# 构建批流一体数据集成平台的一致性语义保证

陈肃 · DataPipeline

Kafka X Flink Meetup 深圳 - 2019年08月31日

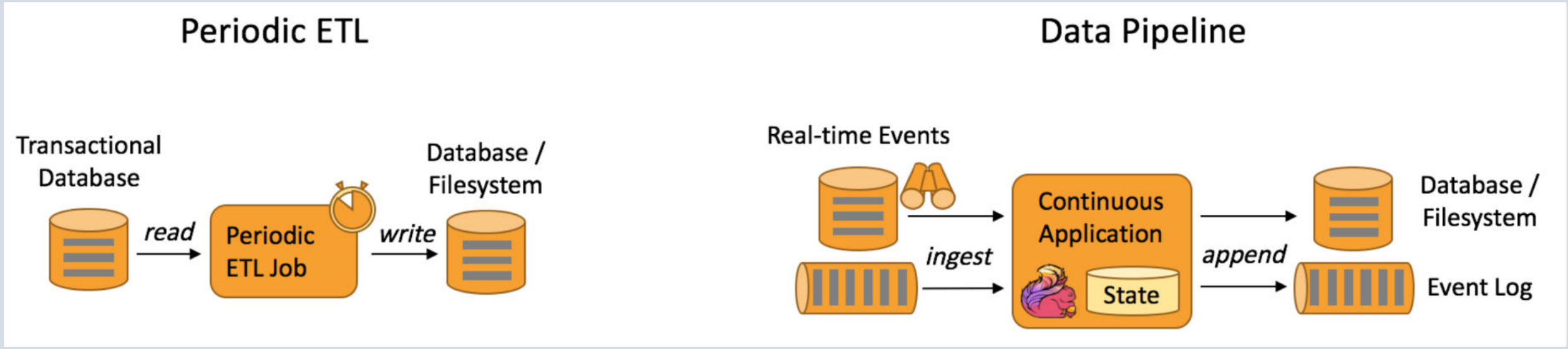
# Contents

## 目录 >>

# PART 01

## 批流一体架构

# 批和流是数据集成的两种应用形态




https://flink.apache.org/usecases.html

# 数据集成的基本问题

动态性

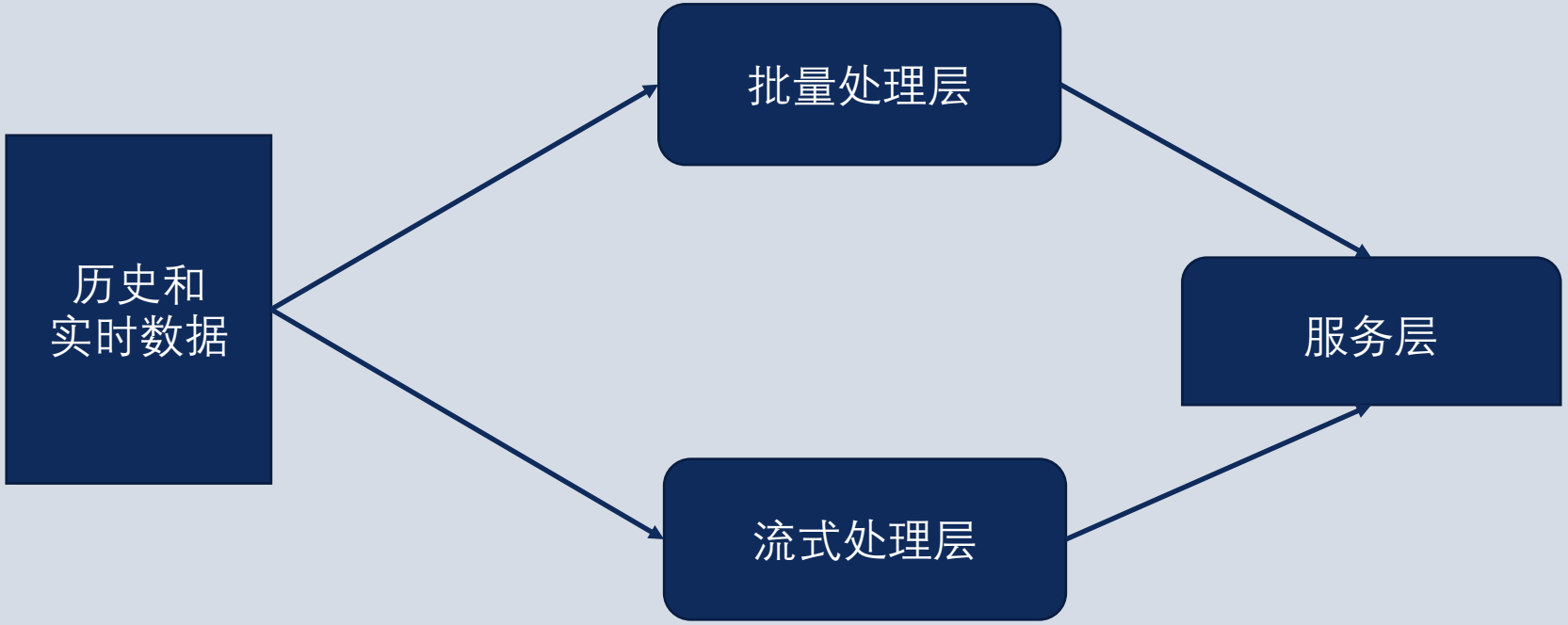
可伸缩性

一致性

容错性

异构性

# Lambda架构是批流一体化的必然要求




核心是按需使用批量和流式，以取得延时、吞吐和容错方面的平衡

