SISTEMAS DISTRIBUIDOS

EJERCICIOS DEL TEMA 5

1. Un reloj C se sincroniza con una fuente externa UTC S una vez por minuto. Si el reloj S de la fuente UTC genera 1000 tics por milisegundo y C genera 990, calcular la máxima desviación entre ambos relojes.

2. Dos ordenadores conectados a una LAN están ejecutando un algoritmo de sincronización interna de sus relojes. La sincronización se realiza por la noche, en un momento en que la carga de la red es muy baja, lo que da una mayor predictibilidad al retraso de transmisión de los mensajes, lo que nos permite asumir que se trata de un sistema síncrono, con retrasos mínimo y máximo de transmisión de $0,0035$ ms y $0,0052$ ms, respectivamente, para el tipo de mensaje considerado.

   Si uno de ellos manda al otro un mensaje con información de su hora local $t = 100$ ms (desde una referencia común, normalmente las 0 horas del 1 de Enero 1970 para los sistemas Unix), indicar cuál será la hora tomada para el ajuste por el otro ordenador al recibir ese mensaje y la desviación máxima respecto al emisor. Obsérvese que tomamos valores muy pequeños para t por simplicidad, normalmente serán valores enteros grandes.

3. Un ordenador P está sincronizando su reloj con una fuente UTC externa S mediante el algoritmo de Cristian. Para ello envía 5 mensajes consecutivos al servidor S. En la tabla siguiente se muestra la información de tiempos de envío y de recepción de los mensajes, junto a la hora incluida dentro de cada mensaje.

T. Envío	T. recepción	Hora
100	100,0045	99,9977
101	101,0049	100,9975
102	102,0048	101,9977
103	103,0047	102,9975
104	104,0044	103,9976

   Sabiendo que el tiempo mínimo para la transmisión de un mensaje entre ambos es $0,0020$ u.t., indicar la hora que tomará P finalmente para su ajuste, así como la desviación máxima de dicho ajuste con respecto a S.

4. Consideremos una aplicación del método simétrico para la sincronización de relojes entre ordenadores, en la cual tenemos los mensajes siguientes, con los datos de tiempo siguientes:

MENSAJE	T. Envío	T. Recepción
$A \rightarrow B$	16:34:13.430	16:34:23.480
$B \rightarrow A$	16:34:25.700	16:34:15.725

1. Un reloj C se sincroniza con una fuente externa UTC S una vez por minuto. Si el reloj S de la fuente UTC genera 1000 tics por milisegundo y C genera 990, calcular la máxima desviación entre ambos relojes.

① desviación de C respecto a S = −10 tics/ms
1 min = 60 s → −60.000 tics → −600 ms
} desviación respecto al reloj perfecto

2. Dos ordenadores conectados a una LAN están ejecutando un algoritmo de sincronización interna de sus relojes. La sincronización se realiza por la noche, en un momento en que la carga de la red es muy baja, lo que da una mayor predictibilidad al retraso de transmisión de los mensajes, lo que nos permite asumir que se trata de un sistema síncrono, con retrasos mínimo y máximo de transmisión de $0,0035$ ms y $0,0052$ ms, respectivamente, para el tipo de mensaje considerado.

   Si uno de ellos manda al otro un mensaje con información de su hora local $t = 100$ ms (desde una referencia común, normalmente las 0 horas del 1 de Enero 1970 para los sistemas Unix), indicar cuál será la hora tomada para el ajuste por el otro ordenador al recibir ese mensaje y la desviación máxima respecto al emisor. Obsérvese que tomamos valores muy pequeños para t por simplicidad, normalmente serán valores enteros grandes.

② Incertidumbre → fórmula

- Sistema síncrono → $u = max - min = 0,0052 - 0,0035 = 0,0017$

$t_{promedio} = \frac{min + max}{2} = \frac{0,0035 + 0,0052}{2} = 0,00435$ ms

$t_{ajustado} = t + T_{trans\,promedio} = 100$ ms $+ 0,00435$ ms $= 100,00435$ ms

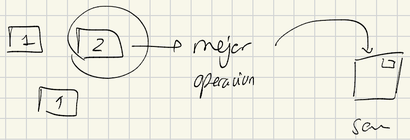
$Desv\,Max = \frac{u}{2} = \frac{0,0017}{2} = 0,00085$ ms

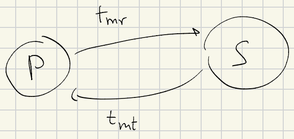
3. Un ordenador P está sincronizando su reloj con una fuente UTC externa S mediante el algoritmo de Cristian. Para ello envía 5 mensajes consecutivos al servidor S. En la tabla siguiente se muestra la información de tiempos de envío y de recepción de los mensajes, junto a la hora incluida dentro de cada mensaje.

T. Envío	T. recepción	Hora
100	100,0045	99,9977
101	101,0049	100,9975
102	102,0048	101,9977
103	103,0047	102,9975
104	104,0044	103,9976

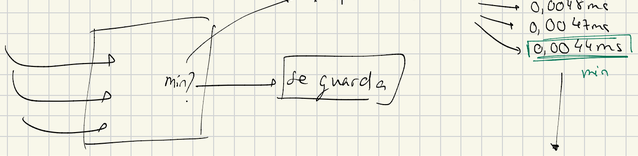
Sabiendo que el tiempo mínimo para la transmisión de un mensaje entre ambos es $0,0020$ u.t., indicar la hora que tomará P finalmente para su ajuste, así como la desviación máxima de dicho ajuste con respecto a S.

ALGORITMO DE CRISTIAN





$T_{rand} = t_{mr} + t_{mt}$



T_i recepción − T_i envío
0,0045 ms
0,0049 ms
0,0048 ms
0,0047 ms
0,0044 ms
mín

$t = \textcircled{t} + \frac{t_{rand}}{2}$
hora del servidor (t la manda)

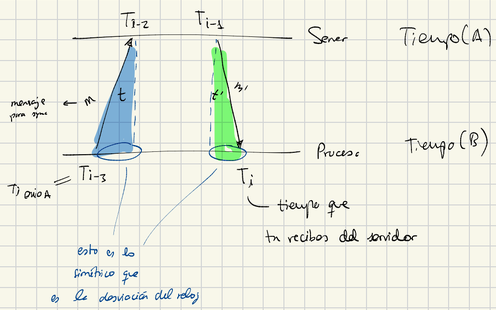
nueva hora $= t + \frac{0,0044}{2} = \underbrace{103,9976}_{\text{hora}} + 0,0022 = 103,9998$ ms

4. Consideremos una aplicación del método simétrico para la sincronización de relojes entre ordenadores, en la cual tenemos los mensajes siguientes, con los datos de tiempo siguientes:

MENSAJE	T. Envío	T. Recepción
$A \to B$	16:34:13.430	16:34:23.480
$B \to A$	16:34:25.700	16:34:15.725

Indicar por qué el tiempo de recepción del mensaje de B a A es menor que el tiempo de envío. Obtener una estimación de la desviación entre ambos relojes, indicando la precisión de dicha estimación. Suponiendo que ambos ordenadores tienen similares características (no hay uno más fiable que el otro en la medida del tiempo), indicar la mejor forma de calcular la nueva hora por parte de A al final de este ciclo de mensajes.

$T_{j-2} = T_{j-3} + t + O$ → discrepancias: $O = C_A - C_B$



esto es lo simétrico que es la desviación del reloj

Método Simétrico

$t = 23.480 - 13.430$

$t' = 15.725 - 25.700$

$d_i = t - t' = 0,075$ s

$O_j = \frac{10.050 + 9.975}{2} = 10.0125$ s

(23.480 − 13.430) ↑ cambio de orden (25.700 − 15.725)

intervalo $[9.975, 10.05]$

Regla de 3 para % error

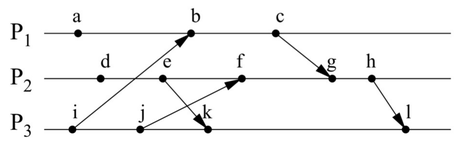
10,0125 — 0,00375

100 — X

$X = \frac{100 \cdot 0{,}00375}{10{,}0125}$ → margen error

y mejor hora

5. Tomando como referencia el gráfico siguiente, que ilustra las acciones llevadas a cabo por 3 procesos, donde los arcos indican la transmisión de mensajes (send-receive). Se pide: anotar los valores de los relojes lógicos de cada uno de ellos sobre dicho diagrama.

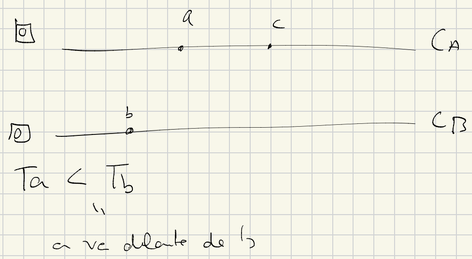


Tiempo lógico y Relojes lógicos

→ contador de eventos que vamos haciendo

Suele caer en examen

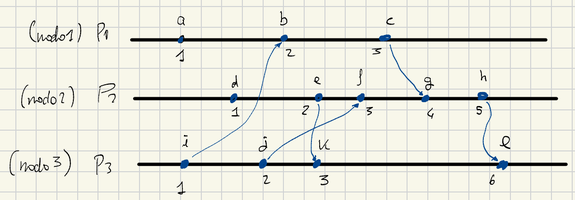
- Preguntar con el chat como sería



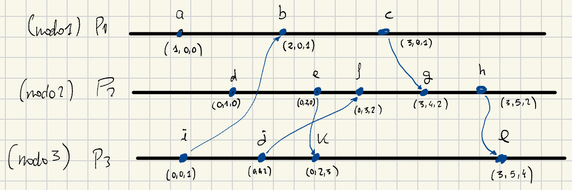
Ta < Tb

a va delante de b

Ordenación Lamport



Cada nodo lleva su orden, excepto si le llega de uno después de actualizar nodo 2.



Básicamente sumas en tu vector y si te llega actualizas tus amigos no sea del vector

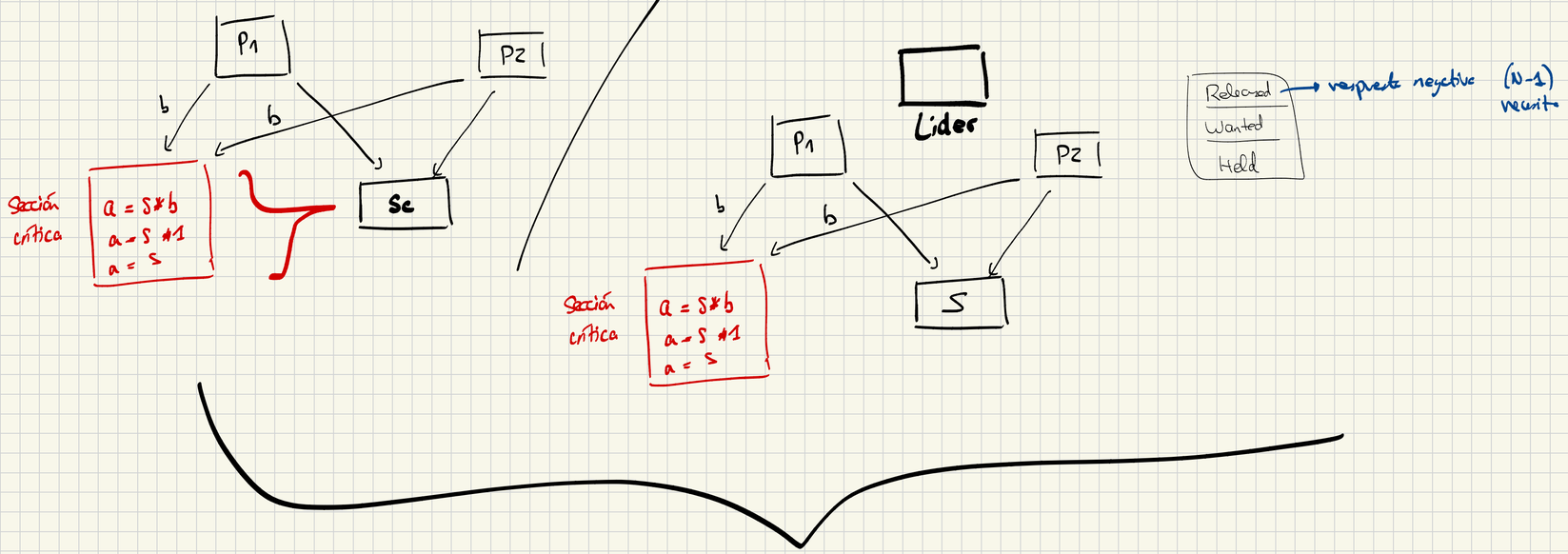
6

Reloj vectorial

9. En el algoritmo de Ricart y Agrawala, si se considera la posibilidad de que los procesos fallen, podemos encontrarnos con la situación siguiente: un proceso P_i pide permiso a P_j enviándole el mensaje de petición (como a los demás), pero P_j ha abortado. ¿ Cómo se interpreta la ausencia de respuesta por P_i?.

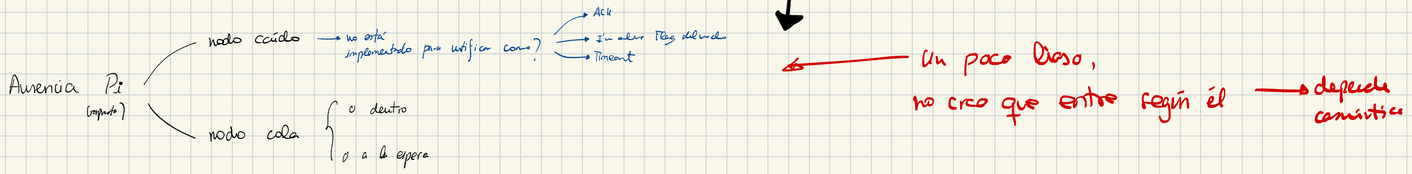
   Indicar una forma de resolverlo.

RICART y GRAWALA

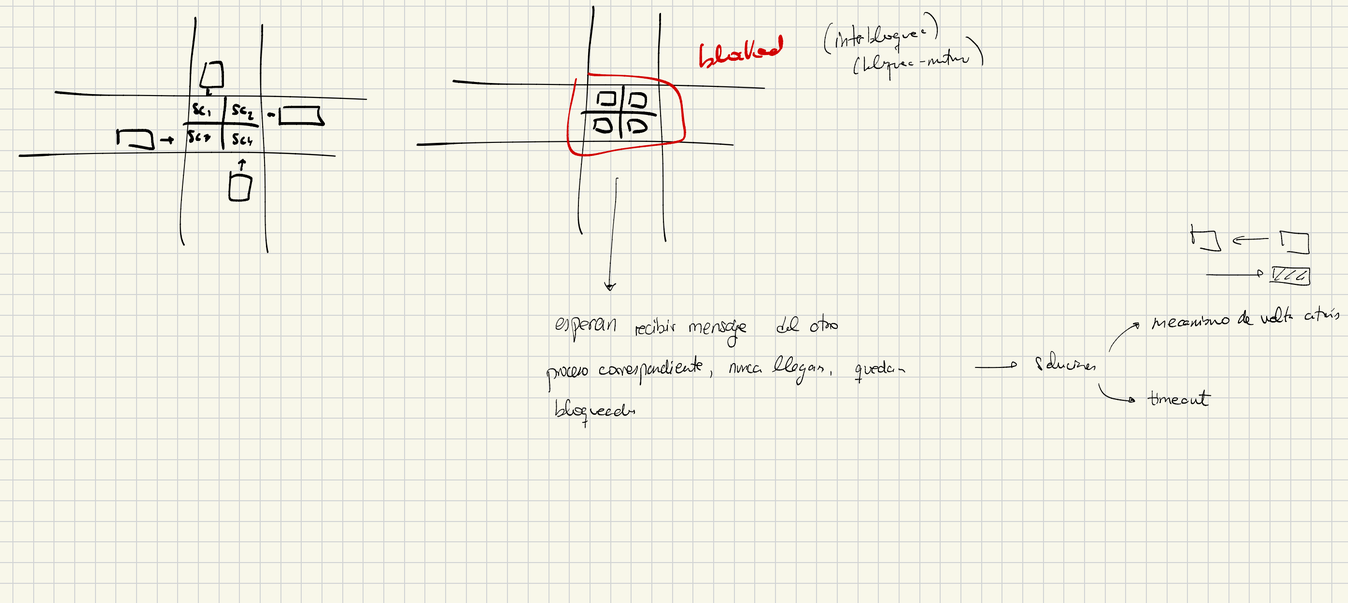


→ respuesta negativa (N-1) recibir

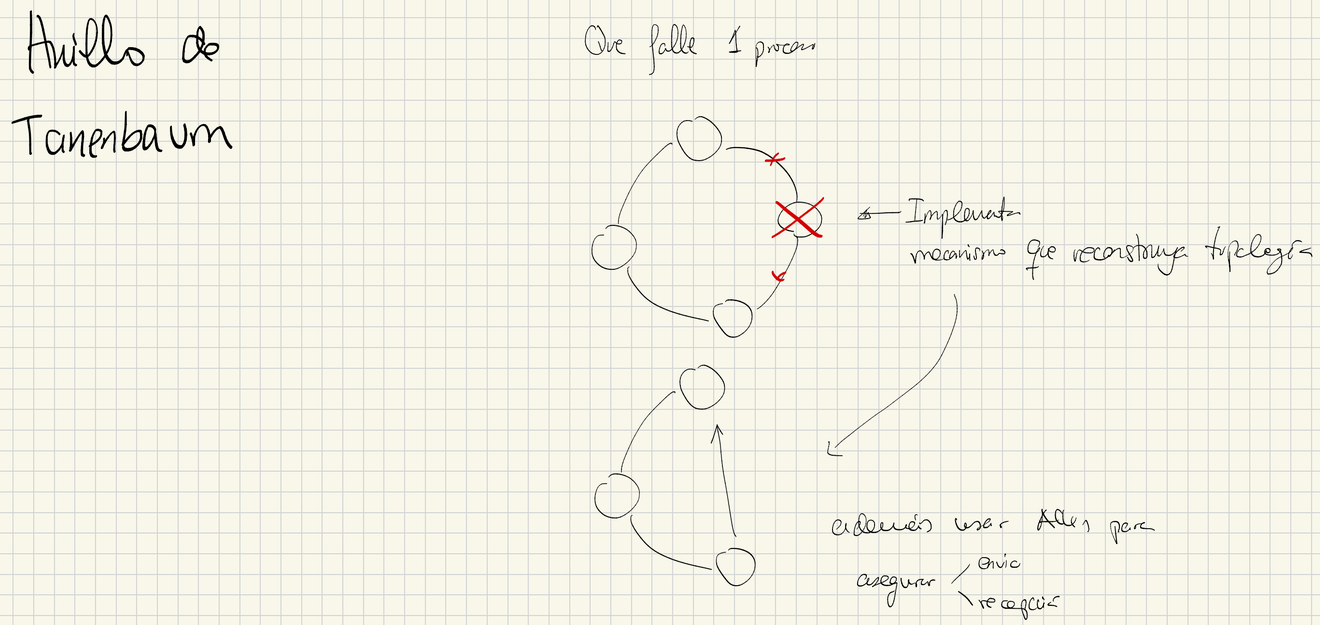
ejercicio examen



10. En un sistema distribuido podemos tener un número arbitrario de recursos independientes. Así, dos procesos P_1, P_2 pueden pretender acceder de forma concurrente a dos recursos A, B. Indicar de qué forma se puede producir un bloqueo mutuo y relacionarlo con el algoritmo de Ricary y Agrawala, indicando lo que ocurre en su ejecución.

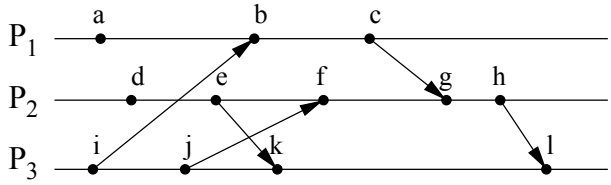


11. En el algoritmo de anillo de Tanenbaum para elecciones se supone que los procesos no fallan. Indicar los cambios necesarios en el caso de que estos puedan fallar.



Indicar por qué el tiempo de recepción del mensaje de B a A es menor que el tiempo de envío. Obtener una estimación de la desviación entre ambos relojes, indicando la precisión de dicha estimación. Suponiendo que ambos ordenadores tienen similares características (no hay uno más fiable que el otro en la medida del tiempo), indicar la mejor forma de calcular la nueva hora por parte de A al final de este ciclo de mensajes.

5. Tomando como referencia el gráfico siguiente, que ilustra las acciones llevadas a cabo por 3 procesos, donde los arcos indican la transmisión de mensajes (send-receive). Se pide: anotar los valores de los relojes lógicos de cada uno de ellos sobre dicho diagrama.

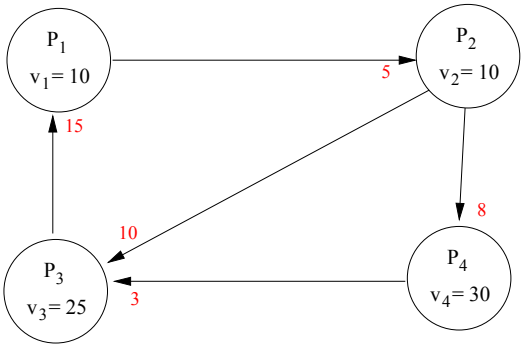


6. Sobre el ejercicio anterior, etiquetar las acciones con los relojes vectoriales correspondientes.

7. Analizar si es posible extender el orden causal definido a través de los relojes vectoriales a un orden total mediante un criterio similar al establecido para extender el orden de Lamport a un orden total, es decir, tomando:

$(e_i, i) \leq (e_j, j)$ si y sólo si $((V(e_i) \leq V(e_j)) \vee ((V(e_i) \not\leq V(e_j) \wedge V(e_j) \not\leq V(e_i) \wedge i < j)$

8. Considerar la red de procesos de la figura siguiente, con los valores iniciales indicados para las variables locales a cada proceso.



t=3	P_1 : $v_1 := 2*v_1$
t=5	P_2 : Inicia algoritmo snapshot.
t=7	P_2 : $v_2 := 5$
t=10	P_3 envia mensaje transfer(5) a P_1, decrementa v_3
t=17	P_1 envia mensaje transfer(7) a P_2, decrementa v_1

Tomar los eventos indicados en el cuadro a la derecha para construir un estado consistente mediante el algoritmo de *snapshot*, que es iniciado por P_2 en el instante 5, según se indica en el cuadro.

