



小红书云原生ClickHouse的 建设与实践

王成 / OLAP研发专家 / ClickHouse Contributor

REDtech

2023/03/25

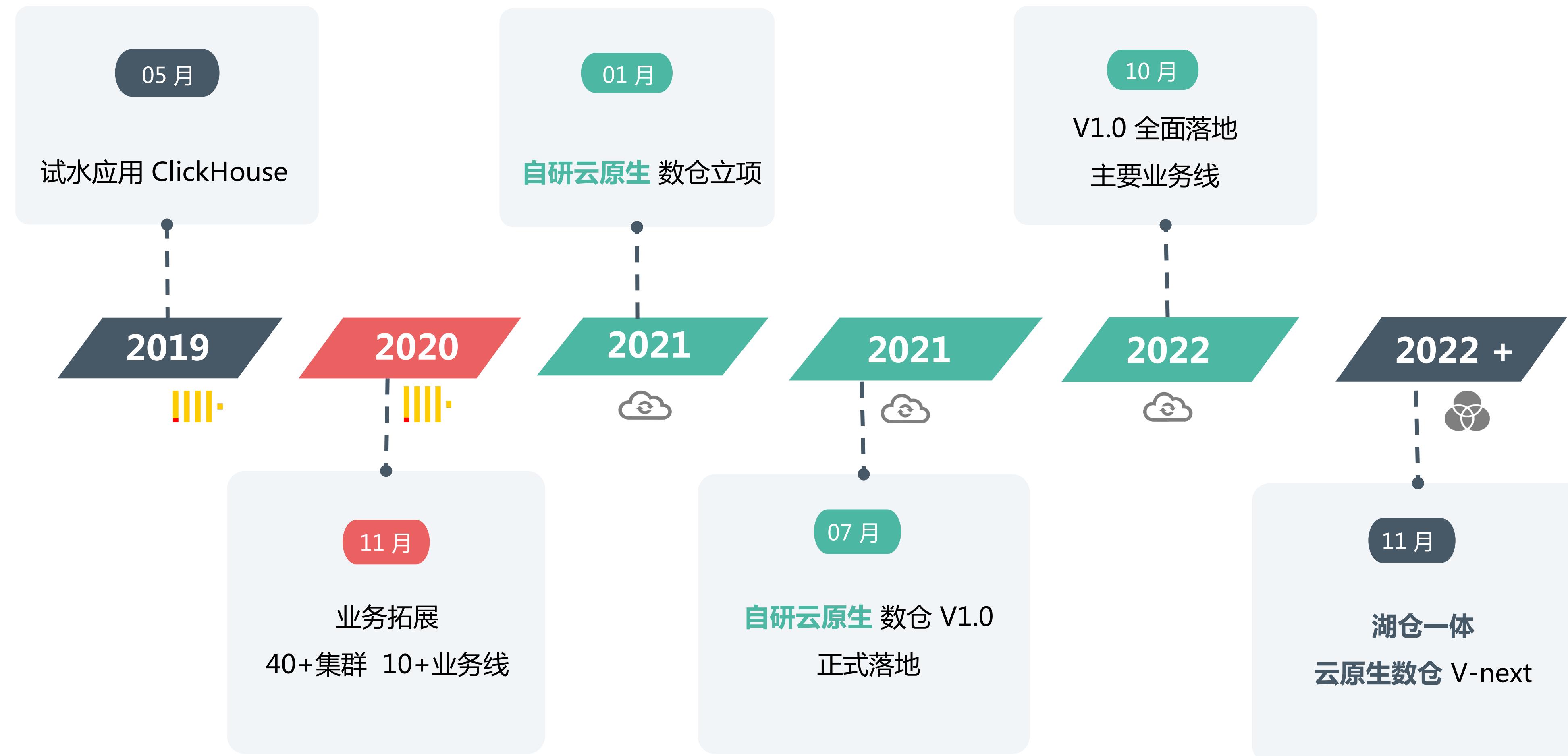
王成

- 2020年12月加入小红书大数据技术部
- 从0到1主导/负责小红书云原生实时数仓架构、落地与迭代工作

CONTENT

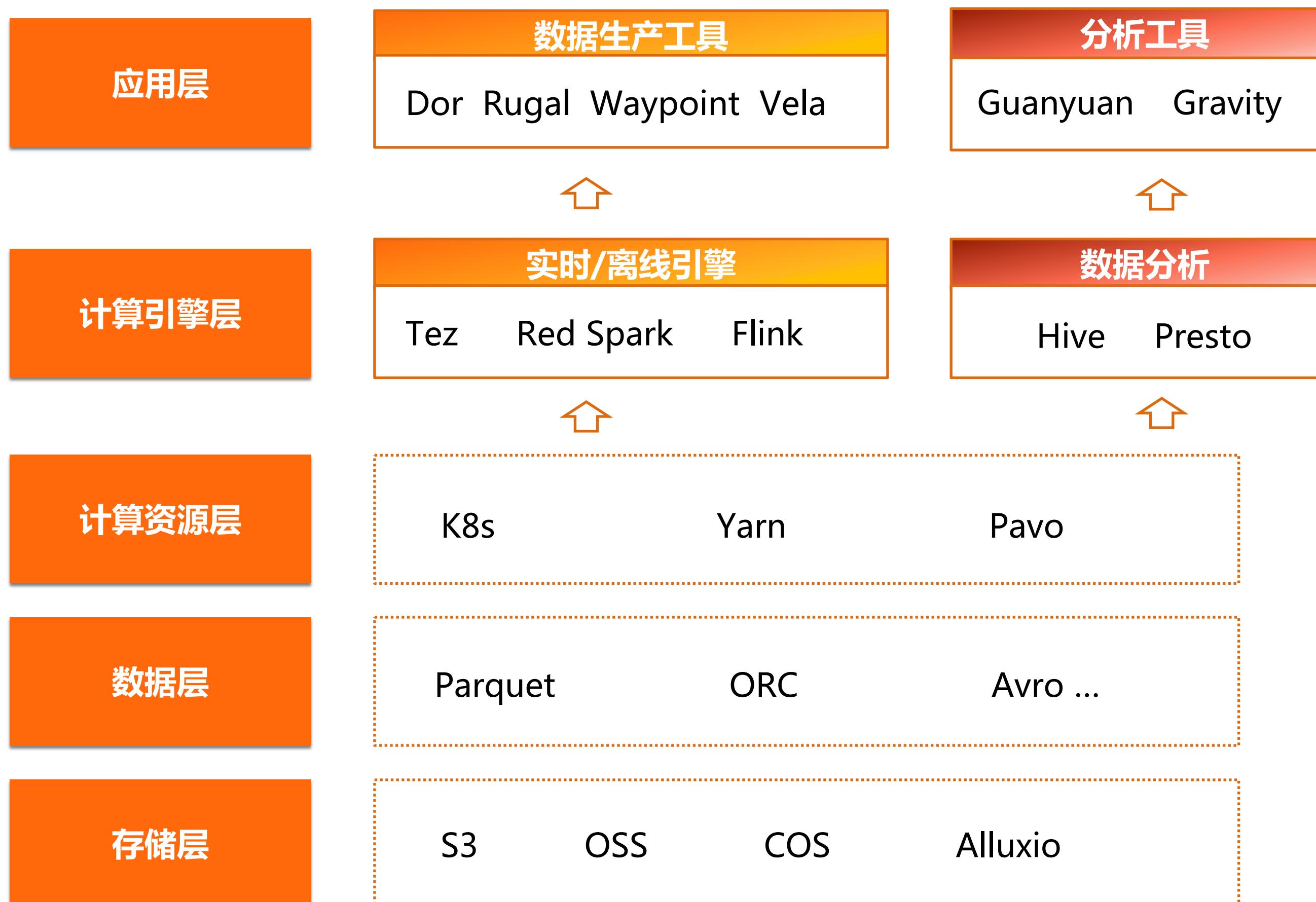
- 背景：云原生落地前，使用ClickHouse遇到的问题和挑战
- 云原生 ClickHouse V1.0 建设之路
- V-next 湖仓一体建设规划

