Sistemas Distribuídos

Aula 6 – Sincronização

DCC/IM/UFRRJ
Marcel William Rocha da Silva

Conteúdo Programático

- Introdução e visão geral
- Princípios de sistemas distribuídos
  - Arquiteturas
  - Processos
  - Comunicação
  - Nomeação
  - Sincronização
  - Consistência e replicação
  - Tolerância à falha
  - Segurança
- Seminários

Objetivos da aula

- **Aula anterior**
  - Nomeação
- **Aula de hoje**
  - Sincronização de relógios
    - Relógios físicos
      - Algoritmos
    - Relógios lógicos
      - Relógios de Lamport
      - Relógios vetoriais

Sincronização

- Promover a coordenação entre processos depende em alguns casos de **sincronismo**
  - **Exemplo 1**: Vários processos tentando acessar um mesmo recurso compartilhado devem se sincronizar para evitar o uso simultâneo
  - **Exemplo 2**: Duas cópias de um mesmo banco de dados precisam saber em qual ordem aconteceram as ações sobre os dados replicados para evitar inconsistências

- Veremos diferentes desdobramentos do sincronismo entre processos...

Sincronização

- Sincronização em relação ao tempo
  - **Relógios Físicos e Lógicos**
- Sincronização em relação ao compartilhamento de recursos
  - **Exclusão Mútua**
- Sincronização em relação ao coordenador de um grupo
  - **Algoritmos de Eleição de Líder**

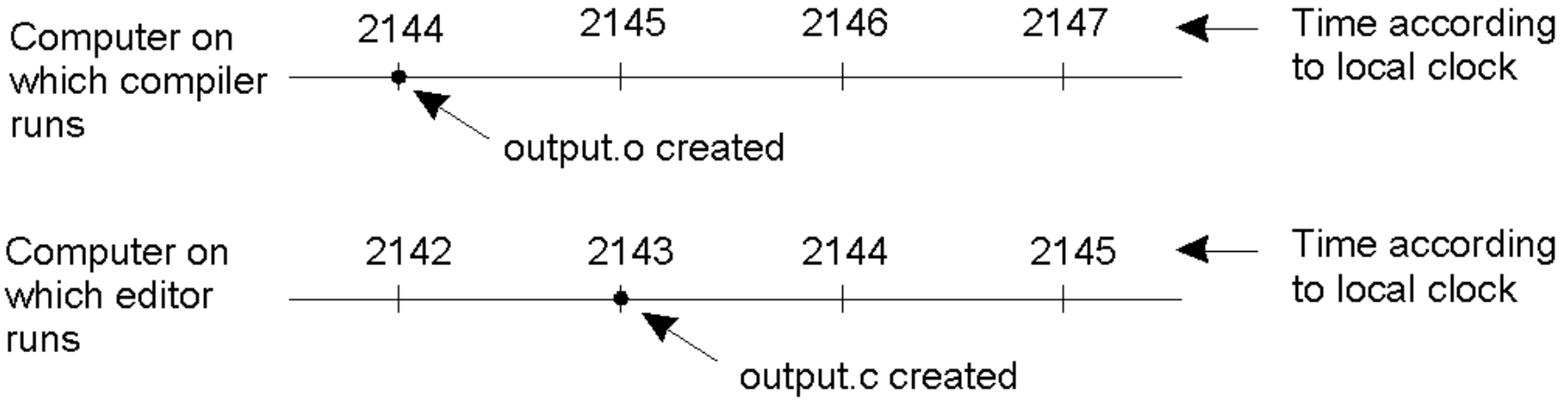
Sincronização de relógios

- Nos **sistemas centralizados** o tempo não é ambíguo
  - Se A “pergunta” a hora e logo depois B “pergunta” a hora, tempo A <= tempo B

- Em um **sistema distribuído** (ex. com a aplicação **make**)
  - Se arquivo fonte (*output.c*) possui horário maior que o arquivo objeto (*output.o*) → make recompila o arquivo fonte
  - Se o arquivo objeto é mais recente que o fonte → nenhuma modificação é feita
  - Relógios sincronizados são importantes para o funcionamento do make

Sincronização de relógios

- Exemplo do make quando cada máquina possui um relógio diferente
  - Relógio da máquina do editor atrasado em relação ao da máquina do compilador



Relógios físicos

- Quase todo sistema possui um circuito **temporizador** para contar a passagem do tempo
  - Cristais de quartzo, quando sob tensão, geram uma corrente elétrica com oscilação senoidal de frequência bem definida
  - Após n oscilações é gerada uma interrupção → a cada 60 segundos é atualizado uma memória que armazena a hora atual (nº de interrupções a partir de uma hora de referência)

Relógios físicos

- Com um único computador não há problema caso o relógio esteja defasado
  - Processos da máquina usam o mesmo relógio → consistência interna estará presente
- Em sistemas distribuídos com relógios diferentes → diferença nos relógios pode acontecer
  - Cada relógio possui sua própria frequência, que nunca é igual! → relógios gradativamente perdem sincronia
  - **Defasagem de relógio** (*clock skew*) → diferença na leitura de dois relógios

Relógios físicos

- Um pouco de história...
  - Nos primórdios, o tempo era medido de acordo com o "trânsito solar" → **dia solar**
    - Intervalo de tempo entre duas vezes consecutivas que o sol aparece no ponto mais alto do céu
    - **Segundo solar médio** → tempo médio transcorrido em 1/86400 de um dia solar
  - Problemas
    - A Terra está desacelerando → dias ficam gradativamente mais longos
    - Turbulências no núcleo da Terra afetam momentaneamente sua rotação
  - Consequência → Segundo solar não é preciso!

