Capítulo 5
Tiempo y estados globales

Sistemas Distribuidos

Universidad Nacional de Asunción

Facultad Politécnica

Ingeniería Informática

Ing. Fernando Mancía

Métodos en los que los que los relojes pueden ser sincronizados aproximadamente, utilizando paso de mensajes.

Relojes lógicos y relojes vectoriales.

Algoritmos cuyo propósito es capturar los estados globales de los sistemas distribuidos cuando se ejecutan.

Un sistema distribuido consta de una colección Ώ de *N* procesos i, 2, .,. *N*.

Cada proceso se ejecuta en un único procesador, y los procesadores no comparten memoria.

Cada proceso p_i en Ώ tiene un estado s_i que, en general, se transforma a medida que se ejecuta. El estado del proceso incluye los valores de todas sus variables.

El estado puede incluir también los valores de cualquiera de los objetos en el entorno de su sistema operativo local que lo afecte, como archivos.

A medida que cada proceso *pi* se ejecuta, toma una serie de acciones, cada una de las cuales es un mensaje *Envía o Recibe*, o una operación que transforma el estado *pi* que cambia uno o más valores de si.

Evento: ocurrencia de una única acción que un proceso realiza a medida que se ejecuta, una acción de comunicación o una acción de transformación del estado.

La secuencia de sucesos en un único proceso pi puede ser colocada en orden único, que se indica con la relación -> i entre eventos.

$e \rightarrow_i e'$ si y sólo si el evento *e* ocurre antes del *e'* en pi.

Esta ordenación está bien definida, sea o no el proceso multihilo, puesto que hemos supuesto que el proceso se ejecuta en un procesador único.

Historia del proceso pi: serie de eventos que tienen lugar en él , ordenados de la forma que hemos descrito con la relación -> i :

historia (pi)= hi = $< e_i^o , e_i^1 , \ldots >$

Relojes: Los computadores pueden disponer de su propio reloj físico. Estos relojes son dispositivos electrónicos que cuentan las oscilaciones que ocurren en un cristal a una frecuencia definida, y que normalmente dividen esta cuenta y almacenan el resultado en un registro contador.

El sistema operativo lee el valor del reloj hardware $H_i(t)$ del nodo, lo escala y añade una compensación para producir un reloj software

$C_i(t) = \alpha H_i(t) + B$ que mide aproximadamente el tiempo real, físico t para el proceso Pi .

Los relojes de los computadores, como los otros, tienden a no estar en perfecto acuerdo.

Sesgo: diferencia instantánea entre las lecturas de dos relojes cualesquiera.

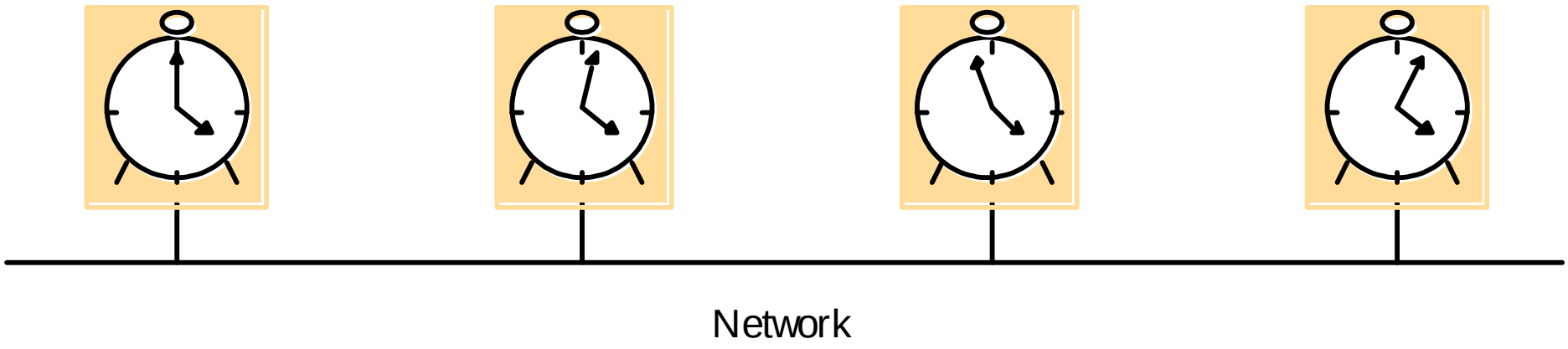
Derivas de reloj: significa que cuentan el tiempo a diferentes ritmos, y por lo tanto divergen. Los relojes basados en cristal utilizados en los computadores están sujetos a la deriva de reloj.

Los osciladores subyacentes están sujetos a variaciones físicas, con la consecuencia que sus frecuencias de oscilación difieren. Allí misma frecuencia de los relojes varía con la temperatura.

Figura 8.1
Sesgo entre los computadores en un sistema distribuido

Un ***ritmo de deriva de reloj*** es el cambio en la compensación entre el reloj y un reloj de referencia nominal perfecto por unidad de tiempo medido por el reloj de referencia.

Para relojes basados en un cristal de cuarzo, suele ser de aproximadamente 10^{-6} segundos/segundo, dando una diferencia de 1 segundo cada 1.000.000 segundos, o 11,6 días. Relojes de cuarzo de alta precisión 10^{-7} o 10^{-8}



Tiempo Universal Coordinado (UTC). Los relojes pueden sincronizarse a fuentes externas de tiempo de gran precisión. Los relojes físicos más precisos utilizan osciladores atómicos cuyo ritmo de deriva es aprox. de una parte en 10^{13}.

La salida de relojes atómicos se utiliza como estándar para el tiempo real transcurrido, conocido como *Tiempo Atómico Internacional*. Establecido como 9,192,631,770 periodos de transición entre 2 niveles hiperfinos de Celsio-133 en estado fundamental.

Los segundos, los años y otras unidades de tiempo que nosotros utilizamos están basadas en el tiempo astronómico. Fueron definidos originalmente en términos de la rotación de la tierra sobre su eje y su rotación alrededor del sol.

Sin embargo, el período de rotación de la tierra sobre su eje va siendo gradualmente más largo. Por lo tanto el tiempo astronómico y el tiempo atómico tienen una tendencia a separarse.

El *Tiempo Universal Coordinado* UTC es un estándar internacional de cronometraje. Basado en el tiempo atómico, aunque ocasionalmente se inserta un salto de un segundo (o, más raramente, borrado) para mantenerse en sintonía con el tiempo astronómico.

Las señales UTC se sincronizan y difunden regularmente desde las estaciones de radio terrestres y los satélites, que cubren muchas partes del mundo. Por ejemplo, en USA la estación de radio WWV difunde señales de tiempo en frecuencias de onda corta. Las fuentes de los satélites incluyen el Sistema de Posicionamiento Global (Global Positioning System, GPS).

Hay disponibles receptores comerciales. Comparados con UTC perfecto, las señales recibidas desde las estaciones terrestres tienen una precisión del orden de 0,1-10 milisegundos, dependiendo de la estación utilizada. Las señales recibidas desde GPS tienen una precisión de alrededor de 1 microsegundo. Los computadores con receptores adjuntos pueden sincronizar sus relojes con estas señales de tiempo.

MÉTODO DE CRISTIAN PARA SINCRONIZAR RELOJES

Cristian [1989] sugirió la utilización de un servidor de tiempo, conectado a un dispositivo que recibe señales de una fuente de UTC, para sincronizar computadores externamente. Bajo solicitud, el proceso servidor S proporciona el tiempo de acuerdo con su reloj.

Cristian observó que aunque no hay límite superior en los retardos de transmisión de mensajes en un sistema asíncrono, los tiempos de ida y vuelta de los mensajes intercambiados entre cada par de procesos son a menudo razonablemente cortos, una pequeña fracción de un segundo.

El describe el algoritmo como *probabilístico:* el método consigue sincronización sólo si los tiempos de ida y vuelta entre el cliente y el servidor son suficientemente cortos comparados con la precisión requerida.

