DTCC2013

CAP:理论与实践

童家旺

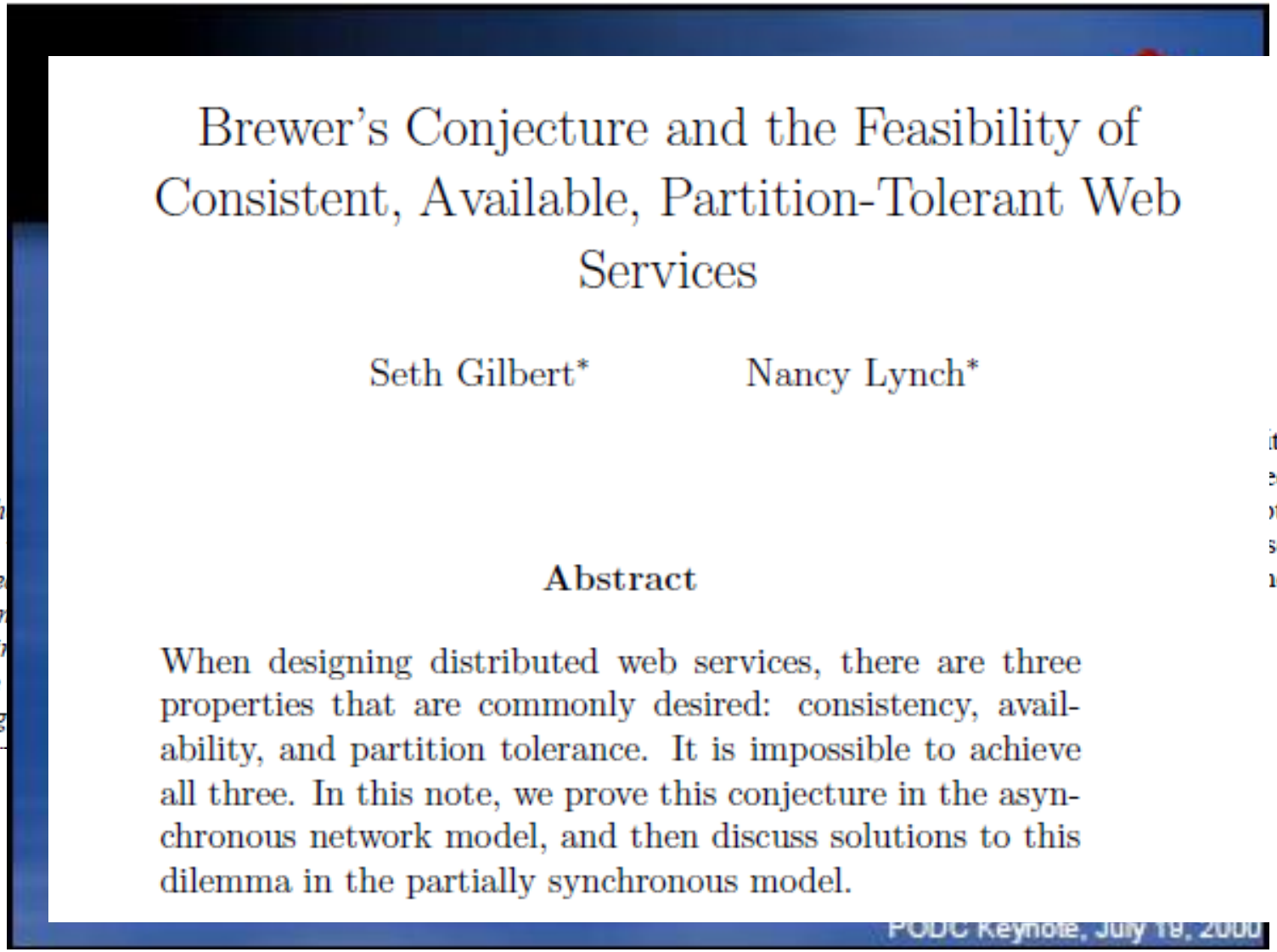
http://www.dbthink.com/

Weibo: jametong

内容大纲

- CAP的历史沿革
- CAP原理介绍
- CAP，Pick Two？
- CAP权衡中的经济考量
- CAP与ACID的关系
- 分区只是另一段Code Path
- CAP在实践中的应用
- 参考资料

CAP原理的历史



Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert* Nancy Lynch*

Abstract

When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three. In this note, we prove this conjecture in the asynchronous network model, and then discuss solutions to this dilemma in the partially synchronous model.

