Programação Concorrente e Paralela

Comunicação por Troca de Mensagens

Noemi Rodriguez

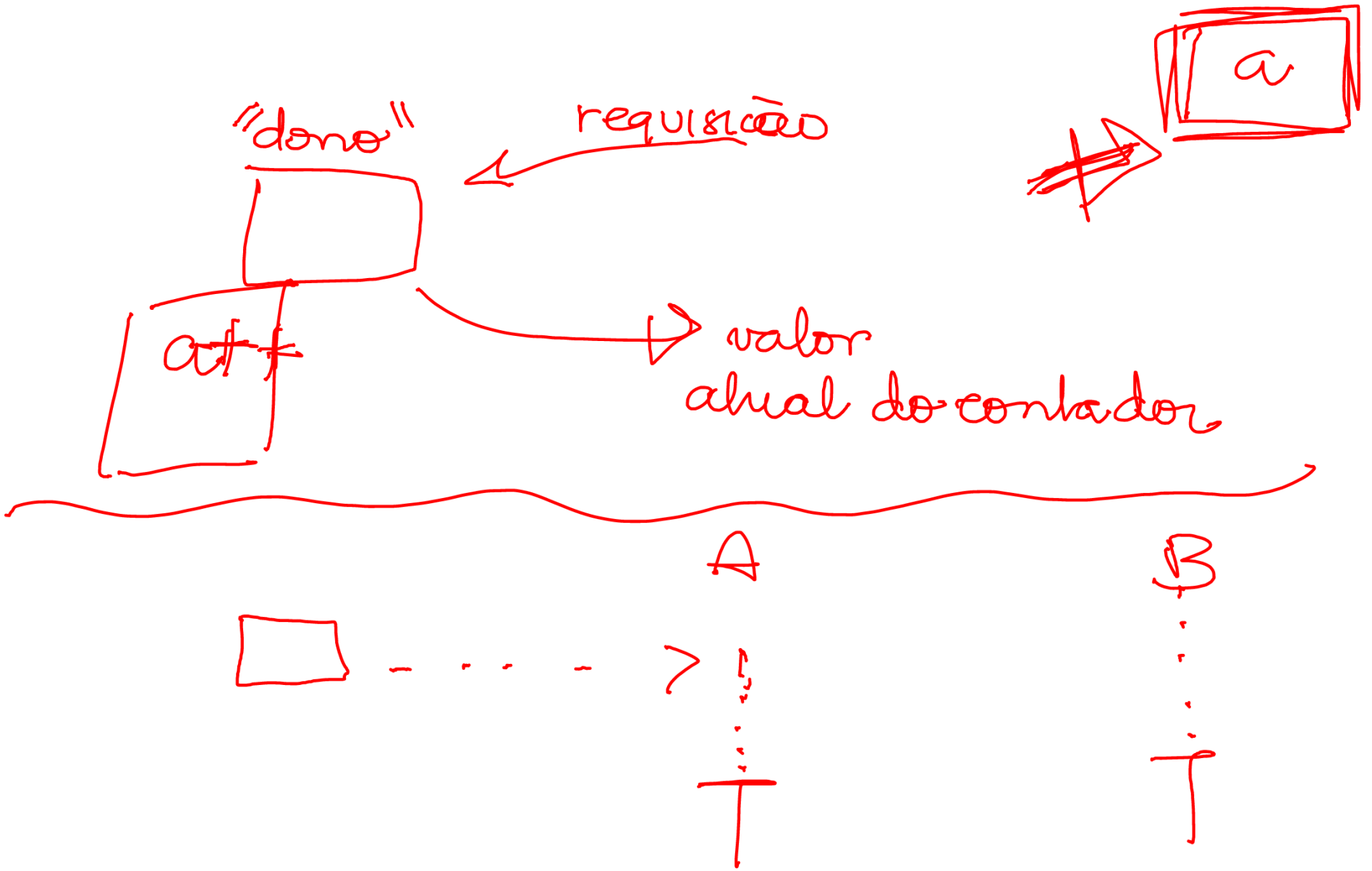
2016

Troca de Mensagens

- forma básica de comunicação em ambientes de memória distribuída
  - outras camadas podem ser construídas sobre trocas de mensagens básicas
- mas também utilizada em ambientes de memória compartilhada
  - grande guerra filosófica...

