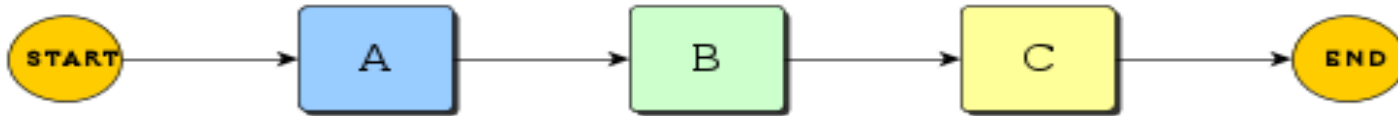




# Programación en Entornos Paralelos: MPI

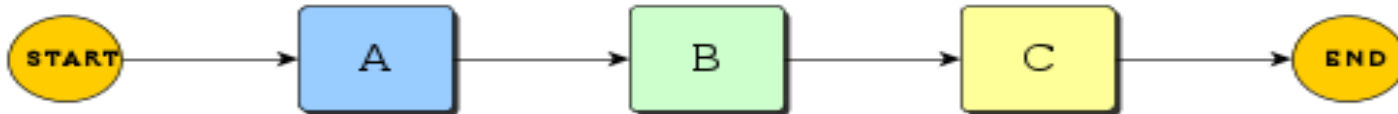
Graciela Molina  
m.graciela.molina@gmail.com

# TRADICIONALMENTE



Procesamiento secuencial

# TRADICIONALMENTE



Procesamiento secuencial

Si ya utilicé técnicas de optimización y aún necesito **mejorar la performance de mi código** ?

Y si agrego un core ... Cuanto mejora?

# ACTUALMENTE

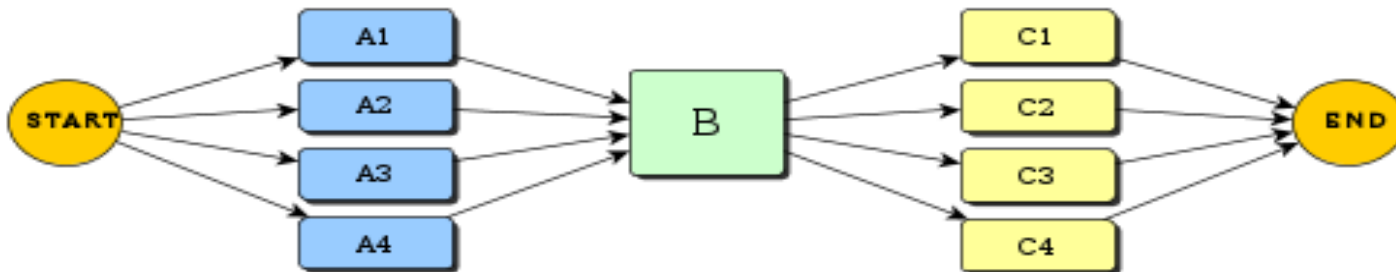
Cuento con hardware multi-core

Mi problema se puede subdividir en problemas independientes o es necesario ejecutar un gran número de veces una misma simulación

# ACTUALMENTE

Cuento con hardware multi-core

Mi problema se puede subdividir en problemas independientes o es necesario ejecutar un gran número de veces una misma simulación



Procesamiento paralelo

# Surge la Computación de Alto Rendimiento

No hay una sola definición sino que depende de la perspectiva.

**HPC es cuando me importa que tan rápido quiero una respuesta**

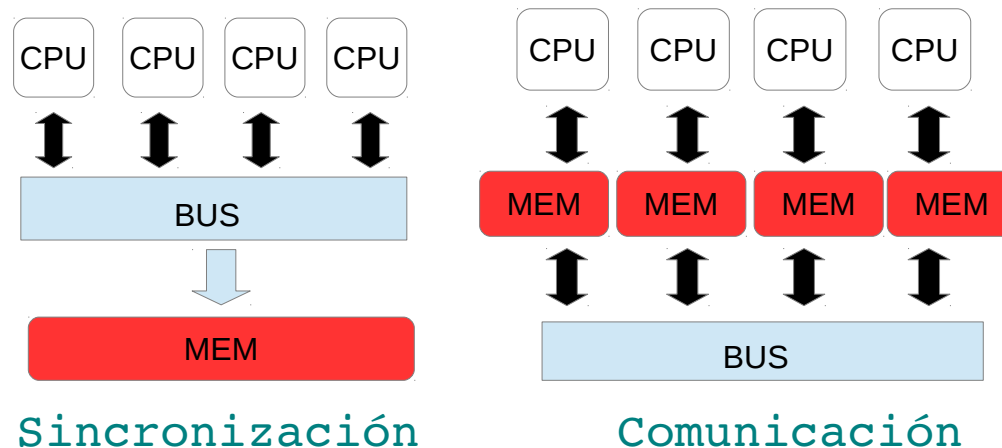
(Computación para Altas Prestaciones o Alta Productividad)

Por lo tanto, HPC puede ocurrir para:

- ❖ Una estación de trabajo (desktop, laptop)
- ❖ Smartphone!
- ❖ Una supercomputadora
- ❖ Un Cluster Linux
- ❖ En una Grid, o Cloud computing , etc

# Programación de aplicaciones paralelas

Programas que sean capaces de utilizar la arquitectura disponible

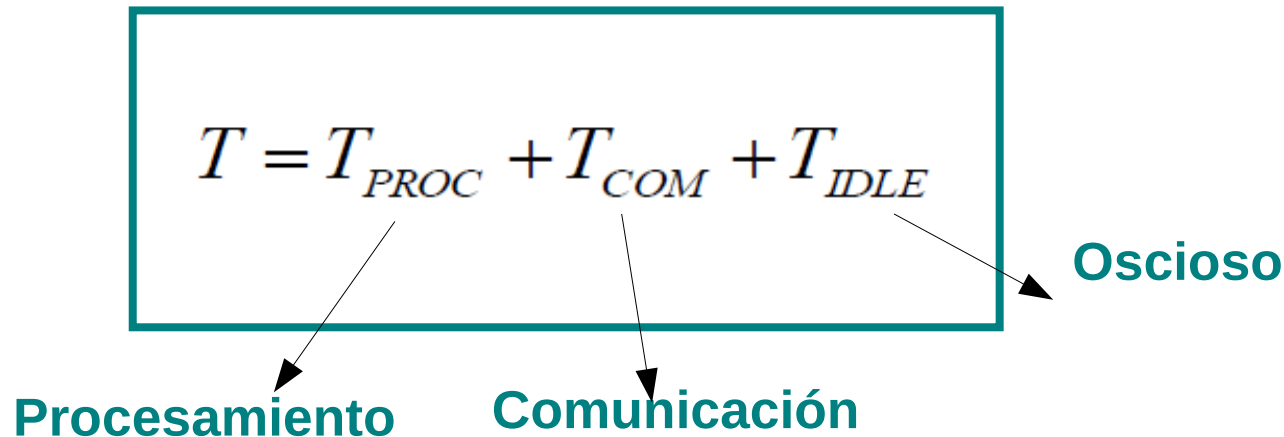


Utilizar eficientemente todos los recursos disponibles

- ❖ Problemas complicados.
- ❖ Modelos complejos.
- ❖ Grandes volúmenes de datos.
- ❖ Capacidad de respuesta en tiempo limitado (sistemas de tiempo real).

