Apache Flink TiDB | COMMUNITY

Flink CDC 详解

徐榜江(雪尽)
Flink Contributor
阿里巴巴-高级开发工程师

Apache Flink x TiDB Meetup · 北京站

CONTENT

目录 >>

#1 CDC 概述

CDC 技术

CDC 的全称**是 Change Data Capture**，在广义的概念上，只要能捕获数据变更的技术，我们都可以称为 CDC 。通常我们说的 CDC 技术主要面向**数据库**的变更，是一种用于捕获数据库中数据变更的技术。

CDC 技术应用场景非常广泛：

- 数据同步，用于备份，容灾
- 数据分发，一个数据源分发给多个下游
- 数据采集(**E**)，面向数据仓库/数据湖的 **ETL** 数据集成

CDC 技术

CDC的技术方案非常多，目前业界主流的实现机制的可以分为两种：

- **基于查询的 CDC**
  - 离线调度查询作业，批处理
  - 无法保障数据一致性
  - 不保障实时性
- **基于日志的 CDC**
  - 实时消费日志，流处理
  - 保障数据一致性
  - 提供实时数据

常见开源 CDC 方案比较

	Flink CDC	Debezium	DataX	Canal	Sqoop	kettle	Oracle Goldengate
CDC 机制	日志	日志	查询	日志	查询	查询	日志
增量同步	✓	✓	✗	✓	✗	✗	✓
断点续传	✓	✓	✗	✓	✗	✗	✓
全量同步	✓	✓	✓	✗	✓	✓	✓
全量+增量	✓	✓	✗	✗	✗	✗	✓
架构	分布式	单机	单机	单机	分布式	分布式	分布式
Transformation	☆☆☆☆☆	☆☆	☆☆	☆☆	☆☆	☆	☆
生态	☆☆☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆	☆☆	☆☆☆

#2 Flink–CDC

Flink Changelog Stream

Append Stream

INSERT, 1001, Sally
INSERT, 1004, Anne
INSERT, 1005, George
UPDATE, 1004, Danny
INSERT, 1006, Edward
DELETE, 1004, Danny

Interpret Changelog

Changelog Stream

I	1001, Sally
I	1004, Anne
I	1005, George
U	1004, Danny
I	1006, Edward
D	1004, Danny

capture changes

materialize changes

PK

id	name
1001	Sally
1005	George
1006	Edward

Updated Dynamic Table

Emit Changelog

Append Stream

INSERT, 1001, Sally
INSERT, 1004, Anne
INSERT, 1005, George
UPDATE, 1004, Danny
INSERT, 1006, Edward
DELETE, 1004, Danny

Flink Changelog Stream

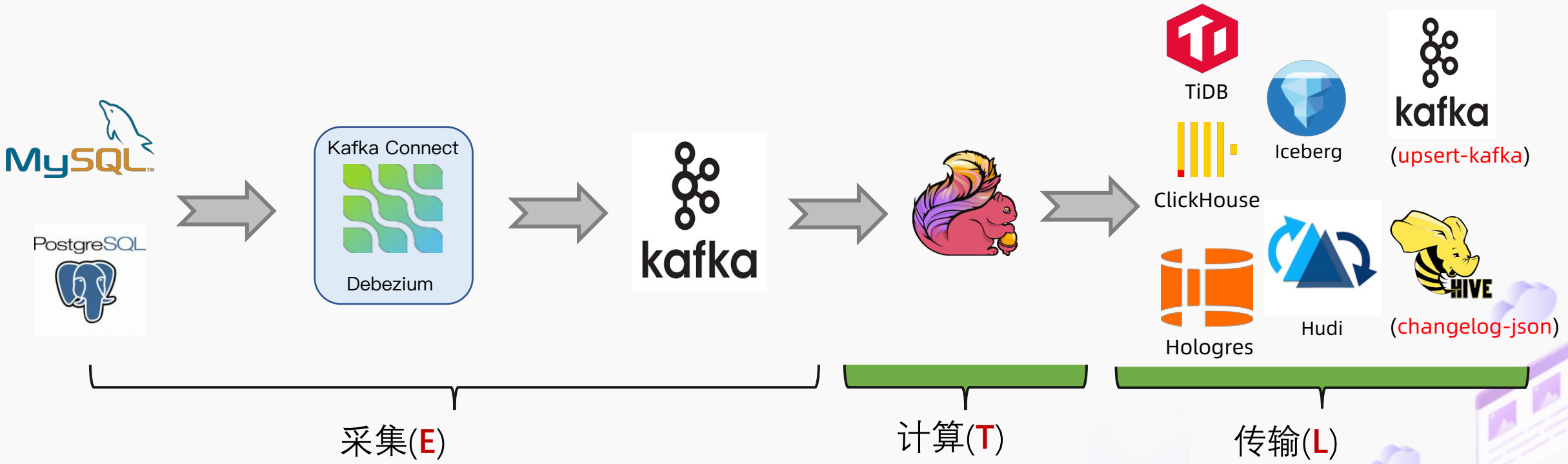
Debezium 数据格式

```
{    "op": "c",
     "before": null,
     "after": {
       "id": 1004,
       "name": "Anne"}}

{    "op": "u",
     "before": {
       "id": 1004,
       "name": "Anne"},
     "after": {
       "id": 1004,
       "name": "Leo"}}

{
     "op": "d",
     "before": {
       "id": 1004,
       "name": "Leo"}},
     "after": null}
```

