CC5303 – Sistemas Distribuidos

5.- Sincronización

Parte 1

Sebastián Blasco V.

Sincronización en SSDD

Un SSDD debe saber sortear correctamente la concurrencia:

- La información está distribuida en diferentes nodos
- Se toman decisiones sólo en base a la información local.
- Debe evitarse un punto único de fallo.
- No existe un reloj común.

La concurrencia exige sincronización entre procesos.

Sincronización en SSDD

¿Qué es el tiempo?

tiempo

Del lat. tempus.

1. m. Duración de las cosas sujetas a mudanza.
2. m. Magnitud física que permite ordenar la secuencia de los sucesos, estableciendo un pasado, un presente y un futuro, y cuya unidad en el sistema internacional es el segundo.
3. m. Parte de la secuencia de los sucesos.

El tiempo en un esquema distribuido **es ambiguo**, no es un valor absoluto, preciso ni común a todos los nodos.

Lograr un acuerdo respecto al tiempo en un SSDD no es trivial.

Sincronización en SSDD

Problemas de sincronización a considerar:

- Tiempo y estados globales.
- Exclusión mutua.
- Algoritmos de elección.
- Problemas de coordinación
- Operaciones atómicas distribuidas
  - *Transacciones*

Motivación para sincronizar

Desorden: Operaciones Bancarias

Aplicación transacciones bancarias:

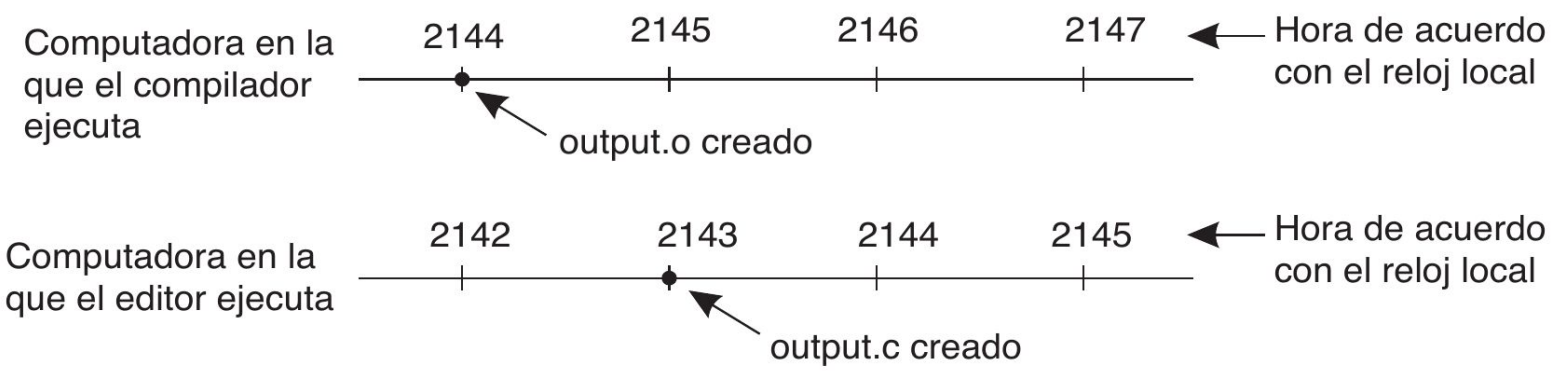
- Sobre una cuenta con $1000
  - Agregar $100
  - Aplicar 5% de interés

$1000	$1000
$1100	$1050
$1155	$1150

Motivación para sincronizar

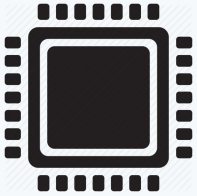
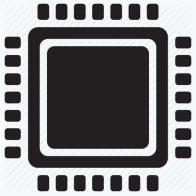
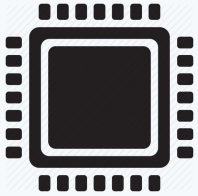
Descoordinación: El caso de *make*

En *make*, para compilar archivos, se verifican los tiempos de modificación de los archivos fuentes y generados. Cuando el tiempo de modificación del fuente es mayor que el del generado, se necesita recompilar.



