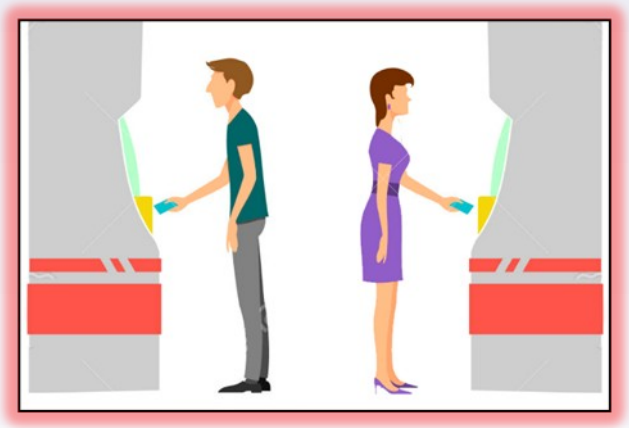
Módulo 4
Concurrente

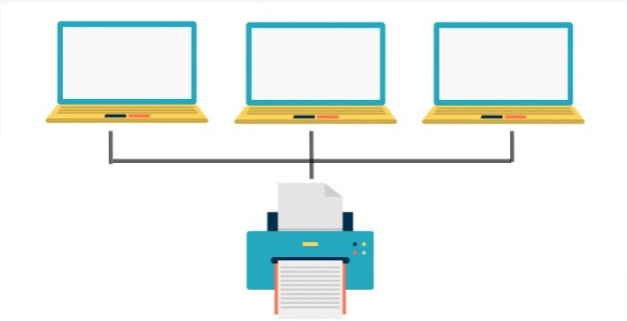
- Clase 1: Conceptos iniciales

ORGANIZACIÓN

- **Clase 1 -** Conceptos de Concurrencia
- **Clase 2 -** Ambiente CMRE
- **Clase 3 -** Problemas de Memoria Distribuida
- **Clase 4 -** Problemas de Memoria Compartida
- **Clase 5 -** Trabajo Final

PROGRAMACIÓN CONCURRENTE





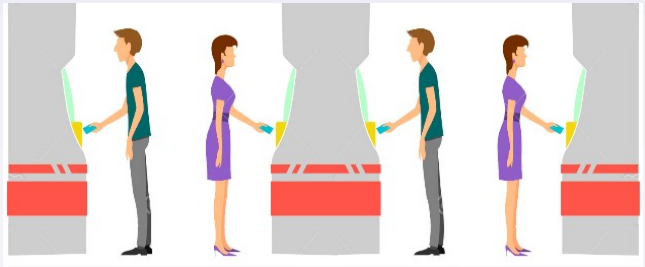
MOTIVACIÓN

¿Qué es la programación concurrente?

MOTIVACIÓN

¿Dónde está la concurrencia?

Diferentes páginas en un navegador web y varios usuarios accediendo a la misma página.



Varias personas accediendo a distintas o a la misma cuenta

El Sistema Operativo de la computadora

Reserva de pasajes, hotel, etc

Un smartphone

CONCEPTOS DE PROGRAMACIÓN CONCURRENTE

¿Nuevos conceptos?

Programa

Conjunto de sentencias/instrucciones que se ejecutan secuencialmente. Concepto estático.

Proceso

Es un programa en ejecución. Concepto dinámico. Es una instancia de ese programa en ejecución tiene su propio espacio en memoria y recursos asignados.

CONCEPTOS DE PROGRAMACIÓN CONCURRENTE

Hilo: Un hilo es una unidad de ejecución dentro de un proceso. Los hilos comparten el mismo espacio de memoria y recursos del proceso, lo que les permite comunicarse y cooperar entre sí.

Concurrencia: Se refiere a la capacidad de ejecutar múltiples tareas de manera simultánea o en superposición. Los procesos y los hilos son las entidades principales en la programación concurrente.

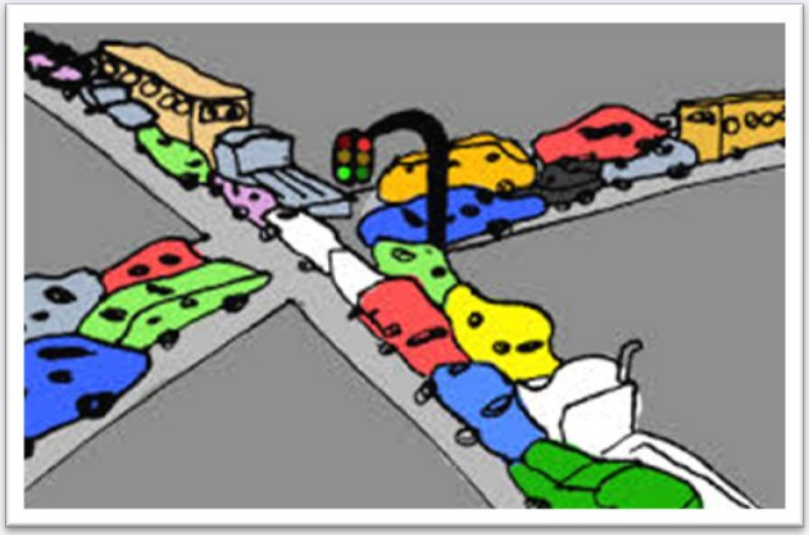
Sincronización: La sincronización se utiliza para coordinar y controlar el acceso concurrente a recursos compartidos. Permite que los hilos se comuniquen y se asegura de que no ocurran condiciones de carrera (cuando dos o más hilos acceden simultáneamente a un recurso compartido).

CONCEPTOS DE PROGRAMACIÓN CONCURRENTE

Bloqueo: El bloqueo es una técnica que se utiliza para evitar que otros hilos accedan a un recurso mientras otro hilo lo está utilizando. Esto garantiza la coherencia y la integridad de los datos compartidos.

Exclusión mutua. Es un principio que garantiza que un recurso compartido solo puede ser accedido por un solo hilo a la vez. La exclusión mutua se logra mediante el uso de bloqueos y semáforos.

Ejemplo



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), trayecto a recorrer (aplicaciones), caminos (hardware), y reglas (comunicación y sincronización).