PODC Keynote, July 19, 2000

CAP流行的几大推手

BASE: An Acid Alternative view issue

by Dan Pritchett | May 1, 2008

Topic: File Systems and Storage

Like 27

In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability.

DAN PRITCHETT, EBAY

Web applications have grown in popularity over the past decade. Whether you are building an application for end users or application developers (i.e., services), your hope is most likely that your application will find broad adoption—and with broad adoption will come transactional growth. If your application relies upon persistence, then data storage will probably become your bottleneck.

There are two strategies for scaling any application. The first, and by far the easiest, is vertical scaling: moving the application to larger computers. Vertical scaling works reasonably well for data but has

CAP Theorem

- 对于共享的数据系统，仅能**同时**满足2项:
  - **C**onsistency (多节点看到数据的**单一/同一**副本)
    - Full Consistency ?
    - Casual Consistency ?
    - Timeline Consistency ?
    - Eventual Consistency ?
  - **A**vailability (系统总是可以执行**变更**操作)
    - 牺牲10秒钟的A ? 牺牲10分钟的A ?
  - **P**artition Tolerance
- 在**广域网**的情况下，分区不可避免
  - => consistency vs. availability

CAP ，Pick Two?

CAP原理

07:27

CAP原理(CAP Theorem)

在足球比赛里，一个球员在一场比赛中进三个球，称之为帽子戏法(Hat-trick)。在分布式数据系统中，也有一个帽子原理(CAP Theorem)，不过此帽子非彼帽子。CAP原理中，有三个要素：

- 一致性(**C**onsistency)
- 可用性(**A**vailability)
- 分区容忍性(**P**artition tolerance)

CAP原理指的是，这三个要素最多只能同时实现两点，不可能三者兼顾。因此在进行分布式架构设计时，必须做出取舍。而**对于分布式数据系统，分区容忍性是基本要求**，否则就失去了价值。因此设计分布式数据系统，就是在一致性和可用性之间取一个平衡。对于大多数web应用，其实并不需要强一致性，因此牺牲一致性而换取高可用性，是目前多数分布式数据库产品的方向。

CAP原理的意思是，一个分布式系统不能同时满足一致性，可用性和分区容错性这三个需求，最多只能同时满足两个。

面对P，真的必须牺牲一致性吗？

- 新浪微
- 淘宝的
- 1230
- 比特币
- 中行信

Reliability & $$

数据状态机的分类

- 何谓状态机
  - 简单的理解是，计算机中会发生变化的数据都是状态机，这个数据的值不同可能会带来不同的后果。
  - 分类：按照三个维度：时间、信息含金量、变更频率

持续时间	信息含金量	变更频繁度	例子
瞬时	高	少	Shopping Card Session（分
瞬时	低	少	Login Cookie（分）
中等时长	高	少	Ecommerce Billing(天)
中等时长	中	少	Product Catalog(年)
中等时长	高	多	Flight/Train Inventory (月)
无限时长	中	少	User Profile（年)
无限时长	高	多	Bank Account Balance（年)

其实，CAP并没有声明...

