Sincronización con Espera Activa

PROGRAMACIÓN CONCURRENTE – TEMA 2

Universidad
Rey Juan Carlos

Sincronización con Espera Activa

- **Introducción**
- Sincronización Condicional
- Exclusión Mutua
- Espera Activa vs Espera Pasiva
- Conclusiones

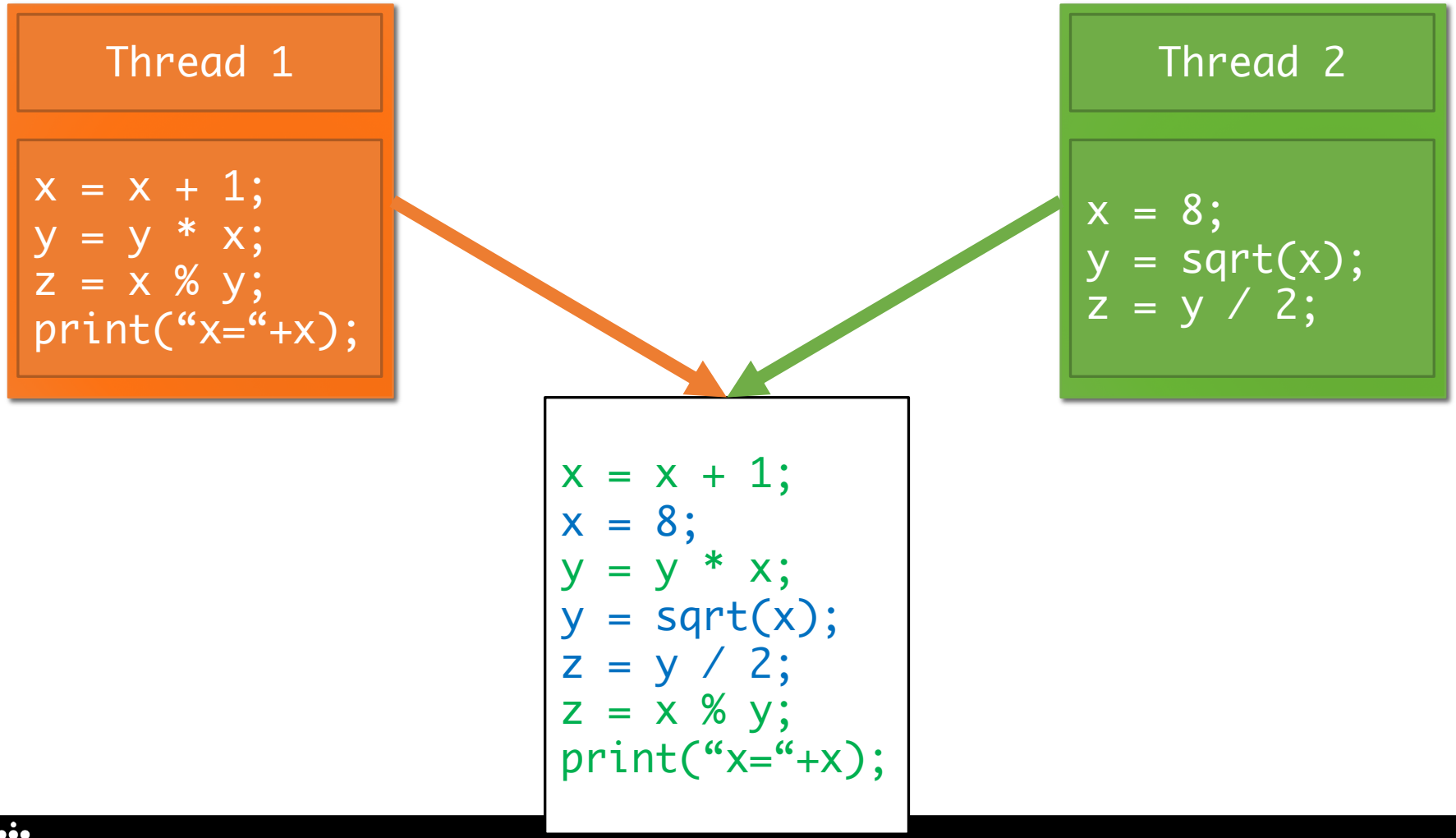
- Modelo de Memoria Compartida
  - Los procesos pueden acceder a una memoria común
  - Existen variables compartidas que varios procesos pueden leer y escribir

¿Qué ocurre si dos procesos leen o escriben de forma simultánea en la misma variable?

