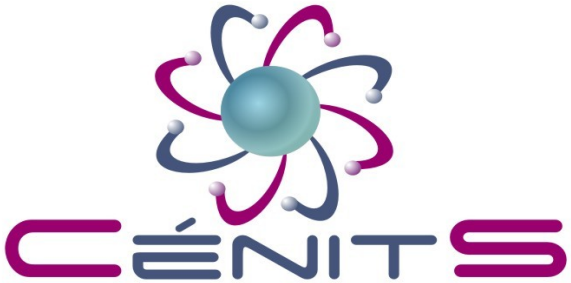
Lusitania

Pensando en Paralelo



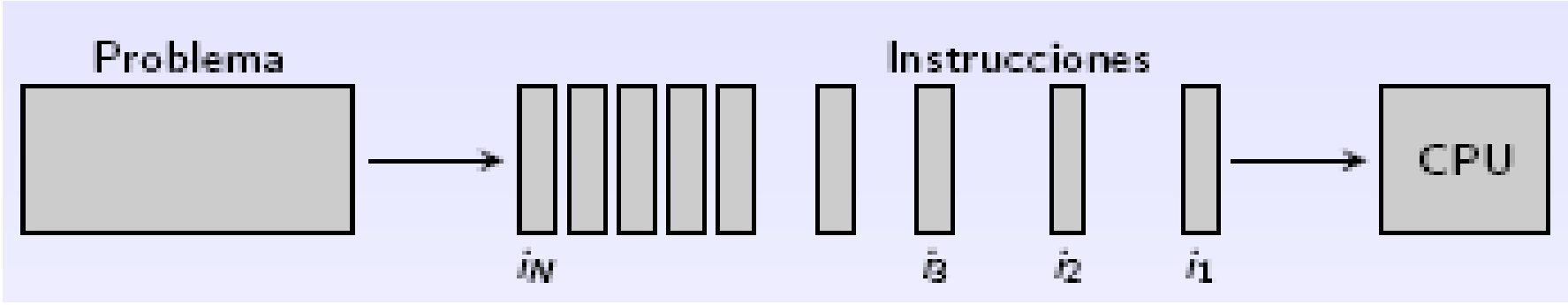
César Gómez Martín
cesar.gomez@cenits.es
www.cenits.es

Esquema

- Introducción a la programación paralela
- ¿Por qué paralelizar?
- Tipos de computadoras paralelas
- Paradigmas de programación paralela
- ¿Puedo paralelizar mi programa?
- Ejemplos

Introducción a la Programación Paralela (I)

- Normalmente los programas realizan cálculos en **serie**:
  - Se ejecutan en un ordenador con **una única CPU**
  - Las instrucciones se ejecutan secuencialmente
  - Una única instrucción se ejecuta a la vez

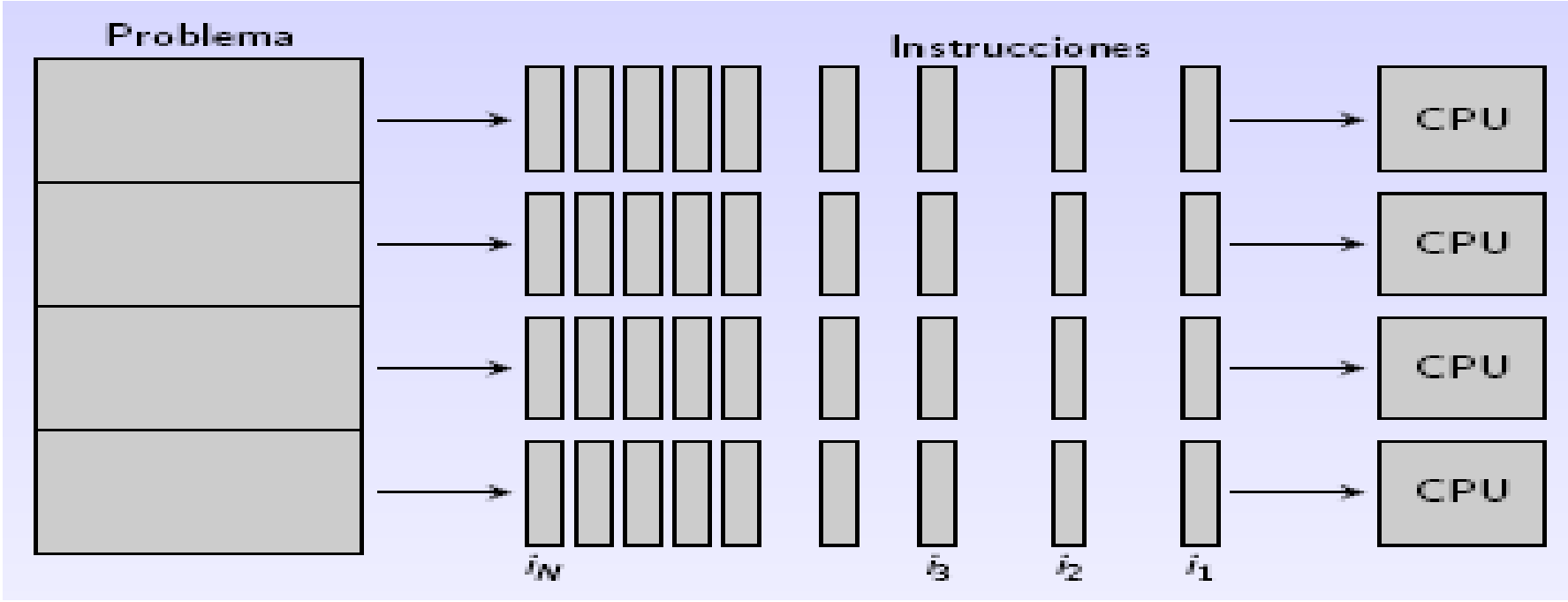


Introducción a la Programación Paralela (II)

- La programación paralela consiste en **usar varios recursos de forma simultánea** para resolver un problema:
  - Se ejecutan en un ordenador con **varias CPUs**
  - El problema se divide en partes independientes
  - Cada parte se ejecuta simultáneamente

