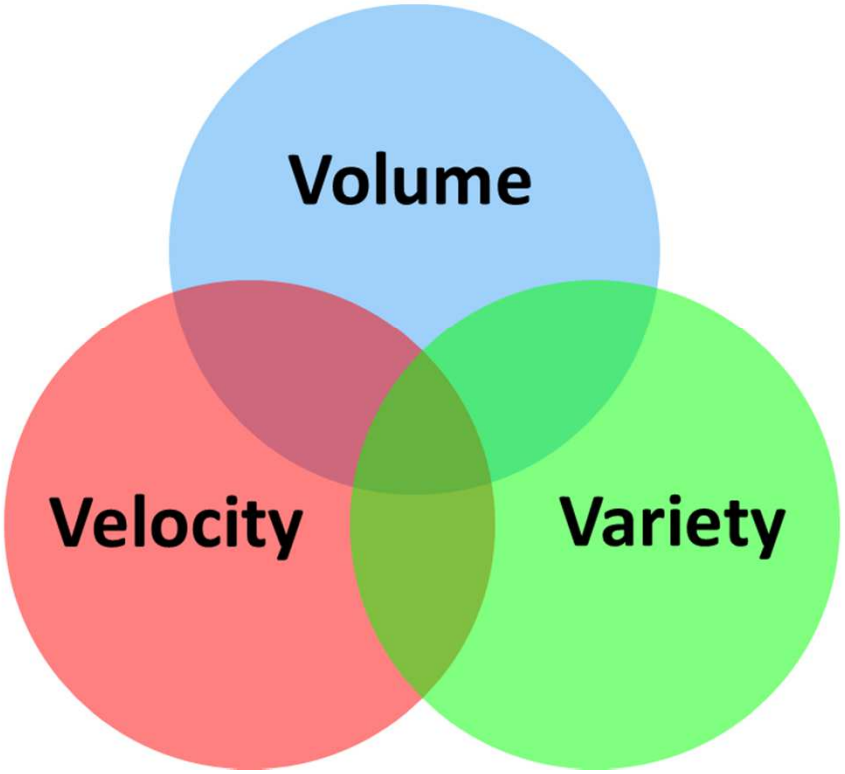
大数据系统与大规模数据分析

大数据运算系统（3）



陈世敏

中科院计算所
计算机体系结构
国家重点实验室

Outline

- MapReduce + SQL系统
- 内存计算
  - 内存数据库
  - 内存键值系统
  - 内存MapReduce

MapReduce+SQL系统介绍

- MapReduce提供了一个分布式应用编写的平台
  - 程序员开发串行的Map和Reduce函数
  - 在串行的环境开发和调试
  - MapReduce系统可以在成百上千个机器节点上并发执行MapReduce程序，从而实现对大规模数据的处理
- MapReduce的问题
  - 这是一个编程的平台
  - 不太适合数据分析师的使用
  - 即使最基础的选择和投影操作，也必须写程序实现
- 对SQL的需求由此产生

MapReduce+SQL系统介绍

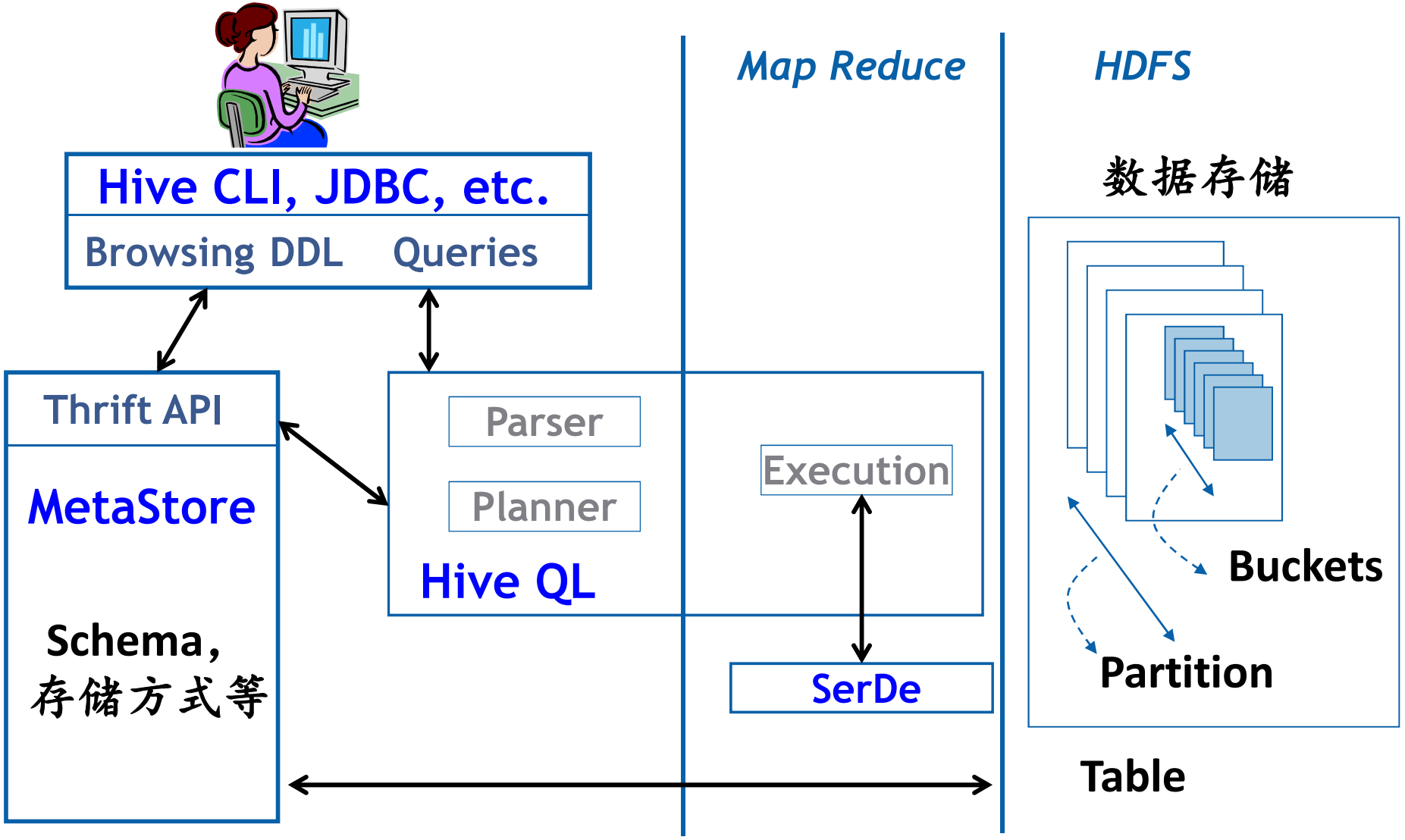
- 产业界研发了许多系统，希望在云平台上增加一层类似SQL的支持
- 这类系统包括
  - Facebook Hive
  - Yahoo Pig
  - Microsoft Scope
  - Google Sawzall
  - IBM Research JAQL
- 一些数据仓库产品也把云计算的能力集成进Execution engine
  - Greenplum, Aster Data, Oracle
  - 基于内部的某种MapReduce实现或者Hadoop
- Hive不是最早出现的，也不是最具创新性的，但是目前它被非常广泛地使用，我们主要介绍Hive

Hive

- Hive是蜂巢
- 简要发展
  - 2008年，Facebook由于数据分析的需求研发了Hive
  - Facebook公开了Hive的源码，Hive成为Apache开源项目
  - 2008年3月，Facebook每天把200GB数据存入Hive系统
  - 2012年，每日存入的数据量超过了15TB
- Hive
  - 管理和处理结构化数据
  - 在Hadoop基础上实现
  - 提供类似SQL的HiveQL语言

Hive系统

Adapted from Hive ApacheCon'08 talk

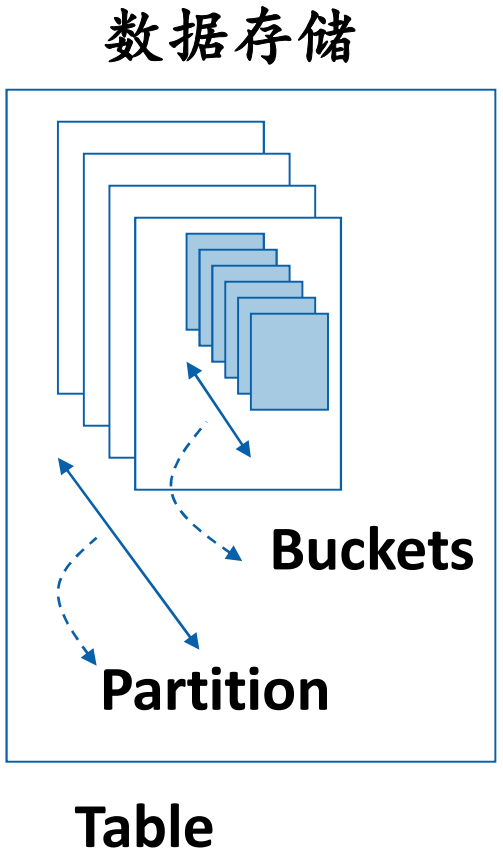


Hive系统

Adapted from Hive ApacheCon'08 talk

Map Reduce | *HDFS*

- 数据存储在HDFS上
  - hdfs目录：/usr/hive/warehouse/
- Table： 一个单独的hdfs目录
  - /user/hive/warehouse/表名
- Table可以进一步划分为Partition
- Partition可以进一步划分为Bucket

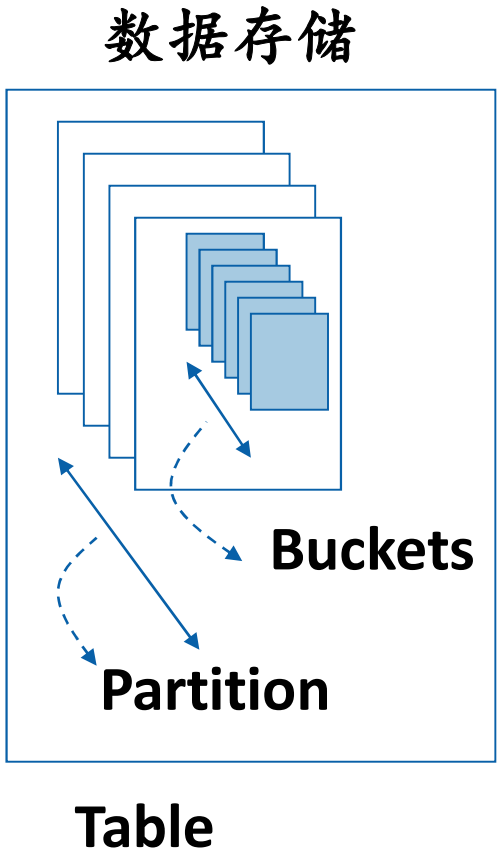


Hive系统

Adapted from Hive ApacheCon'08 talk

Map Reduce | *HDFS*

- Partition：每个Partition是Table目录下的子目录
  - 假设pkey是partition key：
    /user/hive/warehouse/表名/pkey=value
- Bucket：每个Bucket是Partition目录下一个子目录
  - 假设pkey是partition key, bkey是bucket key：
    /user/hive/warehouse/表名
    /pkey=value/bkey=value



Hive系统

Adapted from Hive ApacheCon'08 talk

Hive CLI, JDBC, etc.

Browsing DDL Queries

Thrift API

MetaStore

Schema,
存储方式等

Parser

Planner

Hive QL

Map Reduce

Execution

SerDe

- Hive QL
  - 类似SQL
  - 部分SQL和扩展
  - 采用MapReduce执行
- SerDe
  - 序列化/反序列化

Hive系统

Adapted from Hive ApacheCon'08 talk

Hive CLI, JDBC, etc.

Browsing DDL Queries

Thrift API

MetaStore

Schema,
存储方式等

Map Reduce

HDFS

数据存储

- MetaStore
  - 存储表的定义信息等
  - 默认在本地${HIVE_HOME}/metastore_db中
  - 也可以配置存储在数据库RDBMS系统中
- Hive CLI
  - 命令行客户端，可以执行各种HiveQL命令

Hive数据模型

- 关系型表+扩展
- 关系型表
  - 无序的记录
  - 每个记录可以包含多个列
  - 每个列可以是原子数据类型
    - 例如：integer types, float, string, date, boolean

举例

```
CREATE TABLE status_updates(
          userid int,
          status string
)

STORED AS SEQUENCEFILE;
```

举例

```
CREATE TABLE status_updates(
          userid int,
          status string
)
PARTITIONED BY (ds string, hr int)

STORED AS SEQUENCEFILE;
```

注意：ds是partition key, hr是bucket key
它们都不包括在table schema中

Hive数据模型

- 关系型表+扩展
- 扩展1
  - 列可以是更加复杂的数据类型
  - ARRAY<data-type>
    - 例如： a ARRAY<int>: a[0], a[1], …
  - MAP<primitive-type, any-type>
    - 例如： m MAP<STRING, STRING>: m['key1'],…
  - STRUCT<col_name: data_type, …>
    - 例如： s STRUCT {c: INT, d: INT}： s.c, s.d

Hive数据模型

- 关系型表+扩展
- 扩展1

```
CREATE TABLE t1(
        st string,
        fl float,
        a  array<int>,
        m map<string, string>
        n array<map<string, struct<p1:int, p2:int>>
);
```

系统把复杂数据类型Serialize/deserialize (序列化/反序列化)
使用： t1.n[0]['key'].p2

Hive数据模型

- 关系型表+扩展
- 扩展2
  - 可以直接读取已有的外部数据
  - 程序员提供一个SerDe的实现
  - 只有在使用时，才转化读入

```
add jar /jars/myformat.jar;

CREATE TABLE t2
ROW FORMAT SERDE 'com.myformat.MySerDe';
```

Create/Alter/Drop Table

- 支持SQL的DDL (data definition language)
  - Create table
  - Alter table
  - Drop table

Insert

```
Insert into table status_updates values (123,
‘active’), (456, ‘inactive’), (789, ‘active’);

Insert into table status_updates
  select 语句

Insert overwrite table status_updates
  select 语句
```

Insert into是文件append

Insert overwrite是删除然后新创建文件

Partition使用举例

```
INSERT OVERWRITE TABLE
status_updates PARTITION(ds='2009-01-01', hr=12)
SELECT * FROM t;
```

