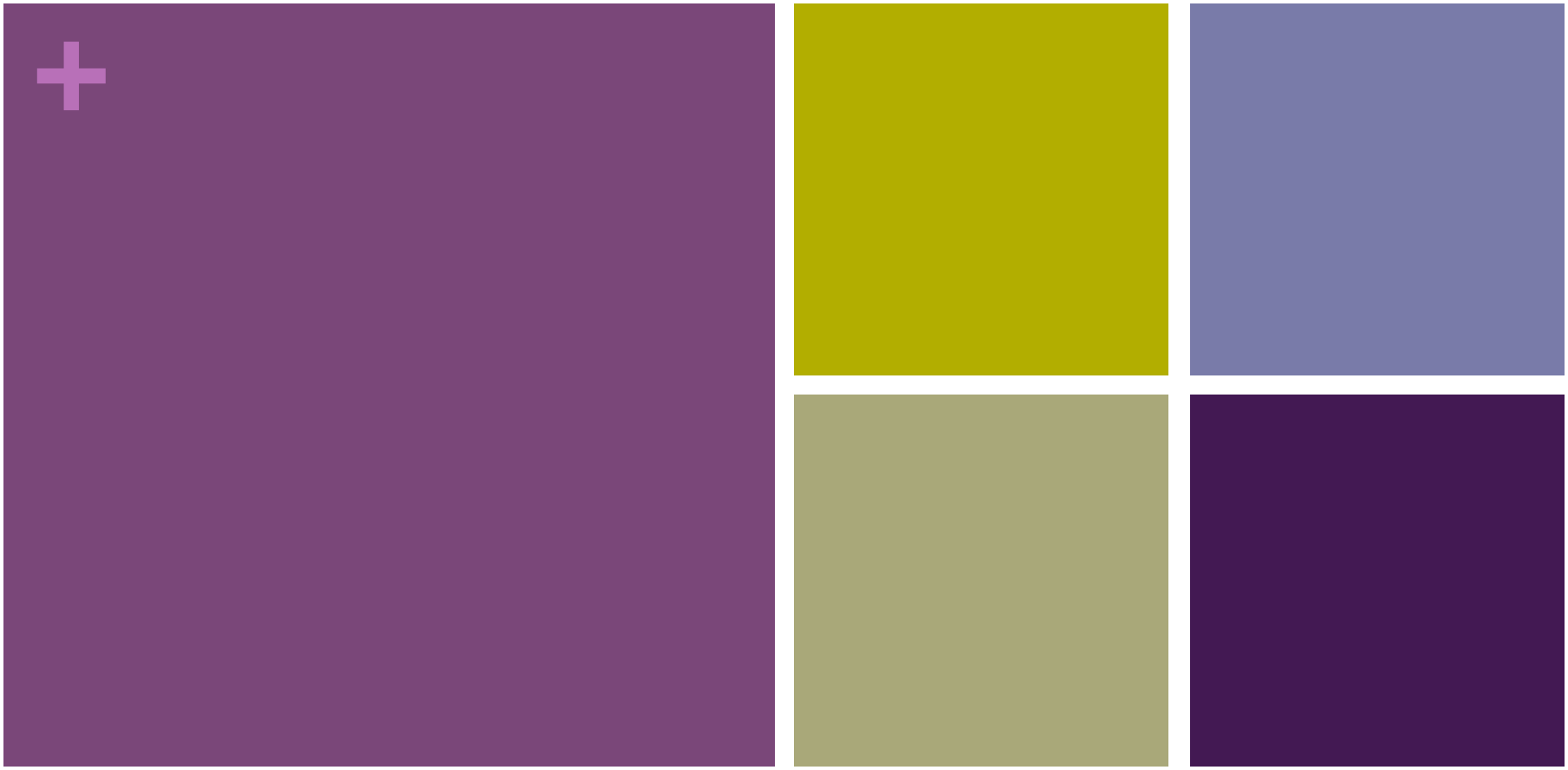


Sistemas Distribuidos
Tiempos y Estados Globales

Rodrigo Santamaría

Tiempos y estados globales

- Introducción
- Sincronización
- Tiempo y relojes lógicos
- Estados globales
- Depuración distribuida

Introducción

- El tiempo es importante en un SD por dos razones:
  - Es una cantidad que puede medirse de manera precisa
  - Existen muchos algoritmos basados en sincronización de relojes para solucionar problemas distribuidos
- Pero también presenta problemas
  - No existe un tiempo absoluto de referencia
  - Los relojes de distintos computadores no están sincronizados
    - Deriva de reloj

Introducción

Reloj hardware

- Cada computador tiene un reloj hardware, que consta de:
  - Un cristal de cuarzo que oscila a una frecuencia bien definida
    - Según el tipo y forma del cristal, y la tensión que soporta
  - Un registro contador
  - Un registro constante
- El registro contador se decrementa con cada oscilación
  - Cuando llega a cero, genera una interrupción (*tick* de reloj), y
  - El registro contador se reinicia al valor del registro constante

Introducción

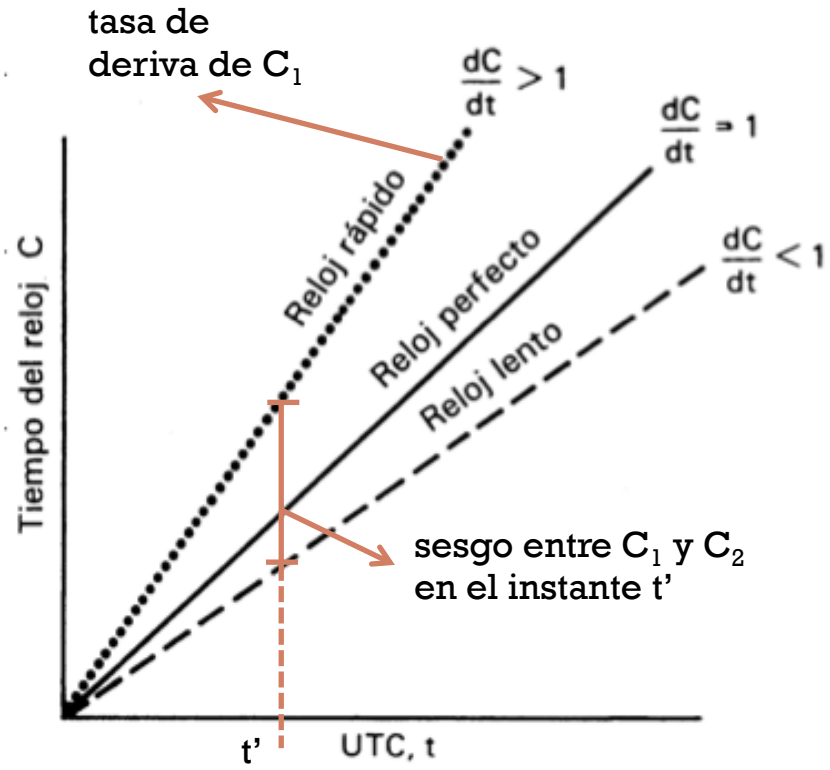
Reloj software

- Imita a un reloj normal a partir del reloj hardware
- Necesita:
  - Una fecha y hora de origen
    - P. ej. en UNIX: 00:00 del 1 de enero de 1970
  - Número de ticks desde esa fecha
    - Al arrancar pide la fecha y hora actual
    - Calcula el nº de ticks desde la fecha de origen
    - Con cada tick, incrementa el nº de ticks y recalcula la fecha actual a partir de la fecha de origen
- Básicamente, realiza un escalado (ticks → segundos) y un desplazamiento (desde la fecha de origen)

Introducción

Sesgo y deriva de reloj

- Sesgo (skew)
  - Diferencia entre las lecturas de dos relojes en un determinado instante
- Deriva (drift)
  - Tasa con la que el reloj se desvía respecto a la hora real
    - Reloj normal: 10^{-6} s/s ~ 1s cada 12 días
    - Reloj de precisión: 10^{-7} o 10^{-8} s/s ~ 1s cada 4 meses o 3 años



Introducción

Reloj atómico y TAI

- Reloj atómico: los relojes físicos más precisos
  - Tasas de deriva de 10^{-12} (hasta 10^{-16} en 2011)
  - Se basan en la oscilación entre estados de un átomo de Cs^{133}
- Tiempo atómico internacional (TAI)
  - Tiempo medio de oscilación del Cs^{133} de 200 mediciones en 70 laboratorios distintos
  - Desde su comienzo en 1972, se ha desviado 34s

Introducción

Reloj astronómico y UT1

- Reloj astronómico:
  - Medición a partir de la posición relativa del Sol y la Tierra
    - Es decir, posición del Sol respecto a un punto en la Tierra
  - Del S XV al XIX, sólo había tiempos locales
  - A finales del XIX, se establecen zonas horarias, y un tiempo universal estándar: GMT (Greenwich Meridian Time)
- Tiempo Universal (UT1)
  - Basado en la medición precisa de la rotación de la Tierra respecto al Sol, satélites GPS y otros astros
  - OJO: la rotación de la Tierra *no* es constante: está disminuyendo

Introducción

UT1

- Tiempo Universal Coordinado (UTC)
  - Se emite regularmente desde estaciones de radios y satélites GPS
  - Determinado a partir de TAI, pero sincronizado con UT1
    - UTC = TAI + 34s (actualmente)
    - UT1-1s < UTC< UT1+1s
- Segundos de salto: segundos añadidos a TAI para sincronizarse con UT1
  - Necesario debidos a eventos geológicos y climáticos que afectan al movimiento de la Tierra (sobre todo el efecto de marea)
    - Aceleran UT1 respecto a TAI/UTC
    - Se han añadido 34s desde 1972 hasta ahora

http://www.eldia.com.do/globales/2012/7/2/86807/Segundo-extra-causa-estragos-en-Internet

Introducción

Sincronización de relojes físicos

- Sean C_i relojes de procesos
  - $C_i(t)$ tiempo del reloj C_i en el momento t
- Sincronización externa de límite D
  - $|S(t) - C_i(t)| < D$
    - S – fuente de tiempo UTC (u otro tiempo de referencia)
- Sincronización interna de límite D
  - $|C_i(t) - C_j(t)| < D$
  - La diferencia entre dos relojes es siempre menor que D
- Un sistema sincronizado externamente con límite D, lo estaría internamente con límite 2D

