QCon 全球软件开发大会
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

BEIJING 2018

《SnappyData在美团酒店实时数据分析中的应用》

演讲 / 焦向

主办方 Geekbang 极客邦科技 InfoQ

极客时间
重拾极客精神·提升技术认知

注册立享
30元
新人红包

基于实践经验总结和提炼的品牌专栏
尽在【极客时间】

重拾极客时间，提升技术认知

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE

全球技术领导力峰会

通往**年薪百万**的CTO的路上，

如何打造自己的技术**领导**力？

扫描二维码了解详情

This talk is ‘opinionated’

- How so?
  - o 特定业务问题出发找方案
  - o 基于自己总结的方法论

- Why?
  - 分布式存储及业务出身

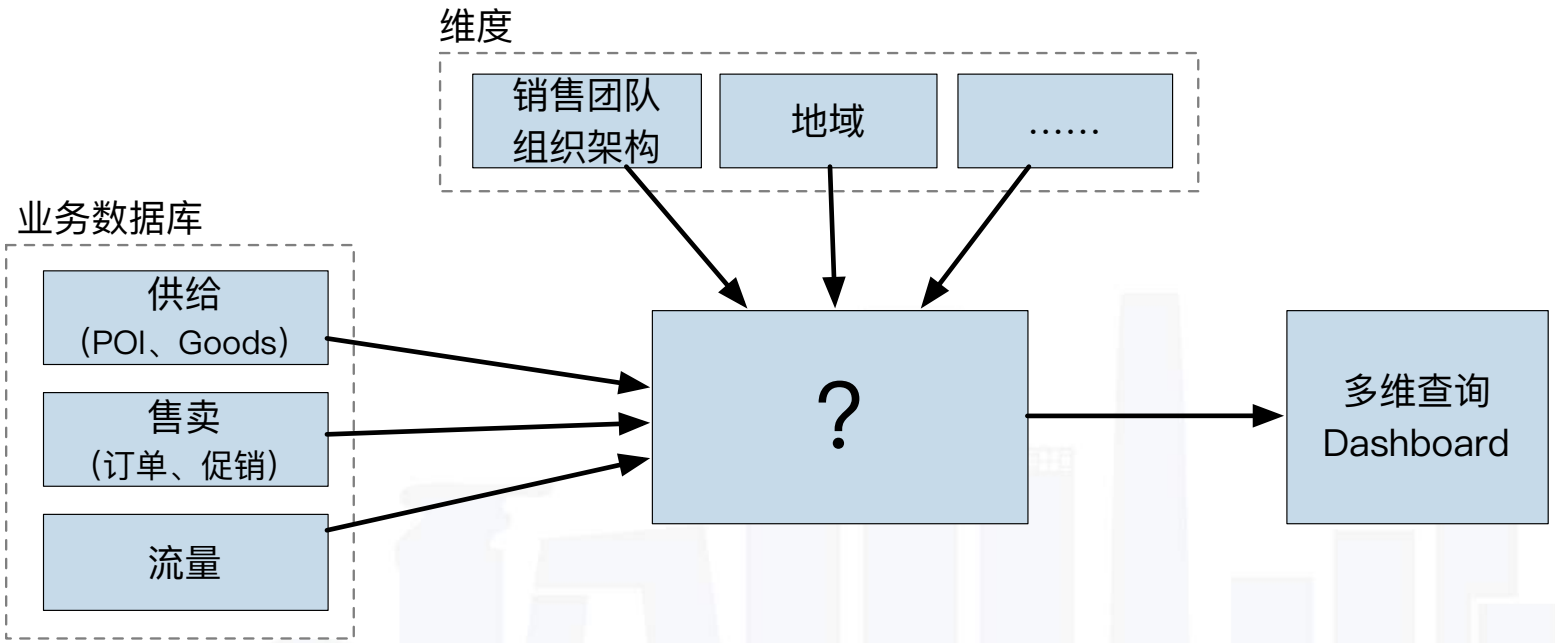
业务

交互式报表、实时情报分析

- 交互式报表（Interactive Reporting）
  - 实时数仓
- 实时情报分析（Intelligence analysis）
  - 类似日志流场景，但复杂得多