Relógios físicos

- Em 1948 surge o **relógio atômico**
  - Um segundo atômico é igual ao tempo que o átomo de césio 133 leva para fazer 9.192.631.770 oscilações
  - Uma média dos horários medidos por relógios atômicos ao redor do mundo dão origem ao **tempo (ou hora) atômico(a) internacional** (**TAI**)

Relógios físicos

- Mas como os dias estão ficando mais longos → TAI tende a ficar adiantado
  - Por isso ela é corrigida de tempos em tempos, dando origem a **hora coordenadada universal** (**UTC**)
  - Periodicamente a UTC é corrigida pela hora astronômica (dia solar médio)
  - UTC é informado de diferentes maneiras
    - Estações de rádio (WWV) e satélites (GPS)
  - Computadores capazes de receber estes sinais sincronizam seus relógios com o UTC

Relógios físicos - GPS

- *Global Positioning System* (sistema de posicionamento global)

- GPS usa 29 satélites que orbitam a Terra a uma altitude de 20,000 km
  - Cada satélite tem até quatro relógios atômicos que são calibrados periodicamente
  - Um satélite transmite sua posição em broadcast e anexa marcas de tempo a cada msg, informando a hora local
  - Essa transmissão broadcast permite que todo receptor na Terra calcule com precisão sua própria posição e também calcule a hora atual

Relógios físicos - GPS

- Calcular a posição/tempo usando GPS depende dos seguintes fatores:
  - Leva tempo para os sinais se propagarem do satélite para o receptor
  - O relógio do receptor não está sincronizado com o satélite
- Entretanto, outros fatores tornam o cálculo impreciso:
  - A Terra não é uma esfera perfeita
  - Velocidade de propagação do sinal não é constante
  - Relógios do satélite podem não estar perfeitamente calibrados
- Erros ficam em torno de 1 à 5 metros para posição e menos de 20 a 35 ns para o tempo

Relógios físicos

- Se uma máquina do SD tiver um relógio preciso (receptor WWV ou GPS), o objetivo é manter o relógio das outras máquinas sincronizadas com o dela

- Se nenhuma máquina tiver relógio preciso, cada uma monitora seu próprio horário, e o objetivo é manter todas as máquinas com o tempo mais próximo possível

Relógios físicos

- Vários algoritmos foram propostos para manter relógios de um SD sincronizados
- Todos assumem o mesmo modelo de sistema:
  - Cada máquina deve ter um temporizador que provoca uma interrupção H vezes por segundo
  - Quando o temporizador esgota o tempo fixado, o manipulador de interrupção soma 1 à um relógio de software que monitora o número de ciclos do relógio

Relógios físicos

- Seja **C** o valor do relógio 'ideal' (hora UTC)
- Seja $\mathbf{C_p}$ o valor do relógio no computador **p**
- Se **t** é o tempo no relógio em sincronia com a hora UTC, então temos o tempo em **p** dado por $\mathbf{C_p(t)}$
- Idealmente para todo **p** e para todo **t**: $C_p(t) = t$
  - Logo: $C_p'(t) = dC/dt = 1$

Relógios físicos

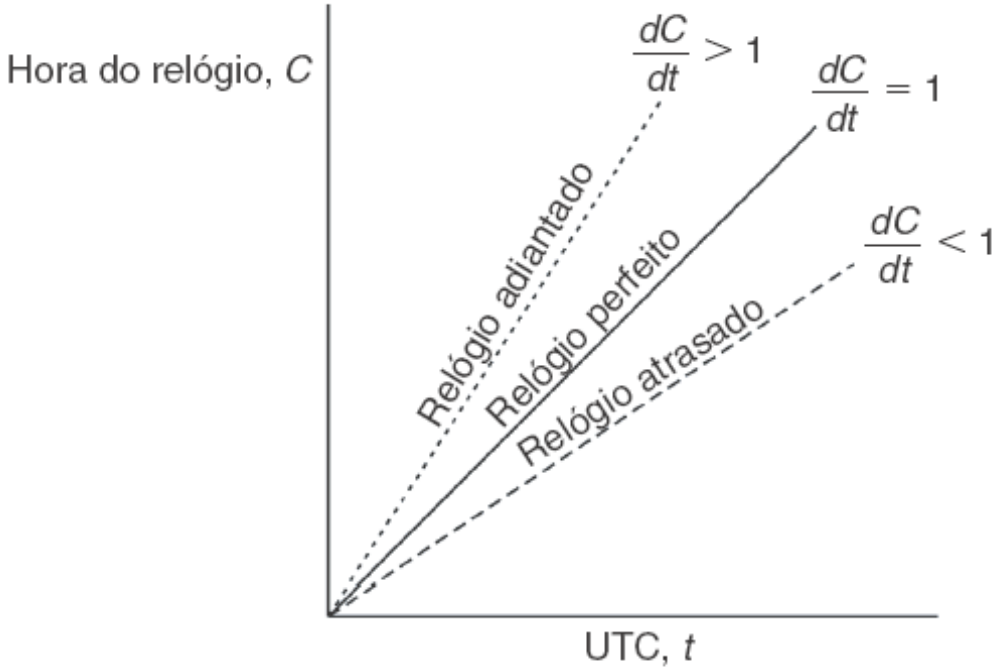


Figura 6.5 Relação entre a hora do relógio e a hora UTC quando as taxas de ciclos de relógios são diferentes.

Como fazer a sincronização periódica entre relógios?

Algoritmos de sincronização

- Se existe um 'servidor de tempo' (receptor WWC ou relógio de precisão)
  - **Algoritmo proposto por Cristian (1989)**
  - **Protocolo NTP**

- Se não existe uma fonte que disponibilize a hora coordenada universal (UCT)
  - **Algoritmo de Berkeley**

Algoritmo 1: Cristian'89

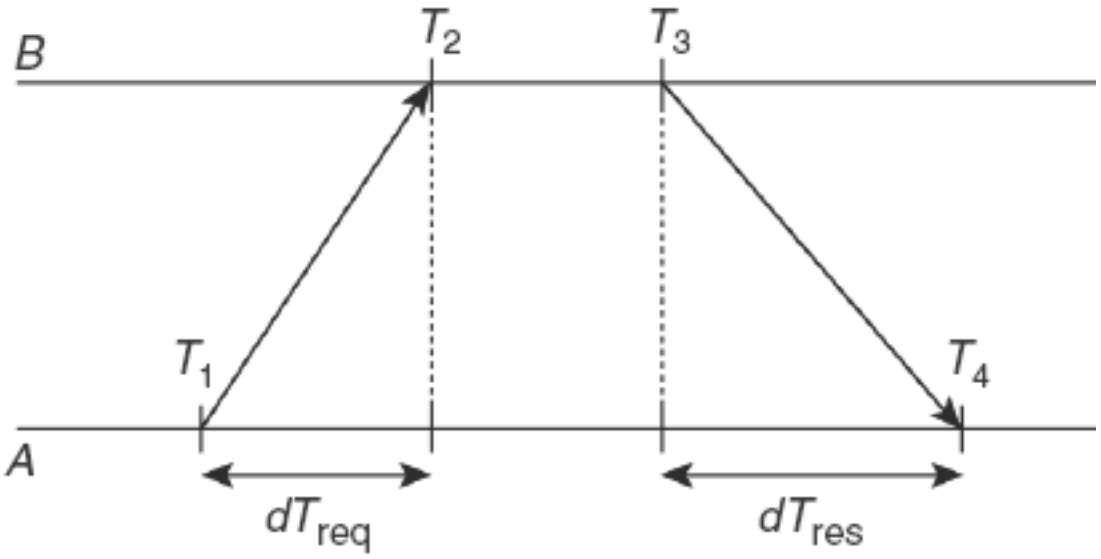
- Consulta a servidores de tempo equipados com um receptor WWV/GPS ou um relógio de alta precisão
- Utilizável em sistemas assíncronos onde os tempos de RTT (round-trip time) são menores que a precisão desejada (redes locais)

- Problema: Atrasos de mensagens farão com que a hora fornecida esteja desatualizada
  - Solução: Estimar os atrasos entre as máquinas!

