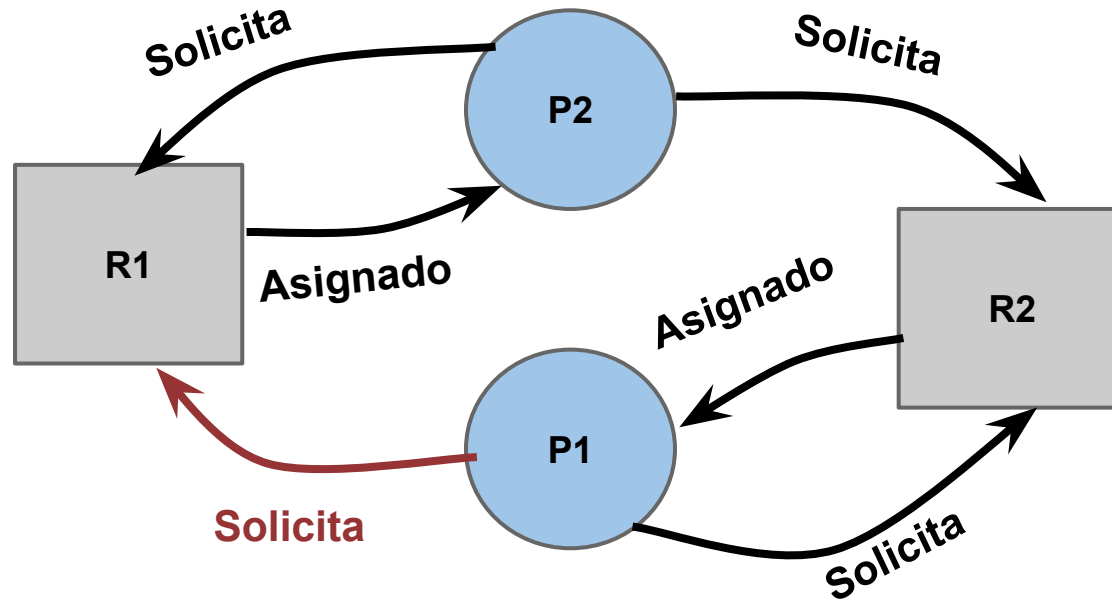


Deadlock

- Los sistemas poseen recursos limitados
 - ◆ Recursos consumibles → Ej: interrupción, señal, mensaje, info en IO buffers
 - ◆ Recursos reusables → Ej: Memoria, archivos, dispositivos IO
- Los procesos para utilizar los recursos deben pedirlos y luego liberarlos a través de llamadas al sistema:
 - ◆ Recursos gestionados a través del SO → **open/close** – **malloc/free**
 - ◆ Recursos no gestionados por el SO → **wait /signal**
- Los recursos pueden tener más de una **instancia** (cualquiera satisface a un proceso por igual)

Situación 2: P2 y P2 necesitan tanto R1 y R2 para ejecutar.

P1 y P2 SE
ENCUENTRAN EN
DEADLOCK




P2 SUFRE DE
STARVATION

Situación 1: P2 necesita R1 y R2 para ejecutar. P1 pide únicamente R2 y no lo libera.

- Un **conjunto de procesos** estará en un estado de **interbloqueo/deadlock** cuando todos los procesos del conjunto estén **esperando** a que se produzca un **suceso** que sólo puede producirse como **resulta-do de la actividad de otro proceso** del conjunto.
- En un deadlock, los procesos nunca terminan de ejecutarse y los recursos del sistema están ocupados, lo que impide que se inicien otros trabajos.

