

Sistemas Distribuídos

Aula 14

Aula passada

- Relógios
- Hora de referência
- Sincronizando relógios
- Algoritmo de Berkeley
- NTP

Aula de hoje

- Relacionando eventos
- Relógios lógicos
- Algoritmo de Lamport
- Propriedades

Sincronização de Relógios



- Problema fundamental e igualmente difícil
 - TAI utiliza satélites
- Relógios reais nunca serão *perfeitamente* sincronizados
- Sistemas distribuídos precisa de ordem
 - ordenação global dos eventos (em alguns casos)
 - escala de tempo de microsegundo (em alguns casos)

Como resolver este impasse?

Relacionando Eventos

- Relógios servem para ordenar a ocorrência de eventos
 - relógio real, sincronizado, é uma forma
- Relação de tempo definida localmente é fácil
 - evento que acontece depois, vem depois!
- Problema está na relação de tempo entre locais diferentes
 - evento no local A e outro no local B, qual aconteceu antes?

Exemplo Futebolístico

- Considere a cobrança de um pênalti e cinco lugares
 - linha do gol, marca do pênalti, linha da pequena área, linha da grande área, linha de fundo
- Considere os seguintes eventos
 - e1: artilheiro na linha da grande área
 - e2: goleiro na linha do gol
 - e3: artilheiro corre para bola na marca do pênalti
 - e4: goleiro abre os braços
 - e5: artilheiro chuta bola
 - e6: goleiro pula para o lado
 - e7: bola bate no goleiro
 - e8: artilheiro corre para linha da pequena área
 - e9: bola vai para linha da pequena área
 - e10: artilheiro chuta bola na linha da pequena área
 - e11: bola sai pela linha de fundo

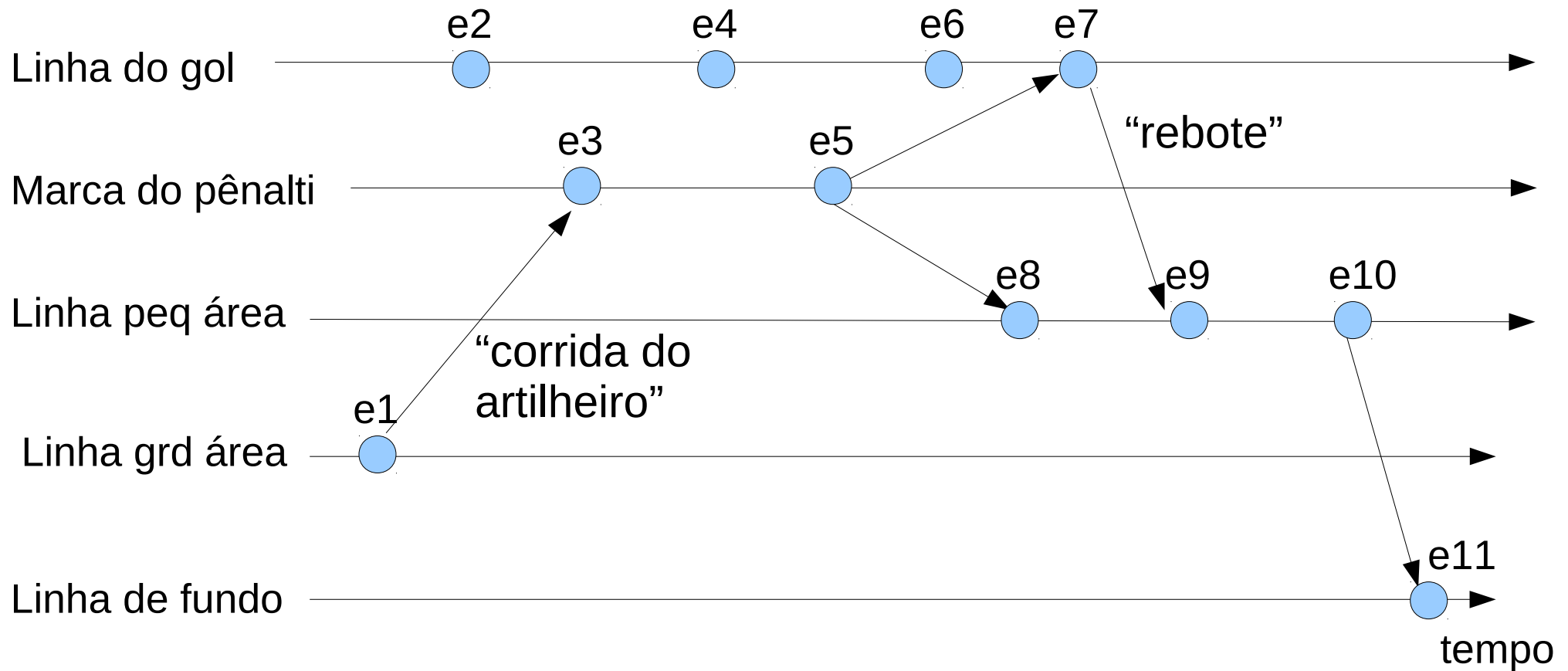
Podemos ordenar os eventos pela ordem de ocorrência?

