Capitulo 1: Introducción a la Computación Distribuida

Leonardo Valdivieso

Escuela Politécnica Nacional

angel.valdivieso@epn.edu.ec

April 3, 2019

1.3 ALGORITMOS DISTRIBUIDOS

1.3.1 Principios de Algoritmos Distribuidos

¿Por qué es difícil la coordinación en sistemas distribuidos?

- El emisor no puede saber:
  - Si el mensaje fue recibido.
  - Si el receptor falló antes o después de procesar el mensaje.

Impossibility of distributed consensus with one faulty process

Michael J. Fischer, Nancy A. Lynch, Michael S. Paterson

Journal of the ACM (JACM)

Volume 32, Issue 2 (April 1985), Pages: 374 - 382

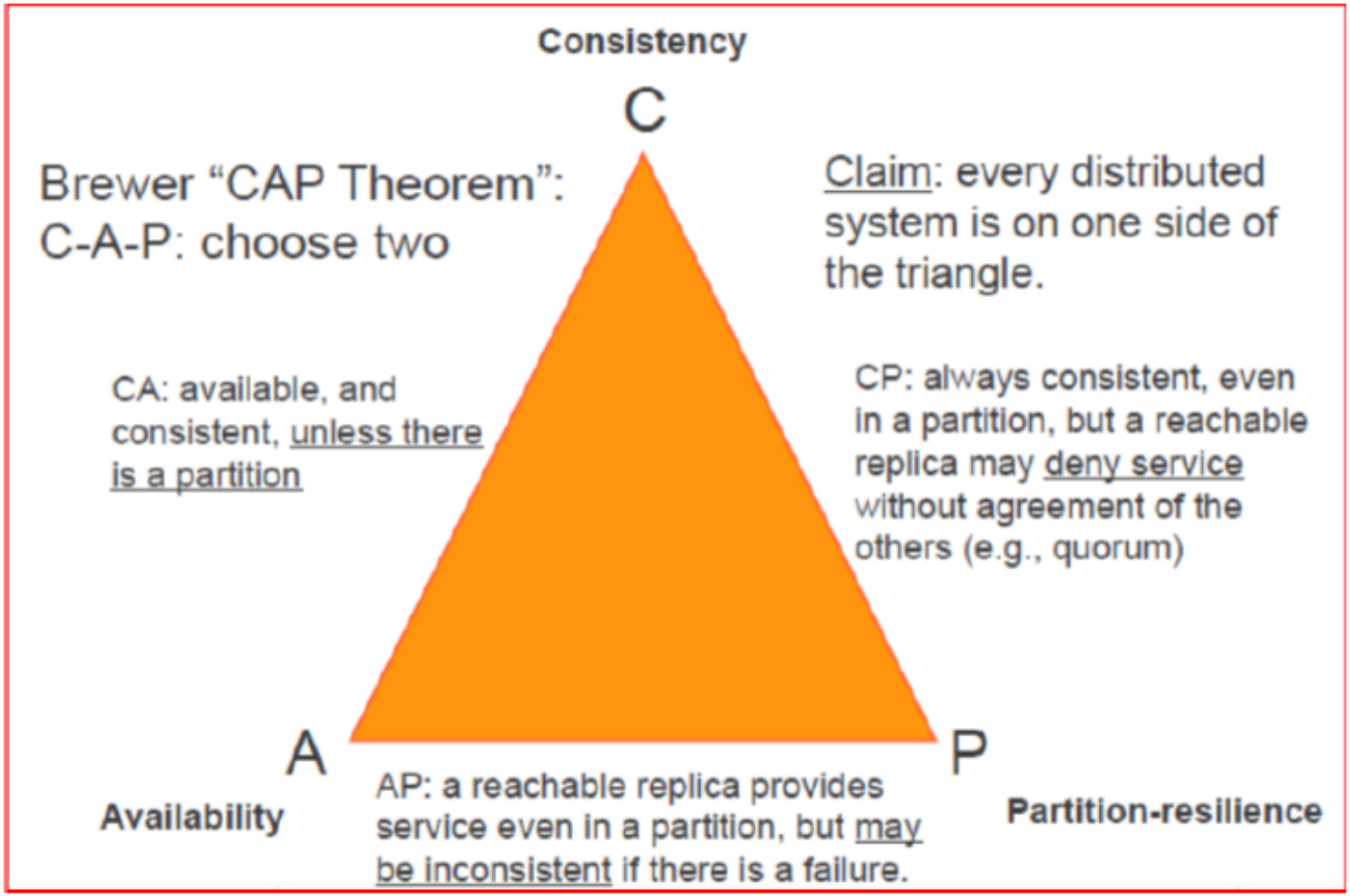
Problemas típicos en sistemas distribuidos

- Sincronización de relojes.
- Exclusión mutua.
- Elección de líder.
- Consenso distribuido.
- Comunicación en grupos.
- Gestión de réplicas.
- Estados globales.

Algoritmos distribuidos

- Algoritmos que trabajan en sistemas distribuidos.
- Realizan tareas:
  - Comunicación
  - Gestión de datos y de recursos
  - Sincronización
  - Consenso
- Deben trabajar bajo:
  - Actividades concurrentes en múltiples localizaciones.
  - Incertidumbre del tiempo, ordenación de eventos,.
  - Posibilidad de fallos (procesos, procesadores, redes)

Teorema CAP


Consistency
C
Brewer "CAP Theorem":
C-A-P: choose two
Claim: every distributed system is on one side of the triangle.
CA: available, and consistent, unless there is a partition
CP: always consistent, even in a partition, but a reachable replica may deny service without agreement of the others (e.g., quorum)
A
P
Availability
AP: a reachable replica provides service even in a partition, but may be inconsistent if there is a failure.
Partition-resilience

Modelos de sistemas distribuidos

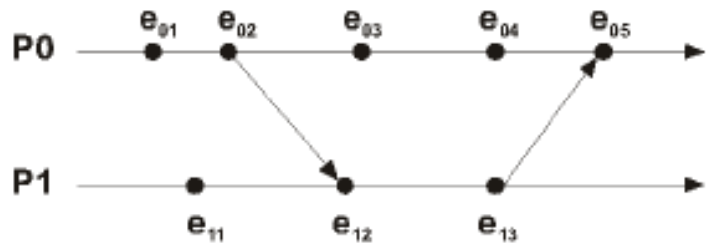
- Modelo síncrono
  - Relojes sincronizados
  - Entrega de mensajes acotada
  - Tiempo de ejecución de procesos acotado
- Modelo asíncrono
  - No hay sincronización de relojes
  - Entrega de mensajes no acotada
  - Tiempor de ejecución de procesos totalmente arbitraria
- Sistemas parcialmente síncronos
  - Tiempos acotados pero desconocidos

Tipos de pasos de mensajes

- Paso de mensajes síncrono
  - El envío y la recepción de un mensaje m, ocurren simultáneamente
- Paso de mensajes asíncrono
  - El envío de un mensaje m y su recepción no están acoplados. No tienen porque ocurrir de forma consecutiva.

Modelo de sistema distribuido

- Modelo de sistema:
  - Procesos secuenciales $\{ P_1, P_2, ...P_n \}$ que ejecutan un algoritmo local.
  - Canales de comunicación.
- Eventos en P_i
  - $E_i = \{ e_{i1}, e_{i2}, ...e_{in} \}$
- Tipos de eventos locales
  - Internos (Cambios en el estado de un proceso)
  - Comunicación (envío, recepción)
- Diagramas espacio-tiempo



Algoritmos locales

- Algoritmo local:
  - Un proceso cambia de un estado al otro (*evento interno*)
  - Un proceso cambia de un estado a otro y envía un mensaje a otro proceso (*evento de envío*)
  - Un proceso recibe un mensaje y cambia su estado (*evento de recepción*)
- Restricciones
  - Un proceso p solo puede recibir un mensaje después de haber sido enviado por otro.
  - Un proceso p puede cambiar de estado c al estado d si está actualmente en el estado c

Algoritmos Locales

- Un sistema distribuido se puede modelar como un sistema de transición de estados STC (C,-,I).
  1. C es un conjunto de estados
  2. $\rightarrow$ describe las posibles transiciones ($\rightarrow \subseteq C \times C$)
  3. I describe los estados iniciales ($I \subseteq C$)

- El estado de un sistema deistribuido, C, se puede describir como:
  - La configuración actual de cada proceso/procesador.
  - Los mensajes de tránsito por la red.

Configuraciones y estados

- En el sistema de transición de estados en un sistema distribuido
  - Los estados se denominan *configuraciones*
  - Las transiciones se denominan transiciones de configuración
- En el sistema de transición de estados de cada proceso
  - Los estados se denominan *estados*
  - Las transiciones se denominan *eventos*
- Una *ejecución* en un sistema distribuido es una secuencia de configuraciones (γ_1, γ_2, γ_3, ...) donde:
  - γ_1 es una configuración inicial,
  - Hay una transición entre $\gamma_1 \rightarrow \gamma_{1+i}$ para todo $i \geq 1$
  - La secuencia puede ser finita o infinita
- Una ejecución o secuencia de estados define el comportamiento del sistema

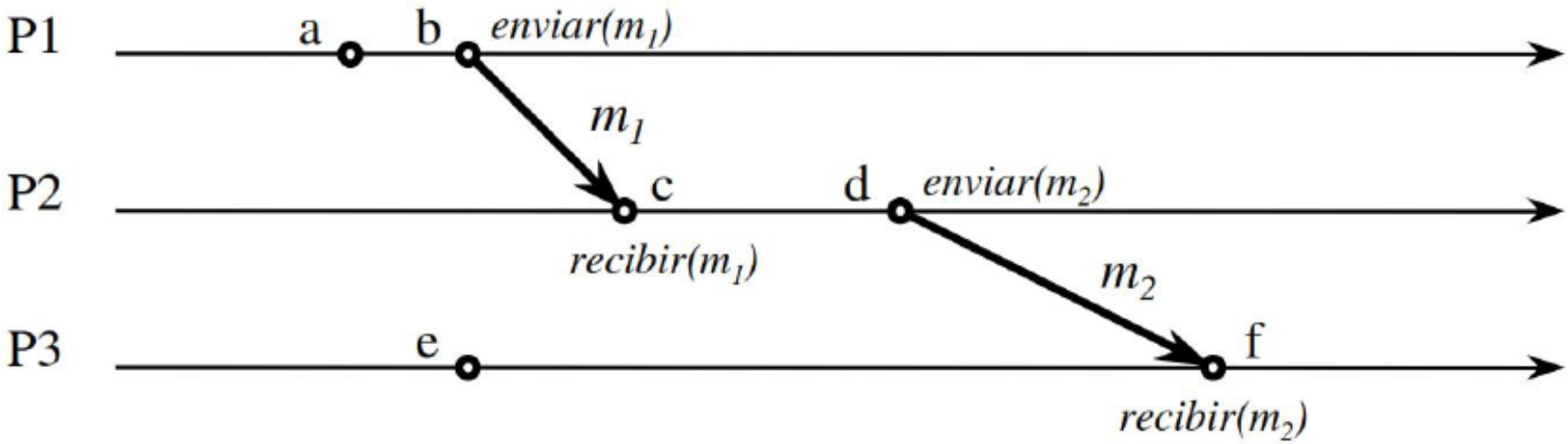
Orden causal

- La relacion $\leq_H$ sobre eventos de una ejecución distribuida, denominada *orden causal*, se define como:
  - Si e ocurre antes que f en el mismo proceso, entonces $e \leq_H f$
  - Si s es un evento de envío y r su correspondiente evento de recepción, entonces $s \leq_H r$
  - $\leq_H$ Es transitiva
    - Si $a \leq_H b$ y $b \leq_H e$ entonces $a \leq_H e$
  - $\leq_H$ Es reflexiva
    - $a \leq_H a$ para cualquier evento
- Dos eventos, a y b, son *concurrentes* si a no $\leq_H b$ y $b \leq_H a$

Toda relación de casualidad establece un nexo entre dos sucesos, uno es la causa y el otro es el efecto.

