



Sincronização e Concorrência

Tópicos da Aula

❑ Sincronização

- ❖ sincronização interna
- ❖ sincronização externa
- ❖ métodos de sincronização
 - Cristian
 - Berkeley

❑ Controle de Concorrência

- ❖ transações
- ❖ serialização
- ❖ granularidade
- ❖ abordagens
 - *locking*
 - *timestamps*
 - otimista



Sincronização

Temporização e Coordenação de Eventos

Sincronização

❑ Devido a diferenças como

- ❖ Capacidade de processamento / carga das máquinas envolvidas em uma aplicação distribuída

- ❖ Banda / Tráfego em trechos / direções de rede

eventos distribuídos podem ser observados fora da ordem esperada...

É necessário algum mecanismo de sincronização

Sincronização em Distribuição

- ☐ Manutenção da consistência de dados distribuídos (uso de *timestamps*)
- ☐ Checagem da autenticidade de um pedido enviado a um servidor (o protocolo de autenticação Kerberos depende de relógios sincronizados)
- ☐ Eliminação do processamento de atualizações duplicadas
- ☐ Multimídia
- ☐ entre outros....

Tipos de Sincronização

❑ Mídias discretas

- ❖ Ordenação

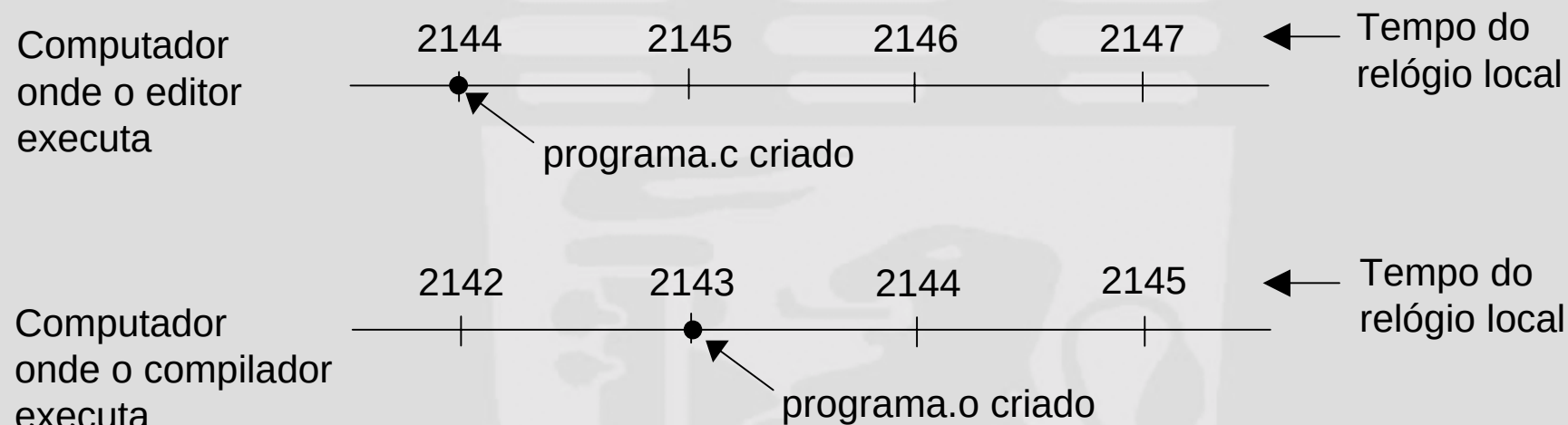
❑ Mídias contínuas

- ❖ *Intrastream*

- ❖ *Interstream*

O papel de *buffering*...

Sincronização de Relógios



O programa foi editado (realmente – em tempo real) antes de compilado, mas os relógios das máquinas dizem o contrário...

