Apache Flink 维表关联实战

Apache Flink Community China

CONTENT

目录 >>

01 /
Join

02 /
Flink SQL Join

03 /
Flink DataStream Join

04 /
Flink 案例实战演练

01

Join

Join 的概念

Join 定义及特点

定义：A JOIN is a means for combining columns from one (self-join) or more tables by using values common to each

JOIN 是一种通过使用每个表的相同值来组合一个（自联接）或多个表中的列的方法

特点：

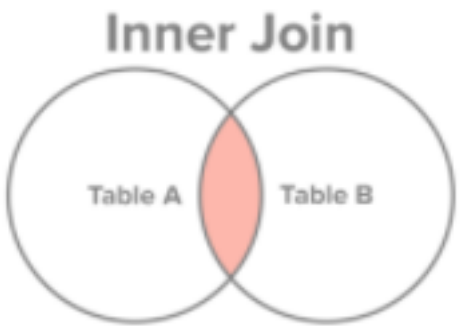
1、业务使用频繁

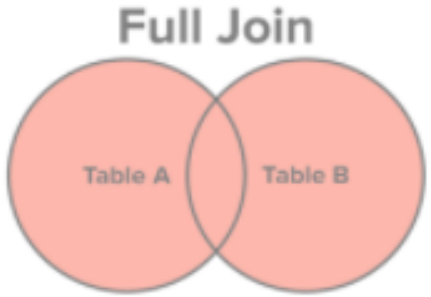
2、优化规则复杂

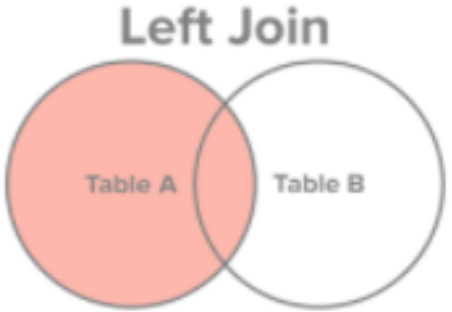
https://en.wikipedia.org/wiki/Join_(SQL)

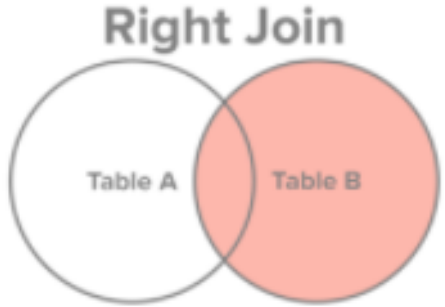
Join 类型

- Cross Join（交叉连接，计算笛卡尔积）
- Inner Join（内连接）
- Left Outer Join（返回左表所有行，右表不存在补 NULL）
- Right Outer Join（返回右表所有行，左表不存在补 NULL）
- Full Outer Join（返回左表和右表的并集，不存在一边补 NULL）









Join 常见实现

◆ Nested loop Join：最为简单直接，将两个数据集加载到内存，并用内嵌遍历的方式来逐个比较两个数据集内的元素是否符合 Join 条件。虽然时间和空间效率都是最低的，但比较灵活，适用范围广。

◆ Sort-Merge Join：分为 Sort 和 Merge 阶段，首先将两个数据集分别排序，然后对两个有序数据集分别进行遍历和匹配，类似于归并排序的合并。缺点：需要对两个数据集进行排序，成本很高。

◆ Hash Join：先将一个数据集转换为 Hash Table，然后遍历另外一个数据集元素并与 Hash Table 内的元素进行匹配。Hash Join 效率较高但对空间要求较大，适合 Join 可以放入内存的小表。

02

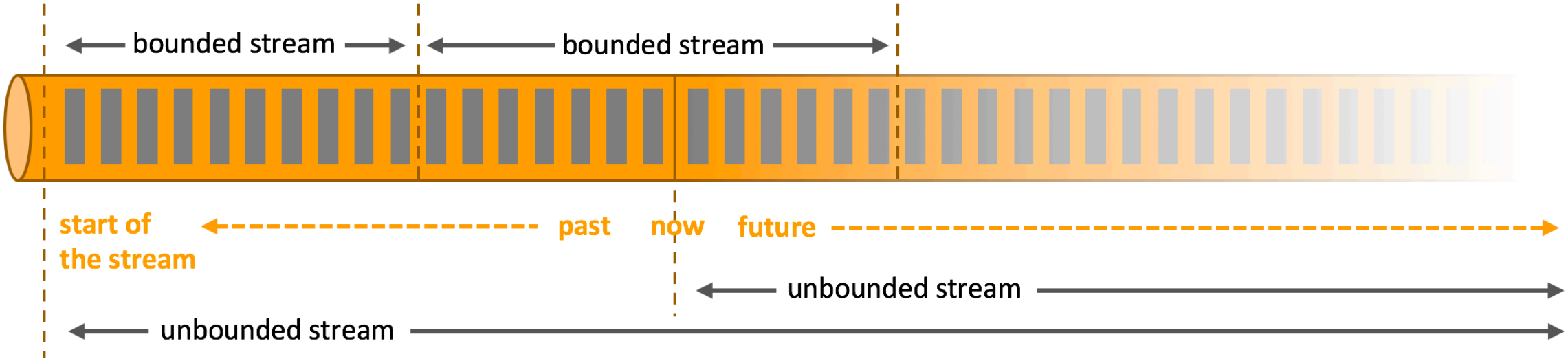
Flink SQL Join

Streaming SQL Join

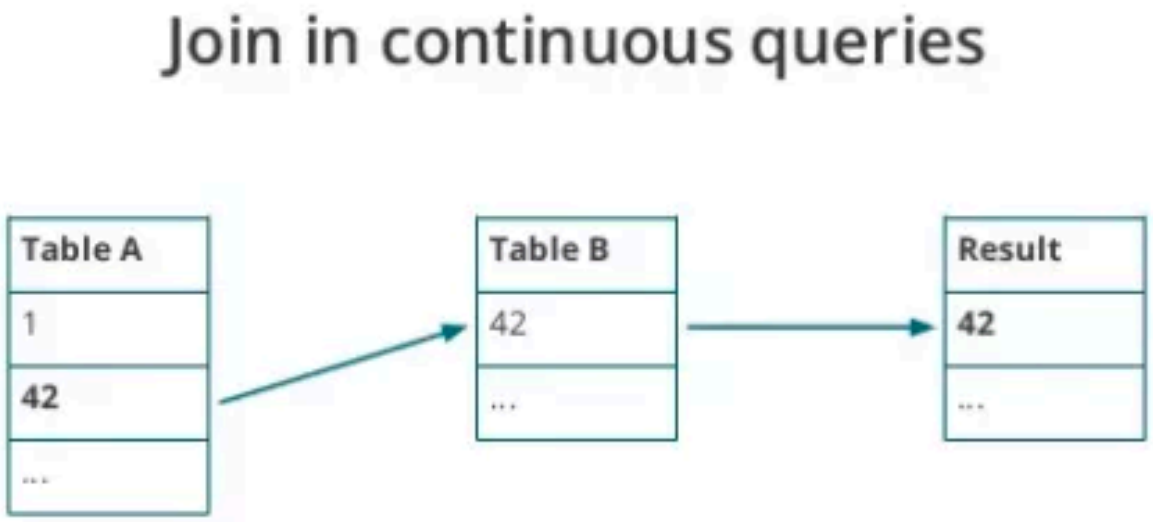
实时 Streaming SQL Join

面向无界数据集的 SQL，无法缓存历史所有数据，因此像 Sort-Merge Join 需要对数据进行排序是无法做到的，

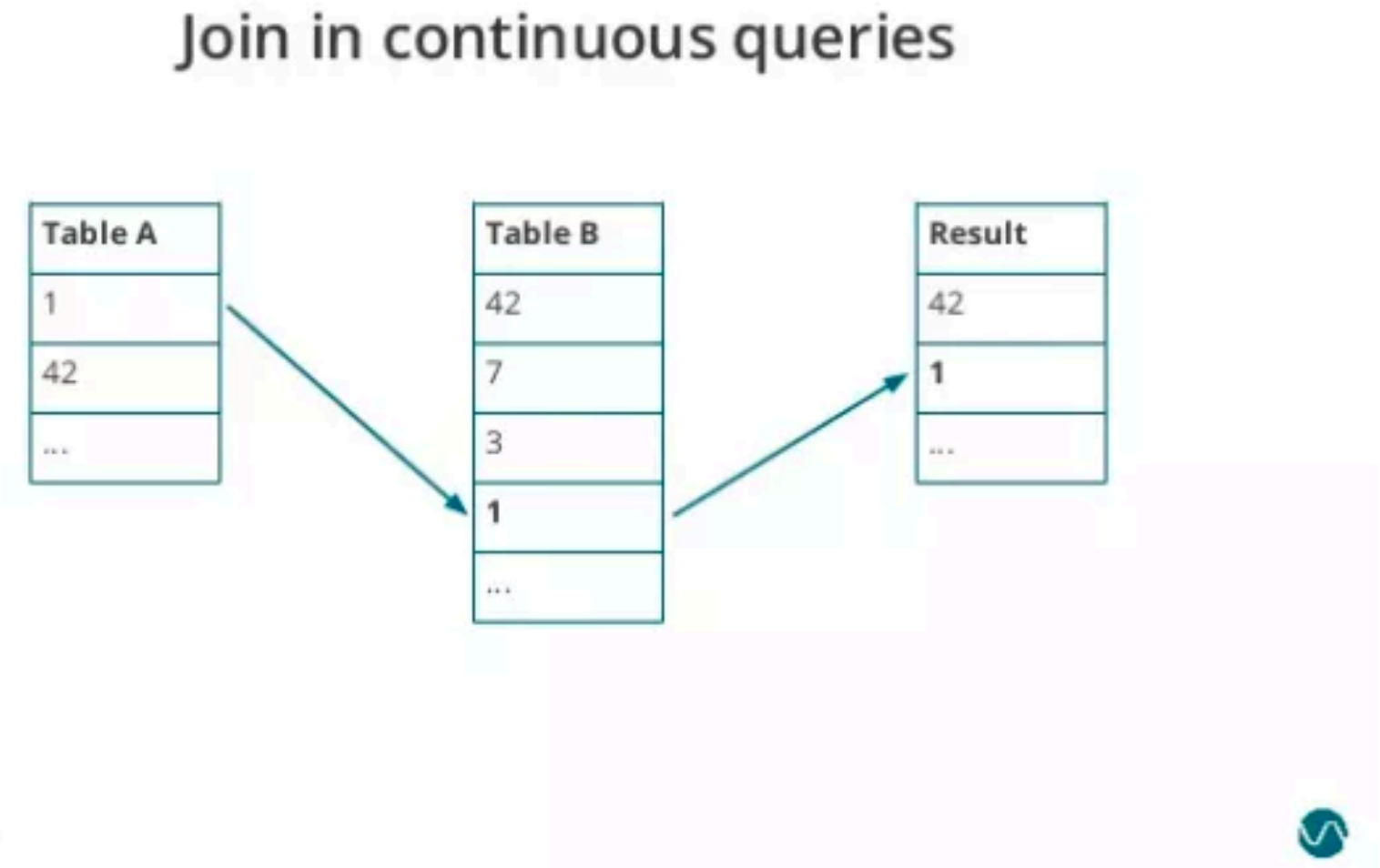
Nested-loop Join 和 Hash Join 经过一定的改良则可以满足实时 SQL 的要求。



