

PROGRAMAÇÃO PARALELA

MPI 04 - COMUNICADORES

Marco A. Zanata Alves

COMUNICADORES

O MPI permite que você converse com todos os processos dentro de um comunicador ao mesmo tempo.

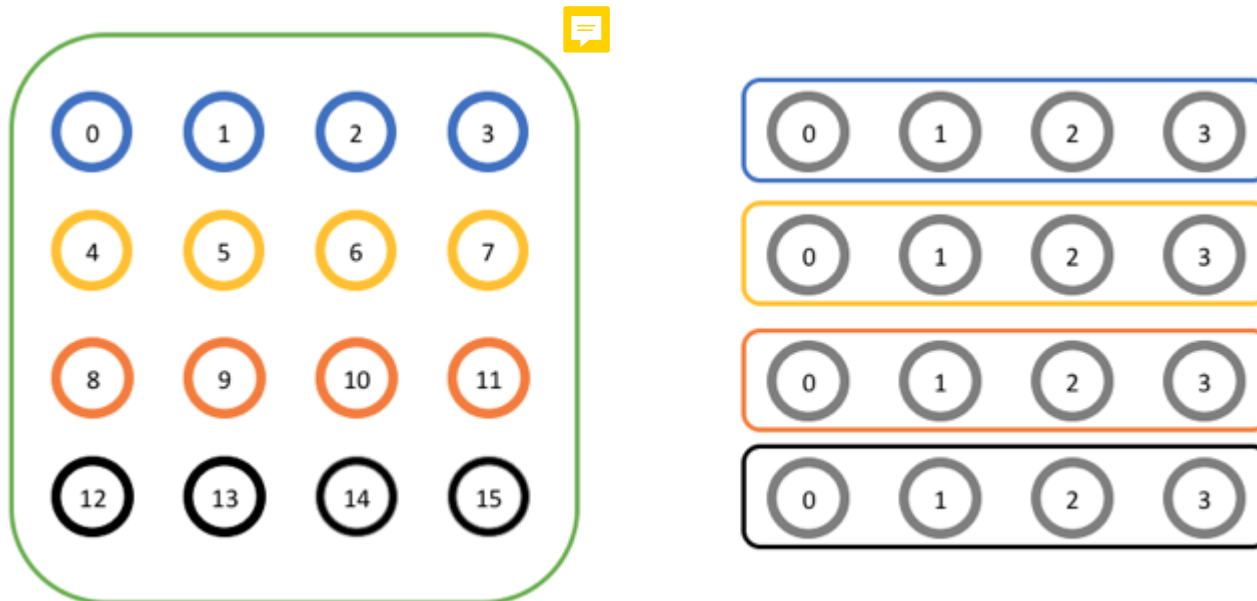
No entanto, até agora, usamos apenas o comunicador padrão, `MPI_COMM_WORLD`.

Para aplicativos simples é comum usarmos apenas `MPI_COMM_WORLD`. Mas para casos de uso mais complexos, pode ser útil ter mais comunicadores.

COMUNICADORES

Podemos assim dividir os processos em times, e cada time poderá fazer suas comunicações coletivas por exemplo.

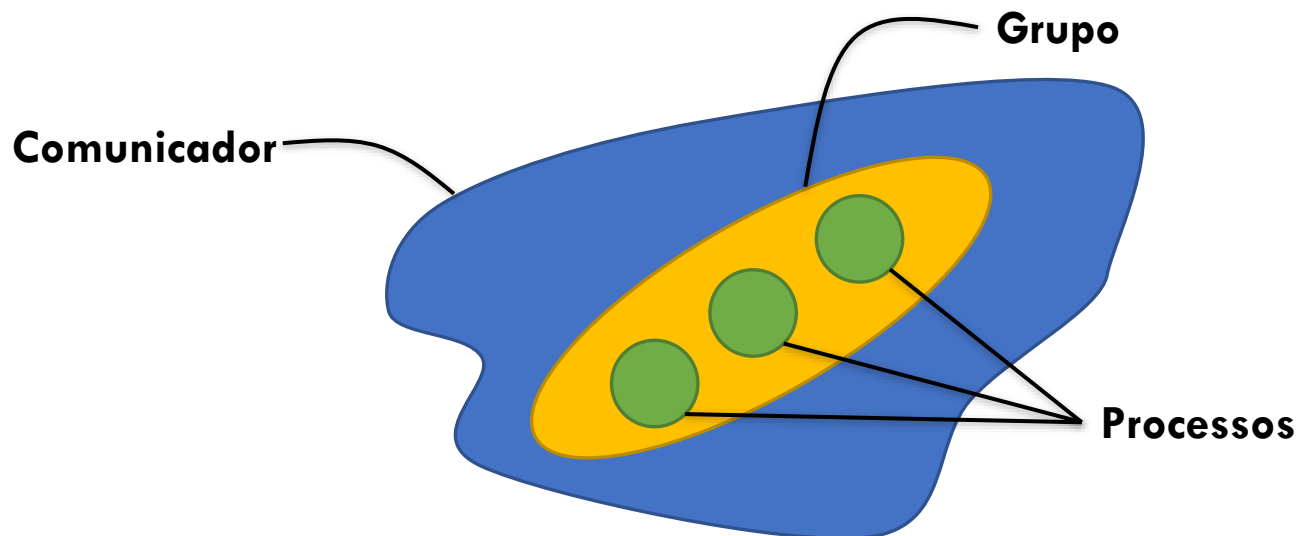
Para além do comunicador universal, o ambiente de execução do MPI permite criar novos comunicadores.



COMUNICADORES E GRUPOS

Um comunicador pode ser descrito como um grupo de processos que podem trocar mensagens entre si. Associado a um comunicador temos:

- Um grupo: conjunto ordenado de processos. Temos objeto desse tipo.
- Um contexto: estrutura de dados que o identifica de forma única. Não define um objeto. O contexto é o que organiza os processos em números independentes.



COMUNICADORES E GRUPOS

Um grupo também pode ser usado como um ID na **distribuição de tarefas** / dados entre diferentes processos.

Por exemplo, podemos testar se um processo está em um grupo para determinar se ele deve fazer algo.

