SISTEMAS DISTRIBUÍDOS

Capítulo 6 - Tempo e ordenação de eventos

NOTA PRÉVIA

A estrutura da apresentação é semelhante e utiliza algumas das figuras do livro de base do curso:

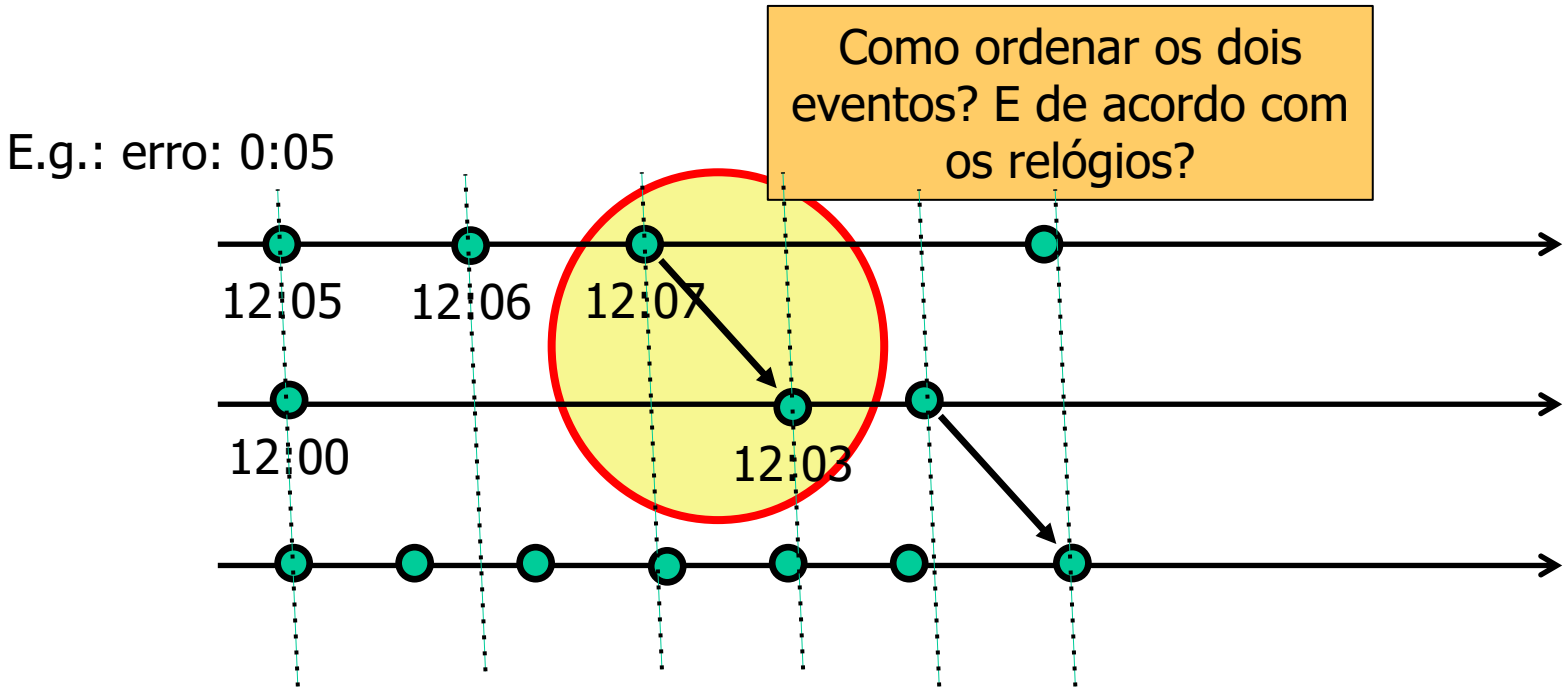
G. Coulouris, J. Dollimore, T. Kindberg, G. Blair
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition, 2011

MOTIVAÇÃO

Num sistema distribuído é impossível sincronizar os relógios de vários computadores a menos dum dado valor.

Porquê?

Assim, é impossível usar o valor do relógio em diferentes computadores para saber a ordem dos eventos.



COMO SE DEFINE "ACONTECEU ANTES" ?

Num sistema distribuído, estamos em geral interessados em conhecer as relações de causalidade, i.e., saber quais os eventos que podem ter causado outros eventos.

E.g.

Rede social

Permissões iniciais: amigos podem ver toda a informação.

Para adicionar fotografia que não se pretenda que amigo X veja:

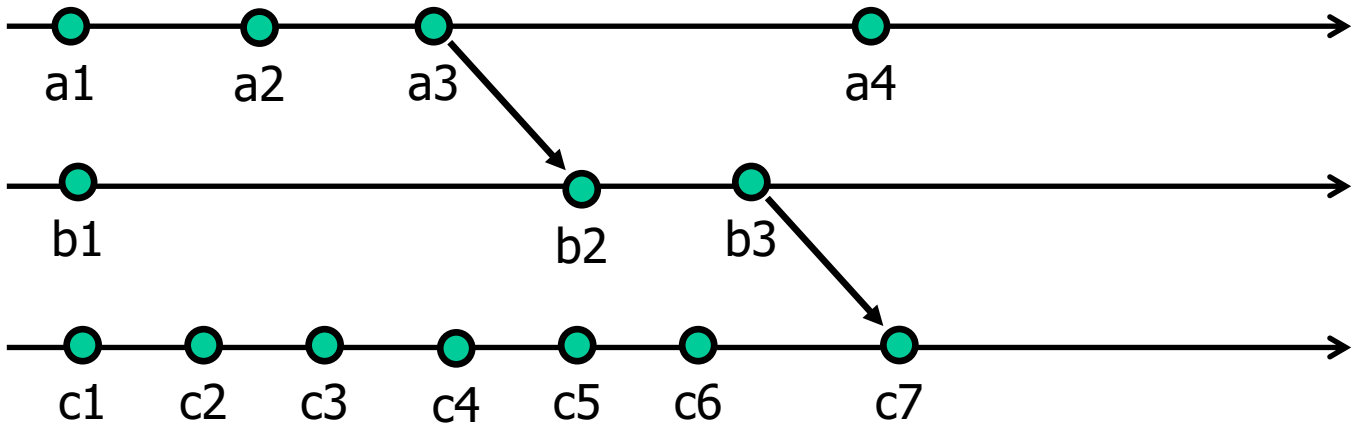
1. Remover X da lista de amigos;
2. Adicionar fotografia.

Relação de causalidade importante porque quem vir a fotografia deve ver também X removido da lista de amigos.

COMO SE DEFINE "ACONTECEU ANTES" ?

Num sistema distribuído, estamos em geral interessados em conhecer as relações de causalidade, i.e., saber quais os eventos que podem ter causado outros eventos.

A relação "aconteceu antes" capta esta propriedade: o evento e_1 aconteceu antes de e_2 se e_1 pode ter causado e_2.



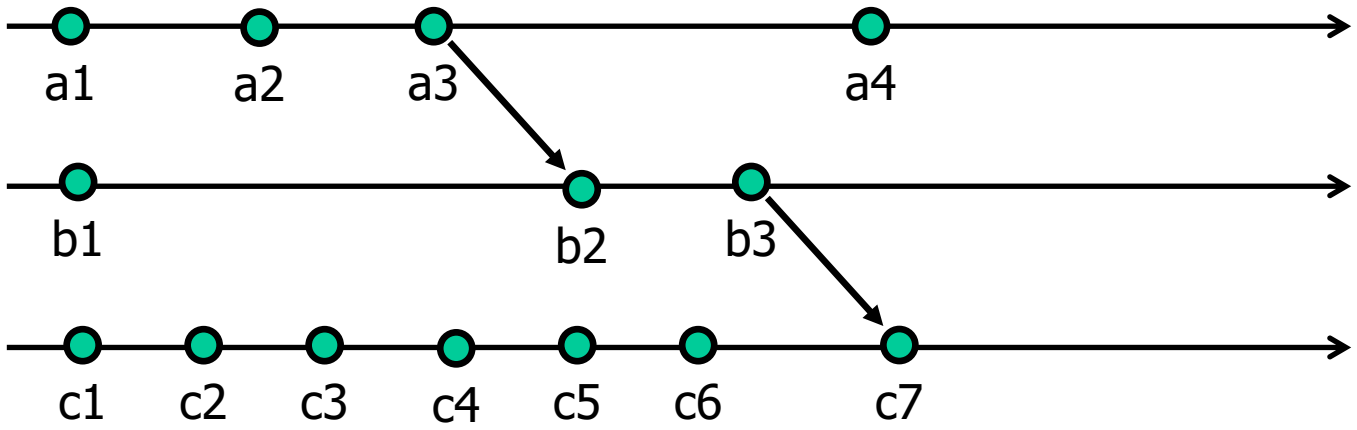
COMO SE DEFINE "ACONTECEU ANTES" ?