CONCURRENCIA

Escenarios

Programa Secuencial

```
{PROGRAMA PRINCIPAL}
var
   v: vector;
   dimL : dim;

begin

     cargarVector(v,dimL);

     writeln('Nros almacenados:
');
     imprimirVector(v, dimL);

     readln;
end.
```

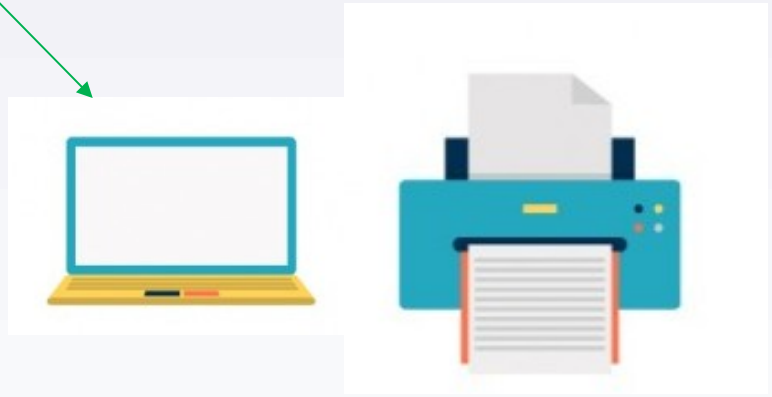
CONCURRENCIA

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
```

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
End;
```

Escenarios

Programa Concurrente



Un **mismo programa** ejecutándose **en la misma máquina** tratando de acceder a un recurso compartido (ej. Impresora)

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
```

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
End;
```

CONCURRENCIA

Escenarios

Programa Paralelo

Un **mismo programa** ejecutándose en varias máquinas tratando de acceder a un **recurso compartido** (ej. Impresora)

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
```

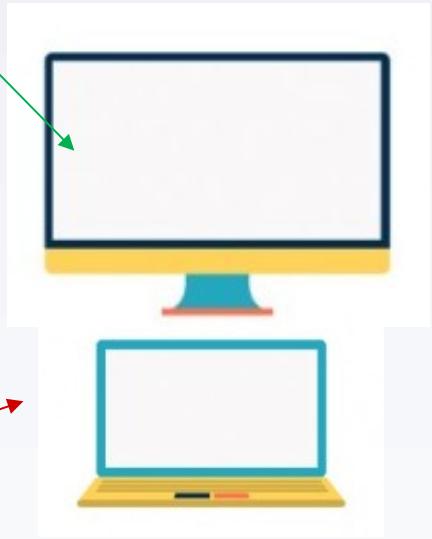
```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
End;
```

CONCURRENCIA

Escenarios

Heterogeneidad

Programa Paralelo



Un **mismo programa** ejecutándose en **varias máquinas** de características **diferentes** tratando de acceder a un recurso compartido (ej. Impresora)

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
```

```
Procedure imprimirVector ( var vec: vector; var dimL: dim );
var
   i: dim;
begin
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     write (' ');
     for i:= 1 to dimL do begin
        if(vec[i] < 9)then
            write ('0');
        write(vec[i], ' | ');
     end;
     writeln;
     for i:= 1 to dimL do
         write ('-----');
     writeln;
     writeln;
End;
```

CONCURRENCIA

Escenarios

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.

CAMBIO EN EL SOFTWARE

¿Y el hardware?

