Sistemas Operativos Distribuidos

Sincronización, Concurrencia y Transacciones

Sincronización en Sistemas Distribuidos

Más compleja que en los centralizados

- Propiedades de algoritmos distribuidos:
  - La información relevante se distribuye entre varias máquinas.
  - Se toman decisiones sólo en base a la información local.
  - Debe evitarse un punto único de fallo.
  - No existe un reloj común.
- Problemas a considerar:
  - Tiempo y estados globales.
  - Exclusión mutua.
  - Algoritmos de elección.
  - Operaciones atómicas distribuidas: Transacciones

Sistemas Operativos Distribuidos

Tiempo y Estados

- Relojes Distribuidos
- Relojes Lógicos

Sincronización de Relojes Físicos

Relojes hardware de un sistema distribuido no están sincronizados.

Necesidad de una sincronización para:

- En aplicaciones de tiempo real.
- Ordenación natural de eventos distribuidos (fechas de ficheros).

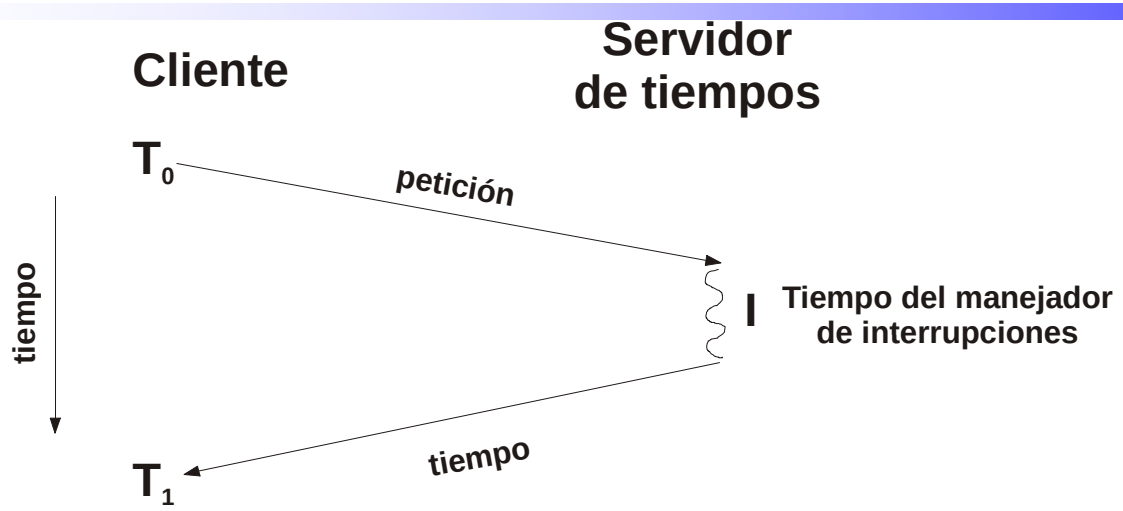
Concepto de sincronización:

- Mantener relojes sincronizados entre sí.
- Mantener relojes sincronizados con la realidad.

UTC: *Universal Coordinated Time*

- Transmisión de señal desde centros terrestres o satélites.
- Una o más máquinas del sistema distribuido son receptoras de señal UTC.

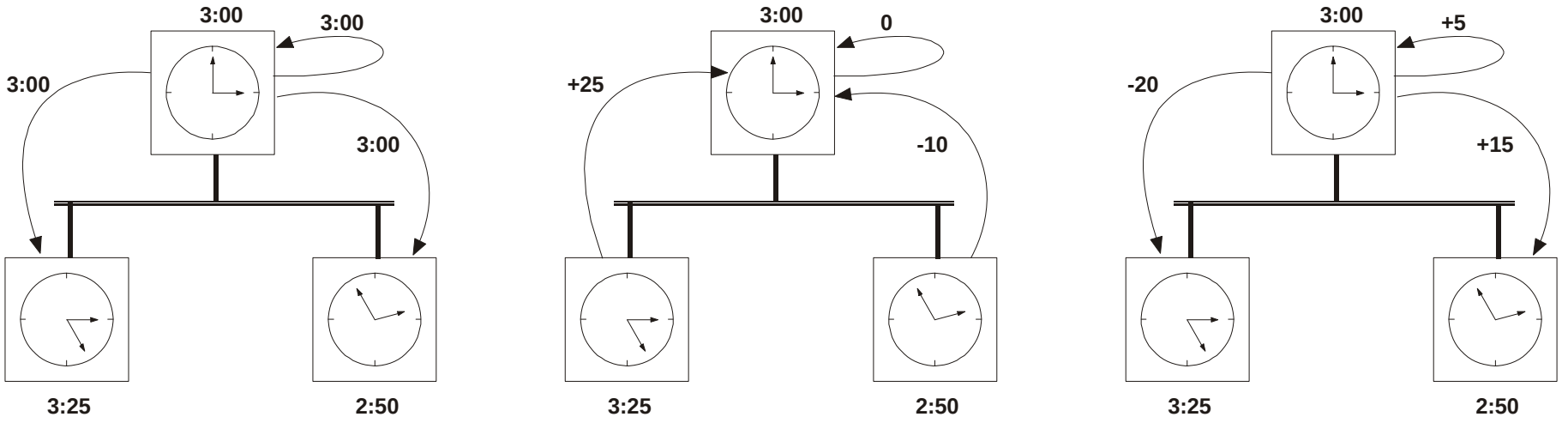
Algoritmo de Cristian



- Adecuado para sincronización con UTC.
- Tiempo de transmisión del mensaje: (T1-T0)/2
- Tiempo en propagar el mensaje: (T1-T0-I)/2
- Valor que devuelve el servidor se incrementa en (T1-T0-I)/2
- Para mejorar la precisión se pueden hacer varias mediciones y descartar cualquiera en la que T1-T0 exceda de un límite

Algoritmo de Berkeley

- El servidor de tiempo realiza un muestreo periódico de todas las máquinas para pedirles el tiempo.
- Calcula el tiempo promedio e indica a todas las máquinas que avancen su reloj a la nueva hora o que disminuyan la velocidad.
- Si cae servidor: selección de uno nuevo (alg. de elección)



Protocolo de Tiempo de Red

NTP (*Network Time Protocol*).

- Aplicable a redes amplias (Internet).
- La latencia/retardo de los mensajes es significativa y variable.

Objetivos:

- Permitir sincronizar clientes con UTC sobre Internet.
- Proporcionar un servicio fiable ante fallos de conexión.
- Permitir resincronizaciones frecuentes.
- Permitir protección ante interferencias del servicio de tiempo.

Organización:

- Jerarquía de servidores en diferentes *estratos*.
- Los fallos se solventan por medio de ajustes en la jerarquía.

Protocolo de Tiempo de Red

La sincronización entre cada par de elementos de la jerarquía:

- Modo multicast: Para redes LAN. Se transmite por la red a todos los elementos de forma periódica. Baja precisión.
- Modo de llamada a procedimiento: Similar al algoritmo de Cristian. Se promedia el retardo de transmisión. Mejor precisión.
- Modo simétrico: Los dos elementos intercambian mensajes de sincronización que ajustan los relojes. Mayor precisión.

Los mensajes intercambiados entre dos servidores son datagramas UDP.

Causalidad Potencial

En ausencia de un reloj global la relación *causa-efecto* (tal como *precede a*) es una posibilidad de ordenar eventos.

Relación de causalidad potencial (Lamport)

- e_{ij} = evento *j* en el proceso *i*
- Si $j < k$ entonces $e_{ij} \rightarrow e_{ik}$
- Si e_i=send(m) y e_j=receive(m), entonces $e_i \rightarrow e_j$
- La relación es transitiva.

