

The background features large, stylized, light blue outlines of the letters 'MIMD'. The 'M' is at the top right, the 'I' is in the middle right, the 'D' is at the bottom right, and the 'A' is at the bottom left. The text is centered within a yellow rounded rectangle with a red border.

Alcuni strumenti per  
lo sviluppo di software  
su architetture MIMD

- Architetture SM (Shared Memory)

A diagram illustrating the mapping of OpenMP to Shared Memory architectures. A red arrow points from the text 'Architetture SM (Shared Memory)' to a yellow box with a red border containing the text 'OpenMP'. From the bottom of the 'OpenMP' box, several lines radiate downwards, suggesting a connection to the Distributed Memory architectures listed below.

OpenMP

- Architetture DM (Distributed Memory)

A diagram illustrating the mapping of MPI to Distributed Memory architectures. A red arrow points from the text 'Architetture DM (Distributed Memory)' to a yellow box with a red border containing the text 'MPI'. From the top of the 'MPI' box, several lines radiate upwards, suggesting a connection to the Shared Memory architectures listed above.

MPI

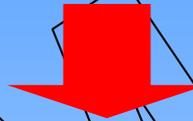
# Message Passing Interface MPI

MPI è una libreria  
che comprende:

- Funzioni per definire l'*ambiente*
- Funzioni per *comunicazioni uno a uno*
- Funzioni per *comunicazioni collettive*
- Funzioni per *operazioni collettive*

# La comunicazione di un messaggio

La comunicazione di un messaggio può coinvolgere due o più processori.



Per comunicazioni che coinvolgono solo **due** processori



Si considerano funzioni MPI per **comunicazioni uno a uno**

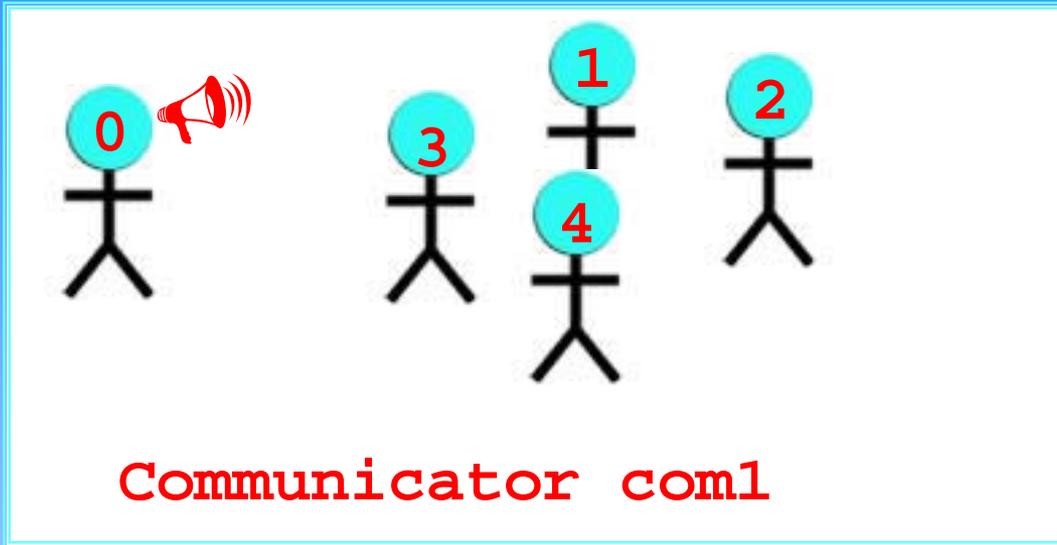
Per comunicazioni che coinvolgono **più** processori



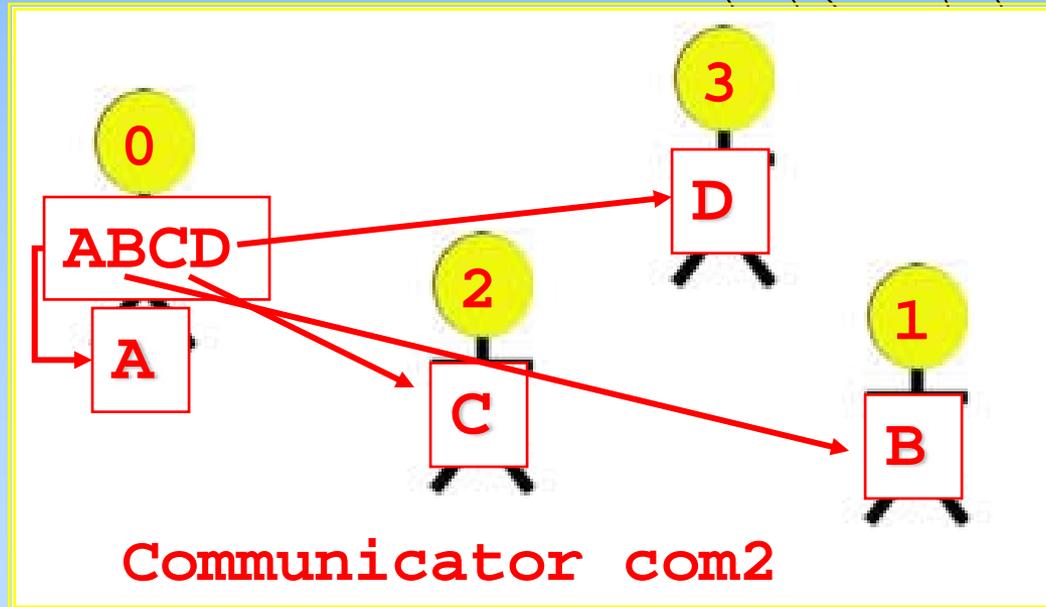
Si considerano funzioni MPI per **comunicazioni collettive**

