



Departamento de Programación  
Facultad de Informática  
Universidad Nacional del Comahue



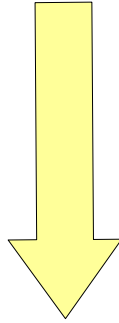
# Programación Concurrente



*Mecanismos de Exclusión mutua*  
*Propiedades*

# Cómo sincronizar?

Usar la palabra **synchronized**

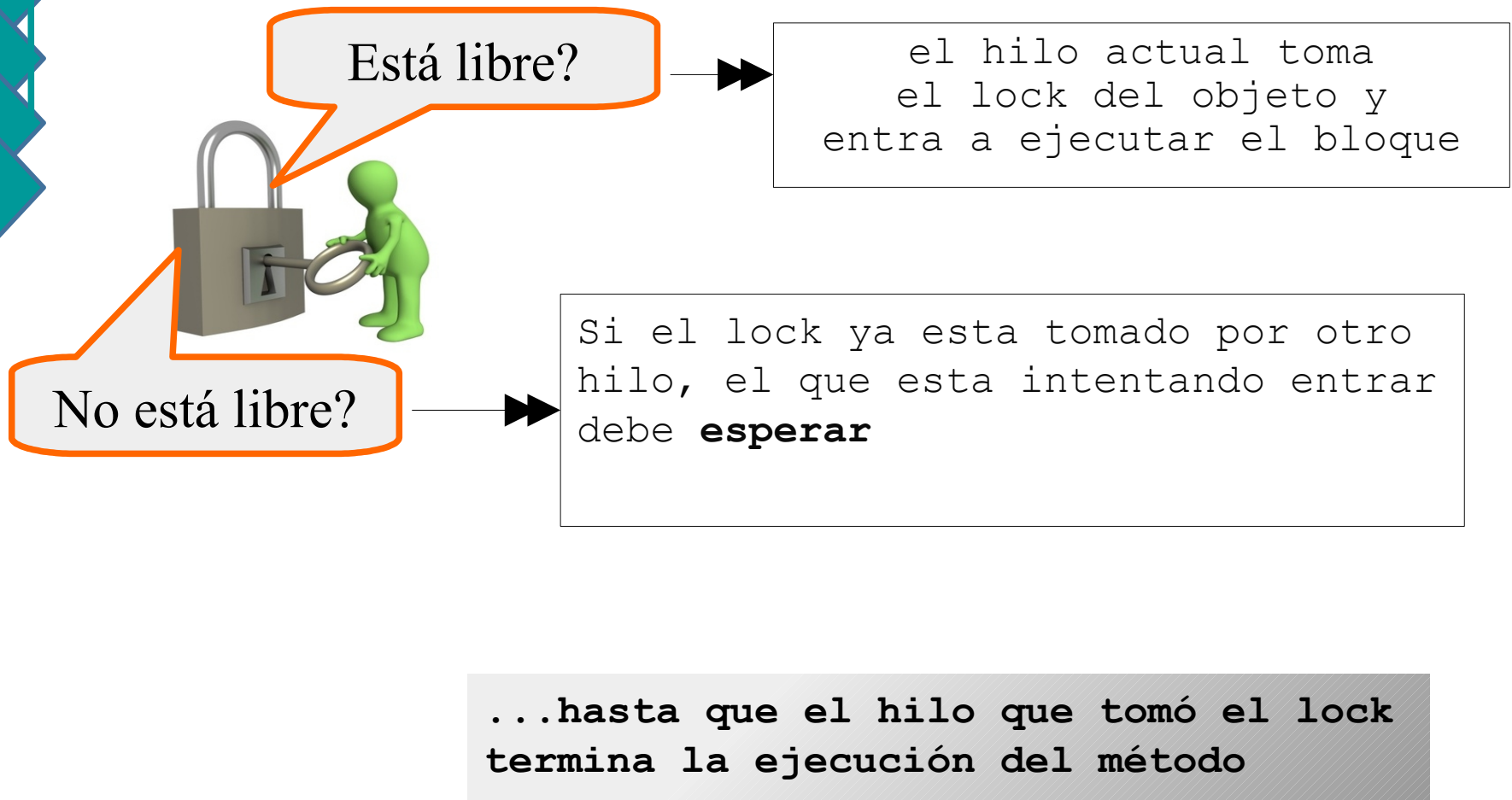


La parte sincronizada  
puede ser ejecutada por  
un sólo hilo por vez



el hilo necesita  
la llave para acceder  
al código sincronizado.

