

# ByConity：基于云原生架构的开源实时数仓系统

火山引擎ByteHouse资深研发工程师 / 游致远

# Agenda

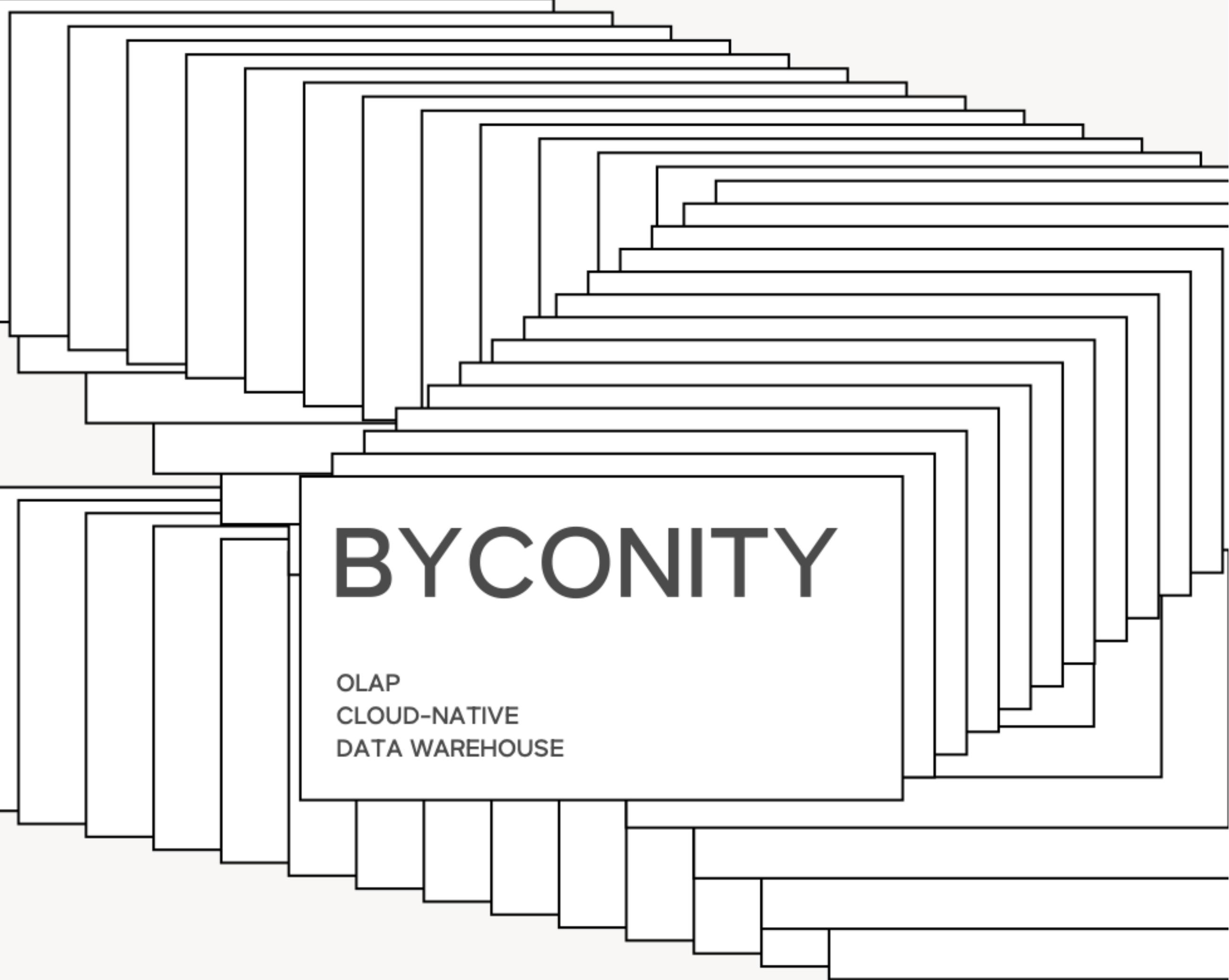
- 01 背景
- 02 整体架构和基础概念
- 03 核心功能
- 04 应用案例
- 05 总结和展望



BYCONITY

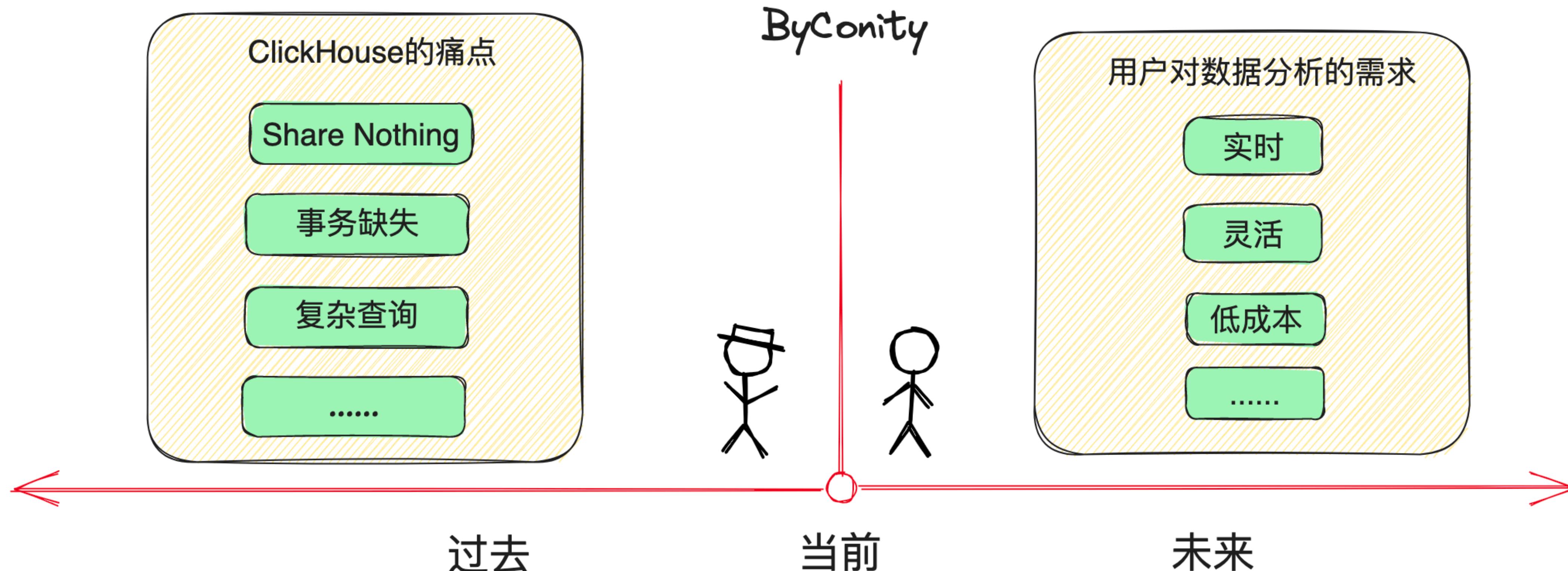
OLAP  
CLOUD-NATIVE  
DATA WAREHOUSE

# Agenda

- 
- 01 背景
  - 02 整体架构和基础概念
  - 03 核心功能
  - 04 应用案例
  - 05 总结和展望

# ByConity想解决什么问题

字节跳动开源的云原生数据仓库(<https://github.com/ByConity/ByConity>)



# ClickHouse的痛点

- Shared Nothing架构
  - 运维困难：扩缩容、读写分离、资源隔离困难
  - 资源浪费：存储和计算无法独立扩容、弹性伸缩
- 事务支持缺失
  - 不满足对数据一致性要求高的场景
  - 提高了使用和运维成本
- 复杂查询性能差（如多表Join）

# ByConity历史



# ByConity的特点

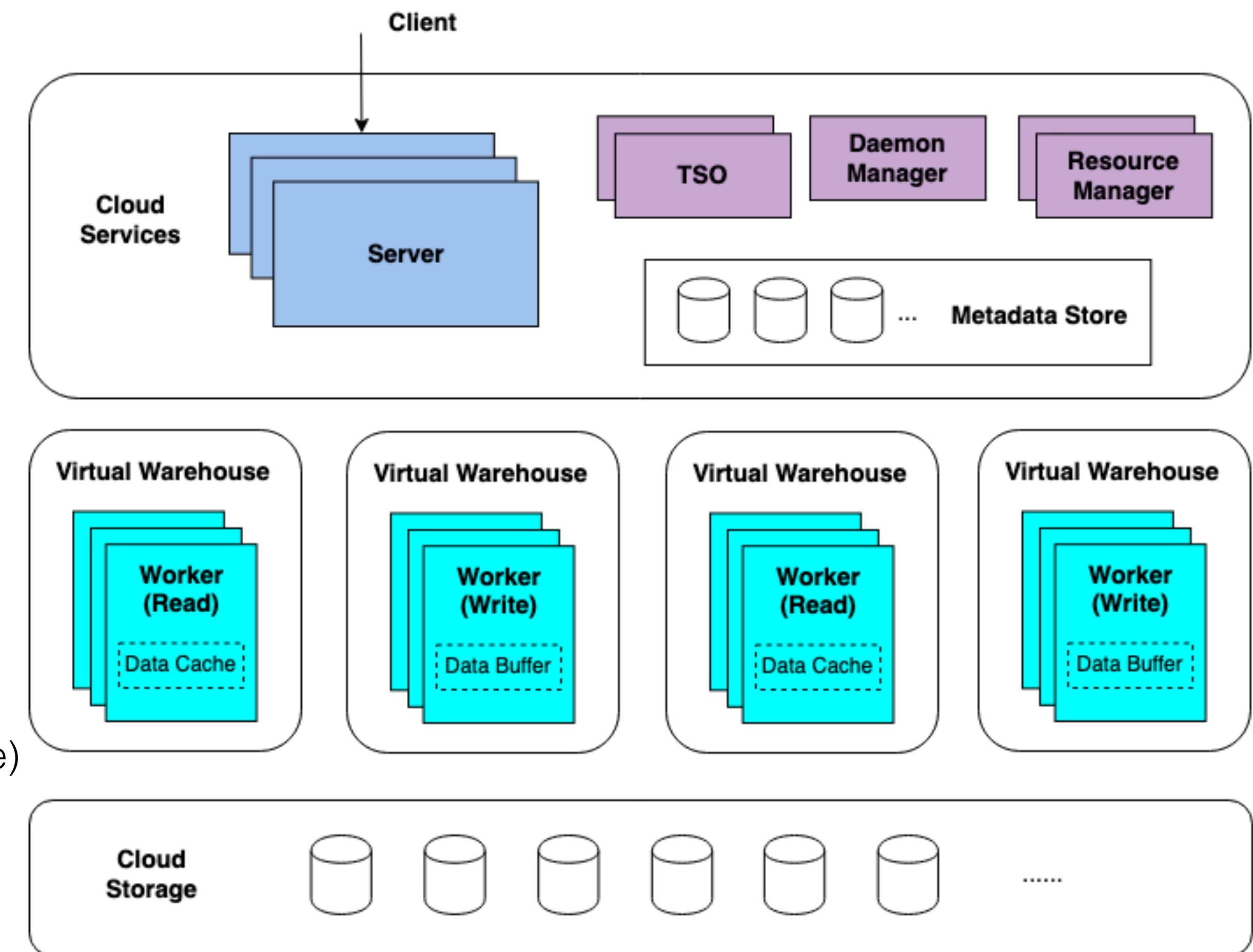
- 实时：查询性能（单机、分布式）极致优化、高性能流式数据导入
- 灵活：通用的功能如事务、一键、预聚合模型
- 低成本：云原生架构、存算分离、计算层弹性伸缩