TM em concorrência e paralelismo

- troca de mensagens em "aplicacões fechadas", entre fluxos de controle fortemente acoplados


"dono"
requisição
a
valor
atual do contador
A
B

A

receive (B, ---)

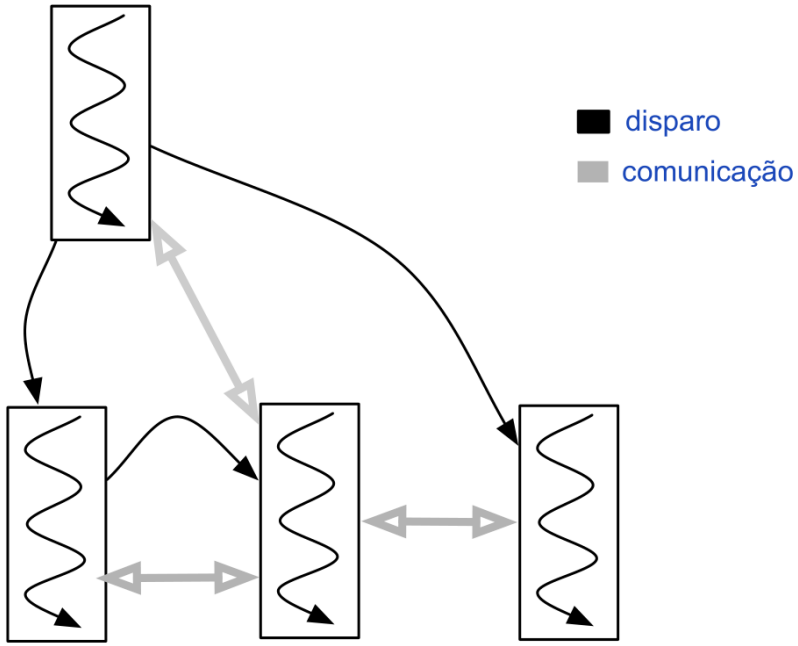
B

receive (A, ...)