实时数仓发展历程

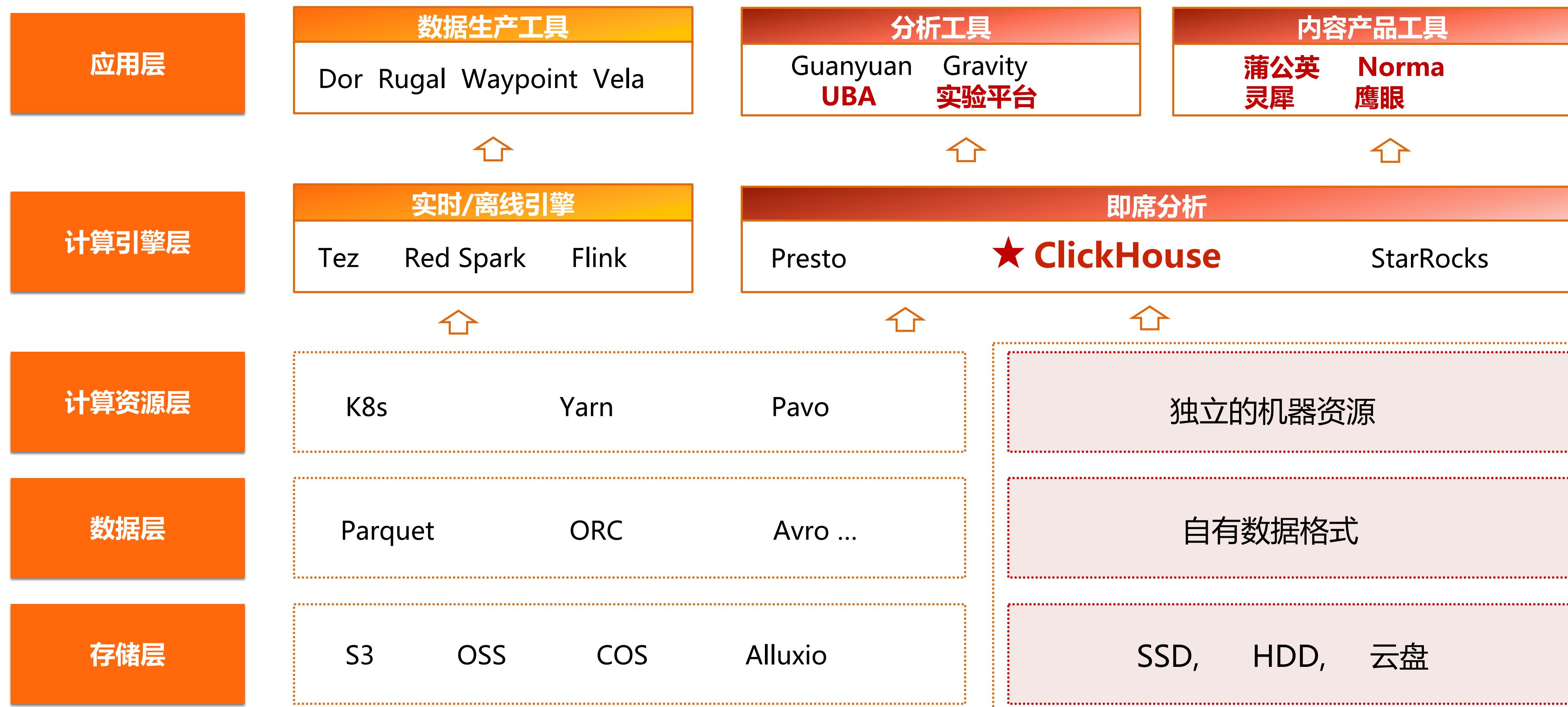


小红书的云原生大数据架构

- 云上的原住民



引入ClickHouse



云原生落地前ClickHouse

遇到的问题和挑战

痛点

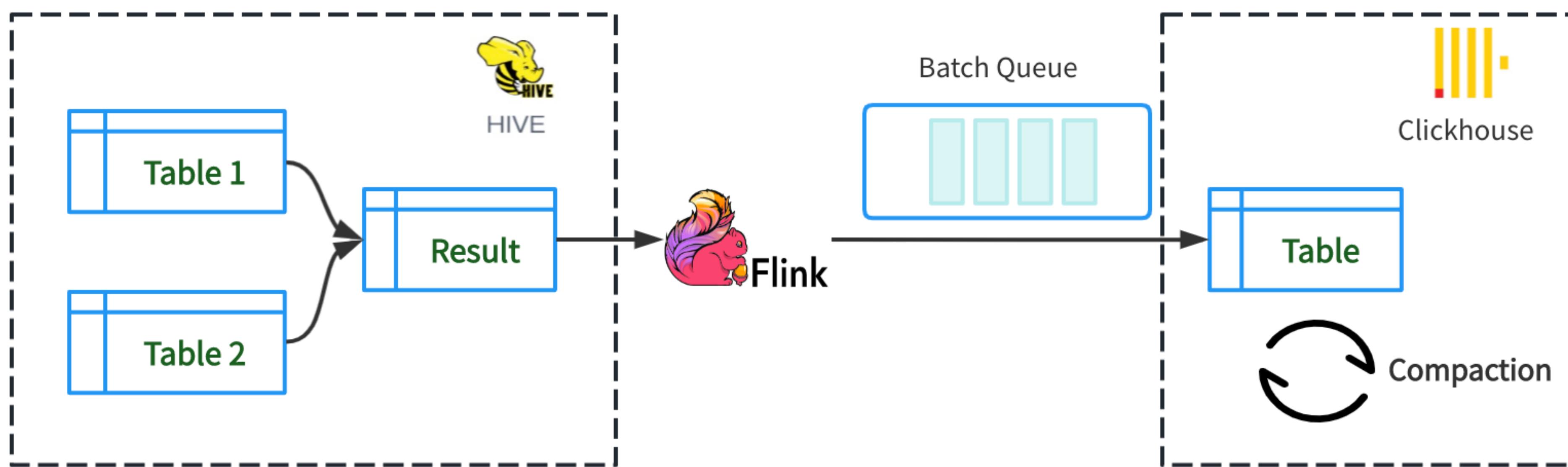
- 扩容难
- 数据同步难
- 算力浪费严重
- 查询体验不稳定

扩容难

- 扩容周期长
- 需要额外的数据搬迁或重写
- 多副本机制引入的中心瓶颈

数据同步难

- 数据摄入的一致性问题
- 数据同步链路复杂



算力浪费严重

- 集群容量预估困难
- 存储和计算需求差距大
- 业务需求呈现规律性的波峰谷振荡

用户查询体验不稳定

