# SQL ON DATA STREAMS

FABIAN HUESKE, SOFTWARE ENGINEER

FLINK FORWARD BERLIN
SEPTEMBER 3, 2018

dataArtisans

# STREAMS & DYNAMIC TABLES




# SQL WAS NOT DESIGNED FOR STREAMS

| | | |
|---|---|---|
| • Table are bounded multisets. | ↔ | Streams are infinite sequences. |
| • DBMS queries can access all data. | ↔ | Streaming queries receive data over time. |
| • DBMS queries return a finite result. | ↔ | Streaming queries continuously emit results and never complete. |

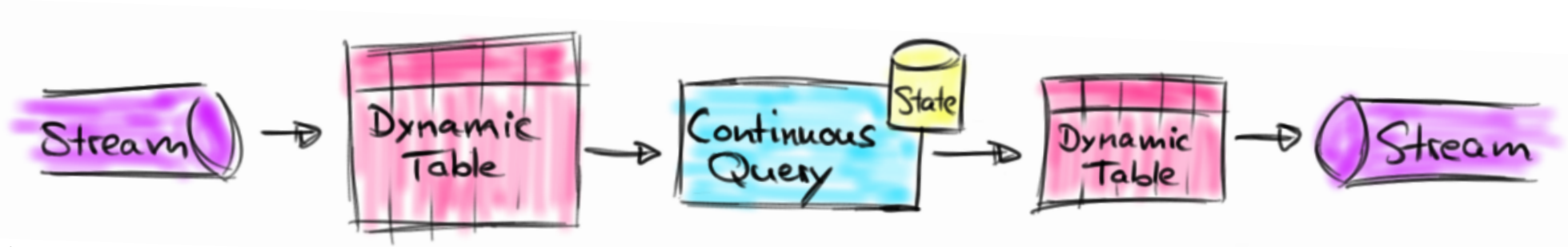
# DATABASE SYSTEMS RUN QUERIES ON STREAMS

- Materialized views (MV) are similar to regular views, but persisted to disk or memory
  - Used to speed-up analytical queries
  - MVs need to be updated when the base tables change
- MV maintenance is very similar to SQL on streams
  - Base table updates are a stream of DML statements
  - MV definition query is evaluated on that stream
  - MV is query result and continuously updated

# CONTINUOUS QUERIES IN FLINK

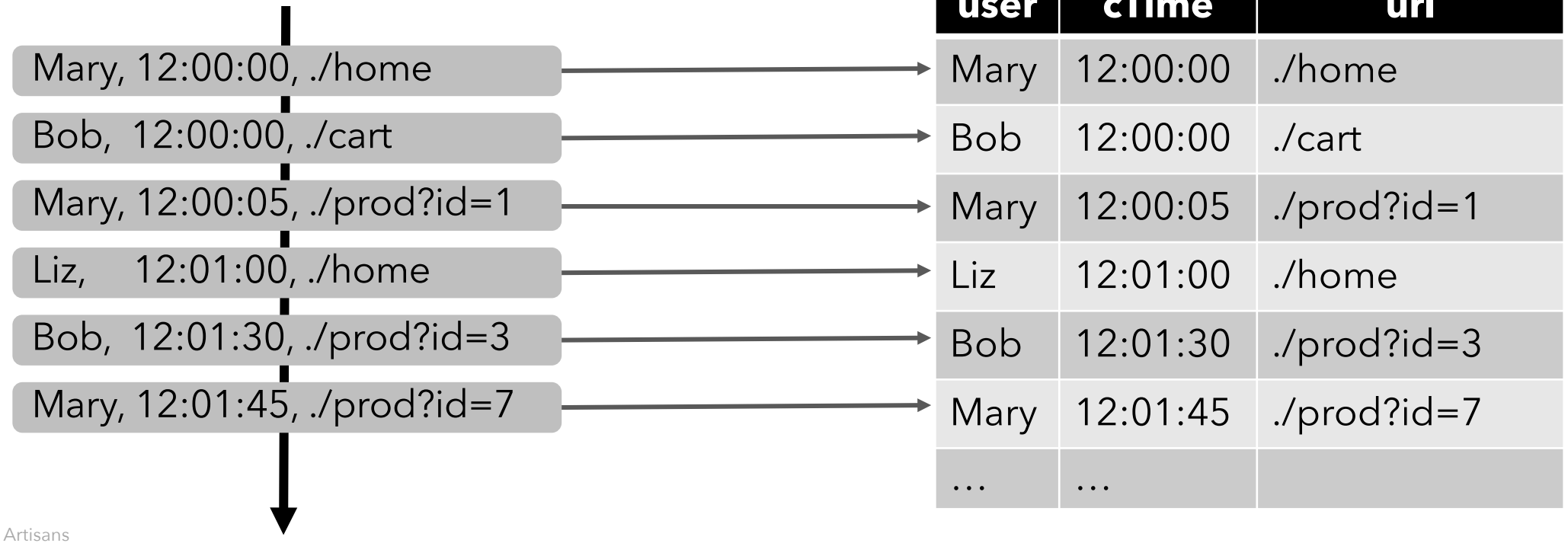
- Core concept is a *"Dynamic Table"*
  - Dynamic tables are changing over time
- Queries on dynamic tables
  - produce new dynamic tables (which are updated based on input)
  - do not terminate
- Stream ↔ Dynamic table conversions




