# TEMA 6
# Interbloqueos

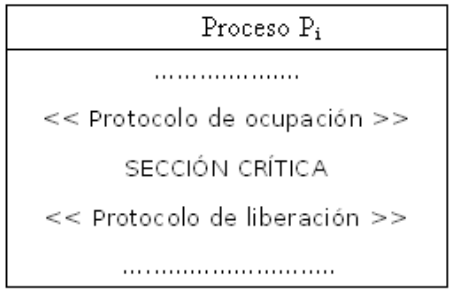








- La forma de realizar la exclusión mutua es haciendo que los distintos procesos accedan de forma secuencial a la **sección crítica**.




- Uno de los problemas que puede provocar dicha forma de actuar es que los procesos se bloqueen de forma permanente en su intento de acceder a la sección crítica. A dicho problema se le conoce como **interbloqueo**.

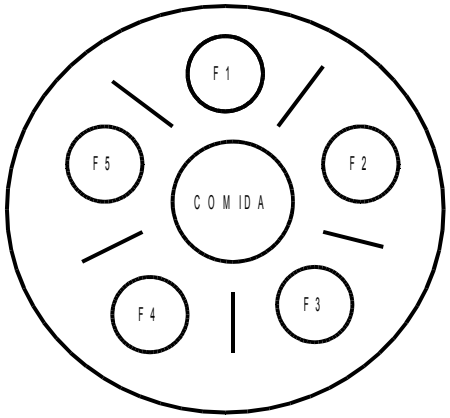




- Un conjunto de procesos alcanza un estado de **interbloqueo** si cada uno de ellos espera que tenga lugar un suceso que sólo puede ser producido por alguno de los procesos de dicho conjunto.
- **Ejemplo**: Dos procesos desean imprimir grandes archivos en una cinta:
  - El proceso A solicita la impresora, que se le concede
  - El proceso B solicita la unidad de cinta, que se le concede.
  - El proceso A solicita la unidad de cinta, pero se deniega la solicitud hasta que B la libera
  - El proceso B solicita la impresora y se produce el interbloqueo
- Otros nombres son: DeadLock, abrazo mortal, bloqueo mutuo

# 1. Introducción

- Si un sistema mantiene indefinidamente a un proceso a la espera de asignarle un recurso mientras otro recibe la atención del sistema decimos que ese aquel proceso sufre **Postergación indefinida (inanición).**
- Ejemplo: cena de los filósofos






# 2. Recursos

- Un recurso es tanto un dispositivo hardware como una cierta cantidad de información.
- Clasificación
  - Apropiativos/no apropiativos
  - Reutilizables/consumibles
  - Compartidos/Exclusivos
  - Un ejemplar/múltiples ejemplares
- **Las técnicas de interbloqueo tratan con recursos reutilizables no apropiativos o recursos reutilizables de acceso exclusivo**.



# 3. Modelo del sistema

- Para la explicación de los distintos algoritmos suponemos que nuestro sistema informático:
  - Tiene un número finito de recursos.
  - Tiene un número finito de procesos que compiten.
  - Los recursos se agrupan (están formados por unidades)
  - Para que un proceso use un recurso, necesita
    - Pedir el recurso
    - Usar el recurso
    - Liberar el recurso
  - No se pueden pedir más recursos de los disponibles
  - Si un proceso no puede realizar ninguna operación estará bloqueado



# 3. Modelo del sistema

- Si tenemos recursos consumibles:
  - El número de unidades disponibles de un recurso varía según sean asignadas y liberadas por los procesos
  - El productor incrementa el número de unidades disponibles y el consumidor disminuye el número de unidades realizando una petición y la posterior liberación.
  - El sistema no limita los recursos consumibles, la producción de recursos se ve limitada por la capacidad de los procesos o por las limitaciones físicas de capacidad de la memoria o del *buffer* destinado a los productos creados.

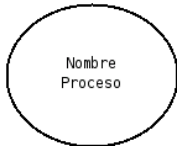
- Se da una situación de interbloqueo si se cumplen todas y cada una de las siguientes condiciones.
  - **Exclusión mutua**
  - **Posesión y espera**
  - **No apropiación**
  - **Espera circular**
- Estas condiciones son **condiciones necesarias pero no suficientes**, o sea, el cumplimiento de las condiciones no asegura la presencia del interbloqueo, pero para que exista un interbloqueo deben cumplirse las cuatro.