交互式报表（Interactive Reporting）

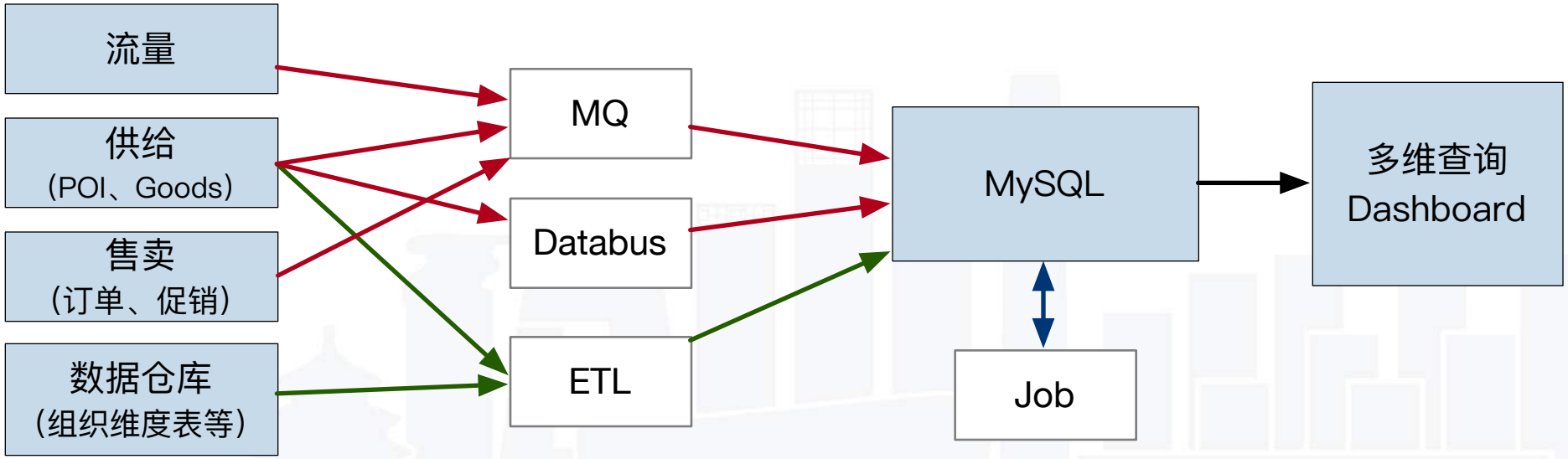
- 供给侧、售卖侧、流量数据多维度查询



- 数据新鲜度
- 低延迟（<3s）
- 研发效率

交互式报表（Interactive Reporting）

- 初期
  - 基于MySQL，人肉lambda架构
  - 以支持业务的思路实现

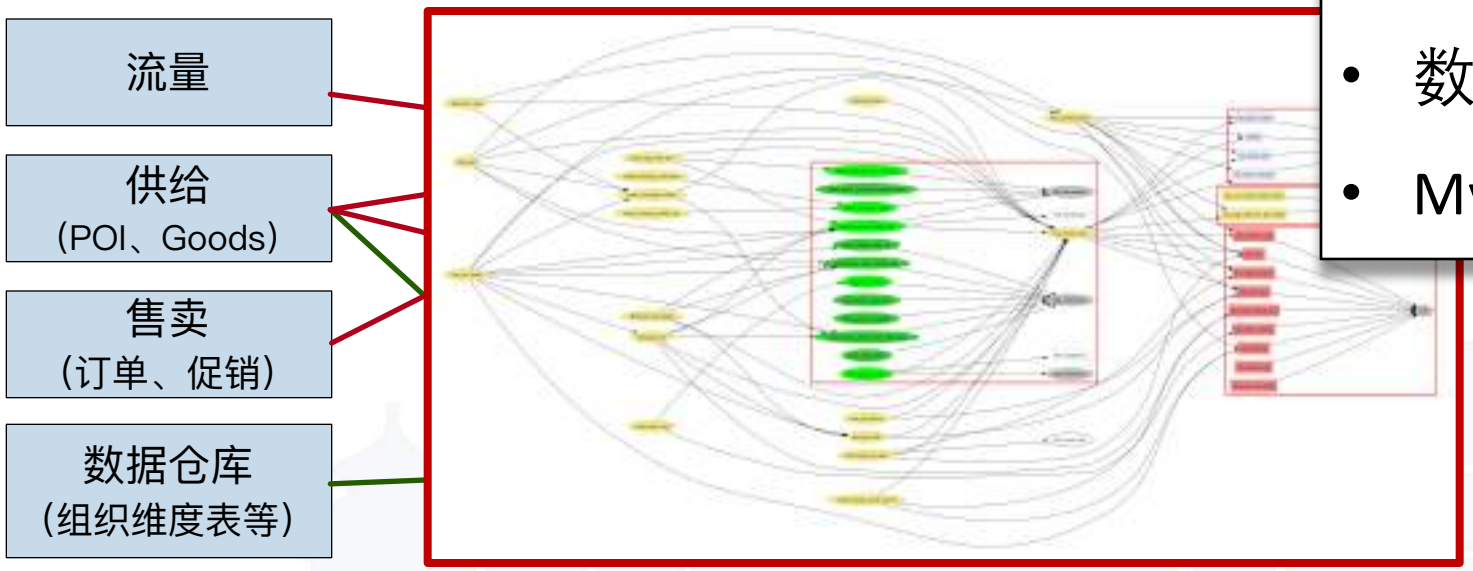


交互式报表（Interactive Reporting）

- 初期
  - 基于MySQL，人肉lambda架构
  - 以支持业务的思路实现

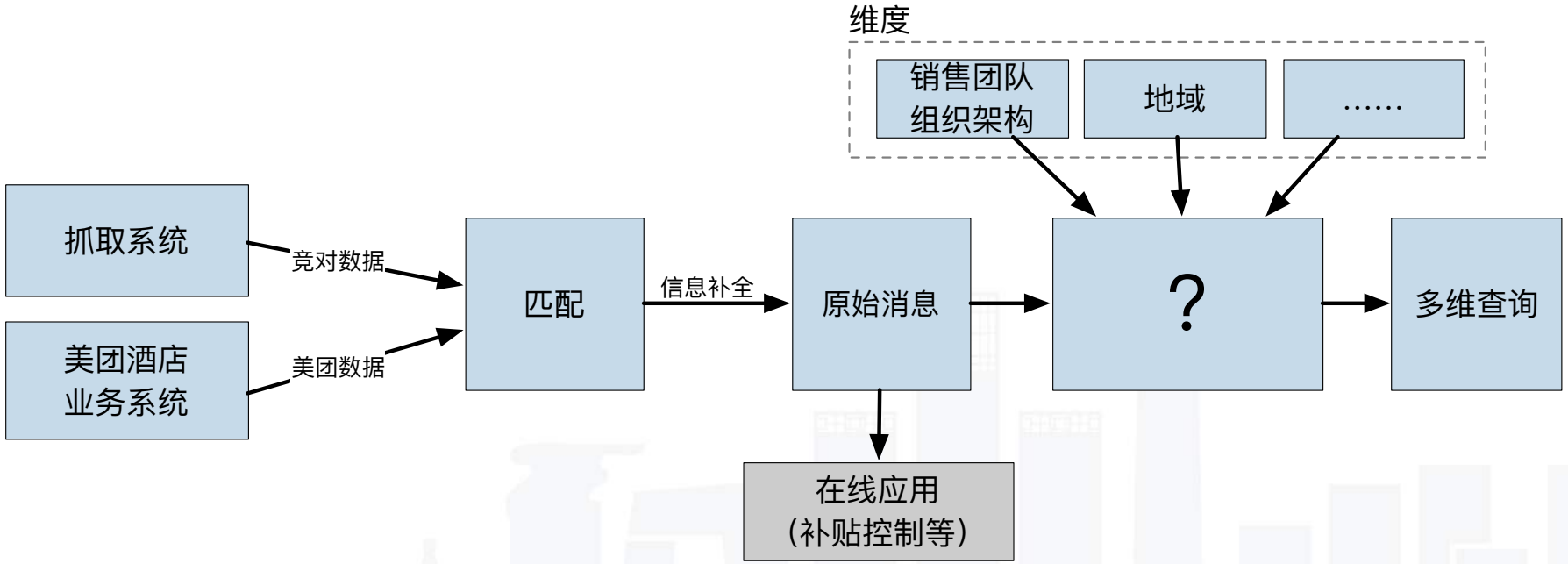
逐渐失控

- 指标一致性
- 可维护性
- 数据准确度
- MySQL容量、性能瓶颈



情报分析（Intelligence analysis）

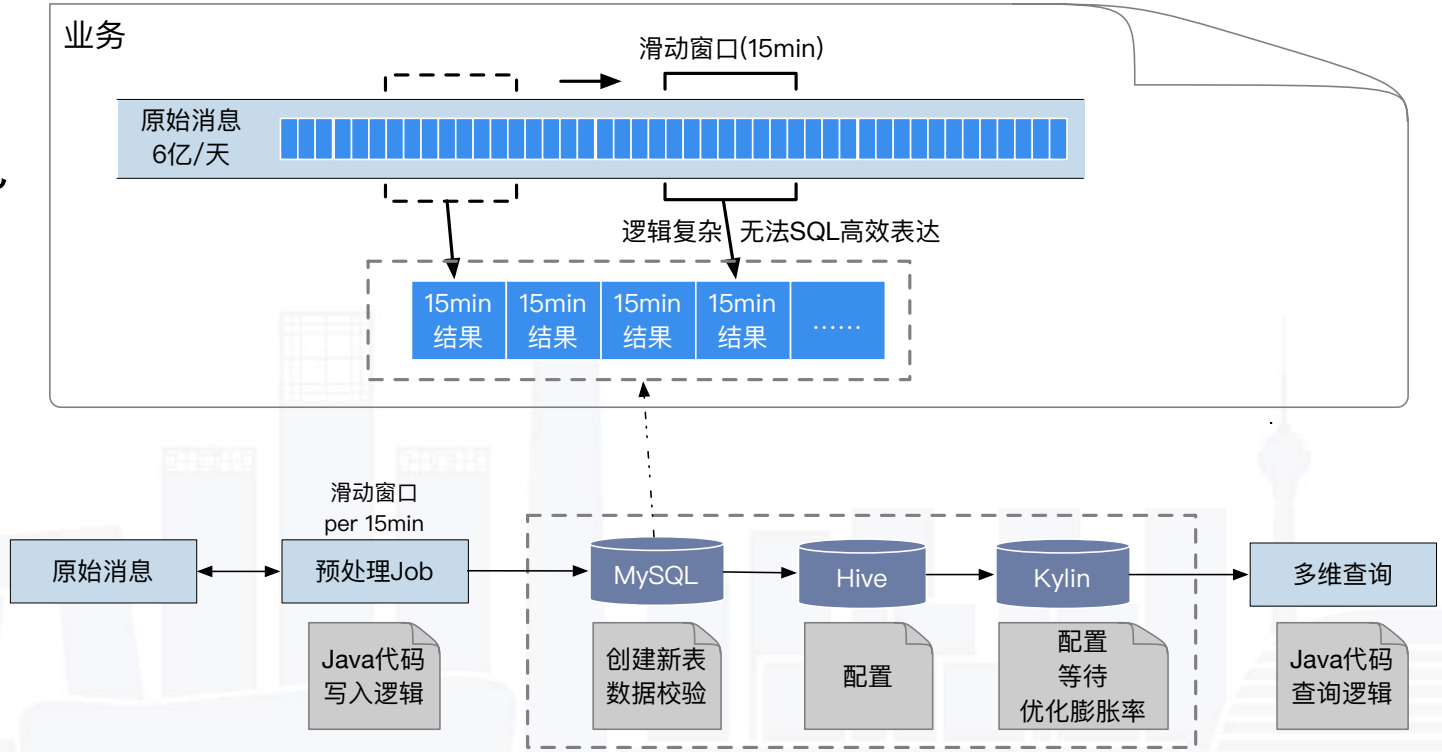
- 对比美团&友商数据，实时分析战况，自动跟进



- 原始消息6~120亿/天
- 8大类维度，120亿种组合
- 实时+30天历史
- 低延迟(<3s)