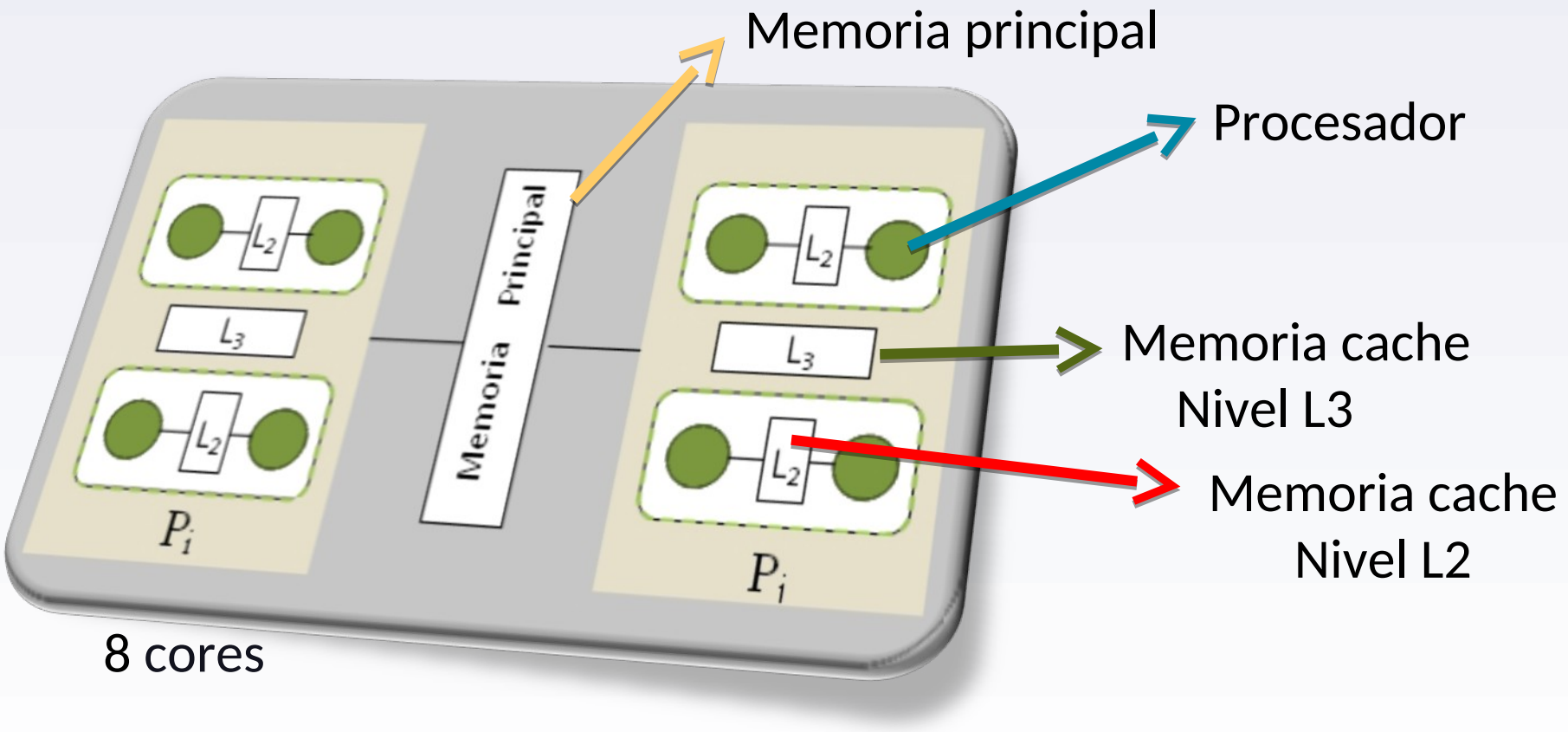
Escenarios

1 sólo núcleo de procesamiento
(1980)

Se agregan placas que agrupan núcleos (2, 4, 8, etc.)
(2000)

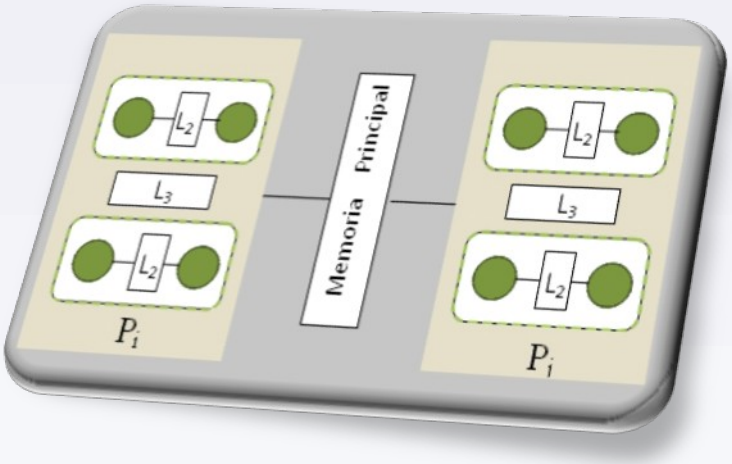
La 1era computadora del Top500
2.3 millones de núcleos!!!!
(2019)

Evolución


Memoria principal
Procesador
Memoria cache
Nivel L3
Memoria cache
Nivel L2
Memoria Principal
L_2
L_3
P_i
8 cores

CONCURRENCIA

Evolución



PARA PODER EXPLOTAR ESTE HARDWARE ES NECESARIO PROGRAMAR PROCESOS CONCURRENTES !

