



CDC Stream Processing with Apache Flink[®]

A peek under the hood of a changelog engine

Timo Walther, Principal Software Engineer

FOSDEM 2023, 2023-02-04

About me



Open source

- Long-term committer since 2014 (before ASF)
- Member of the project management committee (PMC)
- Top 5 contributor (commits), top 1 contributor (additions)
- Among core architects of Flink SQL

Career

- Early Software Engineer @ DataArtisans (acquired by Alibaba)
- SDK Team, SQL Team Lead @ Ververica
- Co-Founder @ Immerok (acquired by Confluent)
- Principal Software Engineer @ Confluent





CONFLUENT

What is Apache Flink?

Building Blocks for Stream Processing



Streams

- Pipeline
- Distribute
- Join
- Enrich
- Control
- Replay



Time

- Synchronize
- Progress
- Wait
- Timeout
- Fast-forward
- Replay



State

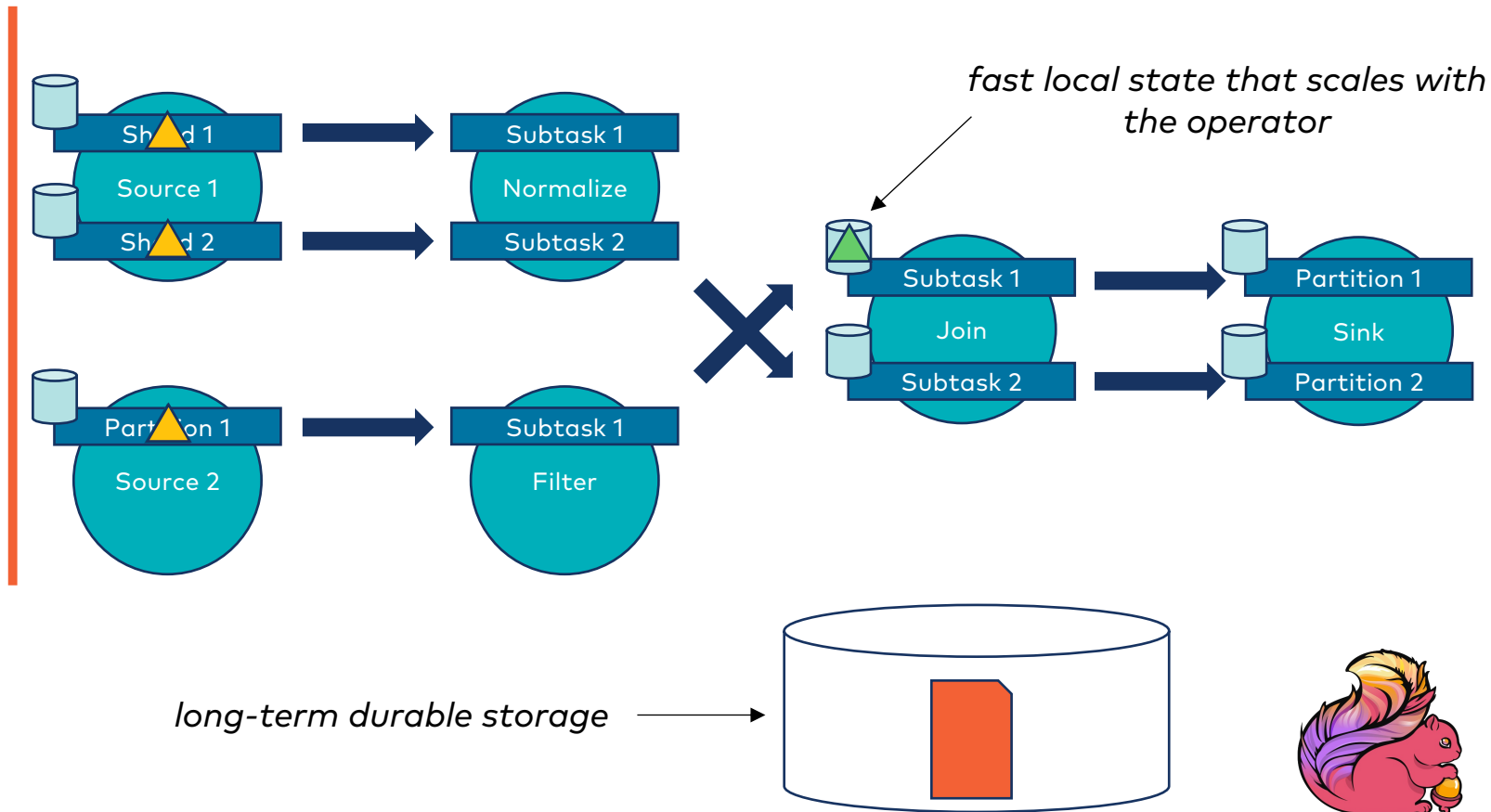
- Store
- Buffer
- Cache
- Model
- Grow
- Expire