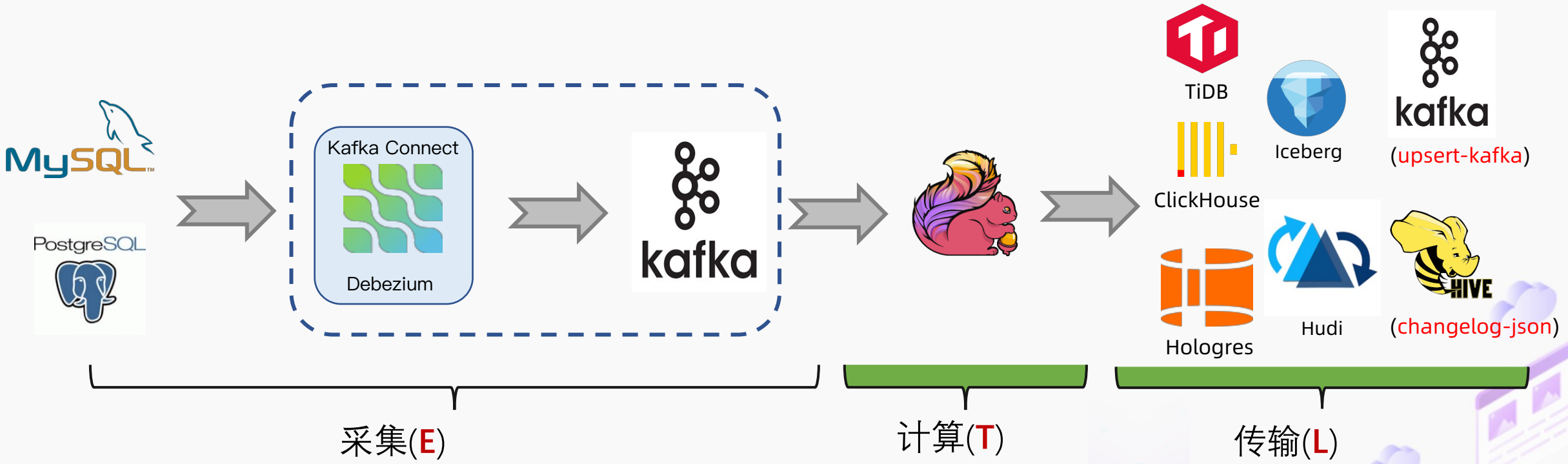
Flink Changelog 数据结构

```
public enum RowKind {
    INSERT,
    UPDATE_BEFORE,
    UPDATE_AFTER,
    DELETE;
}

public interface RowData {
    RowKind getRowKind();
    void setRowKind(RowKind kind);

    boolean getBoolean(int pos);
    byte getByte(int pos);
    ...
}
```

传统 CDC ETL 分析

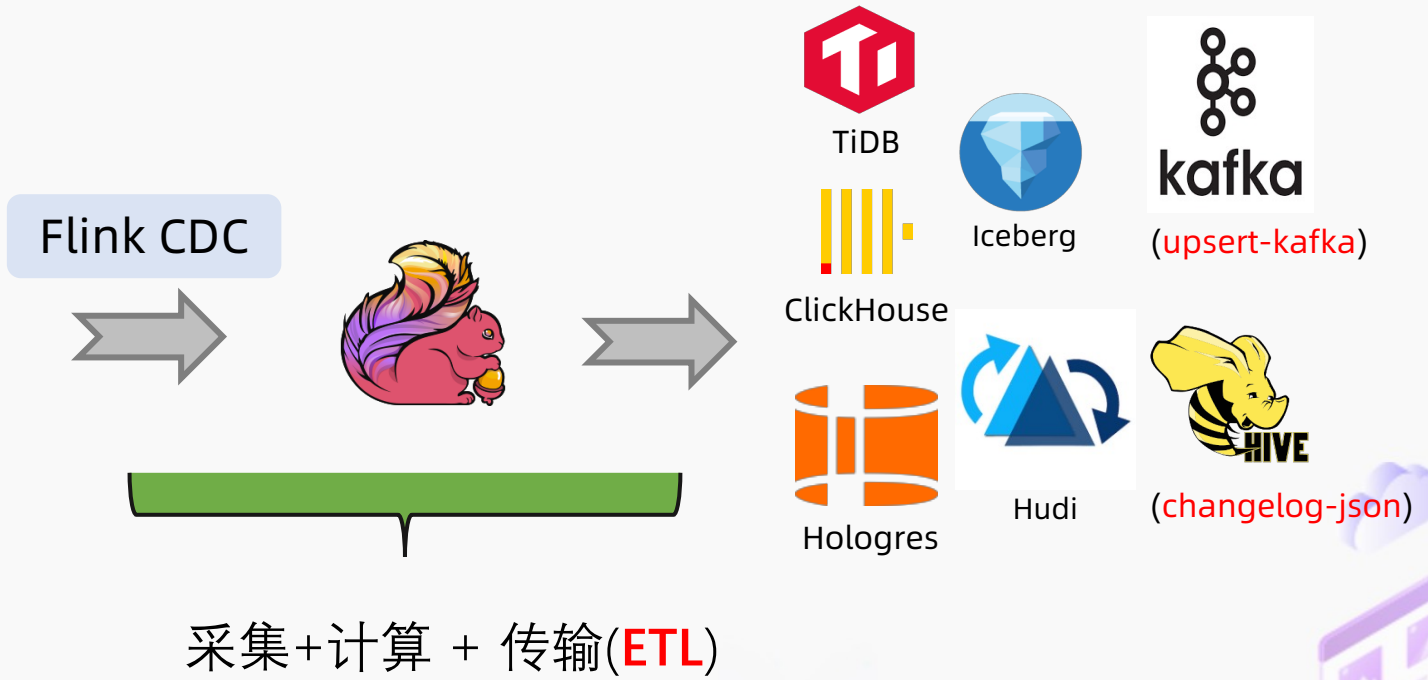

MySQL
PostgreSQL
Kafka Connect
Debezium
kafka
TiDB
ClickHouse
Iceberg
Hologres
Hudi
kafka
(upsert-kafka)
HIVE
(changelog-json)
采集(E)
计算(T)
传输(L)

传统 CDC ETL 分析


MySQL
PostgreSQL
Kafka Connect
Debezium
kafka
TiDB
ClickHouse
Iceberg
kafka
(upsert-kafka)
Hologres
Hudi
HIVE
(changelog-json)
采集(E)
计算(T)
传输(L)