 | 

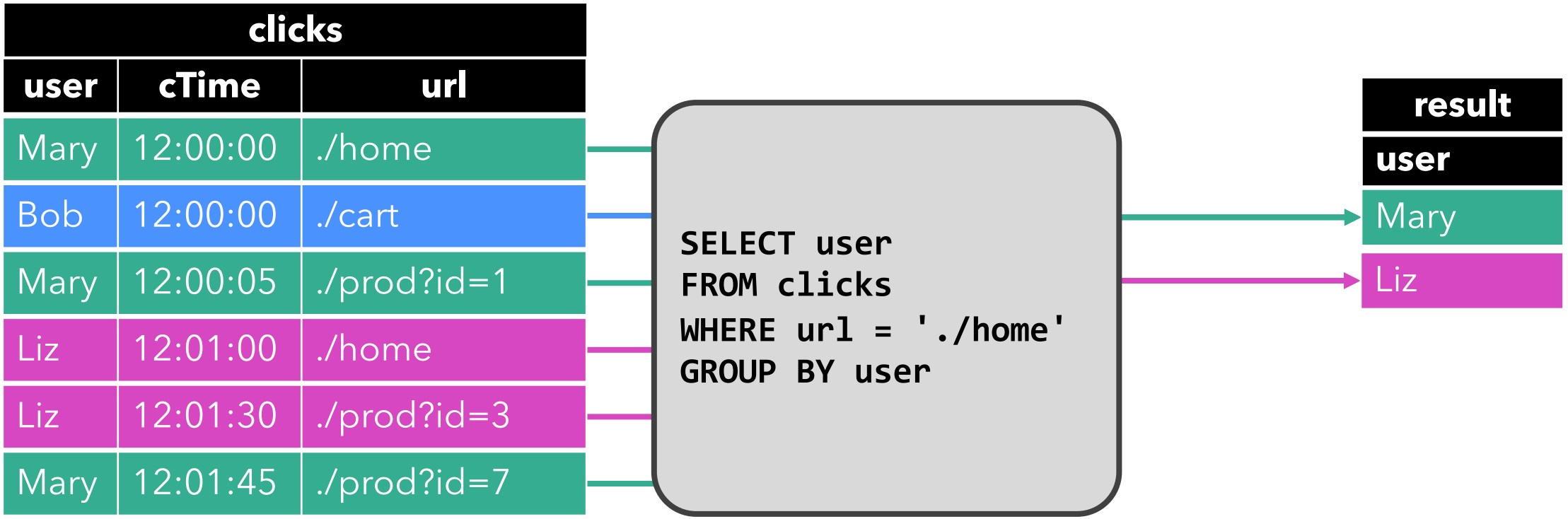
# STREAM → DYNAMIC TABLE: APPEND

- Append mode
  - Stream records are appended to table
  - Table grows as more data arrives



