Tiempo, causalidad y estado global

Sistemas Distribuidos

Alberto Lafuente, Mikel Larrea
Dpto. ATC, UPV/EHU

Tiempo, causalidad y estado global

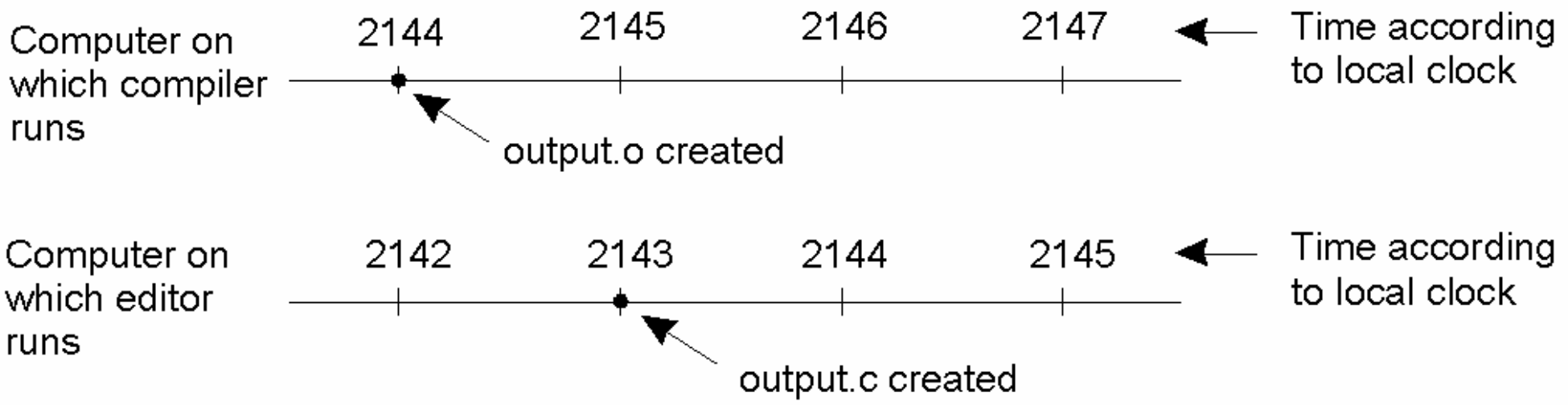
1 Introducción

- En un sistema distribuido el *estado global* se encuentra distribuido entre los nodos
  - reloj estándar en cada nodo
  - comunicación entre nodos: retardos
  - cada nodo posee una *visión subjetiva* del estado global
- Ejemplos:
  - en el instante t1 se manda un mensaje desde el nodo A al nodo B, que lo recibe en el instante t2, con t2 < t1
  - establecer con precisión absoluta el montante de los depósitos en todas las cuentas bancarias de todos los bancos del mundo en un instante de tiempo dado

1 Introducción

- Propuesta de solución: un único reloj preciso + red dedicada para transmitir la señal sin retardos
  - no es práctico (coste), o es inviable (Internet)
- Solución: tiempo distribuido
  - cada nodo posee su propio reloj (tiempo físico local)
  - los relojes son imprecisos: necesario ajustarlos periódicamente a un tiempo físico de referencia
- Por otra parte, la gestión consistente del estado global requiere al menos ordenar los eventos producidos por los nodos (causalidad)
  - tiempo lógico

Clock Synchronization

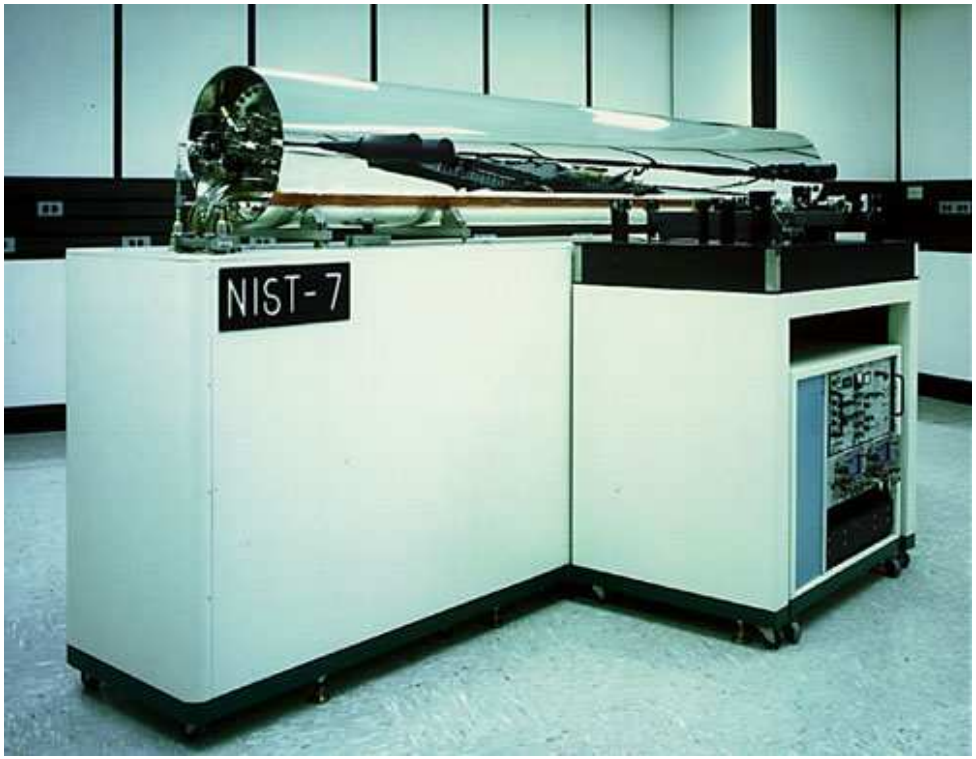


- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

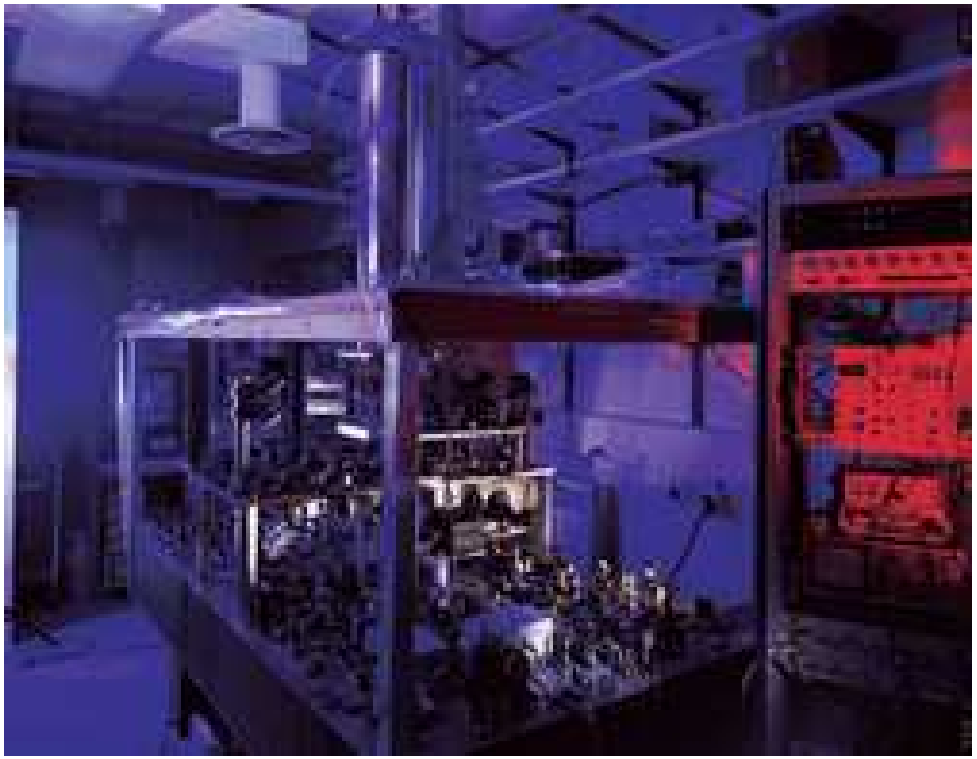
2 Tiempo físico

- Los relojes de los computadores son de cuarzo
  - La frecuencia de oscilación varía con la temperatura
  - deriva: ~10^{-6} (90ms en un día, 1s cada 11,6 días)
- Relojes atómicos: gran precisión, muy caros
  - deriva: ~10^{-13} (9ns en un día, 1s cada 300000 años)
  - precio: $50.000 - $100.000 !!!
- Tipos de sincronización:
  - conocer el instante preciso en que se produce un evento en un nodo: sincronización externa
  - medir el intervalo entre dos eventos producidos en nodos diferentes (por ejemplo, envío y recepción de un mensaje), usando los relojes locales de cada nodo: sincronización interna
  - la interna no implica la externa, pero al revés sí

