FLINK FORWARD # ASIA 2022 ONLINE | 实时即未来

货拉拉基于Flink计算引擎的应用与优化实践

王世涛 | 货拉拉实时研发平台负责人

01 flink 在货拉拉的使用现状

1 flink 在货拉拉的使用现状

部署情况

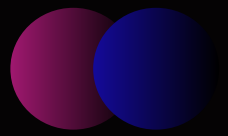
- 4个中心
- 3条业务线
- 混合云

任务使用

- jar/sql 任务
- 任务总数 1800+

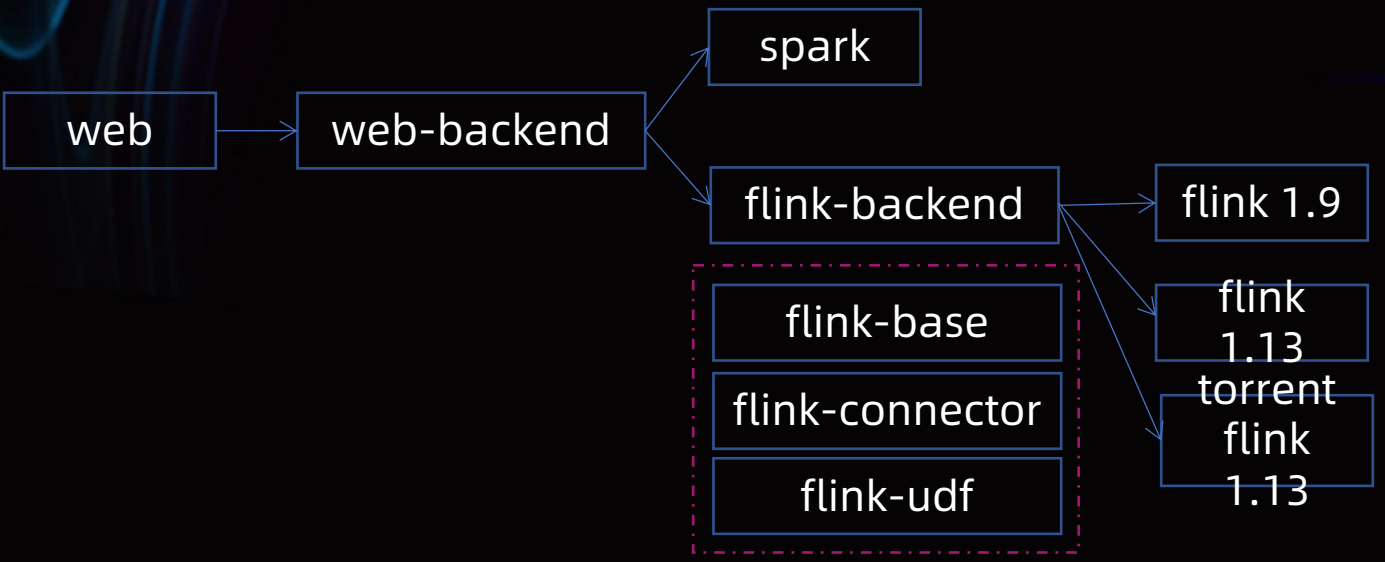
业务使用

- 业务风控
- 交易引擎
- 车联网技术
- 信息安全
- 地图
- 增长中台营销
- 数仓
- 智能运营



02 flink 平台化

2.1 平台架构



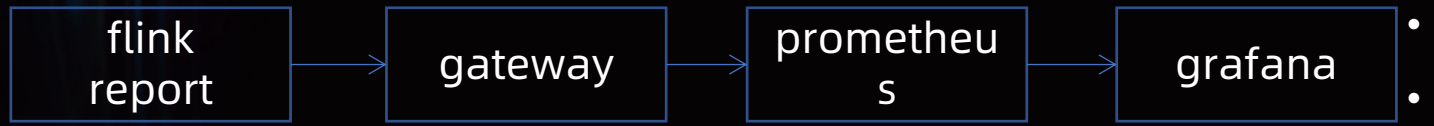
任务提交

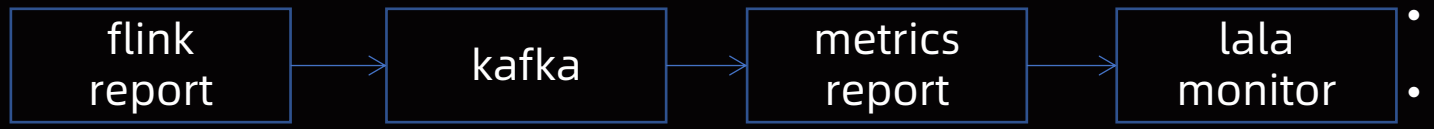
- flink-base实现了flink-sql的改写和控制能力
- flink-base和原生flink双管齐下
- 多引擎提交
- 多版本提交

2.2 指标监控

链路调整

before

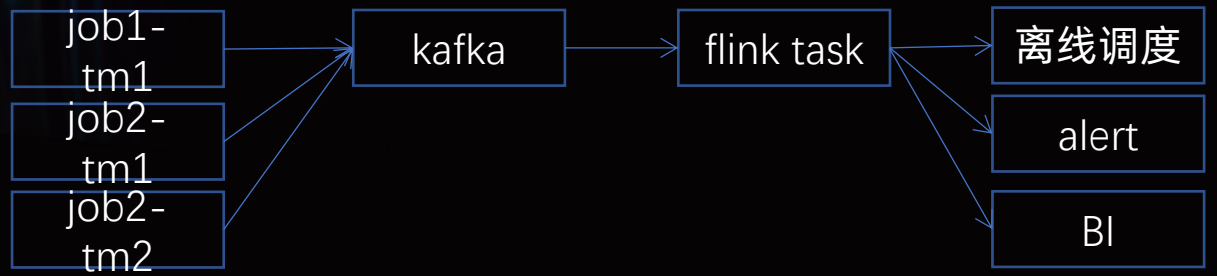




after

- 背景：采集链路调整，对接公司级监控体系
- 解法1：自定义实现flink kafka report
- 解法2：metrics-report 实现指标上报和控制
- 收益1：kafka 可以拓展指标预聚合
- 收益2：减少链路运维和相应开发，由专业团队维护

2.3 分区数据完整性可见性



集中异步式控制

- 背景：每个任务中做可见性判断会侵入任务
- 解法：多任务多并发统一的判断逻辑
- 收益1：离线任务由时间驱动->事件驱动
- 收益2：可以统计小文件/文件压缩比等指标
- 收益3：方便调整逻辑和资源