9. En el algoritmo de Ricart y Agrawala, si se considera la posibilidad de que los procesos fallen, podemos encontrarnos con la situación siguiente: un proceso P_i pide permiso a P_j enviándole el mensaje de petición (como a los demás), pero P_j ha abortado. ¿ Cómo se interpreta la ausencia de respuesta por P_i?.

Indicar una forma de resolverlo.

10. En un sistema distribuido podemos tener un número arbitrario de recursos independientes. Así, dos procesos P_1, P_2 pueden pretender acceder de forma concurrente a dos recursos A, B. Indicar de qué forma se puede producir un bloqueo mutuo y relacionarlo con el algoritmo de Ricary y Agrawala, indicando lo que ocurre en su ejecución.

11. En el algoritmo de anillo de Tanenbaum para elecciones se supone que los procesos no fallan. Indicar los cambios necesarios en el caso de que estos puedan fallar.

SISTEMAS DISTRIBUIDOS

TEMA 5: SINCRONIZACION EN SISTEMAS DISTRIBUIDOS

Valentín Valero

Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha

memilia.cambronero@uclm.es
vicente.sahori@uclm.es
adrian.bernal@uclm.es
david.munozlopez@uclm.es

Tercer Curso del Grado en Informática

JOSE ANTONIO MATEO

A1. 14

Tutoría
- L 9:40 - 11:30
- X 11:45 - 14:00
- J 10:30 - 13:30

INDICE:

RELOJES FISICOS Y LOGICOS

- Teoría de la relatividad:

  No existe una referencia temporal común

- Sólo podemos tener una sincronización aproximada de los relojes físicos de diferentes nodos.

RELOJES FISICOS Y LOGICOS

- Teoría de la relatividad:

  No existe una referencia temporal común

- Sólo podemos tener una sincronización aproximada de los relojes físicos de diferentes nodos.

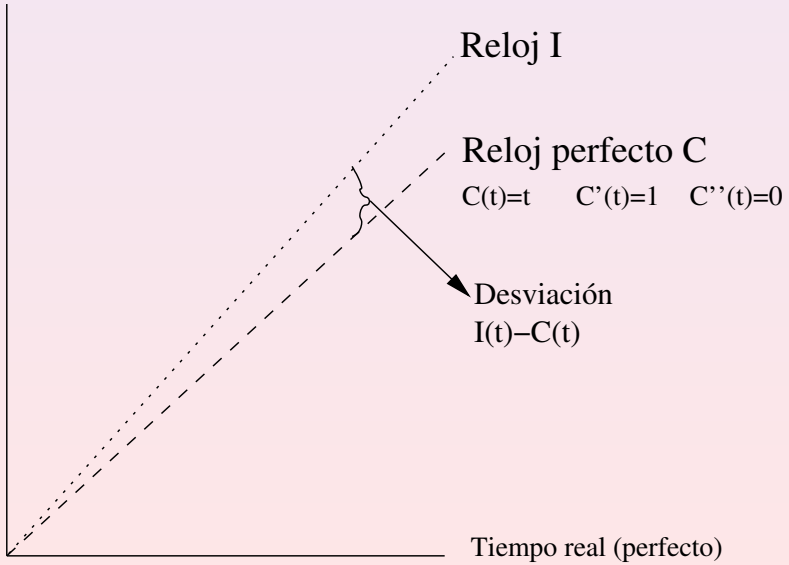
MEDIDA DEL TIEMPO

- Definimos un reloj C como una función $C(t)$ que nos facilita la hora del reloj en el instante real local t.
  Reloj perfecto: $C(t) = t$.
- Se define la *frecuencia* de un reloj C como la tasa a la que avanza: $C'(t)$.
- Se define la *desviación real* de un reloj C como la diferencia $C(t) - t$, y la *desviación entre dos relojes* C_a y C_b como $C_a(t) - C_b(t)$.

MEDIDA DEL TIEMPO

- Se define la *diferencia entre las frecuencias* de dos relojes C_a y C_b como la diferencia: $C'_a(t) - C'_b(t)$. Nos da una medida de cómo se van alejando uno del otro.

  La tasa de deriva de C_a se define tomando $C_b(t) = t$ (reloj perfecto): $C'_a(t) - 1$.



DISPOSITIVOS DE TIEMPO EN UN ORDENADOR

- RTC (Real Time Clock): Independiente de la CPU. Funciona aunque el ordenador esté apagado, con una pequeña batería, que también mantiene la RAM-CMOS. Genera interrupciones periódicas en IRQ-8, con un rango de frecuencias de la interrupción entre 2Hz-8192Hz.

- TSC (Time Stamp Counter): Es un registro de 64 bits que contabiliza el número total de ciclos de la CPU desde el arranque. Se puede leer con la instrucción *rdtsc*. Precisiones de nanosegundos. En multiprocesadores cada CPU tendrá su propio TSC y seguramente tendrán valores ligeramente diferentes.

DISPOSITIVOS DE TIEMPO EN UN ORDENADOR

- PIT (Programmable Interval Timer): Genera interrupciones periódicas en IRQ-0. Baja precisión (frecuencia $\sim$ 1.2 MHz). En Linux se usaba para producir 100 int/seg. Se mantiene por compatibilidad.

- HPET (High Precision Event Timer): Se emplean desde 2005, para aplicaciones con altos requerimientos de tiempo. Frecuencias por encima de 10MHz, que permiten medidas por debajo de microsegundos.

  Se basan en un contador de 64 bits y 256 comparadores, que pueden utilizarse para producir interrupciones en diferentes CPUs.

MEDIDA DEL TIEMPO

Cómo medimos el paso del tiempo?

- Principal fuente históricamente: fenómenos astronómicos (Sol, Luna, estrellas).
- Se define un **día solar** como el tiempo que transcurre entre dos tránsitos consecutivos del Sol por el mismo meridiano.

  Este valor no es constante, por lo que se define un valor promedio: Tiempo Solar Medio (TSM).

  Se define un **segundo** como 1/86400 de un día solar medio.

MEDIDA DEL TIEMPO

Cómo medimos el paso del tiempo?

- Principal fuente históricamente: fenómenos astronómicos (Sol, Luna, estrellas).
- Se define un **día solar** como el tiempo que transcurre entre dos tránsitos consecutivos del Sol por el mismo meridiano.

  Este valor no es constante, por lo que se define un valor promedio: Tiempo Solar Medio (TSM).

  Se define un **segundo** como 1/86400 de un día solar medio.

MEDIDA DEL TIEMPO

- Las observaciones astronómicas tienen baja precisión: afecta la rotación de la Tierra (va disminuyendo la velocidad de rotación por los efectos de las mareas, aunque recientemente se ha observado una cierta aceleración, posiblemente relacionada con cambios atmosféricos y cambio climático) y otros fenómenos.
- Debido a estas discrepancias entre el valor medido y la Astronomía se observó en 1582 una anomalía de 10 días en el equinoccio de primavera. El Papa Gregorio XIII introdujo el calendario *Gregoriano*, que reemplazó al *Juliano*. El nuevo calendario fija el año en 365.2425 días, en lugar de 365.25.

  CURIOSIDAD: Del 4 de Octubre de 1582 se pasó al 15 de Octubre de 1582!

MEDIDA DEL TIEMPO

- Las observaciones astronómicas tienen baja precisión: afecta la rotación de la Tierra (va disminuyendo la velocidad de rotación por los efectos de las mareas, aunque recientemente se ha observado una cierta aceleración, posiblemente relacionada con cambios atmosféricos y cambio climático) y otros fenómenos.
- Debido a estas discrepancias entre el valor medido y la Astronomía se observó en 1582 una anomalía de 10 días en el equinoccio de primavera. El Papa Gregorio XIII introdujo el calendario *Gregoriano*, que reemplazó al *Juliano*. El nuevo calendario fija el año en 365.2425 días, en lugar de 365.25.

  CURIOSIDAD: Del 4 de Octubre de 1582 se pasó al 15 de Octubre de 1582!

MEDIDA DEL TIEMPO

- Mejor precisión: usar la física.

  Se mide la frecuencia de los cambios en los estados de energía en átomos de Cesio (relojes atómicos).

- El primer reloj atómico se construyó en 1955 en UK.

- El tiempo UTC (Tiempo Universal Coordinado) se adoptó en 1963 como referencia estandarizada para la medida del tiempo.

  Se basa en una red de estaciones atómicas, tomando un valor promedio entre ellas.

  Precisión actual: 10^{-15} segundos, pero los usuarios reciben el servicio vía GPS o Internet con (mucha) menor precisión (nanosegundos en el mejor de los casos).

SINCRONIZACIÓN DE RELOJES

- Sincronización de dos relojes: Dada una fuente S de la hora UTC, y un reloj local C, se dice que C **está sincronizado con S** con una precisión D si se cumple:

$$|C(t) - S(t)| \leq D \quad \forall t$$

- La sincronización puede ser **externa** (fuente externa) o **interna** (una red de ordenadores sincroniza sus relojes con cierta precisión - LAN).

SINCRONIZACIÓN DE RELOJES

- Sincronización de dos relojes: Dada una fuente S de la hora UTC, y un reloj local C, se dice que C **está sincronizado con S** con una precisión D si se cumple:

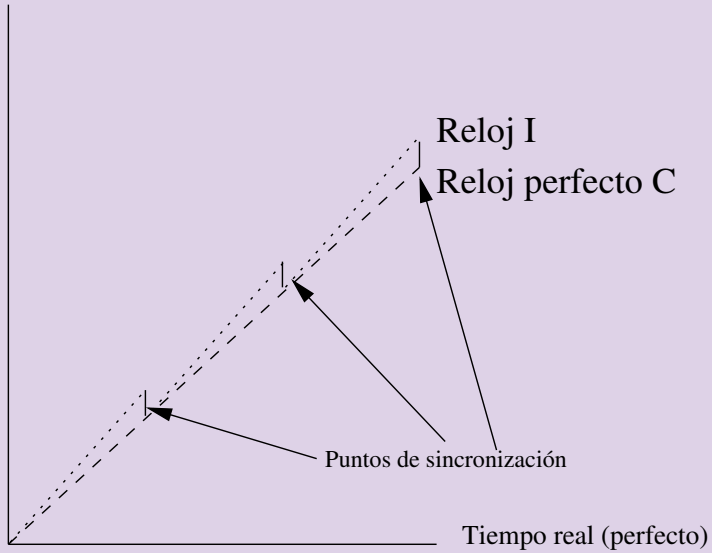
$$|C(t) - S(t)| \leq D \quad \forall t$$

- La sincronización puede ser **externa** (fuente externa) o **interna** (una red de ordenadores sincroniza sus relojes con cierta precisión - LAN).

RELOJES CORRECTOS

Definition (RELOJ CORRECTO)

Un reloj es **correcto** si su tasa de deriva está acotada por un valor conocido. Puede mantenerse el reloj con una desviación acotada mediante sincronizaciones periódicas.



RELOJES CORRECTOS

Período de sincronización:

- Sea ρ la cota conocida de la tasa de deriva de un reloj C ($\approx 10^{-6}$ segundos/segundo):

$$|C'(t) - 1| \leq \rho$$

- Es decir: $(1-\rho) \leq C'(t) \leq (1+\rho)$.
- Por tanto: $(1-\rho) \cdot t \leq C'(t) \cdot t \leq (1+\rho) \cdot t$.
- Tomando $C(t) = C'(t) \cdot t$ en primera aproximación, obtenemos $|C(t) - t| \leq \rho \cdot t$.
- Sea D: desviación máxima permitida: $\rho \cdot t \leq D \Leftrightarrow t \leq \frac{D}{\rho}$
- De ahí, si $t \leq D/\rho$ obtendremos: $|C(t) - t| \leq D$.

RELOJES CORRECTOS

Período de sincronización:

- Tomando entonces D/ρ como período de sincronización tendríamos acotada la desviación en D unidades de tiempo.
- Ejemplo: para un valor normal de la cota, como $\rho = 10^{-6}$ (desviación de 1 microsegundo por segundo), para una desviación máxima de 1 segundo necesitaríamos hacer sincronizaciones con una fuente externa cada 10^6 segundos: 11,5 días.

SINCRONIZACION DE DOS RELOJES IGUALES

- En la práctica tendremos que sincronizar dos relojes no perfectos.
- Supongamos dos relojes iguales con cota de error ρ.
- Tenemos:

$$(1-\rho)t \leq C_a(t) \leq (1+\rho)t$$
$$-(1+\rho)t \leq -C_b(t) \leq -(1-\rho)t$$

- De ahí: $-2\rho t \leq C_a(t) - C_b(t) \leq 2\rho t$, es decir, para una desviación máxima D el periodo de sincronización es $D/(2 \cdot \rho)$.

ERROR EN LA MEDIDA DEL TIEMPO

Dado un reloj C, con una cota de error ρ, el error cometido al medir el intervalo de tiempo entre dos eventos que se producen en los instantes de tiempo real t y t' está acotado por:

$$(1-\rho)(t'-t) \leq C(t') - C(t) \leq (1+\rho)(t'-t)$$

Como los tiempos reales no son conocidos, usamos los tiempos UTC en su lugar.

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Sincronización interna: redes locales normalmente, sin soporte externo de fuente de la hora. Se persigue que todos tengan la misma hora aproximadamente.

- En sistemas síncronos son conocidos:
  - Cotas de los errores de los relojes.
  - Retardos mínimos y máximos en la transmisión de mensajes.
  - Tiempo para ejecutar cada paso de un proceso.
- Un proceso P envía a otro Q un mensaje con su hora local t.
- Q tendría que actualizar su reloj local a $t + T_{trans}$.
  T_{trans} es el tiempo de transmisión del mensaje (desconocido).

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Sincronización interna: redes locales normalmente, sin soporte externo de fuente de la hora. Se persigue que todos tengan la misma hora aproximadamente.

- En sistemas síncronos son conocidos:
  - Cotas de los errores de los relojes.
  - Retardos mínimos y máximos en la transmisión de mensajes.
  - Tiempo para ejecutar cada paso de un proceso.
- Un proceso P envía a otro Q un mensaje con su hora local t.
- Q tendría que actualizar su reloj local a $t + T_{trans}$.
  T_{trans} es el tiempo de transmisión del mensaje (desconocido).

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Sincronización interna: redes locales normalmente, sin soporte externo de fuente de la hora. Se persigue que todos tengan la misma hora aproximadamente.

- En sistemas síncronos son conocidos:
  - Cotas de los errores de los relojes.
  - Retardos mínimos y máximos en la transmisión de mensajes.
  - Tiempo para ejecutar cada paso de un proceso.
- Un proceso P envía a otro Q un mensaje con su hora local t.
- Q tendría que actualizar su reloj local a $t + T_{trans}$.

  T_{trans} es el tiempo de transmisión del mensaje (desconocido).

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Cálculo de la desviación o error cometido:

- Sistema síncrono $\Rightarrow$ $min \leq T_{trans} \leq max$
- Se define la incertidumbre en el tiempo de transmisión del mensaje como $u = max - min$.
- La asignación de Q a los valores $(t + min)$ ó $(t + max)$ tendría un error (desviación respecto a P) igual a u.
- En cambio, tomando $t + \frac{min + max}{2}$ la desviación es como mucho $u/2$.
- Para N relojes, la desviación máxima será: $u(1 - \frac{1}{N})$.

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Cálculo de la desviación o error cometido:

- Sistema síncrono $\Rightarrow$ $min \leq T_{trans} \leq max$
- Se define la incertidumbre en el tiempo de transmisión del mensaje como $u = max - min$.
- La asignación de Q a los valores $(t + min)$ ó $(t + max)$ tendría un error (desviación respecto a P) igual a u.
- En cambio, tomando $t + \frac{min + max}{2}$ la desviación es como mucho $u/2$.
- Para N relojes, la desviación máxima será: $u(1 - \frac{1}{N})$.

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Cálculo de la desviación o error cometido:

- Sistema síncrono $\Rightarrow$ $min \leq T_{trans} \leq max$
- Se define la incertidumbre en el tiempo de transmisión del mensaje como $u = max - min$.
- La asignación de Q a los valores $(t + min)$ ó $(t + max)$ tendría un error (desviación respecto a P) igual a u.
- En cambio, tomando $t + \frac{min + max}{2}$ la desviación es como mucho $u/2$.
- Para N relojes, la desviación máxima será: $u(1 - \frac{1}{N})$.

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Cálculo de la desviación o error cometido:

- Sistema síncrono $\Rightarrow$ $min \leq T_{trans} \leq max$
- Se define la incertidumbre en el tiempo de transmisión del mensaje como $u = max - min$.
- La asignación de Q a los valores $(t + min)$ ó $(t + max)$ tendría un error (desviación respecto a P) igual a u.
- En cambio, tomando $t + \frac{min + max}{2}$ la desviación es como mucho $u/2$.
- Para N relojes, la desviación máxima será: $u(1 - \frac{1}{N})$.

SINCRONIZACION INTERNA EN SISTEMAS SINCRONOS

Cálculo de la desviación o error cometido:

- Sistema síncrono $\Rightarrow$ $min \leq T_{trans} \leq max$
- Se define la incertidumbre en el tiempo de transmisión del mensaje como $u = max - min$.
- La asignación de Q a los valores $(t + min)$ ó $(t + max)$ tendría un error (desviación respecto a P) igual a u.
- En cambio, tomando $t + \frac{min + max}{2}$ la desviación es como mucho $u/2$.
- Para N relojes, la desviación máxima será: $u(1 - \frac{1}{N})$.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

- La mayoría de los sistemas distribuidos son asíncronos: no conocemos esas cotas en los tiempos de transmisión de los mensajes o son muy malas.
- El algoritmo de Berkeley [Gusella y Zatti (1989)] permite sincronizar un conjunto de ordenadores ejecutando Unix BSD. Es un algoritmo distribuido.
- Uno de ellos actúa como *maestro*, y el resto como *esclavos*.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

- La mayoría de los sistemas distribuidos son asíncronos: no conocemos esas cotas en los tiempos de transmisión de los mensajes o son muy malas.
- El algoritmo de Berkeley [Gusella y Zatti (1989)] permite sincronizar un conjunto de ordenadores ejecutando Unix BSD. Es un algoritmo distribuido.
- Uno de ellos actúa como *maestro*, y el resto como *esclavos*.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

- El maestro periódicamente solicita por multienvío a los esclavos que le manden su hora local.
- La nueva hora local en el nodo maestro se computa usando estos valores y los tiempos circulares de estos mensajes.

  Los valores demasiado alejados de la media se descartan. También puede considerarse el reloj del nodo maestro para computar la media.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

Cálculo de la nueva hora local en nodo maestro (sin usar su reloj):

t_0: instante en que el maestro multienvía su petición.

t_i: Hora enviada por nodo esclavo i.

D_i: Tiempo circular para i.

$$T(t_0) = \frac{\sum_{i=1}^{N}(t_i - \frac{D_i}{2})}{N}$$

Nueva hora para instante t_0 en maestro.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

- Ajuste en nodo maestro: $T(t_0) - t_0$.
- Ajuste para nodo i : $T(t_0) - (t_i - \frac{D_i}{2})$.
- Envía a cada esclavo el ajuste (cantidad de tics) de su reloj local.
- El envío de la discrepancia tiene la ventaja de que el ajuste ya no depende del retraso de transmisión.

SINCRONIZACION INTERNA EN SISTEMAS ASINCRONOS

Experimentación ▸ Gusella & Zatti

- Se usaron 15 ordenadores para sincronizar sus relojes con una precisión de 20-25 ms.
- Los tiempos circulares fueron menores de 10 ms.
- Las tasas de deriva locales fueron menores que 2×10^{-5}.

SINCRONIZACION EXTERNA EN SISTEMAS ASINCRONOS

▸ Algoritmo de Cristian (1989):

- Algoritmo probabilístico: se repite varias veces la operación y se toma la mejor solución obtenida.
- Usamos un servidor de hora UTC externo S (fuente).
- El proceso (nodo) P envía un mensaje de petición a S: mensaje m_r.
- S envía un mensaje m_t a P con su hora local t.

ALGORITMO DE CRISTIAN

- Sea T_{round} el tiempo transcurrido desde enviar m_r hasta recibir m_t ($\approx 1-10$ ms). Se le conoce como *tiempo circular*.
- En una LAN los tiempos circulares son del orden de $1-10ms$, y las tasas de deriva son pequeñas ($\approx 10^{-6}$): el error cometido por las derivas en 10 ms sería despreciable ($\approx 10^{-5}$ ms).
- Esta operación se repite varias veces (espaciando las peticiones), y se toma el valor T_{round} más pequeño.
- P asigna su nueva hora local a $t + \frac{T_{round}}{2}$.

ALGORITMO DE CRISTIAN

Supongamos conocido un valor mínimo *min* para el tiempo de transmisión de un mensaje:

- S asigna m_t como muy pronto min u.t. después del envío de m_r por P.
- Y como muy tarde min u.t. antes de la llegada de m_t a P.
- Por tanto, la hora de S cuando llega m_t está en el intervalo:

$$[t + min, t + T_{round} - min]$$

- Anchura del intervalo: $T_{round} - 2.min$.

Precisión media: $\pm(\frac{T_{round}}{2} - min)$.

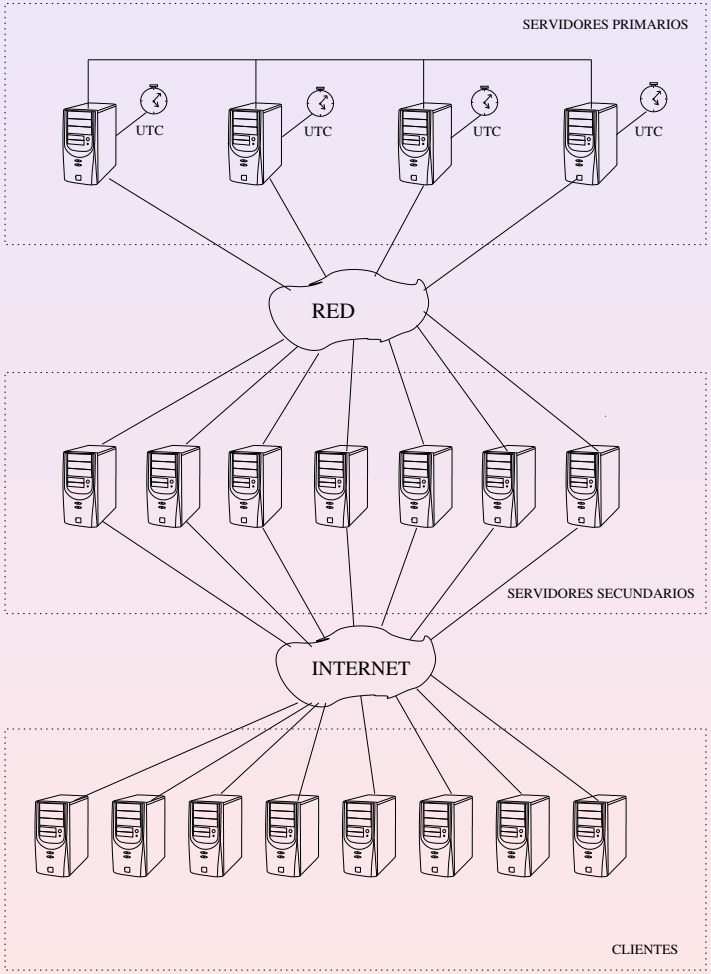
PROTOCOLO NTP (Network Time Protocol)

▸ NTP [1985, David L. Mills]

Define una arquitectura para un servicio de tiempo y un protocolo para distribuir la información de tiempo en Internet, con diferentes niveles de precisión.