在如下的子目录中，存储select的输出
/user/hive/warehouse/status_updates/ds=2009-01-01/hr=12

划分是手工进行的！

```
SELECT * FROM status_updates WHERE ds='2009-01-01';
```

ds是partition key， 所以Hive只使用对应的子目录中的数据

表达MapReduce

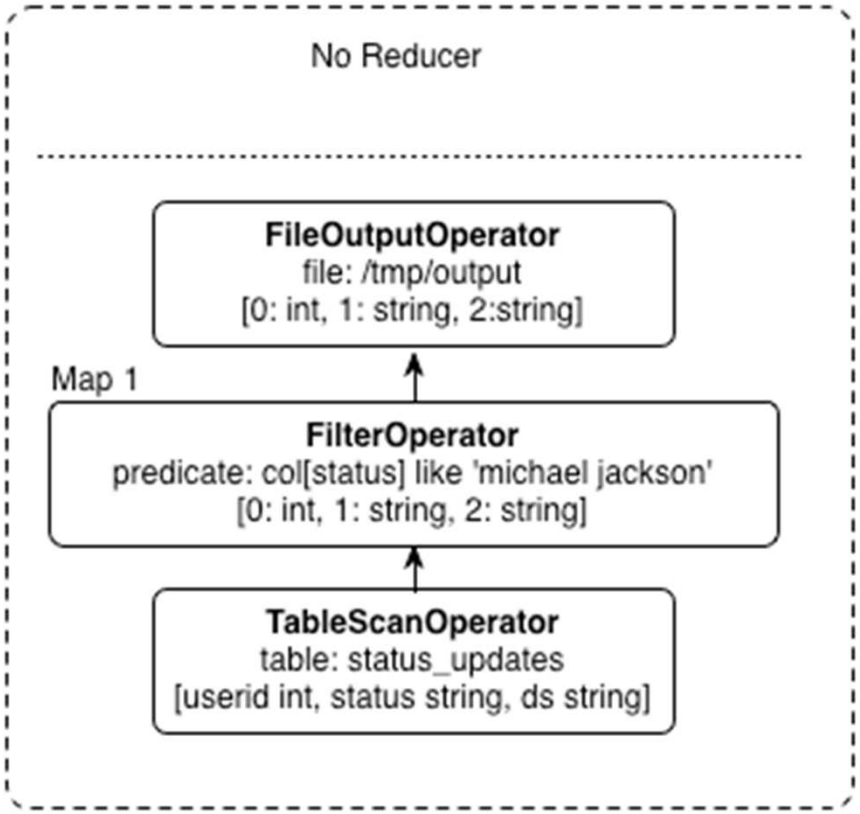
- 扩展：可以表达MapReduce
  - FROM (
    MAP doctext USING 'python wc_mapper.py' AS (word, cnt)
    FROM docs
    CLUSTER BY word
) a
    REDUCE word, cnt USING 'python wc_reduce.py';

Example Query (Filter)

Hive ICDE'10 talk

```
SELECT * FROM status_updates
WHERE status LIKE 'michael jackson'
```

```
status_updates (
        userid int,
        status string
  )
Partition key: ds
```

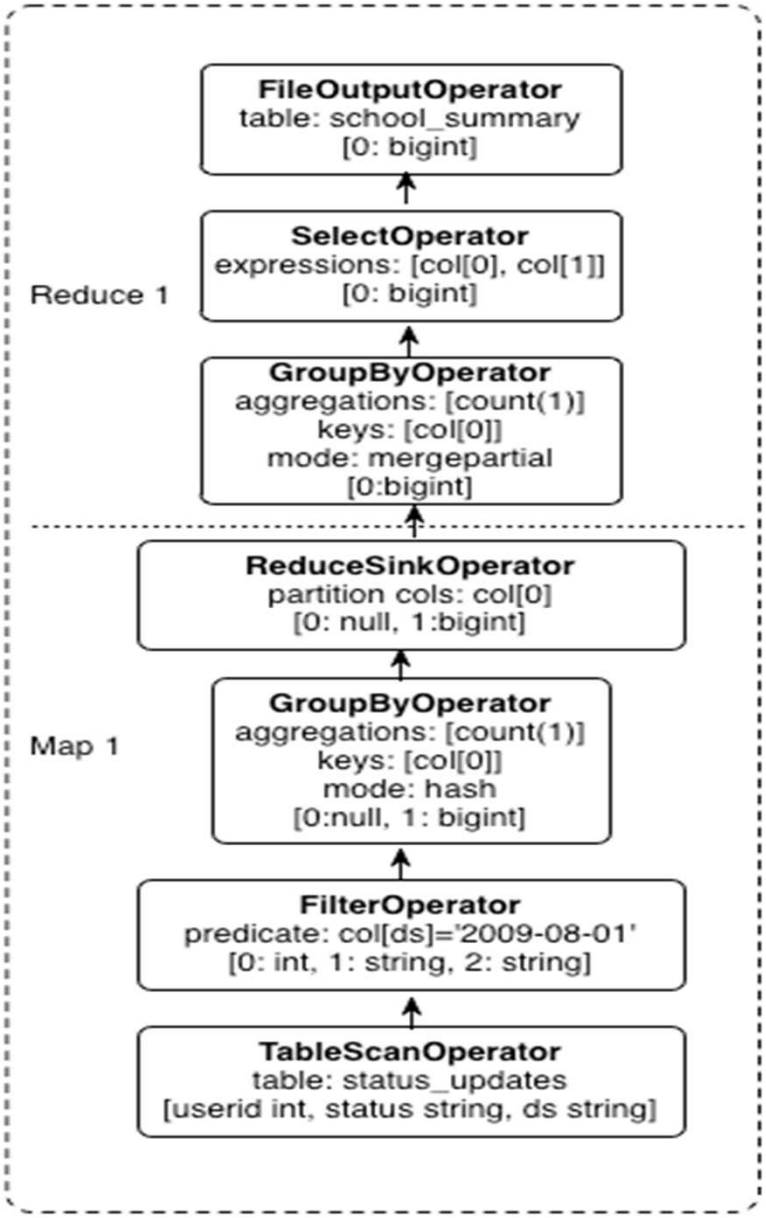


Example Query (Aggregation)

```
SELECT COUNT(1)
FROM status_updates
WHERE ds = '2009-08-01'
```

```
status_updates (
        userid int,
        status string
 )
Partition key: ds
```

Hive ICDE'10 talk

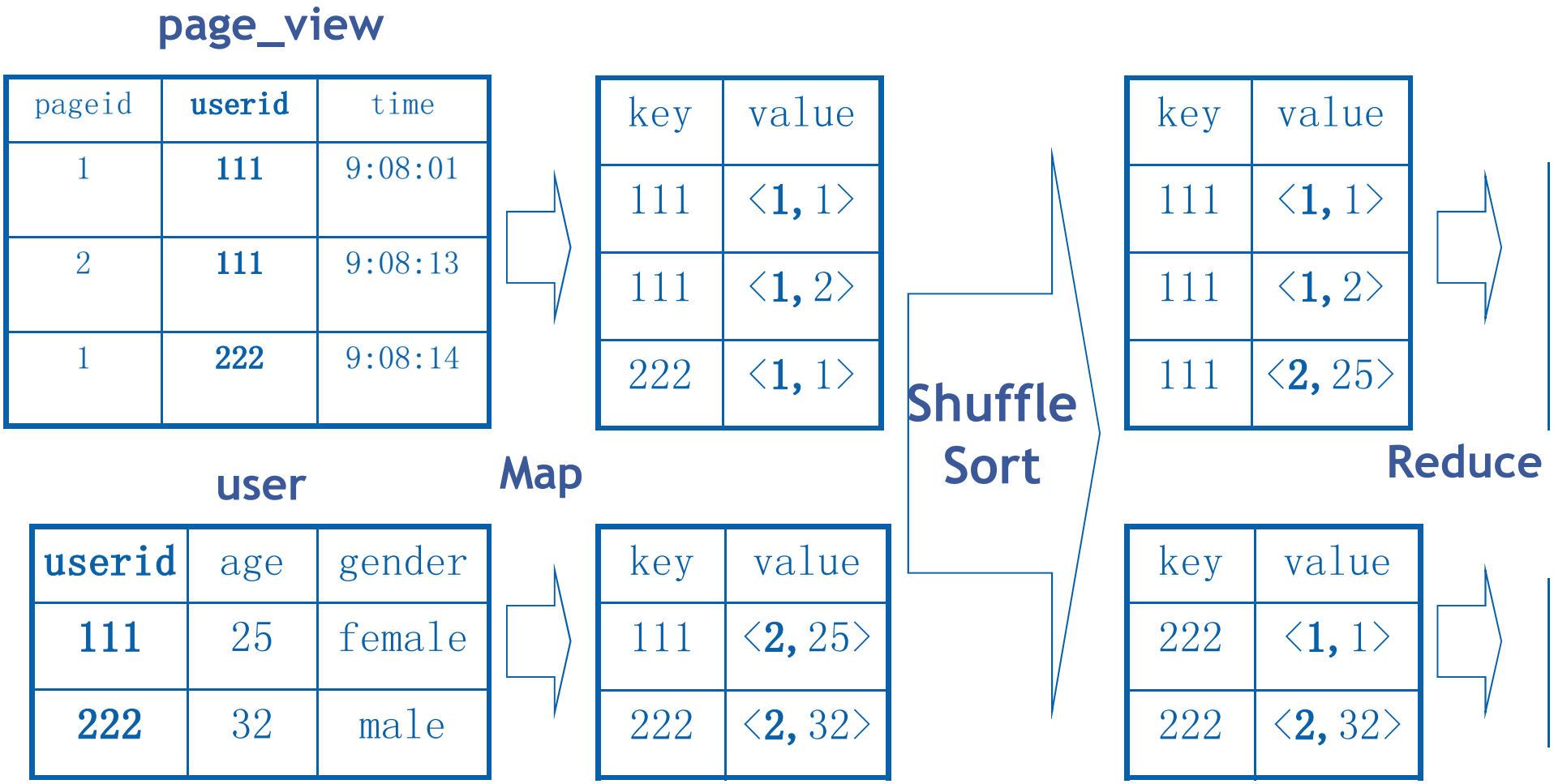


Join实现

- 普通的MapReduce Equi-join

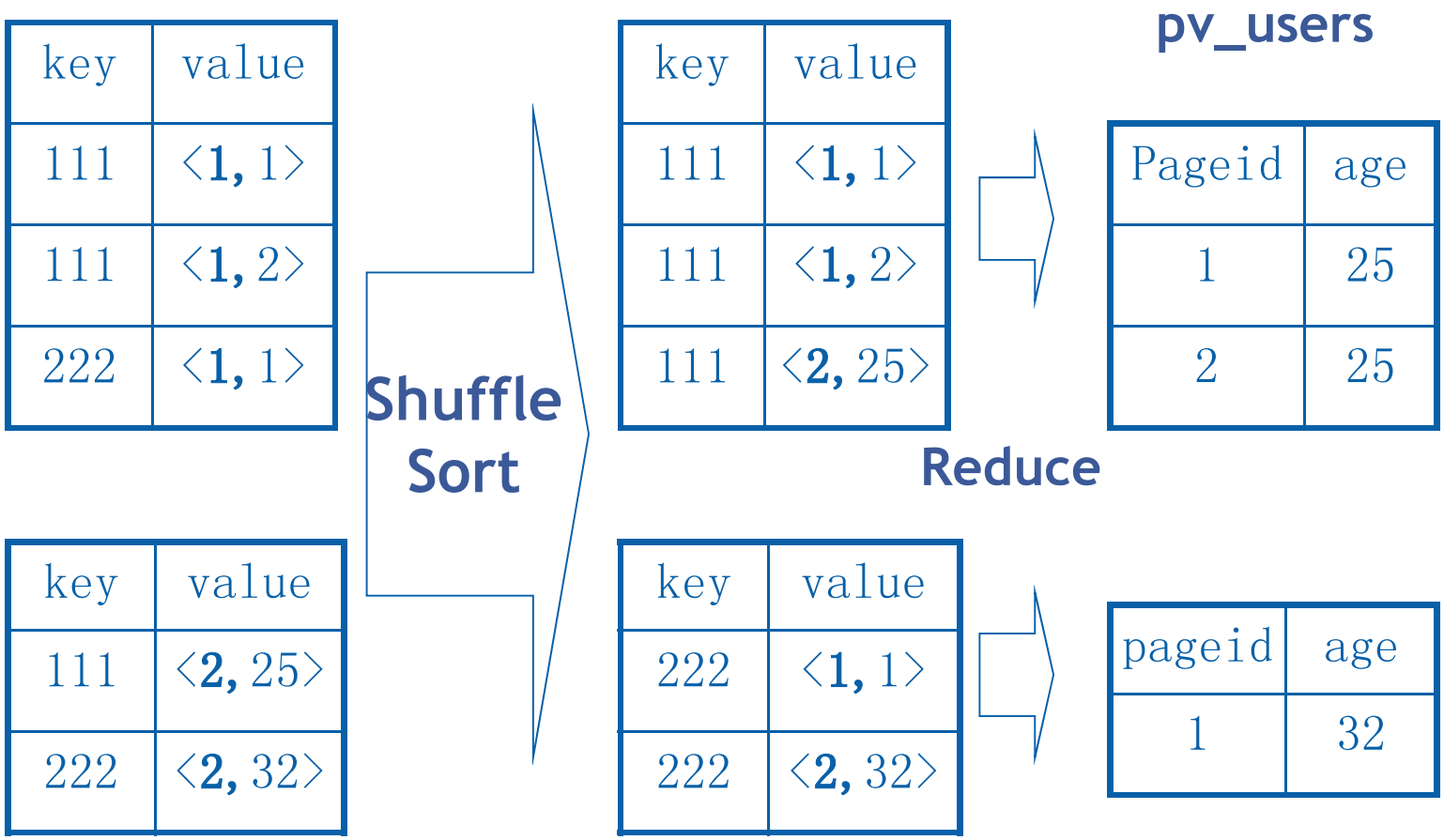
Hive QL – Join in Map Reduce

Hive Apachecon'08 talk



Hive QL – Join in Map Reduce

Hive Apachecon'08 talk

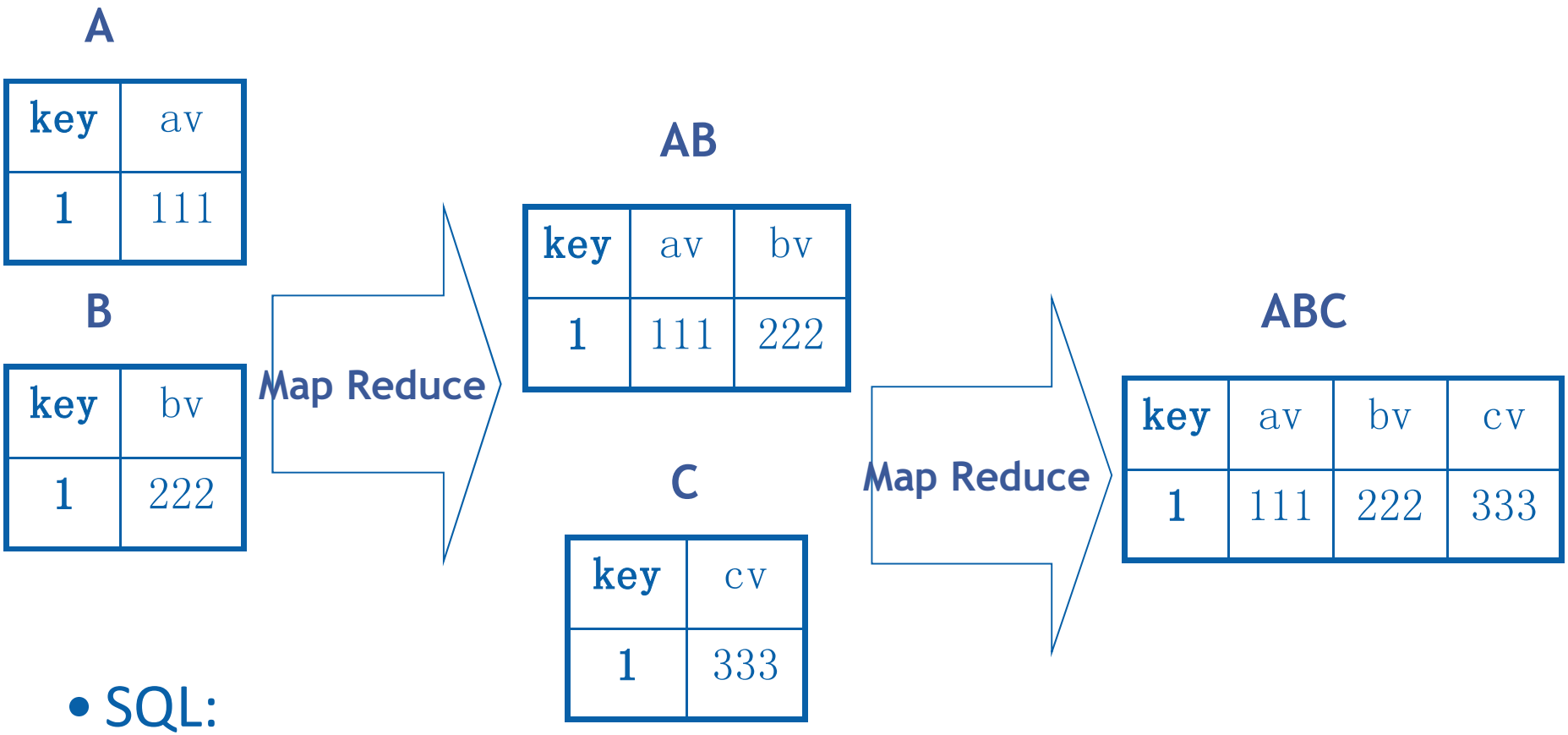


Join实现

- 普通的MapReduce Equi-join
- Multi-way join优化
  - (a.key = b.key) and (b.key = c.key)
  - 同时传输多个表

