Table & SQL API

unified APIs for batch and stream processing

Timo Walther
Apache Flink PMC
@twalthr

dataArtisans

Flink Forward @ San Francisco - *April 11th, 2017*

Motivation

DataStream API is great…

- Very expressive stream processing
  - Transform data, update state, define windows, aggregate, etc.
- Highly customizable windowing logic
  - Assigners, Triggers, Evictors, Lateness
- Asynchronous I/O
  - Improve communication to external systems
- Low-level Operations
  - ProcessFunction gives access to timestamps and timers

… but it is not for Everyone!

- Writing DataStream programs is not always easy
  - Stream processing technology spreads rapidly
  - New streaming concepts (time, state, windows, …)
- Requires knowledge & skill
  - Continous applications have special requirements
  - Programming experience (Java / Scala)
- Users want to focus on their business logic

Why not a Relational API?

- Relational API is declarative
  - User says what is needed, system decides how to compute it
- Queries can be effectively optimized
  - Less black-boxes, well-researched field
- Queries are efficiently executed
  - Let Flink handle state, time, and common mistakes
- ”Everybody” knows and uses SQL!

Goals

- Easy, declarative, and concise relational API
- Tool for a wide range of use cases
- Relational API as a unifying layer
  - Queries on batch tables terminate and produce a finite result
  - Queries on streaming tables run continuously and produce result stream
- Same syntax & semantics for both queries

Table API & SQL

Table API & SQL

- Flink features two relational APIs
  - Table API: LINQ-style API for Java & Scala (since Flink 0.9.0)
  - SQL: Standard SQL (since Flink 1.1.0)

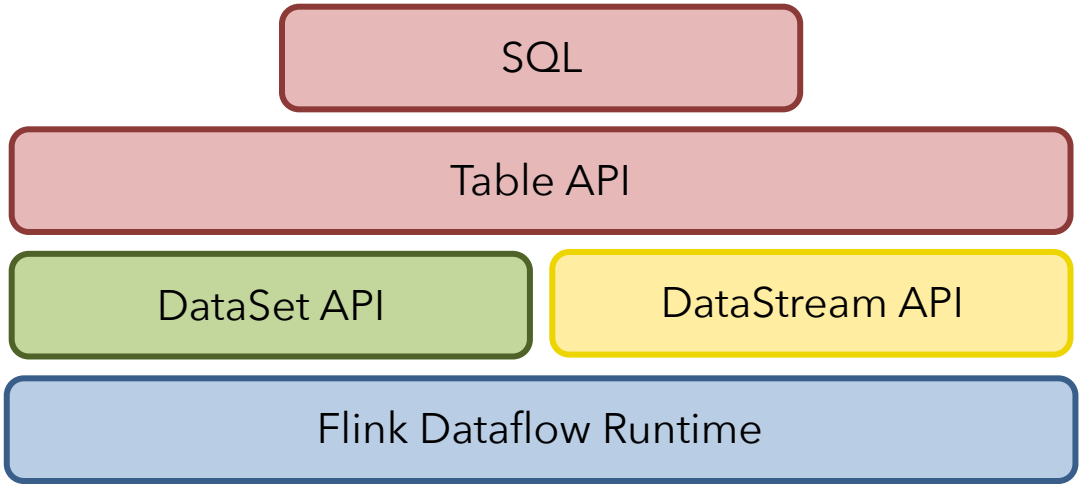


Table API & SQL Example

```
val tEnv = TableEnvironment.getTableEnvironment(env)
// configure your data source
val customerSource = CsvTableSource.builder()
  .path("/path/to/customer_data.csv")
  .field("name", Types.STRING).field("prefs", Types.STRING)
  .build()
// register as a table
tEnv.registerTableSource(”cust", customerSource)
// define your table program
val table = tEnv.scan("cust").select('name.lowerCase(), myParser('prefs))
val table = tEnv.sql("SELECT LOWER(name), myParser(prefs) FROM cust")
// convert
val ds: DataStream[Customer] = table.toDataStream[Customer]
```

Windowing in Table API

```
val sensorData: DataStream[(String, Long, Double)] = ???

// convert DataStream into Table
val sensorTable: Table = sensorData
  .toTable(tableEnv, 'location, 'rowtime, 'tempF)

// define query on Table
val avgTempCTable: Table = sensorTable
  .window(Tumble over 1.day on 'rowtime as 'w)
  .groupBy('location, 'w)
  .select('w.start as 'day,
    'location,
    (('tempF.avg - 32) * 0.556) as 'avgTempC)
  .where('location like "room%")
```

Windowing in SQL

```
val sensorData: DataStream[(String, Long, Double)] = ???

// register DataStream
tableEnv.registerDataStream(
  "sensorData", sensorData, 'location, 'rowtime, 'tempF)

// query registered Table
val avgTempCTable: Table = tableEnv.sql("""
    SELECT TUMBLE_START(TUMBLE(time, INTERVAL '1' DAY) AS day,
         location,
         AVG((tempF - 32) * 0.556) AS avgTempC
    FROM sensorData
    WHERE location LIKE 'room%'
    GROUP BY location, TUMBLE(time, INTERVAL '1' DAY)
  """)
```

