Apache Flink® SQL in Action

Fabian Hueske – Software Engineer

About Me

- Apache Flink PMC member & ASF member
  - Contributing since day 1 at TU Berlin
  - Focusing on Flink's relational APIs since ~3.5 years
- Co-author of "Stream Processing with Apache Flink"
  - Expected release: May 2, 2019!
- Co-founder of data Artisans (now Ververica)

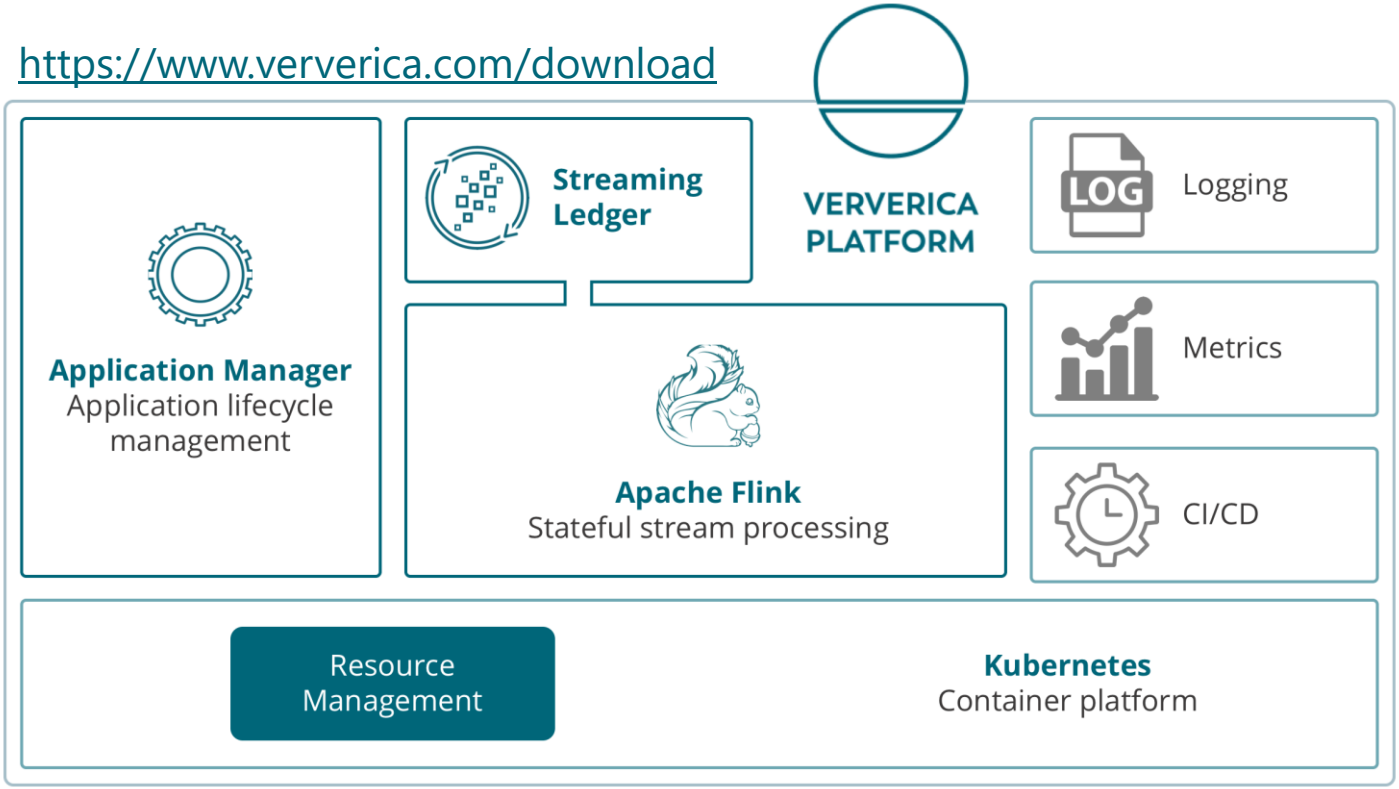
About Ververica

Original creators of
Apache Flink®

Complete Stream
Processing Infrastructure

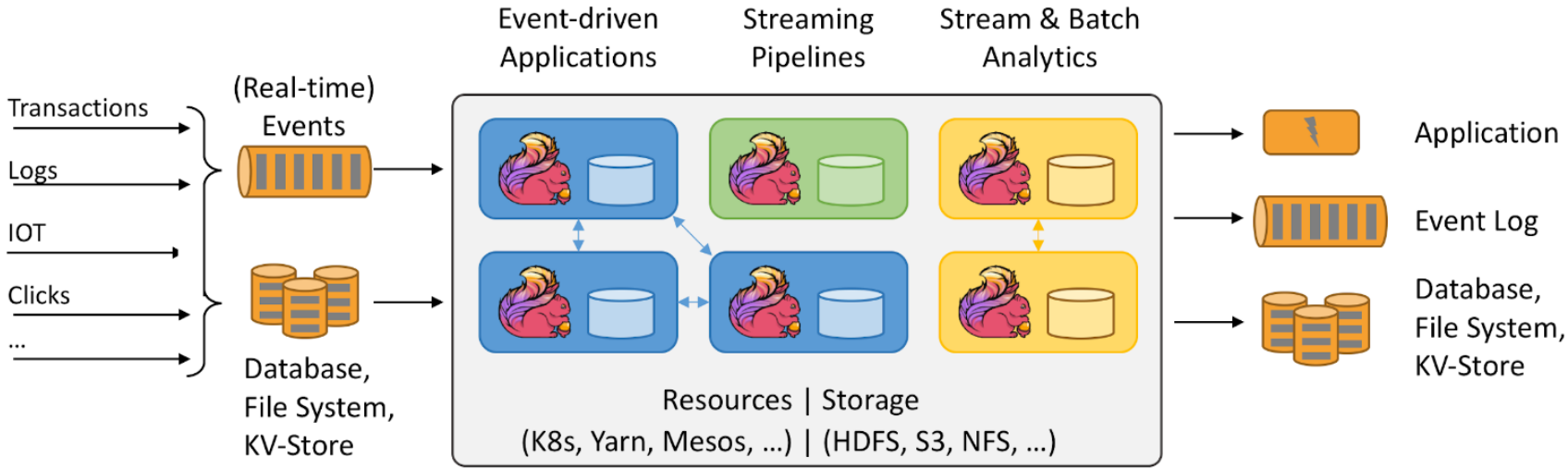
Ververica Platform

https://www.ververica.com/download



What is Apache Flink?

Stateful computations over streams
real-time and historic
fast, scalable, fault tolerant, in-memory
event time, large state, exactly-once



Hardened at Scale

UBER

Streaming Platform Service
billions messages per day
A lot of Stream SQL

NETFLIX

Streaming Platform as a Service
3700+ container running Flink,
1400+ nodes, 22k+ cores, 100s of jobs,
3 trillion events / day, 20 TB state

1000s jobs, 100.000s cores,
10 TBs state, metrics, analytics,
real time ML,
Streaming SQL as a platform

Fraud detection
Streaming Analytics Platform

Powered by Apache Flink

Flink's Powerful Abstractions

Layered abstractions to
navigate simple to complex use cases

High-level
Analytics API

SQL / Table API (dynamic tables)

```
SELECT room, TUMBLE_END(rowtime, INTERVAL '1' HOUR), AVG(temp)
FROM sensors
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

Stream- & Batch
Data Processing

DataStream API (streams, windows)

```
val stats = stream
    .keyBy("sensor")
    .timeWindow(Time.seconds(5))
    .sum((a, b) -> a.add(b))
```

Stateful Event-
Driven Applications

Process Function (events, state, time)

```
def processElement(event: MyEvent, ctx: Context, out: Collector[Result]) = {
  // work with event and state
  (event, state.value) match { … }