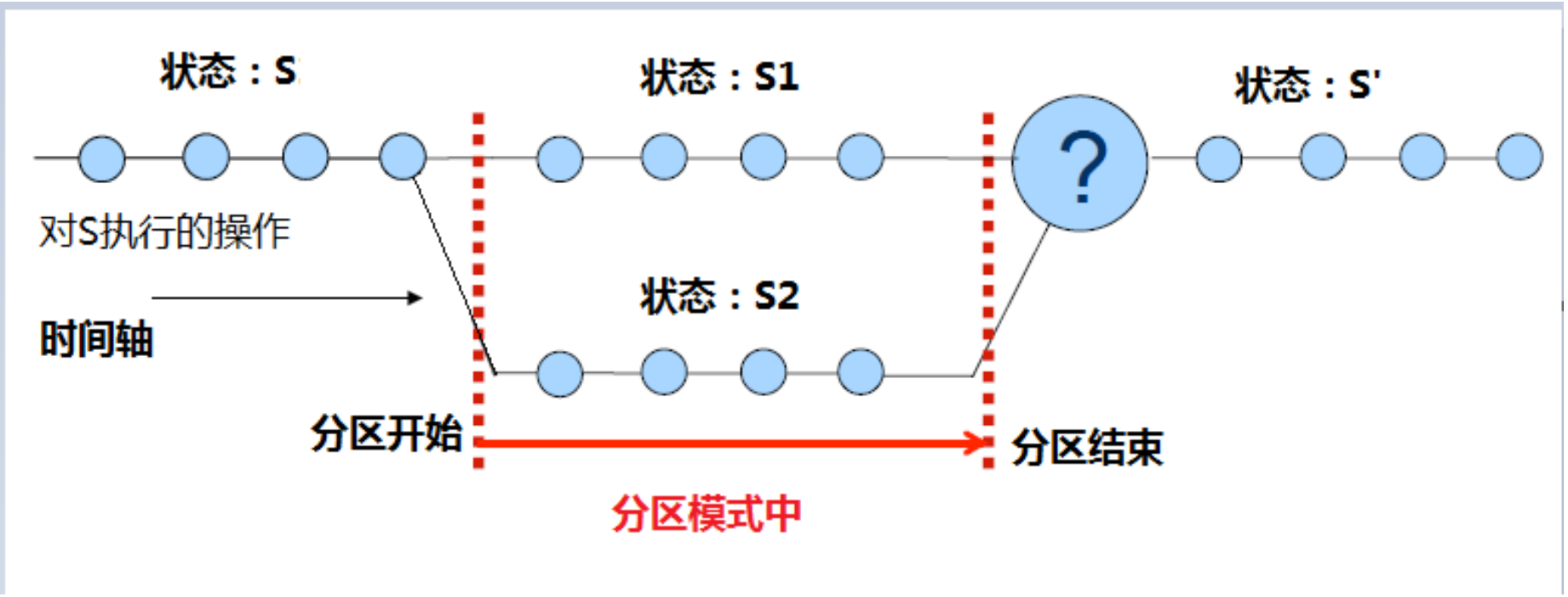
- 放弃一致性
  - 不一致应该仅仅是个例外
  - 很多系统牺牲的内容**远远超过必要**！
- 放弃事务 (ACID)
  - 需要调整C与I的预期 (仅仅)
- 不要使用SQL
  - 很多NOSQL系统中也开始支持SQL
  - 声明性语言(SQL)与CAP配合良好

