# 分布式海量数据库的探索:Wasp


**代志远**

**www.taobao.com**

# 提纲

- HBase在阿里的现状
- 客户的需求
- Wasp的前世今生
- Wasp的架构
- RoadMap
- Q&A

# HBase在阿里的现状

- 概要：
  - HBase的优缺点。
  - 我们的规模、发展以及未来。

# HBase在阿里的现状

- HBase的优缺点
  - 优点：强一致性，高拓展性，写入速度快（ LSM-Tree ），混合行列存储，表结构变更容易，稀松矩阵利于节省存储空间。
  - 缺点：没有跨行事务，没有索引不利于简单的多维分析，API比较复杂学习成本高并且线上关系型数据向HBase迁移工作量比较大。

# HBase在阿里的现状

- 我们的规模、发展以及未来
  - 700-800台的集群规模，离线集群如果上线HBase总规模将会更大。
  - HBase社区活跃，结构完善，前景良好，影响力越来越大，在集团内大有跟现有关系型数据库一较高下的趋势。
  - 大数据时代来临，常年运营积累的数据越来越多，HBase良好的拓展性成为了解决此类问题的利器，并且越来越多的应用会迁移到HBase。
  - 但是我们有如下的客户需求

# 客户的需求

- 概要
  - 多维查询（二级索引）
  - 索引与实体的一致性
  - 简单的用户入口
  - 较强的线性拓展能力

# 客户的需求

- 多维查询（二级索引）
  - 用户需要根据根据条件进行筛选和过滤。
    - 暴力Scan？ Filter？协处理器？
      - 需要根据需求建立索引表

# 客户的需求

- 索引与实体的一致性
  - 实体与索引分别是不同的物理行
  - 需要同时更新索引与实体（增加、删除、修订）
  - 两者之间如何保证一致性？
  - 是否每个业务都需要保证强一致性？
  - 是否每次都要读到最新的一次更新？
  - 针对以上问题我们如何取舍和保证？

# 客户的需求

- 简单的用户入口
  - 由于：繁琐的HBase API，由于对HBase API不熟悉，经常因为使用不当导致线上故障。比如：为何我插入的这一行出现了十几万列？
  - 导致：用户学习成本较高，很多用户就喜欢原来的SQL，甚至使用SQL若干年了。

# 客户的需求

- 较强的线性拓展能力
  - 业务方：给传统单机数据库做分库分表太繁琐、代价太高了，还需要业务应用人员自己维护分库信息。
  - DBA：线上数据最近几年增加太快，每隔段时间就要重新拆库分库一次，每次都还有不同程度的故障。

# Wasp的前世今生

- 概要
  - Google的数据库产品。
  - Like-MegaStore架构能解决哪些问题。
  - 选择MegaStore的初衷和原由。

# Wasp的前世今生

- Google的数据库产品
  - 第一代：Bigtable(06年)
  - 第二代：MegaStore(08年)
  - 第三代：Spanner(12年)
  - 第四代：F1(论文尚未发表 )

# Wasp的前世今生

- Bigtable
  - 开启了NoSQL的大门。
  - 开启了“云”的大门。
  - 在Google中虽然有层出不穷的新技术出现，并有人认为他在Google走向过时，但Bigtable在Google中现在是中流砥柱，仍旧有绝大多数应用在使用Bigtable。

# Wasp的前世今生

- MegaStore
  - MegaStore 的初衷：Google也曾经面临了和我们一样的困境。
  - 现在MegaStore的情况：大名鼎鼎的如Gmail、AppEngine、Picasa、Android market...等重要应用都在此技术基础之上构建。

# Wasp的前世今生

- Spanner
  - 结合了MegaStore和Bigtable的一些优点，更多的可以看做Bigtable的升级版，依靠卫星来解决客户端与服务端以及服务器与服务器之间的时间戳一致性。这种实现代价非常高，只有期盼中，有关此类系统的简化版的尝试可能需要在几年之后。

