udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Interbloqueo (*Deadlock*)

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er. Semestre 2015

Adaptado de Dave Eckhardt

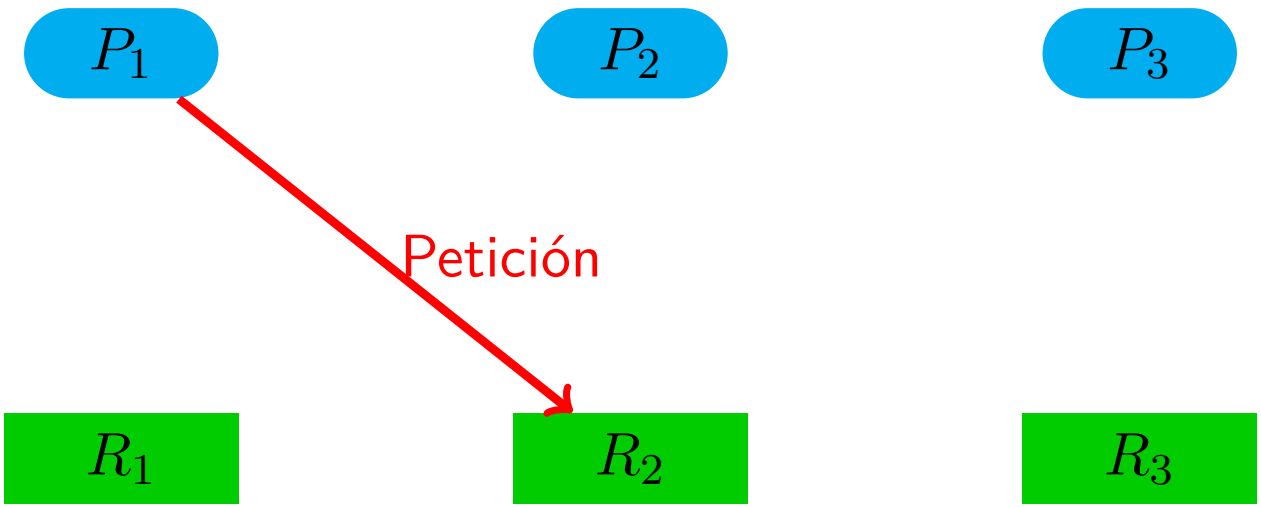
Recapitulando

- Dos operaciones fundamentales
  - Exclusión mutua para secuencias que deben ser atómicas
  - Debemos descalendarizar atómicamente (y despertar después)
- Mutex–variables de condición (estilo de `pthreads` POSIX)
  - Dos objetos
  - Uno para cada operación
- Semáforos y monitores
  - Semáforo: un objeto
  - Monitor: objetos generados transparentemente por el compilador
  - Las mismas ideas están encapsuladas

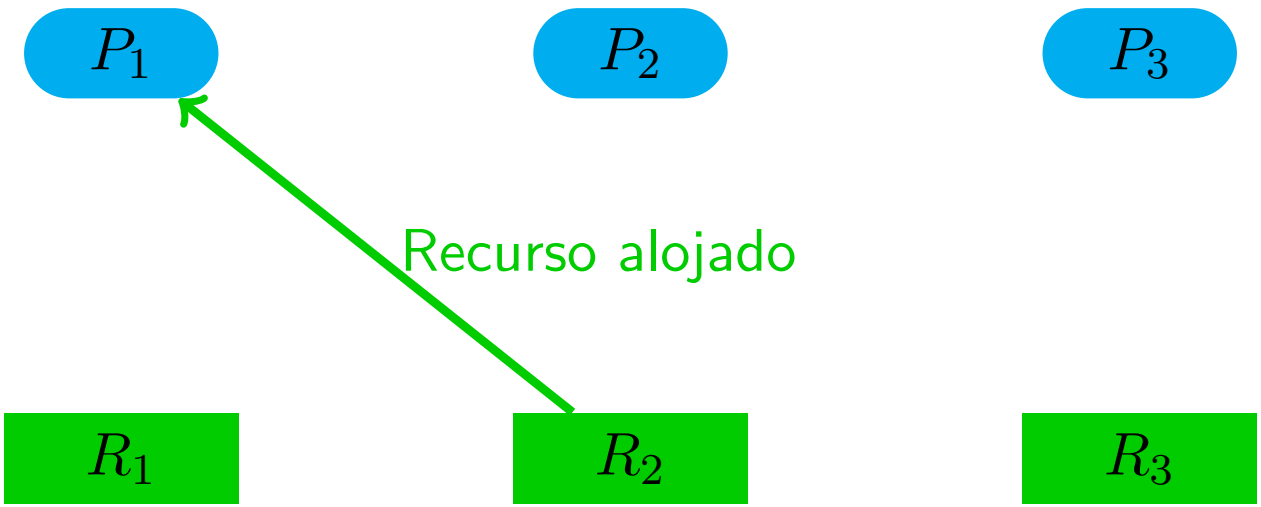
Objetivos de la clase

- Grafo de recursos y de los procesos
- ¿Qué es un interbloqueo?
- Prevención de interbloqueos
- No hablaremos hoy de
  - Evitar interbloqueos
  - Recuperarnos de los interbloqueos