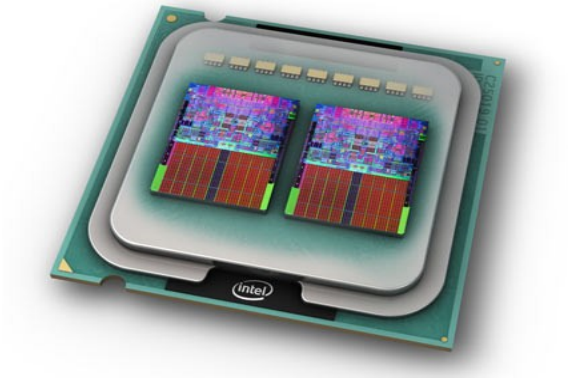
Introducción a la Programación Paralela (y III)

- La programación paralela consiste en usar varios recursos de forma simultánea para resolver un problema:

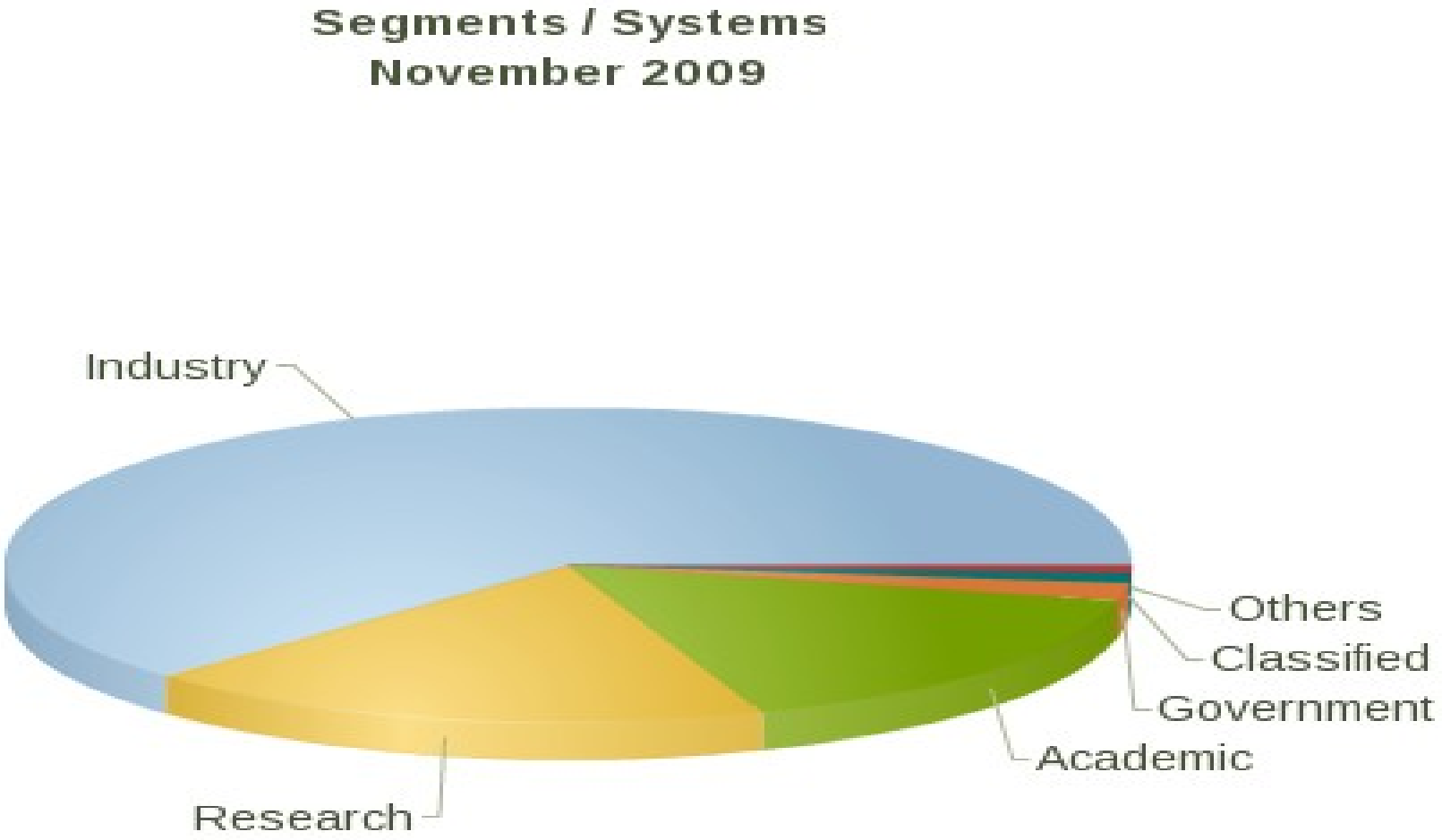


¿Por qué paralelizar? (I)