  out.collect(…) // emit events
  state.update(…) // modify state

  // schedule a timer callback
  ctx.timerService.registerEventTimeTimer(event.timestamp + 500)
}
```


Flink's Relational APIs

ANSI SQL

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

LINQ-style Table API

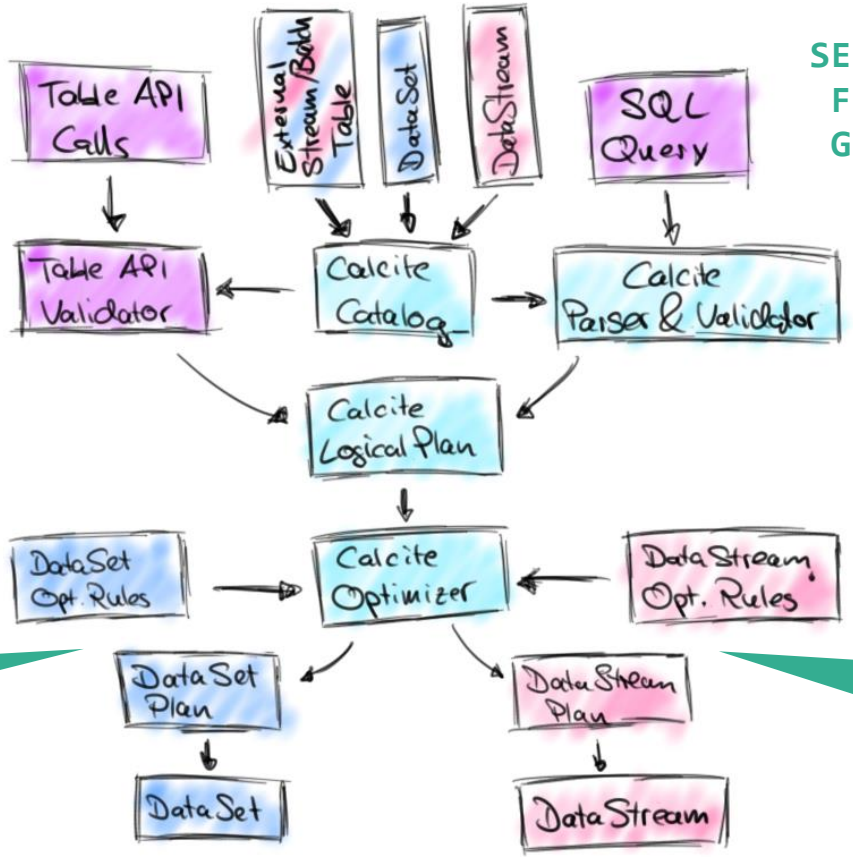
```
tableEnvironment
  .scan("clicks")
  .groupBy('user)
  .select('user, 'url.count as 'cnt)
```

Unified APIs for batch & streaming data

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

Query Translation

```
tableEnvironment
  .scan("clicks")
  .groupBy('user)
  .select('user, 'url.count)
```

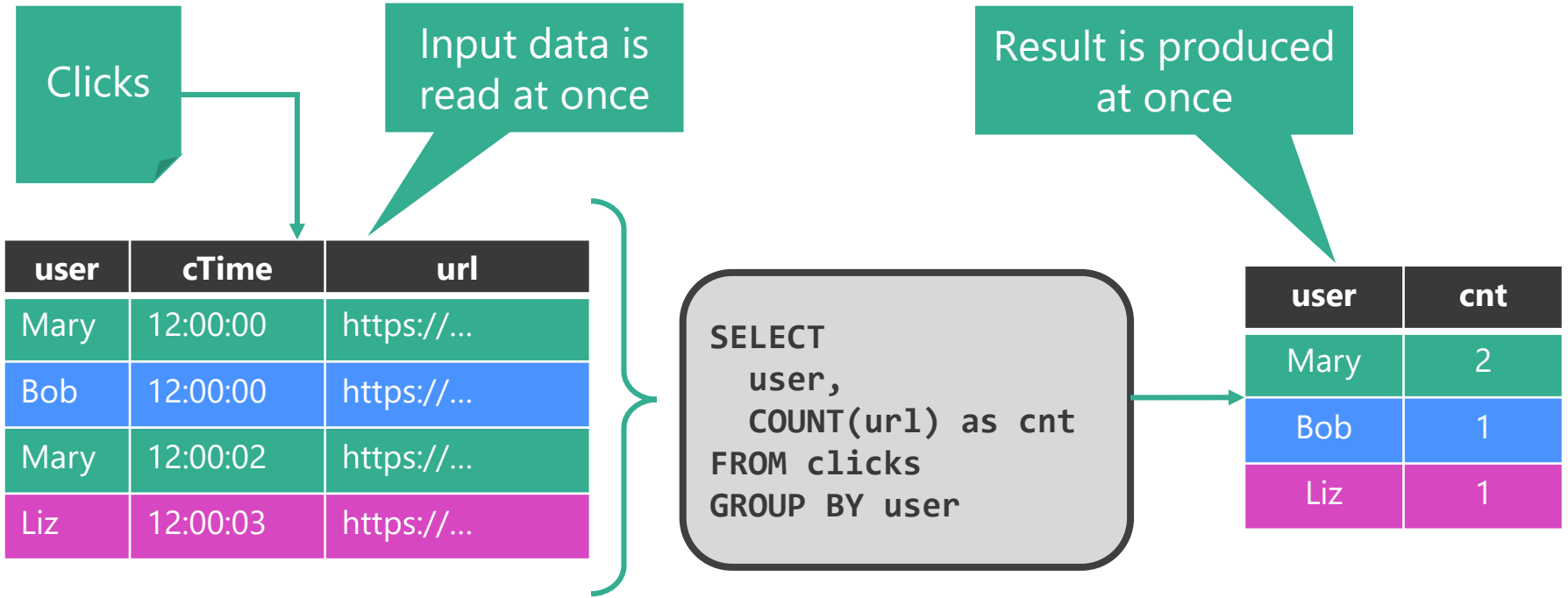



