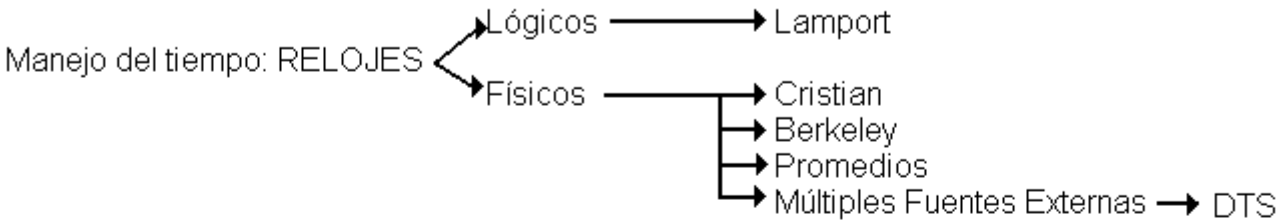
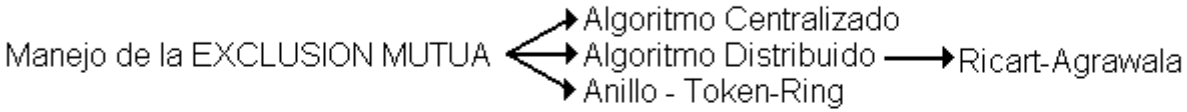
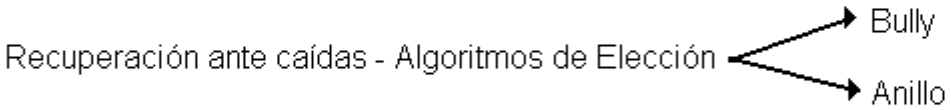
TEMAS DE SINCRONIZACION EN SISTEMAS DISTRIBUIDOS

Problemas de la sincronización - Porqué no es sencillo en un S.D.???

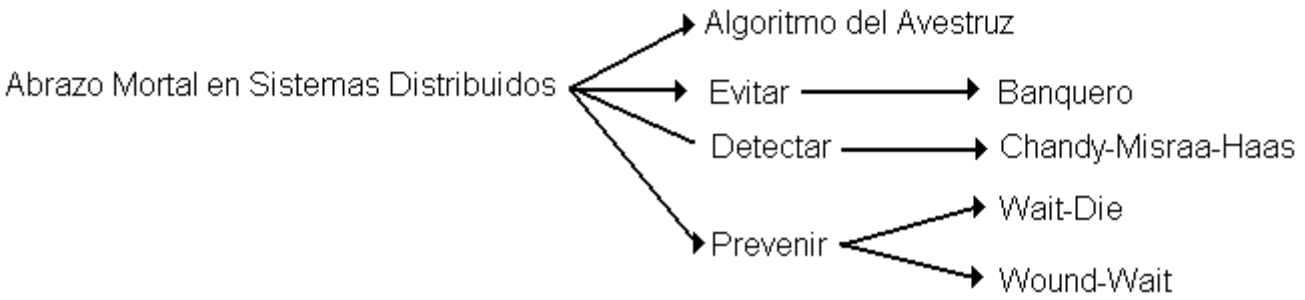




Ejemplo de uso de relojes : Consistencia de Caché



Transacciones



Porqué es dificil sincronizar en un S.D.?

Problemas:

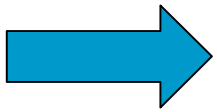
- Información repartida
- No existe timing global
- Decisiones con información local
- Puntos de falla?

- **Relojes Lógicos**

LAMPORT (1978)

Eventos **a** y **b** y sea T(A): Tiempo en el que ocurre el evento **a**

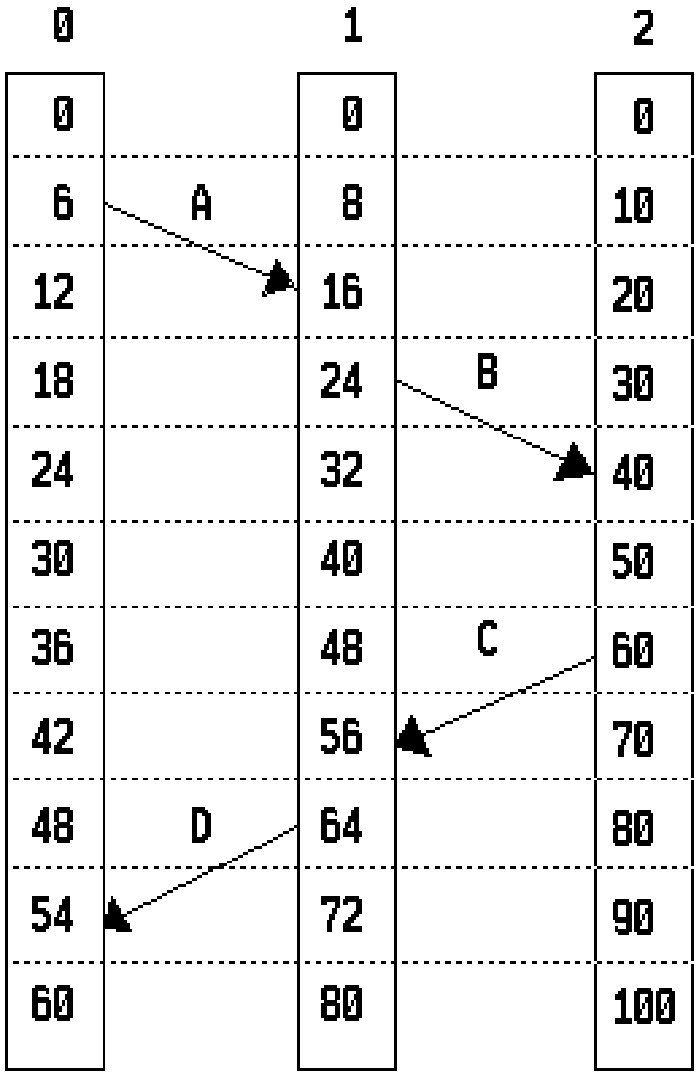
Relación **"sucede antes que"**

a  **b**

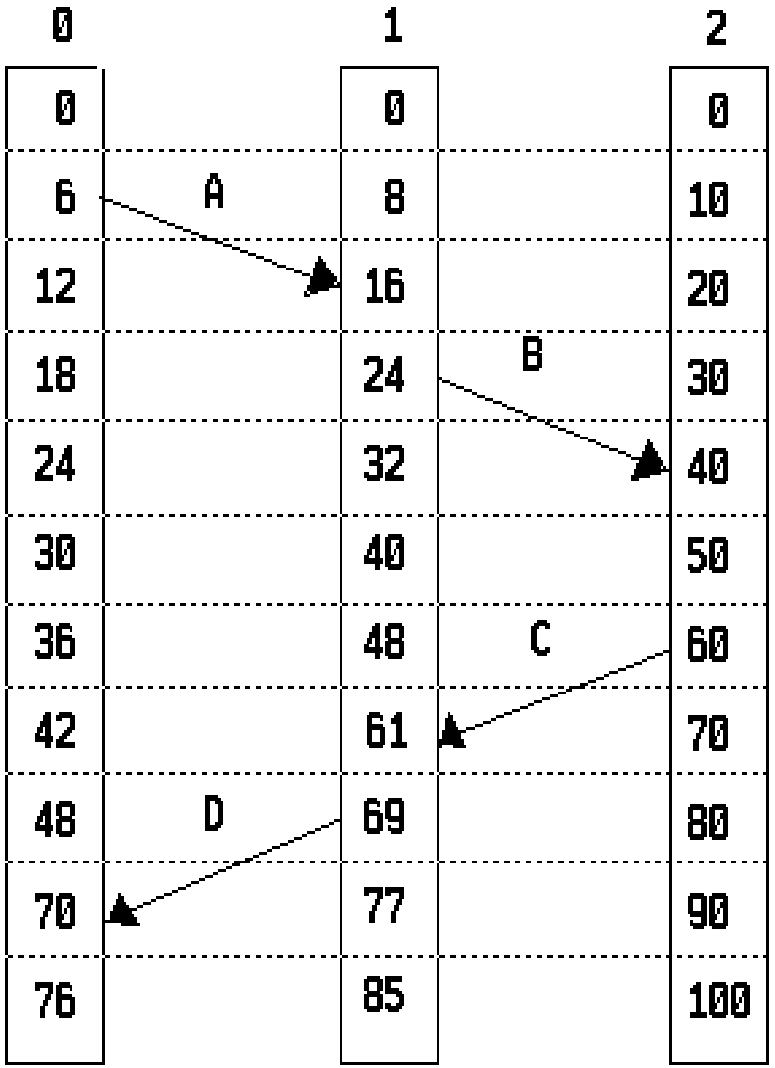
Si a es el evento de envío de un mensaje y b es el evento de recepción de ese mensaje, entonces

$$T(a) < T(b)$$

Sólo necesito sincronizar eventos cuando existe alguna relación entre ellos, sino no es necesario.