Ordenação dos Eventos

- É possível colocar os eventos futebolísticos em ordem?
- Para cada local, sabemos a relação de ordem
 - ex. e2 ocorre antes de e4 que ocorre antes de e7
- Para locais diferentes, sabemos algumas relações de ordem
 - ex. e5 ocorre antes de e7 que ocorre antes de e10
- Mas para locais diferentes, não sabemos algumas relações de ordem
 - ex. e3 ocorre antes ou depois de e4

Ordenação Parcial dos Eventos!

Diagrama de Eventos



- Diagrama ilustrativo: não sabemos se e4 ocorreu antes de e5, como no diagrama
- Diagrama ajuda a entender ordem parcial
 - ex. “seta” indica que origem ocorreu antes do destino

Tempo Lógico

- Capturar apenas relação “ocorreu antes” entre um par de eventos
 - descarta aspecto infinitesimal do tempo
 - corresponde a causalidade
- Passagem de tempo em cada local é bem definida
- Definição de \rightarrow_i : Dizemos que $e \rightarrow_i e'$ se o evento e “ocorreu antes” de e' no local i
 - local vai ser processo, ou computador

Tempo Lógico Global

- Definição de \rightarrow : Definimos a relação $e \rightarrow e'$ usando as regras:
 - ordem local: $e \rightarrow_i e'$ para algum local i
 - mensagens: $\text{send}(m) \rightarrow \text{receive}(m)$ para algum m
 - Transitividade: se $e \rightarrow e'$ e $e' \rightarrow e''$ então $e \rightarrow e''$
- $\text{send}(\cdot)$ e $\text{receive}(\cdot)$ são eventos
- Dizemos que e “ocorreu antes” de e' se $e \rightarrow e'$

Concorrência

- Relação \rightarrow define apenas *ordem parcial* entre os eventos
 - alguns eventos não estão relacionados
- Definição de \parallel : Dizemos que $e \parallel e'$ se não existir $e \rightarrow e'$ e $e' \rightarrow e$, ou seja e “ocorre concorrentemente” com e'
 - não temos como ordenar e e e'

Voltando ao Futebol

- Relação entre os eventos do futebol
- $e2 \rightarrow e4$, $e8 \rightarrow e9$ (por ordem local)
- $e1 \rightarrow e3$, $e7 \rightarrow e9$ (por mensagem)
- $e2 \rightarrow e6$, $e5 \rightarrow e10$ (por transitividade)
- $e3 \parallel e4$, $e5 \parallel e6$ (por falta de relação \rightarrow entre os eventos)
 - “ocorrem concorrentemente”