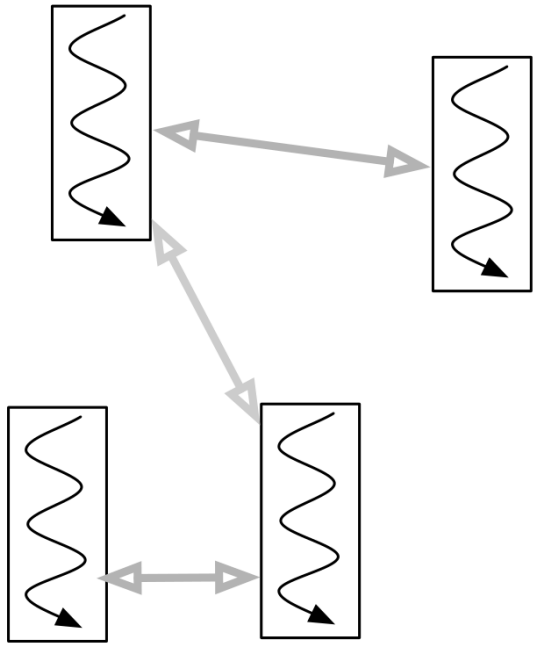
DEADLOCK

Troca de Mensagens

aplicações fechadas: comunicação entre fluxos com "antepassados" em comum



aplicações abertas: comunicação entre fluxos arbitrários



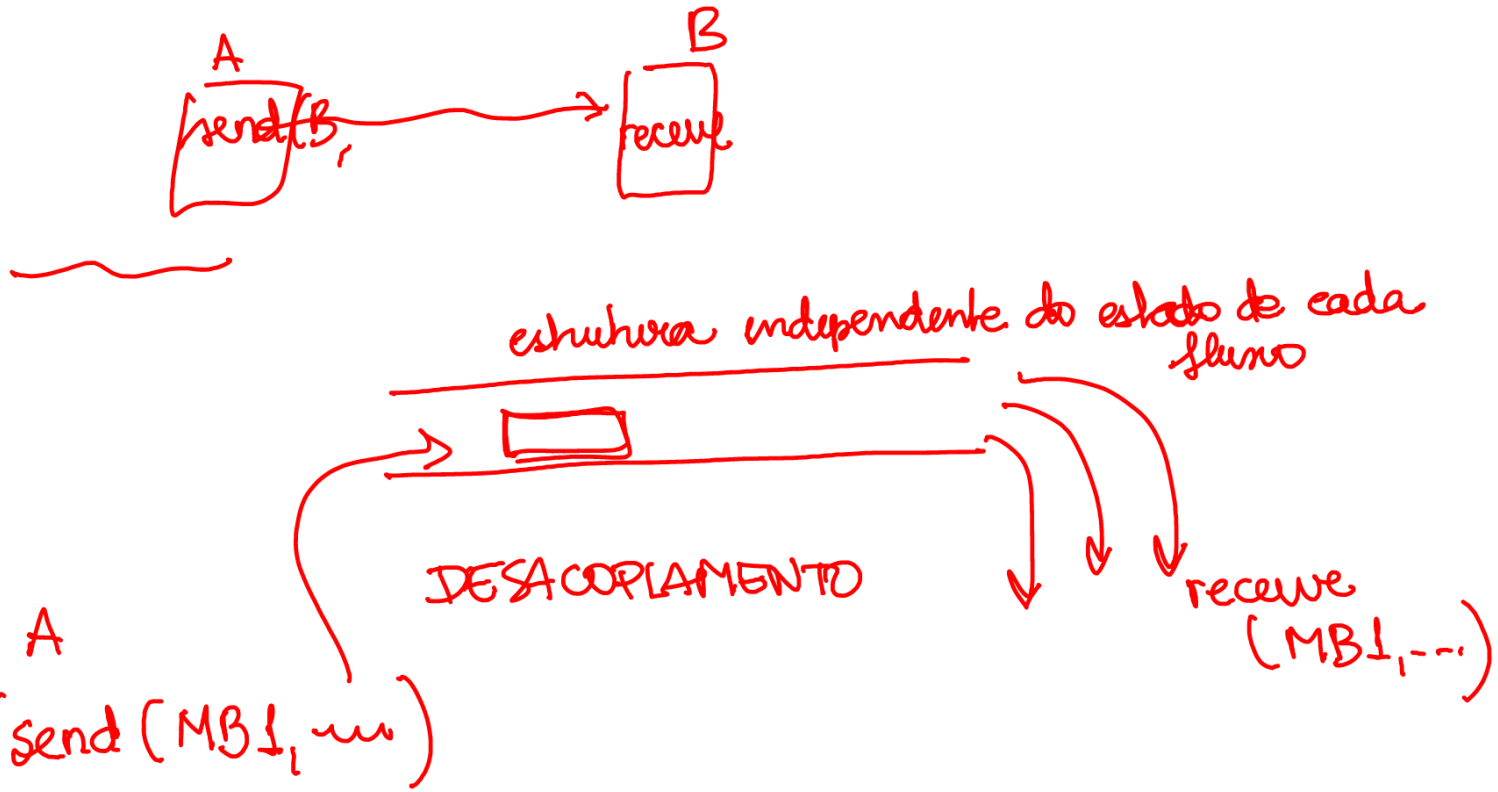
Características

- identificação de pares
- semântica de envio e recebimento
- formatos e tipos de dados
- operações coletivas?

Identificação de Pares

- problema grande em sistemas abertos/distribuídos
- em sistemas fechados (aplicações): atribuição de identificadores
  - tipicamente inteiros de 0 a $n-1$

```
myrank = getrank();
if (myRank == 0)
  send (1, ...);
else if (myrank ==1) {
  receive(0,...);
  printf ("recebi msg de 0!\n");
}
```


A
B
receive
estrutura independente do estado de cada
lugar
DESACOPLAMENTO
receive
(MB1,---)
A
(send (MB1,---)

Identificação de Pares

canais

- *canais* ou *mailboxes* podem ajudar a organizar a comunicação
  - envio para "qualquer um de um grupo"
  - trocas de mensagens de diferentes tipos

```
  send ("trabalhadores", ...);
...
  receive("trabalhadores",...);
```