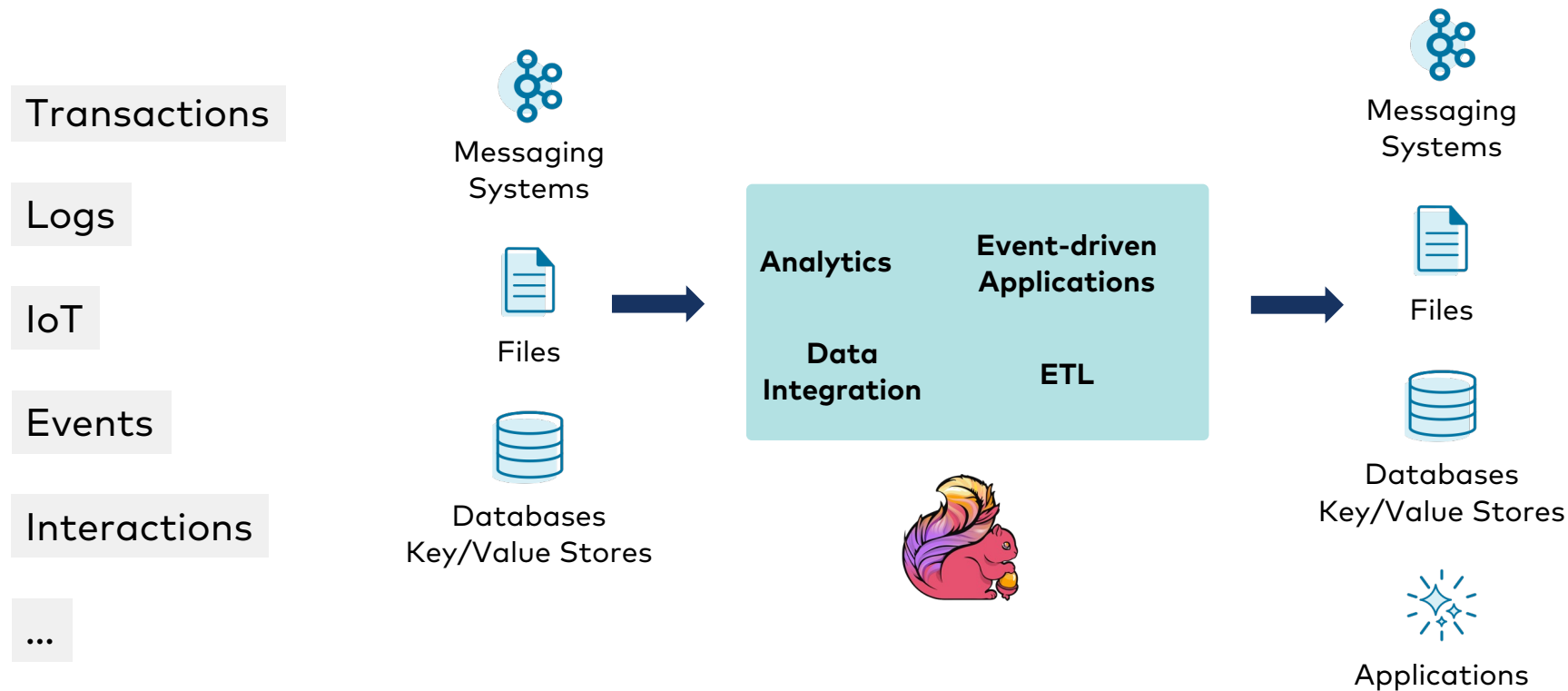
Snapshots

- Backup
- Version
- Fork
- A/B test
- Time-travel
- Restore

What makes Apache Flink unique?