TIPOS DE COMUNICADORES

O MPI distingue 2 tipos de comunicadores:

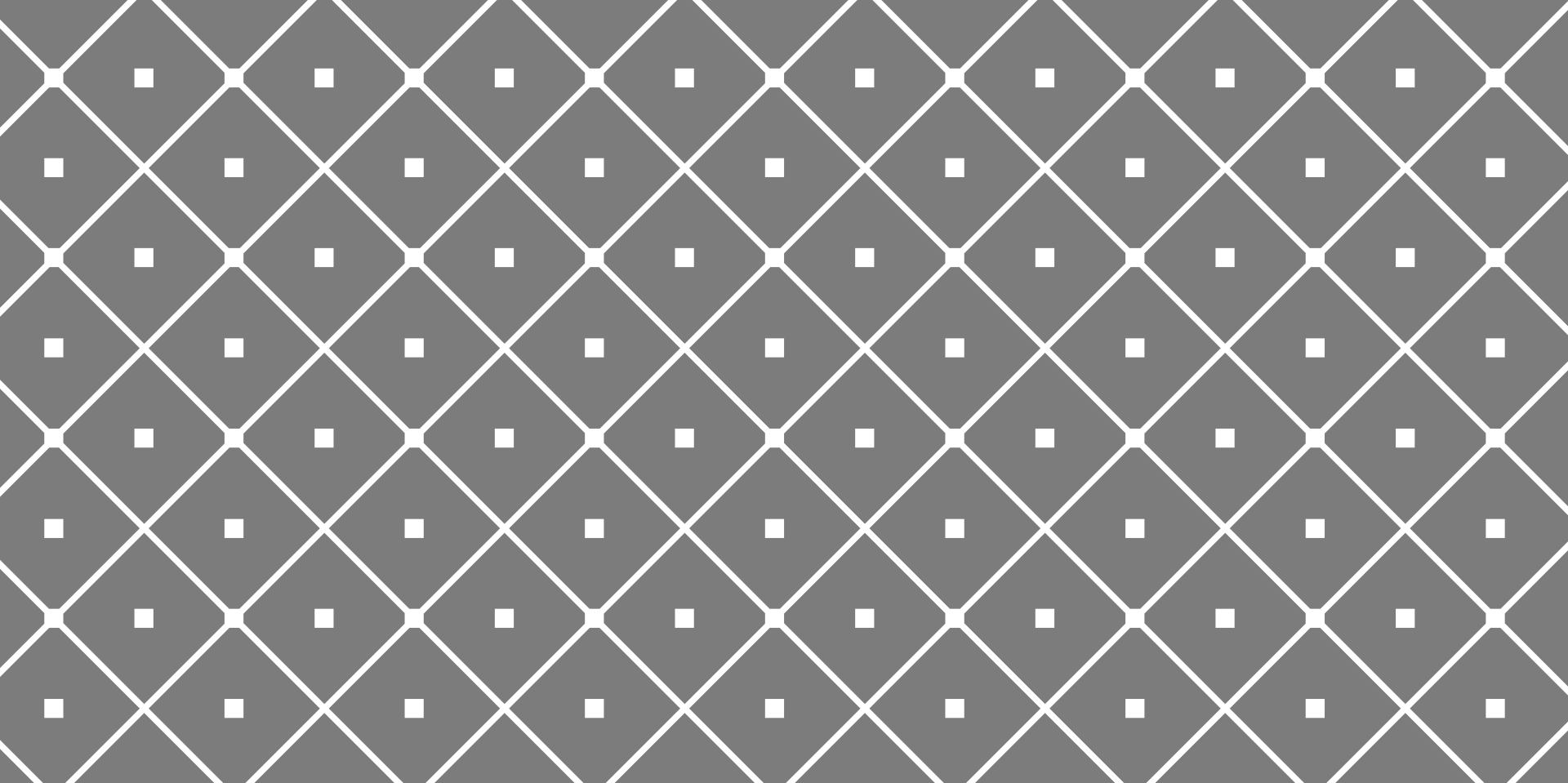
Um **intra-comunicador** envolve um único grupo de processos

- Este é o tipo padrão de comunicador.
- Mais usado por desenvolvedores de aplicativos.

Um **inter-comunicador** envolve dois grupos de processos

- Basicamente ele provê a comunicação entre dois comunicadores/grupos diferentes
- Mais usados por designers de bibliotecas.

Nesse curso focamos apenas em **intra-comunicadores**.



CRIANDO GRUPOS

CRIAR GRUPOS

```
MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
```

MPI_Comm_group() devolve em **group** o grupo de processos do comunicador **comm**.

```
MPI_Group_incl(MPI_Group old_group, int size,  
               int ranks[], MPI_Group *new_group)
```

MPI_Group_incl() cria um novo grupo **new_group** a partir de **old_group** constituído pelos **size** processos referenciados em **ranks[]**.

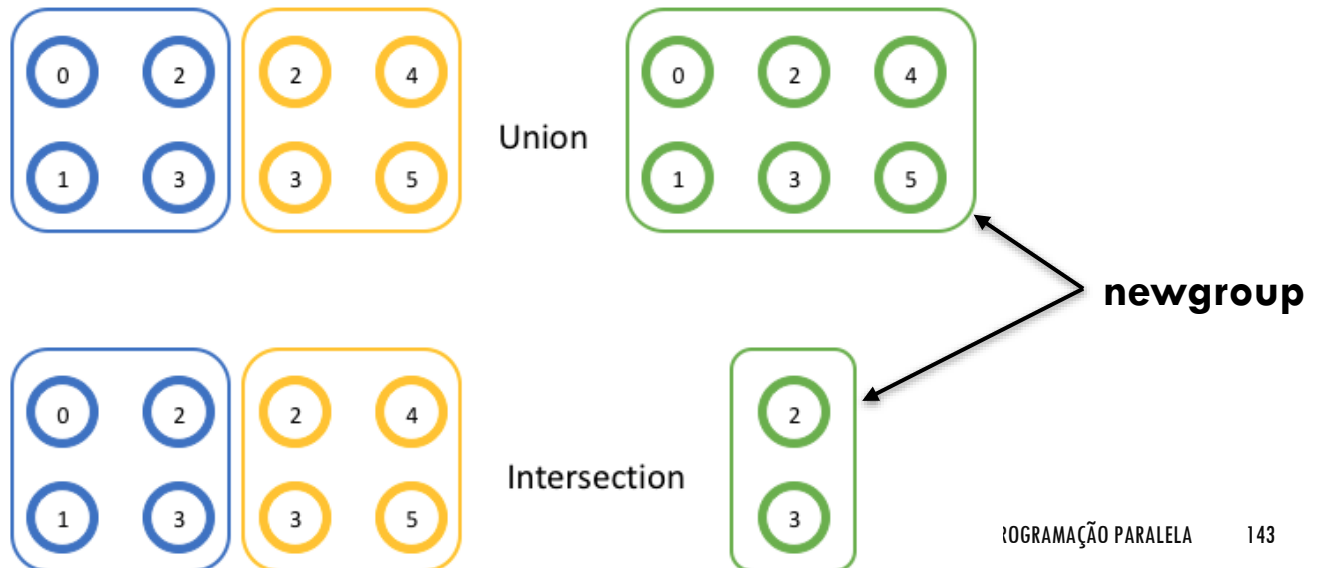
```
MPI_Group_excl(MPI_Group old_group, int size,  
               int ranks[], MPI_Group *new_group)
```

MPI_Group_excl() cria um novo grupo **new_group** a partir de **old_group** e exclui os **size** processos referenciados em **ranks[]**.

