FLINK SQL IN ACTION

TIMO WALTHER, SOFTWARE ENGINEER

DATA ENG CONF, BARCELONA
SEPTEMBER 25, 2018

dataArtisans

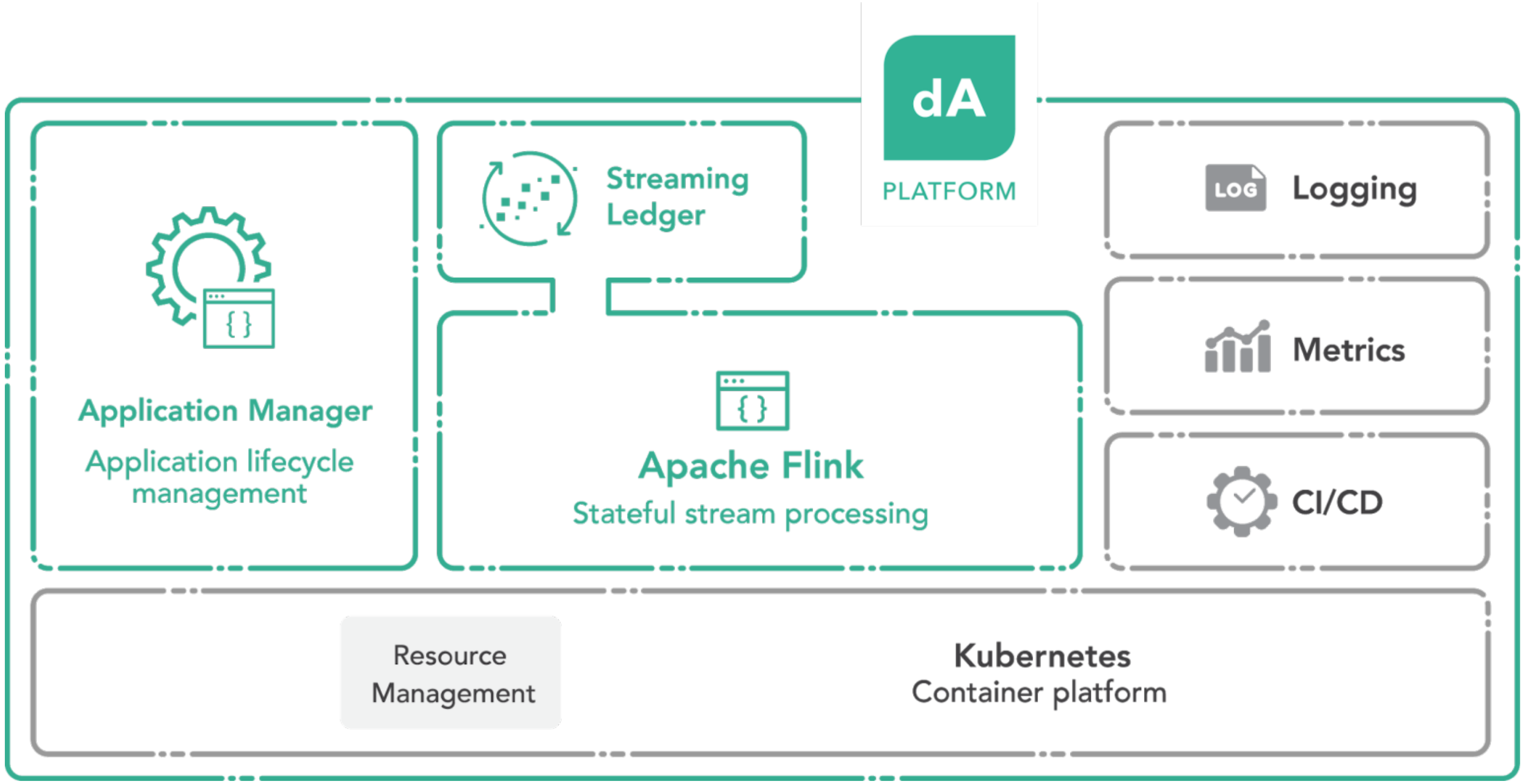
ABOUT DATA ARTISANS

PLATFORM

Original Creators of
Apache Flink®

Real-Time Stream Processing
Enterprise-Ready

DATA ARTISANS PLATFORM



data-artisans.com/download

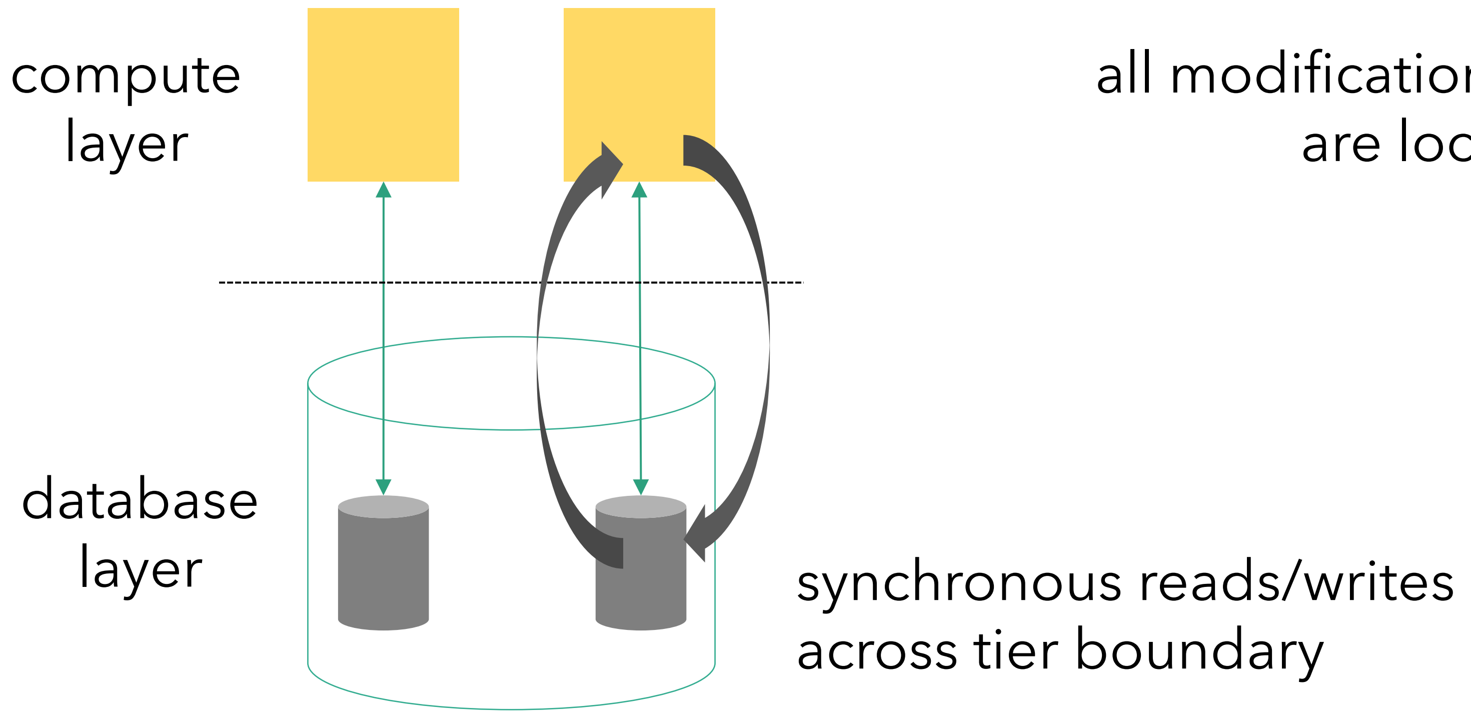
WHAT IS APACHE FLINK?