# Agenda

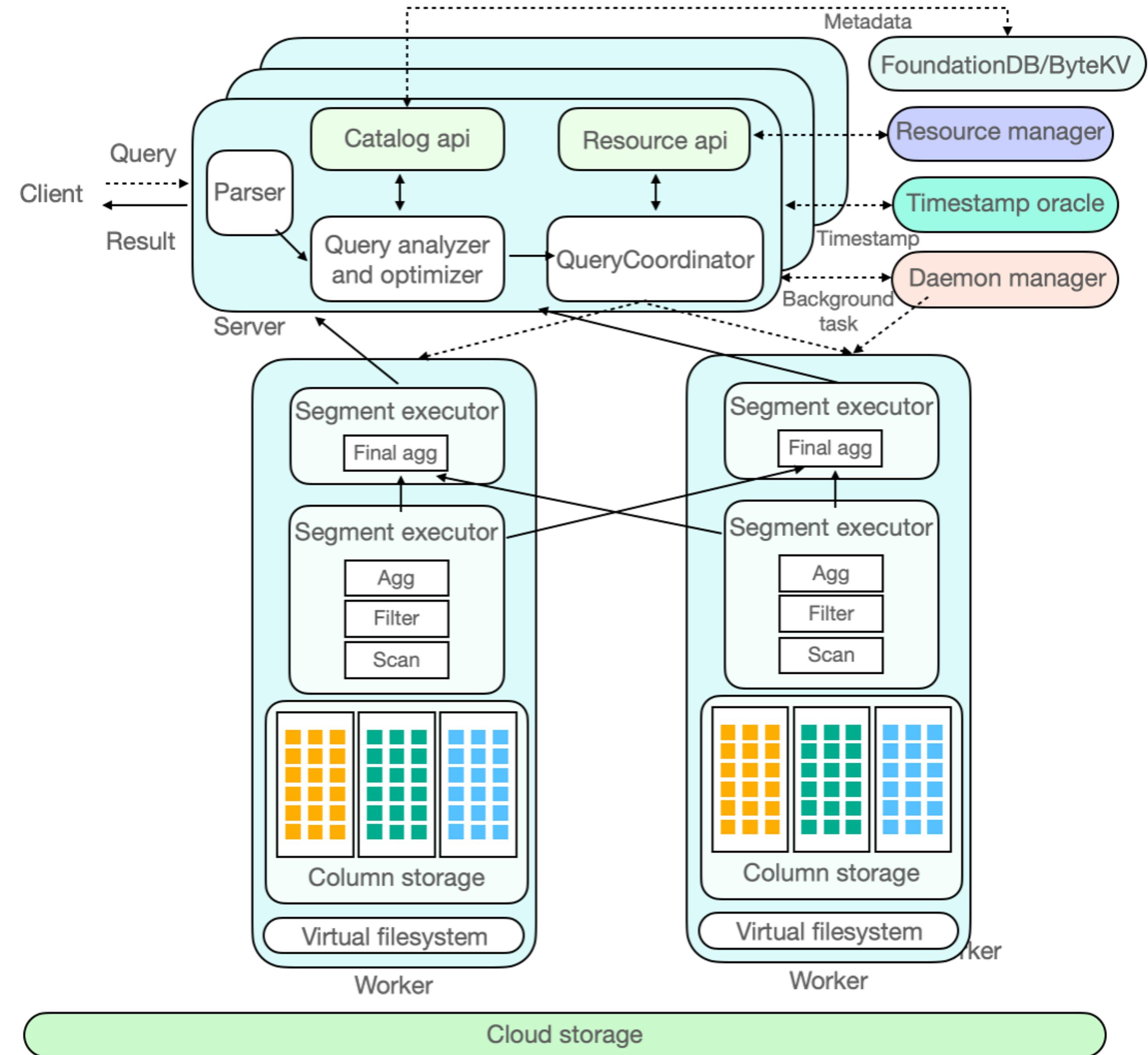
- 
- 01 背景
  - 02 整体架构和基础概念
  - 03 核心功能
  - 04 应用案例
  - 05 总结和展望

# 整体架构

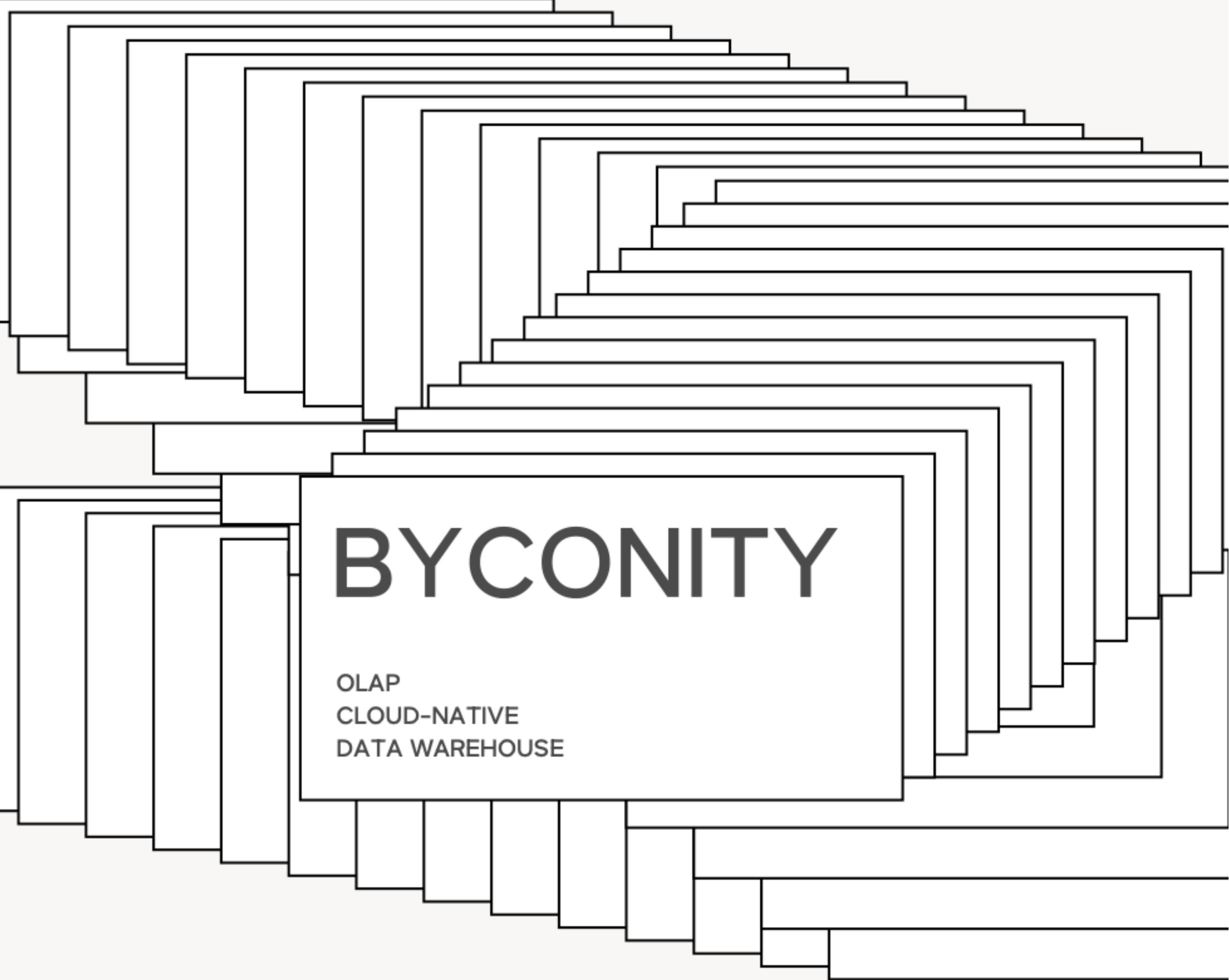
- 服务层 (Cloud Service)
  - MetaDate: FoundationDB/ByteKV
  - Server: 表元数据缓存、查询SQL解析、计划生成、调度和下发
  - Resource Manager: 服务发现、负载心跳检测
  - TSO: 全局唯一单调递增的时间戳
  - Daemon Manager: 调度和管理任务
- 计算组 (Virtual Warehouse, VW)
  - Worker: 查询片段的执行, 后台任务的执行、Local Disk Cache
  - 每个表可以设置默认的Read VW (查询) 和Write VW (导入和Merge)
- 存储层 (Cloud Storage)
  - 支持HDFS、S3



