Detecting Patterns in Event Streams with Flink SQL

Dawid Wysakowicz, SOFTWARE ENGINEER

dataArtisans

ABOUT DATA ARTISANS

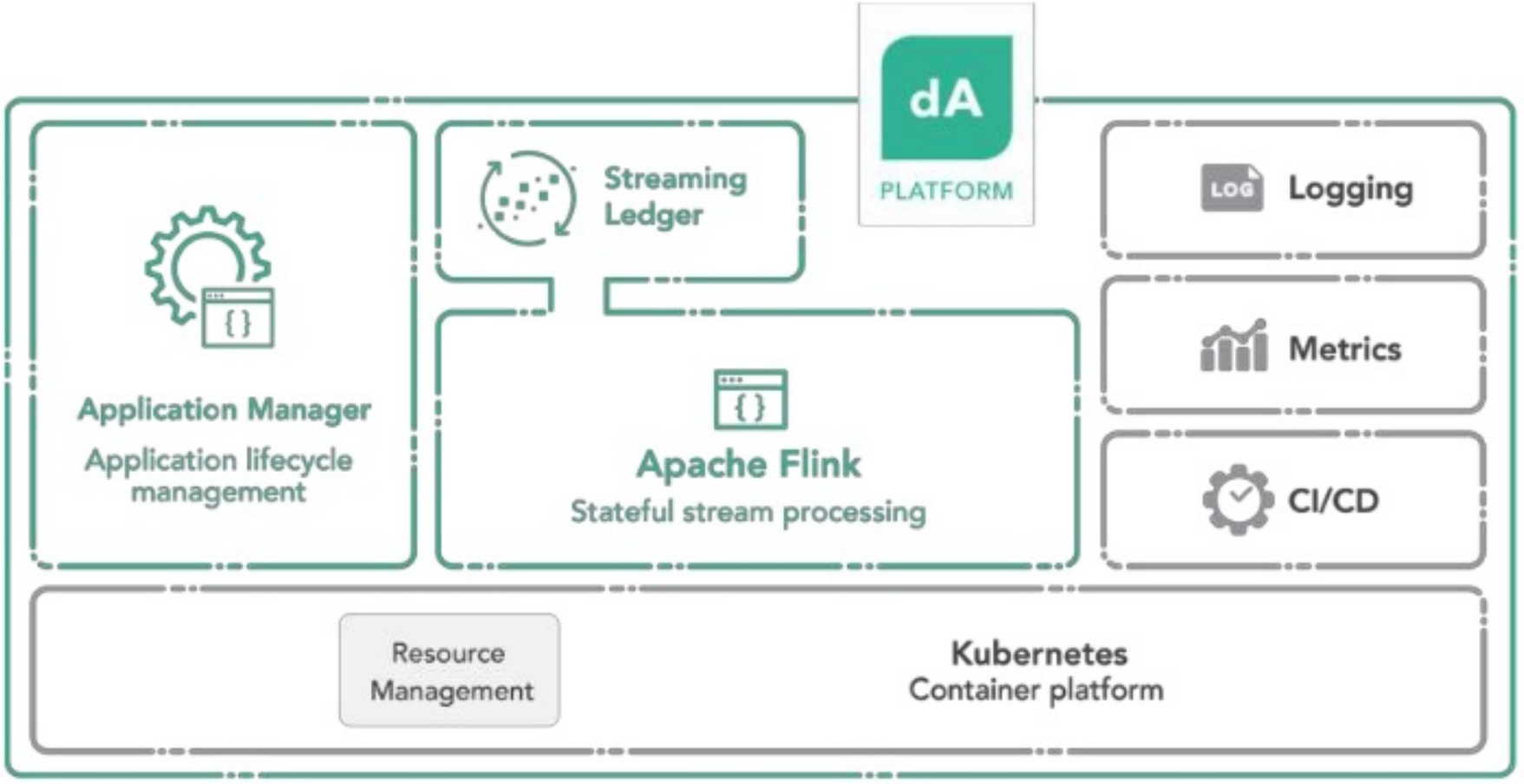
Original Creators of
Apache Flink®

dA

PLATFORM

Enterprise Ready
Real Time Stream Processing

FREE TRIAL DOWNLOAD



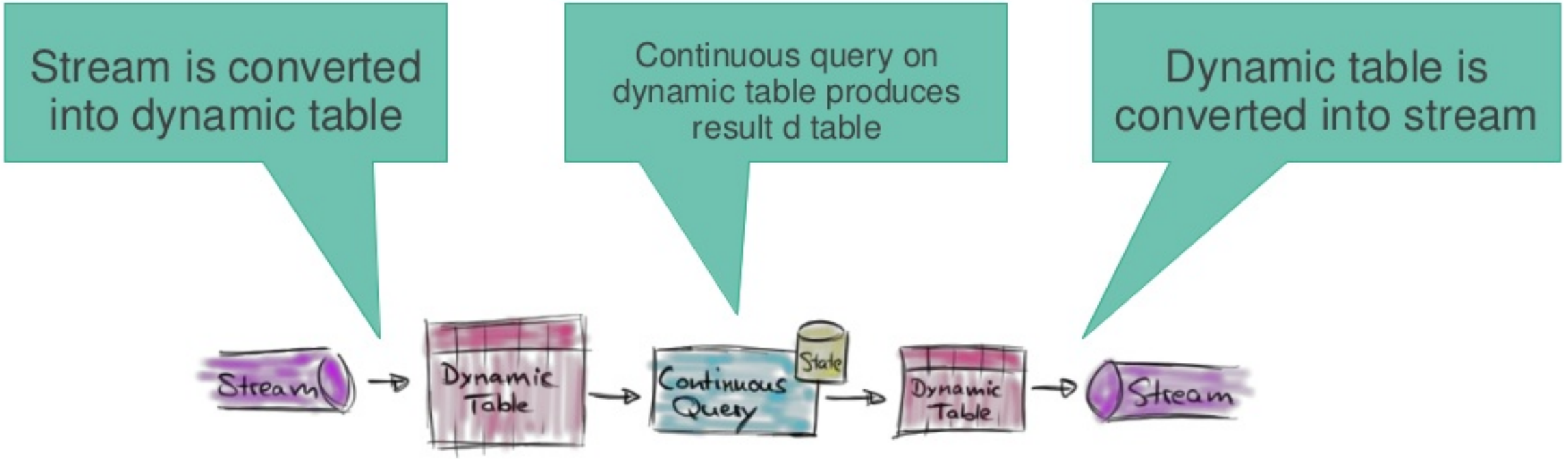
data-artisans.com/download

Why people like SQL?

- Well known interface: "lingua franca of data processing"
- No programming required - easier to learn
