Querying Dynamic Tables with SQL

Flink SQL Training

https://github.com/ververica/sql-training

Streams & Dynamic Tables

SQL Was Not (Originally) Designed for Streaming Data

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

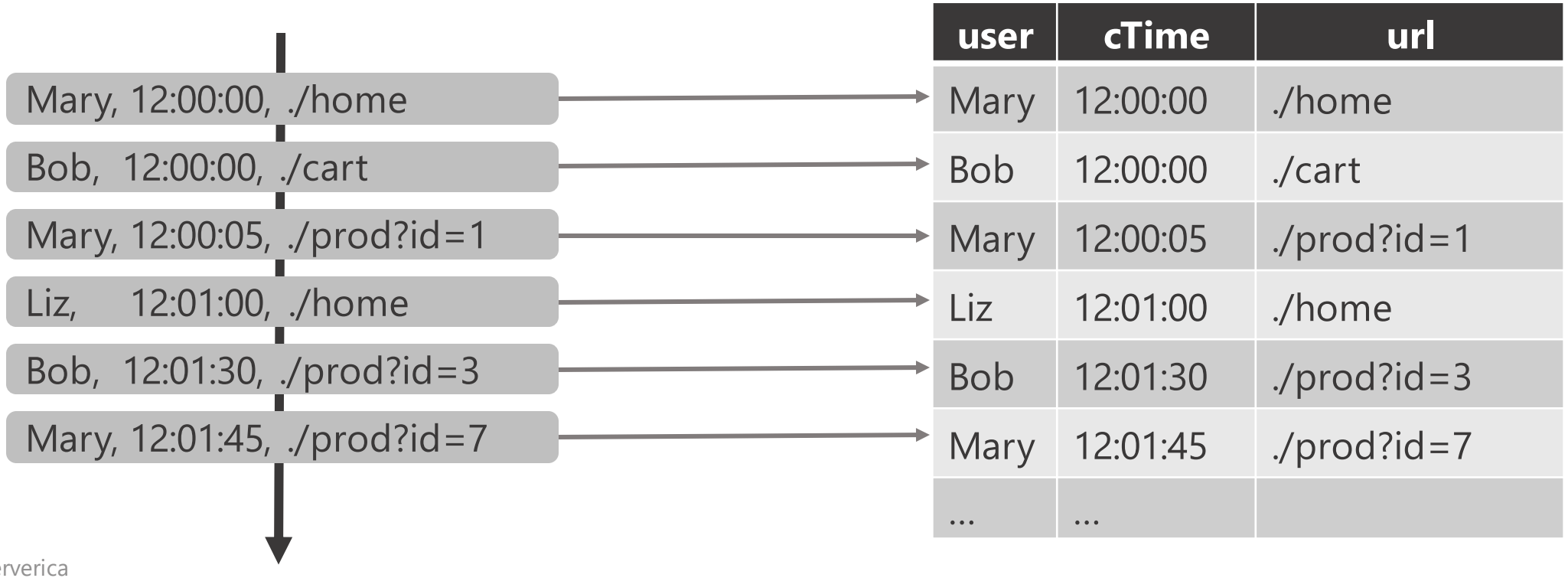
- MV maintenance is very similar to SQL on streams
  - Base table updates are a stream of DML statements
  - MV definition query is evaluated on that stream
  - MV is query result and continuously updated

