Sistemas Distribuídos
Aula 15

Aula passada

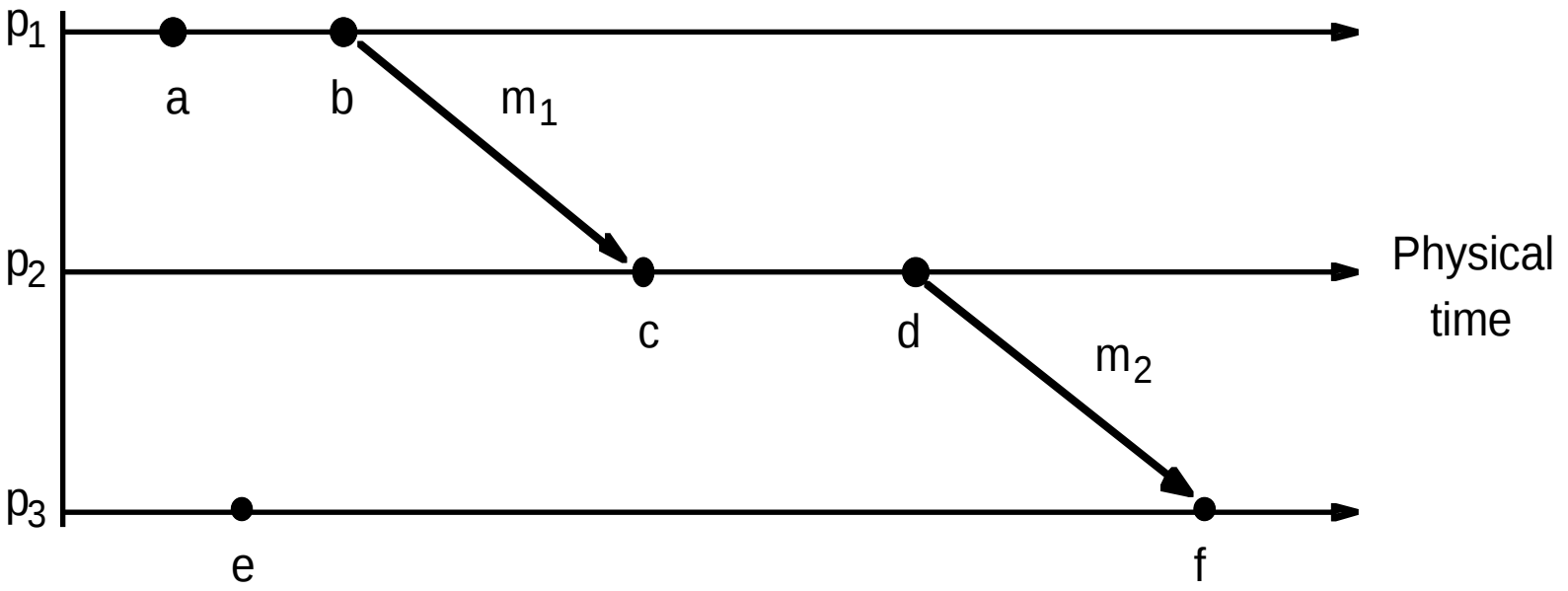
- Relacionando eventos
- Relógios lógicos
- Algoritmo de Lamport
- Propriedades

Aula de hoje

- Limitação de Lamport
- Relógio de vetores
- Propriedades
- Garantindo ordenação total

Relógio de Lamport

- Algoritmo de relógio lógico distribuído para inferir relação entre eventos
  - se $e \rightarrow e'$, então $L(e) < L(e')$
  - se $L(e) = L(e')$, então $e \parallel e'$



- Não faz parte da dinâmica da aplicação distribuída!

