大数据环境下数据库技术的变革
—事务与并发控制

李战怀
西北工业大学
2013-08-16

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

提纲

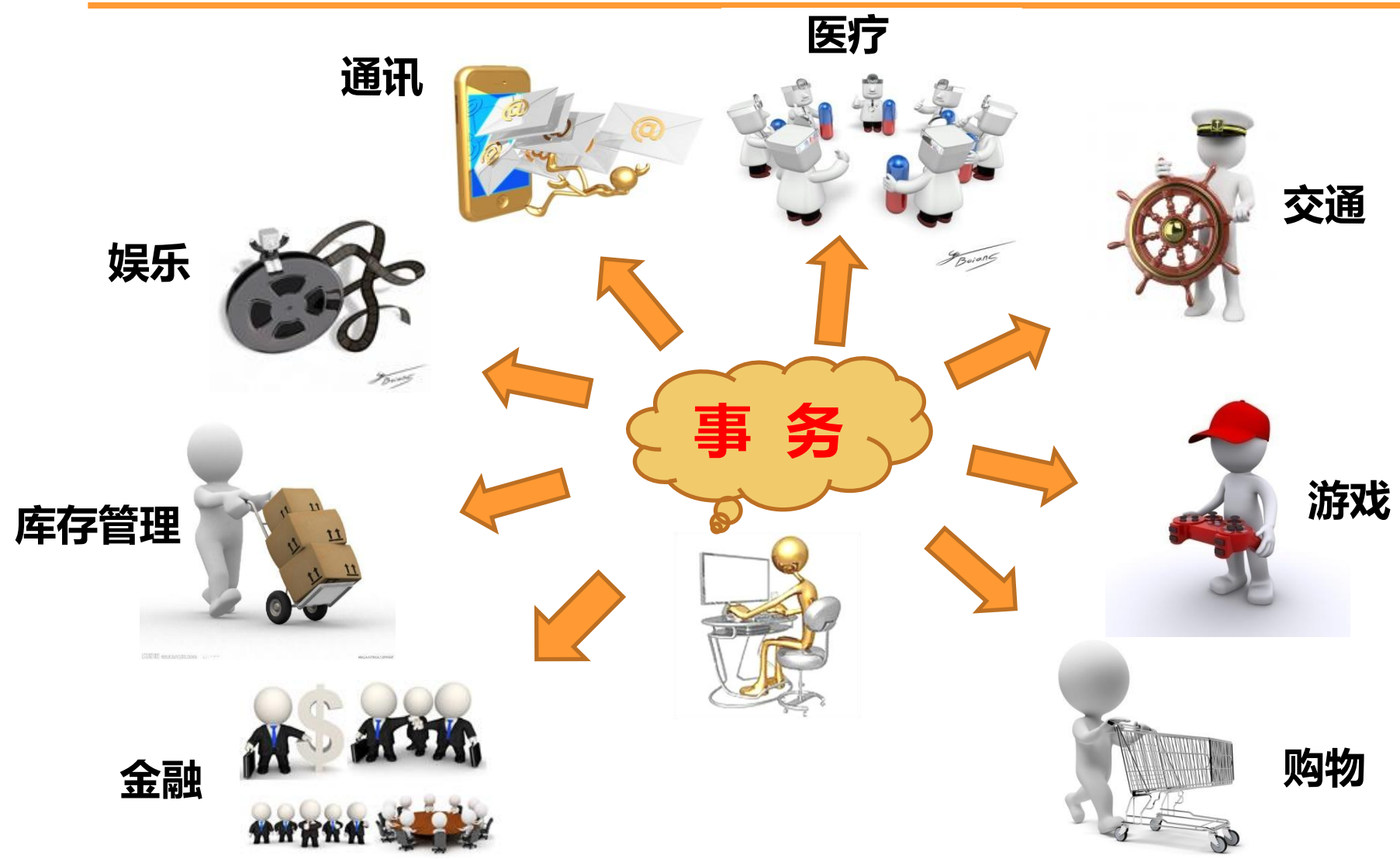
一 • 事务—历史的脉络

二 • 环境—酝酿变革

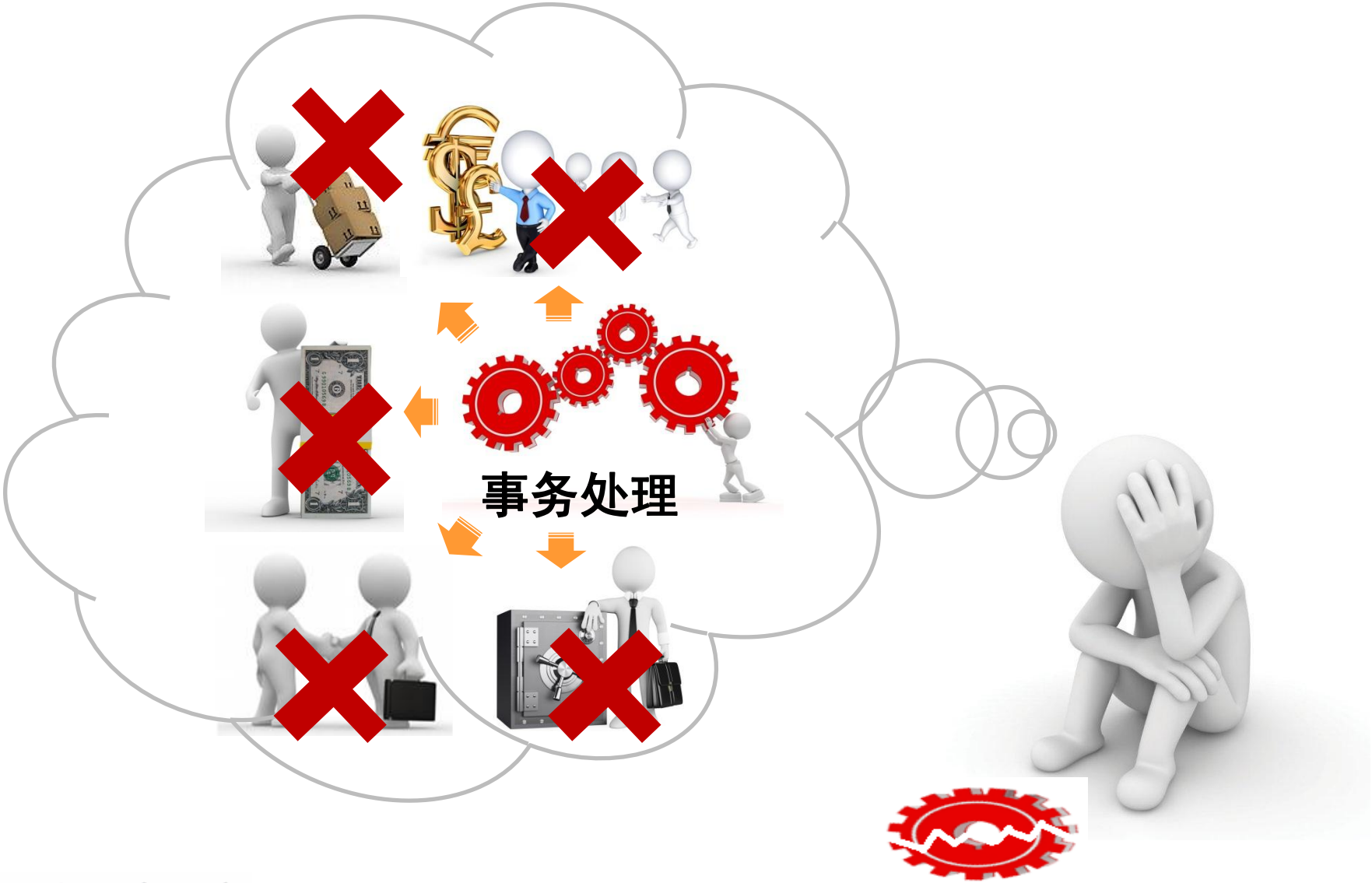
三 • 技术—承载发展

四 • 未来—顺势而为

事务无处不在



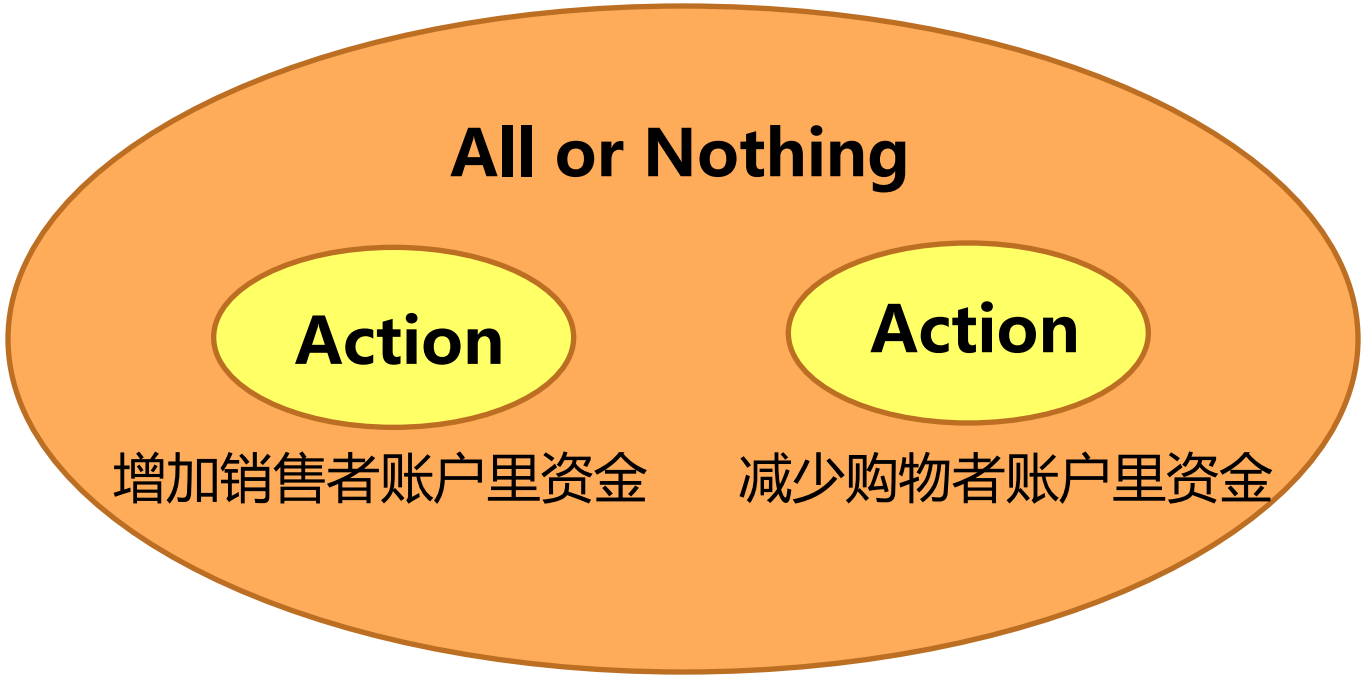
事务的重要性



事务的概念

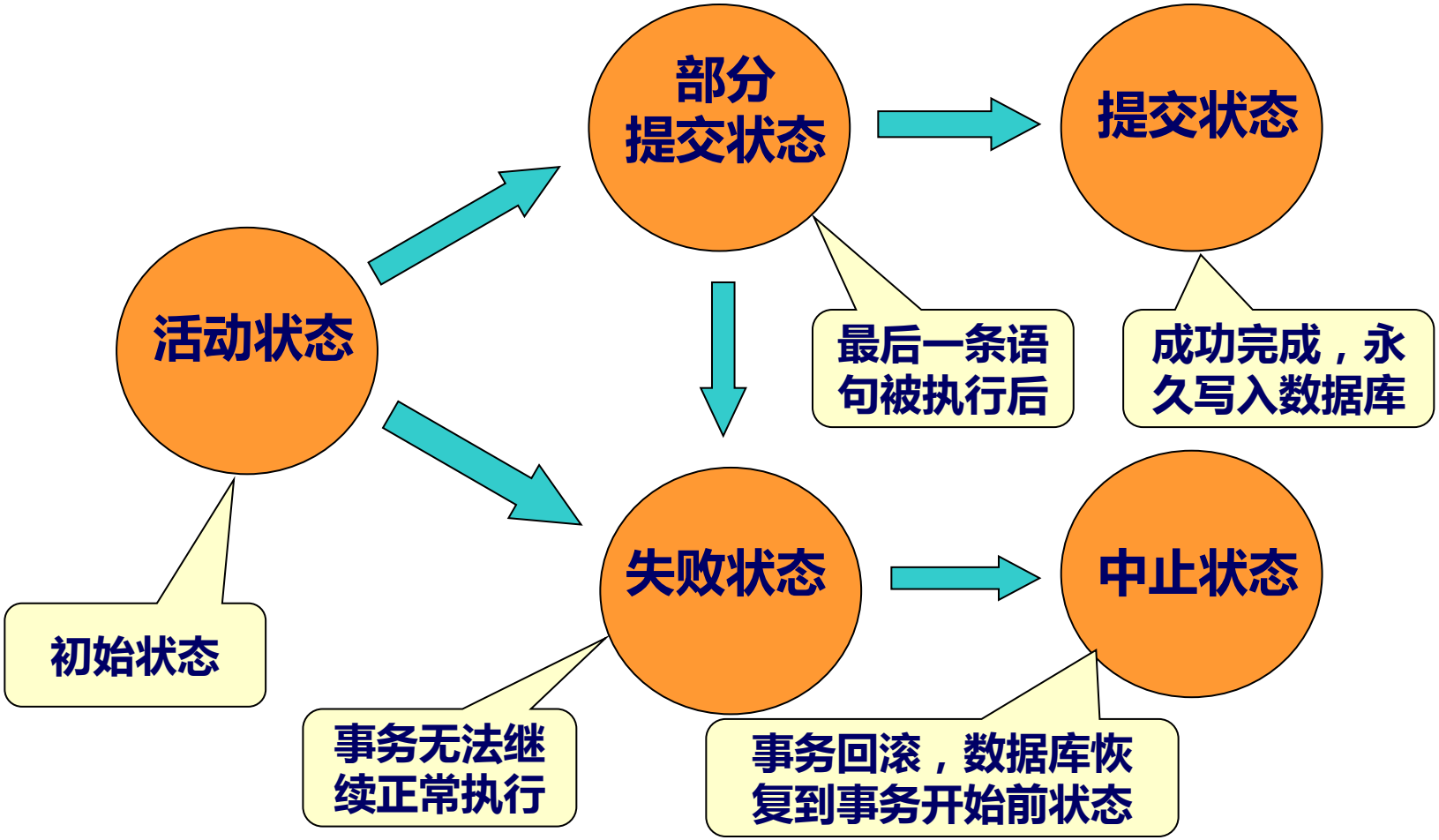
- **事务的基本概念：**

  指作为单个逻辑工作单元执行的一系列操作。一个逻辑工作单元要成为事务，必须满足ACID：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。



