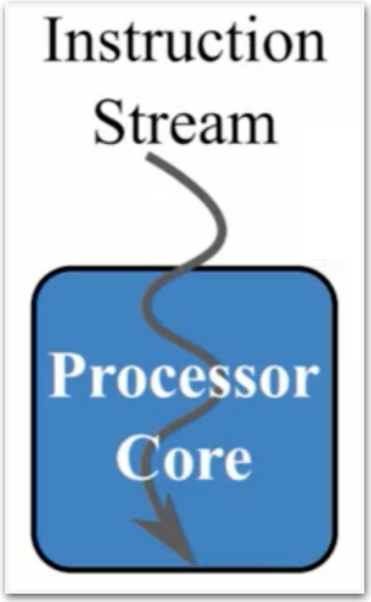
Programación paralela

MPI

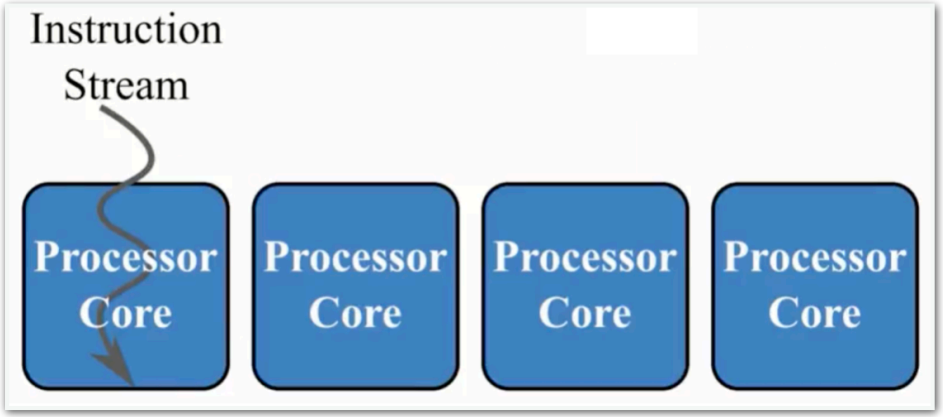
Gabriel Astudillo Muñoz

Descripción General

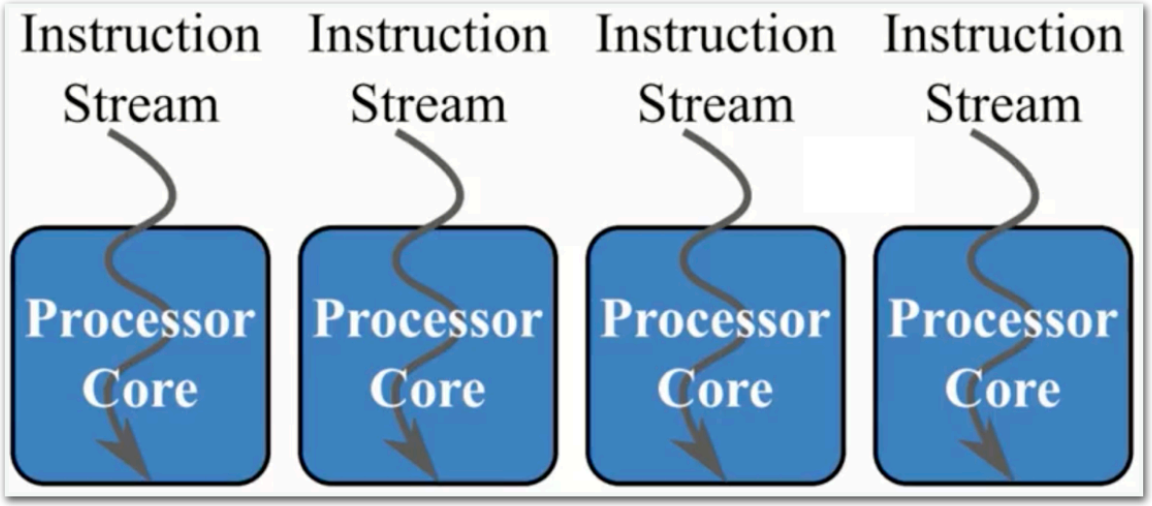
LP secuencial, Código secuencial



LP paralelo, Código secuencial

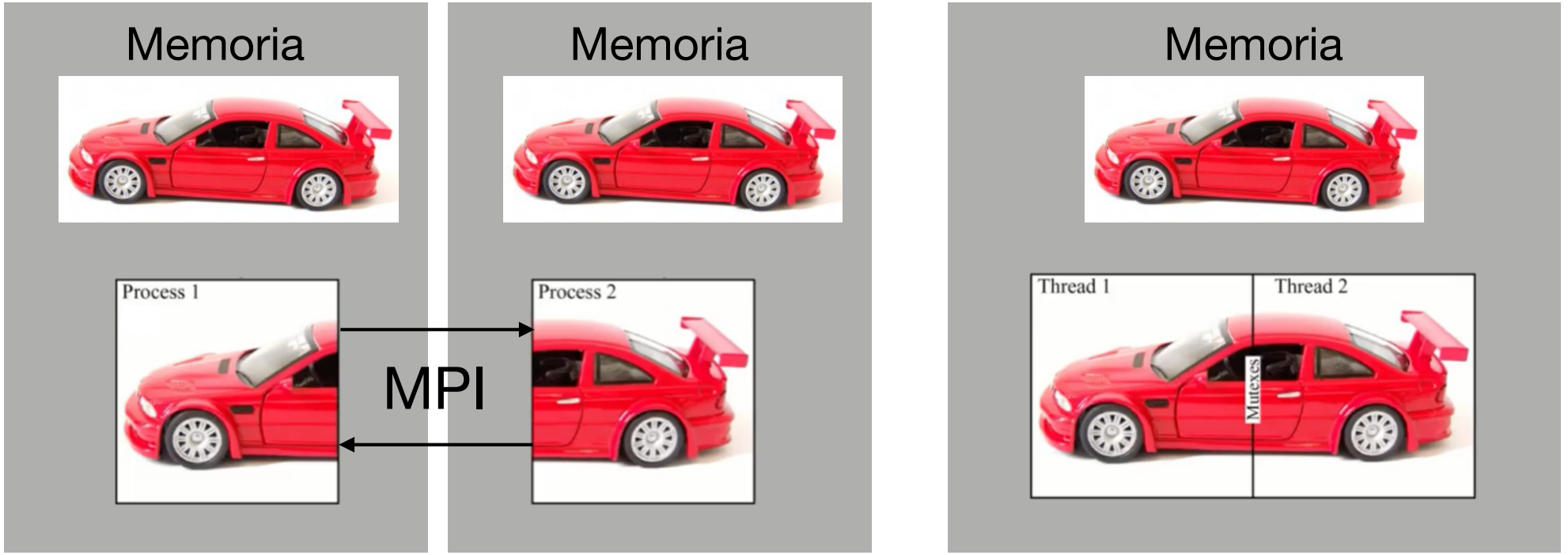


LP paralelo, Código paralelo



Procesos v/s threads

Particionar el data-set entre procesos o threads



similar a crear procesos con fork()