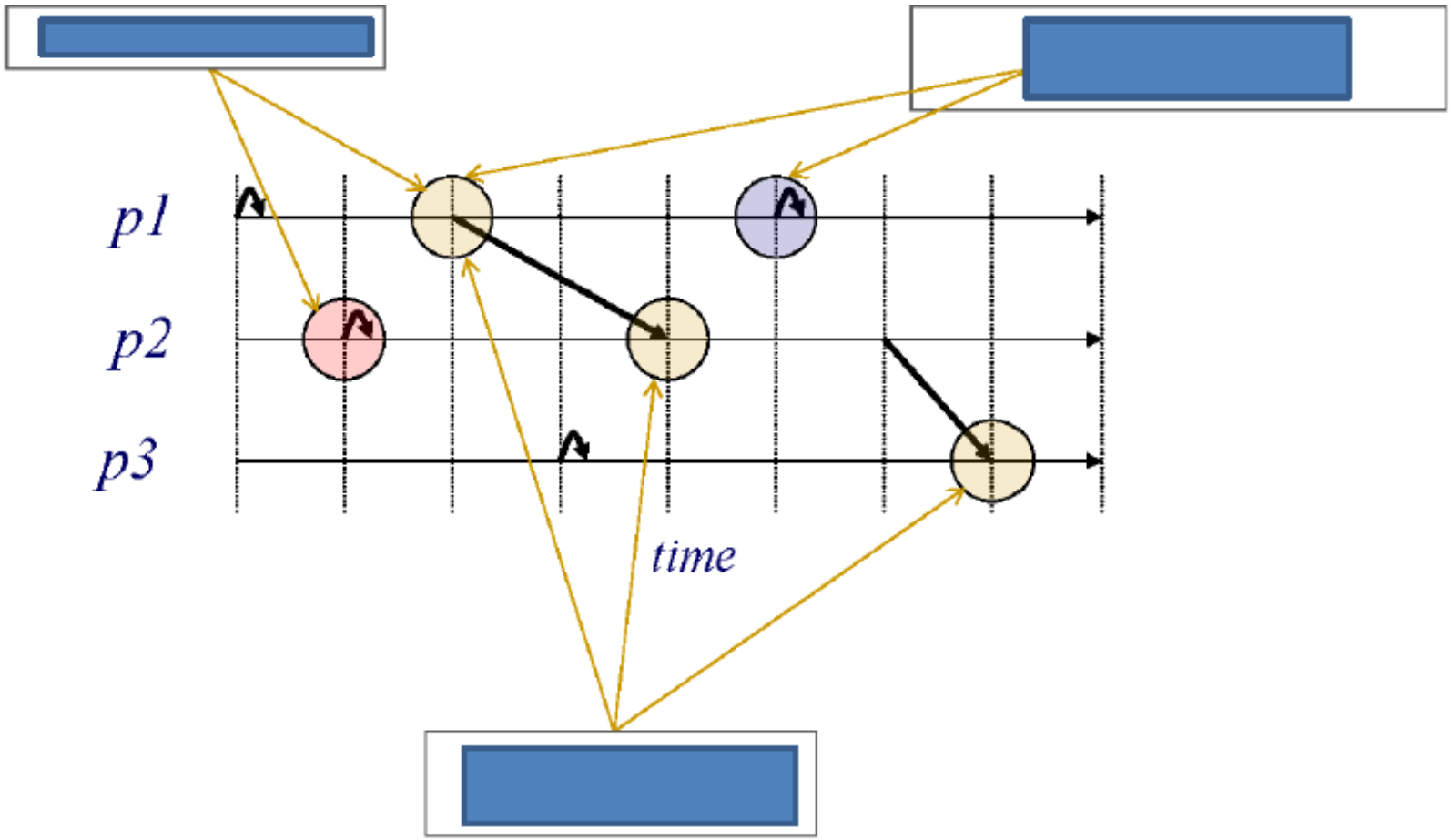
- Para ordenar los eventos de un mismo proceso bastaría con asociar a cada evento x el tiempo local $T(x)$ (si la resolución es suficiente)
- Se dice que existe una relación de causalidad entre dos eventos del sistema ($x \rightarrow y$, "*x ha sucedido antes que y*", "*x happened before y*") si:
  1. 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)

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

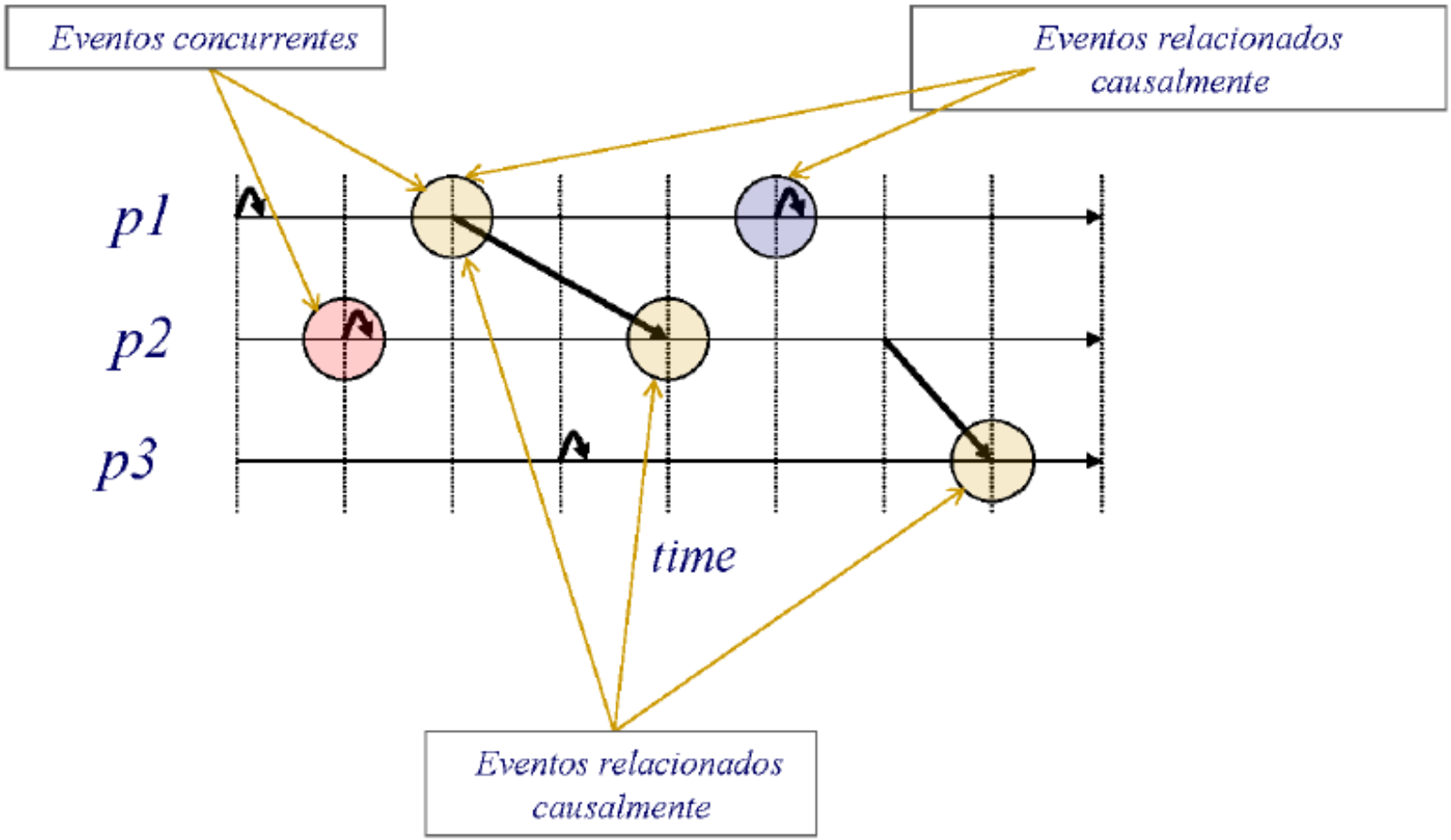


$$a \rightarrow b \quad b \rightarrow c \quad a \rightarrow c \quad a \rightarrow f \quad a \parallel e \quad e \parallel d$$

Ejemplos de eventos



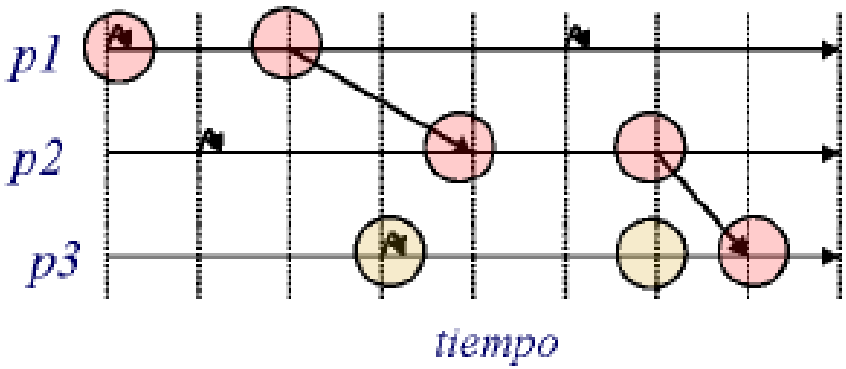
Ejemplos de eventos



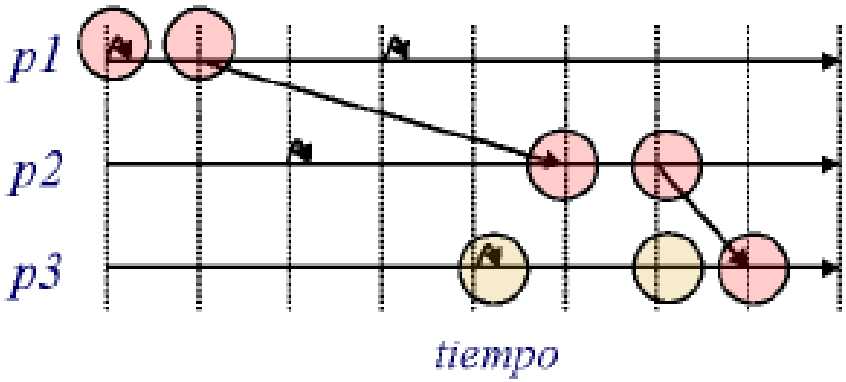
Ejecuciones equivalentes

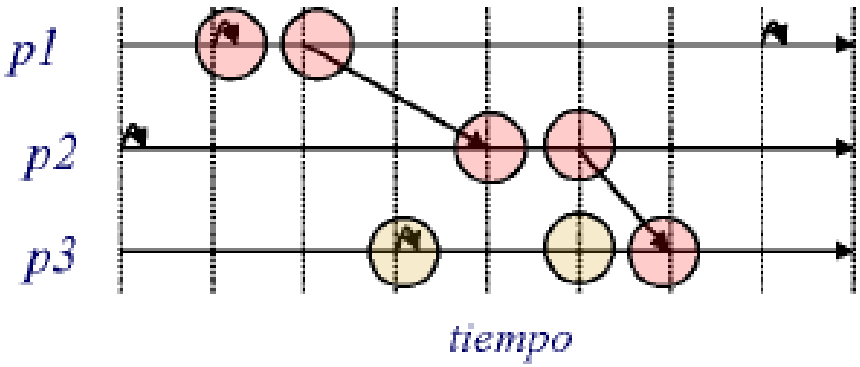
- Dos ejecuciones distribuidas F y E son equivalentes ($F \sim E$) si:
  - Tienen el mismo conjunto de eventos
  - Se mantiene el orden causal

