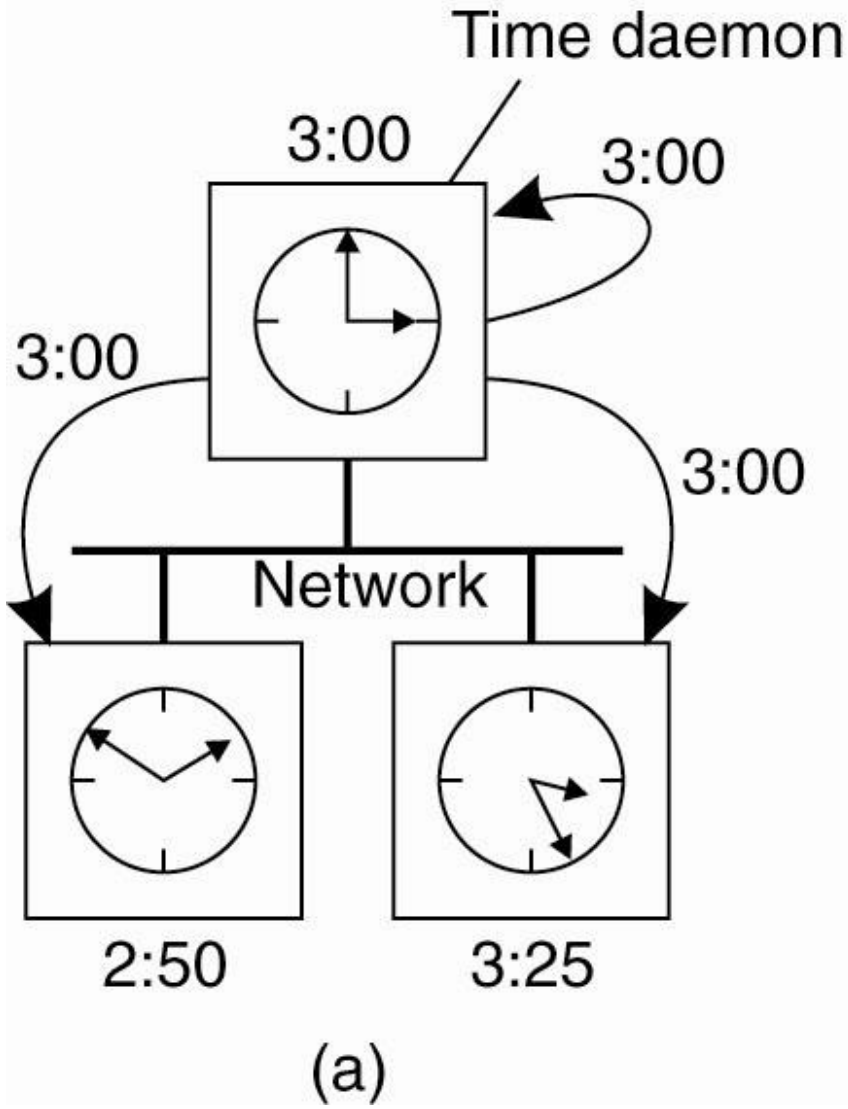
L’algoritmo di Berkeley (1)

- Nel protocollo NTP il time server si comporta come entità passiva.
- Invece in Berkely UNIX il **time server è attivo**, segnalando alle altre macchine di inviare informazioni sul tempo in modo periodico.
- Tenendo traccia delle risposte, il server **calcola un tempo medio**, comunicando alle altre macchine di accelerare o rallentare i loro orologi di conseguenza.
- Questo metodo è adatto per un sistema in cui nessuna macchina è dotata di un ricevitore WWV o di un orologio di riferimento preciso.
- L’idea alla base di questo approccio è che, per la maggior parte degli scopi, sia sufficiente che **tutte le macchine raggiungano un accordo sullo stesso valore temporale**.

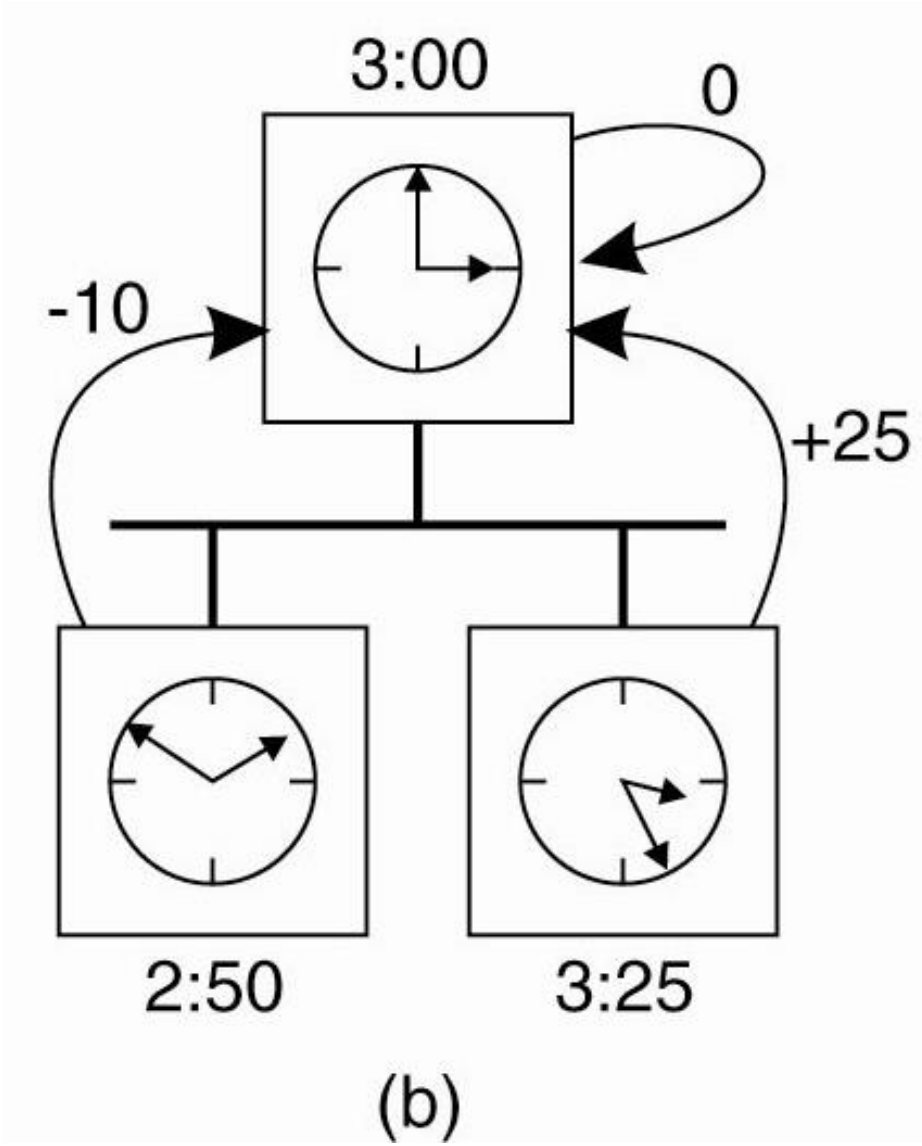
L’algoritmo di Berkeley (2)

(a) Il time daemon chiede a tutte le altre macchine informazioni sui valori dei loro orologi interni.



