从流处理到数据分析：RisingWave+ClickHouse的实时数据架构

RisingWave

自我介绍

- 温一鸣
  - 2022年3月加入RisingWave，负责存储引擎，Connector，向量索引的核心设计与开发。
  - 前字节跳动火山引擎推荐架构工程师
  - 硕士毕业于卡内基梅隆大学(CMU)

RisingWave

- 分布式流式数据库
- rust开发
- 云原生+存算分离
- 物化视图 (materialized view)

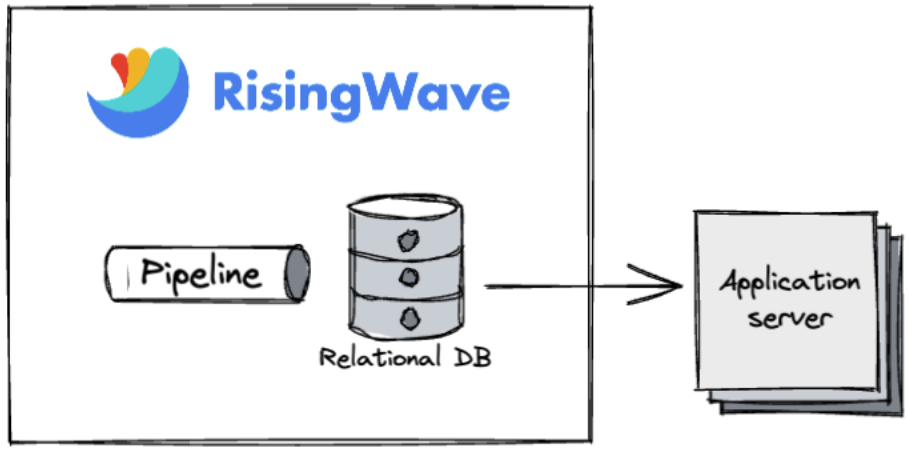
github开源

8.3K github stars

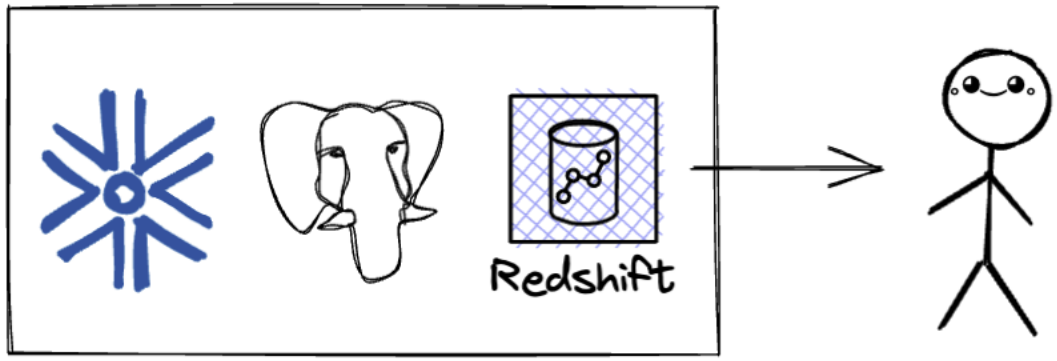
~200 gitHub contributors

~3K slack members

流处理 vs OLAP



- 增量更新
- **事件**驱动异步处理
- **点查提前定义的物化**视图



- 全量计算
- **用**户主动查询**同步**驱动计算
- 面向随机的复杂分析查询

```
CREATE TABLE auction (
    "id" BIGINT,
    "item_name" VARCHAR
);
```

id	name
1	Tesla
2	benz

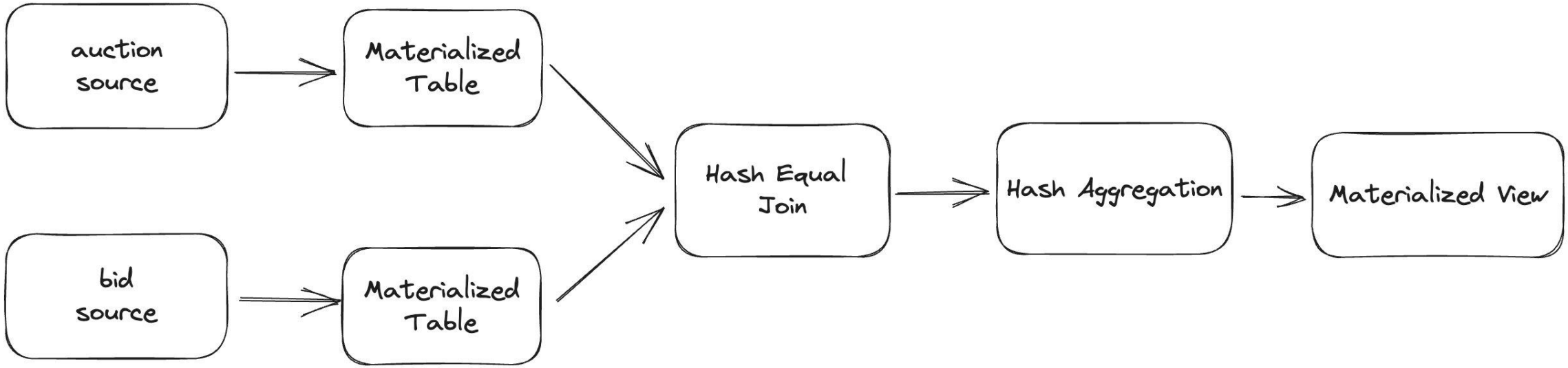
id	auc_id	price
1	1	100
2	2	150
3	1	120

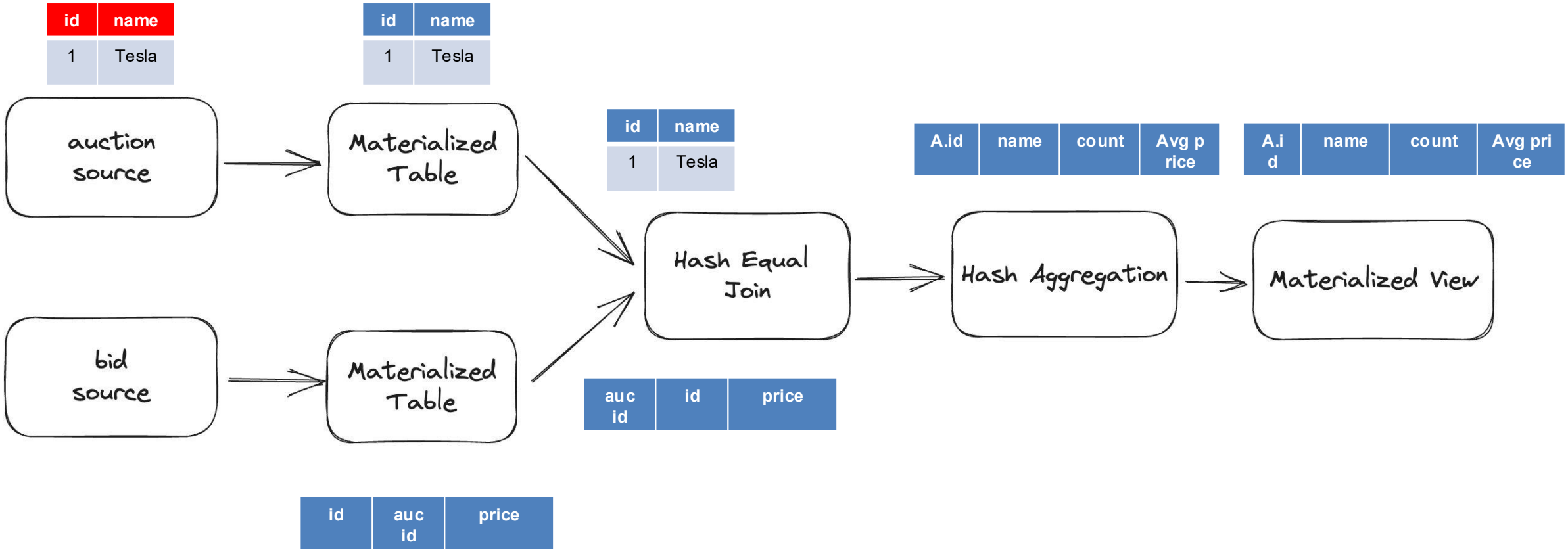
```
CREATE TABLE bid (
"id" BITINT,
    "auction" BIGINT,
    "price" BIGINT
);
```

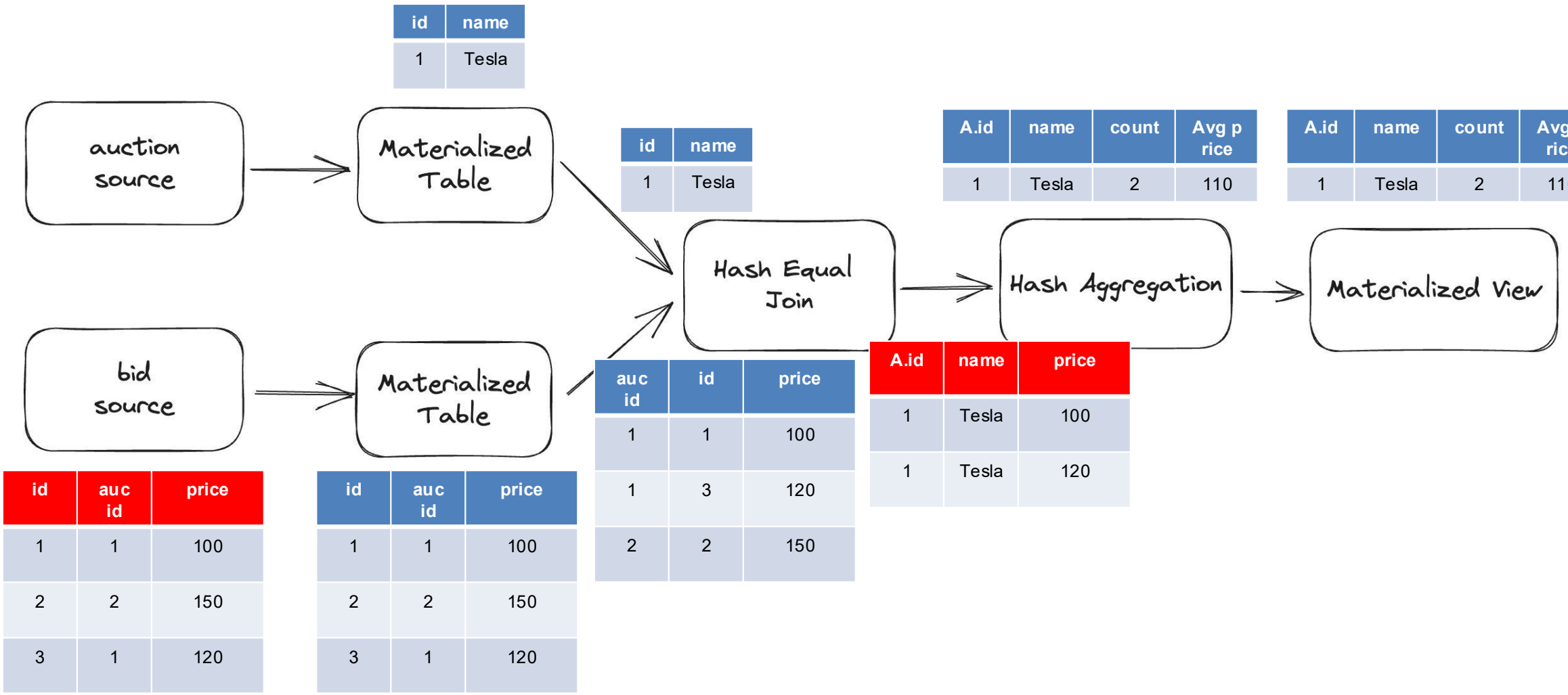
```
SELECT
    A.id AS auction_id,
    A.item_name AS auction_item_name,
    COUNT(B.auction) AS bid_count,
    SUM(B.price) / COUNT(B.auction) as bid_avg_price
FROM auction A
JOIN bid B ON A.id = B.auction
GROUP BY A.id, A.item_name;
```

A.id	name	count	Avg price
1	Tesla	2	110
2	benz	1	150

```
CREATE MATERIALIZED VIEW auction_bid_summary AS
  SELECT
    A.id AS auction_id,
    A.item_name AS auction_item_name,
    COUNT(B.auction) AS bid_count,
    SUM(B.price) / COUNT(B.auction) as bid_avg_price
  FROM auction A
  JOIN bid B ON A.id = B.auction
  GROUP BY A.id, A.item_name;
```




id
name
1
Tesla
id
name
1
Tesla
auction source
Materialized Table
id
name
1
Tesla
A.id
name
count
Avg price
A.id
name
count
Avg price
Hash Equal Join
Hash Aggregation
Materialized View
bid source
Materialized Table
auc id
id
price
id
auc id
price



