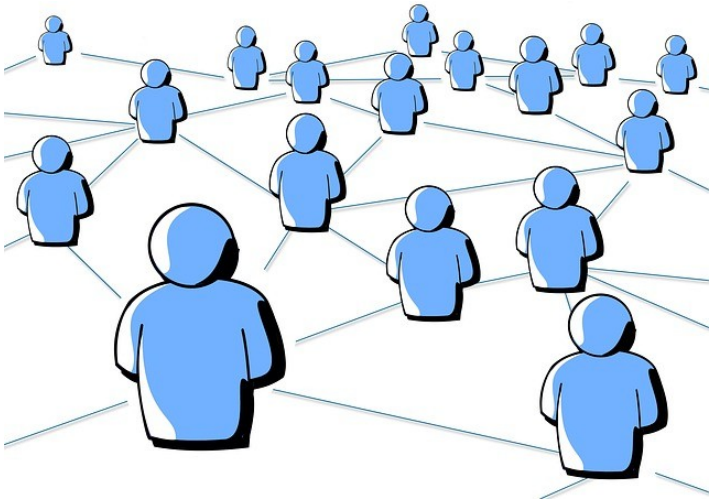


CONCURRENCIA

TEMAS

- Familias de computadoras y evolución
- Programación concurrente
- Evolución de la programación
- Comunicación
- Sincronización

ORGANIZACIÓN DE UNA COMPUTADORA

- Ha posibilitado los últimos avances en rendimiento:
  - Segmentación
  - Paralelismo
  - Computadoras RISC (Reduced Instruction Set Computer).

RISC: 1. Instrucciones de tamaño fijo y presentadas en un reducido número de formatos.

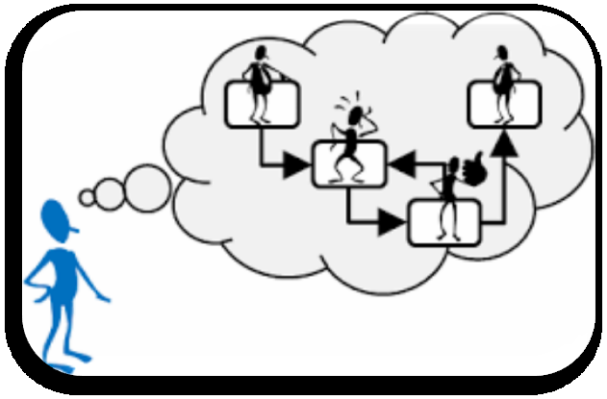
- 2. Sólo las instrucciones de carga y almacenamiento acceden a la memoria por datos.

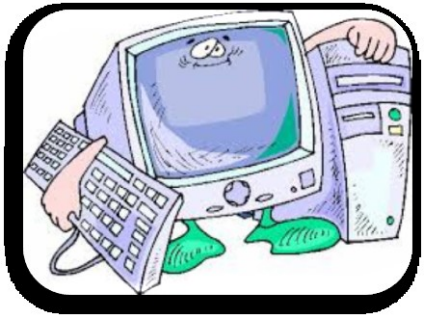
- Además estos procesadores suelen disponer de muchos registros de propósito general.El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria.

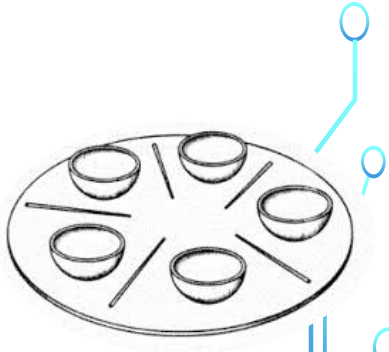
FAMILIA DE COMPUTADORAS

- Un conjunto de computadoras forma una familia cuando tienen la misma arquitectura y diferentes estructuras
- Surge el concepto de compatibilidad:
  - Programa escrito para un modelo, se ejecuta en otro modelo de la serie con diferencias en tiempo de ejecución
  - Sentido ascendente
- Características:
  - Repertorio de instrucciones similar o idéntico
  - Velocidad en incremento
  - N° de puertos E/S en incremento
  - Tamaño creciente de la memoria
  - Coste creciente

PROGRAMACIÓN CONCURRENTE







Programación Concurrente ¿Qué es?

Ejecutar multiples Actividades en paralelo o simultáneo

Concepto clave en la Ciencia de la Computación

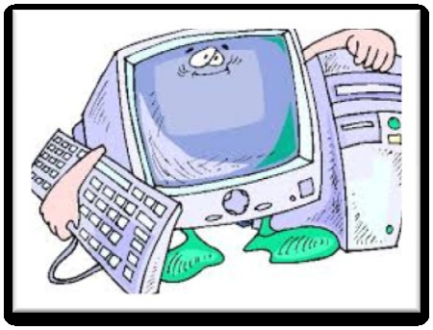
Programación Concurrente

Permite la interacción de distintos procesos al mismo tiempo

Influye en el diseño de hardware, sistemas operativos y multiprocesadores

CONCURRENCIA– ¿Dónde está?

En los accesos a diferentes páginas de un navegador web.
Varios usuarios accediendo a la misma página.