# 组件交互



# Agenda

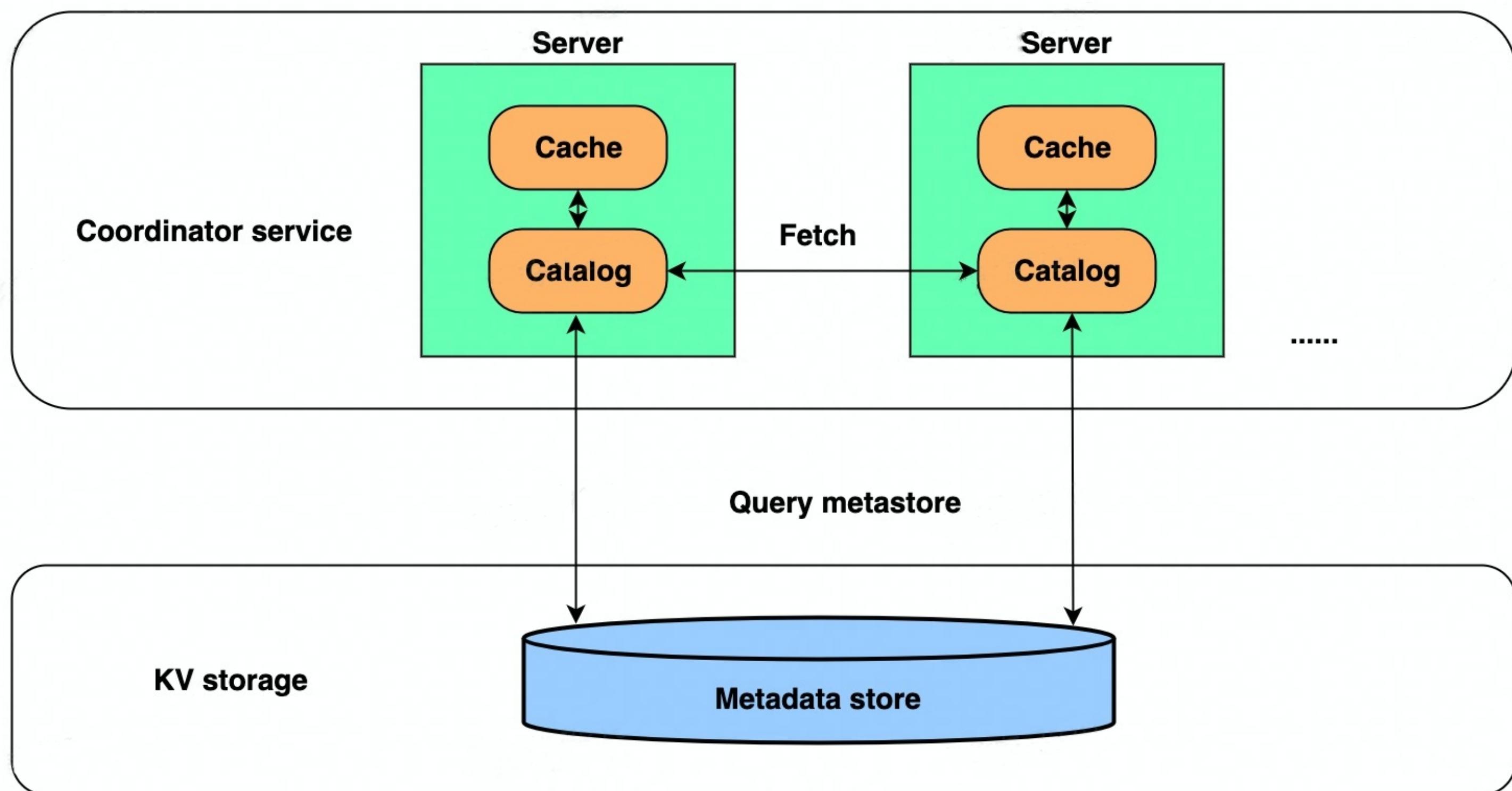
- 
- 01 背景
  - 02 整体架构和基础概念
  - 03 核心功能
  - 04 应用案例
  - 05 总结和展望

# 存算分离-设计考慮

- 需要统一的元信息管理系统
- 分布式文件系统大多数存在元信息管理压力问题
- 分布式统一存储系统大多不支持rewrite，一些对象存储系统甚至不支持append
- 分布式对象存储系统大多move代价都比较高
- io latency通常情况对比本地文件系统下都存在增加的情况

# 存算分离—统一元数据

- 管理Table/Part/  
Transcation等元信息
- 高效的part缓存管
- 拓扑管理



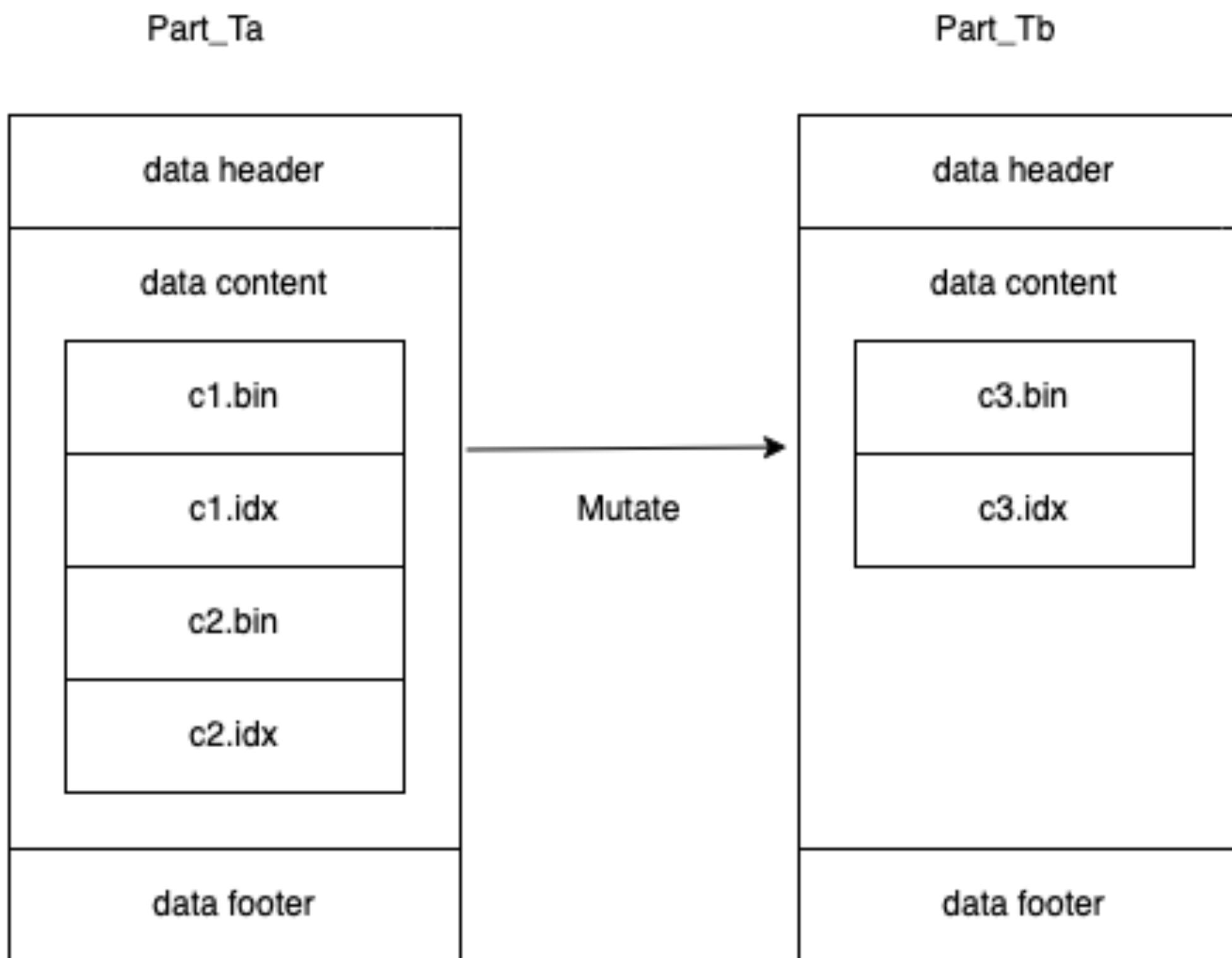
# 存算分离-存储结构

- 合并小文件，每个part所有数据存储在一个文件中
- 保持按列存储特性

```
* Data information will be stored in cloud storage(e.g. s3 hdfs) as one data file,  
* Which will not be updated once generated, only can be rewrited to new data file or be deleted.  
*  
* -----data header-----  
* magic_code(4 bytes)  
* version(8 bytes)  
* deleted(1 bytes)  
* reserved size(256 - 12 bytes)  
* -----data content-----  
* columns data files & idx files  
* primary_index  
* checksums  
* metainfo  
* -----data footer-----  
* primary_index offset(8 bytes)  
* primary_index size(8 bytes)  
* primary_index checksum(16 bytes)  
* checksums offset(8 bytes)  
* checksums size(8 bytes)  
* checksums checksum(16 bytes)  
* metainfo offset(8 bytes)  
* metainfo size(8 bytes)  
* metainfo checksum(16 bytes)  
* unique_key_index offset(8 bytes)  
* unique_key_index size(8 bytes)  
* unique_key_index checksum(16 bytes)  
* metainfo key (32 bytes)  
* -----  
*/
```