id	name
1	Tesla

id	name
1	Tesla

id	auc id	price
1	1	100
2	2	150
3	1	120

id	auc id	price
1	1	100
2	2	150
3	1	120

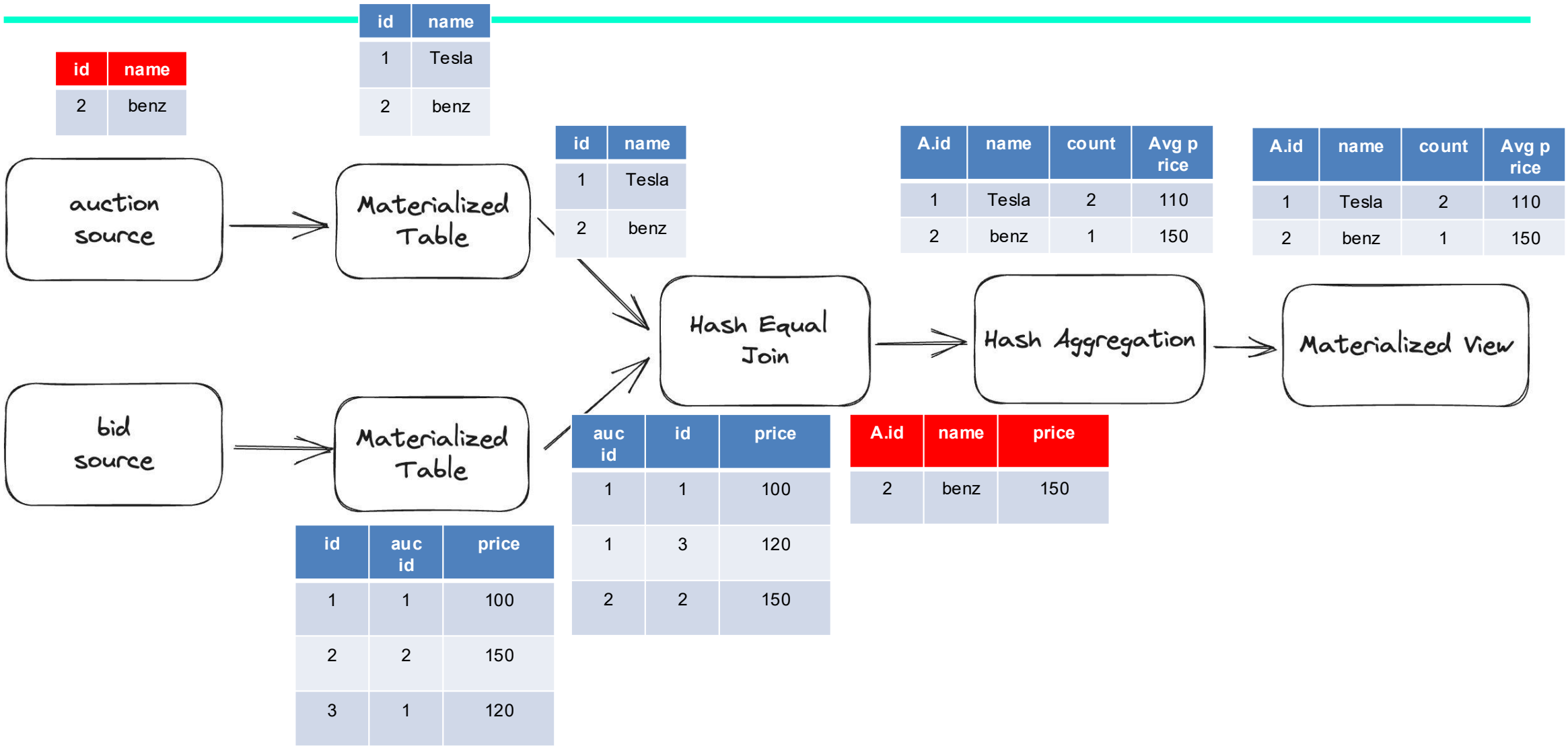
auc id	id	price
1	1	100
1	3	120
2	2	150

A.id	name	price
1	Tesla	100
1	Tesla	120

A.id	name	count	Avg price
1	Tesla	2	110

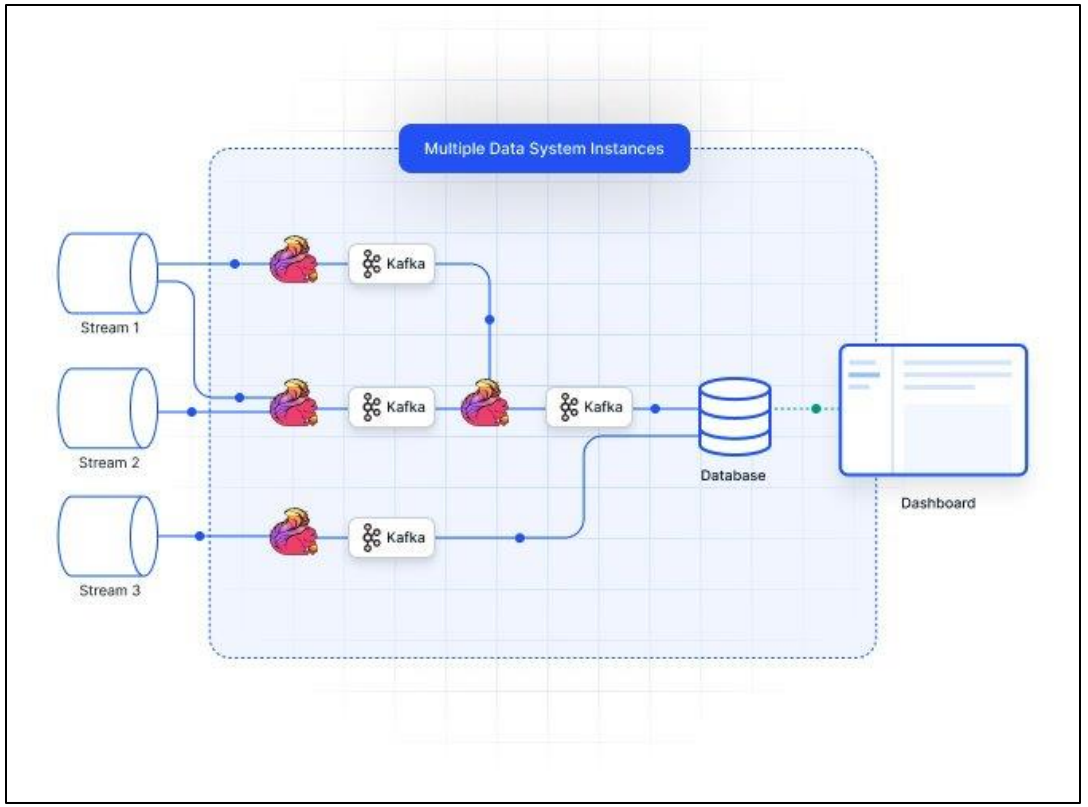
A.id	name	count	Avg price
1	Tesla	2	110

RisingWave


id
name
2
benz
id
name
1
Tesla
2
benz
auction source
Materialized Table
id
name
1
Tesla
2
benz
A.id
name
count
Avg price
1
Tesla
2
110
2
benz
1
150
A.id
name
count
Avg price
1
Tesla
2
110
2
benz
1
150
Hash Equal Join
Hash Aggregation
Materialized View
bid source
Materialized Table
auc id
id
price
1
1
100
1
3
120
2
2
150
A.id
name
price
2
benz
150
id
auc id
price
1
1
100
2
2
150
3
1
120

RisingWave

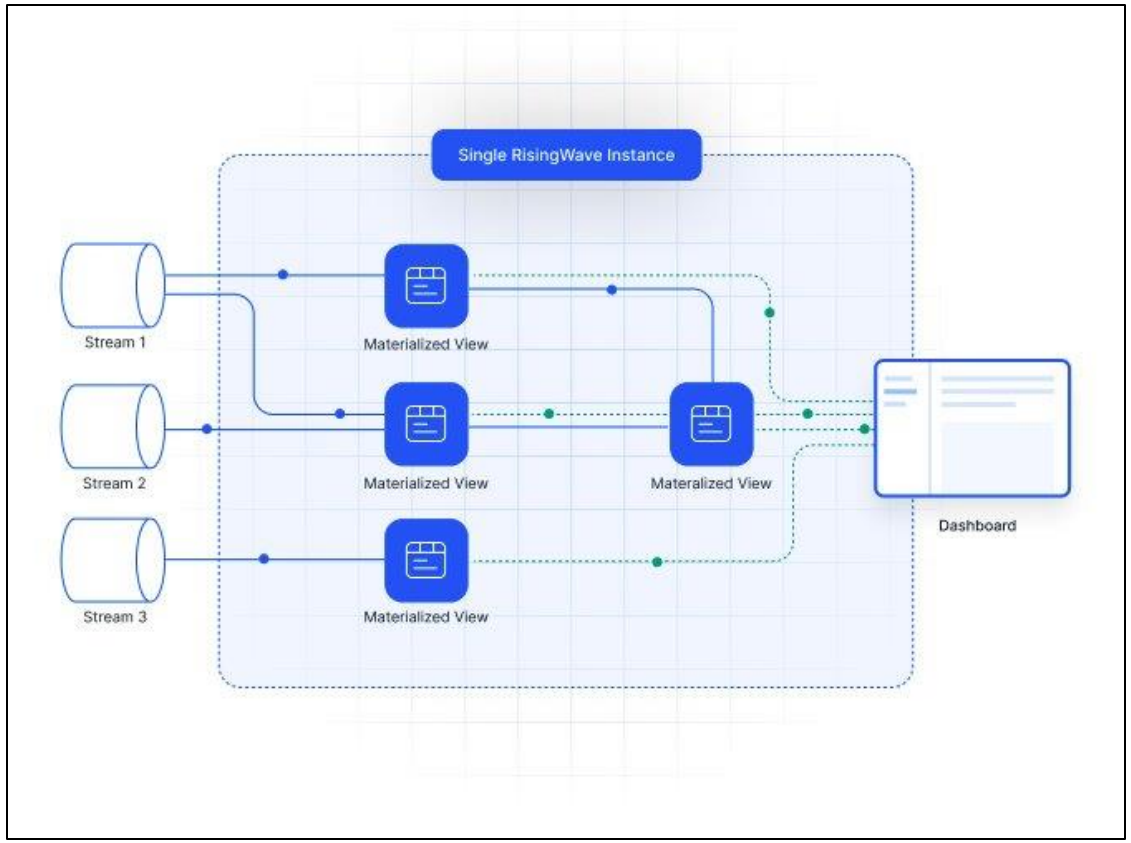
流式数据库 vs 流处理引擎




```
CREATE TABLE employee_information (
    emp_id INT,
    name VARCHAR,
    dept_id INT
) WITH (
    'connector' = 'kafka',
    'topic' = 'employee_information'
    ...
);


CREATE TABLE department_counts (
 dept_id INT,
 emp_count INT
) WITH (
    'connector' = xxx, // 外部系统，kafka or MySQL ...
...
);

INSERT INTO department_counts
SELECT
   dept_id,
   COUNT(*) as emp_count
FROM employee_information
GROUP BY dept_id;
```