Num processo, a relação aconteceu antes pode-se definir pela ordem de execução dos eventos:

$a1 \rightarrow a2 \rightarrow a3 \rightarrow a4$

$b1 \rightarrow b2 \rightarrow b3$

$c1 \rightarrow c2 \rightarrow c3 \rightarrow c4 \rightarrow c5 \rightarrow c6 \rightarrow c7$

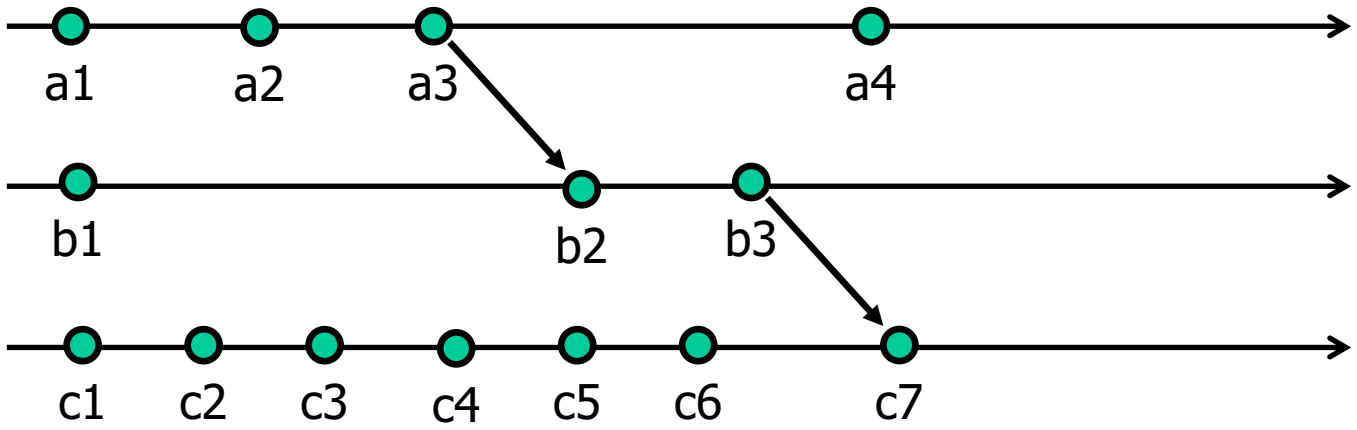


COMO SE DEFINE "ACONTECEU ANTES" ? (CONT.)

Quando um processo envia uma mensagem a outro processo, o envio acontece necessariamente antes da recepção:

$a3 \rightarrow b2$

$b3 \rightarrow c7$



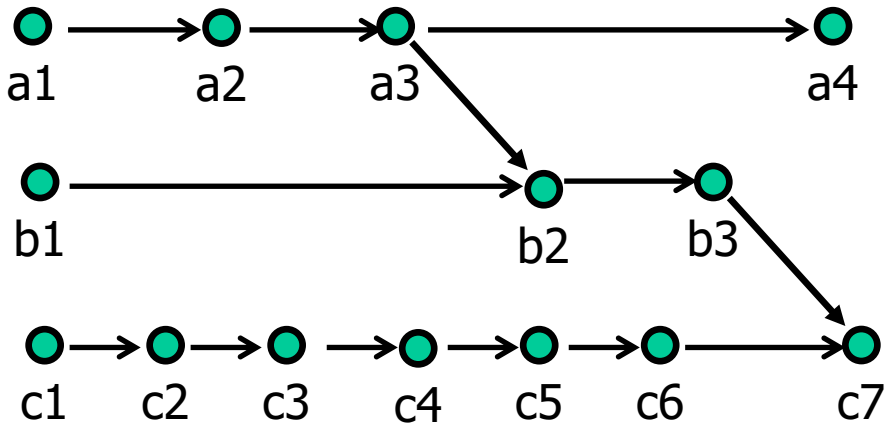
COMO SE DEFINE "ACONTECEU ANTES" ? (CONT.)

A relação *aconteceu antes* é transitiva:

$x1 \rightarrow x2$ e $x2 \rightarrow x3$ => $x1 \rightarrow x3$

Dois eventos e1, e2 dizem-se concorrentes (e1||e2) se
$\neg$ $e1 \rightarrow e2$ e $\neg$ $e2 \rightarrow e1$.

a1||b1 , a1||c1, a1||c2, …



DEFINIÇÃO: RELAÇÃO *ACONTECEU ANTES* (LAMPORT 1978)

A relação aconteceu antes ou precede ($\rightarrow$) é definida por:

- $e1 \rightarrow e2$, se e1 e e2 ocorreram no mesmo processo e e1 ocorreu antes de e2
- $e1 \rightarrow e2$, se e1 e e2 são, respectivamente, os eventos de enviar e receber a mensagem m
- $e1 \rightarrow e2$, se $\exists e3$: $e1 \rightarrow e3$ e $e3 \rightarrow e2$ (relação transitiva)

Dois eventos e1, e2 dizem-se concorrentes ($e1 || e2$) se $\neg\ e1 \rightarrow e2$ e $\neg\ e2 \rightarrow e1$.

Nota: objetivo é criar relação similar à causalidade física:

- Se o evento e_1 pode ser a causa do evento e_2, então e_1 aconteceu antes de e_2.

OBJETIVO

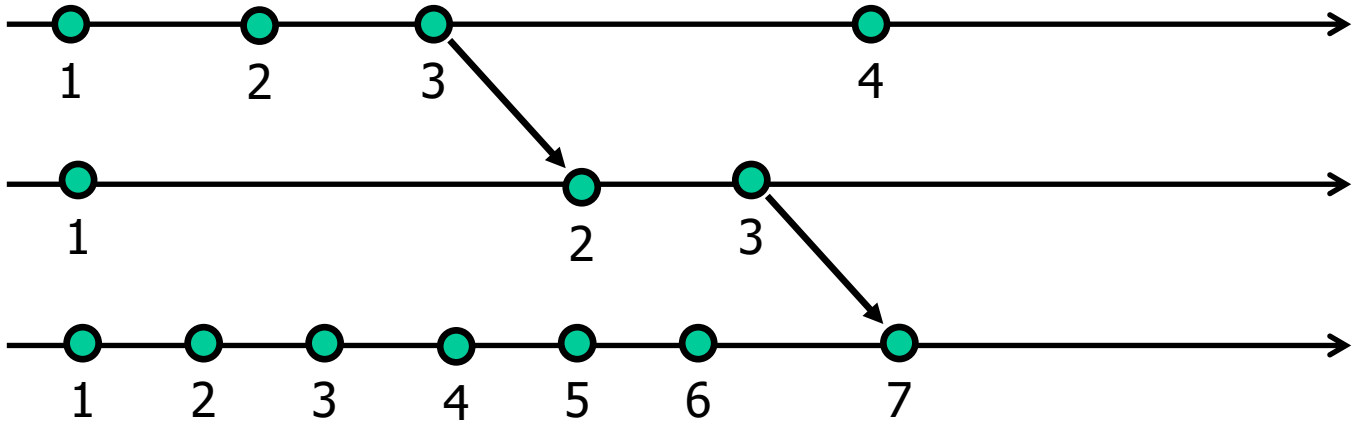
Desenvolver mecanismo que substitua relógios físico e permita:

1. Dados dois eventos, e_1 e e_2, $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$

RELÓGIOS LÓGICOS: PRIMEIRA TENTATIVA

Em cada processo, podemos usar um contador, L_i, para etiquetar os eventos [$T(e_i)=L_i$; $L_i = L_i + 1$]

Problema? Como se pode resolver?