购买商品的事务描述

事务的概念—事务运行状态



并发操作

- **事务ACID性质可能遭到破坏的因素有：**
  - 多个事务并发运行时，不同事务的操作交叉执行；
  - 事务在运行过程中被强行终止。

- **并发操作带来的数据不一致包括：**
  - 丢失修改
  - 不可重复读
  - 读“脏”数据

并发控制

- **概念：**为保持多事务并发时事务的隔离性，系统必须提供一系列的机制来控制并发事务间的相互作用，这些机制就是并发控制机制

- **并发控制机制的任务**
  - 保证事务的隔离性和一致性
  - 对并发操作进行正确的调度

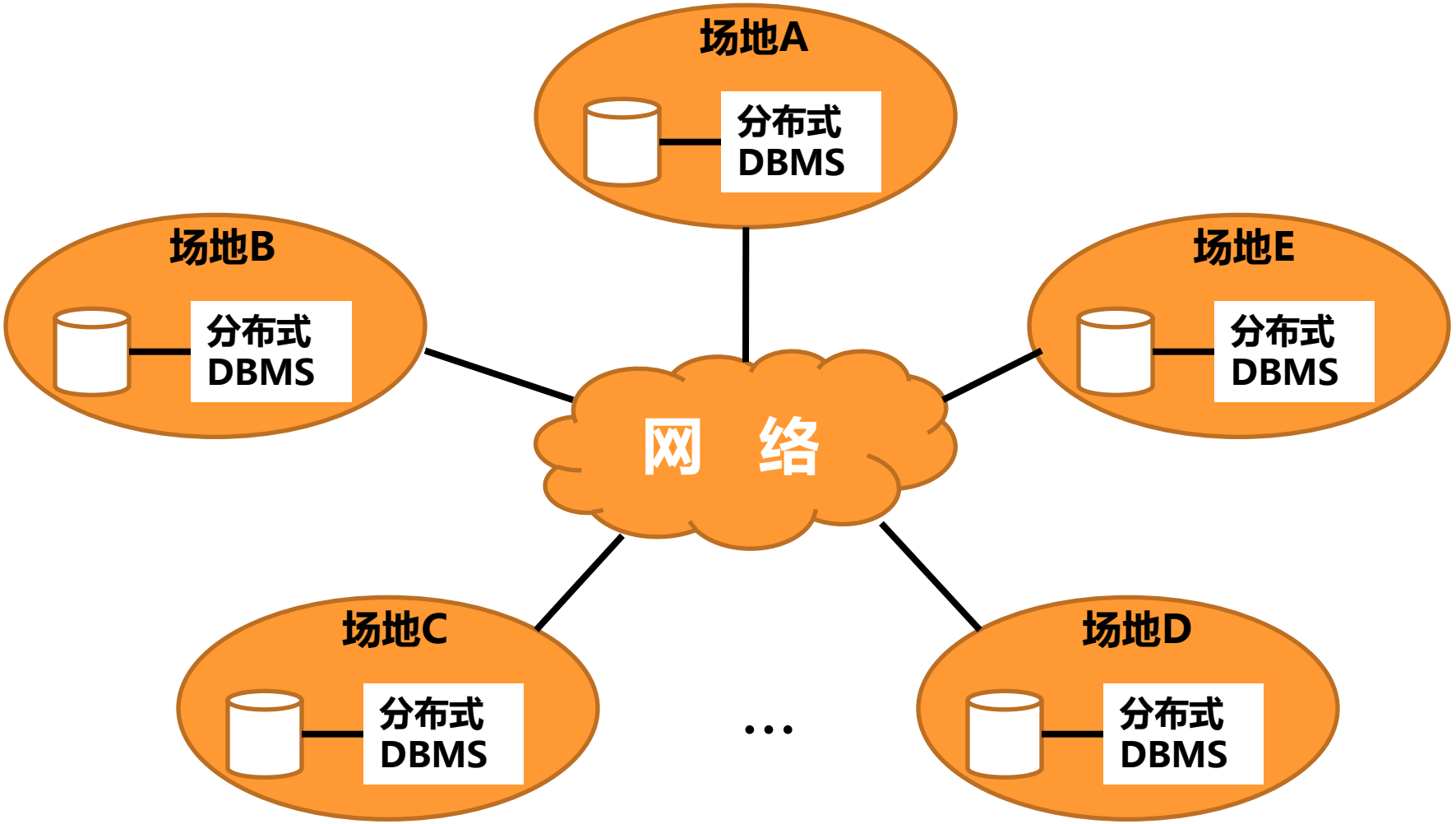
- **并发控制机制**
  - 封锁协议：两阶段封锁
  - 时间戳排序机制
  - 有效性检查技术
  - 多版本机制

事务故障

- **事务故障种类**
  - 逻辑错误、系统错误
- **事务故障种类**
  - 事务故障的恢复是由系统自动完成的，对用户透明
- **系统的恢复步骤：**

  1. 反向扫描日志文件，查找该事务的更新操作
  2. 对该事务的更新操作执行逆操作
  3. 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理，直至读到该事务的开始标记，事务故障恢复就完成

分布式数据库系统

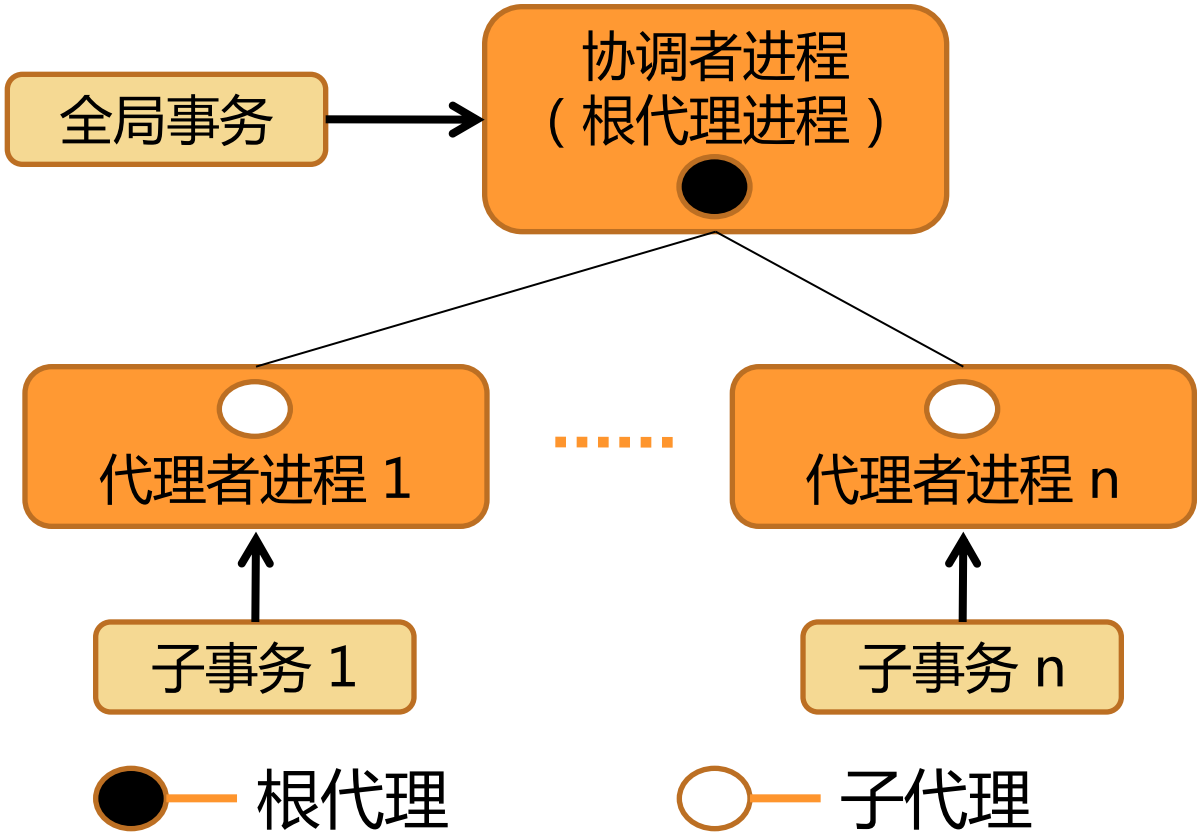


分布式数据库系统结构

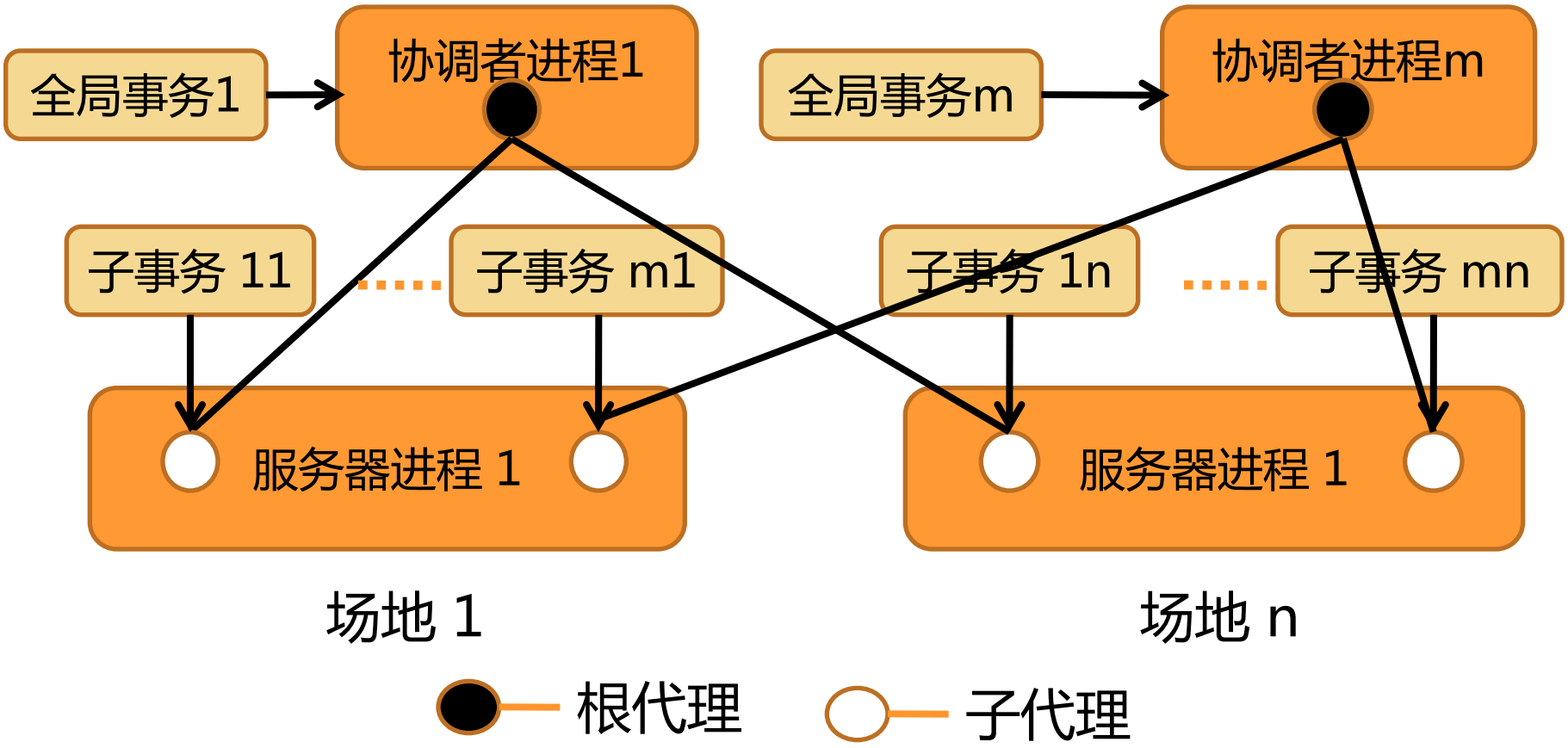
分布式事务

- **分布式事务的一些概念**
  - 局部事务：仅访问和更新一个局部数据库中数据的事务
  - 全局事务：访问、更新多个局部数据库中数据的事务，也称为分布式事务。通常由一个主（父）事务和在不同站点上执行的子事务（局部事务）构成，分布式事务具有ACID特性
- **分布式事务在结构上分为**
  - 平面事务、嵌套事务
- **分布式事务实现模型：**
  - 进程模型和服务器模型

分布式事务实现模型—进程模型



分布式事务实现模型—服务器模型



分布式事务的一致性—Paxos算法

Google Chubby 的作者Mike Burrows： “There is only one consensus protocol, and that's Paxos”-all other approaches are just broken versions of Paxos

- Lamport提出的Paxos是一种基于消息传递的一致性算法，主要解决分布式环境下的“一致性”问题

- 在不同的上下文中“一致性”有不同的解读
  ① **数据库领域**：强调系统中所有的数据的状态一致
  ② **NoSQL领域**：强调读写一致性（能读到最后写入的数据）
  ③ **状态机**：强调在初始状态一致的状态机上执行相同的序列操作后，每个状态机的状态须保持一致，即顺序一致性

- Paxos算法中的“一致性”对应第③种含义

Lamport, The part-time parliament,1998、Paxos made simple, 2001.

分布式事务的一致性—Paxos算法(续)

- **Paxos算法在分布式存储系统中的应用**
  - 分布式存储系统的一致性
    - 采用多副本（高容错）进行可更改数据的存储
    - 多个副本须执行相同的更新操作序列[op_1, op_2, …, op_m]
  - Paxos算法用来确定一个变量(op_i)的取值
    - 取值可以是任意binary
    - 取值具有不可变性和可读取性
    - 以此保证副本执行完全相同的操作序列进而维护一致性
  - Google的Chubby，Megastore，Spanner，以及Yahoo的Zookeeper中都采用了此算法对副本更新序列达成一致
  - Paxos算法不仅可在分布式系统中应用，凡是多个过程需要达成某种一致性的情况都可以使用Paxos算法

Lamport, The part-time parliament,1998、Paxos made simple, 2001.

