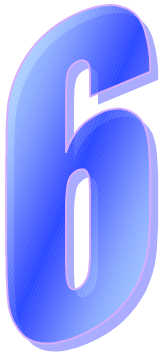
Sistemas Operativos Distribuidos



Sincronización

Sincronización en Sistemas Distribuidos

Más compleja que en los centralizados

- Características de algoritmos distribuidos:
  - La información relevante se distribuye entre varias máquinas.
  - Debe evitarse un punto único de fallo.
  - No existe un reloj común.
- Problemas a considerar:
  - Tiempo y estados globales.
  - Exclusión mutua.
  - Algoritmos de elección. Problemas de consenso.
  - Operaciones atómicas distribuidas: Transacciones

Sistemas Operativos Distribuidos

Tiempo y estados

- Relojes distribuidos
- Relojes lógicos
- Estados globales

Sincronización de relojes físicos

Relojes hardware de un sistema distribuido no están sincronizados.

Necesidad de una sincronización para:

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

Concepto de sincronización:

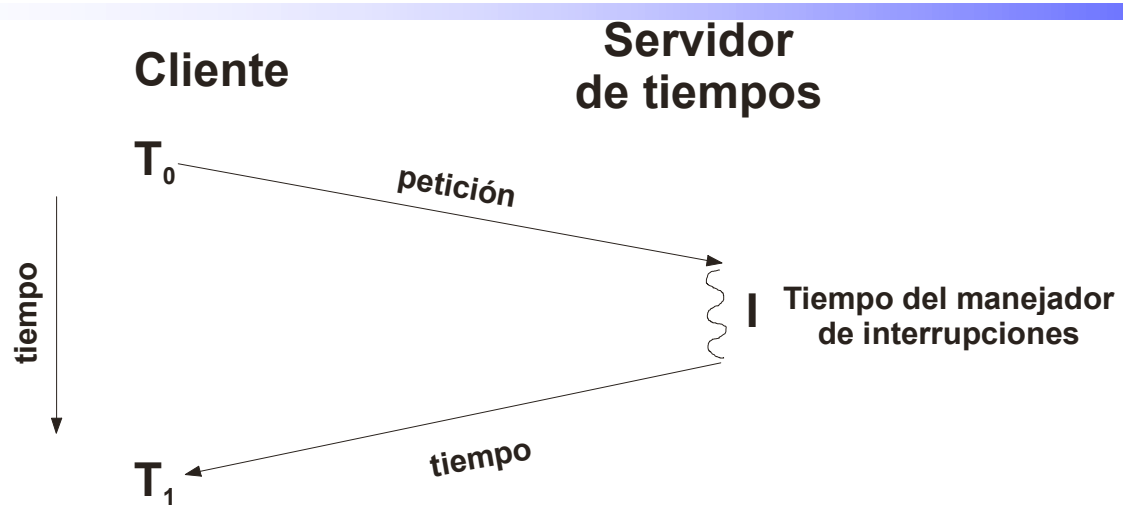
- Mantener relojes sincronizados entre sí.
- Mantener relojes sincronizados con la realidad.

UTC: *Universal Coordinated Time*

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

Fernando Pérez Costoya
José María Peña Sánchez

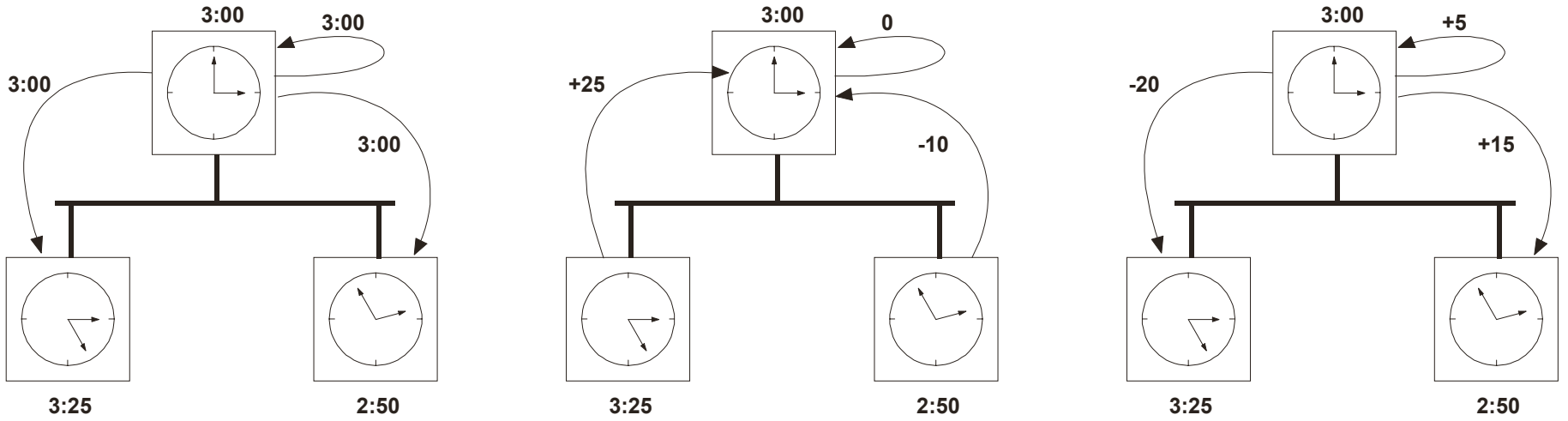
Algoritmo de Cristian



- 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)



Fernando Pérez Costoya
José María Peña Sánchez

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 ataques a la seguridad.

Organización:

- Jerarquía de servidores en diferentes *estratos*.
  - Servidores primarios → conectados directamente a UTC
  - Servidores secundarios → conectados a primarios
  - Servidores finales → "hojas" del árbol (máquinas de usuarios)
- Los fallos se solventan por medio de ajustes en la jerarquía.

Protocolo de tiempo de red

Sincronización entre servidores de la jerarquía:

- Modo *multicast*: Para redes LAN. Servidores transmiten tiempo por la red de forma periódica.
- Modo de llamada a procedimiento: Servidor acepta peticiones de otras máquinas. Similar al algoritmo de Cristian. Se promedia el retardo de transmisión.
- Modo simétrico: Los dos elementos intercambian mensajes de sincronización que ajustan los relojes.