Relojes atómicos



NIST-7 (1993)
Deriva: 5×10^{-15}



NIST-F1 (2005)
Deriva: 5×10^{-16}

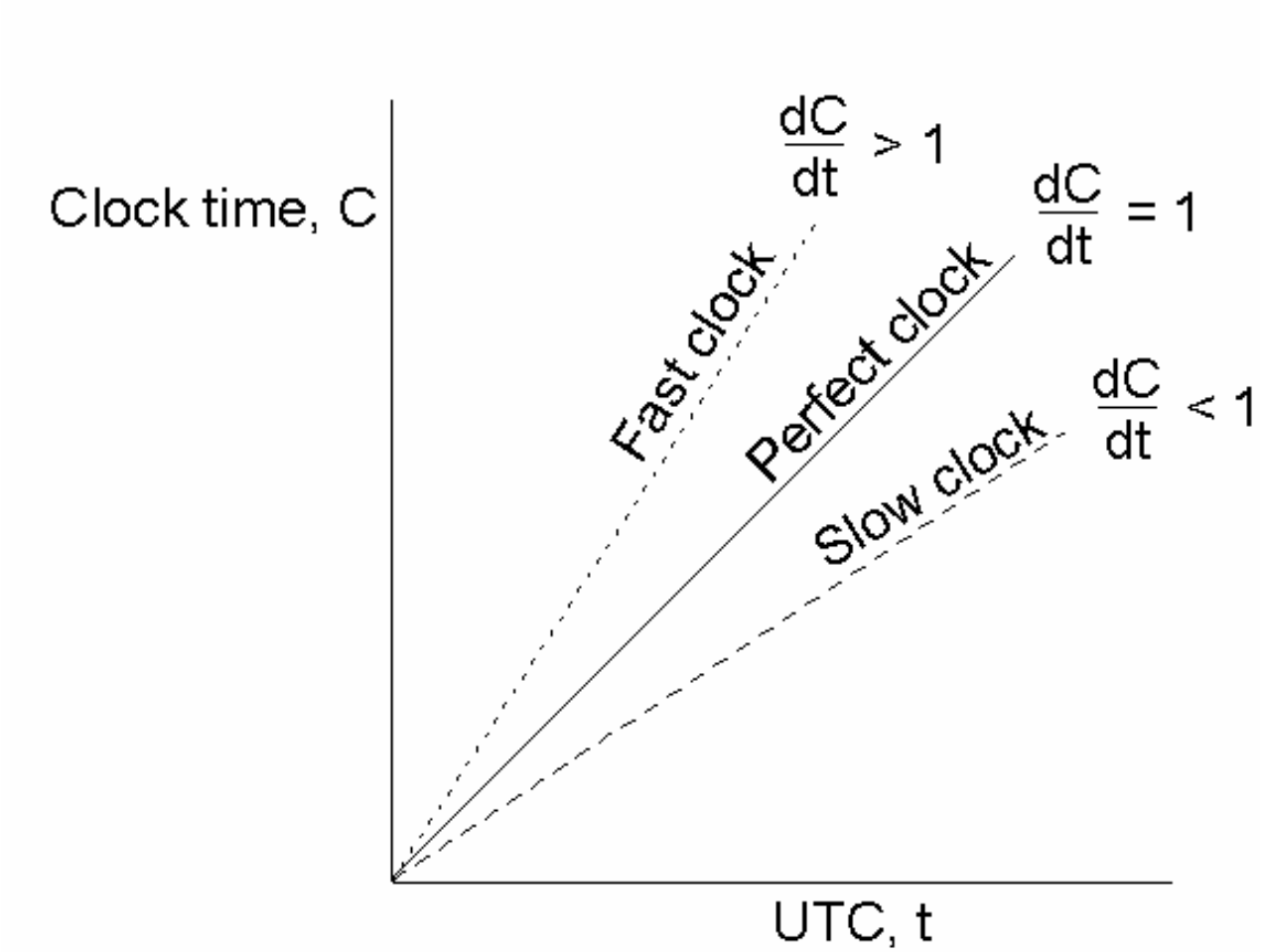
2 Tiempo físico

- Definiciones:
  - Segundo solar o astronómico: 1/86.400 del periodo de rotación de la Tierra (*mean solar second*)
    - pese a ser perfectamente válido para las situaciones de la vida cotidiana, la Tierra no gira a velocidad constante (va perdiendo lentamente velocidad), por lo que no sirve como referencia
  - Segundo atómico (*IAT*, 1967): 9.192.631.770 periodos de transición en un átomo de Cesio-133. Los relojes atómicos miden este tiempo
    - deriva de $3*10^{-8}$ con el segundo solar (~1 s al año)
  - Tiempo universal coordinado (*UTC*): medido en segundos atómicos, sincronizado con tiempo astronómico (diferencia > 900 ms $\Rightarrow$ inserción de 1 s)

2 Tiempo físico

- Definiciones:
  - Tiempo físico de referencia: normalmente *UTC*
  - Resolución: periodo entre dos actualizaciones del registro del tiempo local
    - debe ser menor que el intervalo de tiempo mínimo entre dos eventos producidos consecutivamente en el nodo
  - Desviación (*offset*, *skew*, θ): diferencia entre el tiempo local y el tiempo físico de referencia en un instante
  - Deriva (*drift*, δ): desviación por unidad de tiempo (lo que adelanta o atrasa el reloj)
  - Precisión (*accuracy*): desviación máxima que se puede garantizar en el ajuste de un reloj

Clock Synchronization Algorithms



- The relation between clock time and UTC when clocks tick at different rates.

2.1 Sincronización externa

- Sincronización: procedimiento por el que se ajusta el valor de un reloj a un tiempo físico de referencia con una precisión preestablecida
- Referencias de tiempo *UTC* se difunden periódicamente por radio
  - velocidad de propagación: ~$3*10^8$ m/s (1.000 km: ~3 ms)
  - velocidad variable: pérdida de precisión
- Precisión de receptores comerciales:
  - Estaciones terrestres: 0,1 - 10 ms
  - Satélites geoestacionarios (35.786 km): 0,5 ms
  - Satélites GPS (20.200 km): 1 μs
- Usos: servicio horario preciso, contabilidad…

2.2 Sincronización interna

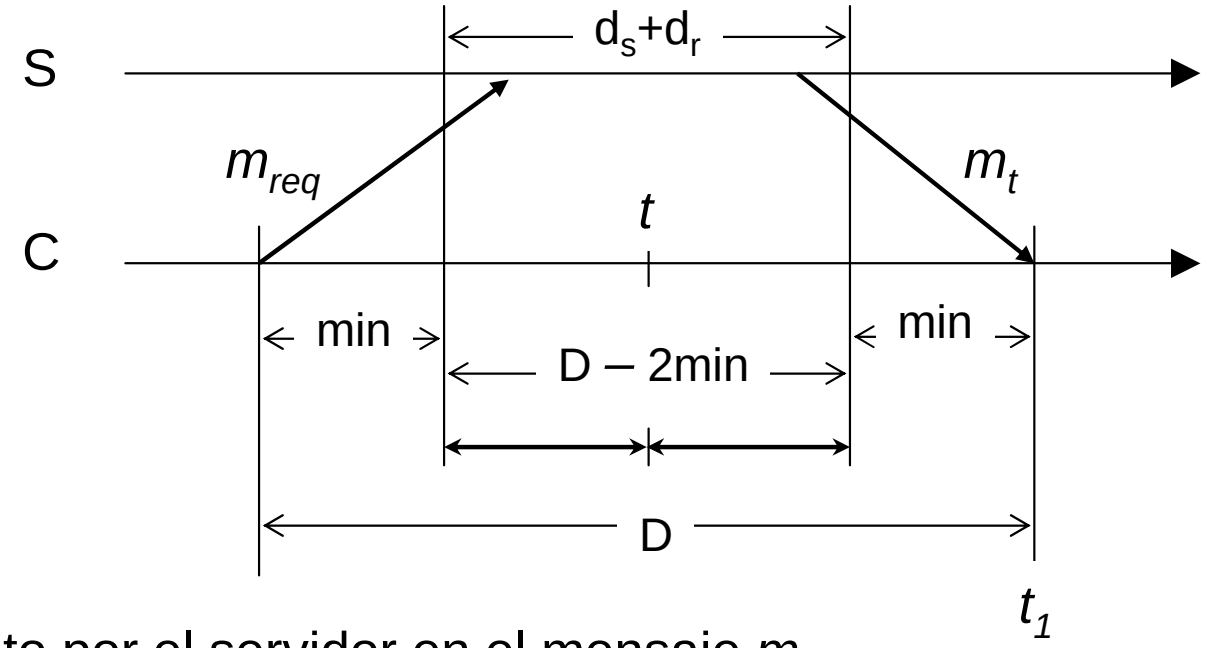
- Para muchas aplicaciones es más importante mantener bien sincronizados entre sí los relojes locales que conseguir una gran precisión en la sincronización externa
  - permite ordenar eventos (causalidad)
- Propuesta de solución: receptor *UTC* en cada nodo
  - no es práctico (coste)
- Solución: algoritmos de sincronización interna
  - centralizados: basados en un servidor específico
  - distribuidos: estadísticos

2.2.1 Algoritmos centralizados

- Principio:
  - utiliza un servidor de tiempos que proporciona referencias fiables (por ejemplo, un servidor que se sincroniza periódicamente con *UTC*)
  - el cliente envía una petición al servidor (m_{req})
  - el servidor contesta enviando la referencia de tiempo en otro mensaje (m_t)
  - retardo de los mensajes: *min* + *d*
    - *min* es el tiempo mínimo de transmisión. Es constante y se puede determinar a partir de las características de la red
    - *d* es variable y depende de la carga del sistema y de la red
- Ejemplo: algoritmo de *Cristian* (1989)

2.2.1 Algoritmos centralizados

Algoritmo de *Cristian*



$t(m_t)$: tiempo devuelto por el servidor en el mensaje m_t

D: tiempo desde que se envía m_{req} y se recibe m_t

min: tiempo mínimo de transmisión de un mensaje

Suposición: el servidor asigna la referencia $t(m_t)$ en la mitad del intervalo *D-2*min* (instante *t* en el cliente). Así se logra la mejor precisión

Desviación: $\theta = t - t(m_t) = t1 - D/2 - t(m_t)$

Precisión = $D/2 - min$

2.2.1 Algoritmos centralizados

- Ejemplo del algoritmo de *Cristian* (3 peticiones):

Pet.	*D (ms)*	*t_1 (hh:mm:ss.ms)*	*$t(m_t)$ (hh:mm:ss.ms)*
(1)	*22*	*10:54:22.236*	*10:54:23.674*
(2)	*26*	*10:54:24.000*	*10:54:25.450*
(3)	*20*	*10:54:26.946*	*10:54:28.342*

