Background

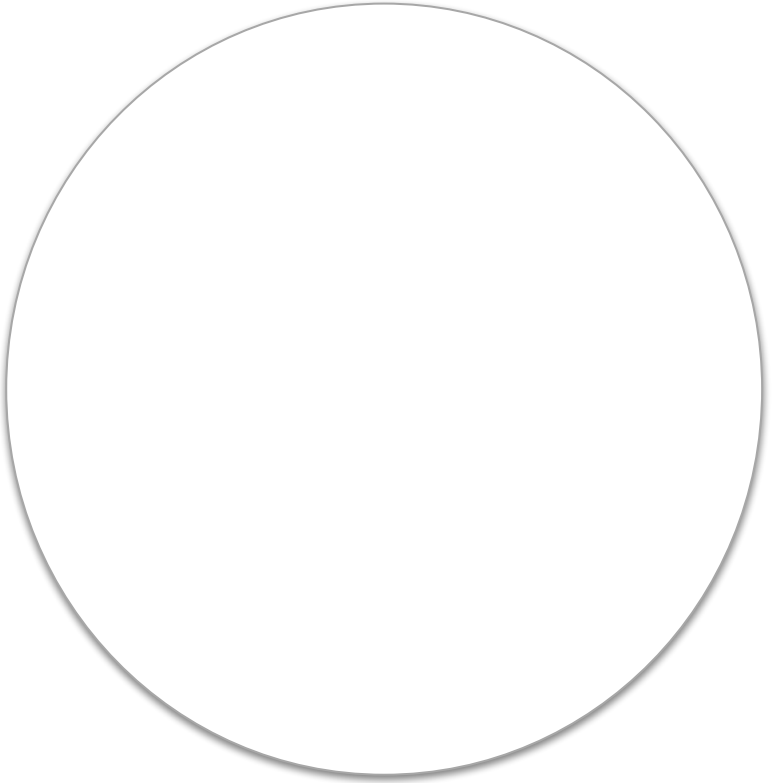
1.In order to reduce marketing costs and achieve precise promotions, DE cooperates with the promotion team to provide marketing-related indicators (standards for coupon distribution)

2.From the calculation history to the present, the number of all orders under each biz status of each user

3.Involves costs and requires strong data consistency

4.Biz status changes frequently and the amount of data is huge

- 7.3 billion historical data
- Average daily increment exceeds 23 million
- 2.16TB storage