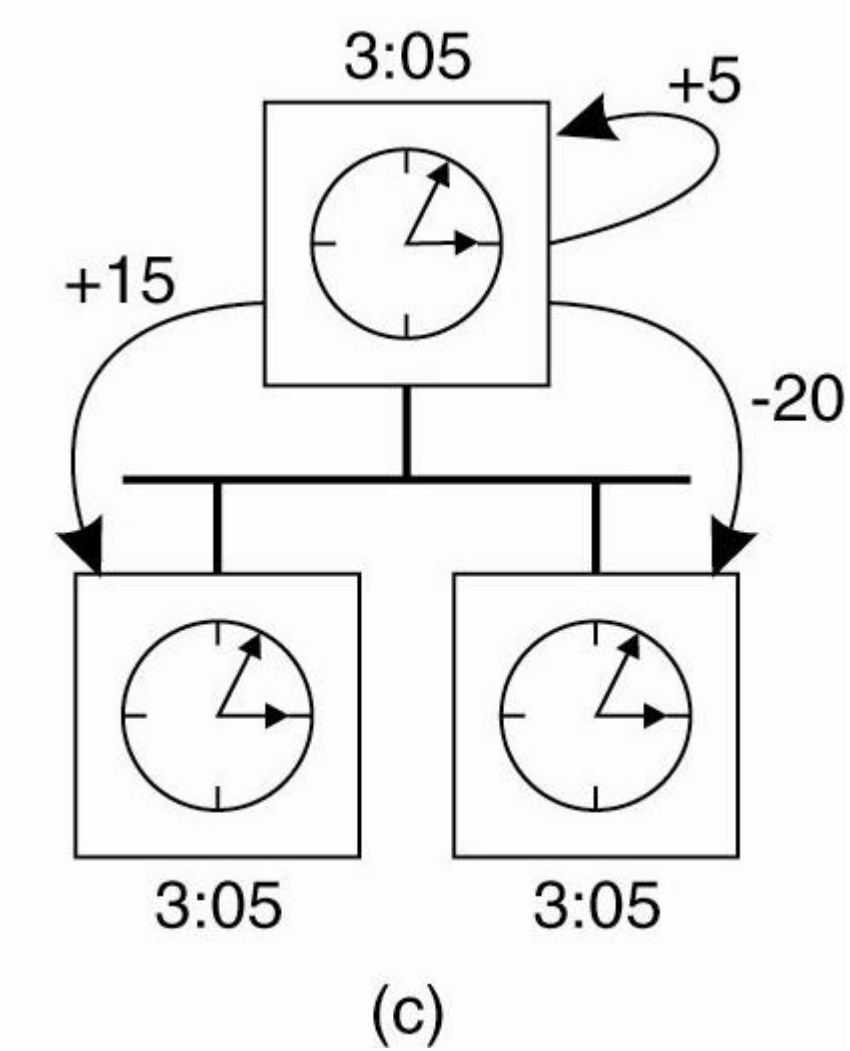
L’algoritmo di Berkeley (3)

(b) Le macchine rispondono.



L’algoritmo di Berkeley (4)

(c) Il time daemon comunica a tutti come correggere il proprio orologio interno.

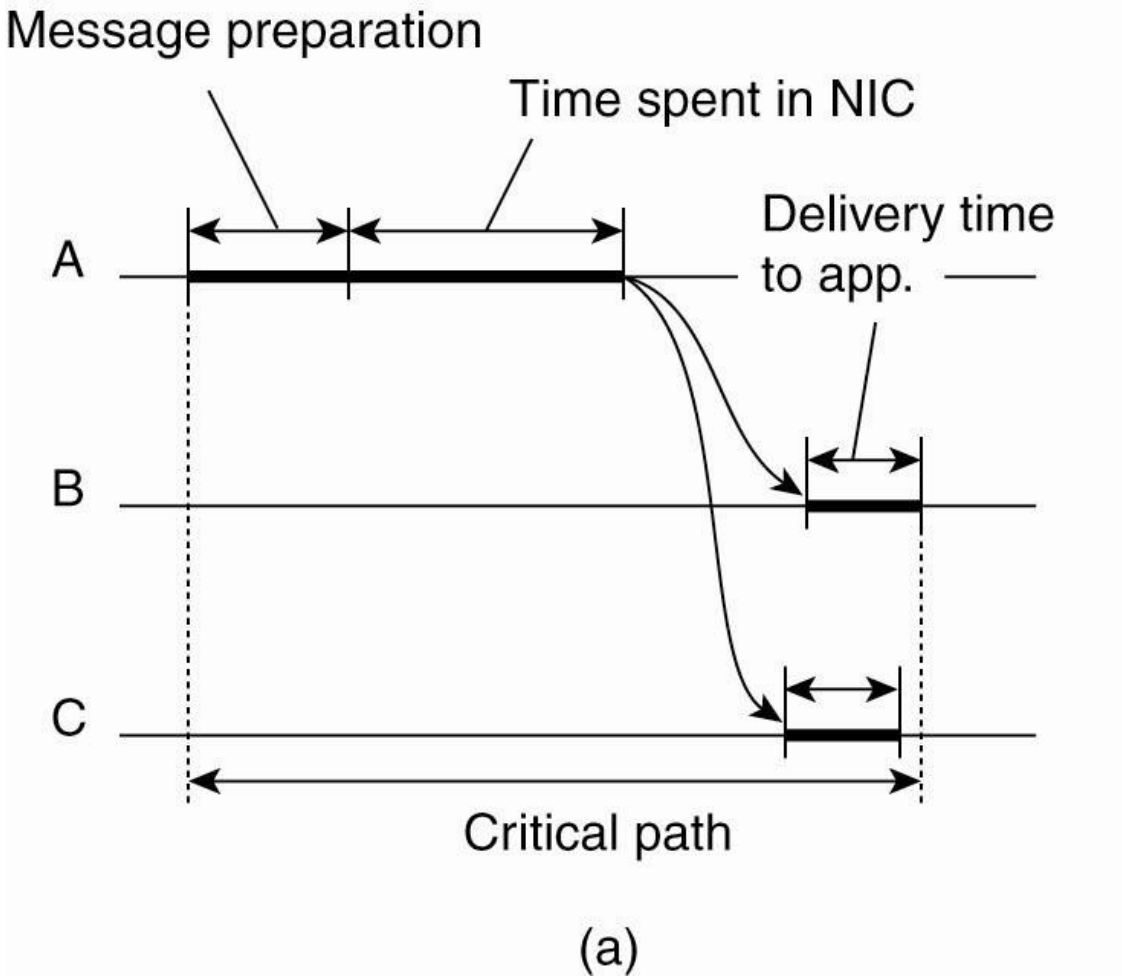


Sincronizzazione degli orologi nelle reti wireless (1)

- Nei sistemi distribuiti tradizionali, i time server possono essere dispiegati in modo facile ed efficiente.
- Inoltre, in tali sistemi, la maggior parte delle macchine è in grado di comunicare con gli altri nodi, diffondendo l'informazione in modo semplice.
- Ciò non è più vero nelle **reti wireless**, dove le **risorse** sono **limitate** ed il routing **multihop** è **costoso**.
- Quindi è necessario progettare degli algoritmi di sincronizzazione degli orologi totalmente diversi per le reti wireless.
- **Reference Broadcast Synchronization** (RBS) è un algoritmo che mira alla sincronizzazione degli orologi internamente ad una rete wireless (in modo simile all'algoritmo di Berkeley).
- RBS non è un protocollo di tipo «two-way»:
  - soltanto i ricevitori si sincronizzano;
  - chi invia viene tenuto fuori dal loop.