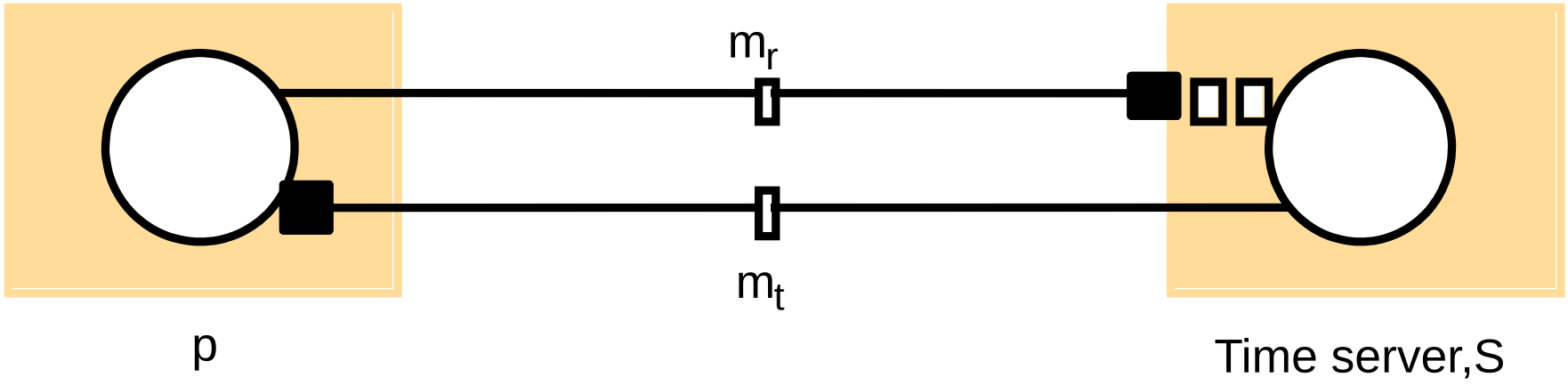
Método de Cristian para sincronizar relojes

Un proceso *p* solicita el tiempo en un mensaje m_r y recibe el valor del tiempo *t* en un mensaje m_t (*t* se inserta en m_t en el último instante posible antes de la transmisión desde un computador de S).

El proceso *p* registra el tiempo total de ida y vuelta T_{round} tomado para enviar la solicitud m_r y recibir la respuesta m_t . Se puede medir este tiempo con precisión razonable si su ritmo de deriva de reloj es pequeño.

Una estimación sencilla del tiempo al que *p* debe fijar su reloj es *t* + *Tround/2*, que supone que el tiempo transcurrido se desdoble igualmente antes y después de que S coloque *t* en *mt*. Esto es normalmente una suposición razonable de precisión, a menos que los dos mensajes sean transmitidos sobre redes diferentes.

Figura 7.2
Sincronización de relojes utilizando un servidor de tiempo



El protocolo de tiempo de red - NTP

El Protocolo de Tiempo de Red (*Network Time Protocol*, NTP) define una arquitectura para un servicio de tiempo y un protocolo para distribuir la información sobre Internet.
Los objetivos y las metas principales de diseño de NTP son los siguientes.

- *Proporcionar un servicio que permita a los clientes a lo largo de Internet estar sincronizados de forma precisa a UTC.*

- *Proporcionar un servicio fiable que pueda sobrevivir a pérdidas largas de conectividad.*

- *Permitir a los clientes resincronizar con suficiente frecuencia para compensar las tasas de deriva encontradas en la mayoría de los computadores.*

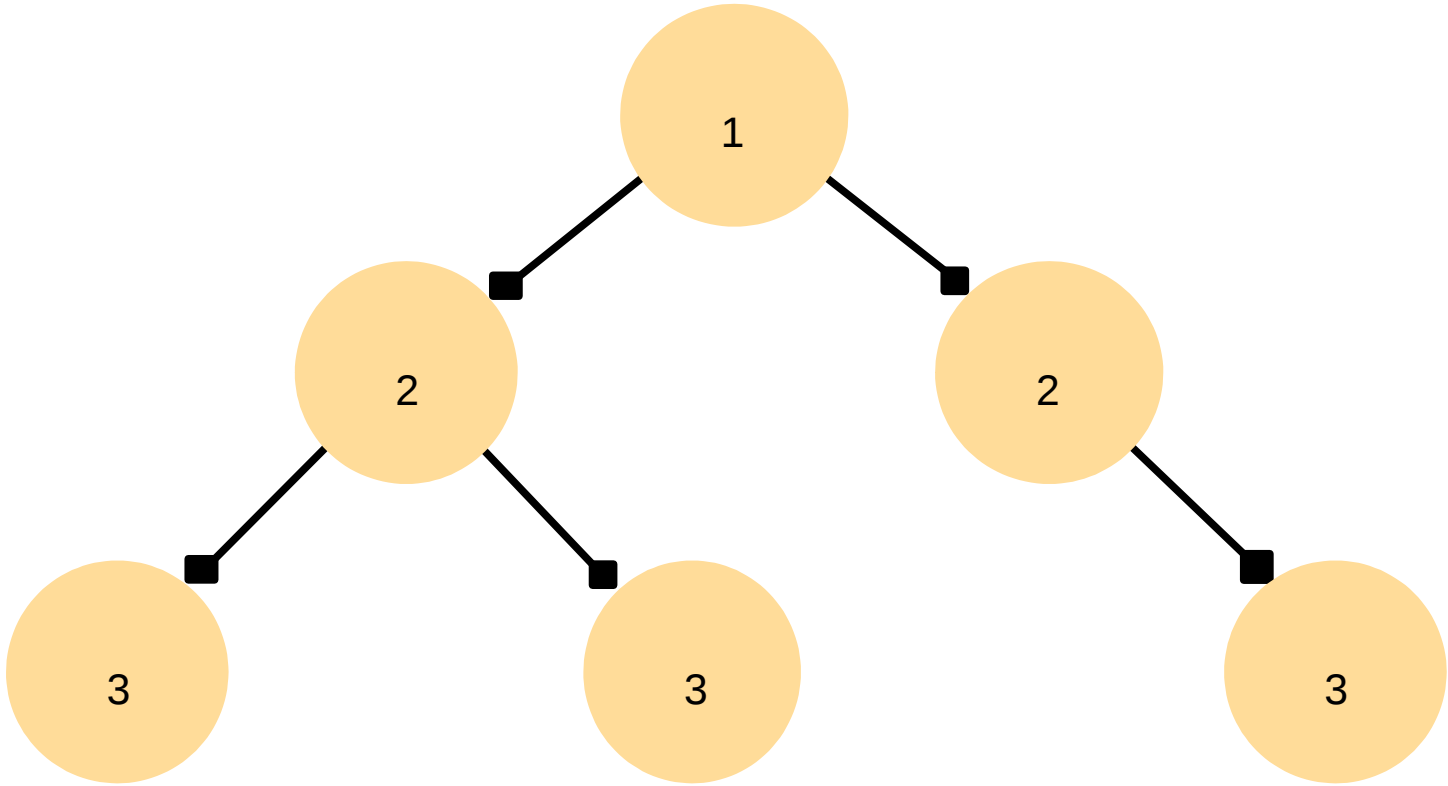
- *Proporcionar protección contra la interferencia con el servicio de tiempo, ya sea maliciosa o accidental.*

El protocolo de tiempo de red - NTP

El servicio NTP está proporcionado por una red de servidores localizados a través de Internet. Los *servidores primarios* están conectados directamente a una fuente de tiempo como un radioreloj recibiendo UTC; los servidores secundarios están sincronizados con servidores primarios. Los servidores están conectados en una jerarquía lógica llamada una *subred de sincronización*, cuyos niveles se llaman *estratos*.

Estrato 1: Servidores primarios, en la raíz.
Estrato 2: Servidores secundarios que están sincronizados directamente con los servidores primarios.
Estrato 3: Servidores que están sincronizados con los del estrato 2, y así sucesivamente.
Los servidores del nivel más bajo (hojas) se ejecutan en las estaciones de trabajo de los usuarios.

Figura 7.3
Ejemplo de una subred de sincronización en una implementación NTP.



Note: Arrows denote synchronization control, numbers denote strata.

Los servidores NTP se sincronizan entre sí en uno de estos tres modos: multidifusión, llamada a procedimiento y modo simétrico.

Multidifusión está pensado para su uso en una LAN de alta velocidad. Uno o más servidores reparten periódicamente el tiempo a los servidores que se ejecutan en otros computadores conectados en la LAN, que fijan sus relojes suponiendo un pequeño retardo. Este modo puede alcanzar sólo precisiones relativamente bajas, pero que son consideradas suficientes para muchos propósitos.

Llamada a procedimiento es similar al funcionamiento del algoritmo de Cristian. Un servidor acepta solicitudes de otros computadores, que el procesa respondiendo con su marca de tiempo (lectura actual del reloj). Este modo es adecuado donde se requieren precisiones más altas que las que se pueden conseguir con multidifusión, o donde la multidifusión no viene soportada por hardware.

Simétrico está pensado para su utilización por servidores que proporcionan información del tiempo en LANs y por los niveles más altos (estratos más bajos) de la subred de sincronización, donde se deben obtener las precisiones más altas.