情报分析（Intelligence analysis）

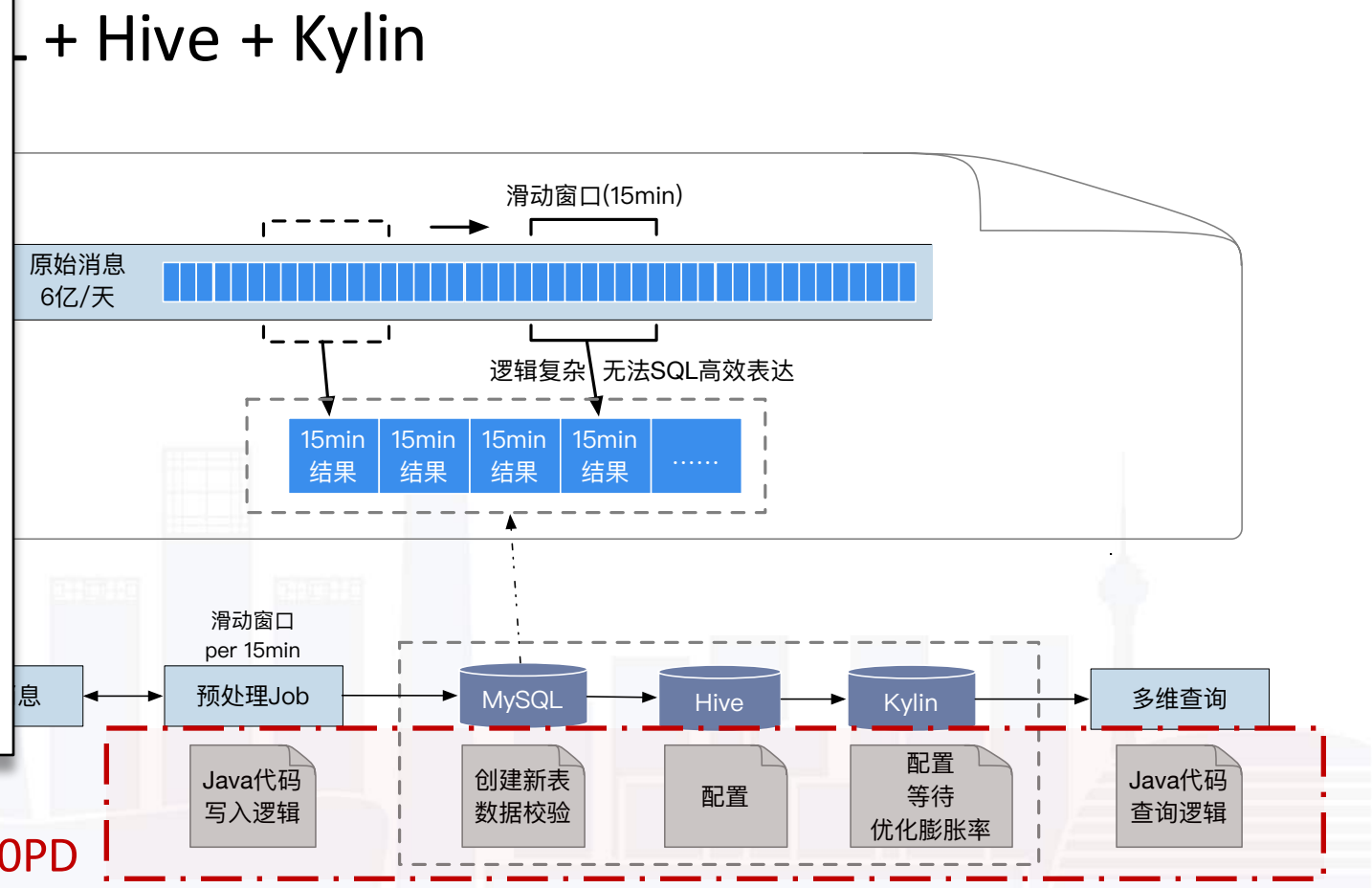
- 从批处理开始，Job + MySQL + Hive + Kylin
- 15min滑动窗口
  - 定时任务
  - 计算逻辑复杂，Java实现
  - 多维查询基于该结果