UNIÃO E INTERSECÇÃO DE GRUPOS

```
MPI_Group_union(MPI_Group group1, MPI_Group  
                group2, MPI_Group* newgroup)
```

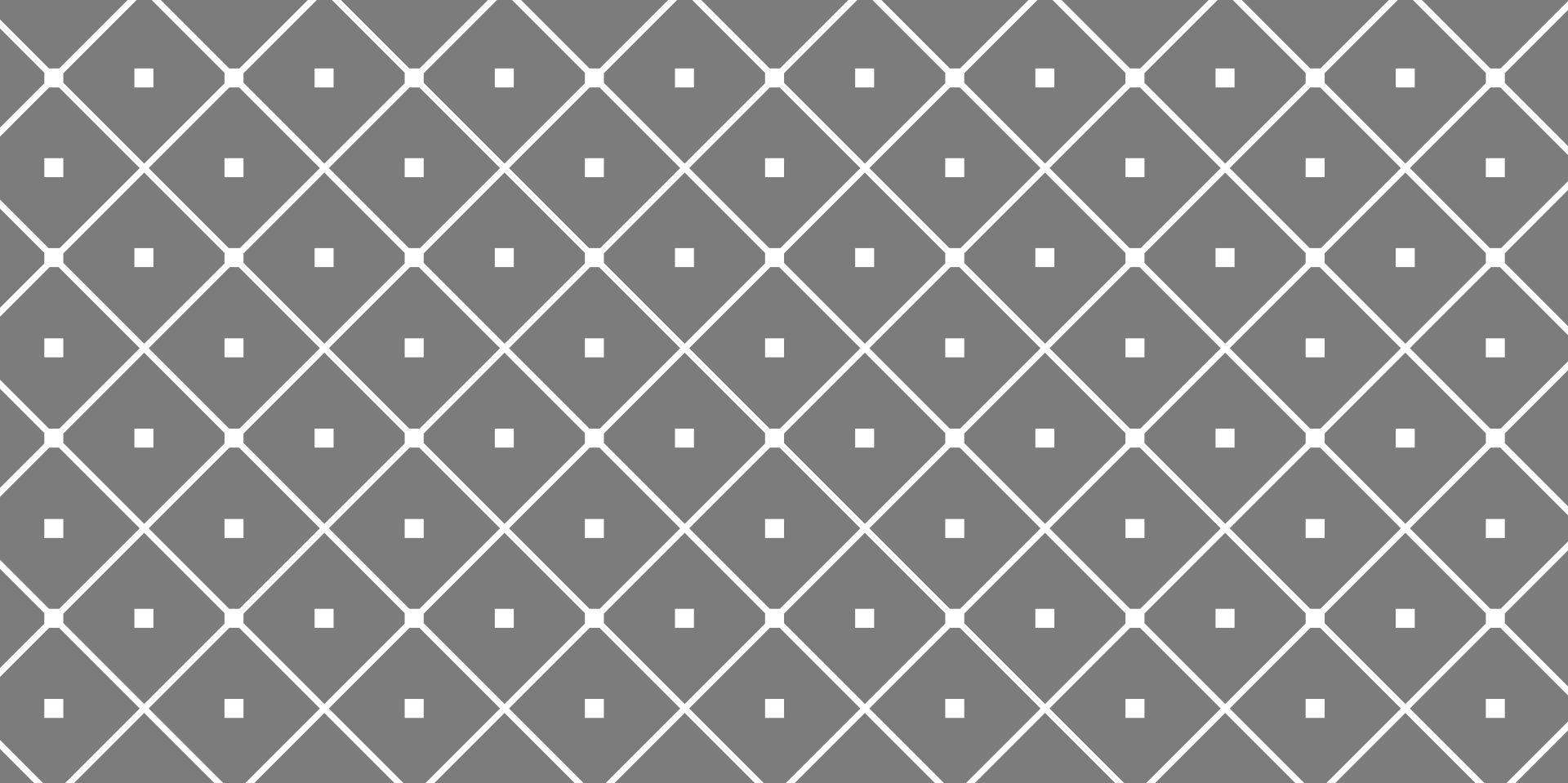
```
MPI_Group_intersection(MPI_Group group1, MPI_Group  
                      group2, MPI_Group* newgroup)
```



LIBERAR GRUPOS E COMUNICADORES

```
MPI_Group_free(MPI_Group *group)
```

MPI_Group_free() libera o grupo **group** do ambiente de execução.



CRIANDO COMUNICADORES

CRIAR COMUNICADORES

```
MPI_Comm_create(MPI_Comm old_comm,  
                 MPI_Group group, MPI_Comm *new_comm)
```

MPI_Comm_create() cria um novo comunicador **new_comm** constituído pelo grupo de processos **group** do comunicador **old_comm**.

MPI_Comm_create() é uma comunicação coletiva, pelo que deve ser chamada por todos os processos, incluindo aqueles que não aderem ao novo comunicador.

No caso de serem criados vários comunicadores, a ordem de criação deve ser a mesma em todos os processos.

DUPLICAR COMUNICADORES

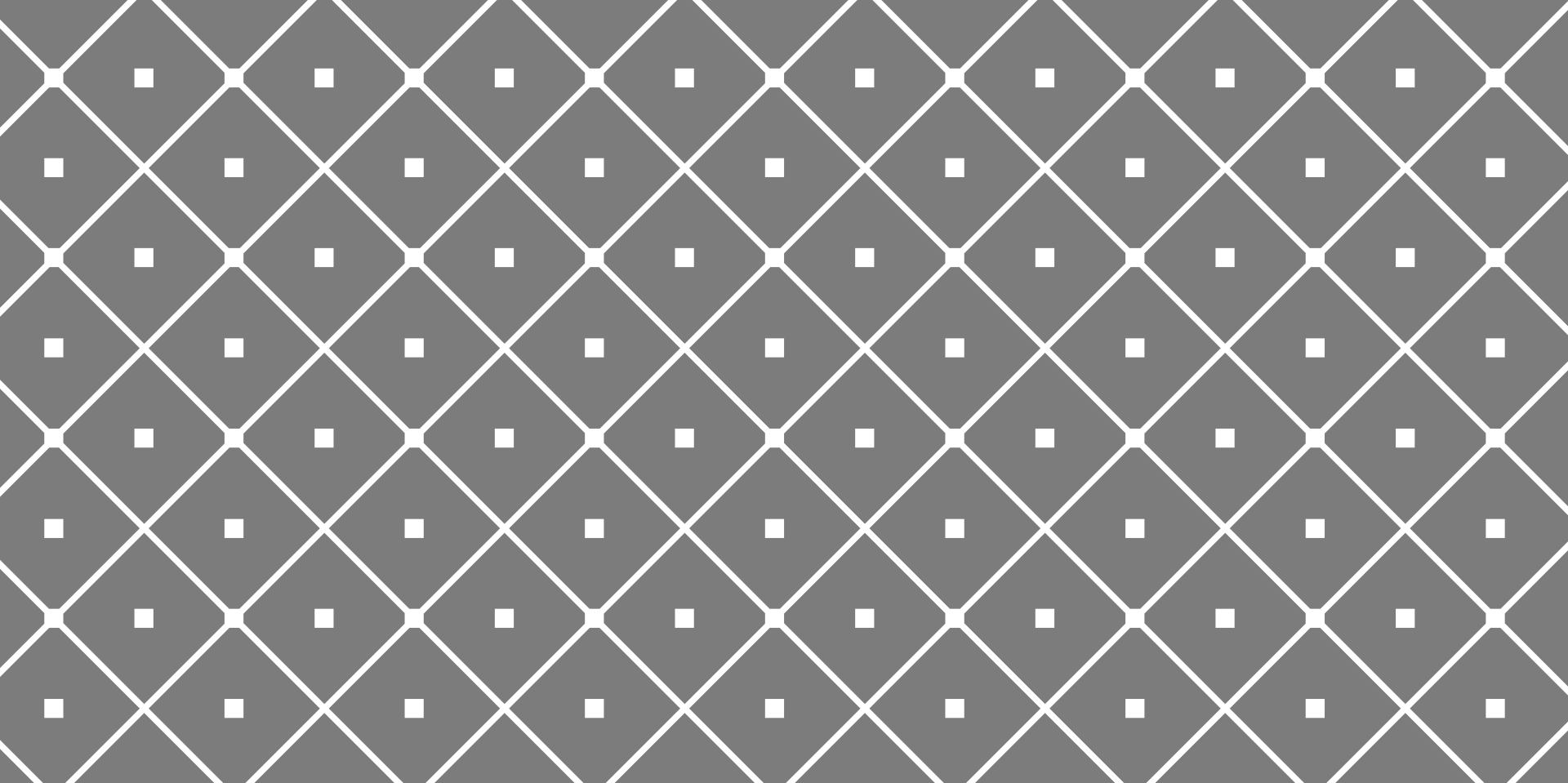
```
MPI_Comm_dup(MPI_Comm old_comm, MPI_Comm  
              *new_comm)
```