Relógio Lógico de Lamport

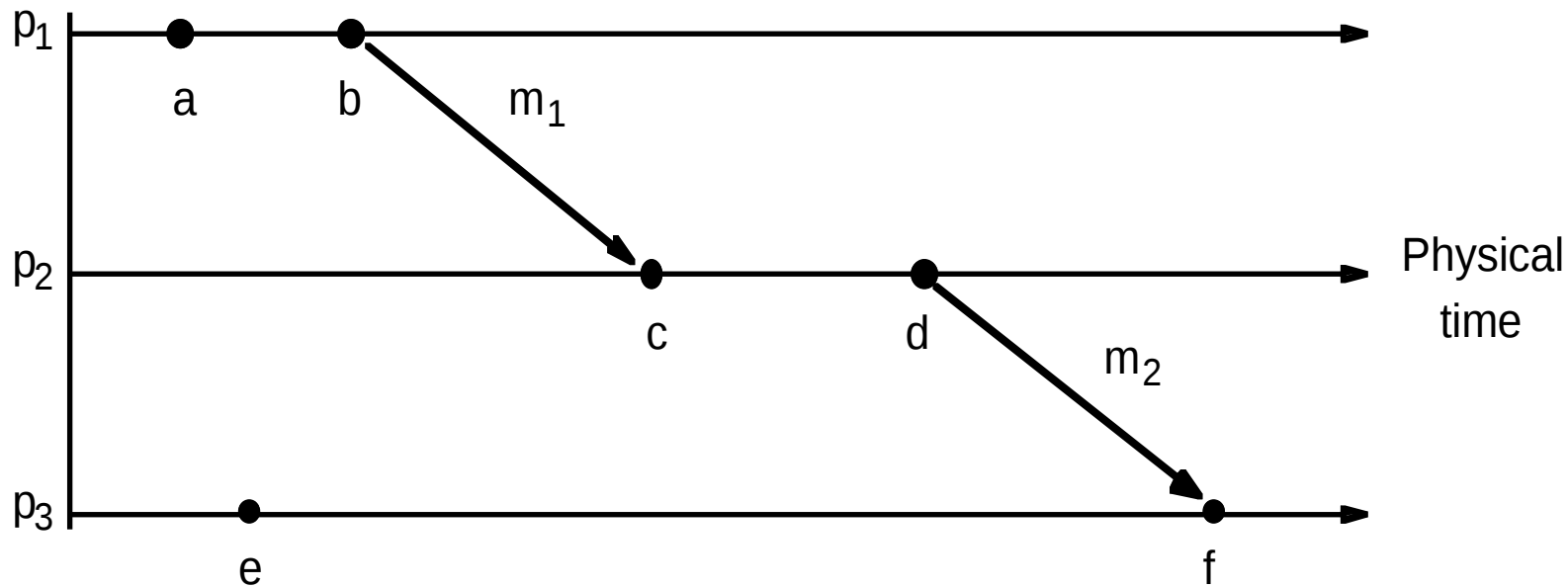
- **Ideia:** construir um relógio lógico baseado na ocorrência dos eventos
 - objetivo: inferir relação entre eventos usando relógio lógico
- Associar a cada evento do sistema um valor inteiro
 - sem relação com relógio real
- Cada processo (local) mantém um relógio lógico, L_i
 - processo associa valor de relógio a seus eventos
- Relógio lógico de cada process cresce monotonicamente

Algoritmo de Lamport

- Como ajustar o relógio lógico? Três regras:
- 1) em cada processo i , incrementar relógio lógico L_i antes de cada evento
- 2) transmissão de mensagem é evento, enviar valor do relógio L_i na mensagem
- 3) ao receber mensagem (m, t) , processo i atualiza seu relógio: $L_i = \max(L_i, t) + 1$
 - t é o valor do relógio lógico recebido na mensagem
- Valor do relógio global de um evento é valor do relógio local onde evento ocorreu
 - $L(e) = L_i(e)$ se e ocorreu em i