Ejemplos de ejecuciones equivalentes



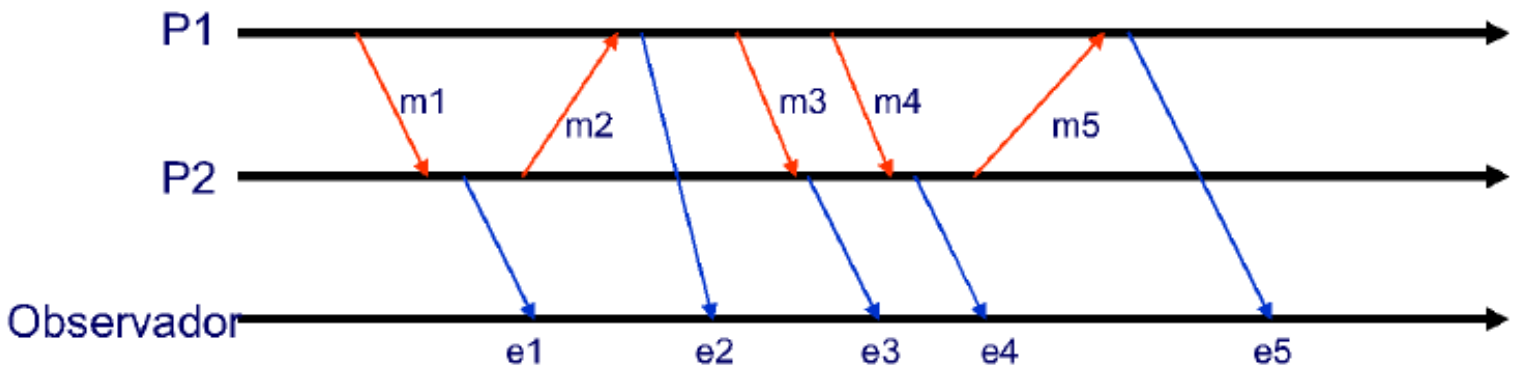
Mismo color ~ orden causal





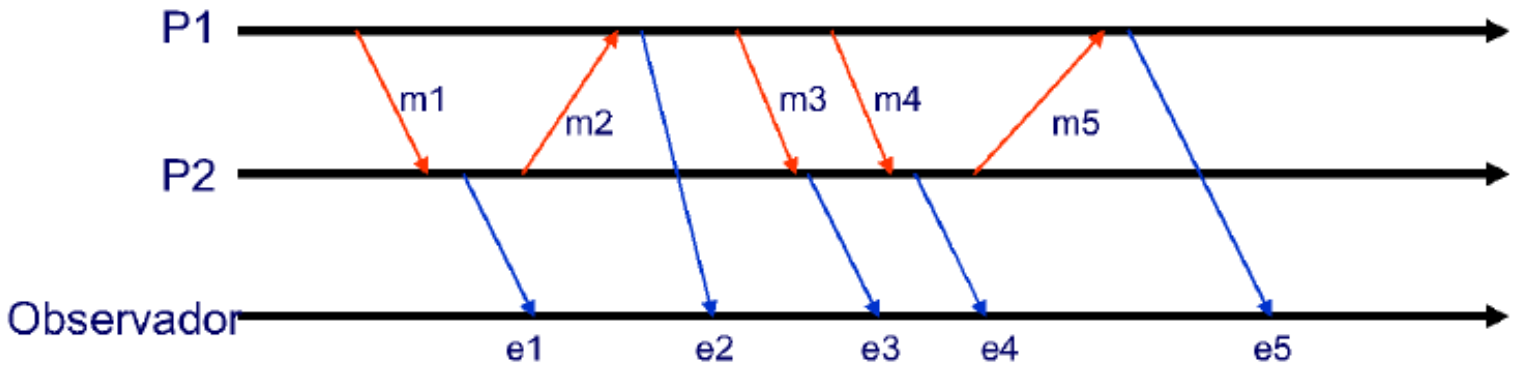
Ordenación de eventos

- Monitorización del comportamiento de una aplicación distribuida.
  - Ejemplo: el observador debe ordenar los eventos de recepción de mensajes en los procesos P1 y P2
    - e1, e2, e3, e4, e5



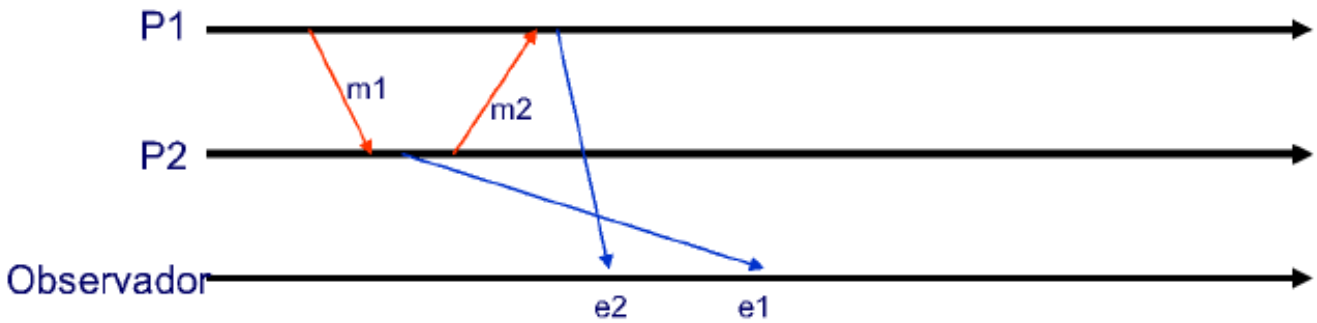
Ejemplo

- Monitorización del comportamiento de una aplicación distribuida.
  - Ejemplo: el observador debe ordenar los eventos de recepción de mensajes en los procesos P1 y P2
    - e1, e2, e3, e4, e5

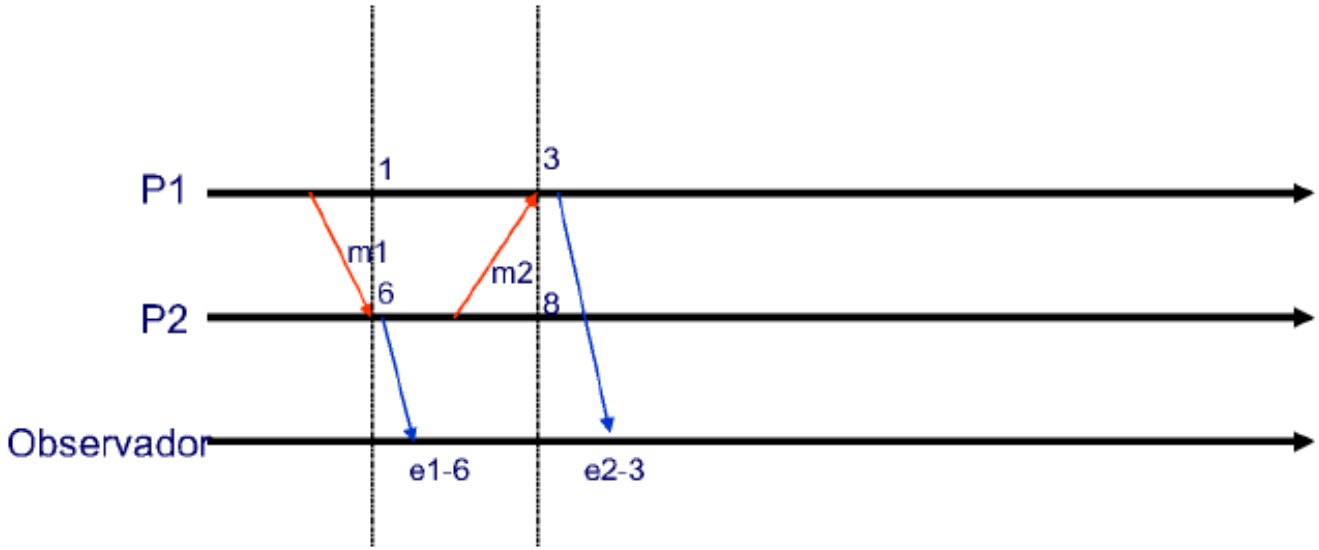


- Para ordenar eventos podemos asignarles marcas de tiempo
  - ei $\leftarrow$ ek $\Leftrightarrow$ MT(ei) $<$ MT(ek)

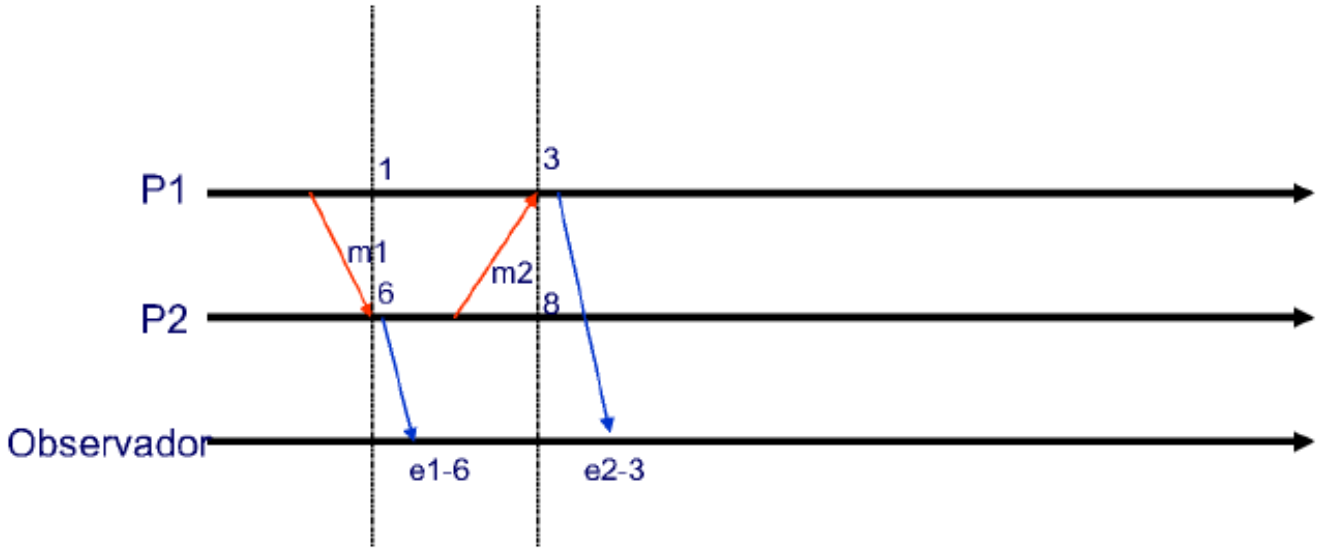
¿Marcas de tiempo en el observador?