Architecture

2 APIs [SQL, Table API]

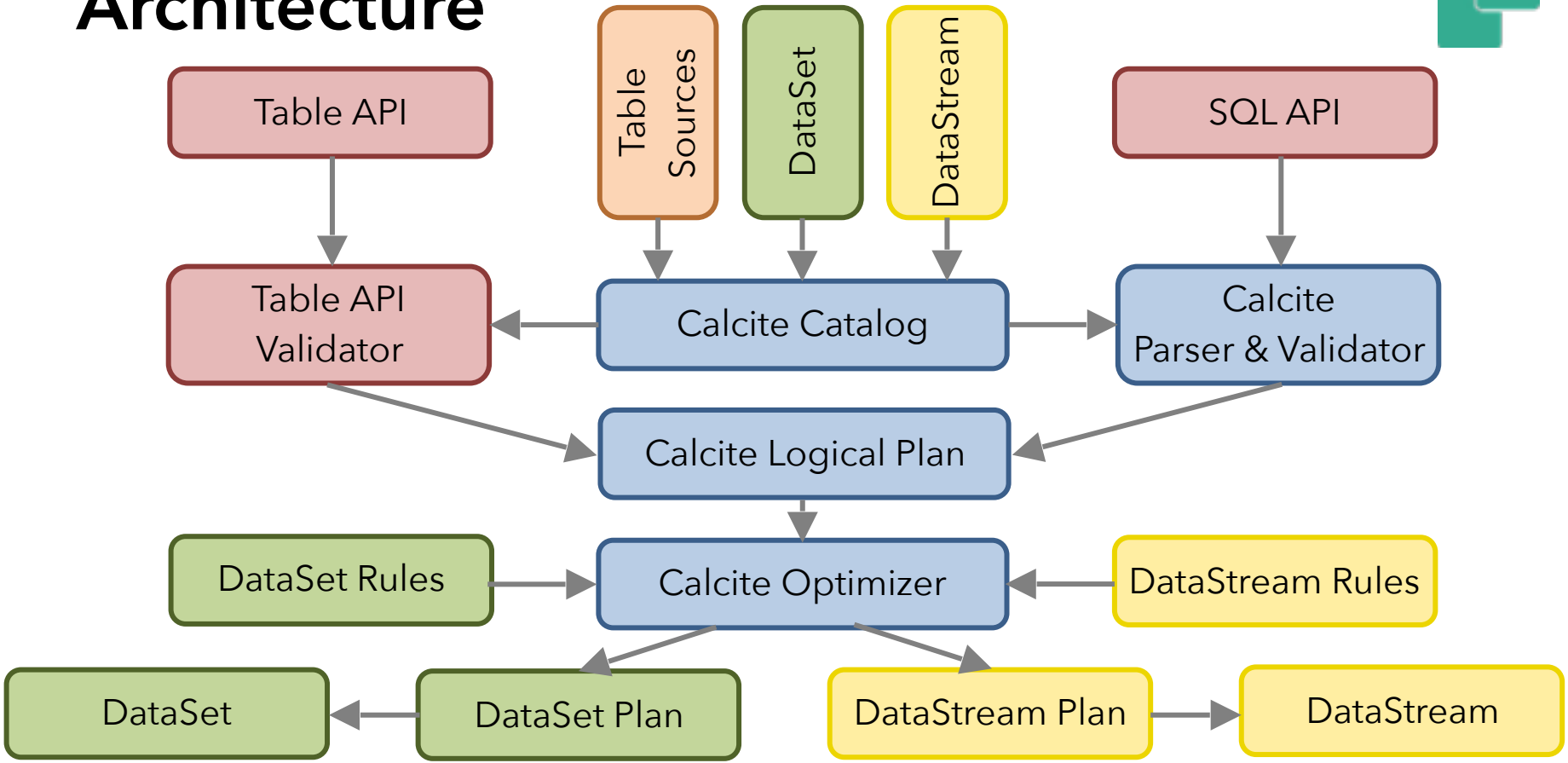
*

2 backends [DataStream, DataSet]

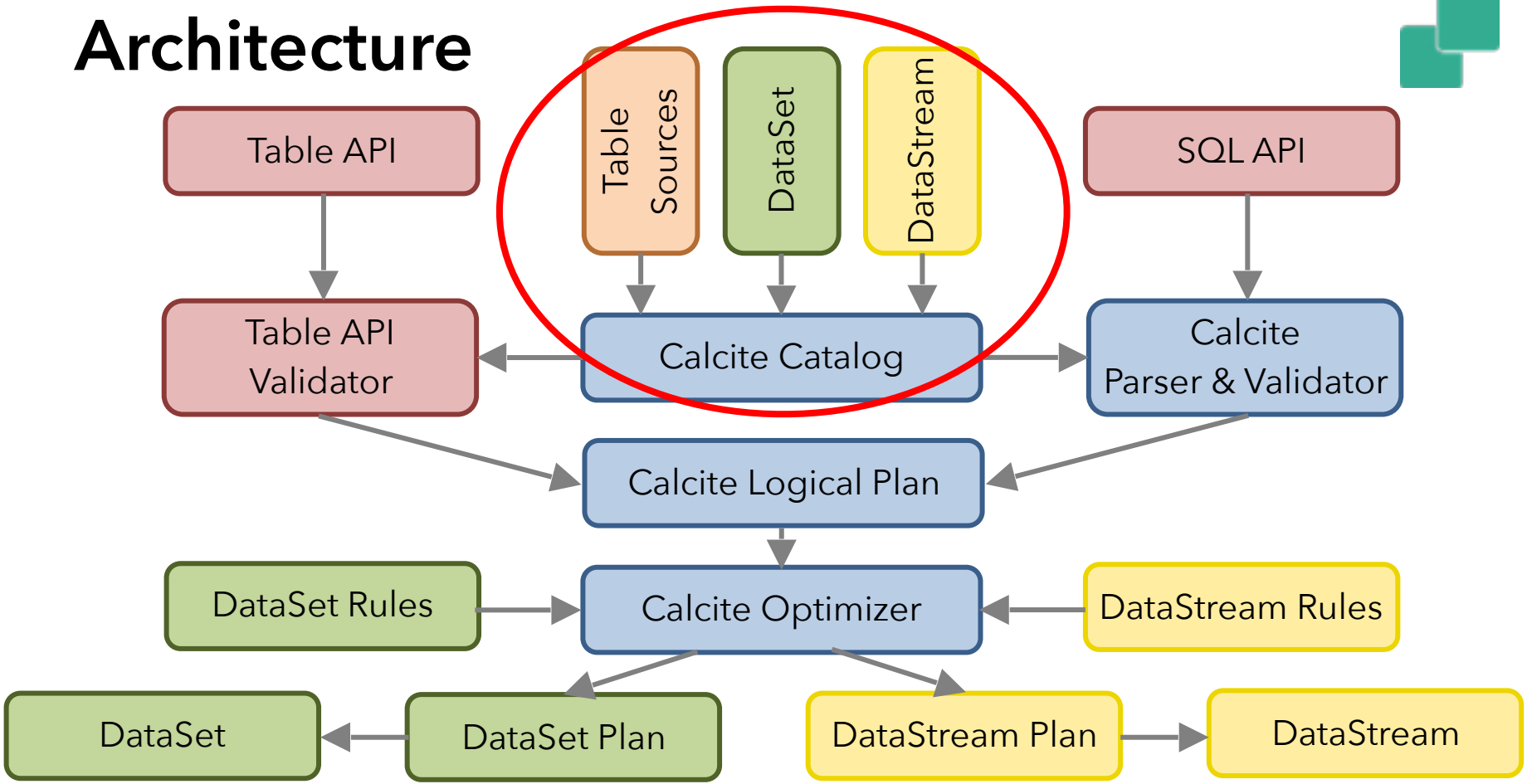
=

4 different translation paths?