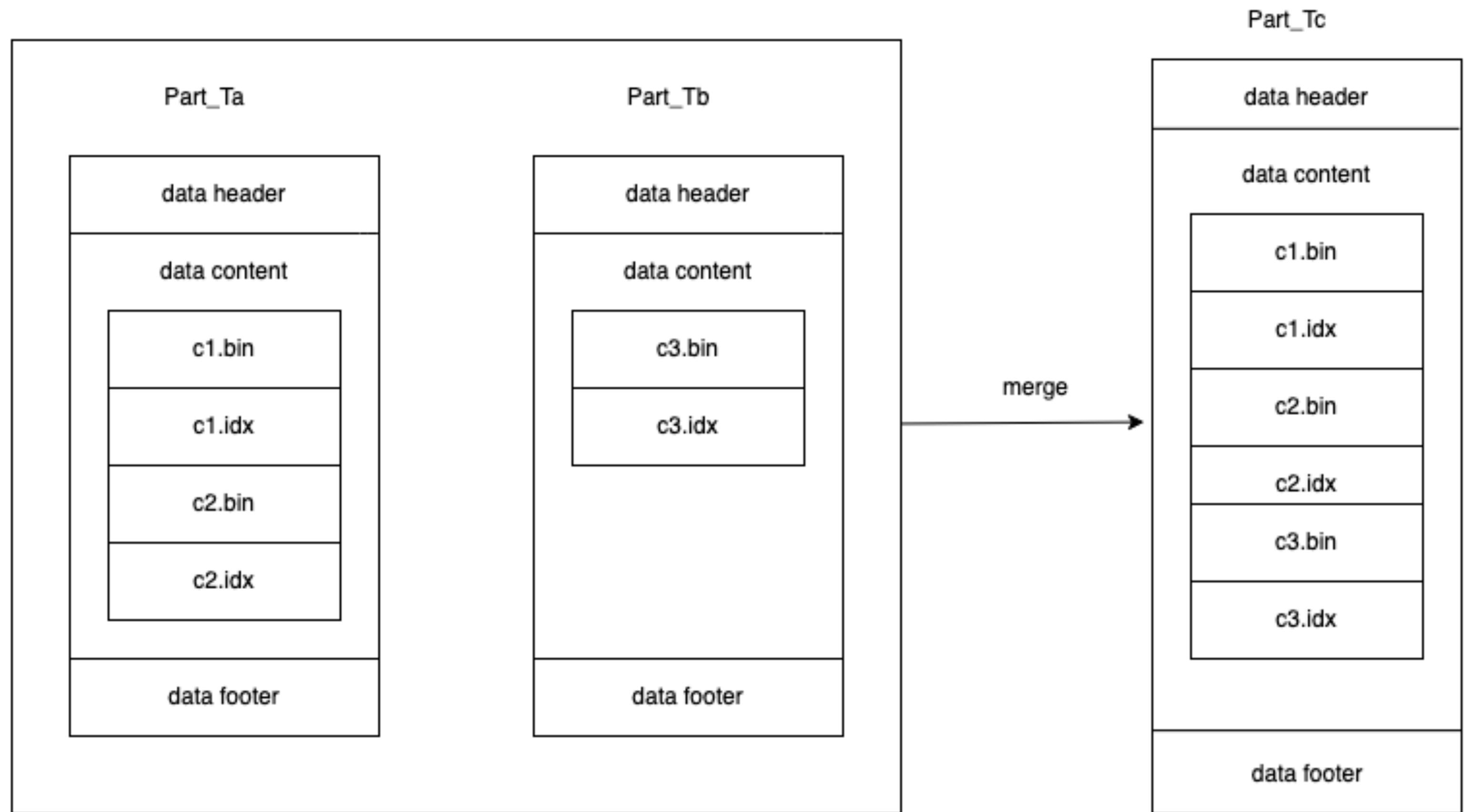
# 存算分离-数据变更

- 文件生成后不再变动
- delta part
- part chain



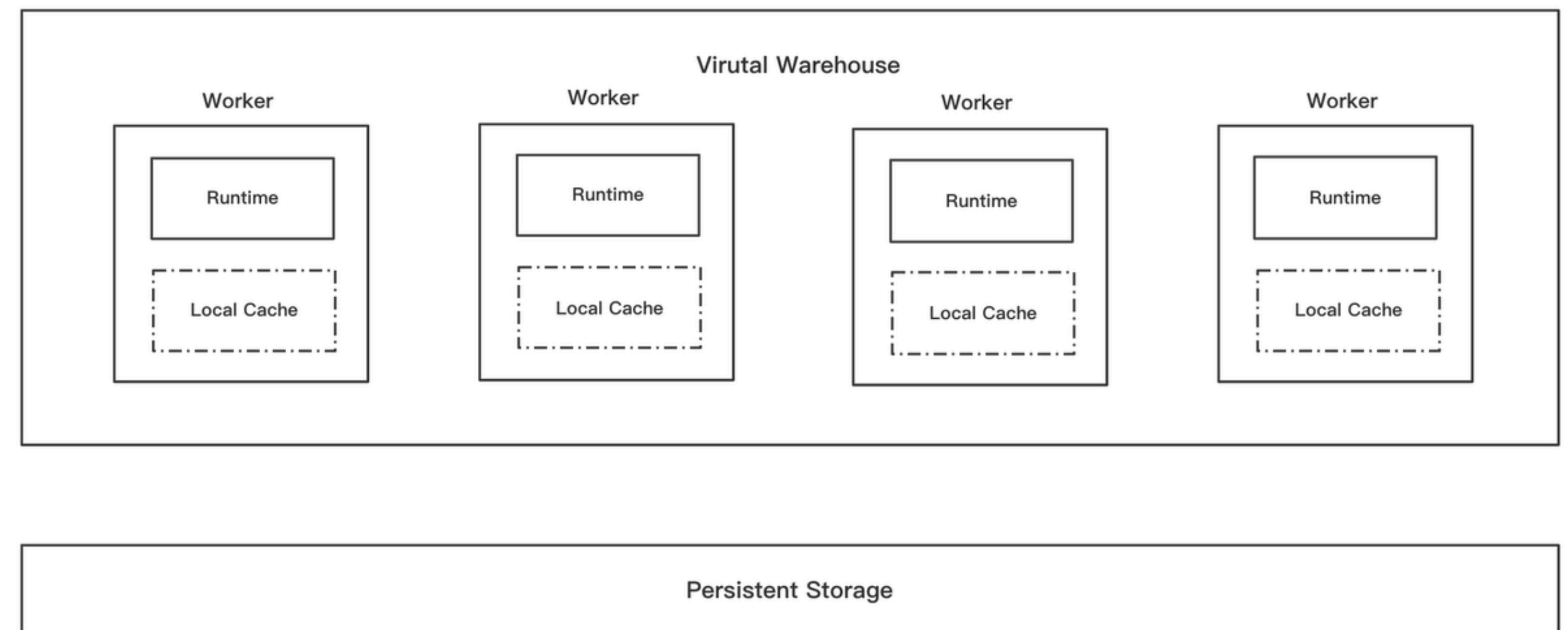
# 存算分离-数据合并

- 异步merge
- old parts通过gc清理



# 存算分离-数据缓存

- 一致性hash分配parts
- 热数据worker节点自动缓存
- 改进bucket-lru算法
- 避免数据reshuffling



# ByConity事务

- 隐式（开源）和显示事务（待开源）
- Read Committed 隔离级别，写不阻塞读
- 两阶段提交实现，支持海量数据的原子写入
- 具备灵活可控的并发控制的功能

# 事务实现-可见性判断

- 中心授时服务TSO(TimeStamp Oracle)
- Timestamp ordering
  - 创建事务：为事务分配开始时间 $t_s$
  - 提交事务：为事务分配提交时间 $t_c$
  - 可见性判断：对 $t_s$ 为TS的事务，能读到所有已提交且 $t_c < TS$ 的事务数据



# ByConity唯一键

- 数据源(如Kafka)包含重复数据，如何保障数仓表的数据质量？
- 业务数据流包含行更新，如何高效实时同步和分析？
- 如何提高RDBMS->数仓的同步时效性，并支持高效分析？

解决方案：唯一键 + Upsert

# 唯一键简介

- 面向读取操作进行优化
- 支持唯一键与排序键不同
- 支持基于版本字段的比较
- 支持行删除
- 支持表级别唯一键

```
CREATE TABLE t
(`d` Date, `id` Int32, `name` String)
ENGINE = CnchMergeTree -- cloud native table engine
PARTITION BY d ORDER BY name
UNIQUE KEY id;

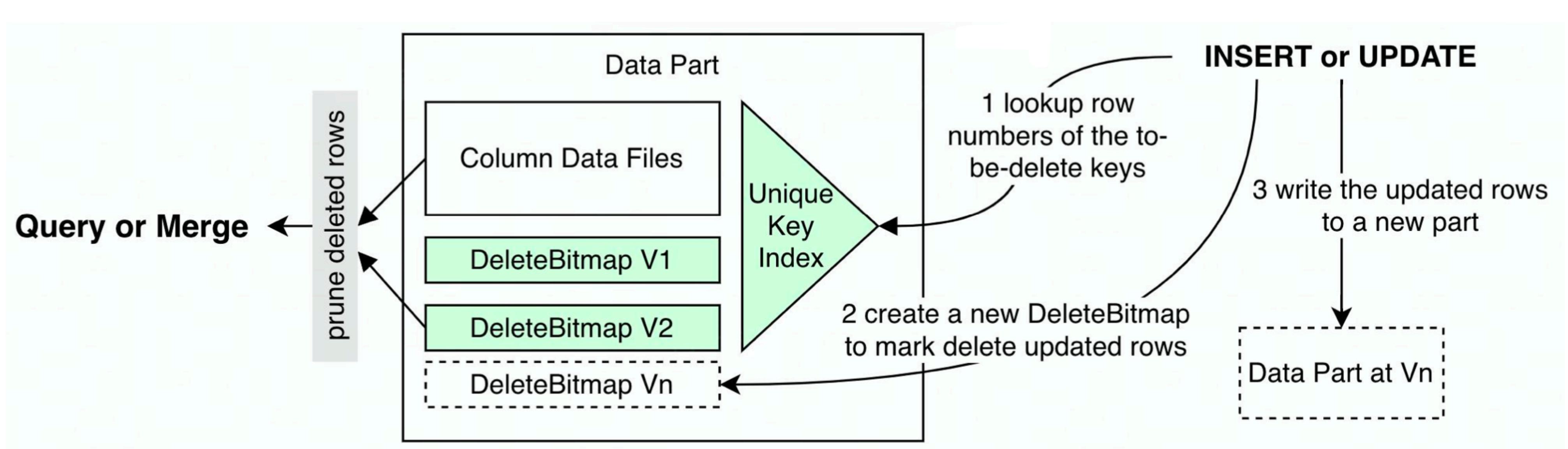
insert into t values (today(),1,'A'), (today(),2,'B');

select * from t order by id;
"2022-03-03",1,"A"
"2022-03-03",2,"B"

-- UPSERT semantics
insert into t values (today(),2,'B1'), (today(),3,'C');

select * from t order by id;
"2022-03-03",1,"A"
"2022-03-03",2,"B1"
"2022-03-03",3,"C"
```

