Implementing Applications Using Streaming SQL

Ivan Mushketyk

@mushketyk brewing.codes

Kinesis Analytics

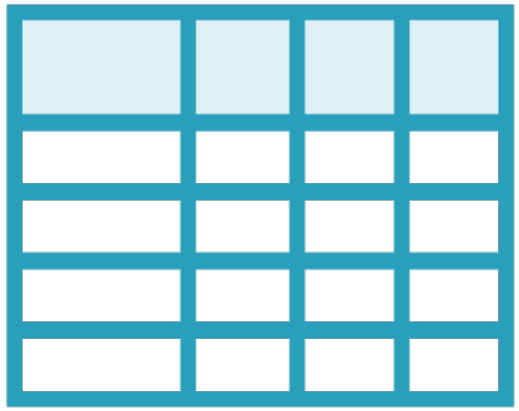
Kinesis Streams

Kinesis Analytics

Kinesis Firehose

Kinesis Analytics

Why Use Kinesis Analytics



ANSI SQL:2008 support

Auto-scalable

Real-time processing

Easy to use

What We Will Implement

Streaming filter

Processing windows

Analytics functions

JOINs on streams

Enjoying the Course?

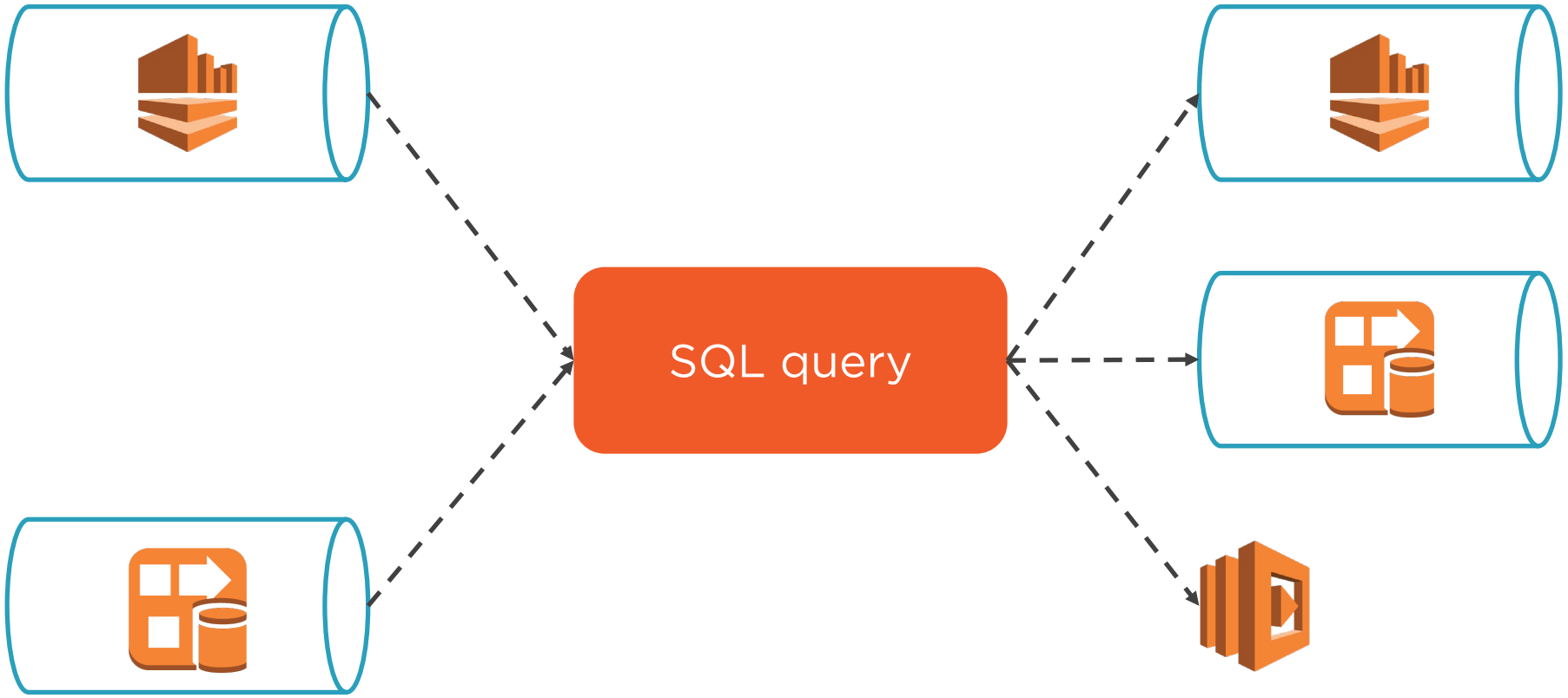
Rate the course

Follow me on Pluralsight

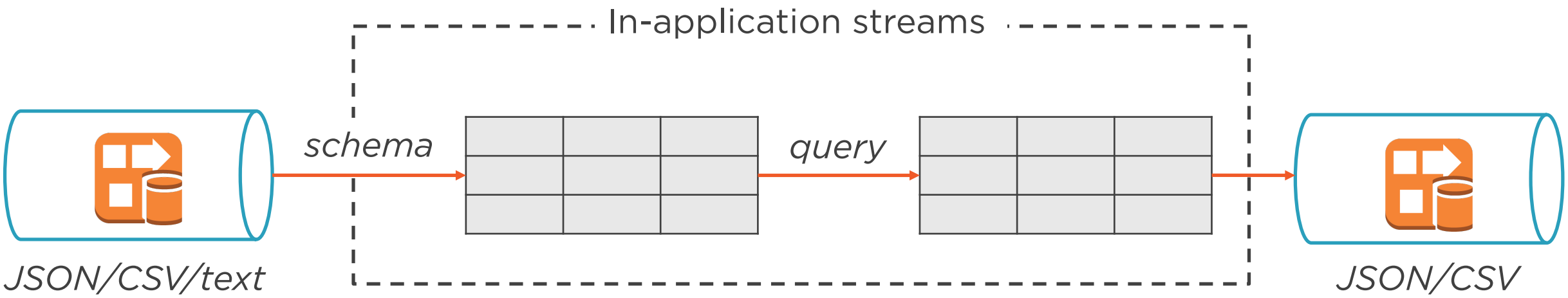
Connect me on Twitter/LinkedIn

- @mushketyk

Kinesis Analytics



Flow of Data



CSV Schema

Kinesis records

```
1,The Dark Knight,9.0
2,The Godfather,9.1
3,Alien,8.8
```

Schema

position	type	name
1	INTEGER	id
2	VARCHAR	movie
3	DOUBLE	rating

id	movie	rating
1	The Dark Knight	9.0
2	The Godfather	9.1
3	Alien	8.8

In-application stream

JSON Path

Kinesis record


```
{
    "id": 123,
    "location": {
        "city": "London"
    },
    "records": [
        {
            "value": 1
        },
        {
            "value": 2
        }
    ]
}
```


Schema

path	type	name
$.id	INTEGER	id

id
123

In-application stream

JSON Path

Kinesis record

```
{