Limitação de Lamport

- Não podemos inferir ordem a partir dos valores dos relógios
  - L(e) < L(e') não implica e → e'
- Como garantir ordem parcial a partir dos valores de relógio?
  - L(e) < L(e') implica e → e'
- **Ideia**: adicionar informação ao valor do relógio lógico associado ao evento
  - usar vetor, ao invés de um número

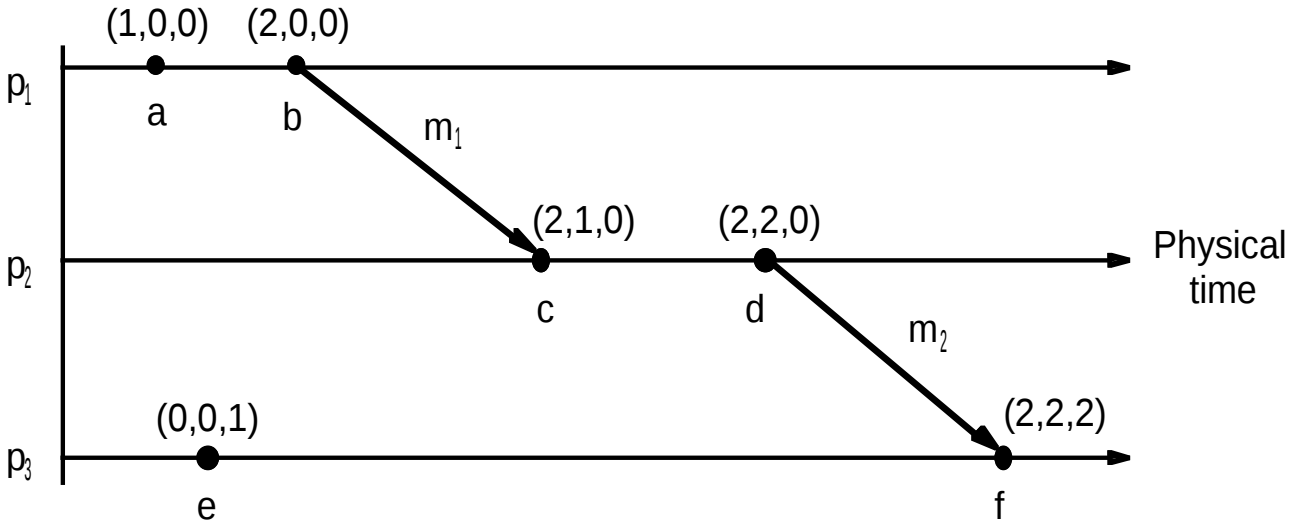
Relógio de Vetor

- Cada evento rotulado com um vetor de valores
- Cada componente do vetor associado a um processo
  - dimensão do vetor é número de processos
- $V(e) = [c_1, c_2, c_3, ..., c_n]$
  - c_i = número de eventos que ocorreram no processo i
  - que o processo *j* já sabe (não sabe de todos)
- Inicialmente, todos processos com vetor em 0

Algoritmo do Relógio de Vetor

- Para cada evento local em i, incrementa próprio c_i
  - receber mensagem é evento
- Para cada mensagem, transmite vetor de relógio atual
- Ao receber mensagem com vetor $[d_1, d_2, \ldots, d_n]$, processo *j* faz
  - ajusta relógio local para $c_k = \max(c_k, d_k)$
  - incrementa c_j