Core Building Blocks for Stream Processing

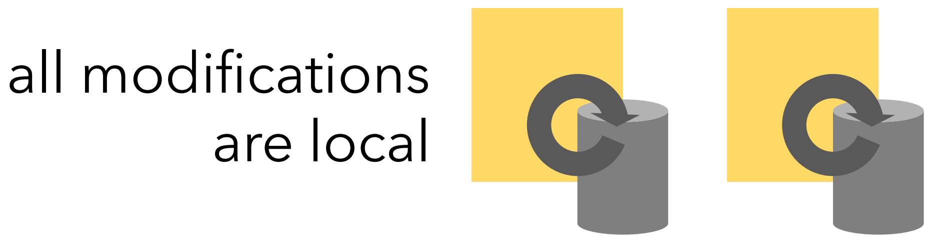
Event Streams	State	(Event) Time	Snapshots
real-time and replay	complex business logic	consistency with out-of-order data and late data	forking / versioning / time-travel

WHAT IS A STREAMING ARCHITECTURE?

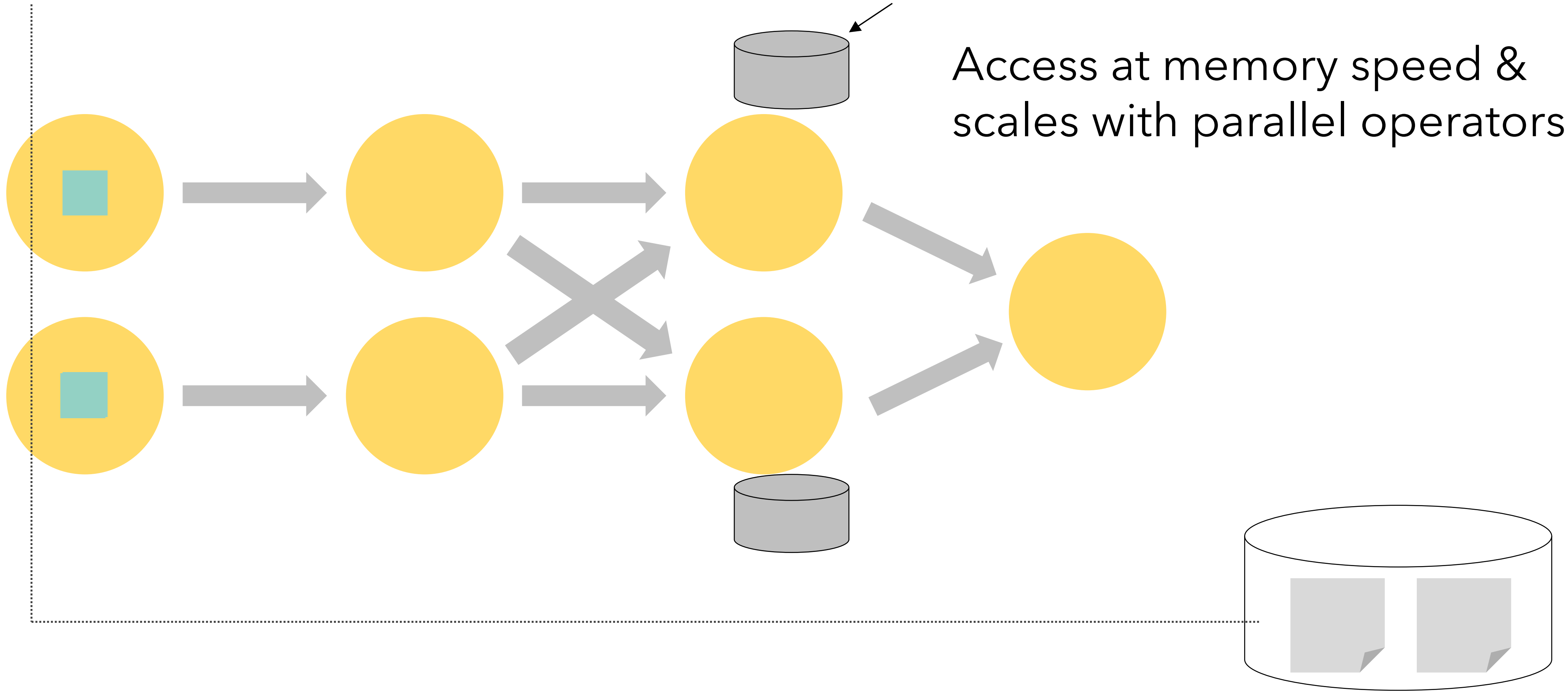
Classic tiered architecture



Streaming architecture



WHAT IS APACHE FLINK?

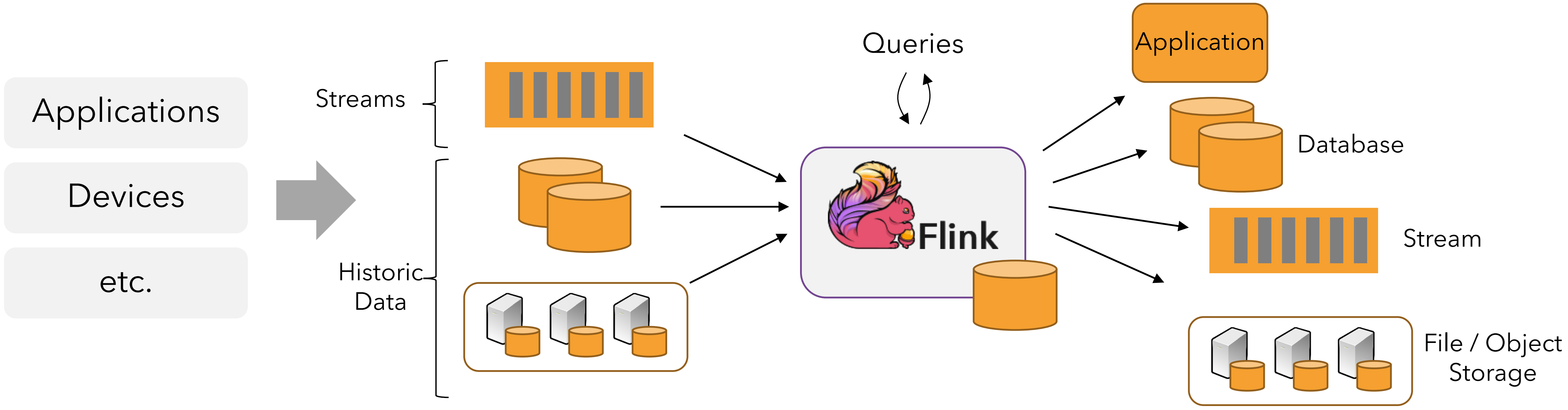


Scalable embedded state

Access at memory speed &
scales with parallel operators.

WHAT IS APACHE FLINK?