```
SELECT user, COUNT(url)
 FROM clicks
 GROUP BY user
```

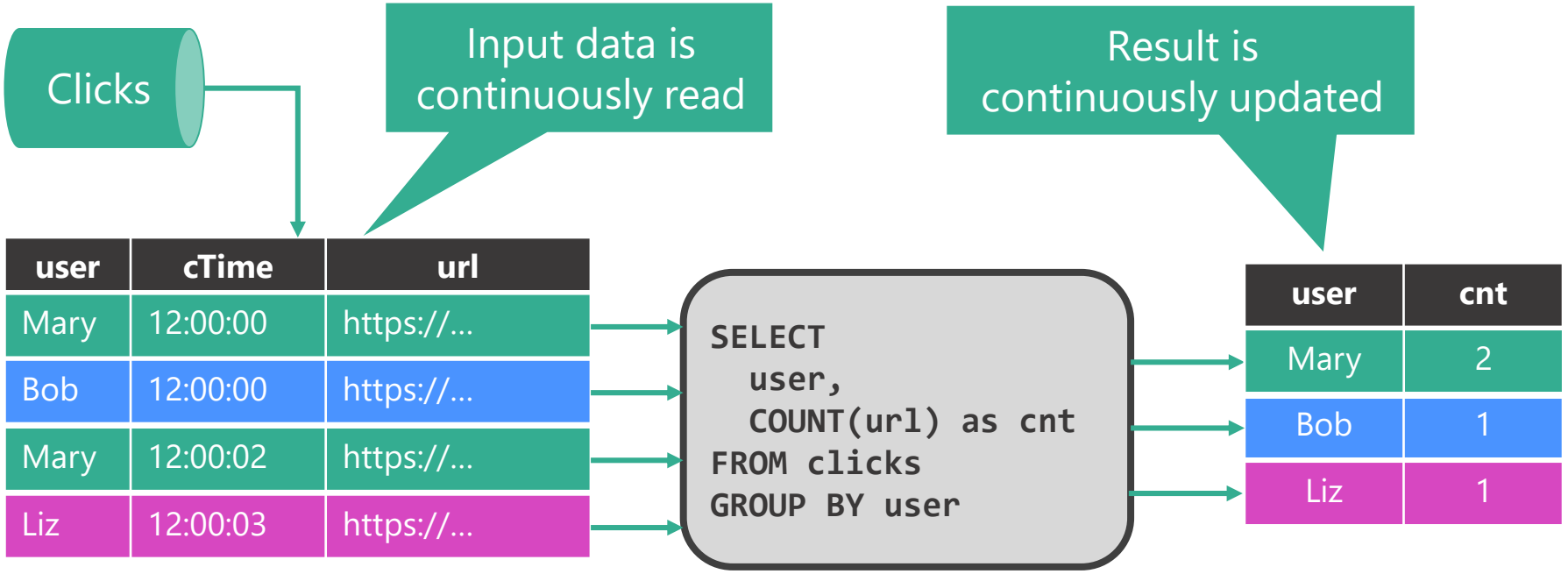
Input data is **bounded** (batch)

Input data is **unbounded** (streaming)

What if “Clicks” is a File?



What if “Clicks” is a Stream?

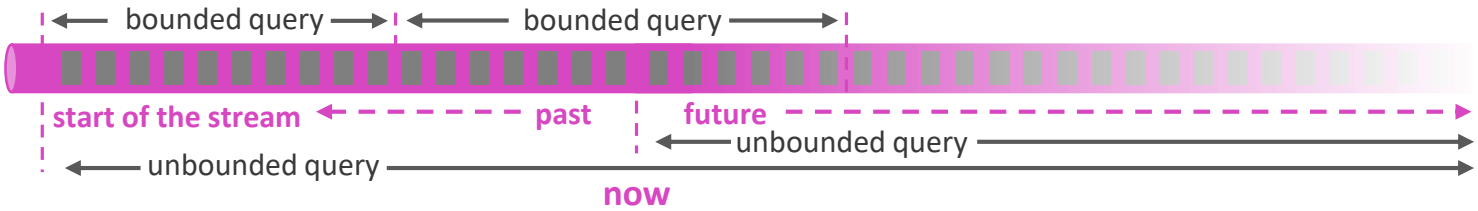


The result is the same!

Why is Stream-Batch Unification Important?

- Usability
  - ANSI SQL syntax: No custom “StreamSQL” syntax.
  - ANSI SQL semantics: No stream-specific result semantics.

- Portability
  - Run the same query on *bounded* & *unbounded* data
  - Run the same query on *recorded* & *real-time* data
  - *Bootstrapping* query state or *backfilling* results from historic data



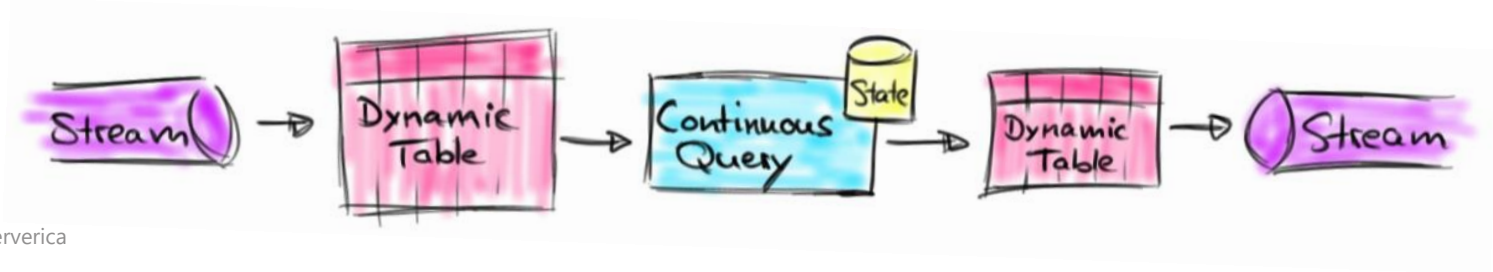
Database Systems Run Queries on Streams

- 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 Conversions

- A stream is the changelog of a dynamic table
  - As change messages are ingested from a stream, a table evolves
