Primera Escuela de Sistemas Distribuidos

6 al 10 de Mayo, Xalapa, Veracruz México

La Algoritmica Distribuida

Roberto Gómez Cárdenas

ITESM-CEM

rogomez@campus.cem.itesm.mx

http://webdia.cem.itesm.mx/dia/ac/rogomez

La algoritmica

- Diseño algoritmos constituye una parte fundamental de las ciencias computacionales.
- Algoritmos estudiados en un contexto de programación secuencial
  - control de flujo del algoritmo es único y en cada paso del cálculo se realiza una acción única.
  - se estudia el manejo de estructuras de datos (arboles, listas, matrices, grafos, etc) y el uso de estas estructuras para producir un cálculo final.

Algoritmica paralela

- Algoritmica secuencial largamente estudiada.
- No es el caso para la algoritmica en un contexto paralelo
  - varios flujos de control pueden coexistir simultaneamente
- En un principio paralelismo y su algoritmica es estudiado dentro del área de sistemas operativos
  - el diseño del sistema esta dividido en actividades elementales de cálculo (procesos), de los que hay que administrar las interacciones

Los algoritmos y las redes

- Fenomeno redes, junto con el progreso de la metodología de programación introdujeron el concepto de comunicación como base del diseño de algortimos paralelos
- Nace una nueva algoritmica basada en el intercambio de mensajes entre actividades.
- Estos algoritmos paralelos son denominados protocolos.

Los protocolos

- Realizan servicios de transferencia de información o de control de actividades, dentro un conjunto de procesos que:
  - constituyen una aplicación
  - se ejecutan sobre distintos sitios
  - son enlazados por vías de comunicación
- Dentro del contexto de sistemas distribuidos estos algoritmos se conocen como *algoritmos distribuidos.*

Computo local vs distribuido

- El termino computo local se refiere a los programas que están confinados a un solo espacio de direcciones.
- El termino de computo distribuido se refiere a programas que hacen llamadas a otros espacios de direcciones, posiblemente en otra máquina.

¿Qué es un sistema distribuido?

- Conjunto de entidades que se comunican entre ellas a través de mensajes, los cuales son enviados sobre vías de comunicación.
- Otras definiciones

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusuable"
Leslie Lamport

"A distributed system is one that stops from getting any work done when a machine you've never never heard of crashes"
Distributed Systems (Ed. Sape Mullender)
edition 1, ACM Press 1989

Tipos sistemas distribuidos

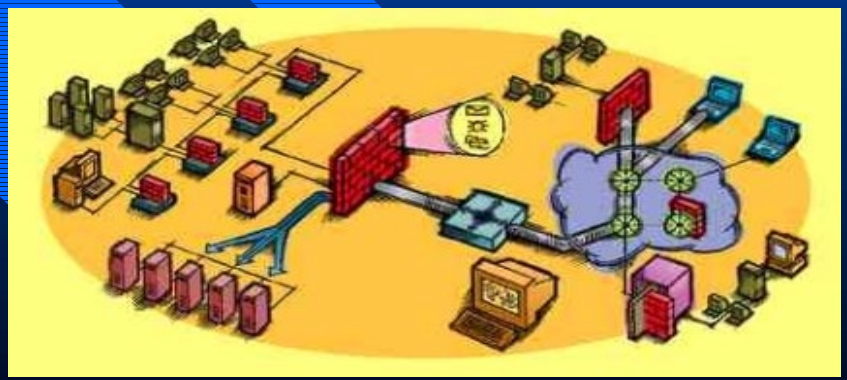
- Redes de computadoras
- Computadoras multiprocesadores
- Procesos cooperantes
- Celdas de manufactura
- Redes inalámbricas - computación móvil

Redes y sistemas distribuidos

- WAN
  - computadoras pertenecientes a organizaciones diferentes
  - distancia física: más de 10kms
  - nodo: es una red
- LAN
  - computadoras pertenecientes a una sola organización
  - distancia física: menor a los 10 kms.
  - nodo: estación trabajo, servidor, PC, etc.

Redes y sistemas distribuidos

- MAN
  - Red metropolitana
  - Tecnología: fibra óptica (FDDI)
  - Transmisión de voz, imágenes y datos hasta 50 km.
  - Diferencia con WAN: latencias pequeñas



Relación WAN con sistemas distribuidos

- Confiabilidad del intercambio de datos
- Selección de rutas de comunicación
- Control de tráfico
- Prevención de cuellos de botella
- Seguridad

Relación LAN con sistemas distribuidos

- Broadcasting y sincronización
- Elección
- Detección de terminación
- Asignación de recursos
- Exclusión mutua
- Deadlock
- Mantenimiento archivos distribuidos

Computadoras multiprocesadoras

- Computadora que consiste de diferentes procesadores generalmente ubicados dentro de un mismo espacio físico
- Procesadores homogéneos
- Pequeña escala geográfica
- Objetivo principal:
  - mejorar la velocidad del cálculo

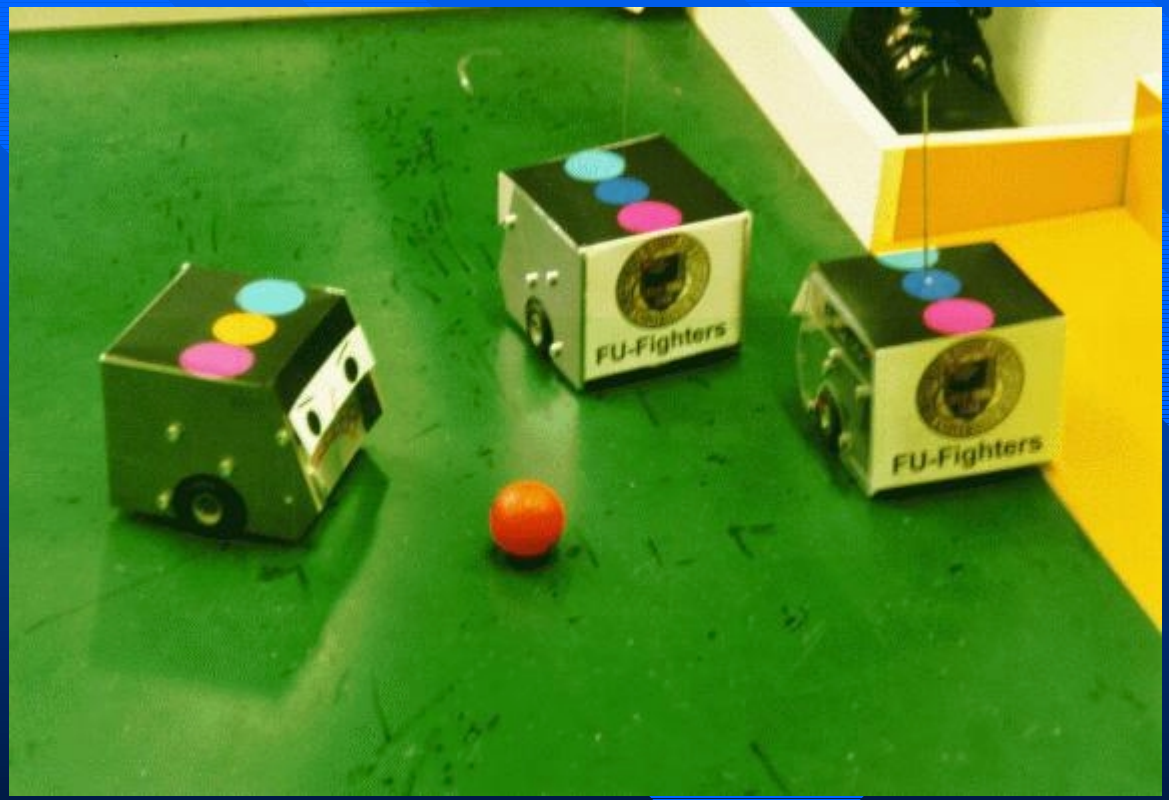
Los Sistemas Distribuidos y los Multiprocesadores

- Implementación sistema envío de mensajes
- Implementación memoria virtual compartida
- Balance de carga
- Tolerancia a fallas

Procesos Cooperantes

- Procesos que interactuan para la solución de un determinado problema, o para proporcionar un servicio
- Comparten memoria en común
- Trabajan sobre el mismo procesador
- Ejemplo: sistemas operativos, (daemons)

Ejemplo: robocup

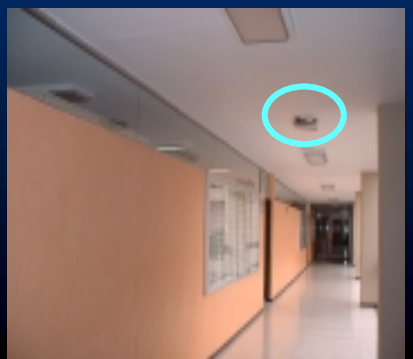


Sistemas distribuidos y procesos cooperantes

- Exclusión mutua
- Deadlocks
- Sincronización
- Intercambio de información
- Asignación recursos

Computación Móvil

- Wireless Local Area Networks o WLAN's.
- Ausencia de cables como medio de comunicación.
- Envío/recepción de ondas electromagnéticas que viajan del emisor al receptor a través del espacio.
- Computadoras desatadas
- Computadoras y aplicaciones móviles.
- Computadoras nómadas.



Relación con Sistemas Distribuidos

- Ruteo
- Pagging y roaming
- Información de la ubicación de la unidad móvil (almacenamiento y actualización)
- Consistencia
- Seguridad
- Transferencias de llamadas

Definición algoritmos distribuidos

- Abstracción lógica de un sistema distribuido, se habla de un conjunto de procesos y de líneas de comunicación virtuales
- Conjunto de procesos que, comunicandose a través de mensajes, llevan a cabo una tarea en común.

algoritmo distribuido = procesos + mensajes

Comentarios algoritmos distribuidos

- Se habla de algoritmos concurrentes ejecutados en diferentes procesadores.
- Originalmente los algoritmos eran diseñados para ejecutarse procesadores distribuidos en un área grande.
- Hoy en día incluye algoritmos usados en redes de área local y multiprocesadores que comparten memoria.

Características algoritmos distribuidos

- Desconocimiento del número de procesos
- Desconocimiento del estado global
  - algoritmo centralizado puede tomar decisiones basados en el estado del sistema
  - nodos en un sistema distribuido solo tienen acceso a su estado local y no al estado global del sistema entero
- Entradas independientes en sitios diferentes
  - proceso P_1 recibe entradas diferentes a proceso P_2
- Varias programas ejecutandose al mismo tiempo, empezando en tiempos diferentes y operando a diferentes velocidades

Características algoritmos distribuidos

- No determinismo en el procesador (processor nondeterminism)
  - un programa centralizado puede describir su cálculo de acuerdo a su entrada, dado un programa y su entrada sólo un cálculo es posible
  - la ejecución de un sistema distribuido es, generalmente, no deterministica, debido a las posibles diferencias en la velocidad de ejecución de los componentes del sistema

Características algoritmos distribuidos

- Falta de un sistema de tiempo global
  - sistema centralizado, los eventos estan totalmente ordenados de forma natural, para cada par de eventos uno ocurre antes o después de otro
  - la relación temporal de los eventos que constituyen la ejecución de un algoritmo distribuido no es total, dado dos eventos no se sabe cual ocurre antes o después
- Tiempos entrega de mensajes diferentes
  - un proceso emite m_1 a dos procesos, dependiendo de la ruta m_1 puede llegar en tiempo diferente a los procesos
- Orden entrega de mensajes desconocido
- Fallas en la comunicación y en los procesos

Elementos Algoritmos Distribuidos

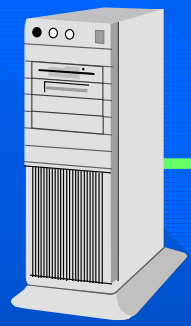
- Procesos
  - reciben, manipulan, transforman y emiten datos
- Vías de comunicación
  - medio sobre el cual circulan los datos y que forman una red local dotado de propiedades estructurales y dinámicas.

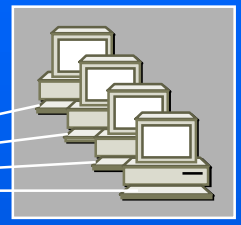
Los procesos

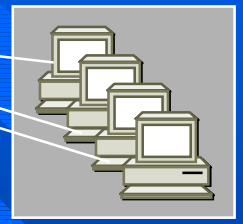
- Término introducido por Dijkstra en 1968 para modelar las relaciones entre diferentes unidades de ejecución independientes que deben compartir recursos comunes, (materiales y lógicos)
- En sistemas/algoritmos distribuidos:

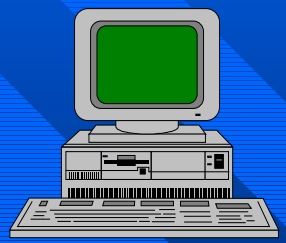
  unidad de ejecución elemental de un algoritmo distribuido o paralelo; diversas de esas unidades pueden ejecutarse simultáneamente, y cada una es indivisible.

Ejemplos procesos









Características de los procesos

- Permite abstraer la noción de actividad
- Se consideran procesos secuenciales
  - presentan un flujo de control único
- Ya que se trata de un conjunto de procesos paralelos, deben tener la posibilidad de ser sensibles al paralelismo de su ambiente.
- Es necesario dotarlos de una estructura de control no-deterministica que los autorice a atender un evento de entre varios posibles.
  - lenguajes como CSP o ADA propocionan dichas estructuras

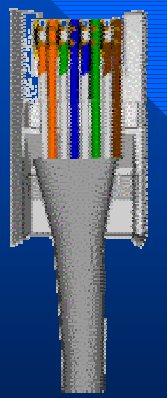
Ejemplo proceso en ADA

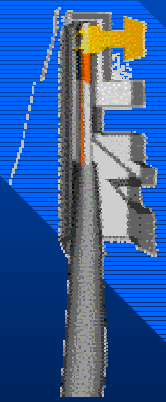
```
task P is
        entry s1, s2;
end P;
task body P is
        <declaración variables internas>
begin
        loop select
                when C1 => accept S1 do
                                        <descripcion del servicio S1>
                                        end;
            or  when C2 => accept S2 do
                                        <descripcion del servicio S2>
                                        end;
                end select;
        end loop;
end  P;
```

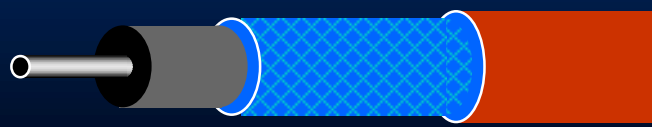
Las vías de comunicación

- Medio a través del cual viajan los mensajes
- Sistema distribuido: vías de comunicación virtuales
- Propiedades:
    1. Propiedades estructurales
    2. Propiedades comportamentales

Ejemplos vias comunicación



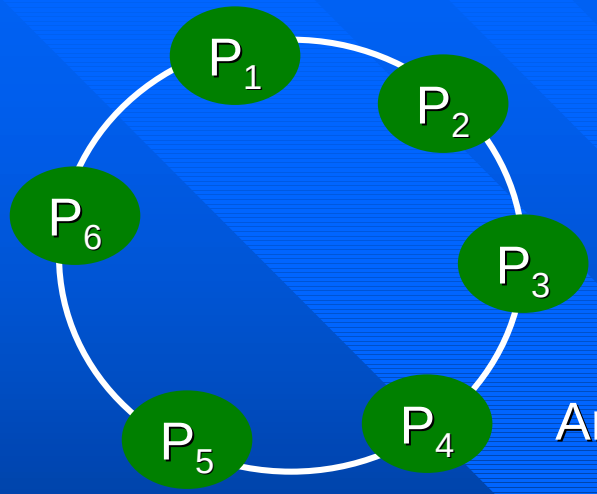




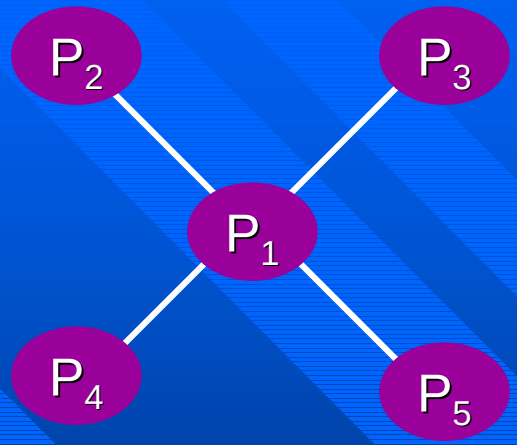
Propiedades estructurales

- Son de naturaleza topológica
- Se refiere a las mallas de comunicación
- Toda topología es posible según el problema tratado y el algoritmo distribuido que lo resuelve.
- Estructuras más comunes:
    1. Anillo
    2. Estrella
    3. Árbol
    4. Completo

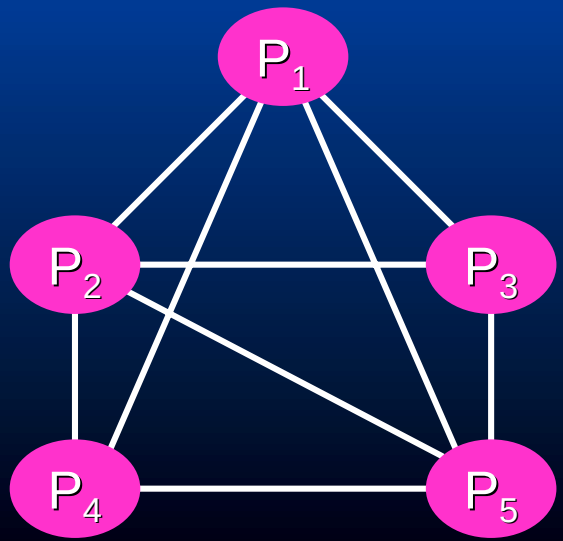
Estructuras más comunes



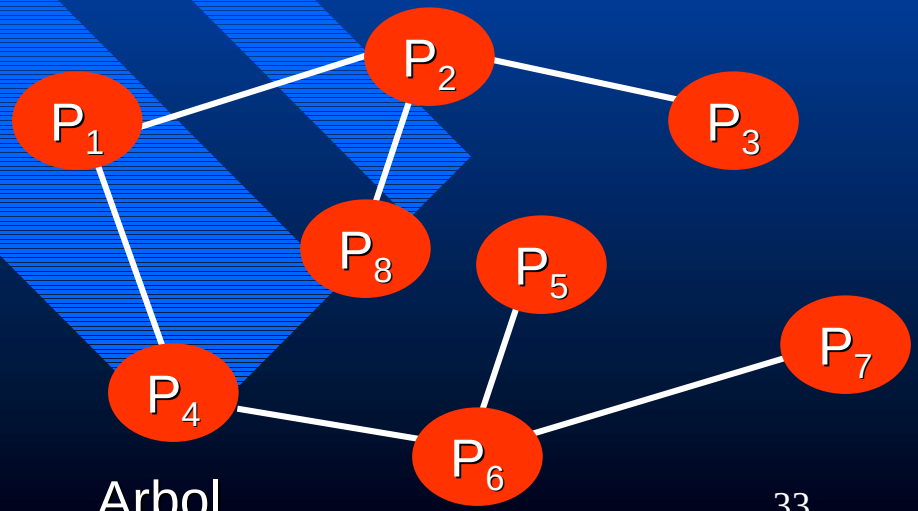
Anillo



Estrella



Completo



Arbol

Principales propiedades comportamentales

- Transmisión se hace sin duplicación de mensajes
- Transmisión sin alteración de mensajes
- Entre dos procesos el orden de recepción de mensajes es idéntico a su orden de emisión: no hay desplazamientos
- Tiempo espera de un mensaje es finito, aunque aleatorio, (i.e. no hay perdida de mensajes).