- Declarative way to express your query, WHAT not HOW
- Out-of-the box optimization
- Reusability
  - built-in functions
  - user defined functions

SQL on Streams

- Unified semantics for stream and batch input
  - Same query, same input, same result



The New York Taxi Rides Data Set

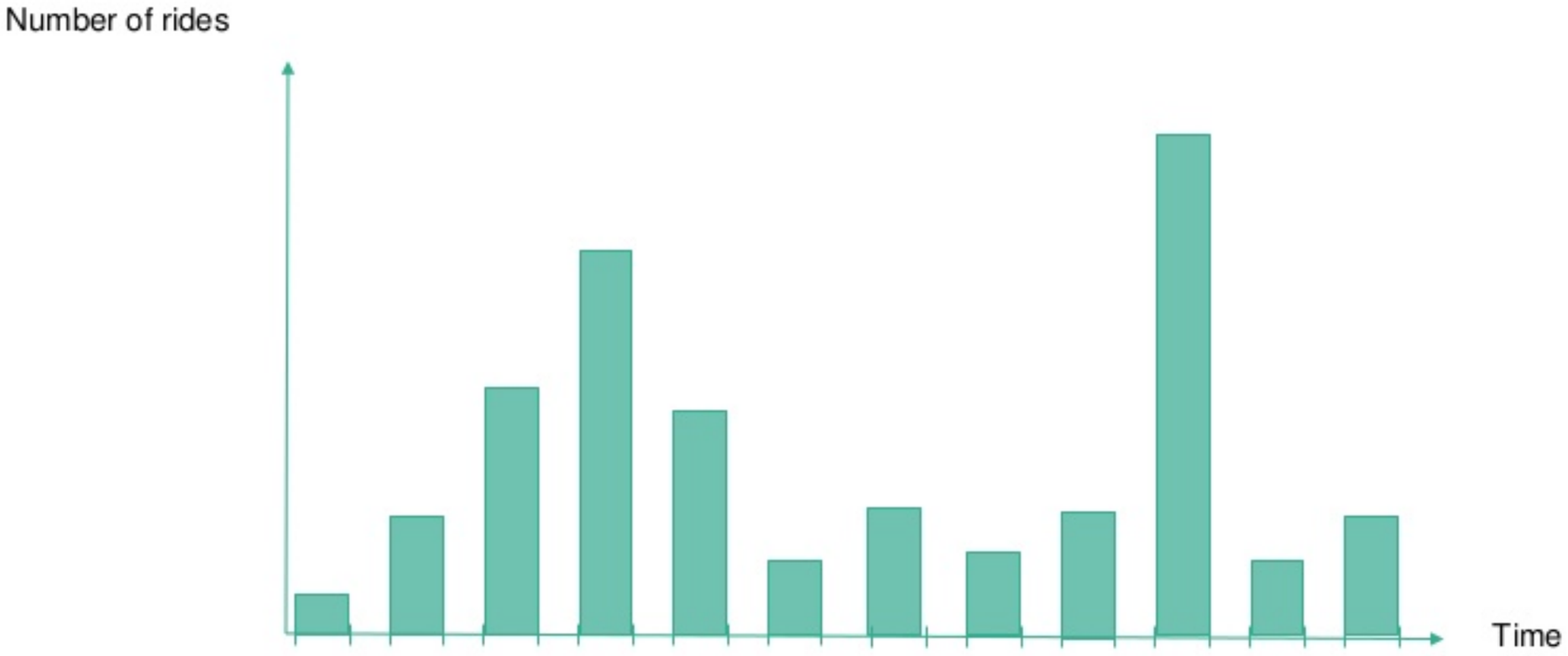
- The New York City & Limousine Commission provides a public data set about past taxi rides in New York City