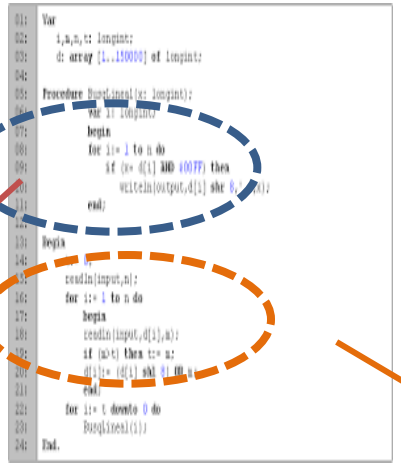


El Sistema Operativo de la computadora

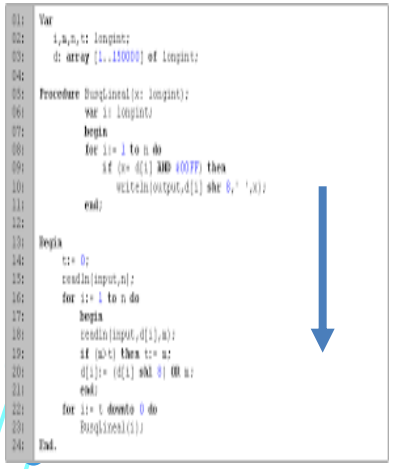
Un smartphone

Reserva de pasajes, hotel, etc

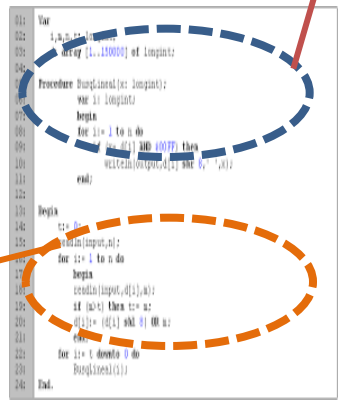
CONCURRENCIA– Escenarios



Paralelismo

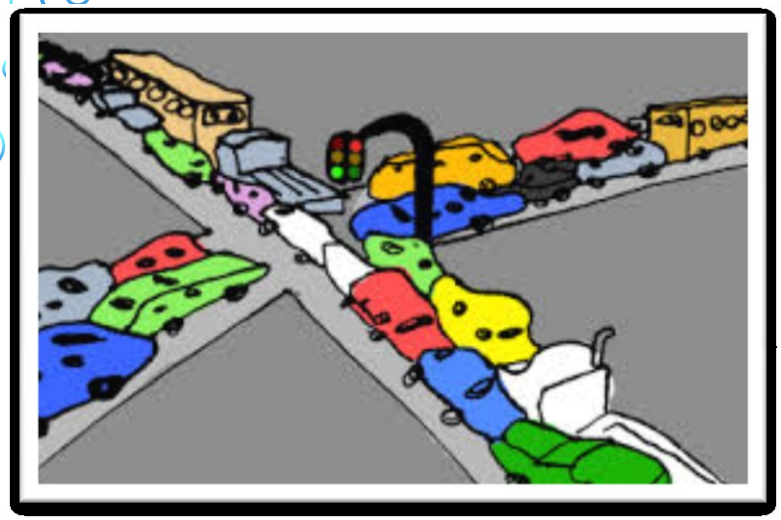


Programa Secuencial



Heterogeneidad

CONCURRENCIA– Escenarios



Se tiene:

- Automóviles = procesos que se ejecutan
- Carriles y rutas = múltiples procesadores
- Los automóviles deben sincronizarse para no chocar.

Objetivo: examinar los tipos de autos (procesos), rutas a recorrer (aplicaciones), caminos (hardware), y reglas (comunicación y sincronización).

CONCURRENCIA– Escenarios

Mundo Real ➔ Concurrencia

Concurrencia es la característica de los sistemas que indica que múltiples procesos/tareas pueden ser ejecutados al mismo tiempo y pueden cooperar y coordinarse para cumplir la función del sistema.

¿Los procesadores acompañaron?