MPI_Comm_dup() cria um novo comunicador **new_comm** idêntico a **old_comm**.

LIBERAR COMUNICADORES

```
MPI_Comm_free(MPI_Comm *comm)
```

MPI_Comm_free() libera o comunicador **comm** do ambiente de execução.



EXEMPLO

PROCESSOS PARES

```
MPI_Group world_group, even_group;
MPI_Comm even_comm;

...
for (i = 0; i < n_procs; i += 2)
    ranks[i/2] = i;

MPI_Comm_group(MPI_COMM_WORLD, &world_group);
MPI_Group_incl(world_group, (n_procs + 1)/2, ranks, &even_group);
MPI_Comm_create(MPI_COMM_WORLD, even_group, &even_comm);
MPI_Group_free(&world_group);
MPI_Group_free(&even_group);
if (my_rank % 2 == 0) {
    MPI_Comm_rank(even_comm, &even_rank);
    printf("Rank: world %d even %d \n", my_rank, even_rank);
    MPI_Comm_free(&even_comm);
}

...
```


SPLIT

```
MPI_Comm_split(MPI_Comm old_comm, int split_key,  
               int rank_key, MPI_Comm *new_comm)
```

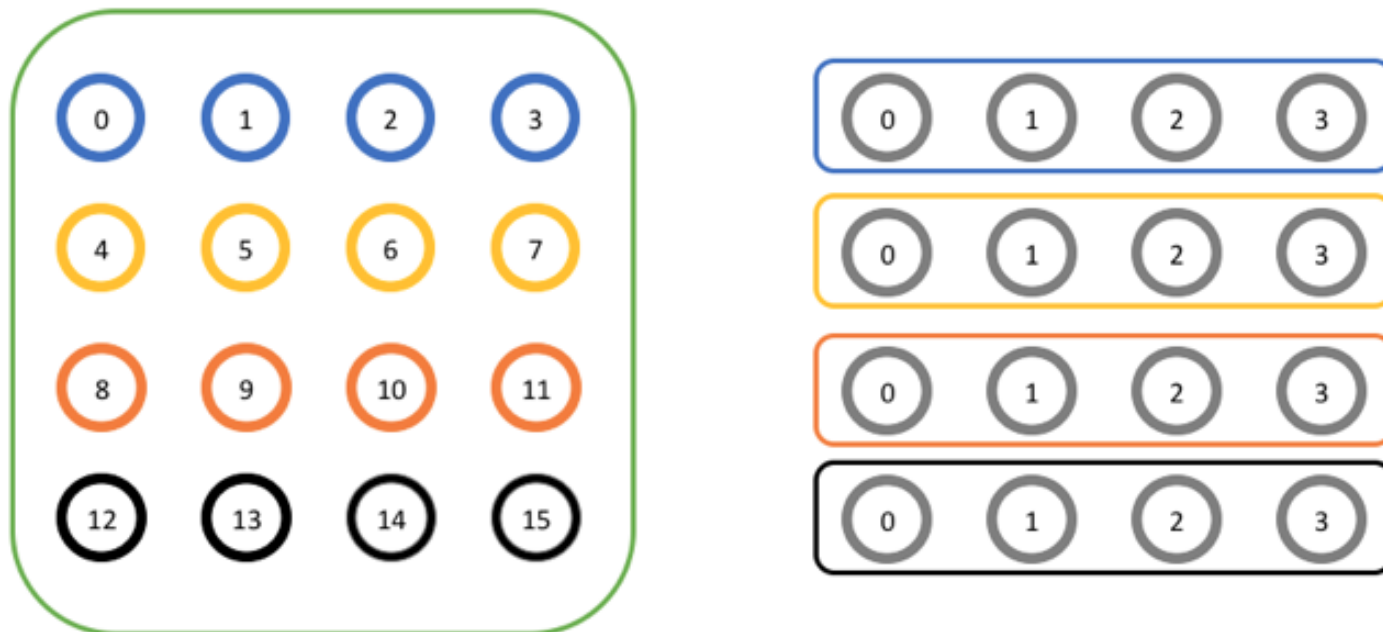
MPI_Comm_split() cria um ou mais comunicadores **new_comm** a partir de **old_comm** agrupando em cada novo comunicador os processos com idênticos valores de **split_key** e ordenando-os por **rank_key**.

Os ranks dos processos nos novos comunicadores são atribuídos por ordem crescente do argumento **rank_key**. Ou seja, o processo com o menor **rank_key** terá rank 0, o segundo menor rank 1, e assim sucessivamente.

Os processos que não pretendam aderir a nenhum novo comunicador devem indicar em **split_key** a constante **MPI_UNDEFINED**.

