

新挑战

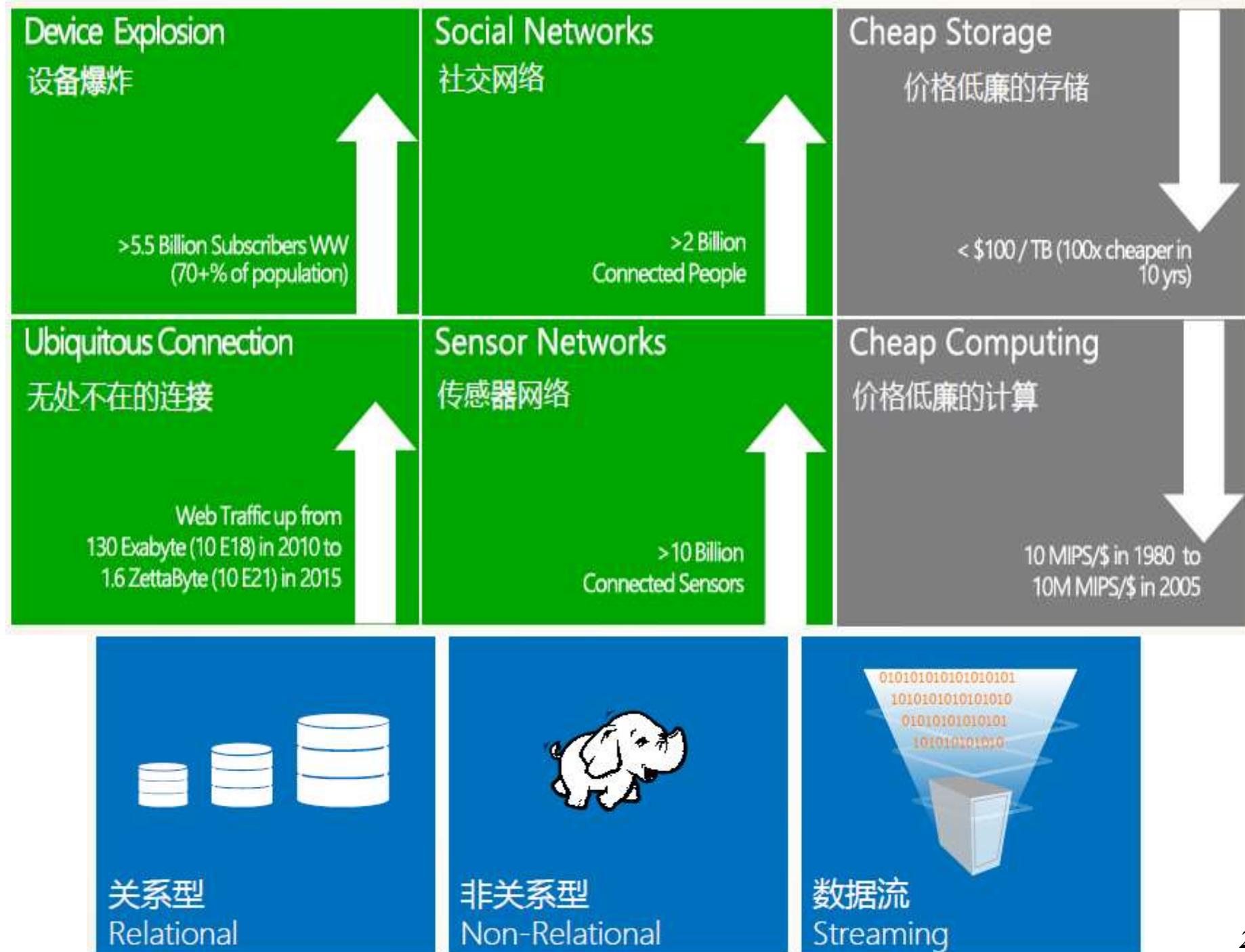
1、大数据时代及非结构化数据的处理

➤ 数据量的增加：指数级

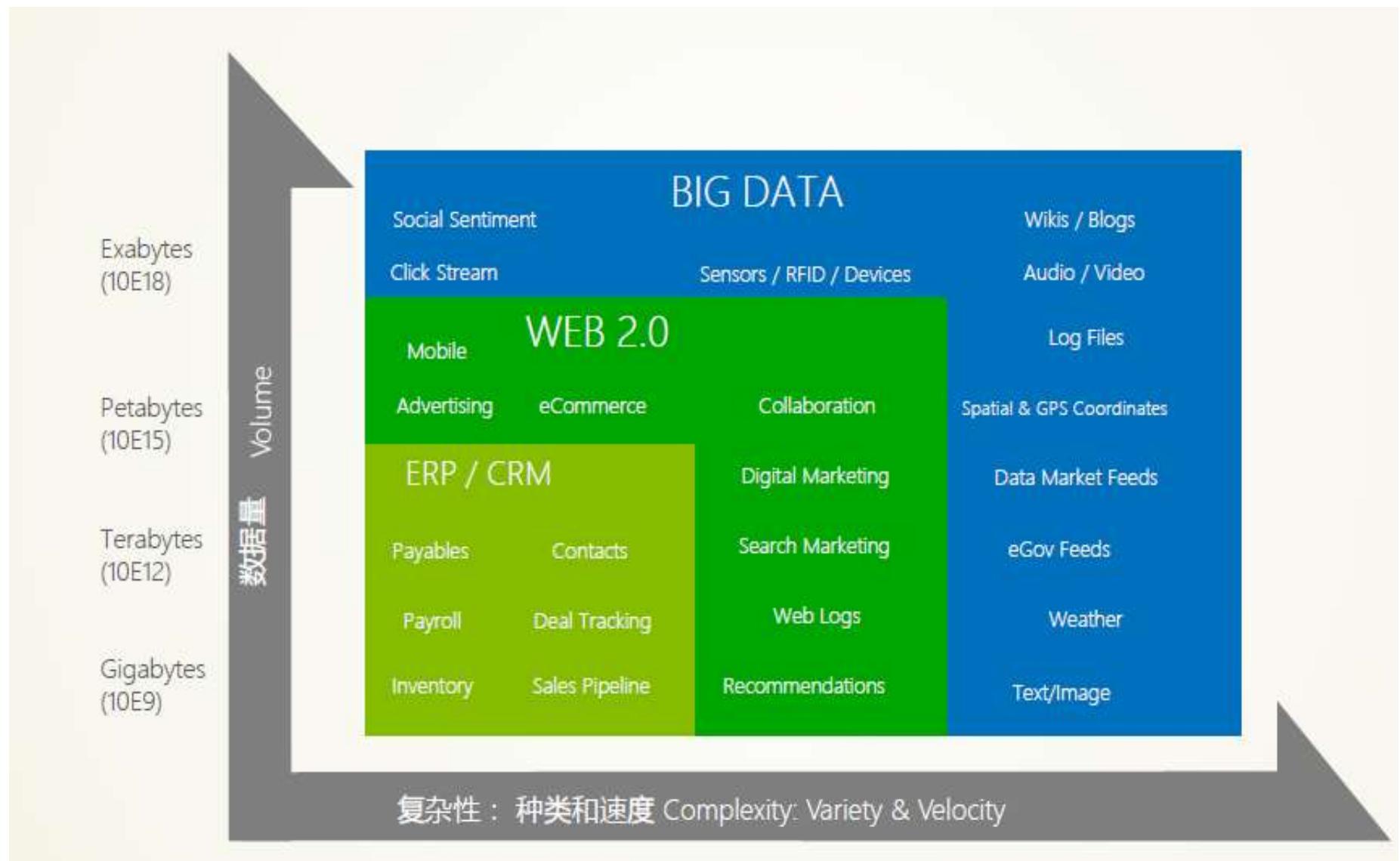
- **2011年产生了大概4个EB的数据** = 人类从公元前**3000**年到**2000**年所产生的全部数据的总和。
- 每**18**个月我们的数据会翻一番

➤ 非结构化数据：

- 社交媒体信息
- 混合存储的体系结构: 结构化/半结构化/非结构化存储
- 多种数据处理技术的融合: 数据挖掘, 文本分析, 图形图像处理...
- 增加新的计算方式: 内存计算, 分布式处理.....

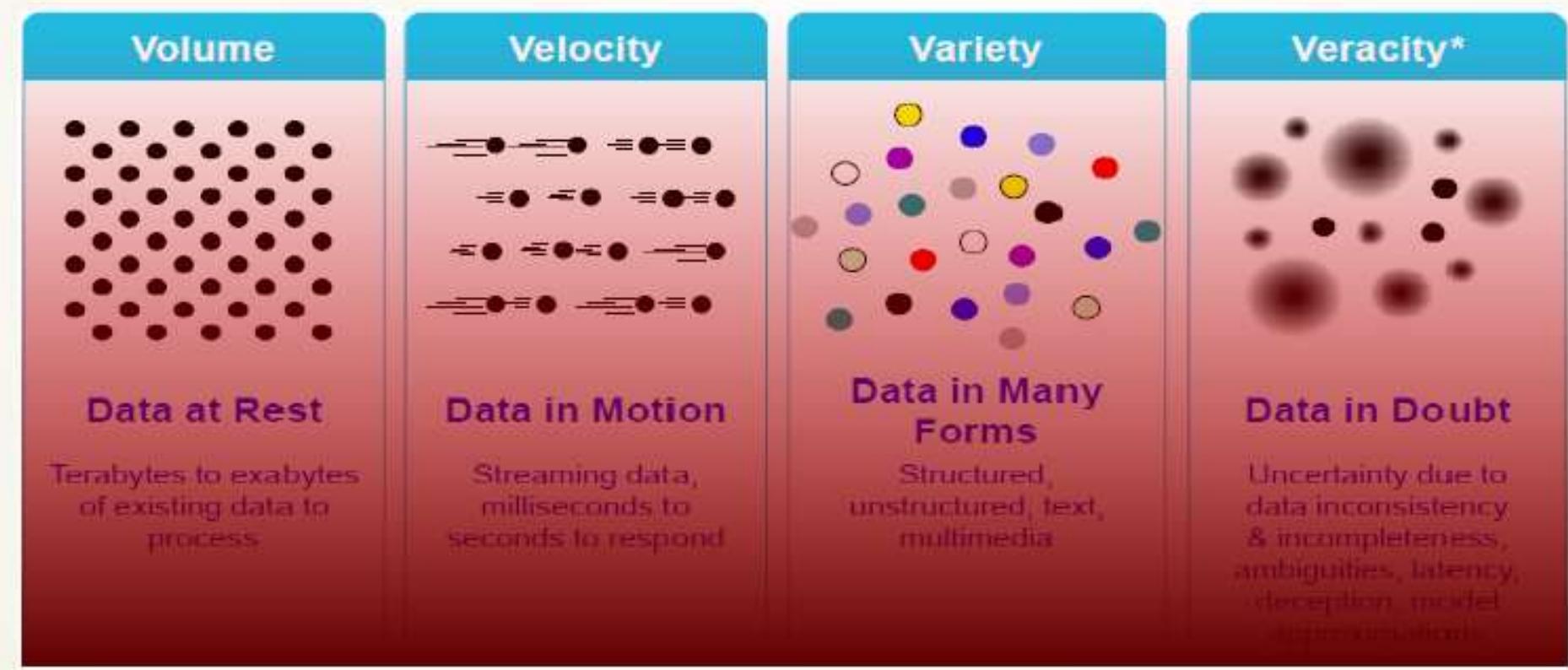


大数据



大数据的4V

1. 数据量大(Volume)
2. 速度快(Velocity)
3. 类型多(Variety)
4. 价值密度低(Value) ●



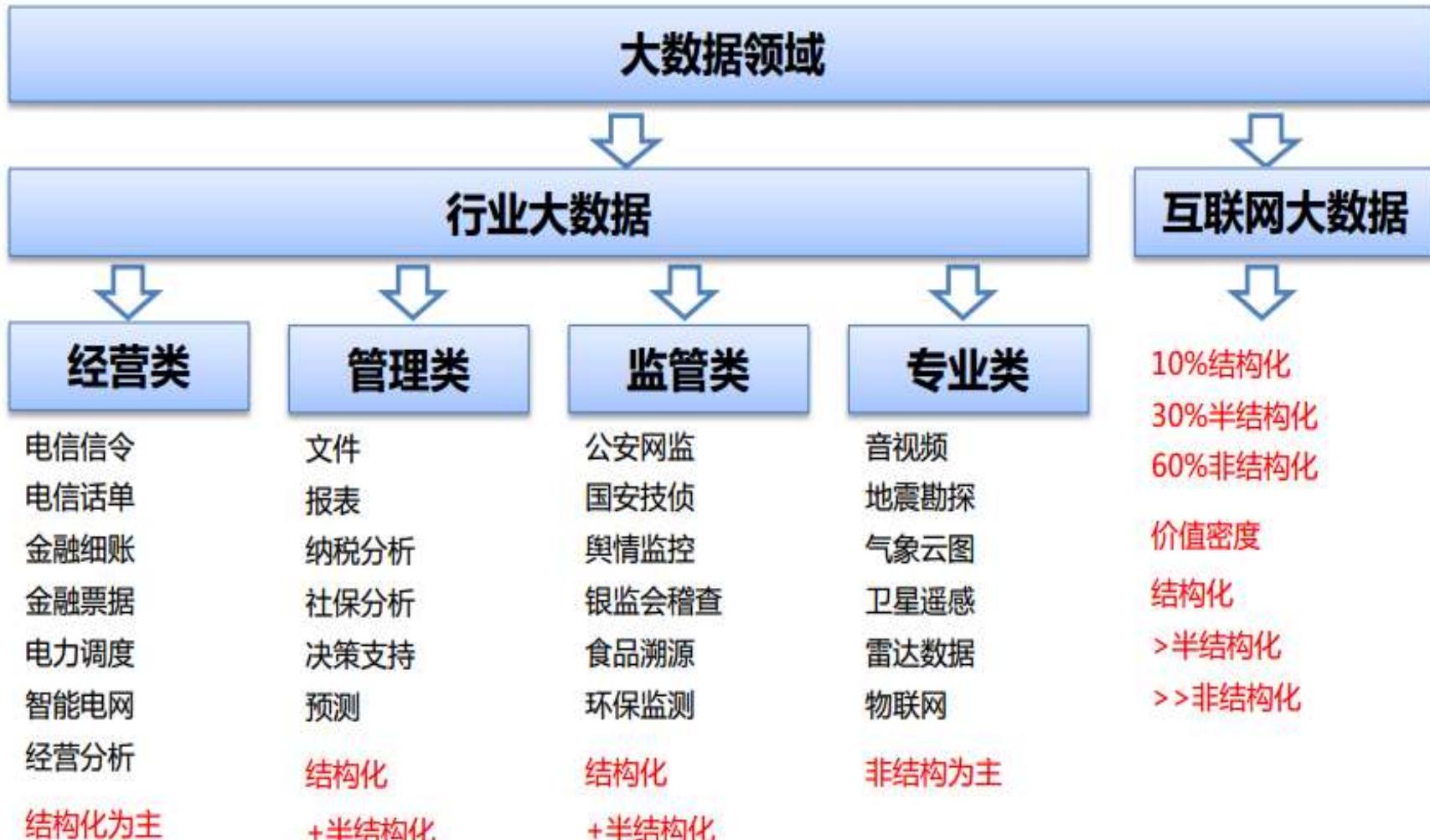
SQL的不适应性

□ SQL弱点一：必须支持Join

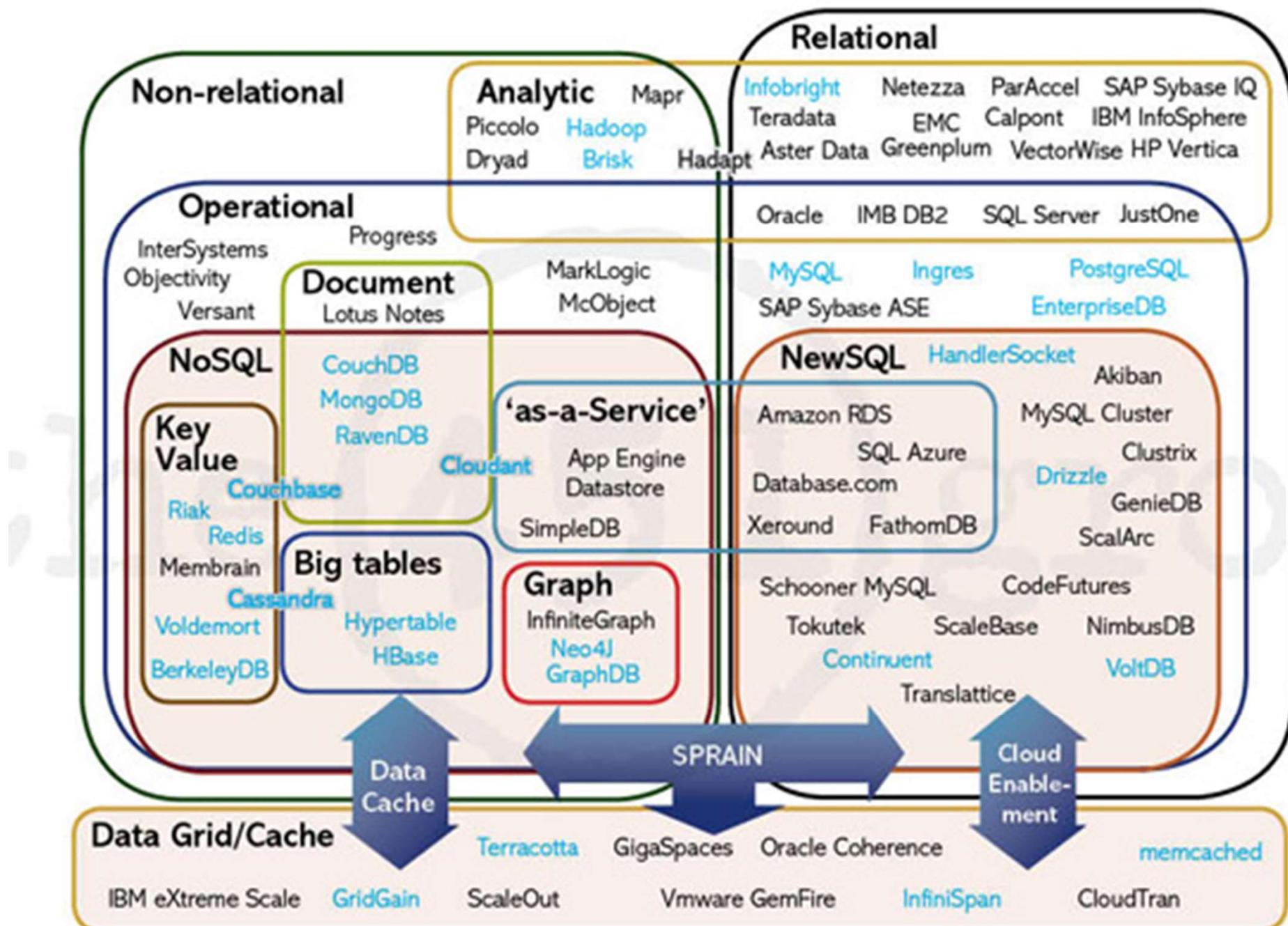
- 数据不能够有重复，所以使用范式的关系模型会不可避免的大量Join
- 如果大表Join或者表分布在多台机器上的话，Join的性能无法忍受

□ SQL弱点二：计算和存储耦合

- 关系模型作为统一的数据模型既可以用于数据分析，也可以用于在线业务
- 但前者强调高吞吐，后者强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的。

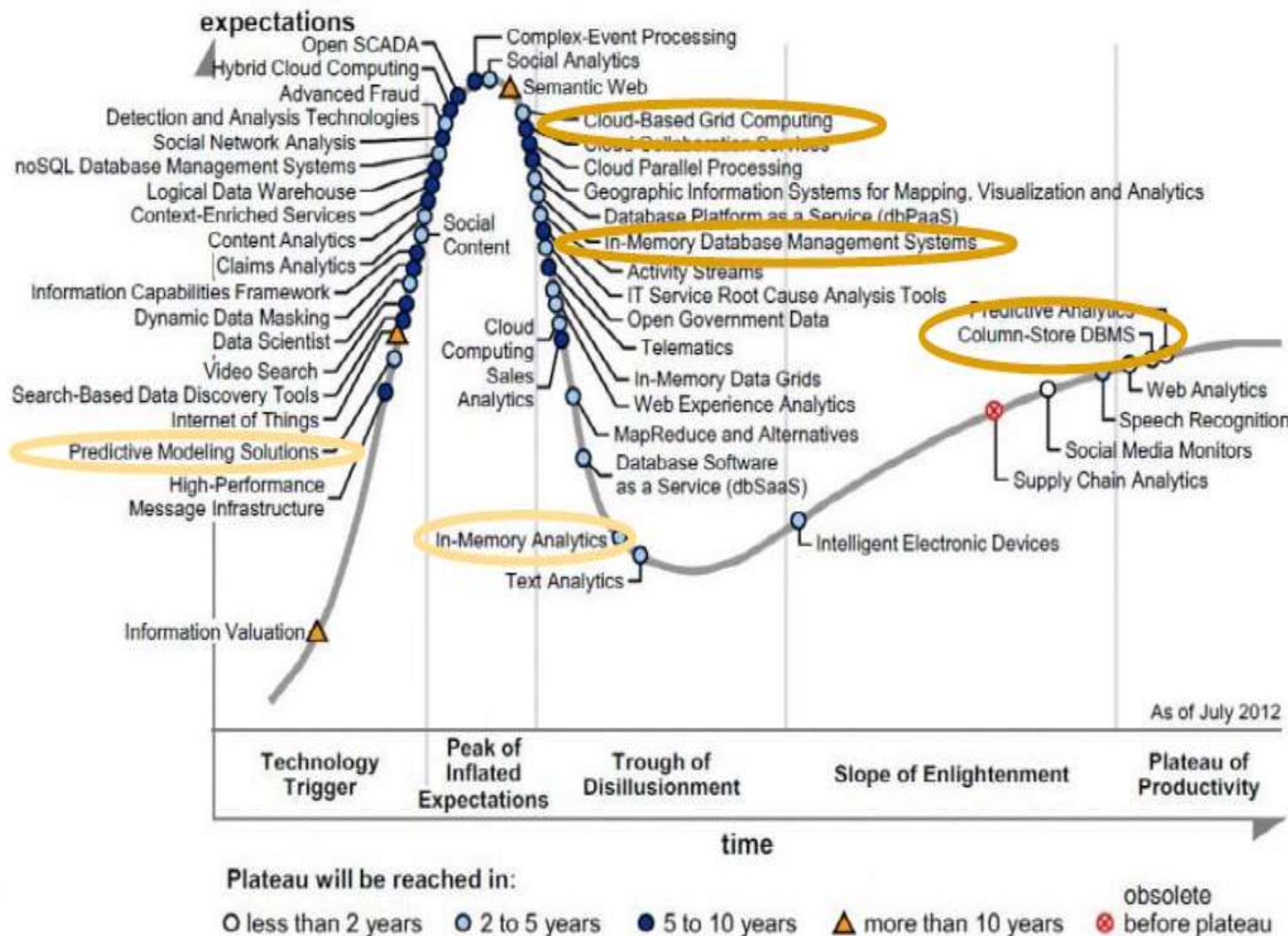


中国既有互联网大数据、更有行业大数据，是世界为数不多的数据大国。
行业大数据的价值密度、对数据库厂商的价值更大。

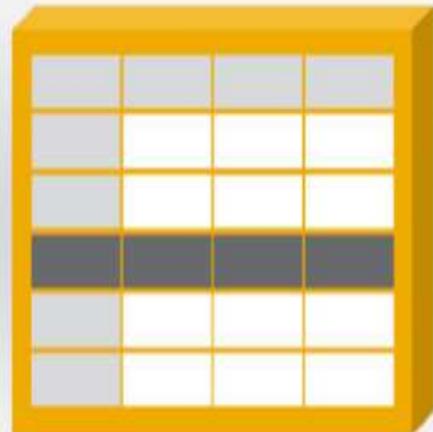


Gartner2012报告：围绕处理大数据的技术如雨后春笋

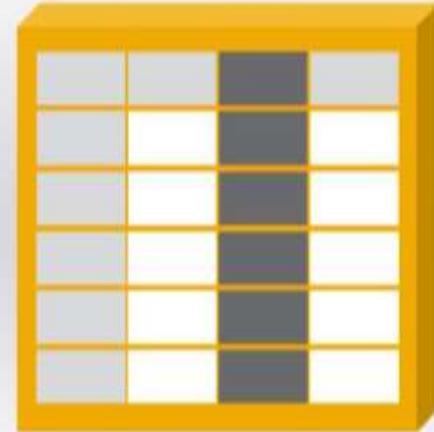
Figure 1. Hype Cycle for Big Data, 2012



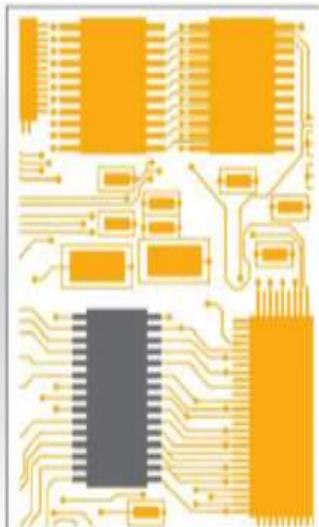
Gartner 2012年7月的分析报告认为：“列式存储数据库(Column-Store DBMS)、云计算和内存数据库(In-Memory DBMS)在未来的5年内将是3个最重要的技术”。



9 in 10 在使
用关系型数据库
的用户中有93%
在考虑其它更多
的技术手段



50%
在考虑列式数据库



63%
在考虑内存数据
库技术



50%
在考虑Hadoop

新技术

1. 内存计算技术
2. 数据库内分析 (**In-Database Analytics**)
3. 实时操作性数据复制 (**ODR**) 和变化数据捕捉 (**CDC**) 技术
4. 非关系型数据库 (**NoSQL**) 的快速发展
5. **Hadoop**

1. 内存计算技术

- 内存价格不断下降
- 突破传统的磁盘存储读写瓶颈
- 问题：
 - 云端应用的问题：计算后数据的分发
 - 散热
 - 临时内存的清理问题
 - 防御内存攻击

2. 数据库内分析 (**In-Database Analytics**)

- 将数据挖掘与分析功能植入数据库内部，帮助用户实现更快速度和更大容量的数据分析
- 避开从数据采集到数据分析的缓慢转换过程，节省数据移动的时间和成本
- 提高数据的安全性
- 数据仓库：后台战略服务  一线运营

