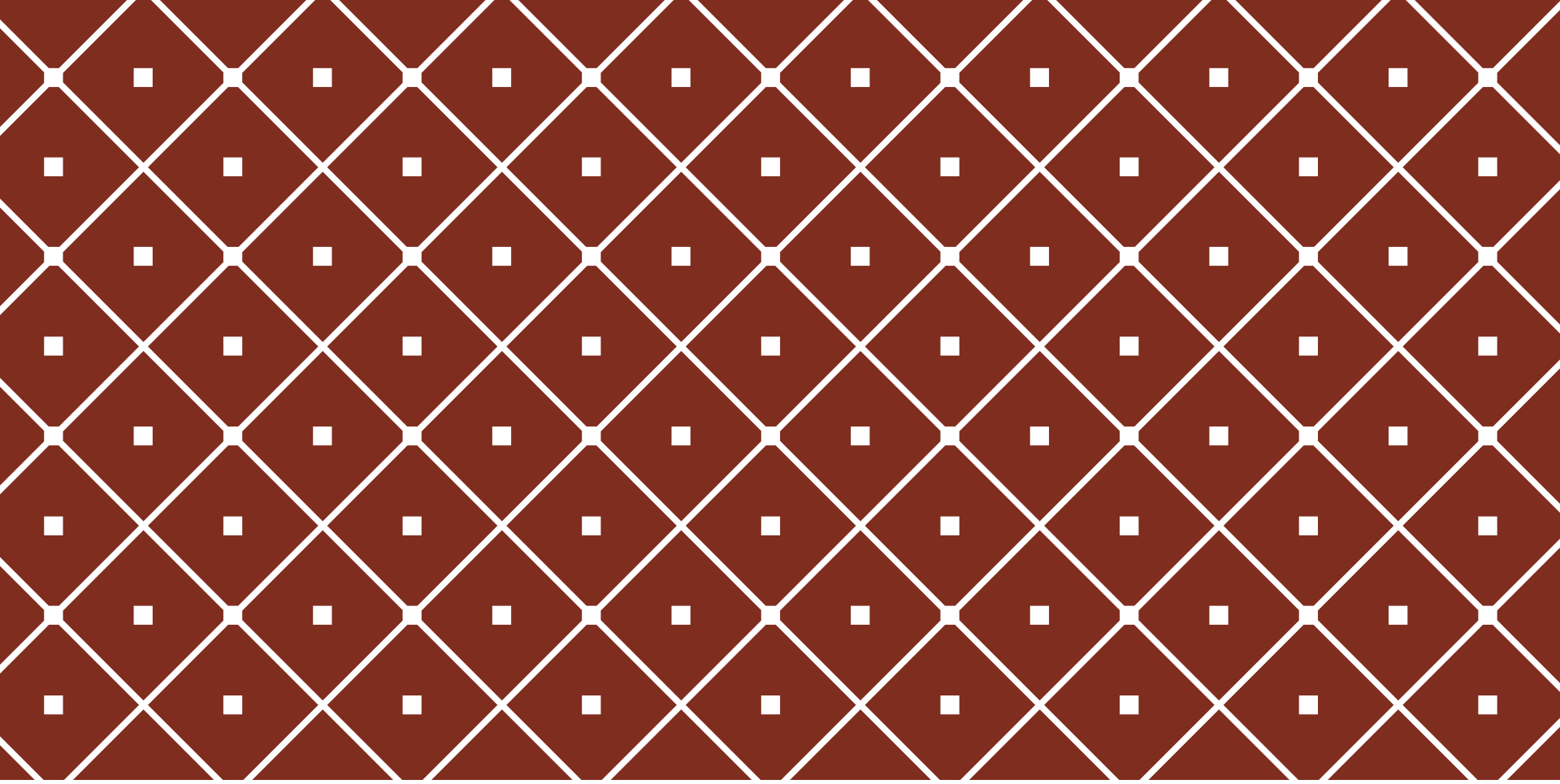


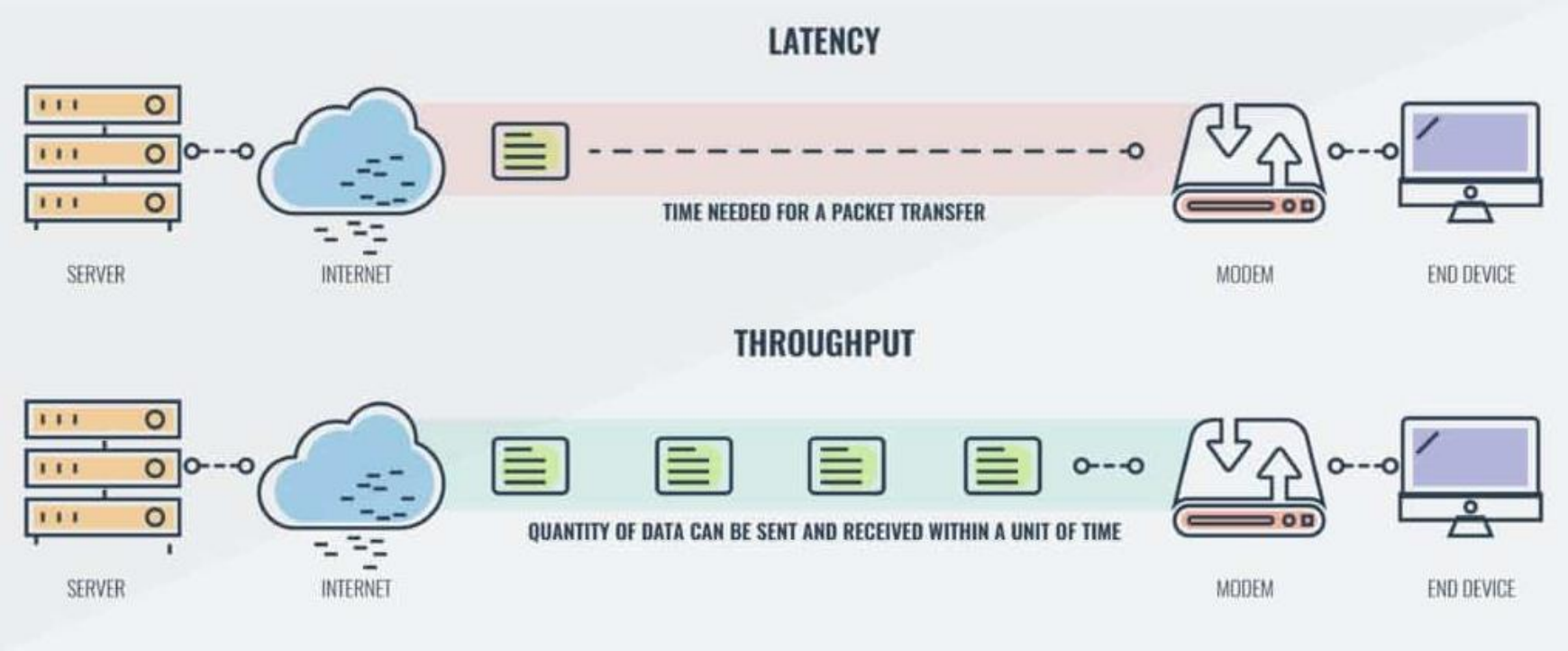
PROGRAMAÇÃO PARALELA MPI 02 –DADOS DERIVADOS

Marco A. Zanata Alves

Slides baseados nos slides de Ricardo Rocha – DCC FCUP (2009/2010)
Programação Paralela e Distribuída

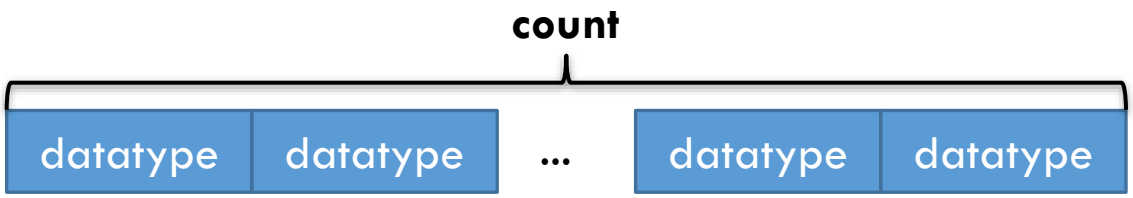
AGRUPAR DADOS PARA COMUNICAÇÃO

Em programação com troca de mensagens, uma heurística natural para maximizar a performance do sistema é minimizar o número de mensagens a trocar.



AGRUPAR DADOS PARA COMUNICAÇÃO

Por padrão, todas as funções de envio e recepção de mensagens permitem agrupar numa mesma mensagem dados do mesmo tipo guardados em posições contíguas de memória.