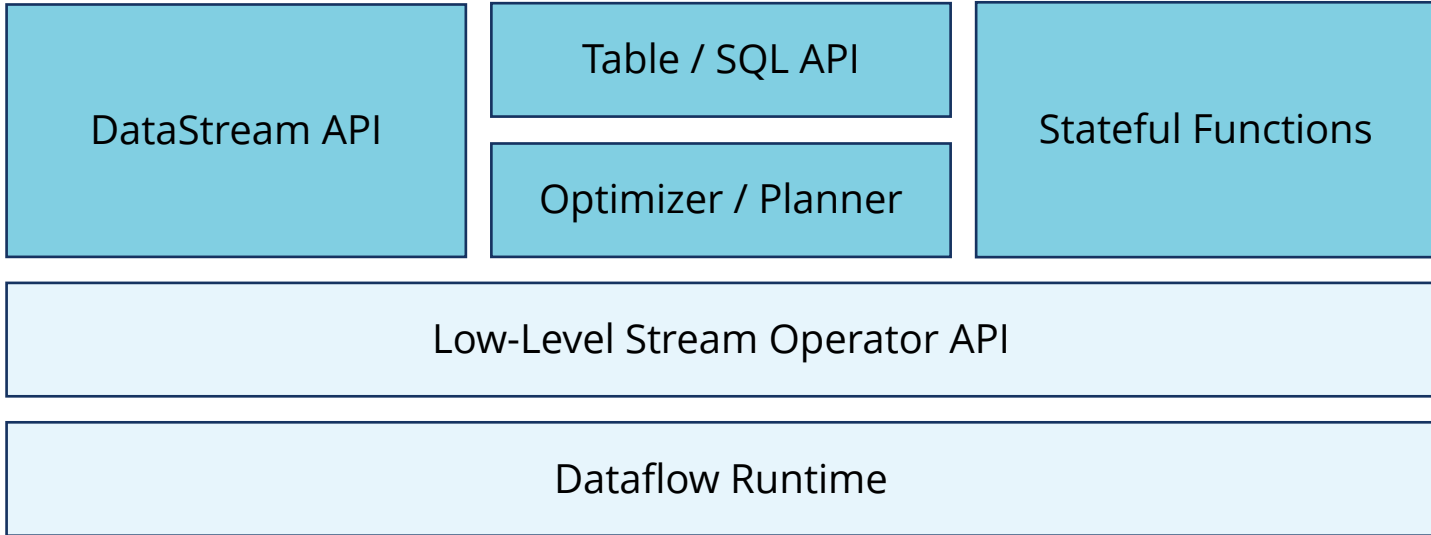
What is Apache Flink used for?





Apache Flink's APIs

API Stack



DataStream API



```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
env.setRuntimeMode(STREAMING);  
  
DataStream<Integer> stream = env.fromElements(1, 2, 3);  
  
stream.executeAndCollect().forEachRemaining(System.out::println);
```

Output

1
2
3

Properties

- **Exposes** the building blocks for stream processing
- Arbitrary operator **topologies** using `map()`, `process()`, `connect()`, ...
- Business logic is written in **user-defined functions**
- Arbitrary **user-defined record** types flow in-between
- Conceptually always an **append-only / insert-only** log!

Table / SQL API



```
TableEnvironment env = TableEnvironment.create(EnvironmentSettings.inStreamingMode());
```

```
// Programmatic
```

```
Table table = env.fromValues(row(1), row(2), row(3));
```

```
// SQL
```

```
Table table = env.sqlQuery("SELECT * FROM (VALUES (1), (2), (3))");
```

```
table.execute().print();
```

Output

op	f0
+I	1
+I	2
+I	3

Properties

- **Abstracts** the building blocks for stream processing
- Operator topology is determined by **planner**
- Business logic is **declared** in SQL and/or Table API
- Internal record types flow, **Flink's Row type** is exposed in Table API
- Conceptually a table, but a **changelog** under the hood!

DataStream API ↔ Table / SQL API



Mix and match APIs!

