

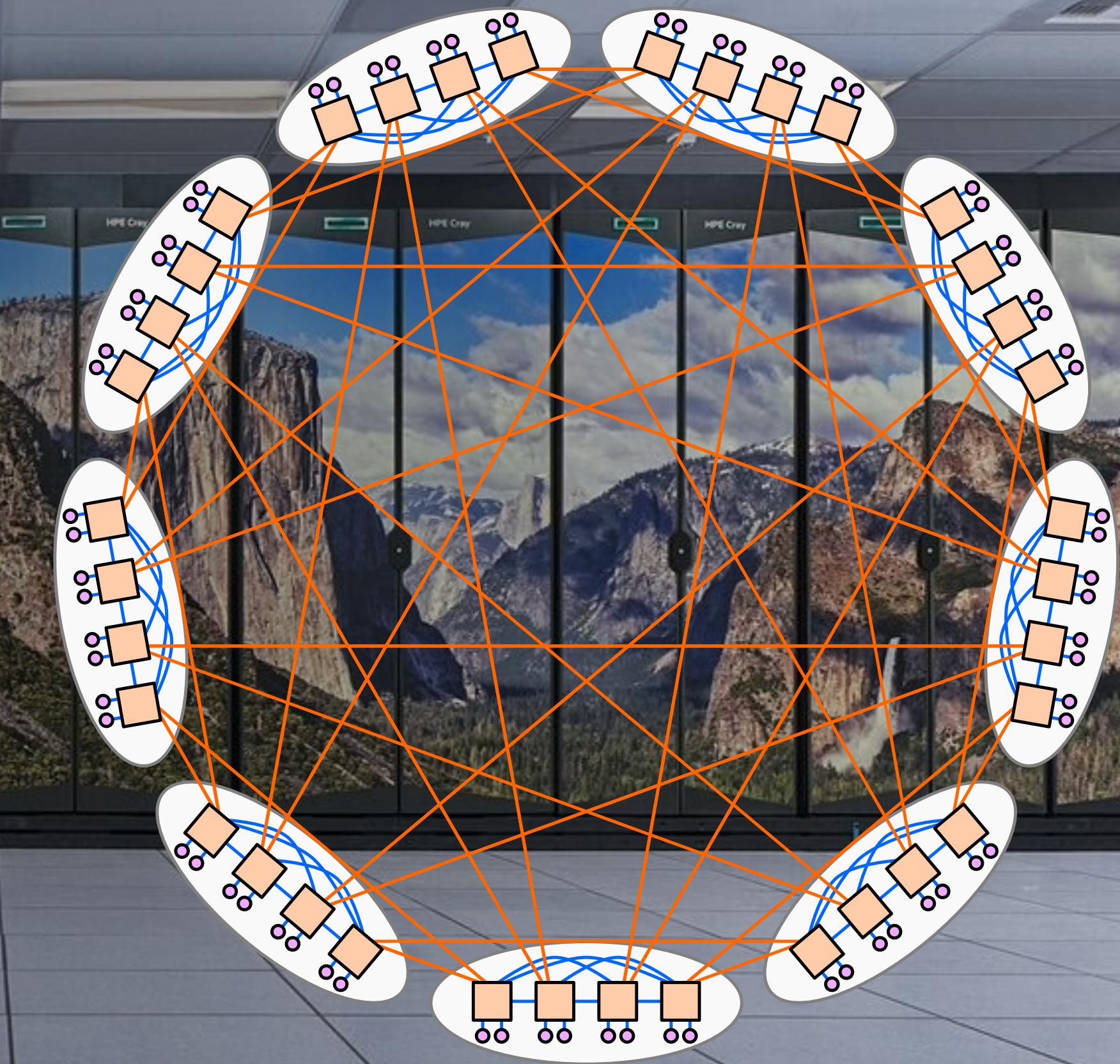
Introdução ao MPI

Sistemas distribuídos, Message Passing Interface e Simulações

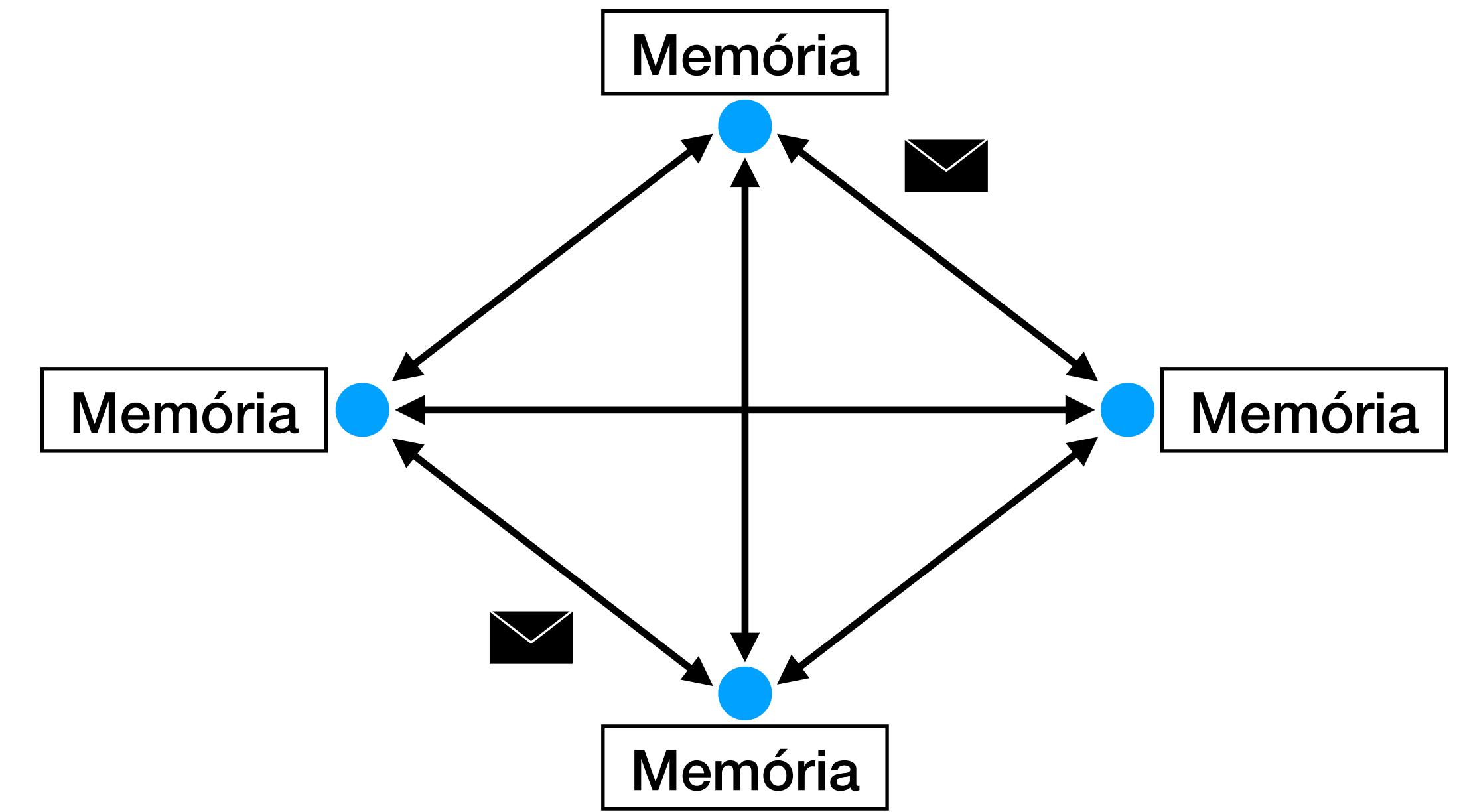
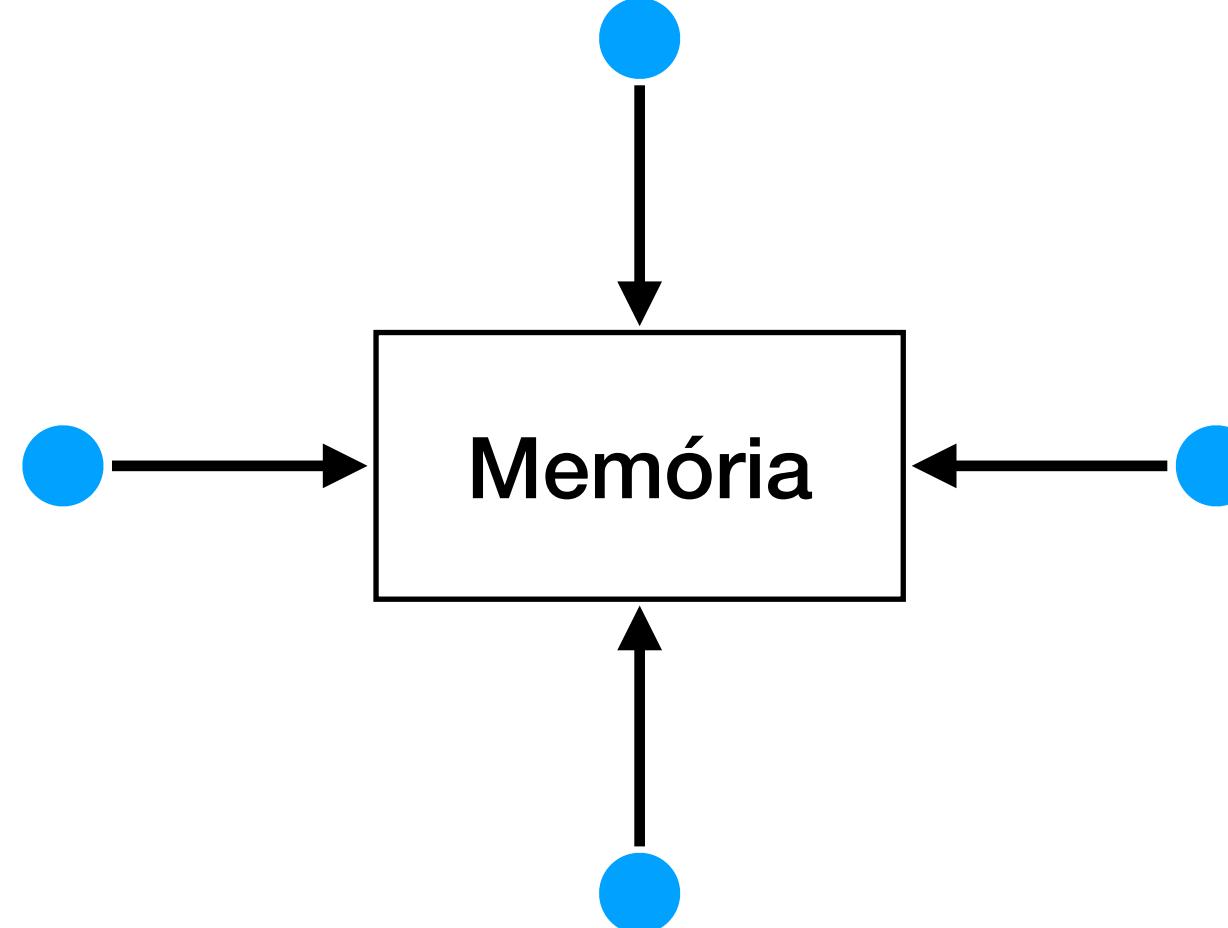
El Capitan #1 do TOP500 (nov. 2024)