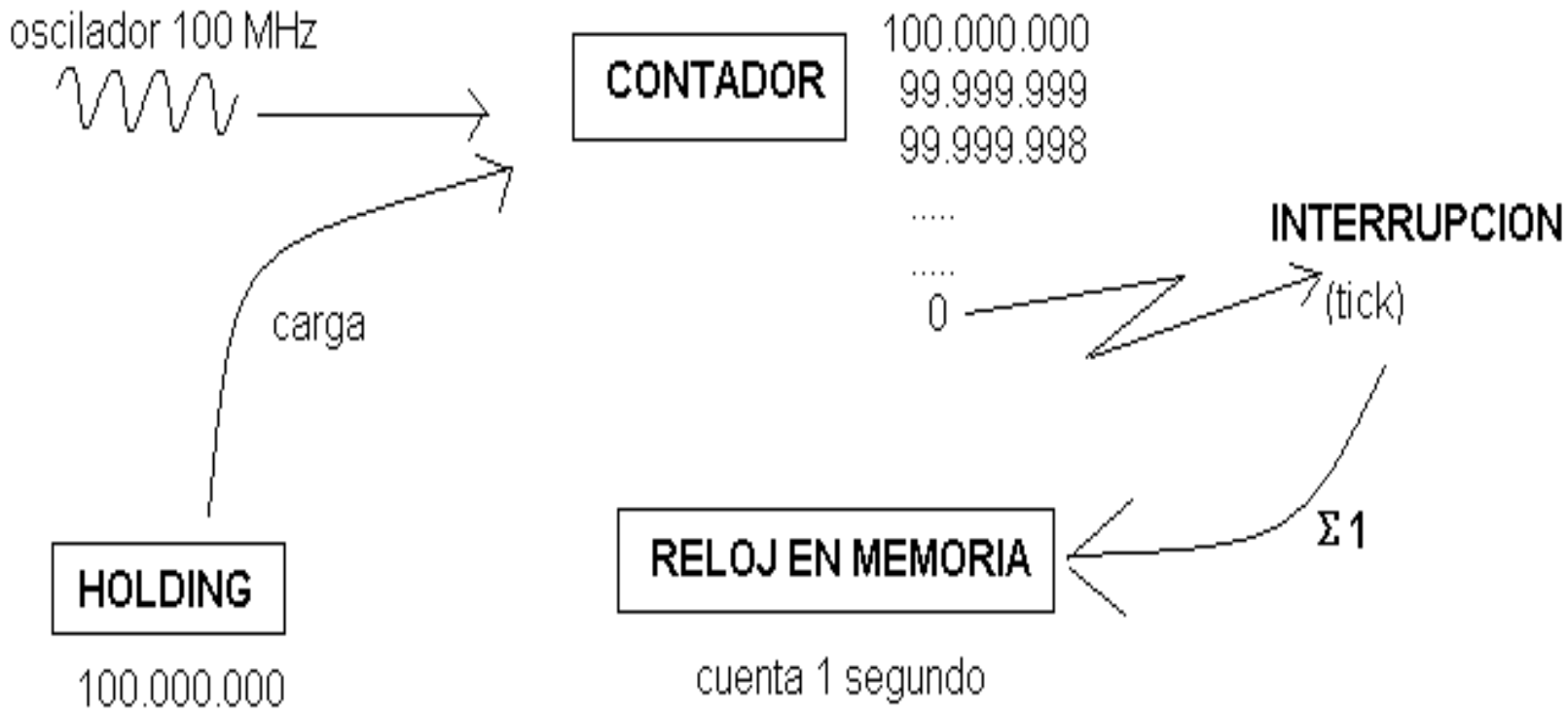


Tres procesos cada uno con su reloj que corre a diferente velocidad



El algoritmo de Lamport corrige los relojes

Cómo funciona el reloj de una computadora?



Y si el holding register tuviera un valor de 90.000.000, el reloj atrasaría o adelantaría?

Y cómo obtengo la hora exacta?

Hora GMT (Greenwich Mean Time)

reemplazada

Hora UTC (Universal Time Coordinated) (relojes de átomos de cesio 133) (*)

brindada por

Emisora de ondas de radio en Fort Collins (Colorado) transmite un pulso al inicio de cada segundo UTC. La emisora se denomina W W V (NIST)

(exactitud 10 msec)

(*) La hora exacta es la TAI que corrige la UTC con “segundos de salto” (TAI Tiempo Atómico Universal)

- **Relojes Físicos**

Algoritmo de CRISTIAN (1989)

- Existe un servidor de tiempo (pasivo). Puede existir W W V

- Cada nodo pregunta la hora al servidor cada x tiempo

Problemas:

- demora en el medio de transmisión

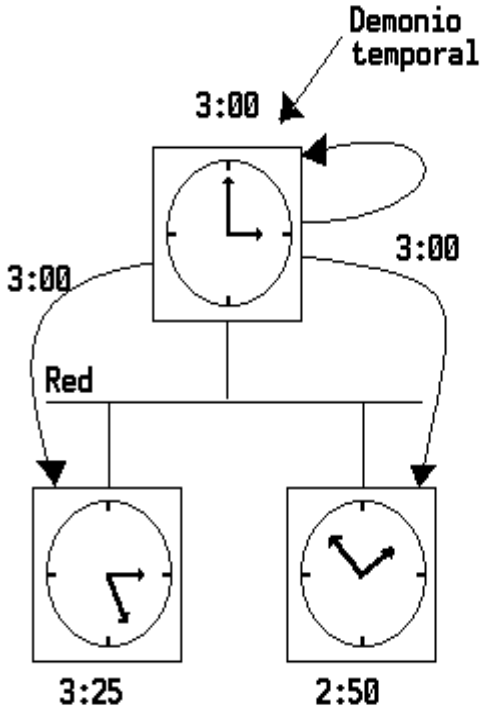
- si la hora del servidor está más atrasada que la hora local? Cómo atraso la hora local?

(**Uso del Holding register!!!**)

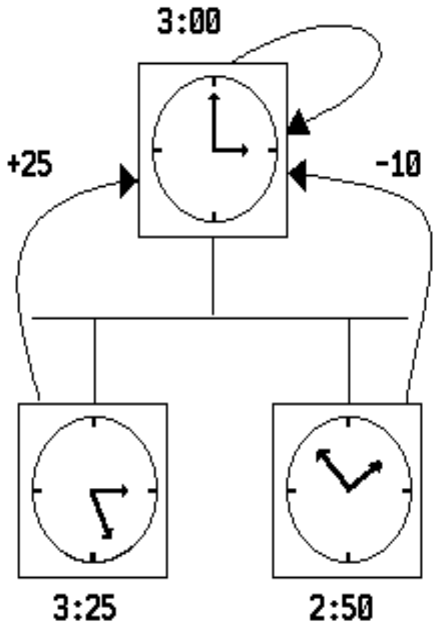
Algoritmo de BERKELEY (1989)

- Existe un servidor de tiempo (activo) que pregunta la hora a cada nodo y envía las correcciones

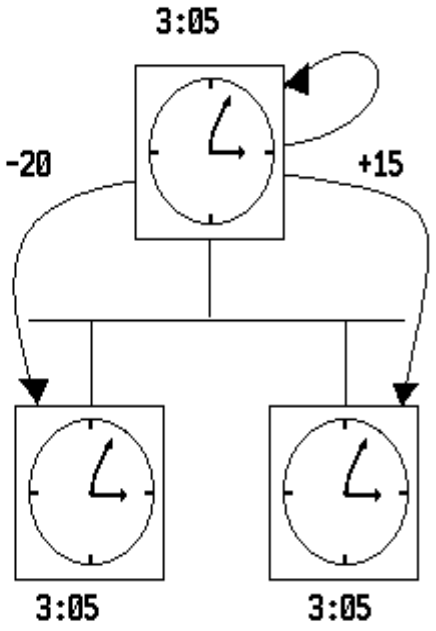
- Mismos problemas que Cristian



El demonio temporal pregunta a las otras máquinas por el valor de sus horas



Las máquinas responden



El demonio dice a todos cómo ajustar sus relojes

Algoritmo de **Promedios**

- Cada nodo hace broadcast (multidifusión) de su hora dentro de un cierto intervalo de tiempo.

- Cada nodo espera recibir la hora de los otros y una vez recolectadas, las promedia.

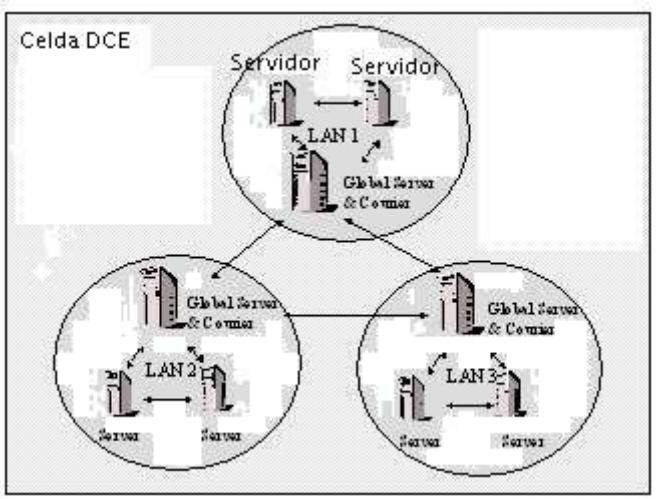
- Puede descartar los valores demorados en su transmisión.

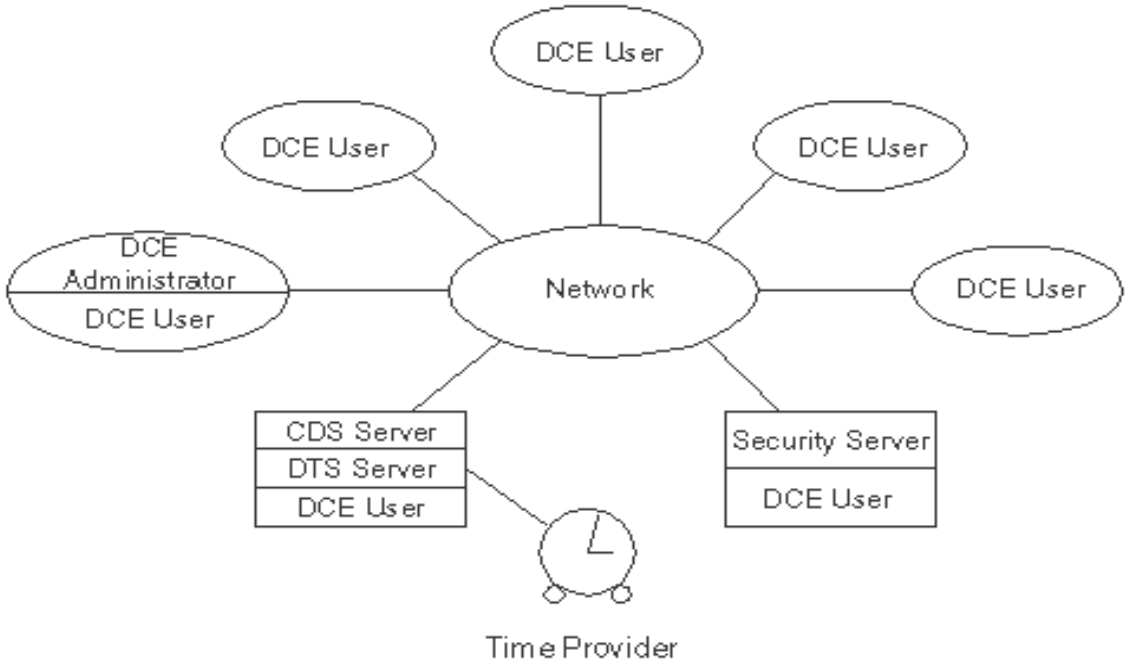
- Puede no haber máquina con WWV.