Stateful computations over streams
real-time and historic,
fast, scalable, fault tolerant,
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

Alibaba Group

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

ING

Fraud detection
Streaming Analytics Platform

POWERED BY APACHE FLINK

Alibaba Group ING NETFLIX UBER HUAWEI COMCAST

King CISCO Microsoft Expedia ebay CapitalOne

DELLEMC yelp lyft here D TRB zalando

MediaMath amadeus ROVIO Telefonica bouygues TELECOM Lightbend

airbnb SK telecom OVH.com criteo BetterCloud otto group

FLINK SQL

dataArtisans

FLINK'S POWERFUL ABSTRACTIONS

Layered abstractions to navigate simple to complex use cases

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

High-level Analytics API — **SQL / Table API (dynamic tables)**

Stream- & Batch Data Processing — **DataStream API (streams, windows)**

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

Stateful Event-Driven Applications — **Process Function (events, state, time)**

```scala
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)
}
```

APACHE 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)
```

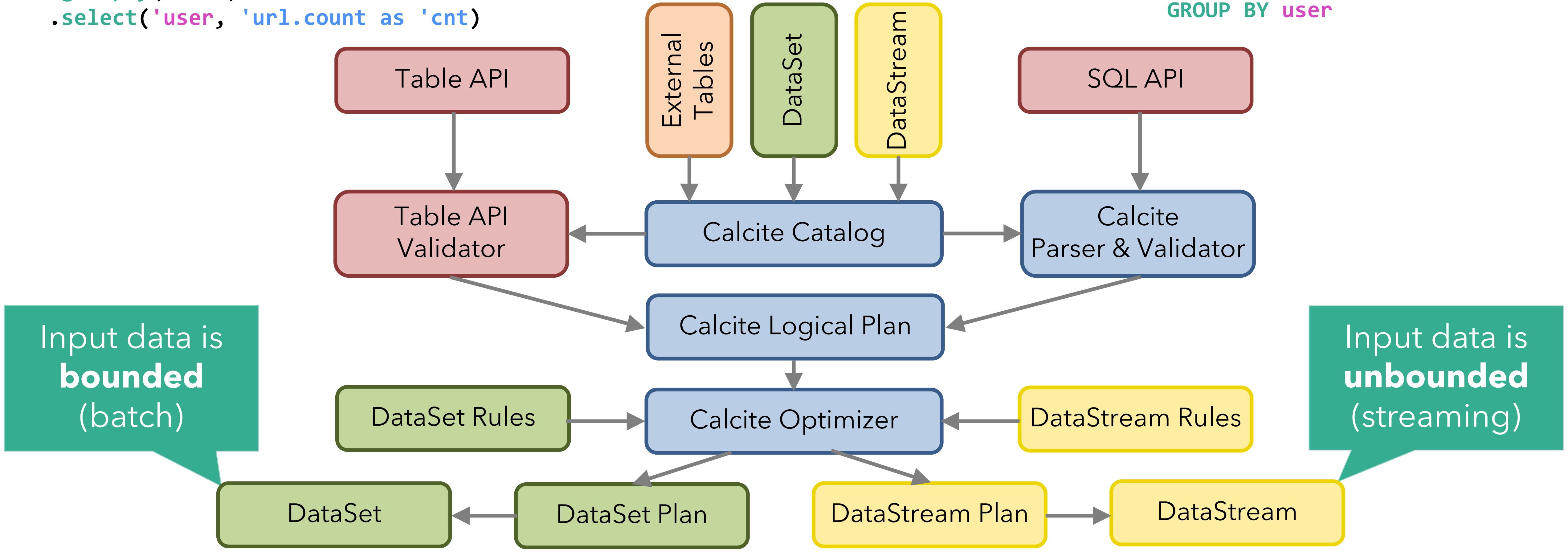
Unified APIs for batch & streaming data

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

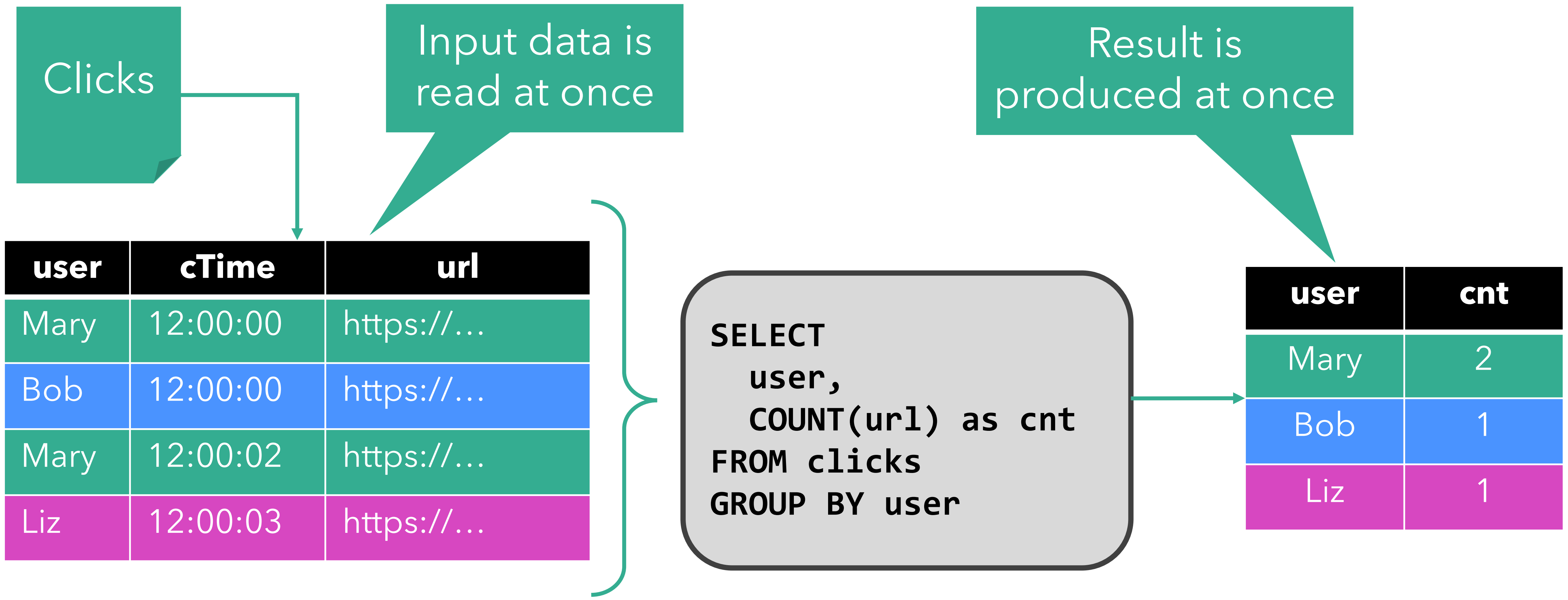
QUERY TRANSLATION

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

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



WHAT IF "CLICKS" IS A FILE?

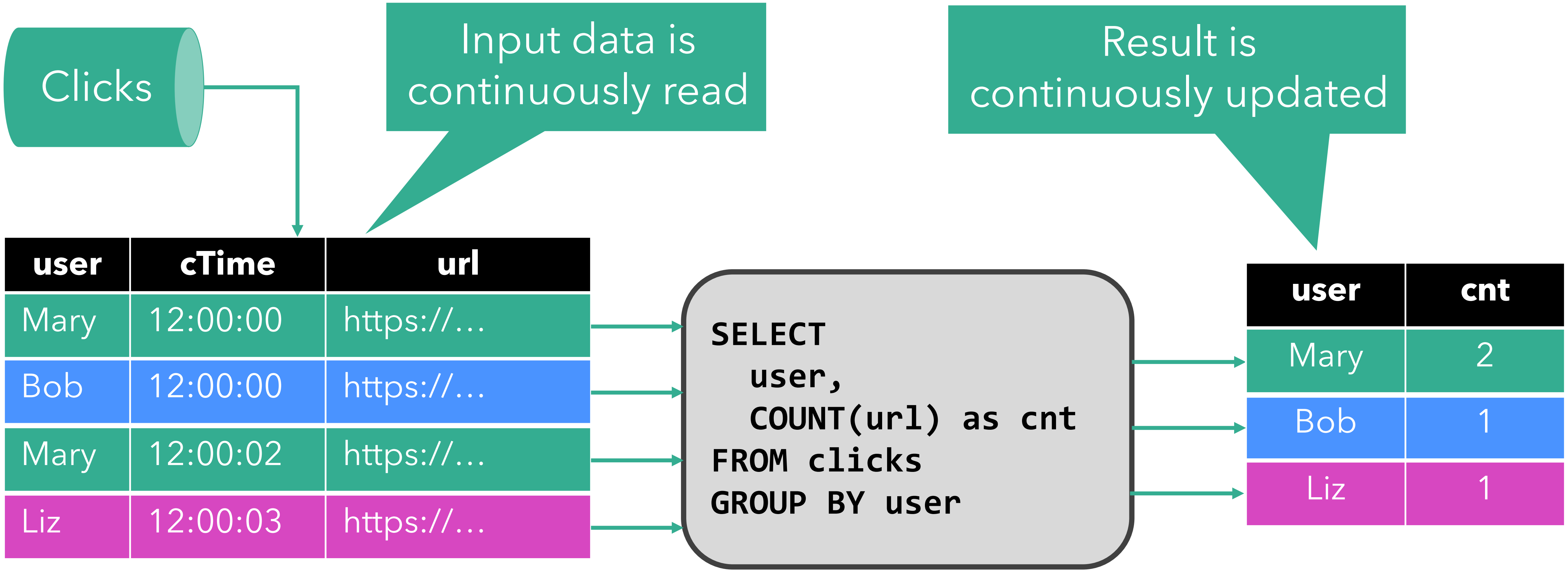


user	cTime	url
Mary	12:00:00	https://…
Bob	12:00:00	https://…
Mary	12:00:02	https://…
Liz	12:00:03	https://…

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

user	cnt
Mary	2
Bob	1
Liz	1

WHAT IF "CLICKS" IS A STREAM?


Clicks
Input data is continuously read
Result is continuously updated

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
```