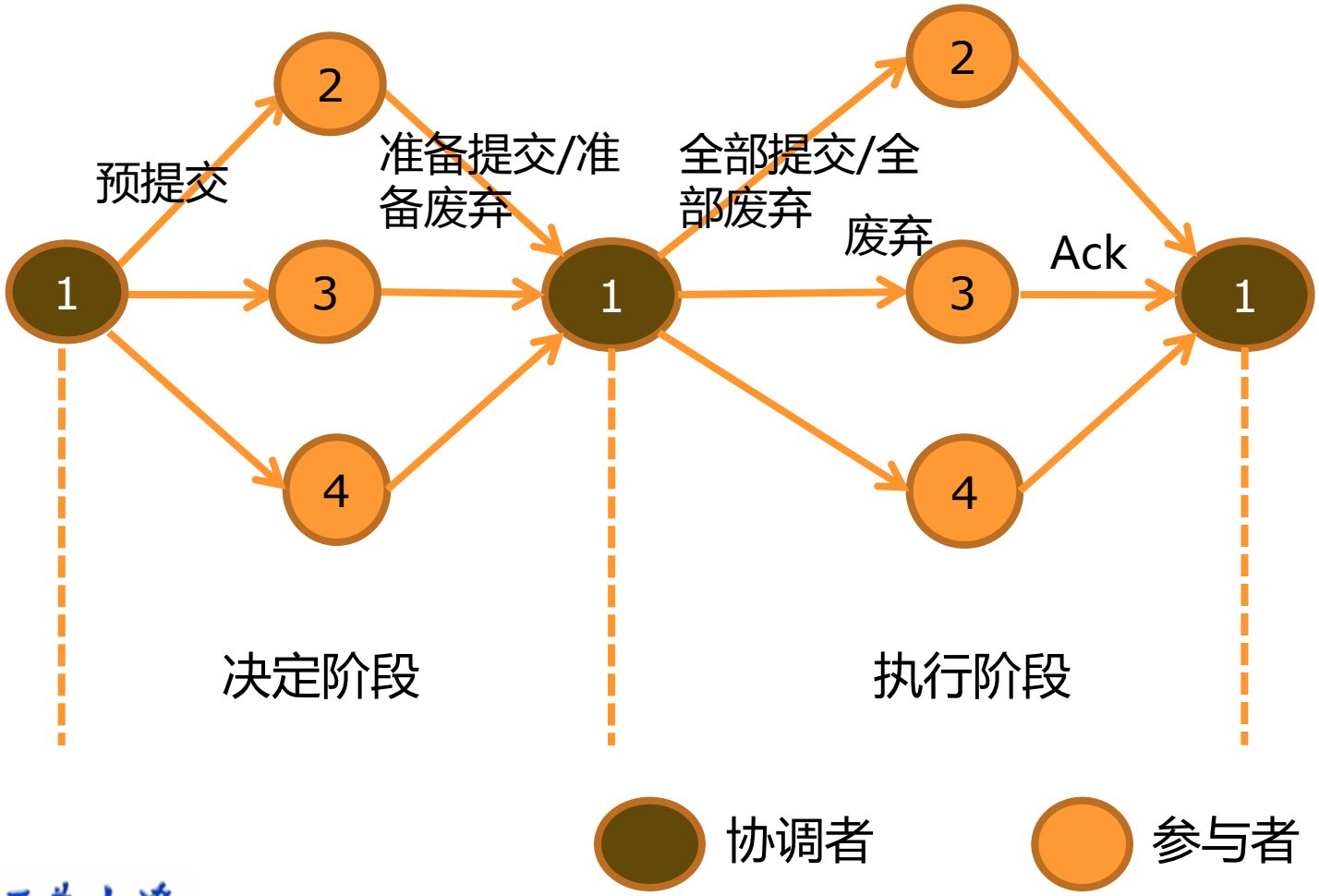
分布式事务的一致性—提交协议

- **为保证原子性，执行事务的site必须在最终执行结果上保持一致，这种特性依赖于提交协议**

- **提交协议**
  - 两段提交协议（2 Phase Commit，2PC）
  - 三段提交协议（3 Phase Commit，3PC）

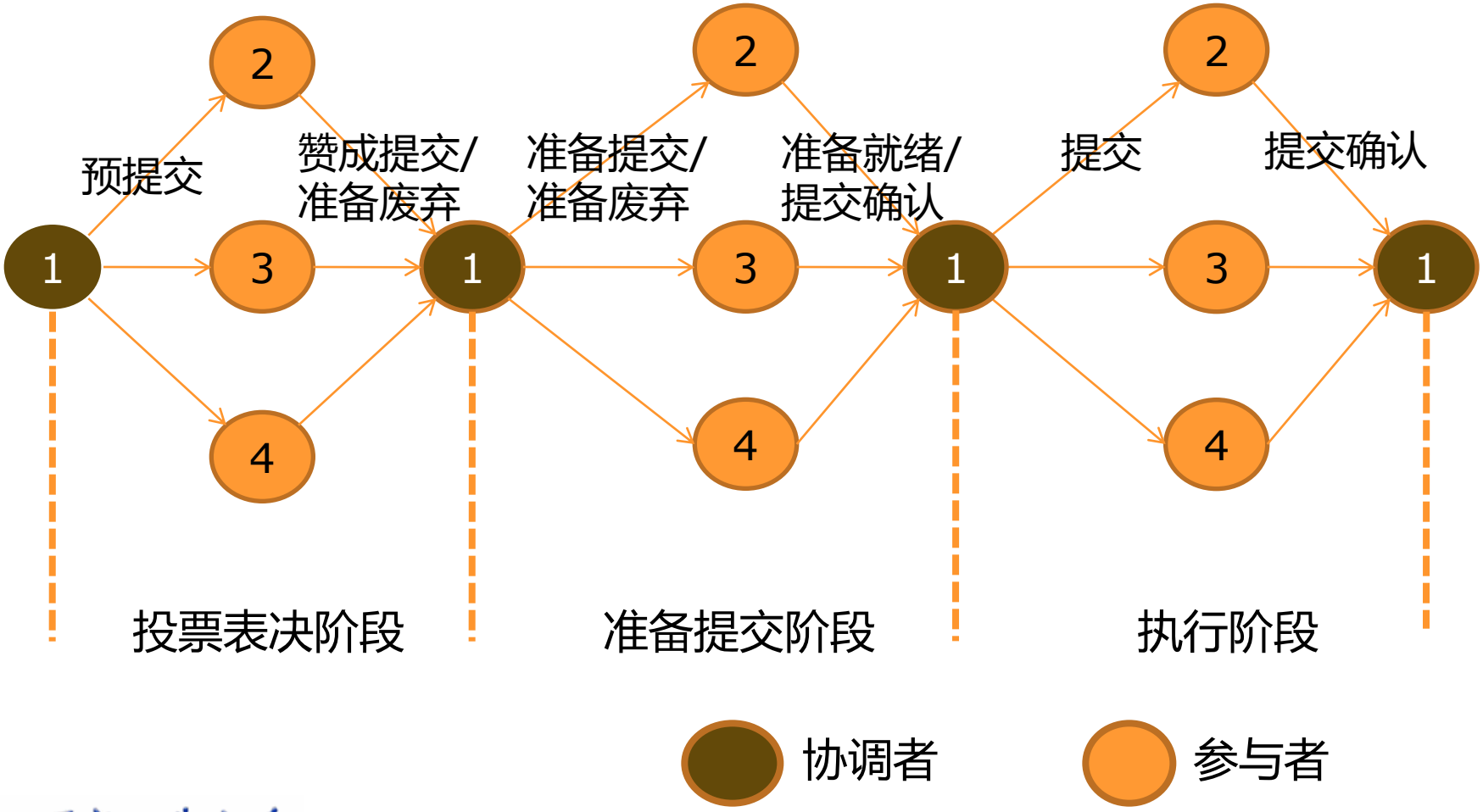
分布式事务协议

- 两段提交协议（2 Phase Commit，2PC）



分布式事务协议

- 三段提交协议（3 Phase Commit，3PC）

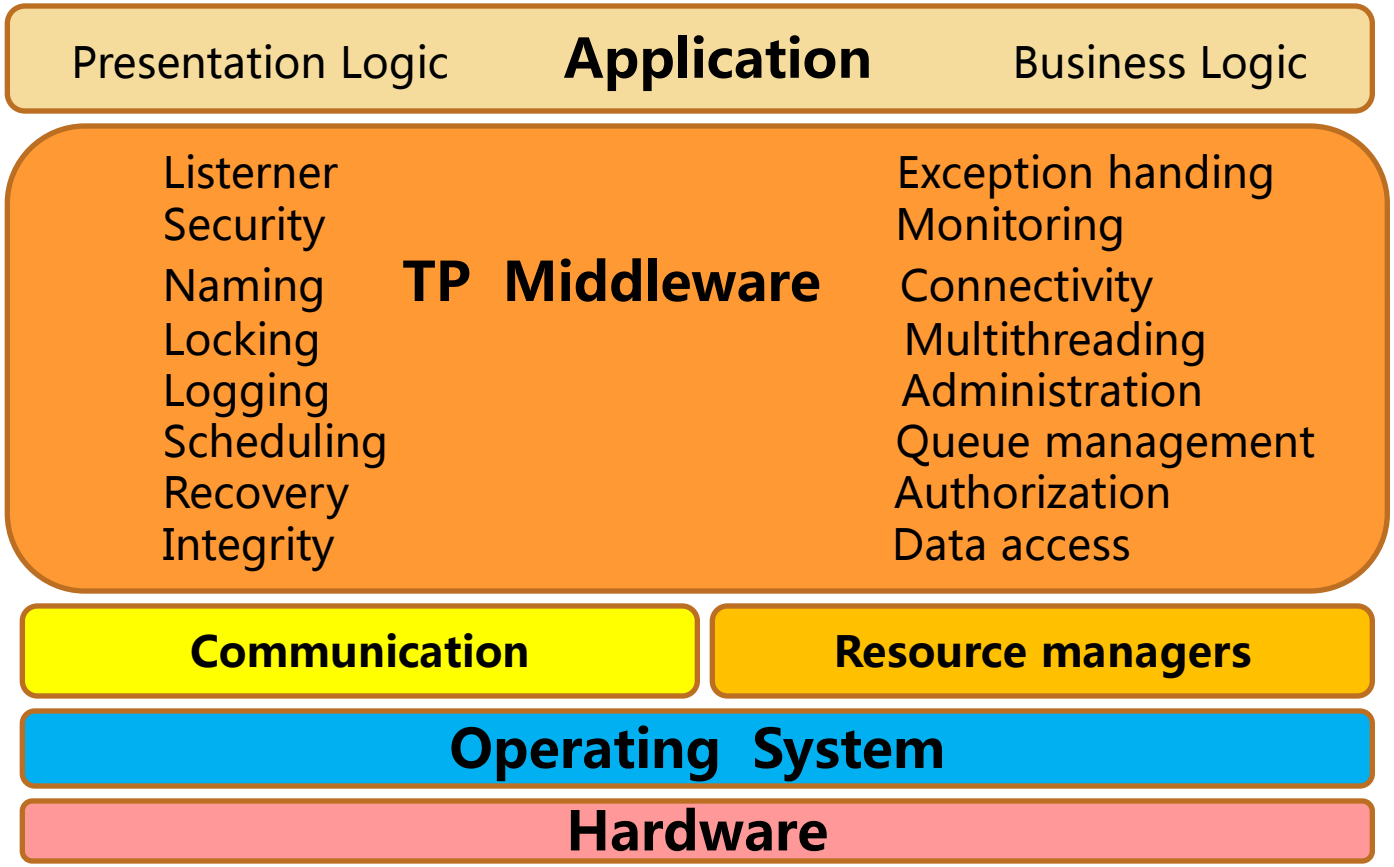


事务处理的特点

- 事务处理不同于其他类型的IT业务分析（如商业分析、网络协作、高性能计算等），主要表现以下方面：
  - 实时性：包含有时间因素，并且通常是用户等待响应。
  - 并行性：事务并行运行，并经常需要访问相同的数据。
  - 一致性：事务必须以完成的形式运行，并且保持数据库的一致性。
  - 安全性：一个事务的运行必须是始终安全的状态。
  - 可恢复性：事务出现故障后，要有完善的恢复机制。

事务的实现方式

（1）数据库系统 （2）中间件



事务管理的中间件结构图

事务评测标准

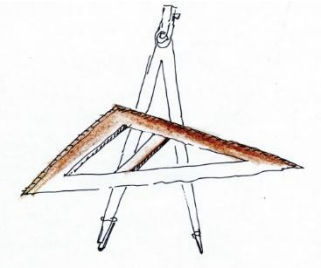
- **性能指标**

  包括： tps（每秒事务处理量）、 tpmC（每分钟事务处理量），Price/tpmC（事务费用）等

- **测试基准（Benchmark）**

  事务处理性能委员会(TPC)制订了多套基准程序

  - TPC-A、TPC-B：简单的银行业事务（过时）；
  - TPC-C：在线事务处理(OLTP)的基准程序；
  - TPC-D：决策支持(Decision Support) 的基准程序。
  - 等等…



事务技术的发展历程

原子性、隔离性；
控制区域概念；
首次提出扁平事务；
2PC协议
恢复块概念；

事务执行模式的形式化描述
事务框架（ACTA）
事务依赖关系理论
OLAP及并发更新
TP系统

20世纪70年代　20世纪80年代　20世纪90年代　21世纪前期

并发控制和恢复；
嵌套事务和多级事务；
层次化结构事务；
保存点和链事务；
长事务和补偿事务；
分布式事务和3PC协议；

分布式实时事务；
实时数据库并发控制；
多数据库系统的事务处理；
. . .

Ref. Jim Gray. Transaction Processing : Concepts and Techniques

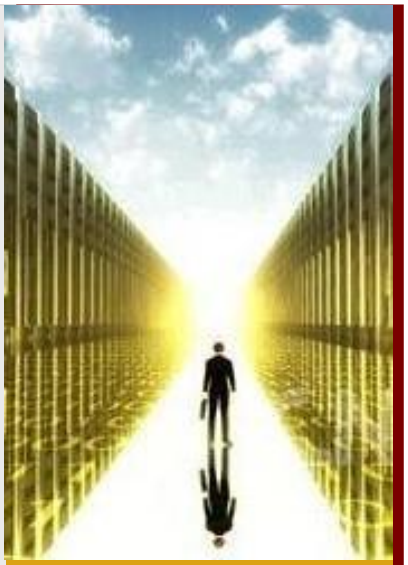
提纲

一 • 事务—历史的脉络

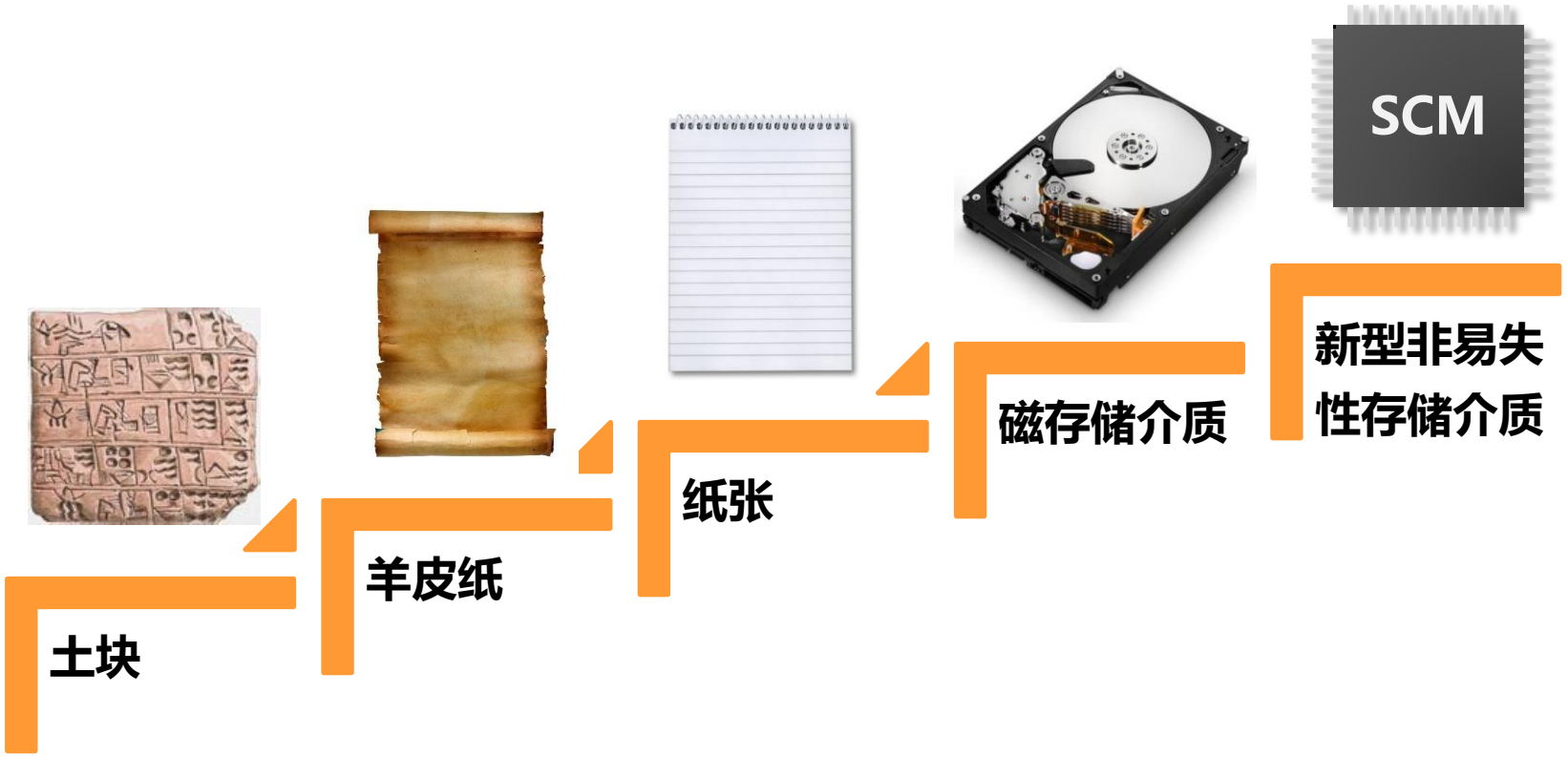
二 • 环境—酝酿变革

三 • 技术—承载发展

四 • 未来—顺势而为



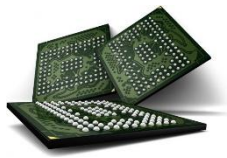
事务在环境中不断变迁

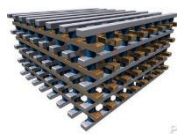


- 事务的思想是古老的，但是内容在不断改变，从扁平事务、链事务、嵌套、分布式事务到多级事务等
- 大数据环境下，随着应用需求，系统架构、应用环境的变更，事务变得越来越复杂、事务处理变得越来越困难

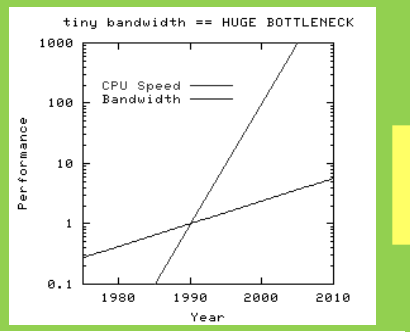
硬件基础的变化—新型存储器件

- **新型存储器件的出现使计算机存储结构发生了变化**
  - 相变存储器（PCM），磁阻式随机存储器（MRAM）和电阻式随机存储器（RRAM）等新型存储级内存(storage class memory, SCM），SSD等

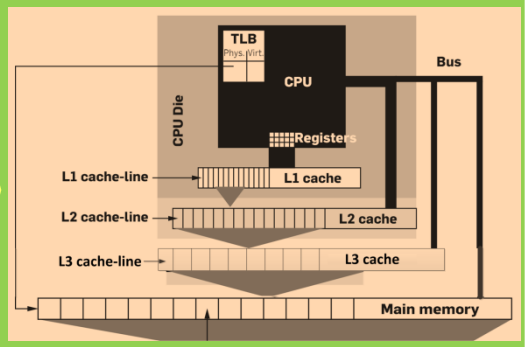



- **新的机遇与挑战**
  - 内容容量大幅增长，具有了存储和管理海量数据的能力，使得基于内存进行事务处理成为可能
  - 计算模式的变化：数据为中心的内存计算模式
  - 数据访问特征的变化：并发性大幅提高，并发控制难度增大
  - 存储特性的变化：如SSD读写代价不对称
  - 能够完全利用新型存储器件的数据库架构和技术需要重新设计（存储层次感知）