Algoritmo de **Múltiples fuentes externas** (DCE/DTS)

Estructura del S.O. DCE

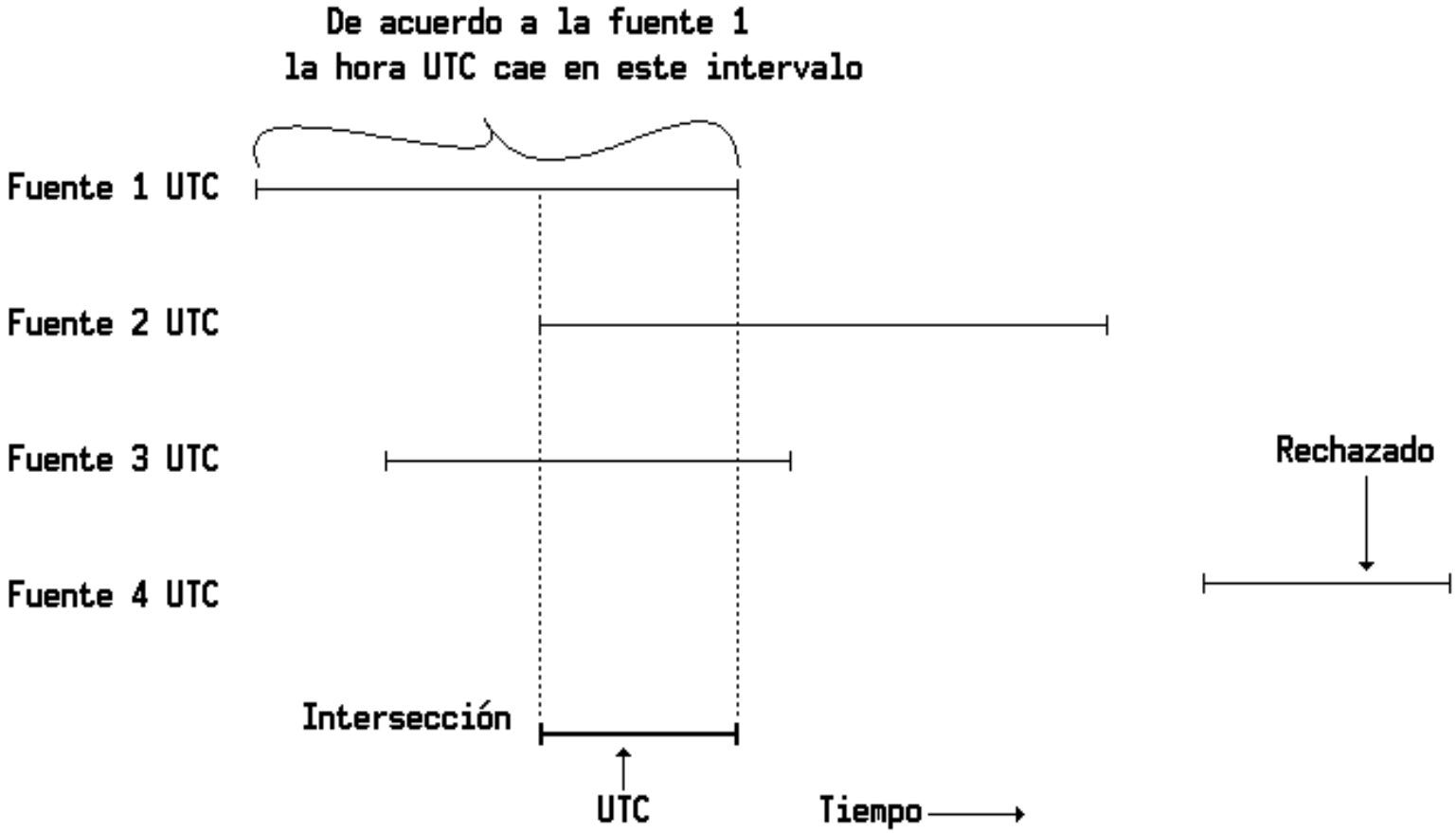
(Distributed Computing Environment corre sobre AIX, Unix, Windows, OS/2, etc. desarrollado por IBM/DEC/HP))





Cada celda tiene su propio servidor DTS (Distributed Time Service). Los DTS de cada celda se sincronizan entre si.

Algoritmo de **Múltiples fuentes externas** (DCE/DTS



Hace la intersección de las horas recibidas. La hora es un intervalo que contiene la hora exacta.

- **<u>Exclusión mutua en S.D.</u>**

Algoritmo Centralizado

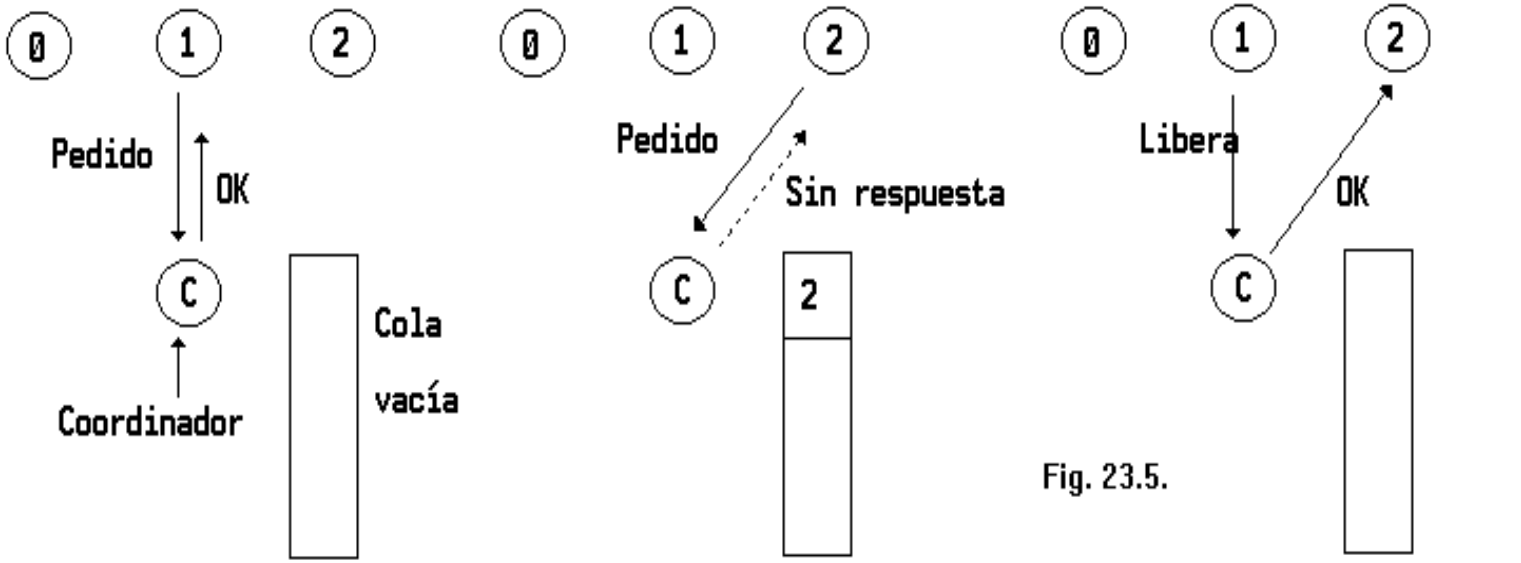


Fig. 23.5.

Proceso 1 pide al coordinador entrar a la zona crítica y se le otorga permiso

Proceso 2 pide luego entrar a la misma zona. El coordinador no responde

Cuando 1 termina y libera avisa al coordinador quien responde ahora al proc. 2

Problemas: Caída del coordinador. Además si el coordinador muere los procesos en espera no se enteran

Cantidad de mensajes necesarios: 3

Algoritmo de **Distribuido**
Ricart-Agrawala 1981

- Requiere sincronización con relojes lógicos

Un nodo pide a todos (broadcast):

- Si alguno retiene la región crítica, entonces no responde

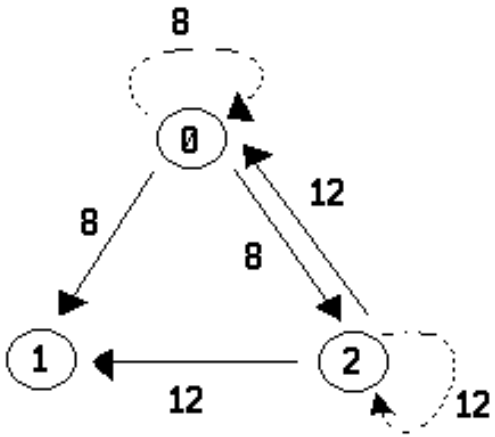
- Si un nodo no la necesita otorga acceso

- Si otro nodo la quiere se resuelve por Lamport

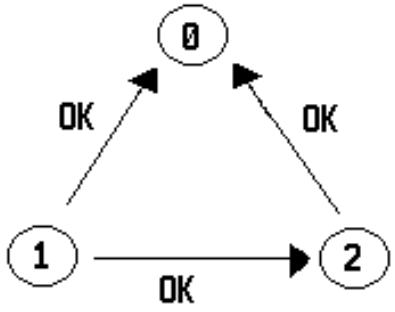
Luego de utilizar la zona crítica el nodo que la libera avisa al resto de nodos (broadcast)

Algoritmo de **Distribuido**

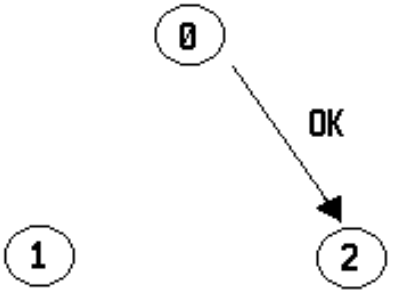
Ricart-Agrawala 1981 (cont.)



Dos procesos quieren la misma región crítica al mismo tiempo



El proceso 0 tiene la menor marca temporal y gana

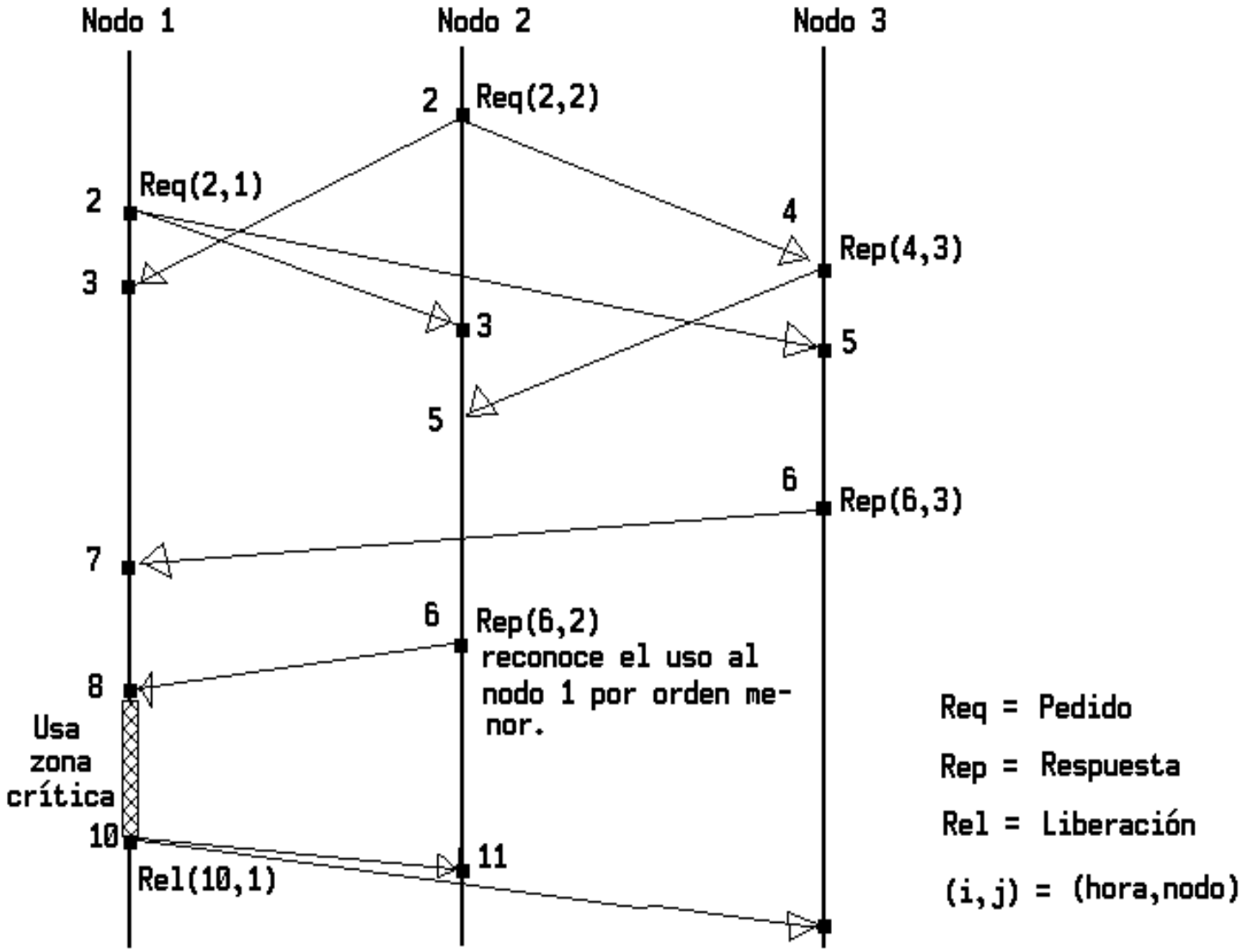


Cuando 0 termina envía un OK entonces el 2 puede entrar a la zona crítica

Y si no responden? --> Ricart-Agrawala **extendido** en el cual TODOS los nodos responden

Algoritmo de **Distribuido** Ricart-Agrawala 1981 (cont.)