情报分析（Intelligence analysis）

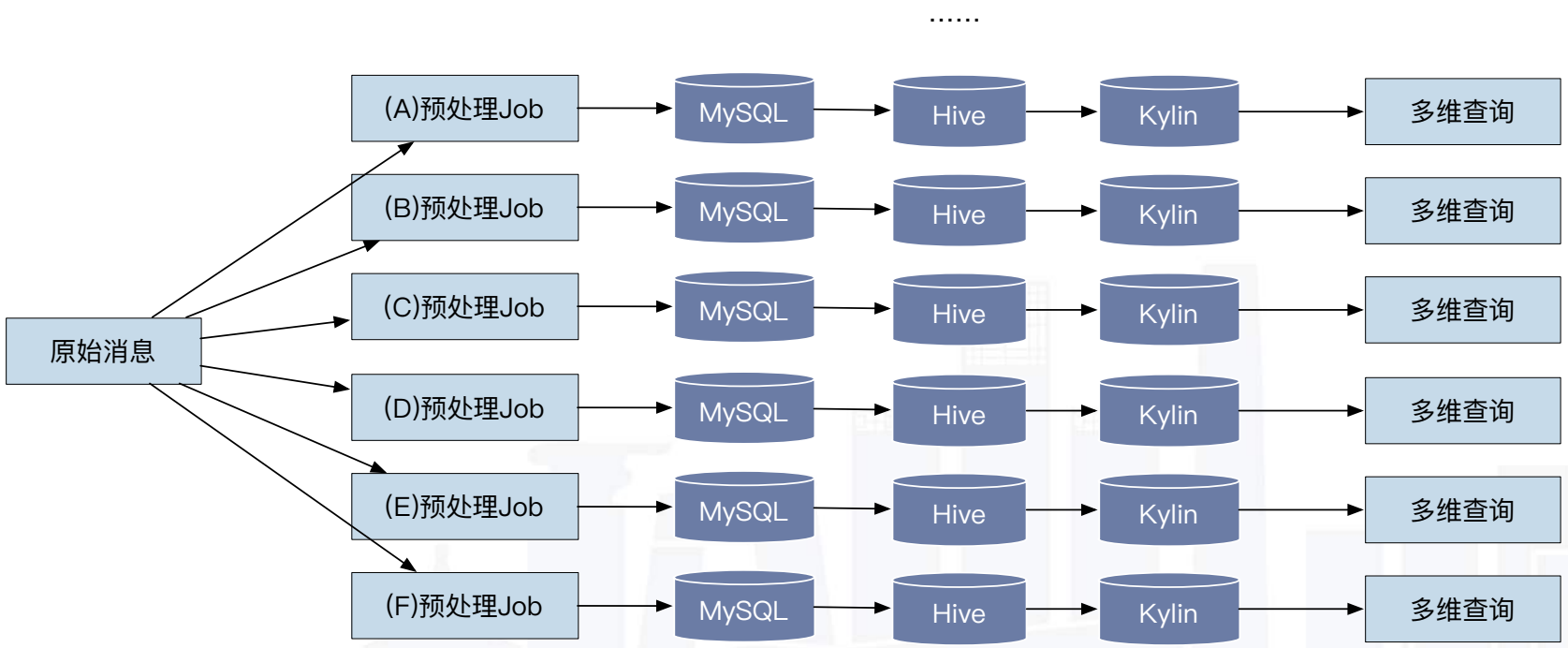
问题

- 研发效率低
  - 链条长，调试周期长，大量等待
- 需求变化频繁
  - 流程相似，体力活
  - 浪费资源，每个需求一套流程
- 无法支持实时多维分析



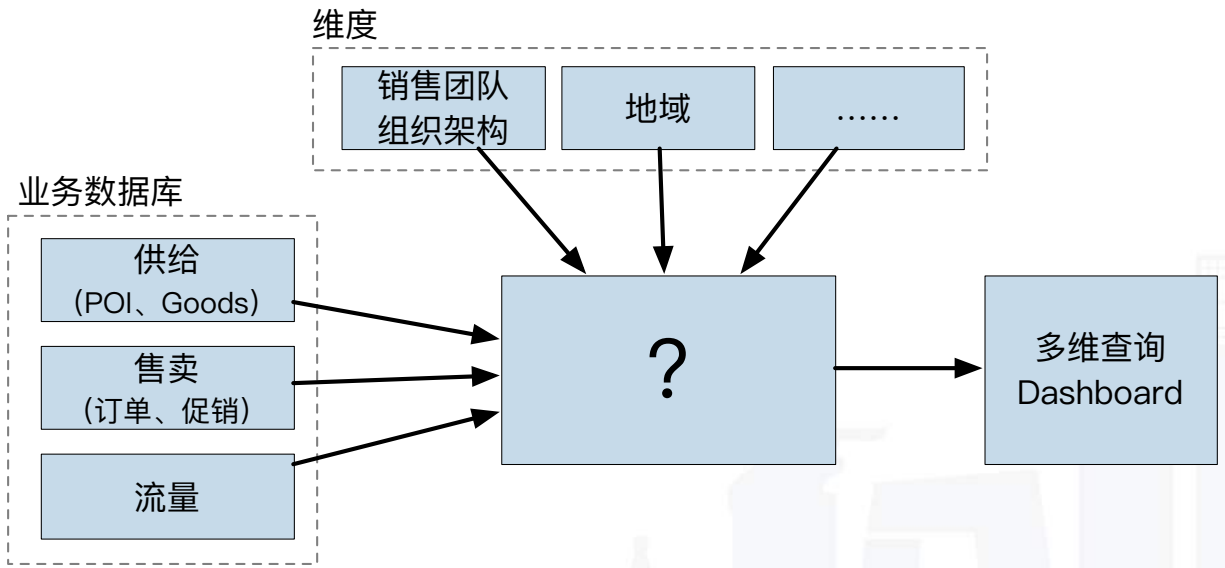
情报分析（Intelligence analysis）

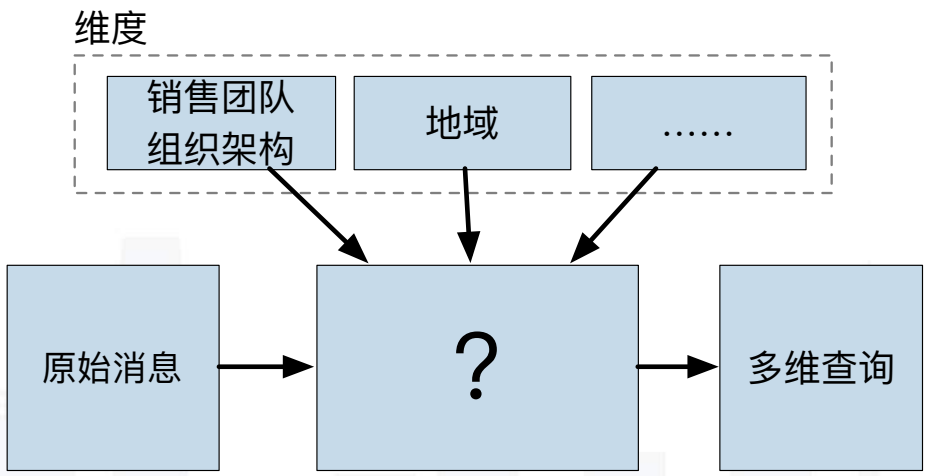
- 而且，这样的流程还有N个……



问题回顾

- 交互式报表（Interactive Reporting）
- 情报分析（Intelligence analysis）





- 可变数据，保证数据新鲜度
- 复杂业务逻辑

选型之路

调研、问题剖析、总结方法论

大数据生态

- Google大数据的脉络完整清晰，业界鱼龙混杂错综复杂

Spanner：《Spanner: Google’s Globally-Distributed Database》

F1：《F1: A Distributed SQL Database That Scales》

Dremel ：《Dremel: Interactive Analysis of Web-Scale Datasets》

PowerDrill ：《Processing a Trillion Cells per Mouse Click》

Mesa：《Mesa: Geo-Replicated, Near Real-Time, Scalable Data Warehousing》

Shasta：《Shasta: Interactive Reporting at Scale》

FlumeJava：《FlumeJava: Easy, Efficient Data-Parallel Pipelines》

MillWheel：《MillWheel: Fault-Tolerant Stream Processing at Internet Scale》

DataFlow：《The Dataflow Model》

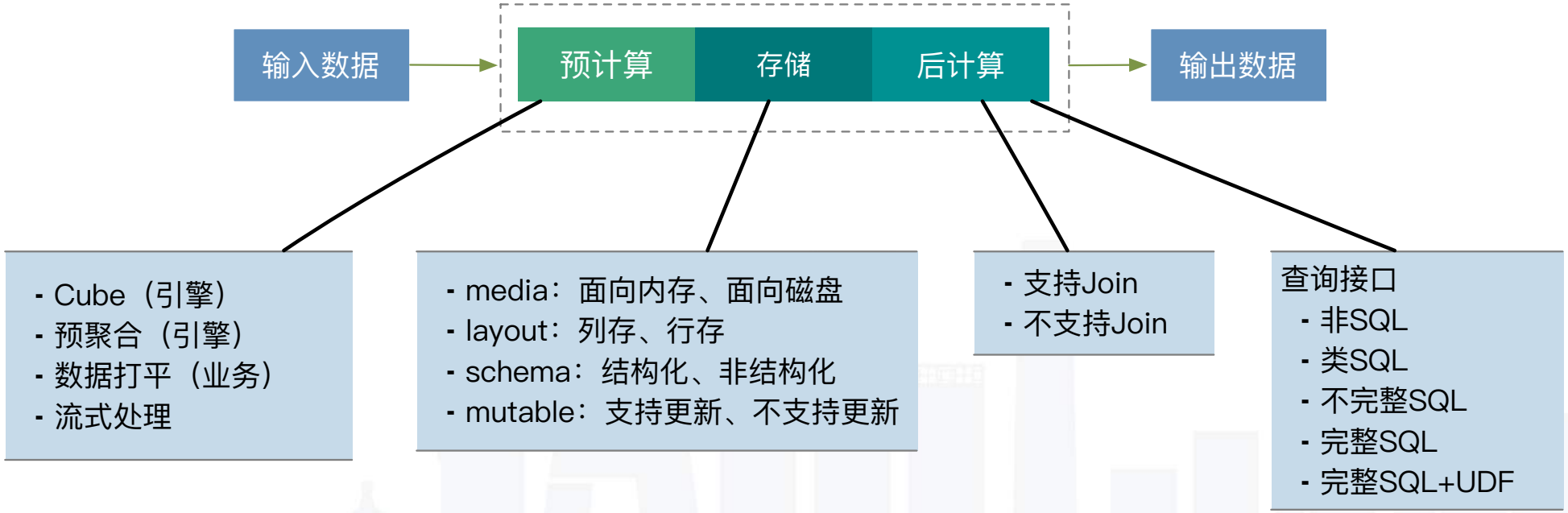
解构

- 全都是trade-off
  - OLAP vs OLTP
  - 流式处理 vs 批处理（Streaming vs Batch）
  - 预处理 vs 后处理（Pre-processing vs Post-processing）
  - 快照 vs 变更记录（Snapshot vs Changelog）
  - 平台 vs 业务
- 第一步是解构问题