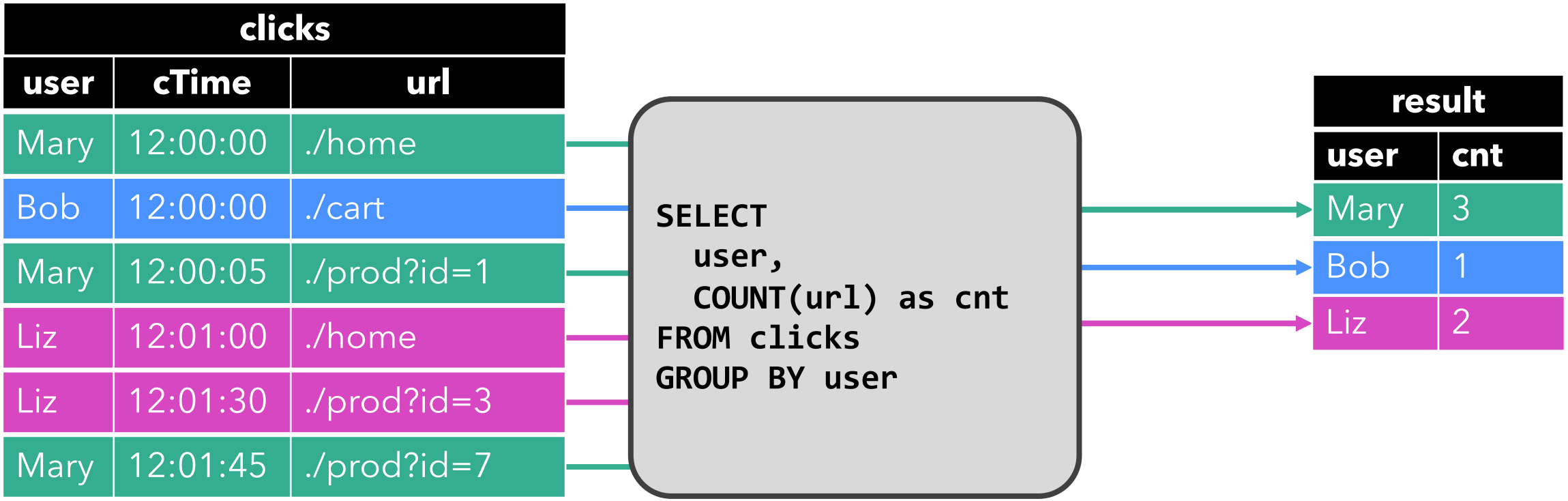

 | 

# QUERYING A DYNAMIC TABLE




Rows of result table are appended.