# Rendimiento de Aplicaciones Paralelas

De que depende el tiempo de ejecución de un programa paralelo?

$$T = T_{PROC} + T_{COM} + T_{IDLE}$$


**Procesamiento**      **Comunicación**      **Oscioso**

Tiempo que transcurre desde el inicio de la ejecución del primer proceso hasta el fin de ejecución del último proceso.



# Rendimiento de Aplicaciones Paralelas

$T_{PROC}$

Depende de la complejidad y dimensión del problema, del número de tareas utilizadas y de las características de los elementos de procesamiento (hardware, heterogeneidad, no dedicación)

$T_{COM}$

Depende de la localidad de procesos y datos (comunicación inter e intra-procesador, canal de comunicación)

$T_{IDLE}$

Es consecuencia del no determinismo en la ejecución, minimizarlo es un objetivo de diseño.

Motivos: ausencia de recursos de computo disponible o ausencia de datos sobre los cuales operar

Solución: técnicas de balance de carga o rediseñar el programa para distribuir los datos adecuadamente

# Rendimiento de Aplicaciones Paralelas

**SPEED UP** Medida de la mejora de rendimiento de una aplicación al aumentar la cantidad de procesadores (comparando con el rendimiento de utilizar un solo procesador)

## SPEED UP ABSOLUTO

$$S_N = T_0 / T_N$$

$T_0$  tiempo del MEJOR ALGORITMO SECUENCIAL

$T_N$  tiempo del algoritmo paralelo ( N procesadores)

$$S_N = T_1 / T_N$$

## SPEED UP ALGORITMICO

$T_1$  tiempo en un procesador serial

$T_N$  tiempo en paralelo.

# Rendimiento de Aplicaciones Paralelas

Como analizo el speed-up?

Lo ideal es tener un **speedup lineal** → **Si uso p procesadores tengo una mejora de factor p**

En general: Speed-up sublineal

En algunos casos se puede llegar a un speed-up superlineal (casos especiales del problema o del hardware disponibles)

# Rendimiento de Aplicaciones Paralelas

## EFICIENCIA COMPUTACIONAL

$$E_N = T_1 / (N \times T_N)$$



$$E_N = S_N / N$$

Valor normalizado del speed-up (entre 0 y 1), respecto a la cantidad de procesadores utilizados

Lo ideal es que se encuentre cerca a 1.

# Rendimiento de Aplicaciones Paralelas

LEY DE AMDAHL (1967):

“La parte serial de un programa determina una cota inferior para el tiempo de ejecución, aún cuando se utilicen al máximo técnicas de paralelismo.”

# Rendimiento de Aplicaciones Paralelas

## Conclusión de la ley de AMDAHL

La razón para utilizar un número mayor de procesadores debe ser resolver problemas más grandes o más complejos, y no para resolver más rápido un problema de tamaño fijo.

# Rendimiento de Aplicaciones Paralelas

Objetivo de Diseño



Grado de paralelismo  
obtenido

vs

Overhead x  
sincronización y  
Comunicación

# MPI (Message Passing Interface)

**Objetivo:** desarrollar un estándar portable y eficiente, para programación paralela.

**Plataforma objetivo:** memoria distribuida.

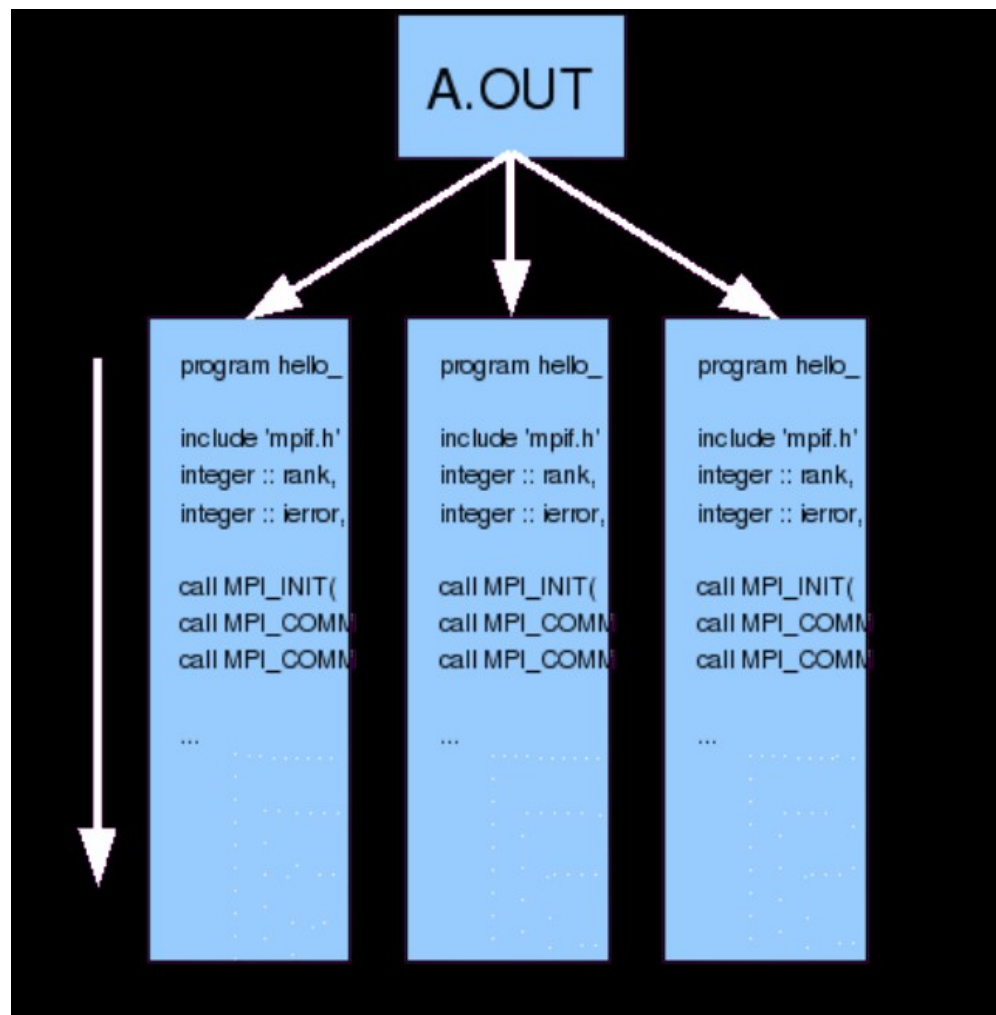
**Paralelismo explícito** (definido y controlado en su totalidad por el programador). Modelo de programas: **SPMD** (single program, multiple data o un programa, múltiples datos).