Switch blade HPE Slingshot (200 Gbps)



Modelos de Programação Paralela



Ajuste natural

- Memória compartilhada = threads executando num único processador
- Memória distribuída = processos em servidores conectados pela rede

Distribuída Vs. Compartilhada

- Comunicação é explícita
 - O programador gerencia a comunicação
- Não tem boas técnicas automáticas para paralelizar o código
- Funciona melhor em uma configuração verdadeiramente distribuída

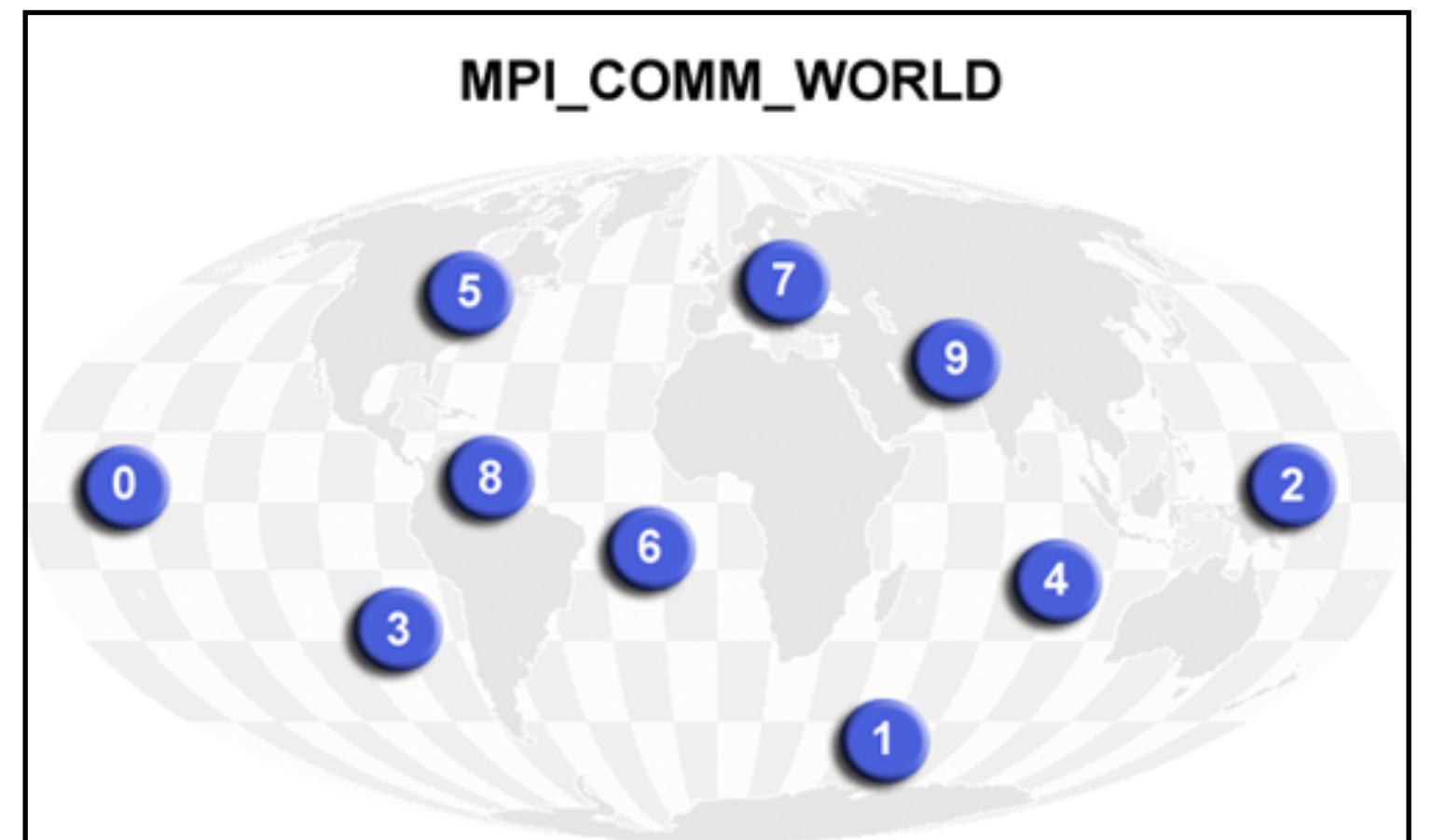
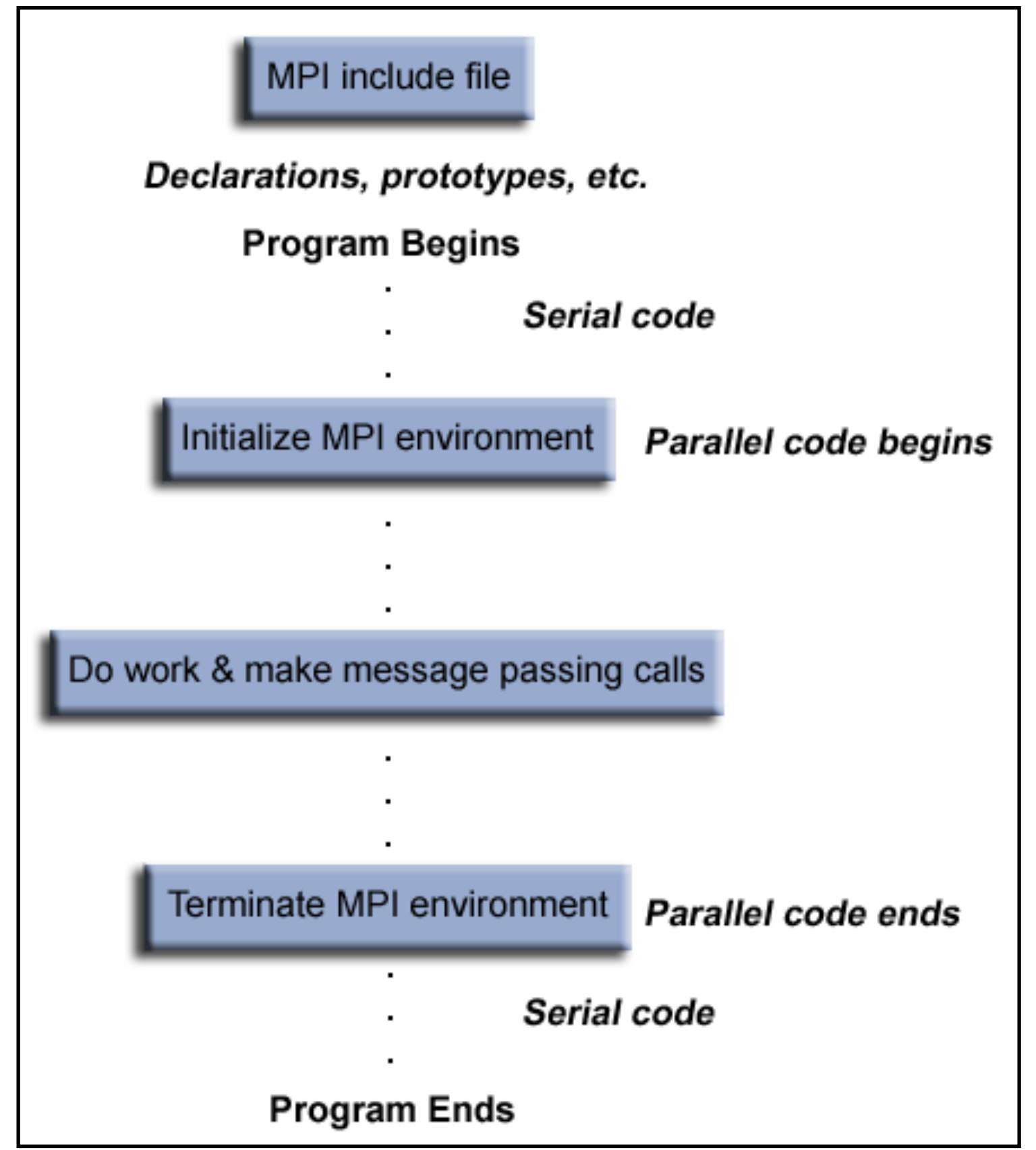
The Message Passing Interface (MPI)

