



Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



Programación Concurrente



*Sincronización I: competencia –
Exclusion mutua*

Exclusión mútua

Mecanismos

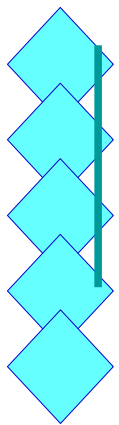
```
graph TD; A[Mecanismos] --> B[Métodos y Bloques sincronizados]; A --> C[Semáforos]; A --> D[Cerrojos];
```

The diagram illustrates the mechanisms for mutual exclusion. A yellow box labeled 'Mecanismos' is connected by arrows to three other boxes: 'Métodos y Bloques sincronizados' (light blue), 'Semáforos' (blue with a double border), and 'Cerrojos' (light blue). The 'Semáforos' box is highlighted with a double border.

Métodos y Bloques sincronizados

Semáforos

Cerrojos



Ejemplo contador

```
public class ProcesoI
implements Runnable{
    private Datos unDato;

    public ProcesoI(Datos unD) {
        unDato = unD;
    }

    public void run() {
        for (int i=1; i<10000;
            i++) {
            unDato.incrementar();
        }
    }
}
```

```
public class Datos {
    private int dato;
    private Semaphore mutex

    public Datos(int nro) {
        dato = nro;
        mutex = new Semaphore(1);
    }

    public int getDato() {
        return dato;
    }

    public void incrementar() {
        mutex.acquire();
        dato++;
        mutex.release();
    }
}
```

Mecanismo del semáforo

Podemos utilizarlos para lograr la exclusión mútua.

Los procesos COMPITEN por entrar a la sección crítica

Entrada a la SC /adquirir el SEM

SECCION CRITICA

Salida de la SC / liberar el SEM

SECCION RESTANTE

*En el algoritmo utilizamos
las directivas adquirir y liberar,
pero cada lenguaje
tiene sus métodos*

ALGORITMO ejemploSem

....

semaforo adquirir

//código sección crítica

semaforo liberar

.....

FIN ALGORITMO ejemploSem

Mecanismo del semáforo

Podemos utilizarlos para lograr la exclusión mútua.

ALGORITMO ejemploSem

.....