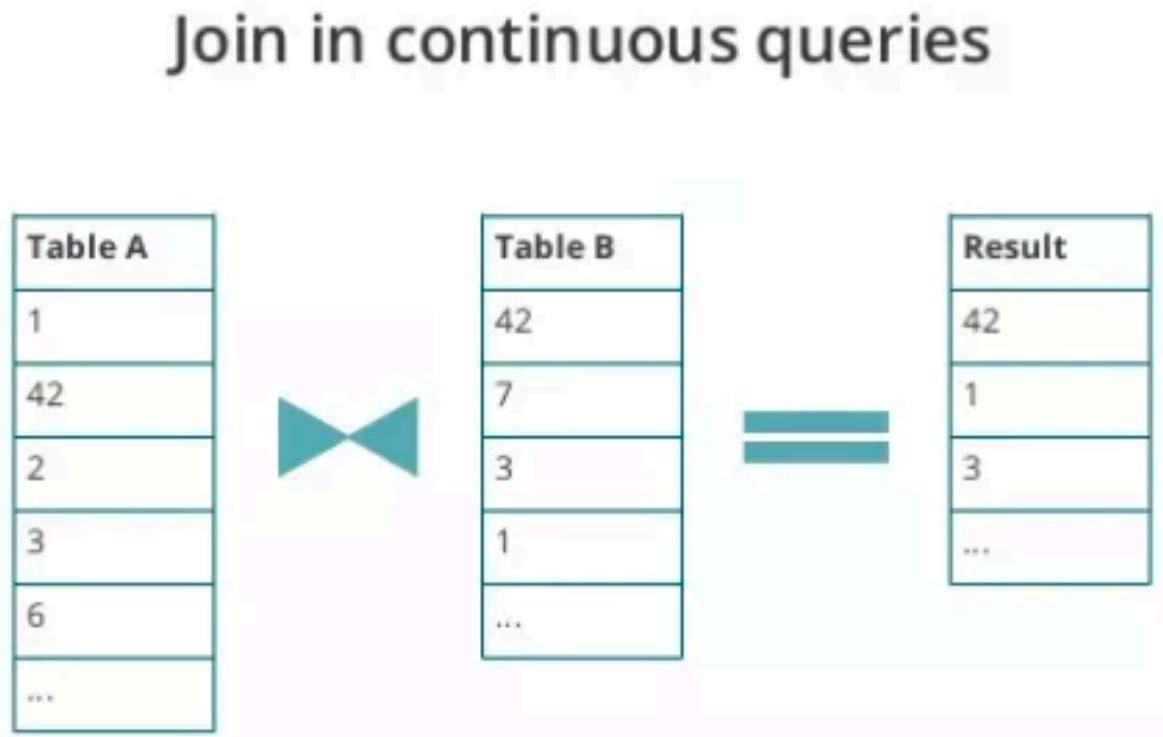
Nested Join 在实时 Join 的实现



Nested Join 在实时 Join 的实现



Nested Join 在实时 Join 的实现



问题和改进措施

问题：需要保存两个数据源表的内容，随着时间的增长，需要保存的历史数据无止境地增长，导致很不合理的内存磁盘资源占用，而且单个元素的匹配效率也会越来越低。

改进措施：设置一个缓存剔除策略，将不必要的历史数据及时清理！

Flink Regular Join

最为基础的、没有缓存剔除策略的 Join，两个表的输入和更新都会对全局可见，会影响之后所有的 Join 结果。

举例，在一个如下的 Join 查询里，Orders 表的新纪录会和 Product 表所有历史纪录以及未来的纪录进行匹配。

```
SELECT * FROM Orders
INNER JOIN Product
ON Orders.productId = Product.id
```

因为历史数据不会被清理，所以 Regular Join 允许对输入表进行任意种类的更新操作（insert、update、delete）。

然而因为资源问题 Regular Join 通常是不可持续的，一般只用做有界数据流的 Join。

Flink Time–Windowed Join

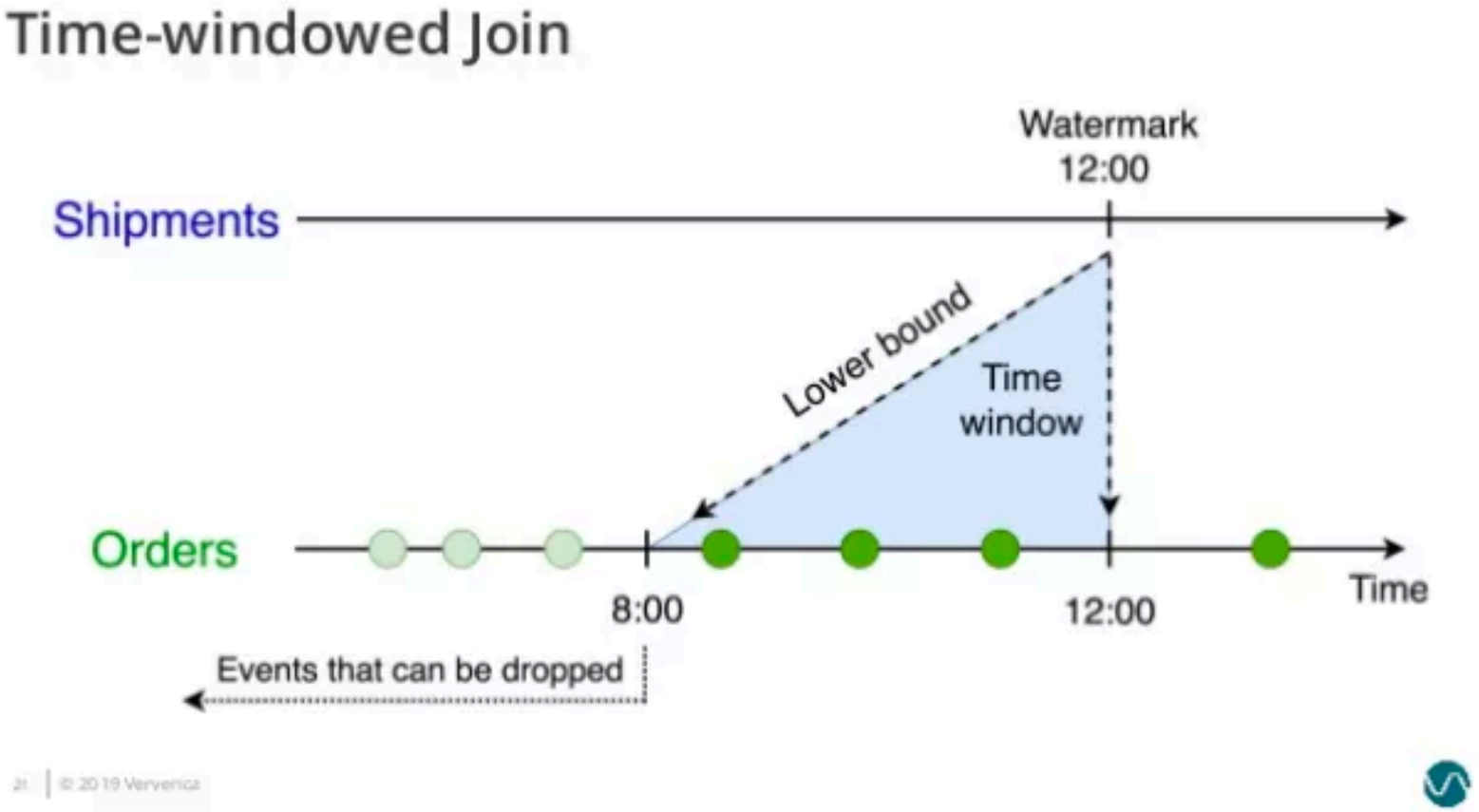
利用窗口给两个输入表设定一个 Join 的时间界限，超出时间范围的数据则对 JOIN 不可见并可以被清理掉。

注意时间语义：如果是 Processing Time，Flink 根据系统时间自动划分 Join 的时间窗口并定时清理数据；

如果是 Event Time，Flink 分配 Event Time 窗口并依据 Watermark 来清理数据。

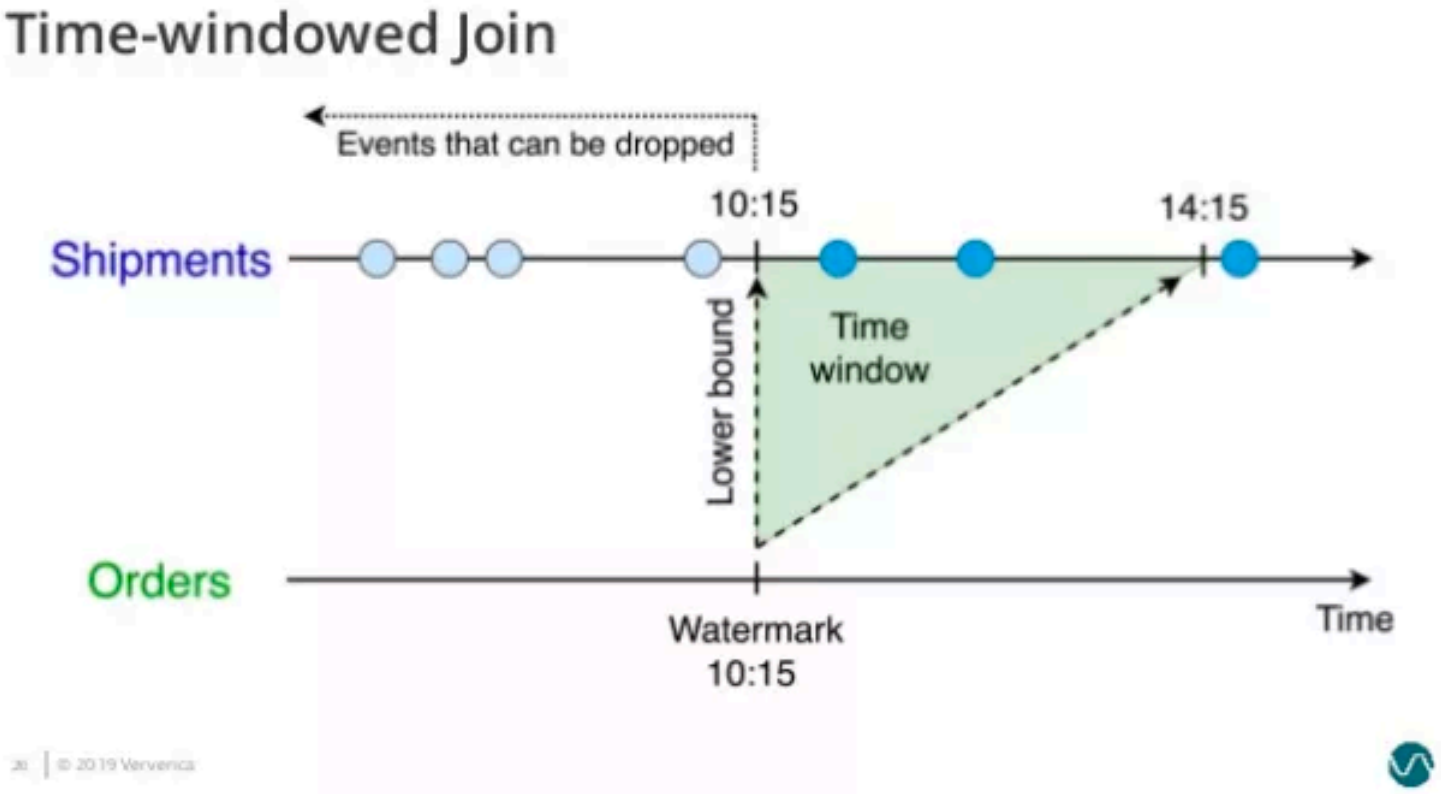
```
SELECT *
FROM
  Orders o,
  Shipments s
WHERE
  o.id = s.orderId AND
  s.shiptime BETWEEN o.ordertime AND o.ordertime + INTERVAL '4' HOUR
```

Flink Time–Windowed Join



为 Orders 表设置了 o.ordertime > s.shiptime- INTERVAL ‘4’ HOUR 的时间下界

Flink Time–Windowed Join



为 Shipmenets 表设置了 s.shiptime >= o.ordertime 的时间下界

Flink Temporal Table Join

Temporal Table Join 将输入分为 Build Table 和 Probe Table。前者是纬表的 changelog，后者是业务数据流。

在 Temporal Table Join 中，Build Table 是一个基于 append-only 数据流的带时间版本的视图，所以又称 Temporal Table。

Temporal Table 要求定义一个主键和用于版本化的字段（通常就是 Event Time 字段），以反映数据在不同时间的内容。

03

Flink DataStream Join

DataStream 方式

关联维表方式

- 实时数据库查找关联（ Per-Record Reference Data Lookup ）
- 预加载维表关联（Pre-Loading of Reference Data）
- 维表变更日志关联（Reference Data Change Stream）
- 灵活组合衍生其他关联方式（More）

衡量指标

- 实现简单性: 设计是否足够简单，易于迭代和维护。
- 吞吐量: 性能是否足够好。
- 维表数据的实时性: 维度表的更新是否可以立刻对作业可见。
- 数据库的负载: 是否对外部数据库造成较大的负载。
- 内存资源占用: 是否需要大量内存来缓存维表数据。
- 可拓展性: 在更大规模的数据下会不会出现瓶颈。
- 结果确定性: 在数据延迟或者数据重放情况下，是否可以得到一致的结果。