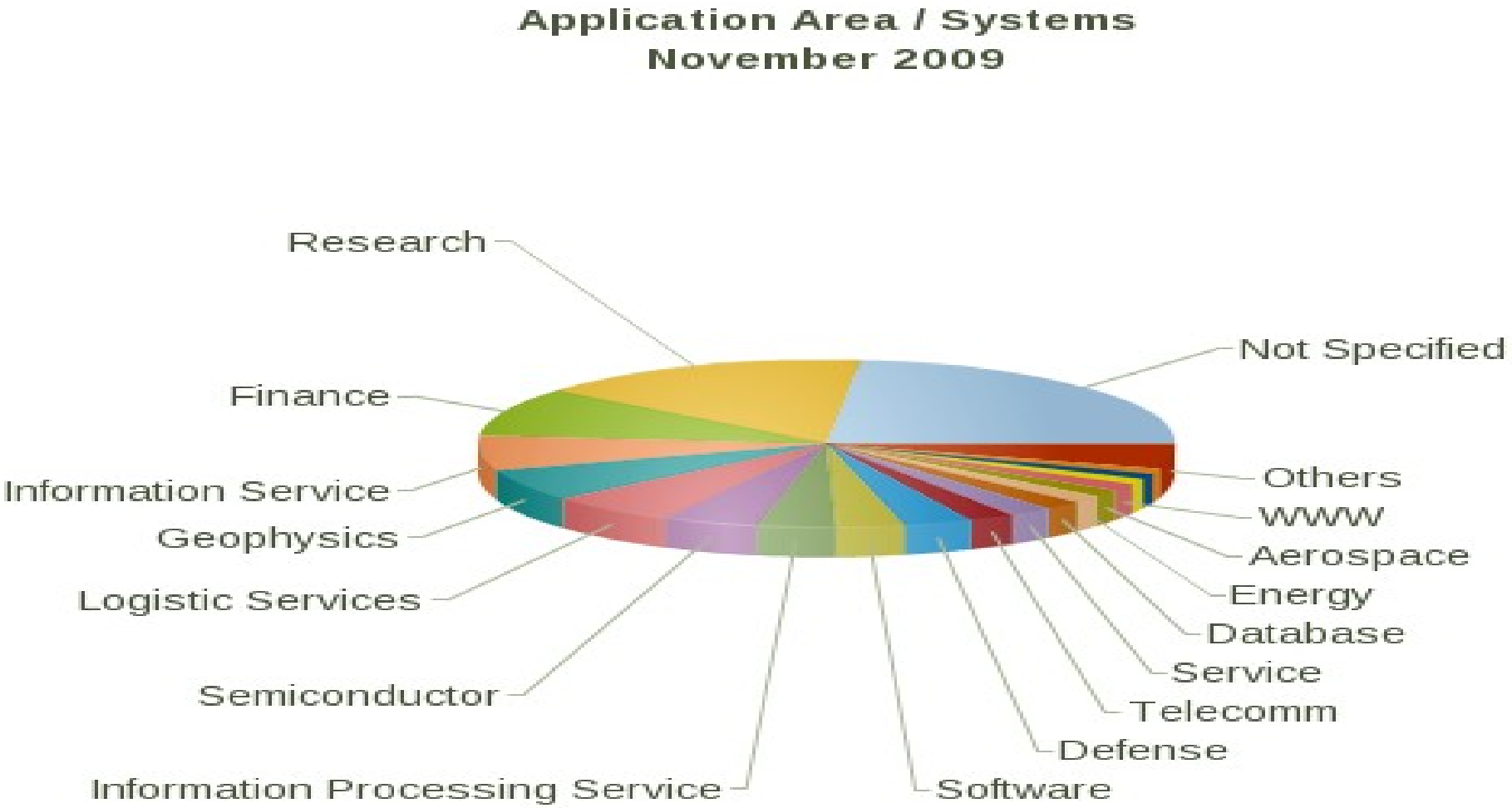
- Resultados en **menos tiempo** (wall clock time)
- Solución a **problemas mas grandes/complejos**
- Posibilidad de realizar barridos paramétricos
- Estudio de diferentes variantes del problema
- Los **procesadores actuales son de n-cores**



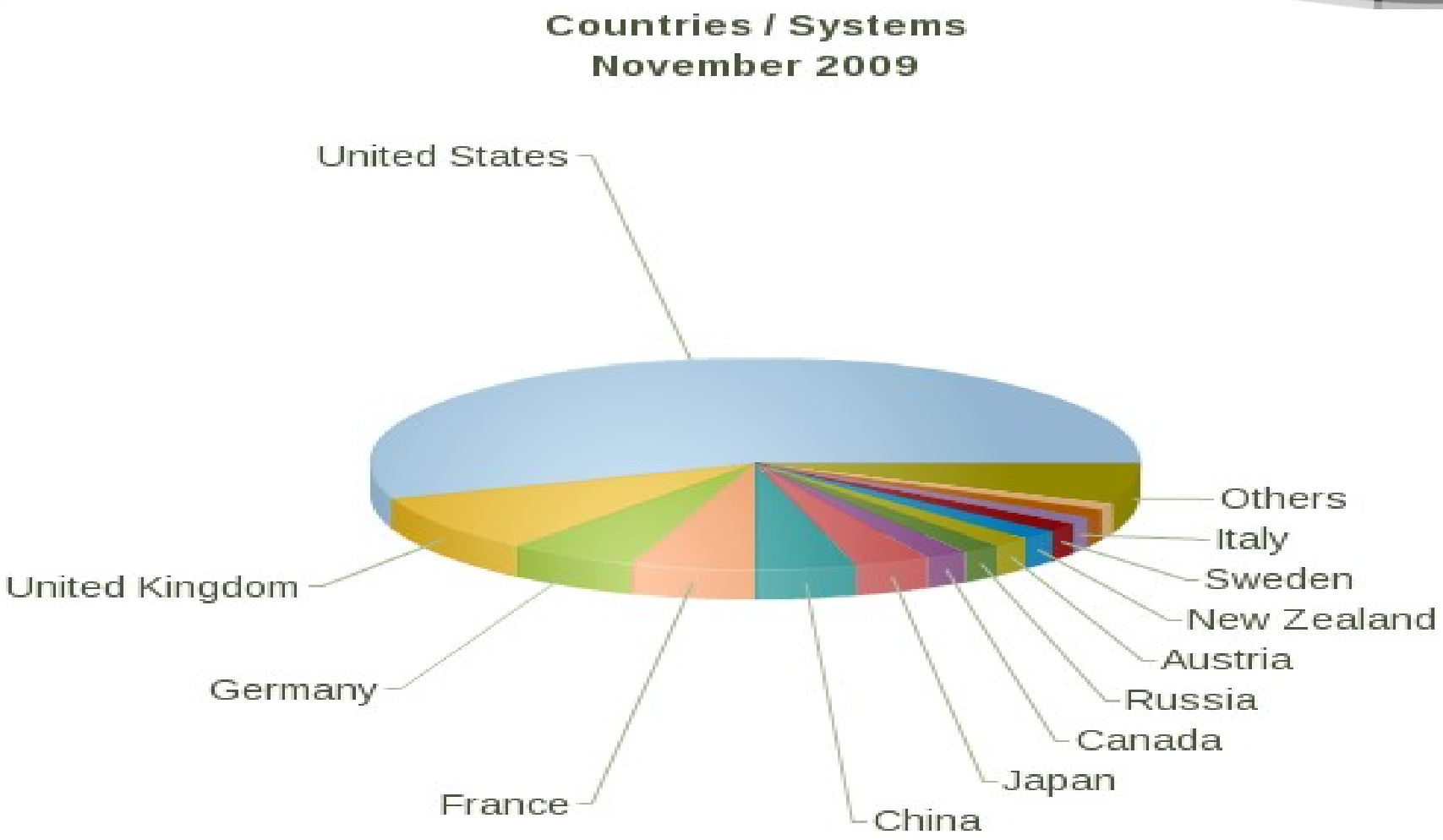
¿Por qué paralelizar? (II)