CAP与ACID

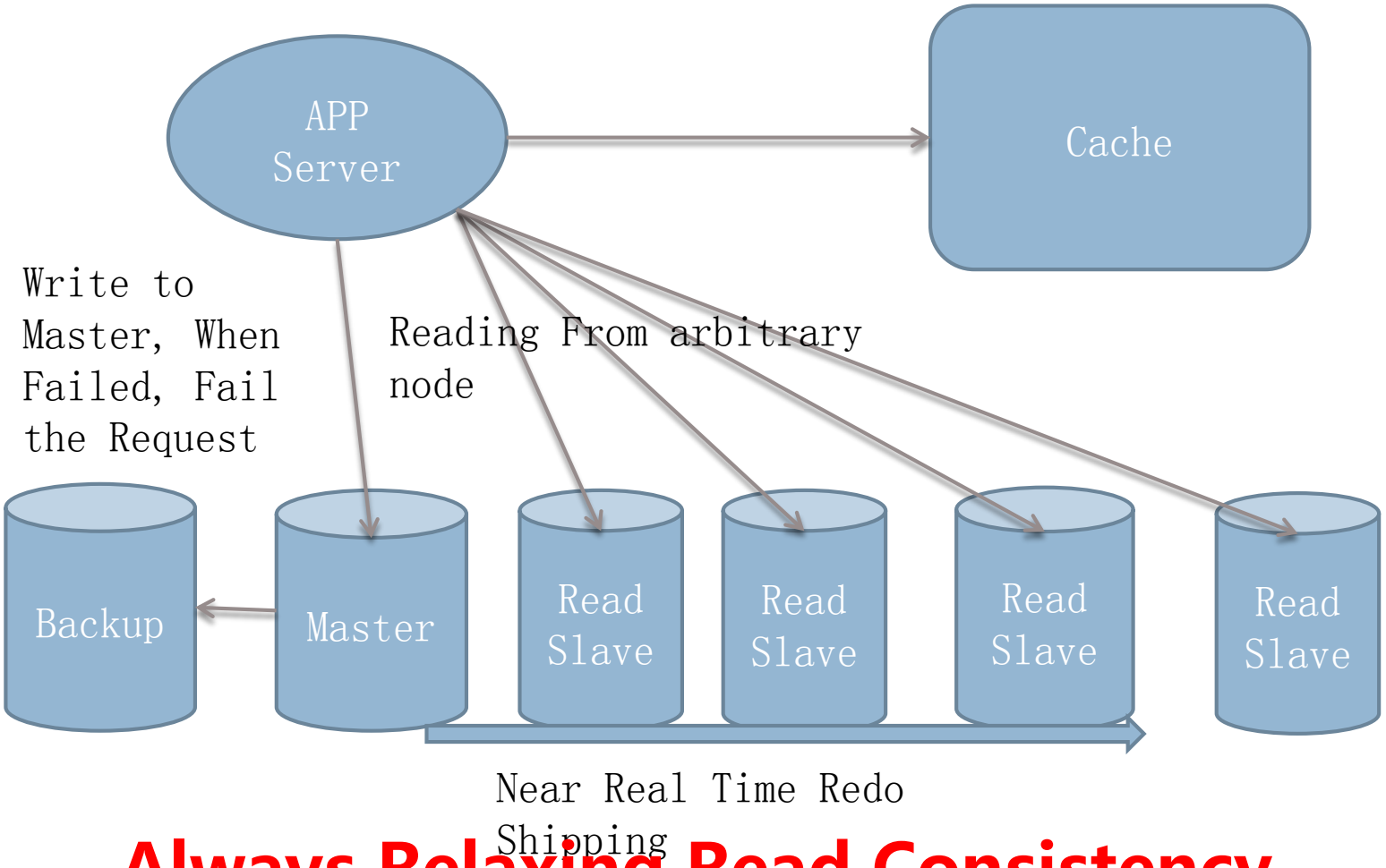
- 当系统没有Partition时：
  - 支持Full ACID
- 当系统出现Partition时：
  - **Atomic**:不同的分区还是应该保持Atomic
  - **Consistent**:临时违背（如：没有重复？）
  - **Isolation**:临时牺牲隔离性
  - **Durable**:永远不该牺牲它（需要保留它用）