Exemplo de Relógio de Vetor

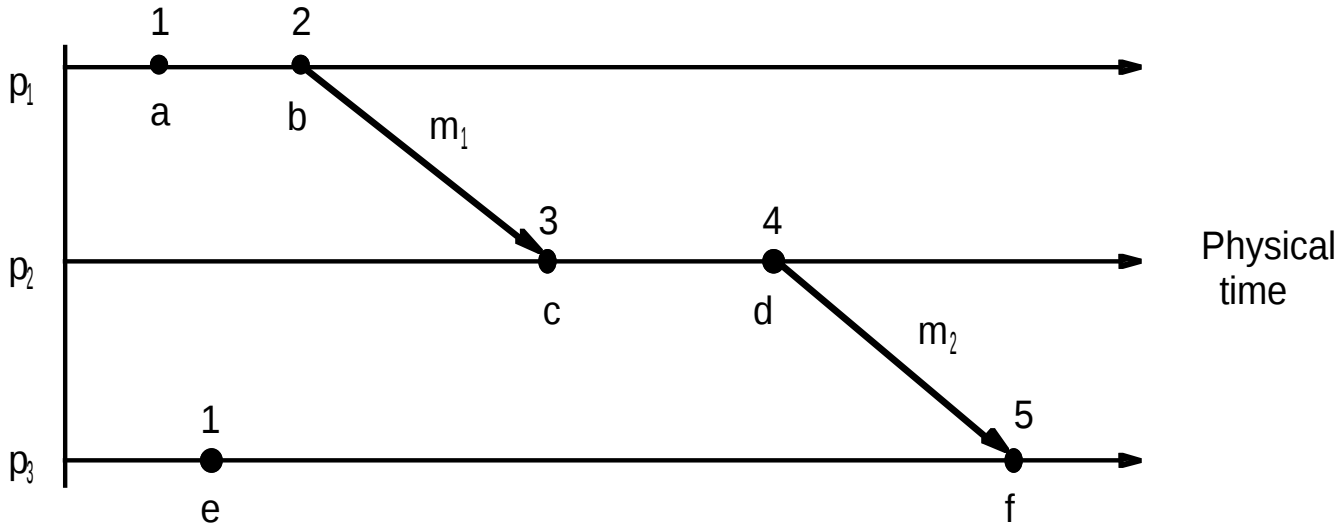


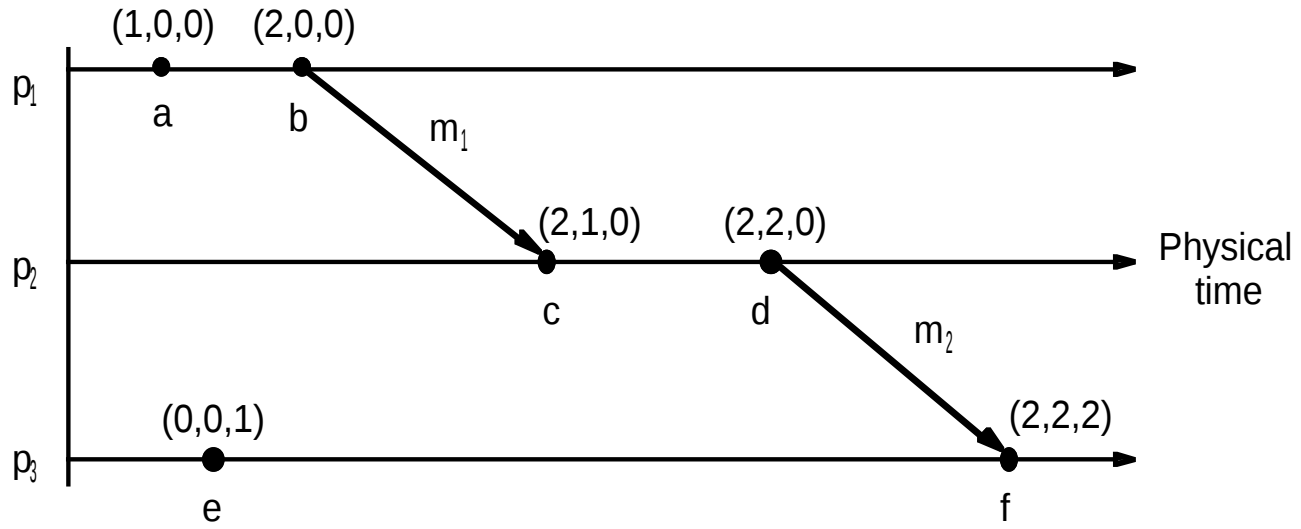
- V(a) = (1,0,0), V(b) = (2,0,0), V(e) = (1,0,0)
- m_1 contém (2,0,0), m_2 contém (2,2,0)