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)

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

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

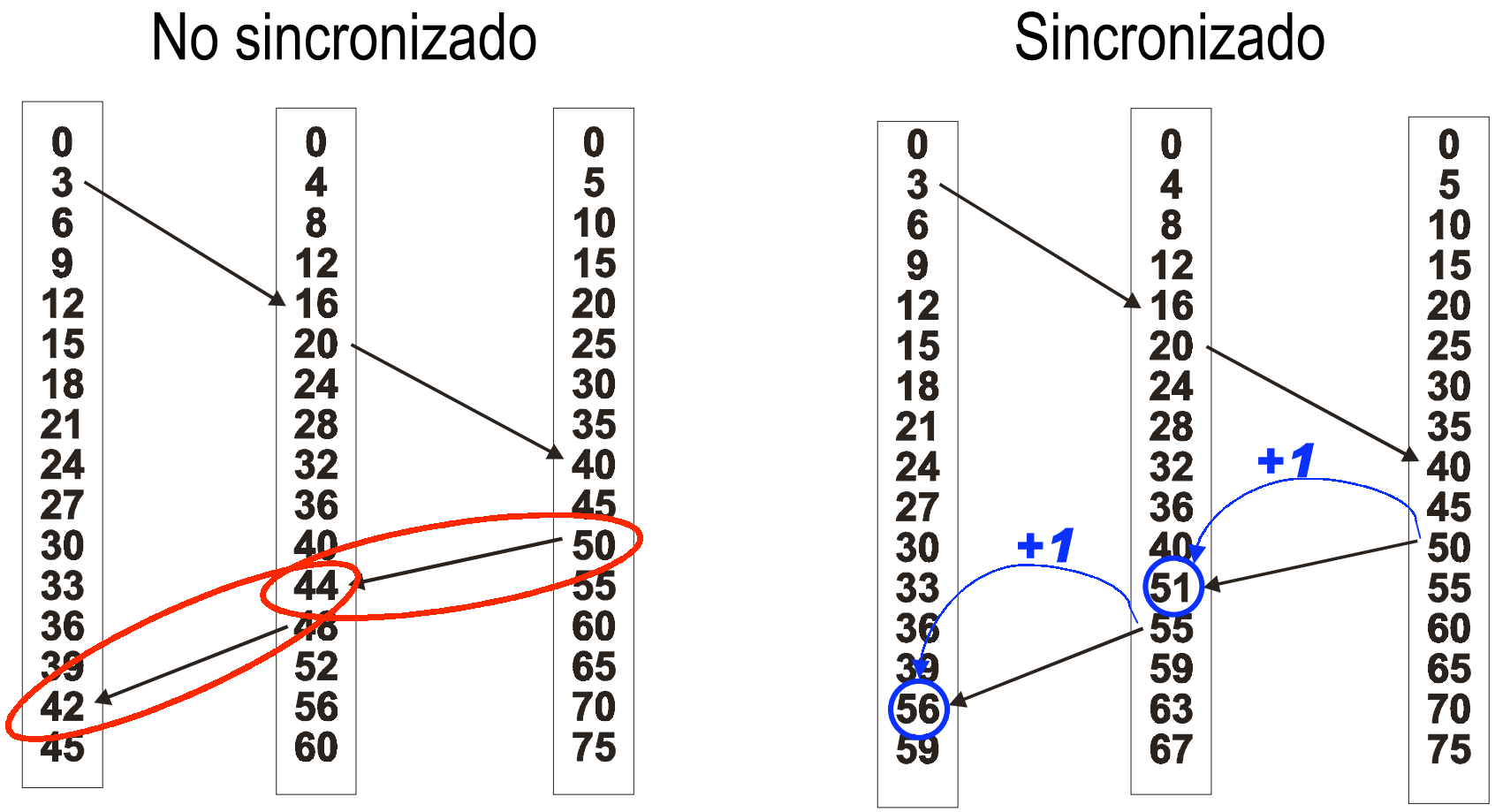
Fernando Pérez Costoya
José María Peña Sánchez

Relojes lógicos (Algoritmo de Lamport)

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

Fernando Pérez Costoya
José María Peña Sánchez

Algoritmo de Lamport



Fernando Pérez Costoya
José María Peña Sánchez

Relojes de vectores (Mattern y Fidge)

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

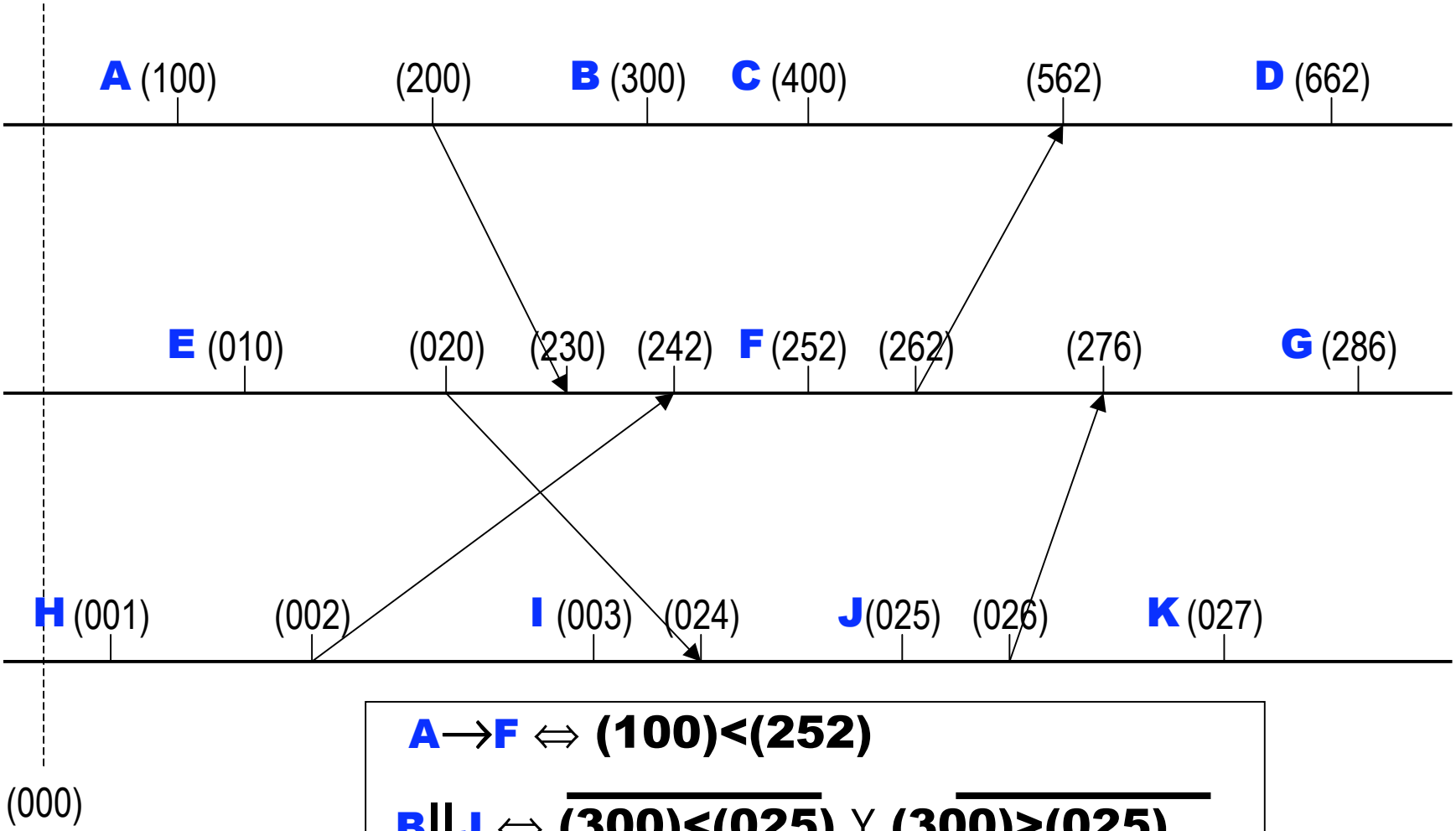
Cada reloj es un array **V** de **N** elementos siendo **N** el número de procesadores (nodos) del sistema.

