

Objetivos

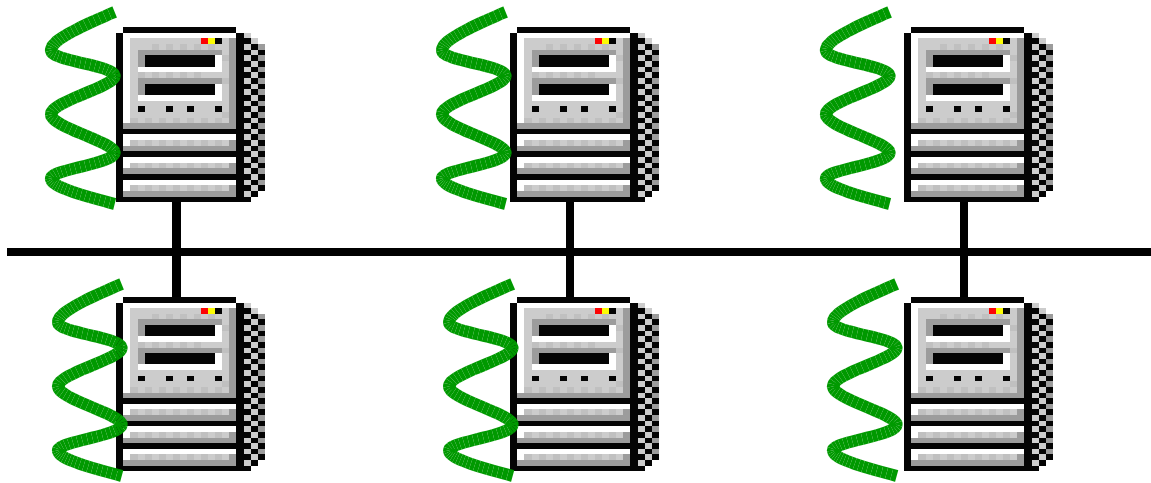
A programação concorrente para o
processamento de alto desempenho

Exploração do poder computacional de
aglomerados

Ferramentas padrões para o processamento
paralelo e distribuído

Objetivos

Explorar a **concorrência entre-nós**



Objetivos

Explorar a **concorrência intra-nó**

Criação de uma **máquina virtual**

Estudo de caso: **MPI**
LAM – Local Area Machine

MPI

Message Passing Interface

Comunicação ponto a ponto
Comunicação em grupo
Tipos de dados derivados

125 rotinas
Padrão de fato

MPI

Message Passing Interface

Antes de 1980: Bibliotecas proprietárias

1989: Parallel Virtual Machine (PVM) (Oak Ridge National Lab)

1992: MPI Primeiros passos MPIForum (40 organizações)

1994: MPI v 1.0

1997: MPI v 2.0 Criação dinâmica de processos

Today: Mais utilizado

Memória Distribuída

MPI - LAM

Programação SPMD

Máquina virtual

LAM básico

Mensagens

Comunicação de grupo

Memória Distribuída

Programação SPMD

Vários processos são criados para executar um mesmo programa.

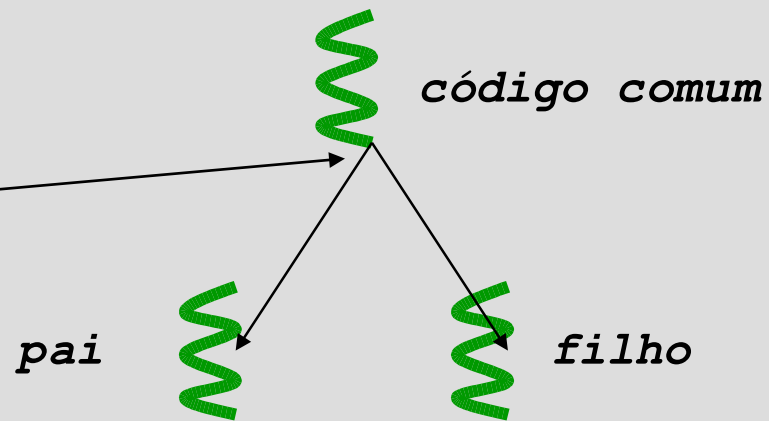
Cada processo possui seu próprio conjunto de dados, o que pode implicar em execuções totalmente diferentes

Memória Distribuída

Programação SPMD

O fork no Unix

```
main() {  
  int id;  
  código comum  
  id = fork();  
  if( id != 0 )  
    { código pai }  
  else  
    { código filho }  
}
```