平台化实现点

工程化

- 压测
- 加密
- 认证
- 鉴权
- 血缘

部署

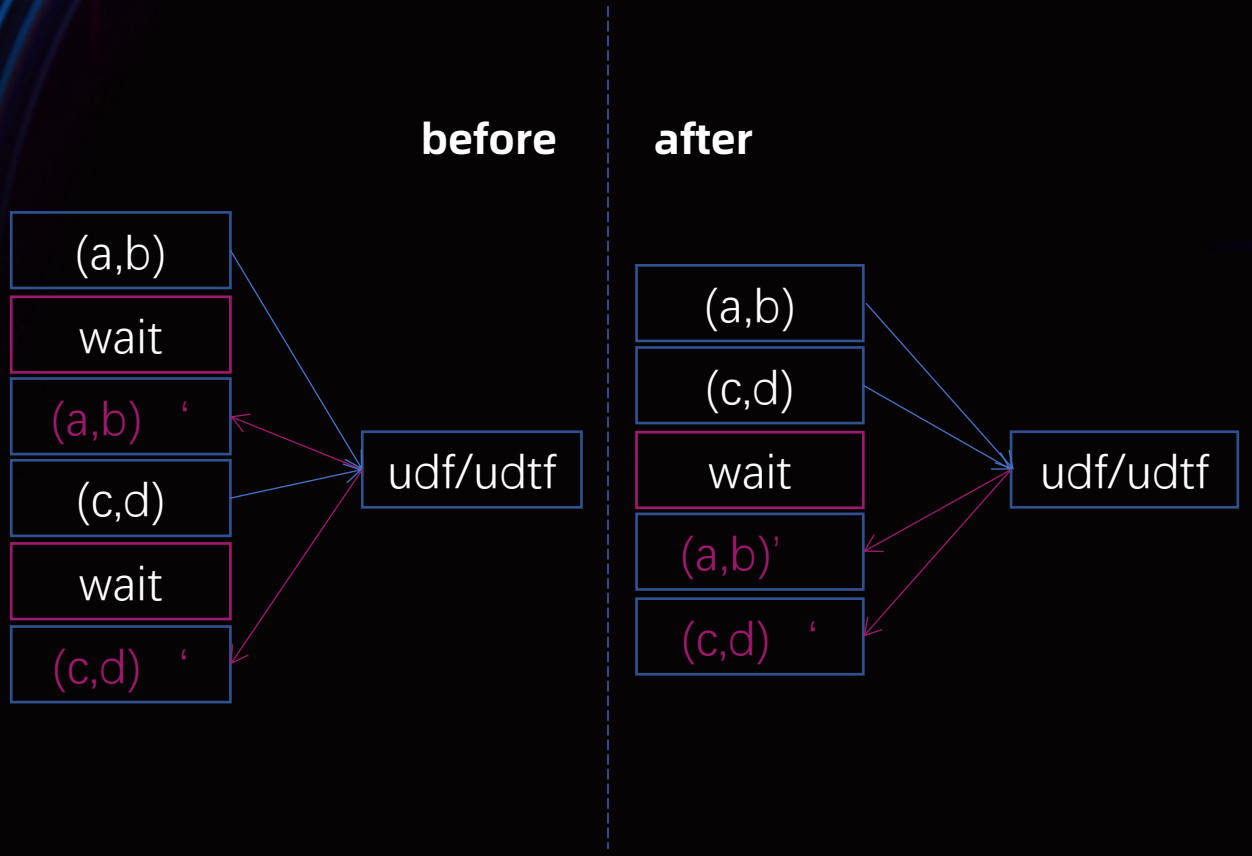
- 多dc部署
- 多fs写入
- 多版本提交
- 多模式提交
- 多队列提交
- **多计算引擎提交**