3. 实时操作性数据复制 (ODR) 和变化数据捕捉 (CDC) 技术

- **ODR:** 实现从ETL到ELT的转变
- **CDC:** 仅捕捉和提供对企业数据源做出的变更，对源数据产生尽量小的影响

4. 非关系型数据库（**NoSQL**）的快速发展

- 可以处理超大量的数据
- 能够运行在便宜的**PC**服务器集群上
- 高并发

5. **Hadoop**技术

- **Hadoop** 的出现为**NoSQL** 快速发展奠定基础
- 占有绝对主导地位的**MapReduce**框架
- 当前的基于**MapReduce**的数据库包括
Hadoop with Hbase, Hadapt, Aster Data, 以
及**CouchDB**

1、 SAP HANA

➤ **SAP High-Performance Analytic Appliance**

2、 Oracle Exalytics

3、 IBM InfoSphere 大数据分析平台

4、 IBM Netezza

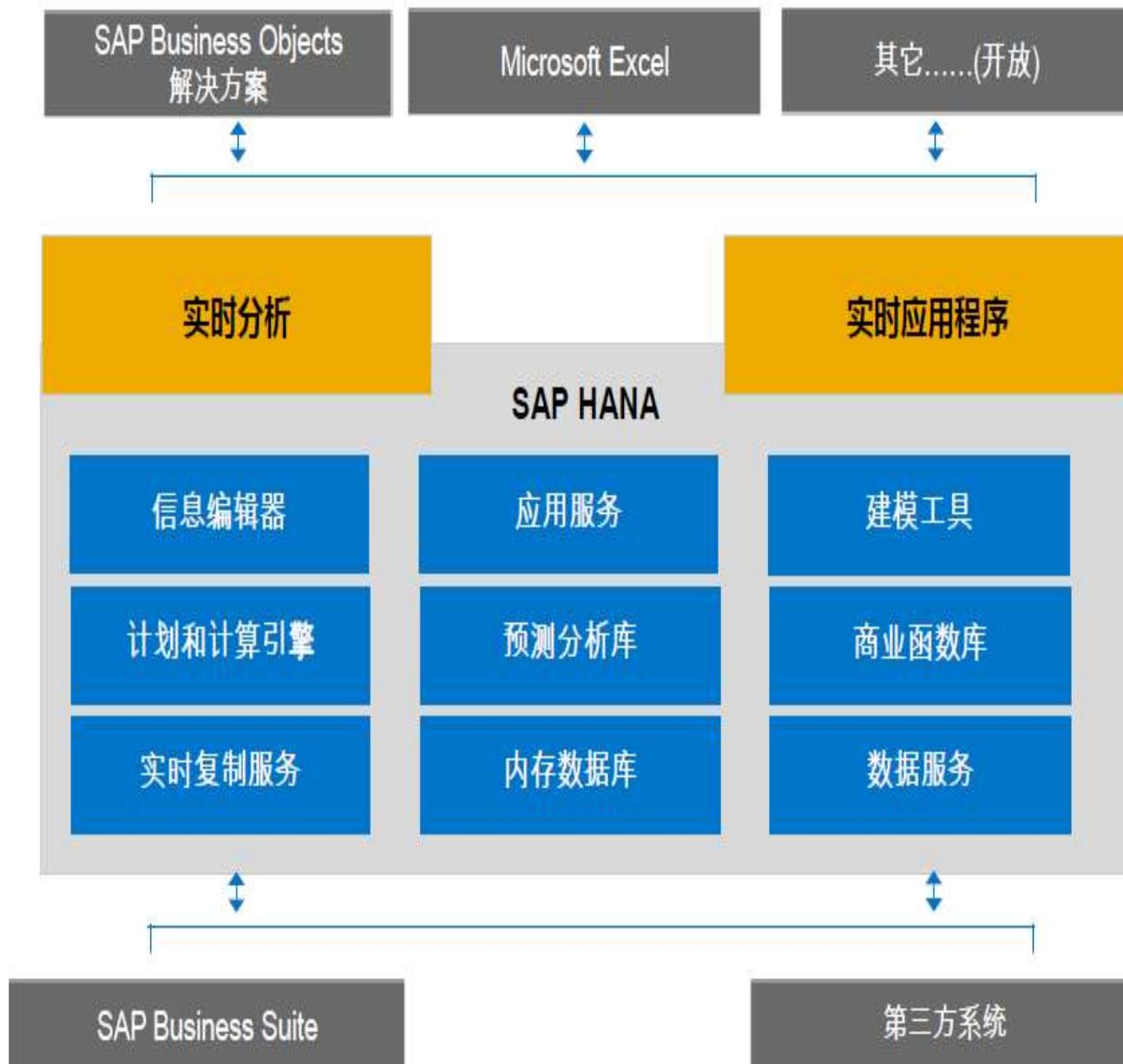
5、 Teradata Aster MapReduce

6、 EMC GreenPlum 统一分析平台

7、 Sybase IQ 15.1 数据库内分析

新产品

- 1、 SAP HANA } In-Memory
- 2、 Oracle Exalytics }
- 3、 IBM InfoSphere 大数据分析平台 } 静态磁盘数据分析 +
动态内存数据实时分析
- 4、 IBM Netezza
- 5、 Teradata Aster MapReduce } MapReduce + SQL
- 6、 EMC GreenPlum 统一分析平台 } 结构化 + 非结构化
- 7、 Sybase IQ 15.1 数据库内分析



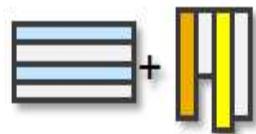
SAP HANA

SAP HANA是一种革命性的内存平台,它简化并合理化了复杂和昂贵的IT体系结构。

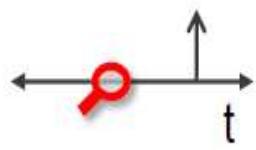
SAP HANA帮助大量数据,并以前所未有的速度提供信息,比以前快了1万倍

。 SAP HANA是一个开放平台:适应性和可扩展性,使能够创建之前无法设想的应用程序,并重新思考和设想新的方法来运行业务

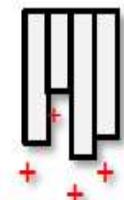
SAP HANA技术特性



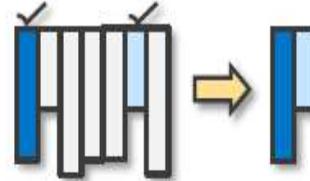
Column and
row store



Analytics on
historical data



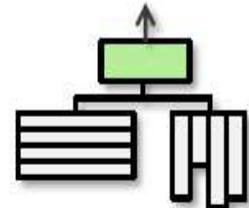
Insert only
on change



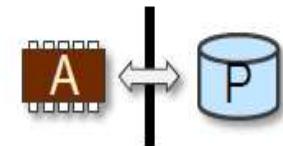
Minimal
projections



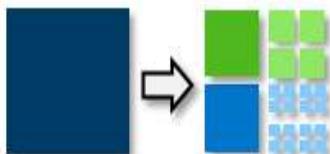
Text Retrieval &
Exploration



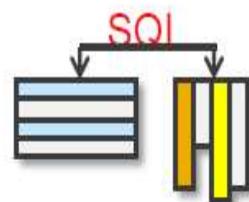
No aggregates



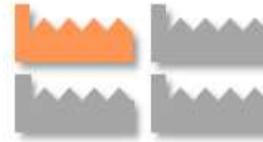
Active/passive
& data aging



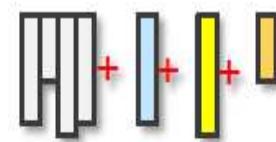
Partitioning



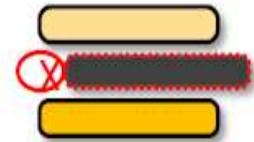
SQL interface on
columns & rows



Single and
multi-tenancy



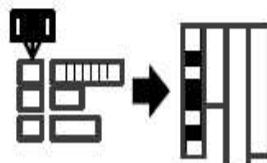
Dynamic
Extensibility



Reduction of
tiers / layers



Map reduce



Group Key

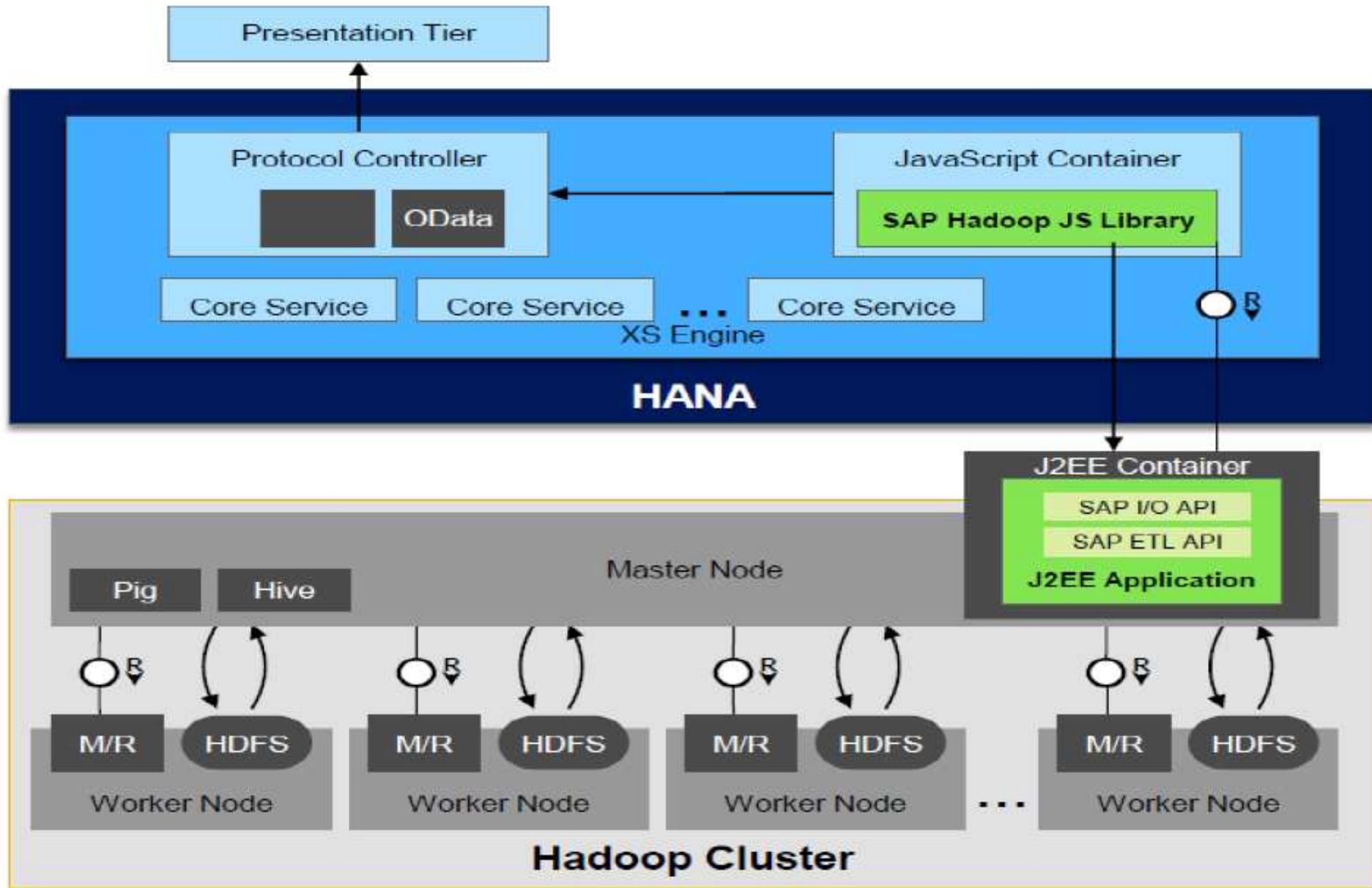


In-memory Apps



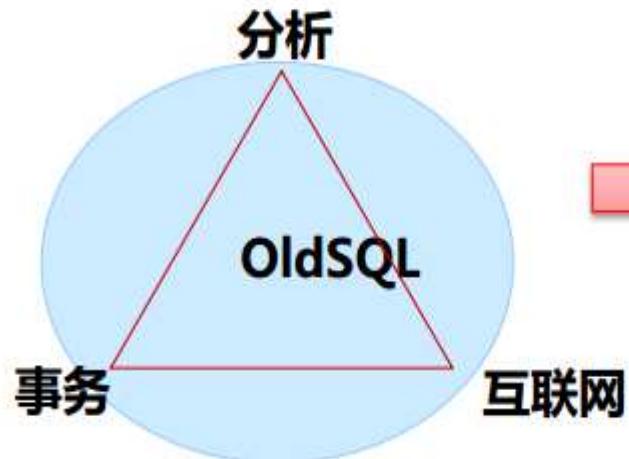
Bulk load

SAP HANA Hadoop整合方案

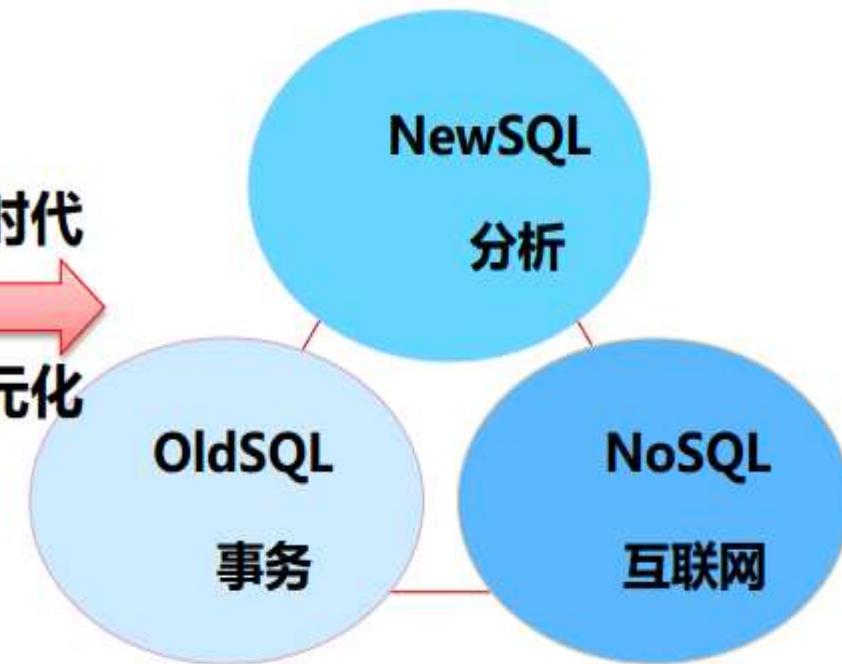


大数据引发的新变革

一种架构支持多类应用
(One Size Fits All)



多种架构支持多类应用



M. Stonebraker

基于Stonebraker教授的论文。传统数据库的基本架构是30年前以事务处理为主要应用设计的。大数据的主要应用是分析类的，应采用新的技术架构。行业的技术大思路应该由“一种架构支持所有应用”转变成“多种架构支持多类应用”。数据库行业出现三个互为补充的三大阵营，OldSQL、NewSQL和NoSQL。

(斯教授主创的数据库产品Ingres、Informix、PostgreSQL和Vertica)

NewSQL

- 列存储
- 关系型
- MPP

NoSQL

- Key-Value
- MapReduce
- MPP

OldSQL

- 行存储
- 关系型
- SMP

分布式计算，分布式文件系统

内存计算 (In Memory Computing)

新的硬件 : Flash Card , SSD , Infiniband (40G/s)



大数据激发了数据库行业技术创新的热情，主要的驱动力是对处理性能的强烈需求。为了提升性能，NewSQL阵营普遍采用了列存储技术；NoSQL阵营普遍采用了KV技术。三个阵营都不同程度地采用了分布式计算、分布式文件系统、内存计算技术，并积极地使用新的硬件技术，如大内存、Flash、SSD和高速网络连接（万兆交换机和Infiniband）

NewSQL

- GBase 8a
- Greenplum
- Vertica
- AsterData
- Sybase IQ
- F1/Spanner

NoSQL

- Hadoop
- HBase
- Bigtable
- Cassandra
- Dynamo
- Dremel
- Impala

OldSQL

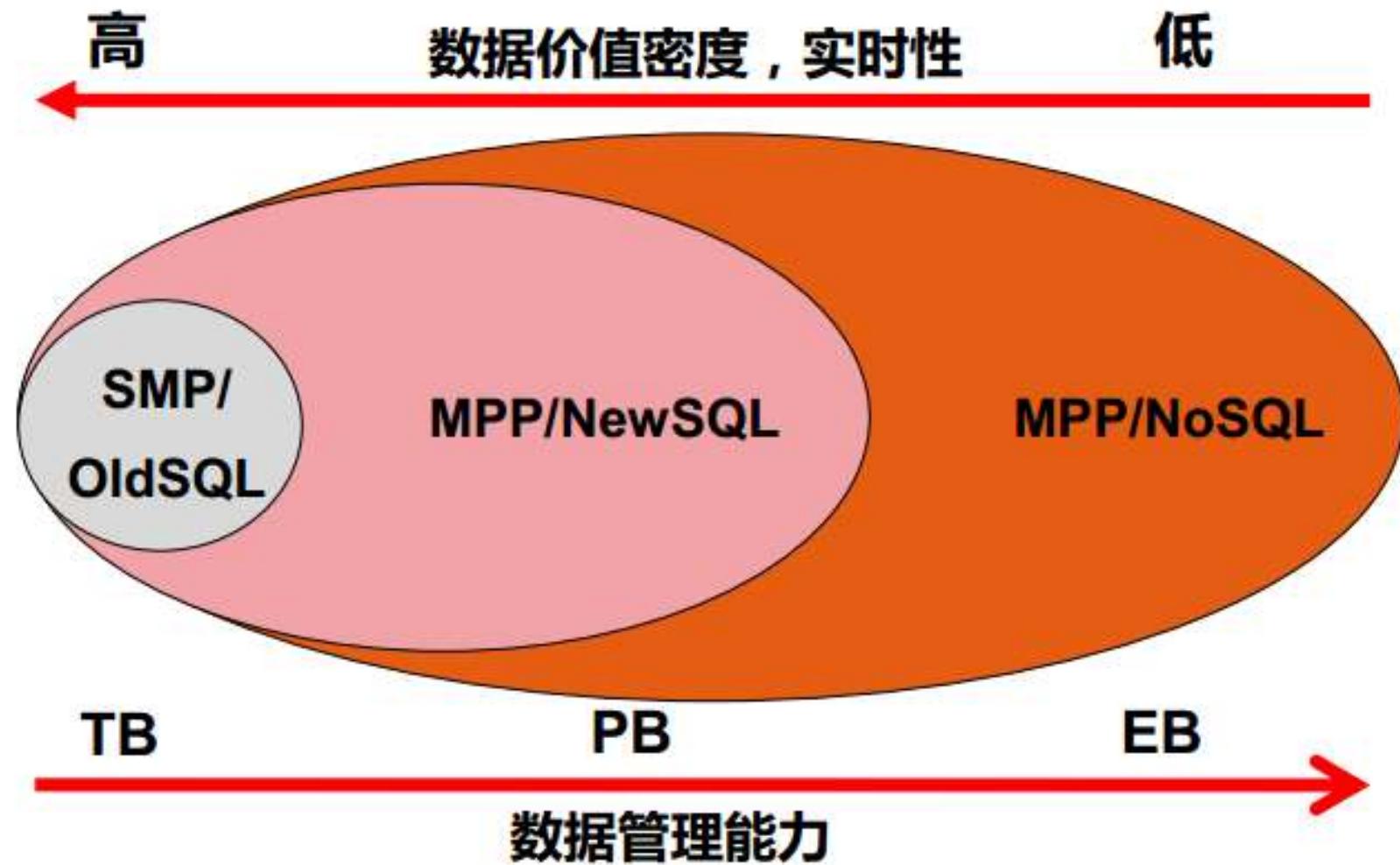
- TimesTen
- Altibase
- SolidDB
- Exadata
- Netezza
- Teradata



大数据推动了数据库行业的产品创新，NewSQL阵营在过去五年里形成了近十个商用的产品，去年Google发表论文介绍了F1/Spanner关系型数据库（未开源）。NoSQL阵营的技术源于互联网公司Google，Yahoo，Amazon，Facebook等。OldSQL阵营在基本架构不变的基础上引入内存计算和一体机技术以提升处理性能。

		公司	产品	技术特点	产品来源
NewSQL	国外厂商		Sybase IQ	列存 + 共享磁盘	收购 (2010)
			HANA	内存数据库 + 列存	收购 (2009)
			Vertica	列存 + MPP	收购 (2011)
			Greenplum	行存 + 列存+MPP集群	收购 (2010)
			PDW	列存 + MPP集群	预计2013年初
	国内厂商		GBase 8a	列存 + MPP集群	国内唯一(2008年)
NoSQL	国外厂商		BigTable	列存 + Key Value	Google (2004年)
			HBase	列存 + Key Value	开源社区 (2006年)

发展趋势



一、并行数据仓库技术

- 大规模并行、非共享（**MPP**）架构数据库平台
- **MPP**代表“**massively parallel processing**”，在多台机器或多个主板上的多处理器执行的关系数据库。
- **Teradata, Netezza, Greenplum**, 以及**DB2**数据仓库版
- **MPP**数据库适用于数据挖掘和分析
- 迄今为止，所有产品级的**MPP**数据库是专有的

一、并行数据仓库技术

- 去小型机化 》基于x-86的云化改造



从SMP 到 MPP



- 逐步取代传统数据库 》使用新型MPP分布式数据库
 - 小型机的垂直扩展能力已经达到极限
 - MPP集群的横向扩展能力优势明显：互联网证明了一切
 - 新型大数据平台在大数据处理上的价值明显：

对比项	SMP (特殊环境)	MPP (开放x-86环境)
每TB数据处理成本	20-50万元	2-5万元
系统处理能力	100TB	1PB
应用支撑能力	单一、固定	多样、弹性

MPP 架构的横向扩展能力

80 台 2U 普通PC Server 意味着什么样的处理能力 ?

内容	测试平台	最大可能容量	
CPU	1280 核	1280核	2*8 cores
内存	5TB	20TB	
内部存储容量	281.25TB	562.5TB	15k , 600GB SAS
I/O 带宽 (读)	62.5GB/秒	80GB/秒	
I/O 带宽 (写)	31.25GB/秒	40GB/秒	
IOPS	80万	100万	
网络带宽	800MB/s (使用10K Ethernet)	2GB/s (使用IB)	

Greenplum: 新一代分析型云数据库



- 由世界级的技术和市场专家组成的团队
- 高端,大规模数据仓库、数据分析的领导者
- 提供超高性价比的数据分析平台(OLAP)
- 新一代数据仓库架构“Enterprise Data Cloud 数据云”平台的先驱

- Architect, Teradata Optimizer
- Architect, Tandem Optimizer
- Architect, MS SQL Server Optimizer
- Architect, Oracle Bit-Map Index
- Architect, Oracle OLAP
- Architect, Informix Bit-Map Index
- Architect, Tandem Transaction Manager
- Architect, MS SQL Server Transaction Manager
- Architect, MS SQL Server NLP

“我们认为Greenplum是数据仓库软件技术的领导者。”

- Steve Hirsch, Chief Data Officer, NYSE Euronext

Greenplum 的竞争优势

□ 高性能、高性价比

- 基于X86 开发式平台，OS支持Redhat、SuseLiunx、Solaris

□ 易用性

- 并行处理由系统自动完成 - 无需人工干预
- 没有复杂的调优需求 - 只需加载数据库和查询

□ 扩展性

- 可线性扩展到10, 000个节点，X86 PC Server 低成本扩展
- 支持在线扩展
- 每增加一个节点，查询、加载性能都成线性增长

□ 高可用性

- 多层级容错保护

□ 灵活性

- 完全并行处理支持 SQL92, SQL99, SQL2003 OLAP, 列数据库, 透明压缩, MapReduce
- 支持任何schema (star, snowflake, 3NF, hybrid, etc)
- 丰富的扩展性和语言支持(Java , Perl, Python, R, C, etc)

案例一：中国电子商务领导者——阿里巴巴

• 业务使用

- 通过分析用户的网络点击日志，进行产品关联分析，让客户可以快速的找到相近产品

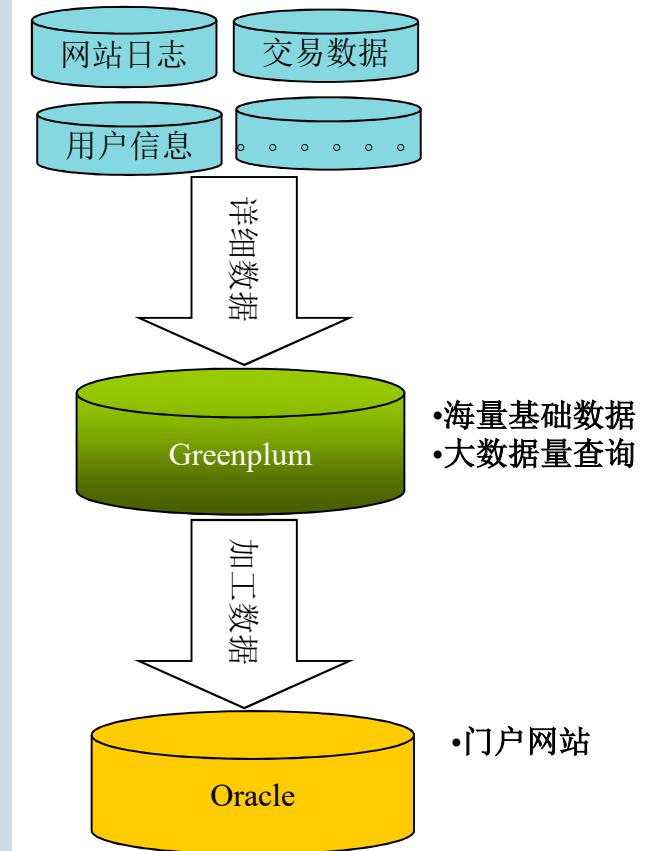
• 原有解决方案及问题

- Oracle RAC (2008)
 - 加载速度非常慢，真的令人无法接受——技术人员天天抱怨
 - 做客户详细复杂的点击查询，要等上半天到一天，有时还出不来结果，浪费我们大量的时间——业务经理已经忍无可忍了
 - 现在系统无法满足海量的历史数据的分析应用