数据处理简化模型

引擎特性解构

- 存储和计算分离考虑

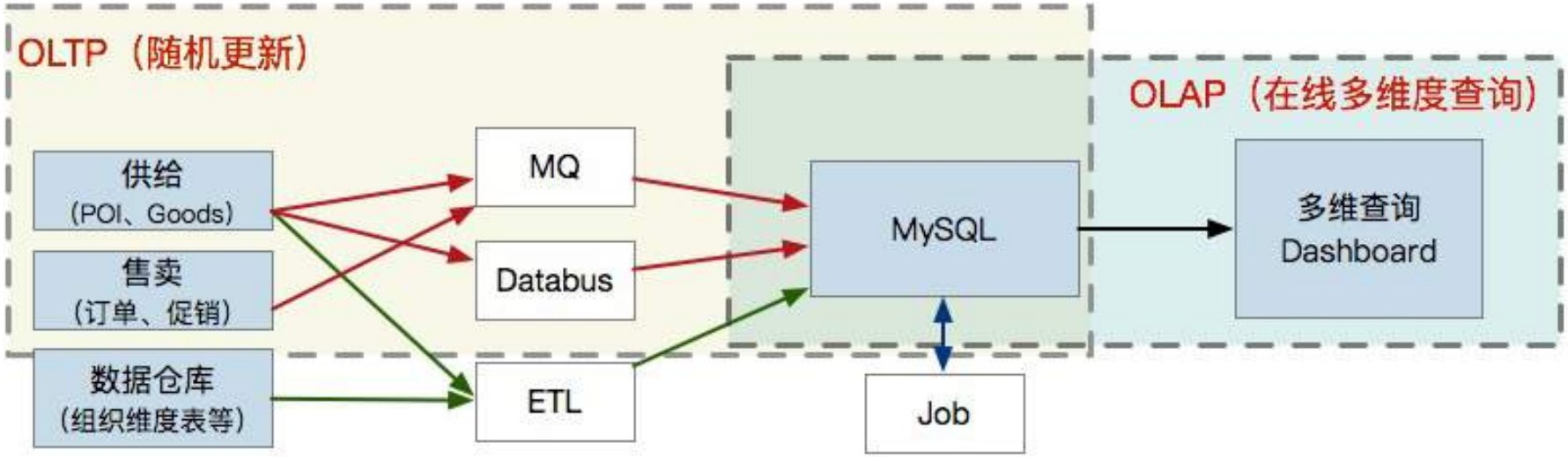


举例：Druid、Kylin、Hbase、Spark、Linkedin Pinot

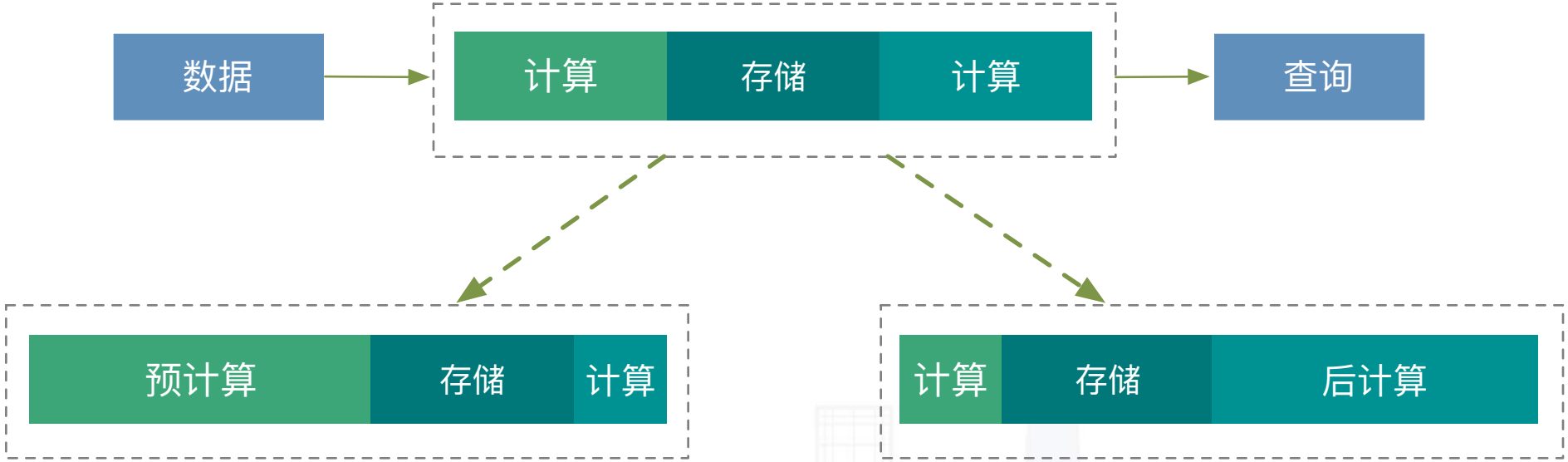
The trade-offs

OLAP vs OLTP

- OLAP：扫表性能，列存储
- OLTP：随机读写，行存储，索引
- 交互式报表是典型的HTAP（OLAP+OLTP）场景



Pre-processing vs Post-processing



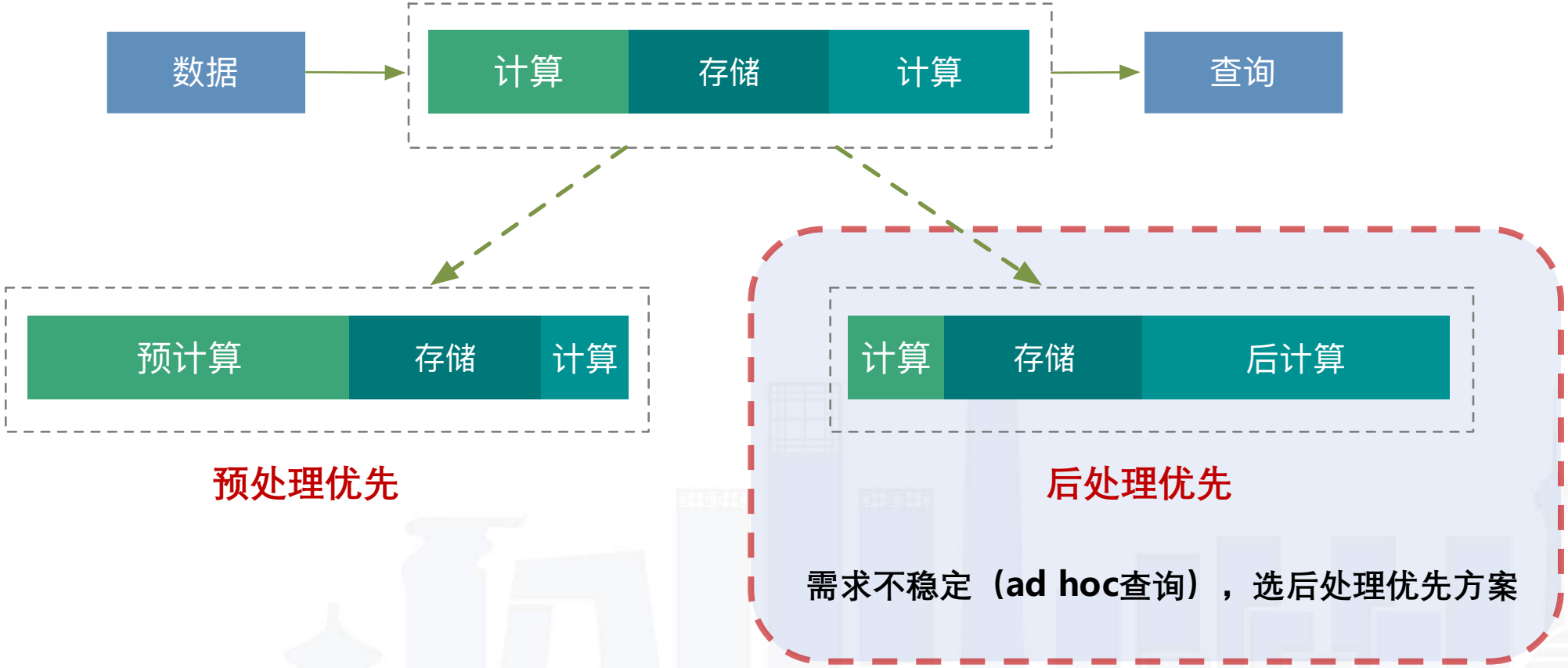
预处理优先

- 面向查询需求生成结果
- 灵活性低：需求变更需要重新生成数据
- 查询性能高
- 耗存储资源、人力

后处理优先

- 在线计算需求数据
- 灵活性高
- 查询性能难以保证
- 耗计算资源

Pre-processing vs Post-processing



人 vs 机器

Streaming vs Batch

- The Dataflow Model by Google
  - 理论层面大一统
- Flink, Spark Streaming
- 主要是预处理

Snapshot vs Changelog

- 历史数据如何保存？
- 数据量
- 信息损耗（数据更新）
- 易用性

平台 vs 业务

- 平台关注通用，业务多特化需求
- 通用方案 vs 专有方案
- 资源利用率未必高
- 研发效率与deadline

业务场景解构

准确度 允许损失精度 → 100%准确

牺牲少量精度，可实现巨大性能提升

数据量 <10GB → 100GB → TB → PB

单机 or 分布式，全内存 or 磁盘

吞吐 低 → 高

延迟 100ms → 3s → 20s → 1min → hour

低吞吐、低延迟比较容易

高吞吐、低延迟比较困难

- 分析场景下，高吞吐意味着预处理和Cache

新鲜度 <1s → 10s → 1min → 15min → hour → day

流式 or 批处理 or 在线计算

需求量 少 → 多

预处理 or 后处理

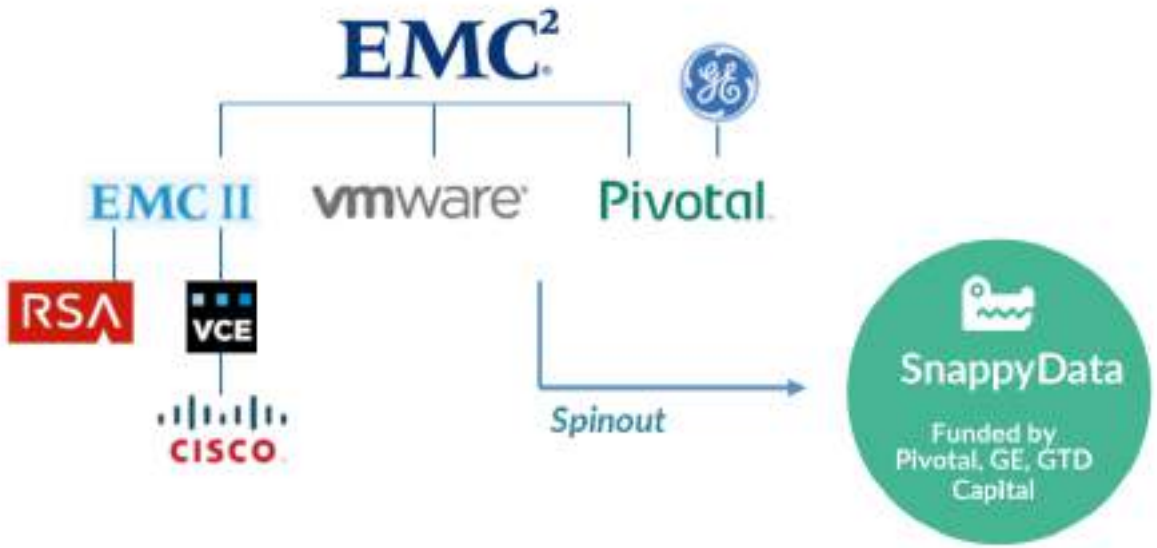
我们的场景

业务类型	分析型	列存储
准确度	100%	无法使用近似计算
数据量	1TB~3TB，热数据500GB	分布式，全内存
吞吐要求	低，<1qps	低吞吐
延迟要求	高，3s	低延迟
新鲜度	高，多维度，数据可更新	后处理，需要高效分布式Join
需求特点	量大、变更频繁	后处理