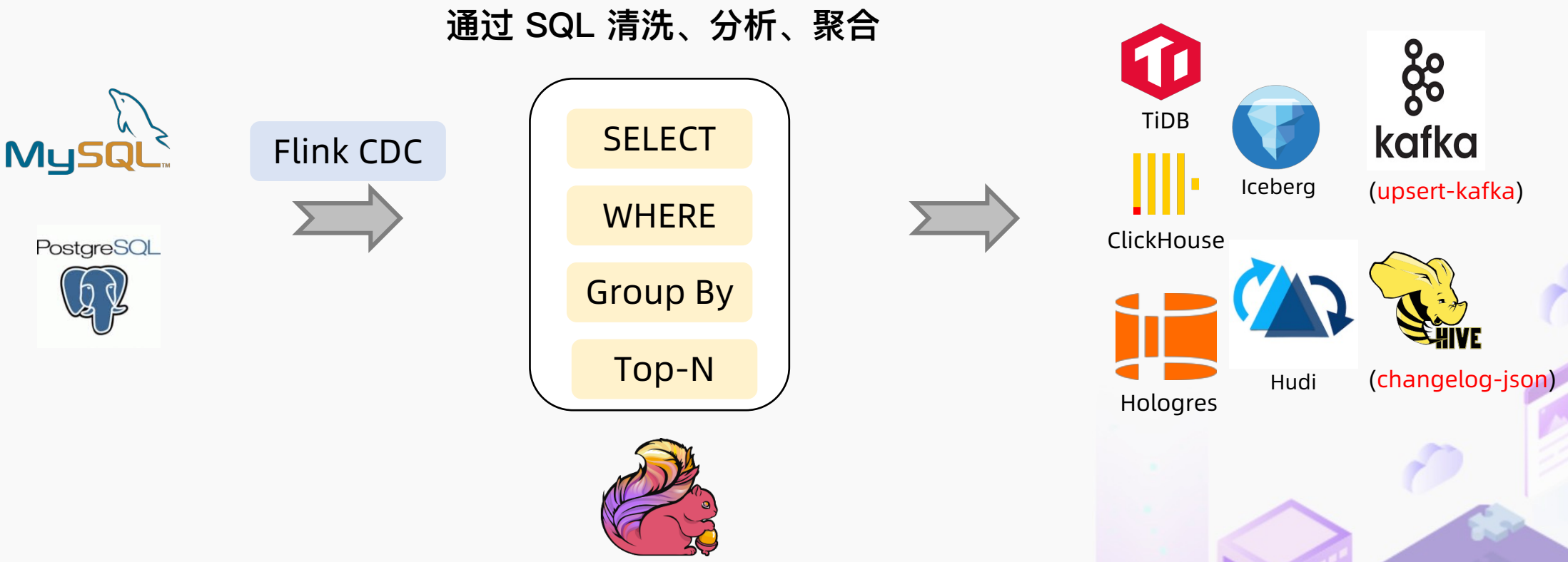
基于 Flink CDC 的 ETL 分析

```sql
-- 定义 MySQL 中的 products 源表
CREATE TABLE mysql_products (
  id BIGINT, name STRING, ...
) WITH (
  'connector'='mysql-cdc',
  ...
);

-- 定义 MySQL 中的 orders 源表
CREATE TABLE mysql_orders (
  id BIGINT, product_id BIGINT, ...
) WITH (
  'connector'='mysql-cdc',
  ...
);

-- 定义 tidb 结果表
CREATE TABLE tidb_orders (
  ...
) WITH (
  'connector'='jdbc',
  'url'='jdbc:mysql://tidb:4000/test'
  ...
);

-- 实时关联 MySQL 中的订单和产品信息，输出到 tidb 做 OLAP 分析
INSERT INTO tidb_orders
SELECT *
FROM mysql_orders AS o LEFT JOIN mysql_products AS p ON o.product_id = p.id;
```

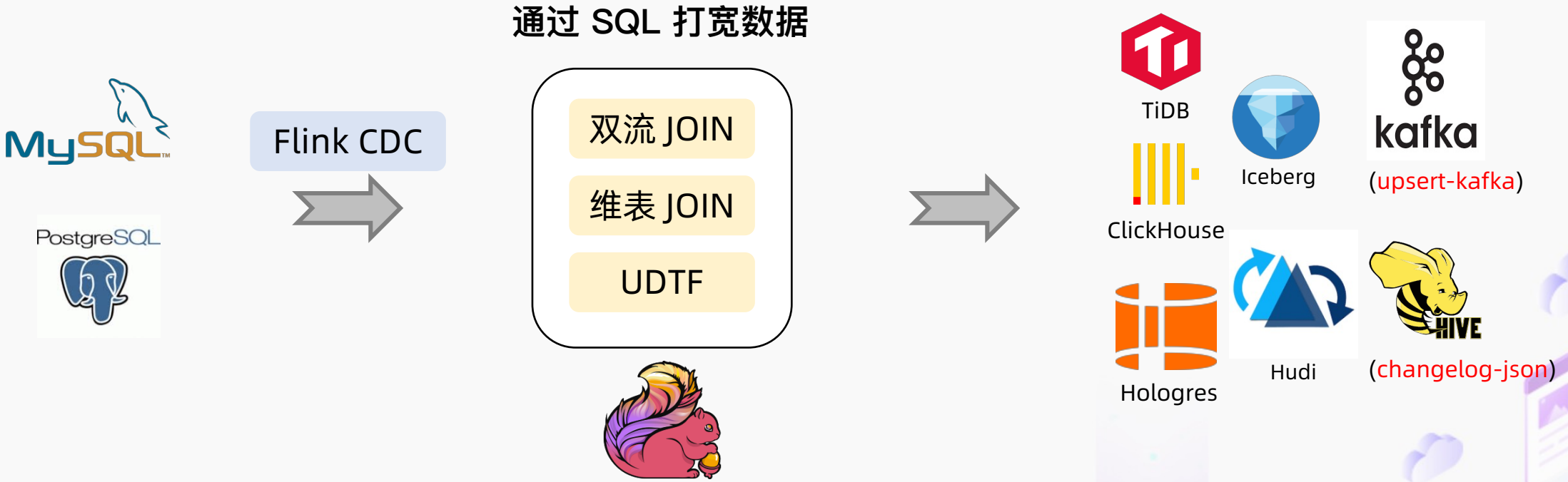


采集+计算 + 传输(ETL)