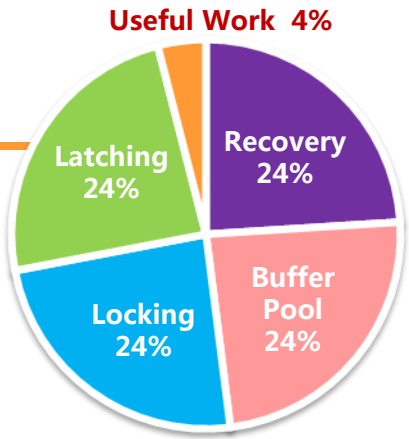


McCalpin's CPU Speed vs. Bandwidth

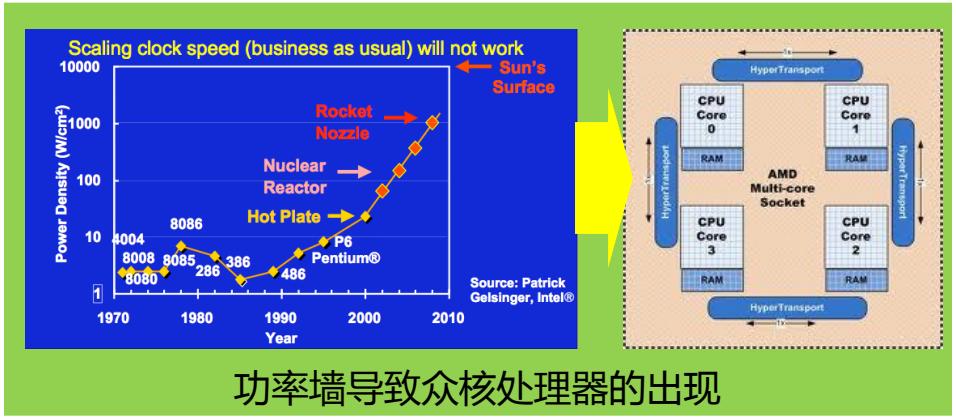


内存墙导致多级缓存的引入

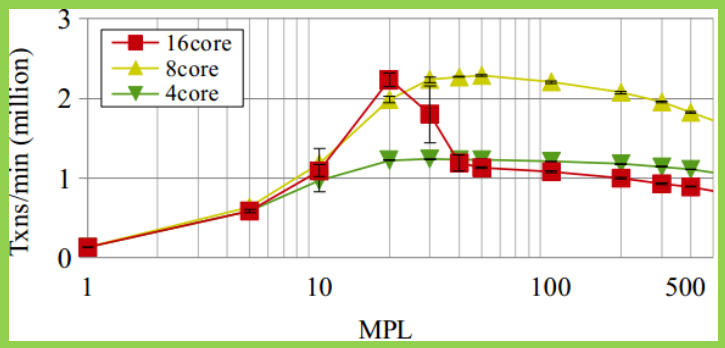
硬件基础的变化—多核、众核



- **芯片的发展**
  - 芯片发展遇到功耗墙、频率墙等问题；处理器也在经历单核到→多核→众核的发展，并出现多核/多处理器共享缓存的新架构



功率墙导致众核处理器的出现



MySQL在多核上的吞吐量

- **问题与挑战**
  - 原有数据库设计之初没有考虑对于多核服务器性能的应用
  - 多核导致共享数据结构访问竞争的可能性增大，传统的锁、日志、事务管理器在多核上的扩展性不佳，且性能较差，需要设计针对多核的并发控制机制，减少lock或latch contention

应用需求的变化

移动设备
- 随时，随地事务处理
- 事务处理量增加
- 数据实时、多样

Web 2.0
- 高交互
- 事务低响应时间
- 非关系型数据

事务处理新需求

云计算
- 大数据时代阶石
- 传统事务原则矛盾

事件检测
- 欺诈等事务
- CEP

OLAP & OLTP
- 实时分析
- 事务与分析融合
- OceanBase

处理范式的变化

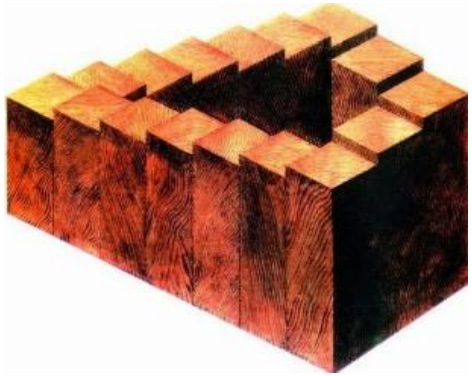
- **Hadoop/MapReduce**
  - 大规模分布式数据处理模型
  - 缺少对事务、索引、完整性约束、视图等特性的内在支持
    [*Michael Stonebraker. MapReduce : A major step backwards*]
- **BigTable/Hbase**
  - 大规模分布式存储系统
  - 缺乏保证数据完整性的全面机制，仅提供单行事务支持（不断改进中，支持单机多行事务等）
- **Pregel/HAMA**
  - 大规模图处理引擎
  - 使用BSP模型、提供高效迭代支持复杂图处理任务，但对于图数据的更新维护、事务等支持， 不如传统的分布式图数据库系统（但是，Trinity等已宣称提供了ACI特性的事务机制）
- **Percolator、Megastore**
  - 利用NoSQL系统实现事务支持（但是具有较高延迟）

事务相关的部分假设的改变

- **传统的假设**
  - 支撑数据库的硬件系统的内存非常有限，数据必须要写入到外存才能保证事务的持久性
- **新的变化**
  - 持久性的对象更加丰富。以前的持久性多是将数据写入磁盘，在新型硬件发展趋势下现在还可以写入到非易失性的随机存储介质（non-volatile memory, NVM）中
  - 持久性被弱化，ACID—ACI。伴随着硬件的发展和云环境存储模式的变化，可以有限度的分离事务处理对外存的操作，将部分任务托管给云进行处理

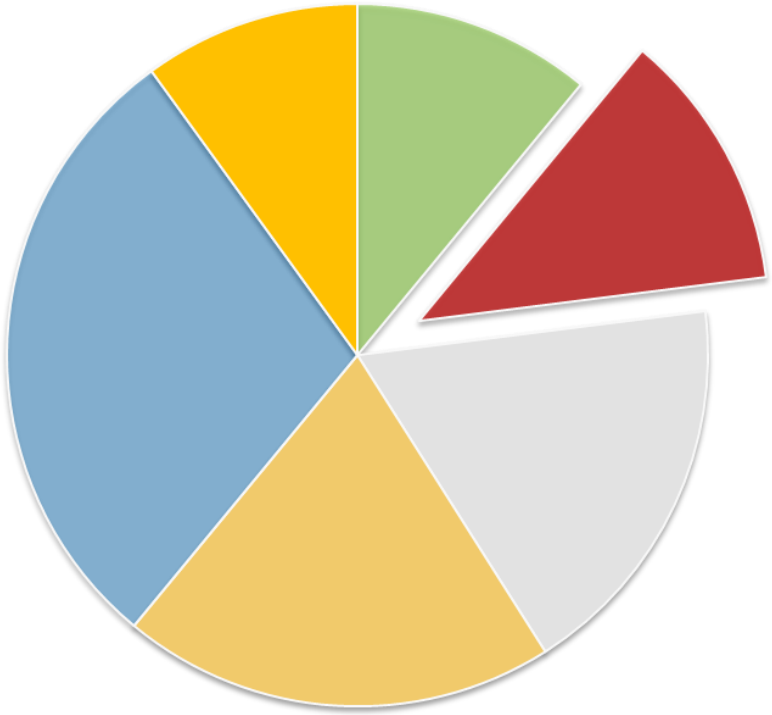
大数据与事务处理的一些paradox

- 分区(Partition)与容忍不一致性
  - *违背ACID 原则*
- 用副本(Replica) 支持容错处理
  - *维护副本一致性引入延迟代价*
- 高扩展和高性能
  - *用scale-out替代scale-up，节点内、跨节点、跨DDB，增加事务处理复杂度*



事务代价与需求的矛盾

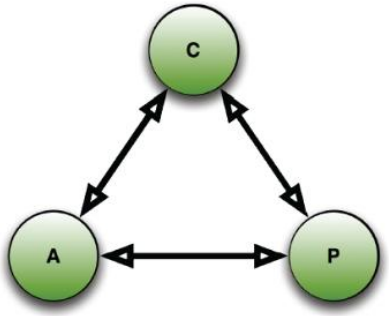
- RDBMS中事务不可取代
- 传统事务处理额外代价较高
- 对一致性要求较弱的应用支付了不必要的开销



OLTP Through the Looking Glass, and What We Found There. S. Harizopoulos, D. Abadi, etc. ACM SIGMOD 2008.

CAP原理与一致性

- ***关系数据库领域 -ACID***
  - 原子性(**A**tomicity)
  - 一致性(**C**onsistency)
  - 隔离性(**I**solation)
  - 持久性(**D**urability)
- ***分布式领域 -CAP***
  - 一致性 (**C**onsistency)
  - 高可用性 (**A**vailability)
  - 分区容忍性 (Tolerance of network **P**artition)



" CAP theorem, which states that of three properties one can only achieve two at any given time. "

P 是必须满足的限制条件

CP：提供一致性保证
AP：提供可用性保证

Ref: Nancy Lynch and Seth Gilbert, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services"

CAP原理与一致性（续）

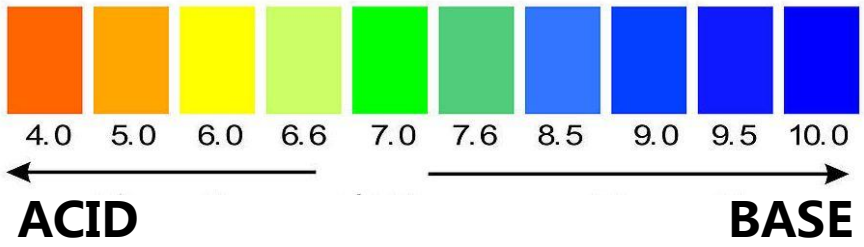
- **AP权衡 != 完全不需要一致性**
  - 很多Web2.0应用对一致性要求较弱
  - 牺牲一致性换取高可用性，是目前多数DDBS产品的设计策略
  - 牺牲一致性=不再要求强一致性=要求系统达到最终一致性
- **常见的几种一致类型**
  - 强一致性：数据被成功更新后，后续任何对该数据的读取操作都得到更新后的值
  - 弱一致性： 存在”不一致性时间窗口”概念
  - 最终一致性：属于弱一致性的一种
  - 其他变体： Causal consistency/Session consistency/...

CAP原理与一致性（续）

- **最终一致性（eventual consistency）：**
  - 不一致窗口关闭后所有访问都将返回更新后的数据
  - 客户端角度：多并发访问时如何获取更新过的数据
  - 服务端角度：如何在小的时间窗口内将更新分布到整个系统中
  - 理解一致性问题时，要结合考虑并发读写的场景
- **NWR模型** [Werner Vogels]
  - N：数据复制的份数
  - W：数据更新完成前需要到达的节点数
  - R：为了读取正确数据需要读取的节点数
  - W+R > N，那么读写节点有重叠，读总是能够得到最新的数据，这就是强一致性

CAP原理与一致性—BASE模型

- **BASE模型（思想）**
  - ACID的反模型，牺牲高一致性，获得高可用性\扩展性
  - ACID强制保证每一个操作完毕后的一致性，而BASE接受该数据库的一致性可能在一定时间内处在不确定的状态中
  - **B**asically **A**vailable：基本可用，容忍分区失败
  - **S**oft state：软状态，状态可以有一段时间不同步，异步
  - **E**ventually consistent：最终一致



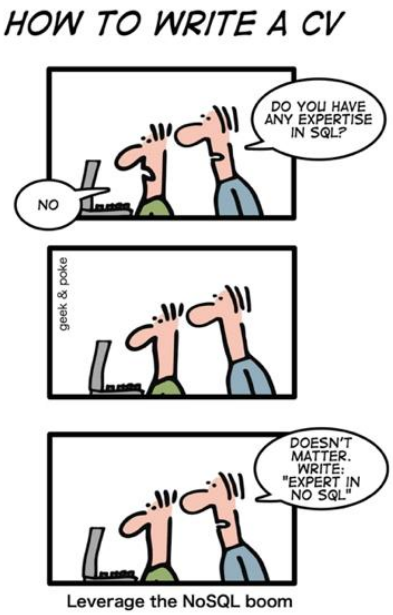
新的运动—NoSQL & NewSQL

- 在一些新的应用场景中RDBMS的关系数据模型并不适用，且在多个方面遇到了问题：
  - Limited SQL scalability
    - Sharding and vertical partitioning
  - Limited SQL availability
    - Master / slave configuration
  - Limited SQL speed of read operations
    - Multiple read replicas
  - SQL limitations for huge amount of data
    - ACID的制约
    - 非结构化数据支持的欠缺

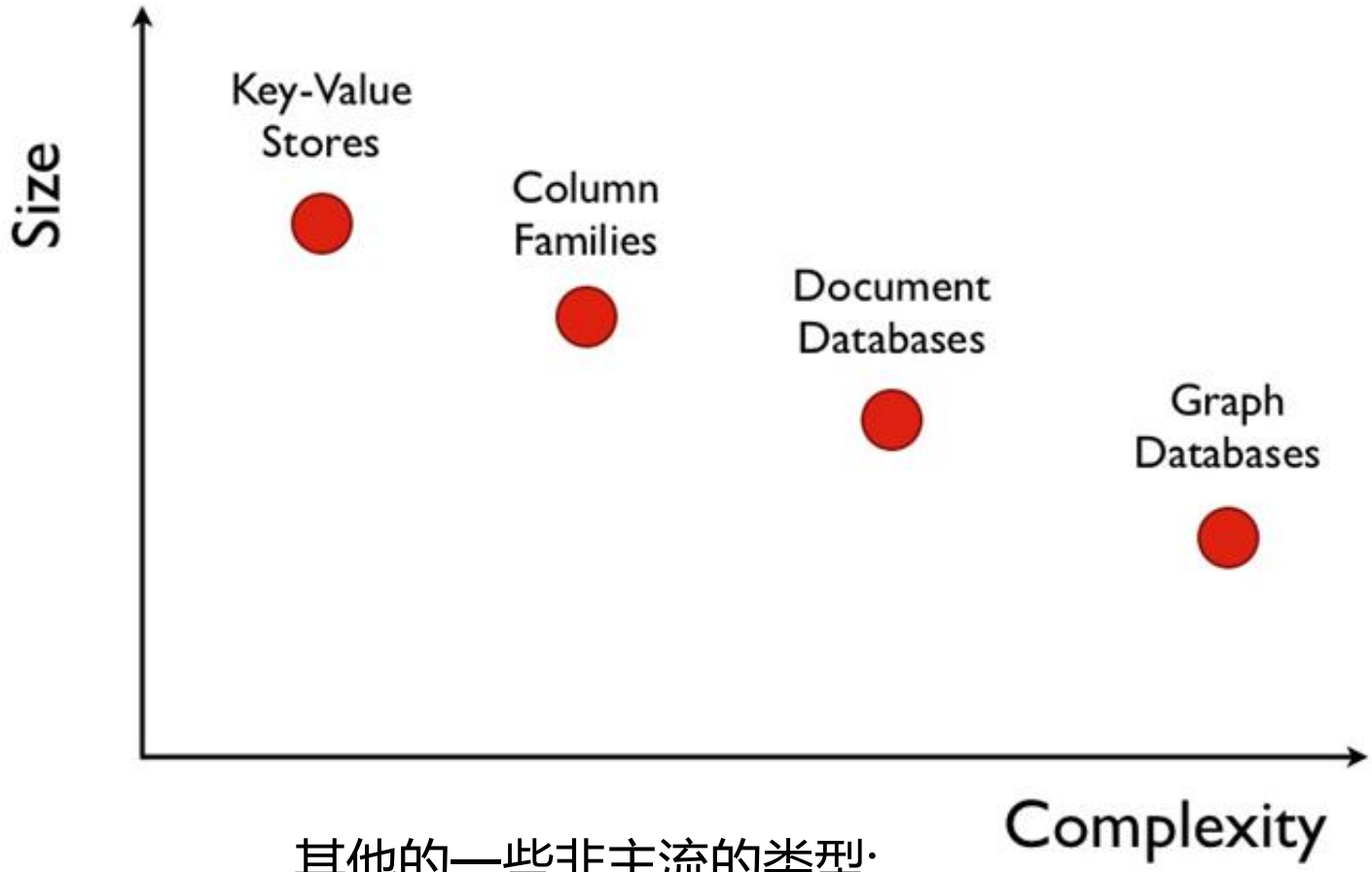
NoSQL

- NoSQL is movement [*wikipedia*]
- NoSQL := "**N**ot **O**nly **SQL**" (*give up SQL, give up ACID*)

- **一些典型特征**
  - 开源、运行于廉价服务器集群上
  - 高可扩展，高可用，容错的分布式架构
  - 无内建的连接（JOIN）操作支持
  - 无模式或对模式的使用更加灵活
  - 不使用SQL作为查询语言
  - 降低对一致性的要求 (放松ACID)
  - 最终一致性



NoSQL数据模型



其他的一些非主流的类型:

- ✓ *XML Databases，如 Berkeley DB XML*
- ✓ *Object Databases，如 db4o, Versant*
- ✓ *Mulitvalue Database，如 InfinityDB*

NoSQL代表性的产品一览

产品	类型	CAP
Cassandra	Columnfamily	AP
CouchDB	Document	AP
HBase	Columnfamily	CP
Hypertable	Columnfamily	CP
MogoDB	Docmument	CP
Neo4J	Graph	CP
Redis	Key/Value	CP
Riak	Document	AP