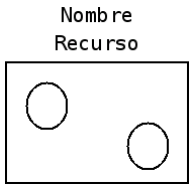


## 4.1 Grafo de asignación de recursos

- Los interbloqueos se representan mediante grafos de asignación de recursos *G(N,A)*
- Los procesos se representan mediante círculos




- Los recursos mediante rectángulos

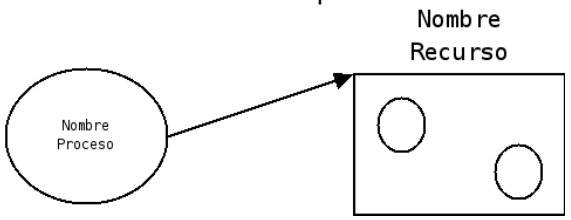




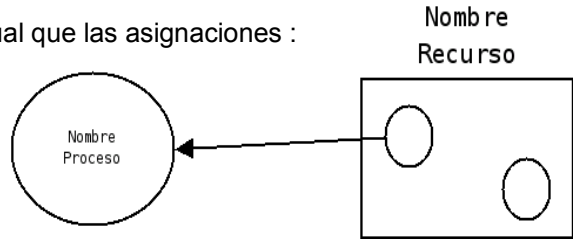



## 4.1 Grafo de asignación de recursos

- Las peticiones de recursos se representan mediante flechas:




- Al igual que las asignaciones :



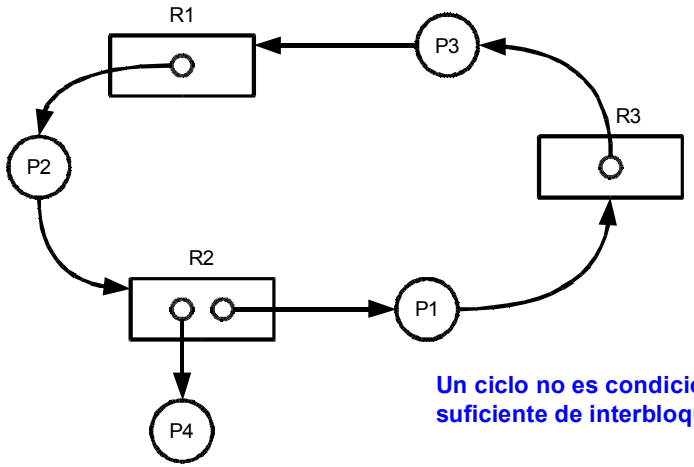





## 4.1 Grafo de asignación de recursos

- Restricciones de los grafos:
  - El número de arcos que salen de un recurso debe ser menor o igual que el número de ejemplares del recurso.
  - Por cada pareja de proceso $_i$ y recurso $_j$, el número de arcos de asignación que van de $R_j$ a $P_i$ más el número de aristas de solicitud que conecta $P_i$ a $R_j$ será menor o igual que el número de ejemplares del recurso.
  - Un ciclo en el grafo **puede indicar** la existencia de un interbloqueo relacionado con los procesos y recursos en el ciclo.

## 4.1 Grafo de asignación de recursos

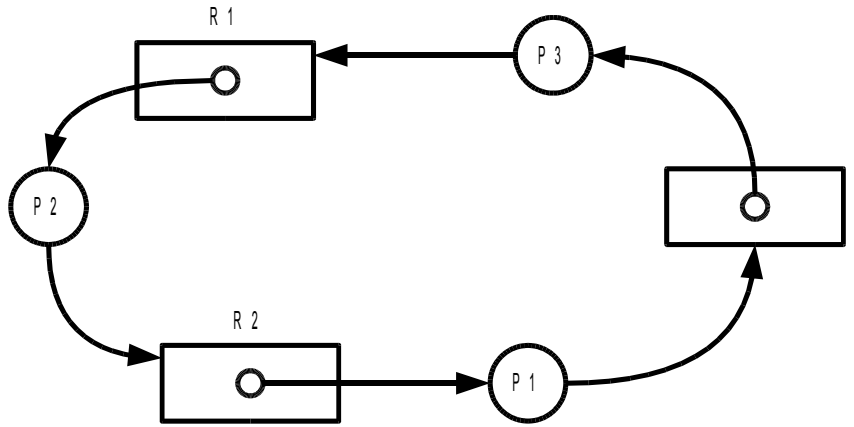



**Un ciclo no es condición suficiente de interbloqueo**





## 4.1 Grafo de asignación de recursos








## 4.1 Grafo de asignación de recursos

- Internamente los grafos se representan mediante:
  - **Matrices** de dimensión n x m
    - Matrices de **asignación** (A)
    - Matrices de petición o **solicitudes pendientes** (S)
    - Vector de recursos Existentes (E)
  - **Listas Encadenadas**
    - **Dos listas por proceso**, una de recursos asignados y otra solicitados. (Ligada al PCB)
    - **Dos listas por recurso**, una para los procesos que están usando el recurso y otra para las solicitudes. (Ligada al RCB)





## 4.1 Grafo de asignación de recursos

- **Ejemplo**:
  - **Recursos existentes**
    - 2 unidades del R1,
    - 3 unidades del R2,
    - 1 unidad del R3.
  - **Secuencia de peticiones**
    - P1 tiene una unidad del R1 y otra del R2.
    - P2 tiene una unidad del R3 y solicita una del R1.
    - P3 tiene 2 unidades del R2 y una del R1, ha solicitado una del R3.
    - P4 solicita una unidad del R1 y otra del R2.

