UNIVERSIDADE FEDERAL DO CEARÁ - Campus Quixadá
Cursos: SI, ES, RC, CC e EC
Código: QXD0043
Disciplina: Sistemas Distribuídos

Capítulo 14 – Tempo e Estados Globais

Prof. Rafael Braga

Agenda

- 11.2 - Relógios, eventos e estados de processos
- 11.3 - Sincronizando relógios físicos
- 11.4 - Tempo lógico e relógios lógicos
- 11.5 – Estados Globais
- 11.6 – Depuração distribuídas

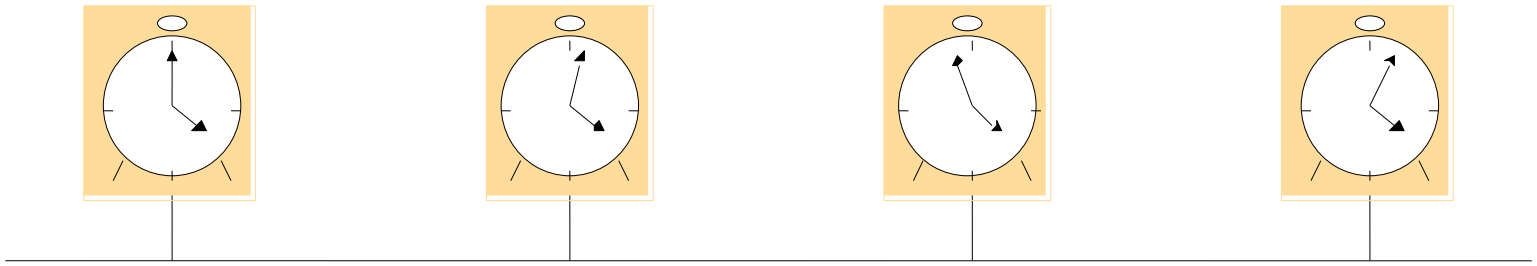
Relógios

- Relógio físico
  - Oscilações em cristal com frequência conhecia;
    - Por exemplo os relógios de cristais de quartzo;
    - Geração de interrupções (*clock ticks*);
  - Relógio de hardware do nó $H_i(t)$
  - Relógio de software $C_i(t)=\alpha H_i(t)+\beta$
    - Eventos sucessivos dependem da resolução do relógio
  - A taxa de ocorrência de eventos depende do cumprimento do ciclo do processador.

© Pearson Education 2012

Defasagem de relógios e derivação de relógios

- Os relógios tendem a não estar sincronizados;
- A diferença entre dois relógios é a ***distorção*** (*skew*);
- A diferença entre relógios físico é a ***derivação*** (*drift*);
- A diferença entre relógios local e de referência é a ***taxa de derivação*** (*drif rate*);
  - Por exemplo, no relógio de quartzo tem-se 1 segundo a cada 1.000.000 seg. (10^6) (11,6 dias)
  - Em quartzo de alta precisão a precisão é de 10^7 ou 10^8;



© Pearson Education 2012

Tempo Universal Coordenado

- Sincronização com fontes externas altamente precisas;
- Relógios físicos mais precisos usam osciladores atômicos;
  - Tempo real decorrido ou tempo atômico internacional;
  - O ***segundo*** padrão é 9.192.631.770 transições do Césio-133;
  - Taxa de derivação de 10^13;
- Descompasso entre o tempo astronômico e o tempo atômico;
- Tempo Universal Coordenado (UTC) é um padrão internacional;
  - Baseado em tempo atômico;

© Pearson Education 2012

Agenda

- 11.2 - Relógios, eventos e estados de processos
- 11.3 - Sincronizando relógios físicos
- 11.4 - Tempo lógico e relógios lógicos
- 11.5 – Estados Globais
- 11.6 – Depuração distribuídas

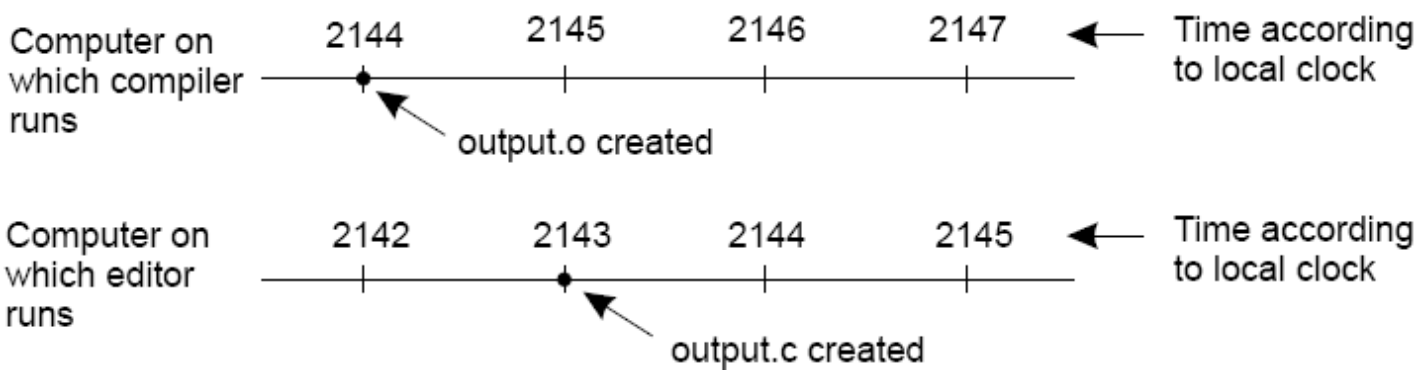
Sincronização de Relógios

- Sincronização internaa x sincronização externa;
  - Limite de sincronização D;
- Correção baseada numa taxa de derivação conhecida (ρ);
- Monotonicidade é a condição de que um relógio apenas sempre avance;
- Um relógio sem correção é dito com falho
  - Falha de colapso é quando o relógio para de “tiquetaquear”;
  - Qualquer outra falha é chamada de falha arbitrária;
    - Bug Y2K (31 de dezembro de 2000);
    - Bateria fraca
    - Derivação muito grande;

© Pearson Education 2012

Sincronização de Relógios

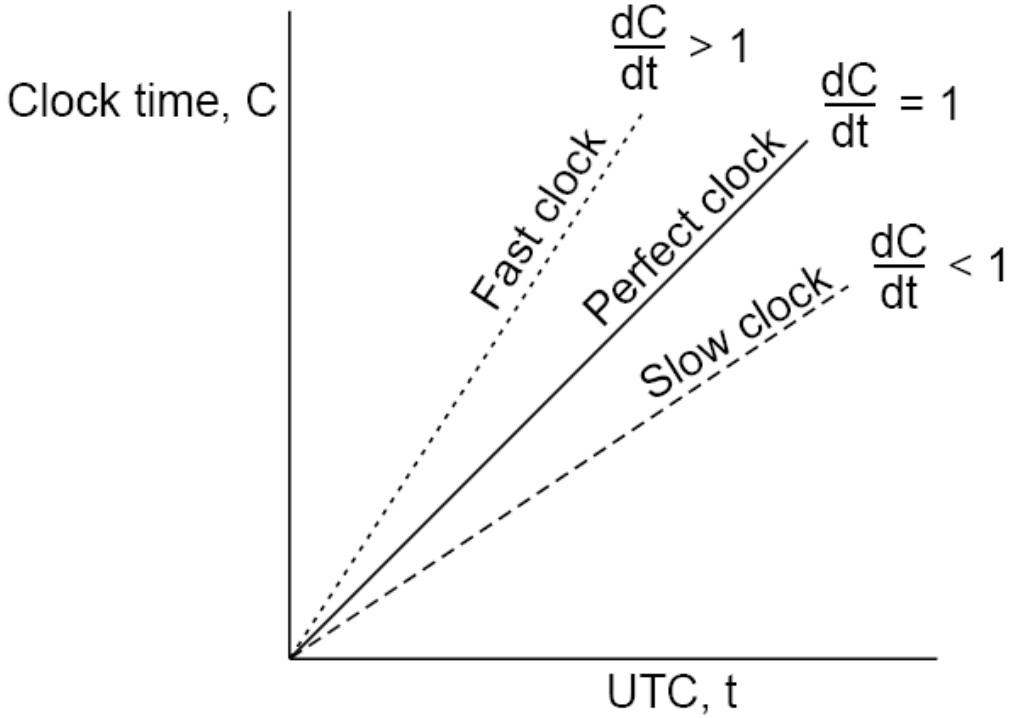
- Os relógios não precisam ser precisos para serem corretos
  - Logo, numa sincronização interna, vale mais o funcionamento correto;
- Quando cada máquina tem seu próprio relógio, um evento que ocorreu depois de outro pode, não obstante, ser associado a um tempo anterior.



