Tema 4.1: Sincronización

Sistemas Distribuidos

2022-2023

Servicios de Infraestructura

Qué son los servicios de infraestructura

- Se construyen sobre los mecanismos de comunicación entre procesos antes vistos,
- Proporcionan funcionalidad necesaria para construir *aplicaciones* distribuídas.
- Pueden considerarse una capa maś entre el *middleware* y la aplicación.

Ejemplos de servicios de infraestructura:

- Sincronización
- Coordinación
- Autenticación
- Seguridad
- Nombrado

Sincronización

Introducción

Algunos algoritmos dependen del tiempo:

- Necesitando conocer la hora exacta
- O necesitando conocer en qué orden ocurrieron los eventos

> El orden sería fácil de determinar si hubiera un reloj global.
>
> Pero un sistema distribuído **no lo tiene**

Los relojes y la hora

- El reloj de un computador se basa en medir las oscilaciones de un cristal de cuarzo.
- Un circuito cuenta las oscilaciones.
- Cuando alcanza un cierto valor, genera una interrupción
- El operativo cuenta las interrupciones.

El oscilador de cada computador es ligeramente diferente.

⇒ Algunos relojes adelantan, otros atrasan.

Sincronización de relojes físicos

El problema es "mantener en hora" todos los nodos del sistema distribuido.

> Si uno de ellos posee la hora correcta (ej: un GPS), los demás usarán esa como referencia.

Problemas:

- ¿Cada cuánto tiempo se resincronizan?
- ¿Cómo tener en cuenta el retardo de la red?
- ¿Cómo actualizar la hora local?

1: Frecuencia de sincronización

Sea $C(t)$ el valor del reloj de la máquina en el instante en que la hora real es t.

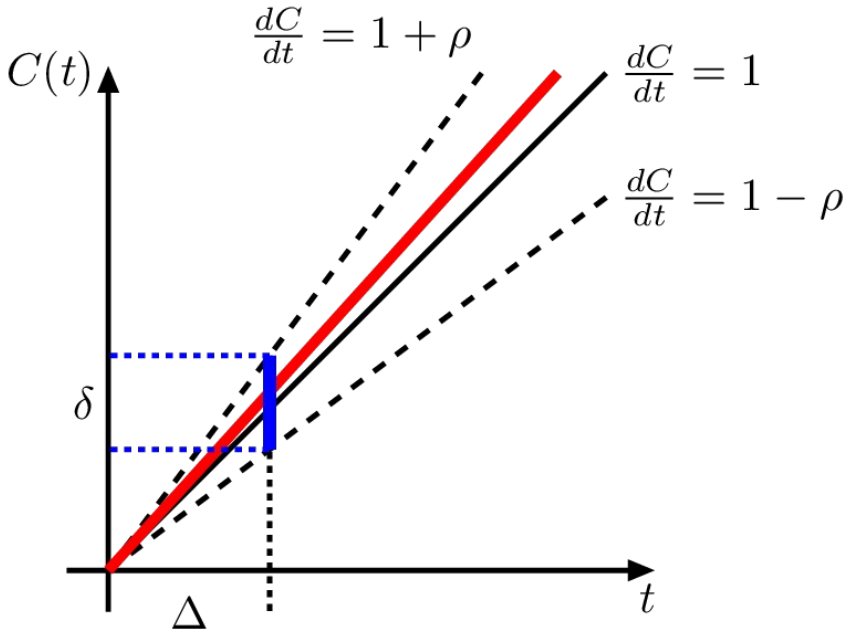
Teniendo en cuenta que para un reloj exacto $C(t)=t$,

- Para un reloj exacto $dC/dt=1$
- Si el reloj adelanta $dC/dt>1$
- Si el reloj atrasa $dC/dt<1$

En general no conoceremos dC/dt para un reloj, pero sí un margen de tolerancia ρ tal que $dC/dt=1 ± \rho$

> ρ = **ratio máximo de deriva**
> (suministrado por el fabricante del reloj)

paso[



Pasado un tiempo Δ un reloj se habrá alejado de la hora exacta, como máximo, $\rho\Delta$.