Para além desta funcionalidade básica o MPI permite:

- Definir novos tipos de dados que agrupam dados de vários tipos.
- Agrupar dados espalhados na memória
- Empacotar e desempacotar dados para/de um buffer.

TIPOS DERIVADOS

A definição de novos tipos de dados é feita em tempo de execução:

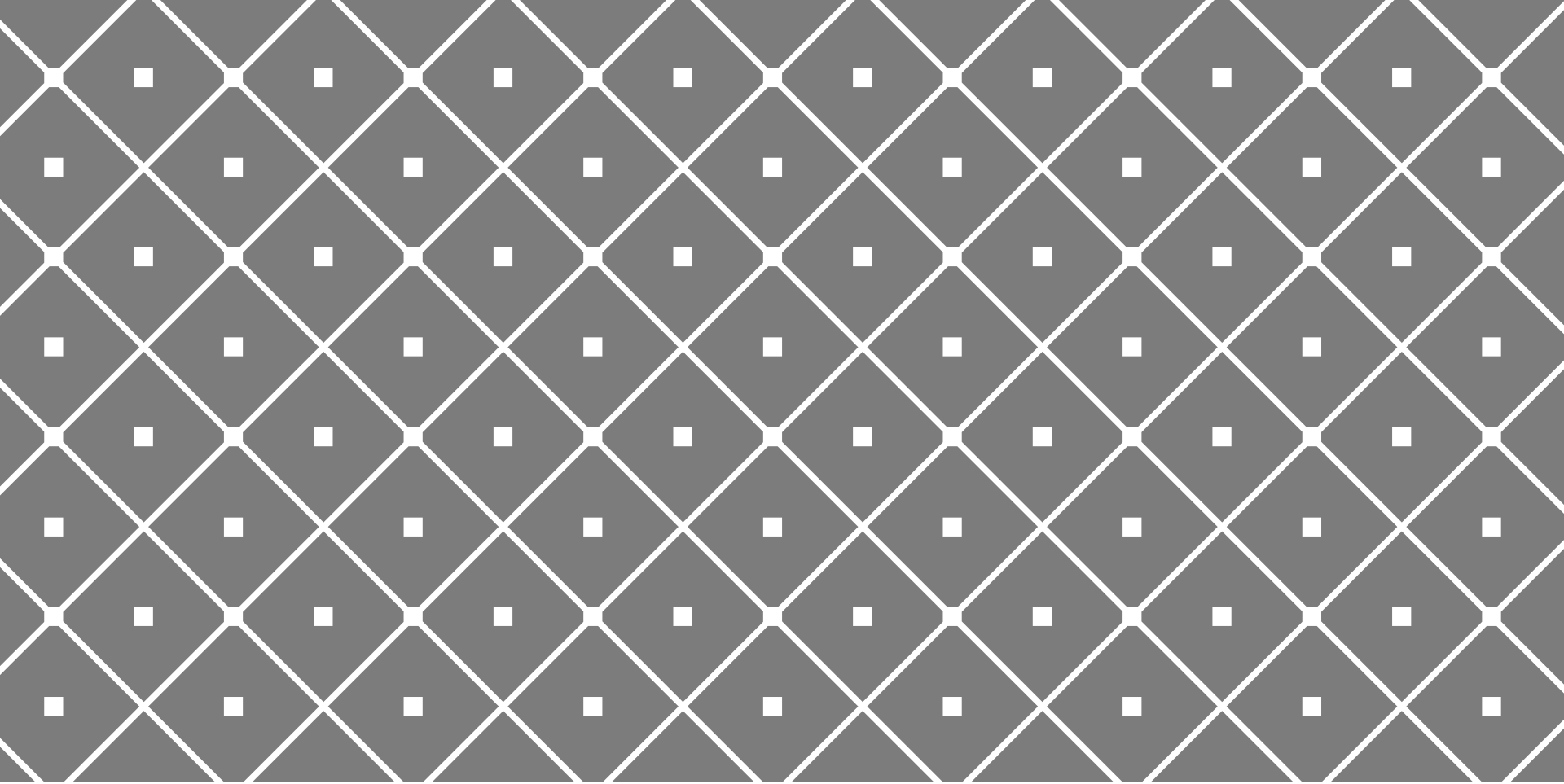
1. Inicialmente, os processos devem construir o **novo tipo derivado**, **feita a partir dos tipos de dados básicos** que o MPI define.
   - `MPI_Type_contiguous, MPI_Type_vector, MPI_Type_struct,`
   - `MPI_Type_indexed, MPI_Type_hvector, MPI_Type_hindexed`
2. Em seguida, devem **certificar** perante o ambiente de execução do MPI **a existência** do novo tipo derivado.
   - `MPI_Type_Commit`
3. Depois, o novo tipo de dados **pode ser utilizado** normalmente (send, receives, etc.).
4. Após a sua utilização, cada processo **libera a certificação** feita.
   - `MPI_Type_Free(newtype, ierr), MPI_Type_free(type)`

TIPOS DERIVADOS

A construção de tipos derivados é custosa, e só deve ser utilizada quando o número de mensagens a trocar for significativo.

Os novos tipos de dados são utilizados nas funções de envio e recepção de mensagens tal como os outros tipos básicos.

Para tal é necessário que ambos os processos emissor e receptor tenham certificado o novo tipo derivado. Normalmente, isso é feito na parte do código que é comum a ambos os processos.



TYPE CONTIGUOUS

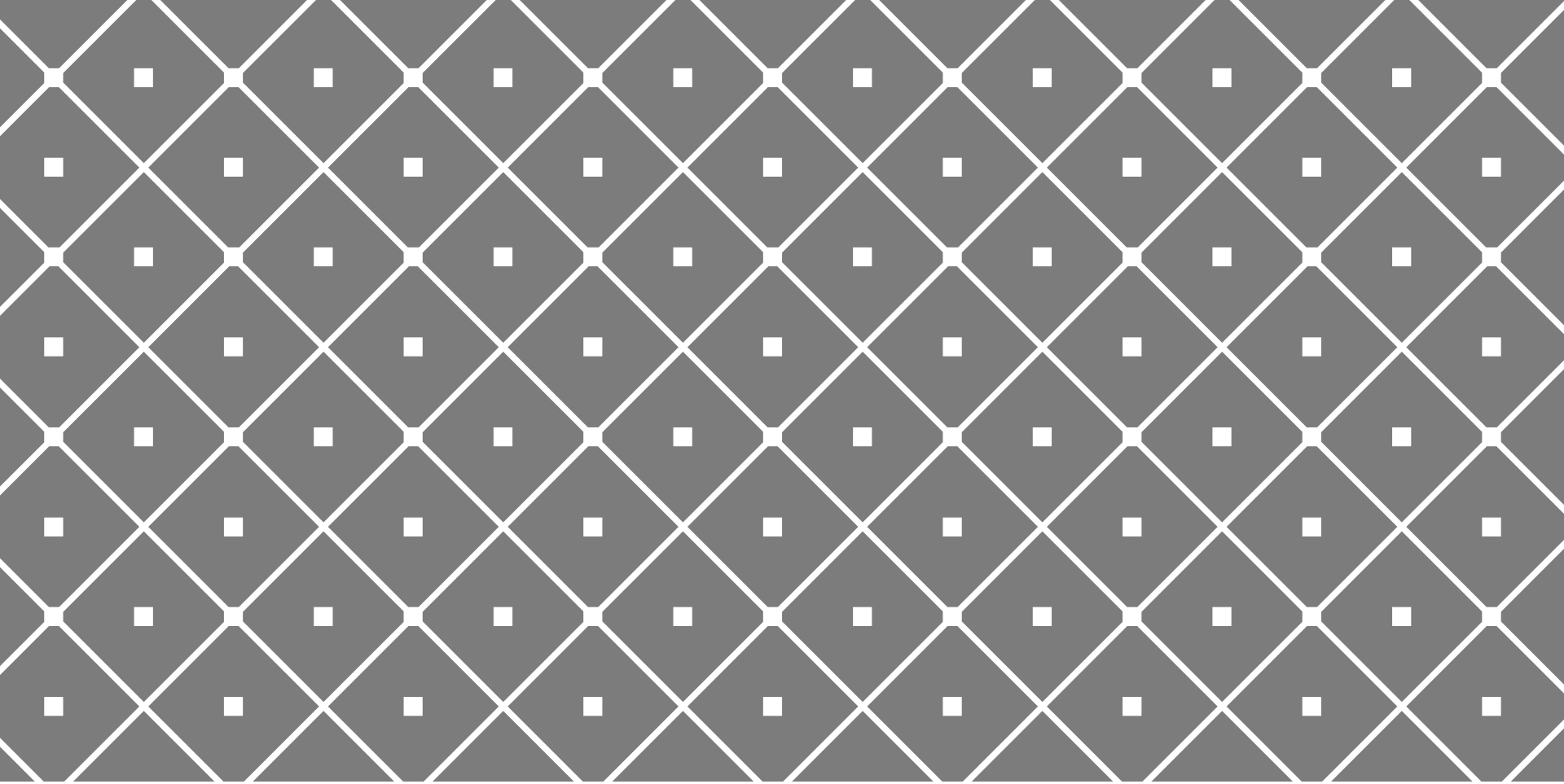
CONSTRUÇÃO DE TIPOS DERIVADOS

```
int MPI_Type_contiguous(int count,
MPI_datatype oldtype, MPI_Datatype *newtype)
```

- O tipo mais simples de dados derivado consiste em uma quantidade de itens contíguos de mesmo tipo.

