# Replicação

## Sistemas Distribuídos

abril de 2019

# Motivação

- tolerância a falhas
  - disponibilidade
- desempenho
  - proximidade
  - divisão da carga de trabalho: escalabilidade

# Consistência

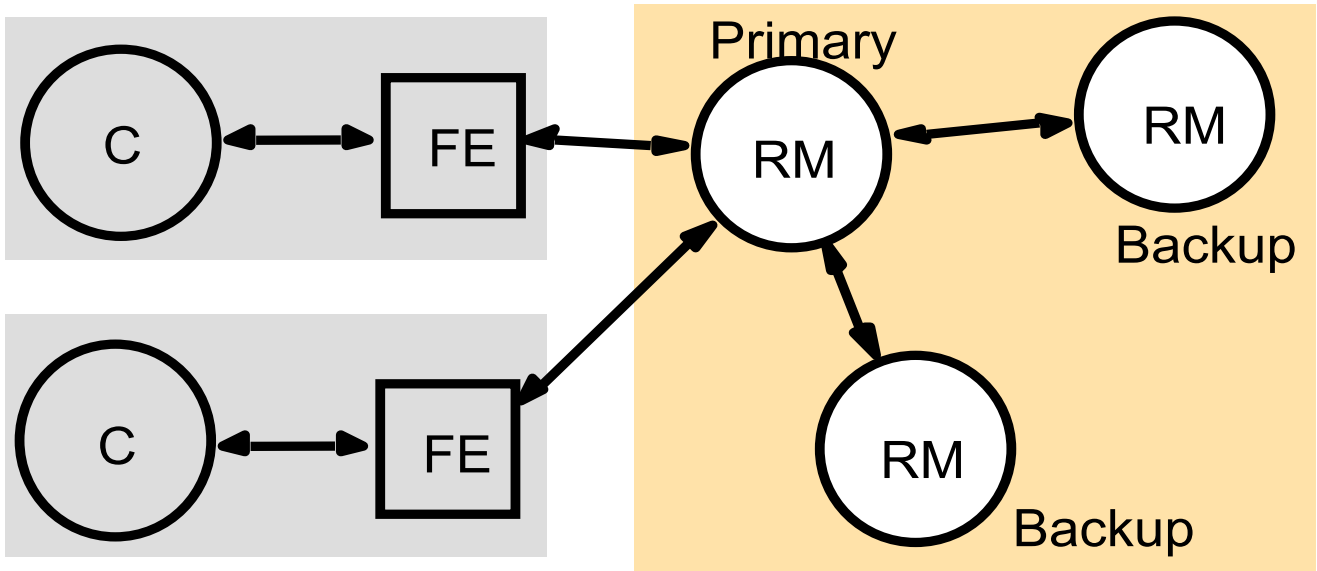
- Se várias cópias de um serviço estão disponíveis, como ficam seus dados?

- intuitivamente, gostaríamos que o resultado de uma operação fosse independente da réplica utilizada...
  - como capturar essa idéias: *modelos de consistência*
  - grau de acoplamento muito forte: custo alto!
    - clientes com caches
    - latência
    - ...
  - uso de modelos de consistência menos rígidos