- ❑ Quando cada máquina tem seu próprio relógio, um evento que ocorreu (realmente) depois de outro pode, no entanto, receber uma marcação de tempo anterior

Medições de Tempo

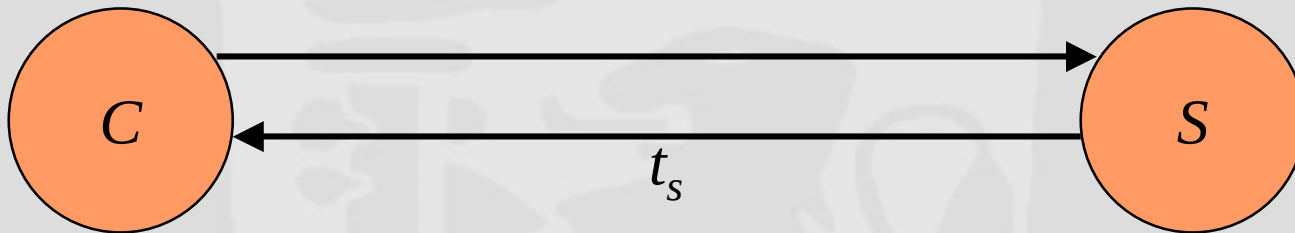
- ❑ Pode-se querer saber a que hora do dia um evento em particular ocorreu em um computador - para contabilidade, por ex.
 - ➔ necessário sincronizar o seu relógio com uma fonte externa de tempo - **sincronização externa**
- Pode-se querer medir o intervalo entre dois eventos que ocorrem em computadores diferentes
 - ➔ apelando para os seus relógios locais - **sincronização interna** (no sistema)

Sincronização de Relógios

- ☐ Em diferentes computadores, depende, na maioria dos casos, de comunicação em rede
- ☐ O problema é que o envio de mensagens leva tempos imprevisíveis

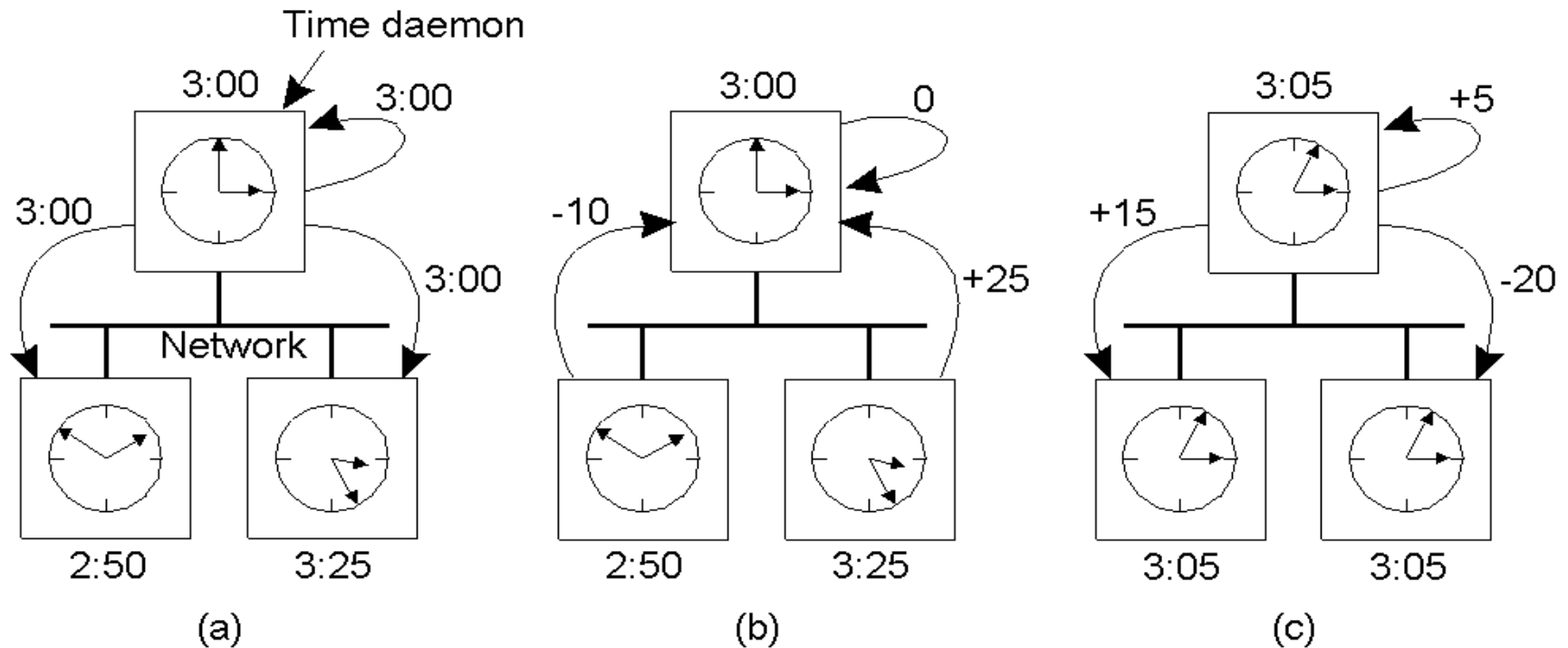
Um Método para a Sincronização de Relógios