- Modelo de Memoria Compartida
  - La abstracción de la programación concurrente nos indica que todas las instrucciones atómicas de cada proceso se intercalan en una única secuencia
  - La lectura y la escritura de atributos de tipo simple (`boolean`, `int`, `char`…) son instrucciones atómicas

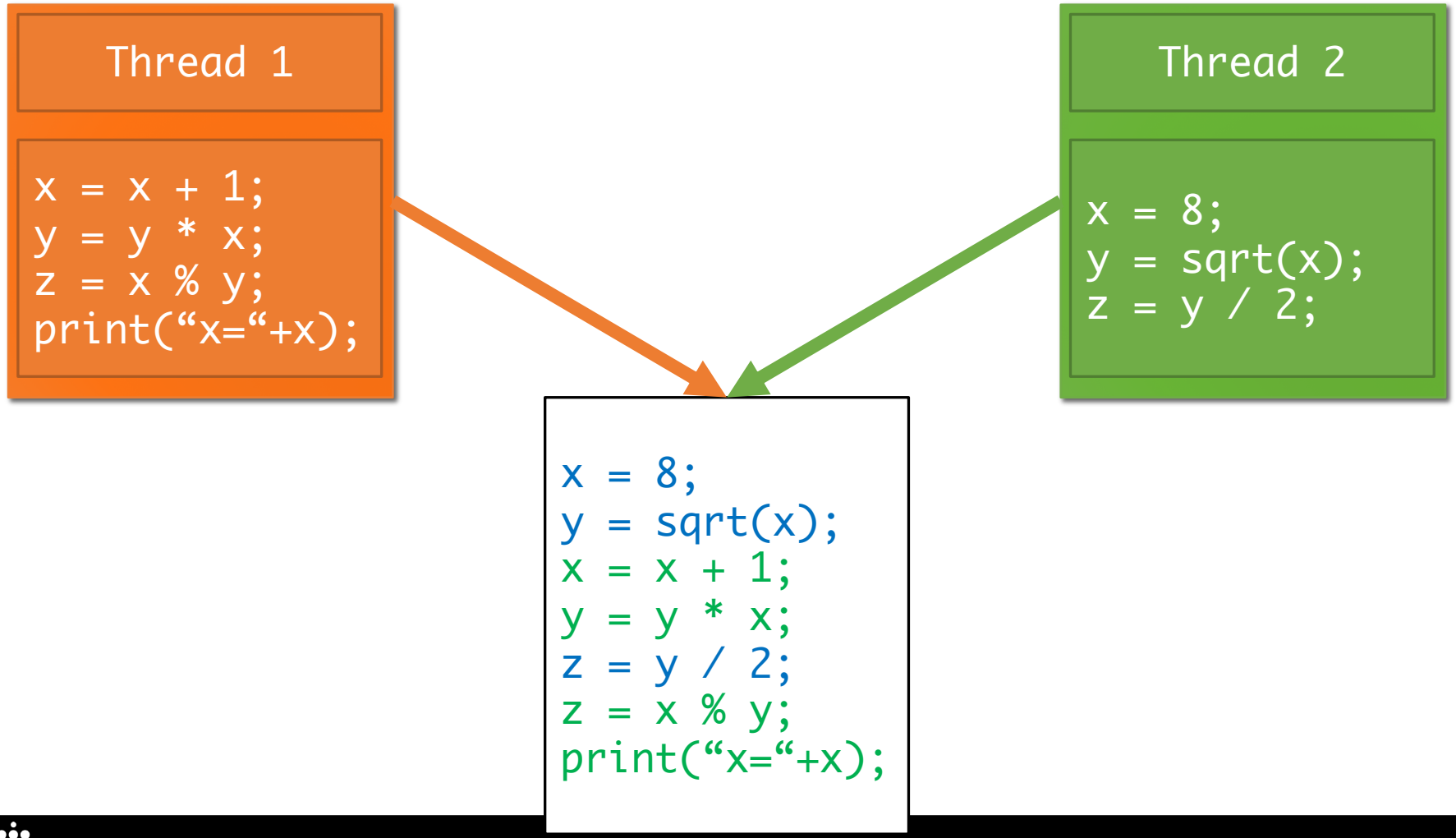
Introducción

Sincronización con Espera Activa



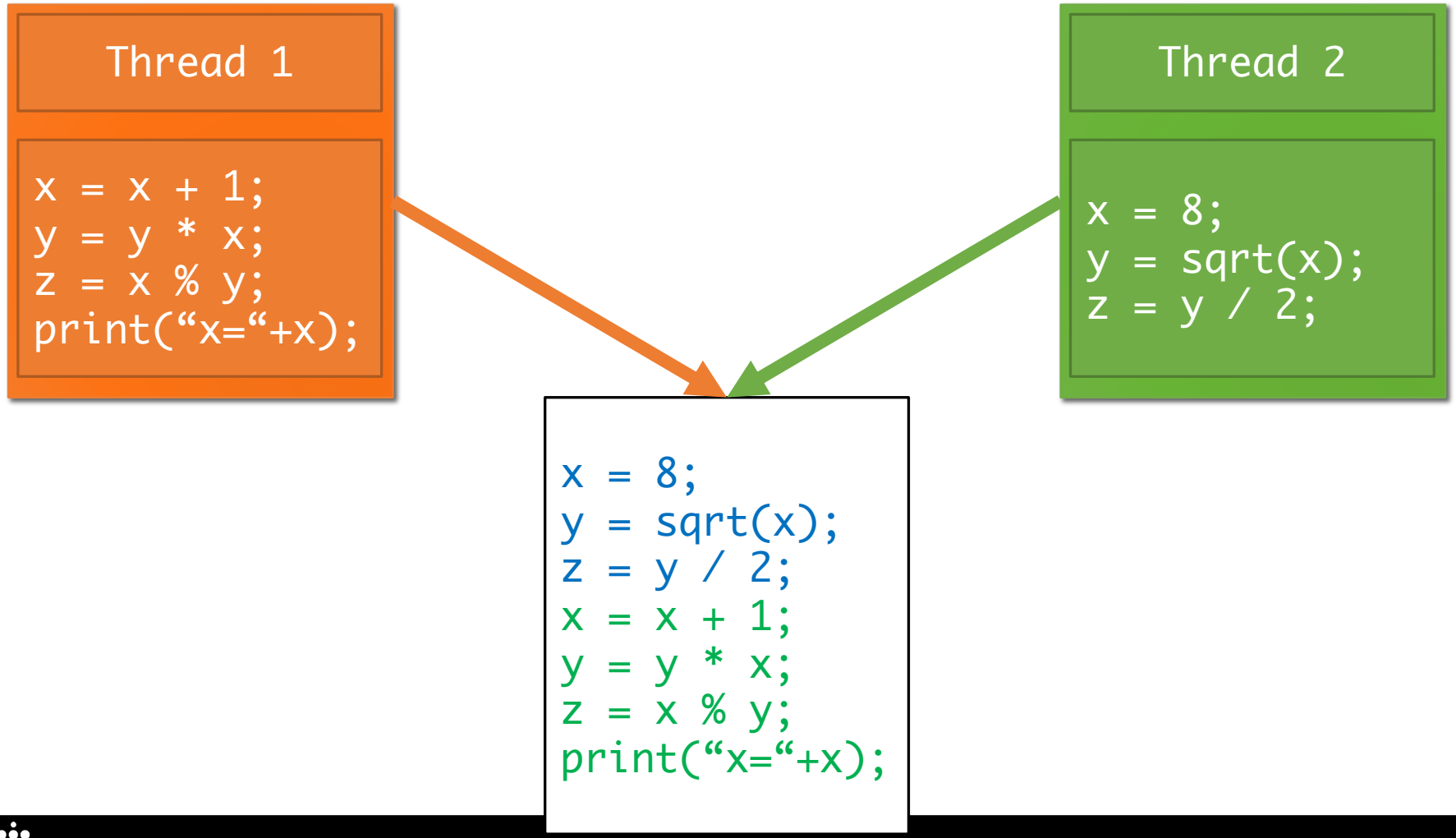
Introducción

Sincronización con Espera Activa



Introducción

Sincronización con Espera Activa



Atributos imprevisibles (`volatile`)

- Los procesadores modernos utilizan **memoria caché** para albergar los valores de las variables
- En sistemas de varios procesadores (núcleos/cores), es habitual que exista una **caché por cada procesador**
- ¿Qué ocurre con las **variables compartidas** si dos procesos se ejecutan cada uno en un procesador?
- ¿Cuándo se **sincronizan** las caches?
- ¿Cuándo son **visibles** los cambios de las variables por todos los hilos?

Atributos imprevisibles (`volatile`)

- Para un compilador **no es viable** determinar qué atributos se comparten entre hilos y cuales no
- **No es eficiente** sincronizar los valores de todos los atributos cada vez que se hace una escritura en ellos por si otro procesador los usa
- Este es un problema de **visibilidad**
- ¿Cómo se **resuelve** el problema?