- Servicio robusto de fuente UTC mediante servidores replicados, para garantizar la disponibilidad del servicio.
- UDP, puerto 123.
- Servicio jerarquizado:
  - Servidores primarios: conectados directamente a una fuente UTC.
  - Servidores secundarios: obtienen la hora desde los primarios. Dan un servicio de menor precisión.
  - Clientes, que obtienen la hora a partir de los servidores secundarios normalmente.

ARQUITECTURA DE NTP



PROTOCOLO NTP

Tres métodos de sincronización:

1. Multicast (servidor central).
   - Usado en redes LAN de alta velocidad.
   - El servidor envía la hora periódicamente al resto de nodos.
   - Los relojes locales de los nodos se actualizan con la hora recibida, asumiendo un pequeño retraso de transmisión para el ajuste.
   - Baja precisión.

PROTOCOLO NTP

Tres métodos de sincronización:

2. Llamada de procedimiento remoto (RPC):
   - Se usa el algoritmo de Cristian.
   - Mejor precisión que el simple multienvío.
   - Uso en LANs o redes LAN cercanas.

PROTOCOLO NTP

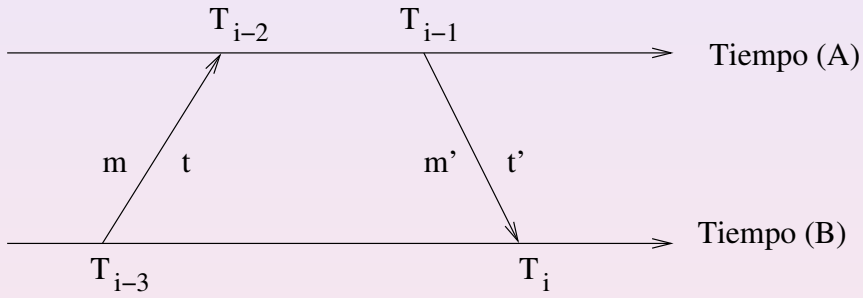
Tres métodos de sincronización:

3 Modo simétrico:

- Alta precisión: usado por los servidores primarios para sincronizar sus relojes, o entre varios servidores en una LAN.
- Los servidores intercambian mensajes con información de tiempos anterior.

MÉTODO SIMÉTRICO

INTERCAMBIO DE MENSAJES:

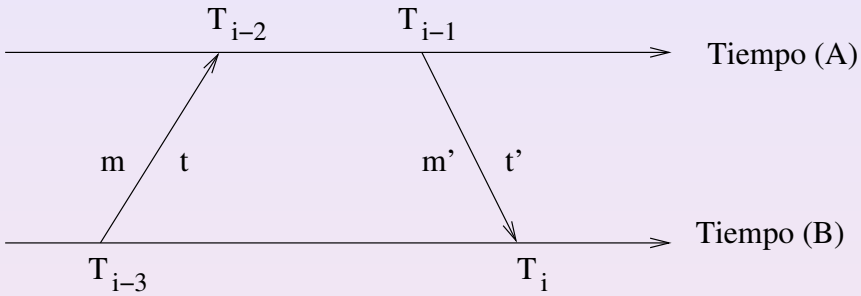


- El mensaje m' de A a B tiene los contenidos siguientes:
  - T_{i-3}: Hora local en B al enviar m.
  - T_{i-2}: Hora local en A cuando llega m.
  - T_{i-1}: Hora local en A cuando se envía m'.

METODO SIMETRICO

- Se usa UDP para reducir los tiempos de transmisión de los mensajes.
- No hay problema si se pierden mensajes, y los tiempos entre transmisiones consecutivas no son relevantes.
- Se guardan los mensajes anteriores.

METODO SIMETRICO

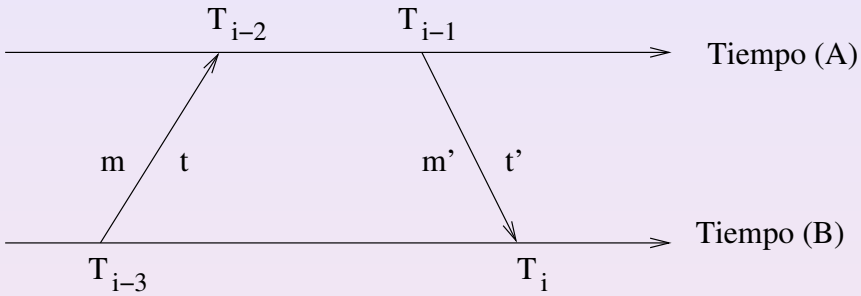


- Sean t y t' los tiempos reales de transmisión de los mensajes.
- Sea o la discrepancia entre ambos relojes (desconocida): $o = C_A - C_B$.
- Entonces: $T_{i-2} = T_{i-3} + t + o$

  $T_i = T_{i-1} + t' - o$
- De ahí, el tiempo circular:

  $d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$.

METODO SIMETRICO

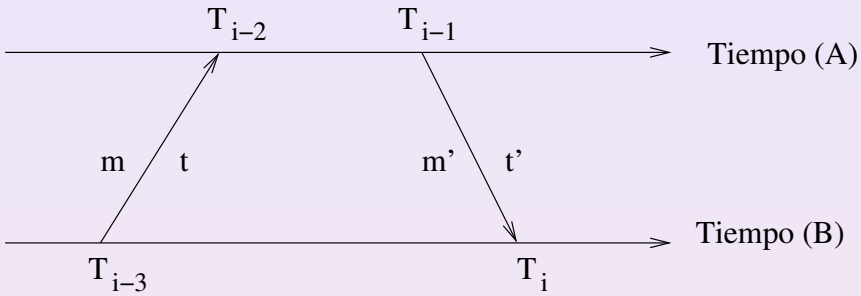


- Sean t y t' los tiempos reales de transmisión de los mensajes.
- Sea o la discrepancia entre ambos relojes (desconocida): $o = C_A - C_B$.
- Entonces: $T_{i-2} = T_{i-3} + t + o$

  $T_i = T_{i-1} + t' - o$
- De ahí, el tiempo circular:

  $d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$.

METODO SIMETRICO

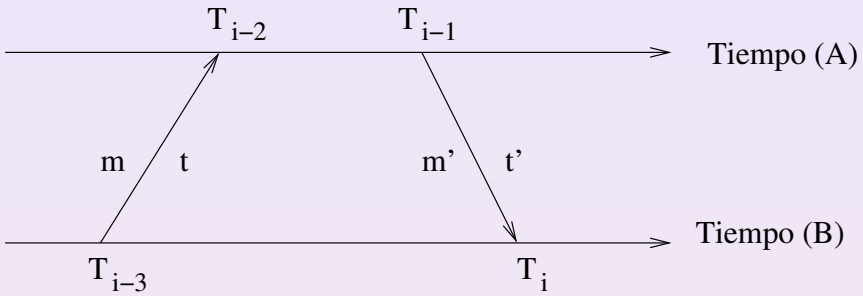


- Sean t y t' los tiempos reales de transmisión de los mensajes.
- Sea o la discrepancia entre ambos relojes (desconocida): $o = C_A - C_B$.
- Entonces: $T_{i-2} = T_{i-3} + t + o$

  $T_i = T_{i-1} + t' - o$
- De ahí, el tiempo circular:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}.$$

METODO SIMETRICO



- Sean t y t' los tiempos reales de transmisión de los mensajes.
- Sea o la discrepancia entre ambos relojes (desconocida): $o = C_A - C_B$.
- Entonces: $T_{i-2} = T_{i-3} + t + o$

  $T_i = T_{i-1} + t' - o$
- De ahí, el tiempo circular:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}.$$

METODO SIMETRICO

- Se obtiene de cada par de mensajes una estimación de la desviación entre ambos relojes o_i:

$$o_i = \frac{T_{i-2} - T_{i-3} + T_{i-1} - T_i}{2}$$

- De donde se obtiene:

$$o = o_i + \frac{t' - t}{2}$$

- De ahí:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

METODO SIMETRICO

- Se obtiene de cada par de mensajes una estimación de la desviación entre ambos relojes o_i:

$$o_i = \frac{T_{i-2} - T_{i-3} + T_{i-1} - T_i}{2}$$

- De donde se obtiene:

$$o = o_i + \frac{t' - t}{2}$$

- De ahí:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

METODO SIMETRICO

- Se obtiene de cada par de mensajes una estimación de la desviación entre ambos relojes o_i:

$$o_i = \frac{T_{i-2} - T_{i-3} + T_{i-1} - T_i}{2}$$

- De donde se obtiene:

$$o = o_i + \frac{t' - t}{2}$$

- De ahí:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

METODO SIMETRICO

- De:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

- o_i es una estimación de o con precisión d_i.
- Se repite hasta 8 veces, y se escoge el valor o_i cuyo d_i es menor.

METODO SIMETRICO

- De:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

- o_i es una estimación de o con precisión d_i.
- Se repite hasta 8 veces, y se escoge el valor o_i cuyo d_i es menor.

METODO SIMETRICO

- De:

$$o_i - \frac{d_i}{2} \leq o \leq o_i + \frac{d_i}{2}$$

- o_i es una estimación de o con precisión d_i.
- Se repite hasta 8 veces, y se escoge el valor o_i cuyo d_i es menor.

MÉTODO SIMÉTRICO

- Cuando ambos relojes son idénticos, se puede repartir a partes iguales la desviación a cada uno para el ajuste ($d_i/2$).
- Se pueden considerar fuentes diferentes, y aquellas de menor calidad (en comparación con las otras) pueden descartarse (dispersión de filtros).

METODO SIMETRICO

NTP aplica varias técnicas para mejorar la calidad del servicio:

- Se tiene en cuenta el tiempo de procesado de los mensajes.
- Filtrado de datos, eliminando aquellos para los que los d_i se alejen de valores promedio.
- Si se pierde un mensaje no pasa nada, no depende del tiempo entre m y m'.
- Se toman los últimos 8 valores recogidos, y de ellos áquel con d_i más bajo.

TIEMPO LOGICO Y RELOJES LOGICOS

- Sistemas centralizados: ordenación por tiempos físicos.
- Sistemas distribuidos: Al no disponer de una referencia temporal común no podemos realizar una ordenación por tiempos físicos.
- Sin embargo: no necesitamos saber la hora exacta a la que ocurre cada evento, basta que las acciones en diferentes nodos se hagan en el orden lógico correcto.
- Se define un orden lógico para relacionar los eventos en un sistema distribuido:

  Debe cumplir dos propiedades intuitivas: debe respetar el orden local, y en una pareja send-receive, el send precede al receive.

TIEMPO LOGICO Y RELOJES LOGICOS

- Sistemas centralizados: ordenación por tiempos físicos.
- Sistemas distribuidos: Al no disponer de una referencia temporal común no podemos realizar una ordenación por tiempos físicos.
- Sin embargo: no necesitamos saber la hora exacta a la que ocurre cada evento, basta que las acciones en diferentes nodos se hagan en el orden lógico correcto.
- Se define un orden lógico para relacionar los eventos en un sistema distribuido:

  Debe cumplir dos propiedades intuitivas: debe respetar el orden local, y en una pareja send-receive, el send precede al receive.

TIEMPO LOGICO Y RELOJES LOGICOS

- Sistemas centralizados: ordenación por tiempos físicos.
- Sistemas distribuidos: Al no disponer de una referencia temporal común no podemos realizar una ordenación por tiempos físicos.
- Sin embargo: no necesitamos saber la hora exacta a la que ocurre cada evento, basta que las acciones en diferentes nodos se hagan en el orden lógico correcto.
- Se define un orden lógico para relacionar los eventos en un sistema distribuido:

  Debe cumplir dos propiedades intuitivas: debe respetar el orden local, y en una pareja send-receive, el send precede al receive.

TIEMPO LOGICO Y RELOJES LOGICOS

- Sistemas centralizados: ordenación por tiempos físicos.
- Sistemas distribuidos: Al no disponer de una referencia temporal común no podemos realizar una ordenación por tiempos físicos.
- Sin embargo: no necesitamos saber la hora exacta a la que ocurre cada evento, basta que las acciones en diferentes nodos se hagan en el orden lógico correcto.
- Se define un orden lógico para relacionar los eventos en un sistema distribuido:

  Debe cumplir dos propiedades intuitivas: debe respetar el orden local, y en una pareja send-receive, el send precede al receive.

ORDEN PARCIAL DE LAMPORT (1978)

- Se define una relación $\rightarrow$. Es un orden parcial: (reflexiva, antisimétrica y transitiva): **ocurre antes que**.
- Si e y e' son eventos locales en el nodo (proceso) P, y e ocurre antes que e', entonces $e \rightarrow e'$.
- Si $e = send(m)$ (en P) y $e' = receive(m)$ (en P'), entonces $e \rightarrow e'$.

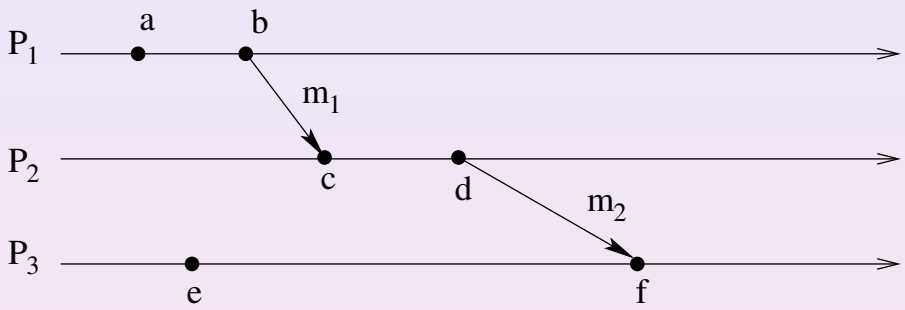
ORDEN PARCIAL DE LAMPORT (1978)

- Se define una relación $\rightarrow$. Es un orden parcial: (reflexiva, antisimétrica y transitiva): **ocurre antes que**.
- Si e y e' son eventos locales en el nodo (proceso) P, y e ocurre antes que e', entonces $e \rightarrow e'$.
- Si $e = send(m)$ (en P) y $e' = receive(m)$ (en P'), entonces $e \rightarrow e'$.

ORDEN PARCIAL DE LAMPORT (1978)

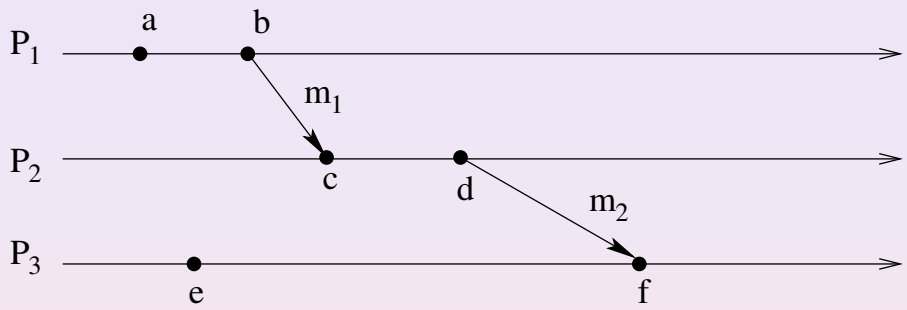
- Se define una relación $\rightarrow$. Es un orden parcial: (reflexiva, antisimétrica y transitiva): **ocurre antes que**.
- Si e y e' son eventos locales en el nodo (proceso) P, y e ocurre antes que e', entonces $e \rightarrow e'$.
- Si $e = send(m)$ (en P) y $e' = receive(m)$ (en P'), entonces $e \rightarrow e'$.

ORDENACION DE LAMPORT



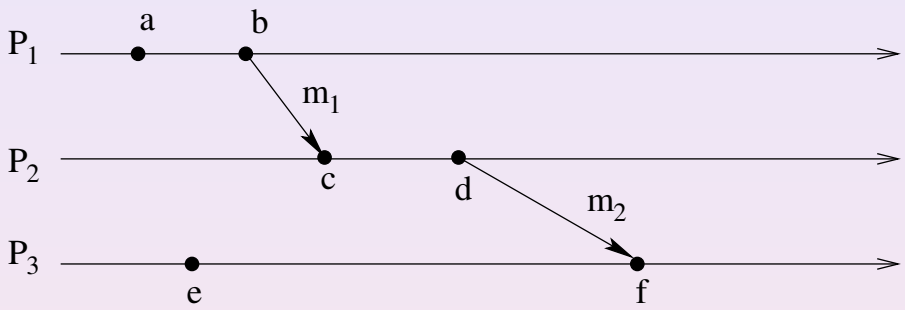
- $a \rightarrow b$ (a ocurre antes que b, localmente).
- $b \rightarrow c$ (b ocurre antes que c, envío-recepción).
- Por transitividad: $a \rightarrow c$, y también: $a \rightarrow f$.
- Sin embargo: $a \not\rightarrow e$, ni $e \not\rightarrow a$. Se dice que son concurrentes: $a || e$.

ORDENACION DE LAMPORT



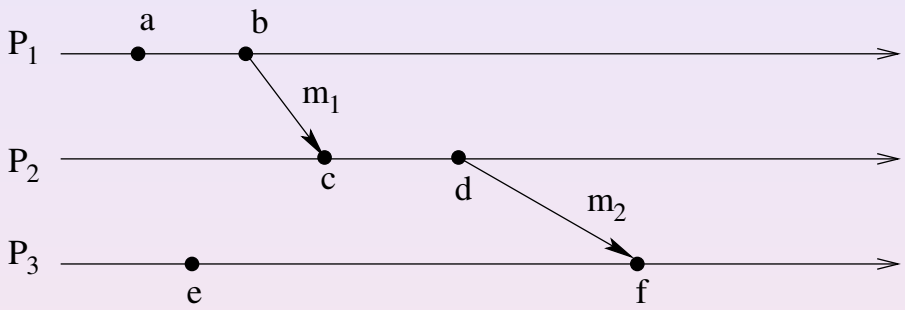
- $a \rightarrow b$ (a ocurre antes que b, localmente).
- $b \rightarrow c$ (b ocurre antes que c, envío-recepción).
- Por transitividad: $a \rightarrow c$, y también: $a \rightarrow f$.
- Sin embargo: $a \not\rightarrow e$, ni $e \not\rightarrow a$. Se dice que son concurrentes: $a \| e$.

ORDENACION DE LAMPORT



- $a \rightarrow b$ (a ocurre antes que b, localmente).
- $b \rightarrow c$ (b ocurre antes que c, envío-recepción).
- Por transitividad: $a \rightarrow c$, y también: $a \rightarrow f$.
- Sin embargo: $a \not\rightarrow e$, ni $e \not\rightarrow a$. Se dice que son concurrentes: $a \| e$.

ORDENACION DE LAMPORT



- $a \rightarrow b$ (a ocurre antes que b, localmente).
- $b \rightarrow c$ (b ocurre antes que c, envío-recepción).
- Por transitividad: $a \rightarrow c$, y también: $a \rightarrow f$.
- Sin embargo: $a \not\rightarrow e$, ni $e \not\rightarrow a$. Se dice que son concurrentes: $a \| e$.

RELOJES LOGICOS DE LAMPORT

Definition (RELOJ LOGICO)

Un reloj lógico es un contador entero que se va incrementando de forma monótona creciente, cuyos valores no guardan relación con ningún reloj físico.

Cada proceso (nodo) P_i lleva asociado un reloj lógico L_i. La marca de tiempo de un evento e se denota por $L_i(e)$.

RELOJES LÓGICOS DE LAMPORT

Reglas de actualización de un reloj lógico:

- L_i se incrementa antes de cada evento en P_i: $L_i = L_i + 1$.
- Cuando P_i envía un mensaje, le asocia la marca de tiempo actual L_i, una vez incrementada en 1. Se envía m con su marca de tiempo.
- Cuando P_j recibe un mensaje $e' = (m, t)$ de P_i, asigna su reloj $L_j = max(L_j, t) + 1$, y le asocia este valor a e'.

RELOJES LÓGICOS DE LAMPORT

Reglas de actualización de un reloj lógico:

- L_i se incrementa antes de cada evento en P_i: $L_i = L_i + 1$.
- Cuando P_i envía un mensaje, le asocia la marca de tiempo actual L_i, una vez incrementada en 1. Se envía m con su marca de tiempo.
- Cuando P_j recibe un mensaje $e' = (m, t)$ de P_i, asigna su reloj $L_j = max(L_j, t) + 1$, y le asocia este valor a e'.

RELOJES LÓGICOS DE LAMPORT

Reglas de actualización de un reloj lógico:

- L_i se incrementa antes de cada evento en P_i: $L_i = L_i + 1$.
- Cuando P_i envía un mensaje, le asocia la marca de tiempo actual L_i, una vez incrementada en 1. Se envía m con su marca de tiempo.
- Cuando P_j recibe un mensaje $e' = (m, t)$ de P_i, asigna su reloj $L_j = max(L_j, t) + 1$, y le asocia este valor a e'.

RELOJES LOGICOS Y ORDENACION

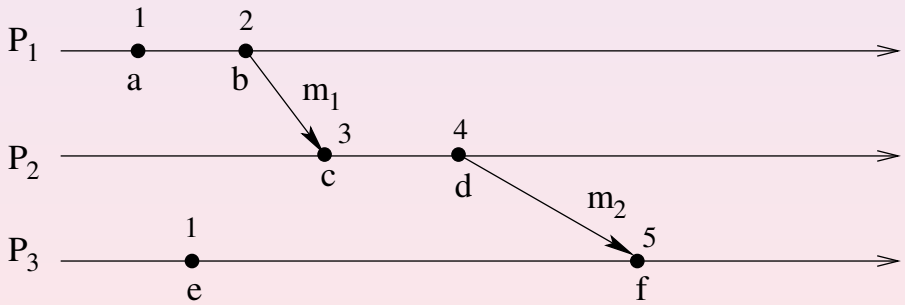
- L así definida es monótona respecto al orden de Lamport:

$$e \rightarrow e' \quad \text{entonces} \quad L(e) \leq L(e')$$

Demostración: Distinción de casos. Inmediato.