EXEMPLO DE DADOS CONTÍGUOS

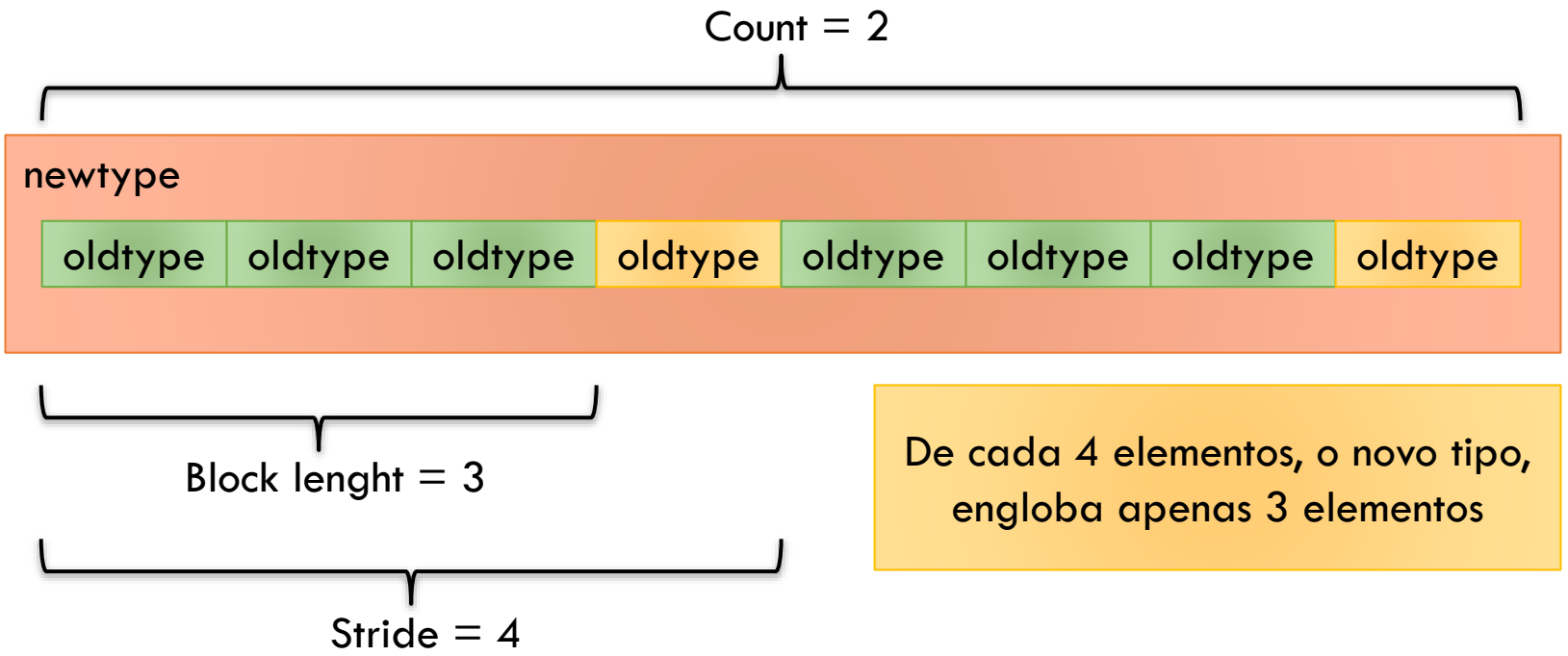
```
int main(int argc, char *argv[]) { /* Run with four processes */
  int rank;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  struct { int x; int y; int z; } point;
  MPI_Datatype pixtype;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Type_contiguous(3,MPI_INT,&pixtype);
  MPI_Type_commit(&pixtype);
  if(rank == 3){
    point.x=15; point.y=23; point.z=6;
    MPI_Send(&point,1,pixtype,1,52,MPI_COMM_WORLD);
  } else if(rank == 1) {
    MPI_Recv(&point,1,pixtype,3,52,MPI_COMM_WORLD,&status);
    printf("P:%d recv colors (%d,%d,%d) \n",rank,point.x,point.y,point.z);
  }
  MPI_Finalize();
}
```



TYPE VECTOR

CONSTRUÇÃO DE TIPOS DERIVADOS

```
MPI_Type_vector(int count, int blocklength, int stride,
    MPI_Datatype oldtype, MPI_Datatype *newtype)
```



De cada 4 elementos, o novo tipo, engloba apenas 3 elementos

CONSTRUÇÃO DE TIPOS DERIVADOS

```
MPI_Type_vector(int count, int blocklength, int stride,
    MPI_Datatype oldtype, MPI_Datatype *newtype)
```

MPI_Type_vector() constrói um novo tipo de dados a partir de um vetor de dados.

count é o número de blocos de dados do novo tipo derivado.

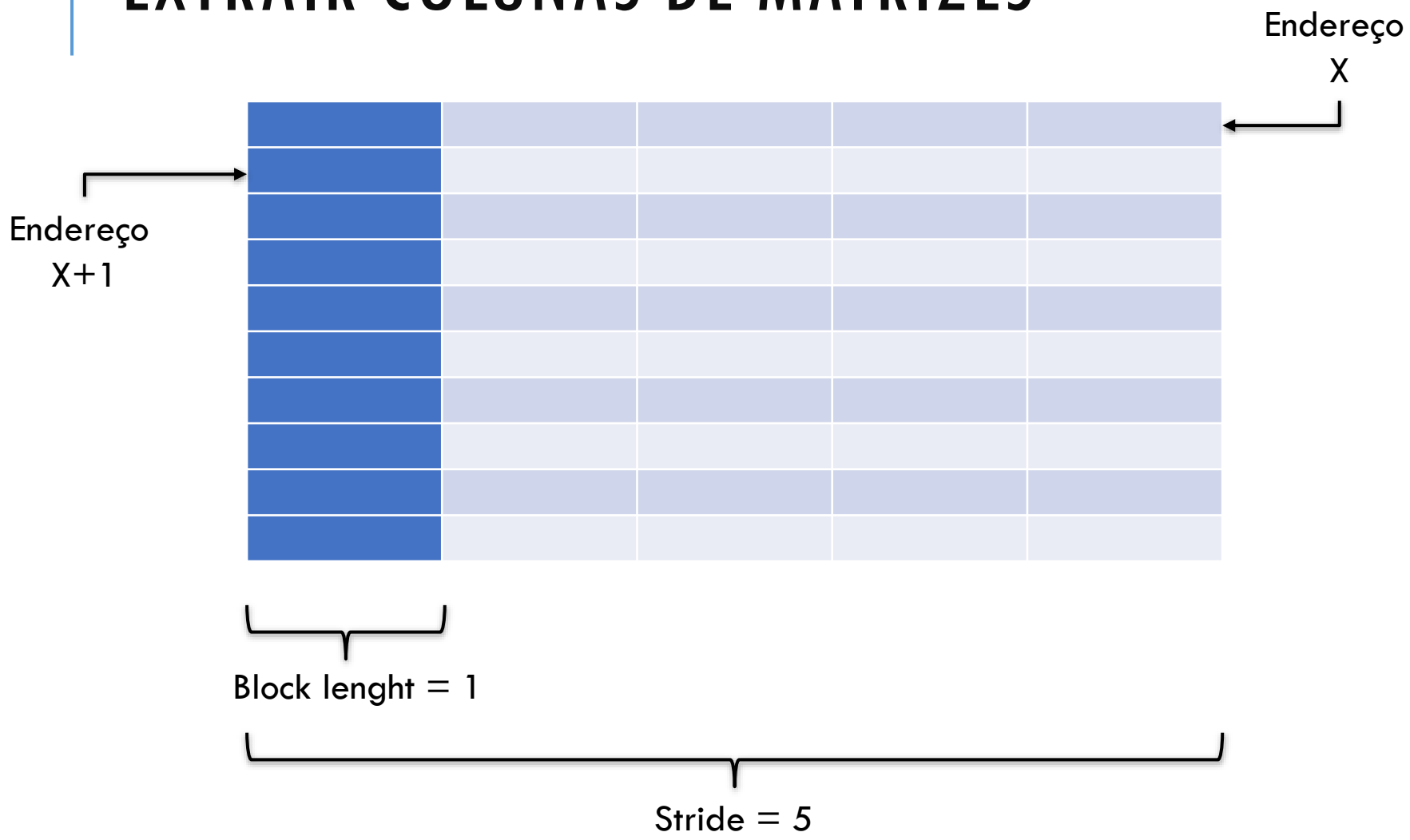
blocklength é o número de elementos contíguos de cada bloco.

stride é o número de elementos contíguos que separa o início de cada bloco (deslocamento / espalhamento).

