Replicação

Replicação

- Conceito simples:
  - Manter **cópias** dos dados e do software do serviço em vários computadores

- Exemplos do nosso dia-a-dia?

Infinispan

Spark

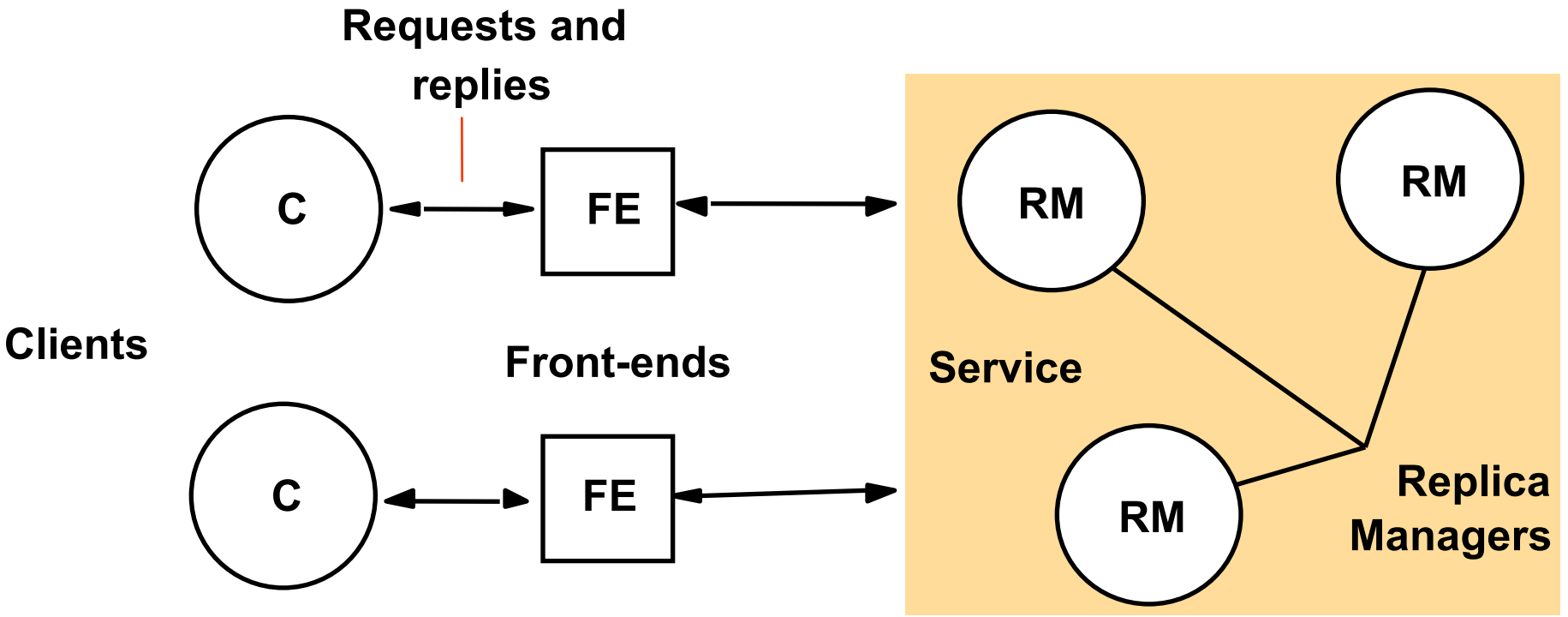
Google Cloud Platform

Dropbox

Microsoft Azure

Google Bigtable

Modelo arquitetural de base

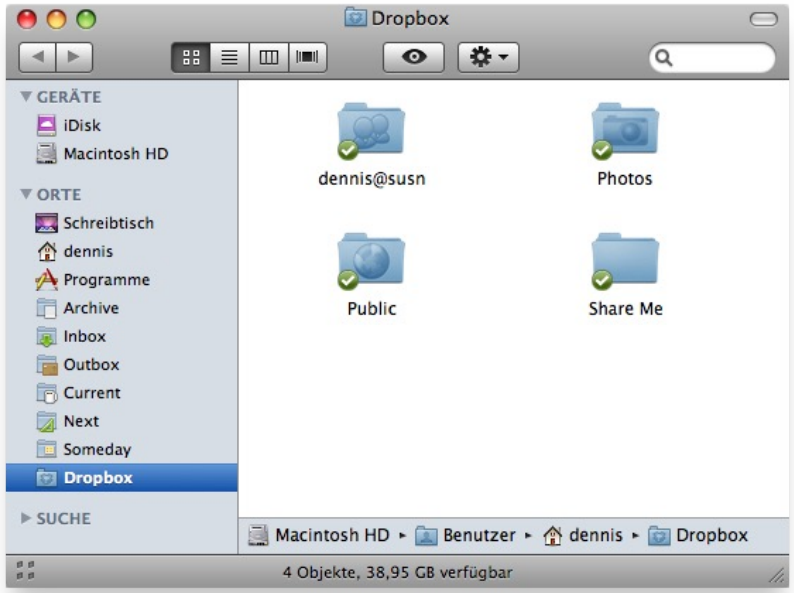


Replicação: que benefícios?

- Melhor **disponibilidade**
  - O sistema mantém-se disponível mesmo quando:
    - Alguns nós falham
    - A rede falha, tornando alguns nós indisponíveis
- Melhor **desempenho e escalabilidade**
  - Clientes podem aceder às cópias mais próximas de si
    - Melhor desempenho
    - Caso extremo: cópia na própria máquina do cliente (*cache*)
  - Algumas operações podem ser executadas apenas sobre algumas das cópias
    - Distribui-se carga, logo consegue-se maior escalabilidade

Replicação: requisitos

- **Transparência** de replicação
  - Utilizador deve ter a ilusão de estar a aceder a um único objeto lógico
  - Objeto lógico implementado sobre diferentes cópias físicas, mas sem que o utilizador se aperceba disso



Replicação: requisitos

- **Coerência**
  - Idealmente, um cliente que leia de uma das réplicas deve sempre ler **o valor mais atual**
    - Mesmo que a escrita mais recente tenha sido solicitada sobre outra réplica
  - Mais tarde definiremos e discutiremos esta propriedade com maior rigor

- *Nota: em inglês diz-se* consistency *mas em português consistência significa "estado de um líquido que se torna pastoso ou se torna espesso"...*

Quantas faltas esperamos tolerar?