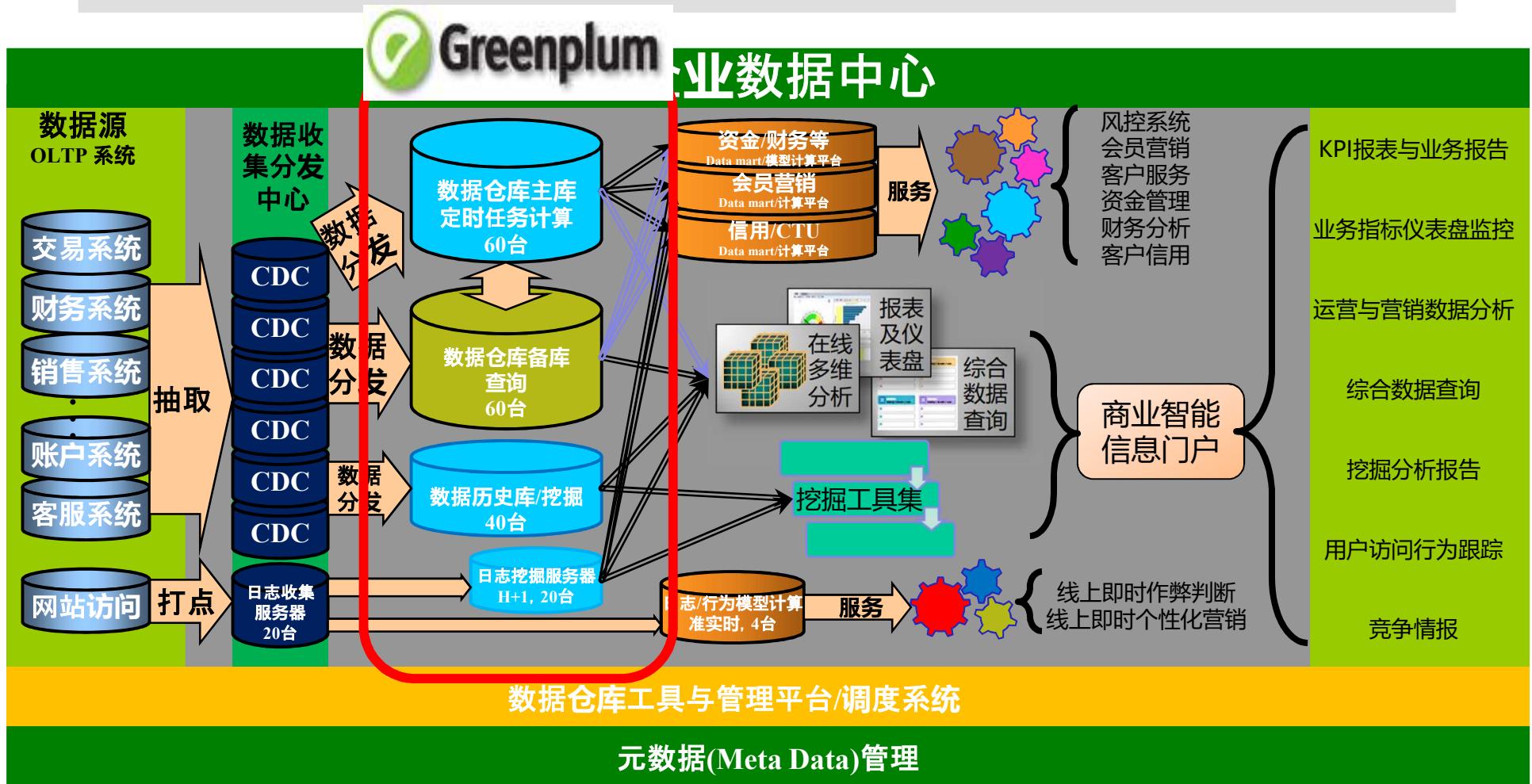
• Greenplum是幕后的英雄

- 海量加载由Greenplum完成
 - 汇集了Alibaba.com/支付宝等所有的历史数据

真是一个超级的海量数据库软件！
——客户技术部门评价



支付宝BI 2010系统架构图



图例



已经上线greenplum集群, 总共120台

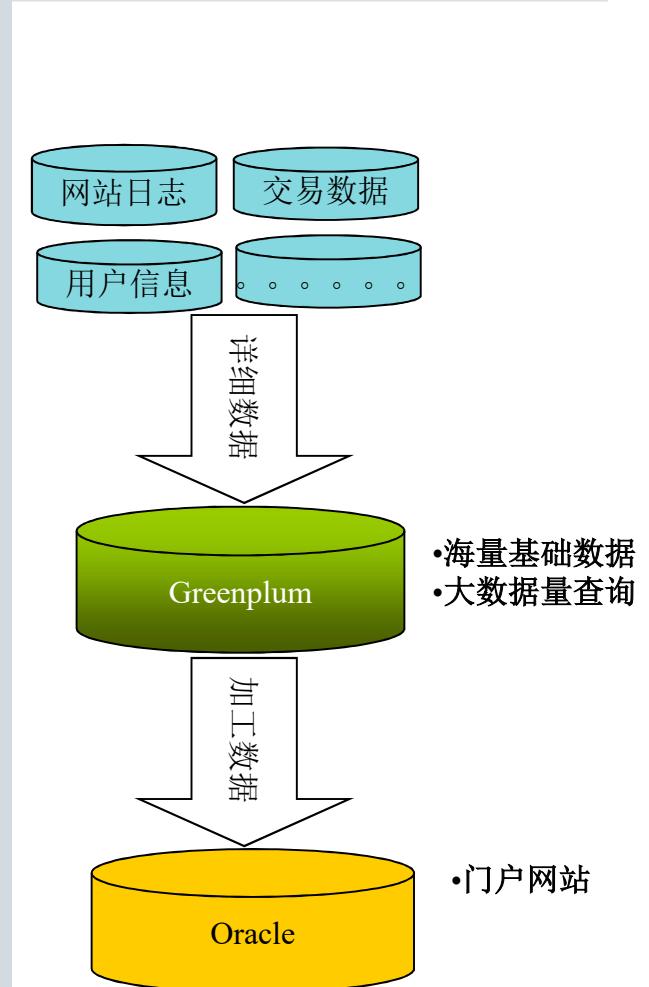


即将上线查询集群 总共60台

案例一：支付宝使用情况说明



- **数据库数据量**: 400T
- **数据库数据增量**: 500G/天, 每年数据量增加2倍以上
- **数据加载频度**: 大多数H+1(每小时加载), 网站访问D+1(每天加载)
- **现有系统规模**: 120个节点, 3套Greenplum数据仓库集群
- **节点服务器**: 华赛T3500 PC 服务器, 每台2个Intel 四核CPU, 32G内存, 24个1TB SATA硬盘, 配置为RAID5, 操作系统是Redhat Enterprise Linux
- **OLTP生产环境**: 约200多套 Oracle OLTP库, 每个应用采用分20个分库的结构部署。
- **ETL/CDC**: 自主开发, 负责把200多个OLTP的库的数据采集过来, 做汇总, 然后同时导入3个Greenplum 数据仓库集群中, 各个数据仓库保持相同全量数据。
- **Greenplum**: 在线扩容。2008年上线时是一个集群20台PC服务器。随着数据量增加, 随时扩加机器, 现在最大的集群已经加到60台。
- **BI中间件**: 现用SAP BO, 正在和Actuate 谈更换为BIRT



案例二 全球最大的电子商务商eBay也采用GP



- 业务需求

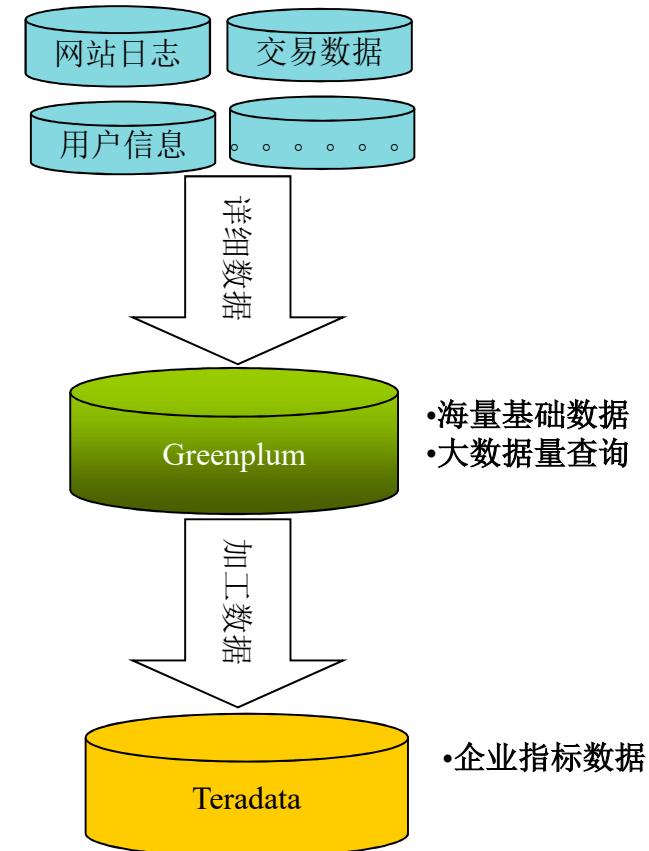
- 分析eBay整个系统采集的详细的历史事件数据，分析功能：有业绩分析、点击分析、欺诈监测等

- 现有设施

- Teradata

- 特征

- 世界上最大的数据仓库
- **6.5 PB 数据量, 每天增长18 TB**
- 2 Master 节点使用Sun x4540
- 96 Segment节点使用72 Sun x4540 and 24 Sun x4500
 - 16 ETL 节点使用 Sun x4540
 - 采用1TB 7.2k rpm SATA硬盘
- 使用Solaris OS 及 ZFS 、RAID Z
- 采用Greenplum 实时压缩 (1:4)
 - 启用Segment Mirroring



eBay Singularity 数据中心



•业务需求

证券交易的合法性及安全性监控

•被替代厂商

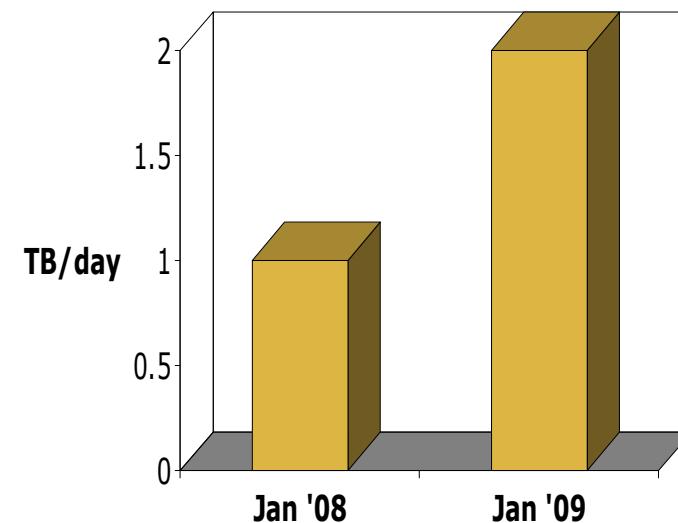
Oracle, Netezza

• Data Size

数据量400T，日增量数据从 1TB 增长
到 2TB/day，34个节点

•Benefit

高速的查询性能，满足海量数据的高度复
杂分析



*"Greenplum is reaching data loading speeds of over **three terabytes per hour**, and we know that the database can scale even further than that. Greenplum's fast performance is critical for us."*

----Steven Hirsch, Chief Data Officer, NYSE Euronext

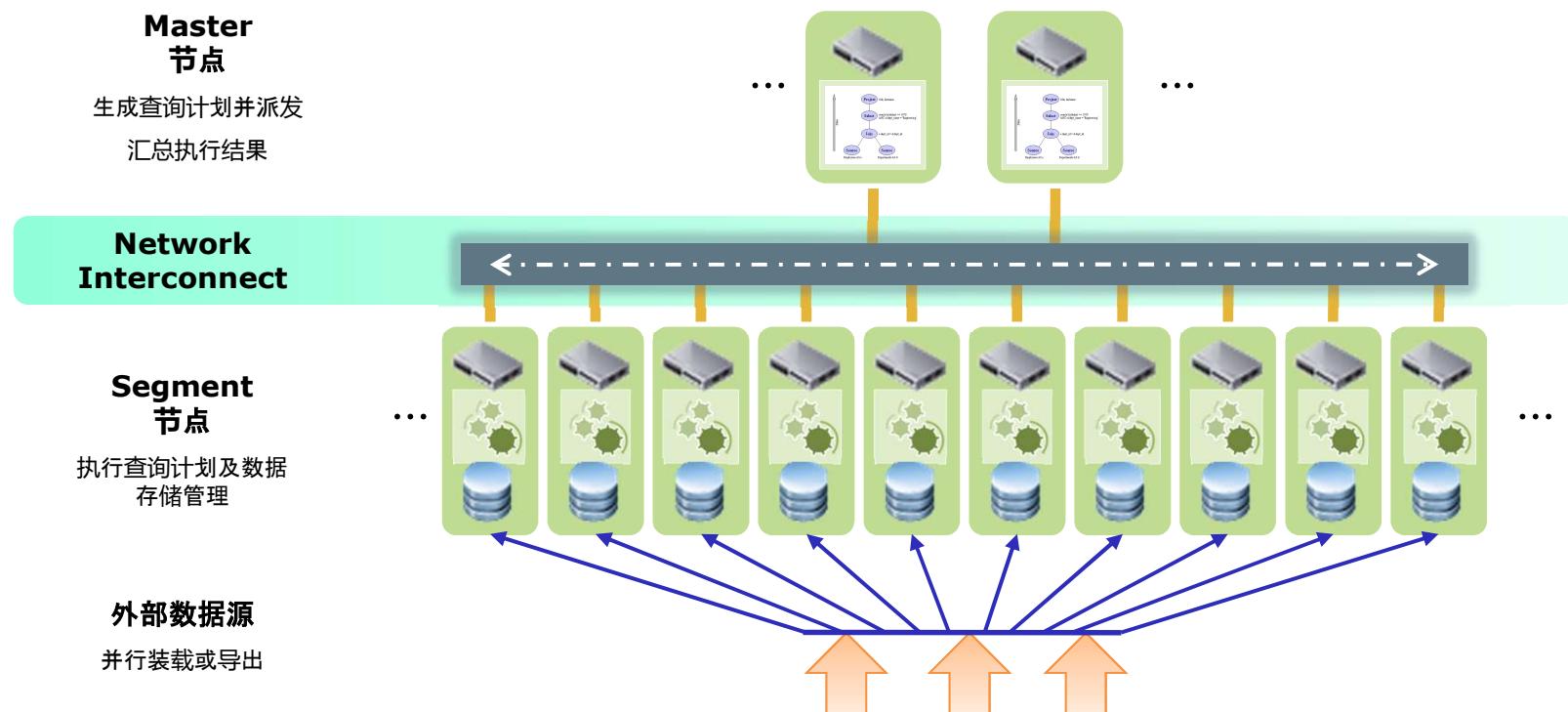
“大” 数字—当前GP客户生产系统的工作状态

- 20 万亿(Trillion)行 - 事实表 (fact table)
- 70K/day - 查询速率 (Query Rate)
- 6.5PB - 数据尺寸
- +100GB/s - 分析数据速度 (Analysis Rate)
- +3GB/s - 装载速度
- 100,000/s - 交易处理速度
- 56 TB / kW, 1.6 GB/s/kW - 功耗效率
- 1000s - 节点规模

大规模并行处理

MPP (Massively Parallel Processing)

无共享架构 Shared-Nothing Architecture



Greenplum

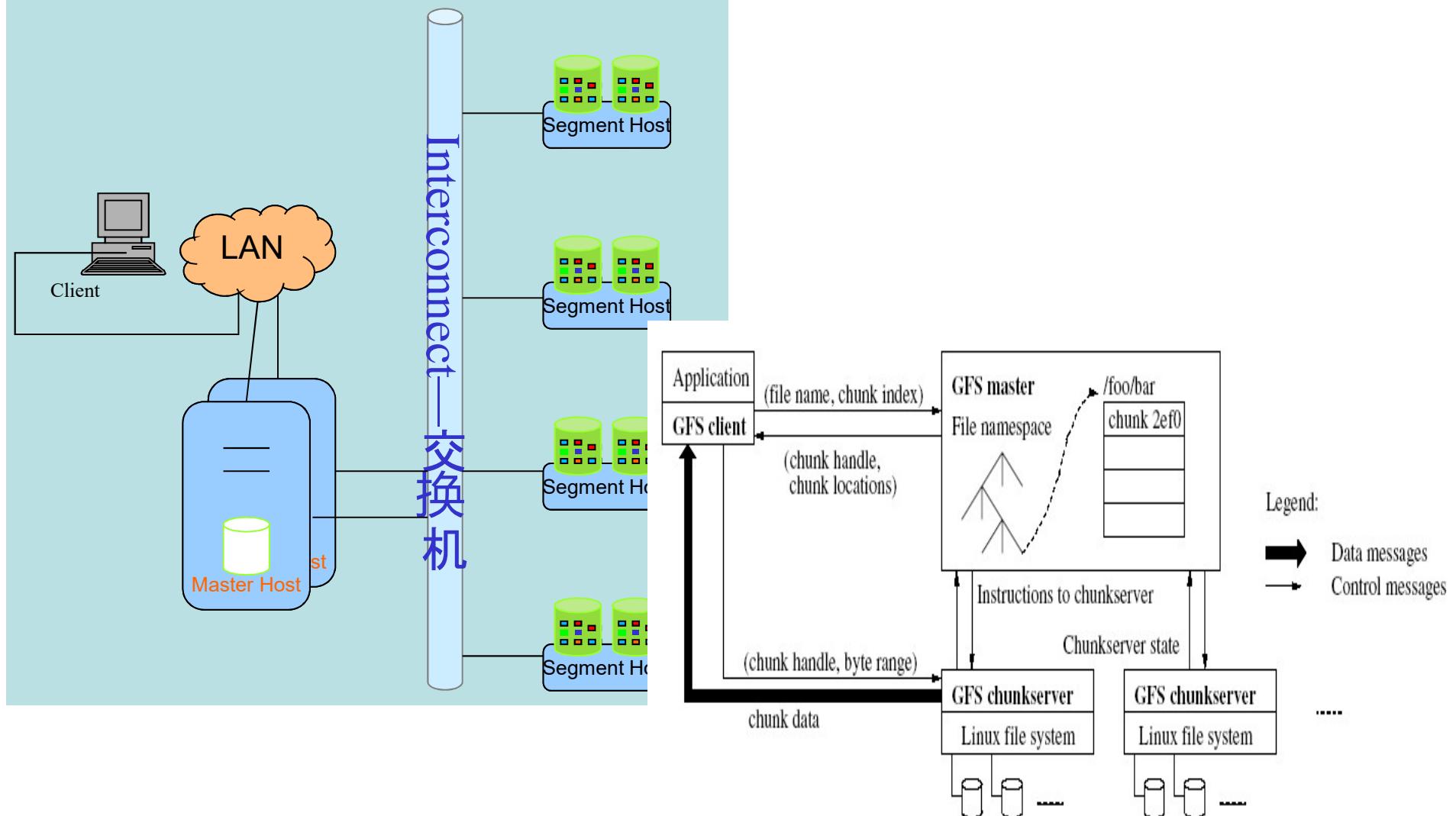
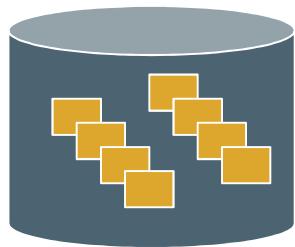


Figure 1: GFS Architecture

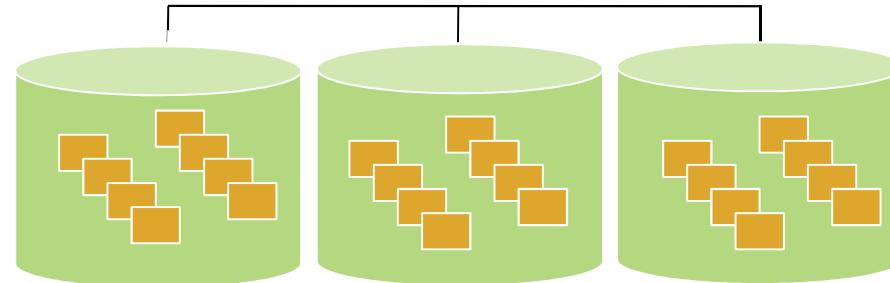
Greenplum - 云计算模式的新方案

现在的解决方案



- “黑盒子”
- “大铁箱”
- 大磁盘

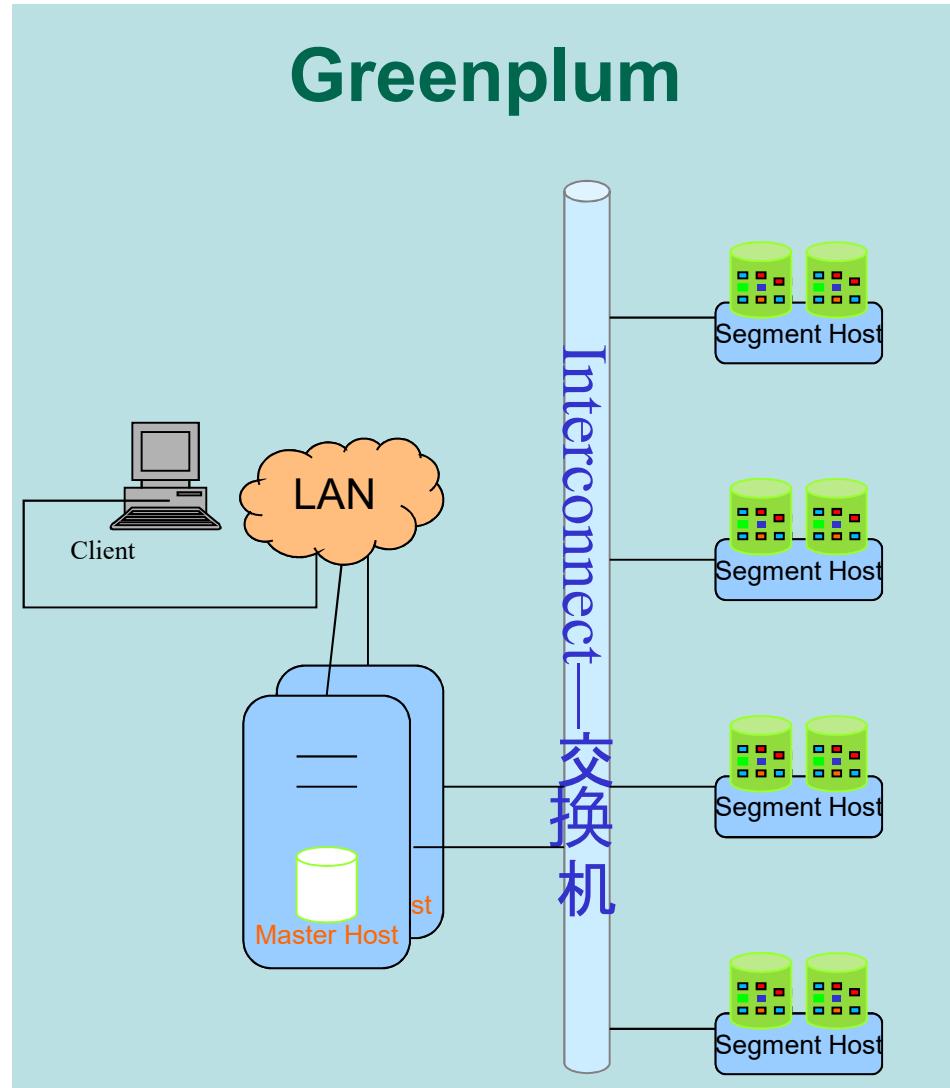
Greenplum



- 普通服务器平台（服务器、网络）
 - 通过软件提升处理能力

过去Google 曾经用来实现信息搜索功能的技术，
现在被Greenplum用于数据管理领域

Greenplum数据库内部架构



□ Master节点负责:

- 建立与客户端的连接和管理
- **SQL**的解析并形成执行计划
- 执行计划向**Segment**的分发
- 收集**Segment**的执行结果
- **Master**不存储应用业务数据，只存储数据字典

□ Segment节点负责

- 业务数据的存储和存取
- 用户查询**SQL**的执行

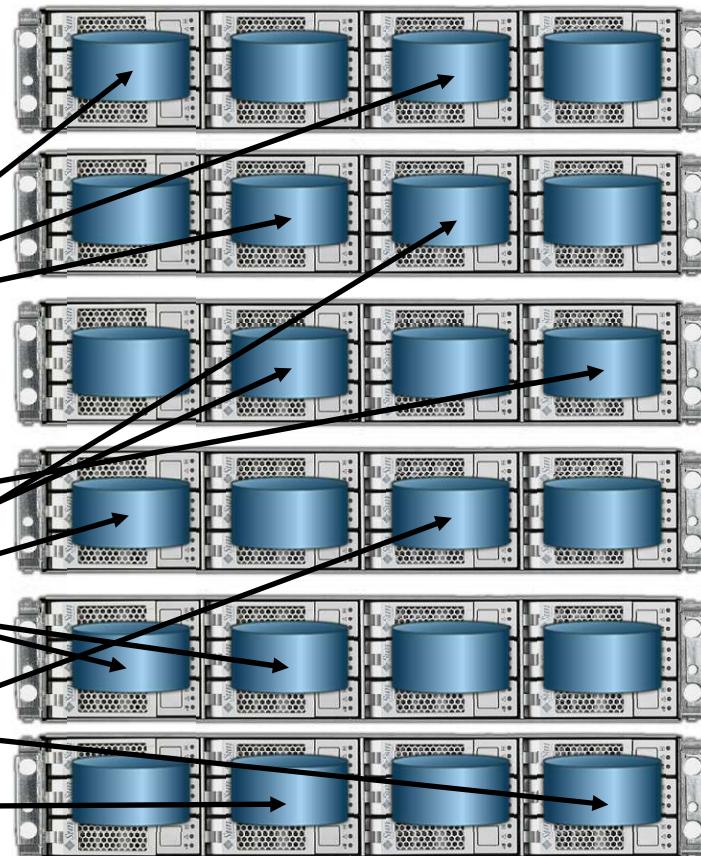
数据均匀分布

- 并行处理的关键

策略：数据尽可能的均匀分布到每个节点，为并行计算提供分布式存储的基础；

每个Segment节点存储数据的一部分

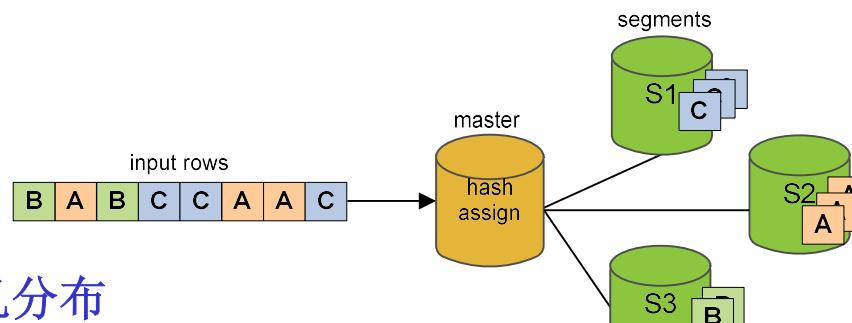
Order		
Order #	Order Date	Customer ID
43	Oct 20 2005	12
64	Oct 20 2005	11
45	Oct 20 2005	42
46	Oct 20 2005	64
77	Oct 20 2005	32
48	Oct 20 2005	12
50	Oct 20 2005	34
56	Oct 20 2005	21
63	Oct 20 2005	15
44	Oct 20 2005	10
53	Oct 20 2005	82
55	Oct 20 2005	55