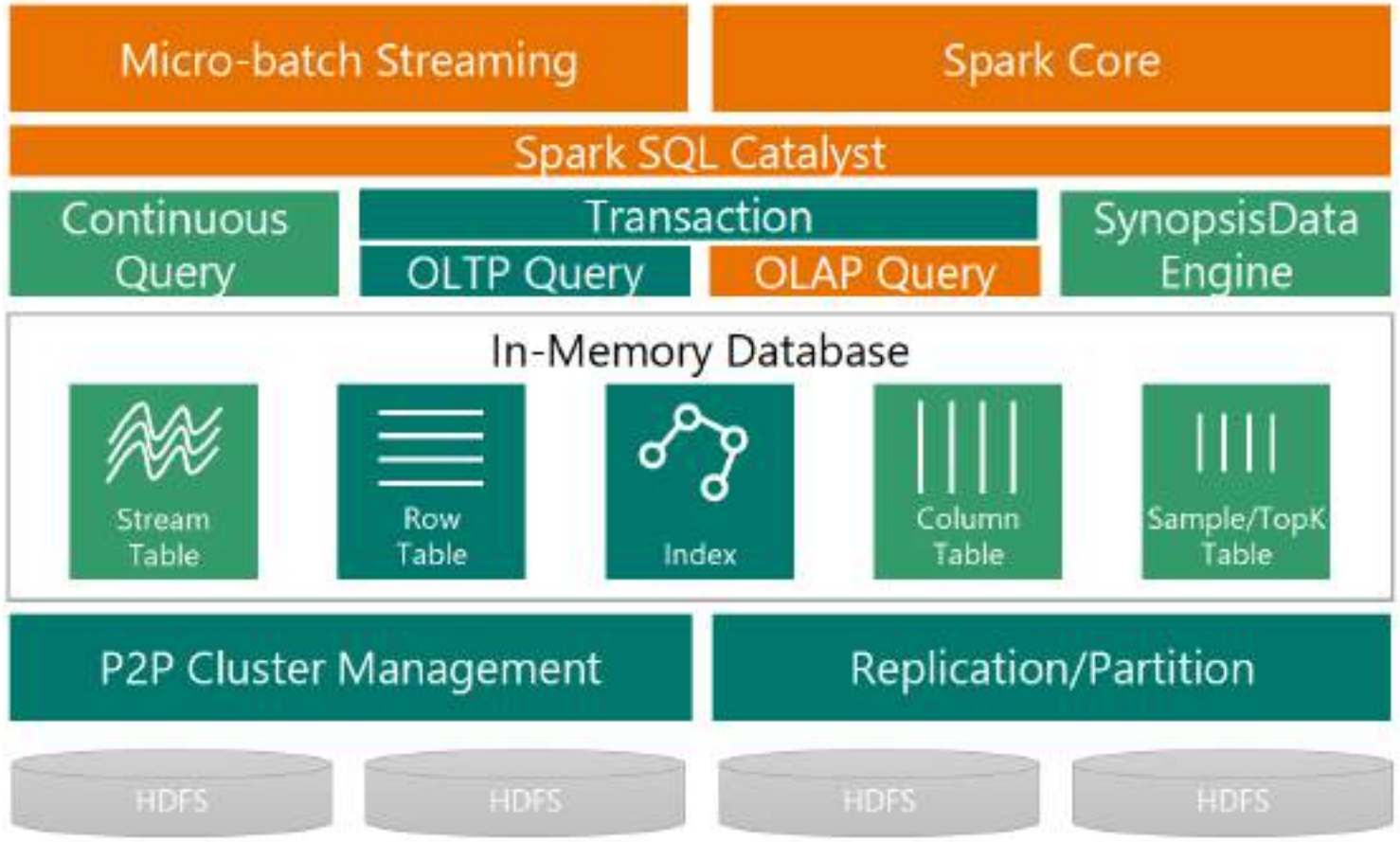
SnappyData

SnappyData

- Database on Spark
  - OLTP + OLAP + Streaming
- 深度整合GemFire XD和Spark
- 100%兼容Spark API
- Row or Column tables
- Replicated and partitioned tables
- In-memory计算，支持overflow
- 近似计算（商业Feature）
  - 牺牲1%准确度，换取200x性能提升

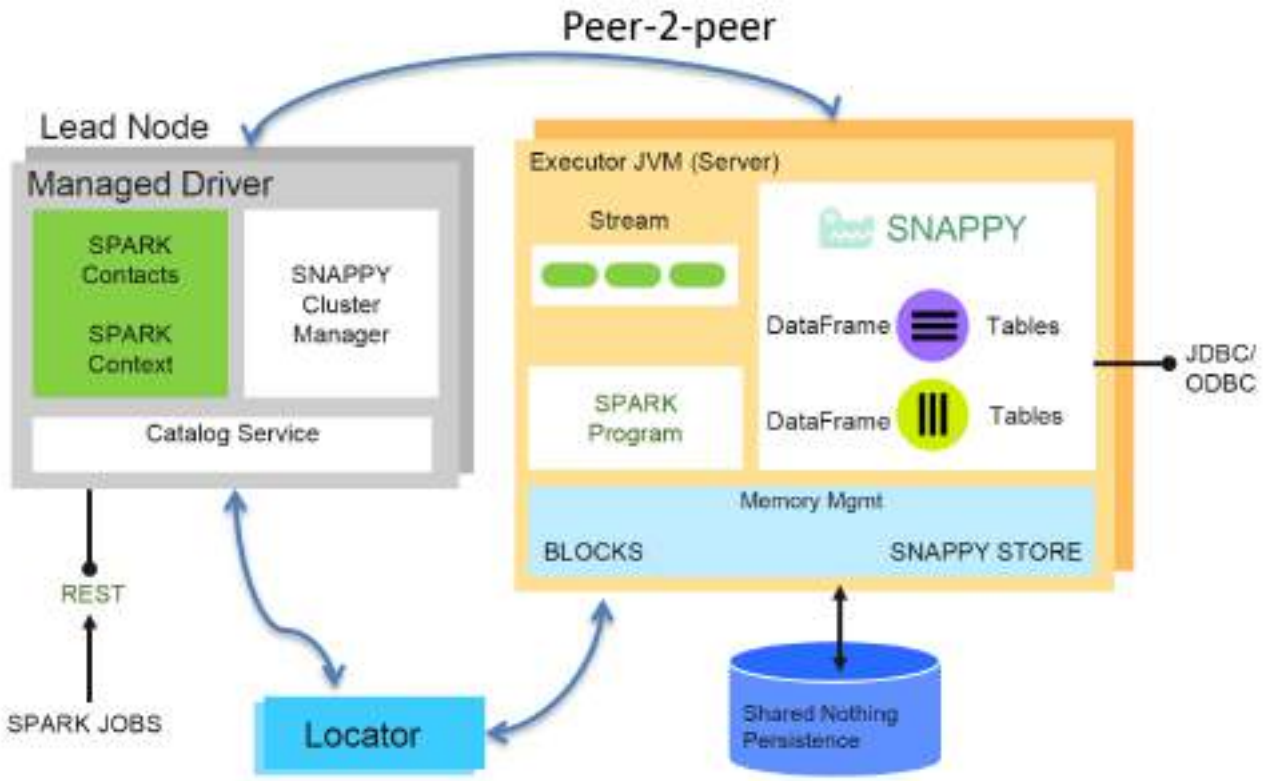


整体结构


Micro-batch Streaming
Spark Core
Spark SQL Catalyst
Continuous Query
Transaction
OLTP Query
OLAP Query
SynopsisData Engine
In-Memory Database
Stream Table
Row Table
Index
Column Table
Sample/TopK Table
P2P Cluster Management
Replication/Partition
HDFS
HDFS
HDFS
HDFS

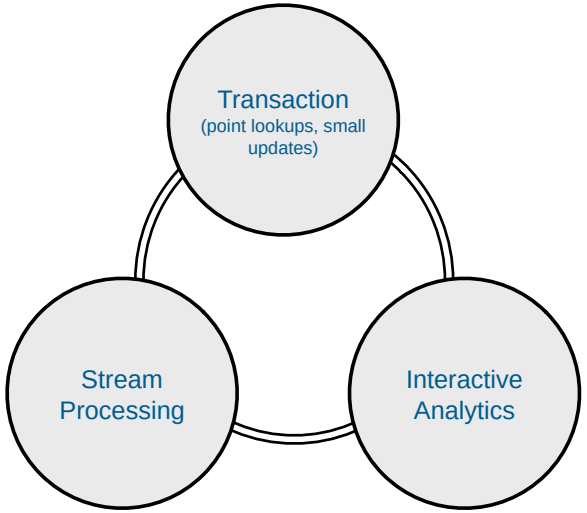
集群架构

- P2P
- 三种角色
  - Lead Node
  - Locator
  - Data Node
- Long running Spark Executors
- Driver HA
- Data HA