CONCURRENCIA– Evolución de los procesadores

1950

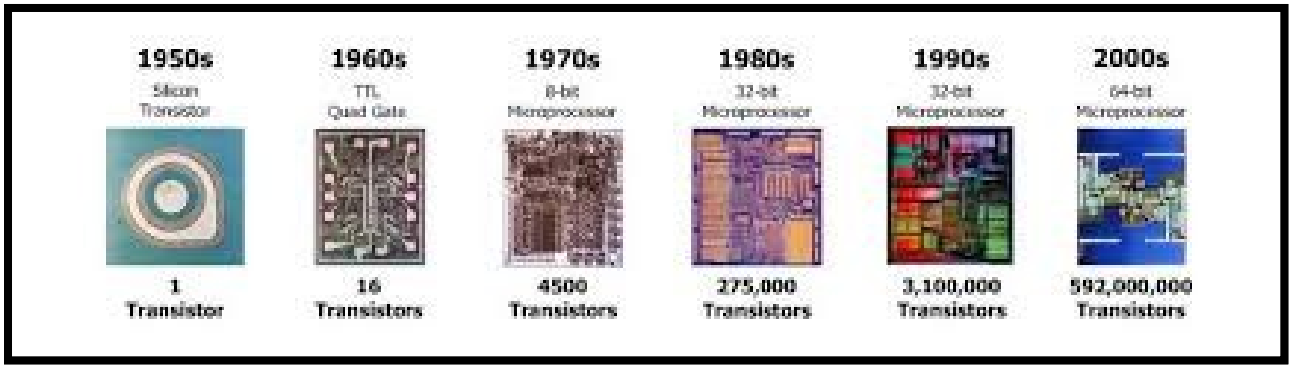
1

Transistor

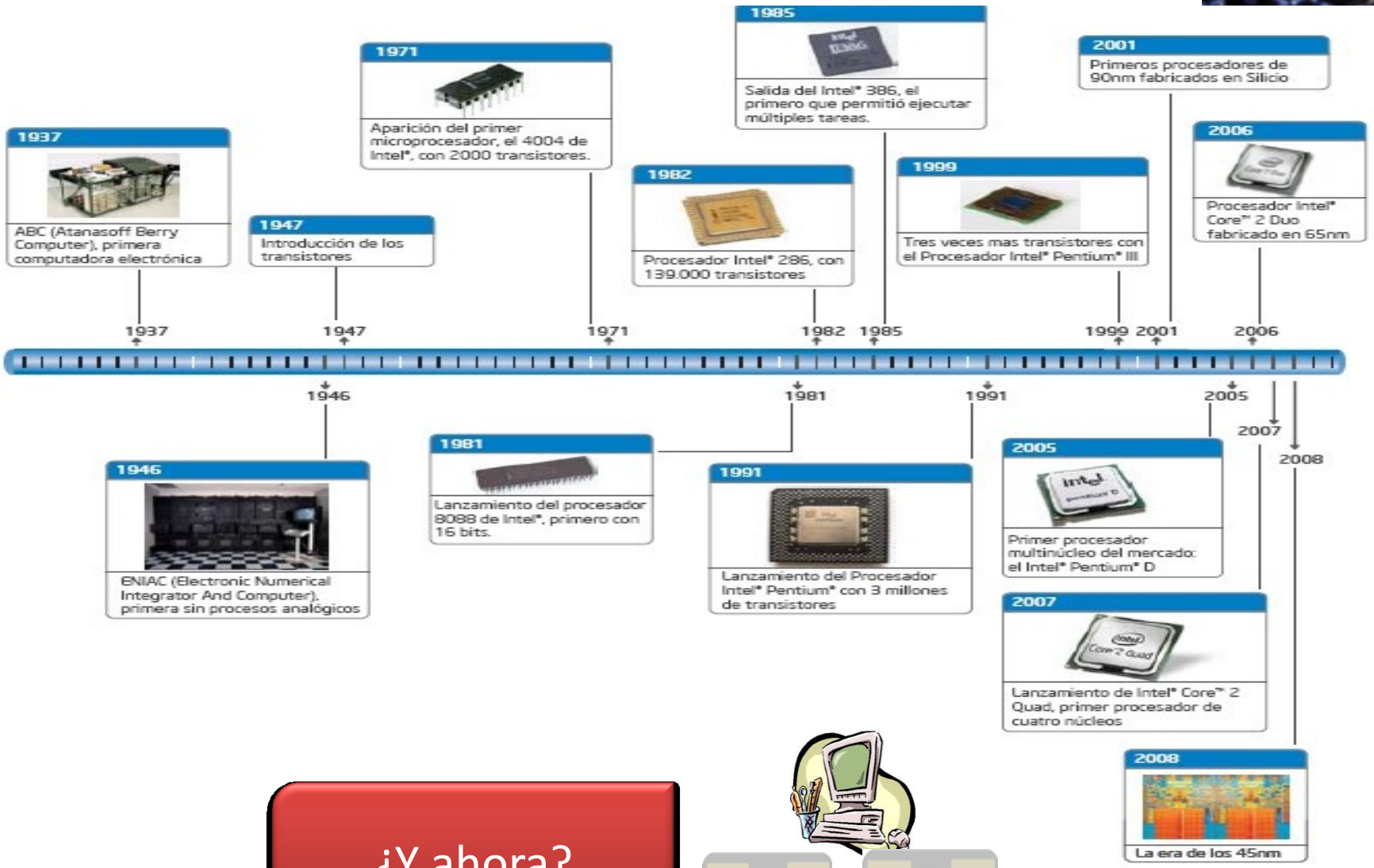
2000s

592.000.000

Transistores

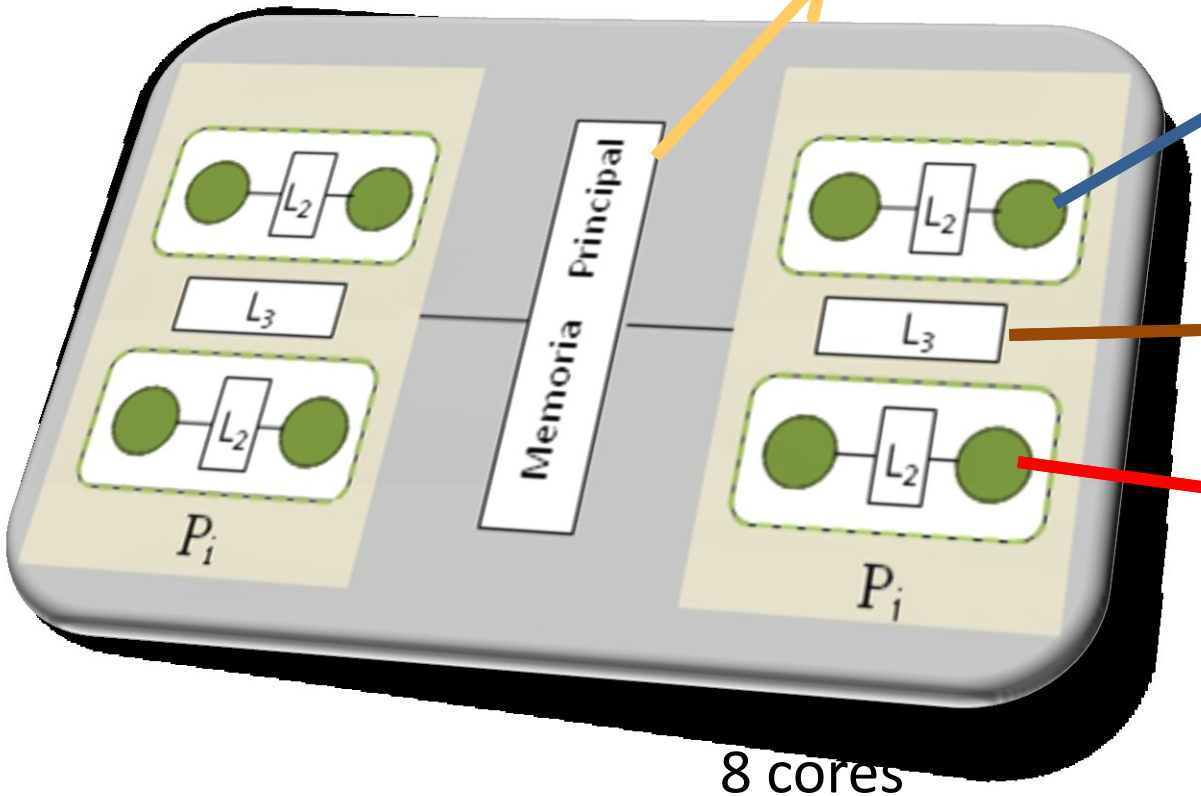


CONCURRENTE DE VON NEUMANN A MULTICORES



¿Y ahora?

Programación Concurrente - Evolución



Memoria principal

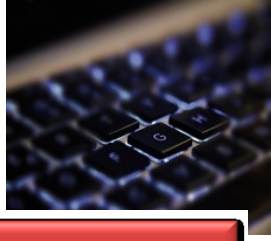
Procesador

Memoria cache Nivel L3

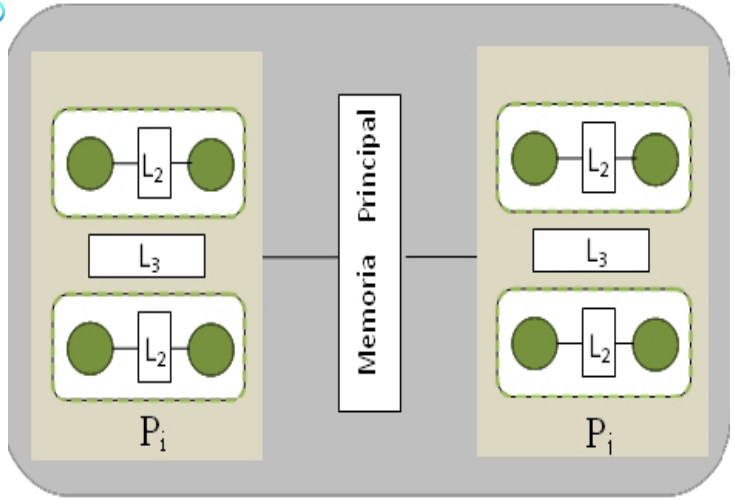
Memoria cache Nivel L2

8 cores

Programación Concurrente - Evolución



PARA PODER EXPLOTAR ESTE HARDWARE ES NECESARIO PROGRAMAR PROCESOS CONCURRENTES !



Preguntas

- ¿Cómo se maneja la concurrencia?
- ¿Cómo interactúan los procesos?
- ¿Cómo armamos un programa correcto?

