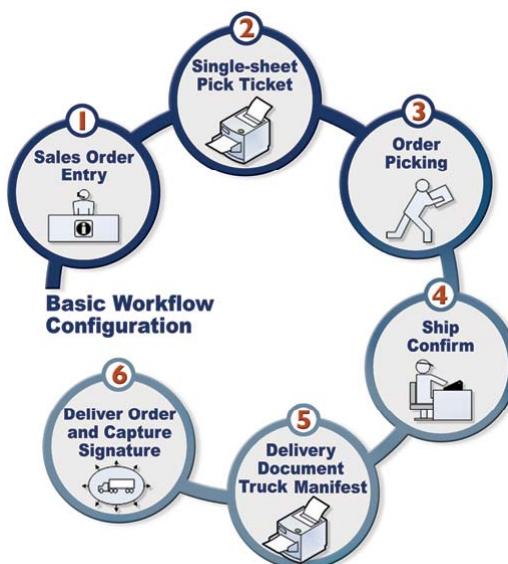


# Projeto de Sistemas Distribuídos

## Atomicidade e Concorrência

Prof. Msc. Marcelo Iury de Sousa Oliveira  
[marceloiury@gmail.com](mailto:marceloiury@gmail.com)  
<http://sites.google.com/site/marceloiury/>

SI – Projeto de Sistemas Distribuídos



# Transações

“Ou vai, ou racha!”

## Transação

- Conjunto de operações que precisam ser executadas atomicamente
  - Ou tudo, ou nada
  - Transferência = saque + depósito
  - Reserva de vôo = reserva de todos os trechos
- Falha de alguma das partes componentes
  - Desfazer o que foi feito
  - Roll back

# Transação

- Propriedades
  - Atomicidade
    - Ou tudo, ou nada
  - Consistência
    - Transação não viola princípios do sistema
  - Isolamento (ou seriabilidade)
    - Transações concorrentes não interferem
  - Durabilidade
    - Uma vez terminada, ela é imutável

5

# Transação

- Para implementar transação é necessário
  - Armazenamento estável
  - Primitivas
    - BEGIN/END\_TRANSACTION
    - ABORT\_TRANSACTION
    - READ
    - WRITE

6

## Seriabilidade

- Transações concorrentes não podem interferir umas com as outras
- Resultado final TEM QUE SER um dos resultados obtidos se elas fossem executadas sequencialmente, sem concorrência
- Sistema operacional é responsável pela ordenação

7

## Seriabilidade

```
BEGIN_TRANSACTION
X = 0
X += 1
END_TRANSACTION
```

```
BEGIN_TRANSACTION
X = 0
X += 2
END_TRANSACTION
```

```
BEGIN_TRANSACTION
X = 0
X += 3
END_TRANSACTION
```

Escalonamento 1	X = 0	X += 1	X = 0	X += 2	X = 0	X += 3	Legal
Escalonamento 2	X = 0	X = 0	X += 1	X += 2	X = 0	X += 3	Legal
Escalonamento 3	X = 0	X = 0	X += 1	X = 0	X += 2	X += 3	Illegal

8

## Armazenamento estável

- Três tipos básicos de armazenamento
  - RAM: perde o conteúdo sem energia
  - Disco: perde o conteúdo no caso de falha
  - Armazenamento estável: não perde o conteúdo
- Solução trivial
  - Duplicação dos discos
  - Depois veremos RAID, que é mais genérico

9

## Como implementar?

- Transações são uma boa!
- Sair simplesmente escrevendo/mudando dados não garante atomicidade nem seriabilidade
- Duas formas de atingir isso
  - Espaço de trabalho privativo
  - Log com escrita antecipada

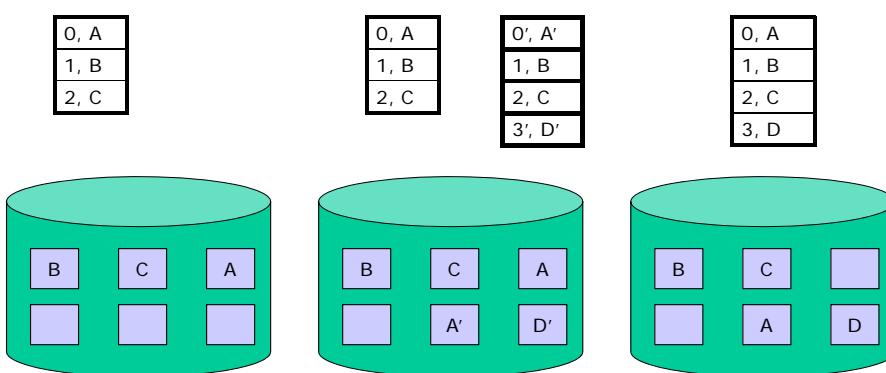
10

## Espaço de trabalho privativo

- Processo copia dados reais para uma área privativa, e trabalha nessa área
  - Shadow page
- Quando a transação for confirmada, os dados reais são substituídos pelos do espaço privativo
- Enquanto em andamento, apenas o processo vê os dados atuais

11

## Espaço de trabalho privativo



12

## Log com escrita antecipada

- Antes de mudar qualquer coisa, escrever como era e como ficou
- Se der tudo certo, escreve no log
- Se houver erro, basta desfazer o que foi feito
- Útil para recuperar de falha
  - Na volta, verifica o log
  - Transação sem fechamento é desfeita

13

## Como confirmar a transação?

- Até agora, dizemos que chegou no fim e pronto
- Necessário alguma forma de confirmar atomicamente as alterações
- Protocolos de Commit
  - Duas fases
  - Três fases

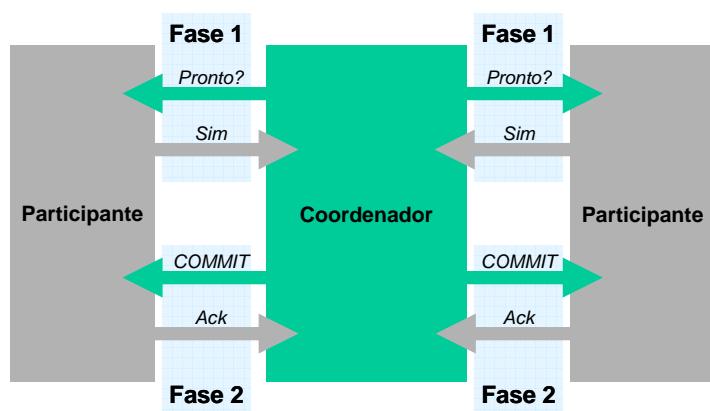
14

## Protocolos de commit

- Commit em duas fases
  - Um coordenador
  - Usa
    - Armazenamento estável
    - Log com escrita antecipada
  - Nenhum nó pára para sempre
  - Todos podem falar com todos

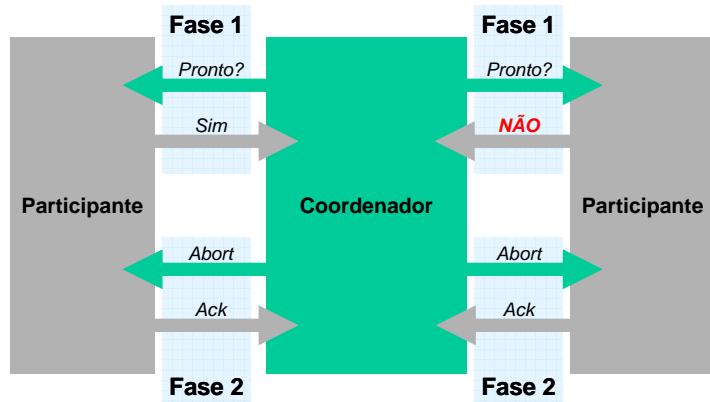
15

## Commit em duas fases



16

## Commit em duas fases



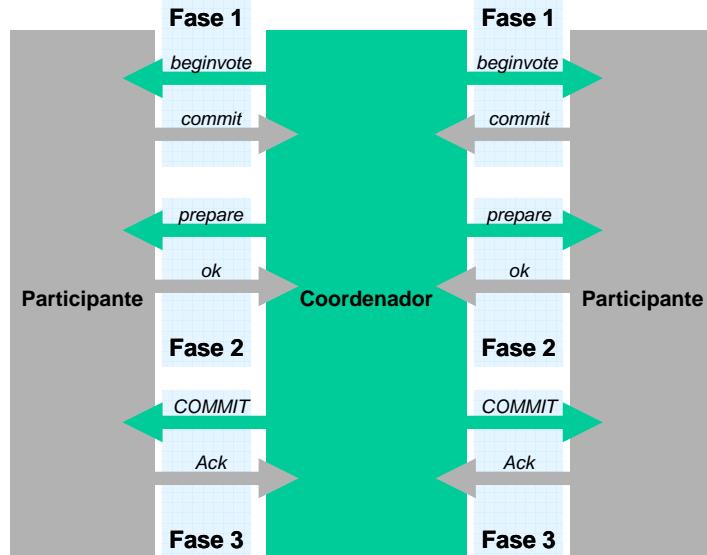
17

## Protocolos de commit

- Commit em três fases
  - Evita bloqueio de recursos no caso de falha
  - Incorporação de timeout

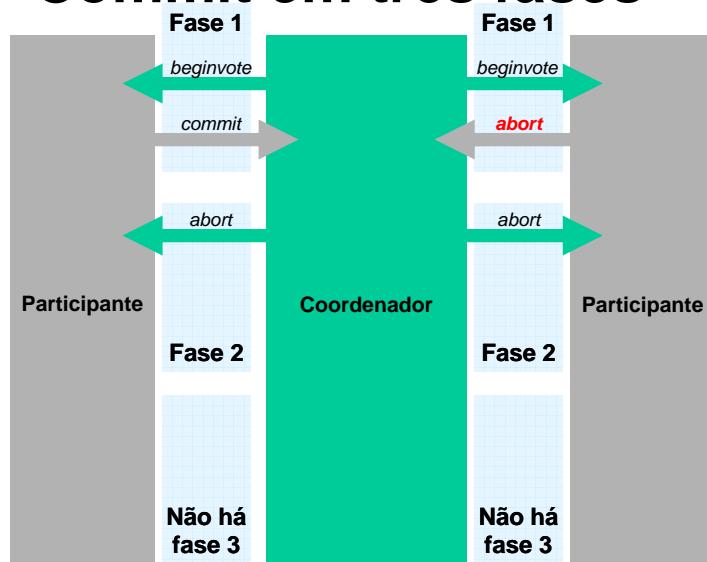
18

## Commit em três fases



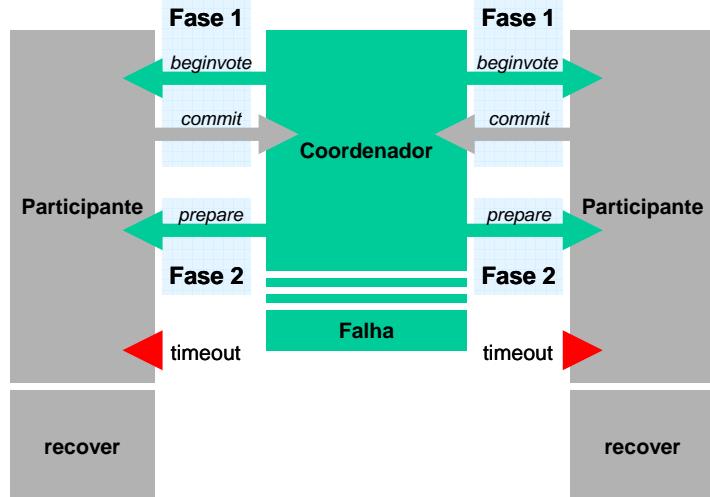
19

## Commit em três fases



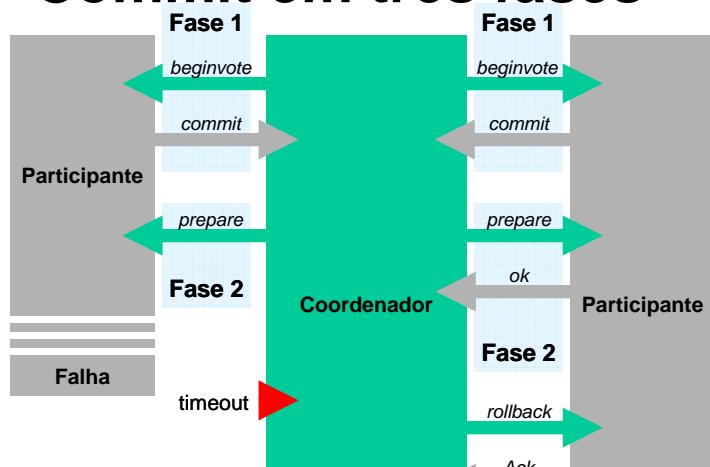
20

## Commit em três fases



21

## Commit em três fases



22

## Controle de concorrência

- Transações podem ocorrer de forma concorrente
- Como garantir que uma não vai atrapalhar a outra
  - Atingir o princípio da seriabilidade
- Basicamente duas formas
  - Lock
  - Controle otimista

23

## Lock

- Solução mais antiga e mais usada
- Antes de usar o arquivo, faz um lock
  - Gerenciador de locks centralizado ou distribuído
  - Gerente nega novo lock quando já há um
- Lock geralmente é feito pelo sistema de transações

24

# Lock

- Granulosidade
  - Arquivo
  - Registro
  - Byte...
- Leitura x escrita
  - Escrita exclusiva, leitura compartilhada

25

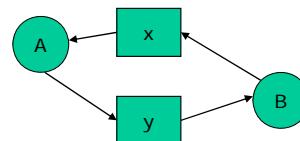
# Lock

- Geralmente feito em duas fases
  - Aquisição de todos e liberação de todos
  - Se todas os locks de transação são em duas fases é garantida a seriabilidade
- Duas fases estritas
  - Só libera após commit ou roll back
- Potencial para deadlock
  - A → x, y; B → y, x

26

## Deadlock

- Dois processos necessitam dos mesmos recursos e fazem o locking em ordem diferente
- Formas de evitar
  - Obrigar uma ordem no locking
  - Gerente de lock mantém um grafo de locks
    - A possui X e quer Y
    - B possui Y e quer X



27

## Controle otimista

- Idéia simples
  - Segue na transação sem se preocupar
- Como é feito o controle?
  - Transação mantém registro do que foi lido/escrito
  - No momento do commit, verifica se alguma outra transação usou os mesmos
  - Se sim, aborta; caso contrário conclui

28

## Controle otimista

- Funciona na grande maioria das vezes
  - Conflitos são raros
- Vantagens
  - Livre de deadlocks
  - Máximo paralelismo possível
- Desvantagem
  - Só percebe o problema no final
  - Ambientes pesados terão desempenho fraco

29

## Controle de concorrência

- Na realidade, mais algumas formas
  - Marcas de tempo
    - Usa a idéia de Lamport e marca os objetos com o tempo das transações
  - Ordenação de marcas de tempo múltiplas
    - Permite leitura durante transação
- Verifiquem no Coulouris (p.396)

30