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```

```
StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env);
```

```
// Stream -> Table
```

```
DataStream<?> inStream1 = ...
```

```
Table appendOnlyTable = tableEnv.fromDataStream(inStream1)
```

```
DataStream<Row> inStream2 = ...
```

```
Table anyTable = tableEnv.fromChangelogStream(inStream2)
```

```
// Table -> Stream
```

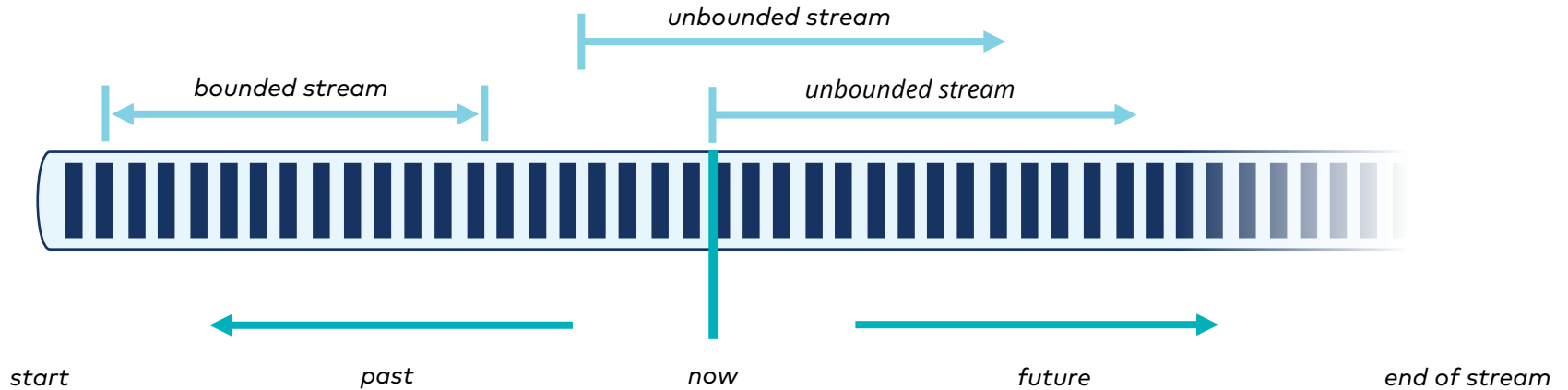
```
DataStream<T> appendOnlyStream = tableEnv.toDataStream(insertOnlyTable, T.class)
```

```
DataStream<Row> changelogStream = tableEnv.toChangelogStream(anyTable)
```