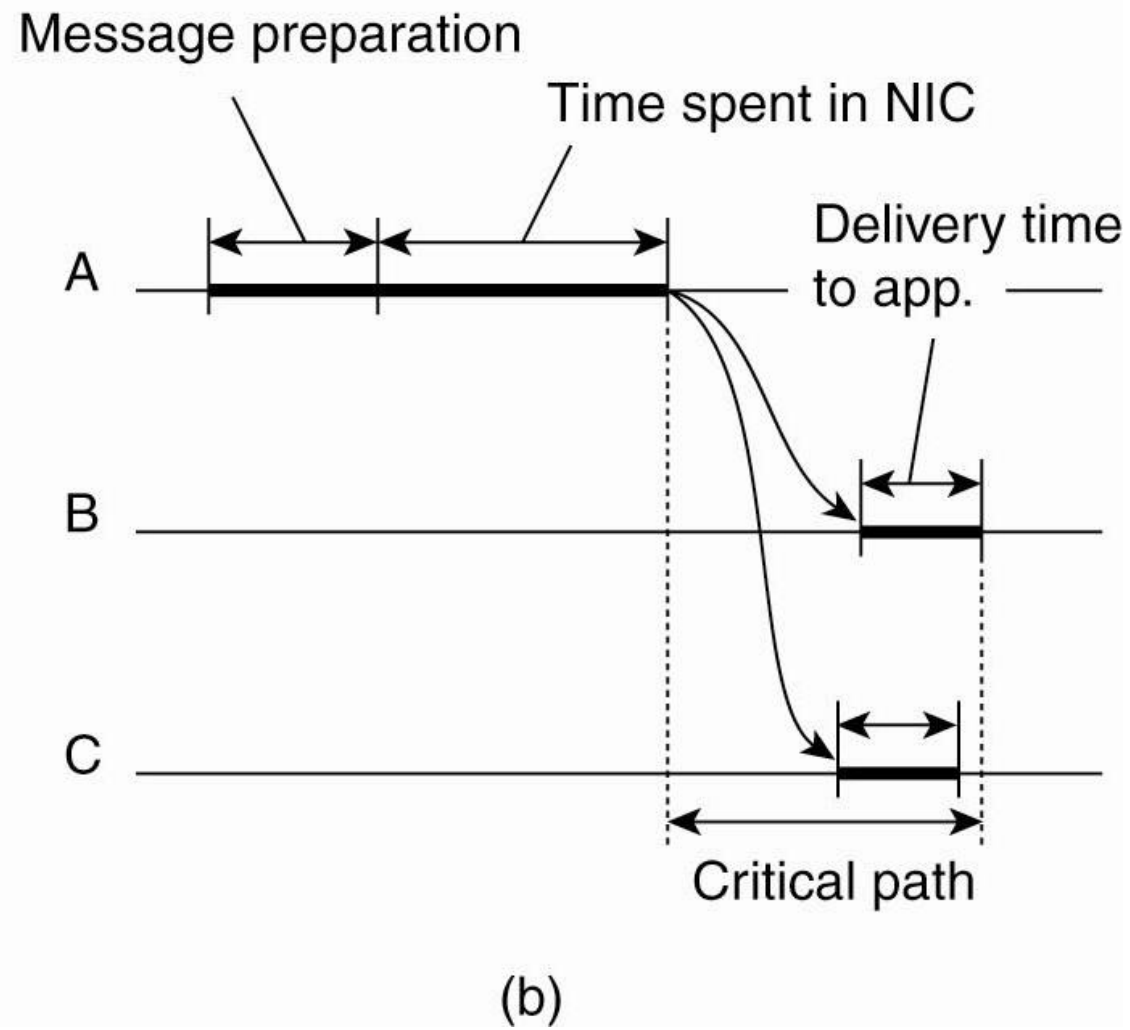
Sincronizzazione degli orologi nelle reti wireless (2)

(a) Il «critical path» tradizionale nel calcolo dei ritardi di rete.



Sincronizzazione degli orologi nelle reti wireless (3)

(b) Il «critical path» nel caso di RBS.



Sincronizzazione degli orologi nelle reti wireless (4)

- RBS funziona come segue:
  - quando un nodo comunica in broadcast un messaggio di riferimento m, ogni nodo p memorizza il valore $T_{p,m}$ (i.e., l'istante in cui ha ricevuto m);
  - quindi, p e q possono scambiarsi l'uno con l'altro i tempi di consegna per stimare i rispettivi offset (M è il numero totale di messaggi di riferimento inviati):

$$\text{Offset}[p,q] = \frac{\sum_{k=1}^{M} (T_{p,k} - T_{q,k})}{M}$$

  - così, il nodo p sarà a conoscenza del valore dell'orologio di q, in relazione al proprio;
  - tali offset vengono semplicemente memorizzati: l'orologio interno non viene corretto (per risparmiare energia).

Orologi logici

- Il concetto di causalità fra eventi è fondamentale nella progettazione e nell'analisi dei sistemi di calcolo distribuiti.
- Di solito si tiene traccia della causalità, usando il tempo fisico.
- Tuttavia, nei sistemi distribuiti, non sempre è possibile avere a disposizione una nozione globale di tempo fisico.
- In quanto asincrone **le computazioni distribuite progrediscono in modo discreto**: il tempo logico è quindi sufficiente a modellare la fondamentale **proprietà di monotonia** associata alla nozione di **causalità** in tali sistemi.
- La conoscenza della relazione di precedenza causale fra gli eventi dei processi aiuta a risolvere molti problemi, come la progettazione degli algoritmi distribuiti, tenendo traccia della dipendenza degli eventi, della conoscenza sullo stato di avanzamento della computazione e del livello di concorrenza.
- In tali casi si parla degli orologi come di **orologi logici**.

La relazione "happens-before" di Lamport

- La relazione "*happens-before*" → può essere osservata direttamente in due situazioni:
- Se *a* e *b* sono due eventi dello **stesso processo**, ed *a* avviene prima di *b*, allora *a* → *b* è vera.
- Se *a* è l'evento relativo all'**invio di un messaggio** da parte di un processo e *b* è l'evento relativo alla **ricezione dello stesso messaggio** da parte di un altro processo, allora *a* → *b*

Un Framework per un sistema di Orologi Logici

Definizione

- Un sistema di orologi logici consiste di un dominio temporale T e di un orologio logico C.
- Gli elementi di T formano un insieme parzialmente ordinato su una relazione <.
- Intuitivamente, la relazione < è analoga alla relazione "*earlier than*" fornita dai sistemi temporali fisici.
- L'orologio logico C è una funzione che mappa un evento e in un sistema distribuito in un elemento del dominio temporale T, denotato come C(e) e detto il timestamp di e, definito come segue:
  - C : H → T tale che:
  - dati due eventi e_i ed e_j, $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$.
- Tale proprietà di monotonia è anche detta **condizione di coerenza dell'orologio**.
- Quando T e C soddisfano la seguente condizione,
  - dati due eventi e_i ed e_j, $e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$

  il sistema degli orologi è detto **fortemente coerente**.