Relojes Físicos

Un reloj en una computadora viene dado por un contador de ciclos que es capaz de llevar un procesador.

En un SSDD de *n* computadoras, se contemplan *n* relojes que funcionan a velocidades (a lo menos) ligeramente distintas. A la diferencia de valores reportadas por los relojes en cuestión se le denomina **distorsión de reloj**

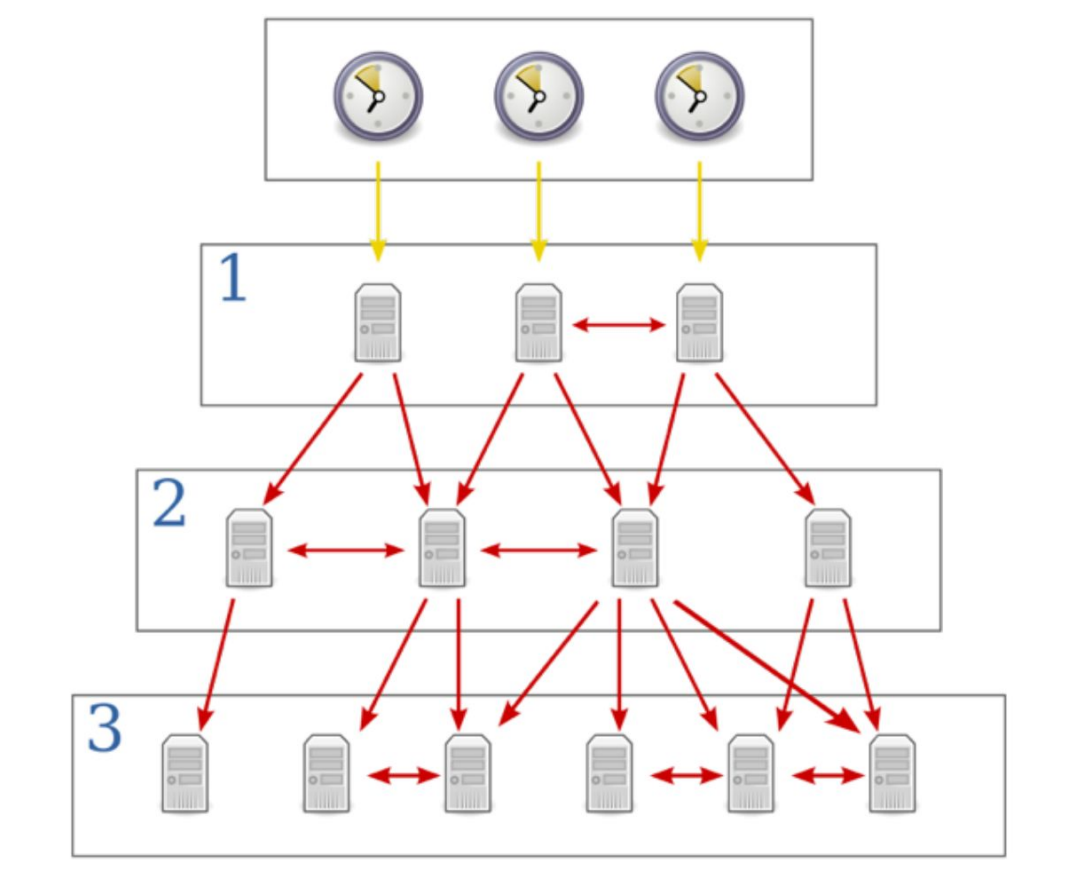
Sincronización de Relojes

Primer Approach: NTP

- Los clientes le solicitan la hora a un servidor de tiempo.
- El ajuste de relojes siempre hacia adelante
- Actúa simétricamente entre servidores, y permite a los mismos coordinarse, ajustando ambos relojes.
- Los servidores se dividen en estratos, según la precisión de su tiempo local. **Un servidor solo ajusta su tiempo si su estrato es más alto que el del servidor consultado**, luego actualizará su estrato a *+1* del estrato del servidor con el cual se actualizó.