- ❑ [F. Cristian]: servidor central de tempo, podendo ter um receptor de sinal para sincronizar com o UTC



- ❑ $t_c = t_s + T_{transmissão}$
- ❑ $T_{transmissão}$ pode variar
- ❑ Para se prevenir de falhas no servidor, usa um grupo de servidores sincronizados

Algoritmo de Berkeley



- O daemon de tempo requisita os valores de relógio a todas as outras máquinas
- As máquinas respondem
- O daemon de tempo informa a todos como ajustar seus relógios

Distributed Systems: Principles and Paradigms

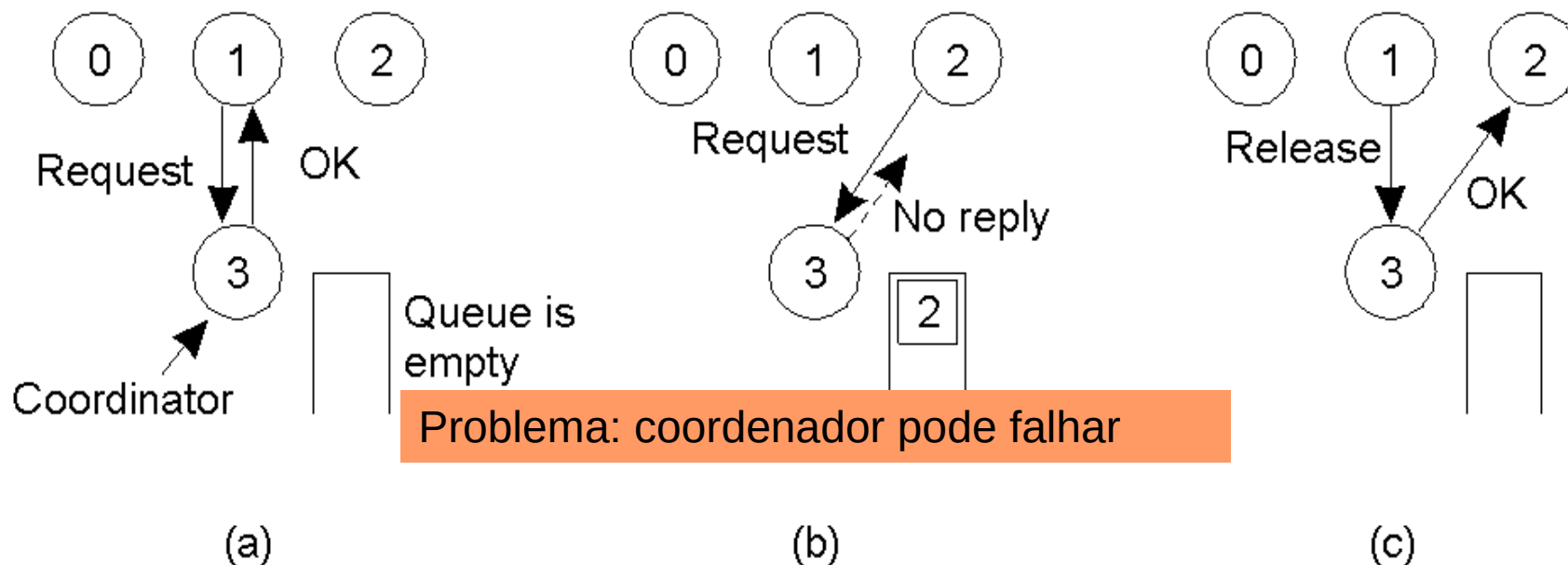
© Tanenbaum and van Steen 2002

© 2003 Carlos A. G. Ferraz



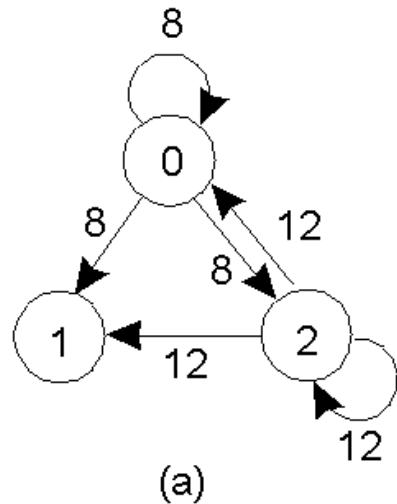
Controle de Concorrência

Exclusão Mútua: Um Algoritmo Centralizado

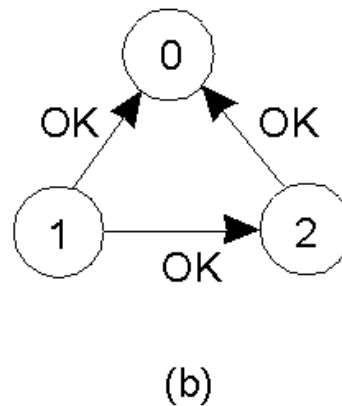