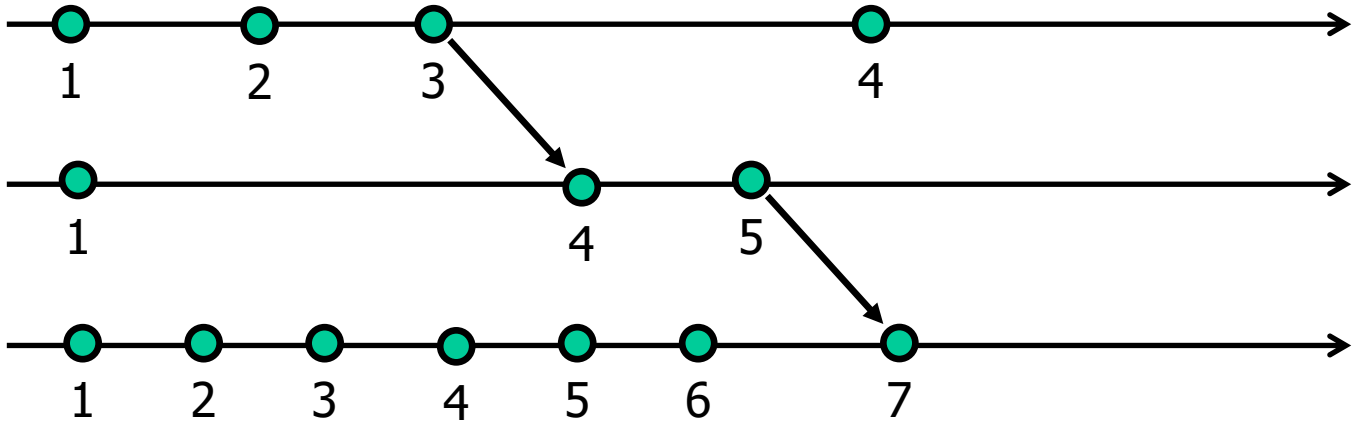


RELÓGIOS LÓGICOS: SEGUNDA TENTATIVA

Em cada processo, podemos usar um contador, L_i, para etiquetar os eventos [$T(e_i)=L_i$; $L_i = L_i + 1$]

Ao enviar uma mensagem, envia-se o valor do contador local

Ao receber uma mensagem, antes de etiquetar a recepção, atualiza-se o relógio local para o máximo do valor recebido e do relógio local [$L_i=\max(L_i,T(msg))$]



DEFINIÇÃO: RELÓGIOS LÓGICOS (LAMPORT 1978)

Um relógio lógico (de Lamport) é um contador monotonicamente crescente, usado para atribuir uma estampilha temporal a um evento. Cada processo i, mantém um relógio lógico Li que atualiza da seguinte forma:

Seja **e** um evento executado no processo i, faz-se:

Li := Li + X (X > 0)

T(e):=Li, com T(e) a estampilha temporal atribuída ao evento *e*.

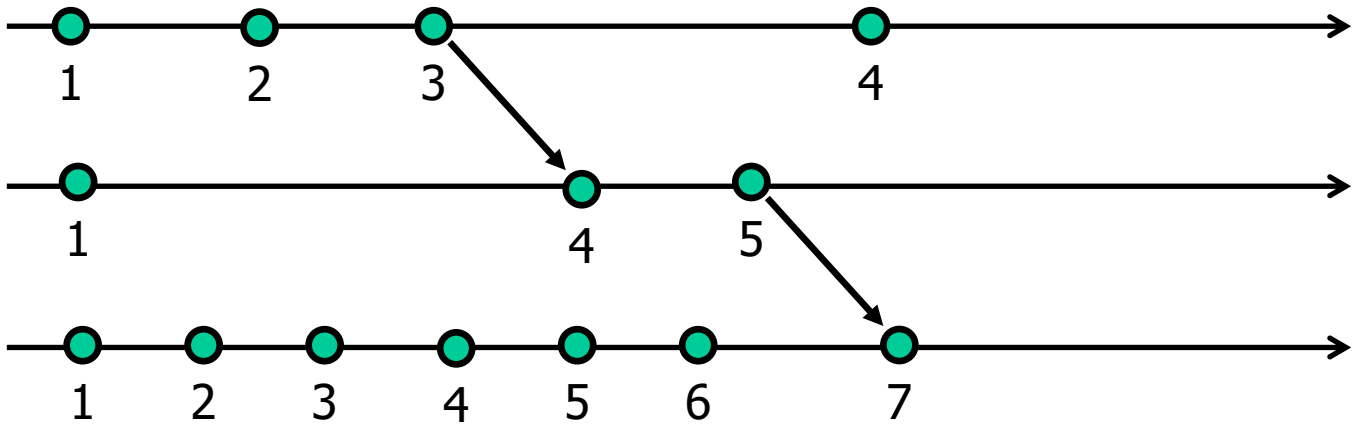
Se **e=send(m)**, aplica-se a regra anterior e envia-se (m,t), com t=T(send(**m**))

Se **e=receive(m,t)**, faz-se Li=max(Li,t) e, de seguida, aplica-se a regra base

RELÓGIOS LÓGICOS: PROPRIEDADES

1. Dados dois eventos, e_1 e e_2, $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$

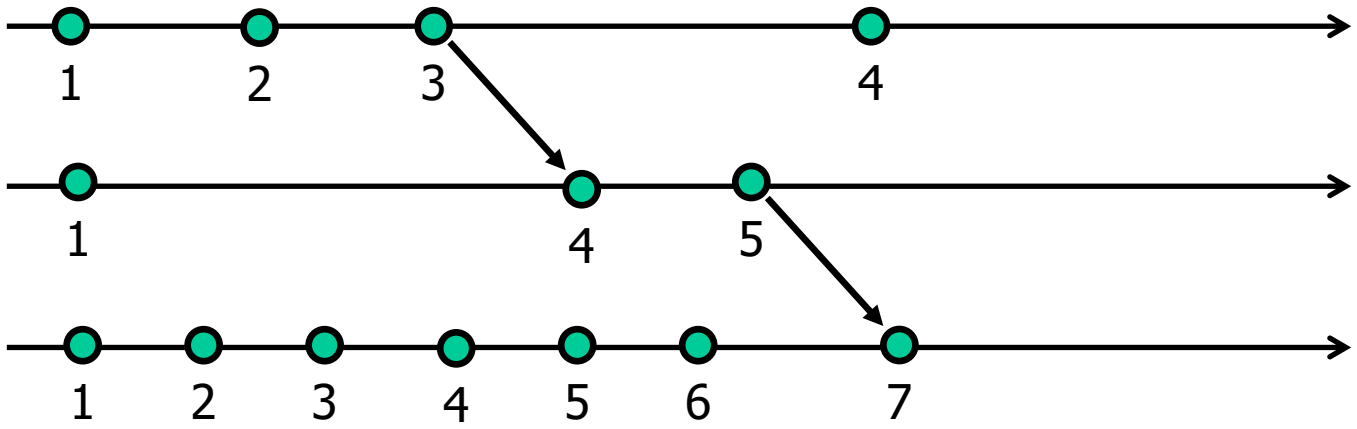
2. Dados dois eventos, e_1 e e_2, $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$?
   Exemplo?



RELÓGIOS LÓGICOS + ORDEM TOTAL

Por vezes é importante estabelecer uma ordem total entre os eventos.