Framework	Functionality
C++11 Threads	Asynchronous functions; only C++
POSIX Threads	Fork/join; C/C++/Fortran; Linux
Cilk Plus	Async tasks, loops, reducers, load balance; C/C++
TBB	Trees of tasks, complex patterns; only C++
OpenMP	Tasks, loops, reduction, load balancing, affinity, nesting, C/C++/Fortran (+SIMD, offload)

MPI

Message Passing Interface

Especificación para la transferencia y recepción de mensajes entre procesos

Permite el multiprocesamiento en sistemas con memoria compartida

https://www.mpi-forum.org

Existen dos grandes implementaciones del estándar

https://www.open-mpi.org

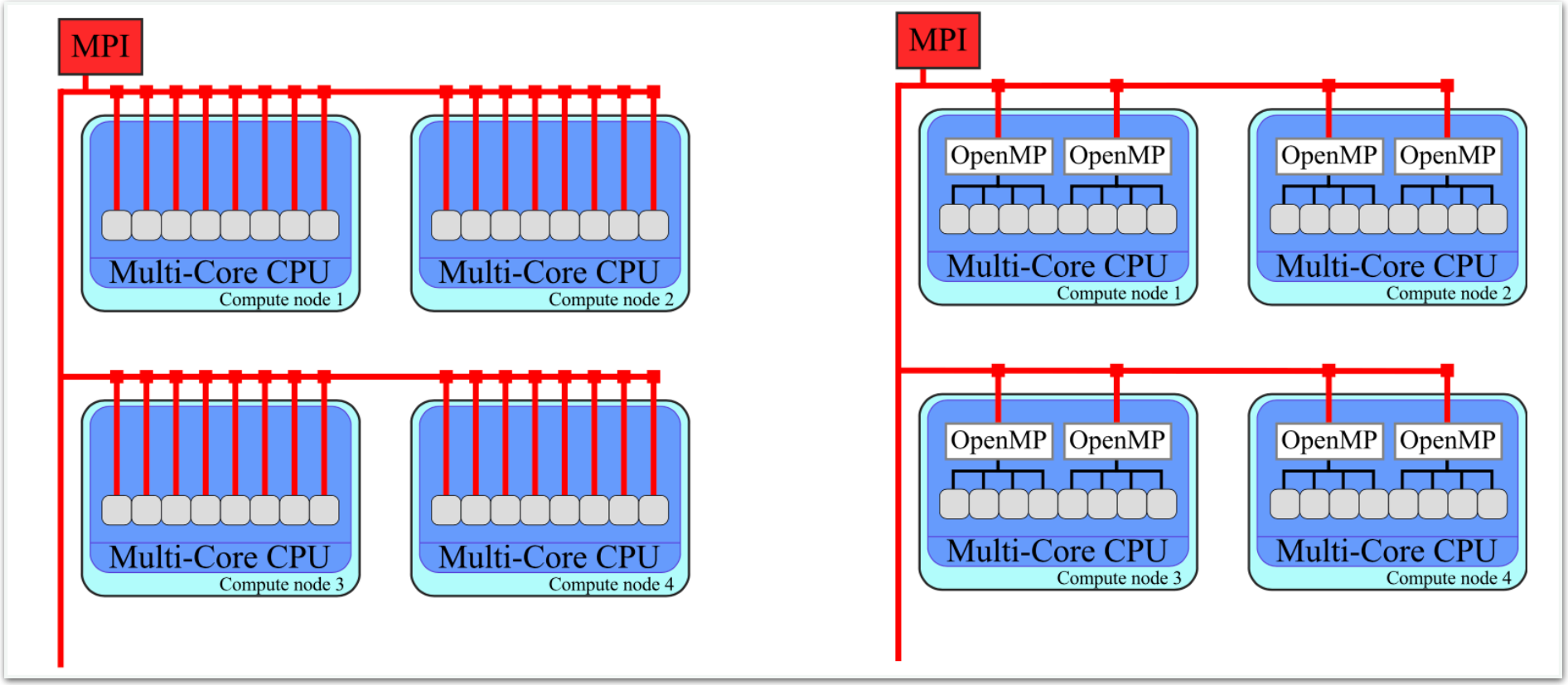
https://www.mpich.org

Debian

mpi-default-dev
openmpi-bin
openmpi-common

libboost-mpi1.71.0

Modelo de uso



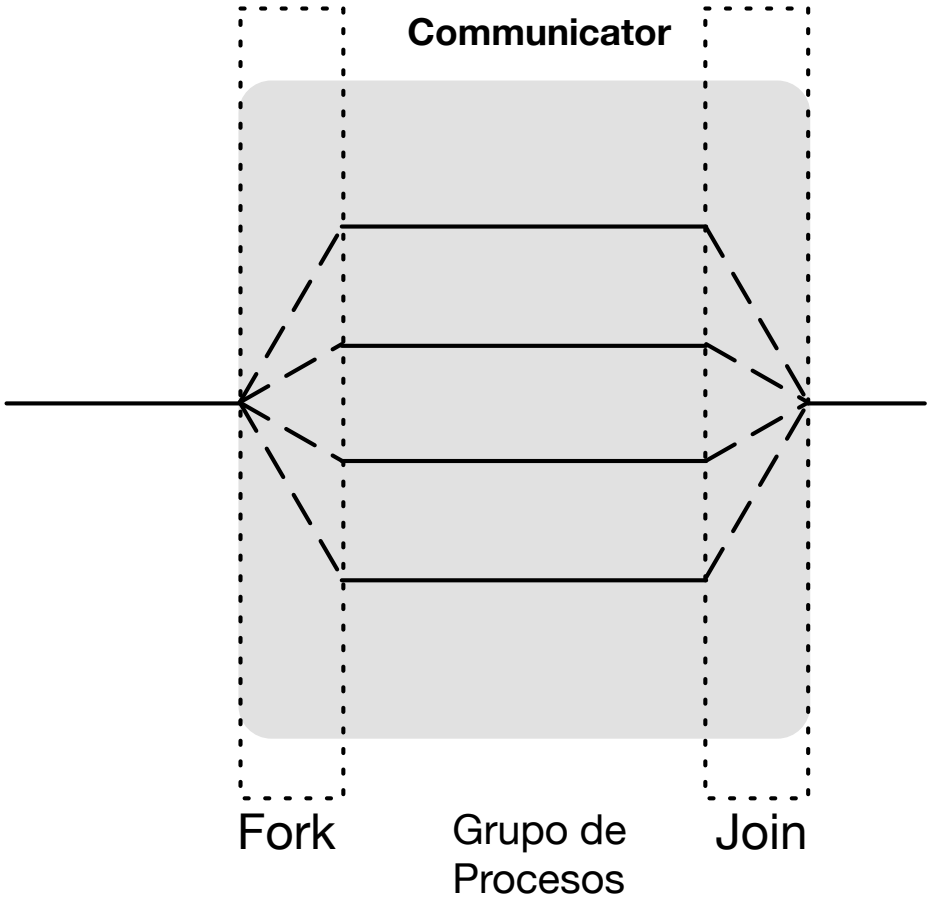
Baja cantidad de cores
(hasta 16 cores/nodo)

Alta cantidad de cores
(desde 16 cores/nodo)

Modelo de programación

El proceso principal genera un grupo (team) de procesos (modelo FORK-JOIN)

El paralelismo se agrega incrementalmente a la solución secuencial

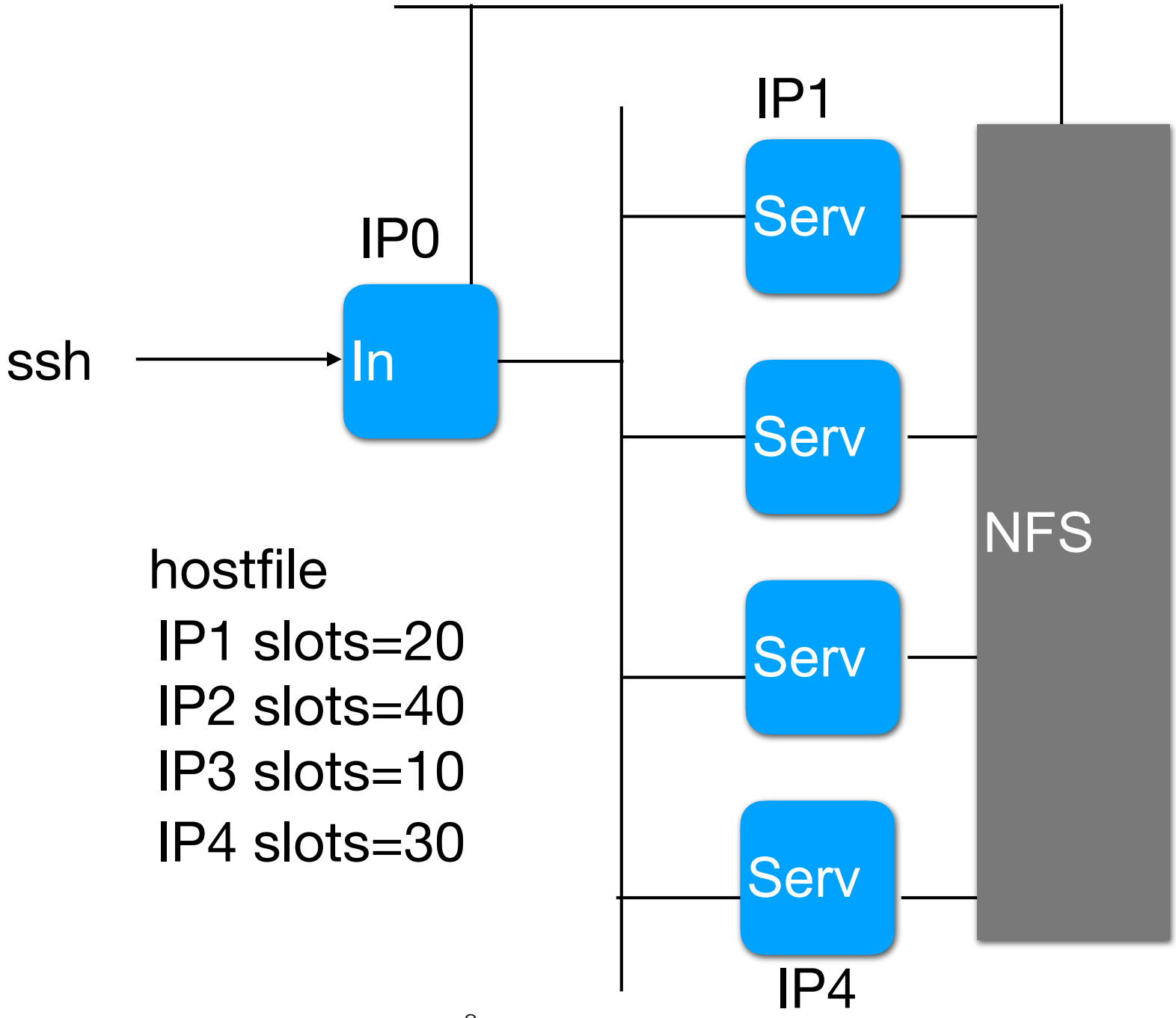


RANK

Identificador de cada proceso

Communicator Size

Cantidad de procesos


IP1
IP0
Serv
ssh
In
Serv
NFS
hostfile
IP1 slots=20
IP2 slots=40
IP3 slots=10
IP4 slots=30
Serv
Serv
IP4

Estructura del código, compilación y ejecución

Se utilizará la librería boost::mpi

Accede a las funciones de MPI a través de C++

https://www.boost.org/doc/libs/1_76_0/doc/html/mpi.html

```cpp
#include <boost/mpi.hpp>
#include <iostream>
#include <random>

namespace mpi = boost::mpi;

int main(int argc, char *argv[])
{
    mpi::environment env{argc, argv};
    mpi::communicator world;

    //Cada proceso genera un número real
    auto drand = [](double min, double max){
        std::random_device rd;  //seed for the random number engine
        std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
        std::uniform_real_distribution<> distrib(min, max);

        return distrib(gen);
    };

    double fooNumber = drand(0,1000);

    uint32_t commSize = world.size();
    uint32_t rank     = world.rank();

    std::cout << "Process:" << rank << " of " << commSize << ", fooNumber:"<< fooNumber << '\n';

}
```