## 4.1 Grafo de asignación de recursos

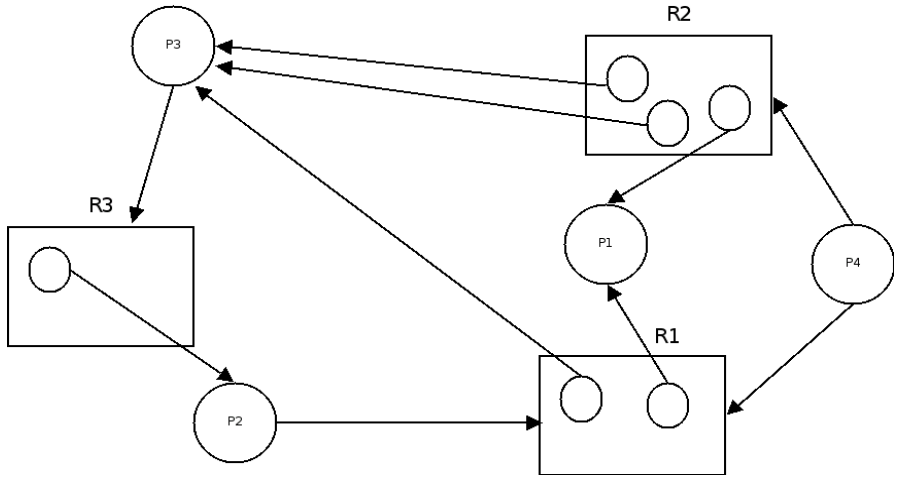
- Matrices de asignación y pendientes:

| Asignados | | | | | Pendientes | | | |
|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | | | R1 | R2 | R3 |
| P1 | 1 | 1 | 0 | | P1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 1 | | P2 | 1 | 0 | 0 |
| P3 | 1 | 2 | 0 | | P3 | 0 | 0 | 1 |
| P4 | 0 | 0 | 0 | | P4 | 1 | 1 | 0 |





## 4.1 Grafo de asignación de recursos








## 4.1 Grafo de asignación de recursos

- **Ejemplo 2:**
  - **Recursos existentes**
    - 2 unidades del R1
    - 3 unidades del R2,
    - 1 unidad del R3.
  - **Secuencia de peticiones**
    - P1 tiene una unidad del R1 y otra del R2. **Ha solicitado otra de R2**
    - P2 tiene una unidad del R3 y solicita una del R1.
    - P3 tiene 2 unidades del R2 y una del R1, ha solicitado una del R3.
    - P4 solicita una unidad del R1 y otra del R2.





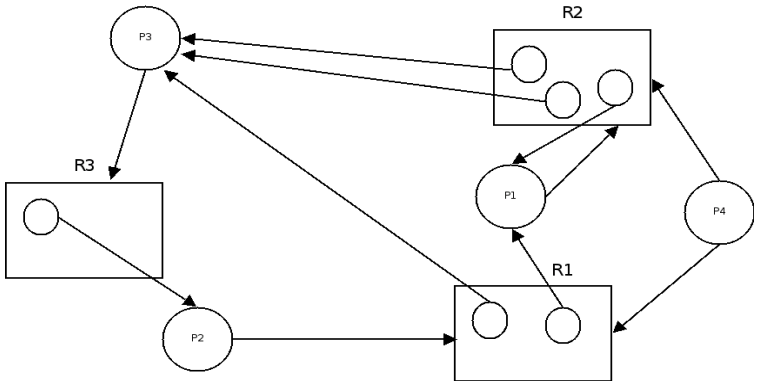
## 4.1 Grafo de asignación de recursos

- Matrices de asignación y pendientes:

| Asignados | | | | | Pendientes | | | |
|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | | | R1 | R2 | R3 |
| P1 | 1 | 1 | 0 | | P1 | 0 | 1 | 0 |
| P2 | 0 | 0 | 1 | | P2 | 1 | 0 | 0 |
| P3 | 1 | 2 | 0 | | P3 | 0 | 0 | 1 |
| P4 | 0 | 0 | 0 | | P4 | 1 | 1 | 0 |

# 4. Caracterización del interbloqueo

## 4.1 Grafo de asignación de recursos








# 5. Tratamientos del interbloqueo

- Cuatro grandes áreas para el tratamiento del interbloqueo:
  - **Ignorar el problema**, donde se engloba el algoritmo de Omisión o algoritmo del avestruz.
  - **Afrontar el problema**, podemos hacerlo tomando una actitud:
    - **Pesimista**, garantizamos que nunca se entra en un estado de interbloqueo. Hablaremos de algoritmos de **Prevención** y **Evitación** (predicción).
    - **Optimista**, permitimos que el sistema entre en interbloqueo y luego se recupere. Dentro de este grupo aparecen los algoritmos de **Detección y corrección** (recuperación).
- Además existen algoritmos que combinan características de varias de estas áreas.