数据分布方法

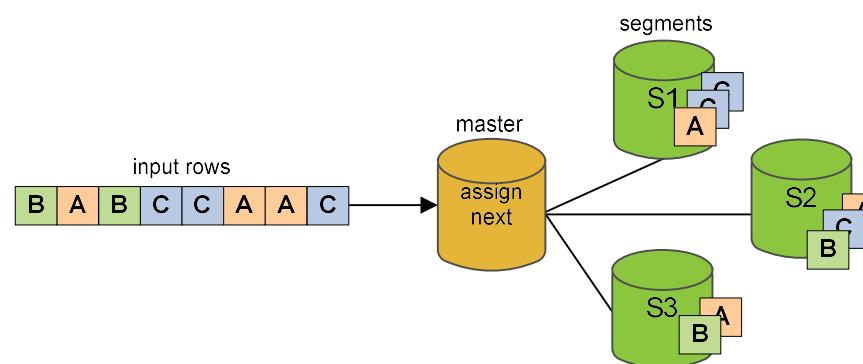
□ 哈希分布

- `CREATE TABLE ... DISTRIBUTED BY (column [,...])`
- 哈希值相同的记录在同一个**Segment**节点



□ 随机分布

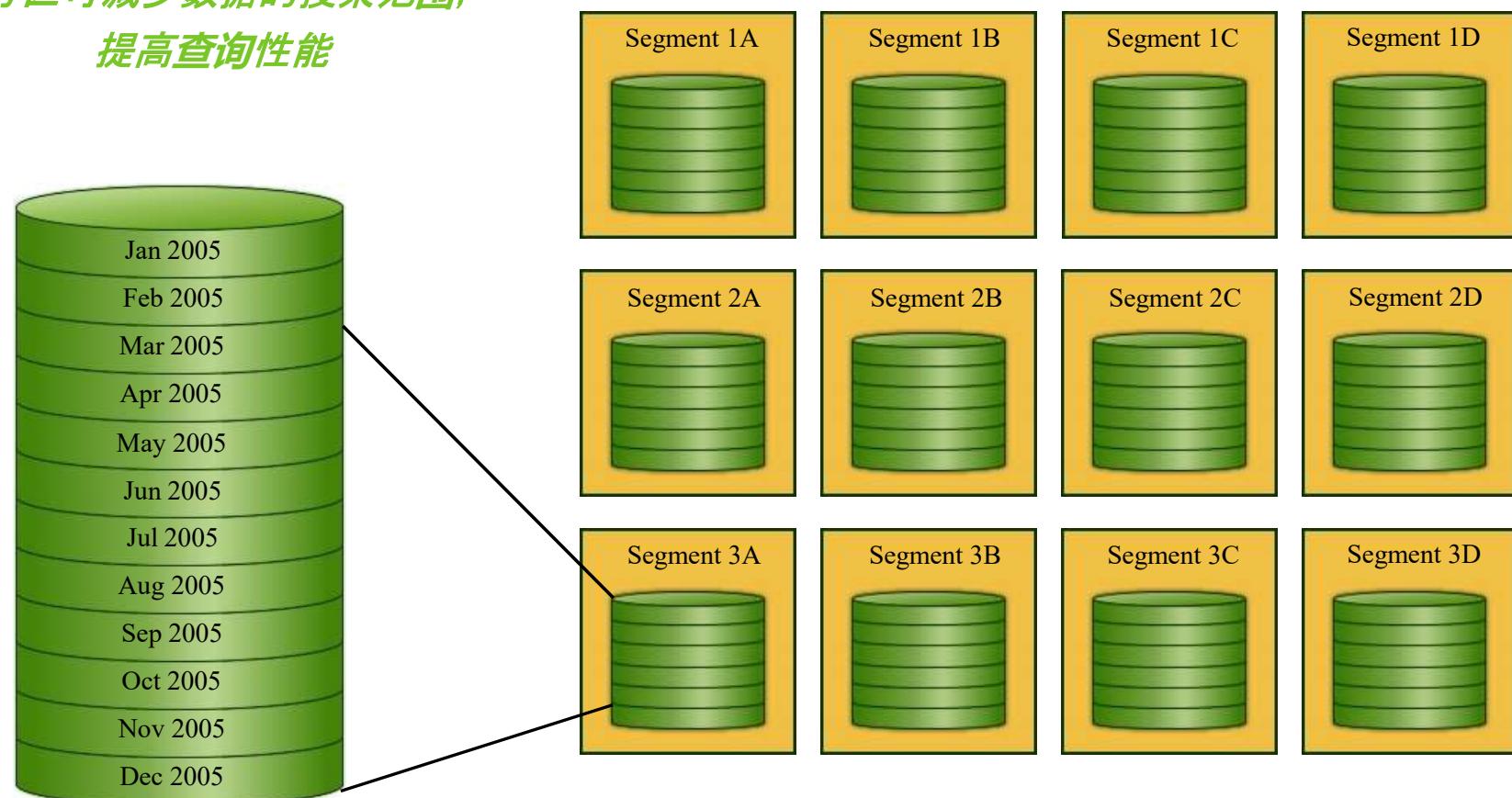
- `CREATE TABLE ... DISTRIBUTED RANDOMLY`
- **Rows with columns of the same value not necessarily on the same segment**



数据分布和分区

表分区可减少数据的搜索范围,
提高查询性能

每个分区表的数据自动分布到各个节点



索引、列数据库与数据压缩

□ 支持索引类型：

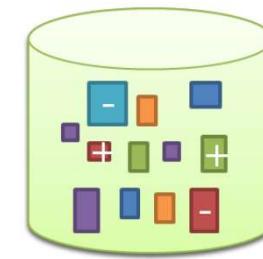
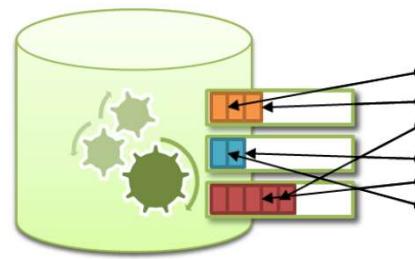
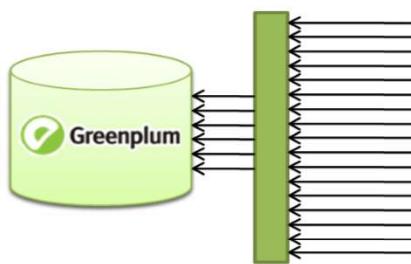
- **B-tree**
- **Bitmap**
- **R-tree**
- **Hash**
- **GIST (Generalized Search Tree)**

□ 支持按列存储数据库，及列数据库索引

□ 透明实时数据压缩类型：

- **gzip: 1 到 9 压缩水平**
- **QuickLZ: 1 到 3 压缩水平**

GP 负载管理技术



Connection 级

- 控制多少用户可以接入.
- 提供连接池和新建连接控制能力

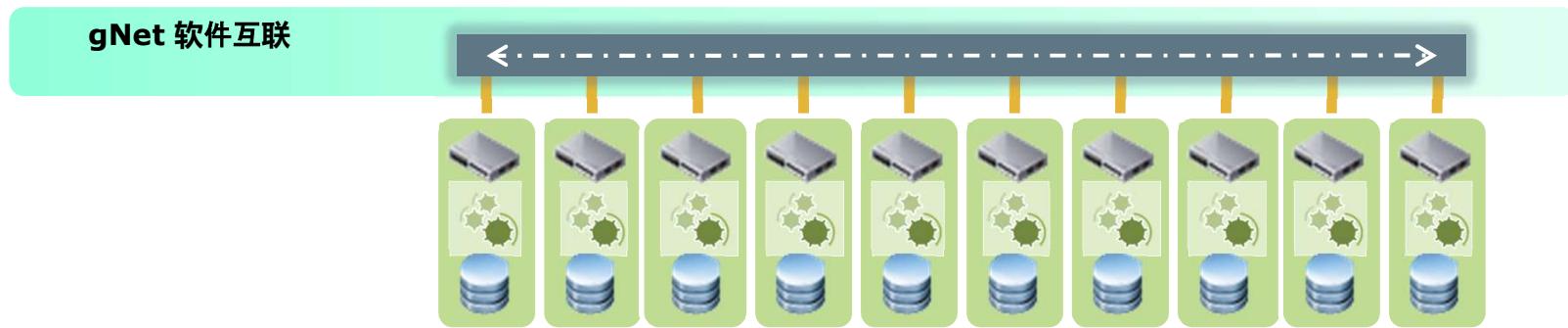
Session 级

- 每个用户可以设定 **resource queue** 来管理任务进入情况
- 可以控制多少查询并发及查询资源成本占比

Query (SQL) 级

- SQL查询语句可设置优先级，并且可在语句执行中实时调整
- 用于优待特定查询，从而缩短其运行时间

gNet 软件互联技术



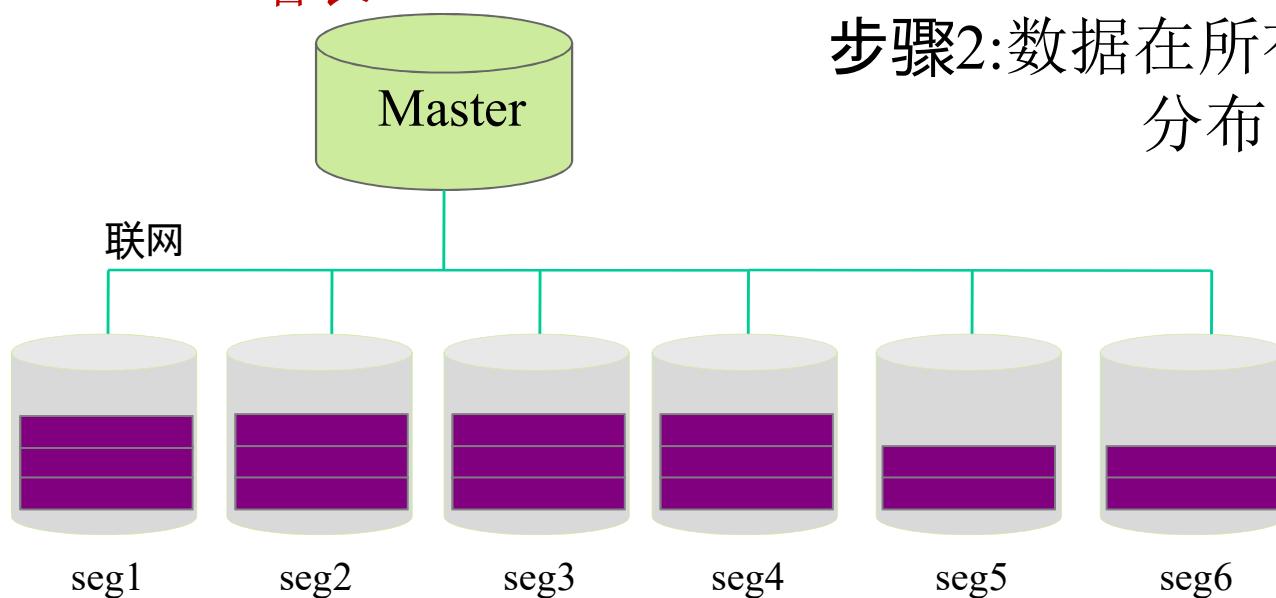
- 基于超级计算的“软件Switch”内部连接层
- 基于通用的gNet (GigE, 10GigE, IB) 网卡和交换机
 - 在节点间传递消息和数据
- 采用高扩展协议，支持扩展到10,000个以上节点

动态在线系统扩容

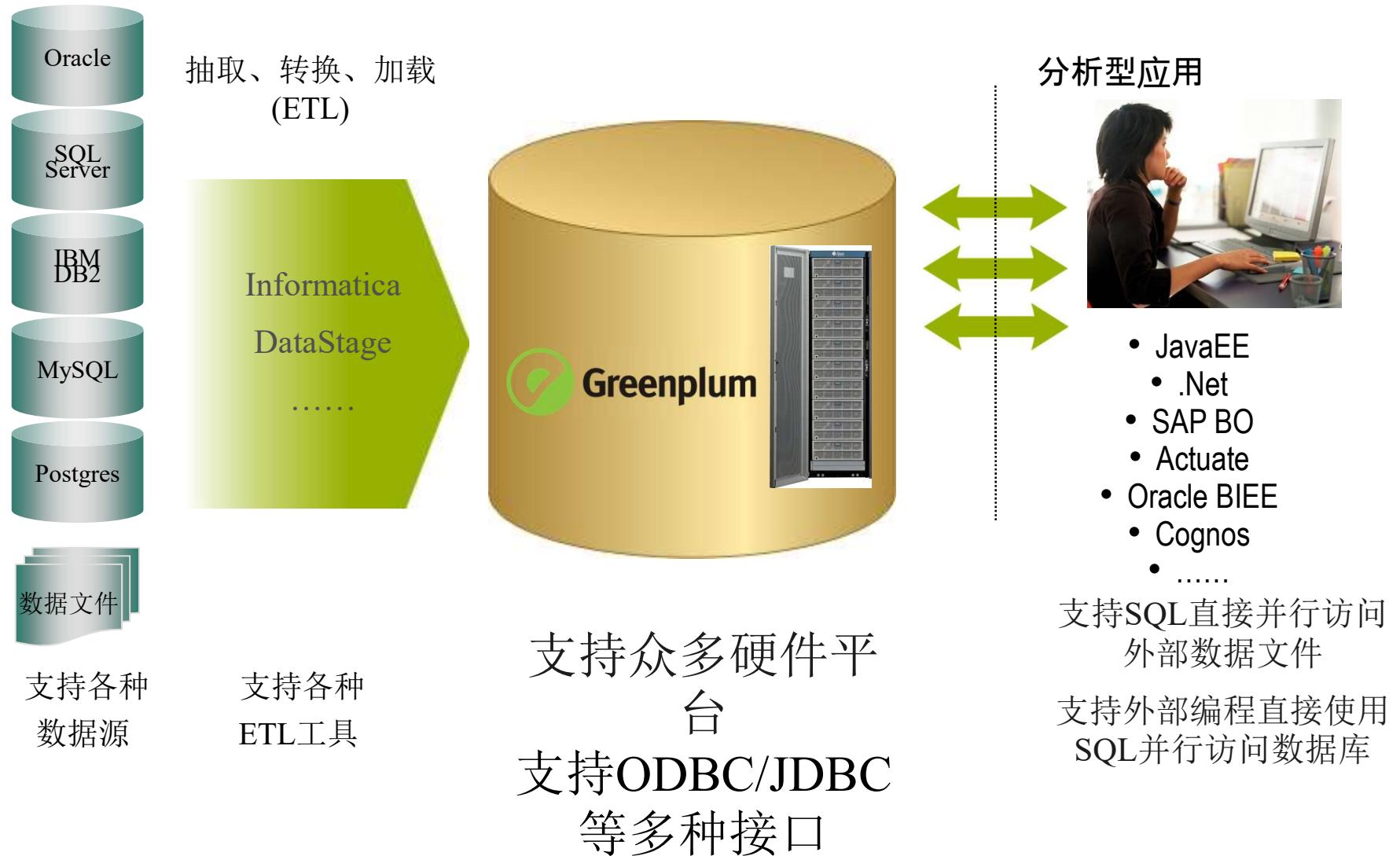
- 数据自动在所有节点上重新分布
- 容量和性能在扩展后线性增长

步骤1:新节点初始化加入MPP集群

步骤2:数据在所有节点上重新分布



分析型应用体系架构

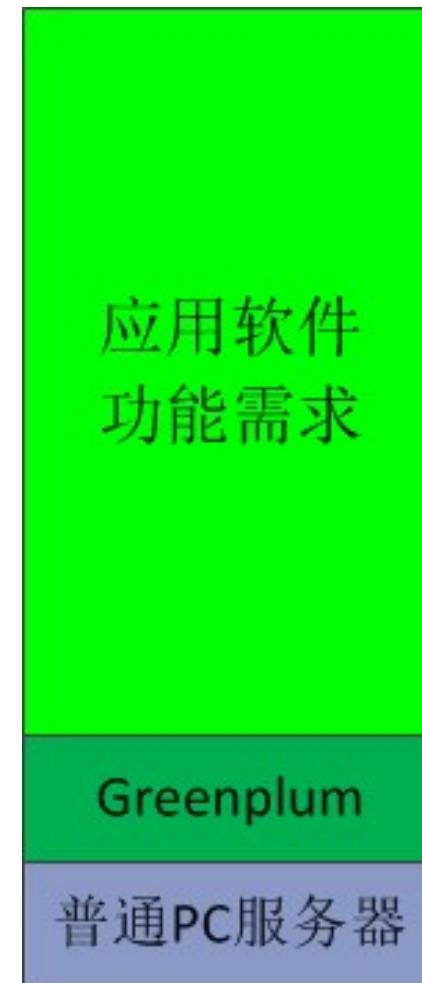


Greenplum方案的优势

现在的解决方案投资



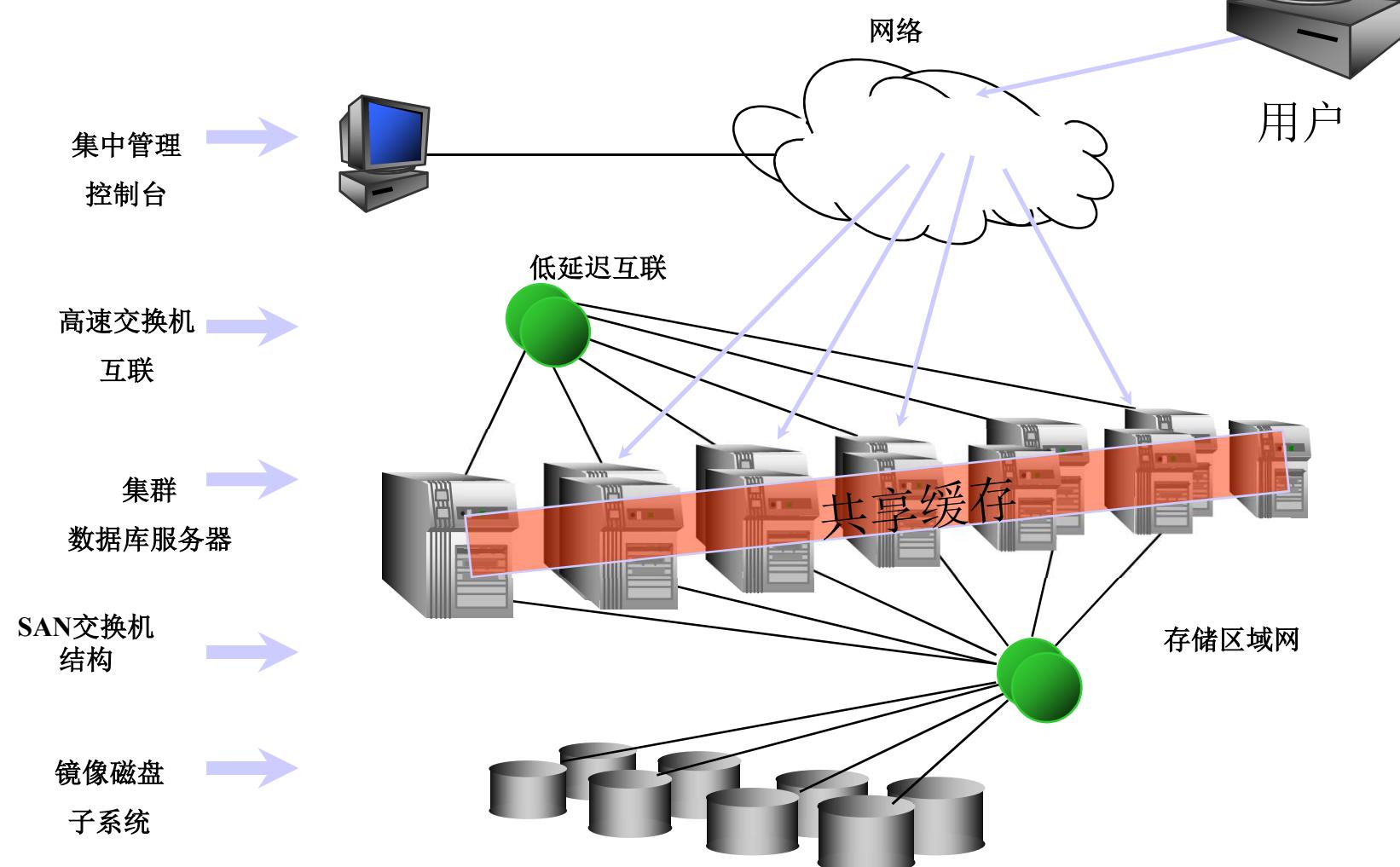
Greenplum方案投资



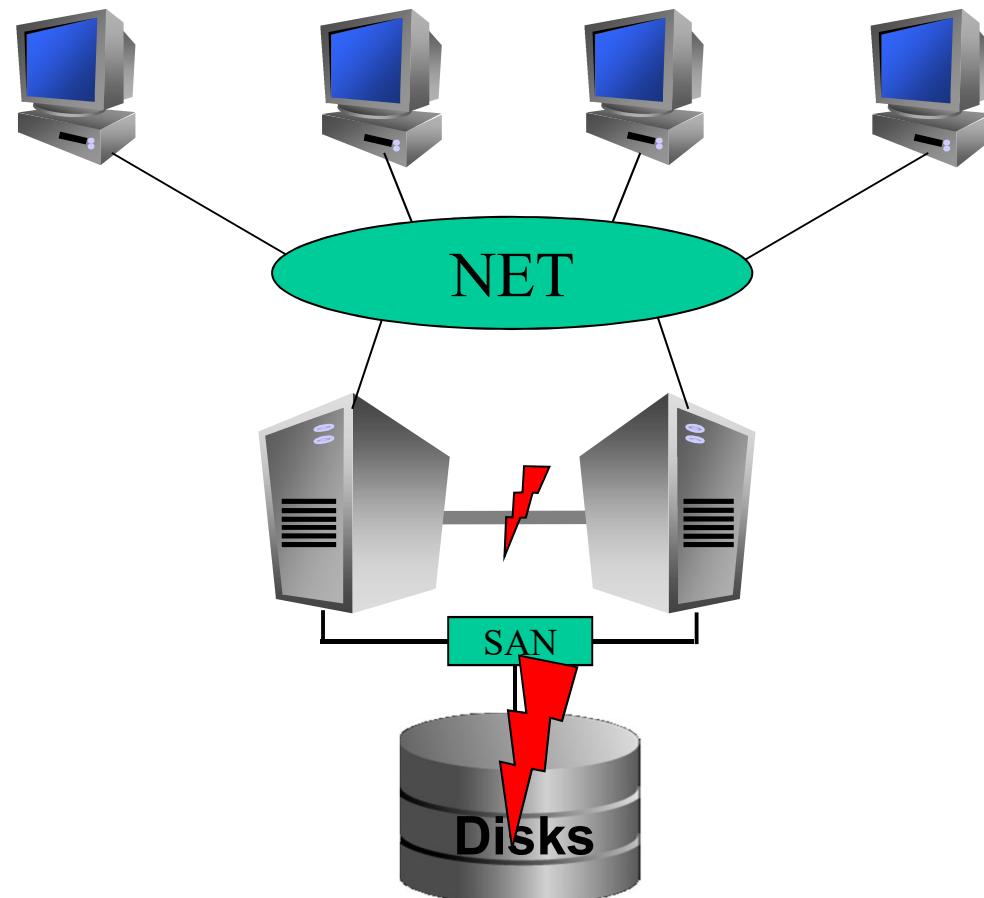
Greenplum Database: 技术构成



Oracle RAC 集群数据库

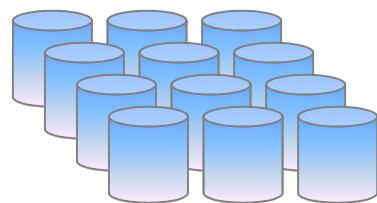
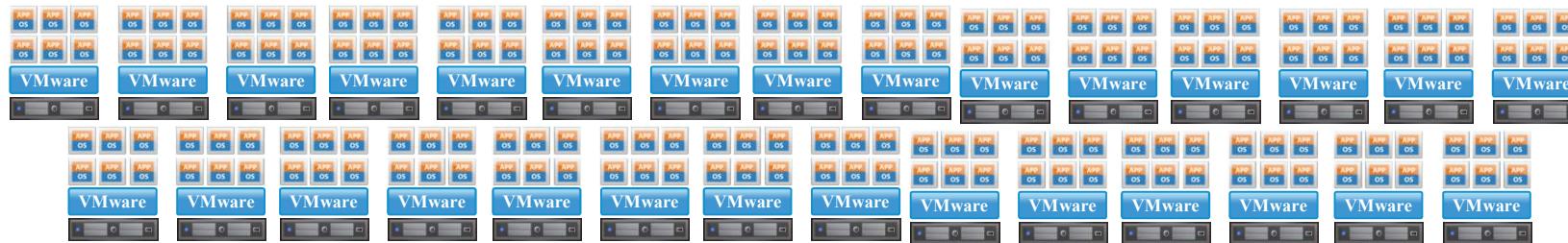