¿Marcas de tiempo en el observador?



¿Marcas de tiempo en el observador?



Los relojes deben esatr sincronizados

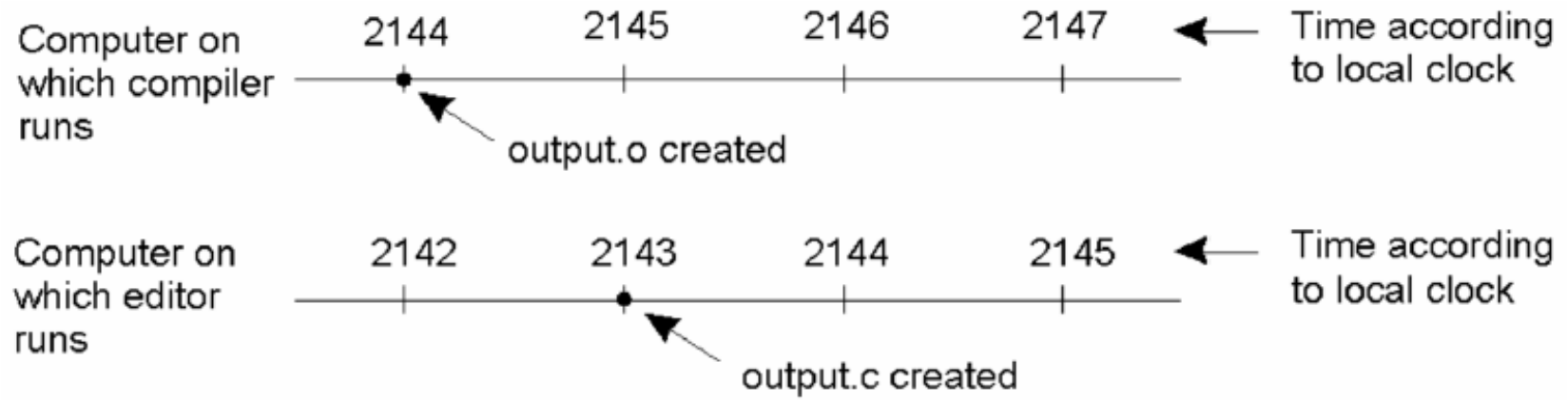
Tiempo y estado global

- 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 intante 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.

Tiempo y estado global

- 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

Tiempo y estado global

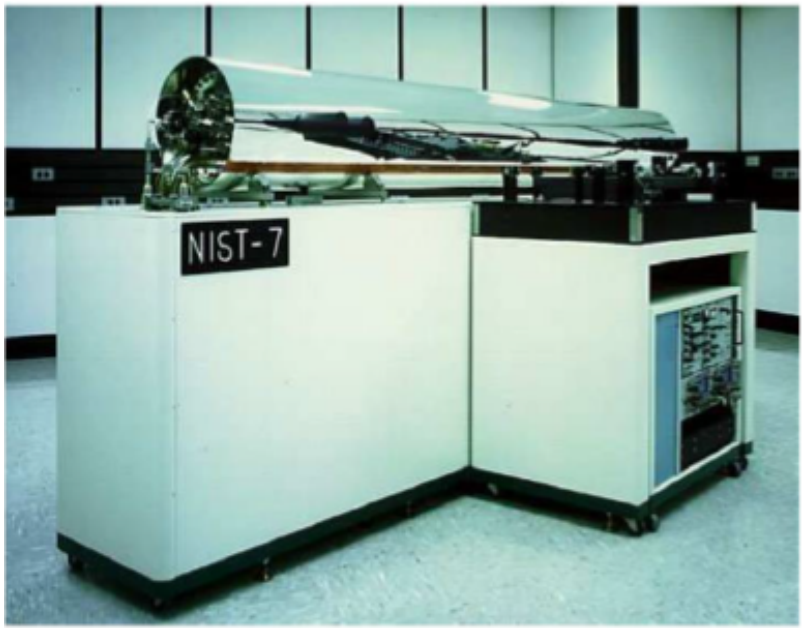


- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Tiempo físico

- Los relojes de los computadores son de cuarzo
  - La frecuencia de oscilación varía con la temperatura
  - Deriva: $\sim 10^{-6}$ (90ms en un día, 1s cada 11,6 días)
- Relojes atómicos: gran precisión, muy caros
  - deriva: $\sim 10^{-13}$ (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: $5x10^{-15}$

NIST-F1 (2005)
Deriva: $5x10^{-16}$

- 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 ($\sim$1s al año).
  - Tiempo universal coordinado (*UTC*): medido en segundos atómicos, sincronizado con tiempo astronómico (diferencia $>$900ms $\Rightarrow$ inserción de 1s)

- 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.

Relojes físicos

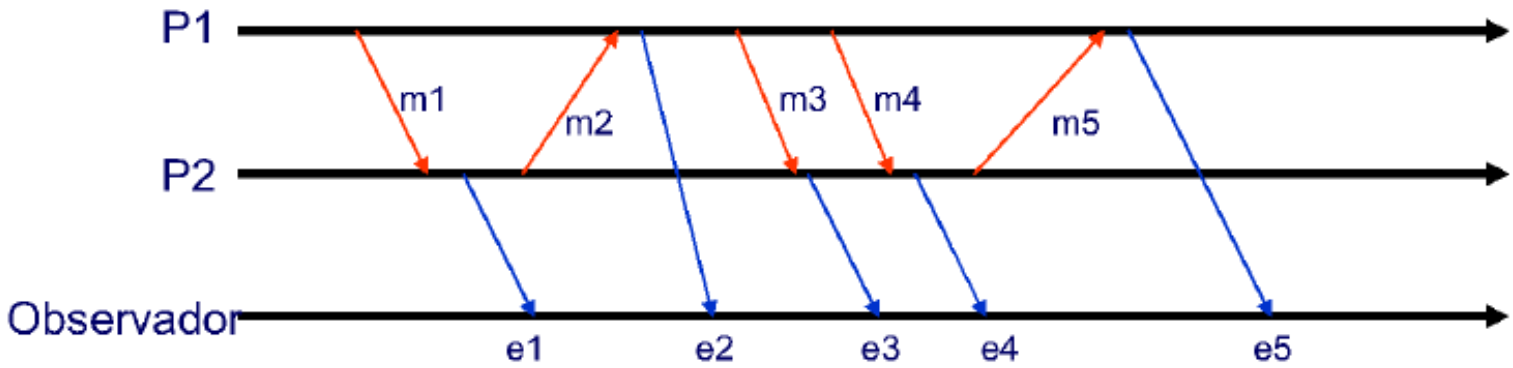
- Para ordenar dos eventos de un proceso basta con asignarles una marca de tiempo.
- Para un instante físico t.
  - $H_i(t)$: valor del reloj HW (oscilador)
  - $C_i(t)$: valor del reloj SW (generado por el SO)
    - $C_i(t) = aH_i(t) + b$
      Ej: # ms o ns transcurridos desde una fecha de referencia.
    - Resolución del reloj: periodo entre actualizaciones de $C_i(t)$
      Determina la ordenación de eventos.
- Dos relojes en dos computadores diferentes dan medidas distintas.
  - Un reloj actual puede tener una deriva de 1s al día.
  - Necesidad de sincronizar relojes físicos de un sistema distribuido.

Sincronización de relojes físicos

- D: Cota máxima de sincronización.
- S: fuente del tiempo UTC, t.
- Sincronización externa:
  - Los relojes están sincronizados si $\mid S(t) - C_i(t) \mid < D$
  - Los relojes se consideran sincronizados dentro de D.
- Sincronización interna entre los relojes de los computadores de un sistema distribuido
  - Los relojes están sincronizados si $\mid C_i(t) - C_j(t) \mid < D$
  - Dados dos eventos de dos computadores se puede establecer su orden en función de sus relojes si están sincronizados
- Sincronización externa $\Rightarrow$ sincronización interna

Relojes lógicos

- Dado que no se pueden sincronizar perfectamente los relojes físicos en un sistema distribuido, no se pueden utilizar relojes físicos para ordenar eventos.
- ¿Podemos ordenar los eventos de otra forma?



Causalidad potencial

- En ausencia de un reloj global la relación causa-efecto (*precede a*) es la única posibilidad de ordenar eventos.
- Relación de causalidad potencial (Lamport)
  - e_{ij} = evento j en el proceso i
  - Si j $<$ k entonces $e_{ij} \leftarrow e_{ik}$
  - Si e_i =send(m) y e_j =receive(m), entonces $e_i \leftarrow e_j$
  - La relación es transitiva
- Dos eventos son concurrentes (a $\|$ b) si no se puede deducir entre ellos una relación de causalidad potencial.

