Stream Analytics with SQL on Apache Flink®

Fabian Hueske
@fhueske

dataArtisans

Flink Forward Berlin
September, 13th 2017

dataArtisans

Original creators of Apache Flink®

Providers of
dA Platform 2, including
open source Apache Flink +
dA Application Manager

The DataStream API

- Flink's DataStream API is very expressive
  - Application logic implemented as user-defined functions
  - Windows, triggers, evictors, state, timers, async calls, …
- Many applications follow similar patterns
  - Do not require the expressiveness of the DataStream API
  - Can be specified more concisely and easily with a DSL

Q: What's the most popular DSL for data processing?

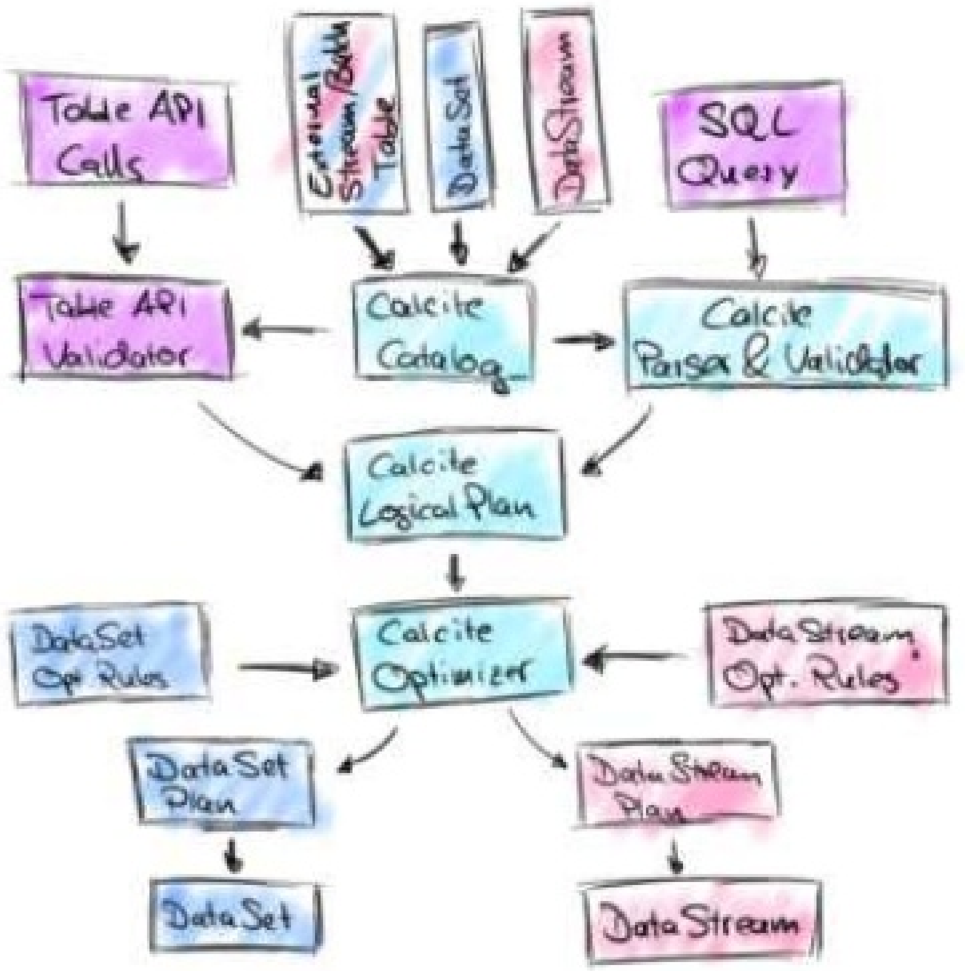
A: SQL!

Apache Flink’s relational APIs

- *Standard* SQL & LINQ-style Table API
- *Unified* APIs for batch & streaming data

A query specifies exactly the same result regardless whether its input is static batch data or streaming data.

- Common translation layers
  - Optimization based on Apache Calcite
  - Type system & code-generation
  - Table sources & sinks



Show me some code!

```
tableEnvironment
  .scan("clicks")
  .filter('url.like("https://www.xyz.com%")
  .groupBy('user)
  .select('user, 'url.count as 'cnt)
```

"clicks" can be a
- file
- database table,
- stream, …

```
SELECT user, COUNT(url) AS cnt
FROM clicks
WHERE url LIKE 'https://www.xyz.com%'
GROUP BY user
```

What if “clicks” is a file?