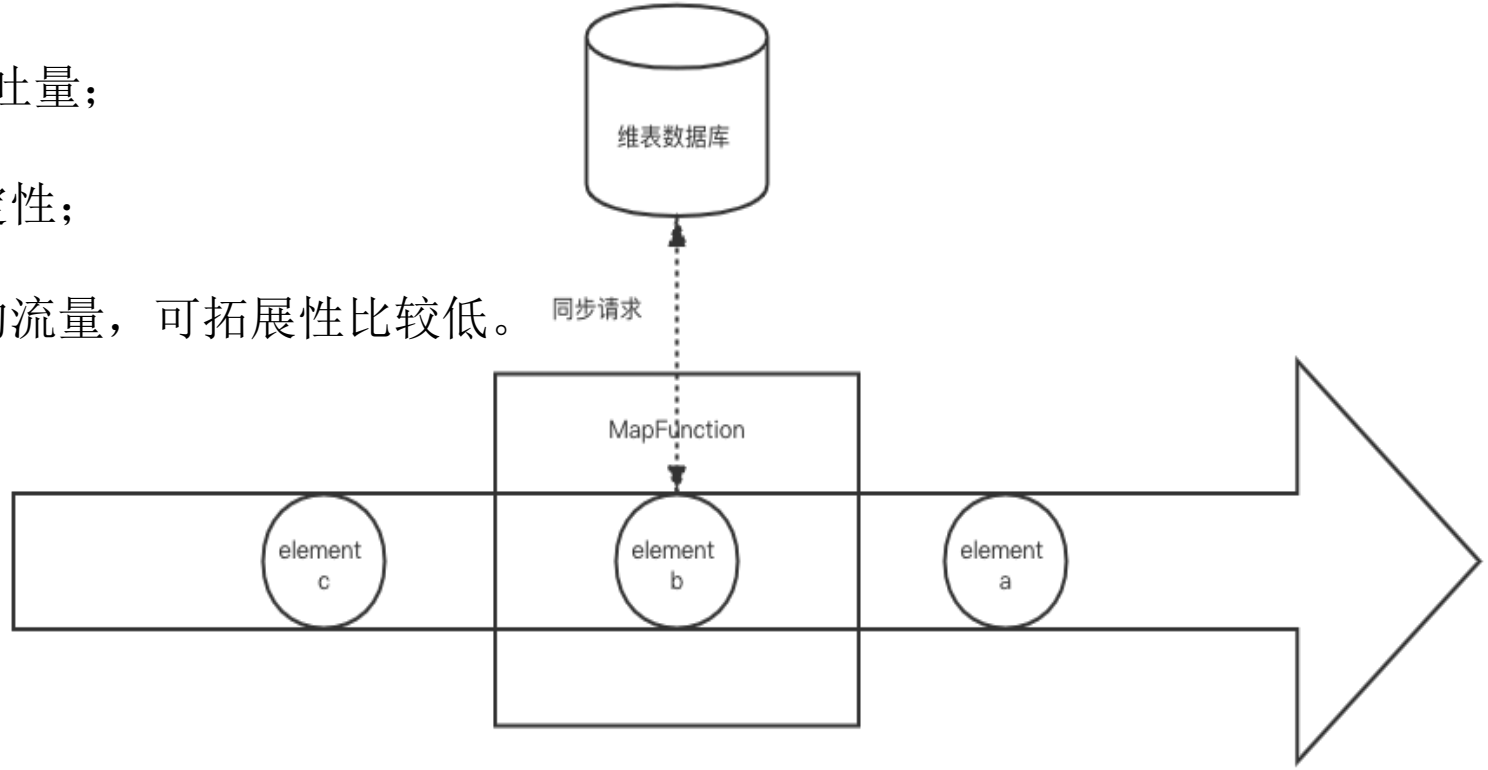
实时数据库同步查找关联

优点：实现简单、不需要额外内存且维表的更新延迟很低。

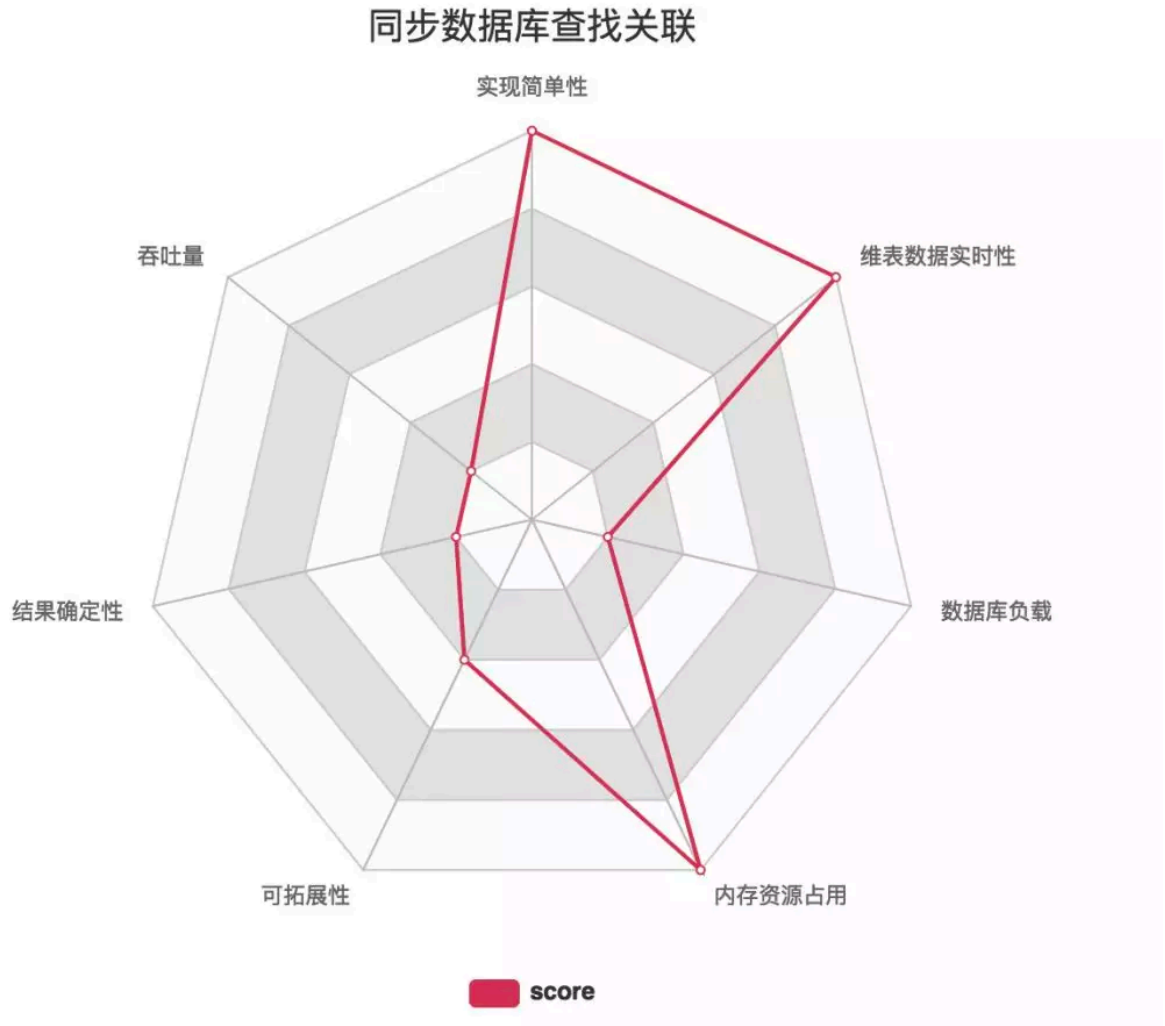
缺点: 数据库压力很大；同步访问，影响吞吐量；

基于 Processing Time 的，结果并不具有确定性；

瓶颈在数据库端，实时流量远大于数据库的流量，可拓展性比较低。



实时数据库同步查找关联

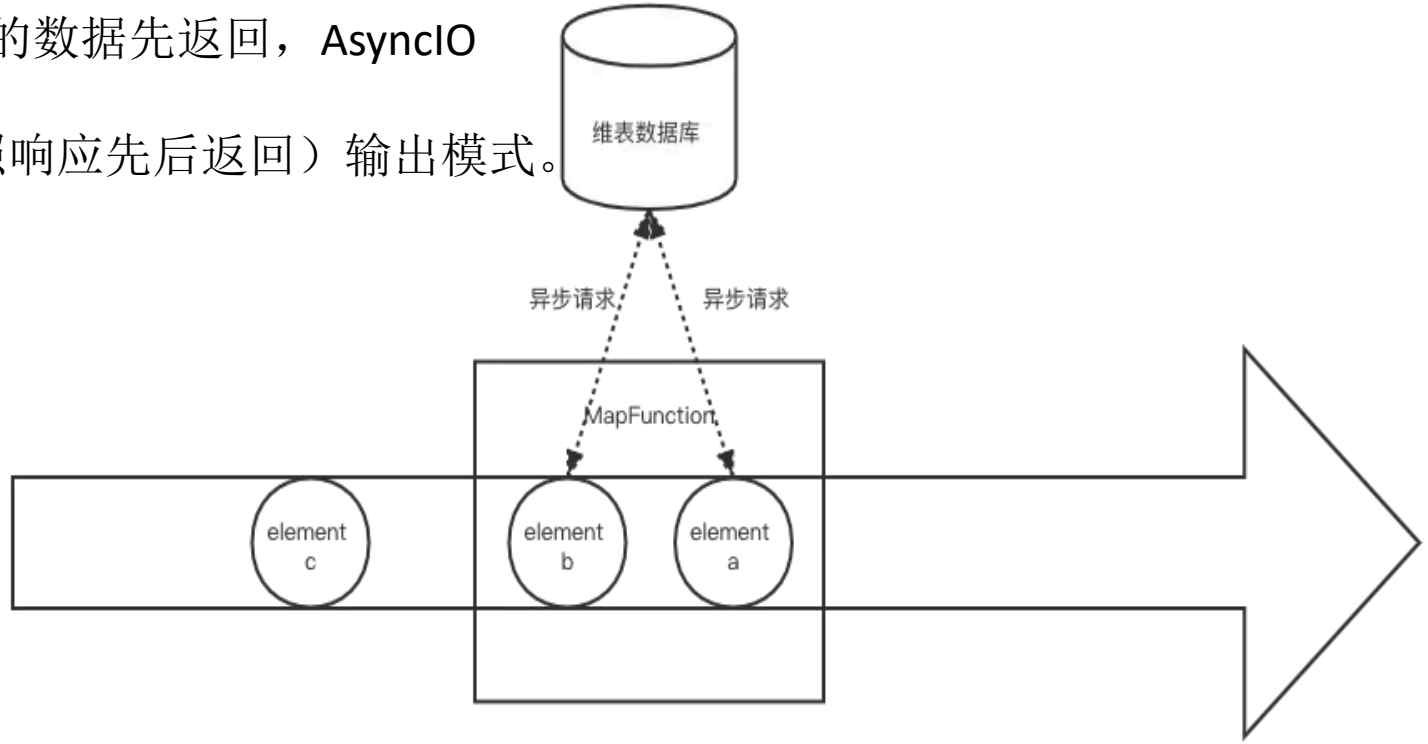


实时数据库异步查找关联

优点：利用数据库提供的异步客户端， AsyncIO 可以并发地处理多个请求。

缺点: 数据库请求响应时长不一致，可能后请求的数据先返回，AsyncIO

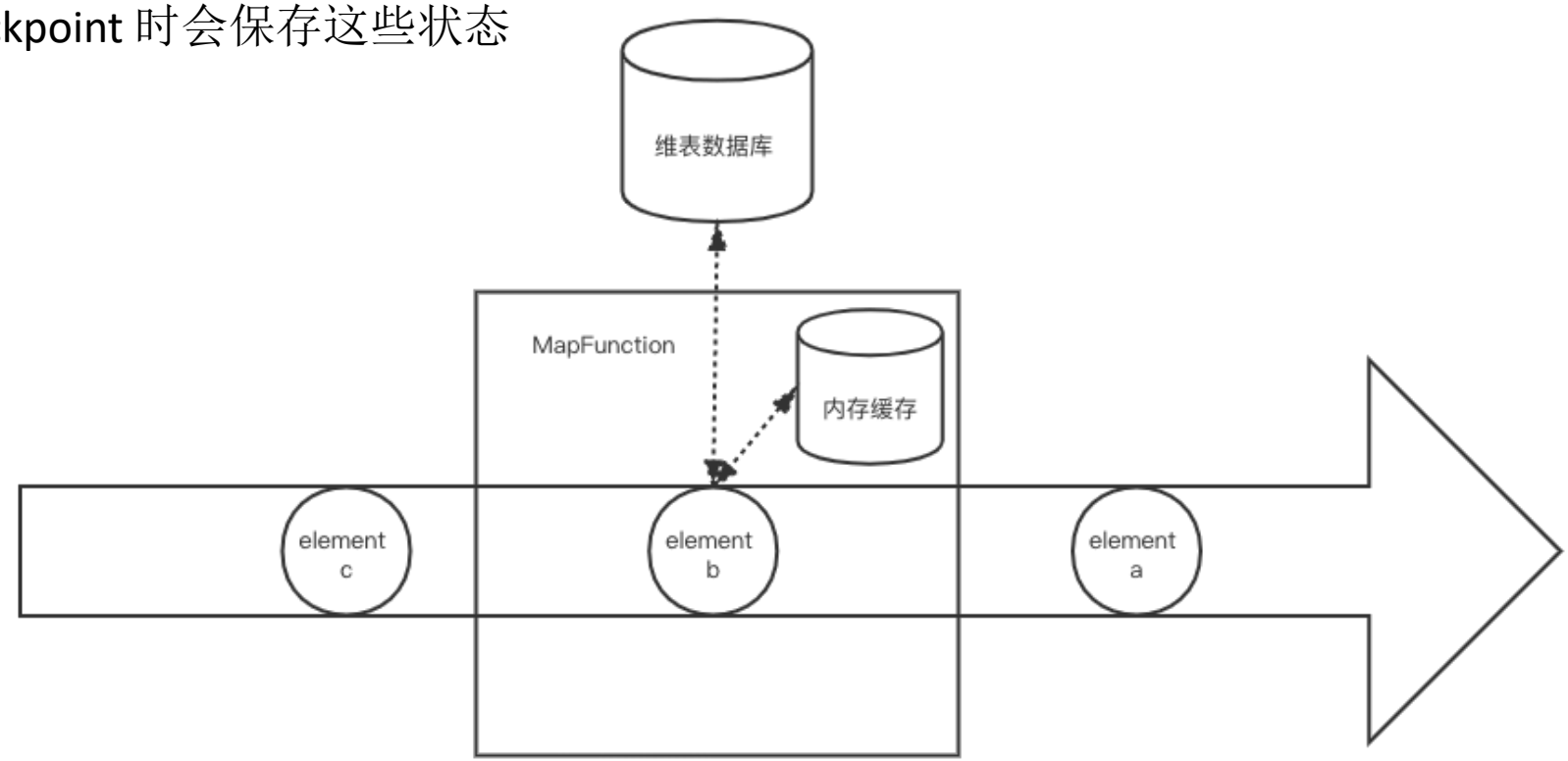