    "id": 123,
    "location": {
        "city": "London"
    },
    "records": [
        {
            "value": 1
        },
        {
            "value": 2
        }
    ]
}
```

Schema

path	type	name
$.id	INTEGER	id
$.location.city	VARCHAR	city

id	city
123	London

In-application stream

JSON Path

Kinesis record

```
{
    "id": 123,
    "location": {
        "city": "London"
    },
    "records": [
        {
            "value": 1
        },
        {
            "value": 2
        }
    ]
}
```

Schema

path	type	name
$.id	INTEGER	id
$.location.city	VARCHAR	city
$.records[0:]	VARCHAR	values

id	city	values
123	London	[{"value"}: 1 ...

In-application stream

JSON Path

Kinesis record

```
{
    "id": 123,
    "location": {
        "city": "London"
    },
    "records": [
        {
            "value": 1
        },
        {
            "value": 2
        }
    ]
}
```

Schema

path	type	name
$.id	INTEGER	id
$.location.city	VARCHAR	city
$.records[0:].value	INTEGER	values

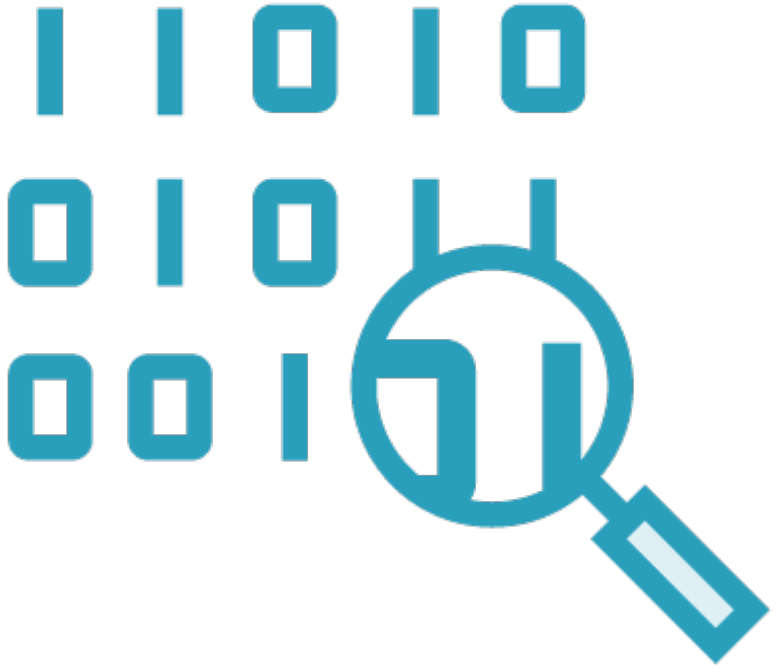
id	city	values
123	London	1
123	London	2

In-application stream

JSON Path Limitations

```
// NOT SUPPORTED
$.sensors[0:].readings[0:].value

// NOT SUPPORTED
$.sensors[0:].name
$.readings[0:]
```