Sincronización de Relojes

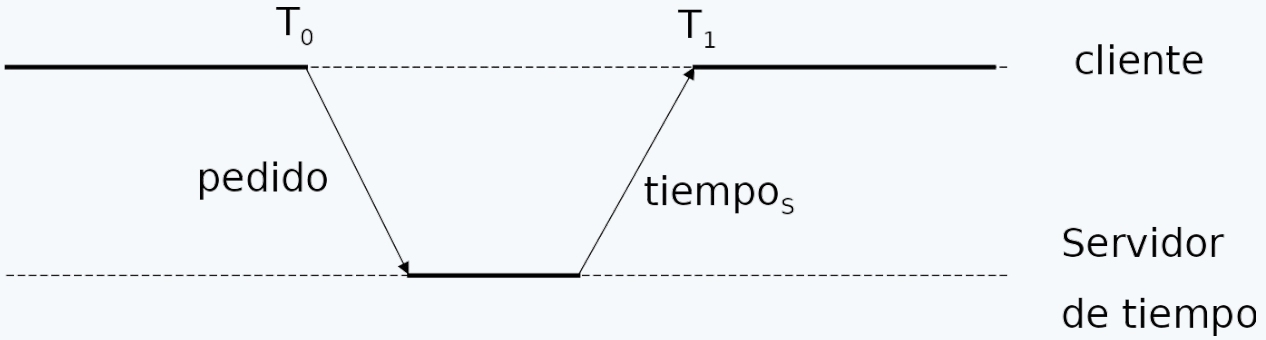
Primer Approach: NTP



Sincronización de Relojes

Primer Approach: NTP

- ¿Problema?
  - Imprecisión producida por los retrasos en la comunicación. La clave está en poder **estimar los retrasos**.



- Algoritmo de Cristian
  - Se hace una estimación del tiempo de propagación como t = (T1-T0)/2
  - Tiempo = tiempoS + t
  - Se puede mejorar si se conoce el tiempo de respuesta del servidor (I), t = (T1-T0-I)/2
  - Se consulta varias veces y se toma aquel tiempo donde t es el menor, descartando aquellos donde t supera un cierto límite.