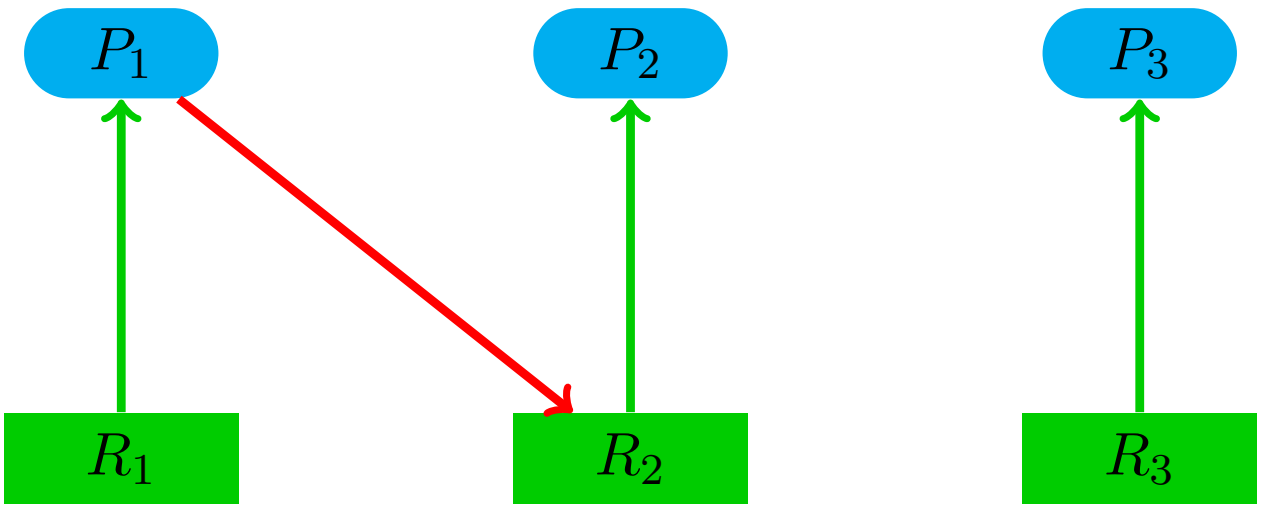
Grafo de recursos y procesos



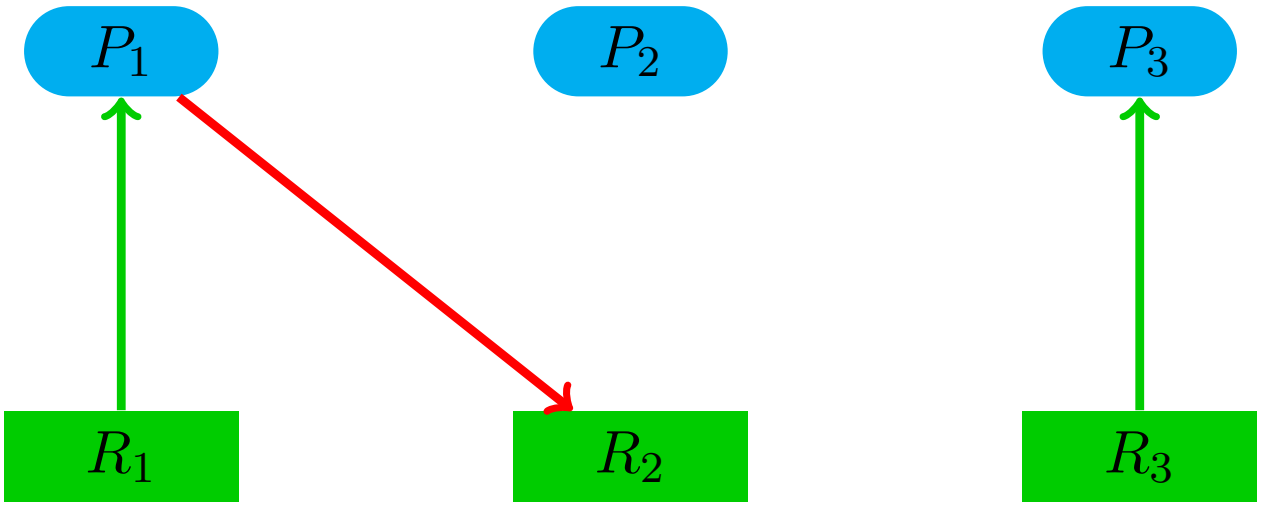
Grafo de recursos y procesos



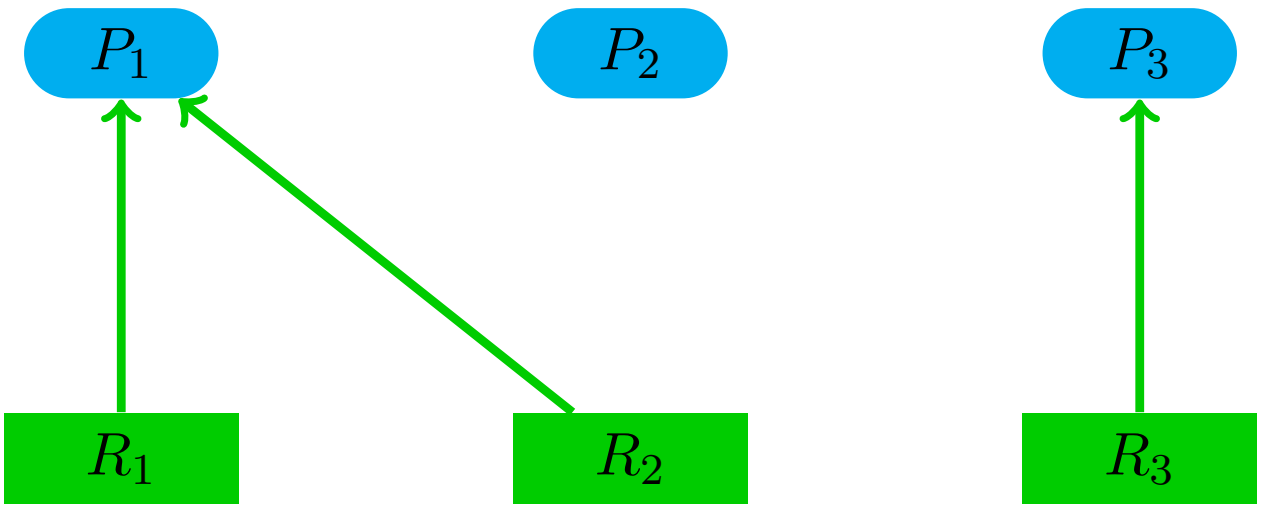
Espera de recursos



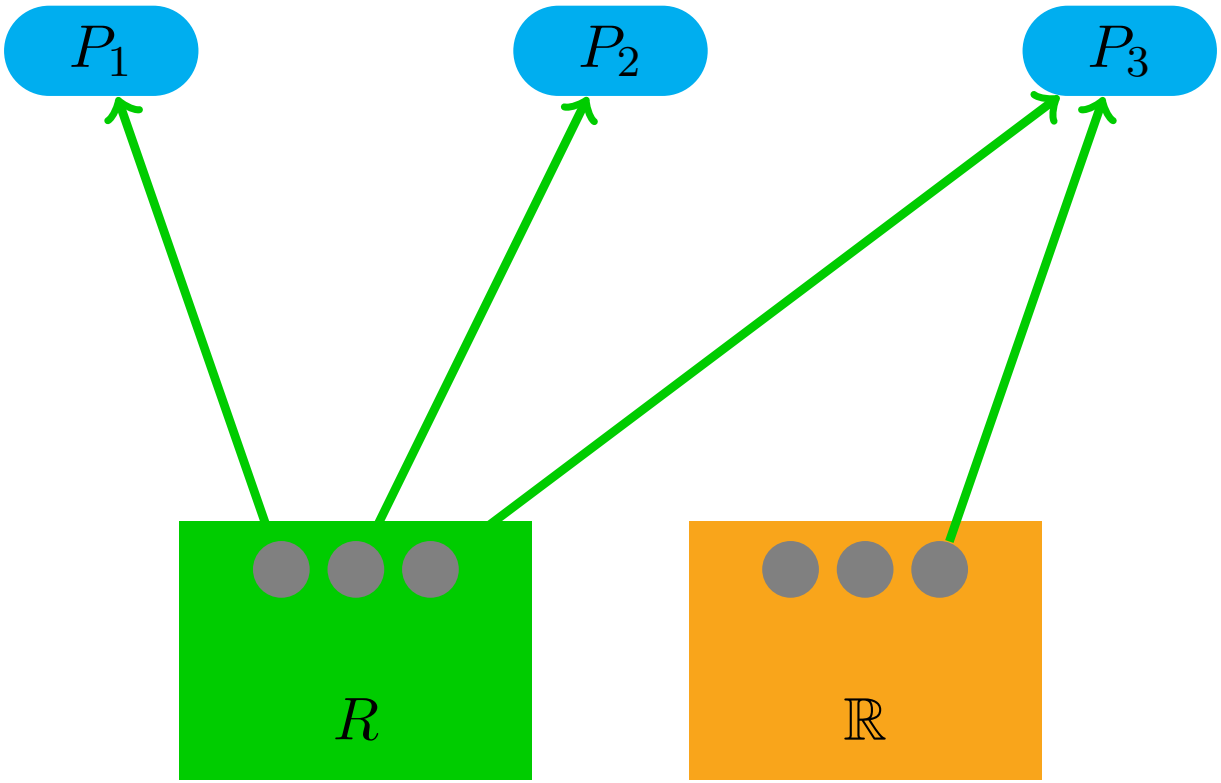
Liberar recursos



Realojar recursos



Múltiples instancias de recursos



Interbloqueo

- Bloqueo mutuo, interbloqueo, o *deadlock*
  - Conjunto de N procesos
  - Cada uno está esperando un evento
  - . . . que puede ser causado solo por otro proceso del mismo conjunto
- Cada proceso esperará por siempre

Ejemplos de interbloqueo

- Forma simple
  - Proceso 1 tiene la impresora (mantiene un seguro —*lock*—), espera acceso al disco
  - Proceso 2 tiene el disco, espera la impresora
- Menos obvias
  - Tres discos
  - Tres procesos
    - Cada uno tiene un disco
    - Cada uno quiere uno más
  - No hay más recursos, pero no puedo soltar mi recurso
  - No podemos avanzar el trabajo

Requerimientos de un interbloqueo

- Exclusión mutua
- Obtener un recurso y esperar
- No hay interrupciones
- Espera circular

Exclusión mutua

- Los recursos no son seguros para ejecución multi-hilo
- Deben ser alojados para un proceso (hilo) en un momento dado
- No pueden ser compartidos
  - Interrupción de tiempo programable
  - No puede tener distintos tiempos para cada proceso

Obtener y esperar

- Los procesos obtienen un recurso y esperan por más

```
mutex_lock(&m1)
mutex_lock(&m2);
mutex_lock(&m3);
```

- Este procedimiento para asegurar recursos es típico

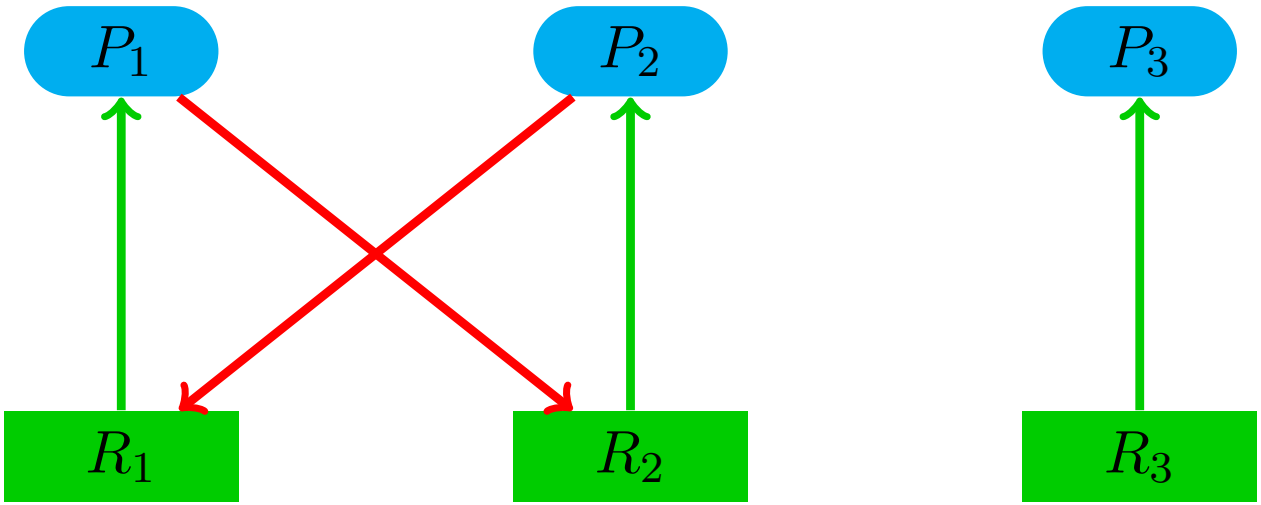
No hay interrupciones

- No podemos forzar a un proceso para que entregue los procesos que tiene asegurados
- No podemos interrumpir a un proceso que está
  - usando la impresora, e imprimir nuestro trabajo
  - escribiendo en el disco, y escribir nuestros datos
- Solución obvia
  - Los controladores (drivers) de los dispositivos no permiten accesos simultáneos
  - Si no podemos abrirlo, no podemos adquirirlo

