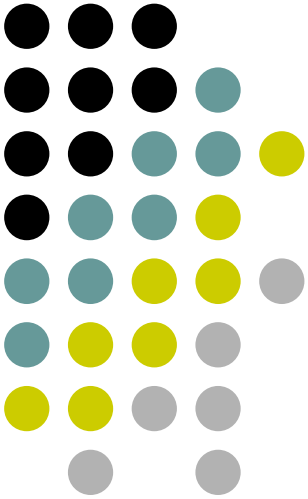
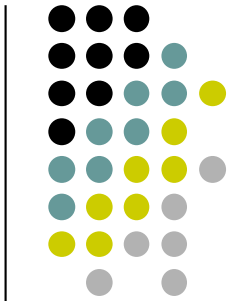
Sistemas Distribuídos

Módulo 4 – Modelo Cliente/Servidor

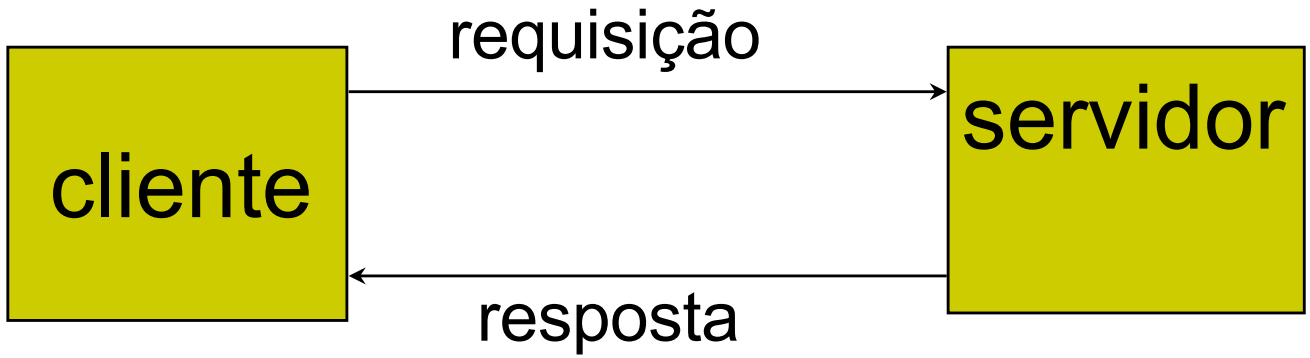


Introdução



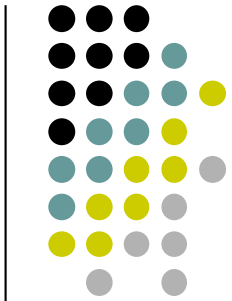
- É composto por um grupo de processos (servidores) que oferecem serviços a outros processos (clientes).
- Protocolo genérico de comunicação entre o cliente e o servidor: solicitação/resposta

Protocolo Requisição/Resposta



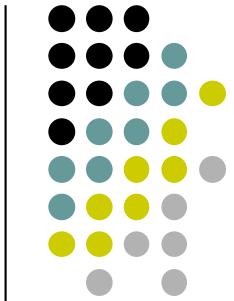
- Protocolo sem conexão
  - Geralmente 3 camadas: física, enlace, requisição/resposta.

Protocolo Requisição/Resposta



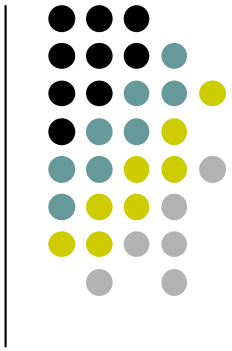
- Obs: as camadas 3 (rede) e 4 (transporte) do modelo OSI da ISO, que tratam do roteamento e da confiabilidade da transmissão, não são necessárias mas são desejáveis
- Vantagem sobre o modelo OSI da ISO:
  - eficiência (menor número de camadas)

Comunicação



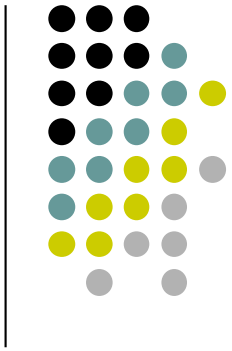
- A comunicação em um modelo cliente servidor pode ser implementada por unicamente dois serviços:
  - *envio de mensagens*:
    - send (dest, &buffer)
  - *recepção de mensagens*:
    - receive (dest, &buffer)

Registro/Lookup



- Registro do servidor
  - Antes de entrar em operação, o processo servidor se registra no sistema, enviando para o mesmo os seus dados de identificação
- Lookup do cliente
  - O cliente só pode utilizar serviços de um servidor ativo, ou seja, que tenha se registrado

Processo Servidor



```
main()
{
  while(1)
  {
    receive(ANY, &mensagem);
    switch (mensagem.tipo)
    {
     case CREATE:cr_proc(&mensagem,&resp);
                 break;
     /* outros tratamentos */
    }
    send(mensagem.origem, &resp);
  }
}
```

Recebe mensagem

Trata mensagem

Envia resposta

Processo cliente

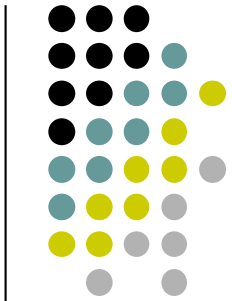
```
main()
{
  mensagem.origem = getpid();
  mensagem.destino = PID_SERVIDOR;
  mensagem.tipo = CREATE;
  /* preenche os outros campos da mensagem */
  send(PID_SERVIDOR, &mensagem);
  receive(PID_SERVIDOR, &resp);

 }
```

Envia mensagem

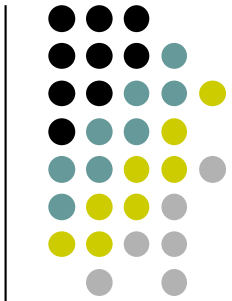
Recebe resposta

Problemas a serem tratados

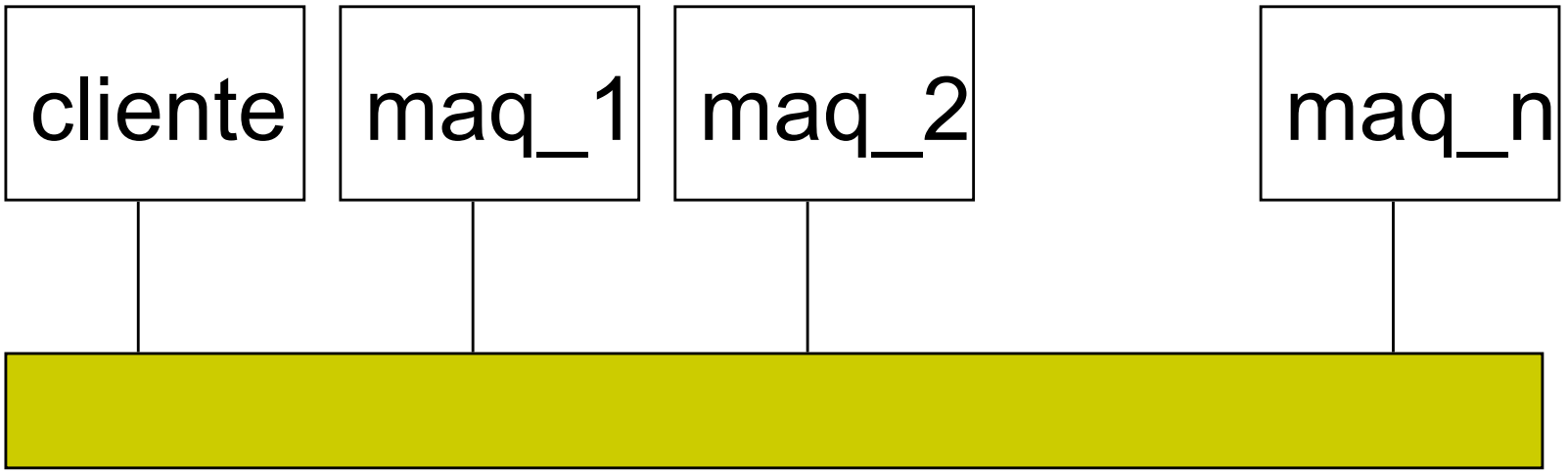


- Em que máquina está o servidor? (endereçamento)
- Que estrutura deve ter o servidor? (no exemplo, é monolítico, pode ser multi-threaded)
- Qual a semântica do envio e recepção de mensagens (blocante, não-blocante)?

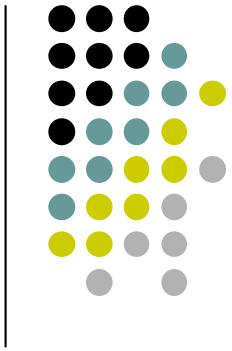
Endereçamento do Servidor



- Problema: como conhecer o endereço do servidor?

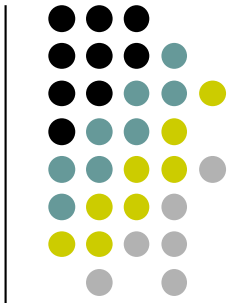


Endereçamento
Soluções



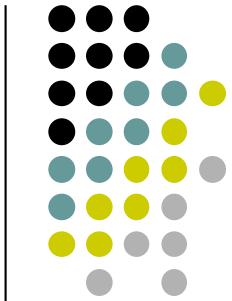
1. Endereçar máquinas: associar estaticamente as máquinas aos servidores e colocar na mensagem o endereço da máquina
2. Endereçar processos: colocar na mensagem a identificação do processo servidor

Endereçamento do Servidor
Solução 1: Utilizar endereço MAC



- Consiste em se definir de maneira estática em que máquinas estão quais servidores.
- Ao entrar no ar, o cliente tem acesso a esta configuração estática.
- Quando desejar solicitar um serviço, o cliente coloca no cabeçalho da mensagem o endereço numérico que é o endereço físico da máquina onde o processo servidor se executa

Endereçamento do Servidor Solução 1: Utilizar endereço MAC

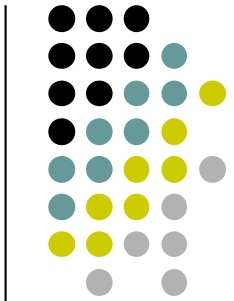


- Desvantagens:
  - O sistema operacional deve abrir a mensagem para identificar para qual servidor é endereçada a mensagem que chegou.
    - Aumento do ovehead de tratamento da mensagem

Solução 2: Endereçar processos

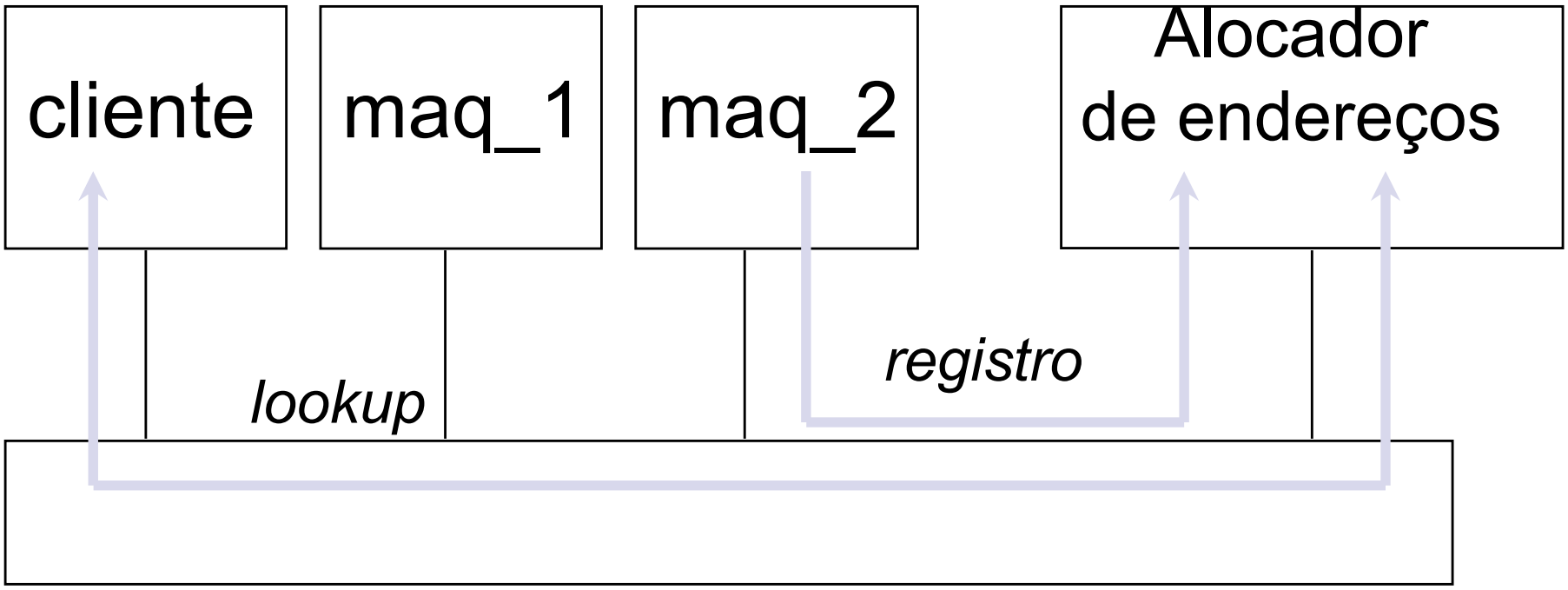
- Colocar na mensagem a identificação do processo ao qual se deseja enviar a mensagem.
  - nomes do tipo : <máquina.processo>. Identificação não transparente; não suporta migração nem servidores múltiplos
  - nomes numéricos únicos: o número do processo é único para todo o sistema distribuído.
    - *Solução normalmente utilizada*