Atributos diferencian algoritmos distribuidos

Caracteristicas propias de los algoritmos distribuidos que permiten evaluar y comparar diferentes algoritmos que llevan a cabo la misma función.

- Método de comunicación entre procesos
- El modelo del tiempo
- El modelo de fallas
- El grado de distribución
- Estado global o local
- Los problemas a los que están dirigidos
- Complejidad (tráfico generado)

Metodo comunicación entre procesos

- Memoria compartida
  - sincronia en base a variables compartidas
- Mensajes punto a punto y/o broadcast
  - sincronia por envio de mensajes
- Ejecución de procesos remotos (RPC)
  - RPC proporciona la base de la sincronia

El modelo del tiempo

- Completamente sincronos
- Completamente asíncronos
- Parcialmente sincronos

Modelo sincrono

- Procesos ejecutan paso por paso, y cada paso se lleva a cabo simultaneamente.
- La ejecución del sistema se divide en turnos.
  - en cada turno cada proceso puede enviar un mensaje a cada vecino
  - mensajes son entregados
  - cada proceso realiza un cálculo en base a los mensajes que recibieron
- Esto no ocurre en los sistemas distribuidos actuales, pero se puede simular.

Modelo asincrono

- Un sistema es asincrono si no hay una cota superior para
  - cuanto tiempo toma que un mensaje en llegar
  - cuanto tiempo pasa entre dos “cálculos” de un proceso
- Se asume que los componentes llevan a cabo sus pasos en forma arbitraria, con velocidades arbitrarias
- Más difícil de programar que el sincrono debido a la incertidumbre del orden de los eventos.
- Modelo permite al programador ignorar consideraciones relacionadas con el tiempo

Modelo parcialmente sincrono (timing based)

- Se asumen algunas restricciones en los tiempos relativos de los eventos.
  - sin embargo la ejecución no es completamente paso a paso como en el caso del modelo sincrono
- Modelos más realistas, pero más difíciles de programar.
- Algoritmos bajo este modelo pueden ser eficientes, pero pueden ser frágiles desde el punto de vista de que no correran correctamente si las consideraciones de tiempo no son respetadas.

El modelo de fallas

- Sistema completamente fiable
  - no se considera ningún tipo de falla
- Sistema tolera algunas fallas
  - sistema debe considerar algunas fallas bien definidas
- Fallas bizantinas
  - cuando un procesador falla, este se comporta de forma completamente arbitraria

Grado de distribución

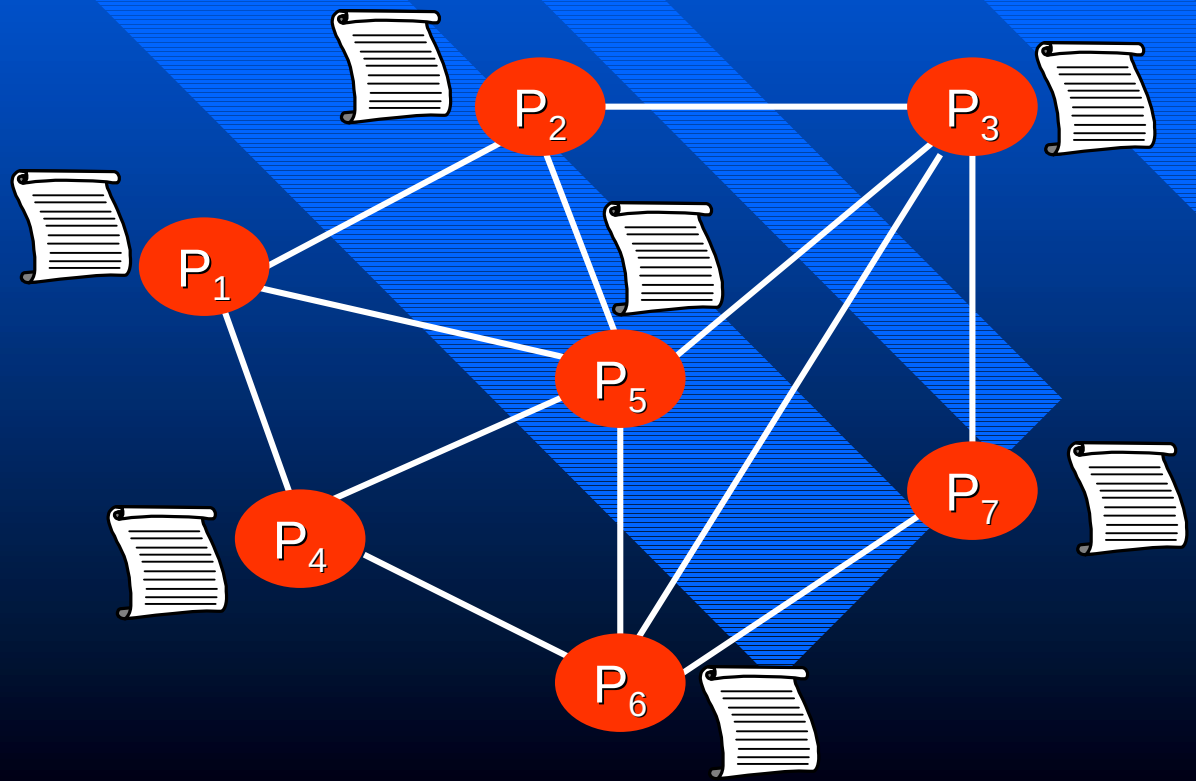
- Se trata de ver si el algoritmo esta más o menos distribuido.
- Noción ligada a la simetría de los roles que llevan a cabo los diferentes procesos
- Podemos hablar de diferentes niveles de simetría:
  - no simétricos
  - simetría del código
  - simetría fuerte
  - simetría total

Los niveles de simetría

- No simétrica
  - cada proceso ejecuta un código diferente
- Simetría código
  - procesos ejecutan el mismo código, pero cada código hace referencia al “nombre” del proceso donde se ejecuta y todos los nombres son diferentes
- Simetría fuerte
  - procesos ejecutan el mismo código y el nombre del proceso no aparece
  - de acuerdo a los mensajes recibidos el comportamiento puede ser idéntico o diferente

Los niveles de simetría

- Simetría total
  - los procesos ejecutan el mismo código lo que provoca comportamientos idénticos



Estado global o local

- Un algoritmo distribuido se puede diseñar a partir de un algoritmo centralizado al cual se le añaden reglas de administración de variables duplicadas.
- En este caso un proceso cuenta con un conocimiento del estado global del sistema
  - aun si la información que representa dicho estado no es necesaria
- Un criterio de comparación (y evaluación) debe ser, que un proceso del algoritmo tome decisiones sin necesidad de contar con un estado global
  - reducir el número de mensajes intercambiados puede permitir asegurar una mejor tolerancia a fallas.

Los problemas a los que están dirigidos

- Problemas de aplicación
- Problemas de control

Algoritmos distribuidos aplicación

- Son los algoritmos que definen una aplicación
- Representan la interfaz final entre los usuarios y el sistema distribuido
- Se apoyan en arquitecturas de software como:
  - CORBA: Common Object Request Broker Architecture
  - COM: Component Object Model
  - EJB: Enterprise JavaBeans

Algoritmos distribuidos de control

- Algoritmos dedicados a llevar a cabo una función de control.
- La palabra control es diferente a la de cálculo.
  - los algoritmos no calculan un resultado el cual, una vez obtenido, termina con los procesos
- Los algoritmos prestan un servicio y lo pueden prestar de nuevo, en todo momento que sea necesario.

Algoritmos distribuidos control

- Están por abajo de las aplicaciones
- Proporcionan dos tipos de servicios
  - Proveedor de primitivas
    - » exclusión mutua
    - » envío/recepción mensajes
    - » control de concurrencia
    - » administración de archivos
  - Observadores de propiedades
    - » interbloqueo
    - » terminación de la ejecución
    - » recolectores de basura

Algoritmos de aplicación y control

CTL_i : control de la i-ésima aplicación

A_i: aplicación

Complejidad algoritmos distribuidos

- Para comparar diferentes algoritmos que resuelven un mismo problema es necesario medir el consumo de recursos de cada uno de los algoritmos.
- Entre menor sea consumo, mejor es el algoritmo.
- Cuatros medidas de complejidad
  - el número de mensajes
  - complejidad bit
  - la cantidad de tiempo
  - la complejidad de espacio

Medidas complejidad algoritmos distribuidos

- Complejidad bit
  - longitud de los mensajes intercambiados
  - mientras más grandes sean los mensajes, más costoso será su manejo
- La cantidad de tiempo
  - se cuentan con diferentes definiciones en la literatura
  - en esta presentación se asume un tiempo ideal de computo:
    - » el tiempo de procesamiento de un evento es de cero unidades
    - » el tiempo de transmisión (envío/recepción mensajes) es de una unidad de tiempo

Medidas complejidad

- El número de mensajes
  - numero total de mensajes intercambiados por el algoritmo
- La complejidad de espacio
  - la complejidad de espacio en un algoritmo equivale a la cantidad de memoria que un proceso necesita para poder ejecutarse
  - el espacio en un proceso es el logaritmo del número de estados del proceso

Tráfico generado

- Un algoritmo es más interesante cuando presenta un mejor desempeño.
- Este desempeño se puede medir por
  - el número de mensajes intercambiados
  - la carga que presentan las vías de comunicación
  - el tiempo de espera de los procesos
- Los resultados con respecto al tráfico permiten comparar los algoritmos.

Ejecuciones algoritmos distribuidos

- Existen varias nociones de lo que es la ejecución de un algoritmo distribuido
- N. Lynch propone lo siguiente:
  - la asignación de un estado del sistema es la asignación de un estado a cada proceso en el sistema
  - una asignación de un mensaje es la asignación de un mensaje (posiblemente vacío) a cada canal
- La ejecución del sistema esta definida como una secuencia finita:

$$C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots ,$$

Modelando ejecuciones

- Secuencia: $C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots$,
  - C_i es un asignamiento de estado, representa el estado del sistema despues de i vueltas
  - M_i es un asignamiento de mensaje, representa los mensajes enviados y recibidos en la vuelta i
  - N_i es un asignamiento de mensaje, representa el estado del sistema despues de i vueltas

Métodos demostración algoritmos

- El método más importante de demostración involucra severaciones invariantes.
- Una severación invariante es una propiedad del estado del sistema (el estado de todos los procesos) que es verdad en cada ejecución, después de cada vuelta.
  - se permite que se mencione el número de vuelta completa en la severación, de tal forma que se pueden hacer afirmaciones acerca del estado después de cada número i de vueltas

Otro metodo demostración algoritmos

- Otro metodo importante es el de simulaciones
  - el objetivo es mostrar que un algoritmo sincrono A “implementa” otro algoritmo sincrono B, desde el punto de vista que produce el mismo comportamiento de entrada/salida
  - la correspondencia entre A y B se expresa por una severación que relaciona los estados de A y B, cuando los dos algoritmos comienzan con las mismas entradas y corren con bajo el mismo patrón de fallas para el mismo número de vueltas.

Metodos diseño algoritmos distribuidos

- Existen diferentes metodos para diseñar y construir un algoritmo distribuido.
- Se siguen dos metodos:

1. a partir de ciertas propiedades de invariancia que caracterizan el problema planteado, se definen procesos de tal forma que en cada etapa de la construcción del algoritmo se conservan las invariantes
2. el otro es de naturaleza empírica y consiste en construir un algoritmo y después verificar que soluciona el problema planteado

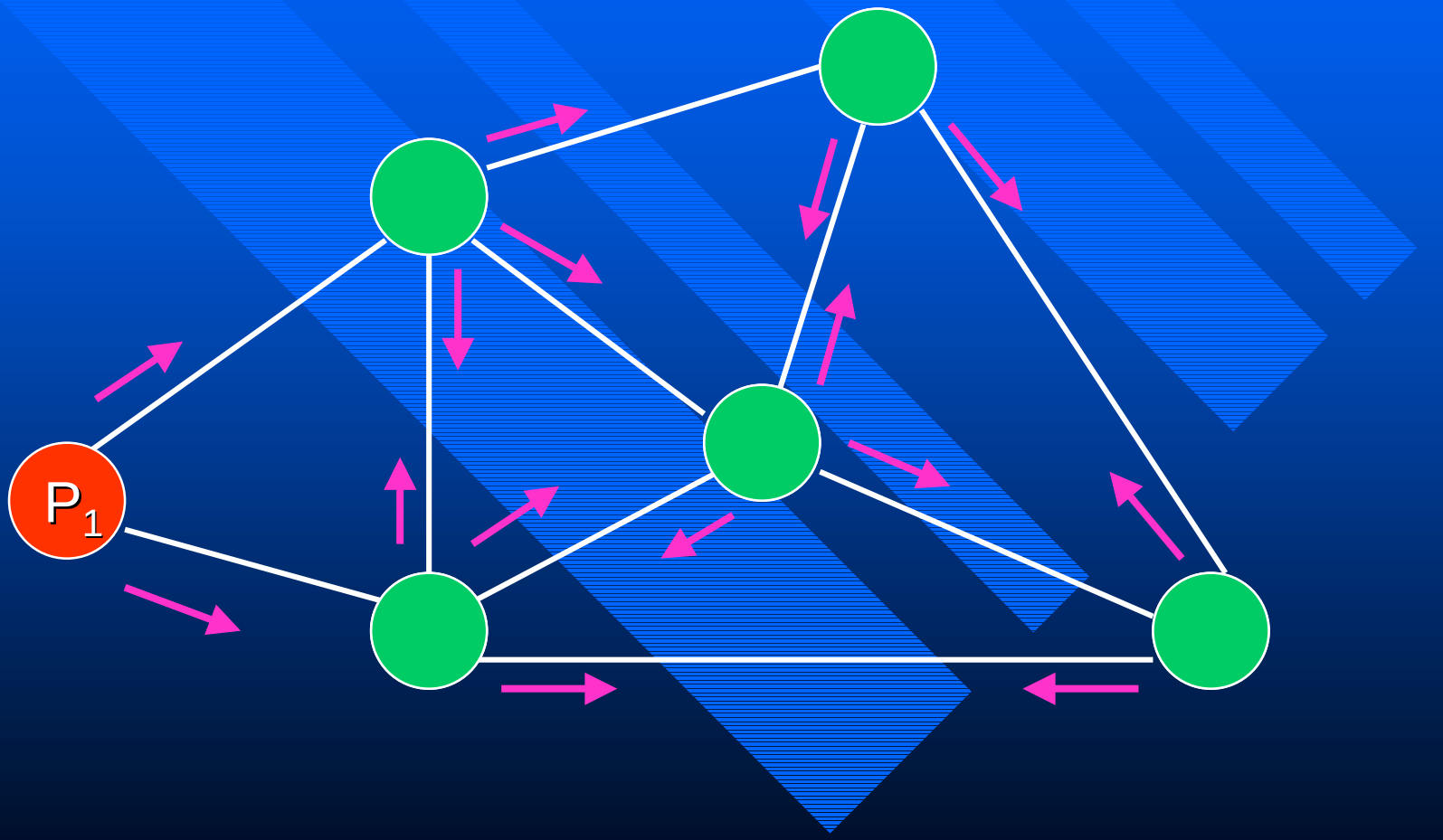
Técnicas diseño algoritmos distribuidos

- Independientemente de la metodología usada, la algorítmica distribuida usa técnicas clásicas que se encuentran en las redes.
  - acuse recepción mensaje, broadcast a un conjunto de procesos, etc
- Existen conceptos propios a la algorítmica distribuida
  - la difusión de cálculo
  - la ficha circulante
  - el estampado

La difusión del cálculo

- Propuesto por Dikstra y Schloten en 1980
- Procesos conectados de forma arbitraria por las vías de comunicación.
- Inicialmente un proceso emite mensajes a sus vecinos.
- El resto de los procesos solo puede emitir mensajes si ha recibido uno.
- Algunos algoritmos distribuidos realizan un control (terminasión, interbloqueo) utilizan este principio sobre un árbol de recubrimiento.

Ejemplo difusión cálculo



La ficha circulante

- Se hace circular un privilegio dentro de un conjunto de procesos conectados a través de una topologia de anillo.
- El anillo puede ser definido estáticamente o se puede reconfigurar dinámicamente.
- Técnica base de algoritmos distribuidos como los de exclusión mutua y de terminación.