Introducción

Corrección de relojes

- Un reloj hardware H es correcto si su tasa de deriva cae dentro de un límite conocido $p > 0$
  - Sean los tiempos reales t y t' (t' > t) y sus relojes hw H(t) y H(t')
  - El error entre los tiempos hardware para un límite *p* conocido:
    - $(1-p)(t'-t) \leq H(t')-H(t) \leq (1+p)(t'-t)$
  - Dos relojes correctos cumplen con la propiedad de *monotonía*:
    - Un reloj C siempre avanza
    - Si t' > t → C(t') > C(t)
  - Un reloj no necesita ser preciso para ser correcto

Introducción

Fallos de reloj

- Fallo de ruptura (crash)
  - El reloj deja de contar ticks
- Fallo arbitrario
  - Cualquier otro fallo relacionado con el reloj
  - Ejemplo: efecto 2000
    - Transición de 31 de dic de 1999 a 1 de enero de 1990
  - Ejemplo: reloj con batería baja
    - Su tasa de deriva se dispara inesperadamente

+ Tiempos y estados globales

- Introducción
- Sincronización
- Tiempo y relojes lógicos
- Estados globales
- Depuración distribuida

Sincronización

Sincronización interna en un sistema síncrono

- Sincronización interna → no nos importa la hora correcta
- Sean dos procesos:
  - Uno de ellos manda su tiempo *t* al otro en un mensaje *m*
  - El mensaje *m* tarda un tiempo T_{trans} en viajar por la red
    - T_{trans} es desconocido e imposible de determinar
      - Depende del tráfico de red y del resto de procesos
    - Pero se pueden estimar sus límites
      - *min*: cuando no hay otros procesos ni tráfico en la red
      - *max*: se puede hacer una estimación conservadora
    - Luego la incertidumbre *u* en el tiempo de transmisión es
      - $u = (max - min)$

Sincronización

Sincronización interna en un sistema síncrono

- El receptor sincroniza su reloj según el *t* del mensaje
  - Si lo pone a $t + min$, el error de la sincronización es u
    - Pues la transmisión puede haber tardado *max*
  - Si lo pone a $t + max$, el error de la sincronización es u
    - Pues la transmisión puede haber tardado min
  - Si lo pone a $t + (max + min)/2$, el error es $u/2$

  - En general, si tenemos N relojes, el error óptimo es $u(1\text{-}1/N)$
  - En un sistema asíncrono, no podemos determinar el máximo
    - $T_{trans} = min + x$, con $x \geq 0$

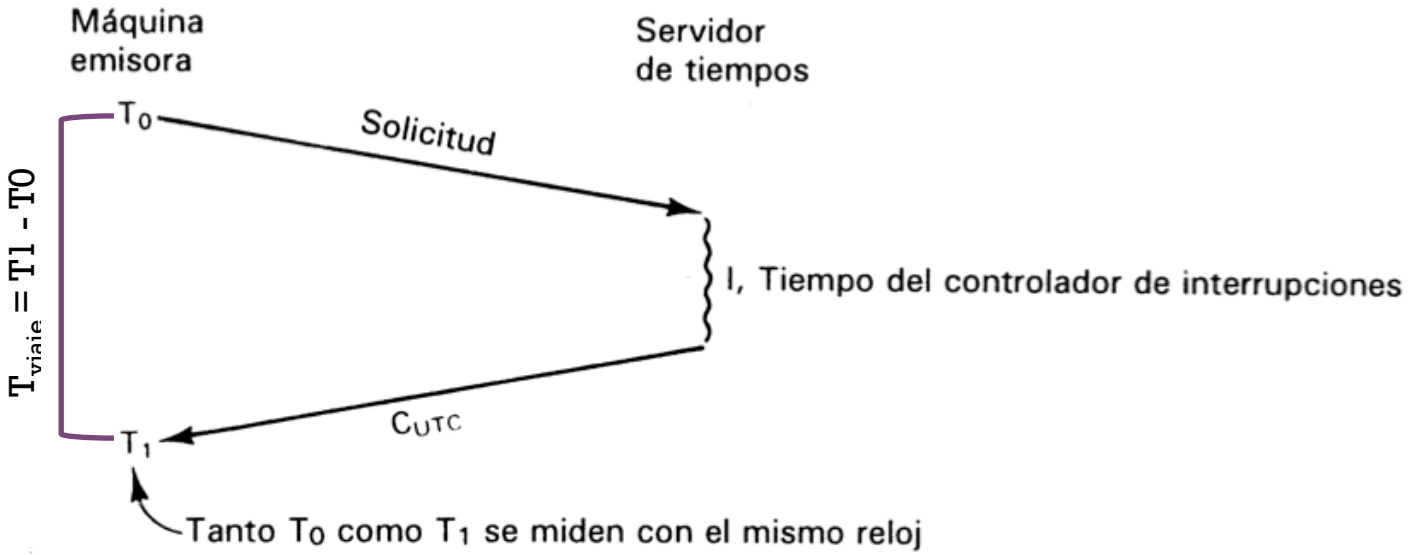
Sincronización

Método de Cristian (1989)

- Algoritmo centralizado
  - Servidor de tiempo
  - Clientes que se sincronizan con el servidor
- Cada nodo, periódicamente, hace una petición al servidor
  - Mensaje "*dame tiempo*"
- El servidor responde
  - Mensaje "*tiempo*=C_{UTC}"
  - Se considera que el servidor responde rápidamente
    - El algoritmo sólo sincroniza si el tiempo de respuesta es razonablemente más corto que la precisión requerida

Sincronización

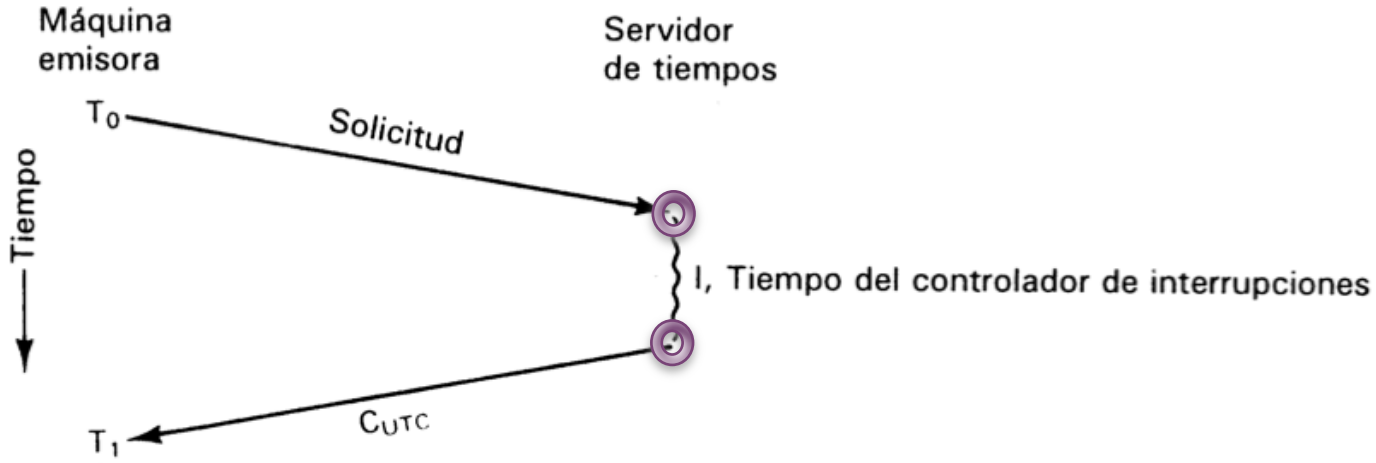
Método de Cristian



- El cliente sincroniza su reloj a $C_{UTC} + T_{viaje}/2$
  - Estima que el servidor puso C_{UTC} en el mensaje justo a la mitad del tiempo transcurrido desde que hizo la petición
  - Es una estimación razonable, si los dos mensajes se enviaron por la misma red

Sincronización

Método de Cristian



- Considerando *min* el tiempo mínimo de transmisión:
  - El tiempo mínimo en el que el servidor escribió C_{UTC} es *min*
  - Y el tiempo máximo es T_{viaje} – *min*
  - Por tanto, el cliente se sincroniza a un valor en el rango
    - $[C_{UTC}+min, C_{UTC}+T_{viaje}-min]$, con anchura $T_{viaje}- 2 \cdot min$
      - Es decir, la precisión de la sincronización es $\pm T_{viaje}/2 - min$

Sincronización

Método de Cristian

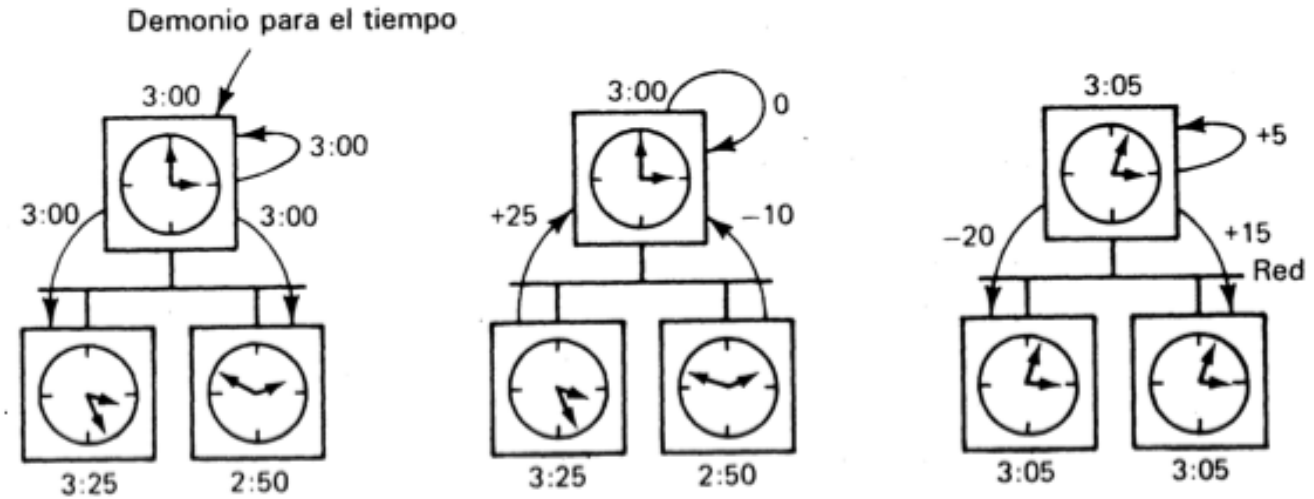
Límites	Desviación límite	Desviación de la estimación
$C_{UTC}+T_{viaje}-min$	$-T_{viaje}/2+min$	
Estimación		
$C_{UTC}+T_{viaje}/2$		$\pm T_{viaje}/2-min$
$C_{UTC}+min$	$T_{viaje}/2-min$	