© Pearson Education 2012

Algoritmos de Sincronização de Relógios

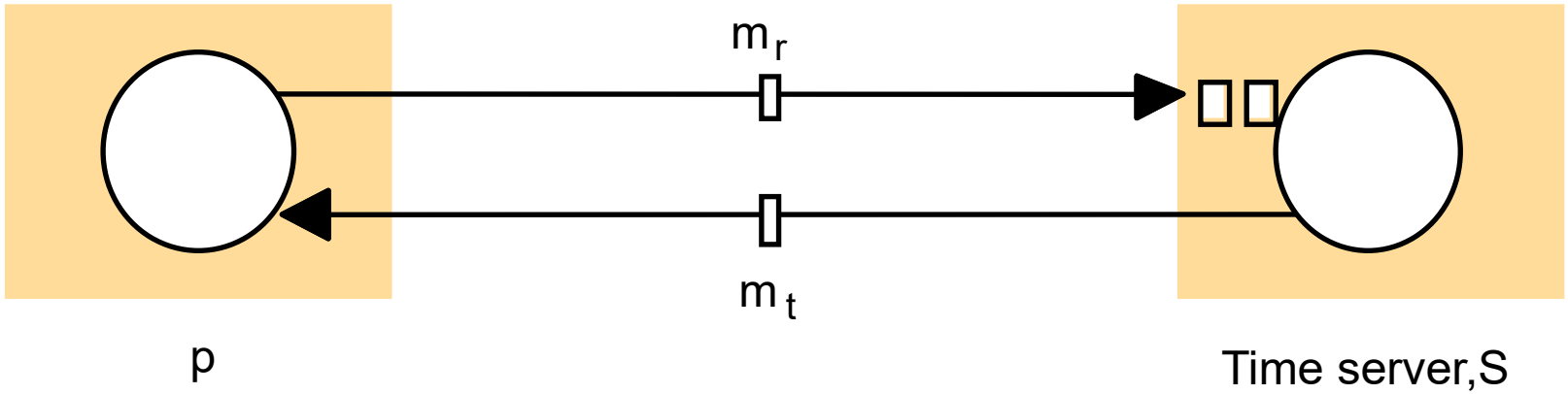
- Algoritmos de sincronização interna e externa;
- A relação entre tempo de relógio e o tempo UTC quando os relógios “ticam” com taxas diferentes.
- Sincronização em sistemas síncronos;
  - Não existe na prática;
  - A maioria dos sistemas distribuídos é composta por sistemas assíncronos;



© Pearson Education 2012

Algoritmo de Cristian

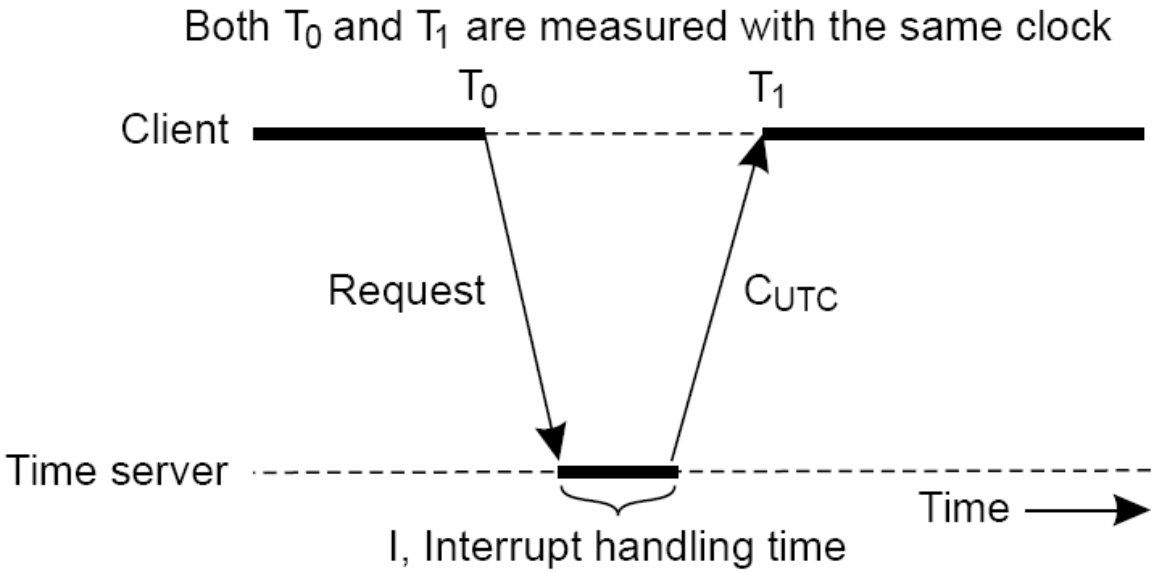
- Em 1989, Cristian sugeriu a sincronização de relógios usando um servidor de tempo que recebe sinais de uma fonte UTC;
- Algoritmo probabilístico (T_{trans} curto, uma pequena fração do segundo);



© Pearson Education 2012

Algoritmo de Cristian

- Obtendo o tempo atual a partir de um servidor de tempo;
- Intervalo [$t+T_{min}$, $t+T_{trans}-T_{min}$];
- O problema é o de utilizar apenas um servidor, que pode falhar;
- Sugestão é de utilizar o conjunto de servidores de tempo UTC;