# 唯一键实现原理



## DeleteBitmap

- 记录已删除数据的行号
- RoaringBitmap压缩
- 多版本支持

## UniqueKeyIndex

- 唯一键值到行号的映射
- 持久化索引
- 创建后不可变

## Dedup

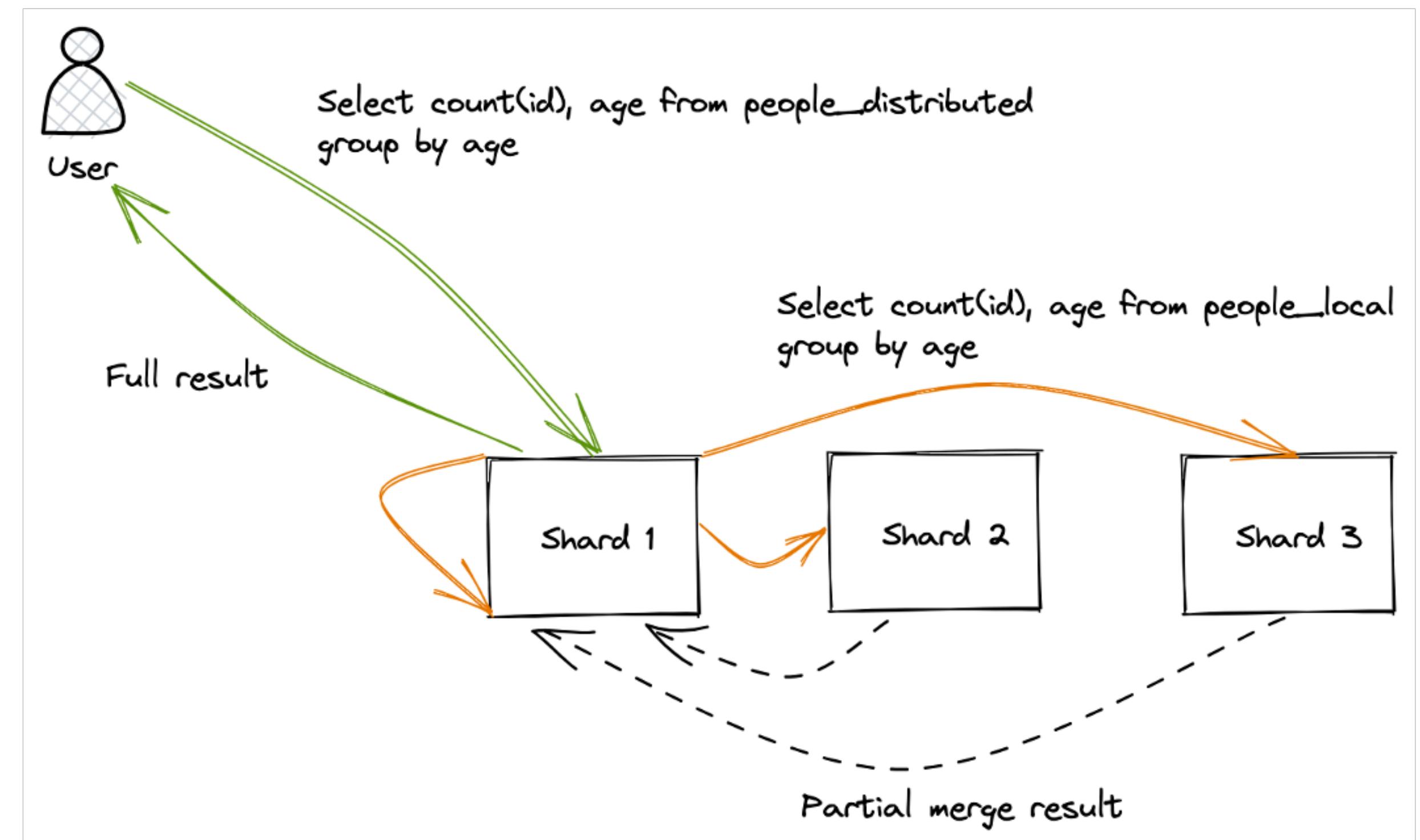
- 移除Part间重复的唯一键
- 并发控制: 分区/表锁

# 查询引擎优化-Clickhouse现状

## 分布式两阶段执行(Scatter-Gather模式)

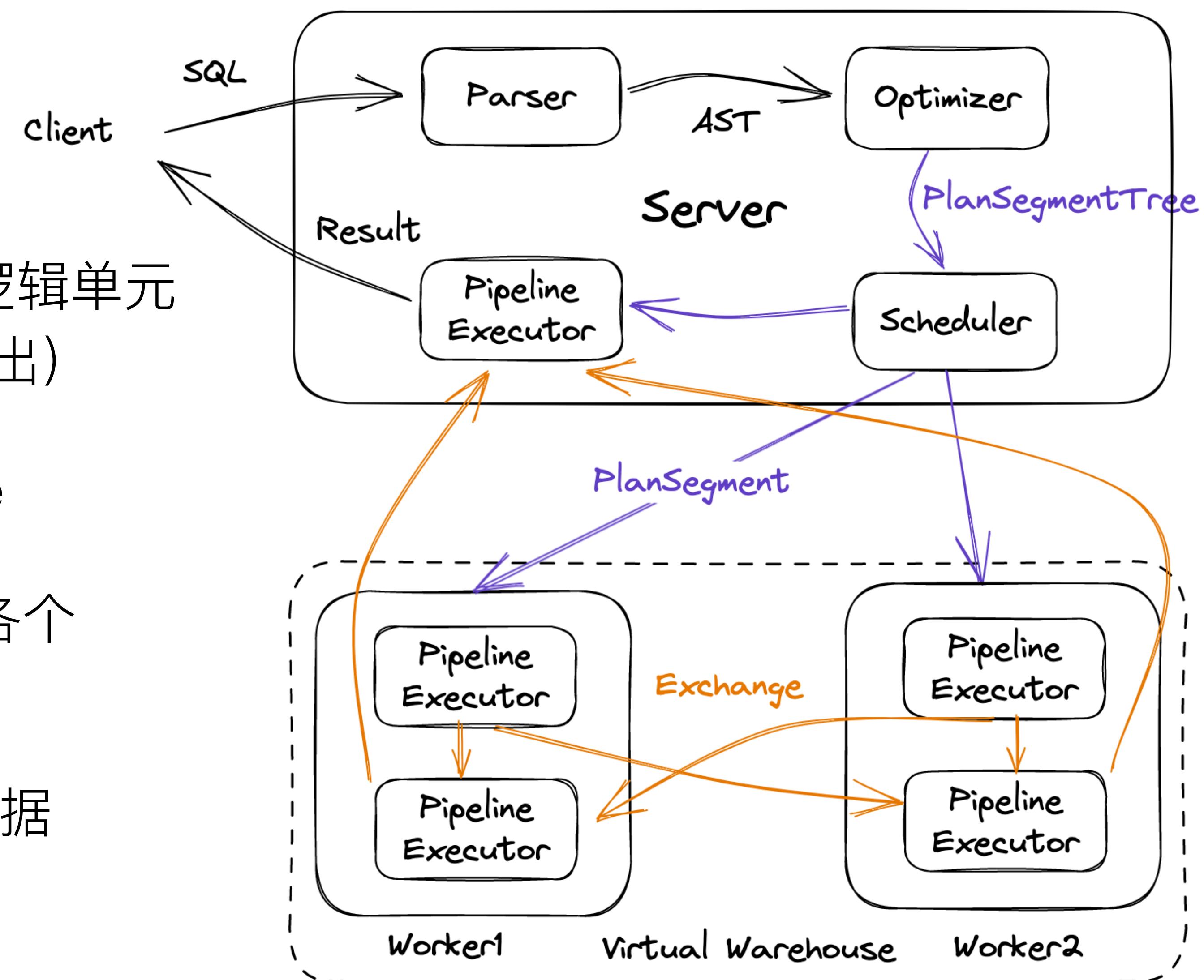
- 优点：简单高效
- 第二阶段的计算在单个节点，容易成为瓶颈
- 不支持持 Shuffle, Hash Join 右表大小不能超出单机内存的限制
- 子查询下发的模式表达能力受限，一些优化信息不能下发，灵活度比较低，同时worker需要重复解析
- 查询优化主要基于规则，优化手段有限，重要的优化如join reorder没办法支持

## 分布式两阶段执行(Scatter-Gather模式)



# 整体流程

- PlanSegment: 分布式执行计划逻辑单元 (QueryPlan + Exchange输入输出)
- Optimizer: 生成PlanSegmentTree
- Scheduler: 发送PlanSegment到各个 Worker
- Exchange: 在Pipeline之间传输数据



# 优化器-RBO

## - 基于 visitor 的改写框架

Top-Down / Bottom-Up 的方式对一个 QueryPlan 做改写，适合于带有上下文依赖的优化规则。

- Predicate Push Down
- Column Pruning

## - 基于 pattern-match 的改写框架

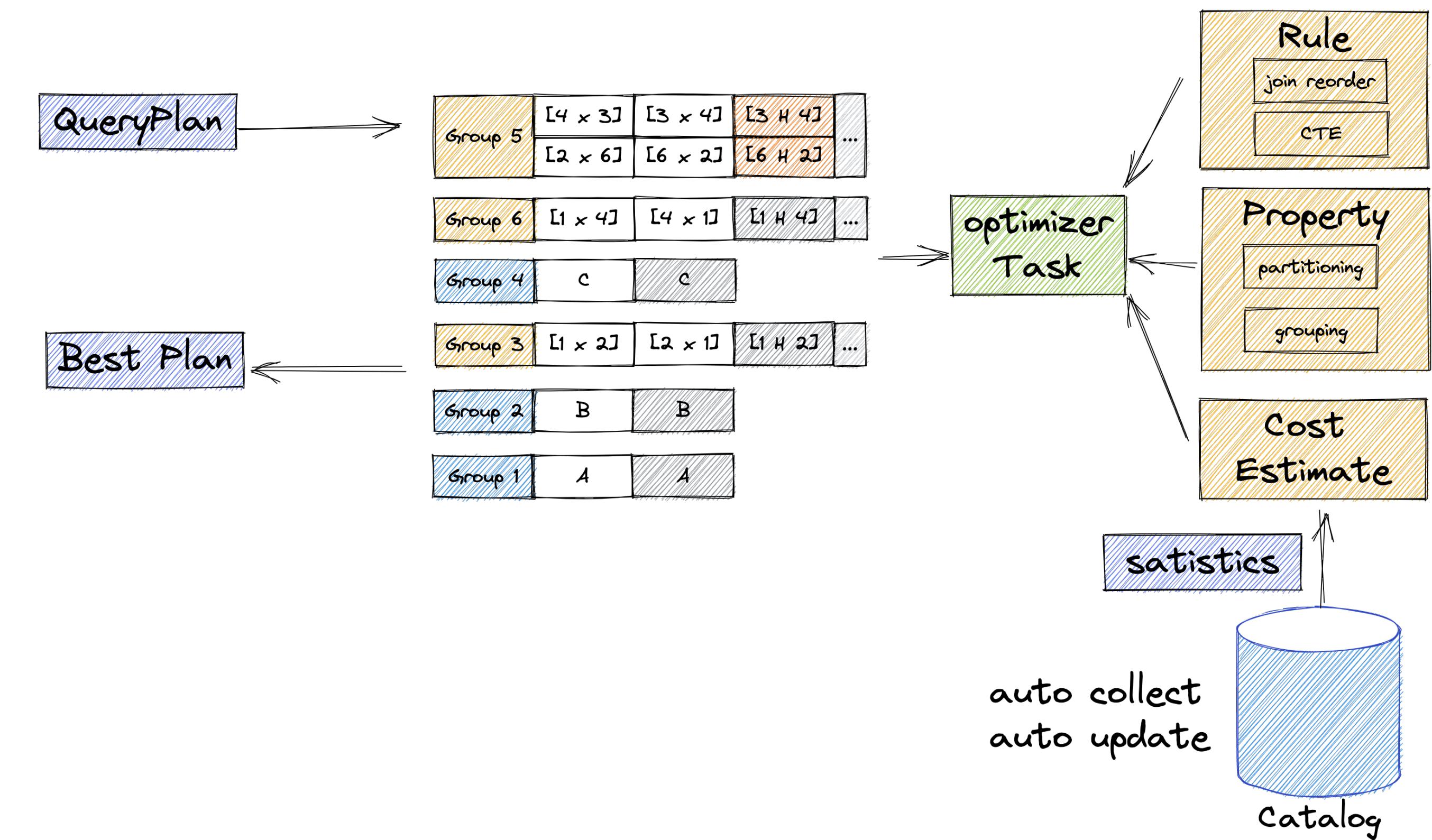
适合简单、通用的改写规则，例如对于两个连续的 Filter 做合并的动作。

- Merge Filter
- Inline Projection

“列裁剪、分区裁剪、表达式简化、子查询解关联、谓词下推、冗余算子消除、Outer-JOIN 转 INNER-JOIN、算子下推存储、分布式算子拆分等常见的启发式优化能力”