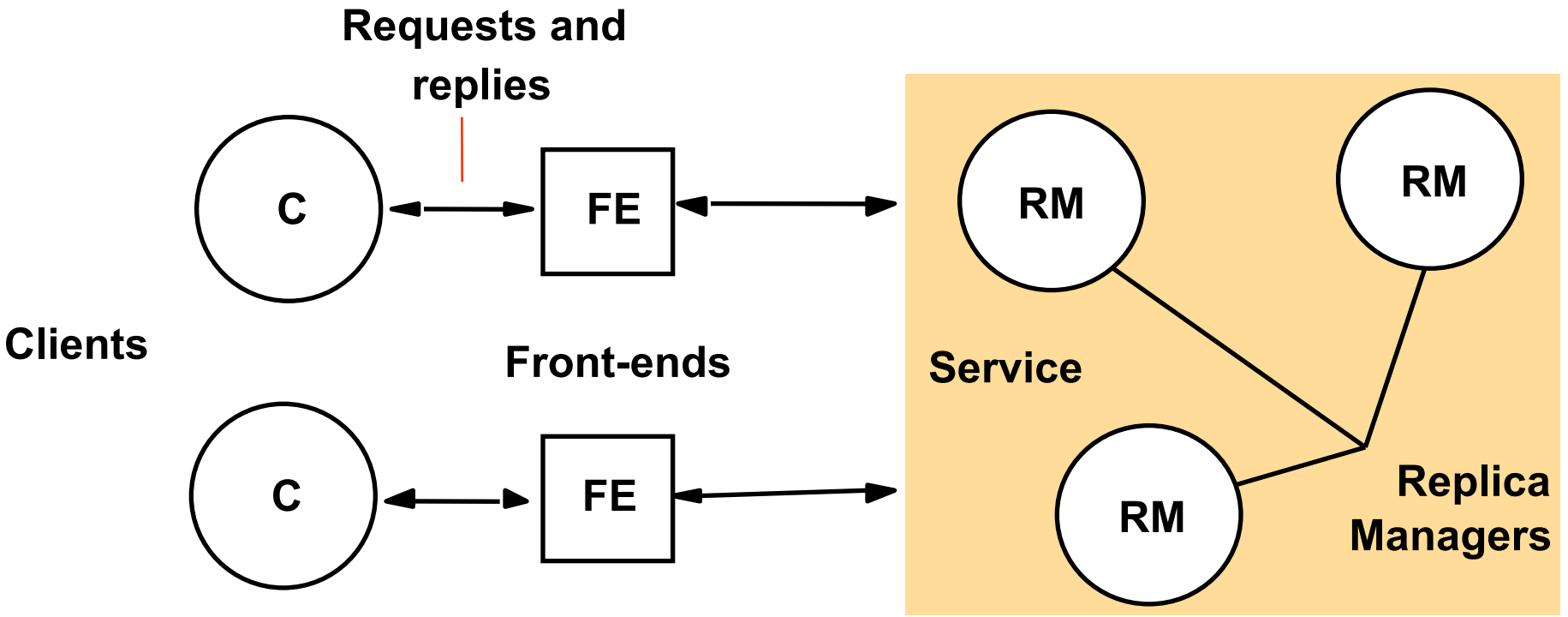
- Se as falhas forem **silenciosas**?
  - Esperaríamos que ***f+1* réplicas** tolerassem *f* nós em falha
  - Basta que **uma** réplica correcta responda para termos o valor correcto

- E se os nós puderem falhar de forma **arbitrária/bizantina**?
  - Aí pode acontecer que, entre as respostas que recebermos, até a um máximo de **f podem ser erradas**
    - Logo a única alternativa é recebermos **respostas iguais de pelo menos f+1 réplicas corretas**
    - Logo esperaríamos que ***2f+1* réplicas** tolerassem *f* nós com falhas bizantinas

- Mas a **realidade é mais complicada** e normalmente precisamos de mais réplicas…

Sistemas replicados

Modelo arquitetural de base



As cinco fases de uma invocação num sistema replicado

- Pedido
  - O *front-end* envia o pedido **a um ou mais** gestores de réplica
- Coordenação
  - Os gestores de réplicas coordenam-se para executarem o pedido **coerentemente**
- Execução
  - Cada gestor de réplica **executa** o pedido
- Acordo
  - Os gestores de réplicas **chegam a acordo** sobre o efeito do pedido
- Resposta
  - **Um ou mais** gestores de réplica respondem ao *front-end*

Diferentes soluções podem omitir ou reordenar algumas fases.

Pressupostos habituais

- Processos podem falhar **silenciosamente**
  - Ou seja, não há falhas arbitrárias de processos
- Operações executadas em cada gestor de réplica **não deixam resultados incoerentes** caso falhem a meio
- Replicação total
  - Cada gestor de réplica mantém cópia de **todos** os objetos lógicos
- Conjunto de gestores de réplica é **estático e conhecido** *a priori*