<https://www mpi-forum.org/>

- MPI é um padrão
 - ▶ Define um **conjunto de operações** para programar aplicações que trocam mensagens
 - ▶ O padrão define a **semântica** das operações (não como elas são implementadas)
 - ▶ Última versão aprovada 4.1 (<https://www mpi-forum.org/mpi-41/>)
- **Open MPI** e **MPICH** são as duas principais implementações de código aberto (fornecem ligações C e Fortran)

Começando com MPI

- Exemplo: `helloworld.c`
 - ▶ Compilar: `mpicc -o helloworld helloworld.c`
 - ▶ Executar: `mpirun -np 2 ./helloworld`
- Comunicadores e grupos (`MPI_COMM_WORLD`)
- O *rank* identifica os processos dentro de um comunicador
 - ▶ `MPI_Comm_rank()` e `MPI_Comm_size()`
- Vamos detalhar a comunicação em MPI...



Mensagens em MPI

Uma mensagem inclui um *payload* (os dados) e metadados (chamados de envelope).

Metadados

- Rank dos processos (remetente e receptor)
- Um **Comunicador** (o contexto da comunicação)
- Uma **etiqueta** (pode ser usada para distinguir entre mensagens dentro de um comunicador)

Payload

- Endereço do início do buffer
- Número de elementos
- Tipo dos elementos

Comunicação Ponto a Ponto

```
int MPI_Send(const void *buf,  
            int count, MPI_Datatype datatype,  
            int dest, int tag, MPI_Comm comm);
```

```
int MPI_Recv(void *buf,  
            int count, MPI_Datatype datatype,  
            int source, int tag, MPI_Comm comm,  
            MPI_Status *status);
```

Mais detalhes em: https://hpc-tutorials.llnl.gov/mpi/routine_args

Modos de Comunicação

- **MPI_Send()** e **MPI_Recv()** são primitivas bloqueantes
- **MPI_Isend()** e **MPI_Irecv()** são primitivas não bloqueantes
- **Envio bloqueante:** Quando a chamada retorna, é seguro reutilizar o buffer que contém os dados a serem enviados
 - ▶ Não significa que os dados foram transferidos (ou que alguém já tenha “pedido” esse dado)
 - ▶ Pode ser que apenas uma cópia local dos dados tenha sido feita
- **Recebimento bloqueantes:** Quando a chamada retorna, os dados recebidos já estão no buffer

Modos de Comunicação

Blocking Send

```
myvar = 0;

for (i=1; i<ntasks; i++) {
    task = i;
    MPI_Send (&myvar ... ... task ...);
    myvar = myvar + 2

    /* do some work */

}
```

Non-blocking Send

```
myvar = 0;

for (i=1; i<ntasks; i++) {
    task = i;
    MPI_Isend (&myvar ... ... task ...);
    myvar = myvar + 2;

    /* do some work */

    MPI_Wait (...);
}
```

Safe. Why?