- 批处理层：提供精确的批次数据视图
- 流处理层：提供（近）实时的数据视图
- 批流是逻辑上的分离，而非具体实现技术的分离

# 数据集成的Ad-Hoc模式

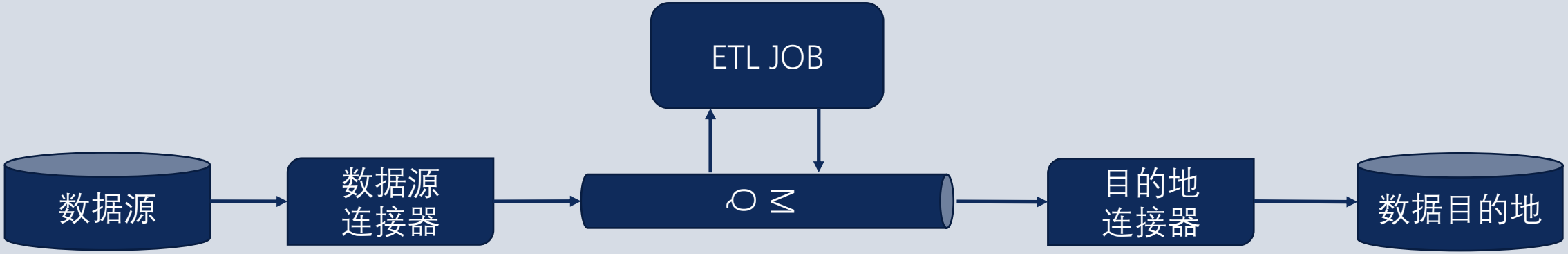

ETL JOB 封装所有的处理逻辑

- 从源端读取/注入数据
- 将结果写入目的地
- 有状态/无状态的转换

批流处理框架提供

- 可重用的源和目的地连接器
- Operator和DAG
- 分布式运行环境+容错

# 数据集成的MQ模式




- 数据源和目的地连接器独立于 ETL JOB, 并拥有独立的运行时
- 数据源和目的地连接器可以执行无状态的清洗和转换
- ETL JOB只与MQ进行交互：消费数据并写入转换后的数据