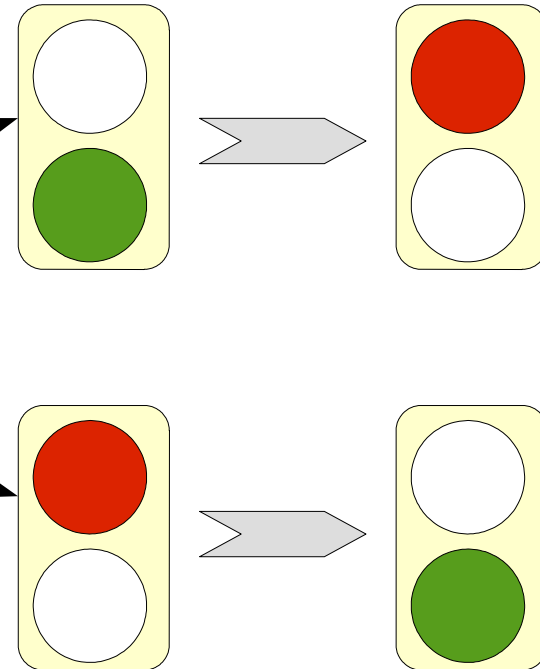
semaforo adquirir

//código sección crítica

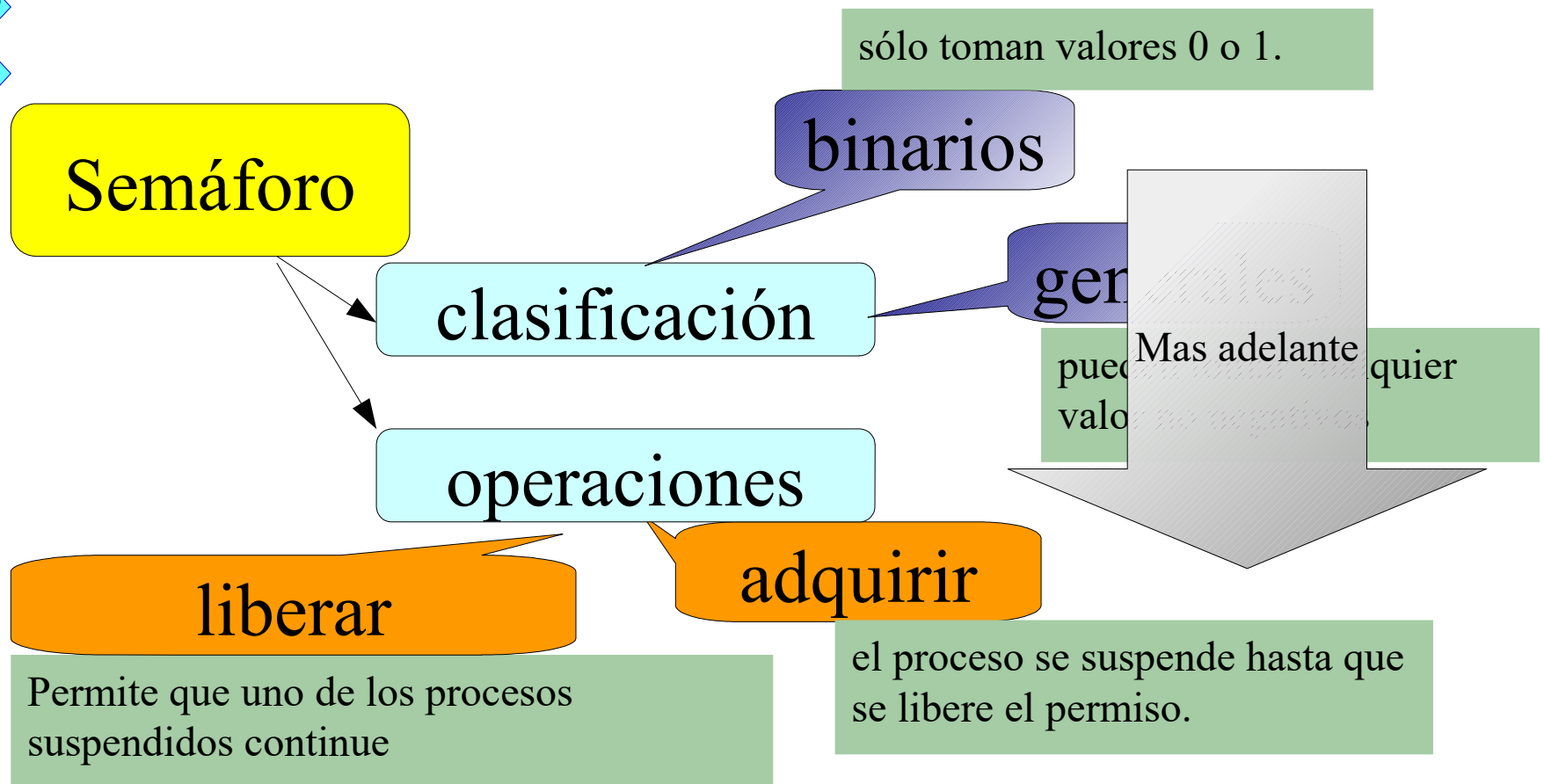
semaforo liberar

.....

FIN ALGORITMO ejemploSem



Semáforos, generalidades



Semáforos, operaciones


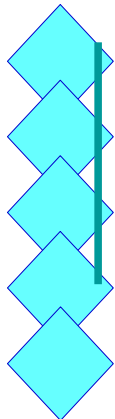
- Adquirir

- Si el valor del **semáforo no es nulo** (está abierto o tiene permiso disponible) decrementa el valor del semáforo.
- Si el valor del **semáforo es nulo** (está cerrado o NO tiene permiso disponible), el hilo que lo ejecuta se suspende y se encola en la lista de procesos en espera de un permiso del semáforo.