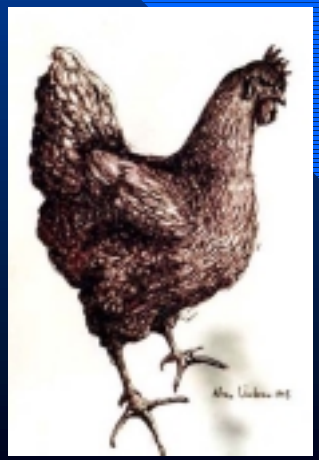
La estampilla

- Basado en el concepto de relojes lógicos de Lamport
- Utilizado en diferentes algoritmos distribuidos en los que las elecciones deben ser equiparables
  - algoritmos de exclusión mutua y de interbloqueo
- En caso de conflictos el proceso no debe ser el mismo para evitar situaciones de hambruna.
  - sistemas centralizados utilizan un mecanismo de acceso a la memoria central o a un reloj global
  - en los algoritmos distribuidos estos dispositivos centralizados no existen
  - Lamport propone un orden total de eventos basados en el concepto de estampilla de tiempo

Los relojes lógicos

- ¿Qué fue primero, el huevo o la gallina?
- Propuestos por Leslie Lamport

  Times, cloks and ordering of events in a Distributing System,
  Leslie Lamport, Comm. ACM, Vol. 21, No. 7, Julio 1978
- Basados en la relación paso antes



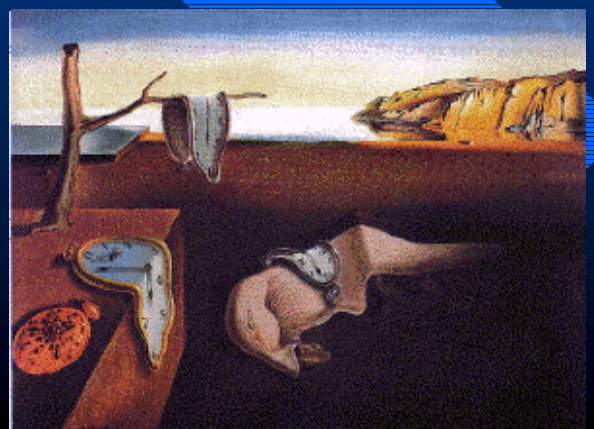
Características relación paso antes

1. Si ***a*** y ***b*** son eventos en el mismo proceso y ***a*** ocurre antes que ***b*** entonces:

$$a \rightarrow b \text{ es verdad}$$

2. Si ***a*** es el evento de envío de un mensaje por parte de un proceso y ***b*** es el evento de recepción de ese mensaje por otro proceso, entonces.

$$a \rightarrow b \text{ es verdad}$$



Características relación paso antes

3. Si $a \rightarrow b$ y $b \rightarrow c \Rightarrow a \rightarrow c$
4. Si dos eventos ***a*** y ***b*** ocurren en diferentes procesos que no intercambian mensajes, (ni siquiera indirectamente), entonces:

$a \rightarrow b$ es falso

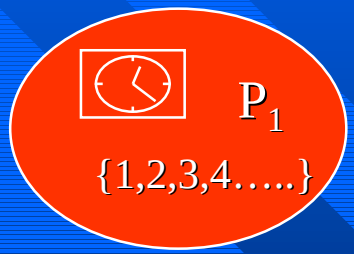
$b \rightarrow a$ es falso

por lo que se dice que

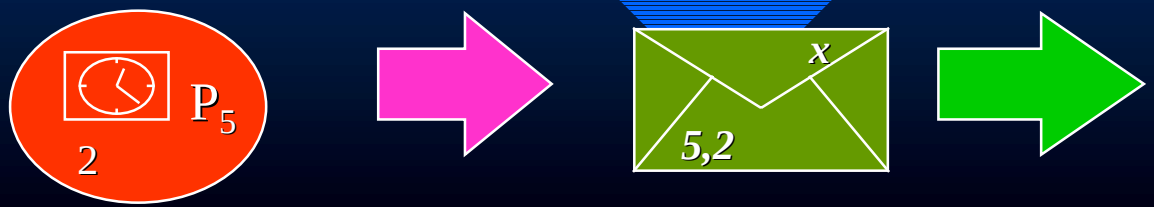
$a \rightarrow b$ (a y b son concurrentes)

La estampilla de Lamport

- Usadas para asignar tiempos a los eventos:
- Cada proceso P_i administra un reloj local r_i, que sirve para definir una medida del tiempo local P_i



- Todos los mensajes m enviados por proceso P_i son estampillados por el valor del reloj local r_i y el número i del proceso: [m, r_i, i]



Relación orden causal entre eventos

- El comportamiento proceso P_i en presencia de un reloj local r_i es el siguiente:
  - Si P_i recibe un mensaje [m, r_j , j] el reloj local r_i toma el valor:

    $r_i := \max(r_i , r_j) + 1$

    r_i : fecha de recepción del mensaje m
  - Si P_i envía un mensaje [m, r_i , i] el reloj local r_i es incrementado antes de incluirlo en el mensaje.

    r_i : fecha de emisión del mensaje m.
  - El reloj r_i puede ser incrementado entre dos eventos internos a P_i

Este mecanismo permite "fechar" las emisiones de mensajes, de forma coherente.

Orden causal total entre eventos

- El mensaje [m, r_i, i] es considerado como anterior al mensaje [m', r_j, j] si y solamente si:
  1. $(r_i < r_j)$

     la emisión de ***m*** precede a la de ***m'*** en el sistema de relojes lógicos
  2. $(r_i = r_j)$ ***e*** $(i < j)$

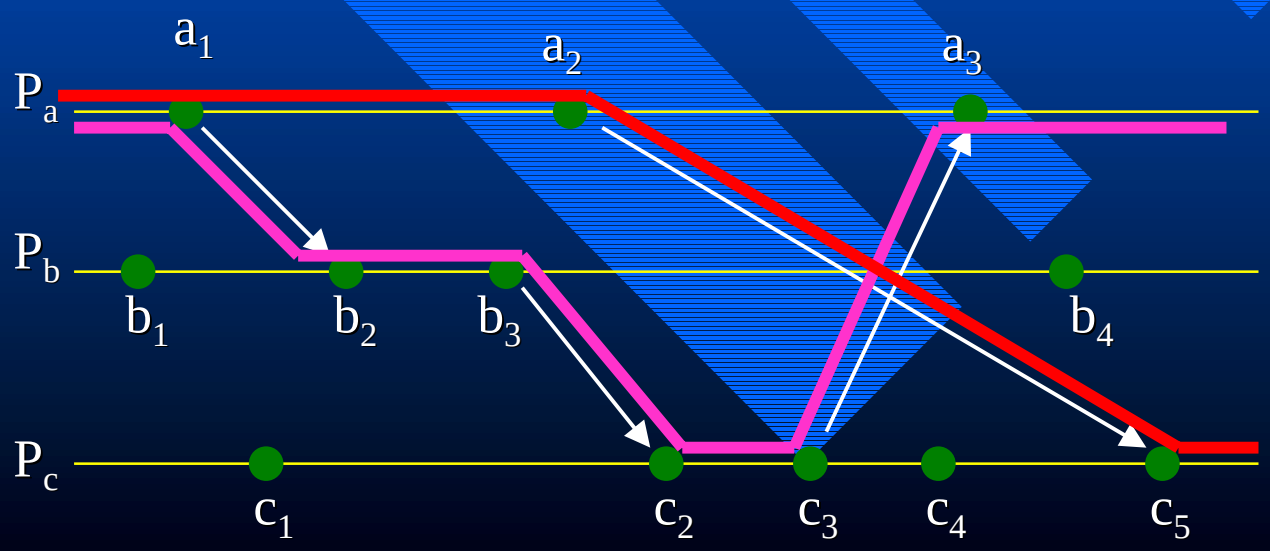
     las dos emisiones tienen la misma fecha lógica:

     2. 1 los eventos no están ordenados en tiempo lógico

     2.2. los eventos son ordenados tomando en cuenta los identificadores que emitieron los mensajes correspondientes.

Cadena causal

- Eventos de diferentes procesos unidos por una relación paso antes.
- Dos eventos ligados por $\rightarrow$ pertenecen al menos a una cadena orientada causal.
- El tiempo progresa a lo largo de una cadena causal.



Ejemplo aplicación

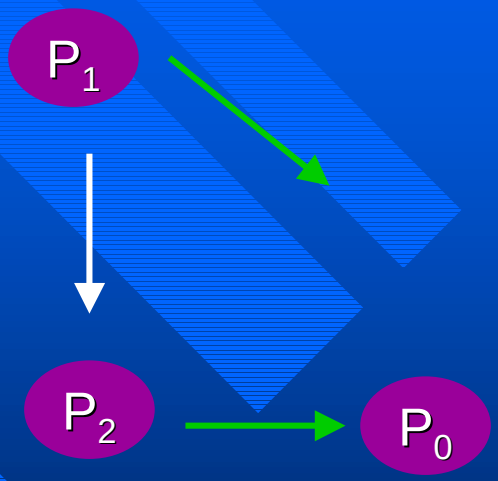
- Algoritmo de asignación de recursos
- Un recurso será otorgado a un proceso si éste cumple con las condiciones siguientes:
  1. El proceso al cual se le asignó un recurso debe liberarlo antes de que este sea asignado a otro proceso.
  2. Si existen diferentes peticiones del recurso deben de atenderse en el orden en que fueron elaboradas.
  3. Si cada proceso al que se le asignó un recurso eventualmente lo libera, entonces cada petición será eventualmente atendida.

Consideraciones del algoritmo

- Se asume que el recurso es inicialmente asignado a exactamente un proceso
- Se está hablando de peticiones *naturales*
- Las condiciones involucran el ordenamiento de los eventos
- Utilizar un proceso de administración central que garantice la asignación de recursos en el orden en que lleguen no funciona, a menos de que se efectuen algunas suposiciones

Solución centralizada

- **P_0** proceso administrador
- P_1 envia petición a
- P_1 envia mensaje a P_2
- Una vez que P_2 recibe el mensaje envia petición a P_0
- Petición de P_2 llega antes que la de P_1
- Entonces no se cumple la segunda condición



Solución: i*mplementar un sistema de asignación en base a relojes lógicos*

Suposiciones de la solución

- Para todo par de procesos, P_i y P_j , los mensajes enviados de P_i a P_j son recibidos en el mismo orden en que son enviados.
- Todo mensaje es eventualmente recibido
- Un proceso puede enviar un mensaje directamente a otro proceso
- Cada proceso cuenta con una spropia cola de peticiones donde almacena los mensajes que le llegan.
- Cada proceso administra su propia cola de peticiones, la cual nunca es vista por otros procesos

Suposiciones de la solución

- La cola de peticiones contiene inicialmente el mensaje [h_0,P_0, [*pedir recurso*]] donde:
  - **P_0** proceso al que inicialmente se le otorgó el recurso
  - **h_0** sello con un valor menor al valor inicial de cualquier reloj

El algoritmo

- Para pedir un recurso el proceso P_i envía, a todos los procesos, el mensaje:
  - [h_m , P_i , [*pedir recurso*]] (h_m = sello del mensaje)
  - y pone ese mensaje en su cola,
- Cuando proceso P_j recibe el mensaje
  - [h_m, P_i, [*pedir recurso*]]
  - lo coloca en su cola
  - y envía un mensaje de [*timestamp*, *ack*] a P_i
- Para liberar un recurso, P_i suprime de su cola cualquier mensaje:
  - [h_m , P_i , [*pedir recurso*]]
  - envía el siguiente mensaje a cada proceso:
  - [h_m+1, P_i , [*liberar recurso*]]

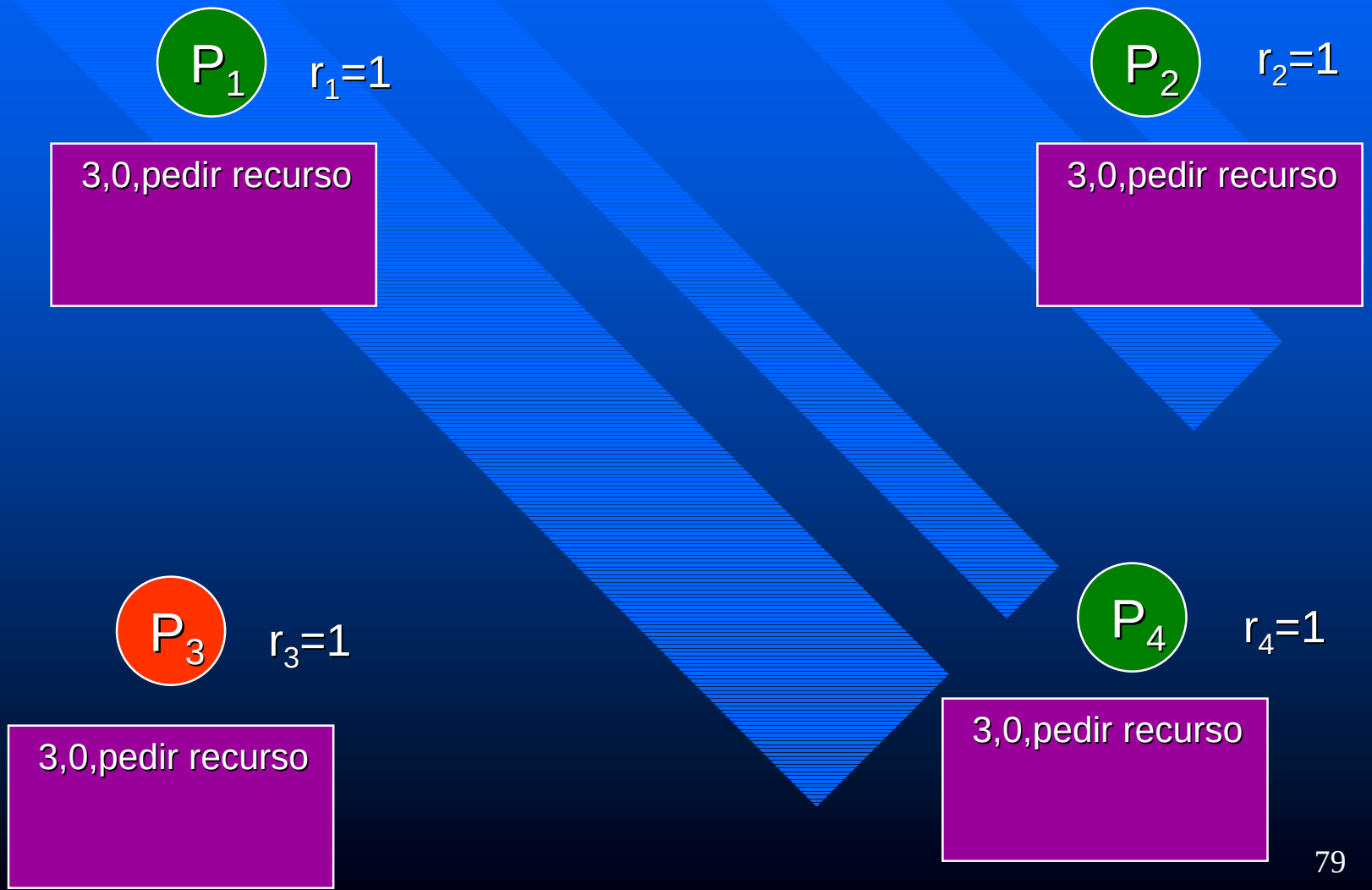
Continuación del algoritmo

- Cuando P_j recibe un mensaje [*liberar recurso*] de P_i suprime de su cola todos los mensajes:
  - [h_m , P_i , [*pedir recurso*]]
- Al proceso P_i se le asigna el recurso cuando se cumplen las dos condiciones siguientes:
  - existe un mensaje [h_m, P_i, [*pedir recurso*]] en su cola, el cual se encuentra antes, (desde el punto de vista relación total), de cualquier otra petición
  - P_i ha recibido un mensaje de cualquier otro proceso con un sello posterior a h_m