Os limites da replicação

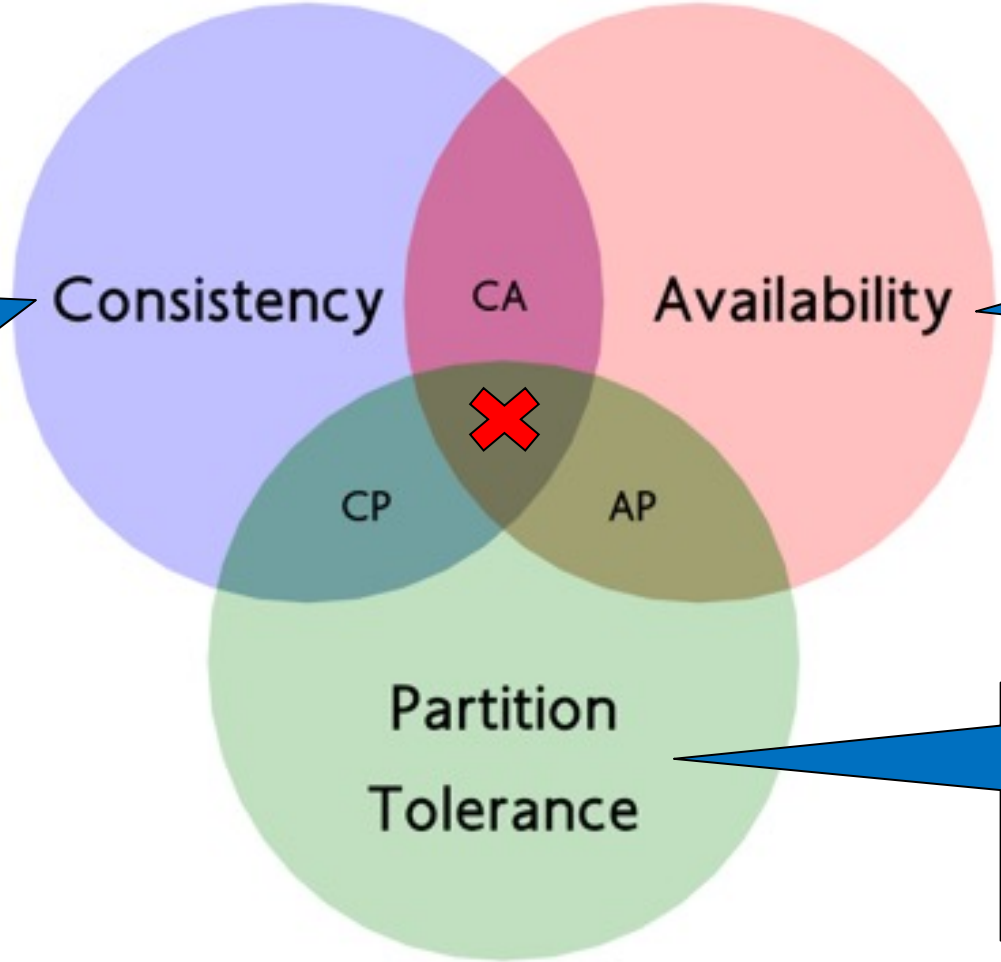
Idealmente, um sistema replicado deveria oferecer...

- Coerência forte (***C**onsistency*)
  - Leituras observam sempre versão **mais recente**
  - Coerência sequencial, linearizabilidade, etc.
- Alta disponibilidade (***A**vailability*)
  - Qualquer acesso recebe uma resposta que não é “erro”
- Tolerância a partições (***P**artition-tolerance*)
  - Sistema funciona mesmo na presença de partições de rede
  - Ou seja, apesar de um número arbitrariamente alto de mensagens se perderem ou atrasarem

CAP

O teorema CAP:
qualquer sistema só pode alcançar 2 destas 3 propriedades

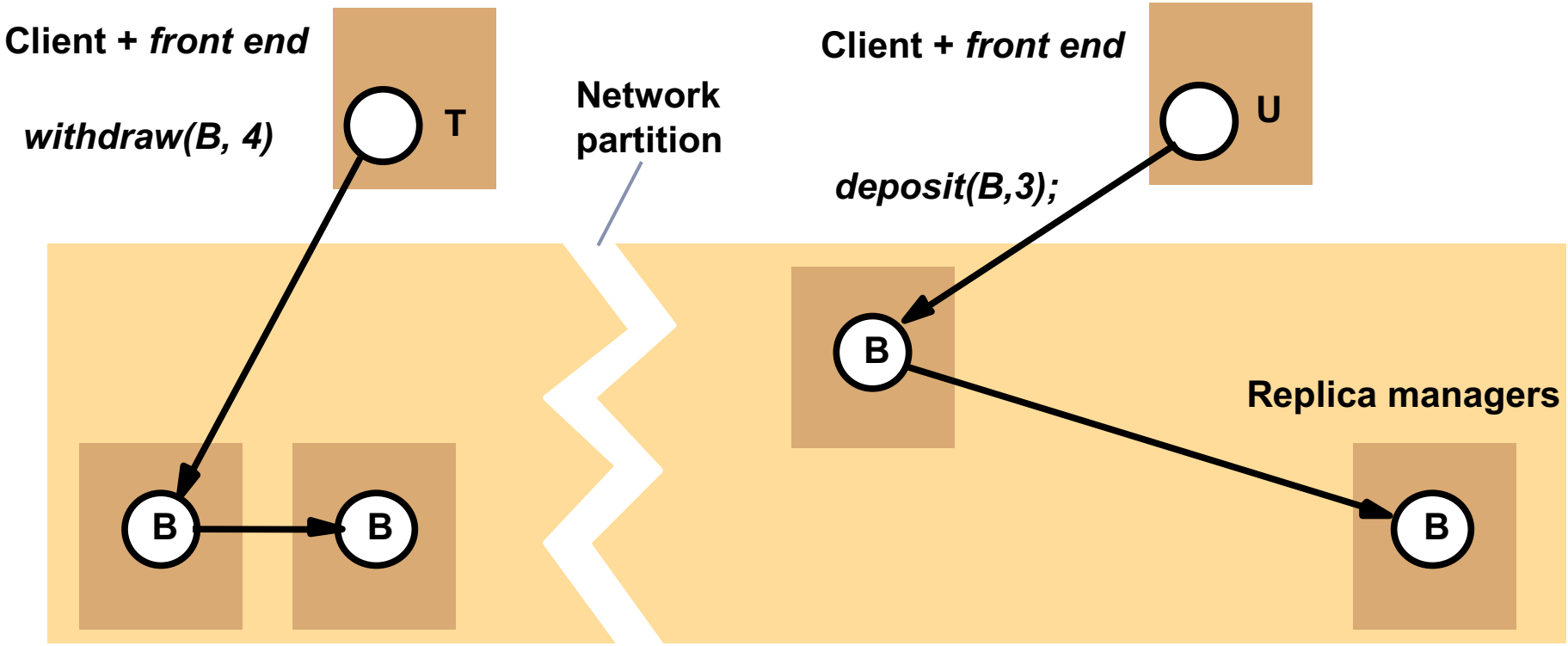
Leituras observam sempre versão mais recente
Coerência sequencial, linearizabilidade



Acessos devolvem sempre resposta

Sistema funciona mesmo na presença de partições de rede

Como lidar com partições de rede?