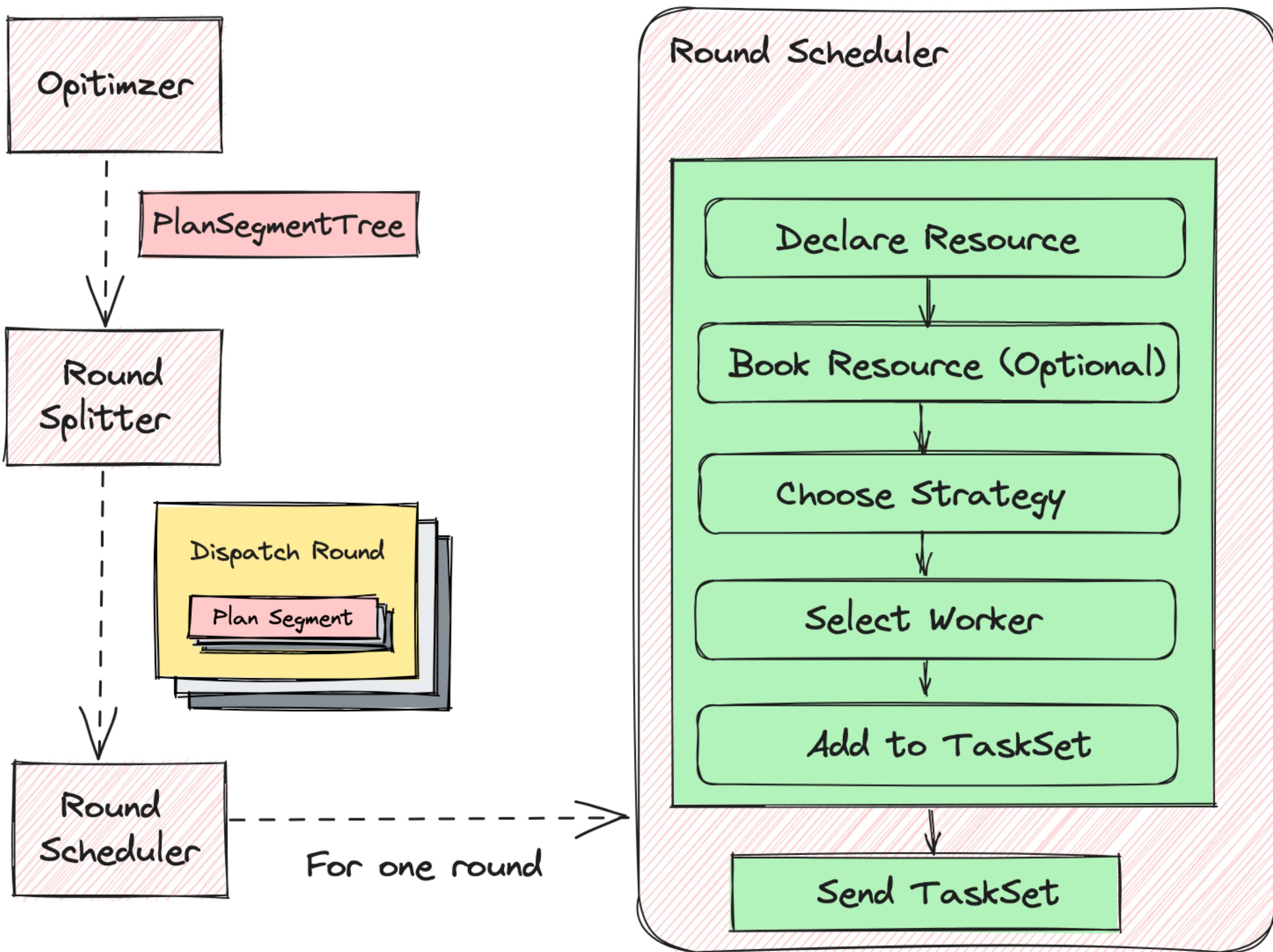
# 优化器-CBO

- Cascades搜索框架：遍历等价计划，并将所有等价计划存储在一个内存空间中，然后评估每种等价计划的代价，进而选出最优解
- Join Reorder 10表级别全量枚举，超过10表启发式搜索
- 分布式执行计划 属性传递，基于代价生成最优的分布式计划