Comunicación y sincronización

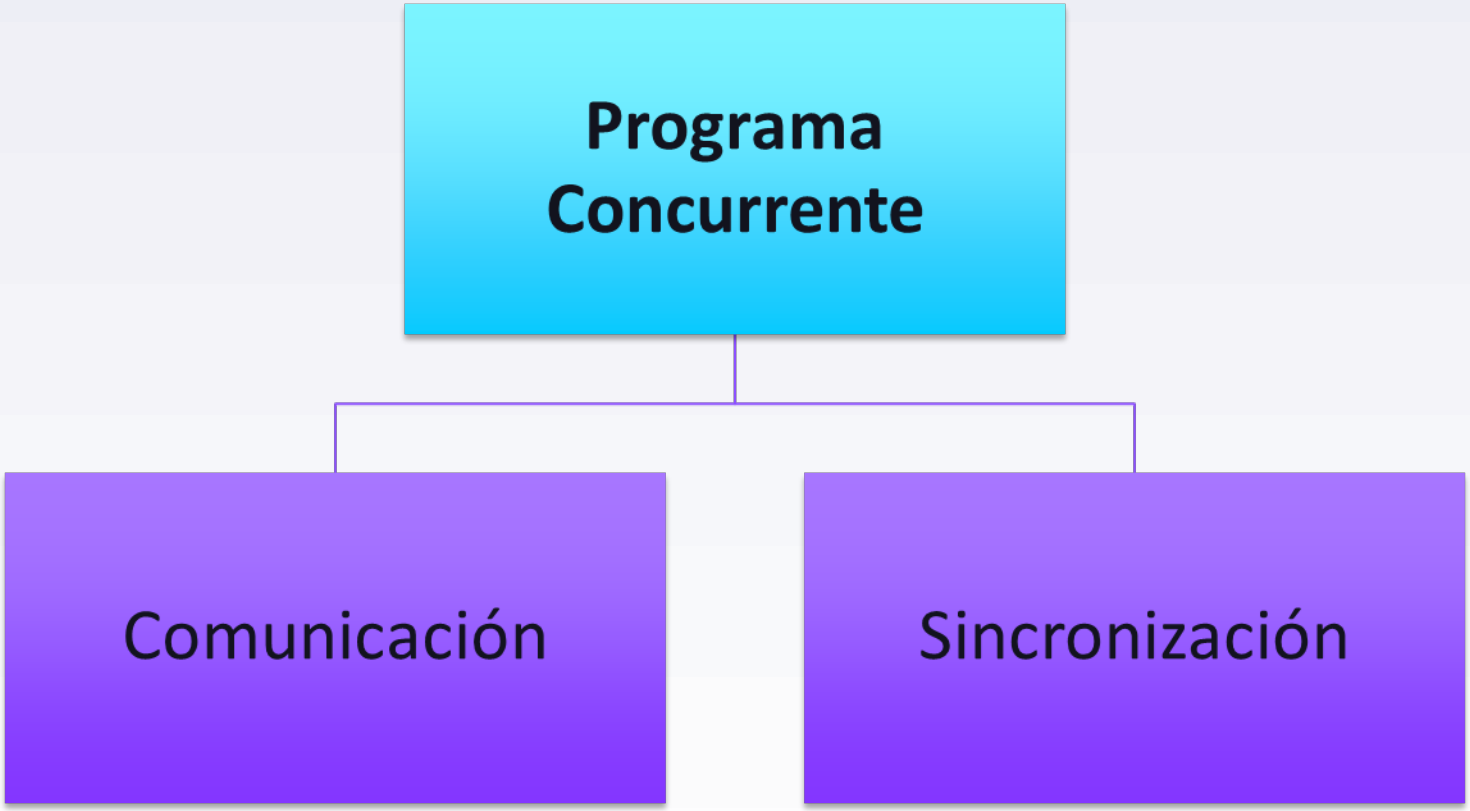
PROGRAMACIÓN CONCURRENTE

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

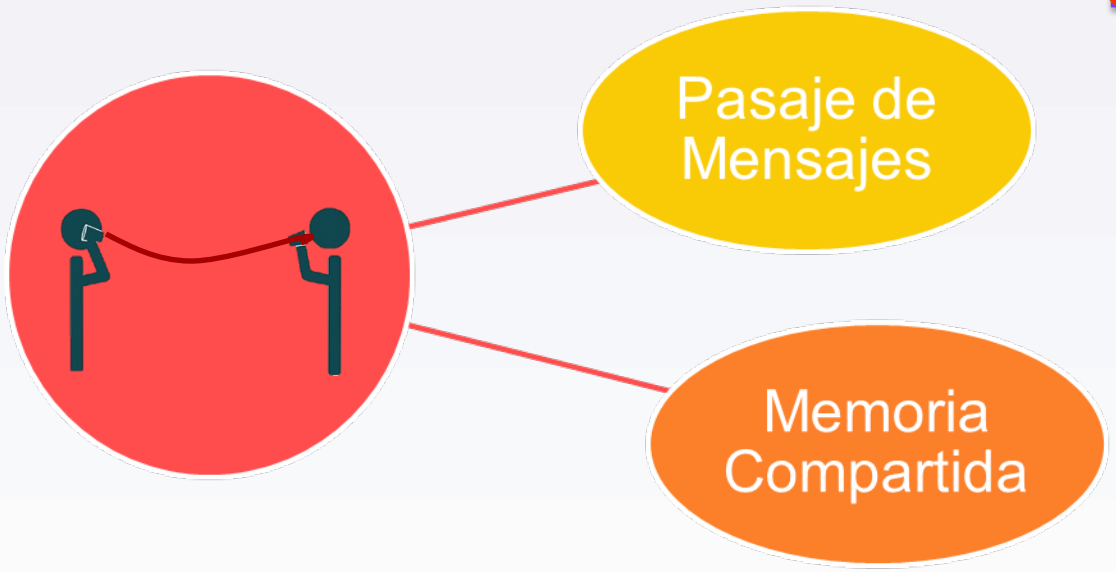
Además, será necesario en ocasiones detener a un proceso hasta que se produzca un determinado evento o se den ciertas condiciones → **sincronización**

Los lenguajes concurrentes deben proporcionar mecanismos de sincronización y comunicación.