# 5. Tratamientos del interbloqueo

## 5.1 Omitir interbloqueos

- Conocido como el algoritmo del avestruz
- Antes de buscar solución a los interbloqueos valoramos los siguientes factores:
  - Número de veces que se produce.
  - Gravedad del mismo.
  - Coste de las consecuencias.
  - Coste de hacer algo.
- Es muy difícil que se de un interbloqueo en el sistema → no proporcionar ningún mecanismo para tratar los interbloqueos.





# 5. Tratamientos del interbloqueo

## 5.2 Prevenir los interbloqueos

- Estas estrategias consisten en diseñar un sistema de manera que esté excluida la posibilidad de interbloqueo.
- Se basa en eliminar alguna de las cuatro condiciones que se deben cumplir para que exista un interbloqueo.

**1. Eliminar la exclusión mutua**

- Si un proceso no se puede asignar exclusivamente a un solo proceso no se producirá jamás un interbloqueo.
- La solución será no dar uso exclusivo de los recursos a los procesos. Esto la mayoría de las veces es imposible.

## 5.2 Prevenir los interbloqueos

### 2. Eliminar la Posesión y espera

- La posesión y espera se produce porque los procesos van adquiriendo los recursos de forma incremental a través del tiempo.
- Podemos evitar la posesión y espera si el proceso debe indicar los recursos que necesite antes de comenzar a ejecutarse. Con esta información el SO se encargará de darle **todos o ninguno** de los recursos.
- Hay dos posibles implementaciones:
  1. El proceso solicita todos los recursos al principio.
  2. El proceso solicita los recursos de forma incremental, pero libera todos los recursos retenidos si se encuentra alguna negativa.
- Nunca se produce interbloqueo, puesto que los procesos que esperan no tienen recursos en su poder.





## 5.2 Prevenir los interbloqueos

### 2. Eliminar la Posesión y espera

- Esta solución presenta los siguientes **problemas**:
  1. Es **difícil conocer a priori las necesidades del proceso**.
  2. Los recursos que son asignados son inalcanzables para el resto, lo que conlleva a una **baja utilización de los recursos**.
  3. Si los recursos se toman incrementalmente, al soltar y recuperar un recurso puede llevar a **estados inconsistentes**.
- Una **mejora** que se puede conseguir es **dividir el proceso en pasos** que se ejecutan con cierta independencia de manera que las asignaciones se hacen al comienzo de cada paso. La complejidad de ejecución que representa esta alternativa hace inviable su aplicación.
- Esta técnica de prevenir los interbloqueos puede provocar **postergación indefinida** (hambre).





## 5.2 Prevenir los interbloqueos

### 3. Eliminar la No Apropiación

- Si todos los recursos fueran **expropiables** podemos proceder así:
  - *Cuando un proceso pide un recurso, se comprueba si está disponible. Si está disponible se le asigna y continua su ejecución. Si no está disponible se comprueba si está asignado a otro proceso que espera por otros recursos. De ser así se lo quita al que espera y continua. Si por el contrario, el recurso que esperamos está asignado a un proceso que se está ejecutando, el proceso se suspende, con lo cual le podrán quitar todos los recursos que tenga asignados.*





## 5.2 Prevenir los interbloqueos

### 3. Eliminar la No Apropiación

- Esta técnica sólo se podría aplicar a procesos cuyos estados puedan ser salvados y recuperados fácilmente.
- Esta solución presenta los siguientes **problemas**:
  - El proceso puede **perder gran parte del trabajo** realizado por tanto sólo se puede usar con procesos que puedan salvar su estado.
  - Puede aparecer **inanición**.

## 5.2 Prevenir los interbloqueos

### 4. Eliminar la Espera Circular

- Para prevenir la espera circular podemos optar por alguno de los dos métodos siguientes:
  - Un proceso **sólo puede tener un recurso en cada instante de tiempo**. Esto no es realista, por ejemplo, ¿qué pasaría con operaciones de lectura e impresión?.
  - Imponer una **ordenación lineal de los recursos**: Para ello asignamos un número a cada recurso, y seguimos la siguiente regla:





## 5.2 Prevenir los interbloqueos

### 4. Eliminar la Espera Circular

- Imponer una **ordenación lineal de los recursos**: Para ello asignamos un número a cada recurso, y seguimos la siguiente regla:
  - "Todas las peticiones de recursos deben realizarse en orden numérico creciente". A esto se le llama **asignación jerárquica de recursos.**
  - De esta forma cuando un proceso pide un recurso `j`, luego sólo puede pedir un recurso `k` tal que `k > j` mientras tenga el recurso `j` en posesión.
  - Si un proceso necesita un recurso de nivel inferior, debe liberar primero todos aquellos de nivel superior que tenga.