Schema Discovery

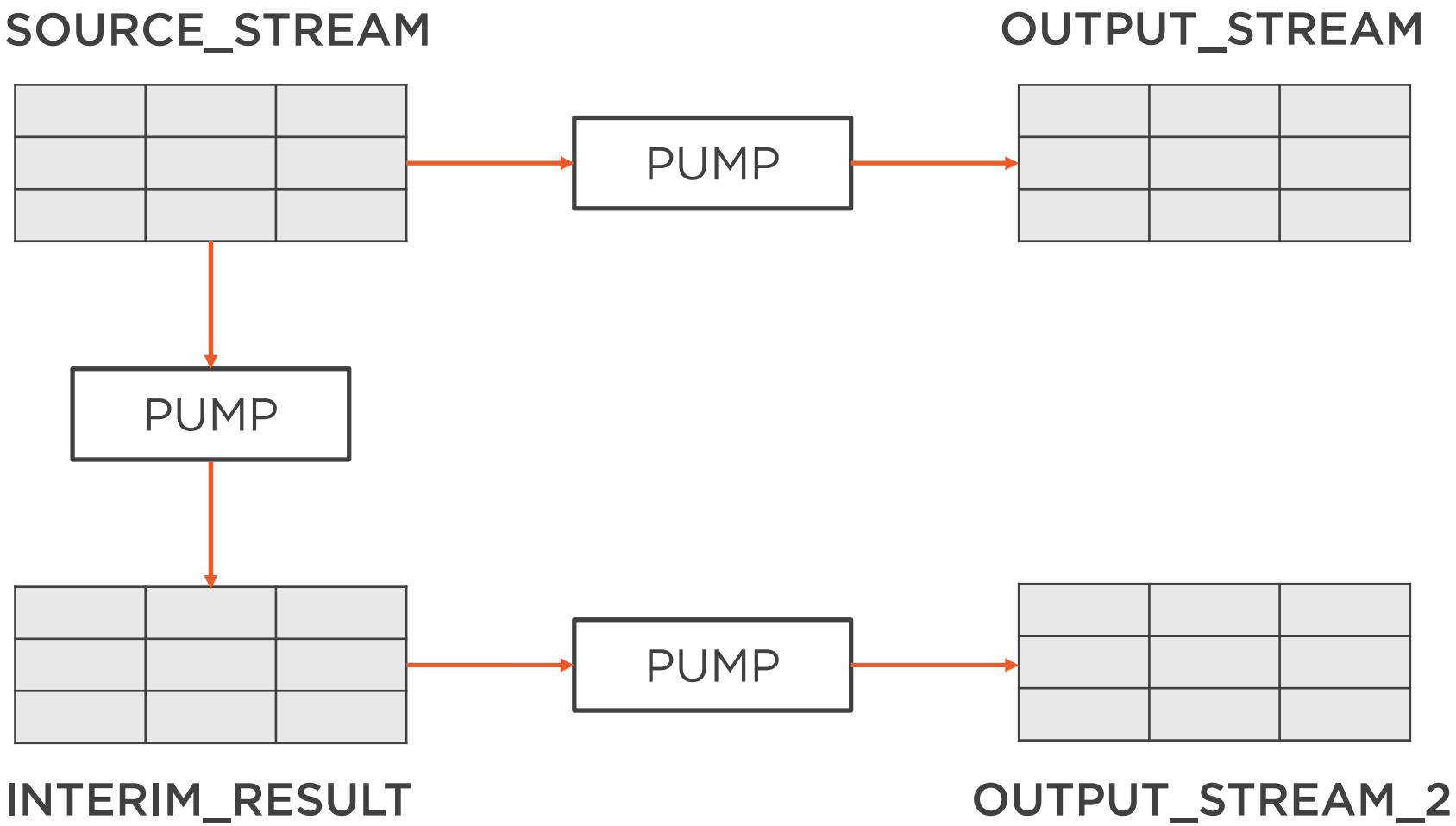


Kinesis automatically discovers schema

Supports CSV/JSON

Good starting point

Streams and Pumps



SQL Query Example

```
CREATE OR REPLACE STREAM "DESTINATION_STREAM" (
   "id" VARCHAR(32),
   "temperature" INTEGER);
CREATE OR REPLACE PUMP "STREAM_PUMP"
   AS INSERT INTO "DESTINATION_STREAM"
SELECT STREAM "id", "temperature"
FROM "SOURCE_SQL_STREAM_001”
WHERE "temperature" > 35;
```

Log Parsing Functions

```
-- Log formats
W3C_LOG_PARSE(string, format_name)
SYS_LOG_PARSE(string)
-- Regex parsing
REGEX_LOG_PARSE(string, regex, columns)
FAST_REGEX_LOG_PARSE(string, regex)
-- Fixed width format
FIXED_COLUMN_LOG_PARSE(string, columns)
-- Parse fields separated with a delimiter
VARIABLE_COLUMN_LOG_PARSE()
```

Additional Data

Additional Kinesis Analytics Fields

- ROWTIME
- Shard id
- Sequence number
- Partition key
- Approximate arrival time

