

Computação por Passagem de Mensagens

Programação por passagem de mensagens

- **Programação de multiprocessadores conectados por rede pode ser realizada:**
 - criando-se uma linguagem de programação paralela especial
 - estendendo-se as palavras reservadas de uma linguagem seqüencial existente de alto nível para manipulação de passagem de mensagens
 - utilizando-se uma linguagem seqüencial existente, provendo uma biblioteca de procedimentos externos para passagem de mensagens

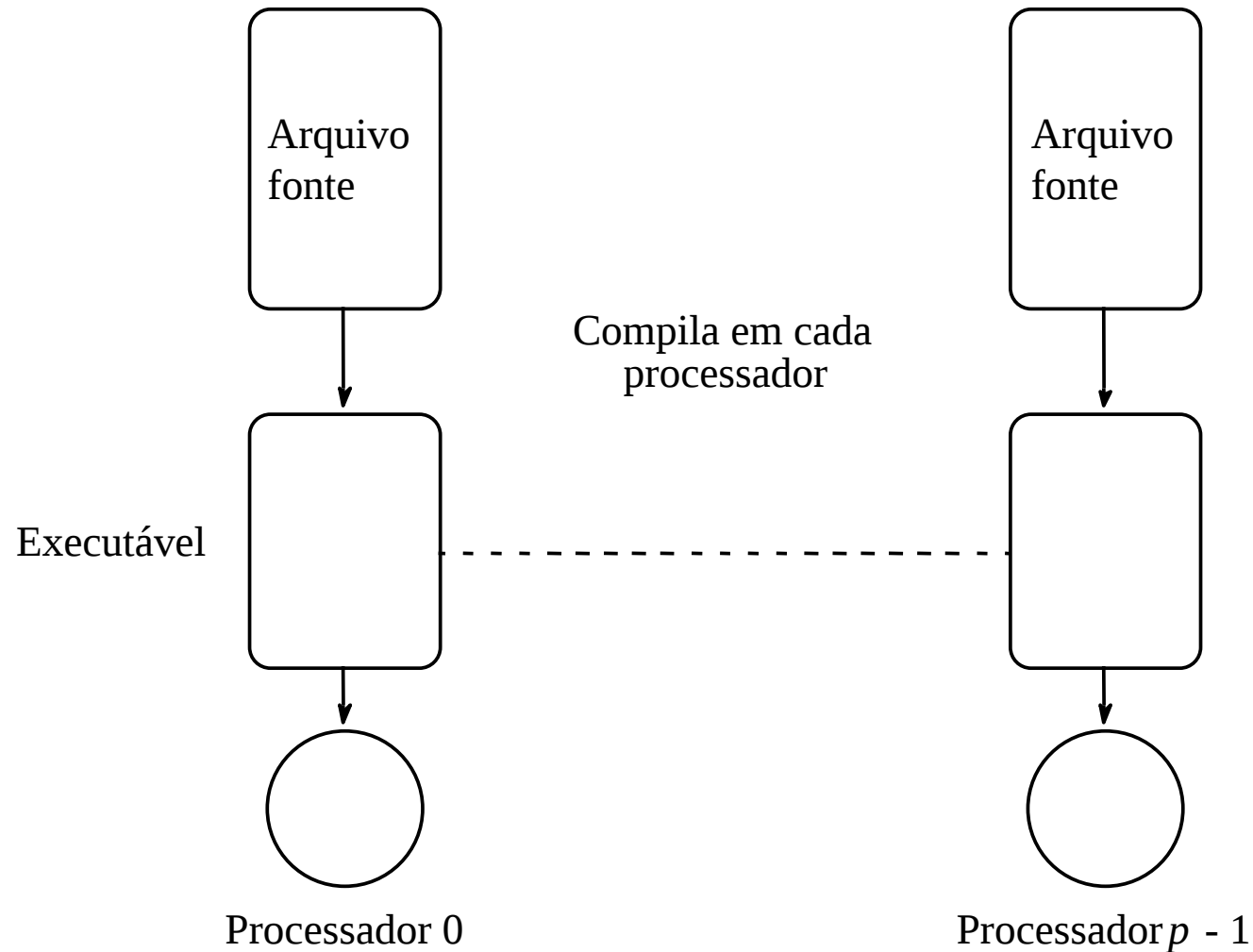
Biblioteca de rotinas para troca de mensagens

