Sistemas Distribuídos
Aula 21

Aula passada

- Estado distribuído
- *2-Phase Commit*
- Falhas
- Deadlocks
- *3-Phase Commit*

Aula de hoje

- Replicação
- Conflitos
- Modelos de consistência
- Modelos de consistência no cliente

Replicação de Dados

- Por que replicar dados em um sistema?

- **Desempenho**: permite reduzir tempo de resposta
  - menos carga, acesso local
- **Confiabilidade**: permite recuperar de falhas
  - redundância

Replicação é fundamental!

- Utilizado amplamente em sistemas reais
- Ex. CDNs, DNS (*root servers*), Gmail, websites (Amazon), etc

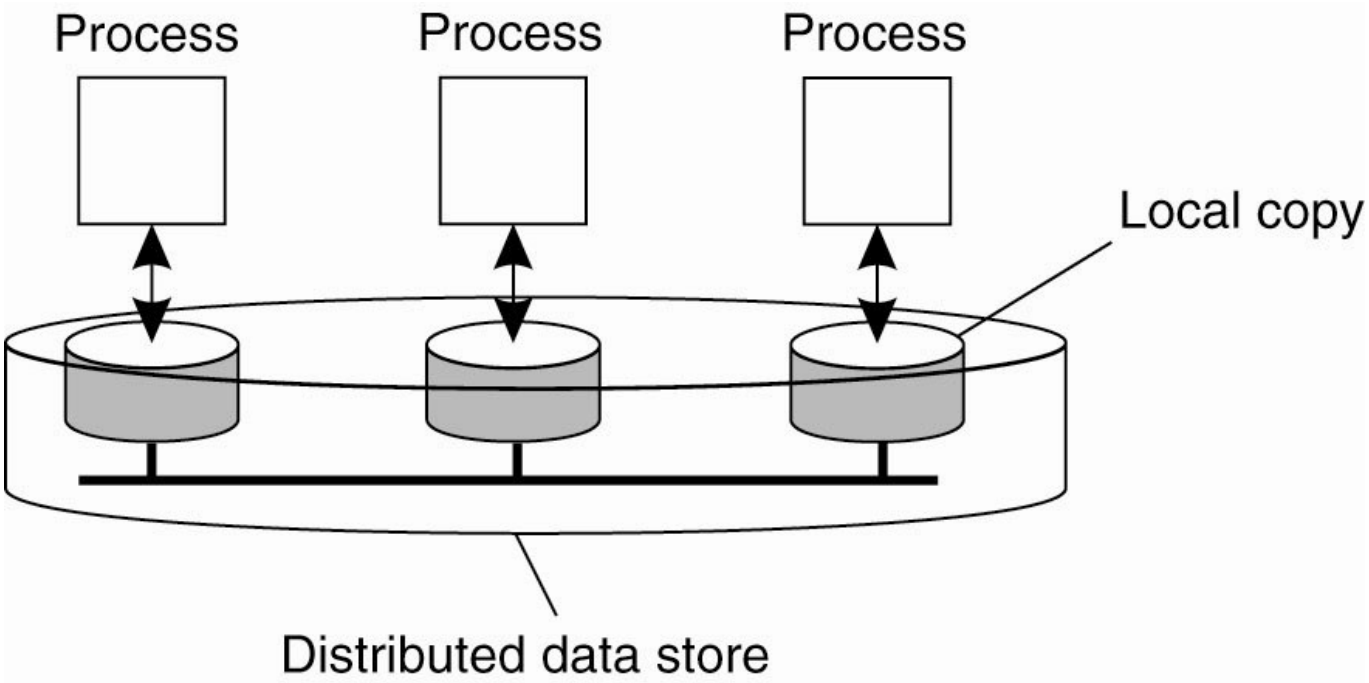
Problemas ao Replicar

- Replicação tem custos
  - ex. custo do hardware (armazenamento)
- Outro problema surge ao replicar dados?

Consistência!

- Diferentes réplicas precisam ser atualizadas
- Conflito entre operações de leitura e escrita
- Precisamos garantir algum tipo de consitência

Modelo Clássico de Replicação



- Diferentes processos possuem réplica local
- Leitura/escrita feita nas réplica locais
- Sistema de armazenamento implementa modelo de consistência (contrato)

Conflitos ao Replicar

- Operações de leitura e escrita do mesmo dado podem ocorrer em dois processos
  - conflito *read-write* e conflito *write-write*
  - geram condição de corrida
- Como resolver estes conflitos?
- **Ideia 0**: garantir ordenação total dos eventos no sistema distribuído
  - *totally ordered multicast* (já vimos)
- Por que isto pode não ser adequado?
  - alto custo compromete escalabilidade