  - ¿Qué petición debería usar el cliente? Calcula la precisión y desviación obtenidas

    Precisión = 10 *ms* Desviación = 1406 *ms*

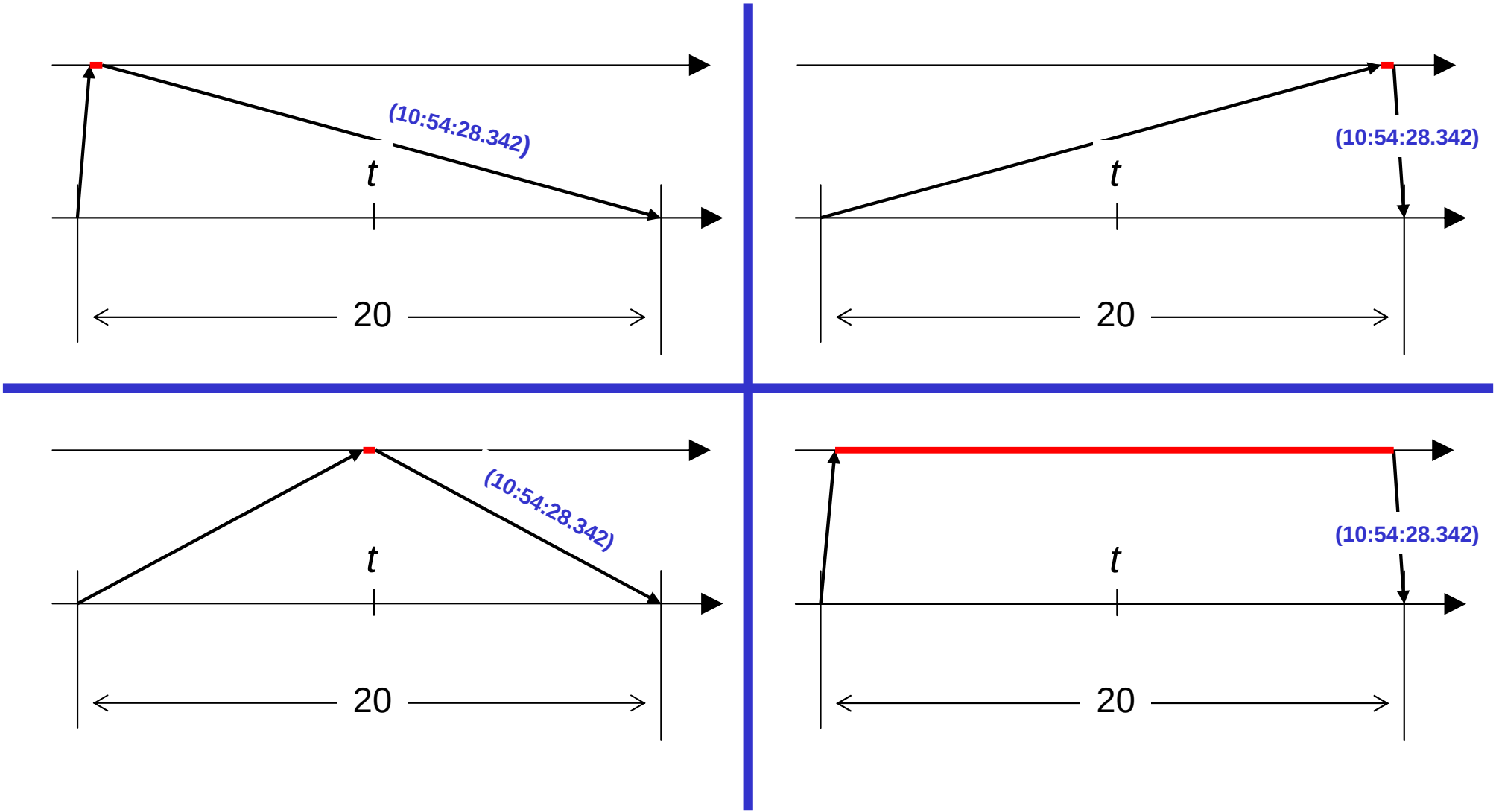
  - Conociendo que el tiempo mínimo de transmisión es de 7 ms, ¿cambia en algo la respuesta anterior?

    Precisión = 3 *ms*

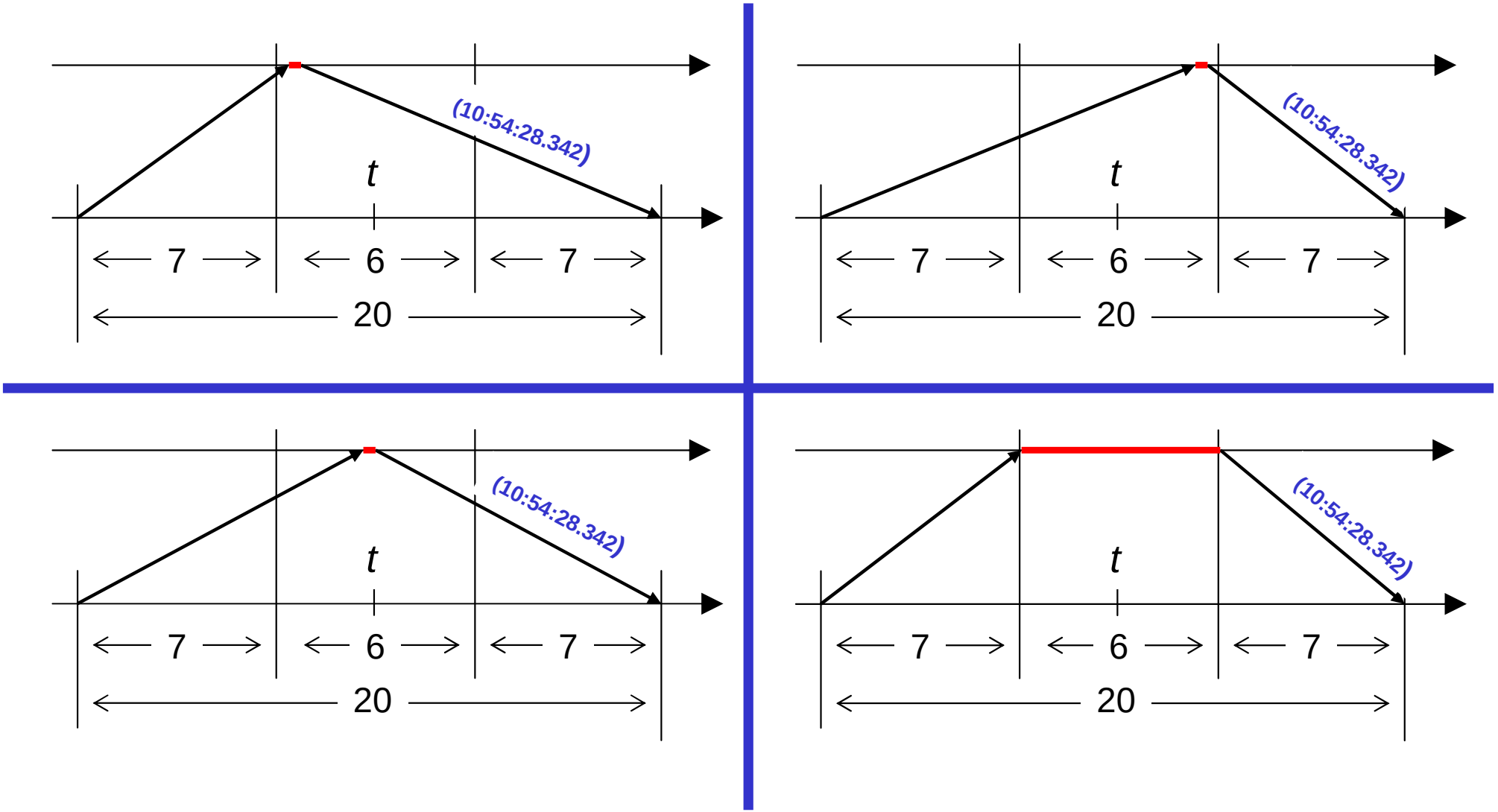
  - ¿Qué se necesitaría para obtener una precisión de 2 ms?

    Siendo min = 7 *ms*, una petición con *D* ≤ 18 *ms*

2.2.1 Algoritmos centralizados



2.2.1 Algoritmos centralizados



2.2.2 Algoritmos distribuidos

- Principio estadístico de los algoritmos distribuidos:
  - las diferentes desviaciones de los tiempos locales tienden a cancelarse entre ellas en torno a un tiempo medio que se puede tomar como referencia fiables para la sincronización interna
- Ejemplo: algoritmo de *Berkeley* (1989)
  - el coordinador solicita los tiempos a los demás nodos
  - el coordinador calcula un tiempo medio a partir de las medidas obtenidas, excluyendo las respuestas espurias y las que no llegan en plazo