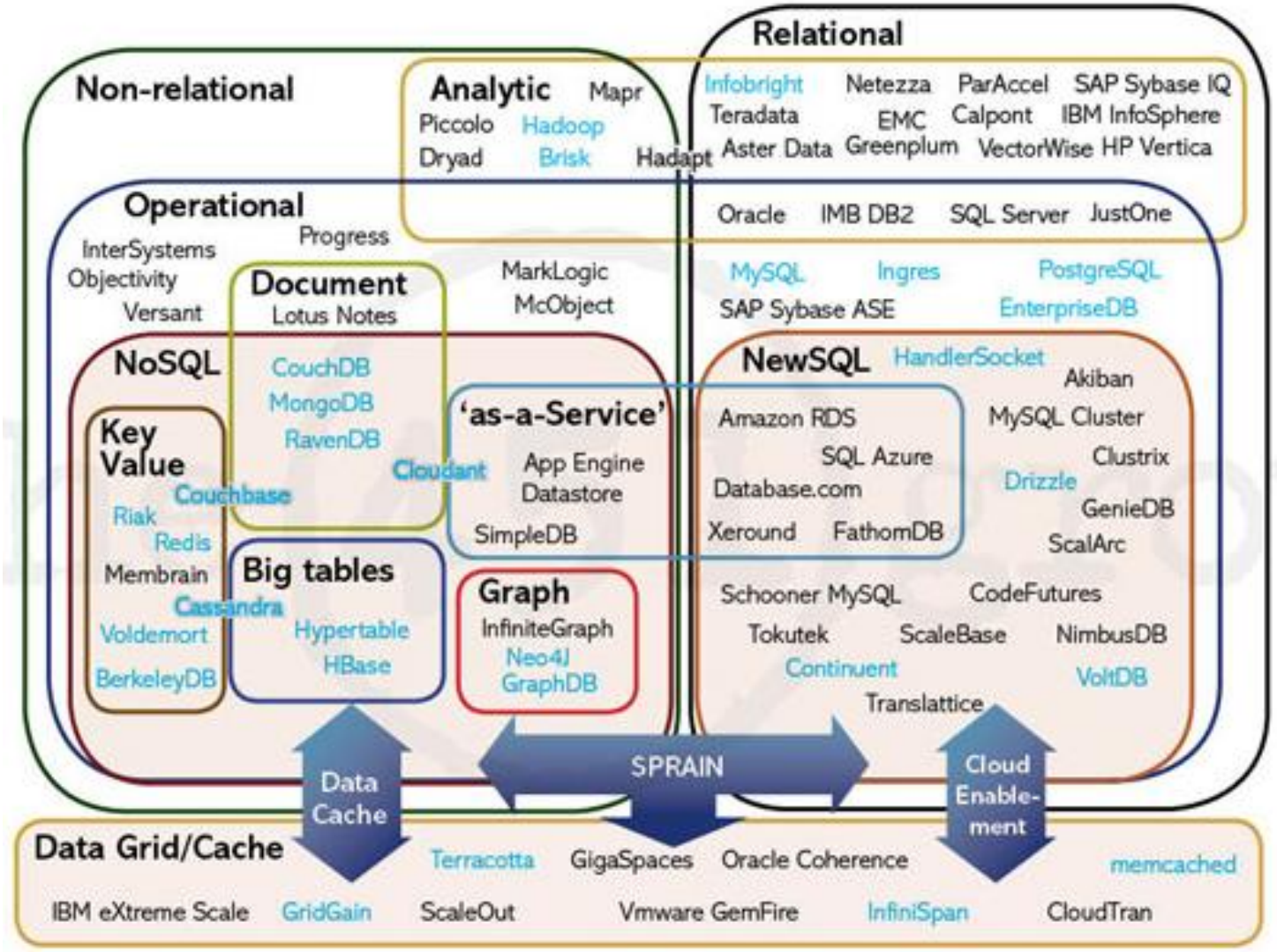
列表未完，更多的产品可参见：

http://en.wikipedia.org/wiki/NoSQL

NewSQL

- **NewSQL := ACID + CAP**
- **NewSQL := OLTP + Scale-out**

NoSQL, NewSQL and Beyond



参考：451 Research

SQL, NoSQL, NewSQL

	SQL	NoSQL	NewSQL
ACID兼容	是	否（CAP\BASE）	是
主要存储介质	Disk	Disk	Memory
支撑架构	高性能服务器	廉价PC集群	混合架构
可扩展性	差（scale-up）	好(scale-out)	好
可用性	较低	可选择	高
一致性	高	可选择	高（有限条件）
容错能力	一般	高	一般
弹性	差	高	一般
延迟	低（有限条件下）	高	低
生态环境	完整	发展中	发展中

TPC-C: Top 10 Performance results

- **各种新技术在改善OLTP数据库可扩展性方面取得了一定的进展，但是在TPC-C的排名表(2013 version5)中前几名仍然被Oracle和DB2盘踞**

Rank	Company	System	Performance (tpmC)
1	ORACLE	SPARC SuperCluster with T3-4 Servers	30,249,688
2	IBM	IBM Power 780 Server Model 9179-MHB	10,366,254
3	ORACLE	SPARC T5-8 Server	8,552,523
4	ORACLE	Sun SPARC Enterprise T5440 Server Cluster	7,646,486
5	IBM	IBM Power 595 Server Model 9119-FHA	6,085,166
***	BULL	Bull Escala PL6460R	6,085,166
6	ORACLE	Sun Server X2-8	5,055,888
7	hp invent	HP Integrity Superdome-Itanium2/1.6GHz/24MB iL3	4,092,799
8	IBM	IBM System p5 595	4,033,378
9	IBM	IBM eServer p5 595	3,210,540
10	IBM	IBM System x3850 X5	3,014,684

注：参与TPC-C的排名的数据库需要提供严格的ACID事务保证

NoSQL\NewSQL系统未进入高排名的原因

- 为实现高可扩展性需要牺牲其他的一些特性（典型的如一致性）
  - **牺牲ACID：NoSQL**
    - **回避两阶段锁，2PC和其他影响可扩展性的部分功能（因此不能参与TPC-C排名）**
  - **限制事务灵活性：NewSQL**
    - **限制参与事务的记录属于同一个分区，对于跨分区的事务要么不支持，要么采用影响性能的2PC，但是TPC-C中包含对跨域多分区（multi-shard\multi-partition transaction）的事务的测试（因此TPC-C排名不理想）**

CAP原理的反思—PACELC

- **不同的声音**
  - Errors in Database Systems, Eventual Consistency, and the CAP Theorem. Michael Stonebraker.
    - CAP/BASE理论中牺牲C来换取AP是不可取的，原因在于导致P问题发生的场景并不常见
    - 且在某些情况下，牺牲C也还是很难保证AP（ *Repeatable DBMS errors*、*hardware failure*、*disaster*、*etc.*）
  - Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. [IEEE Computer] Daniel Abadi.
    - CAP没有将Latency考虑进来（Yahoo PUNTS放弃了AC）
    - 除了在C和A之间权衡，还须在 Consistency 和 Latency 之间权衡（PACELC）

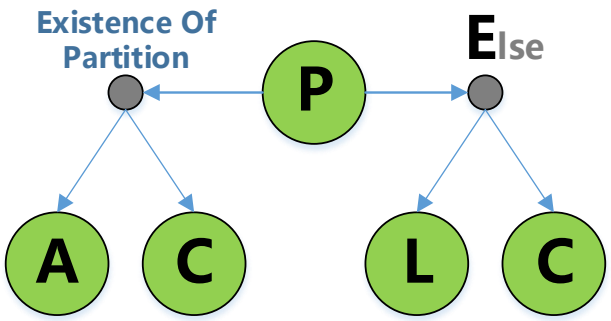
CAP原理的反思—PACELC（续）

It is not merely the partition tolerance that necessitates a tradeoff between consistency and availability; rather, it is the combination of
- *partition tolerance and*
- *the existence of a network partition itself*

As soon as a DDBS replicates data, a tradeoff between consistency and latency arises.

Ignoring the consistency/latency tradeoff of replicated systems is a major oversight, as it is present at all times during system operation.

From: *Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story*



PA/EL	Dynamo, SimpleDB, Cassandra,CouchDB
PC/EC	ACID compliant database systems
PA/EC	GenieDB
others	…

系统设计的时候需要进行各式各样的权衡，这并不是一个简单的CAP或者PACELC就能够解决的，需要按需决策。

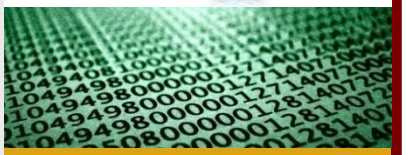
提纲

一 • 事务—历史的脉络

二 • 环境—酝酿变革

三 • 技术—承载发展

四 • 未来—顺势而为



研究进展

1. 针对应用需求的研究
2. 面向新型硬件的研究
3. 面向新型处理范式的研究
4. 典型系统及其相关技术

在Cloud上支持事务

无处不在的移动设备
随时随地的事务

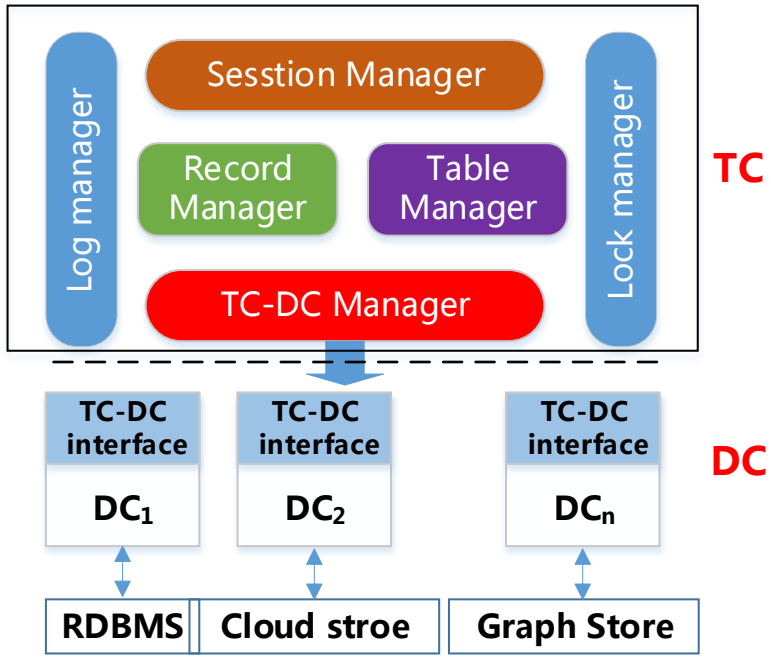
Begin Transaction

1. **Add me to Dave’s friend list**
2. **Add Dave to my friend list**

End Transaction

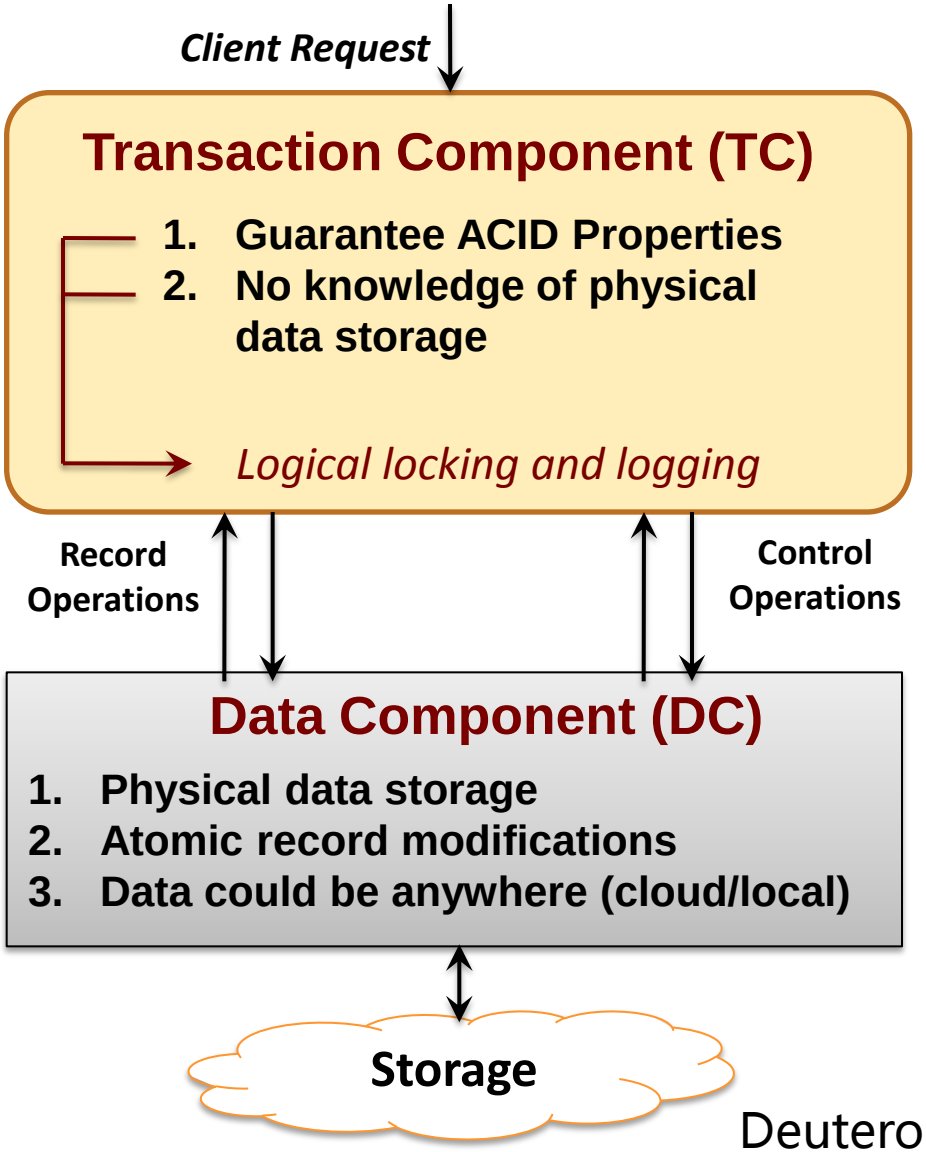
Deuteronomy [CIDR 2009] [CIDR 2011]

- **动机**
  - 支持跨cloud的事务
  - 将事务处理与底层数据存储解耦
  - 保证数据的事务在同一机器节点上



- **特点**
  - 将数据存储引擎核心分解为TC(Transactional Component) 和DC(data component)
  - TC通过逻辑的并发控制和对数据物理布局透明的恢复机制来提供事务支持，不需要将DB在TC间进行分区；DC提供支持原子操作的面向记录的接口，并负责物理数据的组织和缓存
  - 提供与传统的存储引擎依赖于解耦事务执行逻辑和数据物理布局不同的优化策略
  - 事务被限制在逻辑实体（TC）上执行，避免了跨DC的2PC

Deuteronomy [CIDR 2009] [CIDR 2011]



Deuteronomy Architecture

Web2.0下业务驱动的事务处理

淘宝的数据规模及其访问量对RDBMS提出了很大挑战：数十亿条的记录、数TB的数据、数千TPS、数万QPS让传统的关系数据库不堪重负，单纯的硬件升级已经无法使得问题得到解决

OceanBase
A Powerful Database System

OceanBase可扩展数千亿条记录、数百TB数据、数十万QPS以及数万TPS。同时具备实时容错、自动故障恢复和99.999%高可用性

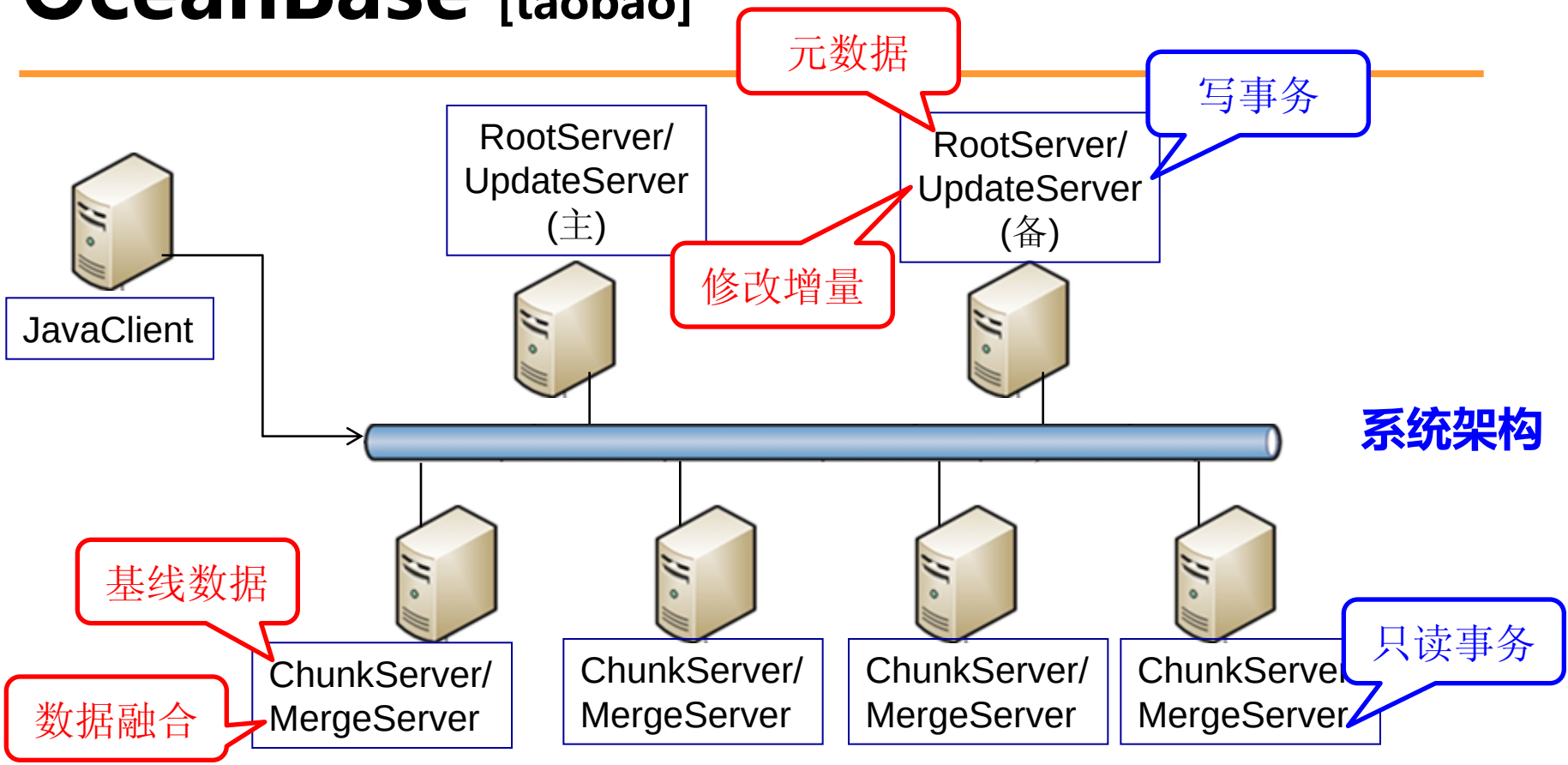
截止到2012年8月为止，OceanBase数据库支持了阿里巴巴集团下多个重要业务的数据存储，支持业务包括收藏夹、直通车报表、天猫评价等