# QUERYING A DYNAMIC TABLE



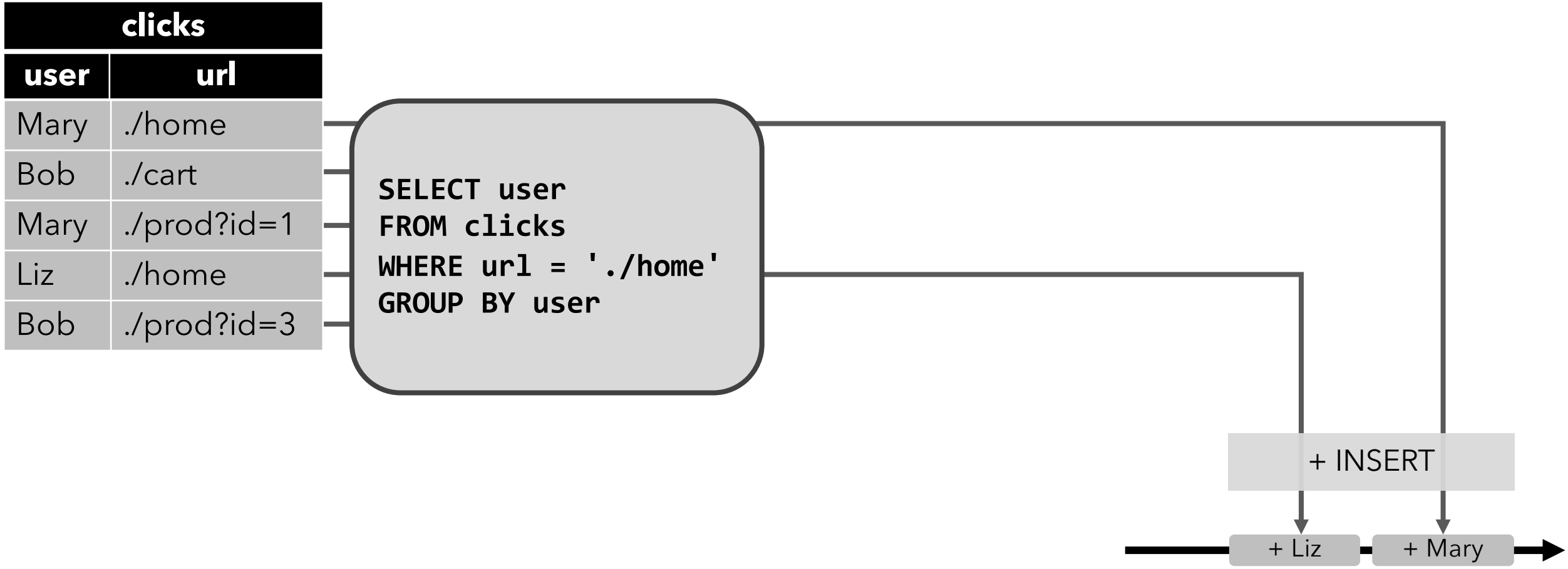

Rows of result table are updated.

# DYNAMIC TABLE → STREAM

- Converting a dynamic table into a stream
  - Dynamic tables might update or delete existing rows
  - Updates must be encoded in outgoing stream