Algoritmo 1: Cristian'89

1. Computador A consulta a hora no computador B (requisição)
2. Computador B inclui na resposta o valor do seu relógio → T3 e T2
3. Computador A registra a hora da chegada da resposta → T4
4. Supondo atraso aproximadamente igual em ambas as direções:

$$\theta = ((T2 - T1) + (T3 - T4))/2$$



Algoritmo 1: Cristian'89

- Se o relógio de A estiver adiantado ($\theta < 0$), significa que A deve atrasar o seu relógio

- Alteração deve ser feita gradativamente
  - Alteração abrupta poderia causar inconsistências
  - Se cada interrupção somaria 10 ms a hora, para atrasar, a rotina de interrupção soma apenas 9 ms, até que a correção tenha sido feita
  - O mesmo mecanismo pode ser usado caso o relógio de A esteja atrasado (soma 11 ms a cada interrupção)

Algoritmo 2: Berkeley

- Algoritmo usado para a sincronização interna de um grupo de computadores
- 'Servidor de tempo' é ativo (***master***) e coleta os valores de relógios de outros (***slaves***)
- *Master* usa estimativas do RTT para estimar o valor dos relógios dos computadores dentro do grupo de *slaves*
- Hora atual é resultante de uma **média**
- *Master* envia aos *slaves* o **AJUSTE** a ser feito no relógio
- Caso o *master* falhe, um novo computador *master* deve ser eleito

Algoritmo 2: Berkeley

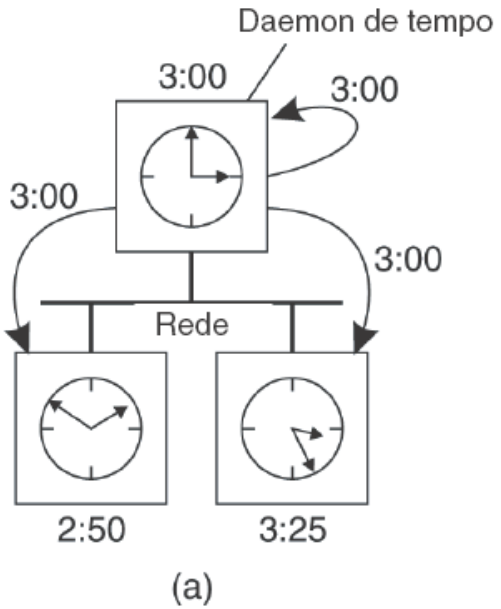


Figura 6.7 (a) O daemon de tempo pergunta a todas as outras máquinas os valores marcados por seus relógios. (b) As máquinas respondem. (c) O daemon de tempo informa a todas como devem ajustar seus relógios.

Algoritmo 2: Berkeley

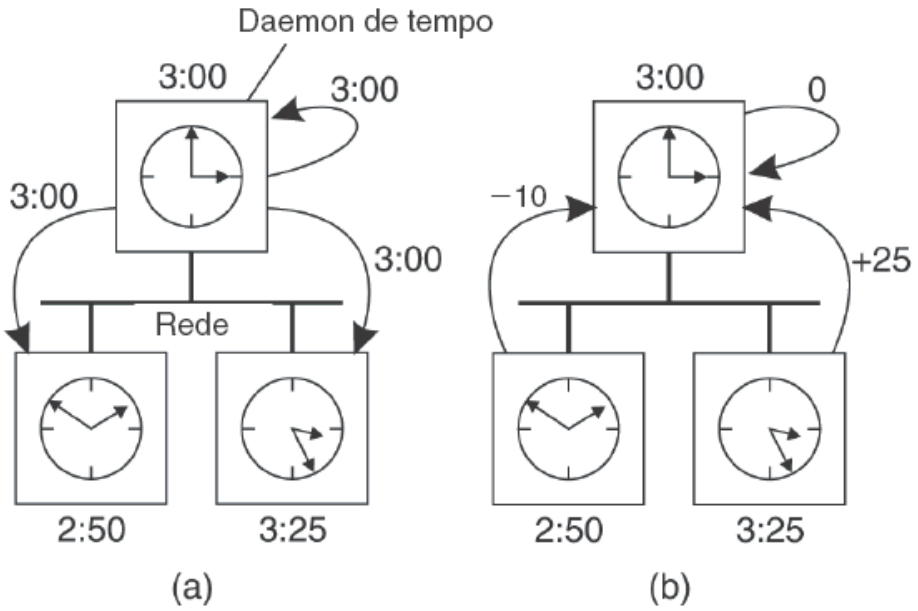


Figura 6.7 (a) O daemon de tempo pergunta a todas as outras máquinas os valores marcados por seus relógios. (b) As máquinas respondem. (c) O daemon de tempo informa a todas como devem ajustar seus relógios.