Implementazione degli Orologi Logici

- L'implementazione di un sistema di orologi logici richiede:
  - delle **strutture dati** locali per ogni processo per la rappresentazione del tempo logico ed un protocollo per l'aggiornamento delle strutture dati che assicuri la condizione di coerenza.
- Ogni processo p_i mantiene le strutture dati che gli conferiscono quanto segue:
  - un **orologio logico locale**, denotato da lc_i, che consente al processo p_i di misurare il suo stato di avanzamento;
  - un **orologio logico globale**, denotato da gc_i, che rappresenta l'interpretazione locale che il processo p_i ha del tempo logico globale. Solitamente, lc_i è una componente di gc_i.
- Il protocollo assicura che l'orologio logico del processo, e quindi la sua rappresentazione del tempo globale, sia gestito in modo coerente per mezzo delle seguenti regole:
  - **R1**: determina come un processo aggiorna l'**orologio logico locale** nel momento in cui esegue un evento;
  - **R2**: determina come un processo aggiorna l'**orologio logico globale** per aggiornare la sua interpretazione del tempo globale e dello stato di avanzamento globale.
- I vari sistemi di orologi logici differiscono fra loro in base alla propria rappresentazione del tempo logico ed in base al protocollo di aggiornamento degli orologi logici.

Gli orologi logici di Lamport (1)

- Sistema proposto da Lamport nel 1978 come tentativo di assicurare un ordinamento totale degli eventi in un sistema distribuito.
- Il dominio del tempo è l'insieme degli interi non negativi. L'orologio logico locale di un processo p_i e la sua **interpretazione locale del tempo globale** sono **schiacciati in una singola** variabile intera scalare C_i.
- Le regole **R1** e **R2** per aggiornare gli orologi sono le seguenti:
  - **R1**: prima di eseguire un evento, il processo p_i esegue l'operazione

    $C_i := C_i + d \quad (d > 0)$

    In generale, ogni volta che R1 viene eseguita, d può assumere un valore diverso; tuttavia, di solito d vale 1.
  - **R2**: ogni messaggio riporta in coda il valore dell'orologio del mittente registrato al momento dell'invio. Quando un processo p_i riceve un messaggio con timestamp C_{msg} , esegue le seguenti operazioni:

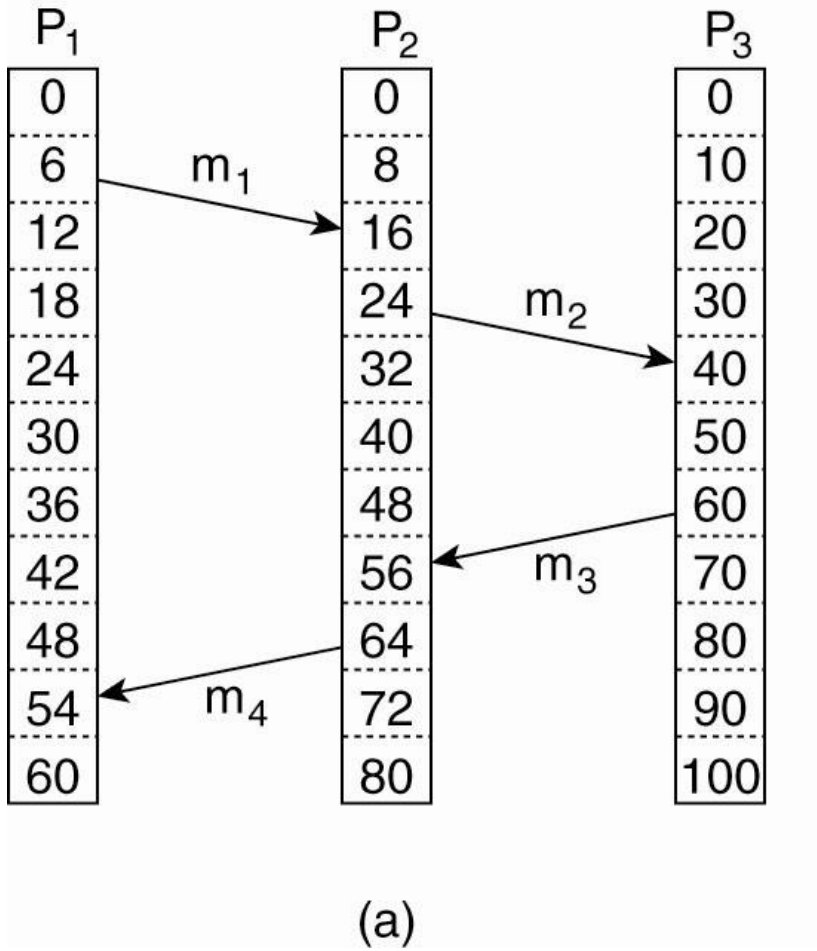
    $C_i := \max(C_i, C_{msg})$,

    esegue R1,

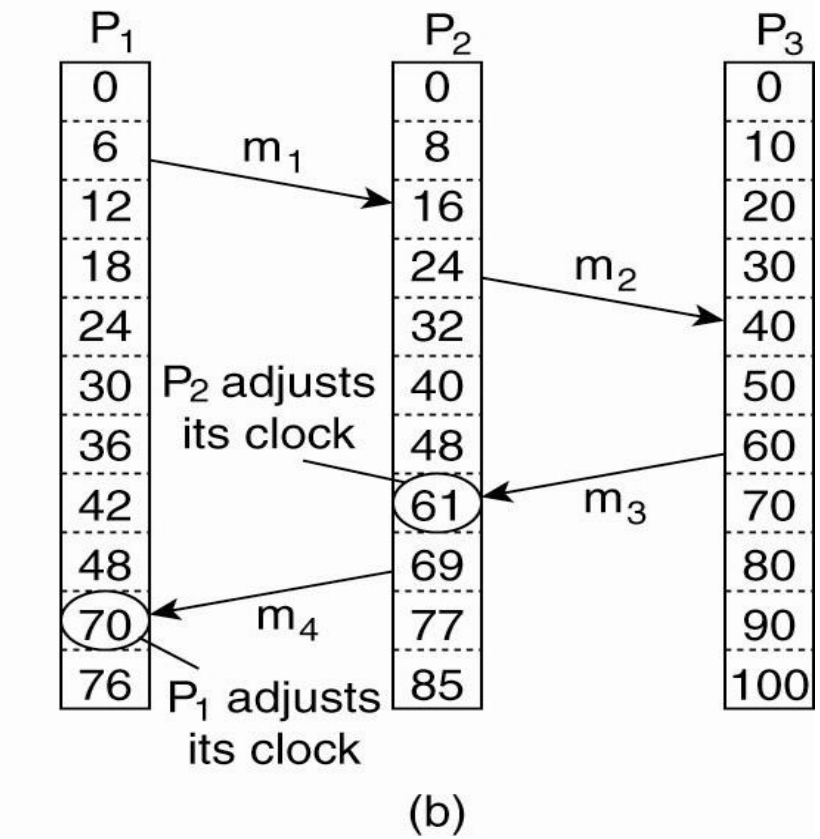
    consegna il messaggio.

Gli orologi logici di Lamport (2)

(a) Tre processi, ognuno con il proprio orologio.
Gli orologi funzionano con velocità differenti.