Continuous Queries in Flink

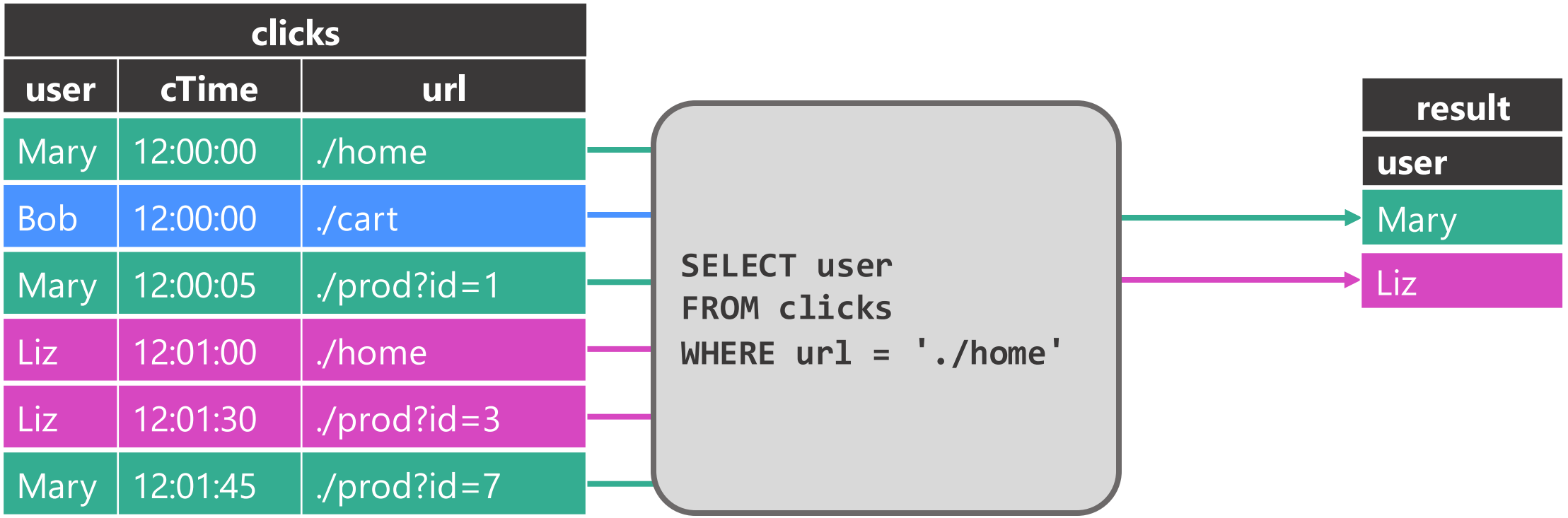
- Core concept is a *"Dynamic Table"*
  - Dynamic tables are changing over time
- Queries on dynamic tables
  - produce new dynamic tables (which are updated based on input)
  - do not terminate
- Stream ↔ Dynamic table conversions

Stream → Dynamic Table: Append

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

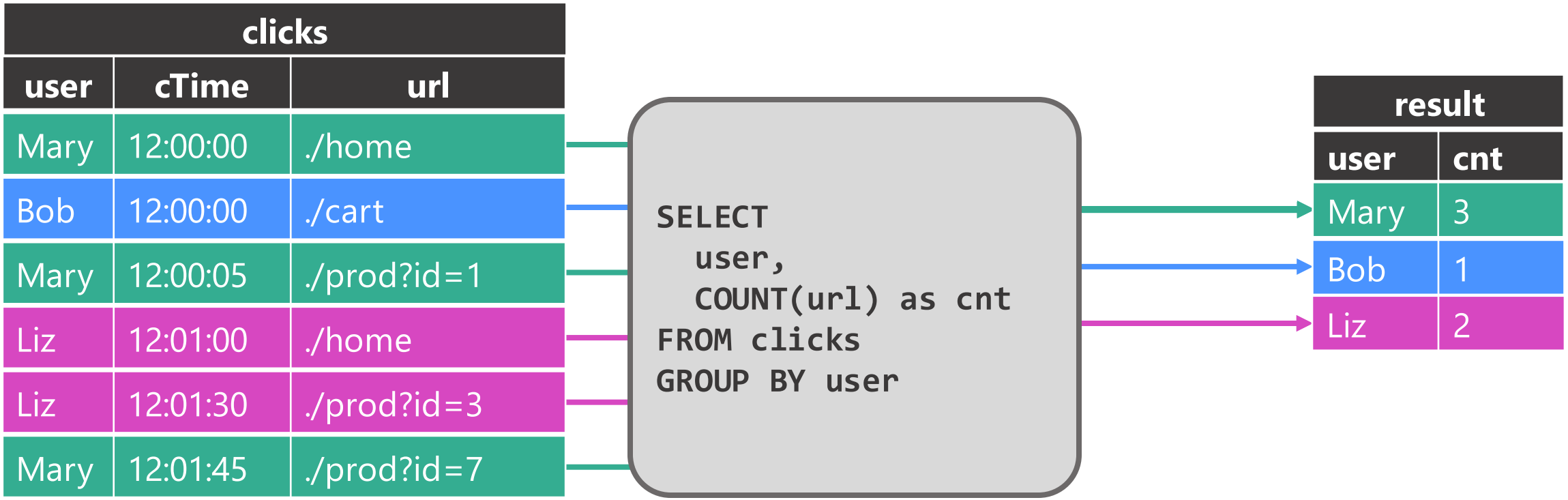


Querying a Dynamic Table



Rows of result table are appended.