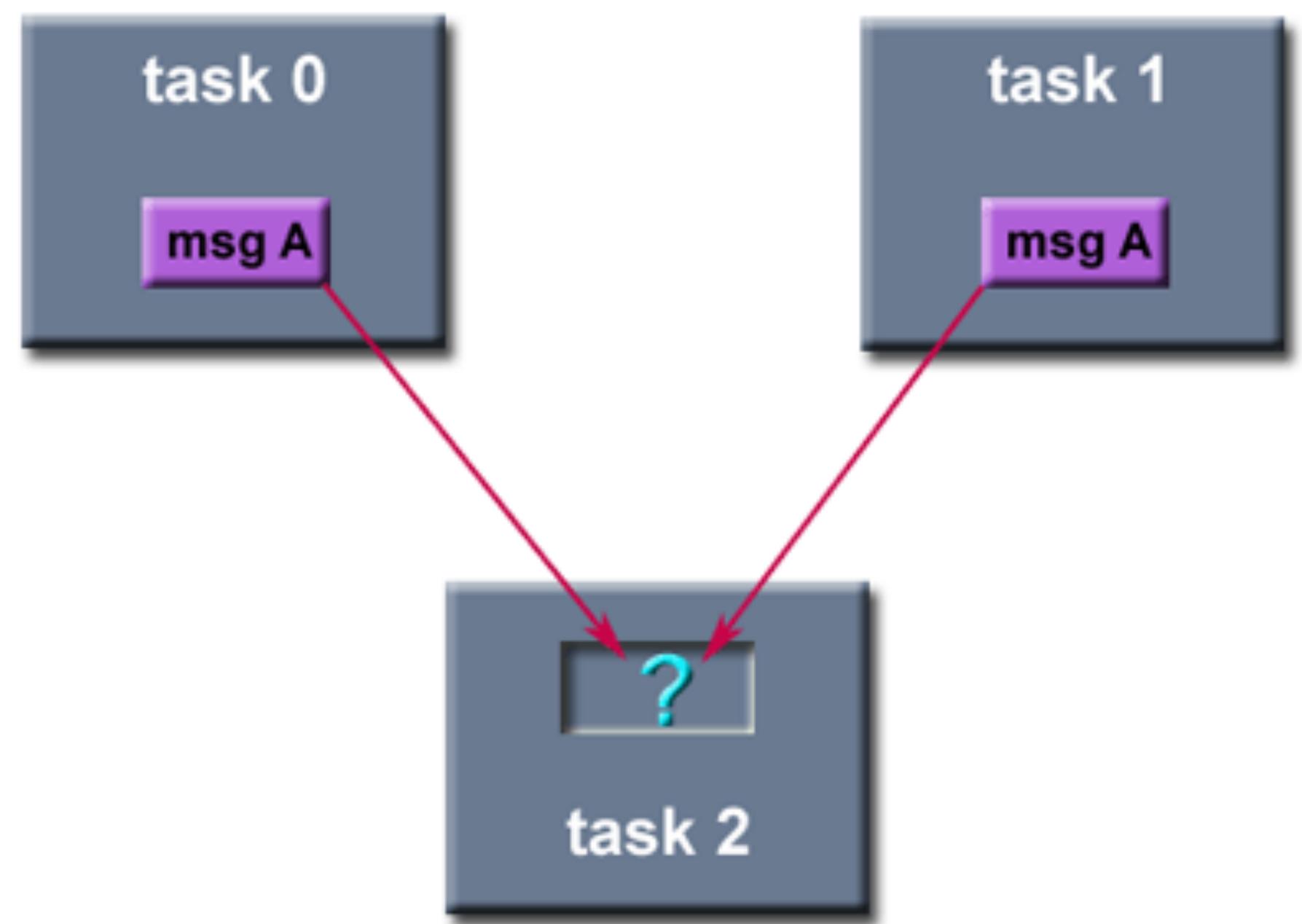
Unsafe. Why?

Modos de Comunicação

- As versões não bloqueantes retornam um **MPI_Request** que pode ser usado para verificar se a operação foi completada
 - Podemos **testar** se a operação foi concluída: **MPI_Test()**
 - Podemos **esperar** até que a operação seja concluída: **MPI_Wait()**
- Existem outras versões para validações múltiplas (sufixo **_any**, **_some**, **_all**)
- Exemplo: **dummyNonBlocking.c**

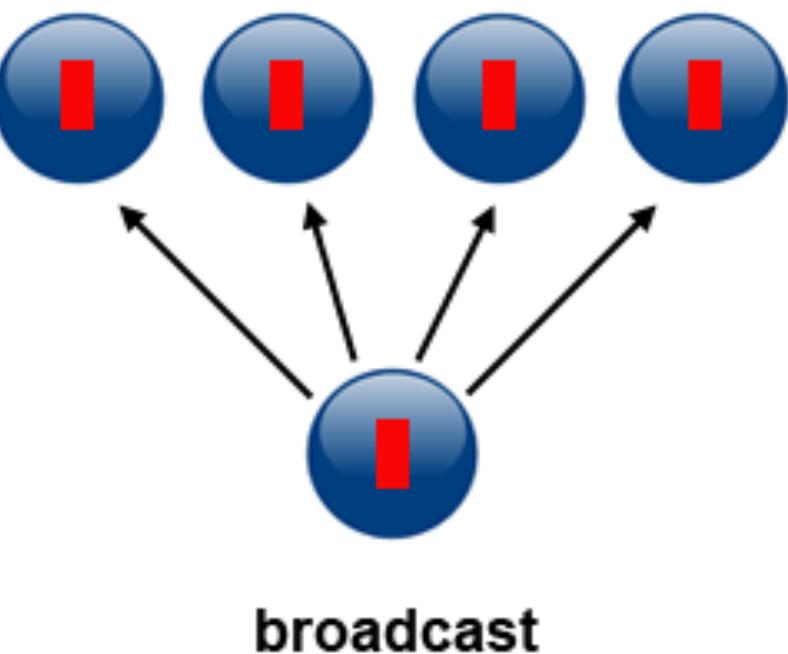
Ordem e Justiça

- Se um remetente enviar duas mensagens (Mensagem 1 e Mensagem 2) em sucessão para o mesmo destino, e ambas corresponderem ao mesmo recebimento, a operação de recebimento receberá a Mensagem 1 **ANTES** da Mensagem 2.
- A tarefa 0 envia uma mensagem para a tarefa 2. No entanto, a tarefa 1 envia uma mensagem concorrente que corresponde ao recebimento da tarefa 2. Apenas **UM** dos envios será concluído.

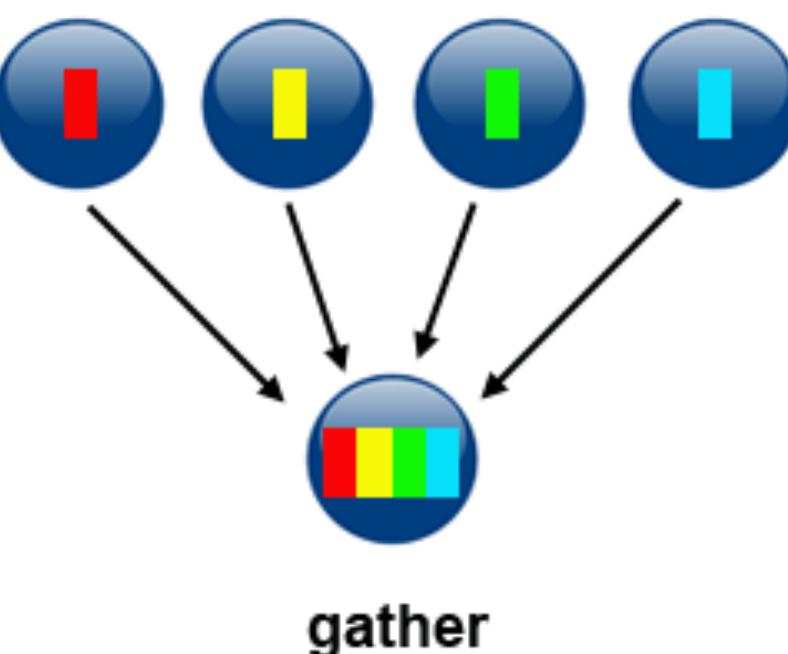


Comunicação Coletiva

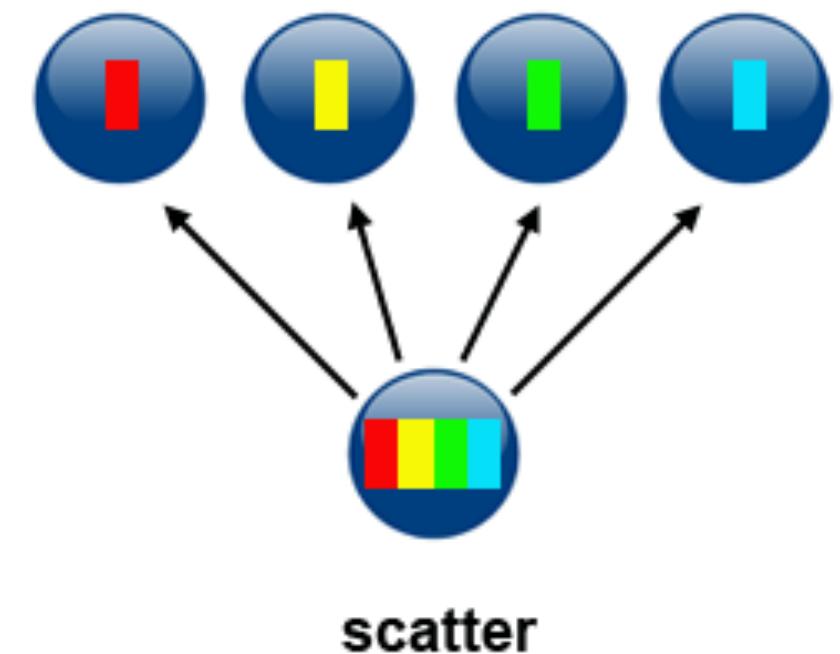
- Tipos de operações coletivas:
 - **Sincronização**
 - **Movimento de dados**
 - **Computação coletiva (reduções)**
- Exemplo: **scatter.c**
 - Mas antes...