Error in-application stream

Lambda Preprocessing

Aggregate Functions

Average

Count

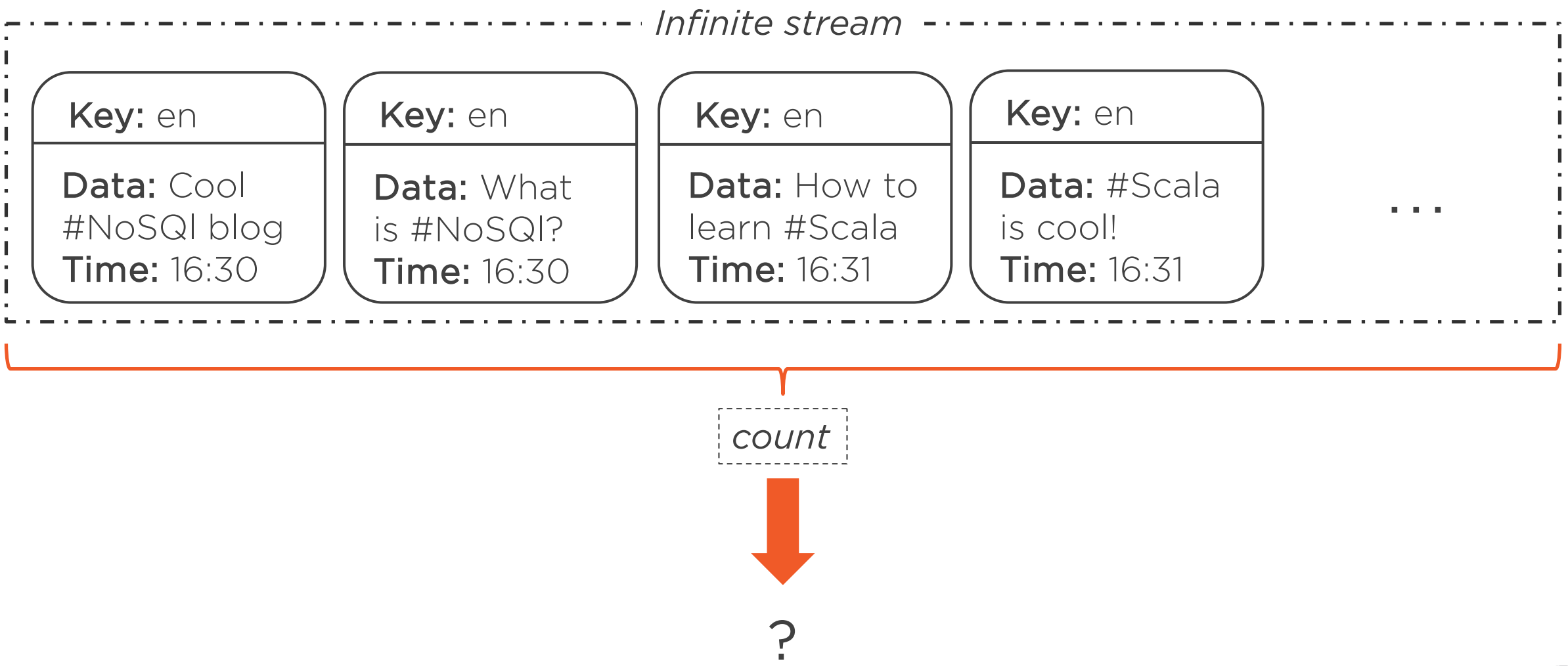
Count distinct

Min

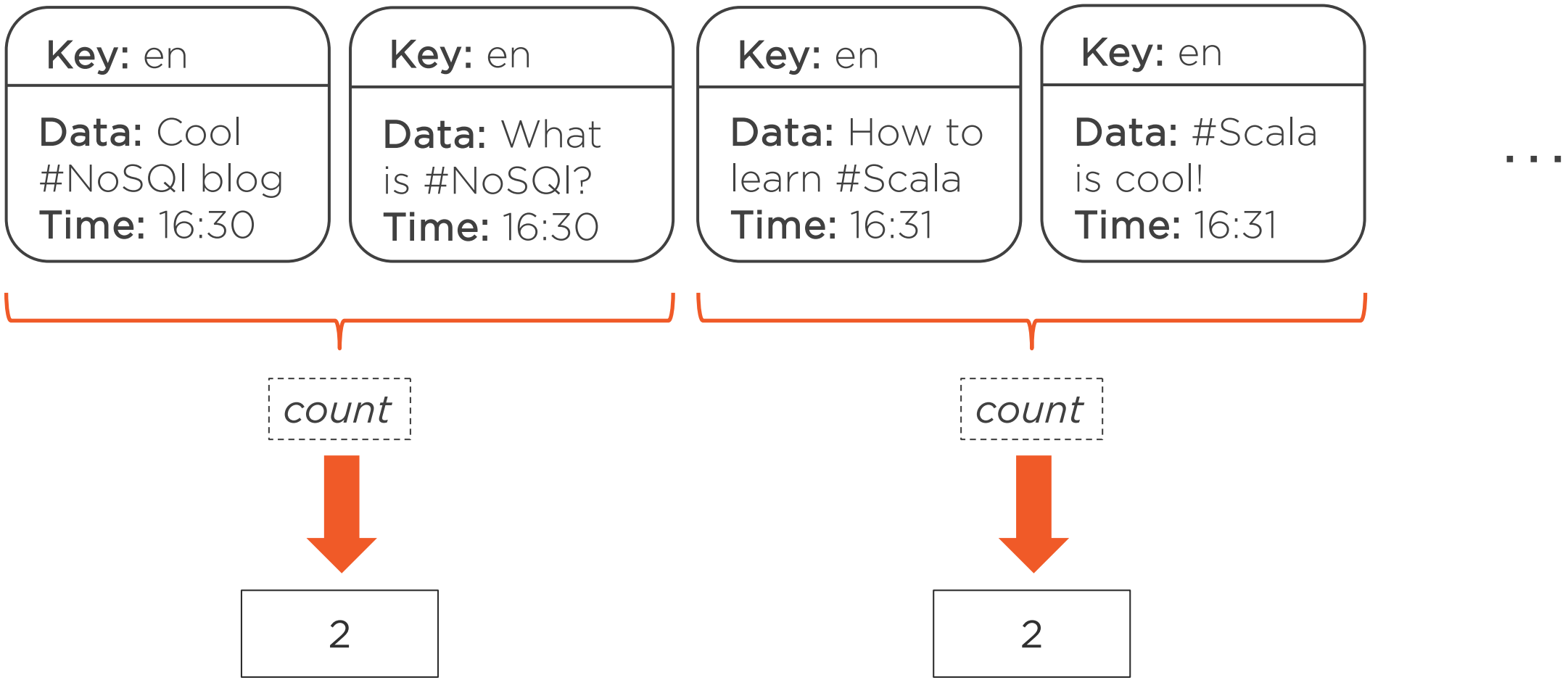
Max

Sum

Aggregate Functions



Aggregate Functions



Windows in Kinesis Analytics

Tumbling Window

Non-overlapping

Output results at fixed intervals

Sliding Windows

Overlapping

Output results when outcome changes

Tumbling Windows

count

count	time
3	16:30

count

count	time
2	16:31

Query Example

```
SELECT STREAM COUNT(*) AS "count"
FROM TweetsStream
GROUP BY
    FLOOR(TweetsStream.ROWTIME TO MINUTE);
    -- 15th Jan 2018 18:30:01 -> 15th Jan 2018 18:30:00
    -- 15th Jan 2018 18:30:25 -> 15th Jan 2018 18:30:00
    -- 15th Jan 2018 18:30:58 -> 15th Jan 2018 18:30:00
```

Query Example with the STEP Function

```
SELECT STREAM COUNT(*) AS "count"
FROM TweetsStream
GROUP BY
  STEP(TweetsStream.ROWTIME BY INTERVAL '30' SECOND);
```

Partition by Column

count

lang	count	time
en	2	16:30
fr	1	16:30

count

lang	count	time
en	1	16:31
jp	1	16:31

Query Example

```
SELECT STREAM "lang", COUNT(*) AS "count”
FROM TweetsStream
GROUP BY
  "lang",
  FLOOR(TweetsStream.ROWTIME TO MINUTE);
```