- 高高峰期查询阻塞，失败率高
- 多用户/业务线互相干扰

解决方案：云原生

问题：

- 存算耦合
- 资源隔离能力弱
- 数据同步成本高、一致性差

目标：

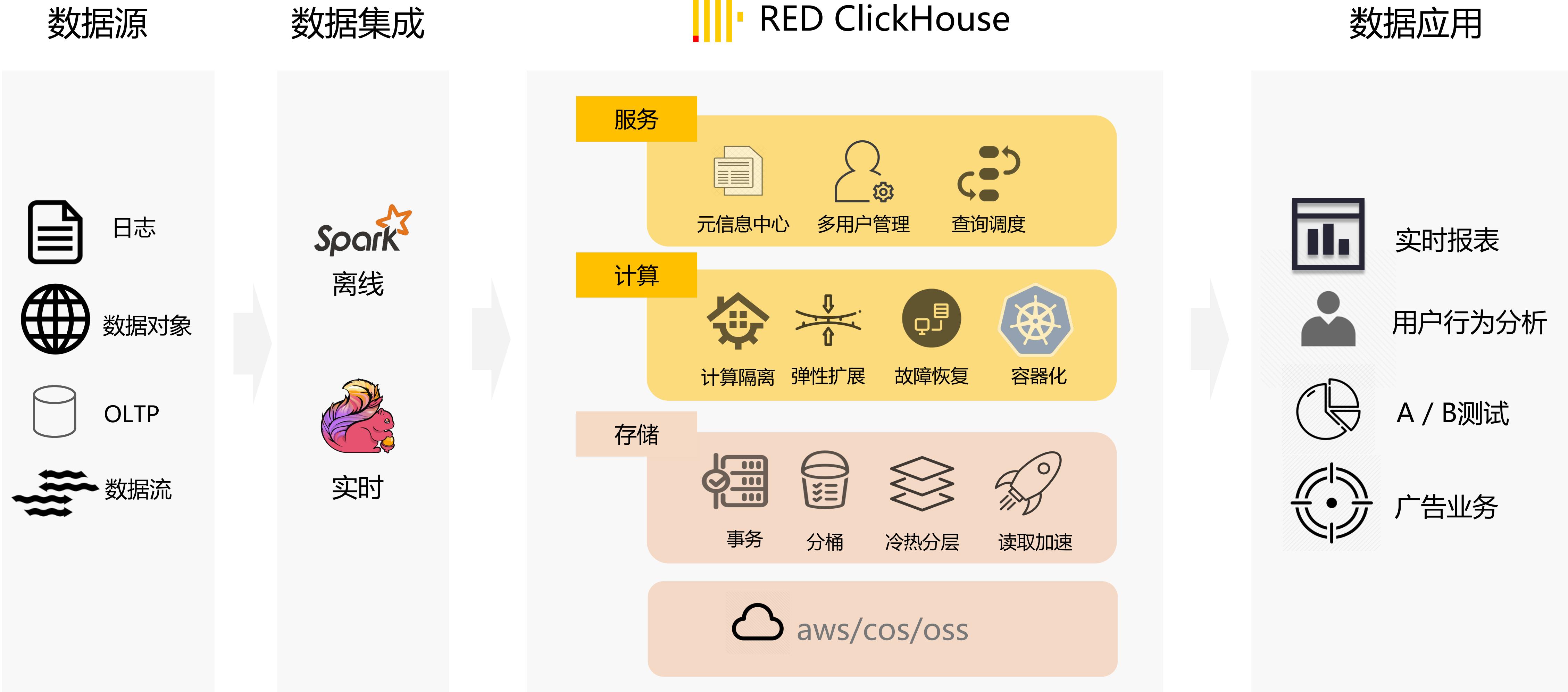
- 存算分离、弹性扩展
- 提供更强的计算资源隔离能力
- 打破数据壁垒
- 支持事务和原子性

自研云原生实时数仓的价值

- 灵活性：快速响应业务
- 自主可控：保障性能、稳定性、安全性
- 符合小红书多云战略

云原生ClickHouse V1.0 建设之路

云原生OLAP V1.0



云原生存算分离架构

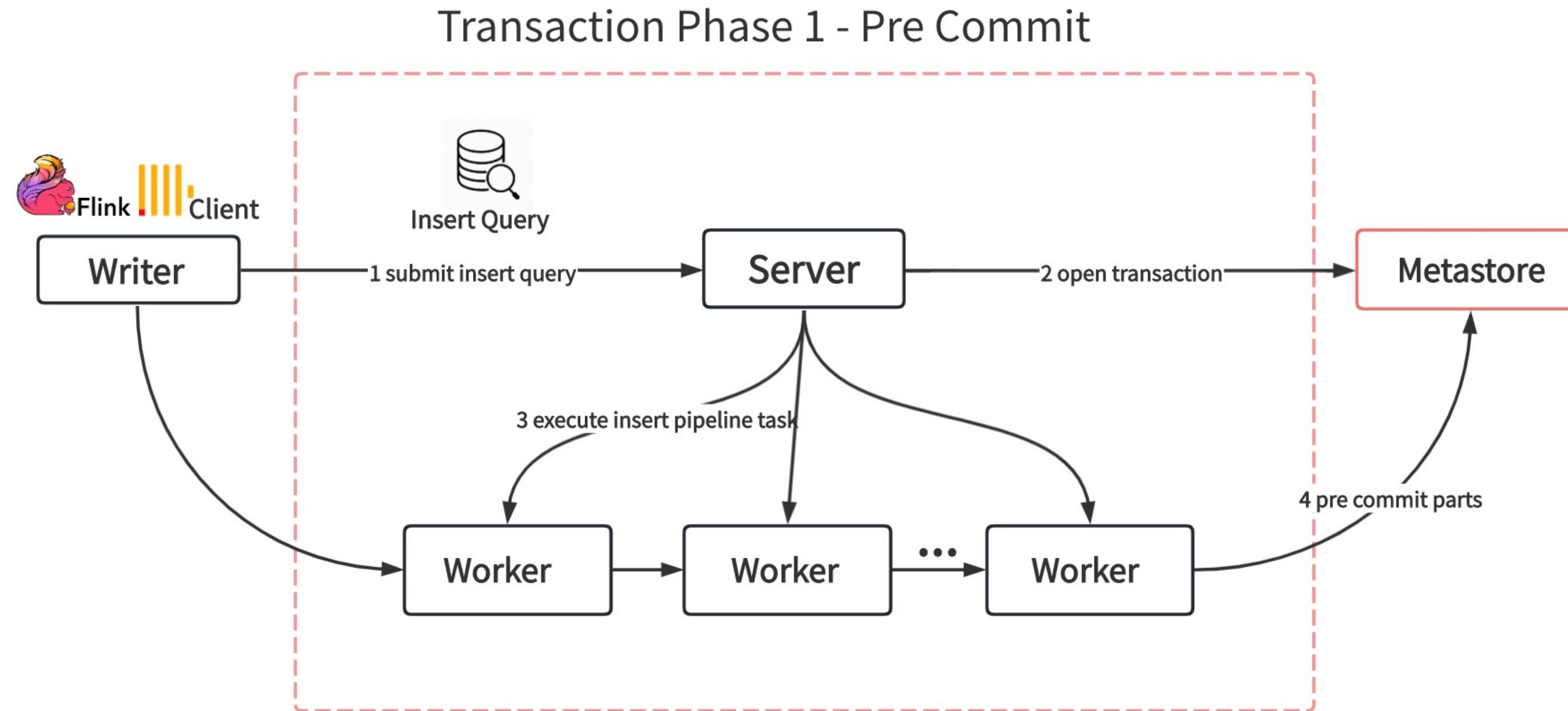
架构特性:

- 共享元信息中心，共享存储
- 基于云存储，按需使用，无限扩展
- 计算资源池化，以计算组为单位，弹性扩展



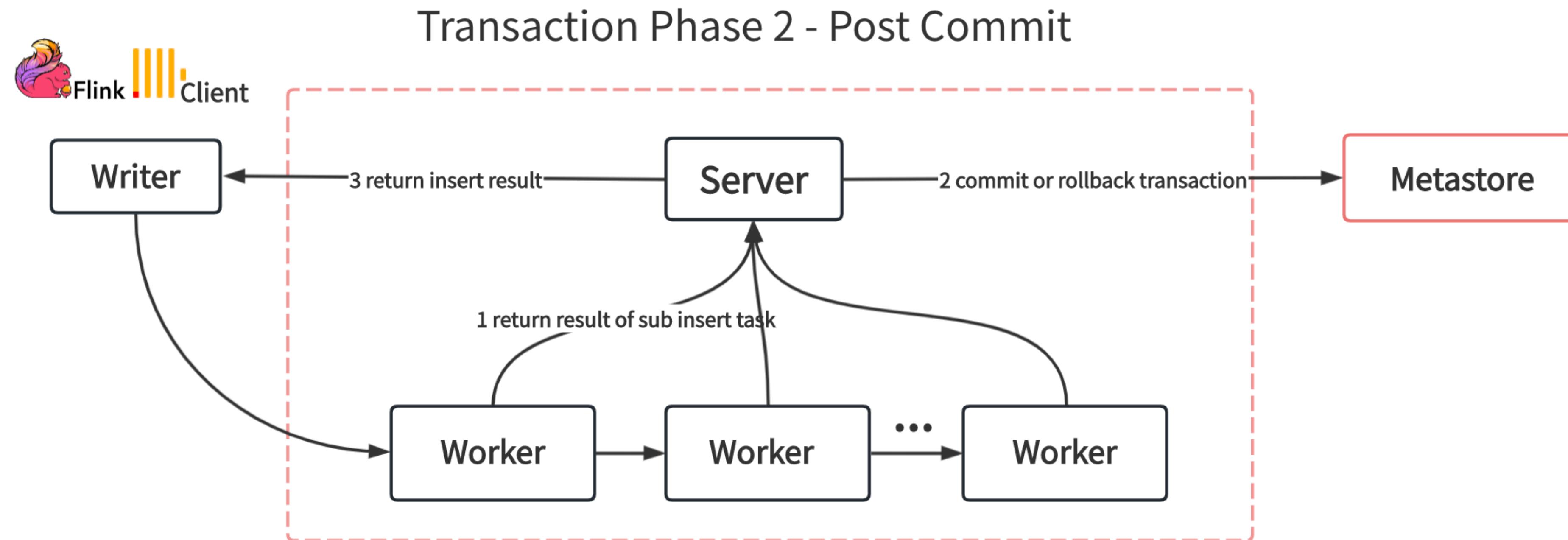
分布式执行框架

- 分布式写入事务



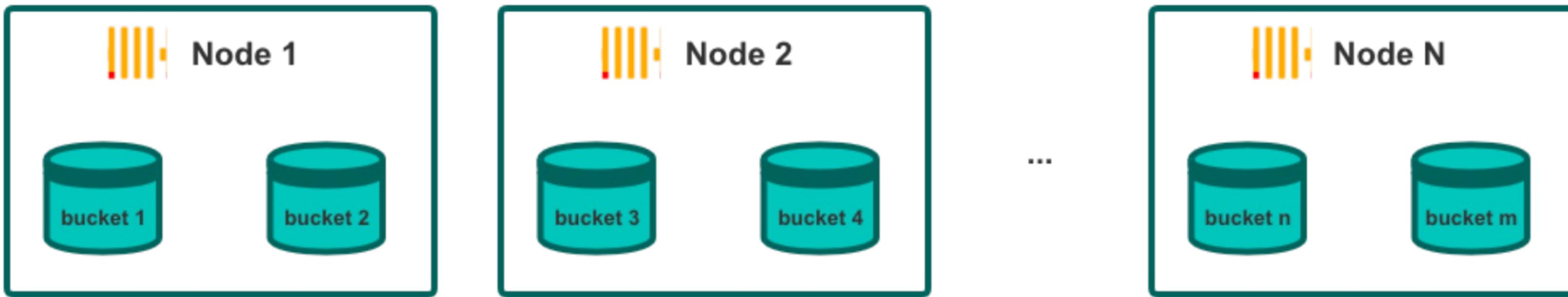
分布式执行框架

- 分布式写入事务

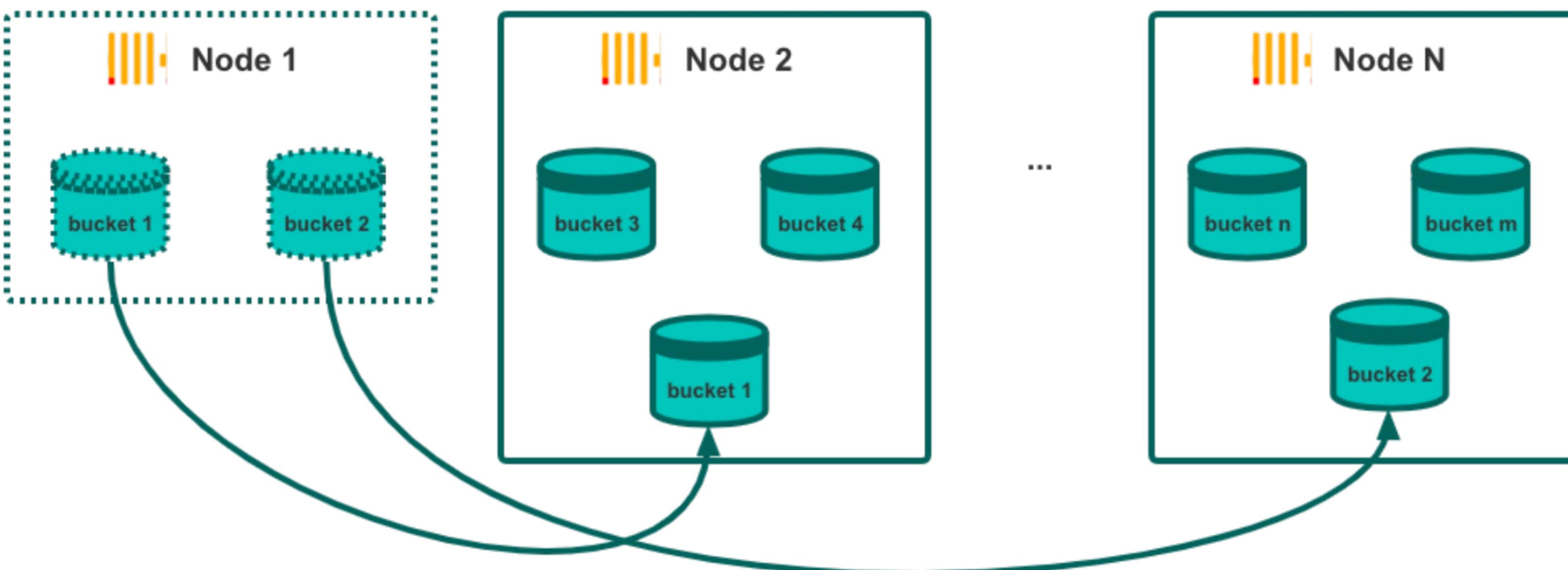


弹性扩容和故障容错

正常情况下，每个节点拥有属于自己的桶



当节点发生异常，异常节点的桶会被重新划分给其他正常运行的节点



分布式存储选型

对象存储 ——

优点

- 价格低
- 无限扩容
- 高并发

缺点

- 延迟高
- 单点性能低
- 不稳定

挑战 ——

查询分析

- 查询性能慢
- 查询受网络影响，成功率不稳定

数据摄入

- 亿级QPS数据无法实时写入
- 写入不稳定，影响数据一致性

查询优化尝试

- 性能优化
 - 执行计划优化
降低总的连接延迟
 - 并行读取
以列为单位并行读取
- 稳定性优化
 - 断点续传

飞跃提升：

优化前：查询成功率基本为0