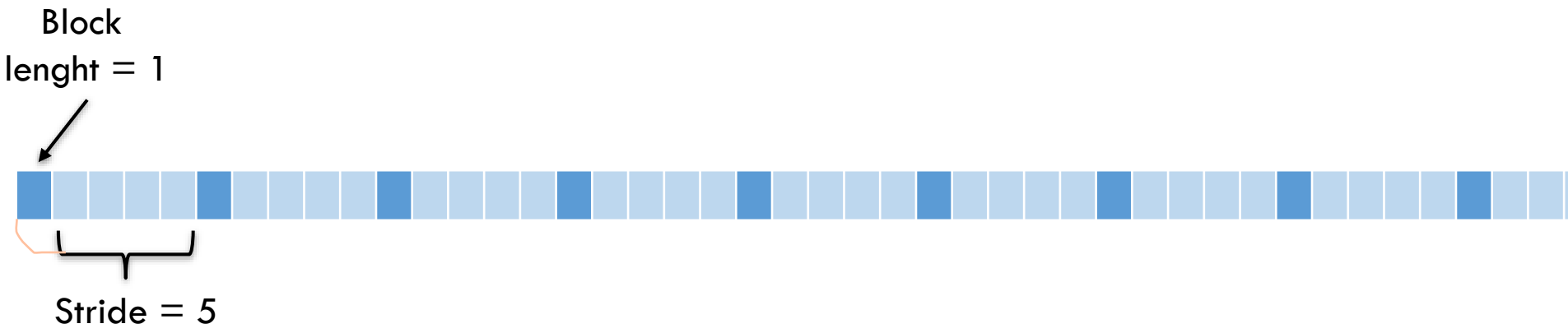
oldtype é o tipo de dados dos elementos do vetor.

newtype é o identificador do novo tipo derivado.

EXTRAIR COLUNAS DE MATRIZES



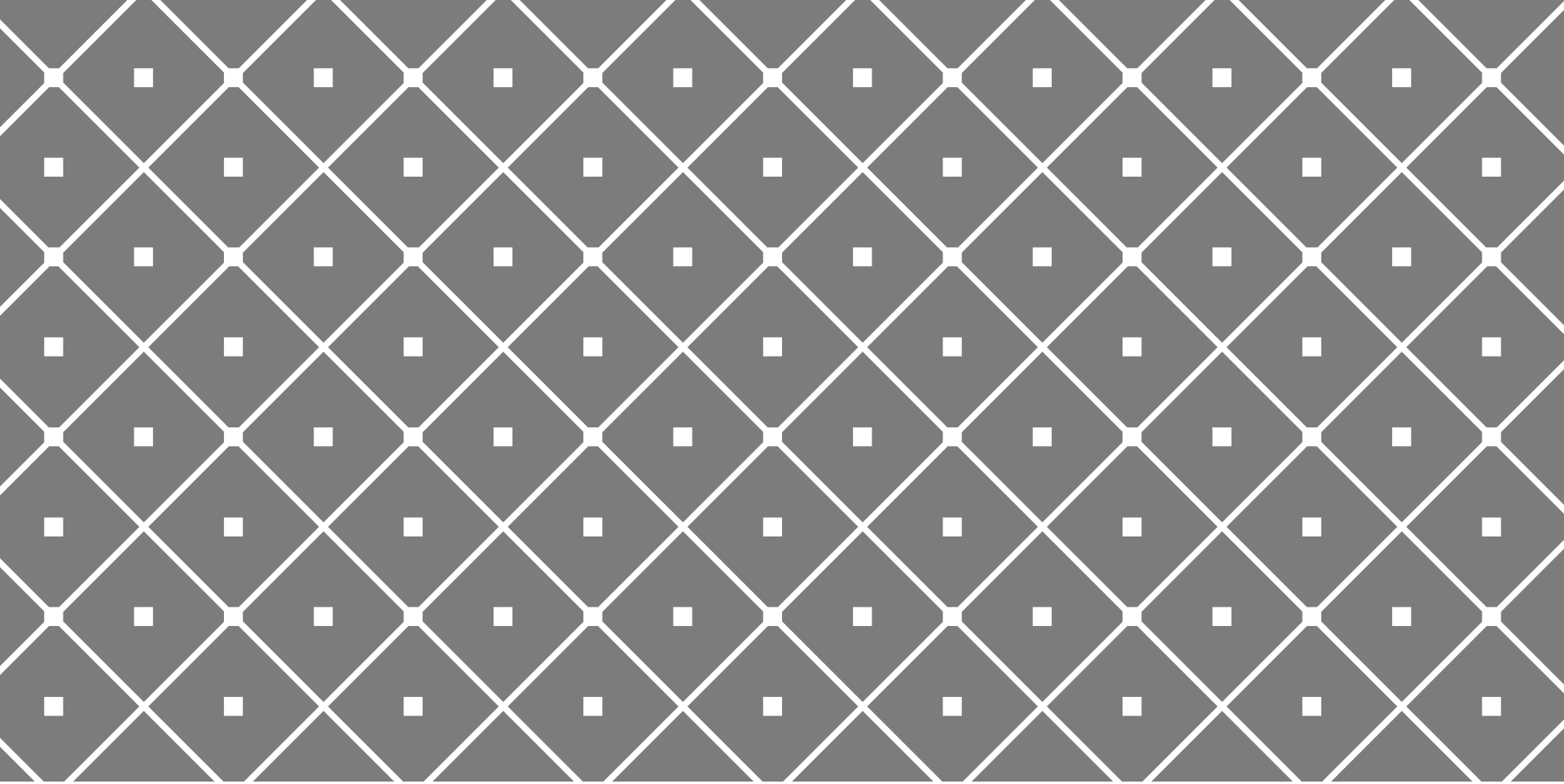
EXTRAIR COLUNAS DE MATRIZES



Representação da matriz alocada na memória

EXTRAIR COLUNAS DE MATRIZES

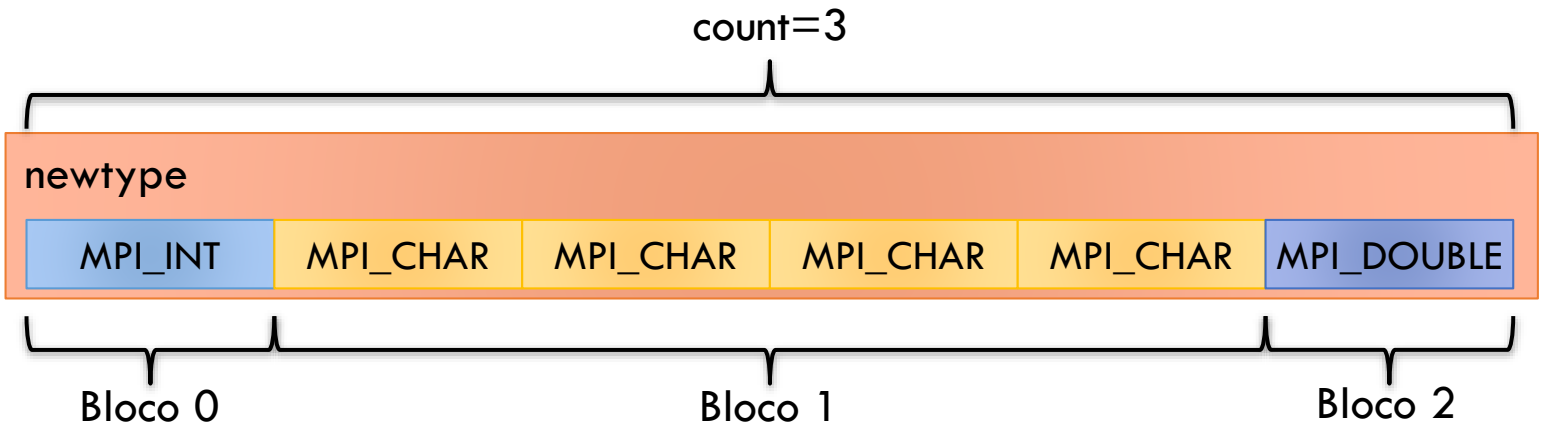
```
int my_matrix[ROWS][COLS];
int my_vector[ROWS];
MPI_Datatype col_matrix;
...
// construir um tipo derivado com ROWS blocos
// de 1 elemento separados por COLS elementos
MPI_Type_vector(ROWS, 1, COLS, MPI_INT, &col_matrix);
MPI_Type_commit(&col_matrix);
...
// enviar a coluna 1 de my_matrix (agrupando o espalhamento)
MPI_Send(&my_matrix[0][1], 1, col_matrix, dest, tag, comm);
...
// receber uma dada coluna na coluna 3 de my_matrix (recebe com espalhamento)
MPI_Recv(&my_matrix[0][3], 1, col_matrix, source, tag, comm, &status);
...
// receber uma dada coluna em my_vector (recebe contíguo)
MPI_Recv(&my_vector[0], ROWS, MPI_INT, source, tag, comm, &status);
...
// libertar o tipo derivado
MPI_Type_free(&col_matrix);
```