Joining multiple tables in a single Map Reduce

Hive Apachecon'08 talk

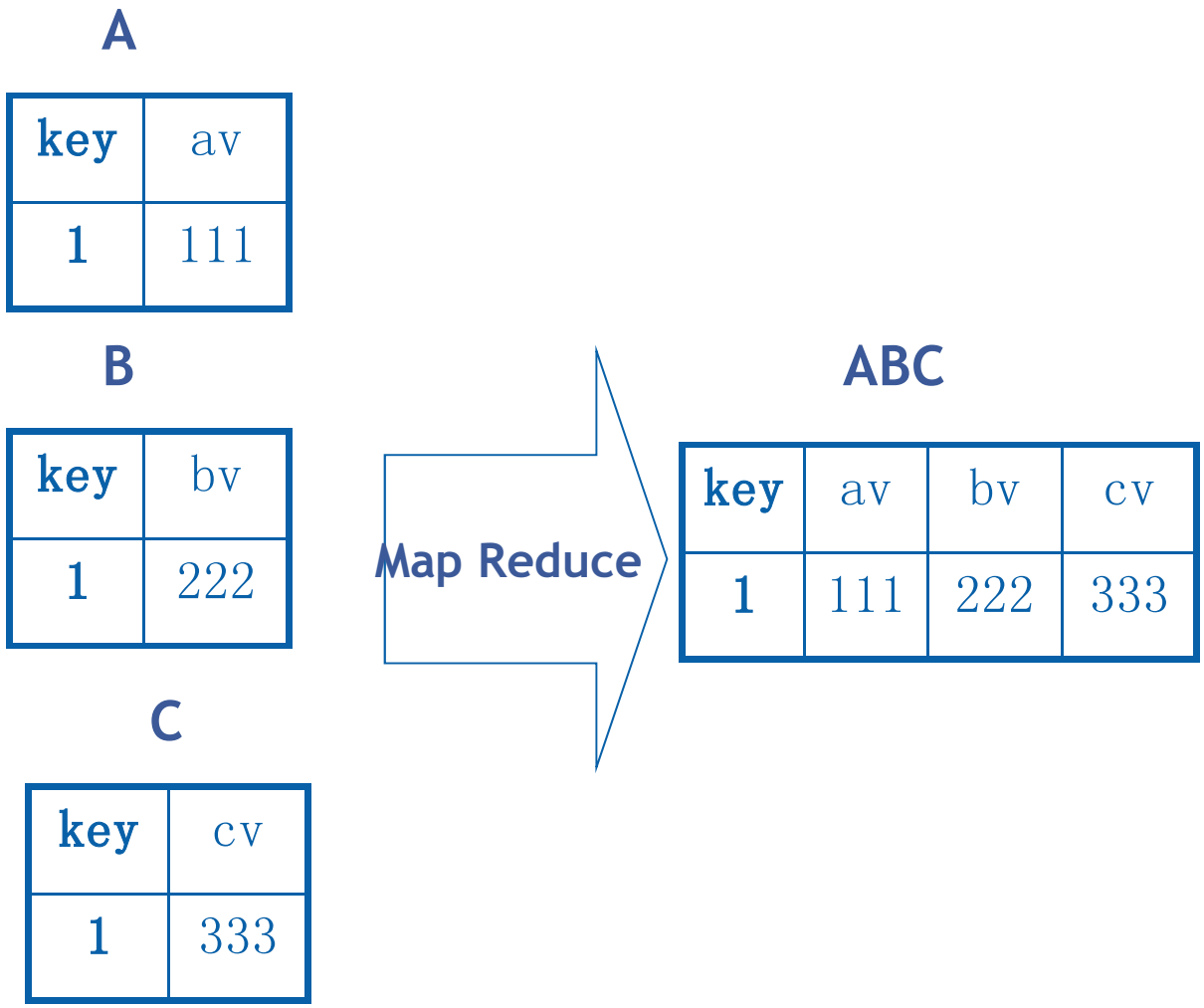


- SQL:
  - SELECT…

    FROM a join b on (a.key = b.key) join c on (a.key = c.key)

Joining multiple tables in a single Map Reduce

Hive Apachecon'08 talk

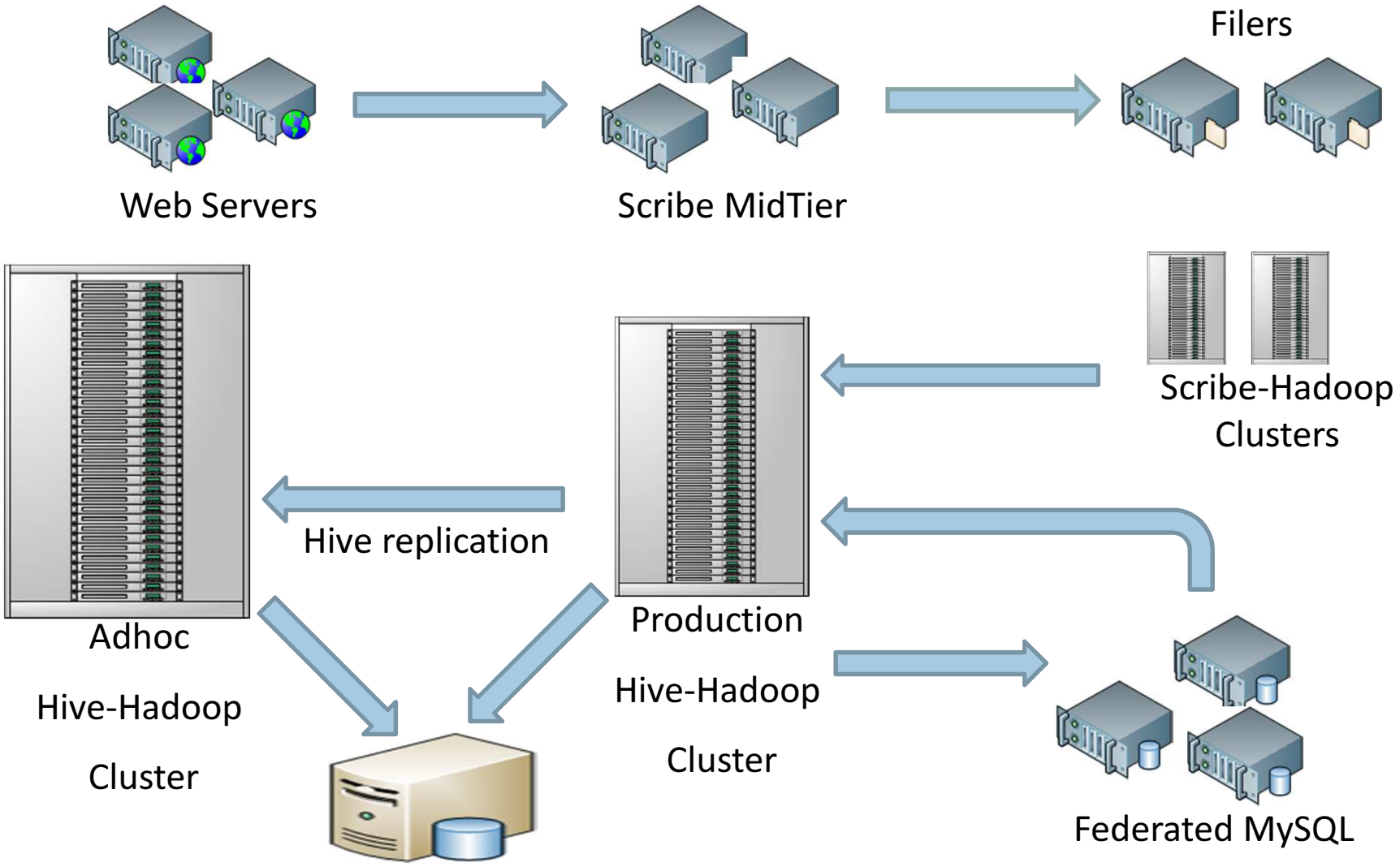


Join实现

- 普通的MapReduce Equi-join
- Multi-way join优化
  - (a.key = b.key) and (b.key = c.key)
  - 同时传输多个表
- Map-side join
  - 没有reducer
  - 其中一个表R很小
  - 那么每个Map task都读整个的R，与S的一部分join
  - Mapper在这里只是运行并行程序的容器，Map函数会完成一整个simple hash join

Data Flow Architecture at Facebook

Hive ICDE'10 talk



Hadoop & Hive Cluster @ Facebook

Hive ICDE'10 talk

- **Production Cluster**
  - 300 nodes/2400 cores
  - 3PB of raw storage
- **Adhoc Cluster**
  - 1200 nodes/9600 cores
  - 12PB of raw storage
- **Node (DataNode + TaskTracker) configuration**
  - 2CPU, 4 core per cpu
  - 12 x 1TB disk (900GB usable per disk)

Hive & Hadoop Usage @ Facebook

Hive ICDE'10 talk

- **Statistics per day:**
  - 10TB of compressed new data added per day
  - 135TB of compressed data scanned per day
  - 7500+ Hive jobs per day
  - 80K compute hours per day

- **Hive simplifies Hadoop:**
  - New engineers go though a Hive training session
  - ~200 people/month run jobs on Hadoop/Hive
  - Analysts (non-engineers) use Hadoop through Hive
  - 95% of hadoop jobs are Hive Jobs

Hive & Hadoop Usage @ Facebook

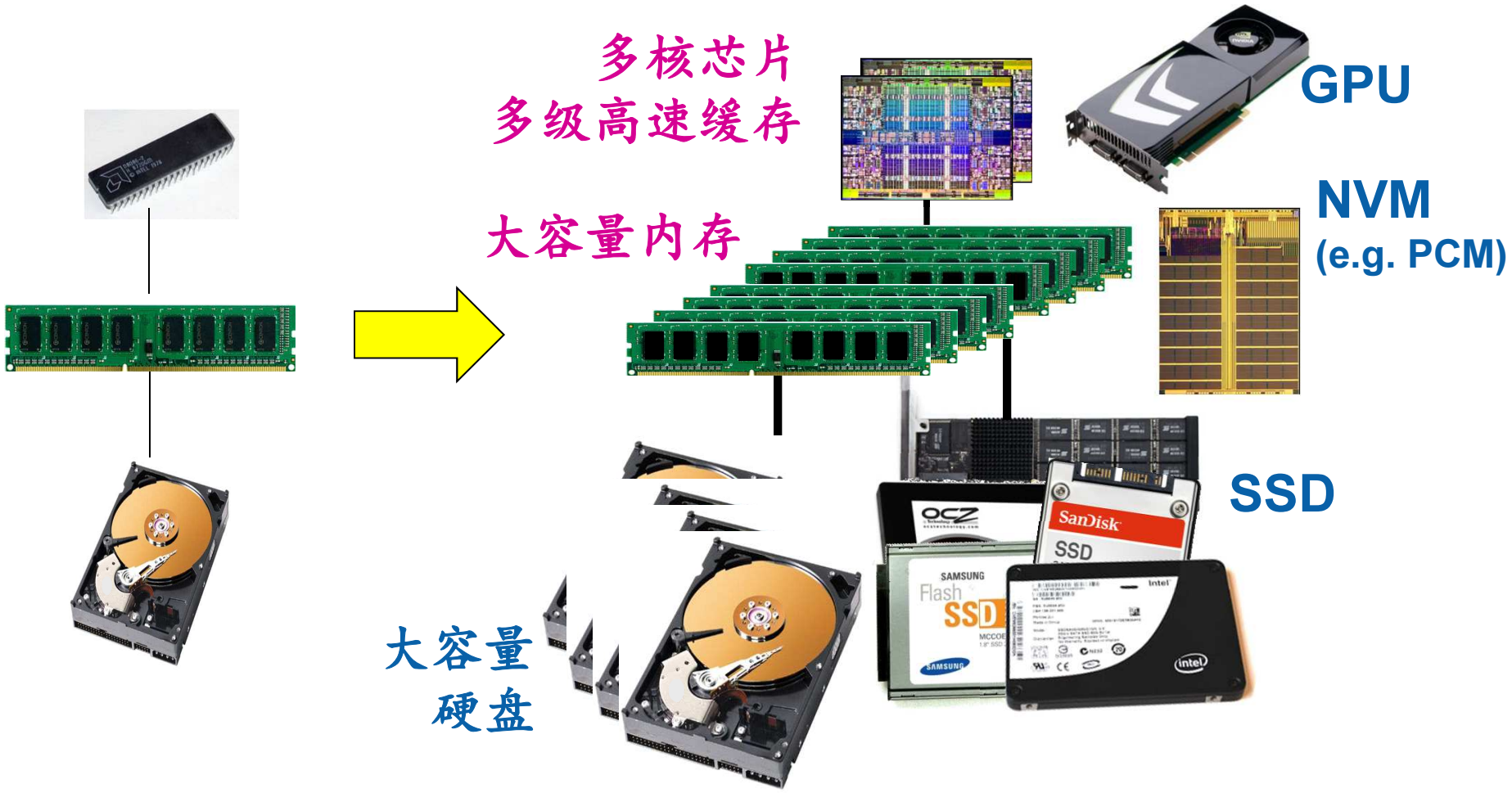
Hive ICDE'10 talk

- Types of Applications:
  - Reporting
    - Eg: Daily/Weekly aggregations of impression/click counts
    - Measures of user engagement
    - Microstrategy dashboards
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country
  - Machine Learning (Assembling training data)
    - Ad Optimization
    - Eg: User Engagement as a function of user attributes
  - Many others

Outline

- MapReduce + SQL系统
- 内存计算
  - 内存数据库
    - 起源发展
    - 关键技术
    - MonetDB
    - 在商用数据库中的实现
  - 内存键值系统
  - 内存MapReduce

体系结构和硬件技术的巨大发展



内存处理

- 随着内存容量的指数级增加
  - 越来越大的数据集可以完全存放在内存中
  - 或者完全存放在一个机群的总和的内存中
  - 例如，每台服务器64GB，一组刀片16台，就是1TB
  - 1TB对于很多重要的热点的数据可能已经足够了

- 内存处理的优点
  - 去除了硬盘读写的开销
  - 于是提高了处理速度

关系型内存数据库

- Main memory database system
- Memory-resident database system
- 上述两个名词有区别
  - Memory-resident：可能是在buffer pool中
  - MMDB: 可能彻底不用buffer pool，改变了系统内部设计