## Comunicazione collettive di un messaggio

# Comunicazioni collettive :



Nel communicator **com1**  $P_0$  comunica con tutti gli altri processori



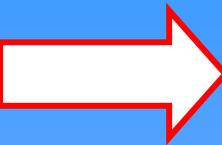
$P_0$  distribuisce a tutti i processori di **com2** un elemento del proprio vettore

# Spedizione collettiva, uno a molti:

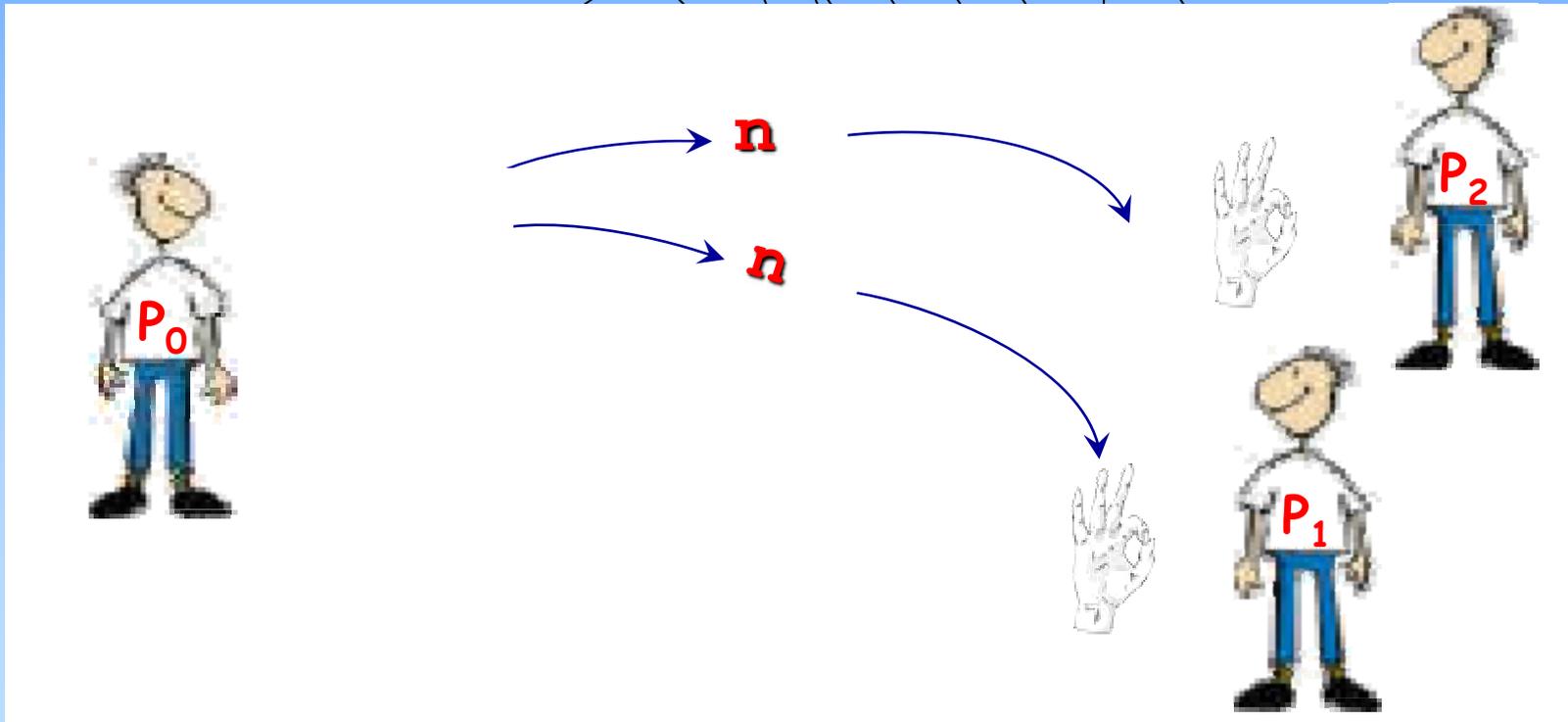
```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int menum, nproc;
    int n, tag, num;
    MPI_Status info;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&menum);
    if(menum==0)
    {
        scanf("%d",&n);
        MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

# Nel programma ... :

 `MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);`

- Il processore  $P_0$  spedisce  $n$ , di tipo `MPI_INT` e di dimensione `1`, a tutti i processori dell'ambiente `MPI_COMM_WORLD`.



# In generale ... :

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

- Il processore con identificativo **root** spedisce a tutti i processori del comunicator **comm** lo stesso dato memorizzato in **\*buffer**.
- **Count, datatype, comm** devono essere uguali per ogni processore di **comm**.