Exemplo

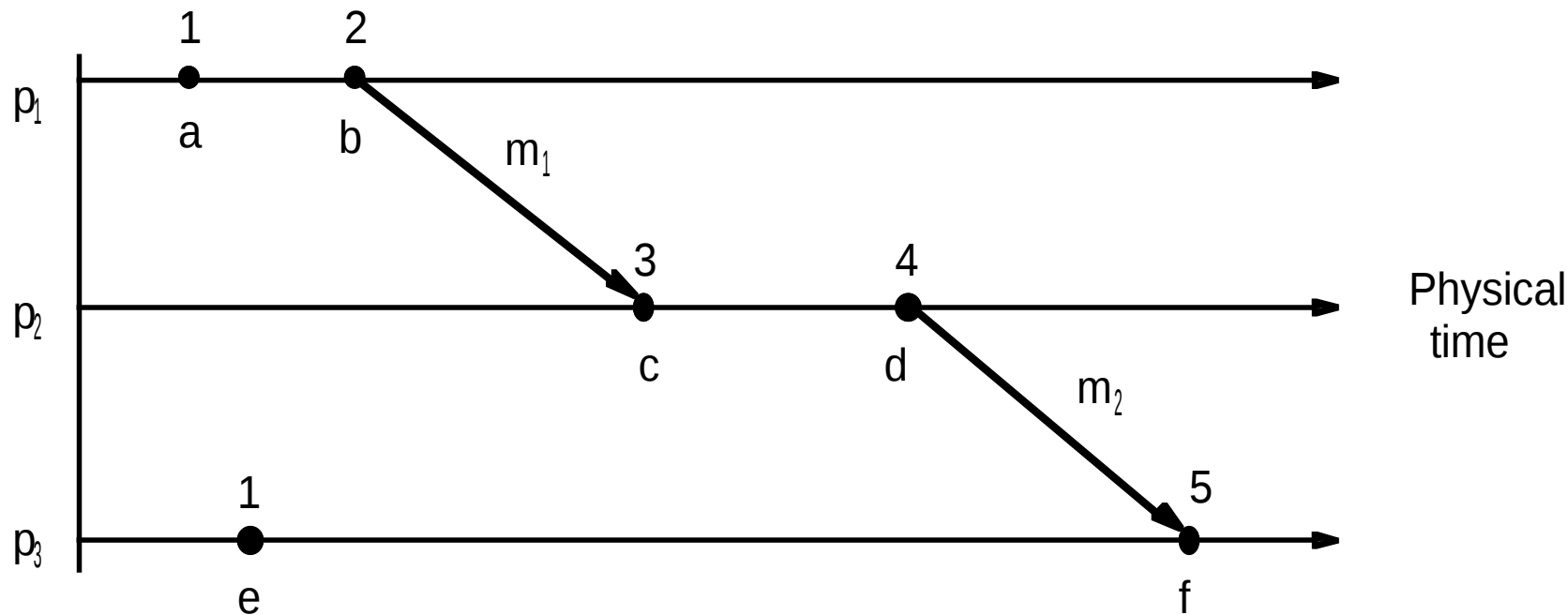
- Três processos, valores iniciais de $L_i = 0$ para todo i



- Quanto vale $L(a)$, $L(b)$, $L(c)$, ... ?