Endereçamento por nomes únicos



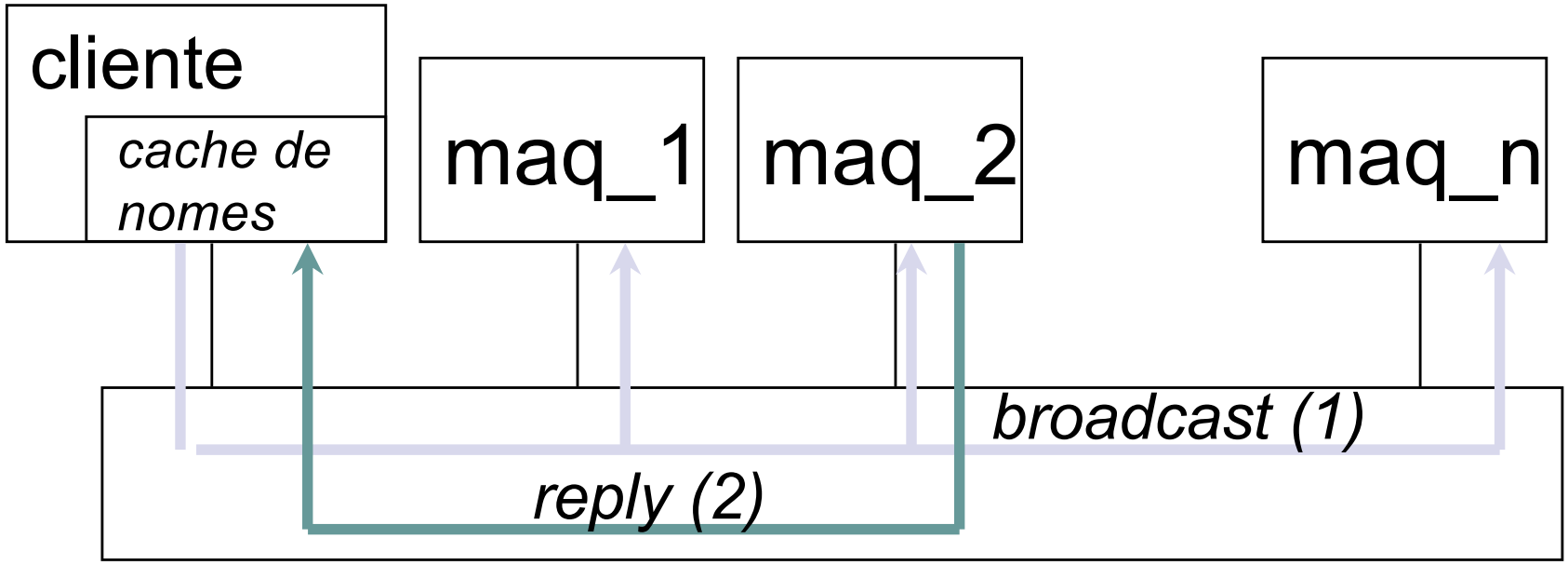
- A atribuição de nomes numéricos únicos necessita de um processo para decidir que nome deve ser atribuído a que processo.
- Como fazer o mapeamento processo x nome?
  - processo alocador de endereços centralizado
  - escolha randômica de nomes
  - servidor de nomes
  - Auxílio de hardware específico

Alocador Centralizado

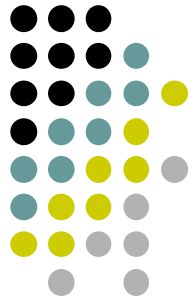


Escolha randômica de nomes

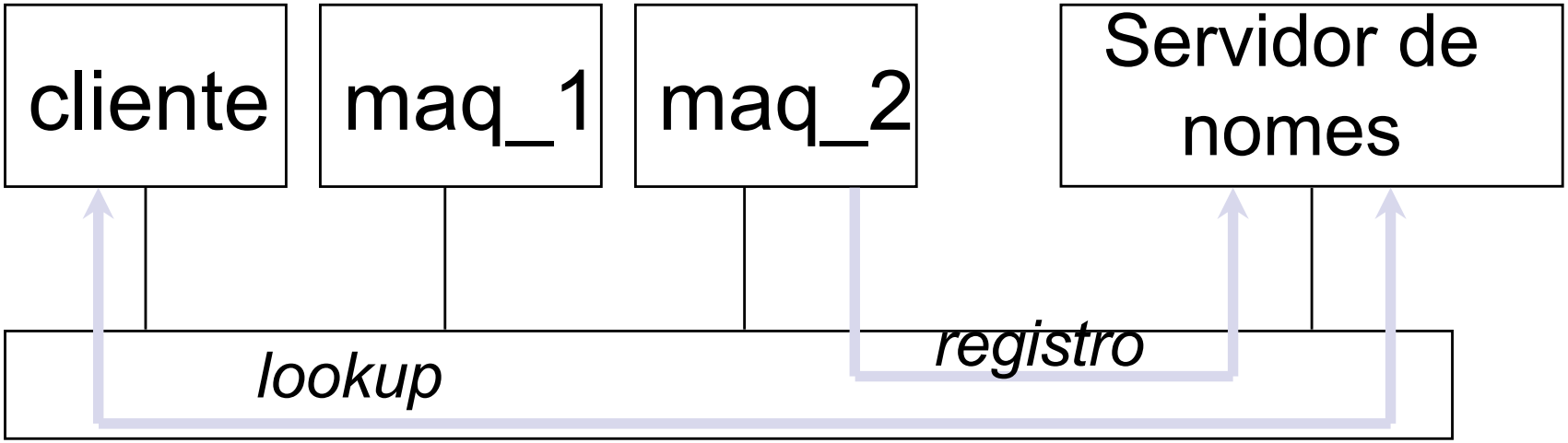
- Cada processo obtem seu identificador único (nome numérico) randômicamente a partir de um espaço de endereçamento grande (ex: 64 bits).
- Problema: Em que máquina está o processo?



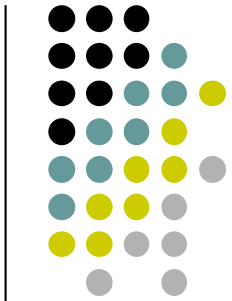
Servidor de nomes



- A cada serviço é atribuído um nome ascii. Um processo chamado servidor de nomes é o responsável pela manutenção da relação nome do serviço x endereço
- O servidor de nomes pode ser centralizado ou distribuído



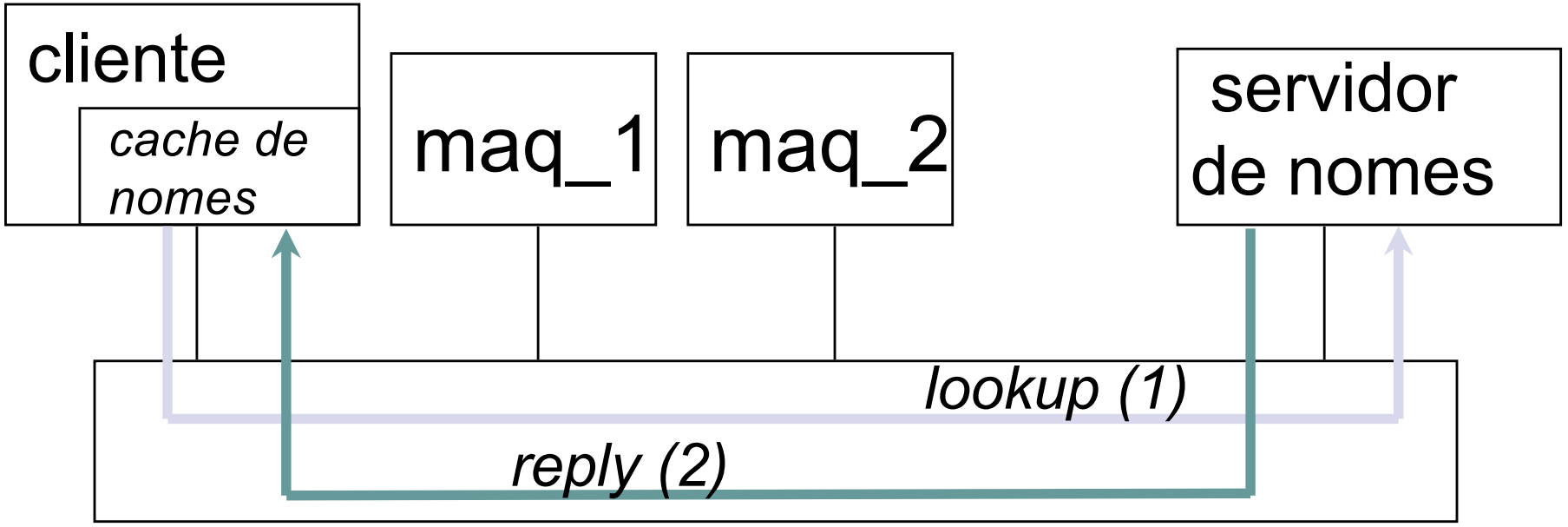
Auxílio de hardware de rede



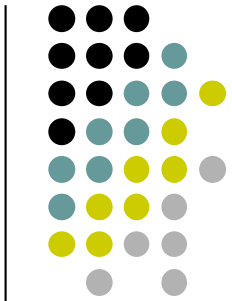
- Cada chip de interface de rede mantem a relação endereço do processo x endereço da máquina. O próprio chip verifica se o processo se encontra na sua máquina
- Solução cara !!!

Abordagem mais utilizada

- Servidor de nomes (centralizado ou por subconjuntos de máquinas) e caches locais

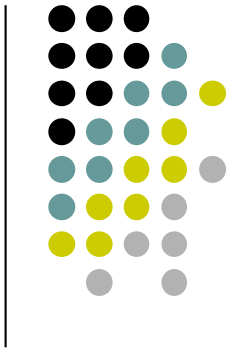


Servidores multithreaded



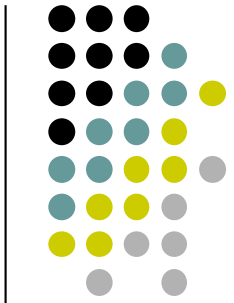
- Os servidores multithreaded surgiram para agilizar o tratamento de requisições.
- Neste modelo, diversas requisições podem estar sendo tratadas ao mesmo tempo.
- Os servidores multi-processos também permitem o tratamento simultâneo das requisições, porém a um custo de gerenciamento mais alto.

Servidores multithreaded



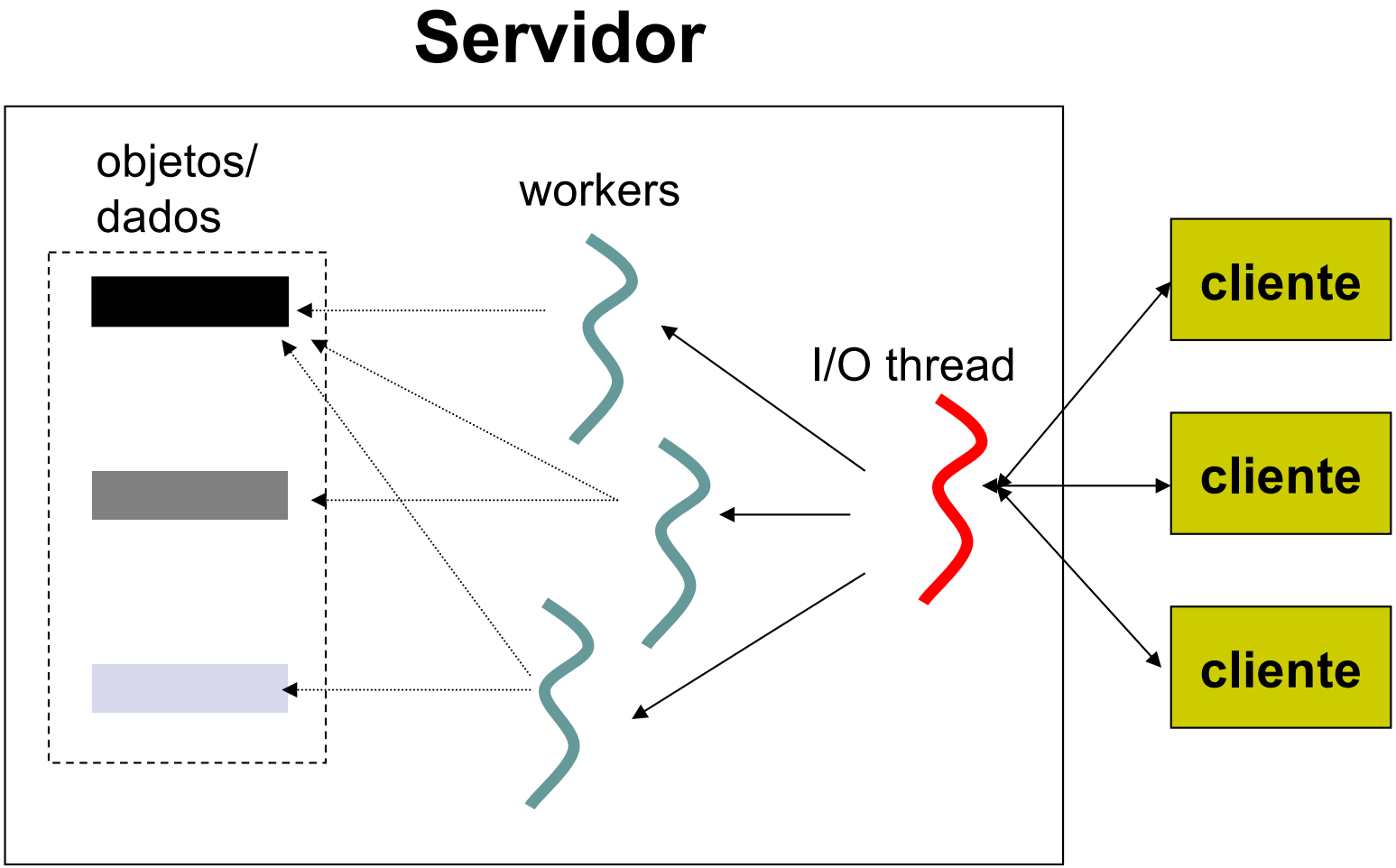
- Modelos com threads fixas
  - Dispatcher/worker
  - Team
  - Pipeline
- Modelos com threads variáveis
  - Thread per request
  - Thread per connection
  - Thread per object

Thread per request

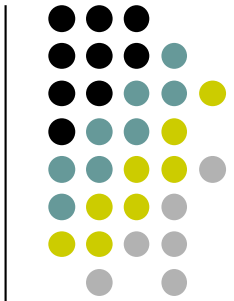


- A thread dispatcher (I/O thread) cria uma nova thread trabalhadora para cada requisição
- A trabalhadora se auto-destrói após o término do processamento da requisição