Espera circular

- Proceso 0 necesita algo que tiene el Proceso 4
  - Proceso 4 tiene algo que necesita el Proceso 7
  - Proceso 7 tiene algo que necesita el Proceso 5
  - Proceso 5 tiene algo que necesita el Proceso 0
- Describe un grafo de recursos que posee un ciclo

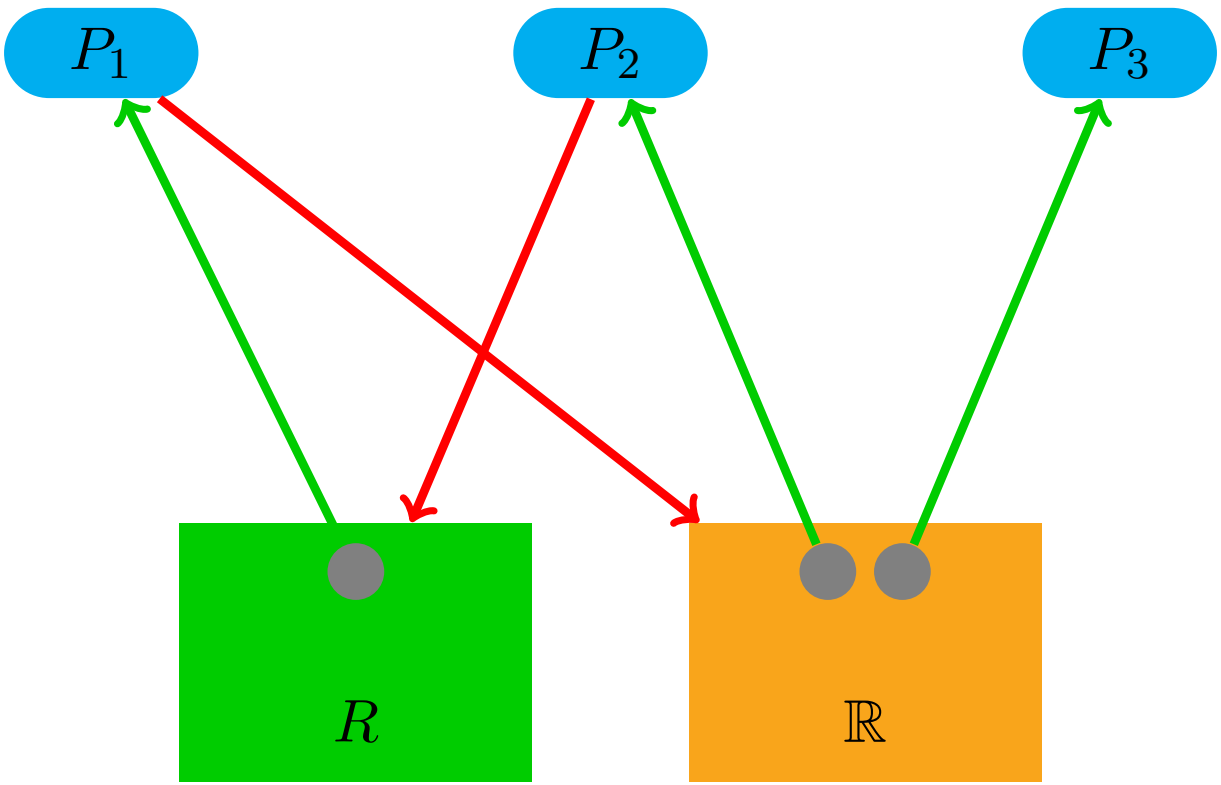
Ciclo en el grafo de recursos



Requerimientos de un interbloqueo

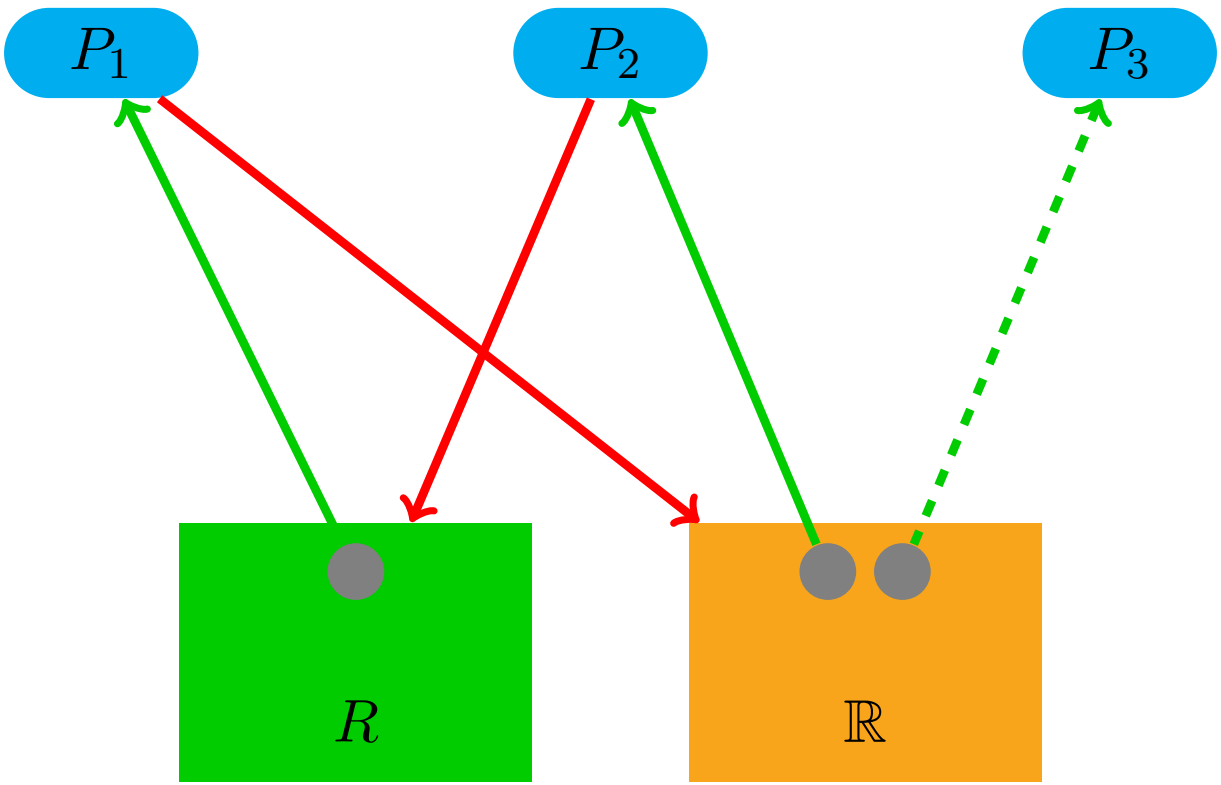
- Exclusión mutua
- Obtener un recurso y esperar
- No hay adquisición por adelantado
- Espera circular
- Necesitamos los cuatro anteriores para tener un deadlock