提供有序（按照流数据顺序返回）和无序（按照响应先后返回）输出模式。



实时数据库异步查找关联

优点：比同步查找，性能吞吐高，但稍复杂，AsyncIO API 已封装好

缺点：有序输出模式下会缓存数据，Checkpoint 时会保存这些状态



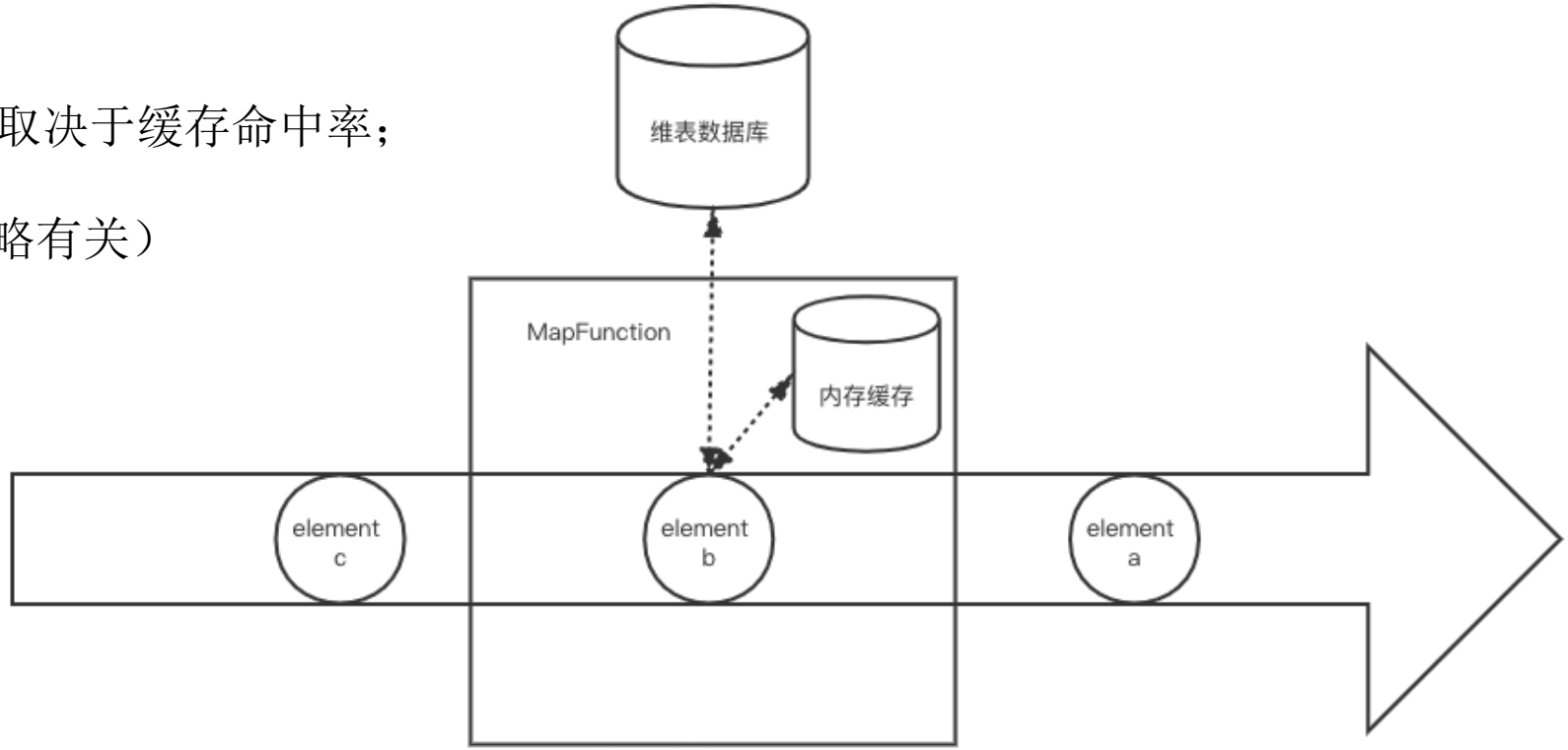
实时数据库带缓存查找关联

优点：引入一层缓存（Guava Cache）解决对数据库造成太大压力的问题

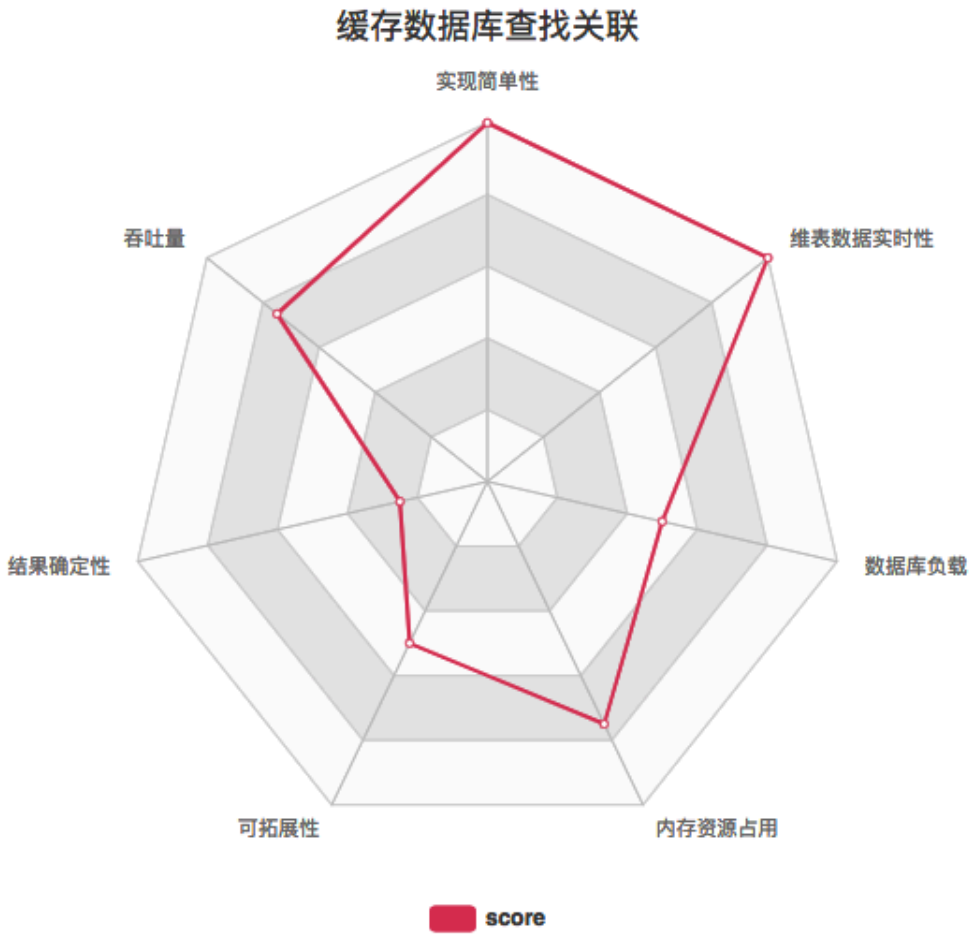
缓存数据不会进行 Checkpoint

缺点：冷启动时仍会造成一定压力，后续取决于缓存命中率；

维表的更新不能及时感知（和缓存 TTL 策略有关）



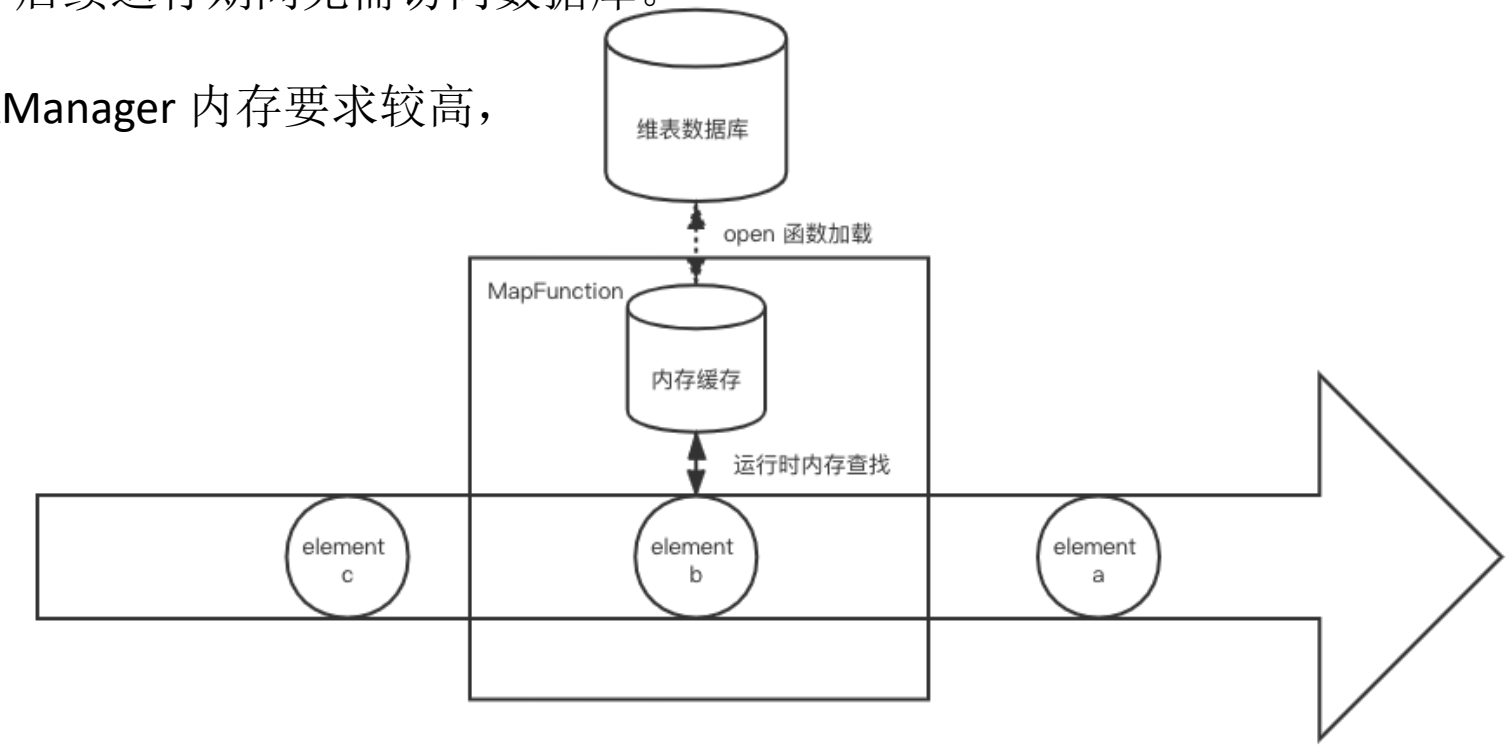
