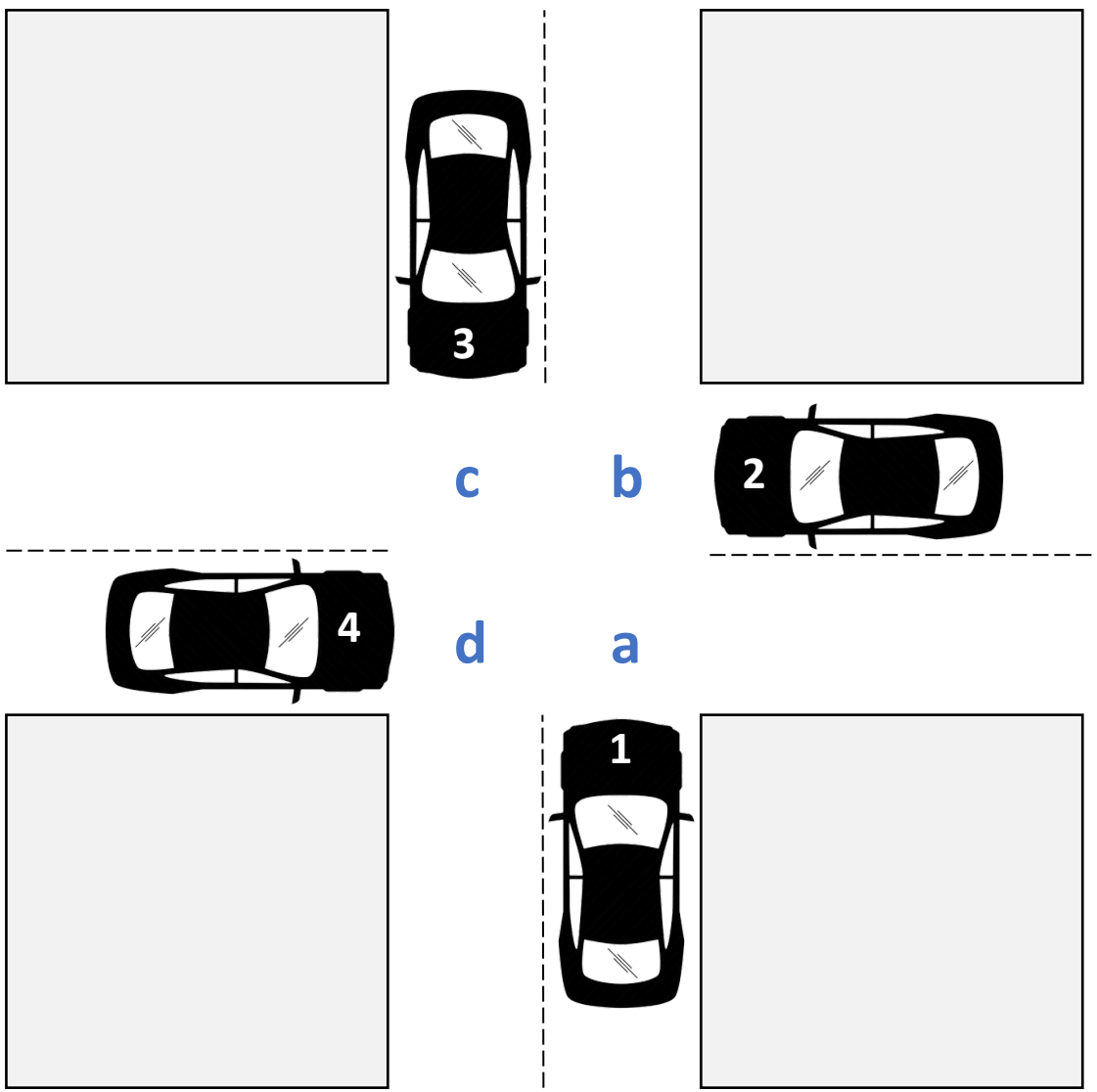
Interbloqueos

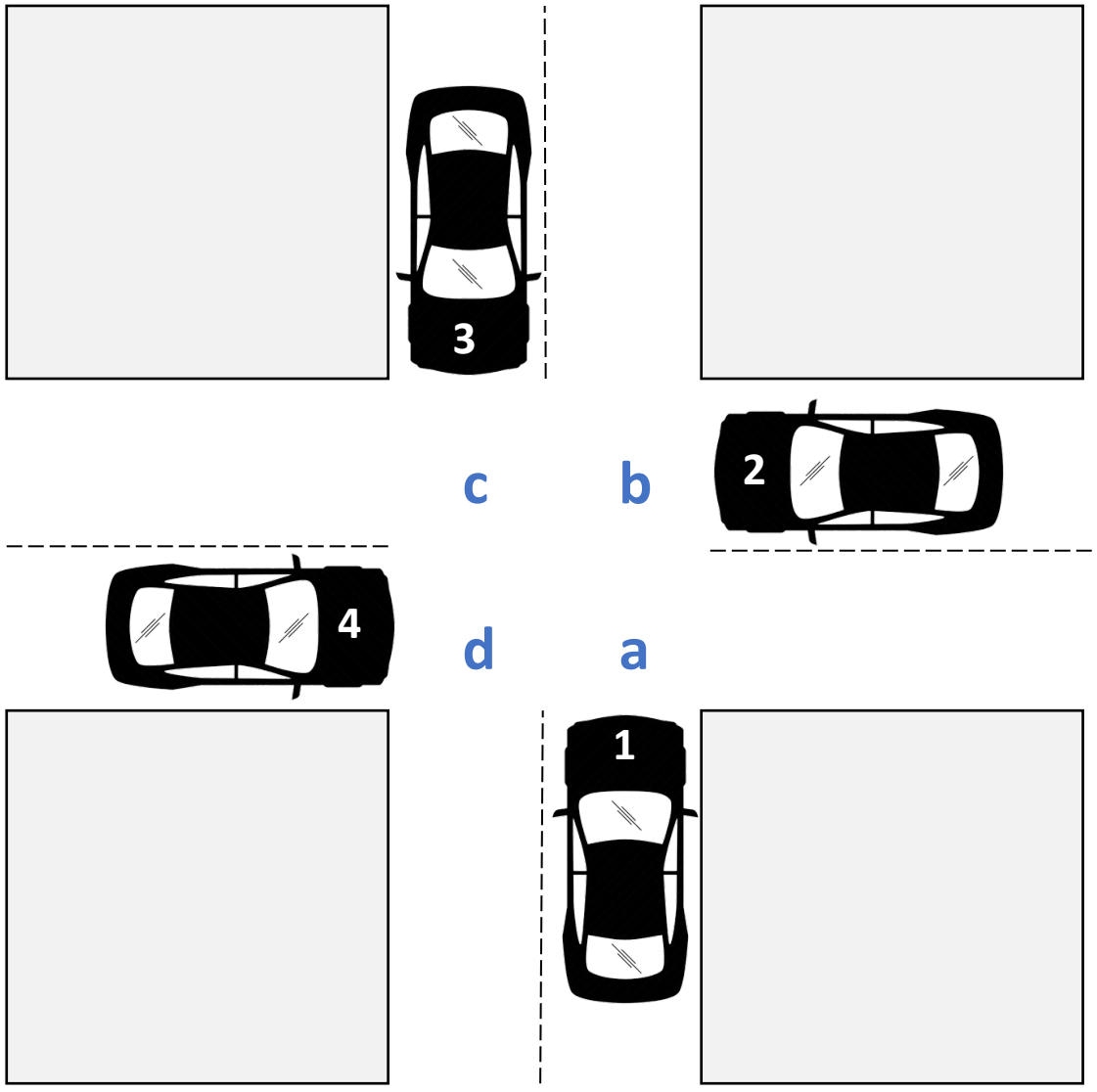
Adaptación de múltiples referencias (ver al final)