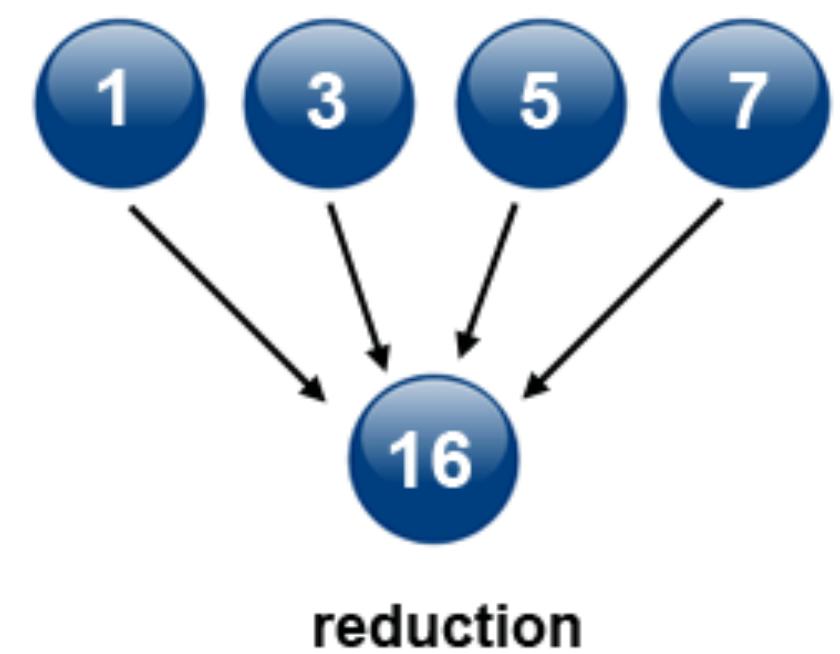
broadcast



gather



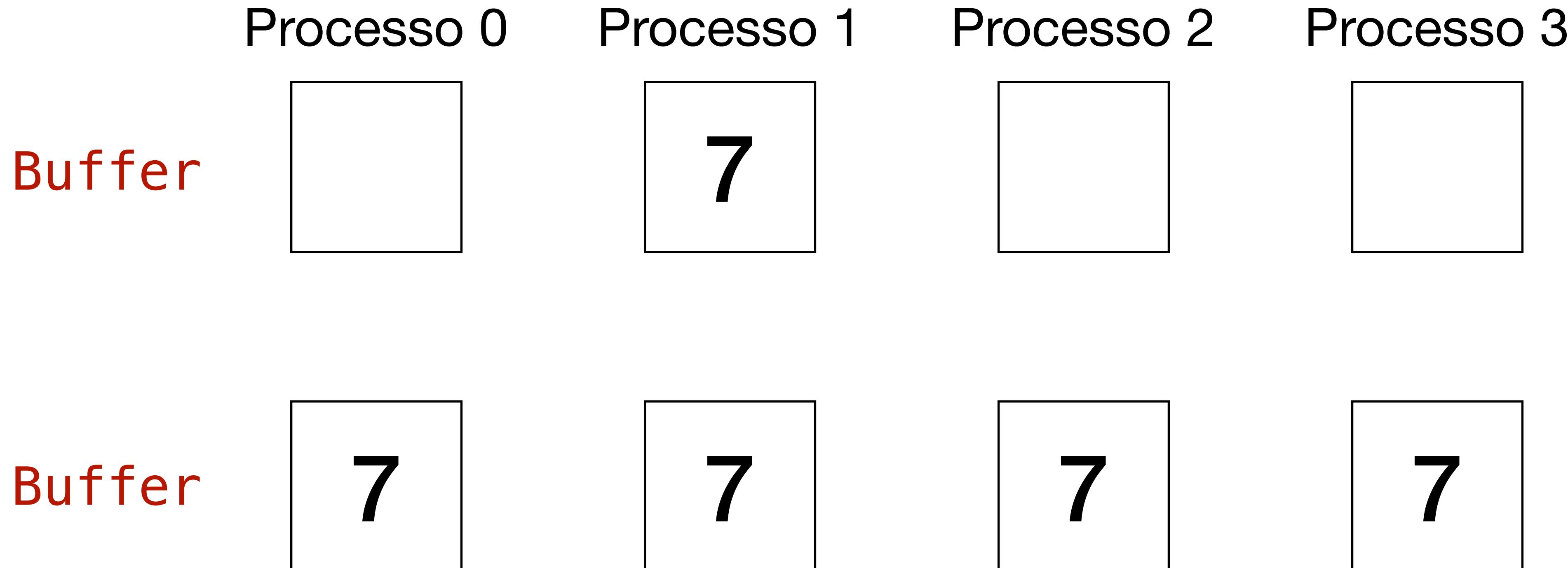
scatter



reduction

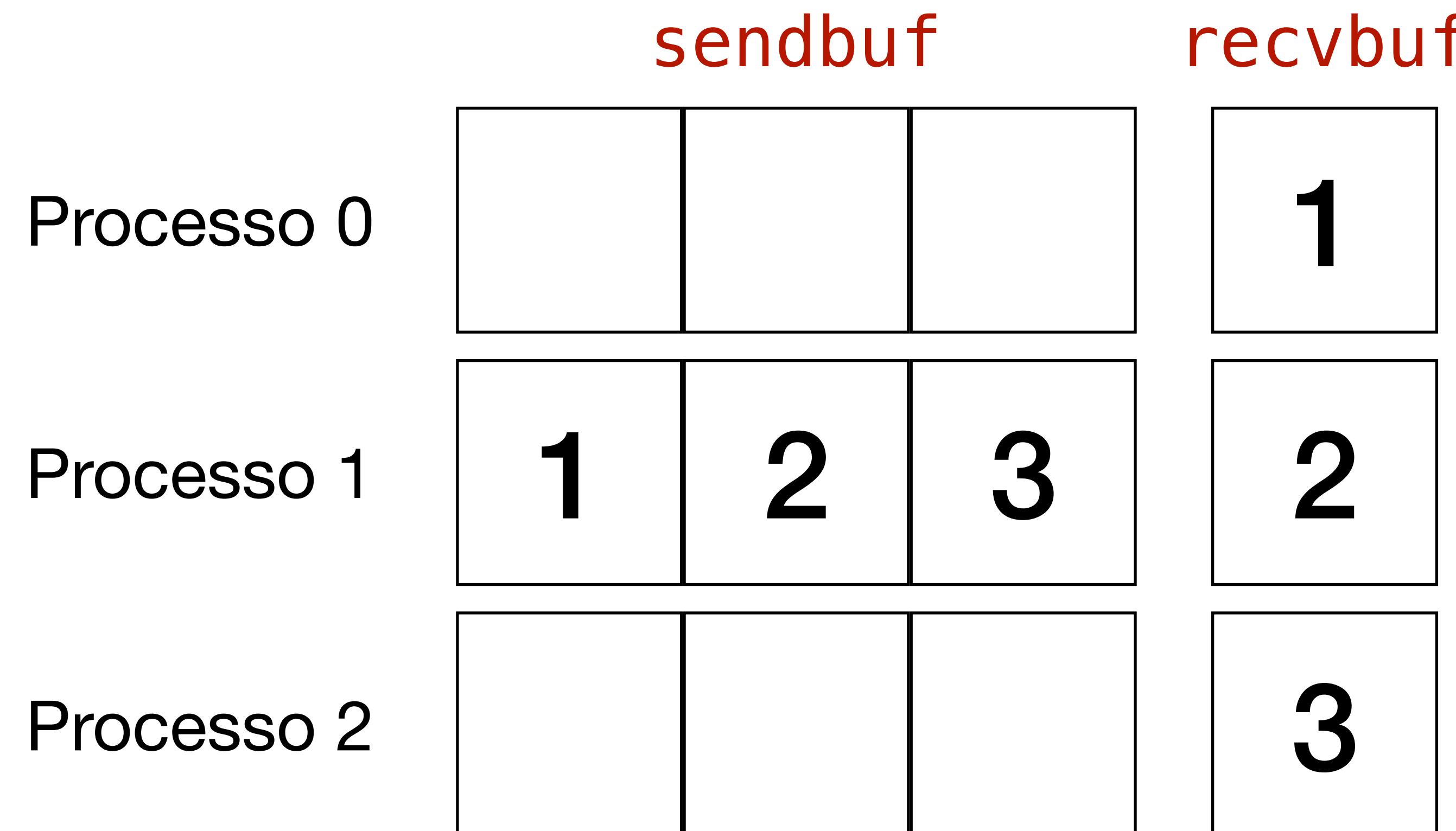
Comunicação Coletiva: Broadcast

```
MPI_Bcast(void *buffer,  
          int count, MPI_Datatype datatype,  
          int root, MPI_Comm comm)
```