Thread per request

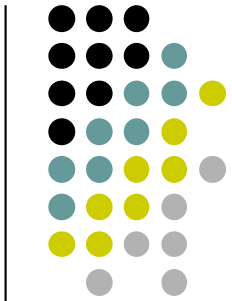


Thread per request



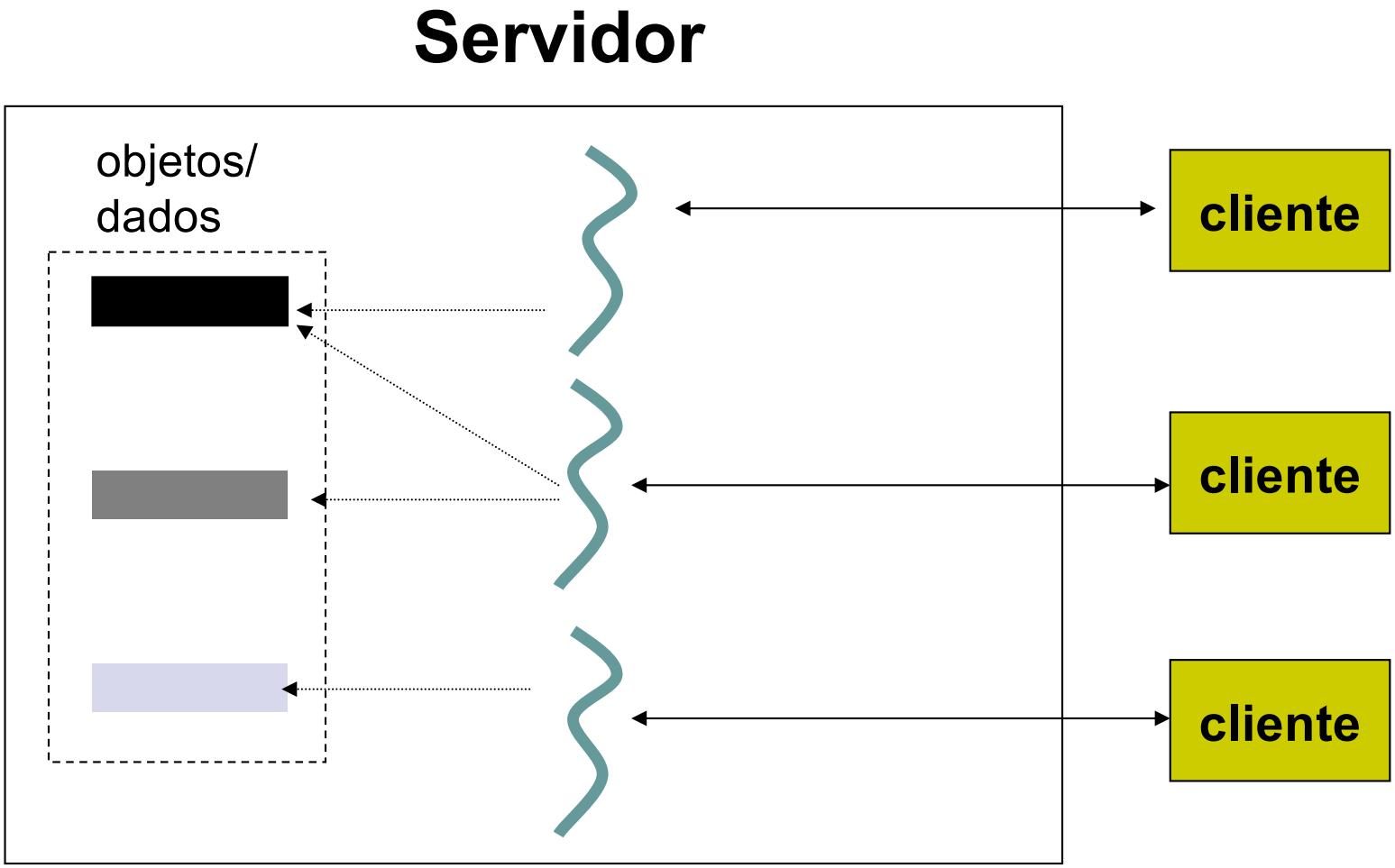
- Como não existe um número fixo de threads, não há contenção na fila de requisições.
- Porém, se a execução do serviço for muito rápida, o overhead de criação e destruição de threads passa a ser muito alto.

Thread per connection

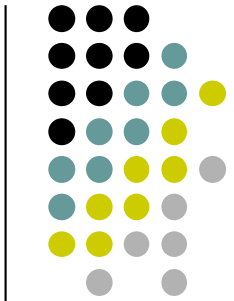


- Neste caso, a cada conexão é associada uma thread.
- O servidor cria uma nova thread quando o cliente estabelece uma conexão e a destrói quando a conexão é finalizada.
- Todas as solicitações para uma mesma conexão são enfileiradas

Thread per connection

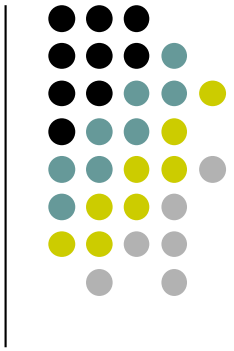


Thread per connection



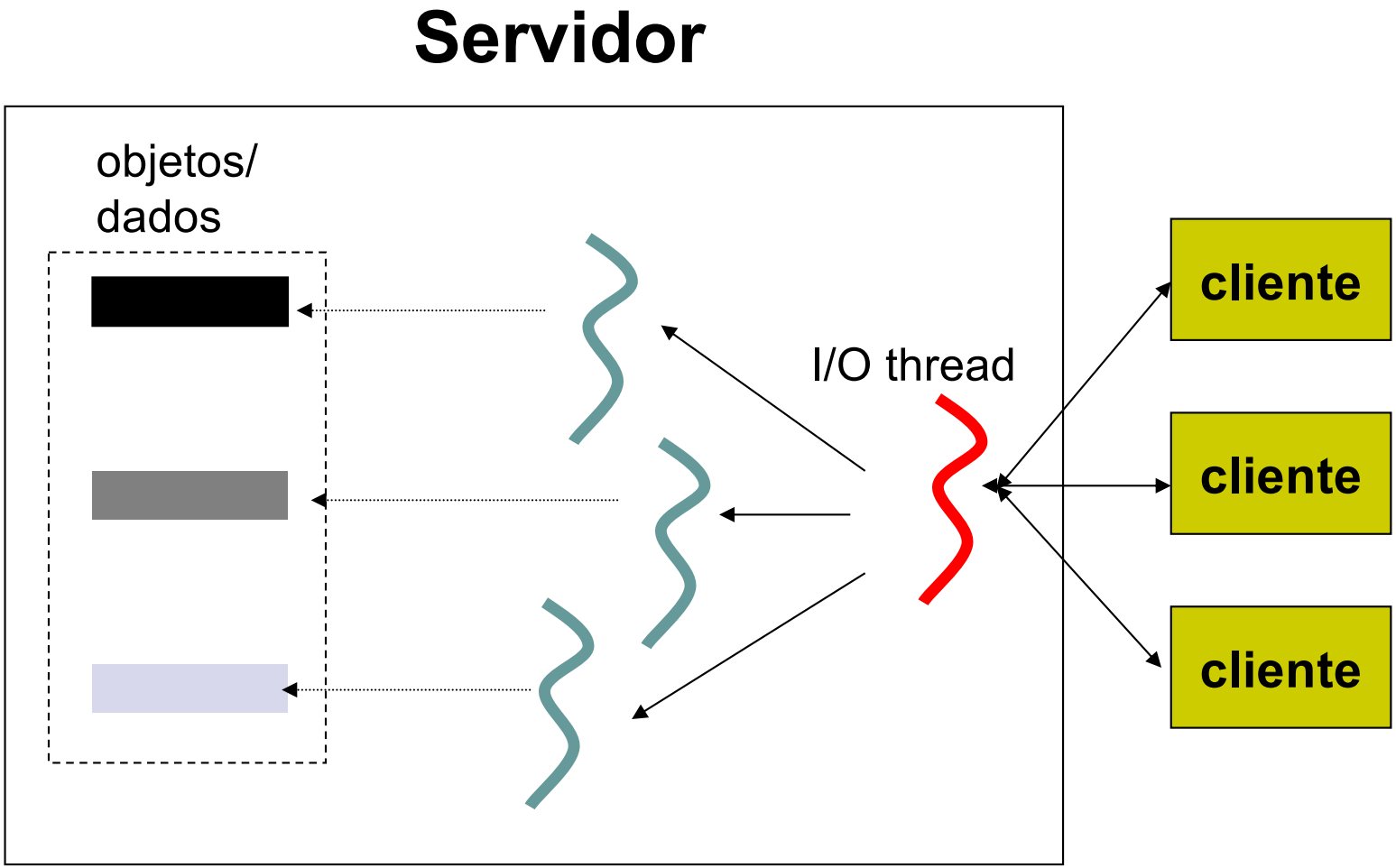
- Em relação à abordagem thread per request, esta abordagem reduz o ovehead de gerenciamento de threads.
- No entanto, as requisições de alguns clientes podem ficar enfileiradas quando existem threads ociosas no servidor.

Thread per object

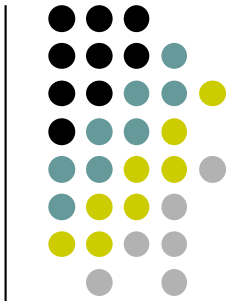


- Neste caso, uma thread é associada a cada objeto envolvido no tratamento da requisição.
- A thread de I/O recebe as solicitações e as encaminha às threads associadas aos objetos.
- Este modelo visa reduzir a conteção no acesso a mecanismos de sincronização.

Thread per object

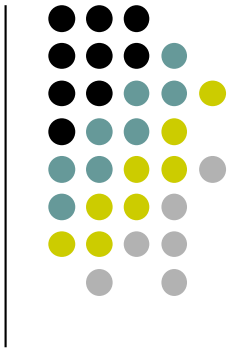


Thread per object



- Da mesma maneira que a abordagem thread per connection, esta abordagem reduz o ovehead de gerenciamento de threads, em relação à abordagem thread per request.
- No entanto, da mesma forma, as requisições de alguns clientes podem ficar enfileiradas quando existem threads ociosas no servidor.

Envio e Recepção de Mensagens

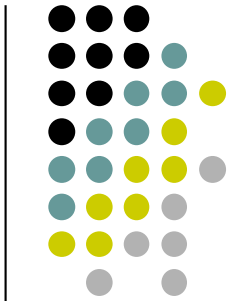


- Tipo das primitivas:
  - bloqueadas/não-bloqueadas
- Tipo do armazenamento de mensagens
  - bufferizado / não bufferizado
- Confiabilidade
  - confiáveis / não confiáveis

Primitivas Bloqueadas e Não-Bloqueadas

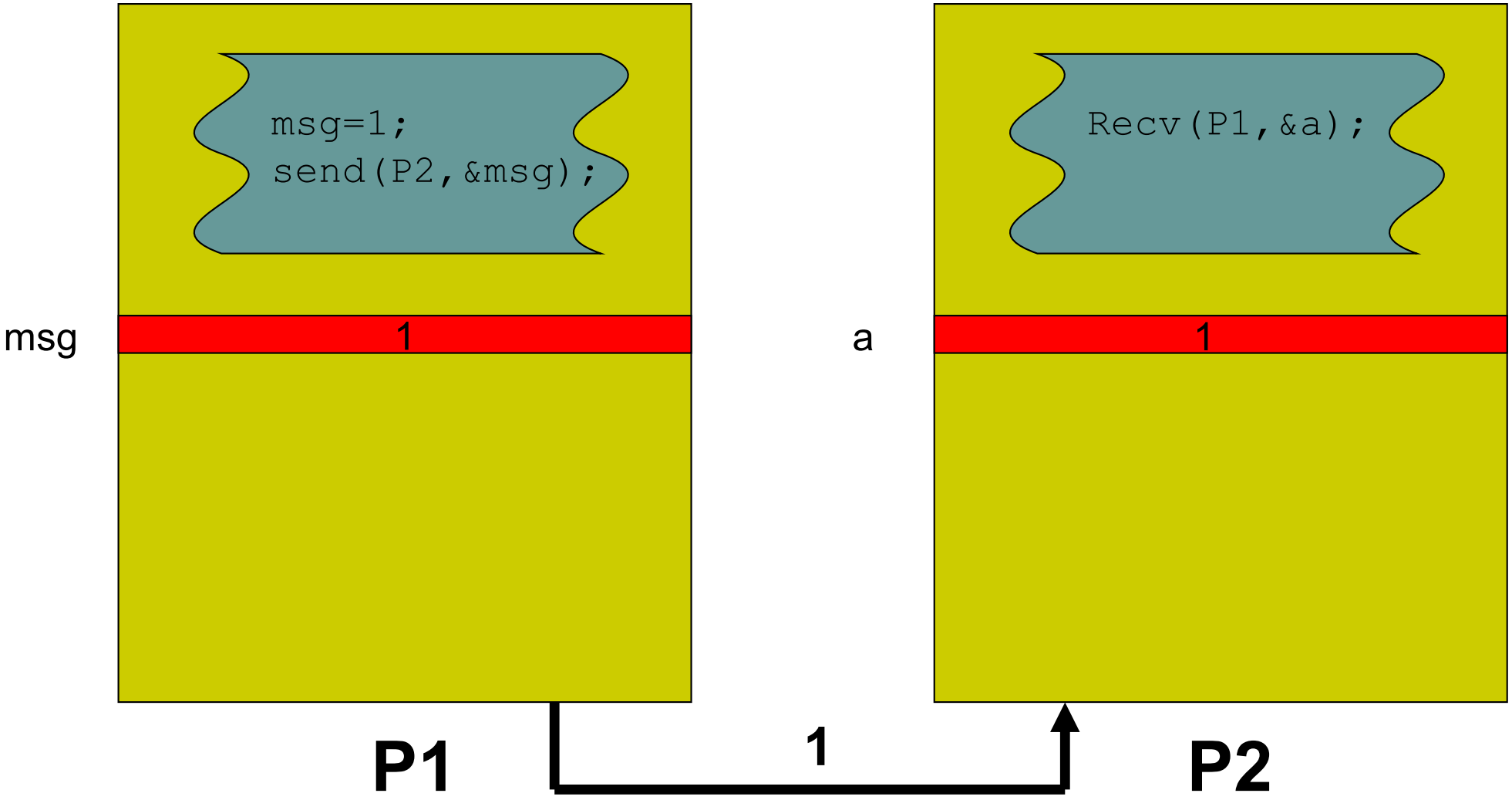
- A comunicação por troca de mensagem é composta por duas partes: uma que quer enviar mensagens e outra que deseja recebê-las. A comunicação somente ocorre quando as duas partes estão prontas.