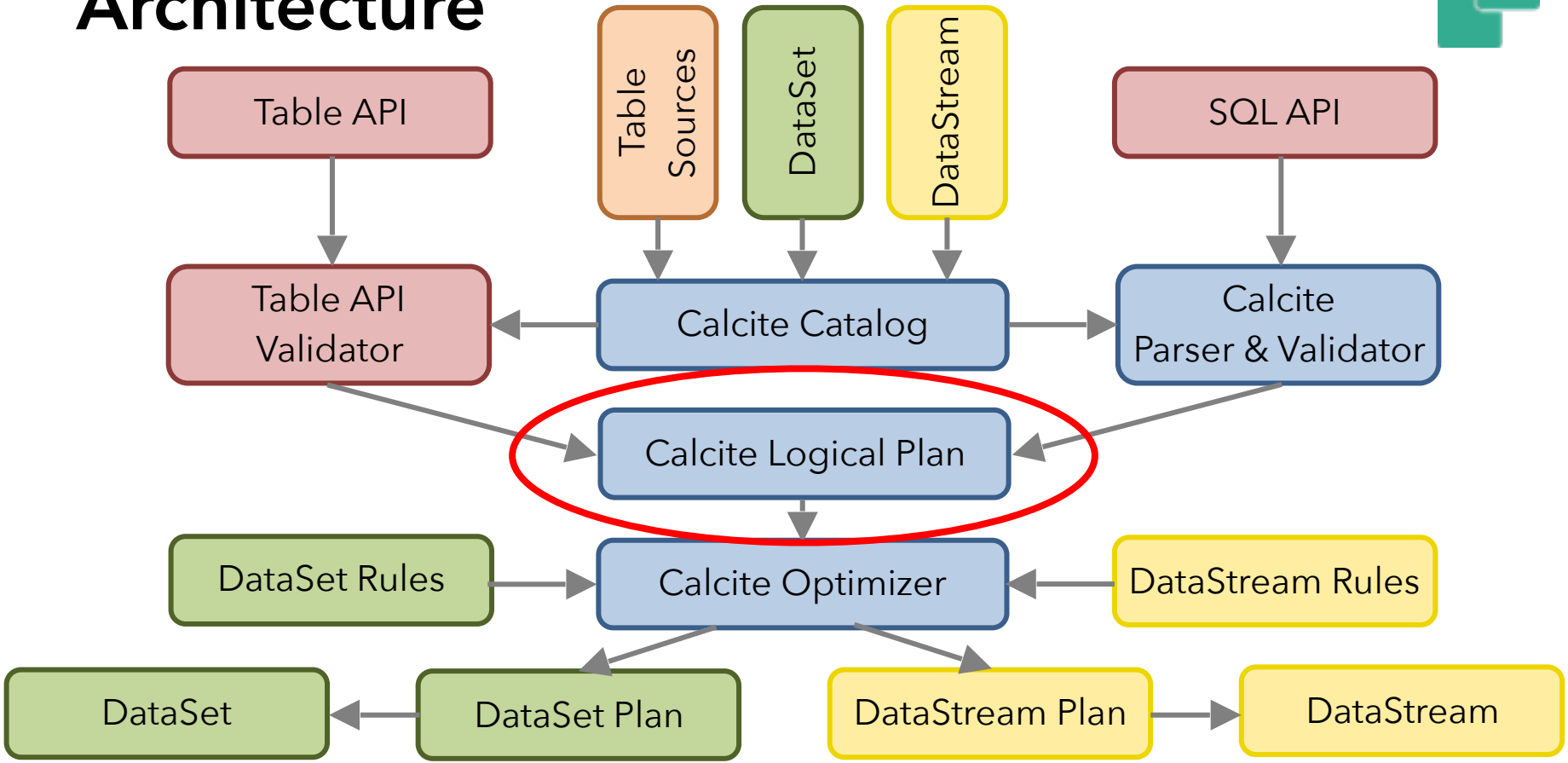
Architecture



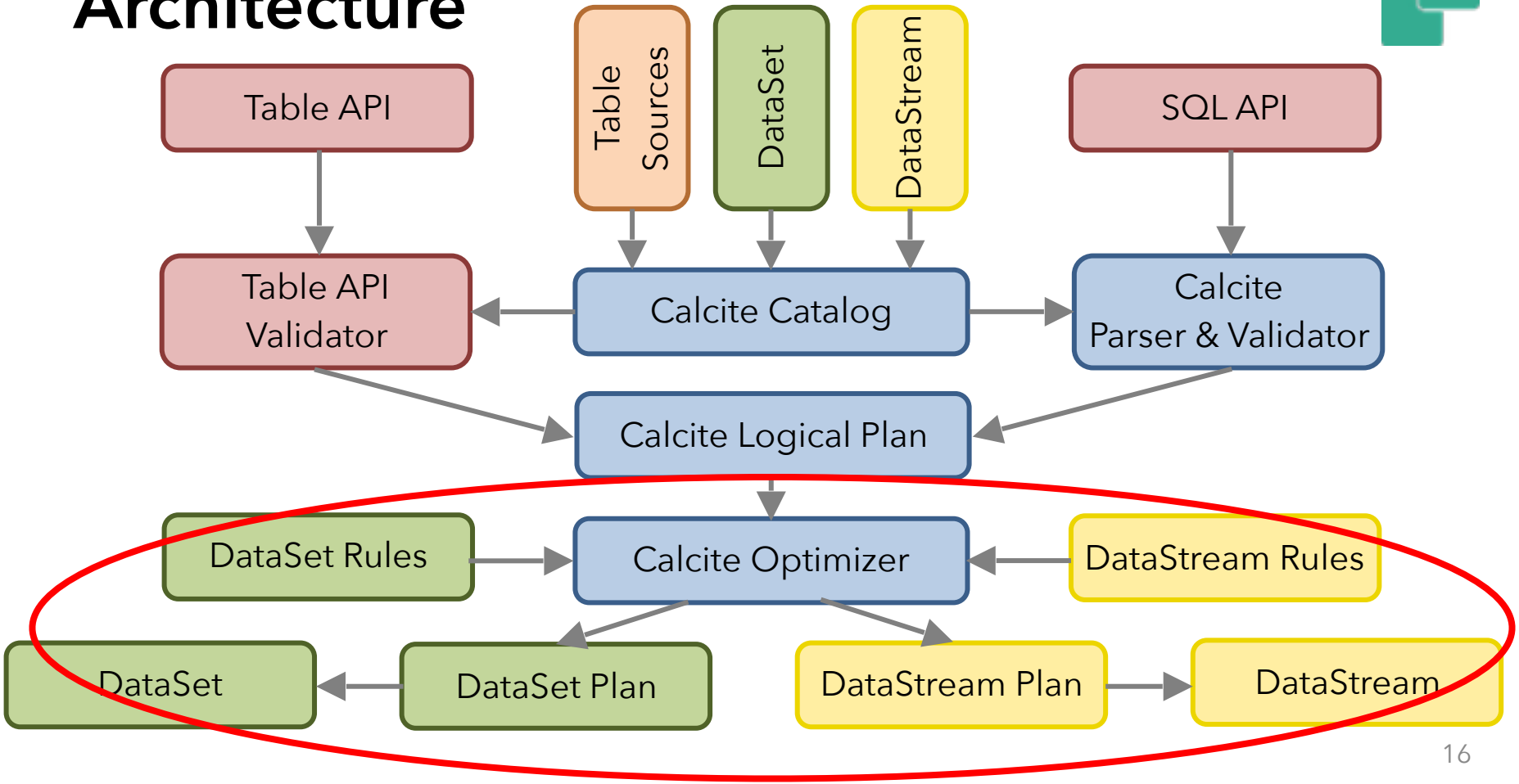
Architecture



Architecture



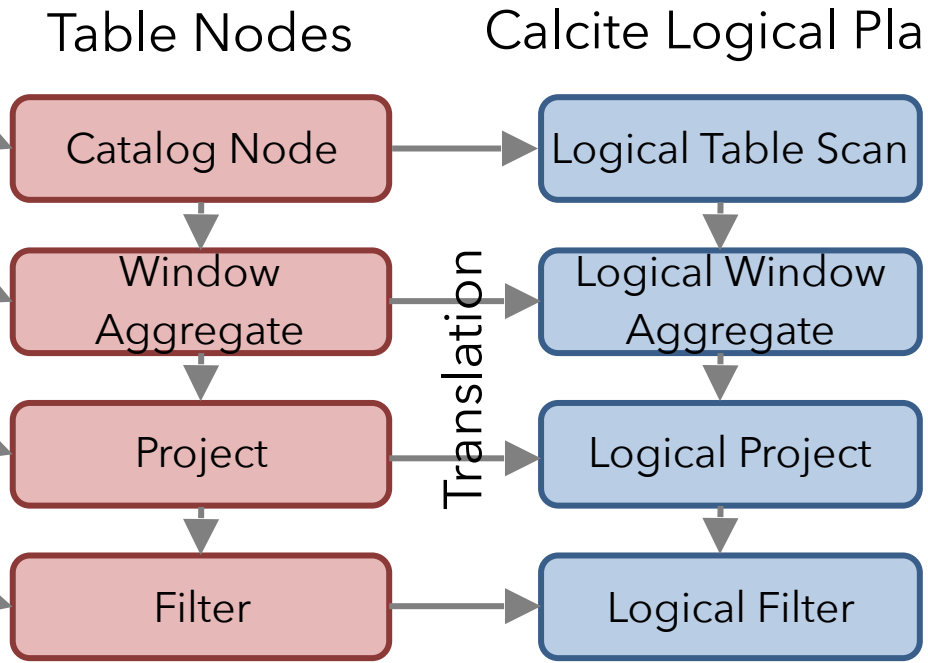
Architecture



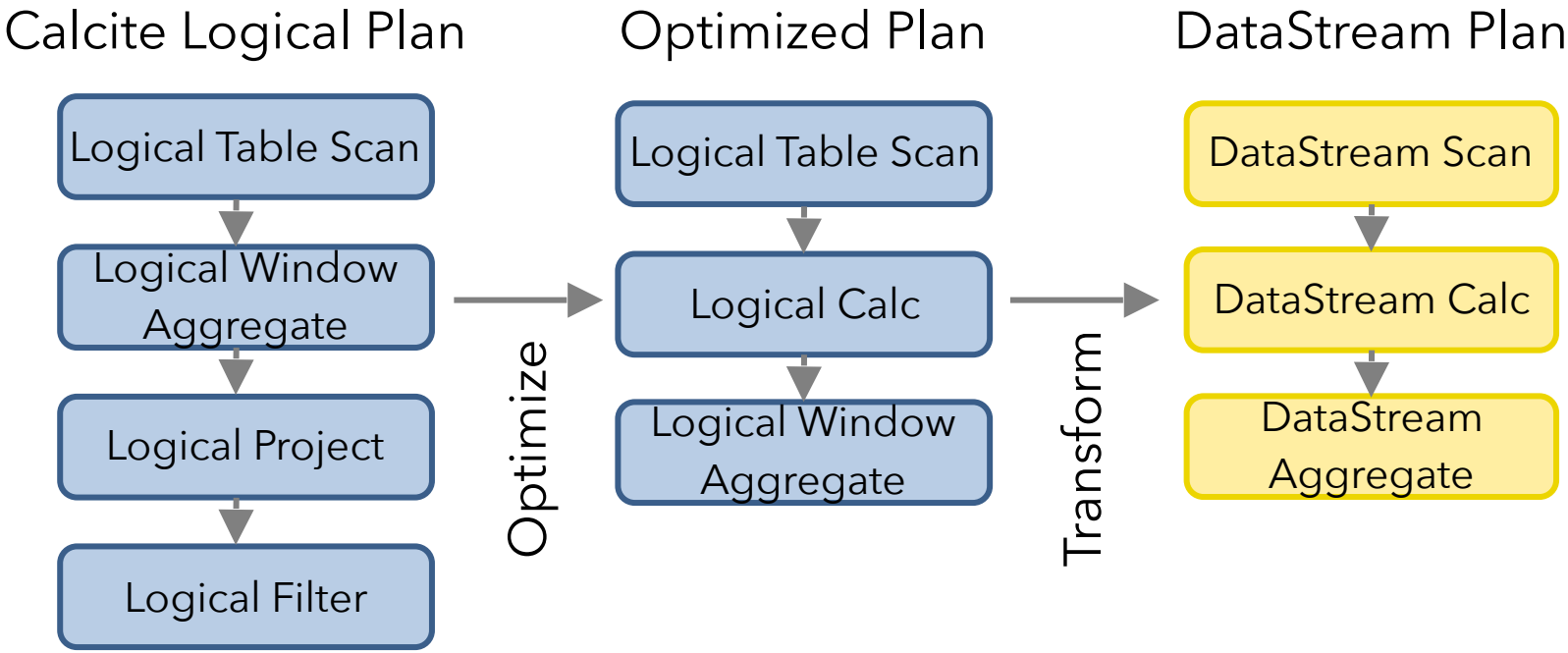
Translation to Logical Plan

```
sensorTable

  .window(Tumble over 1.day on 'rowtime as 'w)
  .groupBy('location, 'w)

  .select(
    'w.start as 'day,
    'location,
    (('tempF.avg - 32) *
        0.556) as 'avgTempC)

  .where('location like "room%")
```