# Wasp的前世今生

- F1
  - 替代Google的关系型数据库应用，刚刚成型，应用还不是很多，PPT刚发布，论文还需要一段时间才能有。但从PPT中可以得到一些应用信息，相似于MegaStore。

# Wasp的前世今生

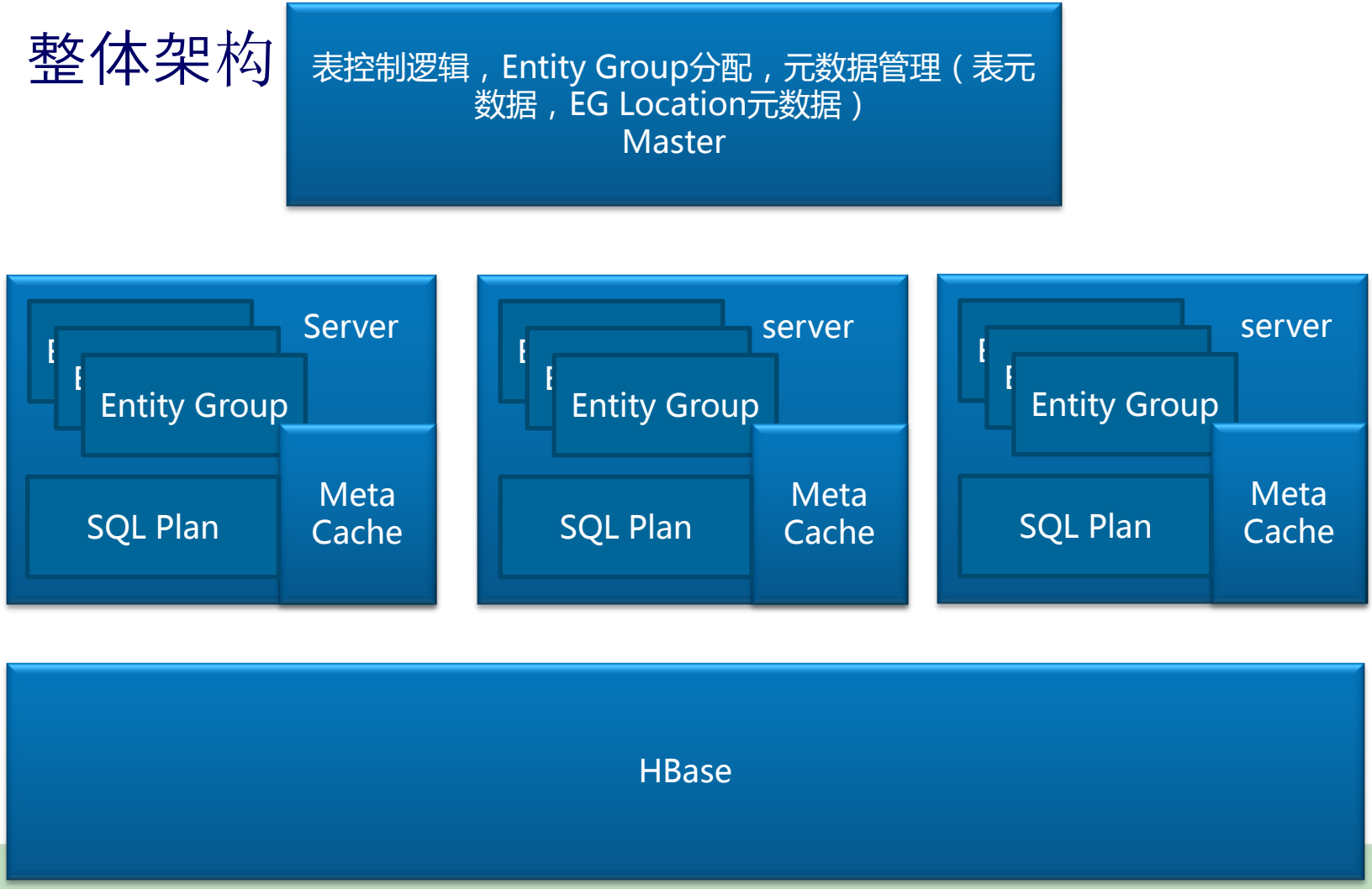
- MegaStore解决的问题
  - 在Bigtable上层构建了一套数据库系统。
  - 解决了实体与索引之间的一致性问题。
  - 提供了简单的SQL语法，支持简单的索引字段查询、主键的Join、主键的GroupBy。
  - 支持良好的线性拓展，不牺牲Bigtable的原有线性拓展能力。
  - 提供跨集群的数据一致性。