$L(e)$ es la marca local de tiempo de e, en el nodo que corresponda.

- El recíproco no es cierto.



$L(b) = 2 > L(e) = 1$, pero son incomparables.

CONTEO DE EVENTOS

- Para cualquier evento e: $L(e) - 1$ indica el número de eventos de la ruta máxima que han debido ocurrir antes que e atendiendo al orden $\rightarrow$.
- **No se cumple la propiedad de consistencia fuerte**: $L(e) < L(e') \not\Rightarrow e \rightarrow e'$.

  Demostración: Contraejemplo anterior con e y b.

ORDENACION TOTAL DE EVENTOS

Definition (ORDEN TOTAL)

Dados dos eventos e_i, e_j, correspondientes a los procesos P_i, P_j (i puede ser igual a j), y con marcas de tiempo T_i, T_j, respectivamente:

- Se les designa de la forma $(T_i, i), (T_j, j)$.
- $(T_i, i) < (T_j, j)$ si y sólo si

$$(T_i < T_j) \text{ o bien } ((T_i = T_j) \wedge (i < j))$$

APLICACIONES DEL ORDEN TOTAL

- Es un orden total: $\forall e_i = (T_i, i), \forall e_j = (T_j, j)$, si $e_i \neq e_j$, entonces o bien $e_i < e_j$, o bien $e_j < e_i$.
- Todos los procesos perciben la misma ordenación total, aunque ésta no tiene por qué corresponderse con la ordenación física en que han ocurrido los eventos.
- Puede utilizarse para regular el acceso a una sección crítica, para seguir un mismo orden de procesamiento por parte de los procesos, etc.

RELOJES VECTORIALES

- Introducidos en 1989 simultáneamente por Mattern y Fidge para resolver el problema de que $L(e) < L(e') \not\Rightarrow e \rightarrow e'$.
- Para un sistema de N procesos (nodos) tendremos un array de N valores enteros (reloj vectorial) en cada nodo:

$$V_i = (V_{i_1}, \ldots, V_{i_N})$$

  V_{i_j} es el último valor conocido del reloj local de P_j por P_i, y V_{i_i} es el reloj local de P_i.
- Cada evento de P_i va etiquetado con un valor del reloj vectorial local.

RELOJES VECTORIALES

- Introducidos en 1989 simultáneamente por Mattern y Fidge para resolver el problema de que $L(e) < L(e') \not\Rightarrow e \rightarrow e'$.
- Para un sistema de N procesos (nodos) tendremos un array de N valores enteros (reloj vectorial) en cada nodo:

$$V_i = (V_{i_1}, \ldots, V_{i_N})$$

  V_{i_j} es el último valor conocido del reloj local de P_j por P_i, y V_{i_i} es el reloj local de P_i.

- Cada evento de P_i va etiquetado con un valor del reloj vectorial local.

RELOJES VECTORIALES

- Introducidos en 1989 simultáneamente por Mattern y Fidge para resolver el problema de que $L(e) < L(e') \not\Rightarrow e \rightarrow e'$.
- Para un sistema de N procesos (nodos) tendremos un array de N valores enteros (reloj vectorial) en cada nodo:

$$V_i = (V_{i_1}, \ldots, V_{i_N})$$

  V_{i_j} es el último valor conocido del reloj local de P_j por P_i, y V_{i_i} es el reloj local de P_i.
- Cada evento de P_i va etiquetado con un valor del reloj vectorial local.

RELOJES VECTORIALES

REGLAS DE ACTUALIZACION:

1. Inicialmente: $V_i[j] := 0, \ \forall i, j = 1, \ldots, N.$
2. Antes de cada evento en P_i:

$$V_i[i] := V_i[i] + 1$$

3. Los mensajes *send* llevan la marca de tiempo (vector) local, tras el paso 2.
4. Cuando P_i recibe un mensaje con marca de tiempo t hace:

$$V_i[j] := max(V_i[j], t[j]) \quad \forall j = 1, \ldots, N$$

Para etiquetar el *receive* aplica entonces el paso 2.

RELOJES VECTORIALES

REGLAS DE COMPARACION:

Las comparaciones $V < V'$ se hacen por componentes:

$V < V'$ si y sólo si $V[i] \leq V'[i]\ \forall i = 1, \ldots, N$, y $\exists j$ tal que $V[j] < V'[j]$

Dos eventos son concurrentes, denotado por $e||e'$, si y sólo si $V(e) \not\leq V(e') \wedge V(e') \not\leq V(e)$.

Ahora sí ocurre que: $V(e) < V(e') \Rightarrow e \rightarrow e'$

RELOJES VECTORIALES

REGLAS DE COMPARACION:

Las comparaciones $V < V'$ se hacen por componentes:

$V < V'$ si y sólo si $V[i] \leq V'[i]\ \forall i = 1, \ldots, N$, y $\exists j$ tal que $V[j] < V'[j]$

Dos eventos son concurrentes, denotado por $e \| e'$, si y sólo si $V(e) \not\leq V(e') \wedge V(e') \not\leq V(e)$.

Ahora sí ocurre que: $V(e) < V(e') \Rightarrow e \rightarrow e'$

RELOJES VECTORIALES

REGLAS DE COMPARACION:

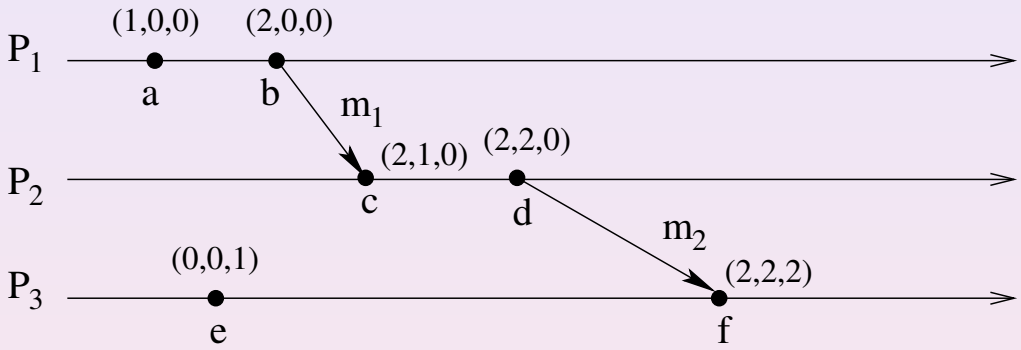
Las comparaciones $V < V'$ se hacen por componentes:

$V < V'$ si y sólo si $V[i] \leq V'[i]\ \forall i = 1, \ldots, N$, y $\exists j$ tal que $V[j] < V'[j]$

Dos eventos son concurrentes, denotado por $e \| e'$, si y sólo si $V(e) \not\leq V(e') \wedge V(e') \not\leq V(e)$.

Ahora sí ocurre que: $\boxed{V(e) < V(e') \Rightarrow e \rightarrow e'}$

RELOJES VECTORIALES



- a y e siguen siendo concurrentes (incomparables).
- Sin embargo, $V(a) = (1,0,0) < V(f) = (2,2,2)$, de modo que $a \rightarrow f$.

RELOJES VECTORIALES

- En P_i, para un evento local e_i, el valor $V(e_i)[i]-1$ denota los eventos que se han producido en P_i antes que el evento local e_i.
- Para $i \neq j$, $V(e_i)[j]$ denota los eventos que se han producido en P_j que preceden causalmente a e_i.
- Para cualquier e_i, se tiene que $\sum_j V(e_i)[j]-1$ es el número total de eventos que preceden causalmente a e_i.

RELOJES VECTORIALES

CONSISTENCIA FUERTE

Demostración de $V(e_i) < V(e_j) \Rightarrow e_i \rightarrow e_j$:

- Caso 1: ambos son del mismo proceso P_k. Entonces ha de ocurrir necesariamente $V(e_i)[k] < V(e_j)[k]$ (un proceso nunca etiqueta de igual forma dos eventos locales).
  De ahí, $e_i \rightarrow e_j$, pues los eventos locales están totalmente ordenados.

RELOJES VECTORIALES

Demostración de $V(e_i) < V(e_j) \Rightarrow e_i \rightarrow e_j$:

- Caso 2: corresponden a procesos distintos, P_i y P_j.
  Sea I el conjunto de índices para los cuales $V(e_i)$ y $V(e_j)$ difieren: $I = \{h \mid V(e_i)[h] < V(e_j)[h]\}$. Claramente $I \neq \emptyset$.
  Procedemos por inducción sobre el cardinal de I:

  Caso base ($|I| = 1$): Tenemos $I = \{h\}$, de hecho sólo puede ocurrir $h = j$, pues para haber cambiado P_j el valor de $V(e_j)[h]$ es porque eventualmente recibió un mensaje con ese valor, y hubiese incrementado a su vez $V(e_j)[j]$, pero si P_i tenía ese valor en e_i(una vez incrementado), es porque P_j le envío después un mensaje, con lo cual a su vez en esa recepción habría incrementado su propio $V(e_i)[i]$, y no podría ser $V(e_i)[i] = V(e_j)[i]$.
  Entonces $h = j$, y e_j puede ser un evento local o recepción de un mensaje. Si es local, podemos mirar el evento anterior y repetir el procedimiento con él. Debe existir tal evento anterior, pues en caso contrario todos los contadores serían 0 anteriormente, y no podría ser $V(e_i) < V(e_j)$. Sea e'_j el evento anterior, que supondremos que es una recepción de mensaje (se puede repetir el razonamiento hasta llegar a uno). Es claro que $V(e_i)[j] < V(e'_j)[j]$, pues P_i no ha podido recibir ese valor. Entonces, al ser todos los demás contadores coincidentes no ha podido haber interacciones intermedias entre otros procesos, y la recepción ha tenido que ser necesariamente de P_i. De ahí: $e_i \rightarrow e_j$.

RELOJES VECTORIALES

Caso general: $|I| = n > 1$:

La hipótesis de inducción dice que el resultado es cierto para un conjunto de índices de cardinal menor que n.
El razonamiento se basa en ir hacia atrás en P_j, buscando el primer evento e'_j en P_j tal que $V(e_i) < V(e'_j) \leq V(e_j)$.
Se demuestra entonces que e'_j tiene que ser un receive. Tomando I' el conjunto de índices para e_i y e'_j, si $|I'| < |I|$ se aplica la hipótesis de inducción. Si son iguales, tomamos el send correspondiente (e_k), se prueba que $V(e_i) < V(e_k)$ y se repite el razonamiento con él. Como se va decrementando la suma total $\sum_j V(e_k)[j]$ en cada uno de estos pasos, el proceso necesariamente termina en la aplicación de la hipótesis de inducción.

RELOJES VECTORIALES

Lemma

Se tiene $V(e_i)[i] \leq V(e_j)[i] \Rightarrow V(e_i)[h] \leq V(e_j)[h], \forall h.$

Demostración: *Inmediato, por la regla de propagación de valores, al transmitir* $V(e_i)[i]$ *va con él el resto de valores de* $V(e_i)$*, como se van tomando máximos, necesariamente* $V(e_j)$ *debe haberlos considerado en sus cómputos cuando recibió directa o indirectamente* $V(e_i)[i]$ *(o un valor actualizado del mismo).*

Lemma

Para el primer e'_j *tal que* $V(e_i) < V(e'_j) \leq V(e_j)$ *(puede ser* e_j*), tomando* $I' = \{h \mid V(e_i)[h] < V(e'_j)[h]\}$*, se tiene:* $|I'| \leq |I|$.

Demostración: *Inmediata, de* $V(e_i) < V(e'_j) \leq V(e_j)$.

RELOJES VECTORIALES

Lemma

Si $|I'| > 1$, entonces el evento e'_j debe ser necesariamente un receive.

Demostración: *De $V(e_i) < V(e'_j)$, y considerando que $|I'| > 1$, debe haber al menos dos índices h_k tales que $V(e_i)[h_k] < V(e'_j)[h_k]$. Para que esto sea posible e'_j no puede ser el primer evento de P_j, y existe e''_j delante de él. Si e'_j fuera local, por la definición de actualización: $V(e'_j)[j] = V(e''_j)[j] + 1$, y $V(e''_j)[h] = V(e'_j)[h]$, $\forall h \neq j$. De ahí: $V(e_i) < V(e''_j)$, lo que contradice que e'_j sea el primero.*

Lemma

Considerando $|I| > 1$ y tomando el evento e_k correspondiente al send asociado al e'_j (receive), se tiene: $V(e_i) \leq V(e_k)$.

Demostración: *Basta ver que $V(e_i)[i] \leq V(e_k)[i]$ por el Lema 1. Por hipótesis tenemos que $V(e_i][i] \leq V(e'_j)[i]$. Puede ser que $V(e'_j)[i]$ obtenga ese valor de e_k o bien de un e''_j anterior (definición). Si fuera uno anterior, tendríamos $V(e_i)[i] \leq V(e''_j)[i]$, y por el Lema 1, $V(e_i) < V(e''_j)$, lo cual contradice que e'_j era el primero.*
Entonces e_k facilita ese valor $V(e_i)[i]$ a e'_j. De ahí: $V(e_i)[i] \leq V(e_k)[i]$.

RELOJES VECTORIALES

Ventajas e inconvenientes:

- Ventaja: Podemos saber si dos eventos son concurrentes o están relacionados causalmente comparando sus relojes vectoriales.
- Inconveniente: Requiere mayor espacio de almacenamiento para las marcas de tiempo, y los tamaños de los mensajes se multiplican por N: mayor carga de transmisión.

  Charron-Bost [1991] demostraron que no se puede reducir el número de relojes (n=número de procesos), para que se mantenga la propiedad de consistencia fuerte.

RELOJES VECTORIALES

Ventajas e inconvenientes:

- Ventaja: Podemos saber si dos eventos son concurrentes o están relacionados causalmente comparando sus relojes vectoriales.
- Inconveniente: Requiere mayor espacio de almacenamiento para las marcas de tiempo, y los tamaños de los mensajes se multiplican por N: mayor carga de transmisión.

  Charron-Bost [1991] demostraron que no se puede reducir el número de relojes (n=número de procesos), para que se mantenga la propiedad de consistencia fuerte.

RELOJES VECTORIALES

Optimización de Singhal y Kshemkalyani [1992]:

- Para un sistema de miles de procesos es inaceptable el coste en términos de tamaño de los mensajes.
- Las interacciones en estos casos son muy localizadas, entre ciertos procesos, e.d., en dos envíos consecutivos entre procesos habrá muy pocas variaciones en los valores.
- Se envían entonces sólo las discrepancias con el envío anterior a ese proceso: $\{(i, v_i)\}$, siendo i los índices y v_i el nuevo valor para el índice i.

INDICE:

ESTADOS GLOBALES

- No tenemos estados globales exactos al no disponer de un reloj global.
- Queremos dar respuestas a preguntas basadas en estados globales: ¿Se ha bloqueado el sistema?.
- Se trata de definir estados globales abstractos o lógicos de sistemas distribuidos.
- Sobre los estados globales se deciden propiedades, como si existe un bloqueo mutuo o si un algoritmo distribuido ha terminado su ejecución.

ESTADOS GLOBALES

Definition (ESTADOS CONSISTENTES)

Dado un sistema formado por n procesos, un estado del mismo es una tupla $s = (s_1, \ldots, s_n)$, donde s_i es el estado local del proceso i.

Cada proceso ha ejecutado hasta llegar a s_i una serie de eventos:

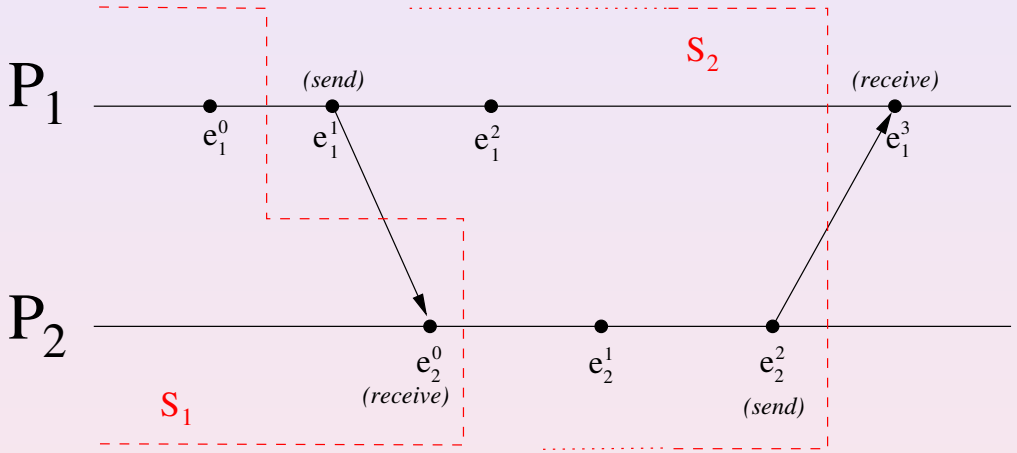
$$h_i^1 h_i^2 \ldots h_i^{k_i}$$

Un estado s es consistente si para todo evento e que forme parte de esas historias, todo evento anterior a él con respecto al orden de Lamport también forma parte de una de esas historias:

$$\forall e \in H : f \rightarrow e \Rightarrow f \in H$$

siendo H todo el conjunto de eventos realizados hasta llegar a s.

EJEMPLO



S_1 es un estado inconsistente

S_2 es un estado consistente

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

[1985. Chandy & Lamport]
Permite determinar estados globales consistentes de sistemas distribuidos. Hipótesis de trabajo:

- No fallan ni los procesos ni los canales.
- Los canales son unidireccionales, y la entrega se produce en el orden de envío.
- Grafo de procesos y canales: fuertemente conexo.
- Cualquier proceso puede iniciar el algoritmo en cualquier momento.
- Mientras se ejecuta el algoritmo los procesos pueden seguir trabajando normalmente.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Se captura información sobre el estado de los procesos y el estado de los canales (mensajes en tránsito).
- Se emplean los marcadores (mensajes especiales) para avisar a los procesos de que deben recopilar su información de estado. Contienen información del iniciador y del instante local en que se inició.
- Cualquier proceso puede iniciar el algoritmo: registra su estado, envía un marcador por todos sus canales de salida y activa el registro de los mensajes entrantes por todos sus canales de entrada.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Se captura información sobre el estado de los procesos y el estado de los canales (mensajes en tránsito).
- Se emplean los marcadores (mensajes especiales) para avisar a los procesos de que deben recopilar su información de estado. Contienen información del iniciador y del instante local en que se inició.
- Cualquier proceso puede iniciar el algoritmo: registra su estado, envía un marcador por todos sus canales de salida y activa el registro de los mensajes entrantes por todos sus canales de entrada.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Se captura información sobre el estado de los procesos y el estado de los canales (mensajes en tránsito).
- Se emplean los marcadores (mensajes especiales) para avisar a los procesos de que deben recopilar su información de estado. Contienen información del iniciador y del instante local en que se inició.
- Cualquier proceso puede iniciar el algoritmo: registra su estado, envía un marcador por todos sus canales de salida y activa el registro de los mensajes entrantes por todos sus canales de entrada.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

REGLAS:

- **(A) P_i recibe el marcador por el canal c:**

  Si P_i aún no ha registrado su estado,
  Registra su estado ahora;
  Registra el estado de c como vacío;
  Activar registro de mensajes que lleguen por los otros canales de entrada;
  Enviar marcador (Regla (B))
  Sino
  P_i registra el estado de c como el conjunto de mensajes recibidos por él, desde activación;
  Fsi;

- **(B) P_i envía el marcador:**

  Cuando P_i ha registrado su estado, envía un marcador sobre cada canal de salida c, antes de enviar ningún otro mensaje sobre c;

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

REGLAS:

- **(A) P_i recibe el marcador por el canal c:**

  Si P_i aún no ha registrado su estado,
  Registra su estado ahora;
  Registra el estado de c como vacío;
  Activar registro de mensajes que lleguen por los otros canales de entrada;
  Enviar marcador (Regla (B))
  Sino
  P_i registra el estado de c como el conjunto de mensajes recibidos por él, desde activación;
  Fsi;

- **(B) P_i envía el marcador:**

  Cuando P_i ha registrado su estado, envía un marcador sobre cada canal de salida c, antes de enviar ningún otro mensaje sobre c;

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Obsérvese que sólo se envía el marcador por los canales de salida la primera vez que recibe el marcador.
- Varios procesos pueden lanzar el algoritmo simultáneamente: los marcadores son diferentes (se toman *fotos* diferentes.
- El algoritmo finaliza para P_i cuando ha recibido el marcador por todos sus canales de entrada. Entonces P_i puede enviar su información local de estado a los demás.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Obsérvese que sólo se envía el marcador por los canales de salida la primera vez que recibe el marcador.
- Varios procesos pueden lanzar el algoritmo simultáneamente: los marcadores son diferentes (se toman *fotos* diferentes.
- El algoritmo finaliza para P_i cuando ha recibido el marcador por todos sus canales de entrada. Entonces P_i puede enviar su información local de estado a los demás.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- Obsérvese que sólo se envía el marcador por los canales de salida la primera vez que recibe el marcador.
- Varios procesos pueden lanzar el algoritmo simultáneamente: los marcadores son diferentes (se toman *fotos* diferentes.
- El algoritmo finaliza para P_i cuando ha recibido el marcador por todos sus canales de entrada. Entonces P_i puede enviar su información local de estado a los demás.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- El estado global consistente se obtiene como la unión de los estados locales de cada proceso, una vez que todos han terminado.
- Todos obtienen la misma *foto* del sistema.
- Coste en número de mensajes marcadores: $O(e)$, siendo e el número de arcos de la red.

ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- El estado global consistente se obtiene como la unión de los estados locales de cada proceso, una vez que todos han terminado.
- Todos obtienen la misma *foto* del sistema.
- Coste en número de mensajes marcadores: $O(e)$, siendo e el número de arcos de la red.