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

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

Fernando Pérez Costoya
José María Peña Sánchez

Algoritmo de Mattern y Fidge



Fernando Pérez Costoya
José María Peña Sánchez

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.

Fernando Pérez Costoya
José María Peña Sánchez

Instantáneas y cortes coherentes

Análisis de estados globales de un sistema mediante *instantáneas*: Agregación del estado local de cada componente así como de los mensajes actualmente en transmisión.

- Representa un *corte* en la ejecución de los distintos componentes.

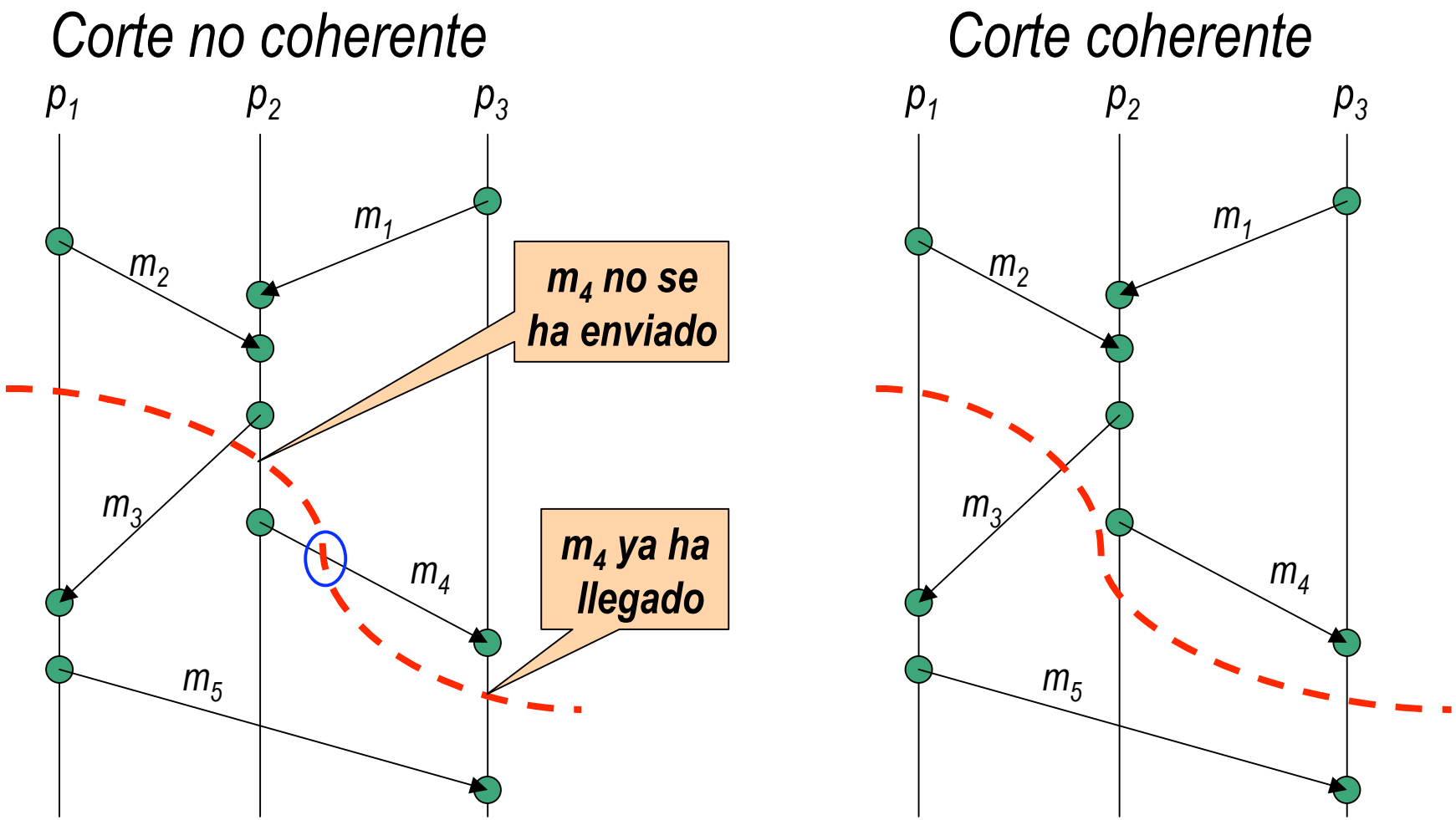
Debido a la imposibilidad de determinar el estado global en un mismo instante el estado obtenido puede ser incoherente.

- *Estado global/corte coherente* → mantiene causalidad.