基于 Flink CDC 的聚合分析



基于 Flink CDC 的数据打宽



Flink CDC 里程碑

2020.07.07
第一个commit
By 云邪

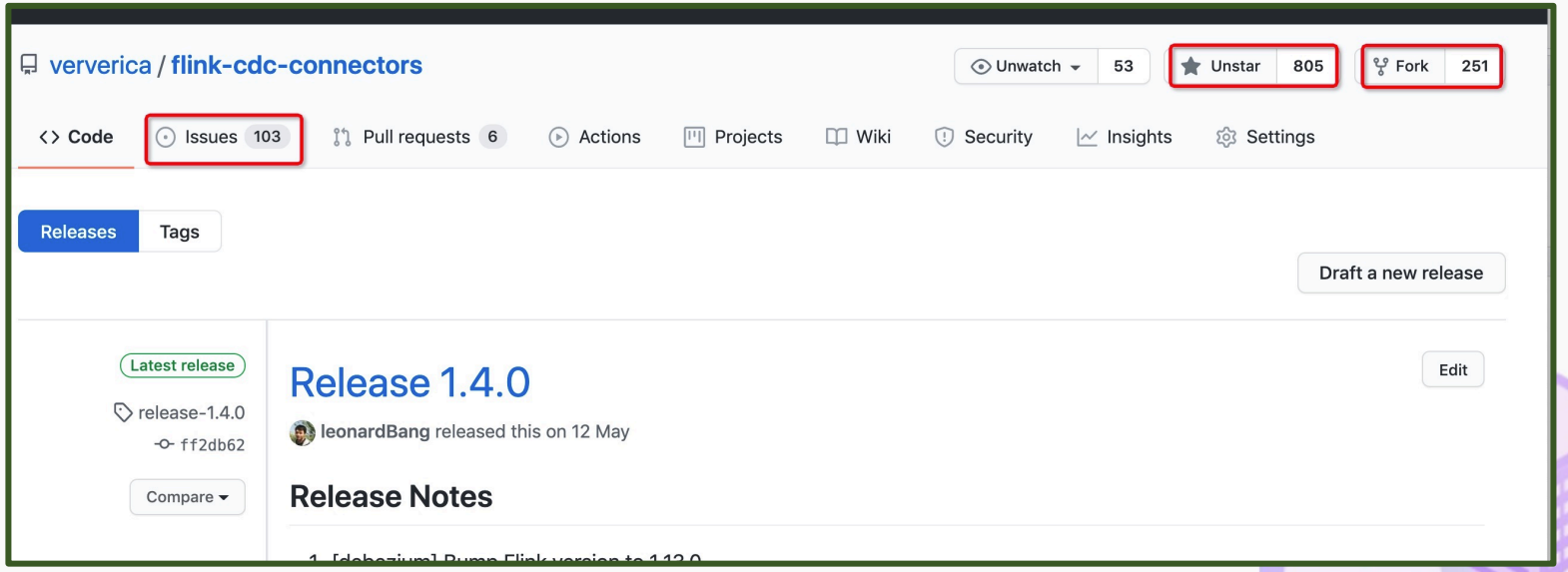
2020.07.18
支持MySQL-CDC

2020.07.31
支持postgres-cdc

2021.02.27
发布1.2版本

2021.03.29
发布1.3版本

2021.05.12
发布1.4版本



#3 Flink CDC 2.0 设计详解

Flink CDC 痛点

- **一致性通过加锁保证**

  Debezium 在保证数据一致性时，需要对读取的库或表加锁，全局锁可能导致数据库hang住，表级锁会锁住加锁的表(无法更新)， **DBA 一般不给锁权限**。