- liberar

Semáforos, operaciones

- adquirir
- liberar
 - Si hay **algún proceso en la lista de procesos** del semáforo, activa uno de ellos para que ejecute la sentencia que sigue al “adquirir” que lo suspendió.
 - Si **no hay procesos en espera en la lista** incrementa en 1 el valor del semáforo y queda un permiso disponible.



Semáforos, en general

- Garantiza que la **operaciones** de chequeo del valor del semáforo, y posterior actualización según proceda, sea siempre **segura**
- La inicialización del semáforo **no es una operación segura** por lo que no se debe ejecutar en concurrencia con otro proceso utilizando el mismo semáforo.



Semáforos, comportamiento interno



- Algoritmos de adquisición y liberación de permisos
(operaciones $(p)/wait$ y $(v)/signal$ en SO)

```
ALGORITMO adquirir
  SI semaforo > 0 HACER
    semaforo  $\leftarrow$  semaforo - 1
  SINO
    suspende proceso y lo pone en la cola del semáforo
  FIN SI
FIN ALGORITMO adquirir
```



Semáforos, comportamiento interno



Aunque haya varios procesos, sólo activa uno

Elegido de acuerdo con un criterio propio de la implementación (FIFO, LIFO, Prioridad, etc.).

ALGORITMO liberar

SI hay algún proceso en la cola de semáforos HACER
activa uno de ellos

SINO

semaforo \leftarrow semaforo + 1

FIN SI

FIN ALGORITMO



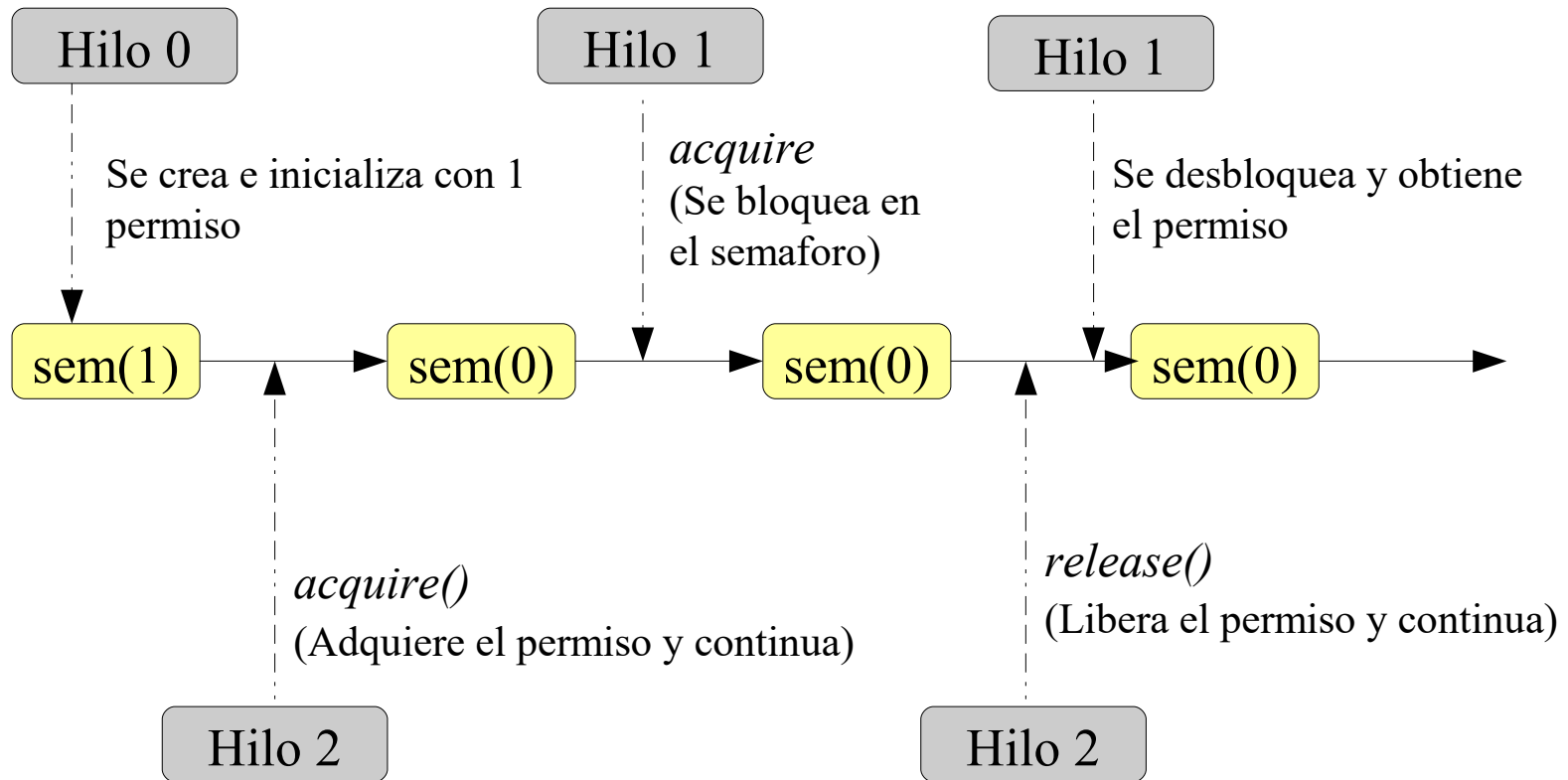