- Problemas
  - Centralizado: ¿qué pasa si el servidor falla?
    - Solución: múltiples servidores
  - Servidores con fallos, deliberados o indeseados
    - Problema de los generales bizantinos

Sincronización

Algoritmo de Berkeley (1989)

- Algoritmo centralizado
  - Pero esta vez, el servidor (maestro) se elige entre todos los computadores conectados (esclavos)
    - No provee su tiempo, si no que lo estima a partir de todos los computadores conectados
      - Estima la deriva de cada reloj y manda la corrección



Sincronización

Algoritmo de Berkeley (1989)

- Problemas
  - Centralizado: fallo del nodo maestro
    - Se elige un nuevo maestro
  - Centralizado: broadcasting
    - Difícil de escalar

Sincronización

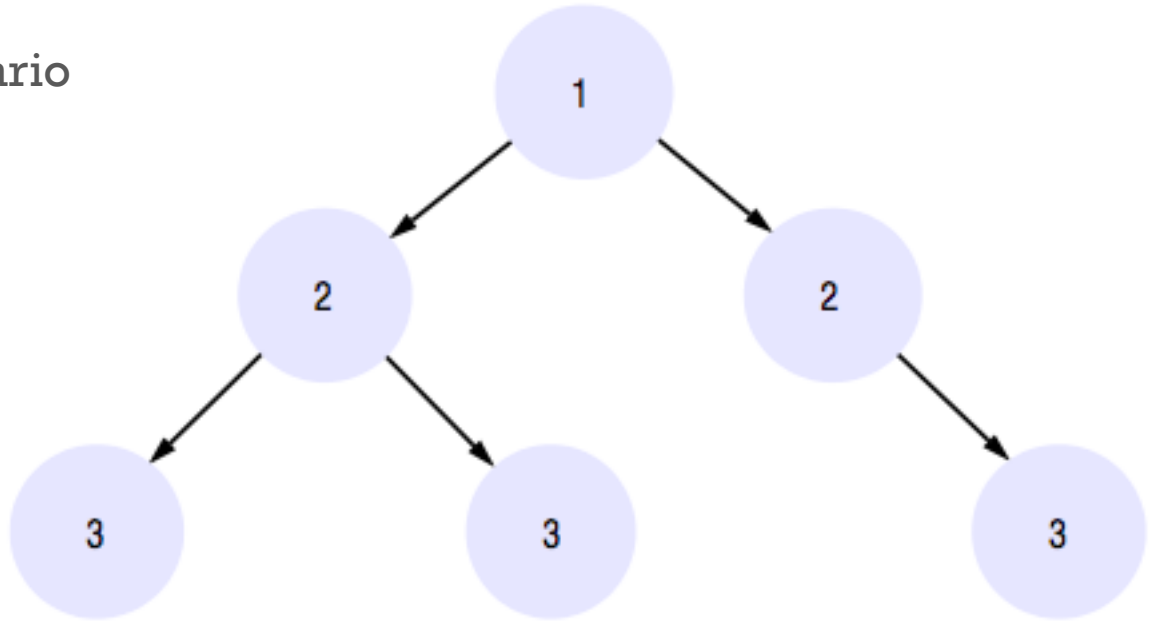
Network Time Protocol (NTP - 1995)

- Cristian y Berkeley se diseñaron para su uso en intranets
  - Al ser centralizados son difícilmente escalables
- Objetivos de NTP
  - Proporcionar un servicio que permita a los clientes a lo largo de Internet sincronizarse con UTC
  - Proporcionar un servicio fiable que pueda aguantar pérdidas de conectividad prolongadas
  - Permitir a los clientes sincronizarse de manera lo suficientemente frecuente como para compensar las tasas de deriva usuales
  - Proporcionar protección de las interferencias con el servicio de tiempos, sea maliciosa o accidental

Sincronización

NTP: jerarquía

- Algoritmo cliente-servidor jerarquizado (estratos)
  1. Servidores de estrato 1: radio-reloj, GPS
     - Reciben directamente la señal UTC
  2. Servidores de estrato 2, sincronizados con los de estrato 1
  3. etc.
  4. Estaciones de usuario



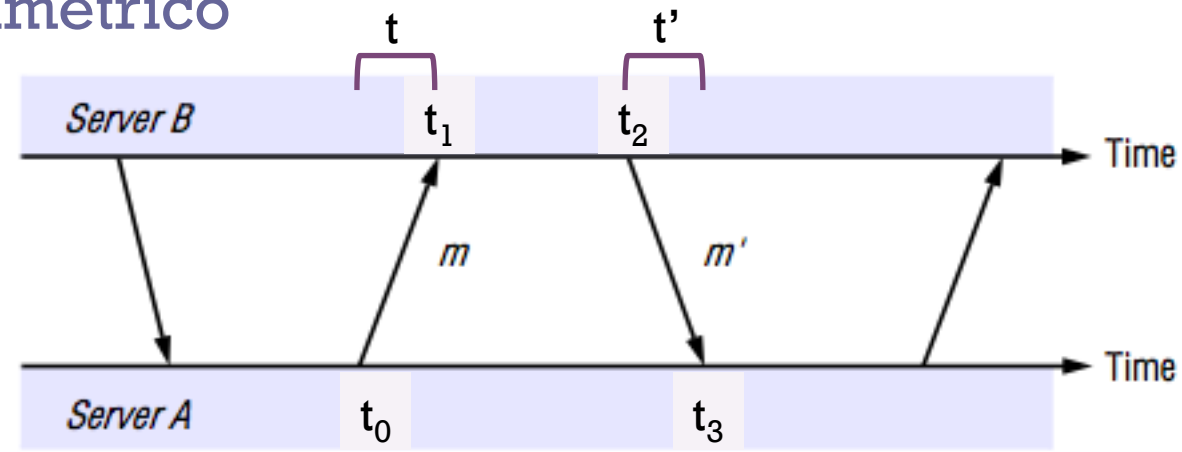
Sincronización

NTP: modos de sincronización

- Multidifusión
  - Reparto periódico del tiempo por los servidores
  - Baja precisión
- Llamada a procedimiento remoto (RPC)
  - Petición de tiempo de un servidor a otro de estrato inferior
  - Similar al algoritmo de Cristian
  - Mayor precisión
- Simétrico
  - Sincronización entre servidores en los estratos más bajos (1,2,3)
  - Máxima precisión
- En modos RPC y simétrico, se intercambian dos mensajes:
  - *m* que contiene el tiempo de envío t_0
  - *m'* que contiene el tiempo de recepción de *m* (t_1) y el de envío de *m'* (t_2)

Sincronización

NTP: simétrico

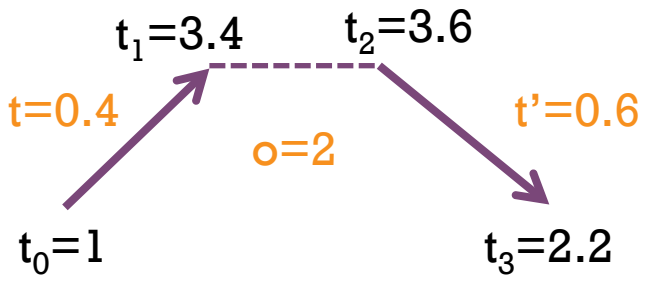


- Los mensajes llegan con un retardo de transmisión (t y t') y con desplazamiento de B respecto a A (offset o)
  - $t_1=t_0+t+o$ $t_3=t_2+t'-o$
- Sumando las dos ecuaciones obtenemos el retardo (delay) d_i:
  - $d_i=t+t'=(t_1-t_0)+(t_3-t_2)$
- Restando las dos ecuaciones obtenemos el desplazamiento (offset) o:
  - $o = (t_1-t_0 + t_2-t_3)/2 + (t'-t)/2 = o_i + (t'-t)/2$
  - Por tanto $o_i - d_i/2 \leq o \leq o_i + d_i/2$ (suponiendo $t, t' \geq 0$)
    - o_i es una estimación del desplazamiento y d_i una medida de la precisión de dicha estimación

Sincronización

NTP: modo simétrico

- Ejemplo



conocidos
desconocidos
estimados

$t_1 = t_0 + t + o \rightarrow t = t_1 - t_0 - o$

$t_3 = t_2 + t' - o \rightarrow t' = t_3 - t_2 + o$

$t + t' = d_i = t_1 - t_0 + t_3 - t_2 = 1.0$

$t - t' = t_1 - t_0 + t_2 - t_3 - 2o \rightarrow o = o_i + d_i/2$

$d_i = t + t' = (t_1 - t_0) + (t_3 - t_2)$
$o_i = (t_1 - t_0 + t_2 - t_3)/2$

$o_i - d_i/2 \leq o \leq o_i + d_i/2$

$1.9 - 0.5 \leq o \leq 1.9 + 0.5$

Sincronización

NTP: ampliaciones

- Algoritmo de filtrado de pares (peers)
  - Tomamos las 8 últimas estimaciones (los últimos 8 pares $<o_i,d_i>$)
  - Algoritmo de Marzullo: elegimos el par que tenga el valor más pequeño de d, y sea consistente con el mayor nº de intervalos
  - Filtrado: puede ser que un servidor nos dé 8 estimaciones muy dispares → servidor poco fiable.
    - NTP pregunta a varios servidores (peers) de la jerarquía y descarta aquellos con alta disparidad
- Modelo de bucle de bloqueo de fase [Mills, 1995]
  - Cada procesador modifica la frecuencia del reloj según su tasa de deriva
    - Si el reloj siempre se adelanta, se baja su frecuencia para que su tasa de deriva se reduzca