# Cómo sincronizar?





# Exclusion mutua en Java - Synchronized

- Java utiliza **synchronized** para lograr la exclusion mutua sobre los objetos.
- Existen 2 posibilidades para sincronizar objetos
  - El bloque **synchronized**
  - Dentro de la clase del objeto sincronizado **los métodos están declarados como synchronized**
- Cada vez que un hilo intenta ejecutar un método sincronizado sobre un objeto lo puede hacer sólo si no hay algún otro hilo ejecutando un método sincronizado sobre el mismo objeto



# Exclusion mutua en Java - Synchronized

- Un **hilo** que intenta ejecutar un método sincronizado sobre un objeto cuyo lock ya está en poder de otro hilo **es suspendido** y puesto en espera hasta que el lock del objeto es liberado.
- El lock se libera cuando el hilo que lo tiene tomado:  
termina la ejecución del método / ejecuta un return / lanza una excepción.



# Exclusión mutua en java – Synchronized

- El mecanismo de sincronización funciona si TODOS los accesos a los *datos delicados* ocurren dentro de métodos sincronizados, es decir con exclusión mútua
- Los datos delicados protegidos por métodos sincronizados deben ser privados

# Exclusion mutua en Java -Synchronized

- Cada instancia de **Object** y sus subclases posee bandera de bloqueo (**lock implícito**)
- Los tipos primitivos (no objetos) solo pueden bloquearse a través de los objetos que los encierran
- No pueden sincronizarse variables individuales
- Los objetos arreglos cuyos elementos son tipos primitivos pueden bloquearse, pero sus elementos NO

# Exclusión mútua

Mecanismos

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

Métodos y Bloques sincronizados

Semáforos

Cerrojos



# Mecanismo del semáforo

Los utilizamos 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

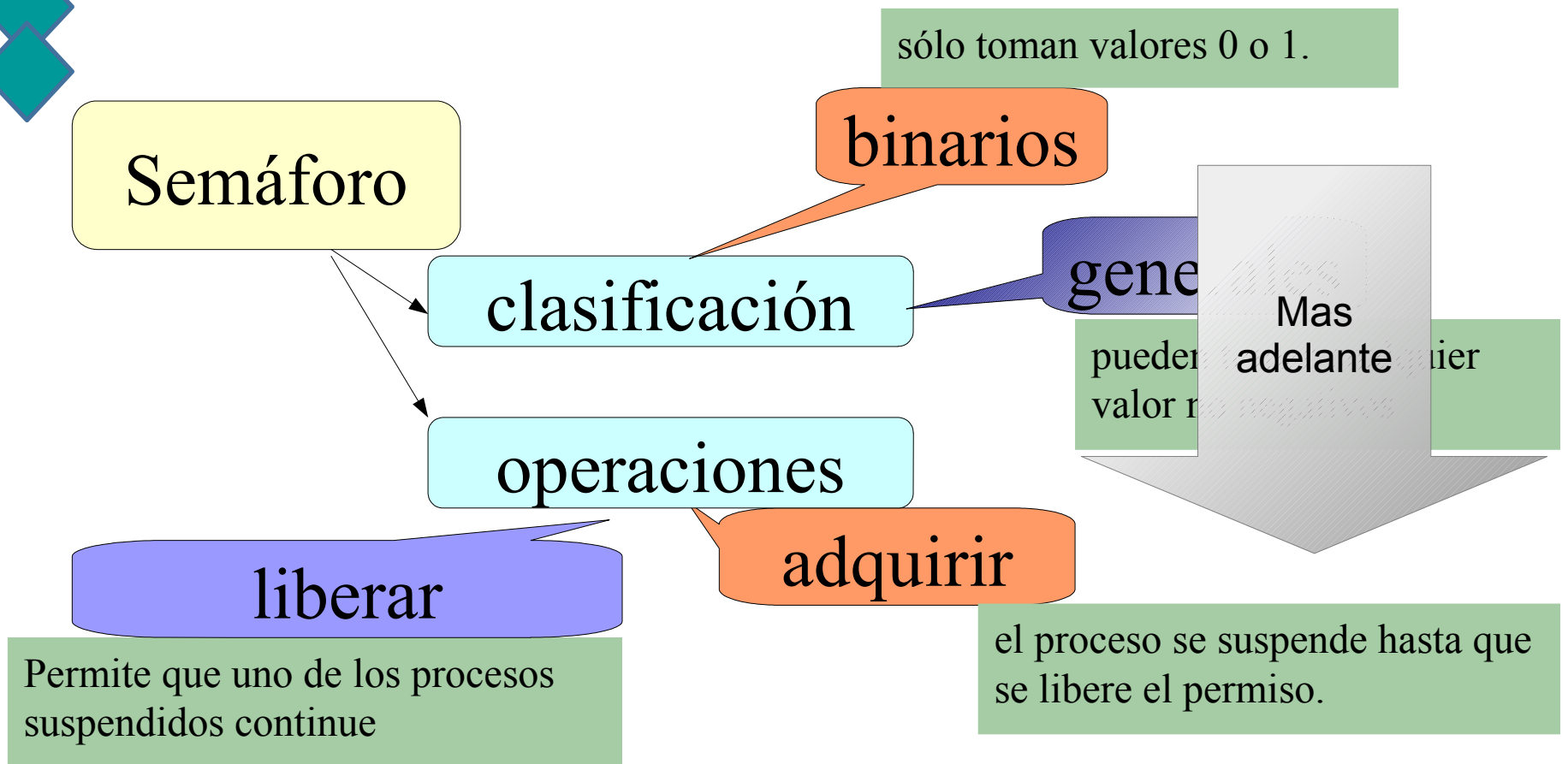
.....