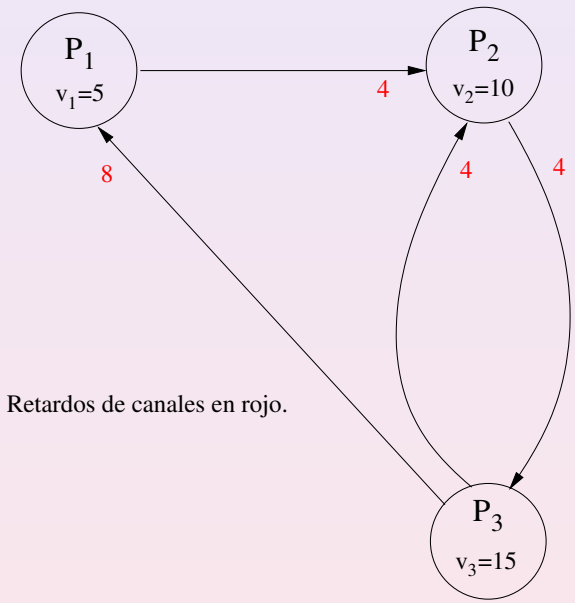
ALGORITMO SNAPSHOT DE CHANDY Y LAMPORT

- El estado global consistente se obtiene como la unión de los estados locales de cada proceso, una vez que todos han terminado.
- Todos obtienen la misma *foto* del sistema.
- Coste en número de mensajes marcadores: $O(e)$, siendo e el número de arcos de la red.

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 0



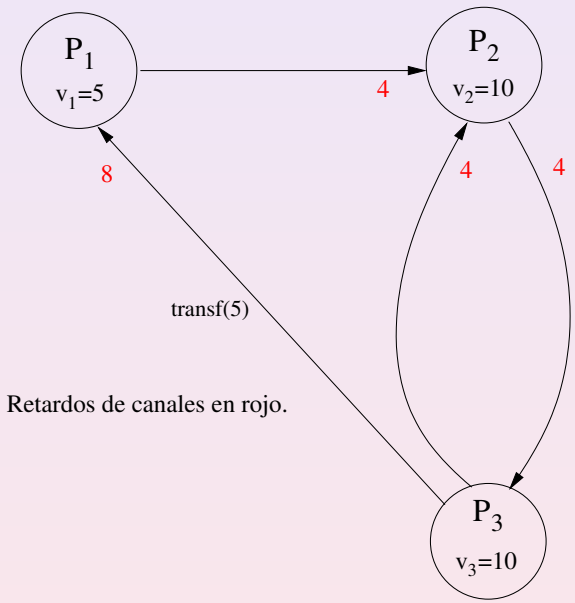
Retardos de canales en rojo.

t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
t=3	P_1 inicia el algoritmo snapshot.
t=7	Llega el marcador de P_1 a P_2
t=9	Llega el mensaje de transferencia de P_3 a P_1
t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 1



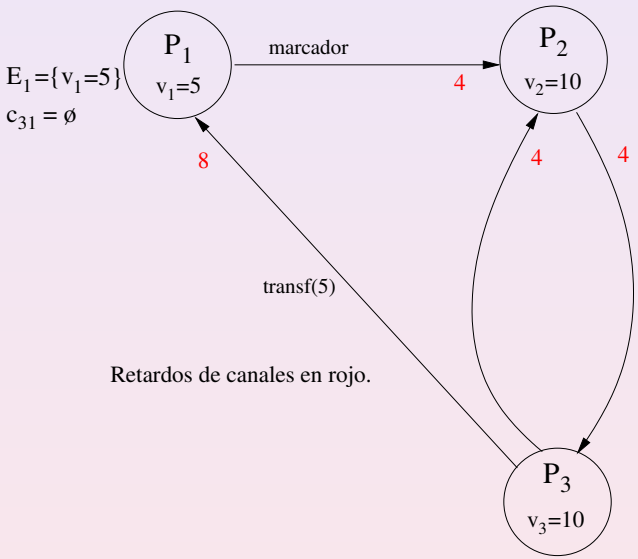
Retardos de canales en rojo.

√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
t=3	P_1 inicia el algoritmo snapshot.
t=7	Llega el marcador de P_1 a P_2
t=9	Llega el mensaje de transferencia de P_3 a P_1
t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 3

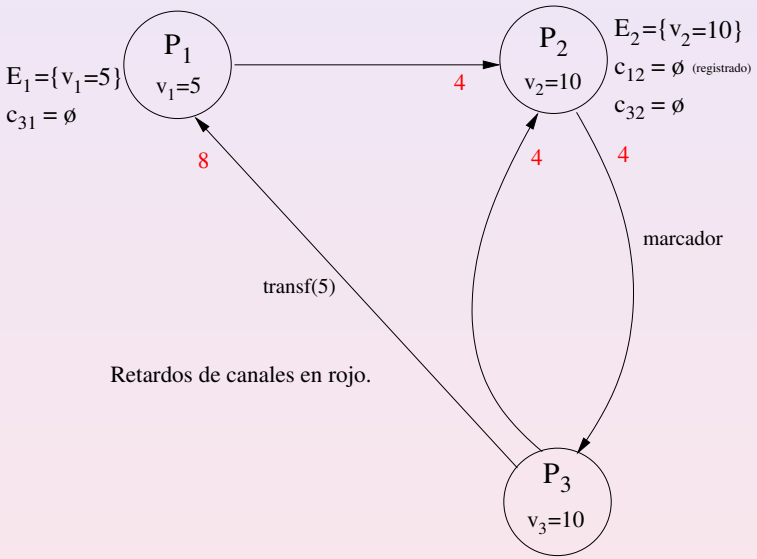


√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
t=7	Llega el marcador de P_1 a P_2
t=9	Llega el mensaje de transferencia de P_3 a P_1
t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 7

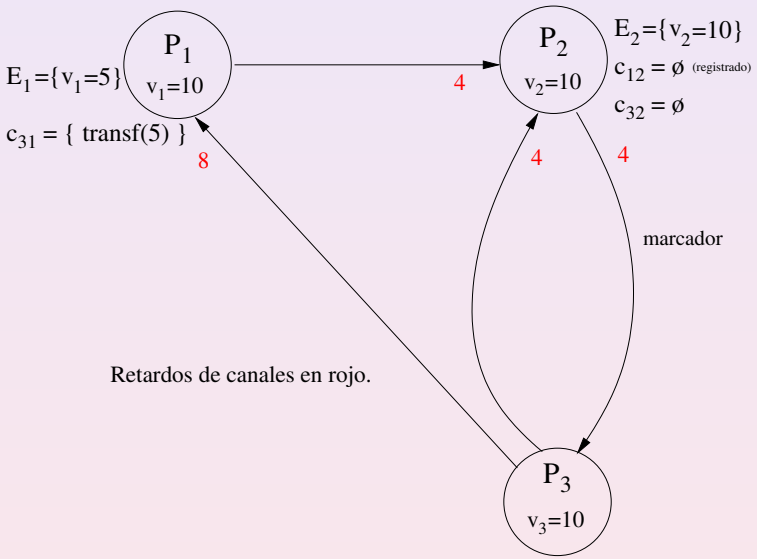


√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
√ t=7	Llega el marcador de P_1 a P_2
t=9	Llega el mensaje de transferencia de P_3 a P_1
t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 9

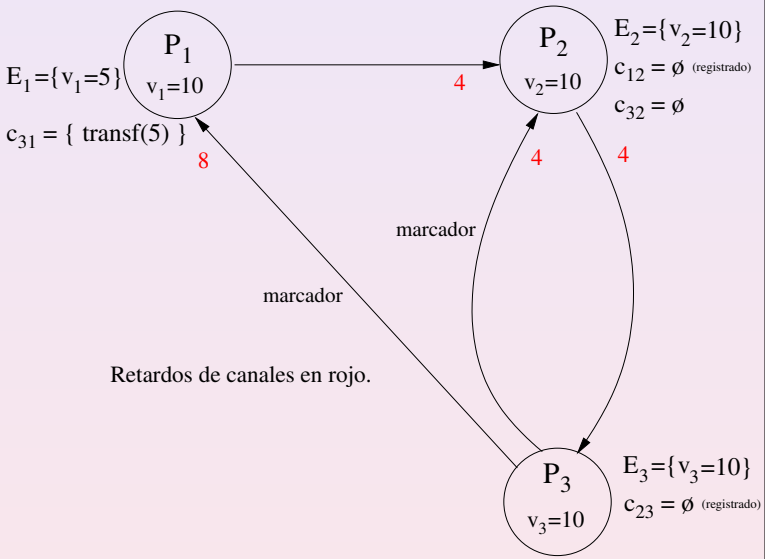


√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
√ t=7	Llega el marcador de P_1 a P_2
√ t=9	Llega el mensaje de transferencia de P_3 a P_1
t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 11

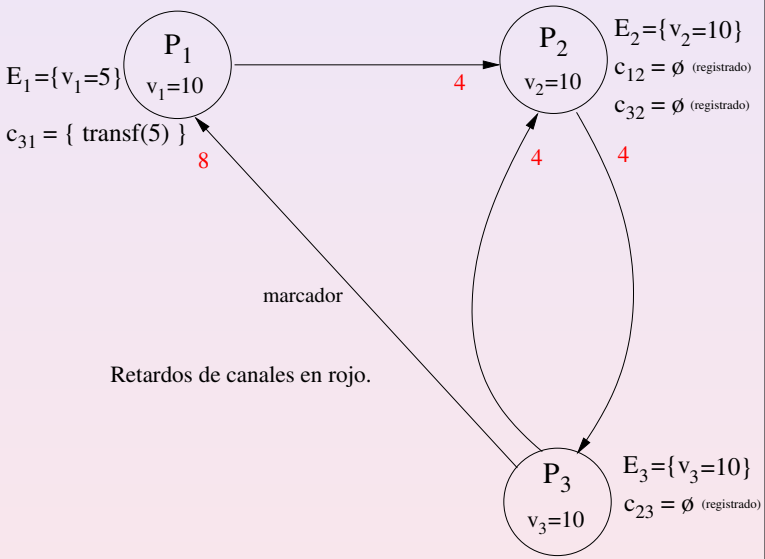


√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
√ t=7	Llega el marcador de P_1 a P_2
√ t=9	Llega el mensaje de transferencia de P_3 a P_1
√ t=11	P_3 recibe el marcador desde P_2
t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 15

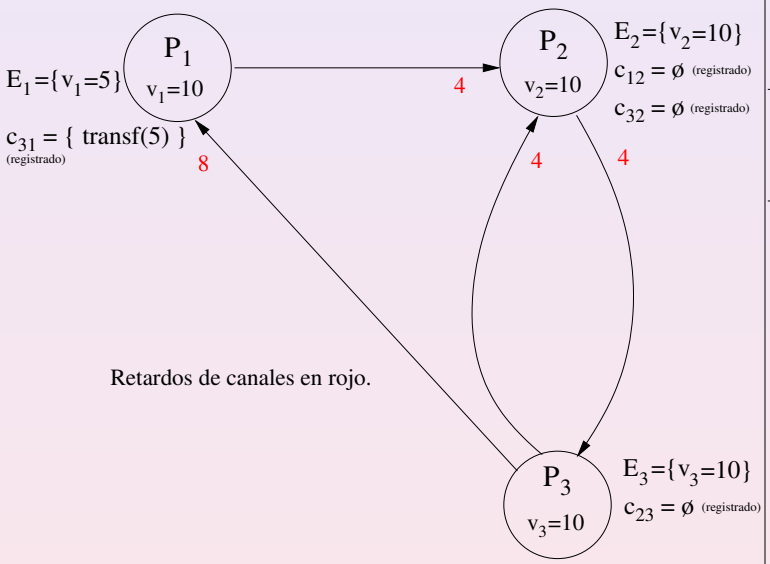


√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
√ t=7	Llega el marcador de P_1 a P_2
√ t=9	Llega el mensaje de transferencia de P_3 a P_1
√ t=11	P_3 recibe el marcador desde P_2
√ t=15	P_2 recibe el marcador de P_3
t=19	P_1 recibe el marcador de P_3

ILUSTRACION DEL ALG. DE SNAPSHOT

(por simplicidad suponemos un reloj global y acciones propias de los procesos sin tiempo)

TIEMPO = 19



√ t=1	P_3 envía mensaje de transferencia a P_1 : transf(5)
√ t=3	P_1 inicia el algoritmo snapshot.
√ t=7	Llega el marcador de P_1 a P_2
√ t=9	Llega el mensaje de transferencia de P_3 a P_1
√ t=11	P_3 recibe el marcador desde P_2
√ t=15	P_2 recibe el marcador de P_3
√ t=19	P_1 recibe el marcador de P_3

ALG. DE SNAPSHOT

Lemma

El algoritmo de Snapshot termina y garantiza estados consistentes.

Demostración:

- *La terminación es inmediata, al recorrer los marcadores los diferentes arcos de la red una sola vez. Como el grafo es fuertemente conexo se garantiza el retorno del marcador al nodo iniciador y al resto de nodos por todos sus canales de entrada.*
- *La consistencia se prueba comprobando que para todo receive su correspondiente send ha sido considerado en el corte del estado. Para ello basta tener en cuenta que cuando se registra el receive es porque ha llegado un marcador por ese canal. No se ha podido realizar ningún envío de mensaje entre la recepción del marcador en el nodo anterior y su envío por el canal, de modo que las acciones hasta el send fueron consideradas antes de registrar su estado (el marcador debió llegar a ese nodo por primera vez, si no, no se hubiera propagado).*

INDICE:

EXCLUSION MUTUA DISTRIBUIDA

¿ Cómo implementar la exclusión mutua distribuida?

- No disponemos de memoria compartida $\Rightarrow$ No tenemos semáforos ni monitores: Uso de mensajes.
- Hipótesis de trabajo:
  - Sistema asíncrono (tiempos de transmisión desconocidos).
  - Procesos que no fallan, ni tienen comportamientos arbitrarios (bizantinos).
  - Envío de mensajes fiable: uso de acks, time-outs y retransmisiones si es necesario.

EXCLUSION MUTUA DISTRIBUIDA

¿ Cómo implementar la exclusión mutua distribuida?

- No disponemos de memoria compartida $\Rightarrow$ No tenemos semáforos ni monitores: Uso de mensajes.
- Hipótesis de trabajo:
  - Sistema asíncrono (tiempos de transmisión desconocidos).
  - Procesos que no fallan, ni tienen comportamientos arbitrarios (bizantinos).
  - Envío de mensajes fiable: uso de acks, time-outs y retransmisiones si es necesario.

EXCLUSION MUTUA DISTRIBUIDA

¿ Cómo implementar la exclusión mutua distribuida?

- No disponemos de memoria compartida $\Rightarrow$ No tenemos semáforos ni monitores: Uso de mensajes.
- Hipótesis de trabajo:
  - Sistema asíncrono (tiempos de transmisión desconocidos).
  - Procesos que no fallan, ni tienen comportamientos arbitrarios (bizantinos).
  - Envío de mensajes fiable: uso de acks, time-outs y retransmisiones si es necesario.

CONDICIONES DE EXCLUSION MUTUA

Definition (Condiciones para la exclusión mutua)

Deben cumplirse las condiciones siguientes:

- A lo sumo un proceso está en su sección crítica.
- Las peticiones de entrada en sección crítica deben atenderse en tiempo finito.
- Los procesos ejecutándose fuera de su sección crítica no pueden bloquear el acceso a la misma a los demás.
- El algoritmo no depende de las velocidades de ejecución de los procesos o del número de procesadores disponibles.

RENDIMIENTO

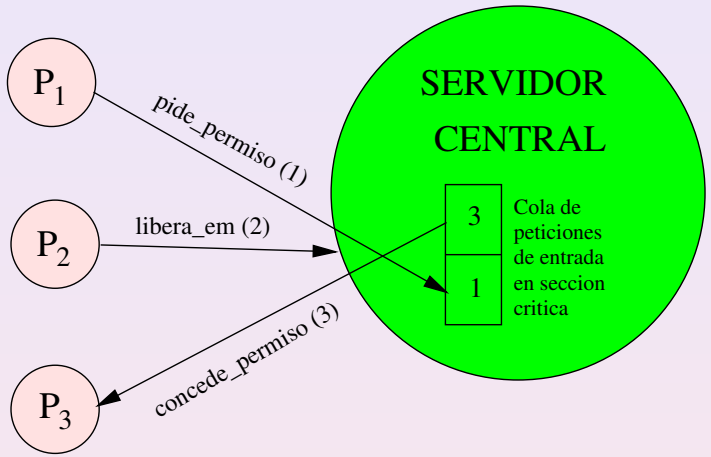
Los algoritmos de exclusión mutua distribuida se evalúan con respecto a:

- Ancho de banda utilizado por el algoritmo, medido en términos del número de mensajes enviados, y tamaño de los mismos.
- Tiempos de demora para entrada y salida en sección crítica (coste de ejecución de los protocolos).
- Efecto sobre la productividad: Tiempo de demora entre un proceso saliente de e.m. y un proceso entrante en e.m. (retraso de sincronización).

ALGORITMOS

- Algoritmo de servidor central.
- Algoritmo de anillo.
- Algoritmo de Ricart y Agrawala.

ALGORITMO CENTRALIZADO



- El servidor emplea un cola de peticiones.
- Las va atendiendo en el orden en que están en la cola, enviando un *testigo* al proceso autorizado.
- Cuando un proceso sale de su sección crítica debe enviar un mensaje (el testigo) al servidor.

ALGORITMO CENTRALIZADO

- Si ningún proceso está en sección crítica el servidor tiene el *testigo*.
- No se garantiza el orden en que fueron realizadas las peticiones, debido a los diferentes retrasos de transmisión de los mensajes.
- Rendimiento:
  - Entrada en S.C.: 2 mensajes.
  - Salida de S.C.: 1 mensaje.
  - El servidor es un cuello de botella potencial.
  - Retraso de sincronización: 2 mensajes.

ALGORITMO CENTRALIZADO

- Si ningún proceso está en sección crítica el servidor tiene el *testigo*.
- No se garantiza el orden en que fueron realizadas las peticiones, debido a los diferentes retrasos de transmisión de los mensajes.
- Rendimiento:
  - Entrada en S.C.: 2 mensajes.
  - Salida de S.C.: 1 mensaje.
  - El servidor es un cuello de botella potencial.
  - Retraso de sincronización: 2 mensajes.

ALGORITMO CENTRALIZADO

- Si ningún proceso está en sección crítica el servidor tiene el *testigo*.
- No se garantiza el orden en que fueron realizadas las peticiones, debido a los diferentes retrasos de transmisión de los mensajes.
- Rendimiento:
  - Entrada en S.C.: 2 mensajes.
  - Salida de S.C.: 1 mensaje.
  - El servidor es un cuello de botella potencial.
  - Retraso de sincronización: 2 mensajes.

ALGORITMO CENTRALIZADO

- ¿Qué ocurre si falla el proceso que tiene el testigo o se pierde un mensaje?.

  Solución: Fijar un tiempo máximo para permanecer en sección crítica. Transcurrido este tiempo no se puede acceder al recurso.

  Lease: clase Java para limitar el tiempo que podemos usar una referencia a un objeto remoto.

- ¿Cómo estimamos el tiempo que vamos a usar un recurso? No podemos, pero es posible *renovar* un lease.

  Para evitar conductas maliciosas se puede limitar el número de renovaciones.

ALGORITMO CENTRALIZADO

- ¿Qué ocurre si falla el proceso que tiene el testigo o se pierde un mensaje?.

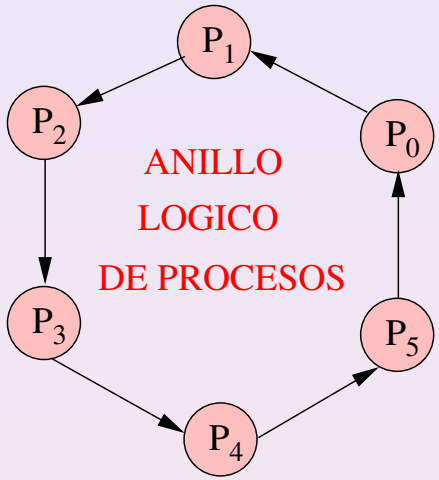
  Solución: Fijar un tiempo máximo para permanecer en sección crítica. Transcurrido este tiempo no se puede acceder al recurso.

  Lease: clase Java para limitar el tiempo que podemos usar una referencia a un objeto remoto.

- ¿Cómo estimamos el tiempo que vamos a usar un recurso? No podemos, pero es posible *renovar* un lease.

  Para evitar conductas maliciosas se puede limitar el número de renovaciones.

ALGORITMO EN ANILLO



- P_i tiene un canal de comunicación con $P_{(i+1) \bmod N}$, $i = 0, \ldots, N-1$.
- Un mensaje especial (testigo o token) va circulando por el anillo.
- P_i puede entrar en S.C. cuando recibe el testigo. Lo enviará a $P_{(i+1) \bmod N}$ cuando salga de S.C.

ALGORITMO EN ANILLO

Evaluación del algoritmo en anillo:

- Requiere un alto ancho de banda: el testigo circula constantemente si ningún proceso quiere entrar en S.C.
- Coste de entrada en S.C.: de 0 a N mensajes circulando, hasta que le llega el testigo.
- Coste de salida de S.C.: Un mensaje.
- Retraso de sincronización: entre 1 y N mensajes.

ALGORITMO EN ANILLO

- Si falla el proceso que tiene el testigo hay que *relanzar* el testigo. Se ejecuta un algoritmo de elección para nombrar un coordinador, reconfigurar el anillo y lanzar el testigo una sola vez.
- Detección de fallo: tiempo máximo en uso del testigo.

  Con el número de procesos y una estimación de los tiempos de transmisión podemos determinar que ha habido un fallo.
- ¿Y si un proceso necesita más tiempo?

  Puede enviar un mensaje *SigoVivo* al anillo, para que los demás sepan que el testigo sigue activo.

ALGORITMO DE RICART Y AGRAWALA

Basado en multienvío y relojes lógicos:

- Cada proceso se identifica numéricamente (i para P_i).
- Cada proceso tiene su reloj lógico T_i.
- Mensaje de petición de entrada en S.C. de P_i anotado con (T_i, i). Lo envía a todos.
- Cada proceso tiene una variable de estado, cuyo valor puede ser:

RELEASED	Fuera de S.C.
WANTED	Quiere entrar en S.C.
HELD	Está en S.C.

- Cada proceso tiene una cola asociada de peticiones.

ALGORITMO DE RICART Y AGRAWALA

- Inicialización: $Estado_i := RELEASED$, $\forall i = 0, \ldots, N-1$.
- Protocolo de entrada en S.C. de P_i:

  $Estado_i := WANTED$;
  $T_i := T_i + 1$;
  $Send((T_i, i), \{P_j\}_{j \neq i})$;
  Espera recibir las $N-1$ respuestas afirmativas;
  $Estado_i := HELD$;

- P_j recibe un mensaje de petición de entrada (T_i, i):

  if $(Estado_j = HELD)$ **or** ($(Estado_j = WANTED)$ **and** $(T_j, j) < (T_i, i)$)
  then
    Inserta (T_i, i) en cola de P_j sin enviar respuesta;
  else
    Responder afirmativamente a P_i;

ALGORITMO DE RICART Y AGRAWALA

- Protocolo de salida de S.C. de P_i:

  $Estado_i$:= *RELEASED*;
  Responder afirmativamente a todas las peticiones en la cola de P_i;

- En caso de conflicto, se decide atendiendo al orden de Lamport.
- Evaluación:
  - Protocolo de entrada: $N-1$ mensajes de solicitud + $N-1$ mensajes de respuesta = $2.(N-1)$ mensajes.
  - Protocolo de salida: $< N$ mensajes (los que hay en la cola).
  - Retraso de sincronización: 1 mensaje.

