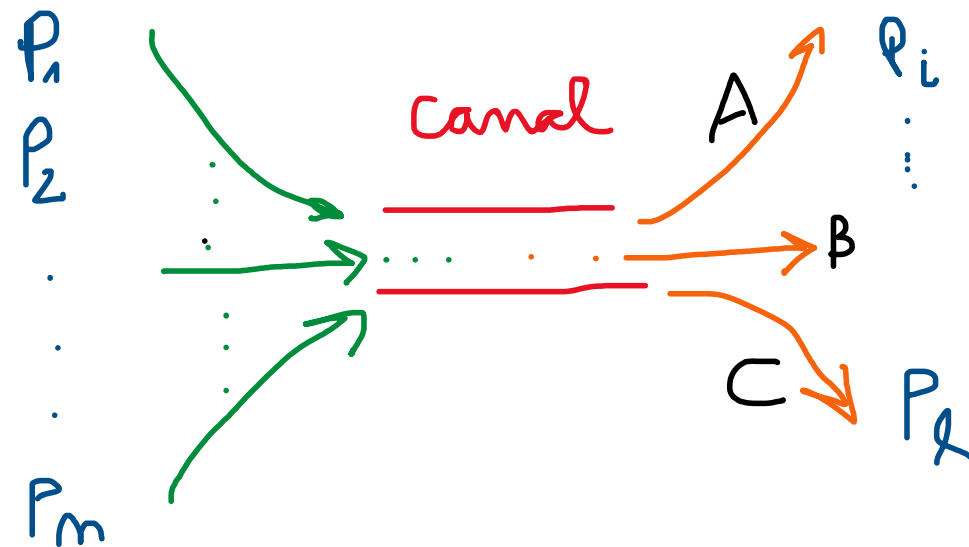


# Canais

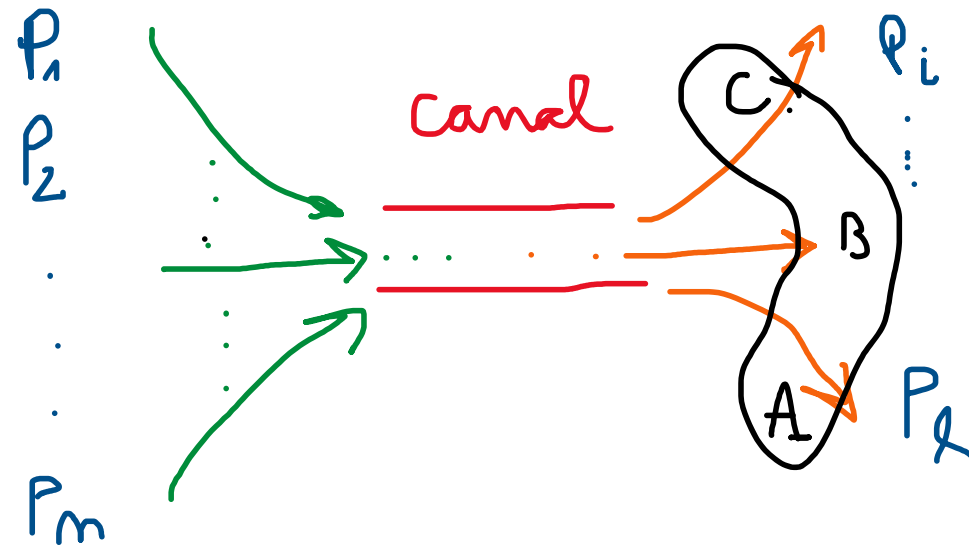




# Canais



# Canais



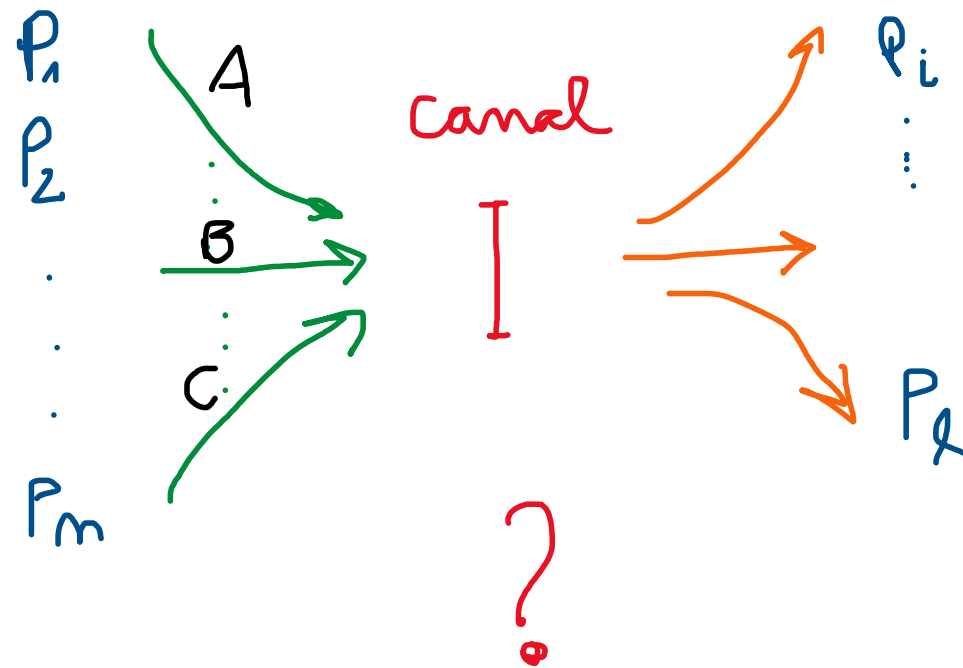
# Canais

## Sincronizantes

- não bufferizados (de tamanho 0)