Primitivas Bloqueadas e Não-Bloqueadas

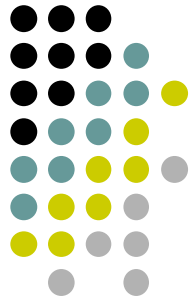


- No projeto das primitivas de comunicação, devemos determinar o comportamento delas no caso da outra parte não estar pronta quando executamos a nossa primitiva:
  - Bloqueio da primeira parte
  - Não-bloqueio da primeira parte

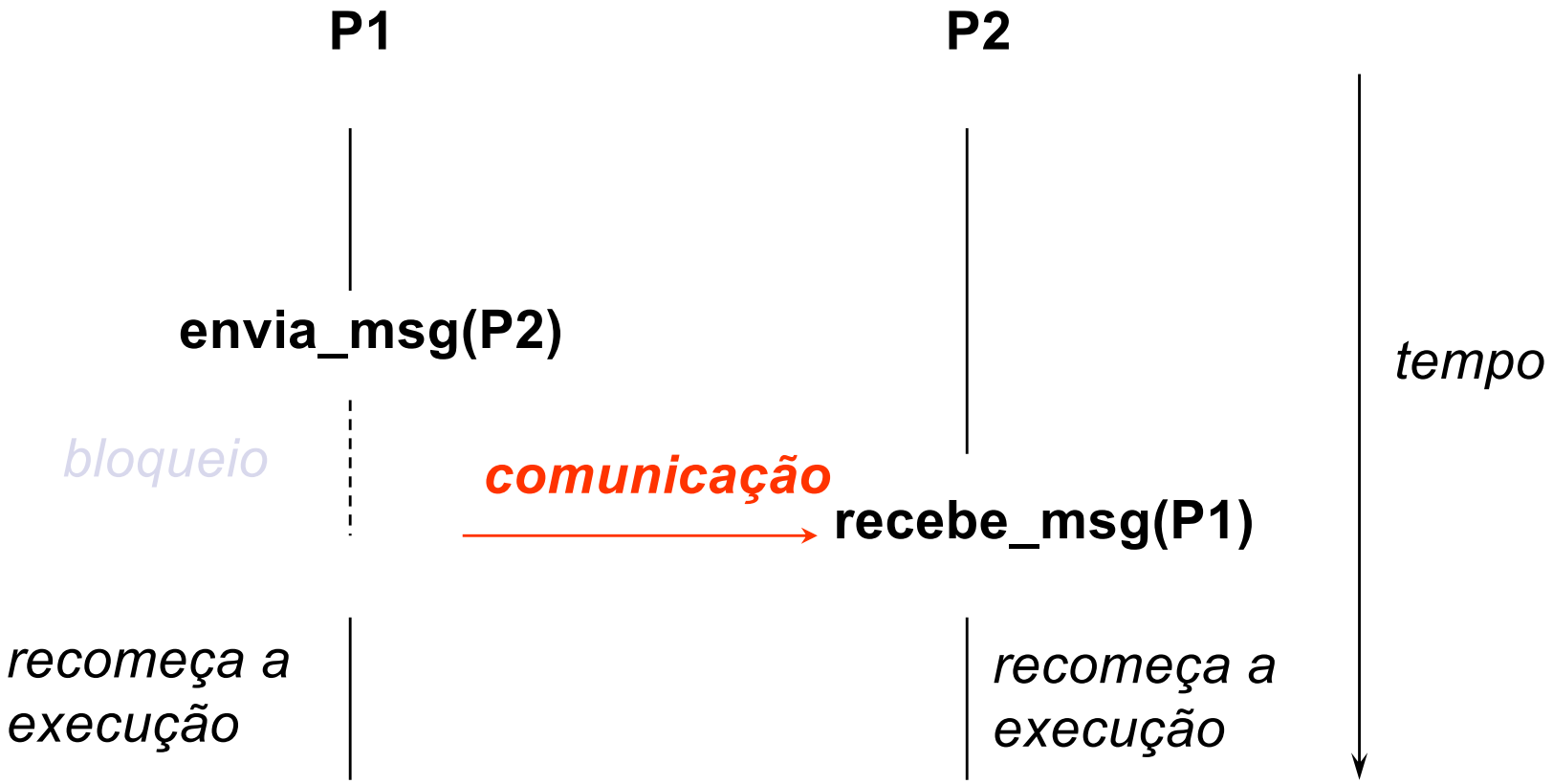
Primitivas Bloqueadas



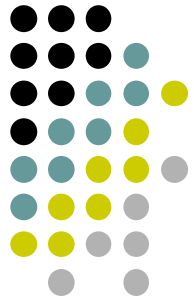
Primitivas Bloqueadas



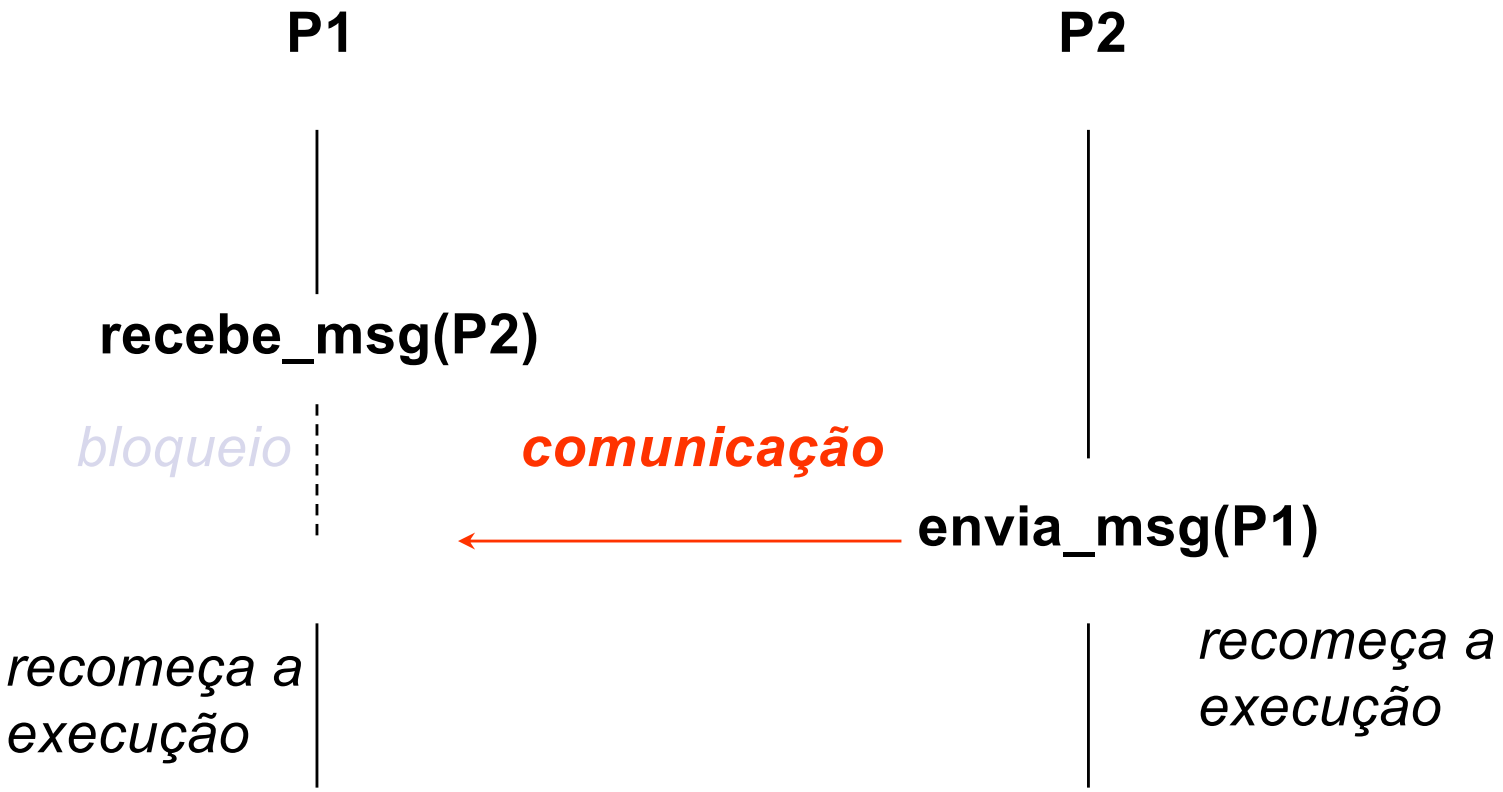
- Comportamento (send antes do receive)



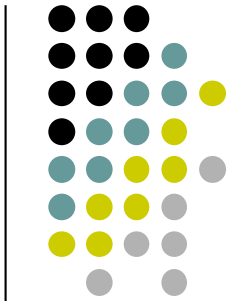
Primitivas Bloqueadas



- Comportamento (receive antes do send)

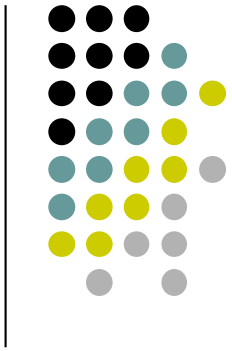


Primitivas Bloqueadas

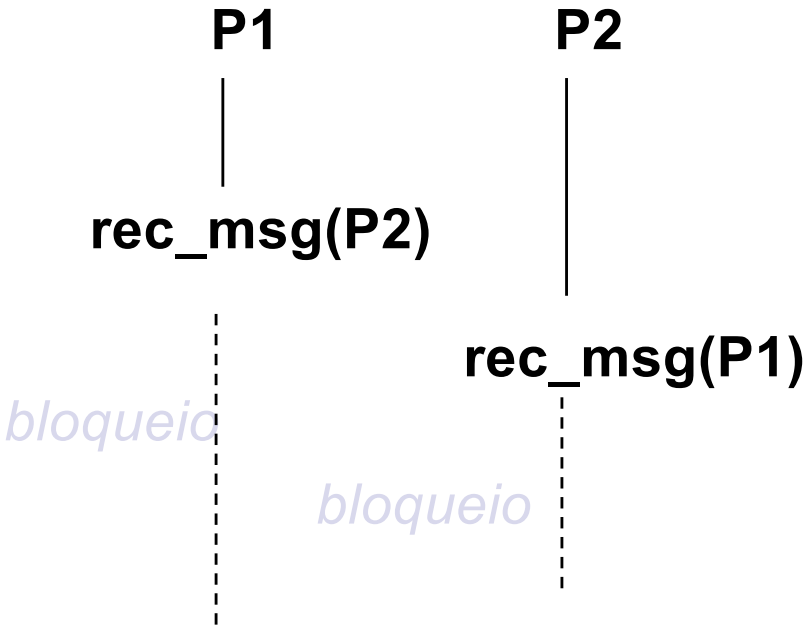


- A comunicação que utiliza primitivas bloqueadas é dita comunicação síncrona porque, no momento da troca de mensagens, cada processo envolvido na comunicação sabe exatamente em que estado da execução se encontra o outro processo (ele está executando a primitiva de comunicação).
- Assim, além de comunicar dados, os processos trocam implicitamente informações sobre o seu estado de execução

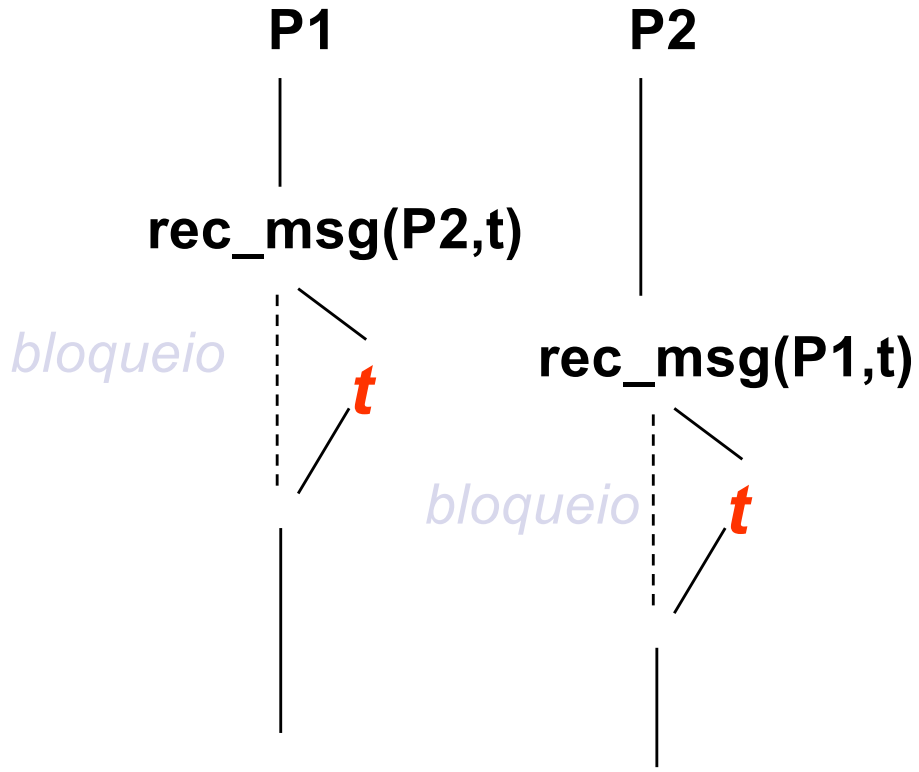
Primitivas Bloqueadas



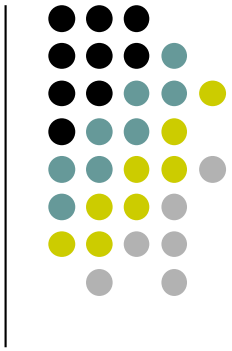
Desvantagem:
Possibilidade de deadlock



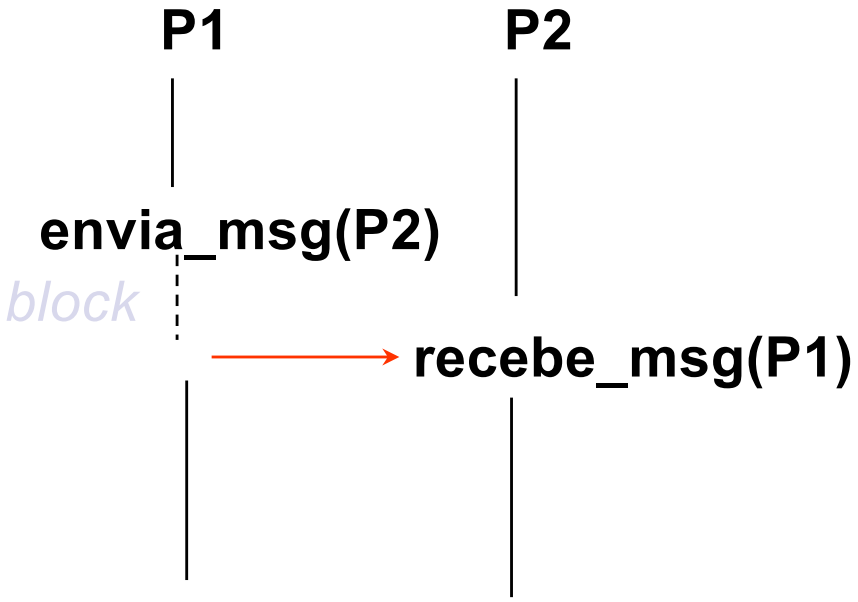
Solução:
Primitivas temporizadas



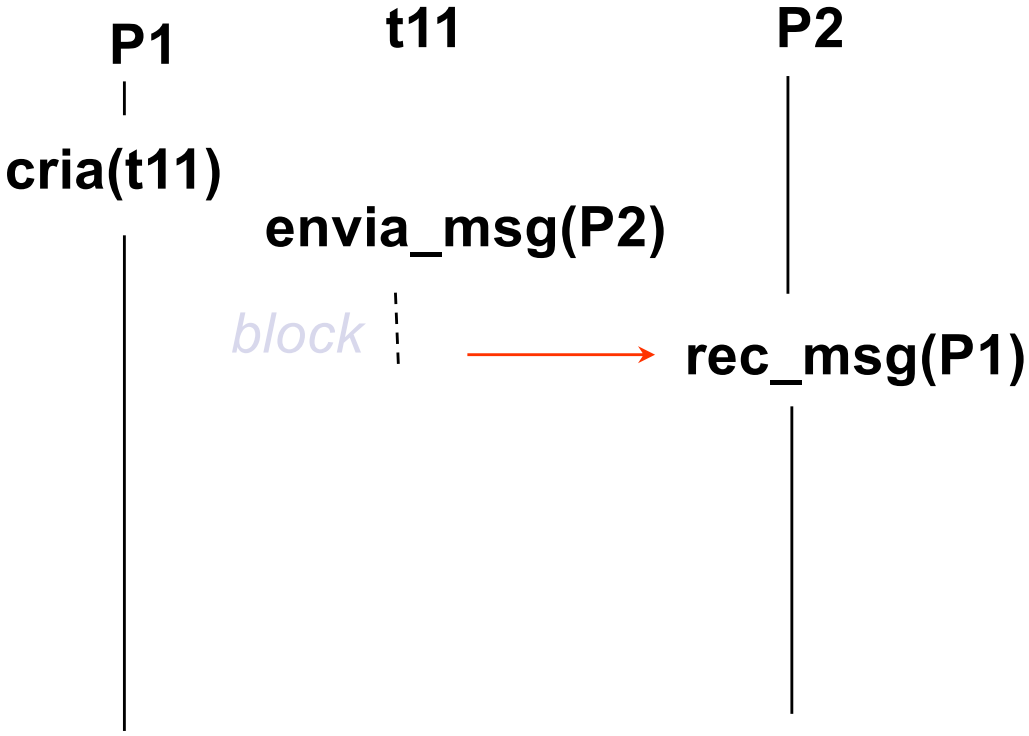
Primitivas Bloqueadas



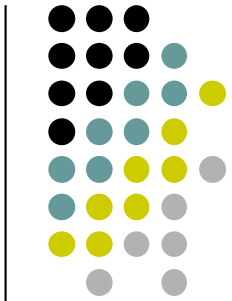
Desvantagem:
Perda de paralelismo devido ao bloqueio