Interbloqueos

- Bloqueo permanente de
  - Procesos/hilos que compiten por recursos del sistema
  - Procesos/hilos que se comunican entre sí
- Un conjunto de procesos está en interbloqueo cuando cada proceso del conjunto está bloqueado esperando un evento que solo puede darse por uno de los procesos en el conjunto.
- El S.O no ofrece mecanismos para prevenir interbloqueos
  - Es responsabilidad de programadores evitar esta situación
  - Cómo se adquieren y se liberan recursos
- Problemas difíciles de resolver en entornos de alta concurrencia y paralelismo



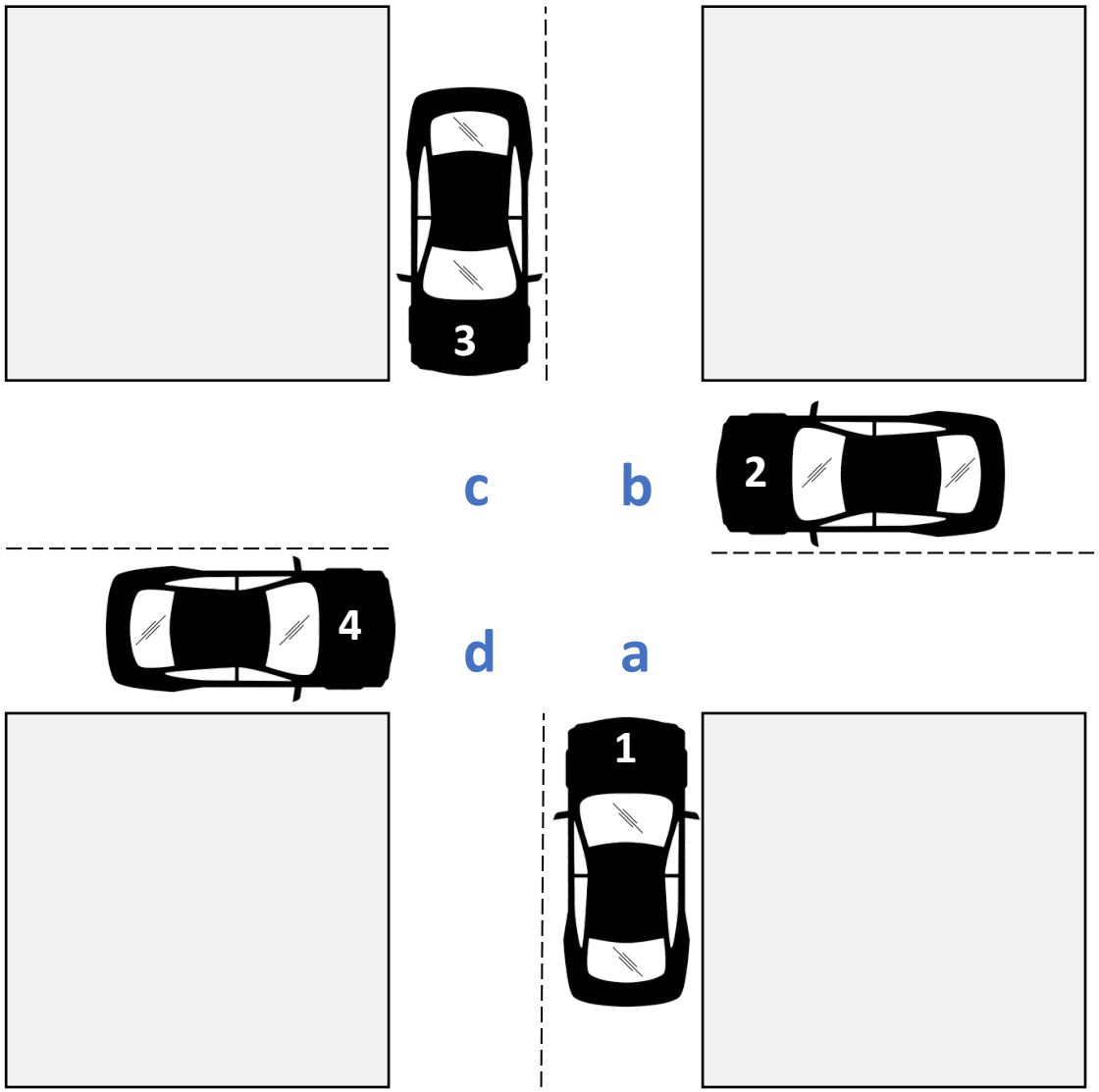
Posible interbloqueo



Interbloqueo

Interbloqueos

- Las intersecciones: **a**, **b**, **c** y **d** son recursos sobre los que se necesita control.
  - Vehículo 1 (norte): necesita intersecciones **a** y **b**
  - Vehículo 2 (oeste): necesita intersecciones **b** y **c**
  - Vehículo 3 (sur): necesita intersecciones **c** y **d**
  - Vehículo 4 (este): necesita intersecciones **d** y **a**



Interbloqueo

Todo el conjunto de procesos **(vehículos)** está esperando a que suceda un evento que solo puede ser generado por alguno de los procesos **(vehículo)** en el conjunto.