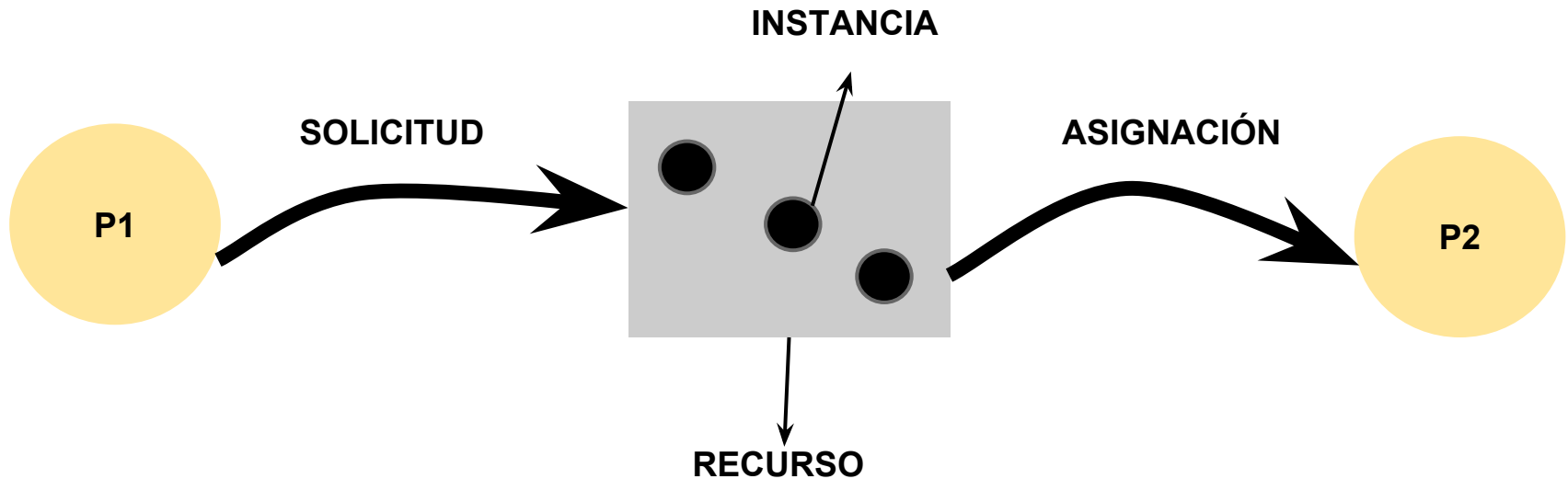
```
mutex = 1;  
sem1 = 1;  
sem2 = 0;
```

P1	P2
<pre>WHILE (true) { wait(sem1); T1) wait(mutex); T3) SECCIÓN CRÍTICA signal(mutex); signal(sem2); }</pre>	<pre>WHILE (true) { wait(mutex); T2) wait(sem2); T4) SECCIÓN CRÍTICA signal(mutex); signal(sem1); }</pre> 

El orden en que se realizan los waits genera un deadlock

- **Exclusión mutua.** Al menos un recurso debe estar en modo no compartido (sólo un proceso puede usarlo a la vez). Si otro proceso solicita el recurso, el proceso solicitante tendrá que esperar hasta que el recurso sea liberado.
- **Retención y espera.** Un proceso debe estar reteniendo al menos un recurso y esperando para adquirir otros recursos adicionales que actualmente estén retenidos por otros procesos.
- **Sin desalojo.** Los recursos no pueden ser desalojados; es decir, un recurso sólo puede ser liberado voluntariamente por el proceso que le retiene, después de que dicho proceso haya completado su tarea.
- **Espera circular.** Debe existir un conjunto de procesos en espera, tal que cada uno espere un recurso retenido por el siguiente.

GRAFO DE ASIGNACIÓN DE RECURSOS



- Si el grafo no contiene ningún ciclo → **No** hay deadlock
- Si el grafo contiene un ciclo → **Puede existir** un deadlock
- Si cada recurso tiene una instancia y hay un ciclo → **Hay** un deadlock

- Utilizar un protocolo para impedir o evitar los deadlocks, asegurando que el sistema nunca entre en dicho estado. **(PREVENCIÓN / EVASIÓN)**