- Support data backfill
- Required data consistency

- Biz status changes for a long time, e.g. 30 days

- Only Have Flink, HBase, MySQL
- Avoid unrestrained use of resources

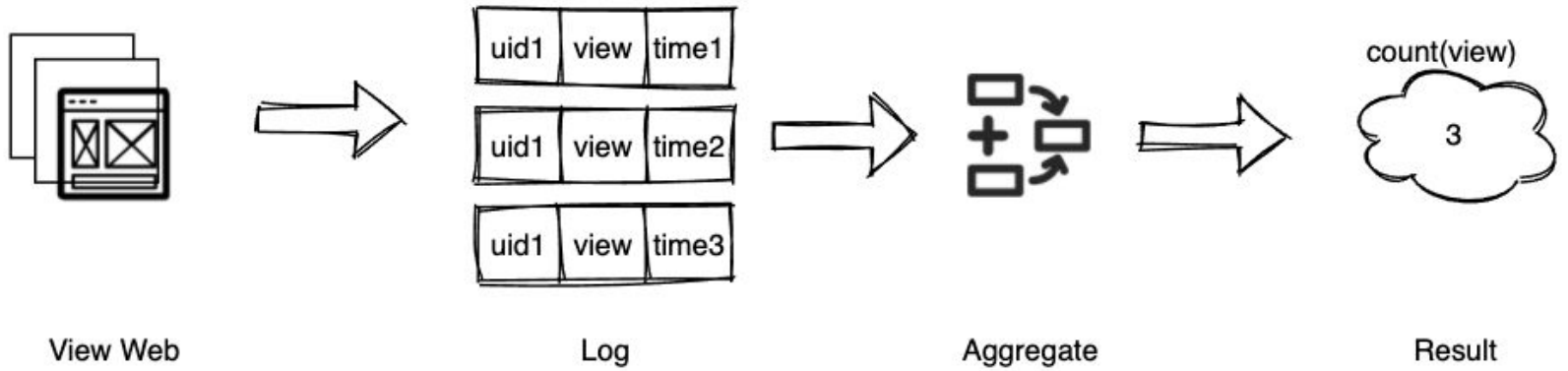
Append And Change Log

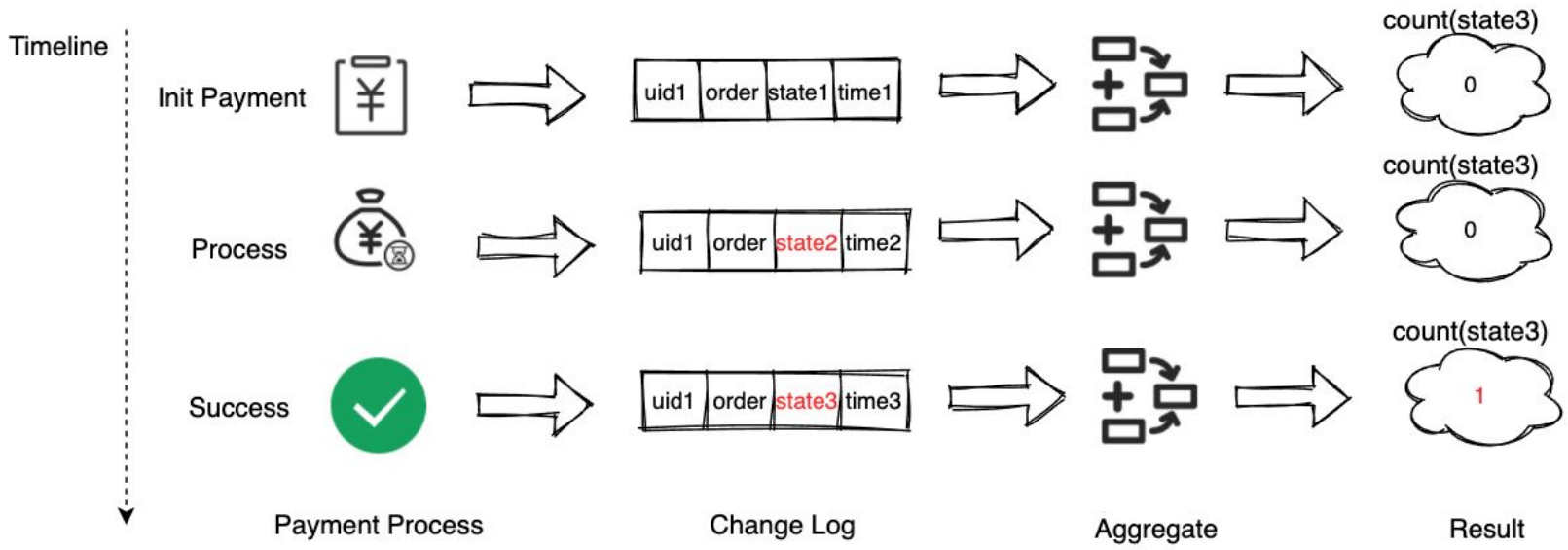
Append-Only

- Only append(insert) data
- A record is an action (fact), which cannot be modified once it occurs

Change Log

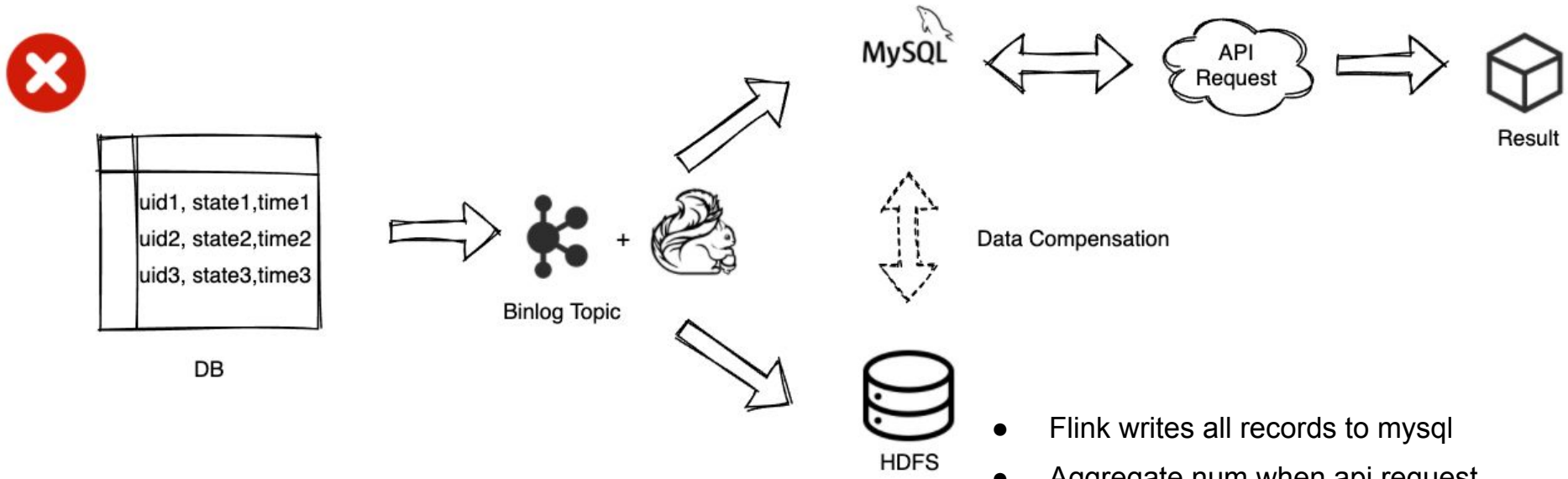
- Records can change
- A row of records only represents the state at that moment





Store Detailed Data

- Resolve biz state changes
- Extremely slow read and write IO
- Doesn't resovle resource overhead