¿Por qué paralelizar? (III)



¿Por qué paralelizar? (y IV)

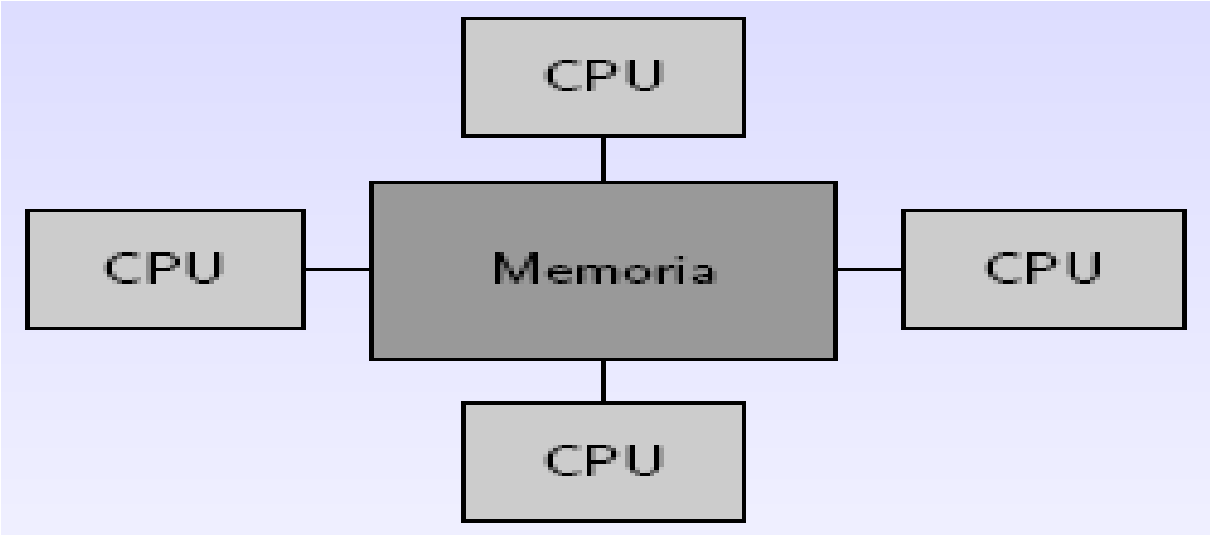


Tipos de Computadoras Paralelas (I)

- Se pueden clasificar dependiendo de:
  - El procesador
  - La distribución de memoria
  - La arquitectura de la máquina
- Se suelen clasificar según la distribución de memoria
  - Memoria compartida
  - Memoria distribuida
  - Híbridos

Tipos de Computadoras Paralelas (II)

- Computadora de memoria compartida
  - Las **CPUs acceden a la misma memoria**
  - Los c**ambios en memoria afectan a todas las CPUs**
  - Hay 2 tipos **UMA** y **NUMA**



Tipos de Computadoras Paralelas (II)

- Computadora de memoria compartida **UMA (Uniform Memory Access)**
  - Las **CPUs** están a la **misma distancia de la memoria**
  - Son máquinas SMP (Symmetric MultiProcessor) puras

Placa base Supermicro X7DB3

Tipos de Computadoras Paralelas (III)

- Computadora de memoria compartida **NUMA (Non-Uniform Memory Access)**
  - Las **CPUs *no* están a la misma distancia de la memoria**
  - A veces son máquinas SMP interconectadas

Tipos de Computadoras Paralelas (IV)

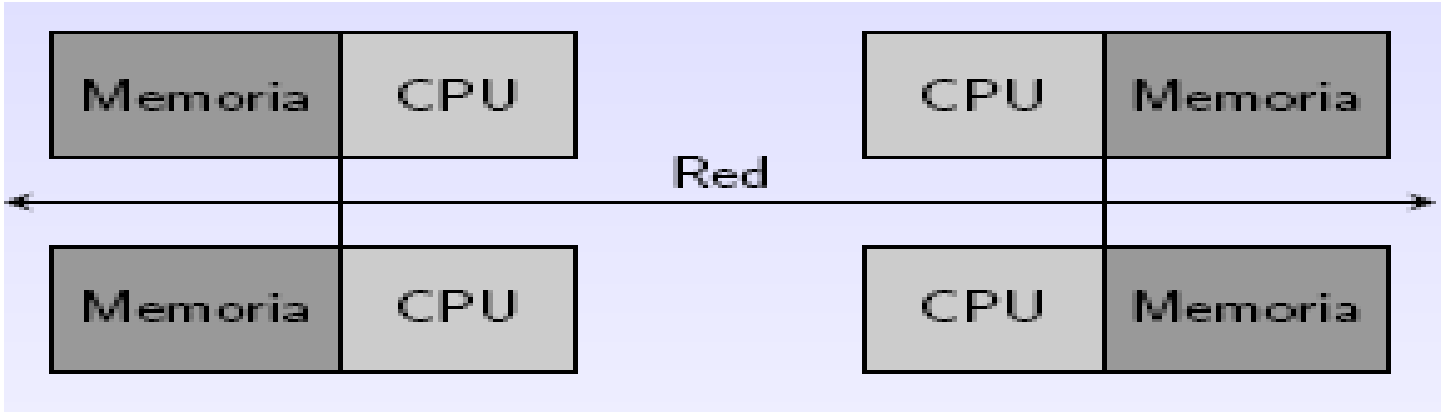
- **HP Integrity Superdome sx2000**
  - Arquitectura **ccNUMA**
  - **128 núcleos** con acceso a la misma memoria
  - **2TB de memoria** RAM