- We can derive a streaming table from the data

- Table: **TaxiRides**

```
rideId:         BIGINT      // ID of the taxi ride
taxiId:         BIGINT      // ID of the taxi cab
driverId:       BIGINT      // ID of the drive
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
rowTime:        TIMESTAMP   // time of pick-up or drop-off event
```


Number of rides per 30 minutes per area



SQL Example

```
SELECT
        toCellId(lat, lon) as cellId,
    COUNT(distinct rideId) as rideCount,
    TUMBLE_ROWTIME(rowTime, INTERVAL '30' minute) AS rowTime,
    cast (TUMBLE_START(rowTime, INTERVAL '30' minute) as TIMESTAMP) AS startTime,
    cast(TUMBLE_END(rowTime, INTERVAL '30' minute) as TIMESTAMP) AS endTime
FROM
    TaxiRides
GROUP BY
    toCellId(lat, lon),
    TUMBLE(rowTime, INTERVAL '30' minute)
```


SQL Support in Flink 1.6

- SELECT FROM WHERE
- GROUP BY / HAVING
  - Non-windowed, TUMBLE, HOP, SESSION windows
- JOIN
  - Windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
  - Non-windowed INNER / LEFT / RIGHT / FULL OUTER JOIN
- Scalar, aggregation, table-valued UDFs
- Many built-in functions

- [streaming only] OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING

SQL Support in Flink 1.6

- SELECT FROM WHERE
- GROUP BY / HAVING
  - Non-windowed, TUMBLE, HOP, SESSION windows
- JOIN
  - Windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
  - Non-windowed INNER / LEFT / RIGHT / FULL OUTER JOIN
- Scalar, aggregation, table-valued UDFs
- Many built-in functions

- [streaming only] OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING

12:20-13:00 Kesselhaus
Timo Walther, data Artisans
Flink SQL in Action

What is hard with SQL?

- Find rides with mid-stops

Mid stops

```
Pattern.<Row>begin("S").where((row) -> {
                    return row.isStart == true;
    }).next("E").where( (row) -> {
                    return row.isStart == true;
          });
```

```
CEP.pattern(input.keyBy("driverId"), pattern)
.flatSelect(
new PatternFlatSelectFunction<Row, Row>() {
          @Override
          public void flatSelect(
          Map<String, List<Row>> pattern,
          Collector<Row> out) throws Exception {
                    out.collect((
               pattern.get("S").get(0).getRideId
          ));
               }
);
```


MATCH_RECOGNIZE

SQL:2016 extension

Common use-cases

- stock market analysis
- customer behaviour
- tracking money laundering
- service quality
- network intrusion detection

Syntax

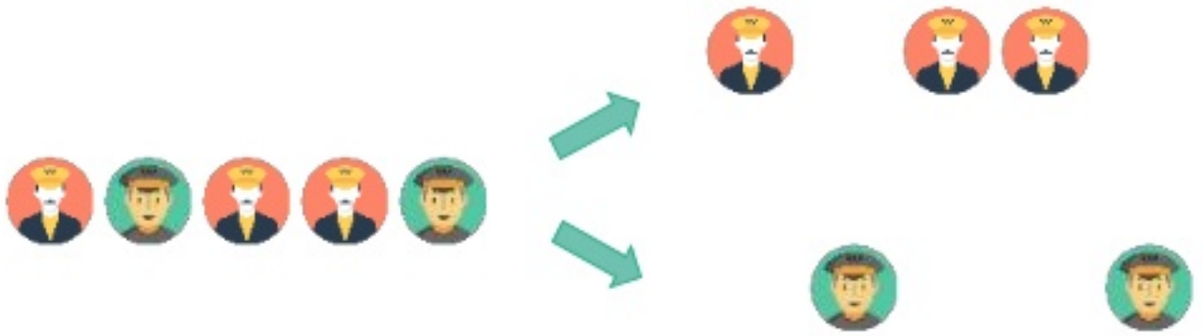
```
tableReference:
      tablePrimary
      [ matchRecognize ]
      [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]
```

```
select:
        SELECT [ STREAM ] [ ALL | DISTINCT ]
              { * | projectItem [, projectItem ]* }
        FROM tableExpression
        [matchRecognize]
              [ WHERE booleanExpression ]
        [ GROUP BY { groupItem [, groupItem ]* } ]
        ....
```

Rides with mid-stops

```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S E)
    DEFINE
        S AS S.isStart = true,
        E AS E.isStart = true
)
```