Interbloqueos

- S.O. mantiene una tabla para identificar si un recurso está disponible o está ocupado
- Para cada recurso ocupado se registra qué hilo/proceso que lo tiene asignado.
- Si un hilo/proceso solicita un recurso que está asignado a otro hilo
  - Hilo/proceso se añade a la cola de los que esperan por el mismo recurso
- Tipos de recursos que llevan al interbloqueos
  - *Mutex*, semáforos, archivos.
  - Mecanismos de IPC.
  - Recursos físicos como una tarjeta de red.

Estrategias para tratar el problema

- Detección y recuperación
  - Uno de los procesos/hilos debe liberar el recurso
  - En el caso de los vehículos: uno de ellos debe retroceder
  - Debe existir algún protocolo o conjunto de políticas que determinen cuál debe retroceder
- Prevención o predicción
  - Deben existir estrategias preventivas
  - En el caso de los vehículos: poner semáforos. Vehículo se detiene así el camino esté libre

```
pthread mutex_t mutex1;
pthread mutex_t mutex2;
pthread mutex_init(&mutex1,NULL);
pthread mutex_init(&mutex2,NULL);

void *hilo1(void *param) {
    pthread_mutex_lock(&mutex1);
    pthread_mutex_lock(&mutex2);

    /* Región crítica */

    pthread_mutex_unlock(&mutex1);
    pthread_mutex_unlock(&mutex2);
    pthread_exit(0);
}
```

```
void *hilo2(void *param) {
    pthread_mutex_lock(&mutex2);
    pthread_mutex_lock(&mutex1);

    /* Región crítica */

    pthread_mutex_unlock(&mutex1);
    pthread_mutex_unlock(&mutex2);
    pthread_exit(0);
}
```

Se presenta interbloqueo en esta situación de ejecución paralela:

- **`hilo1`** obtiene **`mutex1`** y al mismo tiempo **`hilo2`** obtiene el **`mutex2`**

No se presenta interbloqueo si:

- **`hilo1`** libera **`mutex1`** y **`mutex2`** **ANTES** de que **`hilo2`** obtenga los *mutex*.

```c
void *hilo1(void *param) {
    int done = 0;
    while (!done) {
        pthread_mutex_lock(&mutex1);
        if (pthread_mutex_trylock(&mutex2)) {

            /* Región crítica */

        pthread_mutex_unlock(&mutex2);
        pthread_mutex_unlock(&mutex1);
        done = 1;
    }
    else
        pthread_mutex_unlock(&mutex1);
    }
    pthread_exit(0);
}
```

```c
void *hilo2(void *param) {
    int done = 0;
    while (!done) {
        pthread_mutex_lock(&mutex2);
        if (pthread_mutex_trylock(&mutex1)) {

        /* Región crítica */

        pthread_mutex_unlock(&mutex1);
        pthread_mutex_unlock(&mutex2);
        done = 1;
    }
    else
        pthread_mutex_unlock(&mutex2);
    }
    pthread_exit(0);
}
```

Se presenta situación de *livelock*, si:

- **`hilo1`** obtiene **`mutex1`** y al mismo tiempo **`hilo2`** obtiene el **`mutex2`**
- Cada hilo invoca (al tiempo) **`pthread_mutex_trylock()`**, lo cual falla haciendo que cada *mutex* se libere
- Esta secuencia se repite indefinidamente

No se presenta *livelock*, si:

- Se espera un tiempo aleatorio **ANTES** de intentarlo de nuevo.

Representación como grafo de asignación

- Grafo de asignación de recursos en el sistema
  - Conjunto de vértices
  - Conjunto de arcos
- Conjuntos de vértices
  - Nodos que representan hilos/procesos activos en el sistema

$$T = \{T_1, T_2, \dots, T_n\}$$

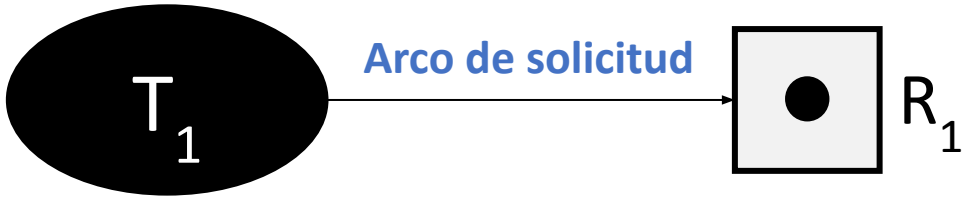
  - Nodos que representan los tipos de recursos

$$R = \{R_1, R_2, \dots, R_m\}$$

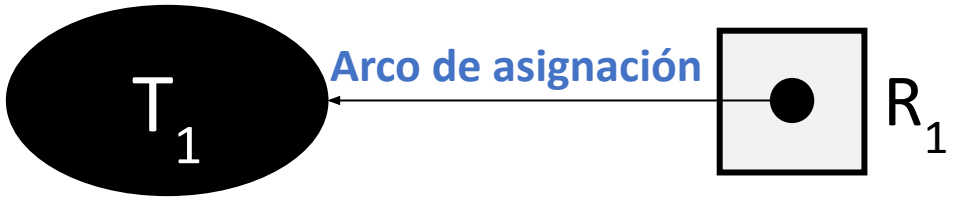
Representación como grafo de asignación

- Conjunto de arcos
  - Arco dirigido desde T_i hasta R_i: $T_i \rightarrow R_j$
    - El hilo/proceso T_i solicitó **una instancia** de R_j y está esperando por el recurso.
    - Arco de solicitud
  - Arco dirigido desde R_j hasta T_i: $R_j \rightarrow T_i$
    - **Una instancia** de R_j ha sido asignada al hilo/proceso T_i.
    - Arco de asignación