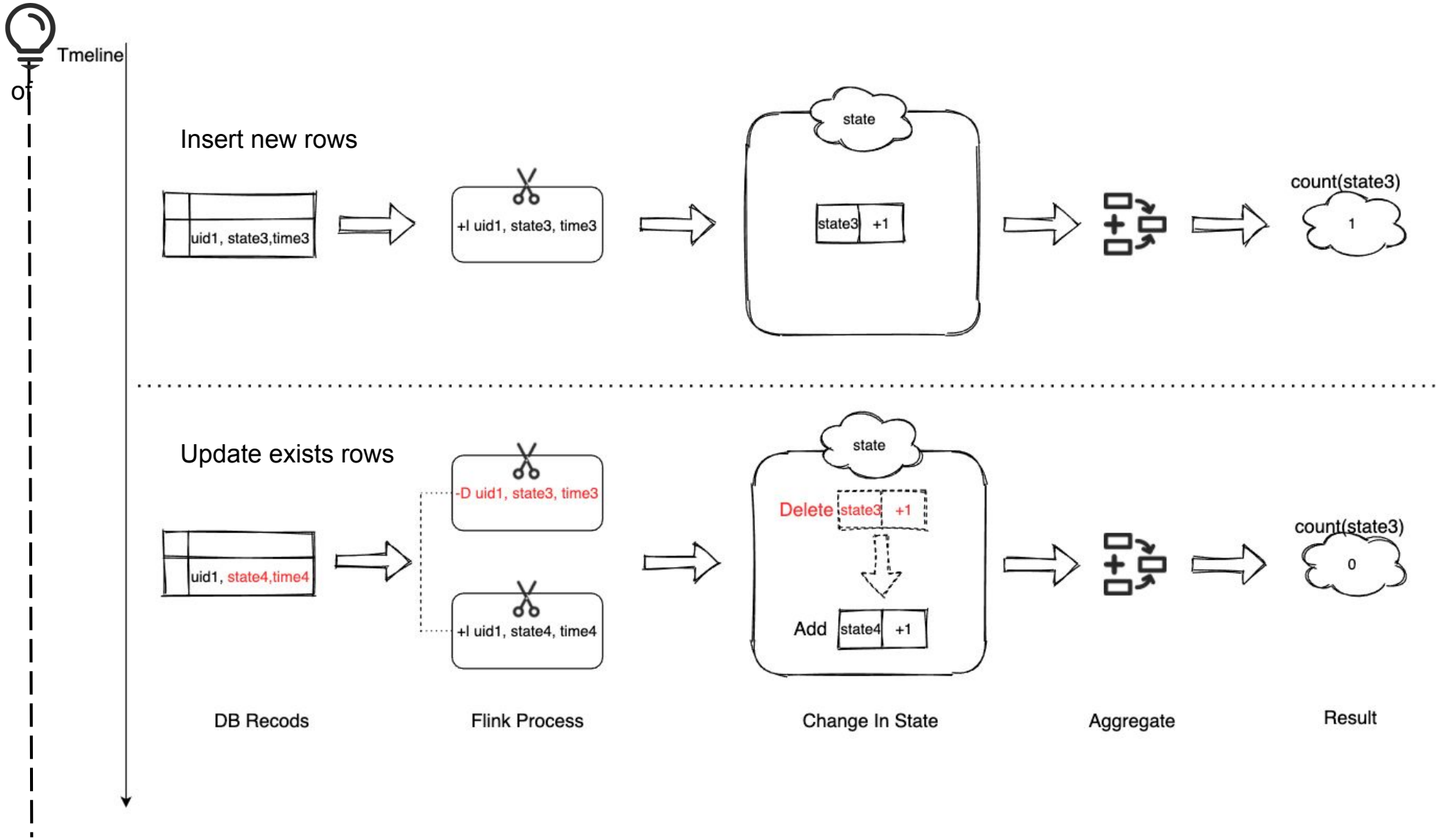


- Flink writes all records to mysql
- Aggregate num when api request

Data Consistency	Time Sensitivity	Backfill Support	State Change	Controllability	ResourceOverhead
√	√	√	√	×	×

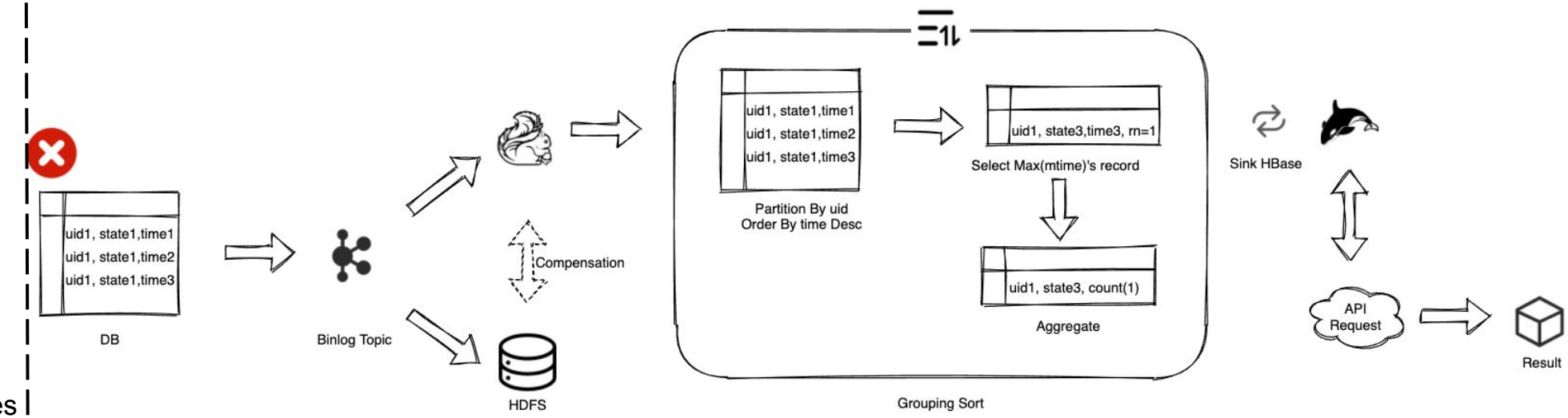
Why Use Retract?