# Wasp的架构

- 概要
  - 整体的架构
  - 事务
    - 写事务
      - 本地事务
      - 全局事务
    - 读事务
  - 二级索引
  - SQL

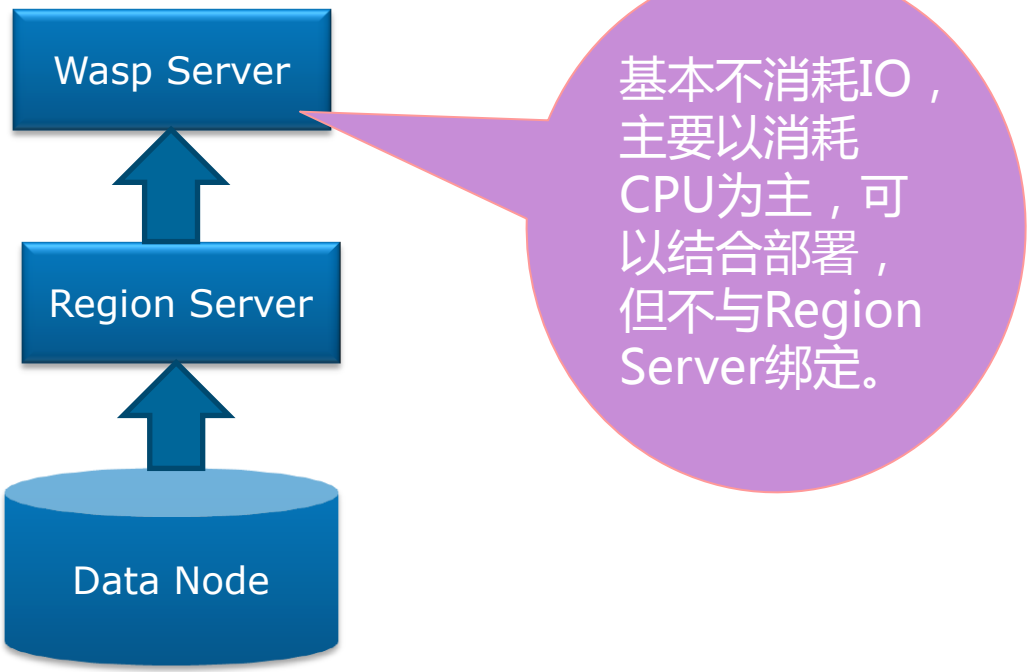
# Wasp的架构

- 整体架构



# Wasp的架构

- 整体架构