优化后：查询性能相比本地磁盘
仍有**10倍**的差距

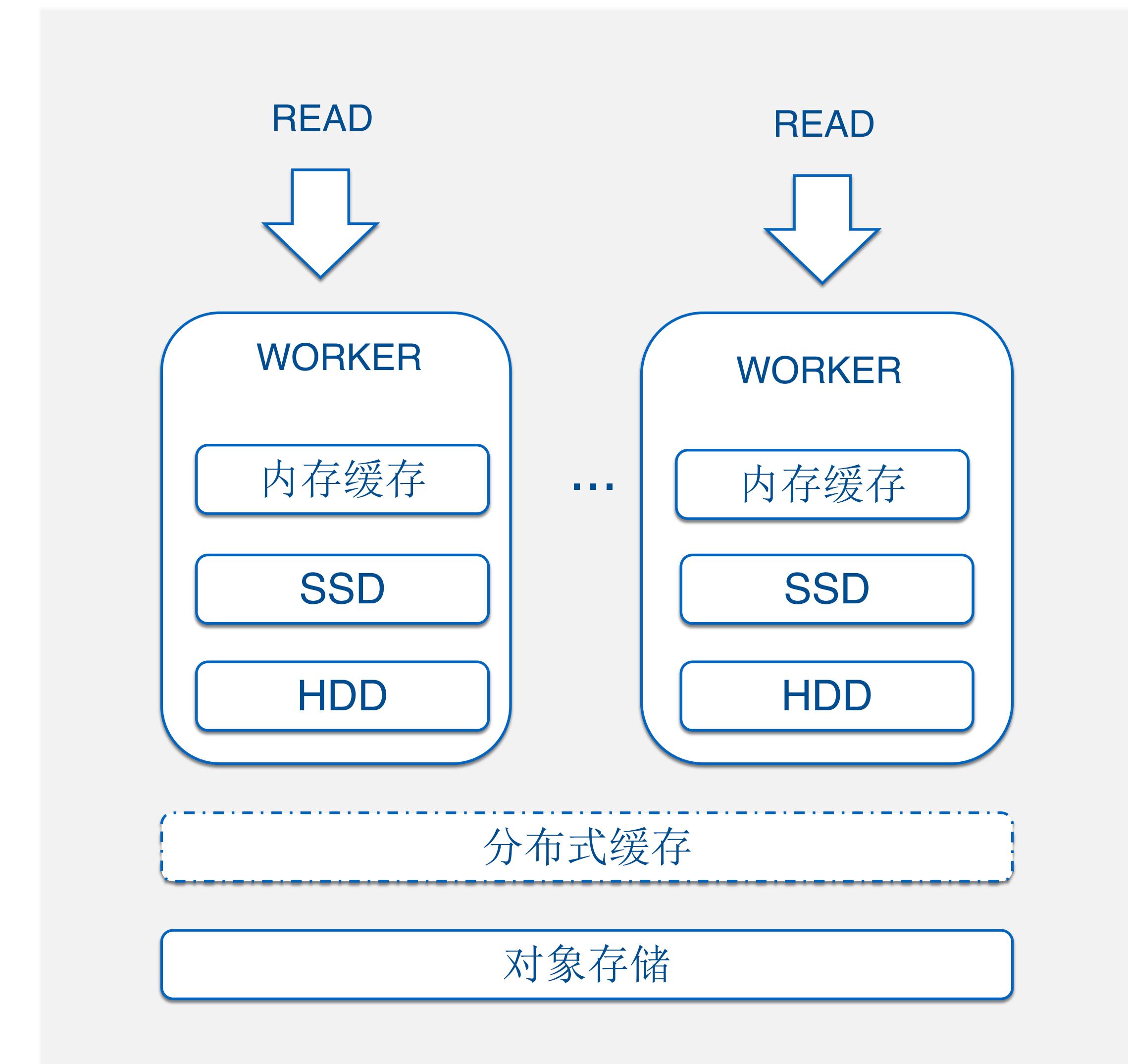
多级智能缓存

- 加速数据查询

- 被动缓存
- 主动缓存
 - 元信息缓存
 - 表级别自定义缓存策略
 - 基于**查询历史**的数据热度预测

数据热度 + 多级磁盘配置 =>

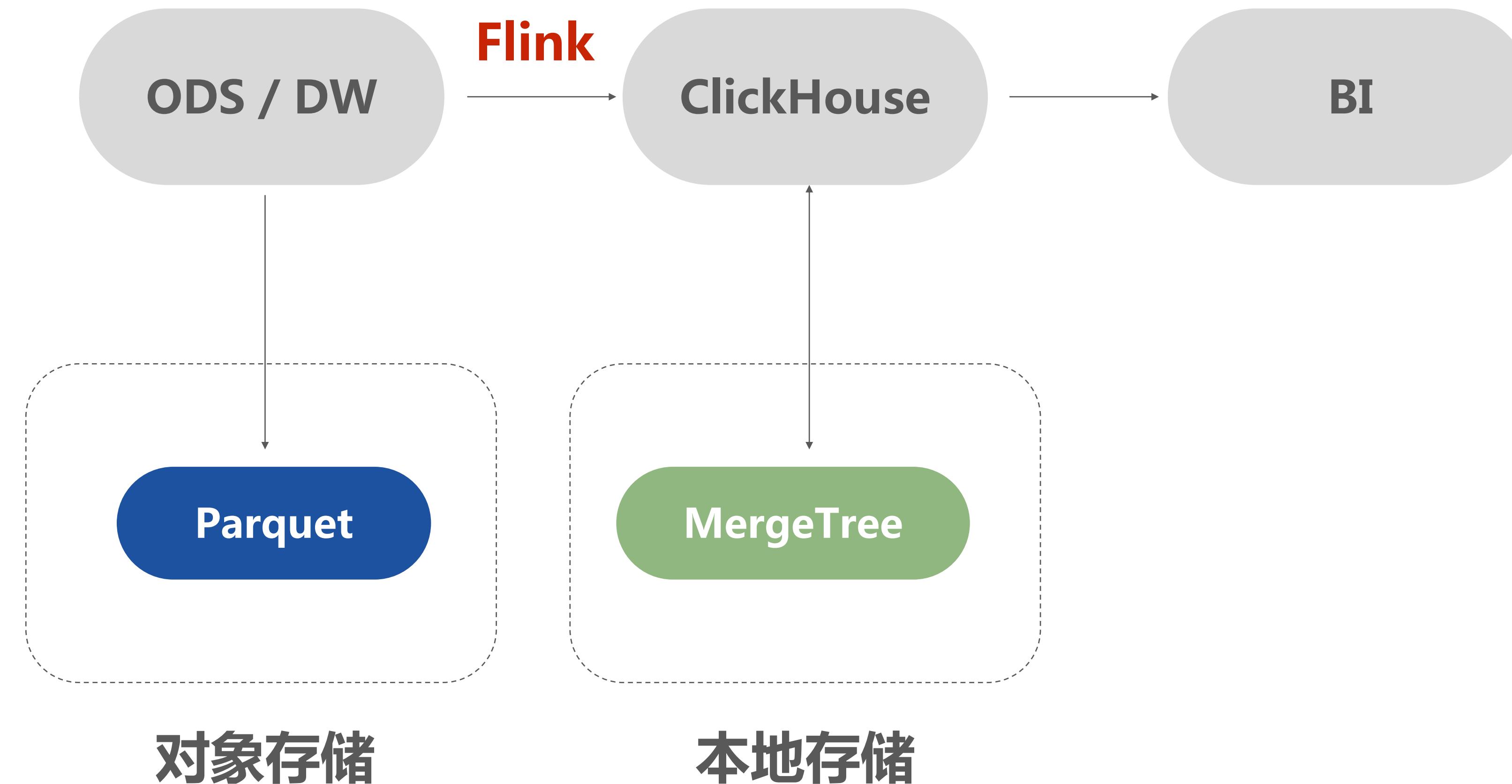
全局最优缓存计划



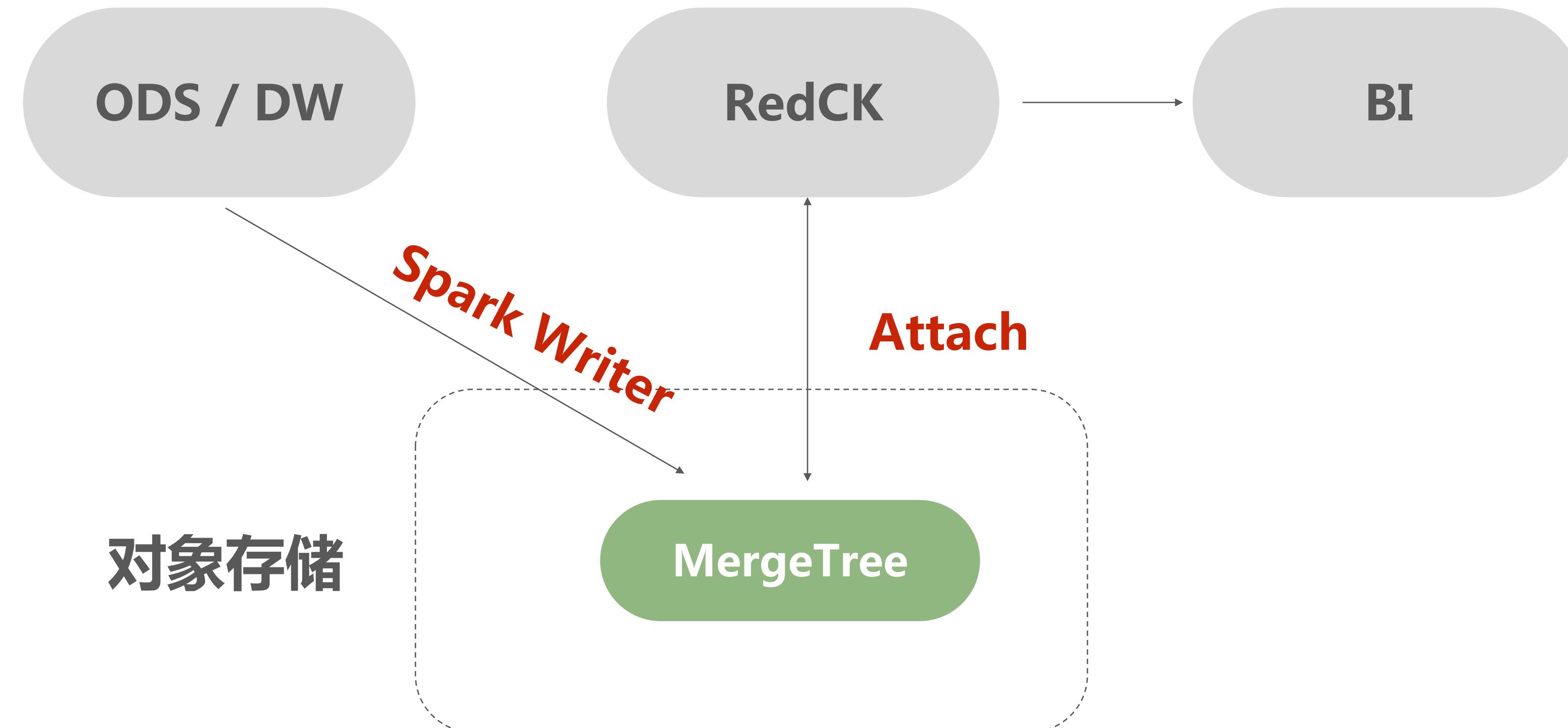
海量数据实时分析 - 分层存储

写入	内存	云盘	对象存储
↓	延迟	***	**
云盘	IOPS	***	**
↓	可靠性	*	***
对象存储	扩展性	*	**
	成本	*	**

离线数据同步链路优化-优化前



离线数据同步链路优化-优化后



总结

- 存算分离
 - 计算资源弹性伸缩
 - 存储资源按需申请

数据摄入

- 千万级实时数据集成 => **分层存储**
- 写入不稳定 => **Exactly-once语义**
- 数据同步链路长 => **Spark MergeTreeWriter**