PROGRAMACIÓN CONCURRENTE

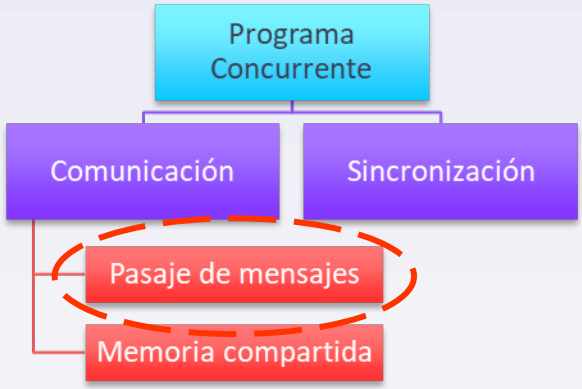


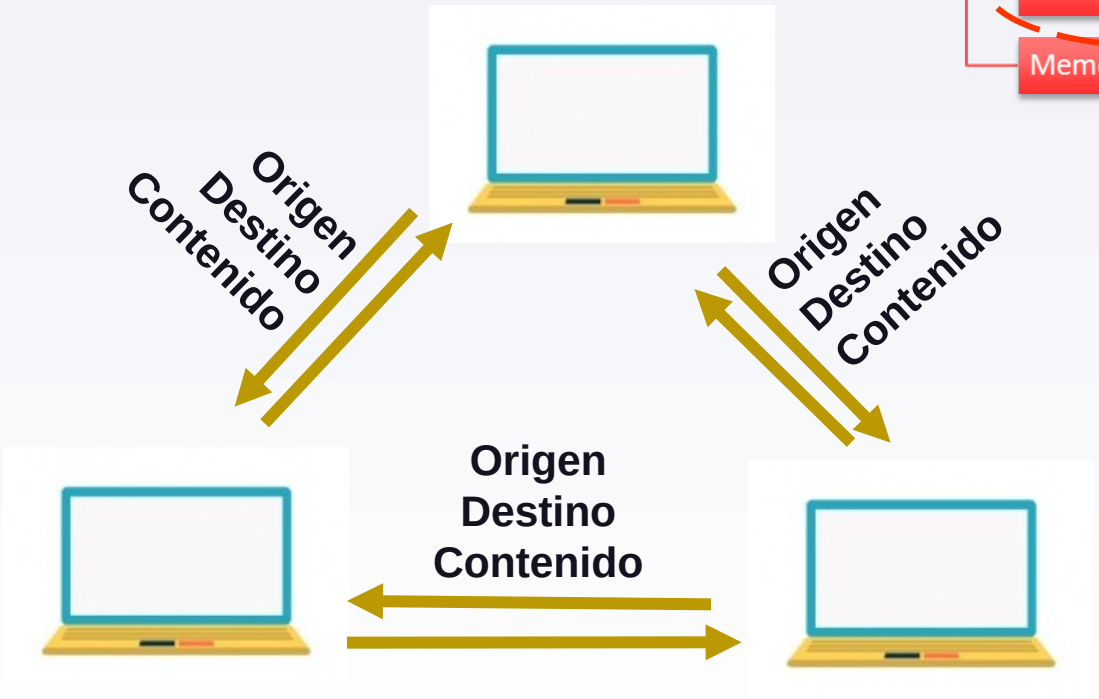
Comunicación



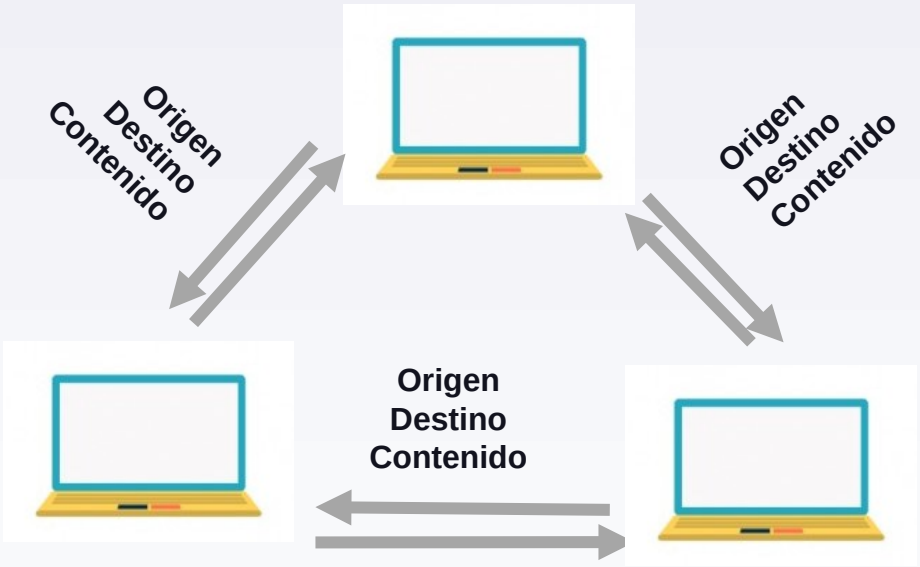
Pasaje de mensajes

Comunicación:
Pasaje de mensajes





Comunicación Pasaje de mensajes



- 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 transmitir mensajes.

ENVIAR y RECIBIR

Comunicación: Pasaje de mensajes - Ejemplos

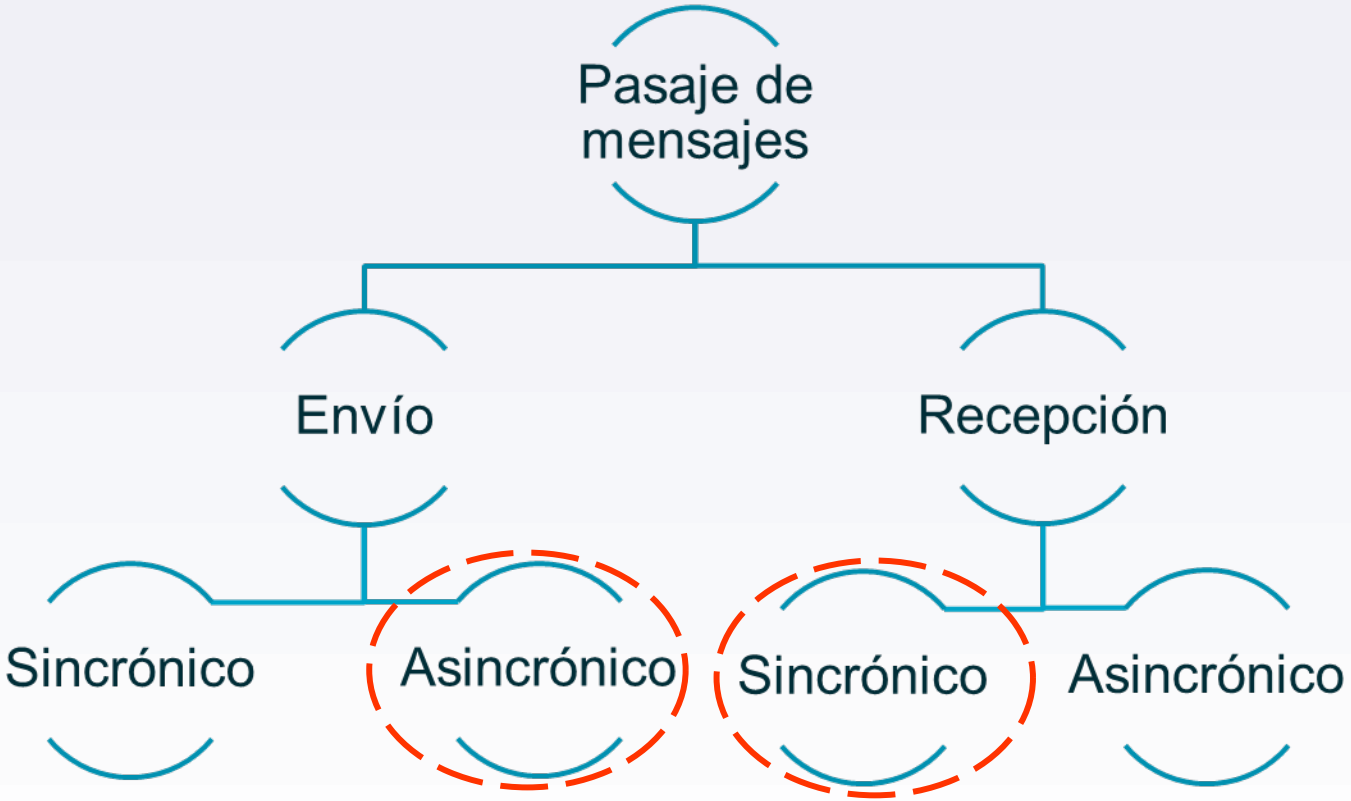
```
MPI_Send (buff, 128, MPI_CHAR,1 , 0,
MPI_COMM_WORLD);
```