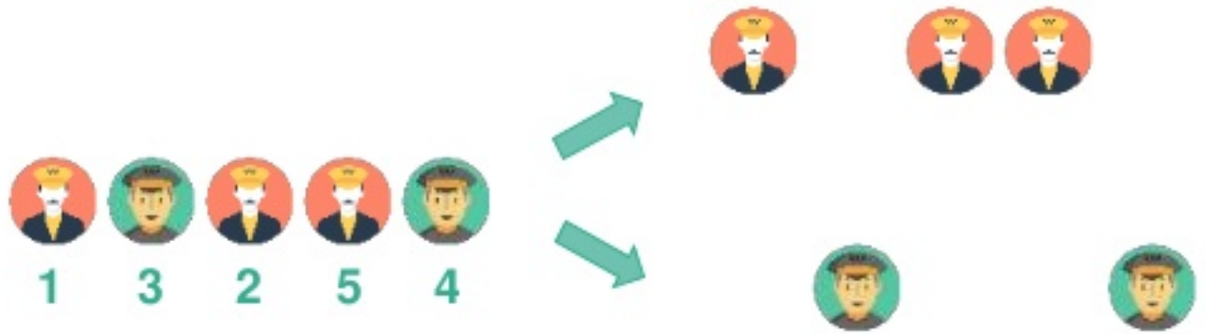
partition the data by given field = keyBy



Rides with mid-stops

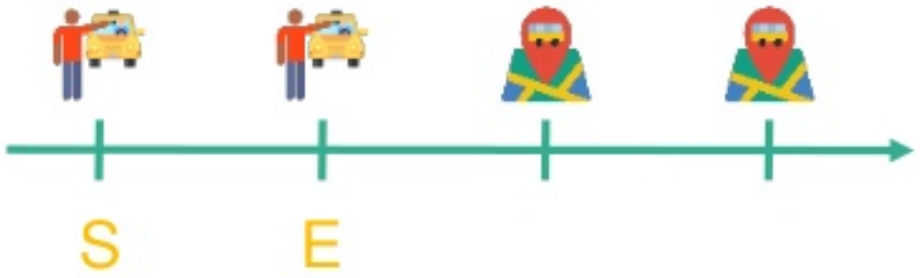
```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S E)
    DEFINE
        S AS S.isStart = true,
        E AS E.isStart = true
)
```

- specify order
- primary order = Event or Processing time



Rides with mid-stops

```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S E)
    DEFINE
        S AS S.isStart = true,
        E AS E.isStart = true
)
```

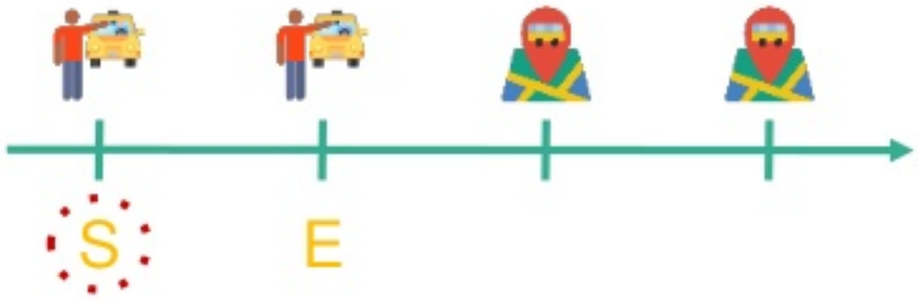


construct pattern

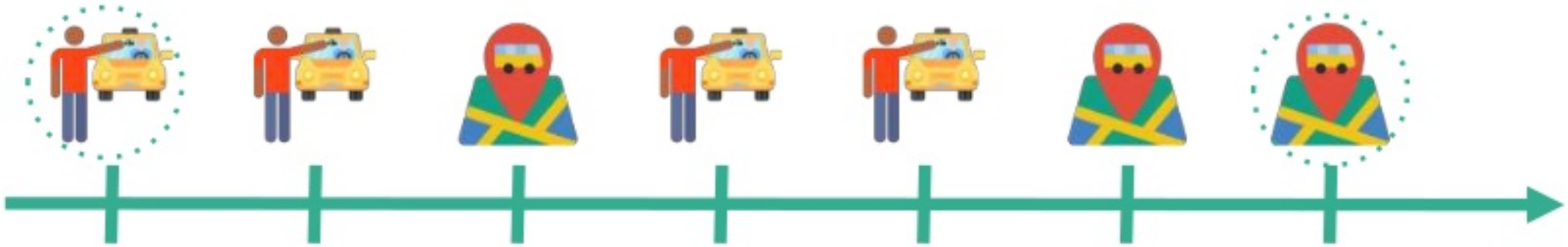
Rides with mid-stops

```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S E)
    DEFINE
        S AS S.isStart = true,
        E AS E.isStart = true
)
```