Tipos de Computadoras Paralelas (V)

- Computadoras con memoria compartida:
  - Ventajas
    - Fáciles de programar
    - Compartir datos entre procesos o threads es fácil y muy rápido
  - Desventajas
    - Es muy caro hacer computadoras con muchas CPUs

Tipos de Computadoras Paralelas (VI)

- Computadoras con memoria distribuida:
  - **Cada CPU** tiene su **propia memoria local**
  - La **memoria local** de una CPU ***no*** **es visible por las demás CPUs**
  - La información se comparte a través de una red de comunicaciones

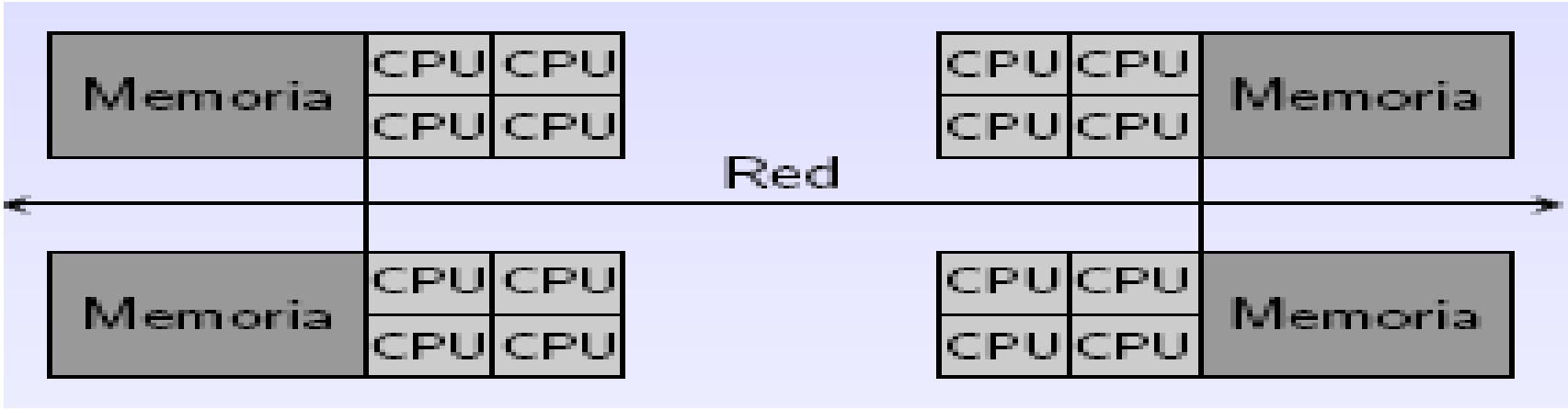


Tipos de Computadoras Paralelas (VII)

- Computadoras con memoria distribuida:
  - Ventajas:
    - El coste es “lineal” en cuanto al número de CPUs
  - Desventajas:
    - El programador es responsable de las comunicaciones
    - La red de comunicaciones suele ser el “cuello de botella”
    - La paralelización de programas puede no ser trivial

Tipos de Computadoras Paralelas (VIII)

- Computadoras híbridas:
  - **Grupos de CPUs comparten** una misma **memoria**
  - Los grupos de CPU se comunican a través de una red
  - Suelen ser máquinas SMP conectadas entre sí



Tipos de Computadoras Paralelas (IX)

- Computadoras con memoria distribuida:
  - **Lusitania**
    - 2 nodos SMP con 128 cores/nodo (Itanium2)
    - 1TB de memoria por nodo
    - **Total: 256 cores y 2 TB de memoria**

Tipos de Computadoras Paralelas (y X)

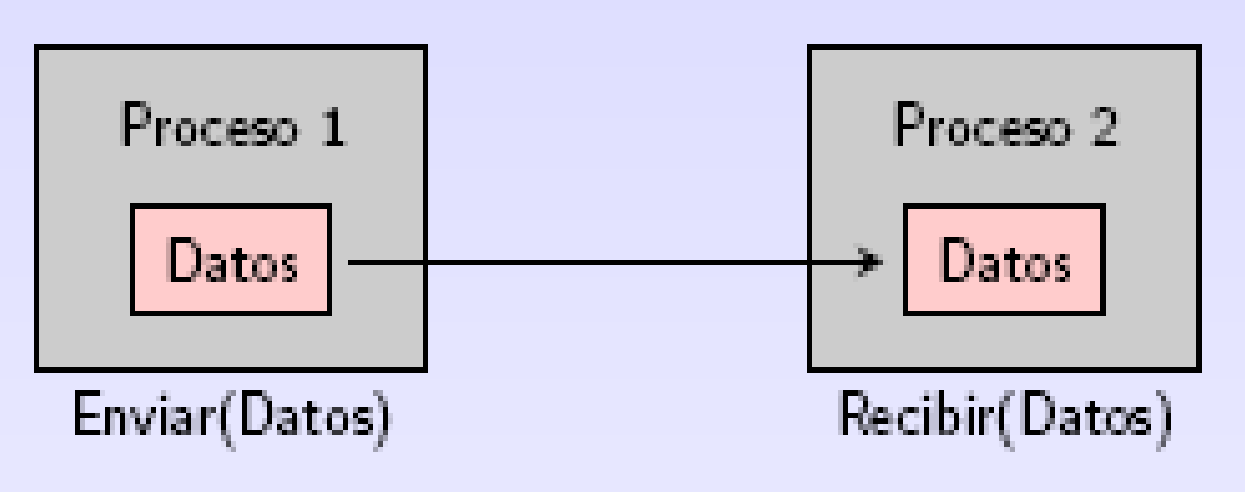
- Computadoras híbridas:
  - Ventajas
    - La escalabilidad entre CPUs y memoria es buena
    - El coste es “lineal” con el número de grupos de CPUs
    - La red de comunicaciones no es crítica
  - Desventajas
    - El programador es responsable de la sincronía entre nodos
    - La paralelización puede no ser trivial