Resolución Lamport cuando 2 nodos desean la misma región crítica



Algoritmo de **Distribuido** Ricart-Agrawala 1981 (cont.)

Problemas:

- Todos los nodos deben llevar la listas de todos los procesos activos
- Todos intervienen en las decisiones

Existen n puntos de falla (caída de cualquier nodo - se soluciona con el extendido)

Asegura exclusión mutua y libre de inanición

Mensajes necesarios 2(n-1)

Algoritmo **Token**

- Existe anillo lógico.
- Existe token que permite acceso a zona crítica.
- La retención del token permite el acceso.

Cada proceso debe mantener configuración actual del a-nillo.

Asegura exclusión mutua y libre de inanición

Problemas:

- Pérdida del token. Se perdió o algún nodo es lento?
- Caída de un nodo

Mensajes necesarios: 1 a infinito

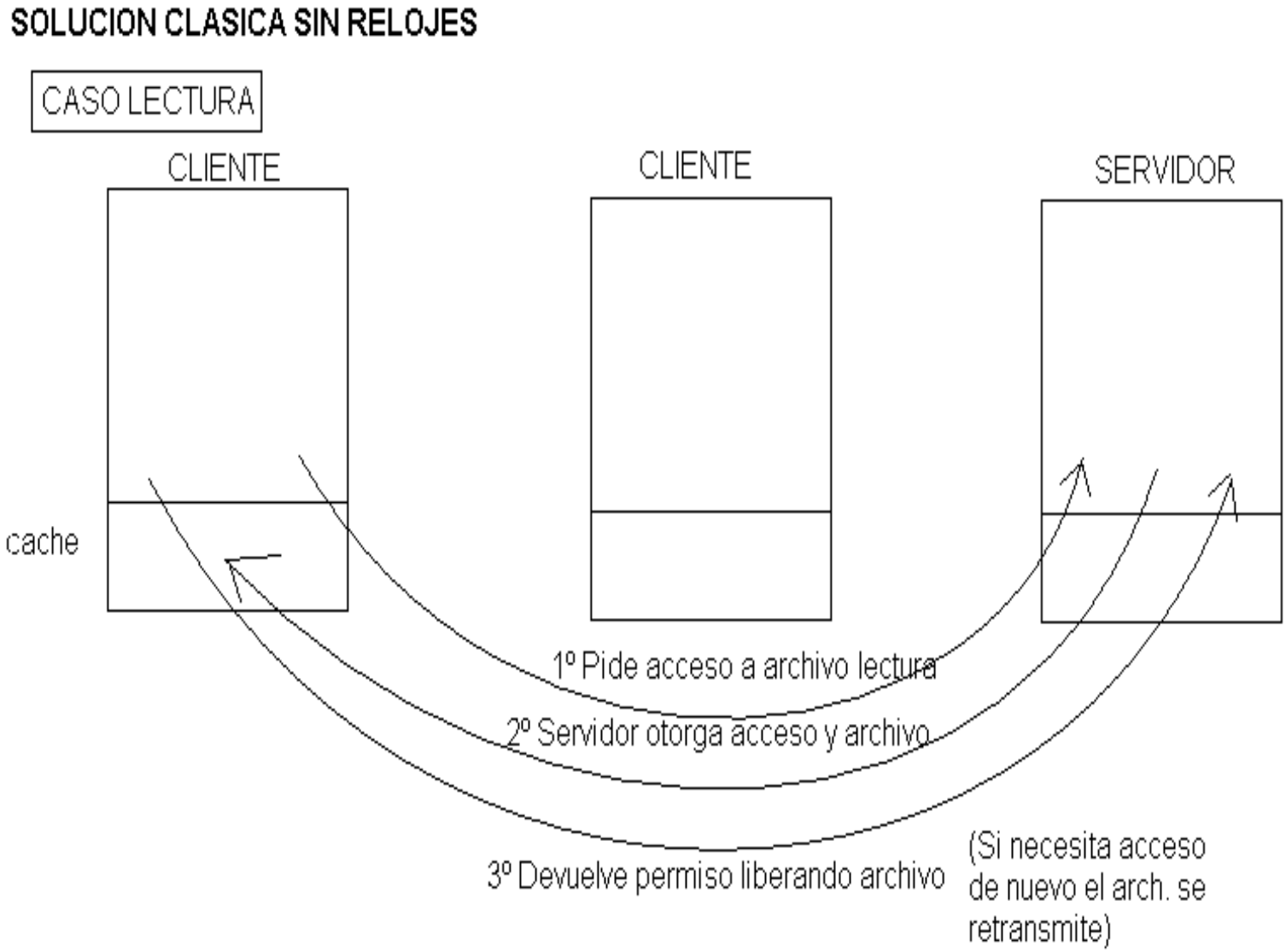
Cuál/es son las ventajas de contar con el uso de algunas de las técnicas de relojes en sistemas distribuidos?

Hay varias, por ejemplo:

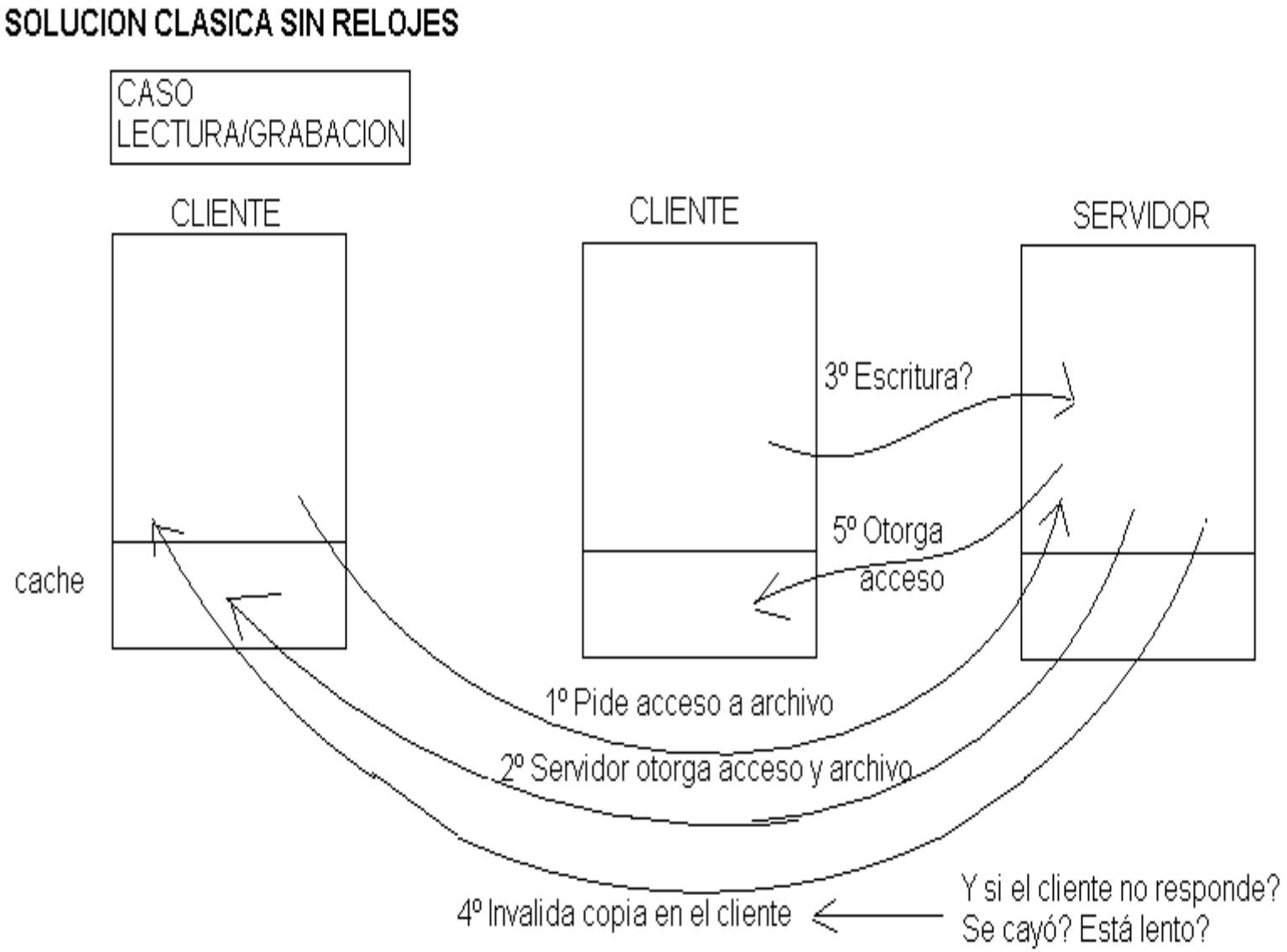
- Entrega de mensajes “a lo sumo 1” a un servidor.