关系型内存数据库

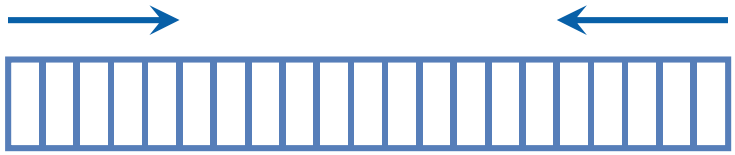
- 最早的提法出现在1980末，1990初
- 第一代MMDB出现于1990初
  - 没有高速缓存的概念
  - 例如：TimesTen
- 第二代MMDB出现于1990末，2000初
  - 对于新的硬件进行优化
  - 主要是学术领域提出的
  - 例如：MonetDB
  - 这一时期，产业界也开始重视MMDB，但主要是用来作前端的关系型cache
- 近年来，主流数据库公司纷纷投入研发MMDB
  - IBM Blink, Microsoft Hekaton & Apollo, SAP HANA, IBM BLU, 等

主要挑战：内存墙问题

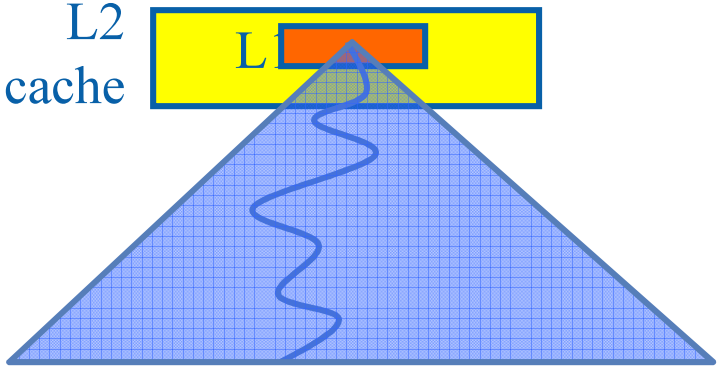
- 内存访问需要100~1000 cycles
- 思路1：减少cache miss
  - 调整数据结构或算法
- 思路2：降低cache miss对性能的影响
  - Software prefetch 预取指令
  - 并行的K个内存访问时，总时间<< k*单个内存访问时间

Sorting

- In-memory sorting / generating runs
- AlphaSort [NBC94]



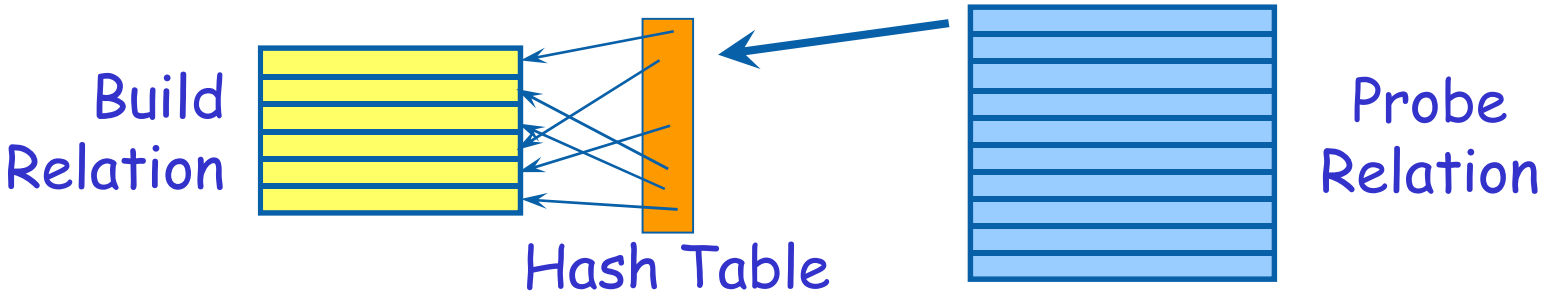
Quick Sort



Replacement-selection

- 使用 quick sort 而不是replacement selection
  - 顺序访问 vs. 随机访问
  - 当快排缩小到cache size以下时，就没有cache miss

Hash Join



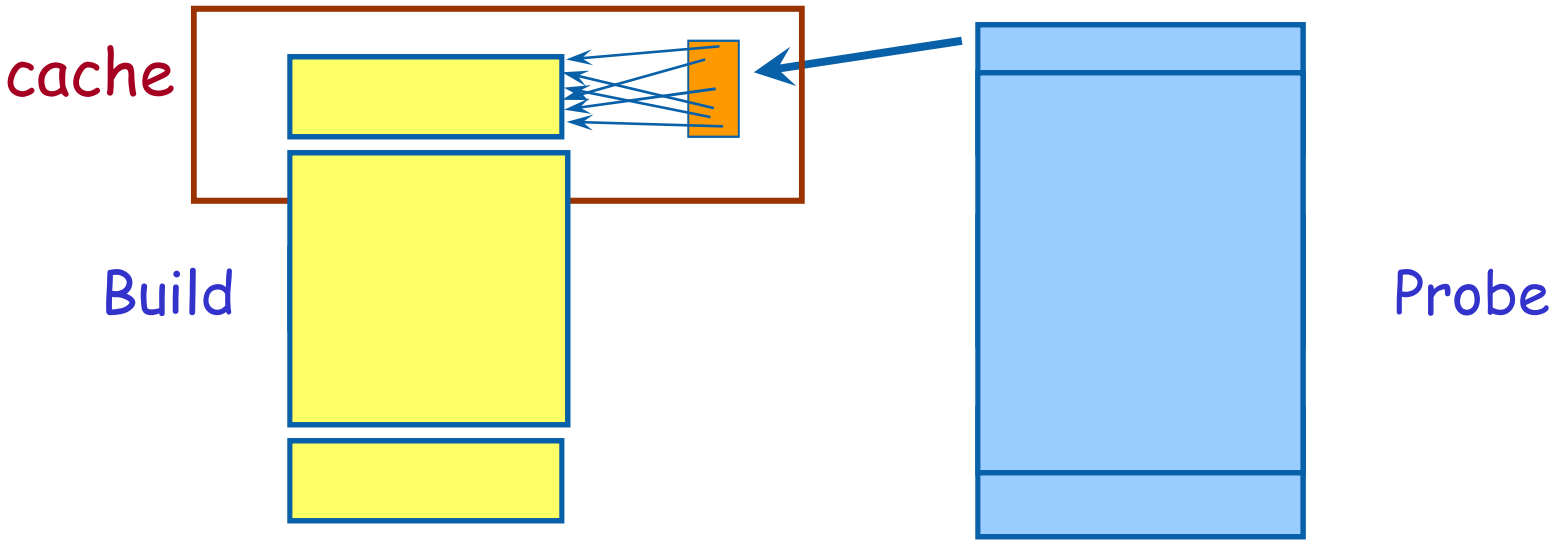
- 随机访问hash table
- 非常差的cache performance
  - ≥ 73% of user time is CPU cache stalls [CAG04]

☞ 解决方案

- Cache partitioning
- Prefetching

Cache Partitioning [SKN94]

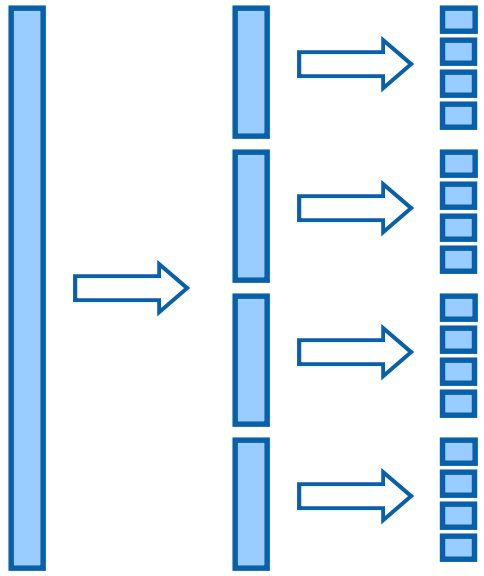
- 类似于I/O partitioning
  - 划分为cache大小的 partitions
  - 从而避免了很多cache miss



Radix Join: Reducing Partition Cost

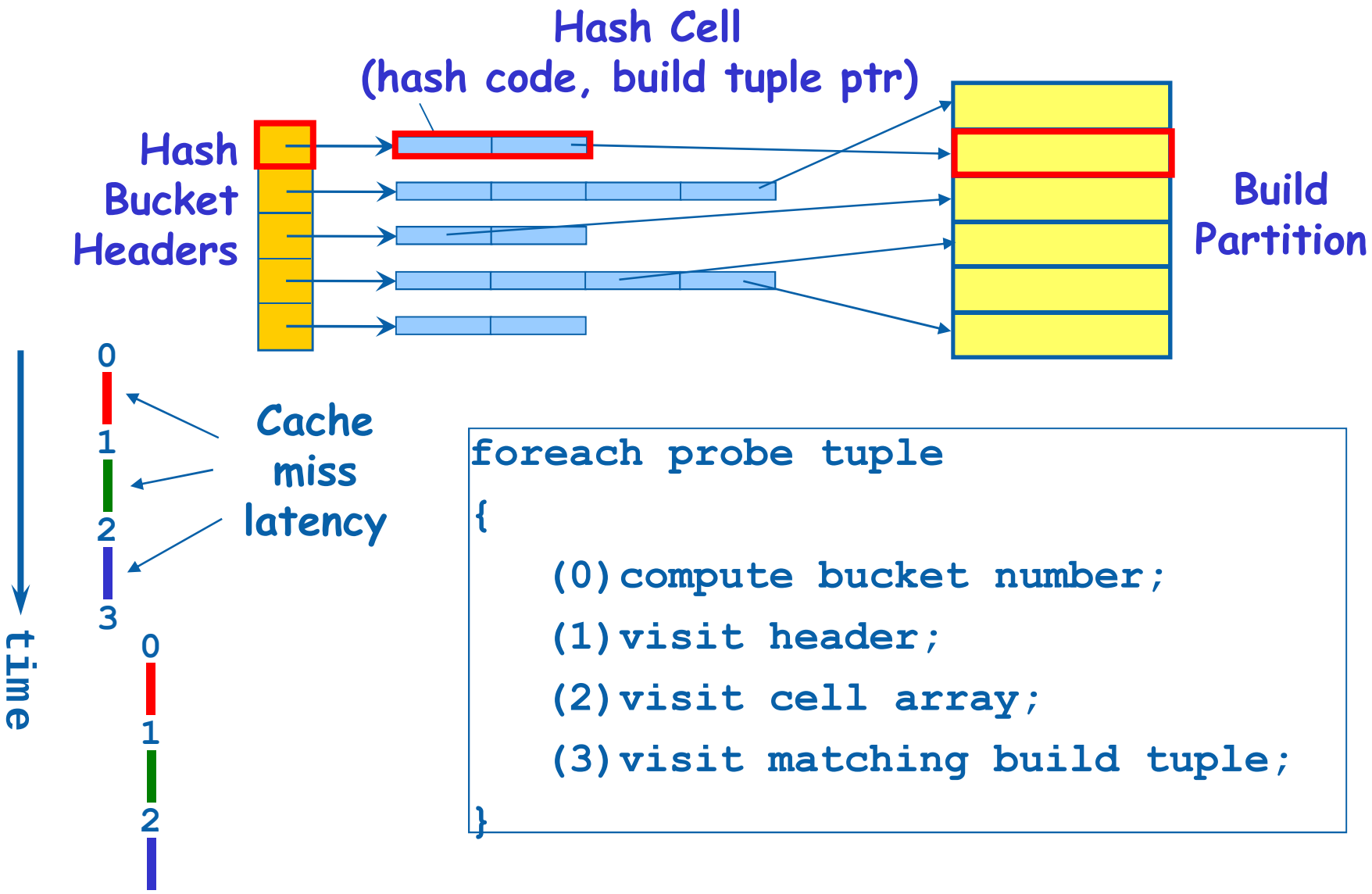
- TLB thrashing if # partitions > # TLB entries
  - Cache thrashing if # partitions > # cache lines in cache

- 解决方案: multiple passes
  - # partitions per pass < TLB size
  - Radix-cluster [BMK99,MBK00]



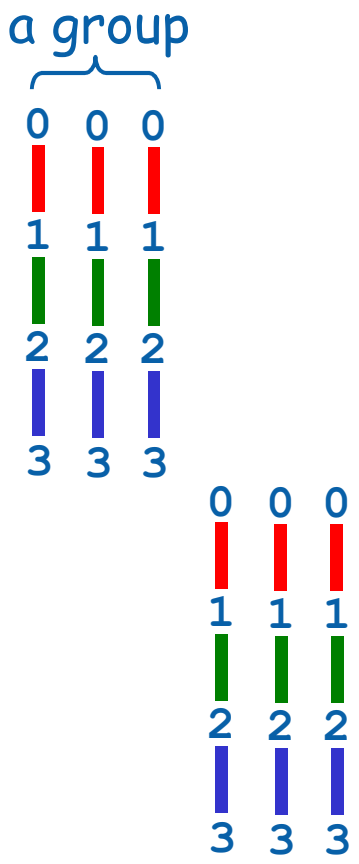
2-pass partition

Simplified Probing Algorithm




```
foreach probe tuple
{
    (0)compute bucket number;
    (1)visit header;
    (2)visit cell array;
    (3)visit matching build tuple;
}
```

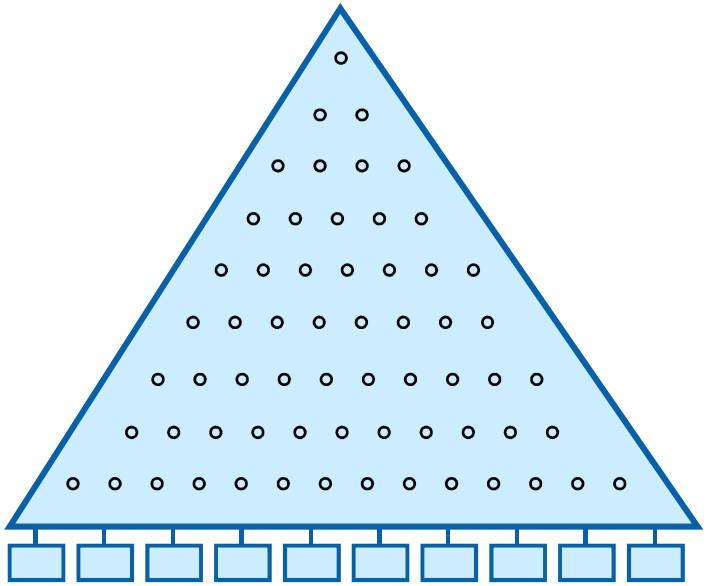
Group Prefetching [CGM04]




```
foreach group of probe tuples {
    foreach tuple in group {
      (0)compute bucket number;
         prefetch header;
    }
    foreach tuple in group {
      (1)visit header;
         prefetch cell array;
    }
    foreach tuple in group {
      (2)visit cell array;
         prefetch build tuple;
    }
    foreach tuple in group {
      (3)visit matching build tuple;
    }
}
```