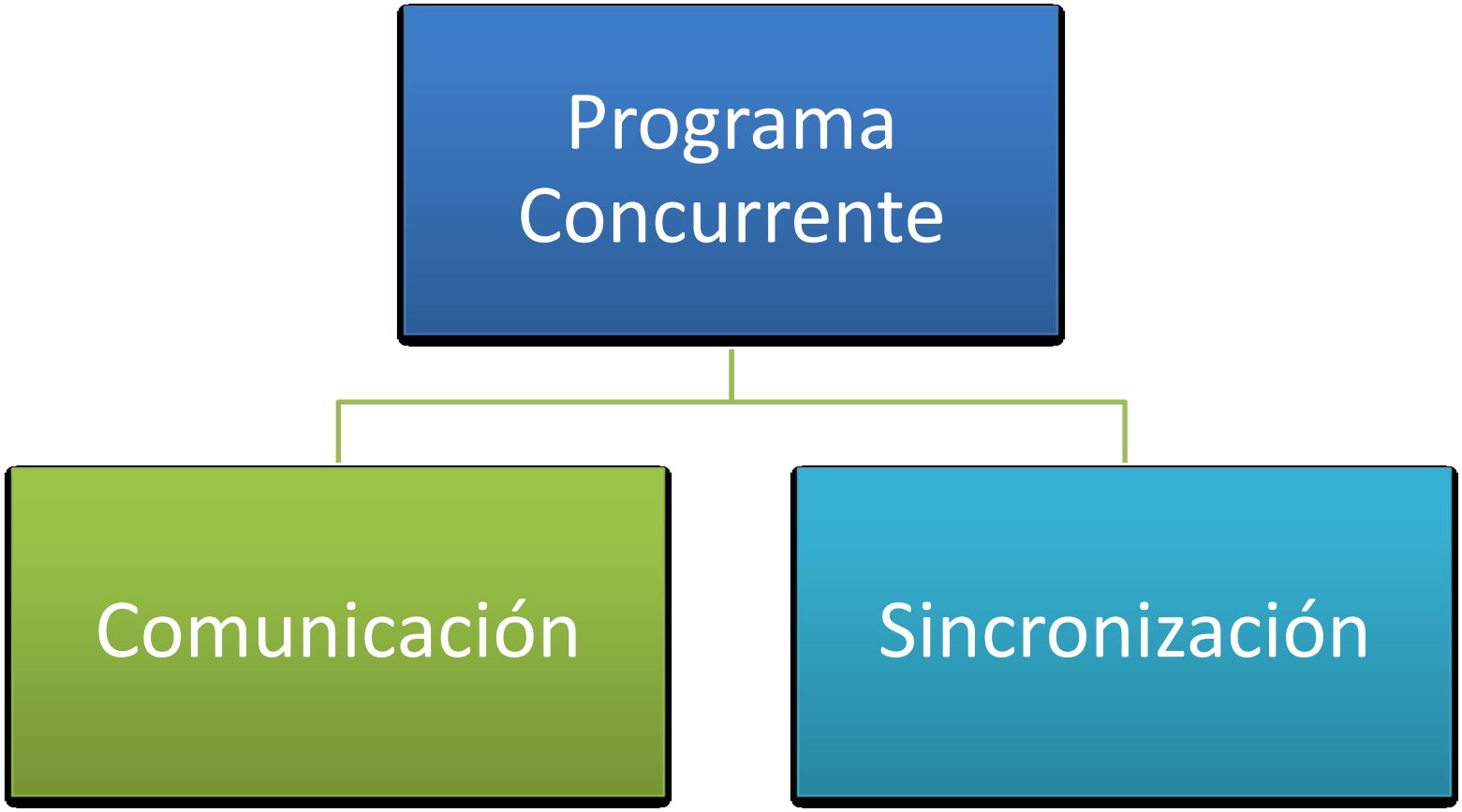
Programación Concurrente

Los procesos concurrentes tendrán necesidad de **comunicarse** información.

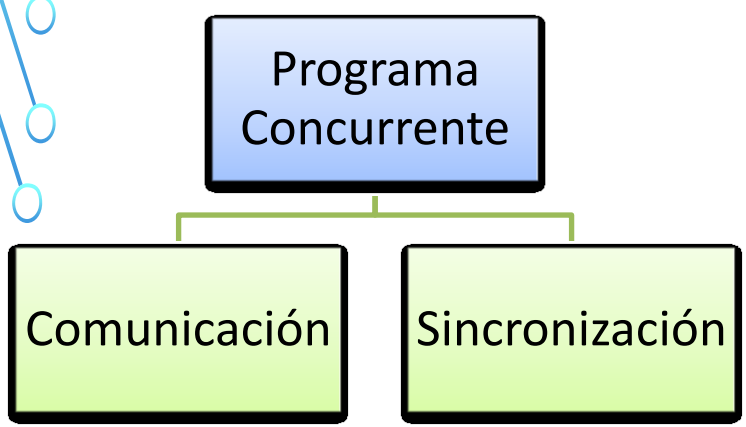
Además, será necesario en ocasiones detener a un proceso hasta que se produzca un determinado evento o se den ciertas condiciones → **sincronización**

Los lenguajes concurrentes deben proporcionar mecanismos de sincronización y comunicación.