Representación como grafo de asignación

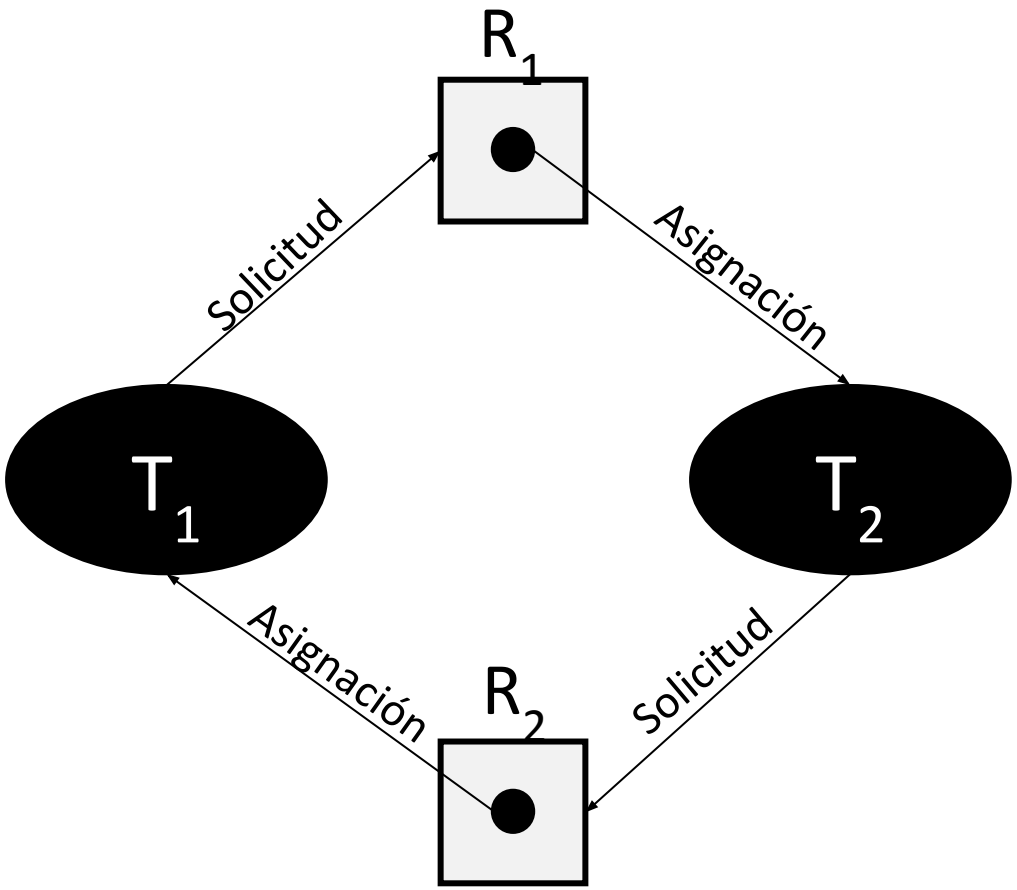


Indica que el recurso R_1 es solicitado por el proceso/hilo T_1

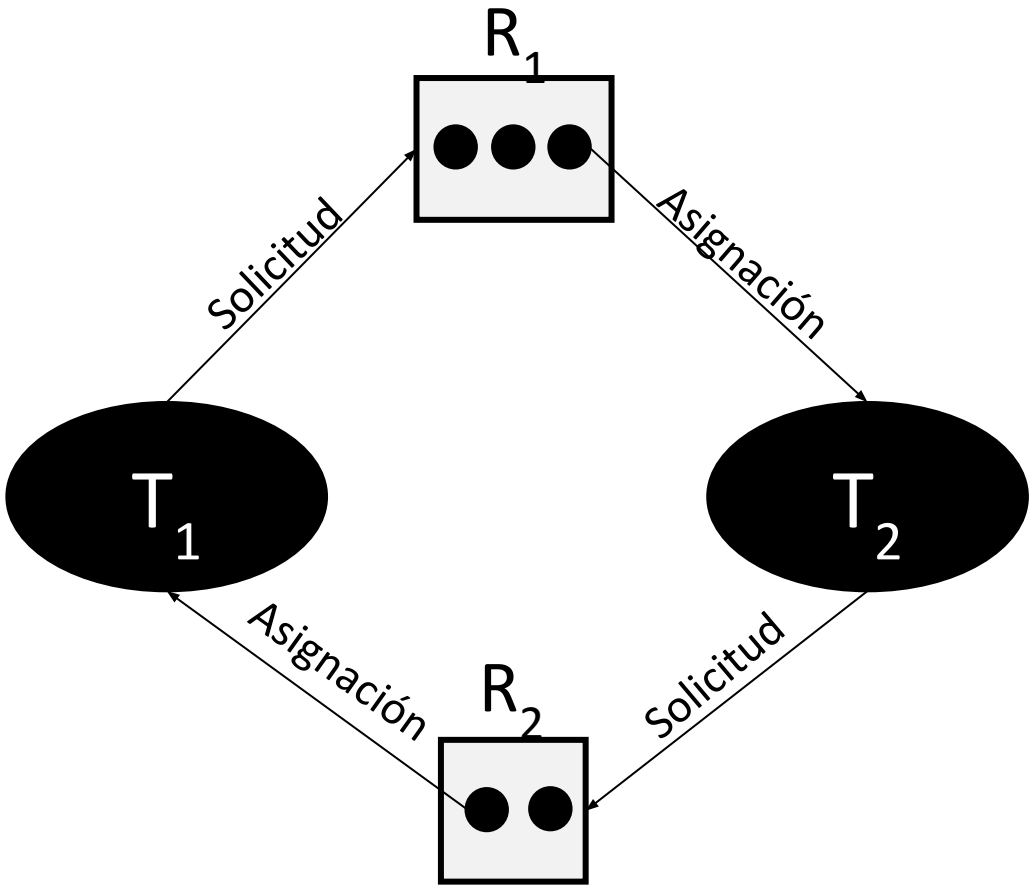


Indica que una instancia del recurso R_1 es asignada al proceso/hilo T_1

Representación como grafo de asignación



Espera circular: *deadlock*



No hay *deadlock*

Representación como grafo de asignación

- Cuando un hilo/proceso T_i solicita una instancia del recurso de tipo R_j
  - Se inserta un arco de solicitud en el grafo.
- Cuando se puede cumplir el requerimiento de T_i sobre la instancia del recurso de tipo R_j
  - El arco de solicitud se transforma en arco de asignación.
- Cuando el hilo/proceso no necesita el acceso al recurso, libera el recurso
  - El arco de asignación se elimina del grafo.

Representación como grafo de asignación

- Semántica de la representación
  - Círculos/óvalos: Procesos o hilos. Un circulo/óvalo por cada proceso/hilo.
  - Rectángulos: Recursos. Un rectángulo por cada tipo de recurso.
  - Puntos dentro de rectángulos: número de instancias disponibles de un recurso
- Restricción de asignación
  - Número de arcos que salen desde R_j debe ser menor o igual que su inventario (unidades disponibles)
- Restricción de solicitud
  - Por cada pareja T_i, R_j se debe cumplir que número de arcos desde R_j a T_i más el número de arcos de T_i a R_j debe ser menor o igual que el inventario.