- **Necessita-se explicitamente definir:**
 - quais processos serão executados
 - quando passar mensagens entre processos concorrentes
 - o que passar nas mensagens
- **Dois métodos primários:**
 - um para criação de processos separados para execução em diferentes processadores
 - um para enviar e receber mensagens

Ferramentas de software que utilizam biblioteca de troca de mensagens

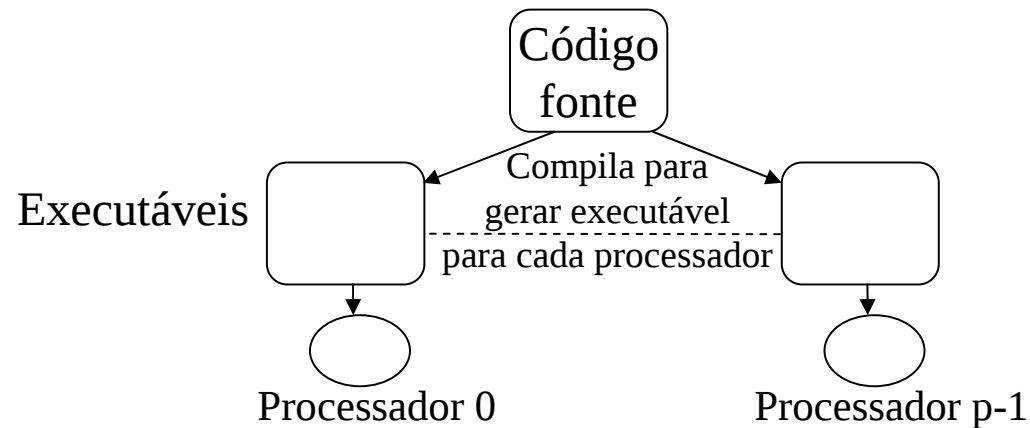
- **MPI (Message Passing Interface)**
 - padrão desenvolvido por um grupo de parceiros acadêmicos e da indústria para prover um maior uso e portabilidade das rotinas de passagem de mensagens
 - Define as rotinas, não o modo de implementá-las
 - Existem várias implementações

Multiple program, multiple data (MPMD) model



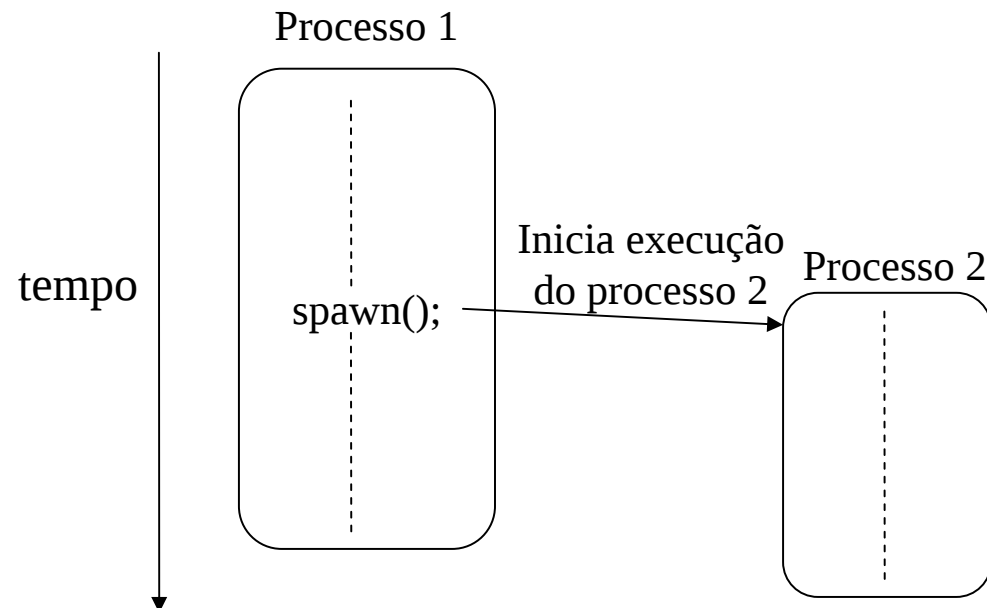
Single Program Multiple Data (SPMD)