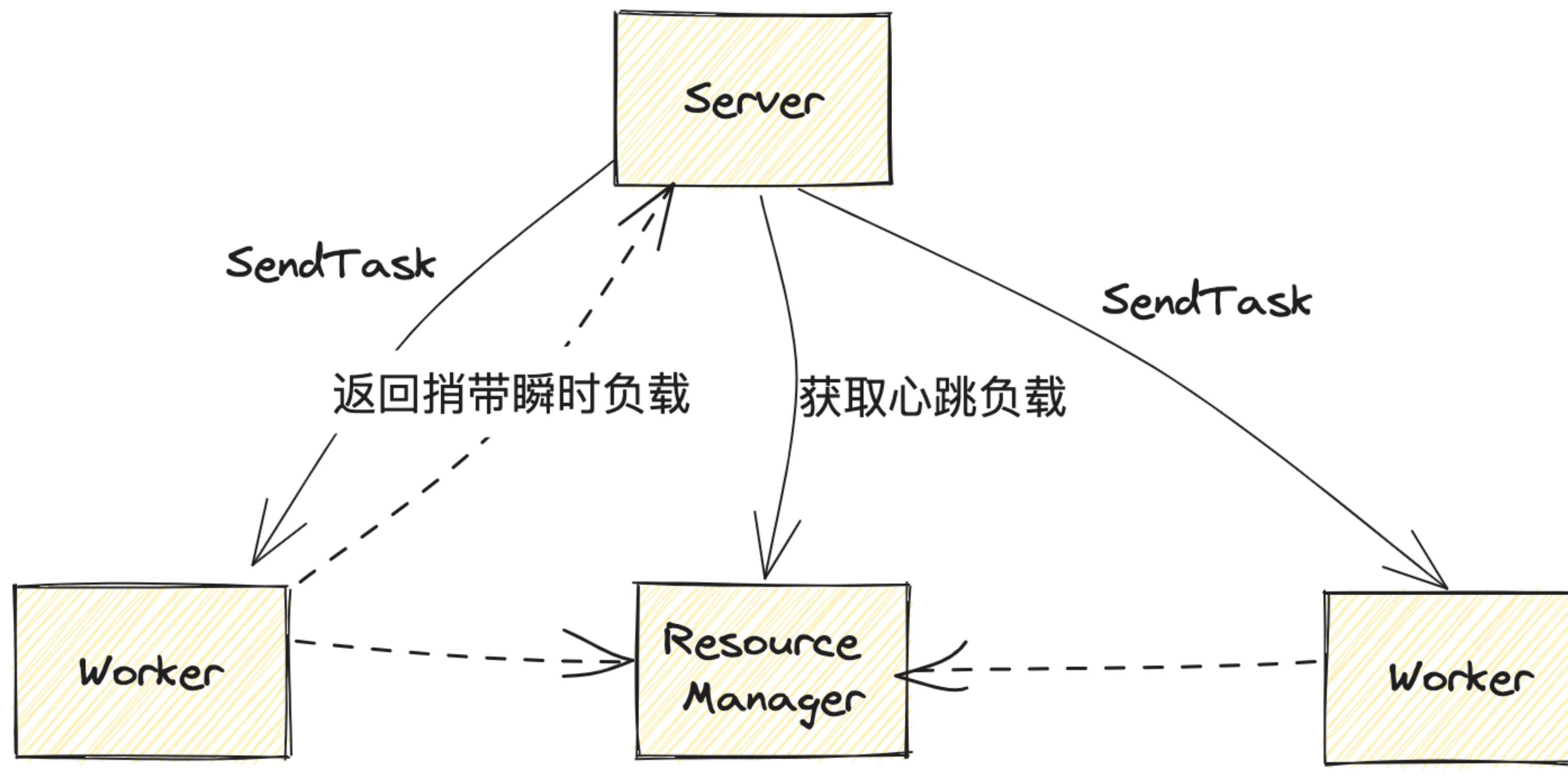
# 调度器

- Source/计算节点分离开调度
- Source考虑Cache亲和性，可裁剪
- 根据负载情况自适应调度（待开源）



# 调度器

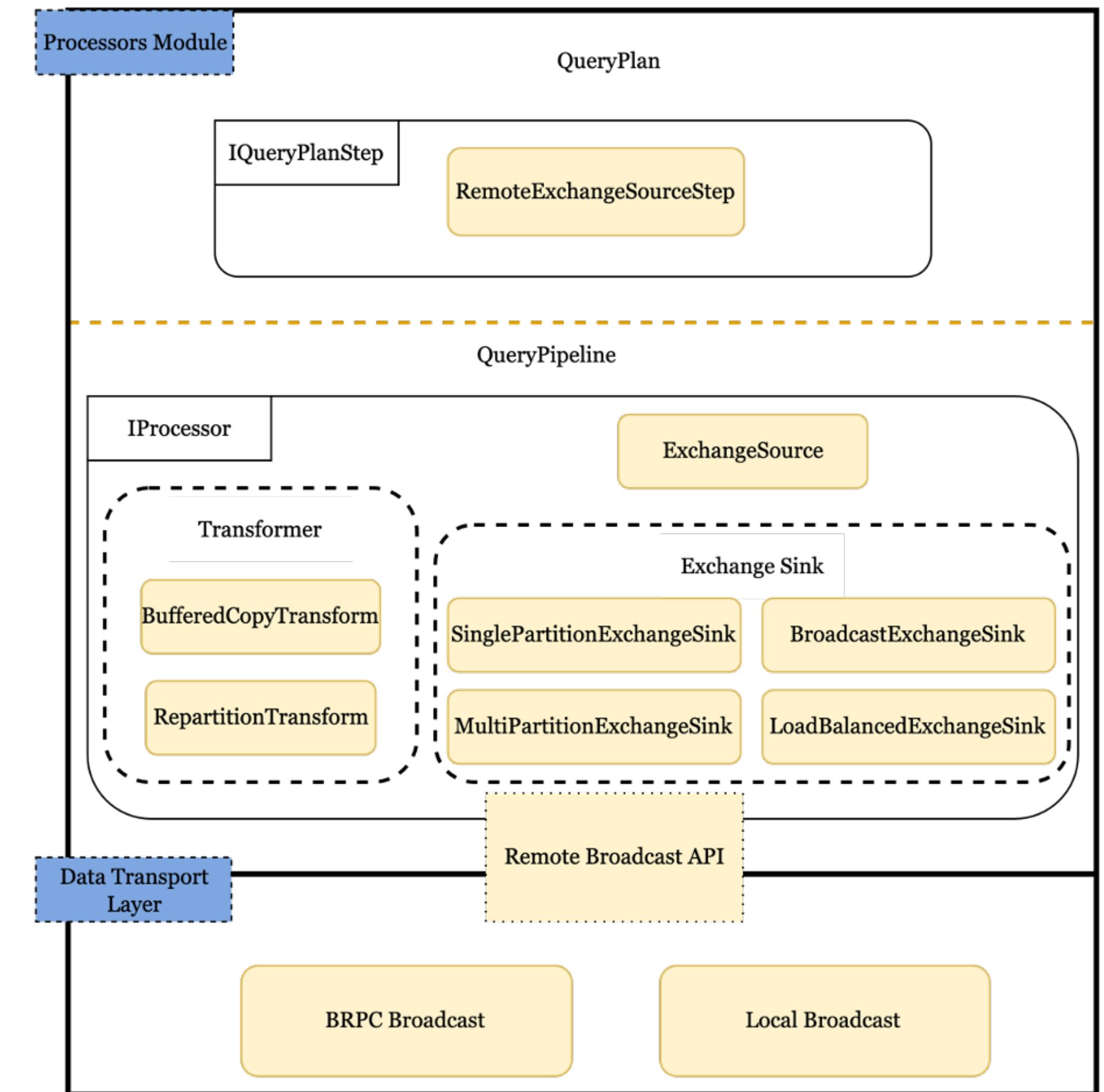
Server感知到的worker负载：  
心跳负载+瞬时负载+预占资源



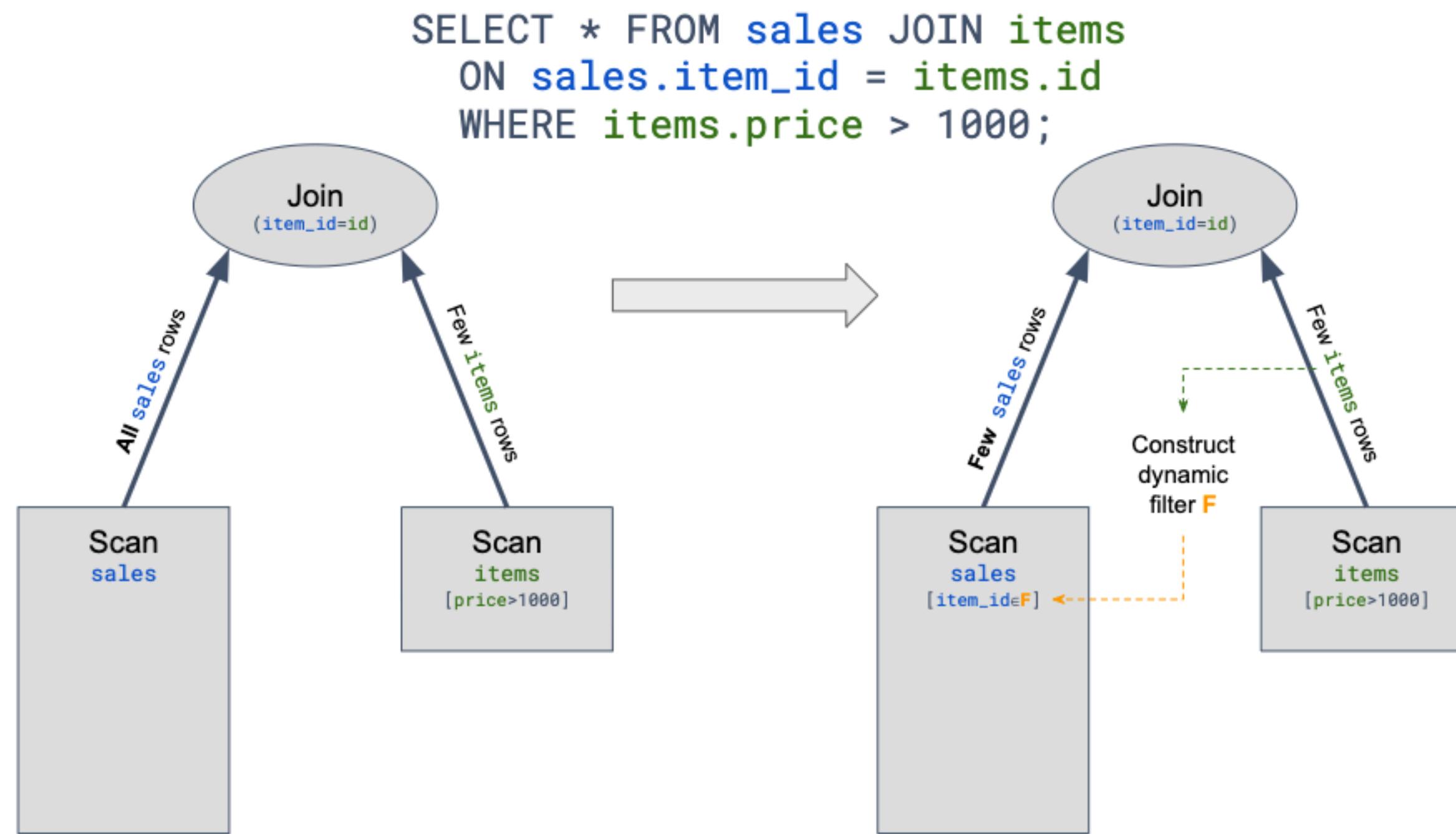
心跳负载 (CPU、内存、磁盘、查询数)

# Exchange

- 数据传输层
  - 同进程通过内存队列，无序列化、copy开销
  - 跨进程基于Brpc Stream RPC，支持保序、固定连接数复用、状态码传输、压缩等
- 算子层
  - 攒批
  - 单线程读取多个分区数据



# RuntimeFilter

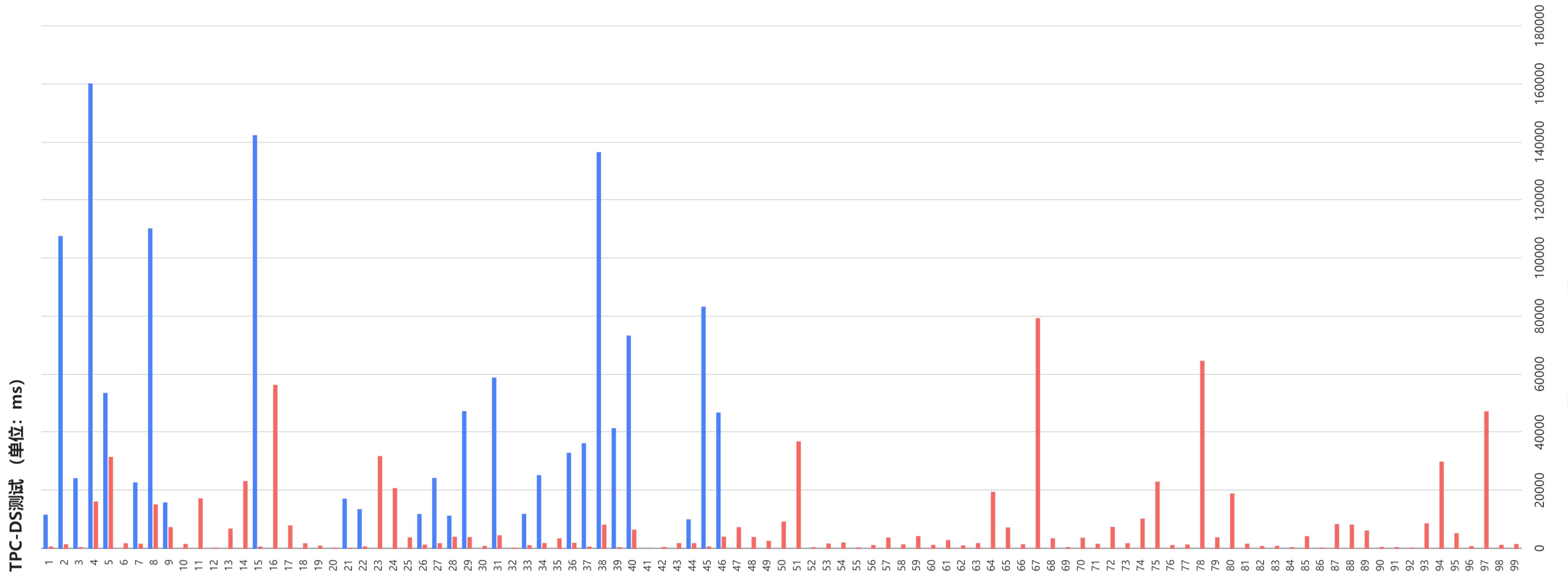


基于 Cost 决策是否生利用 Runtime Filter 的方式加速 Join 计算，并利用 Predicate 下推的能力下推 Runtime Filter 减少数据的扫描和计算。

# 其他优化

- 表级别低基数类型
- Windows算子并行化、 Partition TOP-N 下推
- Common Table Expression
- 重构多阶段模型下算子 (aggregation、join)
- Pipeline执行器协程化

# 优化效果



两阶段模式仅能完成 26 个 SQL 的查询。而多阶段执行和优化器完成后，能够完整跑完 TPC-DS 的全部 99 个 SQL，并且性能也得到了极大的提升。

# Agenda

- 
- 01 背景
  - 02 整体架构和基础概念
  - 03 核心功能
  - 04 应用案例
  - 05 总结和展望

# 数据洞察简介

支持千亿级别数据自助分析的一站式数据分析与协作平台。从数据接入、数据整合，到查询、分析，最终以数据门户、数字大屏、管理驾驶舱的可视化形态呈现给业务用户。



Demo大屏

# 数据洞察挑战

## 超大规模MPP计算组

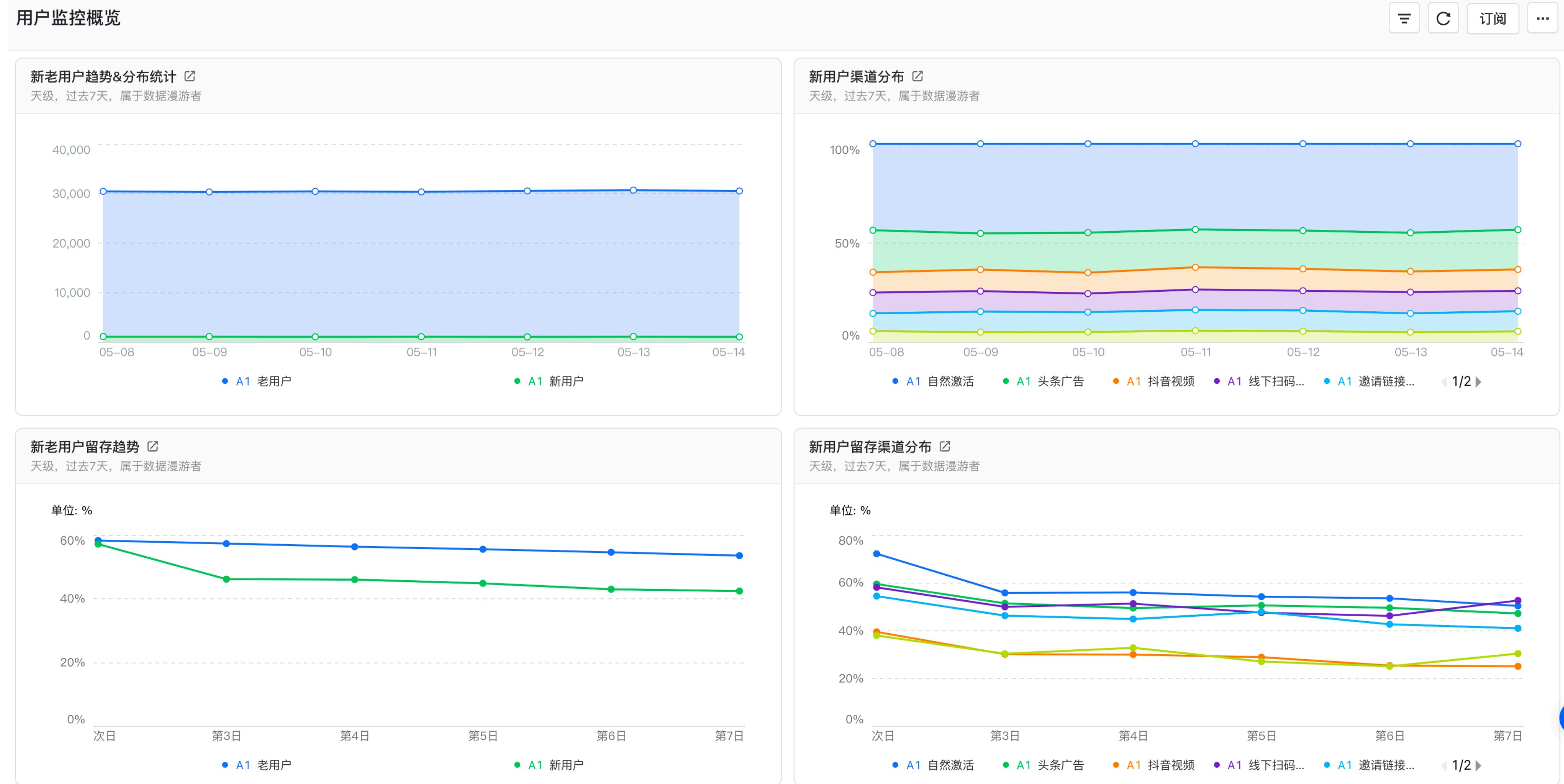
- 慢节点、单节点故障影响大（自适应调度）
- 计划序列化、part下发、调度慢（并行）
- 并发度不好设置（按VW调整、基于历史）
- 大查询影响大（限制查询复杂度、资源限制）