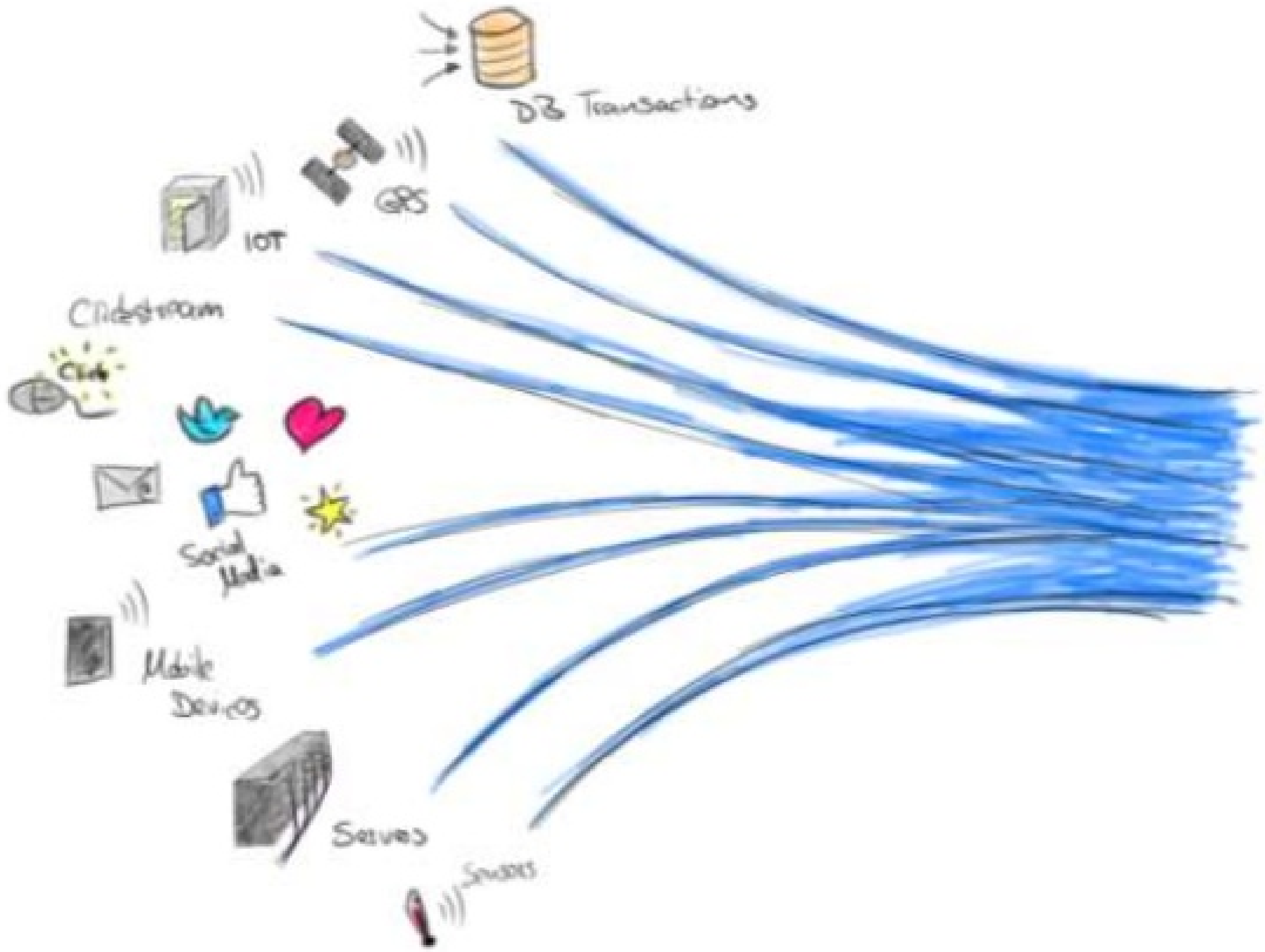
user	cTime	url
Mary	12:00:00	https://…
Bob	12:00:00	https://…
Mary	12:00:02	https://…
Liz	12:00:03	https://…

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

user	cnt
Mary	2
Bob	1
Liz	1

Q: What if we get more click data?
A: We run the query again.

What if “clicks” is a stream?



- We want the same results as for batch input!
- Can we query a stream with SQL as well?

SQL was not designed for streams

▪ Relations are bounded (multi-)sets.	↔	Streams are infinite sequences.
▪ DBMS can access all data.	↔	Streaming data arrives over time.
▪ SQL queries return a result and complete.	↔	Streaming queries continuously emit results and never complete.