- **不支持水平扩展**

  Flink CDC 目前只支持单并发，在全量阶段读取阶段，如果表非常大(亿级别)，读取时间都在 **小时** 级别

- **全量读取阶段不支持 checkpoint**

  CDC 读取分为两个阶段，全量读取和 增量读取， 目前全量读取阶段是不支持 checkpoint 的，fail 后需要重新读取

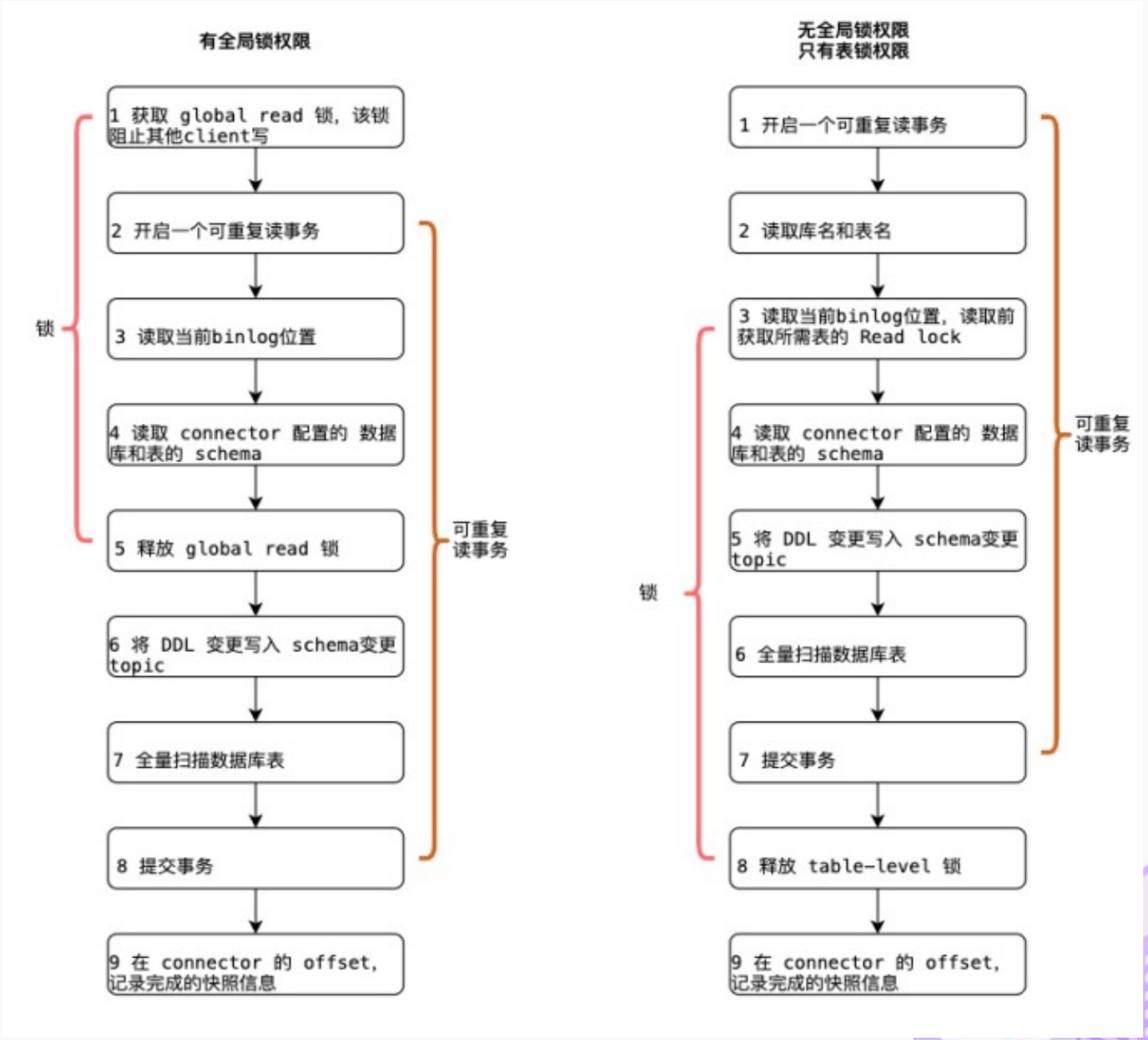
Debezium 锁 分析

Flink CDC 底层封装了 Debezium， Debezium 同步一张表分为两个阶段：

- 全量阶段：查询当前表中所有记录
- 增量阶段：从 binlog 消费变更数据

加锁发生在全量阶段，目的是为了确定增量阶段的初始位点，保证增量 + 全量实现一条不多，一条不少，保证数据一致性

Debezium 锁 分析



Debezium 锁 分析

FLUSH TABLES WITH READ LOCK

- 该命令等待所有正在进行的 update 完成，同时阻止所有新来的 update。
- 该命令执行成功前必须等待所有正在运行的 select 完成，所有等待执行的 update 会等待的更久。更坏的情况是，在等待正在运行 select 完成时，DB 实际上处于不可用状态，即使是新加入的 SELECT 也会被阻止。这是 MySQL Query Cache 机制。
- 该命令阻止其他事务 commit。

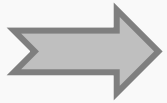
结论：加锁时间是不确定的，极端情况会 hang 住数据库

Percona 文章：
https://www.percona.com/blog/2014/03/11/introducing-backup-locks-percona-server-2/

Flink CDC 2.0 设计

设计目标

- 无锁
- 水平扩展
- 支持 checkpoint



- 借鉴 Netflix 的 DBlog paper[1]，全程无锁
- 基于 FLINK FLIP-27 Source 实现[2]，架构更加优雅

[1] :https://arxiv.org/pdf/2010.12597v1.pdf
[2] : https://cwiki.apache.org/confluence/display/FLINK/FLIP-27%3A+Refactor+Source+Interface

Flink CDC 2.0 设计

无锁算法

Algorithm 1: Watermark-based Chunk Selection

Input: table

(1) pause log event processing

lw := uuid(), hw := uuid()

(2) update watermark table set value = lw

(3) chunk := select next chunk from table

(4) update watermark table set value = hw

(5) resume log event processing

inwindow := *false*

// other steps of event processing loop

while *true* **do**

  e := next event from changelog

  if *not inwindow* **then**

    if *e is not watermark* **then**

      append e to outputbuffer

    else if *e is watermark with value lw* **then**

      inwindow := *true*

  else

    if *e is not watermark* **then**

(6) **if** *chunk contains e.key* **then**

        remove e.key from chunk

      append e to outputbuffer

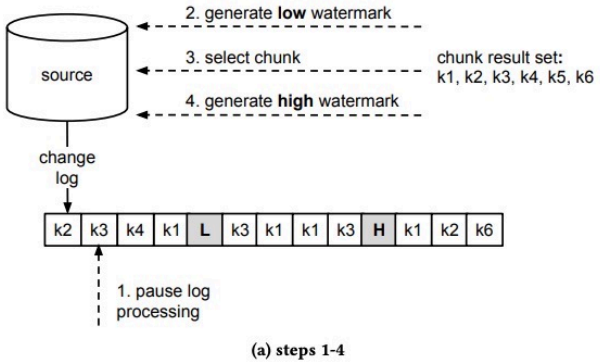
    else if *e is watermark with value hw* **then**

(7) **for** *each row in chunk* **do**

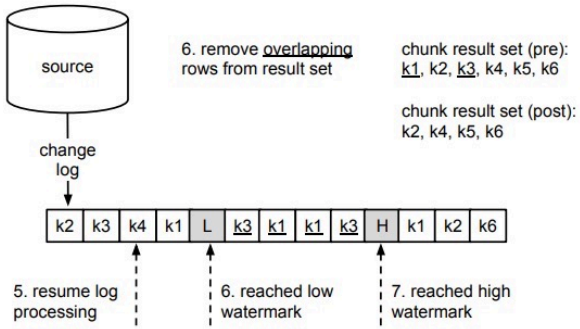
        append row to outputbuffer

  // other steps of event processing loop

  ...