开发

- sql等价改写
- schema 推导
- connector开发
- 自定义udf
- 接入hive udf
- 指标监控
- 分区完整性

03 性能优化主题

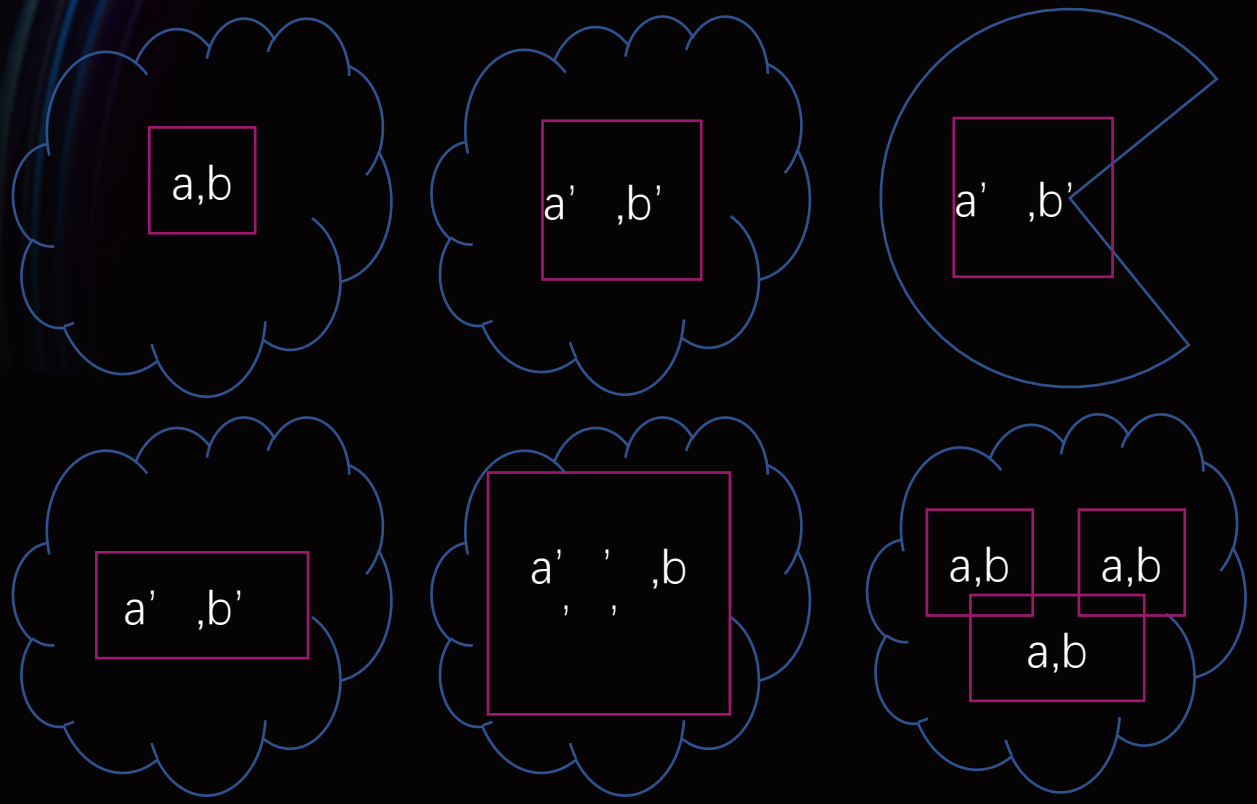
3.1 经纬度求行政区等归属场景



udf/udtf 维表化

- 背景：存在性能慢udf/udtf；udf/udtf方式，只能单线程同步处理
- 解法1：利用异步io特性实现维表化
- 解法2：自动实现udf到udf维表的功能映射
- 收益：支持多线程异步处理

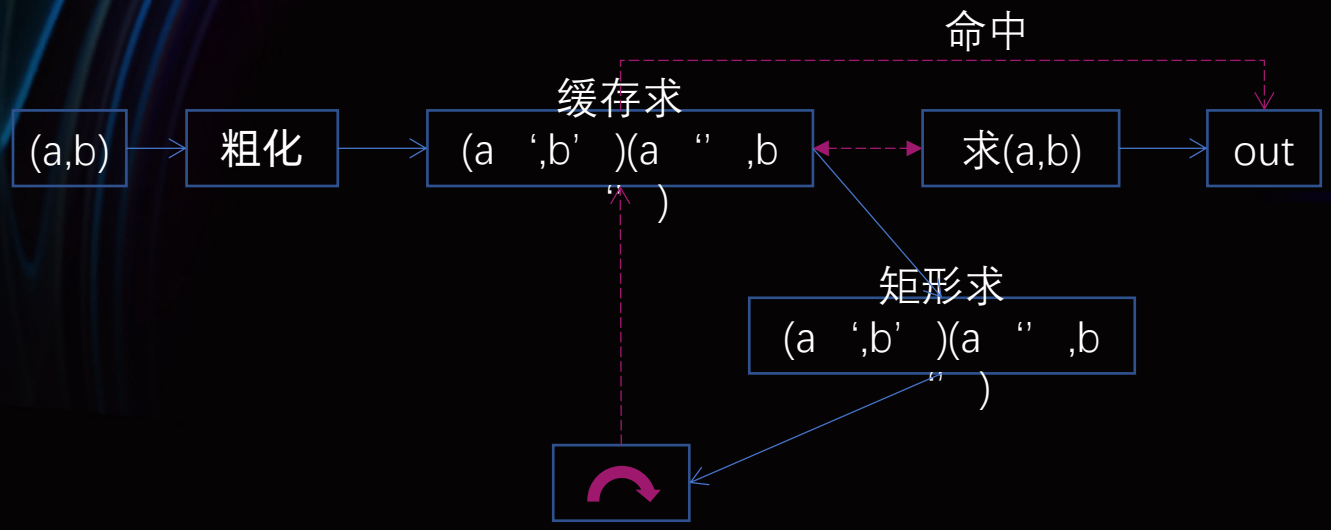
3.1 经纬度求行政区等归属场景

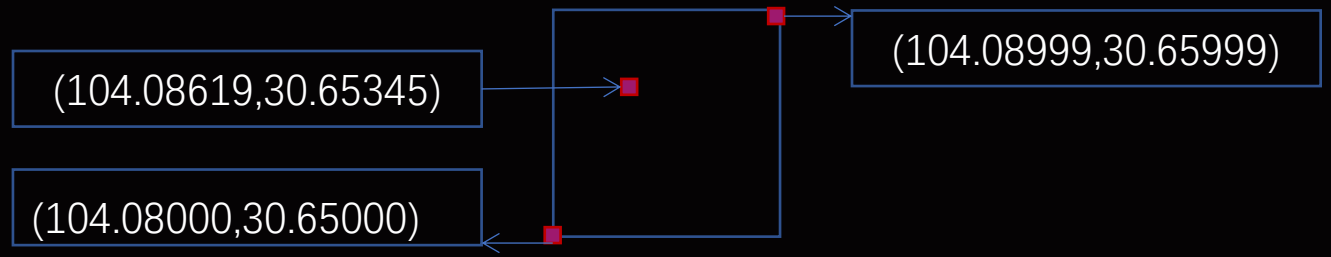


经纬度预处理缓存

- 背景：算点属于多边形内是一个复杂的过程；固定的经纬度求行政区等归属是幂等的；经纬度太多，直接进行缓存，命中率不高
- 解法1：把经纬度的点粗化成一个矩形
- 解法2：最小外接矩形 简化成 多个内接矩形
- 收益：提高缓存命中率，既而提高处理吞吐