实时数据库带缓存查找关联



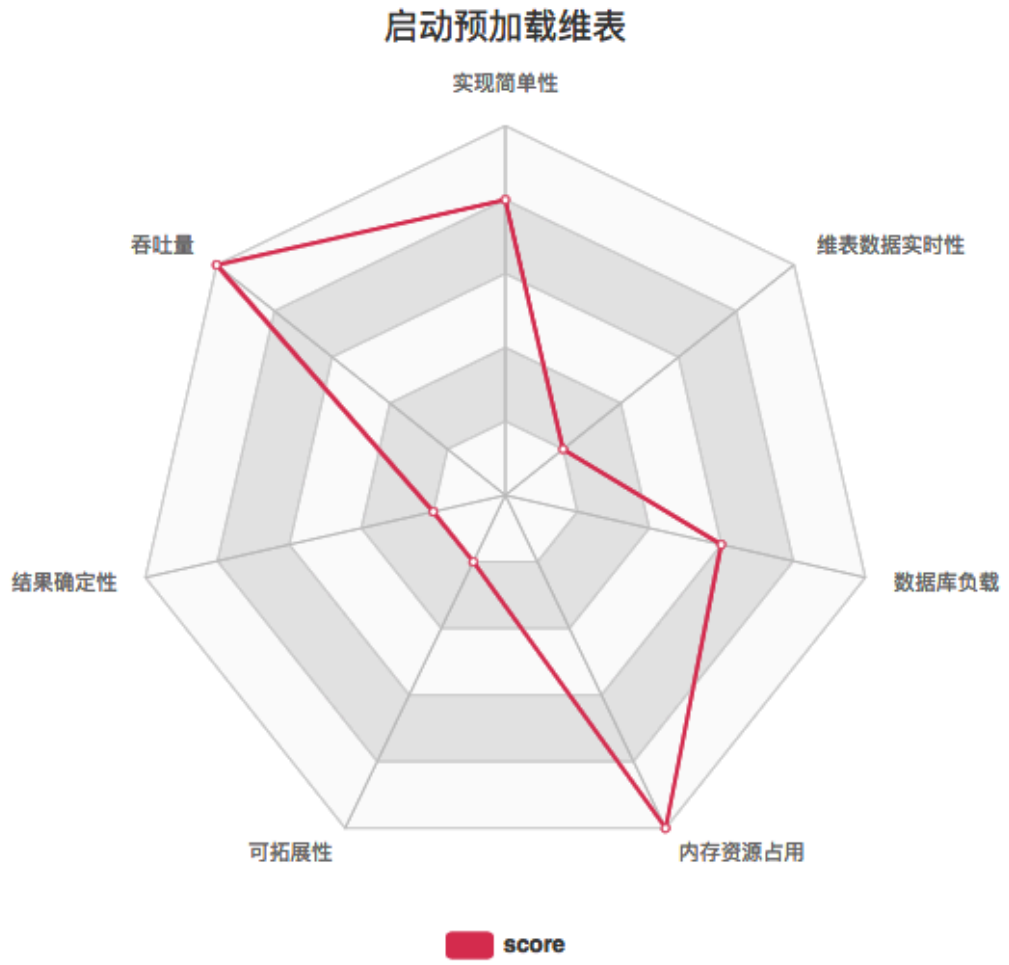
启动预加载维表关联

优点：作业初始化 open 方法会从数据库查找维表拷贝到内存，无论作业手动启动还是失败重启都会初始化得到最新的维表，后续运行期间无需访问数据库。

缺点：初始化拷贝的压力还是比较大，TaskManager 内存要求较高，维表数据不能更新

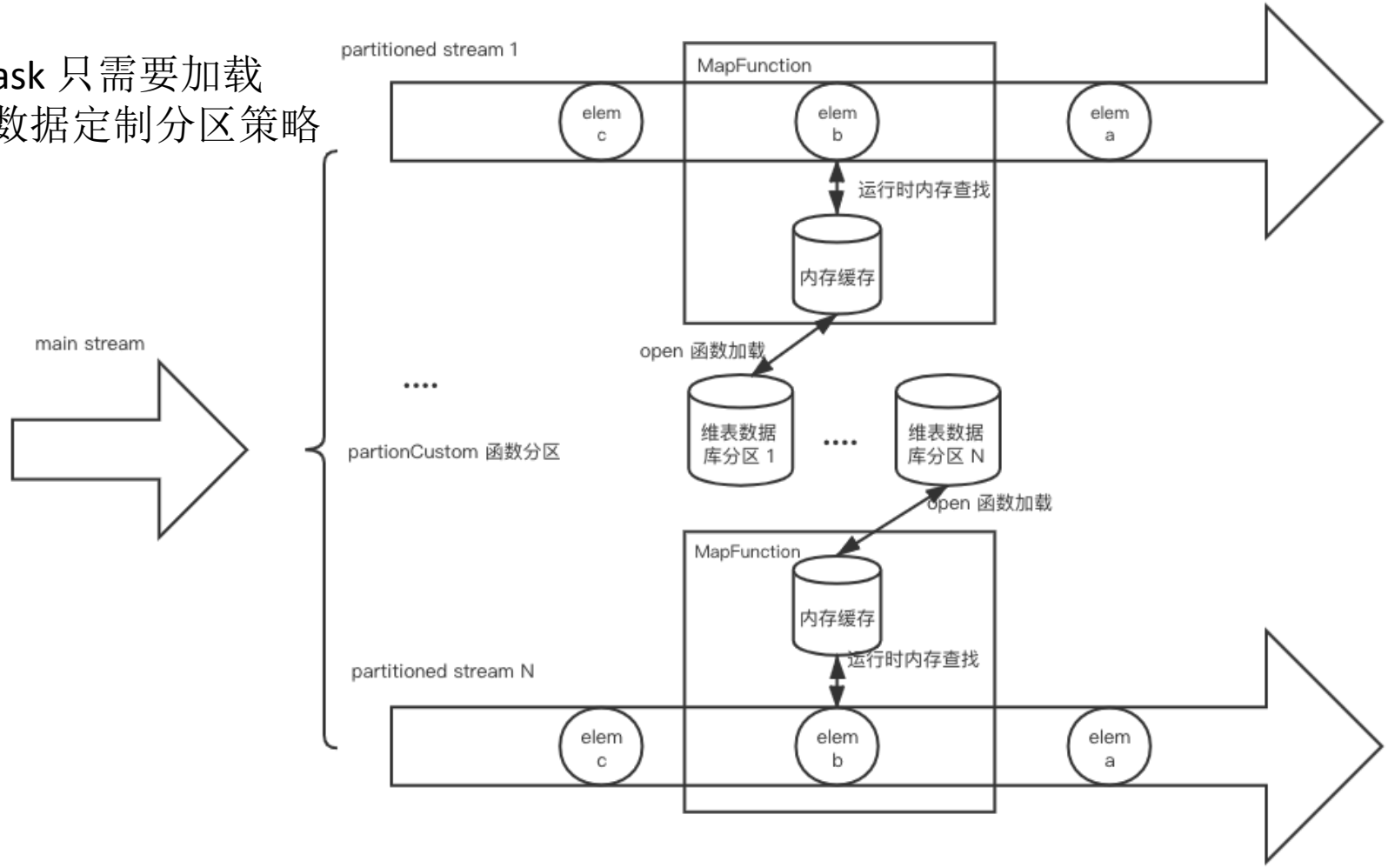


启动预加载维表关联

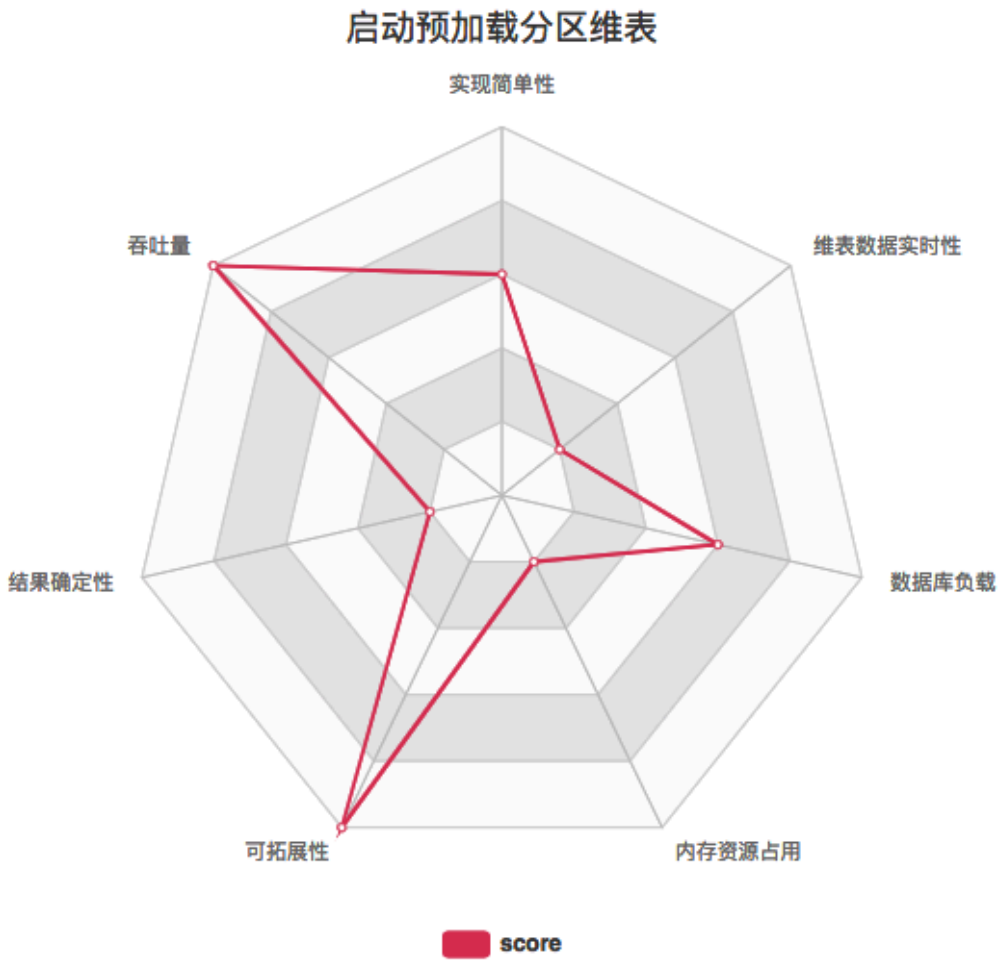


启动预加载分区维表关联

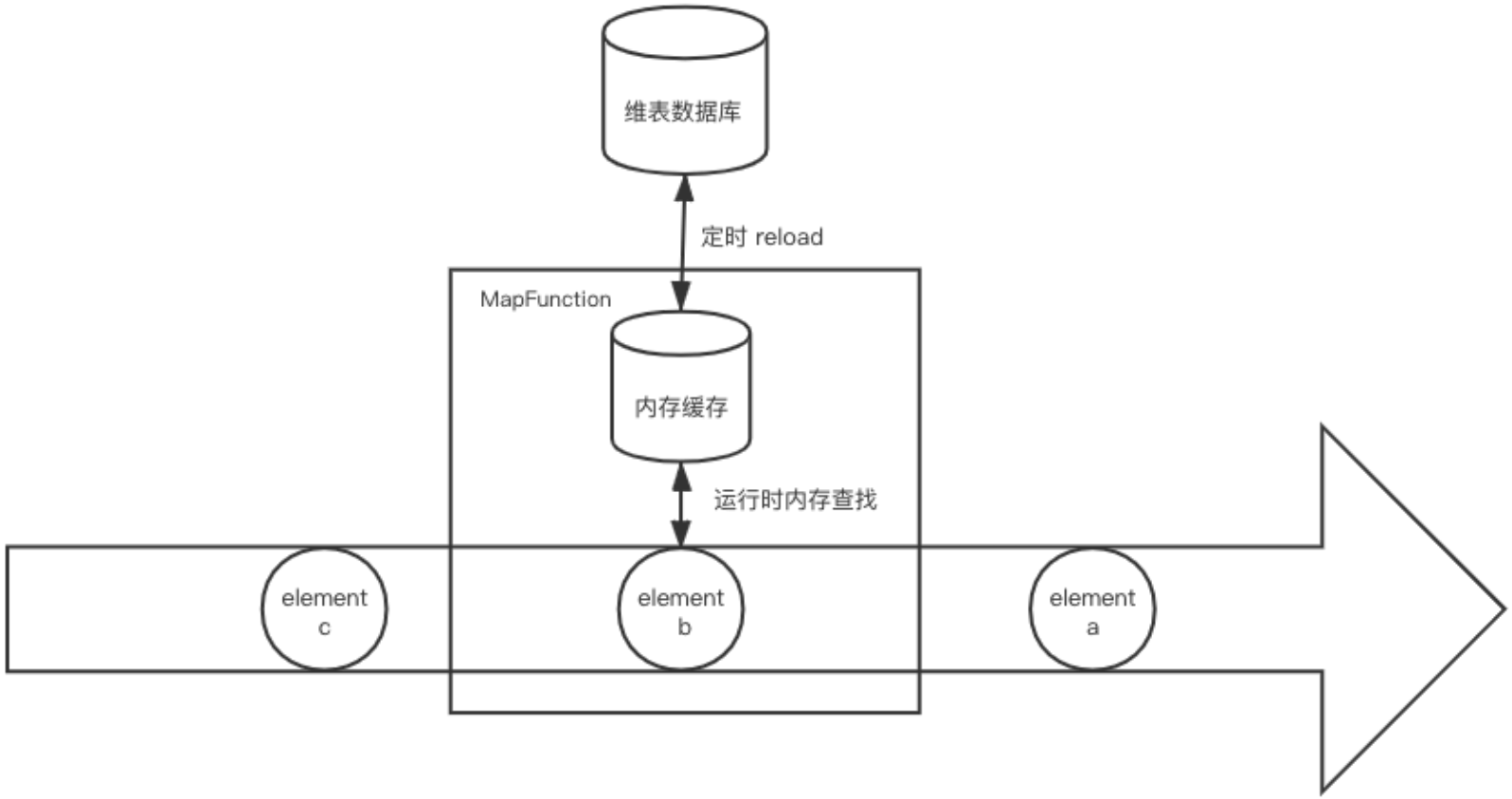
将数据流按字段进行分区，然后每个 Subtask 只需要加载
对应分区范围的维表数据，需要根据业务数据定制分区策略