Se utilizará la librería boost::mpi

Accede a las funciones de MPI a través de C++

```cpp
#include <boost/mpi.hpp>
#include <iostream>
#include <random>

namespace mpi = boost::mpi;

int main(int argc, char *argv[])
{
    mpi::environment env{argc, argv};
    mpi::communicator world;

    //Cada proceso genera un número real
    auto drand = [](double min, double max){
        std::random_device rd;  //seed for the random number engine
        std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
        std::uniform_real_distribution<> distrib(min, max);

        return distrib(gen);
    };

    double fooNumber = drand(0,1000);

    uint32_t commSize = world.size();
    uint32_t rank     = world.rank();

    std::cout << "Process:" << rank << " of " << commSize << ", fooNumber:"<< fooNumber << '\n';

}
```

Inicialización del ambiente MPI

Cantidad de procesos del ambiente e identificador del proceso

Compilación

```cpp
#include <boost/mpi.hpp>
#include <iostream>
#include <random>

namespace mpi = boost::mpi;

int main(int argc, char *argv[])
{
    mpi::environment env{argc, argv};
    mpi::communicator world;

    //Cada proceso genera un número real
    auto drand = [](double min, double max){
        std::random_device rd;  //seed for the random number engine
        std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
        std::uniform_real_distribution<> distrib(min, max);

        return distrib(gen);
    };

    double fooNumber = drand(0,1000);

    uint32_t commSize = world.size();
    uint32_t rank     = world.rank();

    std::cout << "Process:" << rank << " of " << commSize << ", fooNumber:"<< fooNumber << '\n';

}
```

ejecutable

Compilador MPI

Compilador nativo

Librerías

Bibliotecas

Makefile

```
TARGET=./example

CXX=/usr/bin/mpicxx

DIR_OBJ=objs
SOURCES=$(wildcard *.cc)
OBJECTS=$(patsubst %.cc,$(DIR_OBJ)/%.o,$(SOURCES))

INTRINSICS=

BOOST_INCLUDE=-I/usr/include/
BOOST_LIB_DIR=-L/usr/lib/x86_64-linux-gnu
BOOST_LIBS=-lboost_mpi -lboost_serialization -lboost_system -lboost_filesystem

LDFLAGS =-L/usr/local/lib $(BOOST_LIB_DIR)
LDLIBS = -lm -lpthread $(BOOST_LIBS)

INCLUDES =-I. -I./include $(BOOST_INCLUDE)

CPPFLAGS =-std=c++17  -Wno-deprecated $(INTRINSICS)
```

Ejecución

A través del comando `mpirun`

```
gabriel@gserver:~/MPI/01-simple$ mpirun --oversubscribe -np 10 ./example
Process:8 of 10, fooNumber:56.0038
Process:5 of 10, fooNumber:936.764
Process:2 of 10, fooNumber:504.876
Process:3 of 10, fooNumber:785.594
Process:0 of 10, fooNumber:791.733
Process:1 of 10, fooNumber:792.361
Process:6 of 10, fooNumber:79.4813
Process:7 of 10, fooNumber:212.73
Process:4 of 10, fooNumber:829.615
Process:9 of 10, fooNumber:643.368
```

La opción --oversubscribe es necesario para que mpi ignore las restricciones de slots (LP) disponibles en el host.

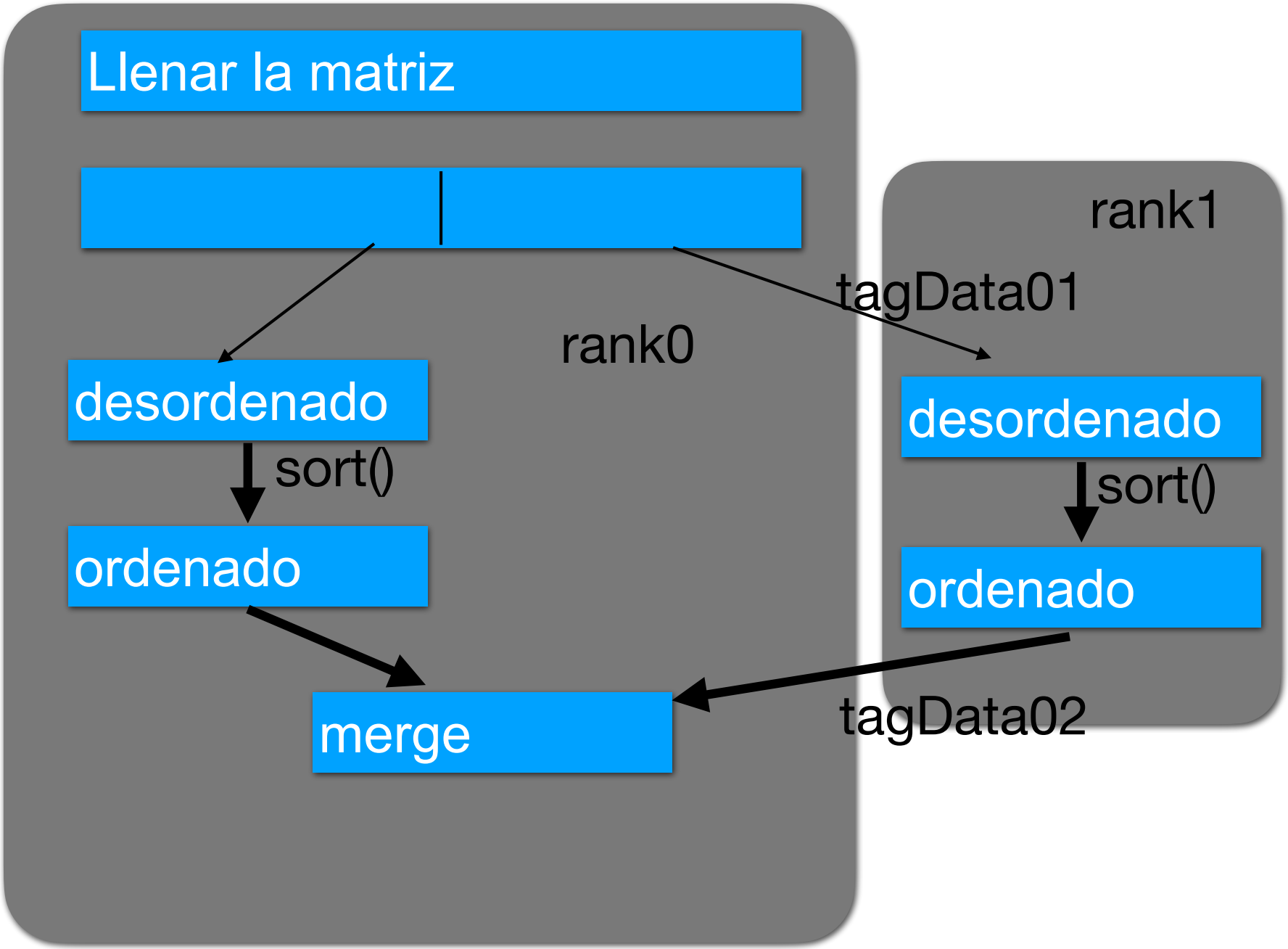
```
gabriel@gserver:~/UV/ICI517/05-MPI/01-simple$ mpirun --hostfile hostfile -np 10 ./example
Process:1 of 10, fooNumber:59.978
Process:2 of 10, fooNumber:927.48
Process:7 of 10, fooNumber:891.215
Process:8 of 10, fooNumber:582.31
Process:6 of 10, fooNumber:568.945
Process:5 of 10, fooNumber:194.421
Process:0 of 10, fooNumber:374.652
Process:3 of 10, fooNumber:710.596
Process:4 of 10, fooNumber:931.139
Process:9 of 10, fooNumber:328.079
```