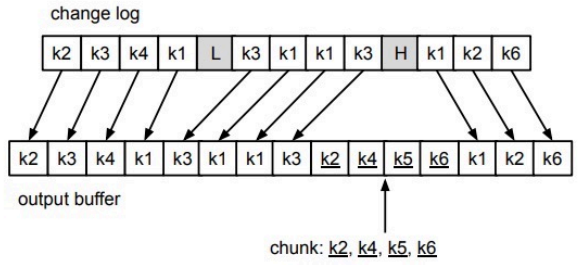


(a) steps 1-4



(b) steps 5-7

Figure 3: Watermark-based Chunk Selection



Flink CDC 2.0 设计

Chunk 切分

ID(PK)	C1	C2
K1	A	
K2	B	
K3	C	
…	..	
K100	X	
K101	Y	
K103	X	
K104	Z	

Chunk-0

Chunk-1

Chunk-14

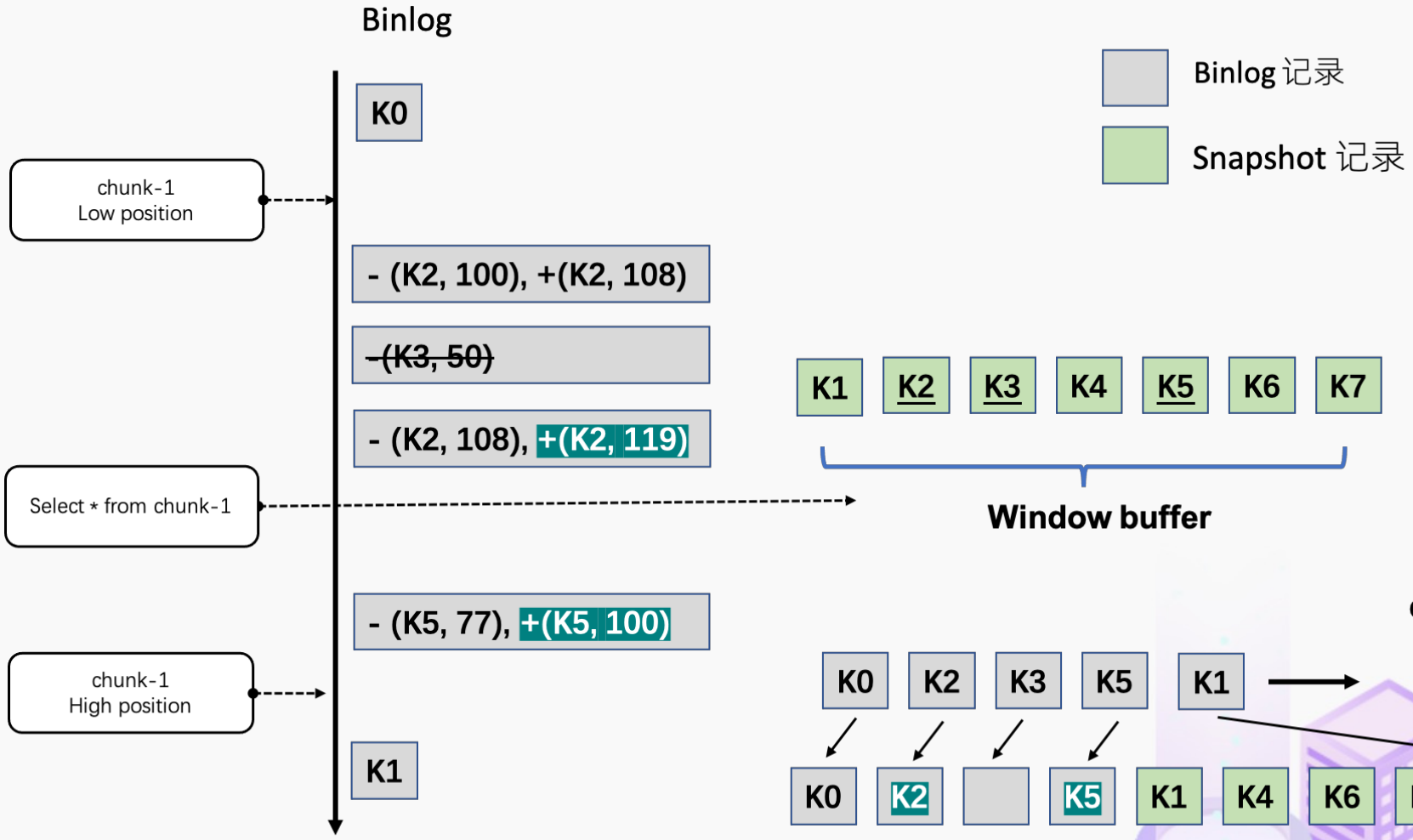
Chunk-15

- select max(id) from T1
  => K104
- select max(id) from T1 where id <= K104 order by id asc limit 10
  => K3
- select max(id) from T1 where id > K101 and id <= K104 order by id asc limit 10
  => K104

Chunk-ID	key range
Chunk-0	(null, k1)
Chunk-1	[k1, k10)
….	
Chunk-14	[k100, K104)
Chunk-15	[K104, null)