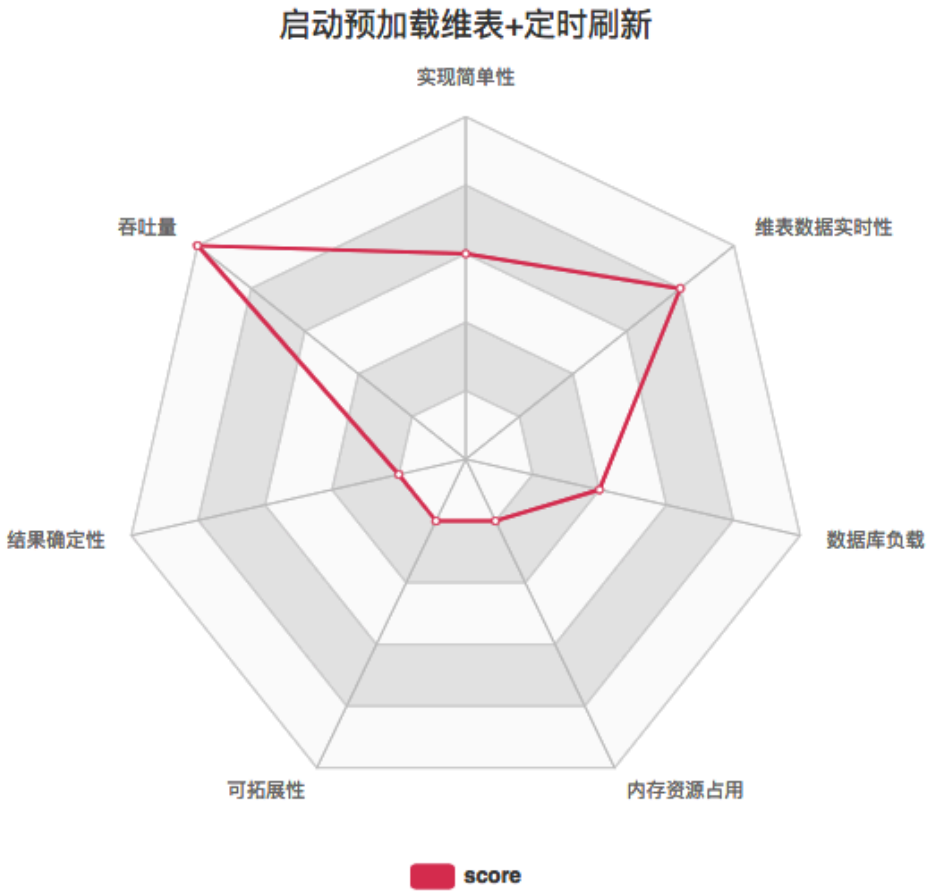
启动预加载分区维表关联



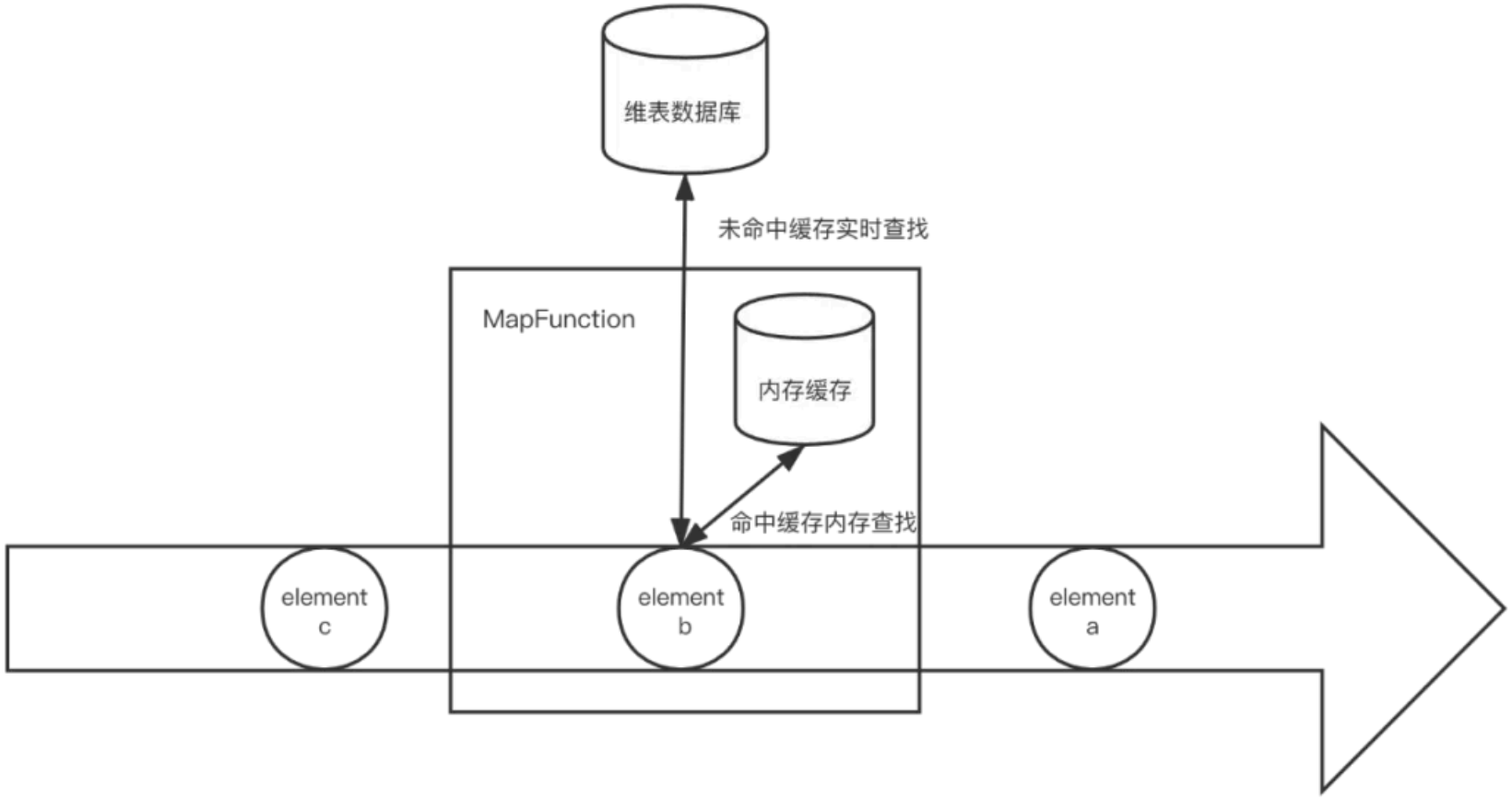
启动预加载维表并定时刷新关联



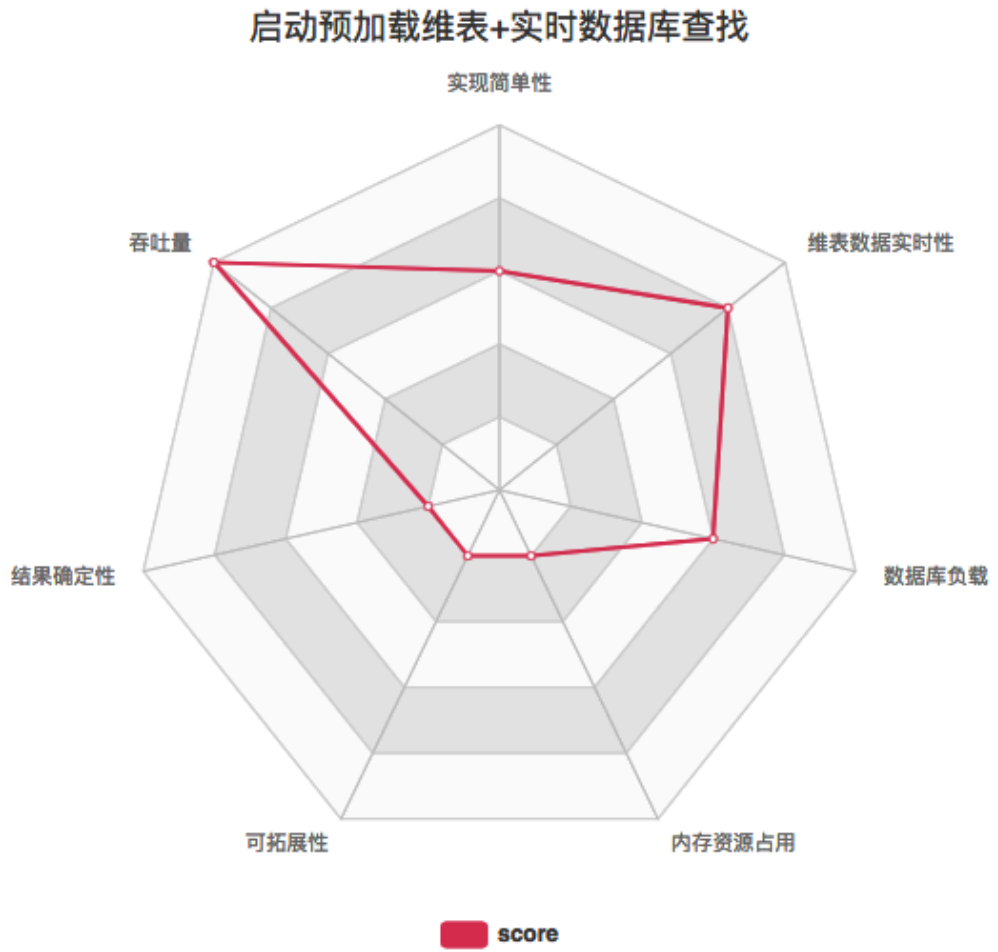
启动预加载维表并定时刷新关联



启动预加载维表+实时数据库查找

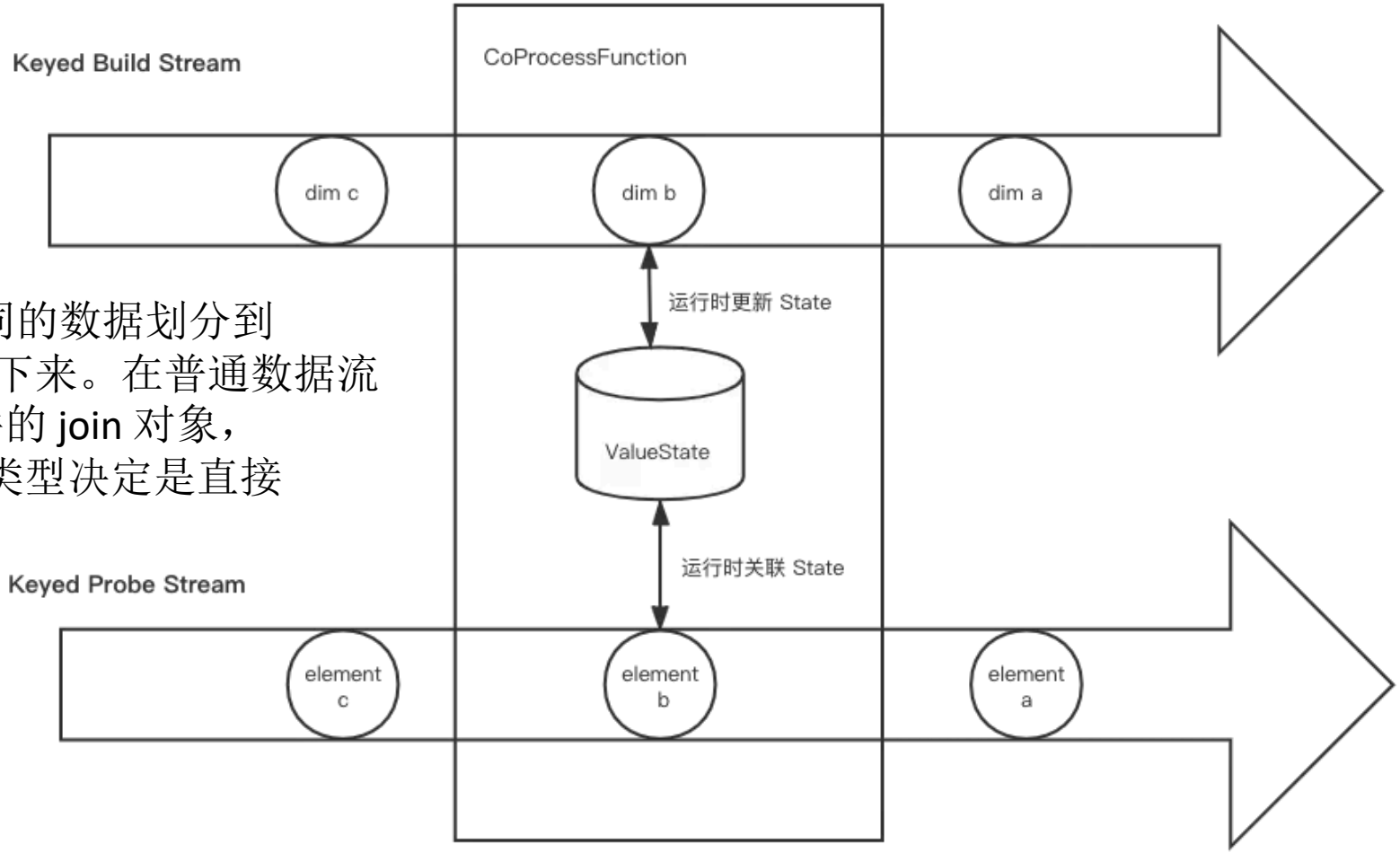


启动预加载维表+实时数据库查找

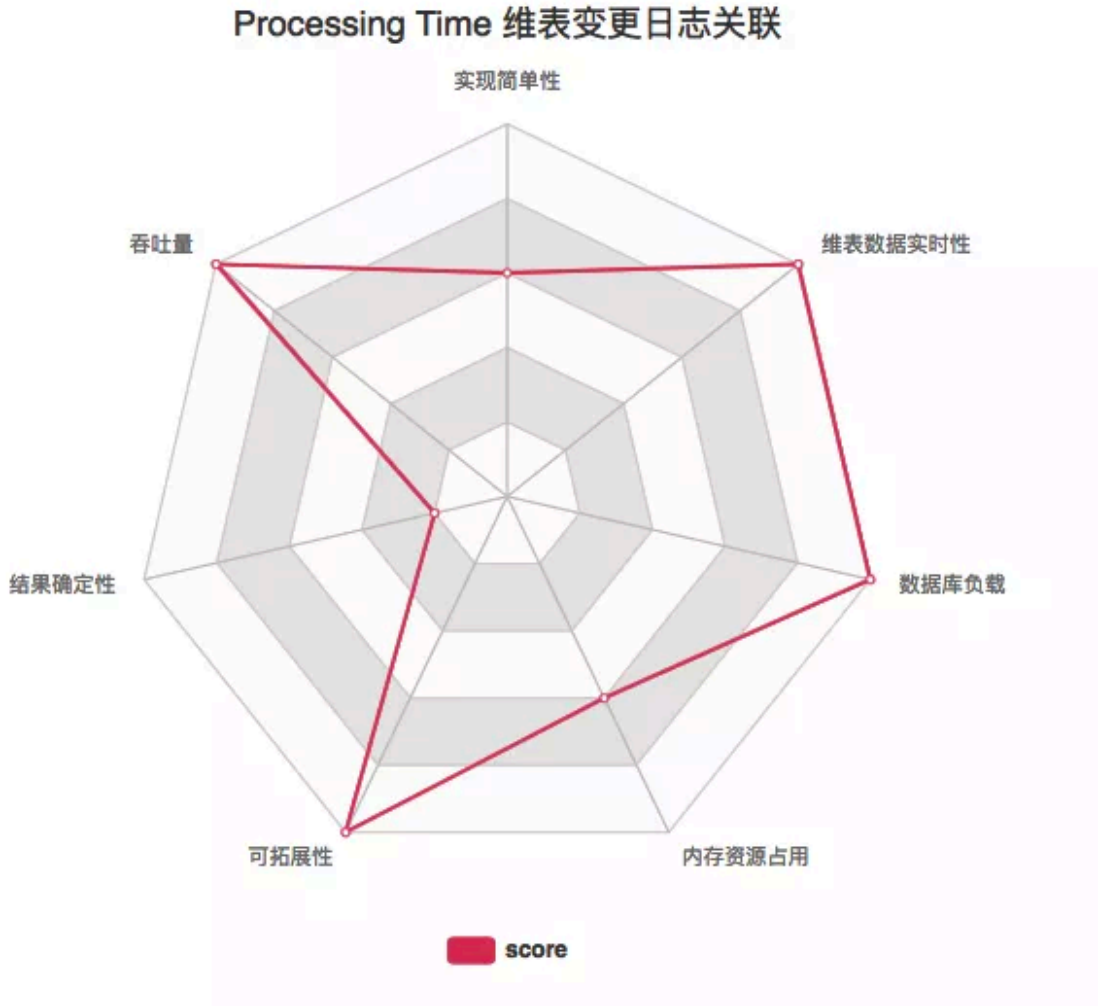


Processing Time 维表变更日志关联

利用 keyby 将两个数据流中关联字段值相同的数据划分到同一个分区，然后用 状态将维表数据保存下来。在普通数据流进到函数时，到 State 中查找有无符合条件的 join 对象，若有则关联输出结果，若无则根据 join 的类型决定是直接丢弃还是与空值关联。