数据来源：http://alibaba.github.io/oceanbase/

OceanBase [taobao]

- **动机**
  - 源于淘宝自身数据的特点和应用需求
    - 数据量大但修改量较小，一千亿 * 1%* 100B = 100G
    - 需要有效的区分最新修改的增量数据和以前的基准数据
    - 查询事务较多，写事务相对较少，但要求一致性强和高实时性
- **特点**
  - OceanBase = transactionality + scalability
    - 数据 = 基准数据+增量数据（数据分类并分离）
    - 增量数据(增删改)：单机，内存+SSD
    - 基准数据：多机，分布式B+树（类似BigTable）
    - 读写分离的偏向于CA特性的高可用性高扩展性的系统
  - 支持跨行跨表事务：集中化单机写事务+分布式读事务（读写事务分开）
  - 支持强一致性：本地实时同步，异地准实时同步

OceanBase [taobao]



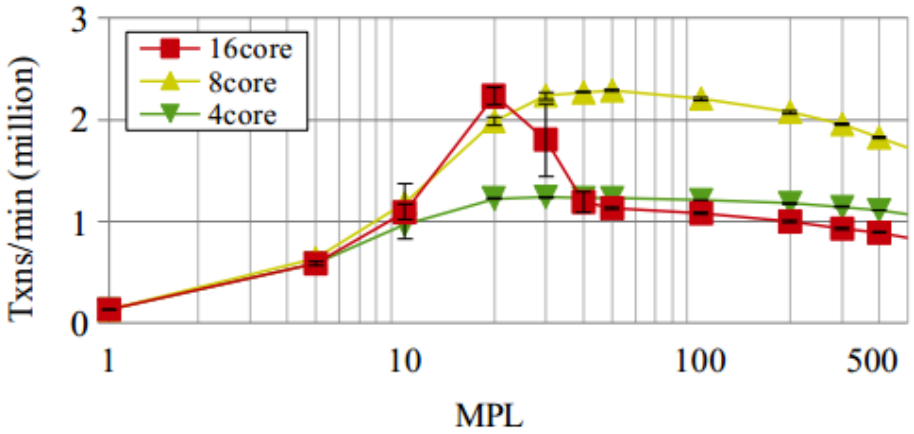
- 主控服务器RootServer：主+备，数据定位/全局Schema/机器管理
- 增量数据服务器UpdateServer：主+备，实时修改(内存+SSD)
- 基准数据服务器ChunkServer：多台，B+树叶子节点 (磁盘或SSD)
- 增量数据定期合并到基准数据中，从而分散到多台ChunkServer

相关研究工作

1. 针对应用需求的研究
2. 面向新型硬件的研究
3. 面向新型处理范式的研究
4. 典型系统及其相关技术

[H. Jung, H. Han, etc. SIGMOD'13]

- 可扩展性有限
  - 多核扩展瓶颈
  - 无锁冲突时依旧

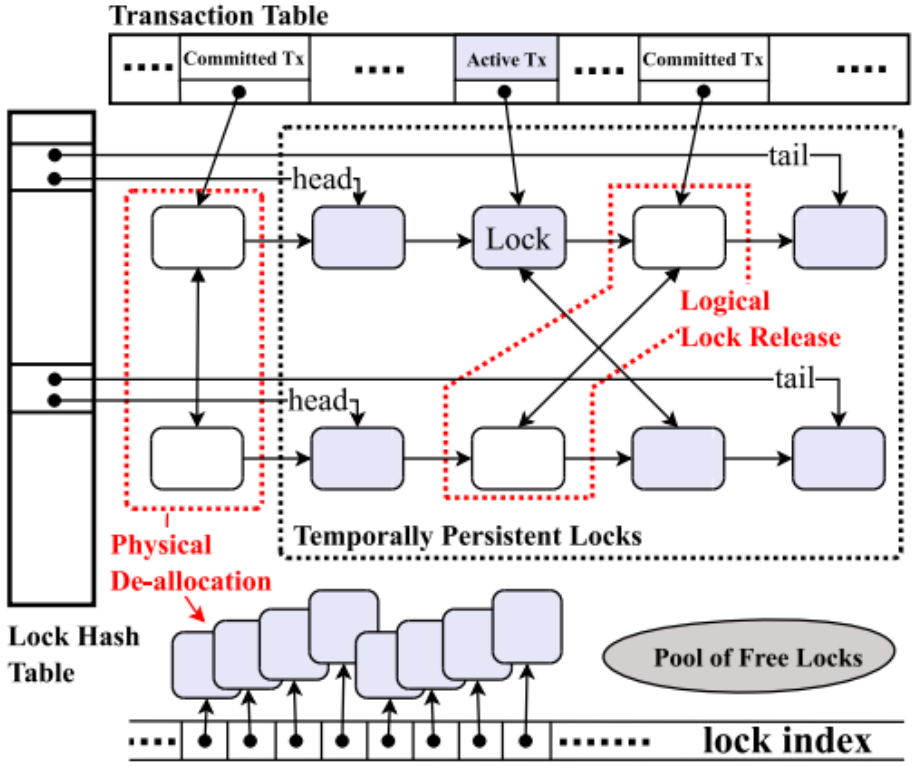


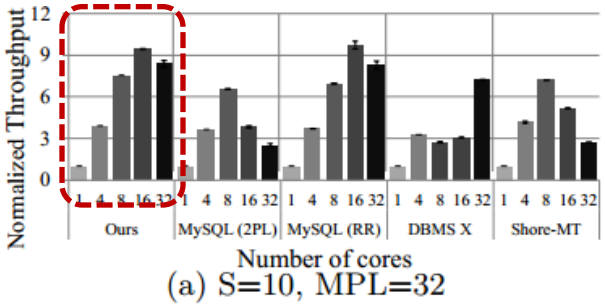
- 原因：Latch冲突
  - 获得/释放锁都需要取得互斥锁
  - 高冲突环境下代价巨大
- 解决方案
  - 获得锁：提前分配(pre-allocation)
  - 释放锁：延迟释放(lazy de-allocation)

NICTA. A Scalable Lock Manager for Multicores. SIGMOD'13.

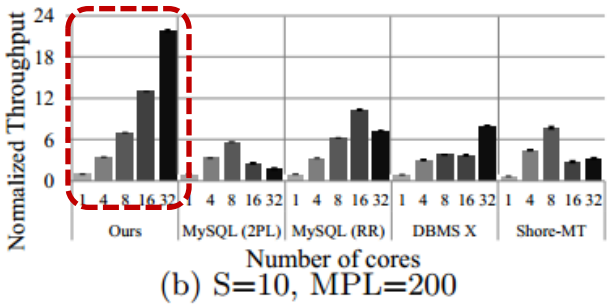
[H. Jung, H. Han, etc. SIGMOD'13]

- 实现
  - 基于MySQL
  - RAW
  - 栅栏同步
- 结论
  - 批处理式的获得/释放锁

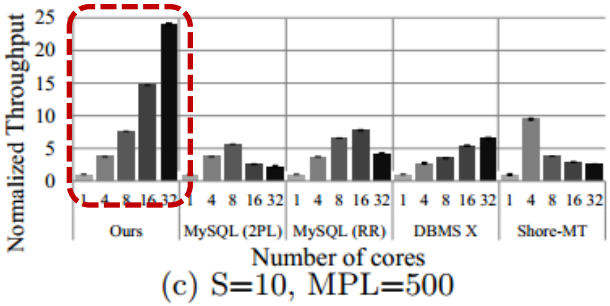




(a) S=10, MPL=32



(b) S=10, MPL=200

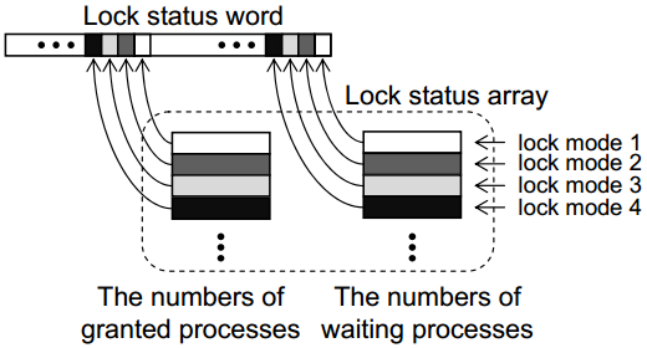


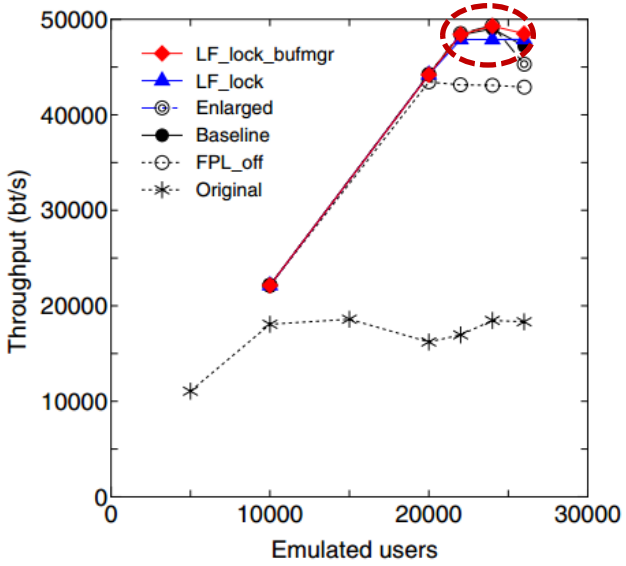
(c) S=10, MPL=500

A Scalable Lock Manager for Multicores. SIGMOD'13.

[T. Horikawa. SIGMOD'13]

- 多核上的Latch-free改进
  - Latch-free HashTable
  - Lock manager：多锁请求
- 实现
  - 基于PostgreSQL
- 结论
  - 吞吐量提高2.5倍
  - 采用Latch-free的数据结构
  - Latch-free扩展性优于FPL





[K. Ren, A. Tomson, D. J.Abadi. VLDB'13]

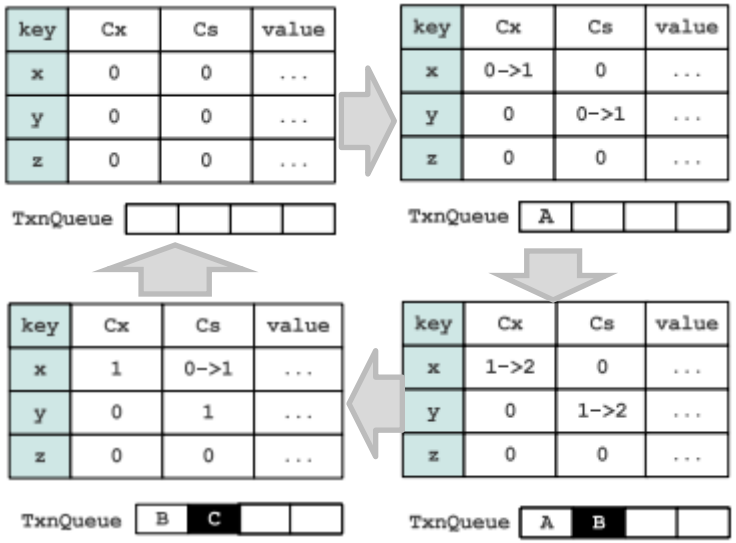
- 内存数据库
  - 单核：16%-25%锁管理代价
  - 多核：代价更大(更多请求)
- 代价来源
  - HashTable查询
  - Latch获得
  - 链表操作
- 解决方案：轻量级锁
  - 去除核心锁数据结构
  - 去除具体特定锁请求信息

[K. Ren, A. Tomson, D. J.Abadi. VLDB'13]

- Very Light Locking(VVL)
  - 代价更低
  - 跟踪更少的信息
  - 用整型对代替链表
- 更少信息的代价
  - 需要判断某个事务是否可以获得锁
- Selective Contention Analysis(SCA)
  - 判断事务得到锁的顺序

Transaction Request Sequence

txn	read set	write set
A	x,y	x
B	x,y	x
C	x	0



[P.Larson, S.Blanas, etc. VLDB'12]

- 动机
  - 主内存数据库可以支持较高的事务率，但标准的并发控制方法并不适用，典型的如长只读事务会阻塞写事务
- 传统事务处理特点
  - 单一版本(Single-version)锁
  - 短事务、无热点
- 提出两种多版本并发控制(MVCC)
  - 基于验证机制的乐观锁
  - 基于锁机制的悲观锁
- 结论
  - 单一版本锁实现简单，但脆弱
  - 多版本锁代价较高，但具有弹性，受长事务和热点影响较小
  - 乐观锁相对悲观锁吞吐更高

optimistic

VS.