Solução:
Threads de comunicação



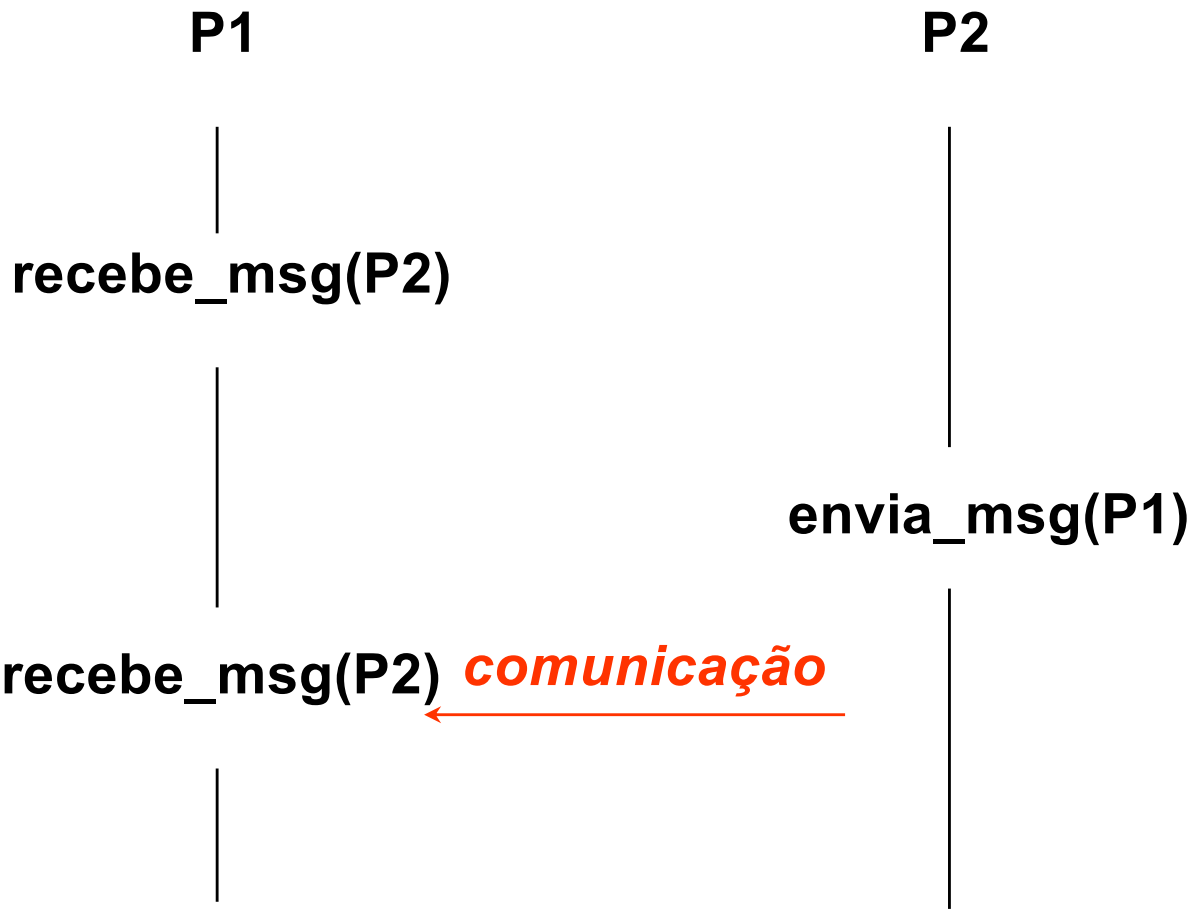
Primitivas Bloqueadas



- A comunicação síncrona combinada às threads de comunicação chama-se comunicação pseudo-assíncrona.
- Este tipo de comunicação está sendo bastante estudado pois engloba vantagens tanto da comunicação síncrona como da comunicação assíncrona.

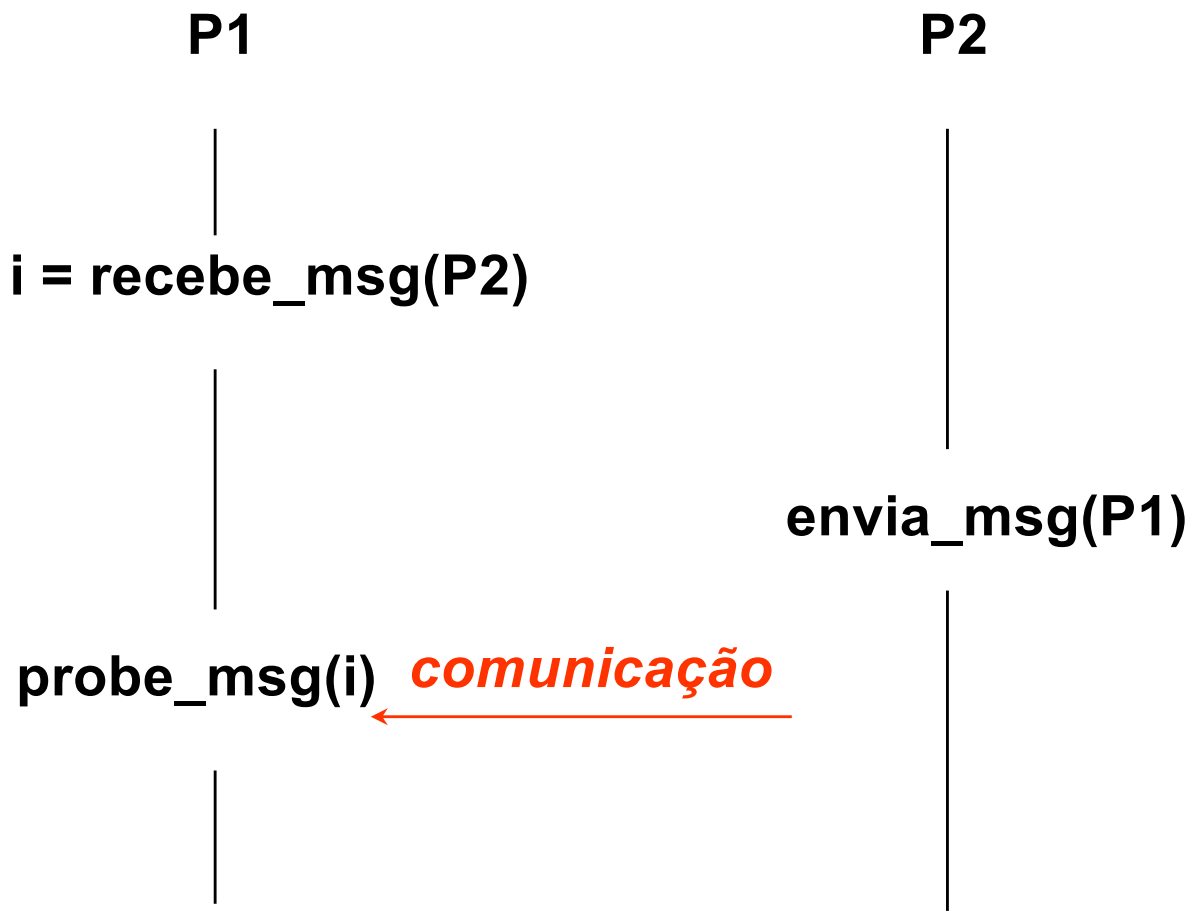
Primitivas não-bloqueadas

- Receive antes do send
  - Comportamento 1: receive retorna NOMSG



Primitivas não-bloqueadas

- Receive antes do send
  - Comportamento 2: receive - probe

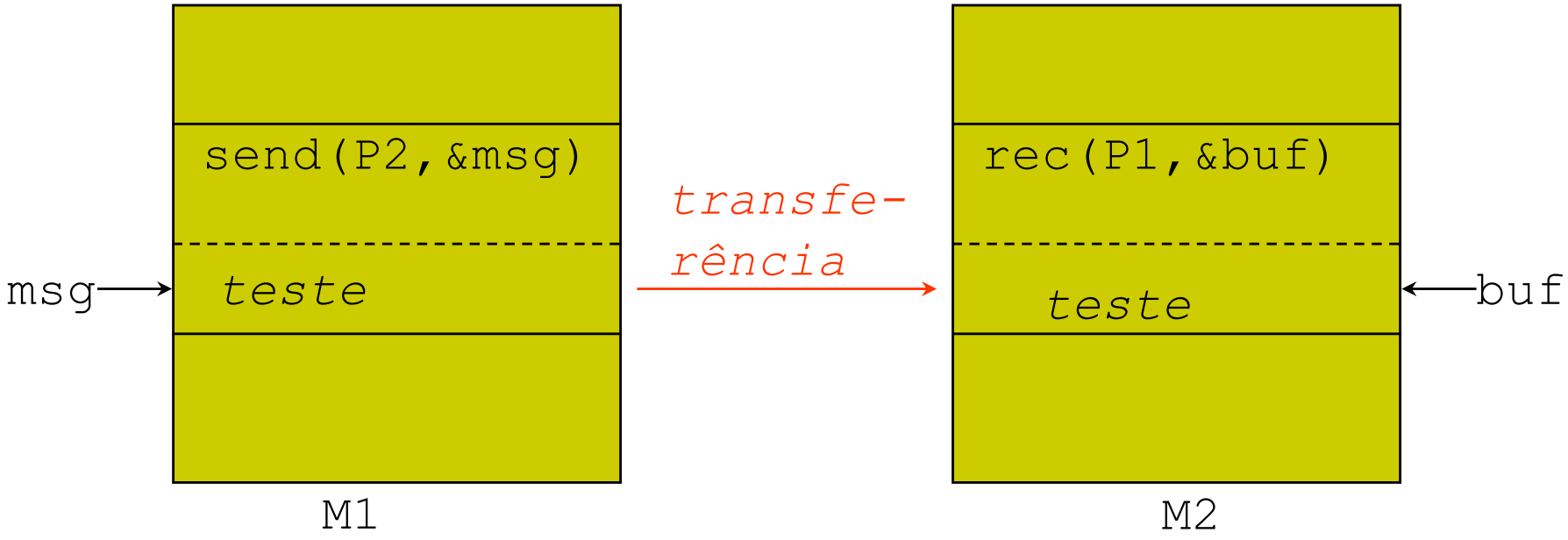


Primitivas não-bloqueadas

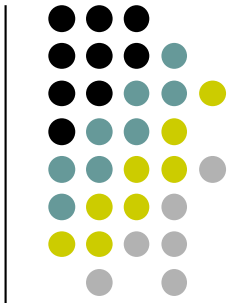
- No caso do send antes do receive, onde guardar a mensagem até que o receive correspondente seja executado?
  - manter a mensagem no próprio espaço de endereçamento do processo
  - copiar a mensagem para uma área interna do sistema operacional

Manter a mensagem na área do processo

- O processo não pode reutilizar o buffer de mensagem até ter certeza de que a mensagem foi transmitida ao processo receptor.

