ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

Aula 20: Consistência e Replicação (parte 1)

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

17/05/2024

Replicação

Por que replicar

Vamos assumir um modelo simples no qual fazemos uma cópia de uma parte específica de um sistema (ou seja, código e dados).

- Aumento da confiabilidade: se uma réplica não estiver operando corretamente, mudar para outra réplica.
- Desempenho: simplesmente espalhar as solicitações entre diferentes partes replicadas buscando balanceamento de carga, ou para garantir respostas rápidas considerando a proximidade.

Problema

Ter várias cópias significa que, quando qualquer cópia muda, essa mudança deve ser feita em todas as cópias: as réplicas precisam ser mantidas iguais, ou seja, mantidas consistentes.

Desempenho e escalabilidade

Principal desafio

Para manter as réplicas consistentes, geralmente precisamos garantir que todas as operações conflitantes sejam feitas na mesma ordem em todos os lugares: sincronização global.

Operações conflitantes:

- Conflito de leitura-escrita: uma operação de leitura e uma operação de escrita agem concorrentemente.
- Conflito de escrita-escrita: duas operações de escrita concorrentes.

Problema

Garantir uma ordem global nas operações conflitantes pode ser uma operação custosa, reduzindo a escalabilidade. Solução: enfraquecer os requisitos de consistência para que a sincronização global possa ser evitada.

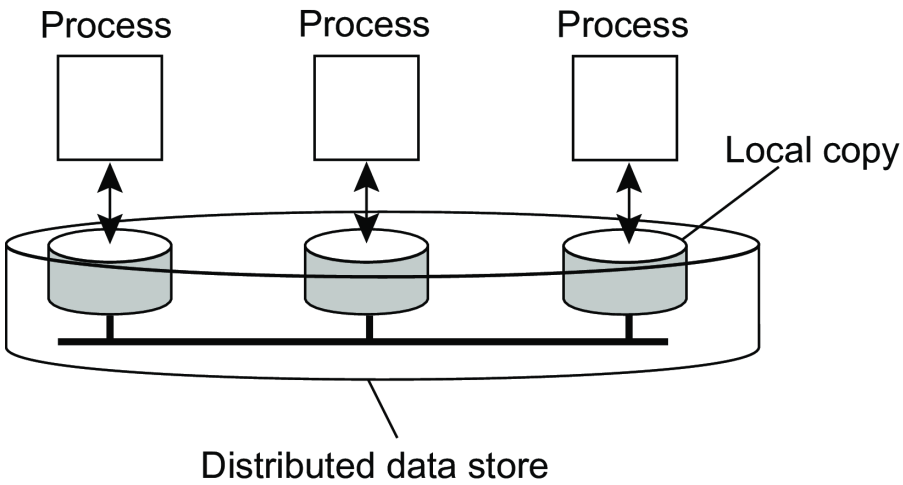
Modelos de consistência centrados em dados

Modelo de consistência

Um contrato entre um data store (distribuído) e os processos, no qual o data store especifica precisamente quais são os resultados das operações de leitura e escrita concorrentes.

Data store (armazém de dados)

Termo genérico para designar dados disponibilizados de forma distribuída, por exemplo, através de memória compartilhada, base dados compartilhada ou sistema de arquivos distribuído.



Modelos de consistência centrados em dados

Modelo de consistência

- Em geral, ao ler um dado, o processo espera obter o valor mais recente (ou seja, após efetuar a última operação de escrita)
- Necessidade de estabelecer uma ordem de operações de escrita
- Em geral, os modelos definem restrições de como as leituras podem ser feitas
- Modelos com mais restrições são mais fáceis de usar (pelo desenvolvedor do sistema), porém não apresentam o melhor desempenho

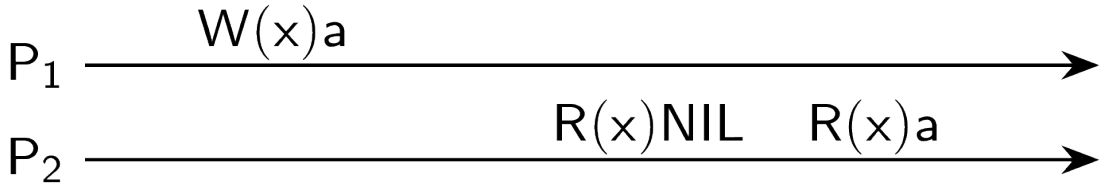
Algumas notações

Operações de leitura e escrita

- $W_i(x)a$: Processo P_i escreve o valor a em x
- $R_i(x)b$: Processo P_i lê o valor b de x
- Todos os itens de dados inicialmente têm valor *NIL*