Oracle RAC 瓶颈

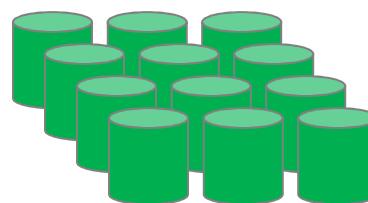




计算云

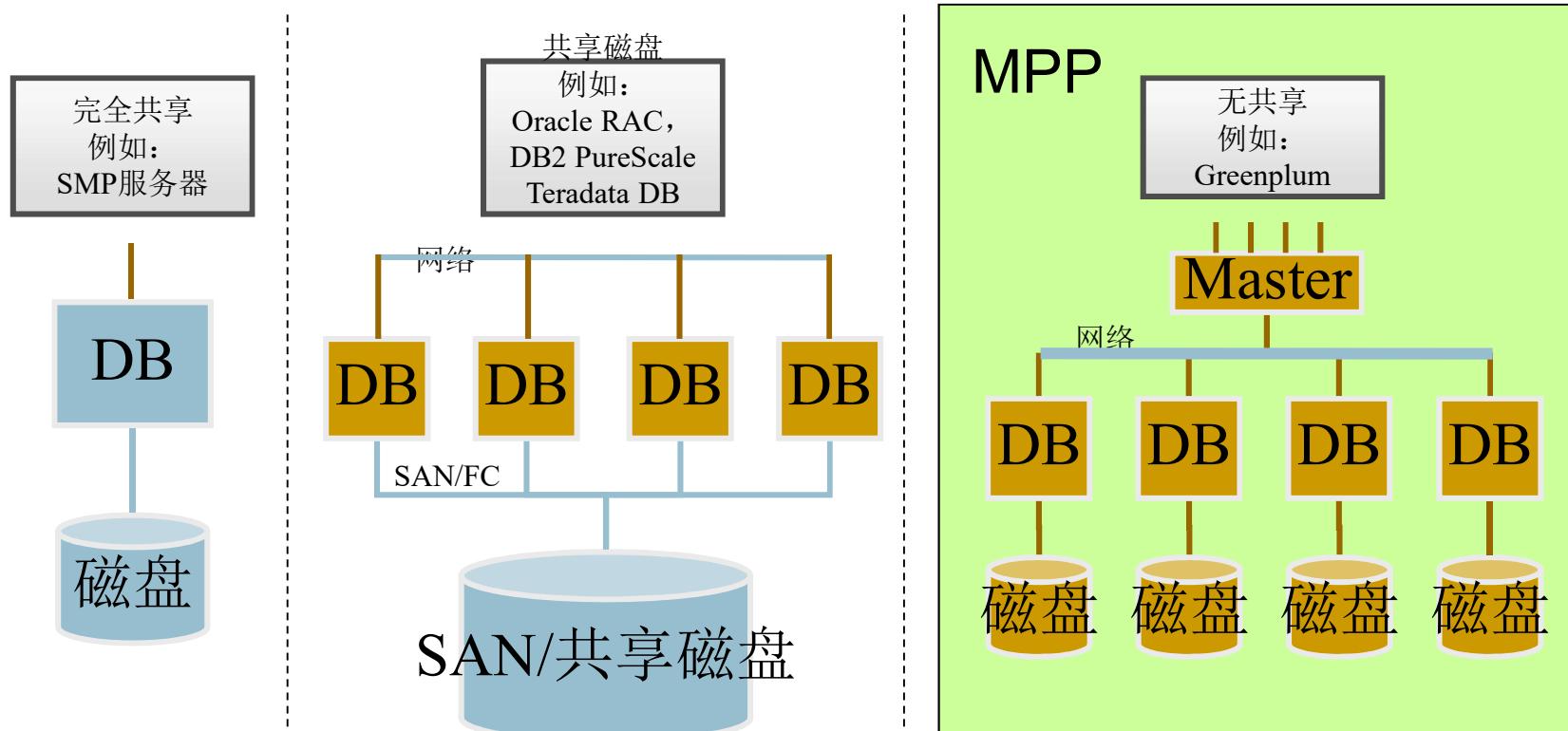


存储云（文件系统）



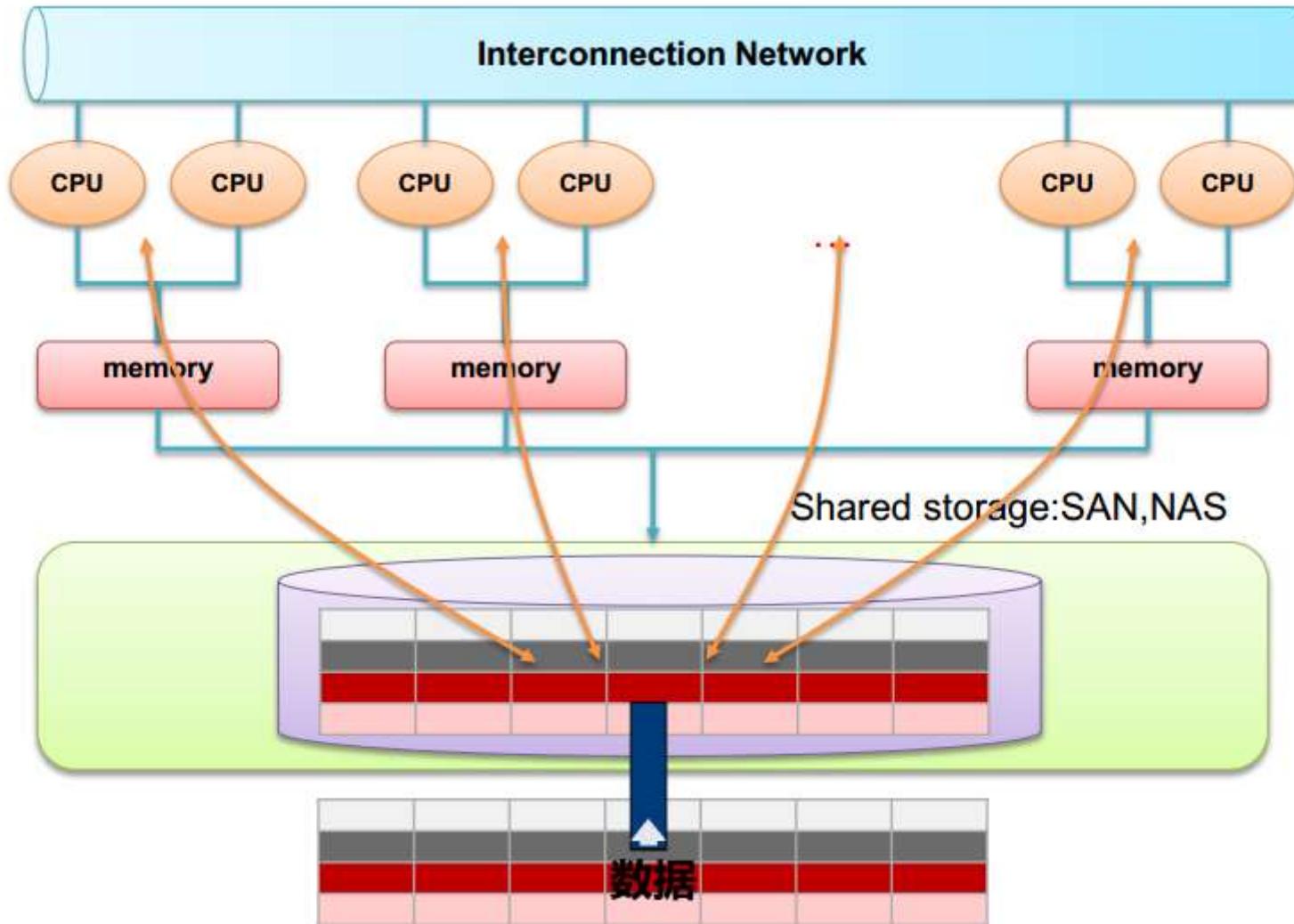
数据库云

无共享的完全大规模并行处理架构对比

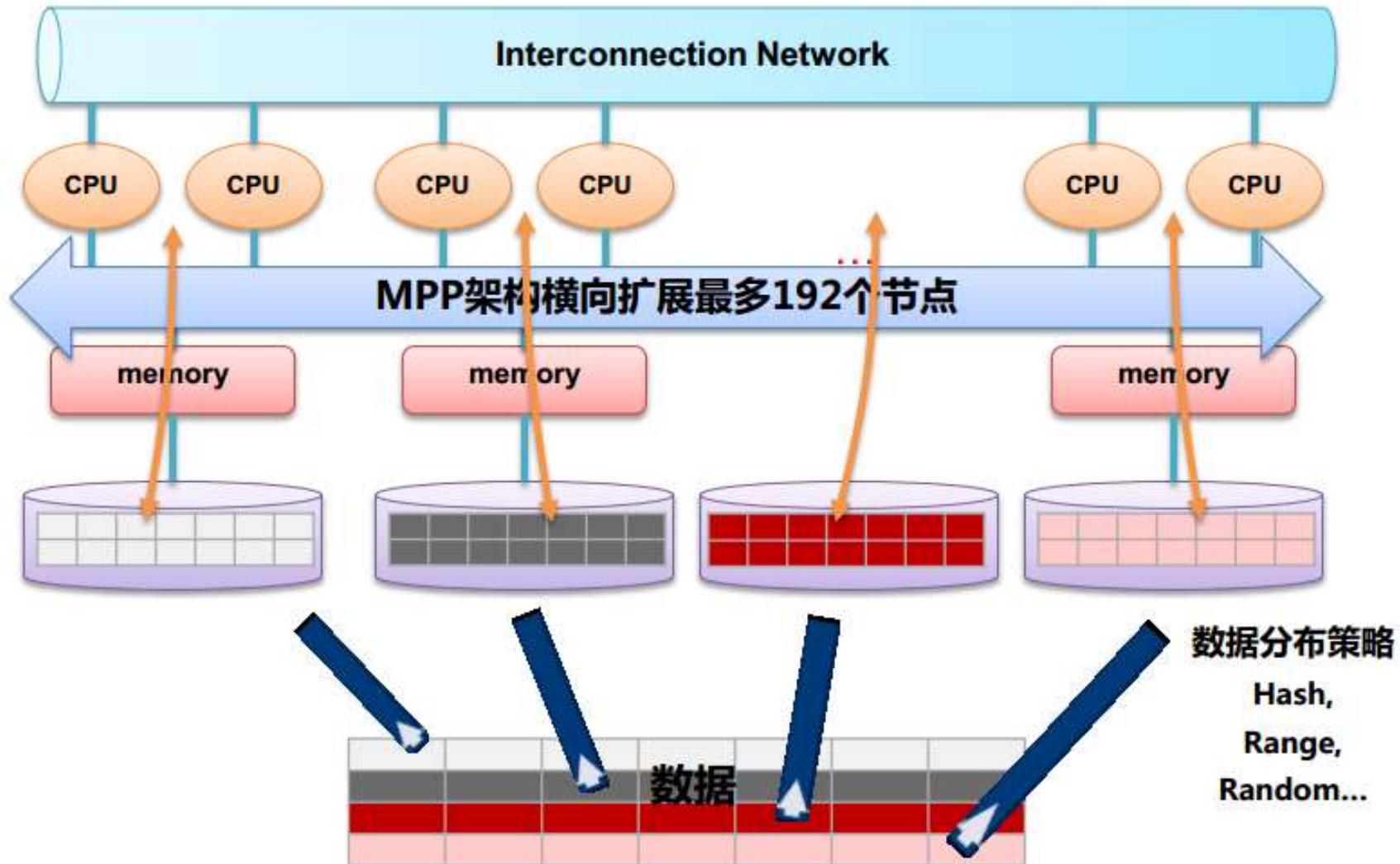


注： 蓝灰色表示共享资源

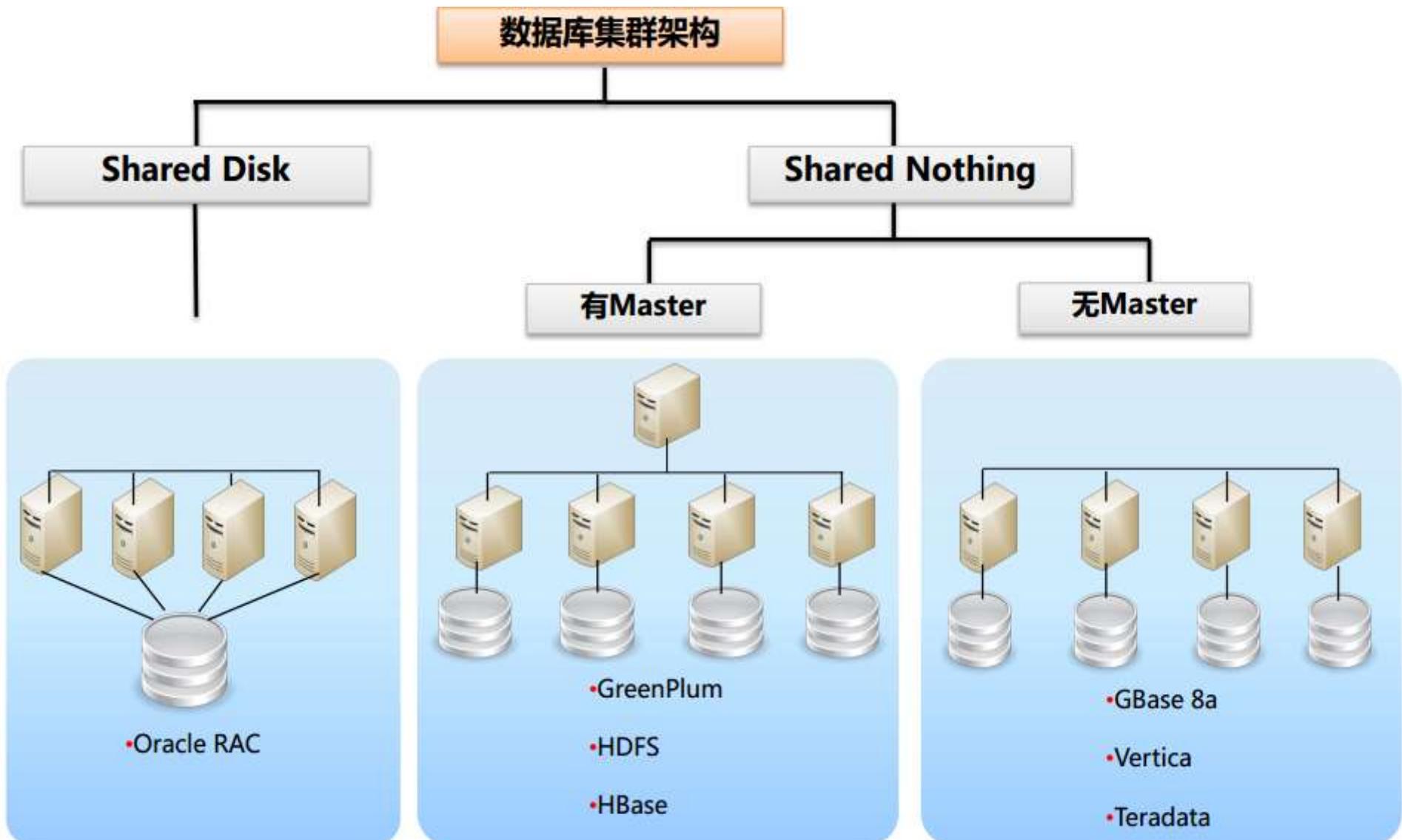
传统的Shared Disk架构



新型的Shared Nothing + MPP架构

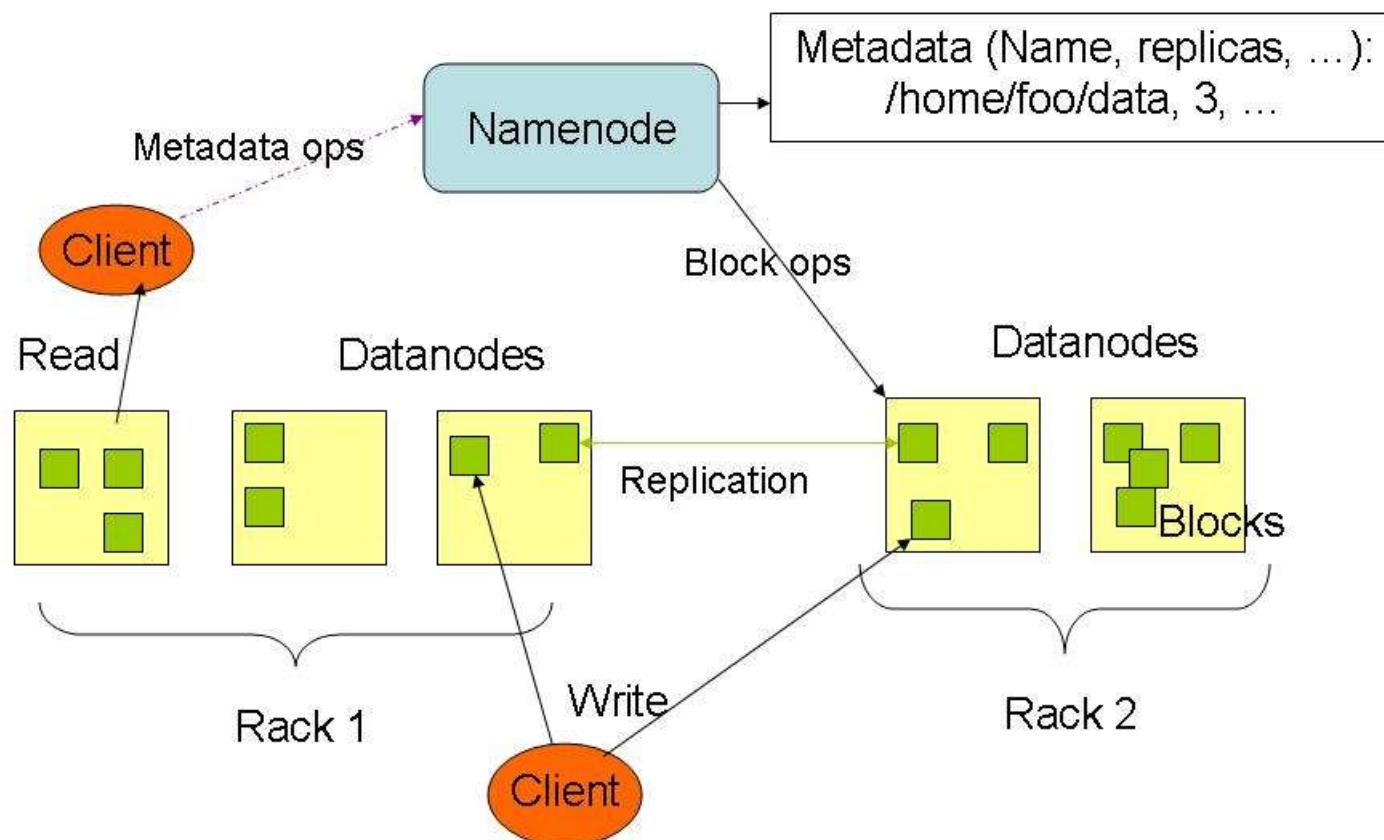


Shared Disk vs. Shared Nothing

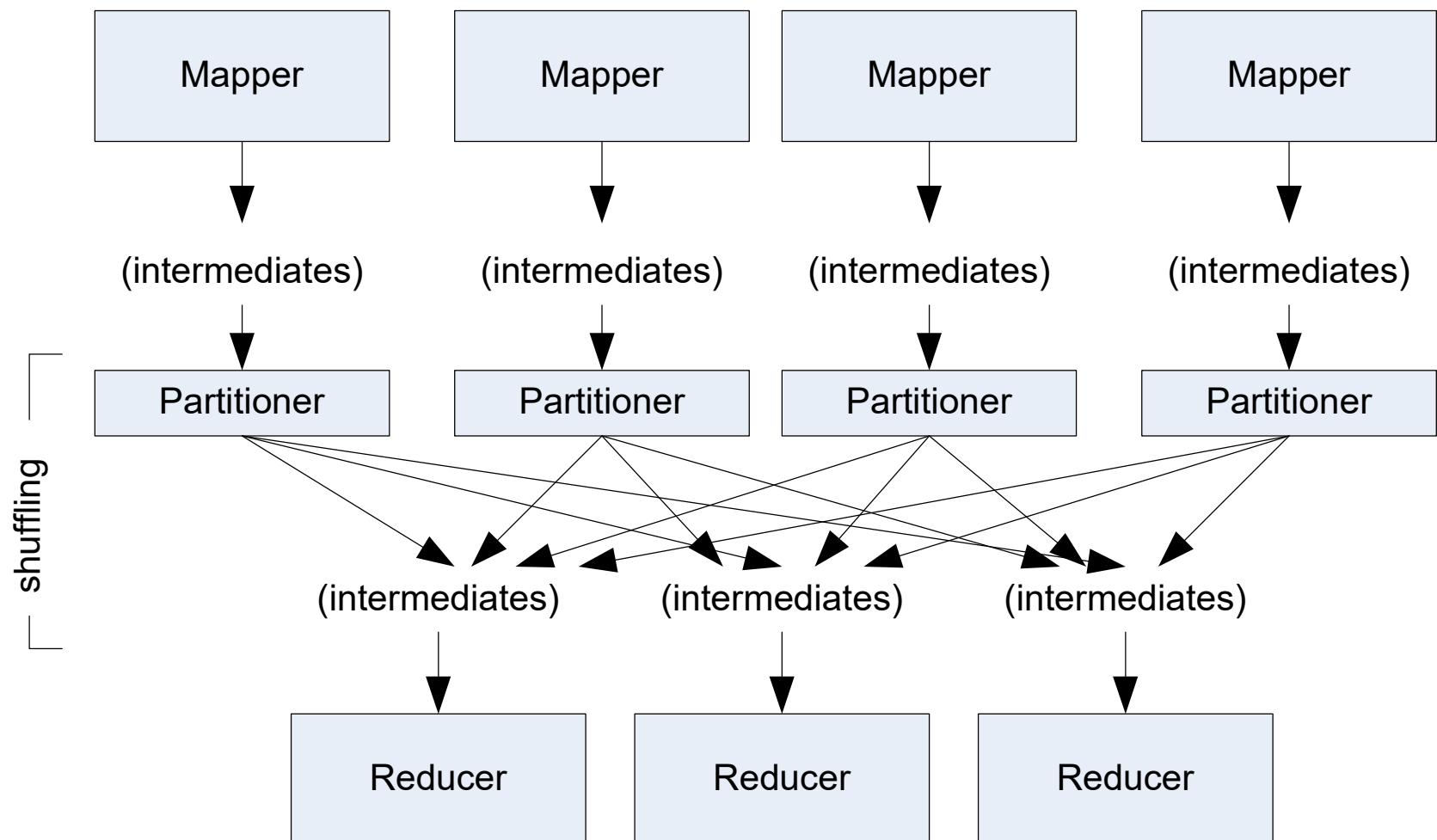


Hadoop HDFS 架构

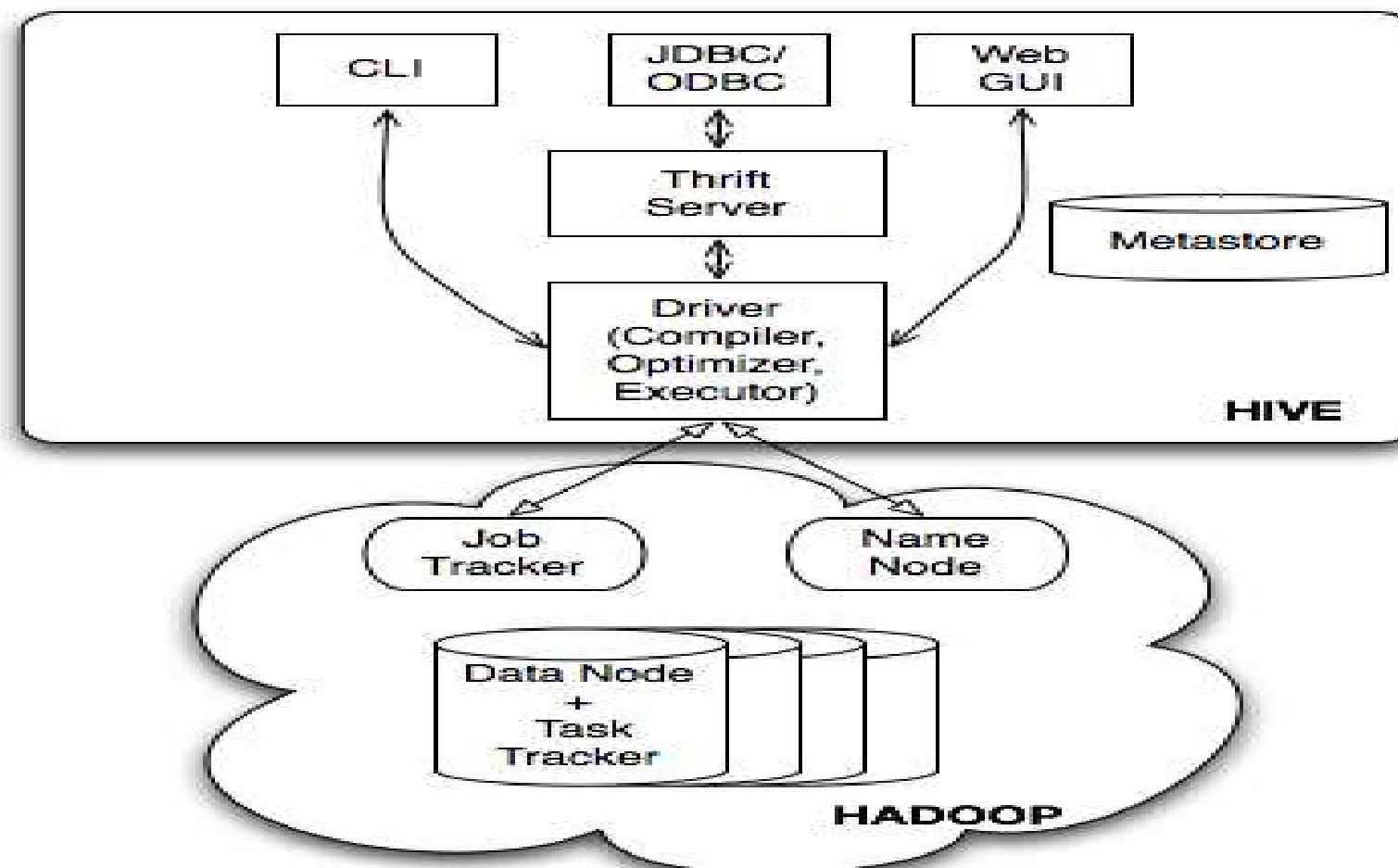
HDFS Architecture



Hadoop MapReduce



HIVE/Hadoop 架构



对比测试结果

- Hadoop+Pig+PigFly (ASC 15个节点)
 - 4m42. 207s
- Hadoop+Hive (ASC 15节点)
 - 3m30s
- ORACLE RAC (4 节点)
 - 41s
- GREENPLUM (2 节点)
 - 6012. 852 ms

数据库产品技术对比

	Greenplum	Teradata	Netezza	Oracle Exadata	DB2	Sybase IQ
无共享 MPP架构	✓	✓	✓			
支持开放硬件平台	✓				✓	✓
高级负载管理	✓	✓		✓	✓	
在线系统扩容	✓			✓	?	
按列存储	✓			✓		✓
按行存储	✓	✓	✓	✓	✓	✓
In-DB MapReduce	✓					
支持SQL2003及OLAP选项	✓	✓	✓	?	✓	?
性能线性扩展	✓	✓	✓		?	
加载能力线性扩展	✓					
Pipelined interconnect	✓		✓			
DAS容错	✓					
表分区	✓	✓	*	✓	✓	✓
索引	✓	✓		✓	✓	✓
最少的管理/调优	✓		✓			

- **Greenplum** 首先将并行处理技术运用到大规模的分析型数据平台
- 已经成为全球许多大型分析型系统的驱动力
- **Greenplum** 是目前唯一能提供单一的并行处理平台，并结合**SQL**数据处理及**MapReduce**搜索能力
- 帮助企业快速构建海量数据分析管理平台，提升企业深入挖掘数据能力及竞争优势