## 数据倾斜

- Part数目< Worker，某些Worker负载高（优化Part分配）
- 聚合计算没有分组键，或者分组键基数太低，聚合没办法并行（优化计划）

# 增长分析简介

一站式用户分析与运营平台



# 增长分析挑战

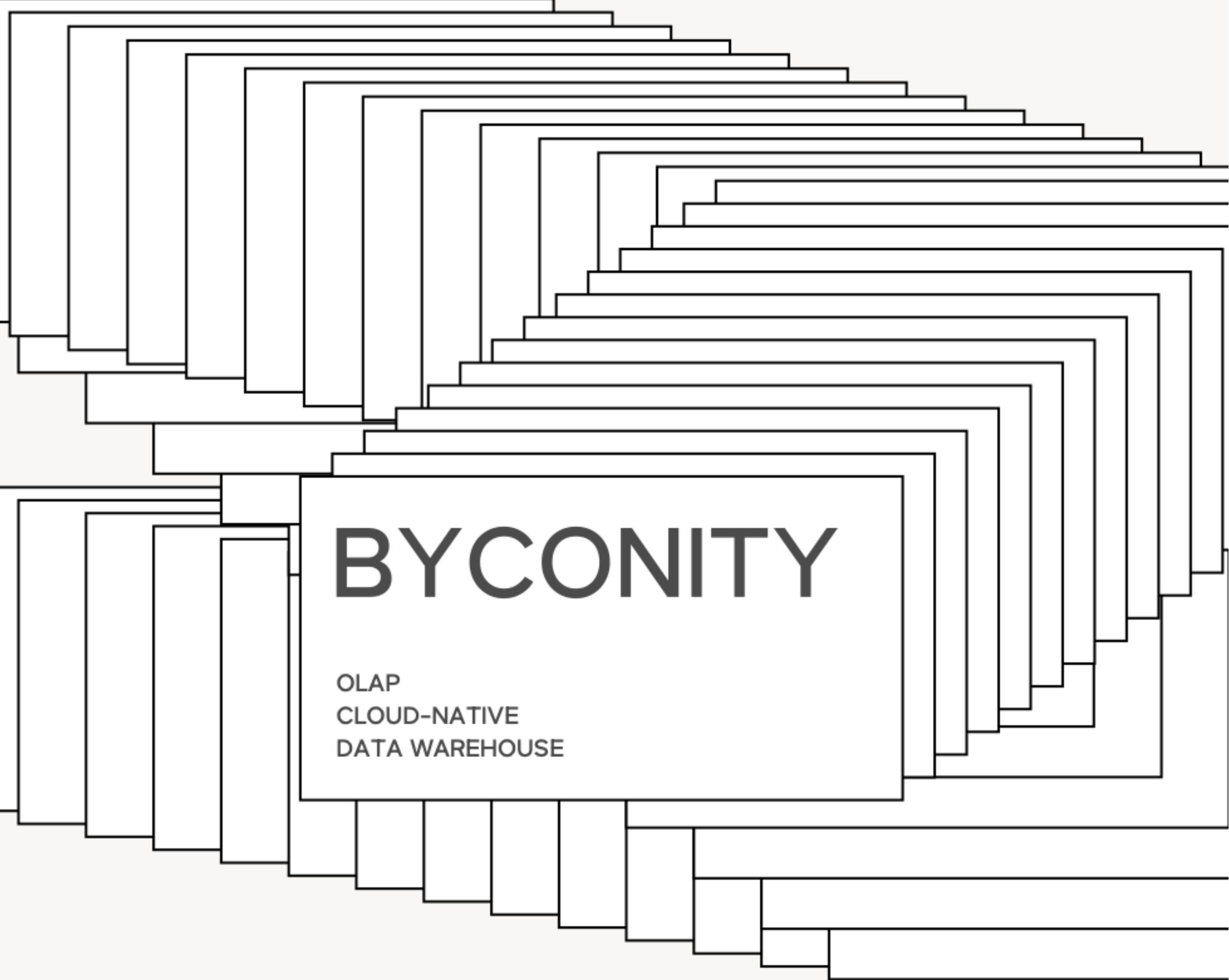
## 数据量巨大&实时摄入

- 消费堆积（优化写入性能、流式消费性能）
- 实时表Cache命中率低（Cache Preload）
- 查询慢（大表按user\_id进行bucket化）
- 数据冗余、存储成本高（存储服务化，减少）
- 单表千万级别Part，内存高
- 存储量大（使用低基数类型，降低存储）

## 非结构化数据类型

- Array类型性能差（Array类型支持BitmapIndex索引）
- Map类型性能差（支持隐式列Map类型）

# Agenda

- 
- The background features a large, stylized staircase graphic composed of many thin black lines. Superimposed on the middle section of the stairs is a white rectangular box containing the text "BYCONITY" in a bold, sans-serif font. Below "BYCONITY", in a smaller font, are the words "OLAP", "CLOUD-NATIVE", and "DATA WAREHOUSE".
- 01 背景
  - 02 整体架构和基础概念
  - 03 核心功能
  - 04 应用案例
  - 05 总结和展望

# 展望—今年规划

- 存储：对象存储支持、分布式DiskCache、数据湖外表支持
- 索引：Space-filling curves、倒排索引、JSON数据索引、自动推荐
- 性能：统计信息自动收集、查询结果Cache、Part分配优化、查询调度优化
- 事务：乐观事务支持、原生数据湖协议/引擎支持
- 企业级特性：FGAC、备份还原、加密

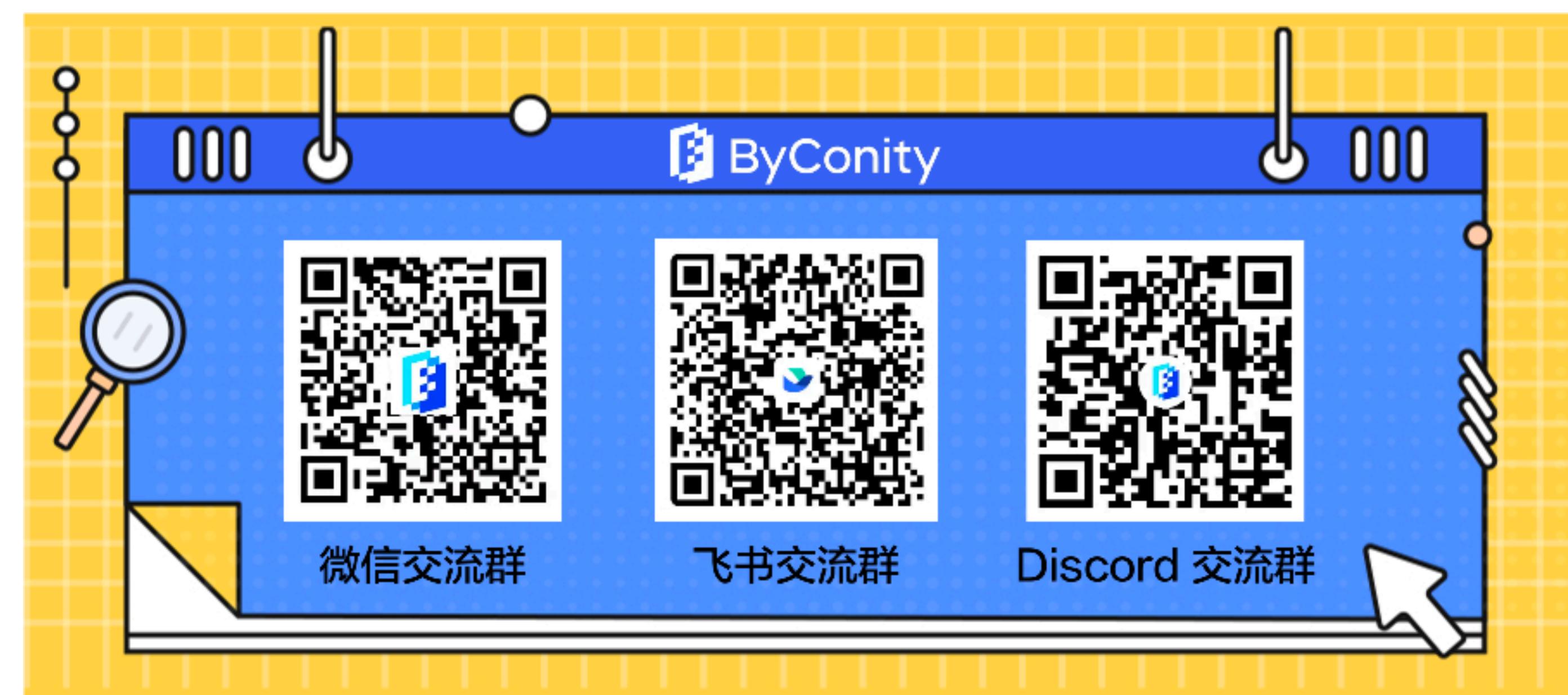
# 展望-远景

- 混合负载支持，从轻载数仓到重载数仓演进，Bubble Execution、Exchange Spill
- MPP查询容错能力构建，支持超大规模集群
- NUMA aware
- 自动扩缩容

理想中的数仓系统是什么样？

# 总结

- ByConity的背景和整体架构
- 核心架构和功能：存算分离、事务实现、一键、优化器、多阶段执行
- 介绍了增长分析和数据洞察场景的挑战和优化测试
- 分享了近期规划和远景



# THANKS

---

软件正在重新定义世界

Software Is Redefining The World