Algoritmo 2: Berkeley

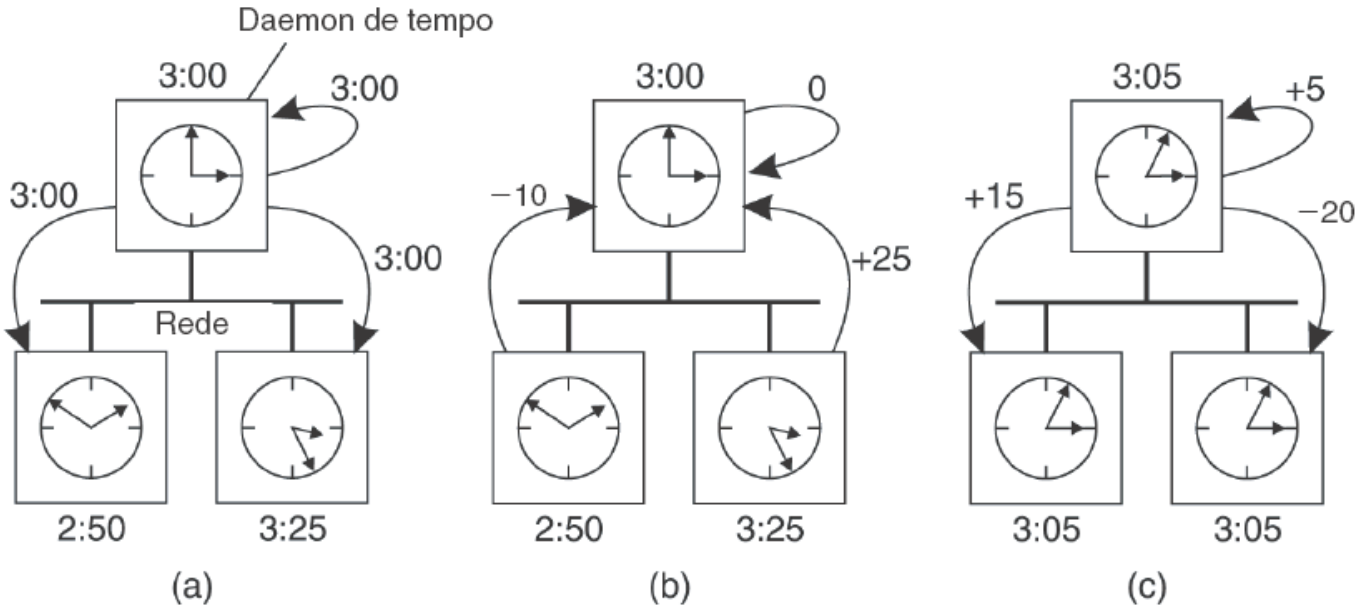


Figura 6.7 (a) O daemon de tempo pergunta a todas as outras máquinas os valores marcados por seus relógios. (b) As máquinas respondem. (c) O daemon de tempo informa a todas como devem ajustar seus relógios.

Protocolo de Tempo de Rede (*Network Time Protocol* - NTP)

- Algoritmos de Cristian e Berkeley são usados para redes locais, enquanto o NTP é um serviço para Internet

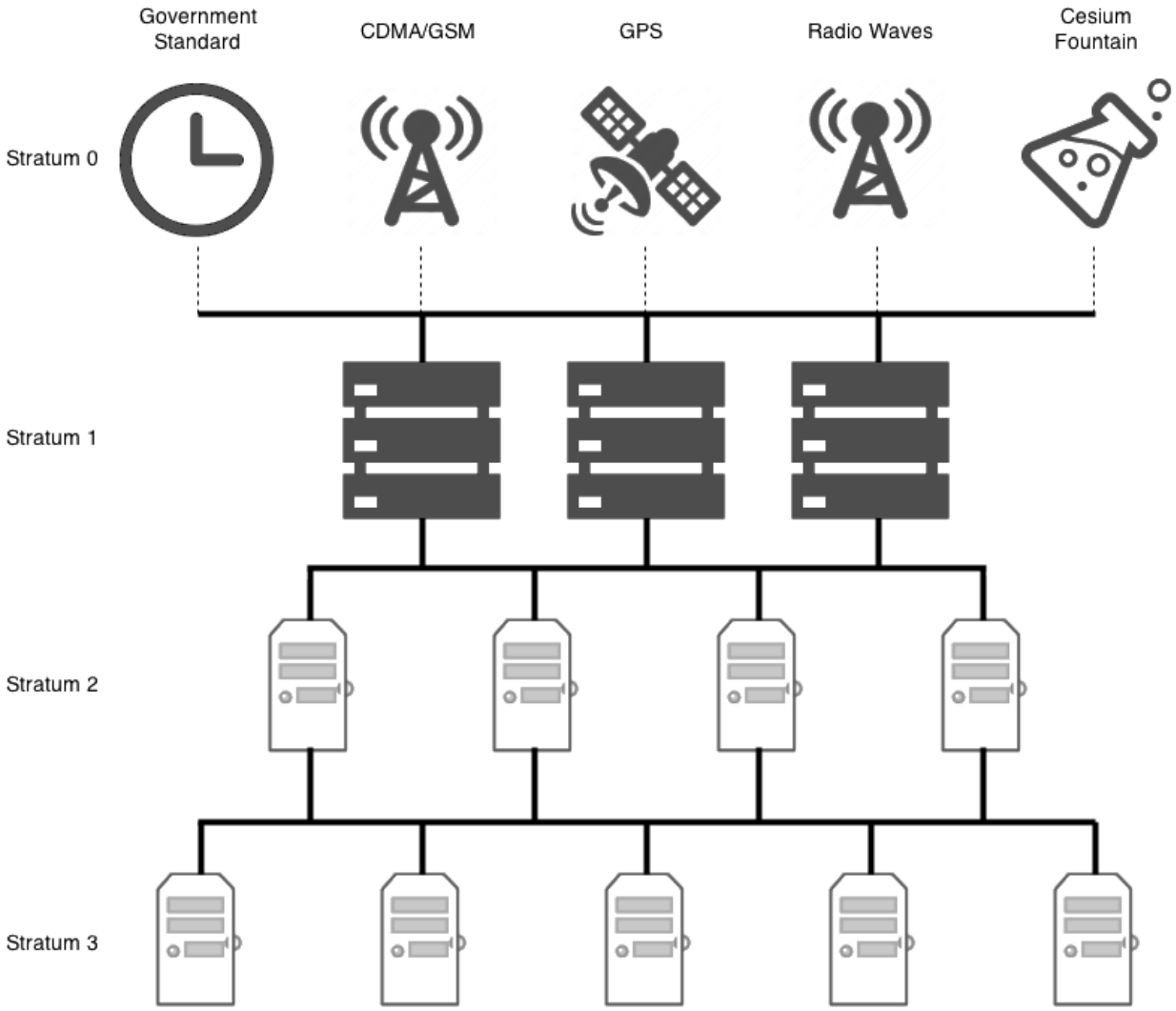
- Protocolo para sincronização de relógios, baseado no UDP
- O ajuste do relógio é feito entre pares de servidores: A consulta B e B consulta A
- É possível calcular o deslocamento θ, bem como a estimativa de atraso δ entre os computadores (RTT/2)
- Oito pares de valores (θ, δ) são armazenados
  - O valor mínimo de δ é usado para a estimativa de atraso entre os dois servidores
  - O valor θ associado a δ é considerado a estimativa mais confiável do deslocamento