Changelog Stream Processing

Data Processing is a Stream of Changes

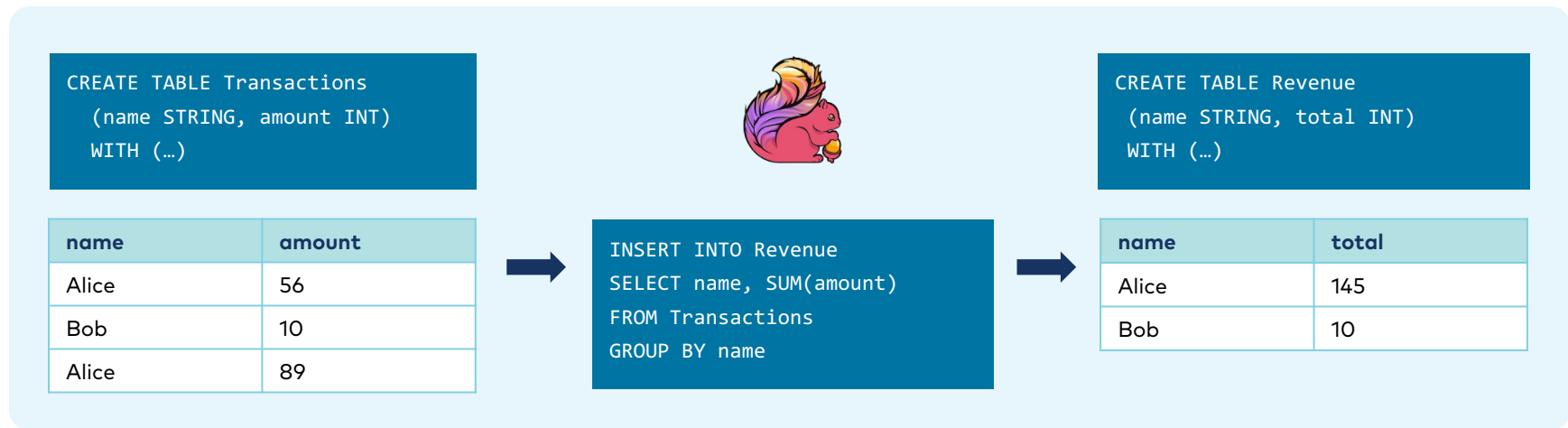


- Business data is always a stream: **bounded** or **unbounded**
- Every record is a changelog entry: **insertion as the default**
- Batch processing is just a **special case** in the runtime

How do I Work with Streams in Flink SQL?



- You don't. You work with **dynamic tables**!
- A concept similar to **materialized views**



So, is Flink SQL a database?

No, bring your own data and systems!

Stream-Table Duality - Basics



- A stream is the changelog of a **dynamic table**
- Sources, operators, and sinks work on **changelogs under the hood**

Short name	Long name	Semantics
+I	Insertion	Default for scans + output of bounded results.
-U	Update Before	Retracts a previously emitted result.
+U	Update After	Updates a previously emitted result. Requires a primary key if -U is omitted for idempotent updates.
-D	Delete	Removes the last result.

- Each component declares the **kind of changes it consumes/produces**

only **+I** Appending/Insert-only

contains **-...** Updating

contains **-U** Retracting

never **-U** but **+U** Upserting

Stream-Table Duality - Example



An applied changelog becomes a real (materialized) table.



```
CREATE TABLE Transactions
(name STRING, amount INT)
WITH (...)
```

name	amount
Alice	56
Bob	10
Alice	89

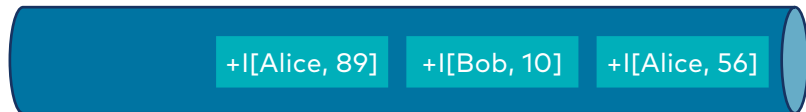
```
CREATE TABLE Revenue
(name STRING, total INT)
WITH (...)
```

name	total
Alice	145
Bob	10

```
INSERT INTO Revenue
SELECT name, SUM(amount)
FROM Transactions
GROUP BY name
```



materialization



changelog

Stream-Table Duality - Example



An applied changelog becomes a real (materialized) table.



```
CREATE TABLE Transactions
(name STRING, amount INT)
WITH (...)
```

name	amount
Alice	56
Bob	10
Alice	89

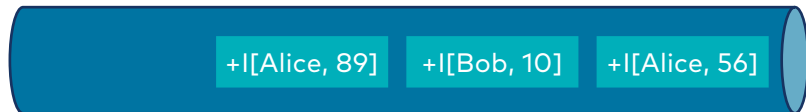
```
CREATE TABLE Revenue
(PRIMARY KEY(name) ...)
WITH (...)
```

name	total
Alice	145
Bob	10

```
INSERT INTO Revenue
SELECT name, SUM(amount)
FROM Transactions
GROUP BY name
```



materialization



Save ~50% of traffic if downstream system supports upserting!

Stream-Table Duality - Propagation



- Source declares set of emitted changes i.e. **changelog mode**

CREATE TABLE ... *(for sources)*

... WITH ('connector'='filesystem') **+I**

... WITH ('connector'='kafka') **+I**

... WITH ('connector'='kafka-uptsert') **+I** **-D**

... WITH ('connector'='jdbc') **+I**

... WITH ('connector'='kafka', 'format' = 'debezium-json') **+I** **-U** **+U** **-D**

- **Optimizer tracks** changelog mode and primary key through pipeline
- Sink declares changes it **can digest**

Retract vs. Upsert



Retract

- No primary key requirements
- Works for almost every external system
- Supports duplicate rows
- In distributed system often unavoidable