## 5.2 Prevenir los interbloqueos

### 4. Eliminar la Espera Circular

- Ambas alternativas proporcionan las siguientes **ventajas**:
  - No se producen ciclos.
  - El respeto de los procesos a la ordenación prescrita puede ser comprobado en tiempo de compilación.
- Aún así la aplicación de las mismas es una quimera ya que ninguna de las dos se salvan de los siguientes **problemas**:
  - Si hay que introducir un nuevo recurso hay que renumerar.
  - Puede ser imposible encontrar una ordenación lineal de los recursos que satisfaga a todos los proceso

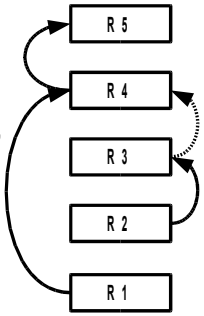




## 5.2 Prevenir los interbloqueos

### 4. Eliminar la Espera Circular

- **Ejemplo**: En la figura siguiente se muestran una serie de recursos con un orden lineal impuesto, un menor número indica un orden menor.
- El proceso A que pide por este orden los recursos R1, R4 Y R5 (lo puede hacer ya que el orden es creciente).
- El proceso B logra la posesión de los recursos R2 y R3. Luego solicita el recurso R4, y dado que lo posee A, debe esperar a que lo libere.
- Si ahora el proceso A solicitase el recurso R2, se produciría un interbloqueo, pero dado que para solicitarlo debe liberar R5, y R4, entonces el proceso B lograría acceso a R4 y no se produciría el interbloqueo.



- **Ejercicio**: Dados los siguientes procesos:

| PROCESO P0 | PROCESO P1 | PROCESO P2 |
|---|---|---|
| ..... | ..... | ..... |
| SOLICITUD DE DISCO | SOLICITUD DE IMPRESORA | SOLICITUD DE IMPRESORA |
| ..... | ..... | ..... |
| SOLICITUD DE IMPRESORA | SOLICITUD DE TERMINAL | SOLICITUD DE DISCO |
| ..... | ..... | ..... |
| SOLICITUD DE TERMINAL | SOLICITUD DE DISCO | SOLICITUD DE TERMINAL |

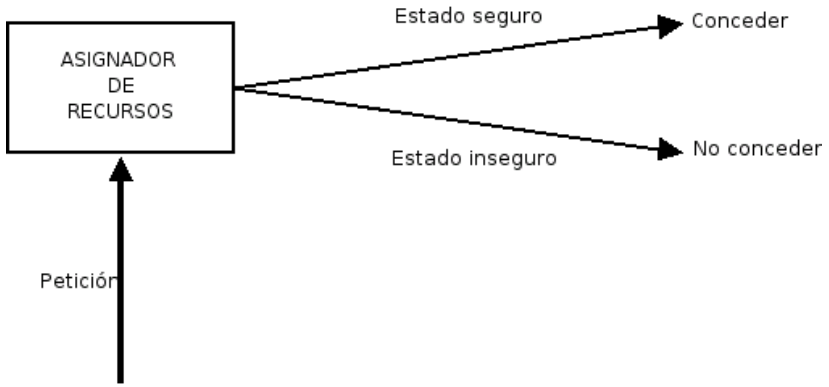
- Y sabiendo que una vez solicitado el disco, se liberará automáticamente tras usarlo (antes de la próxima petición de recursos), mientras que el resto de recursos, una vez concedidos, tan solo se liberarán al finalizar el proceso.
- Imponer una ordenación lineal de los recursos para prevenir posibles situaciones de interbloqueo en el sistema, de tal forma que dichos procesos se ejecuten sin problemas. ¿Qué otras técnicas conoces para prevenir los interbloqueos (enuméralas)?.