Número de tareas fijado en tiempo pre-ejecución.

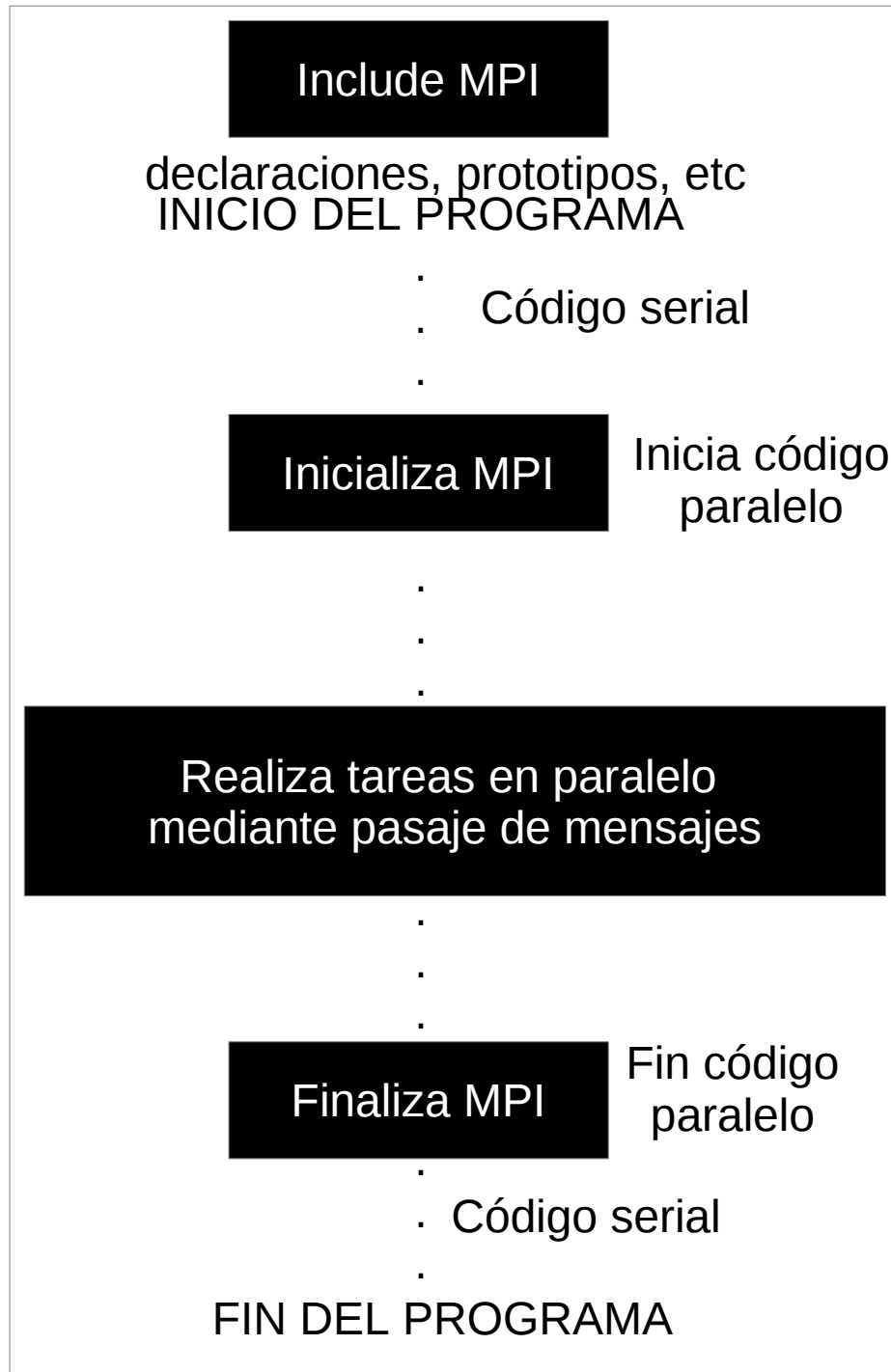


# MPI (Message Passing Interface)

Se basa en ejecutar un mismo código en múltiples procesadores, para ello es necesario gestionar la coordinación/comunicación entre los nodos.