Pessimistic

Microsoft,, University of Wisconsin

High-performance concurrency control mechanisms for main-memory databases

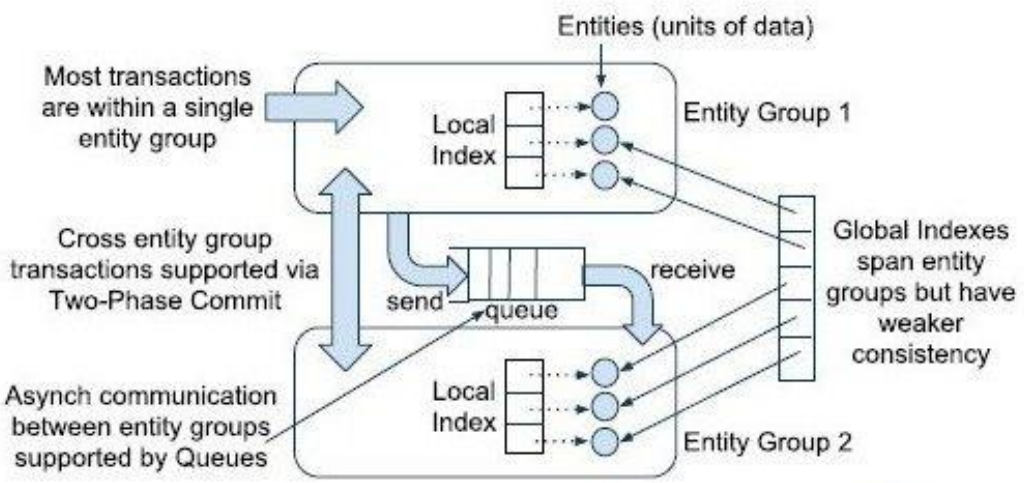
相关研究工作

1. 针对应用需求的研究
2. 面向新型硬件的研究
3. 面向新型处理范式的研究
4. 典型系统及其相关技术

Megastore [CIDR 2011]

- **结合NoSQL与RDBMS的一个谷歌内部的存储系统**
  - 底层存储采用Bigtable
  - 实现了类似RDBMS的数据模型
  - 基于MVCC的事务提供了单分区强一致性保证
  - 采用paxos算法保证不同分区间数据更新的一致性
  - 在同一个datacenter上实现了数据细粒度的分区（Entity Groups集合）
  - 在同一个Entity Group中(相当于单库)的多个Entity的更新事务采用single-phase ACID事务，而跨Entity Group(相当于跨库)的Entity更新事务采用2PC 分布式事务

注：Google目前正在使用的存储系统是Spanner架构（在后面系统篇介绍）



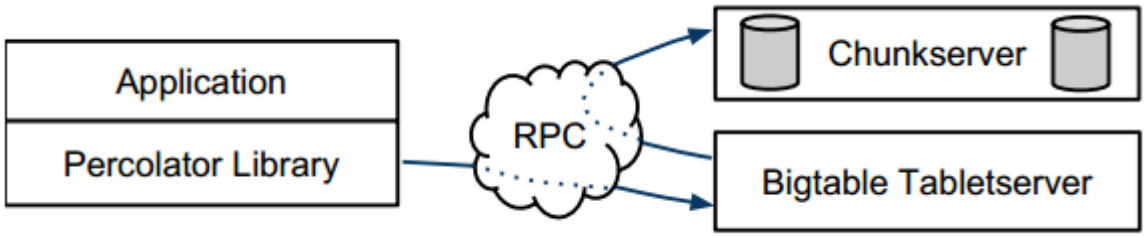
Megastore [CIDR 2011]

- **事务执行**
  - 采用乐观锁的方式实现事务，读取时记录数据的版本号，事务提交时检查Entity Group当前的事务版本号与读取时记录的版本号是否相同，如果相同则成功提交事务，否则重试
  - Megastore实现的事务隔离级别为Serializable（可串行化）

- **完整事务生命周期包括以下步骤**
  - 读：获取时间戳和最后一个提交事务的日志位置
  - 应用逻辑：从BigTable读取并且聚集写操作到一个日志Entry
  - 提交：使用Paxos将日志Entry加到日志中
  - 生效：将数据更新到BigTable的实体和索引中
  - 清理：删除不再需要的数据

Percolator [OSDI 2010]

- **使用分布式事务的大规模数据增量更新系统**
  - 构建于分布式存储系统BigTable之上
  - 是google一个处理增量网页索引的系统，也可以视其为一个具有增量处理功能的mapreduce系统的变体
  - 通过“快照隔离”（利用BigTable的时间戳维度）实现了遵从ACID的跨行和跨表事务，从而满足多线程在多台服务器上对文档库进行转换操作的需求
  - 具有宽松的延迟需求，其不提供用于事务管理的中心节点和全局锁侦测器，采取一种延迟的锁清空方式，会导致事务提交的延迟，但保证了系统的高可扩展能力



相关研究工作

1. 针对应用需求的研究
2. 面向新型硬件的研究
3. 面向新型处理范式的研究
4. 典型系统及其相关技术

基于分布式数据的事务处理

- **ACID与可扩展性的权衡**

We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions.

--from: http://www.infoq.com/news/2012/10/google-spanner

- 一些新的研究在反思，保证ACID语义时是不是必须要牺牲高可伸缩性，反之亦然。
- 部分支持严格ACID语义系统的出现提供了一些关于如何平衡ACID与可伸缩性的新思路（如Spanner、Calvin等），也再次说明，NoSQL并非是高可伸缩性持久化的唯一方案，不同系统有不同的设计策略

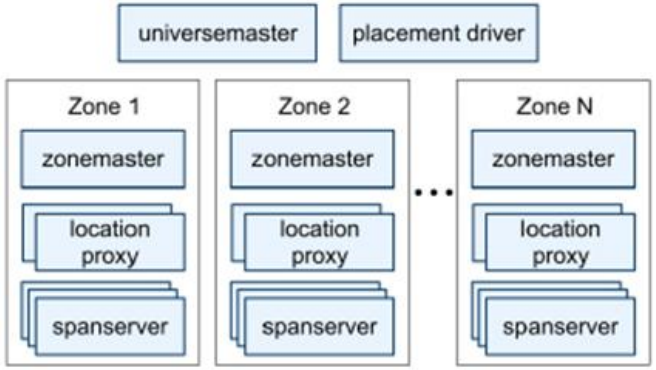
新型系统的事务支持策略

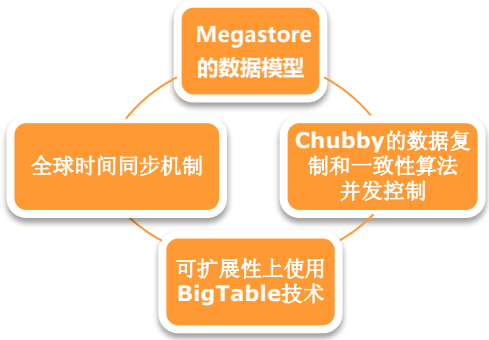
- **现状：支持单操作单数据对象的事务**
- **需求：缺少对访问多个数据对象的多操作的事务支持**
- **实现通用的事务语义会影响系统的可扩展性**
  - **限制事务的访问粒度（Co-located Data）**
    - 将事务处理所涉及的数据共置于一个节点上，可回避分布式事务 ，但是受限于应用的访问模式和特定的schema模式
    - ElasTraS /Megastore/Relational Cloud/Cloud SQL/Deuteronomy/Hyder/…
  - **弱化了支持系统可扩展性的事务保证（Distributed Data）**
    - 支持跨越多个节点的分布式事务，执行代价比较高昂，且多借助弱化隔离级别、一致性保证、或延迟保证，限定可作为事务的操作类型等方法来支持分布式事务
    - Percolator/Walter
  - **可扩展的ACID事务支持**
    - Spanner/Calvin

Ref : Data Management in the Cloud- Challenges and Opportunities

Spanner [OSDI 2012]

- **Spanner(Google)**
  - 可扩展的、多版本的、同步复制的、全球级分布式数据库
  - 具有很高的设计指标：可以扩展到几百万个机器节点，跨越成百上千个数据中心，具备几万亿数据库行的规模
  - 当流量激增、硬件超负荷工作时，数据可以在百万级的数据中心中自动进行转移，提供高可靠性
  - 采用Megastore的数据模型，Chubby的数据复制和一致性算法，在可扩展性上借助了BigTable中的技术
  - Spanner支持通用的事务，提供了基于SQL的查询语言





Spanner: Google's Globally-Distributed Database, OSDI 2012

Spanner [OSDI 2012]

- **外部一致性保证:（宣称是第一个全球可扩展且支持外部一致性的分布式事务）**
  - 利用同步复制和多版本保证外部一致性。为事务分配全球范围内有意义的提交时间戳（即使是分布式事务），利用时间戳保证的事务序列化的顺序，保证并发控制的正确性
- **TrueTime（全球时间同步机制）**
  - 时间戳的分配由TrueTime提供（提供了一组API）
    - 高精度和可观测误差的本地时钟可以同步全球的时间
    - 利用GPS和原子钟实现
  - 还可以支持无锁的只读事务、原子schema更新，和无阻塞的历史数据读操作

操作	并发控制
读写事务	悲观锁
只读事务	无锁
快照读（客户端提供时间戳）	无锁
快照读（客户端提供时间边界）	无锁

Method	Returns
TT.now()	*TTinterval*: [*earliest*, *latest*]
TT.after(t)	true if t has definitely passed
TT.before(t)	true if t has definitely not arrived

TrueTime API. The argument t is of type *TTstamp*.

Calvin [SIGMOD 2012]

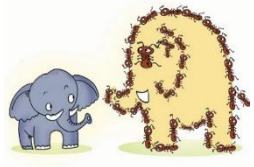
- 高端硬件+传统技术+真实需求 = Rank1（TPC-C）

Rank	Company	System	Performance (tpmC)	Price/tpmC
1	ORACLE	SPARC SuperCluster with T3-4 Servers	30,249,688	1.01 USD

- Calvin
  - 通用的廉价PC和存储部件
  - 支持提供CRUD操作的任意非事务存储系统
  - 通过调度事务与复制，实现可伸缩、高可用的、高性能的分布式事务处理
  - 100 EC2节点，TPC-C New Order达到50w tps

System Information	
Total System Cost:	30,528,863 USD
Performance:	30,249,688 tpmC
Price/Performance:	1.01 USD per tpmC

50w tps
$30,000,000 vs $300

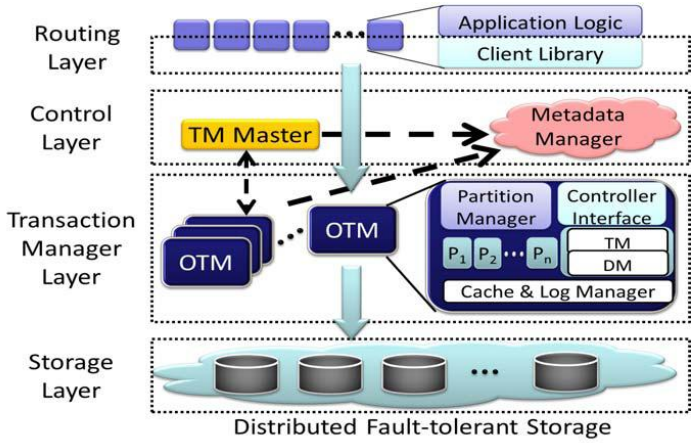


Calvin [SIGMOD 2012]

- **ACID事务的支持**
  - 其是架构在存储服务之上的事务调度和复制协调服务，提供了事务调度层和数据拷贝层等一整套方案，具有高性能的事务特性
- **核心思想**
  - 用顺序执行的事务消除并发控制，消除相应的开销。即通过对多个事务的执行顺序进行重排来提高性能
- **特点**
  - 确定性执行（deterministic execution）模型使得系统在保持强一致性的同时不太影响系统的事务吞吐量
  - 在获取锁并执行事务之前先把那种占用资源多、浪费时间和CPU的“hard”工作做完
  - 在事务获得锁和开始执行之前确定多个节点或多个副本如何执行事务的方法，以及确定所有事务的执行顺序。
  - 不同的副本按照确定的策略来执行事务，节点失效等情况下不回滚这个事务

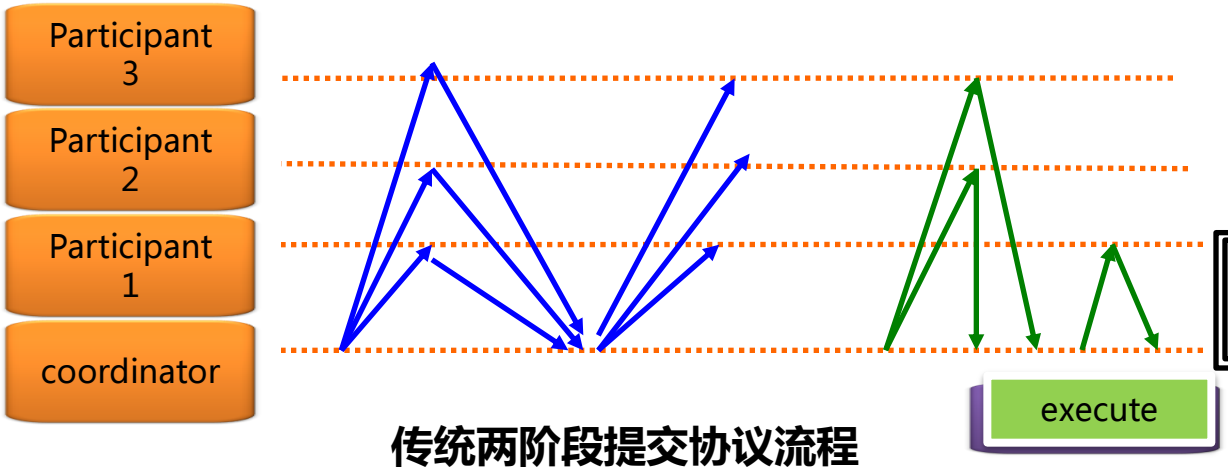
ElasTraS [HotCloud 2009]

- **云上弹性的事务型数据存储**
- **动机**
  - 解决在云计算环境下数据存储的可扩展性和伸缩性
  - 利用可扩展性的底层基础架构，提供可扩展的事务性数据访问

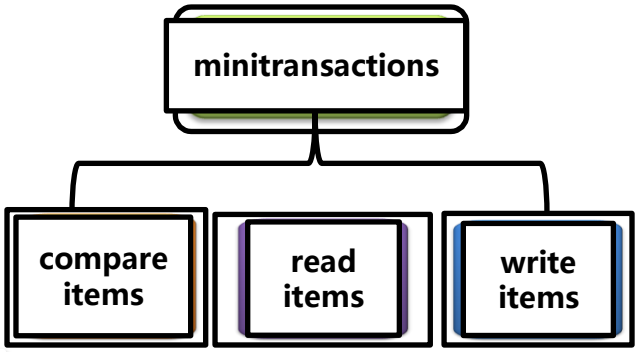


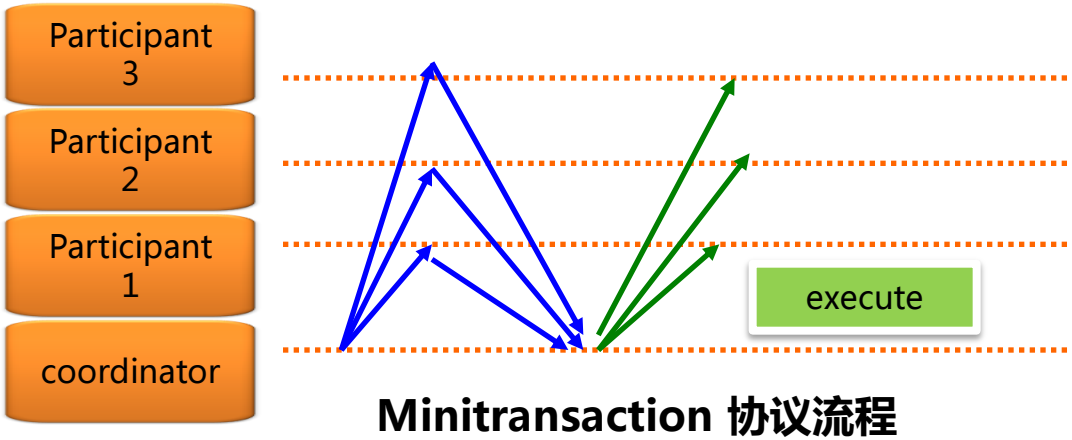
- **数据分区**
  - 静态分区：将分区映射到可保证事务的ACID特性的特定的OTMs（Owning Transaction Manager）中
  - 动态分区：为确保可扩展性和避免分布式事务，只支持**minitransaction**，一个可扩展性和有限的事务语义，能保证恢复但不能保证全局同步。

传统2PC 和 minitransaction [SOSP 2007]



传统两阶段提交协议流程





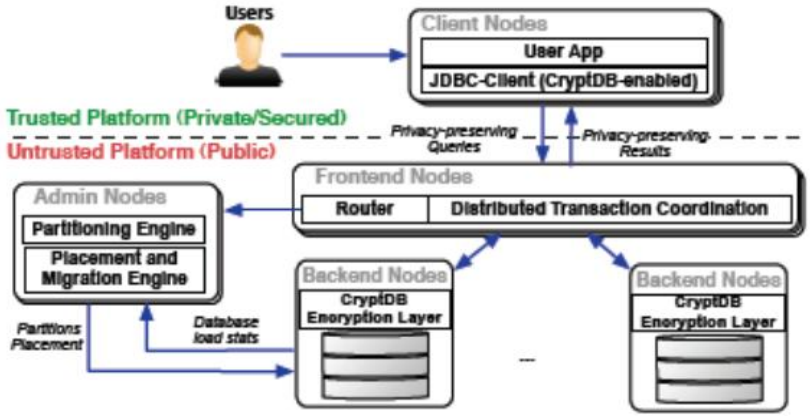
Minitransaction 协议流程

Minitransaction原语对数据并发访问提供了高级别的抽象的和ACID事务语义的支持，开发者仅需关注相应数据结构的设计（如write items），Minitransaction以ACID方式操作上述items。Minitransaction中减少了一次网络交互，避免了2PC在可扩展性上难以回避的阻塞问题

HP, VMware. Sinfonia: a new paradigm for building scalable distributed systems

Relational Cloud [CIDR 2011]

- **云上新型的事务型DBaas**
- **动机**
  - 多租户下的高效率响应
  - 同时支持高可扩展性+安全
- **选择-Cloud**
  - 典型的云数据库产品
  - 硬件配置低+开销小+资源利用率高
- **特点**
  - 多用户workload-aware方法，自动区分负载类型，把类型相近的负载进行共置
  - 基于图的数据划分方法，对复杂事务型workload也具有近线性的可扩展性
  - 利用可调节的Security scheme在加密数据上执行SQL查询，包括聚集，join等



MIT. Relational Cloud: A Database-as-a-Service for the Cloud

Cloud SQL [ICDE 2011]

- **可支持云计算工作负载的关系数据库系统**
  - 使用Microsoft SQL Server为核心
  - 用基于shared-nothing架构的分区数据库支持scale-out
  - 将事务限制在单个partition上执行，同时使用Table Group支持一组丰富的事务操作并能避免2PC
  - 单个分区的所有事务在primary-copy副本上执行，系统同时利用primary-copy副本策略的变体支持高可用性
  - 为数据密集型系统设计了Couple storage，耦合的数据和执行无需经过网络传输数据，提高了性能