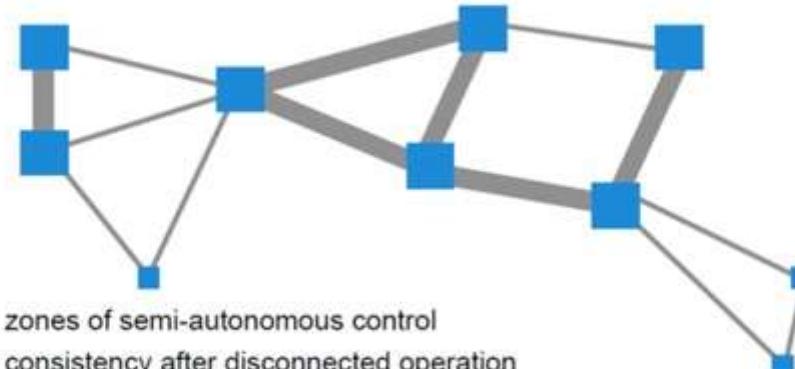


Google Spanner: 分布式数据库

- Google的全球级的分布式数据库 (**Globally-Distributed Database**)
- 扩展性级别高，可以扩展到数百万的机器、数百计的数据中心、上万亿的行
- 能通过同步复制和版本控制满足外部一致性，可用性很好
- 试图突破**CAP**理论，在三者之间实现平衡

Design Goals for Spanner

- Future scale: $\sim 10^6$ to 10^7 machines, $\sim 10^{13}$ directories, $\sim 10^{18}$ bytes of storage, spread at 100s to 1000s of locations around the world, $\sim 10^9$ client machines



- zones of semi-autonomous control
- consistency after disconnected operation
- users specify high-level desires:
 - "99%ile latency for accessing this data should be <50ms"
 - "Store this data on at least 2 disks in EU, 2 in U.S. & 1 in Asia"

Google Spanner: 分布式数据库

- 非开源
- Google开发的可容错可扩展的RDBMS: F1
 - 非NoSQL体系
- 分布式文件系统: GFS
- 支持Schema、事务等

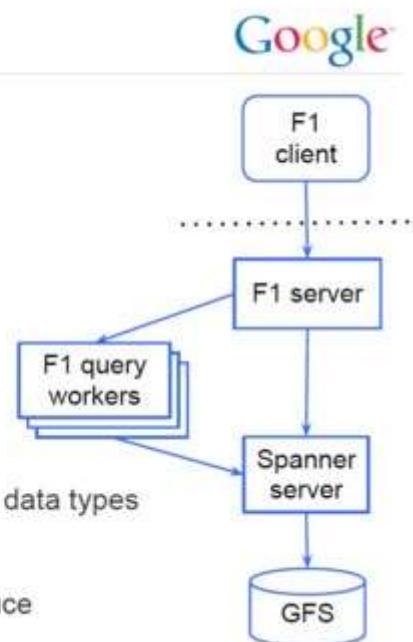
F1

Architecture

- Sharded Spanner servers
 - data on GFS and in memory
- Stateless F1 server
- Pool of workers for query execution

Features

- Relational schema
 - Extensions for hierarchy and rich data types
 - Non-blocking schema changes
- Consistent indexes
- Parallel reads with SQL or Map-Reduce