Memória Distribuída

Programação SPMD

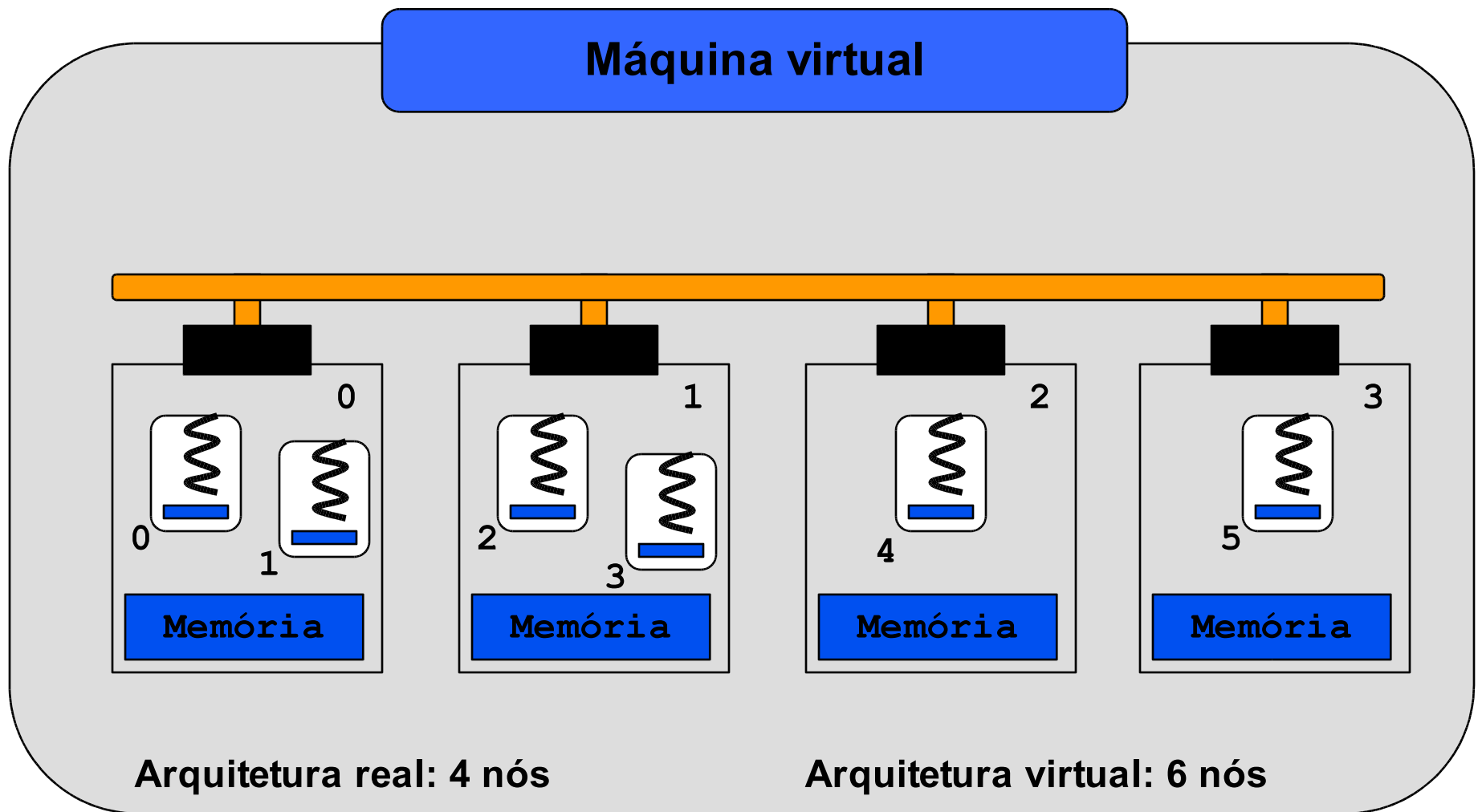
O fork no Unix

```
main() {  
  int id;  
  código comum  
  id = fork();  
  if( id != 0 )  
    { código pai }  
  else  
    { código filho }  
}
```

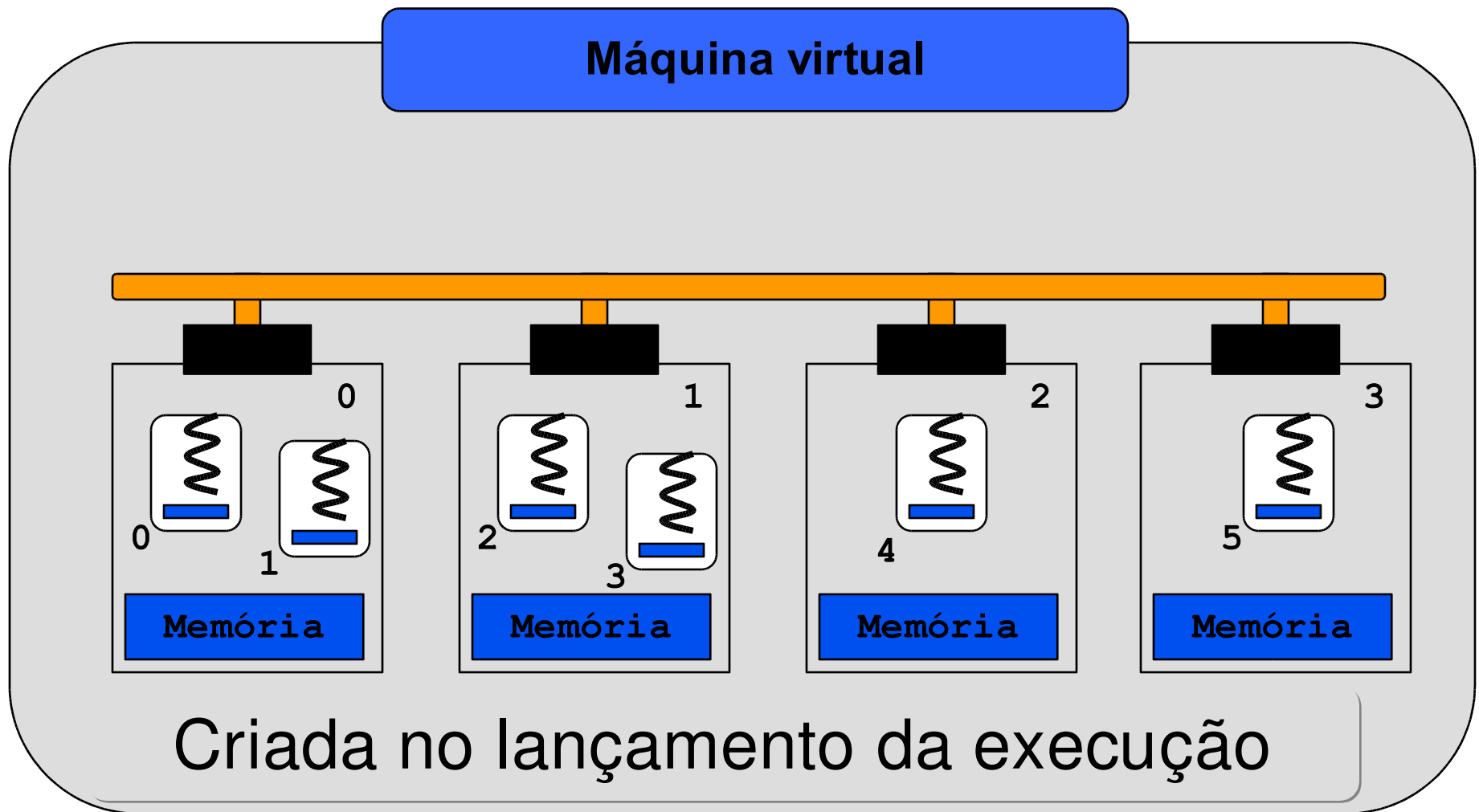
Diferença fundamental:

Em MPI os processos criados não possuem cópias idênticas da área de dados e diversas cópias podem ser criadas simultaneamente

Memória Distribuída



Memória Distribuída



Memória Distribuída

LAM básico

Não é encontrado em todas as distribuições Linux

Processo de make

Memória Distribuída



Memória Distribuída

LAM básico

Manipulação da máquina virtual

lamboot

wipe

mpirun

Inicialização do suporte
à máquina virtual

lamboot maqs

Memória Distribuída

LAM básico

Manipulação da máquina virtual

`lamboot`

`wipe`

`mpirun`

```
$ more maqs
```

```
clu0
```

```
clu1
```

```
clu2
```

```
clu3
```

```
$
```

Memória Distribuída

LAM básico

Manipulação da máquina virtual

lamboot

wipe

mpirun

Shutdown do suporte à
máquina virtual

wipe maqs

Memória Distribuída

LAM básico

Manipulação da máquina virtual

lamboot

wipe

mpirun

Cria a máquina virtual e inicia a execução

mpirun -c 6 prog

Nós virtuais

Memória Distribuída

LAM básico

Primeiro programa

```
#include <mpi h>
```

```
MPI init( &argc, &argv );
```

```
MPI Comm rank( MPI COMM WORLD, &myrank );
```

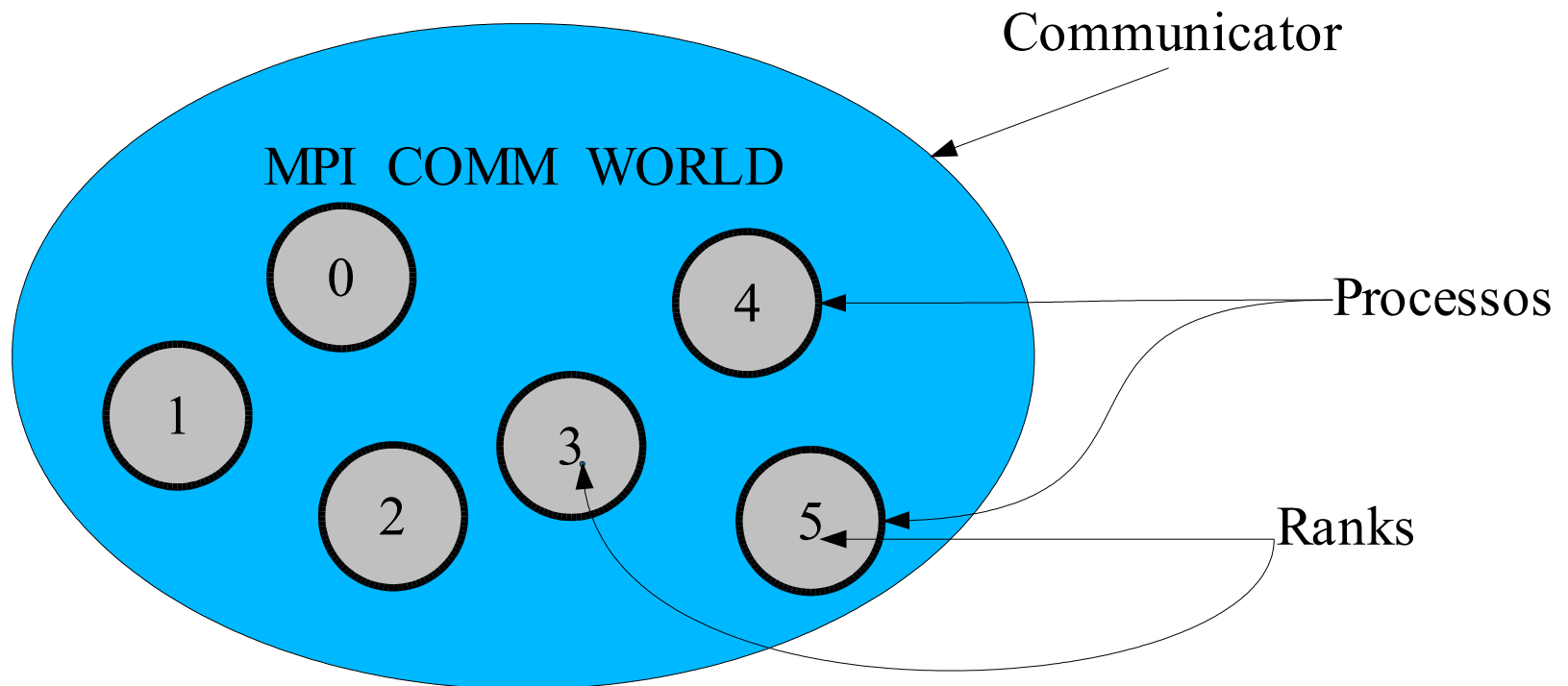
```
MPI Comm size( MPI COMM WORLD, &size );
```

```
MPI Finalize();
```

Memória Distribuída

LAM básico

Rank e Communicator



Memória Distribuída

LAM básico

Primeiro programa

```
#include <mpi h>
int main( int argc, char** argv ) {
    int myrank, size;
    MPI init( &argc, &argv );
    MPI Comm rank( MPI COMM WORLD, &myrank );
    MPI Comm size( MPI COMM WORLD, &size );
    if( myrank == 0 ) { código nó 0 }
    else { código dos outros nós }
    MPI Finalize();
}
```