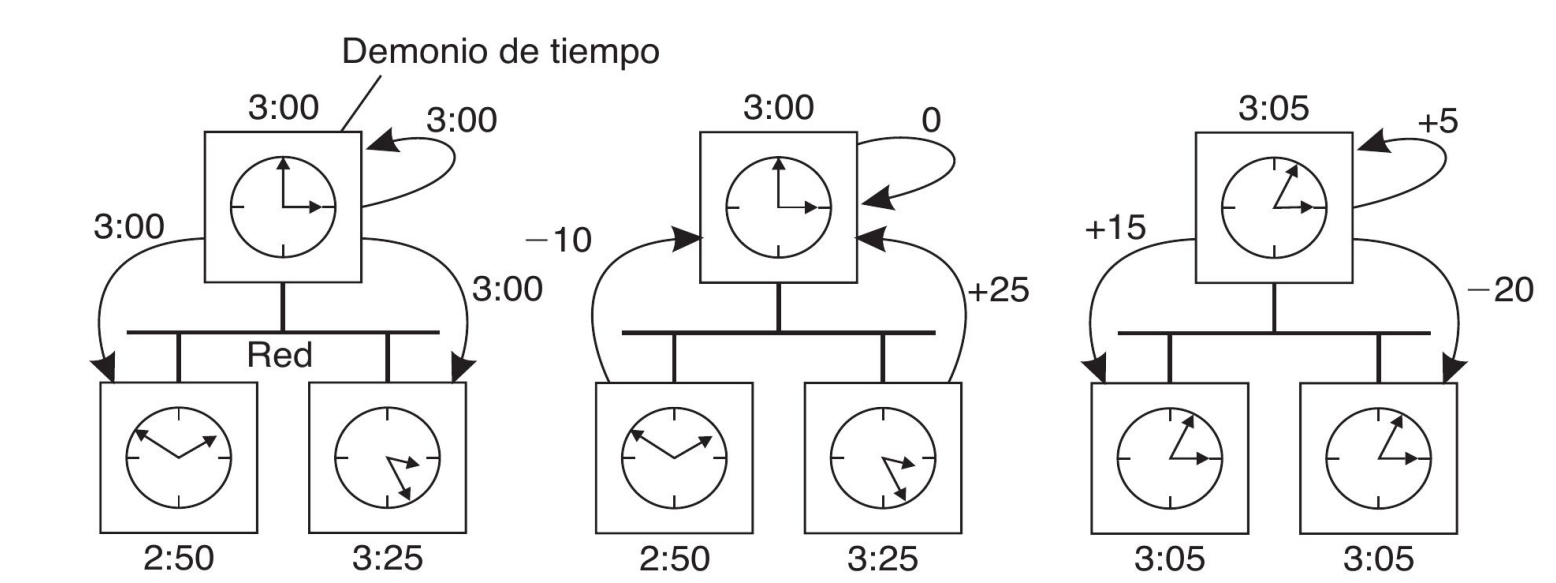
Sincronización de Relojes

Segundo Approach: Algoritmo de Berkeley

- En NTP, el servidor de tiempo es pasivo, pues responde a preguntas que le hacen los clientes. Un enfoque distinto es el dado por los algoritmos de tiempo de Berkeley UNIX de naturaleza activa.
- Pasos:
  - Se elige a un nodo para ser un demonio de tiempo que se encarga de ir cada cierto tiempo entregando su hora a los nodos.
  - A su vez, los nodos le responden con el desfase de su tiempo local respecto de dicho tiempo.
  - Es el demonio quien calcula entonces un valor promedio y dicho valor se elige como el tiempo coordinado.

Sincronización de Relojes

Segundo Approach: Algoritmo de Berkeley



Sincronización de Relojes

Segundo Approach: Algoritmo de Berkeley

- **En este caso, podría darse que el tiempo coordinado no sea el real**, pero en muchas aplicaciones solo basta con que los tiempos estén coordinados y no necesariamente 100% correctos.
- Este algoritmo persigue a la coordinación de tiempos de los nodos, no a la correctitud de los tiempos.

Sincronización de Relojes

- De lo anterior, se deduce que en muchas ocasiones es necesario que las máquinas concuerden en un horario, pero no que ese horario sea el horario real.
  - Denominados **Relojes Lógicos**

"Si dos procesos no interactúan, no es necesario que sus relojes estén sincronizados ya que esa falta de sincronización no es observable. Es irrelevante el tiempo, lo importante es el orden de los eventos."

– Leslie Lamport. "Time, Clocks, and the ordering of events in distributed systems".

Relojes Lógicos de Lamport

Se define la "ocurrencia anterior": **a->b** ("*a antes que b*").

- Todos los procesos coinciden en que ocurre el primer evento a y despues de eso ocurre el evento b.
- Dado por:

  1.- Si a y b son eventos del mismo proceso y a ocurre antes que b, entonces a-> b es verdadero.

  2.- Si a es un evento en el que un proceso envía un mensaje y b es el evento de la recepción del mensaje por algún otro proceso, entonces a-> b es verdadero.

  3.- Si a->b es verdadera y b-> c es verdadera, entonces a->c es verdadera. Relación Transitiva.

Relojes Lógicos de Lamport

Pero ojo!

- Si dos procesos a y b ocurren en distintos procesos que no intercambian mensajes (ni siquiera de manera indirecta a través de terceras partes) entonces, si se cumple tanto:
  - a->b no es verdadera
  - b->a tampoco es verdadera