- Processo 1 pede ao coordenador permissão para entrar em uma região crítica. Permissão é dada
- Processo 2 pede permissão para entrar na mesma região crítica. O coordenador não responde, enfileirando o pedido de 2
- Quando o processo 1 sai da região crítica, avisa o coordenador, que então responde ao processo 2

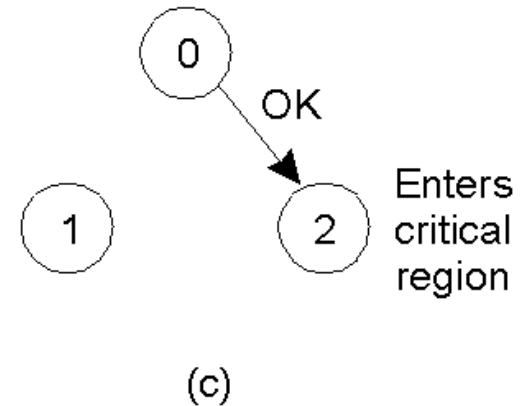
Um Algoritmo Distribuído



Enters
critical
region



Problema: falha em
qualquer processo



- a) Dois processos querem entrar na mesma região crítica no mesmo momento
- b) Processo 0 possui o menor *timestamp* – por isso entra
- c) Quando o processo 0 termina, envia um OK, e assim 2 pode então entrar na região crítica

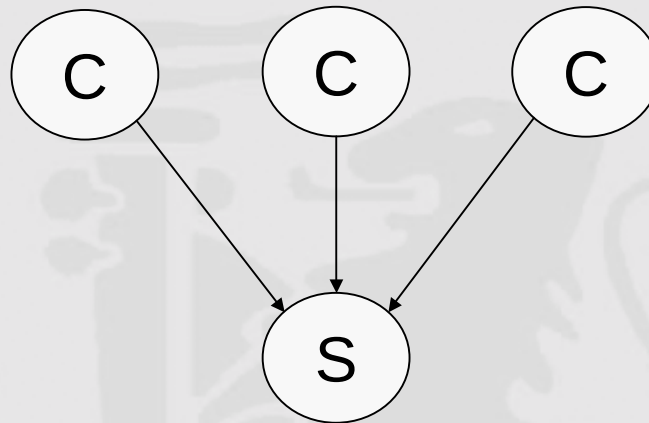
Distributed Systems: Principles and Paradigms