Time in Tumbling Windows

ROWTIME

Approximate arrival time

Creation time

Monotonic Time

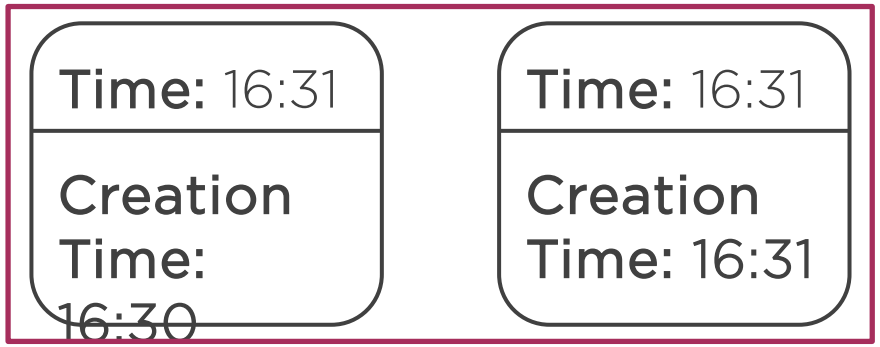
Monotonic time

- ROWTIME
- Approximate arrival time

Solutions

- Use MONOTONIC function
- Group using two time fields

Grouping Using Two Time Fields



count

count

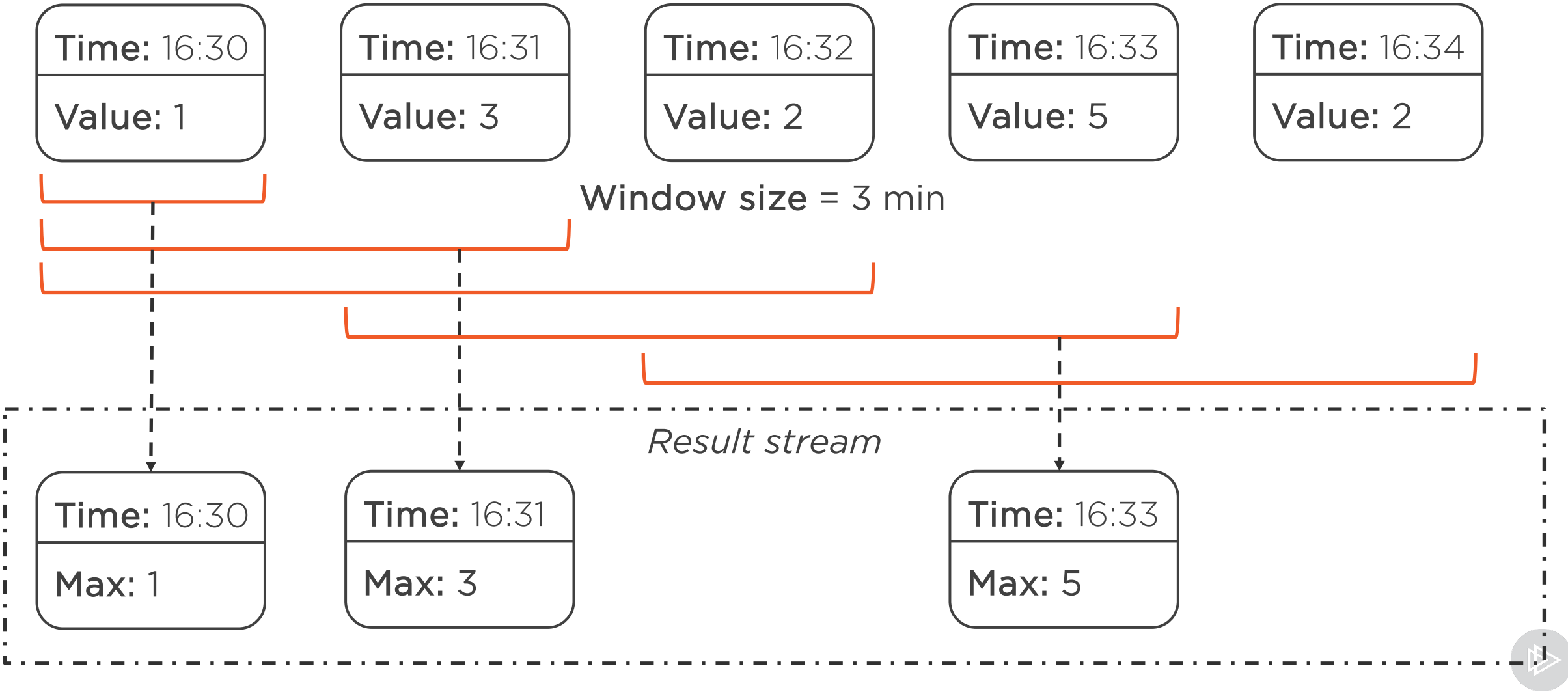
time	created at	count
16:30	16:30	2
16:30	16:29	1

time	created at	count
16:31	16:31	1
16:31	16:30	1

Group by Creation Time

```
SELECT STREAM
  FLOOR(TweetsStream.ROWTIME TO MINUTE) AS ingestion_time,
  FLOOR(TweetsStream.creation_time TO MINUTE) AS creation_time,
  COUNT(*) AS "count"
FROM TweetsStream
GROUP BY
  FLOOR(TweetsStream.ROWTIME TO MINUTE)
  FLOOR(TweetsStream.creation_time TO MINUTE);
```