# In dettaglio...:

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

**\*buffer** indirizzo del dato da spedire

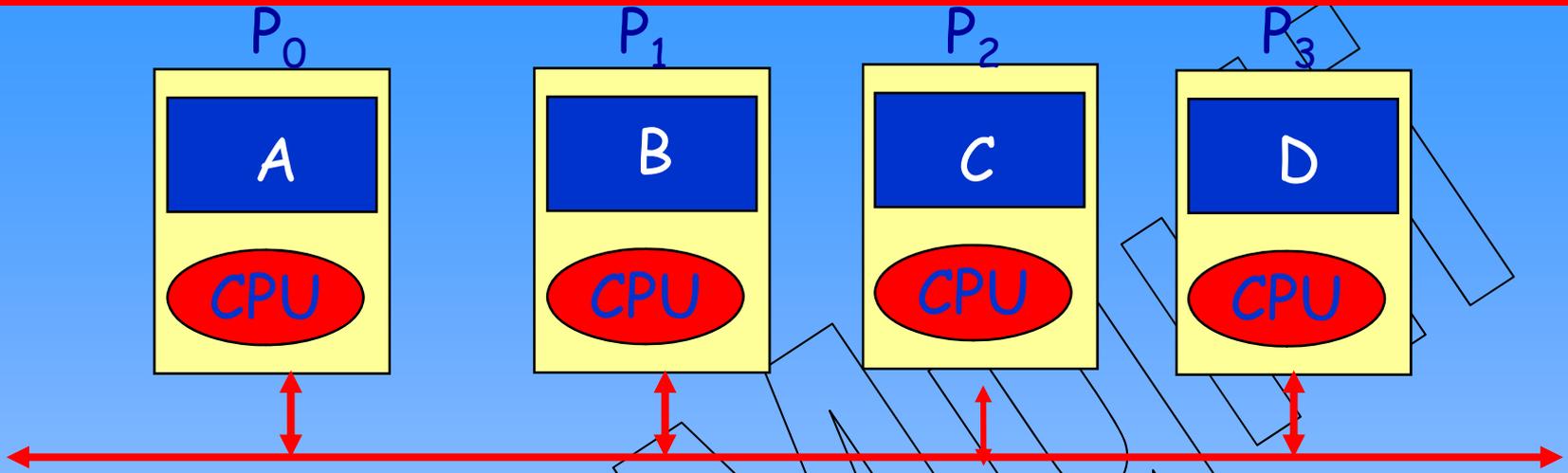
**count** numero dei dati da spedire

**datatype** tipo dei dati da spedire

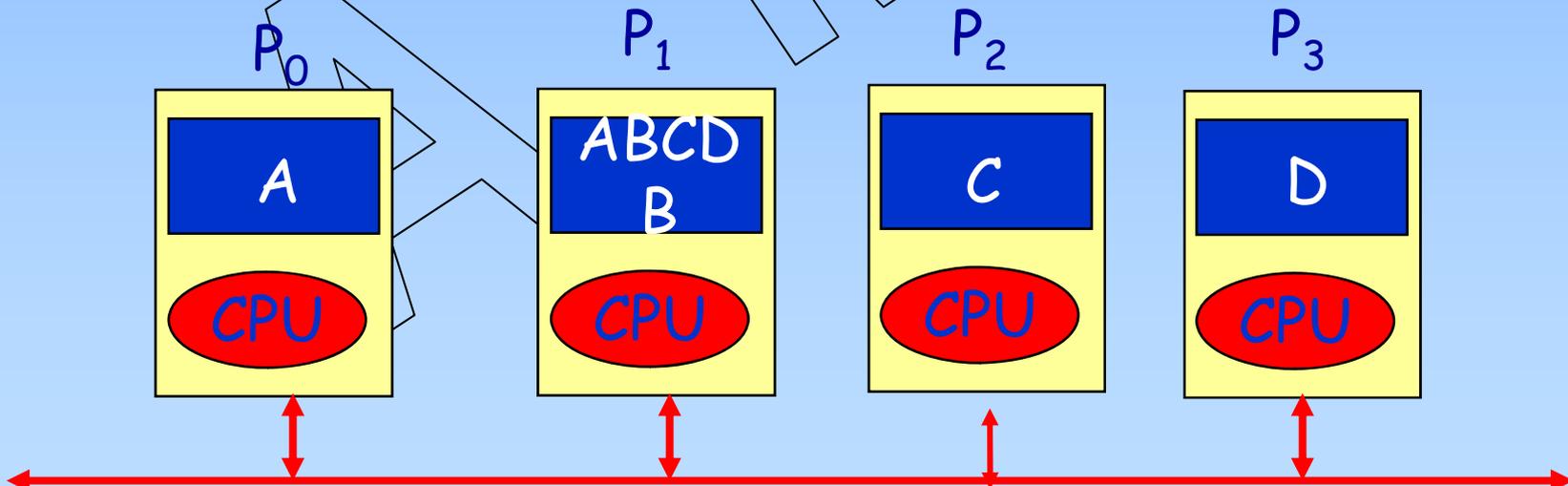
**root** identificativo del processore che spedisce a tutti

**comm** identificativo del communicator

# Spedizione collettiva di tipo: *data gather*



Tutti i processori spediscono i propri dati ad un processore assegnato (es al processore  $P_1$ )



# Il gather in MPI

```
MPI_Gather(void *send_buff, int send_count,  
           MPI_Datatype datatype,  
           void *recv_buff, int recv_count,  
           MPI_Datatype recv_type,  
           int root, MPI_Comm comm);
```

- Ogni processore di **comm** spedisce il contenuto di **\*send\_buff** al processore con identificativo **root**
- Il processore con identificativo **root** *concatena* i dati ricevuti in **recv\_buff**, memorizzando prima i dati ricevuti dal processore 0, poi i dati ricevuti dal processore 1, poi quelli ricevuti dal processore 2, etc...

# Il gather in MPI: in dettaglio...