© Tanenbaum and van Steen 2002

© 2003 Carlos A. G. Ferraz

Transações

- ❑ Em geral, servidores executam operações para vários clientes cujas requisições podem ser intercaladas



- ❑ Clientes podem especificar transações atômicas
- ❑ O efeito de transações sobre dados compartilhados deve ser sequencialmente equivalente (equivalente a uma execução sequencial/ordenada)

O Modelo de Transação (1)

❑ Primitivas

Primitiva	Descrição
BEGIN_TRANSACTION	Início da transação
END_TRANSACTION	Fim da transação e tentativa de <i>commit</i>
ABORT_TRANSACTION	Aborta a transação e recupera valores anteriores
READ	Leitura de dados de um arquivo, tabela, ...
WRITE	Escrita de dados em um arquivo, tabela, ...

O Modelo de Transação (2)

```
BEGIN_TRANSACTION  
  reserve REC -> RIO;  
  reserve RIO -> POA;  
  reserve POA -> BSB;  
END_TRANSACTION
```

(a)

```
BEGIN_TRANSACTION  
  reserve REC -> RIO;  
  reserve RIO -> POA;  
  reserve POA -> BSB full =>  
ABORT_TRANSACTION
```

(b)

- a) Transação para a reserva de 3 vôos *commits*
- b) Transação aborta quando o terceiro vôo não está disponível

Serialização

❑ Uso de *locks* exclusivos

Transação T:

Bank\$Withdraw(A,4)
Bank\$Deposit(B,4)

Transação U:

Bank\$Withdraw(C,3)
Bank\$Deposit(B,3)

BeginTransaction
balance := A.Read()lock A
A.Write(balance-4)

balance := B.Read()lock B

B.Write(balance+4)
EndTransaction

•
unlock A,B

BeginTransaction
balance := C.Read()lock C
C.Write(balance-3)