  - As a table evolves, change messages are emitted to a stream

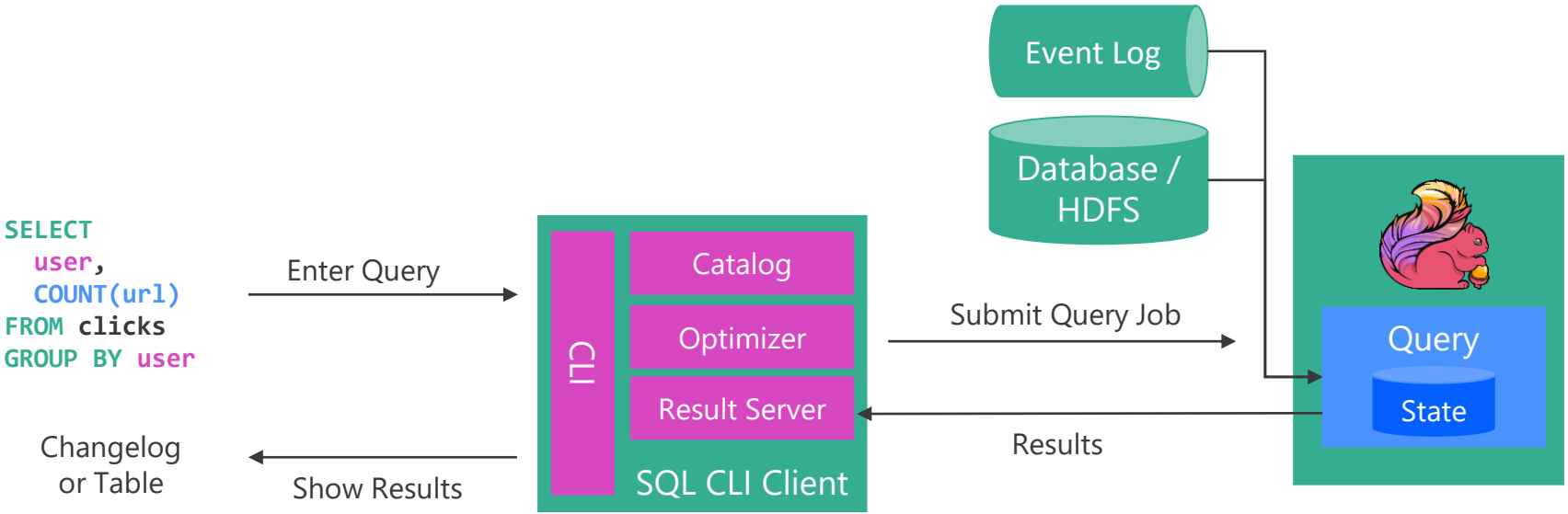
- Different changelog interpretations
  - Append-only change messages
  - Upsert change messages
  - Add/Retract change messages

How Can I Use It?

- Embed SQL queries in regular Flink applications
  - Tight integration with DataStream and DataSet APIs
  - Mix and match with other libraries (CEP, ProcessFunction, Gelly)
  - Package and operate queries like any other Flink application

- Run SQL queries via Flink's SQL CLI Client
  - Interactive mode: Submit query and inspect results
  - Detached mode: Submit query and write results to sink system

SQL CLI Client – Interactive Queries



The New York Taxi Rides Data Set

- A public data set about taxi rides in New York City
- Rides are ingested as append-only (streaming) table
  - Each ride is represented by a start and an end event
- Table: **`Rides`**