extract measures from matched sequence



Multi-Stop

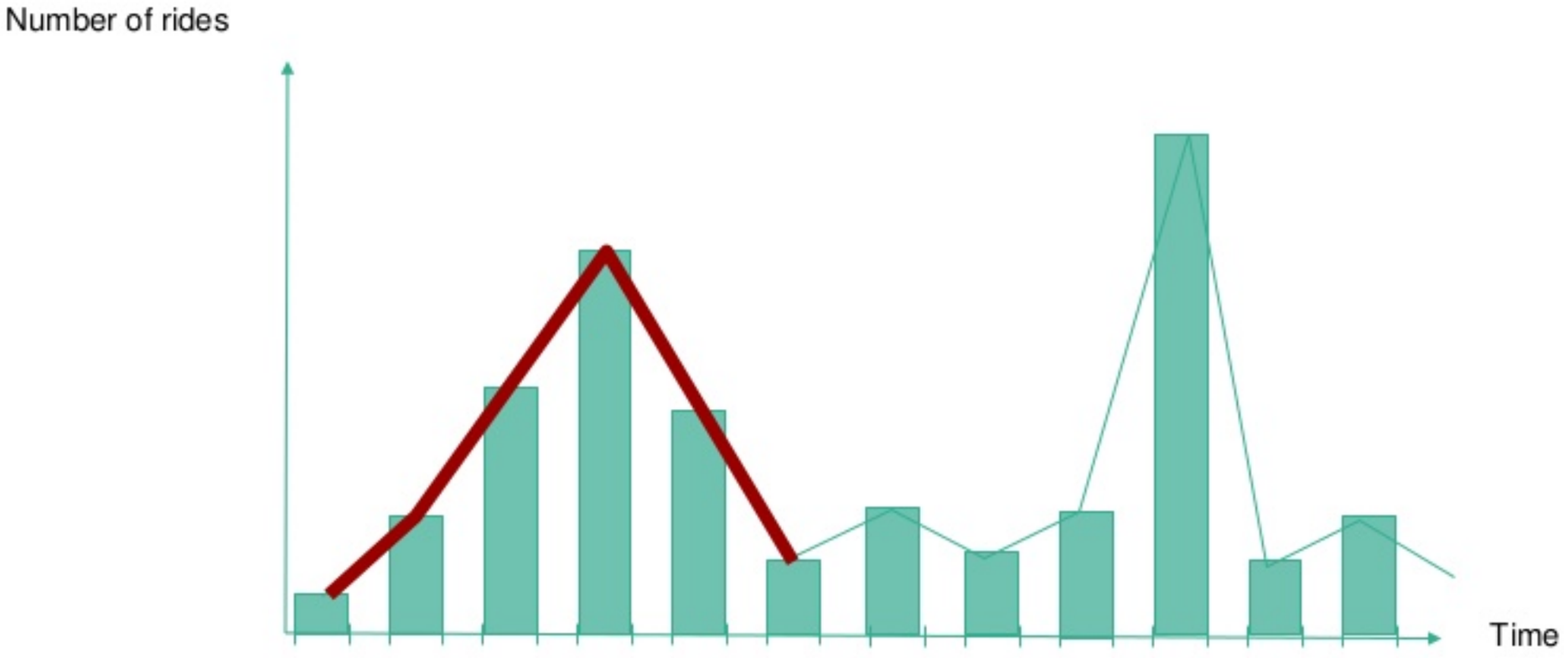


Rides with more than one mid-stop

```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S E)
    DEFINE
        S AS S.isStart = true,
        E AS E.isStart = true
)
```

```
SELECT *
FROM TaxiRides
MATCH_RECOGNIZE (
    PARTITION BY driverId
    ORDER BY rowTime
    MEASURES
        S.rideId as sRideId
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (S M{2,} E)
    DEFINE
        S AS S.isStart = true,
        M AS M.rideId <> S.rideId,
        E AS E.isStart = false
                AND E.rideId = S.rideId
)
```


Rush (peak) hours - V shape

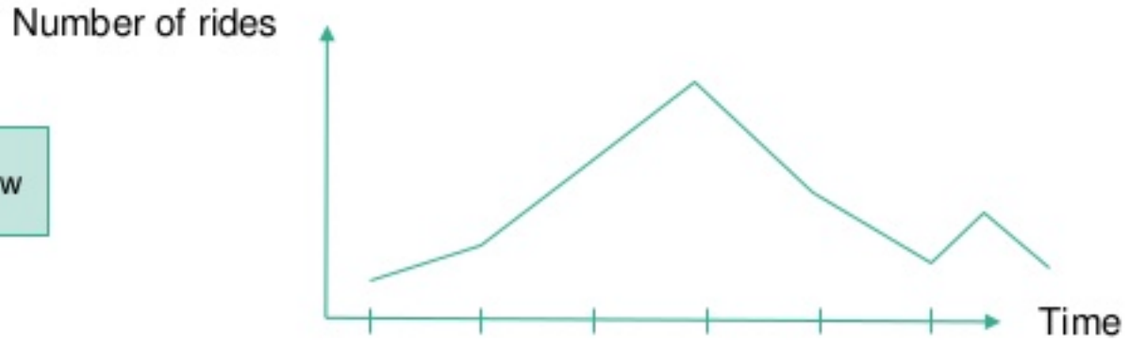


 |

Statistics per Area

```
CREATE VIEW RidesInArea AS
SELECT
      toCellId(lat, lon) as cellId,
   COUNT(distinct rideId) as rideCount,
   TUMBLE_ROWTIME(rowTime, INTERVAL '30' minute) AS rowTime,
   cast (TUMBLE_START(rowTime, INTERVAL '30' minute) as TIMESTAMP) AS startTime,
   cast(TUMBLE_END(rowTime, INTERVAL '30' minute) as TIMESTAMP) AS endTime
FROM
   TaxiRides
GROUP BY
   toCellId(lat, lon),
   TUMBLE(rowTime, INTERVAL '30' minute)
```