O crear un archivo que especifique la cantidad de slots del nodo

hostfile

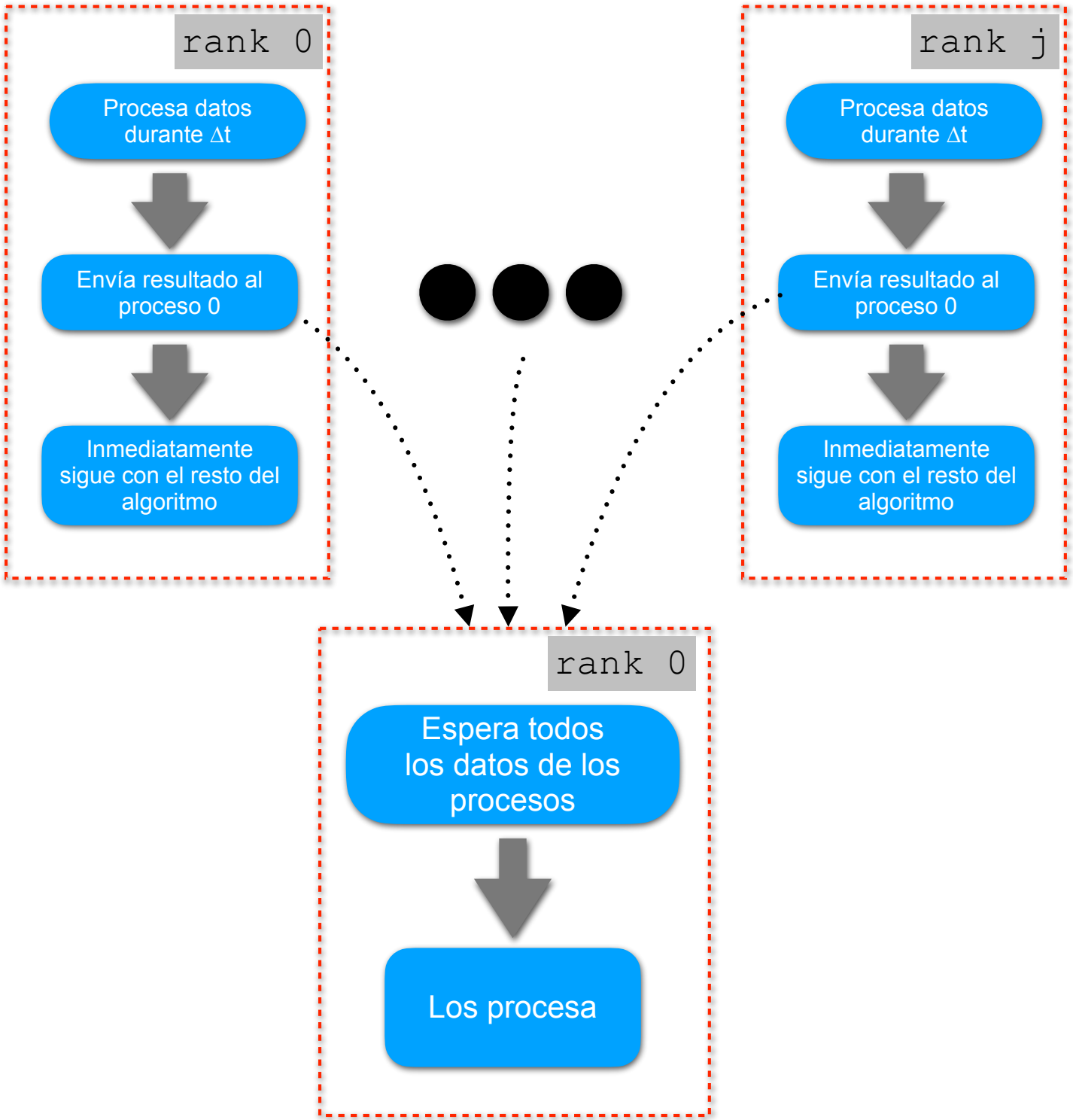
```
localhost slots=100
```

Comunicación Asincrónica entre procesos


Llenar la matriz
rank0
desordenado
sort()
ordenado
merge
rank1
tagData01
desordenado
sort()
ordenado
tagData02

Point to point

Comunicación Point2Point



Comunicación Point2Point


```cpp
//Cada proceso genera un número real
auto drand = [](double min, double max){
    std::random_device rd;  //seed for the random number engine
    std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
    std::uniform_real_distribution<> distrib(min, max);

    return distrib(gen);
};
//Cada proceso genera un número real aleatorio
double fooNumber = drand(0,1000);

//Cada proceso simula una carga de trabajo por timeProc segundos
int timeProc = drand(100,5000);
std::cout << T.curr<milli>() << " Process:" <<  world.rank() << " -> Time proc:"<< timeProc<< "ms\n";
std::this_thread::sleep_for(std::chrono::milliseconds(timeProc));
```

```cpp
//Una vez terminado el trabajo, envía el dato generado al proceso 0
std::cout << T.curr<milli>() << " Process:" <<  world.rank() << " -> send "<< fooNumber << '\n';
world.isend(0, tagData, fooNumber);
```