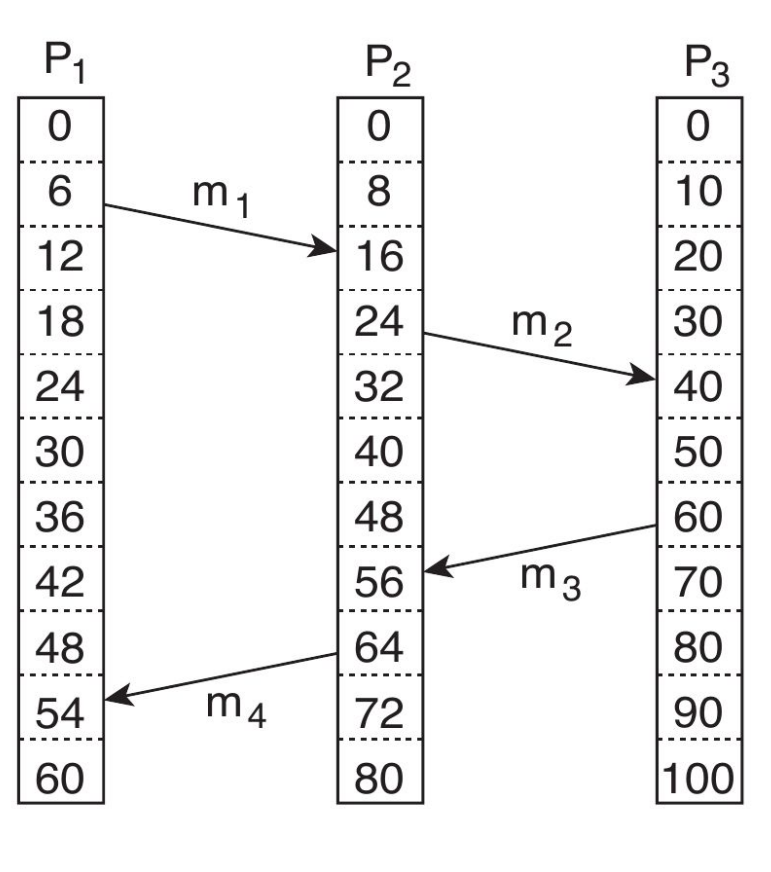
  en este caso se dice que a y b son **concurrentes** (o paralelos), es decir nada se puede decir respecto de cual ocurrió primero.

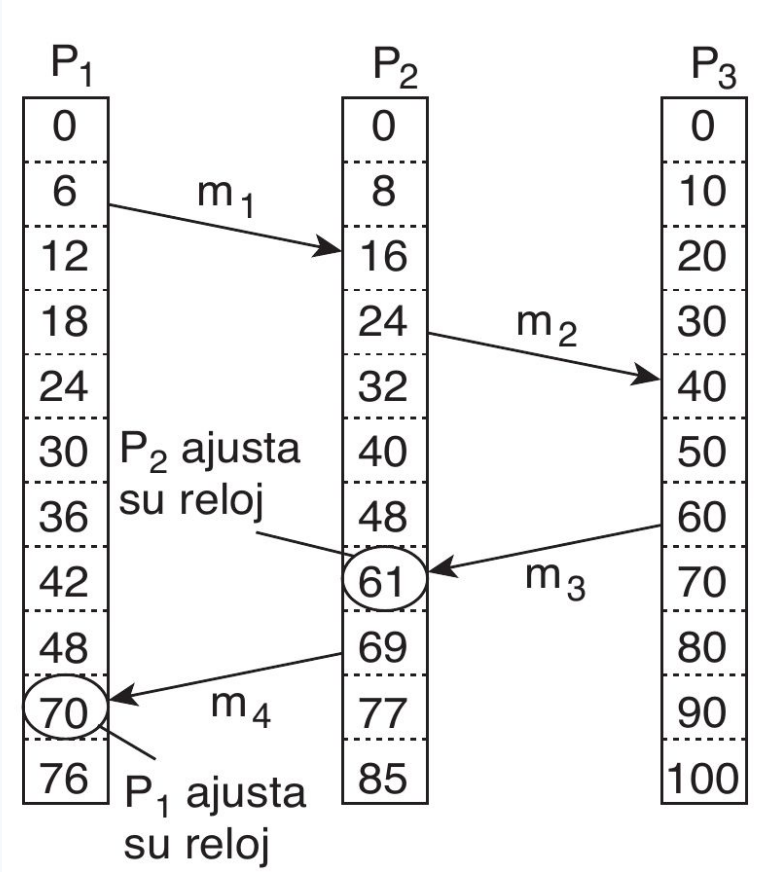
Relojes Lógicos de Lamport

Con el concepto anterior, definimos los Relojes Lógicos (LC)