semaforo adquirir  
//código sección crítica  
semaforo liberar

.....

FIN ALGORITMO ejemploSem

# 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

- Actúa como **mediador entre un proceso y el entorno** del mismo.
- Proporciona una forma simple de **comunicación sincrónica**.
- 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

ALGORITMO *exclusionMutua*

**mutex: SemaforoBinario**

**p,q, r: Proceso**

**iniciar (mutex,1);**

**cobegin p; q; r; coend;**

FIN ALGORITMO

- Se plantea una solución del problema de exclusión mutua entre tres procesos *p,q, r* respecto de una sección crítica dada.
- Se utiliza un semáforo “**mutex**” para controlar el acceso a la misma.



# Exclusión mutua con semáforos

ALGORITMO *proceso*

REPETIR

adquirir (**mutex**)

//código de la sección crítica

liberar (**mutex**)

HASTA ...

FIN ALGORITMO *proceso*

Cuando *p, q, o r*  
ejecutan “proceso” de  
forma concurrente ...

- 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.
- El semáforo es inicializado a 1, ya que al comenzar, ningún proceso se encuentra en la zona crítica.



# 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 el semáforo, en 0 o en 1?
- ¿qué significa inicializar el semáforo en 0?
- ¿qué significa inicializar el semáforo en 1?

# Semáforos

```
Semaphore semaforo = new Semaphore (1, true);
```

```
....
```

```
semaforo.acquire();
```

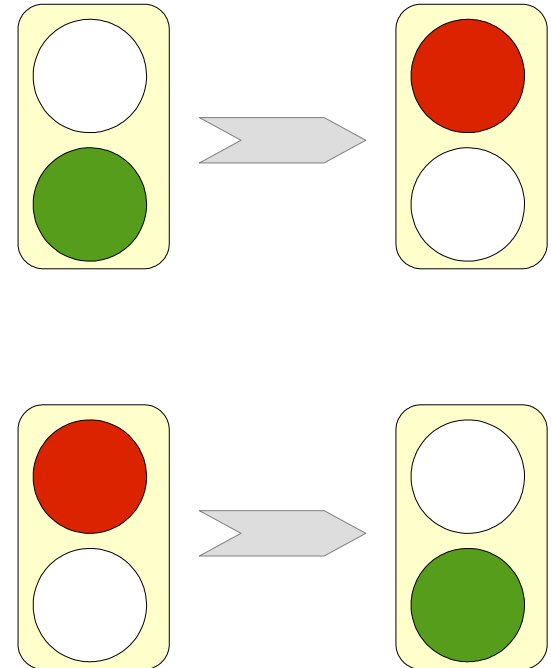
```
    int valor = cc.getN(id);
```

```
    valor++;
```

```
    cc.setN(id, valor);
```

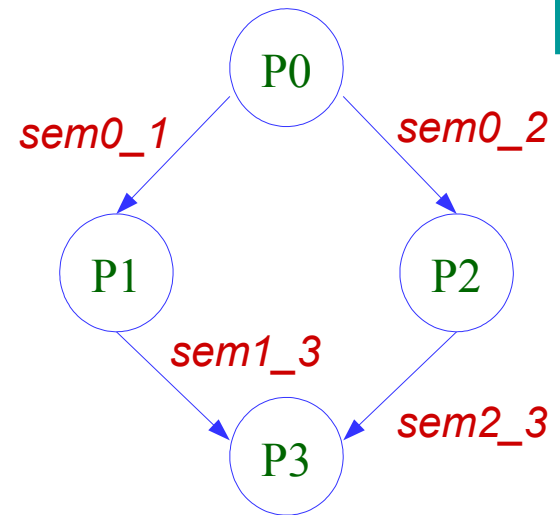
```
semaforo.release();
```

```
.....
```



# Semáforos

- El semáforo también se utiliza para comunicar procesos/hilos
- Permiten establecer un orden de ejecución entre los hilos
- 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
- Los semáforos se inician en 0 en este caso



# Semáforos

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 ...