Atributos imprevisibles (`volatile`)

- Hay que **marcar explícitamente** aquellos atributos compartidos entre hilos que pueden cambiar de valor
- Se marcan con la palabra reservada `volatile` (que puede traducirse por imprevisible)
- Cuando un hilo lee el valor de un atributo `volatile`, el sistema se encarga de que lea el valor que otro hilo escribió antes en él

Atributos imprevisibles (`volatile`)

- ¿Qué ocurre si un atributo es leído por varios hilos y no se ha marcado como **volatile**?
  - Es posible que el hilo lea un valor **antiguo**
  - Es posible que el compilador o la JVM **elimine** la variable si puede optimizar el código considerando que sólo un hilo accede a ella
  - Si el atributo es de tipo **long** o **double** (64bits), las lecturas y escrituras **no son atómicas** a no ser que esté marcado como **volatile**

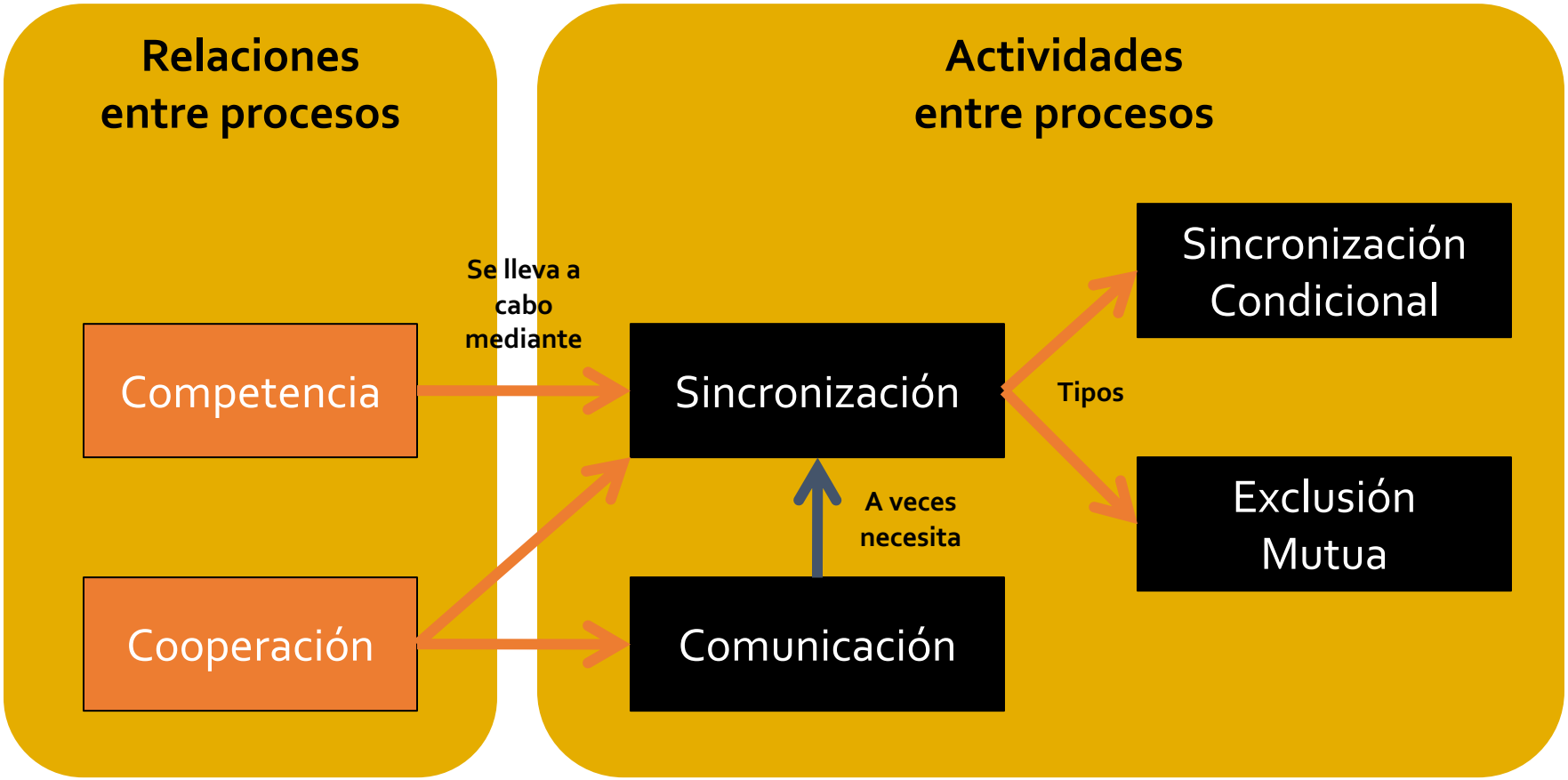
Atributos imprevisibles (`volatile`)