Exclusión mutua con semáforos



- Si trabajamos la exclusión mutua dentro del recurso compartido, el semáforo debe crearse en el constructor del recurso compartido.
- En general se le llama “**mutex**” por “mutual Exclusion”

- Se utiliza el semáforo “**mutex**” para tener el acceso exclusivo a la sección crítica.
 - Cuando “**mutex**” tiene el valor 0 (por efecto de un adquirir), algún proceso está en su sección crítica.
 - Cuando “**mutex**” tiene el valor 1 (por efecto de un liberar) no hay ningún proceso ejecutando la sección crítica.

Funcionamiento de un semáforo






Semáforos para Exclusion Mutua

package java.util.concurrent - class Semaphore

- semáforo binario: gestiona 1 permiso de acceso
 - void *acquire()* (se toma el permiso)
 - void *release()* (se libera el permiso)
 - boolean *tryAcquire()* (se intenta tomar el permiso)
- ¿cómo se inicializa un semáforo, en 0 o en 1?
- ¿qué significa inicializar el semáforo en 0?
- ¿qué significa inicializar el semáforo en 1?

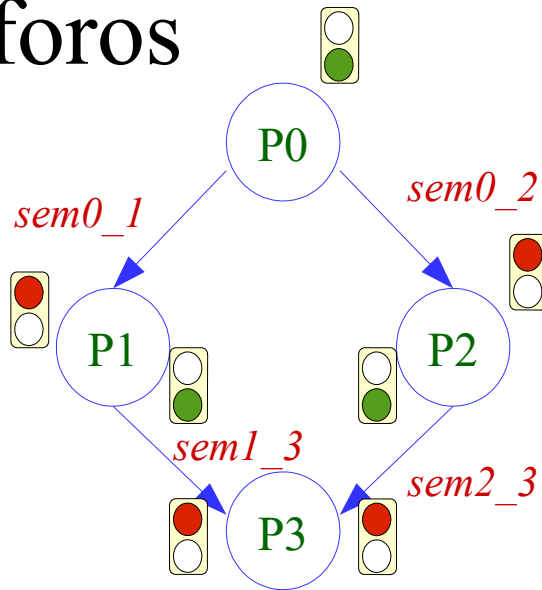
Semáforos, en general

- Actúa como **mediador entre un proceso y el entorno del mismo.**
- Proporciona una forma simple de **comunicación sincrónica.**
- El semáforo también se utiliza para comunicar procesos/hilos
- Permiten establecer un orden de ejecución entre los hilos