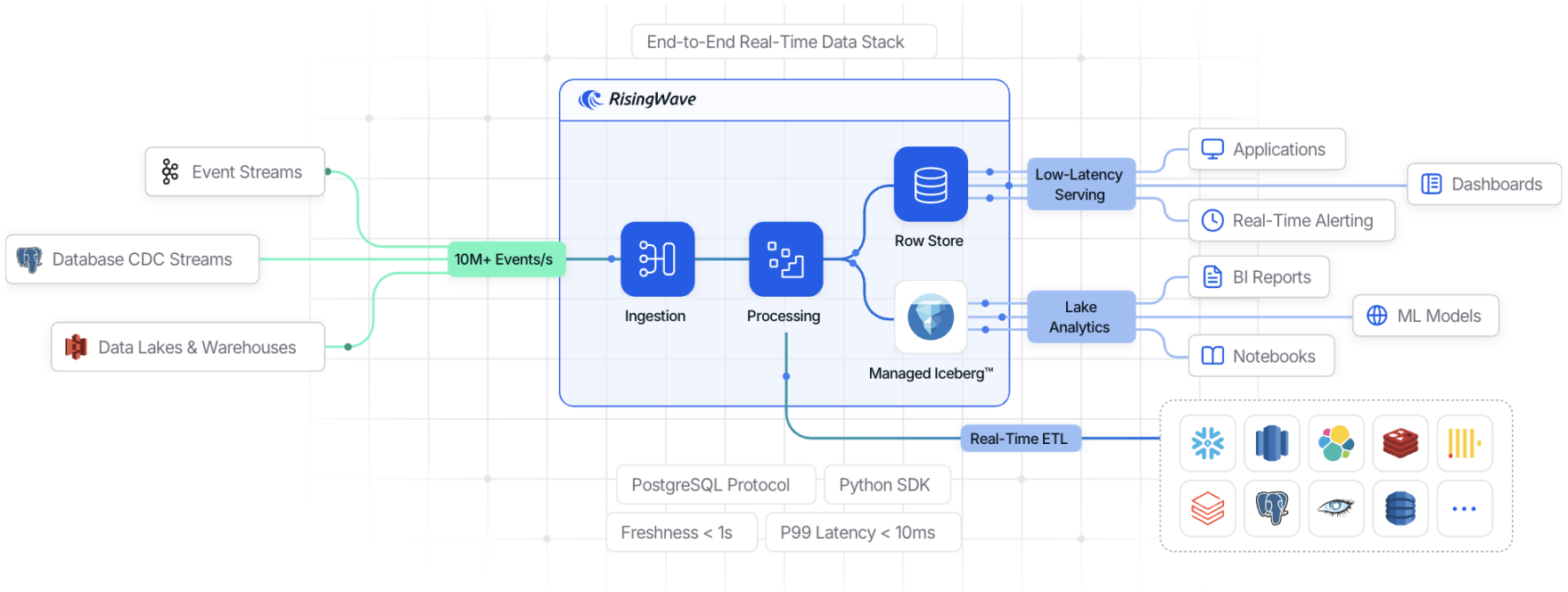
RisingWave

流式数据库 vs 流处理引擎



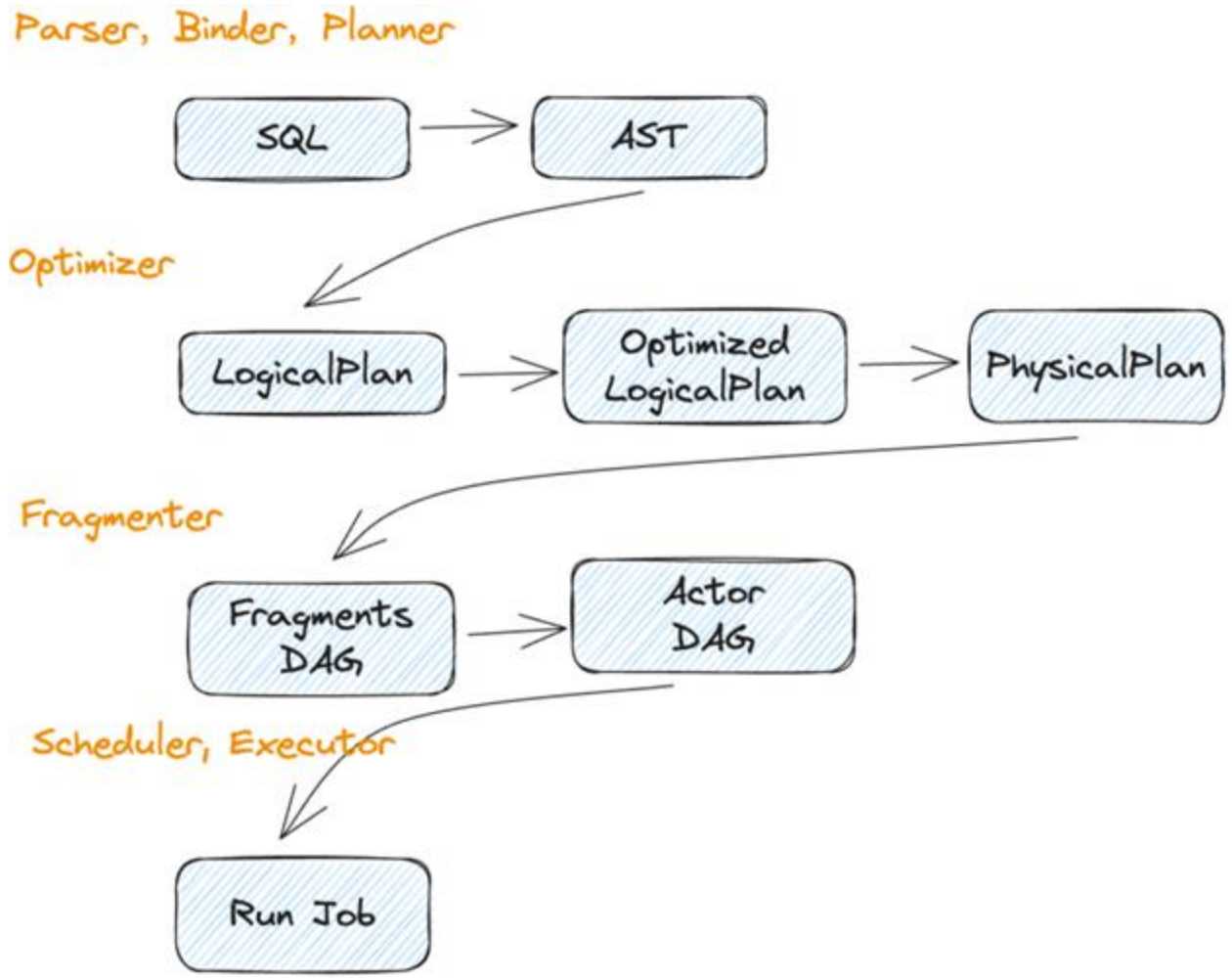

```
CREATE TABLE employee_information (
    emp_id INT,
    name VARCHAR,
    dept_id INT
) WITH (
    'connector' = 'kafka',
    'topic' = 'employee_information'
    ...
);


CREATE MATERIALIZED VIEW department_counts
AS SELECT
   dept_id,
   COUNT(*) as emp_count
FROM employee_information
GROUP BY dept_id;


SELECT * FROM department_counts where dept_id = 1;
```

RisingWave


End-to-End Real-Time Data Stack
RisingWave
Event Streams
Database CDC Streams
Data Lakes & Warehouses
10M+ Events/s
Ingestion
Processing
Row Store
Managed Iceberg™
Low-Latency Serving
Lake Analytics
Applications
Dashboards
Real-Time Alerting
BI Reports
ML Models
Notebooks
Real-Time ETL
PostgreSQL Protocol
Python SDK
Freshness < 1s
P99 Latency < 10ms

SQL即流处理

- 基于SQL构建流作业
- 丰富的查询优化
  - 列裁剪，Filter下推
  - 子查询解关联
  - Join Ordering
  - Agg改写
  - 共享SubPlan
  - 公共表达式提取
  - 自动索引选择
  - Streaming改写

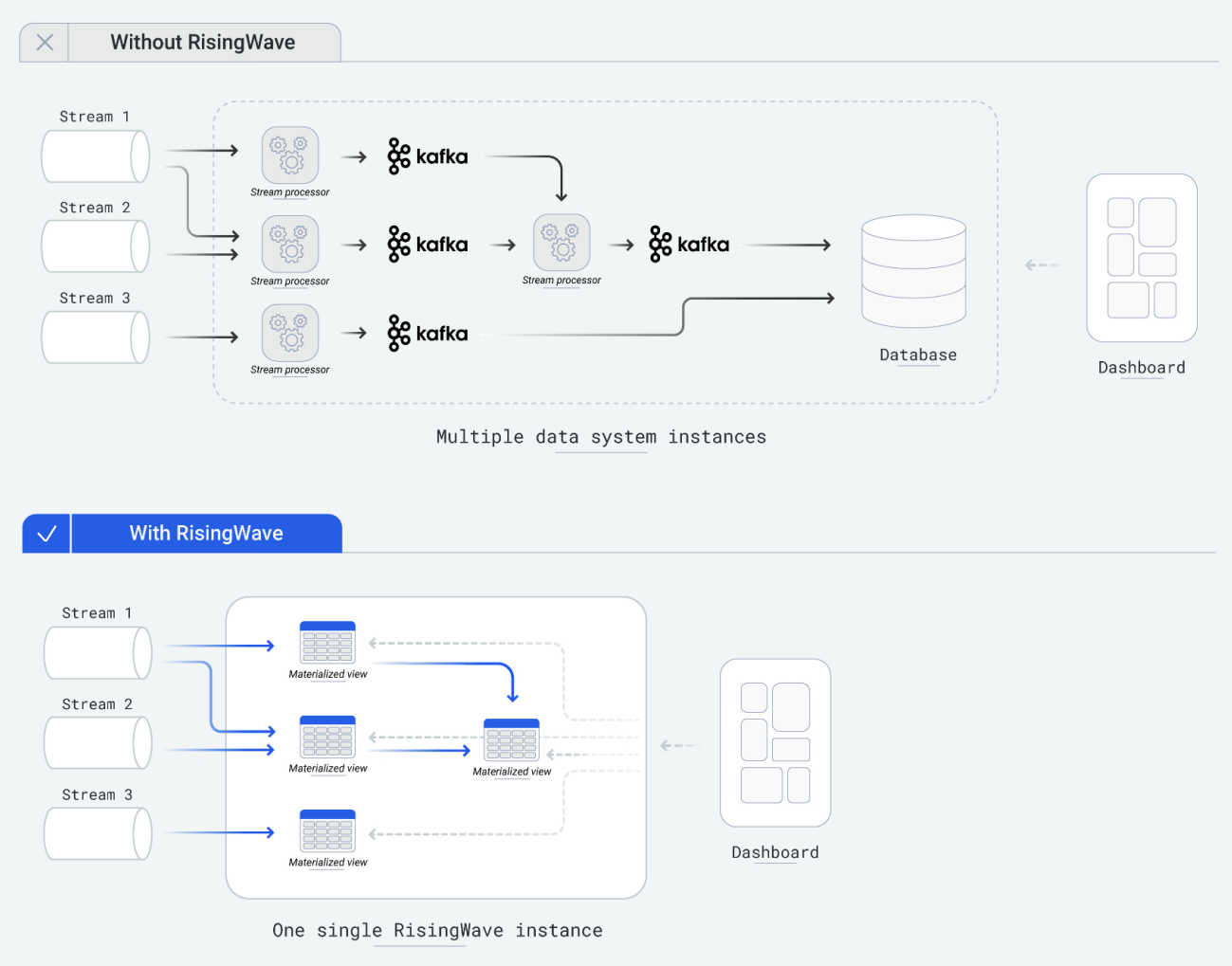


物化视图 MATERIALIZED VIEW

- Materialized View**增量**实时维护流处理结果
- **支持丰富的**SQL语法
  - Join, Agg, Filter, **集合运算**, **窗口函数**, **子**查询, UDF, Grouping Sets, CTE, Lateral, Watermark, …
  - PG**常**见表达式, Lambda**表达式**, **半**结构化数据处理函数
- **支持**实时查询Materialized View结果

```
CREATE MATERIALIZED VIEW ad_ctr AS
SELECT
    ad_clicks.ad_id AS ad_id,
    ad_clicks.clicks_count :: NUMERIC / ad_impressions.impressions_count AS ctr
FROM
    (
        SELECT
            ad_impression.ad_id AS ad_id,
            COUNT(*) AS impressions_count
        FROM
            ad_impression
        GROUP BY
            ad_id
    ) AS ad_impressions
    JOIN (
        SELECT
            ai.ad_id,
            COUNT(*) AS clicks_count
        FROM
            ad_click AS ac
            LEFT JOIN ad_impression AS ai ON ac.bid_id = ai.bid_id
        GROUP BY
            ai.ad_id
    ) AS ad_clicks
    ON ad_impressions.ad_id = ad_clicks.ad_id;
```

MV on MV

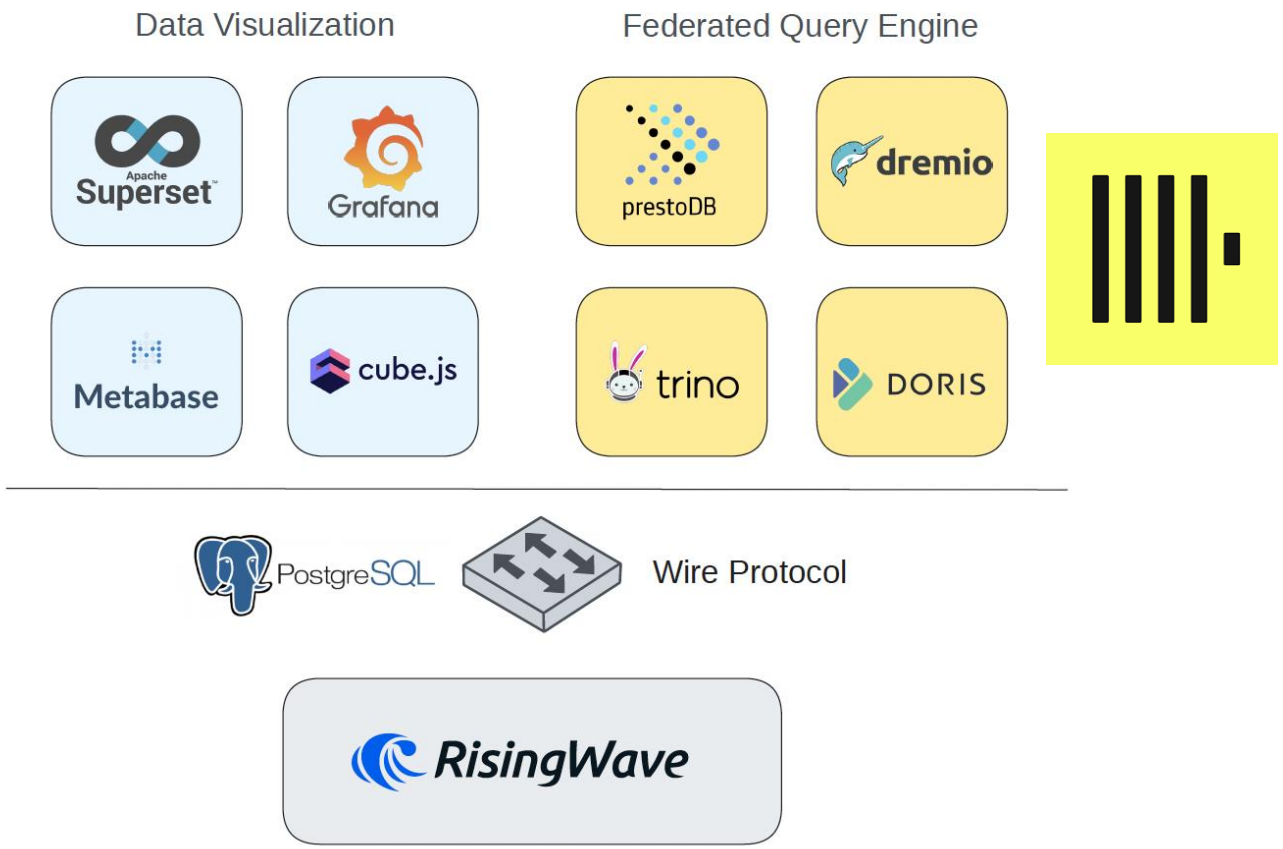


- **支持MV-on-MV构建层级Streaming Pipeline**
- 流式数仓建模
- 中间结果可查询可复用

```
CREATE TABLE employee_information (
    ...
) ...

CREATE MATERIALIZED VIEW department_counts
AS SELECT
   dept_id,
   COUNT(*) as emp_count
FROM employee_information
GROUP BY dept_id;


CREATE MATERIALIZED VIEW top_5_count_department
AS SELECT
    dept_id
FROM department_counts
ORDER BY emp_count DESC
LIMIT 5;
```