ALGORITMO DE RICART Y AGRAWALA

- Protocolo de salida de S.C. de P_i:

  $Estado_i$:= *RELEASED*;
  Responder afirmativamente a todas las peticiones en la cola de P_i;

- En caso de conflicto, se decide atendiendo al orden de Lamport.
- Evaluación:
  - Protocolo de entrada: $N-1$ mensajes de solicitud + $N-1$ mensajes de respuesta = $2.(N-1)$ mensajes.
  - Protocolo de salida: $< N$ mensajes (los que hay en la cola).
  - Retraso de sincronización: 1 mensaje.

ALGORITMO DE RICART Y AGRAWALA

- Protocolo de salida de S.C. de P_i:

  $Estado_i$:= *RELEASED*;
  Responder afirmativamente a todas las peticiones en la cola de P_i;

- En caso de conflicto, se decide atendiendo al orden de Lamport.
- Evaluación:
  - Protocolo de entrada: $N-1$ mensajes de solicitud + $N-1$ mensajes de respuesta = $2.(N-1)$ mensajes.
  - Protocolo de salida: $< N$ mensajes (los que hay en la cola).
  - Retraso de sincronización: 1 mensaje.

INDICE:

ELECCIONES

- Un coordinador muere.
- ¿Cómo elegir un proceso coordinador?.

  Ejemplo:
  Servidor central en el alg. de exclusión mutua distribuida.

- Algoritmos:
  - De anillo de Tanenbaum.
  - De anillo de Chang y Roberts.
  - “Bully”.
  - Elecciones en wireless.

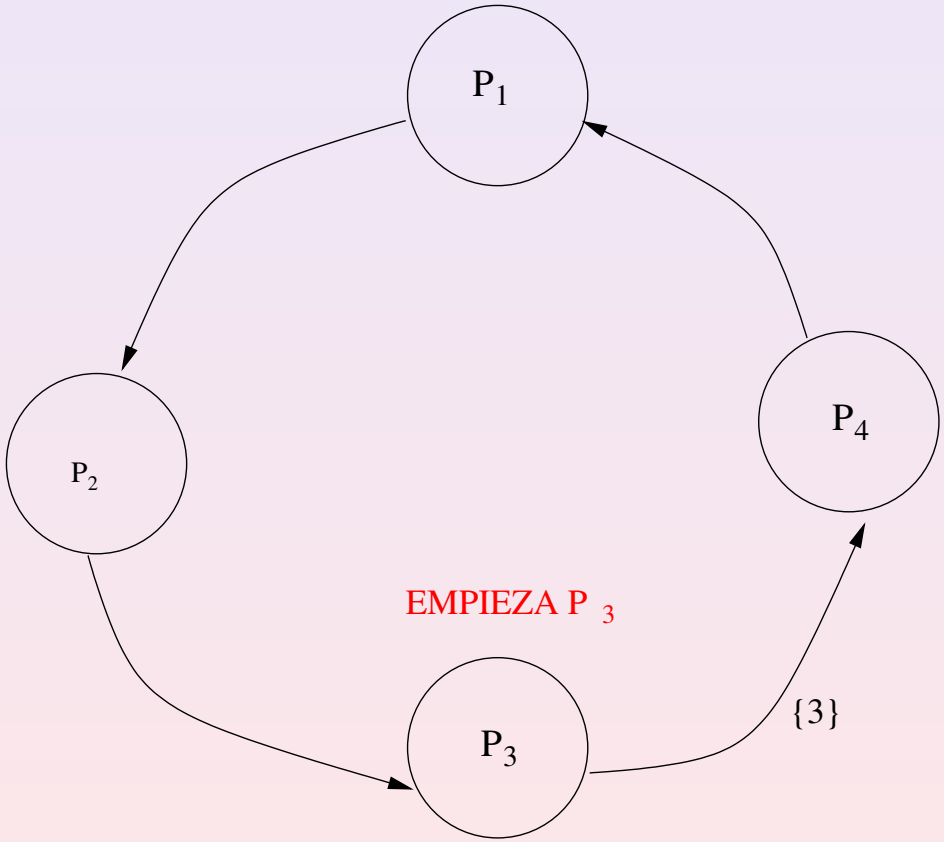
ALGORITMO DE ANILLO DE TANENBAUM

- Procesos en anillo, con canales de P_i a $P_{(i+1) mod N}$. En realidad no es necesario que los identificadores sean consecutivos, basta configurar un anillo lógico.
- Hipótesis: sistema asíncrono, libre de fallos.
- Se elige aquél cuyo identificador numérico es mayor.
- P_i envía al sucesor su identificador para iniciar la elección.

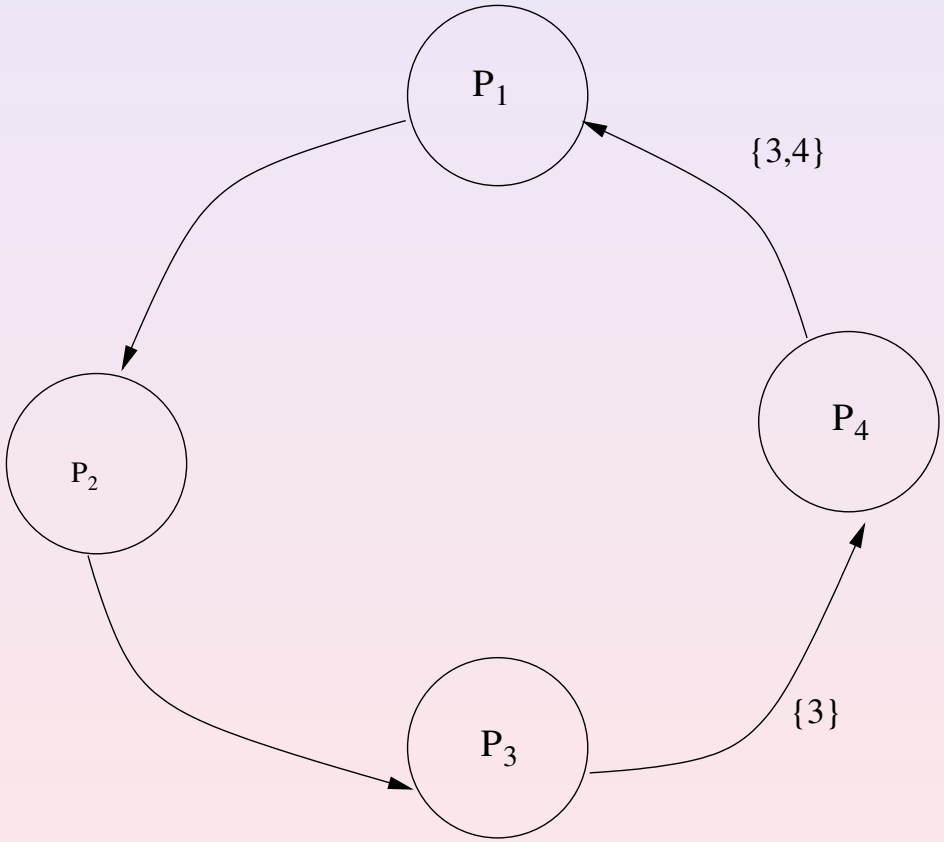
ALGORITMO DE ANILLO DE TANENBAUM

- Cuando P_j recibe un mensaje de elección con una lista de identificadores:
  - Si no contiene su identificador, lo añade a la lista asociada al mensaje y reenvía el mensaje al sucesor.
  - Si contiene su identificador, toma el mayor de la lista y ése es el coordinador. Envía un mensaje *"elegido"* al sucesor con el identificador del coordinador, ese mensaje recorrerá todo el anillo.
- Coste del acuerdo: $2.N$ mensajes hasta que todos obtienen un mensaje de tipo *"elegido"*.

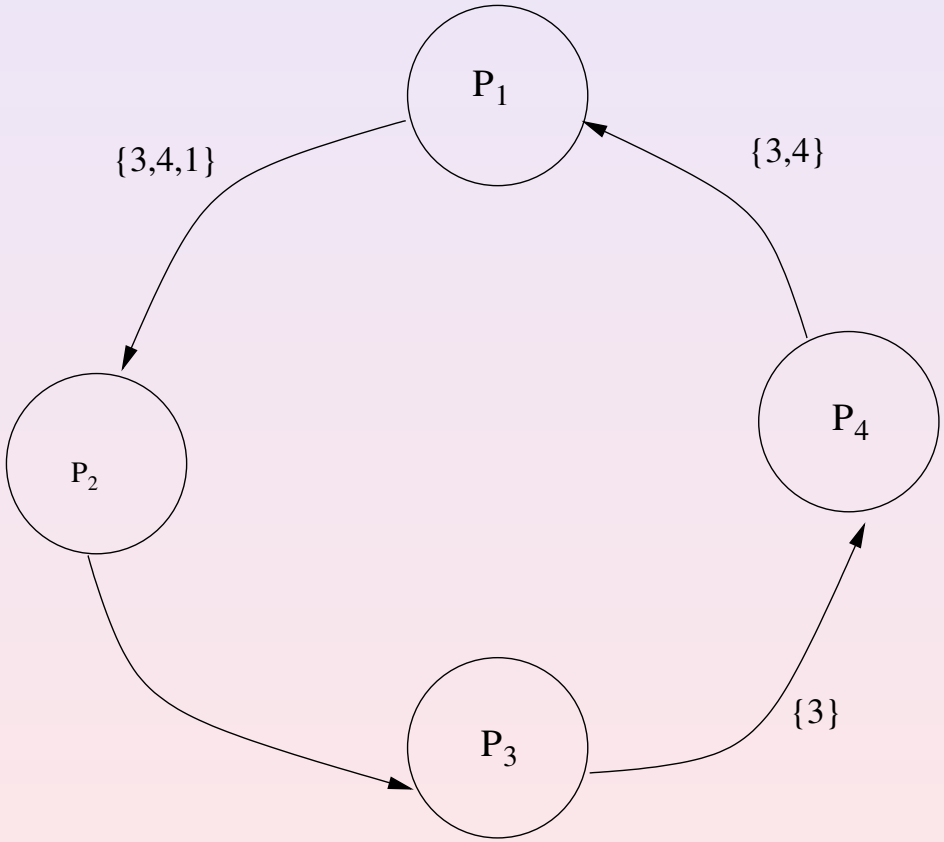
ALGORITMO DE ANILLO DE TANENBAUM



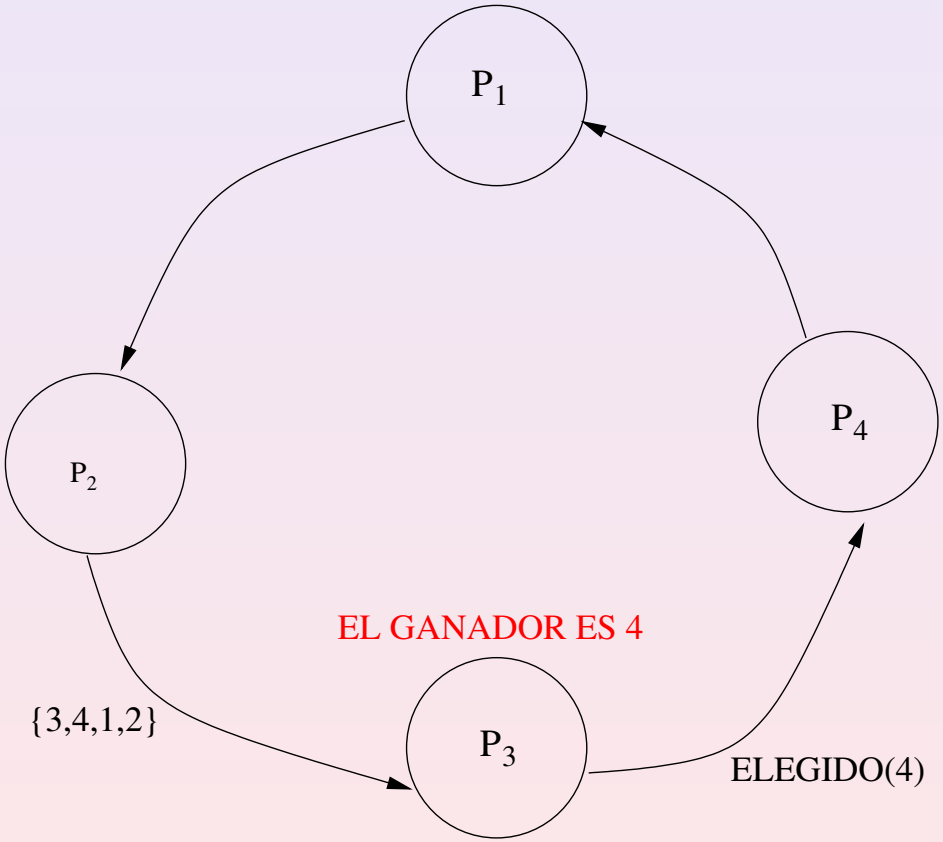
ALGORITMO DE ANILLO DE TANENBAUM



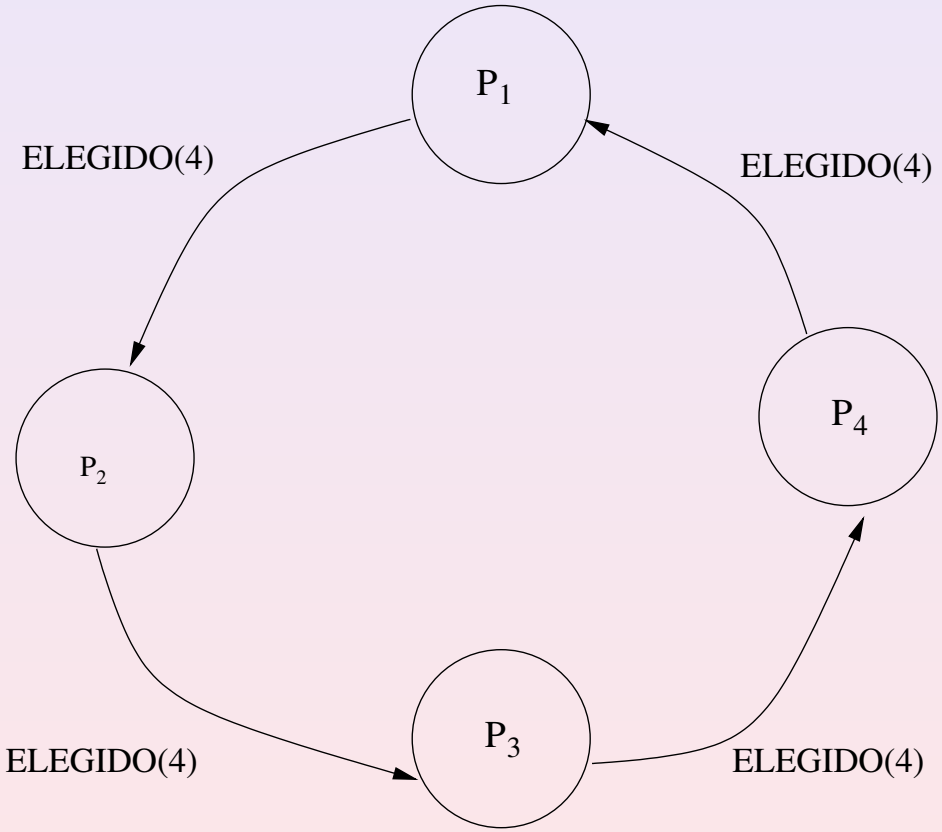
ALGORITMO DE ANILLO DE TANENBAUM



ALGORITMO DE ANILLO DE TANENBAUM



ALGORITMO DE ANILLO DE TANENBAUM



ALGORITMO DE ANILLO DE CHANG Y ROBERTS

- Se les marca como participantes o no participantes para eliminar elecciones múltiples.
- Inicialmente se marcan todos como “no-participantes”.
- Permite evitar múltiples elecciones simultáneas.
- Si P_i no es participante puede iniciar el algoritmo, enviando un mensaje de elección a su sucesor (contiene su identificador), marcándose como *participante*.

ALGORITMO DE ANILLO DE CHANG Y ROBERTS

- P_j recibe un mensaje de elección (k):
  - Si es participante, y $j < k$, reenvía el mensaje al sucesor.
  - Si es participante, y $j > k$, no hace nada (ya hay una elección en marcha).
  - Si no es participante, se marca como participante y envía $max(j, k)$ al sucesor.

ALGORITMO DE ANILLO DE CHANG Y ROBERTS

- P_j recibe un mensaje de elección (k):
  - Si es participante, y $j < k$, reenvía el mensaje al sucesor.
  - Si es participante, y $j > k$, no hace nada (ya hay una elección en marcha).
  - Si no es participante, se marca como participante y envía $max(j, k)$ al sucesor.

ALGORITMO DE ANILLO DE CHANG Y ROBERTS

- P_j recibe un mensaje de elección (k):
  - Si es participante, y $j < k$, reenvía el mensaje al sucesor.
  - Si es participante, y $j > k$, no hace nada (ya hay una elección en marcha).
  - Si no es participante, se marca como participante y envía $max(j, k)$ al sucesor.

ALGORITMO DE ANILLO DE CHANG Y ROBERTS

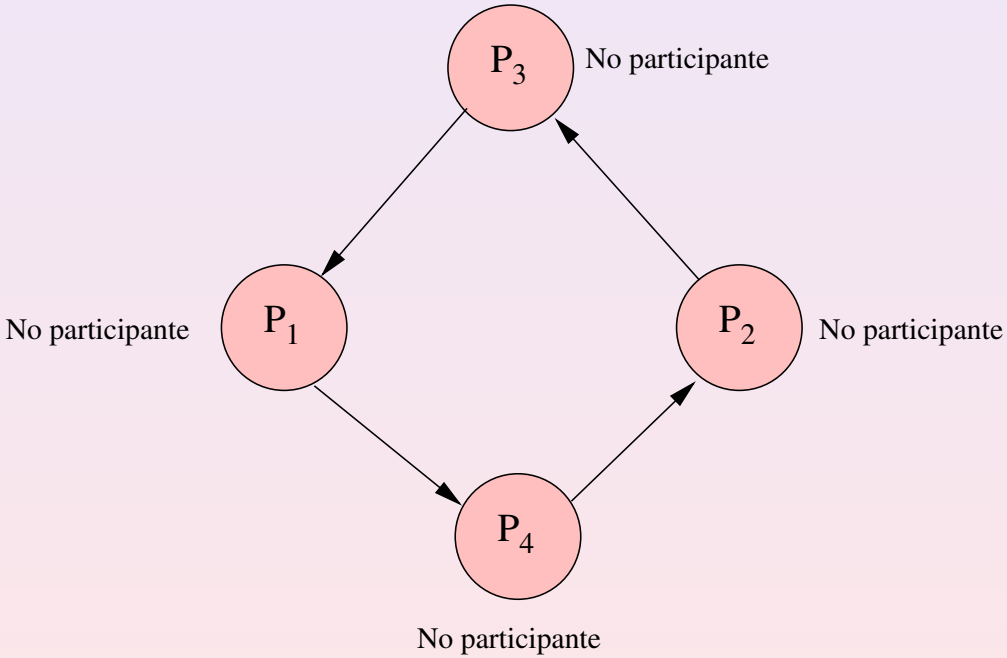
- Cuando P_j recibe un mensaje *elección(j)*: se convierte en coordinador. Envía al sucesor un mensaje *elegido* con su identificador.
- Cuando P_k recibe un mensaje *elegido*(j), anota que P_j es el coordinador, y se marca como "no-participante". Si $j \neq k$ reenvía el mensaje al sucesor.

ALGORITMO DE ANILLO DE CHANG Y ROBERTS

- Cuando P_j recibe un mensaje *elección(j)*: se convierte en coordinador. Envía al sucesor un mensaje *elegido* con su identificador.
- Cuando P_k recibe un mensaje *elegido*(j), anota que P_j es el coordinador, y se marca como “no-participante”. Si $j \neq k$ reenvía el mensaje al sucesor.