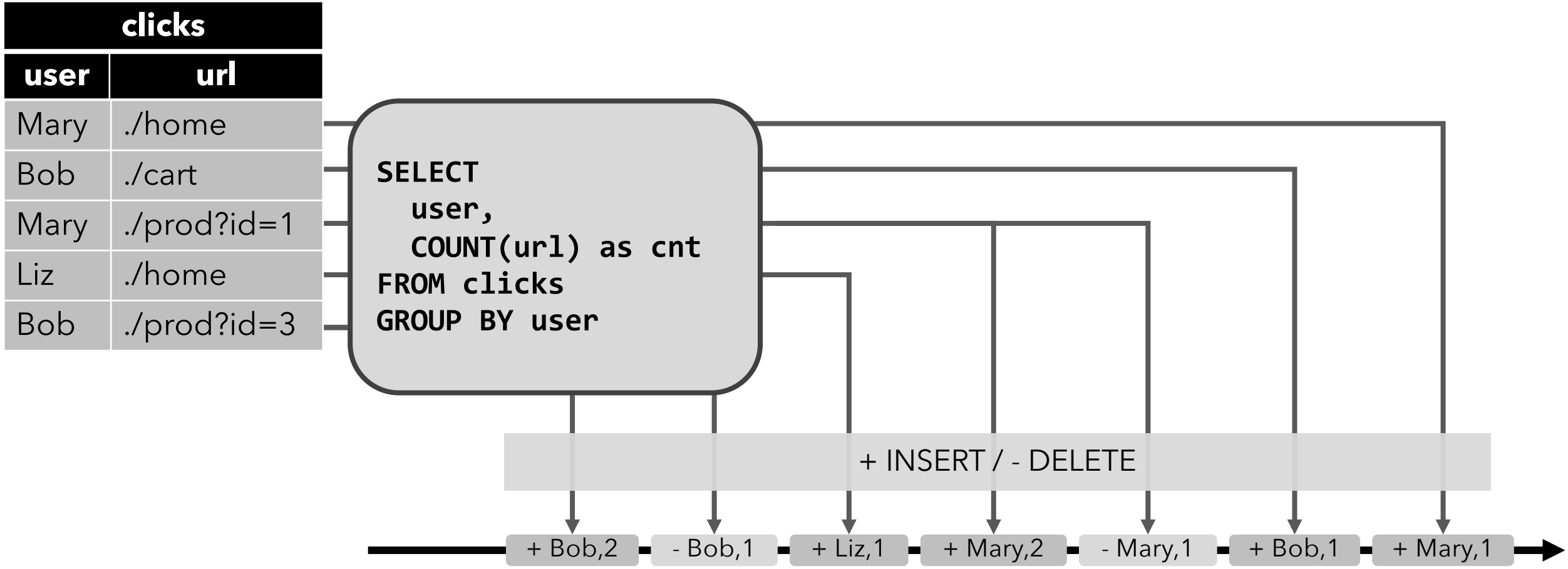



# DYNAMIC TABLE → STREAM: APPEND-ONLY



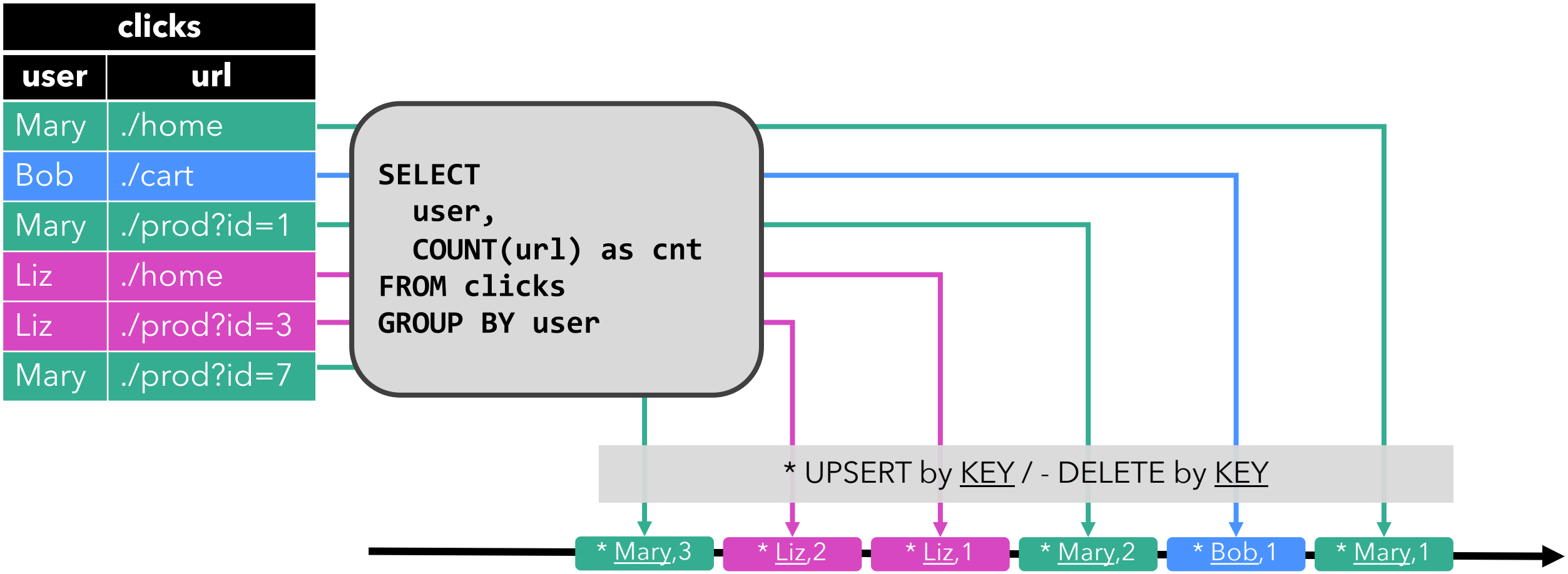

 | 

# DYNAMIC TABLE → STREAM: RETRACTION




 | 

# DYNAMIC TABLE → STREAM: UPSERT




 | 

# SUMMARY

- Streams are interpreted as changelog for a Table
  - Flink 1.6.0 supports append-only stream to table conversion
  - Upsert and Insert/Update/Delete conversions are on the roadmap
- SQL queries on dynamic tables yield another dynamic table
  - Input and query determines whether resulting dynamic table is append-only or updating
- Dynamic tables can be converted back into streams
  - Append-only, Upsert, or Retraction