# DP选择MQ模式的理由

优点：

- 一次读取多次处理
- 一源对多目的地分发
- 重用开源的连接器
- 更加灵活的集成能力

缺点：

- MQ成为吞吐瓶颈
- 针对批需要边界消息
- 数据留存限制问题
- 重新同步的数据清理

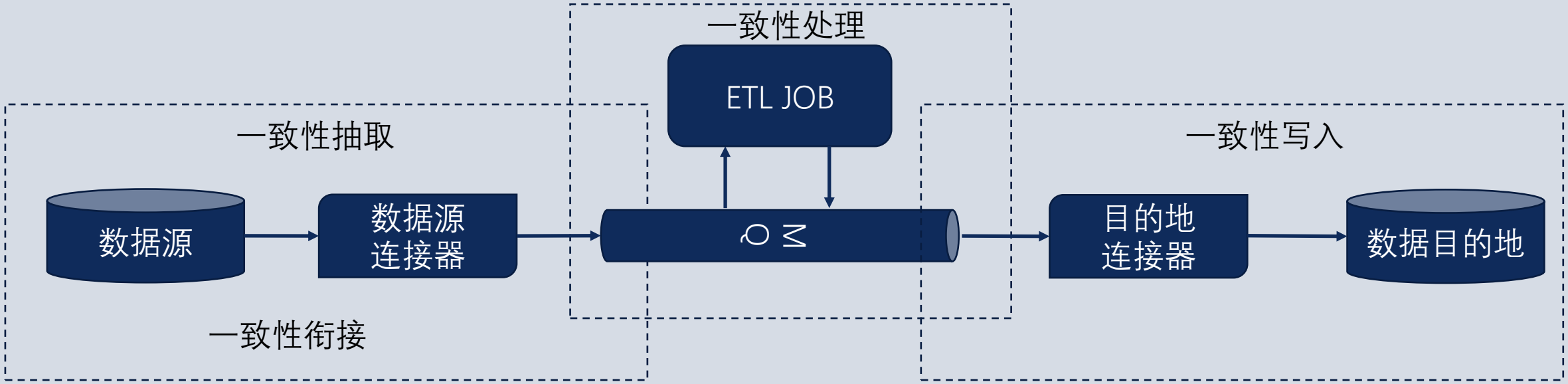
# PART 02

## 一致性语义保证

# 来自我们客户需求

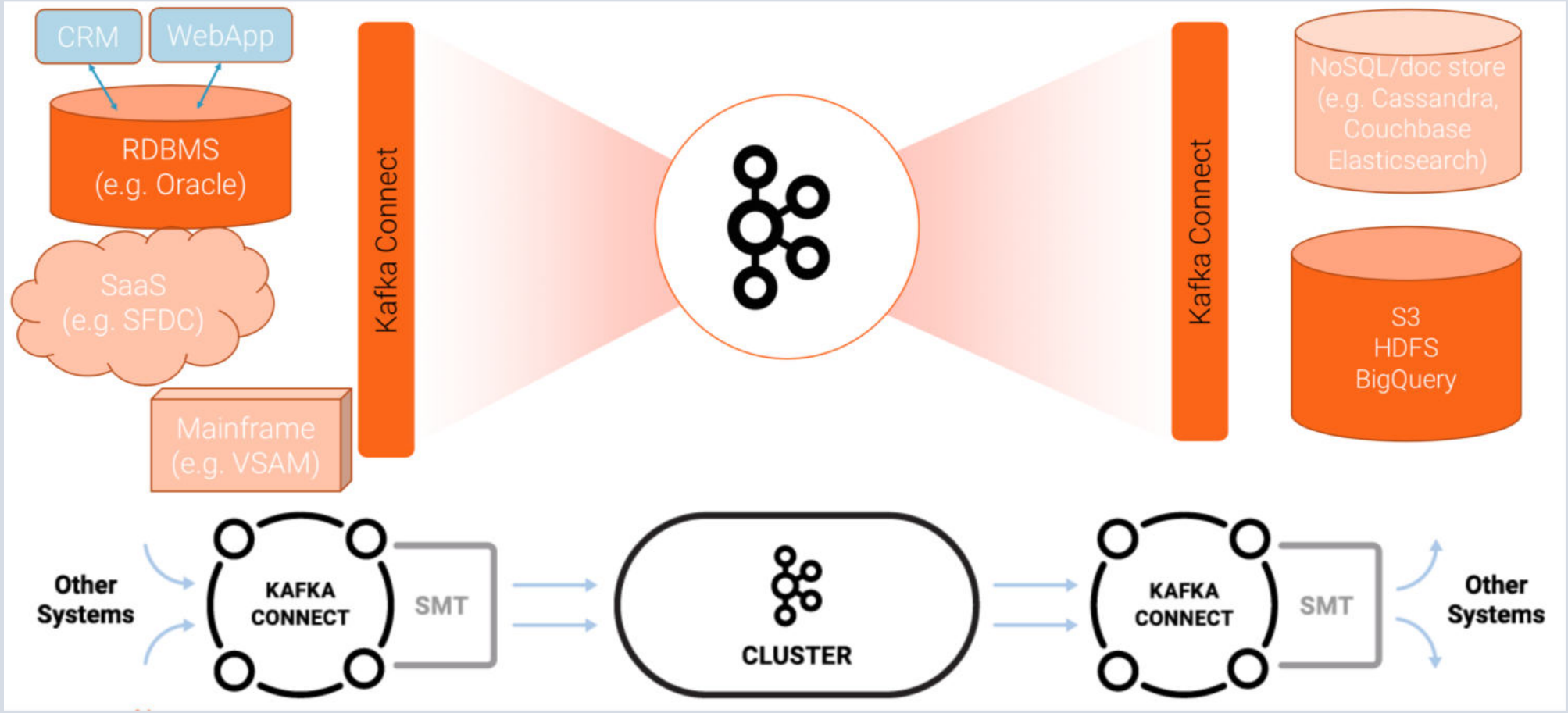
- 批量需要以事务方式完成同步
- 流式数据尽可能快的完成同步
- 批量和流式可能共存于一个JOB
- 按需灵活选择一致性语义保证
  - At Least Once
  - Exactly Once