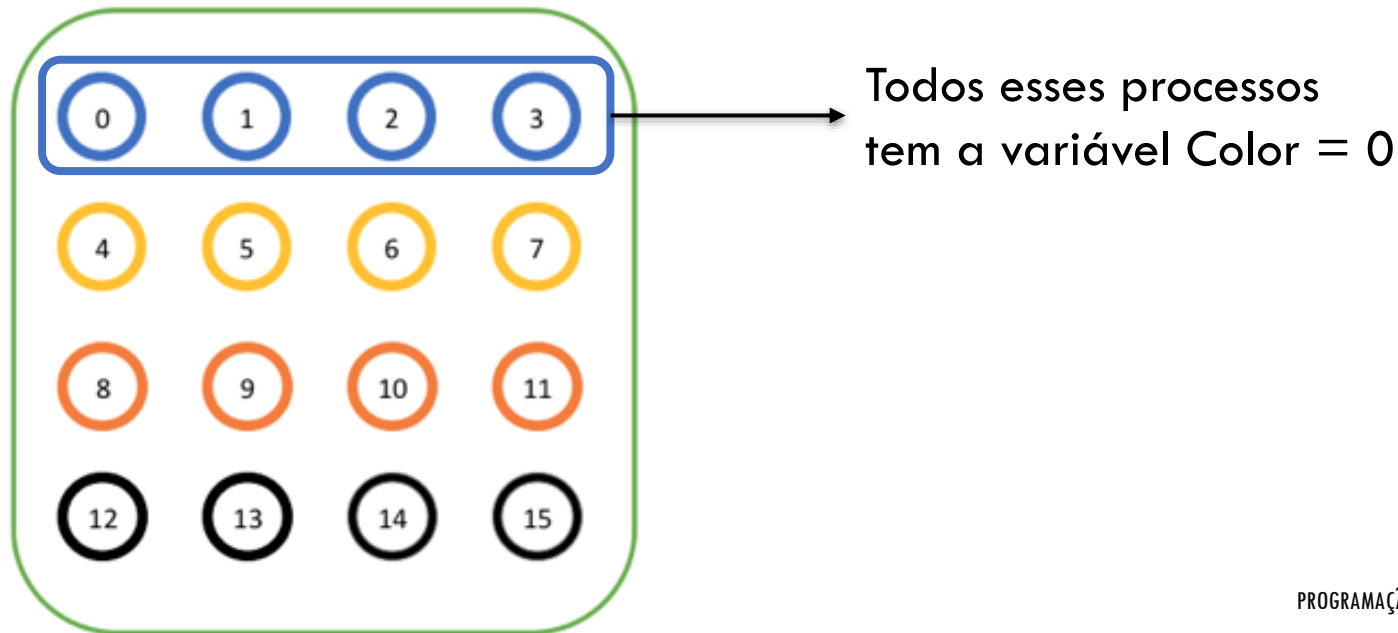
EXEMPLO COM SPLIT

Vamos supor que queremos dividir os processos globais em 4 grupos de cores, conforme apresentado na figura abaixo



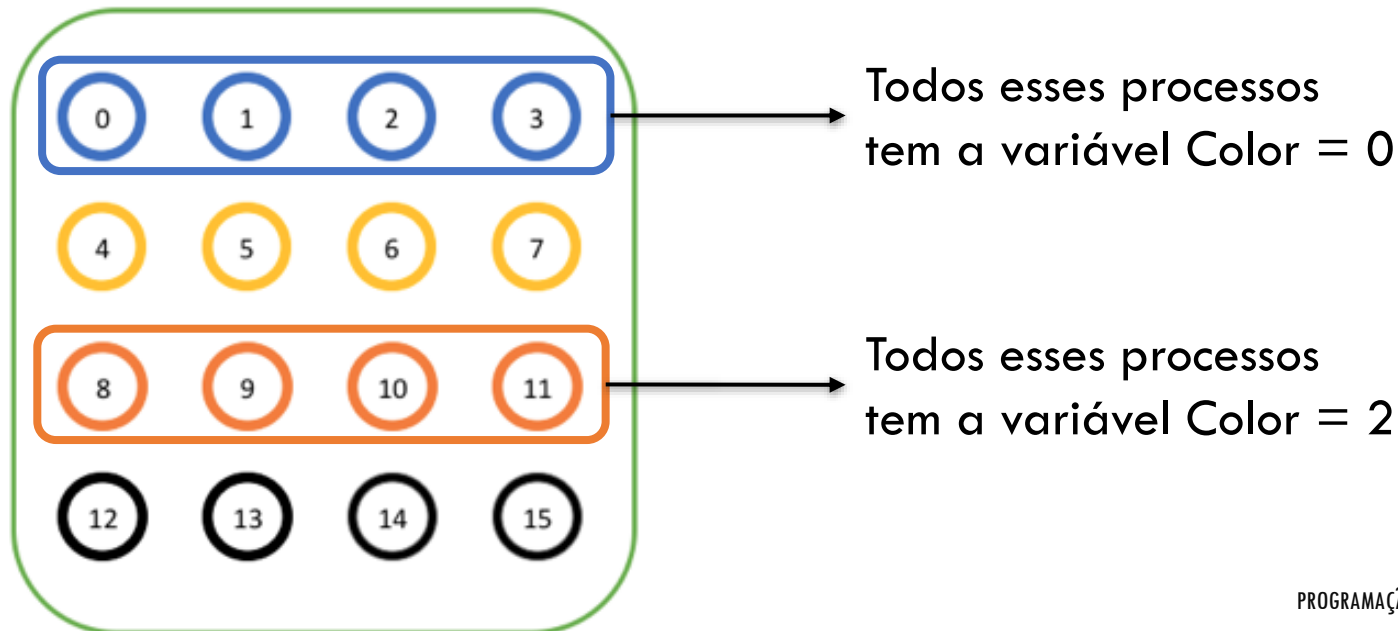
EXEMPLO COM SPLIT

```
// Obtém o rank e size do comunicador original
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor baseado na linha
int color = world_rank / 4;
// Divide o comunicador baseado na cor e
// usa o rank original para ordenamento
```



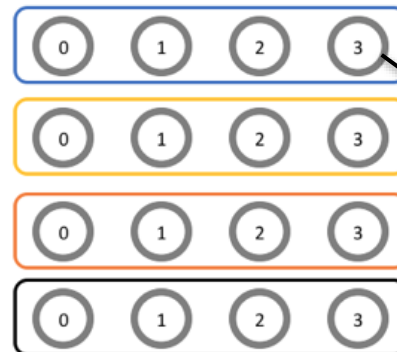
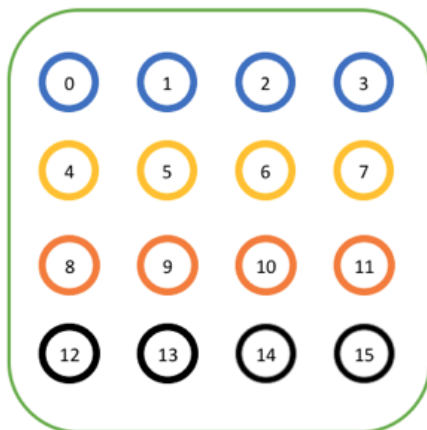
EXEMPLO COM SPLIT

```
// Obtém o rank e size do comunicador original
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor baseado na linha
int color = world_rank / 4;
// Divide o comunicador baseado na cor e
// usa o rank original para ordenamento
```