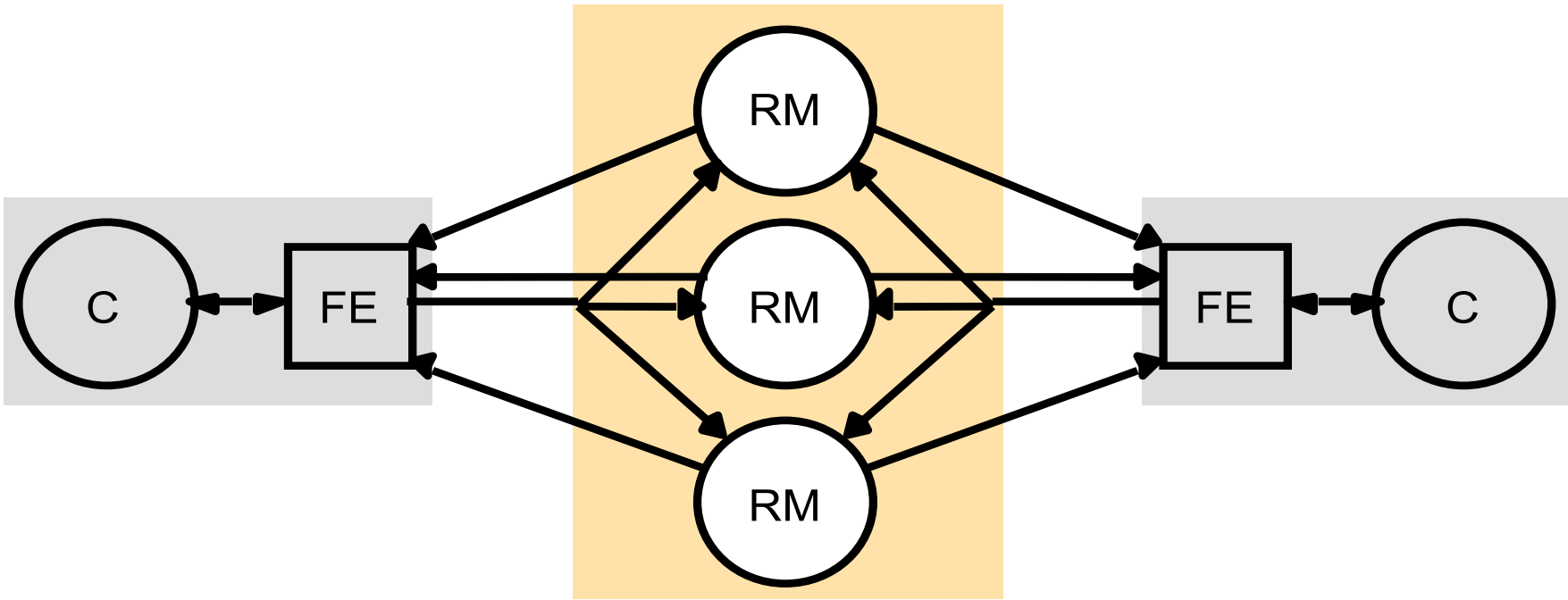
# Servidores: Arquiteturas de Replicação

- cópia primária
- replicação ativa

# Replicação de processos com cópia primária



# Replicação de processos sem cópia primária



# Replicação baseada em cópias primárias

- cada item de dados (ou todos) associado a um servidor primário
- leituras ocorrem em todas as cópias
- escritas:
  1. sempre no servidor primário
     - atualização em réplicas dispara atualização remota em servidor primário
     - atualização em servidor primário pode ou não bloquear até atualização ser propagada a todas as cópias
  2. local-write: cópia do dado primário transferido para local da atualização

# Replicação Ativa

- todas as réplicas realizam escritas "simultaneamente"
- necessidade de um protocolo que garanta uma determinada ordem na realização das operações
  - *broadcast* total ou causal
- ou consenso sobre estado/atualizações
  - algoritmo de consenso

# Replicação de processos: mascaramento de falhas

- tolerância a falhas *fail-stop* com uso de uma réplica primária e n secundárias
- monitoramento do número de réplicar ativas em funcionamento

# Localização de réplicas

- clusters — replicação sem distribuição geográfica
- cração de réplicas em resposta a padrões de acesso
- modelos de previsão de acessos
- dinamismo: sistemas p2p

# Modelo de servidores como máquinas de estado

- Servidores mantêm conjuntos de dados e processam requisições de clientes
- Um servidor processa uma requisição por vez[1]
- Cada servidor é uma *máquina de estado*

**propriedades**

- estado inicial igual para todas as réplicas corretas
- execução determinística
- coordenação entre réplicas

[1] ou equivalente

# Modelo de servidores como máquinas de estado

## propriedades

- estado inicial igual para todas as réplicas corretas
- execução determinística
- coordenação entre réplicas
  - RME tradicional: todas as réplicas corretas executam a *mesma* sequencia de operações.
  - consistência forte!