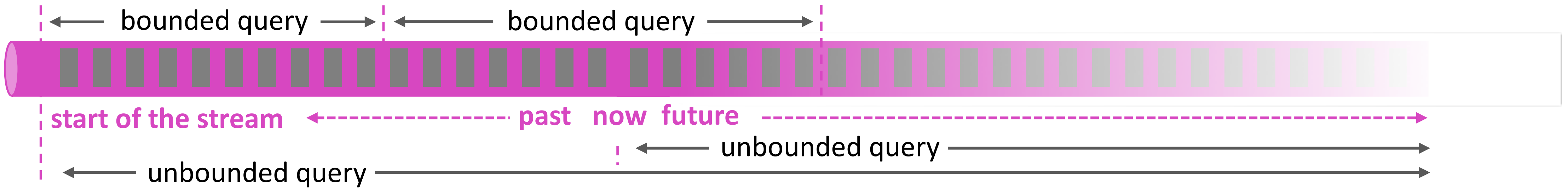
user	cnt
Mary	2
Bob	1
Liz	1

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 results.

- Portability
  - Run the same query on *bounded* and *unbounded* data
  - Run the same query on *recorded* and *real-time* data



- How can we achieve SQL semantics on streams?

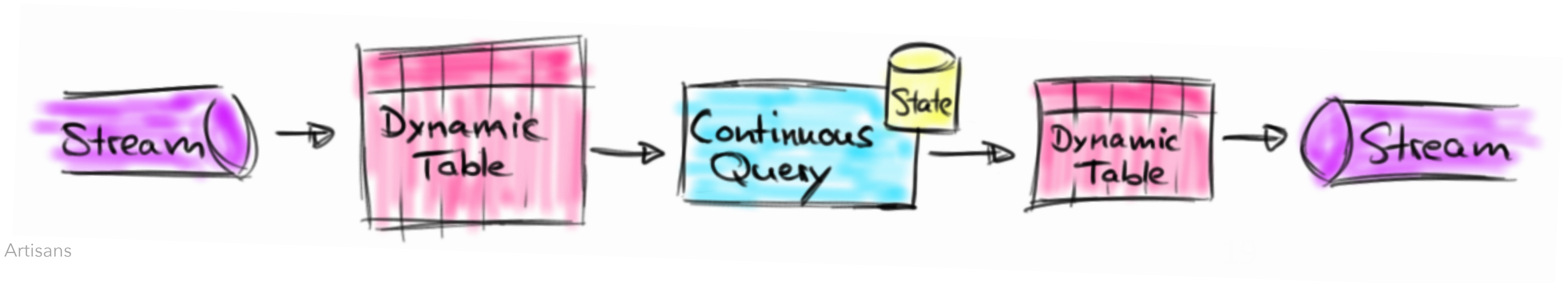
DATABASE SYSTEMS RUN QUERIES ON STREAMS

- 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

- Append Conversions
  - Records are only inserted (appended)

```
SELECT user, url
FROM clicks
WHERE url LIKE '%xyz.com'
```

- Upsert Conversions
  - Records have a (composite) unique key
  - Records are upserted/deleted by key

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

- Retract Conversions
  - Records are inserted/deleted
  - Update = delete old version + insert new version

SQL FEATURES

dataArtisans

SQL FEATURE SET IN FLINK 1.6.0

- SELECT FROM WHERE
- GROUP BY / HAVING
  - Non-windowed, TUMBLE, HOP, SESSION windows
- JOIN / IN
  - Windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
  - Non-windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
- [streaming only] OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING
- [batch only] UNION / INTERSECT / EXCEPT / ORDER BY

SQL FEATURE SET IN FLINK 1.6.0

- Support for POJOs, maps, arrays, and other nested types
- Large set of built-in functions (150+)
  - `LIKE, EXTRACT, TIMESTAMPADD, FROM_BASE64, MD5, STDDEV_POP, AVG, …`
- Support for custom UDFs (scalar, table, aggregate)

See also:
https://ci.apache.org/projects/flink/flink-docs-master/dev/table/functions.html
https://ci.apache.org/projects/flink/flink-docs-master/dev/table/udfs.html