  - leitor e escritor sincronizam-se na leitura/escrita

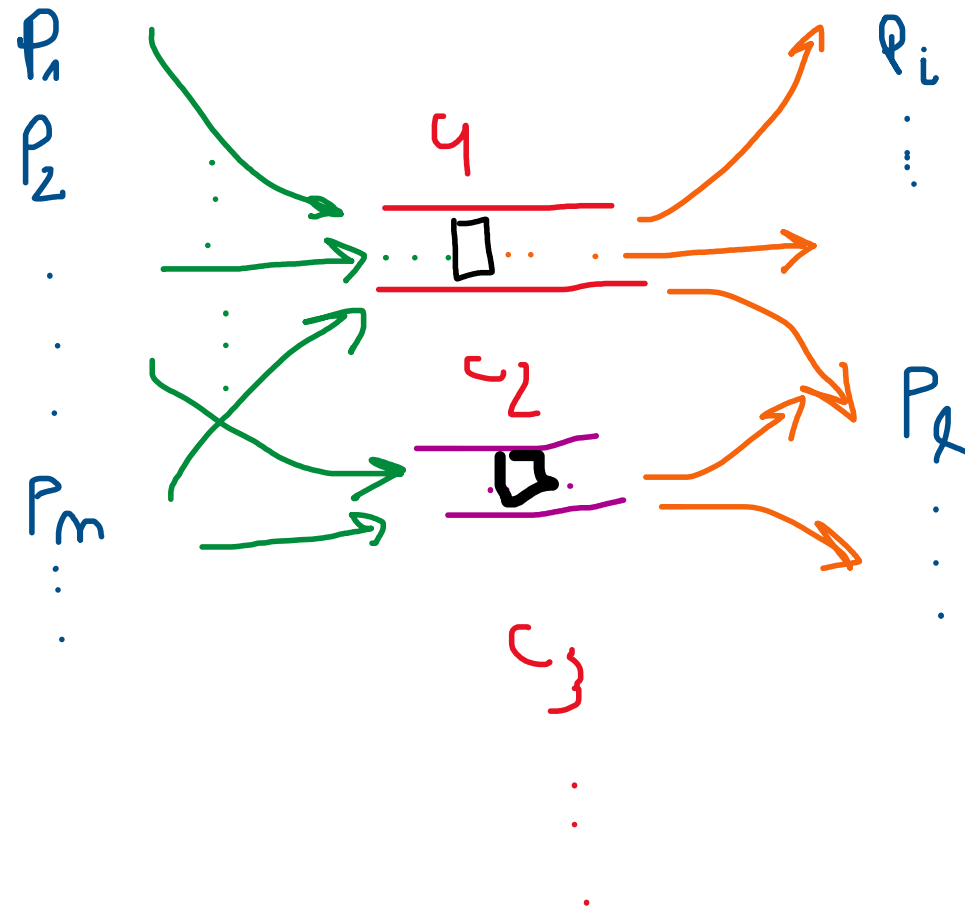
# Representação

Síncrono = tamanho 0



# Canais

- disposição de canais e processos é decisão de quem projeta o sistema concorrente