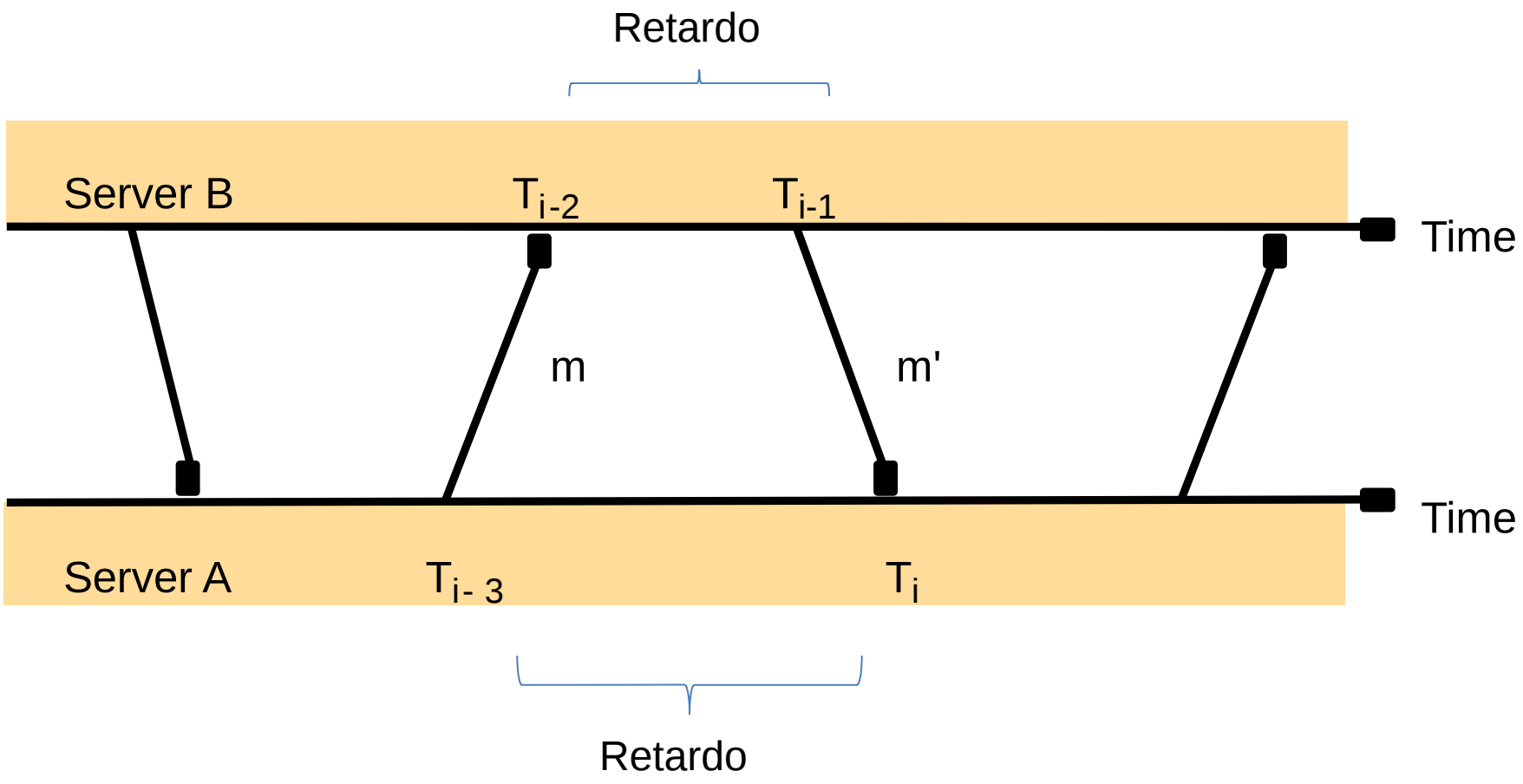
Un par de servidores operando en modo simétrico intercambian mensajes llevando información del tiempo. Los datos del tiempo son retenidos como parte de una asociación entre los servidores que se mantiene con el fin de mejorar su sincronización en el tiempo.

En todos los modos, los mensajes se entregan de modo no fiable, utilizando el protocolo de transporte estándar Internet UDP

En el modo de llamada a procedimiento y en el simétrico, los procesos intercambian pares de mensajes. Cada mensaje lleva marcas de tiempo de los sucesos del mensaje reciente: los tiempos locales cuando el mensaje anterior NTP entre el par fue enviado y recibido y el tiempo local cuando el mensaje actual fue transmitido. El receptor del mensaje NTP anota el tiempo local cuando el recibe el mensaje.

Los cuatro tiempos *Ti-3*, *Ti-2*, *Ti-1*, *Ti se* muestran en la Figura 8.4 para los mensajes *m* y *m'* enviados entre los servidores *A* y *B* . En el modo simétrico, a diferencia del algoritmo de Cristian, puede haber un retardo no despreciable entre la llegada de un mensaje y el envío del siguiente. También, se pueden perder mensajes, pero sin embargo las marcas de tiempo son válidas.

Figura 8.4
Mensajes intercambiados entre un par de iguales NTP



Por cada par de mensajes enviados entre dos servidores NTP calcula una *compensación oi* que es una estimación de la deriva actual entre los dos relojes, y un *retardo di*, que es el tiempo total de transmisión para los dos mensajes. Si el verdadero desplazamiento del reloj en *B* con relación al de *A* es o, y si los tiempos de transmisión actuales para *m* y *m'* son *t* y *t'* relativamente, entonces tenemos:

$T_{i-2} = T_{i-3} + t + O$ y $Ti = Ti\text{-}1 + t' + O$

Esto conduce a:

$di = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$ Retardo! Verificar en gráfico.

También se verifica que:

$o = oi + (t' - t)/2$ donde $oi = (Ti\text{-}2 - Ti\text{-}3 + Ti\text{-}1 - Ti)/2$

Usando el hecho de que *t*, *t'* >= 0 se puede ver que o*i* - *di/2* <=o<=oi+ *di/2*. Por tanto o*i* es una estimación de la deriva, y *di* es una medida de la precisión de esta estimación.

Los servidores NTP aplican un algoritmo de filtrado de datos a pares sucesivos de <o*i*, *di* > , que estima la deriva ***o*** y calculan la calidad de esta estimación como una cantidad estadística llamada el *filtro de dispersión*. Un filtro de dispersión relativamente alto implica datos relativamente poco fiables. Los ocho pares <oi, *di* > más recientes son retenidos. Como con el algoritmo de Cristian, se elige como estimación de ***o*** el valor de ***oi*** que corresponde con el mínimo valor ***di***.

Tiempo lógico y relojes lógicos

Desde el punto de vista de un único proceso, los sucesos están ordenados de forma única por los tiempos mostrados en el reloj lógico.

Sin embargo, como apuntó Lamport [1978], puesto que no podemos sincronizar perfectamente los relojes a lo largo de un sistema distribuido, no podemos usar, en general, el tiempo físico para obtener el orden de cualquier par arbitrario de sucesos que ocurran en él.

En general, podemos utilizar un esquema que es similar a la causalidad física, pero que se aplica en los sistemas distribuidos, para ordenar algunos de los sucesos que ocurren en diferentes procesos.

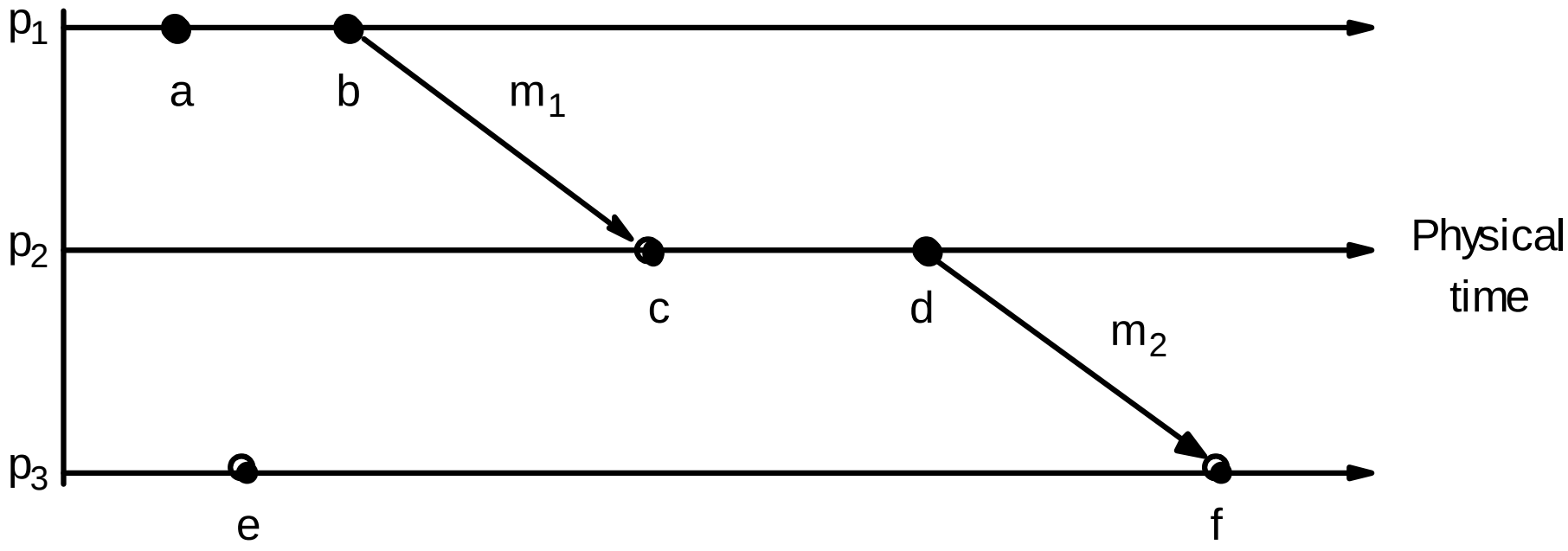
Esta ordenación está basada en dos puntos sencillos e intuitivamente obvios:

- Si dos sucesos han ocurrido en el mismo proceso $p_i (i = 1,2, \ldots, N)$, entonces ocurrieron en el orden en el que les observa p_i, éste es el orden ->i que hemos definido anteriormente.