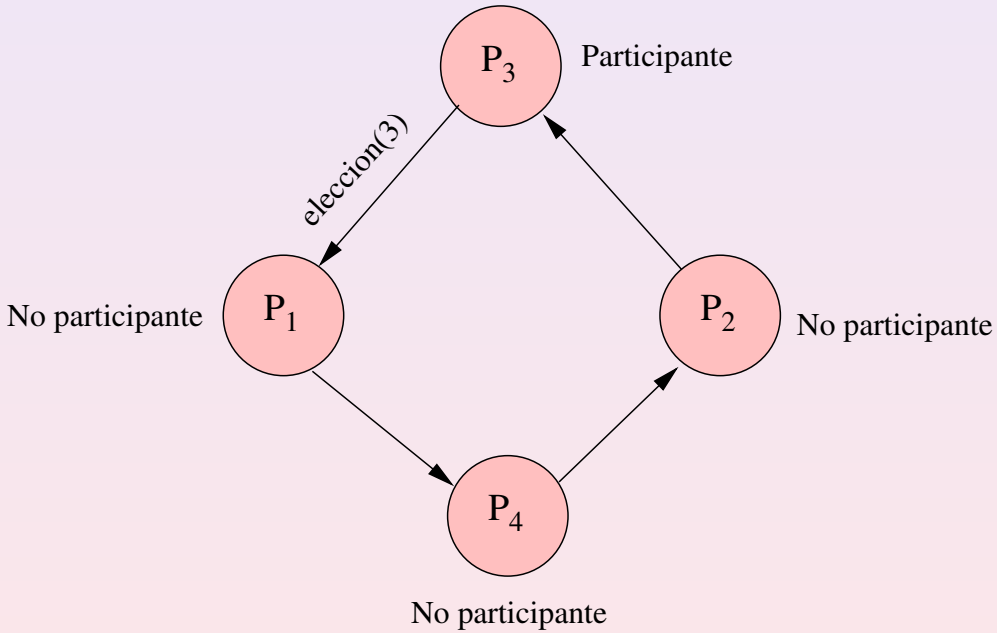
EJEMPLO

SITUACION INICIAL



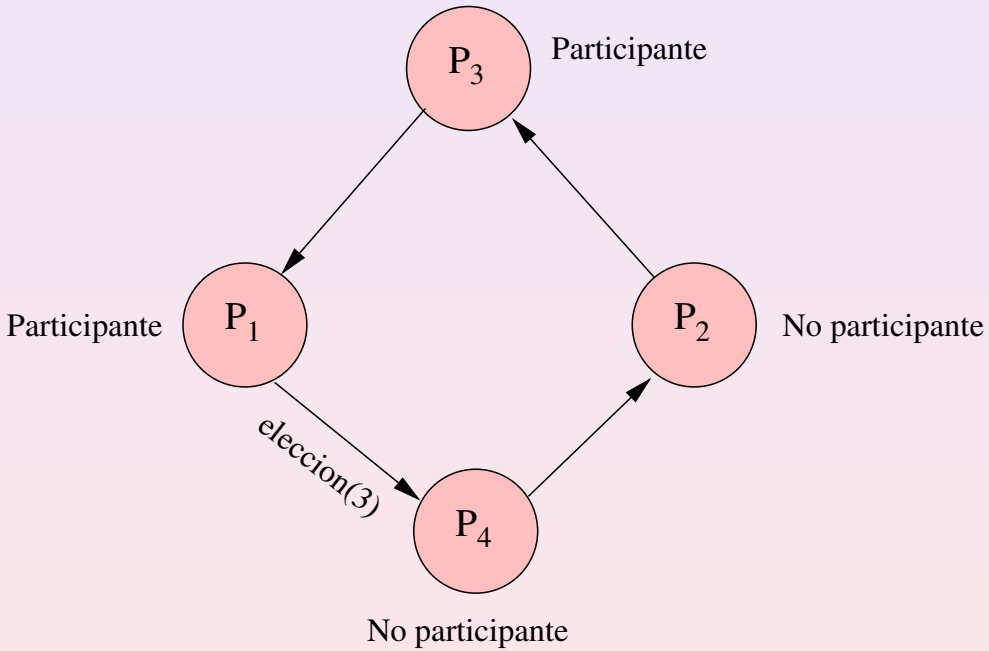
EJEMPLO

PASO 1.– P3 INICIA EL ALGORITMO



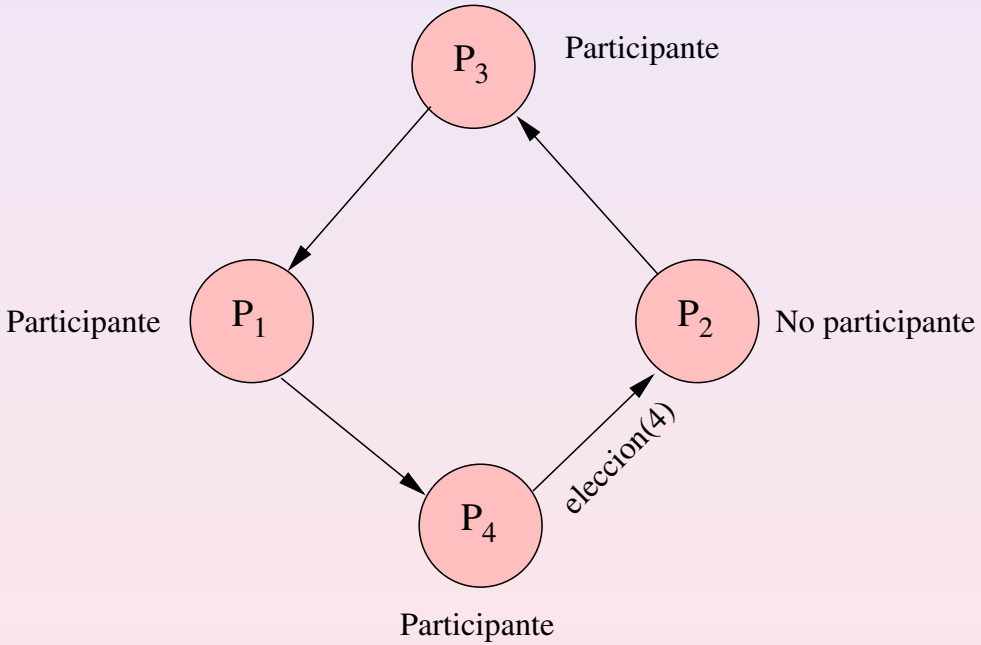
EJEMPLO

PASO 2.– P1 RECIBE EL MENSAJE DE ELECCION



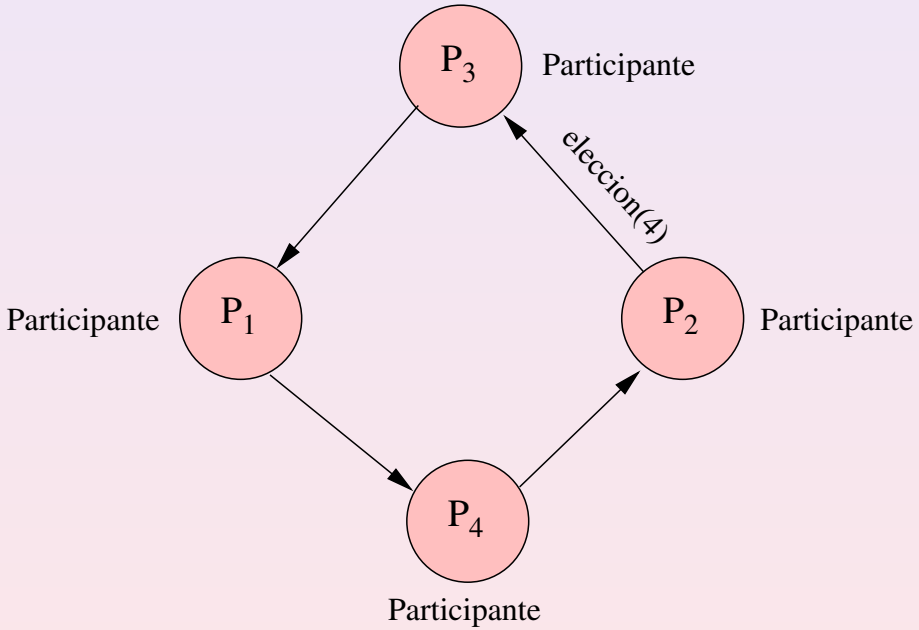
EJEMPLO

PASO 3.– P4 RECIBE EL MENSAJE DE ELECCION



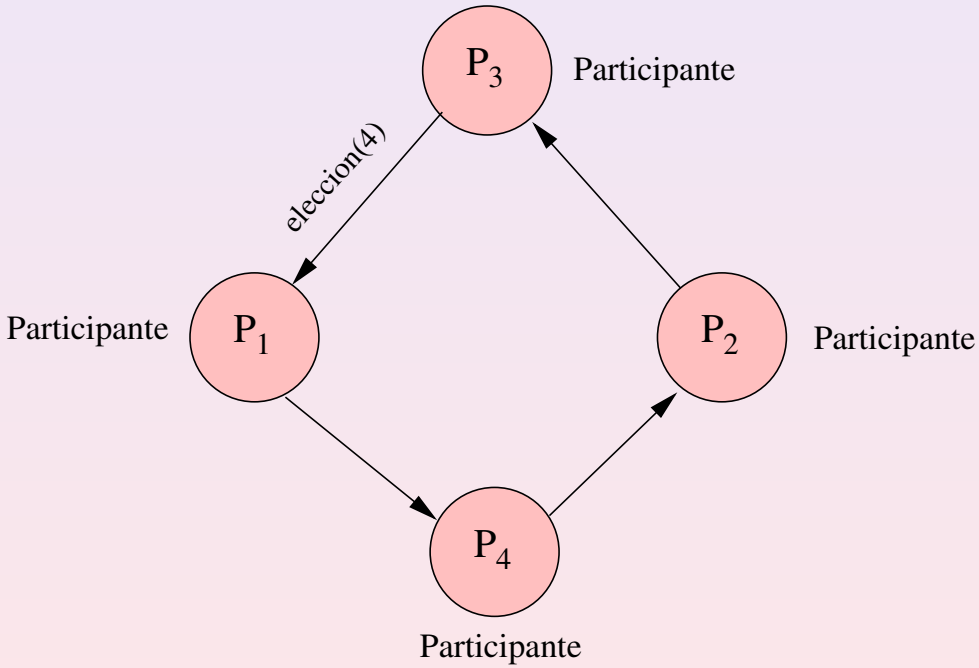
EJEMPLO

PASO 4.– P2 RECIBE EL MENSAJE DE ELECCION



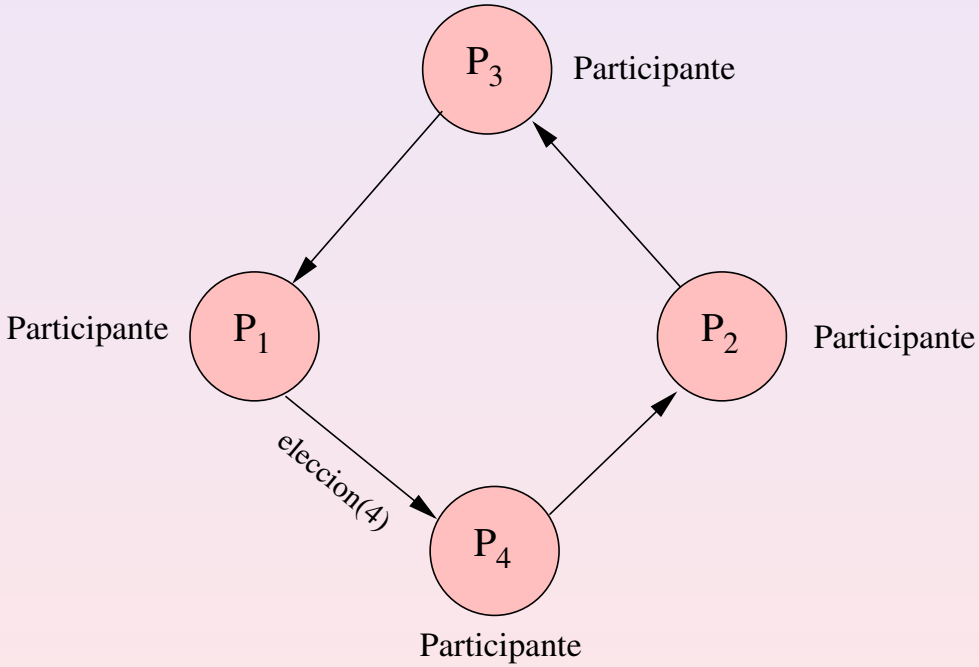
EJEMPLO

PASO 5.– P3 RECIBE DE NUEVO EL MENSAJE



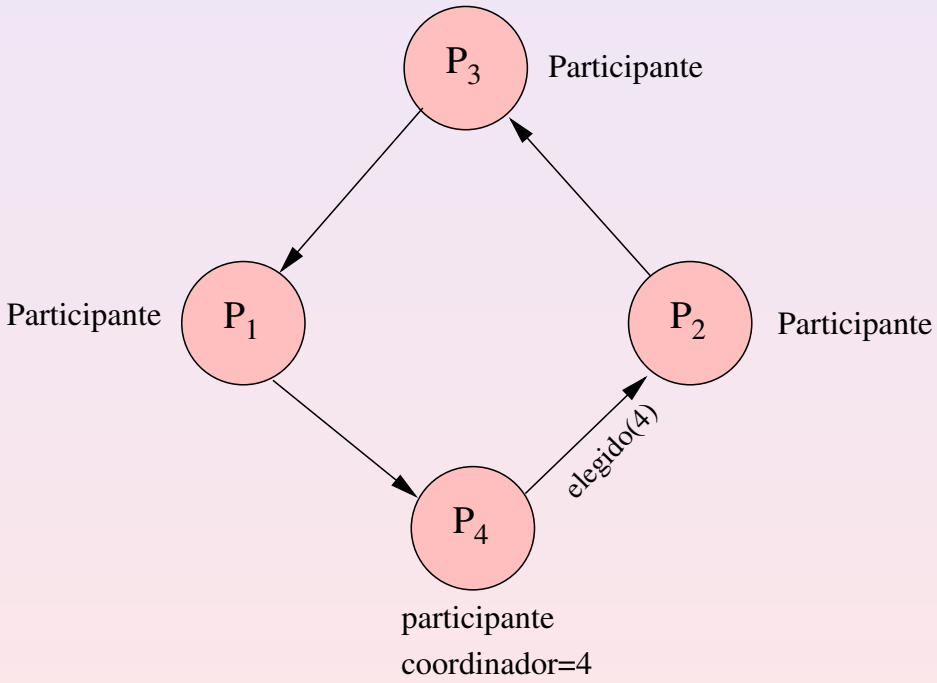
EJEMPLO

PASO 6.– P1 RECIBE DE NUEVO EL MENSAJE



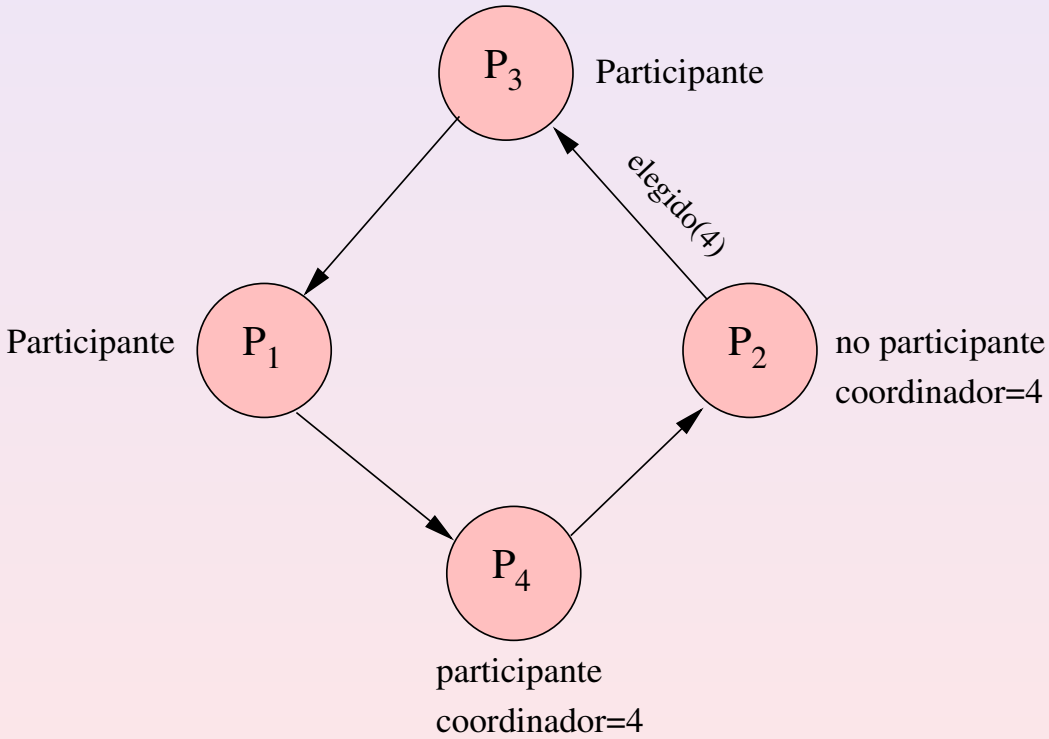
EJEMPLO

PASO 7.– P4 SE CONVIERTE EN COORDINADOR



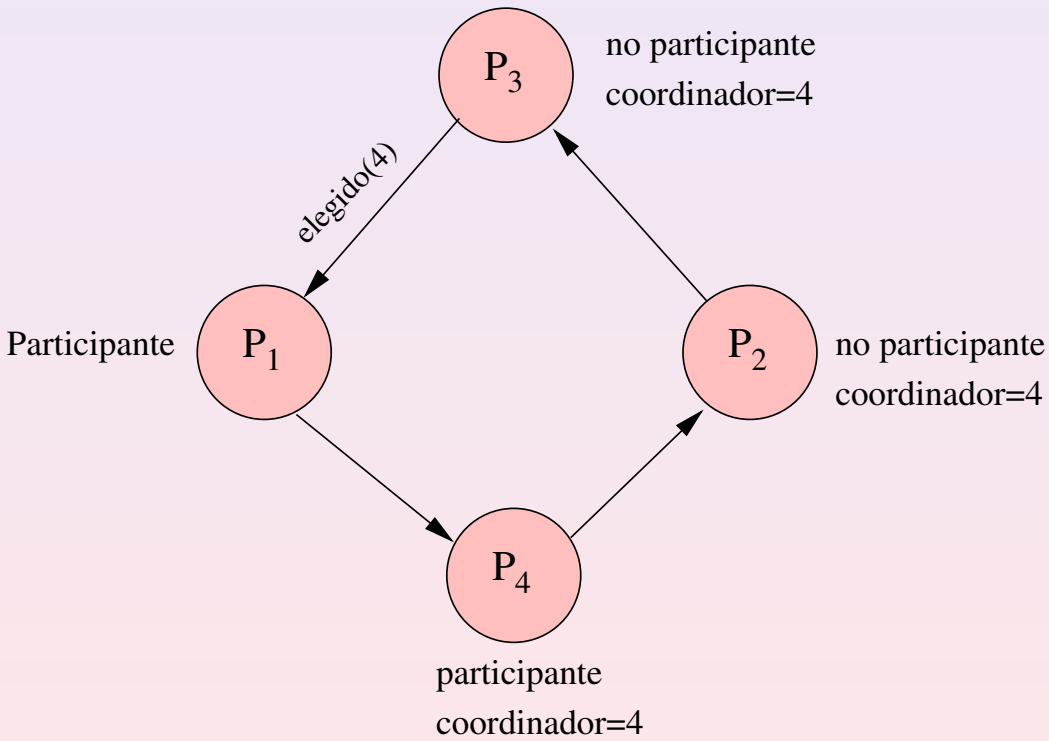
EJEMPLO

PASO 8.– P2 RECIBE MENSAJE DE ELEGIDO



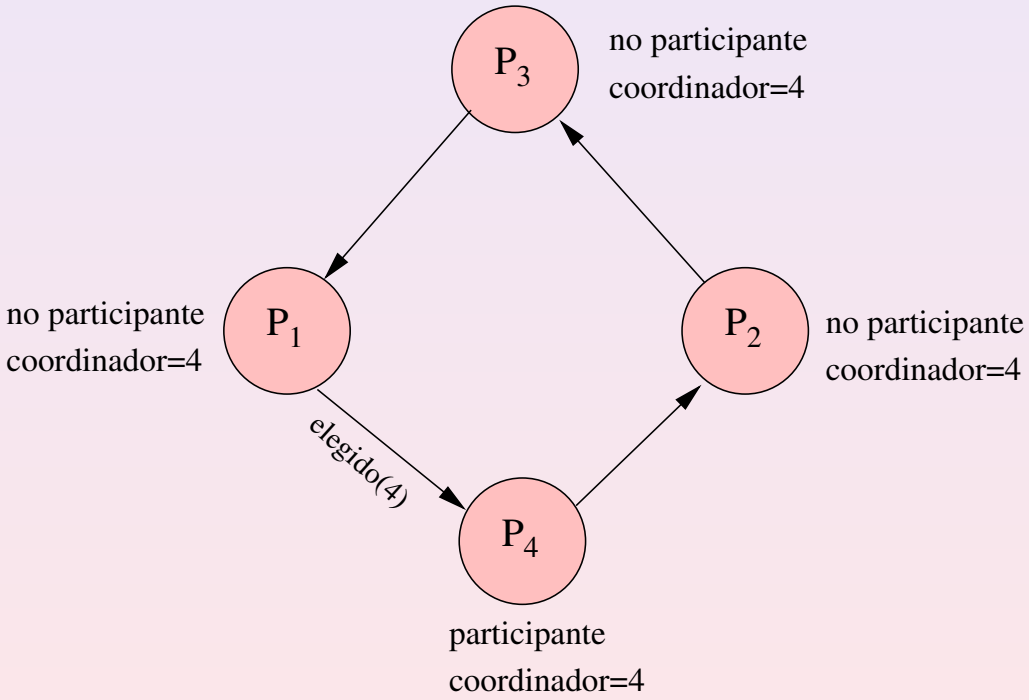
EJEMPLO

PASO 9.– P3 RECIBE MENSAJE DE ELEGIDO



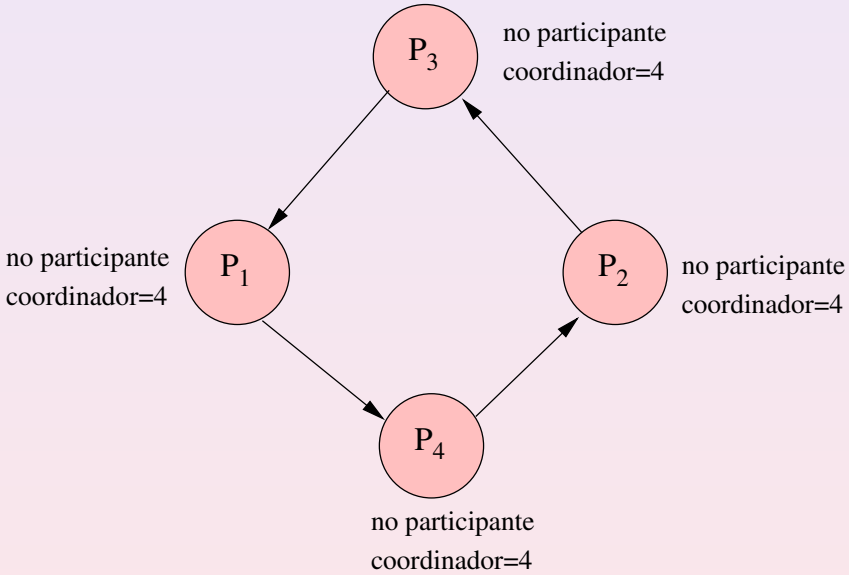
EJEMPLO

PASO 10.– P1 RECIBE MENSAJE DE ELEGIDO



EJEMPLO

PASO 11.– P4 RECIBE MENSAJE DE ELEGIDO Y ACABA



ALGORITMO DE ANILLO DE CHANG Y ROBERTS

Evaluación:

- Coste del acuerdo:
  - El peor caso se produce cuando el coordinador va a ser el nodo antecesor del iniciador.
  - Habría N mensajes de elección (primera ronda), más $N-1$ mensajes de elección de la segunda ronda, más N mensajes de tipo *elegido*.
  - Total: $3.N-1$ mensajes.

ALGORITMO BULLY

[1982. Hector García-Molina].

- Permite el fallo de los procesos, pero asume envío fiable.
- Cada proceso conoce los identificadores de los demás, aunque no sabe si están vivos.
- Sistema síncrono: *time-out* conocido en interacciones, para poder detectar fallos de los procesos.