# 数据一致性的链路视角



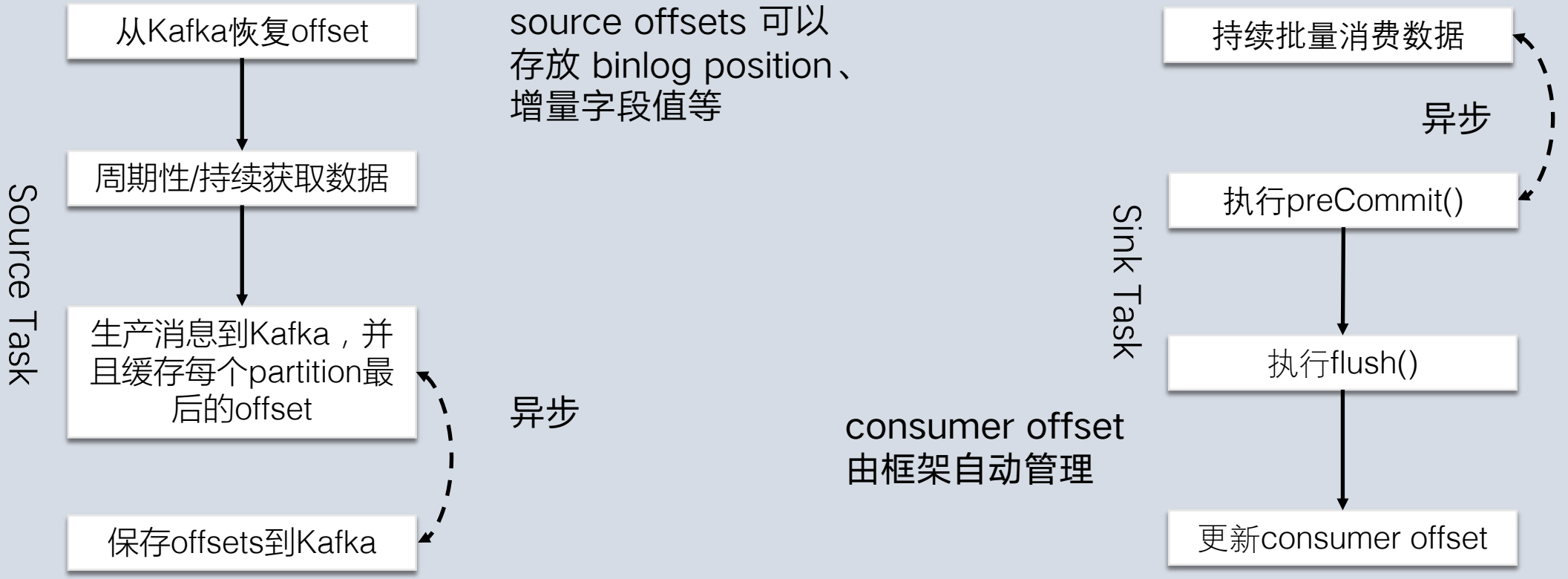

- 一致性抽取：数据和对应的offset以事务方式进入MQ
- 一致性处理：数据的消费、处理、回写MQ以事务方式进行
- 一致性写入：consumer offset和数据以事务方式写入目的地
- 一致性衔接：历史数据和流式数据，以及数据批次之间的无缝衔接