UPCOMING SQL FEATURES

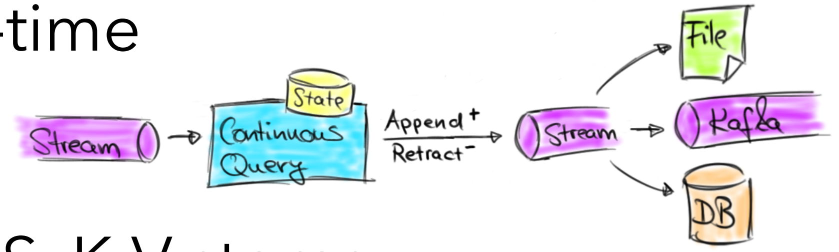
- Streaming enrichment joins (Temporal joins) [FLINK-9712]

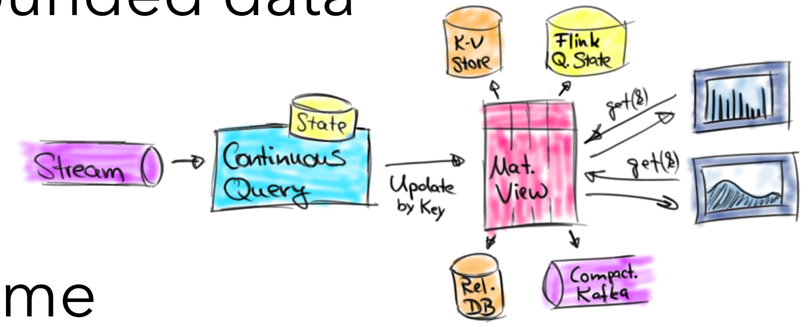
```
SELECT
  SUM(o.amount * r.rate) AS amount
FROM
  Orders AS o,
  LATERAL TABLE (Rates(o.rowtime)) AS r
WHERE r.currency = o.currency;
```

- Support for complex event processing (CEP) [FLINK-6935]
  - MATCH_RECOGNIZE
- More connectors and formats [FLINK-8535]

WHAT CAN I BUILD WITH THIS?

- Data Pipelines
  - Transform, aggregate, and move events in real-time
- Low-latency ETL
  - Convert and write streams to file systems, DBMS, K-V stores, indexes, …
  - 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





SOUNDS GREAT! HOW CAN I USE IT?

- Embed SQL queries in regular (Java/Scala) 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

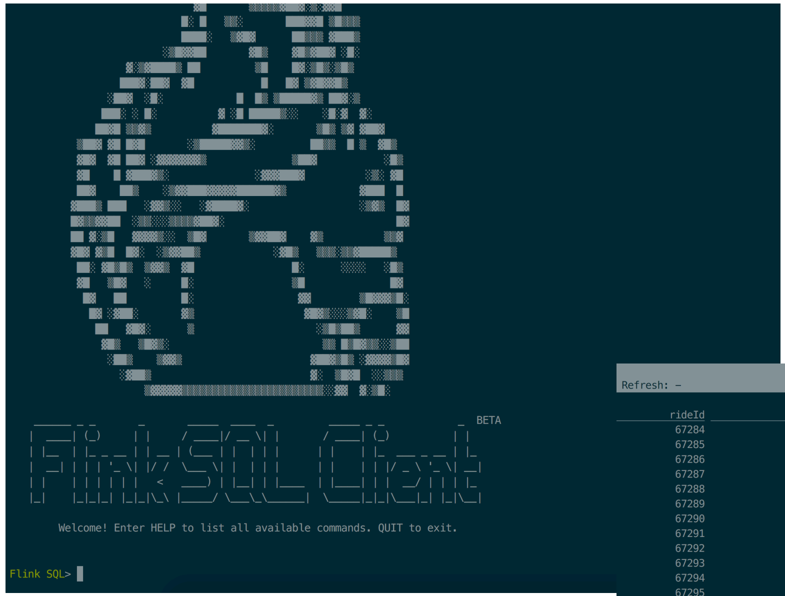
- Run SQL queries via Flink's SQL CLI Client
  - Interactive mode: Submit query and inspect results
  - Detached mode: Submit query and write results to sink system

SQL CLIENT BETA

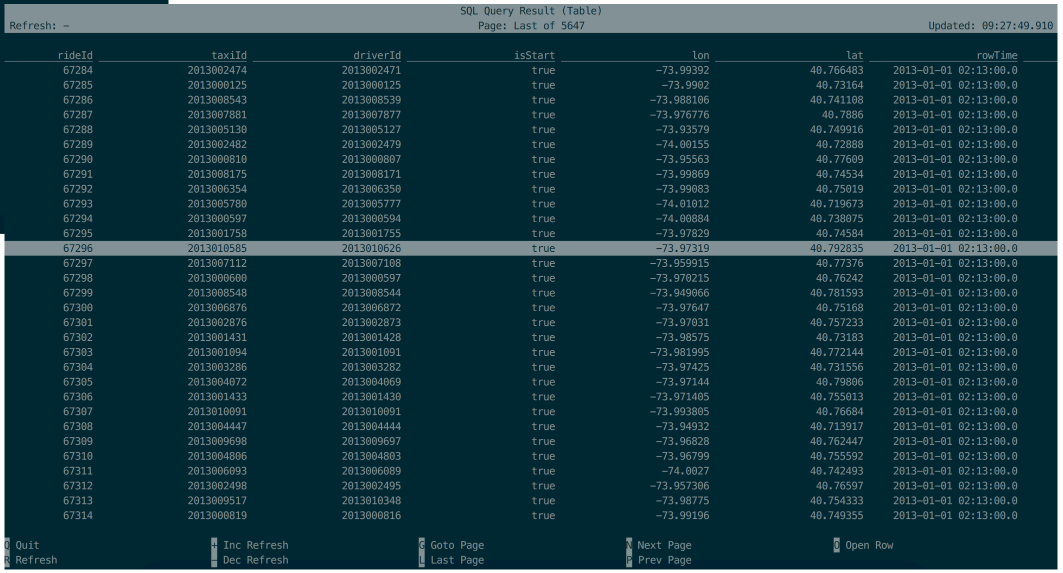
dataArtisans

INTRODUCTION TO SQL CLIENT

- Newest member of the Flink SQL family (since Flink 1.5)



```
                                                    BETA
       Welcome! Enter HELP to list all available commands. QUIT to exit.

Flink SQL> 
```