Algoritmos de instantáneas distribuidas (Chandy y Lamport,1985)

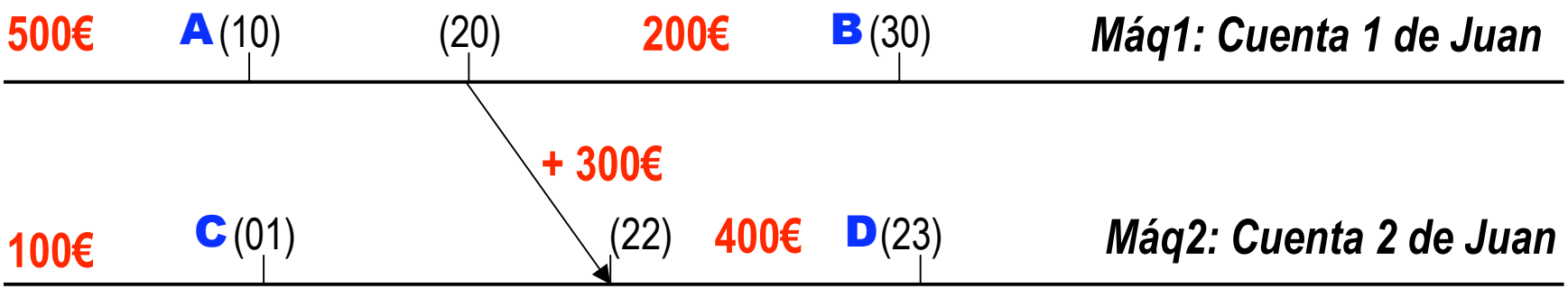
- Fuera del alcance de la exposición

Cortes coherentes



Fernando Pérez Costoya
José María Peña Sánchez

Ejemplo de cálculo del estado global



Muestra en **A-C**: **A||C** ⇔ $\overline{\textbf{(10)<(01)}}$ Y $\overline{\textbf{(10)>(01)}}$

Estado global coherente; Total: 500€ + 100€ = 600€

Muestra en **B-C**: **B||C** ⇔ $\overline{\textbf{(30)<(01)}}$ Y $\overline{\textbf{(30)>(01)}}$

Estado global coherente; Total: 200€ + 300€ + 100€ = 600€

Muestra en **B-D**: **B||D** ⇔ $\overline{\textbf{(30)<(23)}}$ Y $\overline{\textbf{(30)>(23)}}$

Estado global coherente; Total: 200€ + 400€ = 600€

Muestra en **A-D**: **A→D** ⇔ **(10)<(23)**

Estado global incoherente ; Total: ~~500€ + 400€ = 900€~~

¿Total en cuentas?

↓

Cálculo de estado global

Fernando Pérez Costoya
José María Peña Sánchez

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

Fernando Pérez Costoya
José María Peña Sánchez

Exclusión mutua

Funciones básicas de exclusión mutua:

- `enter()`: Acceso a la región critica (bloqueo).
- `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
  - acceden en orden real de petición
  - acceden teniendo en cuenta causalidad

Fernando Pérez Costoya
José María Peña Sánchez

Exclusión mutua

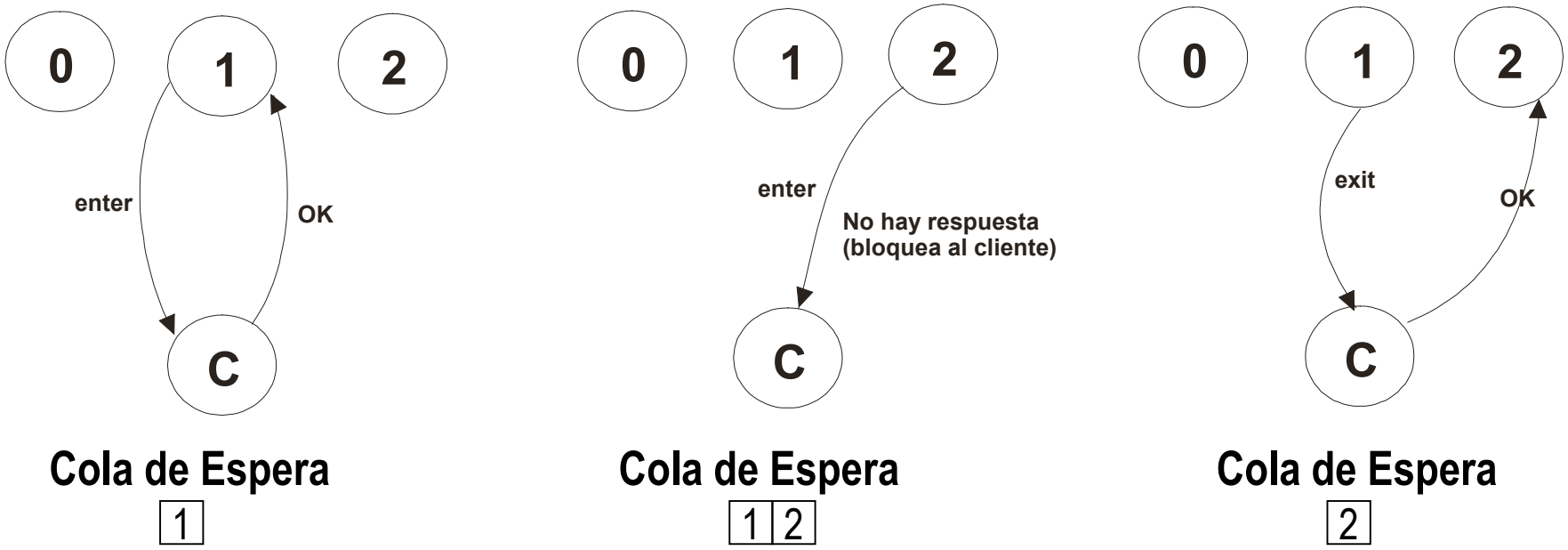
Evaluación de algoritmos de exclusión mutua basada en:

- Ancho de banda consumido: Proporcional al número de mensajes transmitidos en el protocolo.
- Retardo del cliente: En la ejecución de cada **`enter()`** y **`exit()`**.
- Retardo de sincronización entre clientes: entre un cliente que deja la sección crítica y otro que va a entrar en ella.
- Tolerancia a fallos: Comportamiento del algoritmo ante diferentes modalidades de fallo.

Exclusión mutua centralizada

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.



Fernando Pérez Costoya
José María Peña Sánchez

Exclusión mutua centralizada

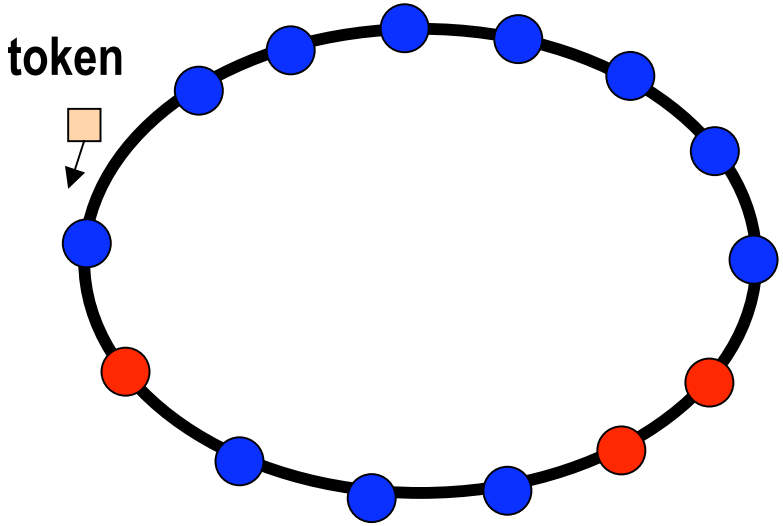
Rendimiento:

- Ancho de banda consumido:
  - El acceso a la sección crítica implica tres mensajes (aunque el recurso esté libre): **`enter`**, **`OK`** y **`exit`**
- Retardo del cliente:
  - **`enter()`**: El retardo de transmisión de dos mensajes.
  - **`exit()`**: Con comunicación asíncrona no implica retraso en cliente.
- Retardo de sincronización entre clientes:
  - 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 pérdida 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



Fernando Pérez Costoya
José María Peña Sánchez

Exclusión mutua distribuida

Rendimiento:

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

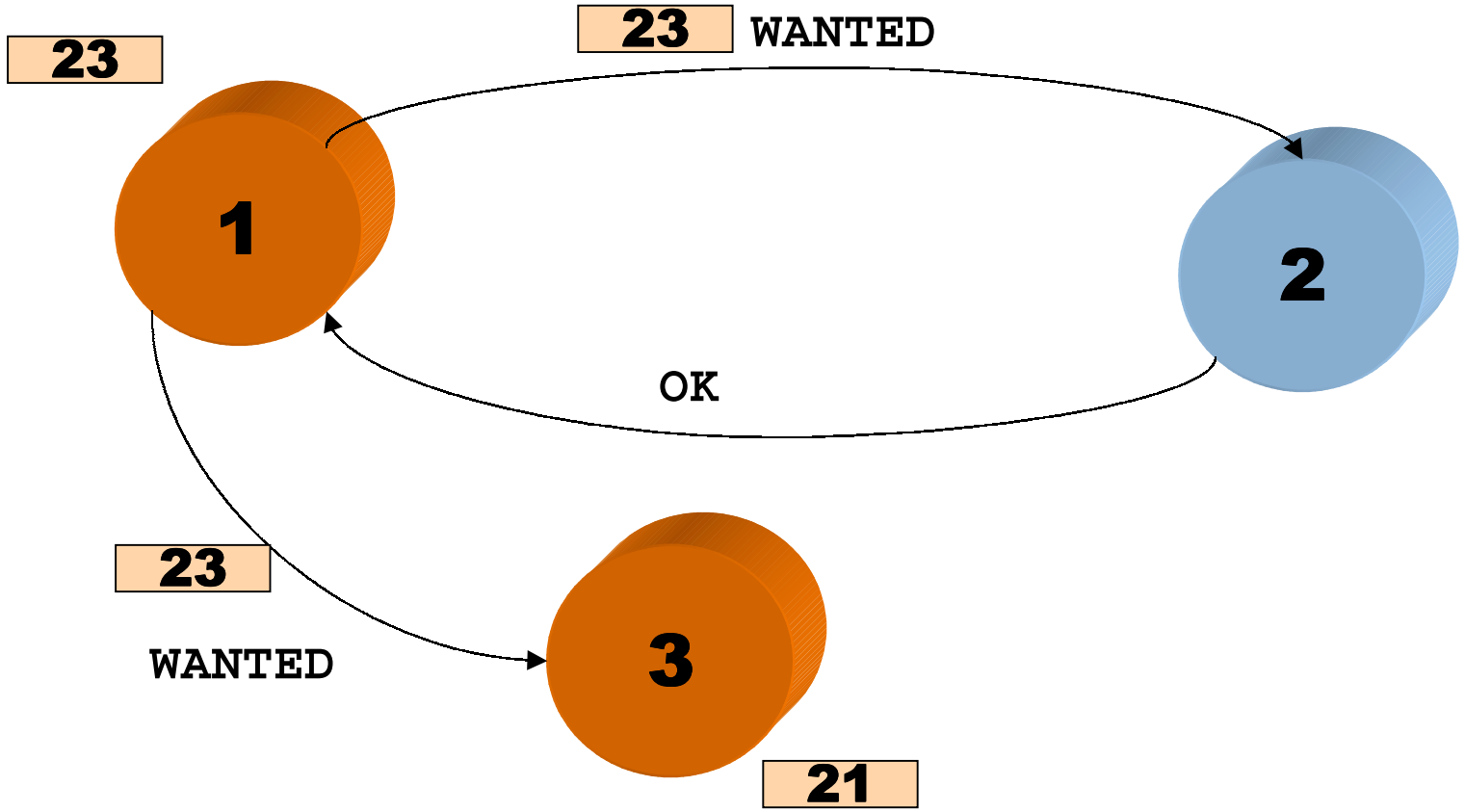
Exclusión mutua con relojes lógicos

Algoritmo de Ricart y Agrawala: Usa relojes lógicos

Pasos:

- Un proceso que quiere entrar en sección crítica (SC_i) envía mensaje de solicitud a todos los procesos.
- Cuando un proceso recibe un mensaje de solicitud de entrada a SC_i
  - Si receptor no está en SC_i ni quiere entrar envía OK al emisor
  - Si receptor ya está en SC_i no responde (pero encola solicitud)
  - 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 no responde (pero encola solicitud).
  - Cuando un proceso recibe todos (N-1) los mensajes puede entrar.
- Cuando un proceso termina SC_i
  - envía OK a todas las solicitudes encoladas
- 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.

Importancia de r. lógicos en el algoritmo

Relojes lógicos permiten “desempatar” peticiones simultáneas, pero además aseguran corrección del algoritmo.

Suponga P1 en sección crítica y que P2 ha recibido N-2 OK, P3 quiere entrar ahora: ¿Cuántos OK recibirá?

- Si r. lógico de petición de P3 es menor que el de petición P2
  - puede obtener N-2 OK y, por tanto, al salir P1 ¡entran P2 y P3!
- Absurdo: Si P3 ha enviado previamente OK a P2 el reloj lógico de su solicitud posterior tiene que ser mayor
  - Hay causalidad entre las solicitudes

Exclusión mutua con relojes lógicos

Rendimiento:

- Ancho de banda consumido :
  - El protocolo consume 2(N-1) mensajes. N-1 para la petición y N-1 respuestas. Si existe comunicación multicast sólo N mensajes.
- Retardo del cliente:
  - La entrada en la sección critica requiere N-1 mensajes.
  - La salida requiere tantos mensajes como procesos esperando.
- Retardo de sincronización entre clientes:
  - Si dos procesos compiten por el acceso a la sección crítica ambos habrán recibido N-2 respuestas. El de menor reloj tendrá la respuesta del otro. Al salir éste al siguiente se lo indicará con sólo 1 mensaje.
- Tolerancia a fallos:
  - Retardo de respuesta elevado o pérdida de mensajes: Se reduce por medio de mensajes NO-OK (asentimientos negativos).