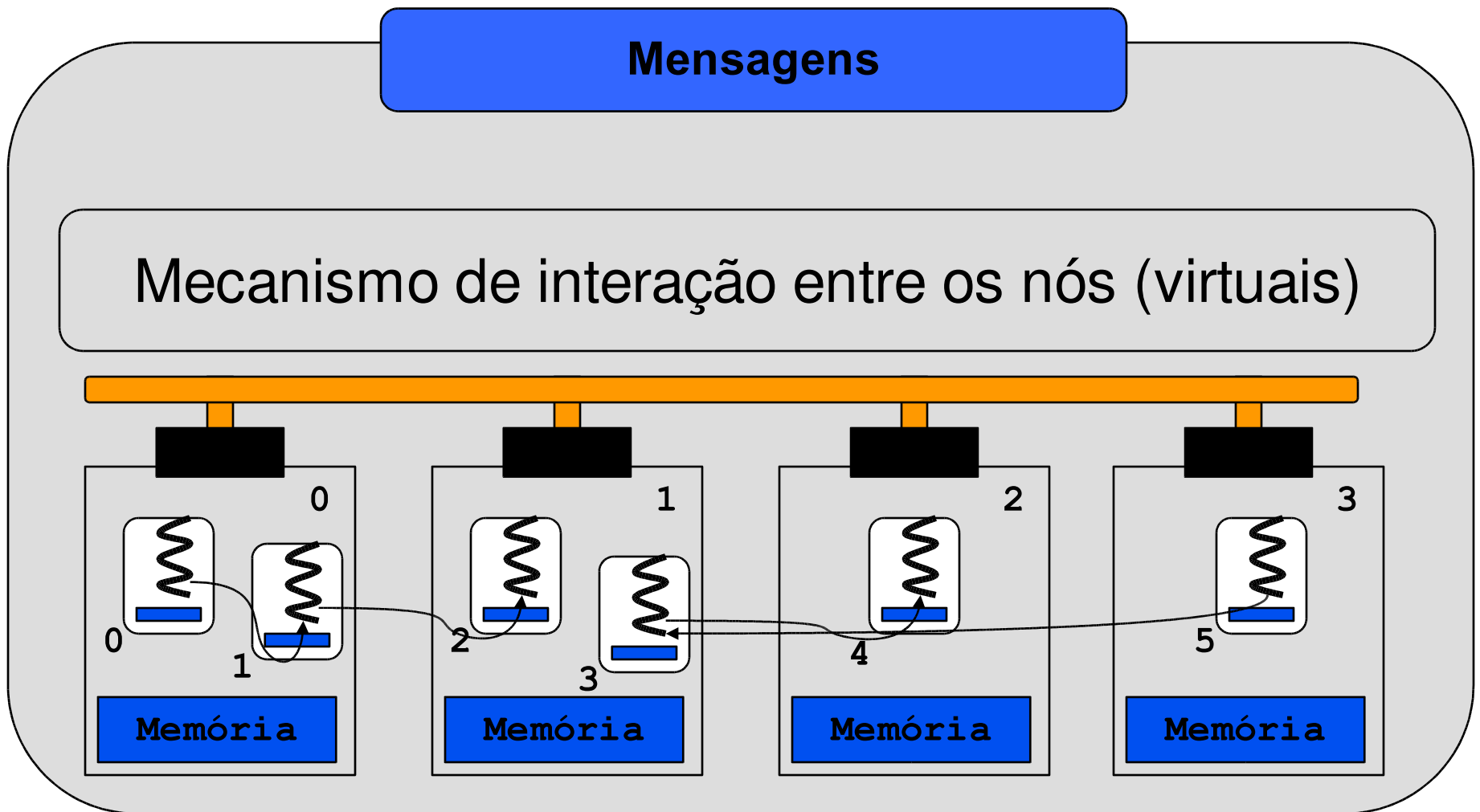
Memória Distribuída

LAM básico

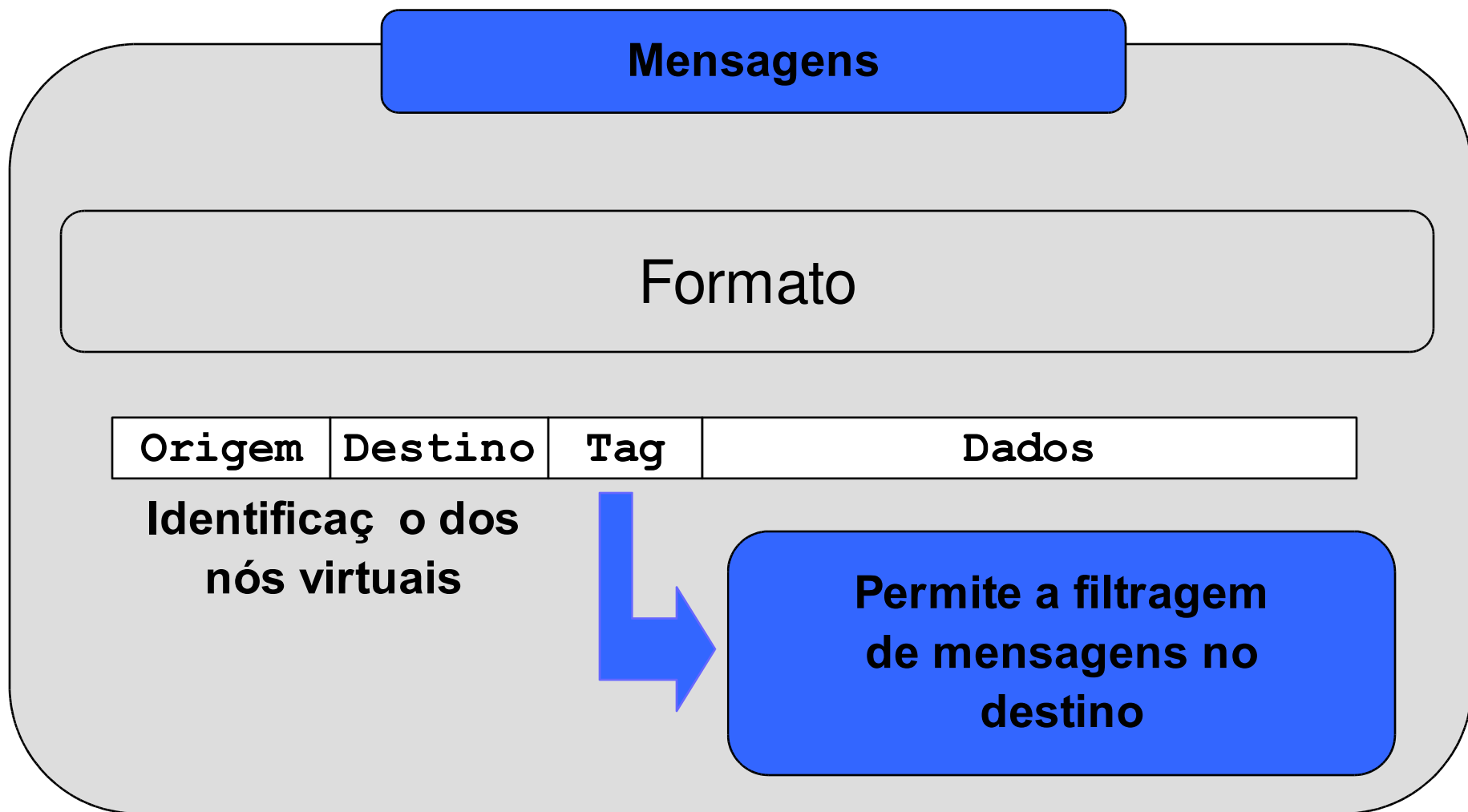
Uma sessão típica

```
$ mpicc prog.c -o prog  
$ lamboot maqs  
$ mpirun -c 6 prog  
$ wipe maqs
```