Paradigmas de Programación Paralela (I)

- Existen varias formas de programar en paralelo
  - Memoria compartida
  - **Paso de mensajes**
  - **Tareas**
  - Paralelismo de datos
  - Operaciones remotas en memoria
  - Modelos combinados
- **No son excluyentes entre sí**
- **No depende del tipo de [super]computador**

Paradigmas de Programación Paralela (II)

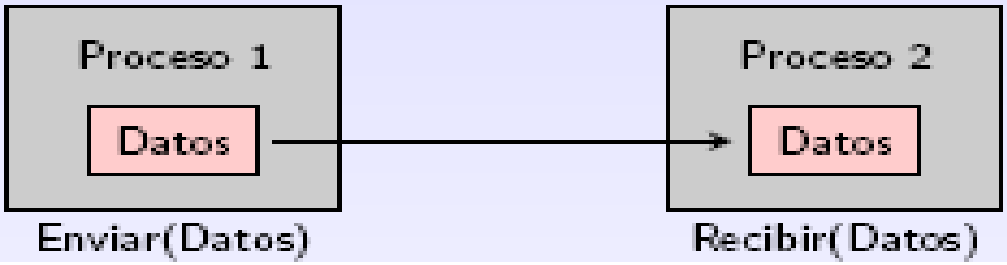
- **Modelo de paso de mensajes**
  - Los procesos intercambian datos mediante mensajes
  - El programador es responsable del envío y la recepción de esos mensajes



Paradigmas de Programación Paralela (III)

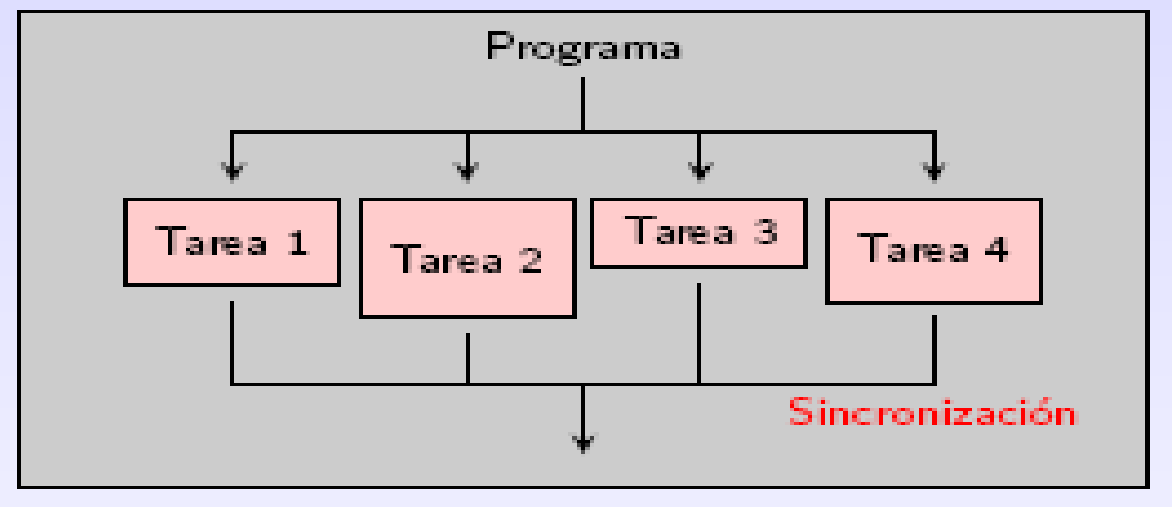
- Modelo de paso de mensajes – **Biblioteca MPI**

```
ID = Quien_soy_yo

Si soy ID = 1 entonces
  envio Datos a 2 y espero confirmacion

Si soy ID = 2 entonces
  recibo Datos de 1 y envio confirmacion
```



Paradigmas de Programación Paralela (IV)

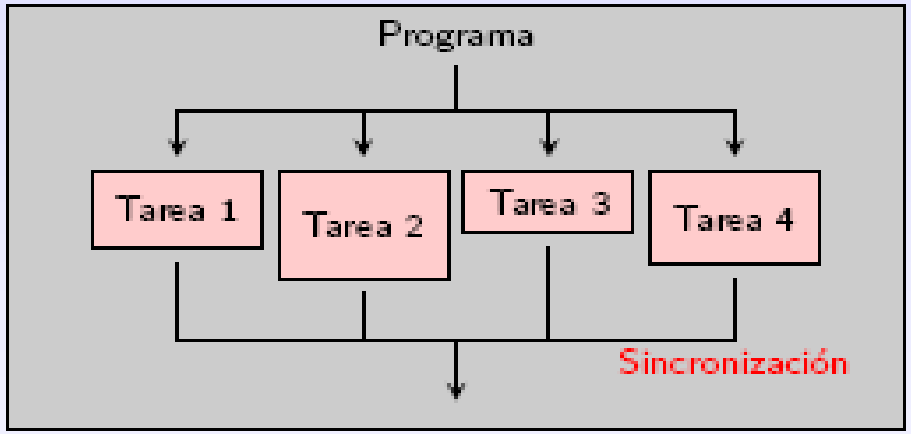
- **Modelo de tareas**
  - El programa define un conjunto de tareas
  - Cada tarea tiene su memoria local y tiene acceso a una memoria conjunta
  - El programador es responsable de la sincronización entre tareas



Paradigmas de Programación Paralela (y V)