# Kafka Connect简介



# Kafka Connect的一致性保证

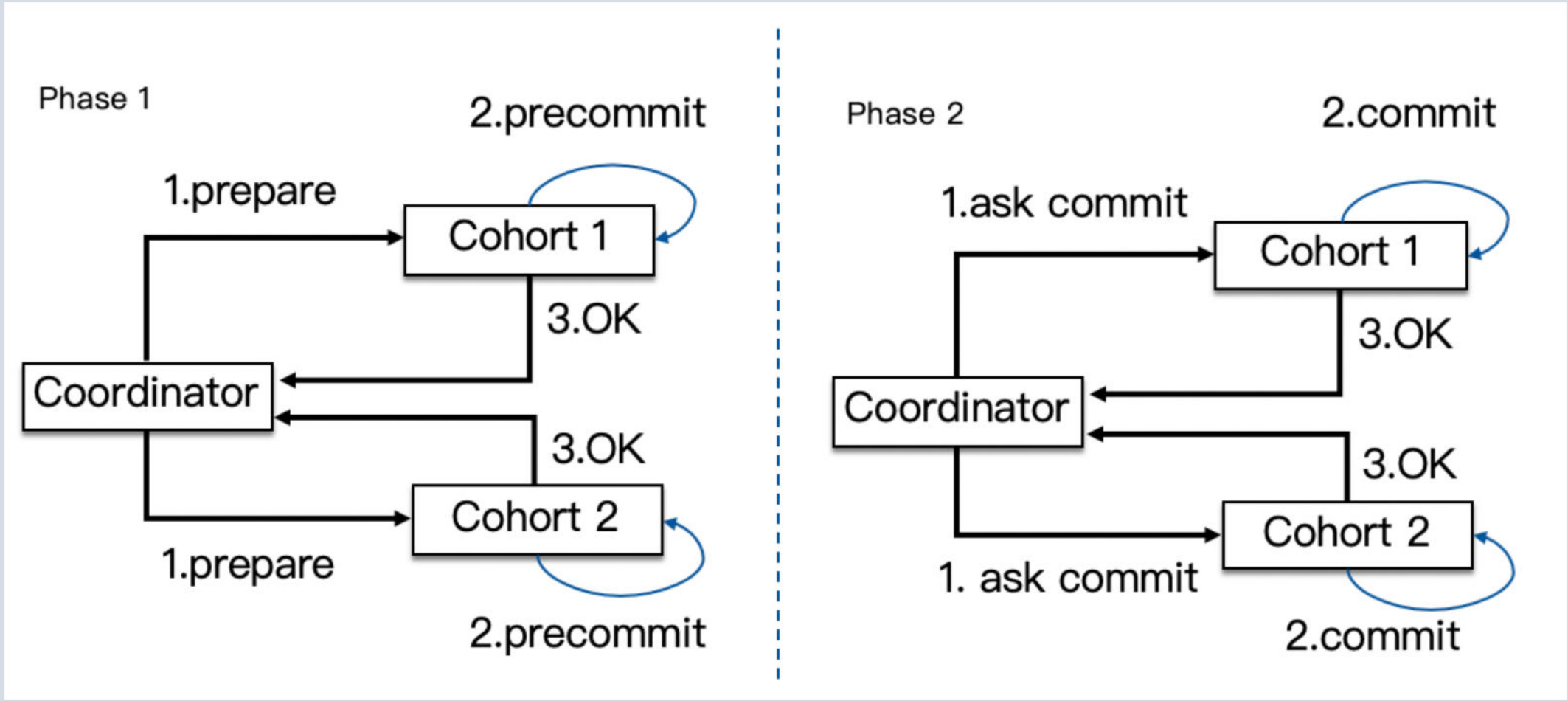
--原生只支持At Least Once



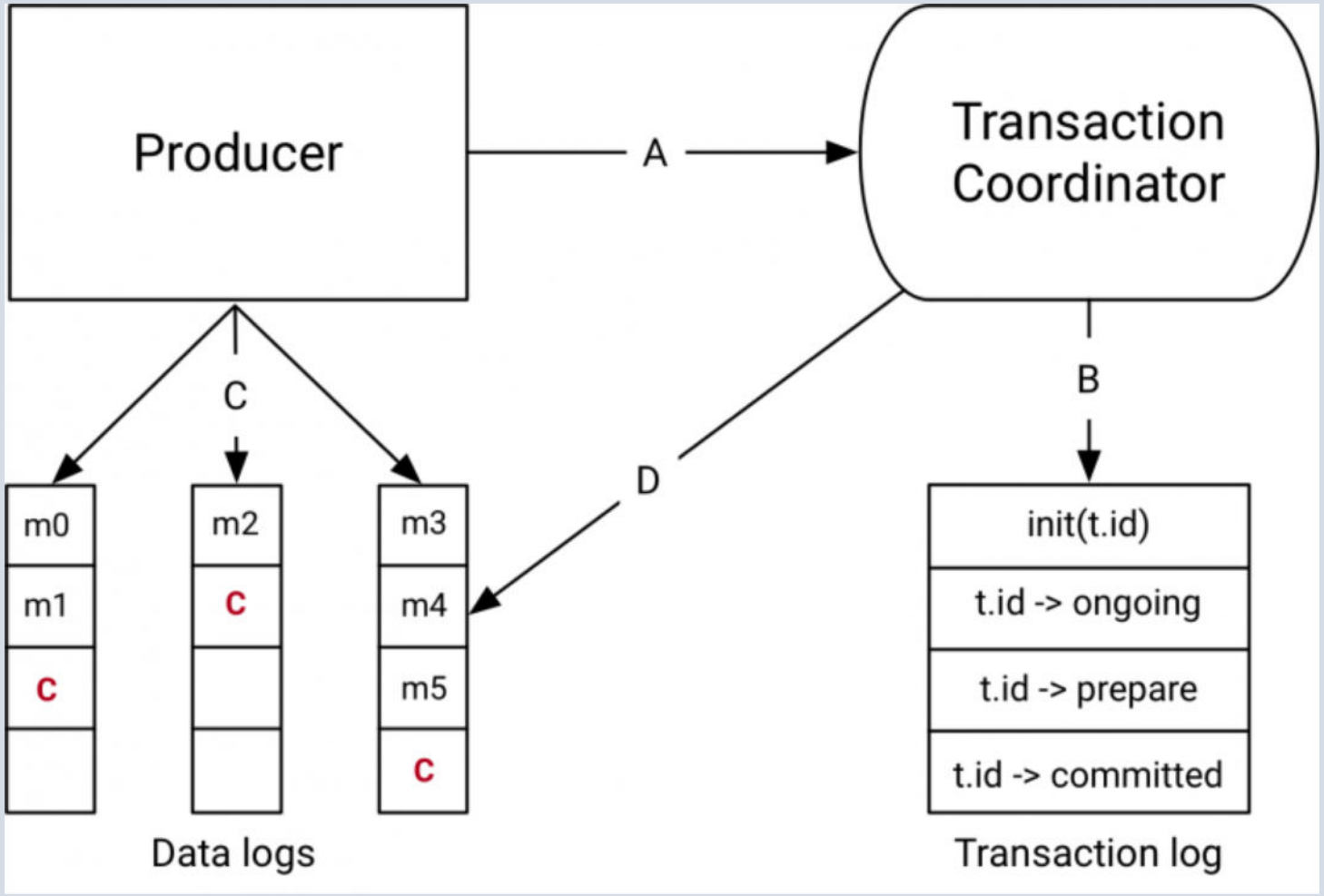
# PART 03

## DP的解决之道

# 二阶段提交协议


Phase 1
2.precommit
1.prepare
Cohort 1
3.OK
Coordinator
3.OK
Cohort 2
1.prepare
2.precommit
Phase 2
2.commit
1.ask commit
Cohort 1
3.OK
Coordinator
3.OK
Cohort 2
1. ask commit
2.commit