NoSQL体系

- HBase
- BigTable

```
CREATE TABLE Users {  
    uid INT64 NOT NULL, email STRING  
} PRIMARY KEY (uid), DIRECTORY;
```

```
CREATE TABLE Albums {  
    uid INT64 NOT NULL, aid INT64 NOT NULL,  
    name STRING  
} PRIMARY KEY (uid, aid),  
INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

Users(1)	Directory 3665
Albums(1,1)	
Albums(1,2)	
Users(2)	
Albums(2,1)	Directory 453
Albums(2,2)	
Albums(2,3)	

Figure 4: Example Spanner schema for photo metadata, and the interleaving implied by INTERLEAVE IN.

二、列存储数据库

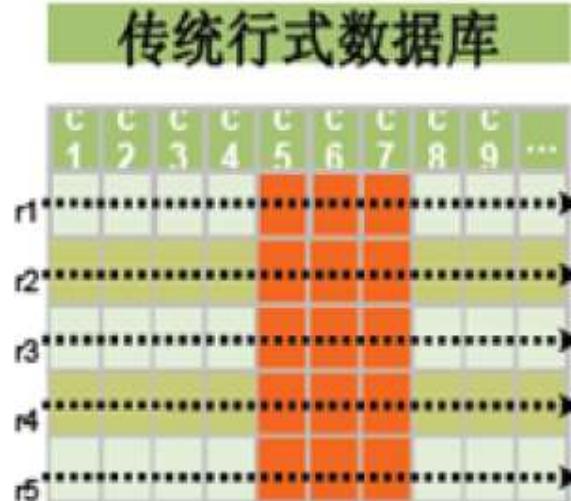
- 1999年，列存储数据库致力于改变用于关系型数据库或者“基于行”的数据库的基础存储模型。
- 在基于行的数据库中，数据存储在属性的连续行中，列通过表的元数据相关。
- 列存储数据库把这个模型转了90度，把列的属性存储在一起，而只通过元数据关联行。这允许了不少的优化，包括各种形式的压缩和非常快的聚合。
- 当前的列存储包括 **Vertica, Paraccel, Infobright, LucidDB** 和 **MonetDB**.
- 一些其他类型的数据库，包括 **Aster Data** 和 **Greenplum**，已经将列存储作为一个选项加入

二、列存储数据库

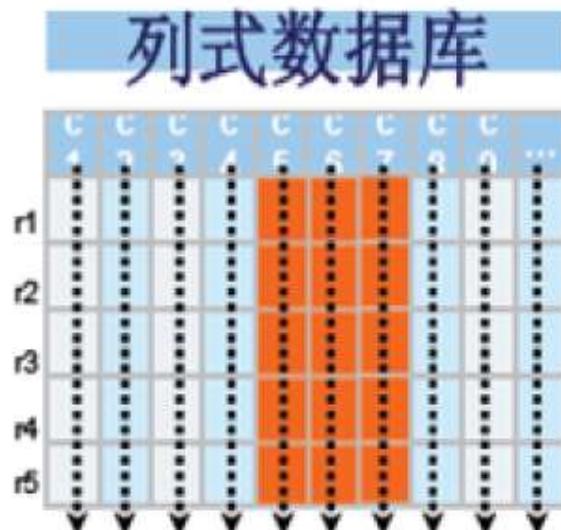
- 列存储更适合**OLAP**类型的分析应用
- 列存储对于数据有很好的效率，可以减少数字和类别列表的数据。
- 主要缺点：更新或导入数据缓慢，单行更新不可能



二、列存储数据库



- 数据是按行存储的
- 没有索引的查询使用大量I/O
- 建立索引和物化视图需要花费大量时间和资源
- 面对查询的需求，数据库必须被大量膨胀才能满足性能要求

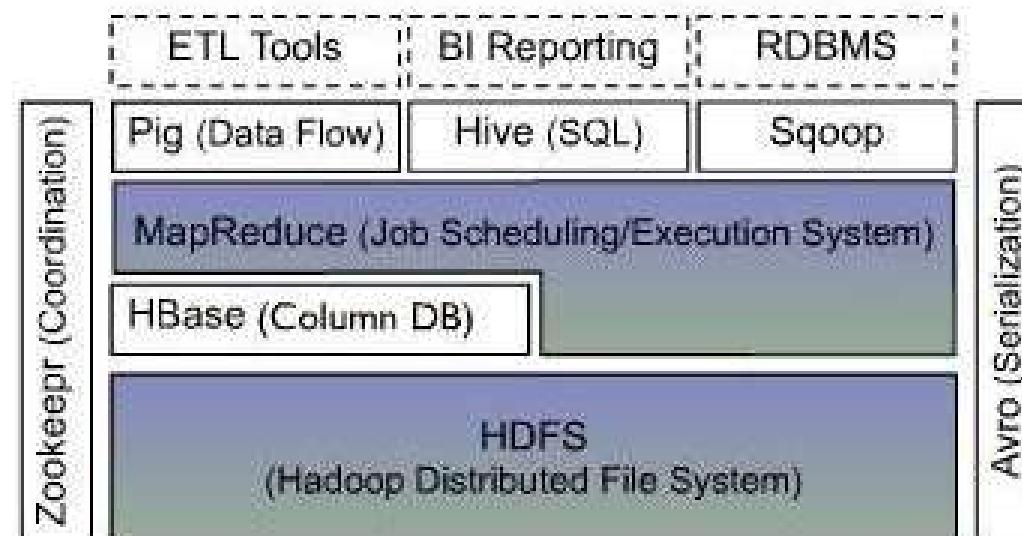


- 数据按列存储 - 每一列单独存放
- 数据即是索引
- 只访问查询涉及的列 - 大量降低系统I/O
- 每一列由一个线索来处理 - 查询的并发处理
- 数据类型一致，数据特征相似 - 高效压缩

二、列存储数据库

□ HBase

The Hadoop Ecosystem



Hbase

- Hbase是bigtable的开源版本
 - 建立在hdfs之上，提供高可靠性、高性能、列存储、可伸缩、实时读写的分布式数据库系统
 - -海量数据（TB 以上）
 - -很高的随机写能力
 - -在海量数据中实现高效的读取
 - -很好的伸缩能力
 - -强一致性
 - -能够同时处理结构化和非结构化的数据
 - -动态列
 - -不需要复杂的查询需求: SQL、事务、Join 、多维索引等

Hbase

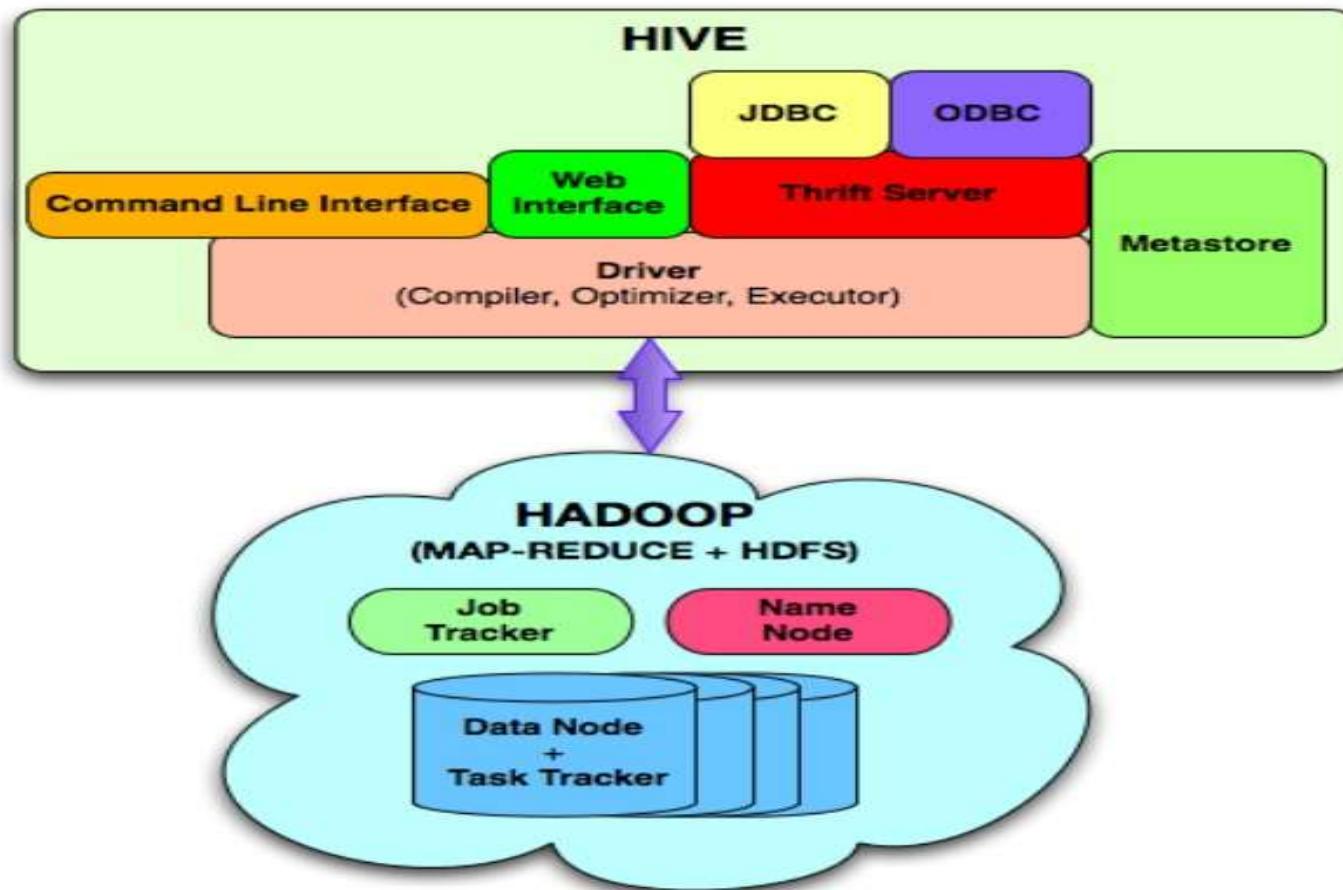
- 介于**nosql**和**RDBMS**之间，仅能通过主键(**row key**)和主键的**range**来检索数据，仅支持单行事务(可通过**hive**支持来实现多表**join**等复杂操作)。

- 与**hadoop**一样，**Hbase**目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

HIVE

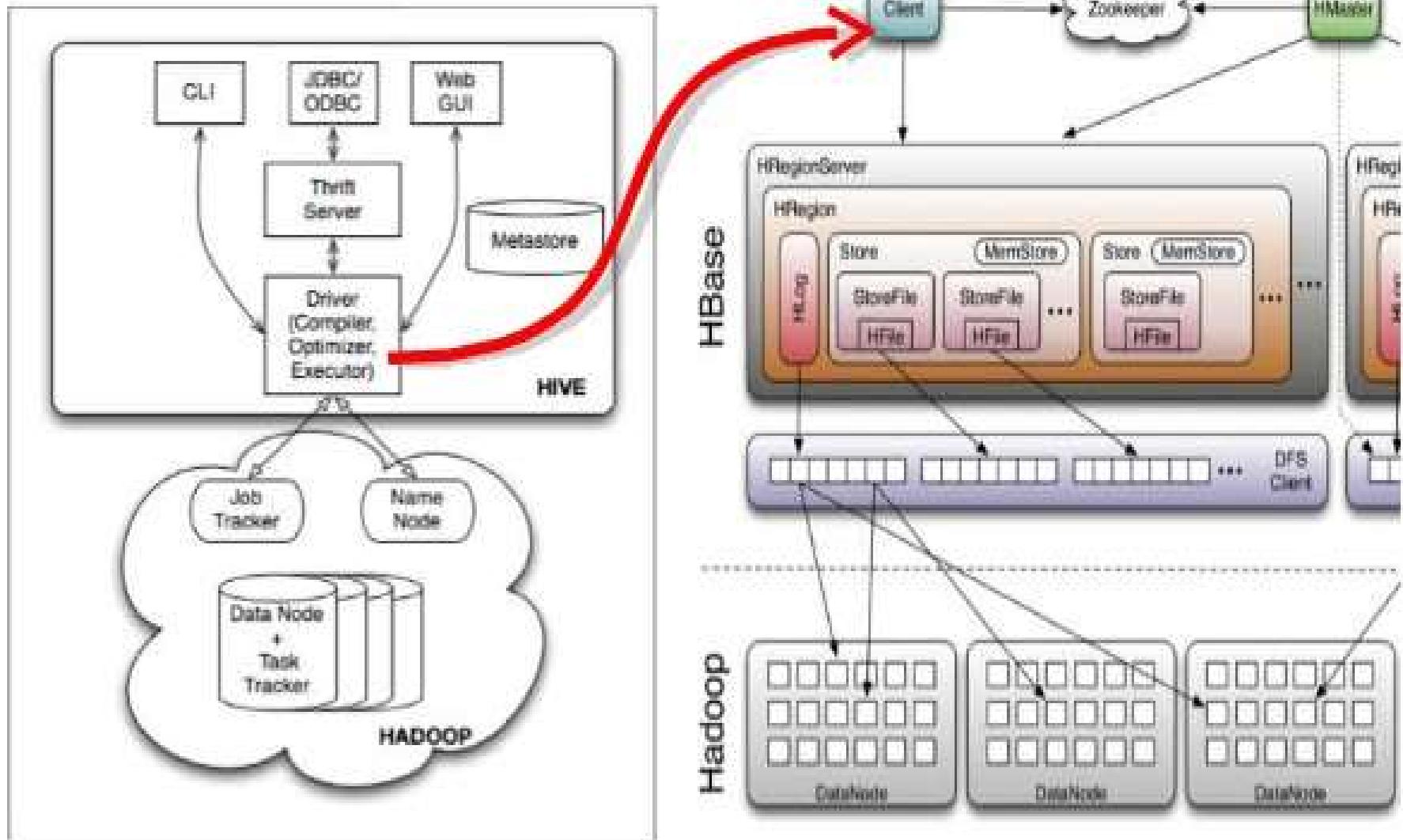
- **hive**是一个基于**hadoop**的数据仓库
 - 使用**hadoop-hdfs**作为数据存储层，将结构化的数据文件映射为一张数据库表
 - 提供类似**SQL**的语言（**HQL**），将**sql**语句转换为**MapReduce**任务完成数据计算
 - 通过**HQL**语言提供表格查询特性和分布式存储计算特性，不必开发专门的**MapReduce**应用，十分适合数据仓库的统计分析

HIVE架构



- 1、操作界面： CLI, Web, Thrift
- 2、driver： hive系统将用户操作转化为mapreduce计算的模块（重点）
- 3、hadoop： hdfs+mapreduce
- 4、metastore： 存储元数据(存储在传统RDBMS： mysql中)

HIVE结合HBase



三、NoSQL

□ Not Only SQL

- NO+SQL：暗示其与SQL之间的对立性
- No RDBMS? No Relational?
- 泛指不遵循经典RDBMS原理，且常与Web规模的大型数据集有关的一族产品，以及一系列不同的、有时相互关联的、有关数据库、数据存储及处理的概念
- 2009年后兴起

- 处理超大数据量，TB or PB级别（Search）
- 高并发（万/s），不注重事务（CAP原则）
- 易部署、易扩展、易开发（透明）
- 便宜

三、NoSQL

□ Why NoSQL?

➤ Web 级别大规模数据处理对RDBMS 的挑战

- 松散结构的海量稀疏数据处理
- SNS中大部分需求并不要求ACID，如Like/Unlike投票

➤ High performance -高并发读写的需求

- web2.0网站数据库并发负载非常高
 - 要根据用户个性化信息来实时生成动态页面和提供动态信息，无法使用动态页面静态化技术
 - 每秒上万次读写请求
 - 关系数据库无法应付上万次SQL写数据请求（硬盘IO就已经无法承受）
- SNS网站每一个点击都是多条查询，对数据库写的并发要求非常高

三、NoSQL

➤ **Huge Storage** -对海量数据的高效率存储和访问的需求

- **SNS**网站每天产生海量的用户动态，**SQL**查询效率无法承受

➤ **High Scalability & High Availability**-对数据库的高可扩展性和高可用性的需求

- 基于**web**的架构中，数据库最难横向扩展的
- 希望：像**web server**和**app server**那样
 - 通过添加更多硬件和服务节点来扩展性能和负载能力

□ 关系型的数据模型基本上是分析数据间的结构和关系。其设计理念是：“” **What answers do I have?**”

□ **NoSQL** 数据模型基本上是从应用对数据的存取方式入手，如：我需要支持某种数据查询。其设计理念是” **What questions do I have?**”

SQL Characteristics

- Physical Layer Abstraction
- Data Manipulation Language
- Data Definition Language
- Transactions

Physical Layer Abstraction

- Applications specify what, not how
- Query optimization engine
- Physical layer can change without modifying applications
 - Create indexes to support queries
 - In Memory databases

Data Manipulation Language (DML)

❑ Select, Insert, Update, Delete...

➤ **Select T1.Column1, T2.Column2 ...
From Table1, Table2 ...
Where T1.Column1 = T2.Column1 ...**

❑ Data Aggregation

❑ Compound statements

❑ Functions and Procedures

❑ Explicit transaction control

Data Definition Language

- Schema defined at the start
- Create Table (Column1 Datatype1, Column2 Datatype 2, ...)
- Constraints to define and enforce relationships
 - Primary Key
 - Foreign Key
- Triggers to respond to Insert, Update , & Delete
- Stored Modules
- Alter ...
- Drop ...
- Security and Access Control

Transactions – ACID Properties

- **Atomic** – All of the work in a transaction completes (commit) or none of it completes
- **Consistent** – A transaction transforms the database from one consistent state to another consistent state.
- **Isolated** – The results of any changes made during a transaction are not visible until the transaction has committed.
- **Durable** – The results of a committed transaction survive failures

ACID不再适用

□ 数据库事务一致性需求

- 很多**web**系统不要求严格的数据 读一致性要求很低，有些场合对写一致性要求也不高
- 数据库事务管理成了数据库高负载下的负担

□ 数据库的写实时性和读实时性需求

- 关系数据库：插入一条数据之后可以立刻查询
- 但**web**应用不要求这么高的实时性
 - 用户发一条消息之后，过几秒乃至十几秒之后，订阅者才看到这条动态，是完全可以接受的

ACID不再适用

- 对复杂的**SQL**查询，特别是多表关联查询的需求
 - 任何大数据量的**web**系统，都注意尽量避免多个大表的关联查询，以及复杂**SQL**报表查询
 - 更多的只是单表的主键查询，以及单表的简单条件分页查询
 - **SQL**的功能被极大弱化

NoSQL Definition

www.nosql-database.org: (122种NoSQL数据库)

Redis, Tokyo Cabinet, Cassandra, Voldemort, MongoDB,
Dynomite, HBase, CouchDB, Hypertable, Riak, Tin, Flare,
Lightcloud, KiokuDB, Scalaris, Kai, ThruDB,

**Next Generation Databases mostly addressing some
of the points:**

**being non-relational, distributed, open-source
and horizontal scalable**

**The original intention: modern web-scale
databases**

Often more characteristics apply as:

**schema-free, easy replication support, simple
API, eventually consistent / BASE (not ACID),
a huge data amount...**

NoSQL + RDBMS

□ Oracle NoSQL 产品

- 基于 BerkeleyDB 的 NoSQL 数据库
- 适用 Web 服务和云环境、可横向扩展的键值数据库
- 与 Oracle 数据库和 Hadoop 集成

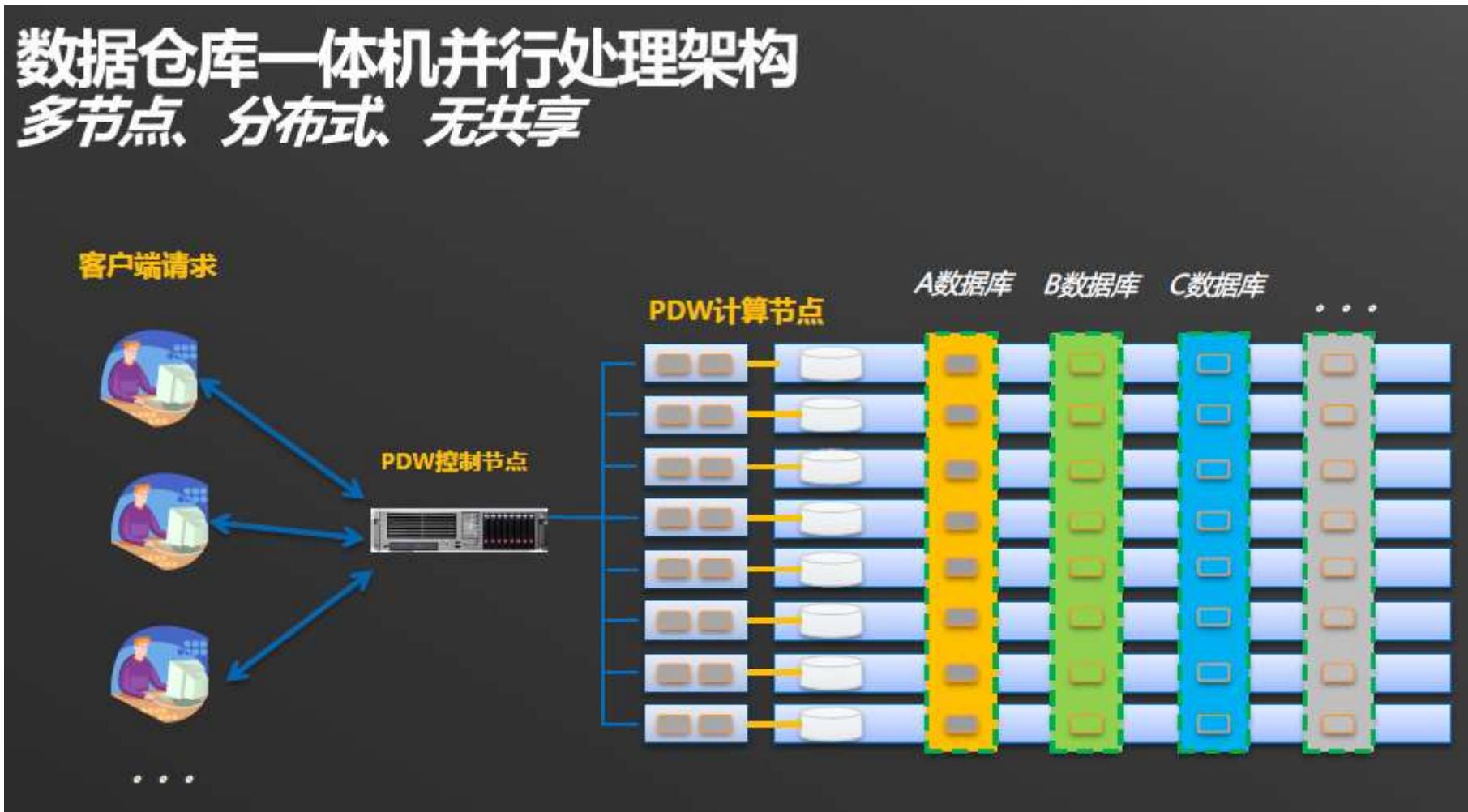
□ IBM DB2

- DB2 NoSQL JSON
- 收购 NoSQL 数据库公司 Cloudant (基于 Apache CouchDB 提供分布式云文件存储服务)
- 2014.5 Mellanox 与 IBM 携手打造高速 NoSQL 数据库

□ Microsoft

- PDW 数据仓库一体机

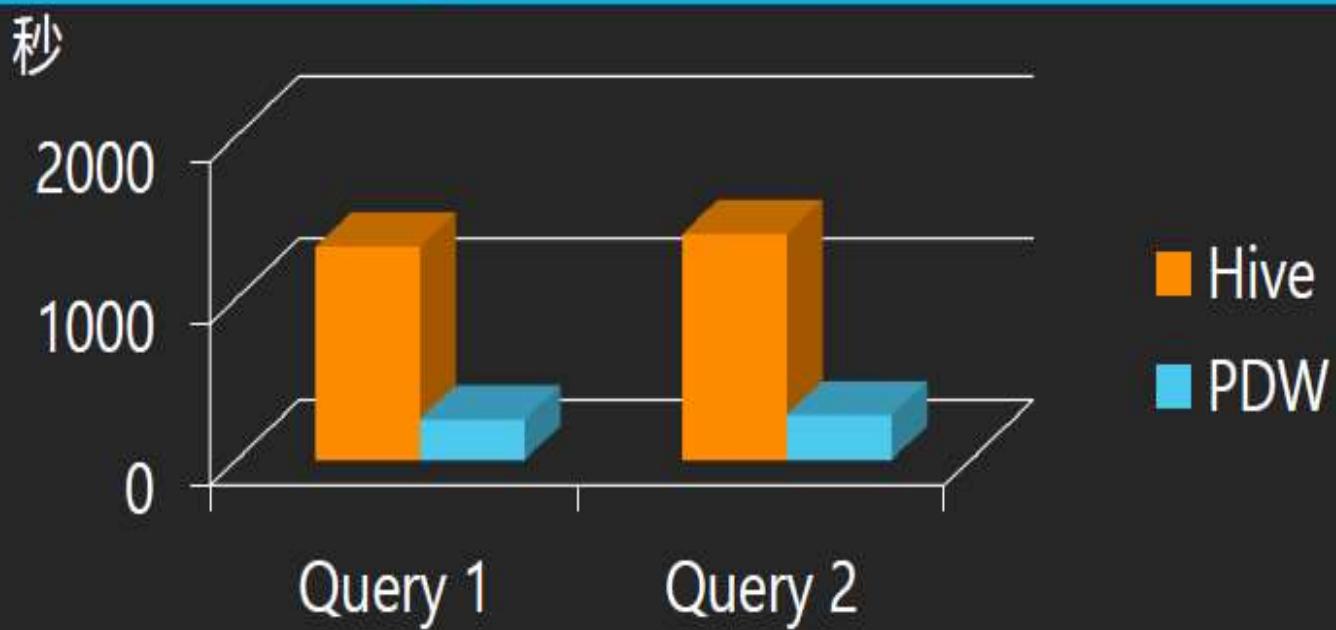
Microsoft SQL Server PDW



PDW数据仓库一体机 VS. HADOOP数据仓库 (HIVE)

Query 1: `SELECT count(*) FROM lineitem`

Query 2: `SELECT max(l_quantity)`
 `FROM lineitem`
 `WHERE l_orderkey > 1000 and l_orderkey < 100000`
 `GROUP BY l_linenumber`



NoSQL Distinguishing Characteristics

- ❑ Large data volumes
- ❑ Scalable replication and distribution
 - Potentially thousands of machines
 - Potentially distributed around the world
- ❑ Queries need to return answers quickly
- ❑ Mostly query, few updates
- ❑ Asynchronous Inserts & Updates
- ❑ Schema-less
- ❑ ACID transaction properties are not needed – BASE
- ❑ CAP Theorem
- ❑ Open source development

Summary:

Alternative to traditional relational DBMS

- + Flexible schema
- + Quicker/cheaper to set up
- + Massive scalability
- + Relaxed consistency → higher performance & availability

- No declarative query language → more programming
- Relaxed consistency → fewer guarantees

Examples

Example #1: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Task: Find all records for...

- Given UserID
- Given URL
- Given timestamp
- Certain construct appearing in additional-info

Example #2: Social-network graph

Each record: UserID₁, UserID₂

Separate records: UserID, name, age, gender, ...

Task: Find all friends of friends of friends of ... friends of given user

Examples

Example #3: Wikipedia pages

Large collection of documents

Combination of structured and unstructured data

Task: Retrieve introductory paragraph of all pages
about U.S. presidents before 1900

CAP理论

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

➤ (Seth Gilbert*) Nancy Lynch*



Abstract

□ 一
能
➤ (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.)

CAP理论



三、NoSQL

□ 1. 一致性(Consistency):

➤任何一个读操作总是能读取到之前完成的写操作结果，也就是在分布式环境中，多点的数据是一致的；

□ 2. 可用性(Availability):

➤每一个操作总是能够在确定的时间内返回，也就是系统随时都是可用的。(指的是快速获取数据)

□ 3. 分区容错性(Partition Tolerance):

➤在出现网络分区(比如断网)的情况下，分离的系统也能正常运行。

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**

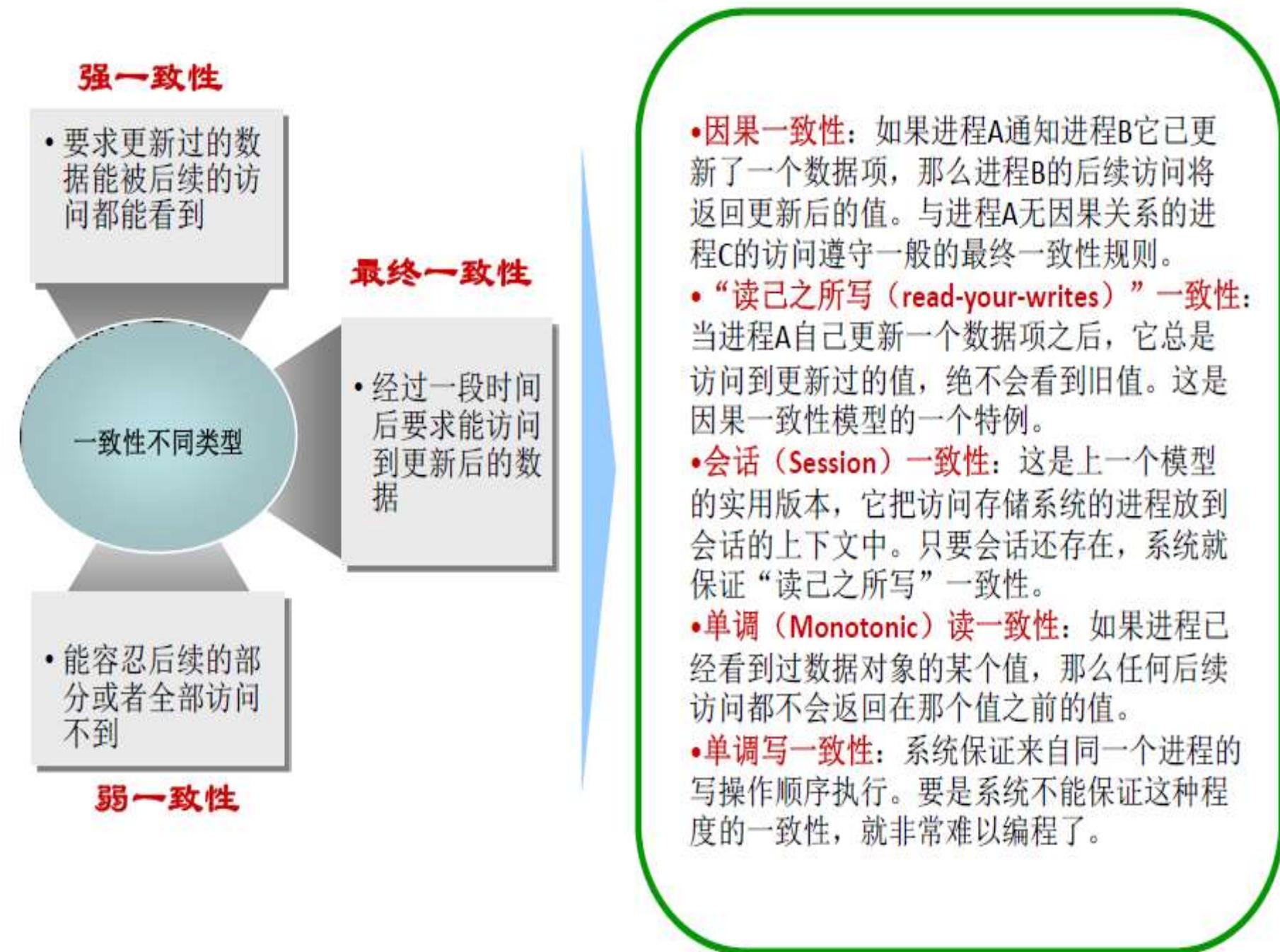
三、NoSQL

□ BASE模型

- **Basically Available(基本可用):** 支持分区失败
数据在多个节点中被复制与分区,故障时可用
- **Soft-state(软状态):** 状态可以有一段时间不同步
往往出现在数据更改但没有完全复制之后
- **Eventually consistent(最终一致):** 最终数据是一致的
 - 过程松, 结果紧, 最终结果必须保持一致
 - 无须时时一致
 - 在数据完全被复制后, 最终会恢复一致性状态

不同程度的一致性

- 强一致性（即时一致性）
- 弱一致性（存在“不一致性窗口”）
- 最终一致性
 - 弱一致性的一种特例
- e.g. DNS系统
 - 当更新一个域名的IP以后，根据配置策略以及缓存控制策略的不同，最终所有的客户都会看到最新的值。
 - 假如A首先**write**了一个值到存储系统，存储系统保证如果在A,B,C后续读取之前没有其它写操作更新同样的值的话，最终所有的读取操作都会读取到最A写入的最新值。

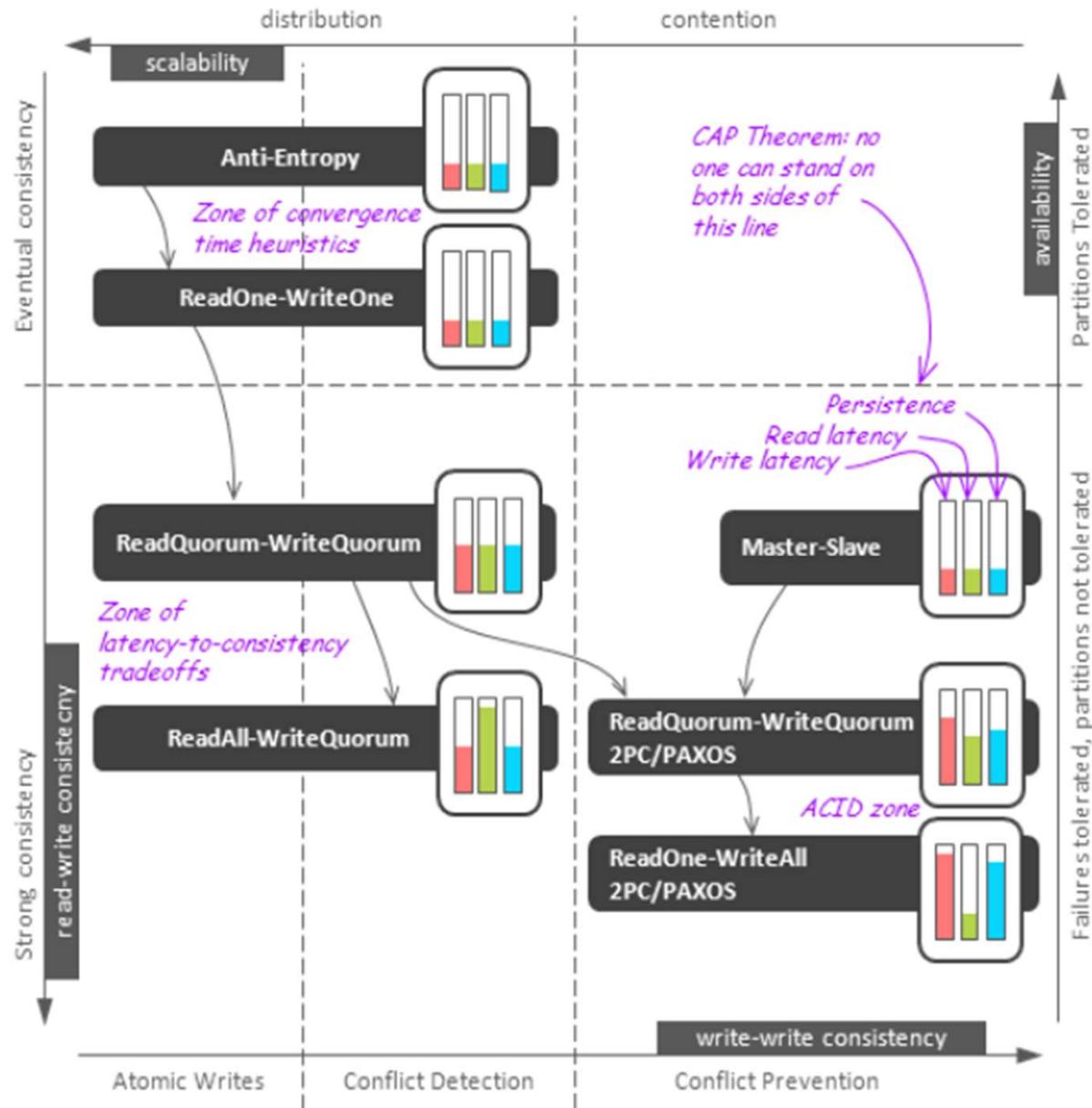


三、NoSQL

- **BASE**思想主要强调基本的可用性
- 和**ACID**模型相反，**BASE**模型是以牺牲高一致性，来获得可用性或可靠性的
- **CAP**, **BASE**和最终一致性是**NoSQL**数据库存在的三大基石

NoSQL的分布式策略：复制

- 数据一致性：数据复制和数据恢复。
- 分区容错性：如何分布以及调整数据分布才能够及时解决故障
 - 数据放置



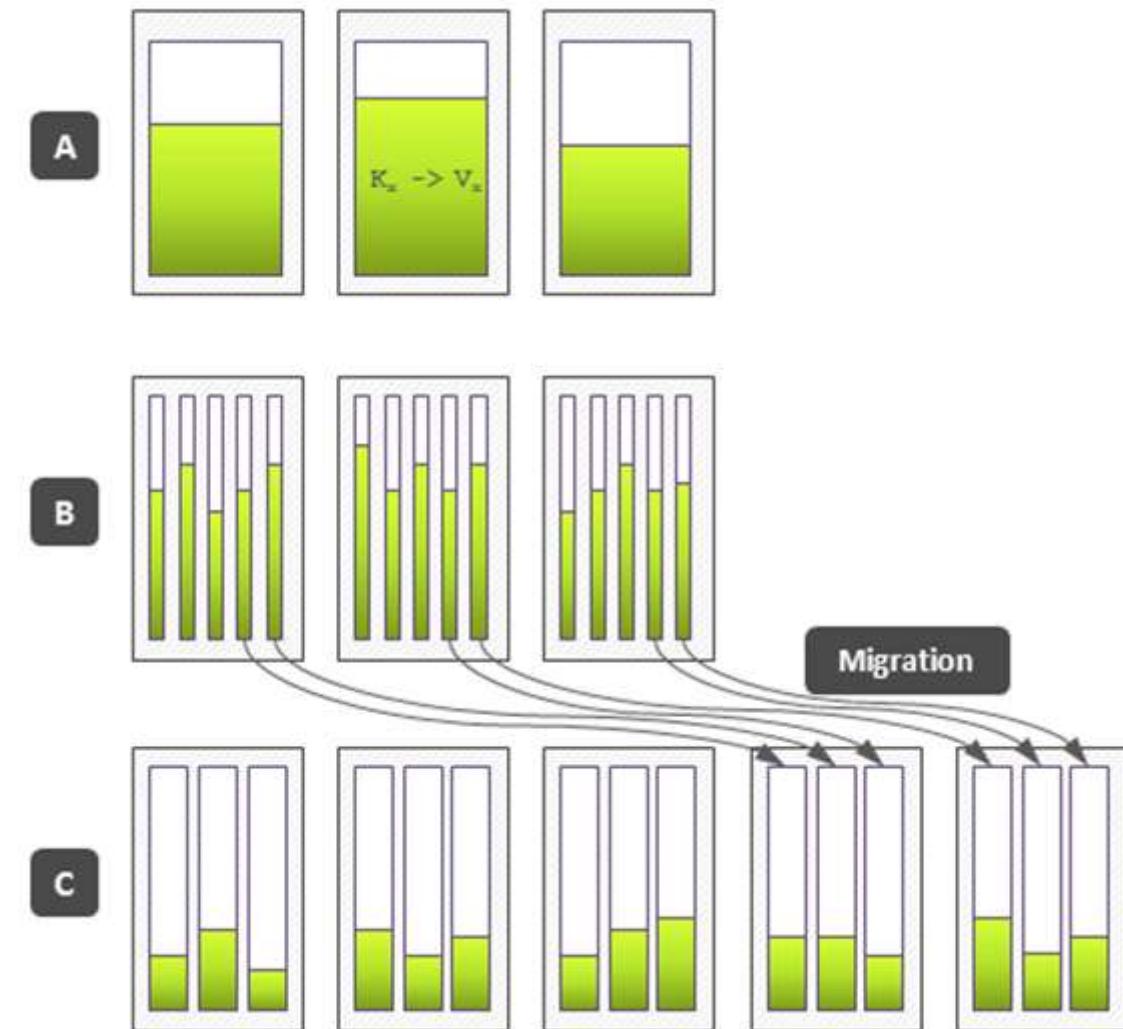
NoSQL的分布式策略：数据放置

□ 把数据项映射到合适的物理节点上，在节点间迁移数据，资源的全局调配

A:数据随便分布在三个节点上

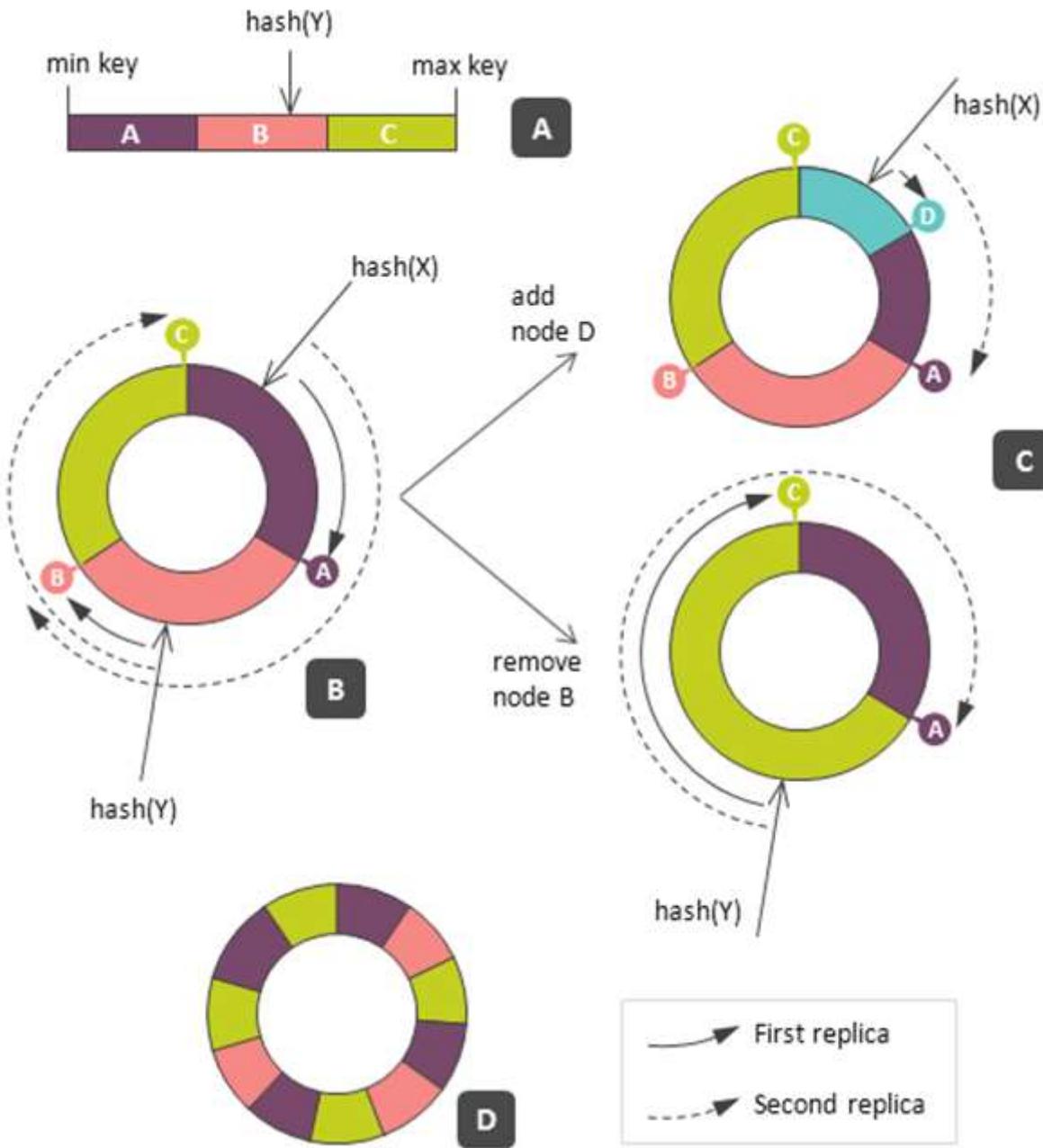
B:数据库不支持数据内部均衡，就要在每个节点上发布数据库实例

C:进行集群扩展，将数据分片并把每个数据分片作为迁移的最小单位(自动均衡)



动态环境中的数据分片和复制

- 怎么把记录映射到物理节点？
- 同时考虑进行复制和故障恢复？
- 一致性hash



NoSQL Database Types

- ❑ **Column Store**— Each storage block contains data from only one column
- ❑ **Document Store**— stores documents made up of tagged elements
- ❑ **Key-Value Store**— Hash table of keys
- ❑ **XML Databases**
- ❑ **Graph Databases**
- ❑ **Object Oriented Databases**
- ❑ ...

三、NoSQL

□ 兴起

➤ **Google : BigTable**

- 列数据排序存储, 数据值按范围分布在多台机器, 数据更新操作有严格的一致性保证。
- 以此为基础: **Hbase+Hive, cassandra, Hypertable**

➤ **Amazon: Dynamo**

- 数据按**key**进行**hash**存储 (**key-value**)
- 类似: **Redis**

➤ 其他: 文档数据库**CouchDB**, **Graph**数据库 **Neo4J**

三、NoSQL

□ 优点

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
 - Down nodes easily replaced
 - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

三、NoSQL

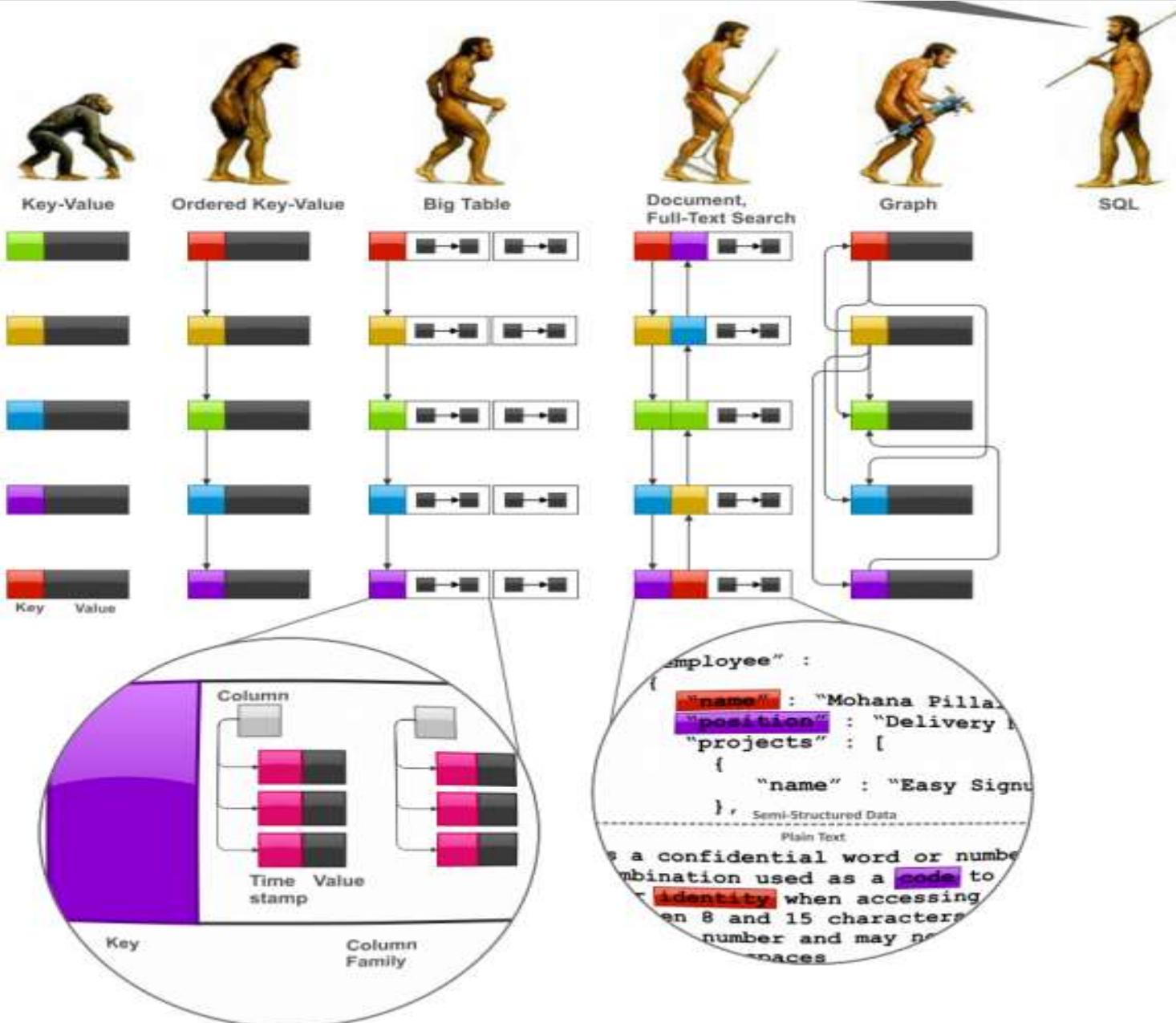
□ 缺点

- 缺乏：事务完整性+灵活的索引及查询+SQL
- 放弃了：
 - joins
 - group by
 - order by
 - ACID transactions
 - SQL强大的查询能力
 - 与支持SQL的应用程序的整合
- 缺少大规模部署应用的验证
- 外围工具不成熟

Bigtable vs. Dynamo Comparison

	Bigtable	Dynamo
Data model	Column-oriented	Key-Value
Storage layer	Commit log, Memtable, SSTables	BDB, cache
Node types	Master node and tablet nodes	Symmetric nodes
Consistency	Strong consistency	Configurable
Transactions	Per-row	None
Data locality	Sorted keys. Good for scans.	Depends on hash function

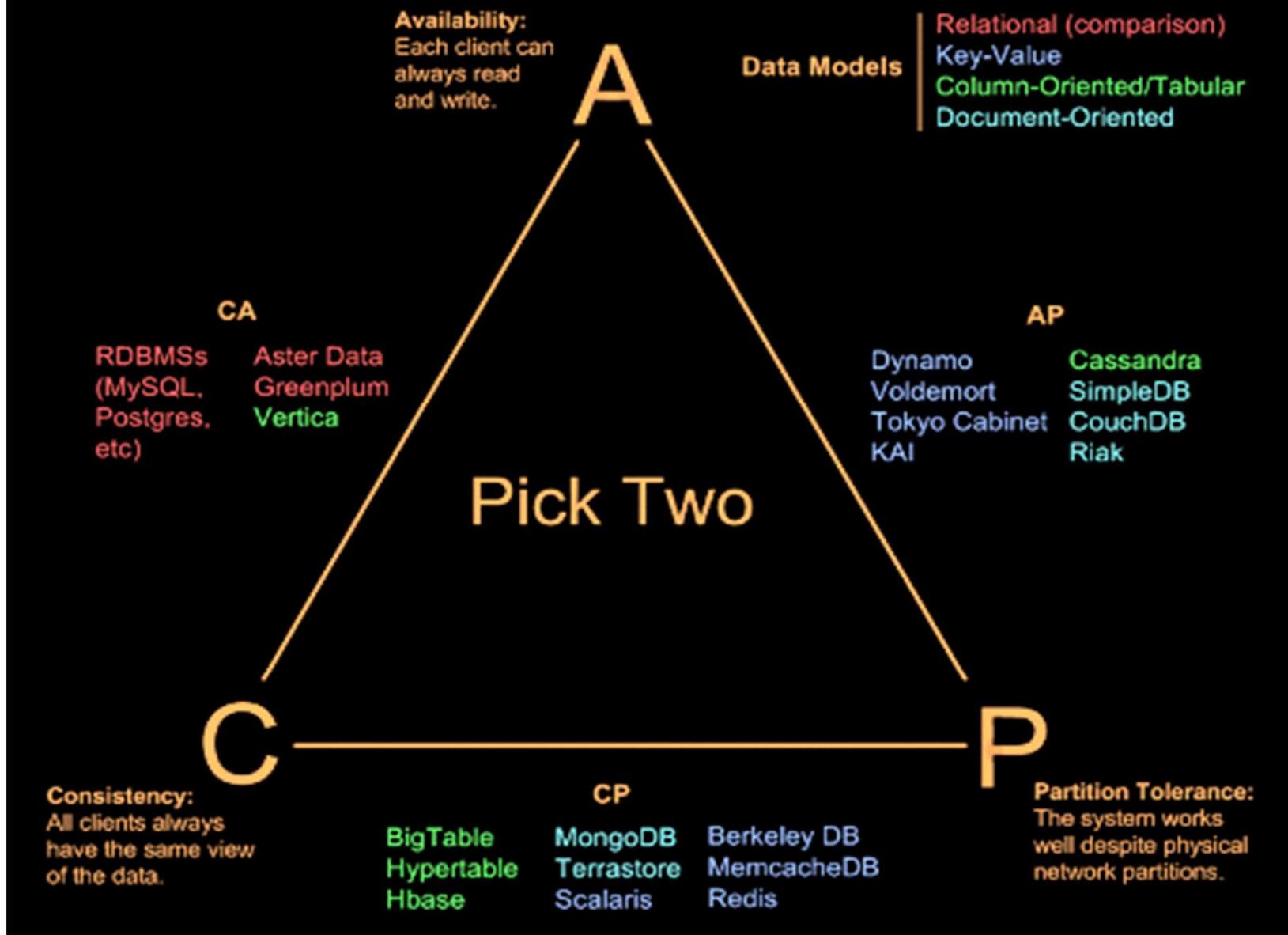
NoSQL数据模型



NoSQL数据模型

- Key-Value 存储
 - Oracle Coherence, Redis, Kyoto Cabinet
- 类BigTable存储
 - HBase, Cassandra
- 文档数据库
 - MongoDB, CouchDB, Riak
- 全文索引
 - Apache Lucene, Apache Solr
- 图数据库
 - neo4j, FlockDB, HyperGraphDB

Visual Guide to NoSQL Systems



三、NoSQL

□ 关注一致性和可用性的 (CA)

➤ 这些数据库对于分区容错性方面比较不在乎，主要采用复制(**Replication**)这种方式来保证数据的安全性

□ 常见的**CA**系统有：

- 1. 传统关系型数据库，比如**Postgres**和**MySQL**等(**Relational**)；
- 2. **Vertica (Column-oriented)**；
- 3. **Aster Data (Relational)**；
- 4. **Greenplum (Relational)**；

三、NoSQL

□关注一致性和分区容错性的(CP)

- 这种系统将数据分布在多个网络分区的节点上，并保证这些数据的一致性，但是对于可用性的支持方面有问题，比如当集群出现问题的话，节点有可能因无法确保数据是一致的而拒绝提供服务

□主要的CP系统有：

- **1. Column-oriented**
 - BigTable, Hypertable, Hbase
- **2. Document**
 - MongoDB, Terrastore
- **3. Key-value**
 - Redis, Scalaris, MemcacheDB, Berkeley DB

三、NoSQL

- 关注可用性和分区容错性的(**AP**)
 - 这类系统主要以实现“最终一致性(**Eventual Consistency**)”来确保可用性和分区容错性
- **AP**的系统有：
 - 1. **Dynamo (Key-value)**;
 - 2. **Voldemort (Key-value)** ;
 - 3. **Tokyo Cabinet (Key-value)** ;
 - 4. **KAI (Key-value)** ;
 - 5. **Cassandra (Column-oriented)** ;
 - 6. **CouchDB (Document-oriented)** ;
 - 7. **SimpleDB (Document-oriented)** ;
 - 8. **Riak (Document-oriented)** ;

三、NoSQL

公司	NoSQL产品
Google	BigTable
Facebook	Cassandra
Amazon	Dynamo
Apache	HBase
日本第一大SNS网站mix	Tokyo Cabinet (TC)
日本第二大SNS网站green.jp	Flare
淘宝网	Tair (淘宝网自主开发的Key/Value结构数据 存储系统)
新浪网	memcachedb

NoSQL Example: Column Store

- Each storage block contains data from only one column
- Example: Hadoop/Hbase
 - <http://hadoop.apache.org/>
 - Yahoo, Facebook
- Example: Ingres VectorWise
 - Column Store integrated with an SQL database
 - <http://www.ingres.com/products/vectorwise>

Column Store

- More efficient than row (or document) store if:
 - Multiple row/record/documents are inserted at the same time so updates of column blocks can be aggregated
 - Retrievals access only some of the columns in a row/record/document

- 方便存储结构化和半结构化数据
- 方便数据压缩
- 对针对某一列或者某几列的查询有非常大的I/O优势

Cassandra

- ❑ Originally developed at Facebook
- ❑ Follows the BigTable data model: column-oriented
- ❑ Uses the Dynamo Eventual Consistency model
- ❑ Written in Java
- ❑ Open-sourced and exists within the Apache family
- ❑ Uses Apache Thrift as it's API
 - Created at Facebook along with Cassandra

Cassandra

❑ column:value: timestamp

Twitter Keyspace

Users

User_id	name	email
1	hellodba	freezr@gmail.com
2	fennng	dbanotes@gmail.com

Statuses

Status_id	Text
100	DBA Tips: ...
101	@Fennng ...
102	@Hellodba ...
103	DBAnotes: ...
104

StatusRelationships

User_id	user_timeline	timeUUID:101	timeUUID:108...
1	timeUUID:100	timeUUID:101	timeUUID:108...
2	timeUUID:102	timeUUID:103	timeUUID:200...

UserRelationships

user_id	friends_timeline	timeUUID:5...
1	timeUUID:2	timeUUID:5...
2	timeUUID:1	timeUUID:10

:Users name

Key	Value
1	hellodba
2	fennng
3	donny
4	sky

:Users email

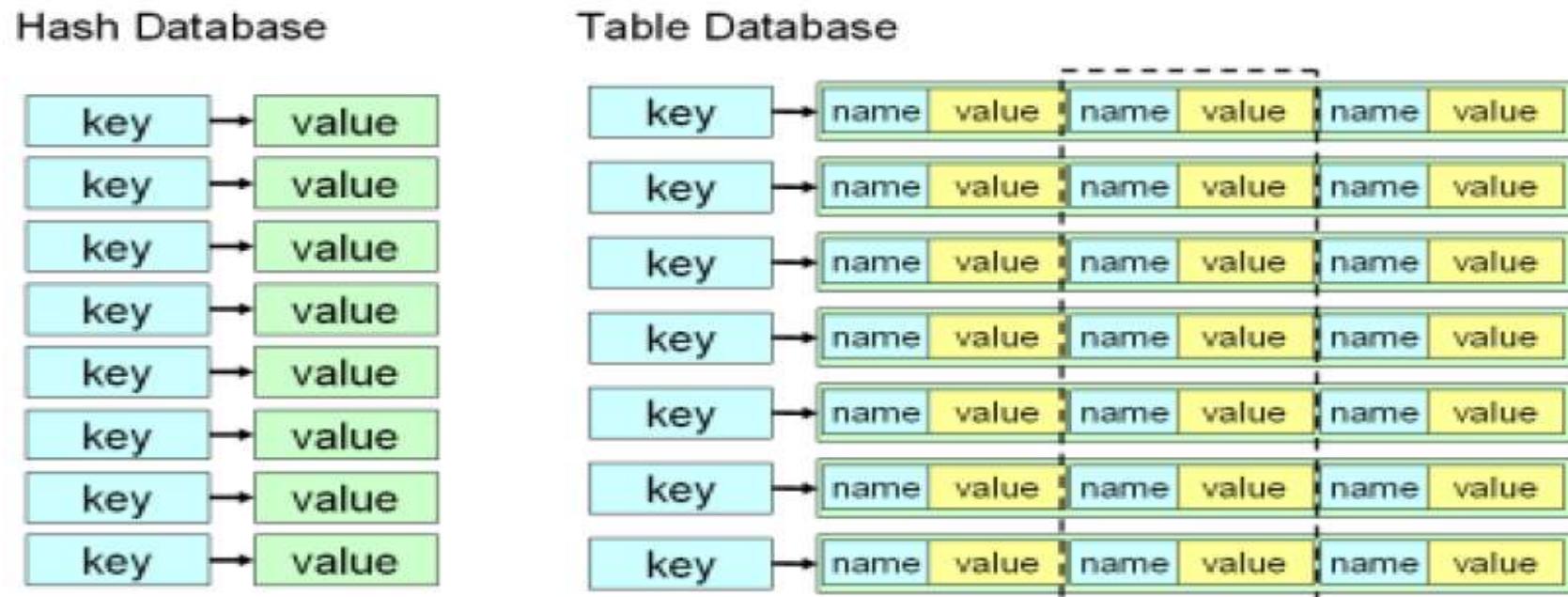
Key	Value
1	freezr@gmail.com
2	dbanotes@gmail.com
3	donny@oracle.com
4	sky@mysql.

:Users profile

Key	Value
1	Oracle DBA
2	Master
3	LingDao
4	MySQL DBA

NoSQL Examples: Key-Value Store

- Hash tables of Keys
- Values stored with Keys
- Fast access to small data values
- Example – Google BigTable, Amazon Dynamo, Voldemort, MemCacheDB



NoSQL Examples: Key-Value Store

Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key),
Update(key), Delete(key)

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions, “eventual
consistency”

NoSQL Example: Document Store

Like Key-Value Stores except value is document

- Data model: (key, document) pairs
- Document: JSON, XML, other semistructured formats
- Basic operations:
Insert(key,document), Fetch(key),
Update(key), Delete(key)
- Also Fetch based on document contents

NoSQL Example: Document Store

□ Example: CouchDB

- <http://couchdb.apache.org/>
- BBC

□ Example: MongoDB

- <http://www.mongodb.org/>
- Foursquare, Shutterfly

□ JSON – JavaScript Object Notation

□ 文档：一个使用**JSON**格式以**key-value**方式存储数据的结构

□ { "item": "pencil", "qty": 500, "type": "no.2" }
□ **JSON**支持嵌套结构

CouchDB JSON Example