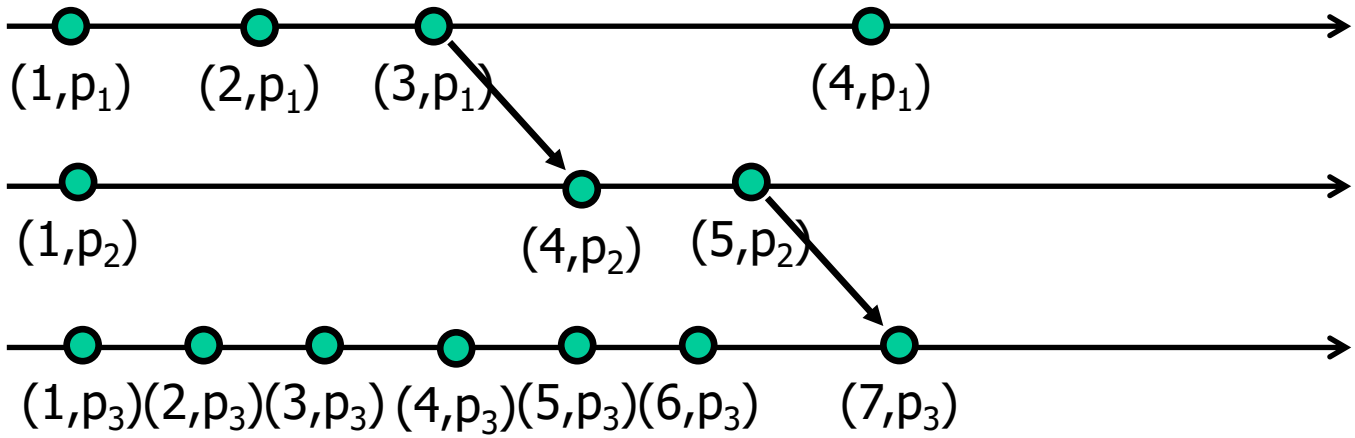
Apenas os relógios lógicos não o permitem. Porquê?



RELÓGIOS LÓGICOS + ORDEM TOTAL

Podemos obter uma ordem adicionado ao relógio o identificador do processo, $T(e_i)=(L_i,p_i)$

$(l_i,p_i) < (l_j,p_j) \Leftrightarrow l_i < l_j \vee (l_i = l_j \wedge p_i < p_j)$



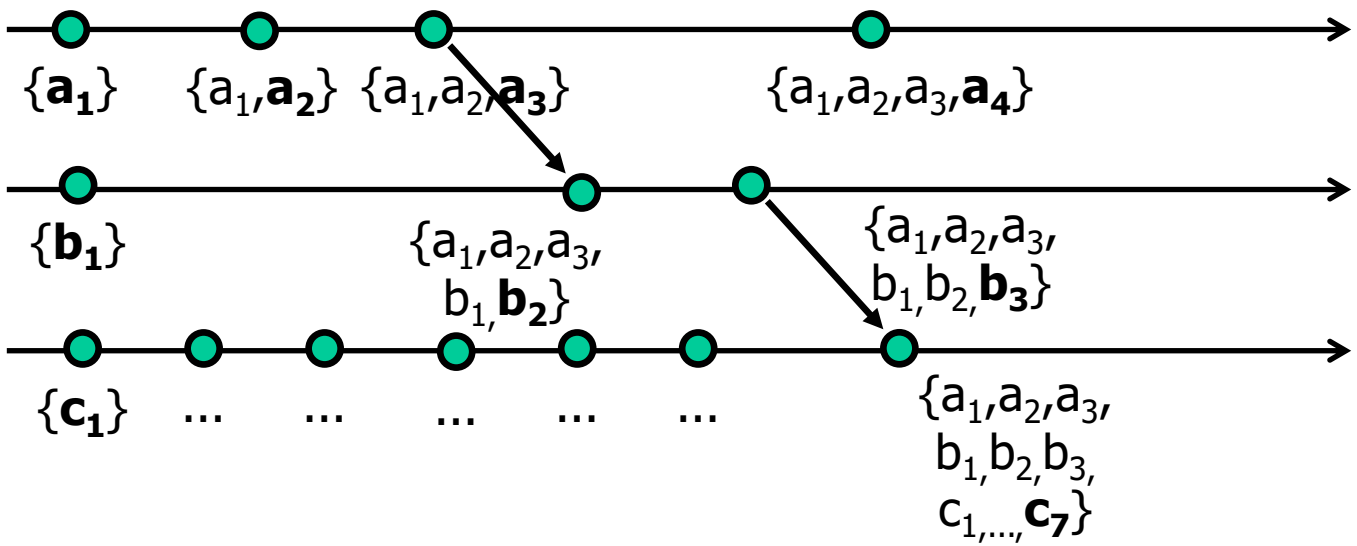
OBJETIVO

Desenvolver mecanismo que substitua relógios físico e permita:

1. Dados dois eventos, e_1 e e_2, $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$
2. Dados dois eventos, e_1 e e_2, $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$

HISTÓRIA CAUSAL

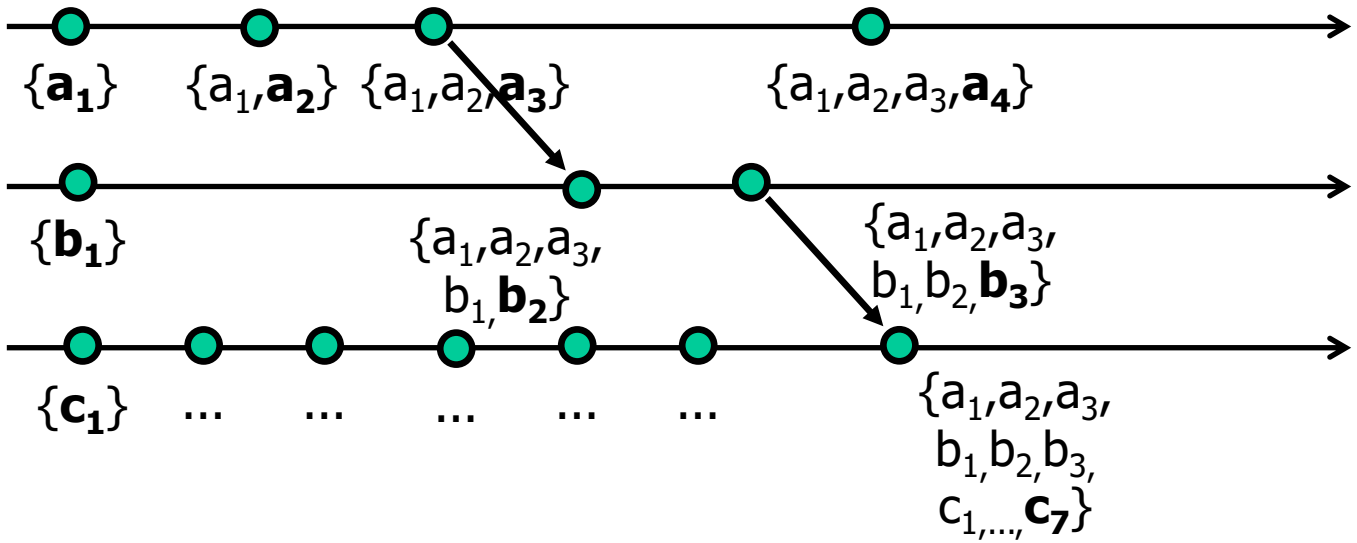
Se se quer saber o que aconteceu antes, porque não manter para cada evento o conjunto de eventos que ocorreram antes desse evento?



HISTÓRIA CAUSAL

A história causal dum evento, H(e), é um conjunto que inclui o próprio evento e os evento que aconteceram antes.

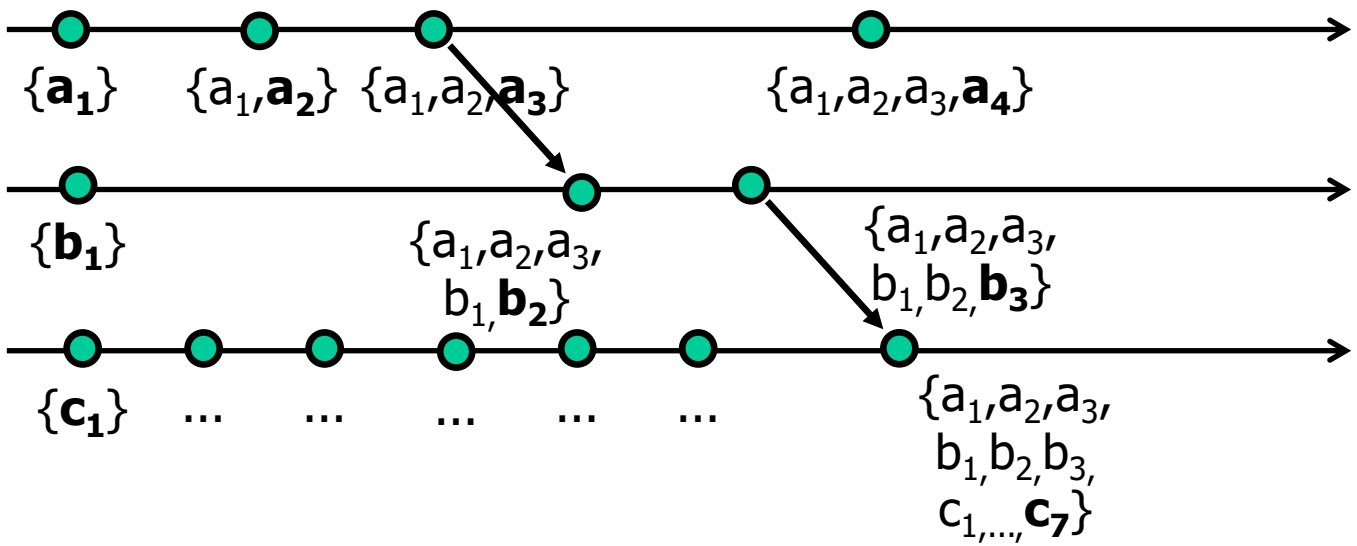
Cada evento, e, é etiquetado com um identificador, E(e); a história causal desse evento é a reunião desse evento com os eventos anteriores (uma mensagem propaga a história no momento da emissão).