O teorema CAP, por outras palavras

- Na presença de uma partição de rede, o sistema replicado precisa de **escolher** entre:

1. Garantir que as respostas são **sempre coerentes**
   - Quando não seja possível garantir isso, retornar erro ou não responder
   - Foco na **coerência**

2. **Responder aos pedidos** de qualquer *front-end* sem dar erro
   - Por vezes retornando valores desatualizados
   - Foco na **disponibilidade**

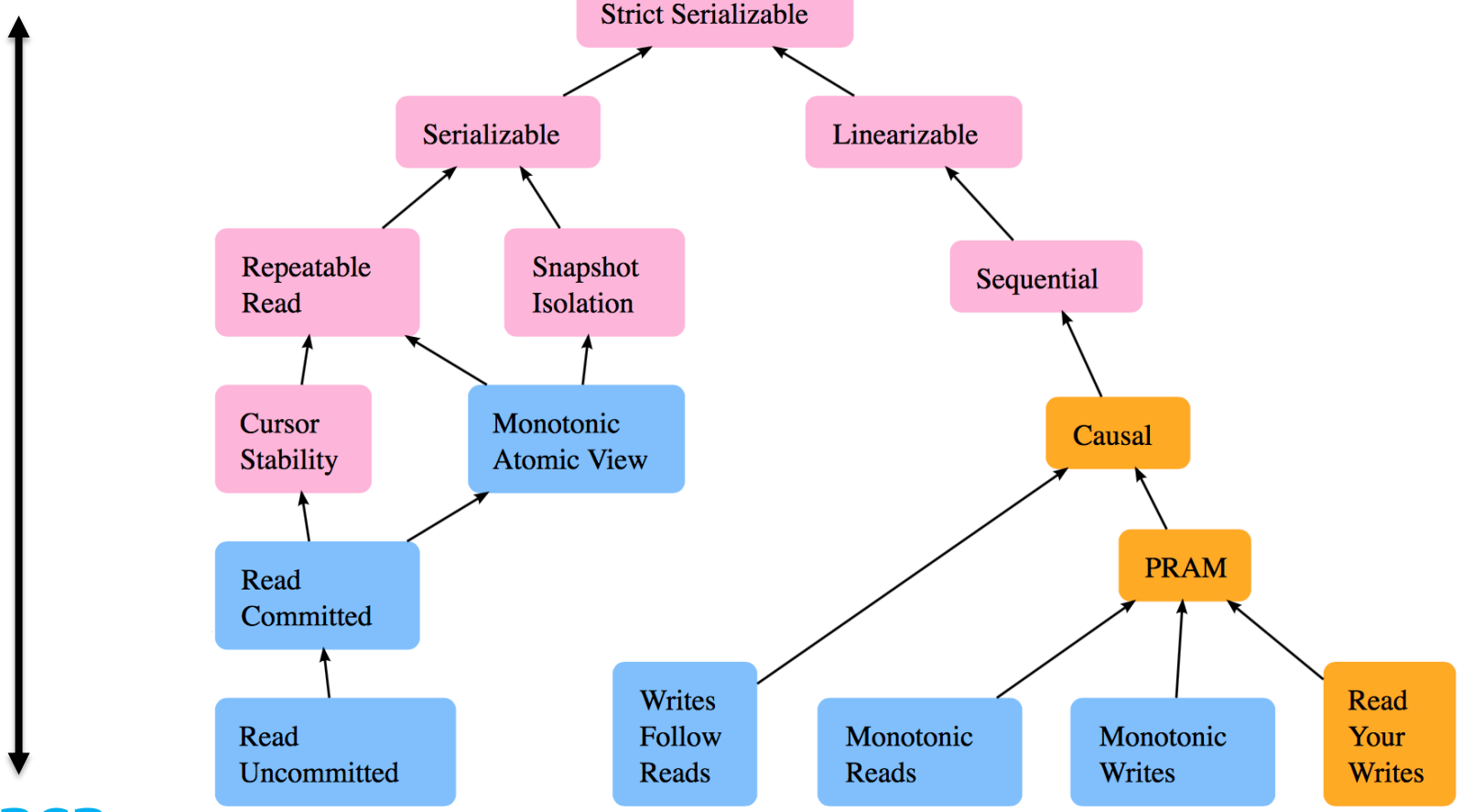
Para refletir

- Dada a importância de assegurar elevados níveis de **disponibilidade** em muitos sistemas modernos, é comum usarem-se protocolos de replicação onde se **relaxam** os requisitos de coerência.

- Considere no entanto que a probabilidade de existirem partições é **muito baixa**. Será razão para se reconsiderar esta opção?

Alguns modelos de coerência…

Forte



Fraca

Há uma grande **variedade** de opções

Significado das cores

Unavailable

Sticky Available

Total Available

Exemplos de sistemas replicados que vamos estudar

- Coerência **fraca**
  - *Gossip*
  - *Bayou*

- Coerência **forte**
  - *Primary-backup*
  - Registo distribuído coerente
  - Replicação de máquina de estados