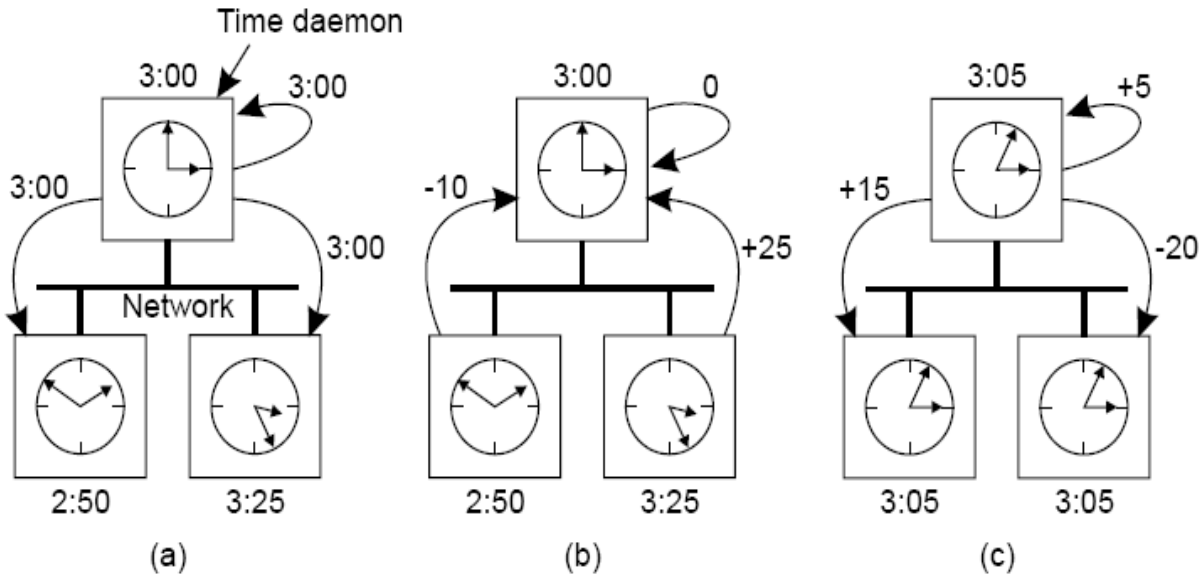


© Pearson Education 2012

O Algoritmo de Berkeley

- Algoritmo de sincronização interna;
- Desenvolvido para o UNIX Berkeley;
- Utilizada a arquitetura de computador coordenador;

a) O *daemon* de tempo pergunta a todas as máquinas os valores de seus relógios locais
b) As máquinas respondem
c) O *daemon* de tempo instrui todas as máquinas a atualizarem seus relógios



© Pearson Education 2012

Network Time Protocol (NTP)

- Enquanto o algoritmo de Berkeley é adequado para LAN's e intranets, o NTP [Mills 1995] visa prover um "Serviço de Tempo" para sincronização de relógios na Internet.
  - Fornecer um serviço que permita aos clientes na Internet serem sincronizados precisamente com o UTC;
  - Fornecer um serviço confiável que possa sobreviver a longas perdas de conectividade;
  - Permitir que os clientes sejam sincronizados de forma suficientemente frequente para compensar as taxas de derivação encontradas na maioria dos computadores;
  - Fornecer proteção contra interferência no serviço de tempo, seja mal-intencionada ou acidental.

© Pearson Education 2012

Network Time Protocol (NTP)

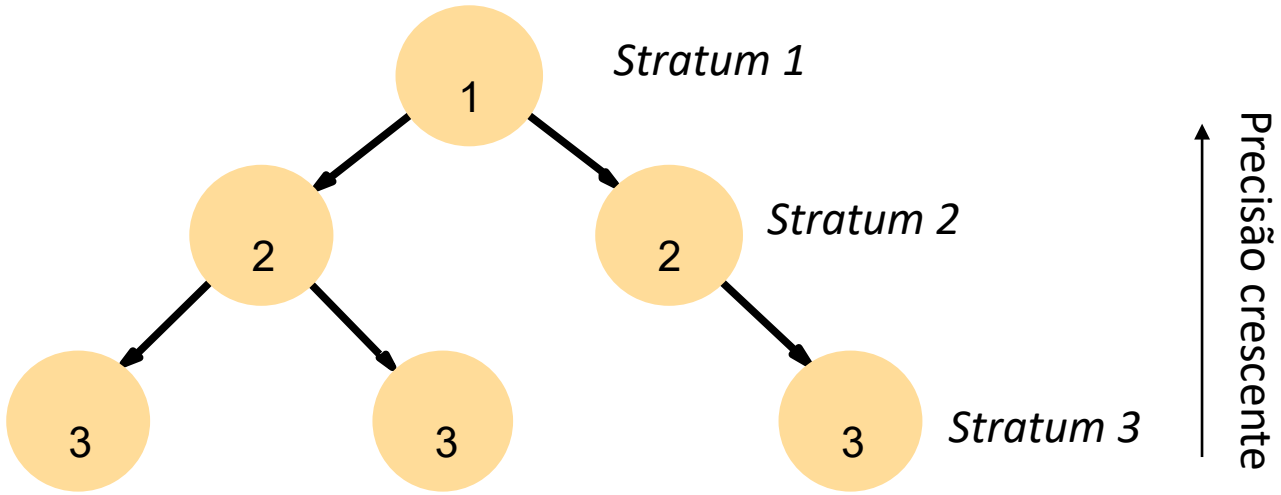
NTP consiste de uma hierarquia lógica de servidores de tempo, na qual:

- servidores primários têm uma fonte externa de tempo (relógio de rádio provendo UTC)
- servidores de uma camada N são fonte de sincronização para servidores da camada N+1, e os hosts dos usuários são as folhas da árvore
- servidores podem se sincronizar de 3 modos: multicast, RPC e simétrico
- servidores mais altos na hierarquia têm relógios mais precisos do que os abaixo

© Pearson Education 2012

Network Time Protocol (NTP)

- Um exemplo de sub-rede de sincronização em uma implementação de NTP.
- a sub-rede de sincronização pode se reconfigurar (p.ex. se um servidor falha)



Nota: Setas denotam controle de sincronização, números denotam estratos.

© Pearson Education 2012

NTP: Modos de Sincronização

Multicast:

- deve ser usado em LANs de alta velocidade
- periodicamente um servidor primário difunde o seu tempo para certo conjunto de servidores, que ajustam os seus relógios assumindo *delay* pequeno de transmissão
- com este modo não se consegue alta precisão na sincronização (pode ser suficiente!)

Chamada a Procedimento (RPC):

- sincronização através de invocação ponto-a-ponto tipo *request-reply*
- valor recebido da consulta e estimativa do *round-trip delay* são usados para ajustar o próprio relógio (como em [Cristian89])
- adequado para servidores NTP em redes distintas; garante alta precisão

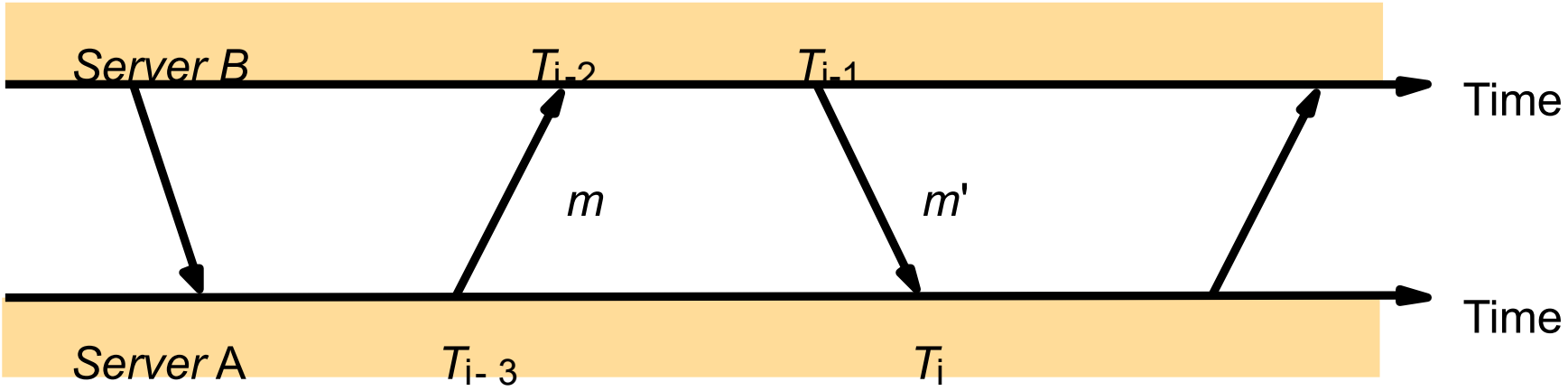
Simétrico:

- usado para sincronização entre servidores de tempo (baixo stratum), que requer altíssima precisão
- servidores trocam mensagens e mantém registros dos valores medidos/ajustados ao longo do tempo (associação) para melhorar cada vez mais a precisão

© Pearson Education 2012

NTP: O protocolo

- Troca de mensagens em ***todos*** os modos usa o **UDP** (não-confiável).
- Nos modos RPC e simétrico servidores trocam pares de mensagens, e cada mensagem carrega os *timestamps* de eventos de tempo recentes.