```
MPI_Gather(void *send_buff, int send_count,  
MPI_Datatype sendtype,  
void *recv_buff, int recv_count,  
MPI_Datatype recv_type,  
int root, MPI_Comm comm);
```

**\*send\_buff** indirizzo del dato da spedire

**send\_count** numero dei dati da spedire

**sendtype** tipo dei dati da spedire

**\*recv\_buff** indirizzo del dato in cui **root** riceve

**recv\_count** numero dei dati che root riceve

**recv\_type** tipo dei dati che root riceve

**root** identificativo del processore che riceve da tutti

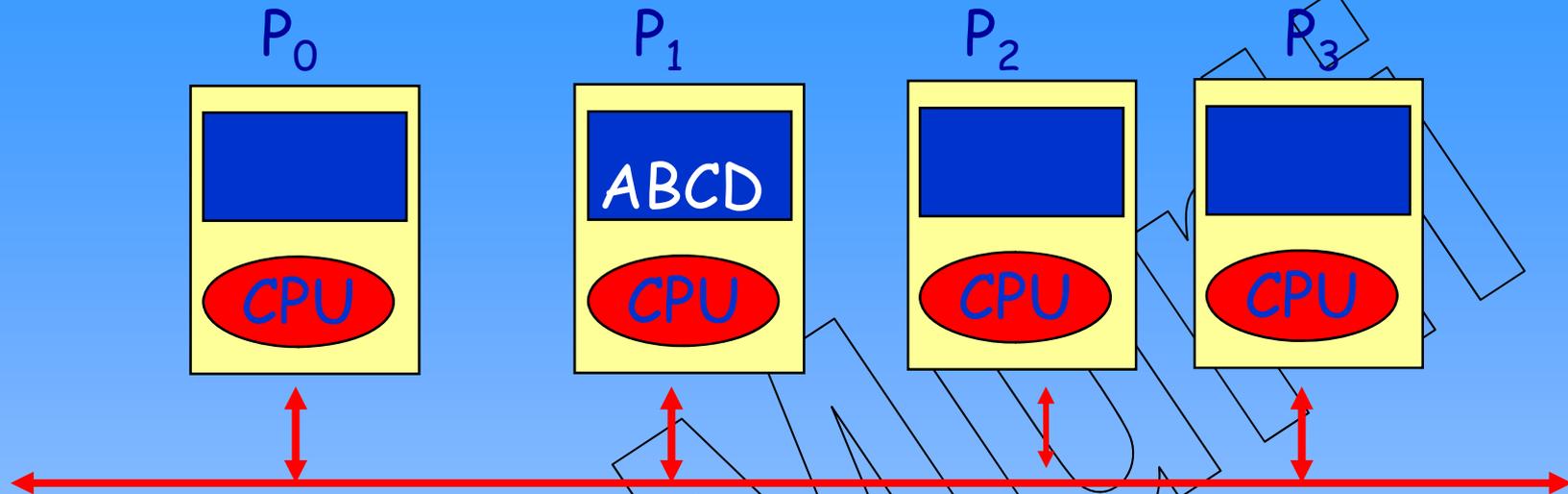
**comm** identificativo del communicator

# Il gather in MPI

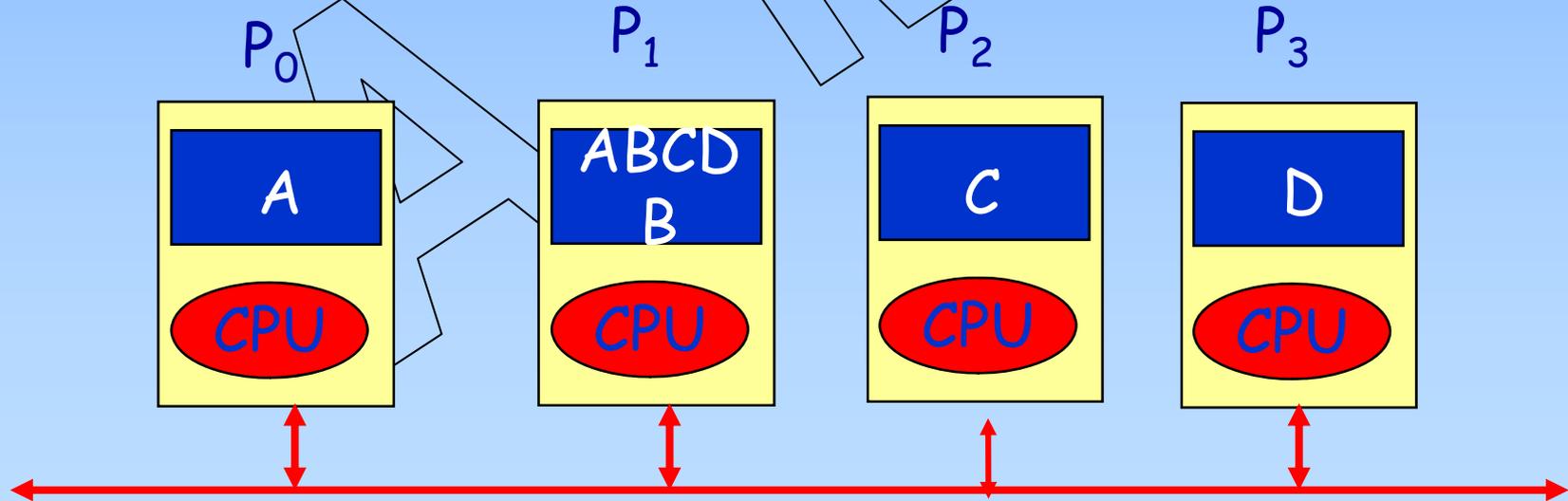
```
MPI_Gather(void *send_buff, int send_count,  
           MPI_Datatype datatype,  
           void *recv_buff, int recv_count,  
           MPI_Datatype recv_type,  
           int root, MPI_Comm comm);
```

- Gli argomenti **recv\_** sono significativi solo per il processore **root**
- L'argomento **recv\_count** è il numero di dati da ricevere da ogni processore, non è il numero totale dei dati da ricevere.

# Spedizione collettiva di tipo: *data scatter*



Un solo processore distribuisce i propri dati agli altri processori, se stesso compreso.



# Lo scatter in MPI