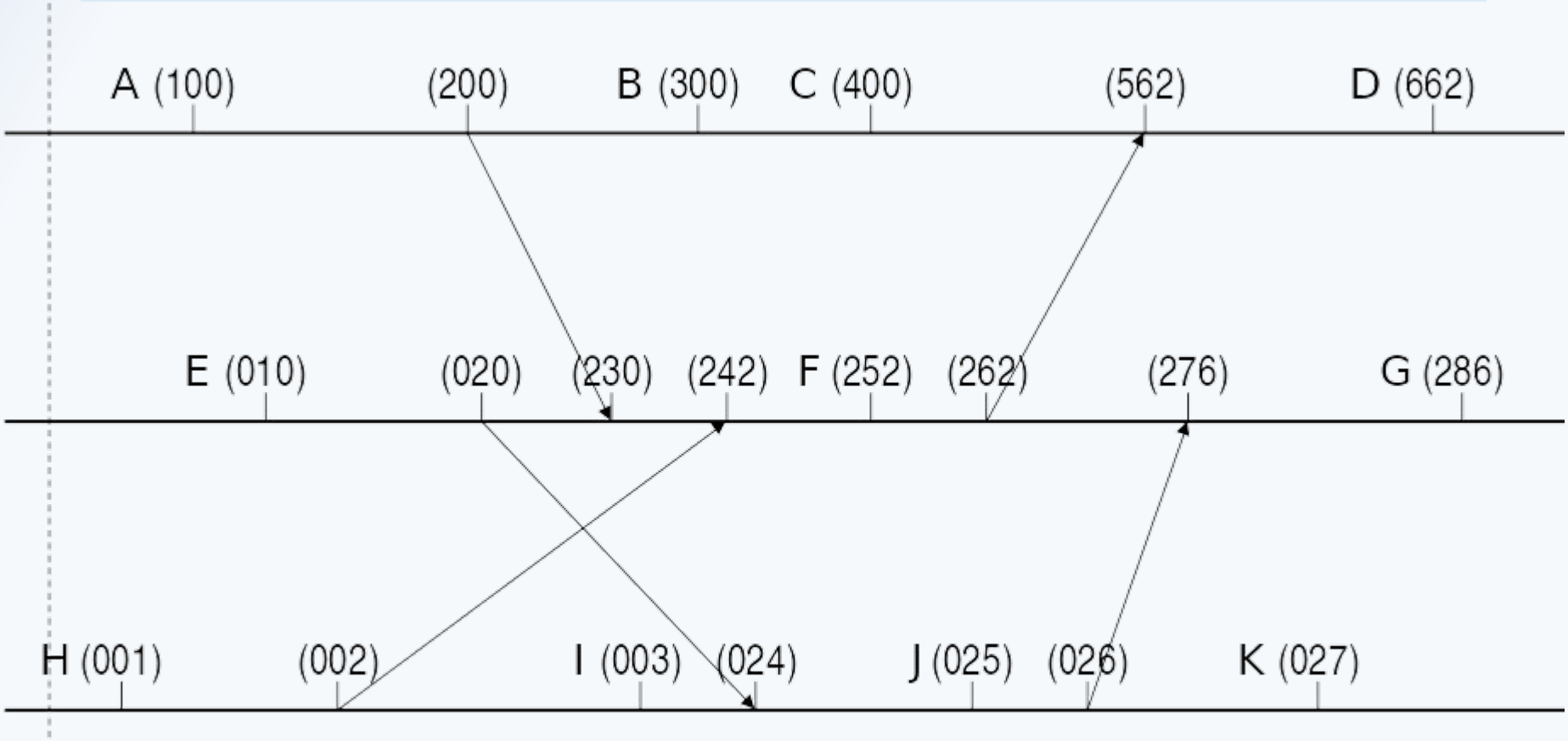
1. Cada proceso Pi mantiene un contador local Ci
2. Antes de ejecutar un evento, Pi ejecuta Ci <- Ci + 1
3. Cuando Pi envía un mensaje m a Pj, se ajusta el registro de tiempo de m para dejarlo igual a Ci , ts(m) <- Ci (solo después del paso anterior)
4. Cuandos se recibe m en Pj, Pj ajusta su contador local Cj <- max(ts(m), Cj)