Comparando Valores

- Cada evento possui vetor
  - valor de relógio do evento
- Comparação componente a componente
  - V(*e*) < V(*e*') se (i) V(*e*)[k] <= V(*e*')[k] para todo k, (ii) existe z tal que V(*e*)[z] < V(*e*')[z]
  - ao menos uma componente menor
- Agora temos a propriedade
  - se V(e) < V(e') então e → e'
- Concorrência
  - dizemos que V(*e*) = V(*e*') se não for o caso de V(*e*) < V(*e*') ou V(*e*') < V(*e*)
  - Se V(*e*) = V(*e*') então *e* || *e*'

Exemplo de Relógio de Vetor





- Lamport: L(*e*) < L(*c*) mas não temos $e \rightarrow c$
- Relódio de Vetor V(*e*) = V(*c*) então $e \parallel c$

Outras Propriedades

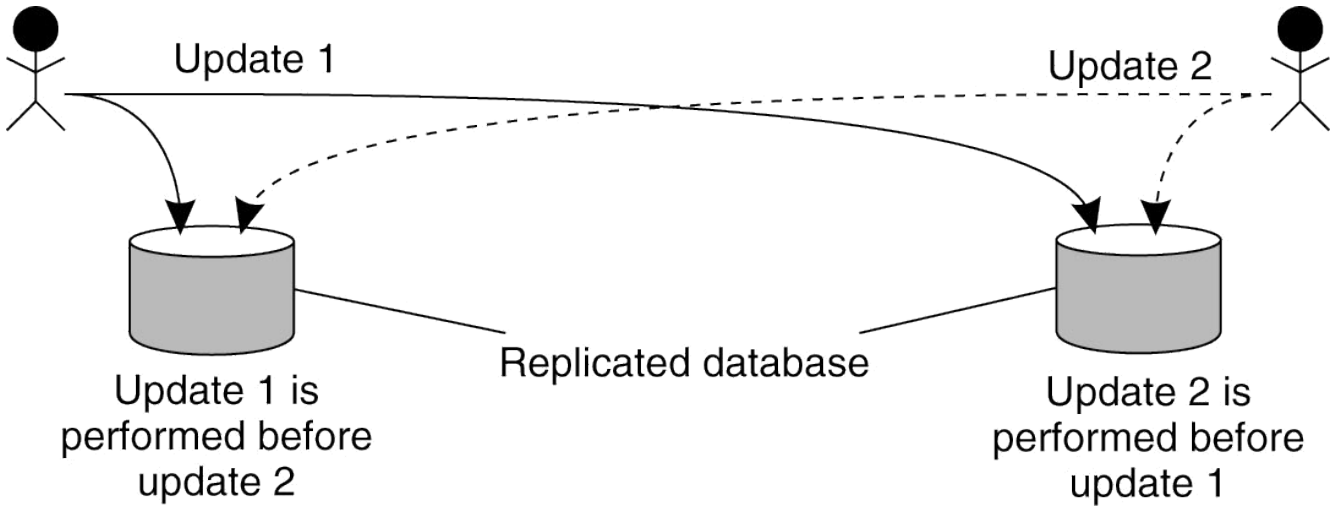
- Seja RT(x) o valor de tempo universal de x
  - se V(a) < V(b) então RT(a) < RT(b)
- Seja L(x) o valor do relógio de Lamport de x
  - se V(a) < V(b) então L(a) < L(b)
- Oferece apenas uma ordem parcial, assim como a relação → “ocorreu antes”
  - não oferece ordem total dos eventos (veremos como ainda hoje)