Semântica de envio e recebimento

```
send (destino, blablabla);
/* o que podemos afirmar nesse ponto de execução?!? */
```

alternativas

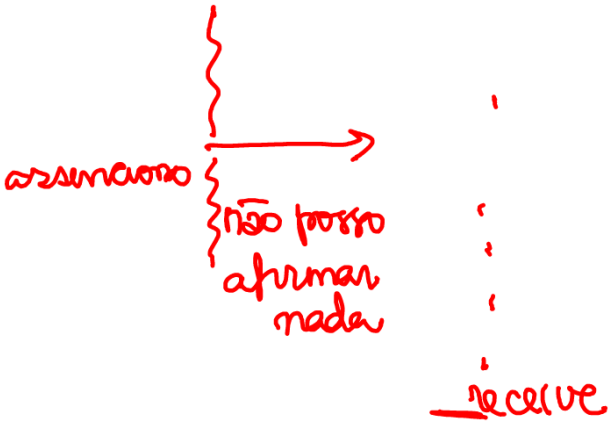
- envio síncrono (ou bloqueante): controle retorna depois que mansagem é entregue ao destino
- envio assíncrono (ou não bloqueante): controle retorna imediatamente
- recebimento síncrono (ou bloqueante): controle retorna apenas quando mensagem solicitada já foi recebida
- recebimento assíncrono: ?!? diferentes interpretações

Semântica de envio e recebimento

relação com bufferização, determinismo e oportunidades de concorrência, ...

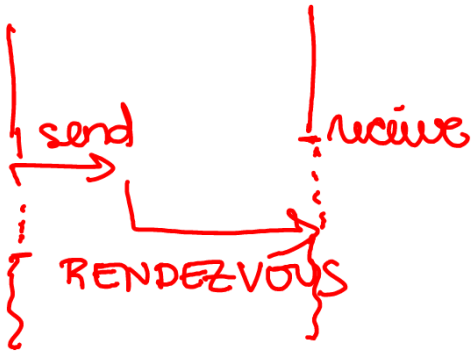
- envio síncrono (ou bloqueante): controle retorna depois que mansagem é entregue ao destino
- envio assíncrono (ou não bloqueante): controle retorna imediatamente
- recebimento síncrono (ou bloqueante): controle retorna apenas quando mensagem solicitada já foi recebida
- recebimento assíncrono: ?!? diferentes interpretações