Condiciones para un interbloqueo

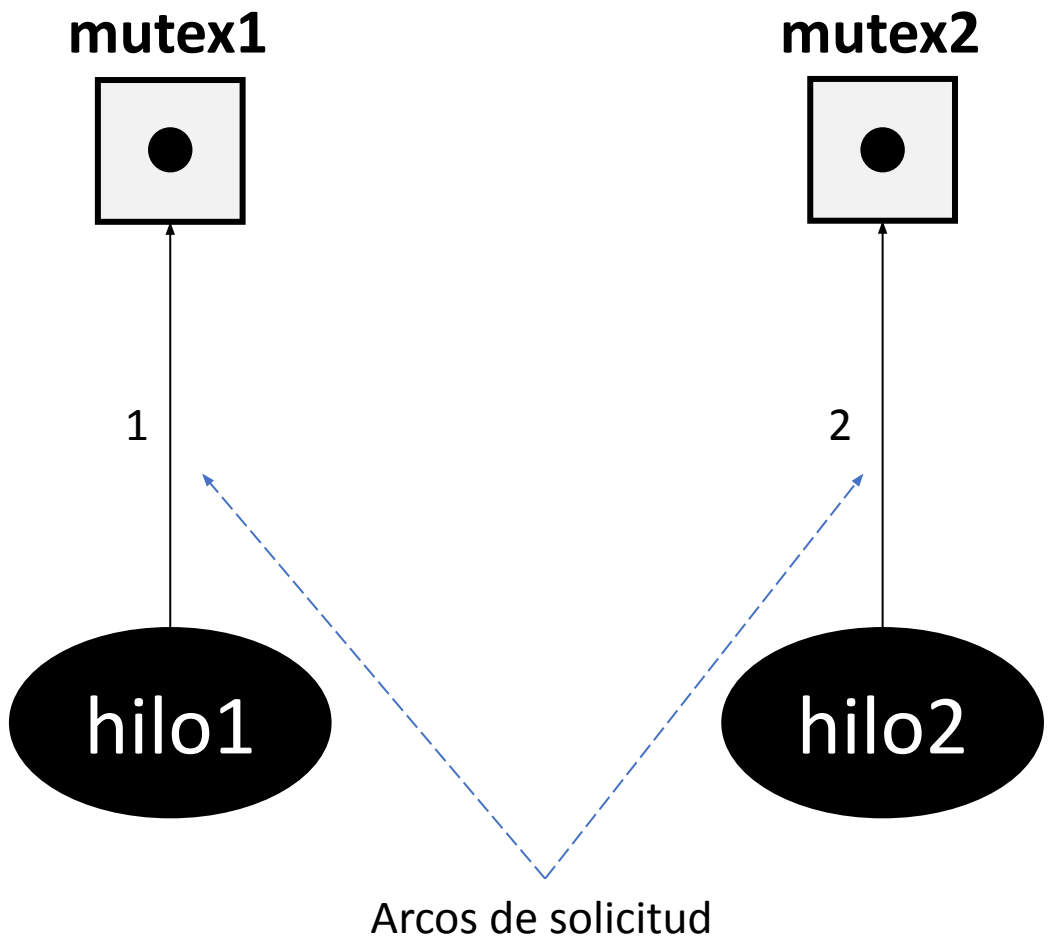
1. **Exclusión mutua**
   - Existe un recurso compartido que se está usando por un hilo/proceso a la vez.
   - Los que necesiten el recurso deben esperar hasta que se libere.
2. **Retención y espera**
   - Un hilo mantiene/bloquea un recurso compartido y a la vez espera a que se liberen otros recursos bloqueados por otros hilos.
3. **No expropiación**
   - Los recursos son liberados a voluntad por quien los usa.
4. **Espera circular**
   - T_1 espera por recurso asignado a T_2 y T_2 espera por un recurso asignado a T_1

Representación como grafo de asignación

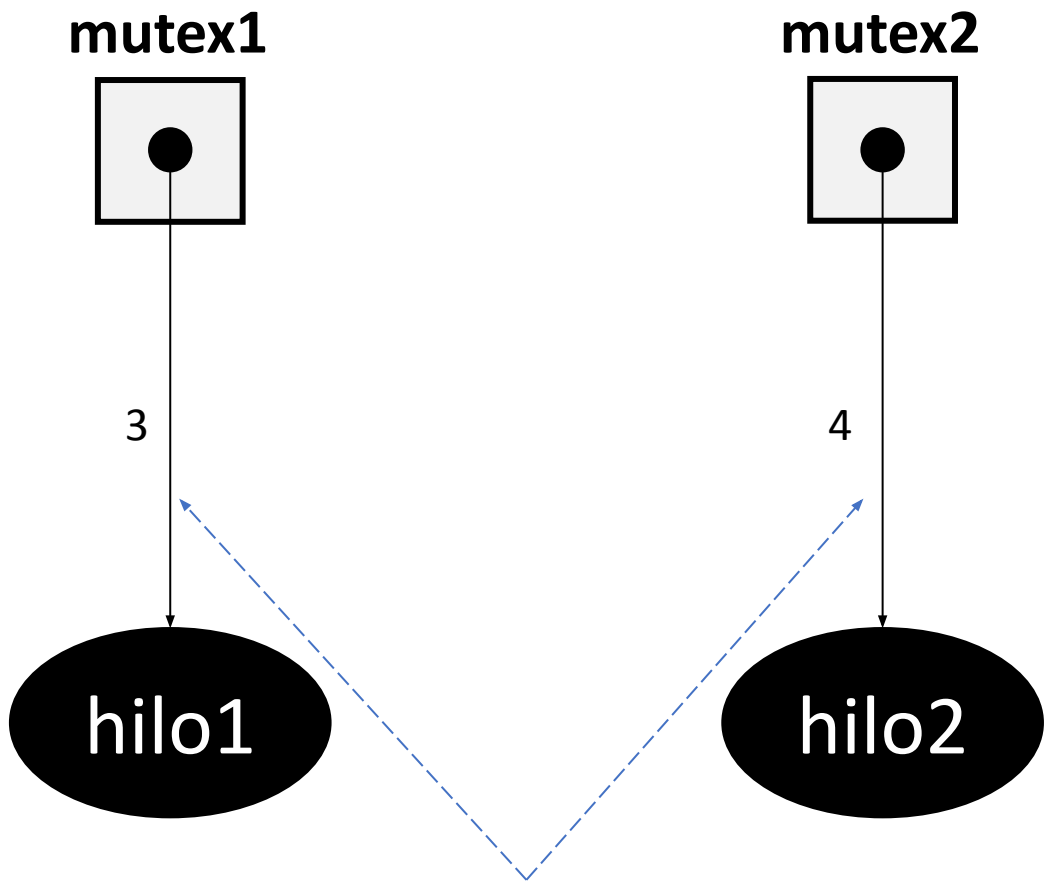
- Ejemplo con el código que produce *deadlock* con dos hilos