# Kafka事务消息原理



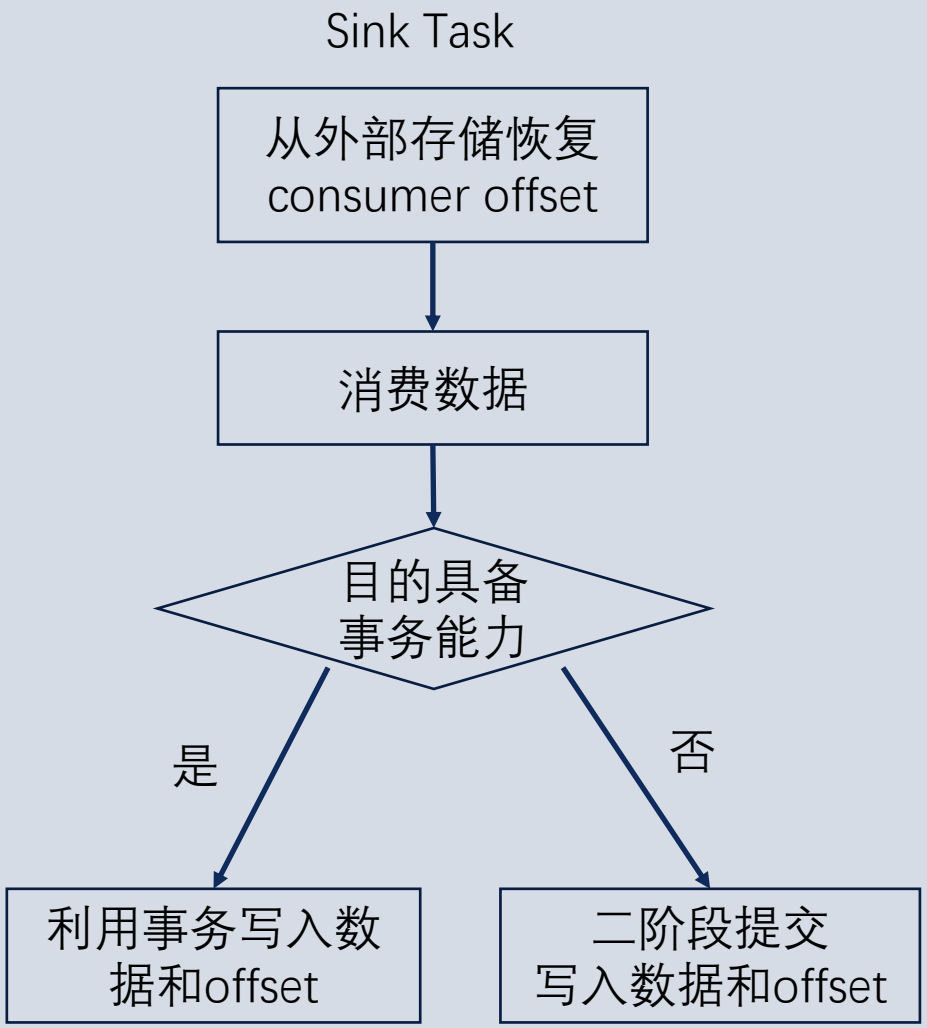

## 一致性抽取

- Source Connector 可以将一批数据和offset作为一个事务发送
- 下游不会看到这一批数据，直到offset也更新成功
- 需要对Kafka Connect的Source Worker进行改造

## 一致性处理

- 基于Kafka Streams构建ETL JOB

# 数据一致性写入




- 核心是将consumer offset管理从Kafka Connect框架中独立出来，实现事务一致性提交
- 类似Flink的TwoPhaseCommitSinkFunction，我们在Sink Task内定义了二阶段提交所需的接口方法，各个Sink Connector做相应的实现