# EVENT & PROCESSING TIME

 | 

# HOW IS TIME HANDLED IN FLINK SQL?

- Flink SQL supports Event and Processing Time

- Tables may include *Time Attributes*
  - Time Attributes provide access to event or processing time
  - Time Attributes are (mostly) treated as regular attributes
  - Time Attributes have special type extended from SQL TIMESTAMP

- Time Attributes are declared with the table schema

# EVENT TIME ATTRIBUTE

- Event time attributes carry an actual timestamp
- Timestamps are extracted during table scan
  - Typically taken from an existing field
- Watermarks are generated based on the timestamps
  - Different strategies available
- Event time attributes can be used like a regular TIMESTAMP
  - Event time property (and watermark alignment) are lost when being modified

# PROCESSING TIME ATTRIBUTE

- Processing time attributes are virtual and do not hold data
  - Local machine time is queried on attribute access

- A proc time attribute be used like a regular TIMESTAMP
  - Loses its processing time property when being modified

# HOW ARE TIME ATTRIBUTES USED?

- Some operators require time attributes in certain clauses
  - GROUP BY windows
  - OVER windows
  - Time-windowed joins

- The clicks table is used for the following examples
  - cTime (clickTime) is an event time attribute.

| user | cTime | url |
|---|---|---|
| Mary | 12:00:00 | ./home |
| Bob | 12:00:00 | ./cart |
| Mary | 12:00:05 | ./prod?id=1 |
| … | … | … |




# GROUP BY WINDOW AGGREGATION

- Compute number of clicks per hour and user

Time attribute

```
SELECT user,
       TUMBLE_END(cTime, INTERVAL '1' HOURS) AS endT,
       COUNT(url) AS cnt
FROM clicks
GROUP BY TUMBLE(cTime, INTERVAL '1' HOURS),
         user
```

Time attribute

# GROUP BY WINDOW AGGREGATION

**clicks**

| user | cTime | url |
|---|---|---|
| Mary | 12:00:00 | ./home |
| Bob | 12:00:00 | ./cart |
| Mary | 12:02:00 | ./prod?id=2 |
| Mary | 12:55:00 | ./home |
| Bob | 13:01:00 | ./prod?id=4 |
| Liz | 13:30:00 | ./cart |
| Liz | 13:59:00 | ./home |
| Mary | 14:00:00 | ./prod?id=1 |
| Liz | 14:02:00 | ./prod?id=8 |
| Bob | 14:30:00 | ./prod?id=7 |
| Bob | 14:40:00 | ./home |

```
SELECT
 user,
 TUMBLE_END(
   cTime,
   INTERVAL '1' HOURS)
  AS endT,
  COUNT(url) AS cnt
FROM clicks
GROUP BY
 user,
 TUMBLE(
  cTime,
  INTERVAL '1' HOURS)
```

**result**

| user | endT | cnt |
|---|---|---|
| Mary | 13:00:00 | 3 |
| Bob | 13:00:00 | 1 |
| Bob | 14:00:00 | 1 |
| Liz | 14:00:00 | 2 |
| Mary | 15:00:00 | 1 |
| Bob | 15:00:00 | 2 |
| Liz | 15:00:00 | 1 |

Rows are appended to result table.

# OVER WINDOW AGGREGATION

- Compute for each click
  how often the URL was clicked in the previous 10 minutes

```
SELECT
  user,
  url,
  COUNT(*) OVER (
    PARTITION BY url
    ORDER BY cTime
    RANGE BETWEEN INTERVAL ’10’ MINUTE PRECEDING AND CURRENT ROW)
FROM clicks
```

Time attribute




# TIME-WINDOWED JOIN

- Find all product URLs that were clicked by a user less than 5 seconds after being served.
  - Assuming a table `serves` with an event time attribute `sTime`

| url | sTime | user |
|---|---|---|
| ./home | 12:00:00 | Mary |
| ./cart | 12:00:00 | Liz |
| ./prod?id=1 | 12:00:05 | Bob |
| … | … | … |

```
SELECT url
FROM clicks c, serves s
WHERE s.url LIKE './prod%' AND
      s.url = c.url AND
      s.user = c.user AND
      c.cTime BETWEEN s.sTime AND s.sTime + INTERVAL '5' SECOND
```