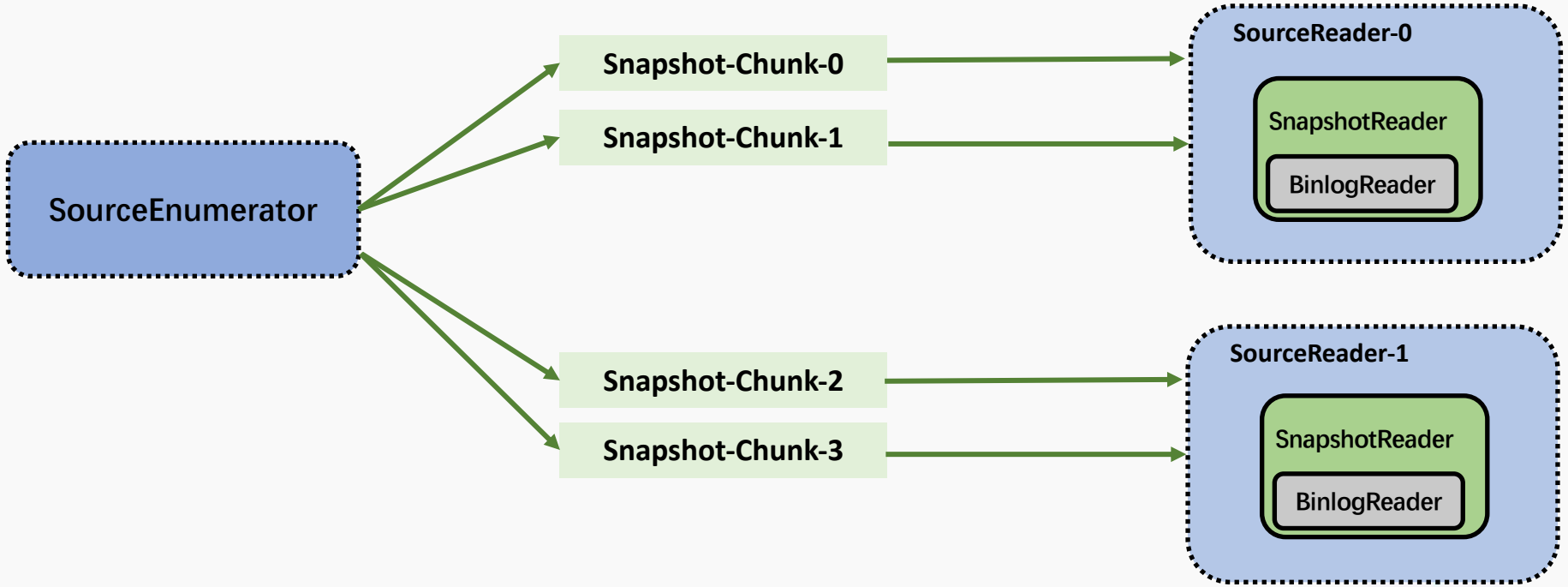
Flink CDC 2.0 设计

Chunk 读取



Flink CDC 2.0 设计

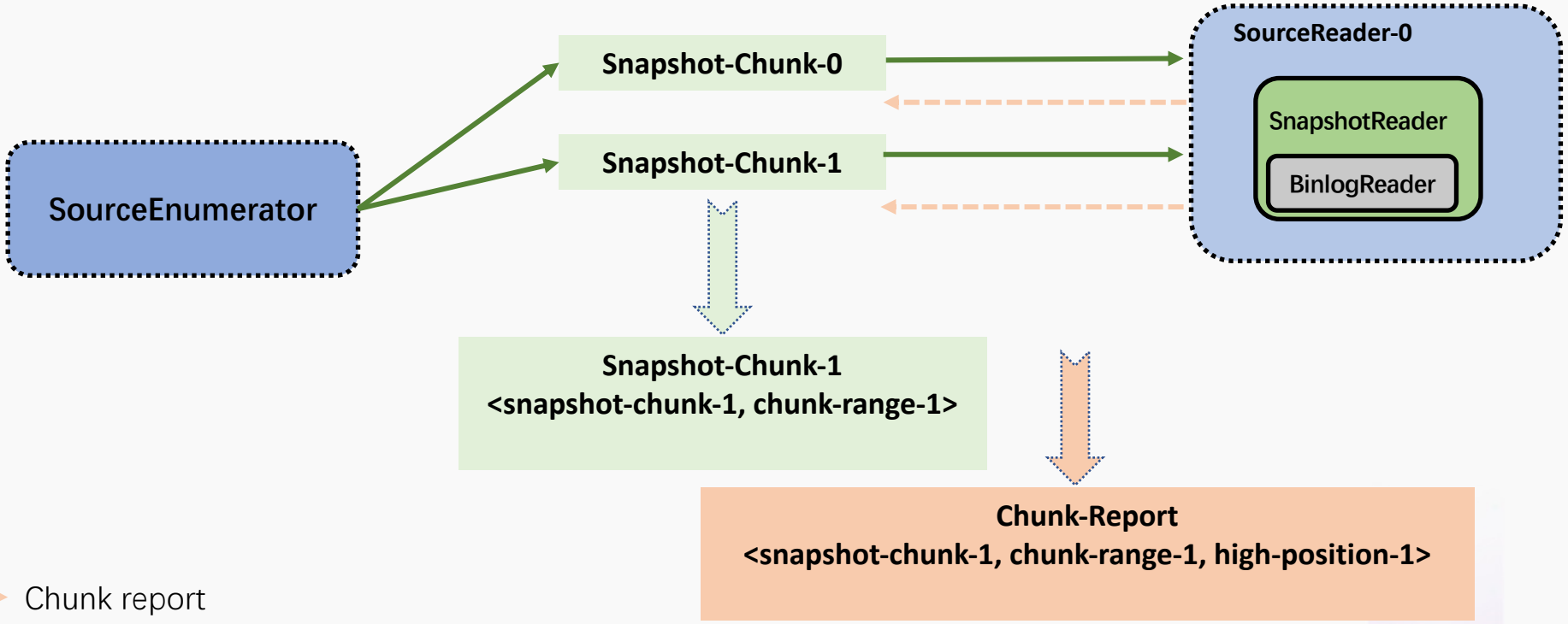
Chunk 分配： snapshot-chunk



Snapshot chunk

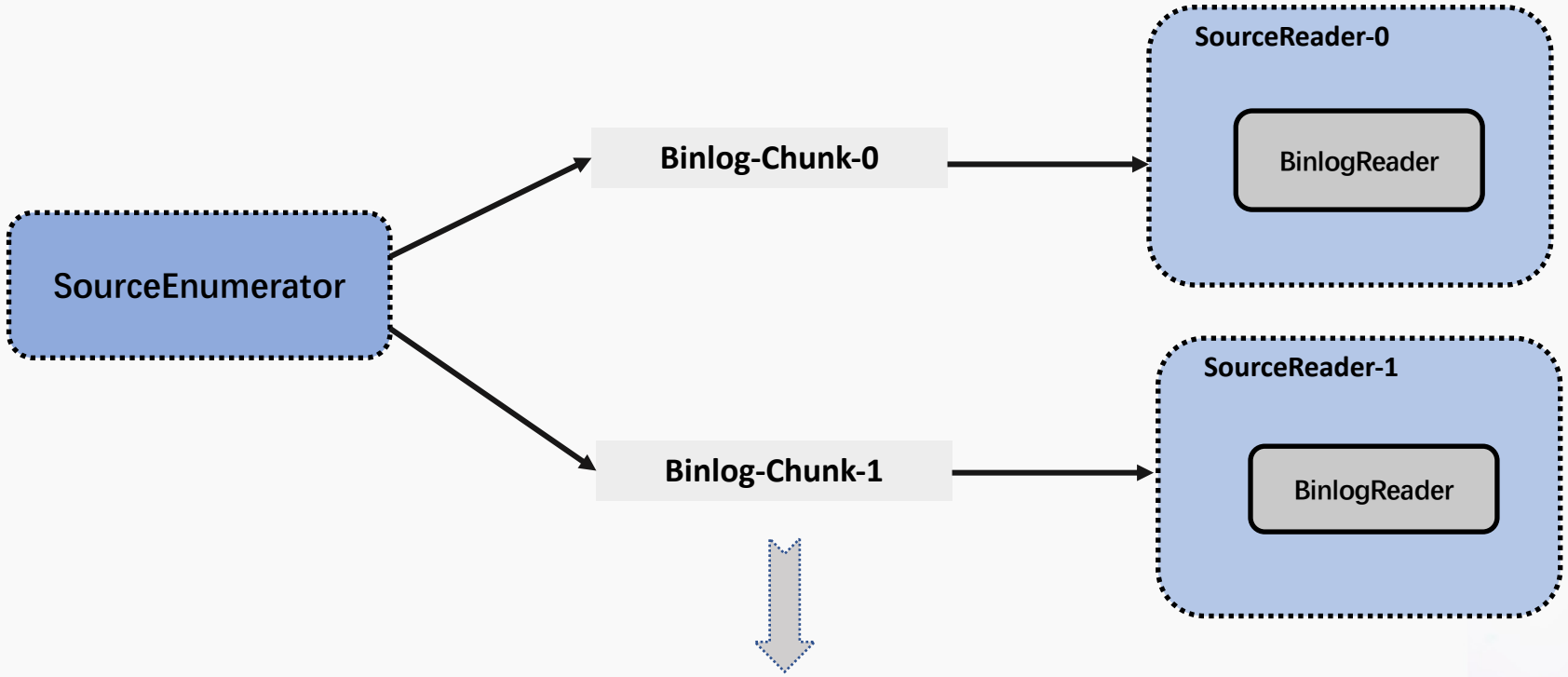
Flink CDC 2.0 设计

Chunk 汇报



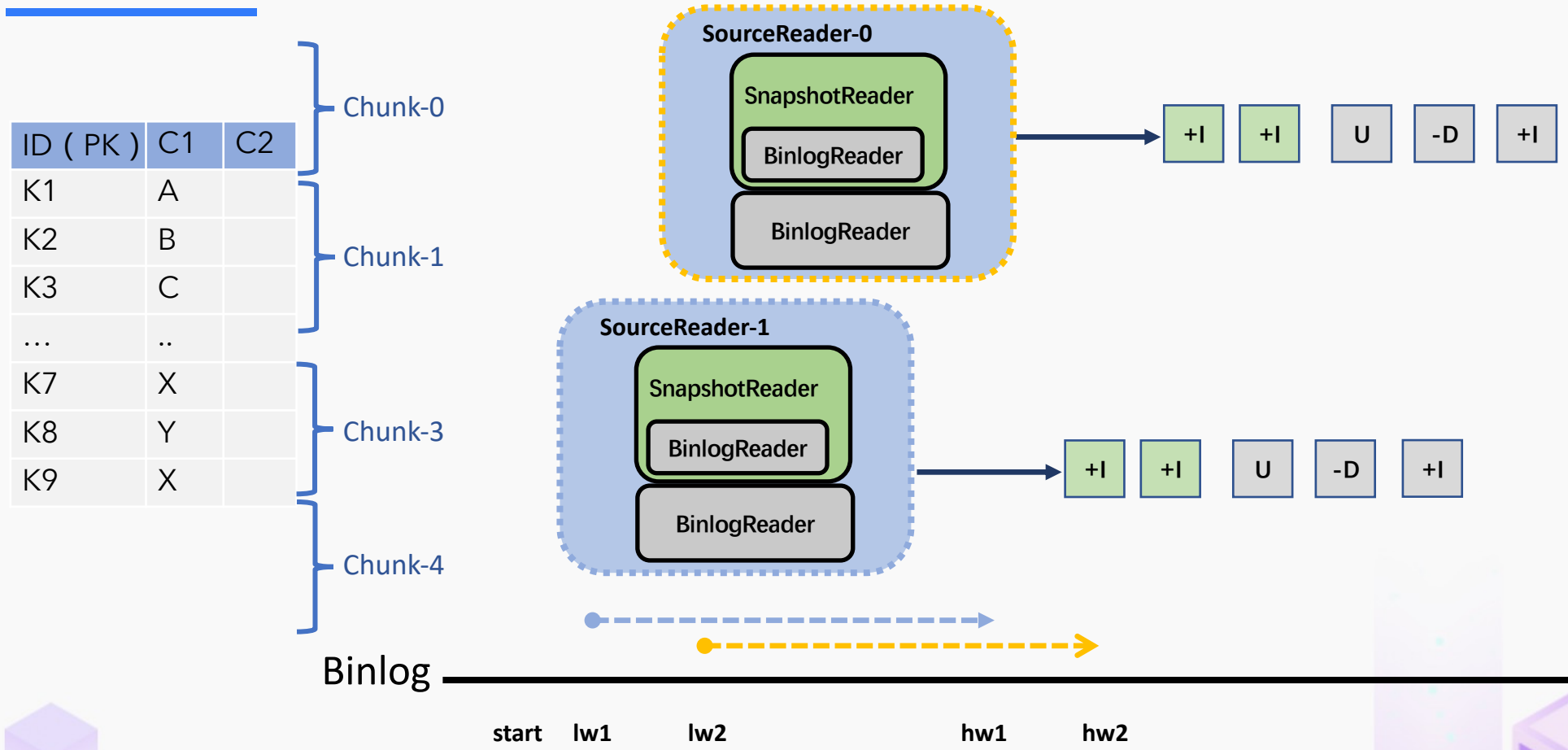
Flink CDC 2.0 设计

Chunk 分配： binlog-chunk



Flink CDC 2.0 设计

整体流程



Flink CDC 2.0

代码实现

完全开源， 贡献社区