Ciclos en múltiples instancias de recursos



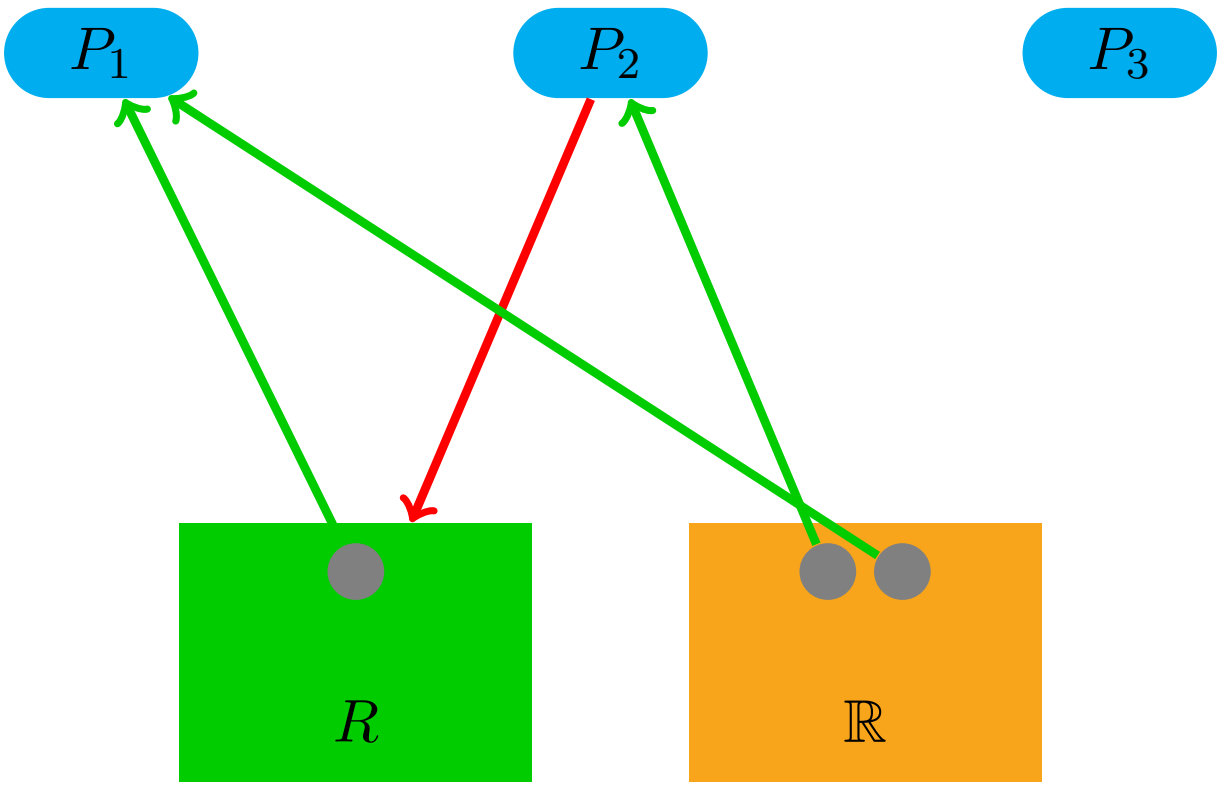
Ciclos en múltiples instancias de recursos

- Podemos solucionar la situación



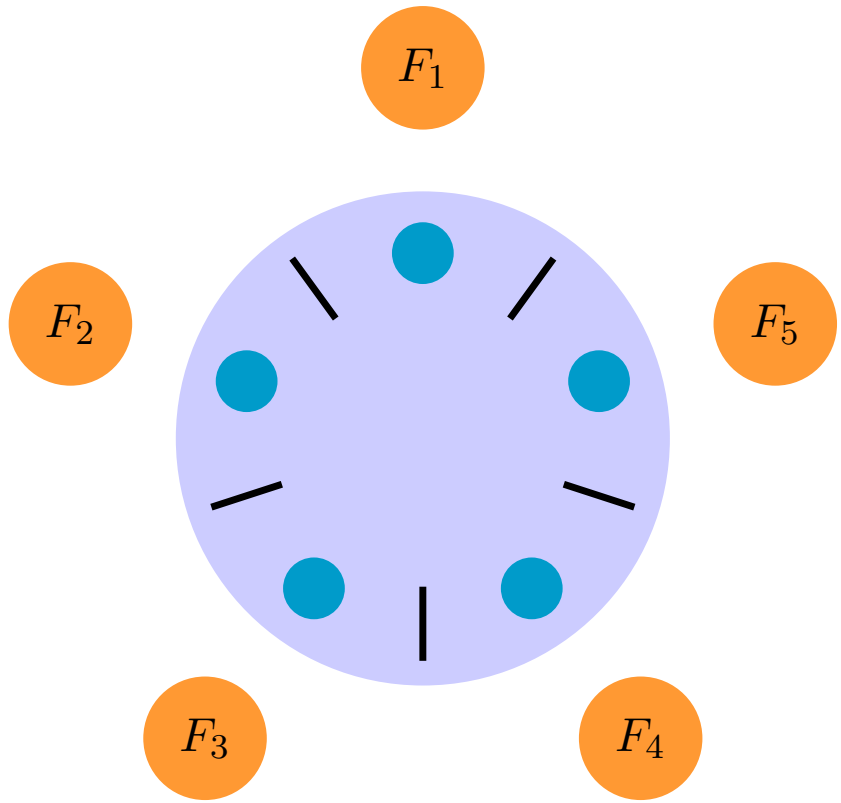
Ciclos en múltiples instancias de recursos

- Y rompemos el ciclo



Los filósofos comensales

- Procesos
  - 5, uno por filósofo
- Recursos
  - 5 platos (dedicado a cada filósofo, no hay carrera podemos ignorar)
  - 5 palillos (1 adyacente a un par de filósofos)
- El ejemplo ilustra
  - Contención
  - Inanición (*starvation*)
  - Interbloqueo (*deadlocks*)

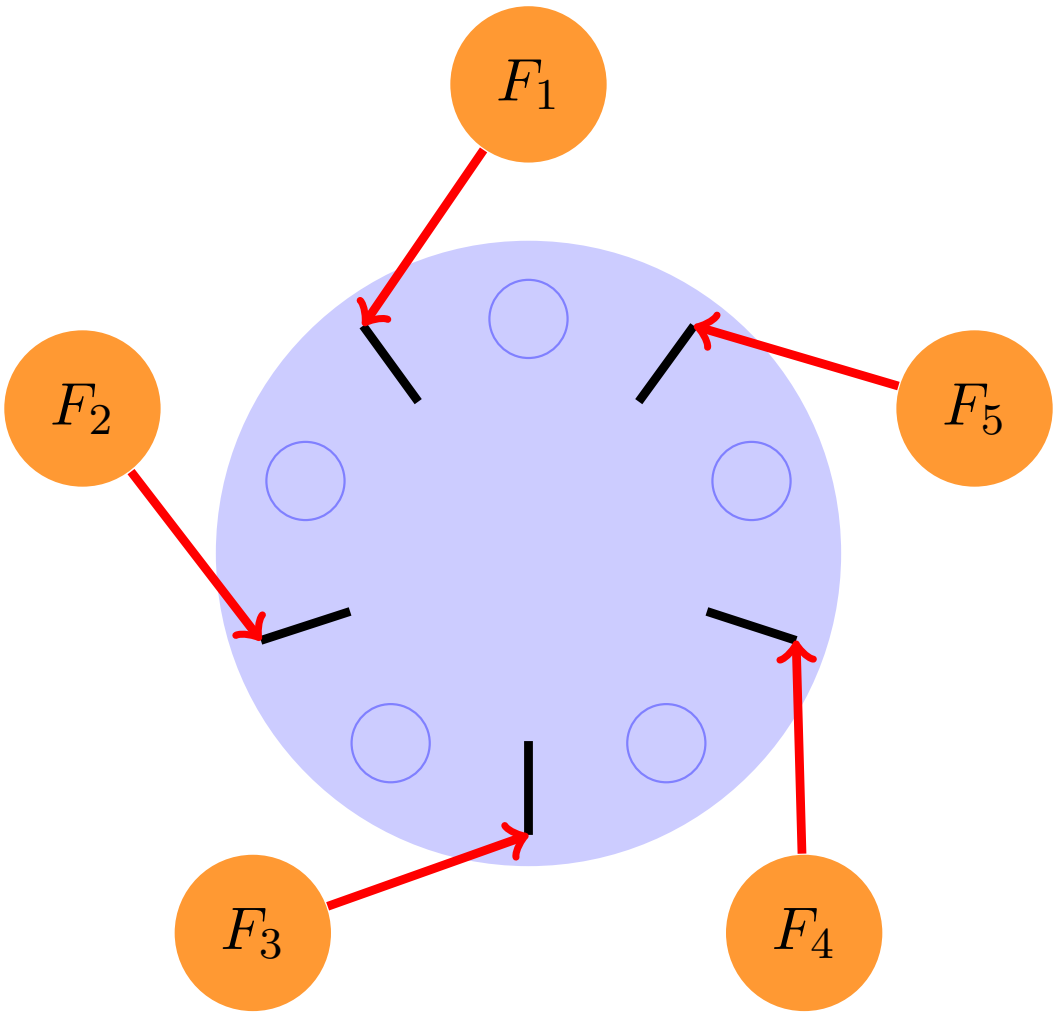


Reglas para comer

- Esperar hasta que el palillo de la derecha esté libre, tomarlo
- Esperar hasta que el palillo de la izquierda esté libre, tomarlo
- Comer por un momento
- Colocar los palillos en la mesa