+ Tiempos y estados globales

- Introducción
- Sincronización
- Tiempo y relojes lógicos
- Estados globales
- Depuración distribuida

Tiempo y relojes lógicos

- Lamport (1978)
  - *"Como no podemos sincronizar perfectamente los relojes en 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"*
- Intuitivamente
  - Si dos sucesos han ocurrido en el mismo proceso, entonces ocurrieron en el orden en que los observa dicho proceso
  - El suceso de envío de un mensaje ocurre antes que el suceso de recepción de dicho mensaje
- Relación "sucede antes que"
  - Ordenación parcial resultante de la aplicación de estos dos axiomas
  - La denotamos por "→"

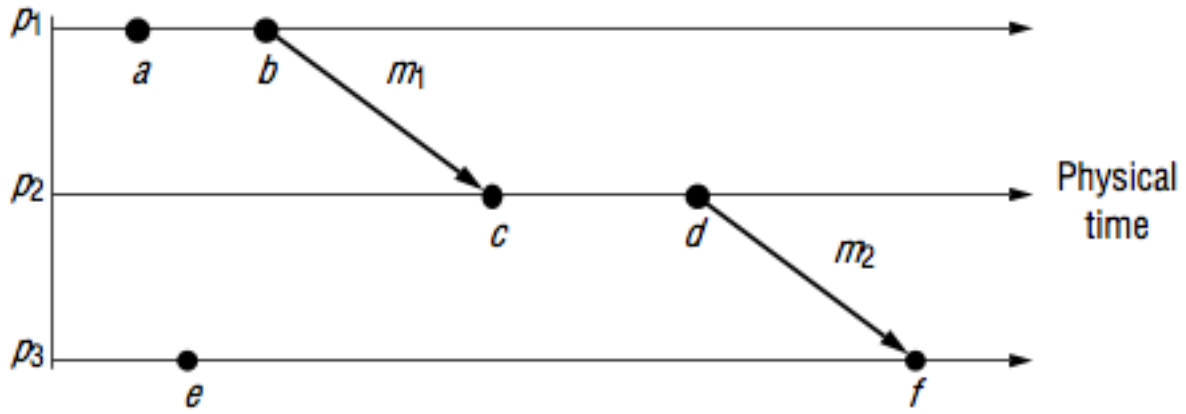
Tiempo y relojes lógicos

Tiempos de Lamport

- La relación "sucede antes que" satisface las condiciones:
  - Si a y b son sucesos del mismo proceso y a ocurre antes que b
    - a→b
  - Si a es un suceso de envío y b es su suceso de recepción
    - a→b
  - Si a→b y b→c, entonces a→c (propiedad transitiva)
  - a→a es siempre falso
  - Si no se cumple a→b ni b→a, entonces a y b son **concurrentes**, no se puede afirmar nada sobre su ordenación o causalidad.

Tiempo y relojes lógicos

Tiempos de Lamport



- Relaciones verdaderas:
  - a→b
  - b→c
  - c→d
  - d→f
  - e→f
  - +transitivas
- Sucesos concurrentes
  - e con todos los demás salvo f

Tiempo y relojes lógicos

Tiempo lógico de Lamport [1978]

- Objetivo:
  - Asociar una marca de tiempo (*timestamp*) a todos los sucesos que se pueden ordenar mediante relaciones "sucede antes que"
- Sea un proceso P_i con un reloj lógico asociado C_i
  - Se asigna un numero $C_i(e)$ a cada evento *e* de P_i
  - Independiente del reloj físico!
- Condiciones dados dos eventos a y b
  - Si ambos son del mismo proceso P_i y $a \rightarrow b$, entonces $C_i(a) < C_i(b)$
  - Si a es un evento de envío de un mensaje *m* de P_i y b un suceso de recepción de dicho mensaje por P_j, entonces $C_i(a) < C_j(b)$
  - Un reloj C_i nunca debe ir hacia atrás, cualquier corrección se realizará incrementando el reloj

Tiempo y relojes lógicos

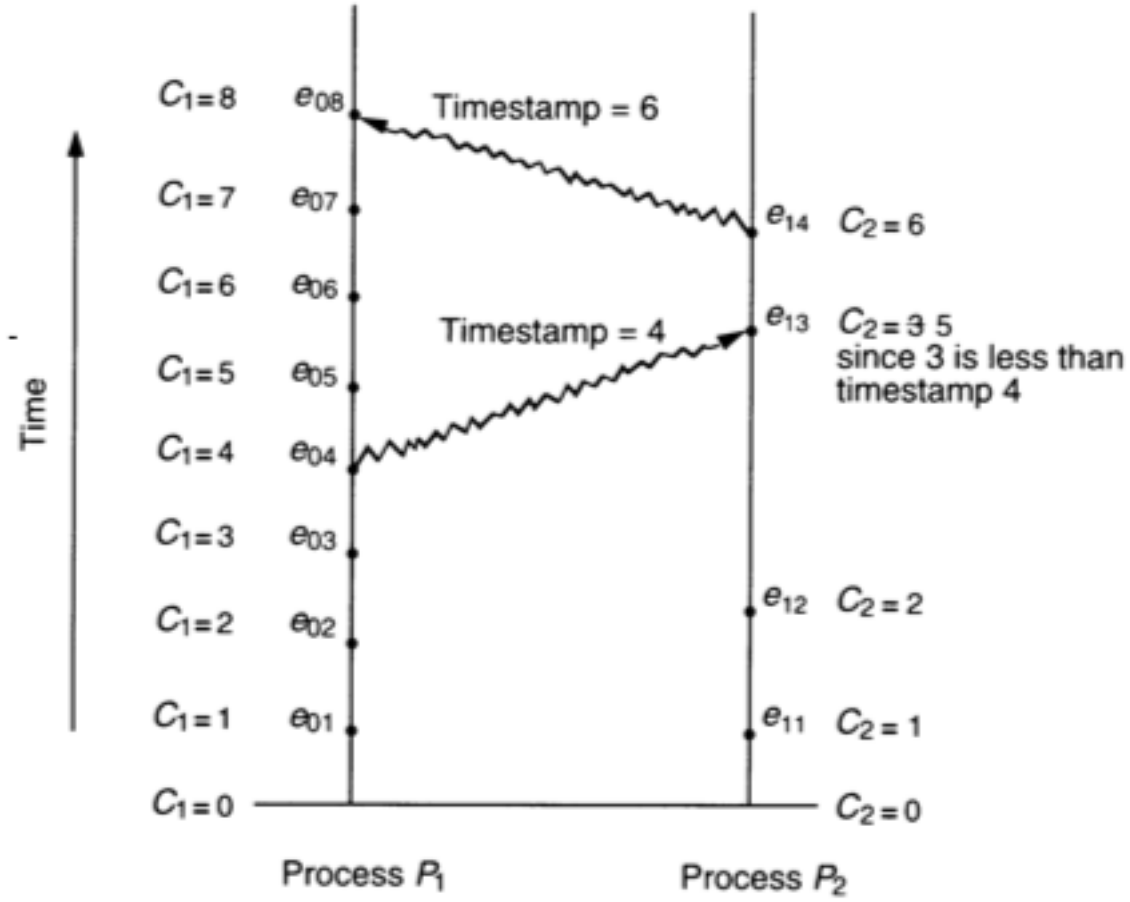
Algoritmo de Lamport

- LC1
  - Antes de cada evento relevante del proceso P_i:
    - $C_i = C_i + 1$
- LC2
  - Cuando P_i manda un mensaje m, le adjunta su timestamp $t=C_i$
  - Cuando P_j recibe un mensaje (m,t):
    - $C_j = \max(t, C_j) + 1$

- Este algoritmo captura todas las relaciones 'sucede antes que'
- La decisión de qué eventos de nuestro proceso son relevantes (y por tanto hay que aplicar LC1) depende de cada implementación

Tiempo y relojes lógicos

Algoritmo de Lamport: ejemplo



Tiempo y relojes lógicos

Algoritmo de Lamport: ordenación total

- En el algoritmo anterior, dos sucesos de procesos distintos pueden tener la misma marca de tiempo del reloj lógico
- Ordenación total:
  - Sean dos eventos que suceden en tiempos lógicos de Lamport T_i y T_j en los procesos P_i y P_j, respectivamente
    - $(T_i, i)<(T_j,j)$ si $T_i<T_j$ o $T_i=T_j$ y $i<j$
  - Básicamente, es un modo de 'desempatar' tiempos lógicos iguales mediante el identificador de proceso
- La ordenación total no tiene ningún sentido físico, pero puede ser útil para modelar tareas distribuidas, como por ejemplo la entrada en una sección crítica

Tiempo y relojes lógicos

Relojes vectoriales

- Limitación del algoritmo de Lamport
  - $C_i(e) < C_j(e')$ no implica necesariamente que e → e'
- Un reloj vectorial en un sistema de *N* procesos es un vector de *N* valores *V*
  - Cada proceso P_i guarda su propio vector V_i
    - $V_i[i]$ es el tiempo de Lamport de P_i
    - $V_i[j]$ (i!=j) es el último timestamp de P_j del que tiene constancia P_i
- Comparación de vectores lógicos
  - V=V' si V[i]=V'[i] para i=1, 2, …, N
  - V≤V' si V[i] ≤V'[i] para i=1, 2, …, N
  - V<V' si V≤V' y V≠V'