TYPE STRUCT

CONSTRUÇÃO DE TIPOS DERIVADOS/MISTOS

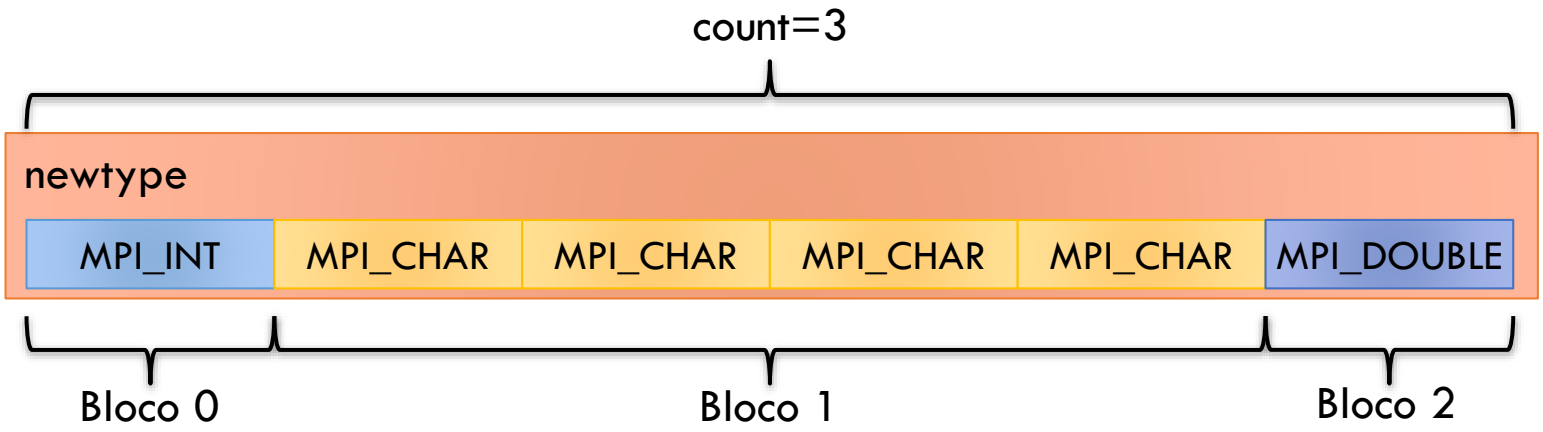
```
MPI_Type_struct(int count, int lengths[], MPI_Aint offsets[],
      MPI_Datatype oldtypes[], MPI_Datatype *newtype)
```



Usaremos 3 estruturas auxiliares...

CONSTRUÇÃO DE TIPOS DERIVADOS/MISTOS

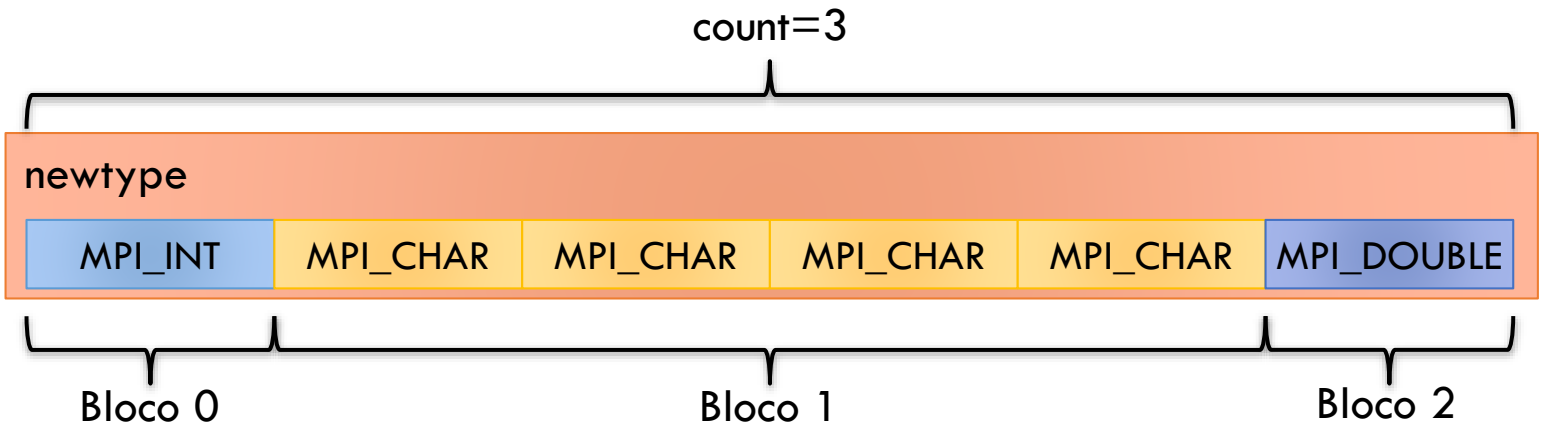
```
MPI_Type_struct(int count, int lengths[], MPI_Aint offsets[],
      MPI_Datatype oldtypes[], MPI_Datatype *newtype)
```




```
lengths[3] = {1, 4, 1}
```

CONSTRUÇÃO DE TIPOS DERIVADOS/MISTOS

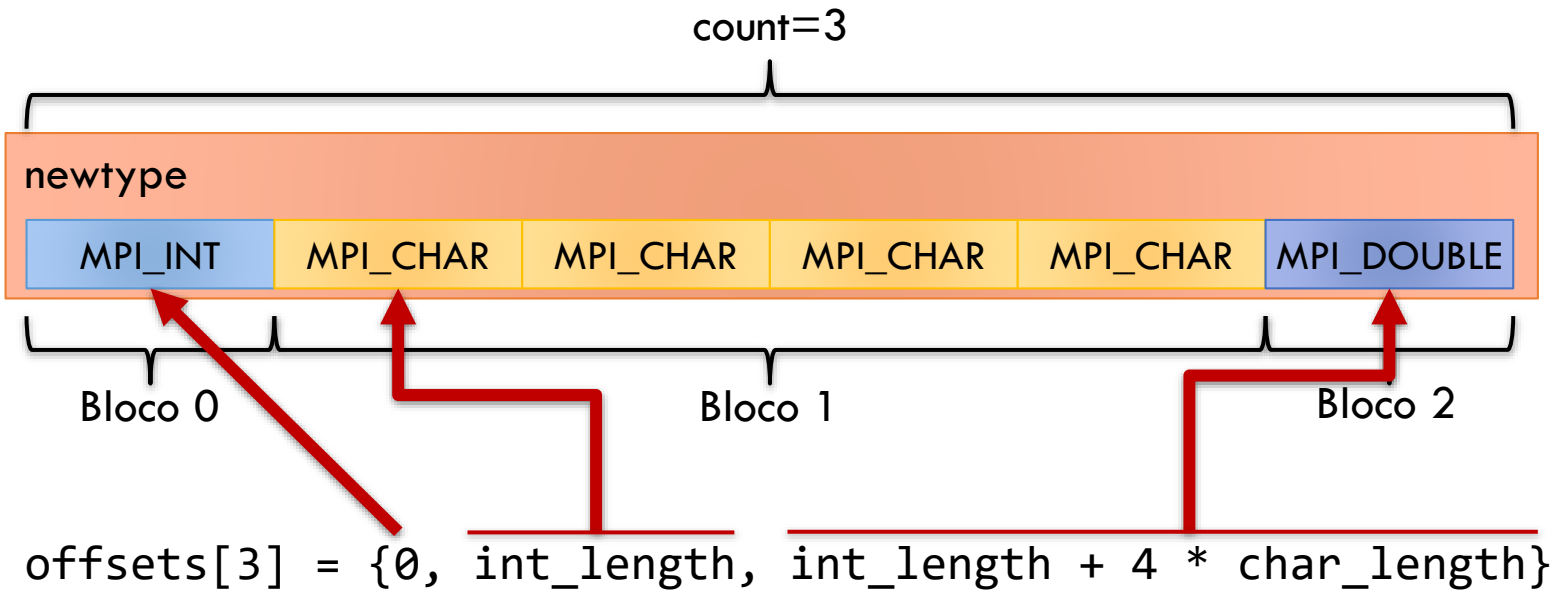
```
MPI_Type_struct(int count, int lengths[], MPI_Aint offsets[],
      MPI_Datatype oldtypes[], MPI_Datatype *newtype)
```




```
oldtypes[3] = {MPI_INT, MPI_CHAR, MPI_DOUBLE}
```