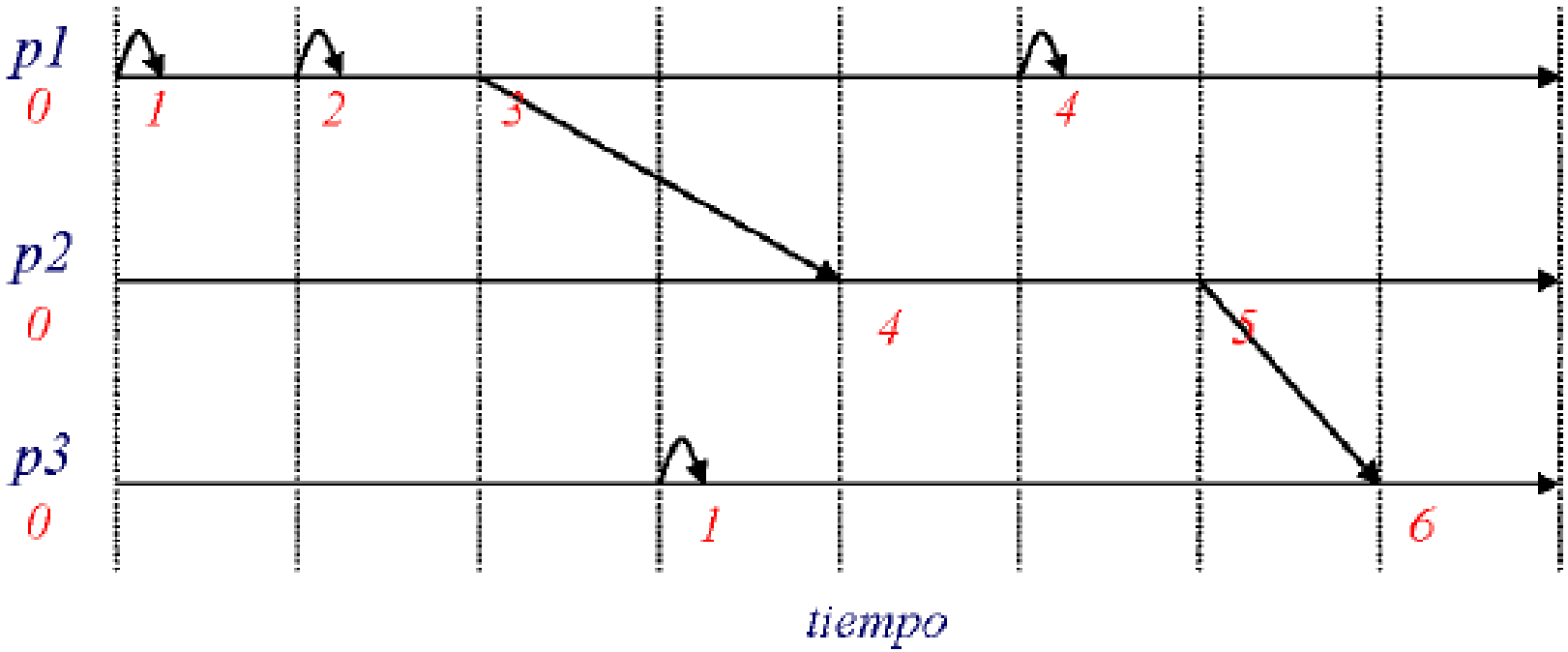
Aplicación de los relojes lógicos

- Sincronización de relojes lógicos
- Depuración distribuida
- Registro de estados globales
- Monitorización
- Entrega causal
- Actualización de réplicas

Relojes lógicos (algoritmo de Lamport)

- Algoritmo de Lamport (1978)
- Cada proceso P mantiene una variable entera RL_p(reloj lógico)
- Cuando un proceso P genera un evento, $RL_p = RL_p + 1$
- Cuando un proceso envía un mensaje m a otro le añade el valor de su reloj
- Cuando un proceso Q recibe un mensaje m con un valor de tiempo t, el proceso actualiza se reloj, $RL_q = max(RL_q t) + 1$
- El algoritmo asegura que si a $\leq_H$ b entonces RL(a) $<$ RL(b)

Ejemplo



Relojes lógicos totalmente ordenados

- Los relojes lógicos de Lamport imponen sólo una relación de orden parcial:
  - Eventos de distintos procesos pueden tener asociado una misma marca de tiempo.
- Se puede extender la relación de orden para conseguir una relaciñon de orden total añadiendo el nº de proceso
  - (T_a, P_a): marca de tiempo del evento a del proceso P

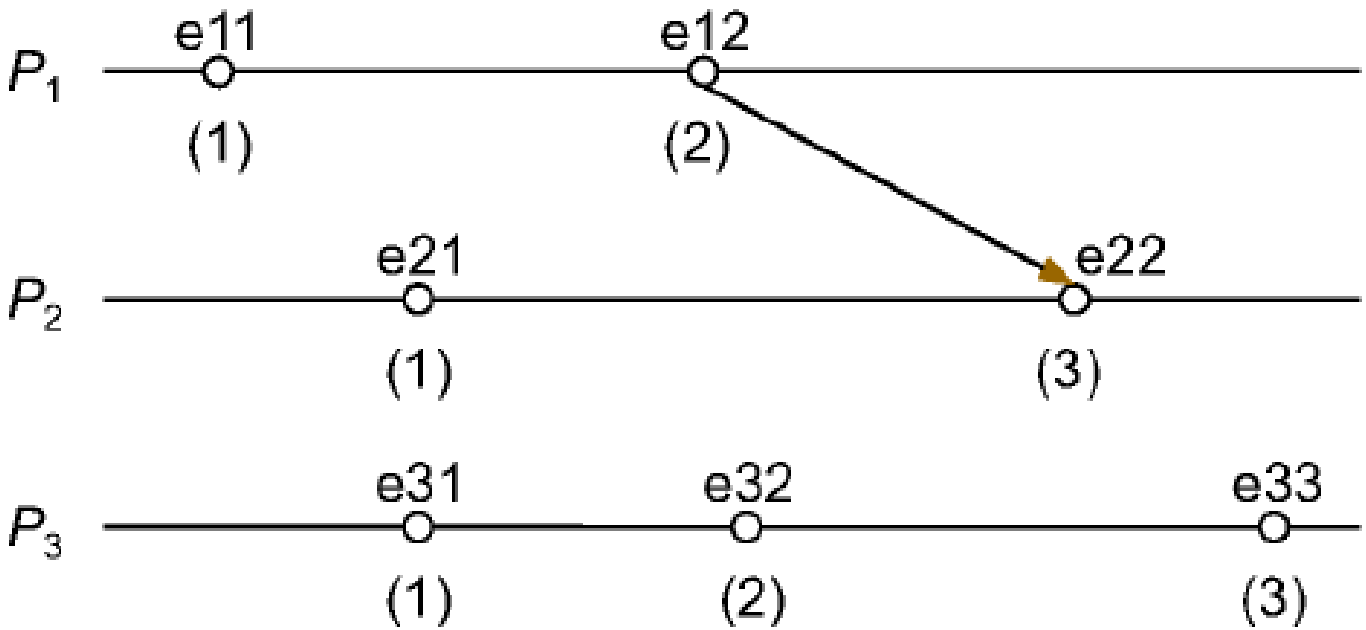
Algoritmo de Welch

- ¿Qué ocurre si el sistema ya dispone de un reloj?
  - No se puede cambiar el valor del reloj si es mantenido por un algoritmo diferente.
- Algoritmo de Welch
  - En lugar de avanzar el reloj en respuesta a los mensajes que llegan, se retrasa la entrega de esos mensajes hasta que se alcanza el valor de tiempo.
  - Los mensajes que llegan se almacenan en un buffer FIFO si su marca de tiempo es menor que la marca de tiempo del proceso receptor.

Problemas de los relojes lógicos

- No bastan para caracterizar la causalidad
  - Dados $RL(a)$ y $RL(b)$ no podemos saver:
    - si a precede a b
    - si b precede a a
    - si a y b son concurrentes
- Se necesita una relación $(F(e), <)$ tal que:
  - $a \leq_H b$ si y sólo si $F(a) < F(b)$
  - Los relojes vectoriales permiten representar de forma precisa la relación de causalidad potencial

Problemas de los relojes lógicos

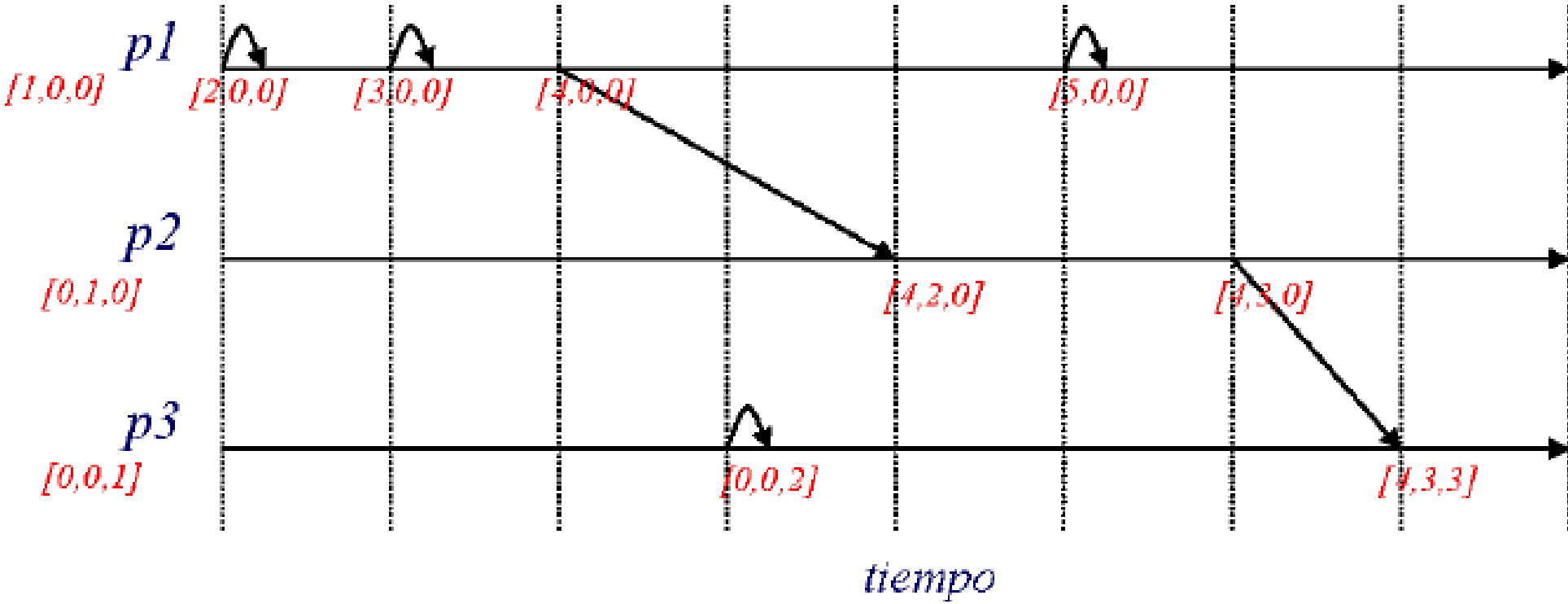


$C(e11) < C(e22)$, y $e11 \leftarrow e22$ es cierto
$C(e11) < C(e32)$, pero $e11 \leftarrow e32$ es falso

Relojes vectoriales

- Desarrollado independientemente por Fidge, Mattern y Schamuck.
- Todo proceso lleva asociado un vector de enteros RV
- $RV_i[a]$ es el valor del reloj vectorial del proceso i cuando ejecuta el evento a.
- Mantenimiento de los relojes vectoriales
  - Inicialmente $RV_i = 0 \forall i$
  - Cuando un proceso i genera un evento
    - $RV_i[i] = RV_i[i] + 1$
  - Todos los mensajes llevan el RV del envío
  - Cuando un proceso j recibe un mensaje con RV
    - $RV_j = max(RV_j, RV)$ (componente a componente)
    - $RV_j[j] = RV_j[j] + 1$ (evento de recepción)