查询分析

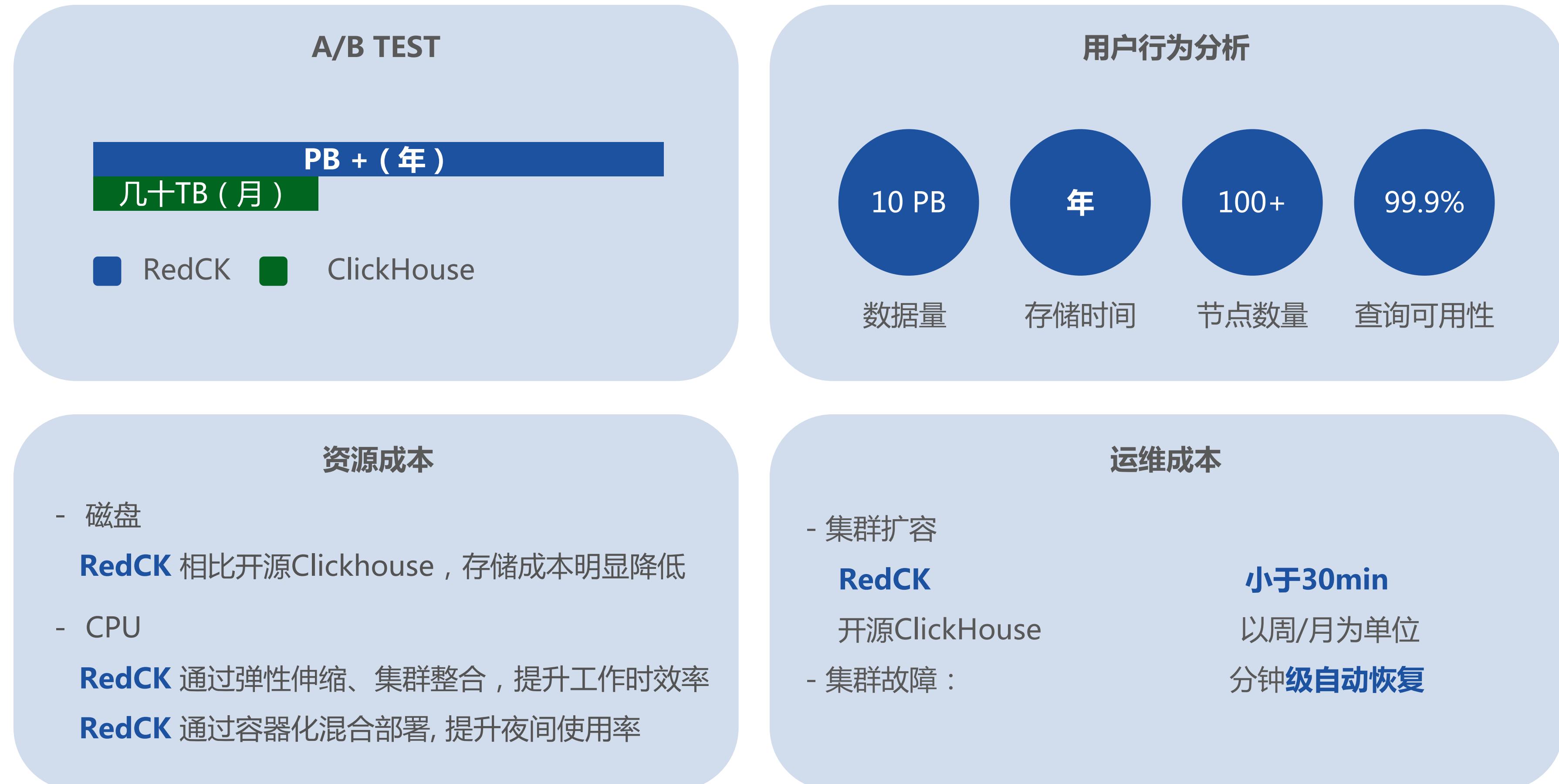
- 查询性能慢 => **多级智能缓存**
- 关联分析瓶颈 => **Bucket模型、数据本地化**