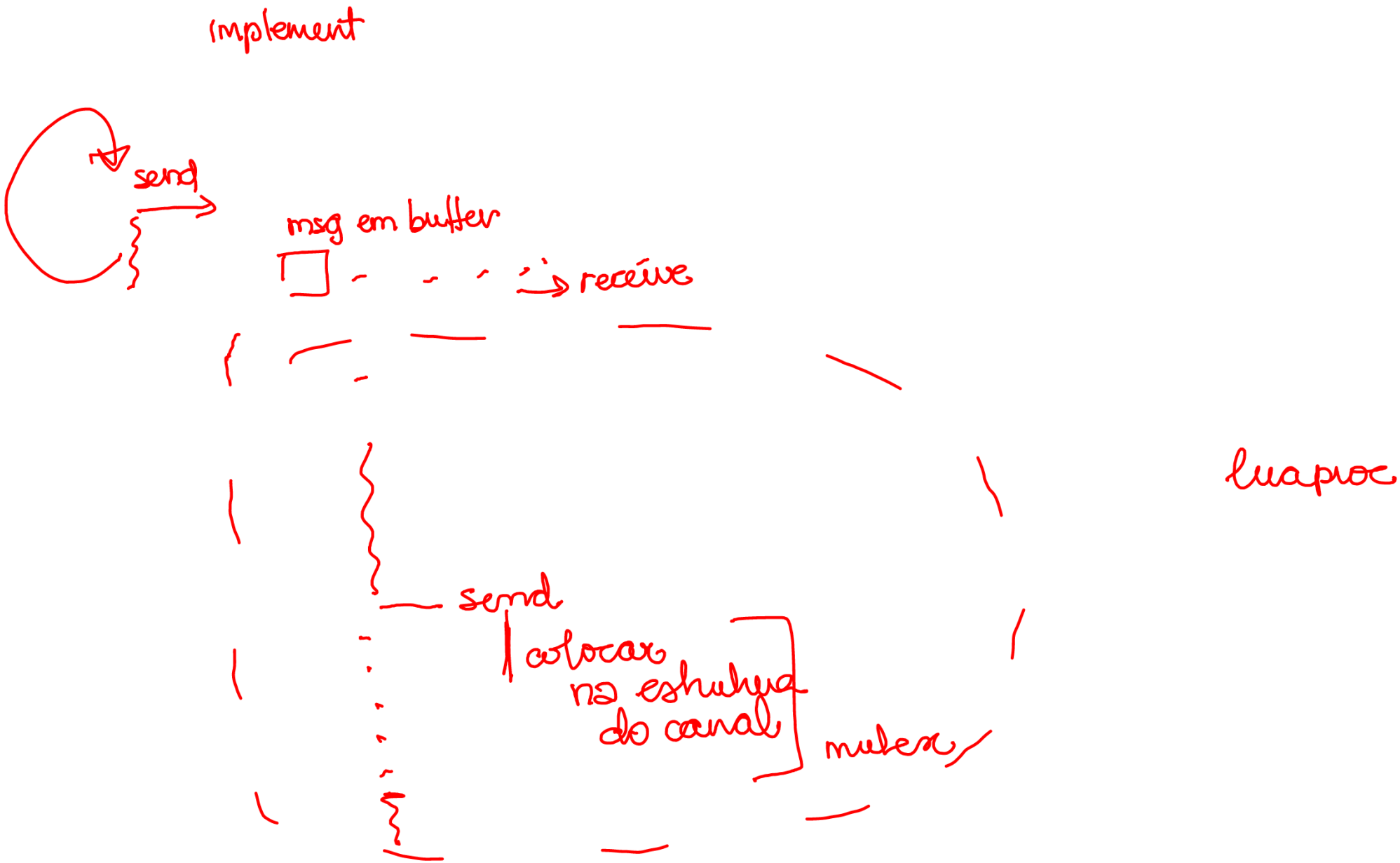
ENVIO



criar pontos explícitos

simplicidade
sincronização implícita



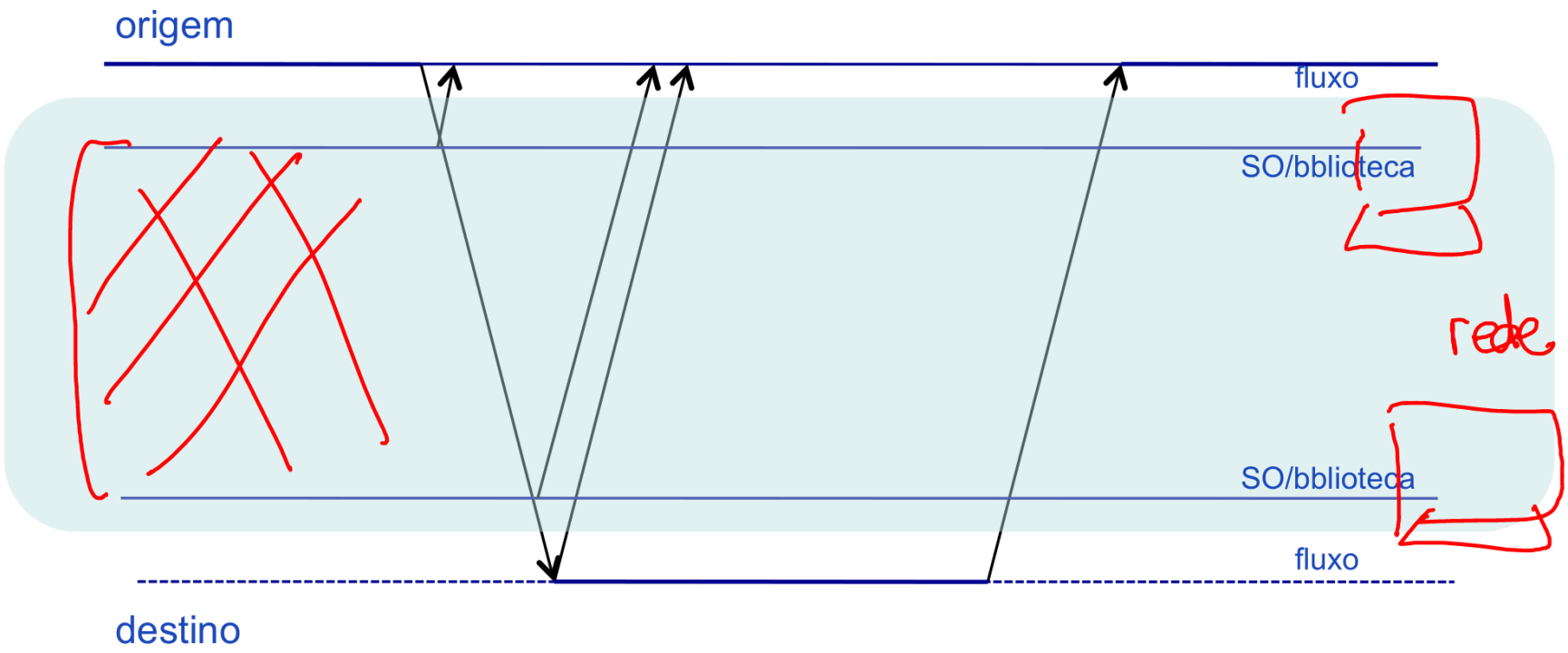

implement
send
msg em buffer
receive
send
colocar na estrutura do canal
mutex
luaproc

sistemas de filas de comunicação

serviços de nuvem - Amazon, Azure. . . .

AMQP

envio e recebimento: alternativas



recebimento assíncrono

①

loop [receive (~, timeout)
outras atividades]

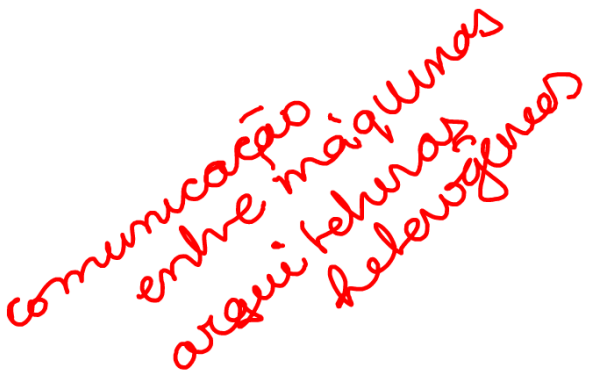
②

recebimento por callbacks

[on mousechck()
≡
on receive (~,)
~
~
~]

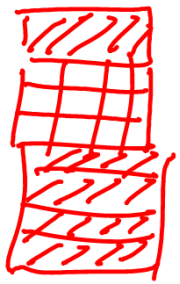