## 5.3 Evitar interbloqueo

- Conceder únicamente las peticiones de recursos disponibles que no conduzcan a estados propensos al interbloqueo








## 5.3 Evitar interbloqueo

- Intenta detectar “puntos de no retorno” a partir de los cuales el interbloqueo es inevitable.
- **Ejemplo**:

| Proceso P1 | Proceso P2 |
|---|---|
| Solicita(C) | Solicita(I) |
| Solicita(I) | Solicita(C) |
| Uso de recursos | Uso de recursos |
| Libera (I) | Libera (C) |
| Libera (C) | Libera (I) |

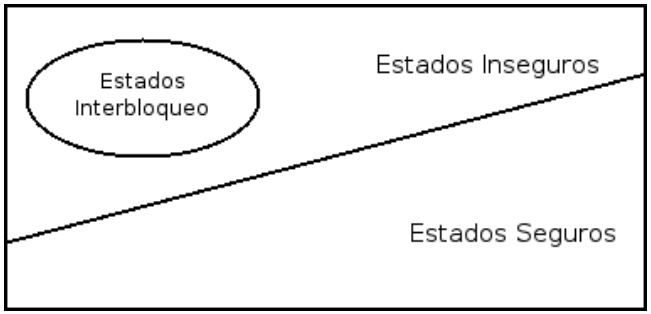




## 5.3 Evitar interbloqueo

- El sistema debe poder decidir si la concesión de un recurso a un proceso es **seguro** o no.
- Existen algoritmos que nos permita detectar si nos estamos acercando a este **“punto de no retorno”** pero todos parten de tener disponible cierta **información previa sobre las necesidades máximas de cada proceso**. A partir de esta información estos algoritmos determinan si el estado del sistema es seguro.
- **Si un sistema está interbloqueado está también en un estado inseguro.** De hecho, habrá pasado por un estado inseguro antes de alcanzar el interbloqueo.
- En cambio **si un sistema está en estado inseguro no tiene porque estar interbloqueado.**

## 5.3 Evitar interbloqueo

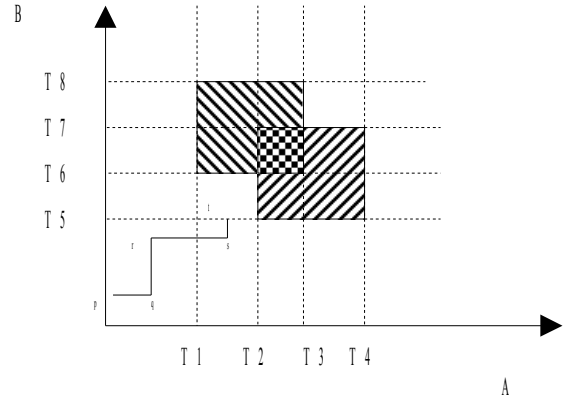



- **Técnicas de detección de estados seguros**
  - Diagrama de ejecución
  - Algoritmo del banquero





## 5.3.1 Diagrama de ejecución








## 5.3.2 Algoritmo del banquero

- Los diagramas de ejecución son muy difíciles de extender a un caso más general.
- Para el caso en los que existan más de dos procesos y recursos se usa el Algoritmo del Banquero (Dijkstra, 1965).
  - *Este algoritmo parte de una serie de procesos que pueden pedir una serie de recursos. Dichos procesos se comprometen a devolver los recursos que se les concedan. El banquero (el SO) no tiene la suma total de todos los recursos que le pueden pedir, pero sabe que no todos los procesos le solicitarán todos los recursos a la vez. Así pues el banquero sólo dará los recursos a aquellos procesos que puedan acabar su ejecución y devolver los recursos.*





## 5.3.2 Algoritmo del banquero

- **Ejemplo**: Veamos como funcionaría con un ejemplo:
  - Tenemos 5 procesos
  - Tenemos 3 recursos: X con 6 ejemplares, Y con 3 ejemplares y Z con 4 ejemplares.
  - En un momento dado se produce la siguiente situación:

| | Recursos Asignados | | | Recursos totales necesitados | | |
|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z |
| **A** | 3 | 0 | 1 | 4 | 1 | 1 |
| **B** | 0 | 1 | 0 | 0 | 2 | 1 |
| **C** | 1 | 1 | 1 | 4 | 2 | 1 |
| **D** | 1 | 1 | 0 | 1 | 1 | 1 |
| **E** | 0 | 0 | 0 | 2 | 1 | 1 |

Recursos Totales del sistema: **VECTOR E = (6,3,4)**
Recursos Totales asignados: **VECTOR A = (5,3,2)**
Recursos Totales sin asignar: **VECTOR L = (1,0,2)**

## 5.3.2 Algoritmo del banquero

- El algoritmo del banquero consiste en:
  1. Buscar una fila cuyas necesidades pendientes sean menores que el vector L. Si no encontramos ninguna entonces estamos en un estado **INSEGURO**.
  2. Si lo hay se le conceden todos los recursos que necesita, que serán liberados al finalizar su ejecución, tras lo cual se vuelve a calcular el vector L
  3. Volver al paso 1. Si finalizan todos los procesos estaremos en un estado **SEGURO**.
- **¿Cómo actuaremos?**
  - A cada petición miraremos si el estado es **SEGURO** o no según el algoritmo anterior. Si el estado es seguro se le concede la petición, sino se deniega la petición, para evitar llegar a un estado **INSEGURO**.