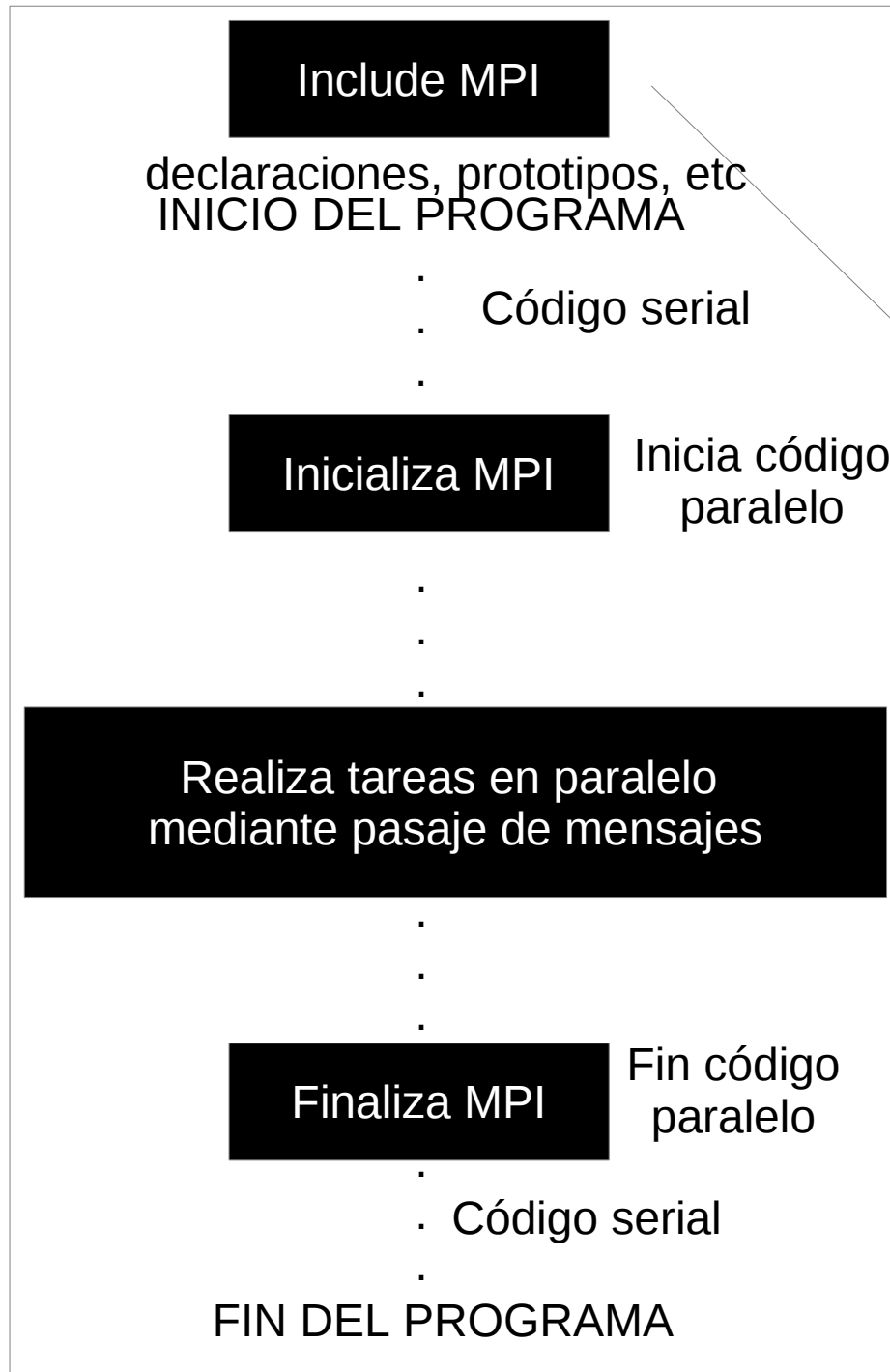


# MPI (Message Passing Interface)



Forma general de un programa que utiliza MPI

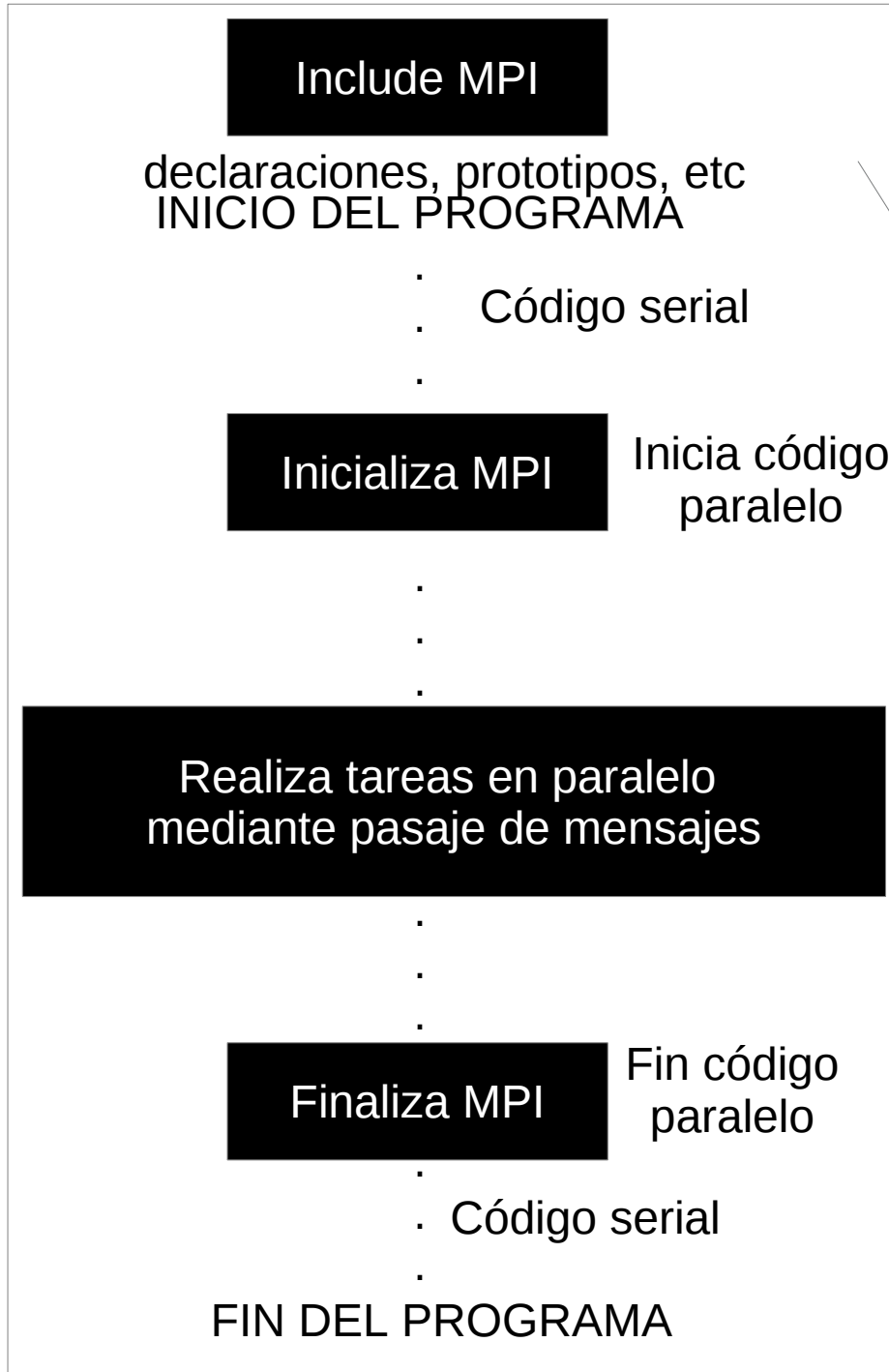
# MPI (Message Passing Interface)



C:  
`#include <mpi.h>`

Fortran:  
`include 'mpif.h'`

# MPI (Message Passing Interface)



## Formato de funciones en MPI

C:

```
error = MPI_Xxxxx(parameter, ...);
```

```
MPI_Xxxxx(parameter, ...);
```