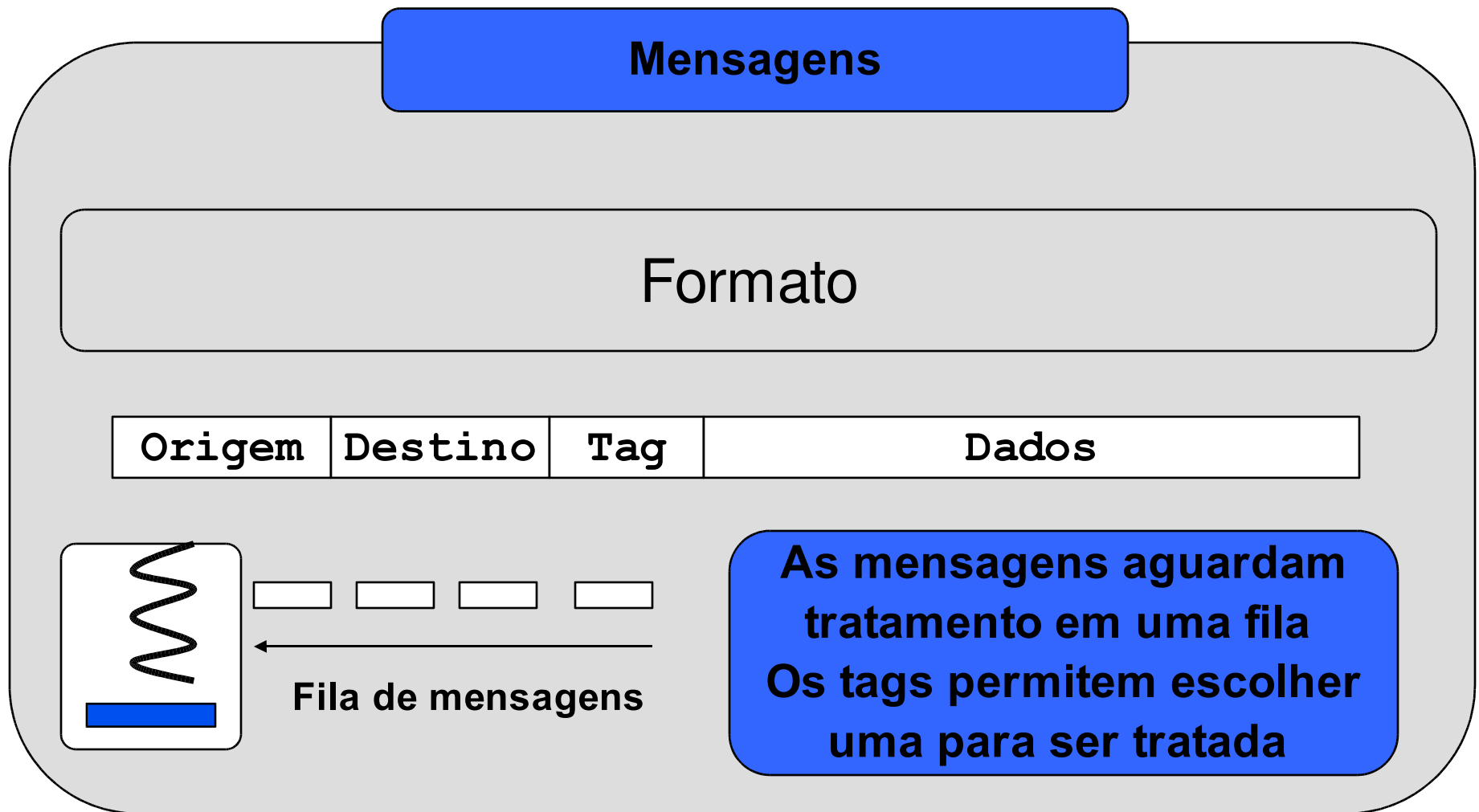
Memória Distribuída



Memória Distribuída



Memória Distribuída



Memória Distribuída

Mensagens

Serviços básicos

Envio e recebimento síncrono:

MPI Send

Retorna quando o envio foi completado localmente.
Não quer dizer que tenha sido recebida!!

MPI Recv

Retorna quando uma mensagem for recebida.