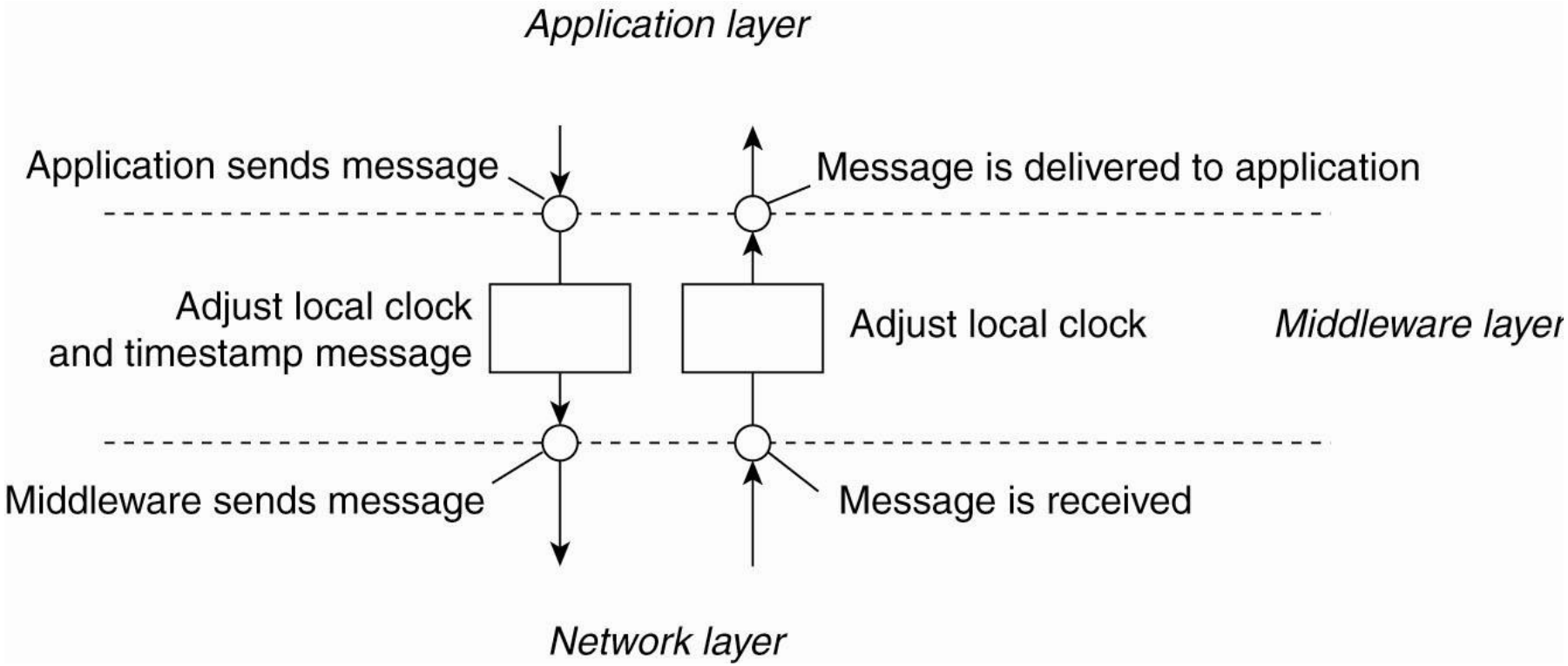
Gli orologi logici di Lamport (3)



(b) L'algoritmo di Lamport corregge gli orologi.

Gli orologi logici di Lamport (4)

Il posizionamento degli orologi logici di Lamport nei sistemi distribuiti.



Proprietà di base

Proprietà di coerenza

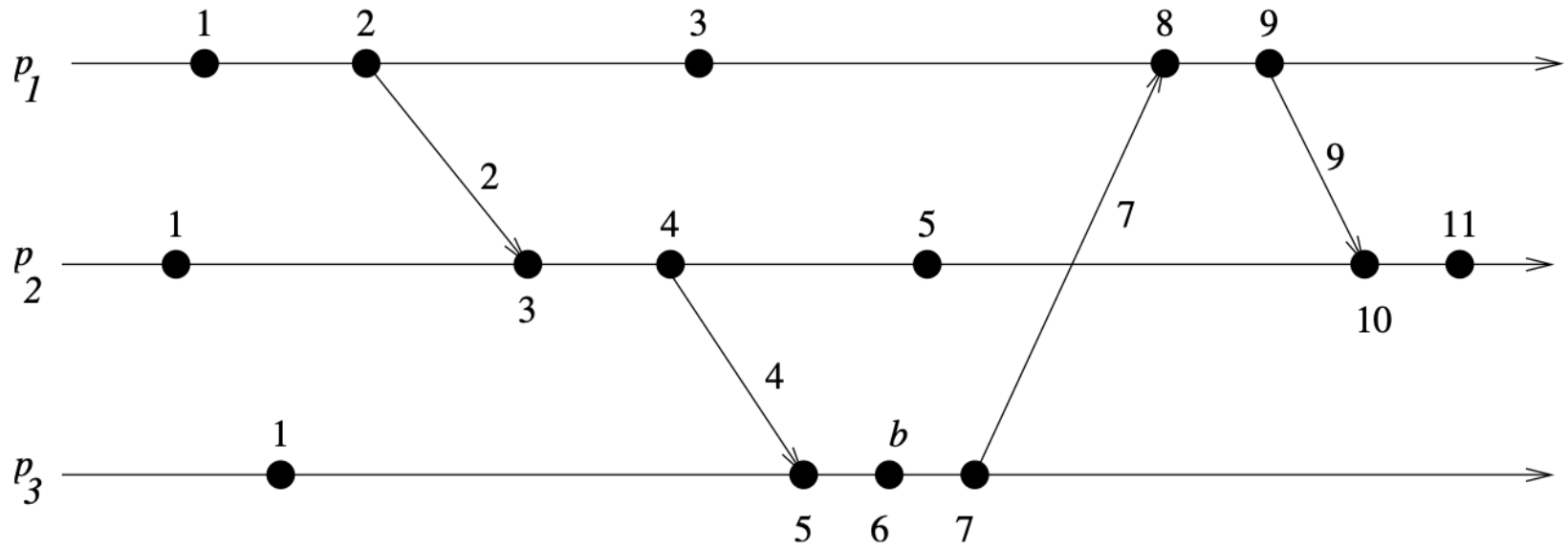
- Gli orologi scalari soddisfano la proprietà di monotonia e quindi anche quella di coerenza: dati due eventi e_i ed e_j, $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$.

Ordinamento totale

- Gli orologi scalari possono essere utilizzati per **ordinare totalmente gli eventi** in un sistema distribuito.
- Il problema principale nell'ordinamento totale degli eventi è che due o più di questi **possono avere gli stessi timestamp**, in processi differenti.

Ordinamento totale (1)

Il terzo evento del processo p_1 ed il secondo evento del processo p_2 hanno gli stessi valori scalari di timestamp.