Fortran:

```
CALL MPI_XXXXX(parameter, ..., IERROR)
```

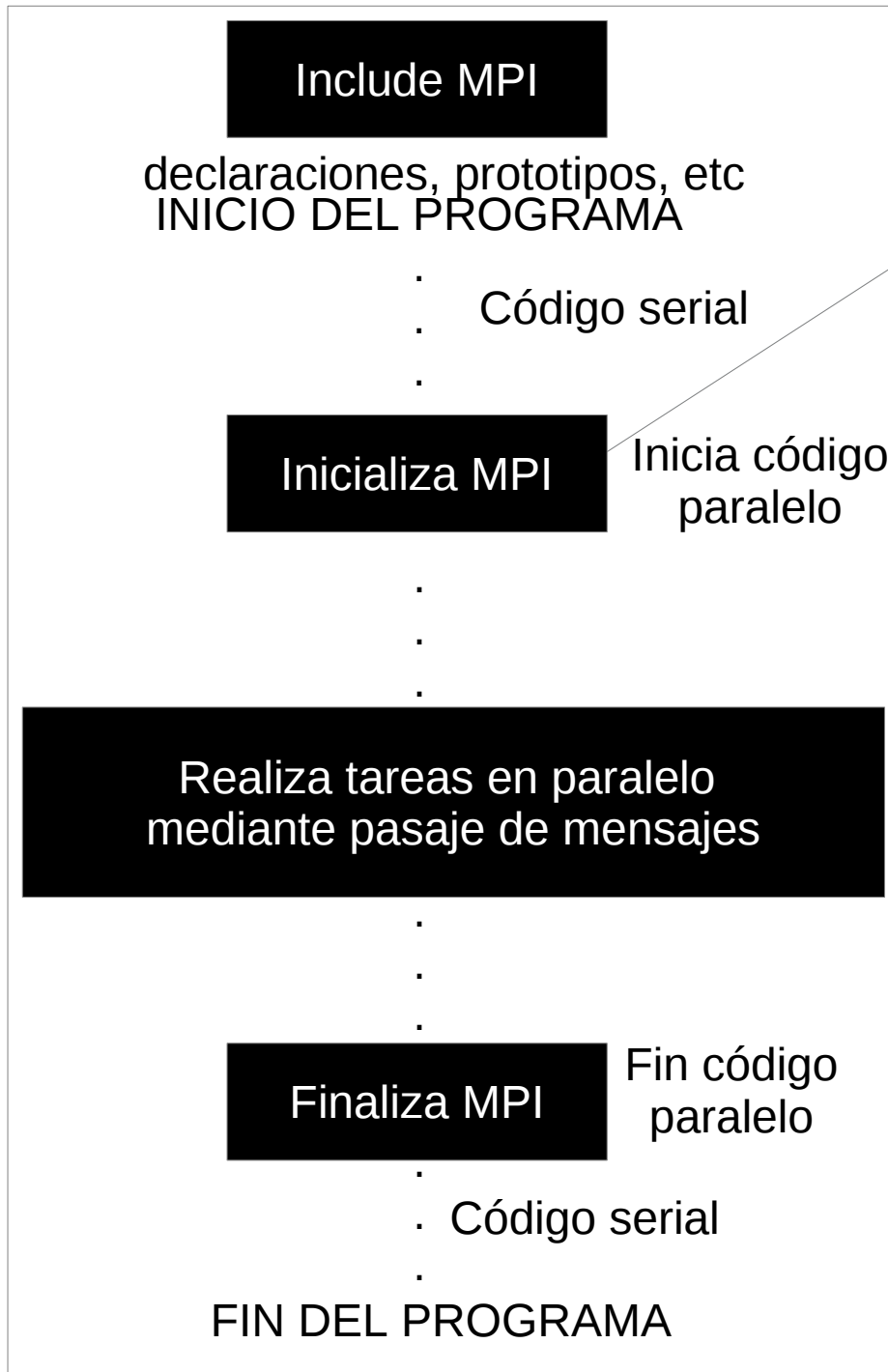
MPI posee estructuras de datos propias pero:

El programador puede acceder a estas mediante "handles"

C → defined typedefs.      20

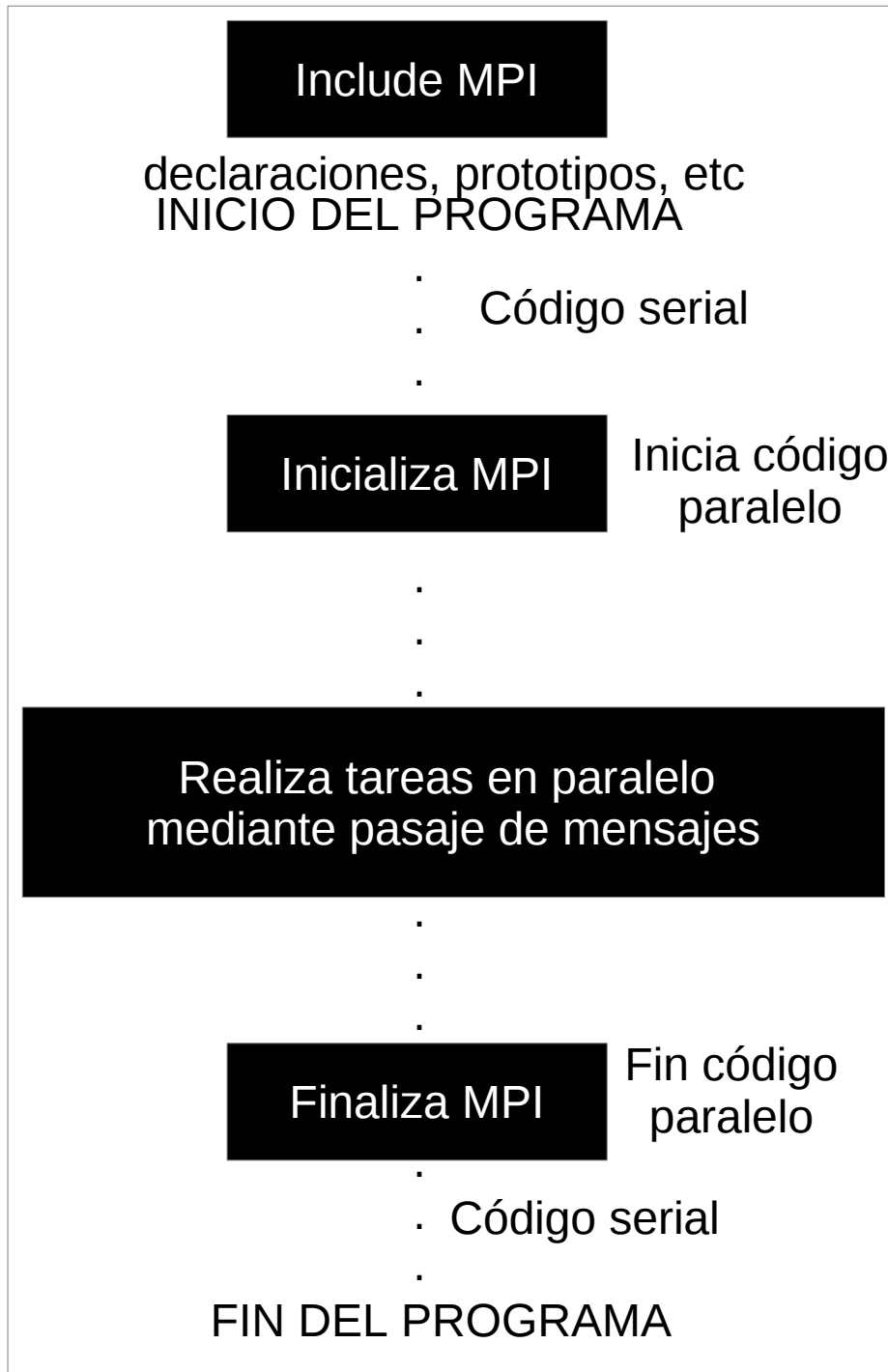
Fortran → Enteros.