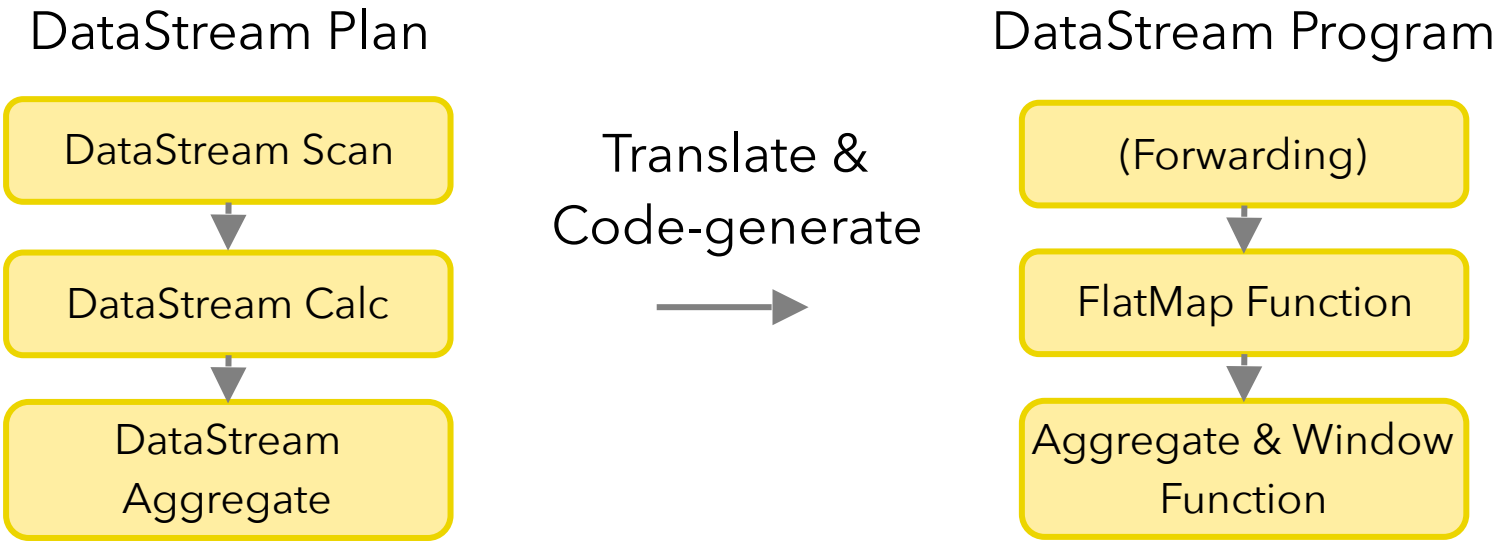
Table API Validation



Translation to DataStream Plan



Translation to Flink Program

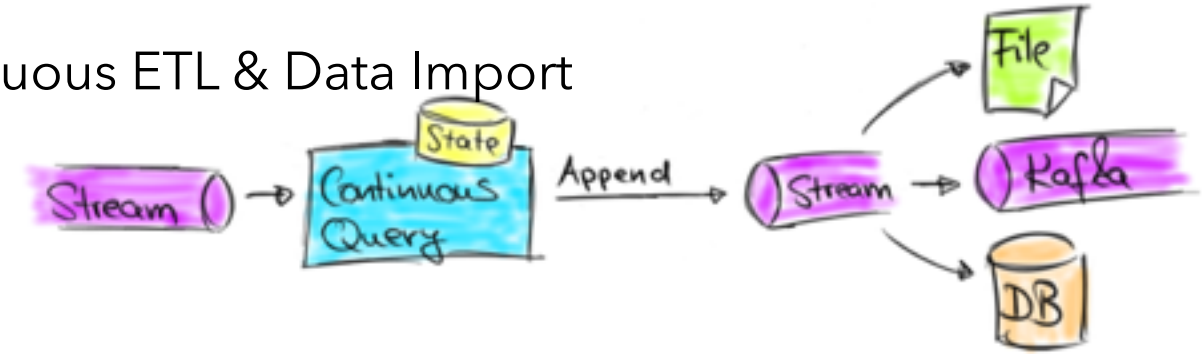


Current State (in master)

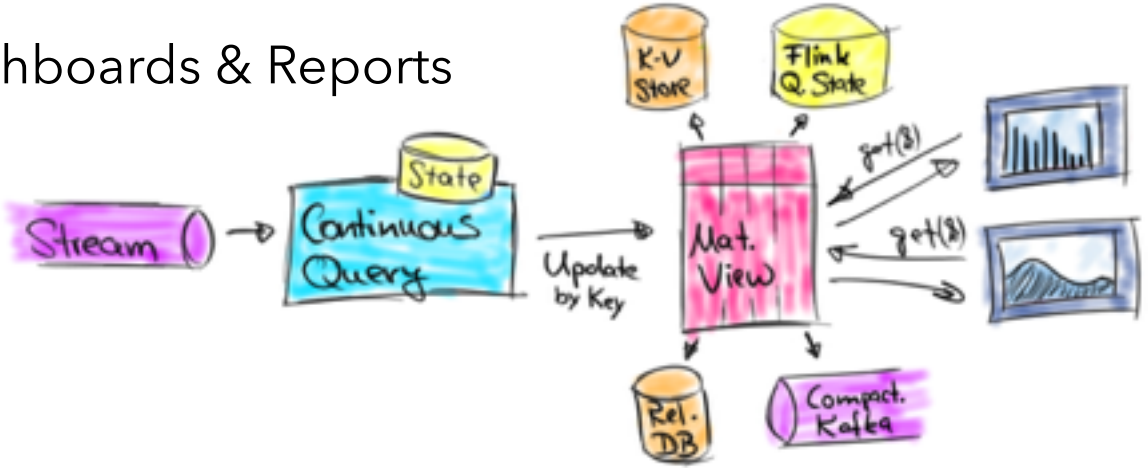
- Batch support
  - Selection, Projection, Sort, Inner & Outer Joins, Set operations
  - Group-Windows for Slide, Tumble, Session
- Streaming support
  - Selection, Projection, Union
  - Group-Windows for Slide, Tumble, Session
  - Different SQL OVER-Windows (RANGE/ROWS)
- UDFs, UDTFs, custom rules