- Cuando se envía un mensaje entre procesos, el suceso de enviar el mensaje ocurrió antes del de recepción del mismo.

Lamport llamó a la ordenación parcial obtenida al generalizar estas dos relaciones la realización ***suceder antes***. Esto también se conoce a veces como la relación de *orden causal* o *ordenación causal potencial*.

Figura 7.5
Sucesos ocurridos en tres procesos



Relojes lógicos. *Lamport* inventó un *mecanismo simple* por *el* que *la relación "sucedió antes" puede* capturarse numéricamente, *denominado reloj lógico*. *Un reloj lógico* de Lamport es un contador software que se incrementa monótonamente, cuyos *valores* no *necesitan* tener *ninguna relación* particular con *ningún reloj físico*.

Cada proceso p_i *mantiene* su propio *reloj lógico*, L_i , que *él* utiliza para aplicar *las llamadas marcas de tiempo de Lamport* a *los* sucesos.

Representamos la marca de tiempo *e del* suceso en p_i por L_i *(e)*, y representamos por *L(e) la* marca de tiempo del suceso *e* cualquiera que sea el proceso en el que ocurrió.

Para capturar la relación "*sucedió antes*" ->, *los* procesos actualizan sus relojes lógicos y transmiten los valores de sus relojes lógicos en mensajes como sigue:

RL1: Li se incrementa antes de emitir cada suceso en *el* proceso *Pi:*

$L_i = L_i + 1$

RL2: (a) Cuando un proceso P_i envía un mensaje *m*, acarrea en *m el* valor de $t = L_i$.

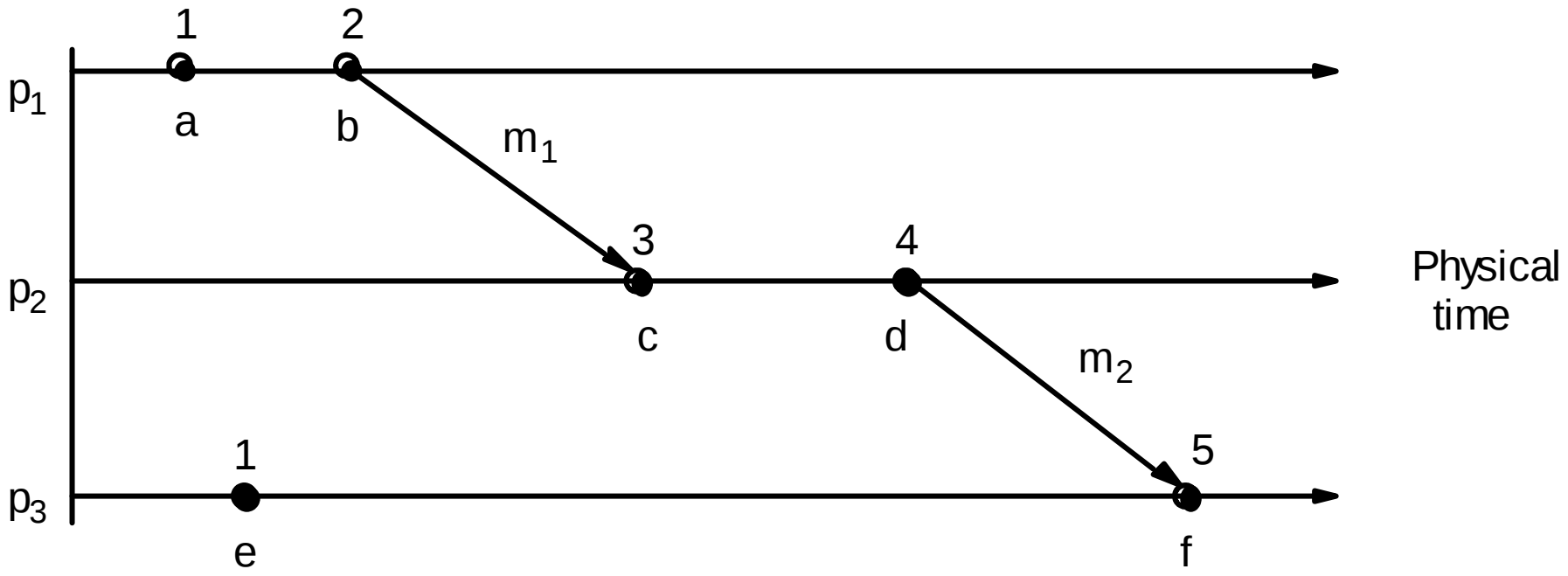
(b) *Al* recibir *(m, t)*, cada proceso P_i *calcula* $L_j = max(L_j, t)$ y entonces aplica *RL1* antes de realizar la marca de tiempo del suceso *recibe(m)*.

Aunque nosotros incrementamos los relojes en 1, podríamos haber elegido cualquier valor positivo.

Se puede ver fácilmente, por inducción en la longitud de cualquier secuencia de sucesos relacionando dos sucesos e y e', que e-> e' => L(e) < L(e').

Hay que señalar que el inverso no es verdadero. Si L(e) < L(e'), no podemos inferir que e-> e' (que ***e*** ocurrió antes que ***e'***)

Figura 7.6
Marcas temporales de Lamport para los eventos mostrados en la Fig. 8.5.



Relojes lógicos totalmente ordenados.

Algunos pares de sucesos distintos, generados por diferentes procesos, tienen marcas de tiempo de Lamport numéricamente idénticas.

Podemos crear un orden total sobre los sucesos, esto es, uno para el que todos los pares de sucesos distintos están ordenados, teniendo en cuenta los identificadores de los procesos en los que ocurren los sucesos.

Si *e* es un suceso que ocurre en *pi* con marca de tiempo local *Ti*.
Si *e'* es un suceso que ocurre en *pj* con marca de tiempo local *Tj*.

Definimos las marcas de tiempo globales para esos sucesos como *(Ti, i) y (Tj, j)* respectivamente. y definimos *(Ti, i) <= (Tj, j)* si:

- *Ti < Tj*
- *Ti = T j siendo i < j*.

Esta ordenación no tiene significado físico general (porque los identicadores de los procesos son arbitrarios), pero a veces es útil.

Lamport la utilizó, por ejemplo, para ordenar la entrada de procesos en una sección crítica

Relojes vectoriales. Mattern [1989] y Fidge [1991] desarrollaron relojes vectoriales para vencer la deficiencia de los relojes de Lamport: del hecho que *L(e) < L(e')* no podemos deducir que *e -> e'*.

- Un reloj vectorial para un sistema de *N* procesos es un vector de *N* enteros.
- Cada proceso mantiene su propio reloj vectorial *Vi*, que utiliza para colocar marcas de tiempo en los sucesos locales.
- Como las marcas de tiempo de Lamport, cada proceso adhiere el vector de marcas de tiempo en los mensajes que envía al resto, y hay unas reglas sencillas para actualizar los re]ojes.

Reglas en relojes vectoriales.

RV1: Inicialmente, *Vi[j]* = 0, para *i*, *j* = 1,2..., *N*.

RV2: Justo antes que *Pi* coloque una marca de tiempo en un suceso, coloca *Vi[i]:=Vi[i]+1*

RV3: *pi* incluye el valor *t* = *Vi* en cada mensaje que envía.

RV4: Cuando *pi* recibe una marca de tiempo y en un mensaje, establece *Vi[j]:* =max(*Vi[j]*, *t[j])*, para *j* = 1, 2 ..., *N*. Esta operación de *mezcla* toma el vector máximo entre dos, componente a componente.