Nombre Operación | Mensaje | Tipo Mensaje | Destino | Origen

Mensaje | Destino

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

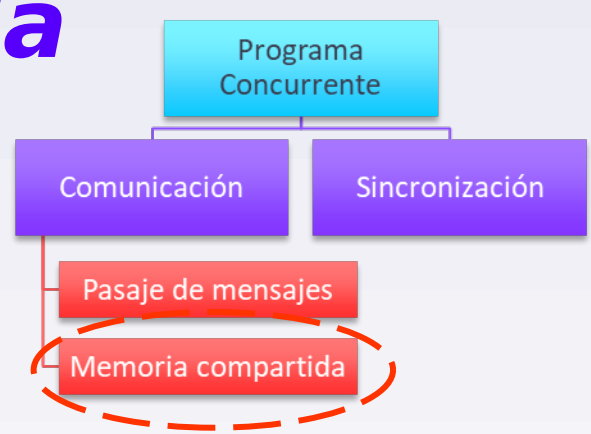
Pasaje de mensajes - Ejemplos

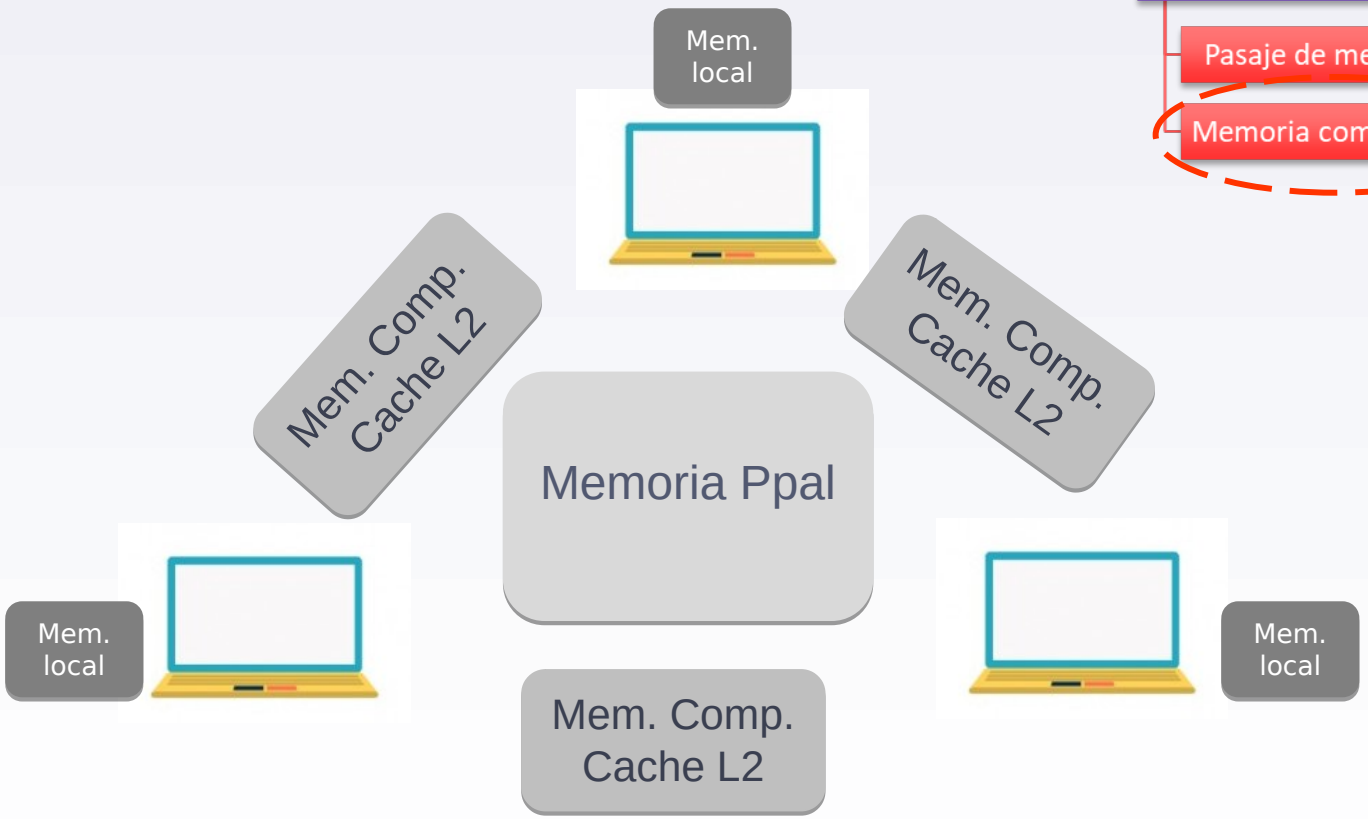


En CMRE
ROBOT

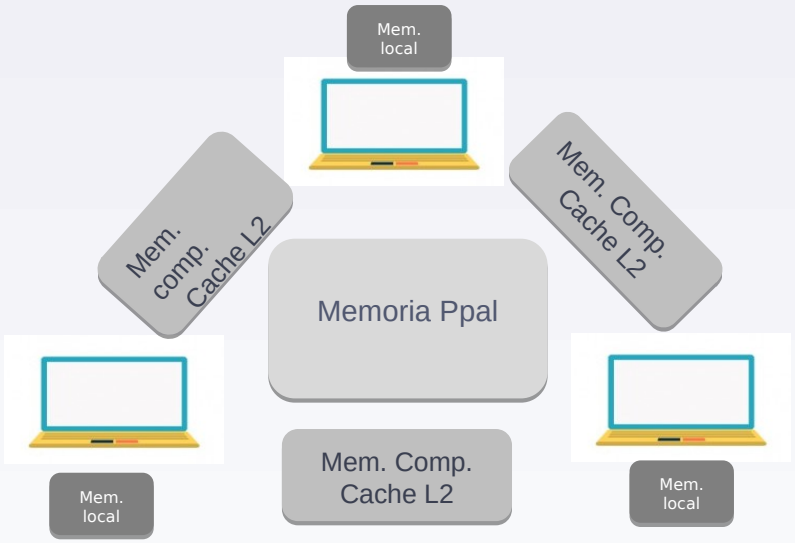
Memoria compartida

Comunicación: Memoria compartida





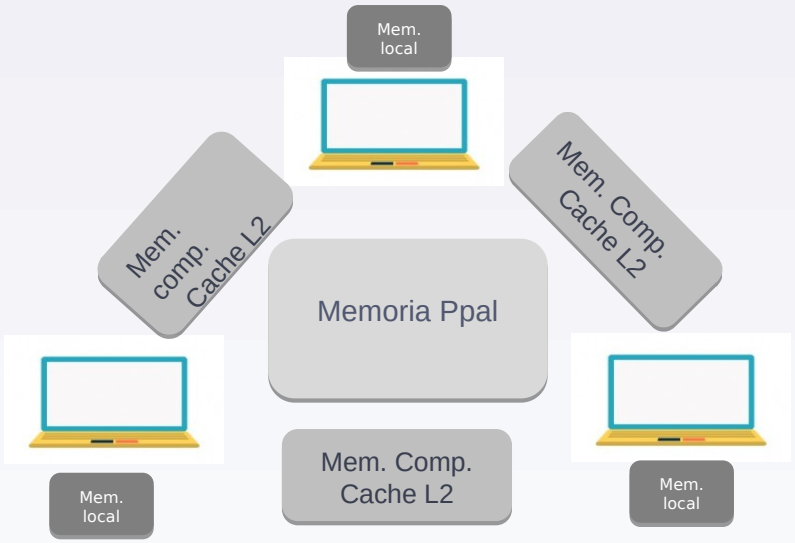
Comunicación: Memoria compartida

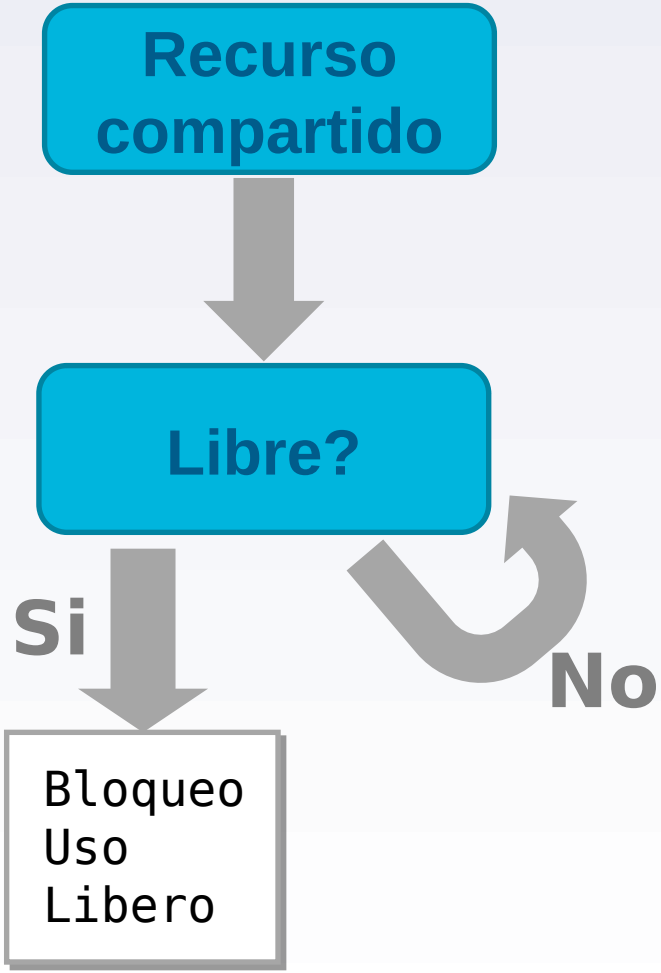


- Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
- Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a **bloquear y liberar** el acceso a la memoria.
- La solución más elemental es una variable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

BLOQUEAR Y LIBERAR

Comunicación: Memoria compartida





SEMÁFORO

Comunicación Memoria compartida - Ejemplos

```
P (variableSemaforo);
```

Protége un recurso

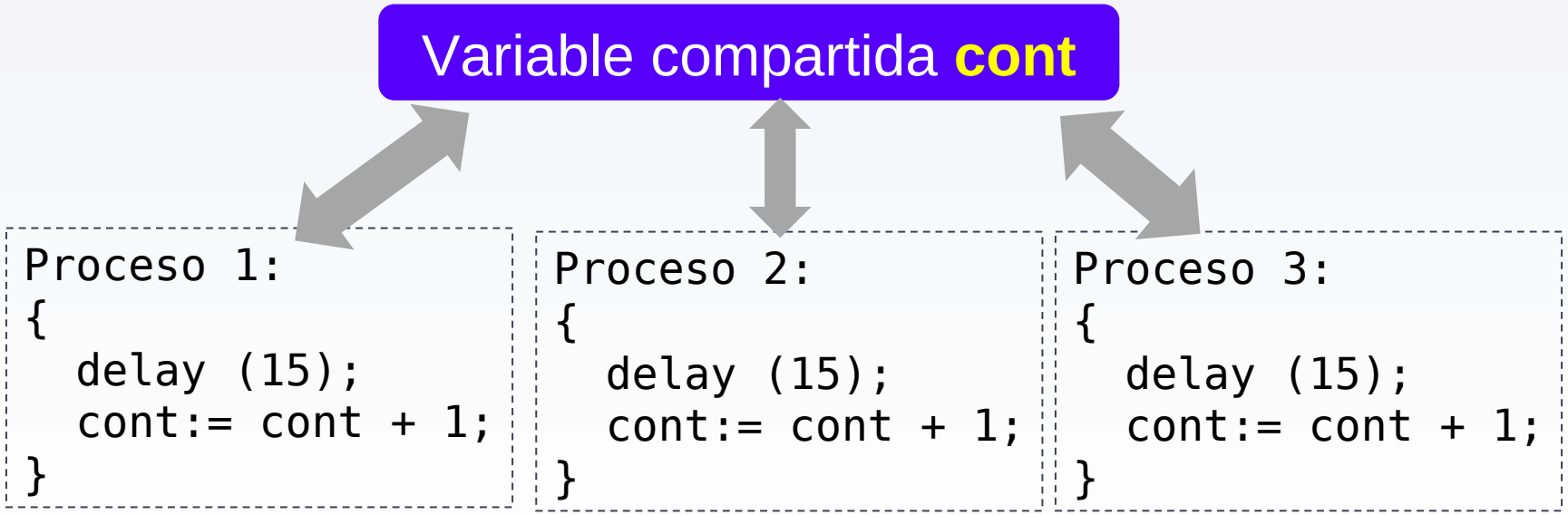
```
V (variableSemaforo);
```

Libera un recurso

```
BloquearEsquina(av,calle);

LiberarEsquina(av,calle);
```

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?