Comportamento possível

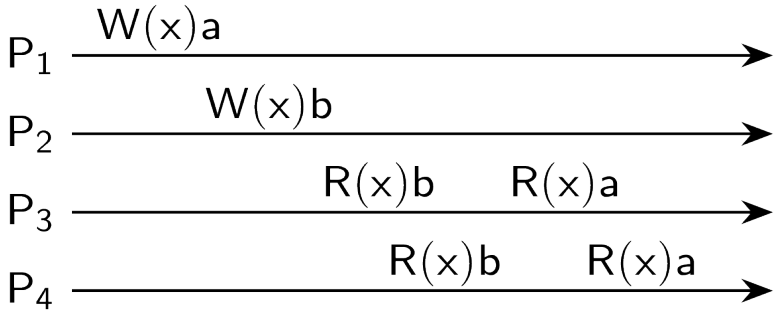
Omitimos o índice quando for possível e desenhamos de acordo com o tempo (eixo x):



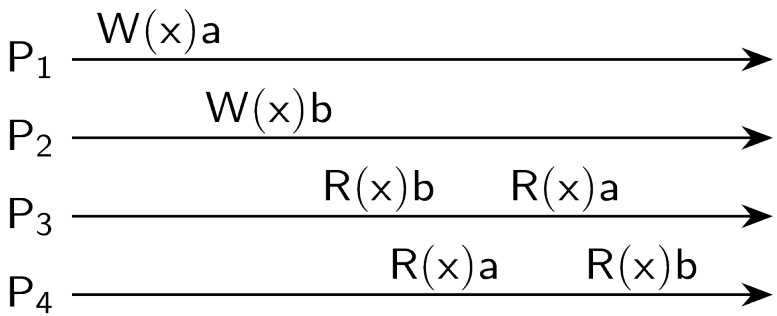
Consistência sequencial

Definição

O resultado final da execução em qualquer processo é o equivalente a uma dada ordem de execução das operações de todos os processos executadas em alguma ordem sequencial, sendo que as operações de cada processo individual aparecem na ordem especificada por seu programa.



Um data store sequencialmente consistente



Um data store que não é sequencialmente consistente

Exemplo

Três processos concorrentes (valores iniciais: 0)

Processo P_1	Processo P_2	Processo P_3
`x ← 1;`	`y ← 1;`	`z ← 1;`
`print(y,z);`	`print(x,z);`	`print(x,y);`

Exemplo

Três processos concorrentes (valores iniciais: 0)

Processo P_1	Processo P_2	Processo P_3
`x ← 1;`	`y ← 1;`	`z ← 1;`
`print(y,z);`	`print(x,z);`	`print(x,y);`

Exemplo de sequências de execução (todas **válidas**!)

Execution 1

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

Prints: 001011
Signature: 00 10 11

(a)

Execution 2

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

Prints: 101011
Signature: 10 10 11

(b)

Execution 3

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

Prints: 010111
Signature: 11 01 01

(c)

Execution 4

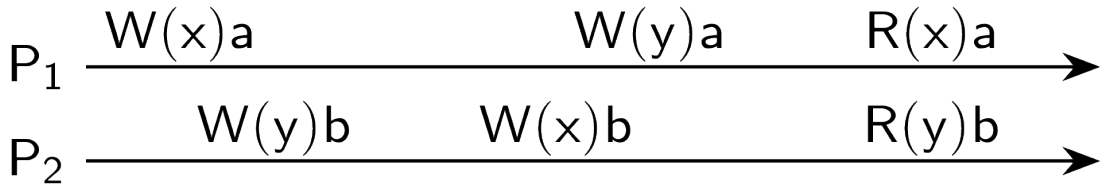
```
P2: y ← 1;
P1: x ← 1;
P3: z ← 1;
P2: print(x,z);
P1: print(y,z);
P3: print(x,y);
```

Prints: 111111
Signature: 11 11 11

(d)

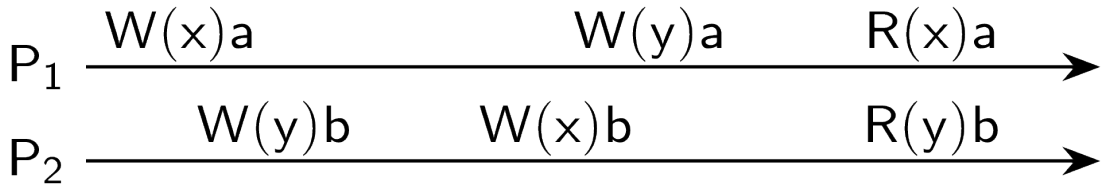
Cenário não óbvio

Aparentemente ok



Cenário não óbvio

Aparentemente ok

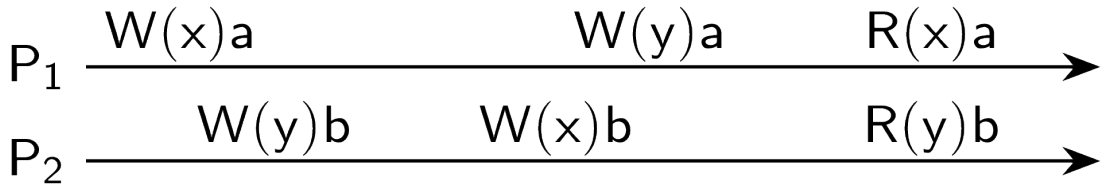


Mas não é! (não esqueça que P_1 e P_2 agem concorrentemente)

Possível ordenação de operações	**Resultado**	
$W_1(x)a$; $W_1(y)a$; $W_2(y)b$; $W_2(x)b$	$R_1(x)b$	$R_2(y)b$
$W_1(x)a$; $W_2(y)b$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a$; $W_2(y)b$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_2(x)b$; $W_1(x)a$; $W_1(y)a$	$R_1(x)a$	$R_2(y)a$

Cenário não óbvio

Aparentemente ok



Mas não é! (não esqueça que P_1 e P_2 agem concorrentemente)

Possível ordenação de operações	**Resultado**	
$W_1(x)a$; $W_1(y)a$; $W_2(y)b$; $W_2(x)b$	$R_1(x)b$	$R_2(y)b$
$W_1(x)a$; $W_2(y)b$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a$; $W_2(y)b$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_2(x)b$; $W_1(x)a$; $W_1(y)a$	$R_1(x)a$	$R_2(y)a$

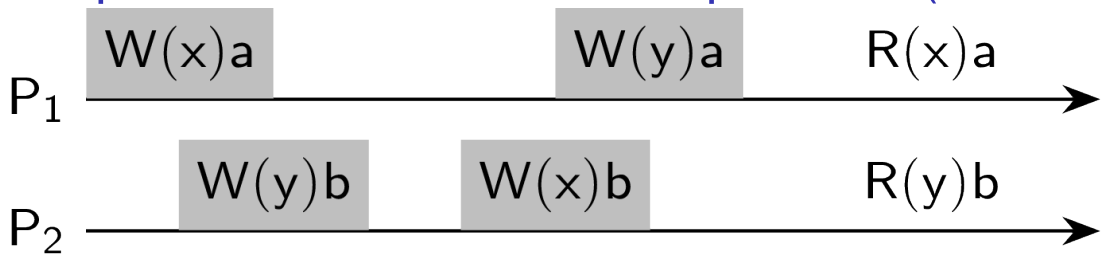
Operações do diagrama não são serializáveis

Adicionando mais restrições

Linearizabilidade

Cada operação deve parecer ter efeito instantâneo em algum momento entre seu início e conclusão.

Operações completam dentro de um tempo dado (área sombreada)



Redução das combinações possíveis

Possível ordenação de operações	**Resultado**	
$W_1(x)a$; $W_2(y)b$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a$; $W_2(y)b$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_1(y)a$; $W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b$; $W_1(x)a$; $W_2(x)b$; $W_1(y)a$	$R_1(x)b$	$R_2(y)a$

Consistência causal

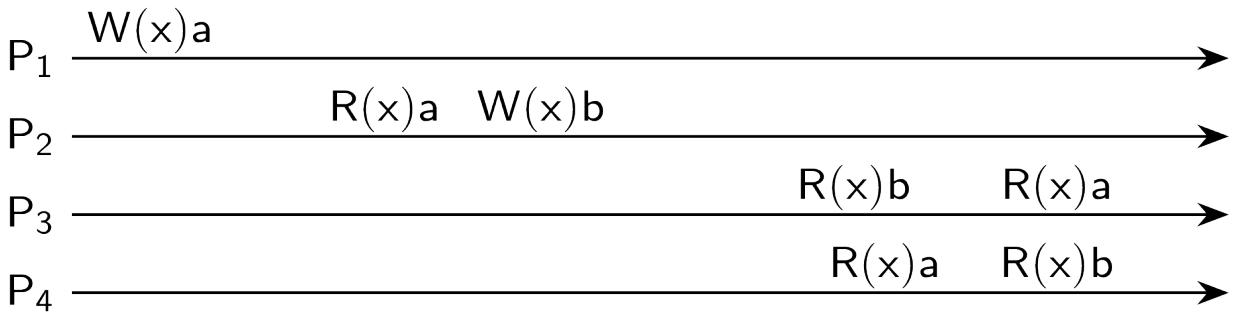
Causalidade

- Um processo P1 escreve no dado X, e, na sequência, o processo P2 lê X e escreve Y $=>$ Y potencialmente depende de X
- Processo P1 escreve um dado X1 e processo P2 escreve um dado X2 de forma independente $=>$ operações concorrentes

Definição de consistência causal

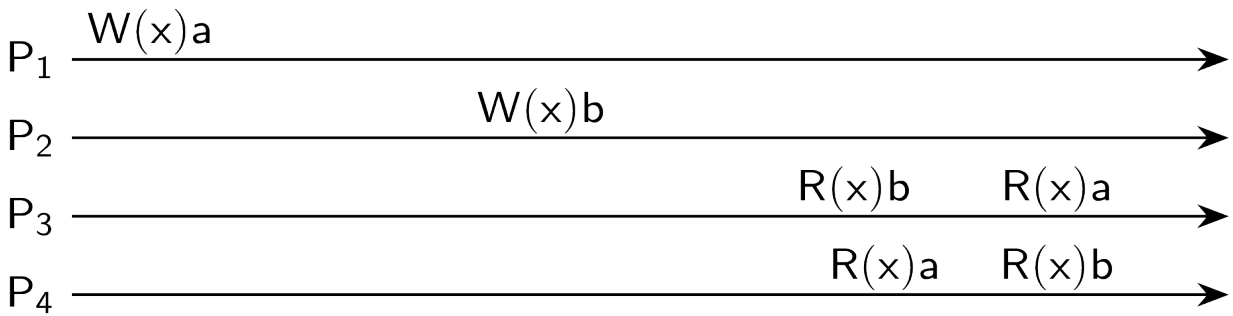
Escritas que têm potencial de ser causalmente relacionadas devem ser vistas por todos os processos na mesma ordem. Escritas concorrentes podem ser vistas em uma ordem diferente por processos diferentes.

Exemplos



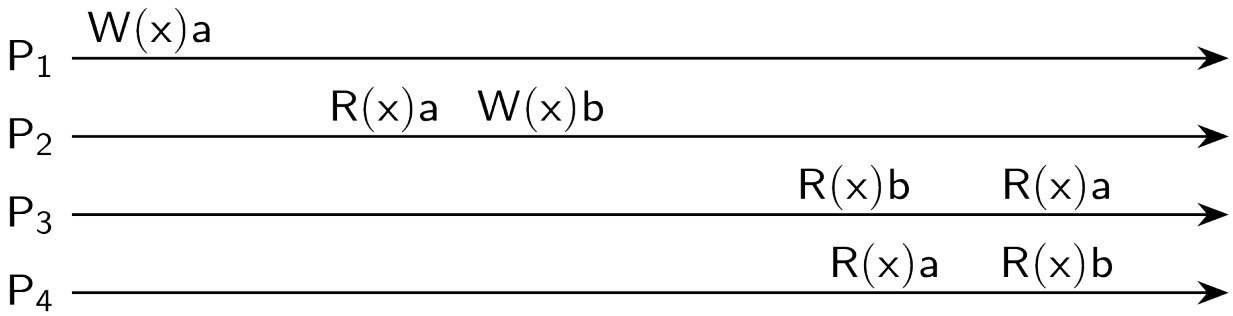
Uma violação de um banco de dados causalmente consistente

Exemplos

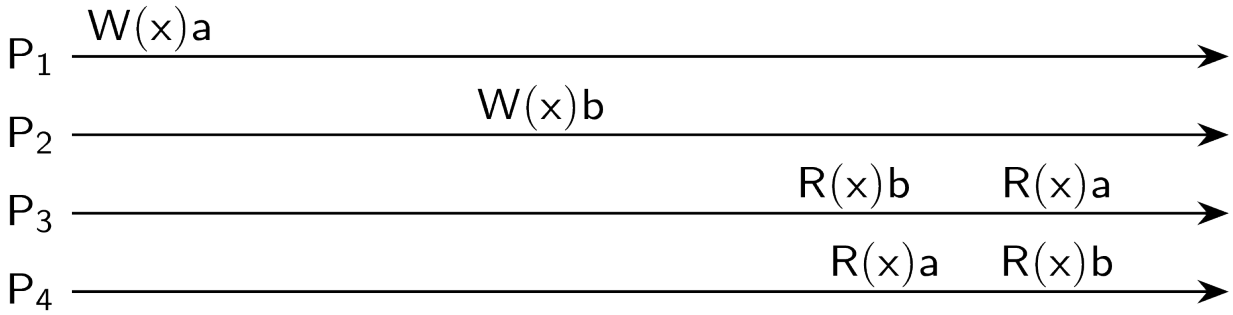


Uma sequência correta de eventos em um banco de dados causalmente consistente

Exemplos



Uma violação de um banco de dados causalmente consistente



Uma sequência correta de eventos em um banco de dados causalmente consistente

Agrupando operações

Consistência de entrada: definição

- Acessos a travas (locks) são sequencialmente consistentes.
- Nenhum acesso a uma lock pode ser executado até que todas as operações de escrita anteriores tenham sido concluídas em todos os lugares.
- Nenhum acesso a dados pode ser executado até que todos os acessos anteriores às locks tenham sido executados.

Agrupando operações

Consistência de entrada: definição

- Acessos a travas (locks) são sequencialmente consistentes.
- Nenhum acesso a uma lock pode ser executado até que todas as operações de escrita anteriores tenham sido concluídas em todos os lugares.
- Nenhum acesso a dados pode ser executado até que todos os acessos anteriores às locks tenham sido executados.

Ideia básica

Não é importante que leituras e gravações de uma série de operações sejam imediatamente conhecidas por outros processos. O importante é que o efeito da série como um todo seja conhecido.

Agrupando operações

Uma sequência de eventos válida para a consistência de entrada

Observação

A consistência de entrada implica que precisamos travar e destravar dados (implicitamente ou não).

Questão

Qual seria uma maneira conveniente de tornar essa consistência mais ou menos transparente para os programadores?

Consistência eventual

Definição

Considere uma coleção de data stores e operações de escrita.
Os data stores são ditos eventualmente consistentes quando, na ausência de atualizações a partir de um certo momento, todas as atualizações feitas até esse ponto são propagadas de tal forma que réplicas terão os mesmos dados armazenados (até que novas atualizações sejam feitas).
Geralmente usado em sistemas nos quais não há (ou é improvável) de haver conflitos escrita-escrita (e.g. CDN).

Consistência eventual

Definição

Considere uma coleção de data stores e operações de escrita.
Os data stores são ditos eventualmente consistentes quando, na ausência de atualizações a partir de um certo momento, todas as atualizações feitas até esse ponto são propagadas de tal forma que réplicas terão os mesmos dados armazenados (até que novas atualizações sejam feitas).
Geralmente usado em sistemas nos quais não há (ou é improvável) de haver conflitos escrita-escrita (e.g. CDN).

Consistência eventual forte

Ideia básica: se houver atualizações conflitantes, tenha um mecanismo de resolução globalmente determinado (por exemplo, usando NTP, a atualização ”mais recente” simplesmente vence).

Consistência eventual

Definição

Considere uma coleção de data stores e operações de escrita.
Os data stores são ditos eventualmente consistentes quando, na ausência de atualizações a partir de um certo momento, todas as atualizações feitas até esse ponto são propagadas de tal forma que réplicas terão os mesmos dados armazenados (até que novas atualizações sejam feitas).
Geralmente usado em sistemas nos quais não há (ou é improvável) de haver conflitos escrita-escrita (e.g. CDN).

Consistência eventual forte

Ideia básica: se houver atualizações conflitantes, tenha um mecanismo de resolução globalmente determinado (por exemplo, usando NTP, a atualização ”mais recente” simplesmente vence).

Consistência de programa

Pode ser usada para problemas monotônicos: problemas em que é possível chegar a uma resposta (parcial) mesmo se parte das informações de entrada estiverem faltando, Exemplo: preenchendo um carrinho de compras.
Observação: evita a sincronização global.

Consistência Contínua

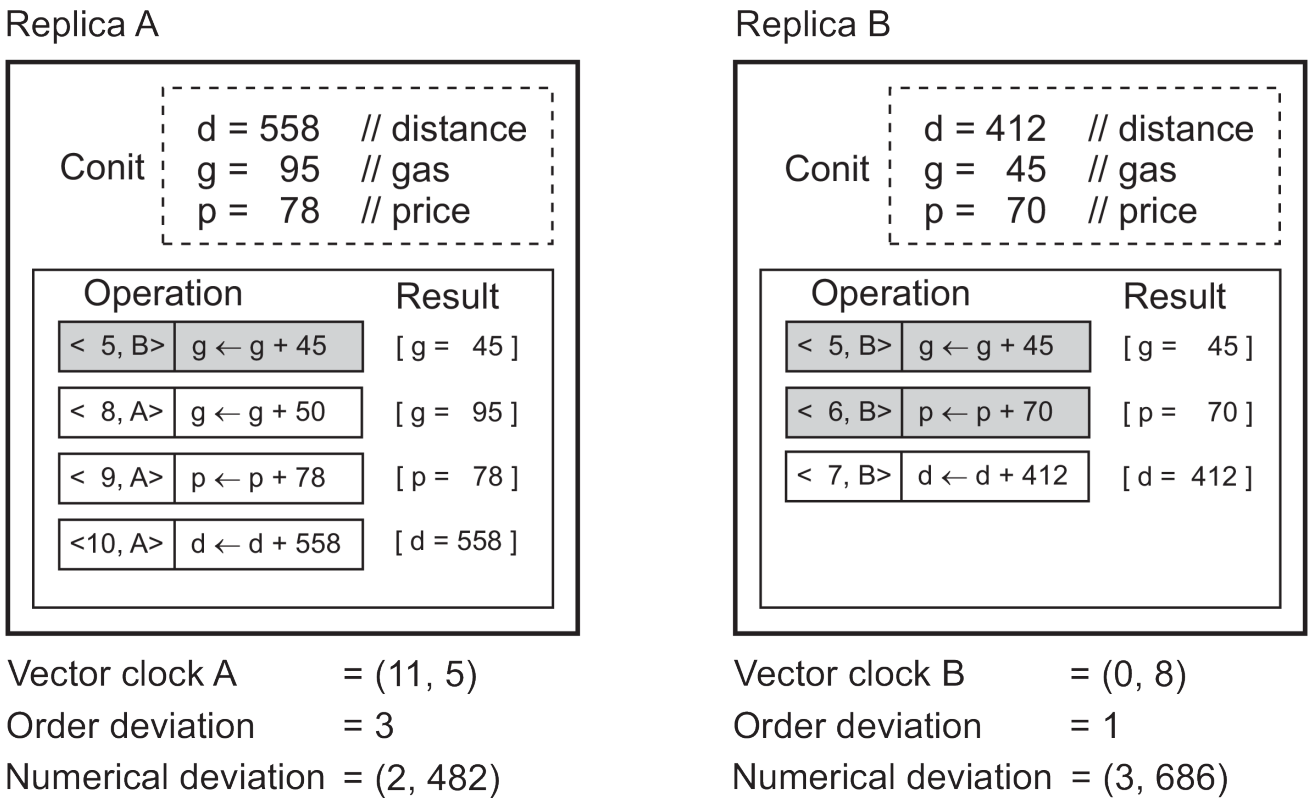
Graus de consistência

- réplicas podem diferir em seu valor numérico
- réplicas podem diferir em sua antiguidade relativa (staleness)
- pode haver diferenças quanto (número e ordem) de operações de atualização realizadas

Conit

Unidade de consistência (consistency unit) $\Rightarrow$ especifica a unidade de dados sobre a qual a consistência deve ser medida.

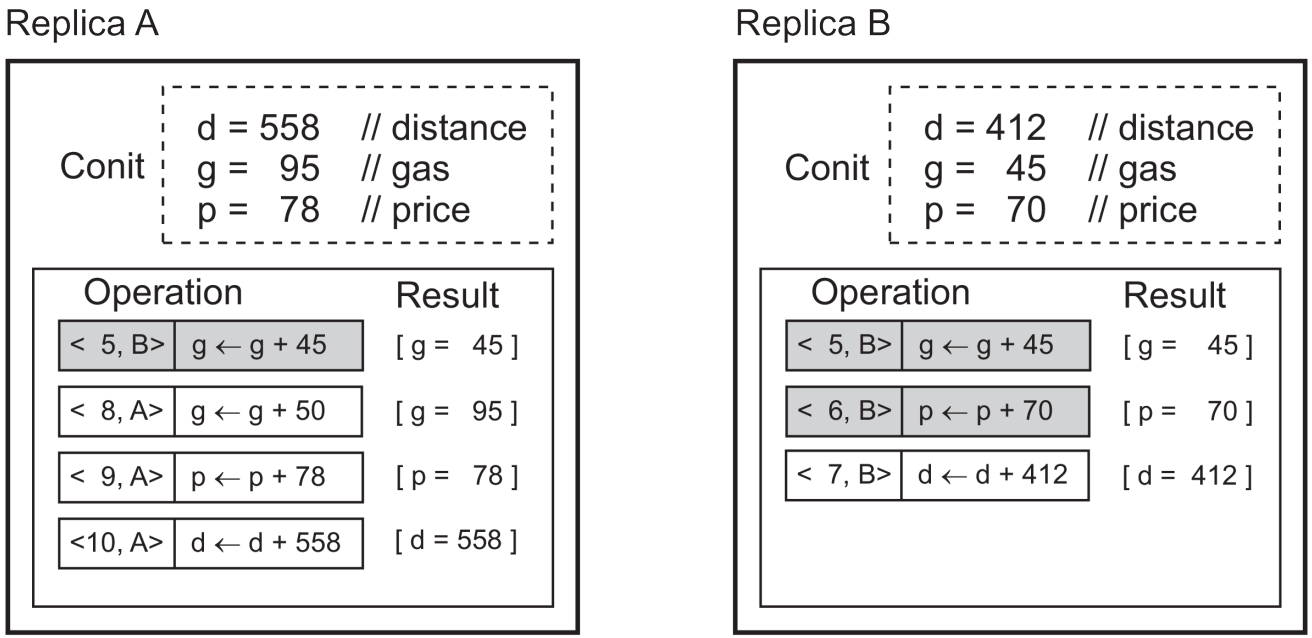
Exemplo: Conit



Conit (contém as variáveis *g*, *p* e *d*)

- Cada réplica tem um relógio vetorial: ([tempo conhecido] em A, [tempo conhecido] em B)
- *B* envia a operação *A* $[\langle 5, B\rangle : g \leftarrow d + 45]$; *A* tornou esta operação permanente (não pode ser revertida)

Exemplo: Conit



Vector clock A = (11, 5)
Order deviation = 3
Numerical deviation = (2, 482)

Vector clock B = (0, 8)
Order deviation = 1
Numerical deviation = (3, 686)

Conit (contém as variáveis *g*, *p* e *d*)

- *A* tem três operações pendentes $\Rightarrow$ desvio de ordem = 3
- *A* ainda não enxergou duas operações de *B* e a diferença máxima é de 70 + 412 unidades $\Rightarrow$ desvio numérico = $(2, 482)$