RisingWave

PostgreSQL兼容

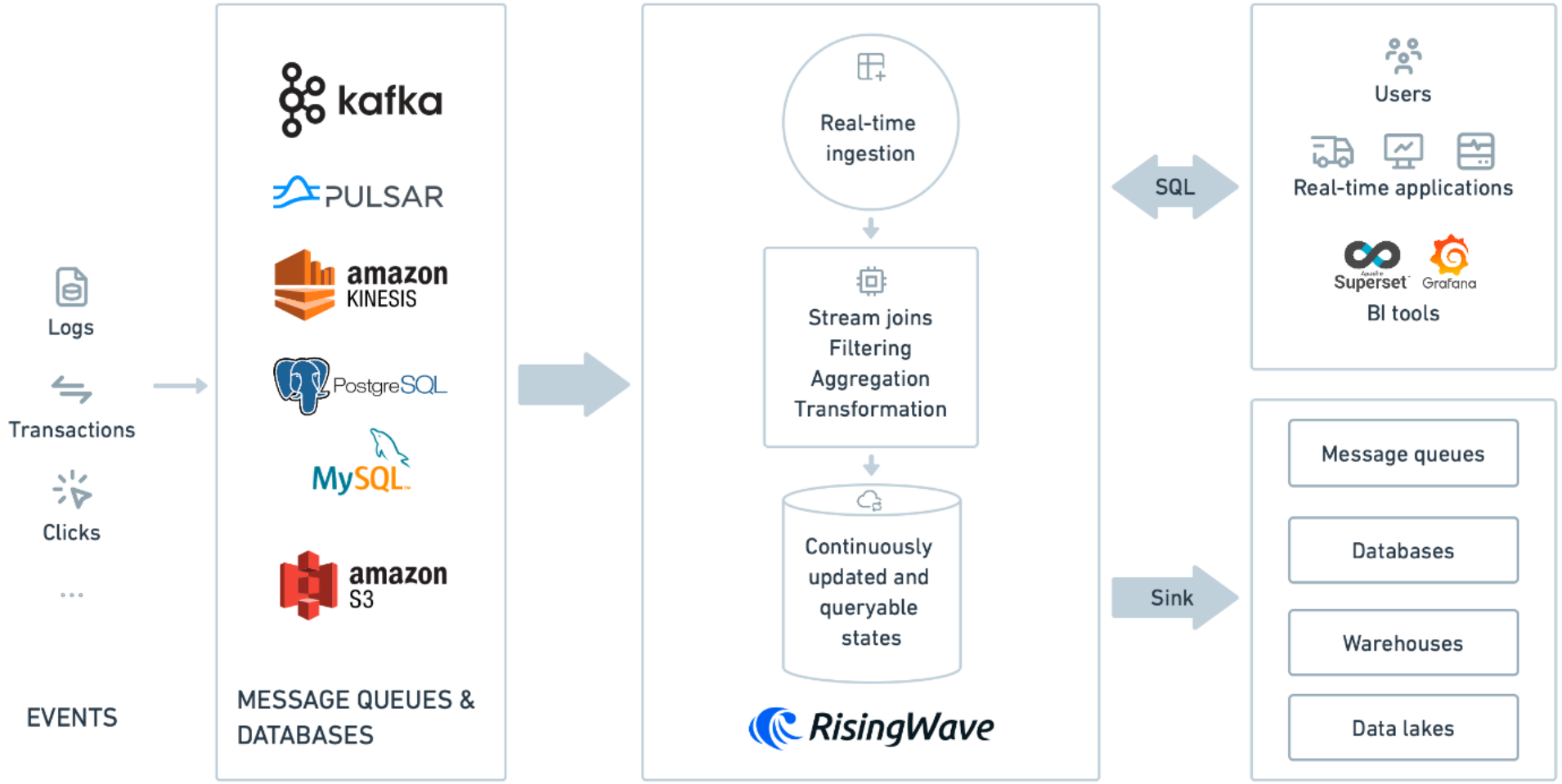


```
→  ~ psql -h localhost -p 4566 -d dev -U root
psql (14.19 (Homebrew), server 13.14.0)
Type "help" for help.

dev=> CREATE TABLE numbers(id INT);
CREATE_TABLE
dev=> INSERT INTO numbers VALUES (1), (2);
INSERT 0 2
dev=> CREATE MATERIALIZED VIEW sum AS SELECT sum(id) as sum FROM numbers;
CREATE_MATERIALIZED_VIEW
dev=> SELECT * FROM sum;
 sum
-----
   3
(1 row)

dev=> INSERT INTO numbers VALUES (3);
INSERT 0 1
dev=> SELECT * FROM sum;
 sum
-----
   6
(1 row)
```

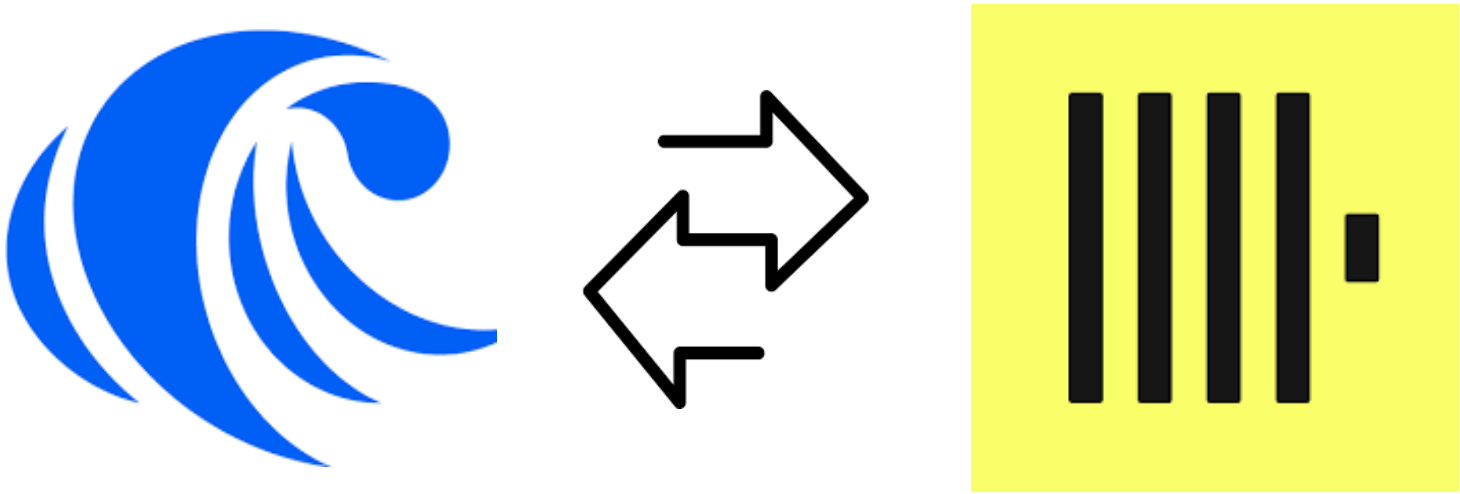
Rust 实现

- 内存安全
- 低运行时额外开销
  - 底层机器语言
  - 无虚拟机，无GC
  - 零成本抽象(类型静态确定，无需动态转发)
- 异步 (async) Rust
  - 无栈协程（用户态线程）调度
  - 轻量，高并发
  - 编译器保证并发安全 (safe rust + Send + Sync)

上下游生态


Logs
Transactions
Clicks
...
EVENTS
kafka
PULSAR
amazon KINESIS
PostgreSQL
MySQL
amazon S3
MESSAGE QUEUES & DATABASES
Real-time ingestion
Stream joins
Filtering
Aggregation
Transformation
Continuously updated and queryable states
RisingWave
SQL
Sink
Users
Real-time applications
Superset
Grafana
BI tools
Message queues
Databases
Warehouses
Data lakes

RisingWave 与 ClickHouse



PostgreSQL Table Engine in ClickHouse

```
:) CREATE TABLE rw_sum
(
    `sum` int
)
ENGINE = PostgreSQL('localhost:4566', 'dev', 'sum', 'root', 'root')

CREATE TABLE rw_sum
(
    `sum` int
)
ENGINE = PostgreSQL('localhost:4566', 'dev', 'sum', 'root', 'root')

Query id: d6c02b1c-a30a-4e45-baef-62a0f2f9826c

Ok.

0 rows in set. Elapsed: 0.003 sec.

:) SELECT sum FROM rw_sum;

SELECT sum
FROM rw_sum

Query id: 92429e46-cca5-4297-8f3c-aa373280cd7d

   ┌─sum─┐
1. │   6 │ space key to toggle the display of the progress table.
   └─────┘

1 row in set. Elapsed: 0.010 sec.
```

```
CREATE TABLE rw_sum
(
    `sum` int
)
ENGINE = PostgreSQL('localhost:4566', 'dev', 'sum',
'root', 'root');
```

RisingWave

简单查询

```
:) explain select sum from rw_sum;

EXPLAIN
SELECT sum
FROM rw_sum

Query id: cd460ce9-4fc8-4bb1-a685-da12e80d875e

    ┌─explain──────────────────────────────────────────────────────────────────┐
 1. │ Expression ((Project names + (Projection + Change column names to column identifiers))) │
 2. │   ReadFromPostgreSQL                                                     │
    └──────────────────────────────────────────────────────────────────────────┘

2 rows in set. Elapsed: 0.003 sec.
```

```
2025-09-19T16:50:29.954713+08:00  INFO handle_query{sql=COPY (SELECT "num" FROM "number") TO STDOUT}
```

RisingWave

与ClickHouse表关联查询

```
:) EXPLAIN SELECT t_ck.name FROM t_ck JOIN rw_number ON t_ck.id < rw_number.num WHERE rw_number.num < 2;

EXPLAIN
SELECT t_ck.name
FROM t_ck
INNER JOIN rw_number ON t_ck.id < rw_number.num
WHERE rw_number.num < 2

Query id: 3e1bac7a-6d68-4484-aa98-14b314270e3f

    ┌─explain──────────────────────────────────────────────────────────────────┐
 1. │ Expression ((Project names + (Projection + )))                           │
 2. │   Filter (Post Join Actions)                                             │
 3. │     Join                                                                 │
 4. │       Expression (Left Pre Join Actions)                                 │
 5. │         Expression (Change column names to column identifiers)           │
 6. │           ReadFromPreparedSource (Read from NullSource)                  │
 7. │       Expression (Right Pre Join Actions)                                │
 8. │         Filter ((WHERE + Change column names to column identifiers))     │
 9. │           ReadFromPostgreSQL                                             │
    └──────────────────────────────────────────────────────────────────────────┘

9 rows in set. Elapsed: 0.002 sec.
```

```
SELECT t_ck.name
FROM t_ck
   INNER JOIN
rw_number
ON t_ck.id < rw_number.num
WHERE rw_number.num < 2
```

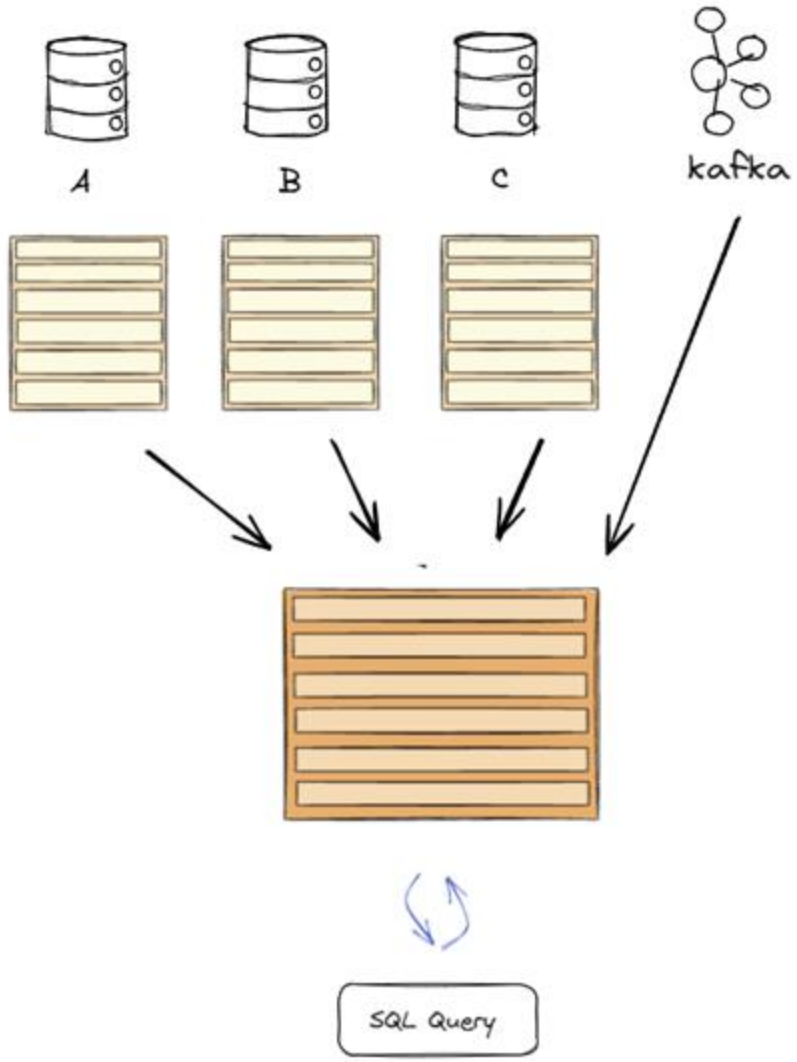
谓词下推

```
2025-09-19T17:13:37.891099+08:00 DEBUG handle_query{mode="simple query" session_id=5
sql=COPY (SELECT "num" FROM "number" WHERE "num" < 2) TO STDOUT}
```