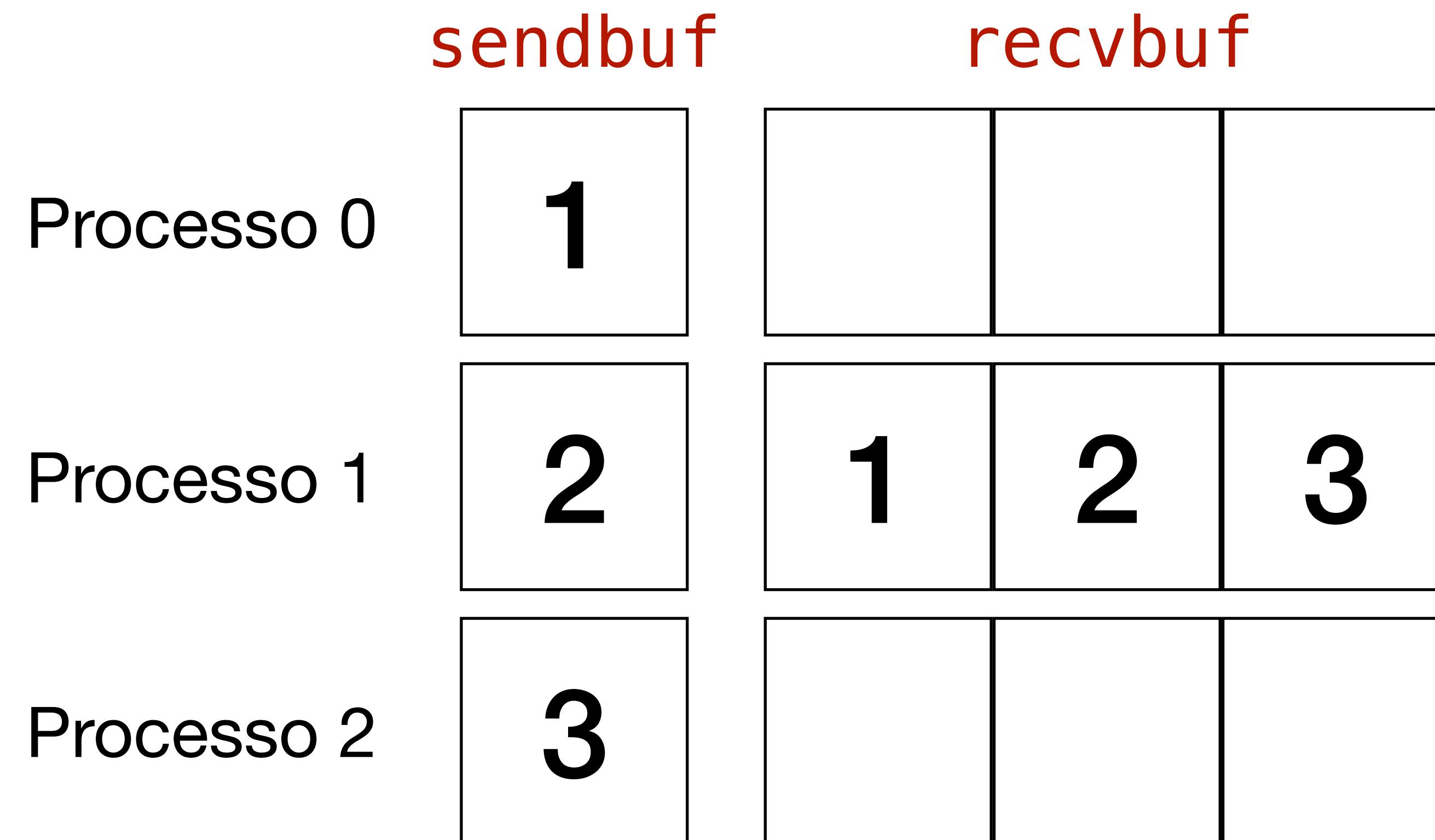
Comunicação Coletiva: Scatter

```
MPI_Scatter(void *sendbuf,  
           int sendcnt, MPI_Datatype sendtype,  
           void *recvbuf, int recvcnt, MPI_Datatype recvtype,  
           int root, MPI_Comm comm)
```



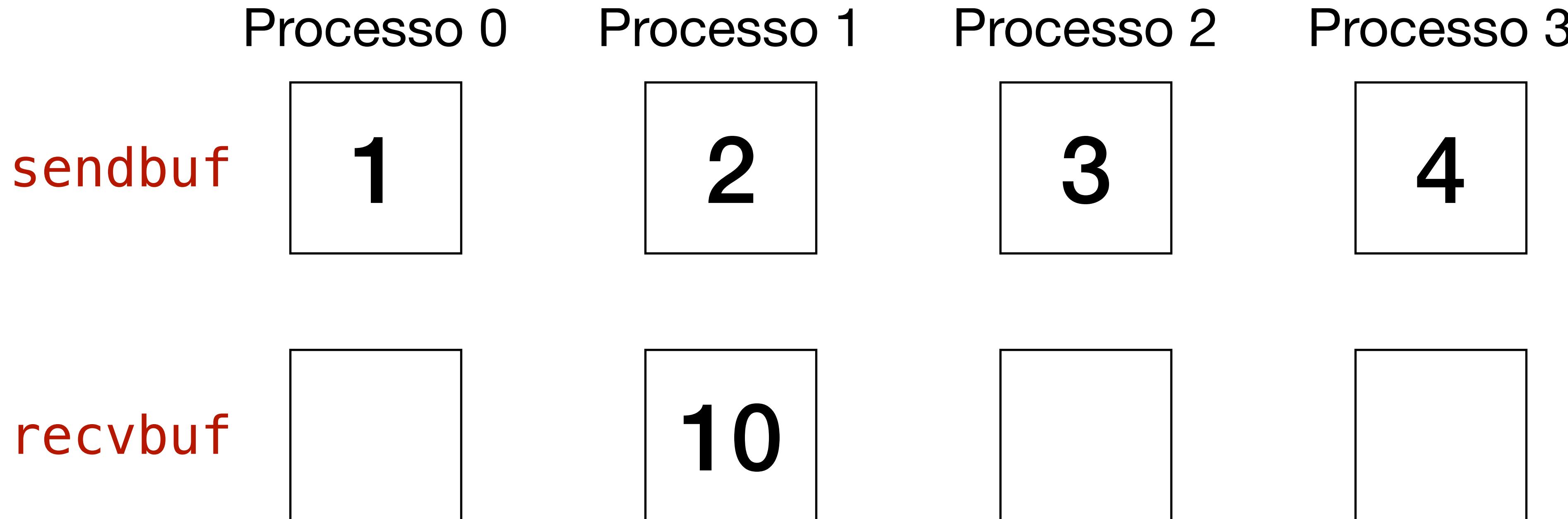
Comunicação Coletiva: Gather

```
MPI_Gather(void *sendbuf,  
          int sendcnt, MPI_Datatype sendtype,  
          void *recvbuf, int recvcnt, MPI_Datatype recvtype,  
          int root, MPI_Comm comm)
```