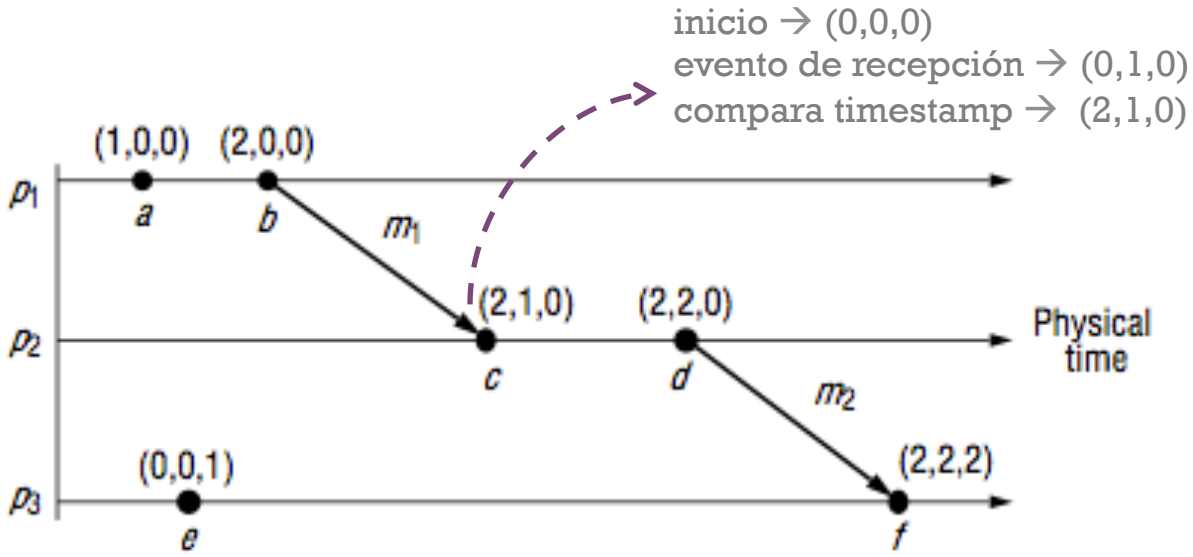
Tiempo y relojes lógicos

Relojes vectoriales: algoritmo

- VC1
  - Inicialmente $V_i[j]=0$ para $j=1, 2, \ldots, N$
- VC2
  - Antes de que ocurra un evento en P_i: $V_i[i]=V_i[i]+1$
- VC3
  - P_i incluye el timestamp $t=V_i$ en cada mensaje que envía
- VC4
  - Cuando P_i recibe un mensaje con timestamp *t*
    - $V_i[j]=\max(V_i[j], t[j])$ para $j=1, 2, \ldots, N$

Tiempo y relojes lógicos

Relojes vectoriales: ejemplo



- $V_a < V_f$, luego a → f
- No se cumple ni $V_e \leq V_c$ ni $V_c \leq V_e$ luego e y c son concurrentes
- Desventaja: carga en la transmisión de mensajes proporcional al número de procesos

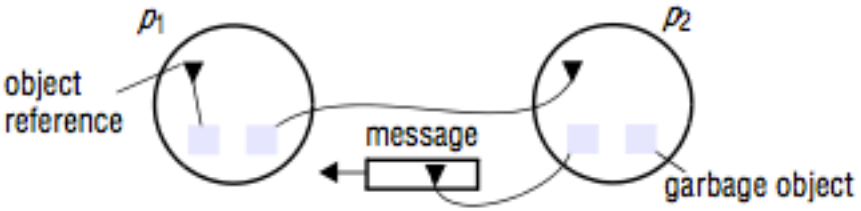
+ Tiempos y estados globales

- Introducción
- Sincronización
- Tiempo y relojes lógicos
- Estados globales
- Depuración distribuida

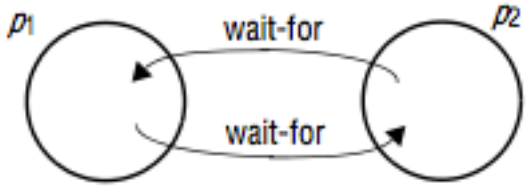
Estados globales

- 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 mensaje del otro
  c) Detectar la terminación de un algoritmo distribuido
- Es vital tener en cuenta el estado de los procesos **y** del canal de comunicación

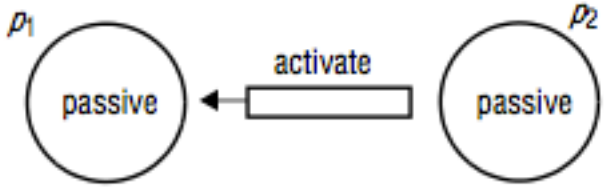
a) Recolección de basura



b) Interbloqueo



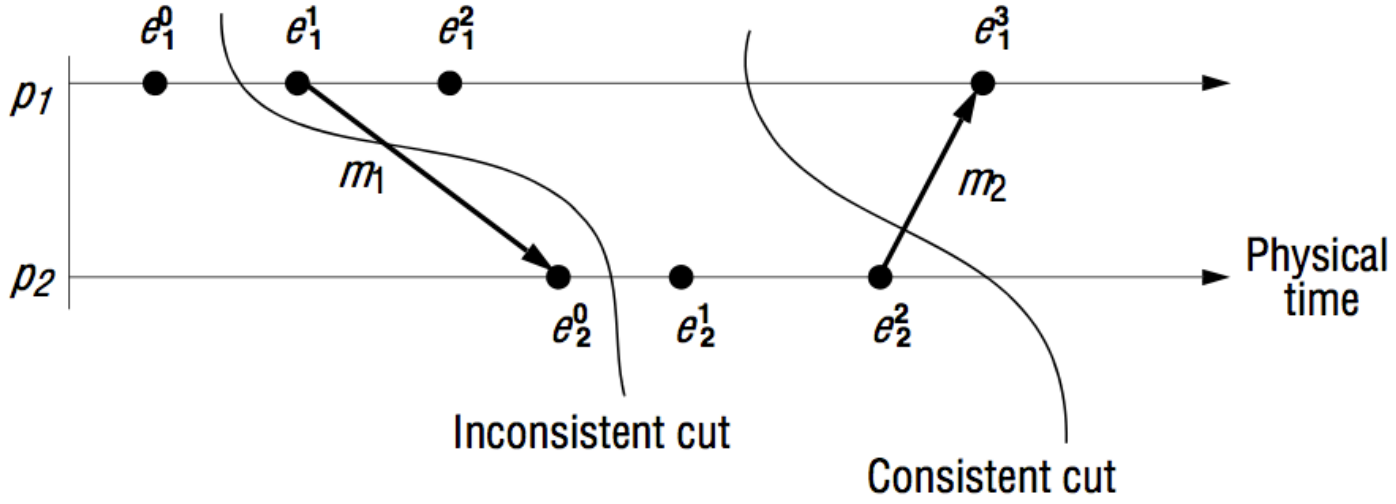
c) Terminación



Estados globales

Cortes consistentes

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

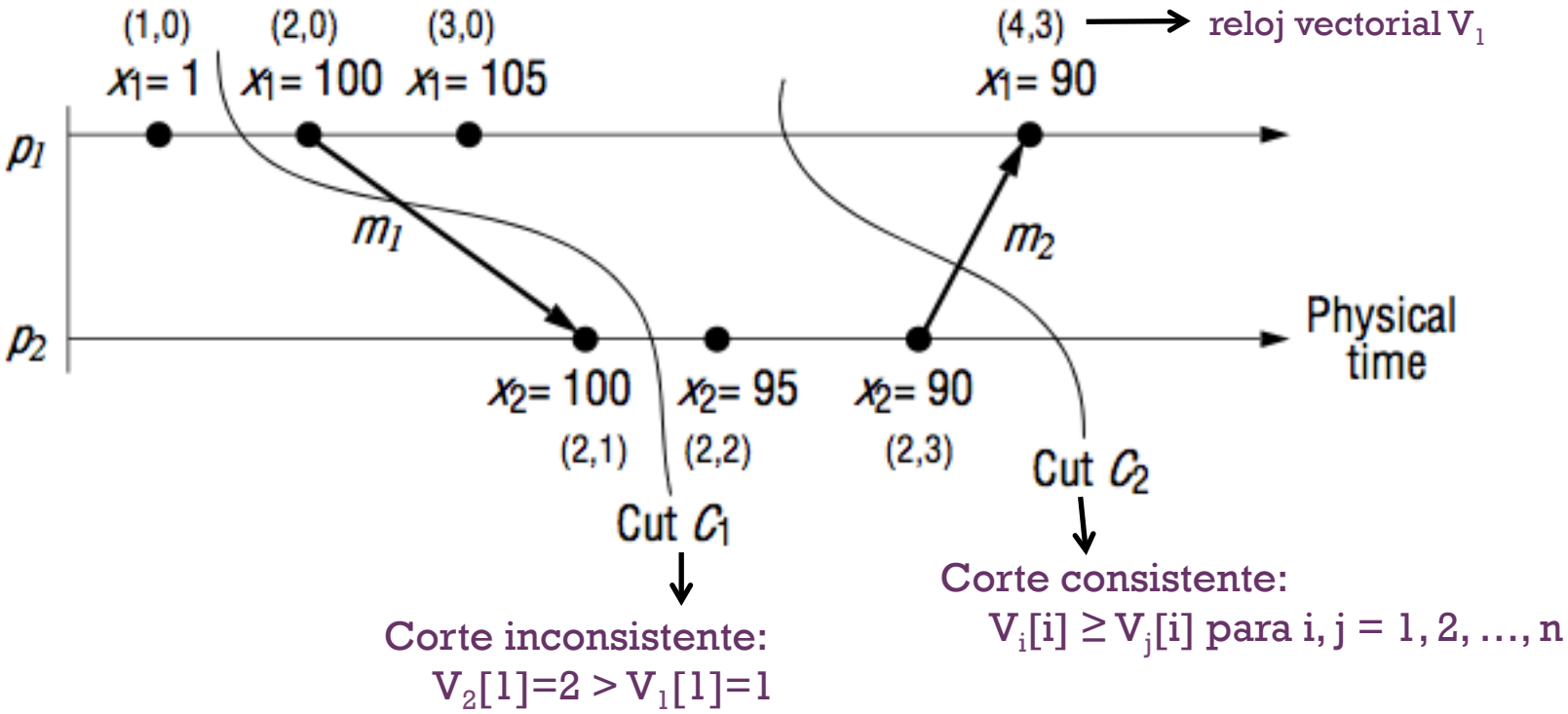


- Estado global consistente
  - Aquél que corresponde con un corte consistente

Estados globales

Evaluación de cortes con relojes lógicos vectoriales

- Un corte es consistente si, para cada proceso P_i, su reloj lógico en ese momento es mayor o igual que todos los registros del valor del reloj de P_i mantenidos por otros procesos