Para un vector *Vi:*

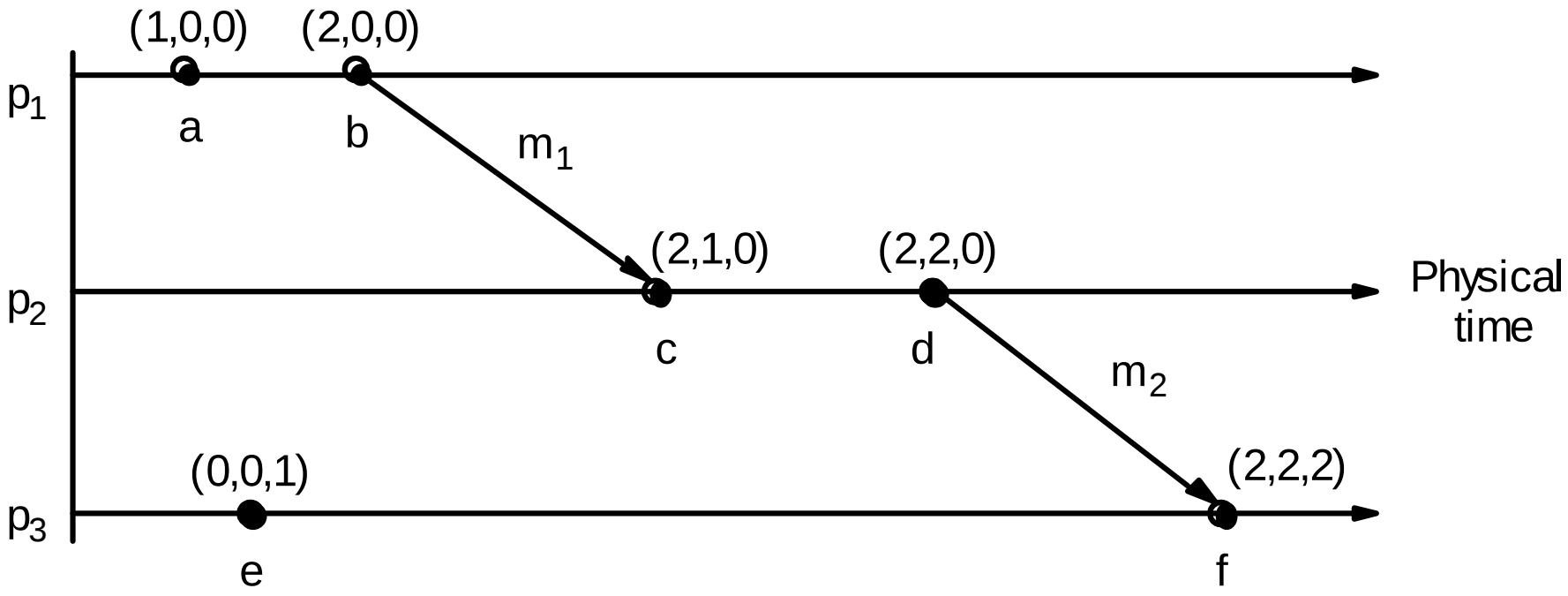
Vi[i] es el número de sucesos a los que *pi* ha puesto una marca de tiempo.
Vi[j](j ≠ i) es el número de sucesos que han ocurrido en pj que han sido potencialmente afectados por pi.

Podemos comparar vectores de marcas de tiempo de la forma siguiente:

V = *V'* si y sólo si *V[j]* = *V'[j]* para *j* = 1, 2..., *N*
V<= *V'* si y sólo si *V[j]* <= *V'[j] para j* = 1,2..., *N*
V < *V'* si y sólo si *V* <= *V'* ^ *V* ≠ *V'*

Sea *V(e)* el vector de marcas de tiempo aplicadas por el proceso en el que ocurre *e*. Es sencillo mostrar, por inducción sobre la longitud de cualquier secuencia de sucesos que relacione los sucesos *e* y *e'*, que *e* -> *e'* => *V(e)* < *V(e')* .

Figura 7.7
Vector de marcas temporales para los eventos mostrados en la Fig. 8.6.



Estados globales

Compactación automática de memoria:
Un objeto se considera desechable si no hay posteriormente ninguna referencia a él desde cualquier parte del sistema distribuido. La memoria ocupada por el objeto puede ser reclamada, una vez que se sabe que éste es desechable. Para comprobar, debemos verificar que no hay referencias a él en cualquier parte del sistema.

En la Figura 8.8a, el proceso P1 tiene dos objetos, ambos con referencias, uno tiene una referencia al propio P1 y el otro tiene una referencia a P2.

El proceso P2 tiene un objeto desechable, al que no existe ninguna referencia en el sistema. Hay también otro objeto al que ni P1 ni P2 tienen acceso, pero hay una referencia a él en un mensaje que transita entre ambos. Esto nos muestra que cuando consideramos las propiedades de un sistema, debemos incluir el estado de los canales de comunicación así como el de los procesos.

Detección distribuida de bloqueos indefinidos: un bloqueo indefinido distribuido ocurre cuando cada uno de los procesos de una colección espera por otro proceso para enviarle un mensaje, y donde hay un ciclo en el grafo de esta relación *espera por*. La Figura 8.8b muestra que cada uno de los procesos P1 y *P2* espera un mensaje del otro, por lo que el sistema nunca progresará.

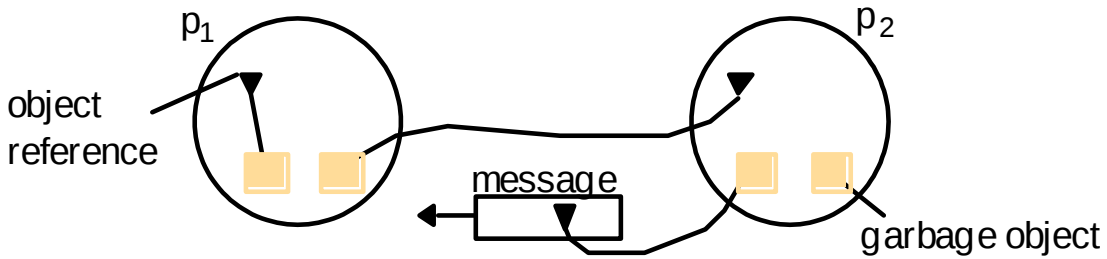
Detección de la terminación distribuida: el problema aquí es detectar que un algoritmo distribuido ha terminado. La detección de la terminación es un problema que parece muy fácil de resolver: al principio parece sólo necesario comprobar si cada proceso se ha detenido.

Para ver que esto no es así, consideremos un algoritmo distribuido ejecutado por dos procesos P1 y *P2* cada uno de los cuales puede necesitar valores del otro. Instantáneamente, podemos encontrar que un proceso es activo o pasivo, un proceso pasivo no está ligado a ninguna actividad por sí mismo pero está preparado para responder con un valor solicitado por el otro. Supongamos que descubrimos que *P1* es pasivo y que *P2* también lo es (véase la Figura 8.8c).

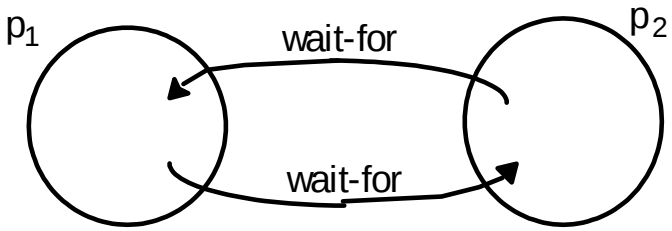
Para ver que no podemos determinar que el algoritmo ha terminado, consideremos el siguiente escenario:

cuando hemos comprobado P1 para ver si es pasivo, un mensaje estaba en su camino desde *P2* que se convirtió en pasivo inmediatamente después de enviarlo. Cuando reciba el mensaje, Pl se convertirá de nuevo en activo, después de encontrar que era pasivo. El algoritmo no había terminado.

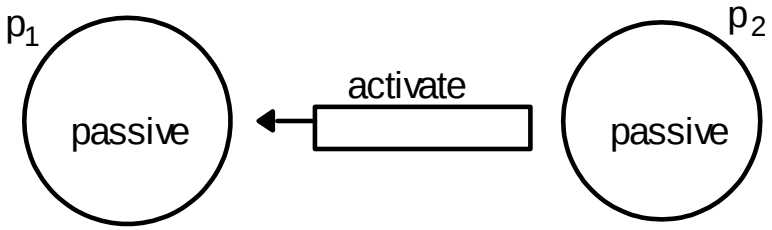
Figura 7.8
Detección de propiedades globales



a. Garbage collection



b. Deadlock



c. Termination

Estados globales y cortes consistentes

En principio, es posible observar la sucesión de estados de un proceso individual, pero cómo establecer un estado global del sistema, el estado de la colección de procesos es más difícil de tratar.

El problema esencial es la **ausencia de tiempo global**. Si todos los procesos tuvieran los relojes perfectamente sincronizados ellos podrían estar de acuerdo en un tiempo en el que cada proceso debía registrar su estado, el resultado sería un estado global actual del sistema.

De una colección de estados de proceso deberíamos decir, por ejemplo, qué procesos estaban bloqueados indefinidamente. Pero no podemos obtener una sincronización perfecta de los relojes, por lo que este método no está disponible.

Para elaborar un estado global significativo de los estados locales registrados en diferentes tiempos reales, debemos presentar algunas definiciones:

Hemos dicho anteriormente que ocurre una serie de eventos en cada proceso, y que podemos caracterizar la ejecución de un proceso por su historia:

$$historia\ (pi) = hi = \langle e^{o}i,\ e^{1}i, \ldots\ldots \rangle$$

Podemos formar la historia global de Ω como la unión de las historias individuales de los procesos:

$$H = h_0 \ U \ h_1 \ U \ldots\ldots\ldots \ h_{n-1}$$

Estados globales y cortes consistentes

Matemáticamente, podemos tomar cualquier conjunto de estados de los procesos individuales para formar un estado global *(s1, s2* … sn*)*.