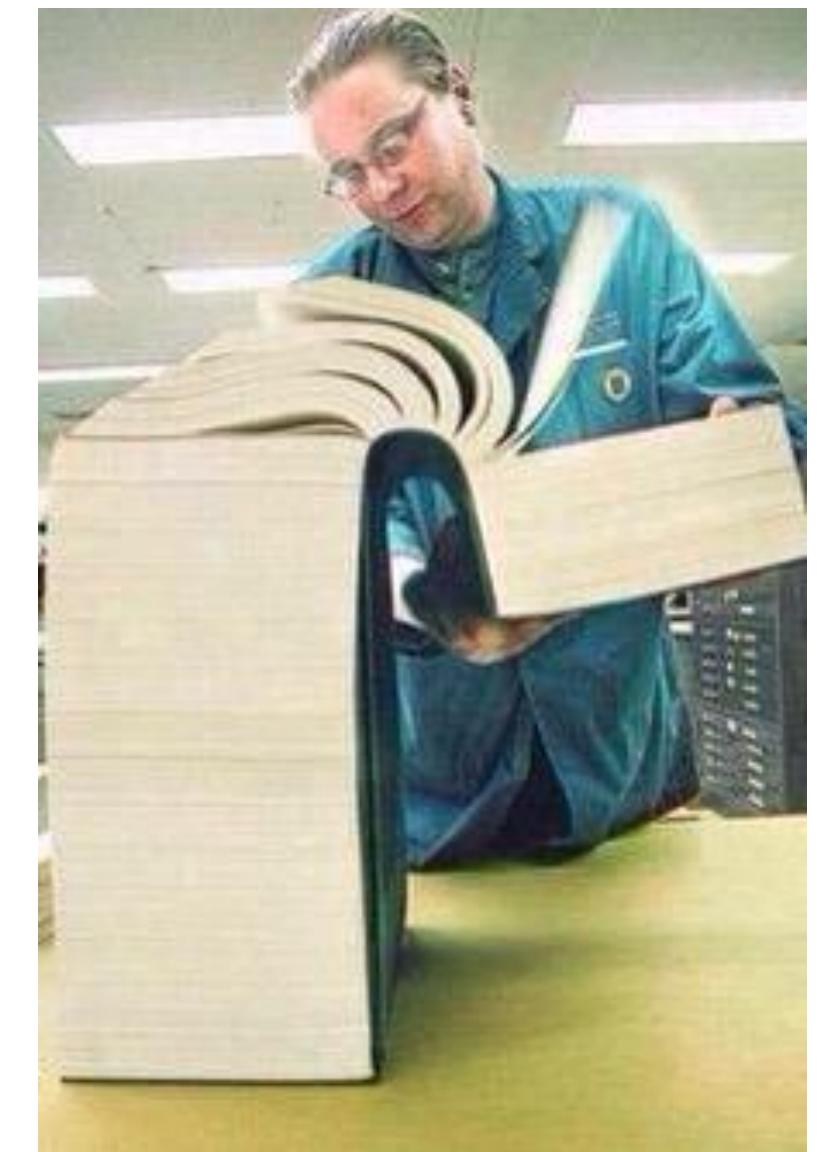
Comunicação Coletiva: Reduce

```
MPI_Bcast(void *sendbuf,  
          void *recvbuf, int count, MPI_Datatype datatype,  
          MPI_Op op, int root, MPI_Comm comm)
```



Antes de Avançar

- O objetivo desta aula é apenas fornecer uma visão geral da interface MPI.
- O padrão 4.1 do MPI é um documento de 1166 páginas 😱
- O que ainda não vimos:
 - ▶ Tipos de dados derivados
 - ▶ Gestão de grupos e comunicadores
 - ▶ Coletivas não bloqueantes
 - ▶ Topologias virtuais



AR21

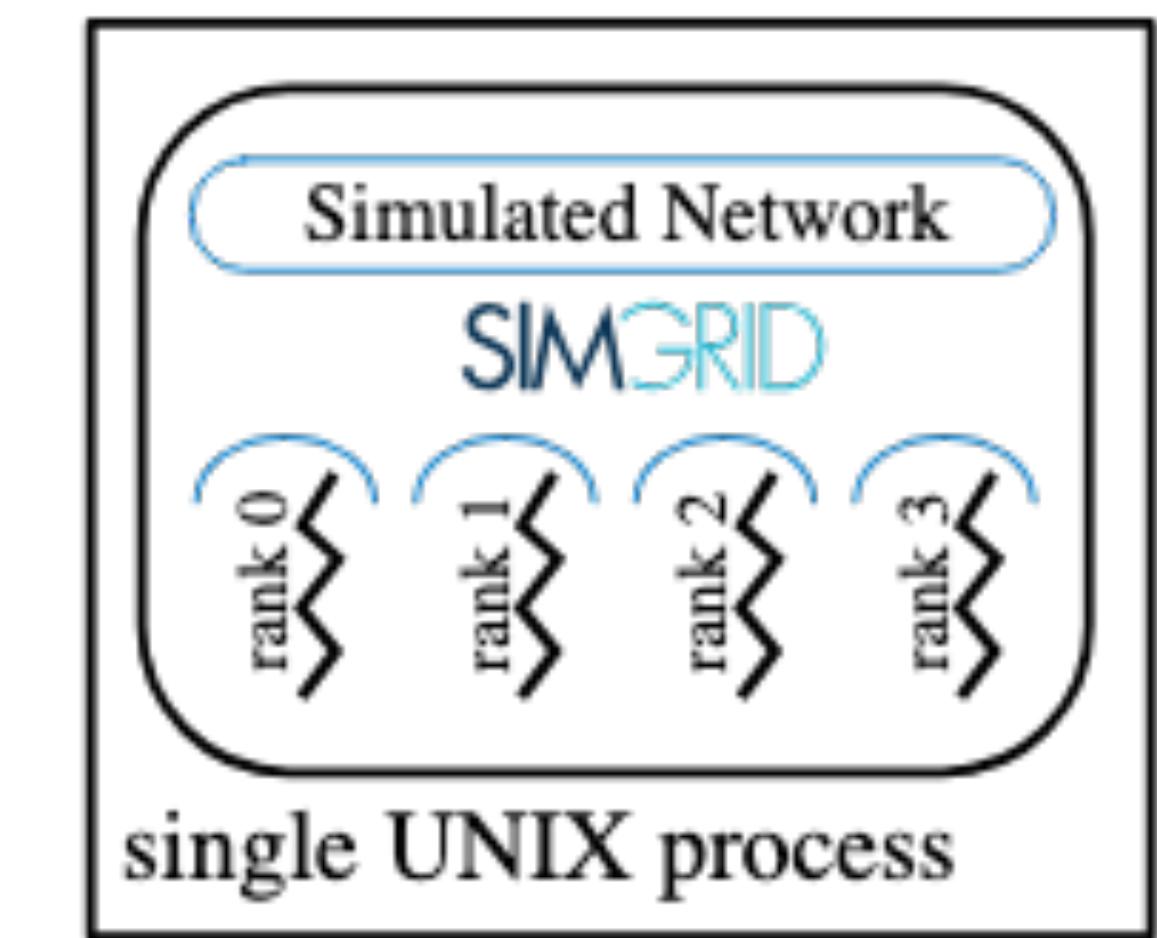
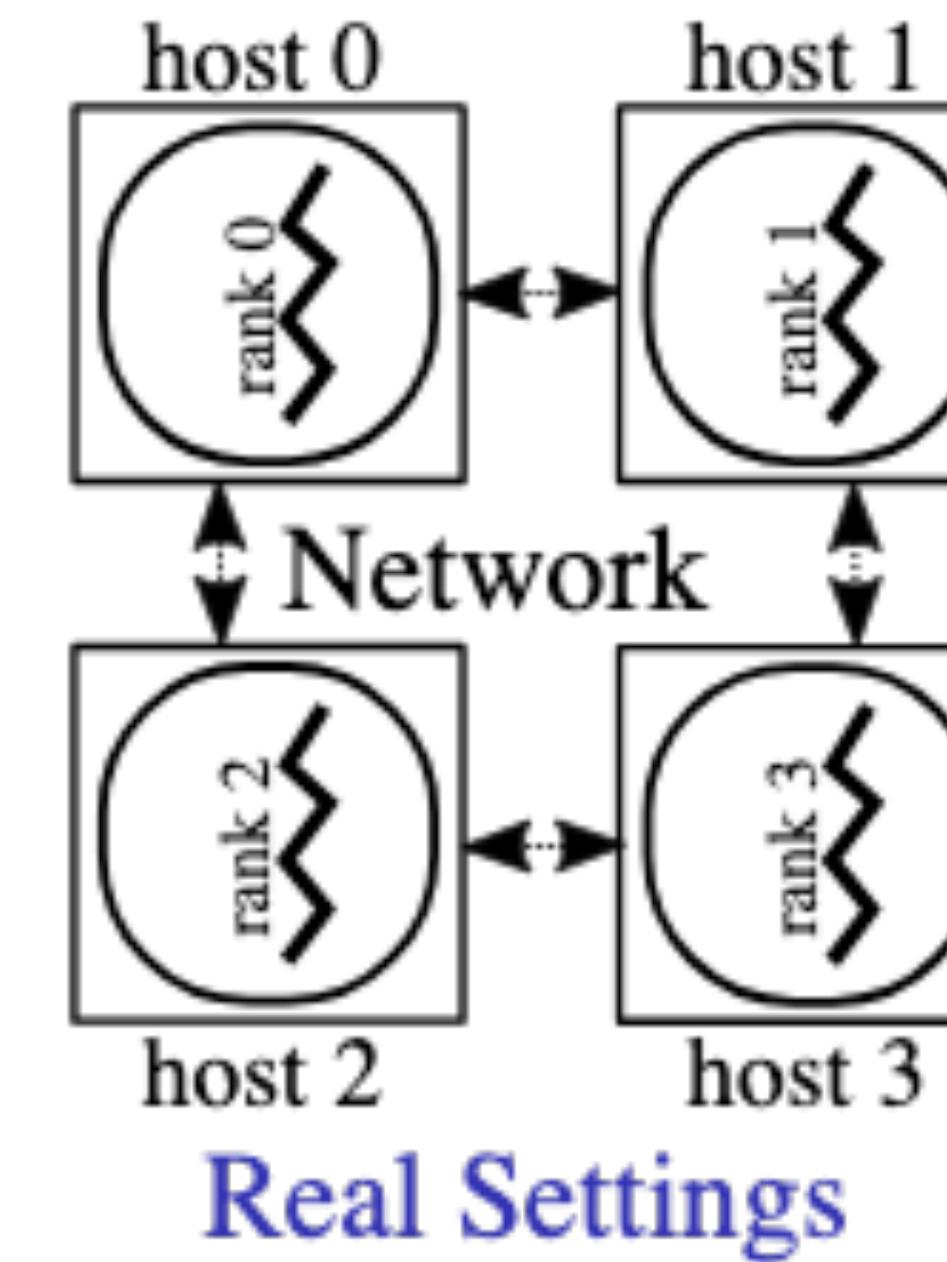
Como executar MPI em um sistema distribuído?