1. Hilo1: solicita bloqueo de mutex1
2. Hilo2: solicita bloqueo de mutex2
3. Hilo1: bloquea mutex1
4. Hilo2: bloquea mutex2
5. Hilo1: solicita bloqueo de mutex2 □ Se bloquea esperando mutex2
6. Hilo2: solicita bloqueo de mutex1 □ Se bloquea esperando mutex1

1. Hilo1: solicita bloqueo de mutex1
2. Hilo2: solicita bloqueo de mutex2

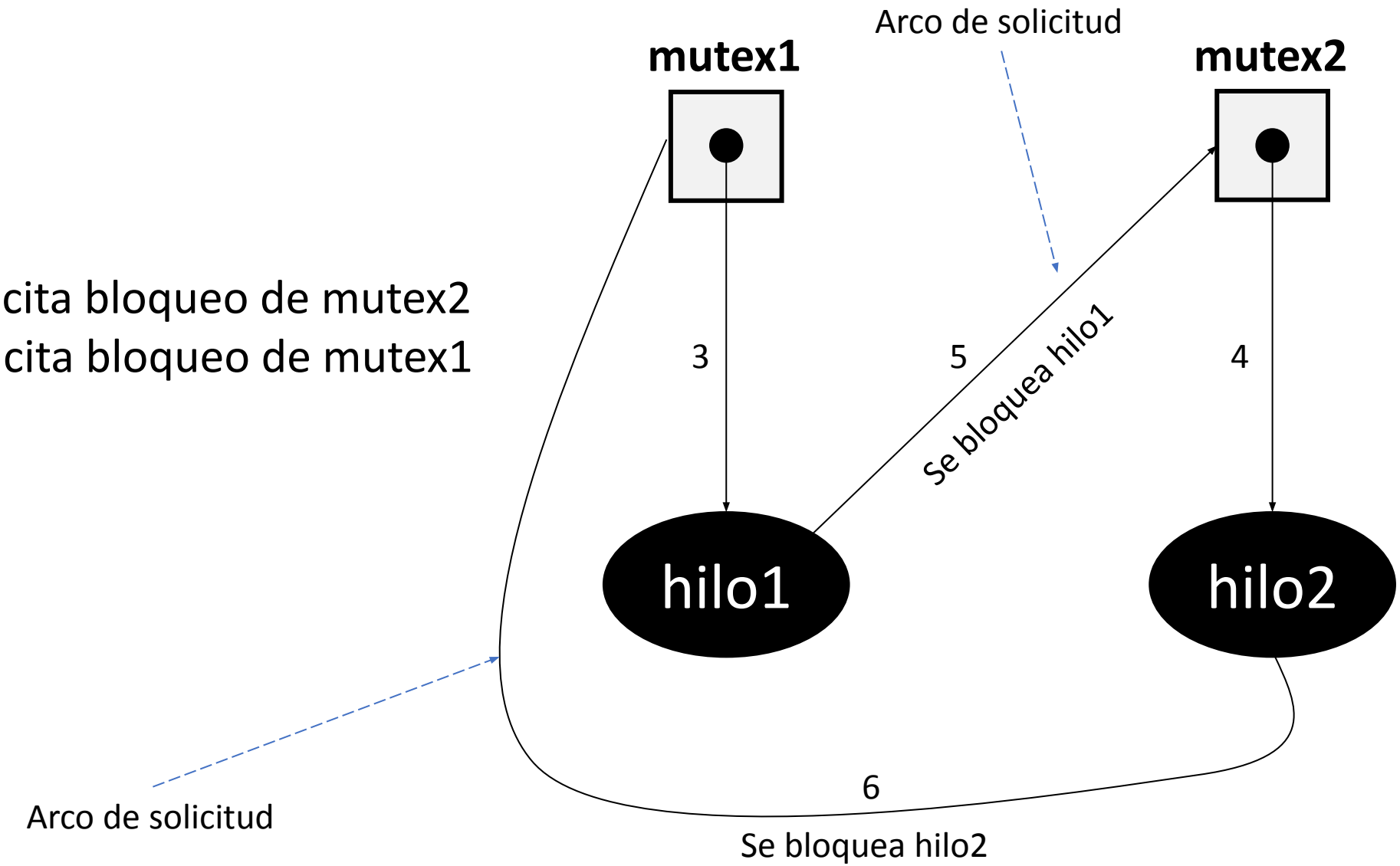


3. Hilo1: bloquea mutex1
4. Hilo2: bloquea mutex2



Arcos de asignación porque los recursos están disponibles. Note que salen desde el número de instancia de cada recurso

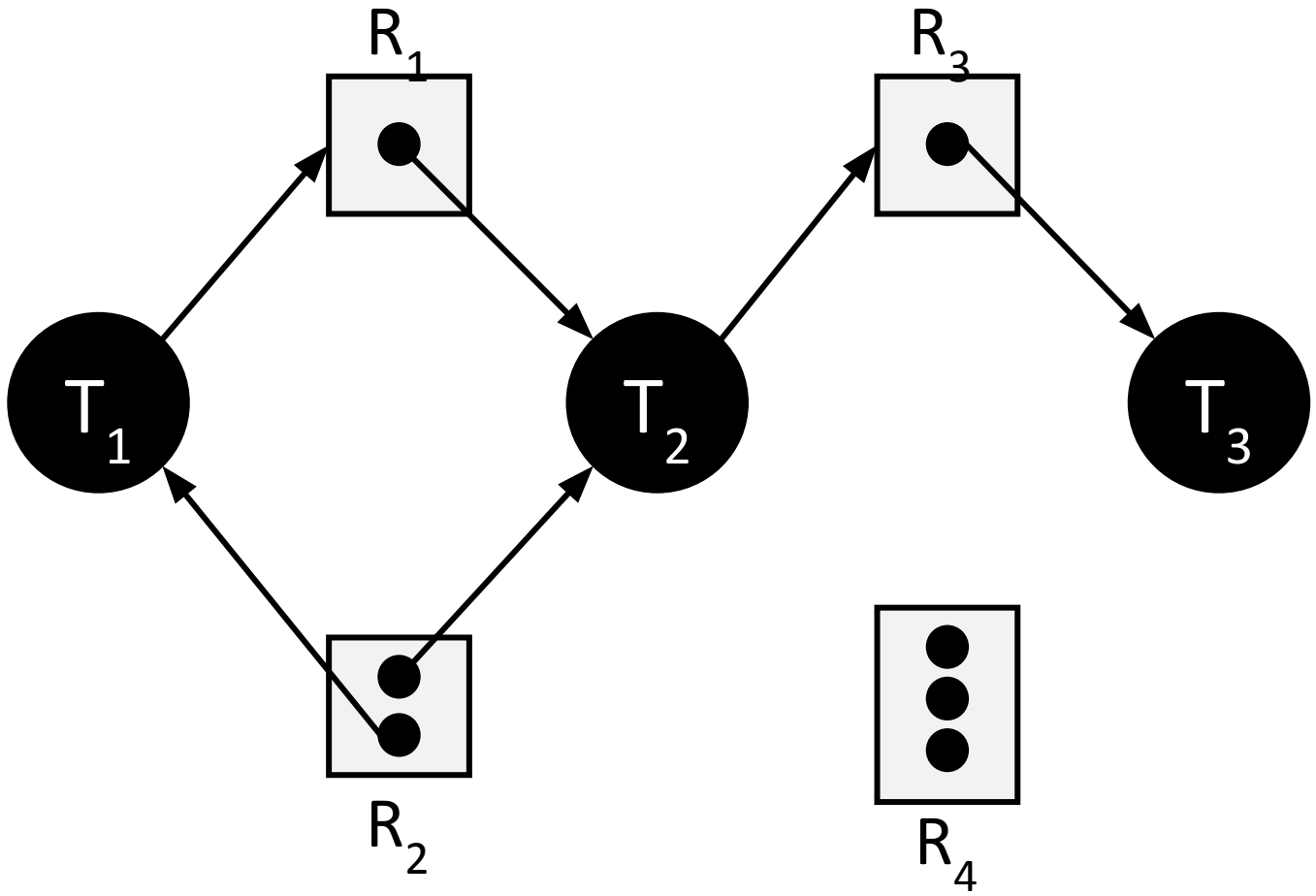
5. Hilo1: solicita bloqueo de mutex2
6. Hilo2: solicita bloqueo de mutex1