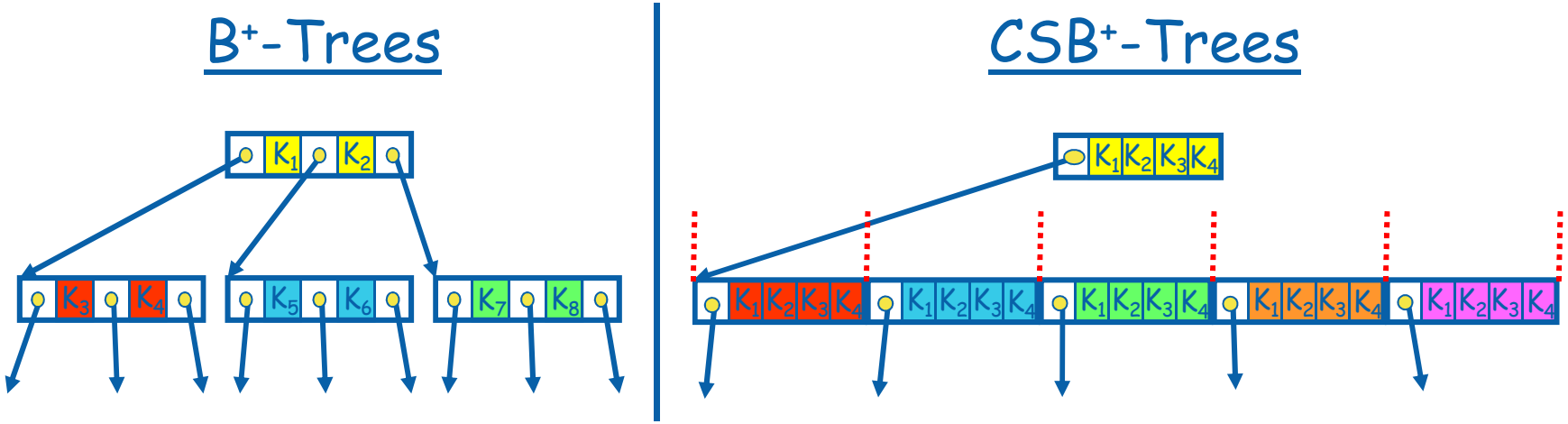
B+-Trees

- Main memory B+-Trees
  - 比第一代内存数据库中的T-Trees更好 [RR99]
- Node width = cache line size



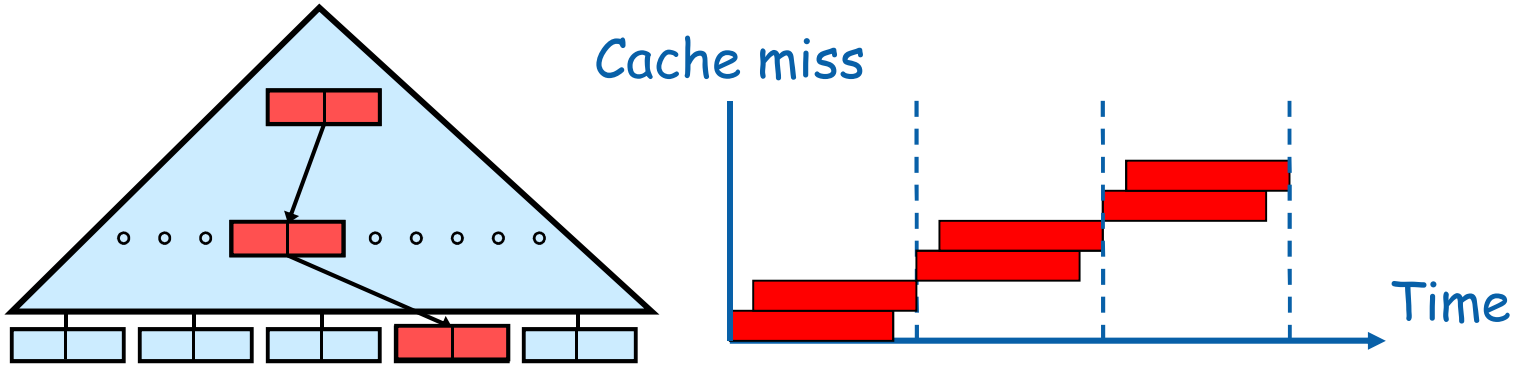
Reducing Pointers for Larger Fanout

- Cache Sensitive B$^+$-Trees (CSB$^+$-Trees) [RR00]
- Layout child nodes contiguously
- Eliminate all but one child pointers
  - *Double fanout of nonleaf node*

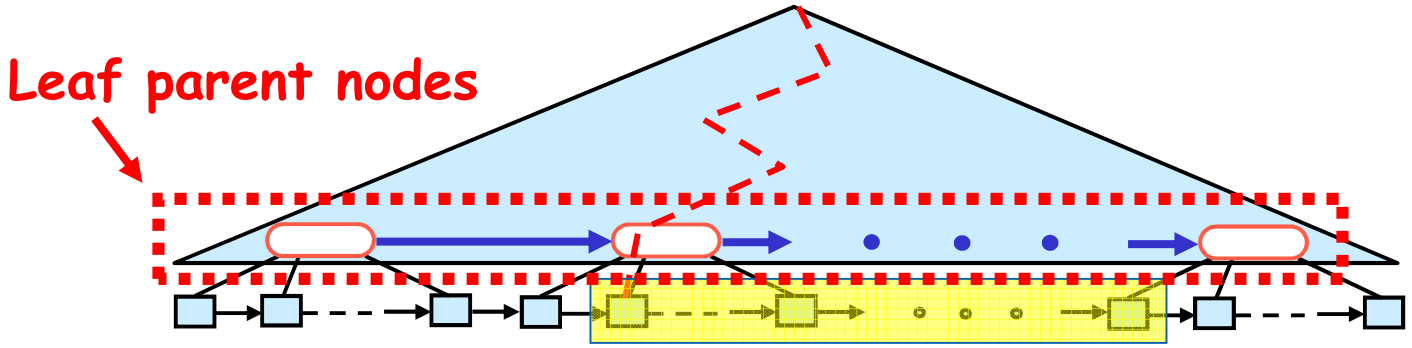


Prefetching for Larger Nodes

- Prefetching B$^+$-Trees [CGM01]
- Node size = multiple cache lines (e.g. 8 lines)



Prefetching for Range Scan Performance

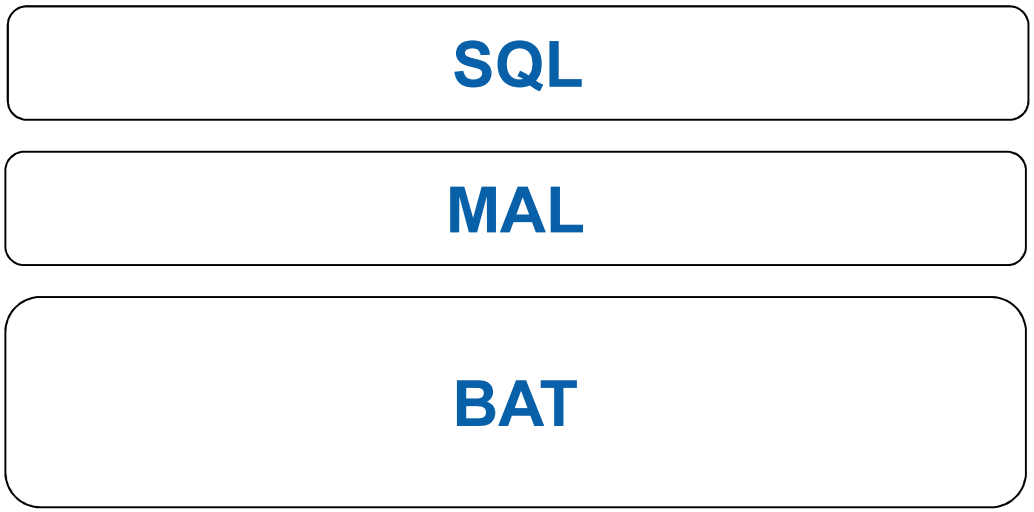


其它关键技术

- Vectorization
  - 每个Operator不是一次调用next()仅返回一条记录
  - 而是返回一组记录，提高代码利用率，形成紧凑循环
- 处理器加速
  - SIMD
  - GPU
  - Multi-core
- 压缩
  - 简单的压缩
    - 例如，根据一个列的取值范围为[0,2^k-1]，采用K bit表示
  - 字典压缩
    - 把数据映射为整数

MonetDB

- MonetDB
  - 荷兰CWI研究所研制
  - 1999年数据库领域第一篇关于cache performance的文章
  - 成熟的内存数据仓库,支持几乎所有的SQL
  - 内存列式存储,数据在类似数组的结构中

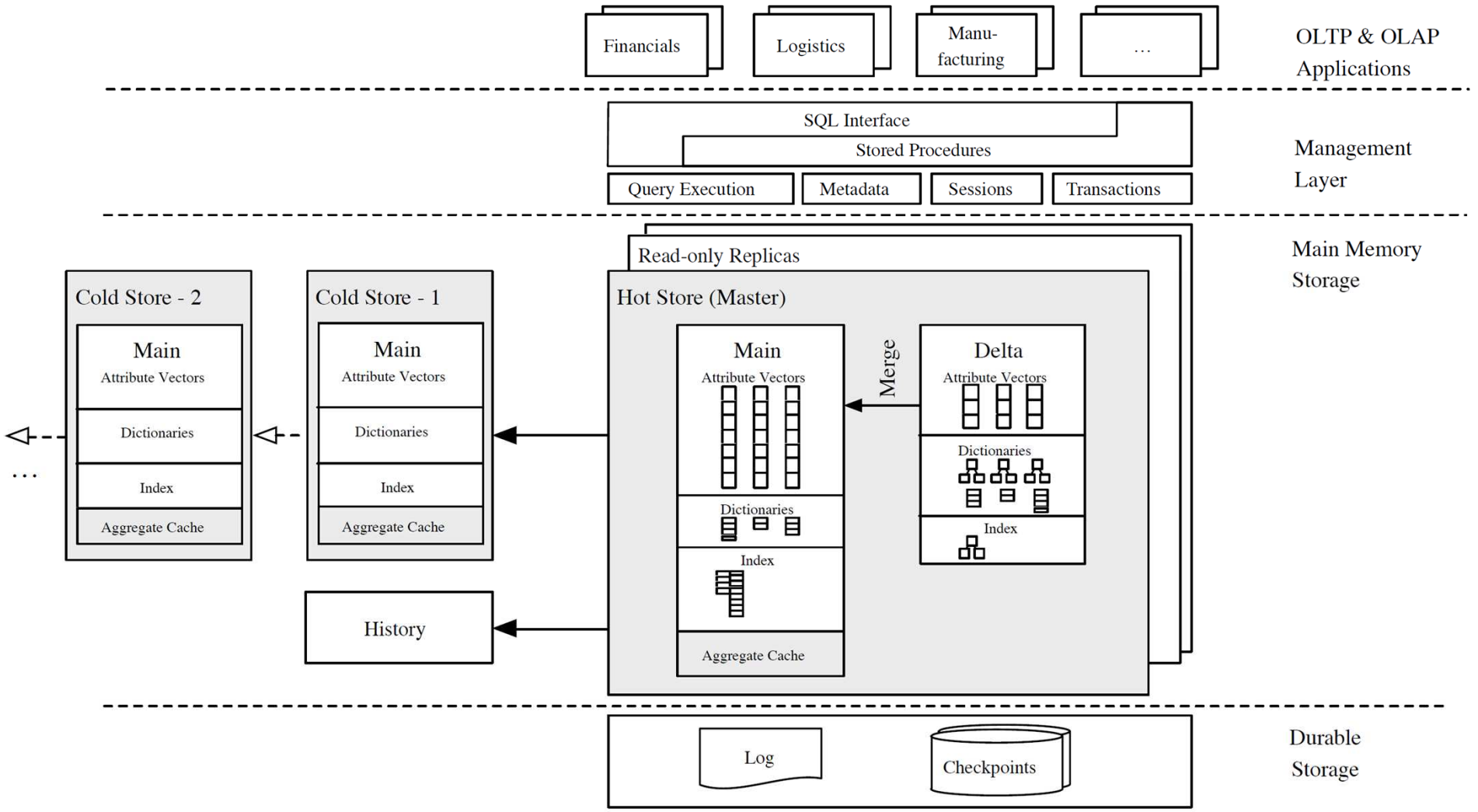


SQL被翻译为中间语言MAL的运算

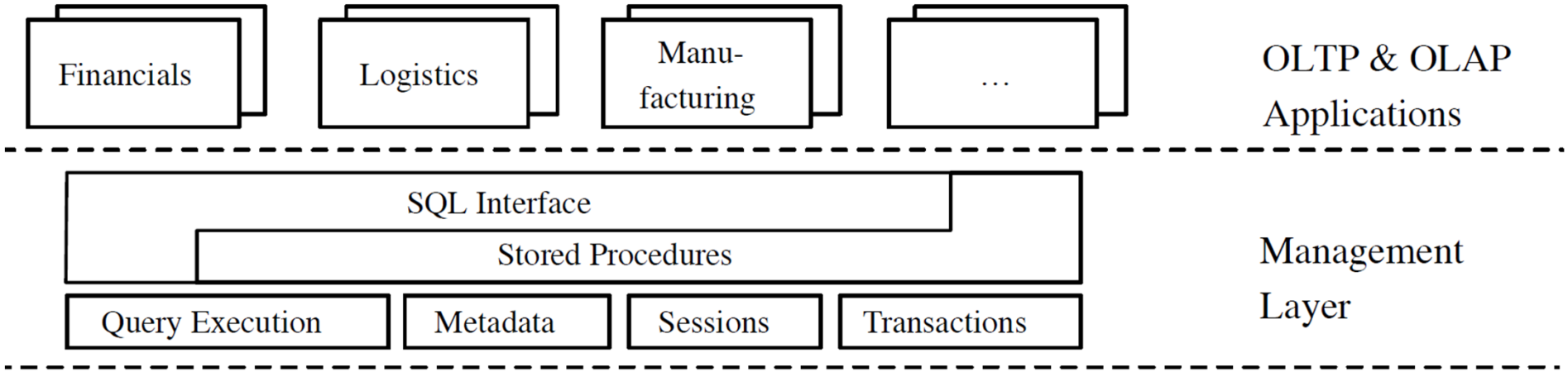
每个MAL运算，实现对整个BAT数据的运算

BAT (Binary Association Table)
相当于一个数组，存储一列数据，数据mmap到内存

SAP HANA

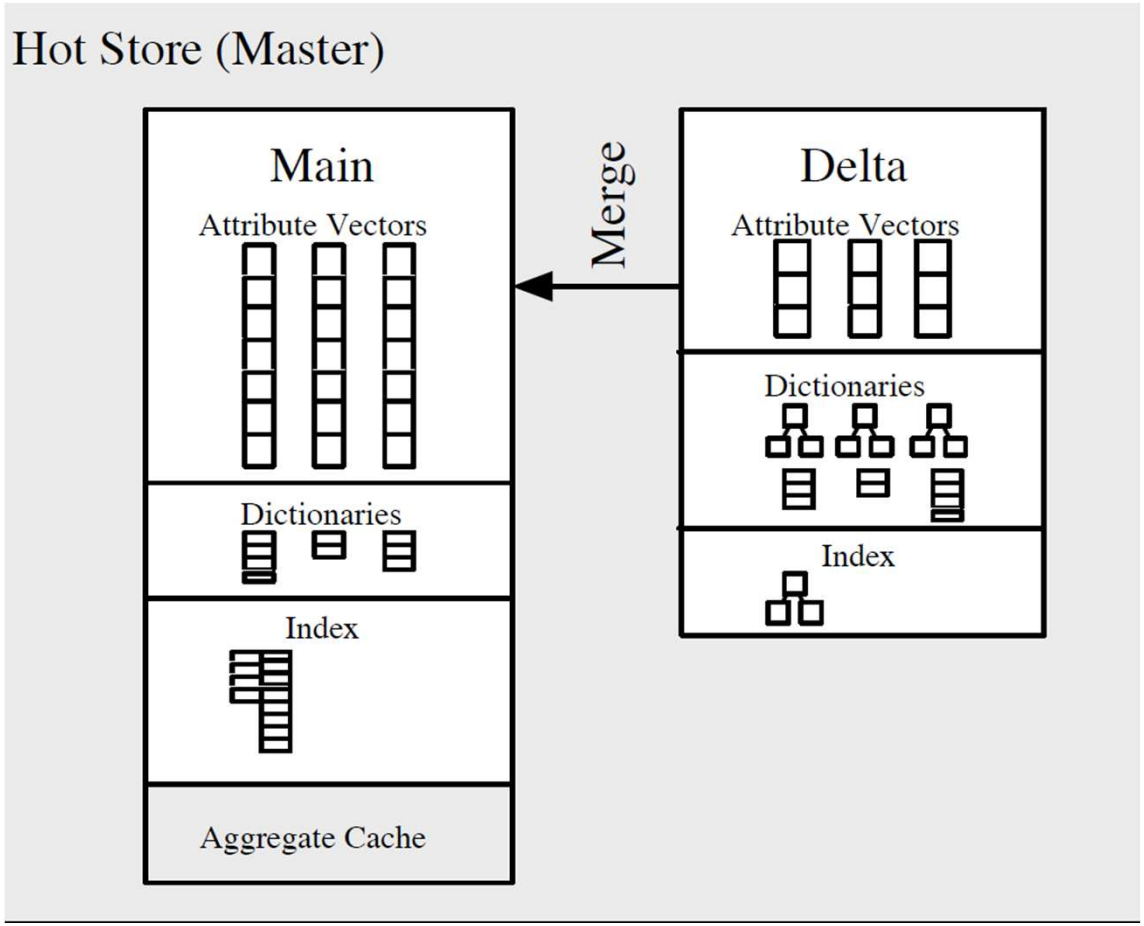

Financials
Logistics
Manu-facturing
…
OLTP & OLAP Applications
SQL Interface
Stored Procedures
Query Execution
Metadata
Sessions
Transactions
Management Layer
Read-only Replicas
Hot Store (Master)
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
Merge
Delta
Attribute Vectors
Dictionaries
Index
Cold Store - 2
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
Cold Store - 1
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
History
Main Memory Storage
Log
Checkpoints
Durable Storage