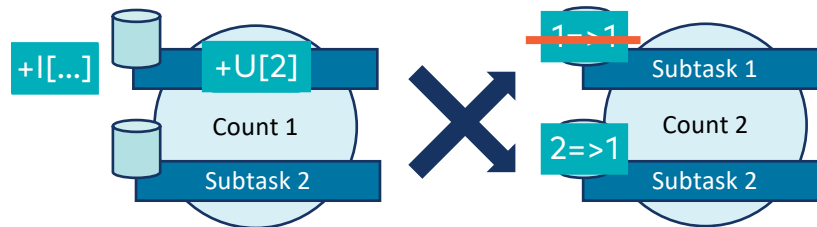
→ **most flexible changelog mode**

→ **default mode**

Upsert

- Traffic + computation **optimization**
- In-place updates (**idempotency**)

```
SELECT c, COUNT(*) FROM (  
  SELECT COUNT(*) AS c  
  FROM T  
  GROUP BY user  
)  
GROUP BY c
```



Changelog Insights – Append-only



```
CREATE TABLE Transaction (tid BIGINT, amount INT);
CREATE TABLE Payment (tid BIGINT, method STRING);
CREATE TABLE Result (tid BIGINT, ...); // accepts all changes
INSERT INTO Result SELECT * FROM Transactions T JOIN Payments P ON T.tid = P.tid;
```

```
Sink(table=[Result], changelogMode=[NONE])
+- Join(leftInputSpec=[NoUniqueKey], rightInputSpec=[NoUniqueKey], changelogMode=[I])
  :- Exchange(changelogMode=[I])
    :- TableSourceScan(table=[[Transaction]], changelogMode=[I])
  +- Exchange(changelogMode=[I])
    +- TableSourceScan(table=[[Payment]], changelogMode=[I])
```

Changelog Insights – Updating



```
CREATE TABLE Transaction (tid BIGINT, amount INT);
CREATE TABLE Payment (tid BIGINT, method STRING);
CREATE TABLE Result (tid BIGINT, ...);
INSERT INTO Result SELECT * FROM Transactions T LEFT JOIN Payments P ON T.tid = P.tid;
```

```
Sink(table=[Result], changelogMode=[NONE])
+- Join(leftInputSpec=[NoUniqueKey], rightInputSpec=[NoUniqueKey], changelogMode=[I,UB,UA,D])
  :- Exchange(changelogMode=[I])
  : +- TableSourceScan(table=[[Transaction]], changelogMode=[I])
+- Exchange(changelogMode=[I])
  +- TableSourceScan(table=[[Payment]], changelogMode=[I])
```