- Modelo de tareas – Estándar OpenMP

```
Hacer en paralelo el bucle con i = 1 hasta 25
  La tarea 1 es i =  1 hasta  5
  La tarea 2 es i =  6 hasta 13
  La tarea 3 es i = 14 hasta 18
  La tarea 4 es i = 19 hasta 25

Haced las tareas y os espero
```



¿Puedo paralelizar mi programa? (I)

- Hay problemas paralelizables y no paralelizables:
  - **Problema paralelizable**: calcular el potencial de energía de cientos de conformaciones posibles de una molécula

¿Puedo paralelizar mi programa? (II)

- Hay problemas paralelizables y no paralelizables:
  - **Problema *no* paralelizable**: Cálculo de la serie de **Fibonacci** mediante su fórmula de recurrencia

$$F_{k+2} = F_{k+1} + F_k, \quad F_1 = 1, F_2 = 1$$

¿Puedo paralelizar mi programa? (III)

- Ejemplos de aplicaciones paralelizables:
  - **Conversión a grises de una imagen**
    - La aplicación de un filtro a un píxel no depende de los píxeles vecinos

¿Puedo paralelizar mi programa? (y IV)

- Ejemplos de aplicaciones paralelizables:
  - **Cálculo de integral definida:**

$$I = \int_a^b f(x)\,\mathrm{d}x$$

  - Se puede descomponer en:

$$I = \int_a^{x_1} f(x)\,\mathrm{d}x + \int_{x_1}^{x_2} f(x)\,\mathrm{d}x + \ldots + \int_{x_N}^{b} f(x)\,\mathrm{d}x$$

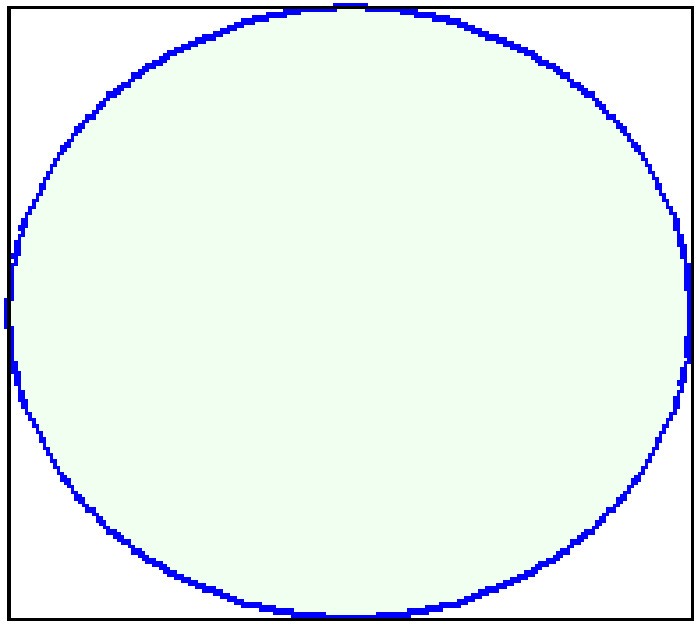
Ejemplos (I)

- **“Hola Mundo” con MPI**:

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

int main (int argc, char *argv[])
{
  int rank, size;

  MPI_Init (&argc, &argv);                    /* Arranca MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);  /* obtiene el Id del proceso actual */
  MPI_Comm_size (MPI_COMM_WORLD, &size);    /* obtiene el número de procesos */

  printf( "Hola desde el proceso %d de %d\n", rank+1, size );

  MPI_Finalize();                             /* Finaliza el entorno de ejecución de MPI */