Exemplos adicionais

Sistemas replicados na atualidade

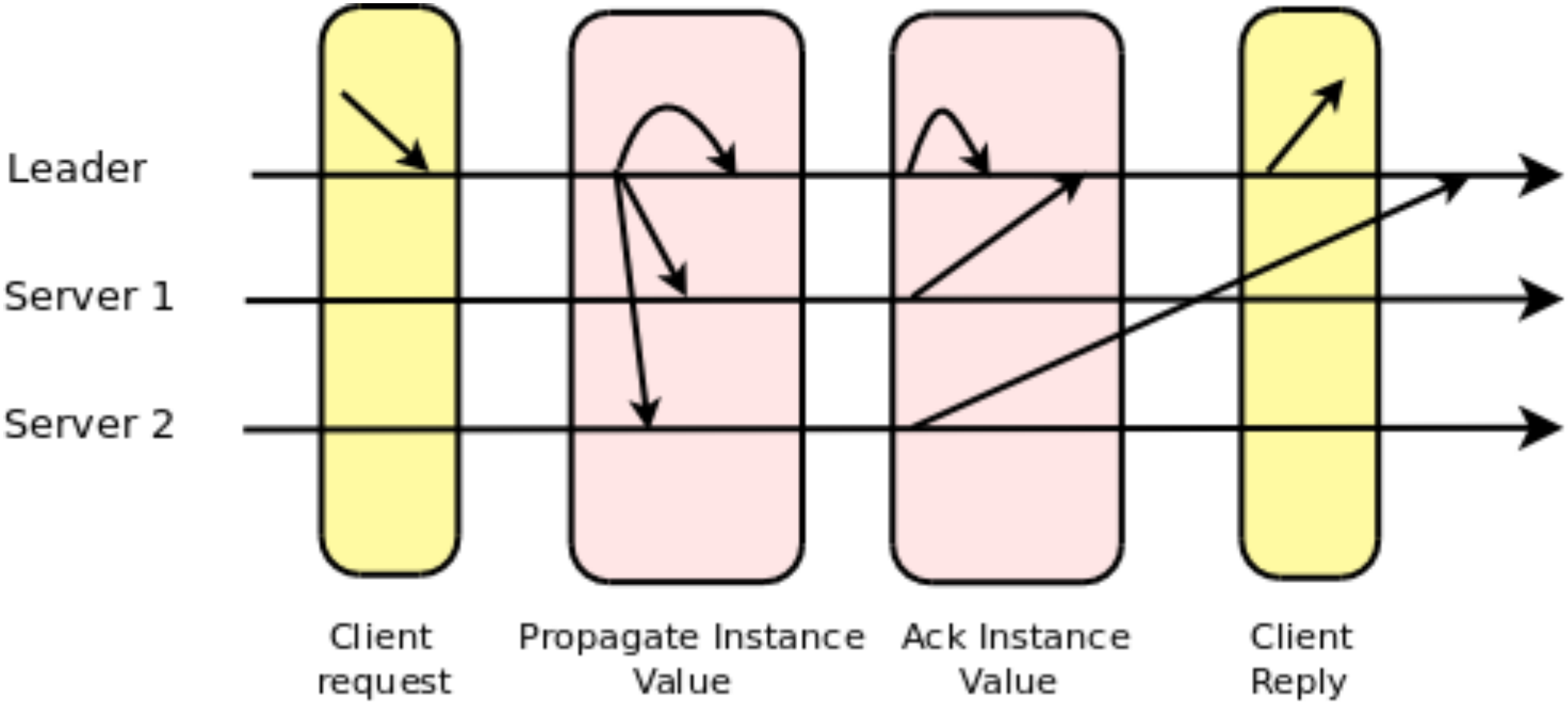
O mundo da coerência forte, hoje

Paxos

- Referência essencial em replicação com **coerência forte**
- Muito tolerante a faltas, mesmo em sistemas assíncronos
- Suporta qualquer operação **determinística** sobre os dados replicados
  - Oferece uma **máquina de estados replicada**

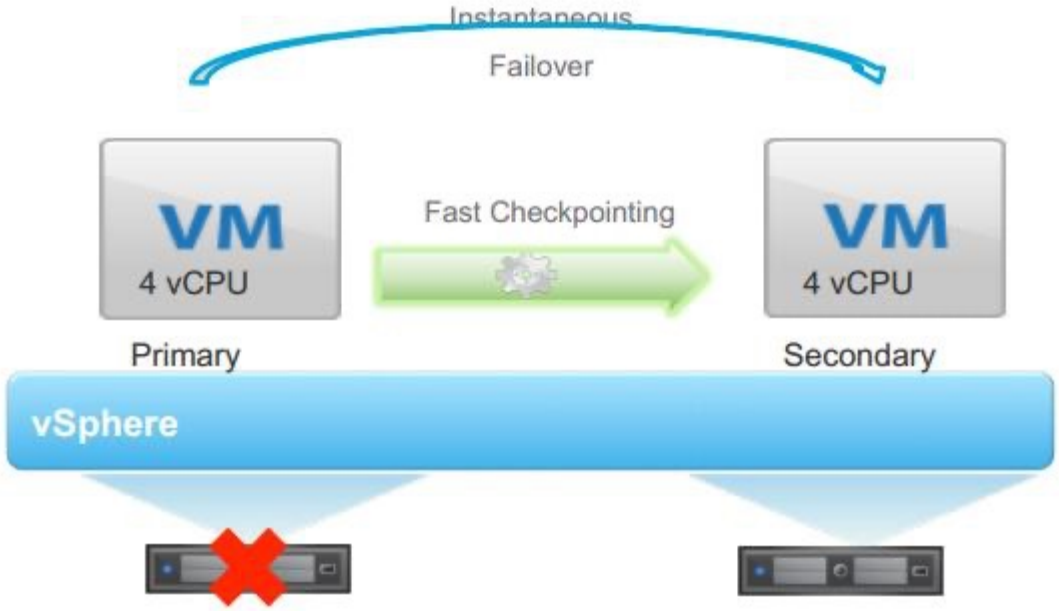
Leslie Lamport

Paxos


Leader
Server 1
Server 2
Client request
Propagate Instance Value
Ack Instance Value
Client Reply

Replicação de máquinas virtuais VMware vSphere

- Permite que servidores num *cluster* passem a ser tolerantes a faltas
  - De forma quase transparente para os programas a correr no servidor
- Recorre a protocolo ***primary-backup***



D. J. Scales et al., "The design of a practical system for fault-tolerant virtual machines", SIGOPS OSR 2010

A tendência para coerência fraca

DynamoDB

- Principal **sistema de armazenamento** usado pelos serviços alojados na Amazon
- Corre sobre dezenas de milhares de servidores espalhados pelo mundo inteiro
  - Pressuposto: a cada momento há sempre alguns servidores em falha