SAP HANA



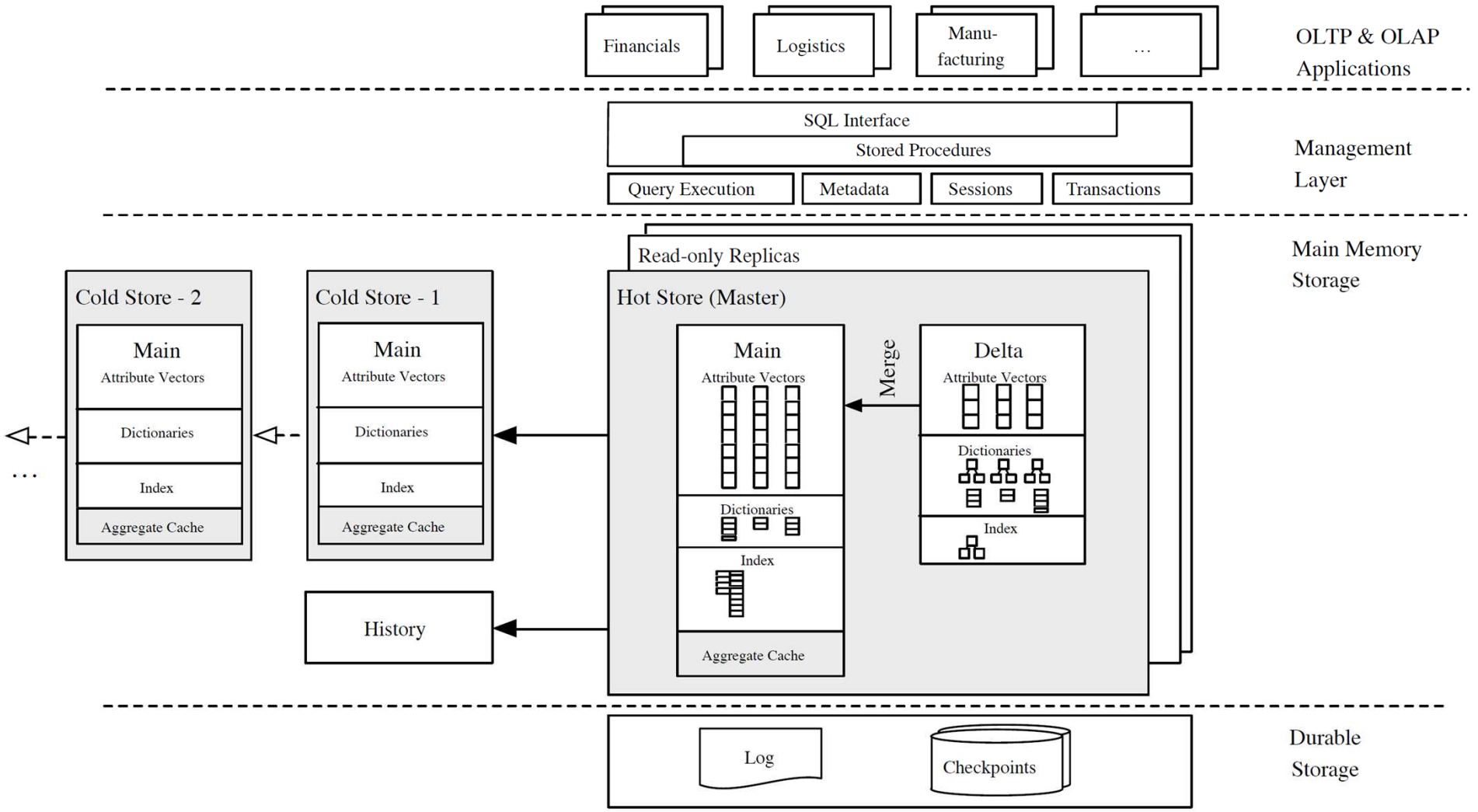
- 前端与普通数据库很相似

SAP HANA



- Main data
- Delta data
  - 新的数据
  - Updates
- Dictionary 压缩
- 轻量index

SAP HANA


Financials
Logistics
Manu-facturing
…
OLTP & OLAP Applications
SQL Interface
Stored Procedures
Query Execution
Metadata
Sessions
Transactions
Management Layer
Read-only Replicas
Hot Store (Master)
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
Merge
Delta
Attribute Vectors
Dictionaries
Index
Cold Store - 2
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
Cold Store - 1
Main
Attribute Vectors
Dictionaries
Index
Aggregate Cache
History
Main Memory Storage
Log
Checkpoints
Durable Storage

其它主流数据库产品

- Microsoft
  - Hekaton：OLTP
  - Apollo: OLAP
- IBM
  - Blink：IBM的数据仓库加速系统
  - BLU：列式数据库engine, 有硬盘存储，使用了许多内存处理技术
- Oracle
  - In-memory data analytics caching

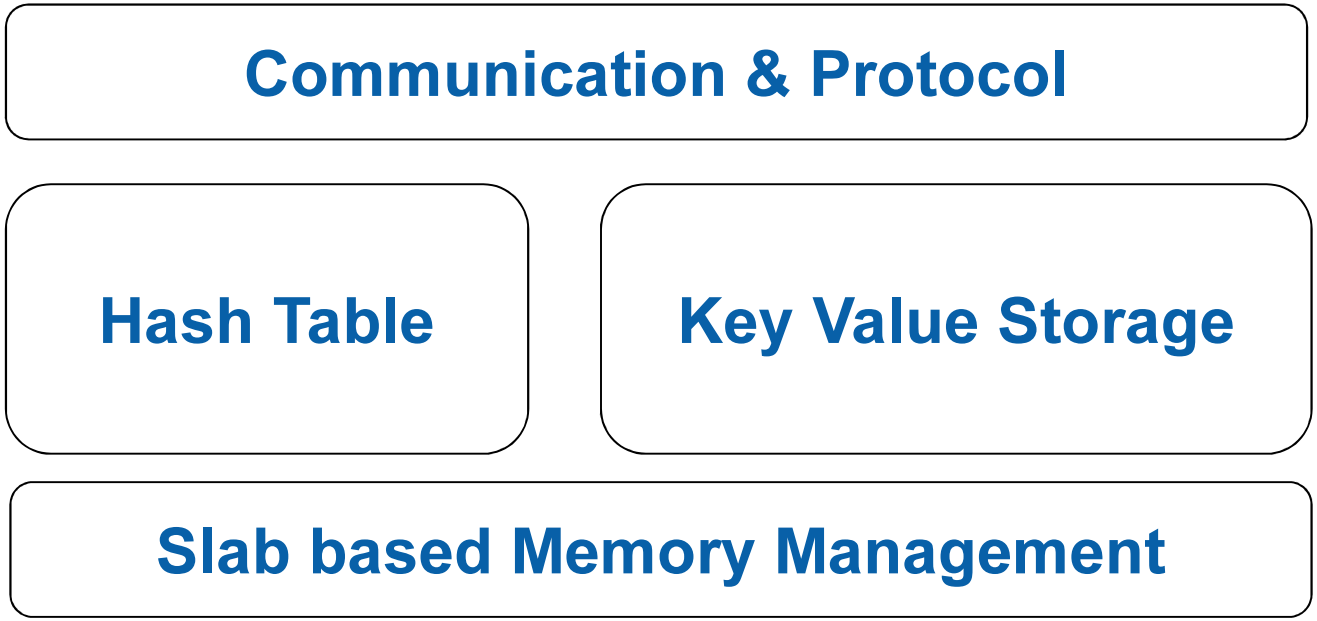
Outline

- MapReduce + SQL系统
- 内存计算
  - 内存数据库
  - 内存键值系统
    - Memcached
    - Facebook scaling memcached
  - 内存MapReduce

内存key-value系统

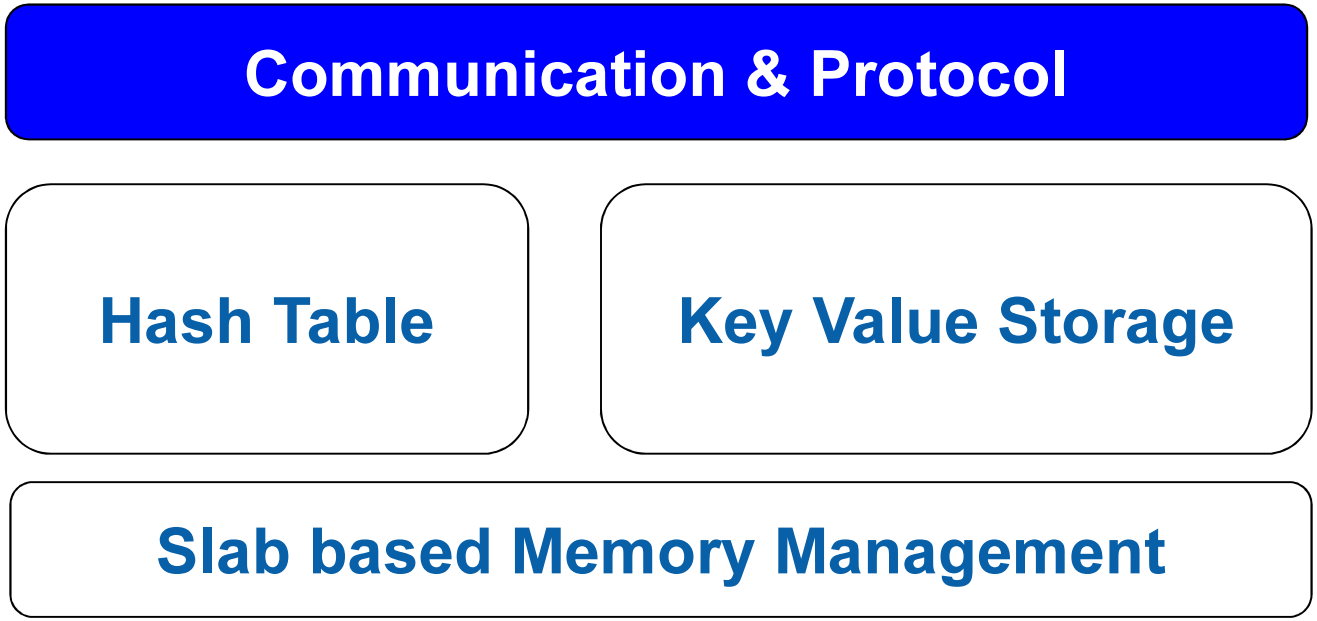
- Memcached
  - 用户： Facebook, twitter, flickr, youtube, …
  - 单机的内存的key-value store
  - 数据在内存中以hash table的形式存储
  - 支持最基础的<key, value>数据模型
  - 通常被用于前端的cache
  - 可以使用多个memcached+sharding建立一个分布式系统
- Redis： 与memcached相比
  - 提供更加丰富的类型
  - key可以包含hashes, lists, sets 和sorted sets
  - 支持副本和集群

Memcached内部结构



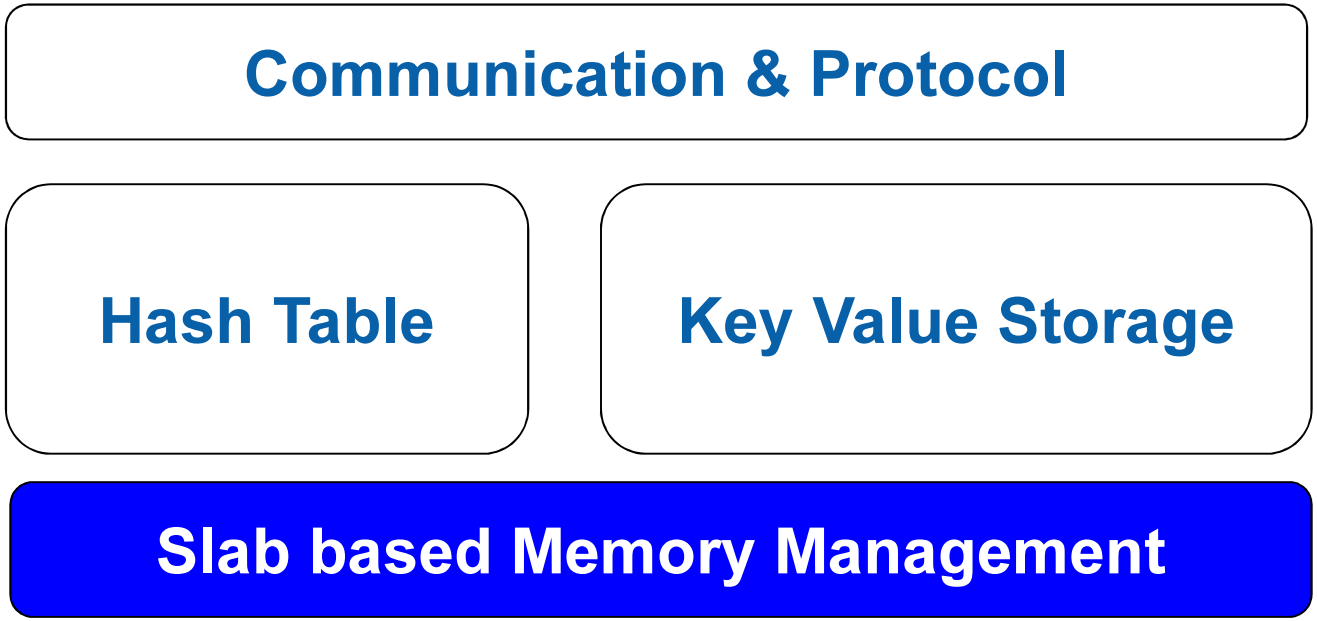
- 单机系统

Memcached内部结构



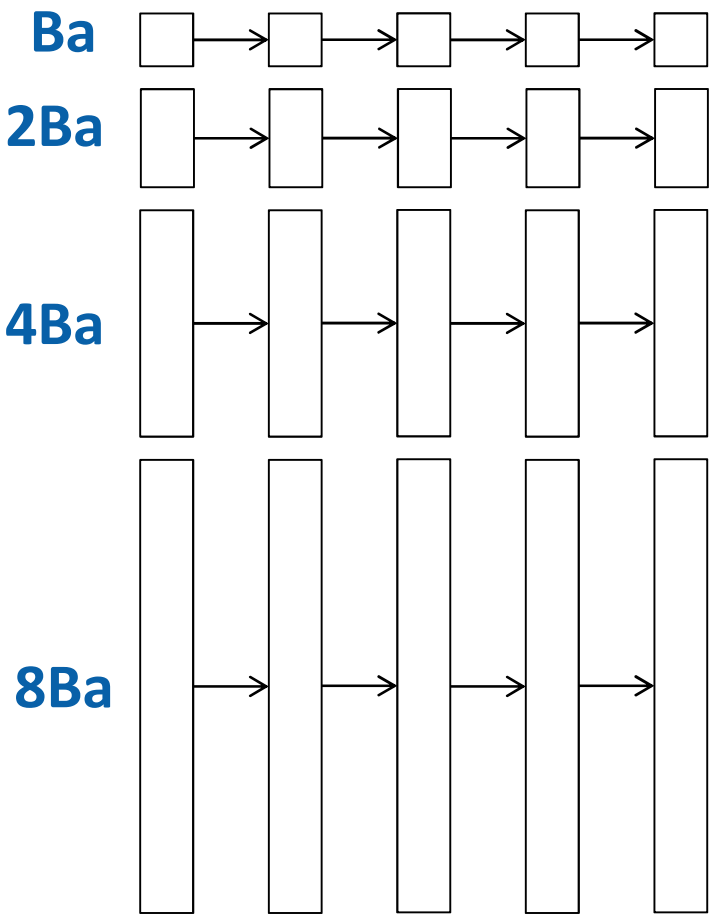
- 实现自定义的协议，支持GET/PUT等请求和响应
  - 文本方式的协议
  - 二进制协议