Sejam $\mathbf{T_M}$ e $\mathbf{T_{M'}}$, tempos reais de transmissão de M e M'.

© Pearson Education 2012

NTP: O protocolo

- Se o offset real do relógio em B relativo ao relógio em A, for o e os tempos de transmissão reais de m e m' forem t e t', então:

$$T_{i-2} = T_{i-3} + t + o_i \quad \text{e} \quad T_i = T_{i-1} + t' - o_i$$

- Ao receber uma mensagem NTP, o servidor registra o momento de chegada (T_i), e usando os tempos recebidos na mensagem, calcula:
  - *Compensação (offset)* o_i (estimativa da diferença entre os relógios)

$$o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$$

  - *Atraso (delay)* d (tempo total de transmissão das duas mensagens)

$$d = T_M + T_{M'} = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)$$

- Assim, obtém-se que $o = o_i + (TN + TM)/2$, ou $o - d/2 \leq \beta \leq o + d/2$, *delay* d indica a precisão da estimativa o.

© Pearson Education 2012

NTP: O protocolo

- Servidores NTP usam um algoritmo para seleção de pares (o, d) obtidos em interações sucessivas (e recentes) com cada outro servidor.
- A partir destes pares calculam a qualidade da estimativa como uma variável estatística (filtro de dispersão). Uma alta dispersão de filtragem indica uma baixa confiabilidade da estimativa.
- Cada servidor NTP mantém armazenados os 8 pares (o, d) mais recentes, e estimativa de *offset* o correspondente ao menor valor d é selecionado.

© Pearson Education 2012

NTP: O protocolo

Além disto cada servidor interage com vários outros servidores, e executa um algoritmo de seleção de fonte de sincronização:

- mantém registrada a precisão obtida na interação com cada outro servidor
- difunde os dados sobre dispersão de filtragem para os demais servidores, (permitindo que cada servidor possa calcular a sua dispersão de sincronismo com relação ao servidor raiz)
- eventualmente escolhe novo servidor de referência para a sincronização, que é
  - um servidor do stratus imediatamente anterior e
  - aquele que apresenta uma menor disperão de sincronismo com a raiz

O NTP consegue uma precisão da ordem de 10^{-2} s na Internet e 10^{-3} s em LANs.

© Pearson Education 2012

Agenda

- 11.2 - Relógios, eventos e estados de processos
- 11.3 - Sincronizando relógios físicos
- 11.4 - Tempo lógico e relógios lógicos
- 11.5 – Estados Globais
- 11.6 – Depuração distribuídas

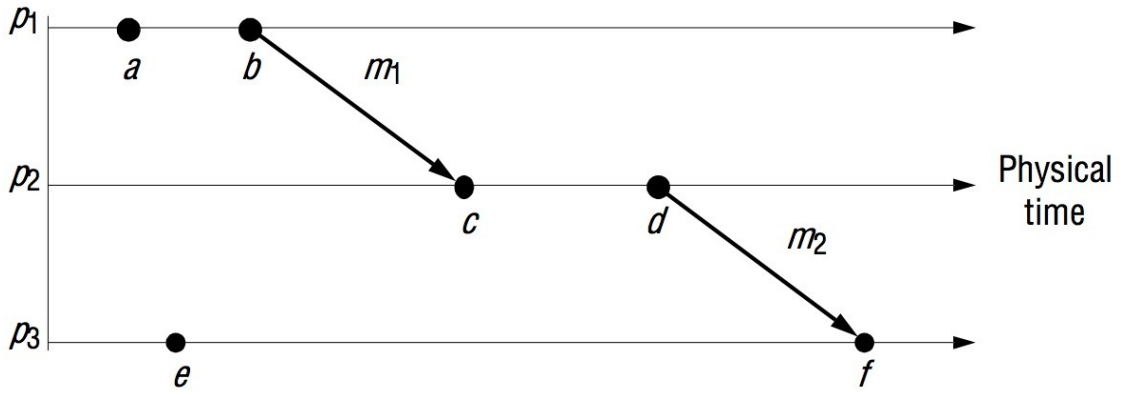
Tempo lógico e Relógios lógicos

- Lamport (1978) afirma que não é possível descobrir a ordem de qualquer par de evento arbitrário.
  - Se dois eventos ocorrem no mesmo processador, então a ordem é a que o processador observou;
  - Se uma mensagem é enviada entre processos, o evento que enviou ocorreu antes do evento de recepção.
- Utilizar um esquema baseado na **causalidade** = **relação *happened-before (hb)*** *= ordem causal = ordem causal potencial**
- Representação: →

© Pearson Education 2012

Tempo lógico e Relógios lógicos

a → b, haja vista que p1 (a → b), semelhantemente, c → d.
Além disso, b → c e d → f, por troca de mensagem. Logo a → f.
a || e são eventos concorrentes.



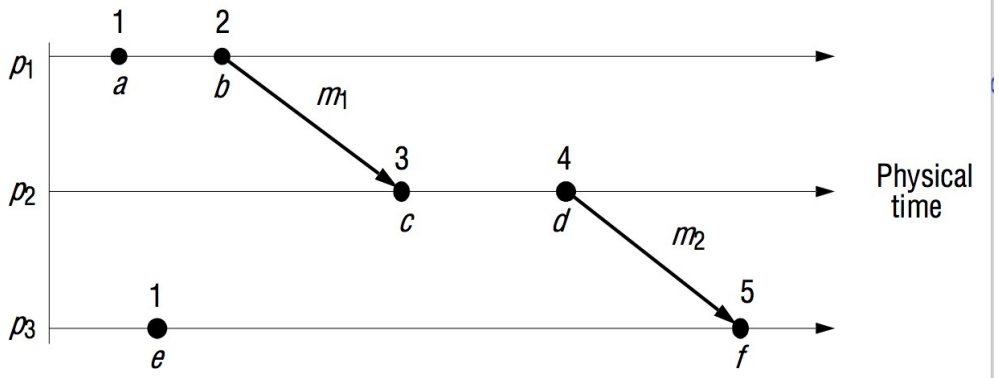
Eventos ocorrendo em três processos

Marcas de tempo de Lamport para os eventos

Leslie Lamport propôs o conceito de relógio lógico que é consistente com a relação de causalidade entre eventos:

Algoritmo:

- cada processo p tem um contador local L(p), com valor inicial 0
- para cada evento executado (exceto receive) no processo p_i faça $L_i(p) := L_i(p) + 1$
- ao enviar uma mensagem m, adicione o valor corrente $L(m) := L_i(p)$
- quando mensagem m é entregue a processo q, este faz $L(q) := \max(L(m),L(q))+1$



Consistência com a causalidade significa:

- se a → b, então: L(*a*) < L(*b*).
- se a || b, então: L(*a*) e L(*b*) podem ter valores arbitrários;

Nesse exemplo:

- *L(b) > L(e)*, mas b || e.