Estas condiciones son probadas localmente en cada proceso P_i

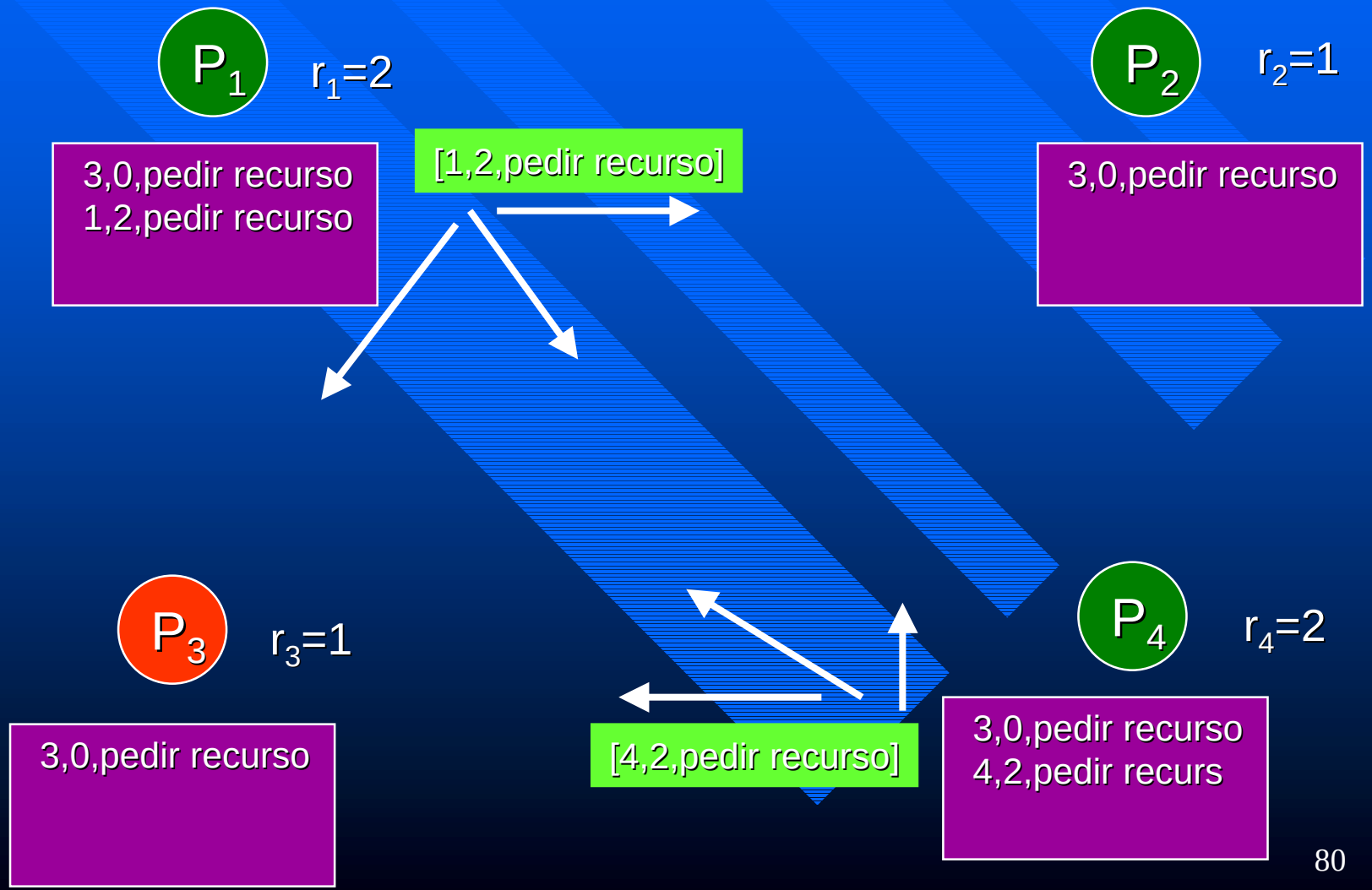
Ejemplo ejecución

P_3 cuenta con el resurso



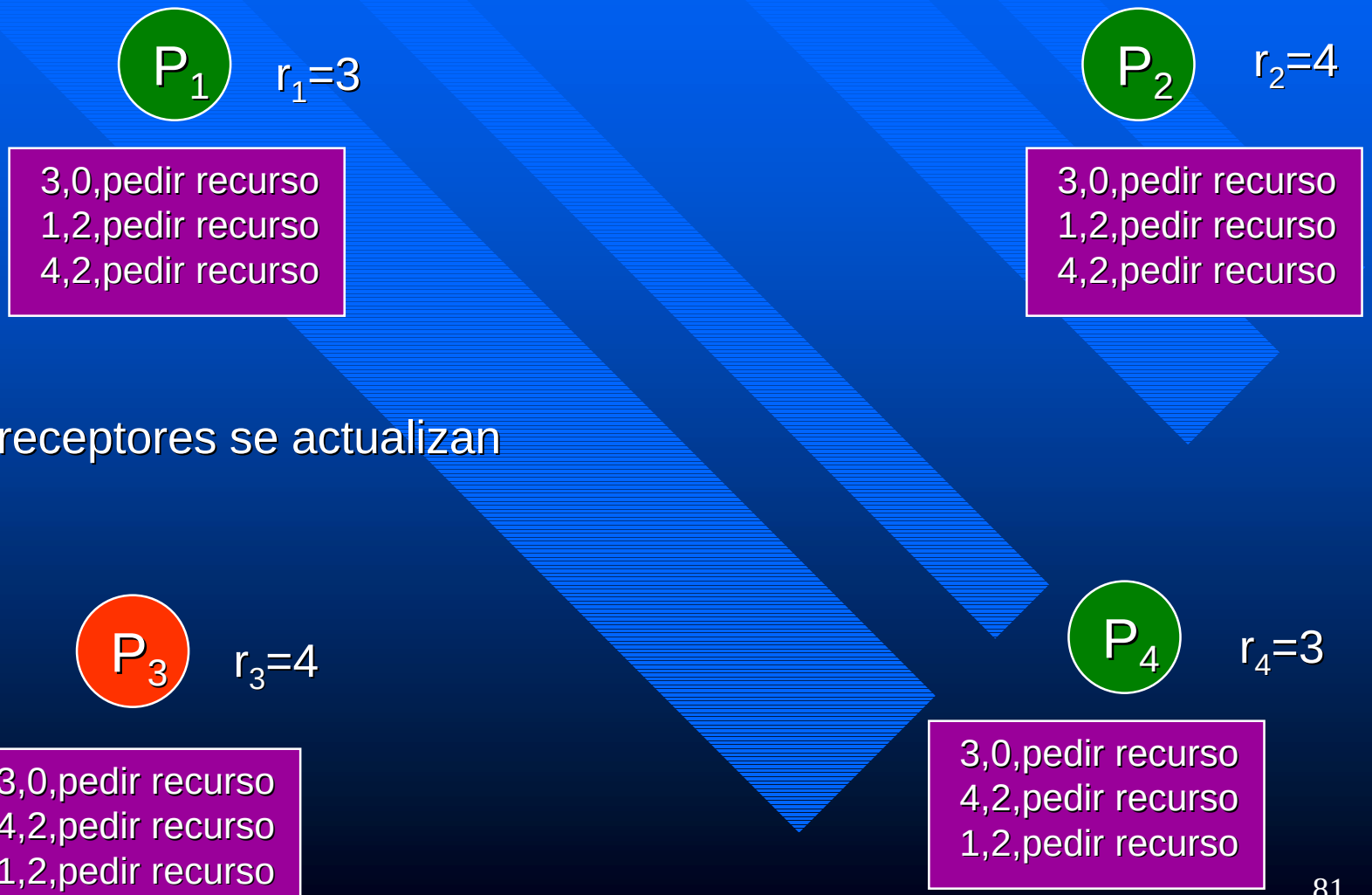
Ejemplo ejecución

P_1 y P_4 solicitan el resurso



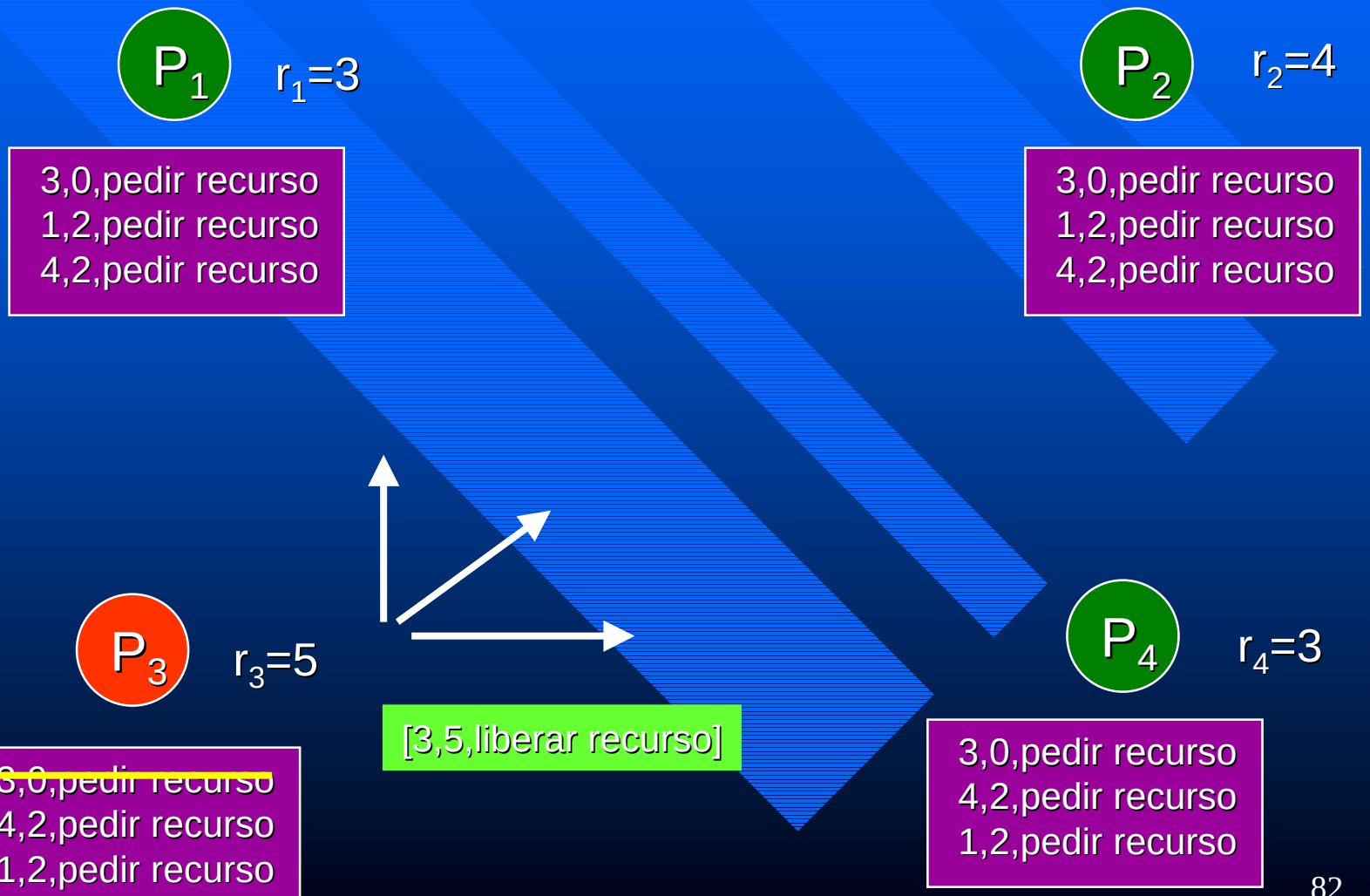
Ejemplo ejecución

Los mensajes llegan en tiempo diferentes



Ejemplo ejecución

Se libera el recurso

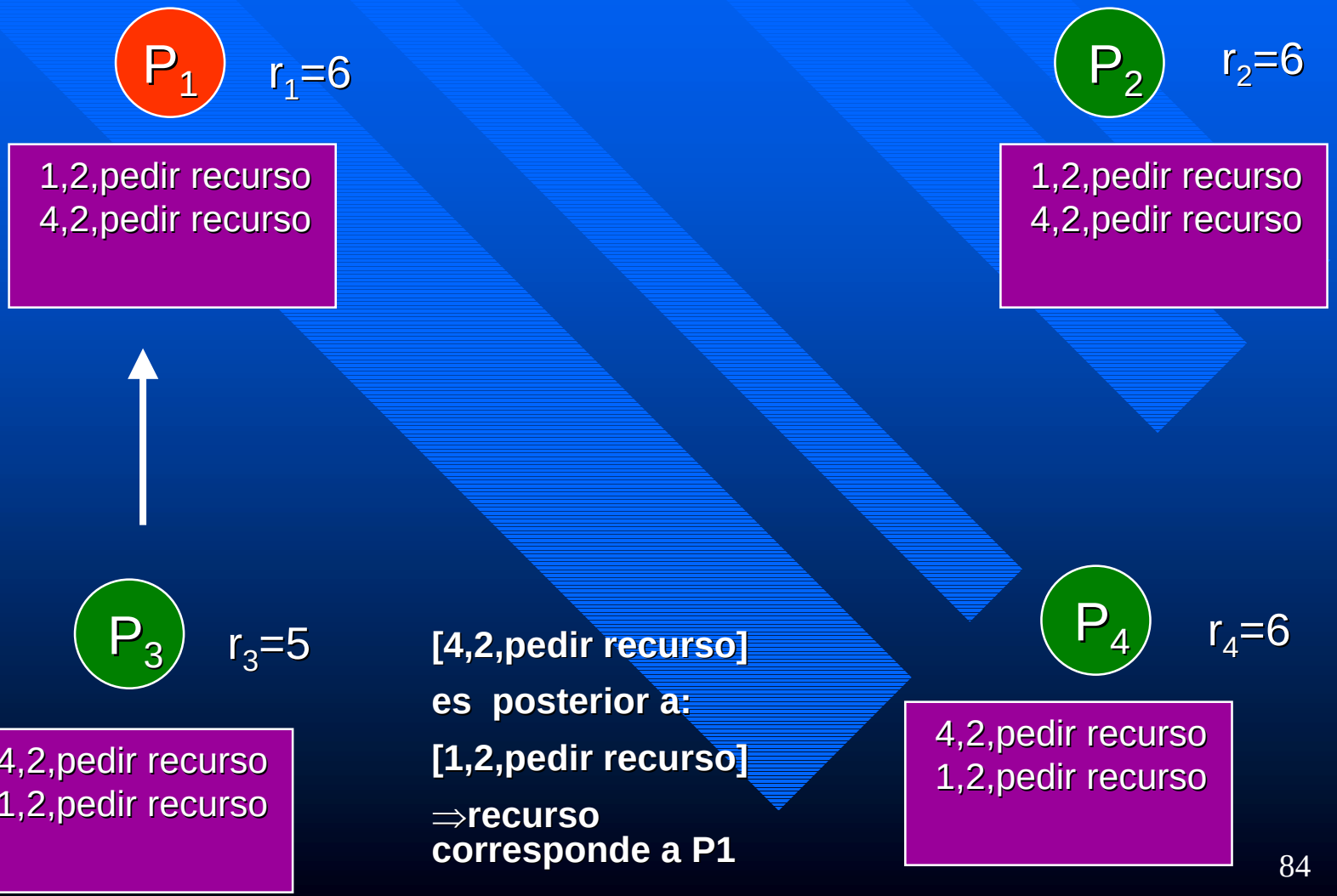


Ejemplo ejecución

Llega mensaje de liberación recurso

Ejemplo ejecución

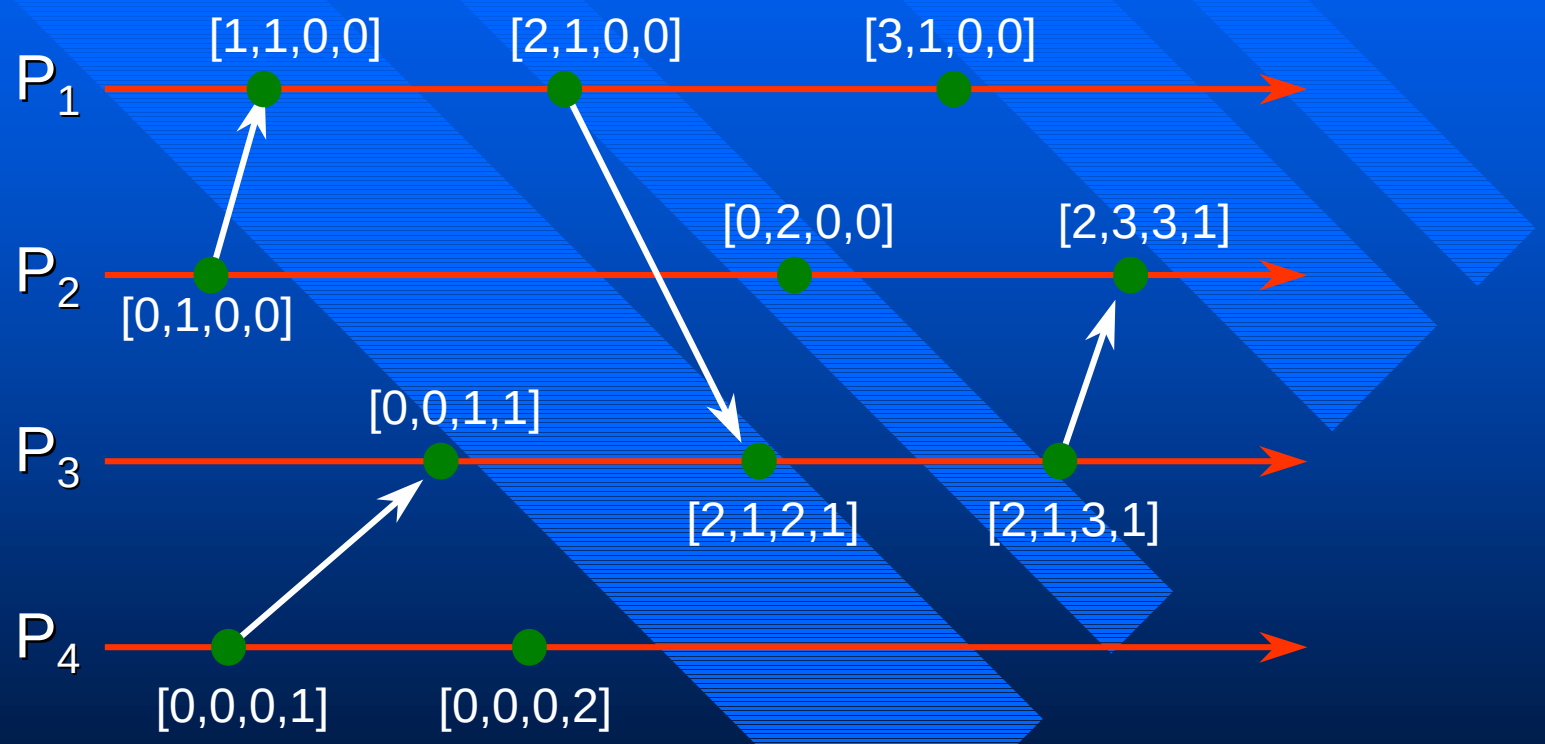
Se decide a quien le corresonde el recurso y se le envia



Otra opción: relojes vectoriales

- Propuestos por Friedmann Mattern en 1980
- Se asocia un vector Vi a cada proceso Pi (o sitio Si) i = 1,2,...n
- Inicialmente: Vi = [0,0,..., 0]
- Por cada evento local en Pi:
  - Vi[i] := Vi[i] + 1
- Cada mensaje local m porta una estampilla Vm (Vm = Vi del emisor)
- Cuando un proceso Pi recibe (m, Vm) de Pj
  - Vi[i] := Vi[i]+1
  - Vi[j] := max(Vi[j], Vm[j]) para j=1,2,...n-1

Ejemplo relojes vectoriales



Algunos algoritmos distribuidos de control

- Algoritmos de exclusión mutua
- Algoritmos de elección
- Algoritmos de ruteo
- Algoritmos de detección de terminación
- Algoritmos de detección de interbloqueo
- Algoritmos de cobertura de árbol (spanning-tree)
- Algoritmos de recorrido de gráfos
- Algoritmos de flujo maximal

La exclusión mutua

- Exclusión mutua: asegurar que sólo un proceso tiene acceso a un recurso compartido por varios procesos en un momento dado
- Sección crítica: parte del código donde el proceso utiliza el recurso compartido
- Procesos no comparten memoria en común
- Procesos se comunican a través de mensajes

Algoritmos Exclusión Mutua

- El algoritmo de LeLann
- El algoritmo de LeLann tolerante a fallas
- El algoritmo de Ricart y Agrawala
- El algoritmo de Misra

El algoritmo de LeLann

- Se hace circular una ficha
- El proceso que tiene la ficha esta en S.C.
- El algoritmo es el siguiente

esperar [ficha] de Pi-1

<Código Sección Critica>

enviar [ficha] a Pi+1

Ejemplo ejecución algoritmo

Algoritmos de elección

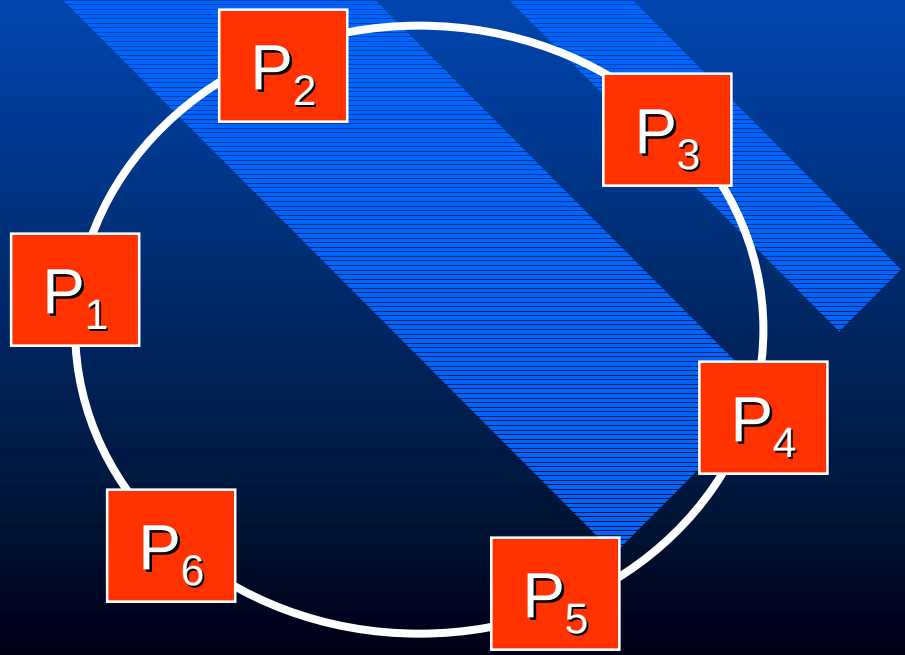
- En algunos algoritmos es necesario un proceso coordinador
- Si el proceso coordinador cae, es necesario que los procesos que queden elijan a uno de entre ellos
- Hipótesis general: cada proceso posee una identidad distinta de los otros, la cual usualmente es representada por un número
- La elección se realiza de acuerdo a dicha identidad

Ejemplos Algoritmos elección

- Algoritmo de LeLann
- Algoritmo de Chang & Roberts
- Algoritmo de Hirschberg y Sinclair
- Algoritmo Bulley (abusador)
- Algoritmo de Dolev, Klawe y Rodeh

Algoritmo Chang & Roberts

- Procesos agrupados en anillo unidireccional
- Cada proceso tiene un identificador único
- Se conoce el número total de procesos



Principo del algoritmo

- Cada proceso Pi envía su número a su vecino izquierdo Pj
- Cuando Pj recibe el mensaje compara el número con el suyo y envía el mejor a su vecino izquierdo
- Proceso que recibe mensaje con su número sabe que su número ha dado la vuelta y es el elegido

Elementos del algoritmo

- Variables de los procesos
- Tipos de mensajes
- Primitivas de comunicación

Variables de los procesos

- ego id_proc;
  - identificador del proceso
- participa booleano;
  - inicializado en falso
- coordinador id_proc;
  - identificador del ganador

Tipos de mensajes

- Mensaje 1: [elección, x]
  - eleccion: mensaje que incluye el identificador de un candidato
  - x es candidato a la elección
- Mensaje 2: [elegido, x]
  - elegido: mensaje que incluye al ganador
  - x ganó la elección