Memcached内部结构



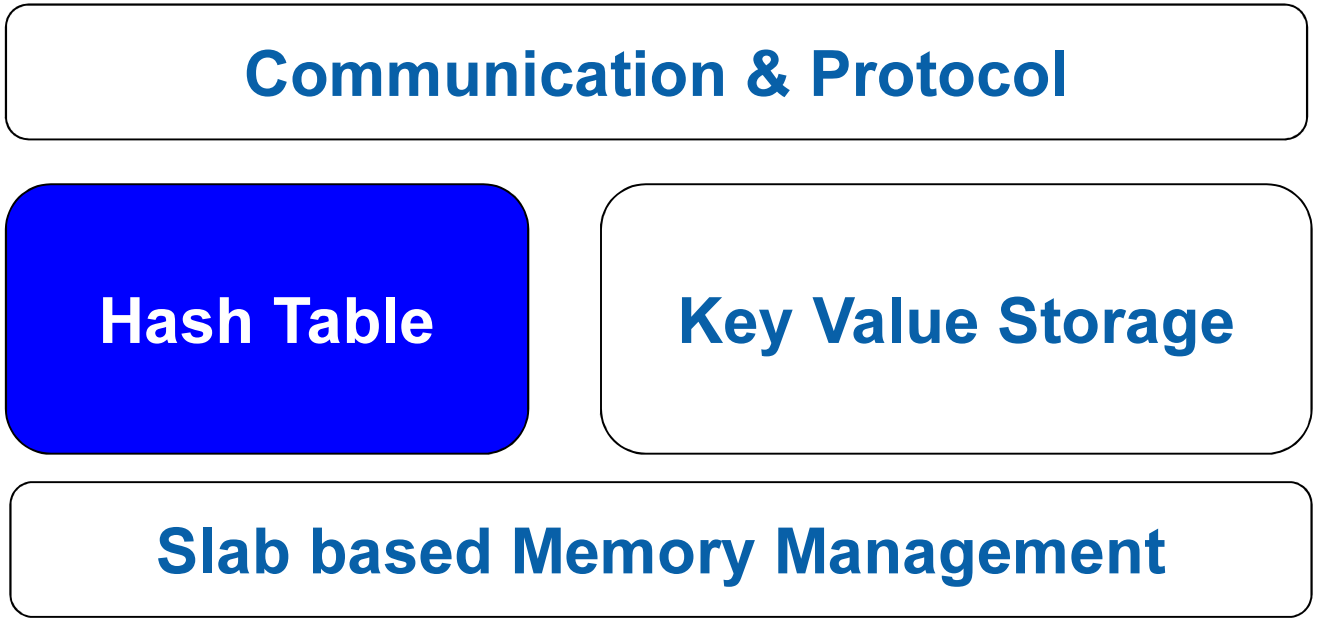
- 采用多个链表管理内存
  - 每个链表中的内存块大小相同
  - 第k个链表的内存块大小是2^kBase字节
  - 只分配和释放整个内存块

Slab-Based Memory Management



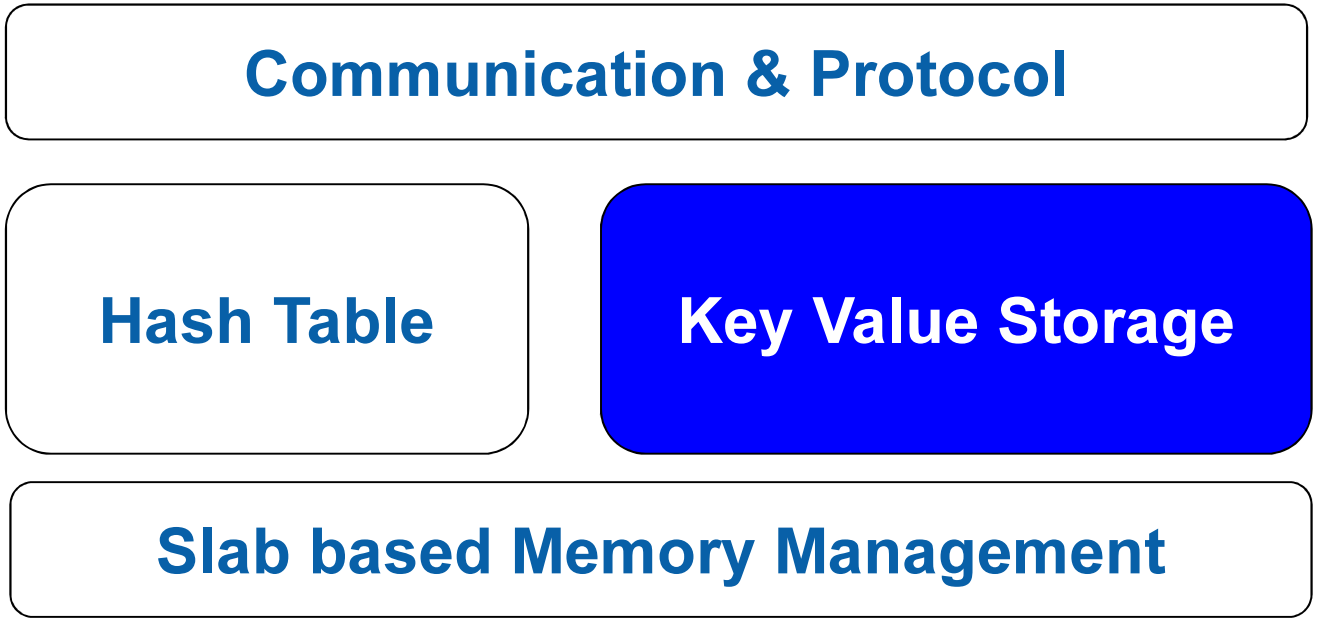
Ba(se)是最小的块的大小

Memcached内部结构



- Key-Value采用一个全局的Hash Table进行索引
- 多线程并发互斥访问

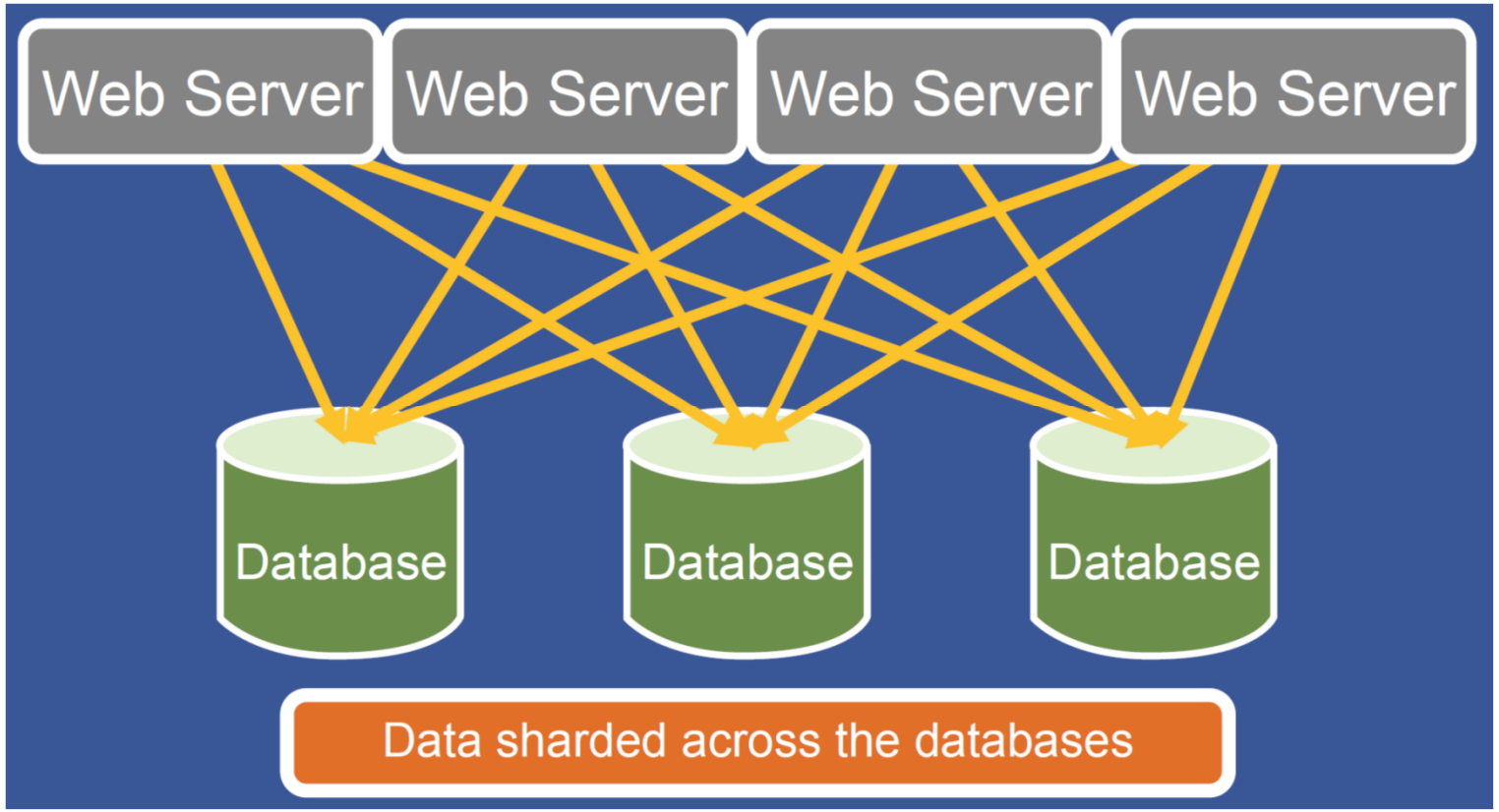
Memcached内部结构



- 每个Key-Value存储在一个内存块中
  - Hash link：chained hash table
  - LRU link: 相同大小的已分配内存块在一个LRU链表上
- 内存不够时，可以丢弃LRU项

Facebook

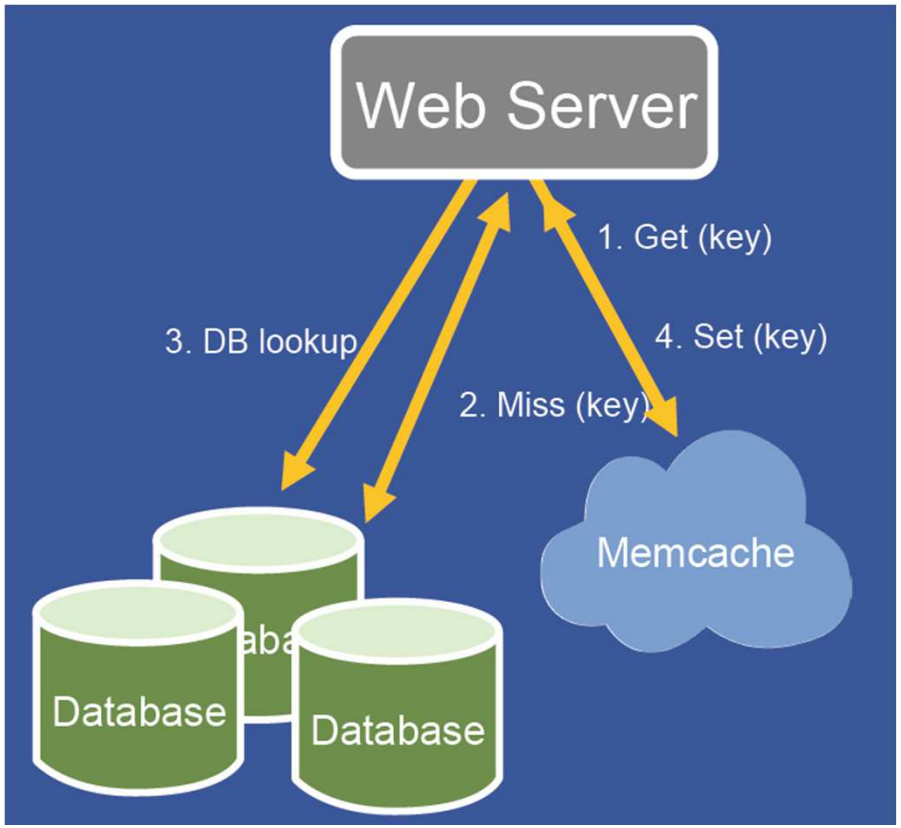
- “Scaling Memcache at Facebook”. NSDI’13
- 刚开始：直接用MySQL就足够了



图来源：NSDI’13 slides

数十台服务器，每秒上百万操作请求

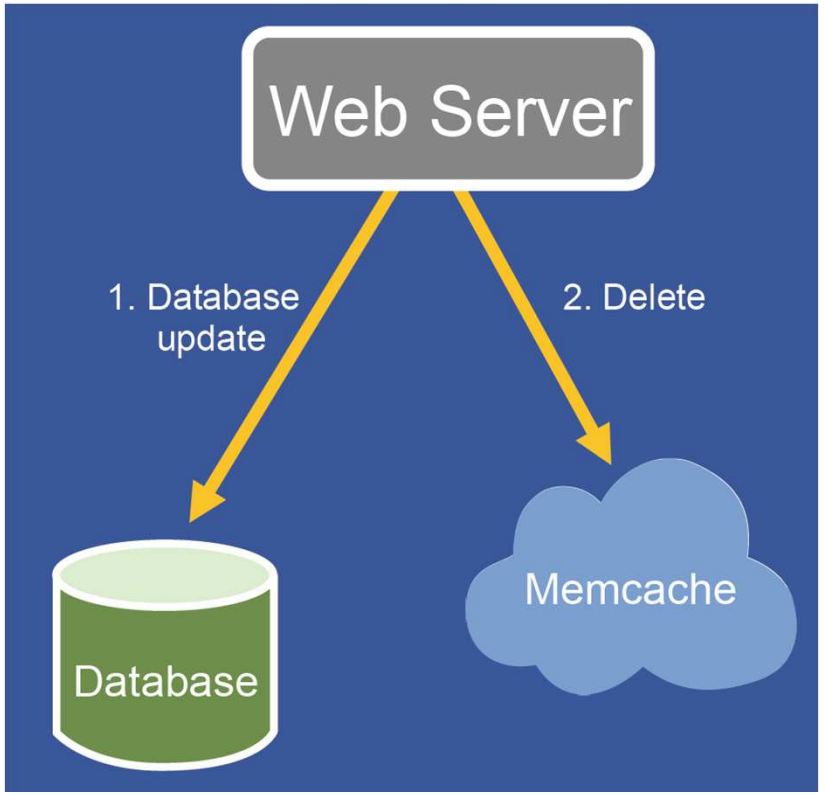
- 增加了几台Memcached服务器
- 读比写多2个数量级
- Look-aside cache
  - 多数hit在memcached
  - 如果miss，再读DB，然后放入memcached