  - 单机进程部署

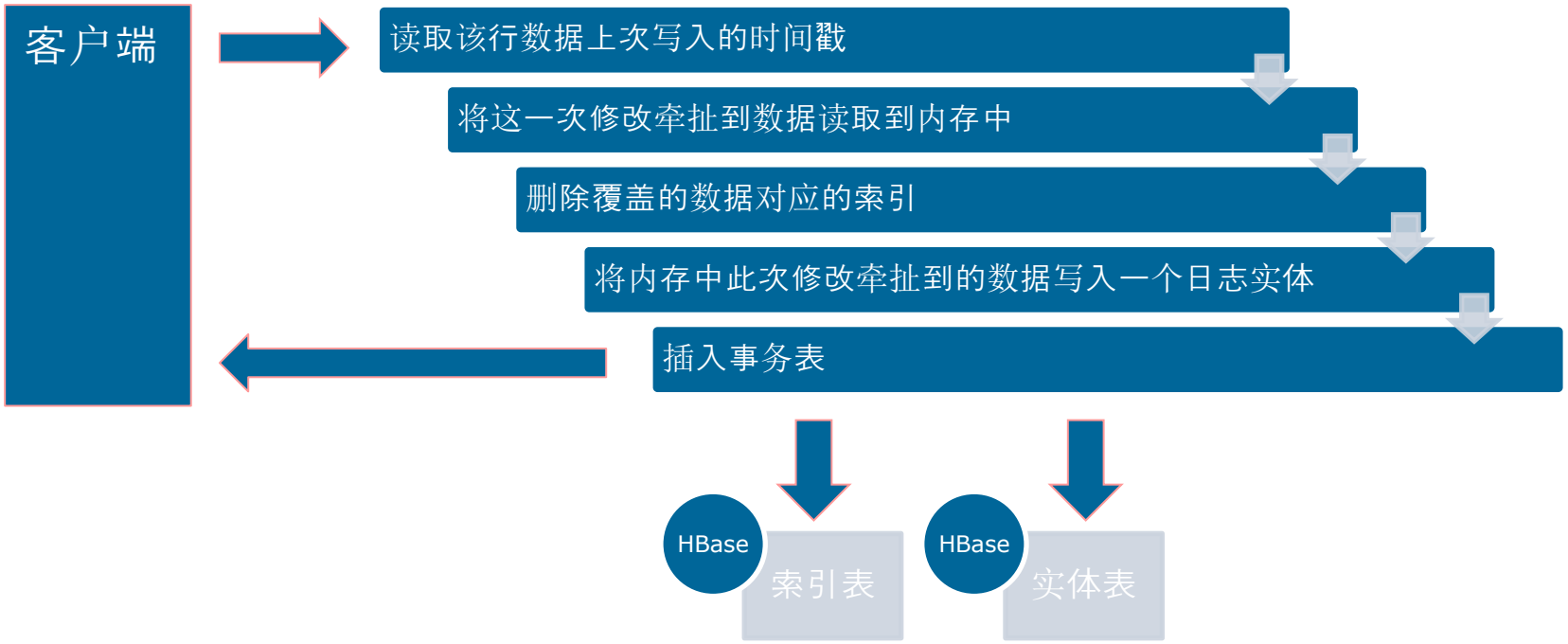


# Wasp的架构

- 事务
  - 写事务
    - 本地事务
      - Entity Group
      - Redo Log
    - 全局事务
      - 两阶段提交
      - 转化为本地事务