RisingWave

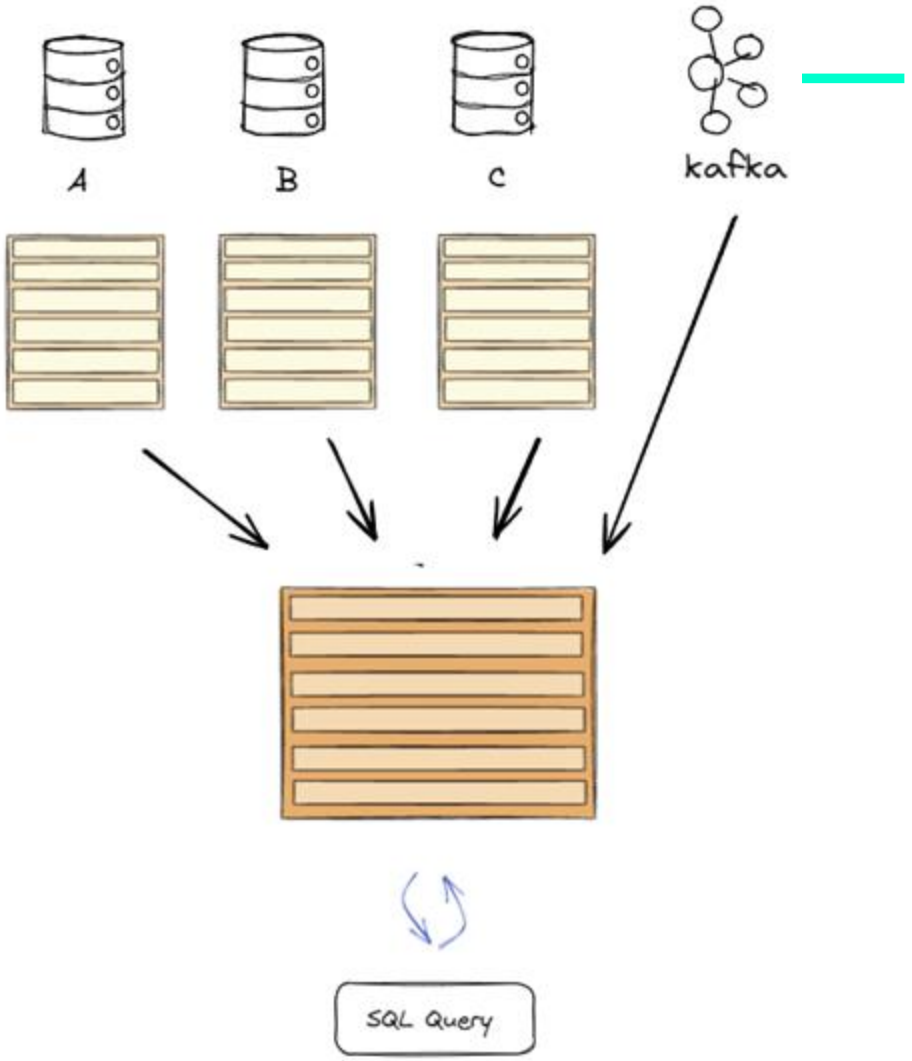
RisingWave 写入 ClickHouse 大宽表

- 大宽表
  - 包含**大量列（字段）**的数据表
  - 相关的数据都在同一行，查询分析更方便
  - 用于**交互式查询分析**



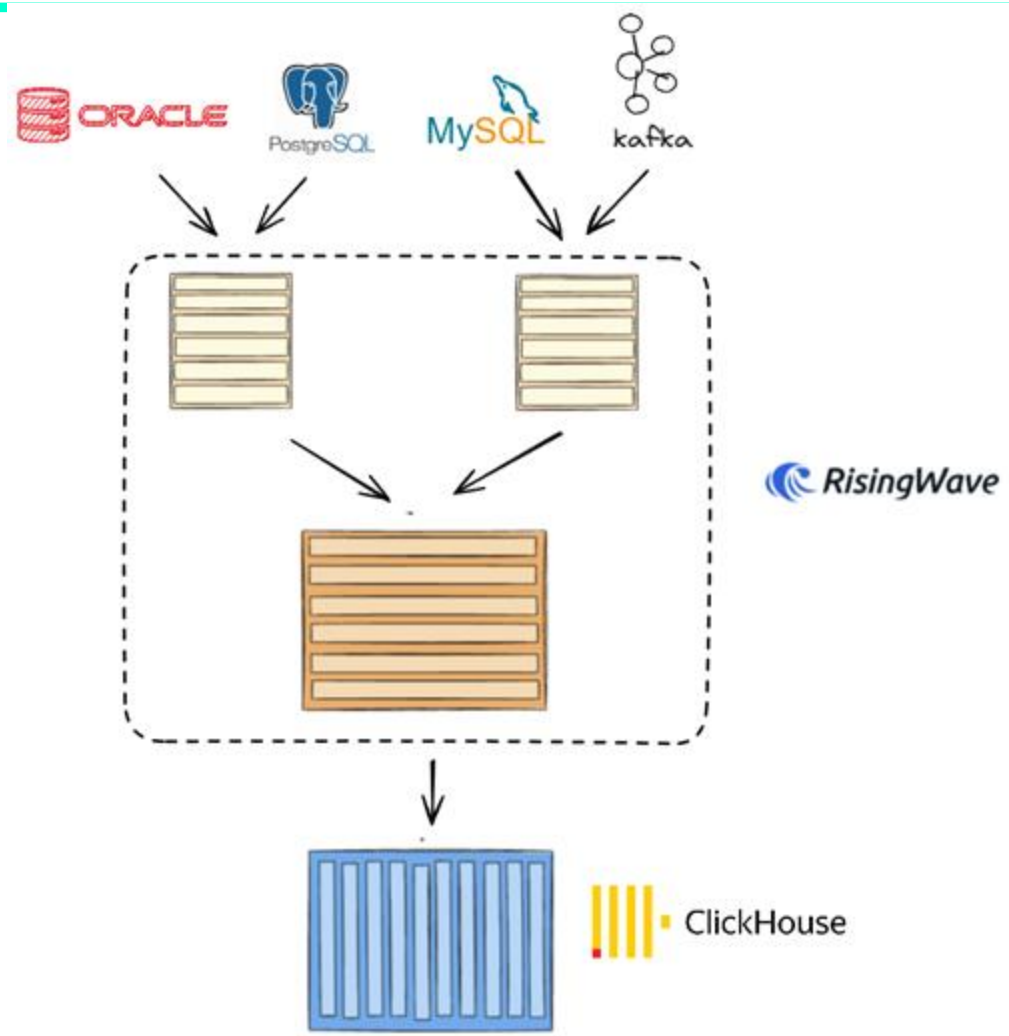
场景分析：实时大宽表

- 实时打宽
  - **又称多流拼接，将来自不同数据源的多个表互相关联**，打平成宽表
  - 打宽的**pattern较为固定**（基于ID列）
  - 打宽的**实时性要求越来越高**



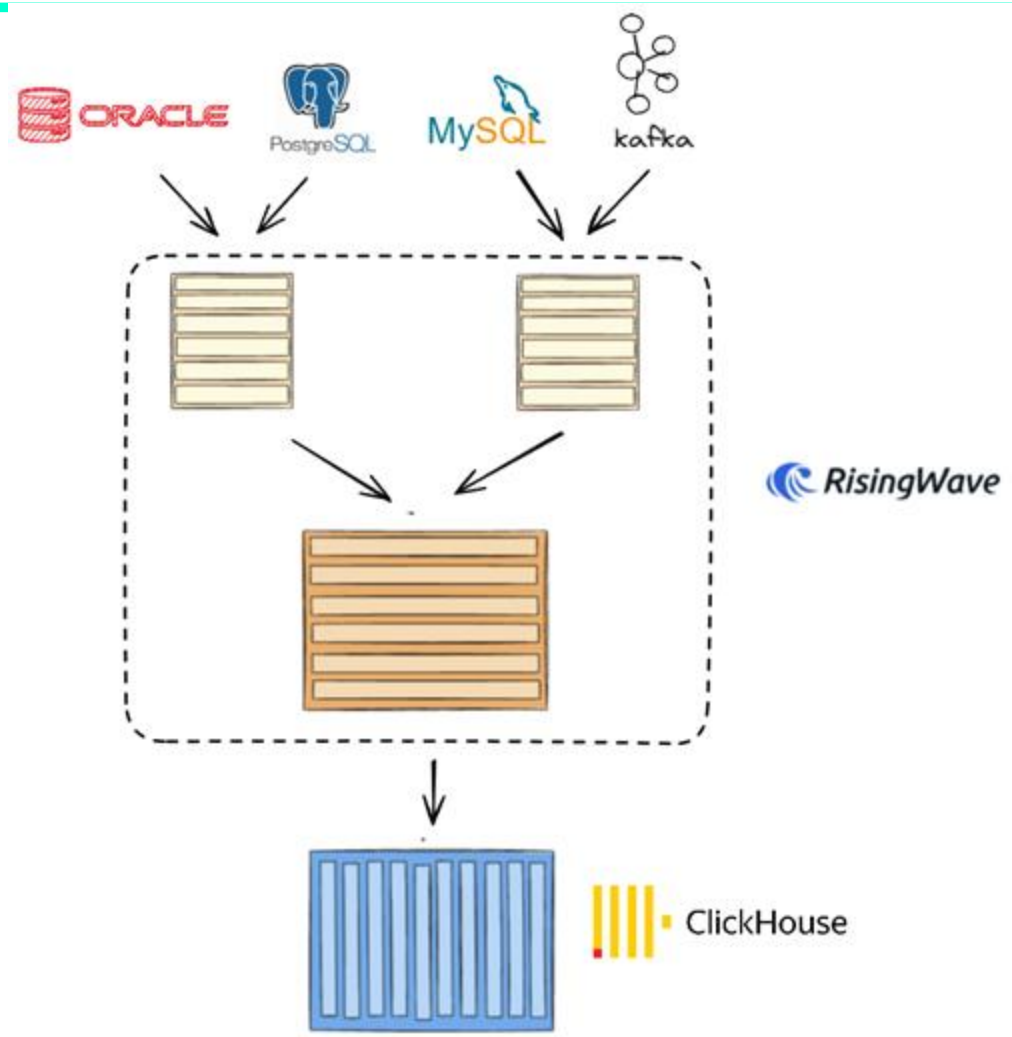
RisingWave

RisingWave x ClickHouse：实时打宽场景分析



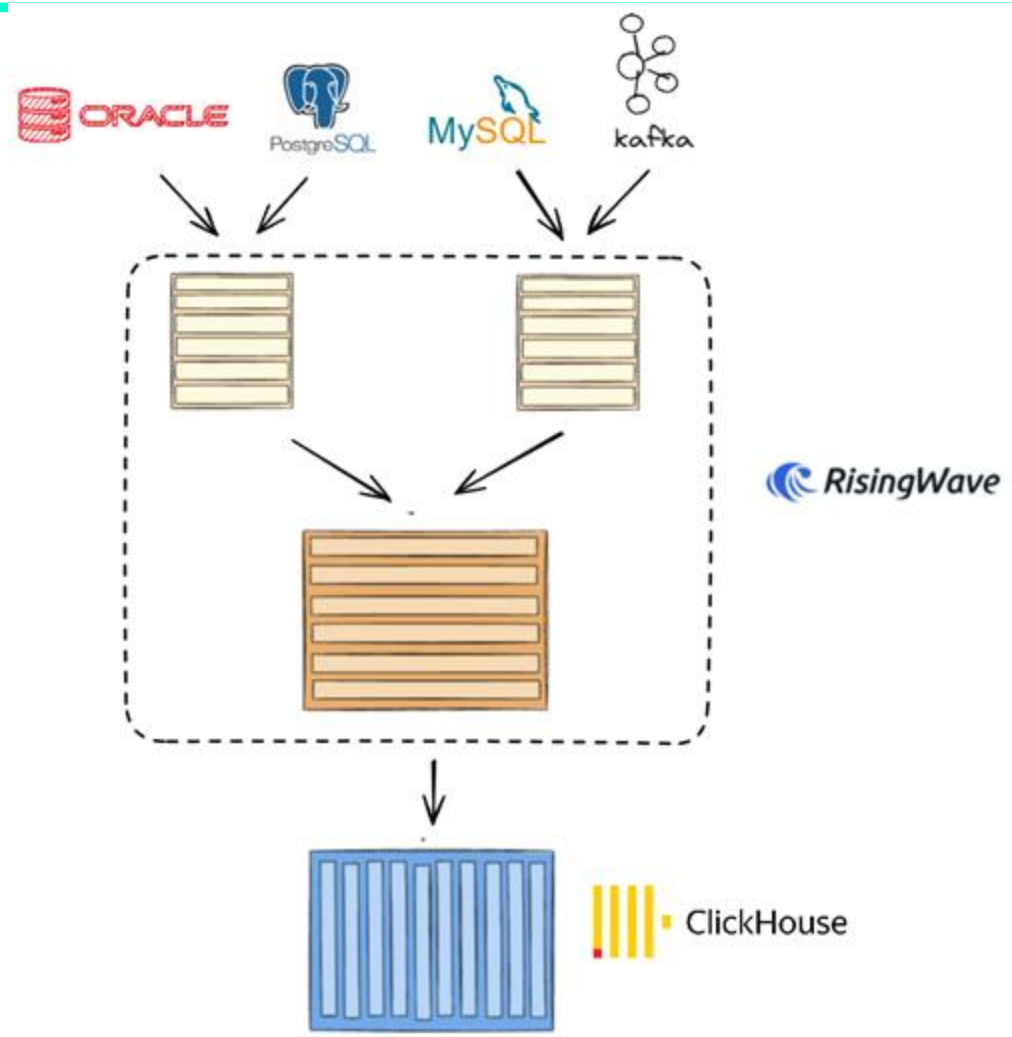
RisingWave x ClickHouse：实时打宽场景分析

- 挑战#1：数据来自不同数据源
- 挑战#2：超大状态维护
- 挑战#3：流式数仓写入



实时打宽挑战#1：数据来自不同数据源

- 来自DB的变更（CDC）：
  - MySQL
  - PostgreSQL
  - Oracle
  - TiDB、MongoDB、...
- 来自事件流（MQ）
  - Kafka
  - Pulsar
  - MQTT



RisingWave基础概念：SOURCE

- Source支持消费多种数据源：Kafka、Plusar、Kinesis、S3
- 消息Encoding支持：AVRO、JSON、PROTOBUF、CSV、BYTES
- 消息队列支持指定消费开始位置
- 支持Schema Registry

```
CREATE SOURCE ad_click (
    bid_id BIGINT,
    click_timestamp TIMESTAMPZ
) WITH (
    connector = 'kafka',
    topic = 'ad_click',
    properties.bootstrap.server = 'xxx',
    scan.startup.mode = 'earliest'
) FORMAT PLAIN ENCODE AVRO (
   schema.registry = 'xxx'
);
```

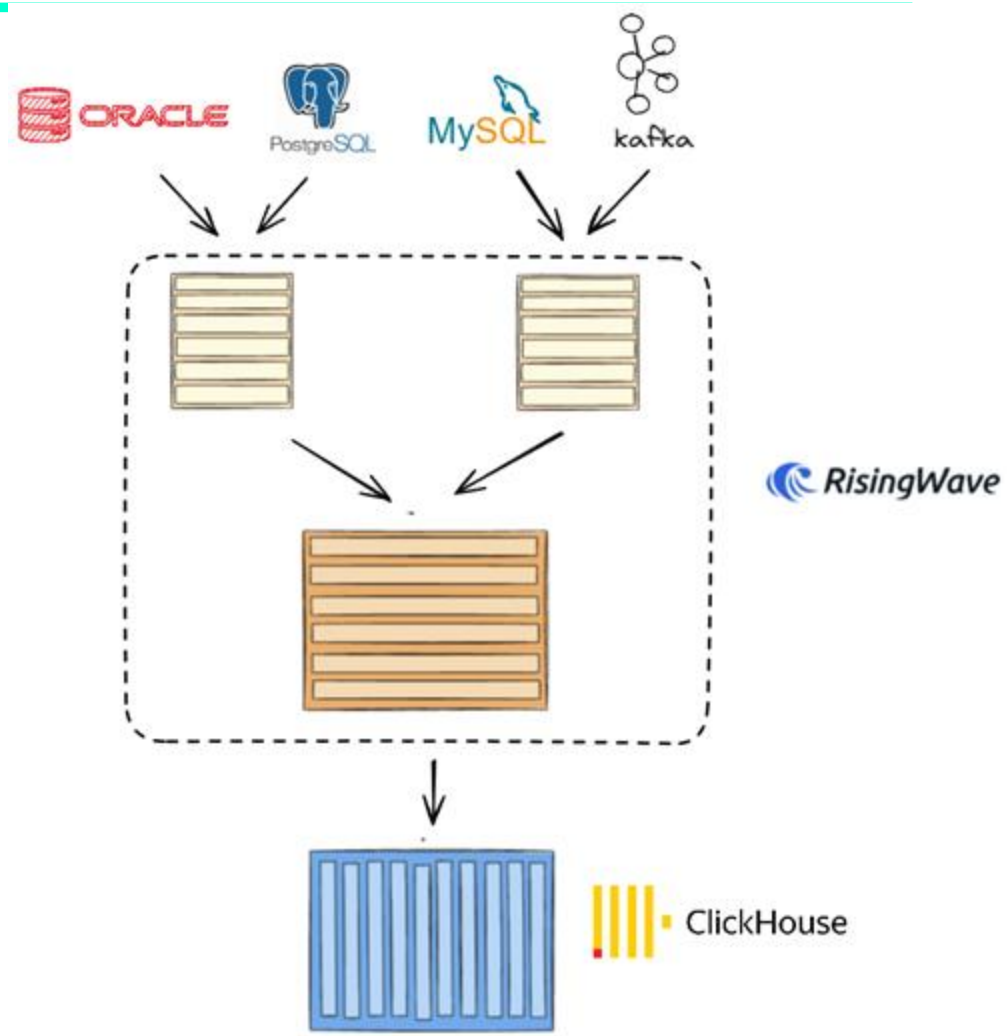
RisingWave基础概念：TABLE

- Table可以消费所有Source支持的数据源
- Table会物化数据到表，支持主键
- Table支持对接上游CDC
  - OLTP数据库（MySQL、PostgreSQL、Oracle、TiDB等）
  - NoSQL数据库（MongoDB等）
- Table支持DML，可增删改
- 消息Format支持：PLAIN、DEBEZIUM、CANAL、MAXWELL、UPSERT