Consistência

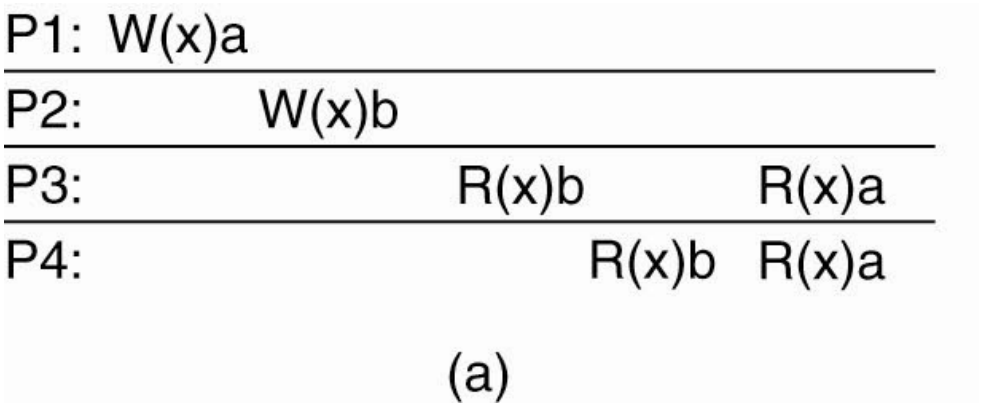
- **Idea**: oferecer um *modelo de consistência* de dados
  - mais fraco que uma ordenação total dos eventos
- Modelo de consistência é um contrato entre aplicação e o armazenamento dos dados
  - define alguma ordem para os conflitos
- Em geral, consistência mais estrita → mais ordem (previsibilidade) → mais complexo
- Aplicação tem que ter em mente o modelo
- **Modelo 0**: consistência do sistema, ordem definida pela execução, sem restrições
  - baixa complexidade, difícil de aplicação usar

Consistência Sequencial

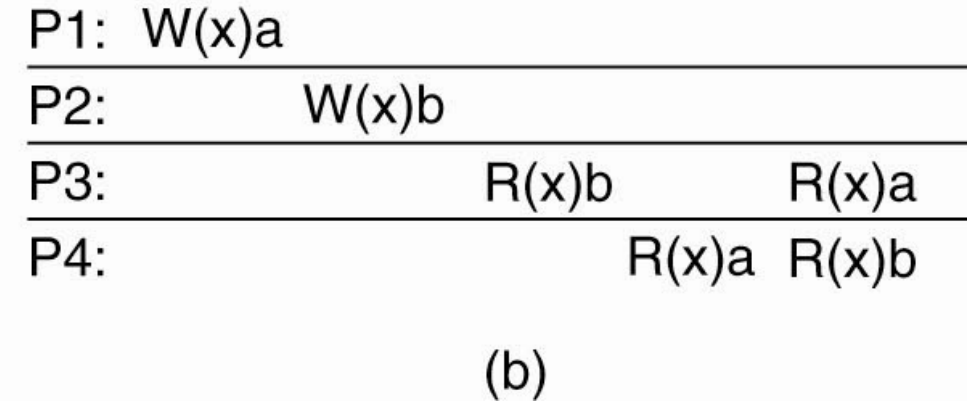
- **Ideia**: colocar em sequência as operações de leitura/escrita em cada processo
- Execução válida de uma sequência de operações que respeita a ordem intra-processo
  - modelo garante apenas isto
- Aplicação tem que estar pronta para as várias possíveis execuções

Exemplo

- W(x)a : escreve no endereço x o valor a
- R(y)b : leitura do endereço y retornou b
- 4 processos (cada um com sua réplica local)



Execução respeita consistência sequencial



Execução não respeita consistência sequencial

Exemplo 2

Process P1	Process P2	Process P3
x ← 1;	y ← 1;	z ← 1;
print(y, z);	print(x, z);	print(x, y);

- Três processos, três variáveis (inicialmente 0)
- O que pode e não pode ser impresso?
  - sem modelo de consitência, tudo!
- Com consistência sequencial?
- Ex: 111111 ?
  101011 ?
  000000 ?