Formatos e tipos de dados

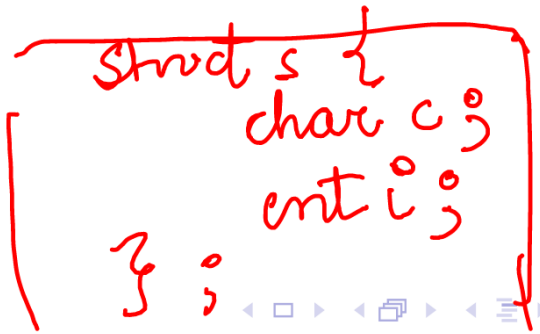
- valores tipados
  - que tipos têm suporte?
- ... sequências de bytes
  - interfaces de sistemas operacionais



problemas de representação

- arquiteturas heterogêneas podem demandar conversões entre formatos



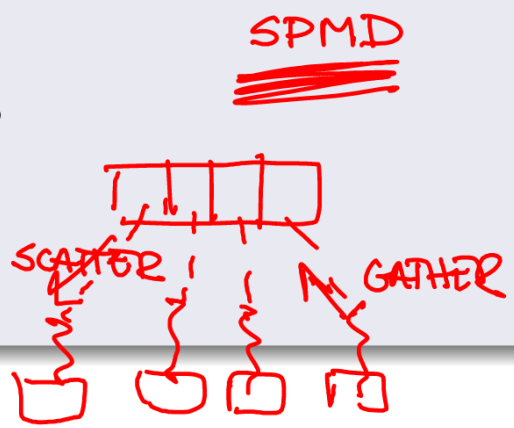


Operações coletivas

- modelo SPMD frequentemente requer operações que envolvem grupos de participantes
  - já vimos operações de *redução* no openMP

operações coletivas comuns

- redução
- barreira
- distribuição de dados
  - broadcast
  - scatter
  - gather