- Convert a record to one of two message types, insert(+I), delete(-D)
- Allow some previously output results to be withdrawn on the output stream



Use Flink Grouping Sort

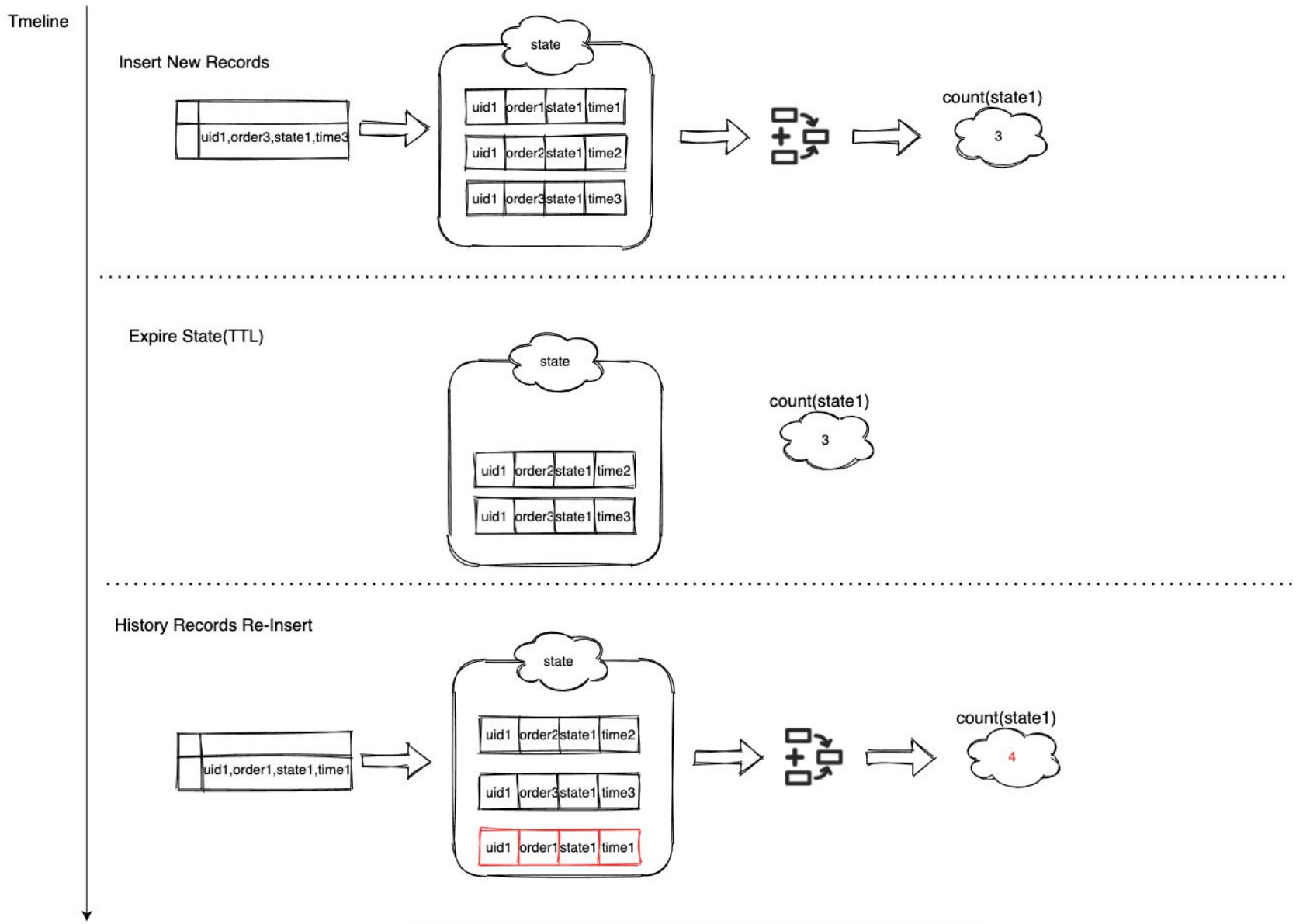
- Resolve catch biz state changes
- Doesn’t resovle resource overhead
- Unable to ensure data consistency
- Although data backfill is supported, Flink program must be consumed all of history records
- There are certain limitations in special cases



Data Consistency	Time Sensitivity	Backfill Support	State Change	Controllability	ResourceOverhead
×	√	×	√	×	×