Protocolo de Tempo de Rede (*Network Time Protocol* - NTP)

- Apesar de ser um protocolo que permite sincronismo simétrico, um relógio somente é ajustado se a sua precisão é pior do que a do relógio do outro computador

- Servidores são organizados em hierarquia, onde um relógio do nível **k**, possui maior precisão do que um relógio do nível **k+1**
  - Se nível de A for mais baixo do que o nível de B → B se ajustará a A

Protocolo de Tempo de Rede (*Network Time Protocol* - NTP)



Sincronização de relógios

- Até agora → sincronização de **relógios físicos**
  - Todos os envolvidos queriam entrar em consenso sobre o **valor absoluto** da hora atual

- Em outros casos não importa saber a hora atual, somente a ordenação (**tempo relativo**) de eventos ocorridos no sistema distribuído
  - **Relógios lógicos**

Definições iniciais

- Um sistema distribuído pode ser visto como uma coleção **P** de **N** processos $\mathbf{P_i}$**, i = 1,2,… N**

- **Evento** → qualquer ação associada ao processo
  - Envio e recepção de mensagens
  - Mudança do valor de variáveis locais

- Eventos em um processo $\mathbf{P_i}$ podem ser totalmente ordenados pela relação "**acontece antes**" ("***happened before***") "→" , ou seja, **a → b**, se e somente se o evento **a** ocorre antes do evento **b** em $\mathbf{P_i}$

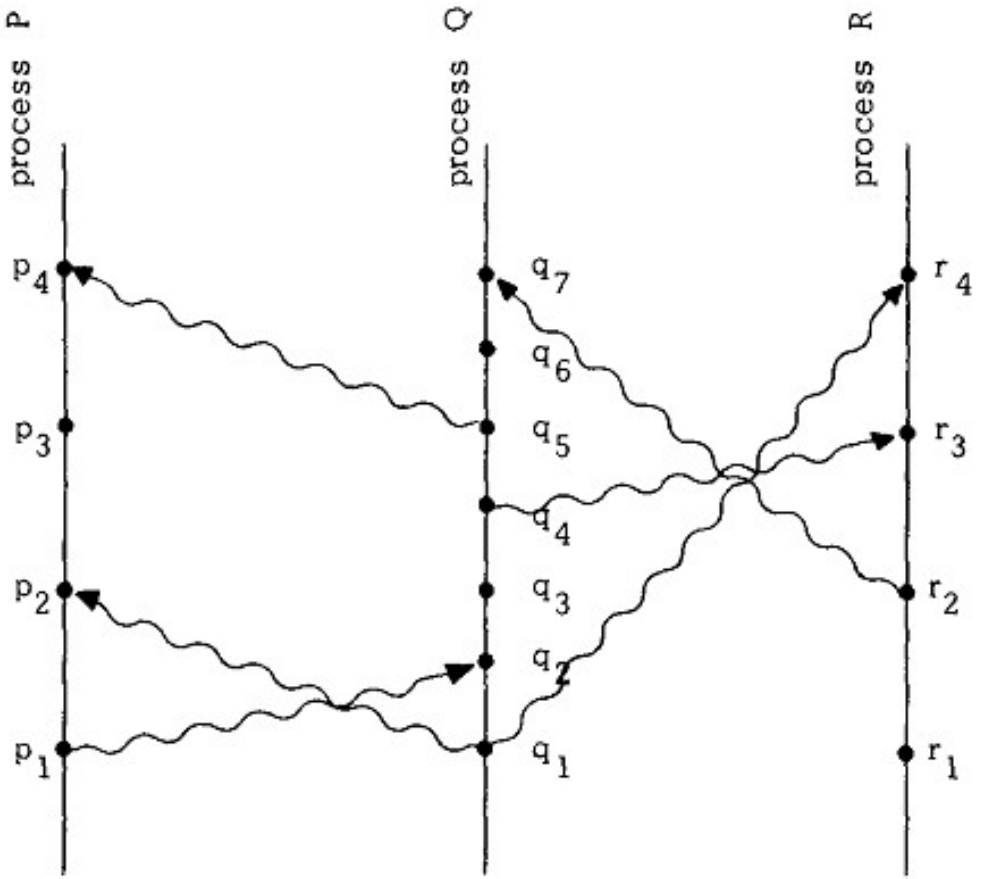
Relação *happened before*

- Se dois eventos ocorrem no mesmo processo, então eles ocorrem na ordem observada pelo próprio processo $\mathbf{P_i}$
- Quando uma mensagem **m** é enviada do processo $\mathbf{P_i}$ ao processo $\mathbf{P_j}$: se **a** é o evento de envio em $\mathbf{P_i}$, e se **b** o de recebimento em $\mathbf{P_j}$, então **a → b**
- Relação “acontece antes” é transitiva:
  - Se **a → b** e **b → c**, então **a → c**

Relação *happened before*

- **Diagrama de tempo**:
  - Linhas verticais: processos
  - Pontos: eventos
  - Setas: mensagens
- Ex.: Podemos dizer $\mathbf{p_1} \rightarrow \mathbf{r_4}$?

- Outra interpretação da relação "acontece antes"
  - Se **a** → **b** significa que **a** pode ter causado o evento **b**
- Dois eventos são ditos concorrentes, se **a** não causa **b** e **b** não causa **a**
  - Ex.: $\mathbf{p_3}$ **e** $\mathbf{q_3}$ são concorrentes!



(mesmo que no diagrama de tempo temos que $\mathbf{q_3}$ tenha ocorrido antes de $\mathbf{p_3}$, o processo **P** não sabe que o processo **Q** fez em $\mathbf{q_3}$ até **P** receber uma mensagem em $\mathbf{p_4}$)

Relação *happened before*

- Se dois eventos **d** e **f** acontecerem em processos diferentes, que não trocam mensagens entre si, nem mesmo indiretamente através de um terceiro processo, então nem **d → f** e nem **f → d** são verdadeiros

- Estes eventos são considerados concorrentes (**d||f** ou **f||d**), o que simplesmente significa que nada pode ser dito a respeito de quando tais eventos ocorreram, ou sobre qual deles ocorreu antes e qual ocorreu depois

Algoritmo de Lamport'78

- **Objetivo:** Introduzir um relógio que atribua um número a um evento, que seja um identificador global do tempo em que o evento ocorreu
  - $\mathbf{C_i}$ é o relógio do processo $\mathbf{P_i}$: função que atribui um número $\mathbf{C_i(a)}$ para qualquer evento **a** em $\mathbf{P_i}$
  - Função não está relacionada com tempo físico
  - Se o evento **a** acontece antes do evento **b**, então **a** recebe uma marca de tempo (relógio) menor que o evento **b**

se a → b, então C(a) < C(b)

O contrário não é verdade!!!