Sliding Window



Sliding Window Computation

When the first item is added to the window

When a new item is added and result is changed

When a new item is added and result is not changed

When an old item is removed

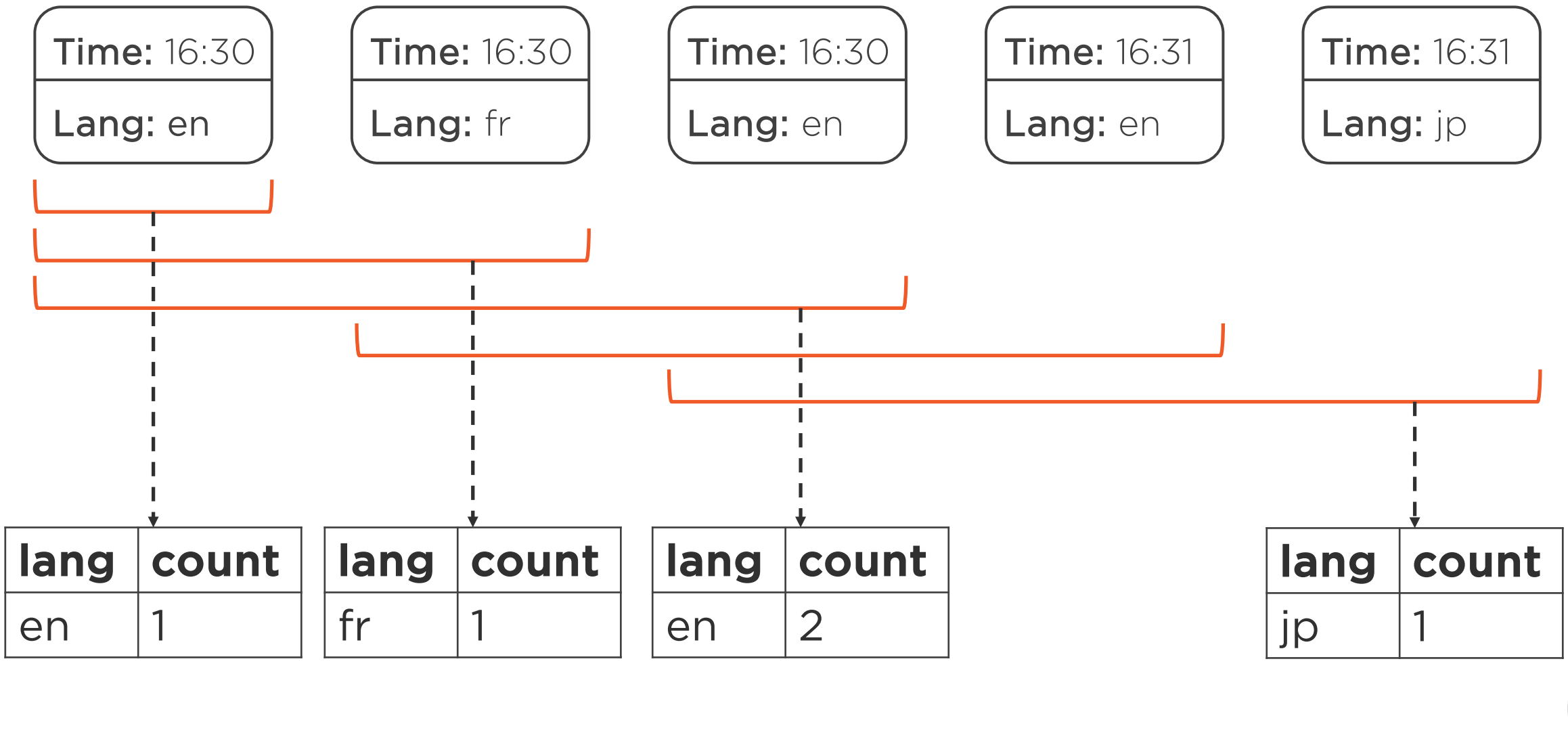
Time Sliding Window

```
SELECT STREAM
  COUNT(*) OVER rangeWindow AS "count"
FROM tweetsStream
WINDOW rangeWindow AS (
  RANGE INTERVAL '10' SECOND PRECEDING);

SELECT STREAM
  COUNT(*) OVER (
    RANGE INTERVAL '10' SECOND PRECEDING) AS "count"
FROM tweetsStream;
```

Row Sliding Window

```
SELECT STREAM
  AVG(CHAR_LENGTH("text")) OVER last100 AS "count"
FROM tweetsStream
WINDOW last100 AS (
  ROWS 100 PRECEDING);
```

Partitioning in Sliding Windows



Partitioning in Sliding Windows

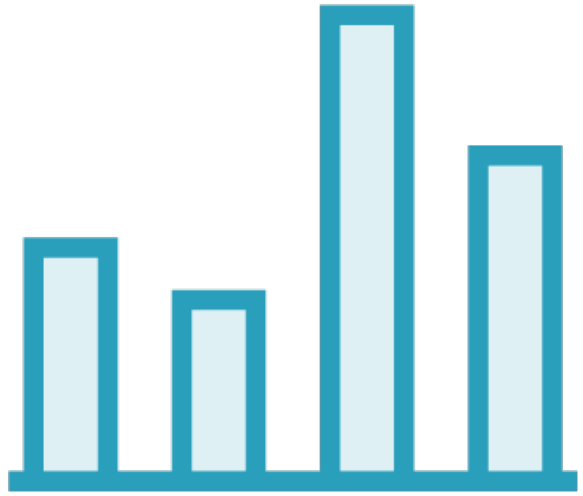
```
SELECT STREAM
  "lang”,
  COUNT(*) OVER rangeWindow AS "count"
FROM tweetsStream
WINDOW rangeWindow AS (
  PARTITION BY "lang"
  RANGE INTERVAL '10' SECOND PRECEDING);
```

Multiple Sliding Windows

```
SELECT STREAM
  COUNT(*) OVER last10Seconds AS "count_in_10”,
  COUNT(*) OVER last60Seconds AS "count_in_60”
FROM tweetsStream
WINDOW last10Seconds AS (
  RANGE INTERVAL '10' SECOND PRECEDING),
WINDOW last60Seconds AS (
  RANGE INTERVAL ’60' SECOND PRECEDING);
```