Primitivas de comunicación

- Solo se maneja una primitiva
  - env-vi[x]
  - envía mensaje x al vecino izquierdo del proceso emisor

Código del algoritmo

- Inicio del algoritmo
  - alguien decide realizar una elección
- Recepción del mensaje
  - que hacer cuando llega un mensaje
- Notificación del ganador
  - notificar al resto de los procesos que el algoritmo termino y quien gano

Inicio del algoritmo

decisión provocar una elección
participa := verdadero;
env-vi [elección, ego]
fin-decisión

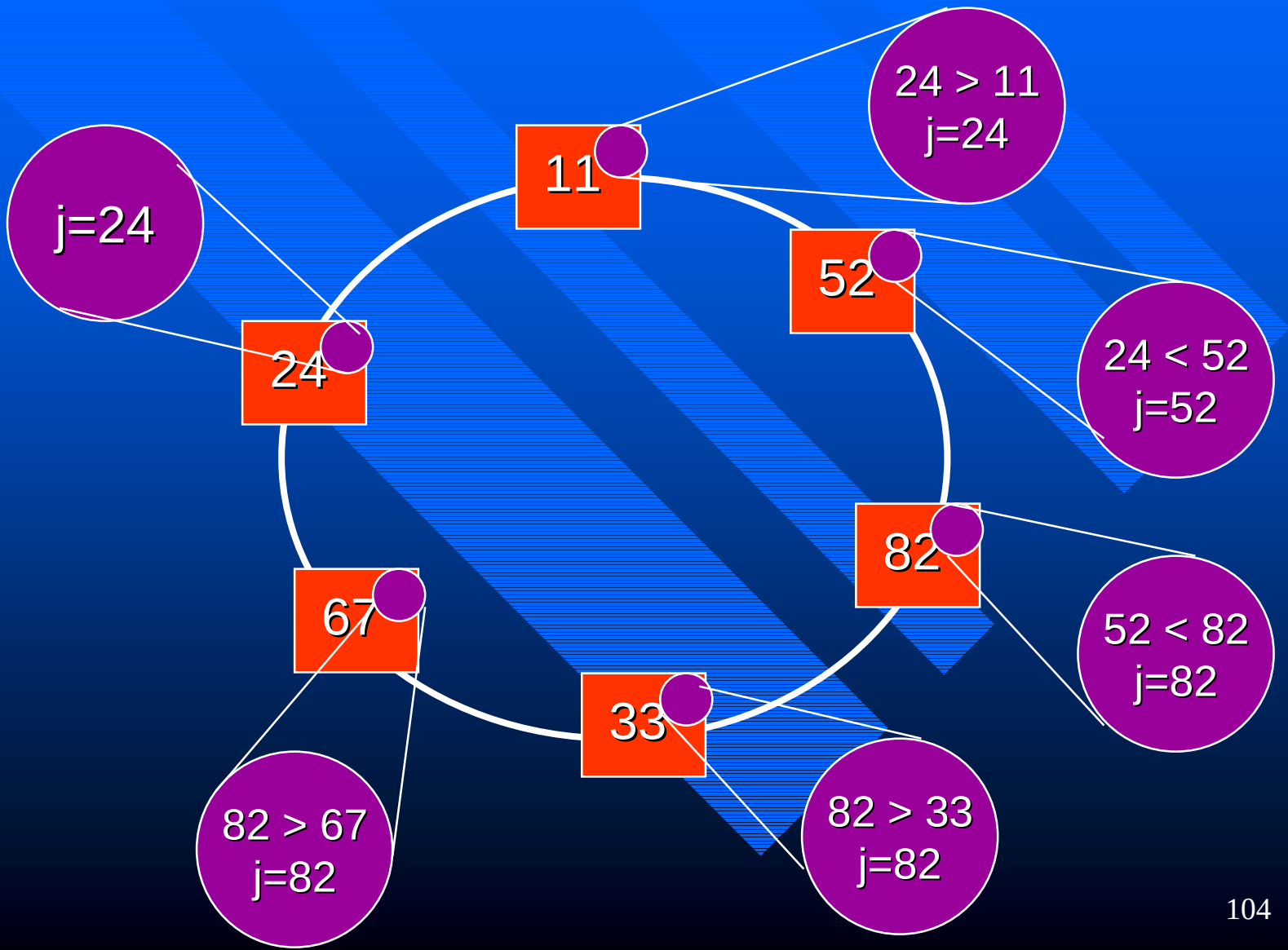
Recepción del mensaje

```
recepción_mensaje [elección, j]
    caso
        si (j > ego) entonces
            env-vi [elección, j]
            participa := verdadero
        si (j = ego) entonces
            env-vi [elección, j]
        si (j < ego) y (~ participa) ) entonces
            env-vi [elección, ego];
            participa := verdadero;
    fin-caso
fin-recepción_mensaje
```

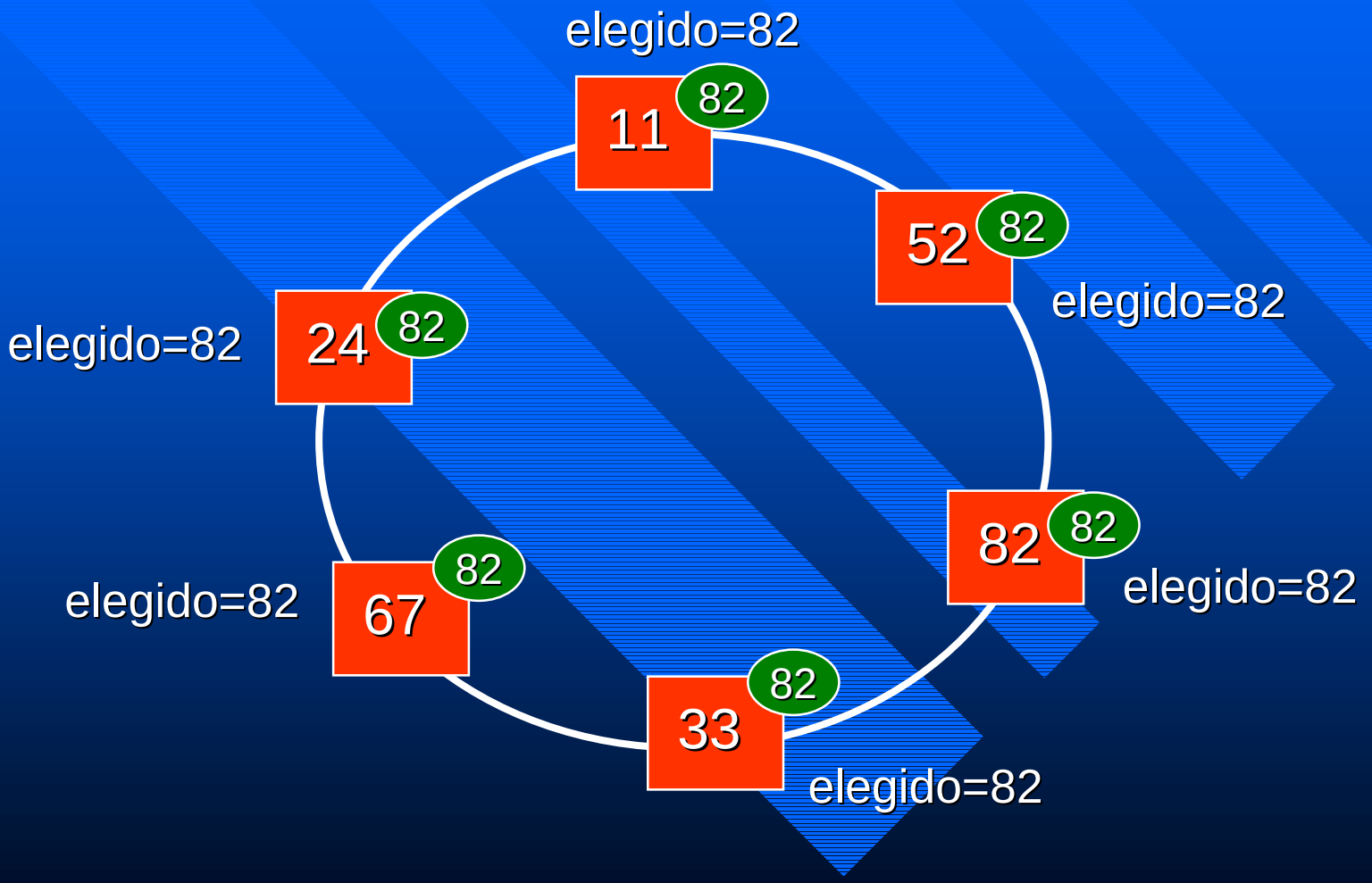
Notificación del ganador

```
recepción_mensaje [elegido, j]
        coordinador:= j;
        participante := falso;
        si (j ≠ ego) entonces
                env-vi [elegido, j];
        fsi
fin recepcion_mensaje;
```

Ejemplo ejecución algoritmo



Ejemplo ejecución algoritmo



El ruteo

- Definición: termino usado para describir el proceso a través del cual un proceso selecciona uno (o algunas veces más) de sus vecinos para que le de seguimiento a un paquete en su camino a un último destino.
- Objetivo: generar (para cada proceso) un procedimiento de toma de decisiones para realizar esta función y garantizar la entrega de cada paquete

Eficiencia algoritmos ruteo

- Existen diferentes parámetros para medir un algoritmo de ruteo:
  - número de saltos
    - » costo de usar una ruta es medido como el numero de saltos de la ruta
  - ruta mínima
    - » a cada canal se le asigna un peso no negativo
    - » costo ruta es la suma de todos los canales
  - retardo mínimo
    - » cada canal se le asigna un peso que depende del tráfico en el canal

Criterios de definición de algoritmos de ruteo

- Correctness
  - el algoritmo debe entregar cada paquete a su último destino
- Complejidad
  - el algoritmo debe utilizar el menor número de mensajes posible
- Eficiencia
  - el algoritmo debe enviar los paquetes a través de buenas rutas
  - rutas que ofrecen un retardo muy pequeño
  - se dice que un algoritmo es óptimo si usa las mejores rutas

Más criterios de ruteo

- Resistencia
  - en caso de un cambio de topología (adición o borrado de un canal o nodo) el algoritmo actualiza las tablas para realizar la función de ruteo en la red modificada
- Adaptabilidad
  - algoritmo balancea las cargas de los canales y nodos adaptando las tablas, para evitar rutas a través de canales o nodos muy saturados
- Equidad
  - algoritmo debe proporcionar el mismo servicio a cada usuario con en el mismo grado

Algoritmos de ruteo

- Algoritmo de Chandy y Misra
- Algoritmo de Toueg
- Algoritmo de tablas compactas
- Algoritmo por intervalos

Algoritmo Chandy Misra

- Considera un grafo G = (V,E), donde V es el conjunto de vértices (procesos) y E el de aristas
- Calcula la distancia mínima entre un vértice (V1) y el resto de los vértices del grafo
- Son aristas dirigidas
- Cada arista tiene asociado un costo.
- Un proceso conoce el costo del canal hacia sus vecinos.

Algoritmo Chandy Misra

- El costo puede ser negativo, por lo que es posible que exista un ciclo de longitud total negativa, (llamado ciclo negativo)
- Solo se tratará el caso de aristas con costos positivos
- Objetivo
  - que cada nodo conosca el costo mínimo para llegar a un nodo V_1

Variables de los procesos

- w_{jk} entero;
  - costo arista entre v_j y v_k
- v_1, v_i id_proc;
  - identificadores proceso inicial y actual
- pred id_proc;
  - identificador predecesor proceso
- d entero;
  - longitud más corta entre v_1 y v_i (inicializado en infinito)
- num entero;
  - número mensajes que se han enviado y no se ha recibido un acuse de recibo

Tipos de mensajes

- Mensaje 1: [s,x]
  - s es la distancia entre v_1 y v_i,
  - *x* es el penúltimo vértice de la cadena de procesos entre v_1 y v_i.
- Mensaje 2: [ack]
  - mensaje de acuse de recibo
  - proceso p_j envía un ack a p_i en respuesta a un mensaje [s,x]

El código

- Inicialización y recepción mensaje [s,x] en el proceso p_1
- Recepción acuses de recibo en p_1
- Inicialización de todos los procesos, salvo el proceso p_1
- Recepción mensaje en proceso p_j

Inicialización y recepción mensaje [s,x] en el proceso p_1

inicialización

d:=0;

pred:= indefinido;

enviar [w_{1k}, p_1] a (todos los sucesores de p_1)

fin-inicialización

recepción_mensaje [s,p_i]

si (s < 0)

<ciclo negativo>

sino

enviar [ack] a p_i

fin-recepción_mensaje

Recepción acuses de recibo en p_1

```
recepción_mensaje [ack]
        num:=num-1;
        si  (num = 0) entonces
                <ciclo negativo>
        fsi
fin-recepción_mensaje
```

Recepción acuses de recibo en resto procesos

```
recepción_mensaje [ack] del proceso p_k
        num:=num - 1;
        si (num = 0) entonces
                enviar [ack] a pred;
        fsi
fin-recepcion_mensaje
```

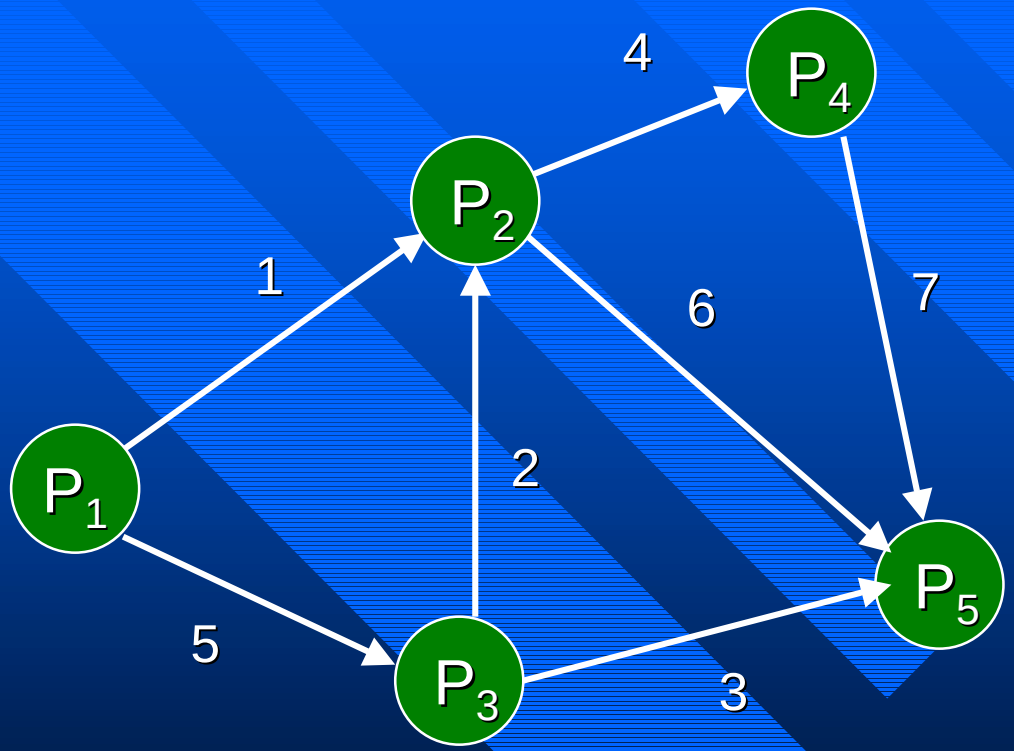
Inicialización de todos los procesos, salvo el proceso p_1

```
inicialización_procesos  pj  (j≠1)
        /* No se ha recibido ningún mensaje,
                no hay mensajes de ack */

        d = ∞;
        pred = indefinido;
        num=0;
fin inicialización_procesos
```

Recepción mensaje en proceso p_j

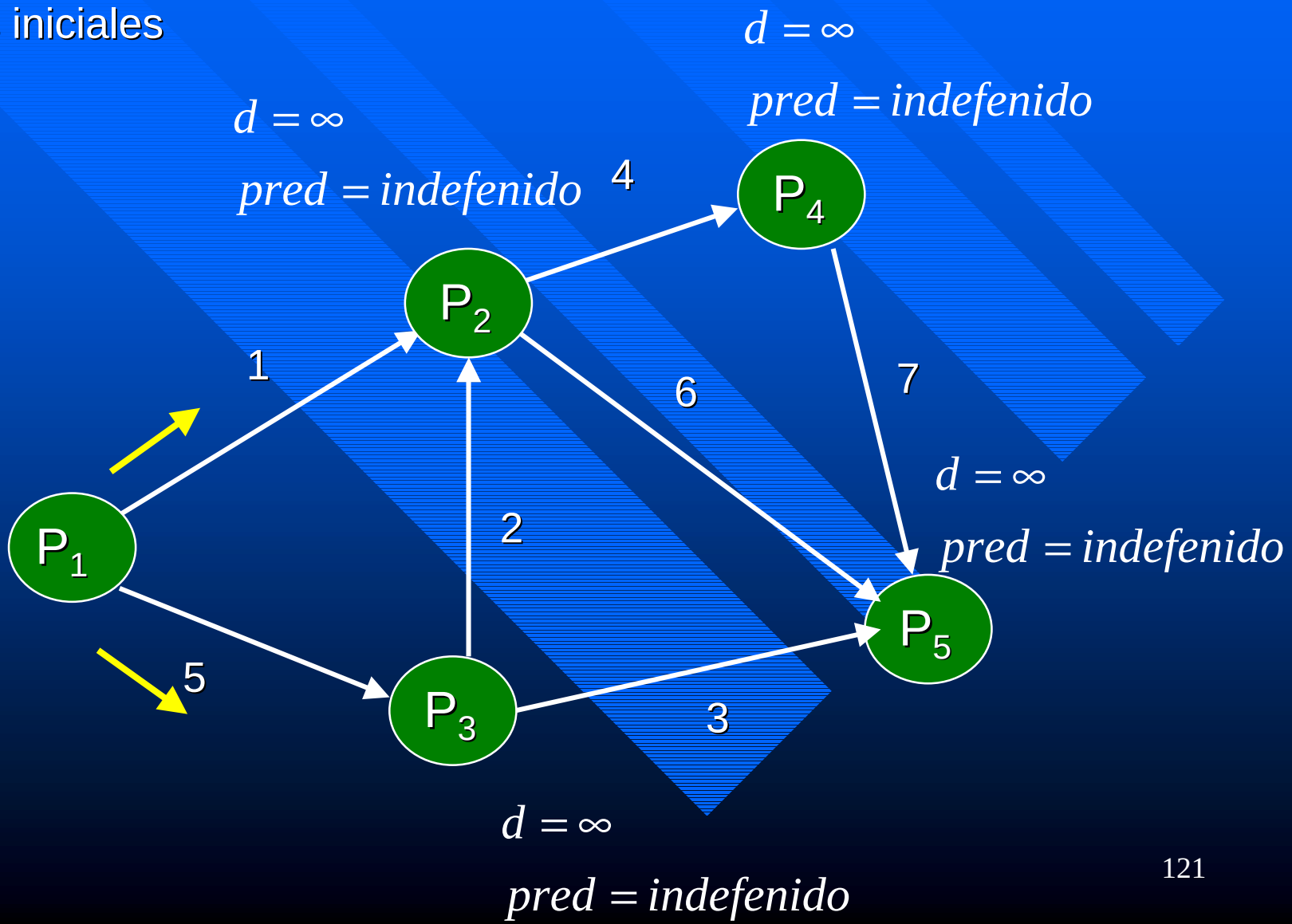
recepción_mensaje [s,p_i]
    si (s < d) entonces:
        si (num > 0) entonces
            enviar [ack] a pred;
        pred:=p_i;
        d:=s;
        enviar [d+w_{jk}, p_j] a (todos sucesores de p_j)
        num:=num+ (numero sucesores de p_j)
        si (num = 0) entonces
            enviar [ack] a pred;
    sino */* mensaje no representa una mejoría */*
        enviar [ack] a pi;
fin-recepcion_mensaje