Operações coletivas

Grupos

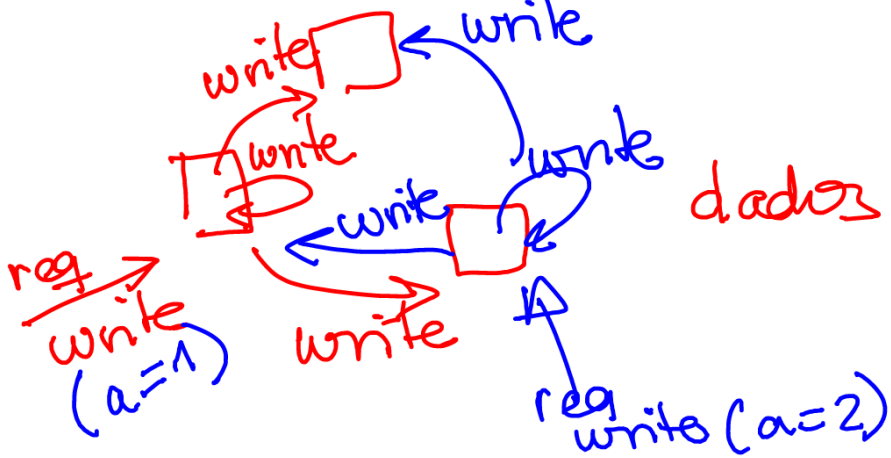
- operações coletivas ocorrem dentro de um grupo de fluxos
- outra visão de *comunicação em grupo*
  - conceito mais ligado a sistemas distribuídos

- como definir grupo?
- ordenação de mensagens

GRUPOS em sistemas distribuídos

comunicação:
broadcast em grupo

servidor replicado
grupo das réplicas



dados

GRUPOS em aplicações SPMD

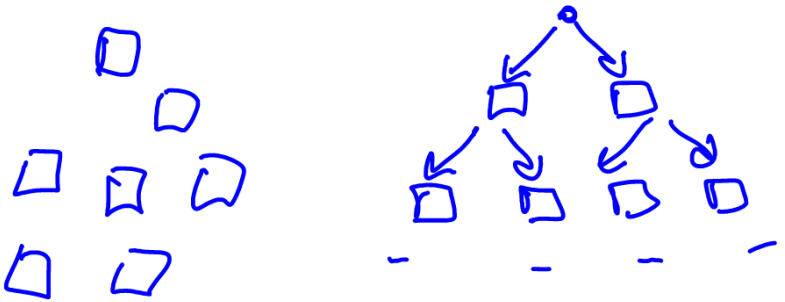
mesmo código

broadcast (Raiz, ...)