Consistência Causal

- **Ideia**: colocar em sequência as operações que possam possuir causalidade
  - dentro do mesmo processo e intra-processo
  - write concorrentes ocorrem em qualquer ordem
- Todos processos veem operações causais na mesma ordem
- Processos diferentes podem ver writes concorrentes em ordem diferente
- Parecido com ordenação parcial induzida por relógio lógico

Exemplo

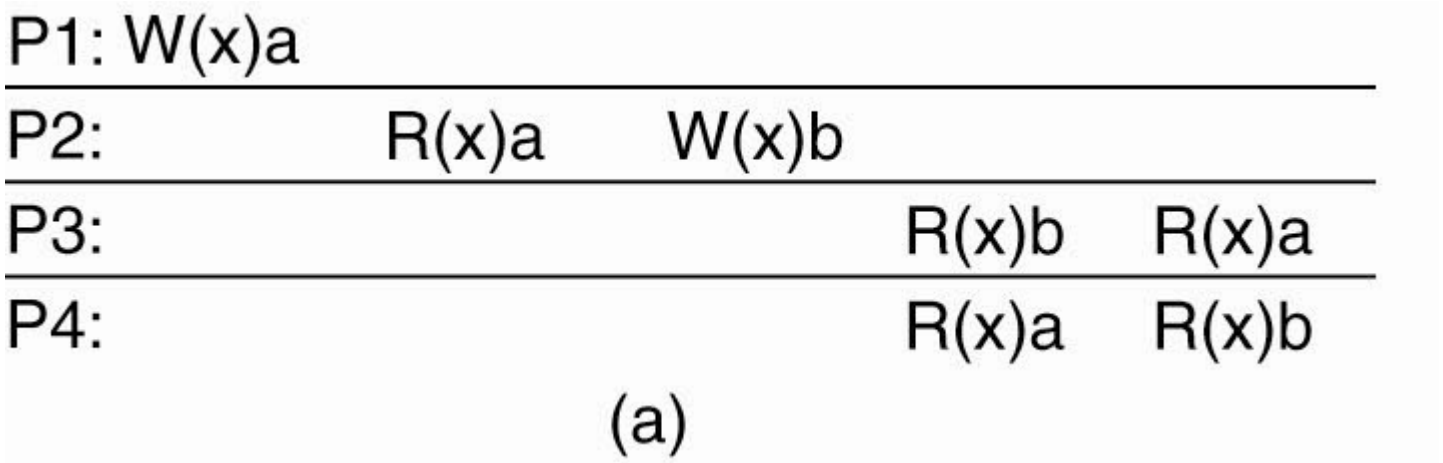
- W(x)a : escreve no endereço x o valor a
- R(y)b : leitura do endereço y retornou b
- 4 processos (cada um com sua réplica local)

P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

Respeita consistência causal

- P1 escreve em, x valor “a”,
- P2, P3, P4 leem x com valor “a”
  - evento ocorre depois

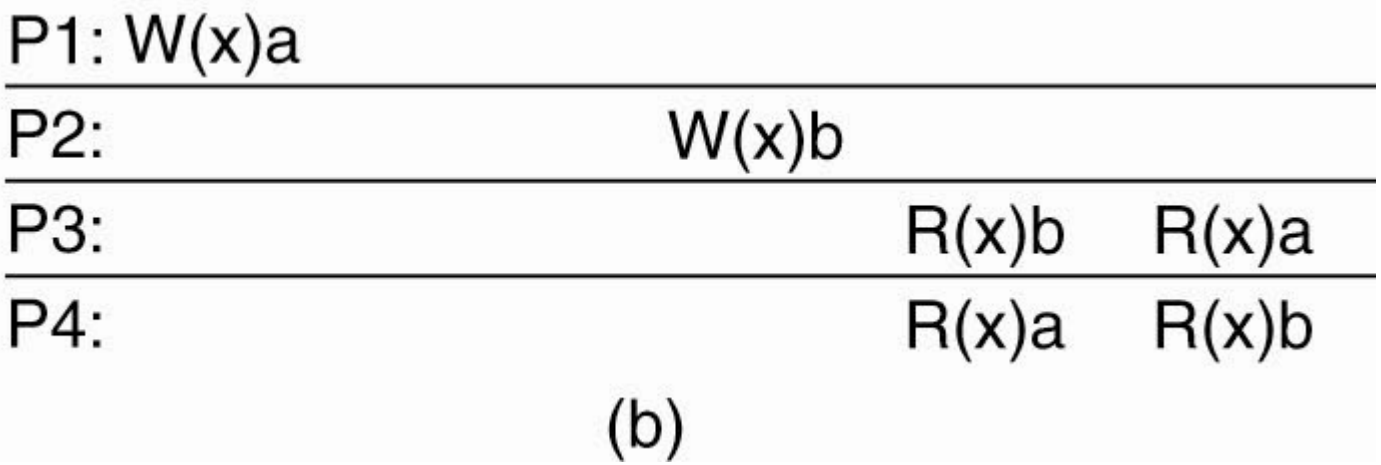
Exemplo 2



(a)

Não respeita consistência causal

Causalidade writes em x devem ser vistos na mesma ordem por todos!