Ordinamento totale (2)

Per tali eventi è necessaria una procedura di decisione per superare l'ambiguità:

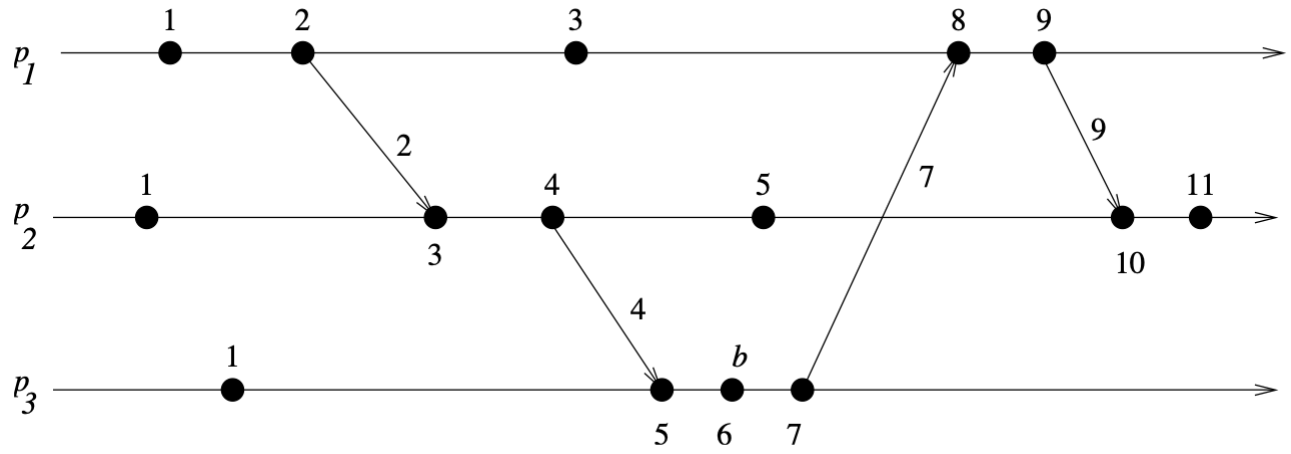
- Gli **identificatori dei processi** vengono **ordinati** linearmente e le ambiguità relative agli eventi con lo stesso valore scalare di timestamp vengono risolte in base ai valori degli identificatori dei processi.
- Più basso è il valore dell'identificatore nell'ordinamento, maggiore è la priorità.
- Il timestamp di un evento è denotato da una coppia (t, i) dove t è il suo istante temporale ed i l'identità del processo in cui si è verificato.
- La relazione di ordinamento totale $\prec$ su due eventi x e y con timestamp (h,i) e (k,j), rispettivamente, è definita come segue:

$$x \prec y \Leftrightarrow (h < k \text{ oppure } (h = k \text{ e } i < j))$$

Proprietà (1)

Conteggio degli eventi

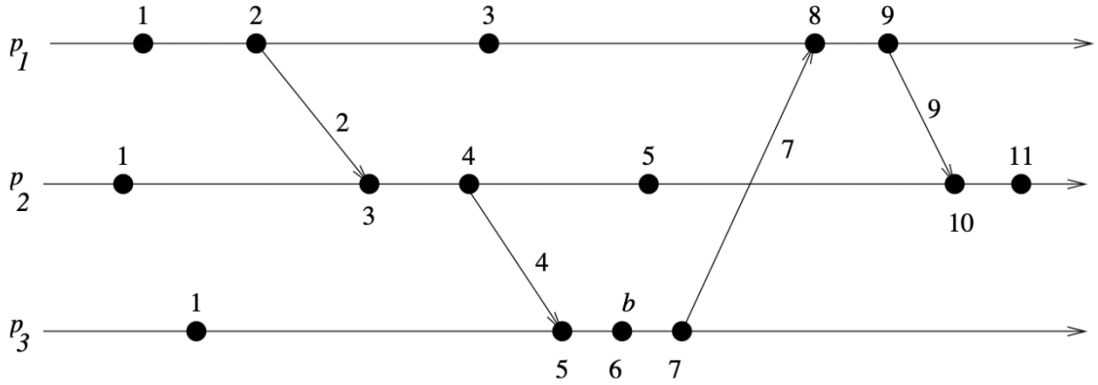
- Se il valore di incremento d è sempre 1, il tempo scalare gode della seguente proprietà: se l'evento e ha il timestamp h, allora h-1 rappresenta la durata logica minima, misurata come numero di eventi, necessaria prima che venga prodotto l'evento e;
- Denominiamo h-1 «altezza dell'evento e».
- In altri termini, h-1 eventi sono stati prodotti in modo sequenziale prima dell'evento e, senza distinzione su quali processi abbiano prodotto tali eventi.
- Per esempio, nel diagramma seguente, cinque eventi precedono l'evento b lungo il più lungo percorso causale che termina in b.



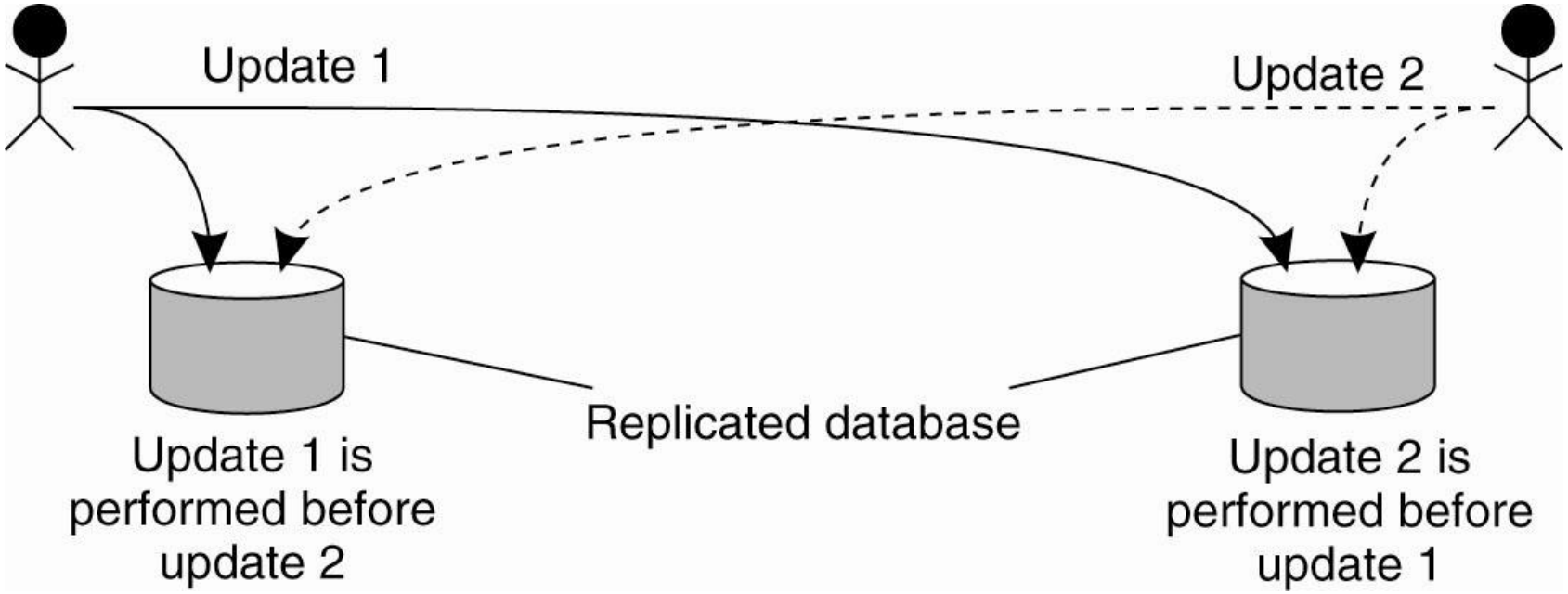
Proprietà(2)

Assenza di corenza forte

- Il sistema di orologi scalari non è fortemente coerente, ovvero, non è vero che per tutti gli eventi e_i and e_j, $C(e_i) < C(e_j) \Rightarrow e_i \rightarrow e_j$.
- Ad esempio, nel diagramma in basso, il terzo evento del processo p_1 ha un timestamp scalare più piccolo del terzo evento del processo p_2. Tuttavia, il primo non è avvenuto prima del secondo.
- Ciò è dovuto al fatto che l'orologio logico locale e quello globale del processo sono «schiacciati» in un unico valore, provocando la **perdita dell'informazione di dipendenza causale** fra eventi avvenuti in processi distinti.
- Ad esempio, nel diagramma in basso, quando il processo p_2 riceve il primo messaggio dal processo p_1, esso aggiorna il suo orologio a 3, dimenticando che il timestamp dell'ultimo evento di p_1 da cui dipende è 2.