HISTÓRIA CAUSAL

Como comparar duas histórias causais?

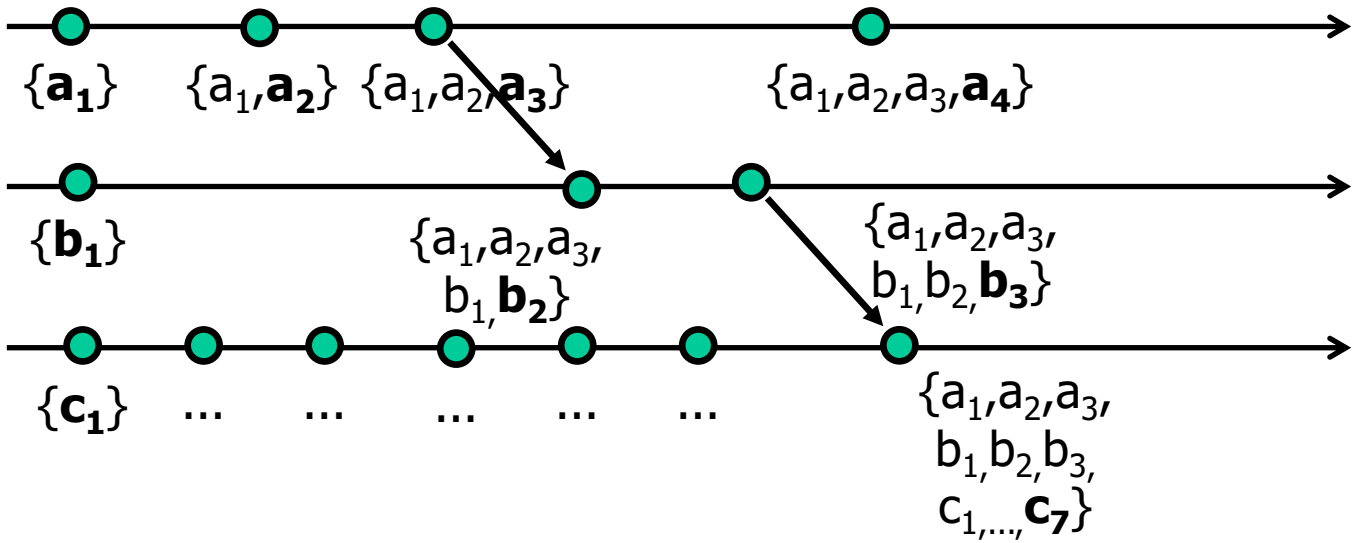
$H(e_1) < H(e_2) \Leftrightarrow H(e_1) \subset H(e_2) \wedge H(e_1) \neq H(e_2)$



HISTÓRIA CAUSAL

1. Dados dois eventos, e_1 e e_2, $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$
   - $e_1 \rightarrow e_2 \Rightarrow H(e_1) \subset H(e_2) \wedge H(e_1) \neq H(e_2)$
2. Dados dois eventos, e_1 e e_2, $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$
   - $H(e_1) \subset H(e_2) \wedge H(e_1) \neq H(e_2) \Rightarrow e_1 \rightarrow e_2$

   [ou mais simples: $E(e_1) \in H(e_2) \Rightarrow e_1 \rightarrow e_2$]



DEFINIÇÃO: HISTÓRIA CAUSAL

Supondo que cada evento, e, é etiquetado com um identificador, E(e)

A história causal, H(e), de um evento, e, é o conjunto (de identificadores) dos eventos que *aconteceram antes* de e.

Num dado processo i:

- para o primeiro evento e^i_0, $H(e^i_0) = \{e^i_0\}$
- para o evento e^i_n, $H(e^i_n)=H(e^i_{n-1}) \cup \{e^i_n\}$ (caso e não seja um receive)

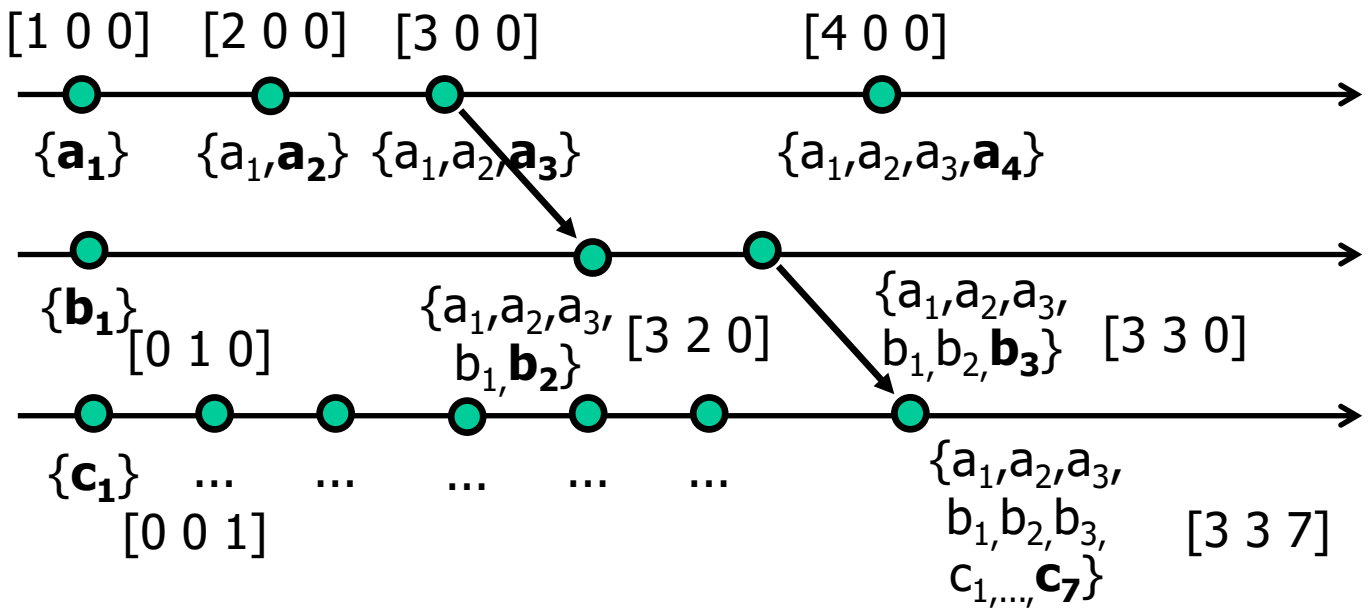
Se **e=send(m)**, aplica-se a regra anterior e envia-se (m,t), com t=H(e)

Se $\mathbf{e^i_n}$**=receive(m,t)**, $H(e^i_n) = H(e^i_{n-1}) \cup t \cup \{e^i_n\}$

RELÓGIO VETORIAL

Um relógio vetorial, V(e), não é mais do que uma forma eficiente de representar uma história causal, mantendo um vetor que em cada posição tem o contador do último evento dum processo (todos os eventos anteriores desse processo são necessariamente conhecidos).