```
rideId:    BIGINT     // ID of the taxi ride
taxiId:    BIGINT     // ID of the taxi
isStart:   BOOLEAN    // flag for pick-up (true) or drop-off (false) event
lon:       DOUBLE     // longitude of pick-up or drop-off location
lat:       DOUBLE     // latitude of pick-up or drop-off location
rideTime:  TIMESTAMP  // time of pick-up or drop-off event
psgCnt:    INT        // number of passengers
```


Compute Basic Statistics

- ***Count rides*** per ***number of passengers***.

```
SELECT
  psgCnt,
  COUNT(*) as cnt
FROM Rides
WHERE isStart
GROUP BY
  psgCnt
```


Identify Popular Pick-Up / Drop-Off Locations

- Compute ***every 5 minutes*** for ***each area*** the ***number of departing and arriving taxis***.

```
SELECT
  area,
  isStart,
  TUMBLE_END(rideTime, INTERVAL '5' MINUTE) AS cntEnd,
  COUNT(*) AS cnt
FROM (SELECT rideTime, isStart, toAreaId(lon, lat) AS area
      FROM Rides)
GROUP BY
  area,
  isStart,
  TUMBLE(rideTime, INTERVAL '5' MINUTE)
```


Average Tip Per Hour of Day

- Compute the ***average tip*** per ***hour of day***. Fare data is stored in a separate table ***Fares*** *that needs to be* ***joined***.