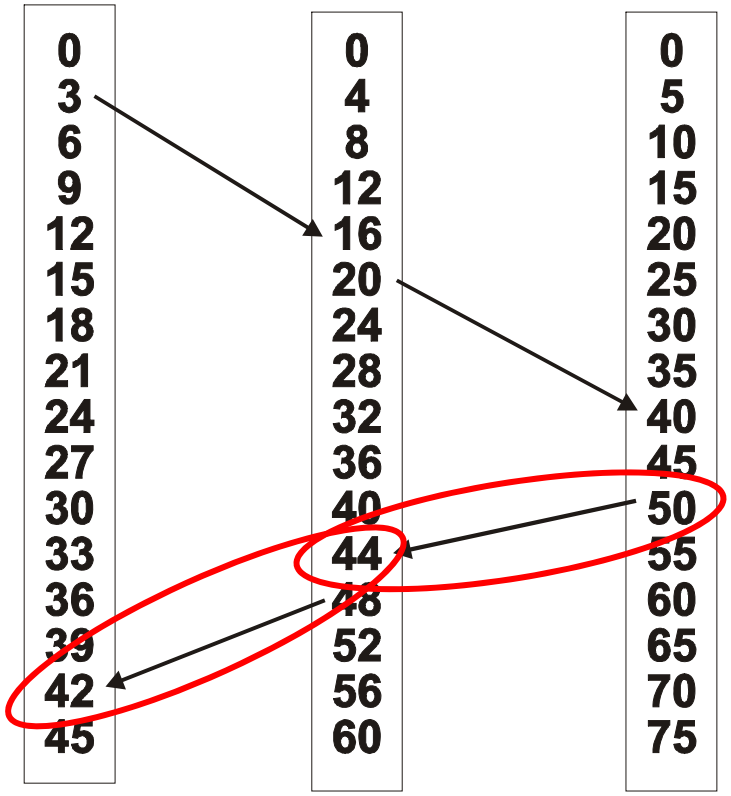
Dos eventos son concurrentes (a || b) si no se puede deducir entre ellos una relación de causalidad potencial.

Relojes Lógicos (Algoritmo de Lamport)

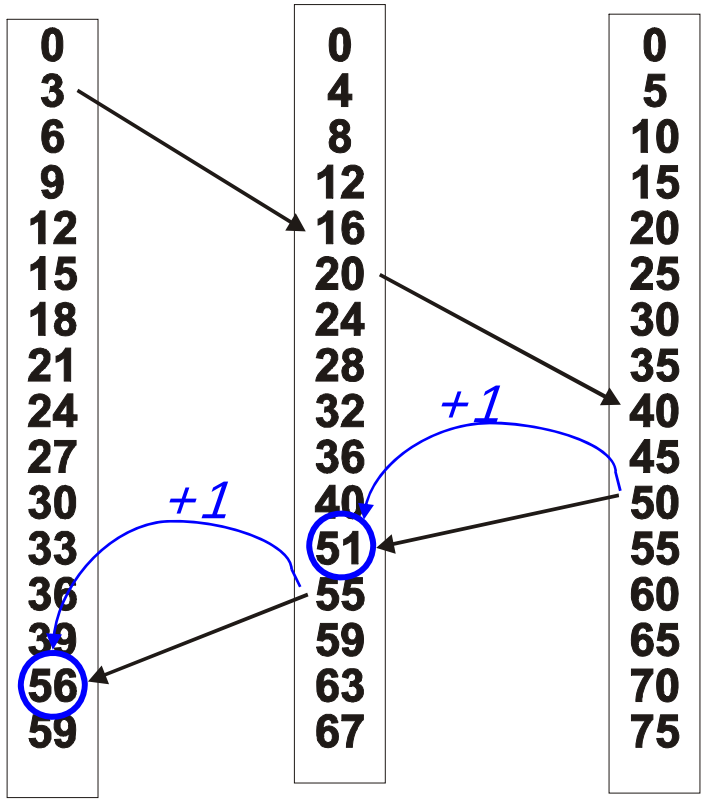
- Útiles para ordenar eventos en ausencia de un reloj común.
  - Cada proceso P mantiene una variable entera LC_P (reloj lógico)
  - Cuando un proceso P genera un evento, $LC_P=LC_P+1$
  - Cuando un proceso envía un mensaje incluye el valor de su reloj
  - Cuando un proceso Q recibe un mensaje m con un valor t:
    - $LC_Q=\max(LC_Q,t)+1$
- El algoritmo asegura:
  - Que si $a \rightarrow b$ entonces $LC_a < LC_b$
  - Pero $LC_a < LC_b$ no implica $a \rightarrow b$ (pueden ser concurrentes)
- Relojes lógicos sólo representan una relación de orden parcial.
- Orden total entre eventos si se añade el número del procesador.

Algoritmo de Lamport

No sincronizado



Sincronizado



María S. Pérez
Fernando Pérez
José María Peña

Relojes de Vectores (Mattern y Fidge)

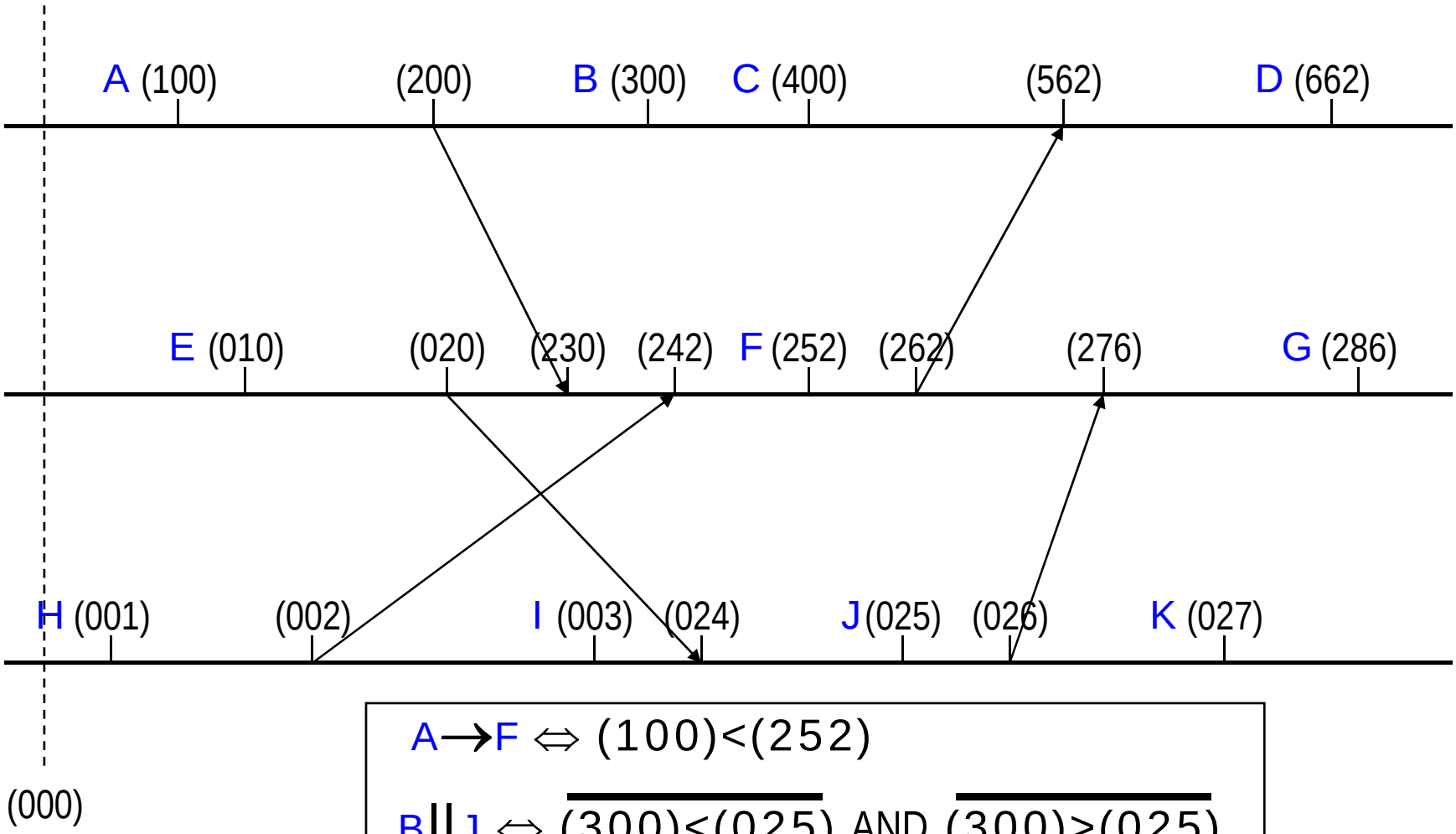
Para evitar los casos en los que $LC_a < LC_b$ no implica $a \rightarrow b$.

Cada reloj es un array V de N elementos siendo N el número de procesadores (nodos) del sistema.