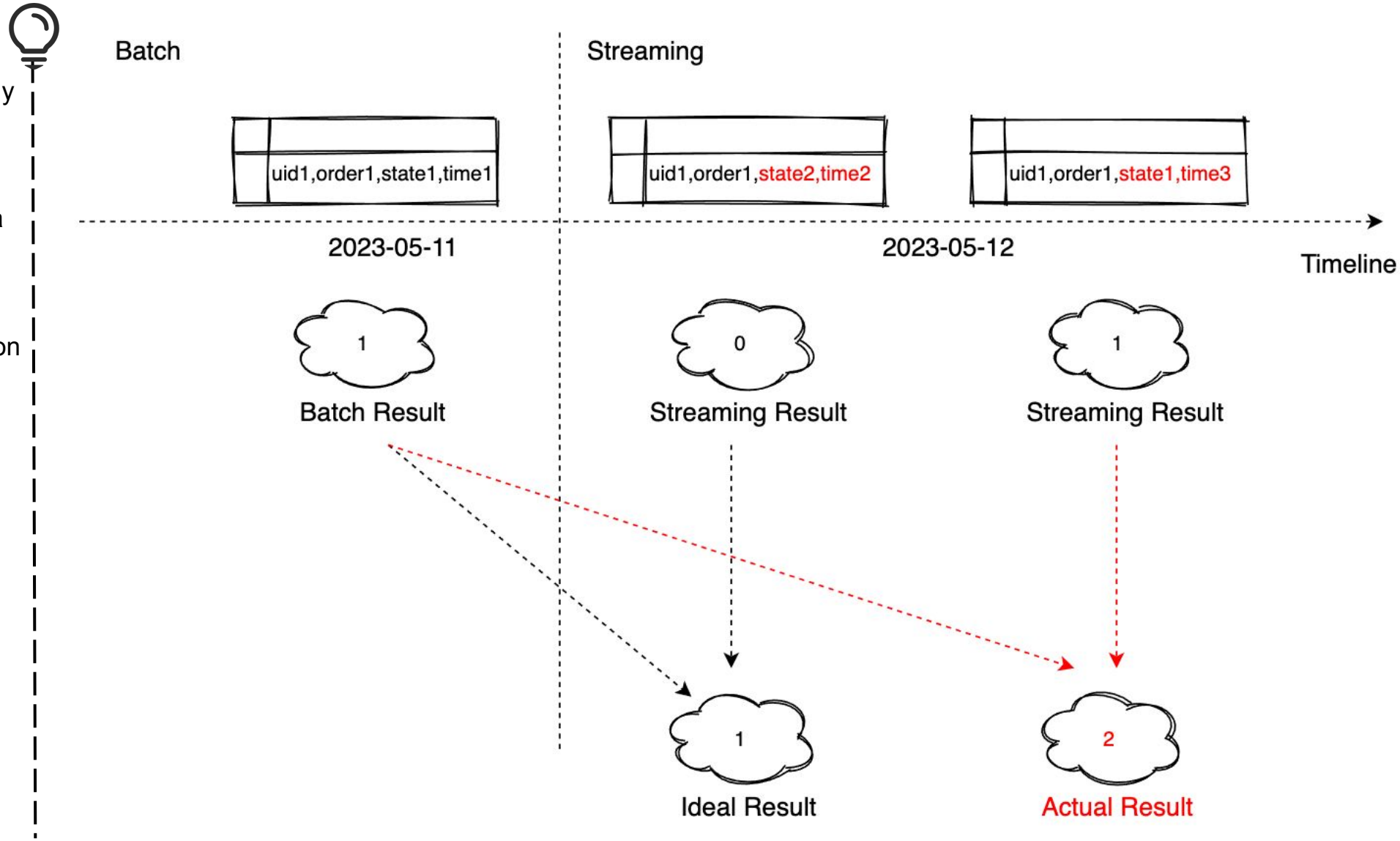
Why not set TTL?

- Cannot resolve data compensation or DI binlog missing and reissuing situation
- Flink state(CP or SP) is not fully available due to cluster instability