Representación como grafo de asignación

- Considere los siguientes conjuntos
  - $T = \{T_1, T_2, T_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{T_1 \rightarrow R_1, T_2 \rightarrow R_3, R_1 \rightarrow T_2, R_2 \rightarrow T_2, R_2 \rightarrow T_1, R_3 \rightarrow T_3\}$
- Número de instancias
  - $1 \times R_1$
  - $2 \times R_2$
  - $1 \times R_3$
  - $3 \times R_4$

Representación como grafo de asignación



Recorridos

- $T_1 \rightarrow R_1 \rightarrow T_2 \rightarrow R_3 \rightarrow T_3$
- $R_2 \rightarrow T_2 \rightarrow R_3 \rightarrow T_3$

No hay ciclos: no hay interbloqueo

Representación como grafo de asignación

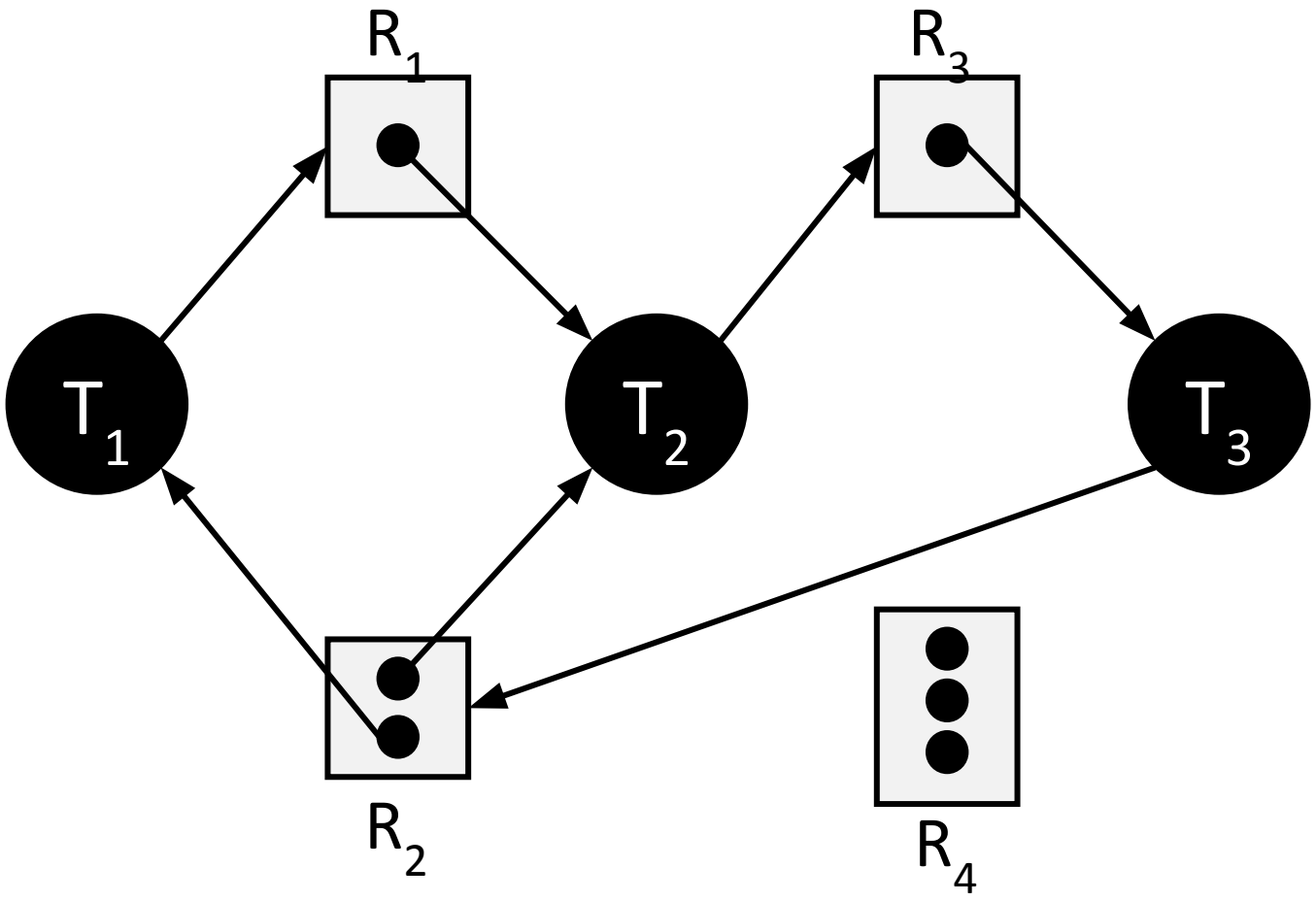
- Si grafo no tiene ciclos, entonces no hay interbloqueo.
- Si el grafo tiene un ciclo, **podría** existir interbloqueo.
- Si de cada R_j hay una sola instancia, entonces el ciclo implica que ocurrió interbloqueo.
- Si el ciclo involucra a únicamente a un conjunto de recursos, de los cuales solo hay una instancia, entonces ocurrió un interbloqueo.
  - Ciclo en el grafo: condición necesaria y suficiente para un interbloqueo
- Si existen varias instancias de un mismo recurso, un ciclo no necesariamente implica que ocurrió un interbloqueo
  - Ciclo en el grafo: condición necesaria pero no suficiente para un interbloqueo