3.1 经纬度求行政区等归属场景

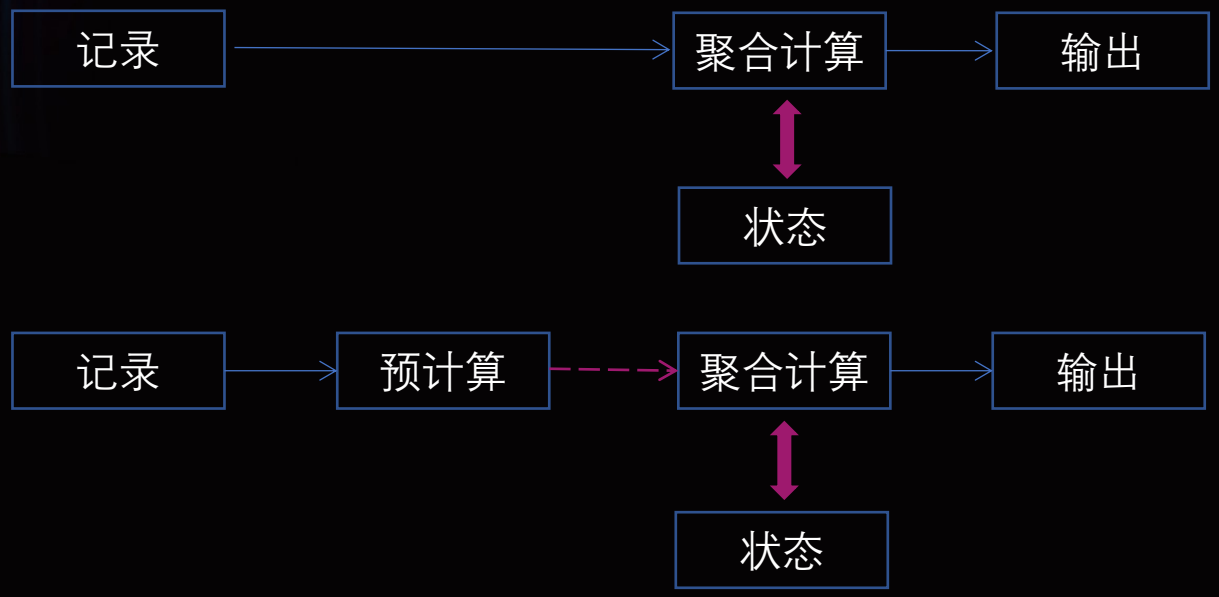




经纬度预处理缓存

- 粗化：由一个点粗化成一个面
- 粗化方式：目前支持去掉一定的精度，然后用0和9去补齐对应的精度
- 缓存的主键：左下和右上两个矩形顶点的组合
- 求值过程1：粗化使用矩形方式求归属是异步过程
- 求值过程2：粗化使用矩形方式求归属，可能会跨多个归属，这种数据不放进缓存

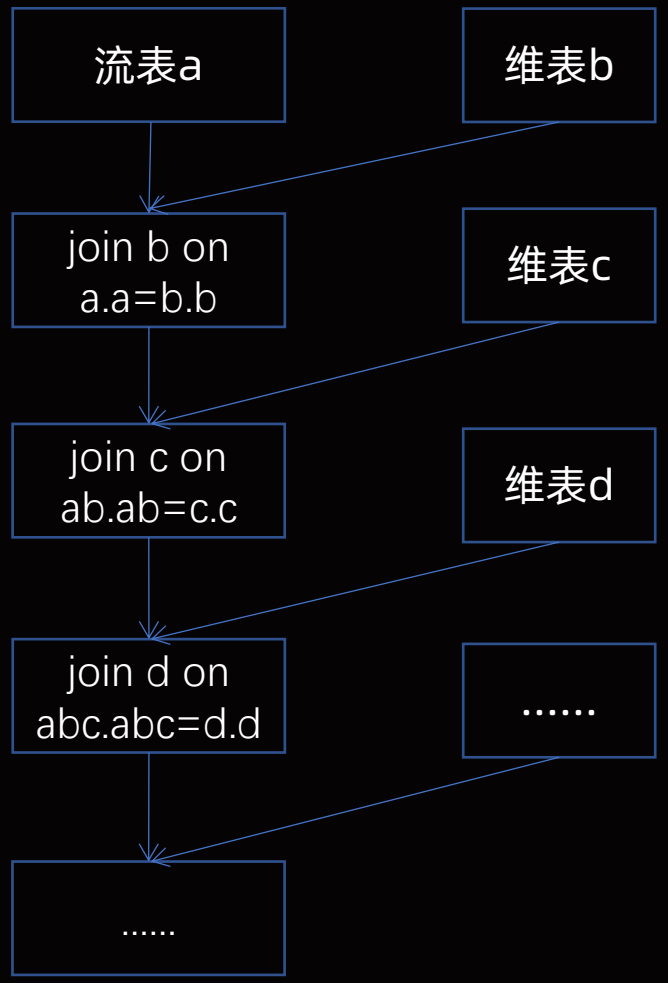
3.2 binlog数据聚合函数计算场景



减少聚合函数计算量

- 背景：binlog数据是一种cdc格式的数据源，包含了数据前后变化的数值；有的数据不聚合计算不影响最终结果
- 解法：如果old属性里，没有聚合的字段，不需要进入聚合计算
- 解法2:不同的聚合逻辑，可以适配不同的前置过滤预计算
- 收益：前置的判断预计算比起聚合计算来说，是轻量的