  return 0;
}
```

Ejemplos (II)

- Cálculo de π usando el **Método Monte Carlo**:
  - Construimos un círculo de **radio=1** dentro de un cuadrado

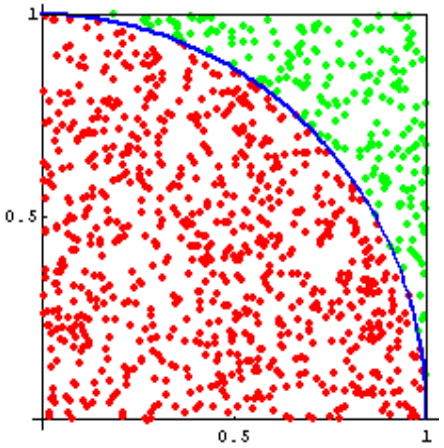


Ejemplos (III)

- Cálculo de π usando el **Método Monte Carlo**:
  - **Área Círculo**= $\pi r^2 = \pi * \mathbf{1}^2 = \pi$
  - **Área Cuadrado**= $(2r)^2 = (2 * \mathbf{1})^2 = 4$
  - El ratio del área del círculo es:
    - **Ratio** = A.Círculo/A.Cuadrado = $\pi/4$
    - **Π = Ratio*4**

Ejemplos (IV)

- Cálculo de π usando el **Método Monte Carlo**:
  - ¿Cómo podemos calcular el ratio?
    - Lanzando dardos de forma aleatoria en una **diana de radio=1** y calculando el ratio de los dardos que dan en la diana (círculo)
    - **Ratio = nº dardos en diana/nº tiradas**

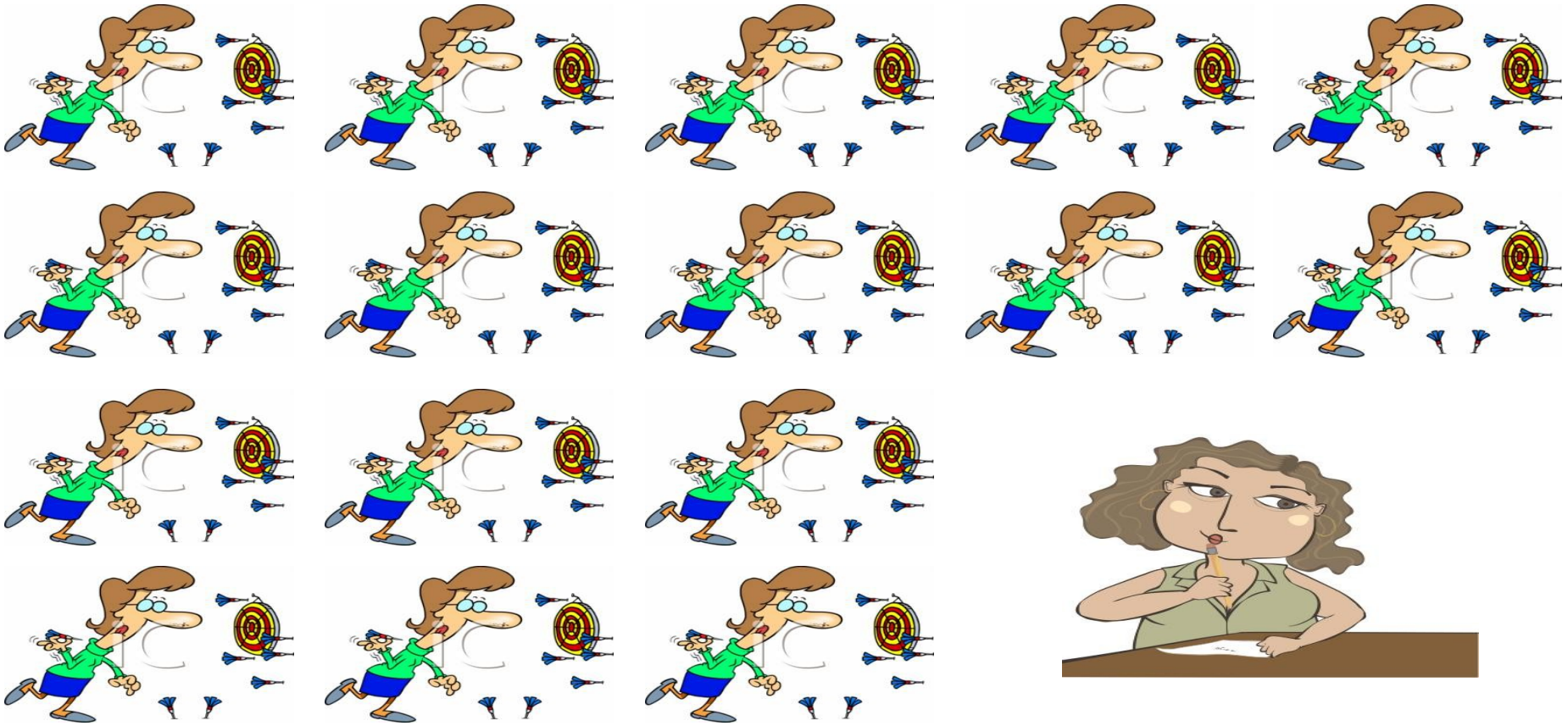


Ejemplos (V)

- **Versión secuencial**:

Ejemplos (VI)

- **Versión paralela**:



Ejemplos (V)

- **Código versión secuencial**:

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

#define DARDOS 10000     /* número de lanzamientos */
#define RONDAS 12800     /* número de rondas */

int main(int argc, char *argv[])
{
double pi;               /* media de pi después de una ronda */
double mediapi;           /* media de pi después de todas las rondas */
float pireal = 3.1415926535897;
int i, n;

srandom (5);
mediapi = 0;
for (i = 0; i < RONDAS; i++) {
   /* Calculo de pi en una ronda */
   pi = diana(DARDOS);
   mediapi = ((mediapi * i) + pi)/(i + 1);
   printf("   Después de %3d lanzamientos, valor medio de pi = %10.8f\n",
         (DARDOS * (i + 1)),mediapi);
   }
printf("\nValor real de PI: %f \n",pireal);
}
```

Ejemplos (VI)

- **Código versión paralela**:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define DARDOS 1000        /* numero de dardos a lanzar */
#define RONDAS 12800       /* numero de rondas */
#define MASTER 0           /* ID del proceso MASTER */

int main (int argc, char *argv[]) {
/* Obtención del número y del ID de tareas */
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

                                        (MASTER)

                                        (WORKER)

MPI_Finalize();
return 0;
}
```

Ejemplos (VII)

- **Código versión paralela (MASTER)**:

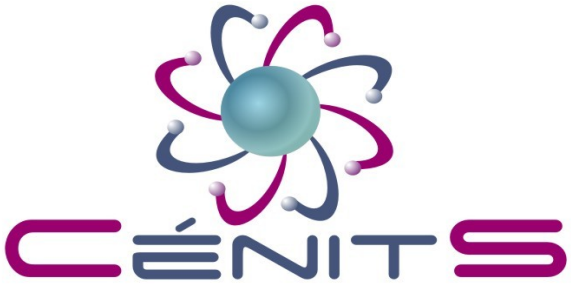
```
...
...
    if (taskid == MASTER) {
    /* El Master recibe mensajes de todos los trabajadores */
    for (n = 1; n < numtasks; n++) {
    rc = MPI_Recv(&pirecv, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
         mtype, MPI_COMM_WORLD, &status);
    /* El Master calcula la media de pi para esta iteración */
    pi = (pisum + homepi)/numtasks;
    /* El Master calcula la media de pi para todas las itereaciones */
    avepi = ((avepi * i) + pi)/(i + 1);
    printf("   Después de %8d lanzamientos, media de pi = %10.8f\n",
            (DARDOS * (i + 1)),avepi);
    }
...
...
```

Ejemplos (VIII)

- **Código versión paralela (WORKER)**:

```
...
...
  /* Los trabajadores envía su cálculo al master */
  if (taskid != MASTER) {
    mtype = i;
    rc = MPI_Send(&homepi, 1, MPI_DOUBLE,
                  MASTER, mtype, MPI_COMM_WORLD);
    if (rc != MPI_SUCCESS)
      printf("%d: Envio fallido en ronda %d\n", taskid, mtype);
    }
...
...
```

Gracias



César Gómez Martín
cesar.gomez@cenits.es
www.cenits.es