DynamoDB

amazon web services™

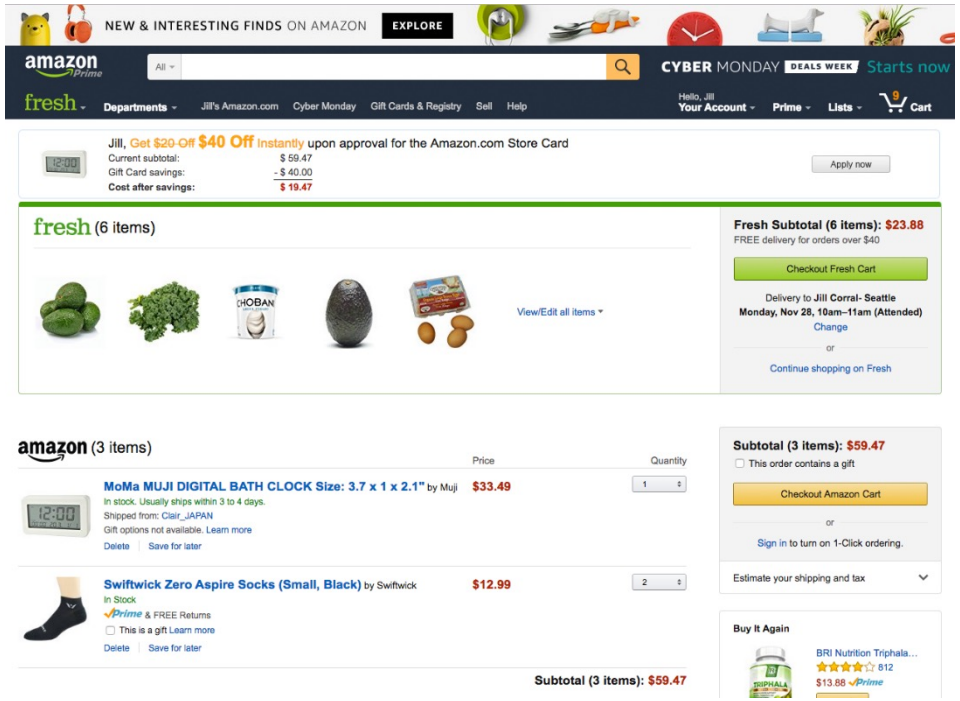
DynamoDB
Como tolerar falhas nesta escala?

- Combina técnicas previamente existentes:
  - Dados particionados e **replicados**
  - Cada objeto mantém *tag* (de versão)
  - Coerência mantida por **protocolo de quóruns**
- …mas **descarta a garantia de coerência forte**
  - Nem suporta transações ACID nem coerência sequencial
  - Escritas entregues, **mais cedo ou mais tarde (*eventualy*)**, atrasos prováveis
  - Usam protocolo de quóruns muito relaxado
- Sistema responde rapidamente mesmo quando está particionado
  - Foco na **disponibilidade**!

DynamoDB
Aplicações aceitam coerência fraca?

- Para manter estruturas de dados cuja **coerência não é crítica**, sim!

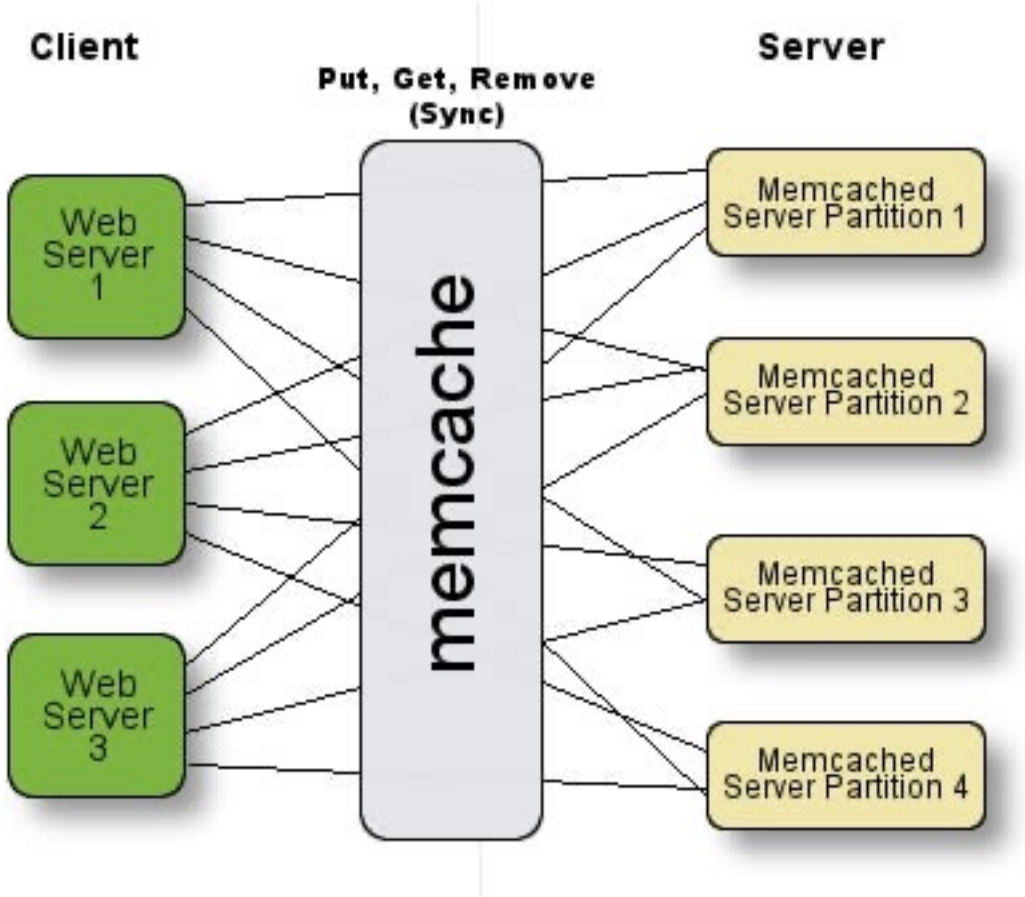
- Exemplos de uso do DynamoDB:
  - Listas de *best sellers*
  - Carrinhos de compras
  - Preferências do cliente
  - Catálogo de produtos
  - Etc.