# 数据一致性衔接

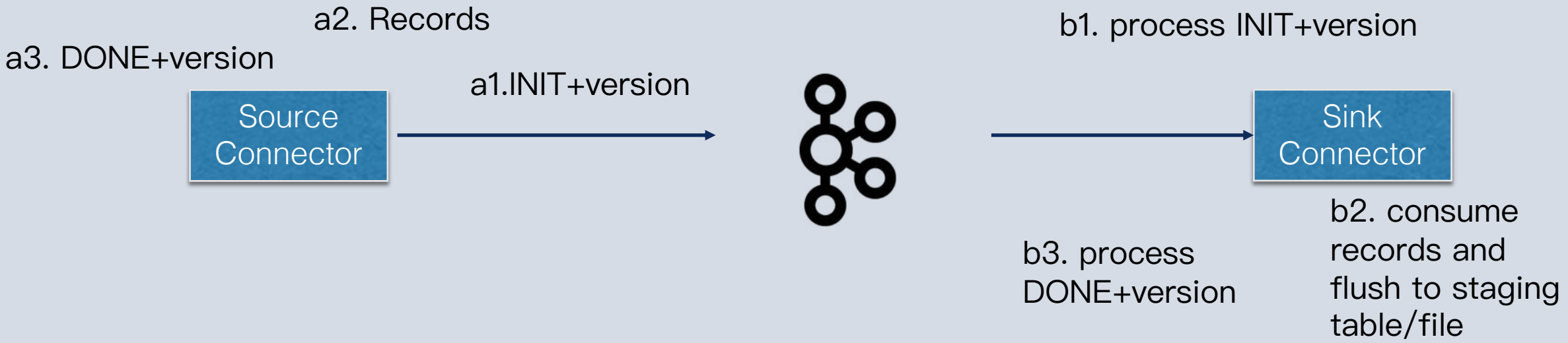
- 全量数据与日志的起始值需要在同一个事务中被获取
  例如：mysql的 START TRANSACTION WITH CONSISTENT SNAPSHOT

- 灵活的增量表达式需要被提供，以避免漏读数据

  SELECT * FROM `table1` WHERE `_id` > 'last_max(_id)'

  SELECT * FROM `table1` WHERE `date` > 'last_max(date)' AND `date` <= CURDATE()-1

# 批量同步的事务实现




- version全局递增，用于追踪批次
- Sink端收到INIT消息后，清空目标的staging表
- 隶属于同一批次的消息首先被写入staging表/文件
- Sink收到DONE消息后，执行 move/rename/copy操作
- 为保证数据一致性，源端读取过程中发生任何异常重新读取

# PART 04

## 问题和思考

# 遇到过的一些问题

| 问题 | 现象和影响 | 解决方法 |
| --- | --- | --- |
| 反压<br>back pressure | Source和Sink完全解耦，源端读取过快，大量数据在Kafka中积压，超过了队列允许的最大值，造成数据丢失或磁盘空间占满 | 通过Manager监控消费的lag，对源端进行限速，使生产消费速度匹配 |
| 资源隔离 | Connect Worker集群无法对task进行资源预留，多个task并行运行会相互影响。Worker的rest接口是队列式的，单个集群任务过多会导致启停缓慢 | 利用外部资源调度框架，例如K8s进行worker节点管理；通过路由规则将不同优先级任务运行在不同的worker集群上，实现预分配和共享资源池的灵活配置 |
| rebalance | 2.3版本以前，Kafka Connect的task rebalance采用stop-the-world模式，牵一发动全身 | 升级到2.3版本，已经提供具有粘性的rebalance，大幅优化效率 |
| 数据丢失 | 2 replicas、ack=1、unclean.leader.election.enable=false<br>依然在客户环境发生过一次leader切换导致的数据丢失 | relicas>=2、ack=all，unclean.leader.election.enable=false<br>启用lz4压缩提升吞吐 |
| 消费停止 | Consumer打开Partition后就没有后继消费动作；有时运行中的任务自己就停止消费数据了，造成数据同步中断。 | 偶现不易定位原因。通过独立的监控子系统，及时发现僵死的consumer，配置告警+重启策略 |

# 未来演进路线

## 针对大批量同步的去消息中间件化

- 目的：提升性能、降低重新同步的清理代价
- 方法：Source Connector同时实现Sink的逻辑，用内存队列替代MQ

## 采用更加灵活的Runtime

- 目的：利用成熟框架实现预分配资源池和共享资源池的统一管理
- 方法：利用K8s优化Kafka Connect的Worker节点管理和任务调度、尝试使用Flink

## 数据质量管理

- 目的：对于数据同步的一致性进行后校验，识别数据流中的模式异常
- 方法：引入支持批+流模式的数据质量框架，例如Apache Griffin，扩展模型层


THANKS
Kafka X Flink Meetup
SHENZHEN