Estados globales

Algoritmo de instantánea de Chandy y Lamport

- Objetivo:
  - Obtener un conjunto de estados de proceso y del canal de comunicación (instantánea) que sea un estado global consistente
- Asunciones:
  - Los canales y procesos no fallan: todos los mensajes se reciben correctamente, y una única vez
  - Los canales son unidireccionales con entrega tipo FIFO
  - Hay canal de comunicación directo entre todos los procesos
  - Cualquier proceso puede tomar una instantánea en cualquier momento
  - Los procesos pueden continuar su ejecución y comunicación mientras se está tomando una instantánea

Estados globales

Algoritmo de instantánea de Chandy y Lamport

- Regla de recepción de *instantánea* en P_i por el canal *c*

  si (P_i no ha registrado su estado todavía)

  registra su estado de proceso

  registra el estado de *c* como vacío

  activa el registro de mensajes que lleguen por otros canales

  si no

  P_i registra el estado de *c* como el conjunto de mensajes recibidos en *c* desde que guardó su estado (mensajes posteriores a la *instantánea*)

- Regla de envío de *instantánea* por P_i

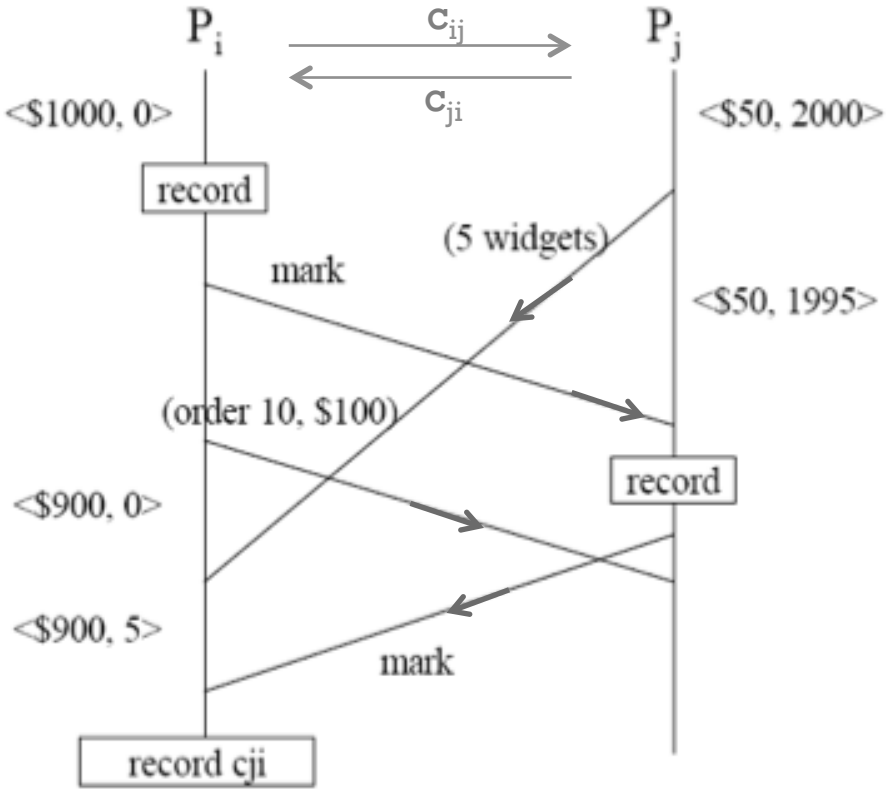
  Tras registrar su estado, para cada canal de salida *c*

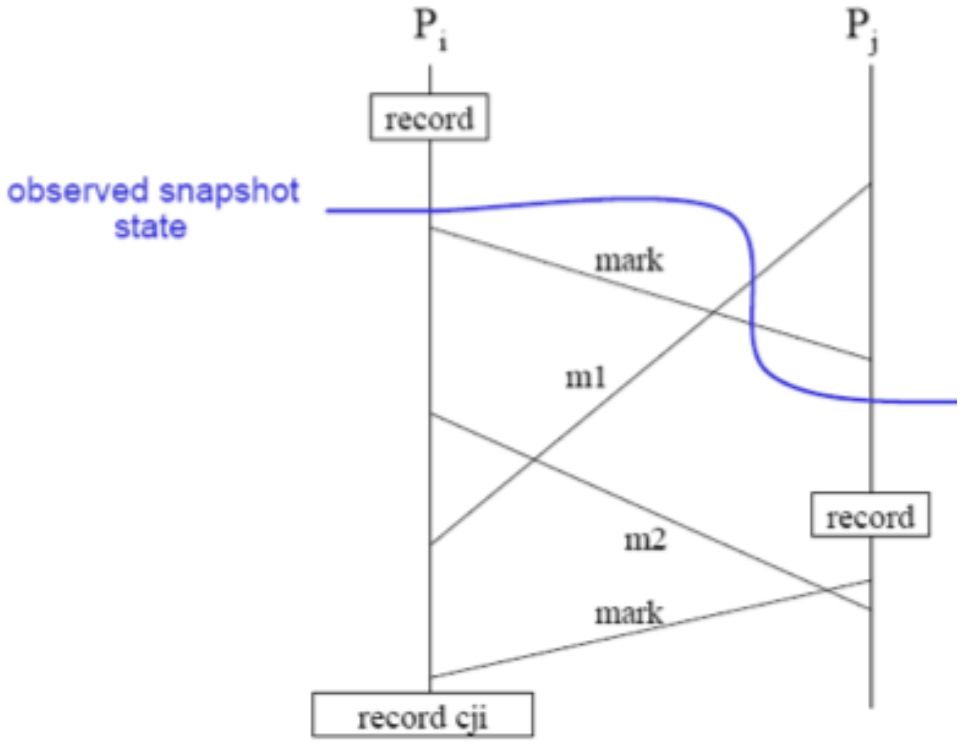
  P_i envía un mensaje de *instantánea* por el canal *c*

Estados globales

Algoritmo de instantánea de Chandy y Lamport

- Ejemplo: 2 procesos (P_i, P_j) hacen transacciones (mark=instantánea)
  - 2 canales de comunicación (c_{ij}, c_{ji})





estado registrado: P_i=<$1000, 0>, P_j=<50, 1995>, c_{ij}=<>, c_{ji}=<5 widgets>

+ Tiempos y estados globales

- Introducción
- Sincronización
- Tiempo y relojes lógicos
- Estados globales
- Depuración distribuida

Depuración distribuida

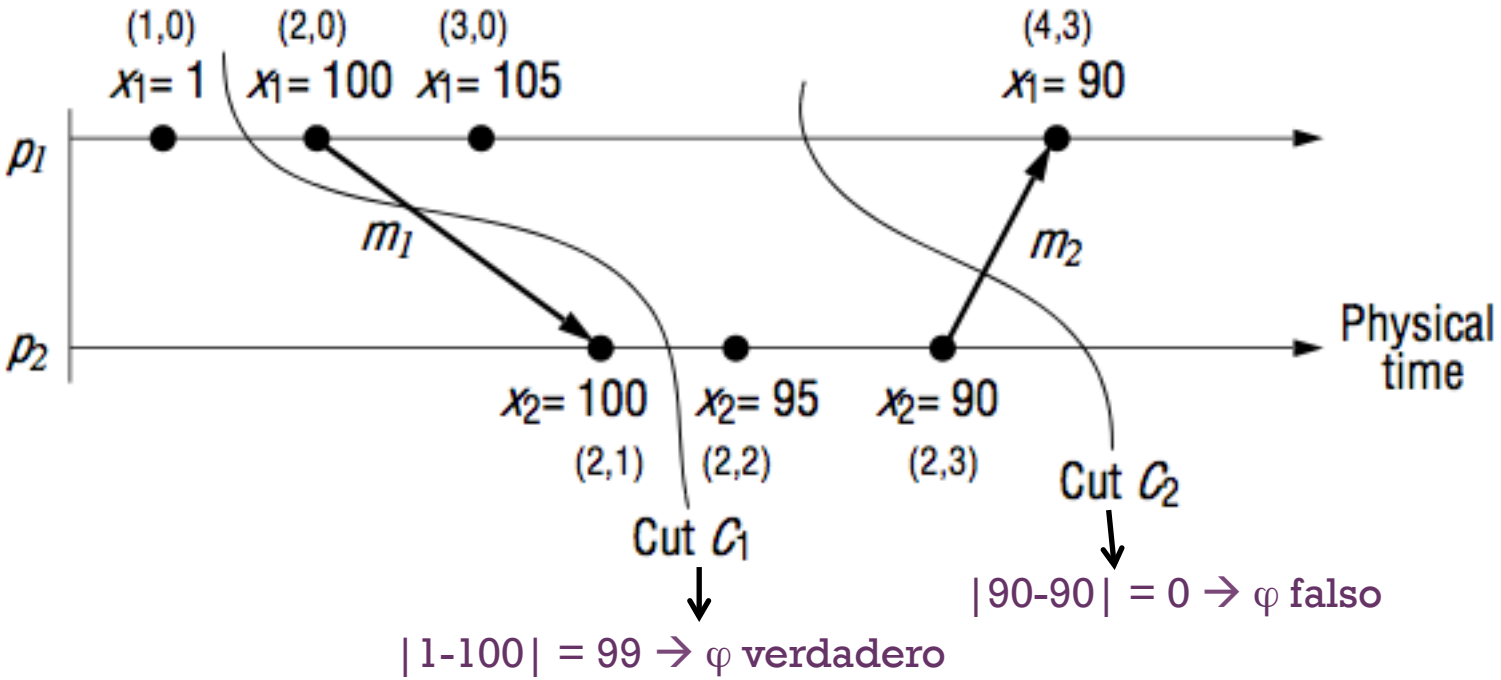
Predicados

- La ejecución de un SD se puede caracterizar (y depurar) por las transiciones entre estados globales consistentes
  - $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$
- Un **predicado** de estado global es una función
  - {Conjunto de estados globales} → {V, F}
  - Determinar una condición del SD equivale a evaluar su predicado
- Características posibles de un predicado
  - **Estabilidad**: el valor del predicado no varía con los nuevos sucesos (por ejemplo, en el caso de interbloqueo o terminación)
  - **Seguridad**: el predicado tiene valor falso para cualquier estado alcanzable desde S_0 (deseable para errores)
  - **Veracidad**: el predicado tiene valor verdadero para algún estado alcanzable desde S_0 (deseables para situaciones necesarias)