EXEMPLO COM SPLIT

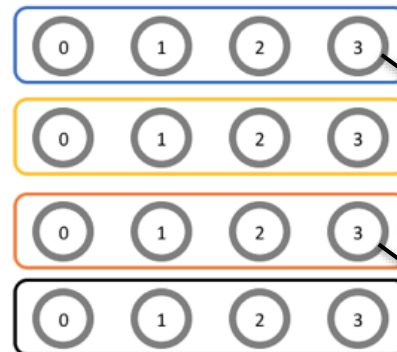
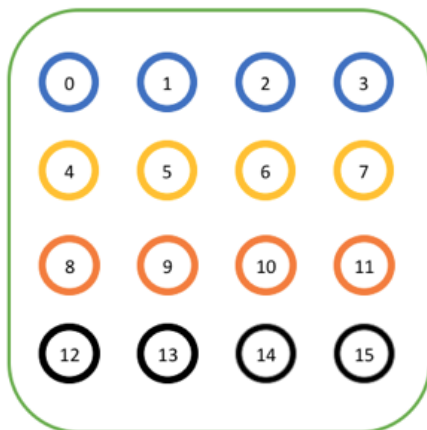
```
// Obtém o rank e size do comunicador original
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor baseado na linha
int color = world_rank / 4;
// Divide o comunicador baseado na cor e
// usa o rank original para ordenamento
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
// A variável row_comm de processo estará apontando para o
comunicador o qual o processo pertence!
```



A variável `row_comm` aponta para o comunicador de seu grupo

EXEMPLO COM SPLIT

```
// Obtém o rank e size do comunicador original
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor baseado na linha
int color = world_rank / 4;
// Divide o comunicador baseado na cor e
// usa o rank original para ordenamento
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
// A variável row_comm de processo estará apontando para o
comunicador o qual o processo pertence!
```



A variável `row_comm` aponta para o comunicador de seu grupo

A variável `row_comm` aponta para o comunicador de seu grupo

EXEMPLO COM SPLIT

```
// Obtém o rank e size do comunicador original
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor baseado na linha
int color = world_rank / 4;
// Divide o comunicador baseado na cor e
// usa o rank original para ordenamento
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
// A variável row_comm de processo estará apontando para o
// comunicador o qual o processo pertence!
int row_rank, row_size;
MPI_Comm_rank(row_comm, &row_rank);
MPI_Comm_size(row_comm, &row_size);

printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
       world_rank, world_size, row_rank, row_size);

MPI_Comm_free(&row_comm);
```

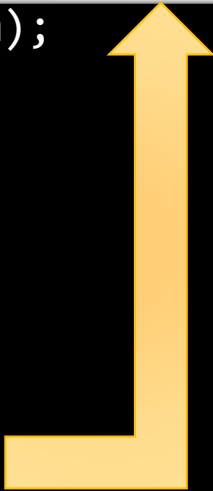
EXEMPLO COM SPLIT

```
// Obtém o rank e size ~$ mpirun -n 8 /home/mazalves/teste
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// Determina a cor base
int color = world_rank % 4;
// Divide o comunicador
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
// usa o rank original
MPI_Comm_rank(row_comm, &row_rank);
MPI_Comm_size(row_comm, &row_size);

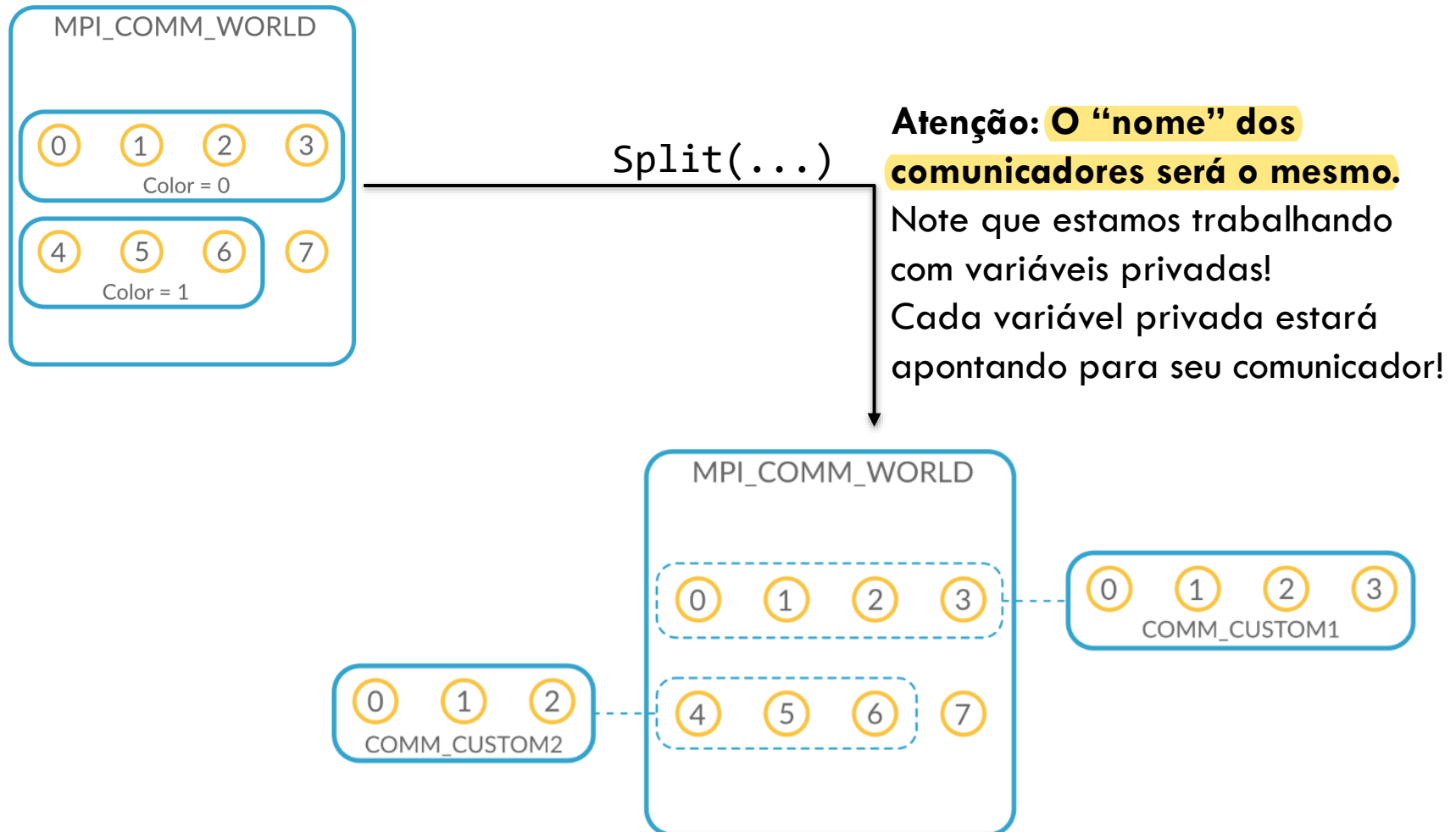
printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
       world_rank, world_size, row_rank, row_size);

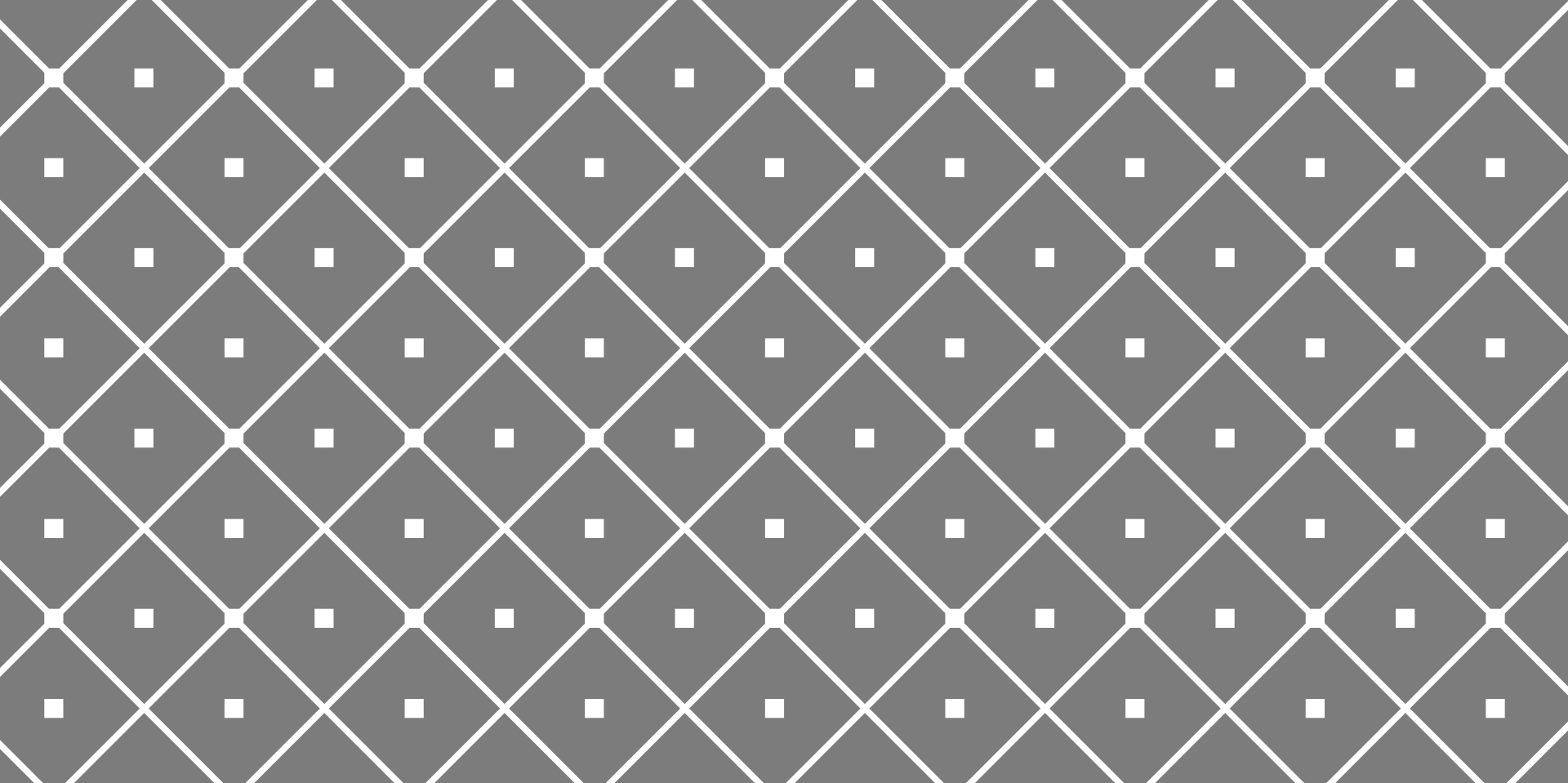
MPI_Comm_free(&row_comm);
```