Memória Distribuída

Mensagens

Serviços básicos

Envio:

```
int MPIS end( void *buf, int cont,  
             MPI Datatype tipo, int dest, int tag,  
             MPI Comm grupo );
```

Memória Distribuída

Mensagens

Serviços básicos

Envio:

```
int MPIS end( void *buf, int cont,  
             MPI Datatype tipo, int dest, int tag,  
             MPI Comm grupo );
```

Recepção:

```
int MPIR ecv( void *buf, int cont,  
             MPI Datatype tipo, int origem, int tag,  
             MPI Comm grupo, MPIS tatus *st );
```

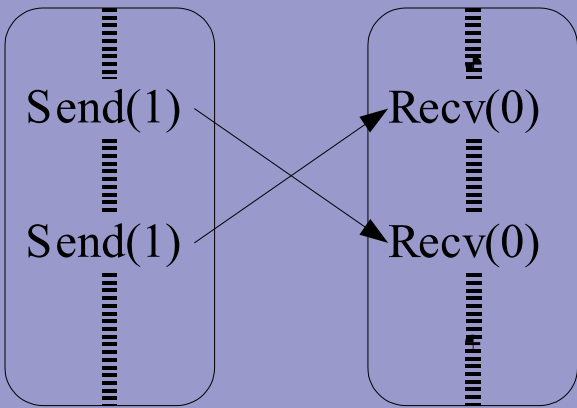
Memória Distribuída

Mensagens

Serviços básicos

Envio e Recepção: A ordem entre dois processos é respeitada

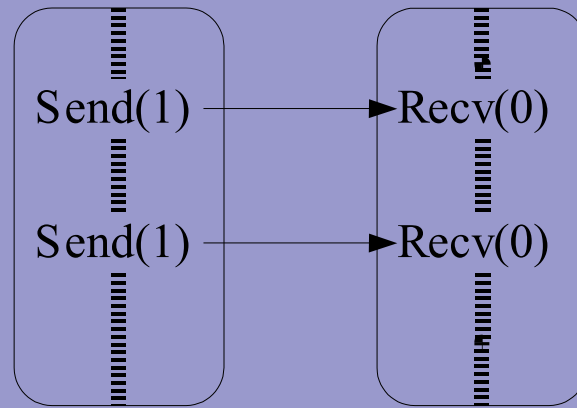
Não ocorre



Rank 0

Rank 1

Ordem causal



Rank 0

Rank 1

Memória Distribuída

Mensagens

MPI_Datatype

MPI_CHAR
MPI_DOUBLE
MPI_FLOAT
MPI_INT
MPI_LONG
MPI_LONG_DOUBLE

MPI_SHORT
MPI_UNSIGNED_CHAR
MPI_UNSIGNED
MPI_UNSIGNED_LONG
MPI_UNSIGNED_SHORT

Memória Distribuída

Mensagens

Exemplo

Serviços básicos

Envio:

```
int vet[10];  
MPI Send( vet, 10, MPI_INT, 4, MPI_ANY_TAG,  
          MPI_COMM_WORLD );
```

Memória Distribuída

Mensagens

Serviços básicos

Envio:

```
int vet[10];  
MPI Send( vet, 10, MPI_INT, 4, MPI_ANY_TAG,  
          MPI_COMM_WORLD );
```

Recepção:

```
MPI Status st;  
int vet[ 10] ;  
MPI Recv( vet, 10, MPI_INT, MPI_ANY_SOURCE,  
          MPI_ANY_TAG, MPI_COMM_WORLD, &st );
```

Memória Distribuída

Mensagens

Serviços básicos

Existem igualmente serviços assíncronos

MPI Isend

MPI Irecv

Isend : retorna mesmo que o dado não tenha sido enviado.

Irecv : retorna mesmo se não há mensagem.

Memória Distribuída

Mensagens

Serviços básicos

Envio:

```
MPI Request req; MPI Status st;  
int vet[10];  
MPI Isend( vet, 10, MPI INT, 4, MPI ANY TAG, MPI COMM WORLD, &req );  
MPI Wait( &req, &st );
```

Recepção:

```
MPI Request req; MPI Status st;  
int vet[ 10] ;  
MPI Irecv( vet, 10, MPI INT, MPI ANY SOURCE, MPI ANY TAG,  
          MPI COMM WORLD, &st );  
MPI Wait( &req, &st );
```

Memória Distribuída

Mensagens

Comunicação de grupo

Barreira

Broadcast: um envia, todos recebem

Reduce: todos enviam, um recebe aplicando uma operação

Gather: recebe em um array dados distribuídos

Scatter: distribui dados de um array

All-to-all: troca de dados entre todos

Scan: uma redução pré-fixada

Memória Distribuída

Mensagens

Comunicação de grupo

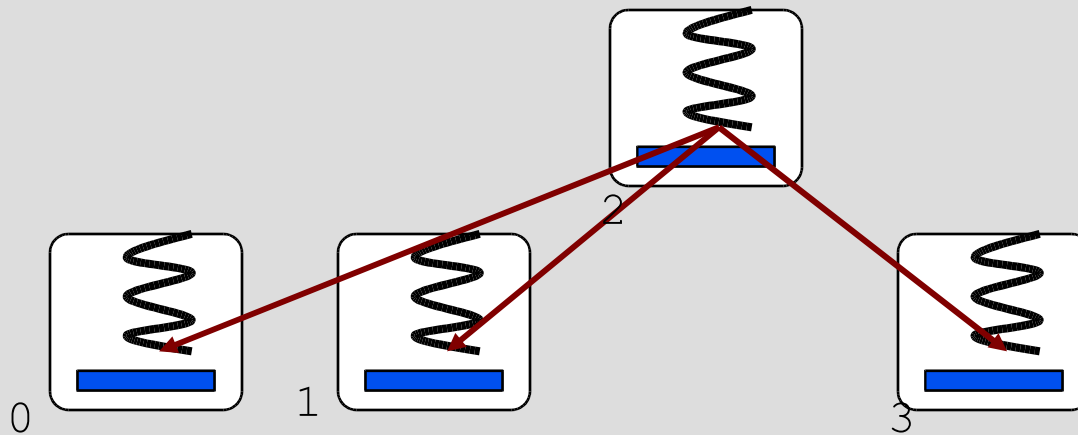
Envio e recepção:

```
MPI Bcast( void *buf, int cont, MPI Datatype t,  
           int raiz, MPI Comm grupo);
```

Memória Distribuída

Mensagens

Comunicação de grupo



```
MPI Bcast( dta, 1, MPI INT, 2, MPI COMM WORLD );
```

Memória Distribuída

Mensagens

Comunicação de grupo

Envio e recepção:

```
MPI Reduce (void *sendbuf, void *recvbuf,  
             int count,  
             MPI Datatype t, MPI Op op,  
             int raiz, MPI Comm grupo);
```