- **Diferentes processos são unidos em um único programa e dentro deste programa existem instruções que customizam o código, selecionando diferentes partes para cada processo por exemplo**

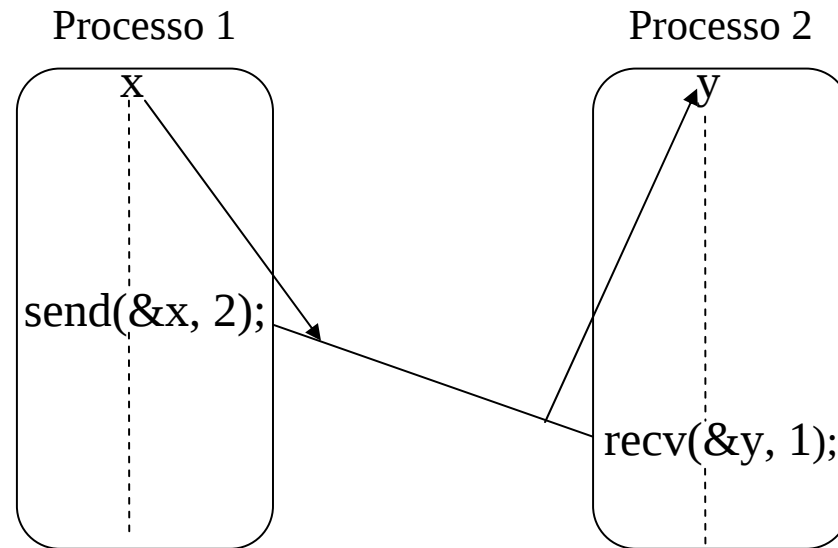


Criação dinâmica de processos

- **Programas separados escritos para cada processador, geralmente se utiliza o método mestre-escravo onde um processador executa o processo mestre e os outros processos escravos são inicializados por ele.**