Pero qué estados son significativos; esto es, ¿cuál de los estados de los procesos podrían haber ocurrido al mismo tiempo? Un estado global corresponde a los prefijos iniciales de las historias individuales de los procesos.

Un *corte* de la ejecución del sistema es un subconjunto de su historia global que es la unión de los prefijos de las historias de los procesos:

$$C = h^{c1}_{1} + h^{c2}_{2} + \ldots\ldots\ldots + h^{cn}_{n}$$

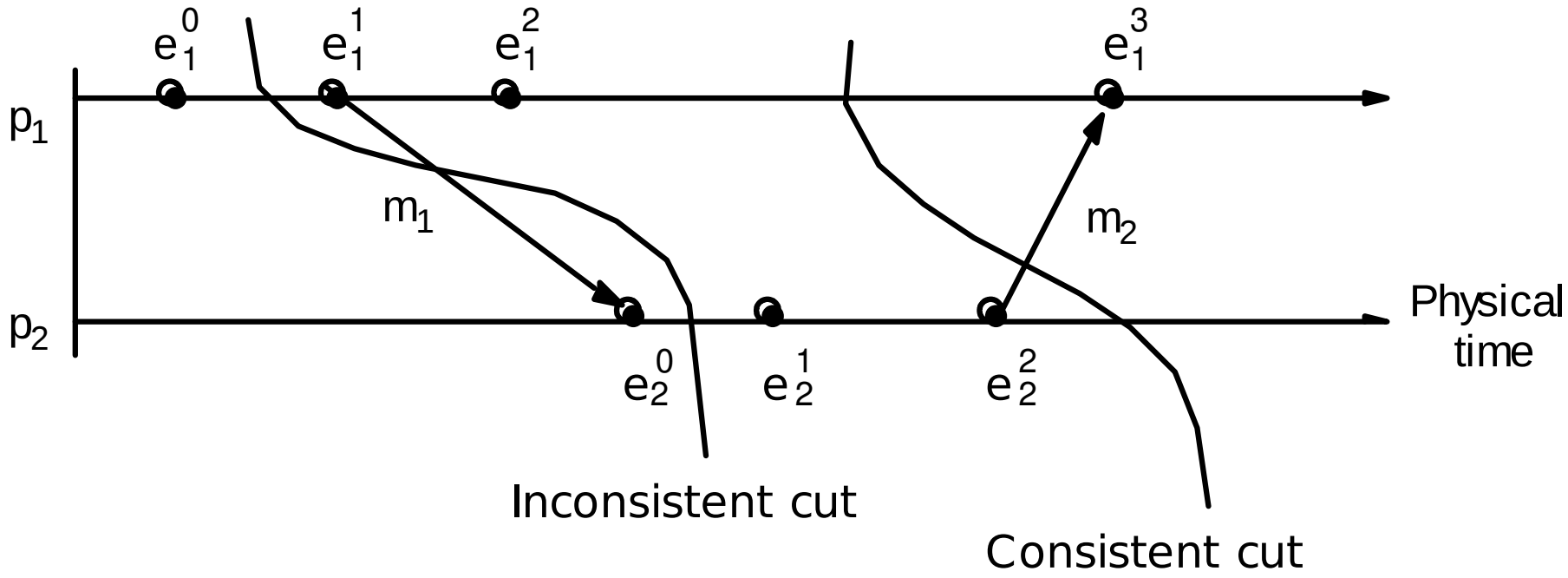
El estado *si* en el estado global S correspondiente al corte C es el de *pi* inmediatamente después del último suceso procesado por *pi* en el corte, $e^{ci}_i(i = 1, 2, ..., N)$. El conjunto de sucesos $\{e^{ci}_i : i = 1, 2, ..., N\}$ se llama la *frontera* del corte.

Consideremos los sucesos que ocurren en los procesos *p1* y p2 mostrados en la Figura 8.9. La figura representa dos cortes, uno con frontera $< e^0_1 , e^0_2 >$ y otro con frontera $< e^2_1 , e^2_2 >$.

El primer corte de más a la izquierda es *inconsistente*. Esto es porque *p2* incluye la recepción del mensaje *m1*, pero en *p1* no incluye el envío del mensaje. Esto se considera como un *efecto* sin una *causa*.

Un corte C es consistente, si para cada suceso que contiene, también contiene todos los sucesos que sucedieron antes del suceso.

Figura 8.9
Cortes



Un estado global consistente es uno que corresponde con un corte consistente. Podemos caracterizar un sistema distribuido como una serie de transiciones entre los estados globales del sistema:

S0 -> S1 ->S2.....

Algoritmo de instantánea de Chandy y Lamport

Chandy y Lamport [1985] describen un algoritmo de ***instantánea*** para determinar estados globales de sistemas distribuidos.

El objetivo es registrar un conjunto de estados de procesos y canales (una *instantánea)* para un conjunto de procesos *pi(i* = 1, 2, ..., *N)* tal que, incluso a través de una combinación de estados registrados que nunca podrían haber ocurrido al mismo tiempo, el estado global registrado sea consistente.

Veremos que el estado *que registra el algoritmo* de *instantánea* tiene propiedades convenientes para evaluar predicados del estado global.

El algoritmo registra el estado localmente en los procesos; no proporciona un método para recoger el estado global en un sitio.

Algoritmo de instantánea de Chandy y Lamport

El algoritmo supone que:

- No fallan ni los canales ni los procesos; la comunicación es fiable por lo que cada mensaje enviado es recibido intacto, exactamente una vez.
- Los canales son unidireccionales y proporcionan la entrega de los mensajes con ordenación FIFO.
- El grafo de los procesos y canales está fuertemente conectado (hay un recorrido entre dos procesos cualquiera).
- Cualquier proceso puede iniciar una instantánea global en cualquier instante.
- Los procesos pueden continuar su ejecución y enviar y recibir mensajes normales mientras tiene lugar la instantánea.

Algoritmo de instantánea de Chandy y Lamport

El algoritmo se define mediante de dos reglas, la *regla de **recepción** del marcador, y la regla de **envío** del marcador*.

La regla de envío del marcador obliga a los procesos a enviar un marcador después de haber registrado su estado, pero antes de que envíen cualquier otro mensaje.

La regla de recepción del marcador obliga a un proceso que no ha registrado su estado a hacerlo. En ese caso, éste es el primer marcador que ha recibido. Él observará qué mensajes llegan posteriormente en los otros canales entrantes. Cuando un proceso que ya ha guardado su estado recibe un marcador (en otro canal), registra el estado de ese canal bajo la forma del conjunto que recibió desde que guardó su estado.

Algoritmo de instantánea de Chandy y Lamport

Cualquier proceso puede comenzar el algoritmo en cualquier instante. Actúa como si hubiera recibido un marcador y sigue la regla de recepción del marcador. Por tanto registra su estado y comienza a registrar los mensajes que llegan sobre todos los canales entrantes.

Figura 7.10
Algoritmo de instantánea de Chandy y Lamport

Regla de recepción del marcador para el proceso p_i

Cuando p_i recibe un mensaje *marcador* sobre el canal *c*:

si (p_i no ha registrado todavía su estado)

registra su estado de proceso ahora;

registra el estado de *c* como el conjunto vacío;

activa el registro de los mensajes que llegan sobre otros canales entrantes;

sino

p_i registra el estado de *c* como el conjunto de mensajes que ha recibido sobre *c* desde que guardó su estado

fin si

Regla de envío del marcador para el proceso p_i

Después p_i ha registrado su estado para cada canal de salida *c*:

p_i envía un mensaje marcador sobre *c* (antes que envíe otro mensaje sobre *c*).

Figura 10.10. Algoritmo de *instantánea* de Chandy y Lamport.

Ilustramos el algoritmo para un sistema con dos procesos *p*1 y *p*2 conectados por dos canales unidireccionales *c*1 y *c*2. Los dos procesos comercian con artículos. El proceso *p*1 envía una orden de compra de artículos sobre c2 hacia p2 incluyendo el pago al precio de 10 dólares por artículo.

Algún tiempo después, el proceso p2 envía artículos a través del canal *c*1 hacia p1. Los procesos están en el estado inicial que se ve en la Figura 8.11. El proceso p2 ha recibido ya una orden por cinco artículos que serán enviadas en breve por p1 .