像程序员一样思考

分区只是个不同的代码路径

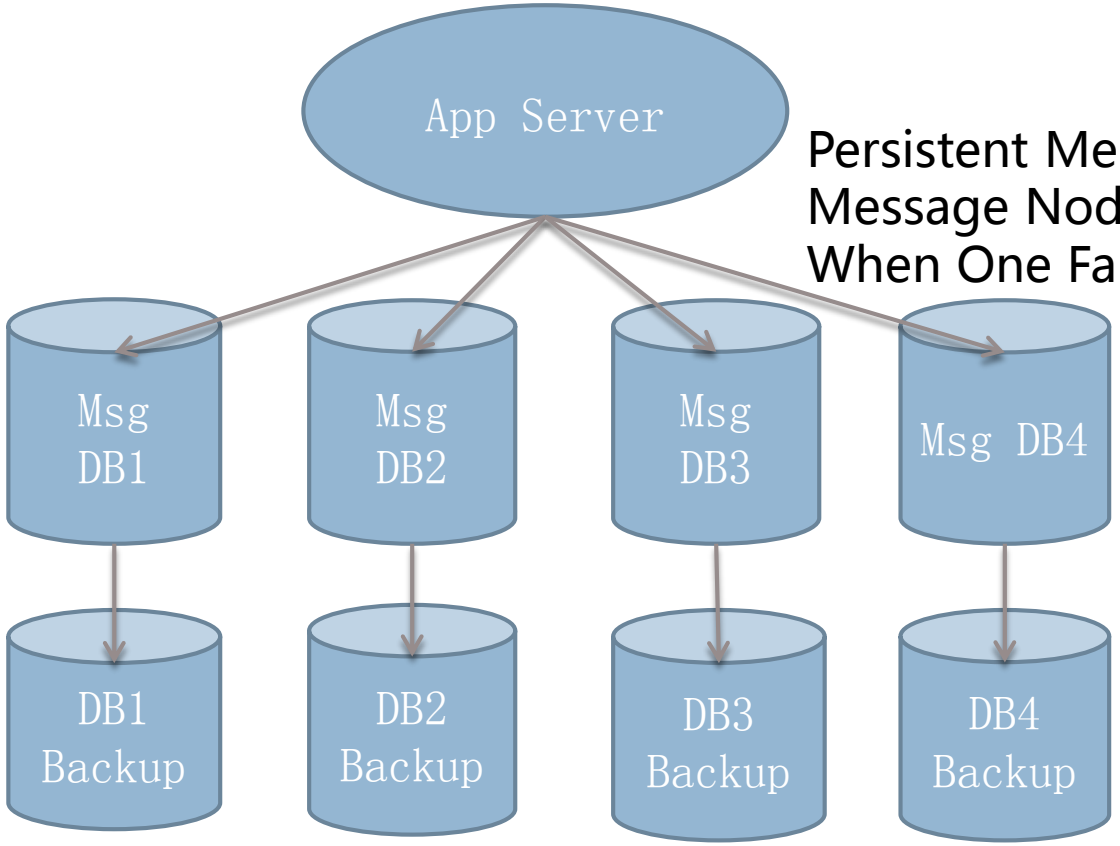


User Profile处理



Always Relaxing Read Consistency

Message Processing

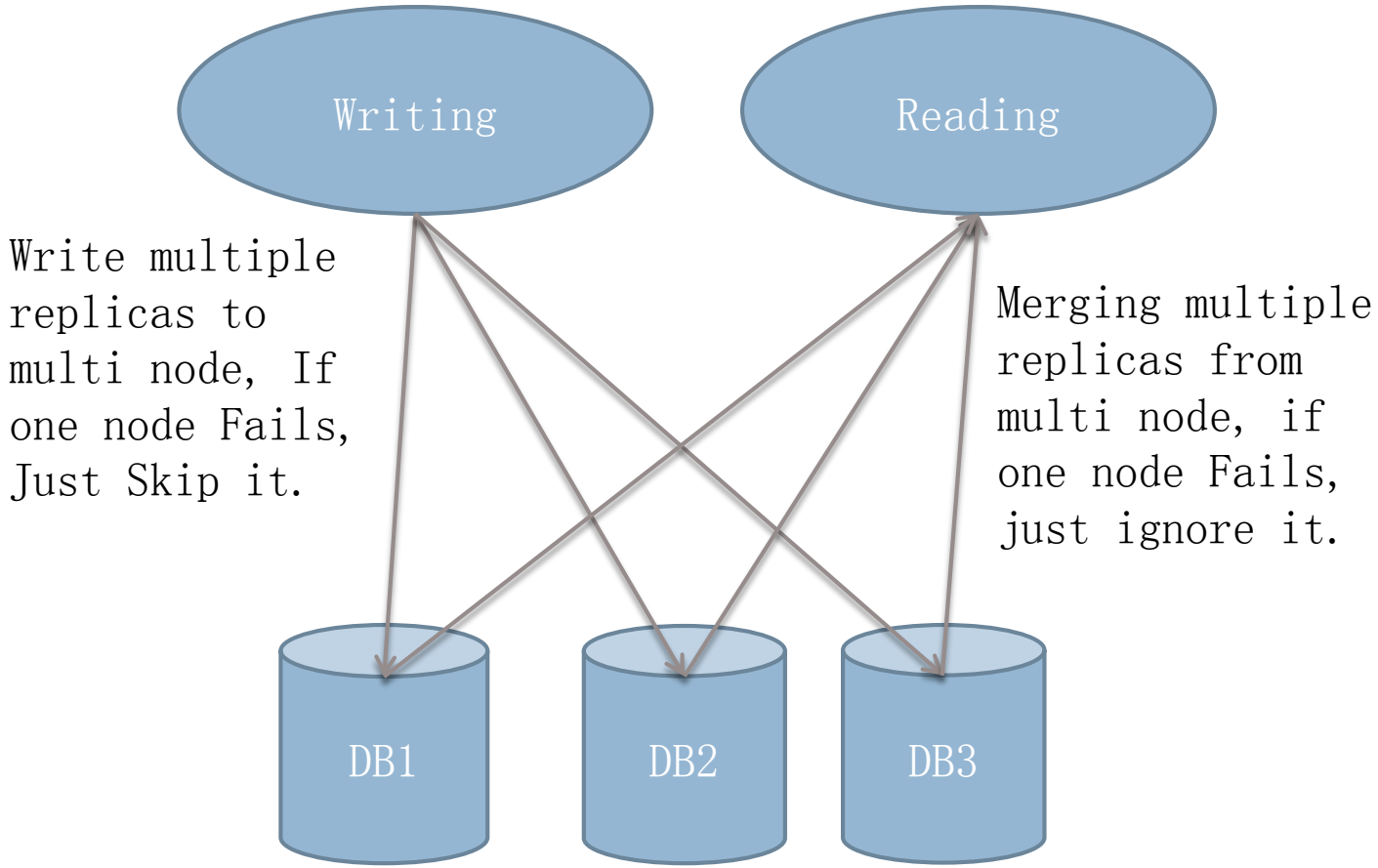


Persistent Message in arbitrary Message Node
When One Failed Just Pick Next

When One Node Failed, Just Delay the Message Sending Persistent in that Node, Aka, Sacrificing Availability