DBMSs run queries on streams

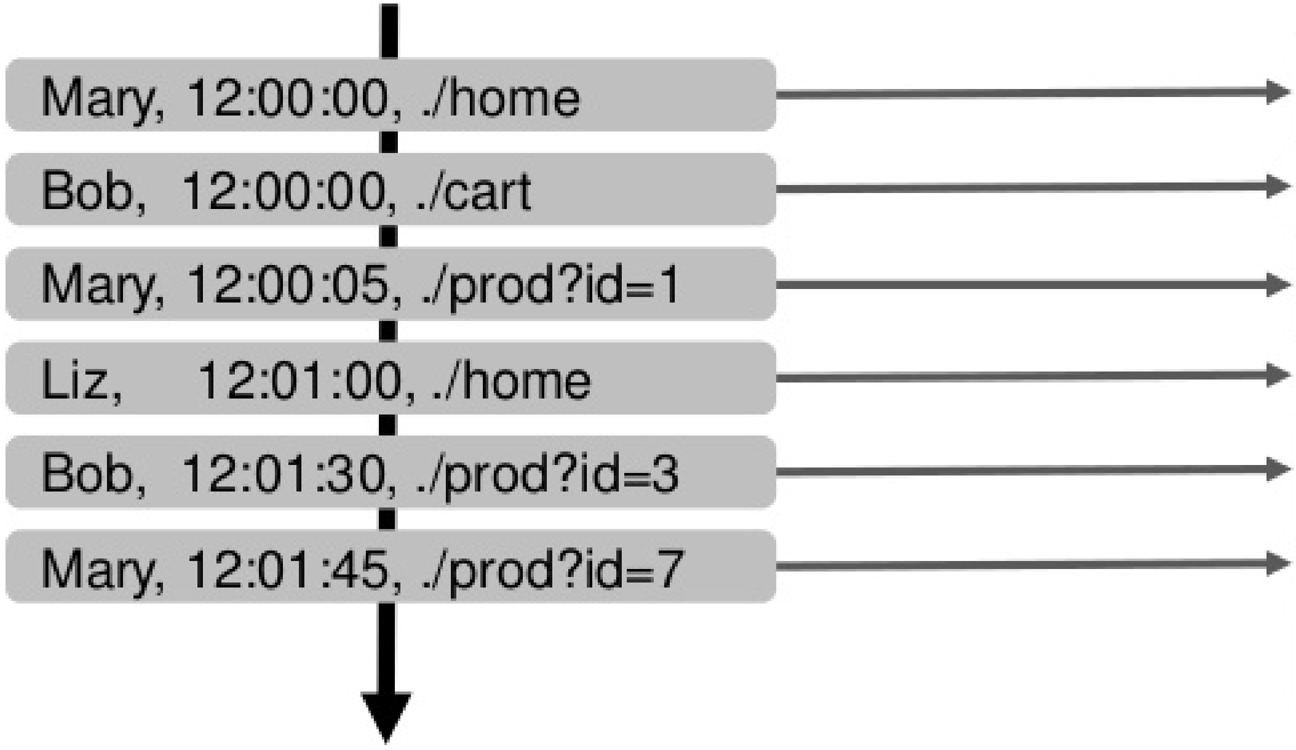
- Materialized views (MV) are similar to regular views, but persisted to disk or memory
  - Used to speed-up analytical queries
  - MVs need to be updated when the base tables change
- MV maintenance is very similar to SQL on streams
  - Base table updates are a stream of DML statements
  - MV definition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