Figura 7.11
Dos procesos y sus estados iniciales

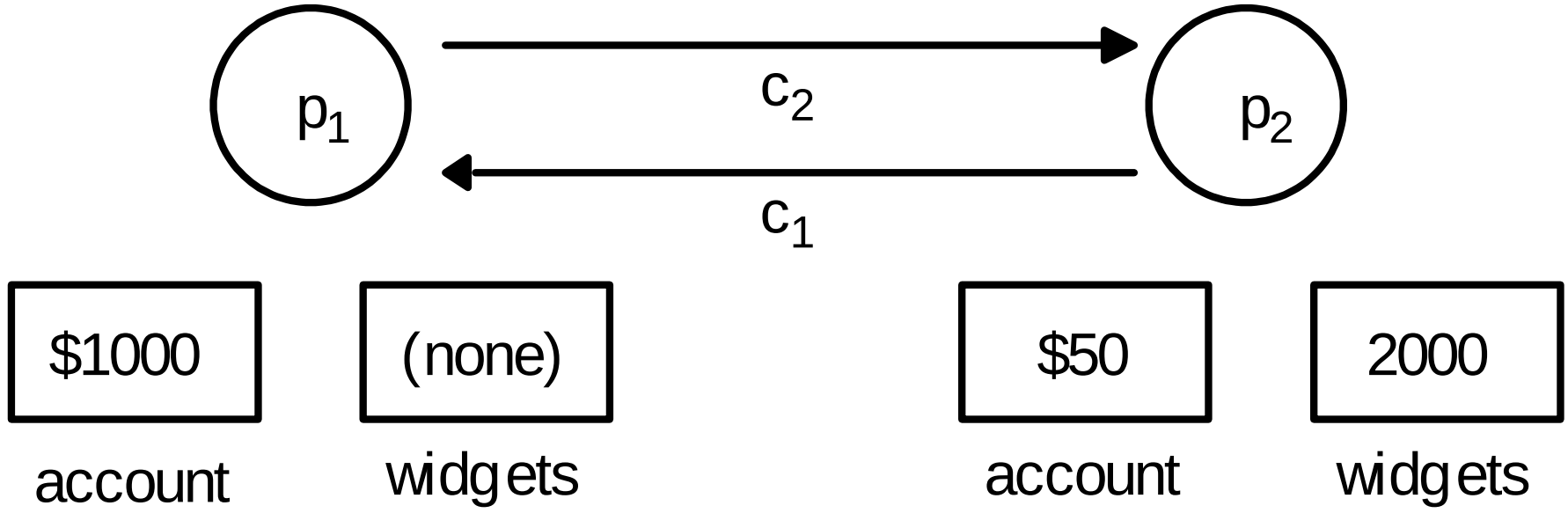


Figura 7.12
La ejecución de los procesos de la Figura 8.11

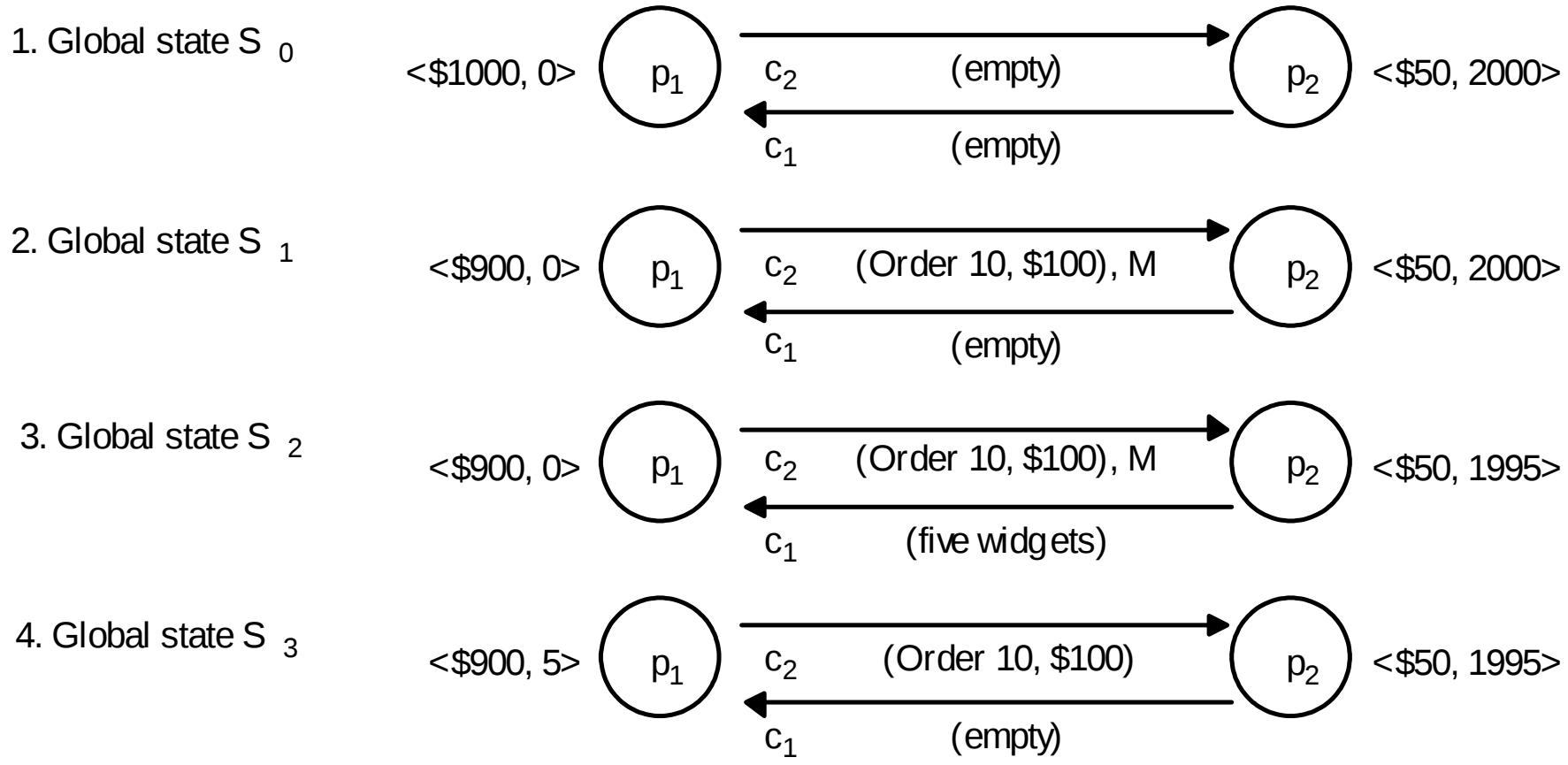
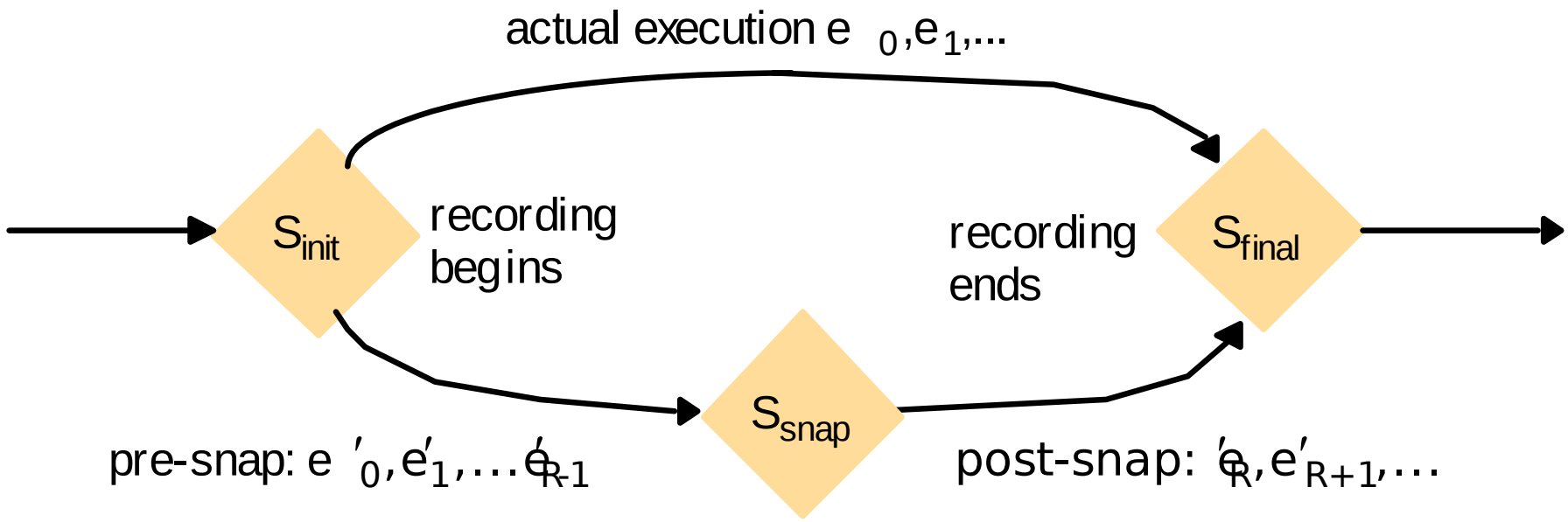


Figura 7.13
Alcanzabilidad entre estado en el algoritmo de instantánea



Depuración distribuida:

El algoritmo de instantánea de Chandy y Lamport recoge el estado de una forma distribuida y se ha indicado ya cómo los procesos del sistema podrían enviar el estado que ellos conforman a un proceso monitor para su recogida.