## propriedades

*safety* todas as réplicas crretas executam mesma sequência de operações

*liveness* todas as invocações corretas de clientes são executadas

# Formalização de modelos de máquinas de estado

**falhas nos servidores:**

- *fail-stop*
- falhas bizantinas

**canais de comunicação:**

- confiável
- não confiável
- *lossy*
- justo

**sincronismo**

- sistemas síncronos, assíncronos e parcialmente síncronos

# falhas em servidores

- *fail-stop*
- falhas bizantinas
  - comportamento arbitrário

# Tipos de canais

- confiável
  - toda mensagem enviada a um servidor correto é recebida
- não confiável
  - mensagens podem ser modificadas, geradas ou destruídas
- *lossy*
  - mesagens podem ser perdidas mas não modificadas
- justo
  - se uma msg é enviada infinitas vezes, será recebida infinitas vezes

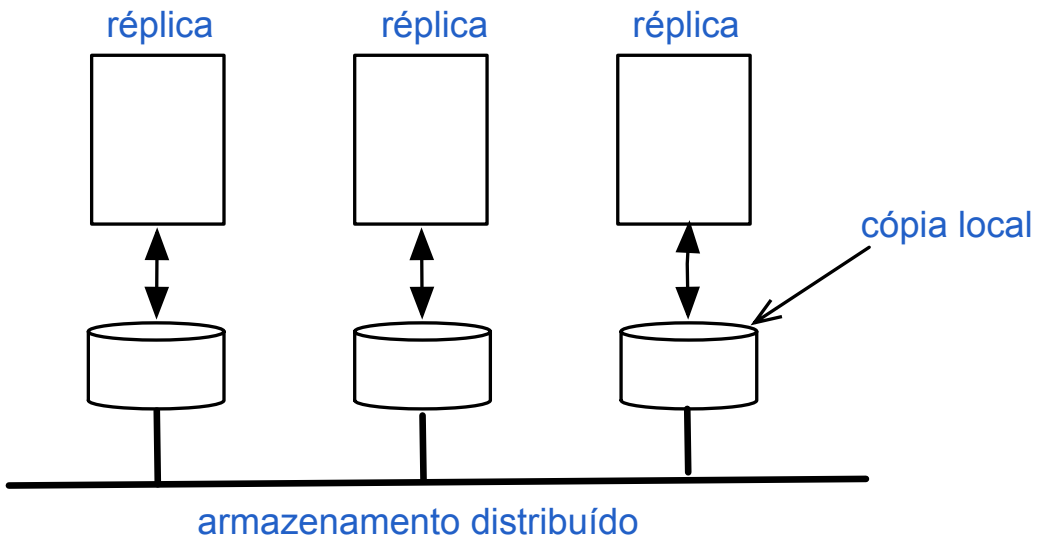
# Resultados teóricos

- equivalência entre ordem total e consenso
- impossibilidade de consenso tolerante a falhas em sistemas assíncronos
  - na prática assume-se que sistemas são parcialmente síncronos
- impossibilidade de entrega de mensagens confiáveis sobre canais não confiáveis

# Modelo de consistência

# Modelos centrados em dados

- modelos de consistência tradicionalmente discutem resultado de *leituras* e *escritas*
- podemos pensar em termos de serviços de *armazenamento de dados*
  - um cliente trabalha sobre uma réplica dos dados



# Consistência Estrita

**definição:**

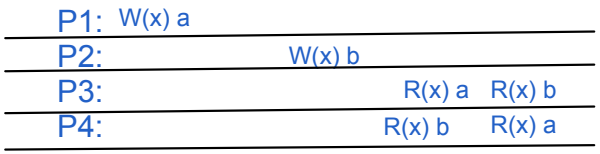
- qualquer leitura de item de dado $x$ retorna o valor correspondente à escrita mais recente em $x$

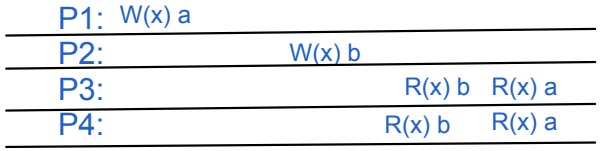
- dificuldades até na definição de "mais recente"
- dependência em tempo global