Programación Concurrente



Programación Concurrente - Comunicación

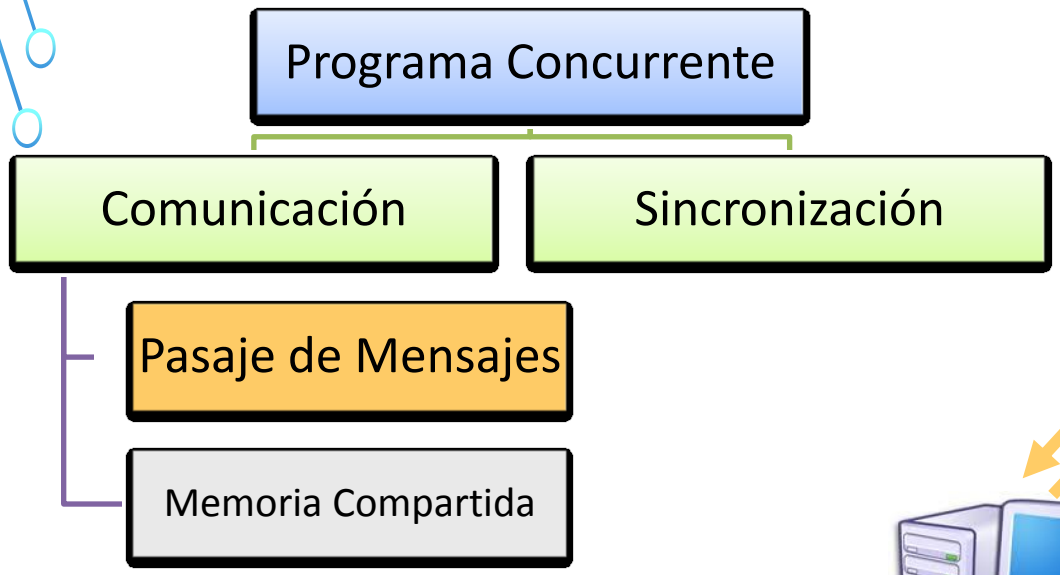


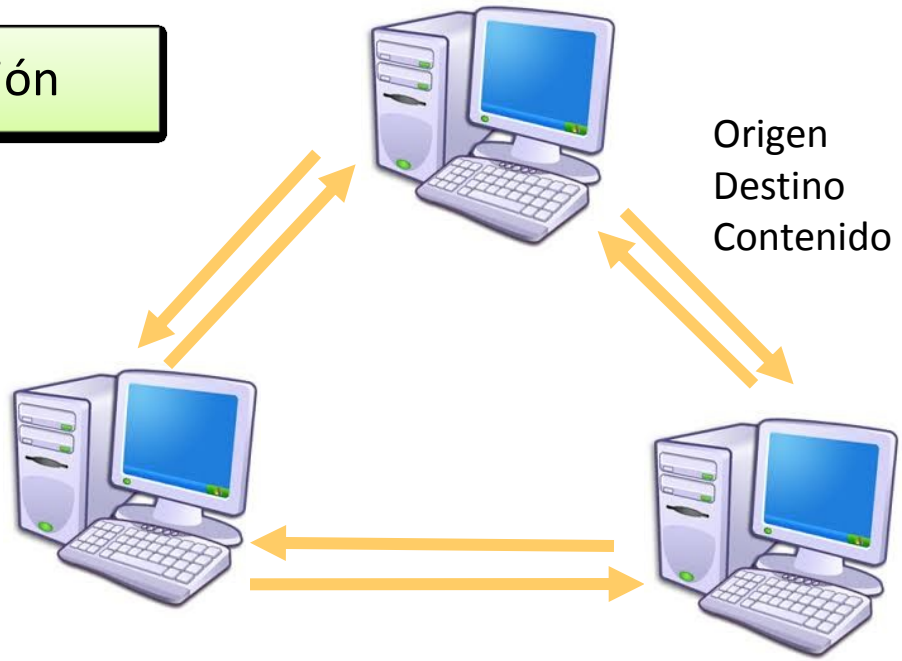
Pasaje de Mensaje

Memoria Compartida

Comunicación

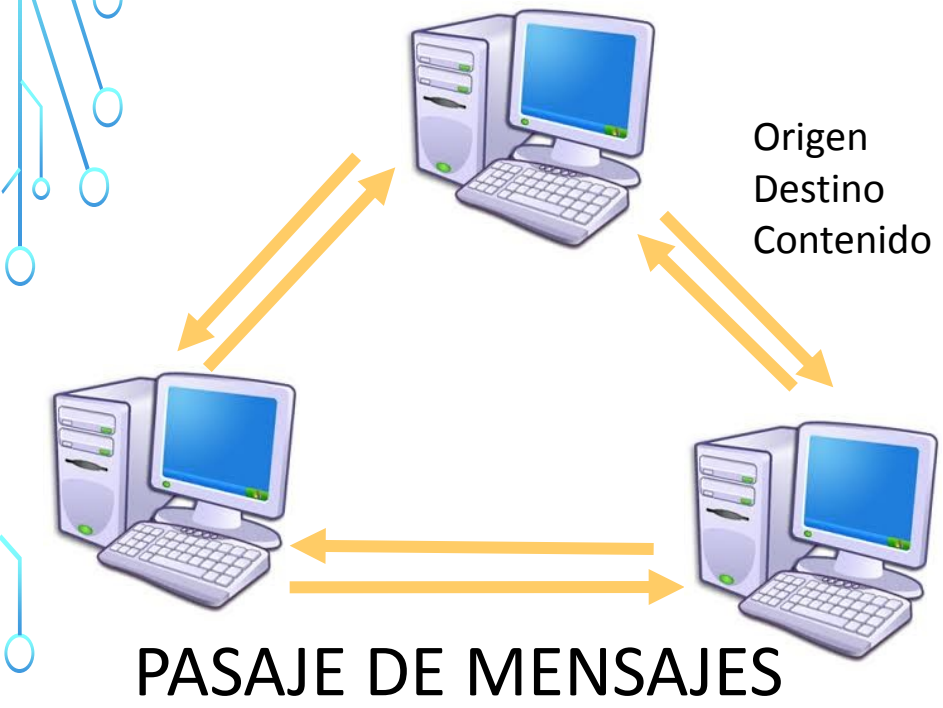
Programación Concurrente - Comunicación





PASAJE DE MENSAJES

Programación Concurrente - Comunicación



Origen
Destino
Contenido

PASAJE DE MENSAJES

ENVIAR y RECIBIR

- Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.
- También el lenguaje debe proveer un protocolo adecuado.
- Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuando deben trasmitir mensajes.

Programación Concurrente - Comunicación

EJEMPLOS

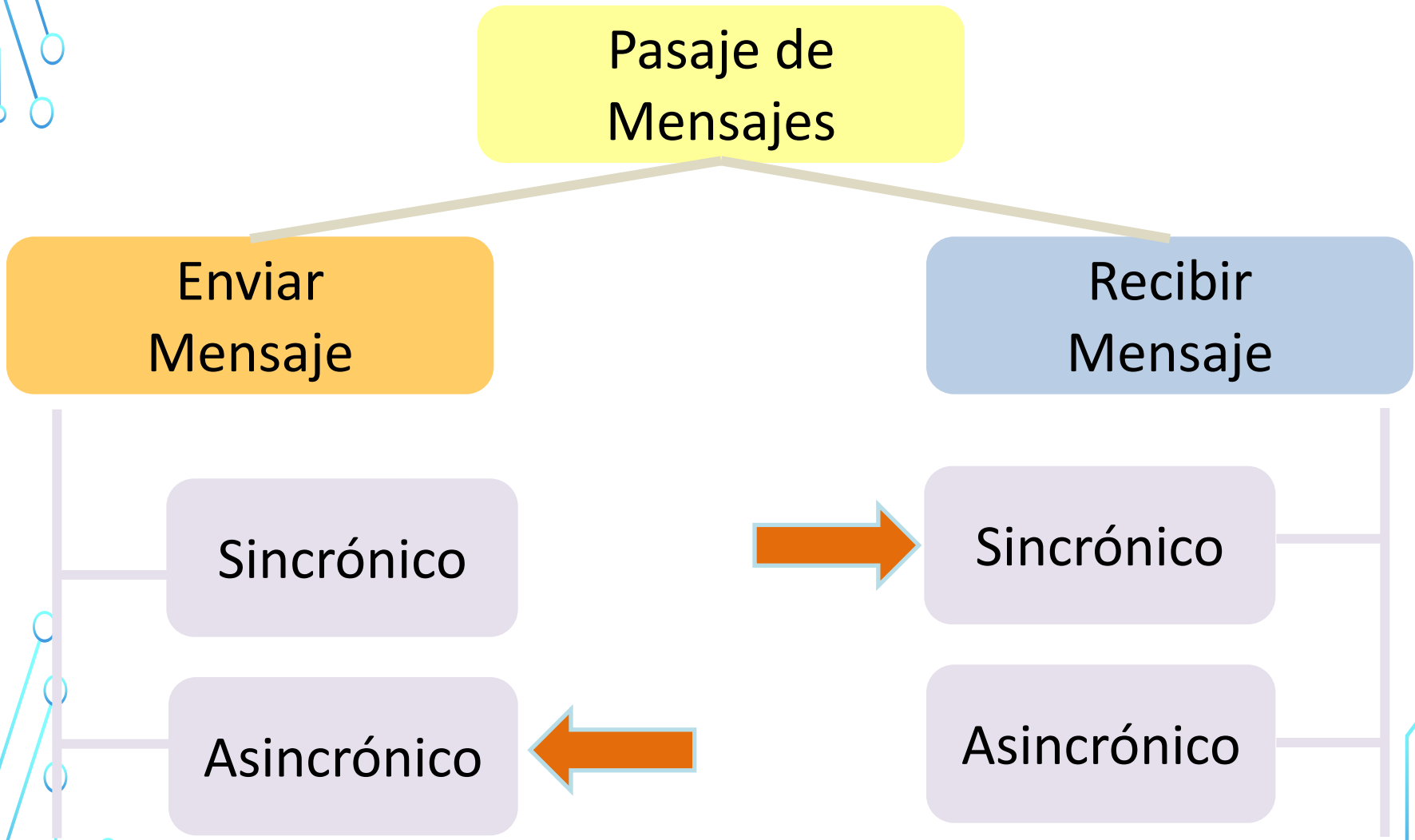
```
MPI_Send (buff, 128, MPI_CHAR,1 , 0, MPI_COMM_WORLD);
```