Memória Distribuída

Mensagens

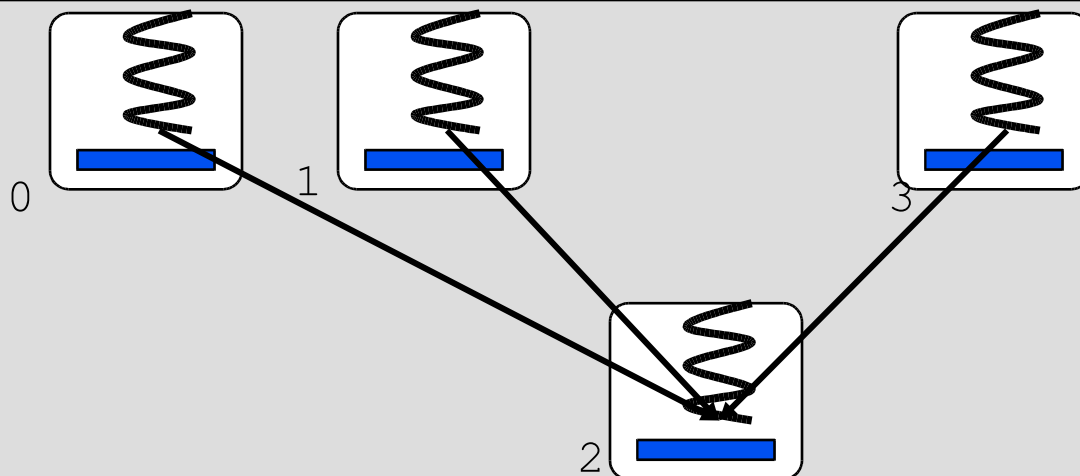
MPI_Op

MPI_BAND	MPI_MAX
MPI_BOR	MPI_MAXLOC
MPI_BXOR	MPI_MIN
MPI_LAND	MPI_MINLOC
MPI_LOR	MPI_PROD
MPI_LXOR	MPI_SUM

Memória Distribuída

Mensagens

Comunicação de grupo



```
MPI Reduce(src, dst, 1, MPI_INT, MPI_SUM, 2, MPI_COMM_WORLD);
```

Memória Distribuída

Mensagens

Comunicação de grupo

Envio e recepção:

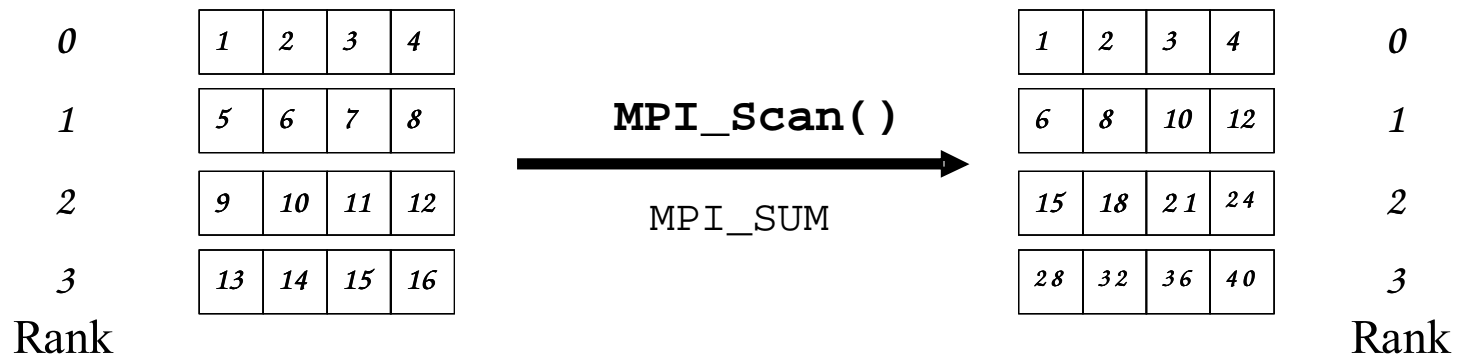
```
MPI Scan(void *sendbuf, void *recvbuf, int count,  
          MPI Datatype t, MPI Op op, MPI Comm grupo);
```