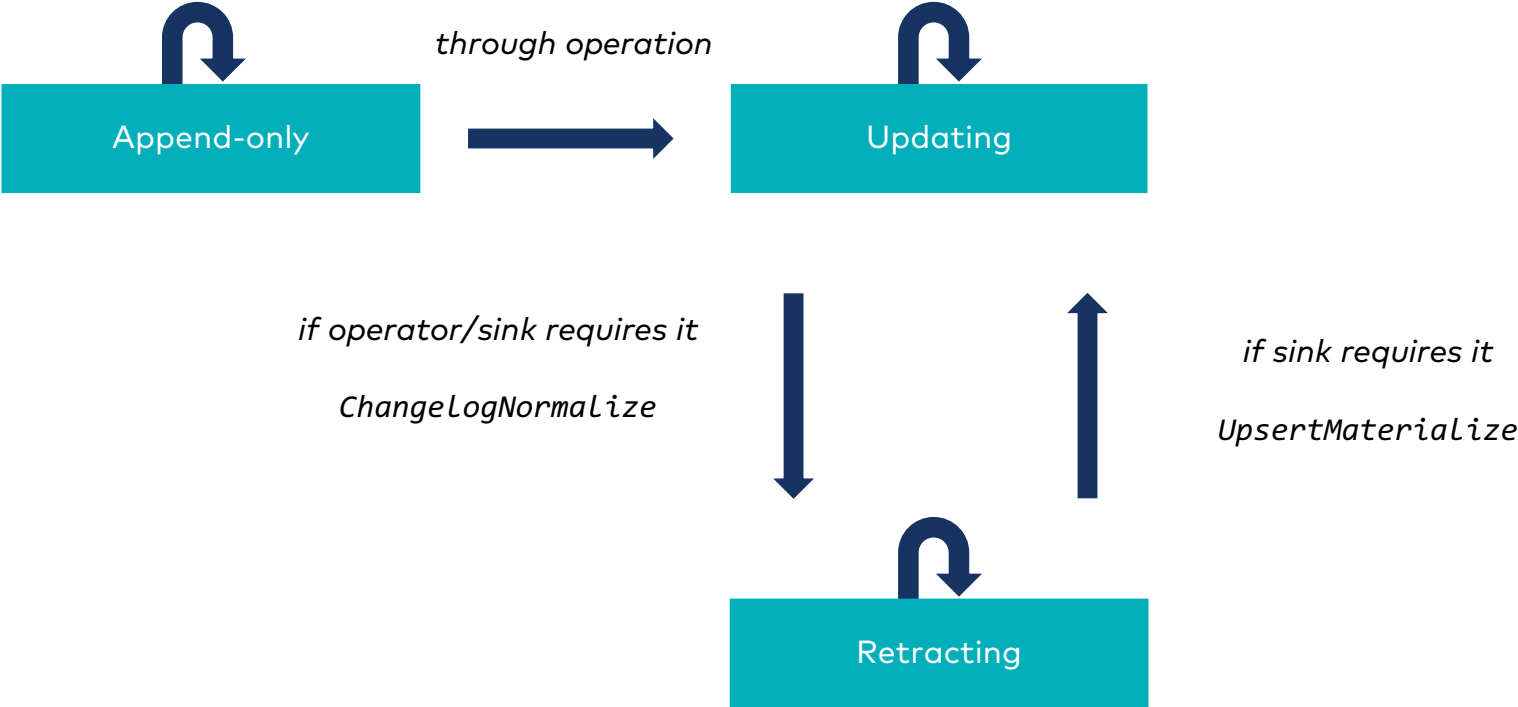
Changelog Insights – Updating with PK



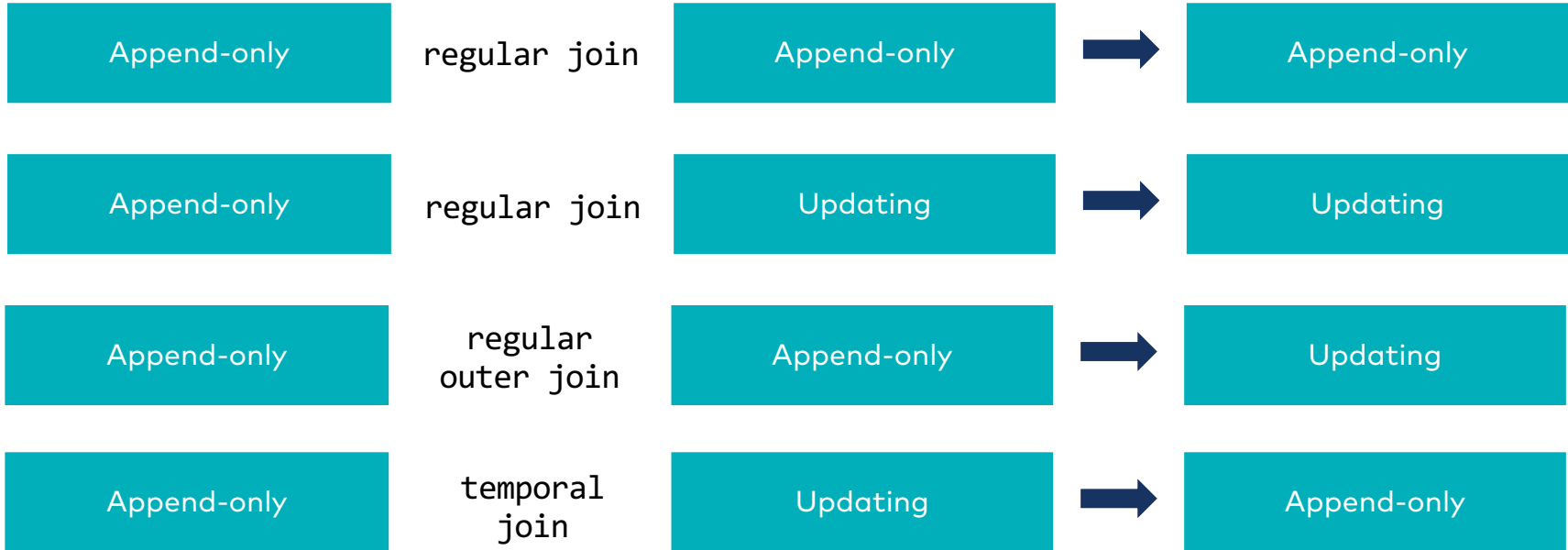
```
CREATE TABLE Transaction (tid BIGINT, ..., PRIMARY KEY(tid) NOT ENFORCED);
CREATE TABLE Payment (tid BIGINT, ..., PRIMARY KEY(tid) NOT ENFORCED);
CREATE TABLE Result (tid BIGINT, ..., PRIMARY KEY(tid) NOT ENFORCED);
INSERT INTO Result SELECT * FROM Transactions T LEFT JOIN Payments P ON T.tid = P.tid;
```

```
Sink(table=[Result], changelogMode=[NONE])
+- Join(leftInputSpec=[UniqueKey], rightInputSpec=[UniqueKey], changelogMode=[I,UA,D])
  :- Exchange(changelogMode=[I])
  : +- TableSourceScan(table=[[Transaction]], changelogMode=[I])
+- Exchange(changelogMode=[I])
  +- TableSourceScan(table=[[Payment]], changelogMode=[I])
```