balance := B.Read()espera

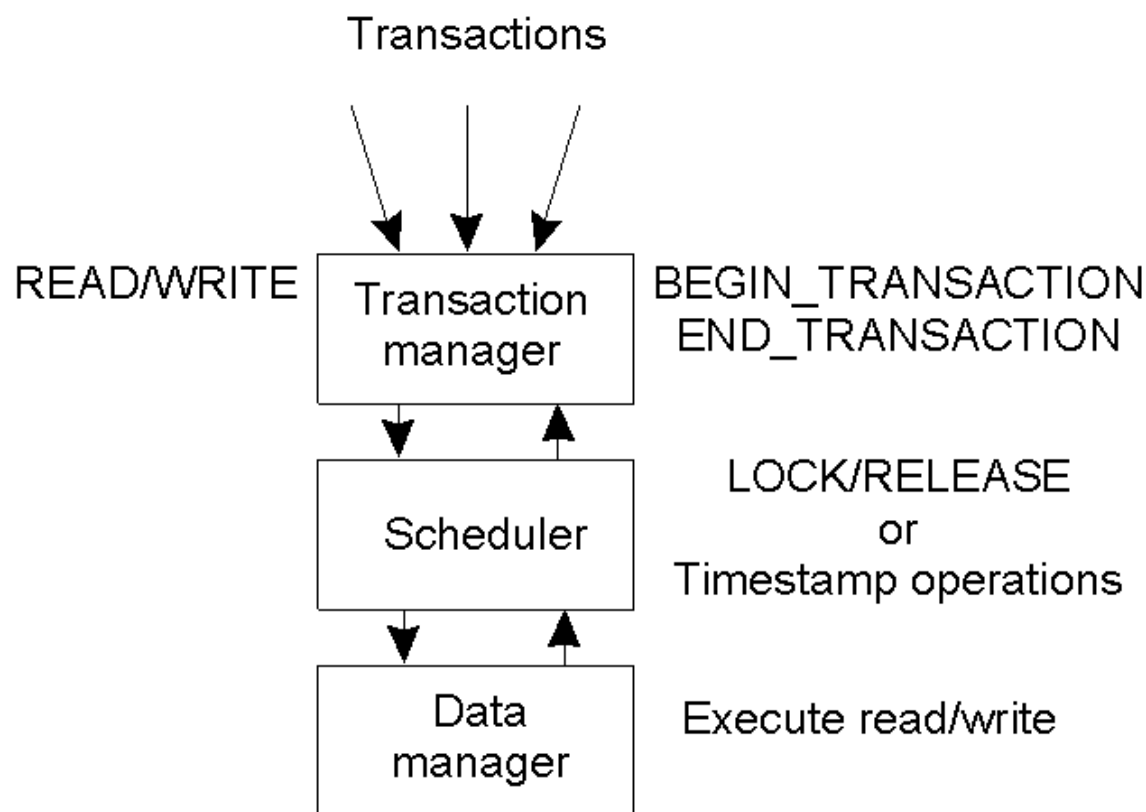
•

•
B.Write(balance+3)
EndTransaction

por T
(unlock B)