Kinesis Analytics Functions

Aggregate functions



Statistical functions

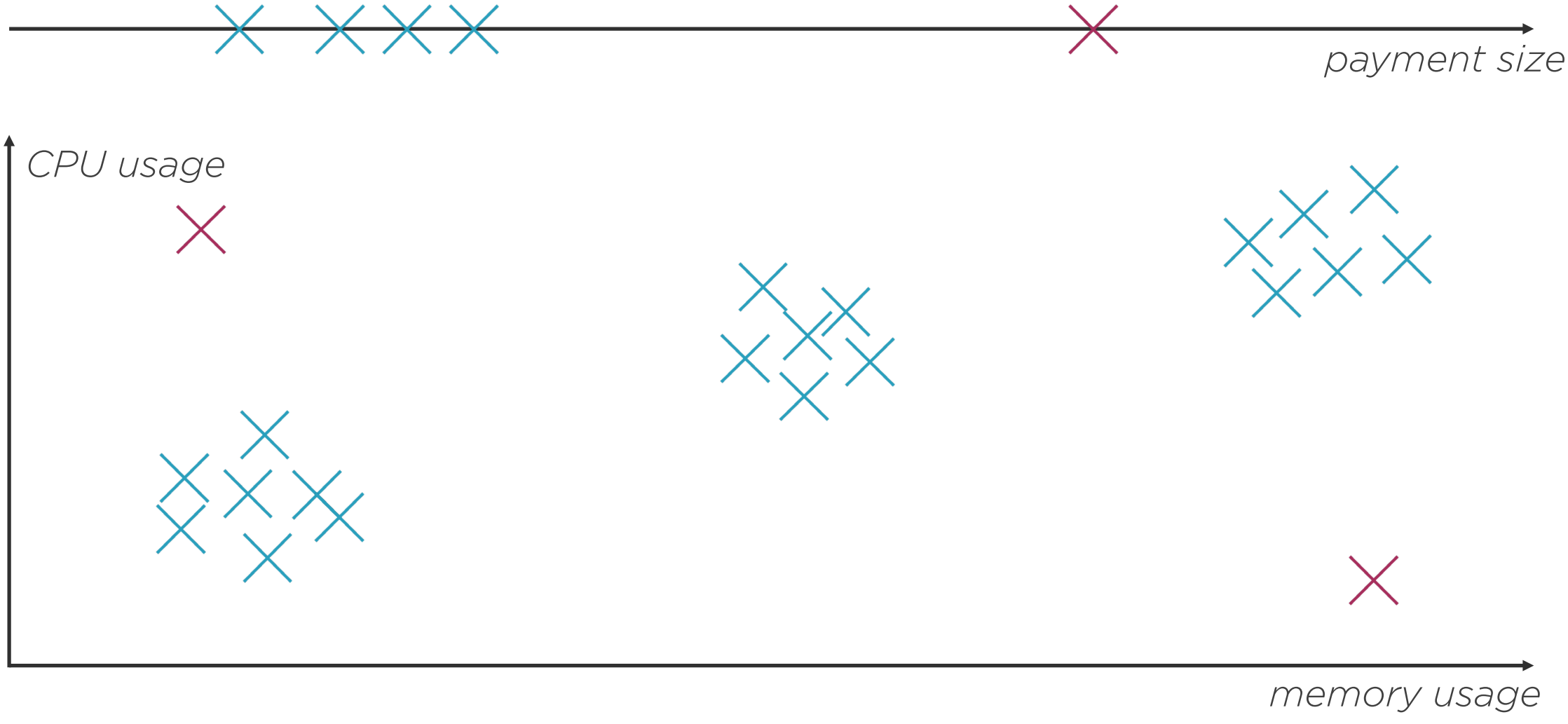
Aggregation Function

```
-- Mathematical functions

AVG()

COUNT()

MAX() / MIN()

-- Kinesis Analytics Functions

COUNT_DISTINCT_ITEMS_TUMBLING(
  stream, columnName, windowSize)
-- Approximate algorithm to find top items

TOP_K_ITEMS_TUMBLING(stream, columnName, K, windowSize)
```

Statistical Functions

```
SELECT STREAM *
FROM TABLE (
  TOP_K_ITEMS_TUMBLING(
    CURSOR(SELECT STREAM * FROM "SOURCE_STREAM"),
    'num_of_retweets', -- name of column
    10,                -- num. of the most frequent items
    60                 -- window size
  )
);
```

Anomaly Detection



Detecting Anomalies

```
RANDOM_CUT_FOREST(
  -- Up to 30 numerical columns
  CURSOR(SELECT STREAM columnName FROM inputStream),
  numberOfTrees = 100,
  subSampleSize = 256, -- to construct a single tree
  timeDecay = 100000,  -- num. of records to consider
  shingleSize = 1,     -- size of a record
  withDirectionality = FALSE -- tells direction
)
-- Score = 0 – no anomalies
-- Score significantly greater than 0 – anomaly
```