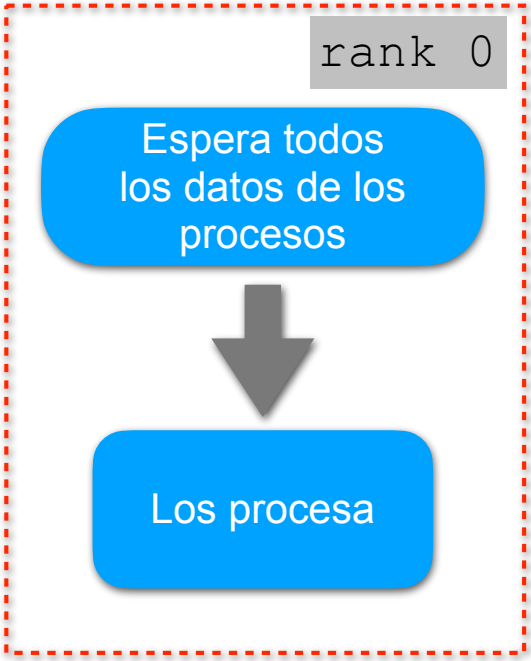
```cpp
std::cout << T.curr<milli>() << " Process:" <<  world.rank() << " -> endind" << '\n';
```

Costo Transmisión: O(S·p)

S: Tamaño del dato

p: # procesos

Comunicación Point2Point




```
if (world.rank() == 0) {
    mpi::request requests[world.size()];
    double numbers[world.size()];

    for(int rk = 0 ; rk < world.size(); rk++){
        requests[rk] = world.irecv(rk, tagData, numbers[rk]);
    }

    mpi::wait_all(requests, requests + world.size());

    for(int rk = 0 ; rk < world.size(); rk++){
        std::cout << T.curr<milli>() << " Process: " <<  world.rank() << " <- " << numbers[rk] << '\n';
    }

}
```