- Mantenimiento de la consistencia de la memoria caché de los clientes

Consistencia de Cache

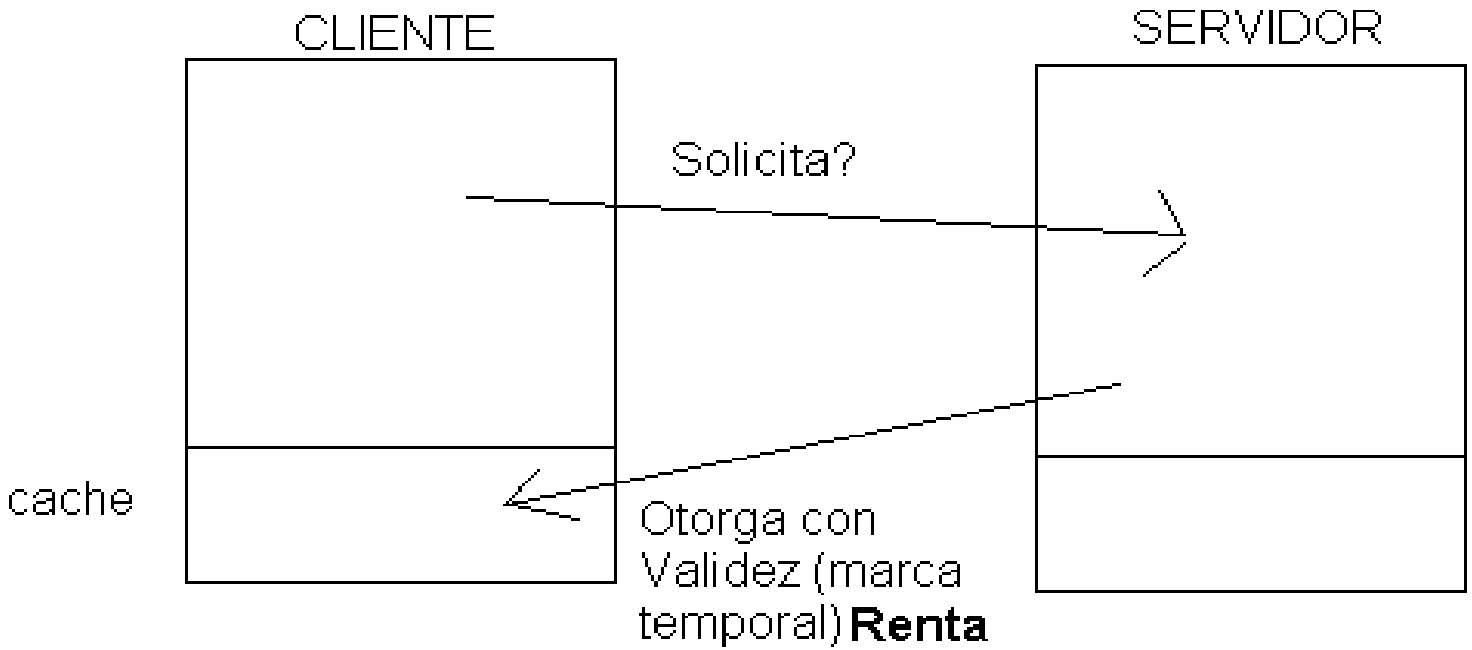


Consistencia de Cache



Consistencia de Cache

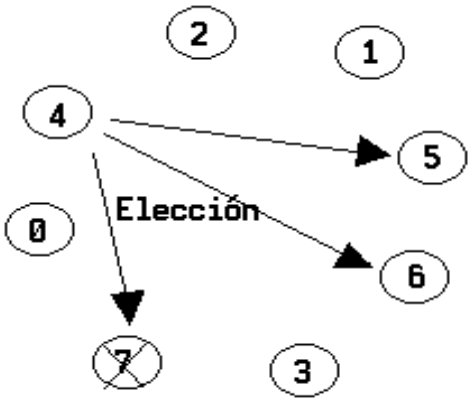
SOLUCION CON RELOJES



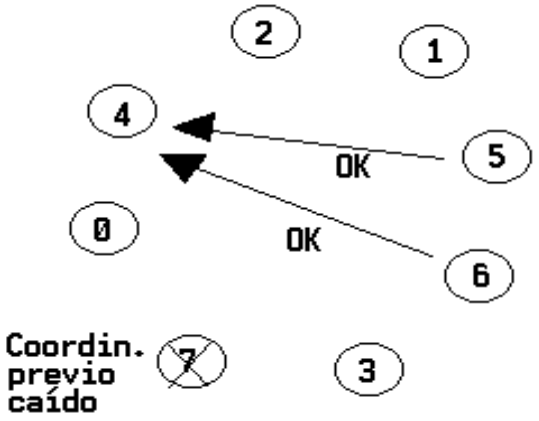
- Expiración de la renta --> Invalida la copia
- Revalidación de la renta
- Si otro cliente lo pide: invalido la copia o espero que expire (cliente caído)
- No necesita reenviar el archivo si la copia es válida.

Caída del Coordinador - **Algoritmos de elección**
Algoritmo Bully

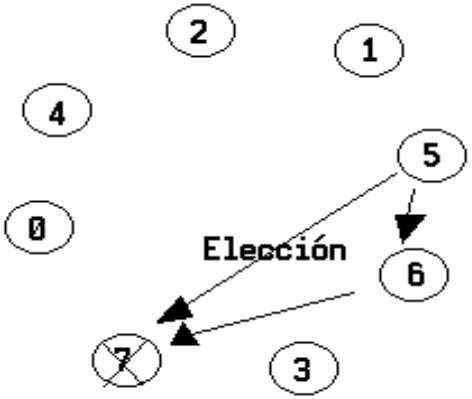
Los nodos están numerados y el de mayor número es el coordinador.



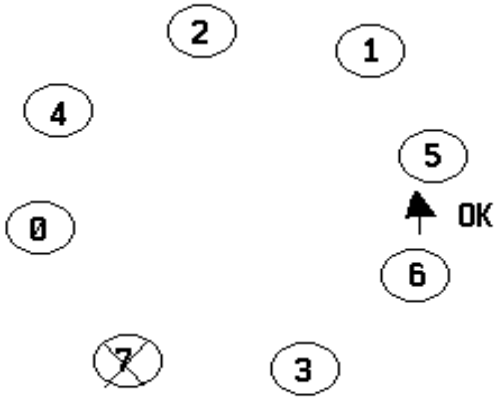
Proceso 4 inicia elección



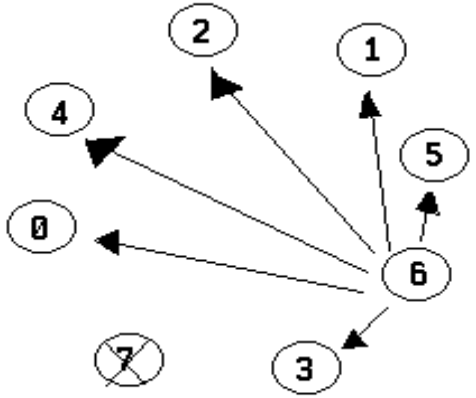
5 y 6 responden diciendo al 4 que se detenga



Ahora 5 y 6 inician elección



6 avisa a 5 que se detenga



Proceso 6 gana y avisa a todos

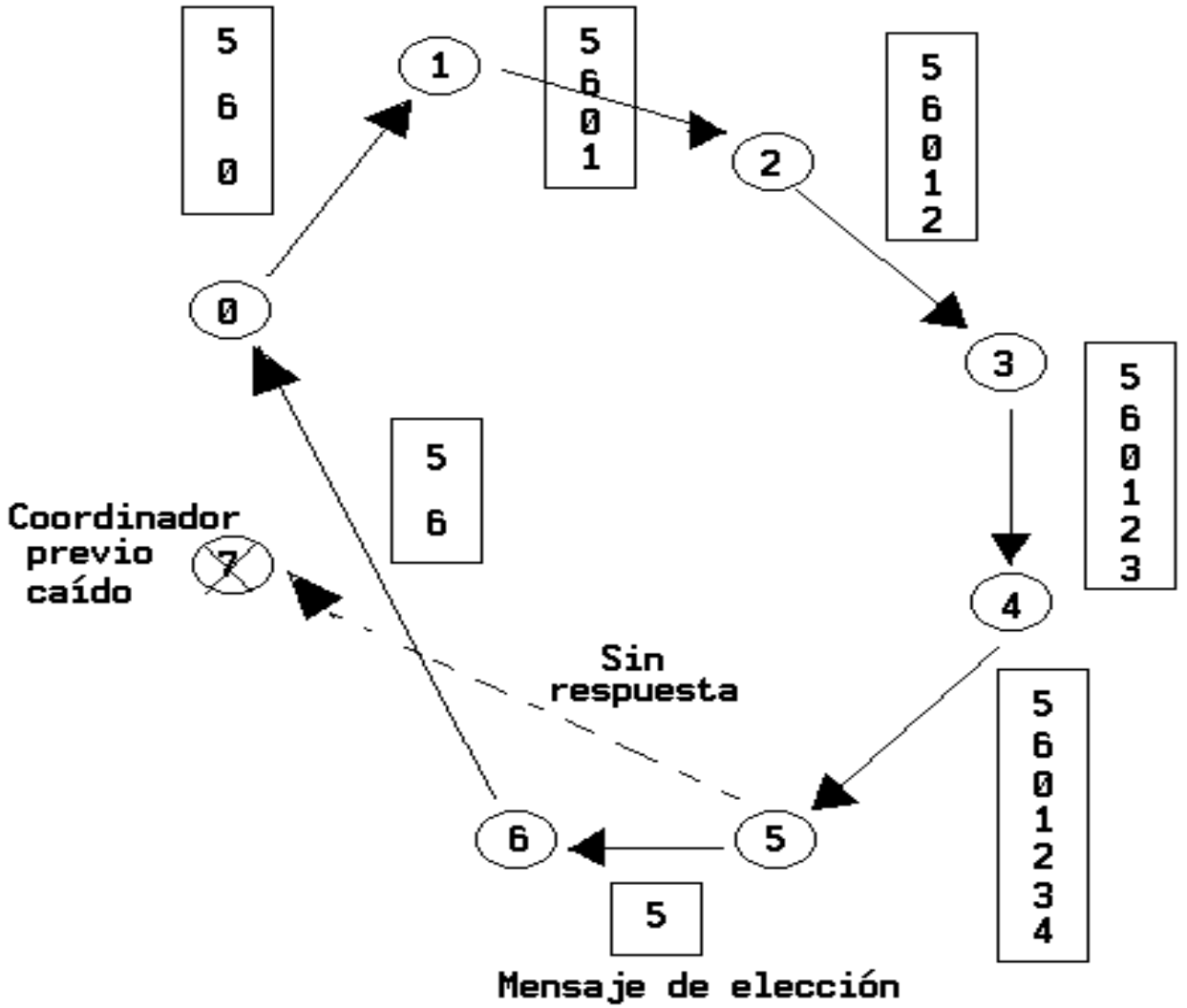
Algoritmo **Bully**

Si el coordinador caído se restaura desplaza al nuevo coordinador elegido. De allí el nombre Bully.