Dos relojes con un mismo ρ se habrán alejado entre sí, como máximo, $\delta=2\rho\Delta$

1: Frecuencia de sincronización

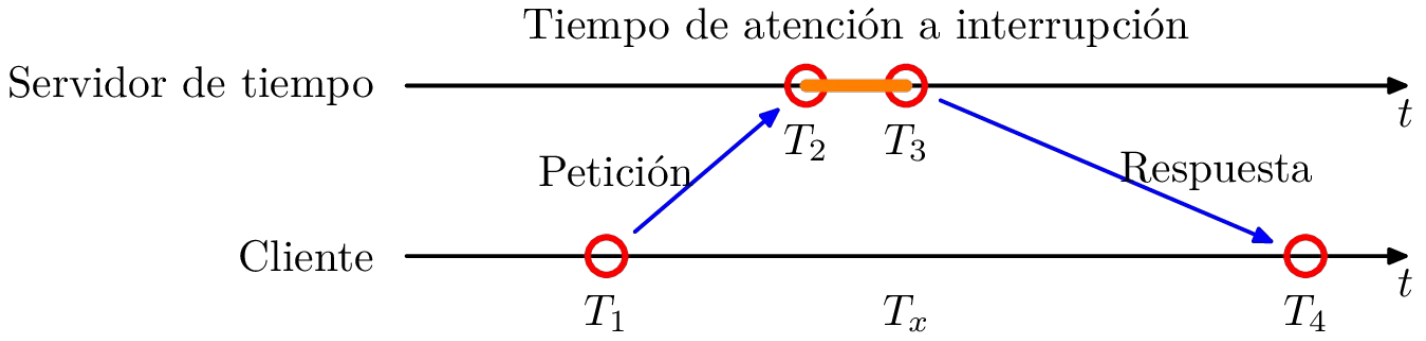
Conclusión

Si no queremos **discrepancia** mayor de δ segundos entre dos relojes del sistema, deben sincronizarse con el exacto cada $$t_\text{sincr} \leq \frac{\delta}{2\rho}$$ segundos.

2: El retardo de la red

Para obtener la hora de un servidor debo usar la red:

- T_1: Envío mensaje pidiendo hora
- T_2: En algún momento llega mi mensaje al servidor
- T_3: En otro momento posterior éste obtiene la hora real T y me la envía en otro mensaje
- T_4: Recibo un mensaje que dice que la hora es T



¡Para cuando el mensaje llega en el instante ${T_{4}}$, T ya no es la hora exacta!

Para obtener la hora exacta habría que sumar a T el tiempo T_x-T_3, pero desconocemos esa cantidad.

2: El retardo de la red

Si se conociera qué valor T_x marcaba el reloj del cliente cuando el servidor marcaba T_3, se deduciría:

- El valor T_4-T_x sería el retardo de la red en la respuesta
- La hora T recibida debería ser corregida en esa cantidad

Solución [Cristian 1989]

En ausencia de más información una buena aproximación es considerar que $$T_x = (T_2 - T_1)/2$$ (O sea, en el punto medio)

3: Actualizar reloj local

¡El reloj nunca debería retroceder!

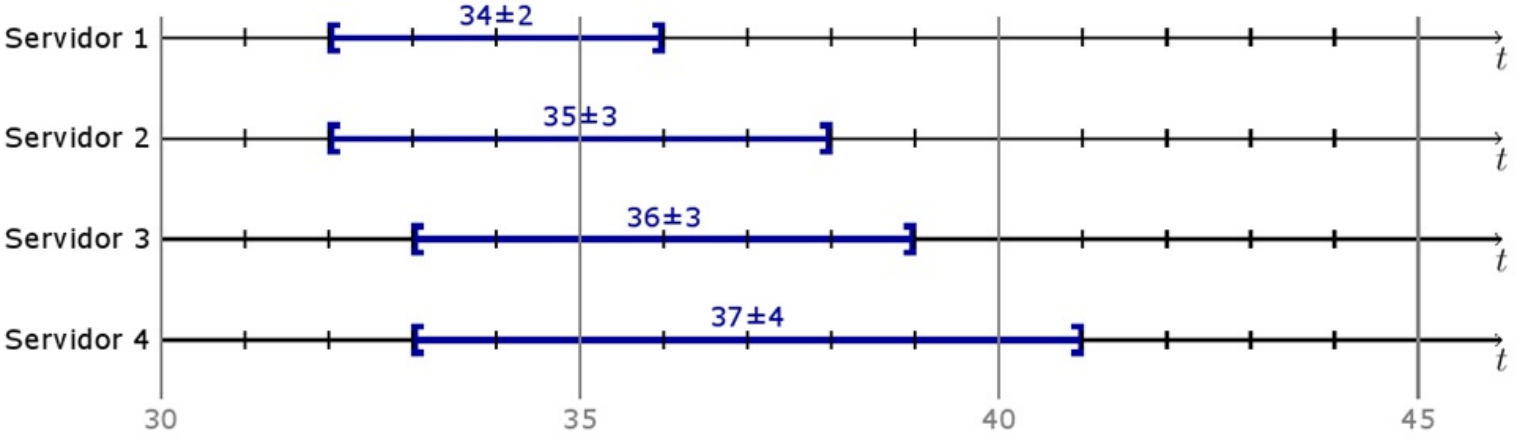
Podría "estropear" aplicaciones como `make`