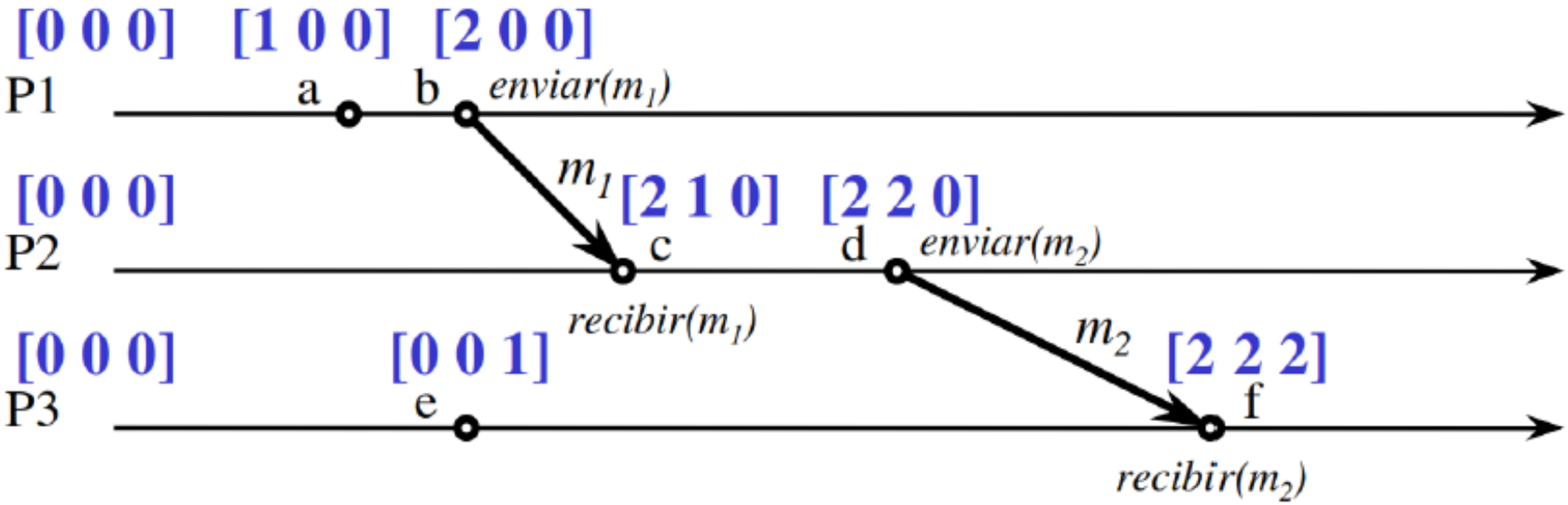
Ejemplo



Propiedades de los relojes vectoriales

- RV $<$ RV´ si y solo si
  - RV $\neq$ RV´ y
  - RV[i] $\leq$ RV´[i], $\forall$ i
- Dados dos eventos a y b
  - a $\leq_H$ b si y solo si RV(a) $<$ RV(b)
  - a y b son concurrentes cuando
    - Ni RV(a) $\leq$ RV(b)[i] ni RV(b) $\leq$ RV(b)

a $\|$ b (concurrent) because neither V(a) $\leq$ V(b) nor V(b) $\leq$ V(a)

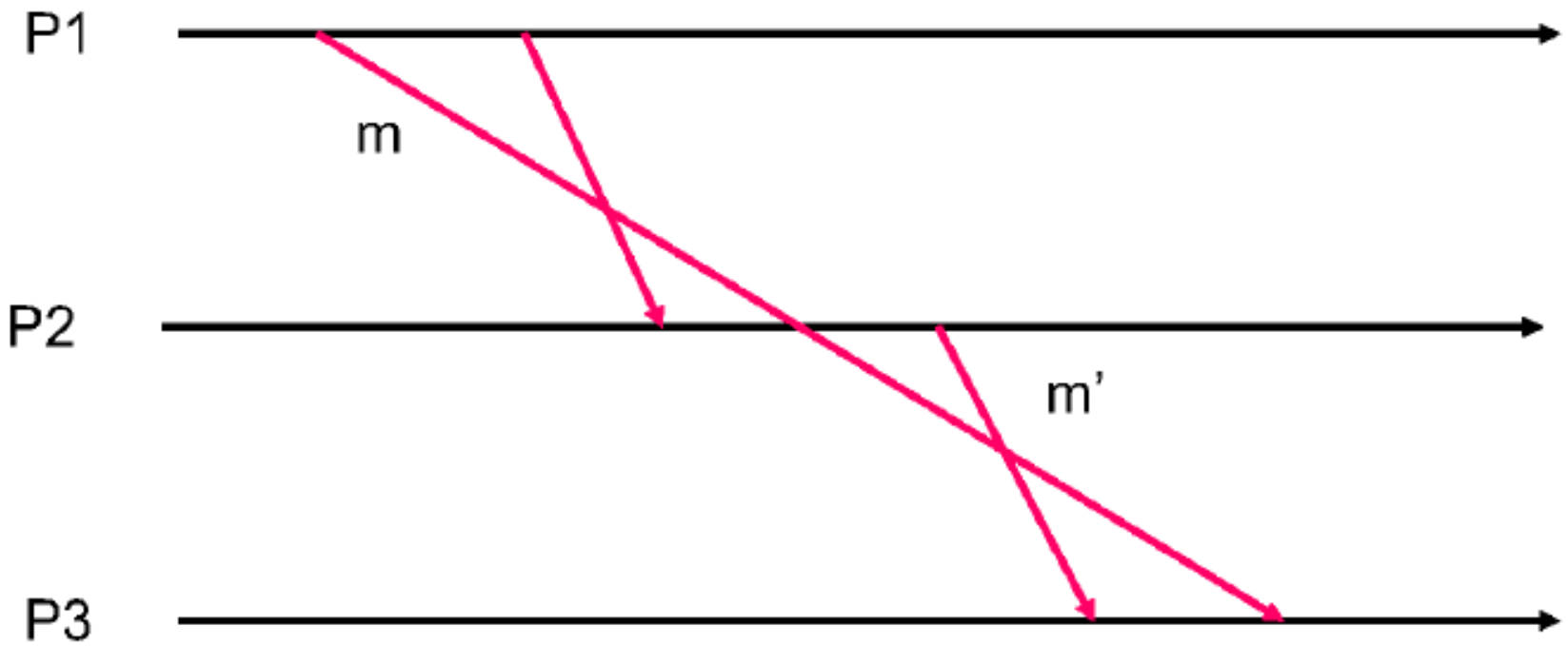


$$a \rightarrow b \quad b \rightarrow c \quad a \rightarrow c \quad a \rightarrow f \quad a \,||\, e \quad e \,||\, d$$

Entrega causal

- Se distinguen los eventos recibir y entregar
- Entrega FIFO
  - $send_i(m) \leq_H send_i(m')$ entonces $entrega_k(m) \leq_H entrega_k(m')$
- Entrega causal
  - $send_i(m) \leq_H send_j(m')$ entonces $entrega_k(m) \leq_H entrega_k(m')$
- La entrega causal requiere retrasar la entrega de un mensaje a un proceso hasta estar seguros que entre dos envíos no hay uno intermedio.
- Se puede implementar con relojes vectoriales

Ejemplo de entrega no causal



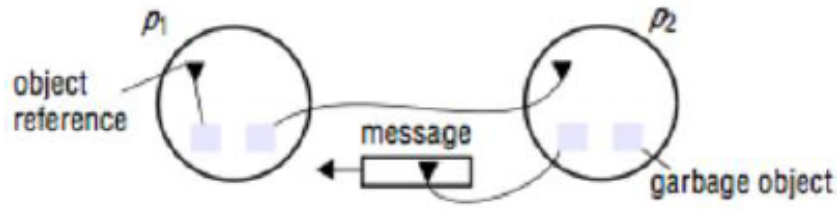
send(m) $\leq_H$ send(m') pero en P3 se recibe antes m'

Estados globales consistentes

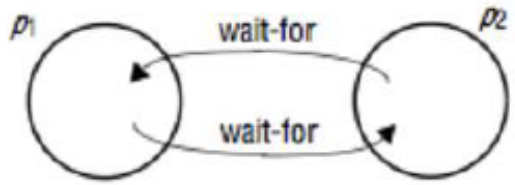
- En un sistema distribuido existen ciertas situaciones que no es posible determinar de forma exacta por falta de un estado global:
  - Recolección de basura: Cuando un objeto deja de ser referenciado por ningún elemento del sistema.
  - Detección de interbloqueos: Condiciones de espera ciclica en grafos 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.

- Hay tareas para las que necesitamos conocer el estado global del sistema:
  a) Detección de objetos distribuidos que ya no se utilizan
  b) Un interbloqueo distribuido ocurre cuando dos procesos esperan un mensahe del otro
  c) Detectar la terminación de un algoritmo distribuido
- Es vital tener en cuenta el estado de todos los procesos **y del canal de comunicación**

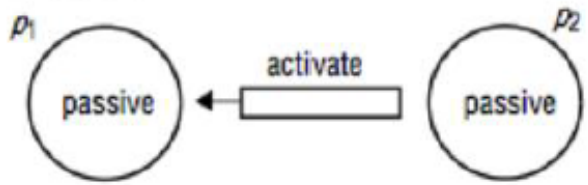
a) Recolección de basura



b) Interbloqueo



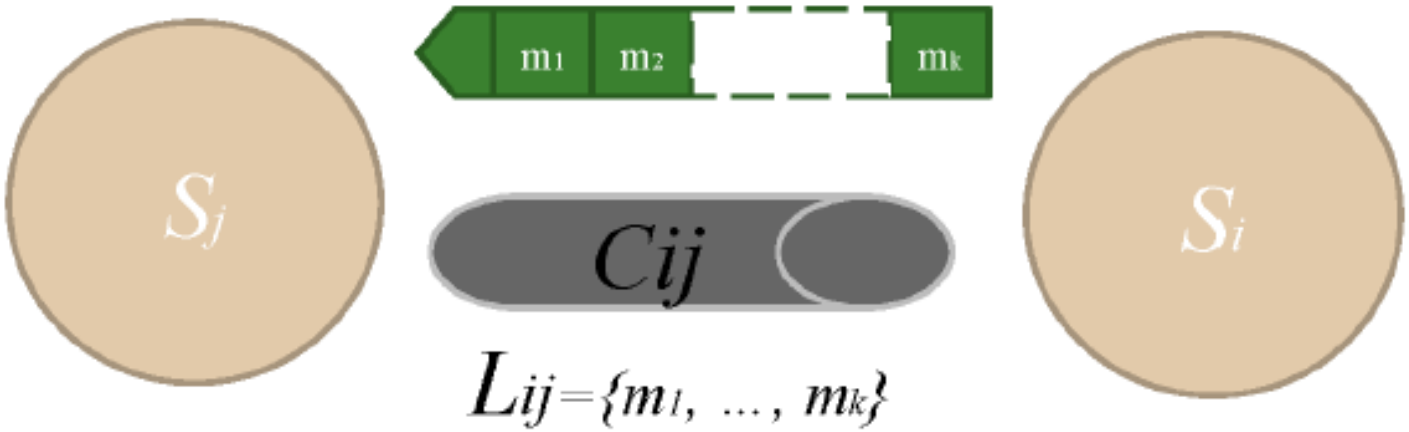
c) Terminación



Definiciones

El estado global de un sistema distribuido se denota por G=(S,L), donde:

- S=$\{S_1, S_2, S_3, S_4, ..., S_M\}$: Estado interno de cada uno de los M procesadores.
- $L = \{L_{i,j} \mid i, j \in 1...M\}$: $L_{i,j}$ Estado de los canales unidireccionales $C_{i,j}$ entre los procesadores. El estado del canal son los mensajes en él encolados



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.