CONSTRUÇÃO DE TIPOS DERIVADOS/MISTOS

```
MPI_Type_struct(int count, int lengths[], MPI_Aint offsets[],
        MPI_Datatype oldtypes[], MPI_Datatype *newtype)
```



CONSTRUÇÃO DE TIPOS DERIVADOS

```
MPI_Type_struct(int count, int lengths[], MPI_Aint offsets[],
      MPI_Datatype oldtypes[], MPI_Datatype *newtype)
```

MPI Type struct() constrói um novo tipo de dados a partir de uma estrutura de dados.

count é o número de blocos de dados do novo tipo derivado. Representa igualmente o número de entradas nos vetores **lengths**[], **offsets**[] e **oldtypes**[].

lengths[] é o número de elementos contíguos de cada bloco.

offsets[] é o deslocamento em bytes de cada bloco dentro da estrutura.

oldtypes[] é o tipo de dados dos elementos de cada bloco.

newtype é o identificador do novo tipo derivado.

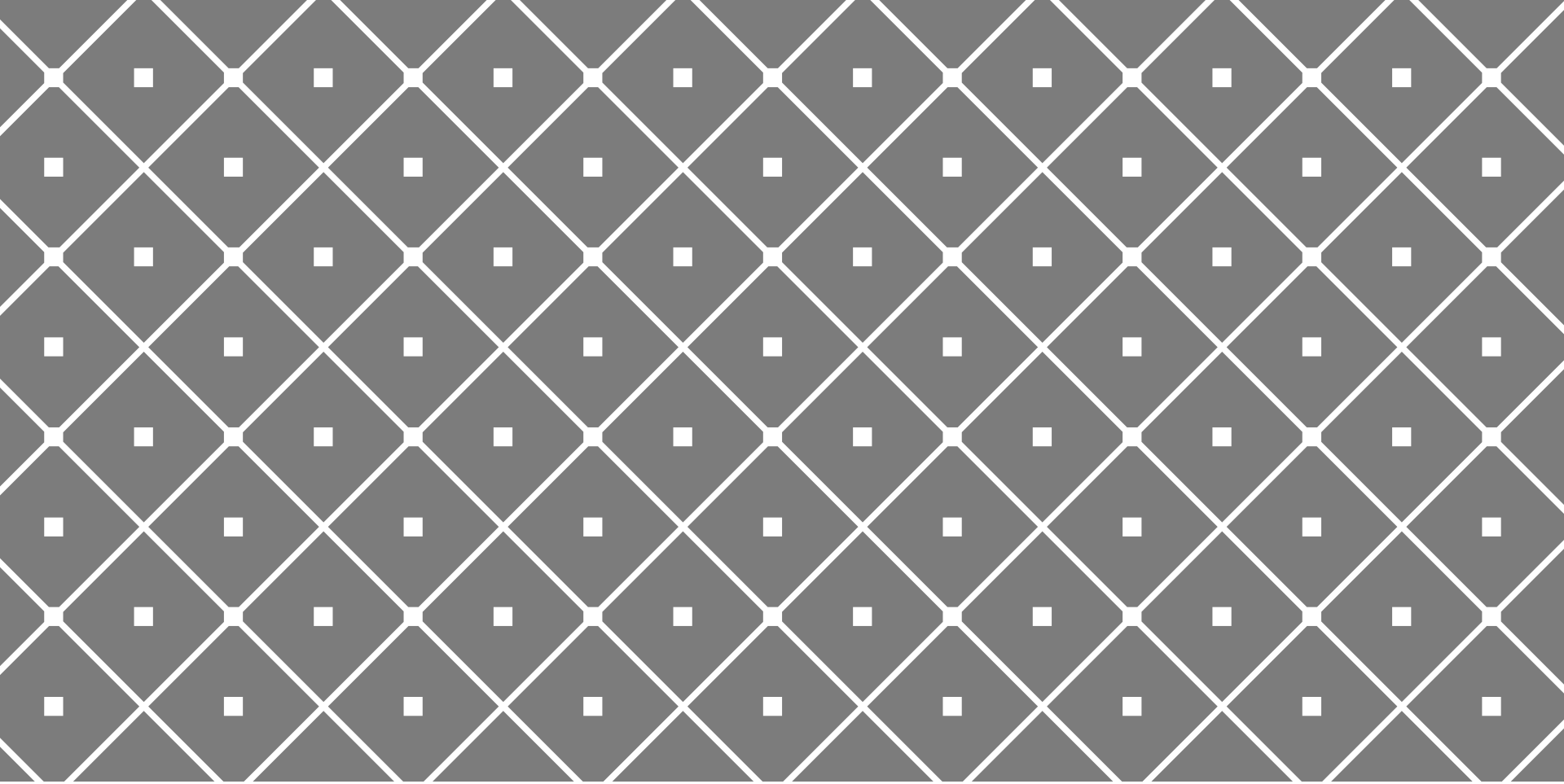
CONSTRUÇÃO DE TIPOS DERIVADOS

```
typedef struct {  // Estruct que queremos representar
  int var;
  char string[STRING_LENGTH];
  double foo;
} bar;

// 1. Vamos indicar que existem 3 blocos, seus tamanhos e tipos
int count = 3;
int lengths[3] = {1, STRING_LENGTH, 1};
MPI_Dataype oldtypes[3] = {MPI_INT, MPI_CHAR, MPI_DOUBLE};

// 2. Os offsets indicam em que byte cada elemento inicia
MPI_Aint offsets[3] = {0, sizeof(int), sizeof(int) + STRING_LENGTH};

// 3. Declarar o novo tipo, a estrutura e informar os processos
MPI_Datatype barDatatype;
MPI_Type_struct(count, lengths, offsets, types, &barDatatype);
MPI_Type_commit(&barDatatype); //Agora usamos com send/recv
```