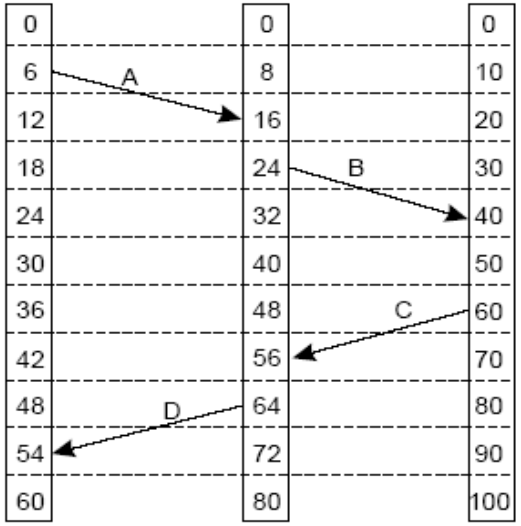
© Pearson Education 2012

Marcas de Tempo de Lamport

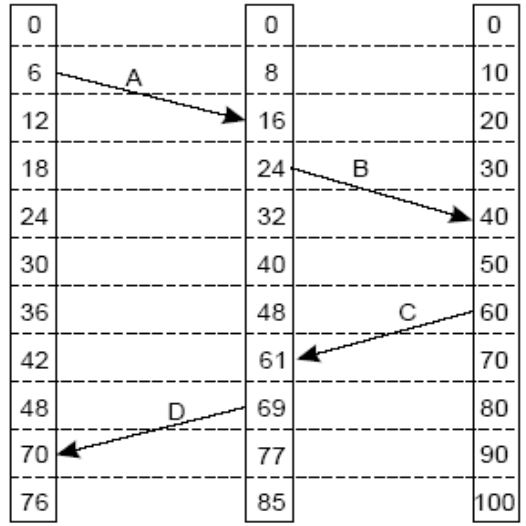
- Exemplo

a) Três processos, cada um com seu próprio relógio. Os relógios “correm” a taxas diferentes.

b) O algoritmo de Lamport corrige os relógios (relógios lógicos).



(a)



(b)

© Pearson Education 2012

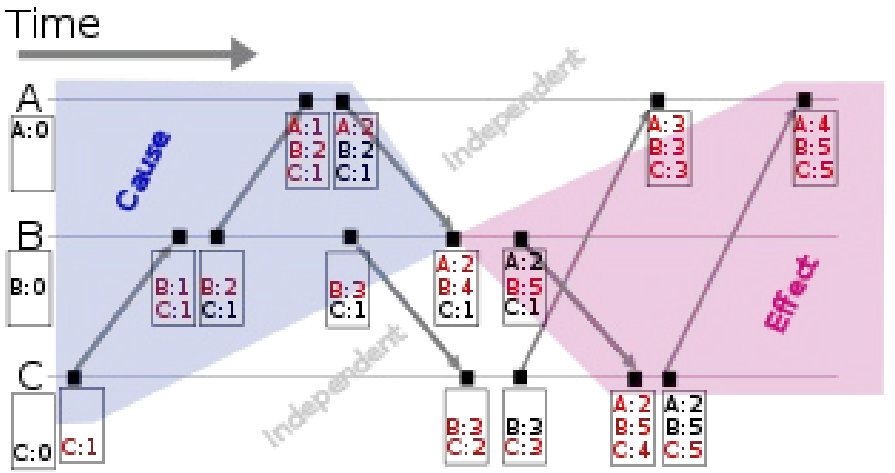
Relógios vetoriais

- Criados por Mattern, Figdge,1988.
- Implementados para evitar a limitação dos relógios de Lamport: **C(a) < C(b)** não implica **a** “acontece antes” de **b**.
  - Ou seja, a limitação de a partir de *L(e) < L(e’)* não ser possível afirmar que e → e’.
- Vetores com marcas de tempo são usados para os eventos locais em cada processo.

© Pearson Education 2012

Relógios vetoriais

- Considere um sistema distribuído com **N** *processos*;
- Cada processo do sistema tem o seu relógio vetorial: **Vi**;
- Cada **relógio vetorial** terá **N** posições: **Vi[N]**;
- A posição **Vi[i]** contém o número de eventos que ocorreram no processo **pi**;
- Uma **posição Vi[j]** com **i, j** contém o número de eventos que ocorreram em **pj** e que potencialmente afetaram **pi**;
- Exemplo: num SD com 3 processos temos inicialmente: $V_1[3] = \{0, 0, 0\}$, $V_2[3] = \{0, 0, 0\}$, $V_3[3] = \{0, 0, 0\}$



Exemplo de um sistema de relógios vetoriais
Fonte: https://pt.wikipedia.org/wiki/Relógios_vetoriais

Relógios vetoriais

- Regras de atualização
  - **Inicialmente (RV1):**

    $V_i[j] = 0$ para j=1 até N
  - **Evento local (RV2):**

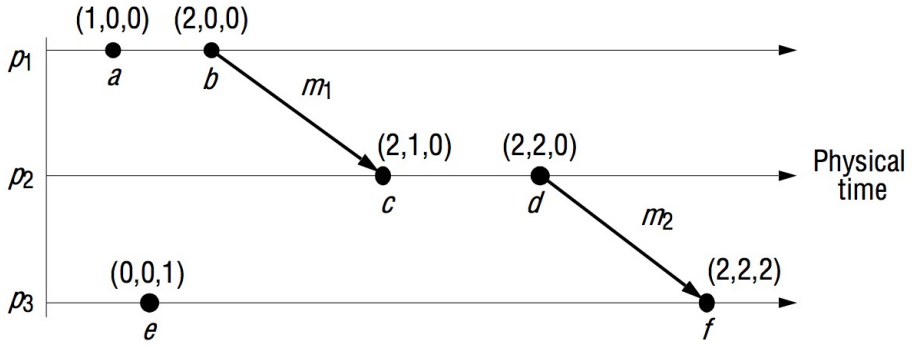
    Antes de colocar o *timestamp* num evento, o processo faz: **$V_i[i] := V_i[i] + 1$**
  - **Send (RV3):**

    p_i inclui o valor de **V_i** em toda mensagem que ele envia
  - **Receive (RV4):**

    quando **p_i** recebe uma mensagem com *timestamp* **t**, ele faz: **$V_i[j] := \max(V_i[j], t[j])$**, para j = 1, 2, ..., N. Conhecido como integração (*merge*).