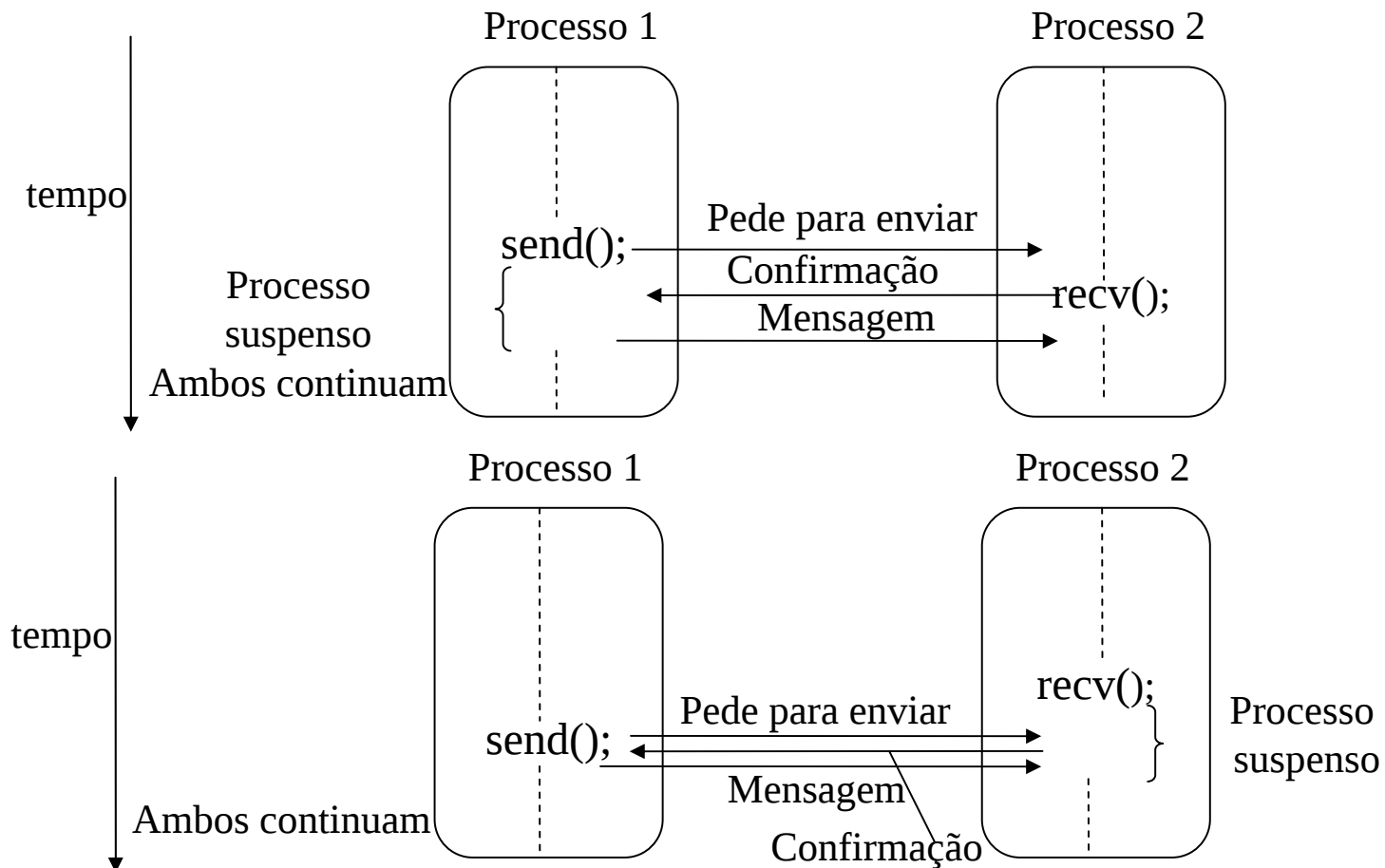
Rotinas básicas de envio e recebimento



Rotinas síncronas

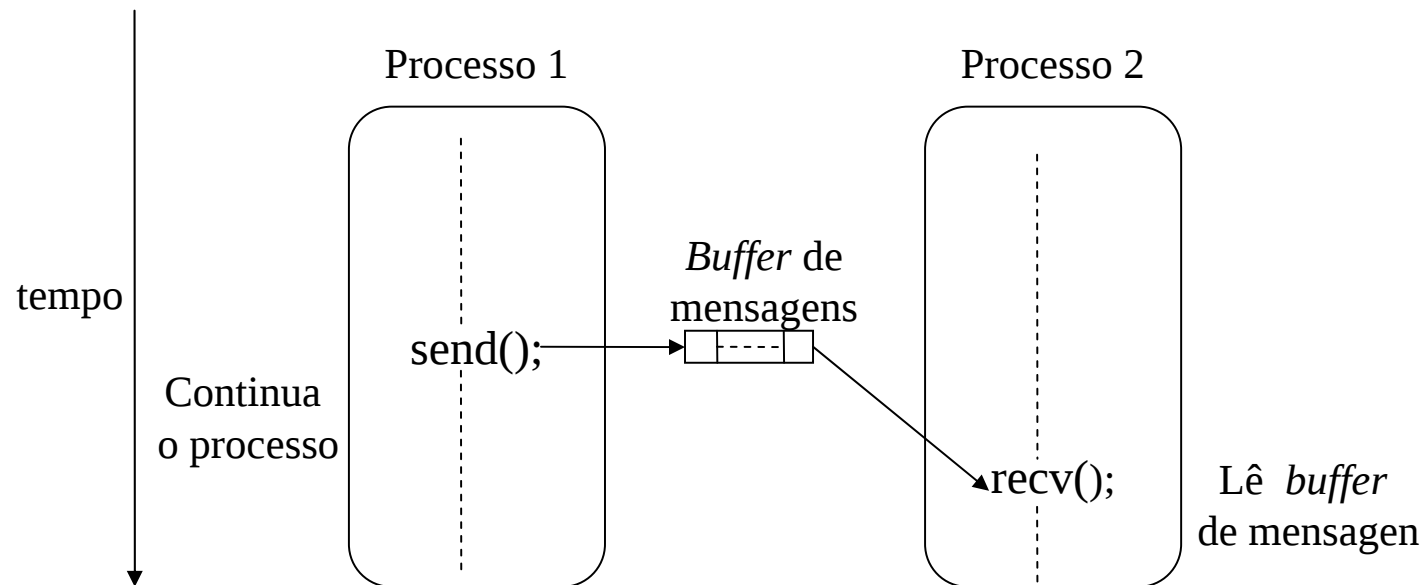
- **Rotinas que somente retornam quando a transferência da mensagem foi completada**
- **Uma rotina síncrona de envio deve esperar até que a mensagem completa possa ser aceita pelo processo receptor antes de enviar a mensagem**
- **Uma rotina síncrona de recebimento espera até que a mensagem que ela está esperando chegue**
- **Rotinas síncronas intrinsicamente realizam duas ações: transferem dados e sincronizam processos**
- **Sugere a existência de alguma forma de protocolo de sinalização**

Rotinas síncronas para recebimento e envio de mensagens usando protocolo de sinalização