# Wasp的架构

- 本地写事务
  - Entity Group



# Wasp的架构

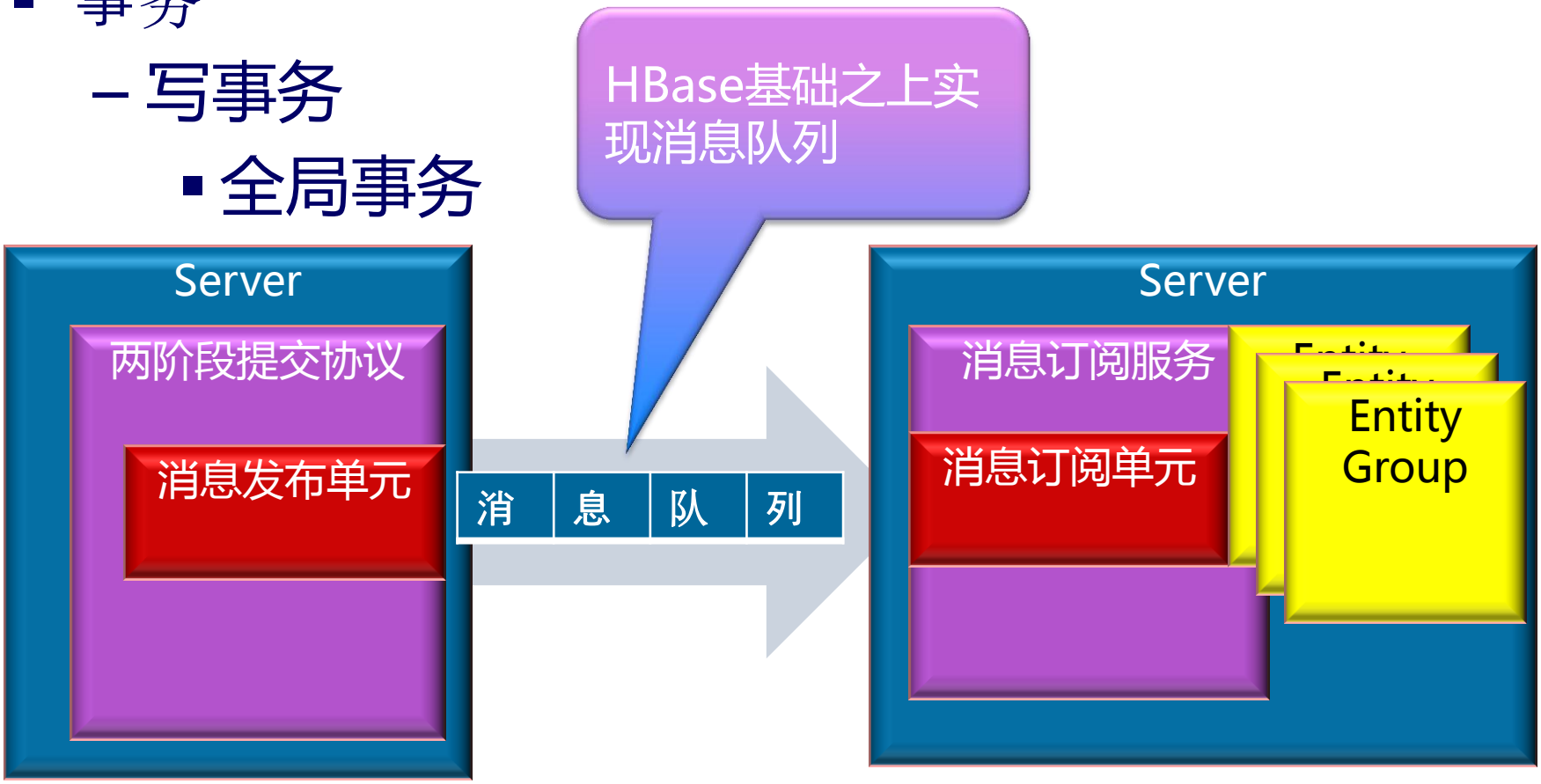
- 本地写事务
  - 不同row的写入相互不阻塞。

# Wasp的架构

- 本地写事务
  - 不同row的写入相互不阻塞。

# Wasp的架构

- 事务
  - 写事务
    - 全局事务