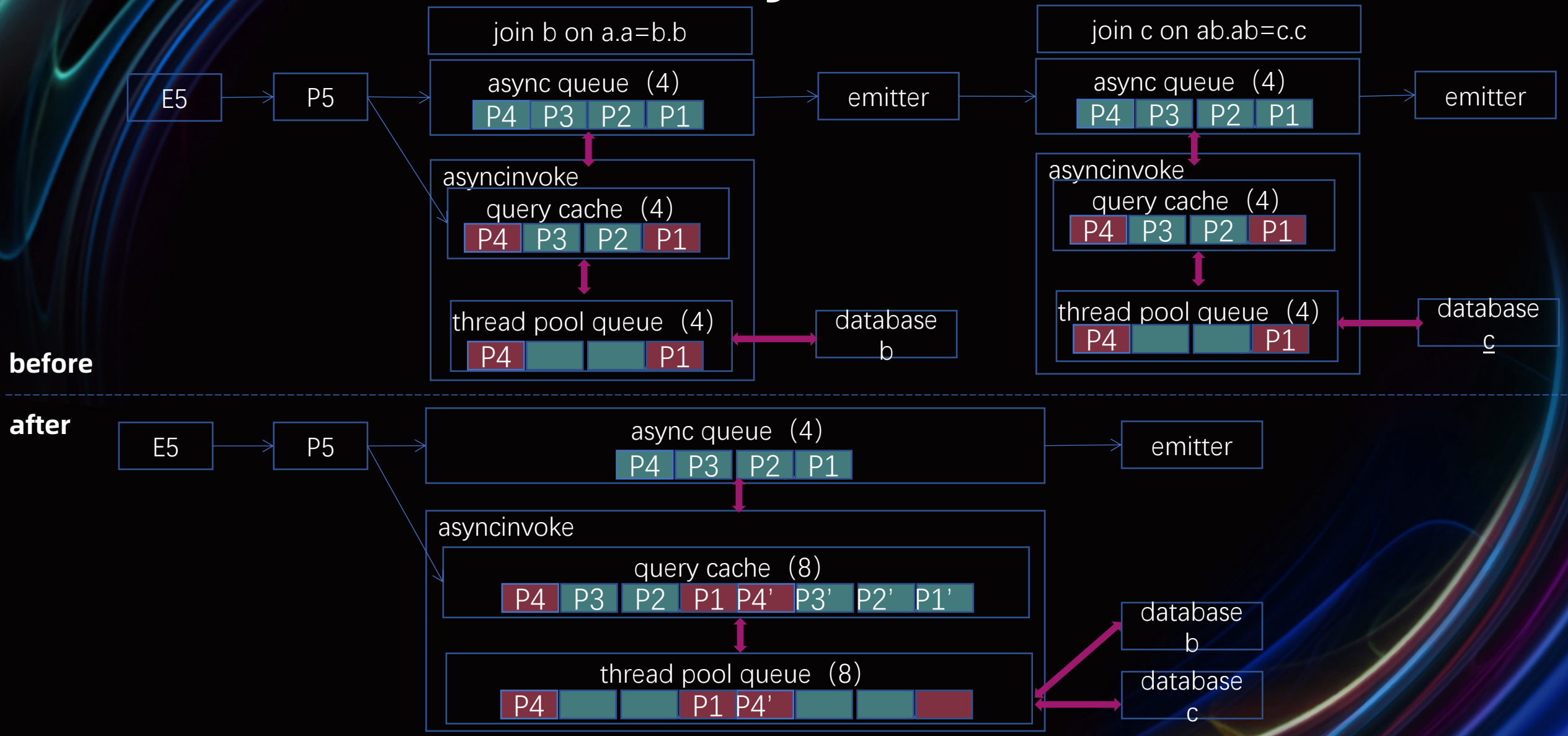
3.3 连续异步有序join-业务场景



业务场景

- 大宽表的业务场景
- 生产中存在使用同一张流表的不同字段连续join N 张hbase维表的场景
- 这类任务重要，且容易存在性能问题

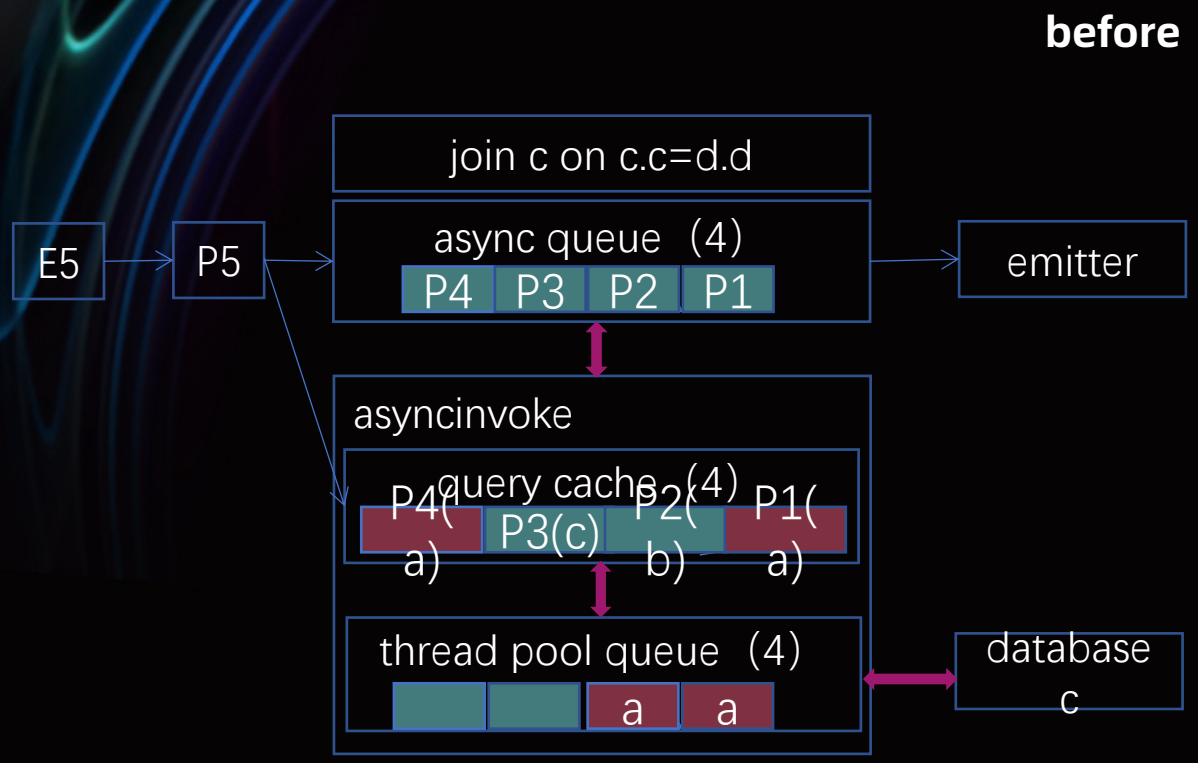
3.3 连续异步有序join-流程分析



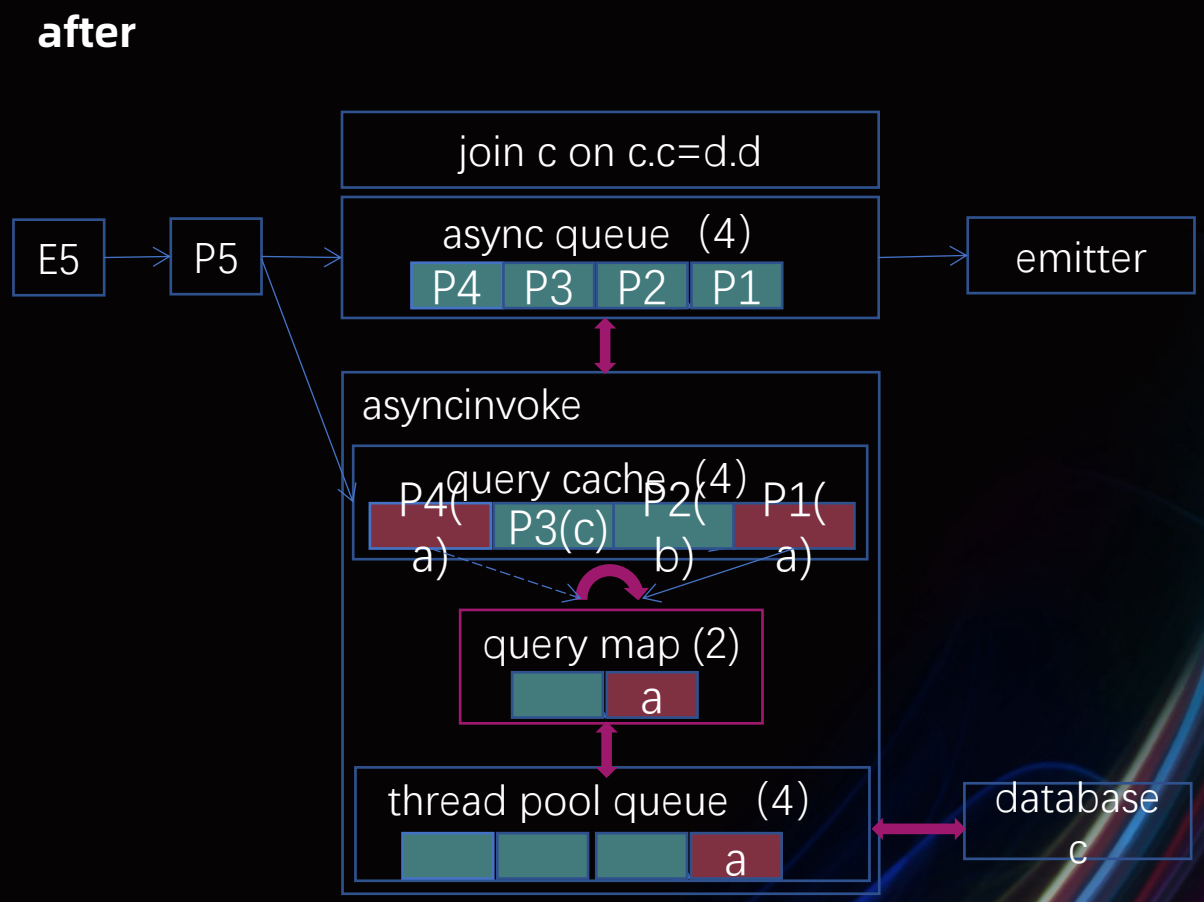
3.3 连续异步有序join-收益分析

- 前后记录在2个join下，在缓存和查询的这个场景里，就有2的N次方个case
- 由于篇幅时间有限，重点对着收益点主要讲几个case

3.3 连续异步有序join-合并查询