Exemplo

- Três processos, valores iniciais de $L_i = 0$ para todo i

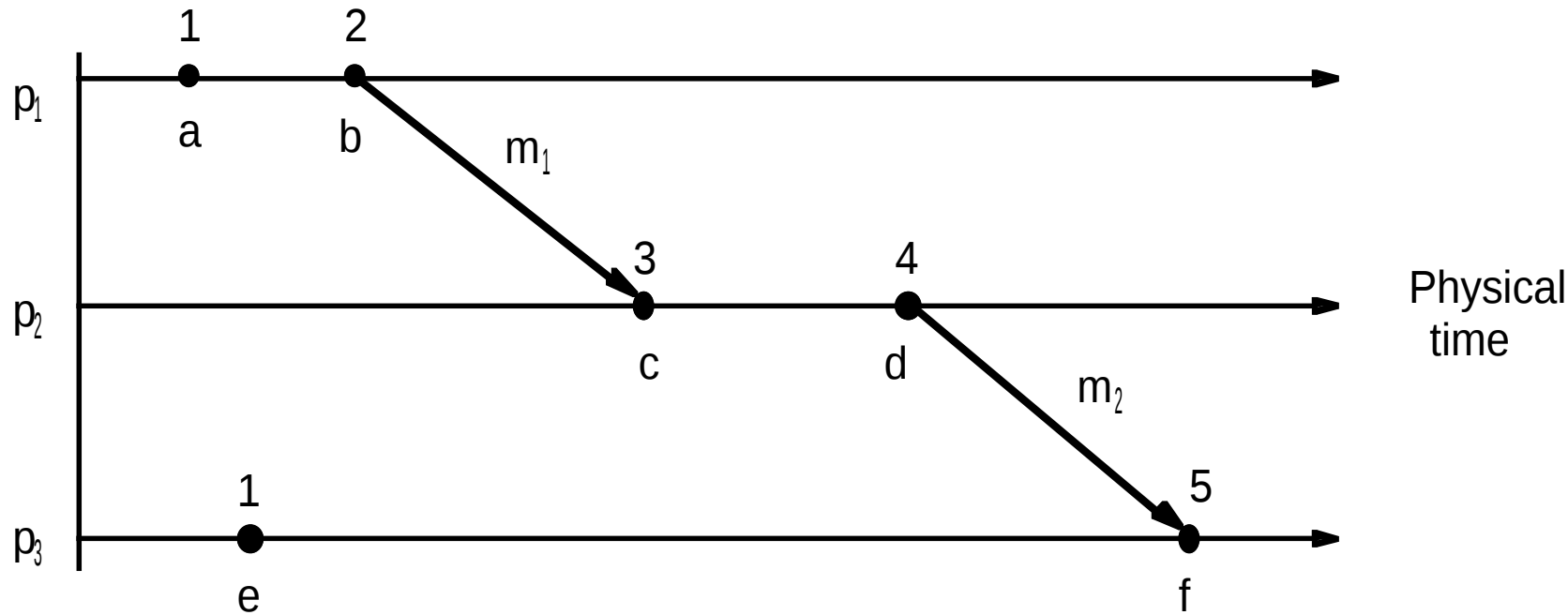


- Regra 1 para evento a, $L(a) = 1$
- Regra 2 para evento b, $L(b) = 2$ (um evento *send*)
 - m_1 contém valor 2
- Regra 3 para evento c, $L(c) = \max(0, 2) + 1 = 3$ (um evento *receive*)

Propriedades

- Relógio de Lamport ordena eventos de forma consistente com relação “ocorreu antes”
 - se $e \rightarrow e'$ então $L(e) < L(e')$
- Mas o reverso não é verdade
 - $L(e) < L(e')$ não implica que $e \rightarrow e'$
- Similar para relação de “ocorreu concorrentemente”
 - se $L(e) = L(e')$ então $e \parallel e'$ (para e, e' distintos)
 - mas $e \parallel e'$ não implica em $L(e) = L(e')$
 - relógio de Lamport ordena eventos concorrentes de forma arbitrária

Verificando Propriedades



- $a \rightarrow b$, de fato $L(a)=1 < L(b)=2$
- $b \rightarrow d$, de fato $L(b)=2 < L(d)=4$
- $L(e)=1 < L(b)=2$, mas não temos $e \rightarrow b$
- $L(a)=1 = L(e)=1$, de fato $a \parallel e$
- Mas $e \parallel c$ e não temos que $L(e) = L(c)$

Leslie Lamport – o próprio

- Contribuições fundamentais para sistemas distribuídos
- Introduziu conceito de relógios lógicos e relação “ocorreu antes”
 - algoritmo de Lamport, 1978
- Introduziu conceito de falhas bizantinas (que veremos)
- E ainda “inventou” o LaTeX (La é de Lamport!)
- Prêmio Turing de 2013
 - palestra “*An incomplete history of concurrency*”