```
MPI_Scatter(void *send_buff, int send_count,  
MPI_Datatype send_type,  
void *recv_buff, int recv_count,  
MPI_Datatype recv_type,  
int root, MPI_Comm comm);
```

- Il processore con identificativo **root** *distribuisce* i dati contenuti in **send\_buff**.
- Il contenuto di **send\_buff** viene suddiviso in **nproc** segmenti ciascuno di lunghezza **send\_count**
- Il primo segmento viene affidato al processore con identificativo 0, il secondo al processore con identificativo 1, il terzo al processore con identificativo 2, etc.

# Lo scatter in MPI: in dettaglio

```
MPI_Scatter(void *send_buff, int send_count,  
            MPI_Datatype send_type,  
            void *recv_buff, int recv_count,  
            MPI_Datatype recv_type,  
            int root, MPI_Comm comm);
```

**\*send\_buff** indirizzo del dato da spedire

**send\_count** numero dei dati da spedire

**send\_type** tipo dei dati da spedire

**\*recv\_buff** indirizzo del dato in cui ricevere

**recv\_count** numero dei dati da ricevere

**recv\_type** tipo dei dati da ricevere

**root** identificativo del processore che spedisce a tutti

**comm** identificativo del communicator

## Le operazioni collettive.

# Caratteristiche delle operazioni collettive

Le operazioni collettive sono eseguite da **tutti** i processori appartenenti ad un communicator.

Inoltre...

- Tutti i processori che eseguono l'operazione collettiva eseguono almeno **una** comunicazione.
- L'operazione collettiva può richiedere una **sincronizzazione**.
- Tutte le operazioni collettive sono **bloccanti**.

# Scopo delle operazioni collettive

Le operazioni collettive permettono:

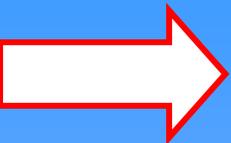
- La **Sincronizzazione** dei processori.
- L'esecuzione di **operazioni globali** (es. ricerca del massimo in un vettore distribuito fra i processori).
- Gestione **ottimizzata** degli input/output seguendo uno schema ad albero.

# Sincronizzazione dei processori ... :

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    ...
    MPI_Init(&argc, &argv);
    ...
    MPI_Barrier(MPI_COMM_WORLD);
    ...
    MPI_Finalize();
}
```

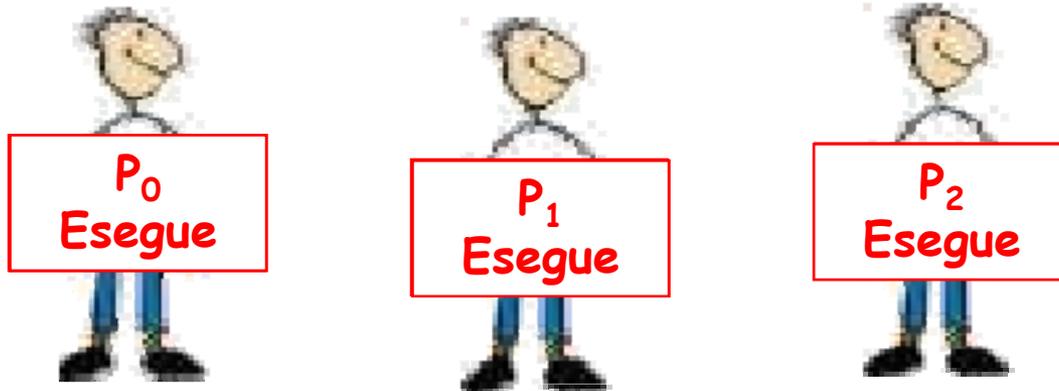


# Nel programma ... :



## `MPI_Barrier(MPI_COMM_WORLD);`

- Ogni processore dell'ambiente `MPI_COMM_WORLD` può procedere solo quando tutti gli altri avranno richiamato questa routine.



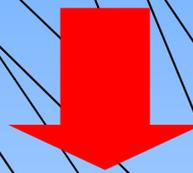
# In generale ... :

```
MPI_Barrier(MPI_Comm comm);
```

- Questa routine fornisce un meccanismo sincronizzante per **tutti** i processori del communicator **comm**.
- Ogni processore si *ferma* fin quando tutti i processori di **comm** non eseguono **MPI\_Barrier**.