- Inicialmente $V_i[j]=0$ para todo *i*,*j*
- Cuando el proceso *i* genera un evento $V_i[i]=V_i[i]+1$
- Cuando en el nodo *i* se recibe un mensaje del nodo *j* con un vector de tiempo *t* entonces:
  - para todo *k*: $V_i[k]=\max(V_i[k],t[k])$ (operación de mezcla) y
  - $V_i[i]=V_i[i] + 1$

Por medio de este mecanismo siempre es posible evaluar si dos marcas de tiempo tienen o no relación de precedencia.

Algoritmo de Mattern y Fidge



Fernando Pérez
María S. Pérez
José María Peña

Uso de los Relojes Lógicos

La utilización de relojes lógicos (Lamport, Vectoriales o Matriciales) se aplica a:

- Mensajes periódicos de sincronización.
- Campo adicional en los mensajes intercambiados (*piggybacked*).

Por medio de relojes lógicos se pueden resolver:

- Ordenación de eventos (factores de prioridad o equitatividad).
- Detección de **violaciones de causalidad**.
- ***Multicast* causal** (ordenación de mensajes).

Estados Globales

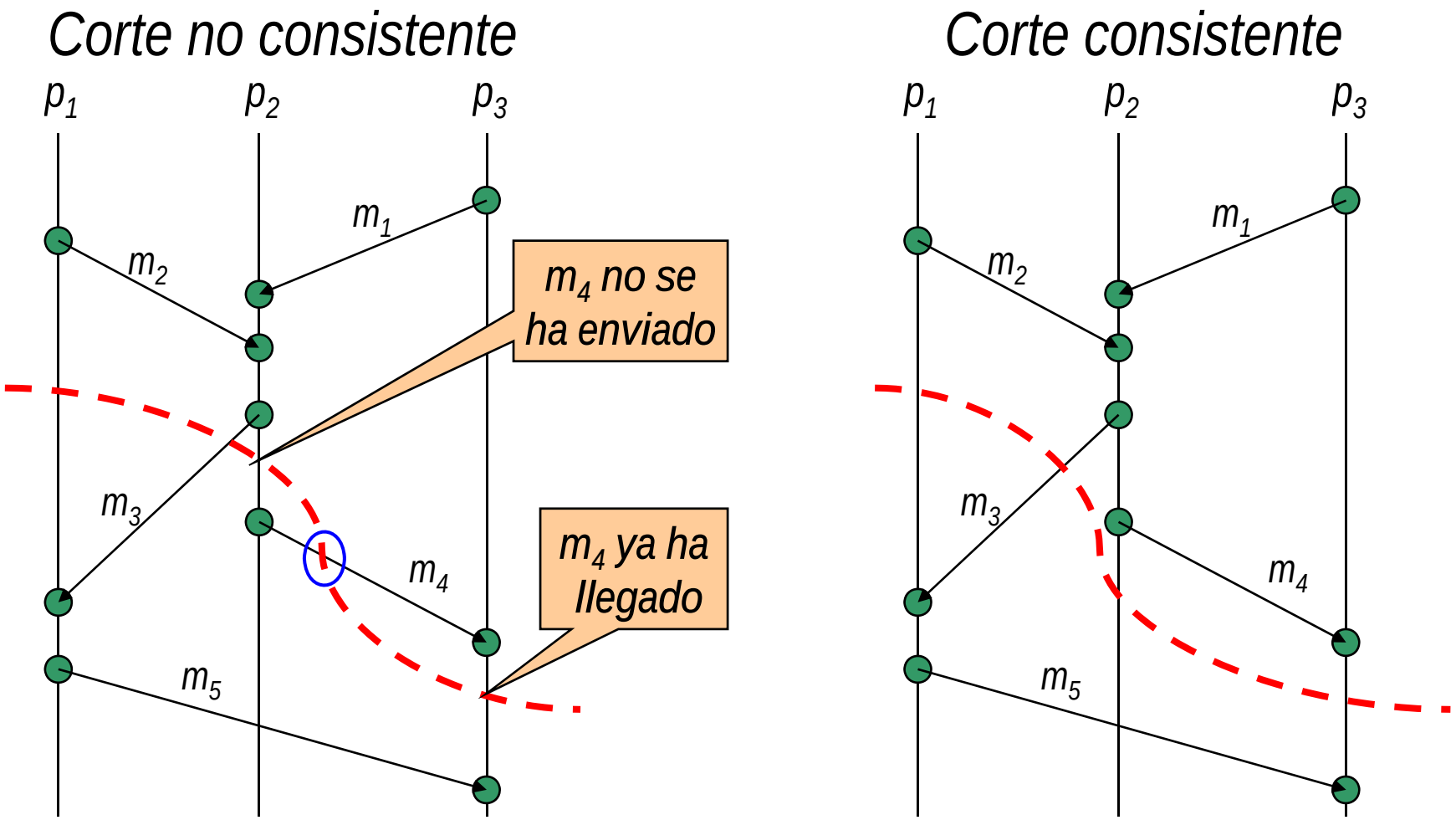
- En un sistema distribuido existen ciertas situaciones que no es posible determinar de forma exacta por falta de un estado global:
  - Recolección de basura: Cuando un objeto deja de ser referenciado por ningún elemento del sistema.
  - Detección de interbloqueos: Condiciones de espera cíclica en grafos de espera (*wait-for graphs*).
  - Detección de estados de terminación: El estado de actividad o espera no es suficiente para determinar la finalización de un proceso.

Snapshots

El análisis de los estados globales de un sistema se realiza por medio de *snapshots*: Agregación del estado local de cada componente así como de los mensajes actualmente en transmisión.

Debido a la imposibilidad de determinar el estado global de todo el sistema en un mismo instante se define una propiedad de consistencia en la evaluación de dicho estado.

Cortes Consistentes



Fernando Pérez
María S. Pérez
José María Peña

Exclusión Mutua

- Algoritmos de Exclusión Mutua

Exclusión Mutua

Mecanismo de coordinación entre varios procesos concurrentes a la hora de acceder a recursos/secciones compartidas.

Las soluciones definidas para estos problemas son:

- Algoritmos centralizados.
- Algoritmos distribuidos.
- Algoritmos basados en marcas de tiempo.

Problemática:

- No existen variables compartidas
- Riesgo de interbloqueos
- Riesgo de inanición

Exclusión Mutua

Funciones básicas de exclusión mutua:

- **`enter()`**: Acceso a la región critica (bloqueo).
- **`operations()`**: Manipulación de los recursos compartidos.
- **`exit()`**: Liberación del recurso (despierta a procesos en espera).

Propiedades:

- Seguridad: Como máximo un proceso puede estar ejecutado en la sección crítica a la vez.
- Vivacidad: Eventualmente se producen entradas y salidas en la sección crítica.
- Ordenación: Los procesadores acceden a la región crítica en base a unos criterios de ordenación (causalidad temporal/Lamport).

Exclusión Mutua

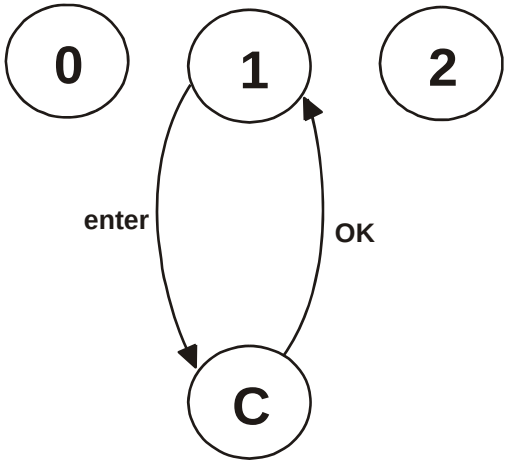
La evaluación de los algoritmos de exclusión mutua se evalúa en base a los siguientes factores:

- Ancho de banda: Proporcional al número de mensajes transmitidos.
- Retardo del cliente: En la ejecución de cada una de las operaciones **`enter()`** y en cada operación **`exit()`**.
- *Throughput* del sistema: Ratio de acceso al recurso por una batería de procesos que lo solicitan. Evalúa el retardo de sincronización entre clientes.
- Tolerancia a fallos: Comportamiento del algoritmo ante diferentes modalidades de fallo.