Ejemplo ejecución

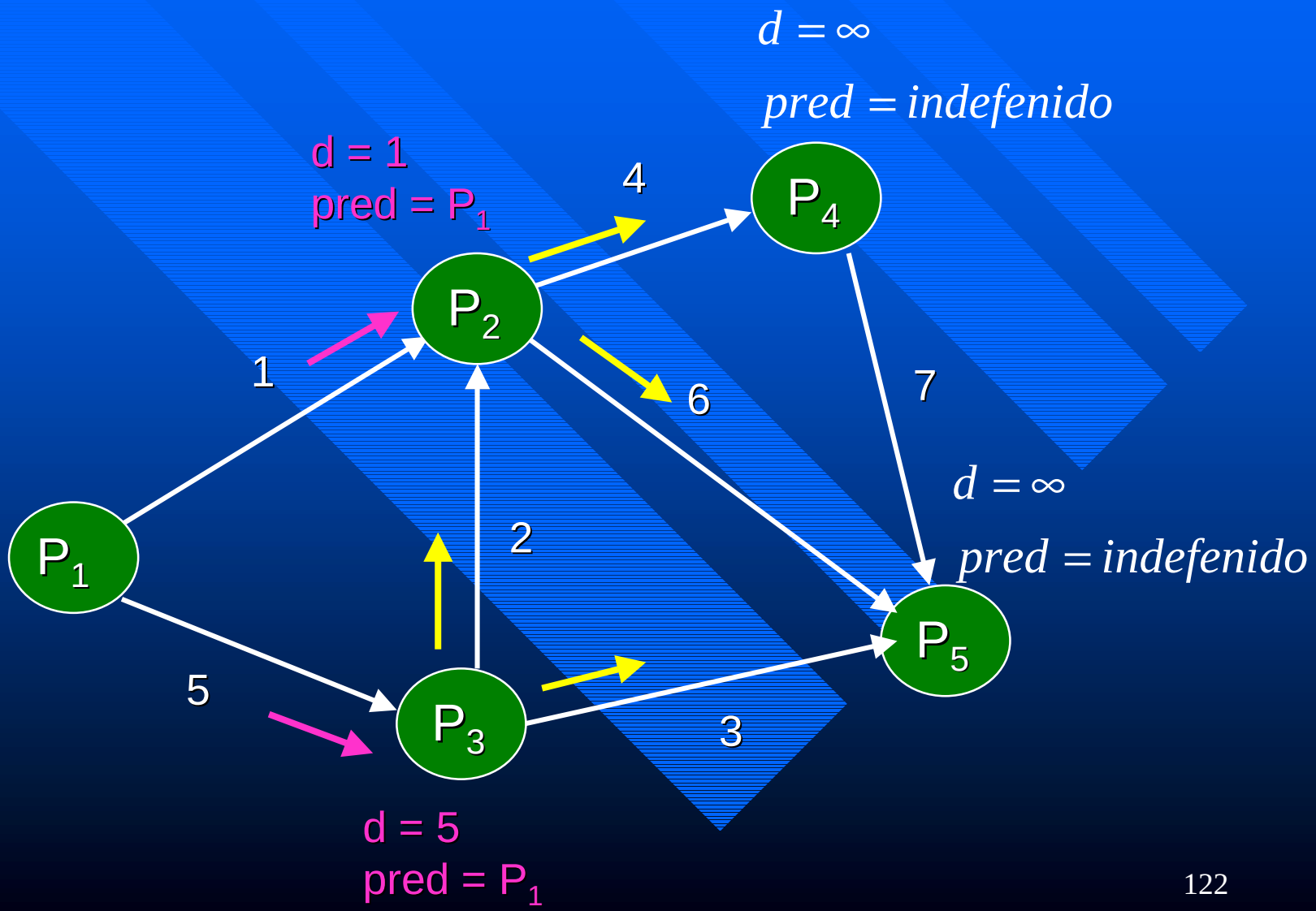


Ejemplo ejecución

Valores iniciales

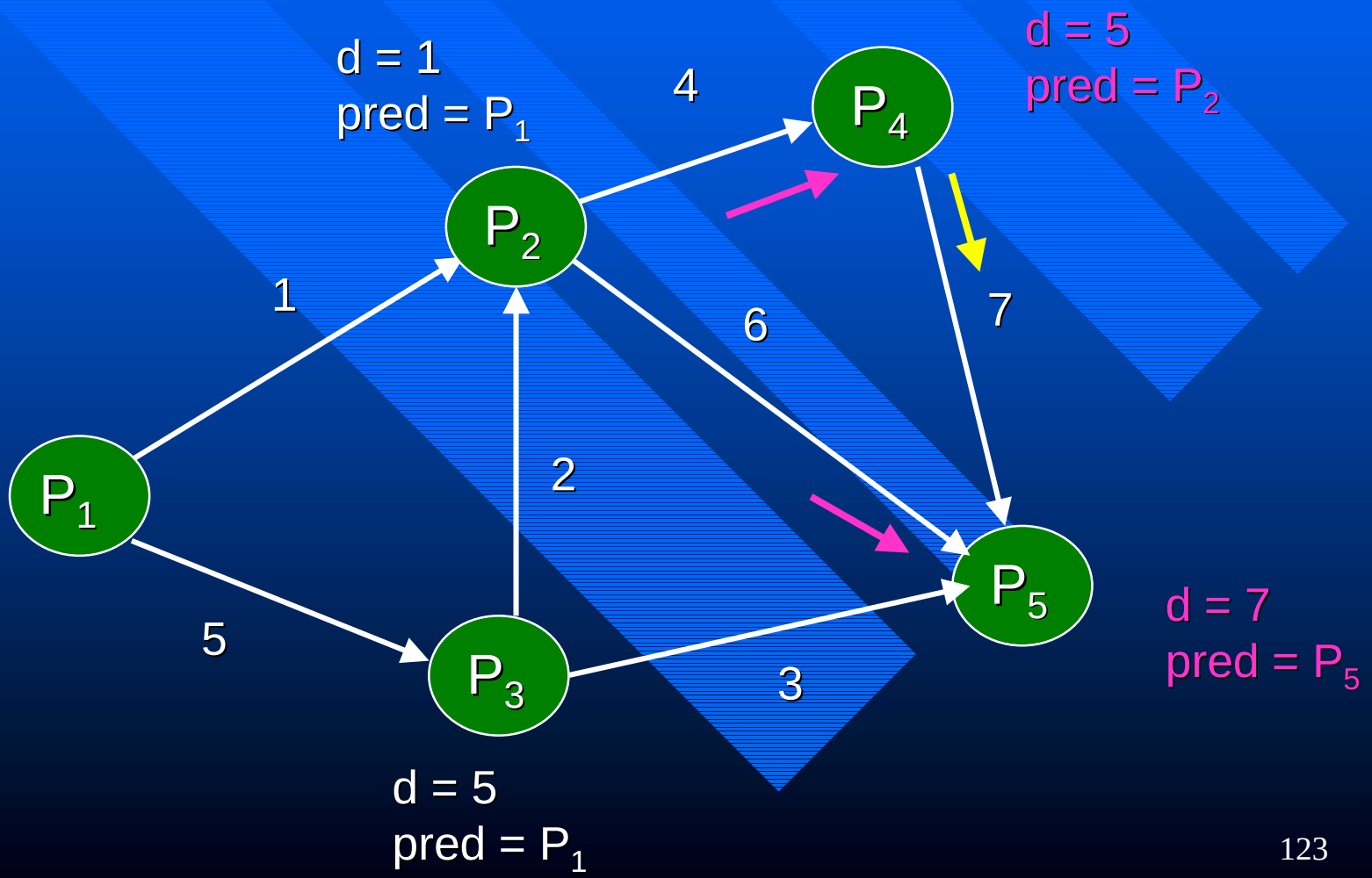


Ejemplo ejecución



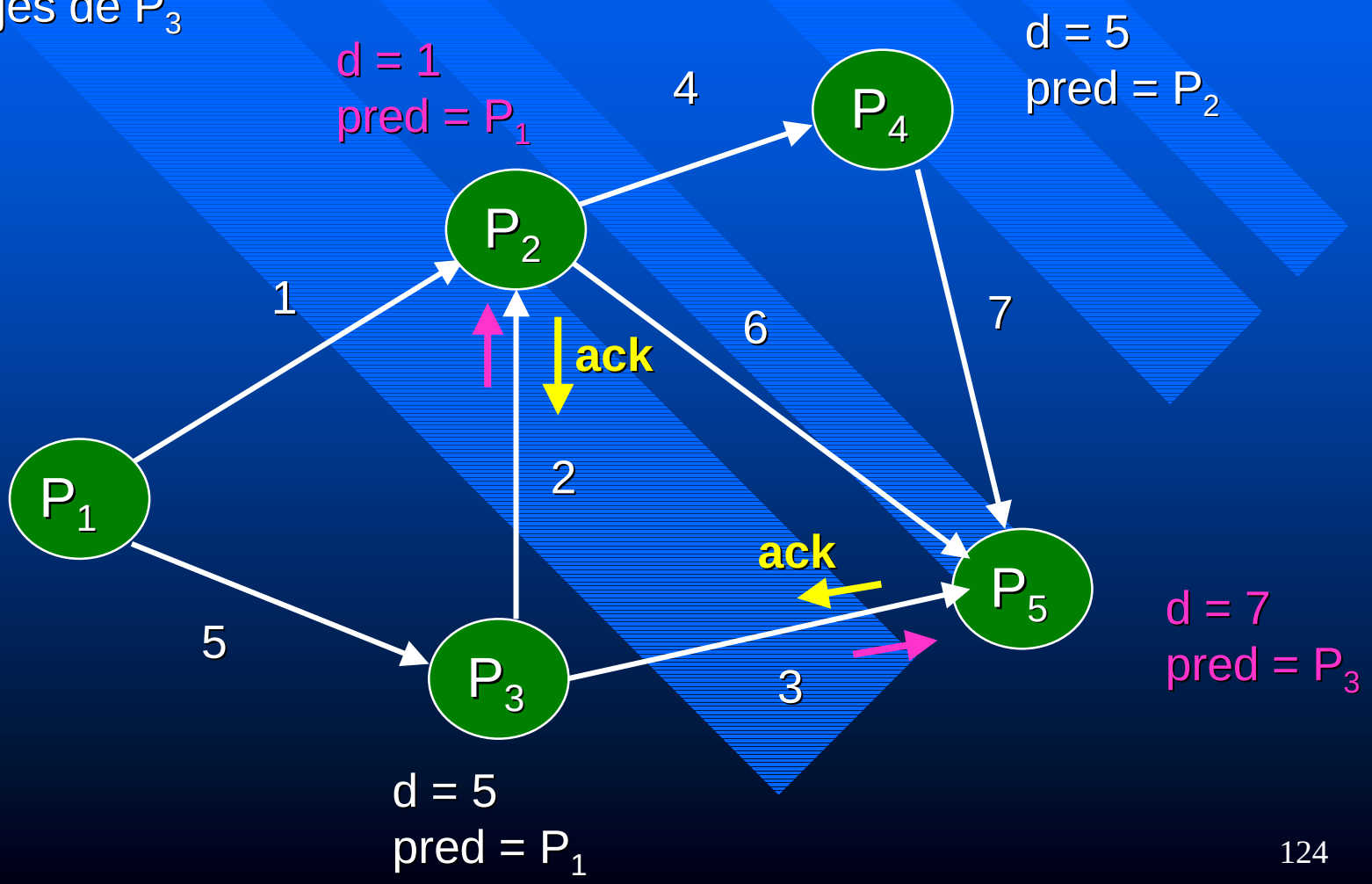
Ejemplo ejecución

Primero llegan mensajes de P_2



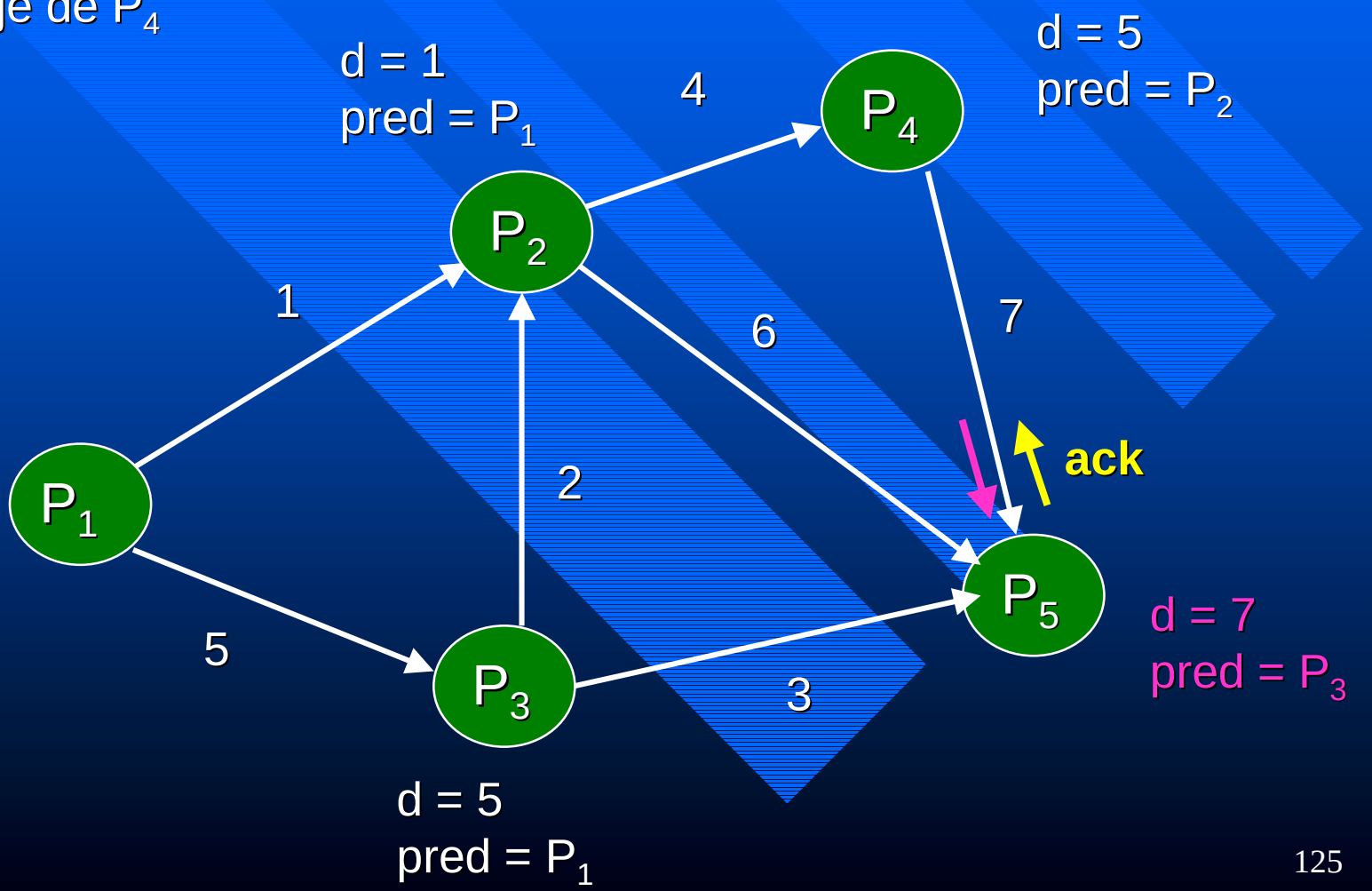
Ejemplo ejecución

En segundo lugar llegan mensajes de P_3



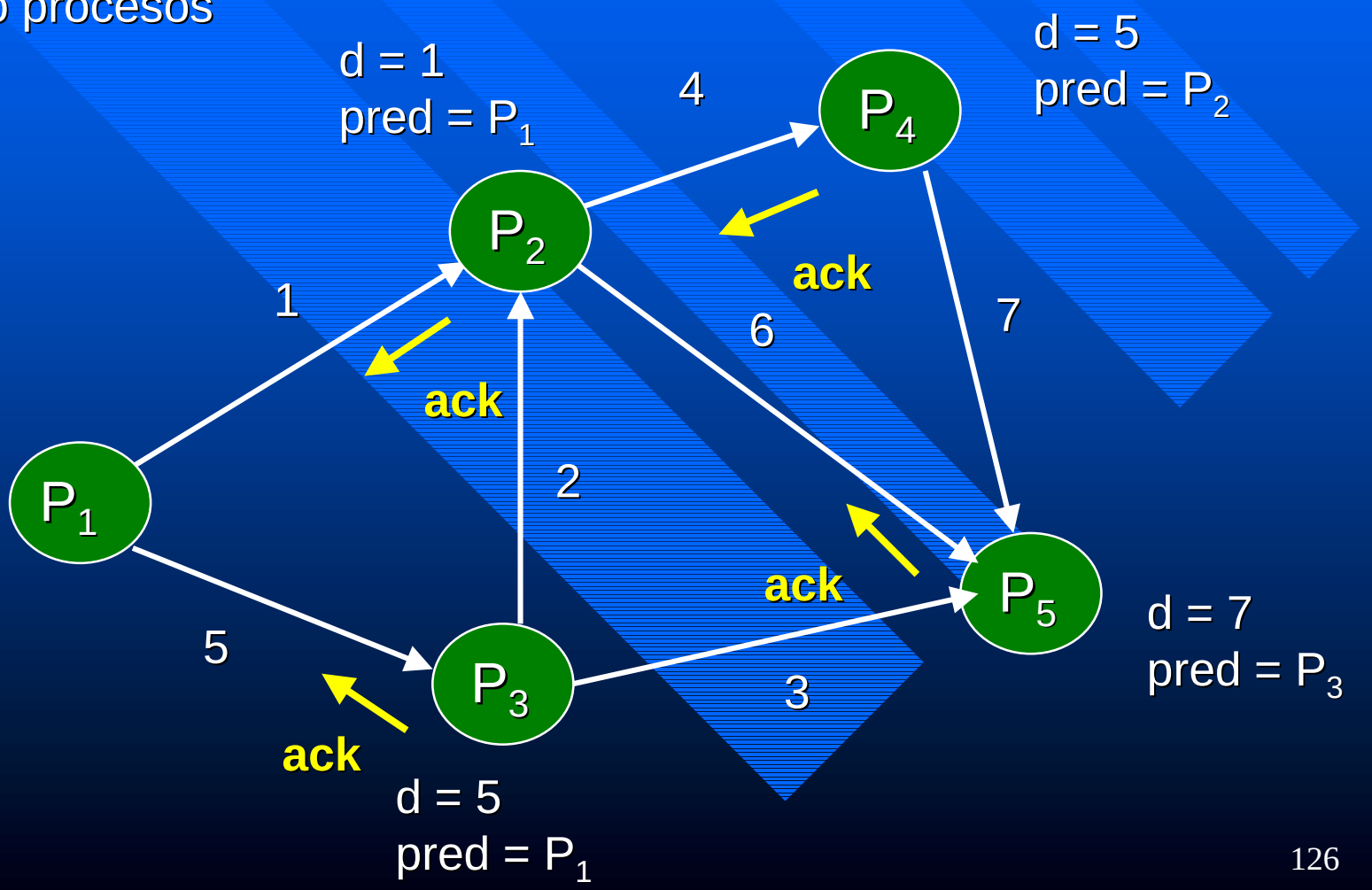
Ejemplo ejecución

En tercer lugar llega mensaje de P_4



Ejemplo ejecución

Se envian los ack
al resto procesos

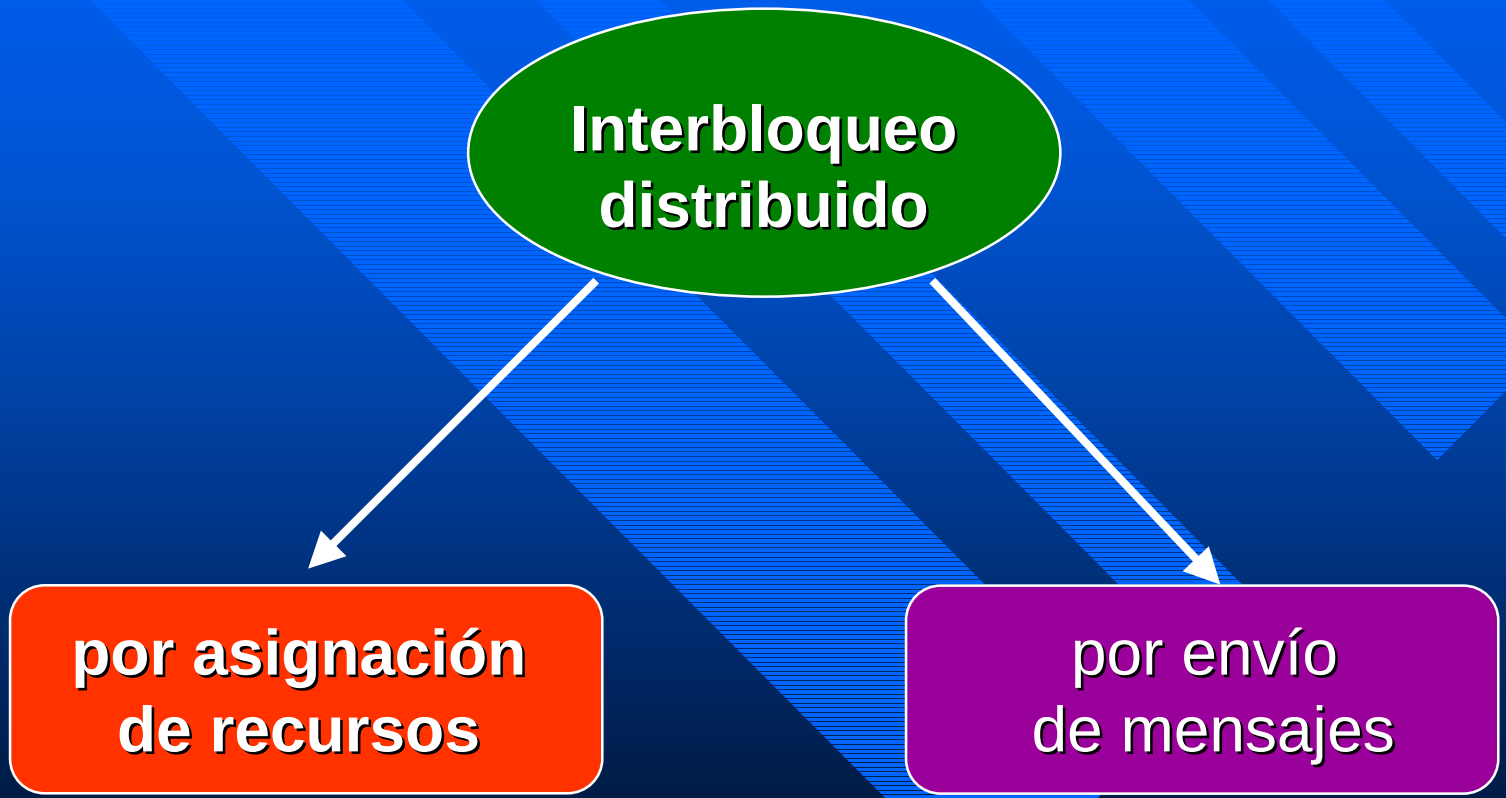


El interbloqueo distribuido

- Interbloqueo:

  Situación en la que se encuentran un conjunto de procesos, (al menos dos), tal que cada proceso del conjunto espera la ocurrencia de un evento que solo se puede ser provocado por otro proceso del mismo conjunto

Tipos interbloqueo



Interbloqueo por asignación recursos

- Conjunto procesos que se comunican por mensajes
- Existe un cierto número de controladores que administran un conjunto de recursos
- Intéres: asignación de recursos a los procesos
- Recursos:
  - acceso exclusivo
  - ejemplares únicos

El concepto de dependencia

- Si P_i requiere recurso *r* lo debe solicitar al controlador:
  - *si esta libre:* se le asigna
  - *si no* P_i se bloquea esperando que el proceso P_j, que utiliza dicho recurso, lo libere
- Entonces se dice que P_i es dependiente de P_j
- En general P_i es dependiente de P_k si existe una secuencia de procesos P_i, P_j, …., P_k en la cual cada proceso, (a excepción de P_i), posee un recurso por el cual su predecesor en la secuencia esta esperando.

Propuesta interbloqueo por asignación de recursos

- Basadas en circuitos de grafos de espera
- Dos grandes clases:
- Técnicas a priori
  - impiden que un circuito aparezca
- Técnicas a posteriori
  - se detecta una vez que el circuito apareció

Técnicas a priori

- Dos variantes
- 1a. variante
  - definir un orden total en la asignación de recursos
  - el recurso R_i solo puede ser solicitado por P_k si este ha obtenido todos los recursos R_j que necesita: ($j < i$)
- 2da. variante
  - los procesos deben hacer anticipadamente, la solicitud de sus recursos.
  - una asignación solo es autorizada si esta no provoca un circuito en la gráfica común

Tecnicas a posteriori

- Consisten en dejar que los procesos soliciten recursos.
- Verificar a posteriori si no hay un circuito en la gráfica de espera.
- Si se detecta un circuito detectado se escoge un proceso y se le obligaa devolver los recursos
- El proceso víctima es devuelto al estado en el que se encontraba antes de pedir los recursos

Falso interbloqueo

- Sea la secuencia de procesos:
  - $P_1, P_2, ..., P_n$
- P_i bloqueado por un recurso poseido por P_{i+1}
- P_n activo:
  - libera el recurso que P_{n-1} esperaba
  - emite 1er. mensaje de liberación de recurso
  - emite 2do. mensaje pidiendo recurso que posee P_1
- El segundo mensaje es visto por procesos detectores de interbloqueo antes que el primer mensaje:

se detecta un circuito ====> ¿INTERBLOQUEO?

Solución

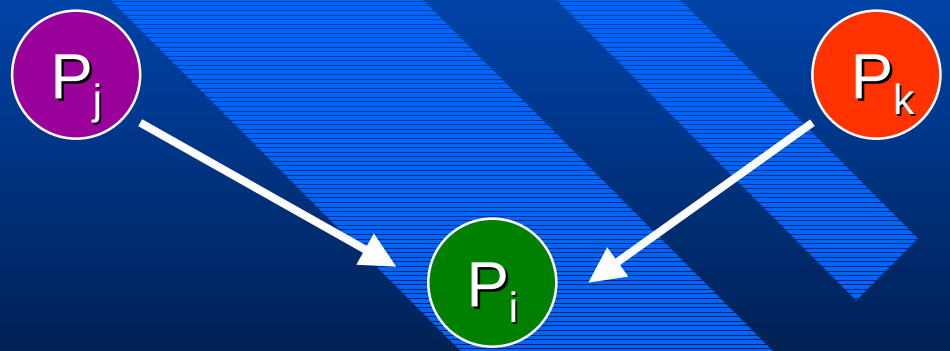
- Adaptar un algoritmo centralizado
- Duplicar función de prevención/detección de interbloqueo en todos los procesos que componen el sistema
- Principio algoritmos de Lomet
- Utilizan técnicas a priori
- Se trata de una técnica de prevención
- Algoritmos desarrollados bajo el esquema de las bases de datos
- Dos tipos de algoritmos:
  1. duplican edo. global en todos los sitios
  2. mantiene sobre cada sitio una parte del edo. global

Interbloqueo por mensajes

- Cada proceso espera un mensaje de otro proceso siendo que ningún mensaje esta en tránsito
- Interbloqueo conjunto S de procesos comunicantes se puede caracterizar de la forma siguiente:
  1. Todos los procesos de S estan bloqueados, (esperando mensajes)
  2. Sea cual sea el proceso de S, su *conjunto de dependencia* esta incluido en S
  3. No hay mensajes en tránsito entre los procesos de S

Conjunto de dependencia

- A cada proceso pasivo P_i (en espera de mensajes) se le asocia un CD (Conjunto dependencia)
- CD de P_i contiene el o los procesos en provenencia de los cuales P_i espera un mensaje
- Ejemplo:



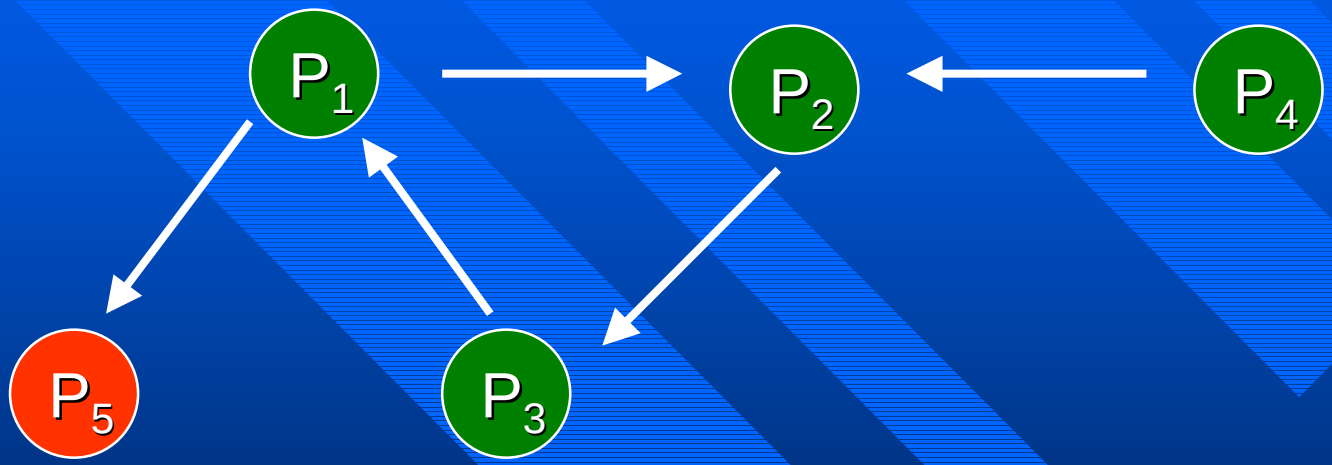
P_i espera un mensaje proveniente de P_j o P_k
P_i depende de los proceso P_j y P_k
Emisión de un mensaje de P_j hacia P_i y su recepción,
tendra por efecto suprimir todos los arcos que parten de P_i

Interbloqueo mensajes vs interbloqueo recursos

- Conocimiento del proceso que bloquea
  - en mensajes, si Proceso A necesita recibir mensaje de B, A puede saber que esta esperando por B
  - en recursos, la dependencia de una transacción en las acciones de otras transacciones no es directamente conocida,
- El desbloqueo