# Wasp的架构

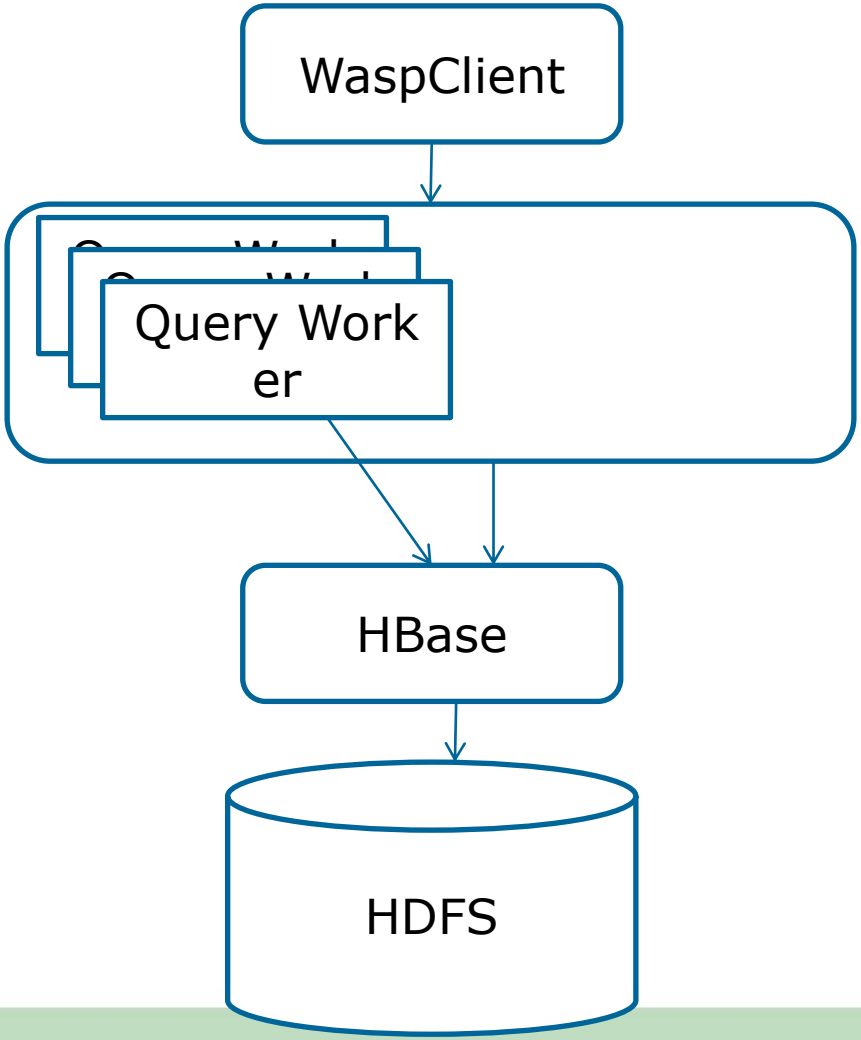
- 事务
  - 读事务
    - MVCC
      - 流式读取(时间戳优化)
      - 快照读取
      - 弱一致性读取

# Wasp的架构

- 读取事务
  - 理论: 乐观锁，MVCC
  - 性能: 流式读取 < 快照读取 < 弱一致性读取
  - 可靠性: 流式读取 > 快照读取 > 弱一致性读取
  - 模式: 用户自己选择应用模式