Retorno das rotinas antes que transferência tenha sido efetuada

- **Necessita de um *buffer* para guardar a mensagem**

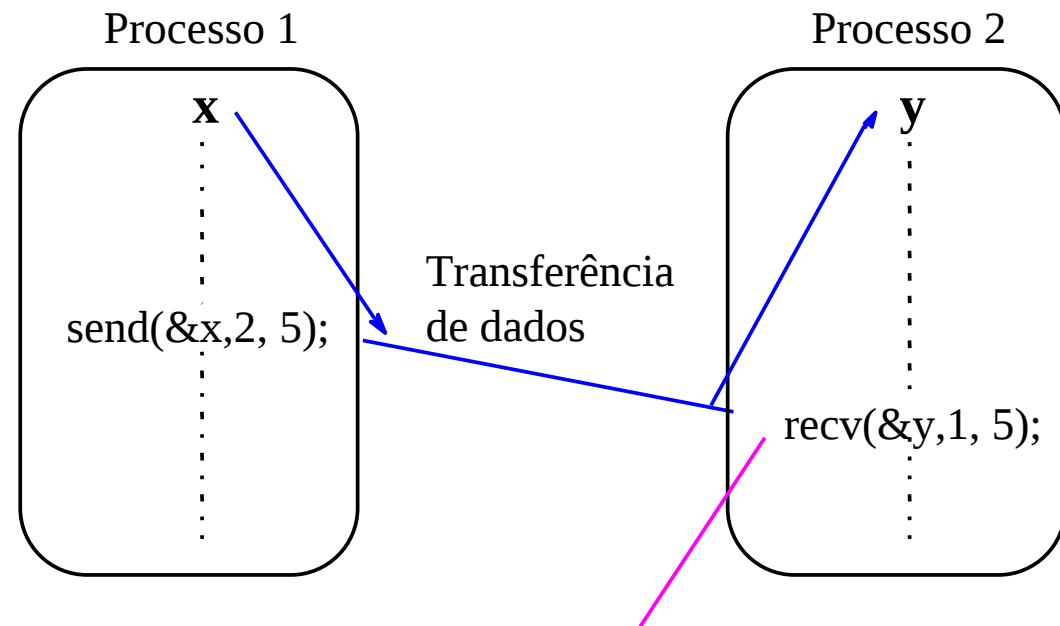


Índice da mensagem

- Utilizado para diferenciar os diversos tipos de mensagem que podem ser enviadas
- Índice enviado com a mensagem
- Utiliza-se um índice especial (*wild card*) quando não se requer casamento de índices das mensagens de modo que a rotina *recv()* aceitará mensagem enviada por qualquer rotina *send()*