Querying a Dynamic Table

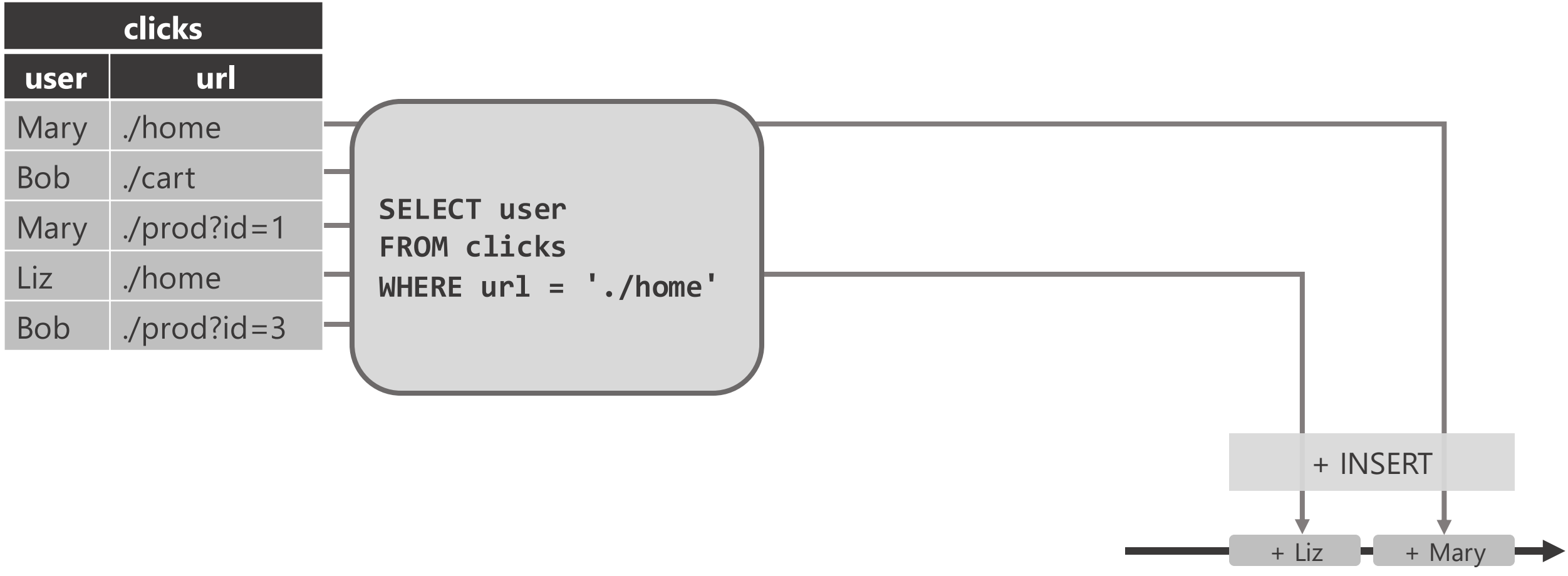


Rows of result table are updated.