Nombre Operación — **Mensaje** — **Tipo Mensaje** — **Destino** — **Origen**

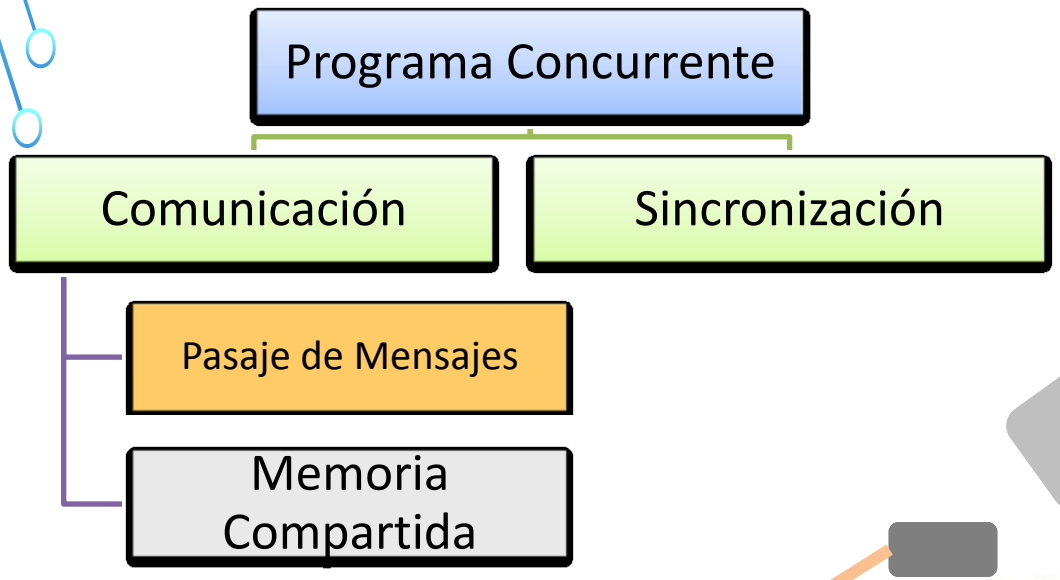
Mensaje — **Destino**

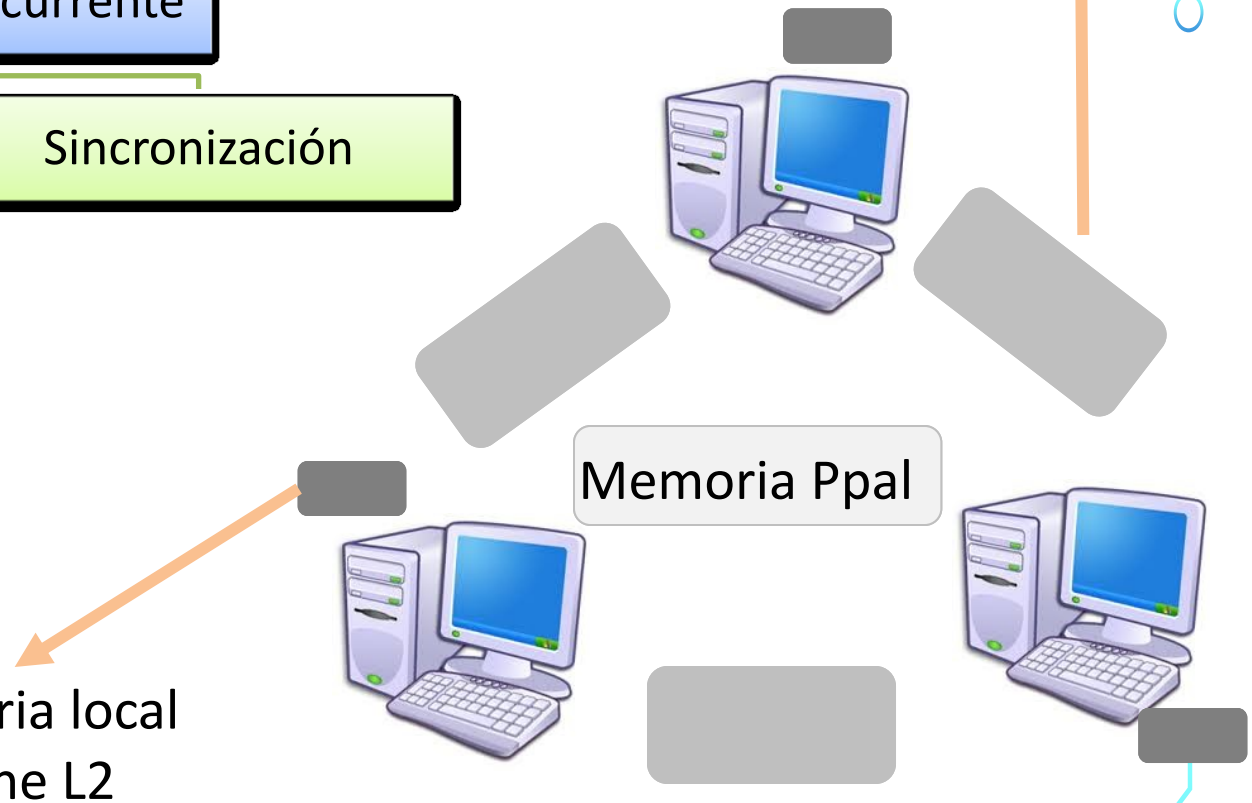
```
EnviarMensaje(dato, robot);
```

Programación Concurrente - Comunicación

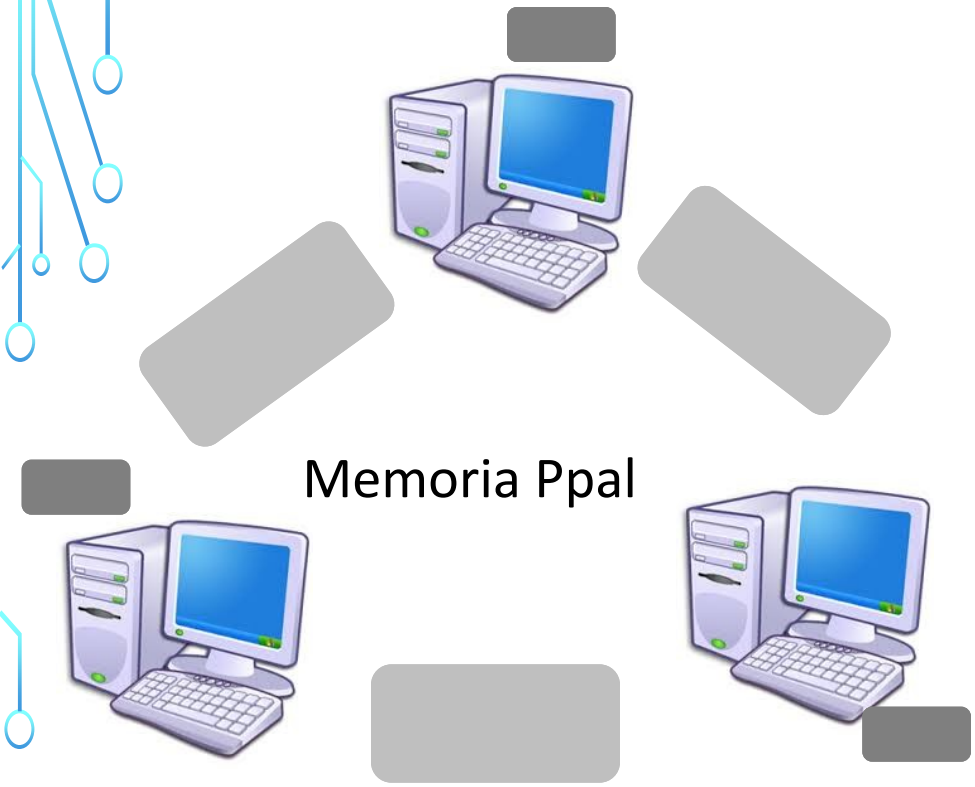


Programación Concurrente - Comunicación