unlock B,C

Controle de Concorrência



Organização geral de gerentes para o tratamento de transações

Distributed Systems: Principles and Paradigms

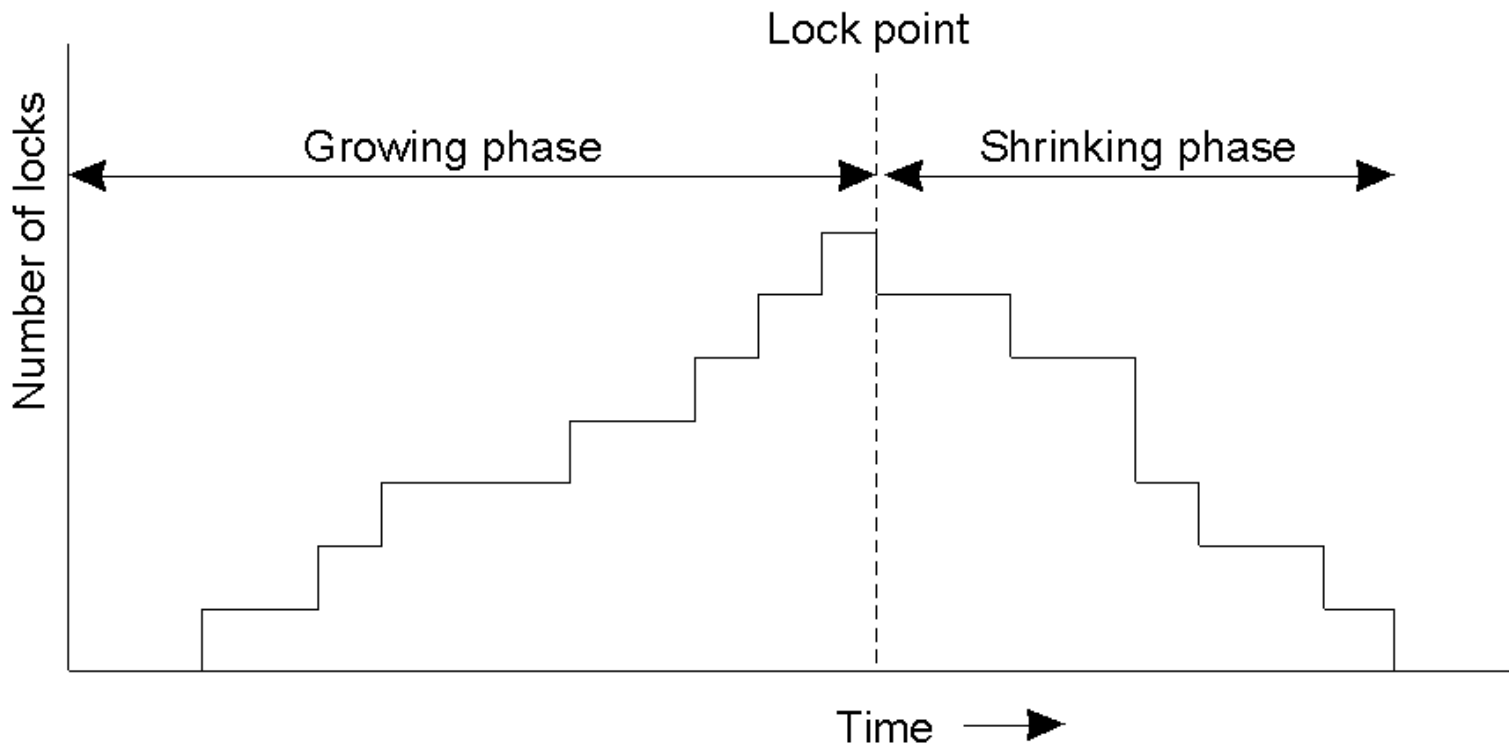
© Tanenbaum and van Steen 2002

© 2003 Carlos A. G. Ferraz

Two-Phase Locking

- ❑ Cada transação tem uma fase de requisição de *locks (growing phase)*
- ❑ Uma segunda fase libera *locks (shrinking phase)*
- ❑ Transações podem abortar: é necessário execução estrita para prevenir leituras erradas ou escritas prematuras
 - ❖ quaisquer locks conseguidos durante uma transação são sustentados até que a transação termine ou aborte → ***strict two-phase locking***

Two-Phase Locking (2)