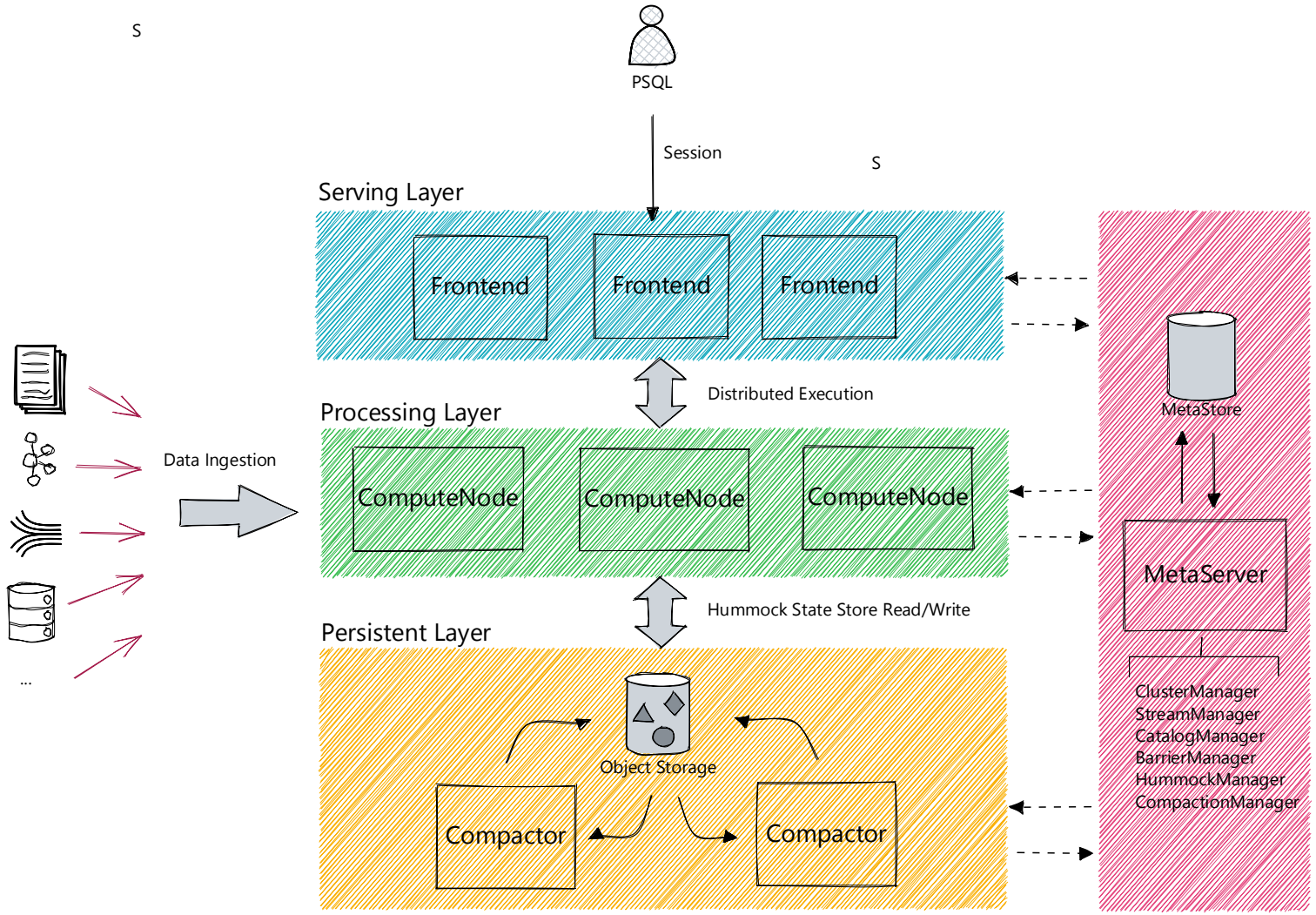
```
CREATE TABLE orders (
    order_id int,
    price decimal,
    PRIMARY KEY (order_id)
) WITH (
 connector = 'mysql-cdc',
 hostname = '127.0.0.1',
 port = '3306',
 username = 'root',
 password = 'xxx',
 database.name = 'mydb',
 table.name = 'orders',
);
```

实时打宽挑战#2：超大状态维护

- Watermark、Window Join等高级Streaming特性不一定适用
  - 宽表的更新不一定有很强的时间局域性
  - 历史数据的修复订正也需要同步到宽表
- SQL Inner/Outer Join更适用于打宽场景
  - 标准SQL（PG）可轻松构建Streaming Pipeline
  - 大状态是常态