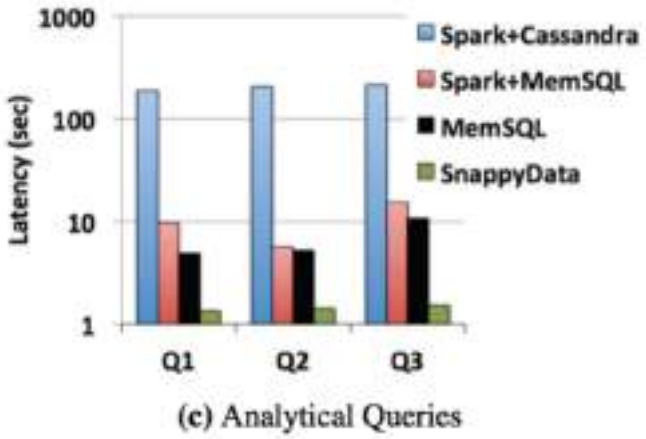


SnappyData的优势

- 简化数据处理架构
  - 流式处理、存储、计算引擎一体

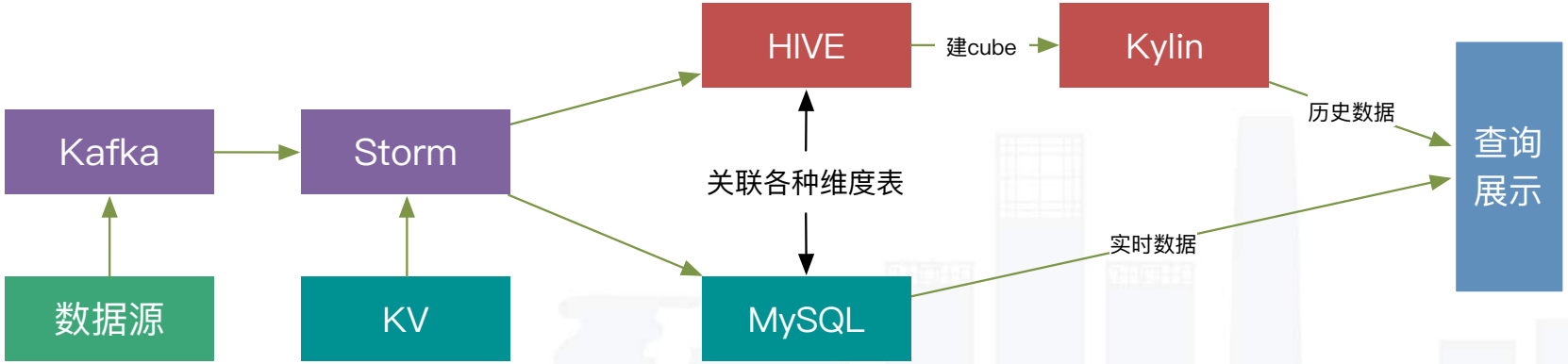


- 高性能
  - 相较于Spark+X方案，有7~142x查询性能提升
  - 尤其是Join性能



简化数据处理架构

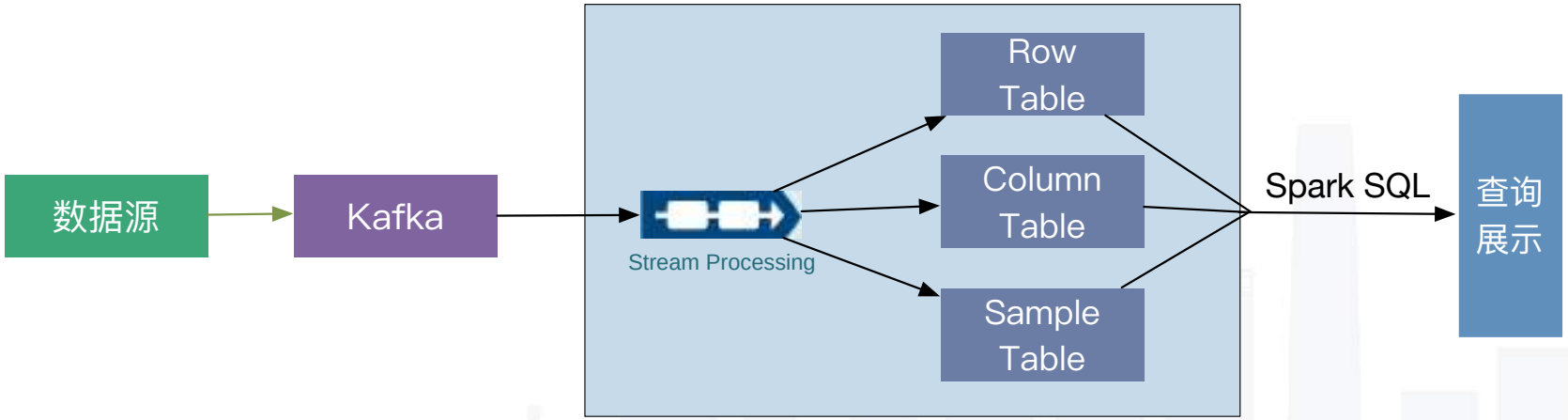
- Lambda架构，不论系统维护还是业务开发都很复杂
  - 两条链路，开发、调试都是两次
  - 开发方式不同，逻辑一致难以保证



举例：实时情报分析（Lambda Architecture）

简化数据处理架构

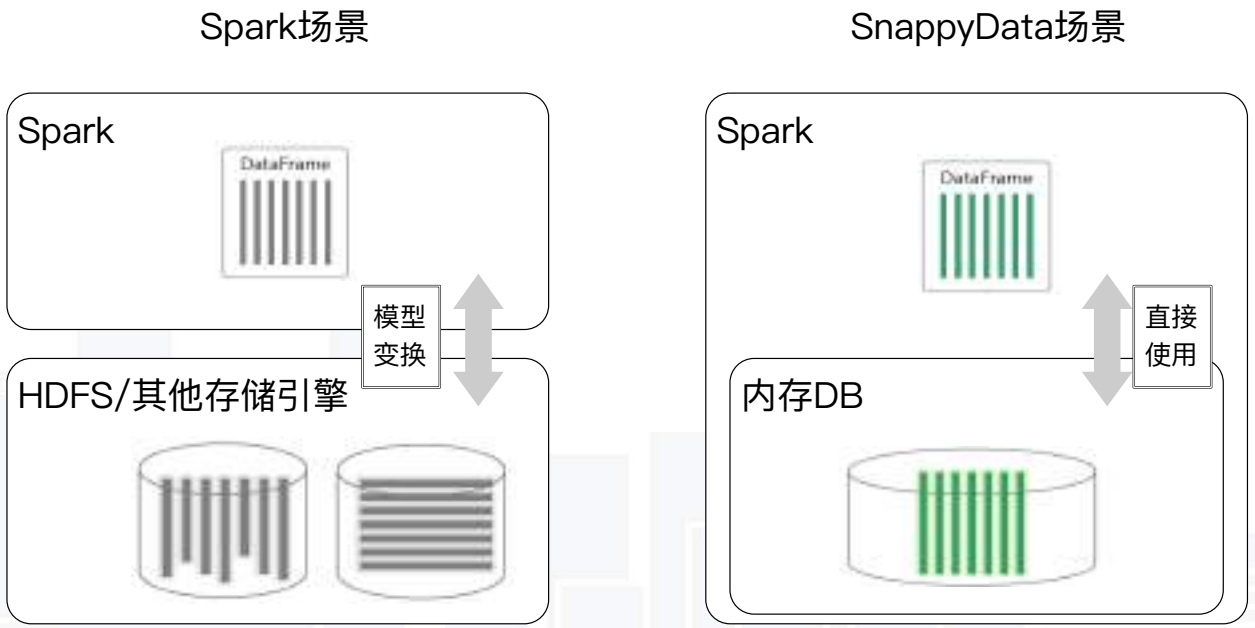
- Lambda架构，不论系统维护还是业务开发都很复杂
  - 两条链路，开发、调试都是两次
  - 开发方式不同，逻辑一致难以保证