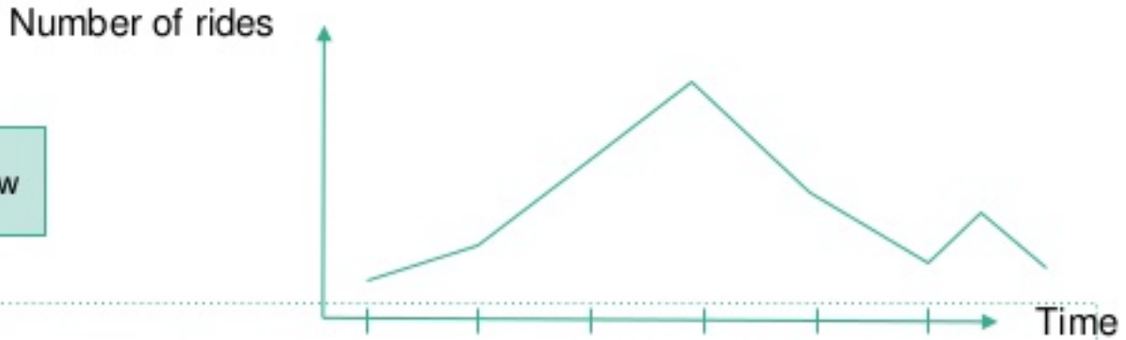

Rush (peak) hours



Use previous table/view

```
SELECT * FROM RidesInArea
MATCH_RECOGNIZE(
      PARTITION BY cellId
      MEASURES
            FIRST(UP.rideCount) AS preCnt,
            LAST(UP.rideCount) AS rushCnt, FIRST(UP.startTime) as rushStart,
      LAST(UP.endTime) as rushEnd,
            LAST(DOWN.rideCount) AS postCnt
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (UP{4,} DOWN{2,} E)
      DEFINE UP AS UP.rideCount > PREV(UP.rideCount) or PREV(UP.rideCount) IS
NULL,
             DOWN AS DOWN.rideCount < PREV(DOWN.rideCount) AND
              DOWN.rideCount < LAST(UP.rideCount),
             E AS E.rideCount > LAST(DOWN.rideCount)
```


Rush (peak) hours

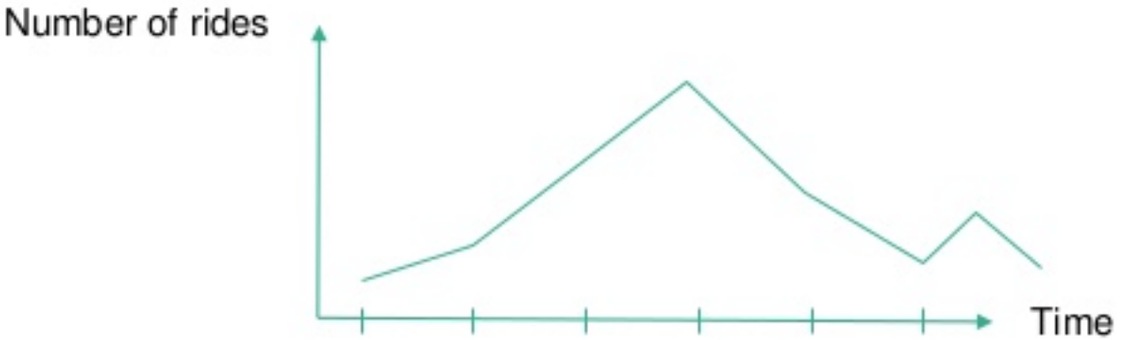


Use previous table/view

apply match to result of the inner query

```
SELECT * FROM RidesInArea
MATCH_RECOGNIZE(
      PARTITION BY cellId
      MEASURES
            FIRST(UP.rideCount) AS preCnt,
            LAST(UP.rideCount) AS rushCnt, FIRST(UP.startTime) as rushStart,
      LAST(UP.endTime) as rushEnd,
            LAST(DOWN.rideCount) AS postCnt
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (UP{4,} DOWN{2,} E)
      DEFINE UP AS UP.rideCount > PREV(UP.rideCount) or PREV(UP.rideCount) IS
NULL,
             DOWN AS DOWN.rideCount < PREV(DOWN.rideCount) AND
              DOWN.rideCount < LAST(UP.rideCount),
             E AS E.rideCount > LAST(DOWN.rideCount)
```