El *algoritmo* que describiremos (debido a Marzullo y Neiger [1*991])* es centralizado. Los procesos observados envían *sus* estados a un proceso llamado ***monitor***, que ensambla estados globalmente consistentes de los que recibe. Consideramos que el monitor se encuentra fuera del sistema observando su ejecución.

Nuestro objetivo es: determinar casos en los que dado un predicado de estado global ***O*** era sin duda alguna *Verdadero* en algún punto de la ejecución que observamos, y los casos en los que posiblemente era *Verdadero*.

La noción *posiblemente* surge como un concepto natural porque podemos extraer un estado global consistente S de un sistema en ejecución y encontrar que ***O****(S)* es V*erdadero*.

Una observación única de un estado global consistente no nos permite concluir si un predicado no estable será evaluado siempre a *Verdadero* en *la* ejecución actual.

La noción *sin duda alguna* se aplica a la ejecución actual y no a una ejecución que hayamos extrapolado de ella. Nos puede parecer paradójico considerar qué sucedió en *la* ejecución actual.

Sin embargo, es posible evaluar *si O* fue *sin* duda alguna *Verdadero* considerando todas las linealizaciones de los sucesos observados.

La afirmación *posiblemente* ***O*** significa que hay un estado consistente S a través del *cual* pasa una linealización *H* tal que O*(S)* es *Verdadero*.

La afirmación *sin duda alguna* ***O*** *significa* que para todas las linealizaciones *L* de *H*, hay un estado consistente S a través del cual pasa *L* tal que O*(S)* es *Verdadero*.

Figura 7.14
Vectores de marcas de tiempo y valores variables para la ejecución de la Figura 8.9

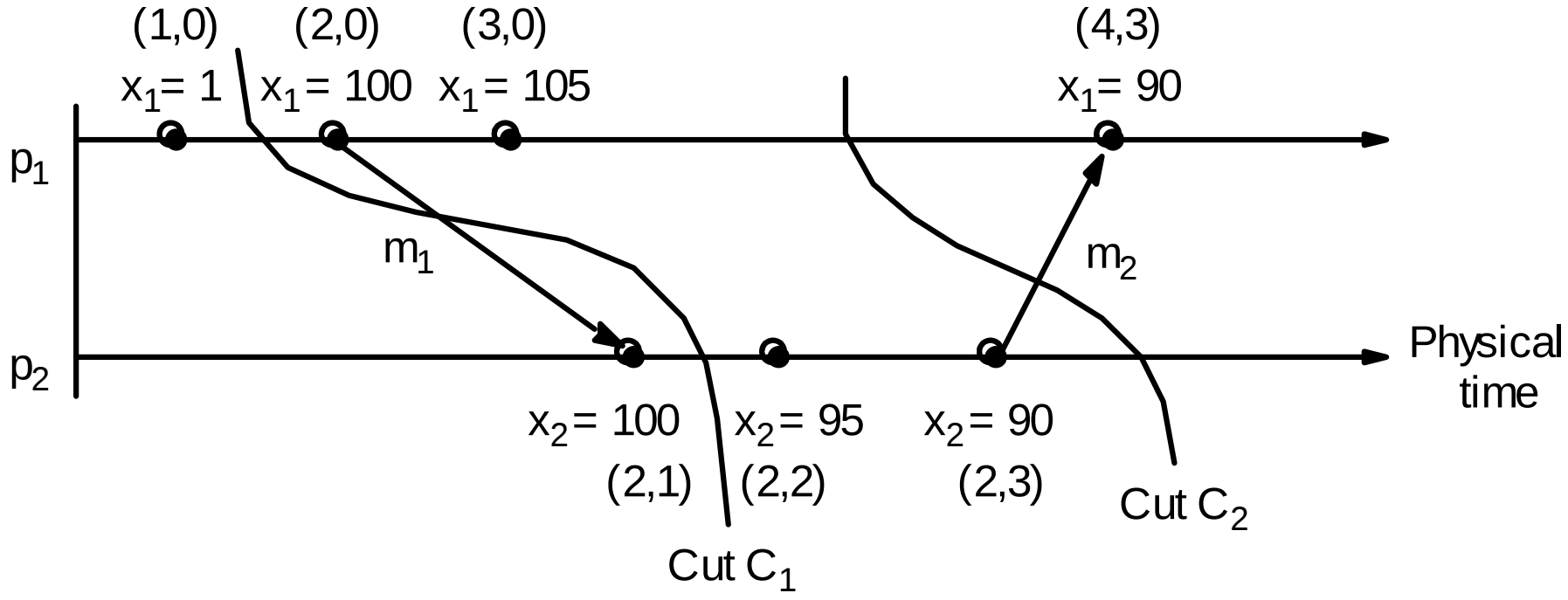
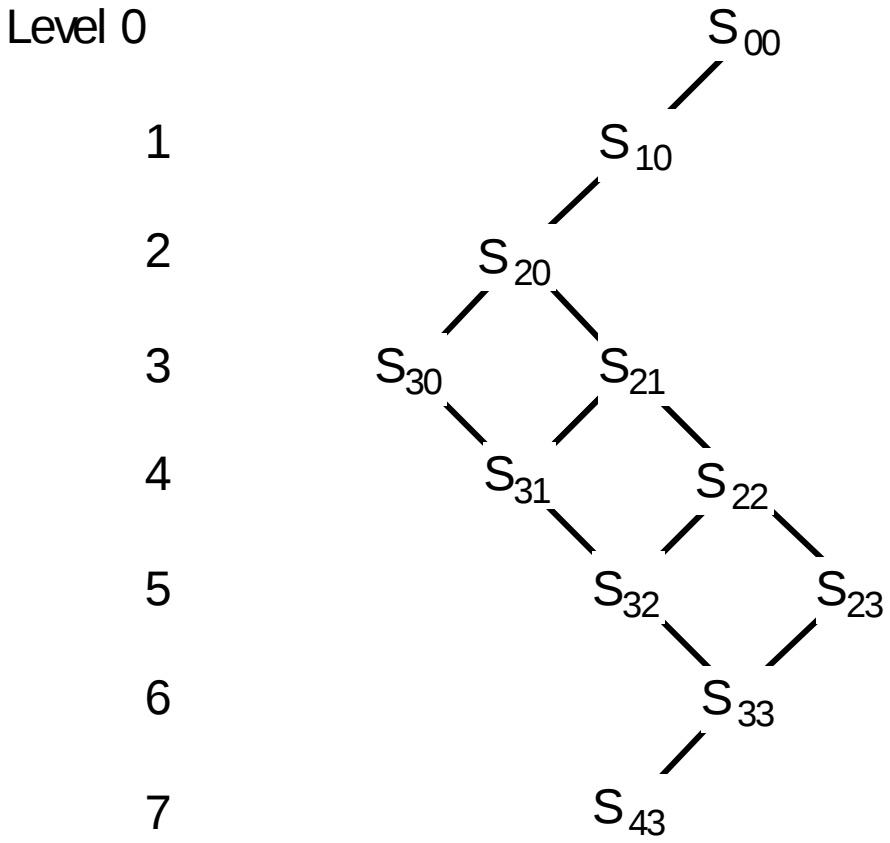


Figura 7.15
La red de estados globales para la ejecución de la Figura 8.14



Sij= global state after i events at process 1 and j events at process 2

Sij = estado global después de i eventos en el proceso 1 y j eventos en el proceso 2.

Figura 7.16
Algoritmos para evaluar posiblemente ϕ y sin duda alguna ϕ

1. *Evaluando posiblemente ϕ para la historia global H de N procesos*
   $L := 0$
   Estados $:= \{ (s_1^0, s_2^0, \ldots, s_N^0)\}$;
   mientras (ϕ (S) = *Falso* para todos los $S \in$ *Estados*)
      $L := L + 1$;
      Alcanzable$:= \{S'$: S' alcanzable en H desde algún $S \in$ *Estados* $\wedge$ *nivel* $(S') = L\}$;
      Estados $:=$ *Alcanzable*
   fin mientras
   salida "*posiblemente ϕ*";

2. *Evaluando sin duda alguna ϕ para la historia global H de N procesos*
   $L := 0$;
   si ($\phi(s_1^0, s_2^0, \ldots, s_N^0)$) *entonces Estados* $:= \{\}$ *sino Estados* $:= \{ (s_1^0, s_2^0, \ldots, s_N^0)\}$;
   mientras (*Estados* $\neq \{\}$)
      $L := L + 1$;
      Alcanzable$:= \{S'$: S' alcanzable en H desde algún $S \in$ *Estados* $\wedge$ *nivel* $(S') = L\}$;
      Estados$:= \{S \in$ *Alcanzable*: $\phi(S) =$ *Falso*$\}$
   fin mientras
   salida "*sin duda alguna ϕ*";

Figura 10.16. Algoritmos para evaluar posiblemente ϕ y *sin duda alguna* ϕ.

Figura 7.17
Evaluando sin duda alguna ϕ