Exclusión Mutua Centralizado

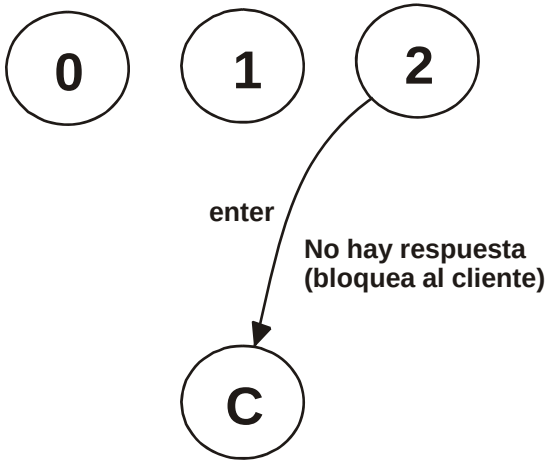
El algoritmo más simple:

- Los clientes solicitan el acceso a un elemento de control que gestiona la cola de peticiones pendientes.
- Tres mensajes: **`enter`**, **`exit`** y **`OK`** .
- No cumple necesariamente la propiedad de ordenación.



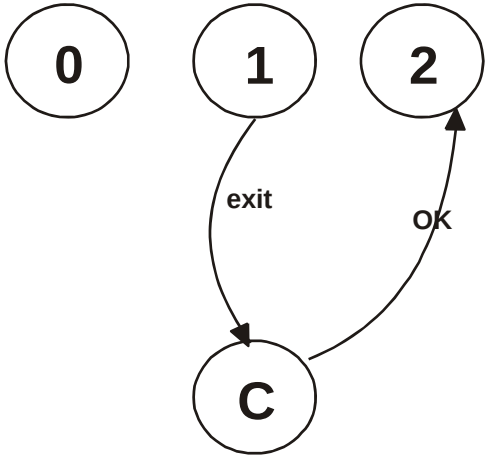
Cola de Espera

1



Cola de Espera

1 2



Cola de Espera

2

Exclusión Mutua Centralizado

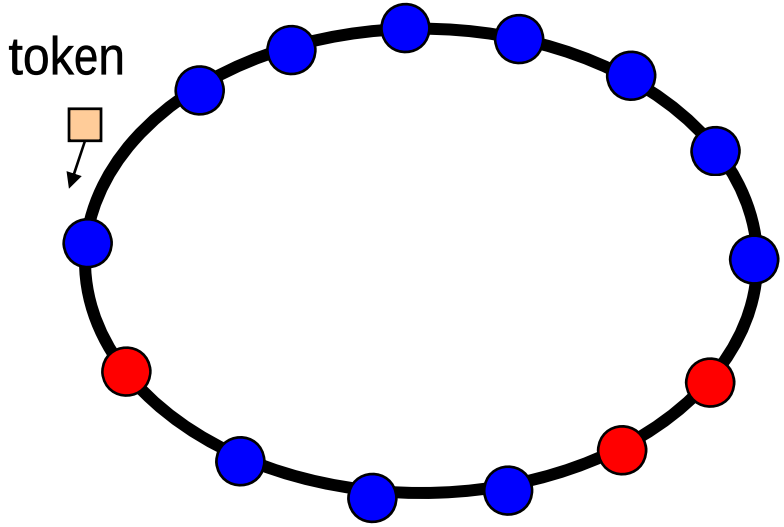
Rendimiento:

- Ancho de banda:
  - El acceso al recurso implica dos mensajes (aunque el recurso esté libre).
- Retardo del cliente:
  - **`enter()`**: El retardo de transmisión de los dos mensajes.
  - **`exit()`**: Con comunicación asíncrona no implica retraso en cliente.
- *Throughput* del sistema:
  - La finalización de un acceso a la región critica implica un mensaje de salida y un **`OK`** al siguiente proceso en espera.
- Tolerancia a fallos:
  - La caída del elemento de control es crítica (alg. de elección).
  - La caída de los clientes o la perdida de mensajes se puede solucionar por medio de temporizadores.

Exclusión Mutua Distribuida

Algoritmos distribuido de paso de testigo:

- Se distribuyen los elementos en un anillo lógico.
- Se circula un *token* que permite el acceso a la región critica.
- El *token* se libera al abandonar la región.
- No cumple la propiedad de ordenación



Exclusión Mutua Distribuida

Rendimiento:

- Ancho de banda:
  - El algoritmo consume ancho banda de forma continua.
- Retardo del cliente:
  - La entrada en la sección critica requiere de 0 a N mensajes.
  - La salida sólo implica un mensaje.
- *Throughput* del sistema:
  - La entrada del siguiente proceso tras la salida del que ocupa la región critica implica de 1 a N mensajes.
- Tolerancia a fallos:
  - Perdida del token: Detección y regeneración
  - Caída de un elemento del anillo: Reconfiguración del anillo.

Exclusión Mutua con Relojes Lógicos

Algoritmo de Ricart y Agrawala: Usa relojes lógicos y broadcast

Pasos:

- Un proceso que quiere entrar en sección crítica (SC) envía mensaje de solicitud a todos los procesos.
- Cuando un proceso recibe un mensaje
  - Si receptor no está en SC ni quiere entrar envía OK al emisor
  - Si receptor ya está en SC no responde
  - Si receptor desea entrar, mira marca de tiempo del mensaje:
    - Si menor que marca tiempo de su mensaje de solicitud: envía OK.
    - En caso contrario será él el que entre.
  - Cuando un proceso recibe todos (N-1) los mensajes puede entrar.
- Garantiza todas las propiedades incluida ordenación

Exclusión Mutua con Relojes Lógicos

- Los procesos 1 y 3 quieren acceder a la sección crítica.
- Los relojes lógicos son respectivamente 23 y 21.

Exclusión Mutua con Relojes Lógicos

Rendimiento:

- Ancho de banda:
  - El protocolo consume 2(N-1) mensajes. N-1 para la petición y N-1 respuestas. Si existen comunicación multicast sólo N mensajes.
- Retardo del cliente:
  - La entrada en la sección critica requiere de N-1 mensajes.
  - La salida no implica ningún mensaje.
- *Throughput* del sistema:
  - Si dos procesos compiten por el acceso a la sección critica ambos habrán recibido N-2 respuestas. El de menor reloj tendrá la respuesta del otro. Al salir éste el siguiente se indicará con sólo 1 mensaje.
- Tolerancia a fallos:
  - Retardo de respuesta elevado o perdida de mensajes: Se reduce por medio de mensajes NO-OK (asentimientos negativos).

Algoritmos de Votación

Algoritmo de Maekawa: Algoritmos de votación.

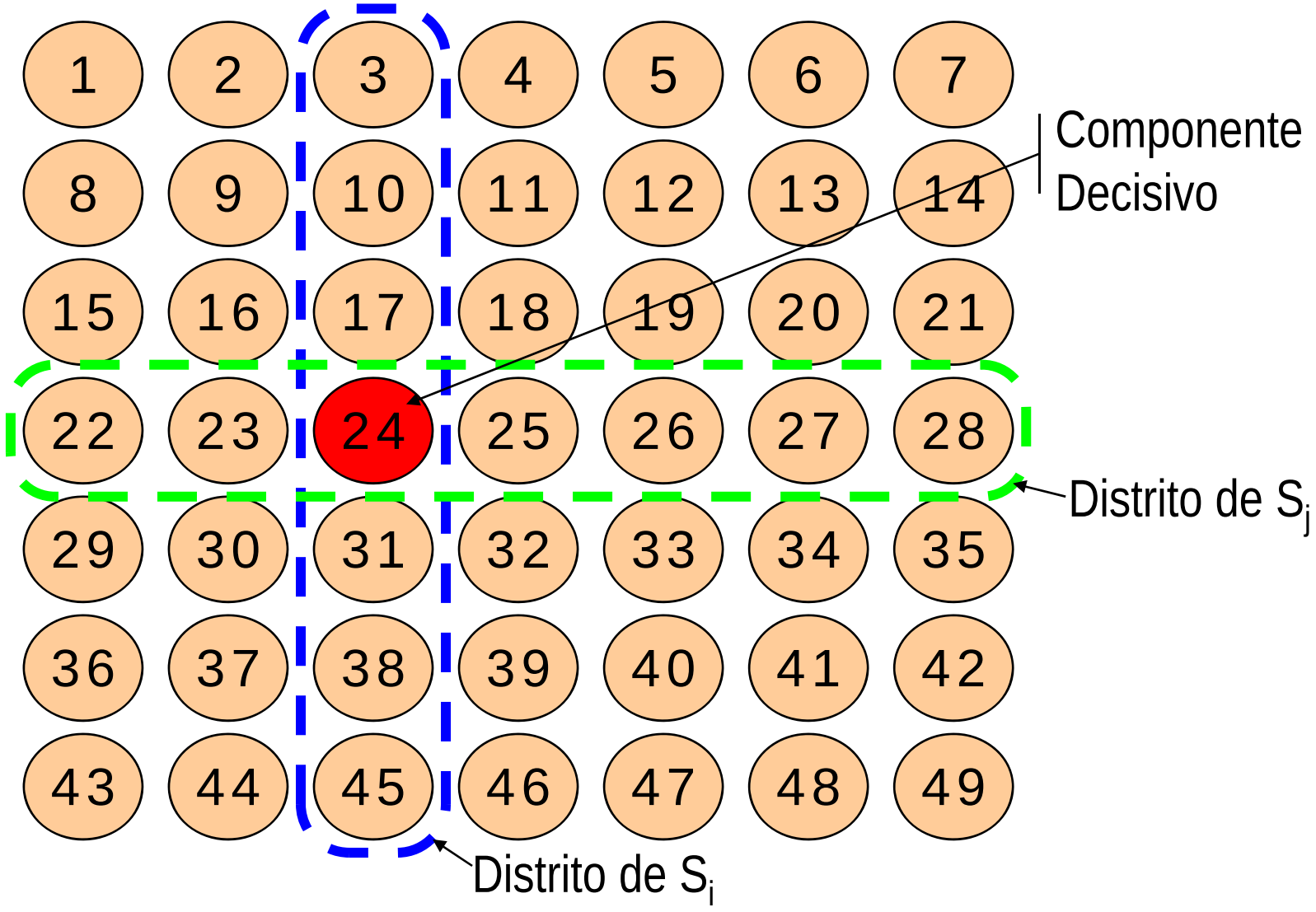
Análogo al algoritmo de relojes lógicos pero reduce el número de mensajes:

- El procesador elegido es aquel que obtiene la mitad más 1 votos.
- Cada procesador es consultado sobre la elección emitiendo un voto.
- Para reducir el número de mensajes cada uno de los procesadores que intentan acceder a la sección critica tiene un *distrito* (S_i), tal que:

$$S_i \cap S_j \neq \varnothing \text{ para todo } 1 \leq i,j \leq N$$

- De esta forma sólo se necesitan $\sqrt{N}$ mensajes.

Algoritmos de Votación



Fernando Pérez
María S. Pérez
José María Peña

Otras Variantes

Para solucionar los problemas de interbloqueo de los algoritmos de acceso a regiones críticas en base a mecanismos de votación tradicionales (Maekawa) existen otras alternativas, por ejemplo:

- Saunders: Algoritmos de votación con marcas de tiempo:
  - Previene problemas de interbloqueo entre 3 o más procesos.
  - Permite retirar votos si la nueva petición tiene una marca de tiempo menor.

Sistemas Operativos Distribuidos

Problemas de Consenso

- Algoritmos de Elección
- Consenso & Acuerdo

Algoritmos de Elección

Son algoritmos diseñados para problemas en los cuales uno de los procesos ha de realizar una tarea especial:

- Elección de un coordinador.

Estos mecanismos se activan también cuando el coordinador ha fallado.

Objetivo: **Elección única**

Algoritmo del Matón

Objetivo

- Elige al procesador “vivo” con un ID mayor

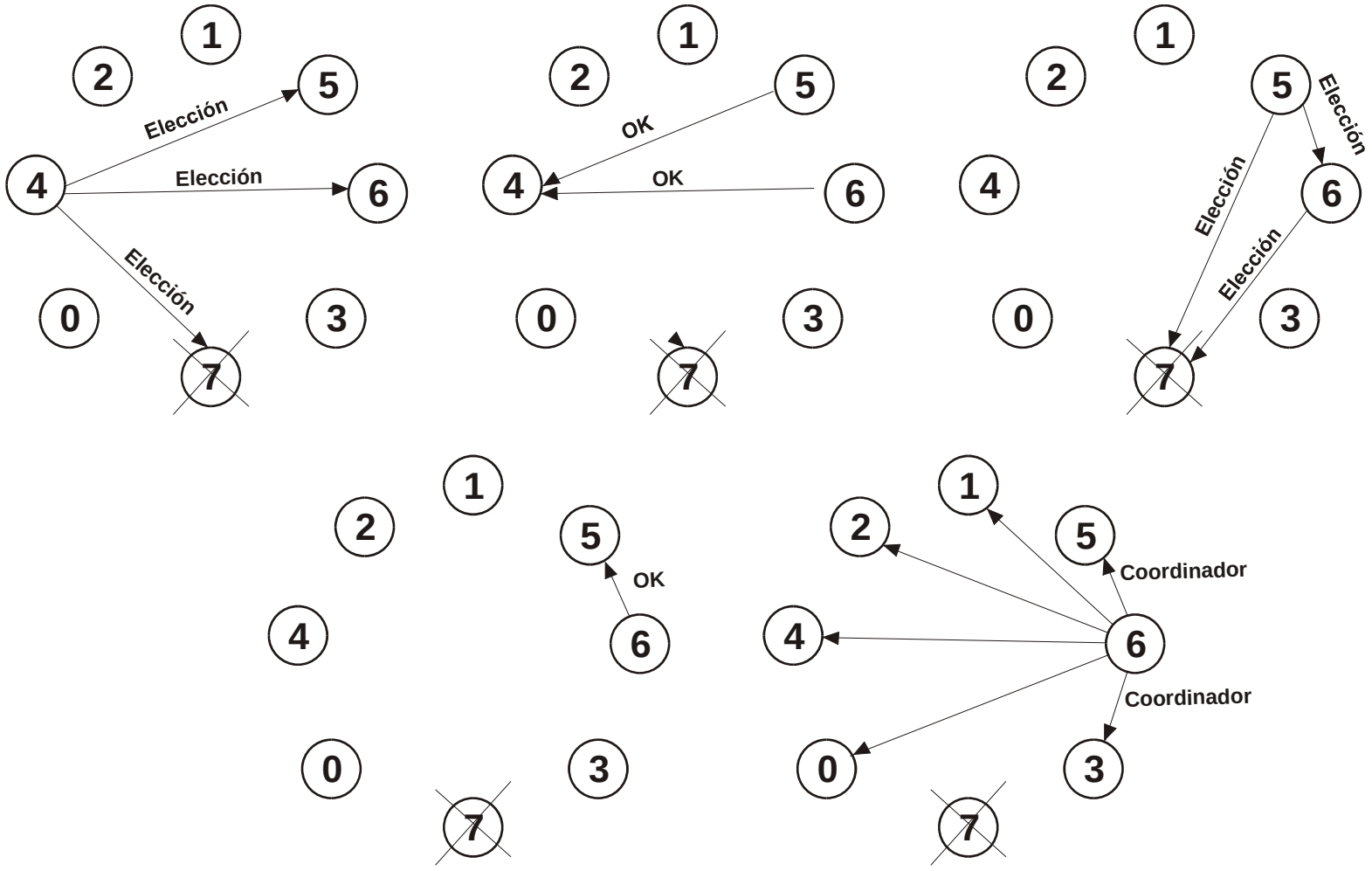
Proceso ve que el coordinador no responde. Inicia una elección:

- Envía mensaje de ELECCIÓN a procesos con ID mayor
- Si ninguno responde: Se hace nuevo coordinador
  - Manda mensajes COORDINADOR a procesadores con ID menor
- Si alguno responde con mensaje OK abandona la elección

Si procesador recibe ELECCIÓN:

- Si tiene ID menor, entonces responde OK e inicia elección (si todavía no lo había hecho).

Algoritmo del Matón



Algoritmos en Anillo

Sobre un anillo lógico de procesos se emite un mensaje de elección.

Si un proceso recibe el mensaje:

- Si el ID del mensaje es menor que el suyo, lo retransmite con el suyo.
- Si es mayor lo retransmite como tal.
- Si es igual, entonces no retransmite y es el coordinador.

Algoritmo de Invitación

Problemática de los algoritmos anteriores:

- Se basan en *timeouts*: Retrasos de transmisión pueden causar la elección de múltiples lideres.
- La perdida de conexión entre dos grupos de procesadores puede aislar permanentemente los procesadores.