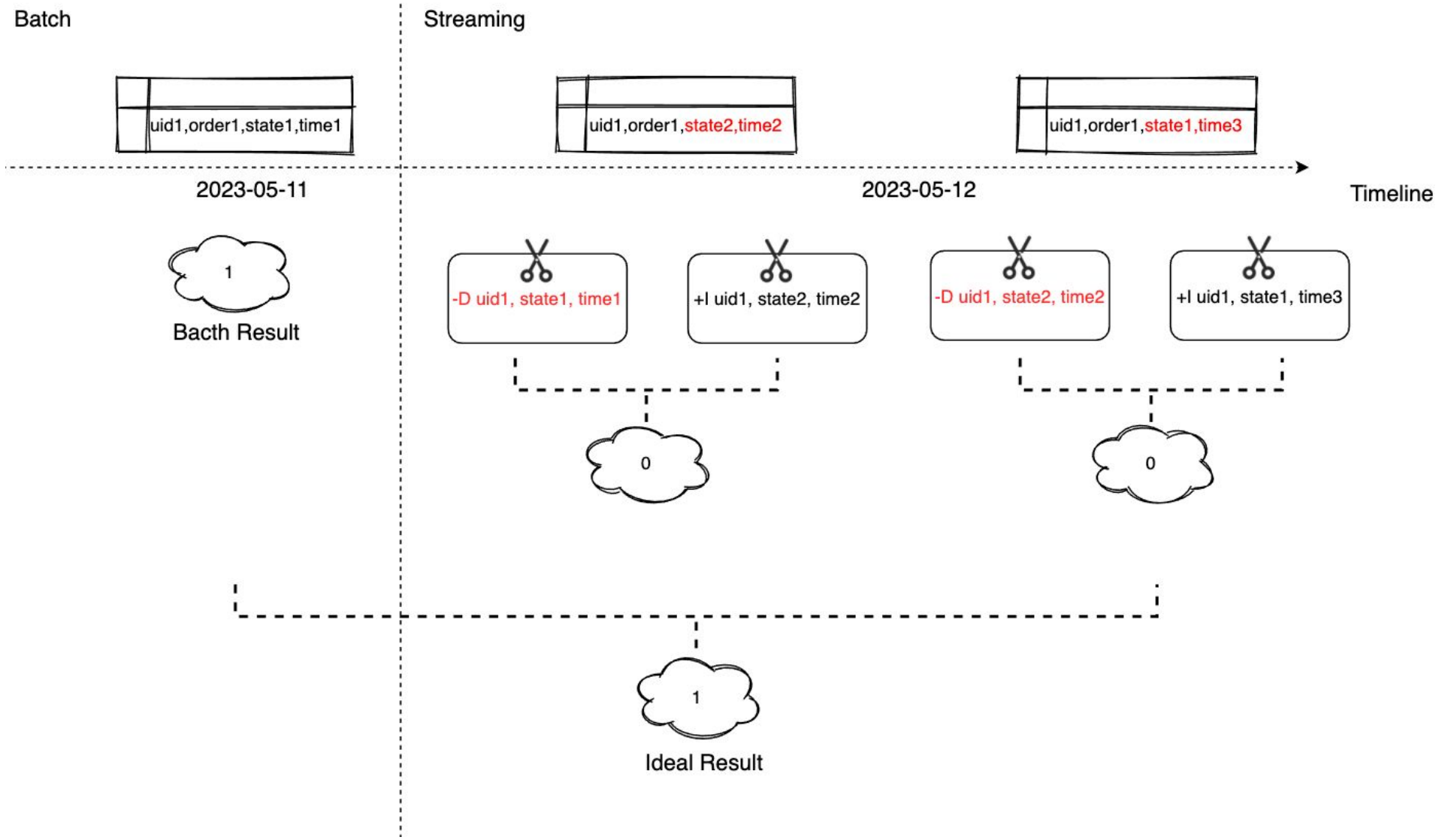


Why not use daily cut?

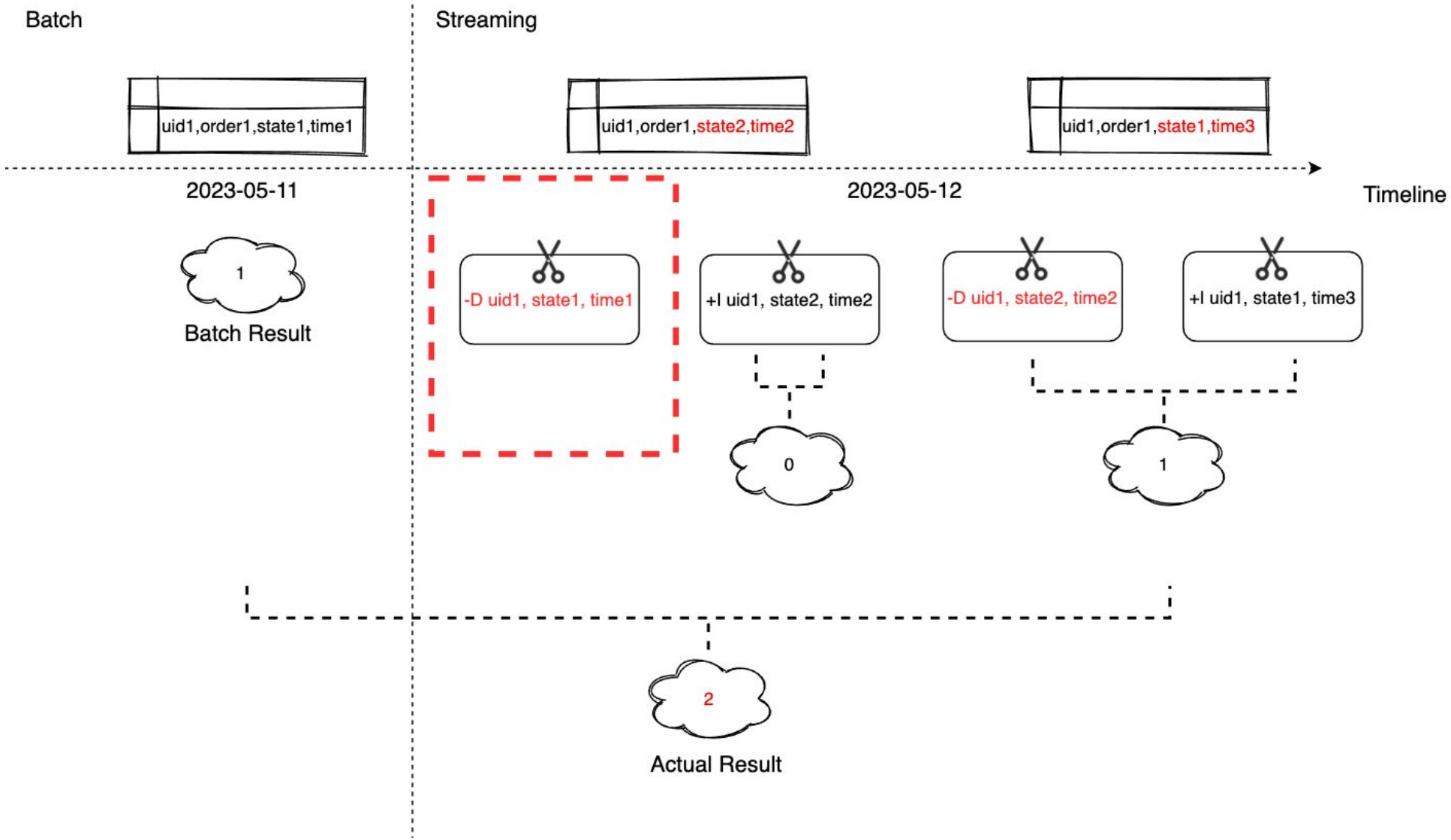
- It's very difficult to strictly cut time boundary between batch and streaming, such as data drift
- If biz state of historical data changes, calculation result will be distorted