MySQL-CDC 2.0 代码：https://github.com/ververica/flink-cdc-connectors/pull/233

#4 Flink-CDC 未来规划

Flink-CDC
未来规划

稳定性

- Lazy Assigning
- 性能提升
- Binlog Merging

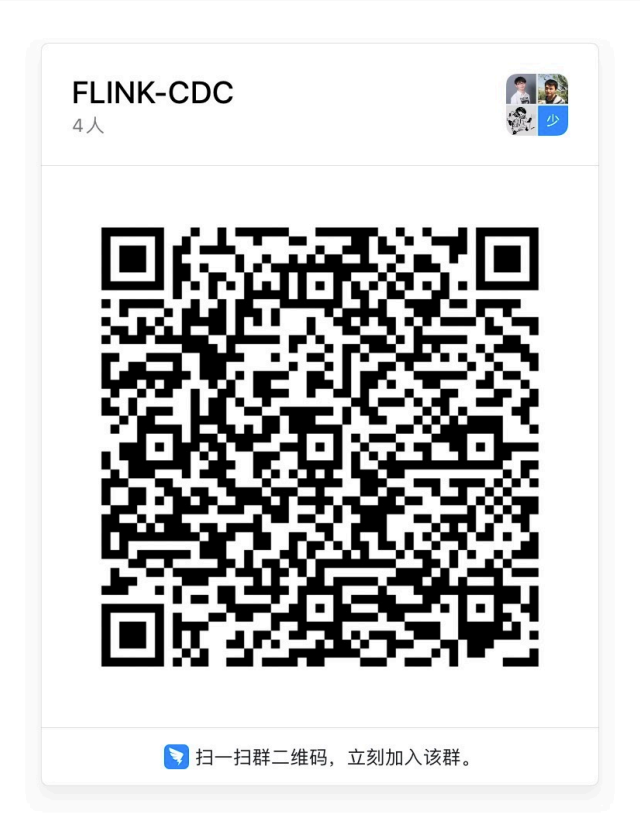
进阶 feature

- Schema Evolution
- Watermark Pushdown
- 支持 META 数据
- 整库同步

生态集成

- 更多 DB, Format
- 入湖 Hudi Iceberg

Flink-CDC
未来规划



项目地址：https://github.com/ververica/flink-cdc-connectors

Apache Flink TiDB | COMMUNITY

THANKS

Apache Flink x TiDB Meetup · 北京站