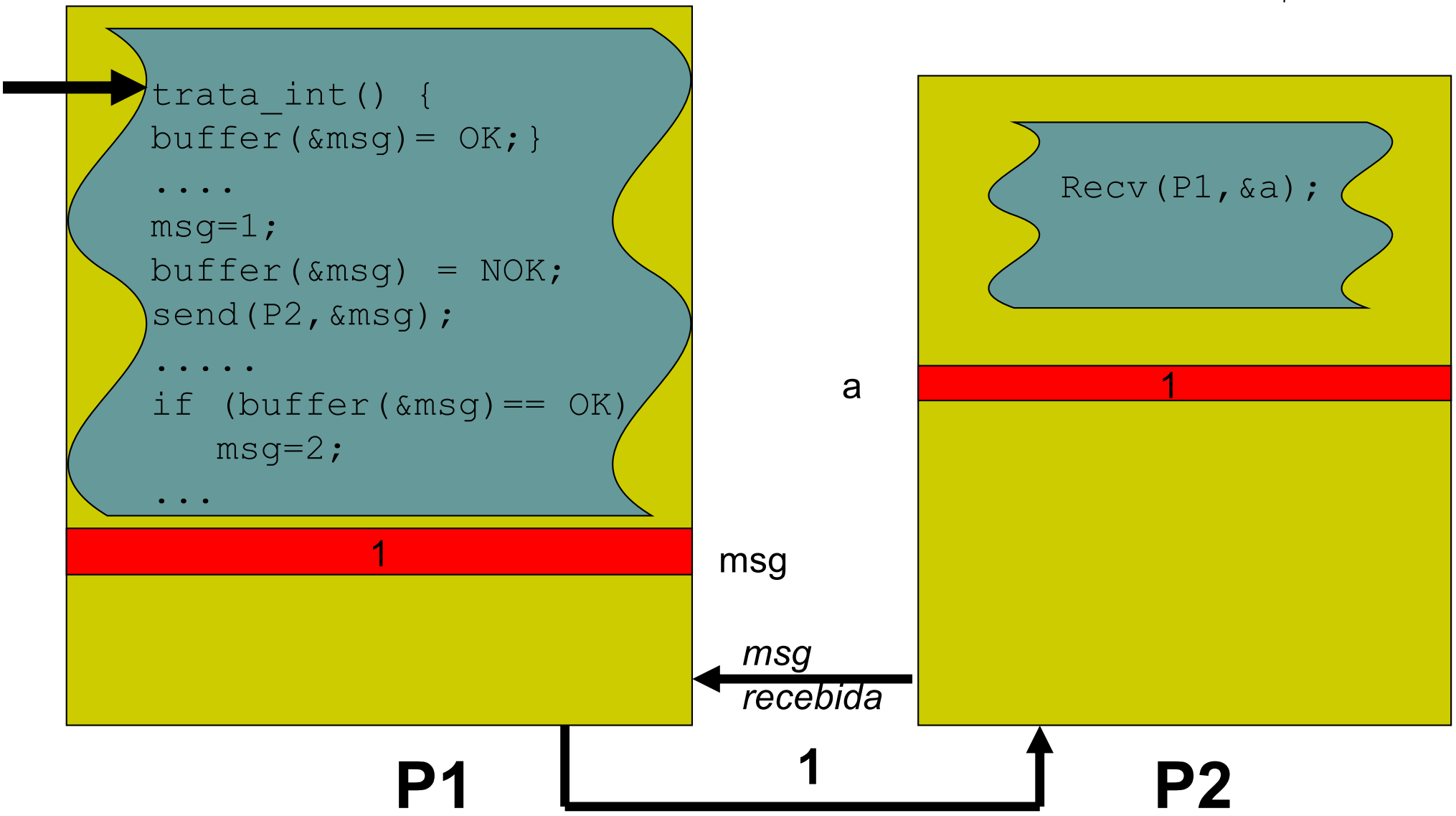


Manter a mensagem na área do processo

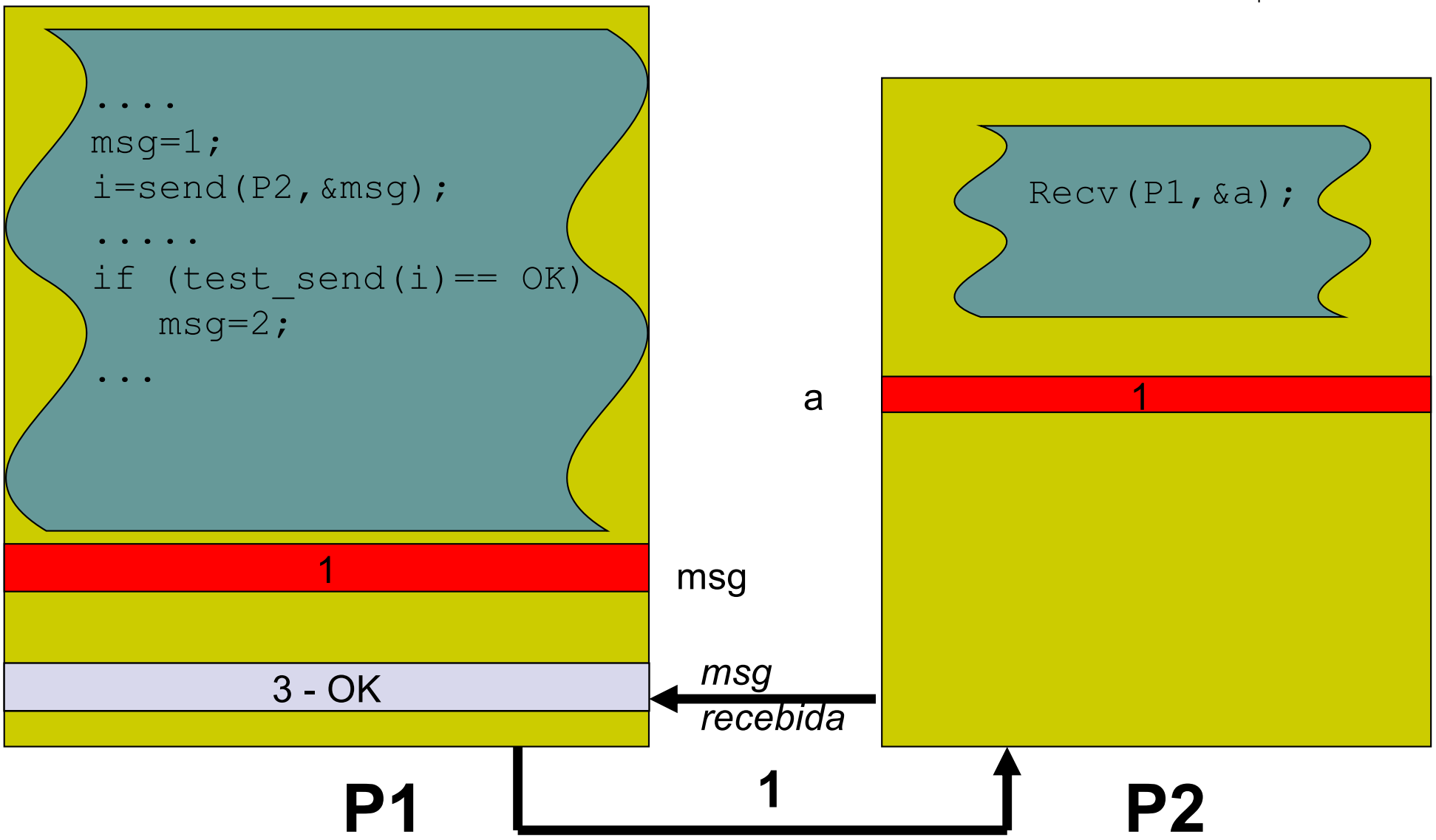


- Como saber se a mensagem já foi enviada?
  - Interrupção do processo emissor no momento da recepção da mensagem
  - Primitiva adicional de teste de envio de mensagem:
    - i = send_assinc(msg);
    - test_send(i);

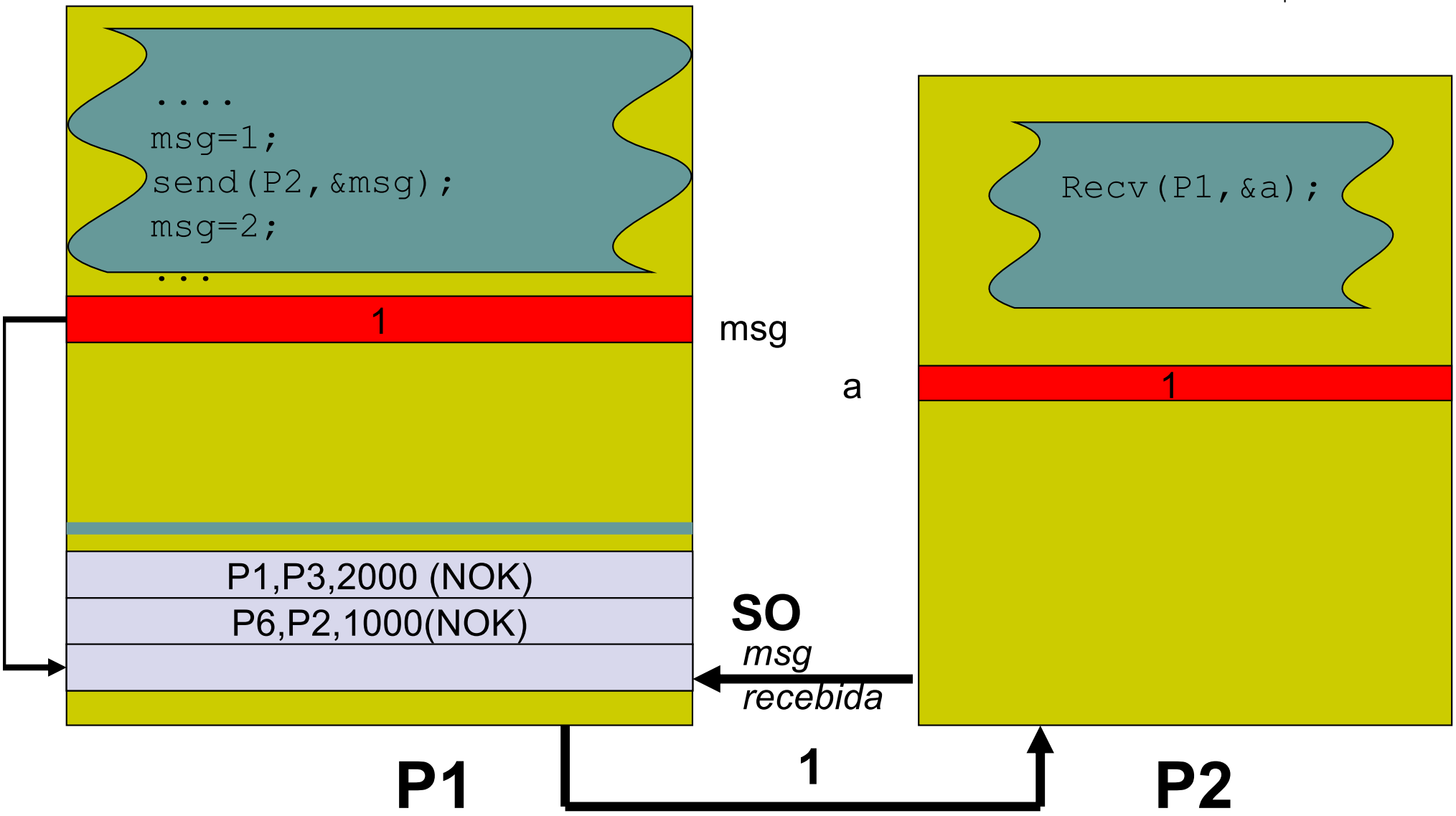
Manter a mensagem na área do processo -Interrupção do Emissor



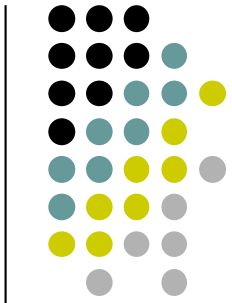
Manter a mensagem na área do processo - Primitiva Adicional



Copiar a mensagem para o SO

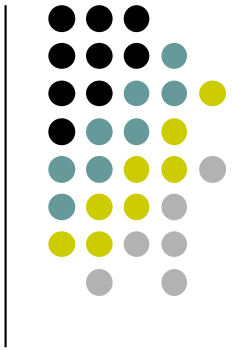


Copiar a mensagem para o SO



- No momento do send, a mensagem é copiada para a área do SO e o processo enviador pode continuar sua execução, sem se preocupar com o seu buffer.
- No entanto, neste caso há uma cópia adicional e há necessidade de mecanismos no kernel para gerenciamento de buffers.

Primitivas Bloqueadas x Primitivas não Bloqueadas



Primitivas Bloqueadas

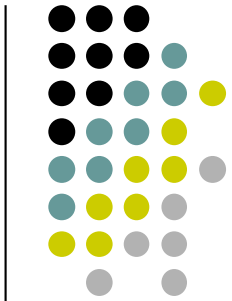
+ Semântica mais simples
- Possibilidade de deadlock
- Perda de concorrência

Primitivas não-bloqueadas

+ Alto potencial de concorrência
- Semântica complexa

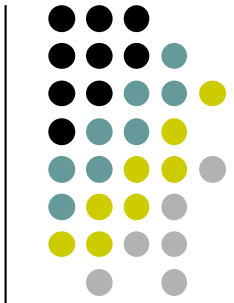
- *A maioria dos sistemas permite tanto a comunicação síncrona como a comunicação assíncrona. O programador escolhe entre as duas ou pode usar uma combinação delas, por exemplo, send não-bloqueado e receive bloqueado.*

Primitivas Bufferizadas x Primitivas não Bufferizadas



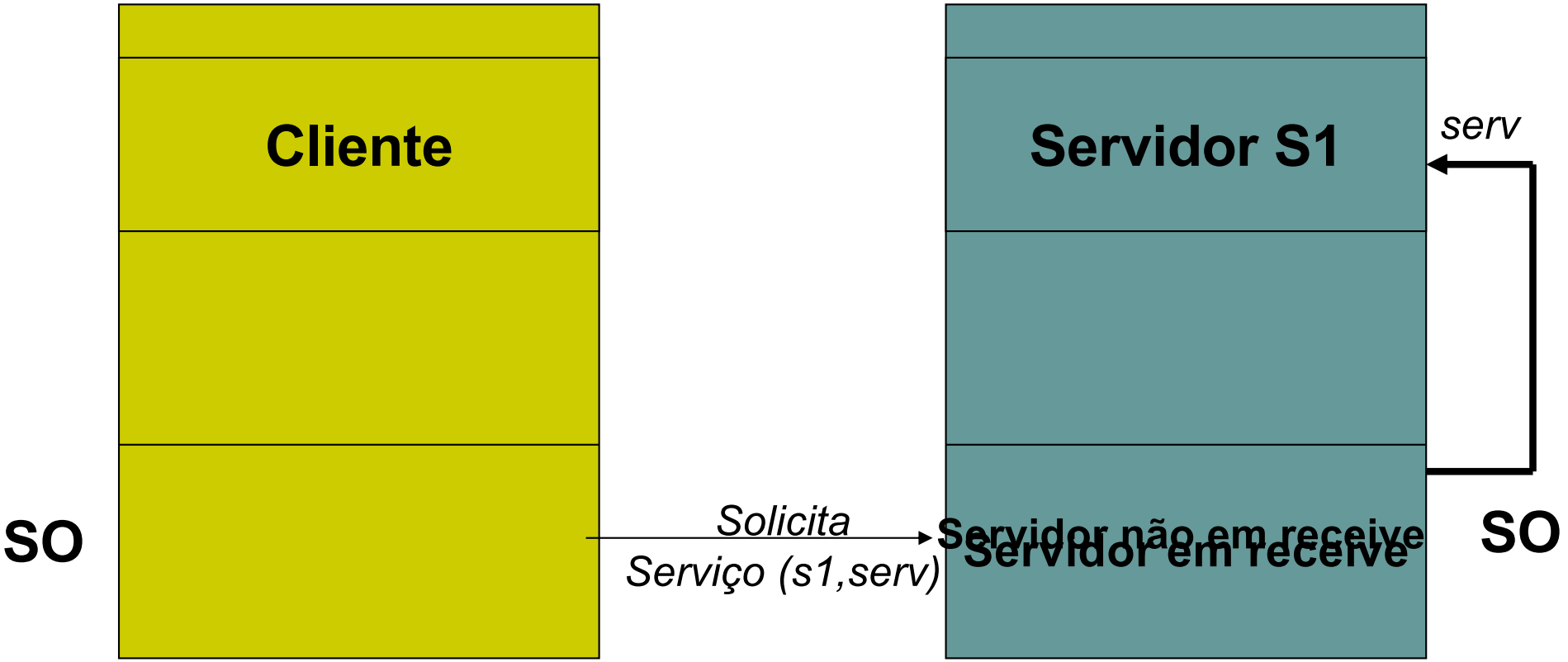
- Questão: Onde armazenar as mensagens que chegam no servidor enquanto este está tratando outra mensagem?
- Soluções:
  1. A mensagem é descartada.
  2. O kernel da máquina receptora guarda a mensagem por um tempo.
  3. Envio de mensagens a estruturas de dados e não a processos.
  4. Bloquear o envio quando não há espaço no destino.