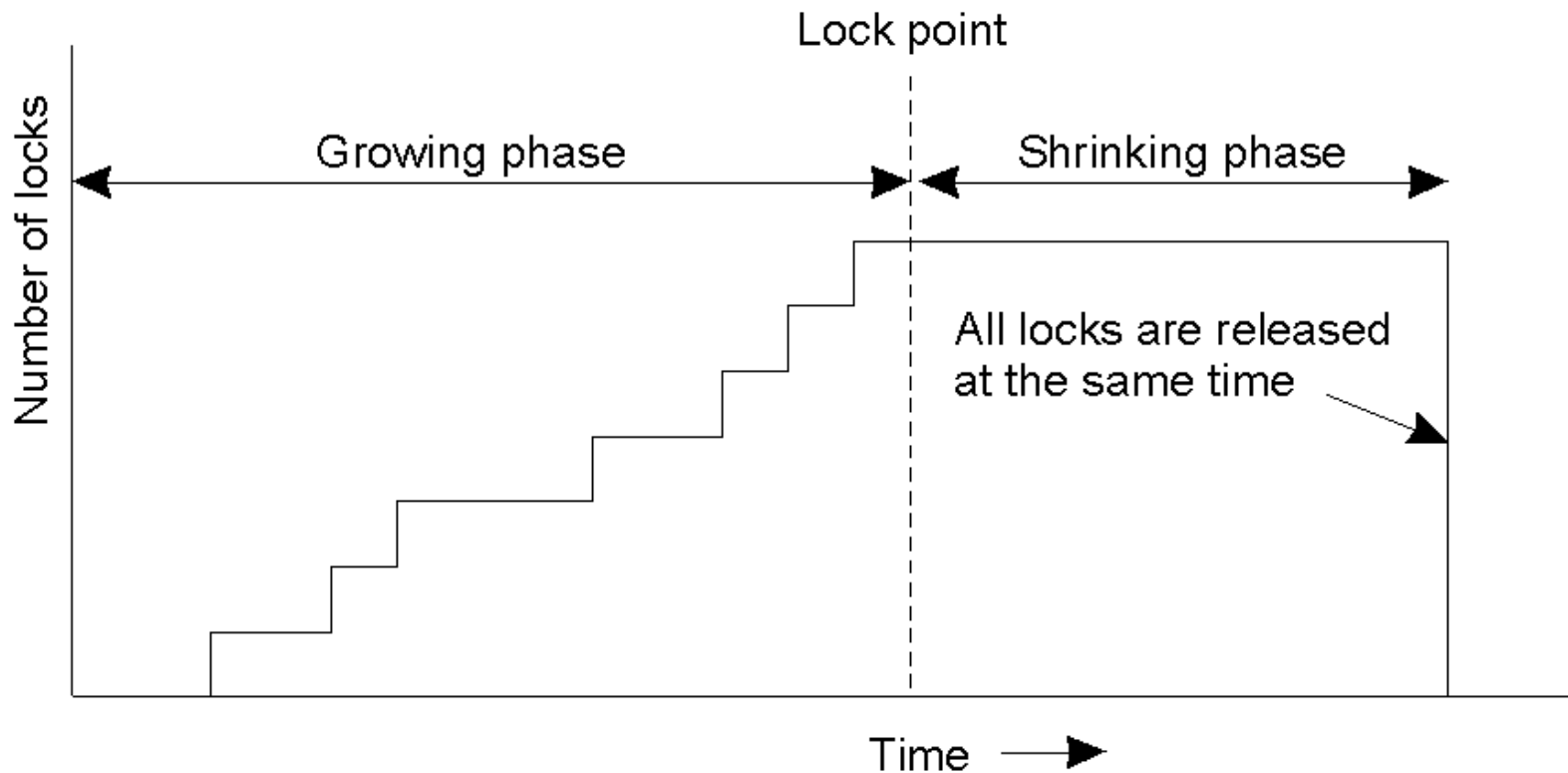
Two-phase locking

Distributed Systems: Principles and Paradigms

© Tanenbaum and van Steen 2002

© 2003 Carlos A. G. Ferraz

Two-Phase Locking (3)



Strict two-phase locking

Granularidade do Controle de Concorrência

- ❑ Uma transação típica acessa apenas uma pequena parte do grande número de itens de dados que um servidor contém e é improvável confundir com outras transações
- ❑ A granularidade do controle de concorrência pode afetar no desempenho de um servidor
- ❑ A porção de itens de dados para os quais o acesso deve ser serializado deve ser a menor possível, ou seja, apenas a parte envolvida em cada operação requisitada por transações
- ❑ Ex: cada operação bancária afeta um ou mais saldos de contas
 - ❖ Depósito e Retirada afetam apenas uma conta
 - ❖ TotalAgência afeta todas

Abordagens para Controle de Concorrência

1. A maioria dos sistemas usam **locking**
 1. O uso de *locks* pode levar a *deadlock*
 2. Ordenamento baseado em **timestamps** (a seguir)
- ☐ 1 e 2 são abordagens **pessimistas**

- ☐ Esquemas **otimistas** permitem que uma transação prossiga até que termine, para então o servidor fazer uma checagem para descobrir se houve conflito
- ❖ se sim, a transação terá que ser refeita

Timestamps

☐ Write

Seja T_j o timestamp da escrita de um item de dado, $T_j \geq$ máximo *timestamp* de leitura do item de dado - **escrita somente após todas as leituras em andamento**

☐ Write

$T_j >$ máximo *timestamp* de escrita do item de dado (*committed* - **somente após a última escrita**)

☐ Read

Seja T_j o timestamp da leitura de um item de dado, não se deve **ler** um item de dado que foi **escrito** por alguma T_i , onde $T_i > T_j$, ou seja, não se pode ler se $T_j <$ *timestamp* de escrita da versão *committed* do item de dado - **leitura somente após escrita em andamento**

Abordagens Pessimistas

- ❑ Em ***two-phase locking*** e **ordenação baseada em *timestamps*** o servidor detecta conflitos entre transações a cada acesso a item de dado
- ❑ **Ordenação baseada em *timestamps*** é melhor para transações de leitura
- ❑ ***Two-phase locking*** é melhor quando as operações são predominantemente de atualizações
- ❑ Quando conflitos são detectados:
 - ❖ **ordenação baseada em *timestamps*** aborta imediatamente
 - ❖ ***two-phase locking*** faz a transação esperar - podendo abortar, no entanto, após um *timeout* para evitar *deadlock*

Abordagem Otimista

- ☐ Relativamente eficiente quando há poucos conflitos
- ☐ Bastante trabalho pode ter que ser repetido quando uma transação é abortada

VIRTUS IMPAVIDA