- No es necesario utilizar **volatile** si los hilos utilizan alguna herramienta de **sincronización**
- Por ejemplo:
  - Si los atributos compartidos están bajo exclusión mutua con un **semáforo**, se garantiza que se leerán los valores correctos
  - Si un proceso escribe un atributo y **desbloquea** a otro proceso, el otro proceso leerá el valor escrito

- **Lectura simultáneas**
  - Ambos procesos leen el valor de la variable
- **Escrituras simultáneas**
  - El resultado de la escritura simultánea sobre una variable compartida de tipo simple será el valor escrito por uno u otro proceso
  - Nunca una mezcla de las dos escrituras
- **Lectura y Escritura simultáneas**
  - Si un proceso lee sobre una variable compartida de tipo simple y simultáneamente otro escribe sobre ella, el resultado de la lectura será o bien el valor de la variable antes de la escritura o bien el valor de la variable después de la escritura
  - Nunca un valor intermedio

Introducción

Sincronización con Espera Activa



- Actividades entre procesos con el Modelo de Concurrencia de **Memoria Compartida**

  - **Comunicación**
    - Se utilizan las variables compartidas para compartir información
    - Un proceso escribe un valor en la variable y otro proceso lee ese valor

- Actividades entre procesos con el Modelo de Concurrencia de **Memoria Compartida**
  - **Sincronización**
    - Si sólo se usan variables compartidas para la sincronización se tiene **Espera Activa**
    - Se pueden usar también herramientas específicas de sincronización de procesos del lenguaje de programación, por ejemplo los **Semáforos** o **monitores**
  - Tipos de sincronización
    - Sincronización condicional
    - Exclusión mutua

Sincronización con Espera Activa

- Introducción
- **Sincronización Condicional**
- Exclusión Mutua
- Espera Activa vs Espera Pasiva
- Conclusiones

Sincronización Condicional

Sincronización con Espera Activa

- La **Sincronización Condicional** se produce cuando un proceso debe esperar a que se cumpla una cierta condición para proseguir su ejecución
- Esta condición sólo puede ser activada por otro proceso

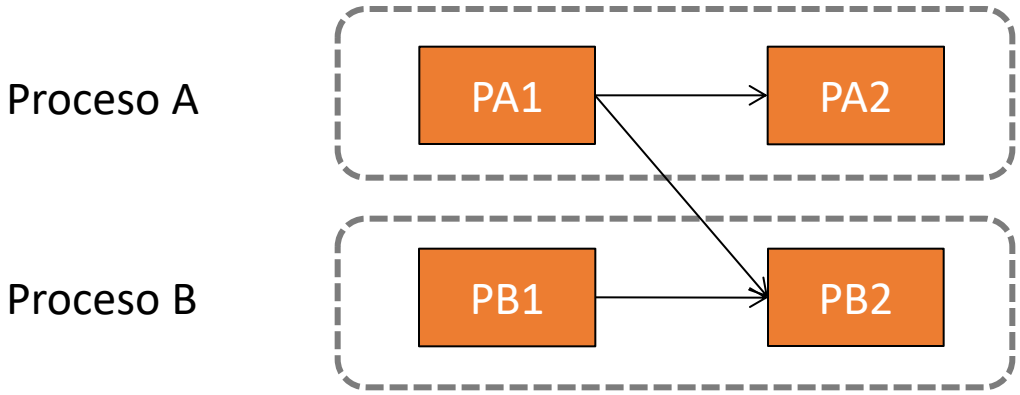


Diagrama de Precedencia

Sincronización Condicional

Sincronización con Espera Activa

- Si los procesos **muestran por pantalla*** el nombre de la acción que han realizado, el resultado de ejecutar el programa concurrente del diagrama podría ser cualquiera de los siguientes:

Salidas Posibles:

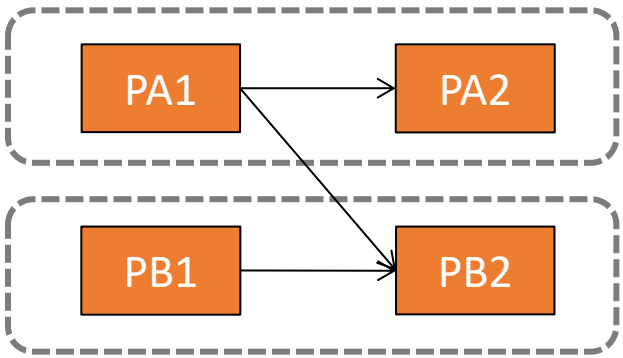
```
PA1 PA2 PB1 PB2
```

```
PA1 PB1 PA2 PB2
```

```
PB1 PA1 PA2 PB2
```

Nunca:

~~PB1 PB2 PA1 PA2~~



* La escritura en pantalla se considera una instrucción atómica

Sincronización Condicional

Sincronización con Espera Activa

Se declara un atributo compartido **`continuar`**

`procA` la pondrá a **`true`** cuando haya ejecutado PA1 y continuará ejecutando PA2

`procB` ejecutará PB1 y comprobará constantemente en un bucle el valor de **`continuar`** Saldrá del bucle cuando valga **`true`** y continuará ejecutando PB2

```
public class SincCondicionalEjemplo1 {

    static volatile boolean continuar;
    public static void procA() throws
                        InterruptedException{
        new Thread(() -> {
            System.out.println("PA1 ");
            continuar = true;
            System.out.println("PA2 ");
        }).start();
    }

    public static void procB() throws
                        InterruptedException{
        new Thread(() -> {
            System.out.println("PB1 ");
            while (!continuar) {}
            System.out.println("PB2 ");
        }).start();
    }

    public static void main(String[] args) throws
                        InterruptedException {
        continuar = false;
        procA();
        procB();
    }
}
```

Sincronización Condicional

Sincronización con Espera Activa

Posible intercalación de instrucciones

	procA	procB	continuar
1	`print("PA1 ");`		`false`
2		`print("PB1 ");`	`false`
3	`continuar = true;`		`true`
4	`print("PA2 ");`		`true`
5		`while (!continuar){}`	`true`
6		`print("PB2 ");`	`true`

Salida: `PA1 PB1 PA2 PB2`

Sincronización Condicional

Sincronización con Espera Activa

Posible intercalación de instrucciones

	procA	procB	continuar
1		`print("PB1 ");`	false
2		`while (!continuar){}`	false
3		`while (!continuar){}`	false
4		`while (!continuar){}`	false
5	`print("PA1 ");`		false
6	`continuar = true;`		true
7	`print("PA2 ");`		true
8		`while (!continuar){}`	true
9		`print("PB2 ");`	true

Salida: `PB1 PA1 PA2 PB2`

Sincronización Condicional

Sincronización con Espera Activa

Posible intercalación de instrucciones

	procA	procB	continuar
1	`print("PA1 ");`		`false`
2	`continuar = true;`		`true`
3	`print("PA2 ");`		`true`
4		`print("PB1 ");`	`true`
5		`while (!continuar){}`	`true`
6		`print("PB2 ");`	`true`

Salida: **PA1 PA2 PB1 PB2**

Ejercicio 1 – Productor Consumidor

Sincronización Condicional

- Se desea implementar un programa concurrente con un proceso que produce información (**productor**) y otro proceso que hace uso de esa información (**consumidor**)
- El proceso **productor** genera un número aleatorio y termina
- El proceso **consumidor** muestra por pantalla el número generado y termina

Ejercicio 2 – Productor Consumidor Infinito

Sincronización Condicional

- Se desea ampliar el programa anterior de forma que el proceso **productor** esté constantemente produciendo
- El proceso **consumidor** estará constantemente consumiendo los productos
- No se puede quedar ningún producto sin consumir
- No se puede consumir dos veces el mismo producto

Ejercicio 3 – Cliente Servidor 1 Petición

Sincronización Condicional

- Programa formado por un **proceso servidor** y otro **proceso cliente**
- El **proceso cliente** hace una petición al **proceso servidor** y espera su respuesta, cuando la recibe, la procesa.
- El **proceso servidor** no hace nada hasta que recibe una petición, momento en el que la contesta.
- El **proceso cliente** va a pedir un número aleatorio al servidor y lo va a procesar mostrándolo por pantalla

Ejercicio 4 – Cliente Servidor N Peticiones

Sincronización Condicional

- Se desea ampliar el programa anterior de forma que el **proceso cliente** esté constantemente haciendo peticiones y el **proceso servidor** atendiendo a las mismas