# Canais

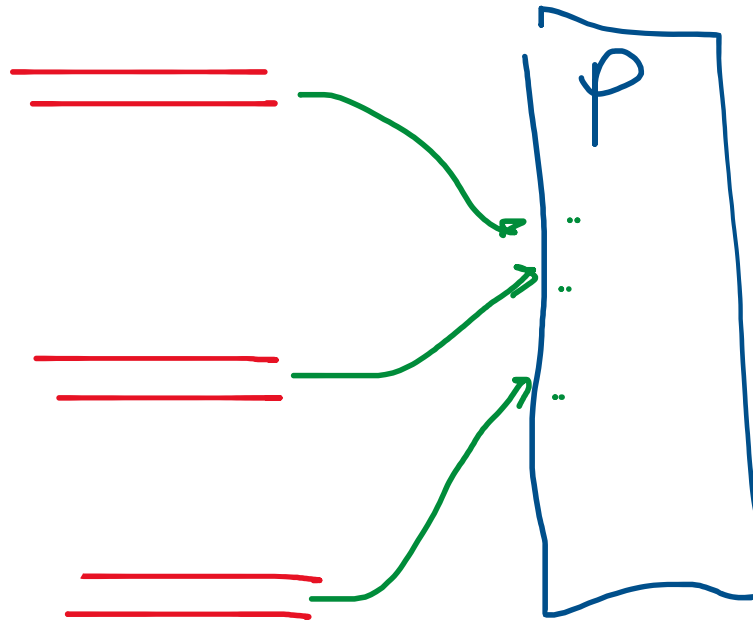
## Não determinismo

- processo segue conforme ação habilitada em *algum* de seus canais
- CSP

# Canais

## Não determinismo

- processo segue conforme ação habilitada em *algum* de seus canais



# Canais

- pensamos em **canais como interfaces dos processos**
  - "processo  $p_i$  recebe pelo canal  $cin_i$ "
  - "processo  $p_i$  responde pelo canal  $cout_i$ "
  - "processo  $p_i$  sinaliza processo  $p_j$  por canal  $c_{i,j}$ "
  - atribuímos "funções" aos canais



# Exemplificação

- escrita e leitura

```
package main

const N = 100
const tamBuff = 10

func fonteDeDados(saida chan int) {
    for i := 1; i < N; i++ {
        println(i, " -> ")
        saida <- i    // escreve no canal
    }
}

func destinoDosDados(entrada chan int) {
    for i := 1; i < N; i++ {
        v := <-entrada // le do canal
        println("                -> ", v)
    }
}

func main() {
    c := make(chan int, tamBuff)
    go fonteDeDados(c) // inicia processo concorrente, que usa c
    destinoDosDados(c) // chama procedimento passando mesmo c
}
```

# Exemplificação

- escrita e leitura

```
package main

const N = 100
const tamBuff = 10

func fonteDeDados(saida chan int) {
    for i := 1; i < N; i++ {
        println(i, " -> ")
        saida <- i    // escreve no canal
    }
}

func destinoDosDados(entrada chan int) {
    for i := 1; i < N; i++ {
        v := <-entrada // le do canal
        println("          -> ", v)
    }
}

func main() {
    c := make(chan int, tamBuff)
    go fonteDeDados(c) // inicia processo concorrente, que usa c
    destinoDosDados(c) // chama procedimento passando mesmo c
}
```