Depuración distribuida

Predicados: ejemplo

- Imaginemos un sistema de 2 procesos donde queremos controlar el predicado φ: $|x_1 - x_2| > 50$



Depuración distribuida

Monitorización

- Depurar un SD requiere registrar su estado global, para poder hacer evaluaciones de predicados en dichos estados
  - Generalmente, la evaluación trata de determinar si el predicado φ cumple con la condición "posiblemente" o "sin duda alguna".
- Monitorización del estado global:
  - Distribuido: algoritmo de instantánea de Chandy y Lamport
  - Centralizado: algoritmo de Marzullo y Neiger
    - Los procesos envían su estado inicial al proceso monitor
    - Periódicamente, le vuelven a enviar su estado
    - El monitor registra los mensajes de estado en colas de proceso
      - Una por proceso

Depuración distribuida

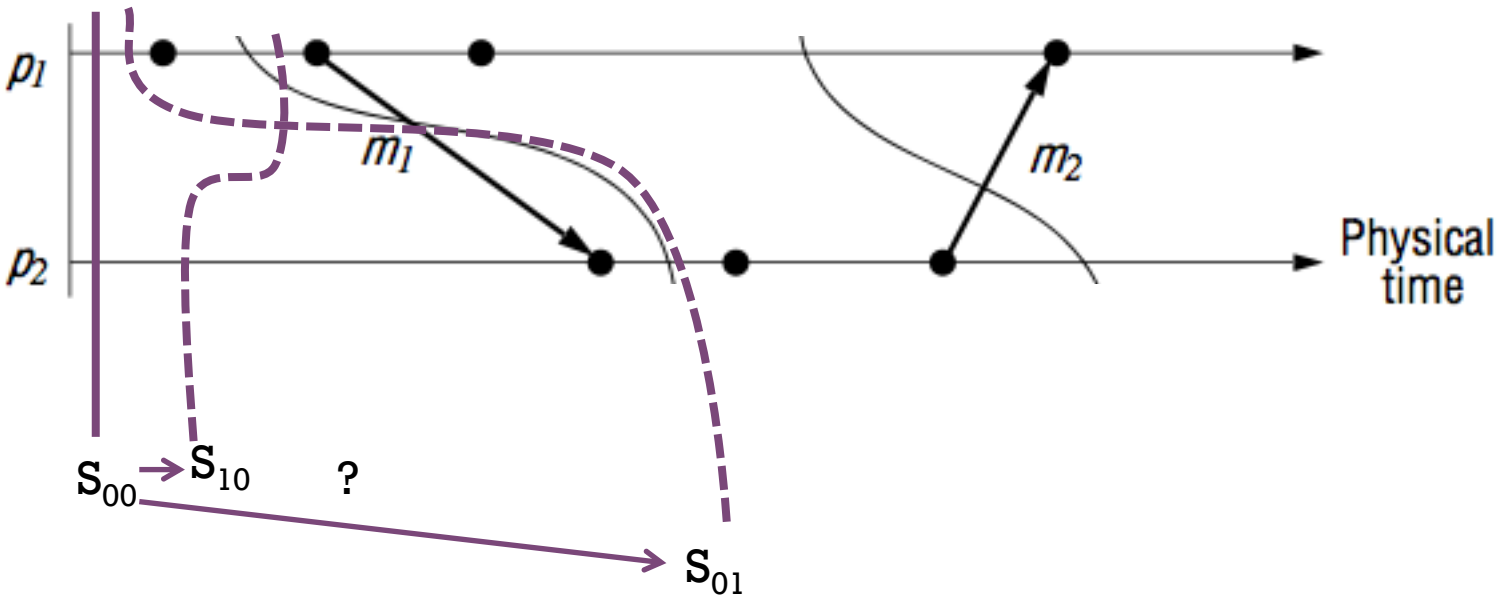
Evaluación de predicados

- Objetivo de la monitorización
  - Determinar si un predicado φ es "posiblemente" o "sin duda alguna" verdadero en un determinado punto de la ejecución
  - El proceso monitor sólo registra los estados globales *consistentes*
    - Los únicos en que podemos evaluar el predicado con certeza
- Teniendo en cuenta el predicado a evaluar, podemos reducir el tráfico de mensajes de estado
  - Tamaño: el predicado puede depender sólo de ciertas partes del estado de un proceso → no es necesario mandar el estado completo
  - Número: el cambio de valor del predicado sólo ocurre en algunos casos → sólo hay recoger los estados en cambios relevantes

Depuración distribuida

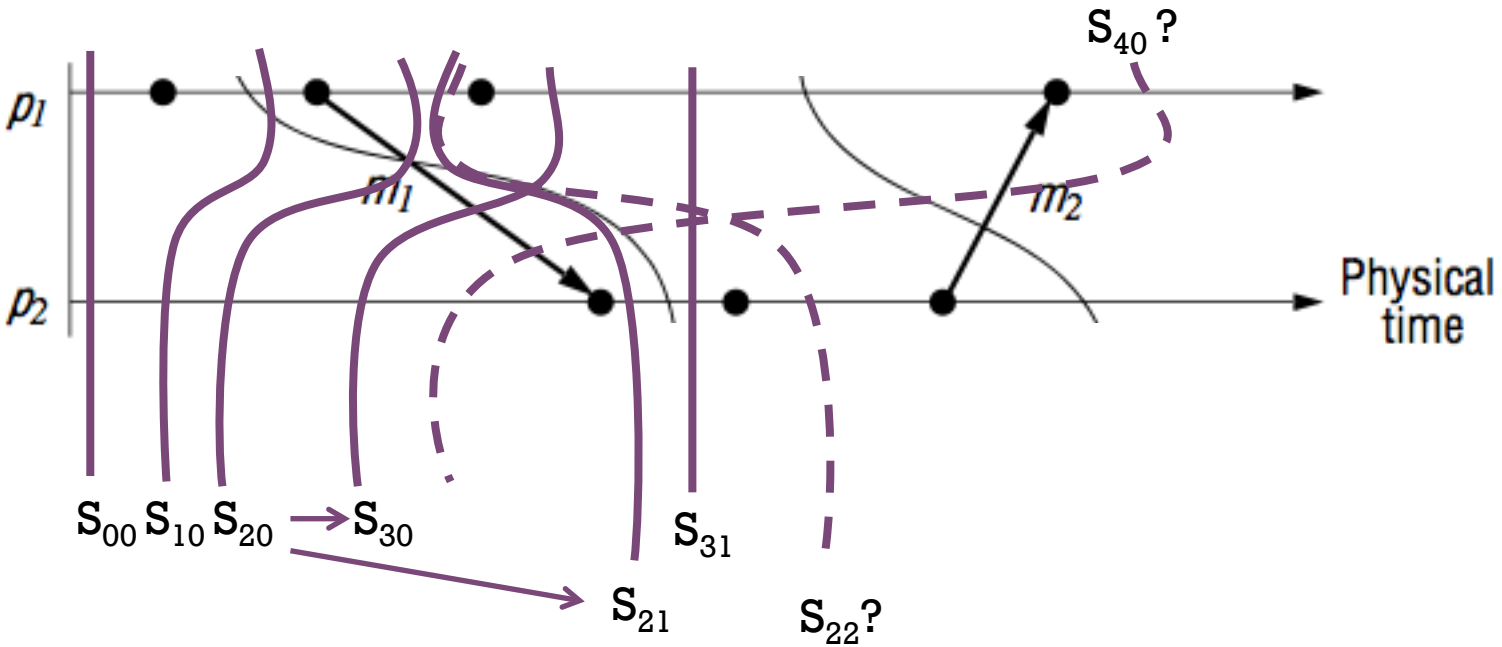
Red de estados globales

- Mediante la monitorización podemos construir una red de estados globales consistentes
  - S_{ij}=estado global tras i eventos en el proceso 1 y j eventos en el proceso 2