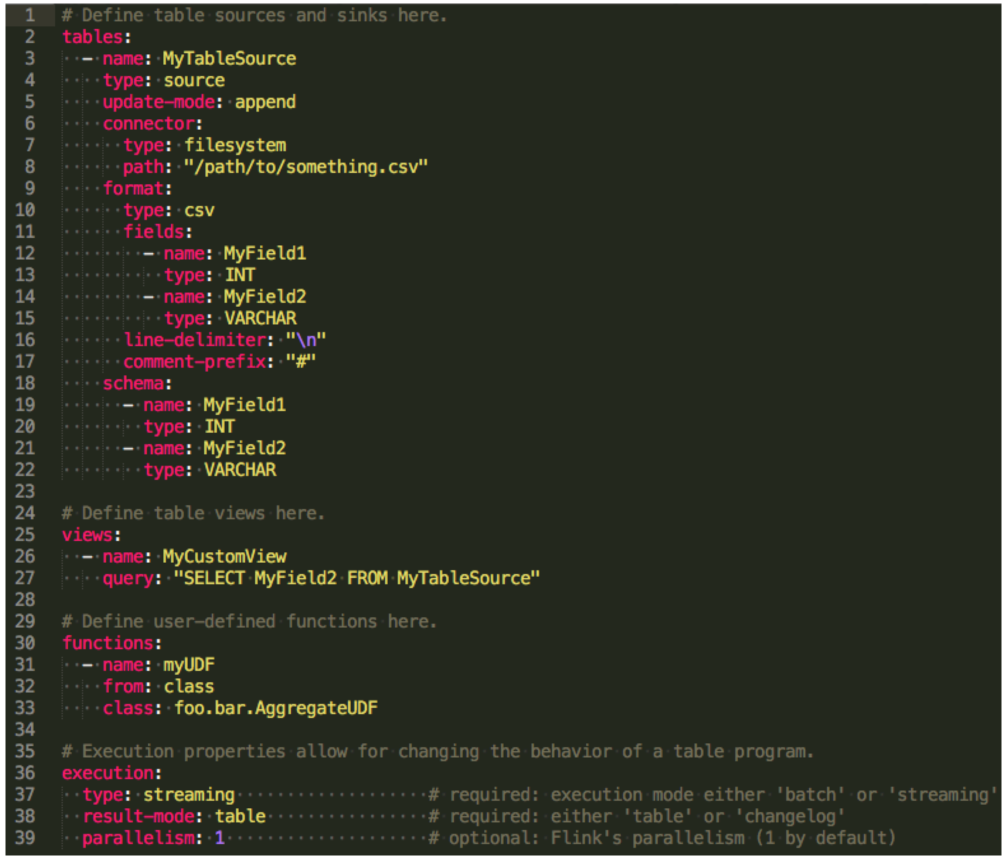


```
                                              SQL Query Result (Table)
Refresh: -                                      Page: Last of 5647                              Updated: 09:27:49.910

     rideId          taxiId        driverId         isStart            lon            lat               rowTime
      67284      2013002474      2013002471            true      -73.99392      40.766483  2013-01-01 02:13:00.0
      67285      2013000125      2013000125            true       -73.9902       40.73164  2013-01-01 02:13:00.0
      67286      2013008543      2013008539            true     -73.988106      40.741108  2013-01-01 02:13:00.0
      67287      2013007881      2013007877            true     -73.976776        40.7886  2013-01-01 02:13:00.0
      67288      2013005130      2013005127            true      -73.93579      40.749916  2013-01-01 02:13:00.0
      67289      2013002482      2013002479            true      -74.00155       40.72888  2013-01-01 02:13:00.0
      67290      2013000810      2013000807            true      -73.95563       40.77609  2013-01-01 02:13:00.0
      67291      2013008175      2013008171            true      -73.99869       40.74534  2013-01-01 02:13:00.0
      67292      2013006354      2013006350            true      -73.99083       40.75019  2013-01-01 02:13:00.0
      67293      2013005780      2013005777            true      -74.01012      40.719673  2013-01-01 02:13:00.0
      67294      2013000597      2013000594            true      -74.00884      40.738075  2013-01-01 02:13:00.0
      67295      2013001758      2013001755            true      -73.97829       40.74584  2013-01-01 02:13:00.0
      67296      2013010585      2013010626            true      -73.97319      40.792835  2013-01-01 02:13:00.0
      67297      2013007112      2013007108            true     -73.959915       40.77376  2013-01-01 02:13:00.0
      67298      2013000600      2013000597            true     -73.970215       40.76242  2013-01-01 02:13:00.0
      67299      2013008548      2013008544            true     -73.949066      40.781593  2013-01-01 02:13:00.0
      67300      2013006876      2013006872            true      -73.97647       40.75168  2013-01-01 02:13:00.0
      67301      2013002876      2013002873            true      -73.97031      40.757233  2013-01-01 02:13:00.0
      67302      2013001431      2013001428            true      -73.98575       40.73183  2013-01-01 02:13:00.0
      67303      2013001094      2013001091            true     -73.981995      40.772144  2013-01-01 02:13:00.0
      67304      2013003286      2013003282            true      -73.97425      40.731556  2013-01-01 02:13:00.0
      67305      2013004072      2013004069            true      -73.97144       40.79806  2013-01-01 02:13:00.0
      67306      2013001433      2013001430            true     -73.971405      40.755013  2013-01-01 02:13:00.0
      67307      2013010091      2013010091            true     -73.993805       40.76684  2013-01-01 02:13:00.0
      67308      2013004447      2013004444            true      -73.94932      40.713917  2013-01-01 02:13:00.0
      67309      2013009698      2013009697            true      -73.96828      40.762447  2013-01-01 02:13:00.0
      67310      2013004806      2013004803            true      -73.96799      40.755592  2013-01-01 02:13:00.0
      67311      2013006093      2013006089            true       -74.0027      40.742493  2013-01-01 02:13:00.0
      67312      2013002498      2013002495            true     -73.957306       40.76597  2013-01-01 02:13:00.0
      67313      2013009517      2013010348            true      -73.98775      40.754333  2013-01-01 02:13:00.0
      67314      2013000819      2013000816            true      -73.99196      40.749355  2013-01-01 02:13:00.0

Q Quit              + Inc Refresh        G Goto Page         N Next Page         O Open Row
R Refresh           - Dec Refresh        L Last Page         P Prev Page
```

INTRODUCTION TO SQL CLIENT

- Goal: Flink without a single line of code
  - only SQL and YAML
  - *"drag&drop"* SQL JAR files for connectors and formats
- Build on top of Flink's Table & SQL API
- Useful for prototyping & submission

