

SISTEMAS DISTRIBUÍDOS

Capítulo 7

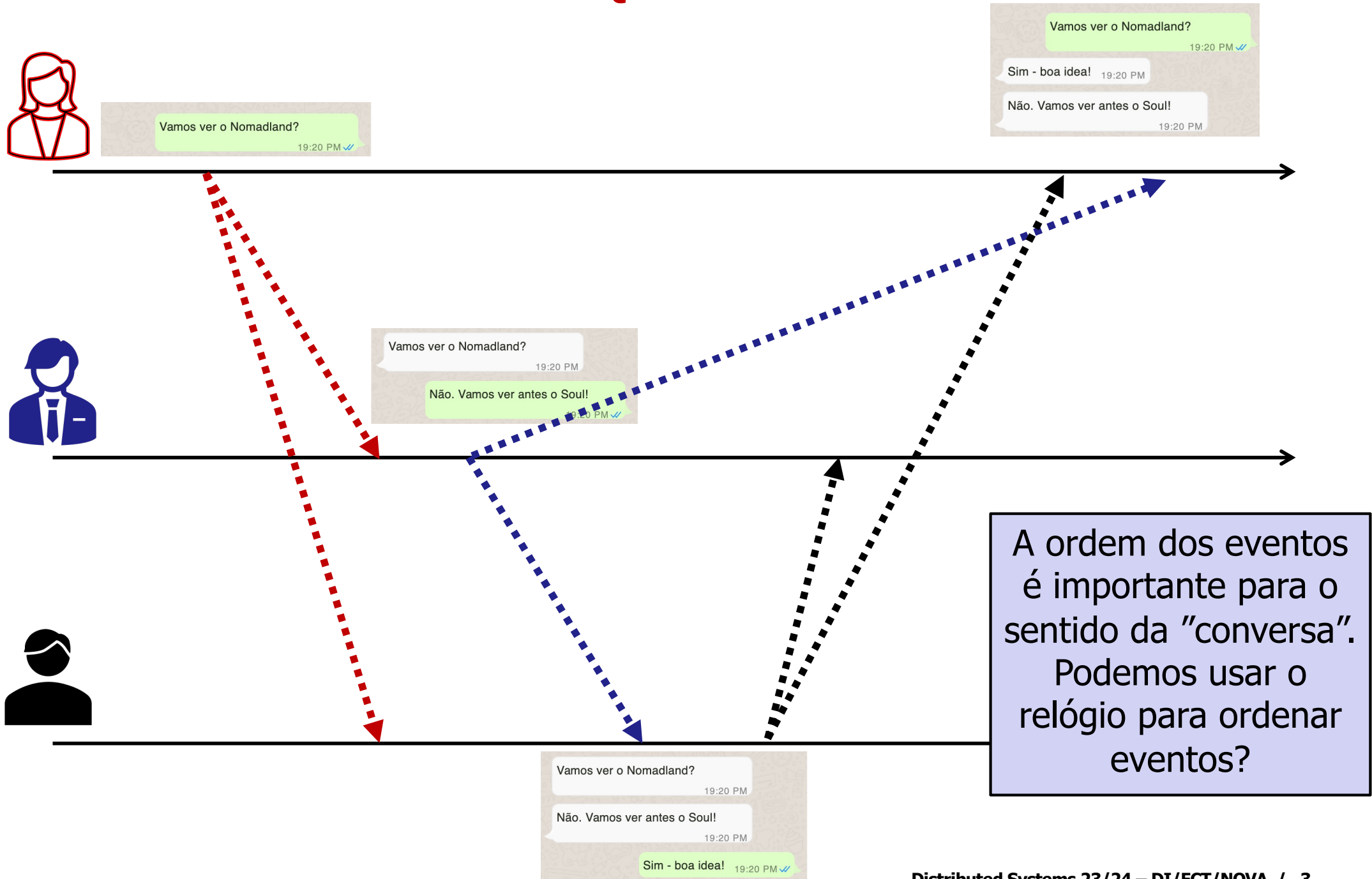
Tempo e ordenação de eventos

NA AULA DE HOJE

Tempo em sistemas distribuídos

- **Motivação para ordenar eventos**
- Relógios lógicos
- História causal e vetor versão

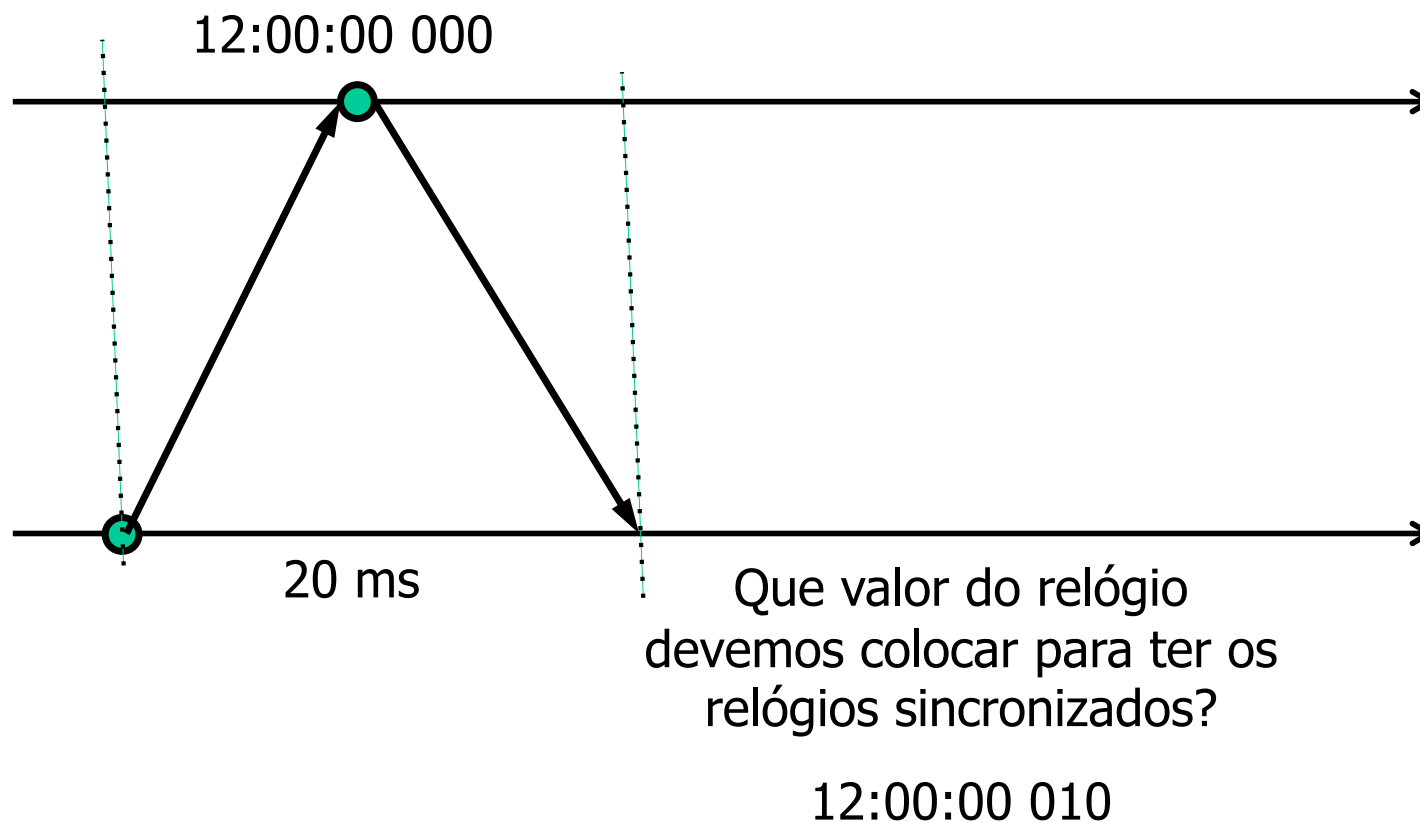
ORDENAR EVENTOS? PORQUÊ?



MOTIVAÇÃO

Num sistema distribuído existem limites para precisão da sincronização dos valores dos relógios de vários computadores.

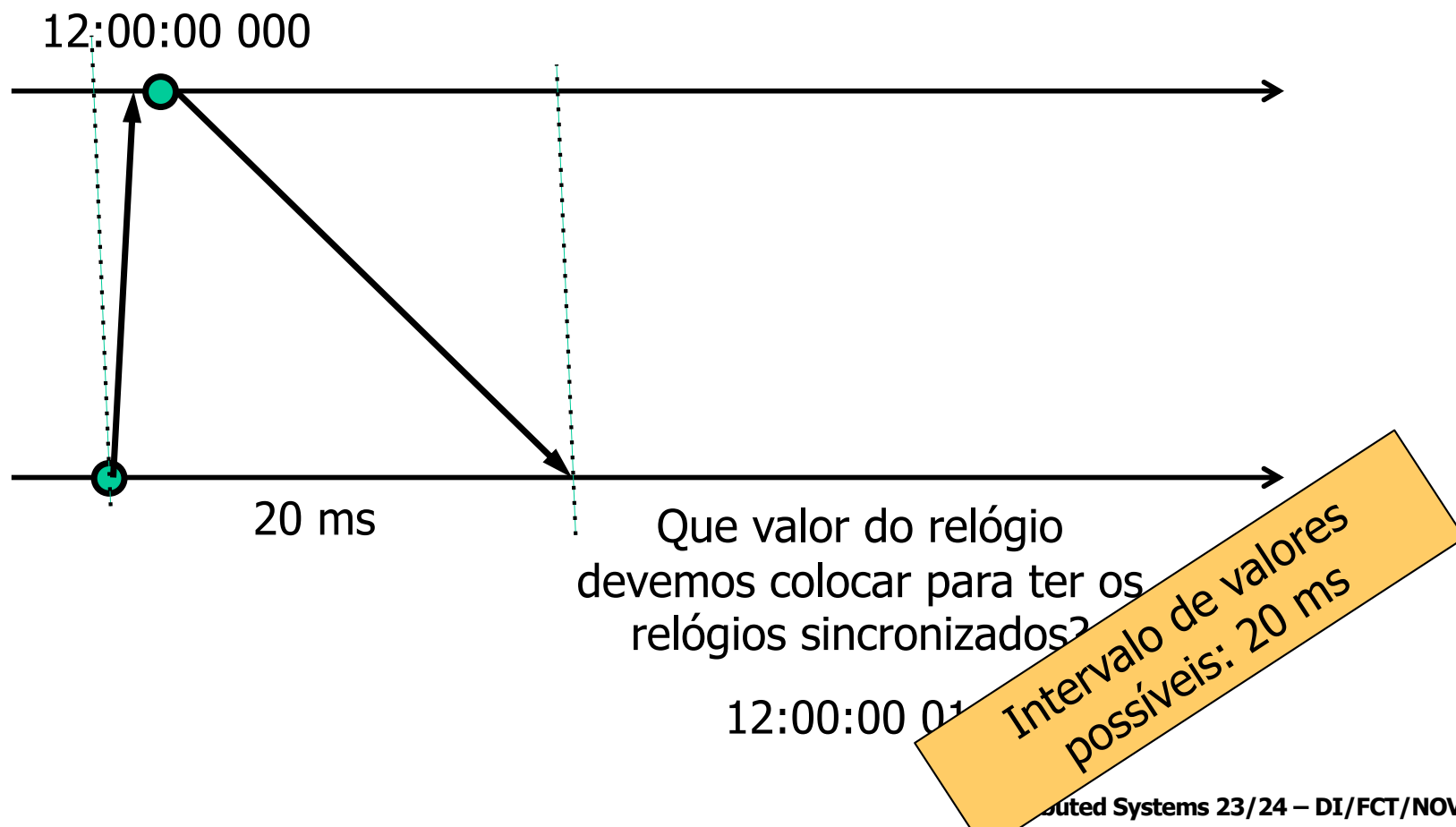
- Porquê?



MOTIVAÇÃO

Num sistema distribuído existem limites para precisão da sincronização dos valores dos relógios de vários computadores.

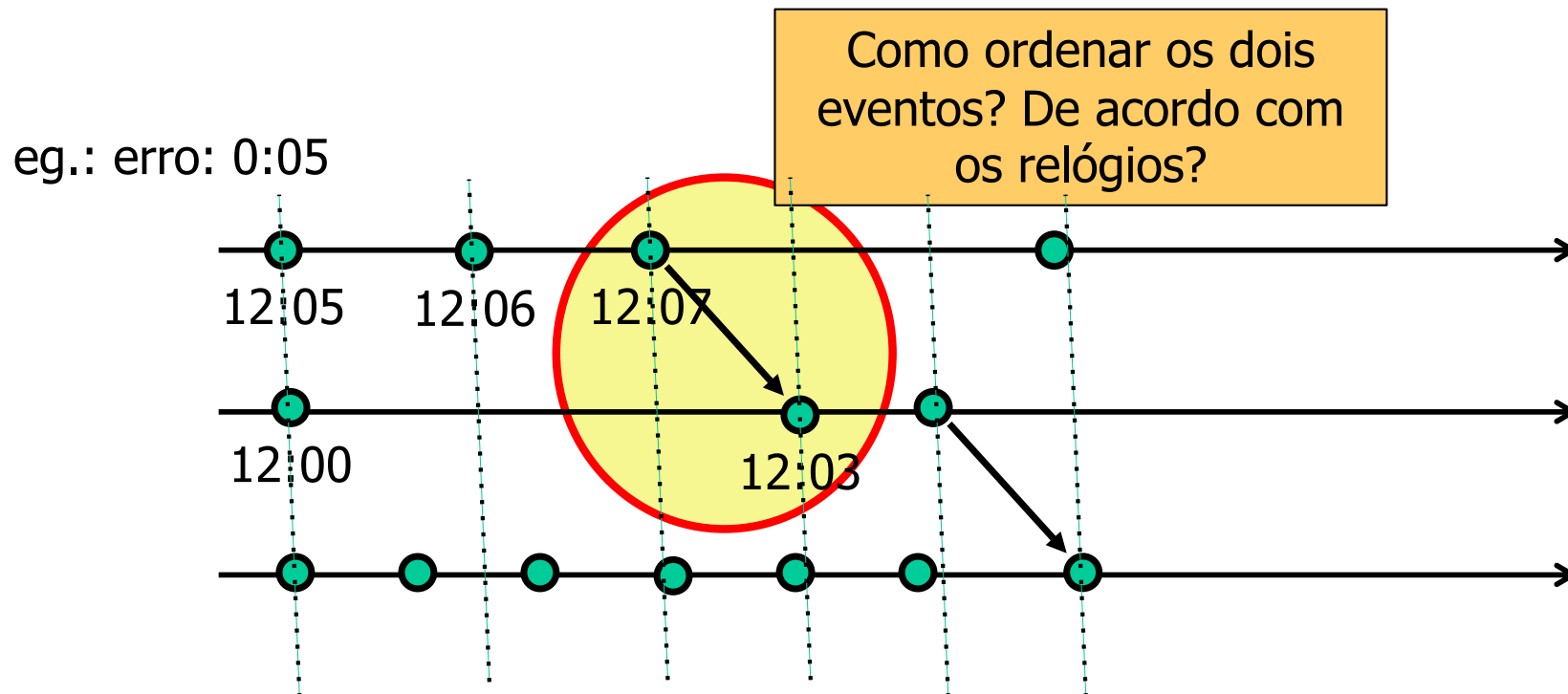
- Porquê?



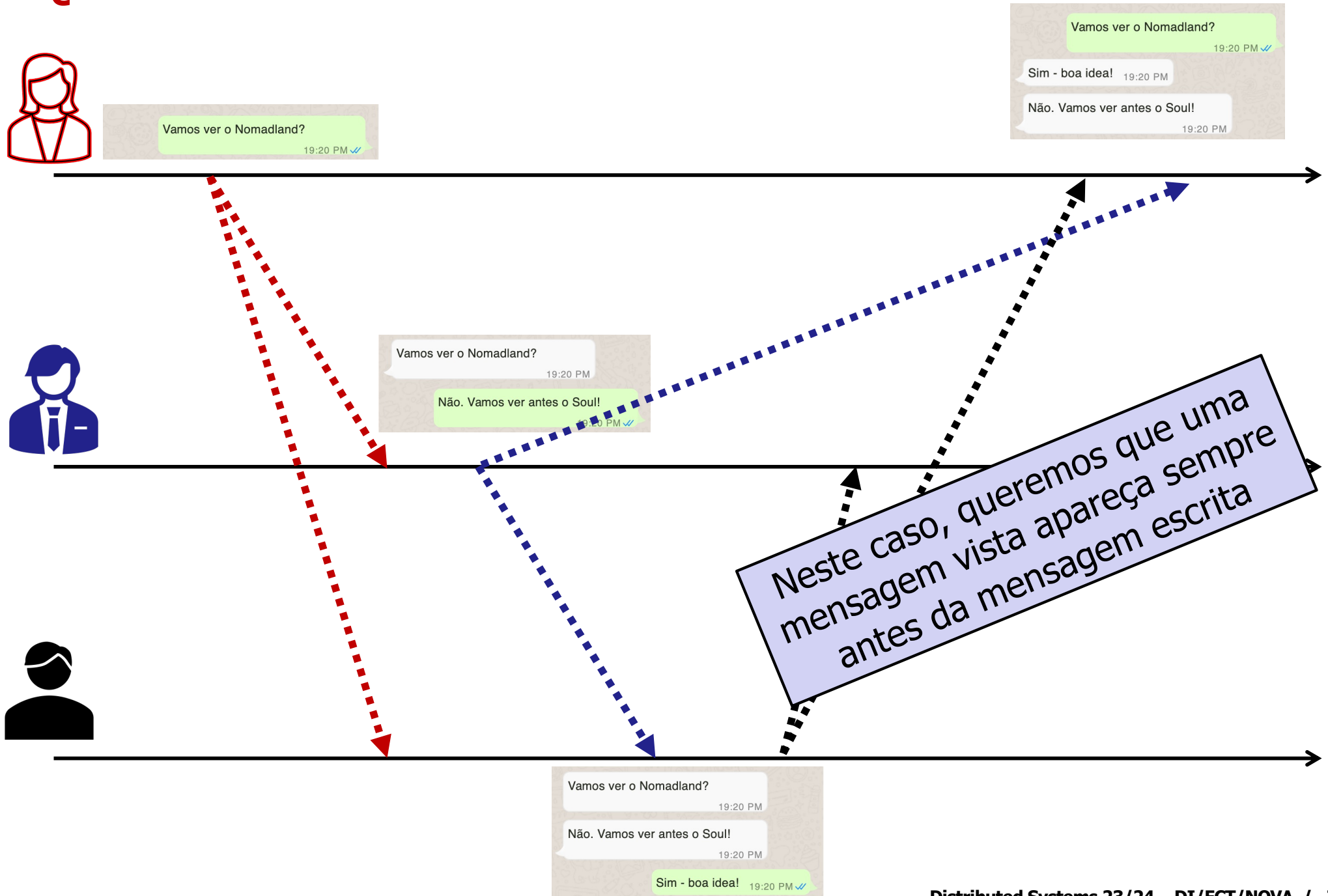
MOTIVAÇÃO

Num sistema distribuído existem limites para precisão da sincronização dos valores dos relógios de vários computadores.

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



QUE ORDENS SÃO IMPORTANTES?

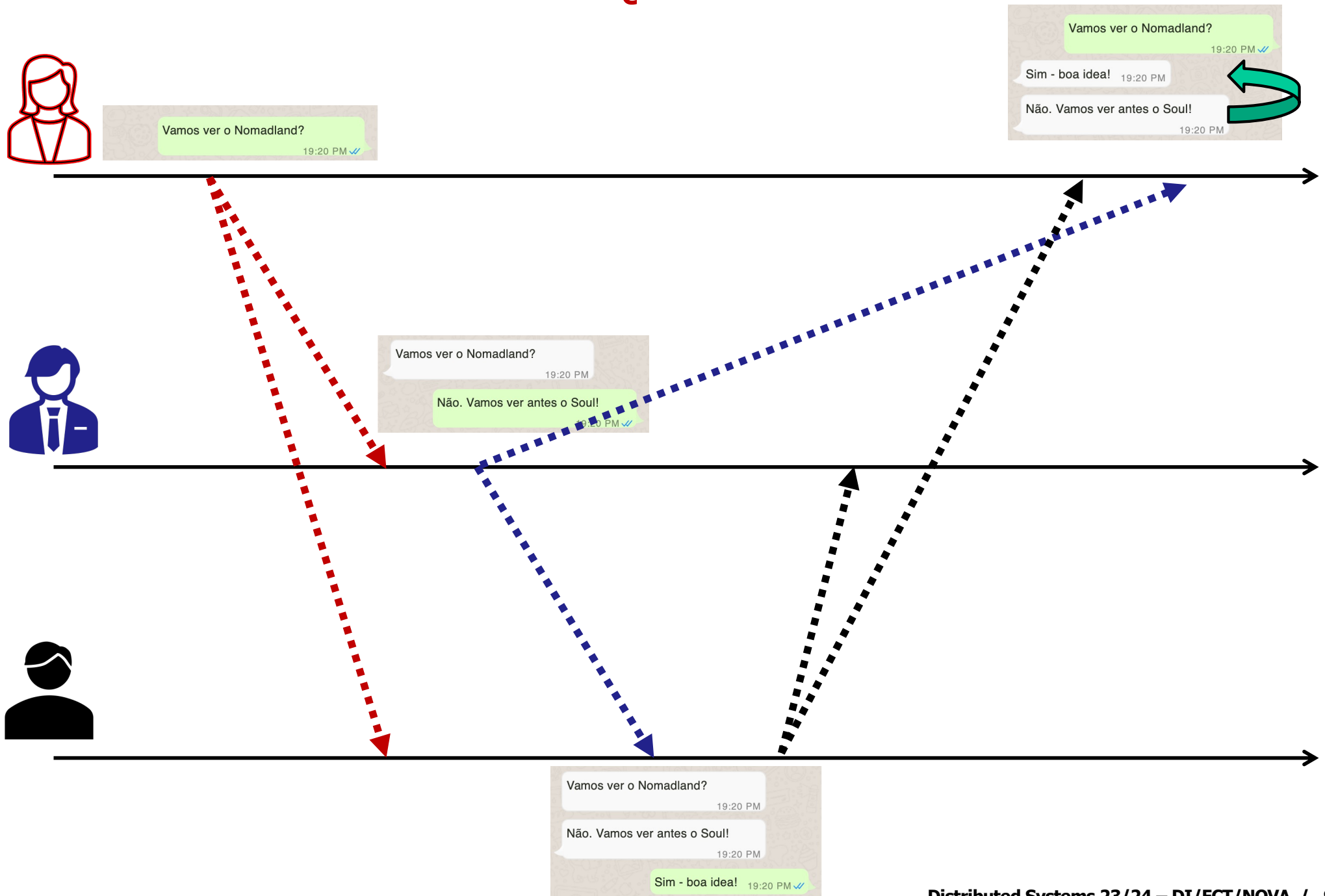


COMO SE DEFINE “ACONTECEU ANTES” ?

Muitas vezes o valor do relógio é apenas usado para determinar indiretamente se um evento pode ter causado outro, i.e., as relações de causalidade.

Num sistema distribuído, estamos em geral interessados em conhecer as relações de causalidade.

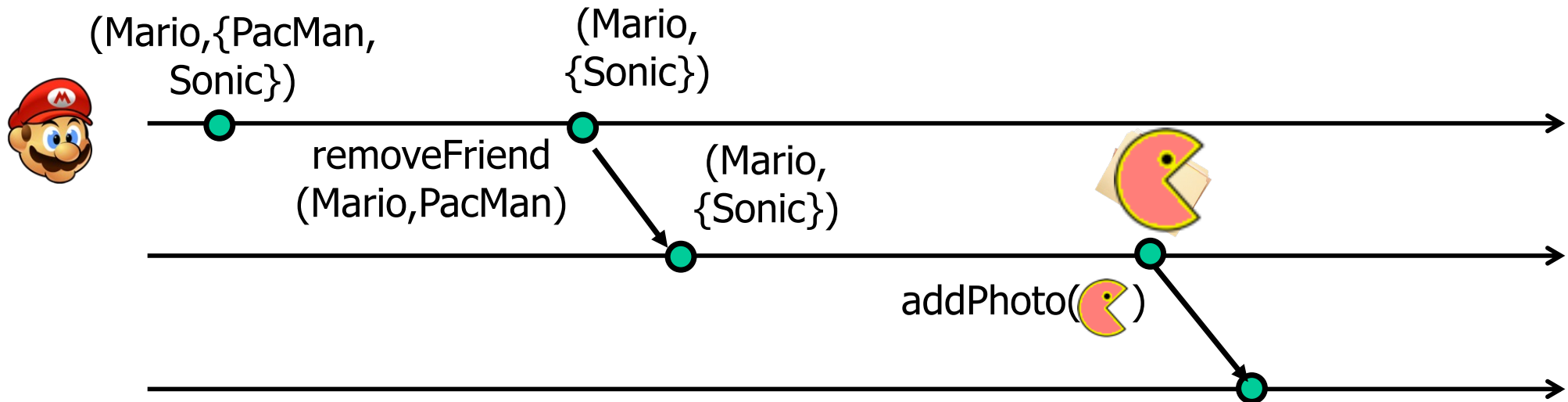
ORDENAR EVENTOS? PORQUÊ?



OUTRO EXEMPLOS

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

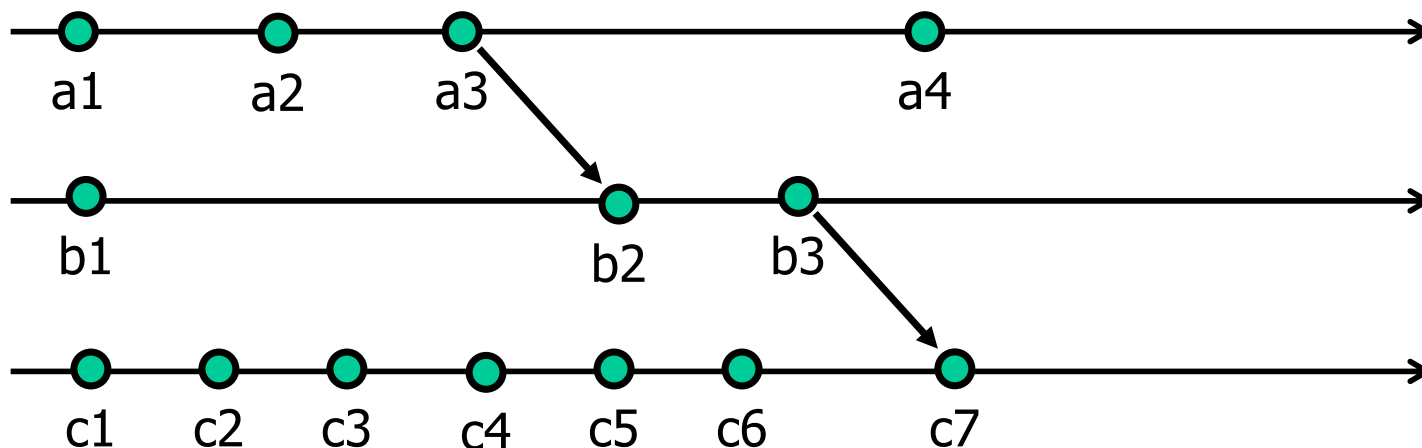
1. Remover X da lista de amigos
2. Adicionar fotografia



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

COMO SE DEFINE "ACONTECEU ANTES" ?

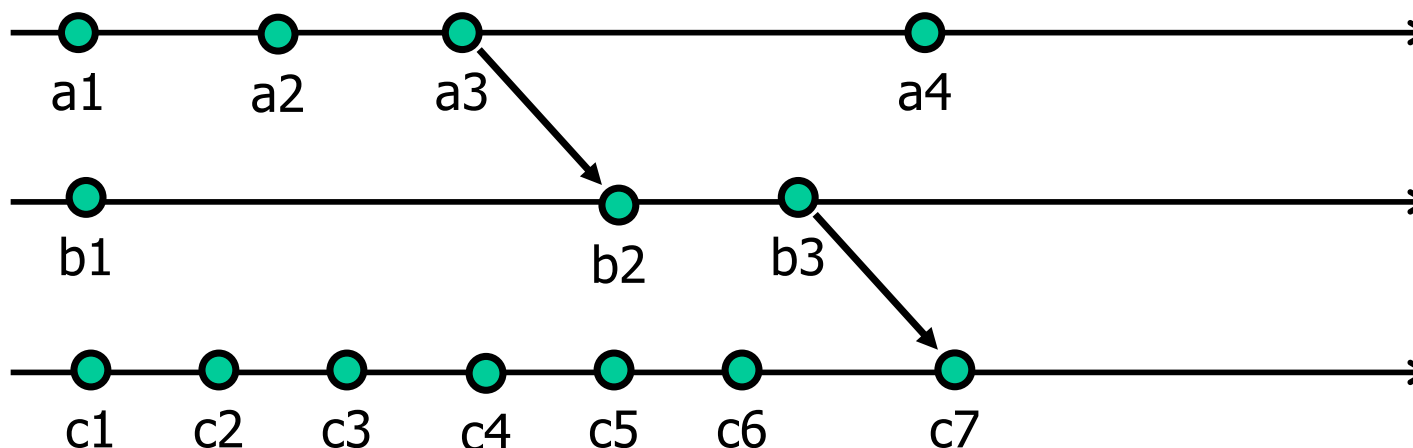
A relação "aconteceu antes" captura a relação de causalidade potencial: o evento e_1 aconteceu antes de e_2 se e_1 pode ter causado e_2 .



COMO SE DEFINE "ACONTECEU ANTES" ?

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

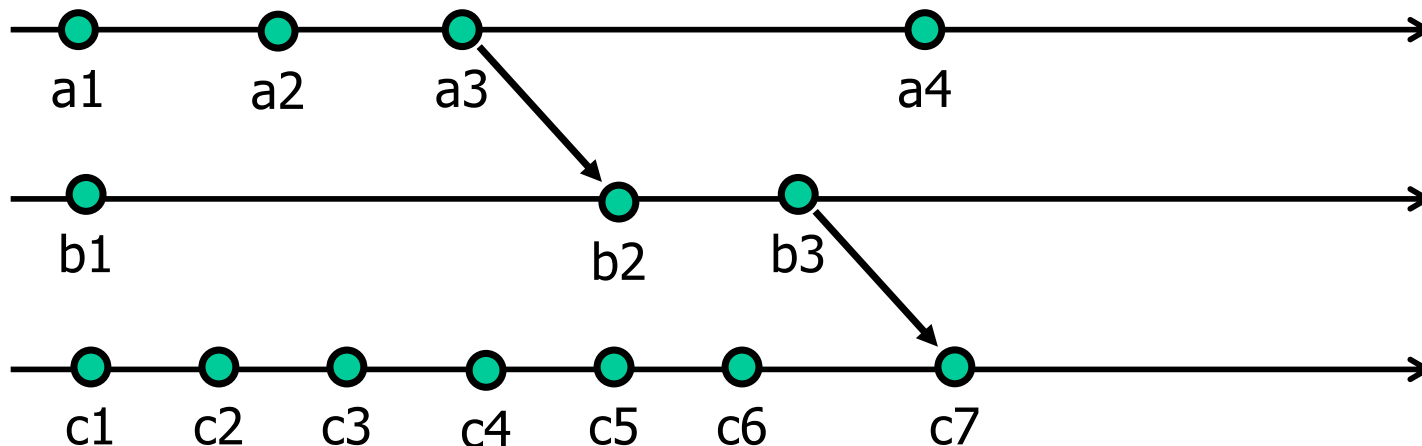
- $a1 \rightarrow a2 \rightarrow a3 \rightarrow a4$
- $b1 \rightarrow b2 \rightarrow b3$
- $c1 \rightarrow c2 \rightarrow c3 \rightarrow c4 \rightarrow c5 \rightarrow c6 \rightarrow c7$



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

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

- $a3 \rightarrow b2$
- $b3 \rightarrow c7$



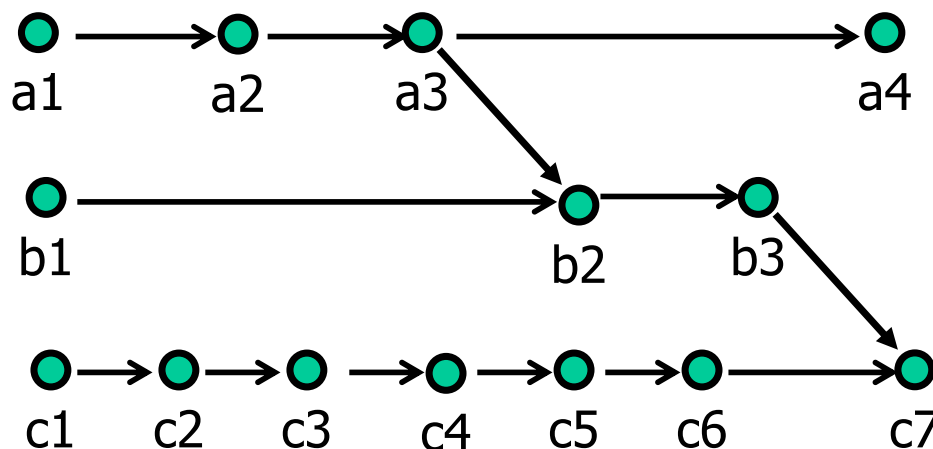
COMO SE DEFINE "ACONTECEU ANTES" ? (CONT.)

A relação *aconteceu antes* é transitiva:

- $x1 \rightarrow x2$ e $x2 \rightarrow x3 \Rightarrow x1 \rightarrow x3$

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

- $a1 \parallel b1, a1 \parallel c1, a1 \parallel c2, \dots, a1 \parallel c6, a2 \parallel b1, \dots$



DEFINIÇÃO: RELAÇÃO *ACONTECEU ANTES* (LAMPORT 1978)

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

$e_1 \rightarrow e_2$, se e_1 e e_2 ocorreram no mesmo processo e e_1 ocorreu antes de e_2

$e_1 \rightarrow e_2$, se e_1 e e_2 são, respetivamente, os eventos de enviar e receber a mensagem m

$e_1 \rightarrow e_2$, se $\exists e_x: e_1 \rightarrow e_x$ e $e_x \rightarrow e_2$ (relação transitiva)

Dois eventos **e_1** , **e_2** dizem-se concorrentes:

$$(e_1 || e_2) \text{ se } \neg e_1 \rightarrow e_2 \text{ e } \neg e_2 \rightarrow e_1.$$

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 .

NA AULA DE HOJE

Tempo em sistemas distribuídos

- Motivação para ordenar eventos
- **Relógios lógicos**
- História causal e vetor versão

OBJETIVO

Desenvolver mecanismo que substitua relógios físicos e permita:

1. Dados dois eventos, \mathbf{e}_1 e \mathbf{e}_2 , $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow \mathbf{C}_{\text{lock}}(\mathbf{e}_1) < \mathbf{C}_{\text{lock}}(\mathbf{e}_2)$

Se um evento e_1 aconteceu antes de e_2 ,
então o relógio de e_1 deve ser menor
que o relógio de e_2 .

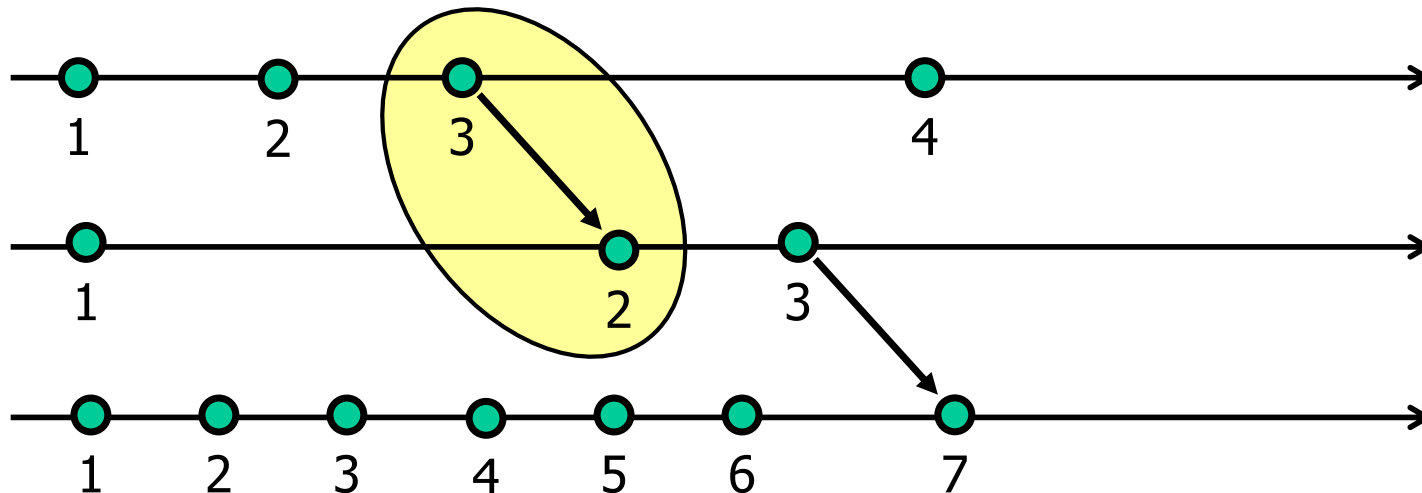
RELÓGIOS LÓGICOS: PRIMEIRA TENTATIVA

Queremos definir $\text{Clock}(e)$

$$e_1 \rightarrow e_2 \Rightarrow \text{Clock}(e_1) < \text{Clock}(e_2)$$

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

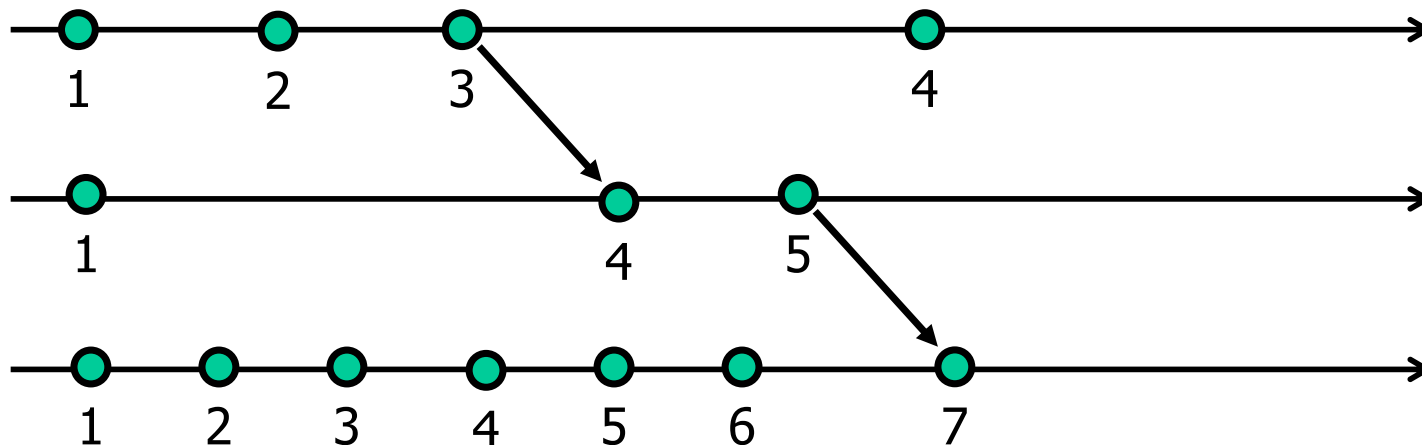
Queremos definir $\text{Clock}(e)$

$$e_1 \rightarrow e_2 \Rightarrow \text{Clock}(e_1) < \text{Clock}(e_2)$$

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 receção, atualiza-se o relógio local para o máximo do valor recebido e do relógio local [$L_i = \max(L_i, T(\text{msg}))$]



DEFINIÇÃO: RELÓGIOS LÓGICOS (LAMPORT 1978)

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

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

$$L_i := L_i + x, \quad (x > 0)$$

$T(e) := L_i$, com $T(e)$ a estampa temporal atribuída ao evento e .

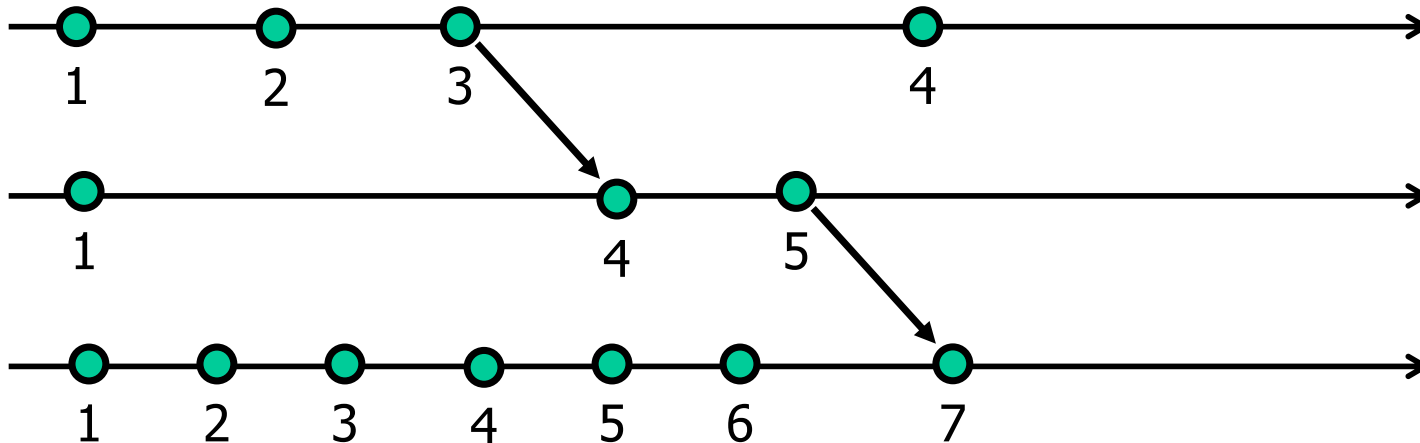
Se $e = \text{send}(m)$, aplica-se a regra anterior e envia-se (m, t) , com $t = T(\text{send}(m))$

Se $e = \text{receive}(m, t)$, faz-se $L_i = \max(L_i, t)$ e, de seguida, aplica-se a regra base (i.e., soma-se $x > 0$)

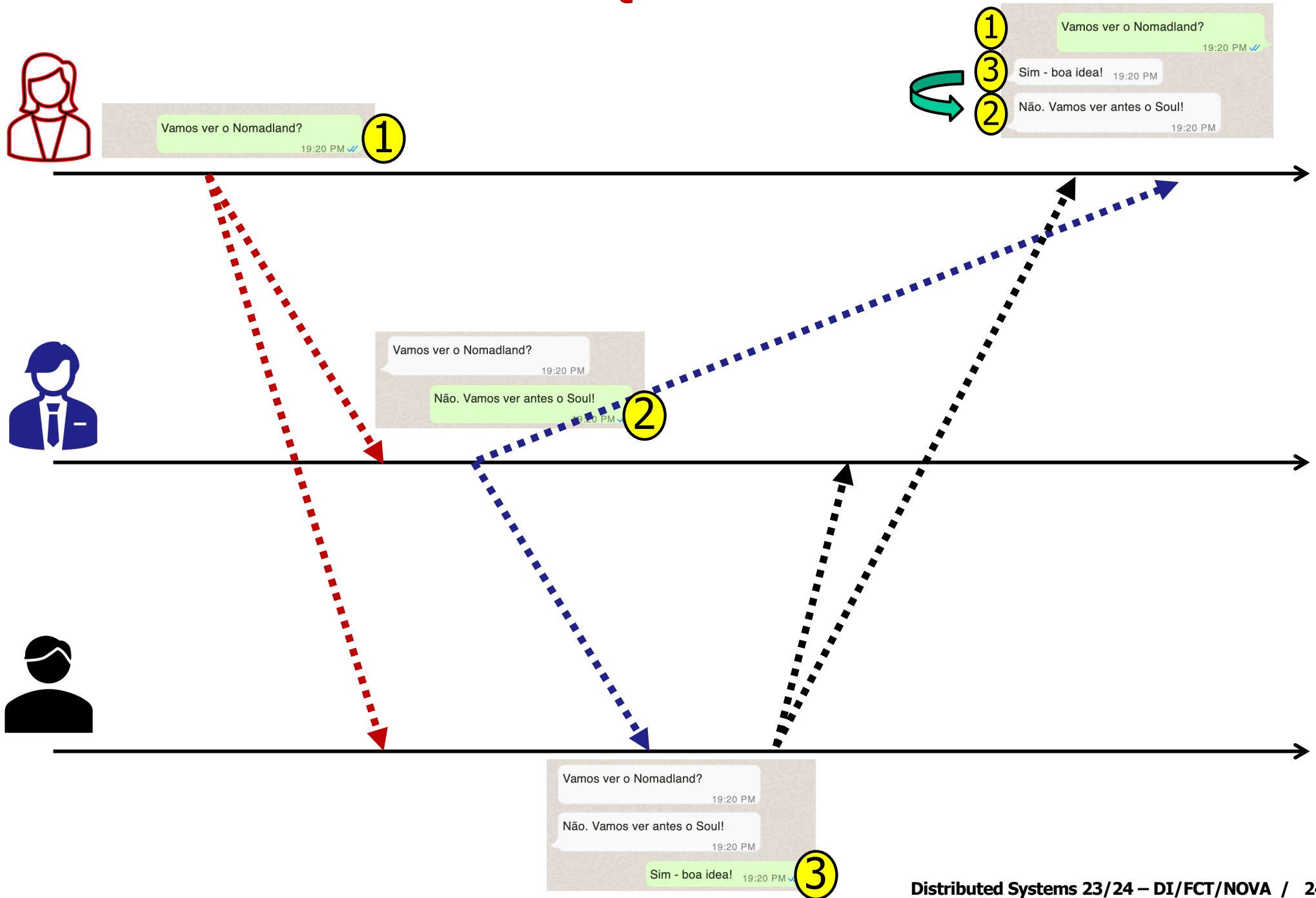
RELÓGIOS LÓGICOS: PROPRIEDADES



1. Dados dois eventos, e_1 e e_2 , $e_1 \rightarrow e_2 \Rightarrow C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2)$



ORDENAR EVENTOS? PORQUÊ?



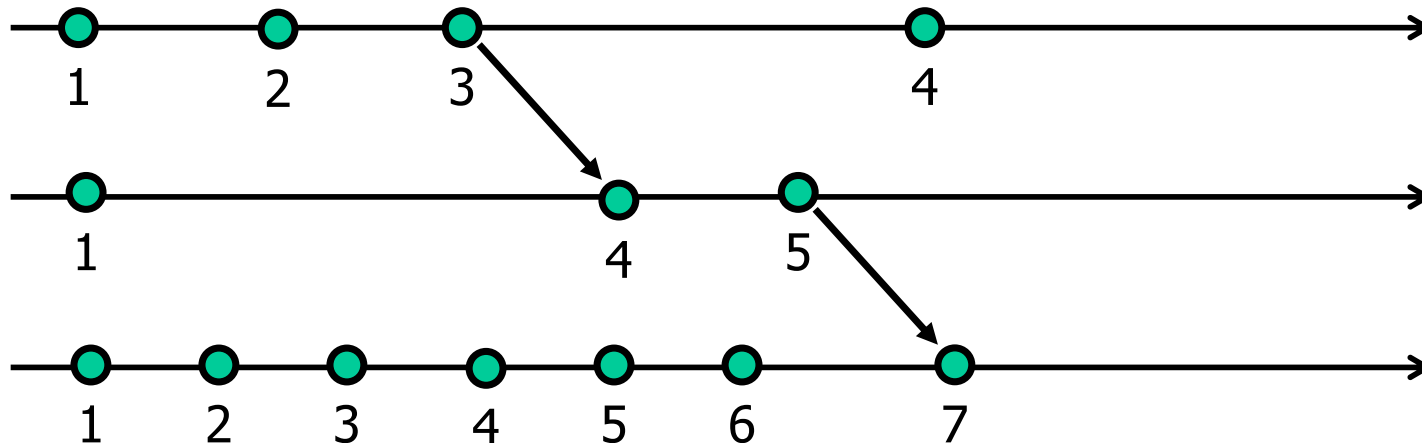
RELÓGIOS LÓGICOS + ORDEM TOTAL

Por vezes é importante estabelecer uma ordem total entre os eventos que respeite a causalidade.

Apenas os relógios lógicos não o permitem. Porquê?

Porque existem múltiplos eventos com o mesmo identificador.

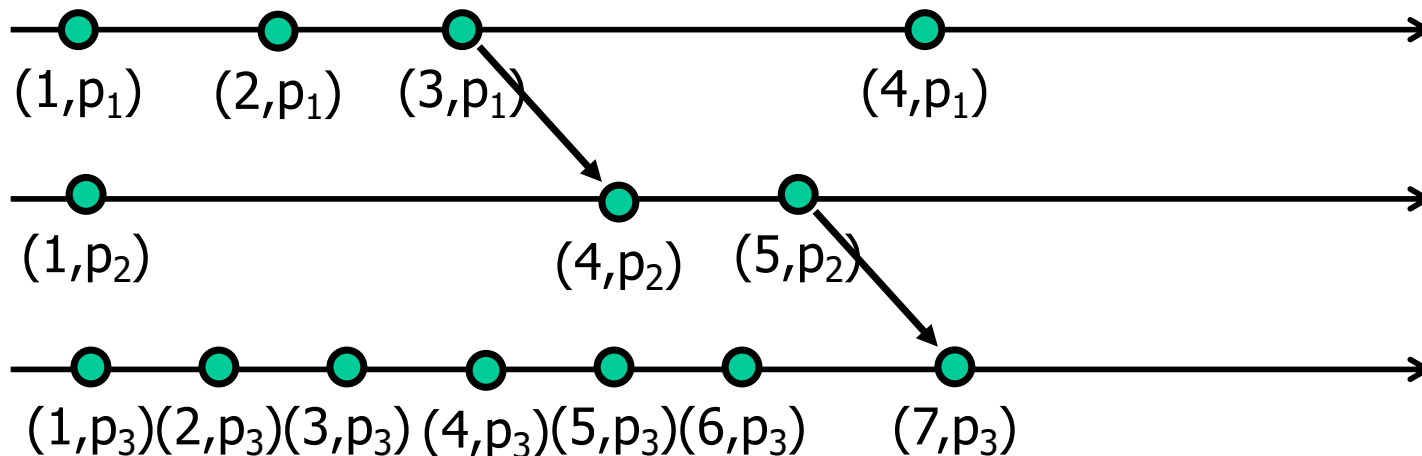
Como resolver?



RELÓGIOS LÓGICOS + ORDEM TOTAL

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

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



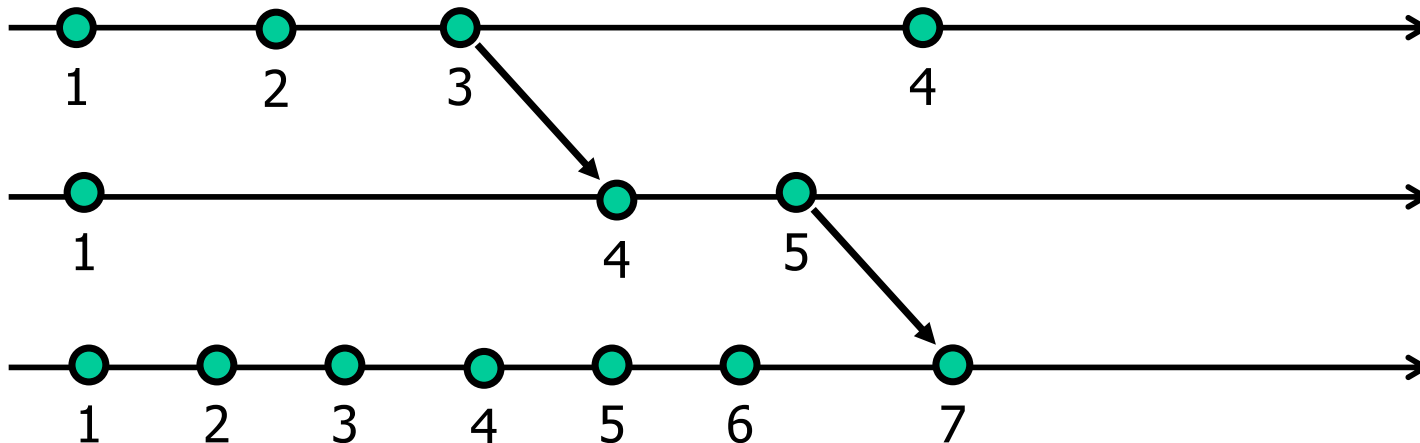
RELÓGIOS LÓGICOS: PROPRIEDADES





1. Dados dois eventos, e_1 e e_2 , $e_1 \rightarrow e_2 \Rightarrow C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2)$
2. Dados dois eventos, e_1 e e_2 , $C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2) \Rightarrow e_1 \rightarrow e_2$?

Em certas situações é interessante saber se um evento aconteceu antes de outro, i.e., se o relógio de e_1 for menor do que o relógio de e_2 é porque e_1 aconteceu antes de e_2 .

Verdade para os relógios físicos? (se completamente sincronizados)

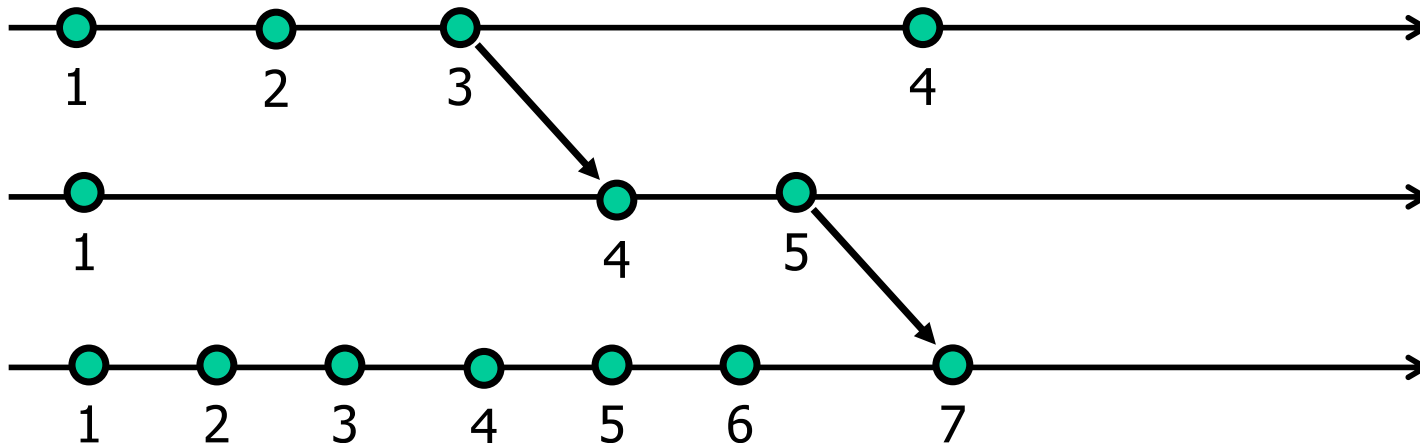


RELÓGIOS LÓGICOS: PROPRIEDADES

1. Dados dois eventos, e_1 e e_2 , $e_1 \rightarrow e_2 \Rightarrow C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2)$ 
2. Dados dois eventos, e_1 e e_2 , $C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2) \Rightarrow e_1 \rightarrow e_2$? 

Em certas situações é interessante saber se um evento aconteceu antes de outro, i.e., se o relógio de e_1 for menor do que o relógio de e_2 é porque e_1 aconteceu antes de e_2 .

E para os relógios lógicos?



NA AULA DE HOJE

Tempo em sistemas distribuídos

- Motivação para ordenar eventos
- Relógios lógicos
- **História causal e vetor versão**

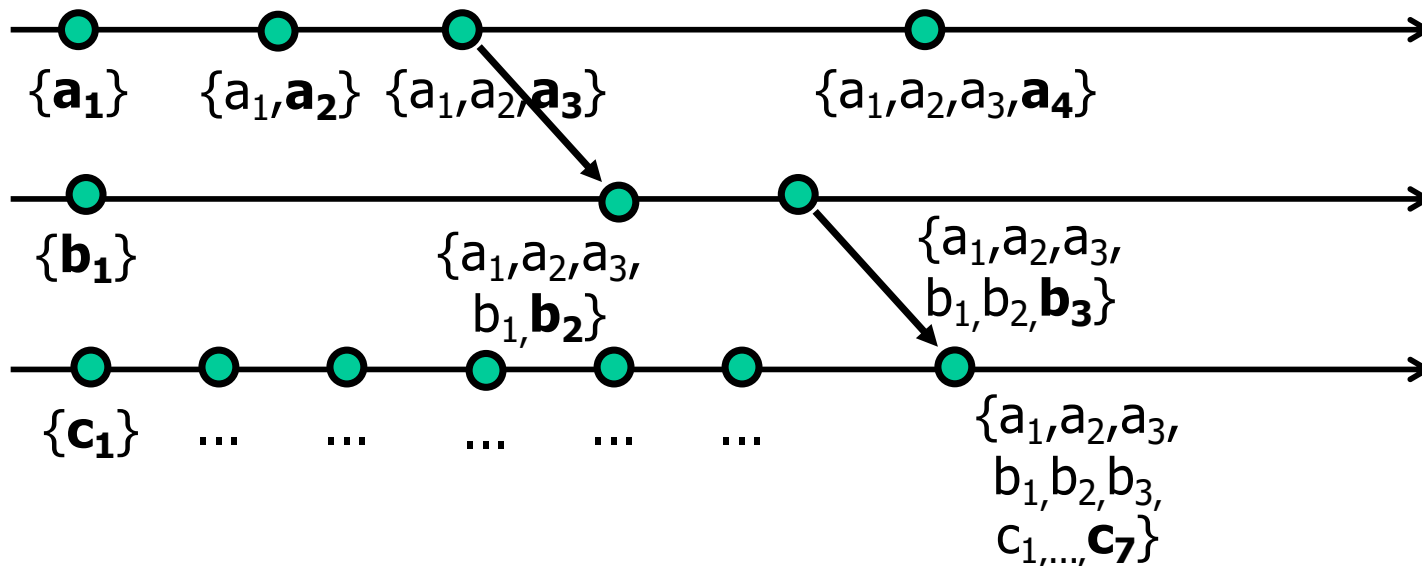
OBJETIVO

Desenvolver mecanismo que substitua relógios físico e permita:

1. Dados dois eventos, e_1 e e_2 , $e_1 \rightarrow e_2 \Rightarrow C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2)$
2. Dados dois eventos, e_1 e e_2 , $C_{\text{lock}}(e_1) < C_{\text{lock}}(e_2) \Rightarrow e_1 \rightarrow e_2$

HISTÓRIA CAUSAL

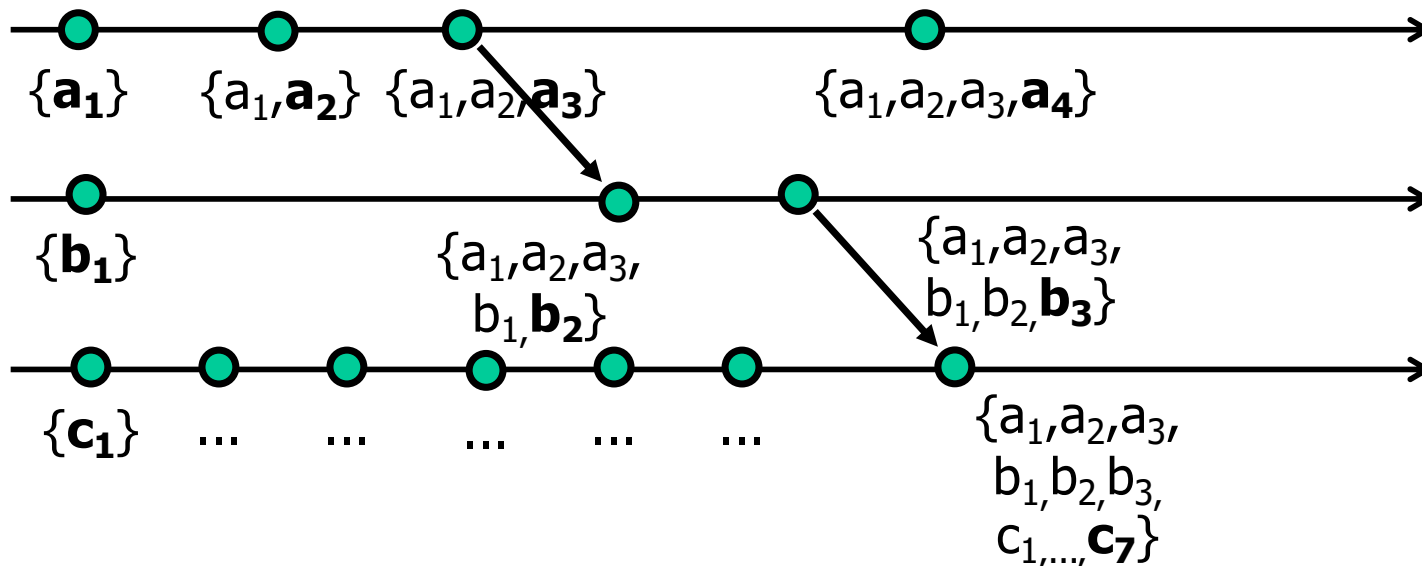
Se se quer saber o que aconteceu antes, porque não manter para cada evento o conjunto de eventos que ocorreram antes desse evento?



HISTÓRIA CAUSAL

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

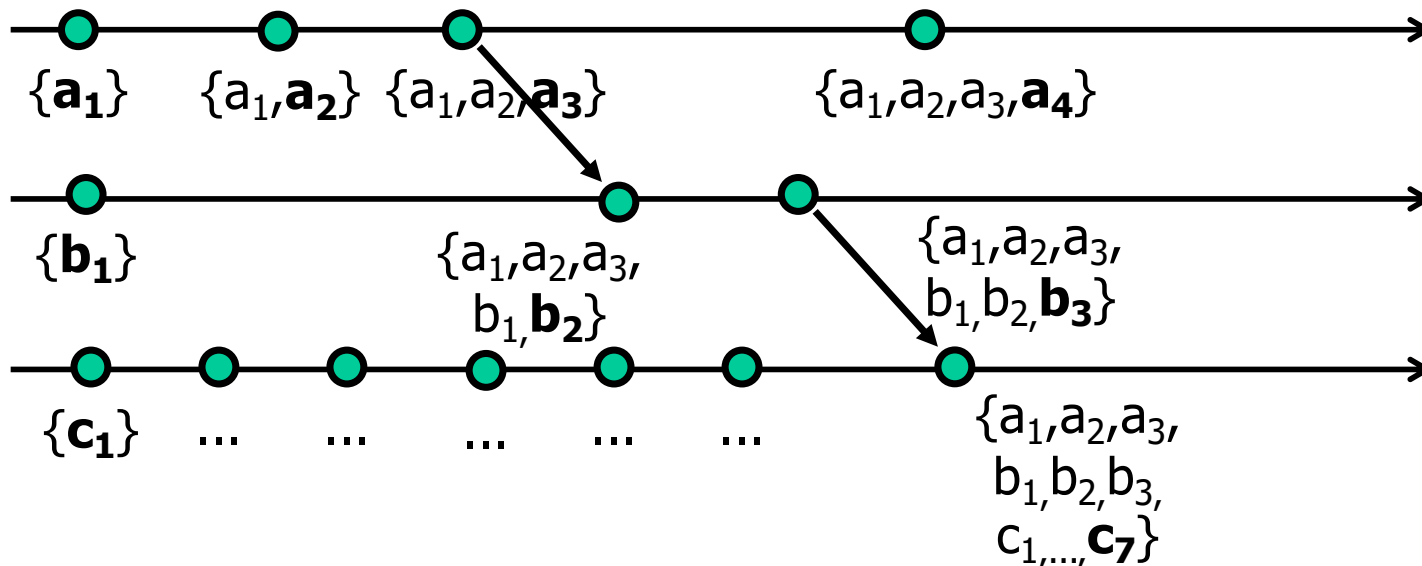
Cada evento, e , é etiquetado com um identificador, $E(e)$; a história causal desse evento é a reunião desse evento com os eventos anteriores (uma mensagem propaga a história no momento da emissão).



HISTÓRIA CAUSAL

Como comparar duas histórias causais?

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



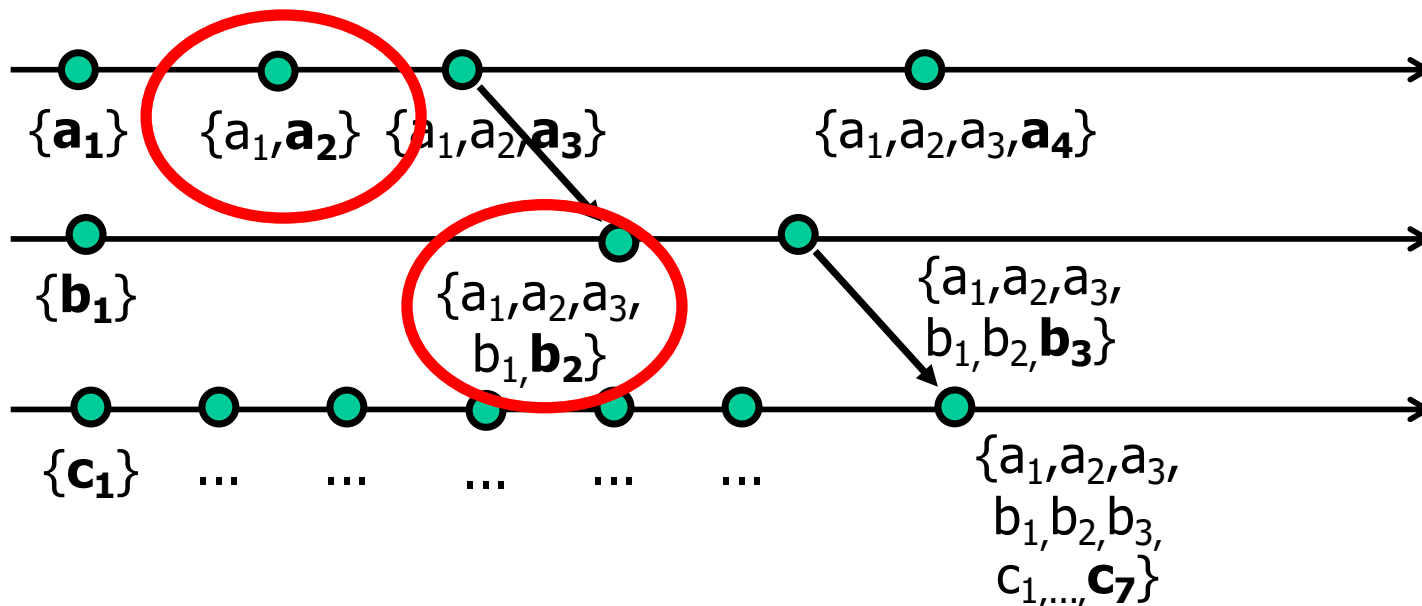
HISTÓRIA CAUSAL

Como comparar duas histórias causais?

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

Exemplo 1

$H(a_2) < H(b_2) \Leftrightarrow \{a_1, a_2\} \subset \{a_1, a_2, a_3, b_1, b_2\}$



HISTÓRIA CAUSAL

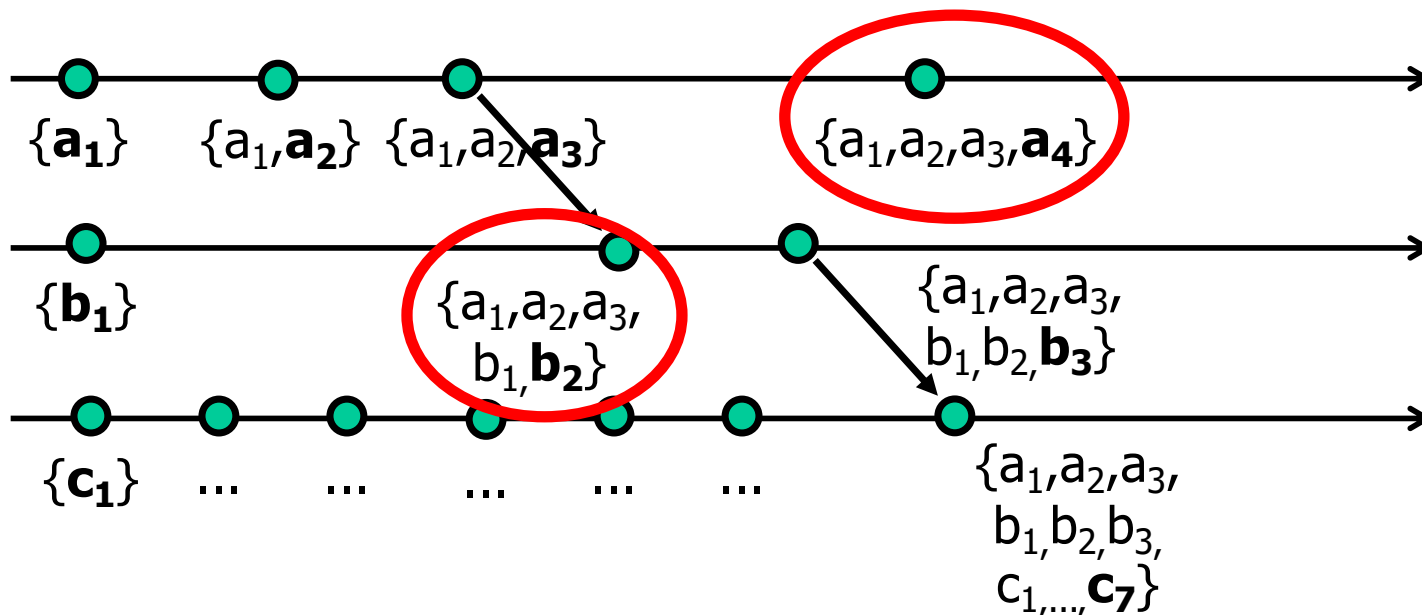
Como comparar duas histórias causais?

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

Exemplo 2

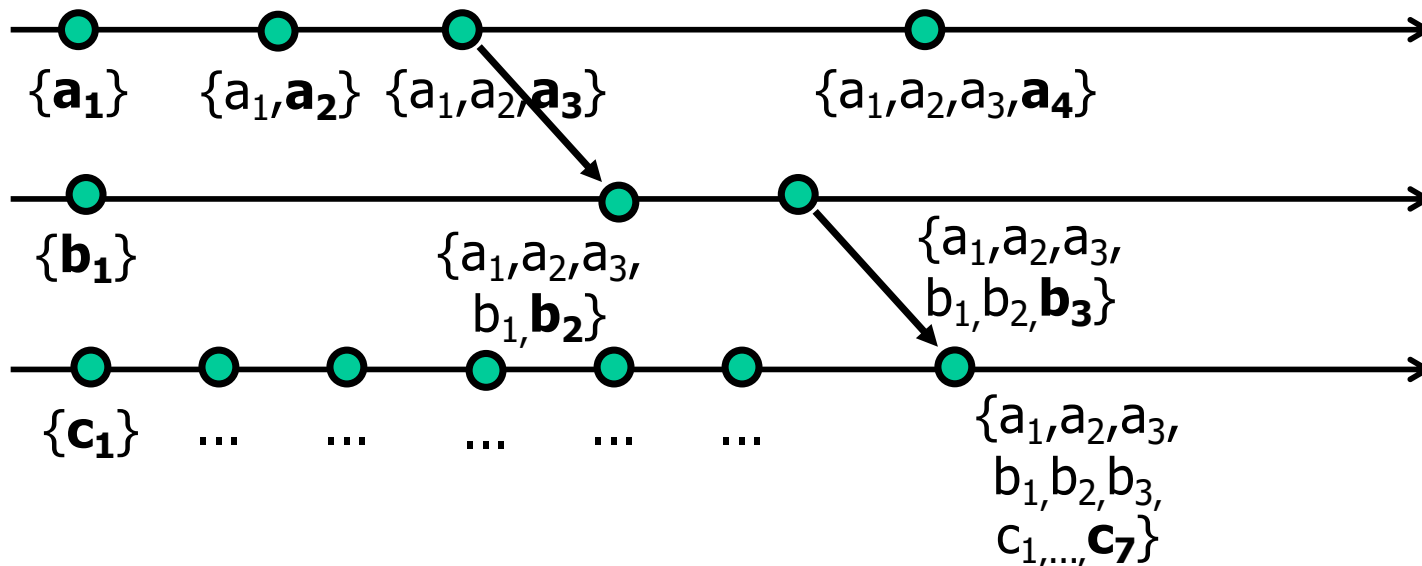
$H(a_4) \not\prec H(b_2) \wedge H(b_2) \not\prec H(a_4) \Leftrightarrow$

$\{a_1, a_2, a_3, a_4\} \not\subset \{a_1, a_2, a_3, b_1, b_2\} \wedge \{a_1, a_2, a_3, b_1, b_2\} \not\subset \{a_1, a_2, a_3, a_4\}$



HISTÓRIA CAUSAL

1. Dados dois eventos, \mathbf{e}_1 e \mathbf{e}_2 , $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow \mathbf{C}_{\text{lock}}(\mathbf{e}_1) < \mathbf{C}_{\text{lock}}(\mathbf{e}_2)$
 - $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow \mathbf{H}(\mathbf{e}_1) \subset \mathbf{H}(\mathbf{e}_2) \wedge \mathbf{H}(\mathbf{e}_1) \neq \mathbf{H}(\mathbf{e}_2)$
2. Dados dois eventos, \mathbf{e}_1 e \mathbf{e}_2 , $\mathbf{C}_{\text{lock}}(\mathbf{e}_1) < \mathbf{C}_{\text{lock}}(\mathbf{e}_2) \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$
 - $\mathbf{H}(\mathbf{e}_1) \subset \mathbf{H}(\mathbf{e}_2) \wedge \mathbf{H}(\mathbf{e}_1) \neq \mathbf{H}(\mathbf{e}_2) \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$
 - [ou mais simples: $\mathbf{e}_1 \in \mathbf{H}(\mathbf{e}_2) \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$]



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 $e = \text{send}(m)$, aplica-se a regra anterior e envia-se (m, t) , com $t = H(e)$

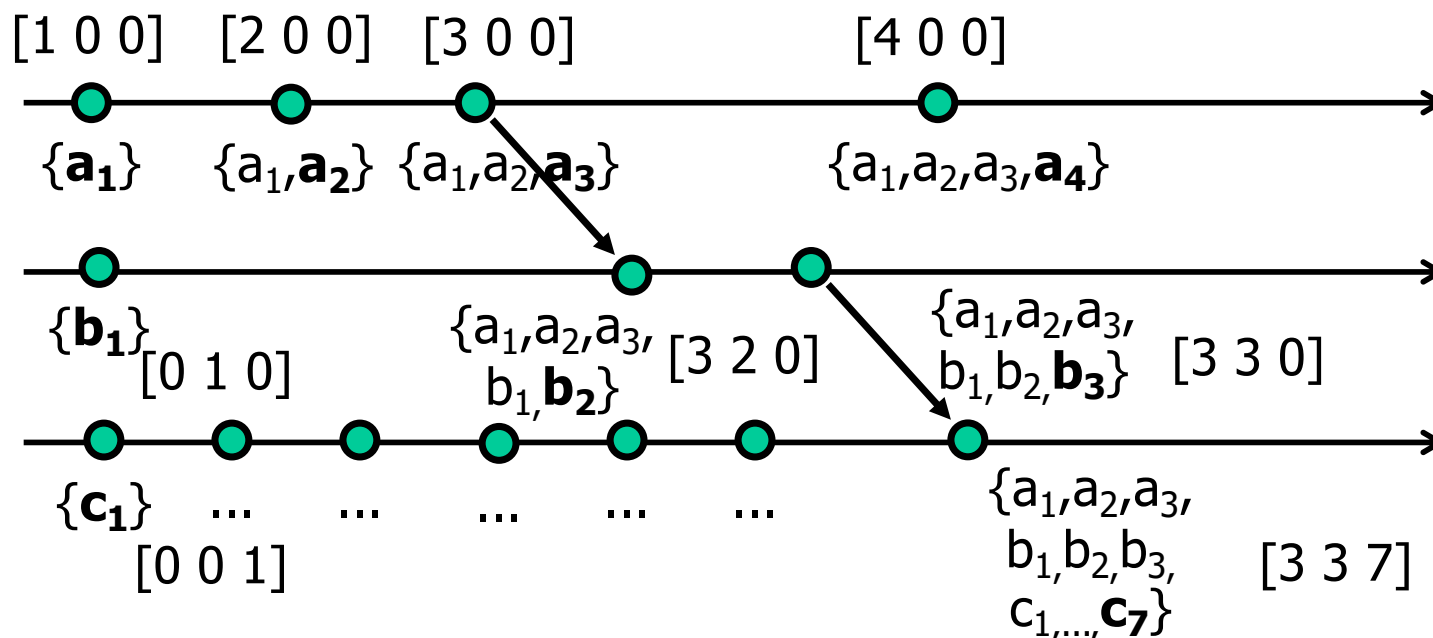
Se $e^i_n = \text{receive}(m, t)$, $H(e^i_n) = H(e^i_{n-1}) \cup t \cup \{e^i_n\}$

RELÓGIO VETORIAL

Um relógio vetorial, $V(e)$, é uma **forma eficiente de representar uma história causal**.

Usa um vetor que **em cada posição i** tem o contador do último evento dum processo i (todos os eventos anteriores desse processo são necessariamente conhecidos).

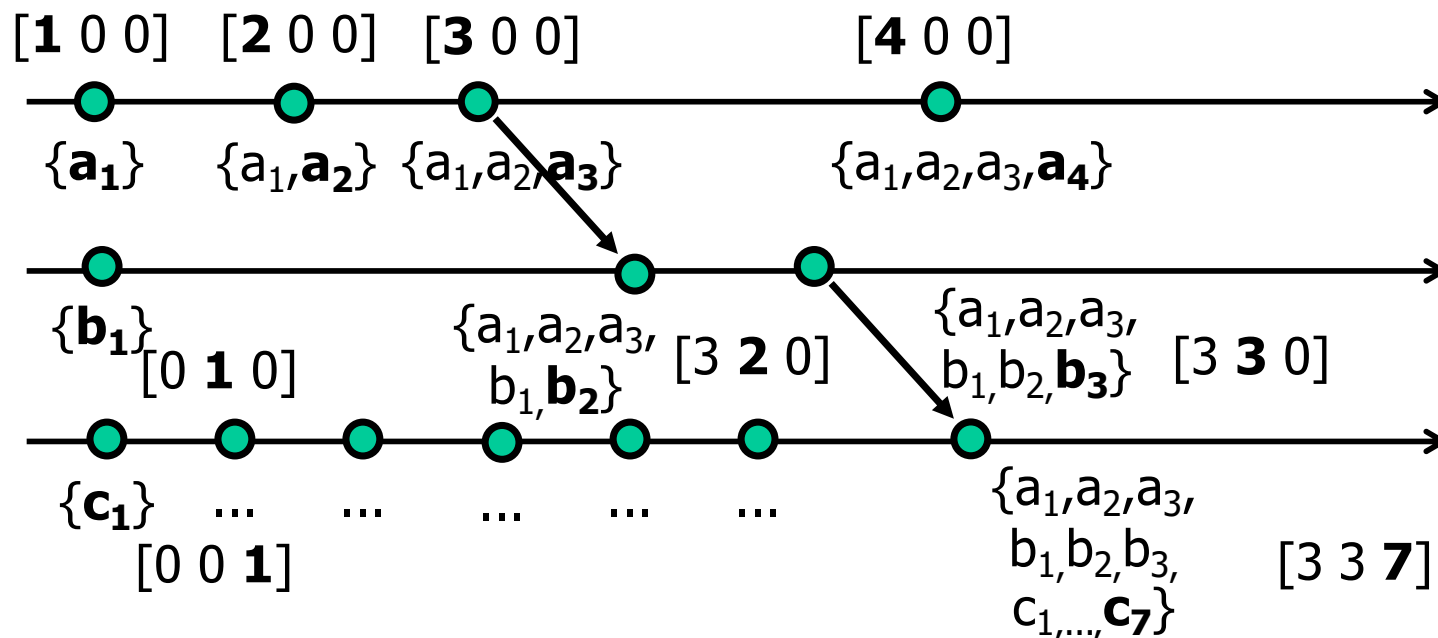
[a b c]



RELÓGIO VETORIAL

Como é que se comparam dois relógios vetoriais ?

- $\mathbf{V1} < \mathbf{V2} \iff \forall i, \mathbf{V1}[i] \leq \mathbf{V2}[i] \wedge \exists j: \mathbf{V1}[j] \neq \mathbf{V2}[j]$



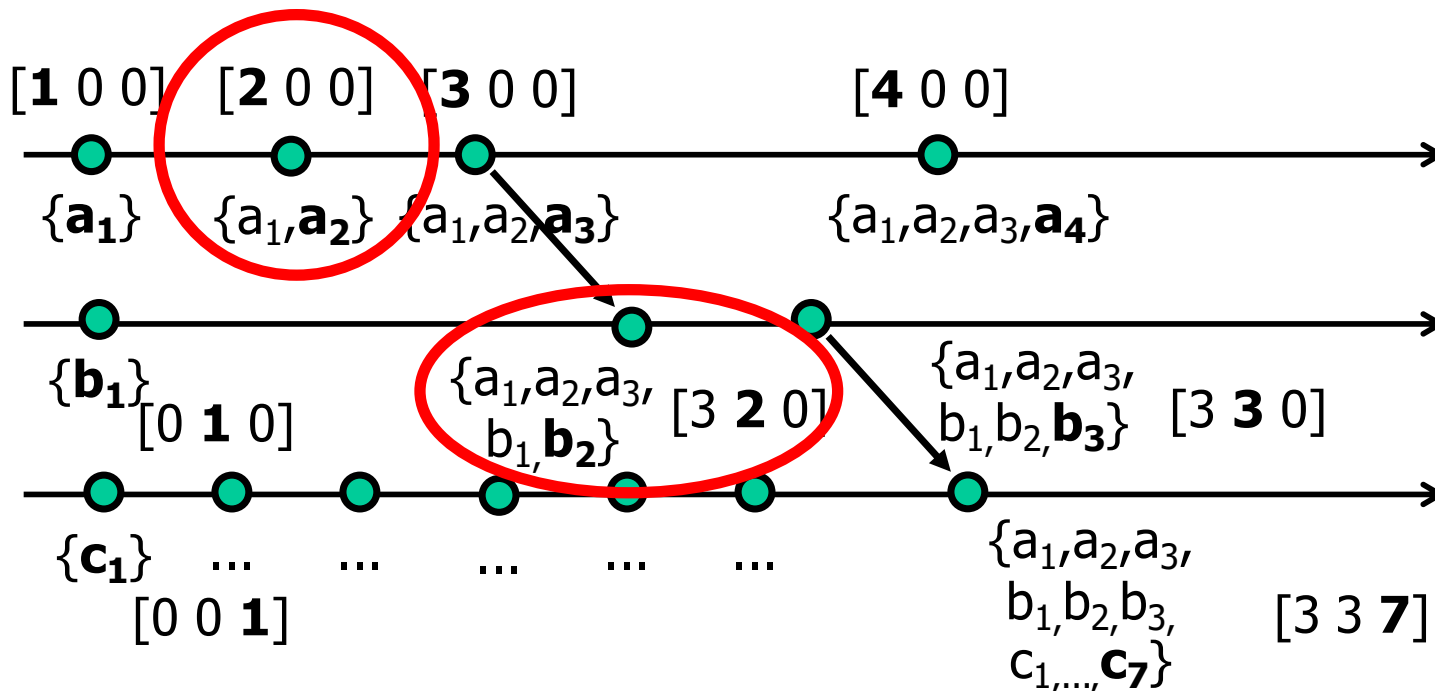
RELÓGIO VETORIAL

Como é que se comparam dois relógios vetoriais ?

- $\mathbf{V1} < \mathbf{V2} \iff \forall i, \mathbf{V1}[i] \leq \mathbf{V2}[i] \wedge \exists j: \mathbf{V1}[j] \neq \mathbf{V2}[j]$

Exemplo 1

$V(a_2) < V(b_2) \iff [2 \ 0 \ 0] < [3 \ 2 \ 0] \quad (2 < 3, 0 < 2, 0 \leq 0)$



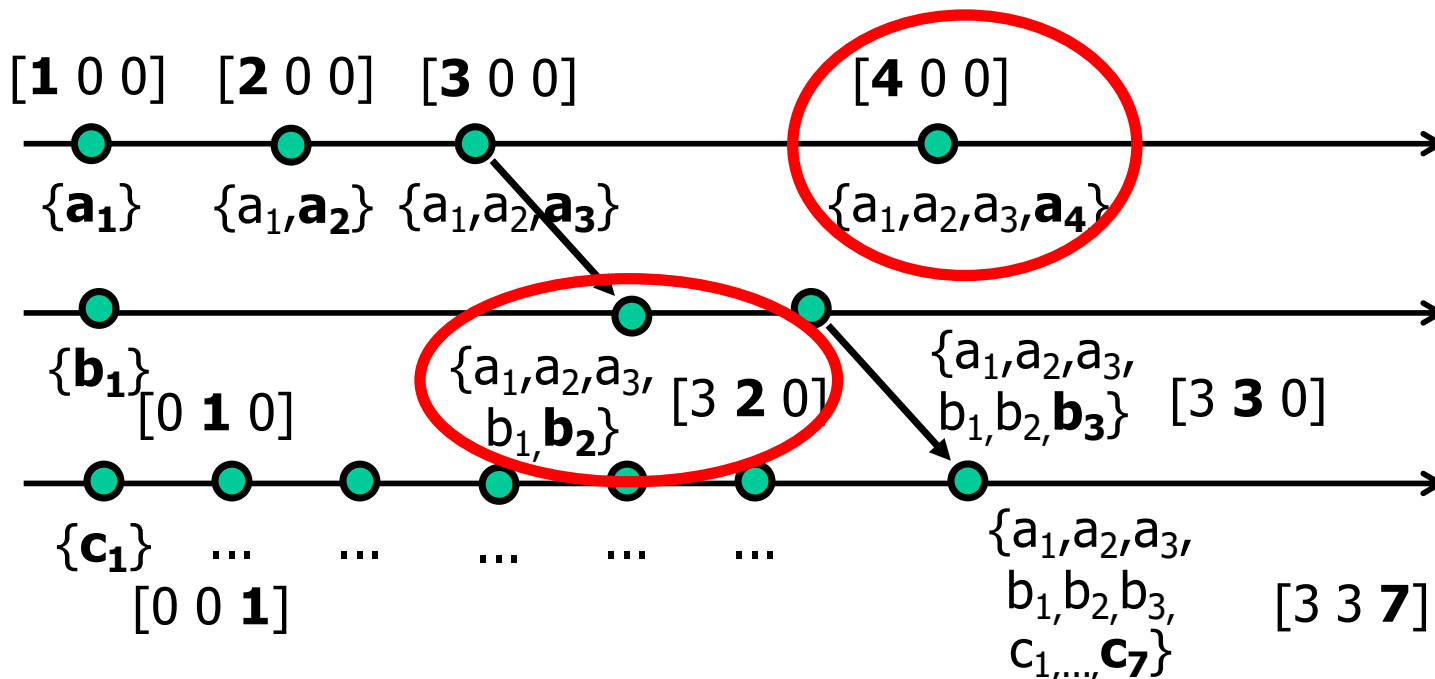
RELÓGIO VETORIAL

Como é que se comparam dois relógios vetoriais ?

- $V1 < V2 \forall i, V1[i] \leq V2[i] \wedge \exists j: V1[j] \neq V2[j]$

Exemplo 2

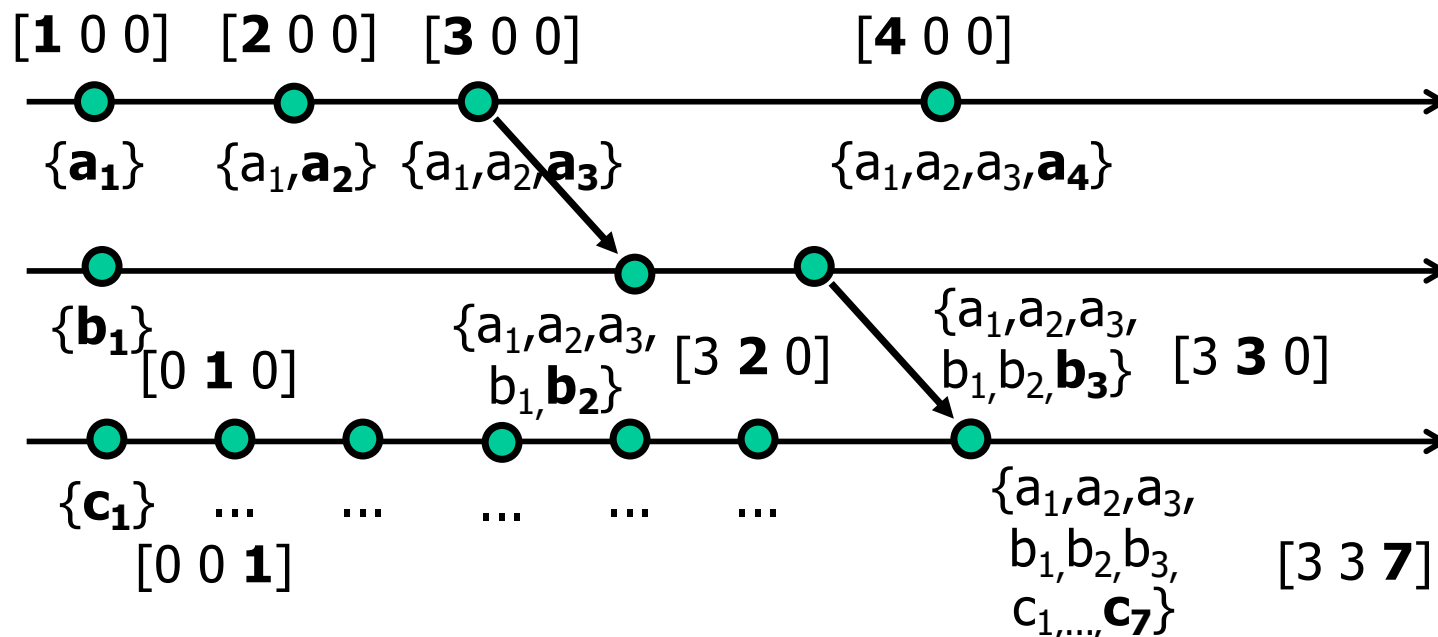
$$V(a_4) \not\prec V(b_2) \wedge V(b_2) \not\prec V(a_4) \Leftrightarrow [4 \ 0 \ 0] \parallel [3 \ 2 \ 0] \quad (4 > 3, 0 < 2, 0 \leq 0)$$



RELÓGIO VETORIAL



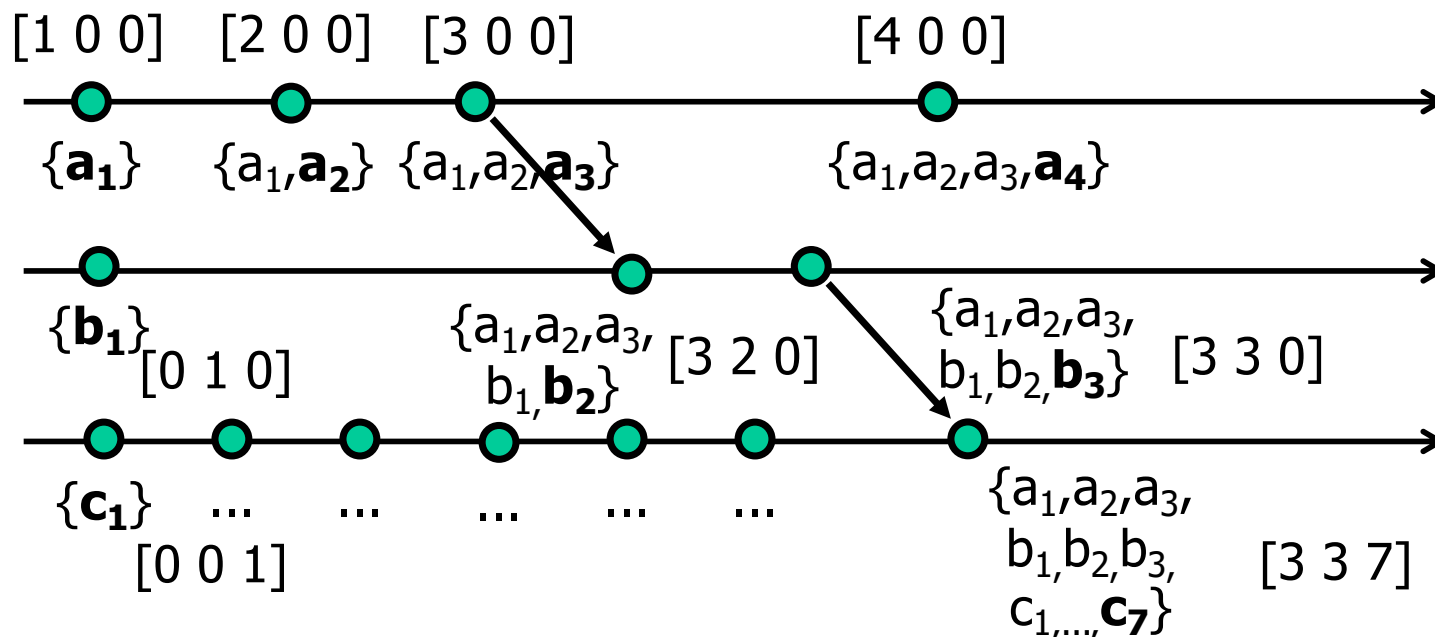
1. Dados dois eventos, \mathbf{e}_1 e \mathbf{e}_2 , $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow C_{\text{lock}}(\mathbf{e}_1) < C_{\text{lock}}(\mathbf{e}_2)$
 - $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow H(\mathbf{e}_1) \subset H(\mathbf{e}_2) \wedge H(\mathbf{e}_1) \neq H(\mathbf{e}_2)$ (com histórias causais)
 - $\mathbf{e}_1 \rightarrow \mathbf{e}_2 \Rightarrow \forall i, V(\mathbf{e}_1)[i] \leq V(\mathbf{e}_2)[i] \wedge \exists j: V(\mathbf{e}_1)[j] \neq V(\mathbf{e}_2)[j]$



RELÓGIO VETORIAL



2. Dados dois eventos, \mathbf{e}_1 e \mathbf{e}_2 , $\mathbf{Clock}(\mathbf{e}_1) < \mathbf{Clock}(\mathbf{e}_2) \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$
- $H(\mathbf{e}_1) \subset H(\mathbf{e}_2) \wedge H(\mathbf{e}_1) \neq H(\mathbf{e}_2) \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$ (com histórias causais)
 - $\forall i, V(\mathbf{e}_1)[i] \leq V(\mathbf{e}_2)[i] \wedge \exists j: V(\mathbf{e}_1)[j] \neq V(\mathbf{e}_2)[j] \Rightarrow \mathbf{e}_1 \rightarrow \mathbf{e}_2$

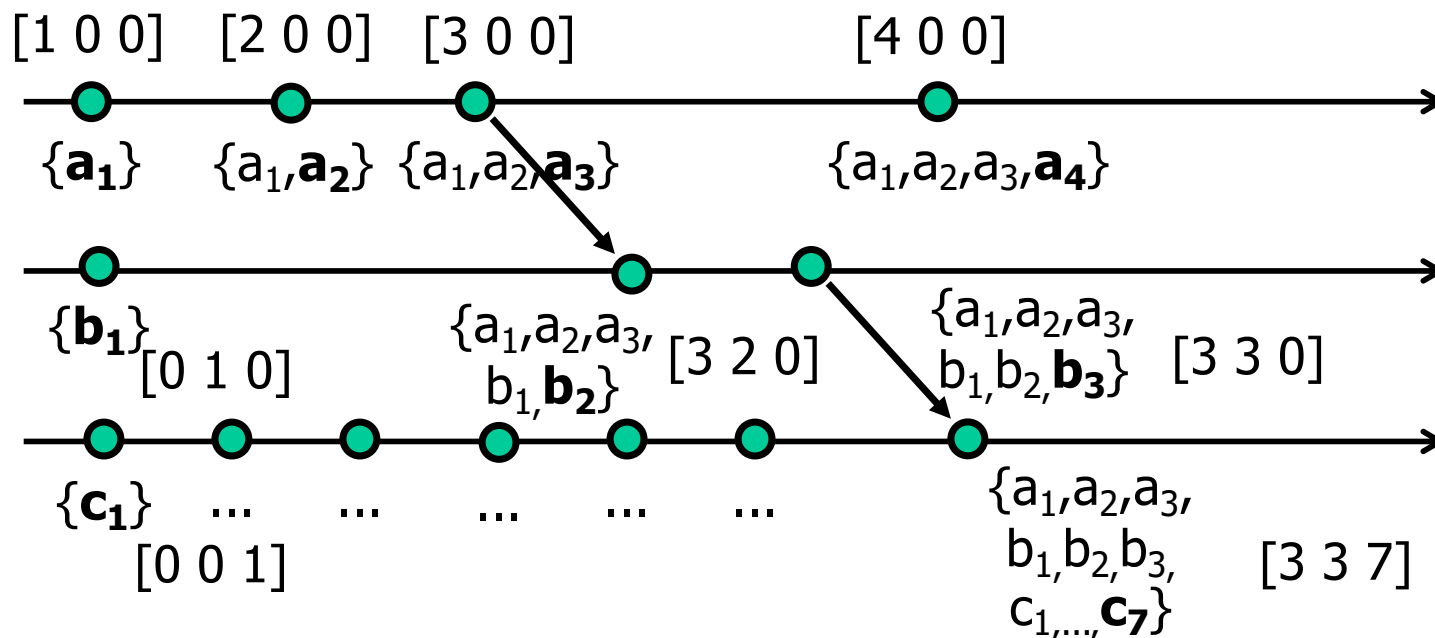


RELÓGIO VETORIAL : CONCORRÊNCIA

Dados dois eventos, e_1 e e_2 ,

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

- $\neg(H(e_1) \subset H(e_2)) \wedge \neg(H(e_2) \subset H(e_1)) \Rightarrow e_1 \parallel e_2$ (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: $V_i[i] := V_i[i]+1$.
 $C(e)=V_i$
- Se $e=\text{send}(m)$, aplica-se a regra anterior e envia-se (m,t) , com $t=C(\text{send}(m))$
- Se $e=\text{receive}(m,t)$ no processo i , faz-se $V_i[j]=\max(V_i[j],t[j]), \forall j$ e, de seguida, aplica-se a regra base

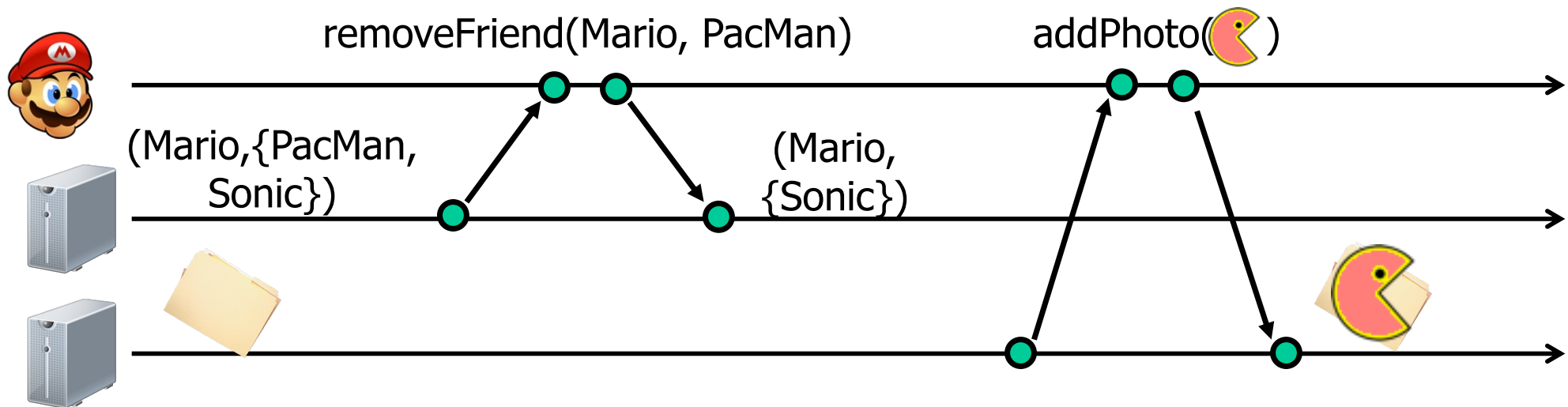
Dados dois relógios vectoriais $V1$ e $V2$, tem-se:

- $V1=V2$, sse $V1[i]=V2[i], \forall i$
- $V1 \leq V2$, sse $V1[i] \leq V2[i], \forall i$
- $V1 < V2$, sse $V1 \leq V2 \wedge V1 \neq V2$

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

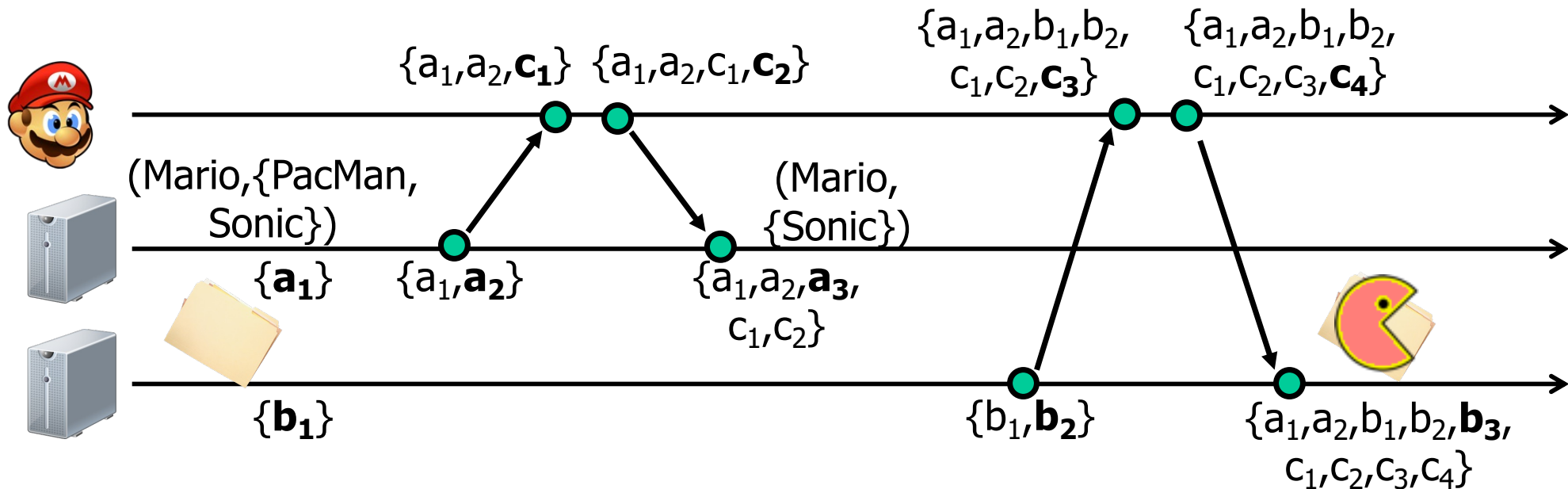


HÁ EVENTOS E EVENTOS

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 que devem ser vistos.

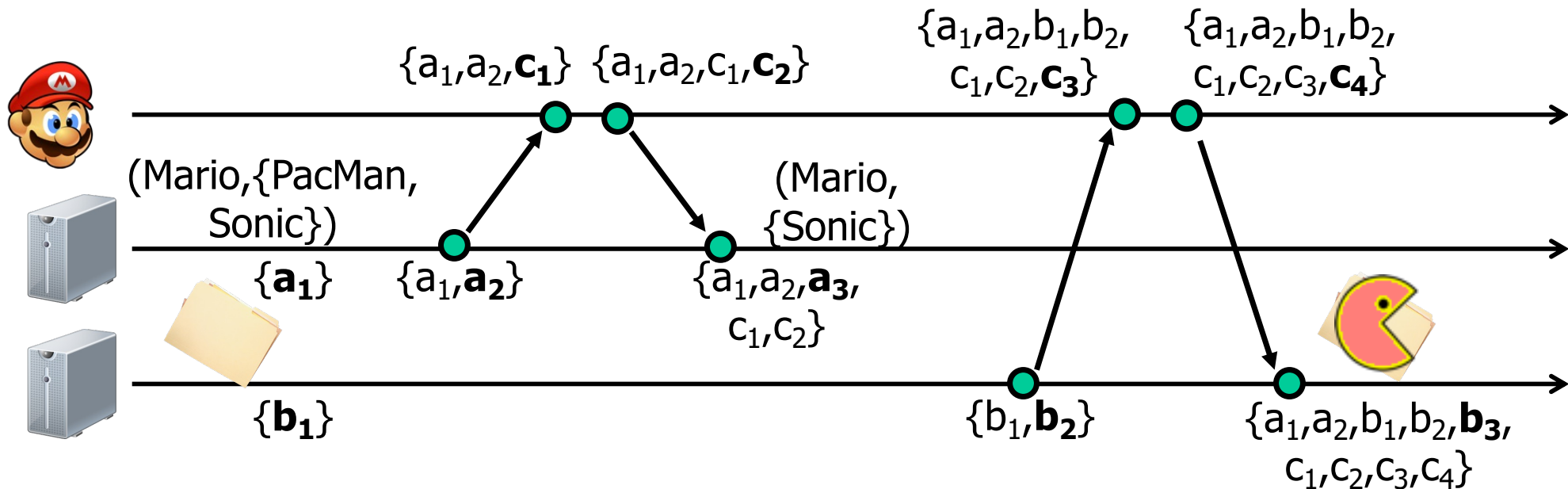


HÁ EVENTOS E EVENTOS

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

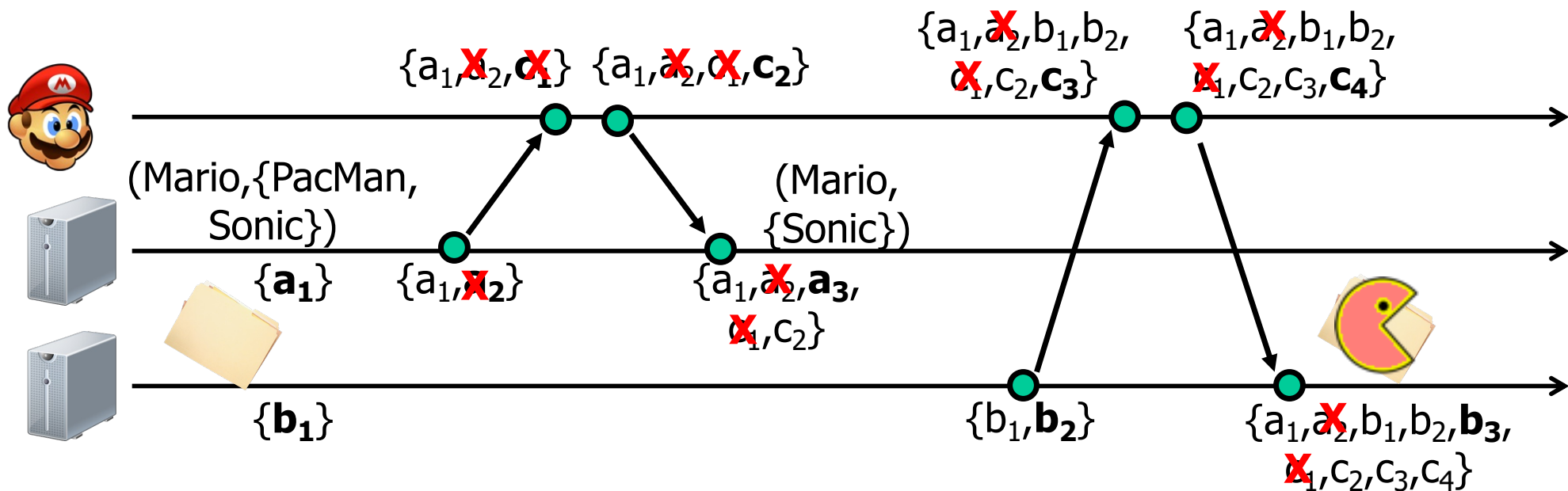
1. Remover X da lista de amigos
2. Adicionar fotografia

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



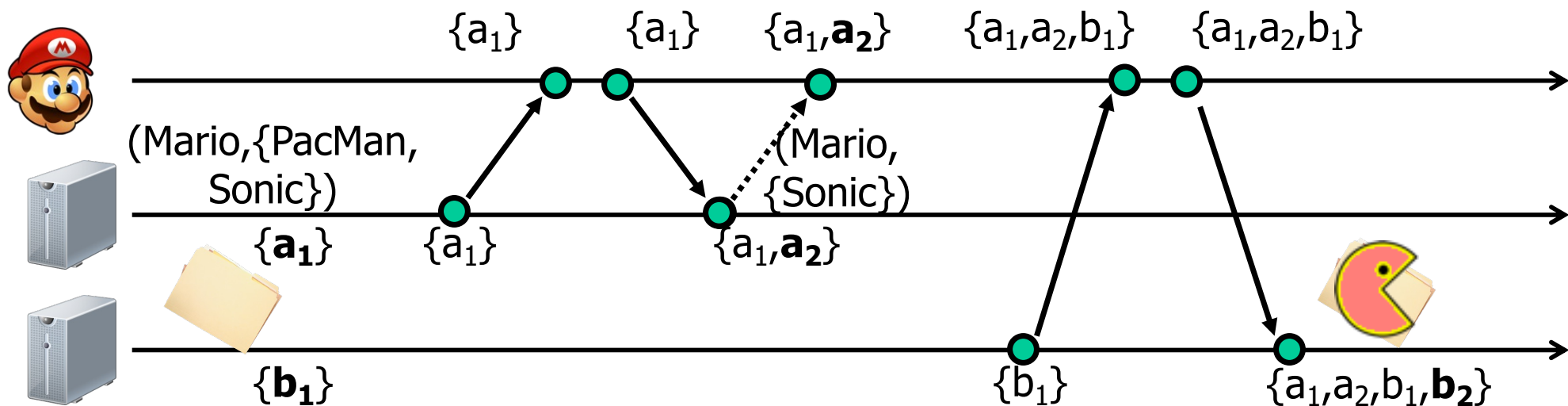
HÁ EVENTOS E EVENTOS

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



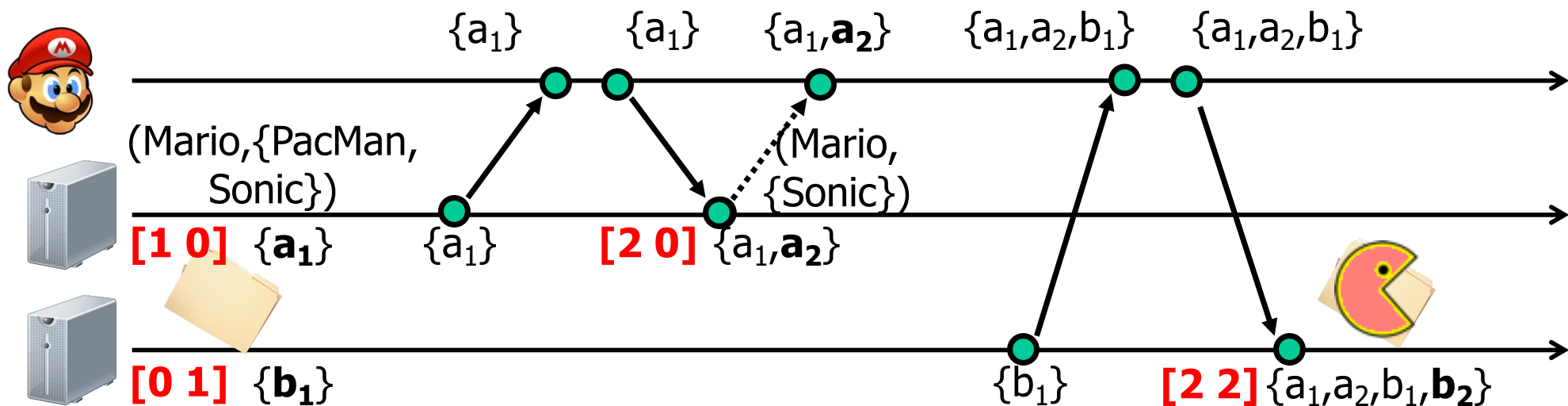
HÁ EVENTOS E EVENTOS

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

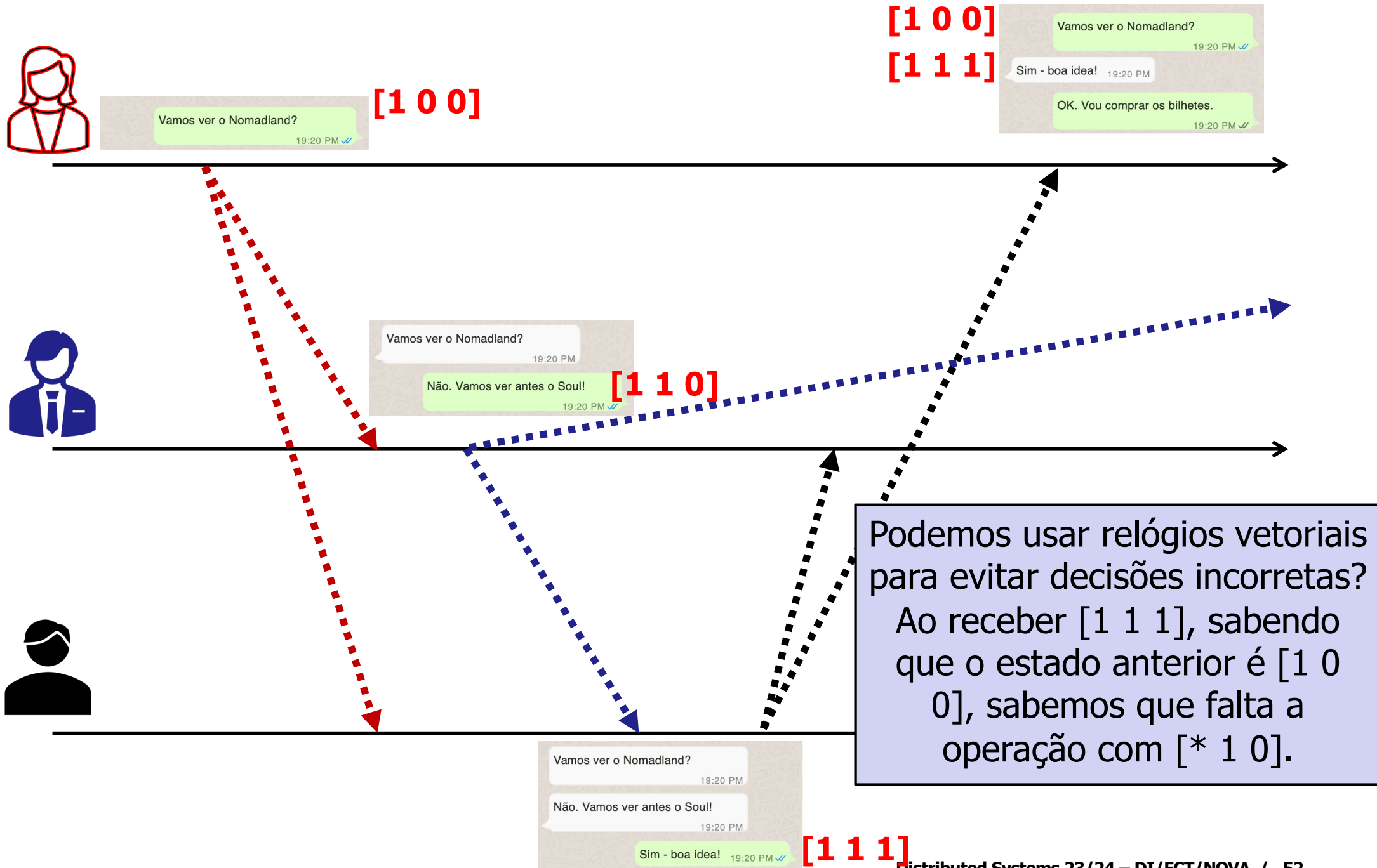


VETOR VERSÃO

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)



ORDENAR EVENTOS? PORQUÊ?



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: $VV_i[i] := VV_i[i]+1$. $C(e)=VV_i$
- Para um evento **e** não importantes, tem-se $C(e)=VV_i$. 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 $t=C(\text{send}(\mathbf{m}))$.
- Se **e=receive(m,t)** no processo i , faz-se $VV_i[j]=\max(VV_i[j],t[j])$, $\forall j$.

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

- $VV1=VV2$, sse $VV1[i]=VV2[i], \forall i$
- $VV1 \leq VV2$, sse $VV1[i] \leq VV2[i], \forall i$
- $VV1 < VV2$, sse $VV1 \leq VV2 \wedge VV1 \neq VV2$

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