举例：实时情报分析（SnappyData）

高性能 - 深度整合

- 计算数据与Spark在同一个JVM中
  - 无数据拷贝/移动
- 内存中数据格式与Spark一致
  - 无序列化/反序列化



高性能 - Colocated Join

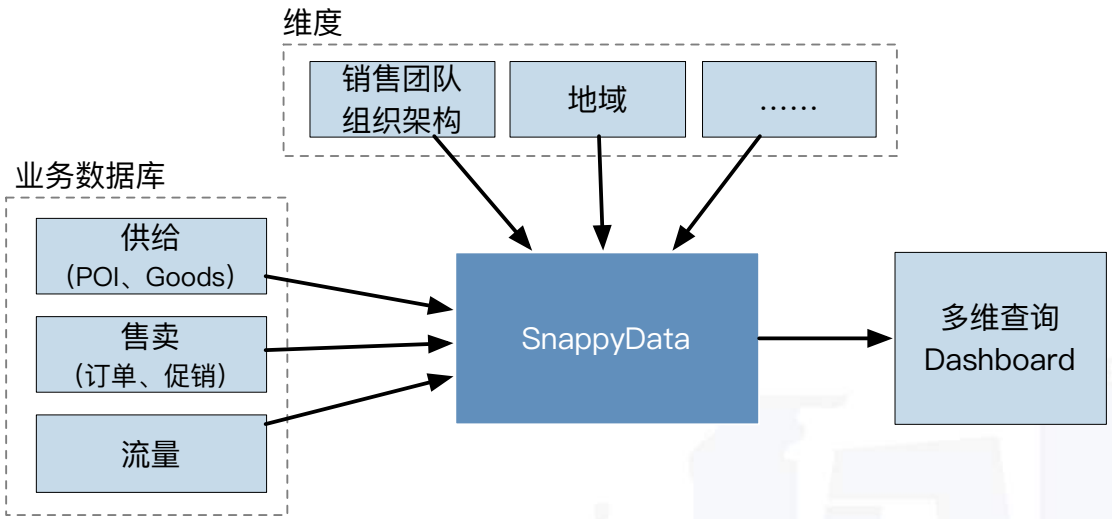
- 控制相关表使用相同的数据分布策略，如POI_ID
- 极大减少了Shuffle，使Join在日常使用成为可能

```
CREATE TABLE OE_DIM_POI(
  mt_poi_id BIGINT,
  poi_name VARCHAR(200),
  brand_id INT,
  ...
  is_high_star INT,
  version TIMESTAMP
) USING COLUMN OPTIONS(
  PARTITION_BY 'MT_POI_ID',
  REDUNDANCY '0',
  BUCKETS '761',
  PERSISTENCE 'ASYNC',
  OVERFLOW 'TRUE'
);
```

```
CREATE TABLE OE_MBL(
  id LONG,
  datekey INT,
  checkin_date INT,
  checkout_date INT,
  ...
  valid_status TINYINT,
  gmt_time TIMESTAMP,
  version TIMESTAMP
) USING column OPTIONS(
  PARTITION_BY 'MT_POI_ID',
  BUCKETS '761',
  COLOCATE_WITH 'OE_DIM_POI',
  REDUNDANCY '0',
  PERSISTENCE 'ASYNC',
  OVERFLOW 'TRUE'
);
```

架构变化

交互式报表（Interactive Reporting）

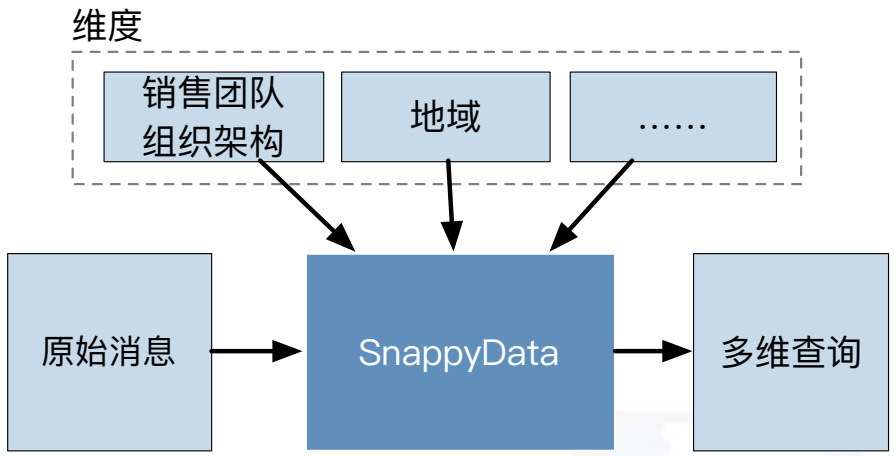


结果

- 线上4节点集群（128G, 32CPUs, SSD）
  - 平均查询耗时 <2s
  - 10亿行事实表，5维度表Join，5s
- 消除20+中间表，仅使用原始表查询
- 消除20+预计算任务
- 指标完全一致
- 研发效率大幅提升
  - Java + 中间表 + SQL → SQL

架构变化

情报分析（Intelligence analysis）



结果

- 线上36节点集群（128G, 32CPUs, SSD）
- 核心业务（UDF实现）
  - 120行事实表，3表join，0.4s
- 研发效率大幅提升
  - Java + 中间表 + SQL → SQL

应用中遇到的几个问题

1. 复杂业务逻辑如何支持
2. 历史数据如何存储
3. 实时数仓如何建模
4. 首次执行性能优化

1. 复杂业务逻辑如何支持

- 情报分析核心逻辑用SQL无法高效表达
  - 涉及对数亿行数据进行排序，无法在线使用
- Spark支持两种UDAF
  - ImperativeAggregate
    - 支持外部注册，用代码直接直接实现逻辑，易于开发，但有函数调用开销
  - DeclarativeAggregate
    - 内部使用，无公开接口
    - 直接操纵Catalyst表达式，支持代码生成，性能强大
    - 写法类似于写Lisp Macro

修改SnappyData，开放Spark的声明式UDAF注册（释放了魔鬼）

2.历史数据如何存储

- 业务数据库的变更历史，如何存储、如何使用
  - 例：查询某一历史时刻整个业务订单状态，进行多维度分析
- 两个选择：定期快照（Snapshot），变更记录（Changlog）
- 方案：
  - 将Changelog转换为SCD Type2形式的事实表

```
# 当前状态
where current_flag = 1
# 历史状态
where <timestamp> between start_time and end_time
```

  - 依赖SD的join update性能，将changelog表转成SCD表

3. 实时数仓建模

- 数仓建模是分层的，存在实体中间层
- 依赖于在线join的实现方式，中间层不存在实体

4. 首次查询性能优化

- SnappyData对Spark SQL代码生成进行了优化
  - 基于Tokenization的QueryPlan复用

```
select name from person where age > 26 and city = 1
```

```
select name from person where age > 30 and city = 20
```

- 首次查询慢根因在于JIT优化需要Profiling数据
  - 不同SQL模板对应不同QueryPlan，需重新生成代码
  - QueryPlan缓存是Connection Scope
- 优化
  - 没有彻底的办法
  - Query预热，适用于查询模式数量较少场景
  - 减少连接数（限制连接池大小）

总结

查询性能	列存储扫表性能	全内存	colocated join
复杂逻辑	声明式UDF		
历史数据&数据量	SCD事实表		

Parting Thoughts

Lesson 1: 机器换人力很划算

- 我们常常低估人力成本、高估机器成本
- 高昂的人力成本通常伴随着低效率

Lesson 2: 预处理不该是数据业务首选

- 需求变更频繁的场景，预处理越多，通常意味维护成本越高
  - 做的越多，错得越多
  - 否则就需要一个强大的约束与检查系统

- 单个垂直业务(or 创业公司)的数据量真的没有那么大
  - 不要搞“尬并行”

Embarrassingly parallel

From Wikipedia, the free encyclopedia

- 作为数据服务，通常不用考虑吞吐量
  - Just串行！

Lesson 3: SQL, not Java

- 表达能力对于研发效率和可维护性至关重要
  - 让机器生成low-level代码，而不是自己写
- 但SQL并不图灵完备
  - UDF and RVL

谢谢！

主办方 Geekbang. 极客邦科技 InfoQ

GMTC 2018

全球大前端技术大会

大前端的下一站

<<扫码了解更多详情>>

ArchSummit
全 球 架 构 师 峰 会

关注 ArchSummit 公众号

获取国内外一线架构设计

了解上千名知名架构师的实践动向

Apple • Google • Microsoft • Facebook • Amazon 腾讯 • 阿里 • 百度 • 京东 • 小米 • 网易 • 微博

深圳站：2018年7月6-9日　北京站：2018年12月7-10日

主办方 Geekbang. 极客邦科技 InfoQ

QCon 上海站

全球软件开发大会【2018】

2018年10月18-20日

7折 预售中，现在报名立减2040元

团购享更多优惠，截至2018年7月1日

极客邦科技
企业培训与咨询
Geekbang.

扫码关注
获取更多培训信息