图来源：NSDI'13 slides

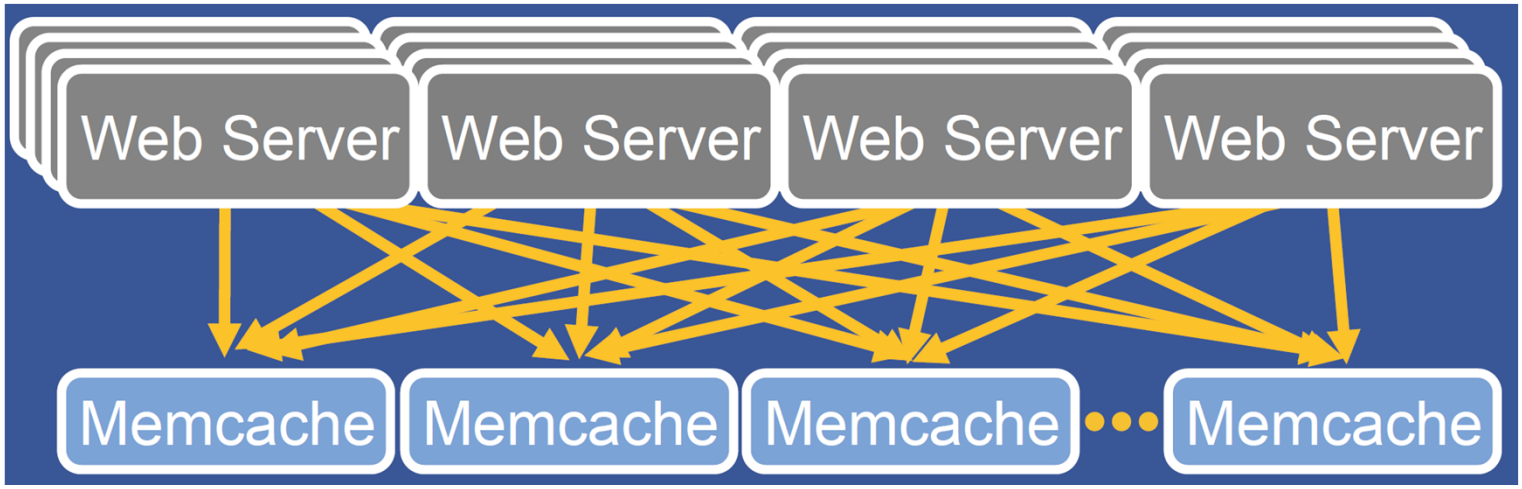
数十台服务器，每秒上百万操作请求

- 增加了几台Memcached服务器
  - 应用进行sharding
- 读比写多2个数量级
- Look-aside cache
  - 在从DB中删除后
  - 也从memcached删除
- 问题
  - 分布式读写，DB与memcached不一致
  - 可能读到稍旧的数据



图来源：NSDI'13 slides

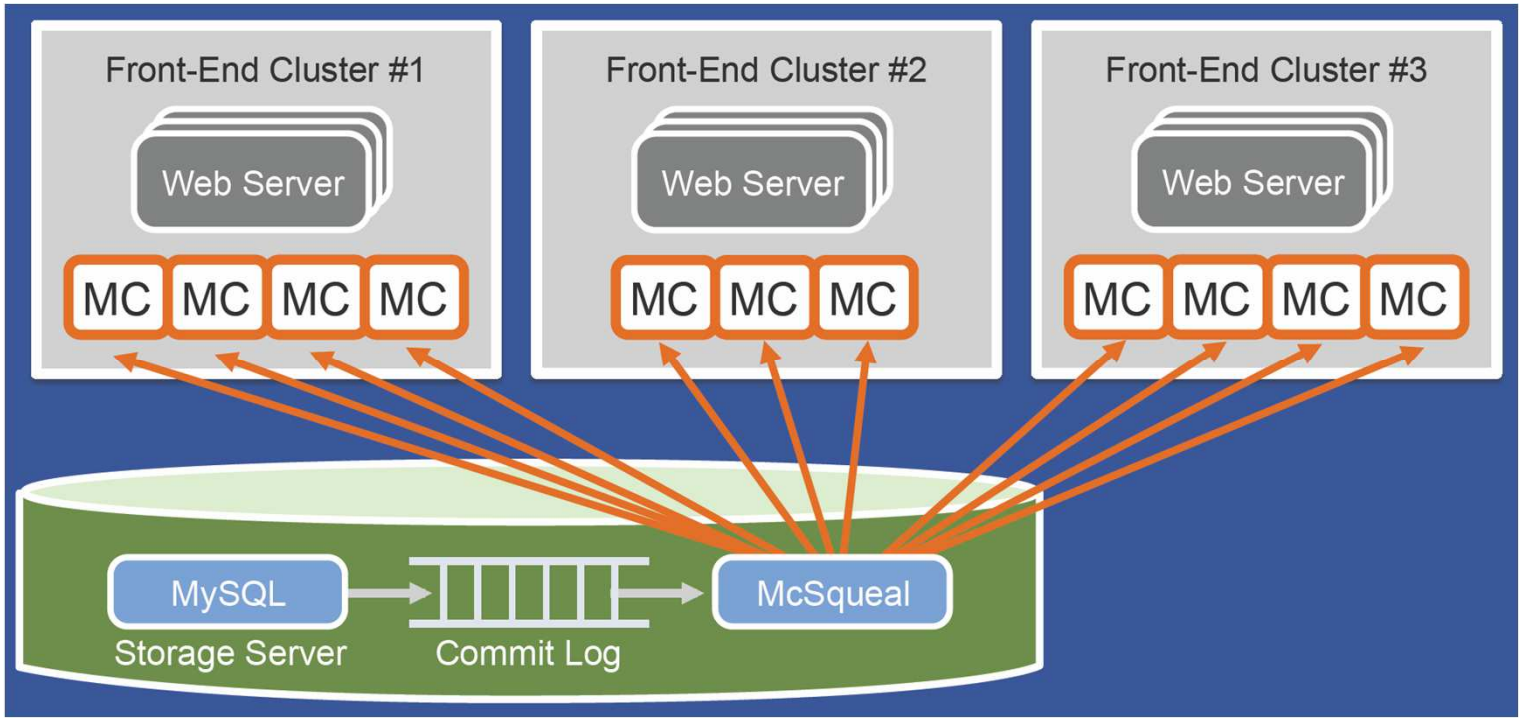
数百台服务器，每秒上千万操作请求



- 多个Memcached组成一个cluster
- 采用consistent hashing管理
- 每个web server为了满足一个网页，可能同时发出上百个请求

图来源：NSDI'13 slides

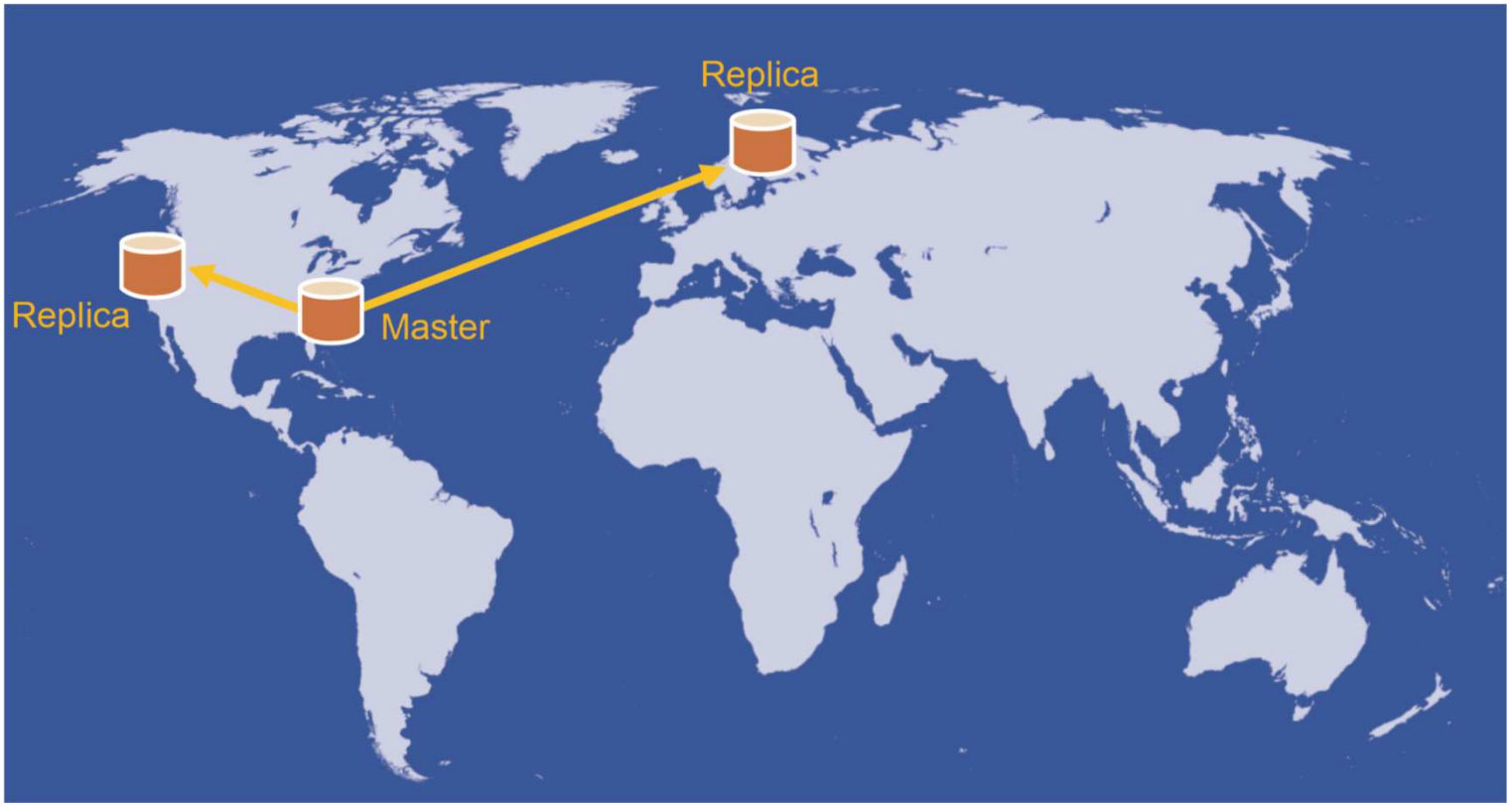
数千台服务器，每秒上亿次操作请求



- 多个Front-end cluster
  - 每个Front-end cluster由Web servers, memcached cluster组成
- 同一组DB服务器
  - 使用commit log来把数据更新发到memcached

图来源：NSDI'13 slides

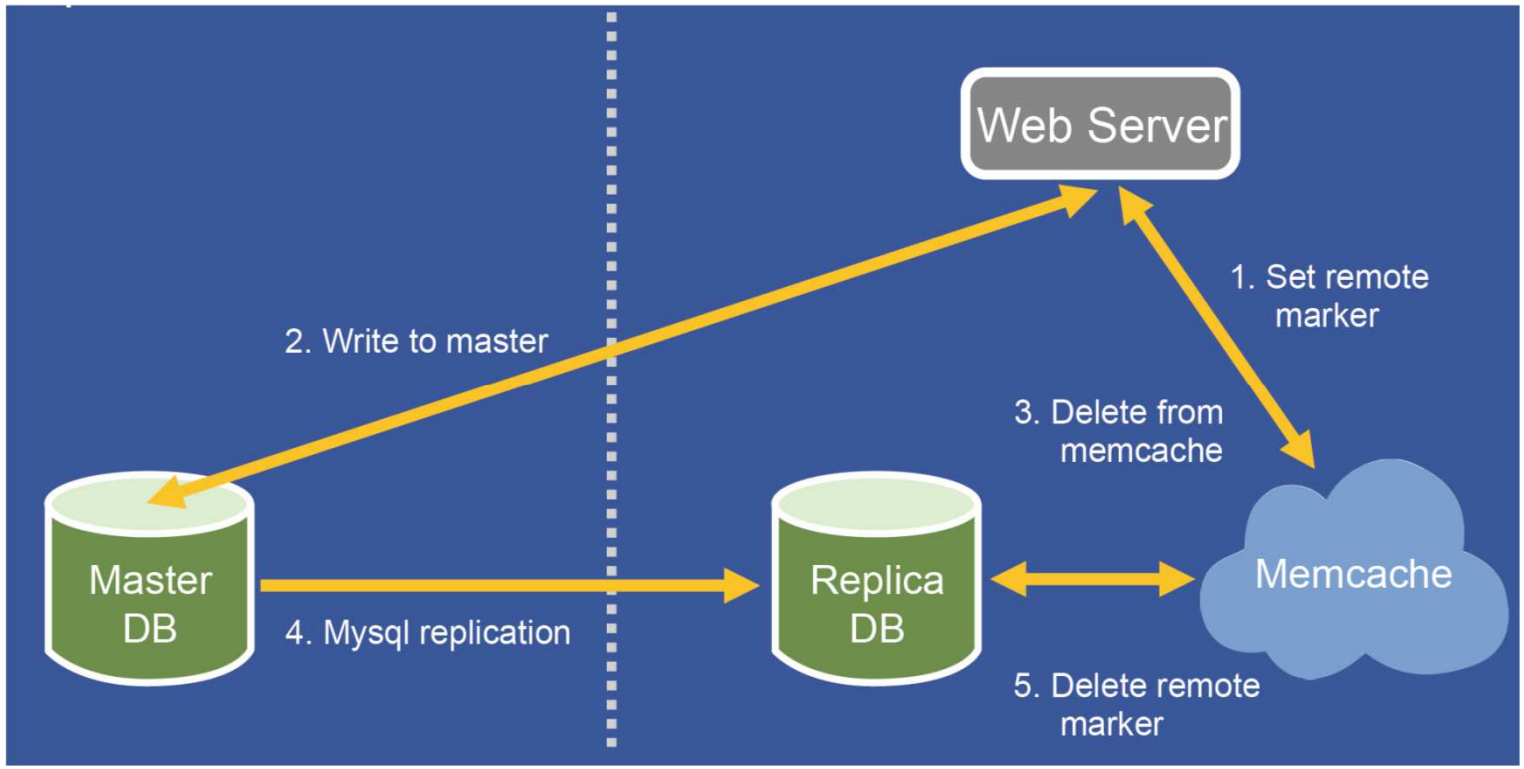
数千台服务器，每秒数十亿次操作请求



- 多个Replica
  - 主DB与Replica DB不一致？

图来源：NSDI'13 slides

数千台服务器，每秒数十亿次操作请求



- 在DB write之前，先在memcached中写一个marker
- 这样读时发现marker，就一定要从主DB中取

图来源：NSDI'13 slides

小结

- MapReduce + SQL系统
- 内存数据库
  - 起源发展
  - 关键技术
  - MonetDB
  - 在商用数据库中的实现
- 内存键值系统
  - Memcached
  - Facebook scaling memcached