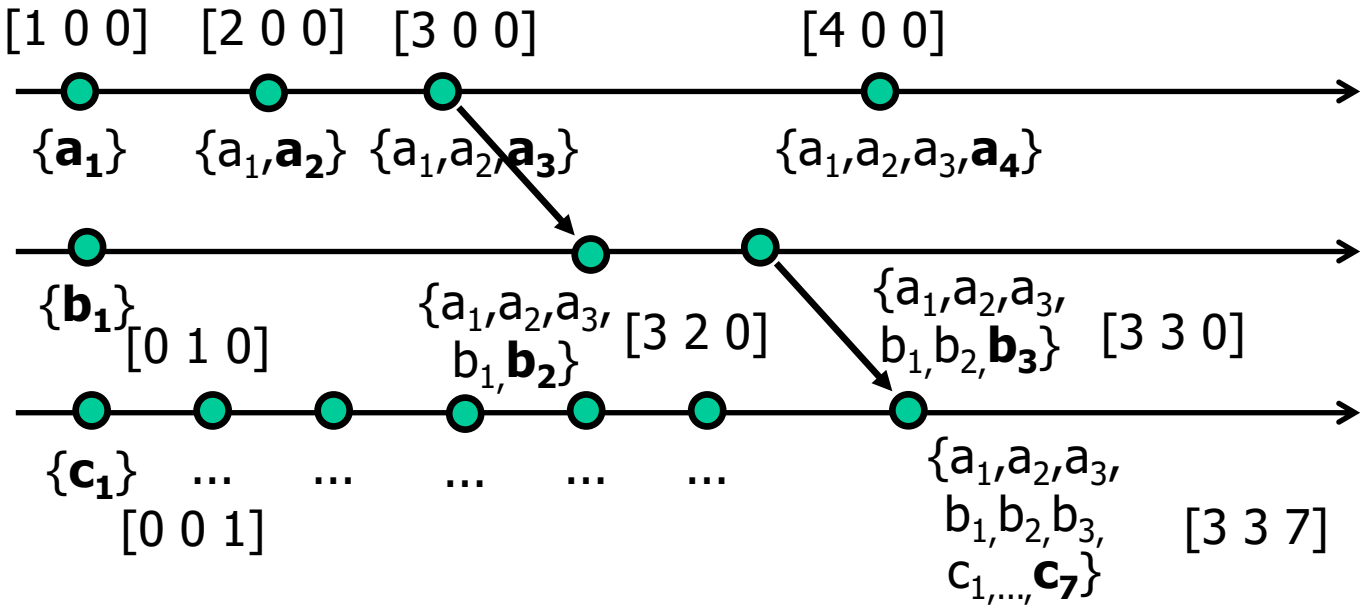
[a b c]



RELÓGIO VETORIAL

Dados dois eventos, e_1 e e_2, $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$

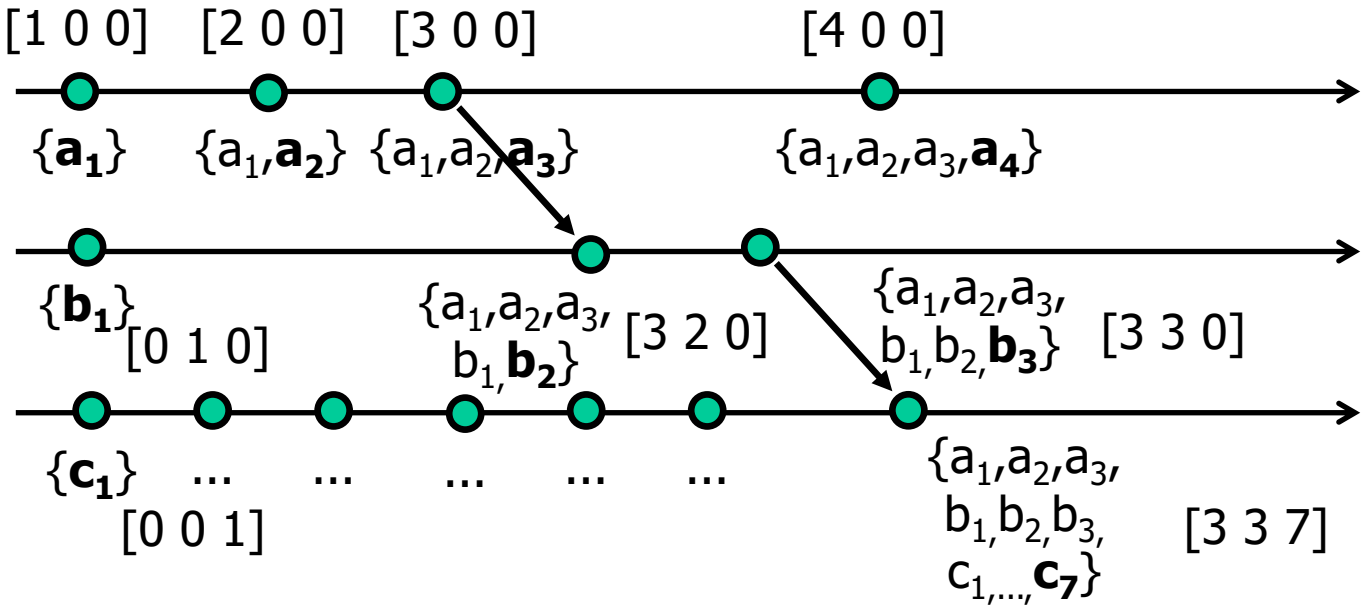
- $e_1 \rightarrow e_2 \Rightarrow H(e_1) \subset H(e_2) \wedge H(e_1) \neq H(e_2)$ (com histórias causais)
- $e_1 \rightarrow e_2 \Rightarrow \forall i, V(e_1)[i] \leq V(e_2)[i] \wedge \exists j: V(e_1)[j] \neq V(e_2)[j]$



RELÓGIO VETORIAL

Dados dois eventos, e_1 e e_2, $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$

- $H(e_1) \subset H(e_2) \wedge H(e_1) \neq H(e_2) \Rightarrow e_1 \rightarrow e_2$ (com histórias causais)
- $\forall i, V(e_1)[i] \leq V(e_2)[i] \wedge \exists j: V(e_1)[j] \neq V(e_2)[j] \Rightarrow e_1 \rightarrow e_2$

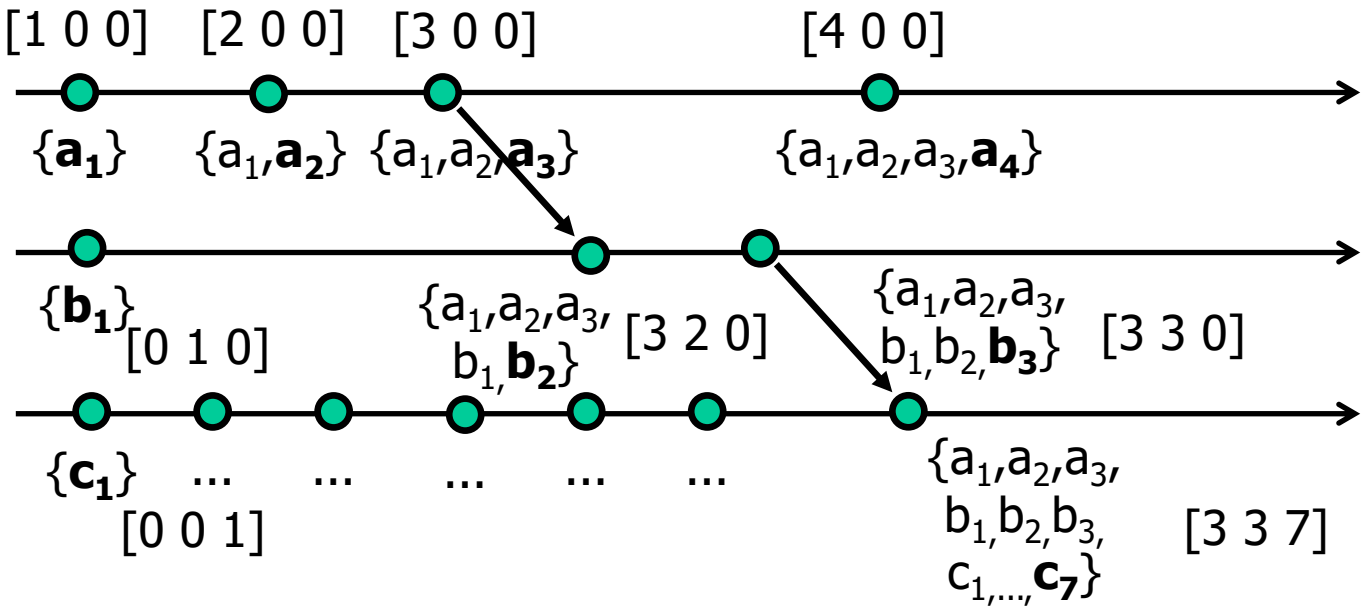


RELÓGIO VETORIAL

Dados dois eventos, e_1 e e_2,

$\neg(C(e_1) < C(e_2)) \wedge \neg(C(e_1) < C(e_2)) \Rightarrow e_1 \parallel e_2$

- $\neg(H(e_1) \subset H(e_2)) \wedge \neg(H(e_2) \subset H(e_1)) \Rightarrow e_1 \parallel e_2$ (com histórias causais)
- $\exists i,j: V(e_1)[i] < V(e_2)[i] \wedge V(e_2)[j] < V(e_1)[j] \Rightarrow e_1 \parallel e_2$



DEFINIÇÃO: RELÓGIO VETORIAL

Um relógio vetorial num sistema com *n* processos é um vetor de n inteiros, V[1..n]. Cada processo i mantém um relógio vectorial V_i que usa para atribuir uma estampilha temporal aos eventos locais e atualiza-a da seguinte forma:

Inicialmente, $V_i[j]=0$, $\forall i,j$

Antes de etiquetar um evento **e** no processo i, faz-se: Vi[i] := Vi[i]+1. C(e)=Vi

Se **e=send(m)**, aplica-se a regra anterior e envia-se (m,t), com t=C(send(**m**))

Se **e=receive(m,t*)*** no processo i, faz-se Vi[j]=max(Vi[j],t[j]), $\forall j$ e, de seguida, aplica-se a regra base

Dados dois relógios vectoriais V1 e V2, tem-se:

V1=V2, sse V1[i]=V2[i], $\forall i$

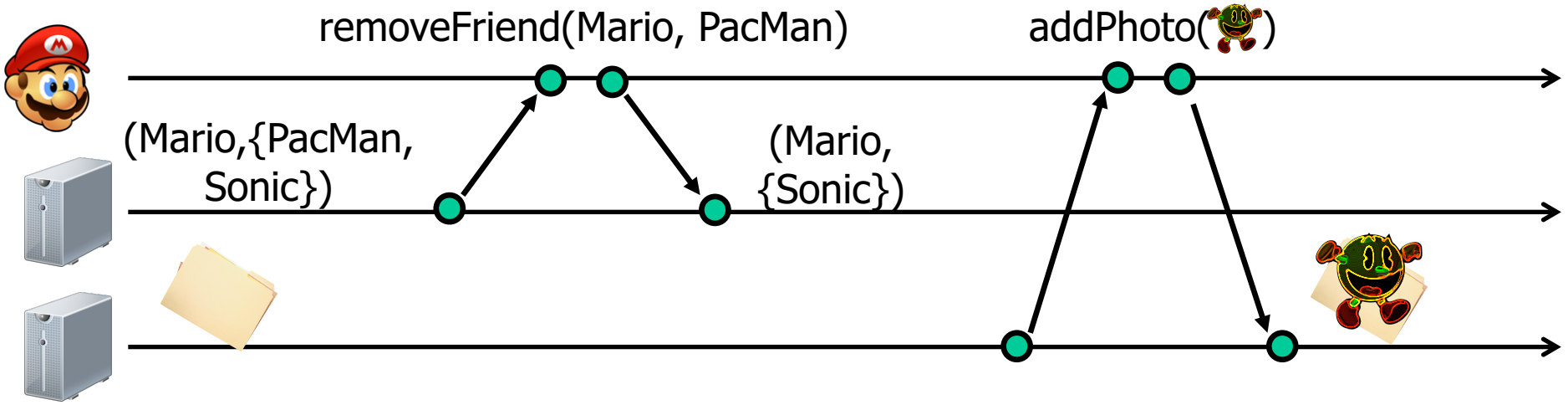
V1≤V2, sse V1[i] ≤V2[i], $\forall i$

V1<V2, sse V1≤V2 $\wedge$ V1≠V2

HÁ EVENTOS E EVENTOS

Problema inicial: Para adicionar fotografia que não se pretenda que amigo X veja, deve-se:

1. Remover X da lista de amigos
2. Adicionar fotografia

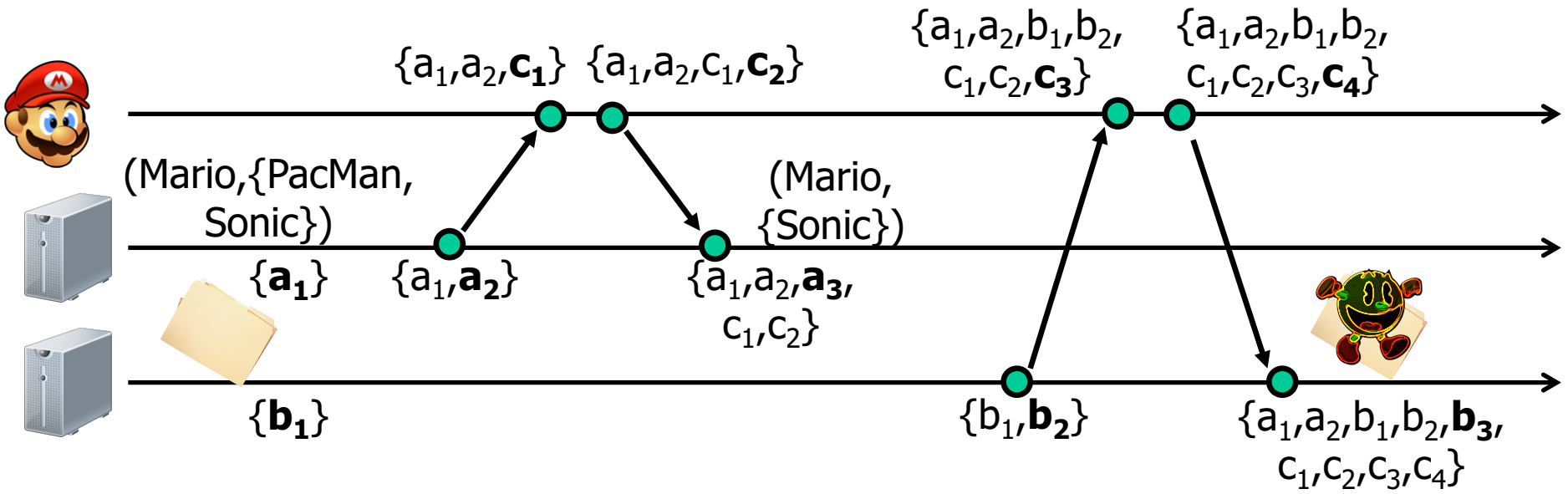


HÁ EVENTOS E EVENTOS

Problema inicial: Para adicionar fotografia que não se pretenda que amigo X veja, deve-se:

1. Remover X da lista de amigos
2. Adicionar fotografia

Como se garante que PacMan não vê foto? A foto regista os eventos na lista de amigos que devem ser vistos.

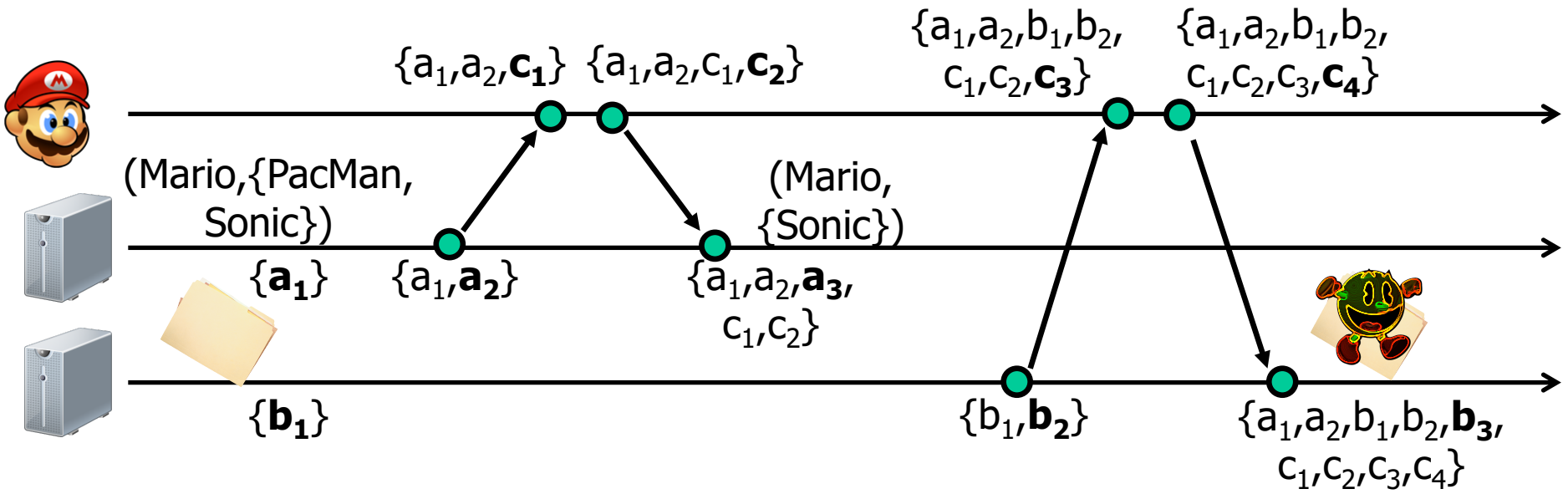


HÁ EVENTOS E EVENTOS

Problema inicial: Para adicionar fotografia que não se pretenda que amigo X veja, deve-se:

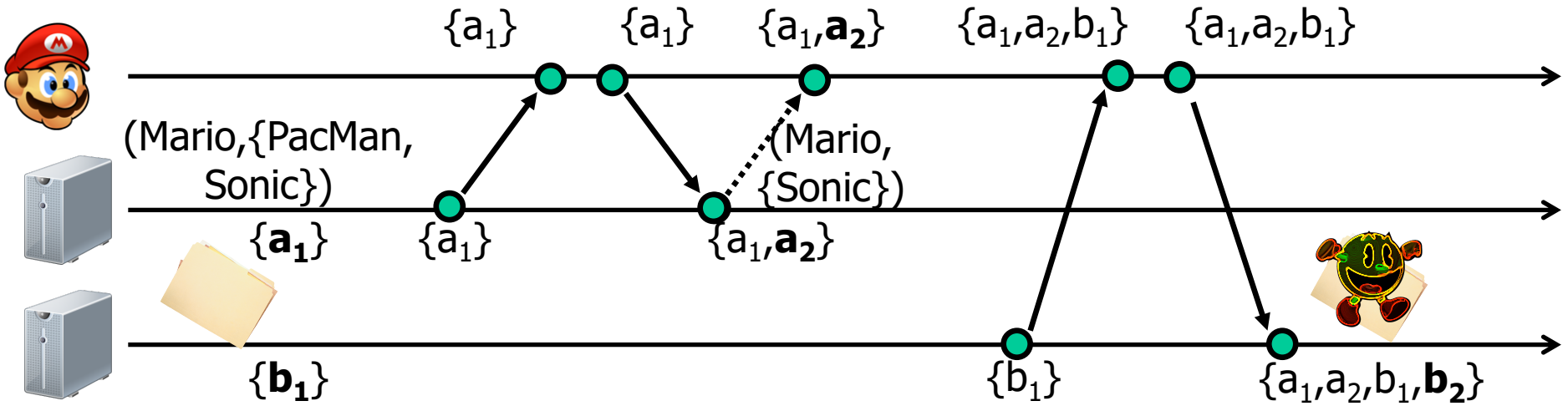
1. Remover X da lista de amigos
2. Adicionar fotografia

Problemas?
Eventos irrelevantes na história causal
Clientes poluem a história causal



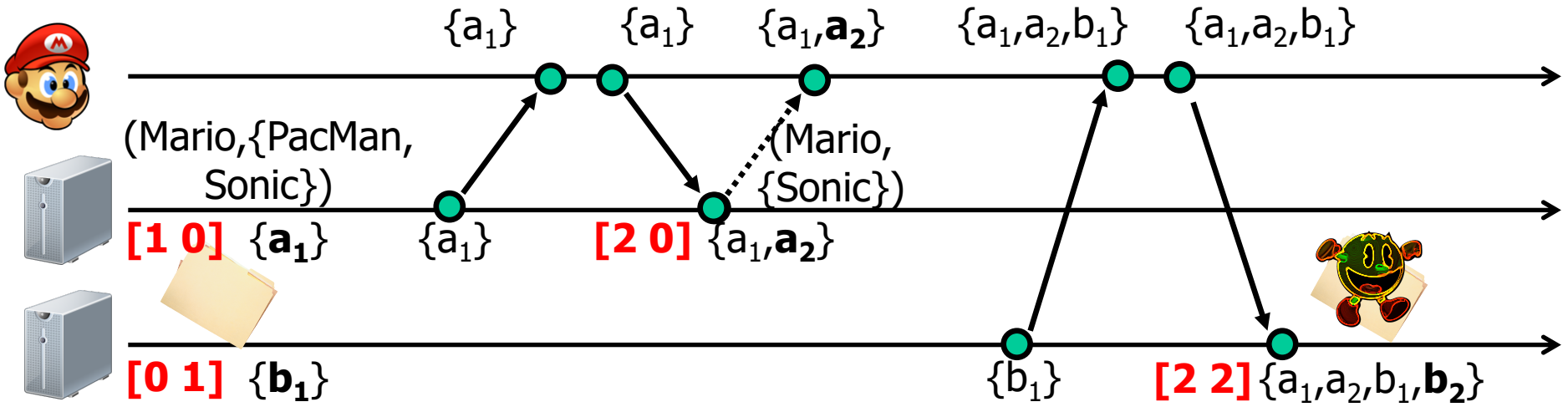
HÁ EVENTOS E EVENTOS

Num sistema não é necessário registar todos os eventos – apenas os eventos importante que fazem o sistema alterar o seu estado



VETOR VERSÃO

Um vetor versão, VV(e), não é mais do que um relógio vetorial em que apenas se registam os eventos importantes para um sistema de gestão de dados (escritas)



DEFINIÇÃO: VETOR VERSÃO

Um vetor versão num sistema de gestão de dados com *n* servidores é um vetor de n inteiros, VV[1..n]. Cada servidor i mantém um vetor versão V_i que usa para atribuir uma estampilha temporal aos eventos locais relevantes e atualiza-a da seguinte forma:

Inicialmente, $VV_i[j]=0$, $\forall i,j$

Os eventos não importantes não são etiquetados.

Antes de etiquetar um evento importante **e** no processo i, faz-se: VVi[i] := VVi[i]+1. C(e)=VVi

Para um evento **e** não importantes, tem-se C(e)=VVi. Os eventos de envio e recepção de mensagem são considerados não importantes. A recepção duma escrita no servidor tem associado um evento importante de criação de um novo estado.

Se **e=send(m)**, envia-se (m,t), com t=C(send(**m**)).

Se **e=receive(m,t*)** no processo i, faz-se VVi[j]=max(VVi[j],t[j]) , $\forall j$.

Dados dois vetores versão VV1 e VV2, tem-se:

VV1=VV2, sse VV1[i]=VV2[i], $\forall i$

VV1≤VV2, sse VV1[i] ≤VV2[i], $\forall i$

VV1<VV2, sse VV1≤VV2 $\wedge$ VV1≠VV2

PARA SABER MAIS

Carlos Baquero and Nuno Preguiça. 2016. Why Logical Clocks are Easy. *Queue* 14, 1, pages 60 (February 2016). http://queue.acm.org/detail.cfm?id=2917756

G. Coulouris, J. Dollimore and T. Kindberg, Distributed Systems –Concepts and Design, Addison-Wesley, 5th Edition, 2011

Secção 14.2, 14.4