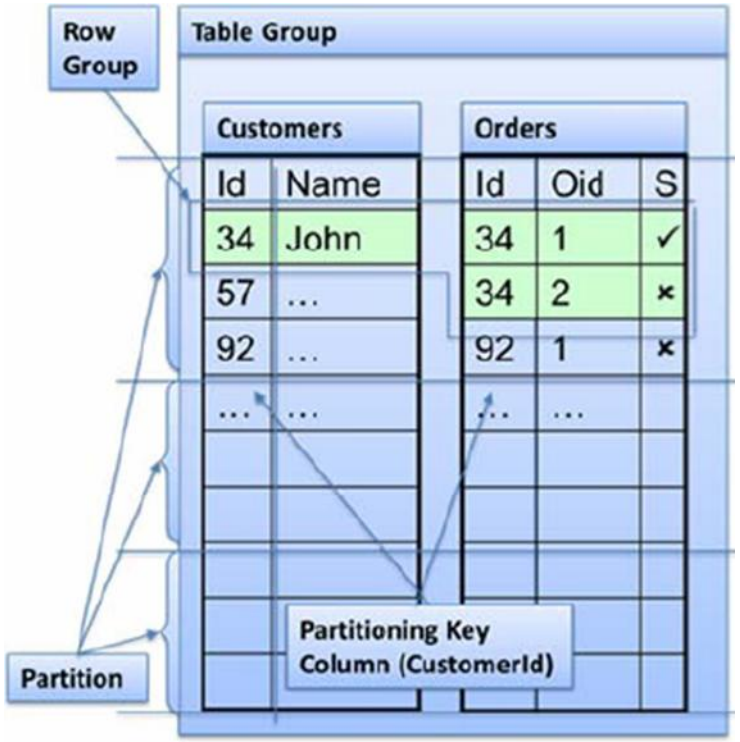


Figure 1 Cloud SQL Server data model

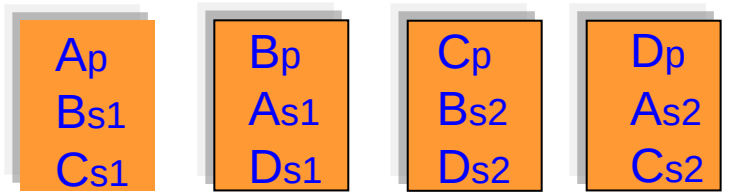
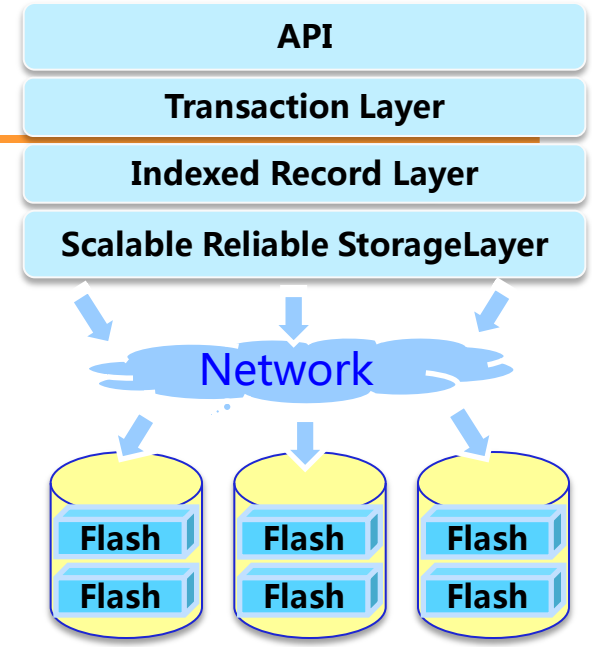
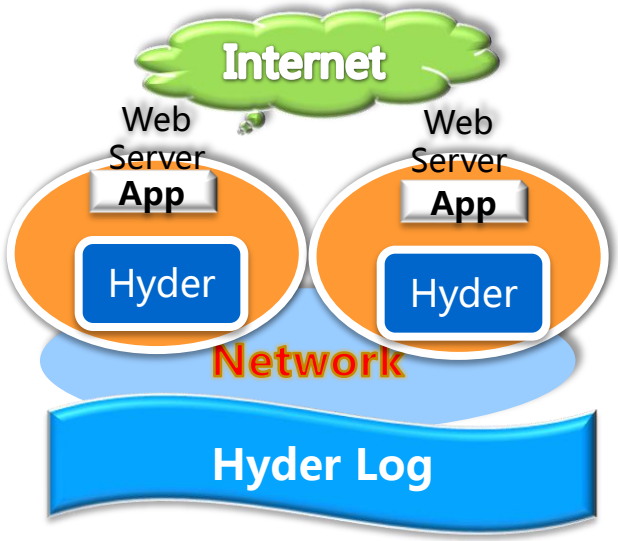


Figure 2 Load balancing primary and secondary replicas

Hyder [CIDR 2011]

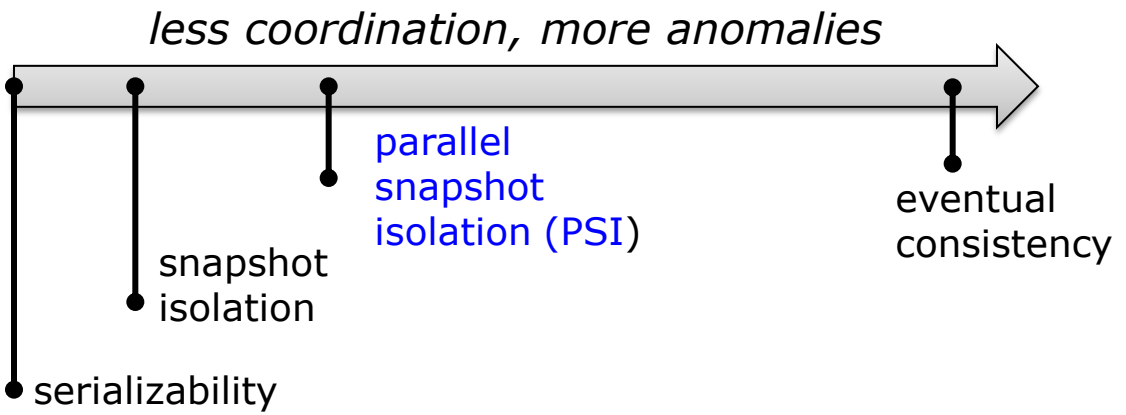
- **基于共享内存的事务型记录管理器**
- **动机**
  - 利用flash和高速网络简化大规模web服务的扩展性问题
- **特点**
  - 在flash中存储整个log-structured的DB
  - 不需要分区，log就是DB，通过广播式的日志更新方式维护一致性（代价较高）
  - DB是multi-versioned的，每个server均可访问log，或者从邻居server cache中抓取。因此，可以在交叉的server间将查询并行化
  - 每个服务器在cache中保存最新提交状态的DB ，通过读取cache中的快照和写intention log record执行事务





Walter [SOSP 2011]

- **支持事务和远距站点复制的键值型存储**
- **动机**
  - 单个站点已经无法支持大规模web应用，应用多是地理分布的
  - 为开发部署在多个站点上的大规模应用提供基础设施
- **特点**
  - 用“平行快照隔离”(PSI, parallel snapshot isolation)事务语义弱化了“快照隔离”，降低支持分布事务所需的代价
  - PSI不对全部更新事务提供全局次序保证，只保证不同数据中心主机间的因果序，允许事务被异步复制，并可避免提交时昂贵的跨数据中心的代价
  - 阻止了write-write冲突，使开发者无需在程序中关注冲突解决逻辑



新型DDBS的设计选择

- **DDBS设计策略：**
  - **不提供ACID支持**
    - Dynamo\MogoDB\CouchDB\Cassandra 等
  - **提供有限的ACID支持**
    - BigTable（HBase）：单行事务
    - Azure,Megastore,OracleNoSql DB：事务限制在小的子集上
  - **完全的ACID支持**
    - VoltDB：用存储过程的方式在内存中实现高性能事务处理
    - Calvin：用确定性执行模型支持高性能事务
    - Spanner：用TrueTime机制实现外部一致性的事务

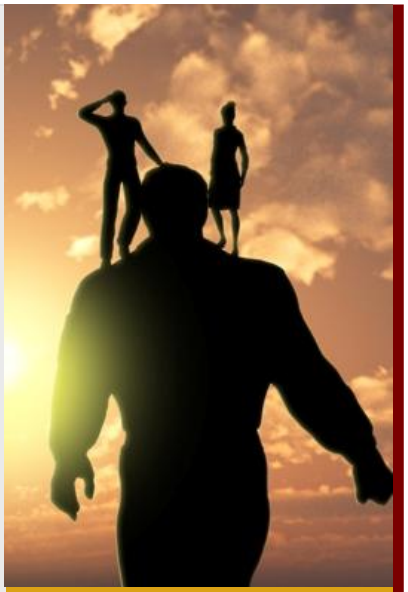
提纲

一 • 事务—历史的脉络

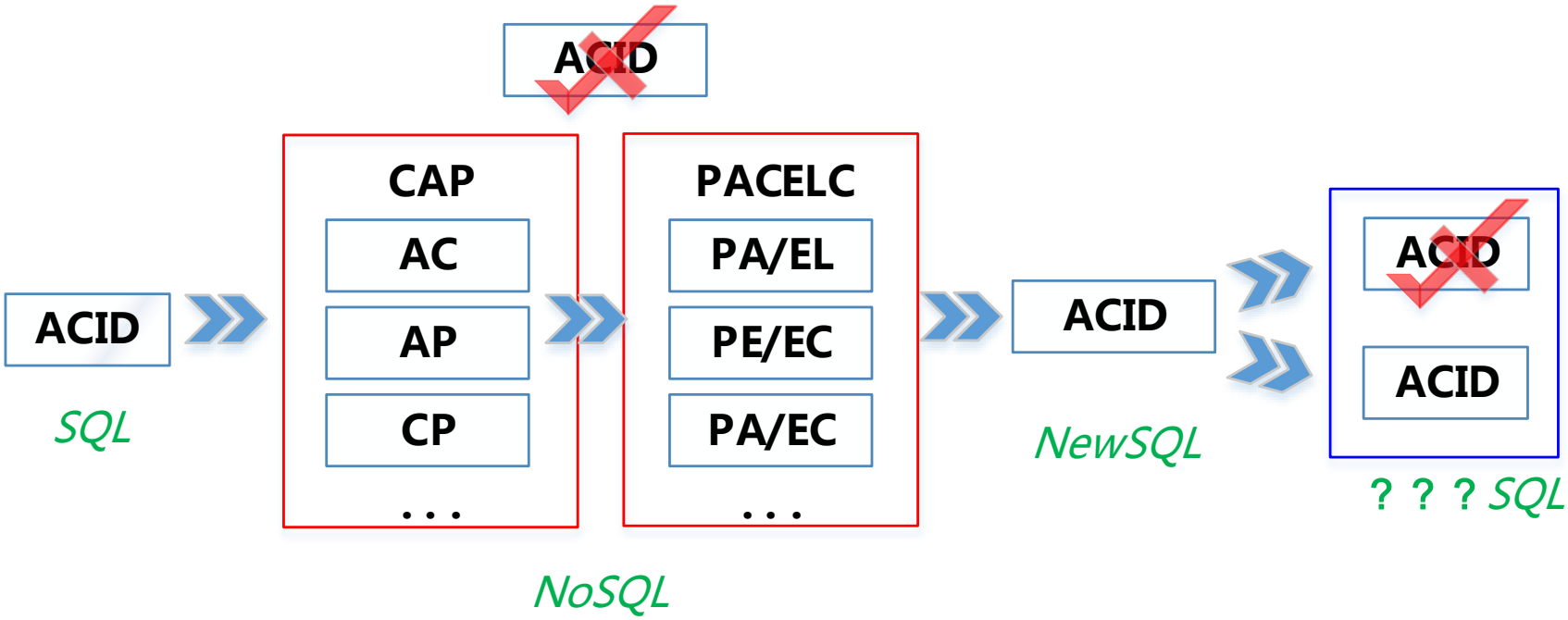
二 • 环境—酝酿变革

三 • 技术—承载发展

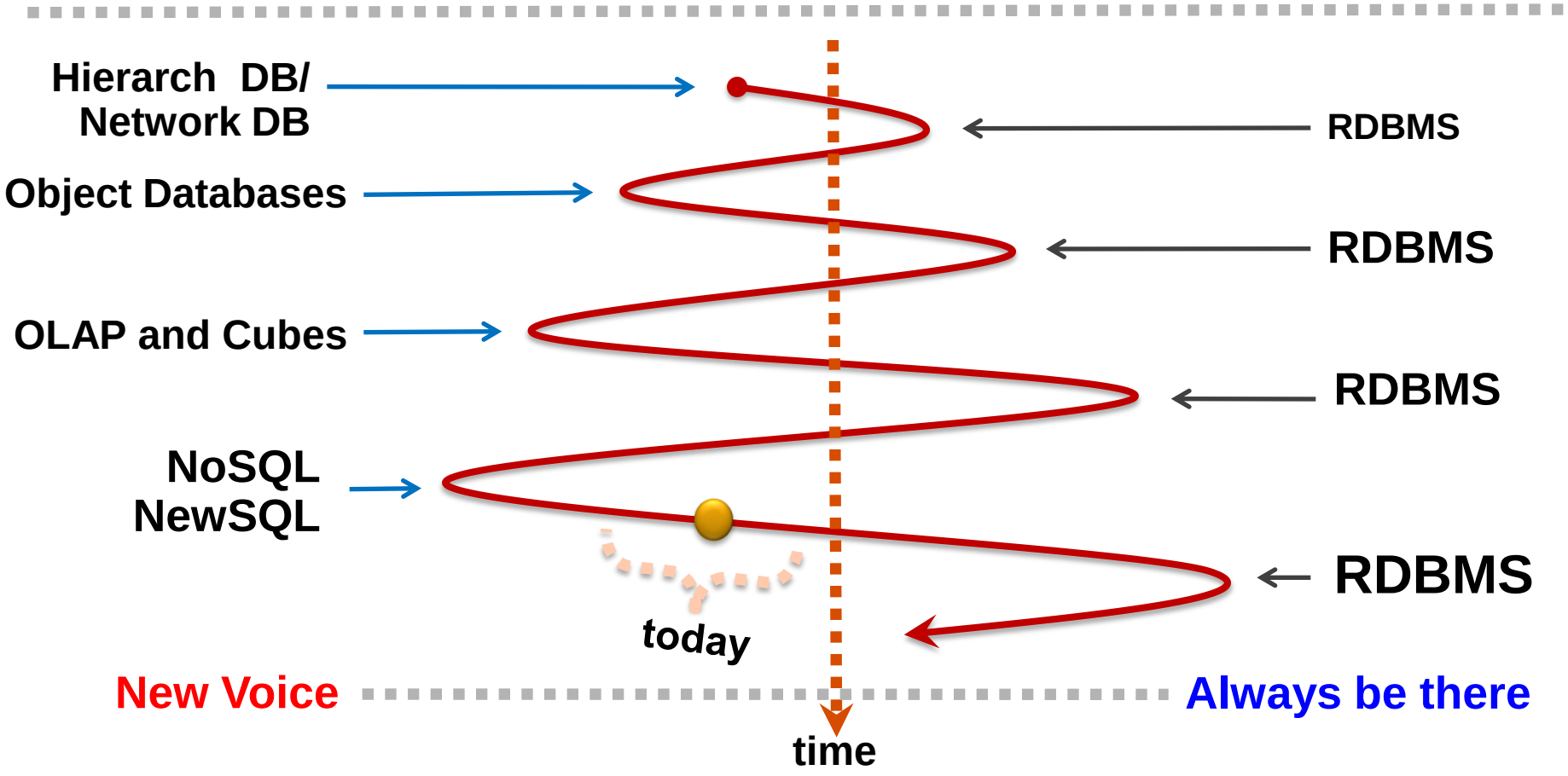
四 • 未来—顺势而为



大数据环境下数据库技术的发展趋势



大数据环境下数据库技术的发展趋势



Gartner新兴技术炒作周期(Hype Cycle)报告-2012

One Size Does Not Fit All

Attempting to force one technology or tool to satisfy a particular need for which another tool is more effective and efficient is like attempting to drive a screw into a wall with a hammer when a screwdriver is at hand: the screw may eventually enter the wall but at what cost?

--from Source: E.F. Codd et al. "Providing OLAP to User-Analysts: An IT Mandate"

- ✓ 需要站在不同的观点看问题
- ✓ 需要站在不同的角度看问题
- ✓ 大数据结合事务处理将带来巨大的变化

- 多样的大数据
- 多变的应用需求
- 多维的技术方案

需求为导，顺势而动！

Thank You

lizhh@nwpu.edu.cn