Sincronização de Relógios

- Relógios em sistemas diferentes serão sempre diferentes
  - até mesmo em relógios atômicos
- Falta de acordo de tempo em sistemas diferentes pode levar a comportamento errático
- Duas abordagens para solução
  - sincronização do relógio
  - consistência na ordem dos eventos
- Na prática, ambos são usados

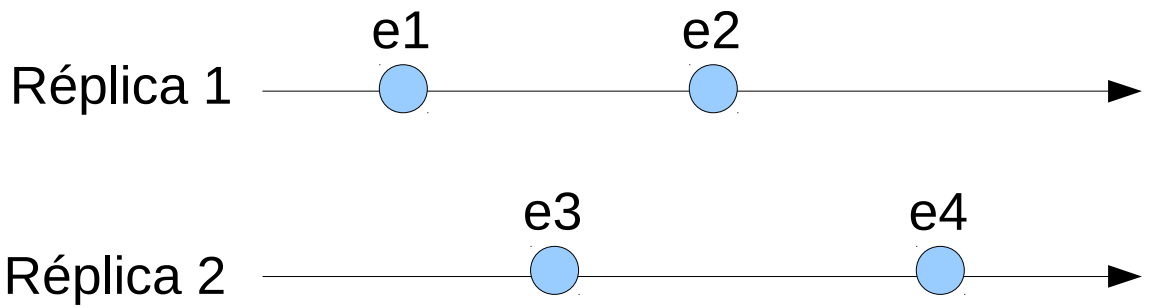
BD Distribuído e Replicado

- Conta com saldo inicial de $1000
- Dois usuários fazem transações



- Usuário 1: aumenta saldo em $100
- Usuário 2: aumenta saldo em 1%
- Saldo final da conta?
  - se u1 → u2 : $1111
  - se u2 → u1 : $1110
- Como obter consistência nos dois bancos?
  - valor final não impornta, consistência sim!

Eventos e Tempos

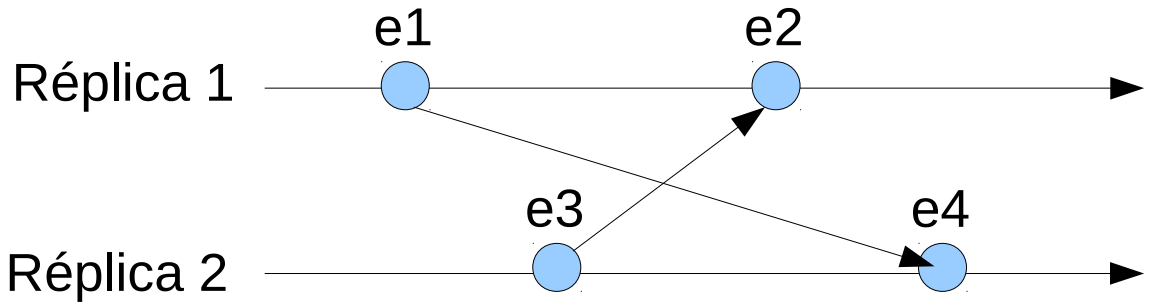


- e1: R1 recebe update u1
- e2: R1 recebe update u2
- e3: R2 recebe update u2
- e4: R2 recebe update u1

- L(e1) < L(e2), L(e3) < L(e4)
  - mas não temos relação entre eventos das réplicas distintas
- Como relacionar eventos (transações) das diferentes réplicas?

Réplicas enviam mensagens!

Eventos e Tempos



- e1: R1 recebe update u1, envia para R2
- e2: R1 recebe u2 de R2
- e3: R2 recebe update u2, envia para R1
- e4: R2 recebe u1 de R1

- Qual relação entre L(e2) e L(e4)?
- Relógio de Lamport não resolve
  - nem relógio de vetor
- Precisamos de um algoritmo distribuído para garantir ordem total
  - que vai usar relógio de Lamport!

