POLITECNICO
MILANO 1863

# Message Passing Interface - MPI

Alessandro Margara
alessandro.margara@polimi.it
https://margara.faculty.polimi.it

# Collective communication

# Collective communication

- Set of procedures that allow a group of processes to collaborate to achieve a common goal

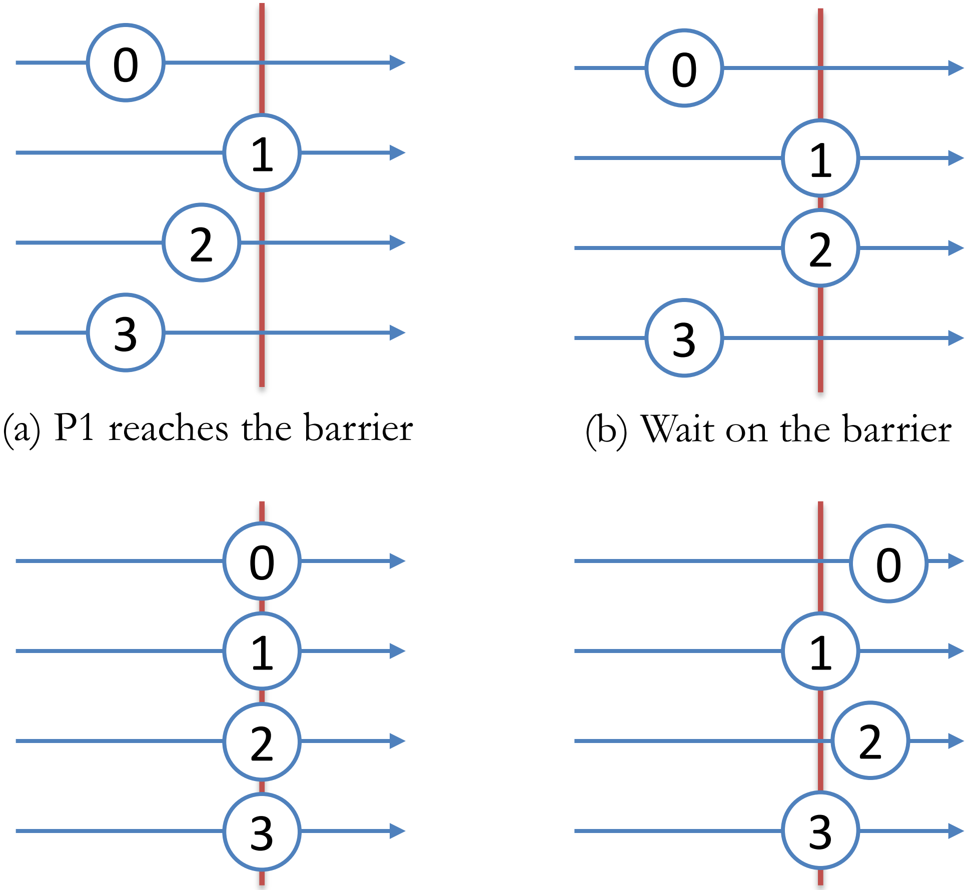
- Synchronization API
  - Barriers

- Communication API
  - Broadcast, reduce, gather, scatter, …

# Barrier

```
MPI_Barrier(
    MPI_Comm communicator)
```

- Creates a barrier for all processes
- No process is allowed to continue before all processes have reached the barrier