  - en mensajes, un proceso no puede continuar su ejecución hasta que puede comunicarse con al menos uno de los procesos por los cuales esta esperando
  - en recursos un proceso no puede continuar su ejecución a menos que haya recibido todos los recursos que espera

Ejemplo de no interbloqueo

- Un circuito en la gráfica de espera no es condición suficiente para detectar un interbloqueo



P_1 espera un mensaje de P_5 o de P_2

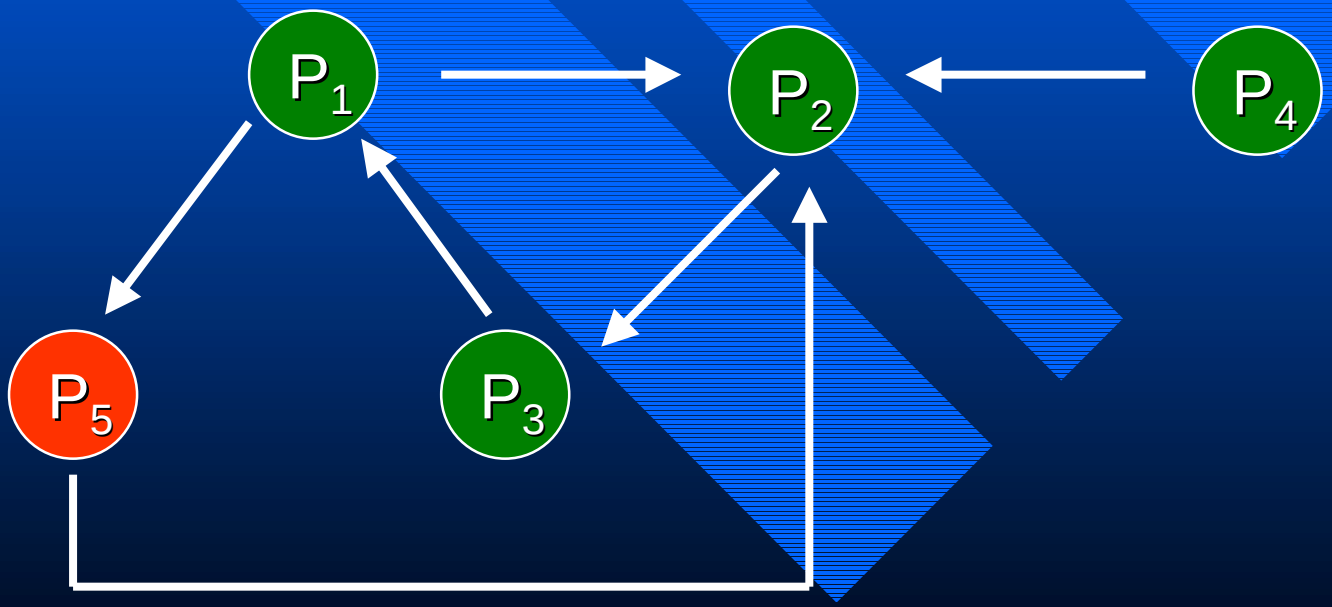
P_5, que no esta bloqueado, envía un mensaje a P_1

P_1 recibe el mensaje y se desbloquea

===>> los arcos (P_1, P_5) y (P_1, P_2) desaparecen

Ejemplo interbloqueo

- Para que se de un interbloqueo:
  - *P_5 tiene que estar esperando mensajes provenientes de procesos en espera de otros*



Previniendo interbloqueo de comunicaciones

- Noción conjunto dependencia CD de P_i conjunto de procesos de los cuales P_i espera un mensaje
- CD de P_i esta vacío si P_i esta activo
- CD puede variar durante la ejecución, los procesos pueden contener una primitiva de espera de mensaje en diferentes lugares de su código
- Interbloqueo conjunto S:
  - Sea P_i un elemento de S
  - P_i esta bloqueado esperando por un mensaje y
    - » todos los elementos de su CD de P_i pertenecen a S
    - » además no hay mensajes en tránsito dirigidos a elementos de S
- Ejemplo: algoritmo de Chandy, Misra y Haas

Algoritmos detección terminación

- Determinar la terminación de un algoritmo distribuido es un problema importante y no trivial
- No es condición suficiente que todos los procesos que intervienen en un algoritmo distribuido sean observados en un estado pasivo
- Mensaje emitido por alguien y que todavía no llega a su destino volverá activo al proceso destino una vez que llegue

Estados procesos

- Diferente del concepto de estado proceso en un sistema operativo.
- Cada vez que un proceso es ejecutado pasa por diferentes estados.
- Estos estados se clasifican en:
  - estado activo
  - estado pasivo
  - estado completo
  - estado terminado

Tipos estados procesos

- Proceso Activo:
  - cuando un proceso ejecuta el texto de su programa
- Proceso Pasivo:
  - en los otros casos; puede ser terminado en espera de mensajes en provenencia de otros procesos.
- Proceso Completo
  - no hay ningún mensaje a enviar y todo su trabajo local fue terminado
  - no se puede modificar el valor de una variable después de la recepción de un mensaje durante el resto de la ejecución del algoritmo
  - puede ser activado por la recepción de un mensaje.

Tipos estados procesos

- Proceso Terminado
  - proceso en estado completo y que no puede ser activado por otro proceso.

Terminación vs interbloqueo

- Terminación: detectar que los procesos participantes se encuentran situación de interbloqueo y no podran volverse activos.
- En este caso interbloqueo y terminación son un solo y mismo problema.
- ¿Qué pasa en el caso general?
- ¿Qué hay de común entre el interbloqueo y la terminación?

Diferencías entre terminación e interbloqueo

- Existen tres diferencias:
  - el objetivo
  - la detección
  - la forma en que los procesos se vuelven pasivos
- El interbloqueo y la terminación son dos problemas vecinos pero diferentes
  - hay que tener cuidado para no confundirse

Principio general de la soluciones

- Detección de una propiedad global, la terminación
- Todas las soluciones comienzan por estructurar el conjunto de procesos de tal forma que se puea realizar una travesía para verificar la propiedad
  - anillo unidireccional
  - árbol de recubrimiento
  - circuito precalculado en la gráfica de procesos
- Determinar el modo de "travesía" de forma tal que no se detecte una falsa terminación.
- Recorrer esta estructura para verificar la propiedad de terminación

Calculando la terminación de la difusión de cálculo

- Concepto difusión cálculo introducido por Dijkstra y Scholten
  - Dijkstra y Scholten, Termination Detection for Diffusing Computations, Inf. Proc. Letters, Vol. 11, 1 (agosto 1980), pp. 1-4
- Conjunto de procesos y vías de comunicación definen un grafo dirigido
- Se elige un proceso que no tenga predecesor y se le denomina iniciador

La difusión de cálculo

- Al principio todos los procesos están pasivos.
- Cuando un proceso recibe un mensaje se vuelve activo y puede emitir mensajes a sus predecedores
- Hipotesis adicionales
  - un proceso solo puede emitir un número finito de mensajes
  - los retardos de transmisión de los mensajes son arbitrarios pero finitos

El problema

- Dijkstra y Scholten proponen el siguiente problema:

  Diseñar un control de la terminación adaptable a dicho esquema de comunicación y cálculo (la difusión de cálculo) de tal forma que cuando termine, el proceso iniciador sea informado, y que este último no perciba una falsa terminación.