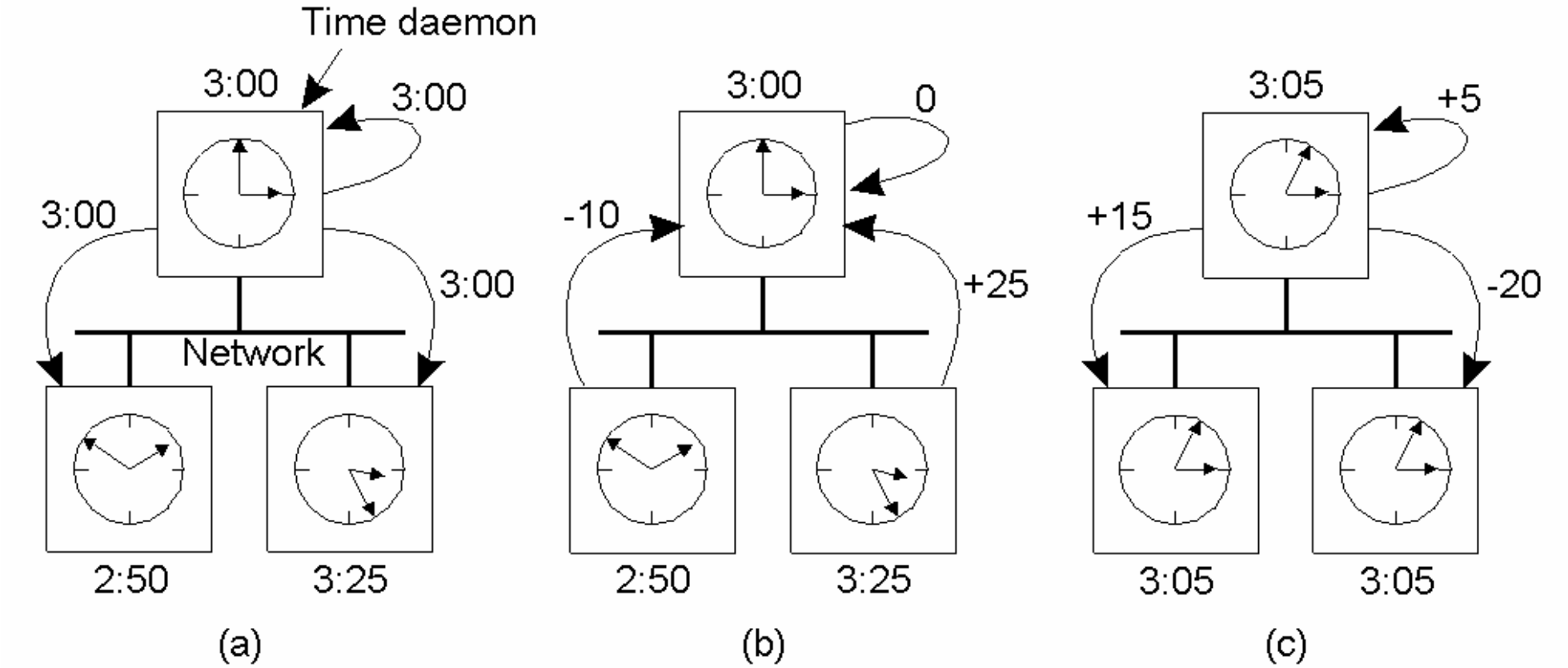
$$\mathbf{T = \Sigma(t(m_i) - D_i/2) / N}$$

  - el coordinador envía a cada nodo su desviación con respecto a la media, para que realice el ajuste

$$\mathbf{\theta_i = t(m_i) - D_i/2 - T}$$

  - si el coordinador falla, se elige un nuevo coordinador

The Berkeley Algorithm



a) The time daemon asks all the other machines for their clock values

b) The machines answer

c) The time daemon tells everyone how to adjust their clock

2.2.2 Algoritmos distribuidos

- Ejemplo del algoritmo de *Berkeley*:

	Nodo	*D (ms)*	*t (hh:mm:ss.ms)*	***Desviaciones***
	N1 (coord.)	*0*	*10:54:23.118*	***-948 ms***
	N2	*22*	*10:54:22.236*	***-1842 ms***
	N3	*26*	*10:54:24.000*	***-79 ms***
Excluido!	*N4*	*190*	*10:41:46.179*	***-757983 ms***
	N5	*20*	*10:54:26.946*	***+2870 ms***

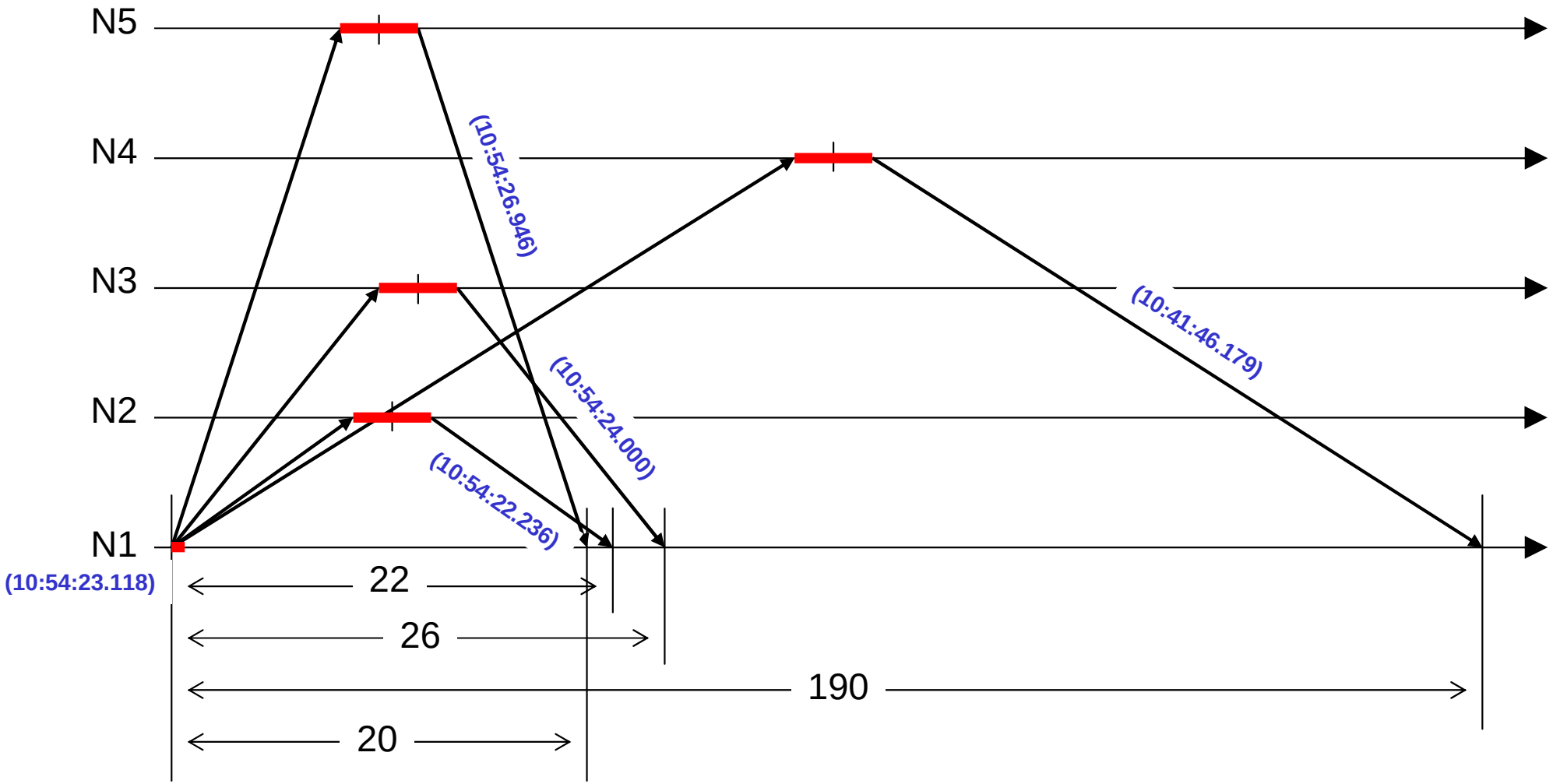
  - Calcula el tiempo medio para la sincronización y la desviación a enviar a cada nodo

    Tiempo medio = 10:54:24.067

  - Usando el método de *Cristian*, calcula la precisión en el ajuste de cada nodo al sincronizarse (*min* no se conoce)

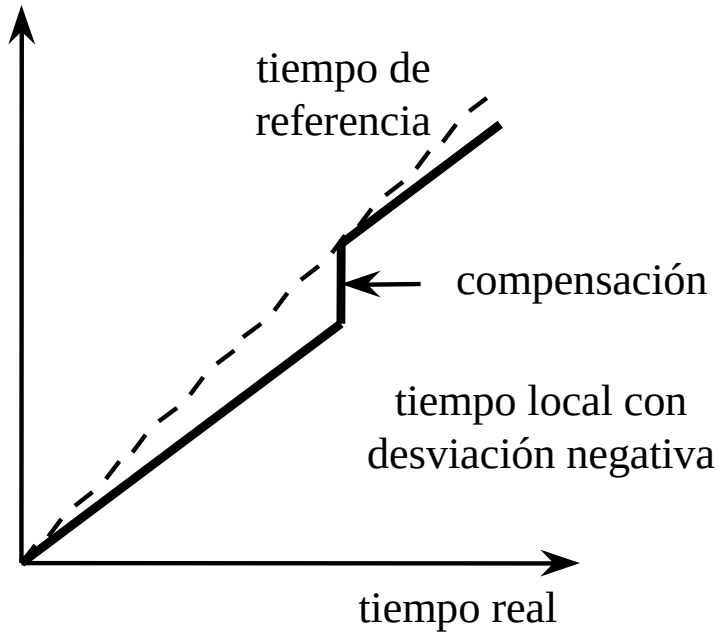
    D/2

2.2.2 Algoritmos distribuidos



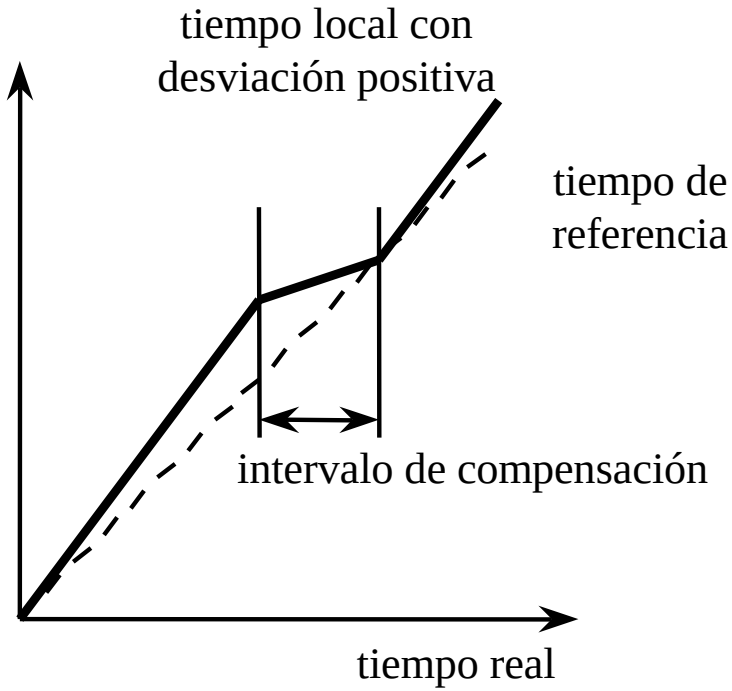
2.3 Compensación de desviaciones

- Propiedad fundamental: <u>monotonicidad creciente</u>



(a)