# Consistência Sequencial

**definição:**

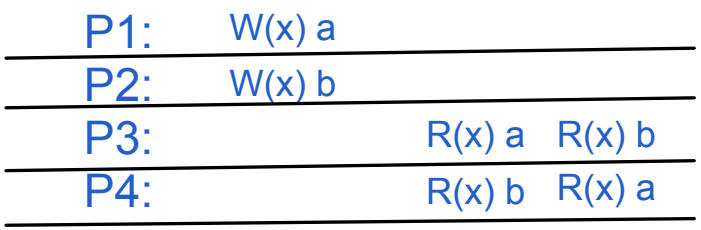
- O resultado de qualquer execução é compatível com alguma execução sequencial (intercalada) das operações (de leitura e escrita) realizadas pelos processos do grupo.



# Consistência Causal

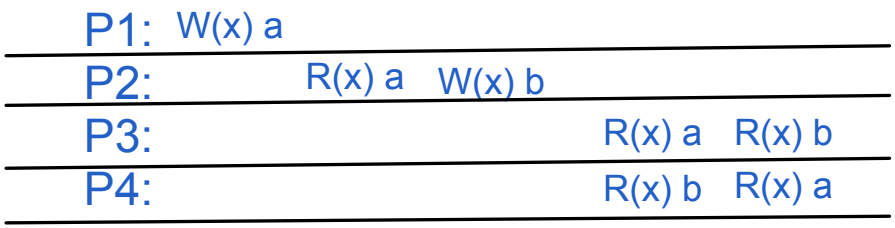
- garantia de que operações são observadas na ordem de possível precedência causal



# Consistência Causal

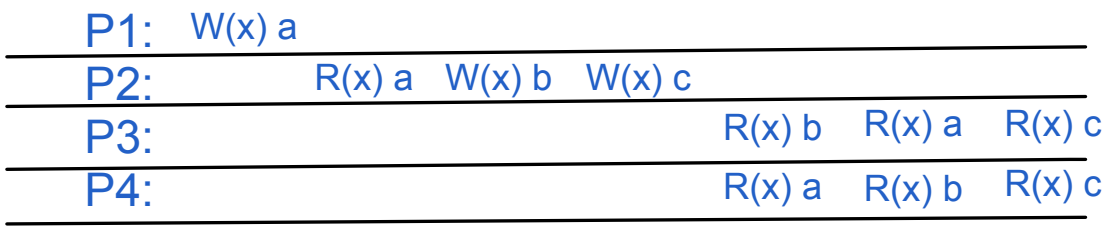
- preocupação apenas com consistência de leituras e escritas causalmente relacionadas

causalmente inconsistente:



# Consistência FIFO

- as operações de escrita feitas por um processo devem ser vistas na ordem em que ocorreram neste processo por todos os demais, mas não há imposições sobre a ordenação das escritas realizadas por diferentes processos



# Modelos centrados no cliente

- garantias do ponto de vista de um cliente

**alterações no modelo**

- cliente não está amarrado a uma réplica específica

# Consistência *read-your-write*

- um determinado processo sempre enxerga as atualizações já feitas por ele mesmo

# Consistência Monotônica

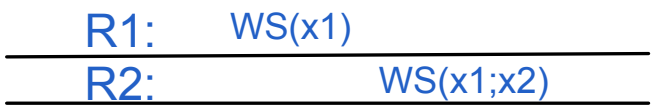
## Leitura monotônicas

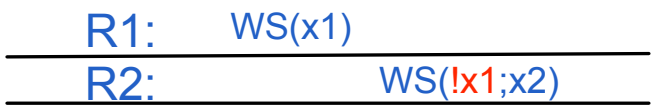
- leitura nunca retorna valores anteriores aos já vistos pelo mesmo processo