- 多次同条件连续查询



- 多次同条件连续查询变成一次查询+多次等待

性能优化点

计算

- **经纬度预处理-面缓存**
- **时间预处理-线缓存**
- **keyby-点缓存**
- **分库分表-表正则解析缓存**
- **udf/udtf 维表化**
- **减少聚合函数计算量**
- **动态过滤条件**
- **join顺序调整**

connector

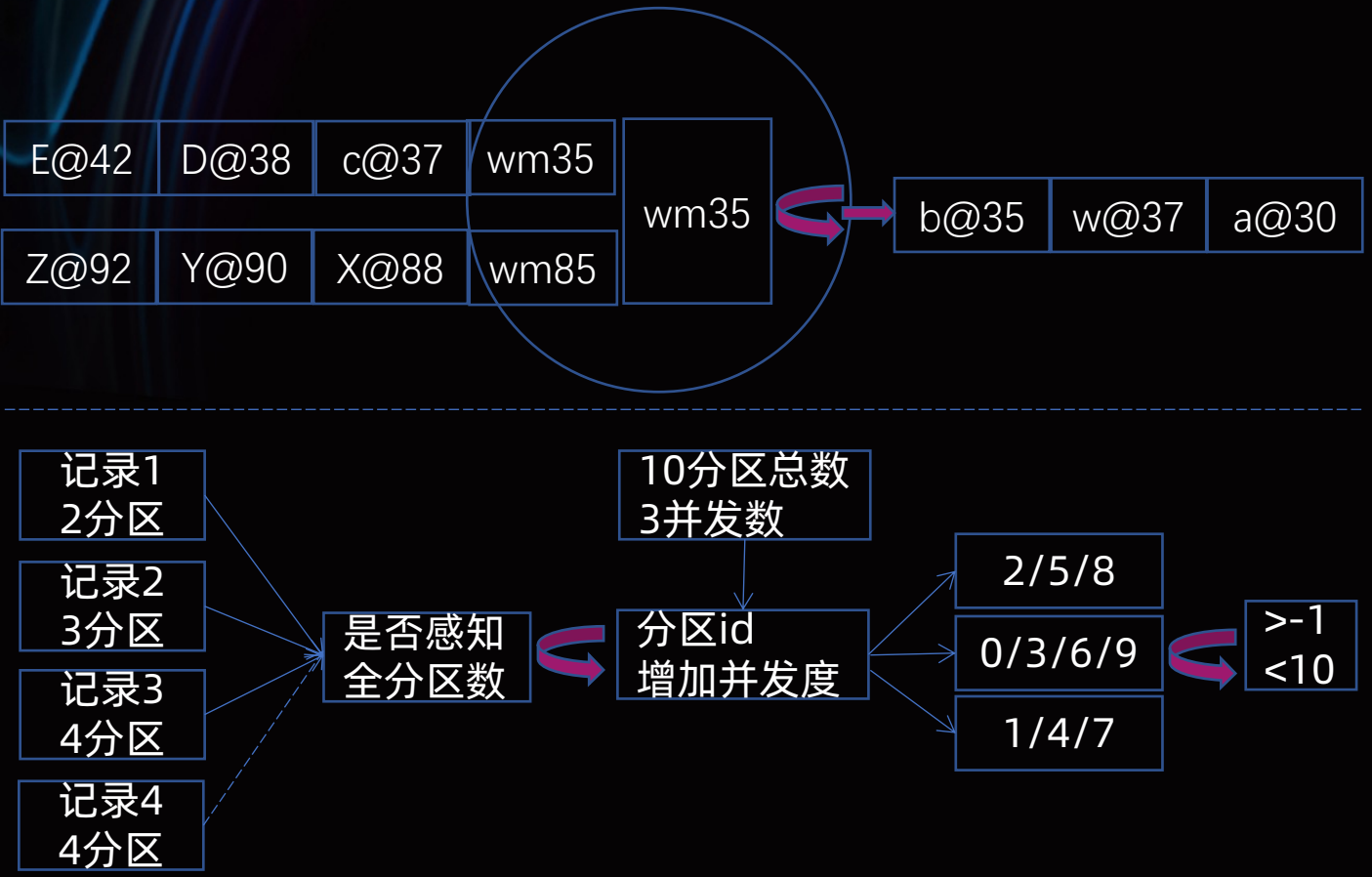
- source过滤下推到序列化之前
- 加快发送，多queue sink
- 减少发送，batch distinct sink
- 不发送-未更新的数据
- 不发送-json空key的字段
- 不发送-回撤流false数据