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?

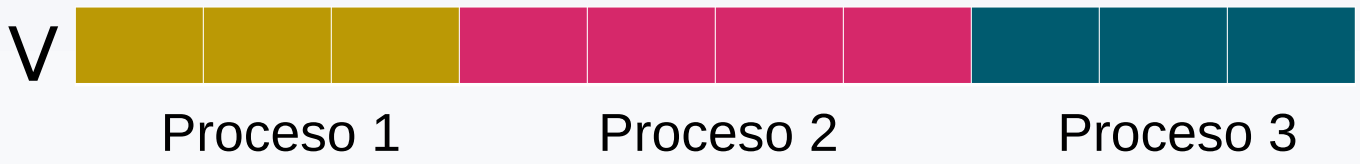

```
Proceso 1:
{
  P(variable)
  delay (15);
  cont:= cont + 1;
  V(variable)
}
```

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

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

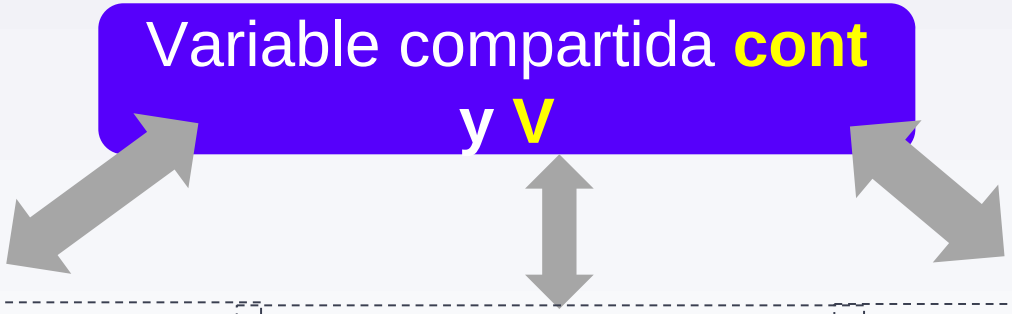
Ejercicio 2

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



Ejercicio 2

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



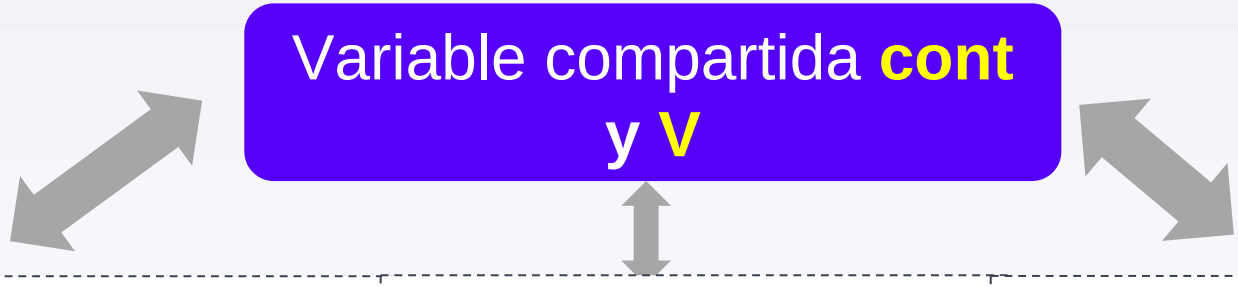

```
Proceso 1:
{inf:=…; sup:= …;

  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;
}
```

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?




```
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

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 **V**

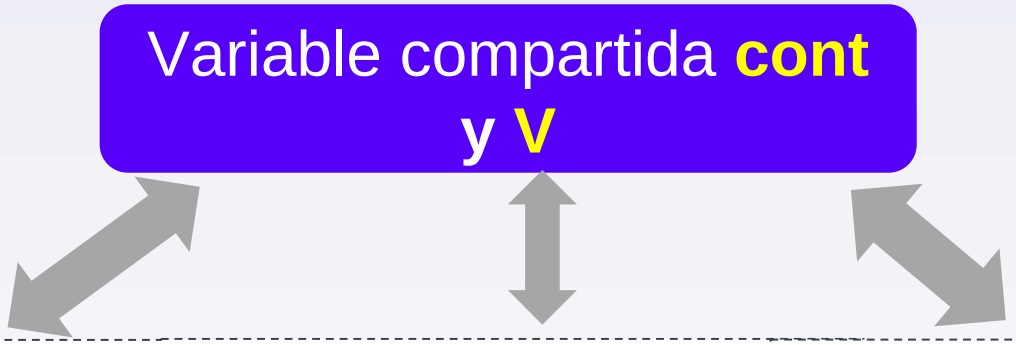
```
Proceso 1:
{inf:=…; sup:= …;
  for i:= inf to sup do
    if v[i] = N then
      P(variable)
      cont:= cont + 1;
      V(variable)
}
```

```
Proceso 2:
{inf:=…; sup:= …;
  for i:= inf to sup do
    if v[i] = N then
      P(variable)
      cont:= cont + 1;
```

```
Proceso 3:
{inf:=…; sup:= …;
  for i:= inf to sup do
    if v[i] = N then
      P(variable)
      cont:= cont + 1;
      V(variable)
}
```

Se puede mejorar más aún?

Ejercicio 2




```
Proceso 1:
{inf:=…; sup:= …; car

  for i:= inf to sup
    if v[i] = N then
      cant := cant +

  P(variable)
  cont:= cont + cant;
  V(variable)
}
```

```
Proceso 2:
{inf:=…; sup:= …; car

  for i:= inf to sup
    if v[i] = N then
      cant := cant +

  P(variable)
  cont:= cont + cant;
  V(variable)
}
```

```
Proceso 3:
{inf:=…; sup:= …; cant

  for i:= inf to sup do
    if v[i] = N then
      cant := cant + 1

  P(variable)
  cont:= cont + cant;
  V(variable)
}
```