# Operazioni di *riduzione*

MPI possiede una classe di operazioni collettive chiamate *operazioni di riduzione*.

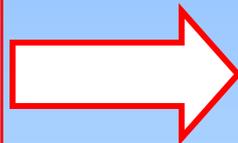


In ciascuna operazione di riduzione *tutti* i processori di un communicator *contribuiscono* al risultato di un'operazione.

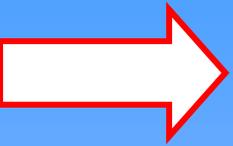
# Un semplice programma :

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int menum, nproc;
    int sum, sumtot;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&menum);
    MPI_Comm_size(MPI_COMM_WORLD,&nproc);
    sum=nproc;
    sumtot=0;
    sum+=menum;
    MPI_Reduce(&sum,&sumtot,1,MPI_INT,MPI_SUM,0,
              MPI_COMM_WORLD);
    printf("Sono %d sum=%d sumtot=%d\n",menum,sum,sumtot);

    MPI_Finalize();
    return 0;
}
```



# Nel programma ... :



```
MPI_Reduce(&sum, &sumtot, 1, MPI_INT, MPI_SUM,  
0, MPI_COMM_WORLD);
```

- Il processore  $P_0$  otterrà la somma dei elementi **sum**, di tipo **MPI\_INT** e di dimensione **1**, distribuiti tra tutti i processori del communicator **MPI\_COMM\_WORLD**. Il risultato lo pone nella propria variabile **sumtot**.

# Operazioni di Reduce ... :

Le operazioni globali di **Reduce** sono implementate in maniera efficiente in quanto:

- 1) Ottimizzano le comunicazioni tra i processori del communicator coinvolto. Le comunicazioni vengono eseguite seguendo uno schema ad albero.
- 2) Sfruttano la proprietà associativa e/o la proprietà commutativa.

# In Generale :

```
MPI_Reduce(void *operand, void *result,  
            int count, MPI_Datatype datatype,  
            MPI_Op op, int root,  
            MPI_Comm comm);
```

- Tutti i processori di **comm** combinano i propri dati memorizzati in **\*operand** utilizzando l'operazione **op**.
- Il risultato viene memorizzato in **\*result** di proprietà del processore con identificativo **root**
- **Count**, **datatype**, **comm** devono essere uguali per ogni processore di **comm**.

## In dettaglio... :