# Wasp的架构

- 读写流程中关键组件

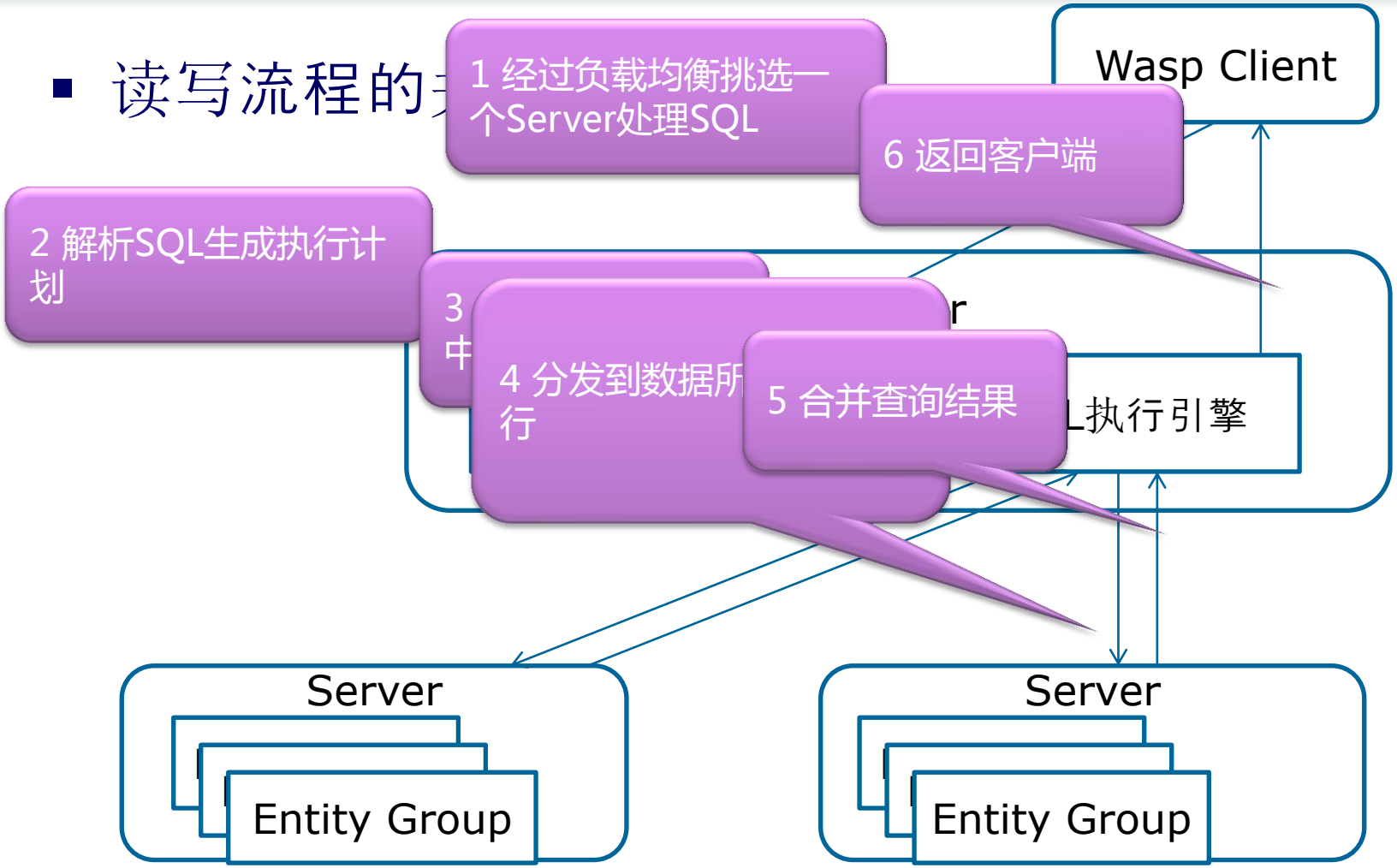


# Wasp的架构

- 二级索引
  - 用户创建索引表
  - 用户指定特定索引模式
  - 索引特性
  - 受限的的范围条件（where条件中同时最多支持一个范围条件）

# Wasp的架构

- 读写流程的



# Wasp的架构

- SQL
  - Create Table
  - Drop Table
  - Alter Table
  - Select
  - Delete
  - Update
  - Where

# Wasp的架构

- SQL
  - <>
  - =
  - 主键Join
  - 统计(Sum, Count)
  - 主键Group By

# Wasp的架构

- SQL
  - 数据类型
    - Float
    - Int
    - Long
    - String
    - Protobuf

# Wasp的架构

- SQL
  - 特性
    - 重复(repeated)
    - 主键(primary)
    - 可选(optional)
    - 必填(required)

# Road Map

- 目标应用
  - Select col1,col2,id from t where col3= ‘a’ and col4 = ‘b’ and col5>10;
  - Update t set col4= ‘b’ where id = 1234
  - Delete from t where id= 1234
  - Insert into t (列1, 列2,...) VALUES (值1, 值2,....)
  - 满足以上需求的有类似消费记录，会员营销，账务等应用。
- 计划发布时间->12月
- 开源->开源到社区

# Q&A

Q&A

# 谢谢！