(a) P1 reaches the barrier

(b) Wait on the barrier

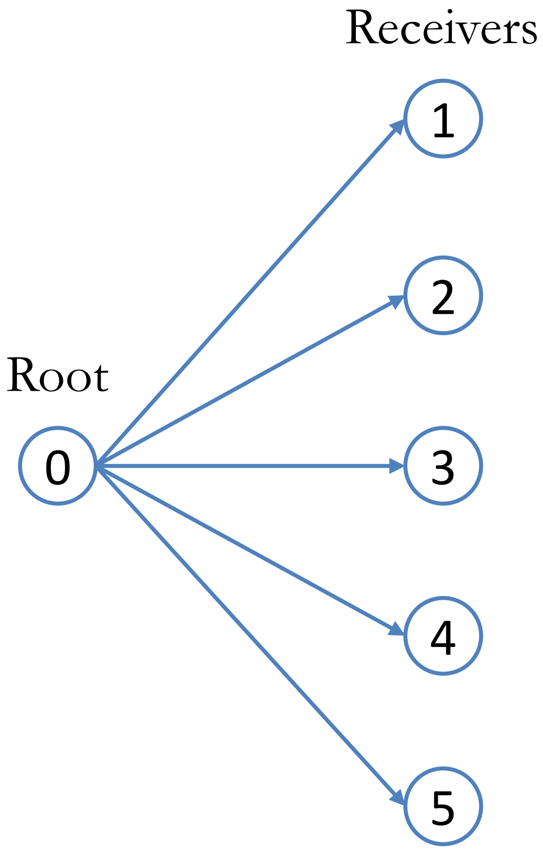
(c) Everyone reaches the barrier

(d) Processes can continue

# Broadcast

```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

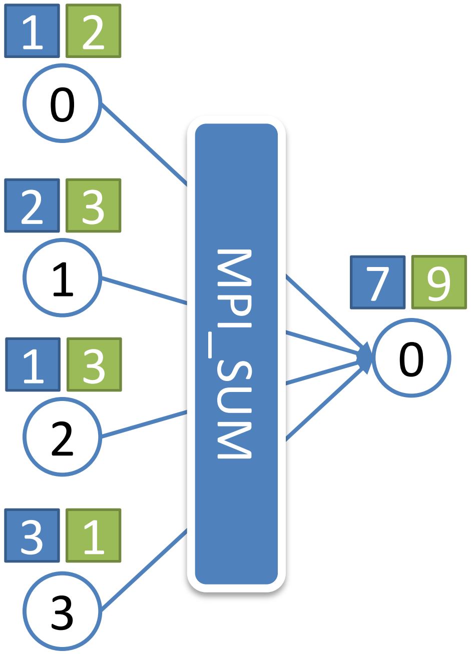
- Broadcast invoked both by a root process and receiver processes
  - Root process sends the `data` variable to receiver processes
  - Receiver processes fill the `data` variable with the data from the root process



# Reduce

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

- `send_data` variable is an array of elements that each process wants to reduce
  - count elements for each process
- `recv_data` variable in the root node contains the reduced data
  - count elements
- The AllReduce procedure stores the reduced data at every node



# Reduce operators

| | |
|---|---|
| MPI_MAX | Returns the maximum element |
| MPI_MIN | Returns the minimum element |
| MPI_SUM | Sums the elements |
| MPI_PROD | Multiplies the elements |
| MPI_LAND | Performs the logical *and* across the elements |
| MPI_LOR | Performs the logical *or* across the elements |
| MPI_BAND | Performs a bitwise *and* across the bits of the elements |
| MPI_BOR | Performs a bitwise *or* across the bits of the elements |
| MPI_MAXLOC | Returns the maximum value and the rank of the process that owns it |
| MPI_MINLOC | Returns the minimum value and the rank of the process that owns it |

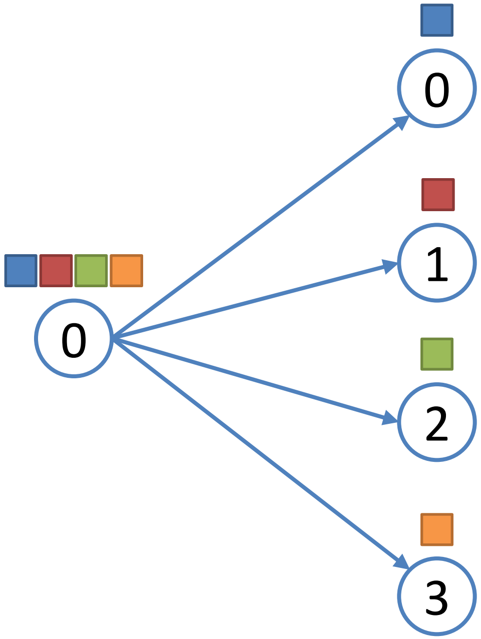
- It is also possible to add user-defined operators through the procedure `MPI_Op_create`

# Scatter

```
MPI_Scatter(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

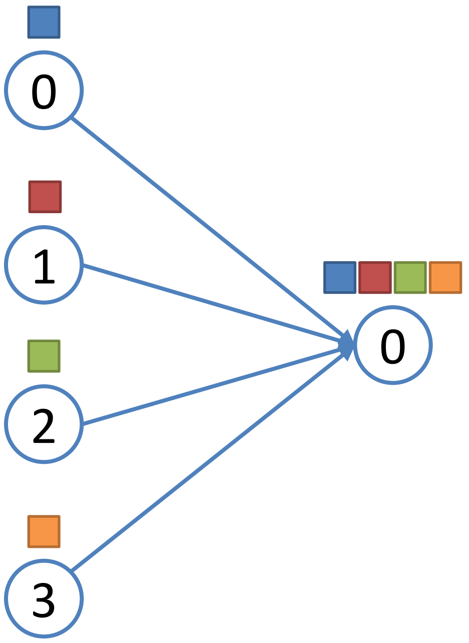
- Root process sends the `send_data` variable to receiver processes
  - `send_count` elements for each process

- Receiver processes fill the `recv_data` variable with the data from the root process



# Gather

```
MPI_Gather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```



- Processes send the `send_data` variable to root process

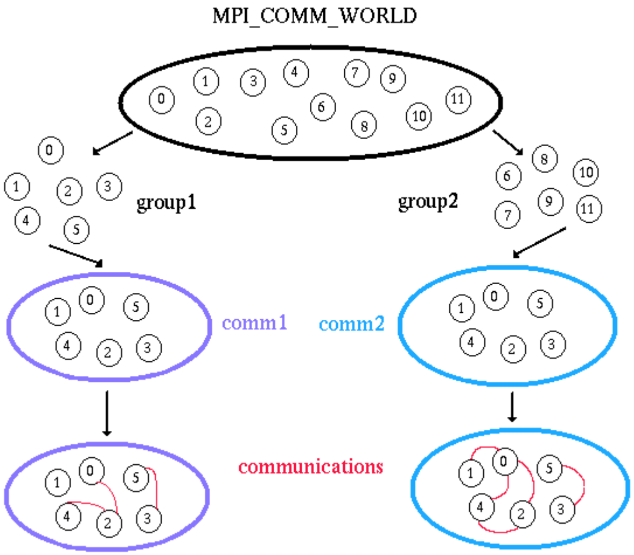
- Root process fills the `recv_data` variable with the data from the receiver processes

# Groups and communicators

- Communicators define the scope of a communication
  - We have only considered `MPI_COMM_WORLD` in this lecture

- MPI allows creating new communicators starting from an existing communicator and a set of processes, using

```
MPI_Comm_create
MPI_Comm_group
MPI_Comm_split
```



# Exercise 5

- Write a program that computes the average value over a large set of numbers
  - Process P0 scatters the array
  - Each process computes the average on its part of the array
  - Process P0 gathers partial results and computes the final value

# Exercise 6

- Write a program that filters large arrays of numbers
  - Process P0 broadcasts a number N
  - Each process creates a random array of numbers
  - Each process filters the input list by retaining only numbers that are multiples of N
  - Process P0 receives the results from all other processes, stores them in an array, and prints them