Coordinación y acuerdo

- Algoritmos de elección
- Problemas de consenso

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**

Fernando Pérez Costoya
José María Peña Sánchez

Algoritmo del matón

Objetivo

- Elige al procesador “vivo” con un ID mayor

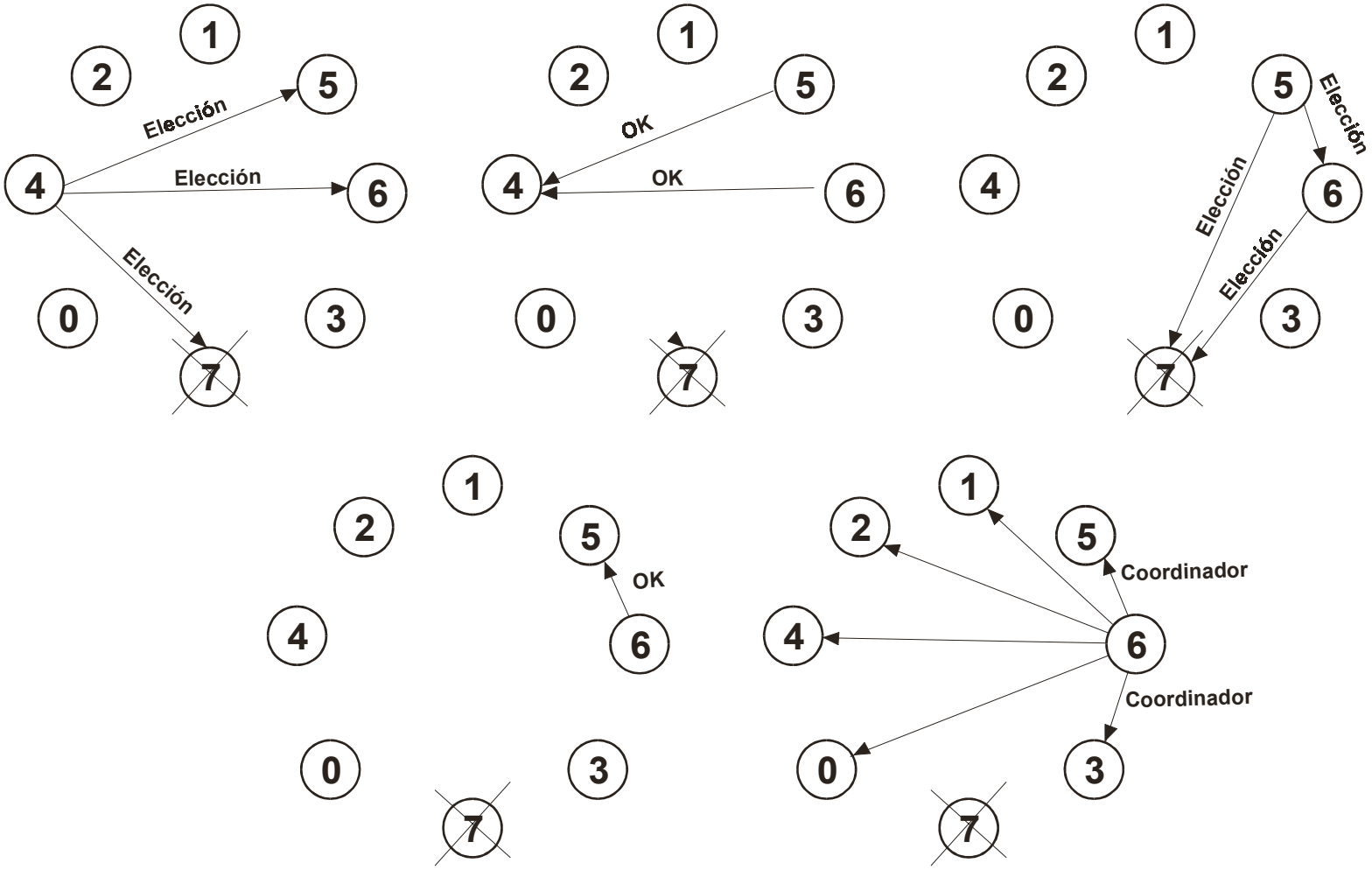
Proceso ve que el coordinador no responde. Inicia una elección:

- 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



Fernando Pérez Costoya
José María Peña Sánchez

Algoritmos en anillo

Sobre un anillo lógico de procesos uno o más procesos que detectan el fallo del coordinador emiten 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 pero cambia su estado a “candidato” de manera que si llega otro mensaje con ID menor lo retransmite.
- Si es mayor lo retransmite como tal.
- Si es igual, entonces no lo retransmite y es el coordinador, por lo manda por el anillo un mensaje indicándolo.

Problemas de consenso

Conjunto de procesos deben ponerse de acuerdo en una valor u operación a realizar, incluso ante fallos en el sistema.

Los elementos del sistema pueden sufrir fallos:

- Por omisión: el elemento no realiza la función encomendada
  - Procesador se para (*fail-stop* o *crash*); Red pierde un mensaje
- Bizantinos: el elemento tiene un comportamiento arbitrario
  - Procesador ejecuta acciones no previstas ni programadas
  - Red cambia contenido de mensajes o entrega mensajes no existentes

Estudio queda fuera del alcance de la exposición:

- Sólo se enuncian dos problemas clásicos (de inspiración militar)

El problema de los 2 ejércitos

Sobre la imposibilidad de acuerdo ante fallos en la comunicación

Descripción del problema:

- El ejército rojo (con 5000 soldados) acampado en el valle.
- 2 divisiones del ejército azul (3000 soldados cada una) en colinas.
- Si coordinan su ataque vencen, sino derrotadas → Consenso
- Sólo se pueden comunicar usando un canal inseguro
  - ejército rojo puede capturar al mensajero (fallo por omisión)
- Comandante de división 1 (Com1) envía mensaje a división 2
  - ¿Atacamos mañana a las 9?
- Comandante de división 2 (Com2) envía mensaje de OK a división 1
- Com1 se da cuenta de que Com2 no sabe si ha llegado OK
  - le manda un mensaje para confirmarlo
  - pero al recibirlo Com2 piensa lo mismo y tiene que enviar otro mensaje
- No hay posible consenso

El problema de los generales bizantinos

Comunicación sin fallos pero procesadores con fallos bizantinos

Descripción del problema:

- El ejército rojo acampado en el valle.
- N divisiones del ejército azul en colinas
  - 1 división con general comandante, N-1 con generales lugartenientes
  - pero hay M traidores (incluso puede serlo el comandante)
- Comandante da orden de ataque o retirada
  - Si es traidor, a unos le dice una cosa y a otros la contraria
- Consenso: todos los generales leales deben tomar la misma decisión
  - Incluso aunque el comandante sea traidor
- Para ello, hablan todos entre sí varias veces hasta tomar decisión

Algoritmo de Lamport (fuera del alcance de la presentación)

- Hay solución si $N \geq 3M + 1$

Fernando Pérez Costoya
José María Peña Sánchez

Transacciones Distribuidas

- Operaciones Atómicas
- *Two-Phase Commit*

Transacciones

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

Cumplen las propiedades **ACID**:

- ***A****tomicity* (Atomicidad): La transacción se realiza completa o no se realiza nada.
- ***C****onsistency* (Consistencia): Los estados anterior y posterior a la transacción son estados estables (consistentes).
- ***I****solation* (Aislamiento): No interferencia entre transacciones. Estados intermedios de transacción visibles sólo dentro de la misma.
  - *Seriability*: Resultado de varias transacciones ejecutando en paralelo equivalente al generado por algún orden de ejecución secuencial de las mismas.
- ***D****urability* (Perdurabilidad): Las modificaciones realizadas por una transacción completada se mantienen.

Transacciones

En SD transacción se refiere a cjto. de operaciones de un cliente

La gestión de transacciones admite tres operaciones:

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

Puede haber transacciones anidadas (jerarquía):

- Transacción dividida en subtransacciones que pueden terminar o abortar de forma independiente
  - aunque si aborta una transacción "padre" deben abortarse hijas

Transacciones concurrentes

La problemática de las transacciones concurrentes se debe a:

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

Debe conseguirse *seriability*

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 basada en 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).

Modos del cerrojo:

- Lectura
- Escritura

Los cerrojos son susceptibles de sufrir **interbloqueos**.

Control de concurrencia optimista

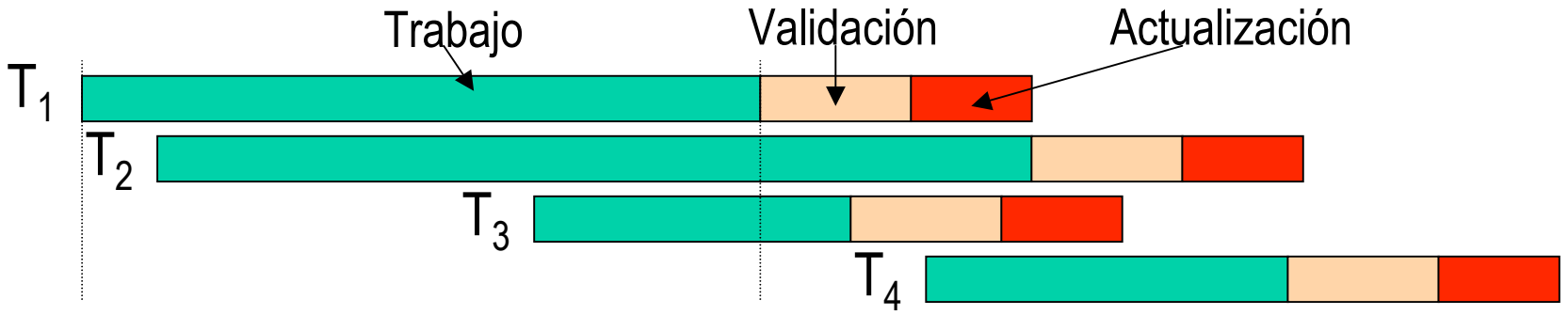
Muy pocas operaciones concurrentes tiene conflictos entre sí.

División de una operación en:

- Fase de trabajo: Los objetos usados por la transacción pueden ser copiados en "*valores tentativos*". Una lectura toma 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.

Fernando Pérez Costoya
José María Peña Sánchez

Control de concurrencia optimista



Validación:

- Validación hacia atrás: Se anula una transacción si otra transacción solapada pero ya comprometida escribió un valor que ésta lee.
- Validación hacia delante: Comprobar si transacciones activas leen valores que esta transacción escribió. Si es así, se puede abortar esta transacción o la activa.

Problemática:

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

Transacciones distribuidas

Transacciones que afectan de forma atómica a objetos residentes en varios servidores.

- Pueden estar anidadas

Modo de operación

- Un servidor actúa de coordinador (recibe `beginTransaction`)
- Cualquiera puede abortar transacción
  - Lo comunica a coordinador que lo propaga a todos para que la aborten
- Cuando cliente solicita *commit*:
  - Si todos los implicados están de acuerdo, se "compromete"
  - Si algún procesador quiere abortarla o está caído, se "aborta"
  - Necesidad de una fase de decisión

Protocolo clásico *two-phase-commit* (2PC)

- Requiere *almacenamiento estable*: ("nunca" pierde la infor.)
  - Uso de dos discos: se escribe primero en uno y luego en otro

Two-Phase Commit

Mensajes intercambiados en *two-phase commit*:

- **`vote-request`**: Coordinador consulta a los servidores.
- **`vote-commit:`** Servidor vota afirmativamente.
- **`vote-abort:`** Servidor vota negativamente.
- **`global-commit:`** Coordinador indica a servidores que la operación se completa.
- **`global-abort:`** Coordinador indica a servidores que la operación se aborta.
- *ACK*: Servidor notifica que ha terminado para que coordinador pueda conocer cuándo puede eliminar información sobre la transacción

Two-Phase Commit

Coordinador:

Escribir **`vote-request`** en mem. estable.