```
MPI_Reduce(void *operand, void *result,  
            int count, MPI_Datatype datatype,  
            MPI_Op op, int root,  
            MPI_Comm comm);
```

**\*operand** indirizzo dei dati su cui effettuare l'operazione.

**\*result** indirizzo del dato contenente il risultato.

**count** numero dei dati su cui effettuare l'operazione.

**datatype** tipo degli elementi da spedire.

**op** operazione effettuata.

**root** identificativo del processore che conterrà il risultato

**comm** identificativo del communicator

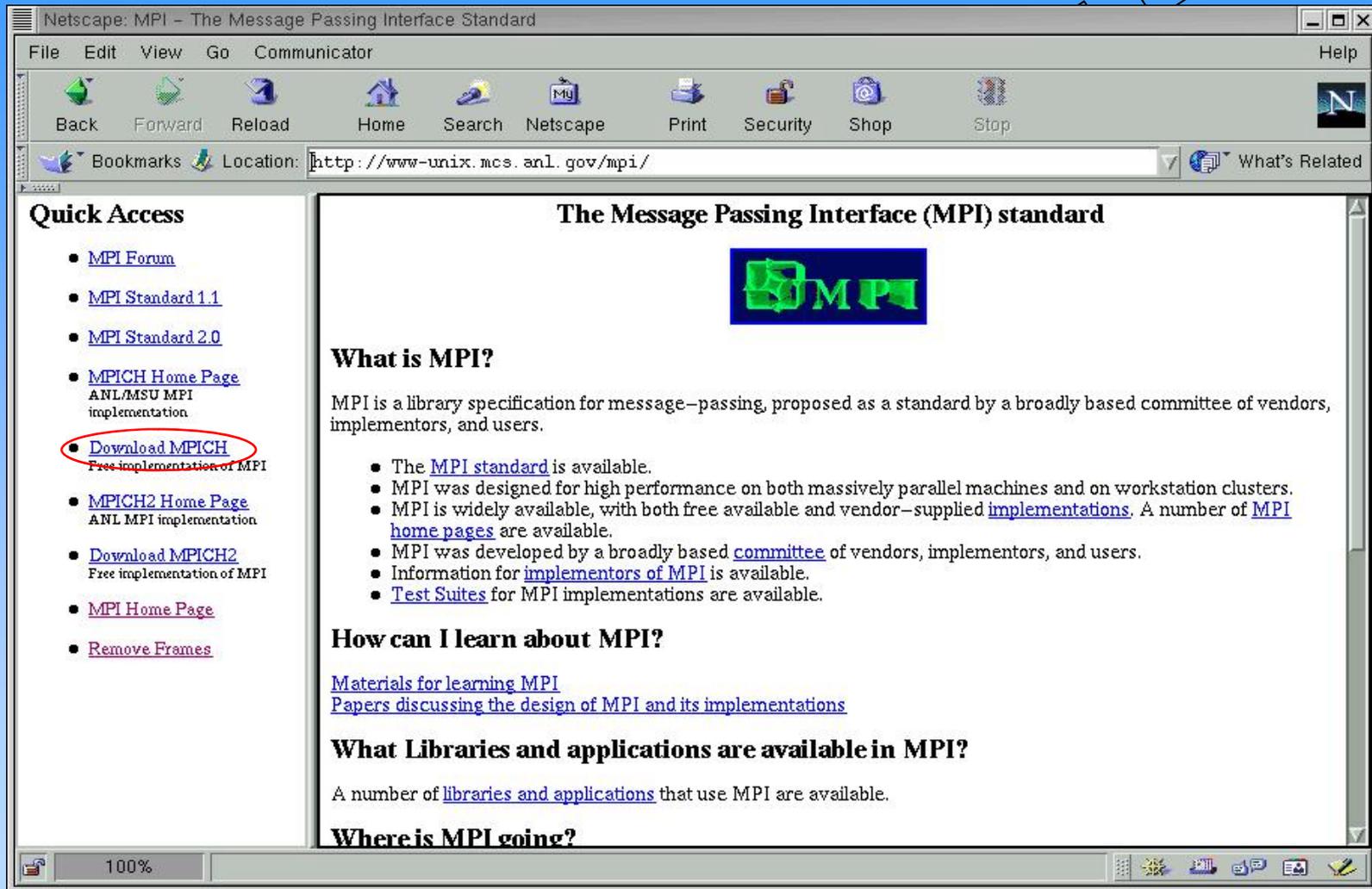
# Operazione collettiva: *Reduce*

L'argomento **op**, che descrive l'operazione da eseguire sugli operandi **operand**, distribuiti fra i processori può essere scelto fra i seguenti valori predefiniti:

- MPI\_MAX** → Massimo di un vettore distribuito
- MPI\_MIN** → Minimo di un vettore distribuito
- MPI\_SUM** → Somma componente per componente fra vettori distribuiti
- MPI\_PROD** → Prodotto componente per componente fra vettori distribuiti

.....

# Dove recuperare MPI



The screenshot shows a Netscape browser window titled "Netscape: MPI - The Message Passing Interface Standard". The address bar contains the URL "http://www-unix.mcs.anl.gov/mpi/". The main content area displays the title "The Message Passing Interface (MPI) standard" and a logo for MPI. Below the logo, there is a section titled "What is MPI?" which explains that MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users. A list of bullet points follows, detailing the availability of the MPI standard, its design for high performance on massively parallel machines and workstation clusters, its wide availability with both free and vendor-supplied implementations, its development by a broadly based committee, and the availability of information for implementors and test suites. Below this, there is a section titled "How can I learn about MPI?" with links to materials for learning MPI and papers discussing the design of MPI and its implementations. Another section titled "What Libraries and applications are available in MPI?" mentions that a number of libraries and applications use MPI. Finally, there is a section titled "Where is MPI going?". On the left side of the browser window, there is a "Quick Access" sidebar with several links, including "MPI Forum", "MPI Standard 1.1", "MPI Standard 2.0", "MPICH Home Page" (ANL/MSU MPI implementation), "Download MPICH" (Free implementation of MPI), "MPICH2 Home Page" (ANL MPI implementation), "Download MPICH2" (Free implementation of MPI), "MPI Home Page", and "Remove Frames". The "Download MPICH" link is circled in red.

**Quick Access**

- [MPI Forum](#)
- [MPI Standard 1.1](#)
- [MPI Standard 2.0](#)
- [MPICH Home Page](#)  
ANL/MSU MPI implementation
- [Download MPICH](#)  
Free implementation of MPI
- [MPICH2 Home Page](#)  
ANL MPI implementation
- [Download MPICH2](#)  
Free implementation of MPI
- [MPI Home Page](#)
- [Remove Frames](#)

## The Message Passing Interface (MPI) standard



### What is MPI?

MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users.

- The [MPI standard](#) is available.
- MPI was designed for high performance on both massively parallel machines and on workstation clusters.
- MPI is widely available, with both free available and vendor-supplied [implementations](#). A number of [MPI home pages](#) are available.
- MPI was developed by a broadly based [committee](#) of vendors, implementors, and users.
- Information for [implementors of MPI](#) is available.
- [Test Suites](#) for MPI implementations are available.

### How can I learn about MPI?

[Materials for learning MPI](#)  
[Papers discussing the design of MPI and its implementations](#)

### What Libraries and applications are available in MPI?

A number of [libraries and applications](#) that use MPI are available.

### Where is MPI going?

[www-unix.mcs.anl.gov/mpi/](http://www-unix.mcs.anl.gov/mpi/)

# Dove recuperare MPI

Netscape: MPI - The Message Passing Interface Standard

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://www-unix.mcs.anl.gov/mpi/> What's Related

## Quick Access

- [MPI Forum](#)
- [MPI Standard 1.1](#)
- [MPI Standard 2.0](#)
- [MPICH Home Page](#)  
ANL/MSU MPI implementation
- [Download MPICH](#)  
Free implementation of MPI
- [MPICH2 Home Page](#)  
ANL MPI implementation
- [Download MPICH2](#)  
Free implementation of MPI
- [MPI Home Page](#)
- [Remove Frames](#)

## Getting the MPICH implementation

The [README](#) is available for this implementation. The current release (1.2.5) can be obtained by anonymous ftp from ftp.mcs.anl.gov in the directory [pub/mpi](#). [Changes from the previous version of MPICH](#) are available.

Description	File	Size
Unix (all flavors)	<a href="#">mpich.tar.gz</a>	12.3 MB
Windows NT/2000	<a href="#">See Web Page</a>	4.3 MB
Performance tests	<a href="#">perftest.tar.gz</a>	0.1 MB

The gunzip utility is needed to uncompress the Unix versions. This is available from the [GNU Gzip](#) page. For those with poor network connections, the gzipped tar file is available in several pieces in [pub/mpi/mpisplit](#). This directory contains compressed files for a split version of the mpich.tar file.

This is a source-distribution; to build the MPI libraries you will need to execute at least