# MPI (Message Passing Interface)



```
C:  
int MPI_Init(int *argc, char ***argv)  
  
Fortran:  
MPI_INIT(IERROR)  
INTEGER IERROR
```

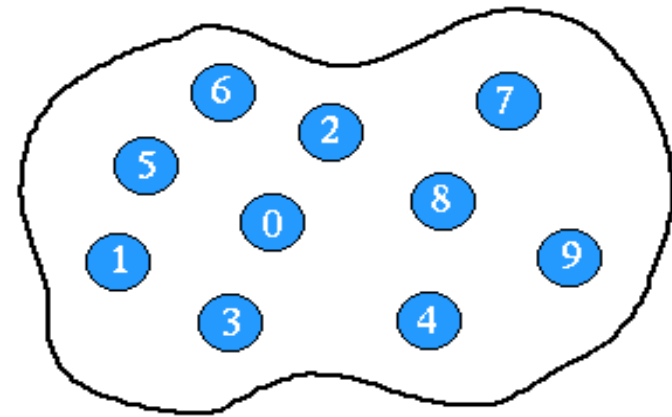
# MPI (Message Passing Interface)



¿ Cómo trabajan los mensajes?

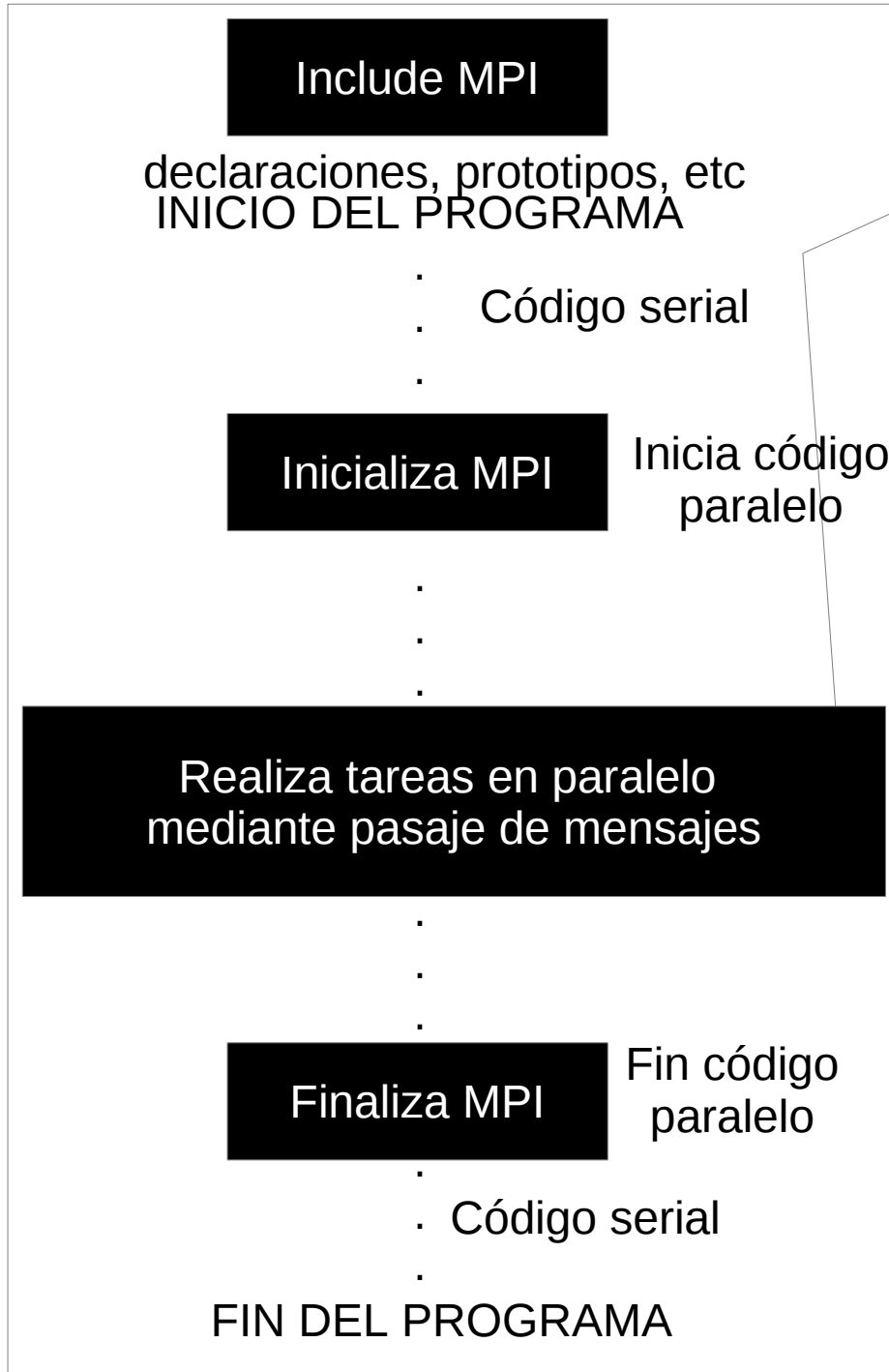
COMUNICADORES

MPI\_COMM\_WORLD



Permite especificar el conjunto de procesos que participan en una operación colectiva.

# MPI (Message Passing Interface)



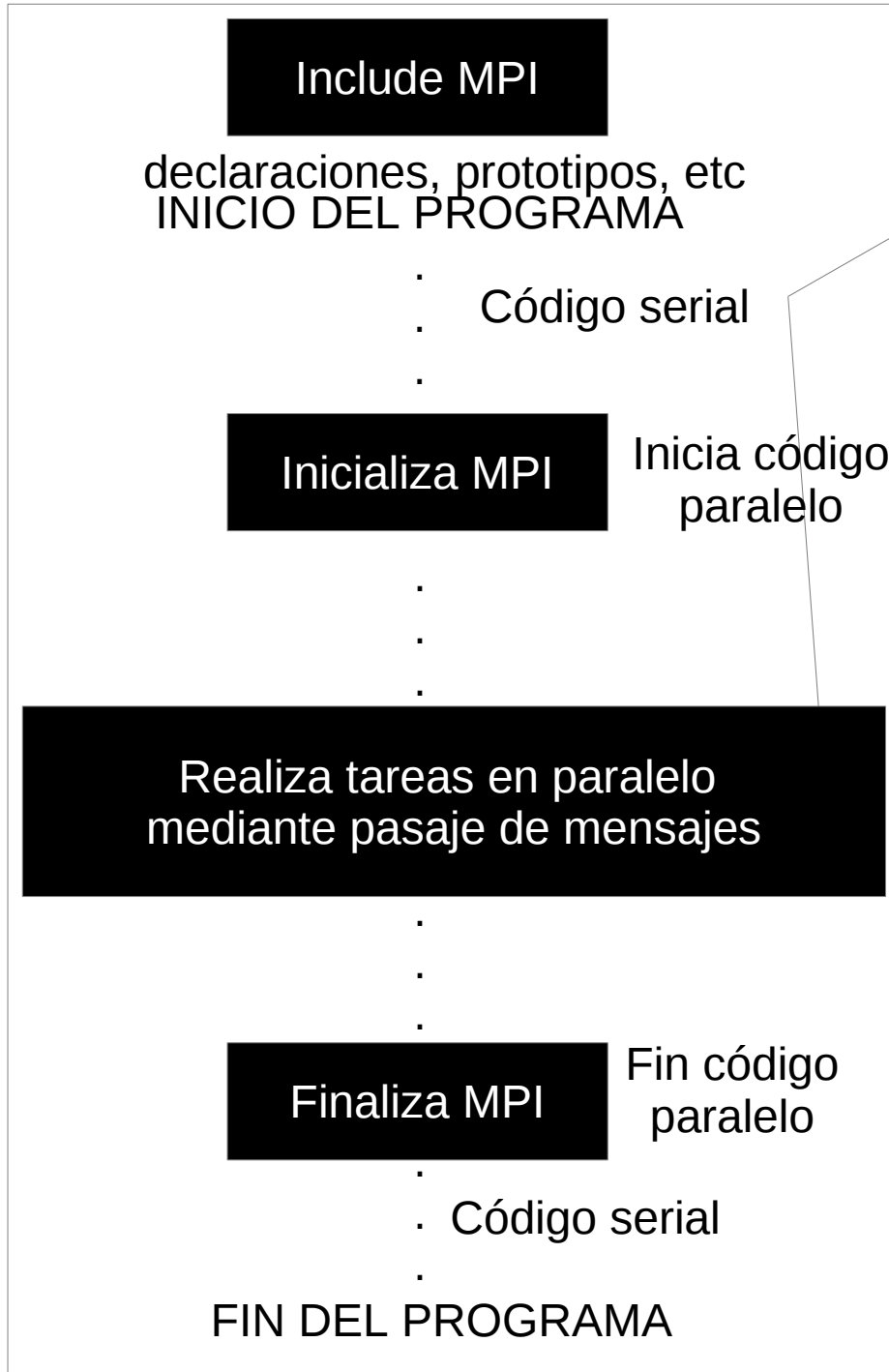
¿Cómo identifico a un proceso dentro de un comunicador?