Use Cases for Streaming SQL

- Continuous ETL & Data Import



- Live Dashboards & Reports



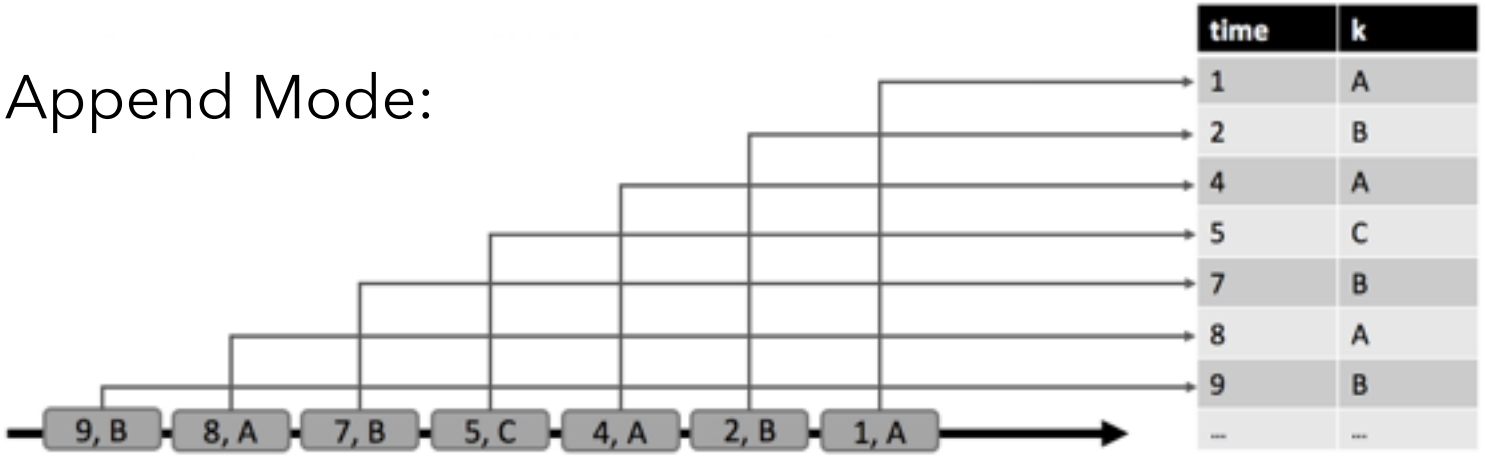
Outlook: Dynamic Tables

Dynamic Tables Model

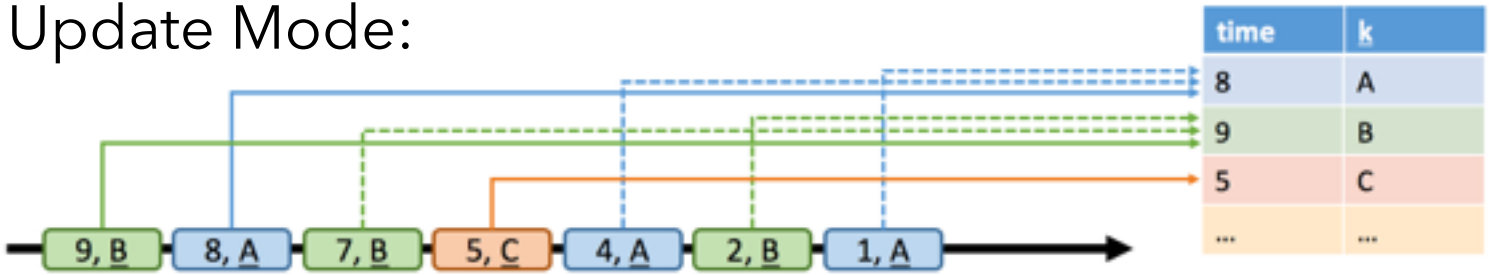
- Dynamic tables change over time
- Dynamic tables are treated like static batch tables
  - Dynamic tables are queried with standard SQL / Table API
  - Every query returns another Dynamic Table
- "Stream / Table Duality"
  - Stream ←→ Dynamic Table
    conversions without information loss

Stream to Dynamic Table

- Append Mode:



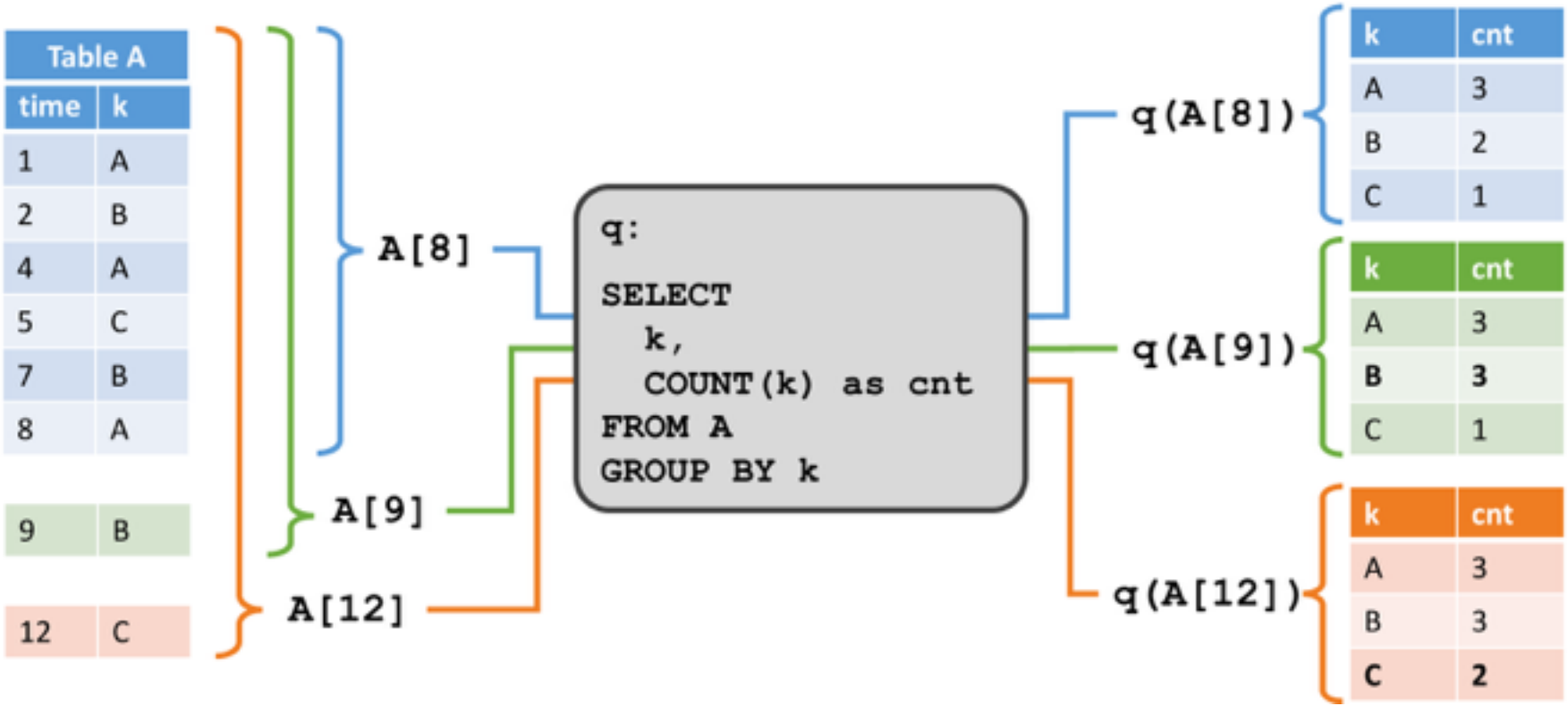
- Update Mode:



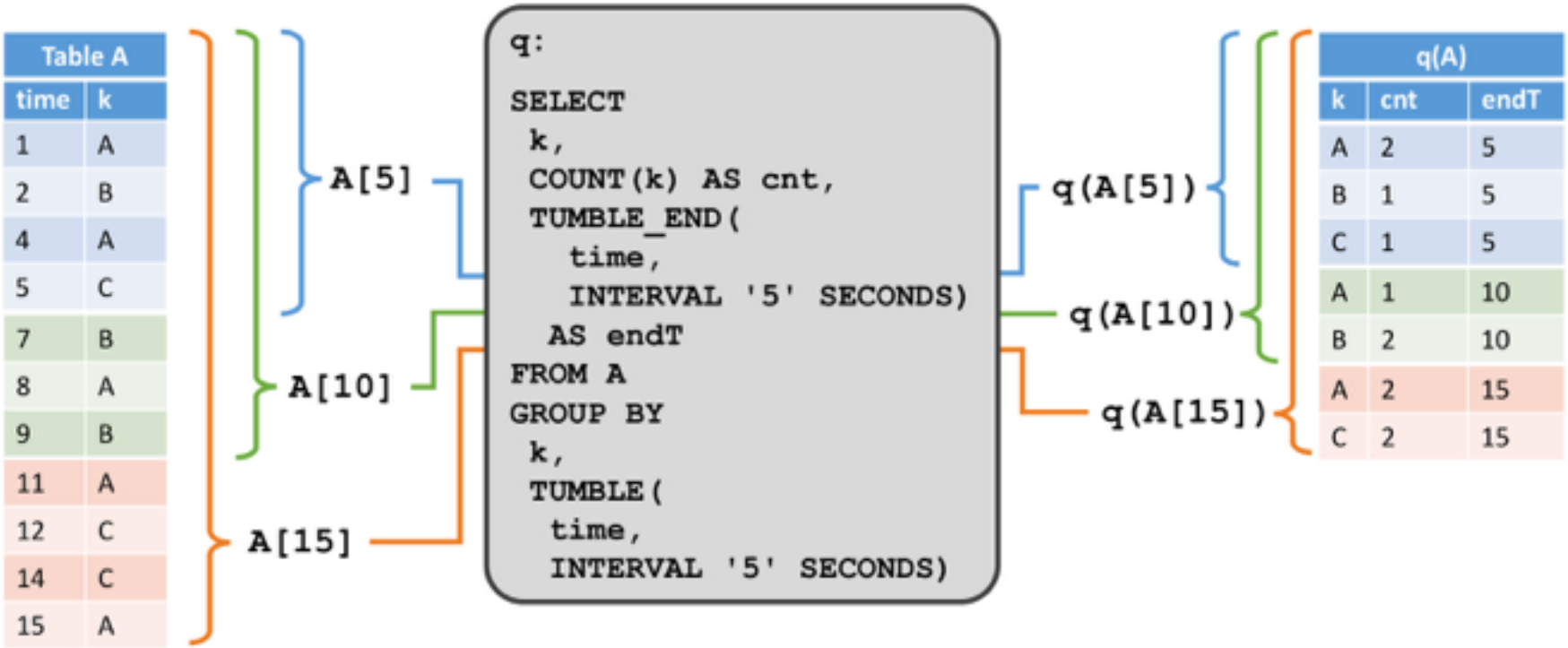
Querying Dynamic Tables

- Dynamic tables change over time
  - *A[t]*: Table *A* at specific point in time *t*
- Dynamic tables are queried with relational semantics
  - Result of a query changes as input table changes
  - *q(A[t])*: Evaluate query *q* on table *A* at time *t*
- Query result is continuously updated as t progresses
  - Similar to maintaining a materialized view
  - *t* is current event time

Querying a Dynamic Table



Querying a Dynamic Table



Querying a Dynamic Table

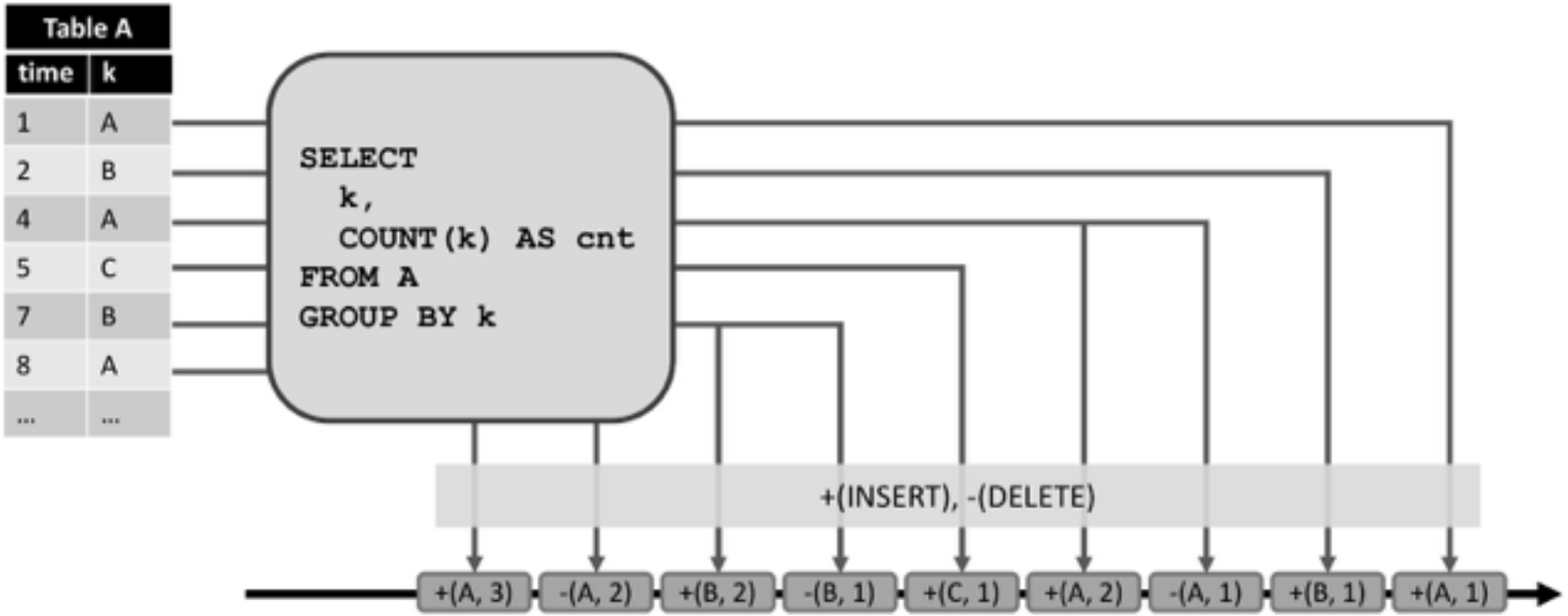
- Can we run any query on Dynamic Tables? No!
- State may not grow infinitely as more data arrives
  - Set clean-up timeout or key constraints.
- Input may only trigger partial re-computation
- Queries with possibly unbounded state or computation are rejected

Dynamic Table to Stream

- Convert Dynamic Table modifications into stream messages
- Similar to database logging techniques
  - Undo: previous value of a modified element
  - Redo: new value of a modified element
  - Undo+Redo: old and the new value of a changed element
- For Dynamic Tables: Redo or Undo+Redo