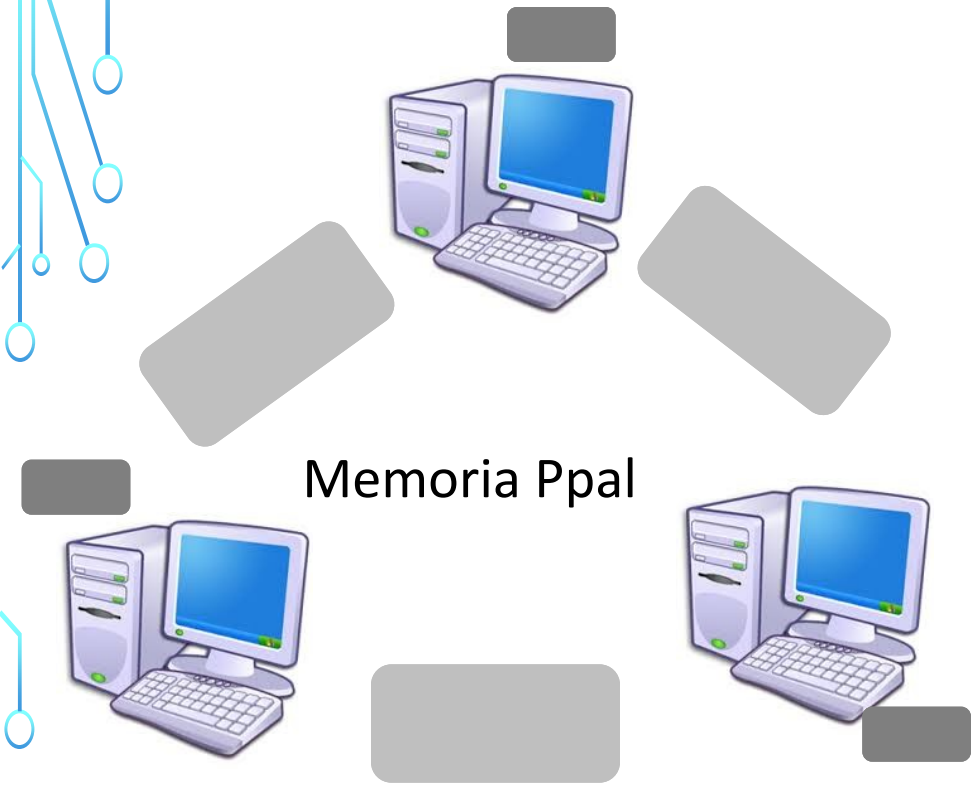
Programación Concurrente - Sincronización



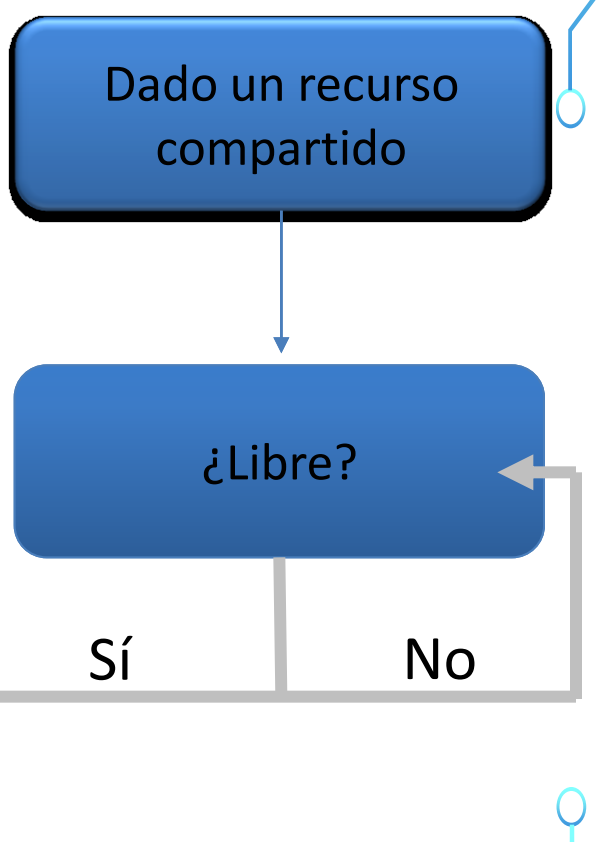
PROTEGER y LIBERAR

- Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
- Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a **bloquear y liberar** el acceso a la memoria.
- La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida.