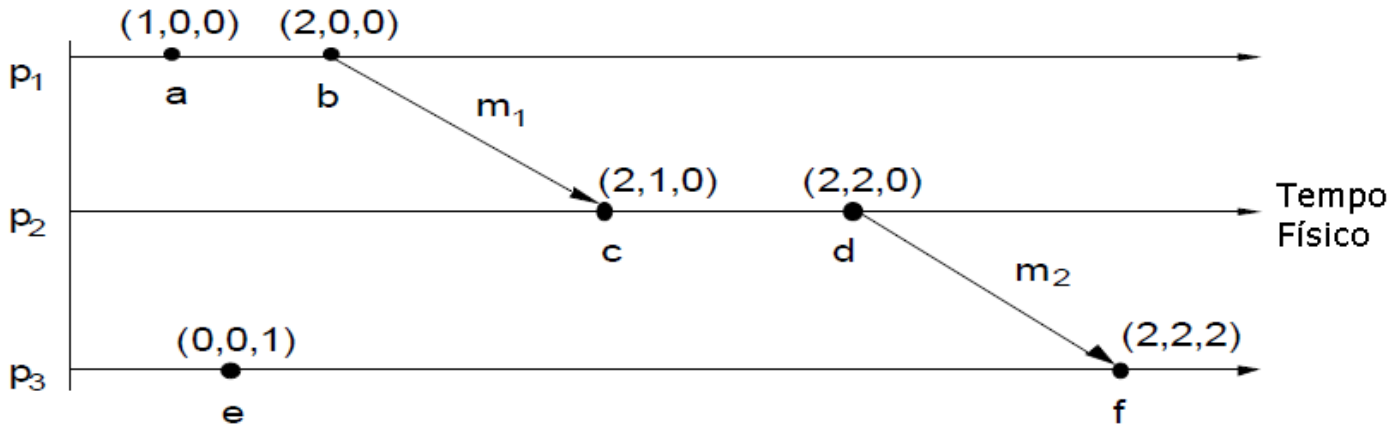
Relógios vetoriais (exemplo)

- p1: a(1,0,0); b (2,0,0) envia (2,0,0) juntamente com a mensagem m1.
- Em p2, no recebimento de m1, o vetor de relógios é modificado para *max*.
- ((0,0,0), (2,0,0)) = (2, 0, 0) adicionando 1 ao seu próprio relógio = (2,1,0)
- Neste caso, o evento c 'sabe' que ocorreram 2 eventos no processo p1 antes da ocorrência do evento c em p2
- **=,<=, max:** devem ser realizadas entre pares de elementos



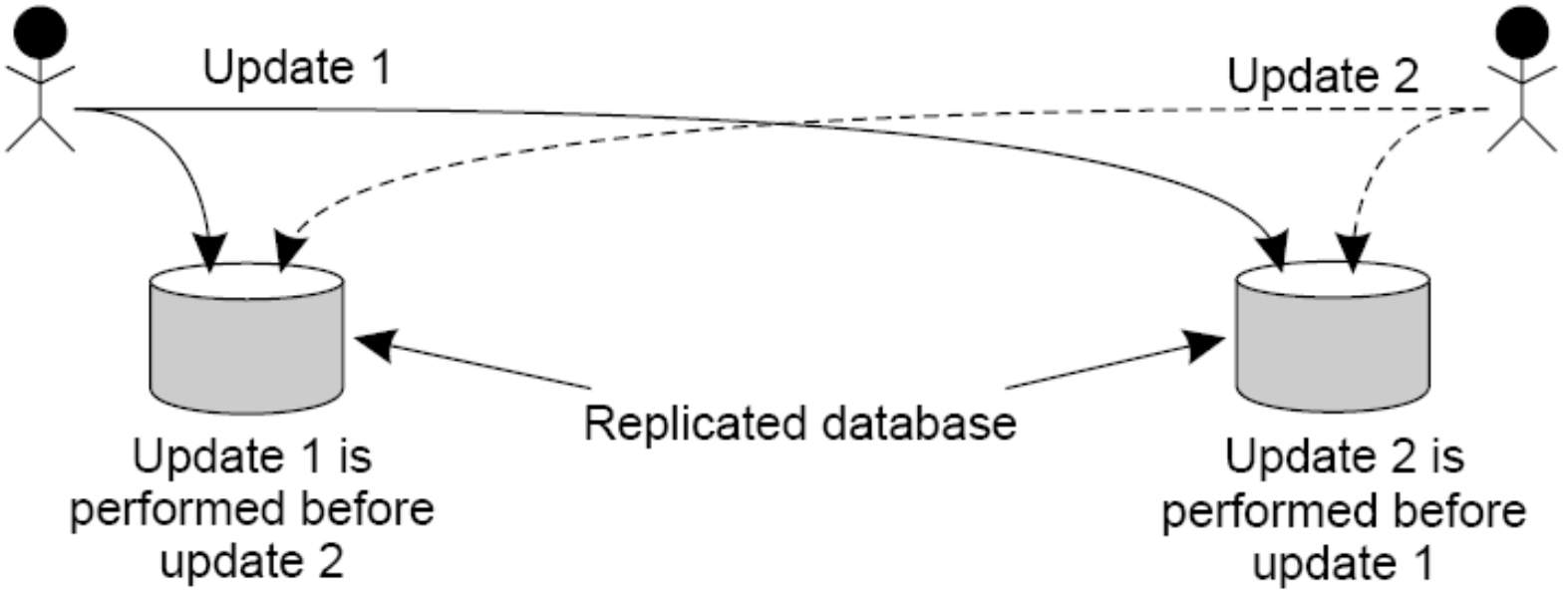
Relógios vetoriais (exemplo)

- Analogamente, para dois processos concorrentes, por exemplo os eventos **c** e **e (c || e)**, mas nem **VC(e) <= VC(c)** nem **VC(c) <=VC(e)** podem ser afirmados!



Exemplo: Multicast Totalmente Ordenado

- Atualização de um banco de dados replicado deixando-o em um estado inconsistente.



© Pearson Education 2012

Agenda

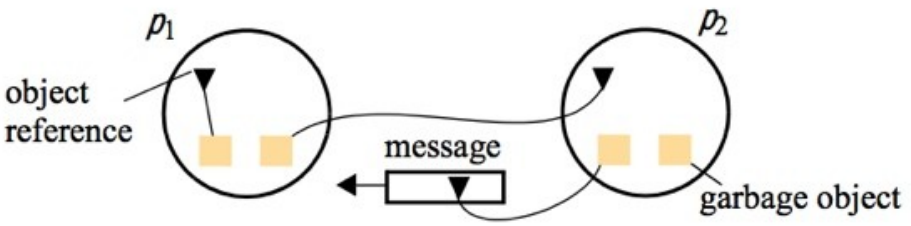
- 11.2 - Relógios, eventos e estados de processos
- 11.3 - Sincronizando relógios físicos
- 11.4 - Tempo lógico e relógios lógicos
- 11.5 – Estados Globais
- 11.6 – Depuração distribuídas

Estados Globais

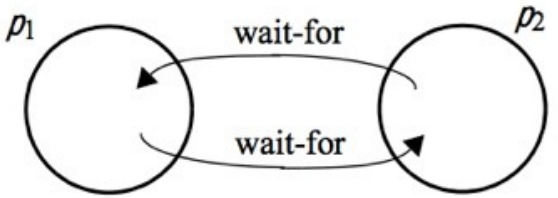
- Motivação
  - Suponha uma computação distribuída onde queremos responder questões tais como:
    - O sistema está “travado” ou em (*deadlock)*?
    - Quantos processos estão acessando um arquivo?
    - Qual o saldo atual de uma agência bancária?
  - Para **respondê-las**, podemos enviar *requests* para todos os processos
  - O **problema** é que enquanto os processos respondem, eles continuam trocando mensagens
  - Exemplos…

Detecção de propriedades globais

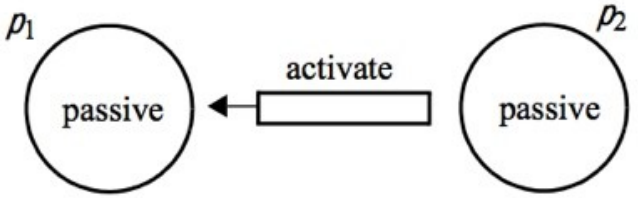
- Alguns problemas relevantes:
  - Coleta de lixo distribuída;
  - Detecção de impasses distribuída;
  - Detecção de término distribuída;
  - Depuração distribuída.