Algoritmo de Lamport'78

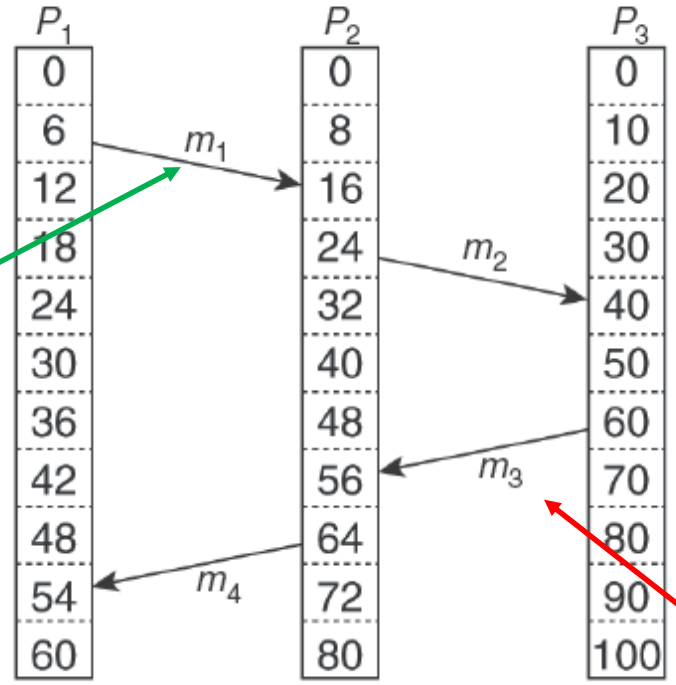
- Condições para garantir a consistência do relógio de Lamport:
  - **C1**: Se **a** e **b** são eventos em um processo $\mathbf{P_i}$ e **a** acontece antes de **b**, então $\mathbf{C_i(a) < C_i(b)}$
  - **C2**: Se no evento **a** o processo $\mathbf{P_i}$ envia uma mensagem, e no evento **b** o processo $\mathbf{P_j}$ recebe esta mensagem, então $\mathbf{C_i(a) < C_j(b)}$

Algoritmo de Lamport'78

- Garantindo as condições para consistência do relógio lógico
  - Para garantir C1: Cada processo $\mathbf{P_i}$ incrementa o relógio $\mathbf{C_i}$ entre dois eventos sucessivos
  - Para garantir C2: Se no evento **a** envia uma mensagem **m** a partir do processo $\mathbf{P_i}$, a mensagem **m** recebe uma marca de tempo (*timestamp*) $\mathbf{T_m = C_i(a)}$. Ao receber a mensagem **m**, $\mathbf{P_j}$ ajusta seu próprio relógio local para $\mathbf{C_j \leftarrow max(C_j, T_m) + 1}$

Algoritmo de Lamport'78

- Ex.: 3 processos com temporizadores não sincronizados



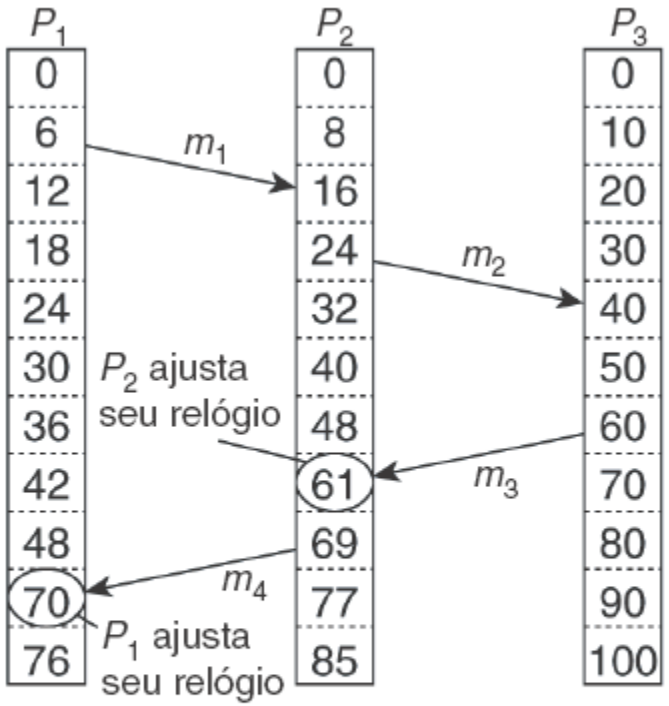
m_1 é enviada no tempo **6** e chega no tempo **16** → **OK!**

m_3 é enviada no tempo **60** e chega no tempo **56** → **Inconsistente!**

Algoritmo de Lamport'78

- Ex.: aplicando o algoritmo de Lamport...