用法

- **object-reuse**
- **micro-group**
- 算子粒度并发调整

04 数据准确性主题

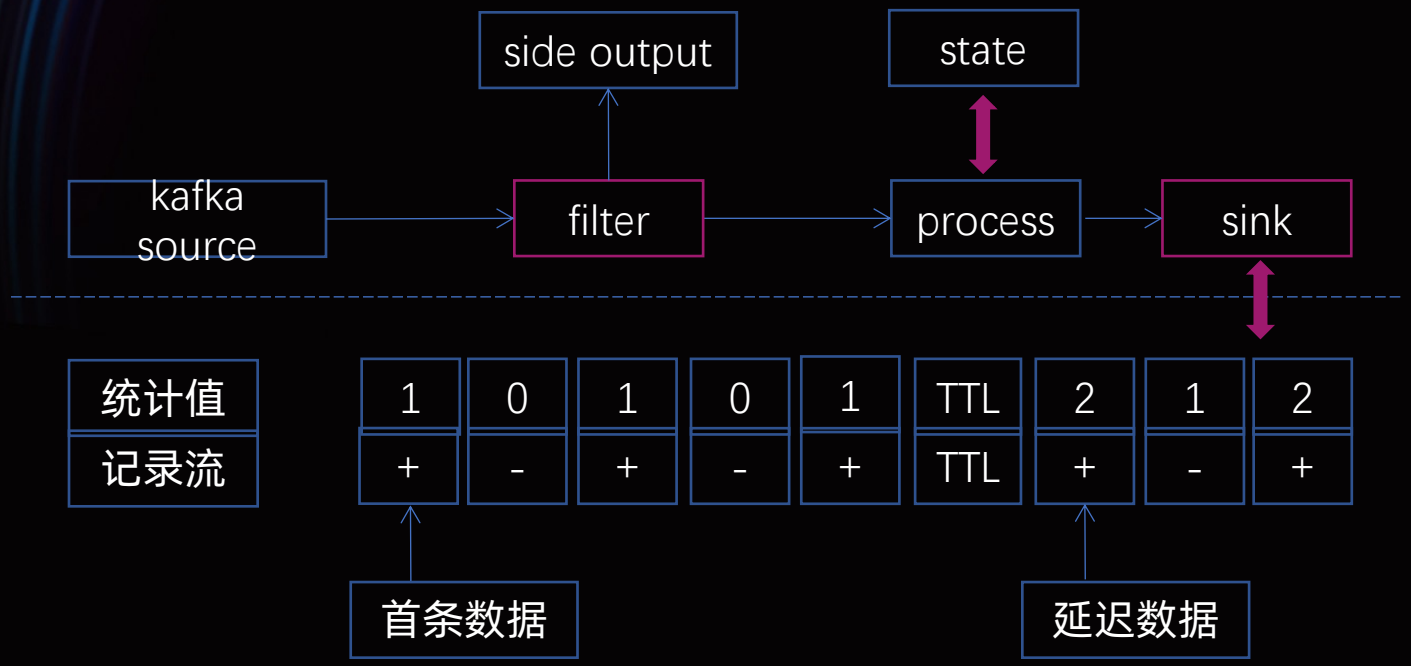
4.1 通用版本 Flink Per-partition watermark问题解决方案



自动感知分区数

- 背景：并发少于分区数下，存在一个并发中有多个分区的数据，水印会互相影响，导致基于水印的逻辑存在正确性问题
- 解法1：在记录中增加记录的kafka分区id
- 解法2：水印提取器，感知分区数和做水印的处理，取不同分区下最小水印进行发送
- 规则：根据分区总数，并发总数，和当前一条数据的分区id，推测出该并发负责的其他分区

4.2 状态过期+延迟数据 聚合数据覆盖



source+sink处理

- 背景：状态TTL过期之后的少量延迟数据会让计算基于无状态聚合，影响整体数据的正确性
- 解法1：source和sink同时进行处理
- 解法2：source处理利用kafka记录时间大于事件时间进行过滤（处理超长乱序数据）
- 解法3：sink处理利用回撤流的标记判断脏数据
- 收益1：保证少量延迟数据不会覆盖聚合数据
- 收益2：实现groupby延迟数据的侧输出

数据准确性实现点

数据过期

- **状态过期延迟数据导致聚合数据覆盖**
- 非HA下重试丢弃状态
- 用不足一天的状态去恢复任务

传递语意

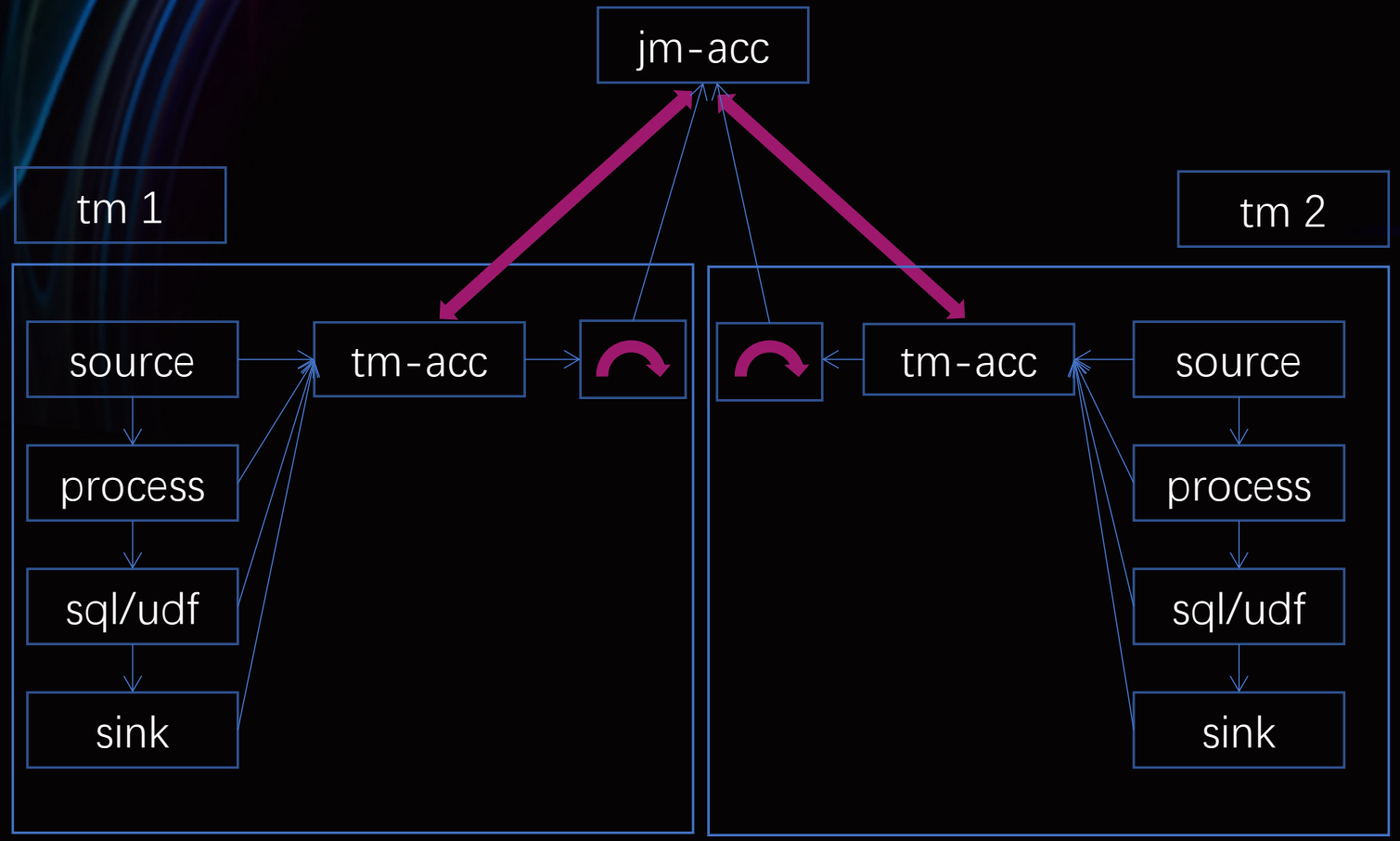
- 数据乱序 kafka keyby sink
- hdfs 精准一次sink
- doris 精准一次sink
- kv类型幂等sink