集群可用性

- 快速的故障自动恢复机制
- 分布式缓存，加速预热

业务落地实践

降本提效

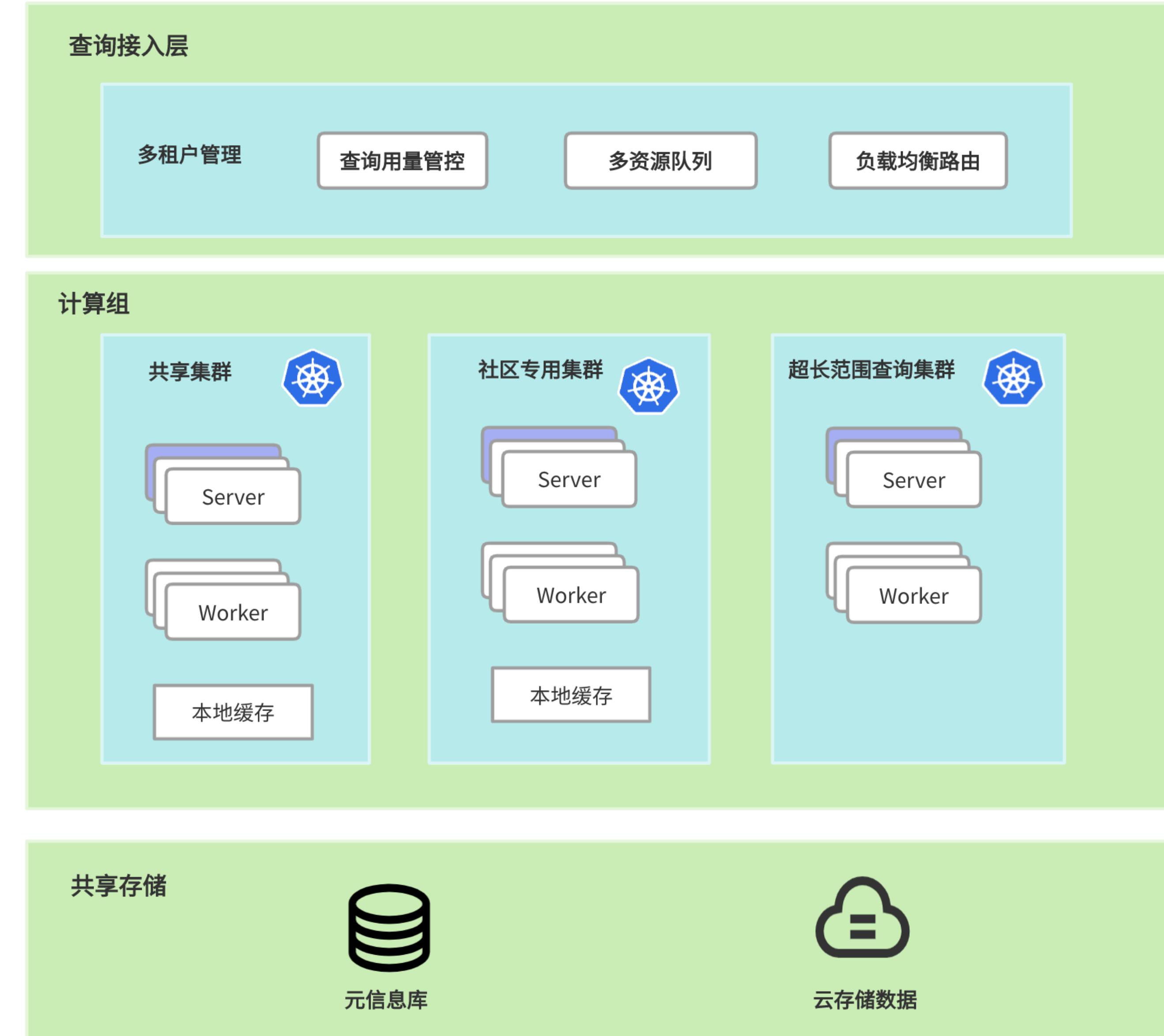


实验平台

- 多租户管理

计算组拆分规则

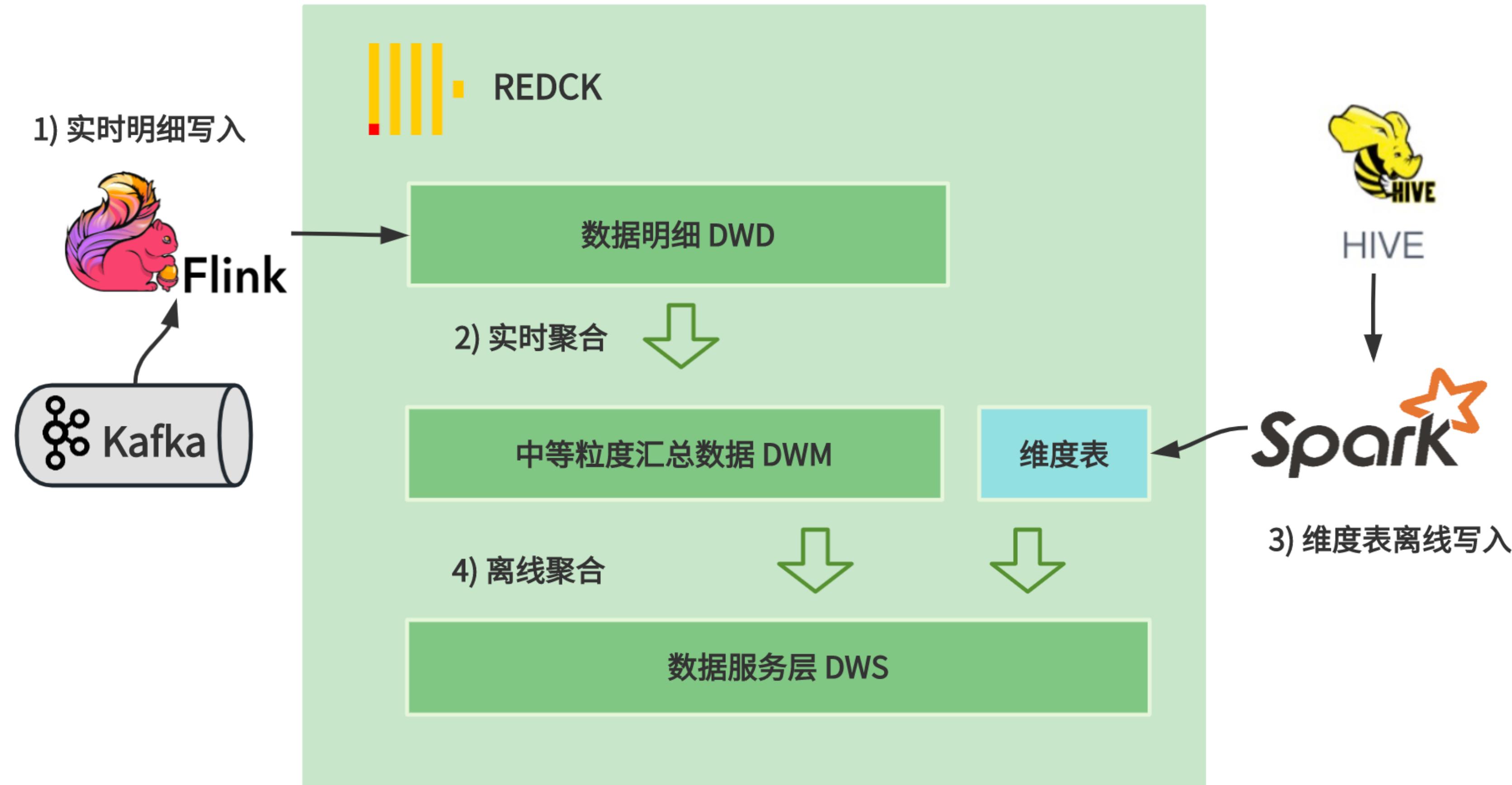
- 业务需求
 - 优先级
 - 查询类型
 - 在线查询
 - 重查询
 - 冷查询
 - 灰度队列
 - 弹性扩容队列