Solução 1 – A mensagem é descartada

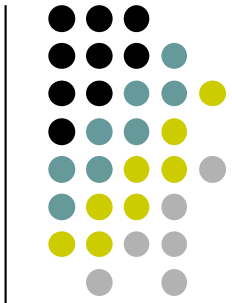


- O kernel da máquina receptora verifica se o servidor está esperando a mensagem.
  - Se sim, entrega a mensagem ao servidor
  - Caso contrário, descarta a mensagem
- O cliente retransmite-a por timeout.
- Problemas:
  - sobrecarga da rede,
  - postergação indefinida,
  - detecção de saída de servidor por timeout.

Solução 1 – A mensagem é descartada

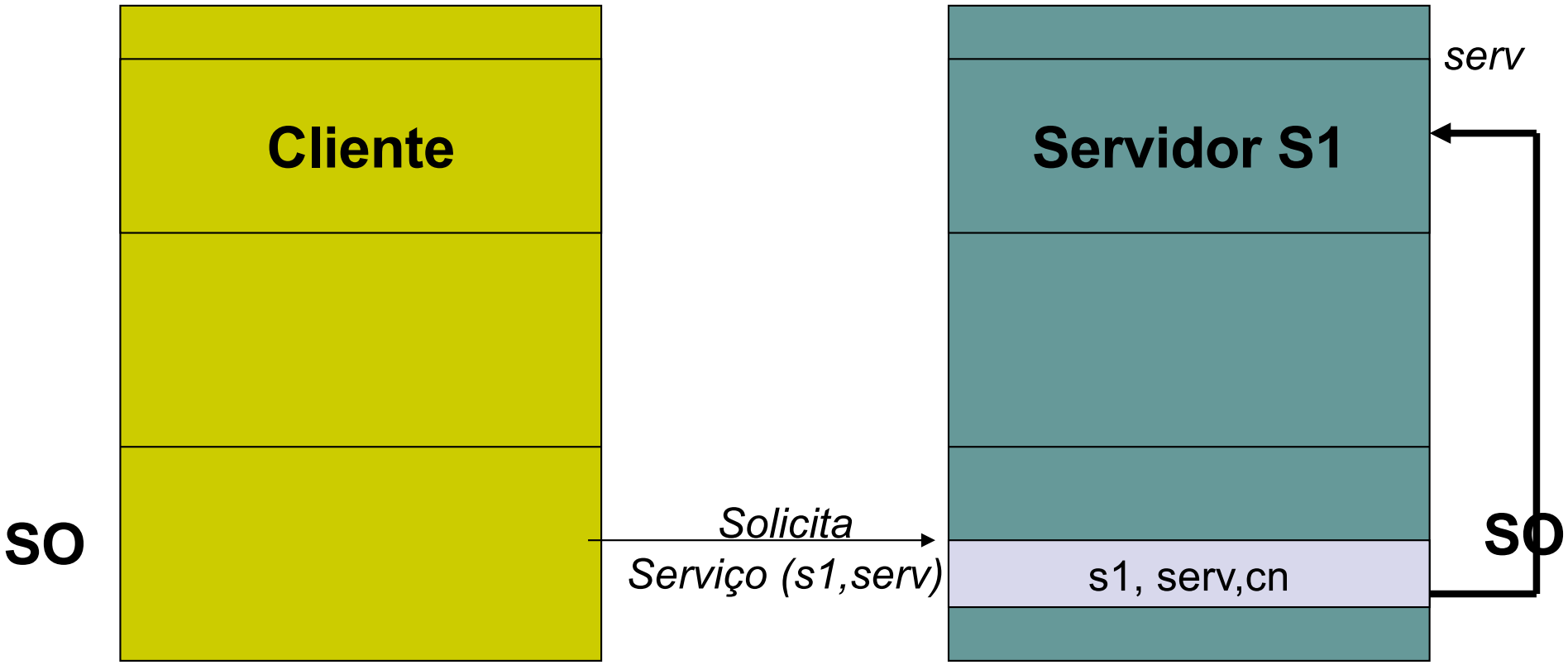


Solução 2 – A mensagem é guardada por um tempo

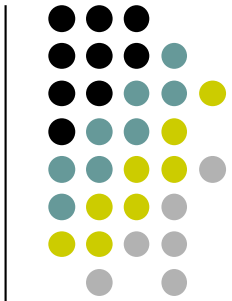


- O kernel da máquina receptora guarda a mensagem por um tempo. Se, após este tempo, a mensagem não tiver sido “recebida”, ela é descartada.
  - Problema: necessidade de gerenciamento de buffers.

Solução 2 – A mensagem é guardada por um tempo

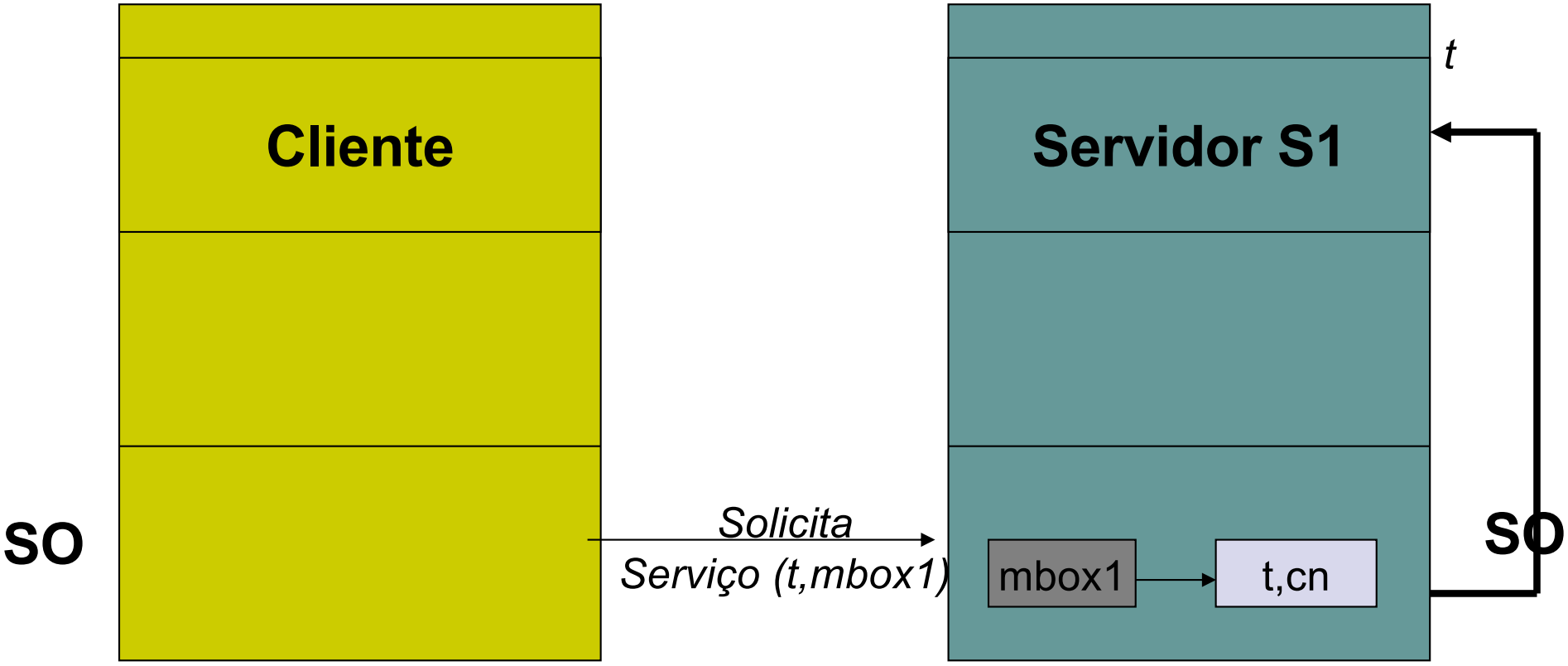


Solução 3 – Envio de msgs a estruturas de dados

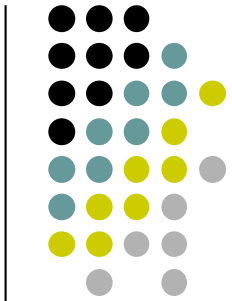


- Neste caso, a mensagem é enviada a uma estrutura de dados gerenciada pelo kernel e não diretamente ao processo servidor.
- Problemas: Gerenciamento de buffers, Tratamento de buffer cheio.

Solução 3 – Envio de msgs a estruturas de dados

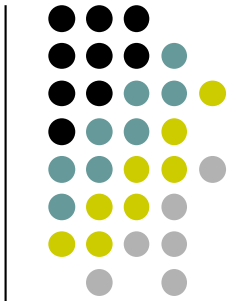


Solução 4 – Bloquear o envio para buffer cheio



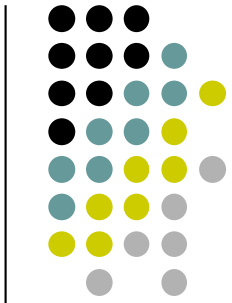
- Neste caso, o envio de mensagens, apesar de não bloqueado, pode ser blocante no caso de buffer cheio.
- Isso torna mais complexa a semântica da primitiva

Processos envolvidos na comunicação

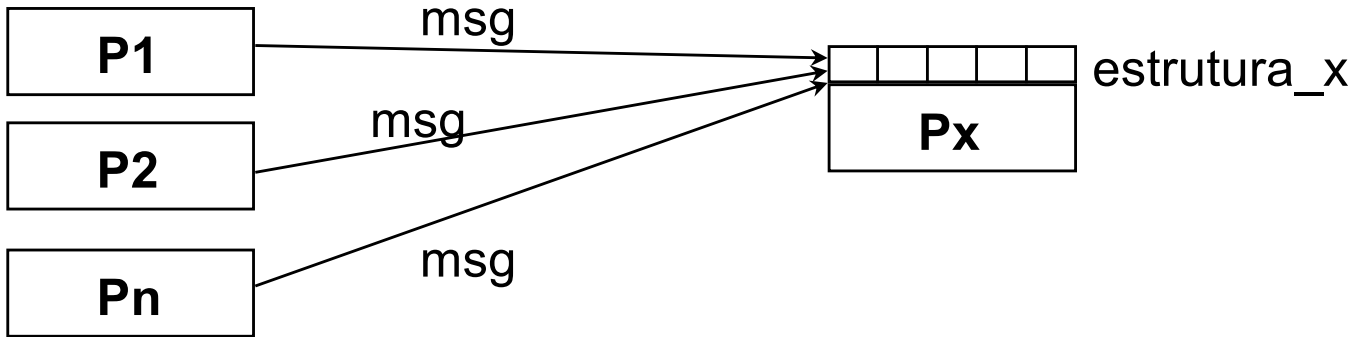


- **1 a 1**: O processo *x* envia uma mensagem ao processo *y*. O processo *y* recebe somente a mensagem do processo *x*.
  - Exemplo:
    - send(y, &msg);
    - receive(x, &msg);

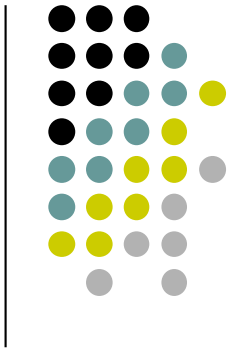
Processos envolvidos na comunicação



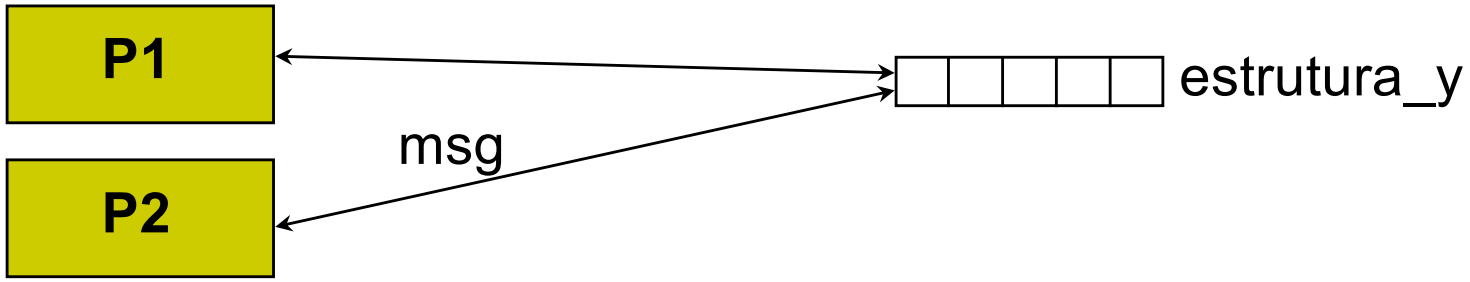
- **n a 1**: O processo *x* envia uma mensagem ao processo *y*. O processo *y* recebe mensagem de qualquer processo.
  - Exemplo:
    - send(y, &msg);
    - receive(ANY, &msg);
  - Mecanismos: portas (SO), caixas-postais restritas.



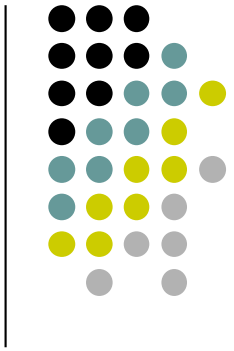
Processos envolvidos na comunicação



- **n a n**: O processo *x* envia uma mensagem a qualquer processo. O processo *y* recebe mensagem de qualquer processo.
  - Exemplo:
    - estrutura_y = aloca_caixa_postal();
    - send(estrutra_y, &msg);
    - receive(estrutura_y, &msg);
  - Mecanismos: caixas-postais genéricas.