Algoritmo **Anillo (Token)**

Existe un anillo lógico entre los nodos

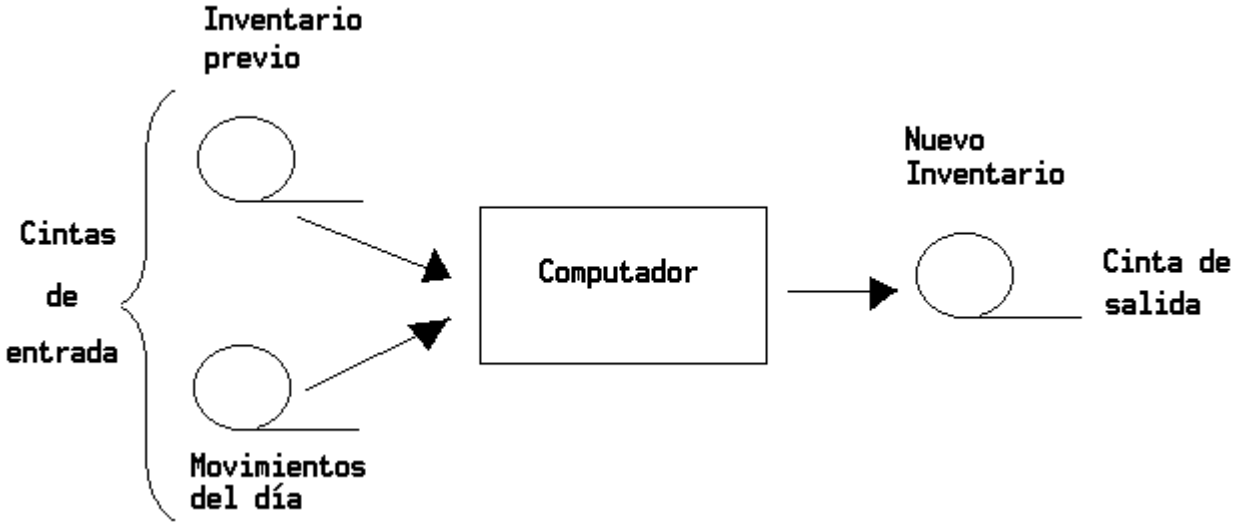
Algoritmo **Anillo**



TRANSACCIONES

- Proveen un mayor nivel de abstracción de la sincronización (no semáforos ni monitores)

- Concepto proviene del mundo de los negocios: transacciones comerciales



- El todo-o-nada

- Uso de almacenamiento estable (**RAID**)

- Primitivas propias: BEGIN_TRANSACTION, END_TRANSACTION, ABORT_TRANSACTION

Transacciones

Otras primitivas

COMMIT : La TR se completa satisfactoriamente

ROLLBACK : La TR se cancela y sus cambios desaparecen

EL MODELO

- Existen procesos independientes y cualquiera puede fallar

- La comunicación no es confiable pero se pueden detectar las fallas y manejarlas (time-out)

- Los errores de comunicación se manejan transparentemente por el software subyacente.

Transacciones - Propiedades (ACID)

Atomicidad: propiedad del todo-o-nada

La transacción se completa totalmente o todo queda como si la TR no hubiera existido

Consistencia: la TR mantiene el invariante del sistema

Isolation (Serialicidad): las transacciones concurrentes no interfieren entre sí.

Durability (Permanencia): una vez que la TR hizo COMMIT sus cambios son permanentes

Transacciones - Propiedades (ACID)

Consistencia: la TR mantiene el invariante del sistema

Ejemplo:

Pablo tiene $100 y María tiene $20 y son los únicos clientes que tiene el banco. Pablo desea transferirle $25 a Maria y ejecuta una TR de transferencia de fondos.

Ya sea que los fondos lleguen o no a María el saldo final de caja del Banco debe ser $120.

Transacciones - Propiedades (ACID)

Isolation (Serialicidad): las transacciones concurrentes no interfieren entre sí.

Si 2 o más transacciones se ejecutan al mismo tiempo el resultado final puede verse como si todas ellas se hubieran corrido secuencialmente en algún orden

TR A	TR B
1. X = 0	3. X = 1
2. Y = X + 8	4. Y = X + 32

Resultados válidos: X=0 e Y=8 o X=1 e Y=33

Resultados inválidos: Y= { 9, 41, 40 } un orden inválido de ejecución posible podría ser 1.3.4.2.

TRANSACCIONES

Permanencia y Transacciones anidadas

Las TR pueden tener subtransacciones

Qué sucede si una subtransacción (TR anidada) hace COMMIT y su TR madre hace ROLLBACK?

La TR anidada debió alterar el universo real por la propiedad de **Permanencia,** pero la TR madre debería deshacer los cambios???!!!

Se cumple o no esta propiedad?

SI ! Reformulamos la propiedad:

Las transacciones anidadas son permanentes en el universo de la TR madre y no en el universo real.

TRANSACCIONES - Implementaciones

ESPACIO PRIVADO DE TRABAJO

- Se copia todo a espacio de trabajo propio

Problemas: Costo de la copia en tiempo y espacio de almacenamiento

Ventaja: Deshacer una TR es fácil. Se borra el espacio y listo :-)

TRANSACCIONES - Implementaciones

LOG DE GRABACION ANTICIPADA (Lista de intenciones)

- La TR graba un log en almacenamiento estable en el que indica todos los cambios. Valor viejo - Valor nuevo

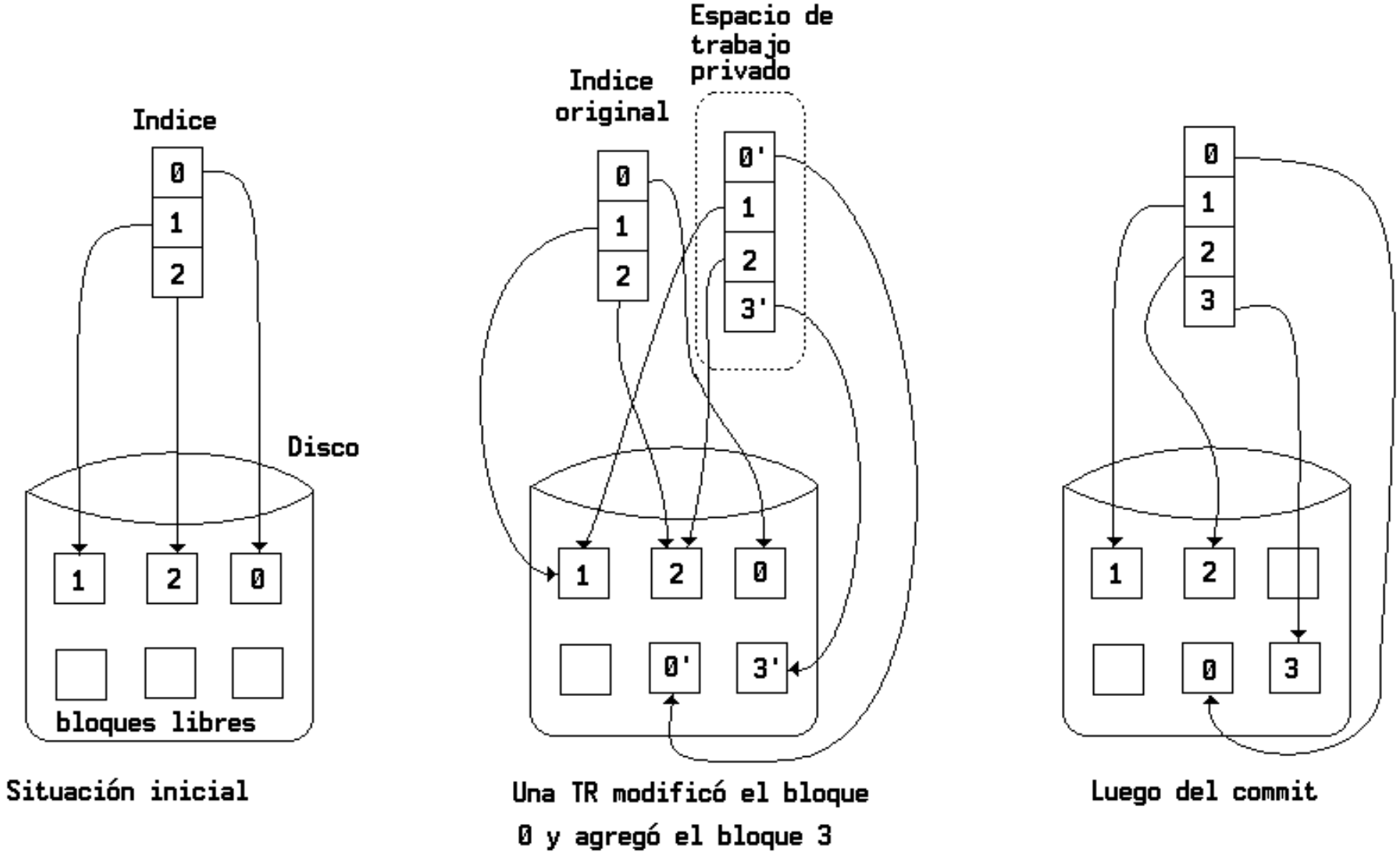
- La TR altera el universo real

Problema: Más difícil de implementar. El log puede ser cuello de botella.

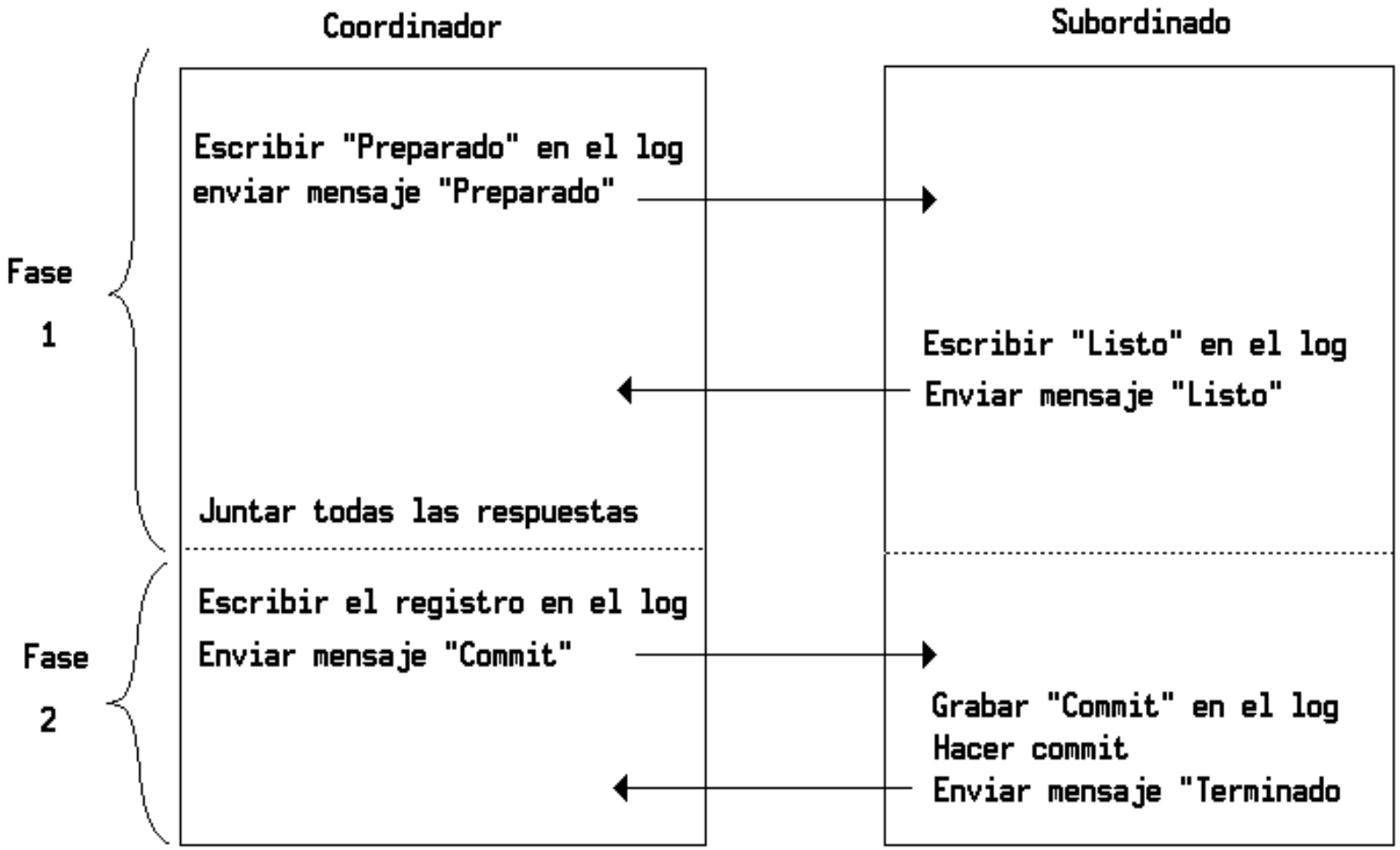
Ventaja: Deshacer (ROLLBACK) una TR es sencillo utilizando el log para recorrerlo hacia atrás y deshacer los cambios.