Esempio: Multicasting Totalmente Ordinato (1)



- Aggiornare un database replicato e lasciarlo in uno stato inconsistente.

Esempio: Multicasting Totalmente Ordinato (2)

- L'esempio illustra un problema che si verifica quando due operazioni di aggiornamento devono avvenire nello **stesso ordine** presso **ogni copia** della risorsa.
- Dal punto di vista della coerenza, l'ordine delle operazioni non è importante.
- La questione di fondo è che **entrambe le copie dovrebbero essere esattamente la stessa cosa**.
- Questo tipo di situazioni è gestito per mezzo di **multicast totalmente ordinati**, i.e., un'operazione multicast tale che tutti i messaggi vengano consegnati nello stesso ordine ad ogni destinatario.

Esempio: Multicasting Totalmente Ordinato (3)

- Gli orologi logici di Lamport possono essere usati per implementare **multicast totalmente ordinati** in modo distribuito, date le seguenti premesse:
  - ogni messaggio è corredato da un timestamp con il tempo logico corrente del mittente;
  - quando un messaggio è inviato in multicast, viene inviato anche al mittente;
  - i messaggi dello stesso mittente vengono ricevuti nell'ordine in cui sono stati inviati e nessun messaggio è andato perso.
- Quando un processo riceve un messaggio, lo memorizza in una **coda locale** ordinata in base ai valori dei timestamp.
- Il ricevente invia un **acknowledgment** a **tutti** gli altri **processi**.
- Seguendo l'algoritmo di Lamport, tutti i processi alla fine avranno gli **stessi contenuti** nella **coda locale**.
- Un processo può consegnare un messaggio accodato soltanto quando si trova in testa alla coda ed è stato riconosciuto (**acknowledged**) da tutti gli altri processi.
- Dato che tutti i processi hanno la stessa coda, tutti i messaggi sono consegnati nello stesso ordine ovunque nel sistema.

Vector Clock (1)

Denotazione:

- **$T_{snd}(m)$** l'istante in cui m viene inviato,
- **$T_{rcv}(m)$** l'istante in cui m viene ricevuto.

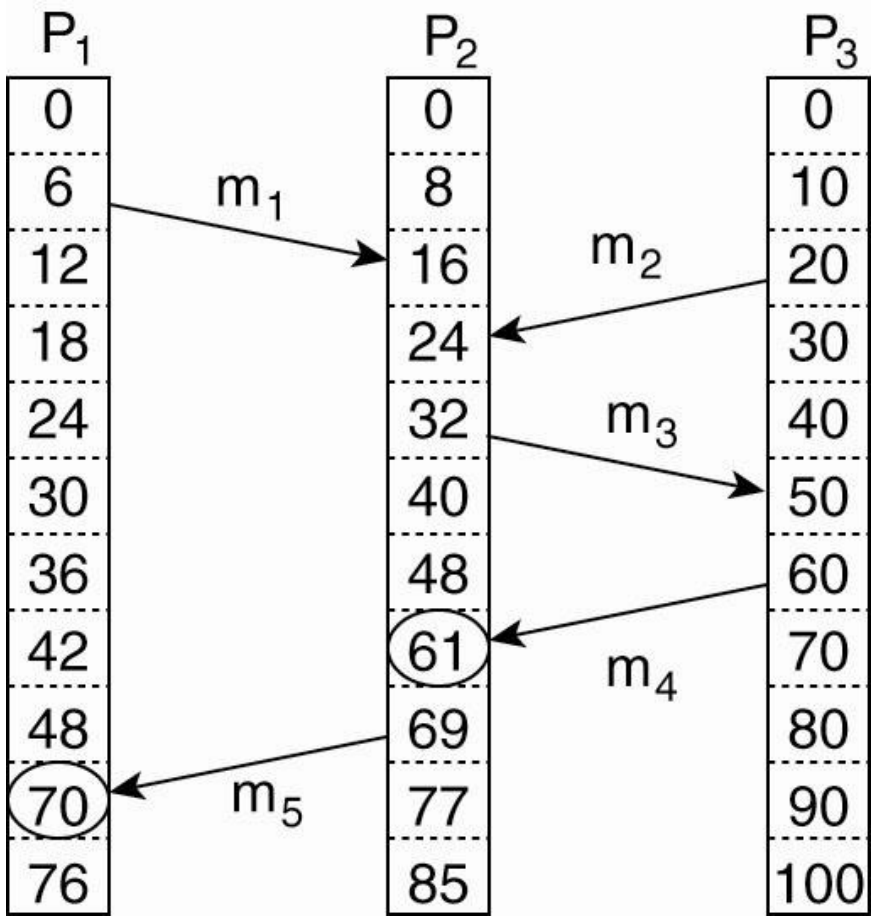
Per costruzione sappiamo che, per ogni messaggio m_i, **$T_{rcv}(m_i) > T_{snd}(m_i)$**.

Ma cosa possiamo dire a proposito di

$T_{rcv}(m_i) < T_{snd}(m_j)$?

Nella figura, se $m_i=m_1$ e $m_j=m_3$, sappiamo che gli eventi si sono verificati in P_2, ovvero, che l'**invio** di **m_3 dipende** dalla **ricezione** di **m_1**.

Tuttavia, sappiamo anche che **$T_{rcv}(m_1) < T_{snd}(m_2)$**, ma che l'**invio** di **m_2** non ha **nulla a che vedere** con la **ricezione** di **m_1**.



- Trasmissioni di messaggi concorrenti con orologi logici.

Vector Clock (2)

- Quindi, gli orologi di Lamport **non catturano la nozione di causalità**.
- Al contrario, **la causalità può essere rappresentata con i vector clock**.
- Le nozioni di tempo locale e rappresentazione locale del tempo globale non sono più «schiacciate» in un singolo valore scalare.
- Ogni processo mantiene una **struttura dati vettoriale** in cui memorizza:
  - il proprio tempo locale,
  - la rappresentazione locale del tempo locale di ogni altro processo del sistema.

Vector Time (1)