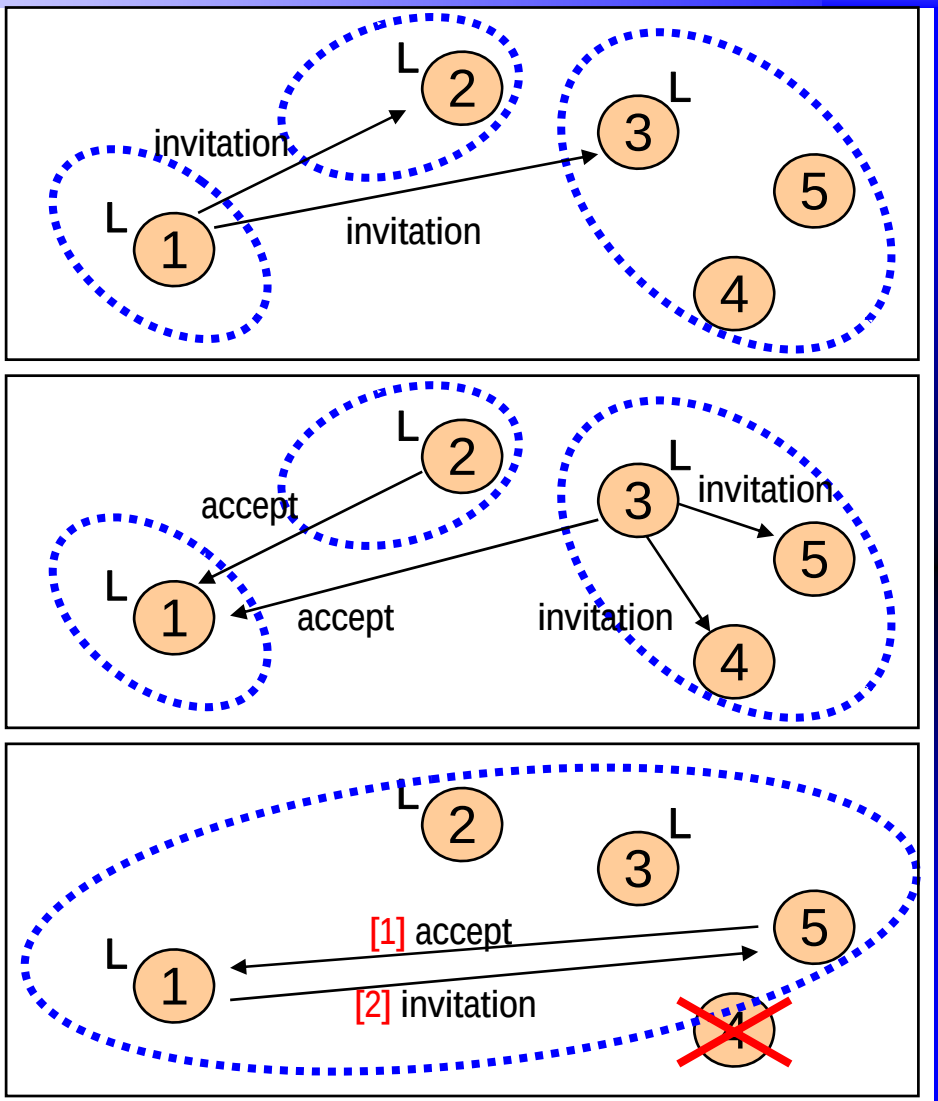
Algoritmo de Invitación, característica:

- Definición de grupos de procesadores con líder único.
- Detección y agregación de grupos.
- Reconocimiento por parte del líder de los miembros del grupo.

Algoritmo de Invitación

Pasos:

- Si un procesador detecta la perdida del líder, entonces se declara líder y forma su propio grupo.
- Periódicamente el líder de cada grupo busca otros líderes de otros grupos.
- Dos grupos se unen por medio de mensajes de aceptación:
  - Como respuesta a mensajes de invitación.
  - De forma explícita.



Problemas de Consenso

Presentes en tareas en las cuales varios procesos deben ponerse de acuerdo en una valor u operación a realizar.

- Problema de consenso general: Los procesos intercambian candidatos y cada elemento elige el mayoritario. Debe ser común.
- Consistencia interactiva: Cada proceso aplica un valor diferente y se debe identificar el vector de valores usado por cada proceso.
- Problema de los generales bizantinos: Por ejemplo, ¿Qué pasa si un proceso transmite valores diferentes a distintos procesos?

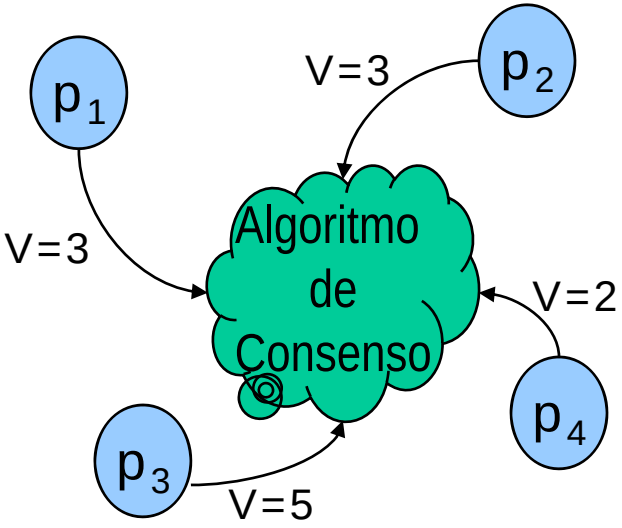
Los procesos del sistema pueden encontrase en dos estados:

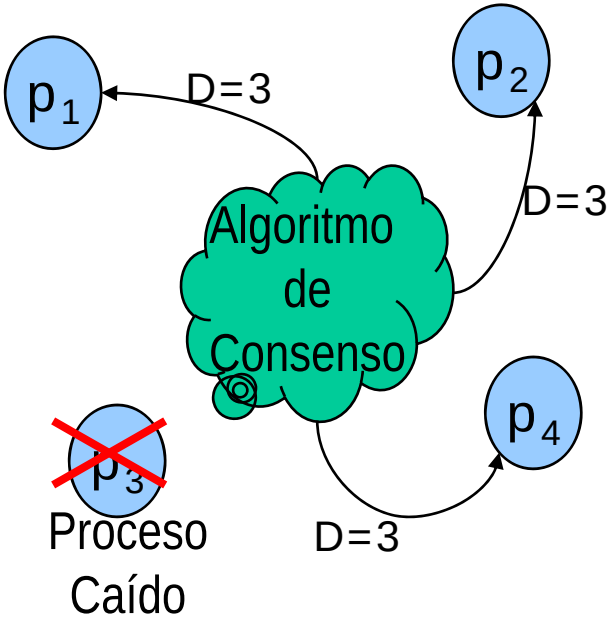
- Correcto: estado válido.
- Incorrecto: procesador caído o funcionando anómalamente.

Problema de Consenso General

Condiciones:

- Terminación: Cada proceso correcto fija un valor.
- Acuerdo: El valor decidido es igual para todos los procesos correctos.
- Integridad: Si todos los procesos correctos eligen el mismo valor entonces dicho valor será el válido.

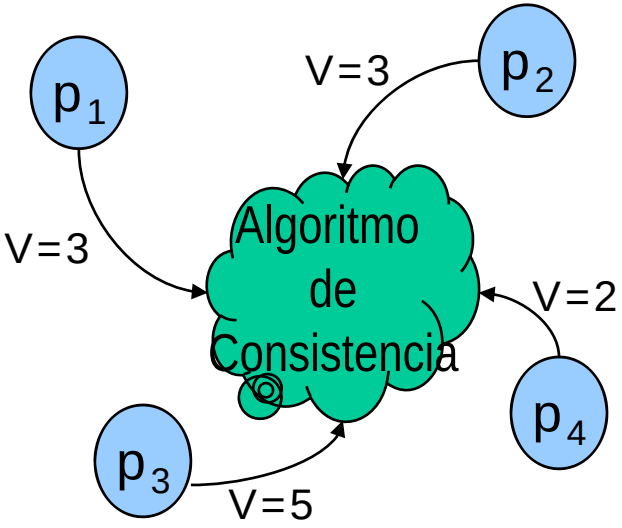


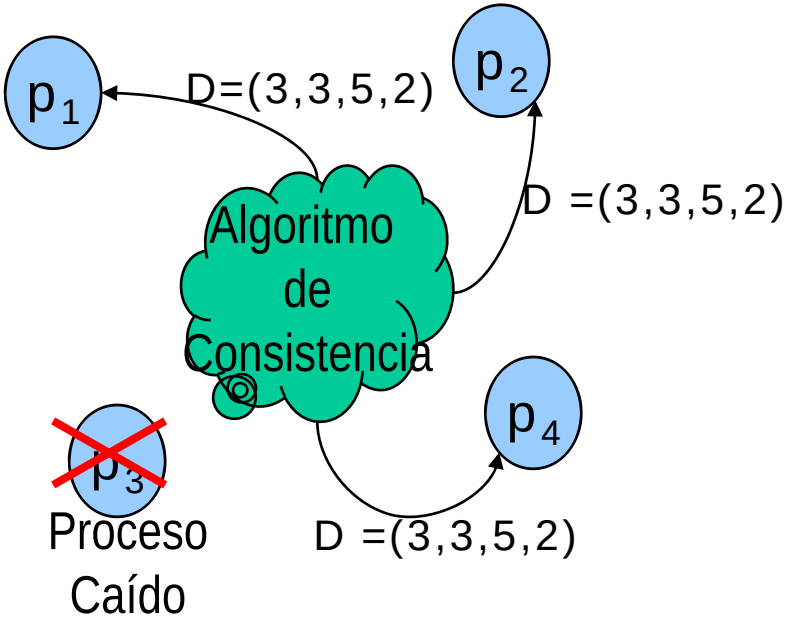


Consistencia Interactiva

Condiciones:

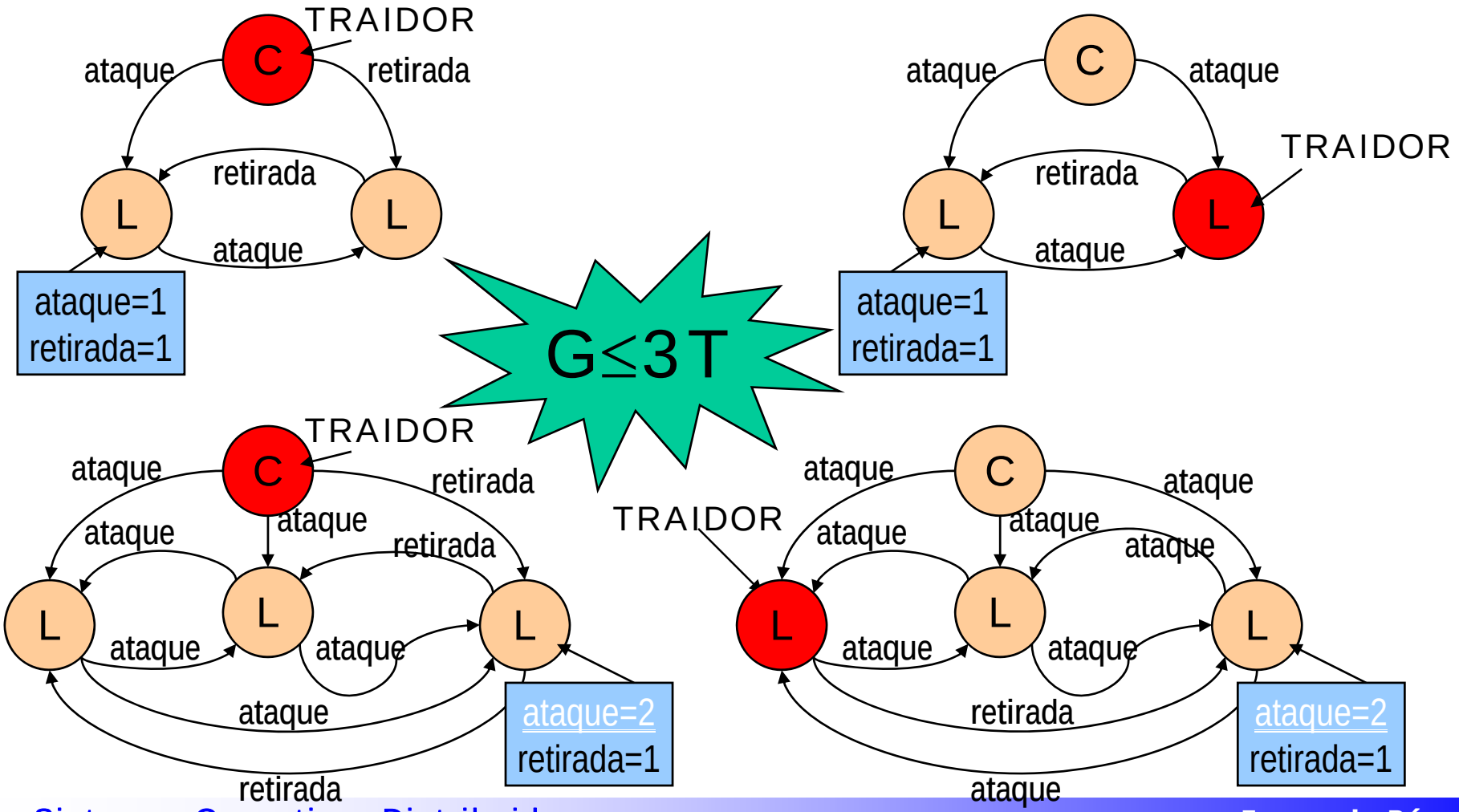
- Terminación: Cada proceso correcto fija un vector valores.
- Acuerdo: El vector decidido es igual para todos los procesos correctos
- Integridad: La posición *i*-esima del vector se corresponde con el valor propuesto por el proceso p_i





Generales Bizantinos

Error bizantino: Un proceso genera valores de forma arbitraria.



Sistemas Operativos Distribuidos

Transacciones Concurrentes

- Operaciones Atómicas

Transacciones

Conjuntos de operaciones englobadas dentro de un bloque cuya ejecución es completa.

Cumplen las propiedades ACID:

- *Atomicity* (Atomicidad): La transacción se realiza completa o no se realiza nada.
- *Consistency* (Consistencia): Los estados anterior y posterior a la transacción son estados estables (consistentes).
- *Isolation* (Aislamiento): Los estados intermedios de la transacción son sólo visibles dentro de la propia transacción.
- *Durability* (Durabilidad): Las modificaciones realizadas por una transacción completada se mantienen.

Transacciones

La gestión de transacciones admite tres operaciones:

- **`beginTransaction()`**: Comienza un bloque de operaciones que corresponden a una transacción.
- **`endTransaction()`**: Concluye el bloque de operaciones que conforman la transacción. Todas las operaciones se completan.
- **`abortTransaction()`**: En cualquier punto se aborta la transacción y se regresa al estado anterior al comienzo de transacción.

- Otra condición para abortar la transacción es debido a errores.

Transacciones Concurrentes

Se dispone de tres cuentas corrientes A, B y C con saldos €100, €200 y €300 respectivamente.

Las operaciones sobre una cuenta son:

- **`balance=A.getBalance()`**: Obtener el saldo.
- **`A.setBalance(balance)`**: Modificar el saldo.
- **`A.withdraw(amount)`**: Retirar una cierta cantidad.
- **`A.deposit(amount)`**: Deposita una cierta cantidad.

Transacciones Concurrentes

Actualización perdida:

`bal=B.getBalance()`	`bal=B.getBalance()`
`B.setBalance(bal*1.1)`	`B.setBalance(bal*1.1)`
`A.withdraw(bal*0.1)`	`C.withdraw(bal*0.1)`
`bal=B.getBalance()` → €200	
	`bal=B.getBalance()` → €200
	`B.setBalance(bal*1.1)` → €220
`B.setBalance(bal*1.1)` → €220	
`A.withdraw(bal*0.1)` → €80	
	`C.withdraw(bal*0.1)` → €280

Transacciones Concurrentes

Recuperaciones inconsistentes:

A.withdraw(100)	**<suma de saldos>**
B.deposit(100)	
A.withdraw(100) → €0	
	tot=A.getBalance() → €0
	tot+=B.getBalance() → €300
	tot+=C.getBalance() → €500
B.deposit(100) → €400	

Transacciones Concurrentes

La problemática de las transacciones concurrentes se debe a:

- Operaciones de lectura y escritura simultánea.
- Varias operaciones de escritura simultánea.

La alternativa es la reordenación de las operaciones a lo que se denominan operaciones secuenciales equivalentes.

Los métodos de resolución aplicados son:

- Cerrojos (*Locks*): Aplicados sobre los objetos afectados.
- Control de concurrencia optimista: Las acciones se realizan sin verificación hasta que se llega a un **`commit`**.
- Ordenación en base a marcas de tiempo.

Cerrojos

Cada objeto compartido por dos procesos concurrentes tiene asociado un cerrojo.