G. DeCandia et al., "Dynamo: amazon's highly available key-value store", SOSP'07

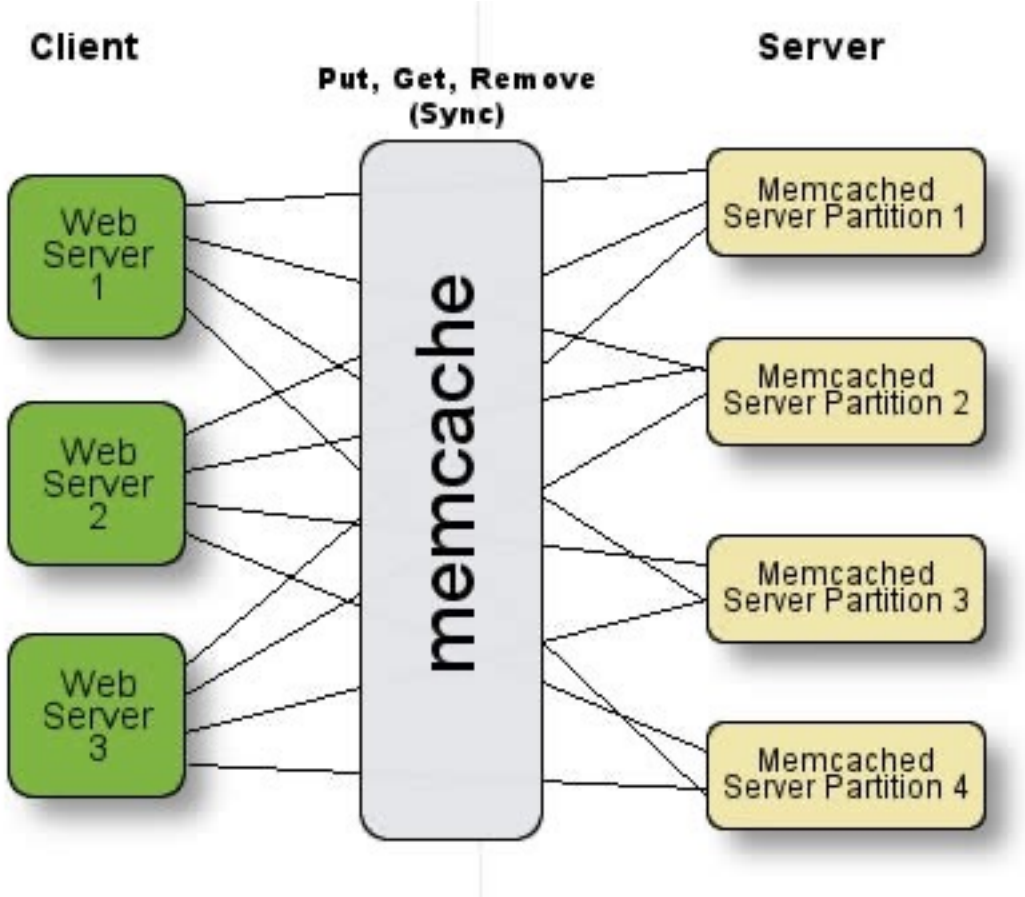
Memcache

- Usado pelo Facebook
- Objetivo: acesso de **alto desempenho e disponibilidade** a dados guardados em fontes diferentes
- Como: *cache* distribuída junto aos consumidores da informação



Memcache:
Como garantir coerência dos dados em *cache*?

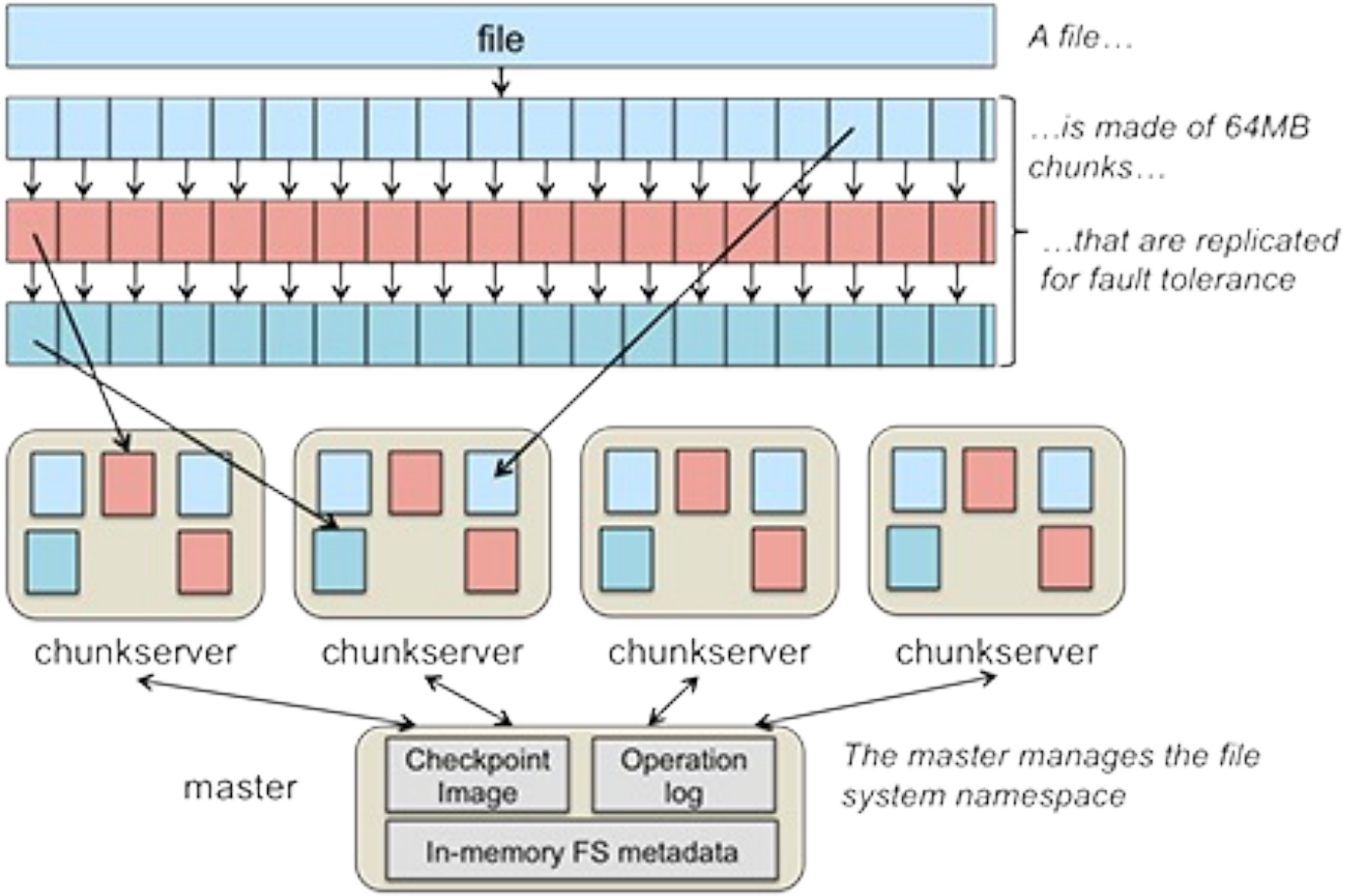
- Suporte limitado a coerência
- Em geral, usado com **coerência fraca**