(a) Garbage collection



(b) Deadlock



(c) Termination

© Pearson Education 2012

Formalização

- Uma computação distribuída é uma **execução** de um **programa distribuído** por uma coleção de **processos**
- Cada processo gera uma sequência de eventos (histórico local).
- Eventos podem ser internos ou de comunicação
  - Histórico local do processo i: **=…**
  - Histórico local parcial:**= …**
  - Estados do estado do processo : **= …**
  - Histórico global:

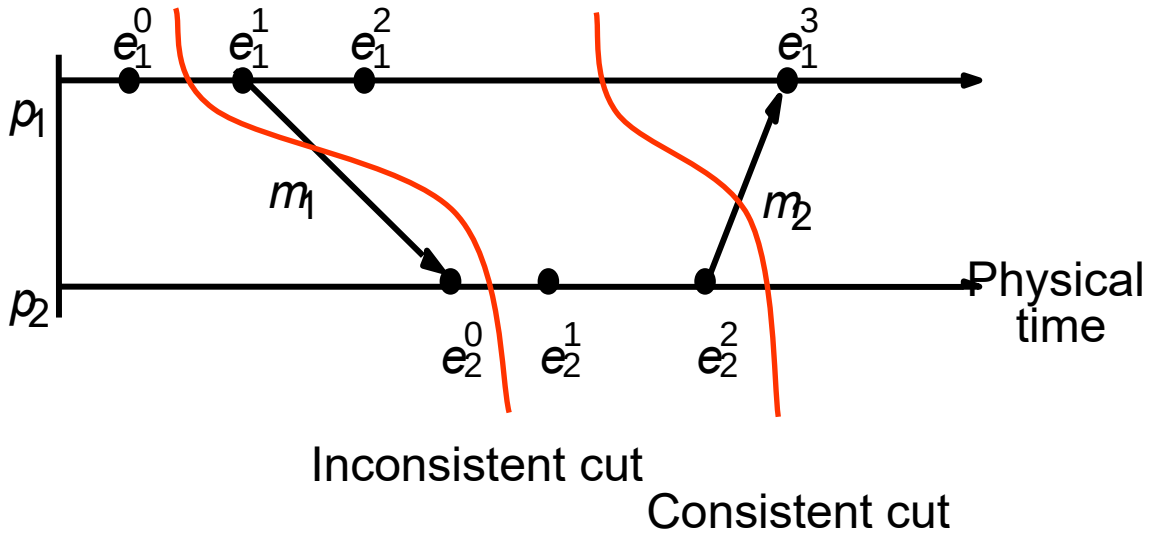
Formalização

- Seja, então, o estado global
- Mas, quais estados globais são significativos? i.e. Quais estados do processo poderiam ter ocorrido ao mesmo tempo?
- Um ***estado global*** corresponde aos prefixos iniciais dos históricos de processo individuais.
- Um corte da execução de um sistema é um subsistema de seu histórico global;

Corte: coleção de históricos locais parciais. *C_i denota até onde vai o histórico parcial para cada proc. i*	$C = \bigcup_{i=1}^{N} h_i^{Ci}$
Fronteira de um corte: são os últimos eventos de um corte	e_i^{Ci} para i=1 a N

Cortes consistentes e inconsistentes

- Um *corte* da execução de um sistema é um subconjunto de seu histórico global que é uma união de prefixos de históricos de processo:
- Um corte consistente
  - Inclui tanto o envio como a recepção da mensagem m.
- Um corte inconsistente
  - Porque em p2, ele inclui a recepção de mensagem m1, mas p1 não inclui o envio dessa mensagem



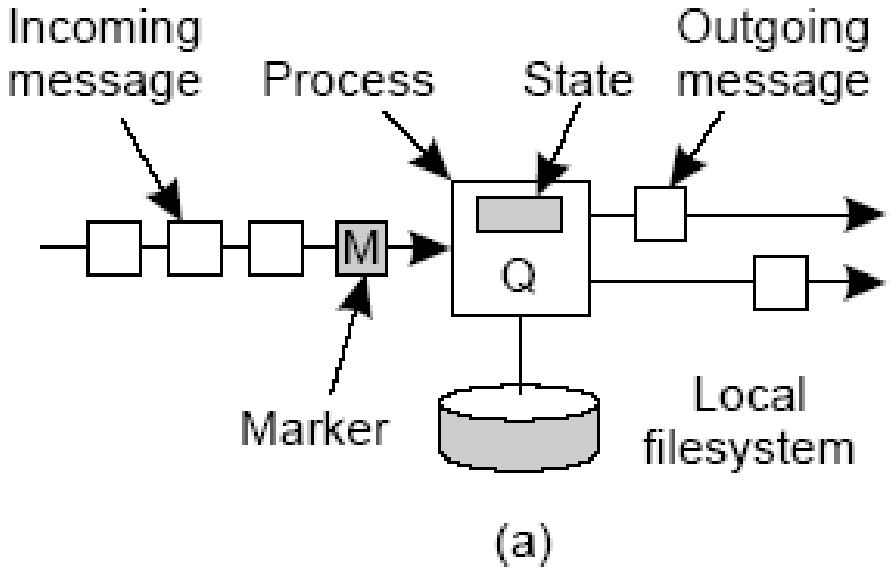
Um corte C é consistente se para cada evento que contém, ele também contém todos os eventos que aconteceram antes desse evento.

© Pearson Education 2012

Algoritmo do instantâneo (*snapshot*)

- Proposto por Chandy e Lamport (1985) para determinar os estados globais em SD's.
- O objetivo é gravar um conjunto de estados de processo e do canal (um "instantâneo" para um conjunto de processos) tal que o estado global gravado é consistente.

a) Organização de um processo e canais para se obter um "instantâneo" do sistema distribuído



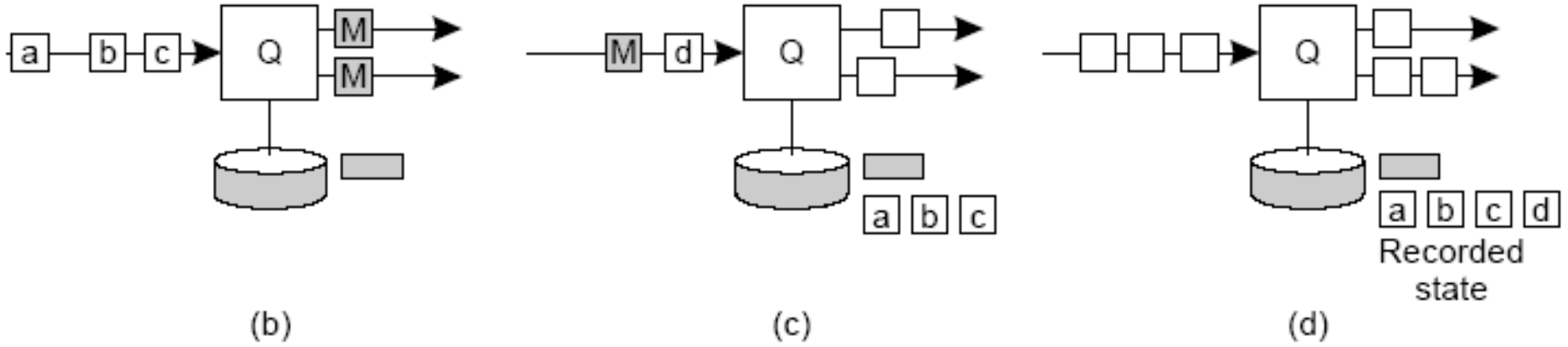
© Pearson Education 2012

Pressupostos

- Nem os canais, nem os processos falham; a comunicação é confiável, de modo que toda mensagem enviada é recebida intacta, exatamente uma vez;
- Os canais são unidirecionais e fornecem entrega de mensagem com ordenamento FIFO;
- O grafo de processos e canais é fortemente conectado (existe um caminho entre quaisquer dois processos);
- Qualquer processo pode iniciar um instantâneo global a qualquer momento;
- Enquanto o instantâneo ocorre, os processos podem continuar sua execução e enviar e receber mensagens normalmente.