Retract Process Limitation



Retract Process Limitation



Binlog Structure Analysis

Flink RowKind example

```
// create a changelog DataStream
DataStream<Row> dataStream =
    env.fromElements(
        Row.ofKind(RowKind.INSERT, "Alice", 12),
        Row.ofKind(RowKind.INSERT, "Bob", 5),
        Row.ofKind(RowKind.UPDATE_BEFORE, "Alice", 12),
        Row.ofKind(RowKind.UPDATE_AFTER, "Alice", 100));

// interpret the DataStream as a Table
Table table = tableEnv.fromChangelogStream(dataStream);

// register the table under a name and perform an aggregation
tableEnv.createTemporaryView("InputTable", table);
tableEnv
    .executeSql("SELECT f0 AS name, SUM(f1) AS score FROM InputTable GROUP BY f0")
    .print();

// prints:
// +----+--------------------------------+-------------+
// | op |                           name |       score |
// +----+--------------------------------+-------------+
// | +I |                            Bob |           5 |
// | +I |                          Alice |          12 |
// | -D |                          Alice |          12 |
// | +I |                          Alice |         100 |
// +----+--------------------------------+-------------+
```

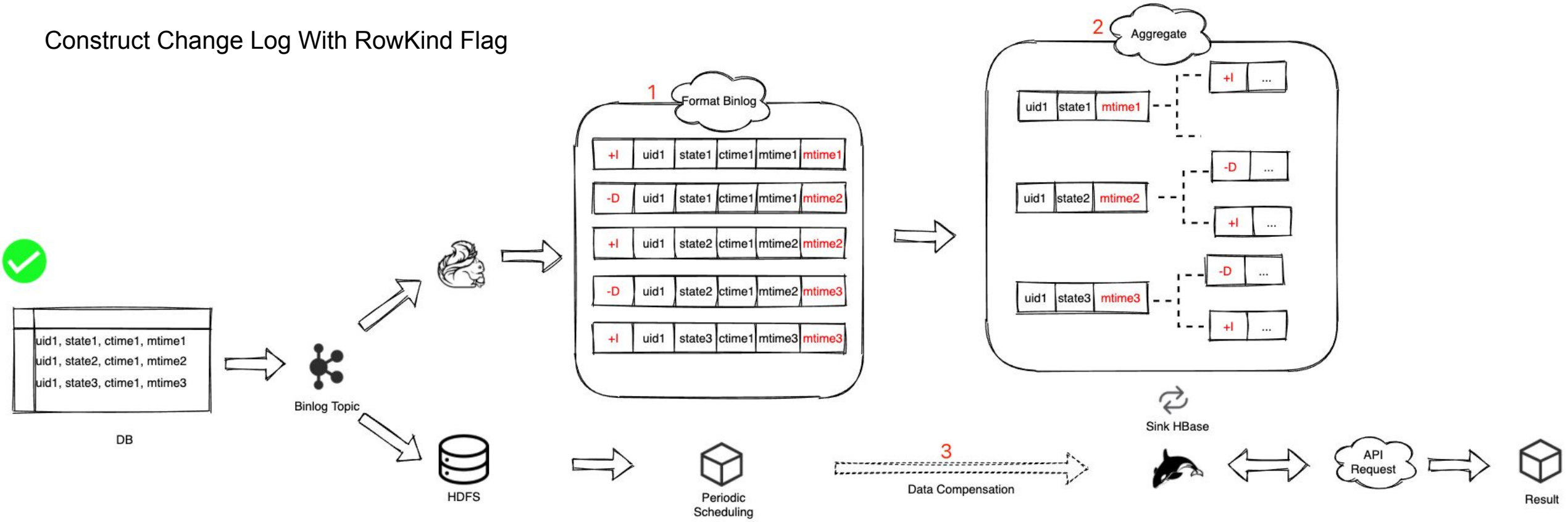
Binlog example,have newRow and oldRow

```
"data":{
   "id":1,
   "m":5.444,
   "c":"2016-10-21 05:33:54.631000",
   "comment":"I am a creature of light."
},
"old":{
   "m":4.2341,
   "c":"2016-10-21 05:33:37.523000"
}
```

Connector upgrade

```
@Override
public void deserialize(byte[] message, Collector<RowData> out) throws IOException {
    if (message == null || message.length == 0) {
        return;
    }
    try {
        final JsonNode root = jsonDeserializer.deserializeToJsonNode(message);
        if (!isMatchDbAndTable(root)) {
            return;
        }
        final GenericRowData row = (GenericRowData) jsonDeserializer.convertToRowData(root);
        String type = row.getString(2).toString(); // "type" field
        if (OP_INSERT.equals(type)) {
            // "data" field is a row, contains inserted rows
            GenericRowData insert = (GenericRowData) row.getRow(0, fieldCount);
            insert.setRowKind(RowKind.INSERT);
            if (this.extendFieldIndex != null) {
                insert.setField(this.extendFieldIndex.f0, insert.getField(this.extendFieldIndex.f1));
            }
            emitRow(row, insert, out);
        } else if (OP_UPDATE.equals(type)) {
            // "data" field is a row, contains new rows
            // "old" field is a row, contains old values
            // the underlying JSON deserialization schema always produce GenericRowData.
            GenericRowData after = (GenericRowData) row.getRow(0, fieldCount); // "data" field
            GenericRowData before = (GenericRowData) row.getRow(1, fieldCount); // "old" field
            final JsonNode oldField = root.get(FIELD_OLD);
            for (int f = 0; f < fieldCount; f++) {
                if (before.isNullAt(f) && oldField.findValue(fieldNames.get(f)) == null) {
                    // not null fields in "old" (before) means the fields are changed
                    // null/empty fields in "old" (before) means the fields are not changed
                    // so we just copy the not changed fields into before
                    before.setField(f, after.getField(f));
                }
            }
            before.setRowKind(RowKind.UPDATE_BEFORE);
            after.setRowKind(RowKind.UPDATE_AFTER);
```