Mode Transitions



Mode Transitions – Joins



Mode Transitions – Temporal Join



```
CREATE TABLE CurrencyRates (  
    WATERMARK FOR update_time AS ..., PRIMARY KEY(currency) NOT ENFORCED,...);  
  
SELECT  
    order_id,  
    price,  
    currency,  
    conversion_rate,  
    order_time  
FROM Orders  
LEFT JOIN CurrencyRates FOR SYSTEM_TIME AS OF Orders.order_time  
ON Orders.currency = CurrencyRates.currency;
```



CONFLUENT

Demo

<https://github.com/twalthr/flink-api-examples>

Summary



TLDR

- Flink's SQL engine is a powerful changelog processor
- Flexible tool for integrating systems with different semantics

There is more...

- **Large coverage of the SQL standard**
 - OVER for streaming aggregation and dedup
 - MATCH_RECOGNIZE for pattern matching
 - TUMBLE/HOP/SESSION for windowing
 - ...
- **CDC connector ecosystem**
 - 3.5k Github stars
 - <https://flink-packages.org/packages/cdc-connectors>
- **Table Store**
 - unified storage engine for dynamic tables
 - <https://nightlies.apache.org/flink/flink-table-store-docs-master/docs/concepts/overview/>

Connector	Database	Driver
mongodb-cdc	<ul style="list-style-type: none">• MongoDB: 3.6, 4.x, 5.0	MongoDB Driver: 4.3.1
mysql-cdc	<ul style="list-style-type: none">• MySQL: 5.6, 5.7, 8.0.x• RDS MySQL: 5.6, 5.7, 8.0.x• PolarDB MySQL: 5.6, 5.7, 8.0.x• Aurora MySQL: 5.6, 5.7, 8.0.x• MariaDB: 10.x• PolarDB X: 2.0.1	JDBC Driver: 8.0.27
oceanbase-cdc	<ul style="list-style-type: none">• OceanBase CE: 3.1.x• OceanBase EE (MySQL mode): 2.x, 3.x	JDBC Driver: 5.1.4x
oracle-cdc	<ul style="list-style-type: none">• Oracle: 11, 12, 19	Oracle Driver: 19.3.0.0
postgres-cdc	<ul style="list-style-type: none">• PostgreSQL: 9.6, 10, 11, 12	JDBC Driver: 42.2.27
sqlserver-cdc	<ul style="list-style-type: none">• Sqlserver: 2012, 2014, 2016, 2017, 2019	JDBC Driver: 7.2.2.jre8
tidb-cdc	<ul style="list-style-type: none">• TiDB: 5.1.x, 5.2.x, 5.3.x, 5.4.x, 6.0.0	JDBC Driver: 8.0.27
Db2-cdc	<ul style="list-style-type: none">• Db2: 11.5	Db2 Driver: 11.5.0.0



Thank you!

Feel free to follow:

@twalthr



CONFLUENT