El log también provee tolerancia ante caídas.

Espacio privado de trabajo - Un ejemplo



TRANSACCIONES - Protocolo Two-Phase-Commit en TR distribuidas



TRANSACCIONES - Protocolo Two-Phase-Commit

Tolerancia a fallas

- Si el coordinador se cae luego de enviar su intención de commit, cuando se levanta ve las respuestas en el log

- Si el hijo se cae luego de enviar su respuesta, al restaurarse ve la decisión en el log

- Si un hijo no responde cuando el coordinador le envía su decisión espera un time-out y *eventualmente* decide si cancela o no la TR.

- Si el coordinador se cae luego de grabar su decisión en el log cuando se levanta puede seguir normalmente.

CONTROL DE CONCURRENCIA

Qué técnicas se pueden utilizar para controlar el acceso concurrente a recursos en sistemas distribuidos?

- Bloqueo (locking)

- Control de concurrencia optimista

- Sellos temporales (timestamp)

BLOQUEO

- Bloqueo de lectura (permite otras lecturas)

- Bloqueo de escritura (no permite otras lecturas)

El bloqueo puede ser a muy bajo nivel o a un alto nivel: GRANULARIDAD DEL BLOQUEO

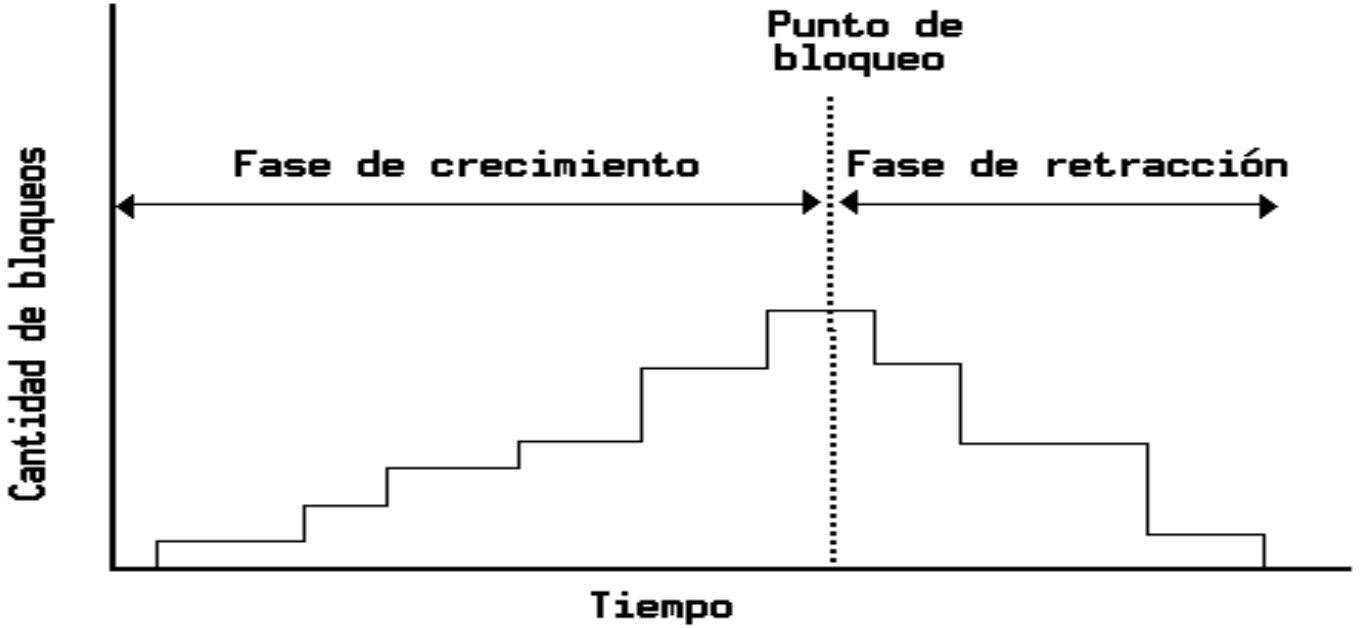
Problemas:

- Alto costo (recursos no disponibles, costo de implementación, impacto negativo a la performance)

- No está libre de deadlock

BLOQUEO

Bloqueo de dos etapas: reduce la posibilidad de abrazo mortal.



Si todas las TR usan esta forma todas sus intercalaciones son serializables.

Si la liberación ocurre en momento de COMMIT se llama bloqueo de dos etapas **estricto**.

Control de concurrencia optimista

Utiliza un log.

Todo recurso que es solicitado se otorga. Cuando la TR quiere hacer COMMIT revisa el log y ve si sus datos fueron accedidos por otras TR.
Si fueron accedidos cancela y sino hace commit.

- Está libre de deadlock y ofrece el mayor paralelismo.

- Funciona bien en la implementación con espacio privado de trabajo

Problema: Cuando hay mucha carga de trabajos cancelar y reiniciar una TR puede traer mayor probabilidad de fallas.(muere nuevamente)

SELLOS TEMPORALES (Timestapms - 1983)

- Cada TR posee un sello de tiempo en el momento en que se inicia.
- Existe ordenamiento por Lamport.
- Cada archivo tiene un sello de lectura y uno de escritura que indican qué TR cumplida (o sea, que hizo COMMIT) es la última que lo leyó o lo actualizó.
- Cuando la TR encuentra un sello mayor aborta.

Asegura orden correcto de las acciones
Si orden es incorrecto: una TR que empezó después que la actual manejó el archivo e hizo COMMIT con lo cual la TR actual es tardía y debe ser abortada.

Están libres de deadlock pero son de compleja implementación

ABRAZO MORTAL en S.D.

Es más difícil de manejar que en sistemas centralizados por lo disperso de la información.

Estrategias de manejo:

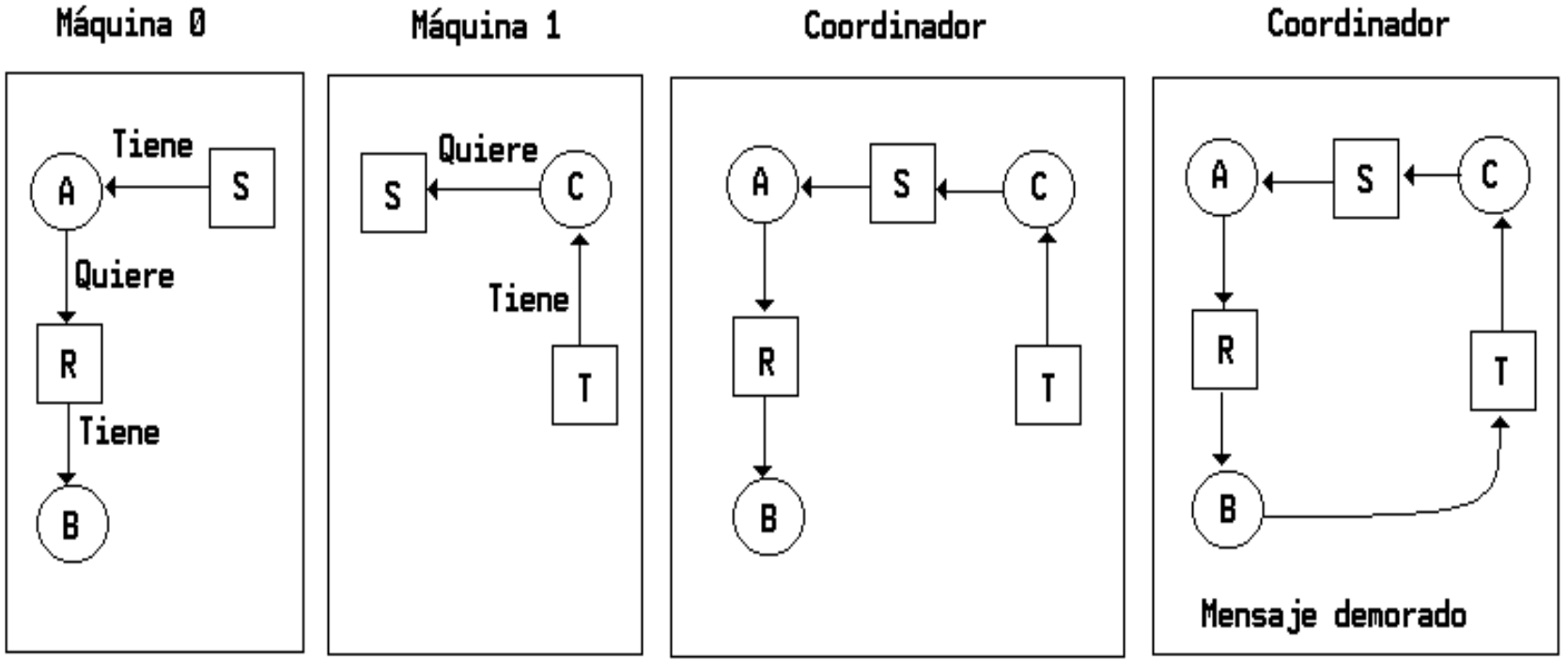
- Algoritmo del avestruz: Ignorar el problema. Se utiliza en entornos de oficina

- Detección

- Evitar que ocurra: Banquero. Imposible de implementar en un S.D.!!!

- Prevenir: hacer que el A.M. sea imposible desde el punto de vista estructural

Detección centralizada de A.M.