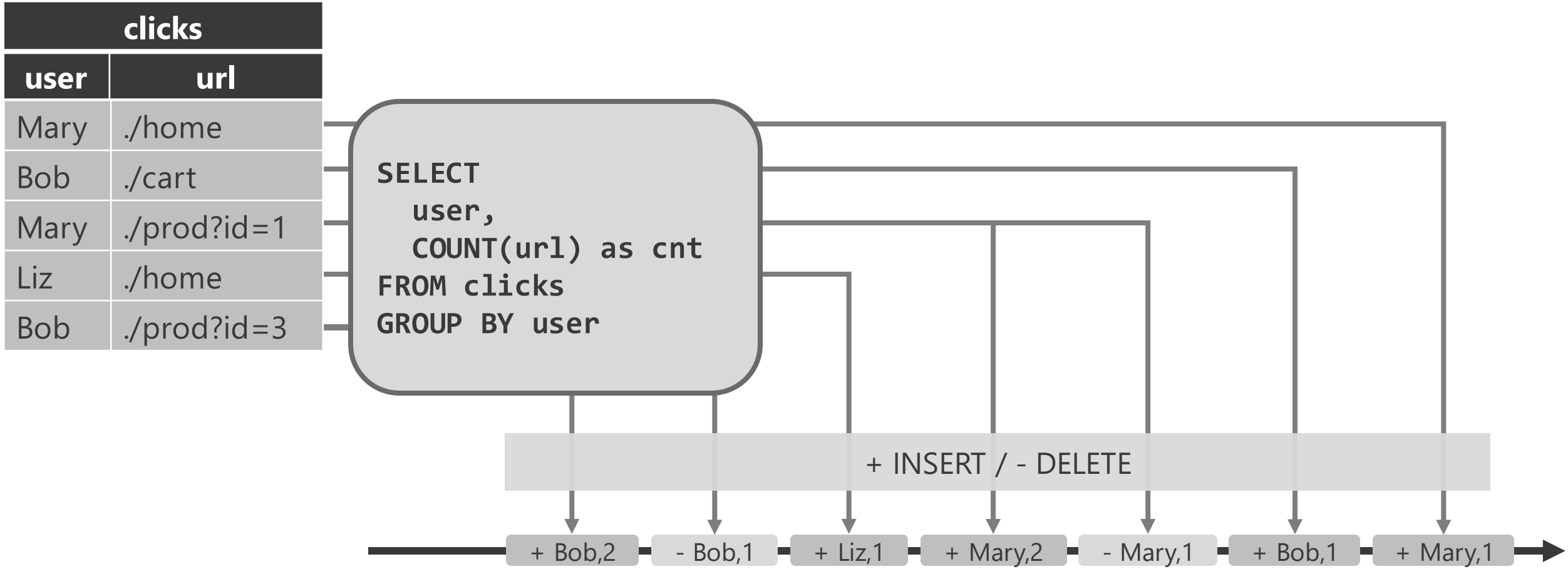
Dynamic Table → Stream

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

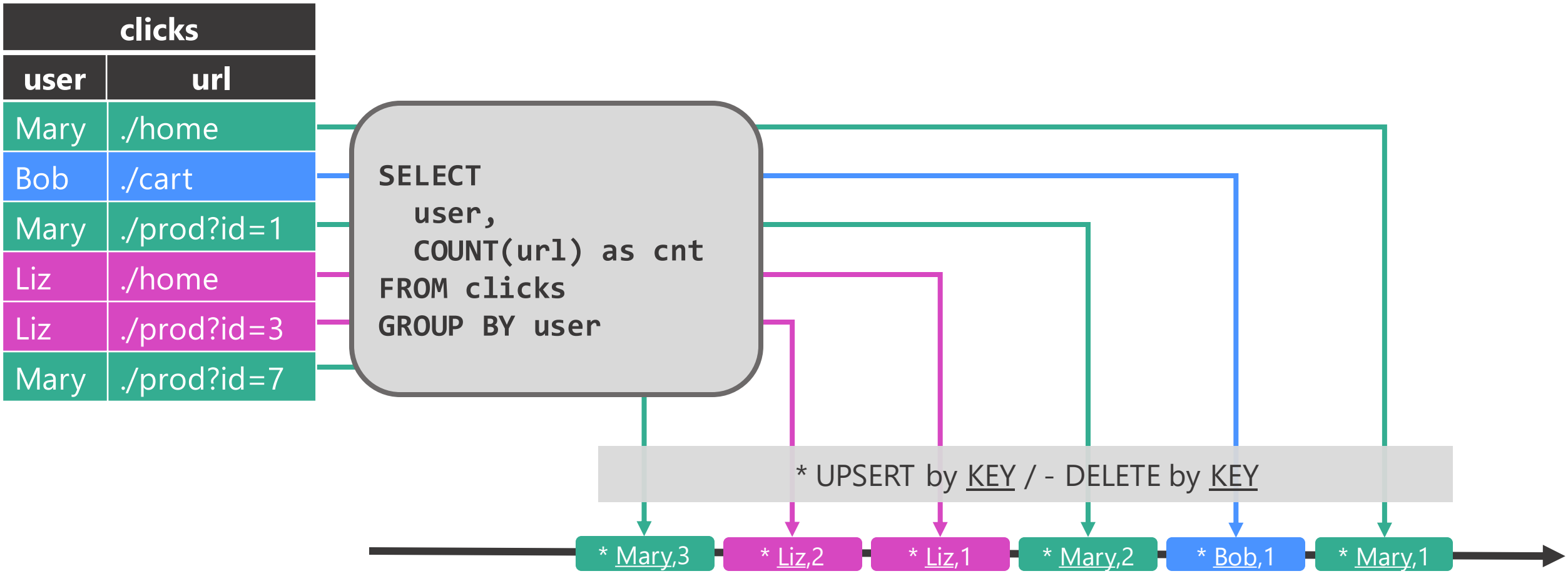
Dynamic Table → Stream: Append-only



Dynamic Table → Stream: Retraction



Dynamic Table → Stream: Upsert



Summary

- Streams are interpreted as changelog for a Table