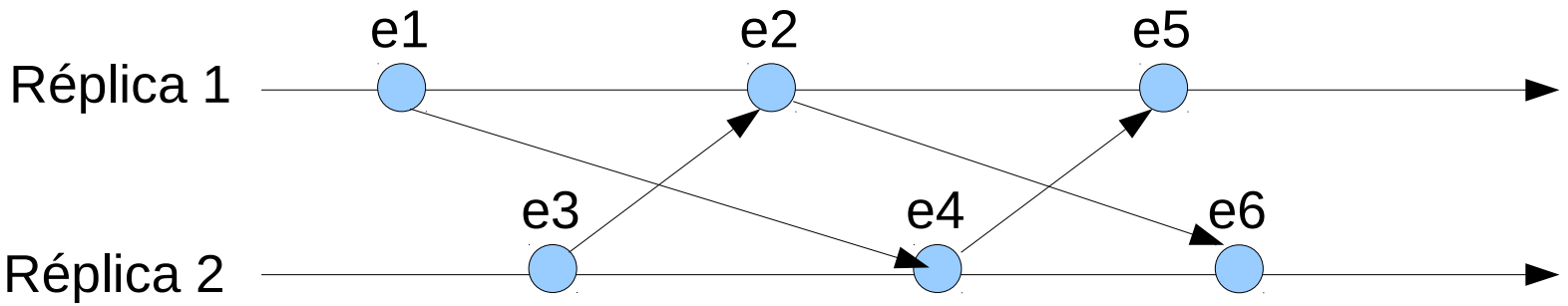
Garantindo Ordem Total

- **Ideia**: processar transações em uma ordem total
  - todos processos executam transações na mesma ordem
  - transação será confirmada antes de ser executada
- **Algoritmo:**
  - relógio lógico de Lamport mantido para transações e eventos
  - cada processo mantém uma fila de transações (incluindo as suas), ordenada pelo valor do relógio lógico da transac
  - cada transação em P_i é enviada a todos processos
  - cada transação recebida por P_j é confirmada (via mensagem) para todos os processos
  - transação da cabeça da fila executada quando recebida a confirmação de todos processos ou relógio do processo confirmando ao menos igual ao da transação

Garantindo Ordem Total

- Assumir rede FIFO e que mensagens não se perdem
  - m1 transmitida antes de m2 (entre mesmo par origem/destino), chega antes de m2
- Filas vão ser idênticas em todos os processos
  - ordenadas por tempo lógico das transações
- **Problema**: tempo lógico em dois processos pode ser igual
- **Solução**: adicionar número do processo como parte do relógio lógico: “L(e).identificador”
  - apenas para quebrar empate na fila
  - não afeta andamento do relógio lógico

Exemplo de Ordem Total



- e1, e3: transações a serem executadas
- e4, e4: recebimento da transação, envio de confirmação
- e5, e6: eventos de chegada de confirmação
- No início, fila de transações em R1: [e1], fila em R2: [e3]
- Depois de e2, fila em R1: [e1 (1.1), e3 (1.2)]
- Depois de e4, fila em R2: [e1 (1.1), e3 (1.2)]
- L(e4) = 2: R2 pode executar e1 neste momento (cabeça da fila), mas não e3 (precisa aguardar chegada da confirmação)
- L(e2) = 2: R1 não pode executar e1, pois precisa aguardar confirmação
- L(e5) = 3: R1 pode executar e1, pois chegou confirmação, e também pode executar e3 pois L(e5) > L(e3)
- L(e6) = 3: R2 pode executar e3 neste momento, pois chegou confirmação e L(e6) > L(e3)

Totally Ordered Multicast

- Algoritmo garante ordem total da execução das transações, de forma distribuída
- Requer a transmissão das transações locais para todos processos do sistema
- Requer a confirmação do recebimento de cada transação para todos os processos do sistema
- Atrasa execução dos eventos
  - execução apenas após chegada das confirmações
- Mas ainda assim é utilizado em alguns contextos
  - e como primitiva em outros