# Exemplificação

- escrita e leitura (mais de um leitor)

```
package main

const N = 100
const tamBuff = 0

func fonteDeDados(saida chan int, n int) {
    for i := 1; i < n; i++ {
        println(i, " -> ")
        saida <- i    // escreve i no canal saida
    }
}

func destinoDosDados(entrada chan int, n int) {
    for i := 1; i < n; i++ {
        v := <-entrada    // le do canal entrada, atribui a v
        println("        -> ", v)
    }
}

func main() {
    c := make(chan int, tamBuff)
    go fonteDeDados(c, N)    // fonte gera N
    go destinoDosDados(c, N/2) // cada destino consome N/2 ...
    destinoDosDados(c, N/2)
}
```

# Exemplificação

- escrita e leitura (mais de um leitor)

```
package main
```

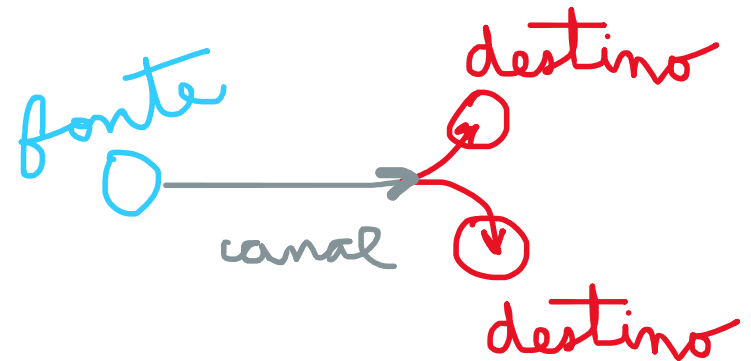
```
const N = 100
```

```
const tamBuff = 0
```

```
func fonteDeDados(saida chan int, n int) {  
    for i := 1; i < n; i++ {  
        println(i, " -> ")  
        saida <- i    // escreve i no canal saida  
    }  
}
```

```
func destinoDosDados(entrada chan int, n int) {  
    for i := 1; i < n; i++ {  
        v := <-entrada    // le do canal entrada, atribui a v  
        println("        -> ", v)  
    }  
}
```

```
func main() {  
    c := make(chan int, tamBuff)  
    go fonteDeDados(c, N)    // fonte gera N  
    go destinoDosDados(c, N/2) // cada destino consome N/2 ...  
    destinoDosDados(c, N/2)  
}
```



# Exemplificação

Concorrência em Go:  
canais e não **determinismo**

```
package main
import "fmt"
func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
        case c <- x: // escreve x em c
            x, y = y, x+y
        case <-quit: // consigo ler de quit ?
            fmt.Println("quit")
            return
        }
    }
}

func main() {
    c := make(chan int)
    quit := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-c) //
        }
        quit <- 0
    }()
    fibonacci(c, quit)
}
```

# Exemplificação

## Concorrência em Go: canais e não **determinismo**

- Escolha não-determinística entre duas entradas:
  - Para func fibonacci tanto escrita em c como leitura de quit podem acontecer a cada vez que entra no select
- Declaração e disparo de go rotina
  - Main prossegue ...

0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
quit

```
package main
import "fmt"
func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
        case c <- x: // escreve x em c
            x, y = y, x+y
        case <-quit: // consigo ler de quit ?
            fmt.Println("quit")
            return
        }
    }
}


func main() {
    c := make(chan int)
    quit := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-c) //
        }
        quit <- 0
    }()
    fibonacci(c, quit)
}
```

# Exemplificação

## Concorrência em Go: canais e **não determinismo**

- **Select/case**  
**como guardas**  
**não determinísticas**  
**usando leitura de**  
**canais**
- **Note que tick e**  
**boom são canais**  
**onde os eventos**  
**acontecem, conforme**  
**os tempos definidos,**  
**disparados pela**  
**biblioteca time**

```
tick.  
tick.  
tick.  
tick.  
tick.  
tick.  
BOOM!
```



```
package main  
  
import (  
    "fmt"  
    "time"  
)  
  
func main() {  
    tick := time.Tick(100 * time.Millisecond)  
    boom := time.After(500 * time.Millisecond)  
    for {  
        select {  
        case <-tick:  
            fmt.Println("tick.")  
        case <-boom:  
            fmt.Println("BOOM!")  
            return  
        default:  
            fmt.Println("  .")  
            time.Sleep(50 * time.Millisecond)  
        }  
    }  
}
```