Solución

- Hacerlo atrasar (ir más lento) hasta que la hora real le alcance.
- Manipulando la frecuencia con la que se generan interrupciones de reloj.

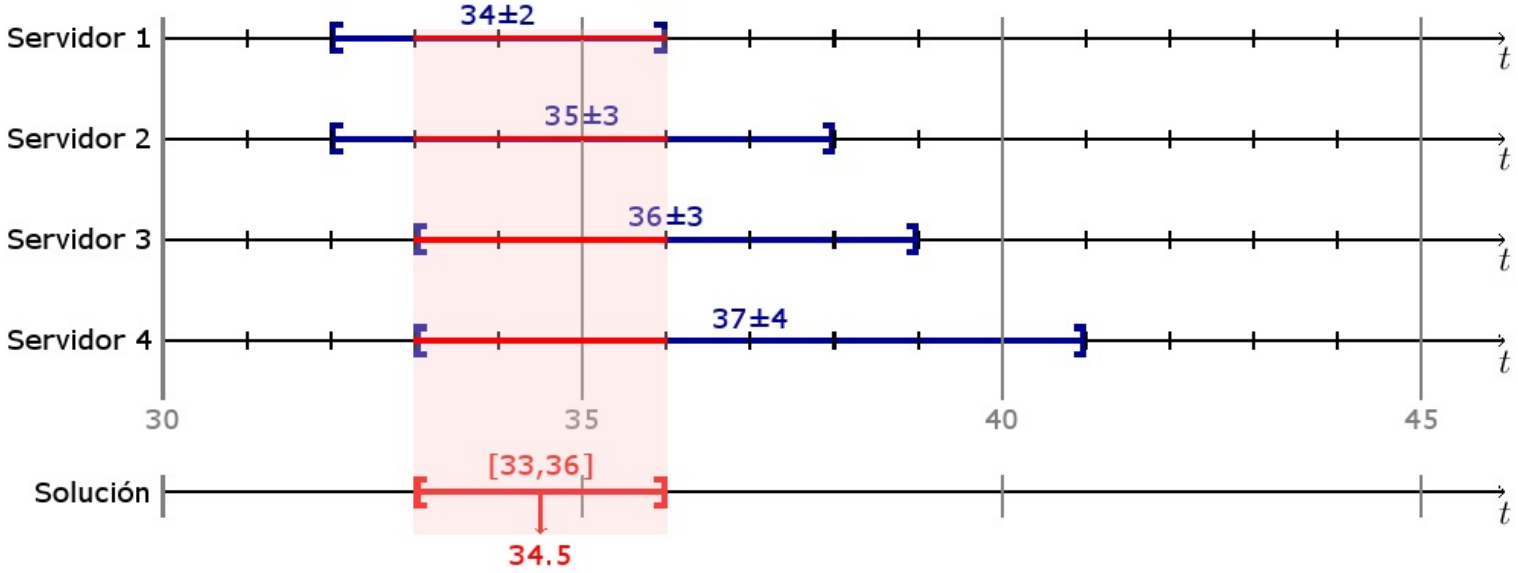
Método de Marzullo

- En lugar de preguntar a un solo servidor, pregunta a varios
- La respuesta incluye además un "margen de error"
- ¿Cómo estimar la hora correcta a partir de las respuestas?