Time attributes

# WHY ARE TIME ATTRIBUTES SPECIAL?

- Time attribute values are (quasi) monotonously increasing
- Operators know when rows are no longer needed
  - Watermarks (event time) or wall-clock time (processing time)
- Time-based operators automatically prune their state
  - Only hold the relevant “tail” of the stream is kept in state

# TIME ATTRIBUTES AND NON-WINDOWED OPERATORS

- Non-windowed aggregations and joins do not forward time attributes
  - Window operators cannot be applied on their results!

- Non-windowed join does not preserve timestamp order
  - Any row in the state could be join with any new arriving row
  - Order of time attributes is not maintained
  - Non-windowed joins must not emit time attributes.

- Non-windowed aggregation does not forward time attributes
  - Time attributes are converted to regular TIMESTAMP attributes and lose their property
  - `SELECT cTime, COUNT(*) FROM clicks GROUP BY cTime`
    - `cTime` is regular `TIMESTAMP`
  - `SELECT user, MAX(cTime) AS cTime FROM clicks GROUP BY user`
    - `cTime` is regular `TIMESTAMP`

# TIME ATTRIBUTES AND BATCH PROCESSING

- Porting SQL queries from streaming to batch or vice versa
- Queries on event time attributes
  - Batch table requires time attribute of type TIMESTAMP
- Queries on processing time attributes
  - Not supported by batch queries
  - What would be the semantics anyway?

# QUERIES AND STATE

# STATELESS AND STATEFUL OPERATORS

- Stateless operators
  - Filter
  - Projection
- Stateful streaming operators
  - Window Aggregation (GROUP BY, OVER)
  - Window Joins
- Stateful materializing operators
  - Aggregation
  - Joins

# STATEFUL MATERIALIZING OPERATORS

- Materializing operators may never remove state
- The aggregation needs to maintain a count for every user forever.
  - Every user could click at any point in time

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

- The aggregation state is continuously growing
  - Unless key domain is bounded
- Joins need to materialize all input rows in state




# MANAGING STATE SIZE

- Query state might grow indefinitely
- Slowly growing state can be addressed by scaling the query
  - `SELECT user, COUNT(*) FROM logins GROUP BY user;`
- State can be automatically pruned
  - `SELECT session, COUNT(*) FROM clicks GROUP BY session;`
  - Rows and persisted results can be removed after an idle timeout

# IDLE STATE CLEAN UP

- The query result is not updated when state is removed
- Query result remains consistent if removed state is not needed again
- Query result becomes inconsistent if query needs to access state that was removed!
- Trade the accuracy of the result for size of state