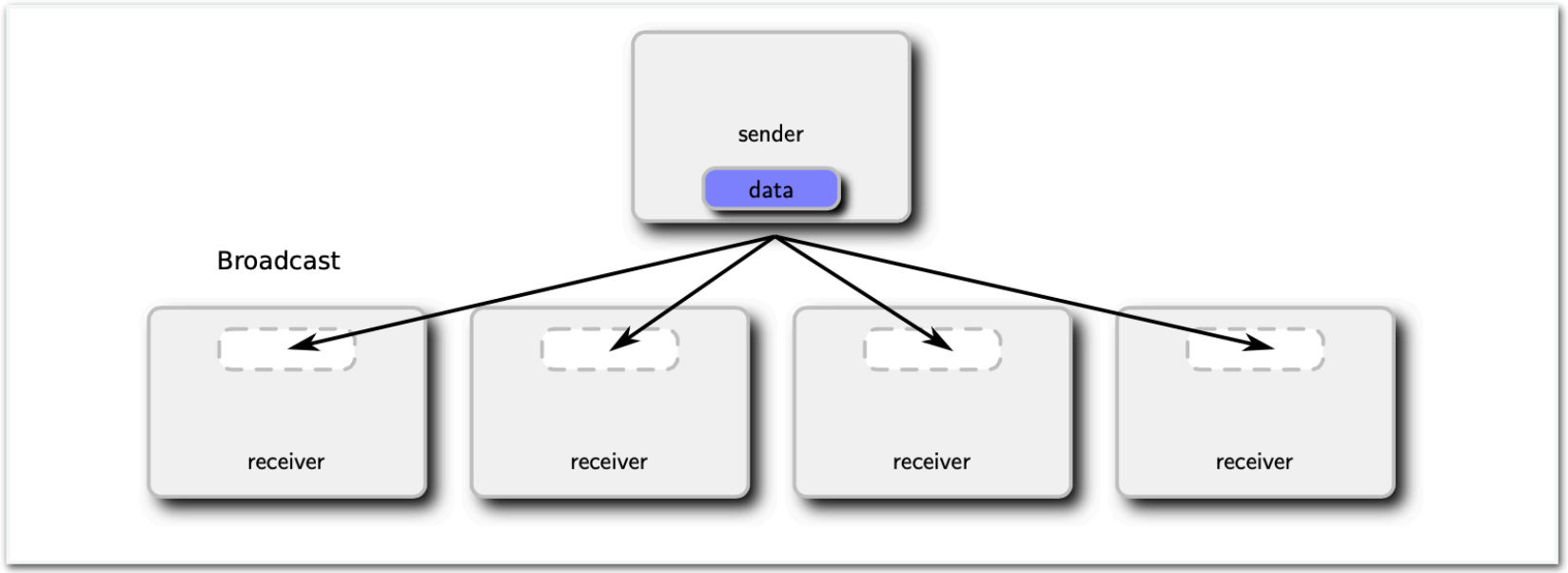
Más info

https://www.boost.org/doc/libs/1_76_0/doc/html/boost/mpi/communicator.html

Broadcast

Comunicación Broadcast

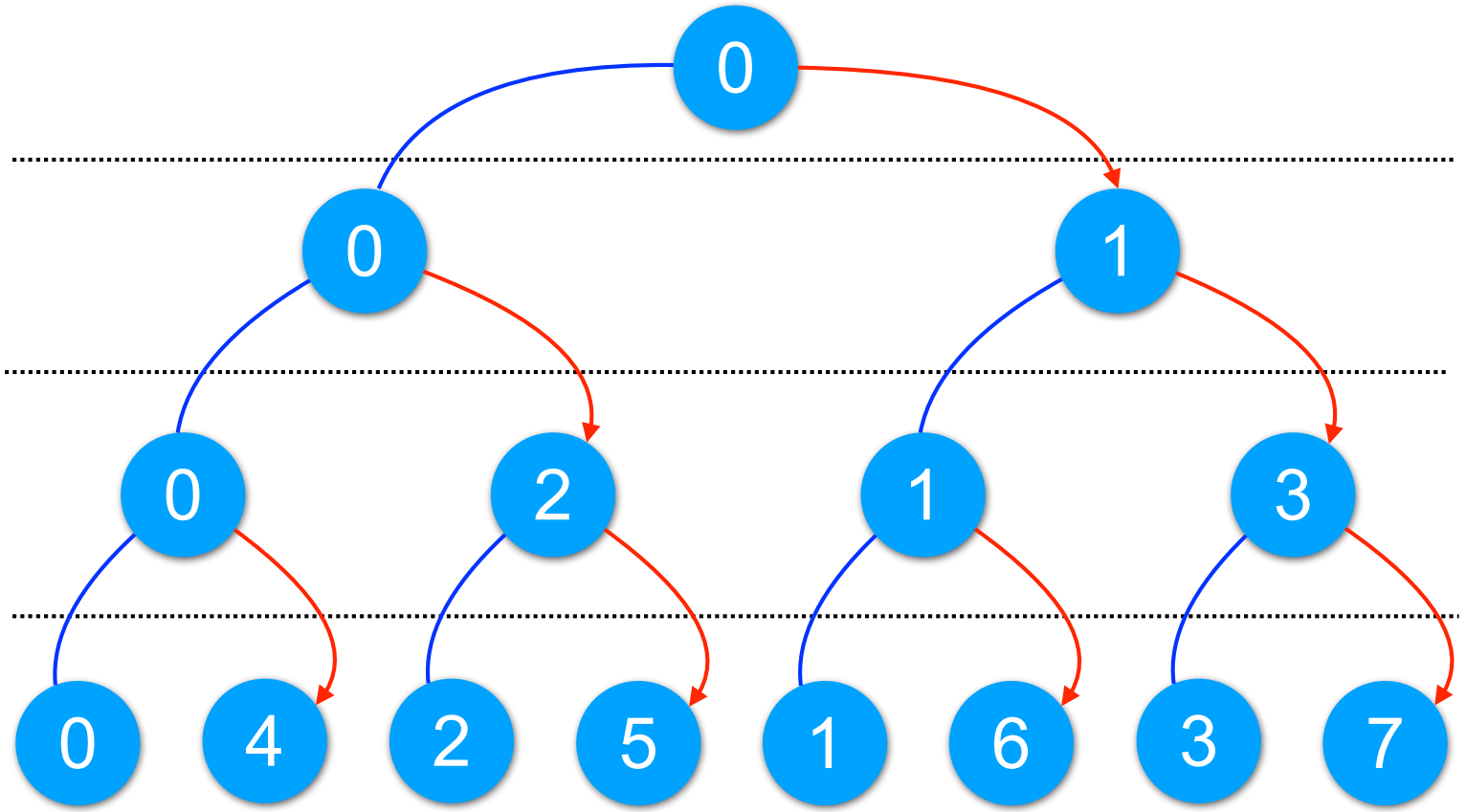
```
template<typename T>
  void broadcast(const communicator & comm, T & value, int root);
template<typename T>
  void broadcast(const communicator & comm, T * values, int n, int root);
template<typename T>
  void broadcast(const communicator & comm, skeleton_proxy< T > & value,
                 int root);
template<typename T>
  void broadcast(const communicator & comm, const skeleton_proxy< T > & value,
                 int root);
```



Costo Transmisión: O(S·log p)

Comunicación Broadcast

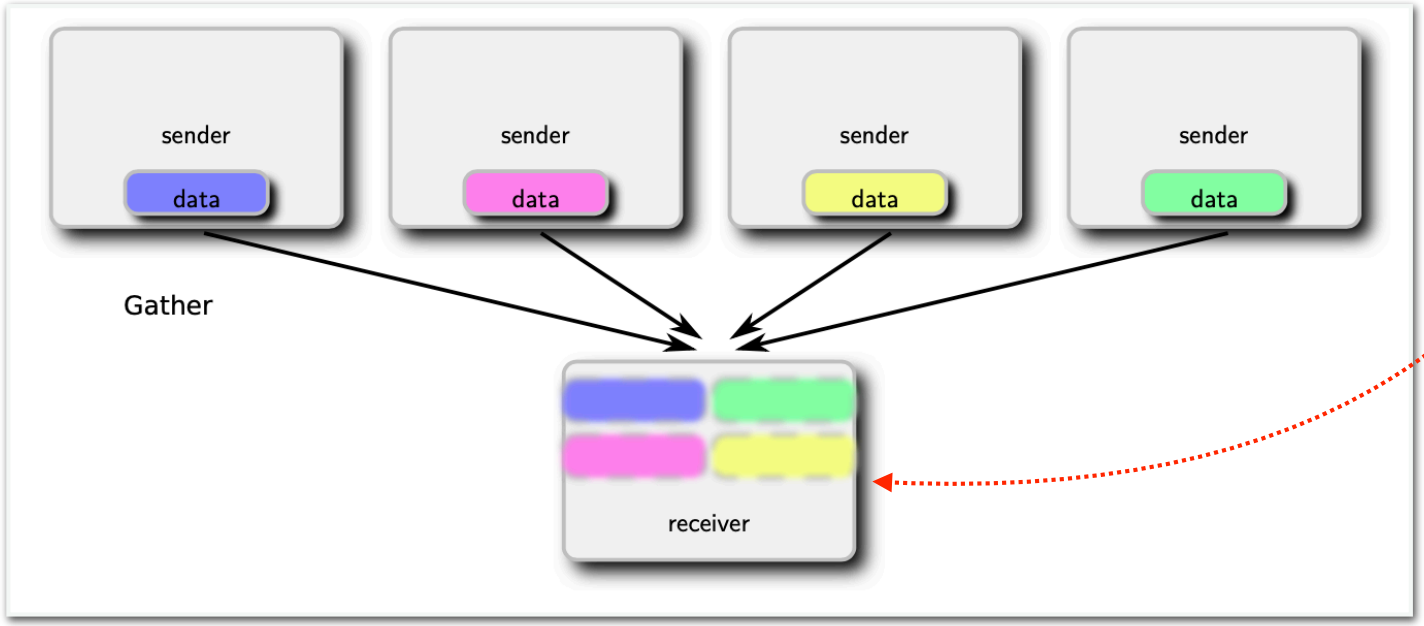
Costo Transmisión: O(S·log p)



Gather

Comunicación Gather

```
template<typename T>
  void gather(const communicator & comm, const T & in_value,
              std::vector< T > & out_values, int root);
template<typename T>
  void gather(const communicator & comm, const T & in_value, T * out_values,
              int root);
template<typename T>
  void gather(const communicator & comm, const T & in_value, int root);
template<typename T>
  void gather(const communicator & comm, const T * in_values, int n,
              std::vector< T > & out_values, int root);
template<typename T>
  void gather(const communicator & comm, const T * in_values, int n,
              T * out_values, int root);
template<typename T>
  void gather(const communicator & comm, const T * in_values, int n, int root);
```