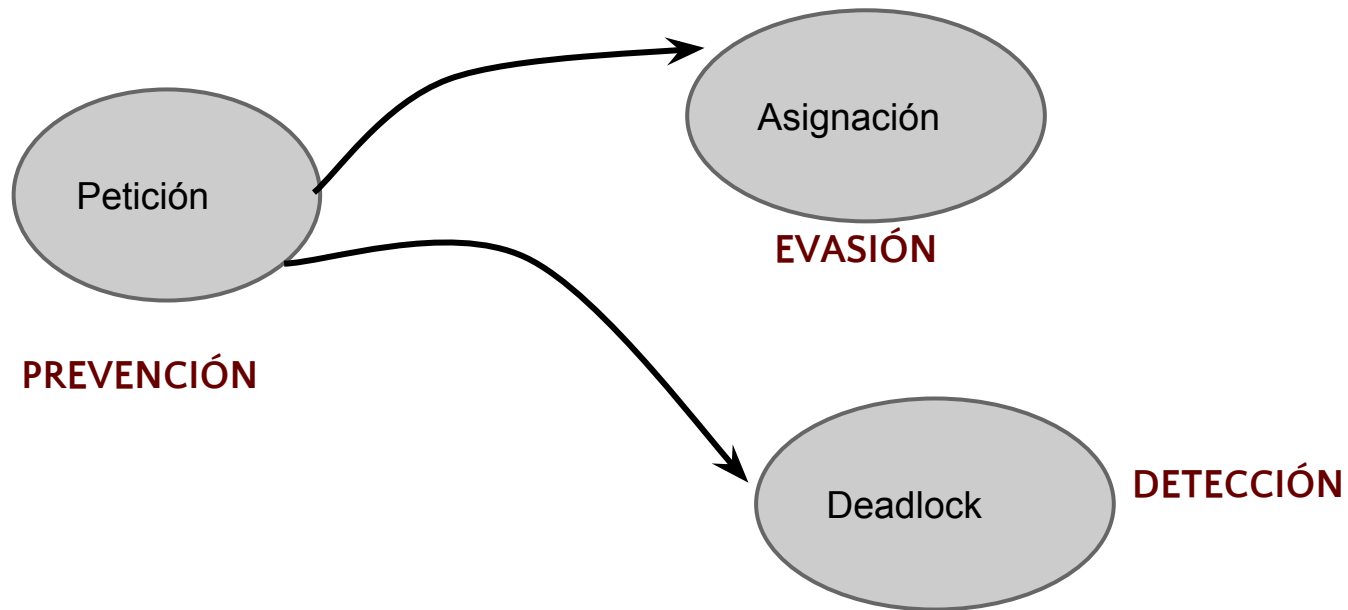


- Permitir que el sistema entre en estado de deadlock, detectarlo y realizar una recuperación. **(DETECCIÓN Y RECUPERACIÓN)**

- Ignorar el problema y actuar como si nunca se produjeran deadlocks en el sistema.



¿En dónde se aplica cada técnica?



- Proporciona un conjunto de métodos para asegurar que al menos una de las condiciones necesarias no pueda cumplirse.
- Se restringe el modo en que se pueden realizar las solicitudes.
- Son políticas del SO definidas a priori

- PREVENIR
 - ◆ **Mutua exclusión**

 - ◆ **Espera y retención**

 - ◆ **Sin desalojo**

 - ◆ **Espera circular:**

- ➔ **Mutua exclusión:** evitar la mutua exclusión sobre recursos compartibles. Ej: apertura de archivos en modo lectura.
- ➔ **Espera y retención**
 - Los procesos pidan y se le asignen previamente todos los recursos que vaya a usar.
 - Un proceso para pedir un nuevo recurso tenga que liberar todos los que retiene.
- ➔ **Sin desalojo**
 - Si un proceso pide un recurso que no está disponible, debe esperar y se liberan todos los recursos que tenía asignados.
 - Si un proceso pide un recurso retenido por otro proceso en espera, los mismos son desalojados y asignados al primero
- ➔ **Espera circular:** requerir que los recursos se pidan en orden.