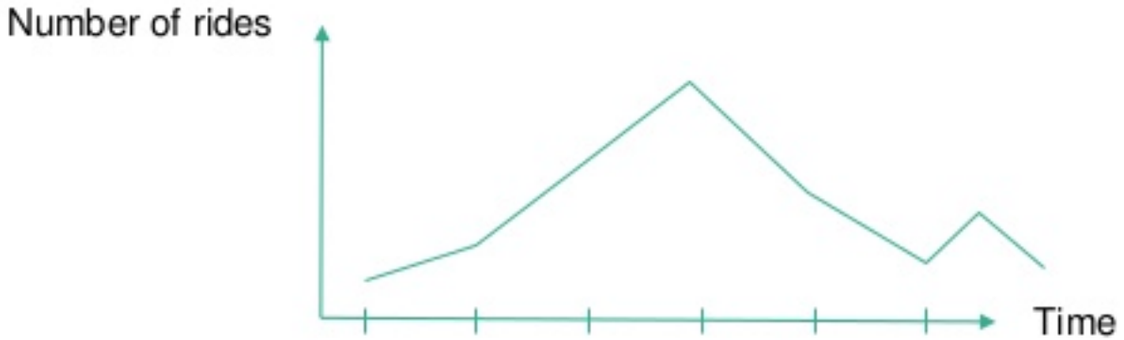

Rush (peak) hours




```
SELECT * FROM RidesInArea
MATCH_RECOGNIZE(
      PARTITION BY cellId
      MEASURES
            FIRST(UP.rideCount) AS preCnt,
            LAST(UP.rideCount) AS rushCnt, FIRST(UP.startTime) as rushStart,
      LAST(UP.endTime) as rushEnd,
            LAST(DOWN.rideCount) AS postCnt
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (UP{4,} DOWN{2,} E)
      DEFINE UP AS UP.rideCount > PREV(UP.rideCount) or PREV(UP.rideCount) IS
NULL,
             DOWN AS DOWN.rideCount < PREV(DOWN.rideCount) AND
              DOWN.rideCount < LAST(UP.rideCount),
             E AS E.rideCount > LAST(DOWN.rideCount)
```

access elements of looping pattern

Rush (peak) hours




```
SELECT * FROM RidesInArea
MATCH_RECOGNIZE(
      PARTITION BY cellId
      MEASURES
            FIRST(UP.rideCount) AS preCnt,
            LAST(UP.rideCount) AS rushCnt, FIRST(UP.startTime) as rushStart,
      LAST(UP.endTime) as rushEnd,
            LAST(DOWN.rideCount) AS postCnt
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (UP{4,} DOWN{2,} E)
      DEFINE UP AS UP.rideCount > PREV(UP.rideCount) or PREV(UP.rideCount) IS
NULL,
            DOWN AS DOWN.rideCount < PREV(DOWN.rideCount) AND
             DOWN.rideCount < LAST(UP.rideCount),
            E AS E.rideCount > LAST(DOWN.rideCount)
```

access elements of looping pattern

 |

Feature set of MATCH_RECOGNIZE (Experimental)

- Quantifiers support:
  - + (one or more), * (zero or more), {x,y} (times)
  - greedy(default), ?(reluctant)
    - with some restrictions (not working for last pattern)
- After Match Skip
  - skip_to_first/last, skip_past_last, skip_to_next
- Not supported:
  - alter(|), permute, exclude ‘{- -}’
  - aggregates within MATCH_RECOGNIZE

Summary

- opens for new use cases, that assume order of events
- enables reuse of SQL goods
- still in experimental phase
  - https://github.com/dawidwys/flink/SQL_MATCH_RECOGNIZE

THANK YOU!

@dwysakowicz

@dataArtisans

@ApacheFlink

WE ARE HIRING

data-artisans.com/careers

dataArtisans

Under the hood

Dissecting MATCH_RECOGNIZE

Under the hood

```
SELECT * FROM TaxiStatistics
MATCH_RECOGNIZE(
        PARTITION BY cellId
        MEASURES
                LAST(UP.rideCount) AS rushCnt,
        FIRST(UP.startTime) as rushStart,
        LAST(UP.endTime) as rushEnd
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (UP{4,} DOWN{2,} E)
        DEFINE
        UP AS UP.rideCount > PREV(UP.rideCount)
            or PREV(UP.rideCount) IS NULL,
                DOWN AS DOWN.rideCount  <
                LAST(UP.rideCount),
                E AS E.rideCount  >
              LAST(DOWN.rideCount)
)
```

```
Pattern.<Row>begin("UP", SKIP_PAST_LAST).timesOrMore(4).where((row, ctx) ->
{
                        Long prevRideCount = ctx.get("UP").tail().get("rideCount");
                        return prevRideCount == null || row.get("rideCount") >
prevRideCount;
        }).next("DOWN").timesOrMore(2).where( (row, ctx) -> {
                        ...
        }).next("E").where( (row, ctx) -> {
        ...
        });

CEP.pattern(input.keyBy("cellId"), pattern).flatSelect(
new PatternFlatSelectFunction<Row, Row>() {
        @Override
        public void flatSelect(Map<String, List<Row>> pattern,
        Collector<Row> out) throws Exception {
                Row last = pattern.get("UP").tail();
                Row first = pattern.get("UP").headl();
                                out.collect(Row.of(...));
                        }
);
```