Programación Concurrente - Sincronización



PROTEGER y LIBERAR



Bloqueo
Uso
Libero

SEMAFORO

Programación Concurrente - Sincronización

EJEMPLOS

P (variableSemaforo);

Protége un recurso

V (variableSemaforo);

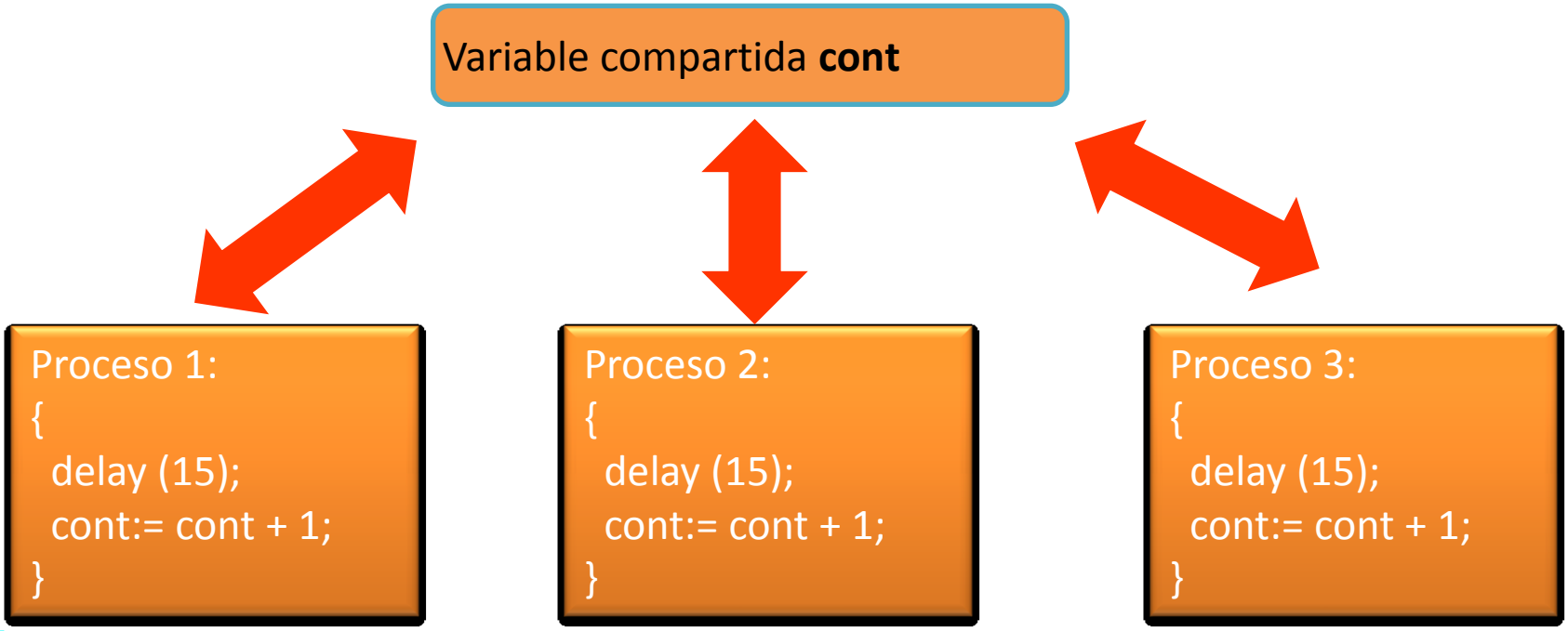
Libera un recurso

BloquearEsquina(av,calle);

LiberarEsquina(av,calle);

Programación Concurrente - Ejercicios

Ejercicio 1: supongamos que en un programa existen 3 procesos que quieren incrementar (en uno) cada 15 segundos el valor de una variable que comparten. ¿El código a continuación es correcto?



Programación Concurrente - Ejercicios

Ejercicio 1: supongamos que en un programa existen 3 procesos que quieren incrementar (en uno) cada 15 segundos el valor de una variable que comparten. El código a continuación es correcto?

Variable compartida **cont**

```
Proceso 1:
{
  P(variable);
  delay (15);
  cont:= cont + 1;
  V(variable);
}
```

```
Proceso 2:
{
  P(variable);
  delay (15);
  cont:= cont + 1;
  V(variable);
}
```

```
Proceso 3:
{
  P(variable);
  delay (15);
  cont:= cont + 1;
  V(variable);
}
```

Programación Concurrente - Ejercicios

Ejercicio 2: en un programa existen 3 procesos, un arreglo de longitud M, un valor N y se quiere calcular cuantas veces aparece el valor N en el arreglo. ¿El código a continuación es correcto?

Variable compartida **cont y arreglo**

```
Proceso 1:
{inf:=…; sup:= …;

  for i:= inf to sup do
    if v[i] = N then
       cont:= cont + 1;
}
```

```
Proceso 2:
{inf:=…; sup:= …;

  for i:= inf to sup do
    if v[i] = N then
       cont:= cont + 1;
}
```

```
Proceso 3:
{inf:=…; sup:= …;

  for i:= inf to sup do
    if v[i] = N then
       cont:= cont + 1;
}
```

Programación Concurrente - Ejercicios

Ejercicio 2: en un programa existen 3 procesos, un arreglo de longitud M y un valor N y se quiere calcular cuantas veces aparece el valor N en el arreglo. El código a continuación es correcto?

Variable compartida **cont y arreglo**

```
Proceso 1:
{inf:=…; sup:= …;

 for i:= inf to sup do
  P(variable);
  if v[i] = N then
     cont:= cont + 1;
  V(variable);
}
```

```
Proceso 2:
{inf:=…; sup:= …;

 for i:= inf to sup do
  P(variable);
  if v[i] = N then
     cont:= cont + 1;
  V(variable);
}
```

```
Proceso 3:
{inf:=…; sup:= …;

 for i:= inf to sup do
  P(variable);
  if v[i] = N then
     cont:= cont + 1;
  V(variable);
}
```

Programación Concurrente - Ejercicios

Ejercicio 2: en un programa existen 3 procesos, un arreglo de longitud M y un valor N y se quiere calcular cuantas veces aparece el valor N en el arreglo. ¿El código a continuación es correcto?

Variable compartida **cont y arreglo**

```
Proceso 1:
{inf:=…; sup:= …;

 for i:= inf to sup do
  if v[i] = N then
    P(variable);
     cont:= cont + 1;
    V(variable);
}
```

```
Proceso 2:
{inf:=…; sup:= …;

 for i:= inf to sup do
  if v[i] = N then
    P(variable);
     cont:= cont + 1;
    V(variable);
}
```

```
Proceso 3:
{inf:=…; sup:= …;

 for i:= inf to sup do
  if v[i] = N then
    P(variable);
     cont:= cont + 1;
    V(variable);
}
```