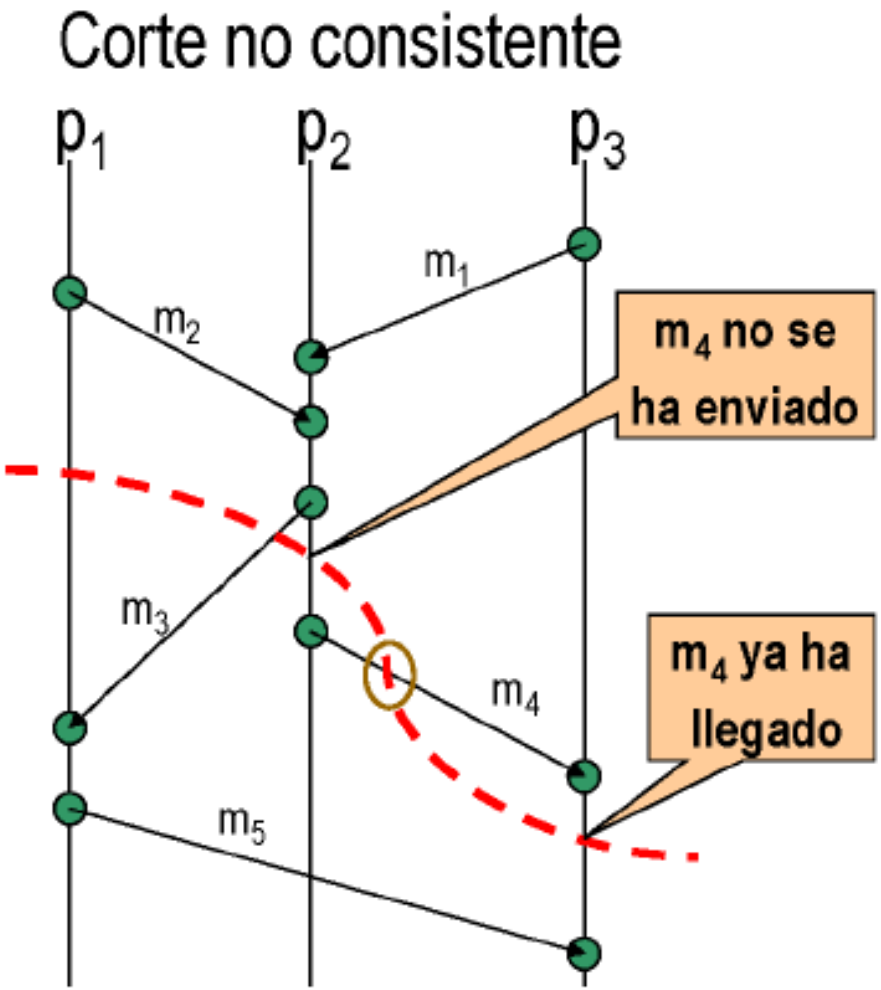
Debido a la imposibilidad de determinar el estado global de todo el sistema en un mismo instante se define una propiedad de consistencia en la evaluación de dicho estado.

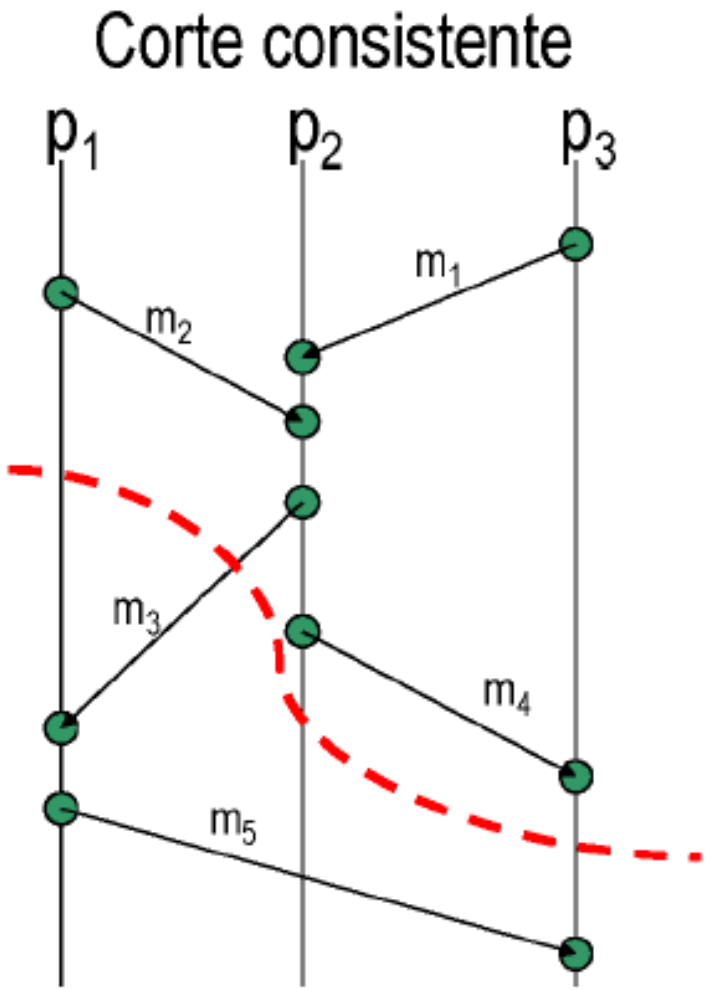
$$\text{Snapshots} = \text{cortes}$$

Cortes consistentes

- Un corte es un conjunto de evento (contiene al menos un evento por proceso)
- Un corte es consistente si para cada evento que contiene, incluye también todos los eventos que le precedin causalmente.
- Si **a** y **b** son dos eventos en un sistema distribuido y **C** un corte consistente, entonces:
  - $(a \in C) \wedge (b \rightarrow a) \Rightarrow b \in C$
- Si para un mensaje **m**, el evento **receive(m)** $\in$ **C**, entonces el evento **send(m)** $\in C$.

Cortes consistentes





Estado global y consistencia

- Para saber

  si un corte es consitente, 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.

Algoritmo de Chandy y Lamport

- Algoritmo para determinar un estado global consistente.
- Supone canales FIFO
- Un proceso denomindao iniciador comienza el agoritmo de *snapshot* distribuido.
- Cualquier proceso puede iniciar el algoritmo.
- El proceso iniciador envía un mensaje especial denominado "marcador".
- El estado global consta de los estados de los procesos y de los canales.
  - Los canales son pasivos: la responsabilidad de registrar el estado de un canal depende del proceso emisor.

Suposiciones

- Entre cada par de procesos el canal es FIFO.
- No hay fallos en los procesos ni en los canales durante la ejecución del algoritmo.
- Cualquier proceso puede iniciar el algoritmo en cualquier instante.
- Los procesos pueden seguir enviando y recibiendo los mensajes de aplicación durante la ejecución del algoritmo.
- Idea: es que cada proceso registre su estado y para cada canal, todos los mensajes que entraron después de que él registrara su estado y antes de que el emisor le envíe un mensaje marcador (lo que supone que ya ha registrado su estado).

Algoritmo

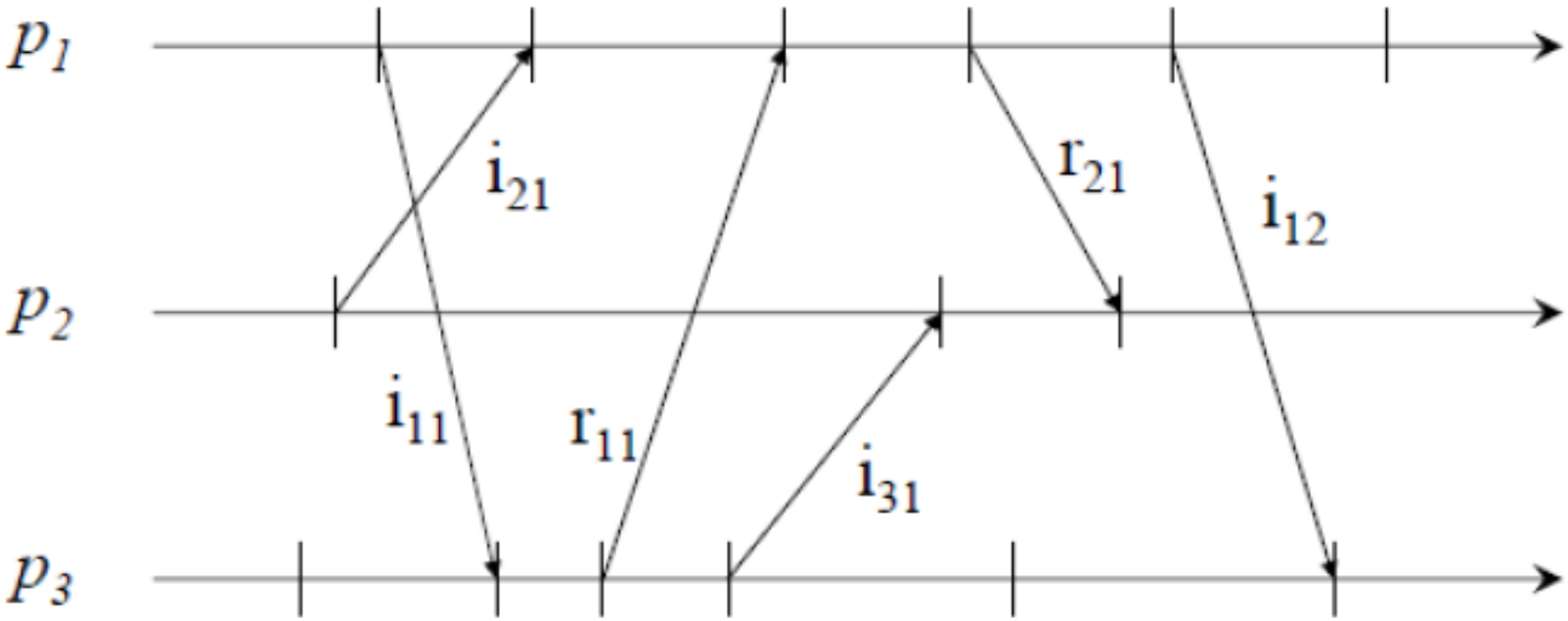
- Entre P_i y P_j el canal se denomina C_{ij}
- Proceso iniciador (P_1) de forma atómica:
  - Registra su estado local s_1
  - P_1 envía el marcador a todos sus vecinos a través de los canales salientes.
  - A partir de este momento almacenará en los canales C_{k1} los mensajes de aplicación que vaya recibiendo.
- Cuando un proceso P_j recibe un marcador de P_j (iniciador o no):
  - Si aun no ha grabado su estado:
    - P_i graba su estado S_i
    - Graba el estado del canal C_{ji} como vacío. Para el resto de canales se almacenarán los mensajes de aplicación a partit de ese momento.
    - Reenvía un mensaje marcador al resto de procesos (entre la recpción del marcador y su reenvío no ejecuta ningún otro evento).
  - Si ya ha grabado su estado:
    - Se registra el estado del canal C_{ji} como completo
- El algoritmo termina para un proceso una vez que ha recibido el marcador de todos los procesos.