# Consistência Monotônica

## Escritas monotônicas

- sistema garante serialização das escritas de um mesmo processo




escritas feitas por um mesmo cliente em réplicas diferentes

# Tradeoffs CA

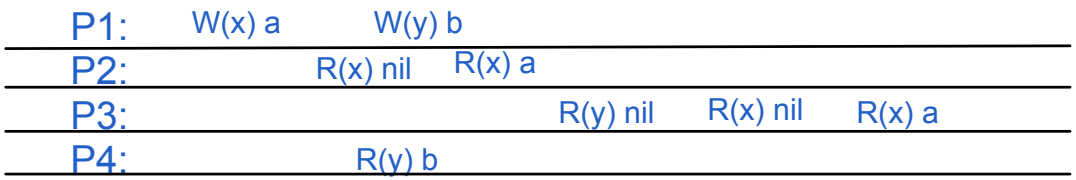
- ênfase em consistência: cliente precisa saber lidar com indisponibilidade para escritas
- ênfase em disponibilidade: aplicações que podem conviver com dados um pouco desatualizados
  - consistência *fraca* e janelas de inconsistência
  - DNS
  - LDAP

# Consistência Eventual

## exemplos de aplicações

- Web
- DNS
- LDAP
- servidores de jogos

- cenário bastante comum em diversas aplicações
- *janela de inconsistência*



# Teorema CAP

**_tradeoffs_**

- consistência
- disponibilidade
- particionamento da rede

- Eric Brewer. Towards robust distributed systems (abstract). in *Proc. 19th ACM Symposium on Principles of Distributed Computing*. 2000.

# Teorema CAP

## prova

Assume the service consists of servers p1,p2,...,pn, along with an arbitrary set of clients. Consider an execution in which the servers are partitioned into two disjoint sets: p1 and p2,...pn. Some client sends a read request to server p2. Because p1 is in a different component of the partition from p2, the system loses every message from p1 to p2. Now consider the following two cases:

1. A previous write of value v1 has been requested of p1 and p1 has sent an ok response.
2. A previous write of value v2 has been requested of p1, and p1 has sent an ok response.

No matter how long p2 waits, it cannot distinguish these two cases, hence it cannot determine whether to return response v1 or v2. Its choice is to either eventually return a (possibly wrong) response or to never return a response.

- S. Gilbert and N. Lynch, *Perspectives on the CAP Theorem*, Computer, 45(2), pp. 30-36, Feb. 2012.

# CAP e redes geograficamente distribuídas: PACELC

**_tradeoffs_**

- consistência
- disponibilidade
- particionamento da rede
- latência

- D. Abadi, *Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story*,Computer, 45(2), Feb. 2012.

# PACELC — sistemas com requisitos de baixa latência

- latência alta $\rightarrow$ indisponibilidade

- para manter disponibilidade alta, necessário manter muitos servidores
- chance de falhas alta leva a replicação de dados
- replicação sobre redes geográficas para garantia maior de disponibilidade

# replicação em sistemas com requisitos de baixa latência

- replicação ativa: atualizações enviadas para todos
- replicação passiva: atualizações enviadas para um mestre

# Replicação, consistência e disponibilidade

## sistemas de quorum

- $N$ réplicas
- $W$ réplicas precisam confirmar atualizações para completá-las
- $R$ réplicas contactadas na leitura

- se $W + R > N$ temos consistência forte
- replicação com cópia primária e modo síncrono: $W = N$
- replicação com cópia primária e modo assíncrono: $W = 1, R = 1$

# Replicação, consistência e disponibilidade

- foco de sistema e *trade-offs*: disponibilidade, tolerância a falhas, consistência, tempo de acesso, ...
  - foco em tolerância a falhas com consistência: $N = 3, R = 2, W = 2$
  - foco em tolerância a falhas sem consistência: $N = MUITOGRANDE, R = 1, W = 1$
  - otimizaçoes de leituras ($W = N$ e $R = 1$) ou de escritas ($W = 1$ e $R = N$)?

# tolerância a falhas e checkpoints

# Referências


- W. Vogels. Eventually Consistent. *Communications of the ACM*. 52(1). 2009.
- E. Brewer. CAP twelve years later: How the "rules" have changed. *Computer*. 45(2). 2012.