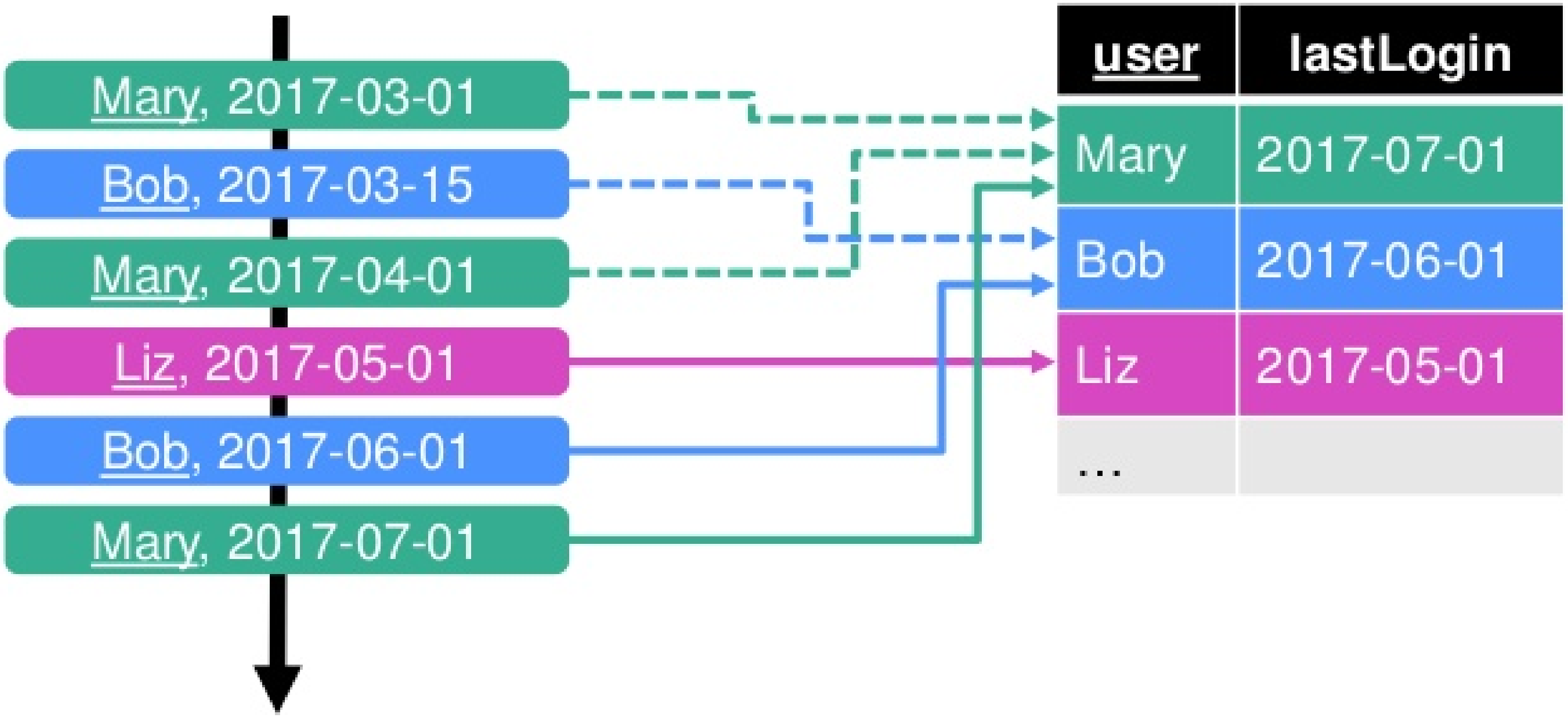
- Append mode
  - Stream records are appended to table
  - Table grows as more data arrives



user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Bob	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7
…	…	

Stream → Dynamic Table

- Upsert mode
  - Stream records have (composite) key attributes
  - Records are inserted or update existing records with same key



Querying a Dynamic Table

clicks		
user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Liz	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7

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

result	
user	cnt
u1	3
u2	1
u3	2

Rows of result table are updated.

What about windows?

```
tableEnvironment
  .scan("clicks")
  .window(Tumble over 1.hour on 'cTime as 'w)
  .groupBy('w, 'user)
  .select('user, 'w.end AS endT, 'url.count as 'cnt)
```

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

Computing window aggregates

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
u1	13:00:00	3
u2	13:00:00	1
u2	14:00:00	1
u3	14:00:00	2
u1	15:00:00	1
u2	15:00:00	2
u3	15:00:00	1

Rows are appended to result table.

Why are results always appended?