```
SELECT
  CEIL(r.rideTime TO HOUR) AS hourOfDay,
  AVG(f.tip) AS avgTip
FROM
  Rides r,
  Fares f
WHERE
  NOT r.isStart AND
  r.rideId = f.rideId AND
  f.payTime BETWEEN r.rideTime - INTERVAL '5' MINUTE AND r.rideTime
GROUP BY
  CEIL(r.rideTime TO HOUR);
```


SQL Feature Set in Flink 1.8.0

STREAMING & BATCH

- SELECT FROM WHERE
- GROUP BY [HAVING]
  - Non-windowed
  - TUMBLE, HOP, SESSION windows
- JOIN
  - Time-Windowed INNER + OUTER JOIN
  - Non-windowed INNER + OUTER JOIN
- User-Defined Functions
  - Scalar
  - Aggregation
  - Table-valued

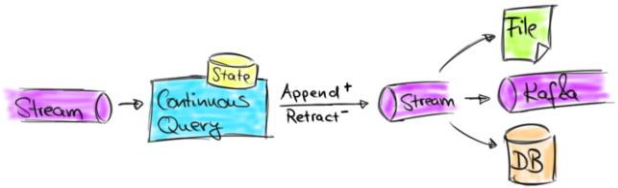
STREAMING ONLY

- OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING
- INNER JOIN with time-versioned table
- MATCH_RECOGNIZE
  - Pattern Matching/CEP (SQL:2016)

BATCH ONLY