R. Nishtala et al., "Scaling Memcache at Facebook", NSDI'13

Google File System

- Sistema de ficheiros distribuído para aplicações de **larga escala**
- Feito para **tolerar muitas falhas**, assegurando escalabilidade
- Modelo de coerência **"relaxado"**



S. Ghemawat et al., "The Google File System", SOSP'03

Concluindo…

- Protocolos de replicação são a base para muitas tecnologias fundamentais

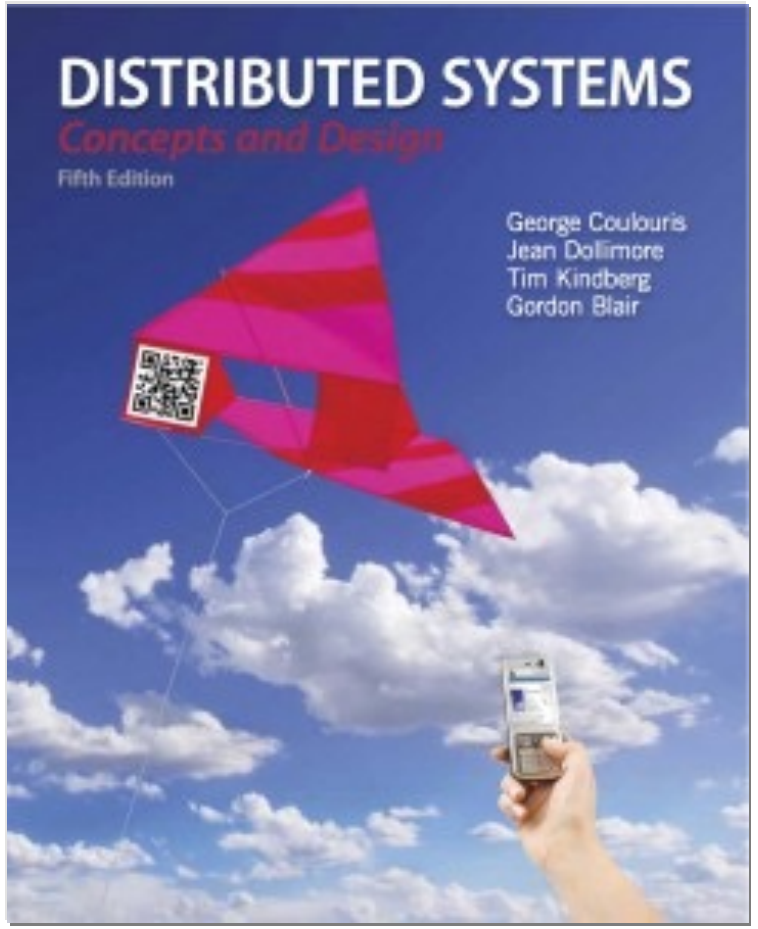
- Teorema **CAP**: “não há almoços grátis”!
- Tendência forte em grandes sistemas à escala planetária: **descartar coerência forte**
  - Nota: uma tendência não descarta a extrema relevância de protocolos de **coerência forte** em muitas situações

- Muitos destas tecnologias estão disponíveis: **explorem**!

Bibliografia recomendada

- Secções 18.2.1 e 18.5.3

- E. Brewer, "CAP Twelve Years Later: How the "Rules" Have Changed", IEEE Computer, 2012



Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

BY WERNER VOGELS

Eventually Consistent

AT THE FOUNDATION of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost-effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and must be accounted for upfront in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these [illegible] be confronted with the [illegible]es of using replication techniques inside the services.

One of the ways in which this manifests itself is in the type of data consistency that is provided, particularly when many widespread distributed systems provide an *eventual consistency* model in the context of data replication. When designing these large-scale systems at Amazon, we use a set of guiding principles and abstractions related to large-scale data replication and focus on the trade-offs between high availability and data consistency. Here, I present some of the relevant background that has informed our approach to delivering reliable distributed systems that must operate on a global scale. (An earlier version of this article appeared as a posting on the "All Things Distributed" Weblog and was greatly improved with the help of its readers.)

Historical Perspective

In an ideal world there would be only one consistency model: when an update is made all observers would see that update. The first time this surfaced as difficult to achieve was in the database systems of the late 1970s. The best "period piece" on this topic is "Notes on Distributed Databases" by Bruce Lindsay et al.[5] It lays out the fundamental principles for database replication and discusses a number of techniques that deal with achieving consistency. Many of these techniques try to achieve *distribution transparency*—that is, to the user of the system it appears as if there is only one system instead of a number of collaborating systems. Many systems during this time took the approach that it was better to fail the complete system than to break this transparency.[2]

In the mid-1990s, with the rise of larger Internet systems, these practices were revisited. At that time people began to consider the idea that availability was perhaps the most impor-

Pesquisar

Início

Hiring Developers Is Hard - But it doesn't have to b

Conectar

Werner Vogels · 2º

VP & CTO at Amazon.com

Grande Seattle e Região, Estados Unidos · + de 500 conexões ·

Informações de contato

Experiência

Amazon.com
15 anos 8 meses

VP & CTO
jan. de 2005 – o momento · 15 anos 4 meses

Director of Systems Research
set. de 2004 – jan. de 2005 · 5 meses

Research Scientist
Cornell University
mai. de 1994 – set. de 2004 · 10 anos 5 meses

Senior Research Engineer
INESC
fev. de 1991 – abr. de 1994 · 3 anos 3 meses

Formação acadêmica

Vrije Universiteit, Amsterdam
PhD, Computer Science

Scalable Cluster Technologies for Mission-Critical Enterprise Computing