Desviación negativa

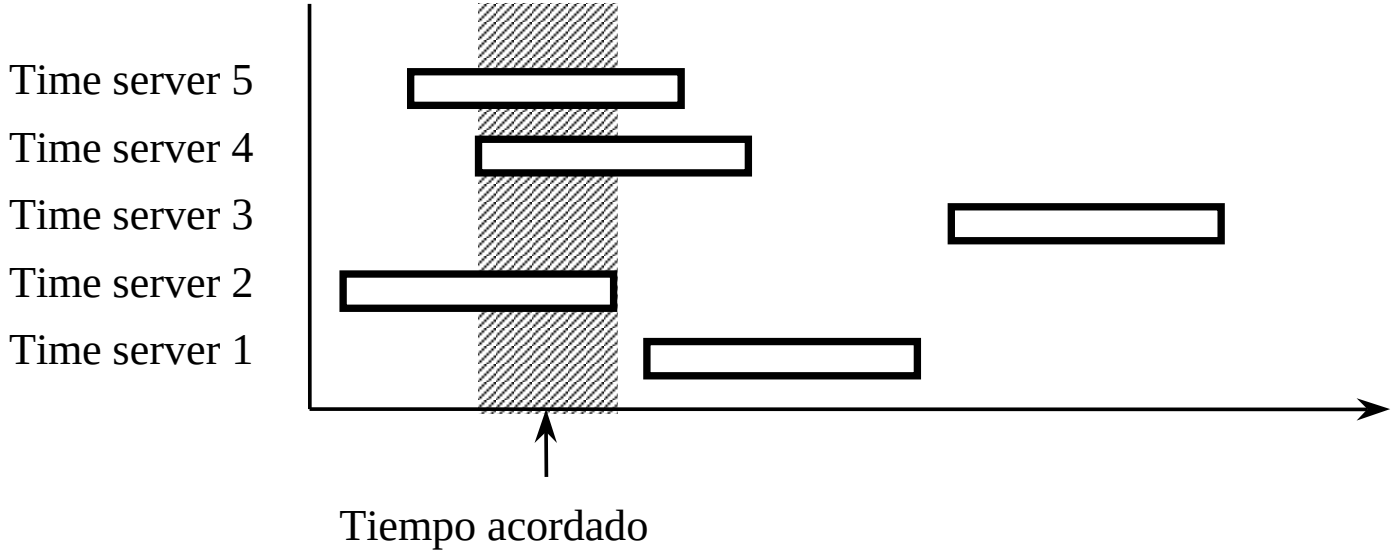


(b)

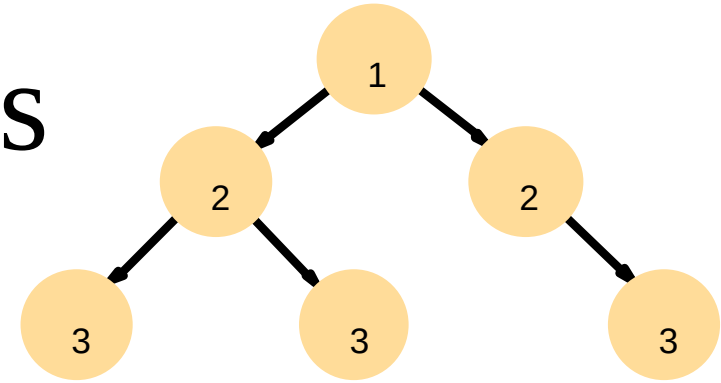
Desviación positiva

2.4 Ejemplos

- *Distributed Time Server (DTS)*
  - algoritmo distribuido
  - servidores (*time servers*): *UTC*
  - clientes (*time clerks*)



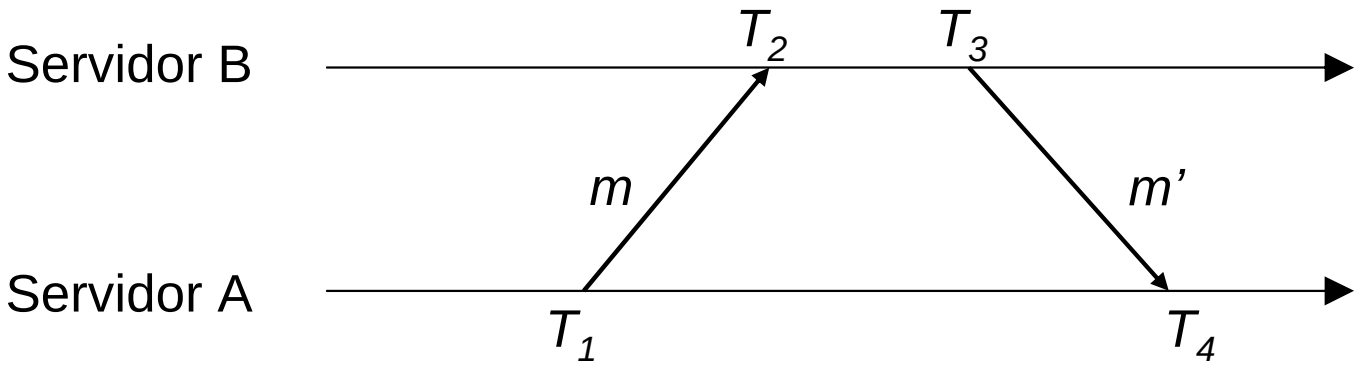
2.4 Ejemplos



- *Network Time Protocol (NTP)*
  - estándar en Internet
  - proporciona sincronización redundante con *UTC*
  - Estructurado en capas (*strata*) de servidores de tiempo
    - servidores primarios (*strata* 1): referencias *UTC* fiables
    - servidores de nivel 2 (*strata* 2): sincronizados con primarios
    - ...
  - Modos de operación:
    - red local con soporte adecuado: modo *multicast*
    - mayor precisión: modo de llamada a procedimiento
    - mejor sincronización interna: modo simétrico
  - La red de servidores se puede reconfigurar

2.4 Ejemplos

- *Network Time Protocol (NTP):* modo simétrico



$T_2 = T_1 + t + \theta$
$T_4 = T_3 + t' - \theta$

Hipótesis: A retrasado respecto a B (θ)

$t, t' \geq 0$ son los retardos de los mensajes m y m'

$d_i = t + t' = T_2 - T_1 + T_4 - T_3 = (T_4 - T_1) - (T_3 - T_2)$

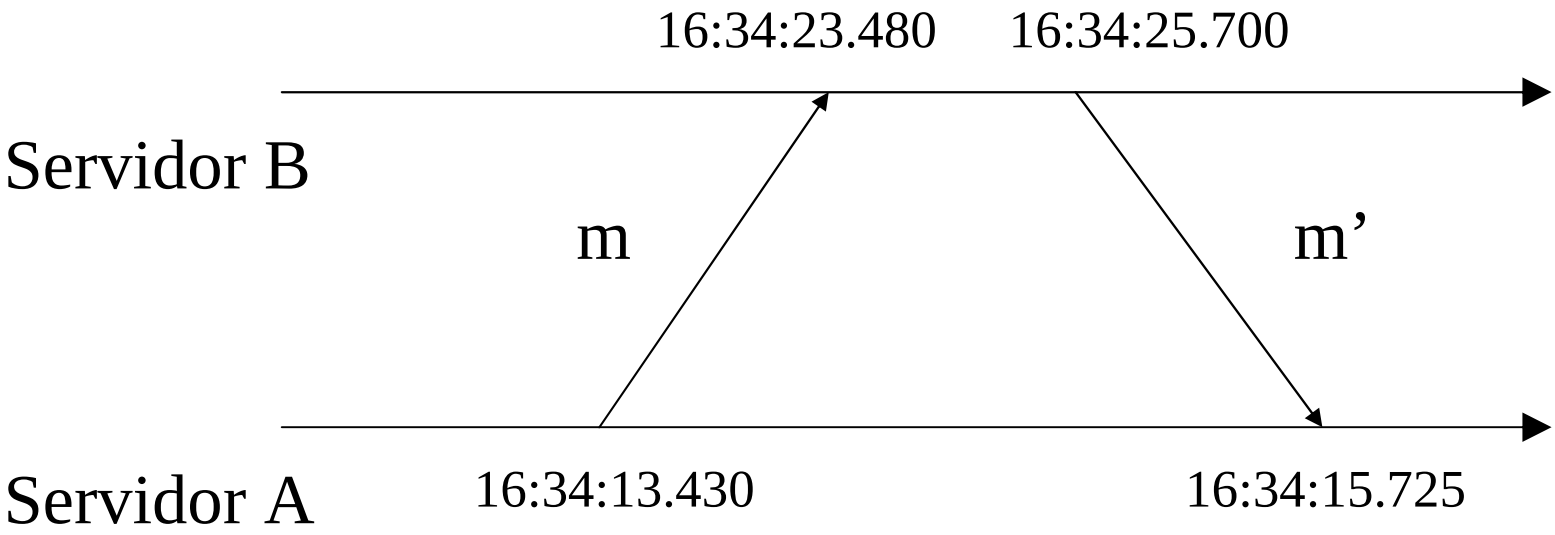
$\theta = \theta_i + (t' - t) / 2$, non $\theta_i = (T_2 - T_1 + T_3 - T_4) / 2$