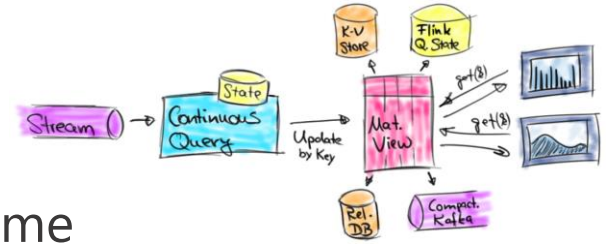
- UNION / INTERSECT / EXCEPT
- ORDER BY

What Can I Build With That?



- Data Pipelines & Low-latency ETL
  - Transform, aggregate, and move events in real-time
  - Write streams to file systems, DBMS, K-V stores, …
  - Ingest appearing files to produce streams

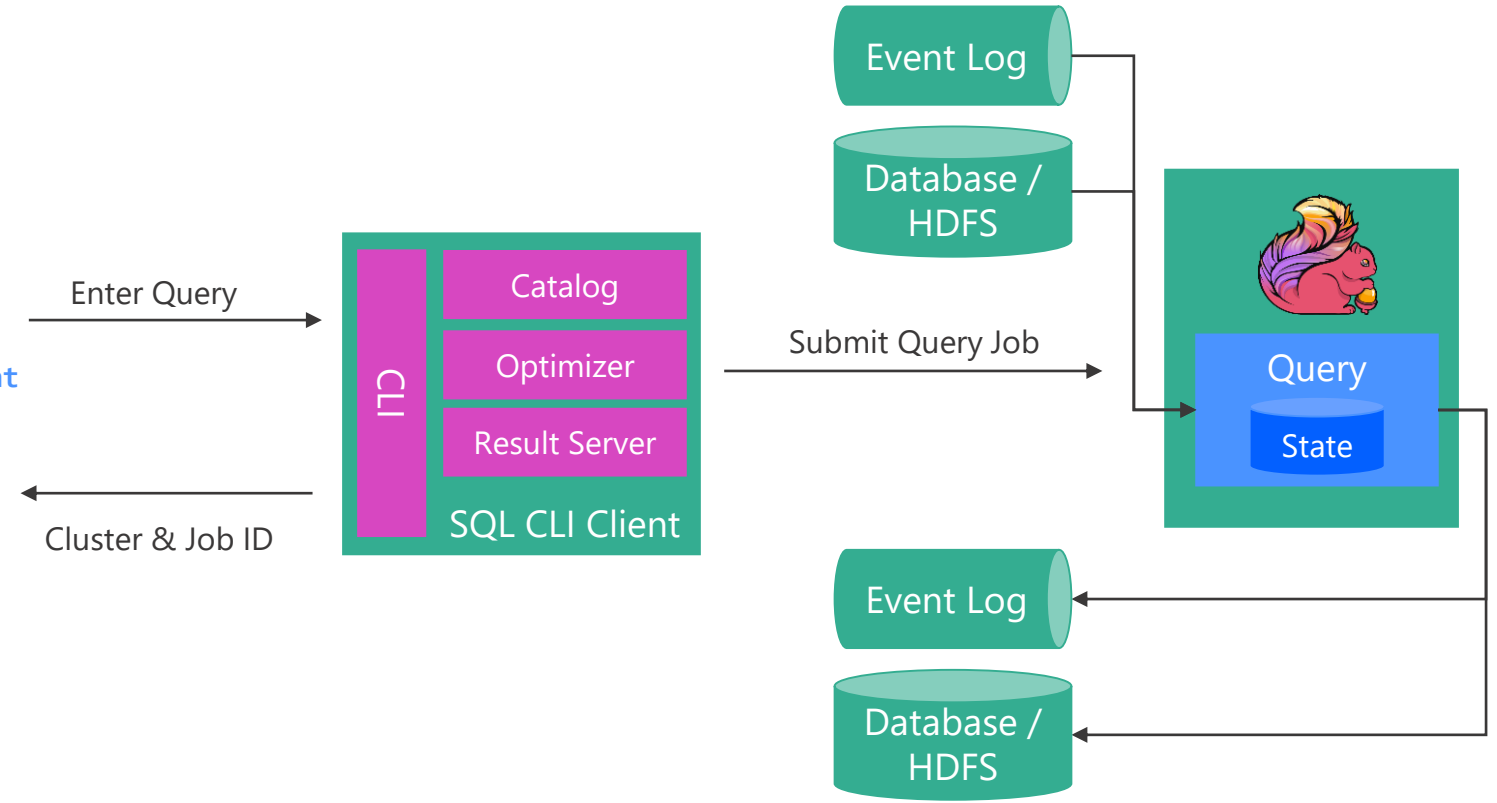
- Stream & Batch Analytics
  - Run analytical queries over bounded and unbounded data
  - Query and compare historic and real-time data



- Power Live Dashboards
  - Compute and update data to visualize in real-time

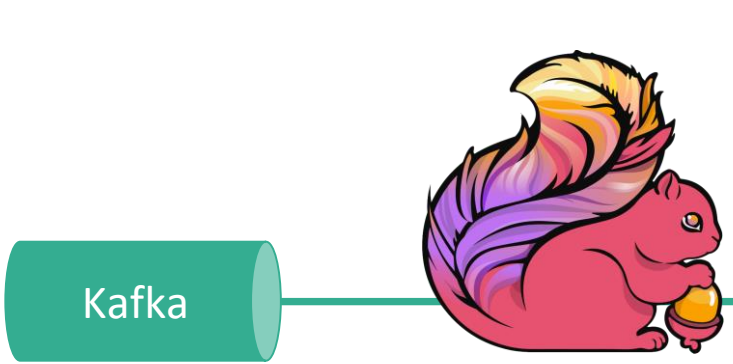
SQL CLI Client – Detached Queries

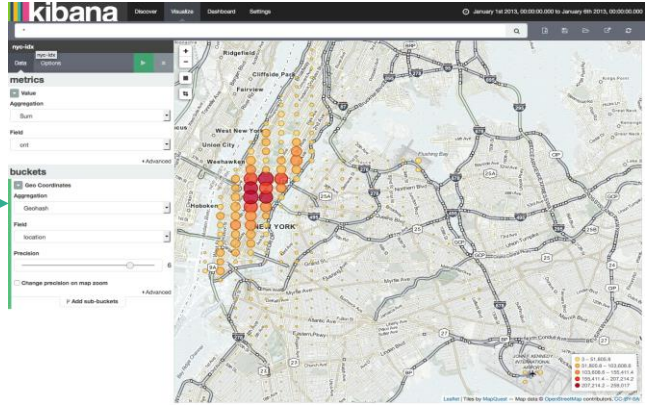
```
INSERT INTO
sinkTable
SELECT
  user,
  COUNT(url) AS cnt
FROM clicks
GROUP BY user
```



Serving a Dashboard