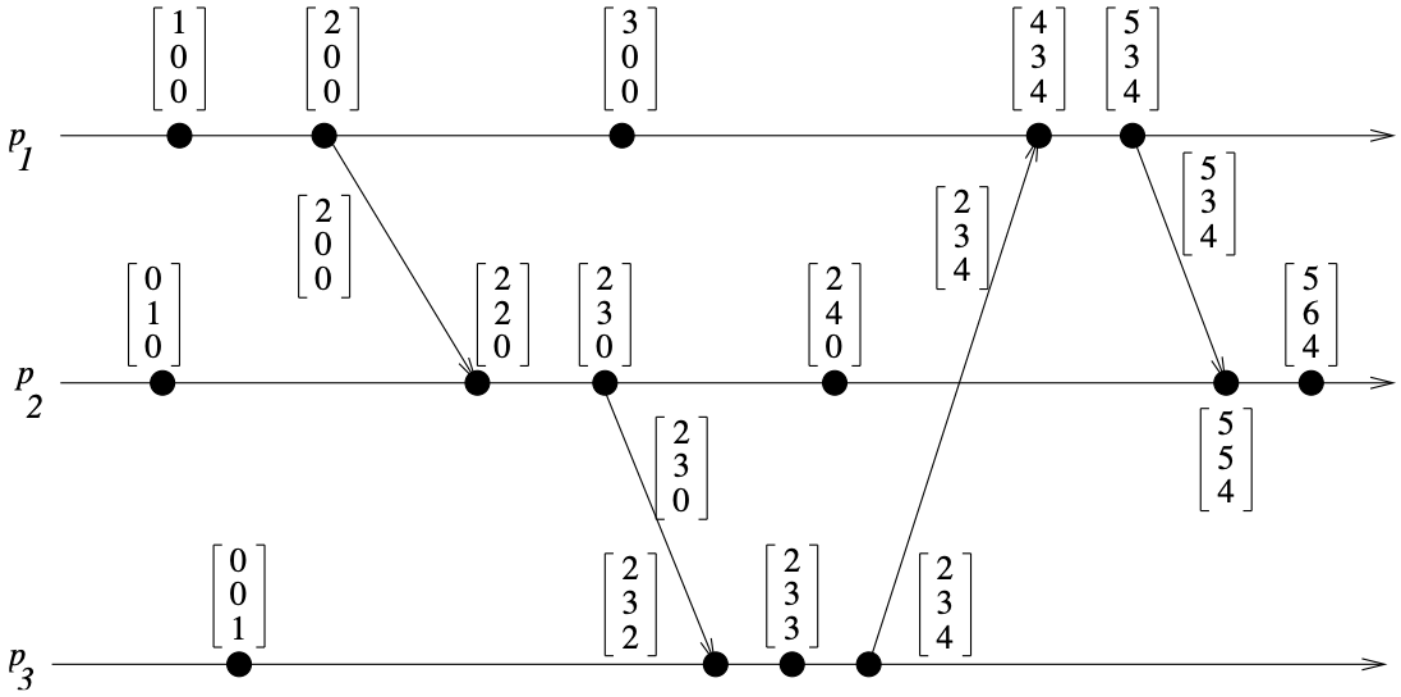
- Il sistema vector clock fu ideato in modo indipendente da Fidge, Mattern e Schmuck.
- Nei sistemi vector clock, il **dominio del tempo** è rappresentato da un insieme di **vettori n-dimensionali di interi non-negativi**.
- Ogni processo p_i utilizza un vettore $vt_i[1..n]$, dove $vt_i[i]$ è il valore dell'orologio logico locale del processo p_i e descrive lo stato di avanzamento del processo p_i.
- $vt_i[j]$ rappresenta **la più recente informazione** del processo p_i's sul tempo locale di p_j.
- Se $vt_i[j]=x$, allora il processo p_i sa che il tempo locale del processo p_j è avanzato fino al valore x.
- L'intero vettore vt_i costituisce **la rappresentazione di p_i** del tempo logico **globale** ed è utilizzato per produrre i timestamp degli eventi.

Vector Time (2)

- Il processo p_i usa le seguenti versioni delle regole R1 e R2 per aggiornare il proprio clock:
  - R1: prima di eseguire un evento, il processo p_i aggiorna il proprio tempo logico locale come segue:
    - $vt_i[i]:=vt_i[i]+d \quad (d>0)$
  - R2: ogni messaggio m è **contrassegnato** con il **vector clock** vt del **processo mittente** al momento dell'**invio**. Al momento della ricezione di tale messaggio (m,vt), il processo p_i esegue la seguente sequenza di azioni:
    - Aggiorna la propria rappresentazione del tempo logico globale come segue:
      $1 \leq k \leq n : vt_i[k] := \max(vt_i[k],vt[k])$
    - Esegue R1.
    - Consegna il messaggio m.

Vector Time (3)

- Il timestamp di un evento è il valore del vector clock del suo processo quando l'evento viene eseguito.
- La figura illustra un esempio dell'aggiornamento dei vector clock con il valore di incremento d=1.
- Inizialmente ogni vector clock vale [0, 0, 0, …., 0].



Vector Time (4)

- **Confronto dei vettori di timestamp**
- Vengono introdotte le seguenti relazioni per consentire la comparazione di due vettori di timestamp, vh e vk:

$$
\begin{aligned}
vh = vk &\Leftrightarrow \forall x : vh[x] = vk[x] \\
vh \leq vk &\Leftrightarrow \forall x : vh[x] \leq vk[x] \\
vh < vk &\Leftrightarrow vh \leq vk \text{ and } \exists x : vh[x] < vk[x] \\
vh \parallel vk &\Leftrightarrow \neg(vh < vk) \wedge \neg(vk < vh)
\end{aligned}
$$

- Se il processo in cui si verifica l'evento è noto, il test per confrontare i due timestamp può essere semplificato come segue: se gli eventi x e y, rispettivamente verificatisi nei processi p_i e p_j, hanno timestamp associati vh e vk, rispettivamente, allora

$$
\begin{aligned}
x \rightarrow y &\Leftrightarrow vh[i] \leq vk[i] \\
x \parallel y &\Leftrightarrow vh[i] > vk[i] \wedge vh[j] < vk[j]
\end{aligned}
$$

Proprietà del Vector Time (1)

- **Isomorfismo**
- Se gli eventi di un sistema distribuito sono contrassegnati usando un sistema di vector clock, vale la seguente proprietà. Se due eventi x e y hanno timestamp vh and vk, rispettivamente, allora

$$x \rightarrow y \quad \Leftrightarrow \quad vh < vk$$

$$x \parallel y \quad \Leftrightarrow \quad vh \parallel vk.$$

- Quindi, esiste un **isomorfismo** fra l'insieme degli eventi parzialmente ordinati prodotto dalla computazione distribuita sui vettori di timestamp.

Proprietà del Vector Time (2)

- **Coerenza forte**
- Il sistema dei vector clock è fortemente coerente; quindi, esaminando il vettore di timestamp relativamente a due esempi, **si può determinare** se i due eventi siano **causalmente collegati**.
- Tuttavia, Charron e Bost hanno dimostrato che **la dimensione dei vector clock** non può essere meno di n, ovvero, del numero totale di processi coinvolti nella computazione, affinché la proprietà valga.
- **Conteggio degli eventi**
- Se d=1 (nella regola R1), allora l'esimo componente i^{th} del vector clock del processo p_i, $vt_i[i]$, denota il **numbero di eventi** che si sono verificati in p_i fino a quell'istante.
- Quindi, un evento e rappresenta il numero di eventi eseguiti dal processo p_j che **causalmente precedono** e. Chiaramente, $\sum vh[j] - 1$ rappresenta il numero totale di eventi che causalmente precedono e nell'intera computazione distribuita.

Garantire la Causalità nella Comunicazione (1)

- I Vector clock consentono ad un sistema di implementare un **multicasting ordinato in base alla relazione di causalità**.
- Ciò rappresenta una nozione **più debole** rispetto al multicasting totalmente ordinato:
  - se due messaggi non sono in relazione, l'ordine in cui vengono consegnati alle applicazioni non è importante.
- Assumiamo che gli **orologi** vengano **aggiornati** soltanto all'**invio** ed alla **ricezione** dei messagi.
- Se P_j riceve un messaggio m da P_i con vector timestamp ts(m), allora esso verrà consegnato allo strato applicativo quando le seguenti condizioni saranno soddisfatte:
  - $ts(m)[i] = vt_j[i]+1$
  - $ts(m)[k] \leq vt_j[k]$ per ogni $k \neq i$

Garantire la Causalità nella Comunicazione (2)

- Comunicazione causale: