Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Relógios Lógicos

Sincronização de Relógios

- Não se pode sincronizar perfeitamente relógios em um sistema distribuído

- Há algoritmos para sincronização externa e interna

- Em sistemas síncronos, é possível determinar o limite máximo para o desvio entre dois relógios

- Entretanto, a maioria dos sistemas são assíncronos

Em geral, não se pode usar o tempo físico para descobrir a ordem entre dois eventos arbitrários ocorrendo em processos diferentes

Relógios de Lamport

- Lamport em 1974 sugere a utilização de relógios lógicos para observar a causalidade entre eventos em processos distribuídos

- Não há garantia (nem necessidade) de ter todos os processos com o mesmo horário de referência

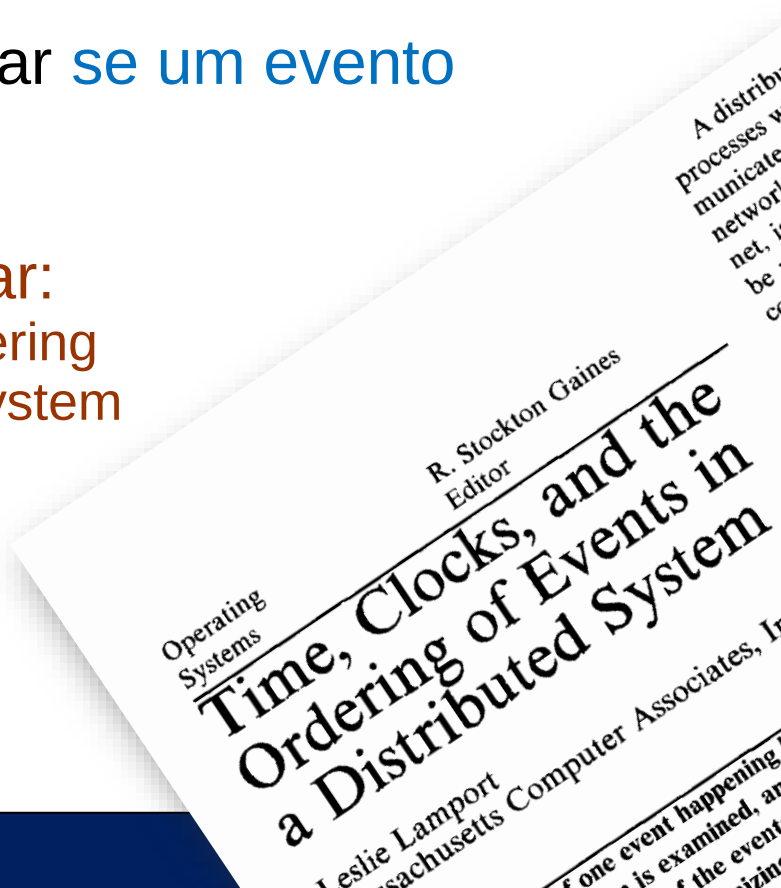
- Basta utilizar uma forma comum para indicar se um evento aconteceu antes do que outro

Leitura Complementar:

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport

Leslie Lamport



Relação *acontece antes (happens-before)*

Lamport sugere observar a ordem em que eventos acontecem:

Definida pela relação: ***acontece antes*** *(happens-before)*

$a \longrightarrow b$: Significa *"a acontece antes de b"*

1) *Se a e b são eventos locais do mesmo processo, e a acontece antes de b: C(a) < C(b), então* $a \longrightarrow b$

2) *Se a é um evento send(m) por P1 e b é evento receive(m) por P2, então* $a \longrightarrow b$

3) *Se* $a \longrightarrow b$ *e* $b \longrightarrow c$, *então* $a \longrightarrow c$

Ou seja, a relação "acontece antes" é transitiva

Relação *acontece antes (happens-before)*

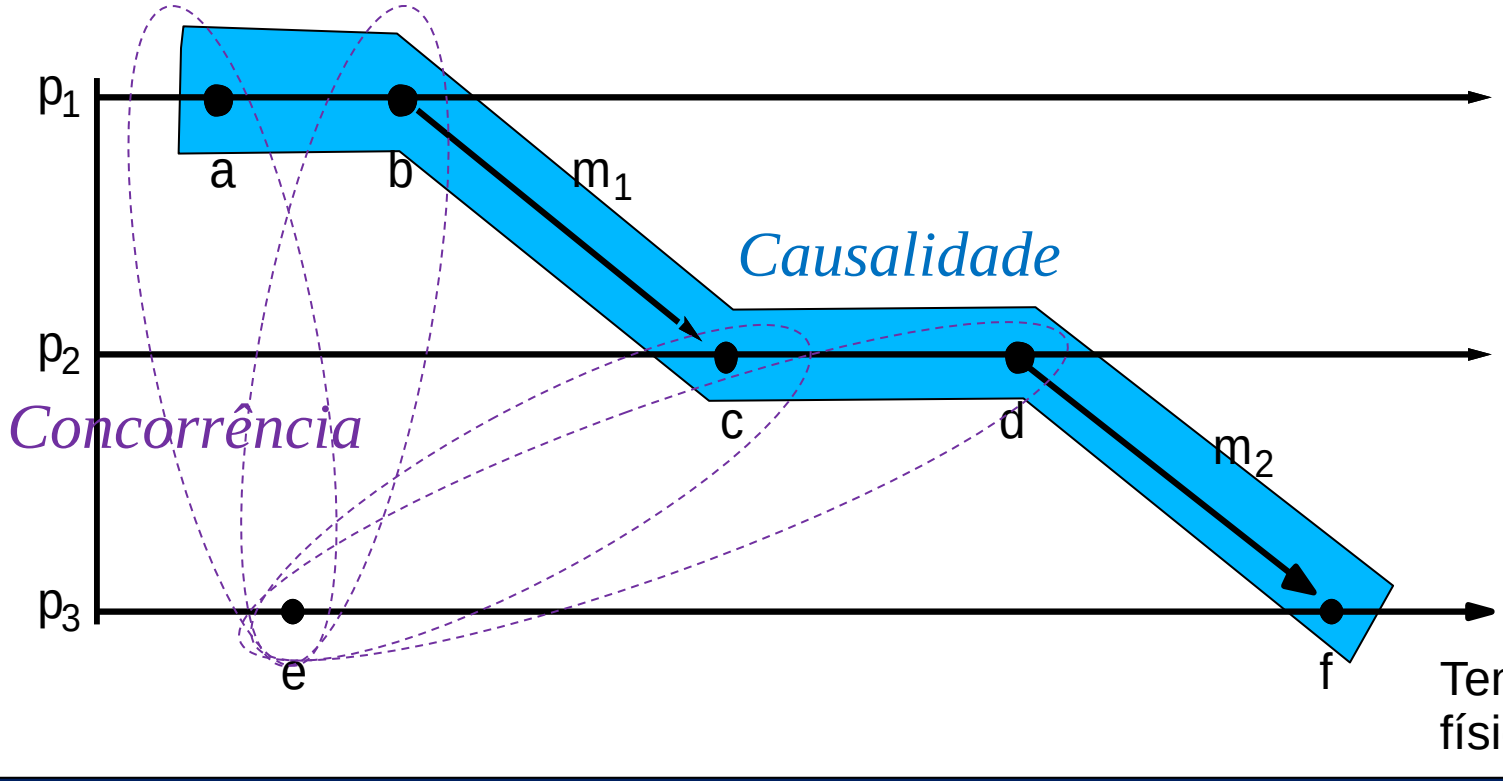
- Se x e y são eventos que acontecem em processos diferentes e estes processos não trocam mensagens, então x e y são ***eventos concorrentes***

Logo:

$x \not\longrightarrow y \;\; ou \;\; x \,||\, y$

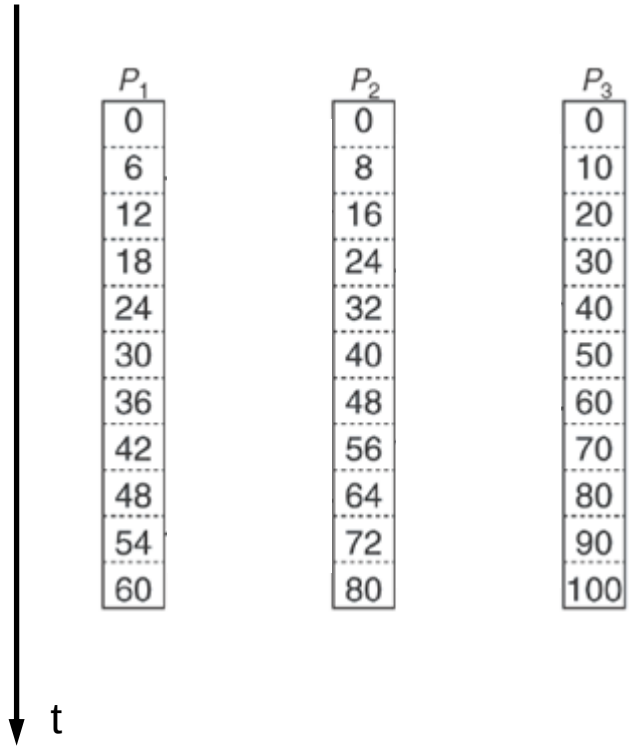
$y \not\longrightarrow x \;\; ou \;\; y \,||\, x$

Estabelece uma ordem parcial entre os eventos



$a \longrightarrow b$
$b \longrightarrow c$
$c \longrightarrow d$
$d \longrightarrow f$
$a \longrightarrow c$
$a \longrightarrow d$
$a \longrightarrow f$
$b \longrightarrow d$
$b \longrightarrow f$
$c \longrightarrow f$
$a \not\longrightarrow e$
$b \not\longrightarrow e$
...

Relógios Lógicos de Lamport



-Três processos distribuídos apresentam defasagem em seus relógios locais

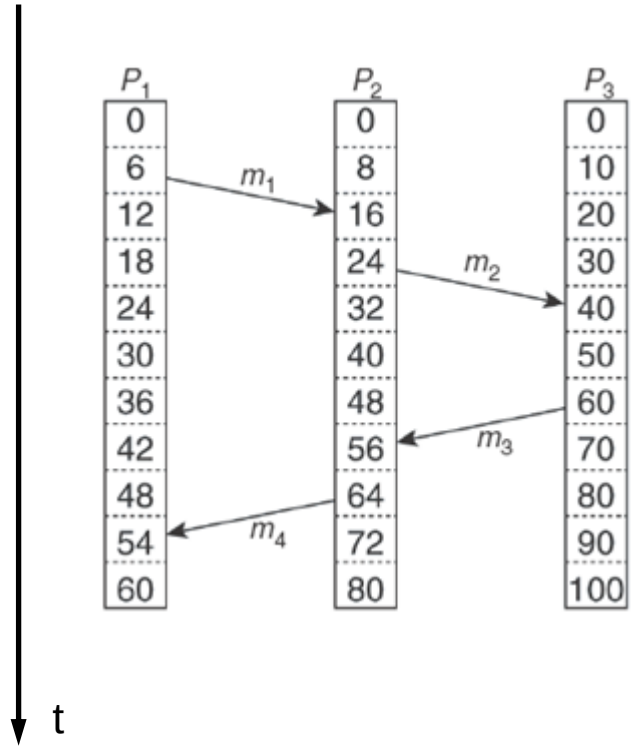
- Lamport sugere observar a ordem em que eventos acontecem:

$C(a)$ – indica uma contagem de quando a aconteceu no processo

- Portanto, se for possível identificar a ordem dos eventos, não será preciso saber a hora absoluta em que eles acontecem, basta contar ordenadamente quando os eventos ocorreram

$a \longrightarrow b$: então $C(a) < C(b)$

Relógios Lógicos de Lamport



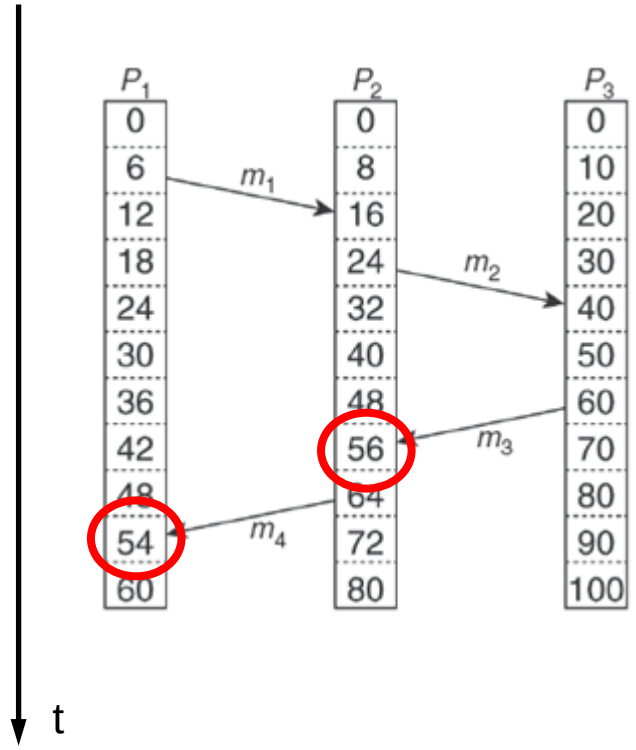
- Observe os eventos a seguir:

(a) P1 envia m1 para P2
(b) P2 recebe m1
(c) P2 envia m2 para P3
(d) P3 recebe m2
(e) P3 envia m3 para P2
(f) P2 recebe m3
(g) P2 envia m4 para P1
(h) P1 recebe m4

- Sendo ***C(evento)*** a contagem dada pela leitura do relógio local, são verificadas as propriedades?

C(a) < C (b)
C(b) < C (c)
C(c) < C (d)
C(d) < C (e)
C(e) < C (f)
C(f) < C (g)
C(g) < C (h)

Relógios Lógicos de Lamport



- Observe os eventos a seguir:

(a) P1 envia m1 para P2
(b) P2 recebe m1
(c) P2 envia m2 para P3
(d) P3 recebe m2
(e) P3 envia m3 para P2
(f) P2 recebe m3
(g) P2 envia m4 para P1
(h) P1 recebe m4

- Sendo ***C(evento)*** a contagem dada pela leitura do relógio local, são verificadas as propriedades?

C(a) < C (b)
C(b) < C (c)
C(c) < C (d)
C(d) < C (e)
C(e) < C (f)
C(f) < C (g)
C(g) < C (h)

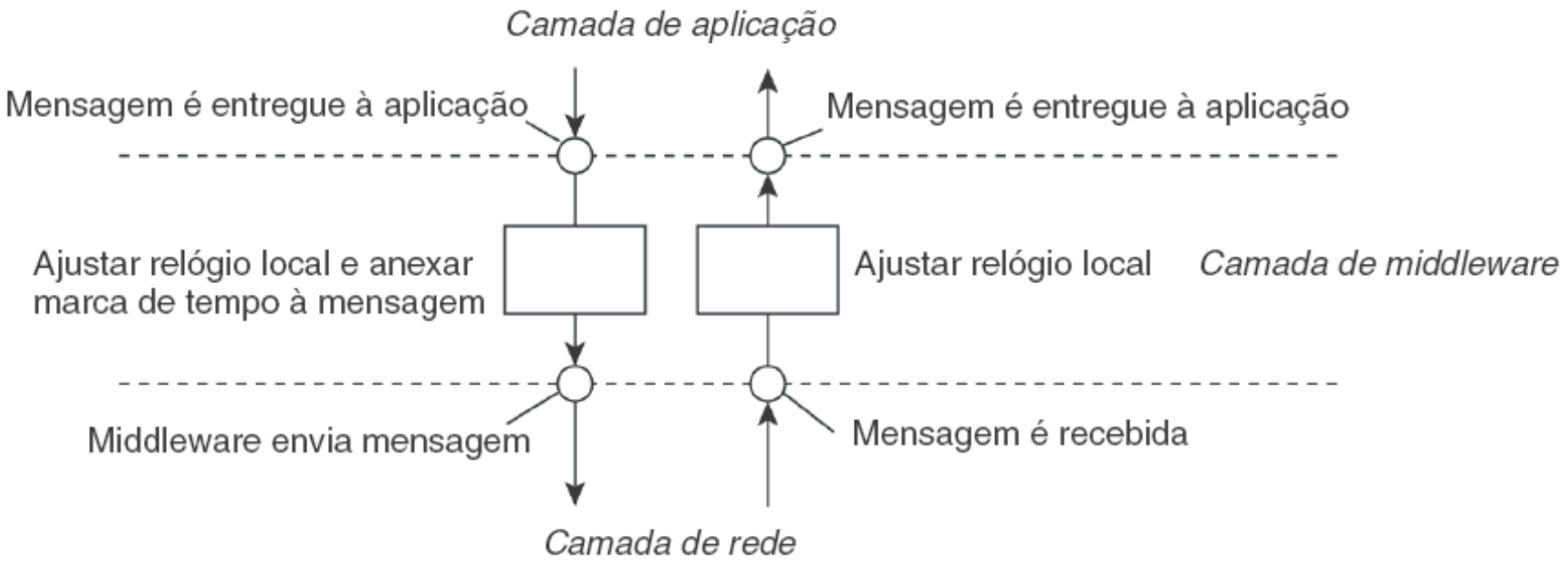
Relógios Lógicos – Algoritmo

- Objetivo é atribuir carimbos de tempo (*timestamps*) para cada evento

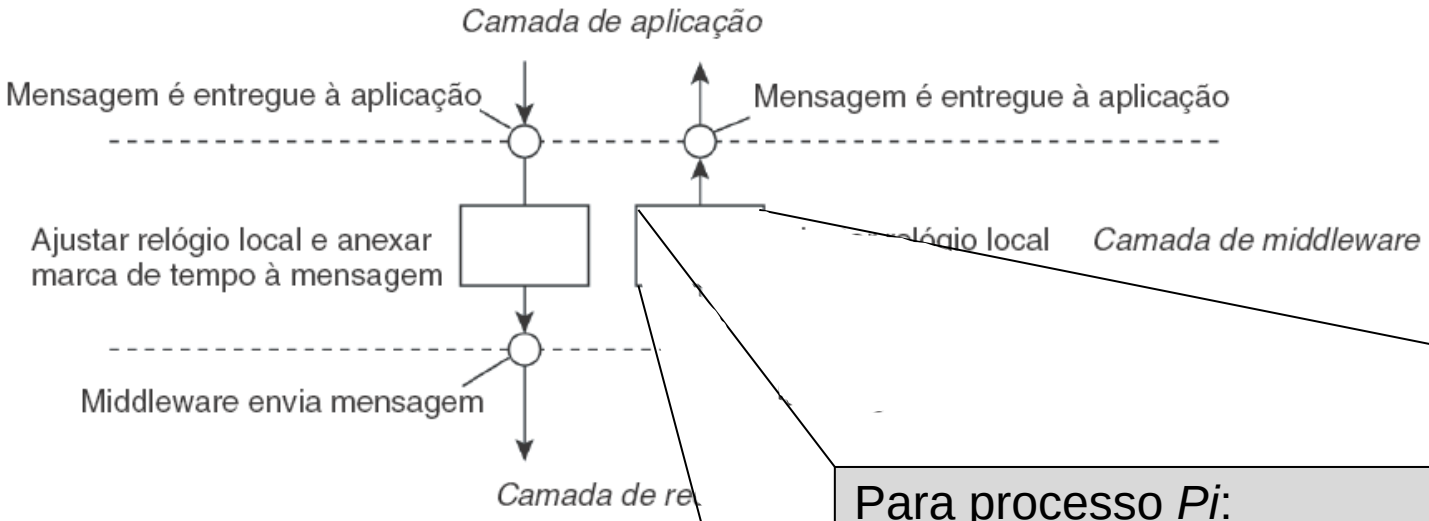
- **Regras:**
  - Cada processo `Pi` tem o seu relógio local, representado por um contador `Ci`:
    - O contador é incrementado em uma unidade a cada evento `send(m)` ou na execução de uma `instrução local`
  - Eventos `send(m)` carregam o *timestamp* do invocador na mensagem, `ts(m)`
  - Para os eventos `receive(m)`, a atualização do contador é dada por:

```
max(Ci, ts(m)) + 1
```

Relógios Lógicos – Aspectos de Implementação



Relógios Lógicos – Aspectos de Implementação



Para processo *Pi*:

1. Sempre antes de executar qualquer evento:

```
Ci = Ci +1
```

2. No envio de mensagens:

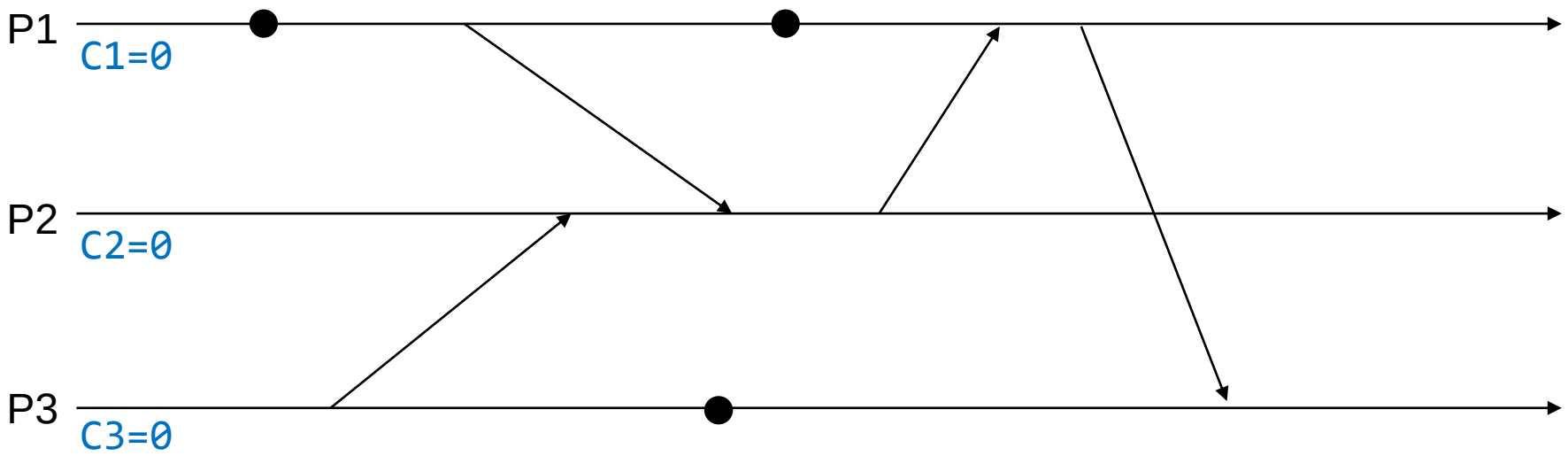
```
ts = Ci
send(m,ts)
```

3. No recebimento de mensagens:

```
[m,ts(m)] = receive(m,ts)
Ci = max(Ci, ts(m))
Ci = Ci + 1
```

Relógios Lógicos – Exemplo

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo

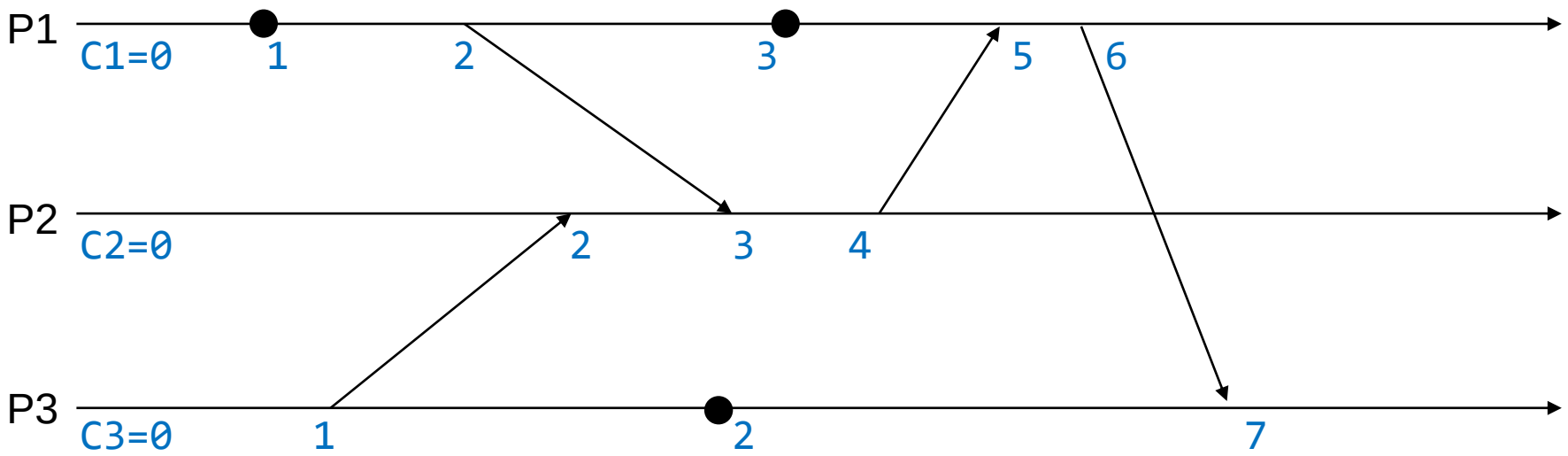


Instrução local

Envio de mensagem (send/receive)

Relógios Lógicos – Exemplo

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo

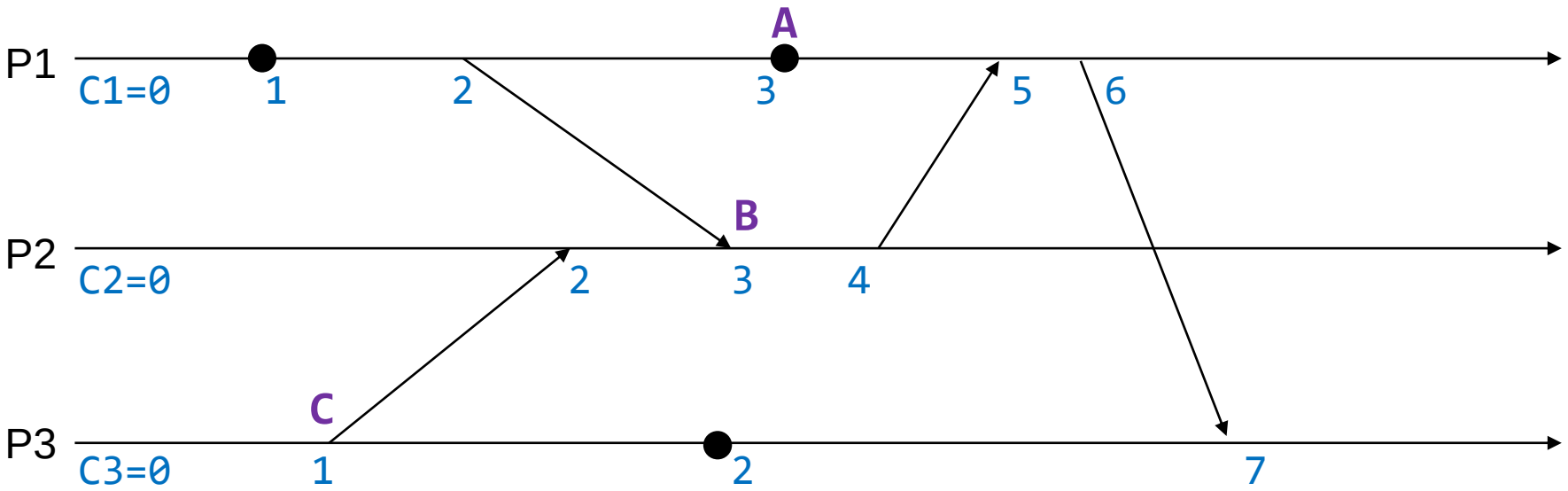


Instrução local

Envio de mensagem (send/receive)

Relógios Lógicos – Exemplo

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo



Instrução local

Envio de mensagem (send/receive)

Há causalidade entre:

A ⟶ B? *Relógios (3 = 3)*
C ⟶ A? *Relógios (1 < 3)*

Os pares (A,B) e (C,A) representam eventos concorrentes!

Relógios Lógicos – Causalidade e eventos concorrentes

- Os relógios de Lamport possibilitam perceber a causalidade entre os eventos localmente ou entre envio e recebimento de uma mesma mensagem

- Mas a mera comparação dos valores de C(A) e C(B), não expressa a causalidade entre os eventos

$$a \longrightarrow b \Rightarrow C(a) < C(b)$$

Porém:

$$C(a) < C(b) \Rightarrow a \longrightarrow b \vee a \,||\, b$$

Relógios Vetoriais

- Dado um grupo de N processos
- Cada processo Pi tem um vetor $VCi[1..N]$, onde $VCi[j]$ indica o conhecimento de Pi sobre o número de eventos ocorridos em Pj

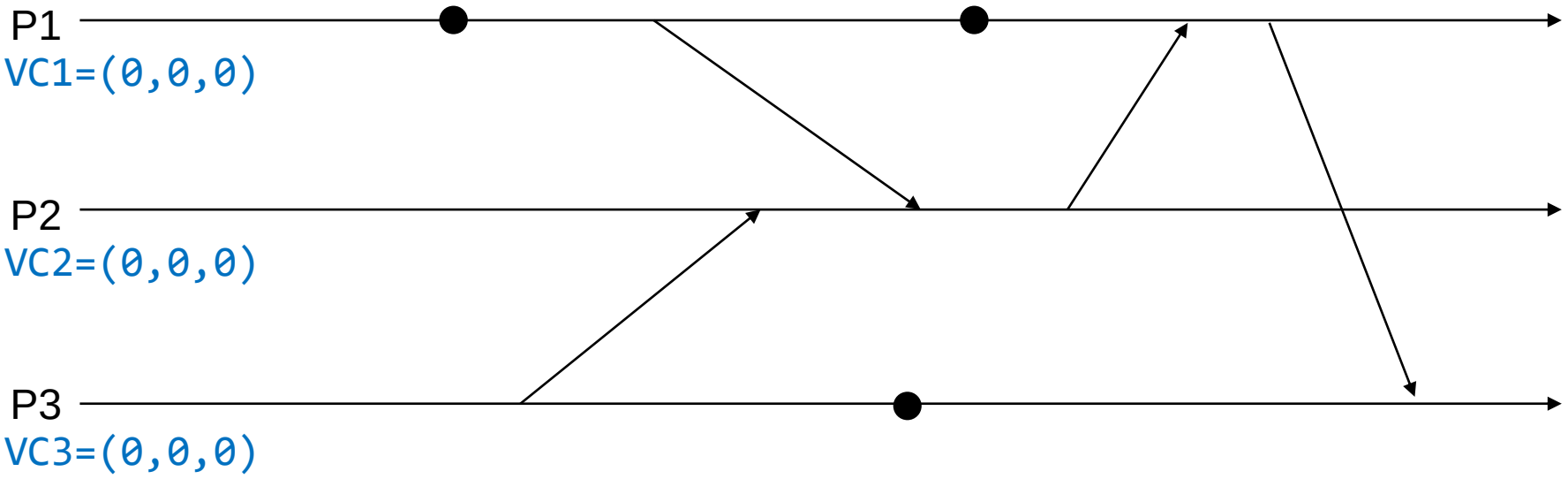
Para processo Pi:

1. Antes de enviar qualquer mensagem:
   $VCi[i] = VCi[i] + 1$
   o vetor VCi é enviado como *timestamp*
   $send(m,VCi)$

2. No recebimento de mensagens, Pi, atualiza cada posição de VCi:
   $receive(m,VCm)$
   $VCi[i] = VCi[i] + 1$
   $VCi[j] = max(VCi[j], VCm[j]),\ para\ todo\ j \neq i$

Relógios Vetoriais

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo

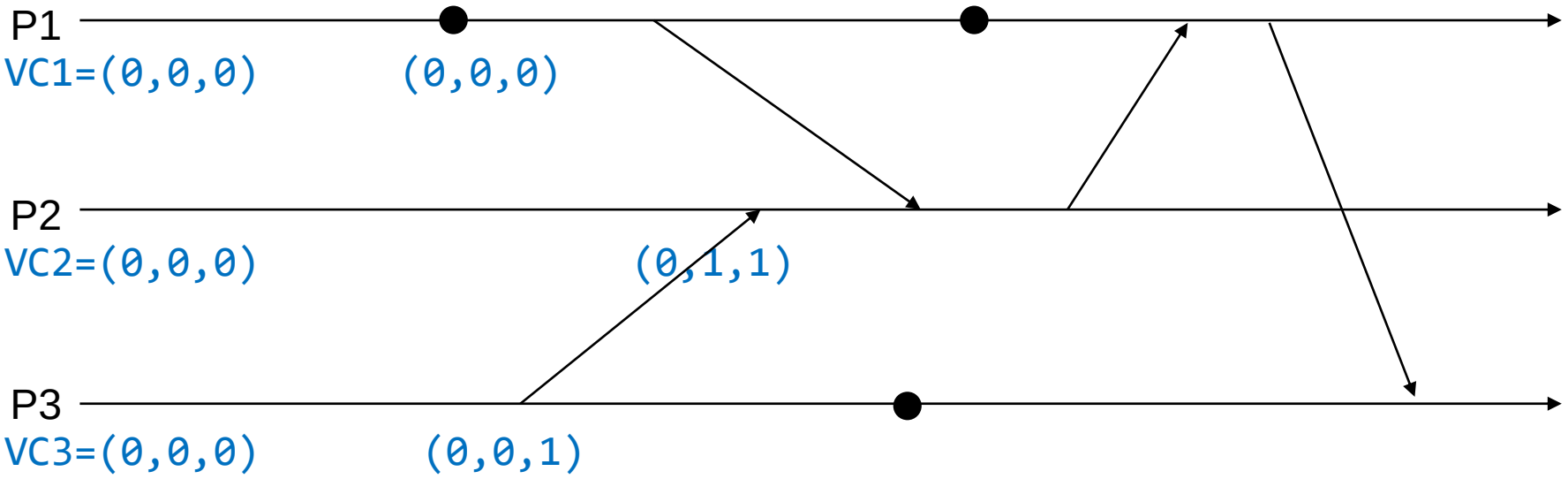


Instrução local

Envio de mensagem (send/receive)

Relógios Vetoriais

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo

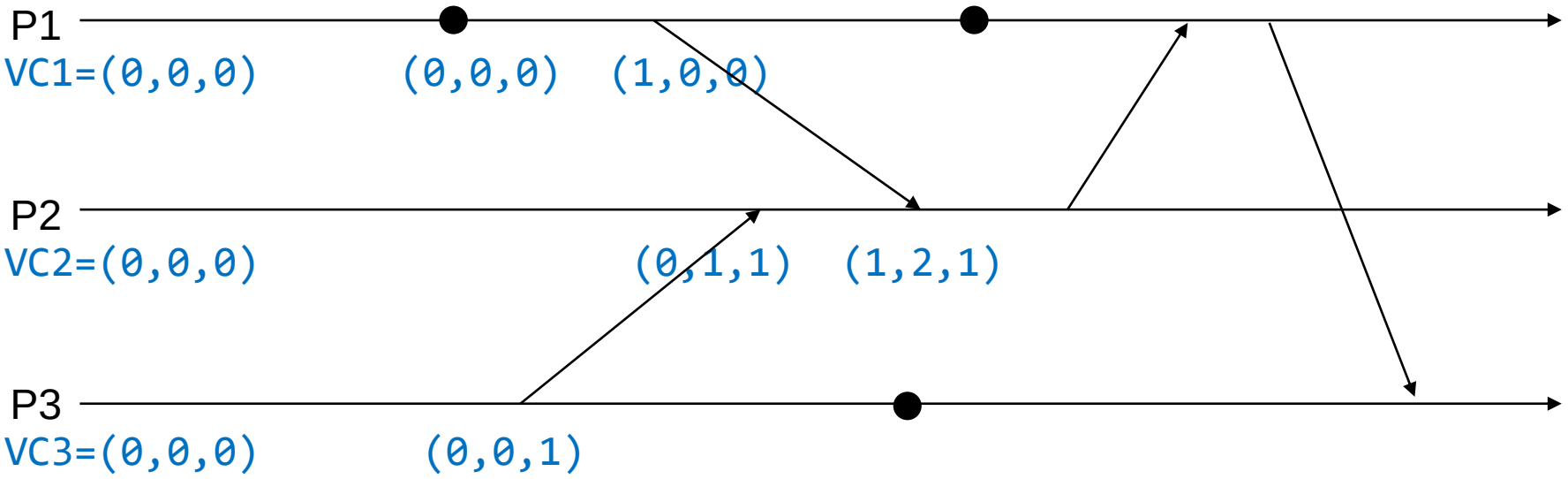


Instrução local

Envio de mensagem (send/receive)

Relógios Vetoriais

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo

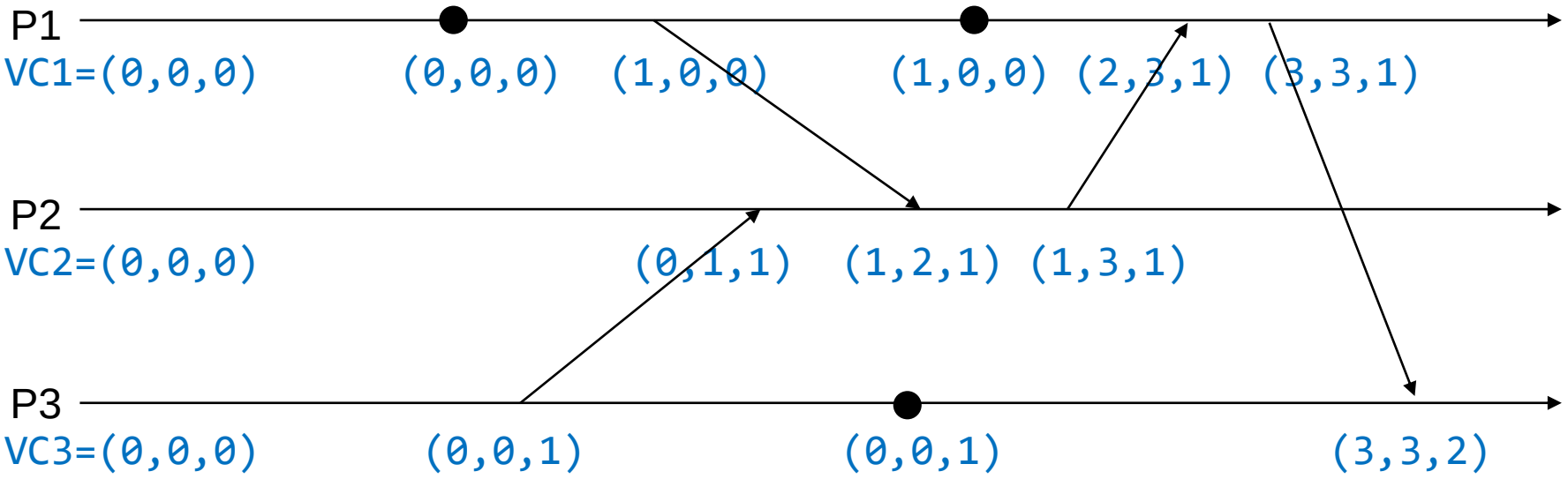


Instrução local

Envio de mensagem (send/receive)

Relógios Vetoriais

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo



Instrução local

Envio de mensagem (send/receive)

Relógios Vetoriais – Causalidade e eventos concorrentes

- $VC1 = VC2$
  - Se e somente se:
    - $VC1[i] = VC2[i]$, para todo $i = 1, \ldots, N$
- $VC1 \leq VC2$
  - Se e somente se:
    - $VC1[i] \leq VC2[i]$, para todo $i = 1, \ldots, N$

Dois eventos têm uma relação causal se e somente se:

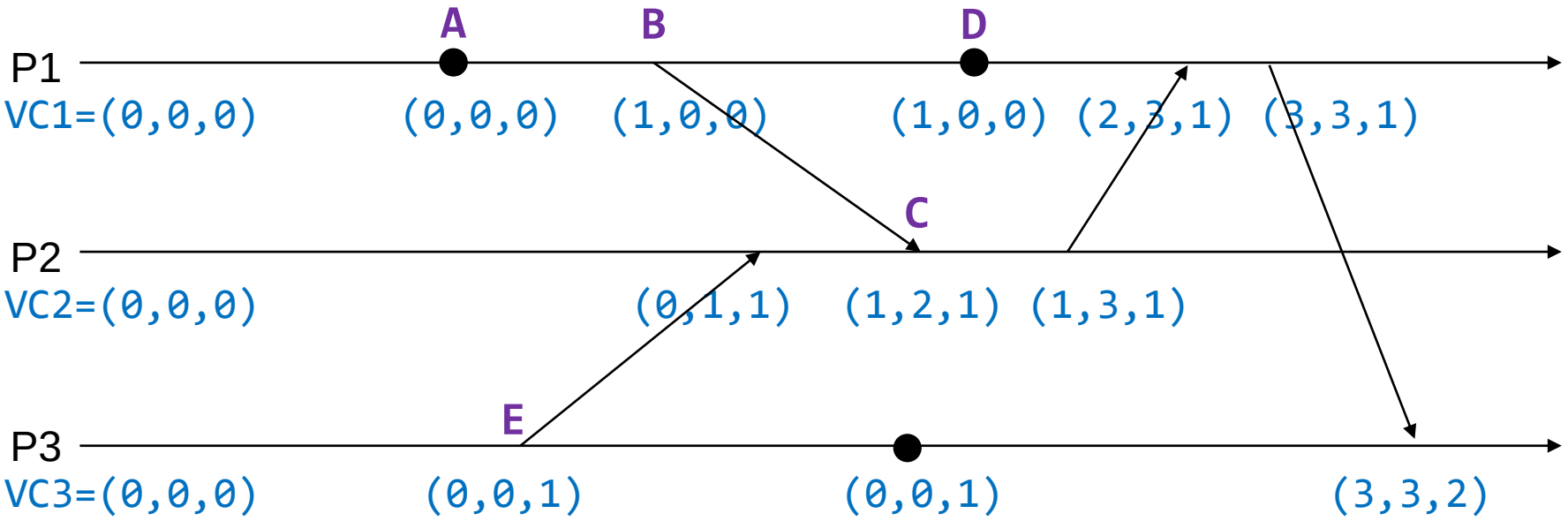
- $VC1 < VC2$
  - I.e., se e somente se:
    - $VC1[i] \leq VC2[i]$ e existe j, *tal que:*
    - $1 \leq j \leq N \wedge VC1[j] < VC2[j]$

Dois eventos são concorrentes se e somente se:

- $\neg (VC1 \leq VC2) \wedge \neg (VC2 \leq VC1)$
  - Isso significa que $VC1 \ || \ VC2$

Relógios Vetoriais

Com base no algoritmo de Lamport para relógios lógicos, rotule os eventos no exemplo



Causalidade:

A ⟶ B *Relógios (0,0,0) < (1,0,0)*

B ⟶ C *Relógios (1,0,0) < (1,2,1)*

Concorrência:

C || D *Relógios (1,2,1) || (1,0,0)*

E || D *Relógios (0,0,1) || (1,0,0)*

● *Instrução local*

⟶ *Envio de mensagem (send/receive)*

Relógios Vetoriais – Exemplos de Uso

Relógios lógicos representam um dos principais blocos de construção para sistemas distribuídos, em geral:

Exemplos de uso:
- Ordenação de recebimento de mensagens

- Implementação de exclusão mútua distribuída

- Eleição de líder em grupo de processos distribuídos

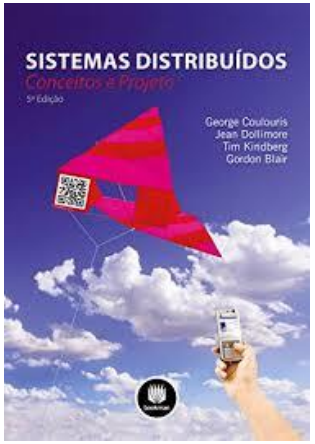
- Controle de transações distribuídas

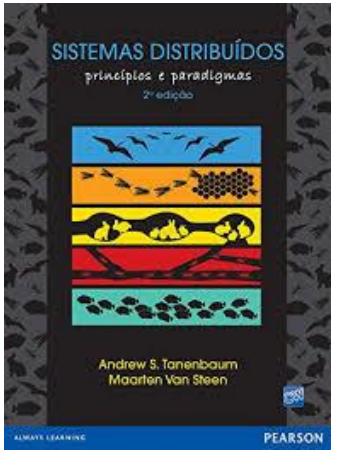
- *Key-value stores*
    - Exemplo Riak: https://riak.com/why-vector-clocks-are-easy/

Referências

Parte destes slides são baseadas em material de aula dos livros:

- *Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. Sistemas Distribuídos: Conceitos e Projetos. Bookman; 5ª edição. 2013. ISBN: 8582600534*
- *Tanenbaum, Andrew S.; Van Steen, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2007. Pearson Universidades; 2ª edição. ISBN: 8576051427*





- *Imagens e clip arts diversos:*
  https://free-icon-rainbow.com/
  https://www.gratispng.com/