Relojes Lógicos de Lamport



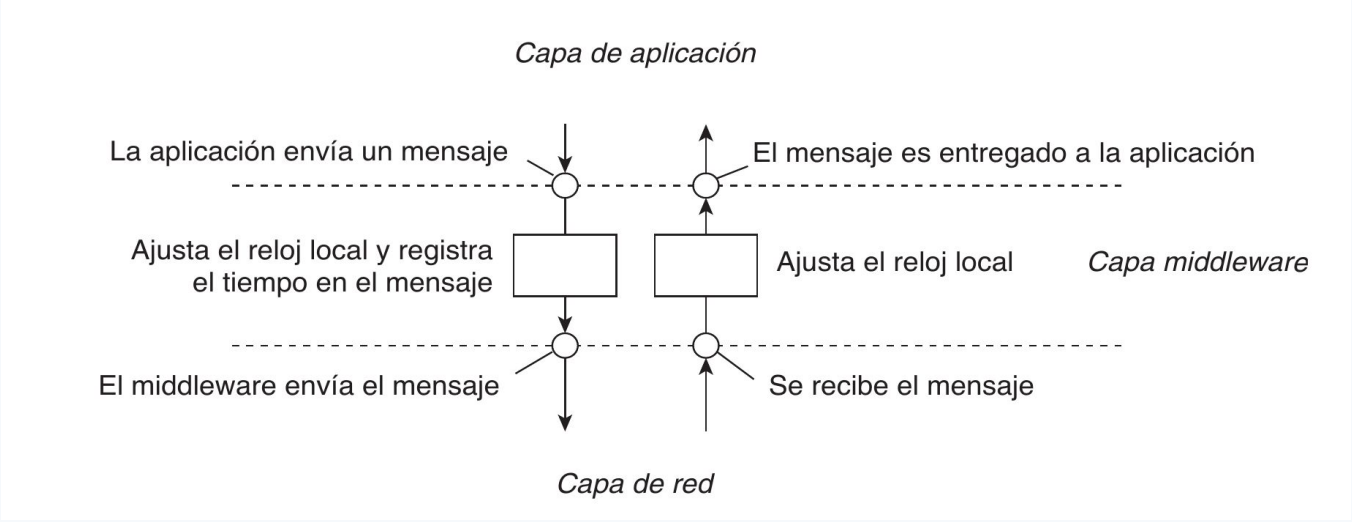


Relojes Lógicos de Lamport



Relojes Lógicos de Lamport

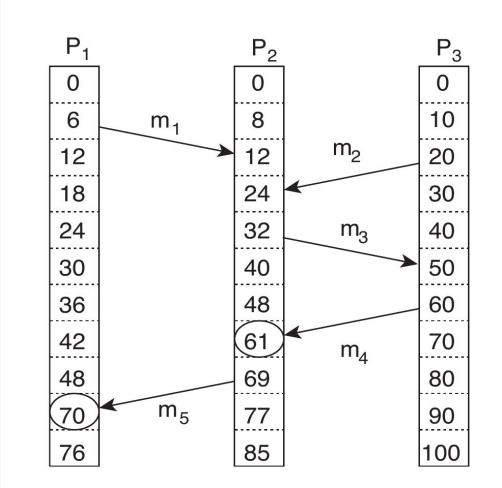
El algoritmo de lamport nos permite tener una implementación distribuida del valor del tiempo global al asignar C(a)<-Ci(a) como el tiempo en que ocurre el evento a, si dicho evento se originó en el proceso i.



Relojes Vectoriales

Lamport nos permite concluir que si a->b, entonces Ca < Cb. Sin embargo lo contrario no es cierto, es decir si Ca < Cb no nos permite concluir que a->b.

Es decir que las marcas de tiempos de dos eventos (Ca,Cb) no nos permite concluir sobre la relación entre ellos.



Relojes Vectoriales

Las marcas de tiempo de Lamport no capturan la causalidad de los eventos. Para ello se usan los vectores de tiempo.

Cada reloj es un arreglo **V** de **N** elementos siendo N el número de procesadores (nodos) del sistema.

1. Inicialmente **Vi[j]=0** para todo i,j
2. Cuando el proceso i genera un evento **Vi[i]=Vi[i]+1**
3. Cuando en el nodo i se recibe un mensaje del nodo j con un vector de tiempo t entonces:
   a. para todo k: **Vi[k]=max(Vi[k],t[k])** (operación de mezcla) y
   b. **Vi[i]=Vi[i] + 1** (operación de recepción)

Relojes Vectoriales