Memória Distribuída

Mensagens

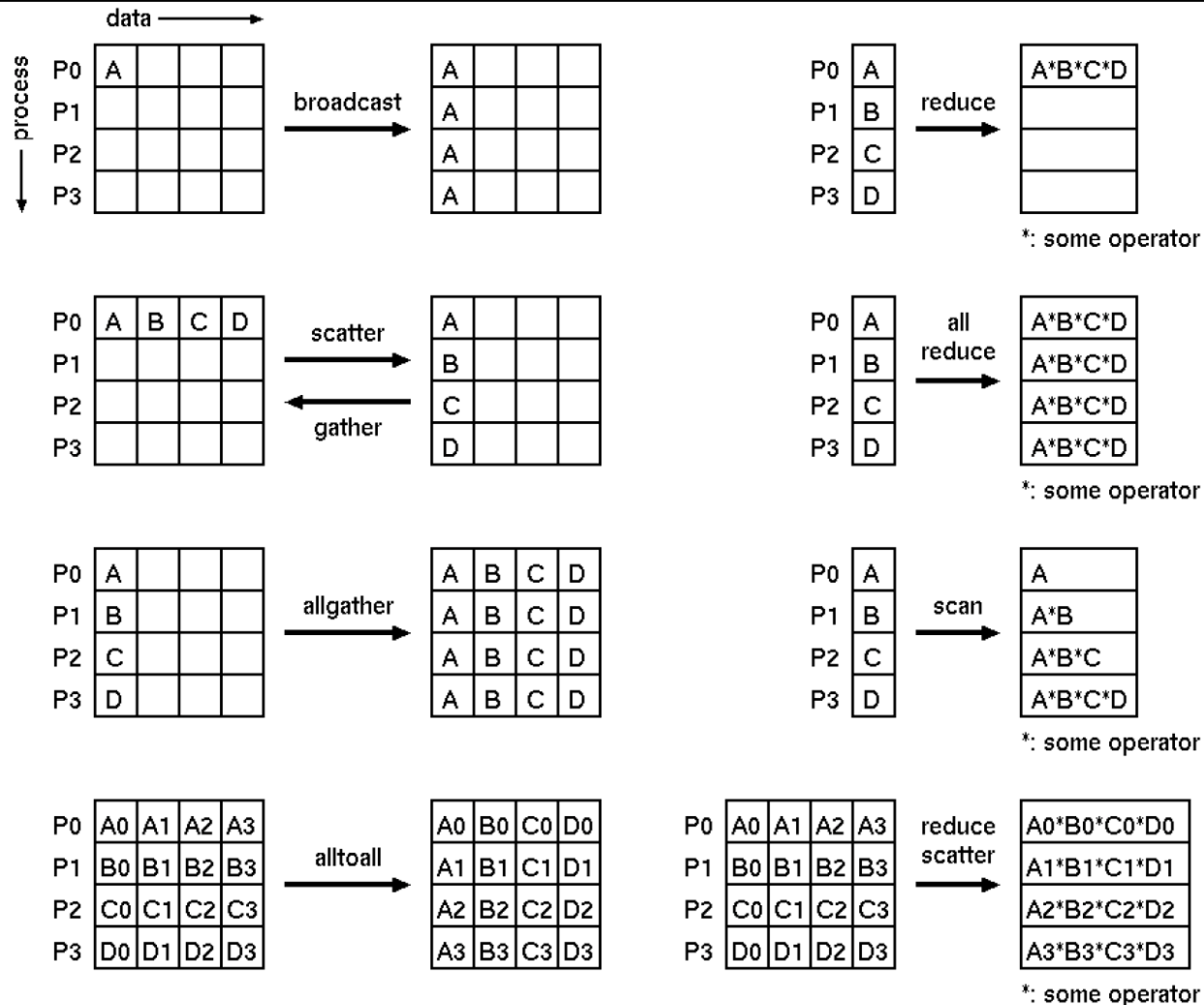
Comunicação de grupo

MPI Scan



Memória Distribuída

Padrões de Comunicação de grupo



Memória Distribuída

Tipos de dados derivados

No envio de mensagens, pode ser necessário enviar tipos de dados definidos pelo programador. Dois casos podem ser observados:

- Uma seqüência de dados do mesmo tipo
 - *Contiguous derived data types*
- Dados são compostos por diferentes tipos
 - *Structured derived data types*

Memória Distribuída

Tipos de dados derivados

Primitivas principais:

- *MPI Type contiguous()* - Constroi um tipo contínuo
- *MPI Type struct()* - Constroi um tipo estruturado
- *MPI Type vector()*
- *MPI Type indexed()*

Primitivas auxiliares:

- *MPI Type extent()* - Retorna o tamanho de um tipo
- *MPI Type commit()* - autoriza o uso

Memória Distribuída

Tipos de dados derivados

Exemplo: Contiguous derived data type

```
#define SIZE 9
int numbers[SIZE] = {2,5,6,8,9,7,1,3,10};
int numberslave[SIZE];
MPI Datatype mytype;
MPI Init(&argc, &argv);
MPI Type contiguous (SIZE/numproc, MPI INT, &mytype);
MPI Type commit(&mytype):
if (myid == 0)
    for (i=1; i<numproc; ++i)
        MPI Send(&numbers[(SIZE/numproc)*i],1,mytype, MPI COMM WORLD);
else
    MPI Recv(&numberslave, 1, mytype, 0, 0, MPI COMM WORLD, &Stat);
```

```
MPI Type contiguous ( int count,
                      MPI Datatype orig,
                      MPI Datatype *novo );
```

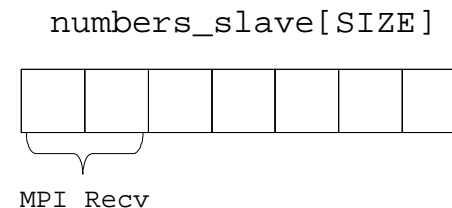
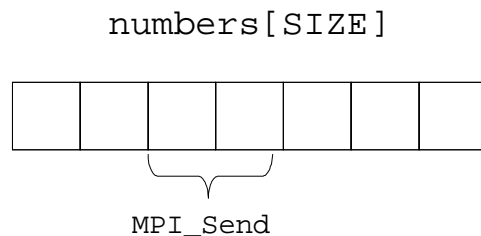
Memória Distribuída

Tipos de dados derivados

Exemplo: Contiguous derived data type

```
#define SIZE 9
int numbers[SIZE] = {2,5,6,8,9,7,1,3,10};
int numbers slave[SIZE];
MPI Datatype mytype;
MPI Init(&argc, &argv);
MPI Type contiguous (SIZE/numproc, MPI INT, &mytype);
MPI Type commit(&mytype);
if (myid == 0)
    for (i=1; i<numproc; ++i)
        MPI Send(&numbers[(SIZE/numproc)*i], 1, mytype, MPIC OMM WORLD);
else
    MPI Recv(&numbers slave, 1, mytype, 0, 0, MPI COMM WORLD, &Stat);
```

```
MPI Type contiguous ( int count,
                      MPI Datatype orig,
                      MPI Datatype *novo );
```



Memória Distribuída

Tipos de dados derivados

Exemplo: Contiguous derived data type

```
int MPI_Type_struct( int count, int blocklens[], MPI_Aint indices[],  
                    MPI_Datatype old types[], MPI_Datatype *newtype )
```

Entrada:

<code>count</code>	número de blocos (integer)
<code>blocklens</code>	número de elementos em cada bloco (array)
<code>indices</code>	deslocamento para cada bloco no dado (array)
<code>old types</code>	MPI_Datatype associado a cada bloco (array)

Memória Distribuída

Tipos de dados derivados

Exemplo: Contiguous derived data type

```
typedef struct {int x, y; float z;} message;
message slave buf[SIZE], master buf[SIZE];
MPI Datatype mytype;
MPI Datatype oldtypes[2] = {MPI INT, MPI FLOAT};
int blocklens[2] = {2,1};
MPI Aint indicies[2], length;

MPI Type extent(MPI INT, &length);
indicies[0]=0;
indicies[1]=2*length;
MPI Type struct(2, blocklens, indicies, oldtypes, &mytype);
MPI Type commit(&mytype);
```