Under the hood

```
SELECT * FROM TaxiStatistics
MATCH_RECOGNIZE(
        PARTITION BY cellId
        MEASURES
                LAST(UP.rideCount) AS rushCnt,
        FIRST(UP.startTime) as rushStart,
        LAST(UP.endTime) as rushEnd
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (UP{4,} DOWN{2,} E)
        DEFINE
        UP AS UP.rideCount > PREV(UP.rideCount)
            or PREV(UP.rideCount) IS NULL,
                DOWN AS DOWN.rideCount <
                LAST(UP.rideCount),
                E AS E.rideCount  >
              LAST(DOWN.rideCount)
)
```

```
Pattern.<Row>begin("UP", SKIP_PAST_LAST).timesOrMore(4).where((row, ctx) ->
{
                        Long prevRideCount = ctx.get("UP").tail().get("rideCount");
                        return prevRideCount == null || row.get("rideCount") >
prevRideCount;
        }).next("DOWN").timesOrMore(2).where( (row, ctx) -> {
                        ...
        }).next("E").where( (row, ctx) -> {
        ...
        });

CEP.pattern(input.keyBy("cellId"), pattern).flatSelect(
new PatternFlatSelectFunction<Row, Row>() {
        @Override
        public void flatSelect(Map<String, List<Row>> pattern,
        Collector<Row> out) throws Exception {
                Row last = pattern.get("UP").tail();
                Row first = pattern.get("UP").headl();
                                out.collect(Row.of(...));
                        }
);
```


Under the hood

```
SELECT * FROM TaxiStatistics
MATCH_RECOGNIZE(
        PARTITION BY cellId
        MEASURES
                LAST(UP.rideCount) AS rushCnt,
        FIRST(UP.startTime) as rushStart,
        LAST(UP.endTime) as rushEnd
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (UP{4,} DOWN{2,} E)
        DEFINE
        UP AS UP.rideCount > PREV(UP.rideCount)
            or PREV(UP.rideCount) IS NULL,
                DOWN AS DOWN.rideCount <
                LAST(UP.rideCount),
                E AS E.rideCount  >
              LAST(DOWN.rideCount)
)
```

```
Pattern.<Row>begin("UP", SKIP_PAST_LAST).timesOrMore(4).where((row, ctx) ->
{
                        Long prevRideCount = ctx.get("UP").tail().get("rideCount");
                        return prevRideCount == null || row.get("rideCount") >
prevRideCount;
        }).next("DOWN").timesOrMore(2).where( (row, ctx) -> {
                        ...
        }).next("E").where( (row, ctx) -> {
        ...
        });

CEP.pattern(input.keyBy("cellId"), pattern).flatSelect(
new PatternFlatSelectFunction<Row, Row>() {
        @Override
        public void flatSelect(Map<String, List<Row>> pattern,
        Collector<Row> out) throws Exception {
                Row last = pattern.get("UP").tail();
                Row first = pattern.get("UP").headl();
                                out.collect(Row.of(...));
                        }
);
```


Under the hood

```
SELECT * FROM TaxiStatistics
MATCH_RECOGNIZE(
        PARTITION BY cellId
        MEASURES
                LAST(UP.rideCount) AS rushCnt,
        FIRST(UP.startTime) as rushStart,
        LAST(UP.endTime) as rushEnd
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (UP{4,} DOWN{2,} E)
        DEFINE
        UP AS UP.rideCount > PREV(UP.rideCount)
            or PREV(UP.rideCount) IS NULL,
                DOWN AS DOWN.rideCount <
                LAST(UP.rideCount),
                E AS E.rideCount  >
              LAST(DOWN.rideCount)
)
```

```
Pattern.<Row>begin("UP", SKIP_PAST_LAST).timesOrMore(4).where((row, ctx) ->
{
                Long prevRideCount = ctx.get("UP").tail().get("rideCount");
                return prevRideCount == null || row.get("rideCount") >
prevRideCount;
        }).next("DOWN").timesOrMore(2).where( (row, ctx) -> {
                ...
        }).next("E").where( (row, ctx) -> {
        ...
        });

CEP.pattern(input.keyBy("cellId"), pattern).flatSelect(
new PatternFlatSelectFunction<Row, Row>() {
        @Override
        public void flatSelect(Map<String, List<Row>> pattern,
        Collector<Row> out) throws Exception {
                Row last = pattern.get("UP").tail();
                Row first = pattern.get("UP").headl();
                        out.collect(Row.of(...));
                }
);
```


THANK YOU!

@dwysakowicz

@dataArtisans

@ApacheFlink

WE ARE HIRING

data-artisans.com/careers

dataArtisans