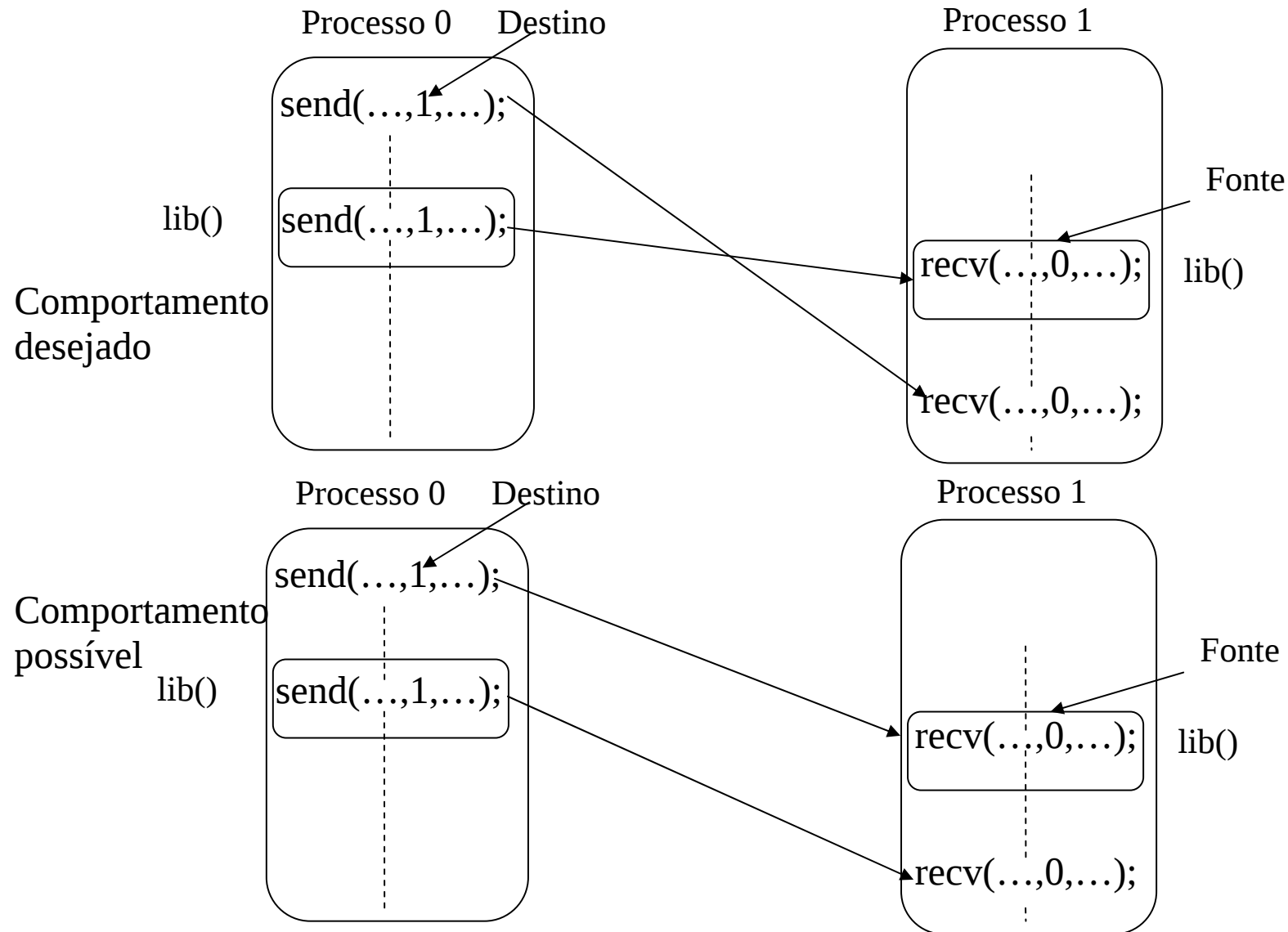
Exemplo de índice de mensagem

Envio de uma mensagem x , com índice de mensagem 5, do processo fonte 1 para o processo destino 2, armazenado-a em y :



Espera por uma mensagem do processo 1 com índice 5

Modo não seguro de envio de mensagens



Solução MPI : Communicators

- **Define o escopo das operações de comunicação**
- **Processos têm posições definidas associadas ao communicator**
- **Inicialmente todos os processos participam de um communicator universal chamado `MPI_COMM_WORLD` e cada processo possui uma única posição (0 a $p-1$), onde p é o número total de processos**
- **Outros communicators podem ser estabelecidos para grupo de processos**

Tipos de communicator

- Intracommunicator
 - **comunicação dentro de um grupo**
- Intercommunicator
 - **comunicação entre grupos**
- Um processo tem um único *rank* em um grupo, que varia de 0 a $m-1$, onde m é o número de processos pertencentes a um grupo
- Um processo pode ser membro de mais de um grupo
- Communicator *default*: *MPI_COMM_WORLD*
 - é o primeiro communicator de todos os processos da aplicação
 - conjunto de rotinas MPI para formar novos communicators

Utilizando o modelo SPMD

```
main (int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    .
    .
    MPI_Comm_Rank(MPI_COMM_WORLD, &myrank);
    if (myrank == 0)
        master();
    else
        slave();
    .
    .
    MPI_Finalize();
}
```

Variáveis locais e globais

- Qualquer declaração global será duplicada em cada processo
- Para não haver duplicação de variável, deve-se declará-la dentro do código executado somente pelo processo
- Exemplo:

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);  
if (myrank==0) {  
    int x, y;  
    .  
    .  
else if (myrank==1) {  
    int x, y;  
    .  
    .  
}
```

Rotinas com bloqueio

- Retornam quando estão completas localmente (local utilizado para guardar a mensagem pode ser utilizado novamente sem afetar envio da mensagem)
- Envia a mensagem e retornam – não significa que a mensagem foi recebida, significa somente que o processo está livre para continuar a execução sem afetar a mensagem

Comunicação ponto-a-ponto

- São utilizadas rotinas para envio e recebimento com especificação de índices de mensagens e communicators
- Caracteres que indicam “qualquer índice” ou “qualquer receptor” podem ser utilizados
 - MPI_ANY_TAG: para não especificar índice
 - Para receber de qualquer fonte: **MPI_ANY_SOURCE**
- Tipo de dados é enviado como parâmetro na mensagem

Rotinas com bloqueio

MPI_Send (buf, count, datatype, dest, tag, comm)

Endereço do
buffer de envio

Número de itens
a enviar

Tipo de dados
de cada item

Rank do processo
destino

Índice da
mensagem

Communicator

MPI_Recv (buf, count, datatype, src, tag, comm, status)

Endereço do
buffer de recepção

Número máximo
de itens a receber

Tipo de dados
de cada item

Rank do processo
fonte

Índice da
mensagem

Status após
operação

Communicator

Exemplo de rotina com bloqueio

Envio de um inteiro x do processo 0 para o processo 1

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
    int x;
    MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
}
else if (myrank == 1) {
    int x;
    MPI_Recv(&x, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD, status);
}
```