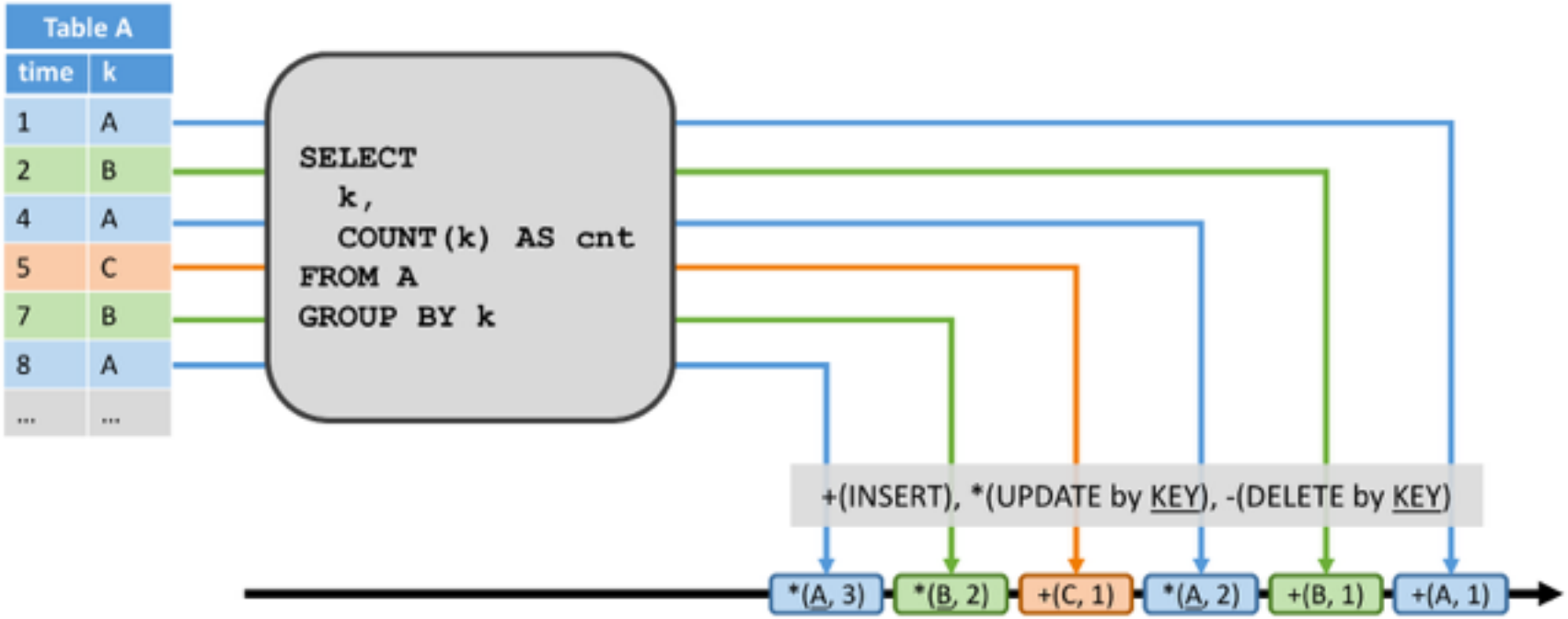
Dynamic Table to Stream

- Undo+Redo Stream (because A is in Append Mode):

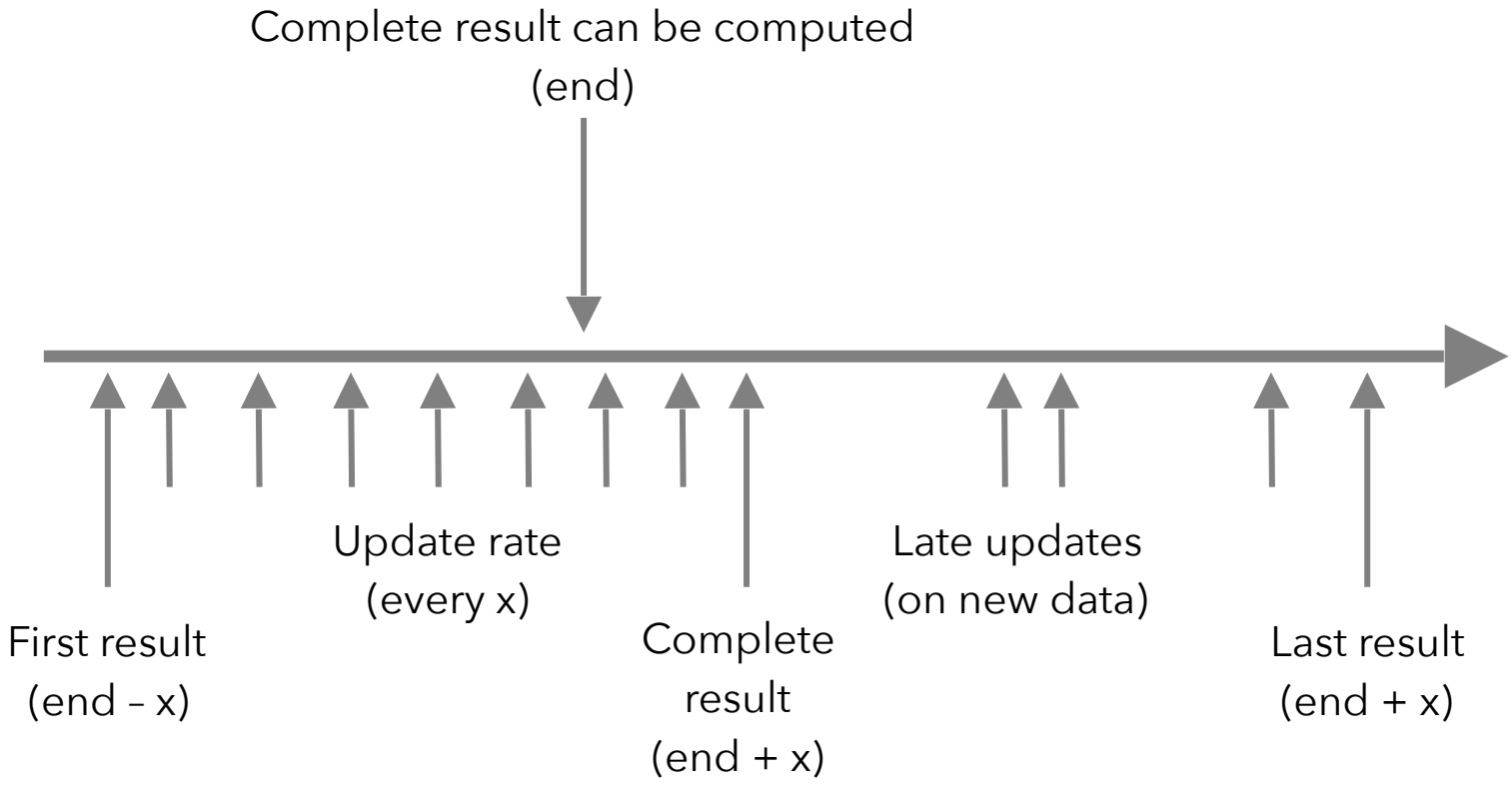


Dynamic Table to Stream

- Redo Stream (because A is in Update Mode):



Result computation & refinement



State is purged.

Contributions welcome!

- Huge interest and many contributors
  - Adding more window operators
  - Introducing dynamic tables

- And there is a lot more to do
  - New operators and features for streaming and batch
  - Performance improvements
  - Tooling and integration
- Try it out, give feedback, and start contributing!

Thank you!

@twalthr

@ApacheFlink

@dataArtisans