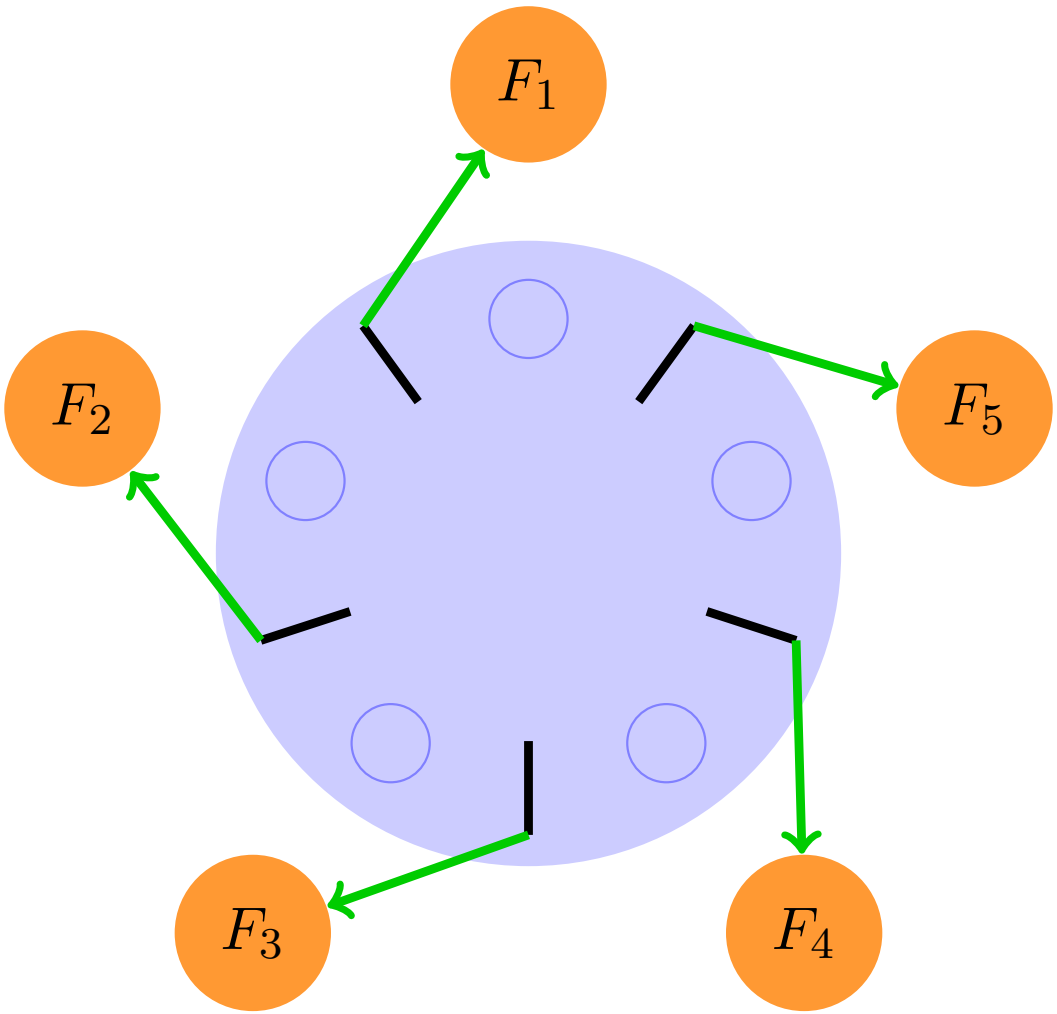
Los filósofos comensales

Interbloqueo



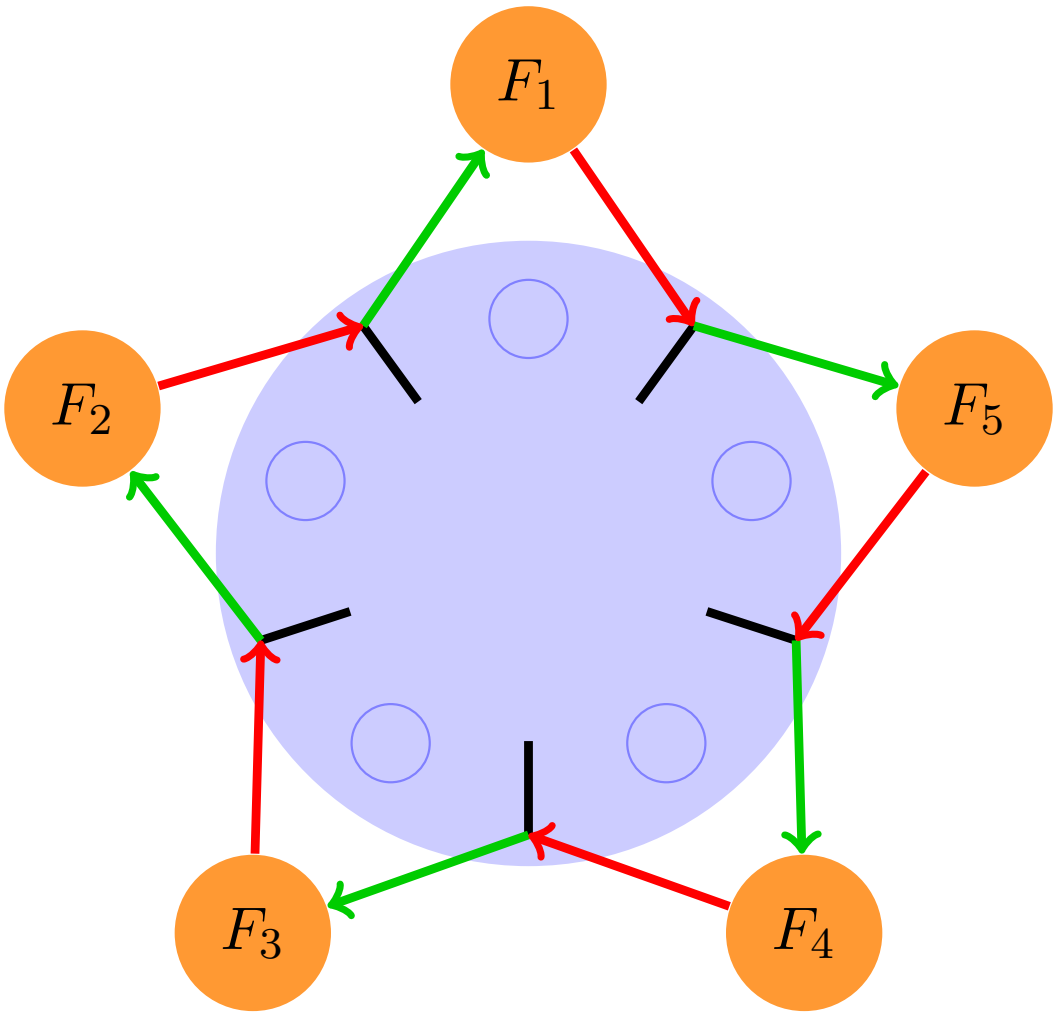
Los filósofos comensales

Interbloqueo



Los filósofos comensales

Interbloqueo



Estado y variables

```
int chopstick[5] = {-1}; // ID del dueño del palillo
condition avail[5]; // nuevo disponible
mutex table = {available};

// Busquemos el palillo a la derecha y la izquierda
// diner: ID del filósofo
right = diner; // 3 -> 3
left = (diner + 4) % 5; // 3 -> 7 -> 2
```

start_eating(int diner)

```
void start_eating(int diner) {
  mutex_lock(table);

  while (stick[right] != -1)
    condition_wait(avail[right], table);
  stick[right] = diner;

  while (stick[left] != -1)
    condition_wait(avail[left], table);
  stick[left] = diner;

  mutex_unlock(table);
}
```

done_eating(int diner)

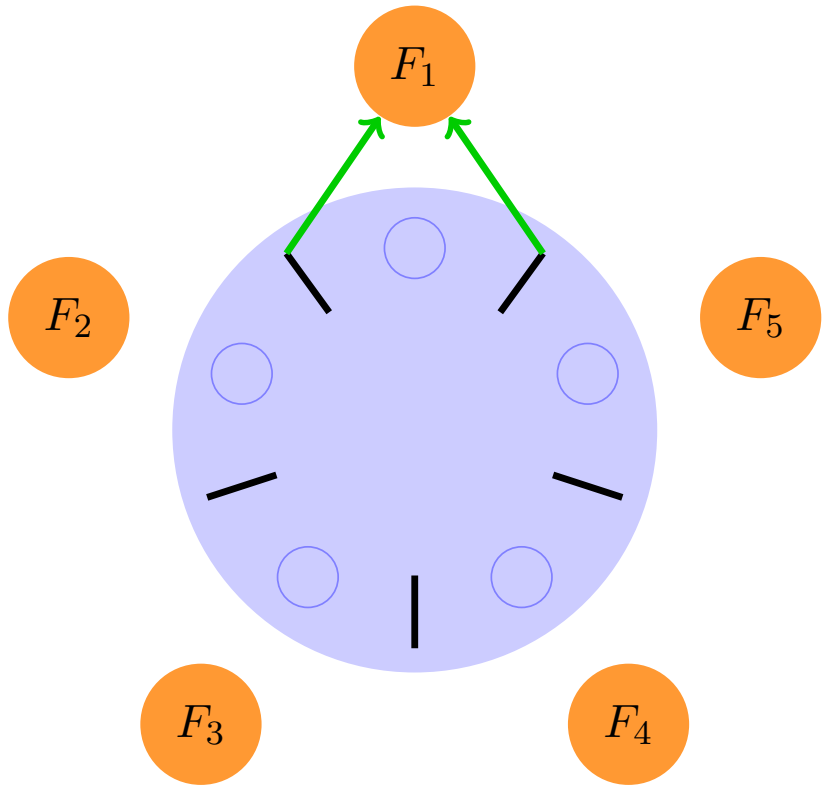
```
void done_eating(int diner){
  mutex_lock(table);

  stick[left] = stick[right] = -1;
  condition_signal(avail[right]);
  condition_signal(aval[left]);

  mutex_unlock(table);
}
```

¿Podemos llegar a un interbloqueo?