```
INSERT INTO AreaCnts
SELECT
  toAreaId(lon, lat) AS areaId,
  COUNT(*) AS cnt
FROM TaxiRides
WHERE isStart
GROUP BY toAreaId(lon, lat)
```





Try Flink SQL Yourself!

- The demo setup is available as a free online training
  - Slides to learn the basics
  - Exercises on streaming data

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

There's a Lot More To Come!

- Alibaba is contributing features of its Flink fork Blink back
- Major improvements for SQL and Table API
  - Better coverage of SQL features: Full TPC-DS support
  - Competitive performance: 10x compared to current state
  - Improved connectivity: External catalogs (Hive) and connectors
- Extending the scope of Table API
  - Expand support for user-defined functions
  - Add support for machine-learning learning pipelines & algorithm library

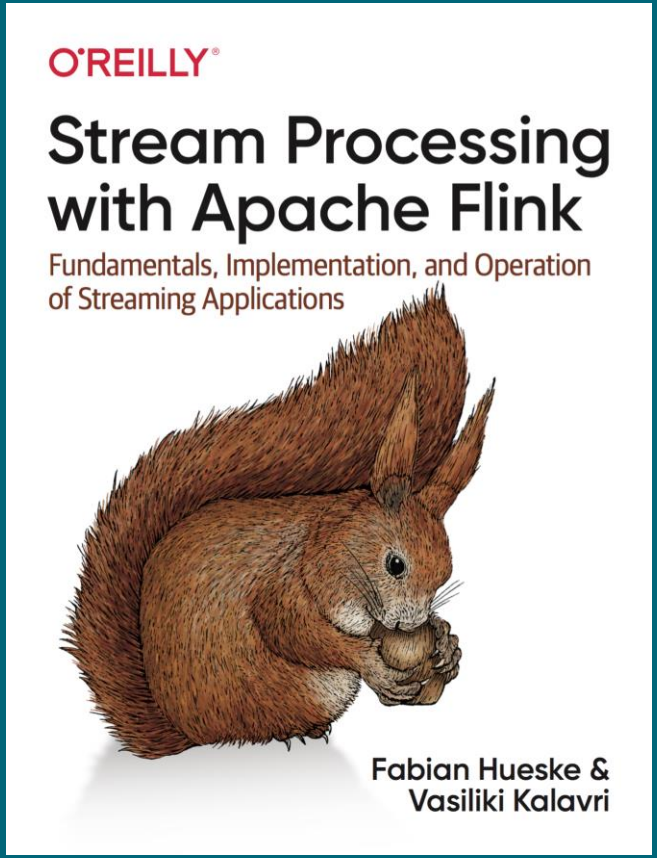
Summary

- Unification of stream and batch is important.

- Flink's SQL solves many streaming and batch use cases.
- In production at Alibaba, Huawei, Lyft, Uber, and others.
- Query deployment as application or via CLI

- Expect major improvements for batch SQL soon!


The Apache Flink® Conference
San Francisco | April 1-2, 2019
FLINK FORWARD
Organized by ververica
Use SquirrelSF19 for 15% off
flink-forward.org
#flinkforward



Available Soon!

www.ververica.com @VervericaData

Average Ride Duration Per Pick-Up Location

- ***Join start ride*** and ***end ride*** events ***on rideId*** and compute ***average ride duration per pick-up location***.

```
SELECT pickUpArea,
       AVG(timeDiff(s.rowTime, e.rowTime) / 60000) AS avgDuration
FROM (SELECT rideId, rowTime, toAreaId(lon, lat) AS pickUpArea
      FROM TaxiRides
      WHERE isStart) s
   JOIN
     (SELECT rideId, rowTime
      FROM TaxiRides
      WHERE NOT isStart) e
    ON s.rideId = e.rideId AND
       e.rowTime BETWEEN s.rowTime AND s.rowTime + INTERVAL '1' HOUR
GROUP BY pickUpArea
```