FUNÇÕES AUXILIARIES PARA NOVOS TIPOS DE DADOS

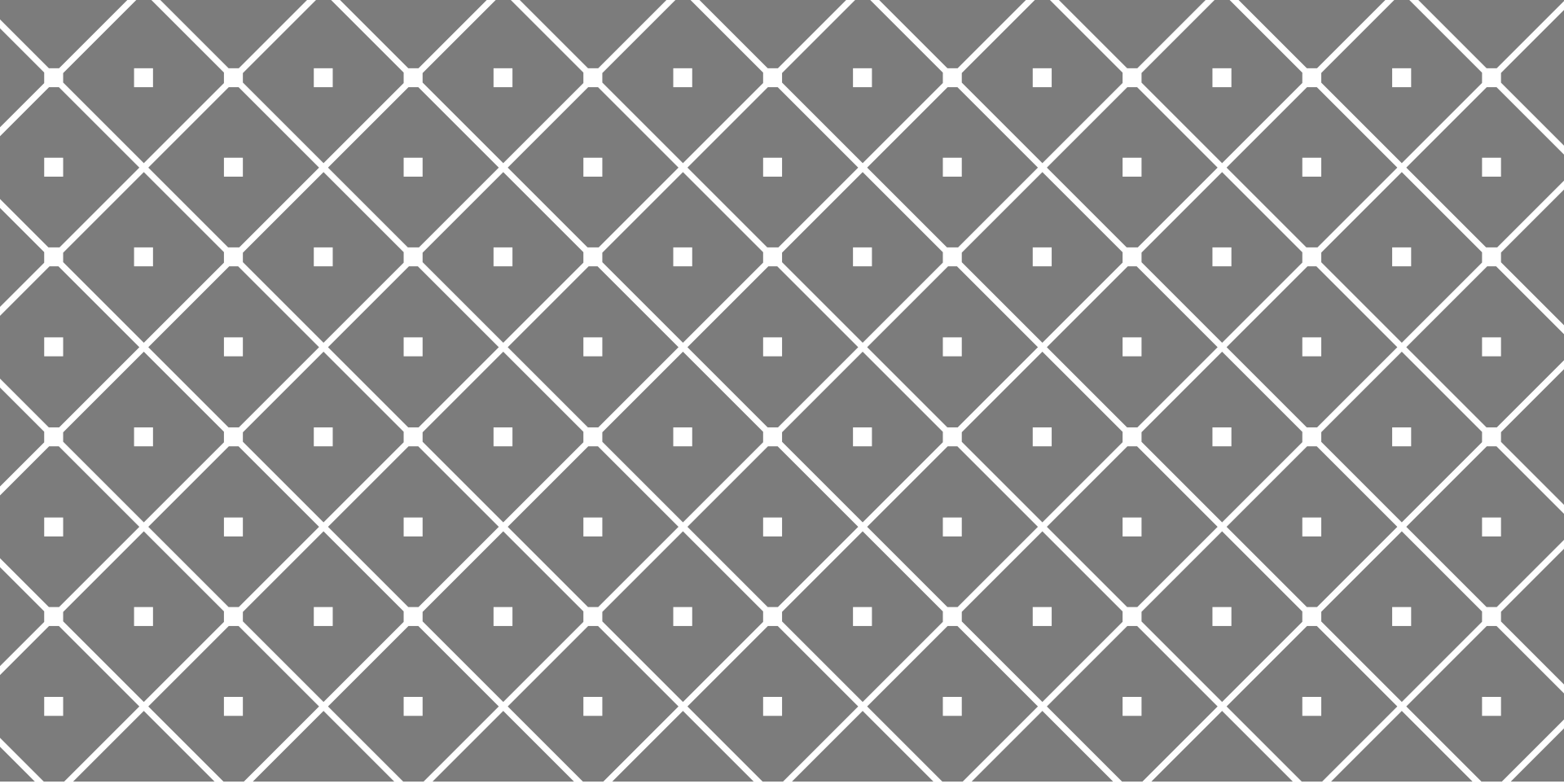
CERTIFICAR E LIBERAR UM TIPO DERIVADO

```
MPI_Type_commit(MPI_Datatype *datatype)
```

MPI_Type_commit() certifica perante o ambiente de execução do MPI a existência de um novo tipo derivado identificado por **datatype.**

```
MPI_Type_free(MPI_Datatype *datatype)
```

MPI_Type_free() libera do ambiente de execução o tipo derivado identificado por **datatype.**



PACOTES DE DADOS

EMPACOTAR DADOS

```
MPI_Pack(void *buf, int count, MPI_Datatype datatype,
void *packbuf, int packsize, int *position, MPI_Comm comm)
```

MPI_Pack() empacota dados não contíguos em posições contiguas de memória.

buf é o endereço inicial dos dados a empacotar.

count é o número de elementos do tipo **datatype** a empacotar.

datatype é o tipo de dados a empacotar.

packbuf é o endereço do buffer onde devem ser colocados os dados a empacotar.

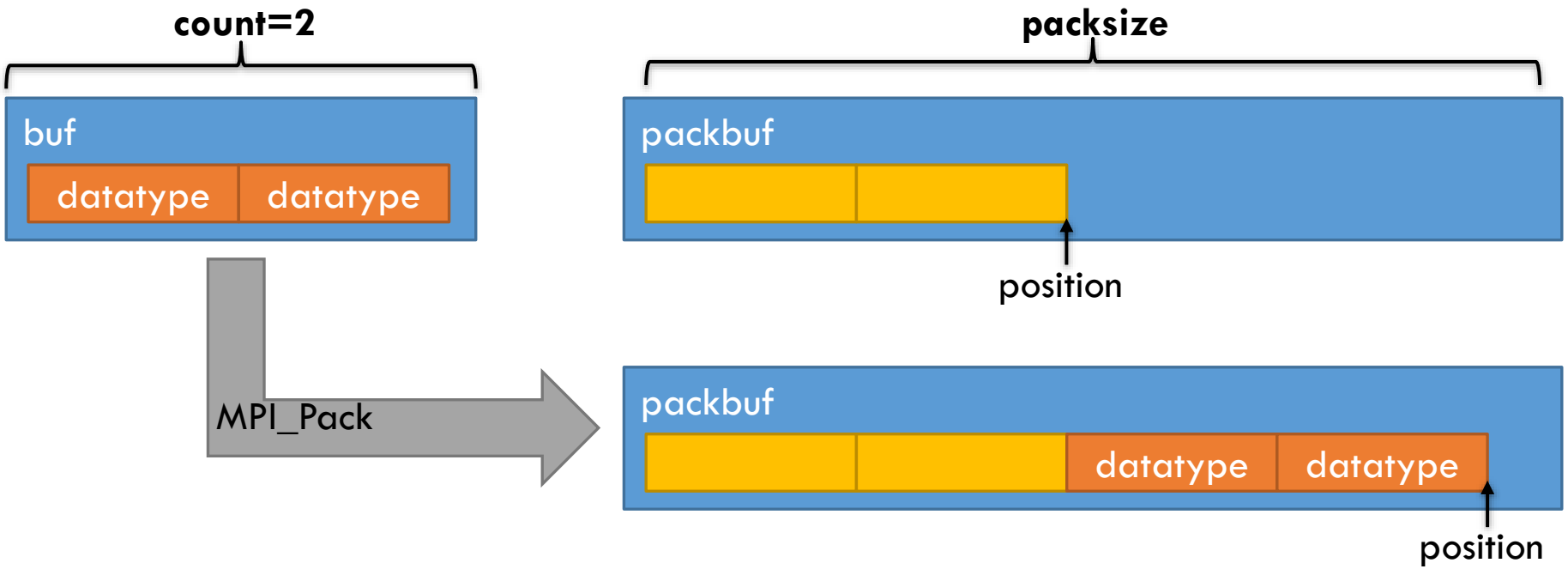
packsize é o tamanho em bytes do buffer de empacotamento.

position é a posição (em bytes) do buffer a partir da qual os dados devem ser empacotados.

comm é o comunicador dos processos envolvidos na comunicação.

EMPACOTAR DADOS

```
MPI_Pack(void *buf, int count, MPI_Datatype datatype,
void *packbuf, int packsize, int *position, MPI_Comm comm)
```



DESEMPACOTAR DADOS

```
MPI_Unpack(void *packbuf, int packsize, int *position,
void *buf, int count, MPI_Datatype datatype, MPI_Comm comm)
```

MPI_Unpack() desempacota dados contíguos em posições não contíguas de memória.

packbuf é o endereço do buffer onde estão os dados a desempacotar.

packsize é o tamanho em bytes do buffer de empacotamento.

position é a posição (em bytes) do buffer a partir da qual estão os dados a desempacotar.

buf é o endereço inicial para onde os dados devem ser desempacotados.