- El cerrojo se cierra al comenzar el uso del objeto.
- El cerrojo se libera al concluir la operación.

El uso de cerrojos puede ser definido a diferentes niveles del objeto a controlar (niveles de granularidad).

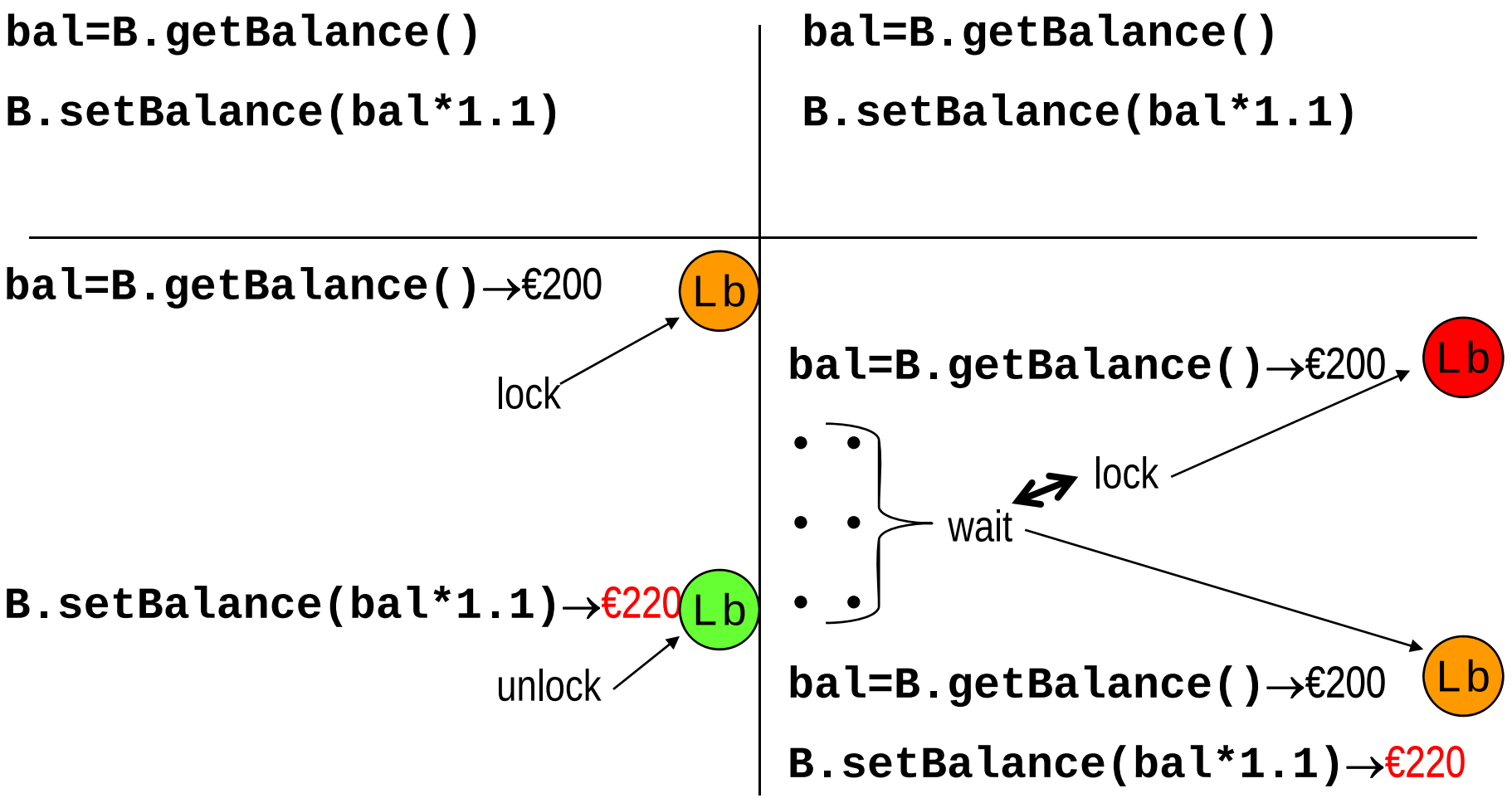
Modelos de cerrojo:

- Lectura
- Escritura

Los cerrojos son susceptibles de sufrir interbloqueos.

Cerrojos

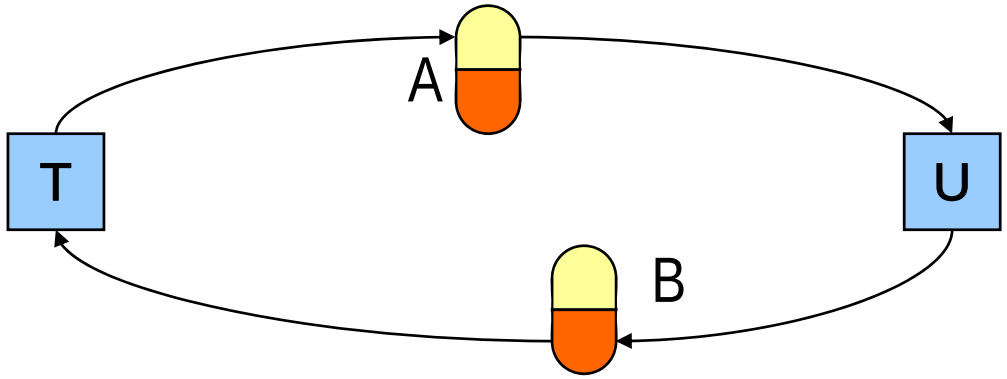
Actualización perdida:



Interbloqueos

Un interbloqueo se produce cuando varios procesos compiten por cerrojos de forma cíclica:

- Detección de interbloqueos: Grafos de espera.



- Prevención de interbloques: Cierre de todos los cerrojos de una transacción antes de comenzar (Poco eficiente).
- Resolución de interbloqueos: Lo más habitual es por medio de *Timeouts* y abortando la transacción propietaria del cerrojo.

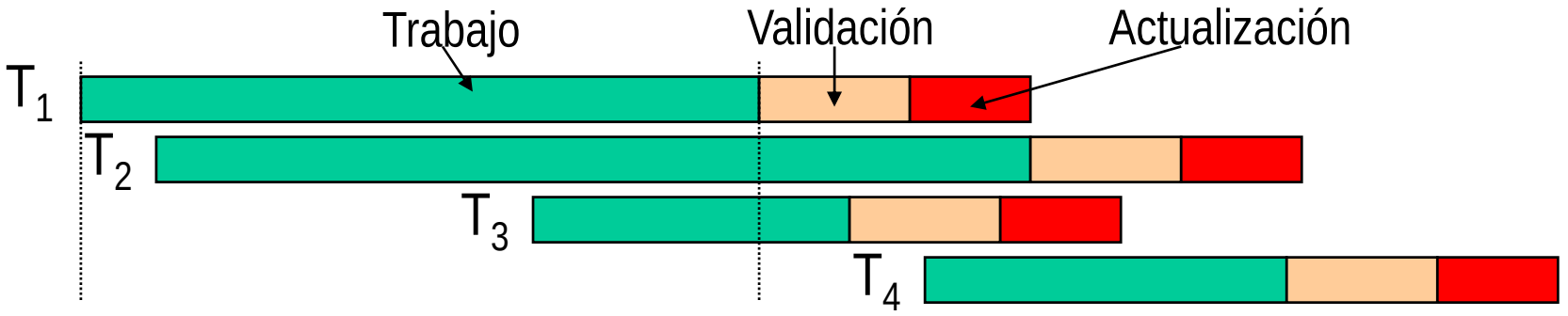
Control de Concurrencia Optimista

Muy pocas operaciones concurrentes tiene conflictos entre sí.

División de una operación en:

- Fase de trabajo: Los objetos usados por la transacción pueden ser copiados en "*valores tentativos*". Una lectura tome este valor si existe sino el último valor validado. Las escrituras se realizan siempre sobre los "*valores tentativos*".
- Fase de validación: Al cerrar la transacción se verifica colisiones con otras transacciones.
- Fase de actualización: Los "*valores tentativos*" son copiados como valores validados.

Control de Concurrencia Optimista



Validación:

- Validación hacia atrás: Se anula una transacción si otra transacción activa escribe un valor que ésta lee.
- Validación hacia delante: Todos los procesos de escritura realizados anulan a las transacciones que los leían.

Problemática:

- Si la fase de validación falla la transacción se aborta y se reinicia. Puede causar inanición.