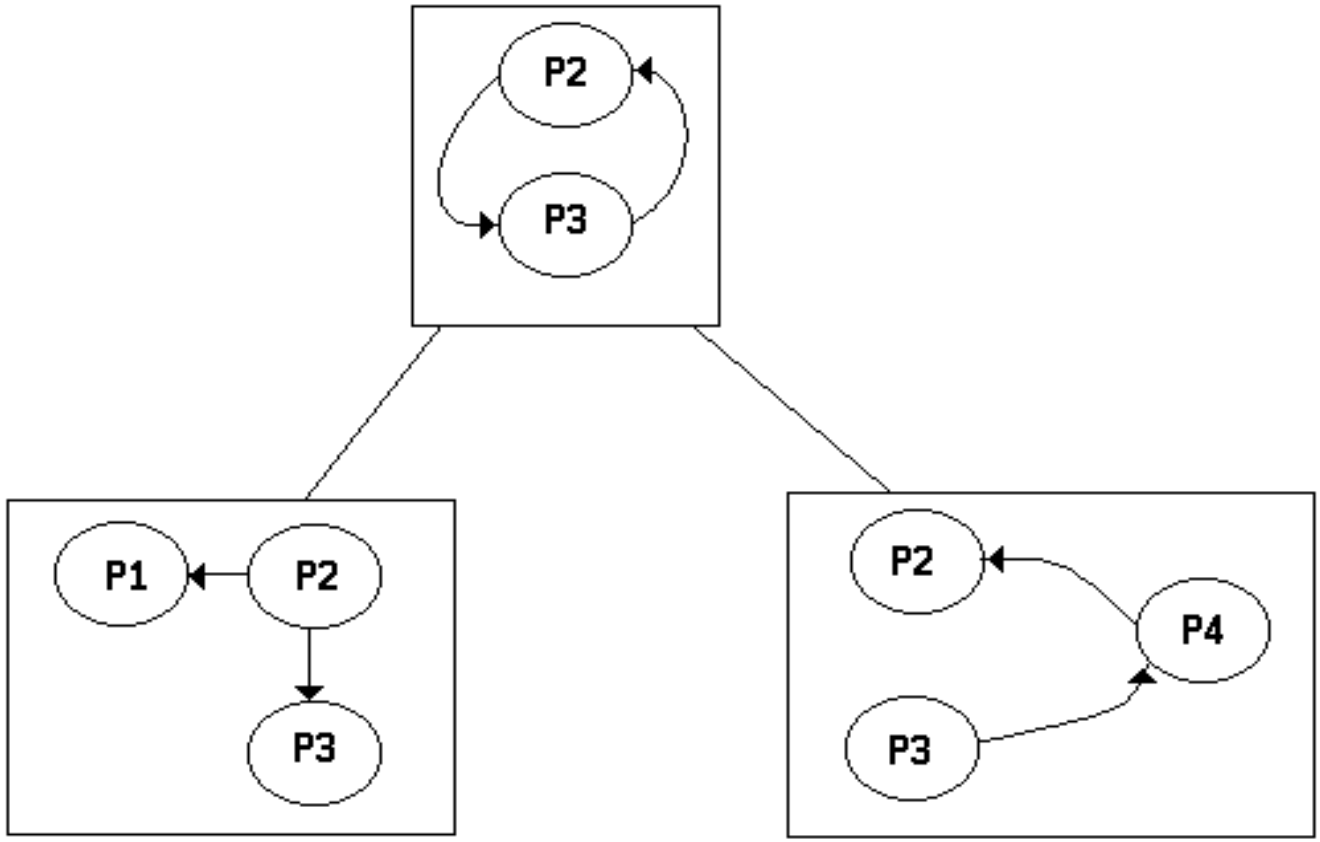


Problema: Falsos abrazos mortales

Solución: Utilizar Lamport y antes de decidir la existencia del A.M. preguntar a los nodos su situación.

Detección jerárquica de A.M.

Los nodos superiores van guardando la información de los grafos de sus subordinados



Detección distribuida de A.M.

(Chandy-Misra-Haas 1983)

Cuando un proceso en un nodo se pone a esperar por un recurso inicia un mensaje que transmite a los nodos que retienen recursos por los cuales él espera. Si el mensaje le vuelve entonces hay ciclo y el proceso se suicida.

El mensaje tiene 3 elementos:

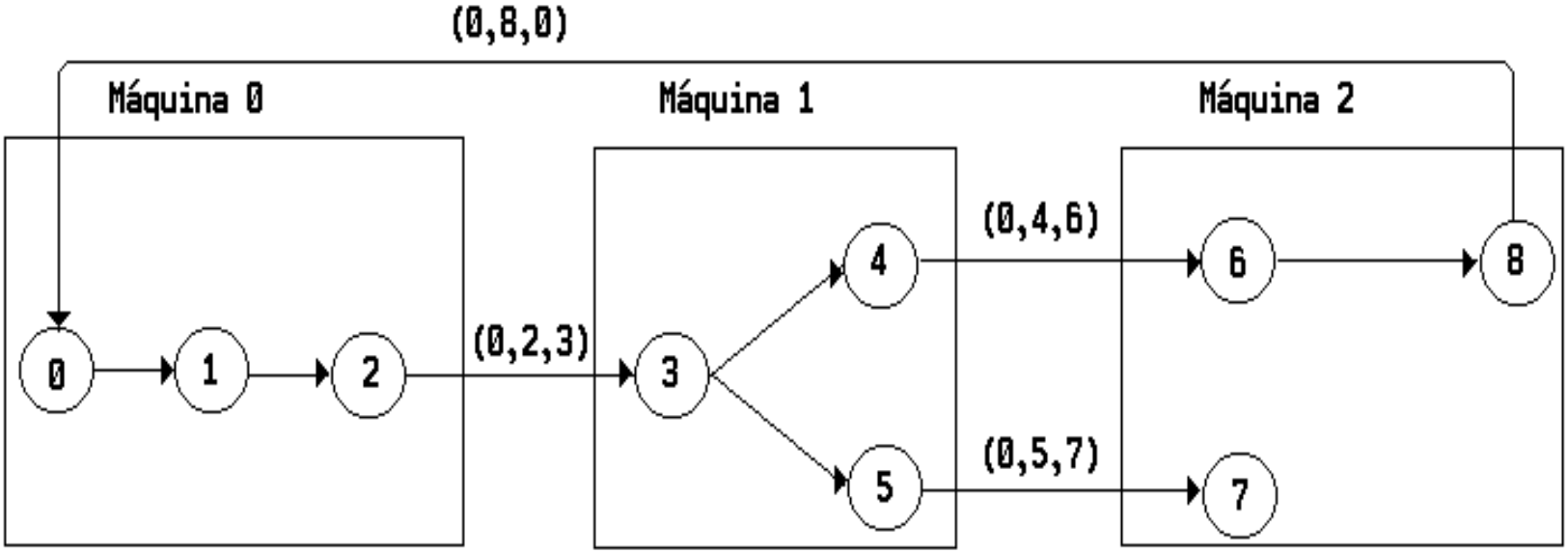
- cuál es el proceso que se bloquea

- cuál es el proceso que envía el mensaje

- cuál es el proceso que retiene un recurso por el cuál el proceso espera

Detección distribuida de A.M.

(Chandy-Misra-Haas 1983)



Problema: Si dos procesos inician el mensaje y pertenecen al mismo ciclo pueden morir ambos

Solución: agregar al mensaje la ruta por la cual pasa y el proceso que lo emitió originalmente decide entonces que se suicide el de menor (o mayor) número.

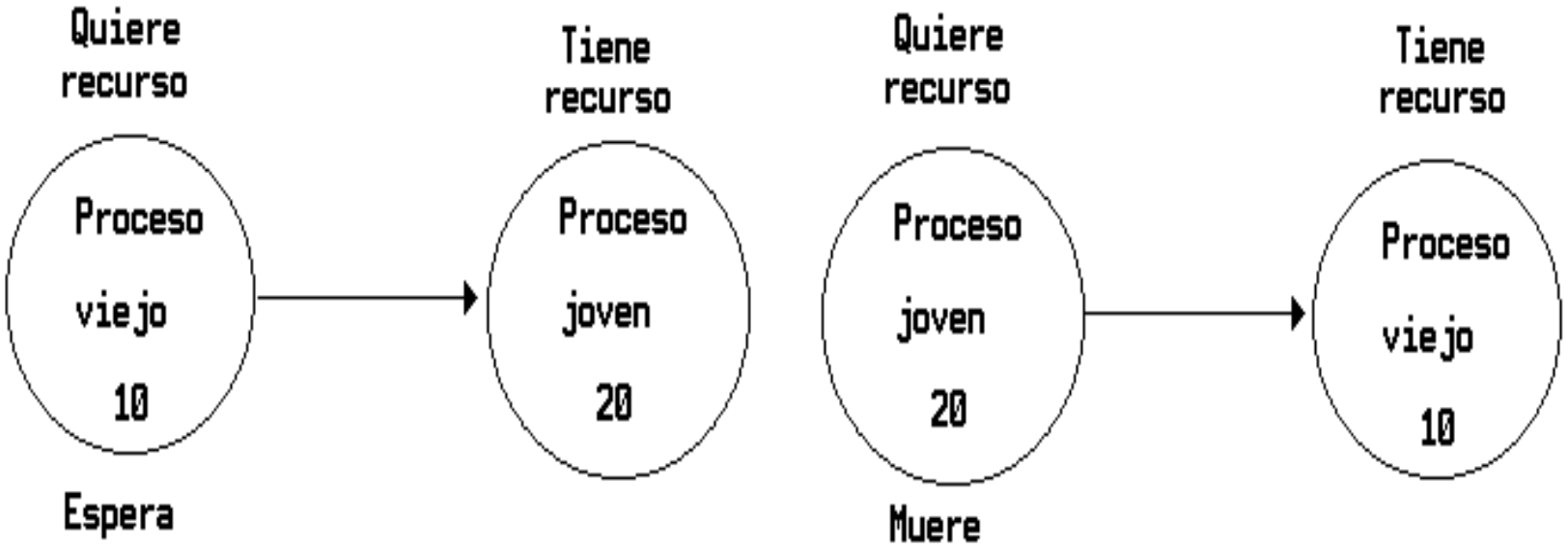
Prevención distribuida de A.M.

Algunas técnicas:
- los procesos sólo pueden retener de a un recurso por vez,
- los procesos deben requerir todos sus recursos al inicio,
- los procesos deben liberar todos sus recursos al requerir uno nuevo.
- Ordenamiento de los recursos.

Pero en un S.D. con relojes lógicos y T.R. hay un par de técnicas nuevas que se pueden utilizar.
Cada TR tendrá un sello temporal de inicio único.
Estas son las técnicas Wait-Die y Wound-Wait.

Prevención distribuida de A.M.

Algoritmo Wait-Die



Prevención distribuida de A.M.

Algoritmo Wound-Wait

Prevención distribuida de A.M

- No es crítico matar una transacción

- Ambos esquemas permiten la inanición.

Solución:
A una nueva transacción no se le asigna un nuevo sello temporal cuando se la reinicia luego de haber sido cancelada (eventualmente envejecerá y adquirirá el recurso).