## 5.3.2 Algoritmo del banquero

- **Ejemplo**: E=(6,3,4), A=(5,3,2), L=(1,0,2)

| | Recursos Asignados | | | Recursos totales necesitados | | | Necesidades Pendientes | | |
|---|---|---|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z | X | Y | Z |
| A | 3 | 0 | 1 | 4 | 1 | 1 | 1 | 1 | 0 |
| B | 0 | 1 | 0 | 0 | 2 | 1 | 0 | 1 | 1 |
| C | 1 | 1 | 1 | 4 | 2 | 1 | 3 | 1 | 0 |
| D | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 2 | 1 | 1 | 2 | 1 | 1 |

*D* puede finalizar, por tanto *L1=L+Dasig*=(2,1,2).
Puede acabar A, B o E. Elegimos A; *L2=L1+Aasig*=(5,1,3).
Puede acabar B, C o E. Elegimos B; *L3=L2+Basig*=(5,2,3).
Puede acabar C o E. Elegimos C; *L4=L3+Casig*=(6,3,4).
Puede acabar E. *L5=L4+Easig*=(6,3,4)
Todos los procesos han acabado, el estado es **seguro**.





## 5.3.2 Algoritmo del banquero

- **Inconvenientes**
  - El número de recursos y procesos debe ser fijo.
  - Los procesos tienen que informar con antelación de los recursos que van a necesitar (Necesidades máximas = S + A).





## 5.4. Detección y recuperación

- Deja que aparezcan y después se lleva a cabo la recuperación.
- Dos **fases** claramente distinguidas:
  - **Fase de detección**
  - **Fase de recuperación**
- Entran en la primera fase en el momento en que se liberen recursos o lleguen solicitudes.
- En estos momentos se estudia la formación de ciclos en los grafos de asignación.
- Si detecta ciclos se entra en la fase de recuperación y se intentan eliminar.
- Las estrategias de detección y recuperación son las más empleadas en los grandes sistemas.

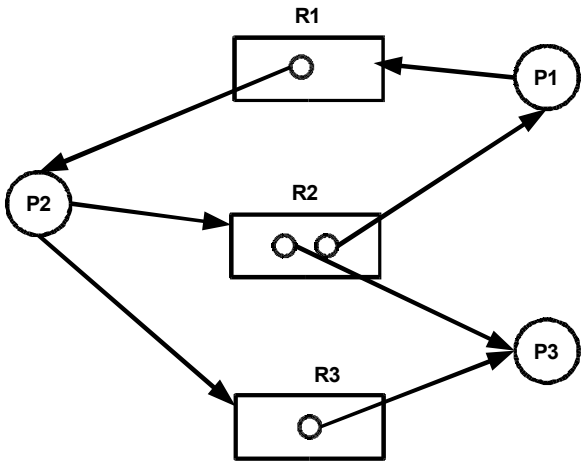
## 5.4.1. Detección: reducción del grafo

- Dado un grafo de asignación de recursos, este **se puede reducir por un proceso Pi si los recursos disponibles satisfacen sus necesidades**.
- La reducción consiste en eliminar las aristas de solicitud que salen del nodo Pi y las aristas que llegan a dicho nodo.
- Cada reducción da lugar a un nuevo grafo donde podrá haber nuevos procesos susceptibles de ser reducidos.
- Si el grafo no se puede seguir reduciendo, llegaremos a un estado de interbloqueo y los procesos contenidos en este grafo irreducible son los que están interbloqueados.





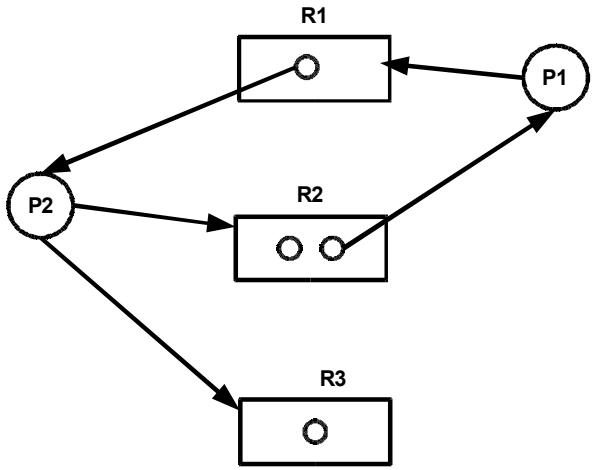
## 5.4.1. Detección: reducción del grafo