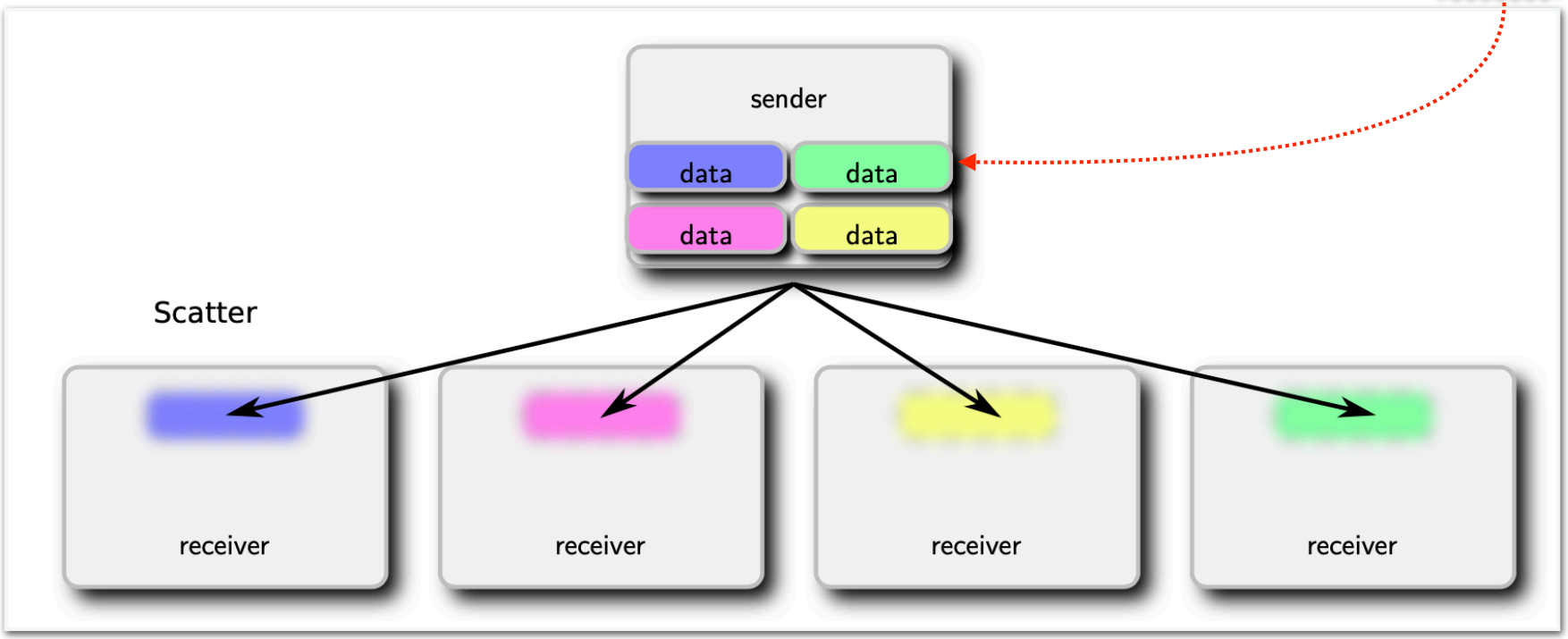


Costo Transmisión: O(S·P)

Scatter

Comunicación Scatter

```
template<typename T>
  void scatter(const communicator & comm, const std::vector< T > & in_values,
               T & out_value, int root);
template<typename T>
  void scatter(const communicator & comm, const T * in_values, T & out_value,
               int root);
template<typename T>
  void scatter(const communicator & comm, T & out_value, int root);
template<typename T>
  void scatter(const communicator & comm, const std::vector< T > & in_values,
               T * out_values, int n, int root);
template<typename T>
  void scatter(const communicator & comm, const T * in_values, T * out_values,
               int n, int root);
template<typename T>
  void scatter(const communicator & comm, T * out_values, int n, int root);
```

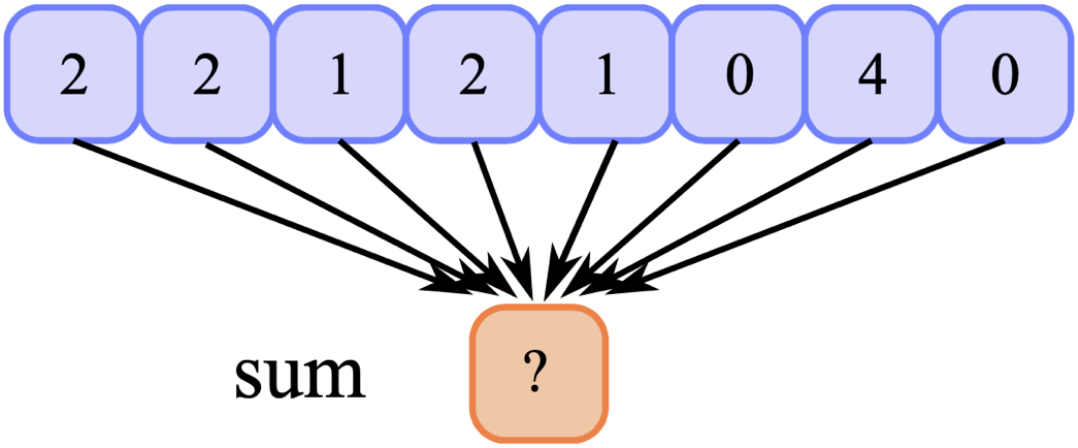


Costo Transmisión: O(max{S})

Reduction

MPI::REDUCE

```
template<typename T, typename Op>
  void reduce(const communicator & comm, const T & in_value, T & out_value,
              Op op, int root);
template<typename T, typename Op>
  void reduce(const communicator & comm, const T & in_value, Op op, int root);
template<typename T, typename Op>
  void reduce(const communicator & comm, const T * in_values, int n,
              T * out_values, Op op, int root);
template<typename T, typename Op>
  void reduce(const communicator & comm, const T * in_values, int n, Op op,
              int root);
```



Available reducers: max/min, minloc/maxloc, sum, product, AND, OR, XOR (logical or bitwise).