- En un principio el mutex `table` nos protege
  - No podemos esperar que todos lleguen al mismo tiempo
  - . . . mutex significa que solo una persona está en la mesa al mismo tiempo
  - . . . entonces, solo uno puede alcanzar al mismo tiempo
  - ¿seguros?
- Tal vez podamos
  - `condition_wait()` es una forma de poder llegar a un interbloqueo
  - ¿Pueden todos llegar a `condition_wait()`?

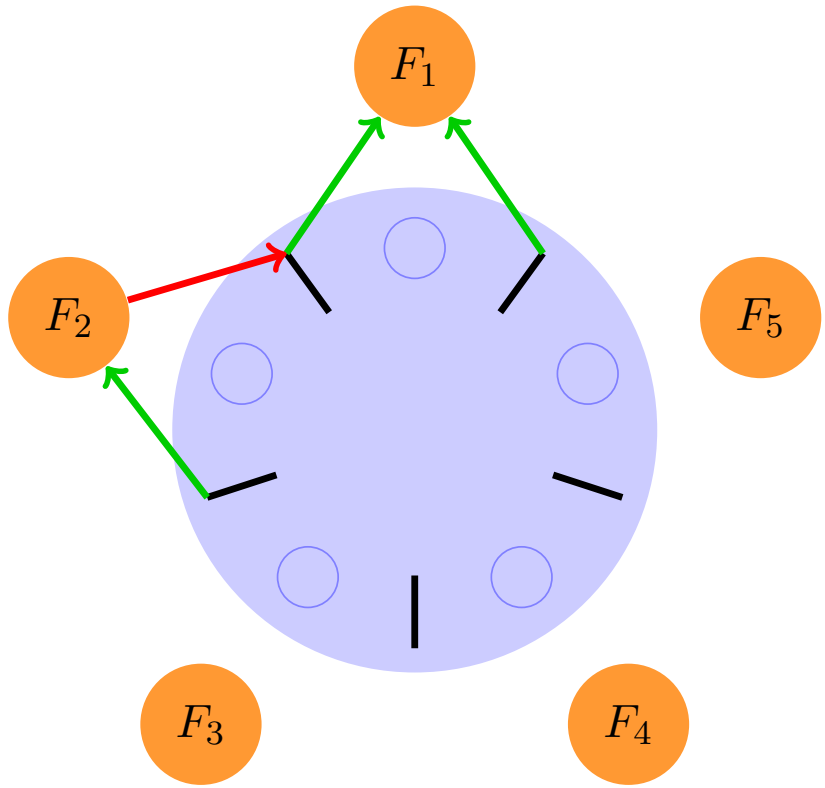
¿Cómo llegar a un interbloqueo?

- El primer filósofo consigue los dos palillos



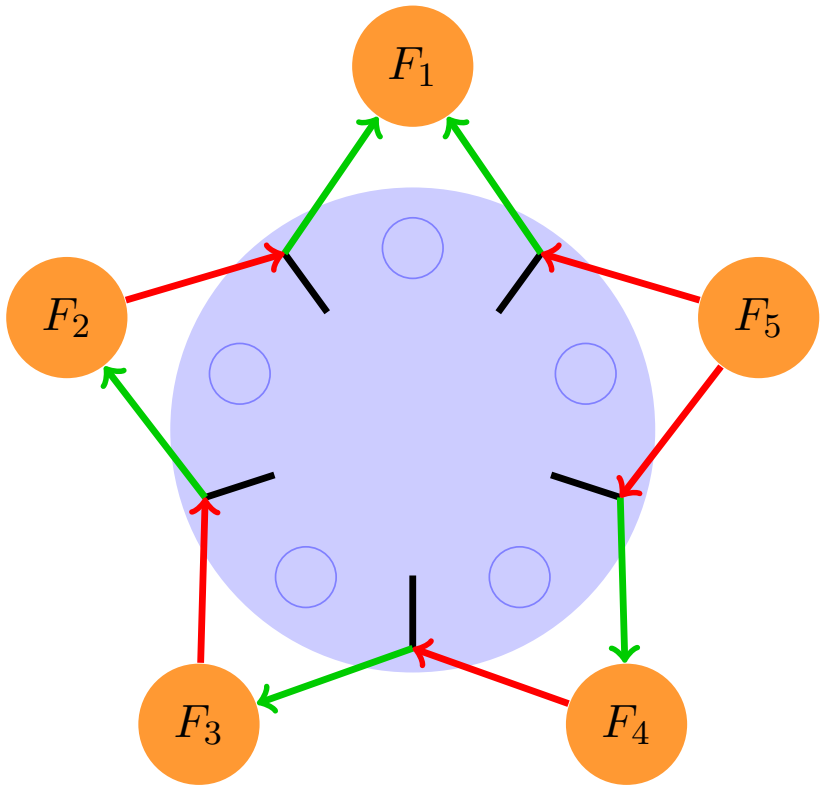
¿Cómo llegar a un interbloqueo?

- El primer filósofo consigue los dos palillos
- El siguiente espera en el primero



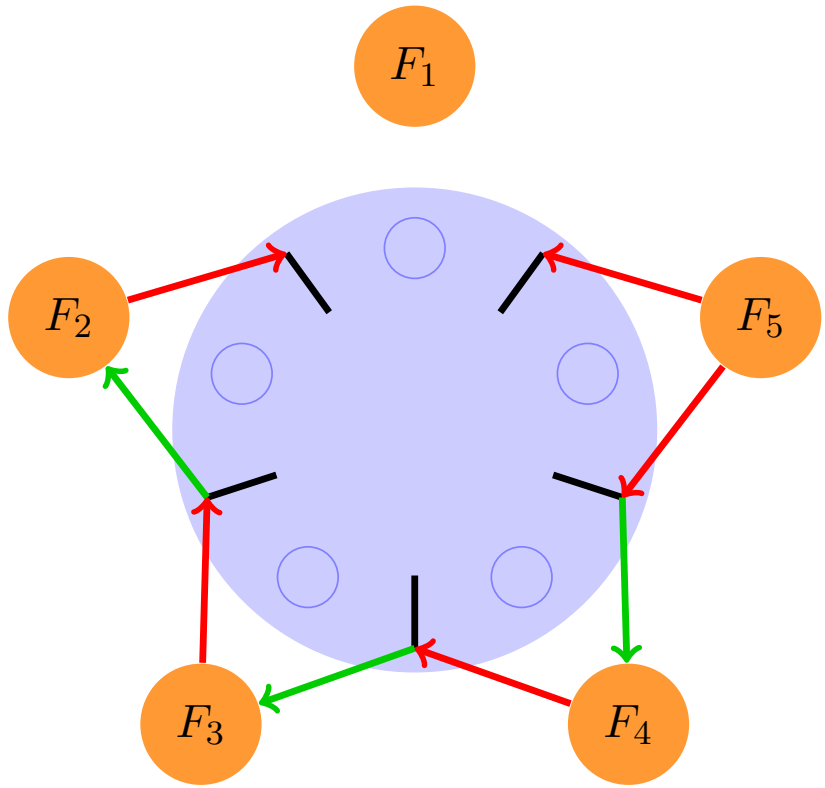
¿Cómo llegar a un interbloqueo?

- El primer filósofo consigue los dos palillos
- El siguiente espera en el primero
- El siguiente espera en el segundo, y sucesivamente



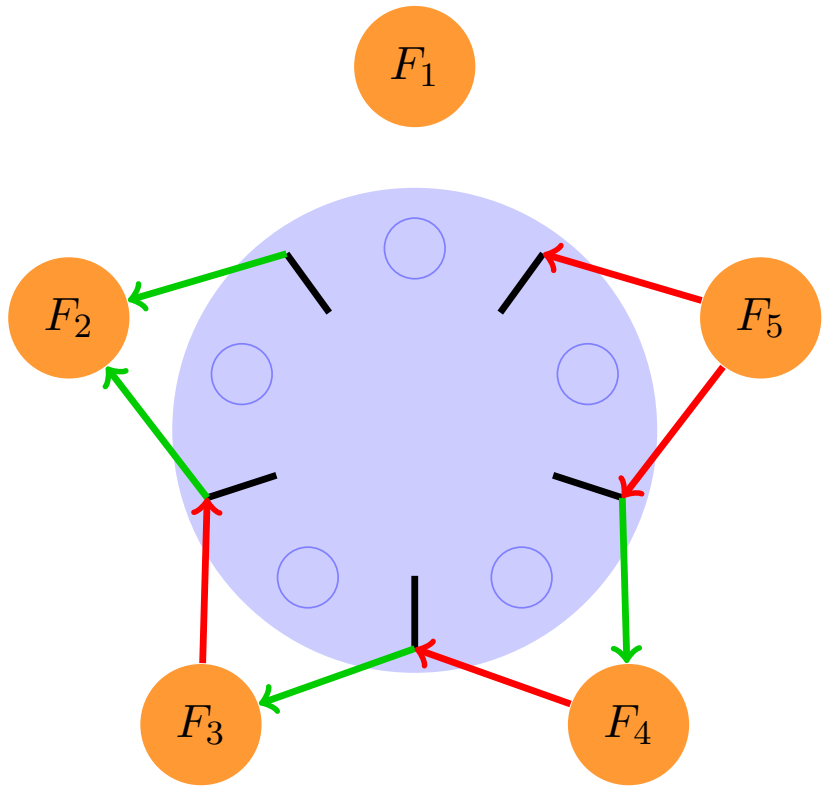
¿Cómo llegar a un interbloqueo?