数据乱序

- 乱序时间导致窗口提前触发
- **Per-partition watermark问题**

05 稳定性主题

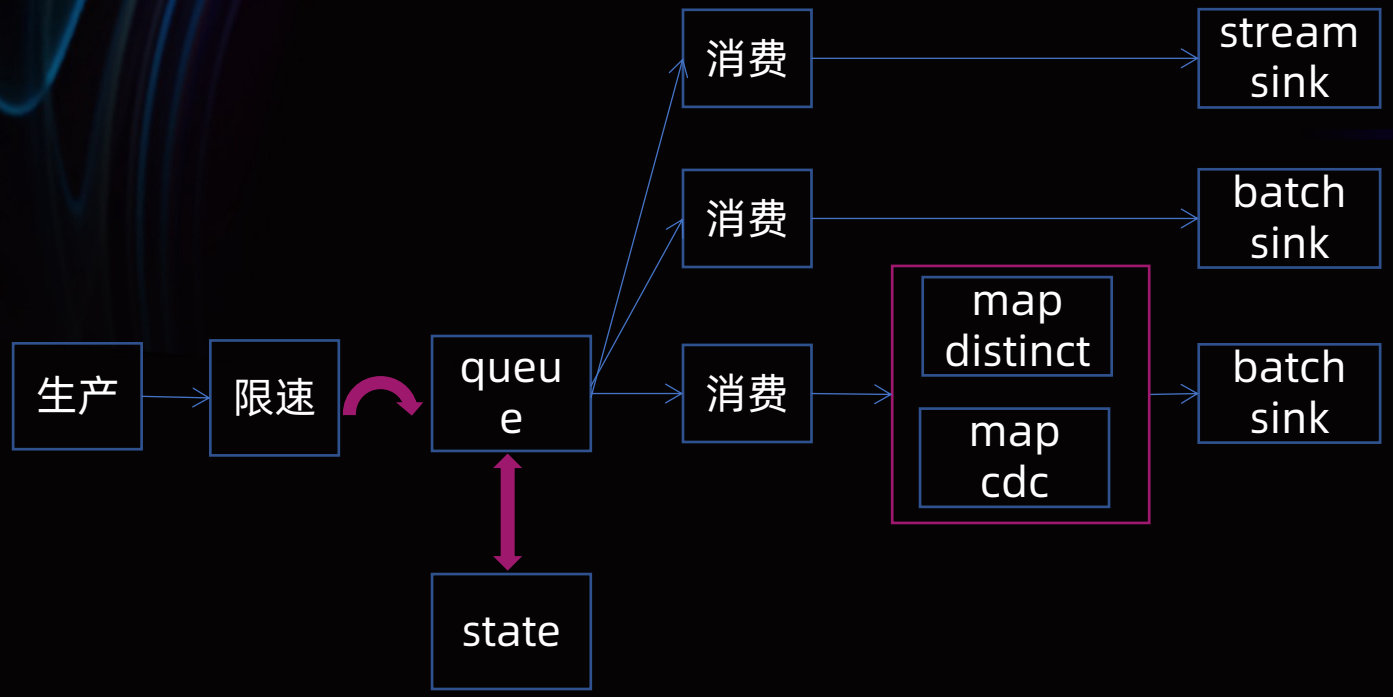
5.1 多场景脏数据容忍



累加器控制

- 背景：线上存在不同场景下1条脏数据让任务失败，用户通过手动增加过滤条件过滤脏数据
- 解法1：累加器集中的控制逻辑，超标任务退出
- 解法2：形成指标
- 收益：保证少量脏数据不影响任务正常运行

5.2 sink 稳定性



多维控制

- 背景：存在跨网络写obs，环境不稳定下，会影响cp稳定性；存在写OLAP类的存储，对写入有稳定性要求；queue 在 cp 的时候需要刷数据到外部存储，导致的cp耗时，cp卡住，小文件等问题
- 解法1：限速
- 解法2：支持queue模式，支持多线程消费
- 解法3：queue写state阈值控制
- 解法4：缓存去重，缓存cdc，batch发送

稳定性优化点

数据问题

- 数据波动，source/sink 限速
- 数据波动，状态增大场景，oom场景
- 脏数据

环境问题

- 动态重试
- kakfa 分区切换
- yarn label隔离
- hive 统一加分区，缓存分区
- 异步sink
- 跨网络写

用法问题

- 高版本
- jm ha
- **分场景使用状态后端**
- **单点恢复**
- 动态cp周期
- **unaligned cp**
- **numberOfTaskSlots client 共享**

06 未来展望

未来展望

资源

- flink on k8s
- new stateBackend
- 机器 均衡
- slot 均衡
- 细粒度资源

开发

- 更强的sql控制能力
- 丰富/增强connector
- 自定义sql算子
- 流批一体

周边

- cdc
- 数据湖
- 智能运维

FLINK FORWARD # ASIA 2022 ONLINE | 实时即未来

THANK YOU

谢 谢 观 看