  - Flink 1.9 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 attribute
  - Event time property (and watermark alignment) are lost when it is modified

Processing Time Attribute

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

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

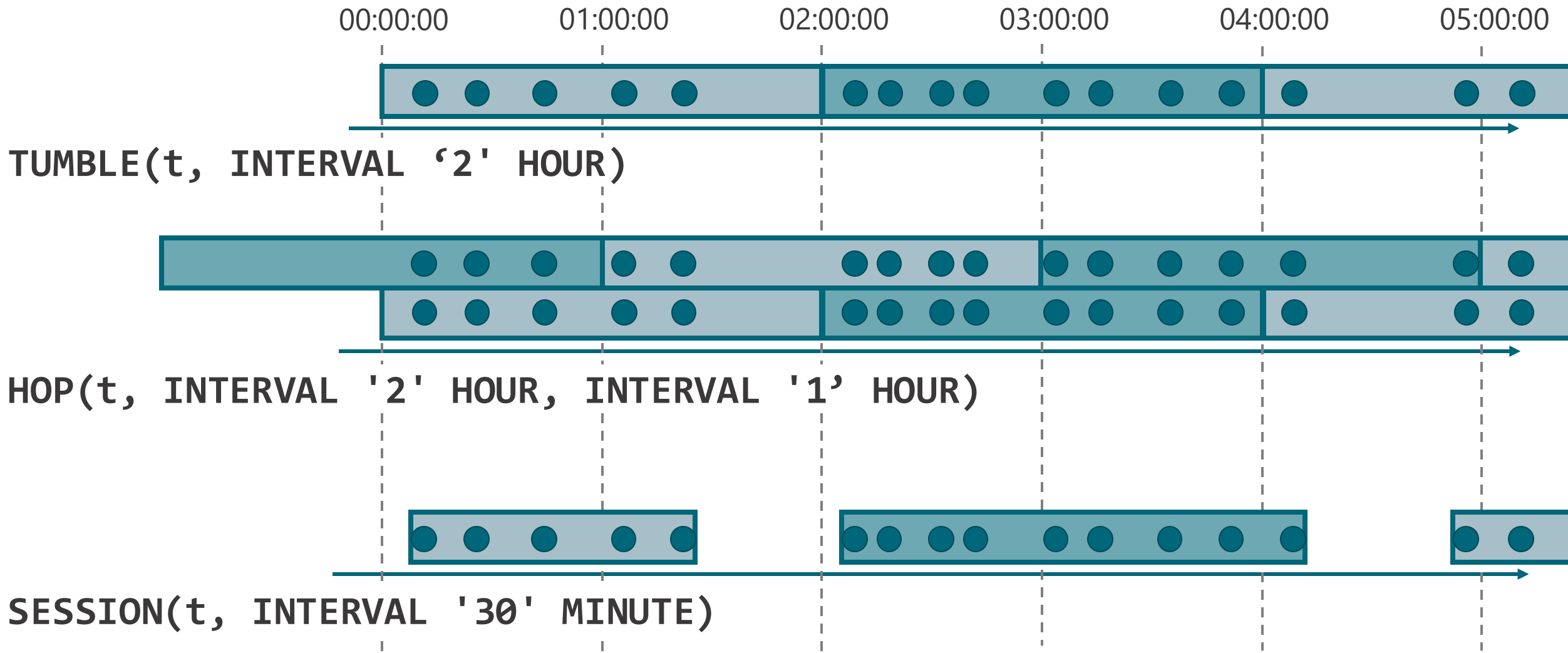
How Are Time Attributes Used?