- El primer filósofo consigue los dos palillos
- El siguiente espera en el primero
- El siguiente espera en el segundo, y sucesivamente
- El primero termina de comer, y entrega los palillos



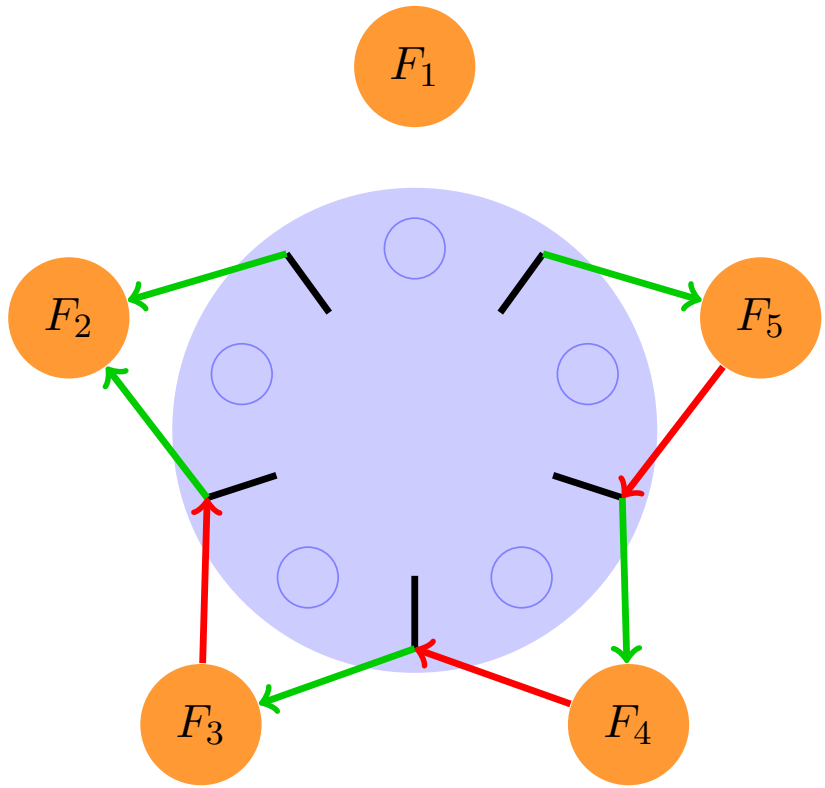
¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo



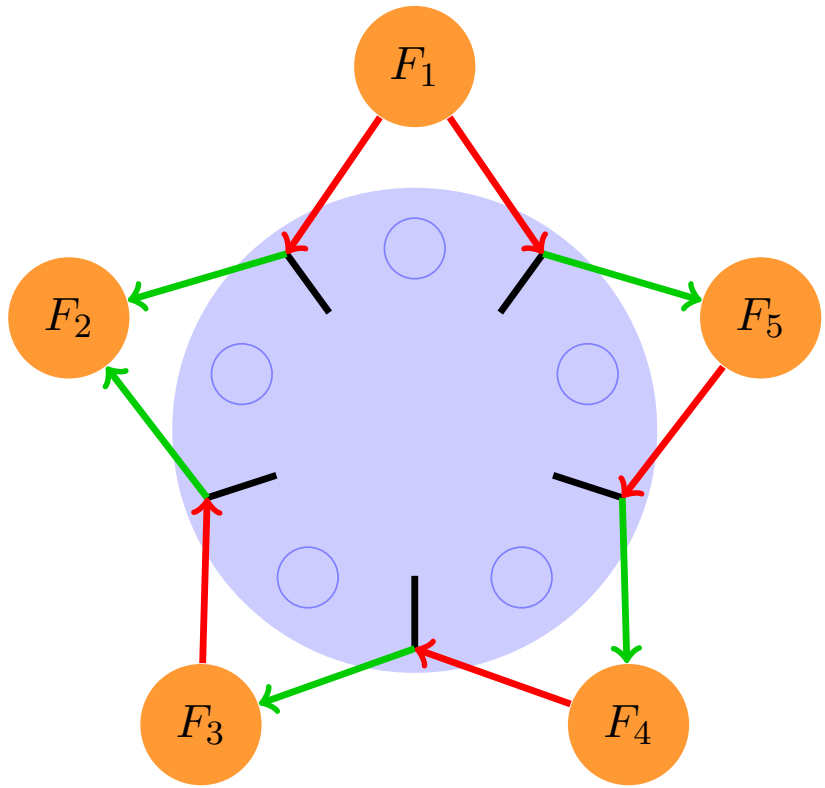
¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo
- El quinto obtiene el palillo, y espera



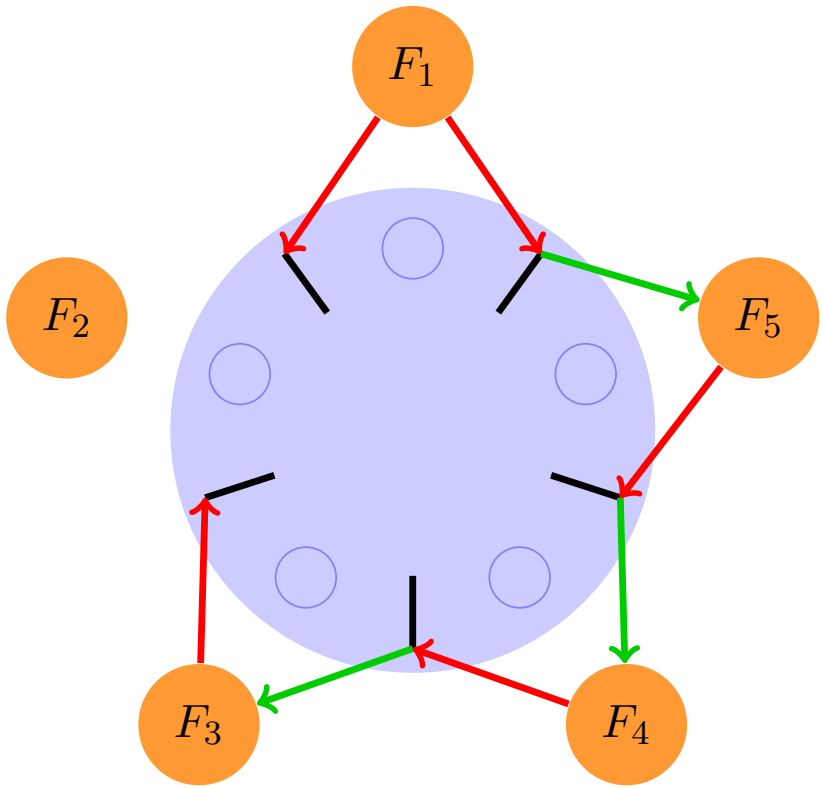
¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo
- El quinto obtiene el palillo, y espera
- El primero quiere comer de nuevo



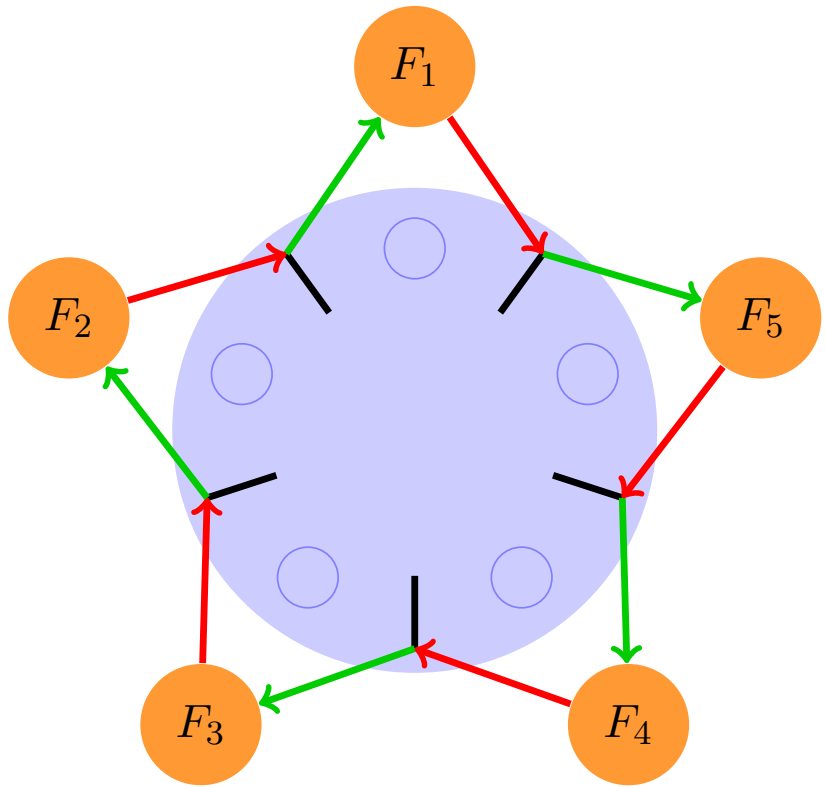
¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo
- El quinto obtiene el palillo, y espera
- El primero quiere comer de nuevo
- El segundo termina de comer