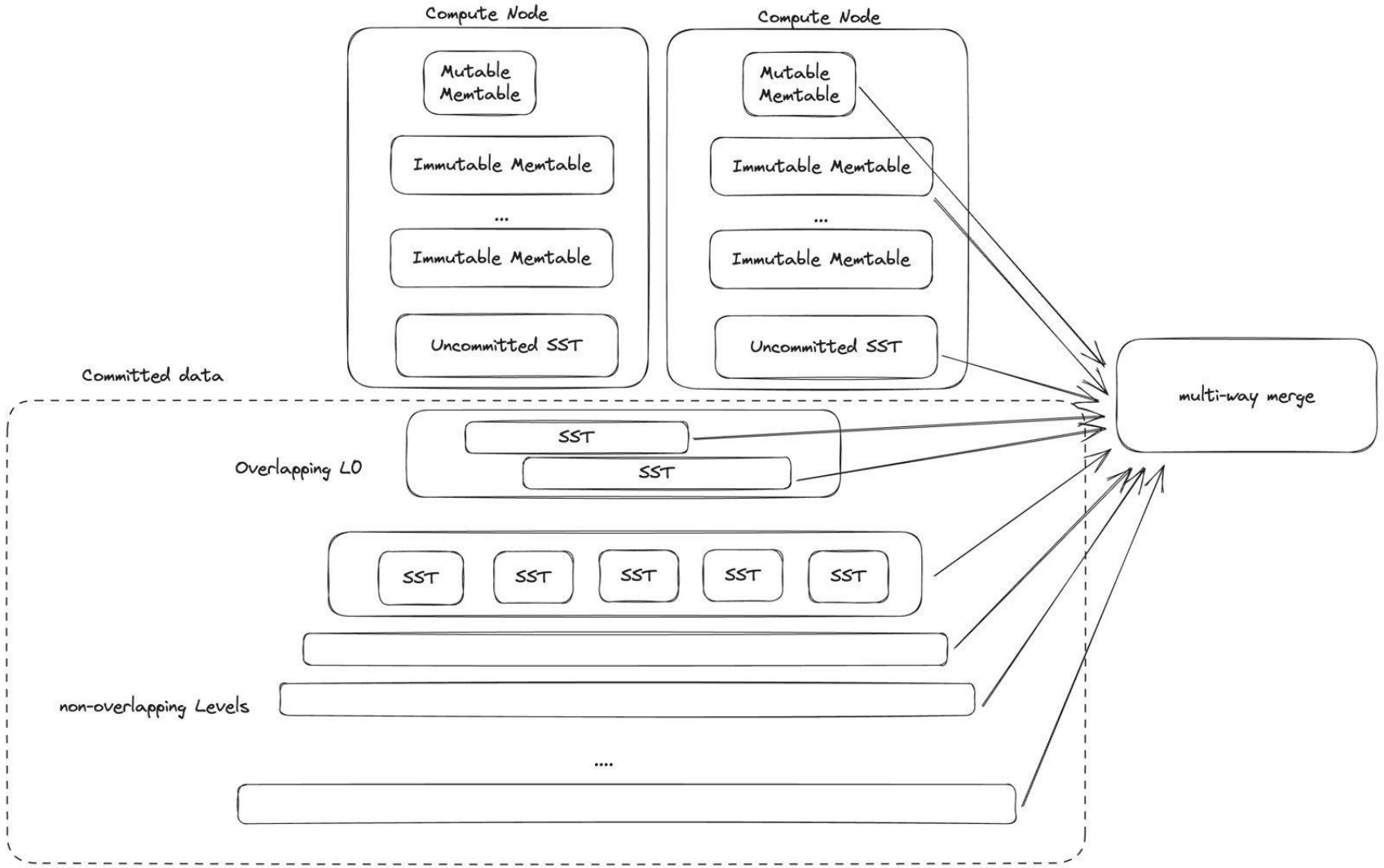


存算分离架构



RisingWave

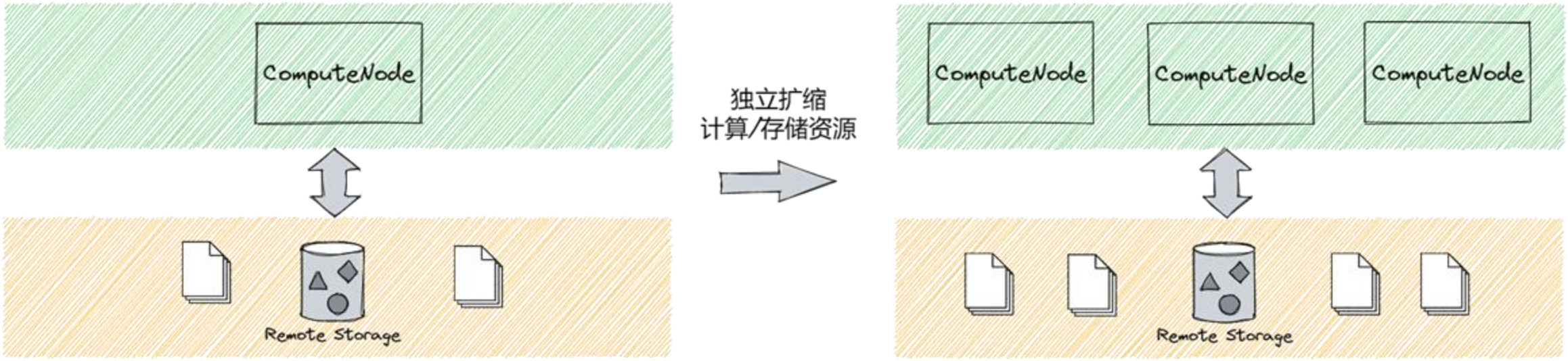
Rust从0实现LSM Tree



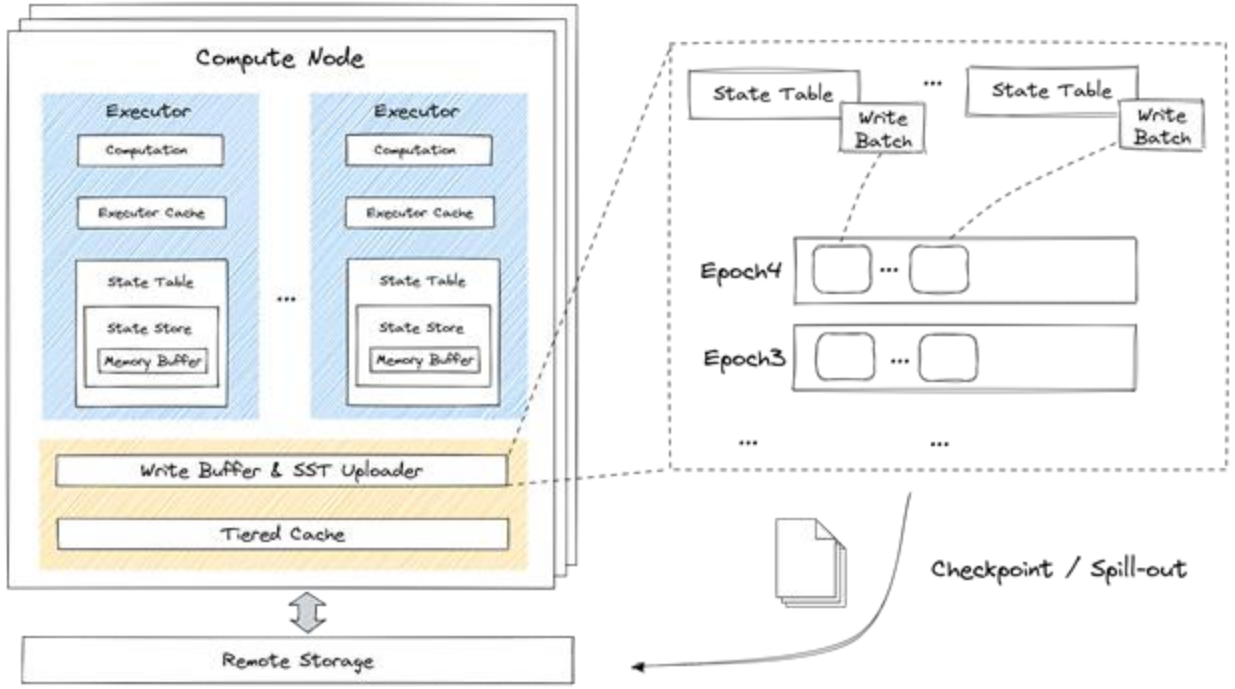
RisingWave

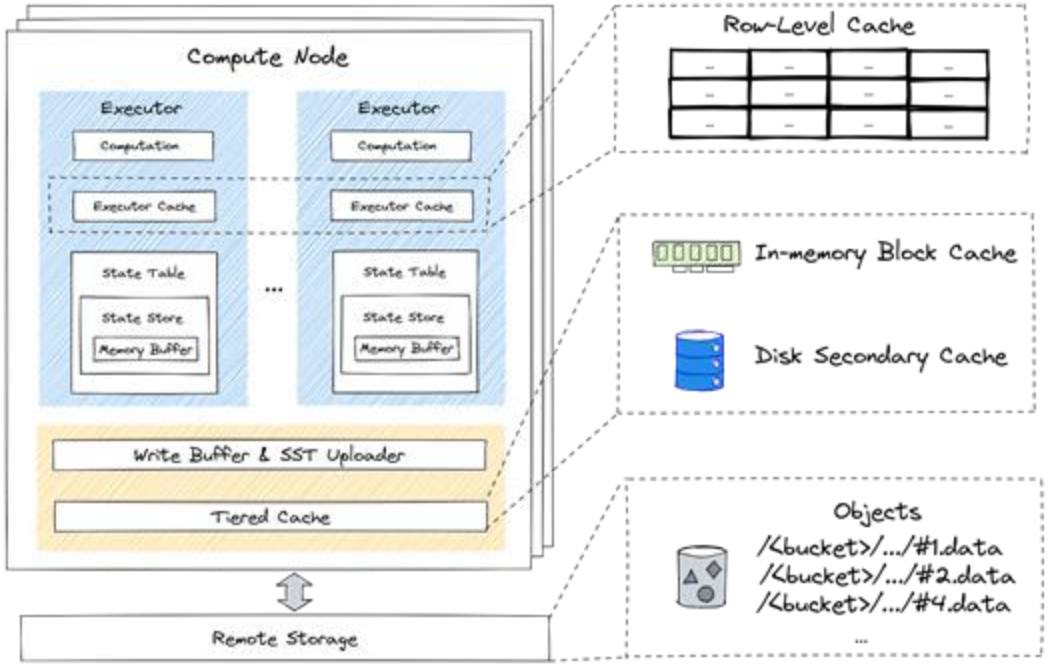
存算分离：大状态 != 高成本

- 状态存储持久化在成本低廉的远端Remote Object Store
- **存算分离的架构**让计算和存储资源独立扩缩，且实现**秒级扩缩容**，**无**须腾挪数据
- **自研云原生LSM存**储引擎，支持Serverless Compaction，**根据**compaction负载auto scale



Tiered Cache：大状态 != 低性能



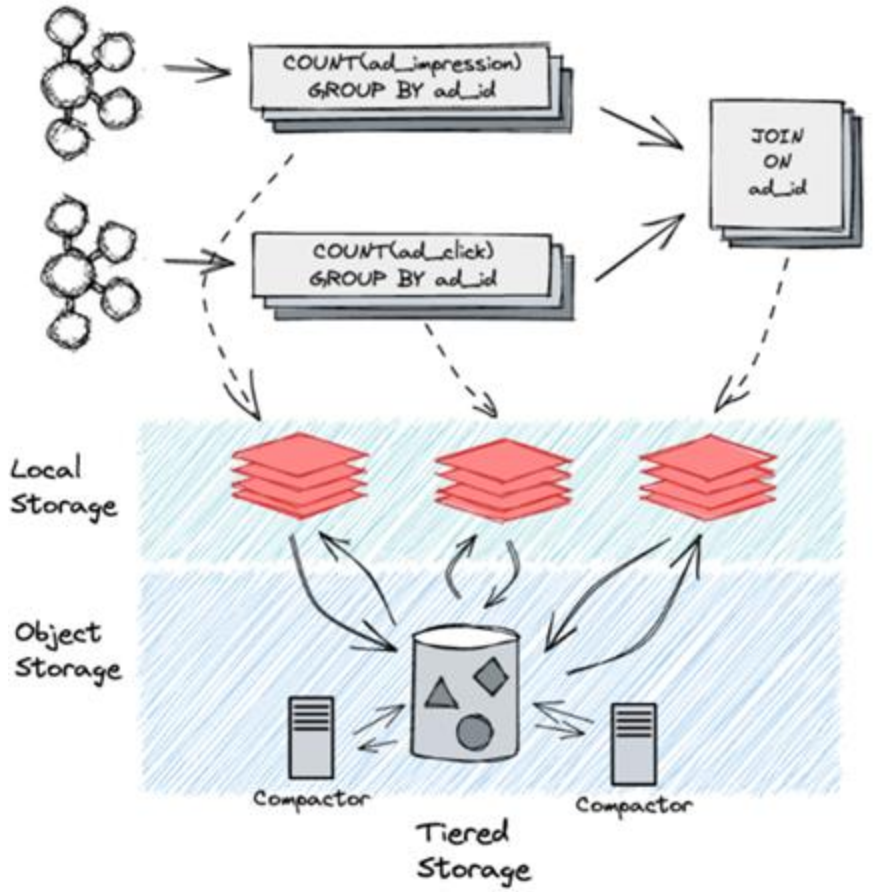


- 计算节点维护Write Buffer攒批写入远端存储
- Checkpoint触发Force Flush
- 大状态可提前Spill

- 计算节点维护Tiered Cache提升读性能
- 热数据缓存在最靠近计算的算子Row Cache
- 温数据缓存在节点的Block Cache

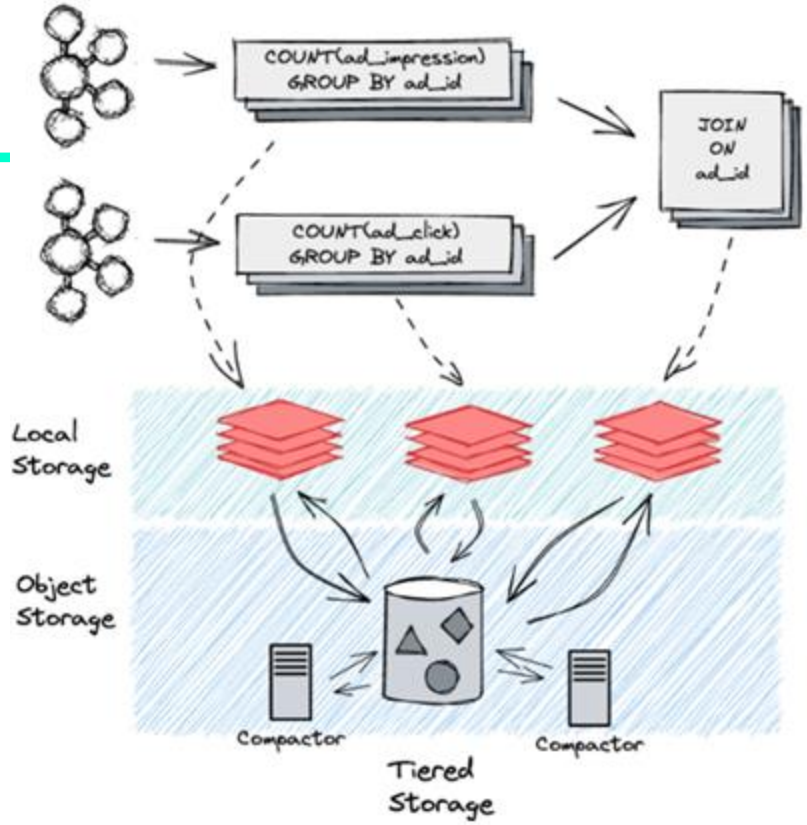
存算分离架构

- 亚秒级新鲜度(freshness)
- 秒级扩缩容
- 超低存储成本
- 超大状态管理
  - 20+ Way Streaming Join



RisingWave：多流Join

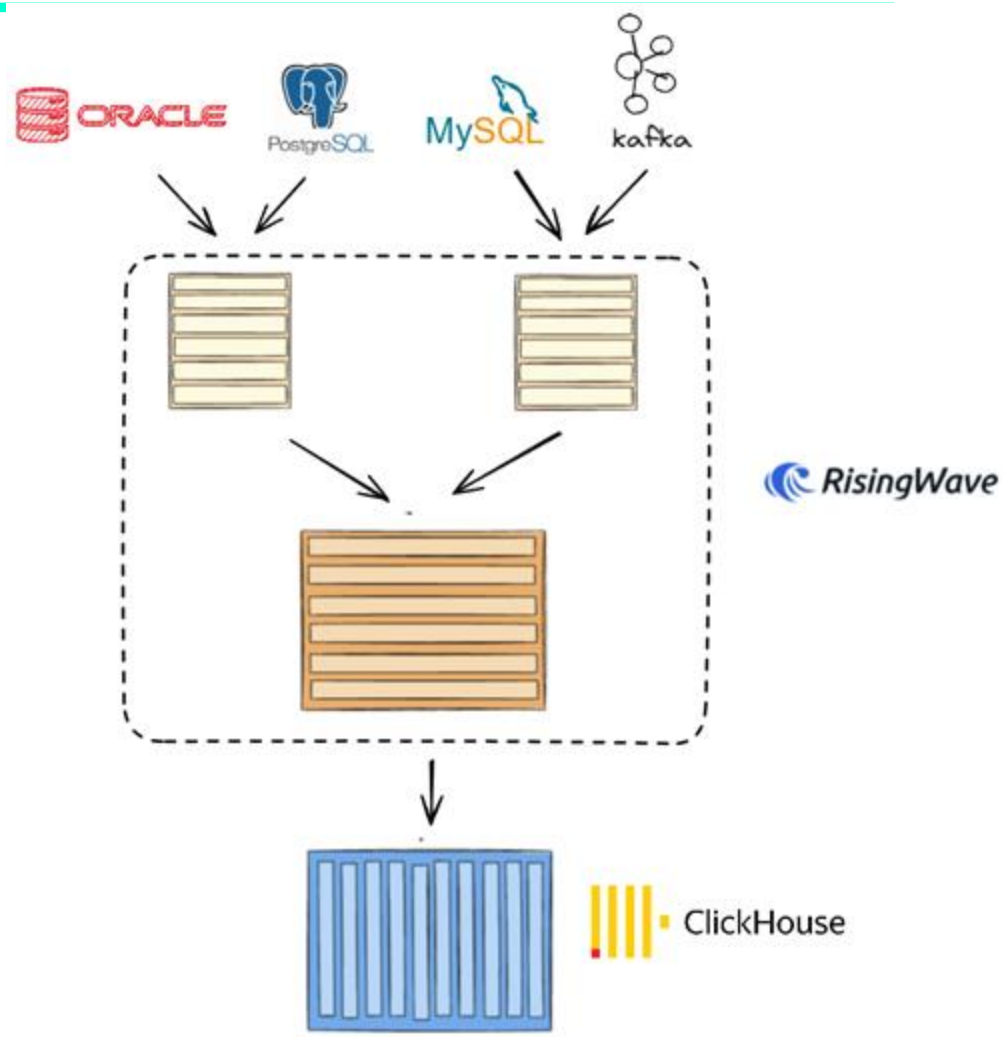
- 多流Join
  - Regular Join：Inner + Outer
  - Interval Join
  - Temporal Join
  - Windows Join (Watermark)
- TB级别长周期状态管理
  - 算子状态持久化在对象存储，无单机状态上限
  - 基于Shared Storage的自研存储引擎



- 生产案例：基于多流Join构建CDC流表打宽
  - 集群包含10+个包含多流Join的物化视图，其中最复杂的包含20路多流Join
  - join状态总量~500GB

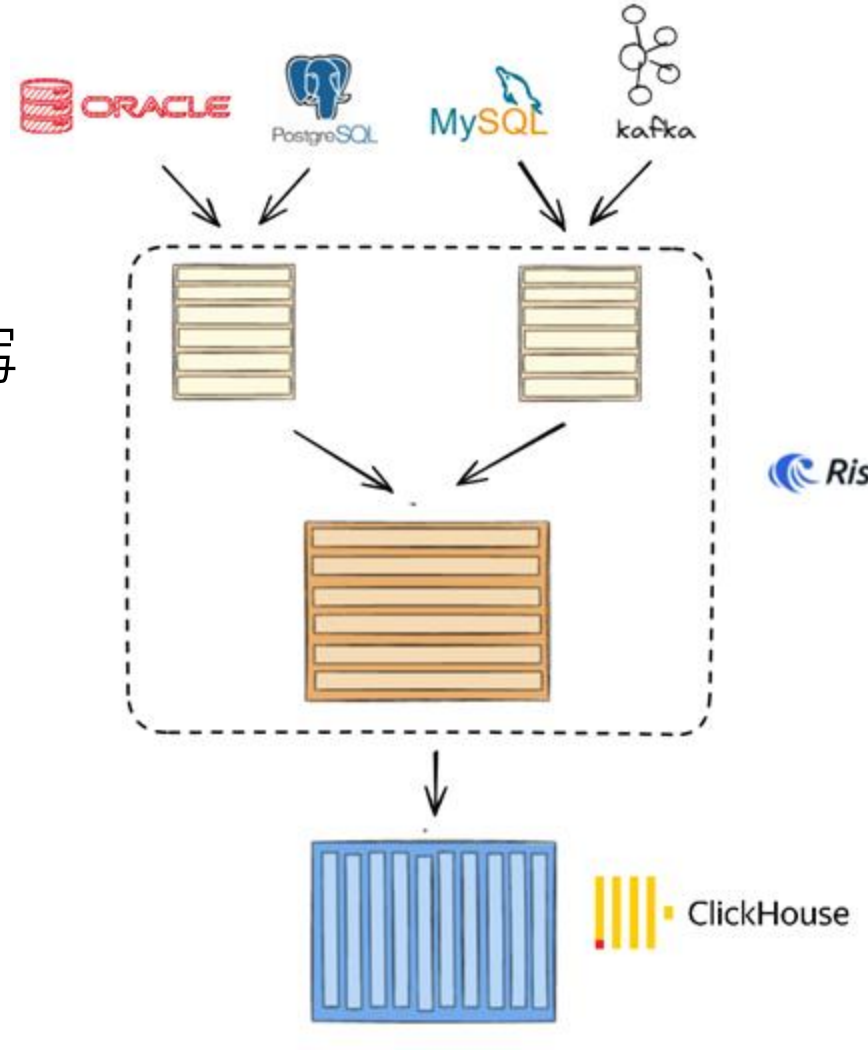
实时打宽挑战#3：流式数仓写入

过去，许多OLAP系统只能支持批量写入，无法高效支持近实时写入与更新（upsert）



实时打宽挑战#3：流式数仓写入

- ClickHouse天然支持流式写入，同时丰富的表引擎选择让流式写入更为灵活
- RisingWave通过Sink对ClickHouse进行流式写入
  - CollapsingMergeTree引擎家族与ReplacingMergeTree引擎家族支持upsert写入
  - 其余引擎家族支持append-only写入
- RisingWave支持“写解耦”(Sink Decoupling)
  - 更灵活地攒批写入下游Sink
  - 同时隔离下游Sink故障和流作业