用户行为分析

- 海量数据实时接入

用户行为分析

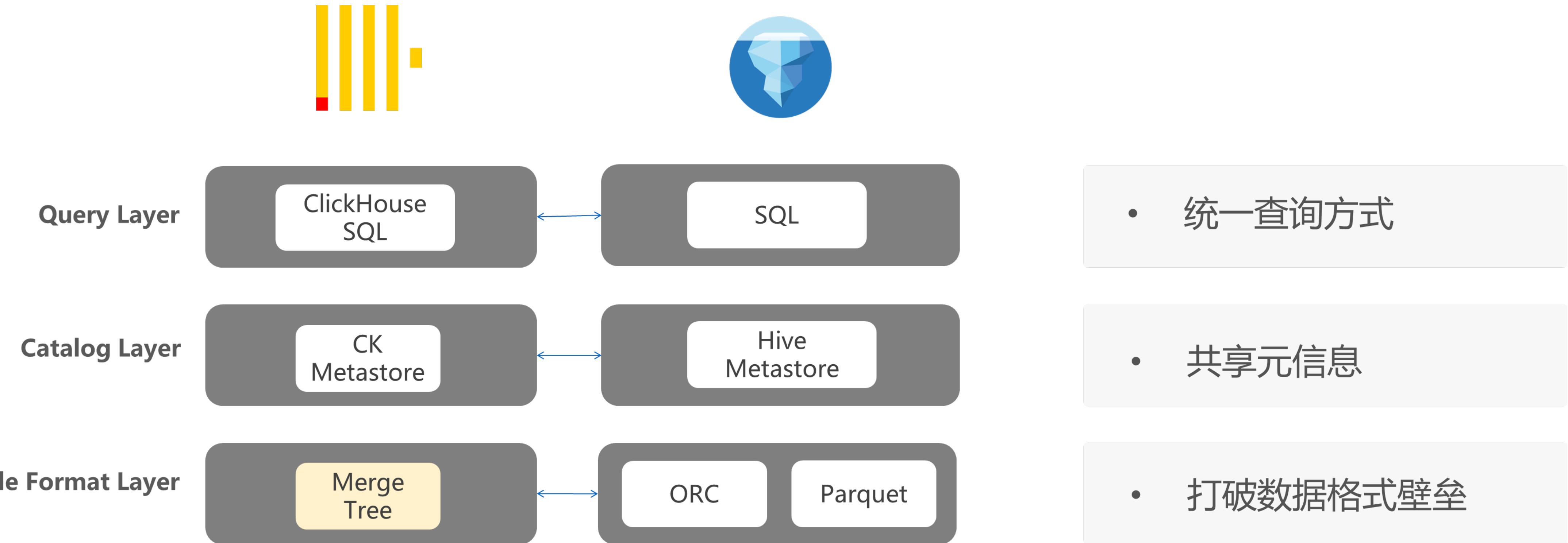


V-next: 湖仓一体建设

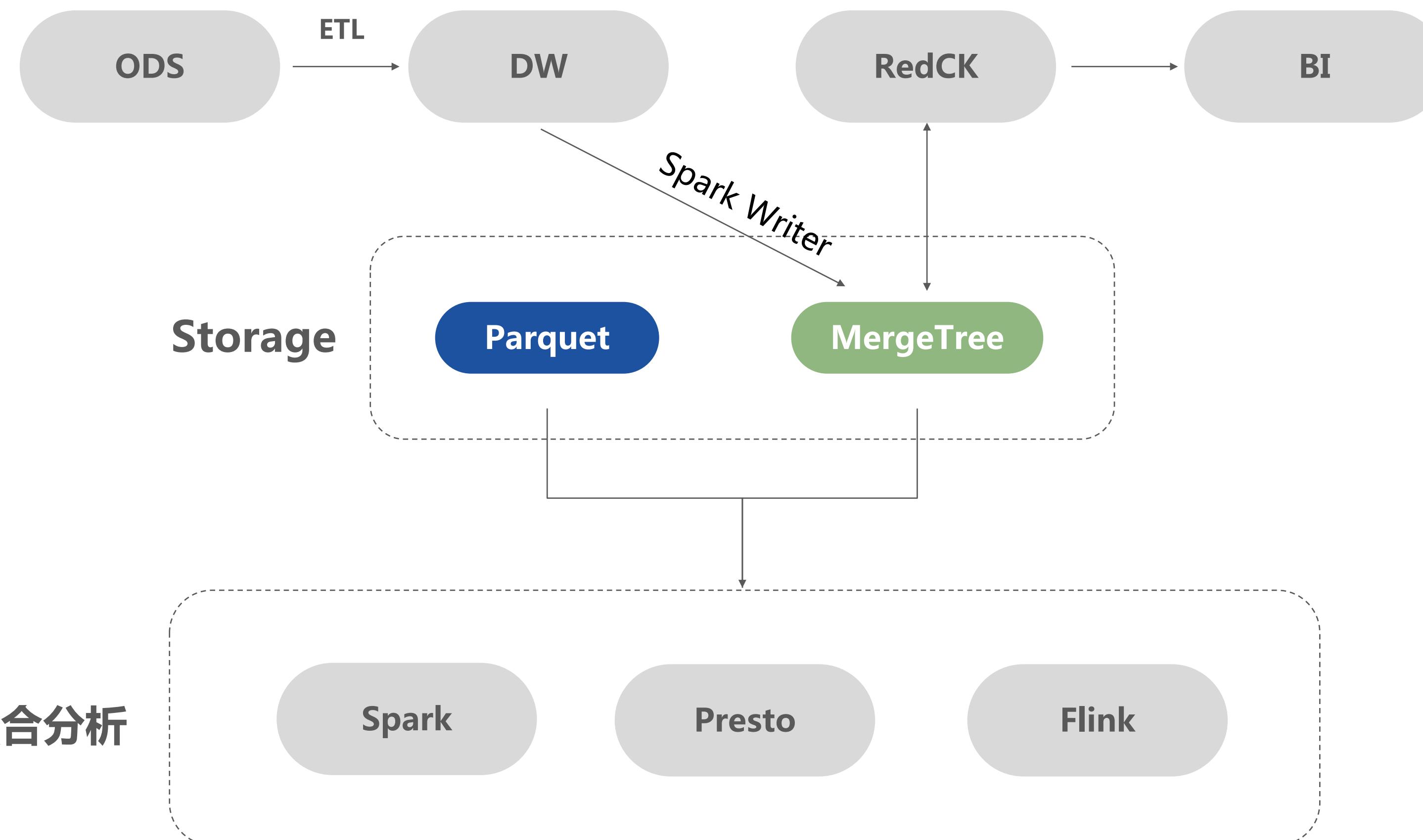
为什么要湖仓一体化？

- 数据湖和实时数仓的数据是割裂的，造成了大量的存储和计算冗余
- 目前的实时数仓无法应对复杂ETL加工的场景
- 查询方式不同引入额外的使用成本

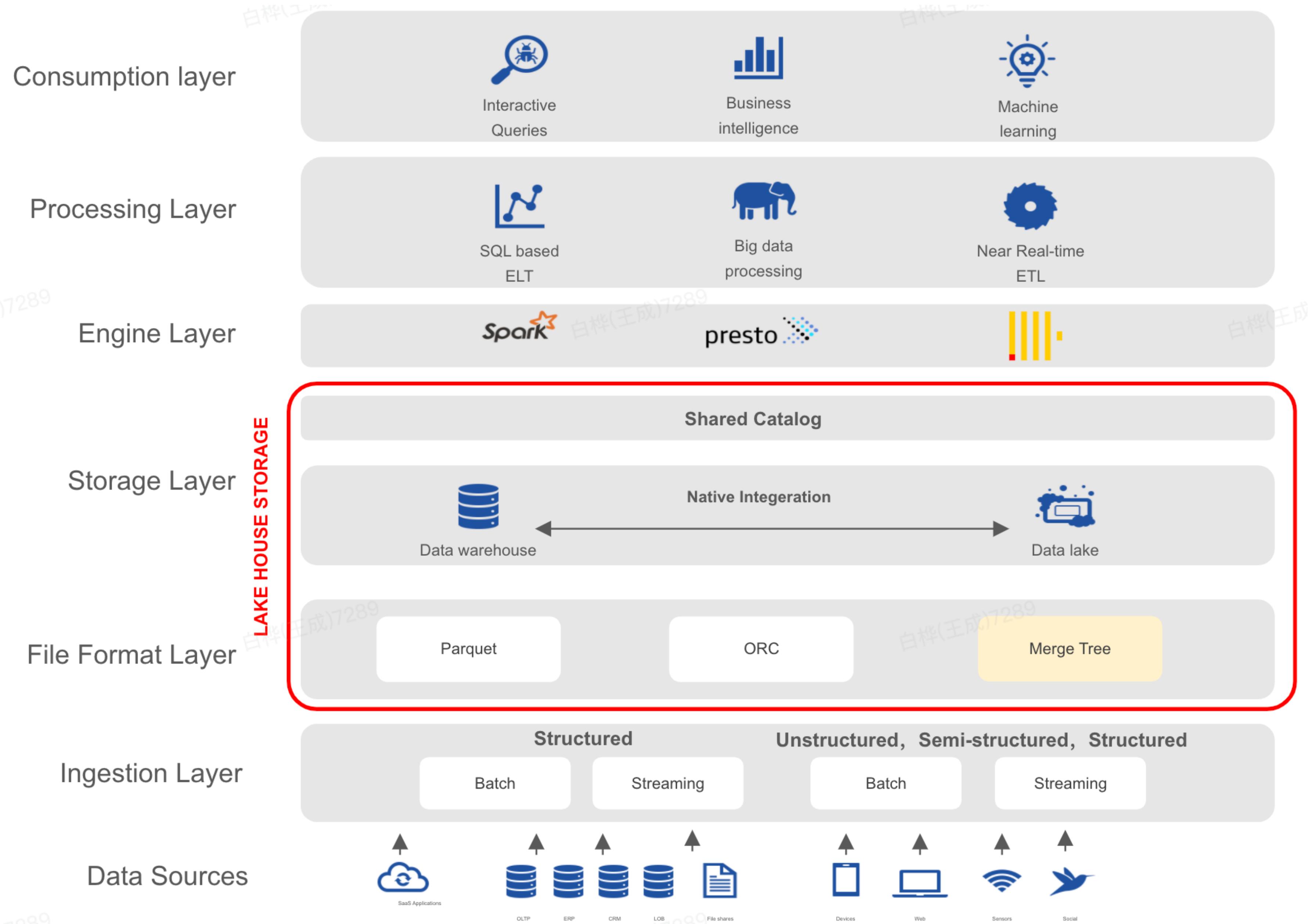
湖仓一体

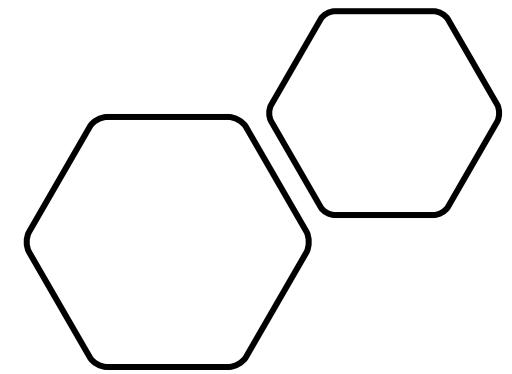


开放的MergeTree格式



湖仓一体





未来规划

- 持续推进湖仓统一建设
- 自动敏捷的弹性伸缩机制

THANKS

小红书

热招中



欢迎关注
小红书技术 REDtech