Principio de la solución

- Asociar a todo mensaje emitido de Pi a Pj un mensaje o señal de Pj hacia Pi
- Mensaje constituyen un tipo especial de reconocimiento y sus emisiones estarán sujetas a una regla relacionada con la propiedad de terminación.
  - una vez que el proceso iniciador reciba todas las señales relativas a los mensajes que emitió, este puede deducir que el cálculo terminó

La nción de deficit

- Asociado a cada conexión (unidireccional) entre dos procesos.
- Definido como el número de mensajes recibidos y el número de señales enviadas sobre esta vía de comunicación.
- Cada proceso Pi se tienen definidas dos varibales:
  - defin: la suma de deficits de los arcos que entran al proceso
  - defout: la suma de deficits de los arcos que salen del proceso
  - inicialmente tienen un valor de cero

Cálculo del deficit

- Para todo proceso se cumple que:
  - envío mensaje: defout = defout + 1;
  - envío señal: defin = defin – 1;
  - recepción de un mensaje: defin = defin + 1;
  - recepción de una señal: defout = defout – 1;
- A excepción del proceso iniciador, el resto solo pueden emitir mensajes si recibieron uno y si no han regresado a su estado inicial
  - caracterisado por defin = defout = 0
- El envío mensaje solo es posible si no se invalida el predicado
  - defin = 0 => defout = 0

Examinando la condición

- La condición: defin = 0 => defout = 0
  - indica que un proceso no puede estar en un estado activo (y por lo tanto no puede emitir mensajes) si no ha recibido un mensaje
  - el envio de una seña no debe invalidar dicho predicado
    - » esto asegura que el envio de la ultima señal de parte de Pi (defin = 0) este proceso habra recibido todas las señales relativas a los mensajes emitidos (defout =0)

Envio de una señal

- Los deficits son contadores positivos o nulos.
- El envio de una señal por parte de Pi decrementa la variable defin y la condición anterior debe ser verificad siempre.
- El envio de una señal esta sujeta a la condición siguiente:

$$(defin - 1 \geq 0) \text{ y } (defin - 1 = 0 \Rightarrow defout = 0)$$

lo que se simplifica en :

$$(defin > 1) \text{ o } ((defin = 1) \text{ y } (defout = 0))$$

Variables del algoritmo

- Los deficits
  - defin, defout: 0 … nbmax inicializados en 0
- El proceso padre en el grafo de los procesos:
  - padre: 1 .. n;
- Conjunto donde se almacenan los identificadores de otros procesos que le han enviado mensajes
  - otros: conjunto de 1 .. n;

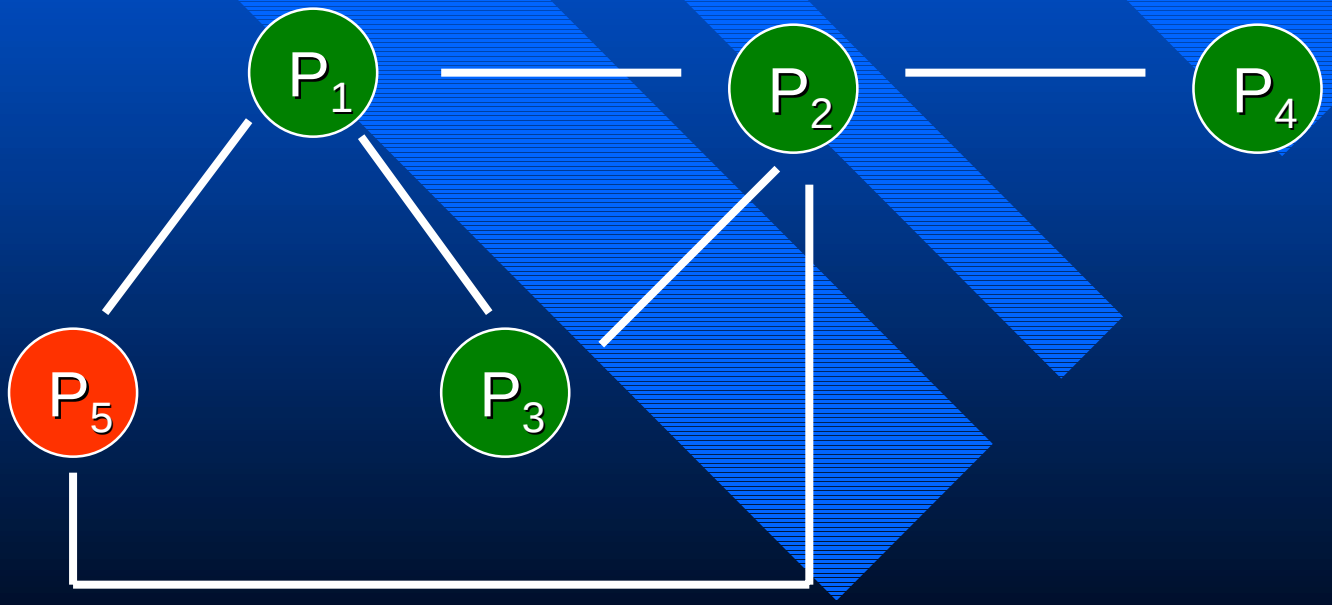
El algoritmo (recepción mensajes)

```
recepcion de [mensaje, expd] de expd
        si (defin = 0) entonces
                padre := expd
        sino
                otros := otros + expd
        defin := defin + 1;

recepcion de [signal, expd] de expd
        defout := defout – 1;
```

El algoritmo (envío mensajes)

```
envio de [mensaje, i ] hacia j
        si ( defin ≠ 0 ) entonces
                defout = defout + 1;
                enviar [mensaje, i ] a j;
        sino
                no es posible enviar mensaje


envio de [signal]
        si ( defin > 1) o ( (defin = 1) y (defout = 0))
                si defin = 1 entonces
                        enviar [signal, i ] a padre;
                sino
                        aux:= un_elemento (otros);
                        otros = otros – aux;
                        enviar [signal, i] a aux;
                defin := defin – 1;
```

Ejemplo ejecución



Un par de metodologías diferentes

- La ola recursiva
- La autoestabilización

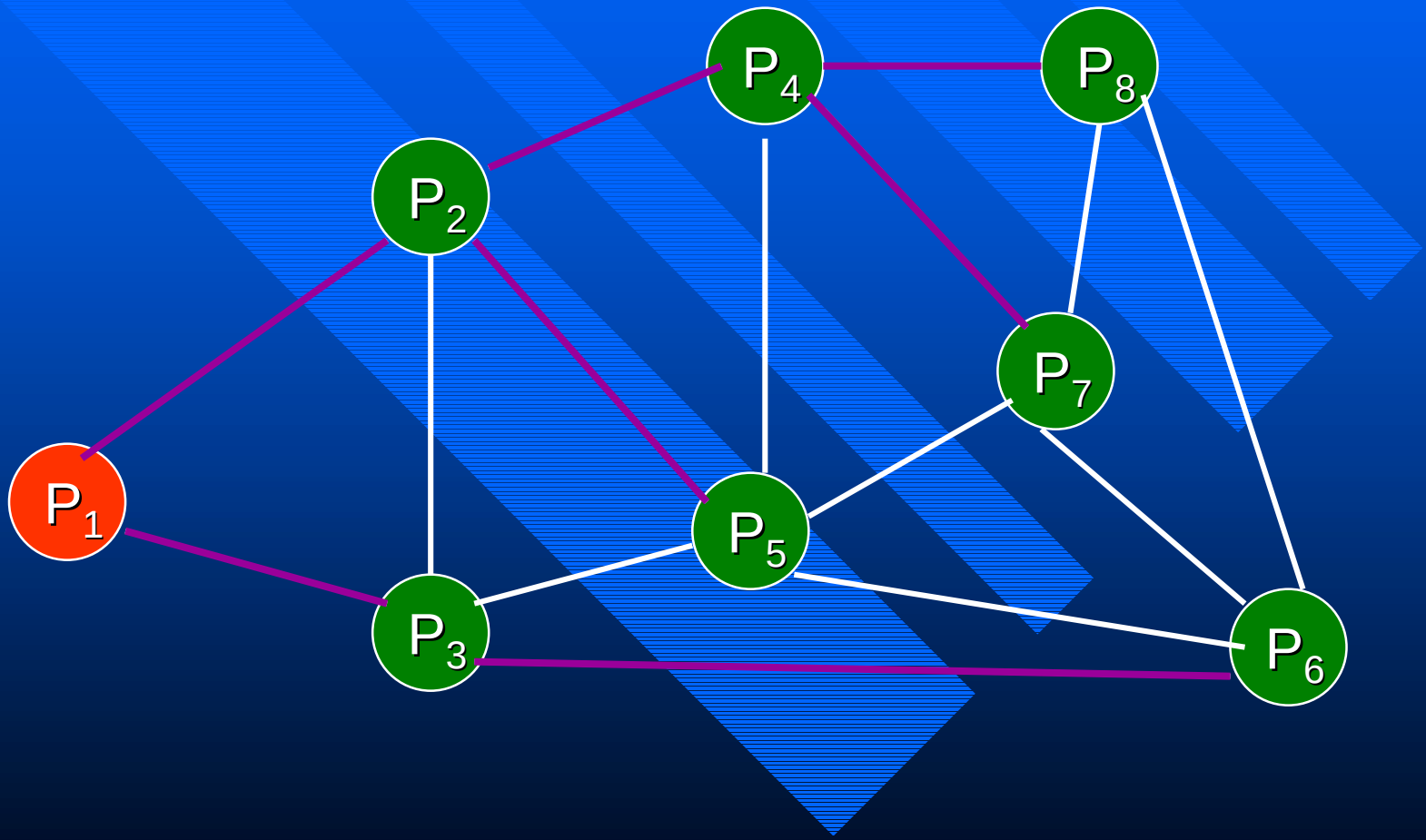
La Ola Recursiva

- Combinación de:
  1. modo de comunicación multipunto
     - ✓ difusión sobre un grupo
     - ✓ RPC sobre un grupo
  2. del principio de recursión
- Algoritmos tipo ola
- Definición de una *ola recursiva*

Características ola recursiva

- Estructura de *control distribuida*
- Procedimiento distante
- llamada *paralela* de n (o ninguna) ejecución de él mismo
- Ejecución induce un árbol
  - raíz asociada a la primera ejecución del procedimiento
  - raíz y nodos asociados a procedimientos en espera de que terminen las llamadas que realizaron
  - las hojas estan asociadas a: procedimientos activos ejecución final sin llamada recursiva

El árbol de la Ola Distribuida



Las primitivas utilizadas

- La instrucción *par*

  par i **in** <dominio> **do**

  <Bloque de instrucciones>

  enddo

- Sintaxis del RPC

  <nom_proc> **(**<lista parámetros>**) on** <ident_procs>

Tipos de ola recursiva

1. Ola recursiva secuencial
   - cada llamada genera solamente una sola llamada recursiva
   - la ola se propaga sobre una lista o anillo
2. Ola recursiva sobre un árbol
   - reposa sobre una topología de árbol definida anteriormente
   - cada nodo conoce a su padre e hijos
   - la raíz representa al proceso iniciador
   - hojas representan procesos encargados de detener la ola
3. Ola recursiva inundante
   - construcción dinámica de la estructura de control
   - uso de una estrategia de inundación

Esquema general de la ola

```
type id_processus is .....;
procedure ola_recursiva (<parámetros>) is
   <Declaraciones>;
   i: id_processus;
   procs_group: setof id_processus;
   begin
        <Bloque A de instrucciones>
        if (condición) then
                < Bloque B de instrucciones>
                par i in procs_group do
                        < Bloque C de instrucciones>
                        ola_recursiva_( f(<parámetros>) )on i;
                        < Bloque D de instrucciones>
                enddo;
                < Bloque E de instrucciones>
        endif;
        < Bloque F de instrucciones>
   end ola_recursiva;
```

Ejemplo ola recursiva

```
procedure difunde ( msj ) is
  hijos: setof id_processus;
  info: <información a recibir/enviar>;
   i: id_processus;
begin
  info:=trata_info(msj);
  if not empty(hijos) then
      par i in hijos do
            difunde( info ) on i;
      enddo;
  endif;
end difunde;
```

La autoestabilización

- En 1973, Dijkstra utilizó el término “autoestabilizante” para distinguir cualquier sistema que tiene la siguiente propiedad:
  - si el sistema empieza en cualquier, posiblemente estado ilegítimo, está garantizado a converger a un estado legítimo en un tiempo finito

Propiedades algoritmos autoestabilizantes

- Topologia Dinámica
  - un algoritmo autoestabilizante que realiza un cálculo dependiente de la topología (tablas de ruteo, arboles de expansión), converge a una nueva solución si se presenta un cambio en la topología

Desventajas

- Inconsistencia inicial
  - antes de llegar a una configuración legitima, el algoritmo puede mostrar salidas inconsistentes
- Complejidad alta
  - son más complejos que su contraparte clásica para el mismo problema
- No detección de la estabilización
  - es muy dificil observar dentro del sistema, cuando se ha alcanzado una configuración legitima

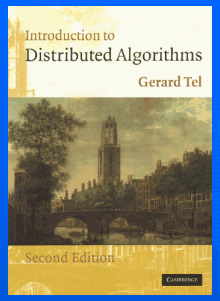
Ejemplos algoritmos autoestabilizantes

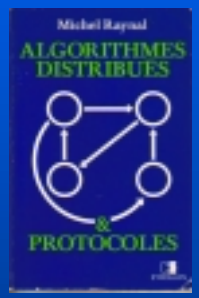
- Algoritmo Token Ring de Dijkstra
- Algoritmos de grafos
  - orientación del anillo
  - maximal matching
  - elección y arbol de expansión
- Calculo de rutas mínimas

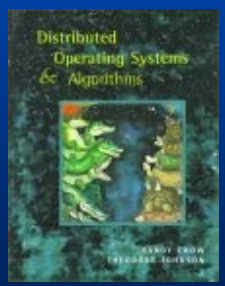
Propiedades algoritmos autoestabilizantes

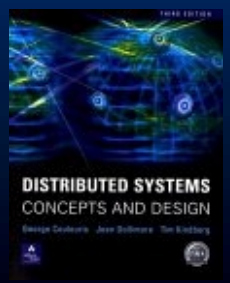
- Tolerancia a fallas
  - ofrecen completa y automatica protección contra todas las fallas transitoria de los procesos
  - ya que los algoritmos se recuperan de cualquier configuración, no importa cuanto se han corrompido los datos
- Inicialización
  - no hay necesidad de una inicialización consistente
  - los proceso pueden empezar cualquier estado arbitrario y garantizar un comportamiento determinado
- Topologia Dinámica

Referencias (libros)



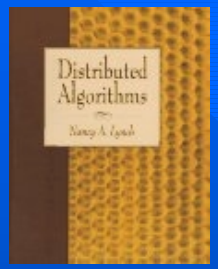


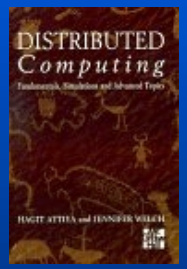


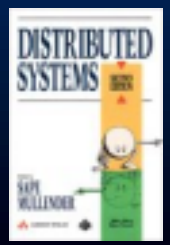


- Introduction to Distributed Algorithms, Gerard Tel, Cambridge University Press , 1994
- Algorithmes Distribues & Protocoles, Michel Raynal, Ed. Eyrolles, 1985
- Distributed Operating Systems & Algorithms, Randy Chow y Theodore Johnson, Ed. Addison Wesley, 1997
- Distributed Systems, Concepts and Design , George Coulouris, Jean Dollimore, Tim Kindberg, Segunda Edición, Ed. Addison Wesley, 1994

Referencias libros

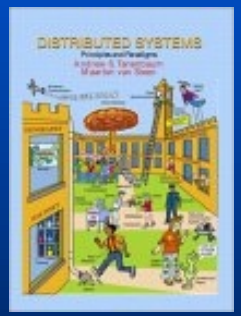






- Distributed Algorithms, Nancy A. Lynch, 1996, Morgan Kaufmann Publishers
- Distributed Computing, Hagit Attiya and Jeeniifer Welch, Mac Graw Hill, 1998
- An introduction to distributed algorithms, Barbosa, V.C.,MIT Press, 1996,
- Algorithmique Paralléle et Distribuée, Ivan Lavallée, Primera Edición, París 1995, Ed. Hermes
- Distributed Systems, S. Mullender, Ed. Addison Wesley

y otros libros más



- Evaluation des algorithmes distribués, Christian Lavault, Ed. Hermes, 1995
- Algorithmique du parallélisme, Michel Raynal, Ed. Dunod, 1984
  - Algorithms for Mutual Exclusion, MIT Press, Cambridge, Massachusetts, 1986
- Synchronisation et controle des systems et des programmes repartis, M. Raynal, J.M. Helary, 1988
- Distributed Systems, A. Tanenbaum, Prentice Hall, 2002

Algunas sitios interesantes

Distributed Algorithms & Systems

- Distributed Algorithms and Systems
  - http://www.cs.chalmers.se/~tsigas/DISAS
- MIT Theory of Distributed Systems Group
  - http://theory.lcs.mit.edu/tds/
- Laboratorio de Investigación Avanzada de la Univ. Paris 8
  - http://lria.univ-paris8.fr/
- Centro de Estudios y de Investigación en Informática (CEDRI-CNAM)
  - http://cedric.cnam.fr/

Cedric
Centre De Recherche en Informatique du Cnam

Los principales congresos

- PODC (2002 – 21)
  - ACM Symposium on Principles of Distributed Computing
- OPODIS (2002 – 6)
  - International Conference On Principles Of DIstributed Systems

- DISC (antes WDAGS, 2002 - 16)
  - Symposium on DIStributed Computing

- ICDCS (2002 – 21)
  - International Conference on Distributed Computing Systems
- SIROCCO (2002 – 9)
  - International Colloquium on Structural Information and Communication Complexity

Otros eventos relacionados

- SPAA (2002 – 14)
  - ACM Symposium on Parallel Algorithms and Architectures
- ISADS
  - International Symposium on Autonomous Decentralized Systems
- International Summer School On Distributed Algorithms (2002 – 7)
  - University of Siena, Isola d'Elba, Livorno, ITALY

Primera Escuela de Sistemas Distribuidos

6 al 10 de Mayo, Xalapa, Veracruz México

La Algoritmica Distribuida

Roberto Gómez Cárdenas

ITESM-CEM

rogomez@campus.cem.itesm.mx

http://webdia.cem.itesm.mx/dia/ac/rogomez