$C_j \leftarrow \max(C_j, T_m) + 1$

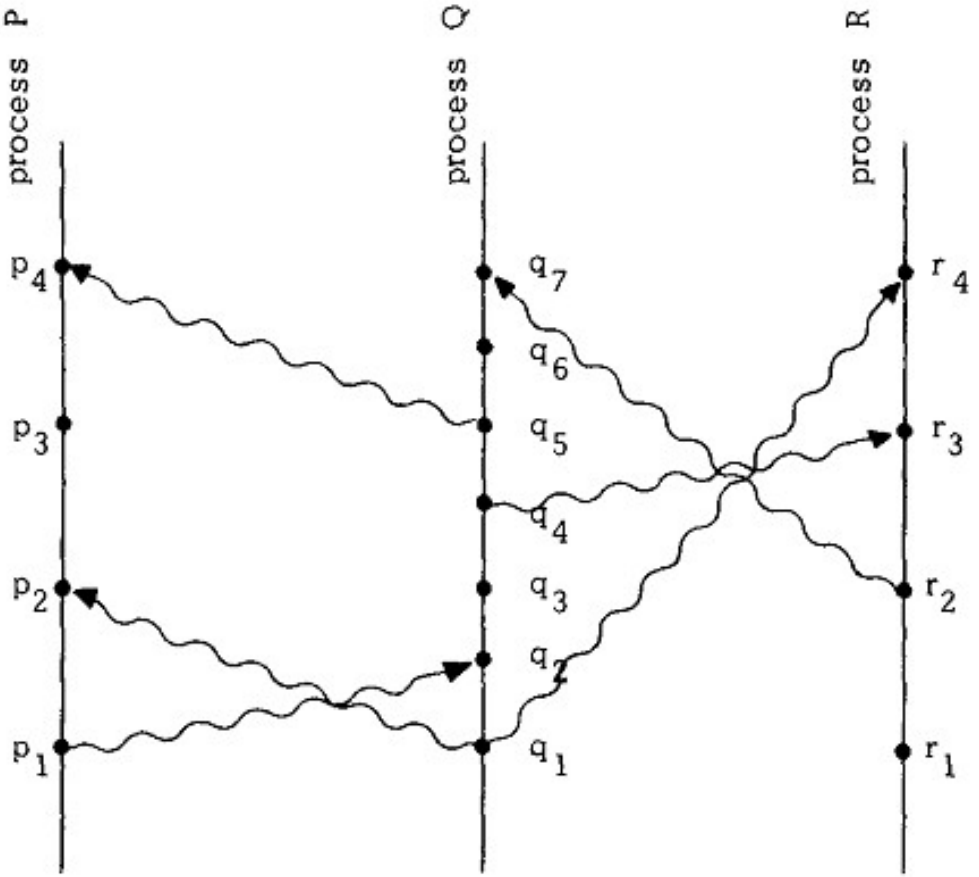


Algoritmo de Lamport'78

- **Exercício**: aplicar Lamport no exemplo abaixo

Condição inicial:
C(i)=0, para todo i
PID(P)=1
PID(Q)=2
PID(R)=3

PIDs dos processos ajudam a ordenar eventos com C(.) iguais



Algoritmo de Lamport'78

- **Aplicação**: banco de dados replicados
  - Exemplo: movimentação em conta de banco
    - Saldo inicial da conta: R$1.000,00
    - Cliente faz depósito de R$100,00 (replica 1)
    - Banco aplica rendimento de 1% de juros (replica 2)
    - Saldo final: **R$1.110,00** ou **R$1.111,00**?

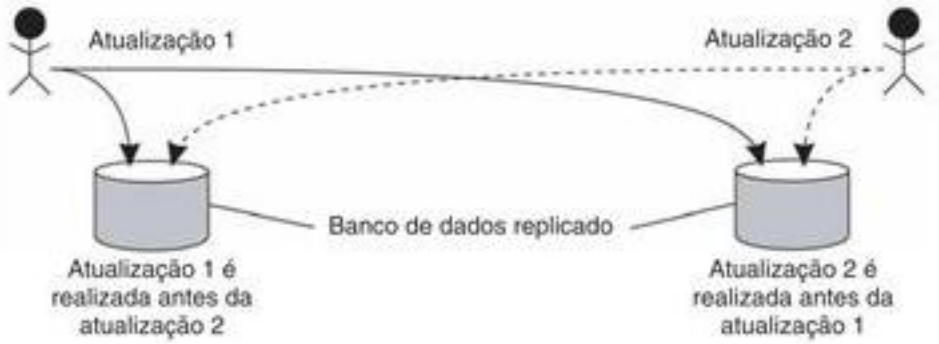


Figura 6.11 Atualização de banco de dados replicado que o deixa em estado inconsistente.

Algoritmo de Lamport'78

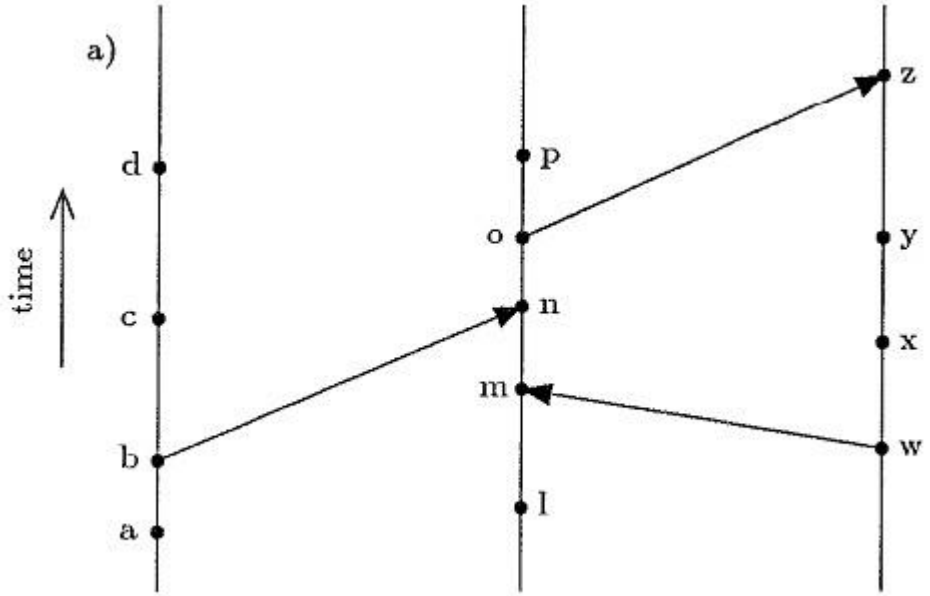
- **Aplicação**: banco de dados replicados
  - **Multicast ordenado**!
  - Mensagens levam C_i do remetente (*timestamp*)
  - Ao receber mensagem, enviar confirmação (ACK) para todos os outros processos
  - Fila de mensagens para a aplicação:
    - Mensagens ordenadas pelo *timestamp*
    - Só são entregues à aplicação quando ACK's de todos os outros processos são recebidos

Relógios Vetoriais

- Por usar inteiros simples como marcas de tempo o algoritmo de Lamport perde informações de vários ordenamentos válidos
  - Ao aplicar o algoritmo de Lamport, temos apenas uma de várias ordenações possíveis, ou seja, um SD totalmente ordenado
- No entanto, em algumas situações, é necessário ter acesso a todas as ordenações parciais possíveis, que representam "fotografias" consistentes (**estados globais**) possíveis do Sistema Distribuído
- Em recuperação a falhas é necessário termos acessos a estes estados globais

Relógios Vetoriais

- Por exemplo, podemos estar interessados em saber se o evento **n** é concorrente com **d**, **c** ou **y**. Aplicando o algoritmo de Lamport, teremos uma ordenação total que fará com que esta informação desapareça!