WORLD RANK/SIZE: 0/8	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 1/8	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 3/8	ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 4/8	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 5/8	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 6/8	ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 7/8	ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 2/8	ROW RANK/SIZE: 2/4



ENTENDENDO O SPLIT



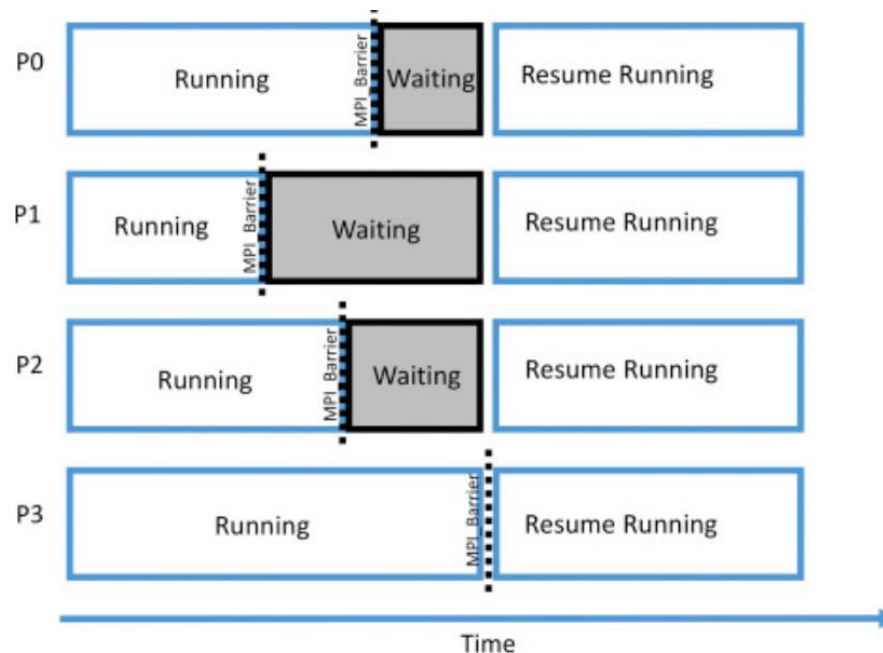


SINCRONIZADOR (BARREIRA)