Establecer orden petición recursos

A cada recurso se le asigna un valor

$$F(R1) = 3 \quad F(R2) = 1 \quad F(R3) = 2$$

Inicialmente se puede realizar cualquier petición

Las siguientes peticiones deben ser a R con valores crecientes

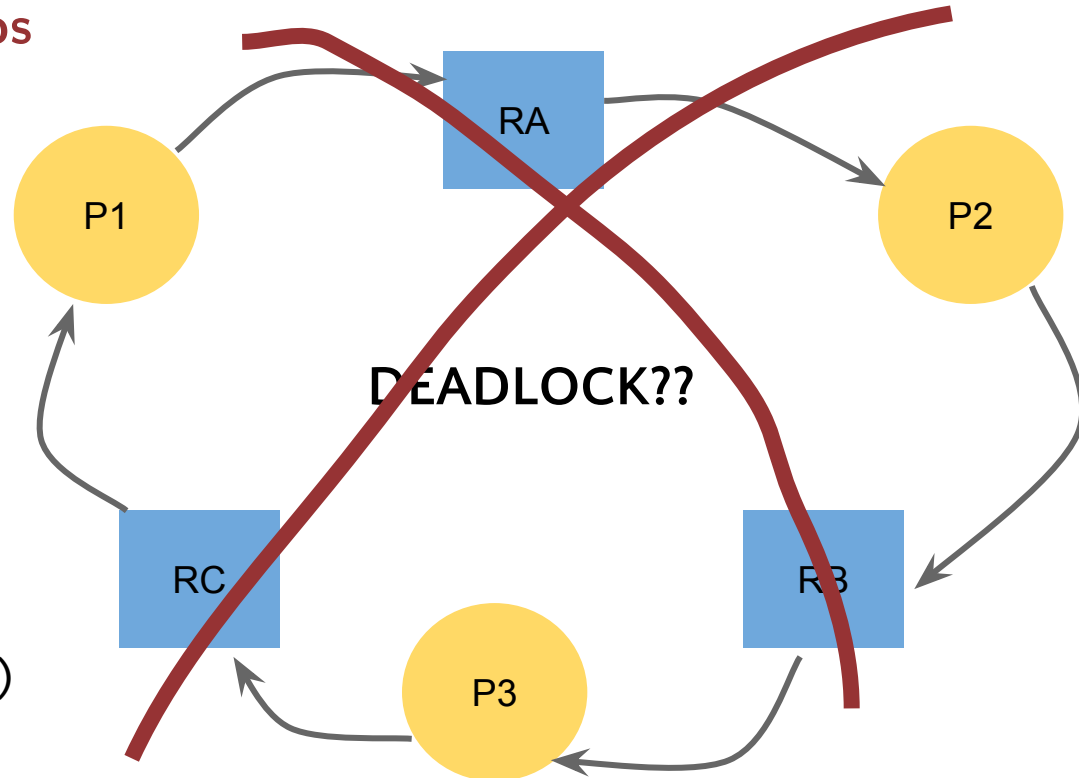
P1	P2	P3
R1	R3	R2
R2	R1	R3
	R2	R1

Establecer orden petición recursos

- 1) $F(RB) > F(RA)$
- 2) $F(RC) > F(RB)$
- 3) $F(RA) > F(RC)$

$\Rightarrow F(RC) > F(RB) > F(RA) > F(RC)$

$\Rightarrow F(RC) > F(RC) ??$



- El proceso debe indicarle al SO cuáles van a ser los **recursos máximos** que puede llegar a solicitar durante su tiempo de vida.
- Ante cada solicitud, se analizará si se le asigna el recurso al proceso o si se lo hace esperar. Para tomar una decisión se realiza una simulación teniendo en cuenta las posibles futuras solicitudes y liberaciones de recursos por parte de todos los procesos del sistema.
- Mantiene al sistema siempre en **Estado Seguro**
 - ◆ Un estado es seguro si el sistema puede asignar recursos a cada proceso (hasta su máximo) en determinado orden sin que eso produzca un deadlock (existe una **secuencia segura**).
 - ◆ Si el estado es **seguro** → no existe ni existirá deadlock
 - ◆ Si el estado es **inseguro** → podría ocurrir deadlock
- Sólo se asigna un recurso si dicha asignación deja al sistema en estado seguro.
- Se podría utilizar un grafo de asignación de recursos para analizar el estado del sistema (agregando el tipo de arista **declaración - - - >**)

Algoritmo del Banquero

- Los procesos al ingresar al sistema declaran la cantidad máxima de cada uno de los recursos que podrá requerir.
- Cuando un proceso solicita un conjunto de recursos, el sistema debe determinar si la asignación de dichos recursos dejará al sistema en un estado seguro. En caso afirmativo, los recursos se asignarán; en caso contrario, el proceso tendrá que esperar hasta que los otros procesos liberen los suficientes recursos.
- Estructuras necesarias:
 - ◆ Matriz de peticiones máximas
 - ◆ Matriz de recursos asignados
 - ◆ Matriz de necesidad
 - ◆ Vector de recursos totales
 - ◆ Vector de recursos disponibles
- Ante cada solicitud se debe simular la asignación, actualizando las estructuras adecuadas, y luego analizar si existe una secuencia segura.

Ejemplo

D
A
T
O
S

Peticiones Máximas (PM)

	R1	R2	R3
P1	2	0	1
P2	1	3	1
P3	1	1	1

Recursos Asignados (RA)

	R1	R2	R3
P1	1	0	0
P2	0	1	1
P3	1	0	0
	2	1	1

Recursos totales

R1	R2	R3
2	3	2

Recursos disponibles

R1	R2	R3
0	2	1

Necesidad = PM - RA

	R1	R2	R3
P1	1	0	1
P2	1	2	0
P3	0	1	1

- 1) Obtengo situación actual:

A. Si P3 pide 1 R2 ... puedo asignárselo inmediatamente??

2) La petición es válida?