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

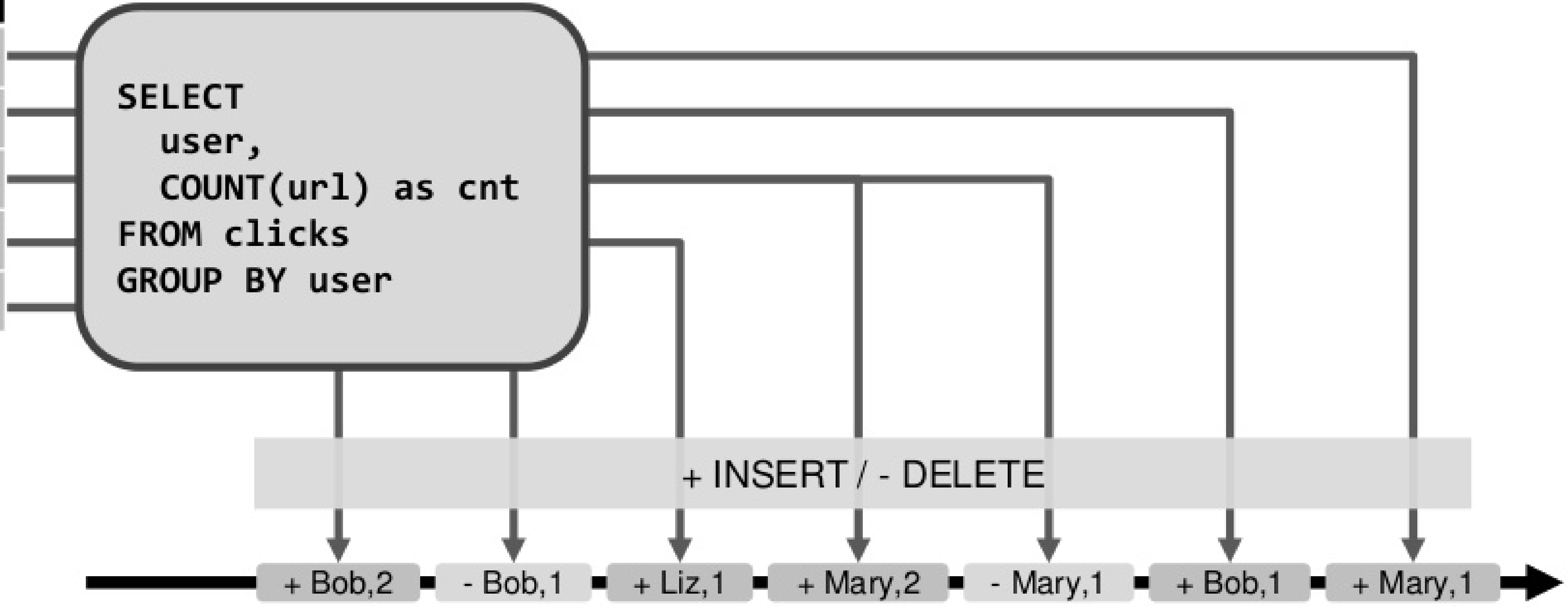
- cTime attribute is event-time attribute
  - Guarded by watermarks
  - Internally represented as special type
  - User-facing as TIMESTAMP
- Special plans for queries that operate on event-time attributes

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
- Conversion of tables to streams inspired by DBMS logs
  - DBMS use logs to restore databases (and tables)
  - REDO logs store new records to redo changes
  - UNDO logs store old records to undo changes

Dynamic Table → Stream: REDO/UNDO

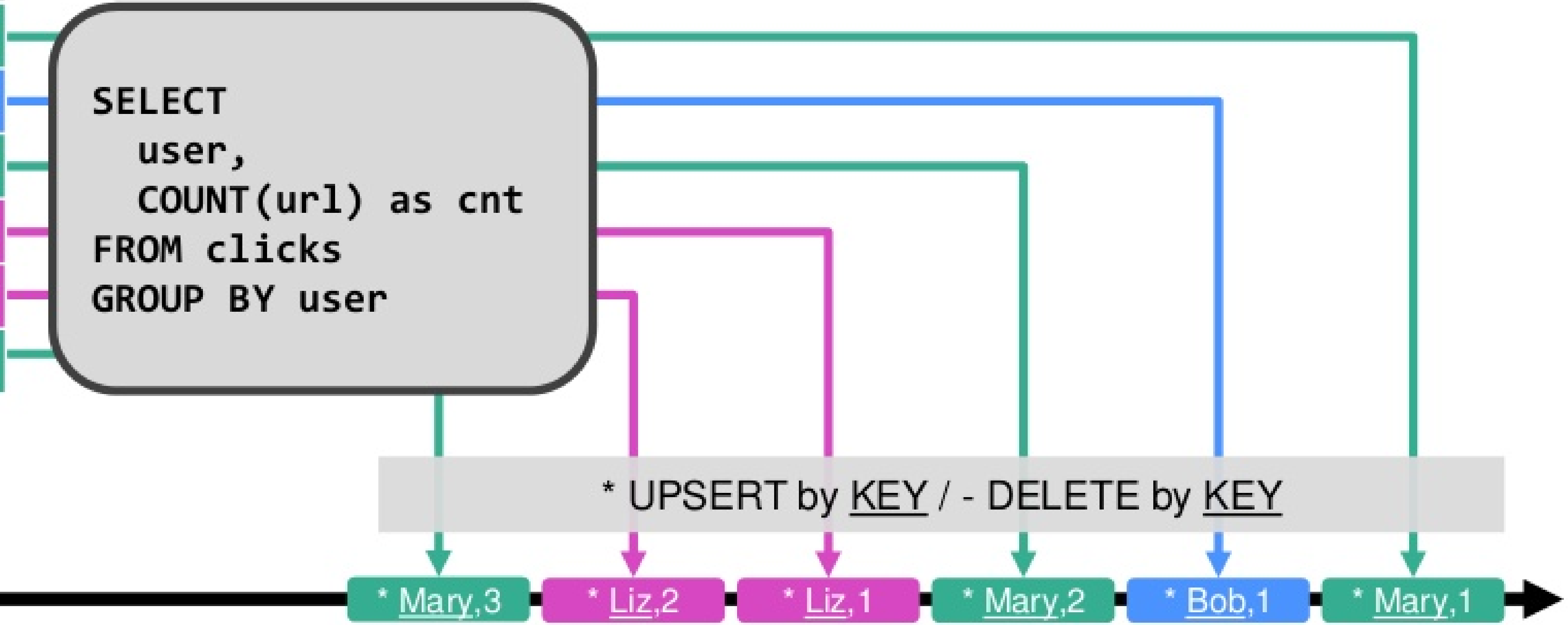
clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Bob	./prod?id=3



Dynamic Table → Stream: REDO

clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Liz	./prod?id=3
Mary	./prod?id=7

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



Can we run any query on a dynamic table?

- No, there are space and computation constraints ☹
- State size may not grow infinitely as more data arrives

```
SELECT sessionId, COUNT(url) FROM clicks GROUP BY sessionId;
```

- 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;
```

Bounding the size of query state

- Adapt the semantics of the query

```
SELECT sessionId, COUNT(url) AS cnt
FROM clicks
WHERE last(cTime, INTERVAL '1' DAY)
GROUP BY sessionId
```

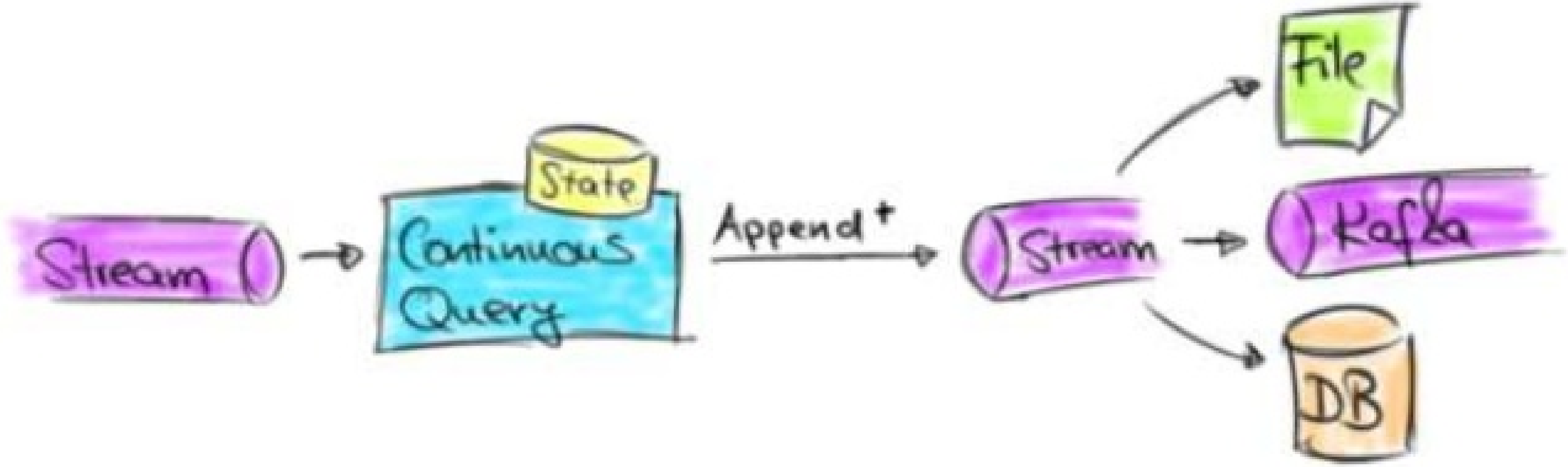
  - Aggregate data of last 24 hours. Discard older data.

- Trade the accuracy of the result for size of state
  - Remove state for keys that became inactive.

Current state of SQL & Table API