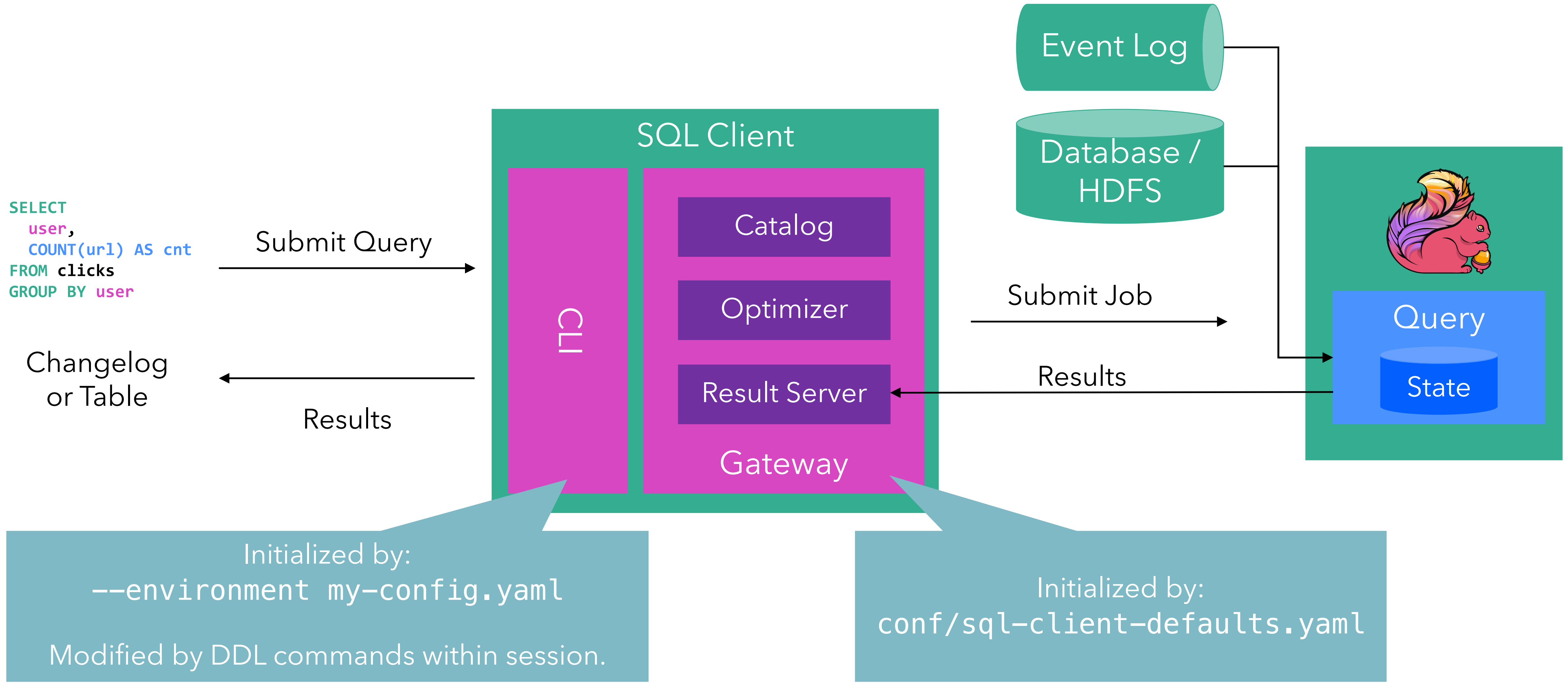
SQL CLIENT CONFIGURATION



```yaml
# Define table sources and sinks here.
tables:
  - name: MyTableSource
    type: source
    update-mode: append
    connector:
      type: filesystem
      path: "/path/to/something.csv"
    format:
      type: csv
      fields:
        - name: MyField1
          type: INT
        - name: MyField2
          type: VARCHAR
      line-delimiter: "\n"
      comment-prefix: "#"
    schema:
      - name: MyField1
        type: INT
      - name: MyField2
        type: VARCHAR

# Define table views here.
views:
  - name: MyCustomView
    query: "SELECT MyField2 FROM MyTableSource"

# Define user-defined functions here.
functions:
  - name: myUDF
    from: class
    class: foo.bar.AggregateUDF

# Execution properties allow for changing the behavior of a table program.
execution:
  type: streaming                   # required: execution mode either 'batch' or 'streaming'
  result-mode: table                # required: either 'table' or 'changelog'
  parallelism: 1                    # optional: Flink's parallelism (1 by default)
```

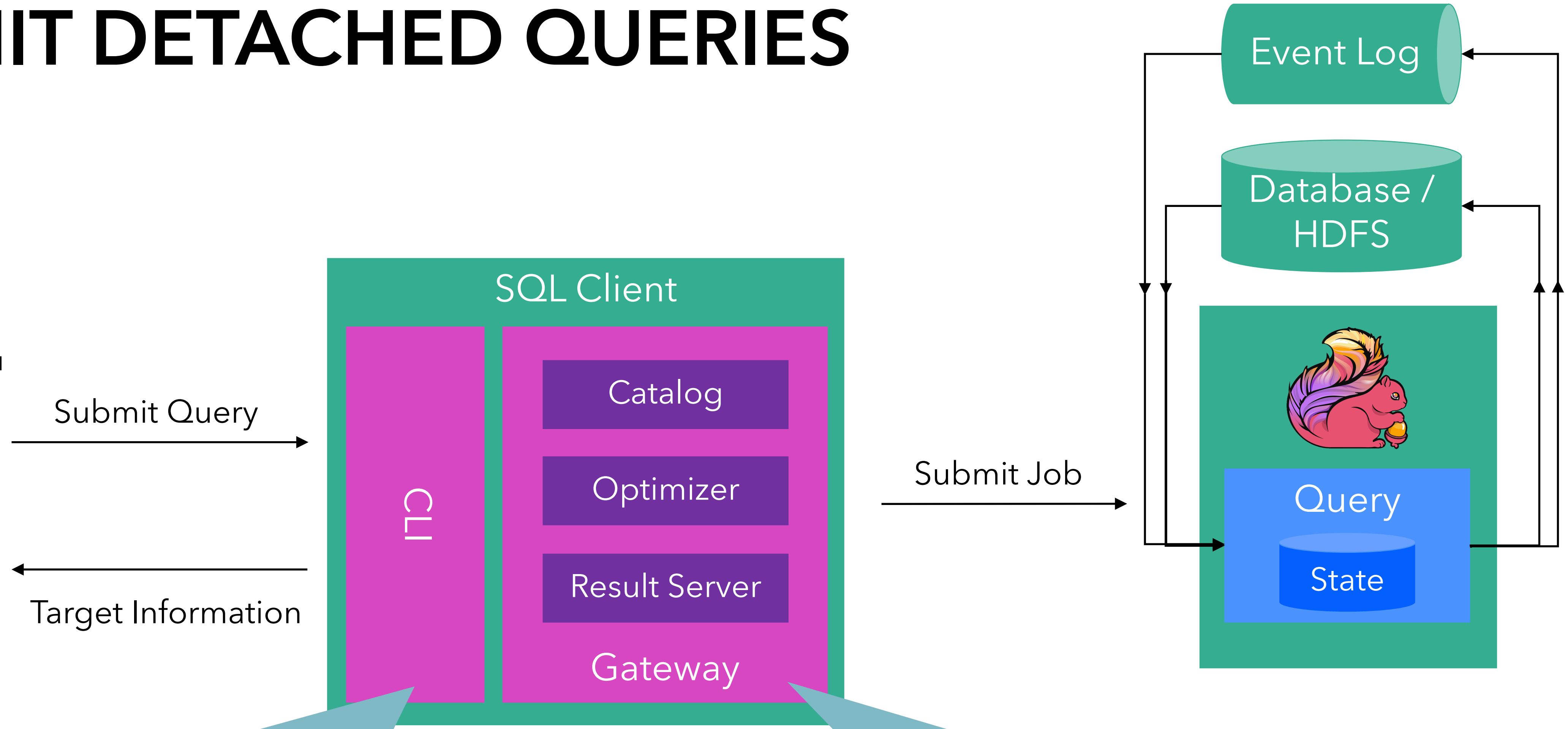
See also:
https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html