(b)

Respeita consistência causal

Focando no Cliente

- Muitas aplicações possuem modelo restrito
  - muitos acesssos de leitura, possivelmente simultâneos
  - poucos acessos de escrita, possivelmente sequenciais
  - cliente define escopo dos dados
- Exemplos
  - webmail, agenda, Dropbox
- Consequência: reduzem conflitos read-write, eliminam conflitos write-write

Exemplo com Dropbox

- Usuário Dropbox usando dois computadores
  - cada computador possui cópia local (réplica)
- Trabalha em casa (C1) nos arquivos
- Vai para a UFRJ e trabalha (C2) nos arquivos

- **O que é de fato importante?**
- Arquivo F modificado em C1 esteja atualizado em C2 caso F seja acessado em C2
  - se F não for acessado em C2 usuário não sabe da “inconsistência”

Exemplo com GMail

- Usuário GMail acessa sua conta via aplicativo em smartphone
- Lê emails no Rio, pega um voo para Nova Iorque (NYC), acessa em NYC
  - Gmail tem servidores replicados
- **O que é de fato importante?**
- Caixa de entrada em NYC é ao menos tão atual quanto caixa de entrada no Rio
  - usuário não sabe que emails que não estavam no Rio também não estão em NYC!

Consistência eventual

Consistência Centrada no Cliente

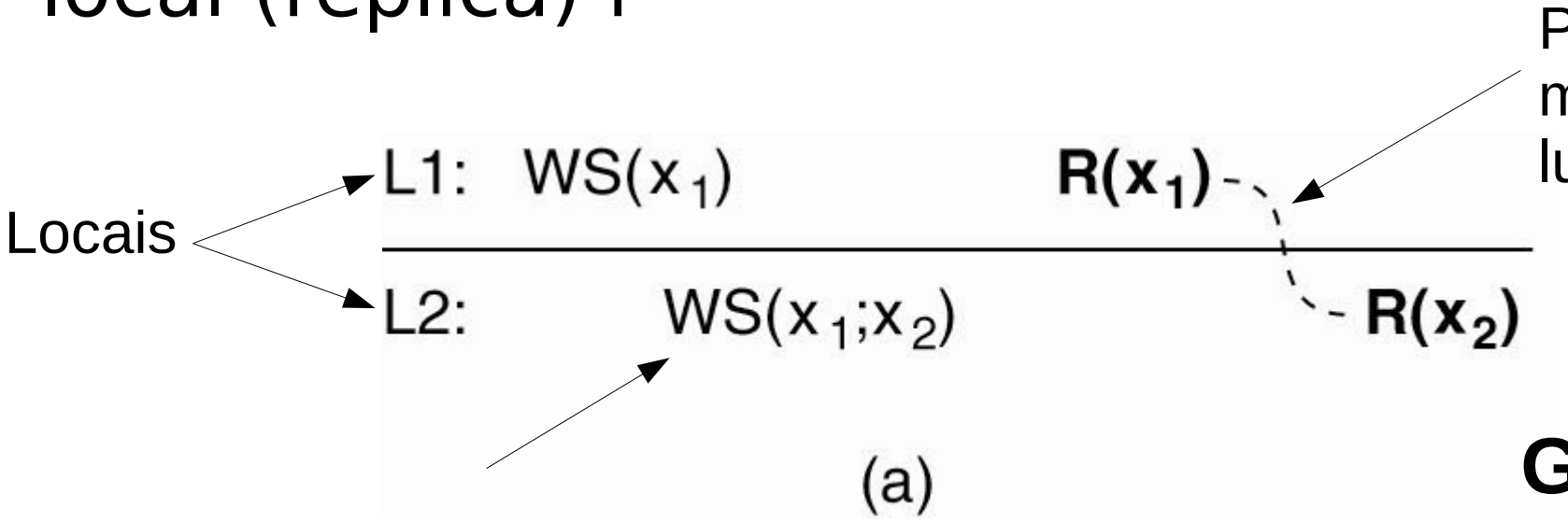
- Modelos anteriores são contrato da aplicação com o sistema (*system-wide*)
  - para todos os processos
- Mais difíceis de garantir, demandam maior complexidade do sistema
- Modelo de consistência centrado no cliente
  - mais leve, e mais fácil
- Cliente não é tão crítico assim
  - ex. não se encomoda que email já chegou em NYC mas não no Rio

Reads Monotônico

- Processo (cliente) faz operação de leitura/escrita em diferentes locais (réplica)
  - ex. browser do seu smartphone acessando o Gmail enquanto você viaja pela Europa
- Modelo de consitência “read monotônico”
  - se processo P ler valor em x, próximas leituras por P de x devem ter valores iguais ou mais recentes
- “mais recente” definido por causalidade (ou de forma explícita)
- Garante que cliente não volta atrás

Exemplo de Reads Monotonico

- Único processo P acessa dados mudando de local (cada local possui sua réplica)
- $WS(x_i)$: write (pelo sistema) no endereço x no local (réplica) i



Garante!