**Semáforo
de paso**

Semáforos



**Semáforos
de paso**

En el grafo de precedencia dado se requiere que

- P0 se ejecute antes que P1 \longrightarrow sem0_1
- P0 se ejecute antes que P2 \longrightarrow sem0_2
- P1 se ejecute antes que P3 \longrightarrow sem1_3
- P2 se ejecute antes que P3 \longrightarrow sem2_3



¿cómo hay que
inicializar los
semáforos?

Semáforos

Crear los semaforos

Proceso P0

actuar P0

liberar sem0_1 //da permiso a P1 para proceder

liberar sem0_2 //da permiso a P2 para proceder

fin P0

Proceso P1

adquirir sem0_1 //espera el permiso de P0 para proceder

actuar P1

liberar sem1_3 //da permiso a P3 para proceder

fin P1

Proceso P2 ...

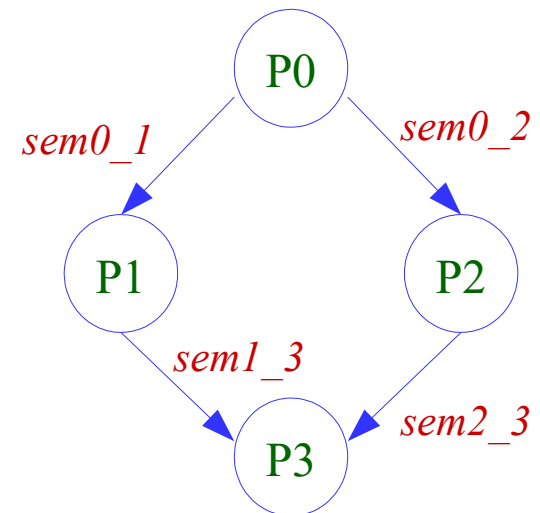
Proceso P3

adquirir sem1_3

adquirir sem2_3

actuar P3

fin P3



Exclusión mútua

Mecanismos

```
graph TD; A[Mecanismos] --> B[Semáforos]; A --> C[Métodos y Bloques sincronizados]; A --> D[Cerrojos];
```

The diagram illustrates the mechanisms for mutual exclusion. A central yellow box labeled 'Mecanismos' has three arrows pointing to three other boxes: 'Semáforos' (light blue), 'Métodos y Bloques sincronizados' (light blue), and 'Cerrojos' (blue with a double border). The 'Cerrojos' box is highlighted with a thicker border.

Semáforos

Métodos y Bloques sincronizados

Cerrojos

Mecanismo del cerrojo

- Mecanismo simple basado en cerrojos

Adquirir cerrojo

SECCION CRITICA

Liberar cerrojo

SECCION RESTANTE

- Las operaciones de adquisición y liberación del cerrojo han de ser atómicas (su ejecución ha de producirse en una unidad de trabajo indivisible).

Cerrojo – Interfaz Lock de Java

- Mecanismo de sincronización de hilos, es más flexible y sofisticado que los métodos sincronizados, pero menos eficiente.
- La interfaz Lock define un conjunto de operaciones abstractas de toma y liberación de un lock.
- las operaciones de suspensión y liberación de un lock son explícitas.



A Investigar!!

Interfaz Lock

Lock es una interfaz de Java dentro del package `java.util.concurrent.locks` con los siguientes métodos:

- El método *lock()*
- El método *tryLock()*
- El método *unlock()*

ReentrantLock es una implementación de la interfaz ***Lock*** de Java dentro del package `java.util.concurrent.locks`

Comparamos...

- **Bloques sincronizados**
- **Métodos sincronizados**
- **Locks explícitos/cerrojos**
- **Semáforos**