$\theta_i - (d_i / 2) \leq \theta \leq \theta_i + (d_i / 2)$

θ_i estimación de la desviación; $(d_i / 2)$ precisión

2.4 Ejemplos

- *Network Time Protocol (NTP):* ejercicio



  - ¿Estimación de la desviación?
  - ¿Precisión? **$t + t' = 0.075$, $\theta_i = 10.013$**

$\theta = 10.013 \pm 0.038$ (9.975 .. 10.051)

3 Tiempo lógico y orden de eventos

- A veces la precisión obtenida al sincronizar los relojes no nos permite usar el tiempo físico para ordenar los eventos de los diferentes nodos de un sistema distribuido
  - siempre es posible ordenar los eventos de un mismo nodo (si se respeta la monotonicidad del reloj)
  - en cambio, relaciones de causalidad entre eventos de nodos diferentes pueden verse distorsionadas
- Muchas aplicaciones requieren únicamente ordenar los eventos (y no tanto conocer el instante exacto en que ocurrieron)

3.1 Modelo de eventos

- Modelo de sistema: conjunto de procesos que comunican únicamente mediante paso de mensajes

  enviar (p_i, *mensaje*)

  recibir (p_j, *mensaje*)

  p_i y p_j son el emisor y el receptor del mensaje

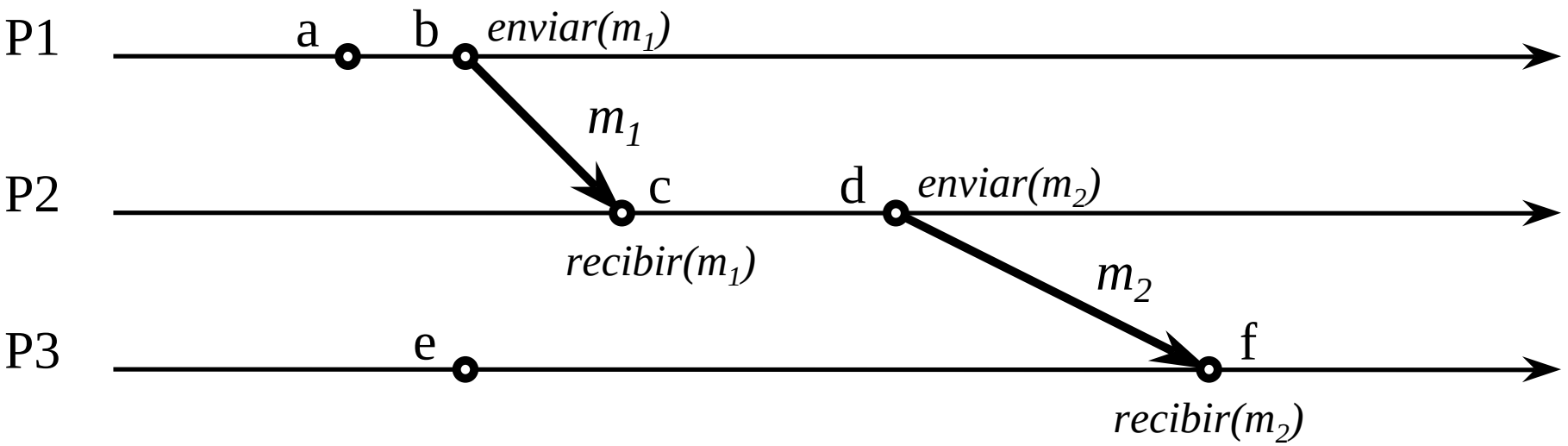
- Simplificación: un proceso por nodo/máquina
- Cada proceso genera una secuencia de eventos. Identificamos tres tipos de eventos:
  - Envío de un mensaje (al ejecutar *enviar*)
  - Recepción de un mensaje (al ejecutar *recibir*)
  - Eventos locales/internos (resto de eventos, sin comunicación)

3.1 Modelo de eventos

- Para ordenar los eventos de un mismo proceso bastaría con asociar a cada evento *x* el tiempo local *T(x)* (si la resolución es suficiente)
- Se dice que existe una relación de causalidad entre dos eventos del sistema ($x \rightarrow y$, "*x ha sucedido antes que y*", "*x happened before y*") si:
  1. *x* e *y* son eventos del mismo proceso y $T(x)<T(y)$
  2. *x* e *y* son los eventos *enviar(m)* y *recibir(m)* del mismo mensaje *m*
  3. Existe otro evento *z* tal que $x \rightarrow z$ y $z \rightarrow y$ (cierre transitivo de la relación)

3.1 Modelo de eventos

- Si entre dos eventos no hay relación de causalidad, se dice que son concurrentes: $x \,||\, y$



$$a \rightarrow b \quad b \rightarrow c \quad a \rightarrow c \quad a \rightarrow f \quad a \,||\, e \quad e \,||\, d$$

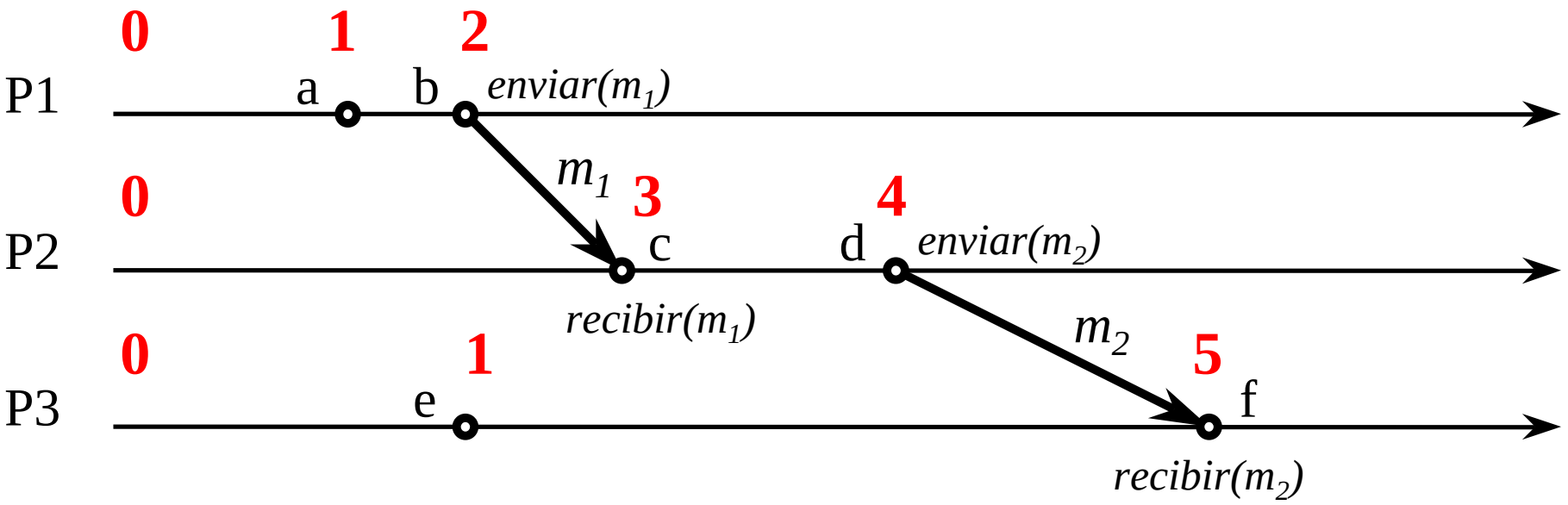
3.2 Relojes lógicos de Lamport

- Propuesto en 1978, para indicar relaciones de causalidad. Cada proceso P_i tiene su reloj lógico local C_i, para asociar marcas de tiempo a sus eventos (un simple contador asíncrono basta)
- Algoritmo:
  - Inicialmente, $C_i = 0, \ \forall i$
  - Antes de un evento local o envío de mensaje en p_i: $C_i = C_i + 1$
  - Cuando p_j envía un mensaje m a p_i, p_j incluye el valor de su reloj lógico en el mensaje, C_m. Al recibir dicho mensaje, p_i actualiza su reloj local de la siguiente manera:

$$(1)\ C_i = max(C_i, C_m), \quad (2)\ C_i = C_i + 1$$

  - Problema: $C_i(x) < C_j(y)$ no implica $x \rightarrow y$

3.2 Relojes lógicos de Lamport



$$a \rightarrow b \quad b \rightarrow c \quad a \rightarrow c \quad a \rightarrow f \quad a \,||\, e \quad e \,||\, d$$

3.3 Vectores de tiempos

Garantizan: $V_i(x) < V_j(y) \Leftrightarrow x \rightarrow y$

- V: vector de N componentes (N=número de procesos)
- $V_i[i]$: reloj lógico (local) del proceso P_i
- $V_i[j]$: último valor que el proceso P_i conoce del reloj del proceso P_j

- Algoritmo:
  - Inicialmente, $V_i[j] = 0, \ \forall i, j$
  - Evento local o envío de mensaje en p_i: $V_i[i] = V_i[i] + 1$
  - Cuando p_j envía un mensaje m a p_i, p_j incluye el valor de su vector de tiempos, V_m. Al recibir dicho mensaje, p_i actualiza su vector de tiempos de la siguiente manera:

    (1) $\forall k: V_i[k] = max(V_i[k], V_m[k])$, (2) $V_i[i] = V_i[i] + 1$

3.3 Vectores de tiempos

- Definición:

$$V1 < V2 \Leftrightarrow \forall i\ V1[i] \leq V2[i] \ \underline{and} \ \exists j\ V1[j] < V2[j]$$

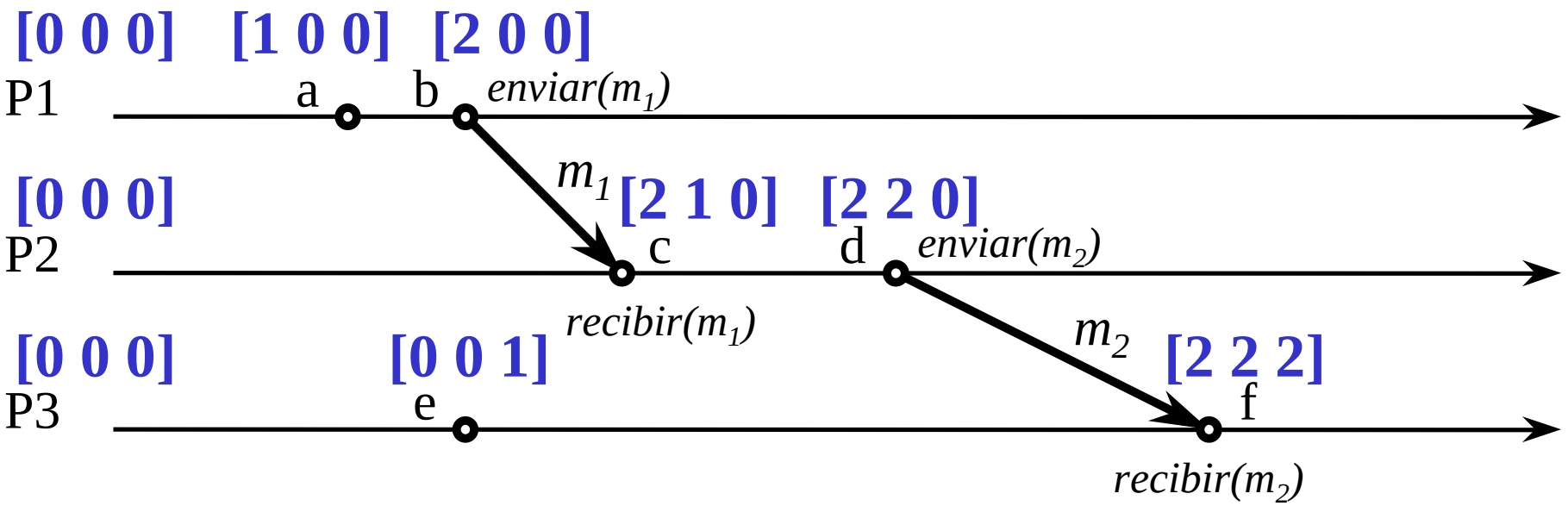
- Existirá relación de causalidad entre dos eventos *(x, y)* si y sólo si

$$V(x) < V(y) \text{ o } V(y) < V(x)$$

- Si no, los eventos son concurrentes
- Los vectores de tiempos permiten determinar relaciones de causalidad con total precisión:

$$V_i(x) < V_j(y) \Leftrightarrow x \rightarrow y$$

3.3 Vectores de tiempos



$$a \rightarrow b \quad b \rightarrow c \quad a \rightarrow c \quad a \rightarrow f \quad a \,||\, e \quad e \,||\, d$$

4 Estado global y consistencia

- Modelo de sistema:
  - N procesos P_i (i = 1, 2, ..., N)
  - historia(P_i) = h_i = < e_i^0, e_i^1, e_i^2, ... >
  - prefijo de historia h_i^k = < e_i^0, e_i^1, ..., e_i^k >
  - s_i^k: estado del proceso P_i tras el evento e_i^k
  - s_i^0: estado inicial del proceso P_i
- Estado global del sistema: S = (s_1, s_2, ..., s_N)
  - Algunos estados globales pueden ser “inconsistentes”
- Corte: unión de prefijos de historia de los procesos
  - C = $h_1^{c1} \cup h_2^{c2} \cup ... \cup h_N^{cN}$ = (c_1, c_2, ..., c_N)

4 Estado global y consistencia

- Frontera de un corte: unión de los últimos eventos

$$(c_1, c_2, ..., c_N) \Rightarrow (e_1^{c1}, e_2^{c2}, ..., e_N^{cN})$$

- A cada corte le corresponde un estado global

$$(c_1, c_2, ..., c_N) \Rightarrow (s_1^{c1}, s_2^{c2}, ..., s_N^{cN})$$

- Un corte C es consistente si para todo par de eventos (e, e') se cumple lo siguiente:

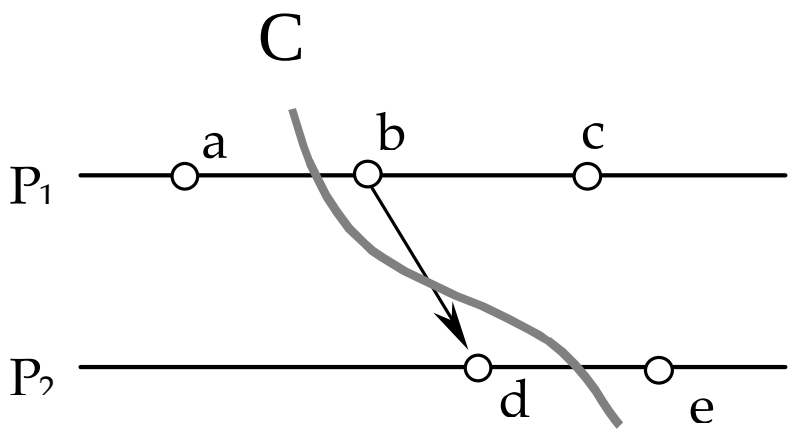
$$(e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C$$

  - Si no, el corte C es inconsistente

- A cada corte consistente le corresponde un estado global consistente

4 Estado global y consistencia

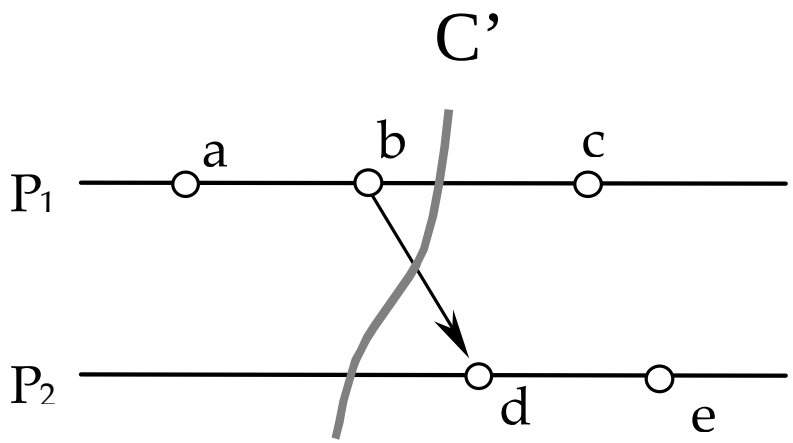
- Corte consistente: si el evento de recepción de un mensaje está “dentro” del corte, entonces el evento de envío de dicho mensaje también debe estar



(a)

inconsistente

$d \in C, b \rightarrow d, b \notin C$



(b)

consistente

4 Estado global y consistencia

- Para saber si un corte es consistente, nos podemos basar en los vectores de tiempos:

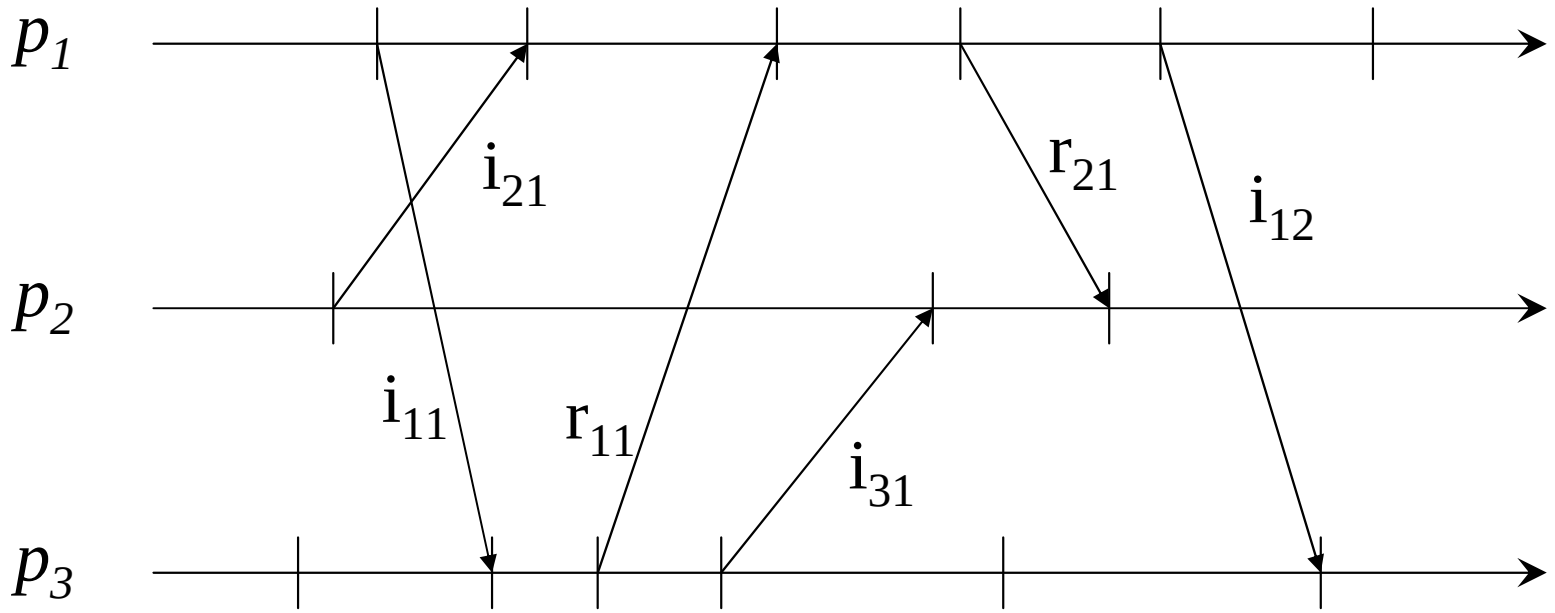
$$\forall i, j: V_i[i](e_i^{ci}) \geq V_j[i](e_j^{cj})$$