Algoritmo

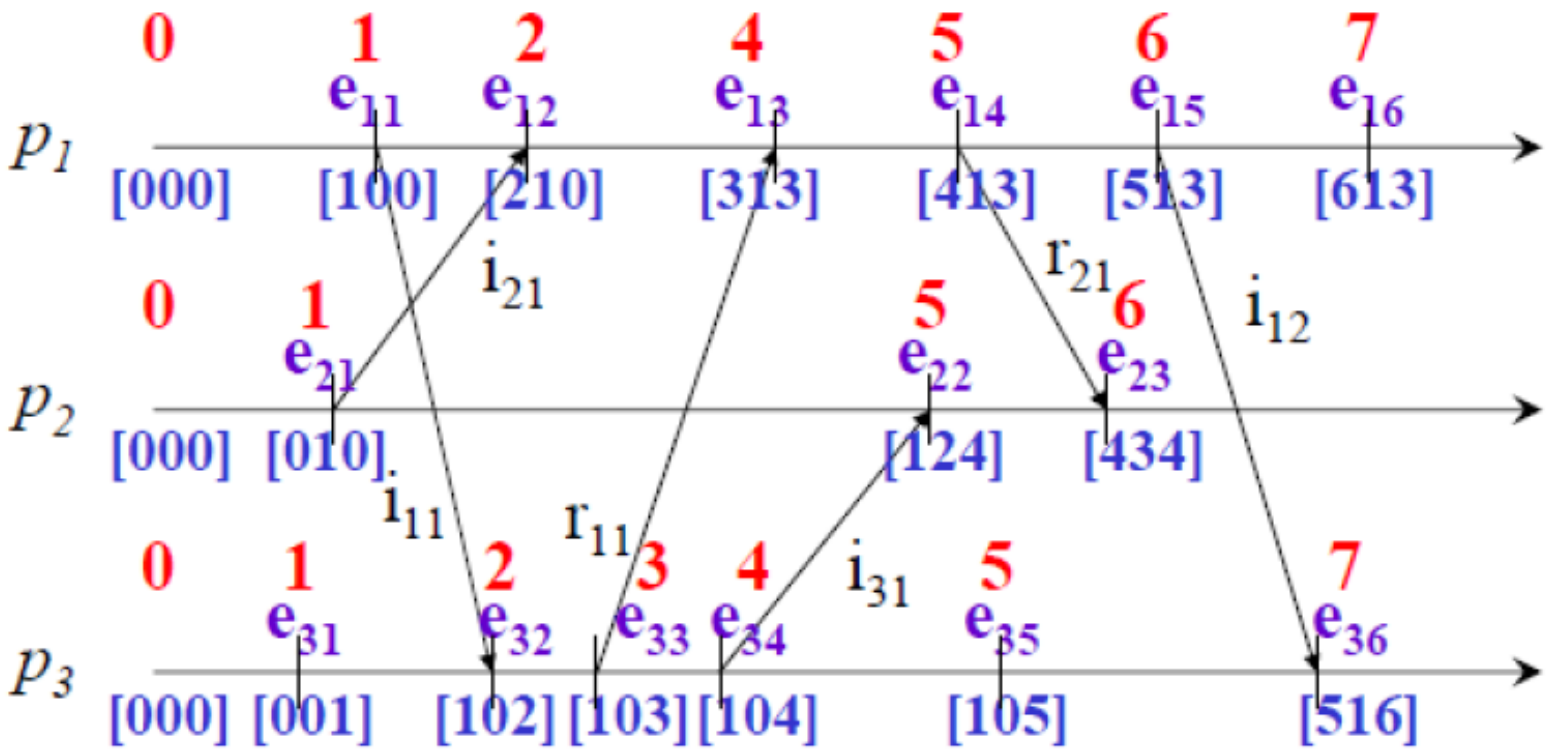
- Una vez registrados todos los estados (el algoritmo ha terminado)
  - El estado global consistente puede ensamblarse en cada proceso haciendo que cada proceso envíe los datos del estado que él ha registrado, por ejempli, a otro proceso.

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 process).
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, an 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 process in P, the computation of the snapshot is terminated.

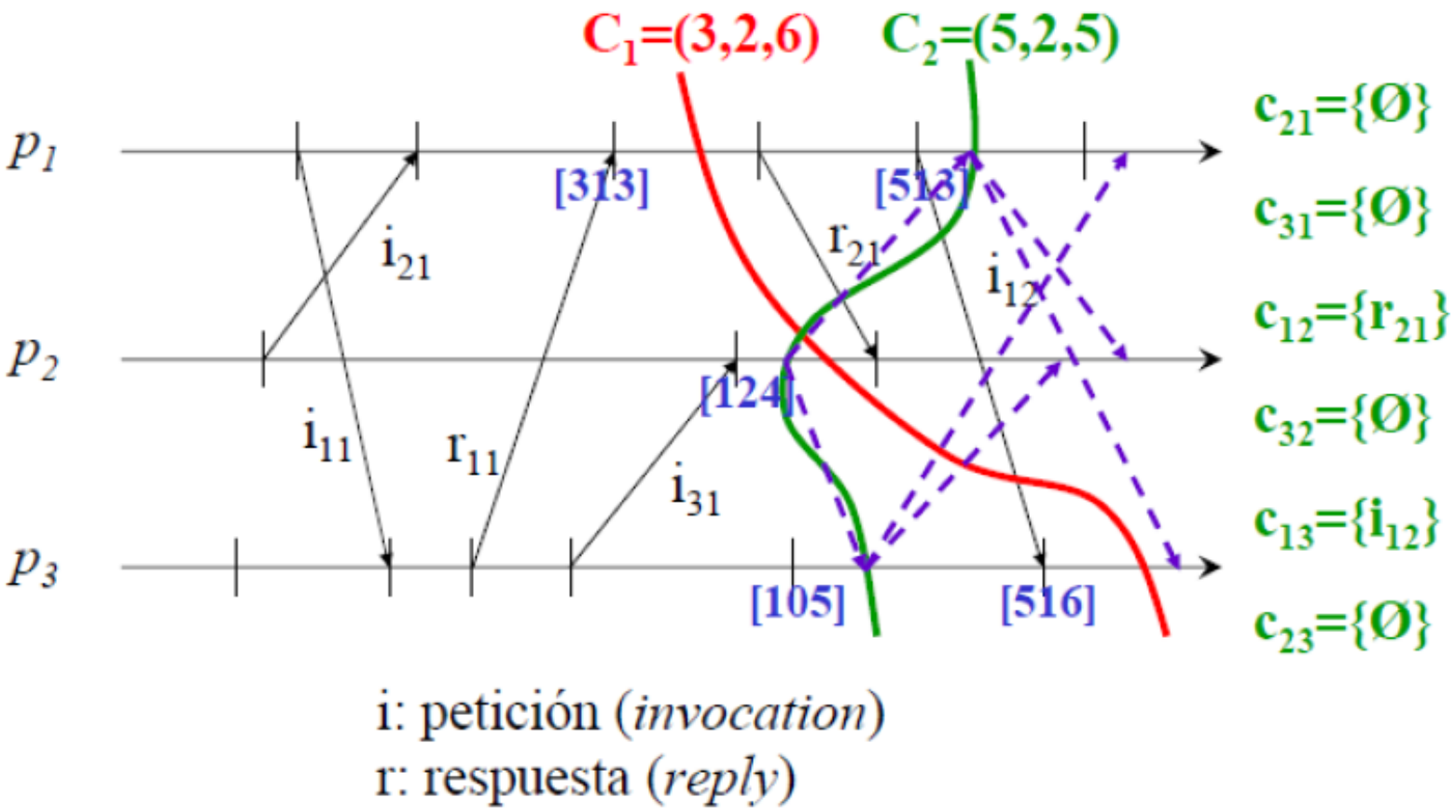


i: petición (*invocation*)
r: respuesta (*reply*)



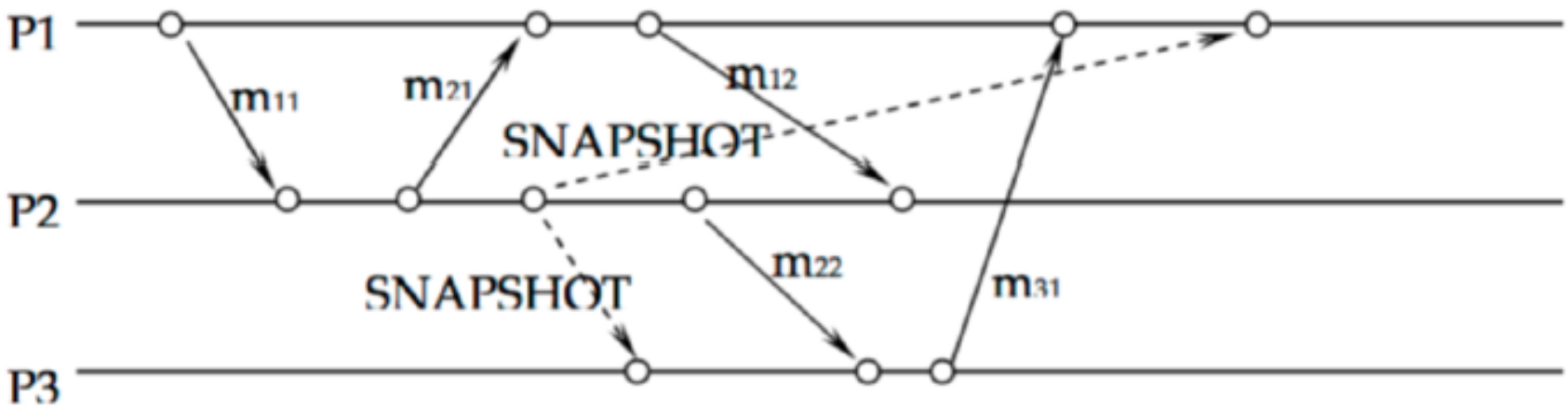
i: petición (*invocation*)

r: respuesta (*reply*)



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

Ejercicio
¿qué estado se registra?



Bibliografía

- Rauber. T, Rünger Gudula. "Parallel Programming: for Multicore and Cluster Systems". 2nd Edition. Springer, ISBN: 978-3-642-37800-3 (Print) 978-3-642-37801-0 (Online) 2013
- Comer, Douglas. "Internetworking With TCP/IP Volume III". 1 edition. Prentice Hall, ISBN: 0-13-032071-4. 2000
- C Lin, L Snyder. Principles of Parallel Programming. USA: Addison-Wesley Publishing Company, 2008. ISBN-13: 978-0321487902
- https://www.arcos.inf.uc3m.es/infodsd/
- http://ocw.uc3m.es/ingenieria-informatica/sistemas-distribuidos-2013/material-de-clase
- http://laurel.datsi.fi.upm.es/docencia/asignaturas/sd#proyectos_practicos
- https://cs.uwaterloo.ca/~rtholmes/teaching/2011winter/cs436/index.html#contact