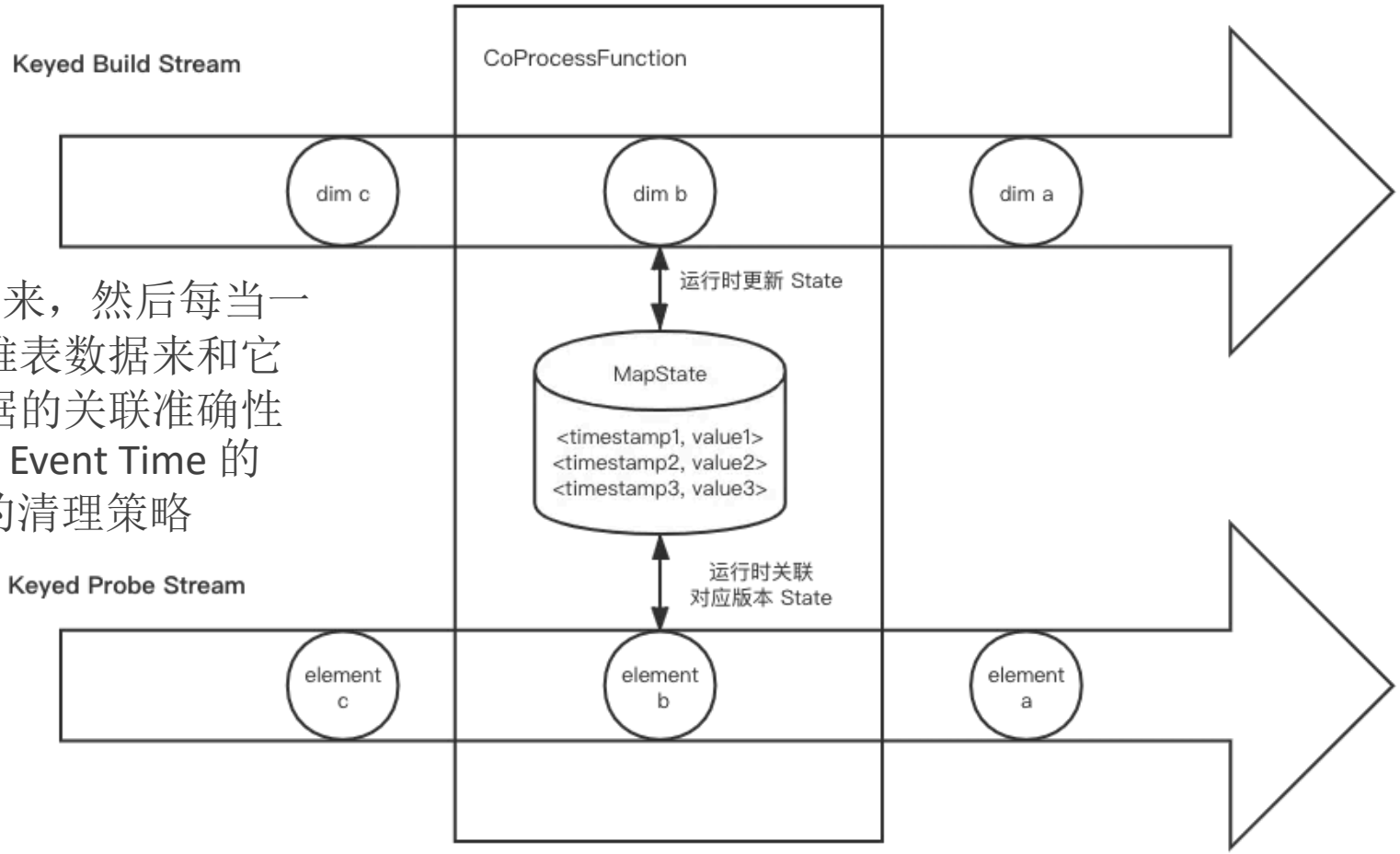


Processing Time 维表变更日志关联

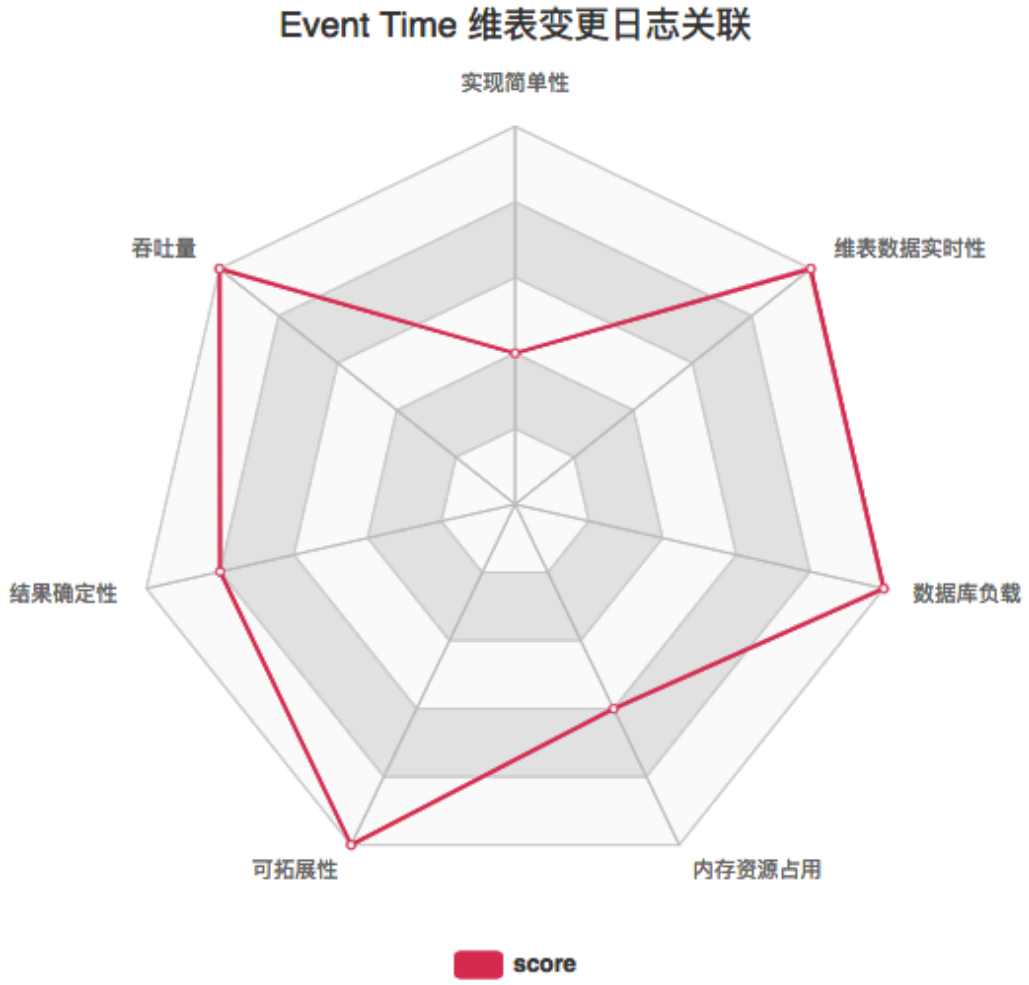


Event Time 维表变更日志关联

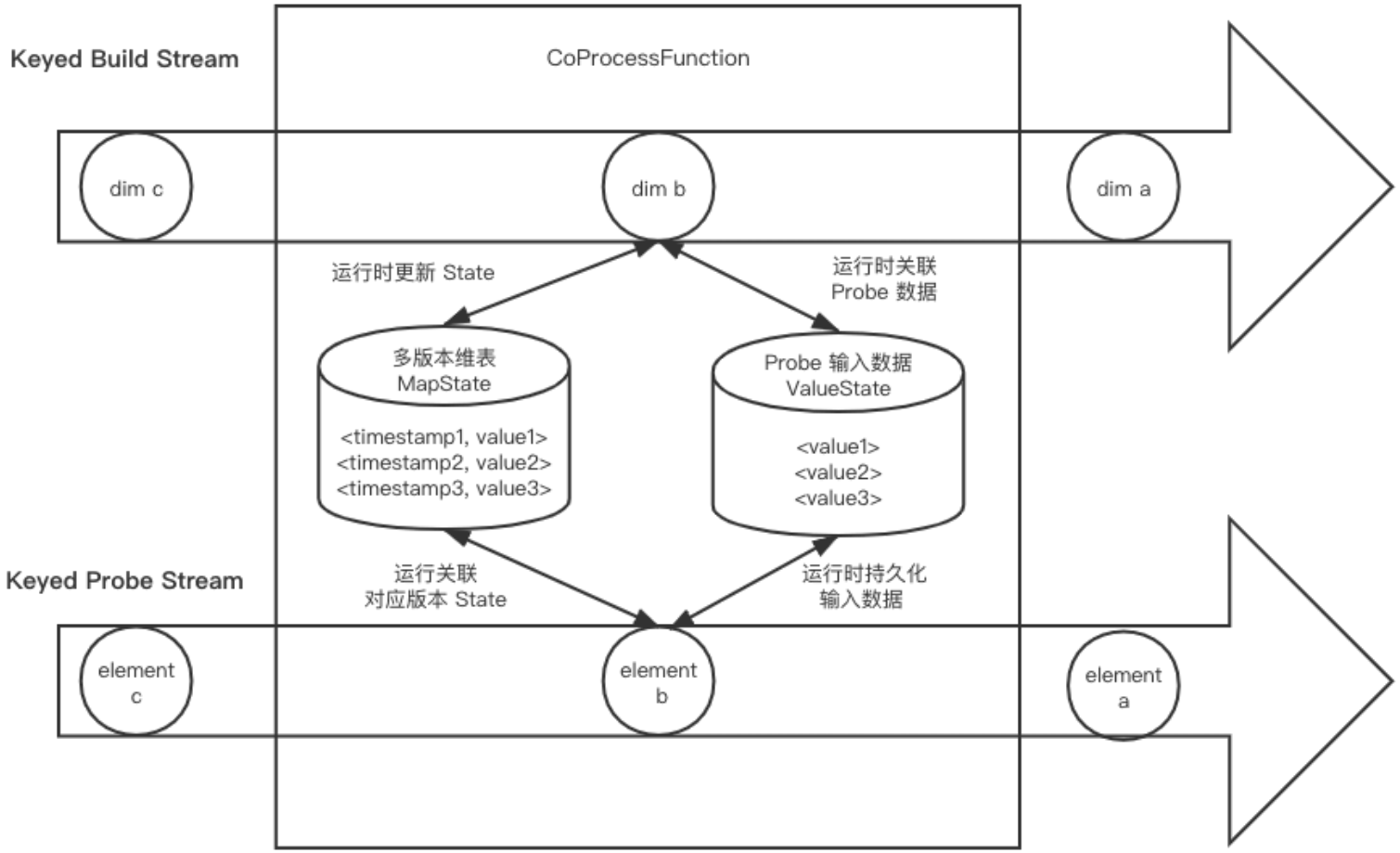
将维表 changelog 的多个时间版本都记录下来，然后每当一条记录进来，我们会找到对应时间版本的维表数据来和它关联，而不是总用最新版本，因此延迟数据的关联准确性大大提高。