C:  
ierr =MPI\_Comm\_rank(MPI\_Comm comm,  
int \*rank)

Fortran:  
MPI\_COMM\_RANK(COMM, RANK,  
IERROR)  
INTEGER COMM, RANK, IERROR

rank (rango) es el  
identificador de un proceso  
dentro de un comunicador

# MPI (Message Passing Interface)



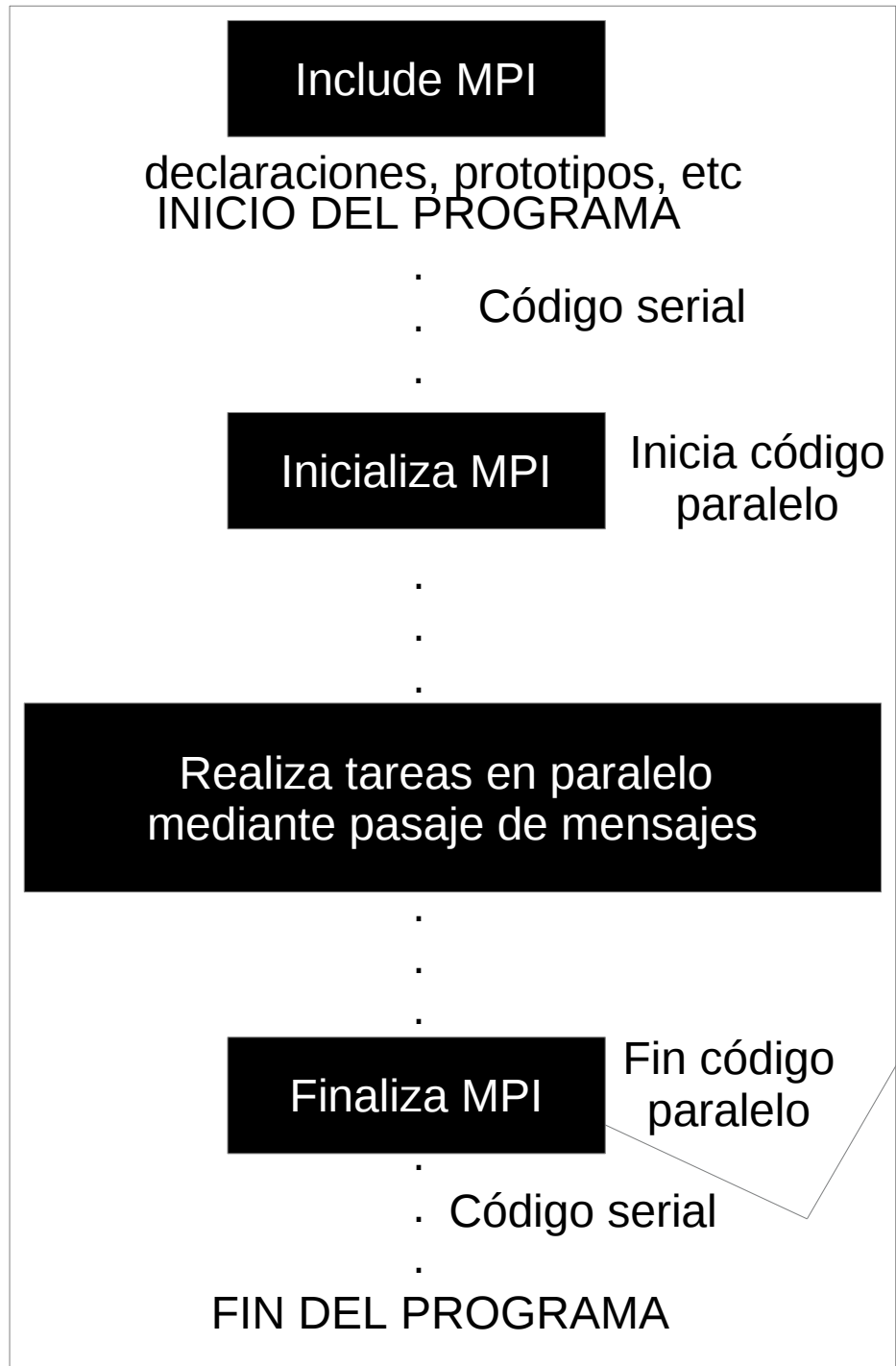
¿Cuántos procesos hay en el comunicador?