Mandar a subordinados. **`vote-request`**

Recoger las respuestas

Si todos *ok* =>

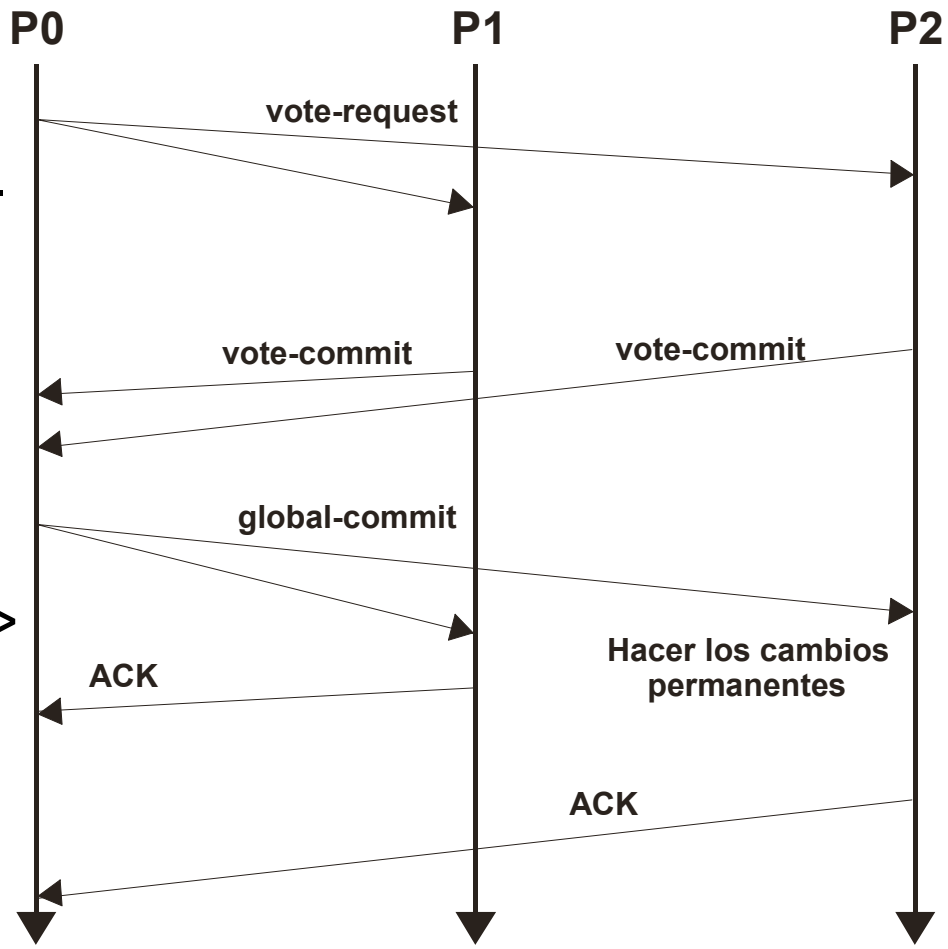
`global-commit`

Si alguno *abort* o no responde =>

`global-abort`

Escribir resolución en m. estable

Mandar resolución

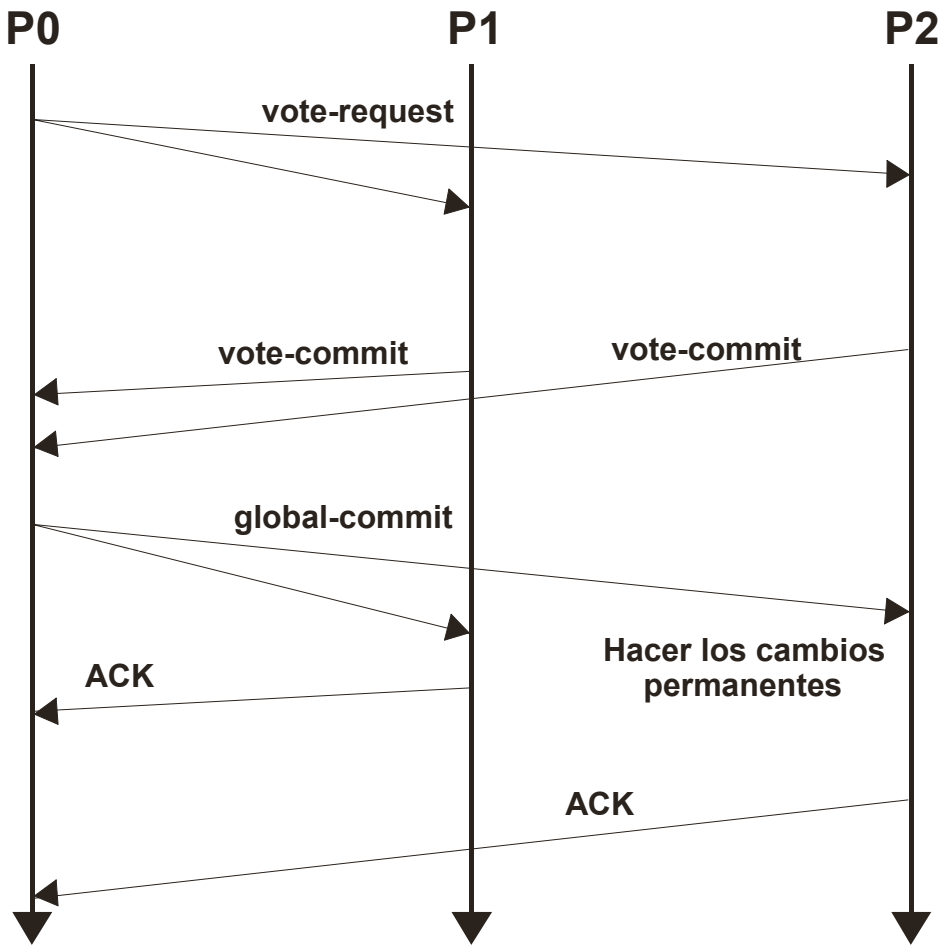


Fernando Pérez Costoya
José María Peña Sánchez

Two-Phase Commit

Subordinados:

- Recibir **`vote-request`**
- Decidir respuesta y grabar en mem.estable
- Mandar respuesta
- Recibir resolución
- Escribir resolución en mem. estable
- Llevar a cabo resolución:
- **`global-commit`** => hacer cambios permanentes
- **`global-abort`** => deshacer cambios



Fernando Pérez Costoya
José María Peña Sánchez

Fallos en 2PC

- Buena tolerancia a fallos
  - Recuperación después de caída: consulta mem. estable
- Recuperación después de caída de un subordinado:
  - Si encuentra en mem. estable la respuesta pero no la resolución:
    - pregunta a coordinador cuál ha sido la resolución
  - Si encuentra en mem. estable la resolución:
    - la lleva a cabo
- Recuperación después de caída y rearranque de coordinador:
  - Si encuentra en m. estable **`vote-request`** pero no resolución:
    - manda a los subordinados mensajes **`vote-request`**
  - Si encuentra en mem. estable la resolución:
    - manda a los subordinados mensajes con la resolución

Fallos en 2PC

- Recuperación sin rearranque de coordinador:
  - Pueden tomarse decisiones aunque coordinador caído
    - subordinado S contacta con otros subordinados
  - Subordinado S ha votado y espera resultado de coordinador. Pasado un plazo contacta con otro subordinado Q para conocer su estado:
    - Si Q ha recibido resultado: P hace lo mismo.
    - Si Q a la espera: se contacta con los otros subordinados.
    - Si todos a la espera: bloqueo hasta que arranque coordinador
- Para evitarlo *Three-Phase Commit*
  - Poco usado
  - Queda fuera de la exposición