- Some operations require time attributes in certain clauses
  - GROUP BY windows
  - OVER windows
  - Time-windowed joins
  - Joins with temporal tables

- 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 Windows



GROUP BY Window Aggregation

Compute number of clicks per hour and user

```
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

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
```

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 & 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

- The aggregation state is continuously growing
  - Unless key domain is bounded

- Joins need to materialize all input rows in state

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


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

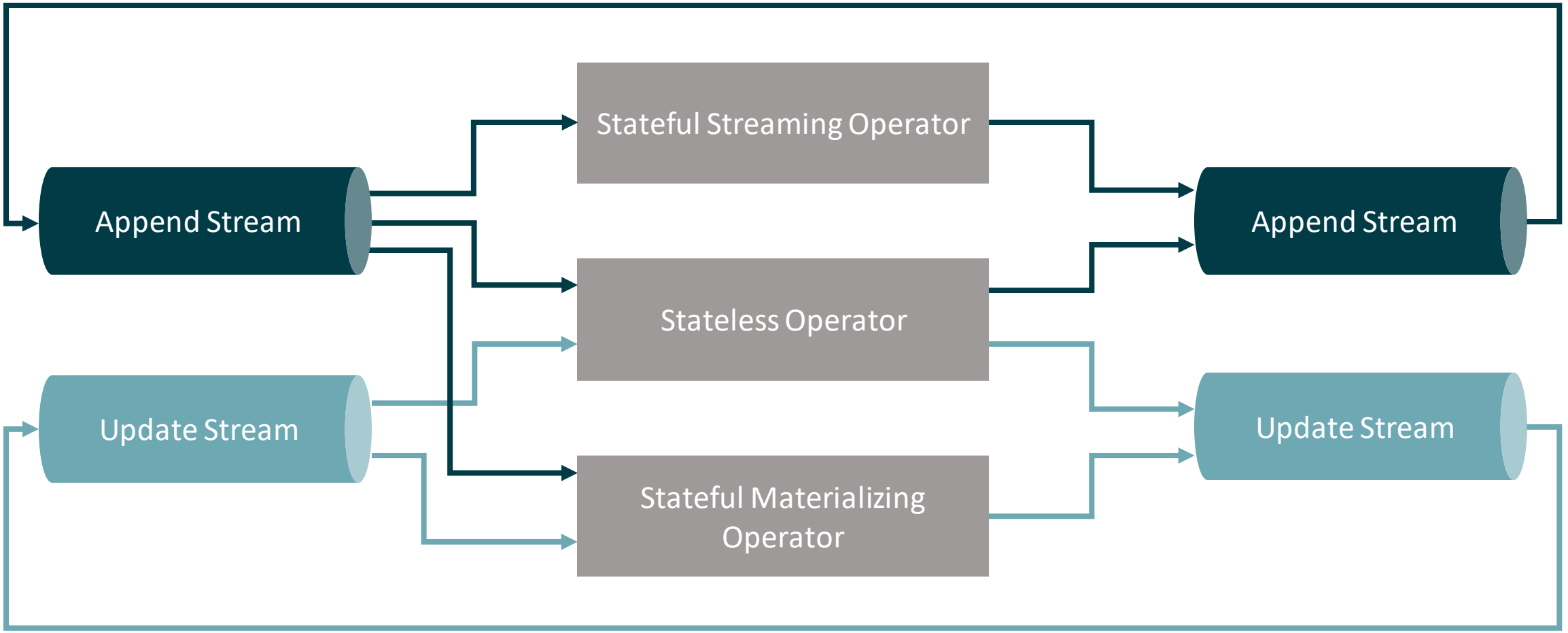
- The query result is not updated when state is removed
- Query result remains consistent if removed state is not needed again
- Query result becomes inconsistent if query needs to access state that was removed!
- Trade the accuracy of the result for size of state

Appending and Updating 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



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 Exercises

Querying Dynamic Tables with SQL

Continue with the hands-on exercises in
“Querying Dynamic Tables with SQL”

https://github.com/ververica/sql-training/wiki/Querying-Dynamic-Tables-with-SQL

We are here to help!

www.ververica.com

@VervericaData