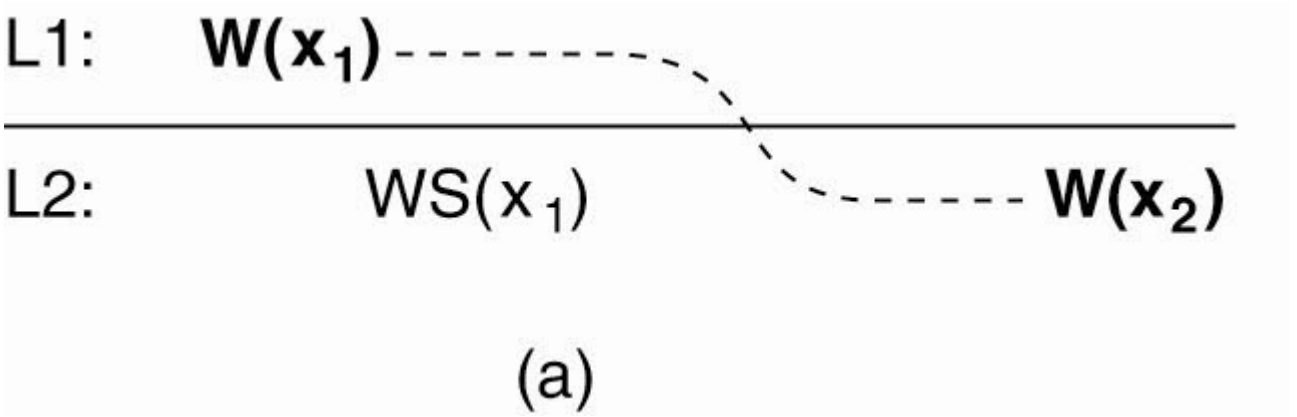
Write em x2 que ocorre depois de write em x1

Writes Monotônico

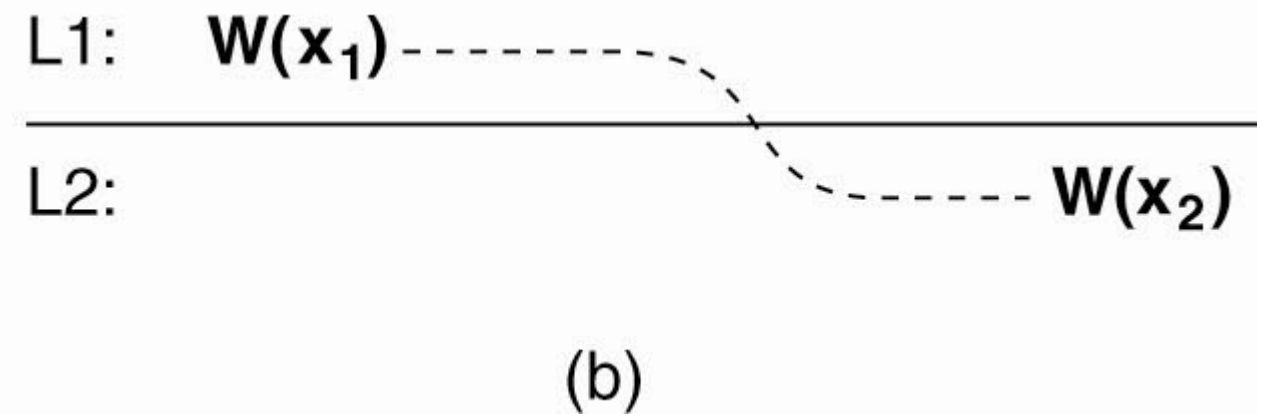
- Modelo de consitência “writes monotônico”
  - operação de escrita em x por P é executada antes de qualquer outra operação subsequente de escrita em x por P
- “antes” definido por causalidade, pois somente P escreve (único processo)
- Garante ordem FIFO de todas as escritas de um mesmo endereço em todas as réplicas

Exemplo de Write Monotonico

- Único processo P acessa dados mudando de local (cada local possui sua réplica)
- $WS(x_i)$: write (pelo usuário) no endereço x no local (réplica) i



Garante



Não Garante