```
{  
  "_id": "guid goes here",  
  "_rev": "314159",  
  
  "type": "abstract",  
  
  "author": "Keith W. Hare"  
  
  "title": "SQL Standard and NoSQL Databases",  
  
  "body": "NoSQL databases (either no-SQL or Not Only SQL)  
           are currently a hot topic in some parts of  
           computing.",  
  "creation_timestamp": "2011/05/10 13:30:00 +0004"  
}
```

CouchDB JSON Tags

❑ **"_id"**

➤ **GUID – Global Unique Identifier**

➤ **Passed in or generated by CouchDB**

❑ **"_rev"**

➤ **Revision number**

➤ **Versioning mechanism**

❑ **"type", "author", "title", etc.**

➤ **Arbitrary tags**

➤ **Schema-less**

➤ **Could be validated after the fact by user-written routine**

问题举例

□ 图书评论打分系统

➤ 把书籍信息和用户打分的信息存到一起

➤ 一组**document**存储到一起：**collection**

同一个**collection**里面的**document**可以不一样

相当于一张表里每一行数据的列都可以不一样

```
{  
  "id": "123",  
  "title": "NoSQL",  
  "author": "SQL",  
  "scores": 1  
},  
{  
  "userid": "abc1",  
  "nickname": "Peter",  
  "score": 3,  
},  
{  
  "userid": "def1",  
  "nickname": "Andy",  
  "score": 4,  
}  
],  
}
```

问题举例

□ 开发要求：

要在书籍信息页面看到所有评论，评论人的信息和打的分也要出现。

一句话评论：共16条



安迪斯晨风 给本书评分：★★★★★

2013-06-26 22:03:04

如果把小说比作做菜，这本书的材料都是非常平淡无奇甚至早被人做滥了的，主角更是集各大狗血于一身。但是经过作者妙手烹制，居然味道还不错，能让人吃得香甜满口还有回味空间。众多大俗已极的情节让他巧妙的安排出一个又一个的悬念和高潮，这就可见猫腻老爷写作功力高明

0 0

分享 添加回复



yippee 给本书评分：★★★★★

2013-07-23 22:12:08

好看的，经典

0 0

分享 添加回复



pencil 给本书评分：★★★★★

2013-07-28 16:53:04

猫腻的书都太装b，但是可以读一读

0 0

分享 1条回复

NoSQL: 只要根据书籍**ID**查询一次就能得到结果
用户为一本书籍打分只需要在一个**document**
里面添加数据

问题举例

□SQL

- 1. 根据书籍**Id**取到书籍信息。
- 2. 根据书籍**Id**取到所有评论信息。
- 3. 根据评论信息中的用户**Id**取到相关用户的信息。
- 4. 根据书籍**Id**和用户**Id**取到打分信息。

问题举例

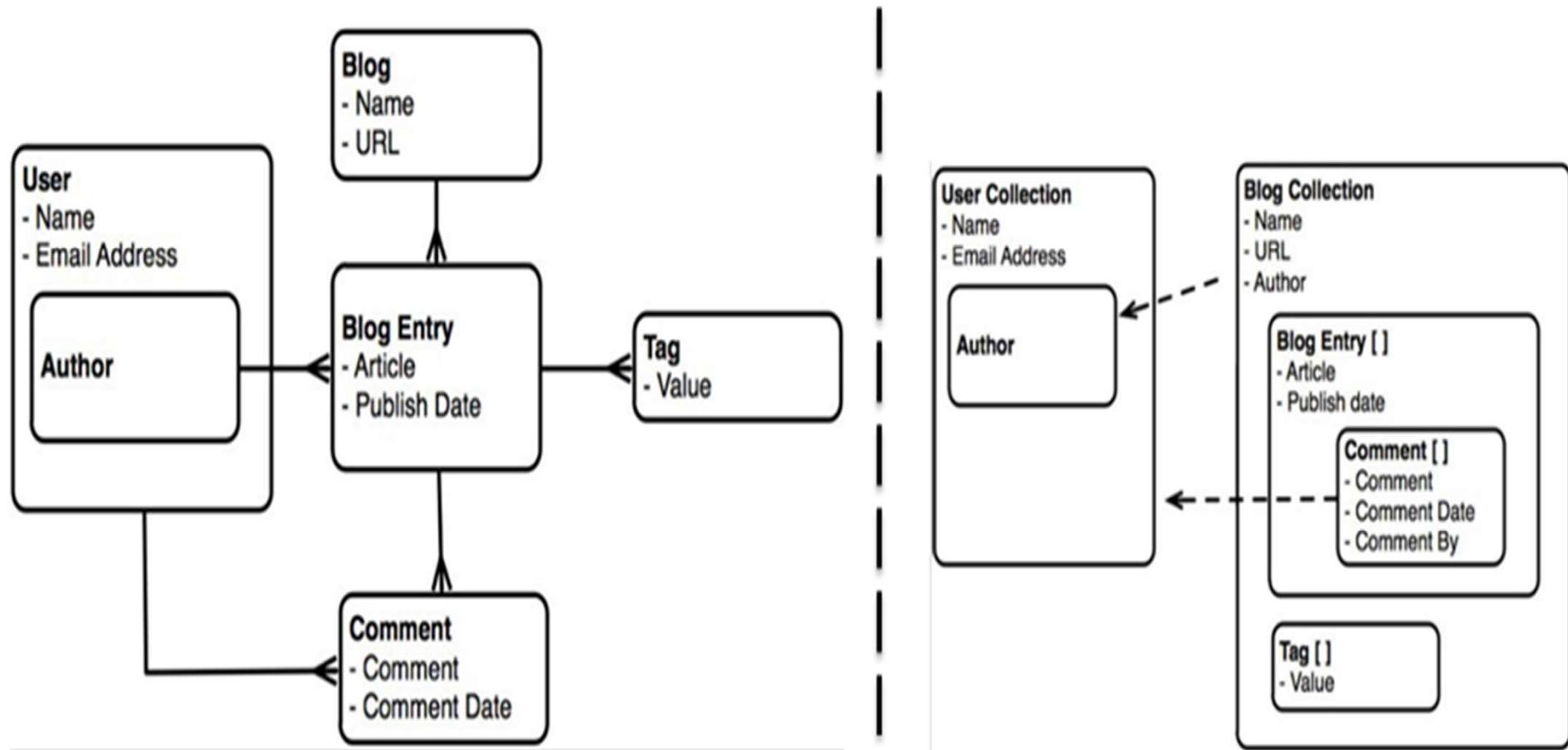
□ 应用需求进一步增加：
书籍详细信息页面上还要显示书评的创建时间

NoSQL: 在“书评”结构中增加**Time**，并增加赋值代码。

不需要去修改历史数据（因为同一个
collection里面的**document**可以不一样）
历史数据中的**Time**会被置为空

SQL: 1. 创建一个为**Review**表增加”**Time**”列的**sql**脚本。
2. 到数据库中运行。
3. 修改相关代码和存储过程。

Relation vs. Document



MongoDB

- 非关系数据库中功能最丰富，最像关系数据库
- 主要解决的是海量数据的访问效率问题
 - 数据量达到**50GB**以上时，数据库访问速度是**MySQL**的 10倍以上
- 数据结构：类似**json**的**bjson**格式
 - 可以存储比较复杂的数据类型
- 自带分布式文件系统**GridFS**，可支持海量数据存储
- 数据查询功能强大

Map Reduce

- Technique for indexing and searching large data volumes
- Two Phases, Map and Reduce
 - Map
 - Extract sets of Key-Value pairs from underlying data
 - Potentially in Parallel on multiple machines
 - Reduce
 - Merge and sort sets of Key-Value pairs
 - Results may be useful for other searches

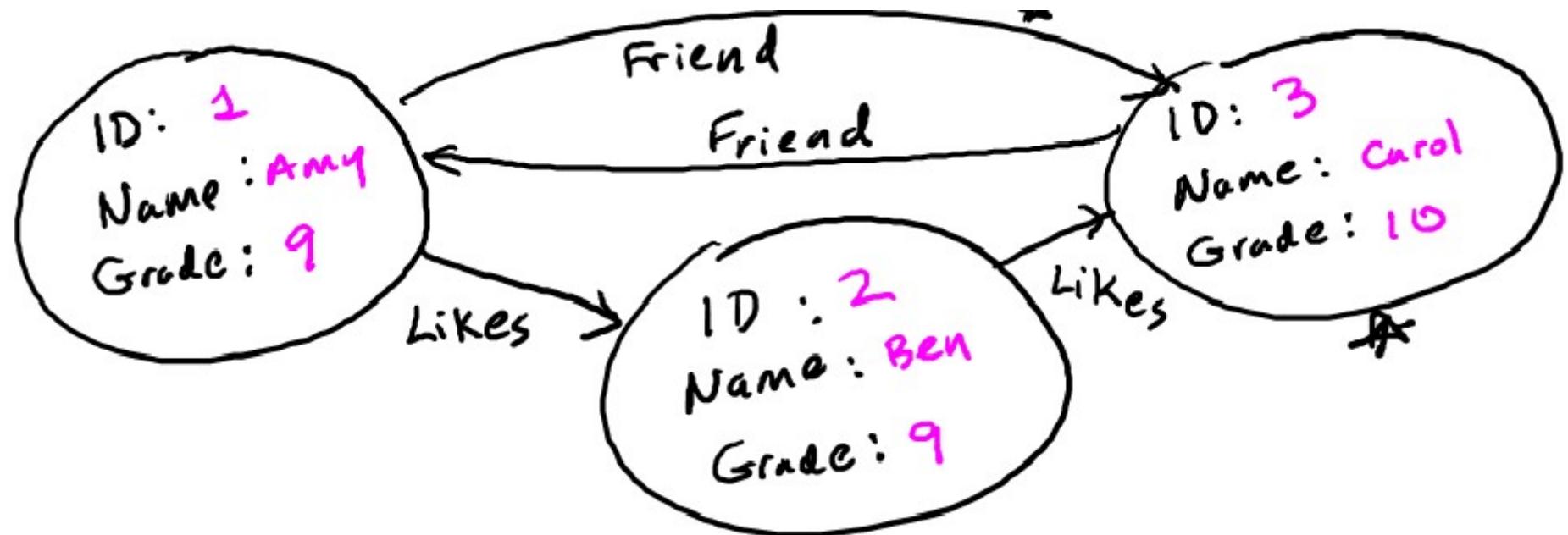
Map Reduce

- Map Reduce techniques differ across products
- Google granted US Patent 7,650,331, January 2010
System and method for efficient large-scale data processing

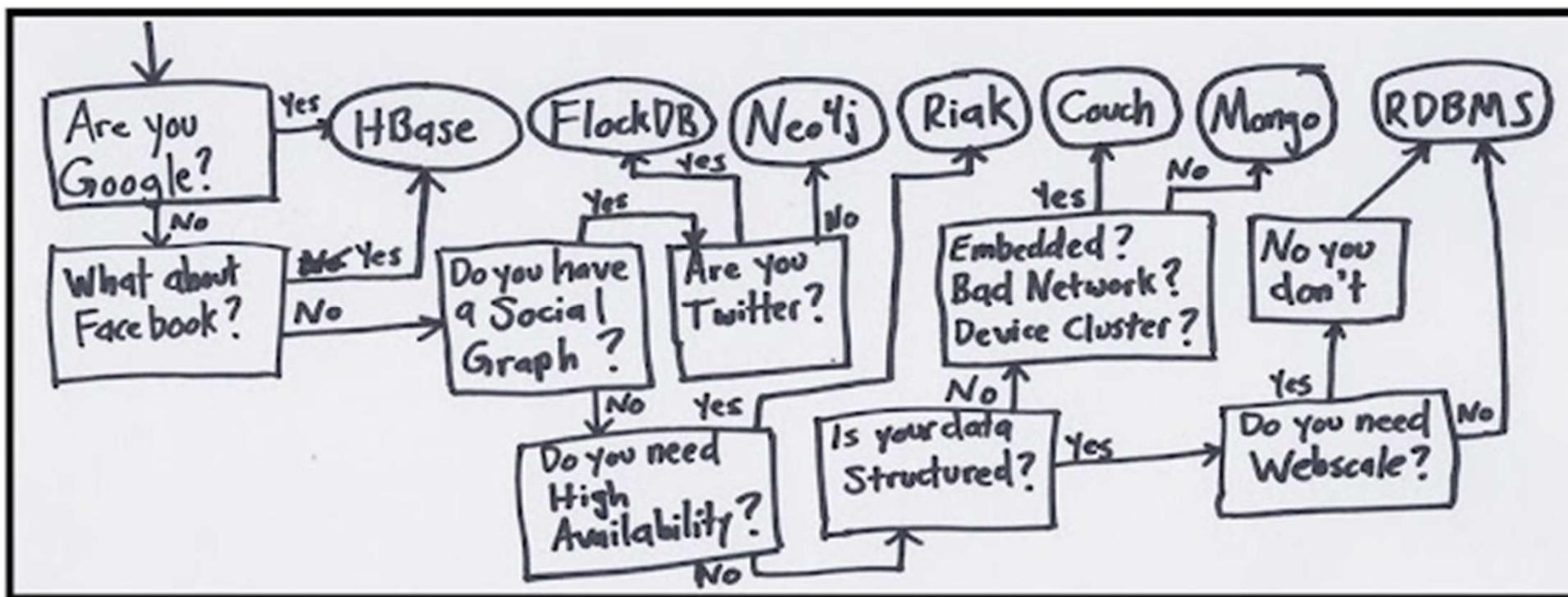
A large-scale data processing system and method includes one or more application-independent map modules configured to read input data and to apply at least one **application-specific map operation** to the input data to produce intermediate data values, wherein the map operation is automatically parallelized across multiple processors in the parallel processing environment. A plurality of intermediate data structures are used to store the intermediate data values. One or more application-independent reduce modules are configured to retrieve the intermediate data values and to apply at least one **application-specific reduce operation** to the intermediate data values to provide output data.

Graph Database

- Data model: **nodes and edges**
- Nodes may have properties (including ID)
- Edges may have labels or roles



- Examples: **Neo4j, FlockDB, Pregel, ...**



	Data Model	Query API
Cassandra	Columnfamily	Thrift
CouchDB	Document	map/reduce view
HBase	Columnfamily	Thrift, REST
MongoDB	Document	Cursor
Neo4J	Graph	Graph
Redis	Collection	Collection
Riak	Document	Nested hashes
Scalarmis	Key/value	get/put
Tokyo Cabinet	Key/value	get/put
Voldemort	Key/value	get/put

NoSQL Summary

- NoSQL databases reject:
 - Overhead of ACID transactions
 - “Complexity” of SQL
 - Burden of up-front schema design
 - Declarative query expression
 - Yesterday’s technology
- Programmer responsible for
 - Step-by-step procedural language
 - Navigating access path

SQL vs. NoSQL

□ SQL Databases

- Predefined Schema
- Standard definition and interface language
- Tight consistency
- Well defined semantics

□ NoSQL Database

- No predefined Schema
- Per-product definition and interface language
- Getting an answer quickly is more important than getting a correct answer

NoSQL CAN & CAN'T

Can do:

支持大型数据集与低延迟操作
可以用来应对特定的数据类型

Can't do:

不能提供确保数据完整并降低数据冗余
和不一致性的粒度控制能力

Google: F1 + Spanner = MPP数据库

NoSQL = No time to support SQL!