Construct Change Log With RowKind Flag

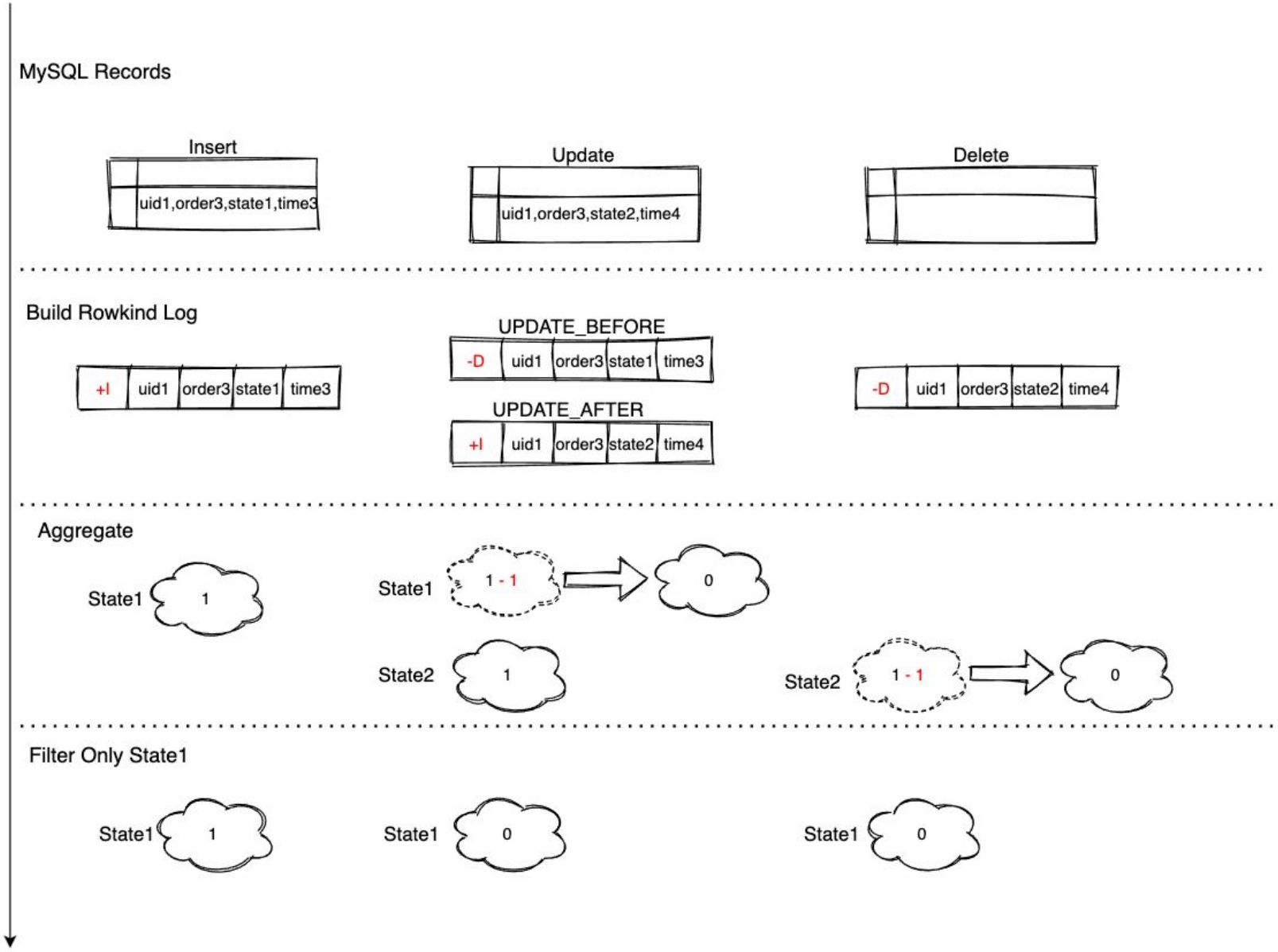


	1	2	3
Description	Build a label for each binlog according to the rowkind,Adding or withdrawing all over the aggregation operator	Stream will aggregate itself according to the rowkind identifier and convert it into a sum (negative value, positive value) operation	Batch processing is divided by date, and data compensation is scheduled regularly

Retract Rebuild

- Flink can’t retract records that are not in Flink state
- Explicitly assign RowKind to each binlog, and split the UPDATE type into two (UPDATE_BEFORE and UPDATE_AFTER)

Process



Daily Cut Flag

- When splitting binlog, update_time(UPDATE_AFTER) of the new value is additionally assigned to the old value (UPDATE_BREFORE)
- Ensure that binlogs that appear in pairs can fall on the latest day
- No longer strongly dependent on Flink state, only keep Flink state of the day