FRONTIER



Simulando um Sistema Distribuído

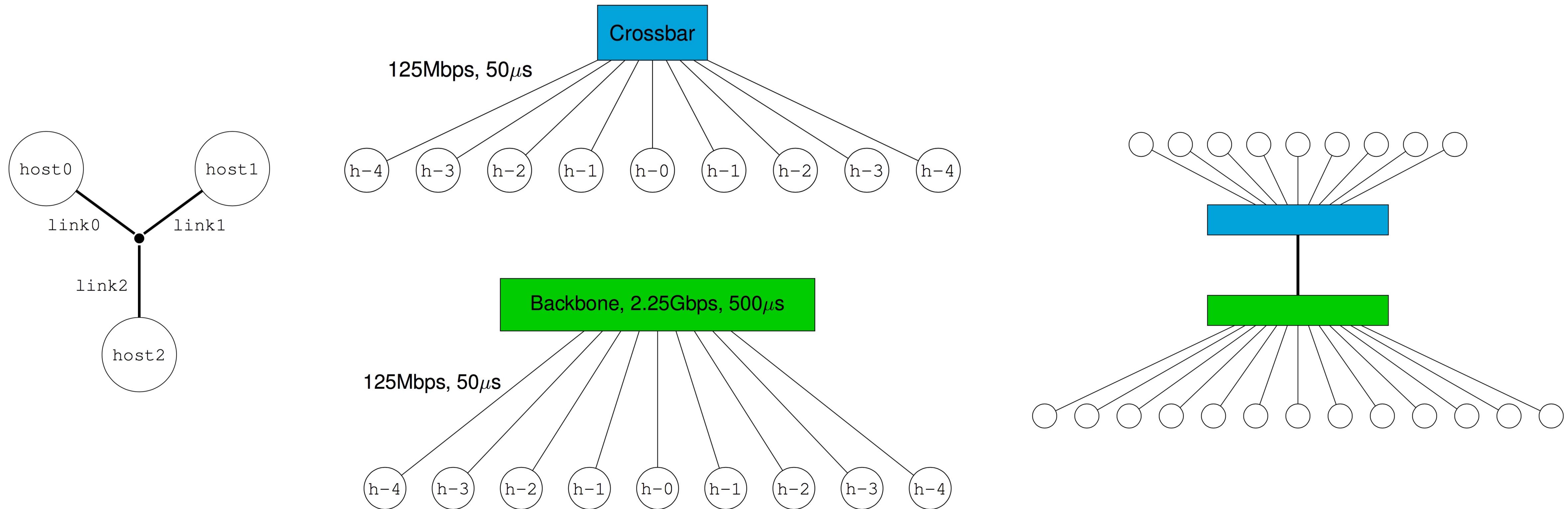
- **SimGrid** é um framework para desenvolver simuladores de aplicações distribuídos direcionados a plataformas distribuídas
- Módulo **SMPI** simula as comunicações enquanto os cálculos são emulados



Descrevendo a Plataforma Simulada

- Como plataformas são representadas no SimGrid?
 - Conjunto de *hosts* e *links* de rede, com algumas características de desempenho
 - Arquivo XML
- Vamos ver alguns exemplos...

Descrevendo a Plataforma Simulada



Mais detalhes: https://simgrid.github.io/SMPI_CourseWare/topic_getting_started/platforms/

Rodando um Código MPI no SimGrid

- Exemplo: `roundtrip.c`
 - ▶ Compilar: `smpicc -O4 roundtrip.c -o roundtrip`
 - ▶ Executar: `smpirun -np 16 -hostfile ./cluster_hostfile.txt -platform ./cluster_crossbar.xml ./roundtrip`
- Pode aparecer alguns avisos/informações sobre a configuração de alguns parâmetros do SMPI. **Podemos ignorar**
 - ▶ `MPI_Get_processor_name(char *name, int *resultlen)`
- Exemplo de mundo real: https://simgrid.org/doc/latest/Tutorial_MPI_Applications.html#lab-1-visualizing-lu

EP3 - Mais fractais

- Baseado no problema: [https://simgrid.github.io/SMPI_CourseWare/
topic basics of distributed memory programming/julia set/](https://simgrid.github.io/SMPI_CourseWare/topic_basics_of_distributed_memory_programming/julia_set/)
- Implementar uma versão sequencial e duas versões paralelas com MPI
 - Escrita “colaborativa” (todos os processos escrevem)
 - Escrita centralizada (com comunicação coletiva)
- Observar a escalabilidade da implementação paralela e comparar com a sequencial
- Comparar o desempenho dos diferentes tipos de escrita

Referências

- <https://hpc-tutorials.llnl.gov/mpi/>
- https://simgrid.github.io/SMPI_CourseWare/
- https://fdesprez.github.io/teaching/par-comput/lectures/slides/7_MPI.pdf
- https://www.youtube.com/watch?v=Zu5JcxZt_f8&list=PLxNPSjHT5qvsGKsAhirvZn7W73pXhXpfv
- Esta apresentação e os exemplos utilizados se encontrar no repositório **fredgrub/aulas-PPD** no GitHub.