RisingWave基础概念：SINK

- 通过Sink发送数据到多种下游系统
- 支持的Connector：Kafka、JDBC、Redis、Clickhouse、StarRocks、Doris、ElasticSearch、Cassandra、Iceberg…
- 支持的Format：APPEND_ONLY、UPSERT、DEBEZIUM
- SINK的输入可以是Table/Materialized View，也可以是SQL query

```
CREATE SINK sink_clickhouse
FROM impression_cnt
WITH (
    connector = 'clickhouse',
    type = 'upsert',
    primary_key = 'ad_id',
    clickhouse.url = '${CLICKHOUSE_URL}',
    clickhouse.user = '${CLICKHOUSE_USER}',
    clickhouse.password = '${CLICKHOUSE_PASSWORD}',
    clickhouse.database = 'default',
    clickhouse.table='impression_cnt'
);
```

ClickHouse Sink 与 clickhouse-rs

▼ **Insert a batch**

```rust
use serde::Serialize;
use clickhouse::Row;

#[derive(Row, Serialize)]
struct MyRow {
    no: u32,
    name: String,
}

let mut insert = client.insert("some")?;
insert.write(&MyRow { no: 0, name: "foo".into() }).await?;
insert.write(&MyRow { no: 1, name: "bar".into() }).await?;
insert.end().await?;
```

动态列名适配

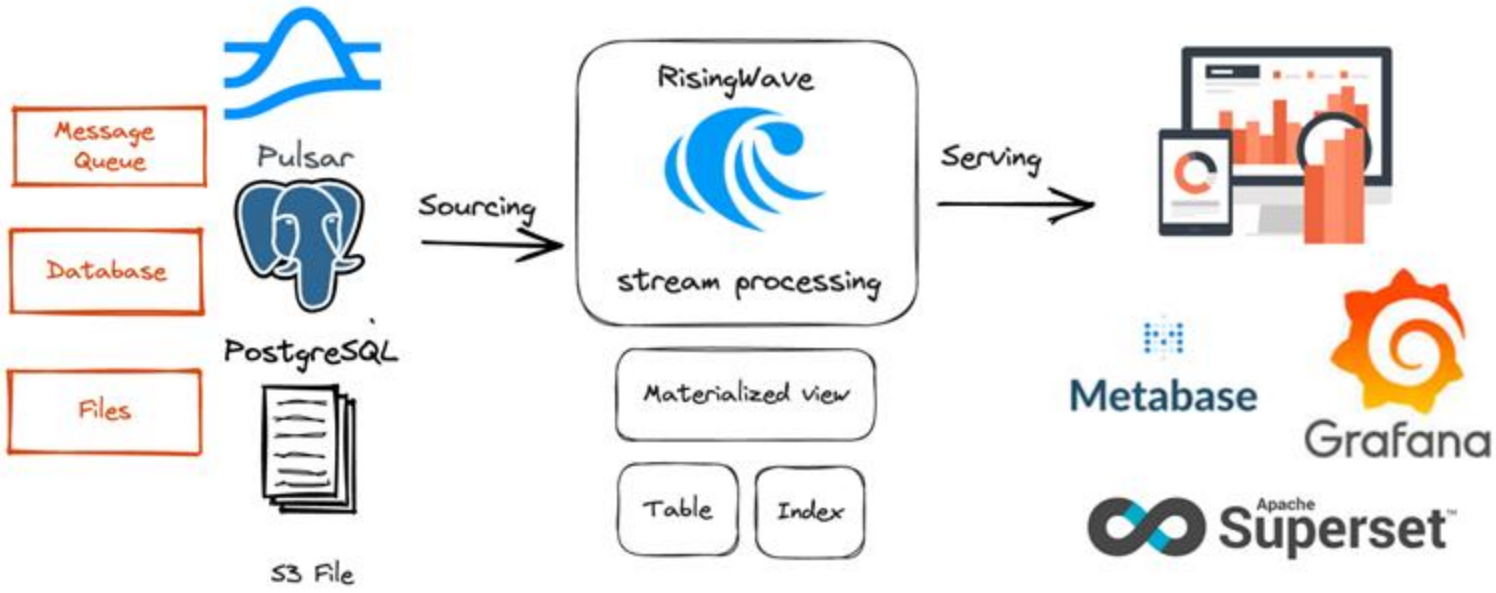
```
121    impl<T> Insert<T> {
122 -      // TODO: remove Result
123        pub(crate) fn new(client: &Client, table: &str) -> Result<Self>
124        where
125            T: Row,
126        {
127 -          let fields = row::join_column_names::<T>()
128                .expect("the row type must be a struct or a wrapper around it");
129
130            // TODO: what about escaping a table name?
131            // https://clickhouse.com/docs/en/sql-reference/syntax#identifiers
132 -          let sql = format!("INSERT INTO {}({}) FORMAT RowBinary", table, fields);
133 -
```

```
121    impl<T> Insert<T> {
122 +      pub(crate) fn new_with_field_names(
123 +          client: &Client,
124 +          table: &str,
125 +          fields_names: Vec<String>,
126 +      ) -> Result<Self> {
127 +          Insert::new_inner(client, table, fields_names.join(","))
128 +      }
129 +
130        pub(crate) fn new(client: &Client, table: &str) -> Result<Self>
131        where
132            T: Row,
133        {
134 +          let fields_names = row::join_column_names::<T>()
135                .expect("the row type must be a struct or a wrapper around it");
136 +          Insert::new_inner(client, table, fields_names)
137 +      }
138
139 +      pub(crate) fn new_inner(client: &Client, table: &str, fields_names: String) ->
    Result<Self> {
140            // TODO: what about escaping a table name?
141            // https://clickhouse.com/docs/en/sql-reference/syntax#identifiers
142 +          let sql = format!("INSERT INTO {}({}) FORMAT RowBinary", table, fields_names);
```

RisingWave

基于宽表做进一步实时分析：实时看板

- RisingWave Table/Materialized View结果支持SQL查询
- **支持Serving和**简单批查询、支持Streaming**和Serving隔离**
- **用**户可基于宽表构建下游MV，轻松对接Metabase/Superset**等BI工具构建**实时看板



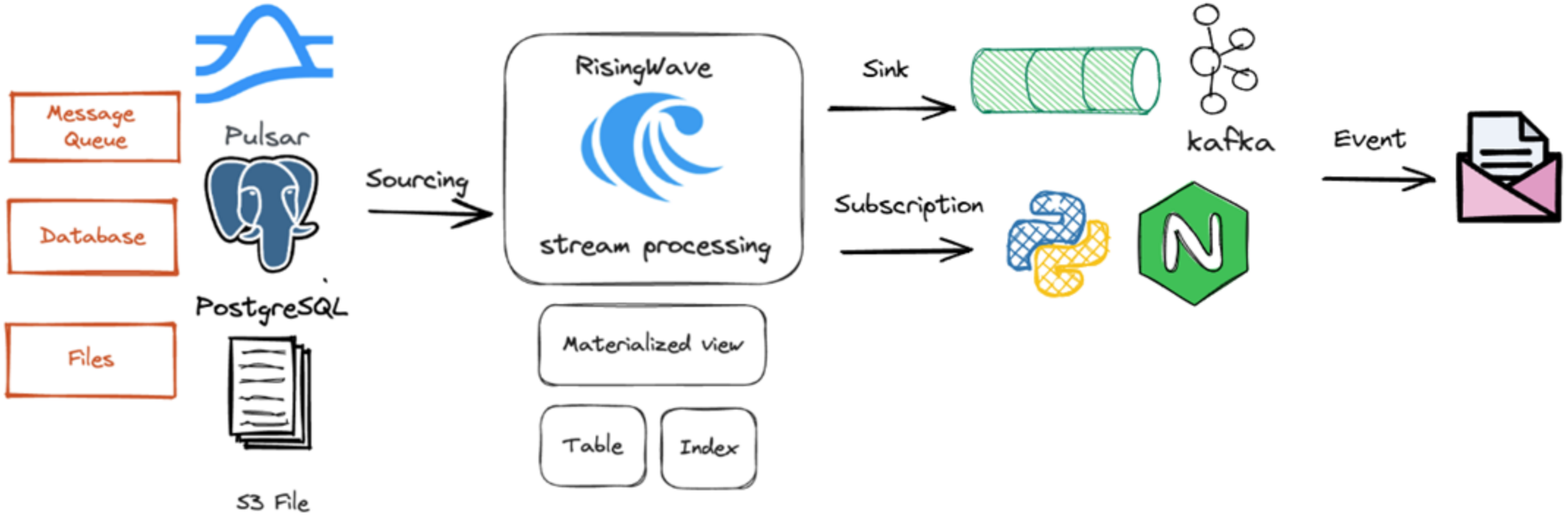
RisingWave基础概念：INDEX

- 索引可以创建在Table和Materialized View上
- 加速Serving查询
- 支持指定Include列、Distributed列
- 支持表达式索引
- 自动索引选择

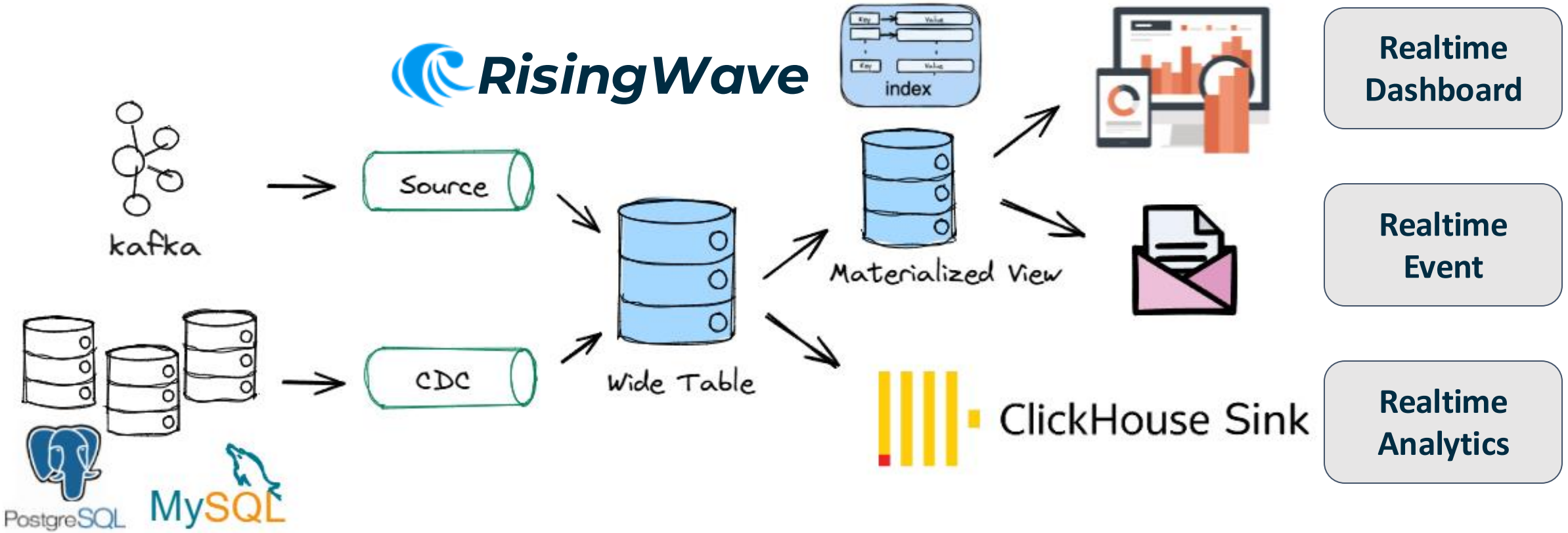
```
-- customers 加速点查
CREATE INDEX idx_c_phone on customers(c_phone);

SELECT * FROM customers where c_phone = '123456789';

-- orders 加速join
CREATE INDEX idx_o_custkey on orders(o_custkey);

SELECT * FROM customers JOIN orders ON c_custkey = o_custkey;

-- json 表达式加速
CREATE TABLE t (j jsonb, v1 int, v2 int);
CREATE INDEX idx1 on t(j->>'k1');

SELECT * FROM t WHERE j->>'k1' = 'abc';
```

基于宽表做进一步实时分析：Event Notification

- RisingWave MV变更支持Sink到下游MQ
- RisingWave MV变更支持通过Subscription进行订阅
- 用户可基于宽表构建下游MV，通过MQ/Subscription订阅MV变更，构建事件驱动的业务与应用。如规则引擎、异常告警、指标监控等。



RisingWave x ClickHouse



START YOUR STREAM PROCESSING JOURNEY WITH

A SINGLE LINE OF CODE!

Install RisingWave for macOS and Linux

```
curl https://risingwave.com/sh | sh
```

关注 RisingWave 订阅号
获取更多

扫一扫进入
社区用户微信交流群

GitHub:	**官网:**	**Slack:**	**B站:**	**知乎:**
risingwave.com/github	risingwave.com	risingwave.com/slack	RisingWave 中文开源社区	RisingWave 中文开源社区

RisingWave

Thank You!

Join Our Slack: https://go.risingwave.com/slack
Github: https://github.com/risingwavelabs/risingwave

RisingWave