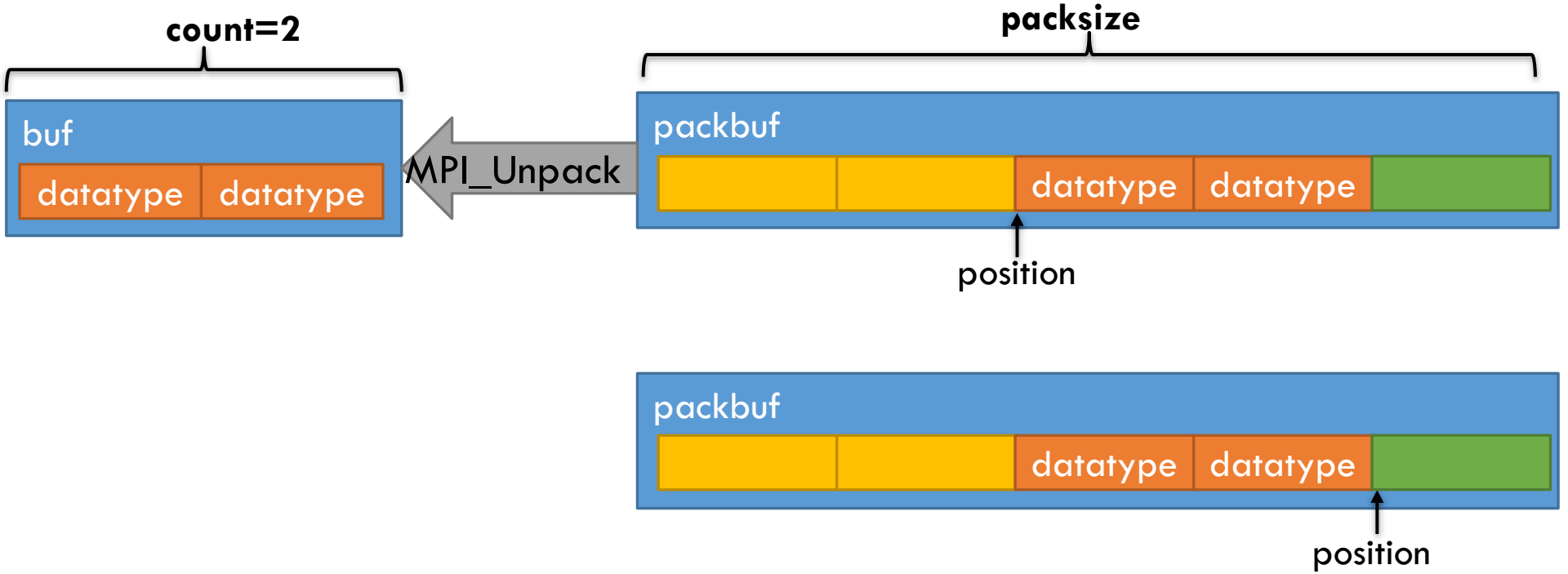
count é o número de elementos do tipo **datatype** a desempacotar.

datatype é o tipo de dados a desempacotar.

comm é o comunicador dos processos envolvidos na comunicação.

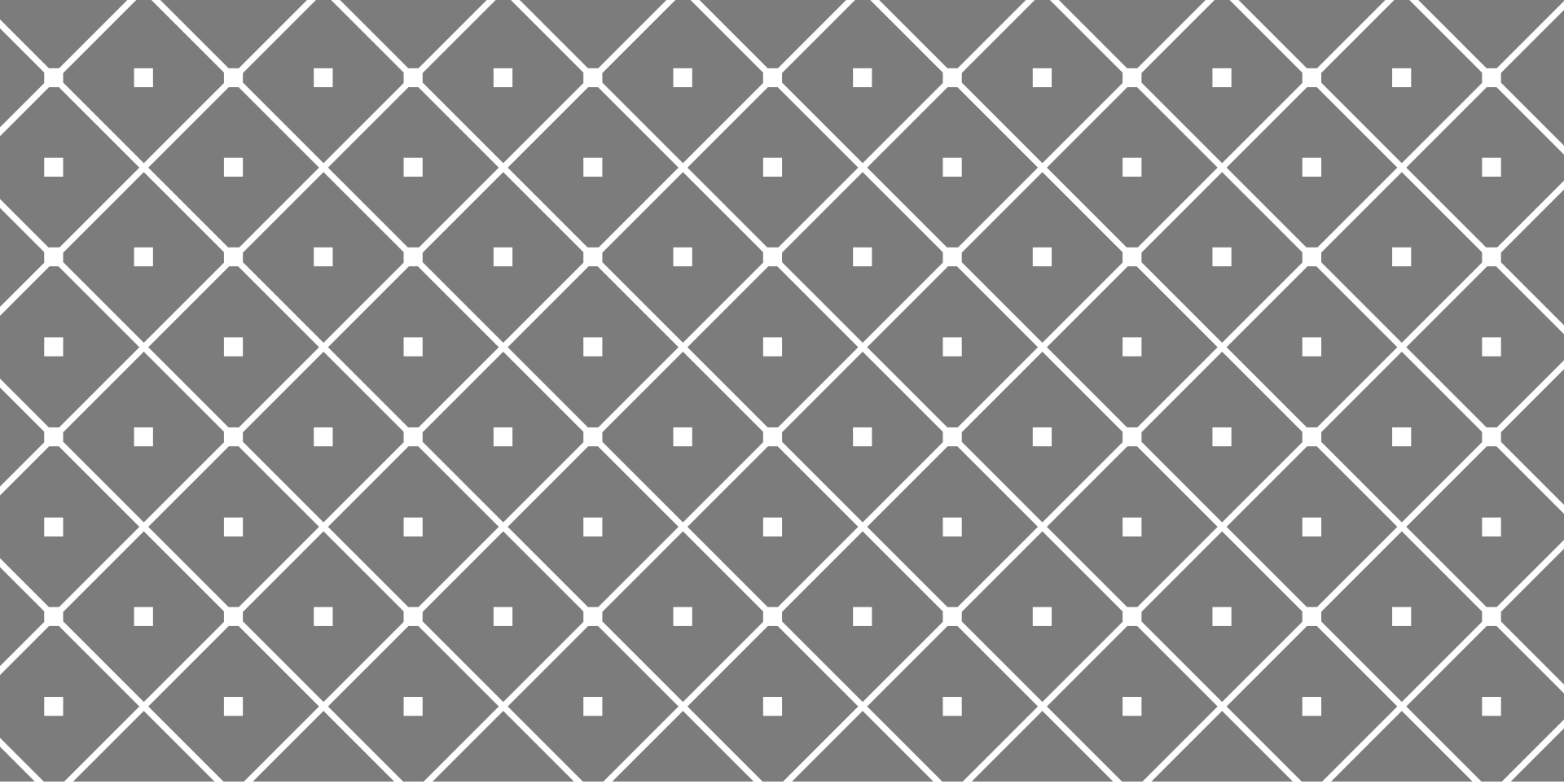
DESEMPACOTAR DADOS

```
MPI_Unpack(void *packbuf, int packsize, int *position,
void *buf, int count, MPI_Datatype datatype, MPI_Comm comm)
```



MATRIZ DE TAMANHO VARIÁVEL

```
// inicialmente ROWS e COLS não são do conhecimento do processo 1
int *my_matrix, ROWS, COLS, pos;
char pack_buf[BUF_SIZE];
...
if (my_rank == 0) {                  // empacotar e enviar ROWS, COLS e my_matrix
  pos = 0;
  MPI_Pack(&ROWS, 1, MPI_INT, pack_buf, BUF_SIZE, &pos, comm);
  MPI_Pack(&COLS, 1, MPI_INT, pack_buf, BUF_SIZE, &pos, comm);
  MPI_Pack(my_matrix, ROWS * COLS, MPI_INT, pack_buf, BUF_SIZE, &pos, comm);
  MPI_Send(pack_buf, pos, MPI_PACKED, 1, tag, comm);
} else if (my_rank == 1) {           // receber e desempacotar ROWS, COLS e my_matrix
  MPI_Recv(&pack_buf, BUF_SIZE, MPI_PACKED, 0, tag, comm, &status);
  pos = 0;
  MPI_Unpack(&pack_buf, BUF_SIZE, &pos, &ROWS, 1, MPI_INT, comm);
  MPI_Unpack(&pack_buf, BUF_SIZE, &pos, &COLS, 1, MPI_INT, comm);
                                     // aloca espaço para representar my_matrix
  my_matrix = (int *) malloc(ROWS * COLS * sizeof(int));
  MPI_Unpack(&pack_buf, BUF_SIZE, &pos, my_matrix, ROWS * COLS, MPI_INT, comm);
}
...
```



QUE TIPO DE DADOS UTILIZAR?

QUE TIPO DE DADOS DEVO UTILIZAR?

Para dados homogêneos (do mesmo tipo):

Se os dados forem todos do **mesmo tipo e se encontrarem em posições contíguas de memória,** então devemos utilizar o argumento count das funções de envio e recepção de mensagens.

Se os dados forem todos do **mesmo tipo mas não se encontrarem em posições contíguas de memória,** então devemos criar um tipo derivado utilizando as funções

- MPI_Type_vector() (para dados separados por intervalos regulares)
- MPI_Type_indexed() (para dados separados por intervalos irregulares).

QUE TIPO DE DADOS DEVO UTILIZAR?

Para dados heterogêneos (tipos diferentes):

Se os dados forem **heterogêneos e possuírem um determinado padrão constante** então devemos criar um tipo derivado utilizando a função MPI_Type_struct().

Se os dados forem **homogêneos/heterogêneos mas não possuírem padrões regulares** então devemos utilizar as funções MPI_Pack()/MPI_Unpack().

As funções MPI_Pack()/MPI_Unpack() podem ser utilizadas para trocar **dados heterogêneos** (**poucas vezes**).