Depuración distribuida

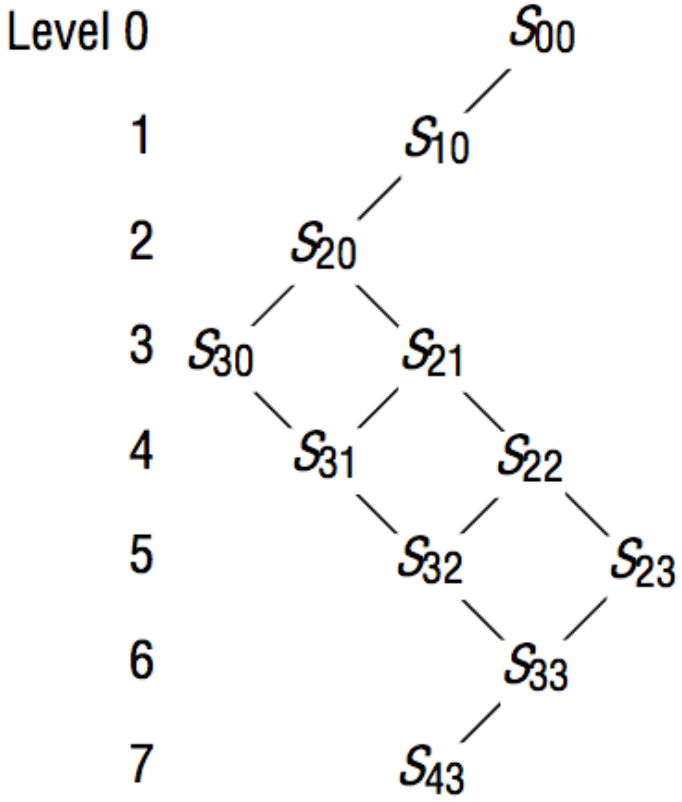
Red de estados globales: ejemplo



Depuración distribuida

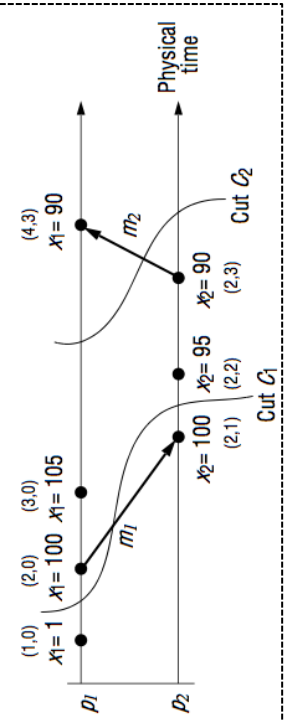
Red de estados globales

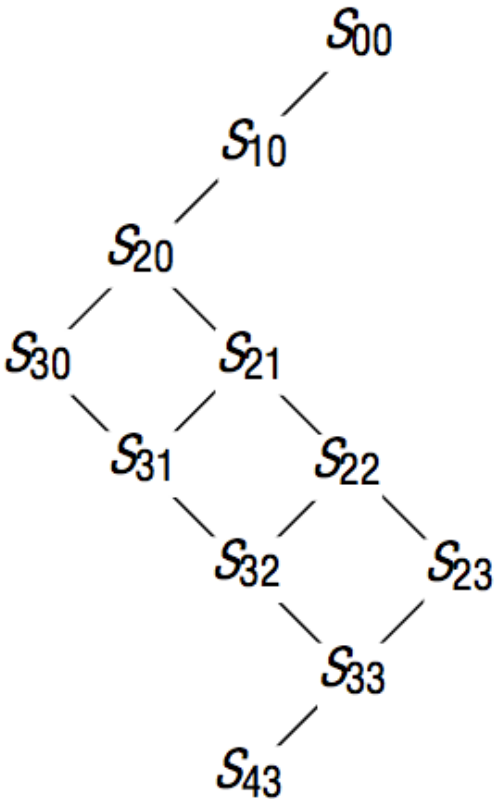
- **Linealización**: ruta entre estados
- **Posiblemente** φ: existe un estado consistente S a través del que pasa una linealización tal que φ(S) = Verdadero
- **Sin duda alguna** φ: existe un conjunto de estados consistentes S* a través del que pasan todas las linealizaciones, tal que, para todo S en S*, φ(S)=Verdadero



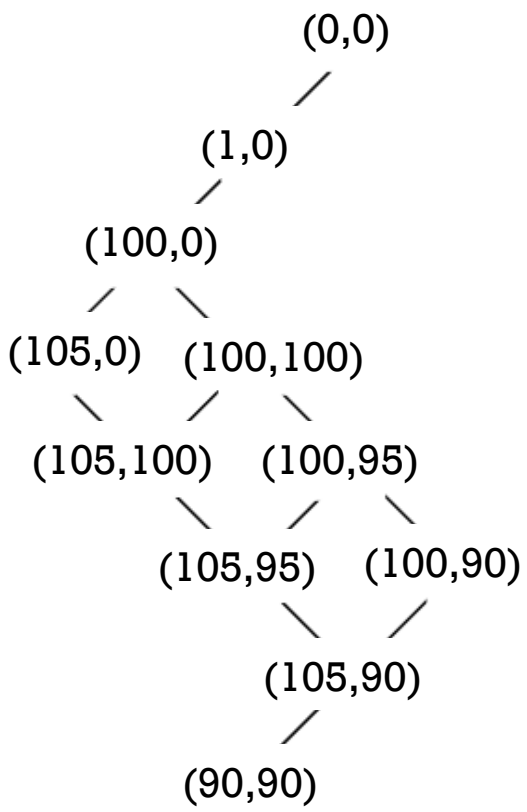
Depuración distribuida

Evaluación "instantánea" de predicados

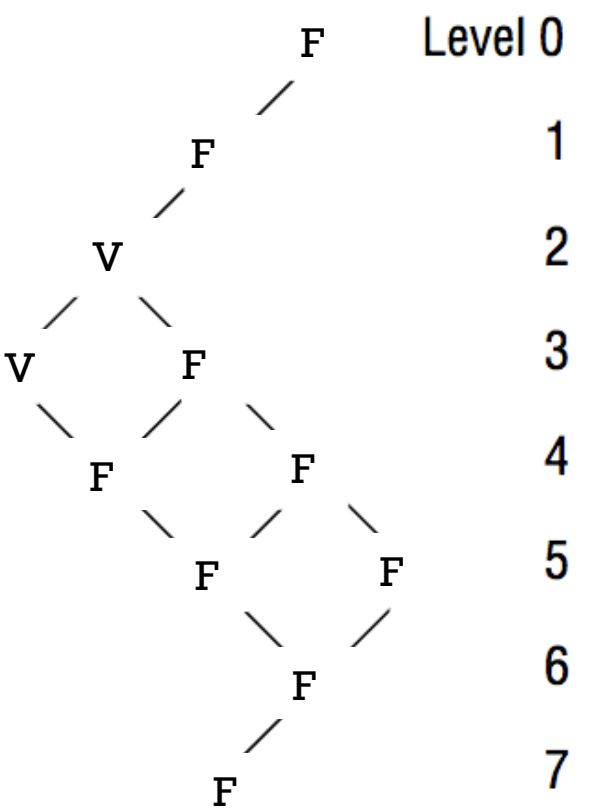




(x1,x2)



φ: |x1-x2| > 50?



Depuración distribuida

Evaluación de predicados posiblemente

```
Evaluar posiblemente φ para la red H de N procesos
  L=0;   //Nivel de la red de estados
  Estados={(s⁰1, s⁰2 … s⁰N )};  //Estados del nivel L
  mientras ( φ(si) = Falso para todos los si en Estados)
    L=L+1;
    Alcanzable = { S' tal que S' es alcanzable en H desde
    algún S en Estados y nivel(S') = L };
    Estados = Alcanzable;
  fin mientras
  si L <= {nivel máximo de H}  salida "posiblemente φ";
```

- Recorremos los estados alcanzables de cada estado inicial
  - Hasta que en algún momento alguno de los estados cumpla que $\varphi(s_i)$ = Verdadero, o terminemos de recorrer la red

Depuración distribuida

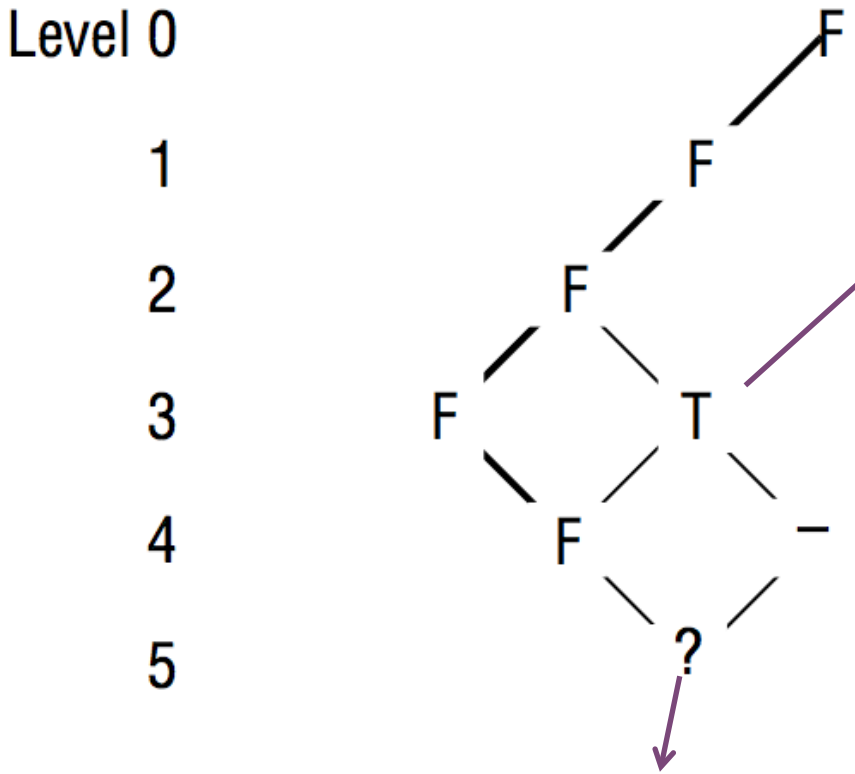
Evaluación de predicados sin duda alguna

```
Evaluar sin duda alguna φ para la red H de N procesos
  L=0;
  si (φ(s⁰1, s⁰2 … s⁰N))  Estados={};
  si no                   Estados={(s⁰1, s⁰2 … s⁰N)};
  mientras (Estados != {})
    L=L+1;
    Alcanzable = { S' tal que S' es alcanzable en H desde
    algún S en Estados y nivel(S') = L };
    Estados = {S en Alcanzable con φ(S)=Falso};
  fin mientras
  salida "sin duda alguna φ";
```

- Recorremos los estados alcanzables desde cada estado inicial
  - Hasta que en algún momento <u>todos</u> los estados cumplan con el predicado ($\varphi(s_i)$ = Verdadero) o terminemos de recorrer la red

Depuración distribuida

Evaluación de predicados: ejemplo



En este punto se puede decir que el predicado se cumple "posiblemente"

En algoritmo "sin duda" sólo investiga los estados alcanzables desde estados donde el predicado es falso. Por ello, en el nivel 4 no hace falta investigar este estado

Si en el único estado del nivel 5 tenemos verdadero (T), entonces el predicado se cumple "sin duda", pues no se puede seguir trazando una ruta de falsabilidad

Depuración distribuida

Resumen

- Consiste en
  - Determinar el predicado que queremos evaluar
  - Especificar un método para construir una red o historia de estados globales consistentes
    - Teniendo en cuenta el predicado para optimizar tráfico
  - Evaluar si nuestro predicado se cumple en algún momento
    - Si es posible, se cumplirá para alguna de las linealizaciones
    - Si es sin duda, se cumplirá para todas las linealizaciones

Referencias

- G. Colouris, J. Dollimore, T. Kindberg and G. Blair. *Distributed Systems: Concepts and Design (5th Ed)*. Addison-Wesley, 2011
  - Capítulo 14
- Sobre UTC, TAI, UT1 y los segundos de salto
  - http://what-if.xkcd.com/26/
- Sobre NTP
  - http://www8.cs.umu.se/kurser/5DV020/HT07/ntp.pdf
  - Wikipedia: "Network Time Protocol", "Marzullo's algorithm"

http://xkcd.com/612/

THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.