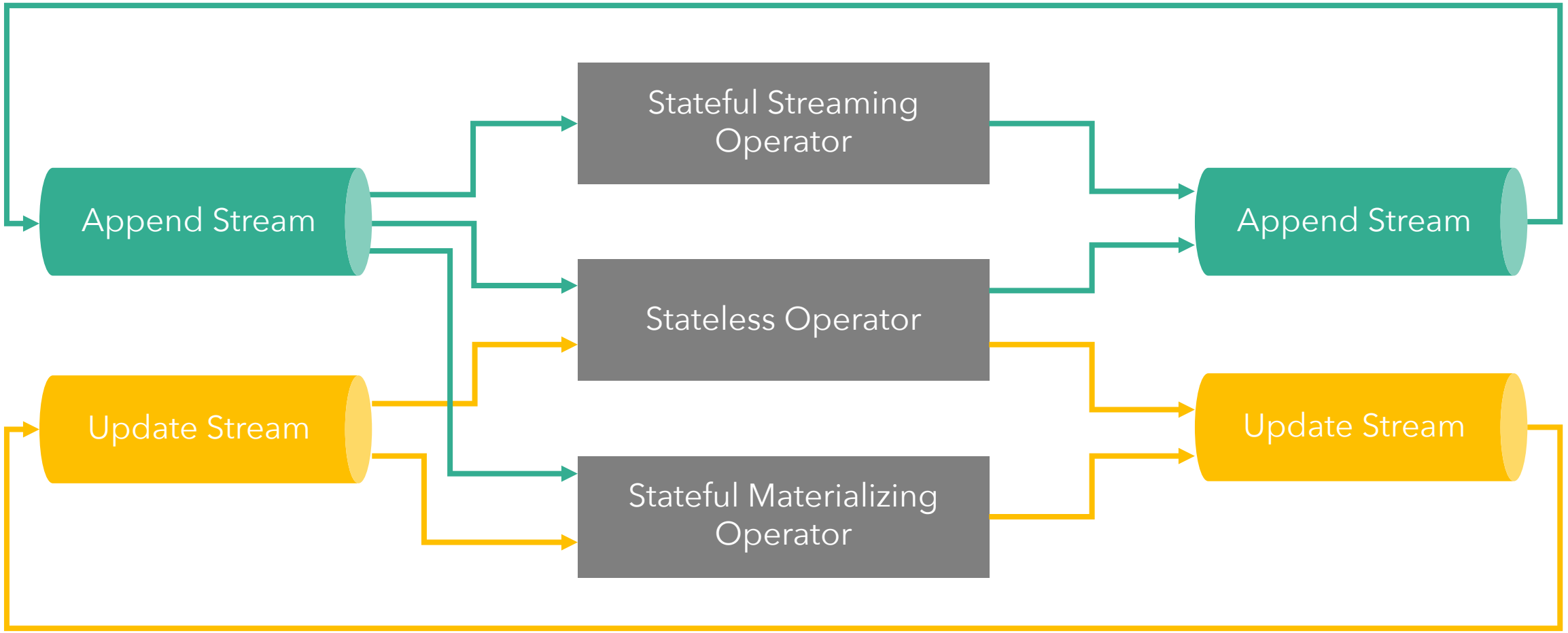
# APPEND AND UPDATE INPUT AND OUTPUT

- Stateful materializing operators may produce updates
  - All required state is available
  - Previous input can be updated
  - Results can be updated

- Stateful streaming operators cannot handle updates
  - State is evicted based on time
  - Results cannot be updated because state might be gone

# APPEND AND UPDATE INPUT AND OUTPUT




 | 

# OTHER CONSTRAINTS?

- A change of an input table may only trigger a partial re-computation of the result table

```
SELECT user, RANK() OVER (ORDER BY lastLogin) FROM users;
```

# SUMMARY

# SUMMARY

- Query evaluation on Dynamic Tables
  - Stream -> Dynamic Table -> Stream conversions
  - append-only, update, retraction
- Event-time and processing-time in SQL
  - Time attributes & windowed operators
- Queries and State
  - State management and automatic clean up

# HANDS ON

 | 

# RUNNING QUERIES ON STREAMS

Continue with Session 4 “Running Queries on Streams”
in the SQL training wiki:

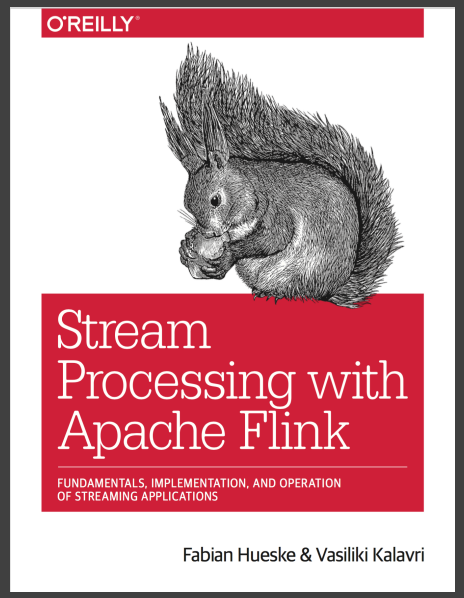
https://github.com/dataArtisans/sql-training/wiki

We are here to help!




# THANK YOU!

@fhueske
@dataArtisans
@ApacheFlink




Available on O'Reilly Early Release!

WE ARE HIRING

data-artisans.com/careers

dataArtisans