- Puesto que cada proceso posee una visión parcial del sistema, para construir un corte consistente (y obtener de paso su estado global asociado) los procesos deben ejecutar un algoritmo distribuido
  - Ejemplo: algoritmo de Chandy y Lamport (1985)
- Utilidad: detección de interbloqueos, establecimiento de puntos de recuperación de un sistema, finalización distribuida

4 Estado global y consistencia

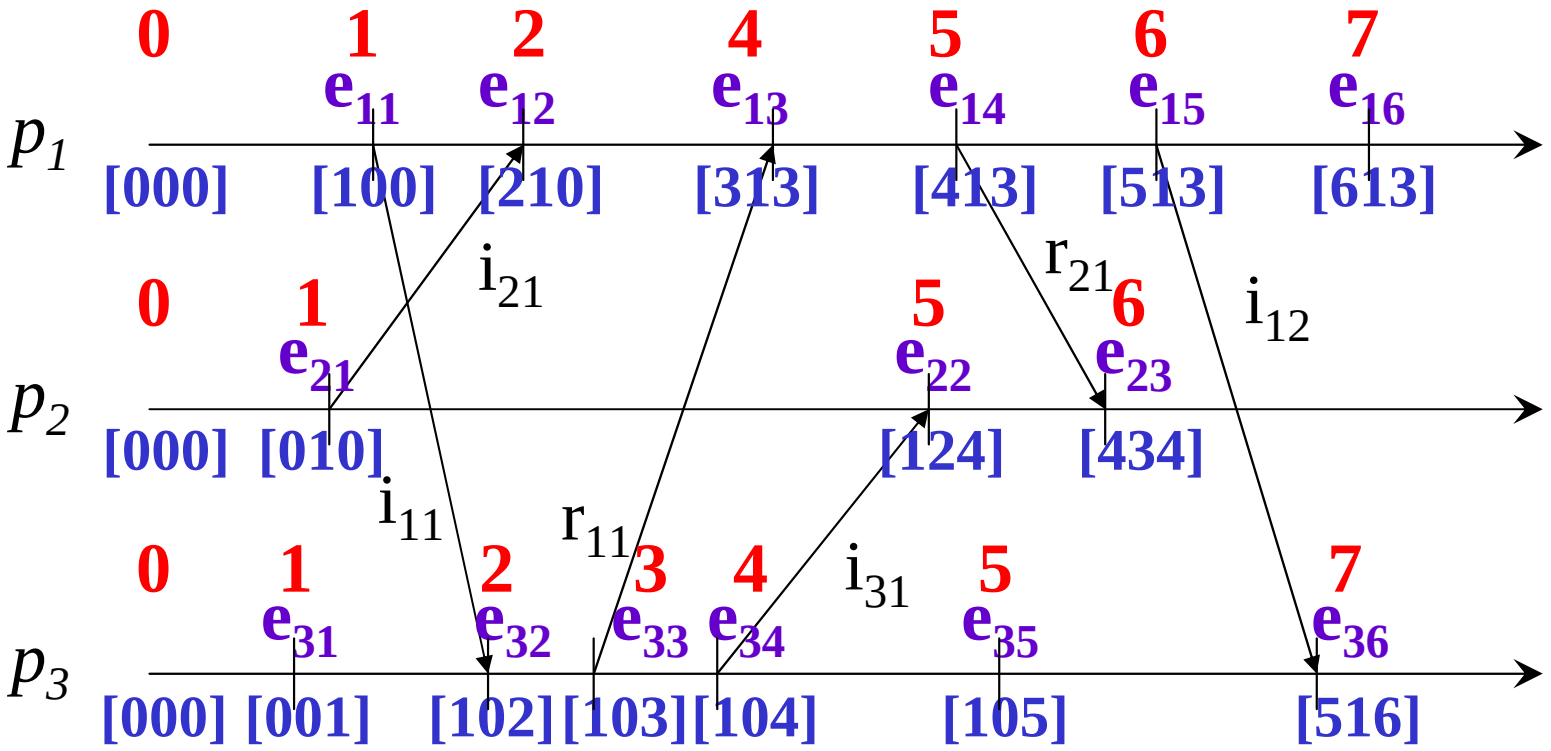
- Algoritmo *snapshot* de Chandy y Lamport:
  1. process p_1 (the initiator of the snapshot) saves its state s_1 and broadcasts the message *SNAPSHOT* to P (the set of processes).
  2. Let process p_i receive the *SNAPSHOT* message the first time from some process p_j (p_j can be different from p_1). At that time, p_i saves its state s_i and forwards the *SNAPSHOT* message to P. The state of the channel c_{ji} is set to empty, and p_i starts logging the messages received on the channels c_{ki} (for all $k \neq j$).
  3. When p_i receives *SNAPSHOT* from p_k, then the computation of the state of the channel c_{ki} is complete. As soon as p_i has received *SNAPSHOT* from all the processes in P, the computation of the snapshot is terminated.

4 Estado global y consistencia



i: petición (*invocation*)
r: respuesta (*reply*)

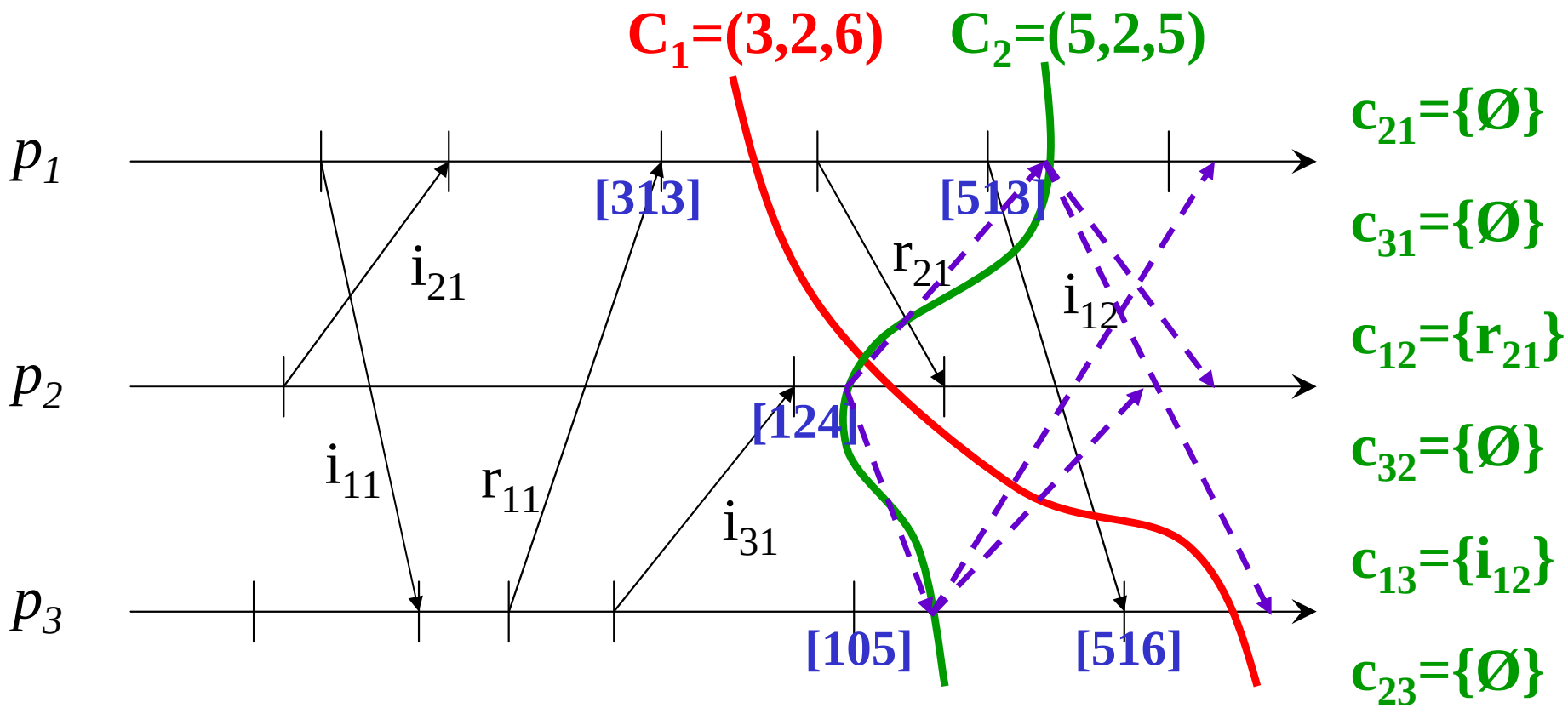
4 Estado global y consistencia



i: petición (*invocation*)

r: respuesta (*reply*)

4 Estado global y consistencia



i: petición (*invocation*)

r: respuesta (*reply*)