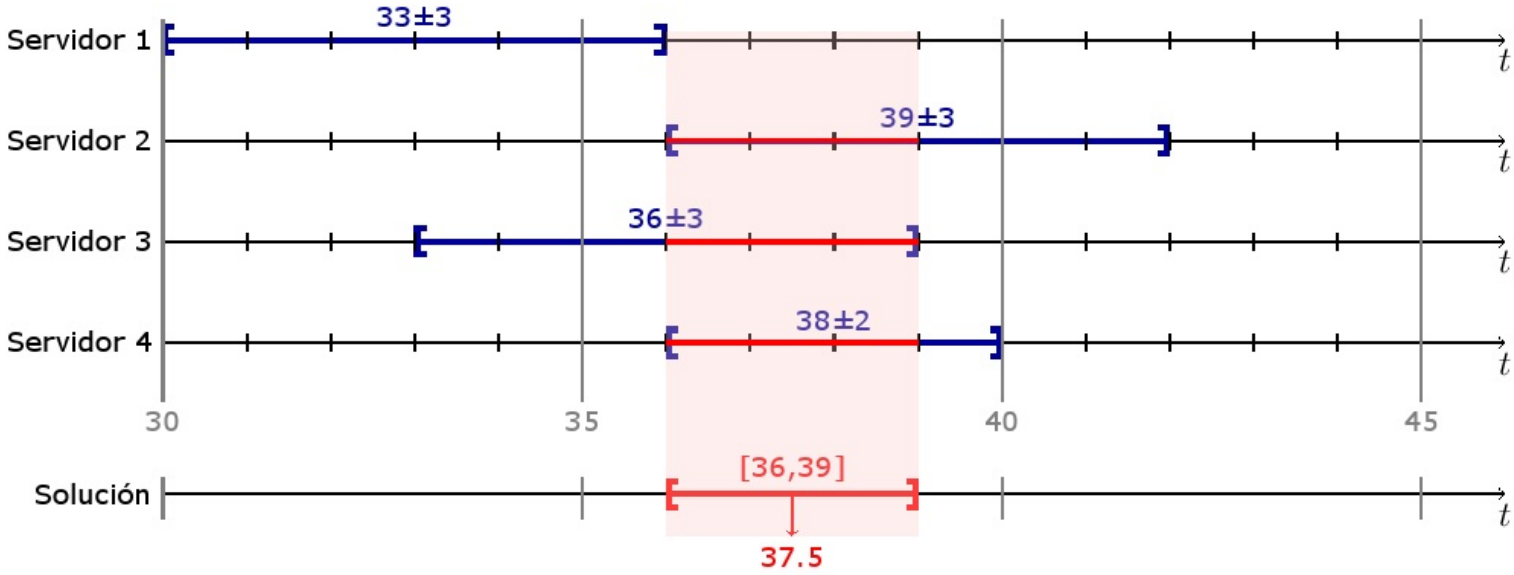
Solución

- Buscar un intervalo con el **máximo de solapamientos** entre respuestas.



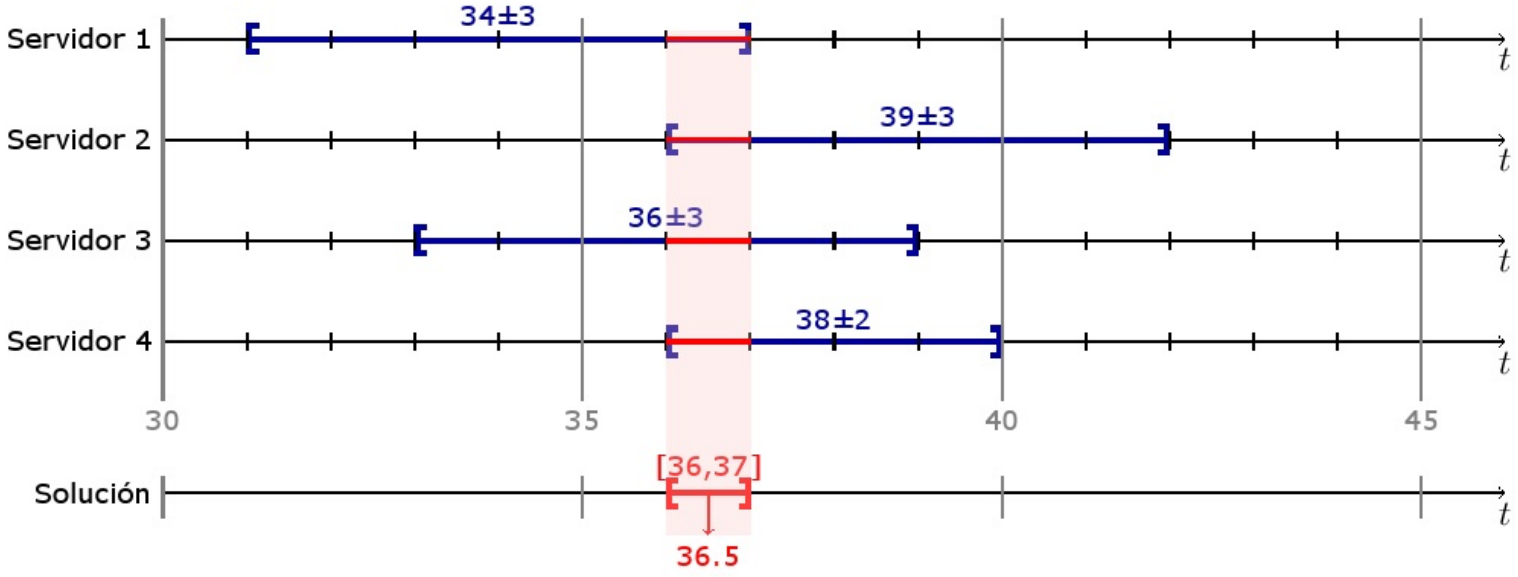
Otro ejemplo

- En este caso el máximo de solapes es 3 y una respuesta se ignora.



Observación

Una variación ligera en una respuesta puede llevar a un resultado muy diferente.



Algoritmo de Marzullo

Para encontrar el intervalo antes descrito.

```
mejor = 0; contador = 0;
izquierda = 0; derecha = 0;
Ordenar de menor a mayor todos los bordes de intervalo
Para cada instante t de apertura o cierre de intervalo:
    if (t es apertura):
        contador = contador + 1
    else:
        contador = contador - 1
    if (contador>mejor):
        mejor = contador
        izquierda = t
        derecha = t siguiente (abra o cierre)

Al finalizar:
  Solución = [izquierda, derecha]
  N_solapamientos = mejor
```

Relojes lógicos

¿Qué es un reloj lógico?

- Algunos algorimos no necesitan conocer la *hora exacta*
- Sólo necesitan una **ordenación** de los eventos
  (saber qué evento fue anterior a otro)

> Un reloj lógico es un contador que permite deducir relaciones del tipo "A ocurrió antes que B", pero no está relacionado con el tiempo real.

Eventos en una misma máquina

El contador de interrupciones, siempre que sea monótono creciente, sirve como reloj lógico:

- El evento a ocurre cuando el contador vale N_a
- El evento b ocurre cuando el contador vale N_b
- $N_a < N_b \Rightarrow t_a < t_b$ (a ocurrió antes que b)

No necesitamos conocer las horas exactas t_a, t_b, podemos comparar los contadores.

Eventos en máquinas separadas

En este caso, ya que cada máquina tiene su propio contador, **no se pueden comparar** éstos para saber cuál fue anterior.

Pero cuando una máquina envía un mensaje a otra, entonces hay una relación de **causalidad** entre el evento "enviar" y el evento "recibir"

- El evento a es un envío de mensaje
- El evento b es una recepción (en otra máquina)
- Sabemos que b es posterior a a, y por tanto $t_b > t_a$

Idea de **Lamport** [1978]:

Forzar a que también $N_b<N_a$ para este caso.

Algoritmo de Lamport

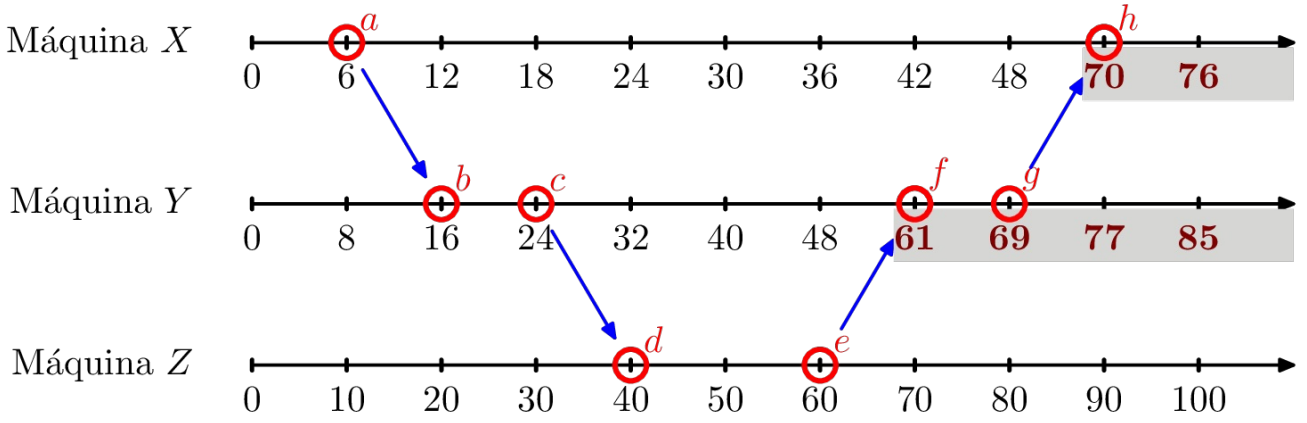
- El reloj lógico es el contador que mantiene cada máquina
- Pero ese reloj debe actualizarse con cada recepción de mensaje

Cómo

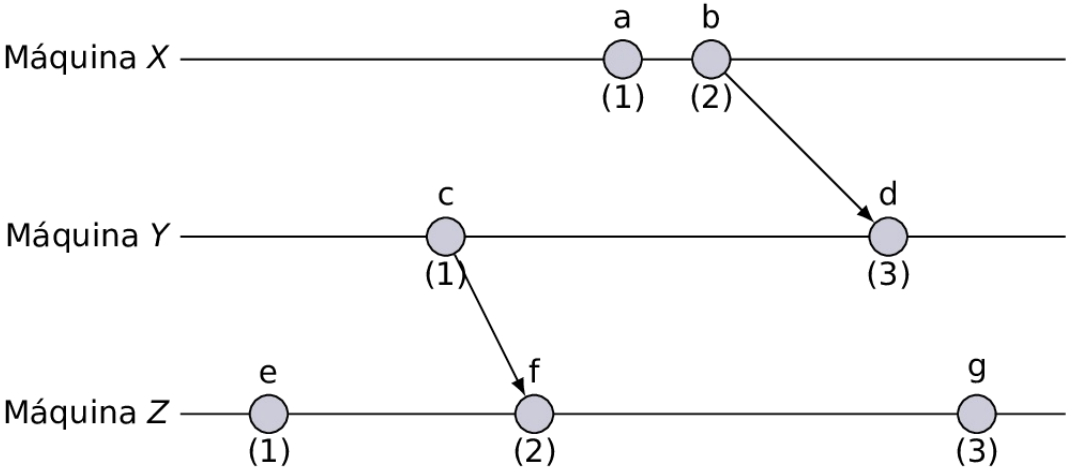
- Cuando A envía un mensaje a B incluye como parte del mensaje el valor de su reloj lógico (N_A)
- Cuando B recibe el mensaje, compara el valor que llega con su propio reloj lógico N_B
  - Si $N_B<N_A$, cambiar N_B y hacerlo igual a N_A+1

Eso garantiza que $N_B > N_A$

Ejemplo de aplicación del algoritmo



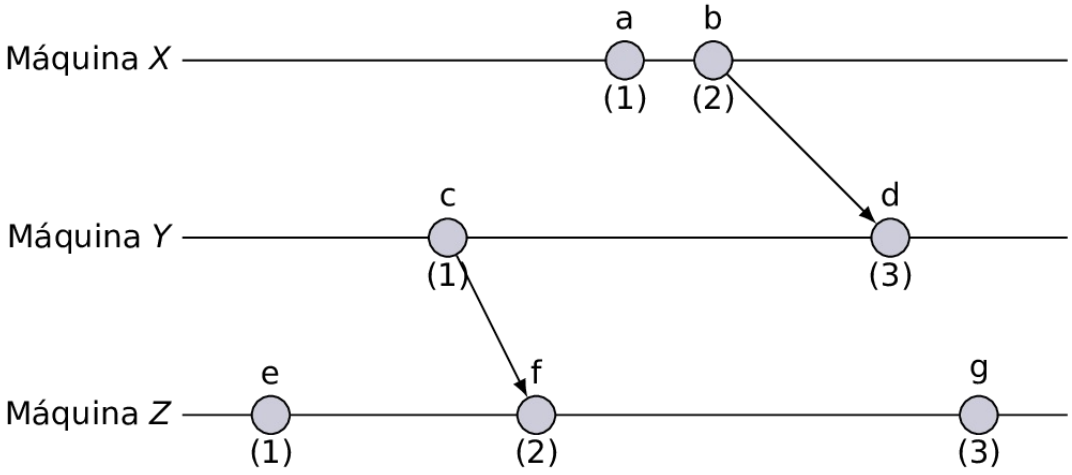
Propiedades del reloj lógico de Lamport



Si a es causa de d, se cumplirá que $t_a < t_d$ (relojes de "hora real").

En este ejemplo podemos ver que el reloj lógico también lo cumple ($1<3$)

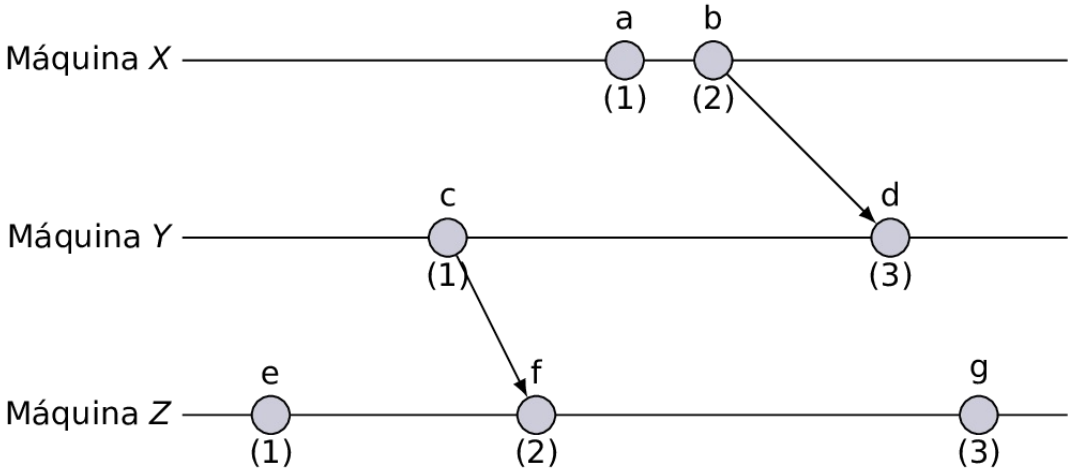
Propiedades del reloj lógico de Lamport



Pero la inversa no es cierta en general.

El que el reloj lógico nos ordene dos eventos, no implica que ese orden ocurra en el mundo real **si no hay relación causal entre eventos**

Propiedades del reloj lógico de Lamport



Ejemplo

- $C(a)<C(f)$ y no es cierto que a causó f.
- Ni siquiera se puede deducir que $t_a<t_f$

El reloj lógico no captura *causalidad* (extensiones posteriores, como el vector de relojes de Lamport, sí)

El reloj lógico define un orden parcial

Dos eventos A y B que ocurran en máquinas separadas estarán en uno de estos tres casos:

- $A\prec B$. El evento A es **anterior** a B y *puede demostrarse* (porque hay relación causal entre ellos u ocurren en la misma máquina).
- $A\succ B$. El evento A es **posterior** a B y *puede demostrarse* (por las mismas razones que antes)
- $A||B$. Se dice que los eventos son **concurrentes**. No puede demostrarse cuál es anterior o posterior (no hay relación causal).

Sólo observando $C(A)$ y $C(B)$ no puede saberse en qué caso estamos. Hay que ver "la figura".

Ejemplos (fig anterior). $(a\prec d)$, $(g\succ c)$, $(a||f)$, incluso $(a||e)$.