Algoritmo do instantâneo de Chandy e Lamport

a) Processo Q recebe um marcador pela primeira vez e registra seu estado local
b) Processo Q retransmite a mensagem de marcador através de seus canais de saída
c) Q registra todas as mensagens que chegam
d) *Q* recebe um marcador para seu canal entrante e pára de registrar o estado desse canal



© Pearson Education 2012

Algoritmo do instantâneo de Chandy e Lamport

Regra de recepção de marcador do processo p_i

Na recepção por parte de p_i de uma mensagem de *marcador* pelo canal c:

if (p_i ainda não tiver gravado seu estado) ele

grava seu estado de processo atual

grava o estado de *c* como o conjunto vazio;

ativa a gravação de mensagens que chegam por outros canais de entrada;

else

p_i grava o estado de *c* como o conjunto de mensagens que recebeu por *c* desde que salvou seu estado.

end if

Regra de envio de marcador do processo p_i

Após p_i ter gravado seu estado, para cada canal de saída *c*:

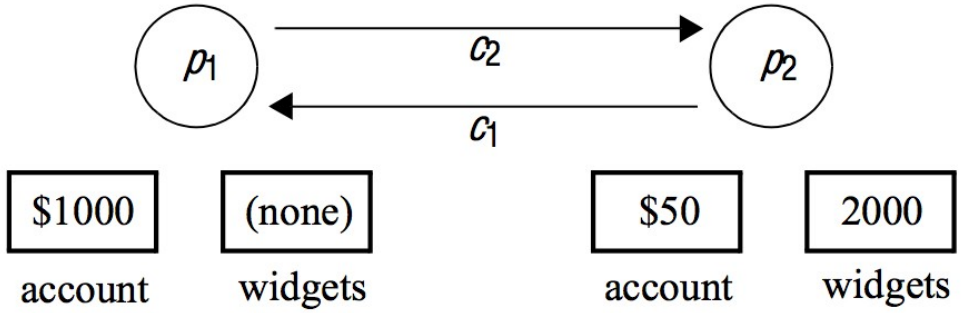
p_i envia uma mensagem de marcado por *c*

(antes de enviar qualquer outra mensagem por *c*).

© Pearson Education 2012

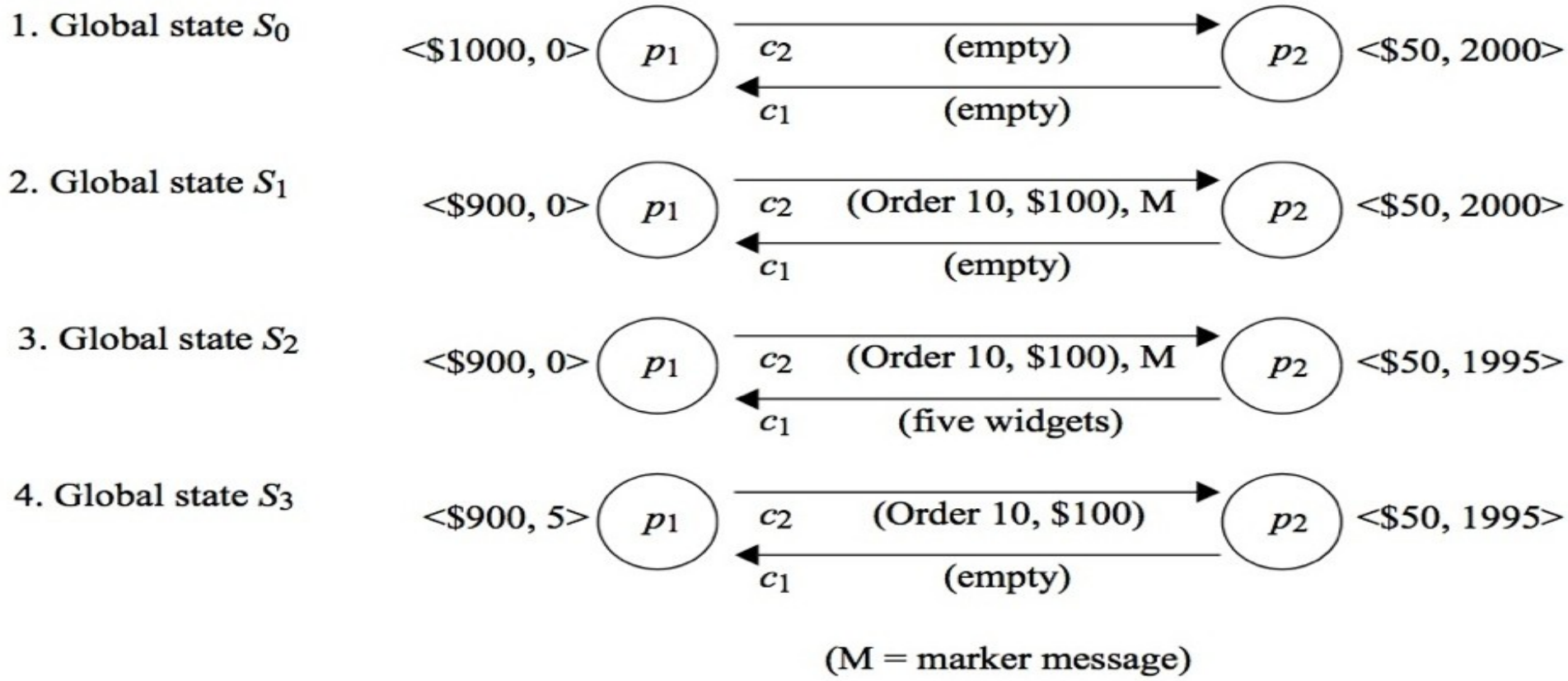
Exemplo

- Dois processos p1 e p2 conectados por dois canais unidirecionais c1 e c2;
- Os dois processos negociam “coisas”;
- O processo p_1 envia uma requisição de “uma coisa” para p_2 por c2;
- O valor de cada coisa é $10.
- Algum tempo depois, o processo p2 envia “coisas” para o p1 pelo canal c1;
- O processo p2 já recebeu uma requisição de cinco ”coisas”;



© Pearson Education 2012

Exemplo



© Pearson Education 2012

Agenda

- 11.2 - Relógios, eventos e estados de processos
- 11.3 - Sincronizando relógios físicos
- 11.4 - Tempo lógico e relógios lógicos
- 11.5 – Estados Globais
- 11.6 – Depuração distribuídas

Leituras recomendadas

- http://olamundo-java.blogspot.com/2019/12/debug-remoto-no-tomcat-com-o-eclipse.html
- https://www.youtube.com/watch?v=DvZ6zvz-8fI
- https://www.devmedia.com.br/depurando-aplicacoes-em-servidores-remotos-com-o-eclipse/37169
- https://dzone.com/articles/how-debug-remote-java-applicat
- https://www.researchgate.net/publication/221107983_Debugging_distributed_object_applications_with_the_Eclipse_platform

Exercícios