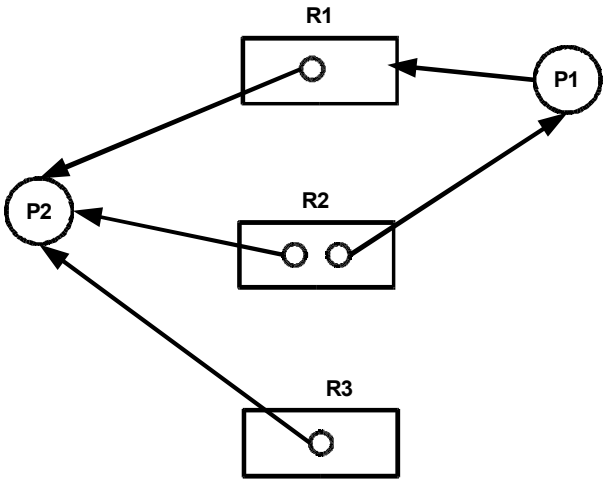

## 5.4.1. Detección: reducción del grafo



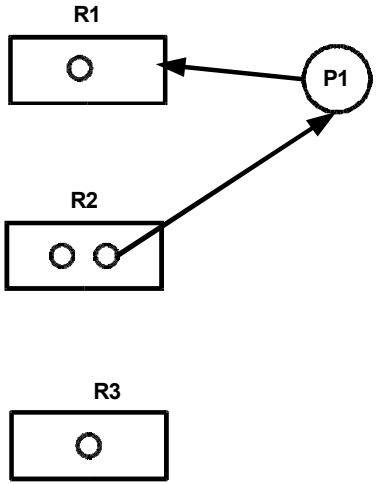





## 5.4.1. Detección: reducción del grafo



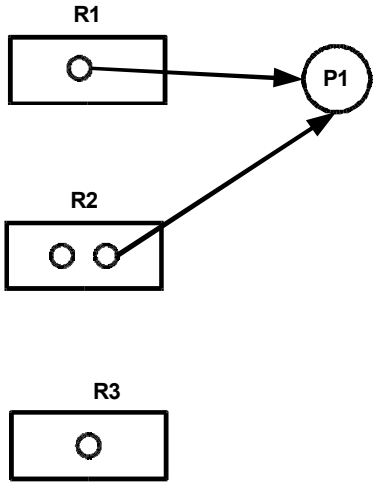
## 5.4.1. Detección: reducción del grafo








## 5.4.1. Detección: reducción del grafo








## 5.4.1. Detección: reducción del grafo

- Activación del algoritmo
  - **Supervisión continua**:
    - Ejecución continuamente
    - Permiten detectar un interbloqueo justo cuando se produce y
    - Provoca una gran carga adicional en el sistema.
    - Mejora:
      - Sólo se ejecuta cuando no se satisface una petición.
  - **Supervisión periódica**:
    - Elegimos un período para realizar la detección.
      - Cada cierto intervalo de tiempo
      - Cuando la CPU presente un % de uso alto





## 5.4.2. Recuperación

- Al detectarse un interbloqueo se debe aplicar una acción para eliminarlo. Es acción puede ser:
  - **Manual**: informar al operador del sistema y este se ocupa manualmente.
  - **Automática**: el sistema se recupera automáticamente.
- ¿Cómo llevar a cabo la recuperación?
  - **Terminación de procesos**
    - Todos los procesos/un proceso a la vez
    - Inconvenientes (Inconsistencias, selección del proceso)
  - **Expropiación de recursos**
    - Selección de víctima
    - Retroceso (¿qué hago con el proceso?)
    - Postergación indefinida (evitar expropiar siempre el mismo proceso)

## 5.5. Modo combinado

- Se buscan soluciones combinas que traten el interbloqueo
- Opción
  - El SO debe separar los recursos por tipos creando **conjuntos disjuntos.**
  - A cada uno de ellos se le asigna una **técnica diferente** según las características y tipos de los recursos.
  - Tenemos solucionados los interbloqueos dentro de los grupos.
  - Para solucionar los **interbloqueos entre grupos imponemos entre ellos una ordenación lineal de los recursos** (evitamos la espera circular).





## 5.5. Modo combinado

- Ejemplo:
  - Grupo 1: Zona de intercambio (swapping). Usamos prevención de eliminación de retención y espera.
  - Grupo 2: Información. Usamos prevención con ordenación lineal.
  - Grupo 3. Memoria principal. Usamos la prevención de eliminación de retención y espera.
  - Grupo 4. Recursos internos. Usamos la evitación.
  - Grupo 5. Dispositivos de E/S. Usamos detección y recuperación.
- Entre grupos de recursos distintos imponemos la ordenación lineal de los recursos según los números de grupos y la asignación jerárquica de recursos. Ordenación lineal entre grupos