Petición = (0, 1, 0)

- Es menor a los recursos totales
- Es menor a lo que le queda por pedir



3) Tengo esa cantidad disponible?

→ SI



4) Deja a mi sistema en estado seguro?

- Simular asignación
- Ver si el estado resultante es seguro



4.1) Simulo asignación

Recursos Asignados (RA)

	R1	R2	R3
P1	1	0	0
P2	0	1	1
P3	1	1	0

Necesidad = PM - RA

	R1	R2	R3
P1	1	0	1
P2	1	2	0
P3	0	0	1

Deja en estado
seguro:
PUEDO ASIGNAR



Recursos disponibles

R1	R2	R3
0	1	1

4.2) Busco al menos UNA secuencia segura

	R1	R2	R3
Inicial	0	1	1
Elijo P3	0	1	0
Finaliza P3	1	2	1
Elijo P2	0	0	1
Finaliza P2	1	3	2
Elijo P1	0	3	1
Finaliza P1	2	3	2

Secuencia: P3 - P2 - P1

B. Si P2 pide 2 R2 ... puedo asignárselo inmediatamente??

2) La petición es válida?

Petición = (0, 2, 0)

- Es menor a los recursos totales
- Es menor a lo que le queda por pedir



3) Tengo esa cantidad disponible?

→ SI



4) Deja a mi sistema en estado seguro?

- Simular asignación
- Ver si el estado resultante es seguro



4.1) Simulo asignación

Recursos Asignados (RA)

	R1	R2	R3
P1	1	0	0
P2	0	2	1
P3	1	0	1

Necesidad = PM - RA

	R1	R2	R3
P1	1	0	1
P2	1	0	0
P3	0	1	1

4.2) Busco al menos UNA secuencia segura

No puedo atender a ningún
proceso:
No existe secuencia segura

Deja en estado inseguro:
NO PUEDO ASIGNAR



Recursos disponibles

R1	R2	R3
0	0	1

- Tiene un coste significativo asociado
 - ◆ Mantenimiento de la información necesaria y la ejecución del algoritmo de detección.
 - ◆ Potenciales pérdidas inherentes al proceso de recuperación.

DETECCIÓN

- Estructuras necesarias
 - ◆ Matriz de asignación
 - ◆ Matriz de peticiones
 - ◆ Vectores de recursos totales y disponibles
- Al igual que el algoritmo del banquero, realiza una simulación de asignación de recursos, en este caso, para ver si puede satisfacer todas las peticiones actuales.

¿Con qué frecuencia hay que correr el algoritmo de detección?

RECUPERACIÓN

→ Finalizar procesos

- ◆ Todos los intervinientes del deadlock → tiene un alto precio
- ◆ Finalizar de a uno hasta que se solucione el deadlock → tiene más trabajo asociado (elegir a la víctima + correr el algoritmo de detección nuevamente luego de la finalización del proceso)

→ Desalojar recursos

- ◆ Es necesario volver al proceso expropiado a un estado seguro desde el cual pueda reanudar su ejecución.
- ◆ Puede llegar a generar inanición si es que un proceso continuamente es elegido como víctima.

Factores a considerar para seleccionar víctimas:

- ❖ La prioridad y el tipo del proceso.
- ❖ Tiempo actual de ejecución.
- ❖ Cuántos y qué tipo de recursos ha utilizado o necesita.

COMPARACIÓN

	Prevención	Evasión	Detección y recuperación
Flexibilidad en las peticiones	Restringida por la política aplicada	Intermedia, los procesos deben declarar sus peticiones máximas	Flexible, cualquier solicitud puede realizarse
Puede ocurrir deadlock	No	No	Si
Overhead requerido	Por lo general es poco, sólo se define en qué forma se realizan las peticiones	Alto, con cada petición se debe correr el algoritmo del banquero	Intermedio, depende de la frecuencia del algoritmo de detección
Utilización correcta de los recursos	Puede ser muy ineficiente dependiendo la estrategia	Por ser pesimista se puede evitar asignar un recurso.	Puede llegar a ser ineficiente en caso de desalojos frecuentes

DEADLOCK EN LA VIDA REAL



Dudas??

- Es una situación similar al deadlock en la cual un conjunto de procesos no puede progresar en la ejecución de su trabajo pero, en este caso, los procesos siguen ejecutándose.
- Como los procesos no se encuentran bloqueados, es más complicada su detección.