todos têm que executar chamada com a mesma raiz

Ø

sugere-se que a implementação faça uma dist em árvore

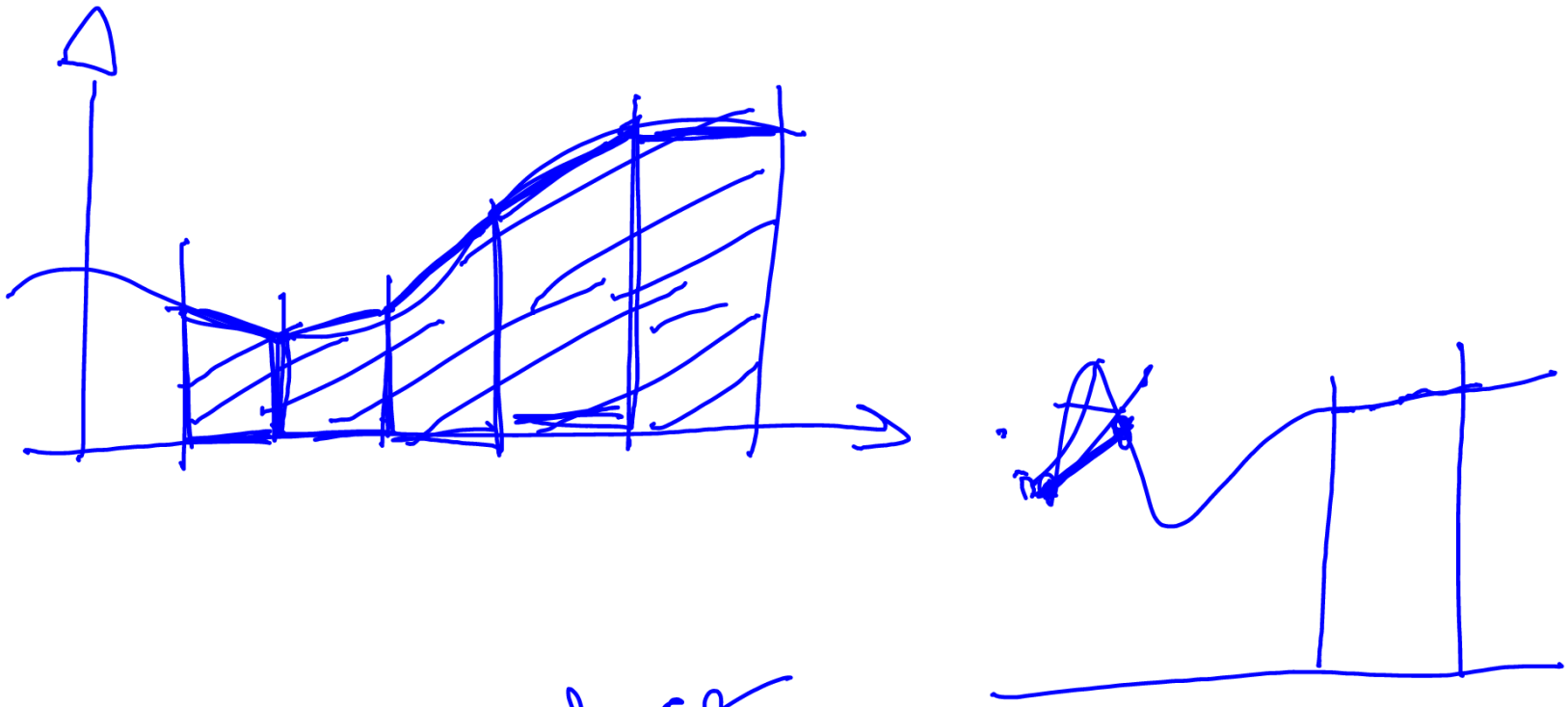


Exemplo

MPI

- exemplos de *An Introduction to Parallel Programming*, de Peter Pacheco.

Exemplo: Aproximação por Trapézios



Message Passing Interface

#mpiron <meuprog> parametros

Exemplo: Aproximação por Trapézios

```
int main(void) {
   int n = 1024;
   double a = 0.0, b = 3.0, h, local_a, local_b;
   ...
   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
   h = (b-a)/n;  local_n = n/comm_sz;
   local_a = a + my_rank*local_n*h; local_b = local_a + local_n*h;
   local_int = Trap(local_a, local_b, local_n, h);
   if (my_rank != 0) {
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   } else {
      total_int = local_int;
      for (source = 1; source < comm_sz; source++) {
         MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         total_int += local_int;
      }
   }
   if (my_rank==0) /* mostra resultados */ ...
```

Exemplo: Aproximação por Trapézios (segunda versão)

com operações coletivas!

```
MPI_Init(NULL, NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
Get_input(my_rank, comm_sz, &a, &b, &n);
h = (b-a)/n;           /* h is the same for all processes */
local_n = n/comm_sz;  /* So is the number of trapezoids  */
local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
local_int = Trap(local_a, local_b, local_n, h);
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

"grupo" participante de ops coletivas é chamado de COMUNICADOR.

Exemplo: Aproximação por Trapézios (segunda versão)

```
void Get_input(int my_rank, int comm_sz, double* a_p, double* b_
      int* n_p) {

   if (my_rank == 0) {
      printf("Enter a, b, and n\n");
      scanf("%lf %lf %d", a_p, b_p, n_p);
   }
   MPI_Bcast(a_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(b_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(n_p, 1, MPI_INT, 0, MPI_COMM_WORLD);
}
```

Operações Coletivas em MPI

- barreiras implícitas
- grupos implantados por *comunicadores*

Visão estática do programa

- tipicamente um ou dois processos por "nó"
  - como tratar máquinas com vários núcleos?
- dificuldades com ambientes dinâmicos

outro exemplo

dlang

Linguagem D

- proposta para o "nicho" C/C++
- ... com mais garantias na compilação
- compartilhamento de memória com restrições

exemplo: produtor/consumidor em D

```
import std.algorithm, std.concurrency, std.stdio;

void main() {
    enum bufferSize = 1024 * 100;
    auto tid = spawn(&fileWriter);
    // Read loop
    foreach (immutable(ubyte)[] buffer; stdin.byChunk(bufferSize)) {
        send(tid, buffer);
    }
}

void fileWriter() {
    // Write loop
    for (;;) {
        auto buffer = receiveOnly!(immutable(ubyte)[])();
        tgt.write(buffer);
    }
}
```

criação da 1 thread

identificador da thread destino