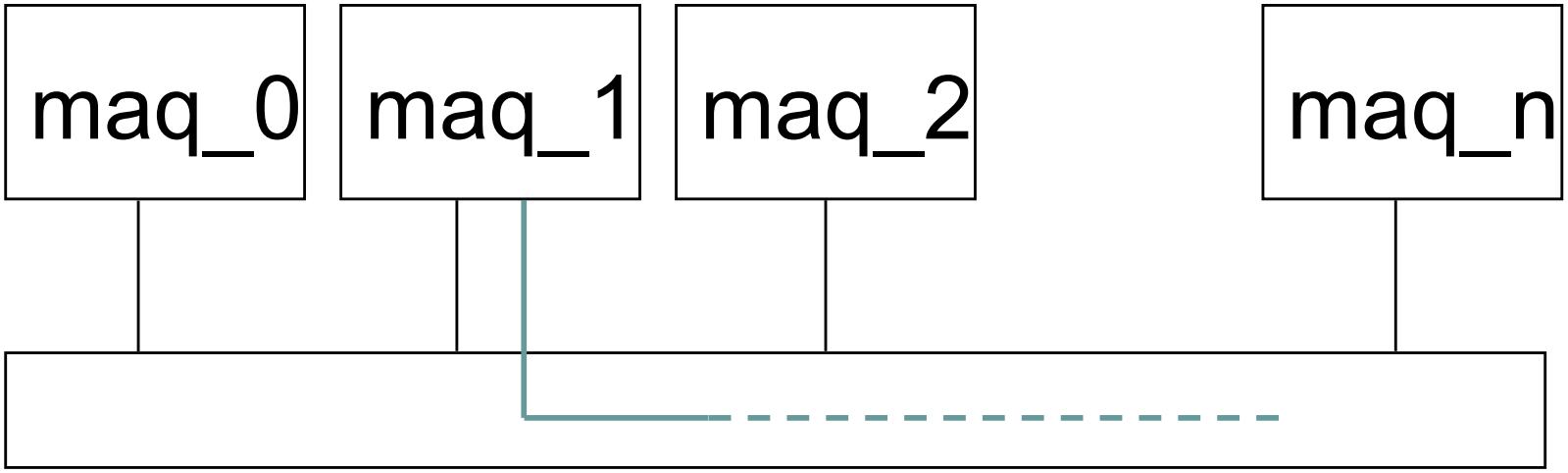
Confiabilidade da Comunicação



- Problemas que devem ser tratados quanto à confiabilidade:
  - perda de mensagens
  - inversão na ordem das mensagens
  - mensagens corrompidas.

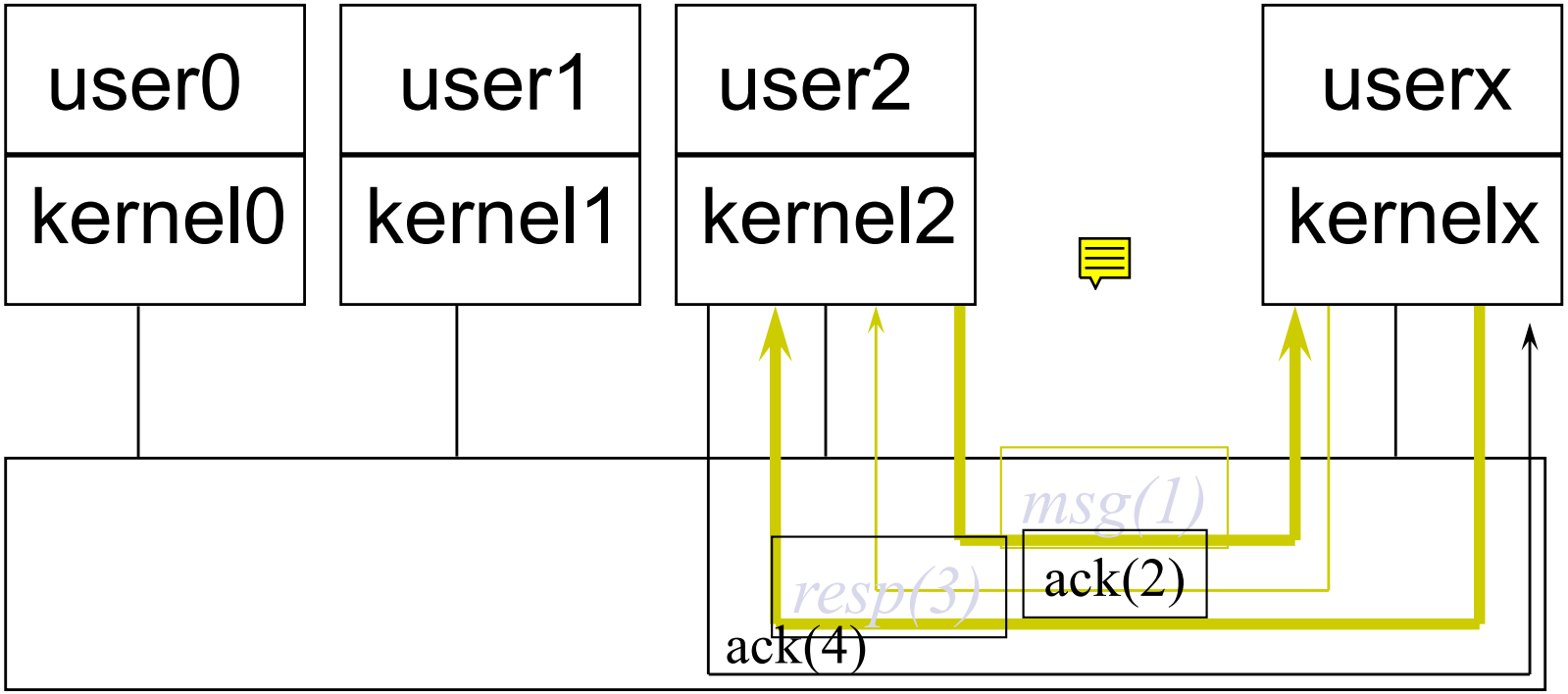
Perda de mensagens

- Problema: Uma mensagem enviada de x para y pode ser perdida no meio do caminho.

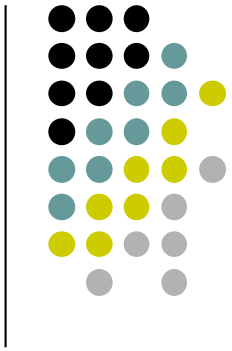


Perda de mensagens – Acknowledgements (ACKS)

- O Kernel da máquina receptora envia um ACK à máquina transmissora cada vez que uma mensagem é recebida.

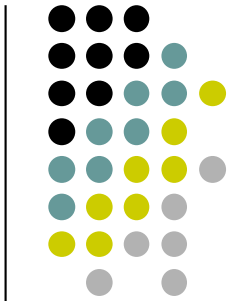


Perda de mensagens – Acknowledgements (ACKS)

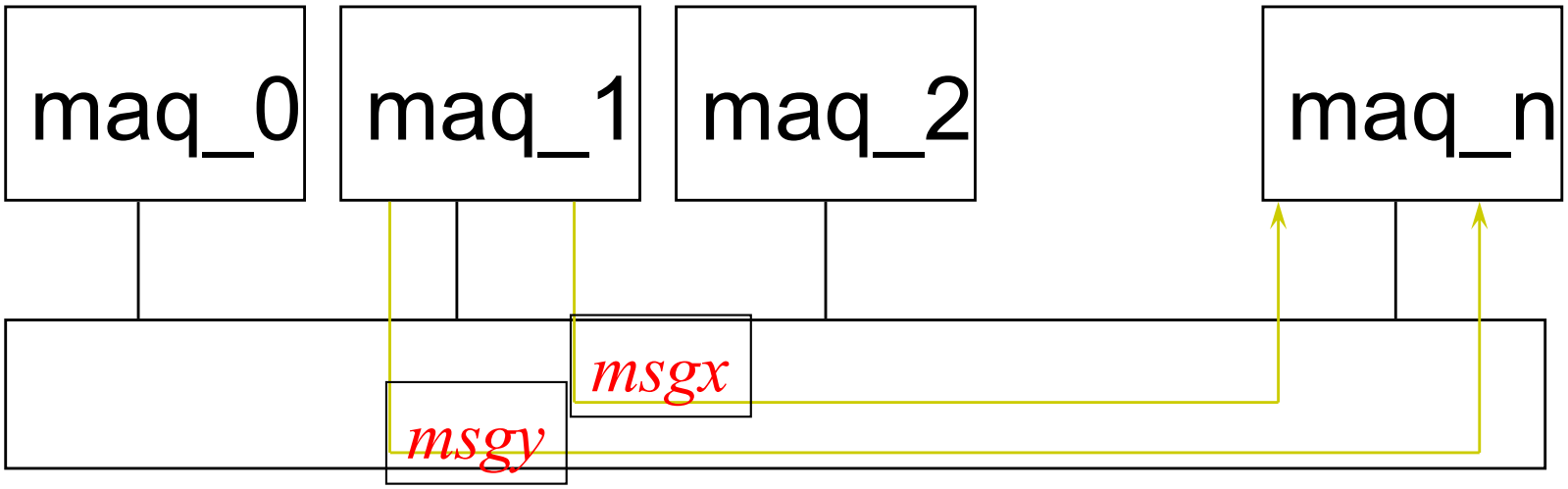


- O que confirmar (ACK)?
  - *Pacotes* - Ao ser enviada, uma mensagem é geralmente quebrada em n pacotes de tamanho fixo. A cada pacote recebido, é enviado um ACK
    - Alto overhead
    - Retransmissão de pacotes
  - *Mensagens inteiras*
    - Overhead mais baixo
    - Retransmissão de mensagens

Inversão na ordem das mensagens

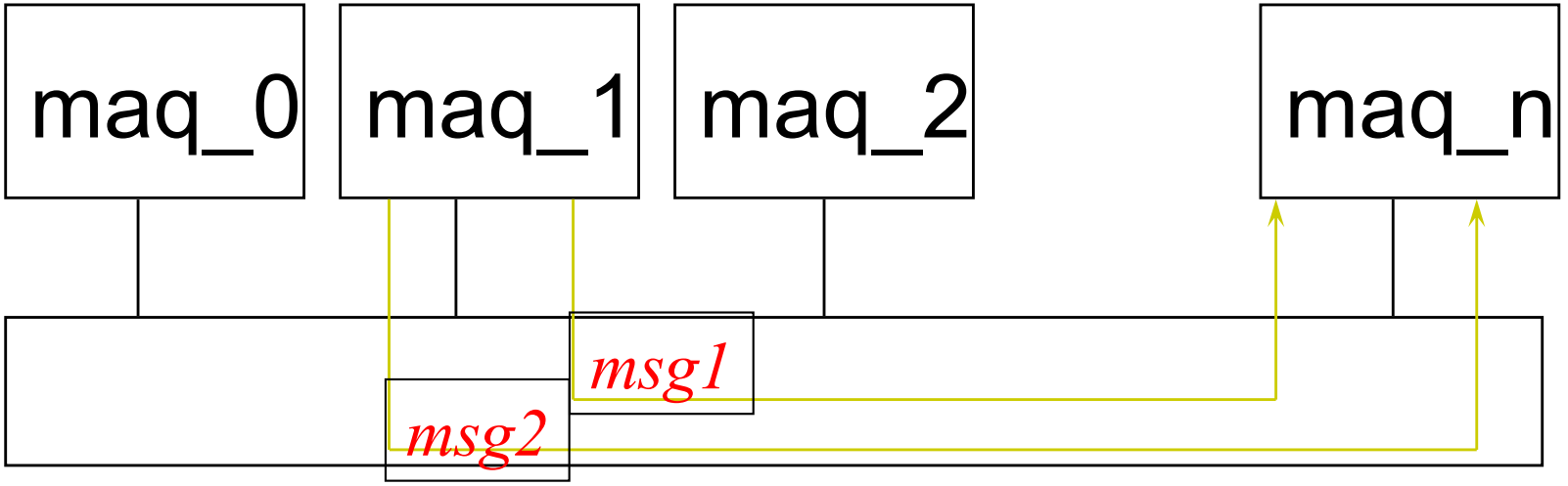


- Problema: Duas mensagens enviadas de x para y podem ser recebidas em uma ordem que seja diferente da ordem de envio.



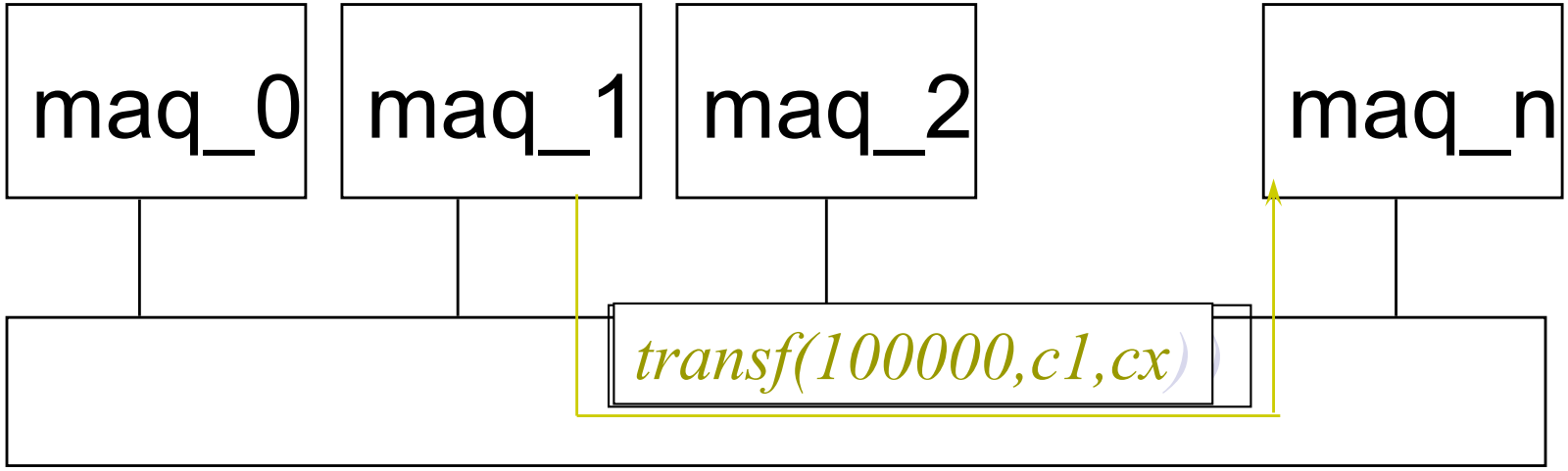
Inversão na ordem das mensagens - Numeração

- Solução: Inclusão de numeração lógica (timestamps lógicos) no header da mensagem

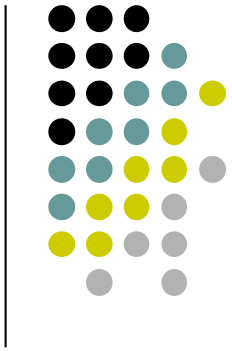


Mensagens Corrompidas

- Problema: Uma mensagem enviada de x para y com um conteúdo c' pode ser recebida em y com o conteúdo c''.



Mensagens corrompidas



- Fonte de corrupção da mensagem:
  - Meio físico de transmissão
    - Detecção por algoritmos padrão: CRC(6), bit de paridade (1), etc.
  - Agente externo
    - A mensagem foi deliberadamente alterada (falha maliciosa).
    - A detecção de falhas maliciosas é bem mais complicada