PLAY AROUND WITH FLINK SQL



SUBMIT DETACHED QUERIES

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



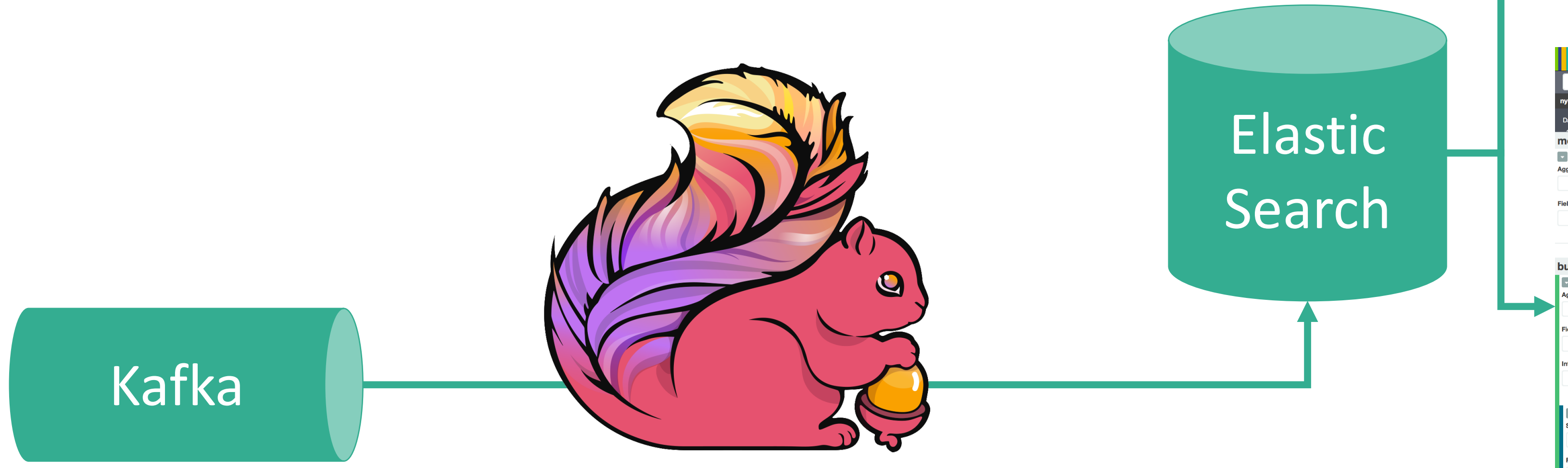
Initialized by:
`--environment my-config.yaml`

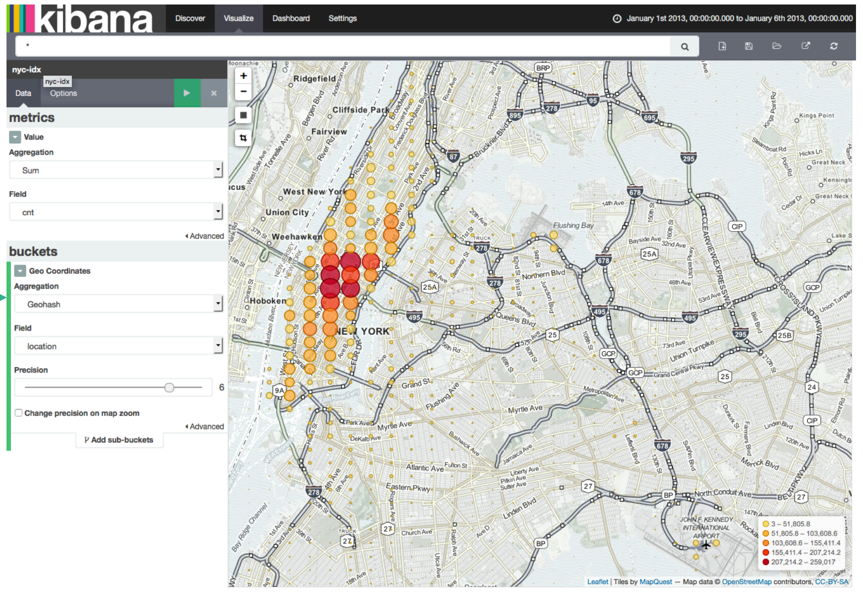
Modified by DDL commands within session.

Initialized by:
`conf/sql-client-defaults.yaml`

SERVING A DASHBOARD

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





kibana
Discover
Visualize
Dashboard
Settings
January 1st 2013, 00:00:00.000 to January 6th 2013, 00:00:00.000

nyc-idx

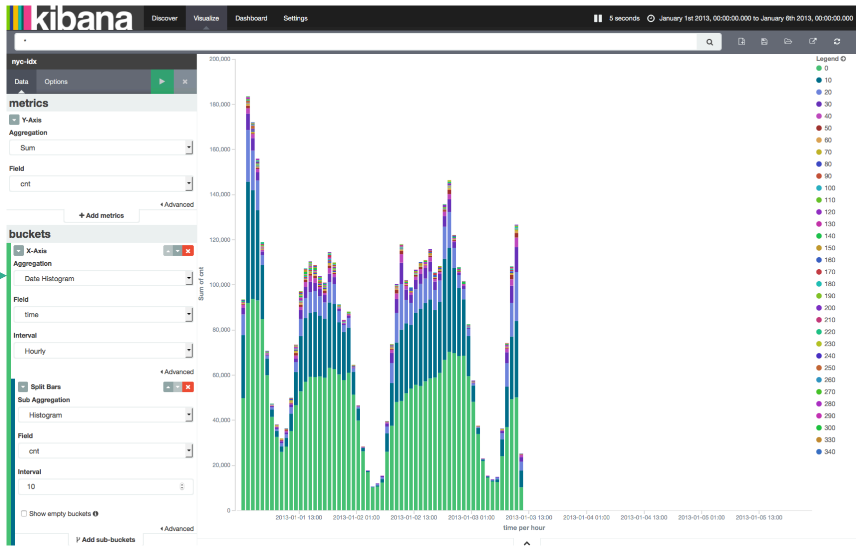
Data | Options

metrics

Value
Aggregation: Sum
Field: cnt

buckets

Geo Coordinates
Aggregation: Geohash
Field: location
Precision: 6
☐ Change precision on map zoom



nyc-idx

Data | Options

metrics

Y-Axis
Aggregation: Sum
Field: cnt

buckets

X-Axis
Aggregation: Date Histogram
Field: time
Interval: Hourly

Split Bars
Sub Aggregation: Histogram
Field: cnt
Interval: 10
☐ Show empty buckets

ACTION TIME!

HTTPS://GITHUB.COM/DATAARTISANS/SQL-TRAINING

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

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
```

SUMMARY

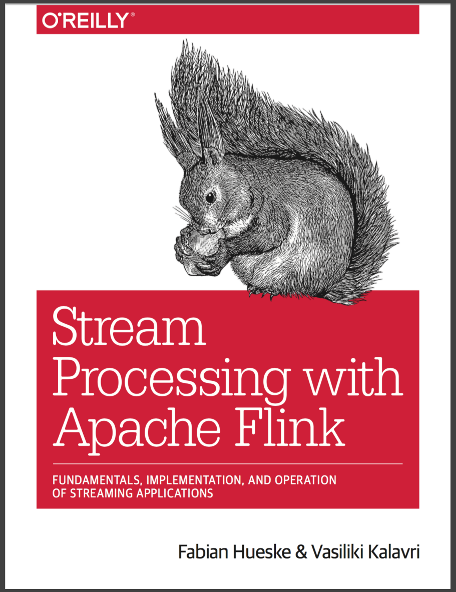
- Unification of stream and batch is important.

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

- Get involved, discuss, and contribute!

THANK YOU!

@twalthr
@dataArtisans
@ApacheFlink


O'REILLY

Stream Processing with Apache Flink

FUNDAMENTALS, IMPLEMENTATION, AND OPERATION OF STREAMING APPLICATIONS

Fabian Hueske & Vasiliki Kalavri

Available on O'Reilly Early Release!

WE ARE HIRING

data-artisans.com/careers

dataArtisans