```
configure
make
```

See the installation and users manual for more information.

### Having trouble with ftp?

Some ftp clients and firewalls have a limit on the length of the "welcome" message that they can handle (1024 characters is common). If you are having trouble, check to see if there is such a limit, and if so, have it raised to 2048. Unfortunately, our government-mandated "welcome" exceeds 1024 characters.

**NEW** Suggestions on [compiler switches](#) is available.

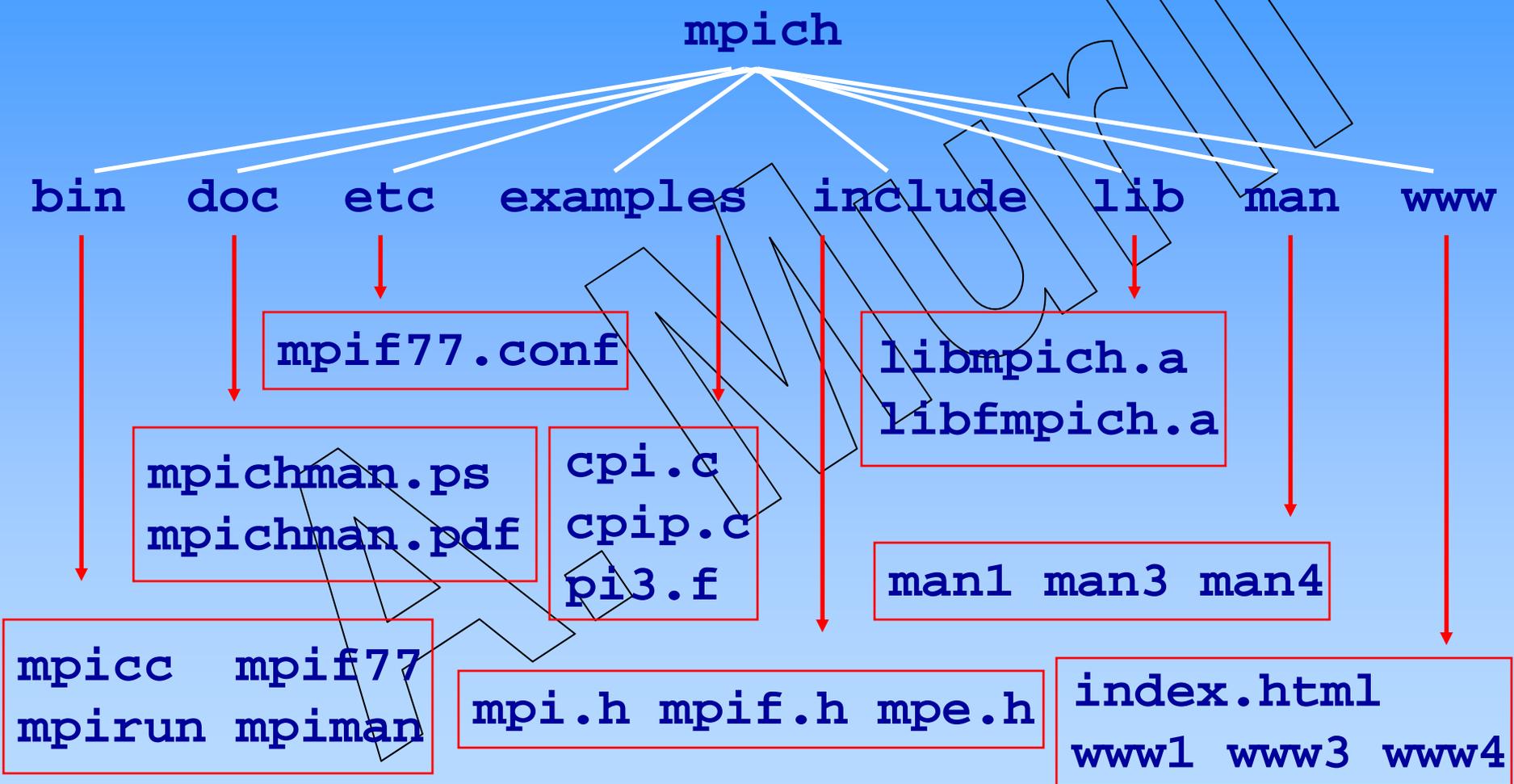
A [list of known bugs and patchfiles](#) for fixes is available (this is under construction and is not yet complete).

The Installation and Users manuals are available in hypertext form and by FTP. Separate manuals for each device are provided:

Device	Format
--------	--------

[www-unix.mcs.anl.gov/mpi/](http://www-unix.mcs.anl.gov/mpi/)

# Esplorazione directory MPI



Alcuni programmi senza linker: `bin/mpicc`, `bin/mpirun`, `bin/mpiman`, `bin/mpif77`