Rotinas sem bloqueio

- **MPI_Isend: retorna imediatamente antes do local da mensagem estar seguro**
- **MPI_Irecv: retorna mesmo que não tenha mensagem a receber**
- **Formatos:**
 - MPI_Isend(buf, count, datatype, dest, tag, comm, request)
 - MPI_Irecv(buf, count, datatype, source, tag, comm, request)
- **MPI_Wait(): retorna somente após término da operação**
- **MPI_Test(): retorna com um flag indicando se a operação já foi completada**
- **Parâmetro *request* indica operação a ser verificada**

Exemplo de rotina sem bloqueio

Envio de um inteiro x do processo 0 para o processo 1

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
    int x;
    MPI_Isend(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD, req1);
    processa();
    MPI_Wait(req1, status);
}
else if (myrank == 1) {
    int x;
    MPI_Recv(&x, 0, MPI_INT, 1, msgtag, MPI_COMM_WORLD, status);
}
```

Rotinas para tratar recebimento não bloqueante

MPI_IPROBE	verifica mensagens pendentes
MPI_PROBE	espera por mensagens pendentes
MPI_GET_COUNT	número de elementos em 1 mensagem
MPI_PROBE (source, tag, comm, &status)	=> status
MPI_GET_COUNT (status, datatype, &count)	=> tamanho da mensagem
status.MPI_SOURCE	=> identificação do remetente
status.MPI_TAG	=> tag da mensagem

Exemplo: Verifica mensagem pendente

```
int buf[1], flag, source, minimum;
while ( ...) {
    MPI_IPROBE(MPI_ANY_SOURCE, NEW_MINIMUM, comm,
               &flag, &status);
    if (flag) {
        /* obtém novo mínimo */
        source = status.MPI_SOURCE;
        MPI_RECV (buf, 1, MPI_INT, source, NEW_MINIMUM, comm,
                  &status);
        minimum = buf[0];
    }
    ... /* execute */
}
```

Exemplo: Recebendo mensagem de tamanho desconhecido

```
int count, *buf, source;  
MPI_PROBE(MPI_ANY_SOURCE, 0, comm, &status);  
source = status.MPI_SOURCE;  
MPI_GET_COUNT (status, MPI_INT, &count);  
buf = malloc (count * sizeof (int));  
MPI_RECV (buf, count, MPI_INT, source, 0, comm,  
          &status);
```

Modos de envio

- Standard:
 - Não assume a existência de uma rotina correspondente de recebimento
 - Quantidade de memória para bufferização não definida
 - Se existe bufferização, envio pode ser completado antes de ocorrer a rotina de recebimento
- Bufferizado
 - O envio pode ser inicializado e retornar antes de uma rotina correspondente de recebimento
 - A utilização do buffer deve ser especificada via as rotinas `MPI_Buffer_attach()` e `MPI_Buffer_detach()`

Modos de envio

- Síncrono:
 - **As rotinas de envio e recebimento podem iniciar seus procedimentos uma antes da outra mas têm que finalizá-los juntas**
- Pronto (ready):
 - **O envio da mensagem só pode ser iniciado se existe uma rotina de recebimento correspondente**

Modos de envio

- Os quatro modos podem ser aplicados para rotinas de envio com e sem bloqueio
- O modo standard é o único disponível para rotinas de recebimento com e sem bloqueio
- Qualquer tipo de rotina de envio pode ser utilizada com qualquer tipo de rotina de recebimento