不过因为目前 State 并没有提供 Event Time 的 TTL，因此我们需要自己设计和实现 State 的清理策略



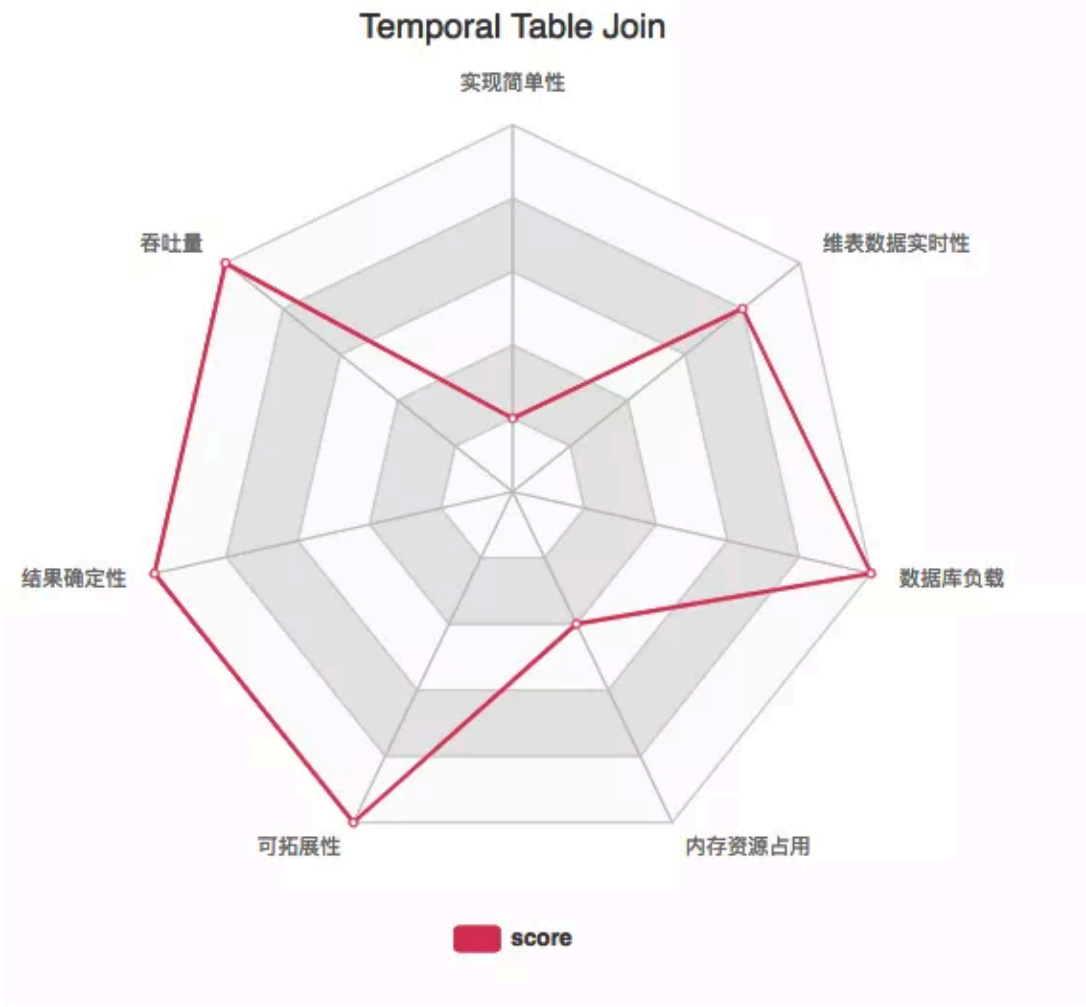
Event Time 维表变更日志关联



Temporal Table Join



Temporal Table Join



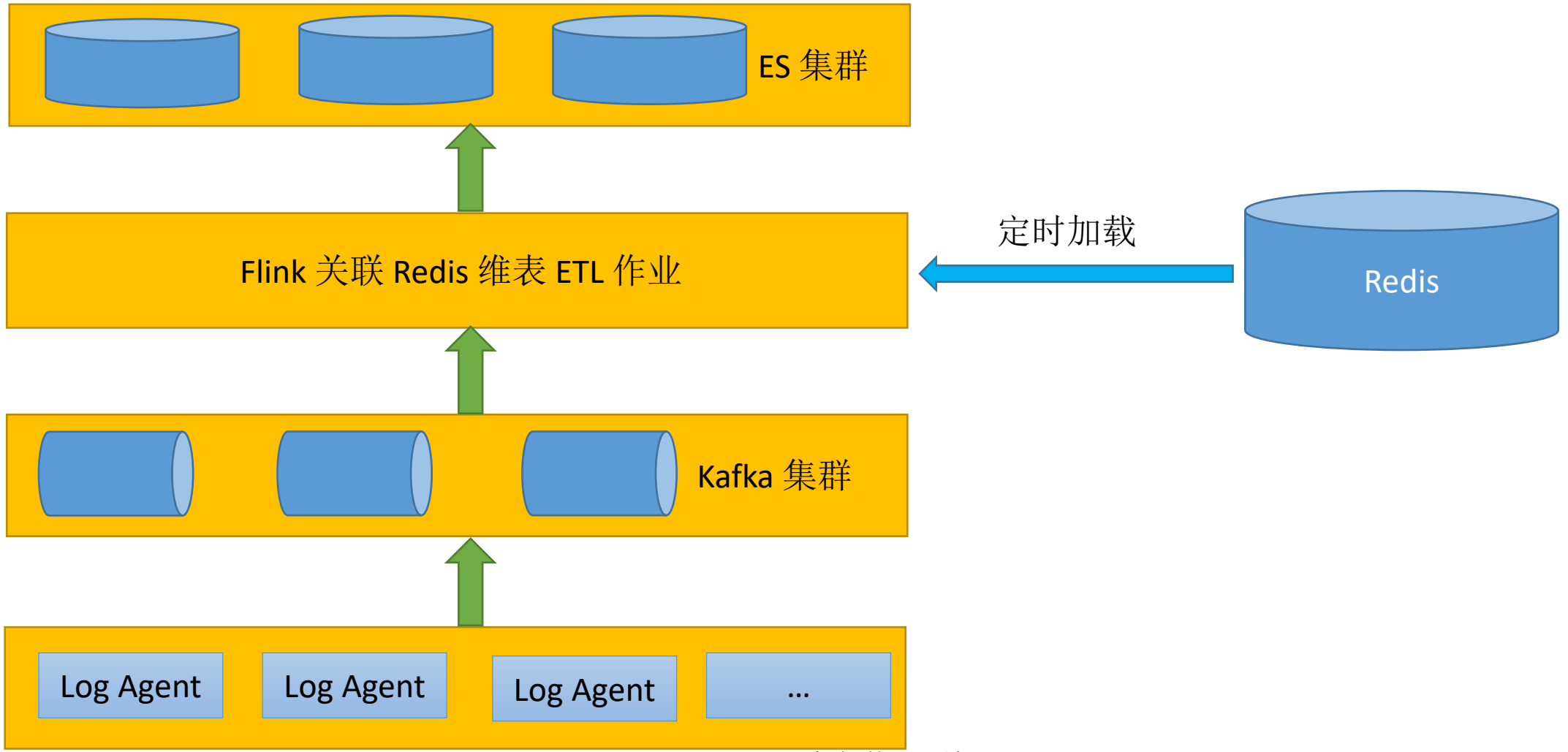
04

Flink 案例实战演练

维表 Join

日志流数据关联 Redis 维表数据

Apache Flink



THANKS