ALGORITMO BULLY

- Tres tipos de mensaje:
  - Mensaje de elección: para anunciar el inicio de una elección.
  - Mensaje de respuesta (a la elección).
  - Mensaje de coordinador: para anunciar la identidad del coordinador.

ALGORITMO BULLY

- Objetivo: elegir como coordinador aquél cuyo identificador sea mayor numéricamente.
- T_{trans}: Retraso máximo de transmisión de un mensaje.
- $T_{process}$: Tiempo máximo para procesar un mensaje.
- $T = 2.T_{trans} + T_{process}$: Tiempo máximo requerido para una interacción entre dos procesos.
- Cuando un proceso detecta que el coordinador actual no responde (T u.t. sin recibir nada de él) inicia el proceso de elección: envía un mensaje de elección a los que tienen identificador más alto.

ALGORITMO BULLY

- Si no recibe respuesta de ninguno en tiempo T él es el nuevo coordinador (envía un mensaje de coordinador a los que tienen identificadores más bajos).
- Si recibe algún mensaje de respuesta en tiempo T, queda a la espera de recibir el mensaje de coordinador. Si no llega en un tiempo limitado, reinicia la elección.
- Cuando un proceso recibe el mensaje de elección envía un mensaje de respuesta al emisor (su identificador es mayor), e inicia la elección.

ALGORITMO BULLY

- Si no recibe respuesta de ninguno en tiempo T él es el nuevo coordinador (envía un mensaje de coordinador a los que tienen identificadores más bajos).
- Si recibe algún mensaje de respuesta en tiempo T, queda a la espera de recibir el mensaje de coordinador. Si no llega en un tiempo limitado, reinicia la elección.
- Cuando un proceso recibe el mensaje de elección envía un mensaje de respuesta al emisor (su identificador es mayor), e inicia la elección.

ALGORITMO BULLY

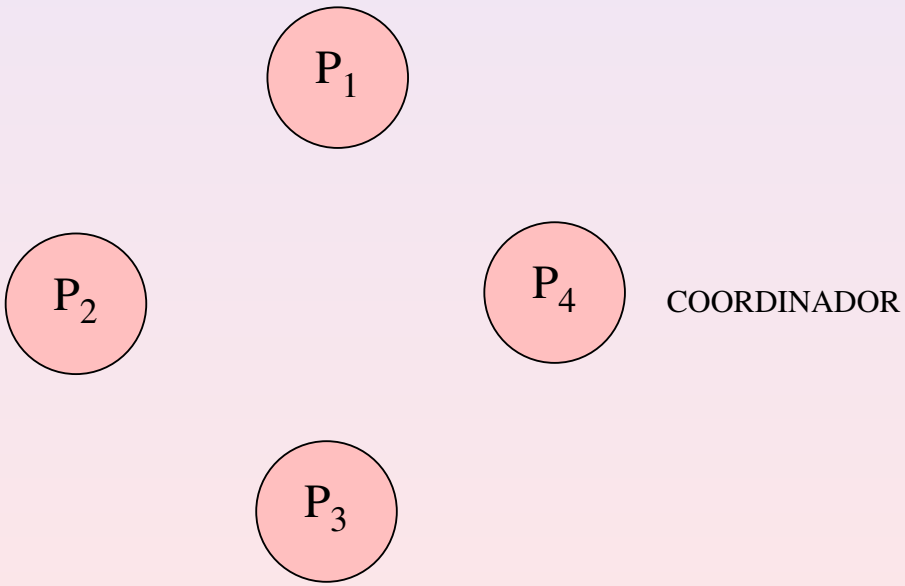
- Si no recibe respuesta de ninguno en tiempo T él es el nuevo coordinador (envía un mensaje de coordinador a los que tienen identificadores más bajos).
- Si recibe algún mensaje de respuesta en tiempo T, queda a la espera de recibir el mensaje de coordinador. Si no llega en un tiempo limitado, reinicia la elección.
- Cuando un proceso recibe el mensaje de elección envía un mensaje de respuesta al emisor (su identificador es mayor), e inicia la elección.

ALGORITMO BULLY

- Si se lanza un proceso nuevo para reemplazar a uno que abortó, se reinicia la elección, incluso aunque exista un coordinador actualmente.
- Será coordinador el que tenga el identificador mayor.

EJEMPLO

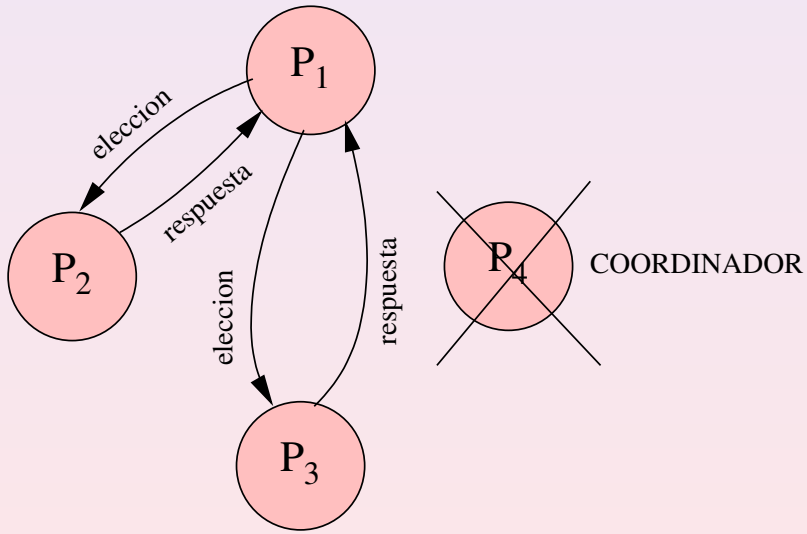
SITUACION INICIAL



EJEMPLO

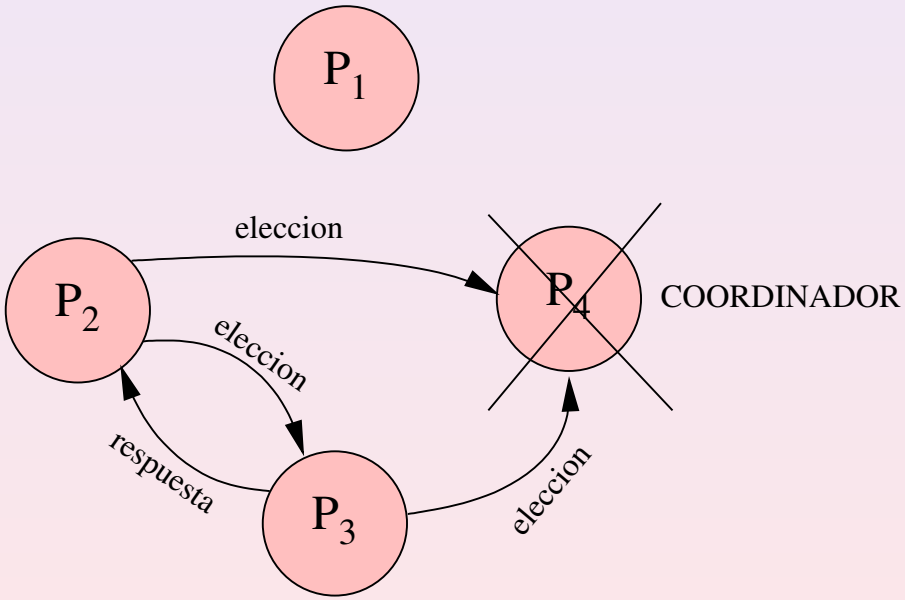
FALLA EL COORDINADOR (P4)

P1 DETECTA EL FALLO: INICIA ELECCION



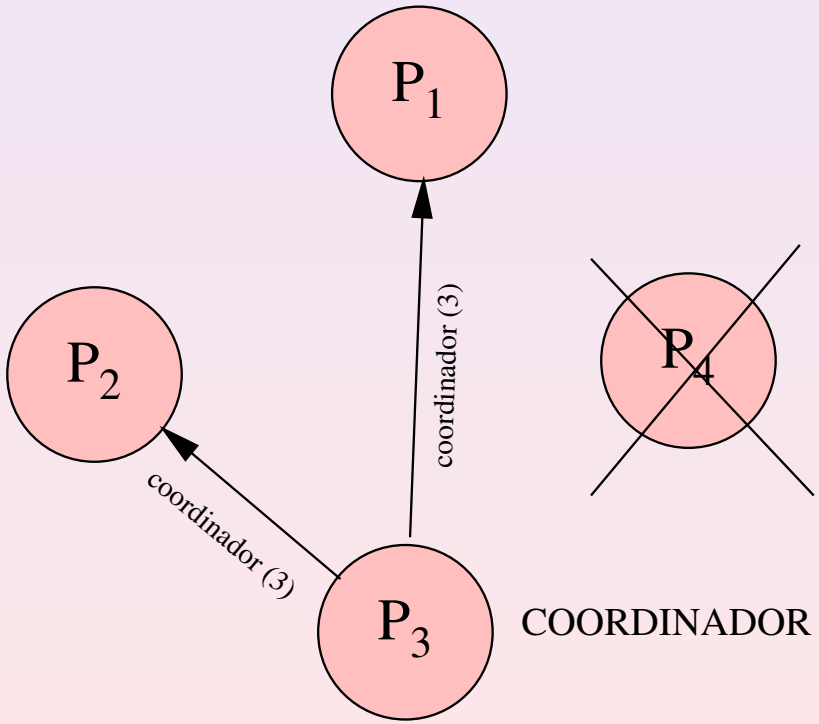
EJEMPLO

P2 Y P3 INICIAN ELECCION



EJEMPLO

P3 ES EL NUEVO COORDINADOR



ELECCIONES EN WIRELESS

- En redes inalámbricas no valen los algoritmos tradicionales, basados en envío fiable, y un número de procesos/nodos estable.
- Algoritmo de Vasudean (2004):
  - Permite fallos en los nodos.
  - Considera la *capacidad* de cada nodo: puede ser la duración de la batería por ejemplo.
  - Se construye un árbol de nodos para propagar los mensajes por la red.

    Se basa en proximidad física (nodos vecinos).
  - Se elige el que mejor capacidad tenga en ese momento.

ELECCIONES EN WIRELESS

- En redes inalámbricas no valen los algoritmos tradicionales, basados en envío fiable, y un número de procesos/nodos estable.
- Algoritmo de Vasudean (2004):
  - Permite fallos en los nodos.
  - Considera la *capacidad* de cada nodo: puede ser la duración de la batería por ejemplo.
  - Se construye un árbol de nodos para propagar los mensajes por la red.

    Se basa en proximidad física (nodos vecinos).
  - Se elige el que mejor capacidad tenga en ese momento.

ELECCIONES EN WIRELESS

- En computación inalámbrica, y específicamente en el caso de la computación móvil, tenemos una alta volatilidad en las comunicaciones y participantes.
- En ciertas aplicaciones necesitamos disponer de un coordinador, que por ejemplo procese los datos recogidos por los demás y los suba a la nube (*fog computing*).
- Cada nodo “conoce” un cierto número de vecinos (están en su rango de alcance).

  No todos ven a todos de forma directa.

ELECCIONES EN WIRELESS

- Iniciador (Raíz): Envía mensaje de *Elección* a sus vecinos inmediatos.
- Un nodo que recibe por primera vez el mensaje de elección:
  - Designa al emisor como *Padre*.
  - Envía mensaje de *Elección* a sus vecinos inmediatos, excepto al emisor. No envía ACK a su padre.
- Un nodo recibe de nuevo un mensaje de elección:
  - Se limita a un acuse de recibo vacío al emisor.

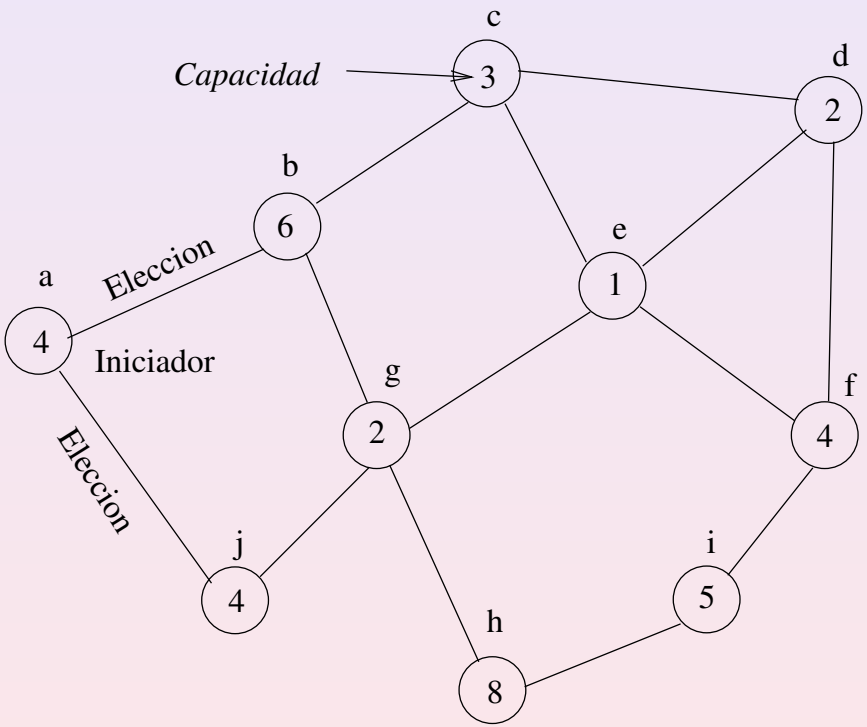
ELECCIONES EN WIRELESS

- Cuando un nodo ha recibido todos los ACK de sus vecinos inmediatos:
  - Envía a su padre el ACK, con la información de capacidad máxima.
  - Información de capacidad máxima: máximo de su propio valor actual y de la información recibida de sus hijos.
- OBSERVACION: Un nodo es *hoja* cuando recibe el mensaje por primera vez, y todos sus vecinos ya habían recibido el mensaje (ya tienen padre).

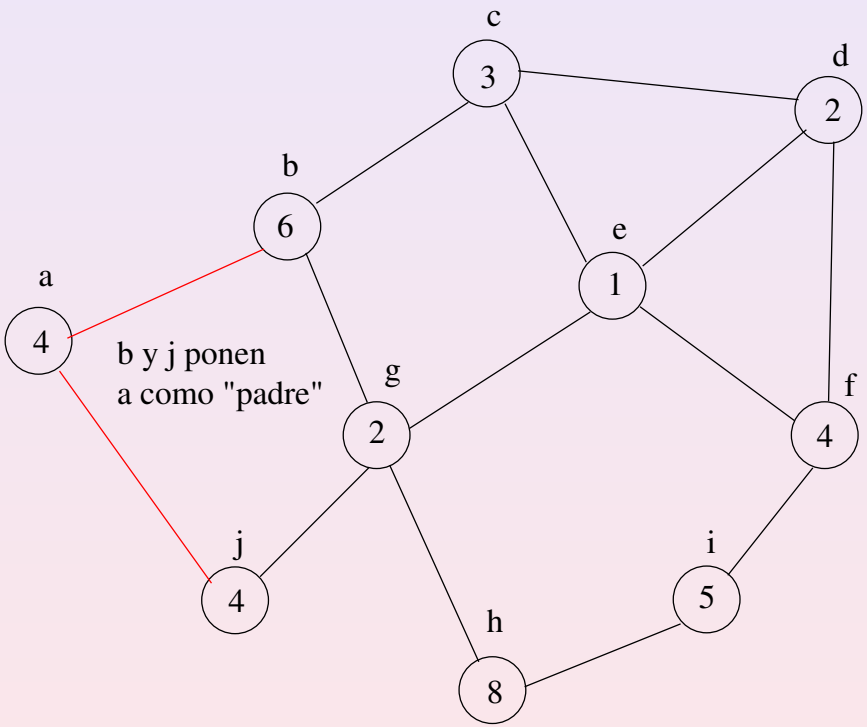
ELECCIONES EN WIRELESS

- Cuando un nodo ha recibido todos los ACK de sus vecinos inmediatos:
  - Envía a su padre el ACK, con la información de capacidad máxima.
  - Información de capacidad máxima: máximo de su propio valor actual y de la información recibida de sus hijos.
- OBSERVACION: Un nodo es *hoja* cuando recibe el mensaje por primera vez, y todos sus vecinos ya habían recibido el mensaje (ya tienen padre).

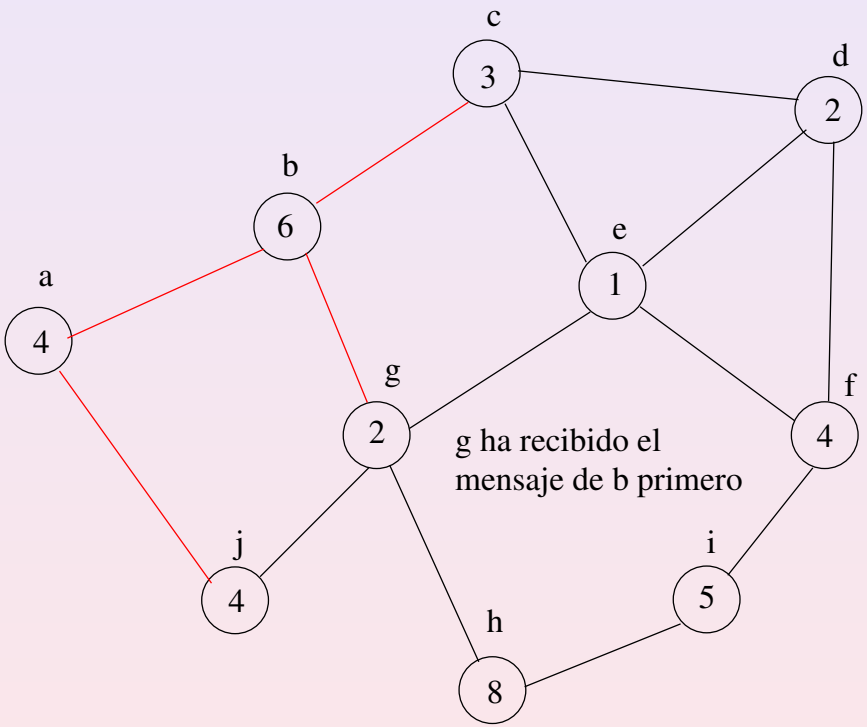
EJEMPLO


c
Capacidad
3
d
2
b
6
e
1
a
Eleccion
4
Iniciador
g
2
f
4
Eleccion
j
4
i
5
h
8

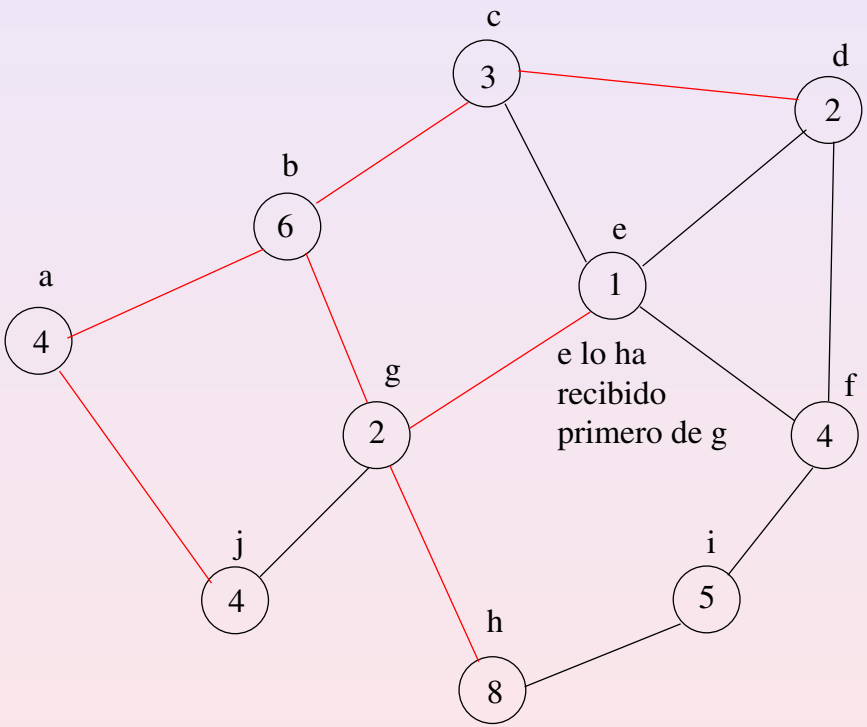
EJEMPLO



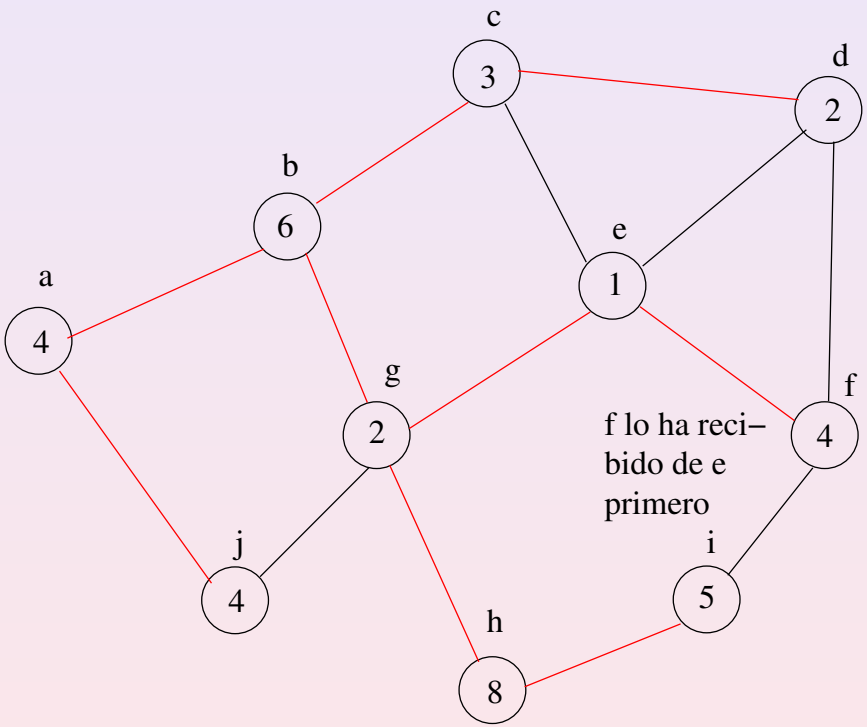
EJEMPLO


c
3
d
2
b
6
e
1
a
4
g
2
f
4
g ha recibido el
mensaje de b primero
j
4
i
5
h
8

EJEMPLO



EJEMPLO



EJEMPLO