Shopping Cart

DTCC2013



Relaxing Consistency When Partitioned

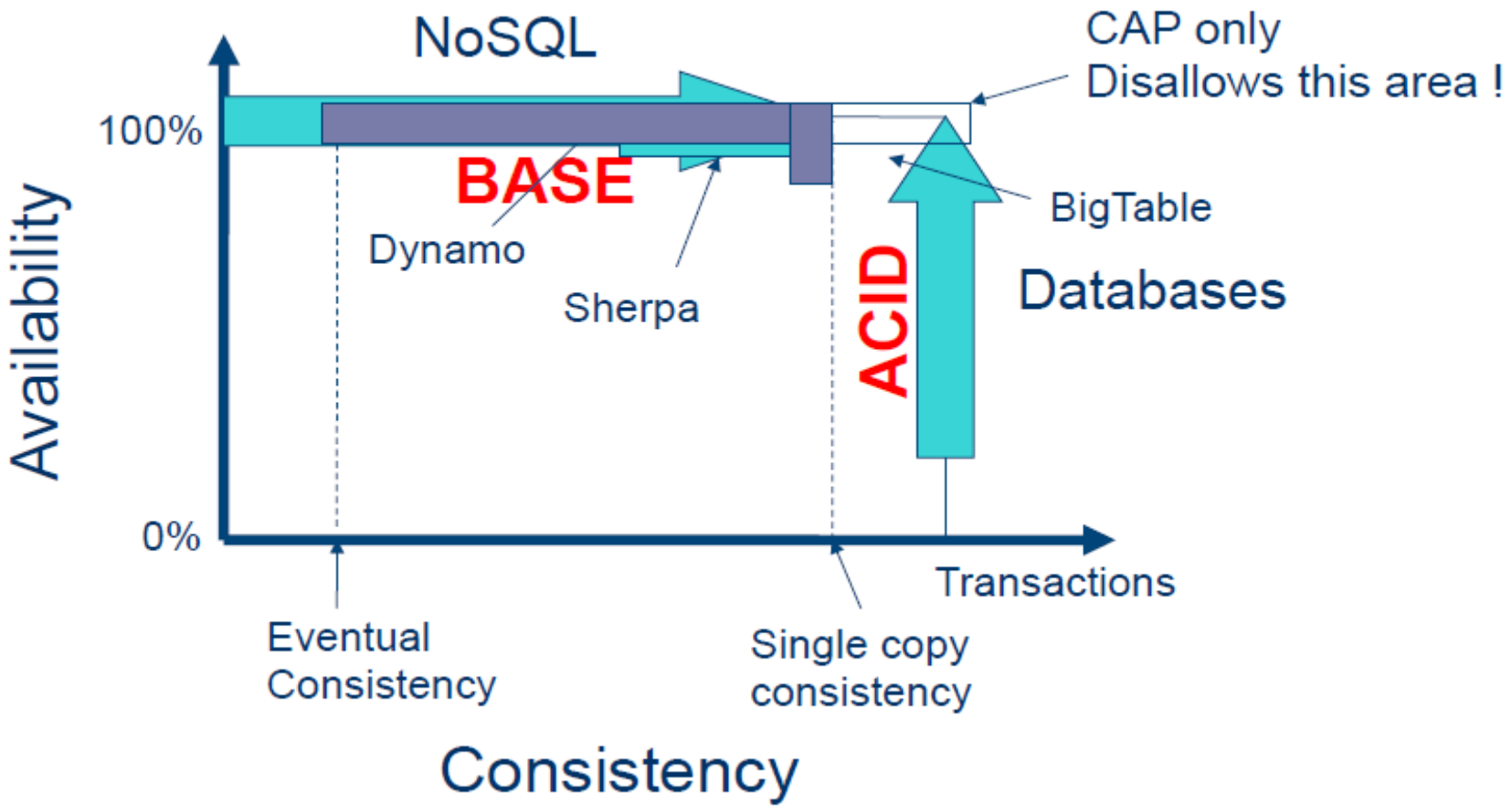
银行ATM机

- When Partitioned
  - 只允许低于200$的提现操作
  - 在本地记录操作的日志
- When Partition Recovered
  - Reapply本地日志
  - 如果有透支，通过外部商业流程进行补偿处理。
- **Sacrifice Some A，Relax Some Consistency**

航空订票系统

- 超卖是一种商业选择
  - 仅限经济舱
- 如果出现超卖
  - 为用户做免费升舱
- **When Partition**
  - **Relaxing Consistency**
  - **一定的补偿机制**

CAP的实际含义


NoSQL
CAP only
Disallows this area !
100%
BASE
Dynamo
Sherpa
BigTable
Databases
ACID
Availability
0%
Transactions
Eventual
Consistency
Single copy
consistency
Consistency

总结

- CAP的实际效果
  - 探索适合不同应用的一致性与可用性平衡
- 在没有分区发生时
  - 可以同时满足C与A，以及完整的ACID事务支持
  - 可以选择牺牲一定的C，获得更好的性能与扩展性
- 分区发生时，选择A（集中关注分区的恢复）
  - 需要有分区开始前、进行中、恢复后的处理策略
  - 应用合适的补偿处理机制

参考资料

- NoSQL: Past, Present, Future
  - By Eric Brewer
- CAP Twelve Years Later: How the "Rules" Have Changed
  - By Eric Brewer
- Towards Robust Distributed Systems
  - By Eric Brewer
- *Dynamo: Amazon's* Highly Available Key-value Store
  - By Giuseppe DeCandia, Werner Vogels etc..

Any Questions?