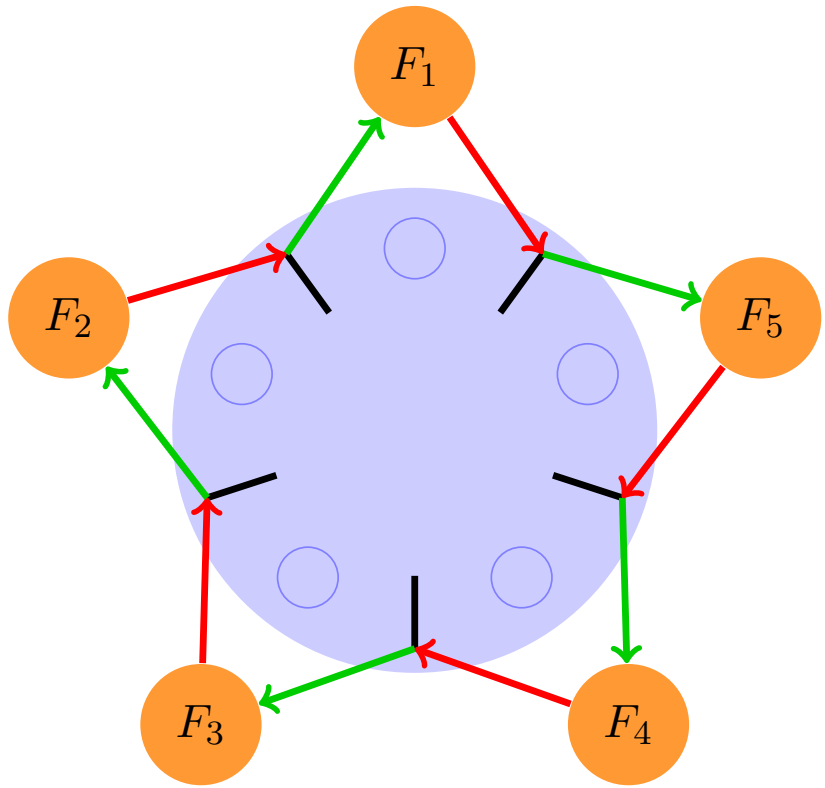
¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo
- El quinto obtiene el palillo, y espera
- El primero quiere comer de nuevo
- El segundo termina de comer
- El primero toma el palillo, y dos quiere volver a comer



¿Cómo llegar a un interbloqueo?

- Si el segundo toma el palillo, podemos seguir comiendo
- El quinto obtiene el palillo, y espera
- El primero quiere comer de nuevo
- El segundo termina de comer
- El primero toma el palillo, y dos quiere volver a comer
- Y . . . interbloqueo!



¿Qué hacemos si llegamos a un interbloqueo?

- Prevenir
- Evitar
- Detectar y recuperarnos
- O reiniciamos cuando este muy tranquilo (no hay acción)

Prevención

- Restringimos el comportamiento o los recursos
  - Encontrar una manera de violar una de las cuatro condiciones que definen un interbloqueo
  - 4 condiciones, 4 posibles maneras de evitarlo

Evitar

- Los procesos pre-declaran patrones de uso
- Examinamos dinámicamente las peticiones
  - Imaginamos lo que los otros procesos puedan pedir
  - Mantenemos el sistema en un estado seguro

Detección y recuperación

- Esperar lo mejor
- Revisar si no hay movimiento o acción en los recursos
- Si está muy tranquilo, y encontramos un ciclo
- Abortemos algunos procesos

Reiniciar el sistema

- ¿En qué sistema podemos hacer esto?
- ¿Es tan simple?

Cuatro posibles soluciones

- Cada interbloqueo necesita las cuatro propiedades
  - Exclusión mutua
  - Obtener y esperar
  - No interrupciones
  - Espera circular
- Prevención de interbloqueos —término técnico
  - Establecemos una ley contra uno (nosotros escogemos)
  - El interbloqueo ocurre si alguien viola la ley

Evitar la exclusión mutua

- Solución: prohibir recursos de un solo usuario
  - Requerimos que todos los recursos se encuentren en modo compartido
- Problema
  - ¿Cómo compartimos los palillos?
  - No todos los recursos funcionan de esa manera

Evitar obtener y esperar

- Obtener los recursos todos o ninguno

```
void start_eating(int diner) {
  mutex_lock(table);
  while(1) {
    if (stick[right] == -1 && stick[left] == -1) {
      stick[right] = stick[left] = diner;
      mutex_unlock(table);
      return;
    }
    condition_wait(released, table);
  }
}
```

Problemas

- Inanición: un conjunto grande de recursos hace que obtener todo simultáneamente sea difícil
  - No podemos garantizar una espera acotada
- Baja utilización
  - Necesidades altas de recursos perjudica al sistema
    - Debemos alojar dos palillos (y el mesero!)
    - Nadie puede utilizar el mesero mientras comemos

Evitar las no-interrupciones

- Robemos los recursos de los procesos que están dormidos

```
void start_eating(int diner) {
  right = diner;
  next_right = (diner+1)%5;
  mutex_lock(table);
  while(1) {
    if (stick[right] == -1)
      stick[right] = diner;
    else if (!is_eating(next_right))
      // el de la derecha no está comiendo
      stick[right] = diner; // tomemoslo!
    // repetimos para la izq.
    if (stick[right] == diner && stick[left] ==
        diner) {
      mutex_unlock(table);
      return;
    } else
    condition_wait(released, table);
  }
}
```

Problemas

- Algunos recursos no se pueden interrumpir de manera limpia
- Ejemplos
  - Un quemador de CDs
  - Una impresora a medio imprimir
  - ¿Otros?

Evitemos la espera circular

- Imponemos un orden total en los recursos
- Debemos obtener los recursos en un orden estricto e incremental
  - Un orden estático puede no funcionar: alojar memoria, luego archivos
  - Dinámico —puede que necesite empezar de nuevo algunas veces
    - Recorremos el grafo de recursos

```
lock(4), visit(4)    // 4 tiene un arco
     a 13
lock(13), visit(13) // 13 tiene un
    arco a 0
lock(0)              // ?
// nop, fallamos: rollback
unlock(0), unlock(13), unlock(4)
```

Orden total para los filósofos comensales

- Orden de recursos: 4, 3, 2, 1, 0 $\equiv$ palillo derecho, luego izquierdo

```
F4 -> lock(4), lock(3)
F3 -> lock(3), lock(2)
F0 -> lock(0), lock(4) // viola el orden
```

- Necesitamos código especial para obtener los seguros

```
if (diner == 0) {
  rigth = (diner + 4) % 5;
  left = diner;
} else {
  right = diner;
  left = (diner + 4) % 5;
}
// ...
```

Problema

- No podemos forzar un orden en todos los casos
  - Algunos caminos van en una dirección
  - Otros en otra
  - No podemos forzarlos a ir en la dirección que nos convenga

Problemas de la prevención de interbloqueo

- Los recursos típicos requieren exclusión mutua
- Alojar todos a la vez puede ser muy doloroso
  - Daña la eficiencia
  - Podemos llegar a inanición
  - Las necesidades de los recursos son impredecibles
- Interrumpir procesos puede ser imposible
  - O puede llevar a la inanición
- Ordenar restricciones puede ser impráctico

Prevención de interbloqueo

- Evitemos una de las características que definen el interbloqueo
- Prohibamos una de ellas a través de una convención
  - Siempre y cuando podamos tolerar el método
- Muy tentador el dejar que los procesos prueben su suerte

Un interbloqueo **no es** . . .

- un bug the la sincronización
  - Interbloqueo se mantiene aun cuando los bugs han sido eliminados
  - Interbloqueo es un problema de diseño de uso de recursos
- lo mismo que la inanición
  - Los procesos en un interbloqueo no obtienen recursos nunca
  - Los procesos en inanición no obtienen recursos oportunamente
  - El interbloqueo es un problema de progreso, mientras que la inanición es un probema de espera acotada
- un baile de “después de usted”
  - Es un *livelock* —cambios continuos en el estado sin poder progresar