C:  
`ierr=MPI_Comm_size(MPI_Comm comm,  
int *size)`

Fortran:  
`MPI_COMM_SIZE(COMM, SIZE,  
IERROR)`  
`INTEGER COMM, SIZE, IERROR`



# MPI (Message Passing Interface)



C:  
int MPI\_Finalize()

Fortran:  
MPI\_FINALIZE(IERROR)  
INTEGER IERROR

# Hello World!

```
#include <stdio.h>
```

```
#include <mpi.h>
```

```
int main (argc, argv)
```

```
    int argc;
```

```
    char *argv[];
```

```
{
```

```
    int rank, size;
```

```
    double inicio,fin;
```

```
    MPI_Init (&argc, &argv);
```

```
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
```

```
    MPI_Comm_size (MPI_COMM_WORLD, &size);
```

```
    printf( "Hello world : procesador %d de %d\n", rank, size );
```

```
    MPI_Finalize();
```

```
    return 0;}
```

# Hello World!

```
maria@maria-UX21E:~/wtpc17$  
maria@maria-UX21E:~/wtpc17$ mpicc helloworld.c -o hola  
maria@maria-UX21E:~/wtpc17$ mpirun -np 4 hola  
Hello world : procesador 3 de 4  
Hello world : procesador 1 de 4  
Hello world : procesador 0 de 4  
Hello world : procesador 2 de 4  
maria@maria-UX21E:~/wtpc17$
```

# Hello World!

```
maria@maria-UX21E:~/wtpc17$  
maria@maria-UX21E:~/wtpc17$ mpicc helloworld.c -o hola  
maria@maria-UX21E:~/wtpc17$ mpirun -np 4 hola  
Hello world : procesador 3 de 4  
Hello world : procesador 1 de 4  
Hello world : procesador 0 de 4  
Hello world : procesador 2 de 4  
maria@maria-UX21E:~/wtpc17$
```

# Hello World!

```
maria@maria-UX21E:~/wtpc17$  
maria@maria-UX21E:~/wtpc17$ mpicc helloworld.c -o hola  
maria@maria-UX21E:~/wtpc17$ mpirun -np 4 hola  
Hello world : procesador 3 de 4  
Hello world : procesador 1 de 4  
Hello world : procesador 0 de 4  
Hello world : procesador 2 de 4  
maria@maria-UX21E:~/wtpc17$
```

# Hello World!

Ver los recursos de procesamiento y memoria del equipo

cat /proc/cpuinfo

```
maria@maria-UX21E:~/CODIGOS MNII/OPTIMIZACION$ cat /proc/cpuinfo
processor      : 0
vendor_id    : GenuineIntel
cpu family   : 6
model       : 42
model name   : Intel(R) Core(TM) i5-2467M CPU @ 1.60GHz
stepping    : 7
microcode   : 0x1a
cpu MHz     : 800.000
cache size  : 3072 KB
physical id : 0
siblings    : 4
core id     : 0
cpu cores   : 2
apicid      : 0
initial apicid : 0
fpu         : yes
fpu_exception : yes
cpuid level : 13
wp          : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pg
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
onstant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16
cid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx lahf
t epb xsaveopt pln pts dtherm tpr_shadow vnmi flexpriority ept vpid
bogomips    : 3192.76
clflush size : 64
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

processor      : 1
vendor_id    : GenuineIntel
cpu family   : 6
model       : 42
model name   : Intel(R) Core(TM) i5-2467M CPU @ 1.60GHz
```

cat /proc/cpuinfo | grep processor | wc -l

# de unidades de  
procesamiento

cat /proc/cpuinfo | grep 'core id'

nproc

free -m    Uso de memo en Mb

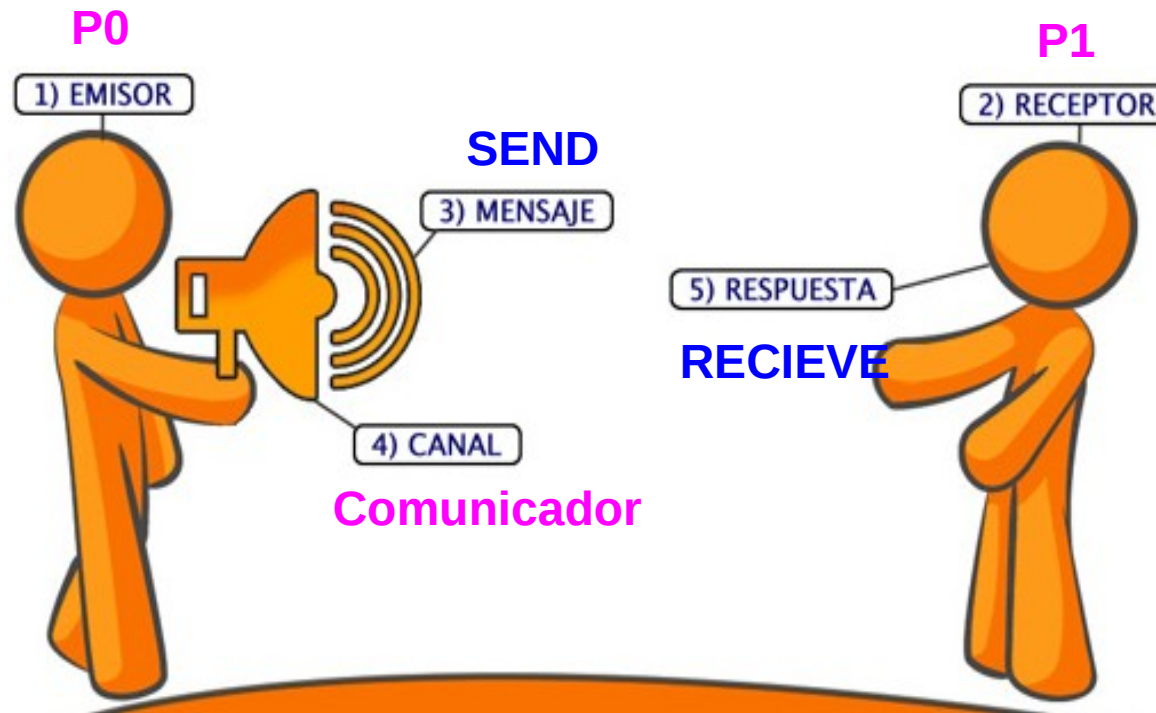
# MPI (Message Passing Interface)

**Pero ... ¿ Cómo se comunican los procesos?**



**MENSAJES!**

# Comunicación Punto a Punto



- Comunicación se realiza entre dos procesos
- El proceso “fuente” envía un mensaje al proceso “destino”
- La comunicación ocurre dentro de un comunicados
- El proceso destino está identificado por su rank (o rango) dentro del comunicador



# MPI (Message Passing Interface)

## MENSAJES

MPI datatypes: básicos y derivados (diferentes para C y Fortran)

MPI Datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Datatype	Fortran Datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

# Comunicación Punto a Punto

## ENVIAR

C:

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

Fortran:

```
MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)  
<type> BUF(*)  
INTEGER COUNT, DATATYPE, DEST, TAG  
INTEGER COMM, IERROR
```

# Comunicación Punto a Punto

## RECIBIR

C:

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

Fortran:

```
MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM,  
STATUS, IERROR)  
<type> BUF(*)  
INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM,  
STATUS(MPI_STATUS_SIZE), IERROR
```

# Comunicación Punto a Punto

## Ejemplo

...

```
MPI_Comm_size(MPI_COMM_WORLD, &N);
```

```
int tag=0;
```

```
MPI_Send(&valor, 1, MPI_INT, N, tag, MPI_COMM_WORLD);
```

...

```
MPI_Recv(&valor, 1, MPI_INT, 0, tag, MPI_COMM_WORLD,  
MPI_STATUS_IGNORE);
```

...



# Programación en Entornos Paralelos: MPI

Graciela Molina  
m.graciela.molina@gmail.com