BARRIER

```
int MPI_Barrier(MPI_Comm comm)
```

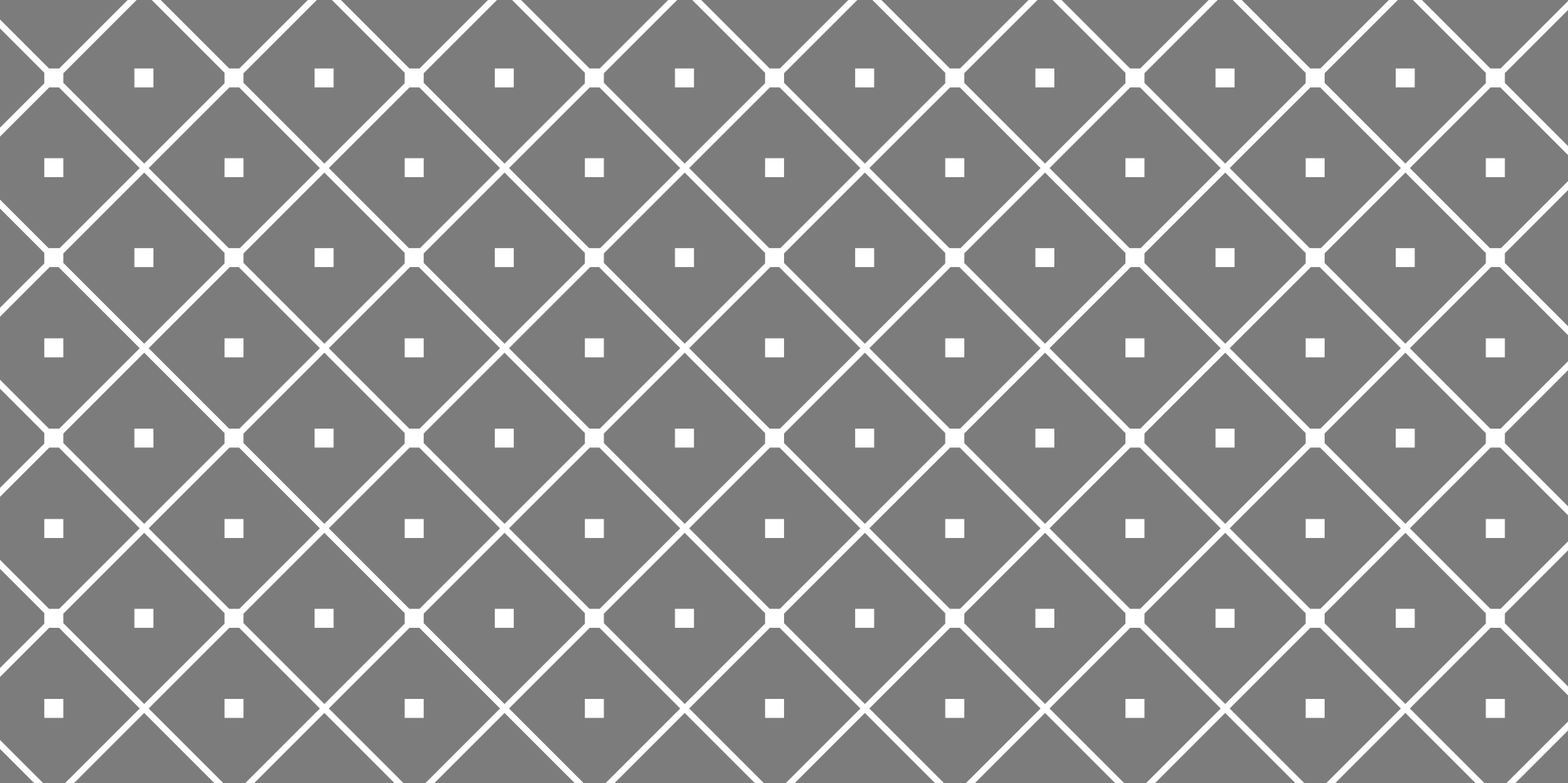
MPI_Barrier() sincroniza todos os processos no comunicador **comm**. Cada processo bloqueia até que todos os processos no comunicador chamem **MPI_Barrier()**.



IBARRIER

A variação **MPI_Ibarrier()** verifica se todos os processos passaram pela barreira

Mas não força os processos a aguardarem pelos demais.



AVALIANDO O TEMPO DE EXECUÇÃO E SINCRONIZAÇÃO

MEDINDO O TEMPO DE EXECUÇÃO

```
double MPI_Wtime(void)
```

MPI_Wtime() retorna o tempo em segundos que passou desde um determinado ponto arbitrário no passado.

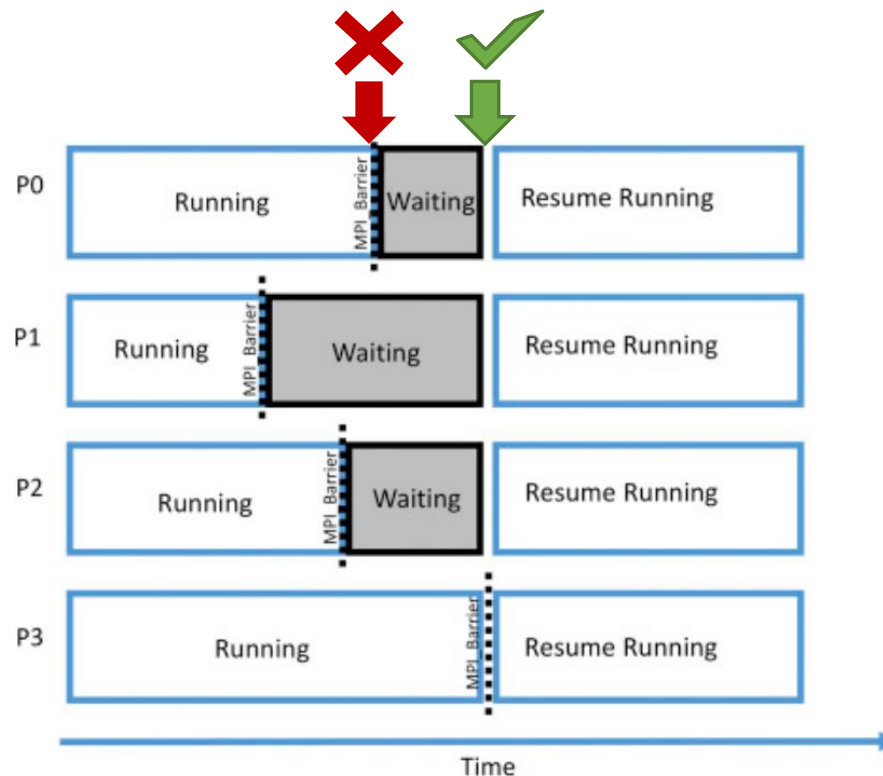
```
double MPI_Wtick(void)
```

MPI_Wtick() retorna a precisão da função **MPI_Wtime()**. Por exemplo, se **MPI_Wtime()** for incrementado a cada microsegundo então **MPI_Wtick()** retorna 0.000001. A precisão depende de como o hardware counter do clock for implementado na máquina.

MEDINDO O TEMPO DE EXECUÇÃO

Importante, devemos garantir que todos os processos estão sincronizados para medir o tempo de forma precisa!

MPI_Barrier() sincroniza todos os processos no comunicador.



MEDINDO O TEMPO DE EXECUÇÃO

```
double start, finish;
...
MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();
...
    // Parte da execução a medir ... (Muitos comandos)
...
MPI_Barrier(MPI_COMM_WORLD);
finish = MPI_Wtime();
if (my_rank == 0)
    printf("Tempo de Execução: %f segundos \n", finish - start);
...
// Os valores devolvidos por MPI_Wtime() são em tempo real, ou seja, todo o
tempo que o processo possa ter estado interrompido pelo sistema é
igualmente contabilizado.
```