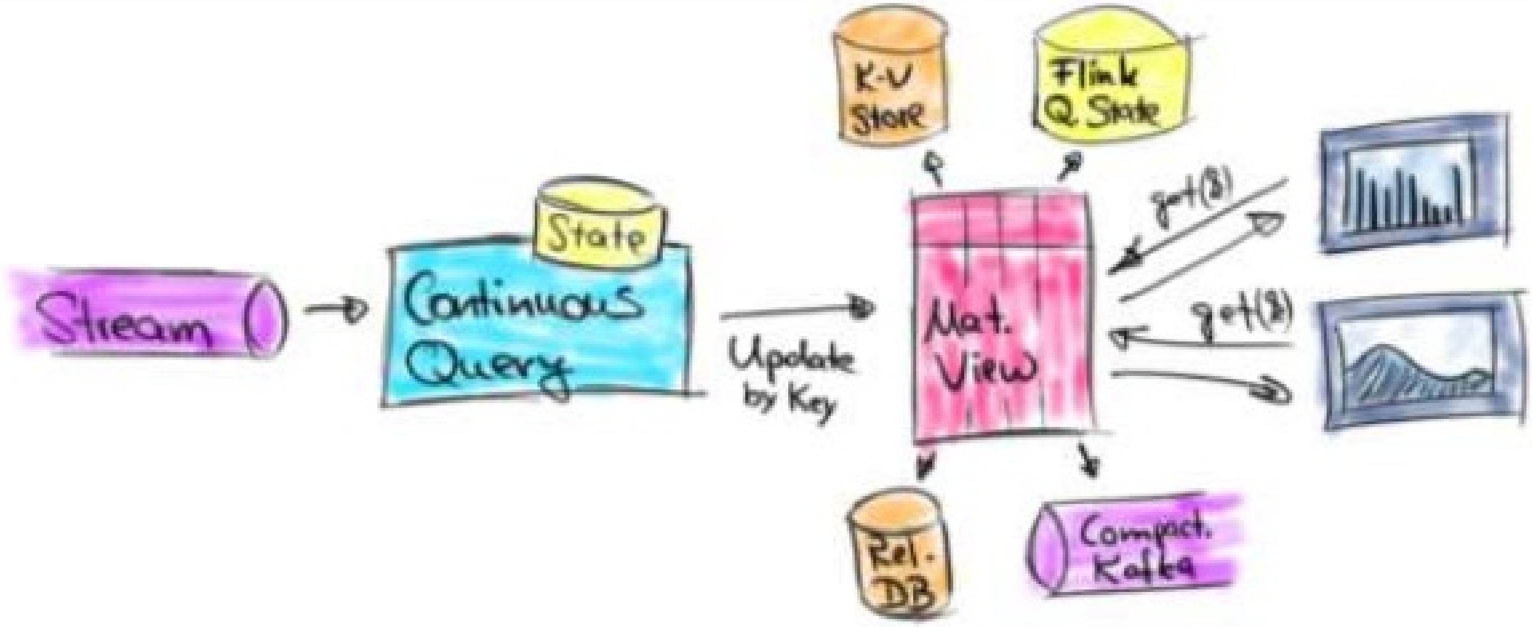
- Flink's relational APIs are rapidly evolving
  - Lots of interest by community and many contributors
  - Used in production at large scale by Alibaba and others
- Features released in Flink 1.3
  - GroupBy & Over windowed aggregates
  - Non-windowed aggregates (with update changes)
  - User-defined aggregation functions
- Features coming with Flink 1.4
  - Windowed Joins
  - Reworked connectors APIs

What can be built with this?



- Continuous ETL
  - Continuously ingest data
  - Process with transformations & window aggregates
  - Write to files (Parquet, ORC), Kafka, PostgreSQL, HBase, …

What can be built with this?



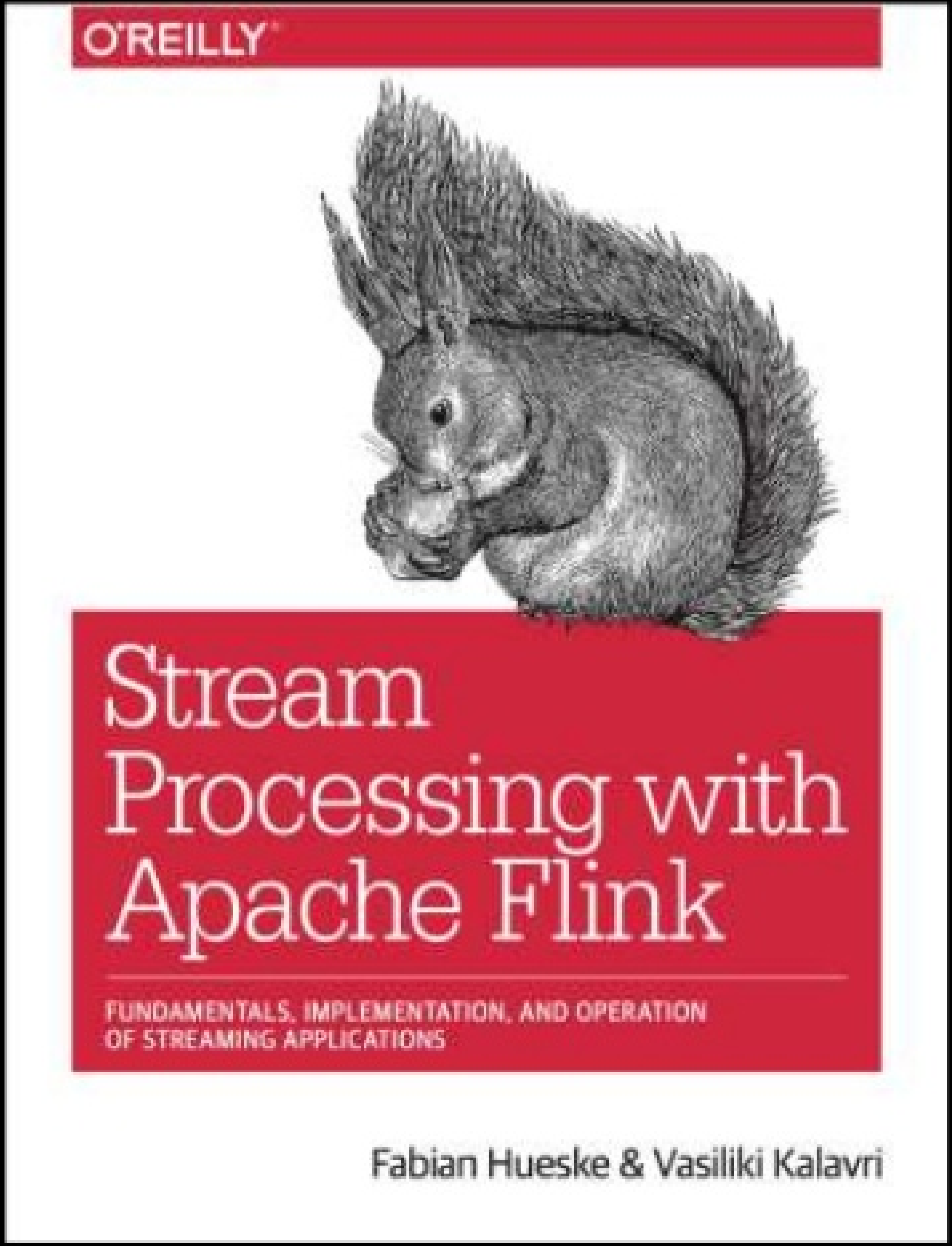
- Dashboards, reporting & event-driven architectures
  - Flink updates query results with low latency
  - Result can be written to KV store, DBMS, compacted Kafka topic
  - Maintain result table as queryable state

Wrap-up!

- Table API & SQL support many streaming use cases
  - High-level / declarative specification
  - Automatic optimization and translation
  - Efficient execution
  - Scalar, table, aggregation UDFs for flexibility

- Updating results enable many exciting applications

- Check it out!



Thank you!

@fhueske
@ApacheFlink
@dataArtisans

Available on O'Reilly Early Release!

dataArtisans

We are hiring!

data-artisans.com/careers

Tables are materialized streams

- A table is the materialization of a stream of modifications
  - SQL DML statements: INSERT, UPDATE, and DELETE
  - DBMSs process statements by modifying tables

```
INSERT (u1, Mary, "2017-03-01")
INSERT (u2, Peter, "2017-05-01")
UPDATE (lastLogin = "2017-06-01")
  WHERE (user = u1)
DELETE WHERE (user = u2)
```

user	name	lastLogin
u1	Mary	2017-06-01
u2	Peter	2017-05-01