Representación como grafo de asignación

- Considere los siguientes conjuntos
  - $T = \{T_1, T_2, T_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{T_1 \rightarrow R_1, T_2 \rightarrow R_3, R_1 \rightarrow T_2, R_2 \rightarrow T_2, R_2 \rightarrow T_1, R_3 \rightarrow T_3, \boldsymbol{T_3 \rightarrow R_2}\}$
- Número de instancias
  - $1 \times R_1$
  - $2 \times R_2$
  - $1 \times R_3$
  - $3 \times R_4$

Se agregó está solicitud

Representación como grafo de asignación



Primer ciclo

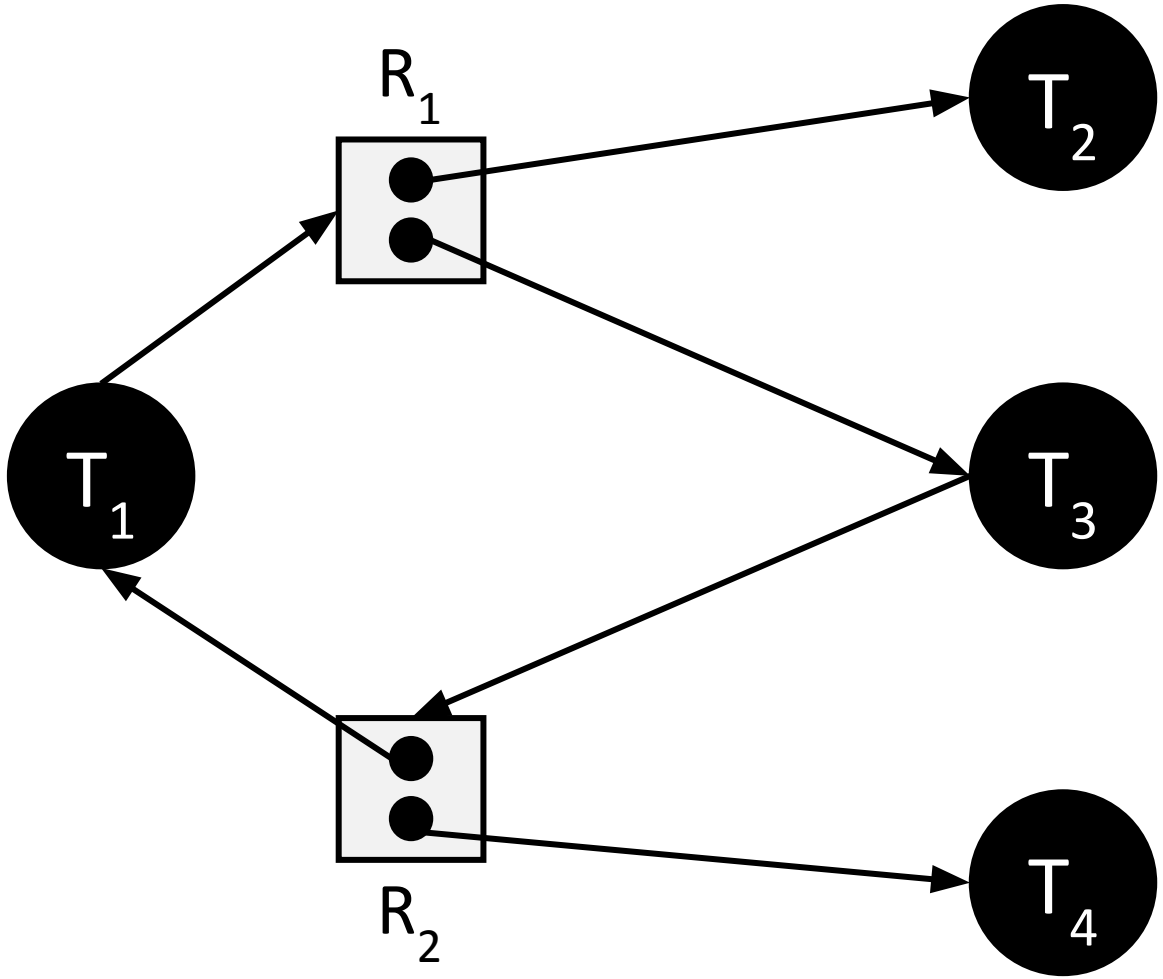
- $T_1 \rightarrow R_1 - T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$

Segundo ciclo

- $R_2 \rightarrow T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2$

T_1, T_2, T_3 Están en interbloqueo

Representación como grafo de asignación



Recorridos

- $T_1 \rightarrow R_1 \rightarrow T_2$
- $\boldsymbol{T_1} \rightarrow R_1 \rightarrow T_3 \rightarrow R_2 \rightarrow \boldsymbol{T_1}$
- $T_1 \rightarrow R_1 \rightarrow T_3 \rightarrow R_2 \rightarrow T_4$

Existe un ciclo pero si T_4 libera voluntariamente R_2, se rompe el ciclo.
Podría existir un interbloqueo pero no lo hay.

Estrategias para tratar el interbloqueo

- Ignorar el problema
  - Windows, Linux
- Diseñar y usar un protocolo para que nunca se entre en interbloqueo
  - Estrategias para prevenir interbloqueos
  - Estrategias para evitar interbloqueos
- Permitir entrar en interbloqueo, detectarlo y recuperar el sistema del interbloqueo
  - DBMS

Referencias

- Carretero Pérez, J., De Miguel Anasagasti, P., García Carballeira, F., & Pérez Costoya, F. (2001). Interbloqueos. In *Sistemas operativos. Una Visión Aplicada* (pp. 309–325). McGraw Hill.
- Silberschatz, A., Baer Galvin, P., & Gagne, G. (2018). Deadlocks. In *Operating Systems Concepts* (10th ed., pp. 317–327). John Wiley & Sons, Inc.
- Stallings, W. (2018). Concurrency: Deadlock and Starvation. In *Operating Systems Internals and Design Principles* (9th ed., pp. 289–299). Pearson Education Limited.