Relógios Vetoriais (Mattern e Figdge, 1988)

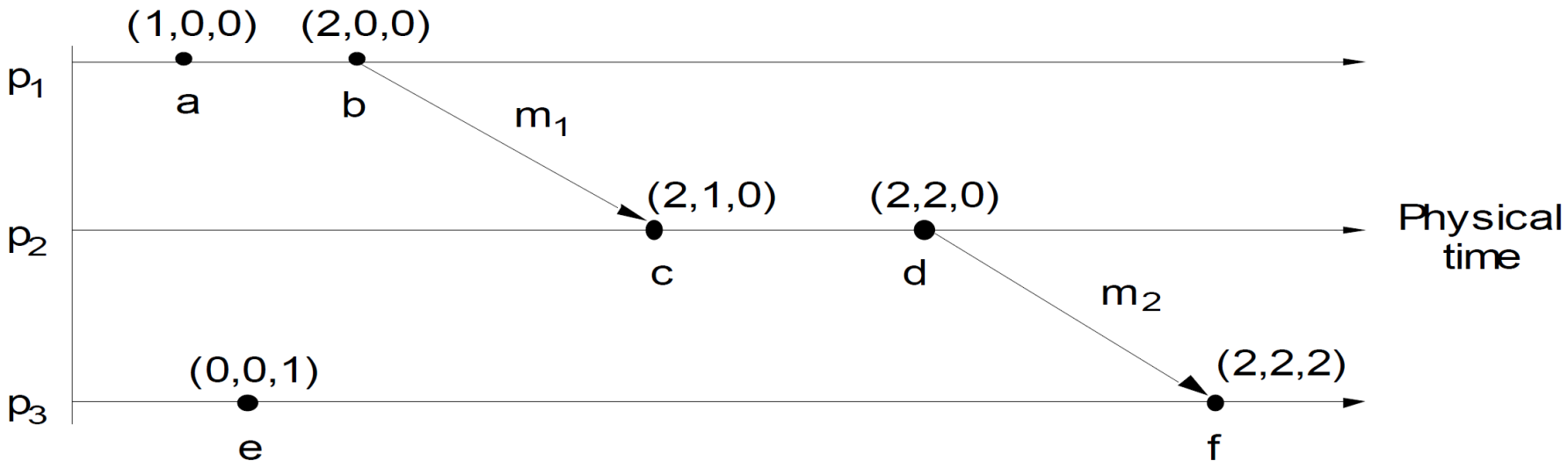
- Vetor de relógios $\mathbf{VC_i}$ no processo $\mathbf{p_i}$ é um vetor de **N** inteiros

  Inicialmente $\mathbf{VC_i[a] = 0}$ para cada **i, a = 1, 2, … N**

  1. antes de cada evento, $\mathbf{p_i}$ executa $\mathbf{VC_i[a] := VC_i[a]+1}$
  2. $\mathbf{p_i}$ anexa $\mathbf{T = VC_i}$ em cada mensagem transmitida
  3. quando recebe **(m, T)** o processo $\mathbf{p_j}$ ajusta seu vetor de relógios $\mathbf{VC_j[b] := max(VC_j[b] , T[b]),}$ b = 1, 2, … N (depois executa etapa 1 e entrega mensagem a aplicação)

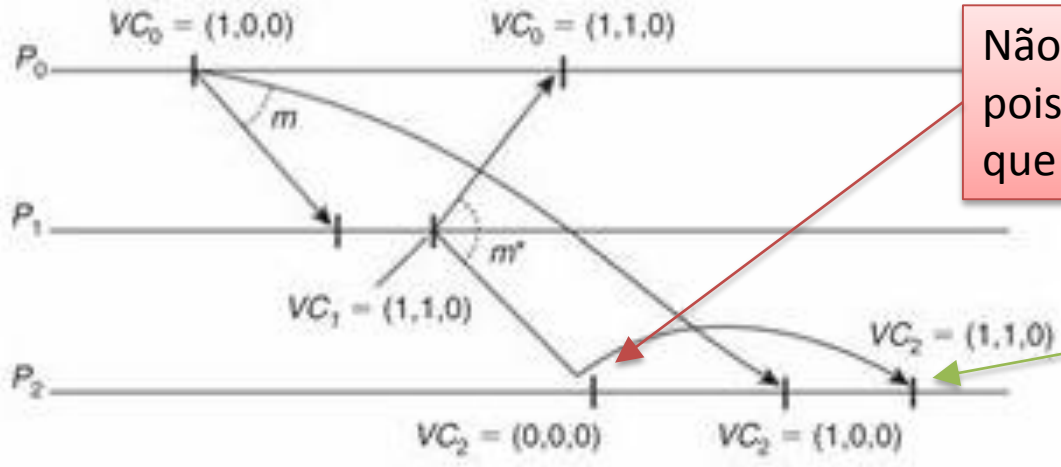
Relógios Vetoriais (Mattern e Figdge, 1988)

- **Exemplo**:



Relógios Vetoriais

- **Aplicação**: multicast ordenado
  - Mesmo objetivo da aplicação de Lamport
    - Só que mais simples... sem ACK's!
  - Atualiza $\mathbf{VC_i}$ apenas quando envia/recebe mensagem
  - Mensagens são entregues apenas se:
    - Mensagem recebida de $\mathbf{P_i}$ é a próxima, e
    - Todas as mensagens vistas por $\mathbf{P_i}$ foram recebidas



Não entrega a mensagem a aplicação, pois P1 já viu uma mensagem de P0 que ainda não chegou em P2

Entrega a mensagem após receber a primeira mensagem de P0

Seminários

- **Grupos**
  - Java RMI
    - Geovani, Rodolpho e Ana Paula
  - RPC (em qualquer linguagem)
    - Cassiano Honório, Diogo Vieira, William Messias
  - WebServices SOAP
    - Victor Pedro, Felipe Melo
  - WebServices RESTfull
    - Miguel, Marcos e Válber
  - Globus Toolkit (middleware para grid computing)
  - SDs Baseados em Objetos
    - Jéssica Raposo e Juliane Marinho
  - Sistemas de Arquivos Distribuídos
    - Larissa Ferrarez e Kesia Braga, Jaqueline
  - SDs Baseados na Web
    - Camilla, Thiago
  - SDs Baseados em Coordenação
    - Elaine e Lucas Pinheiro