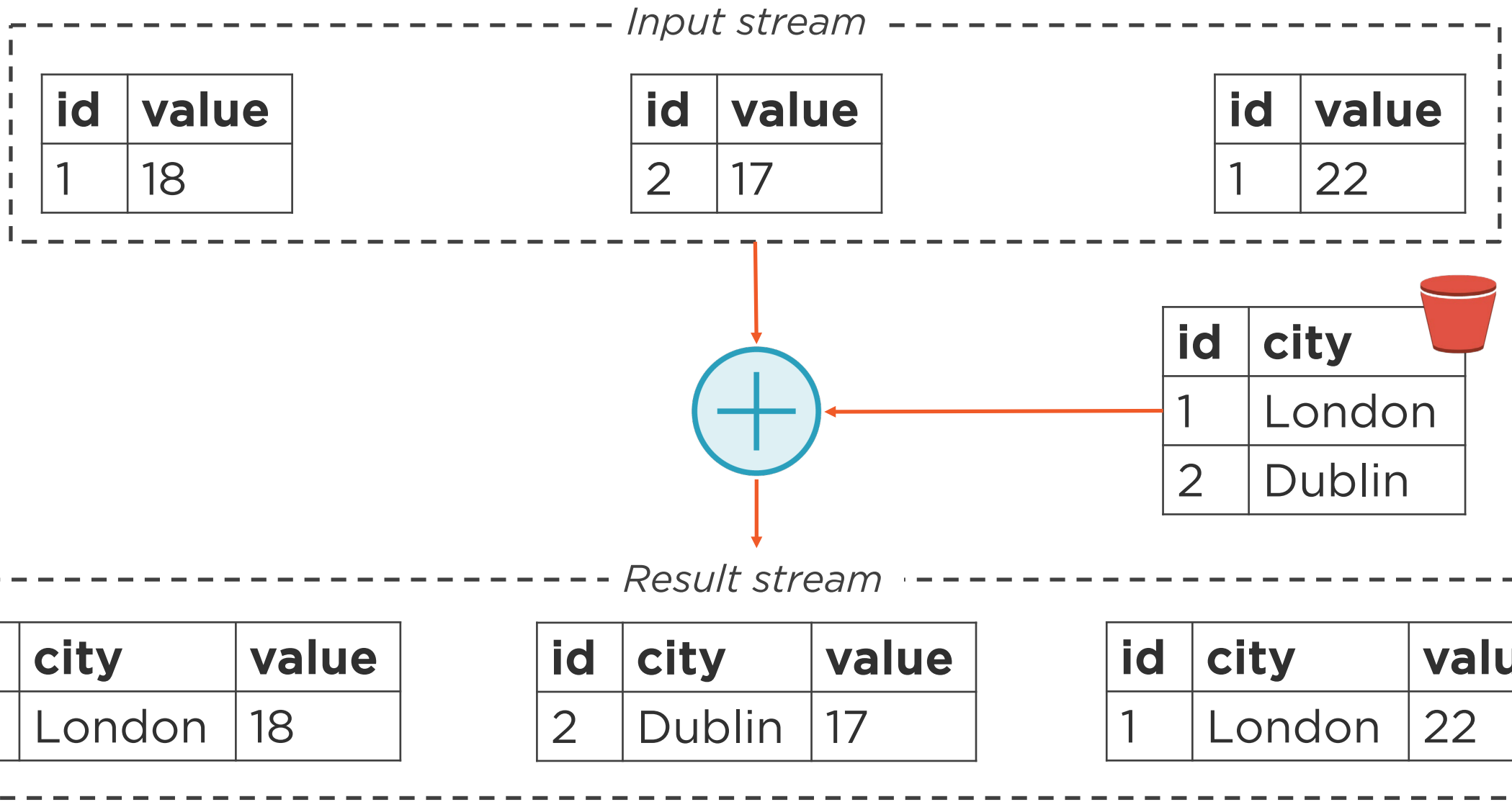
Stream Joins

Static data from S3 bucket

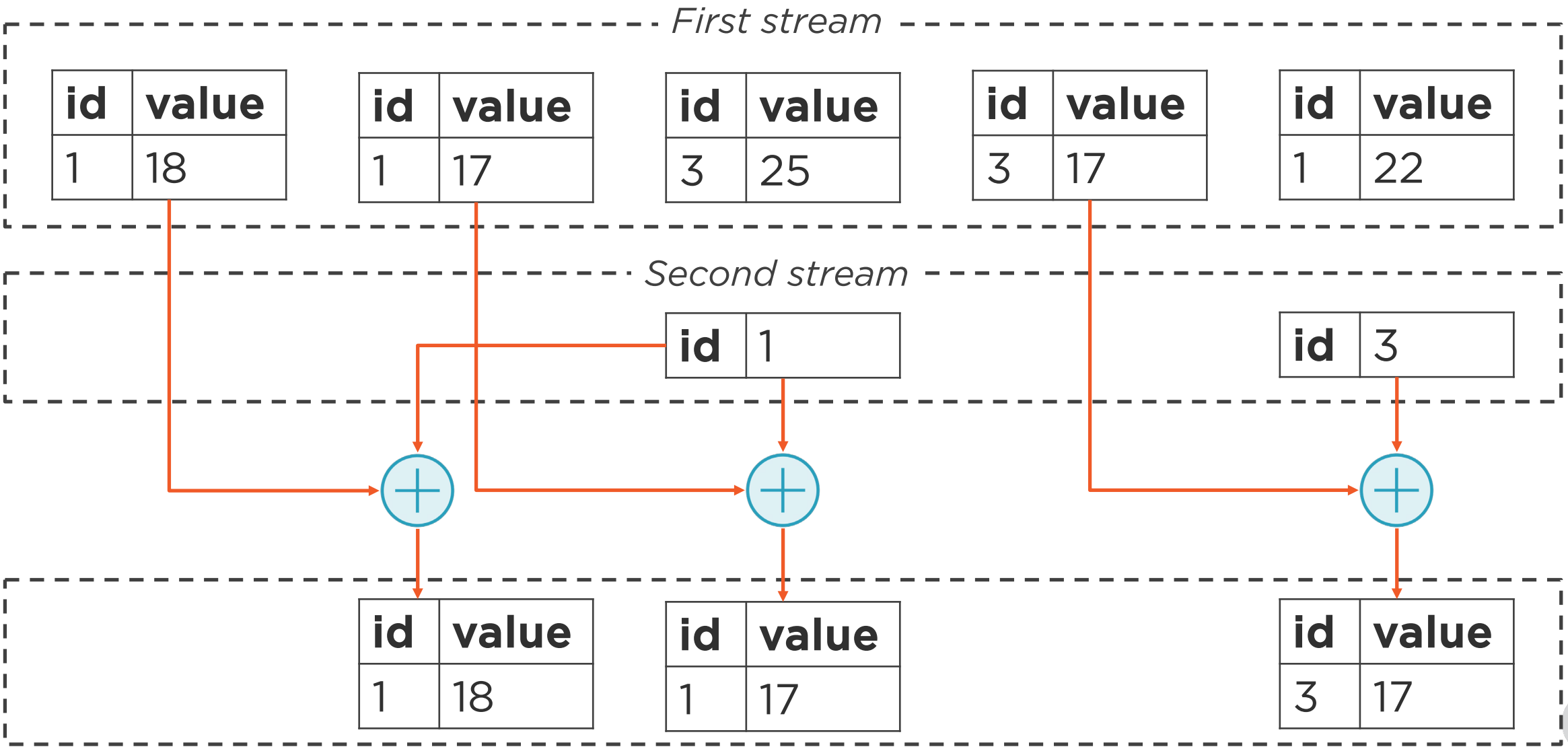
Enrich data stream

Join two streams

Join with Static Data



Join with Stream Data



Filter Removed Tweets

```
SELECT STREAM ”id", ”text”
FROM TWEETS_STREAM AS source
JOIN REMOVED_TWEETS_IDS OVER
  (RANGE INTERVAL '5' MINUTE PRECEDING) AS removed
  ON removed."id" = source."id";
```

Kinesis Analytics Pricing

Pay for used computing resources

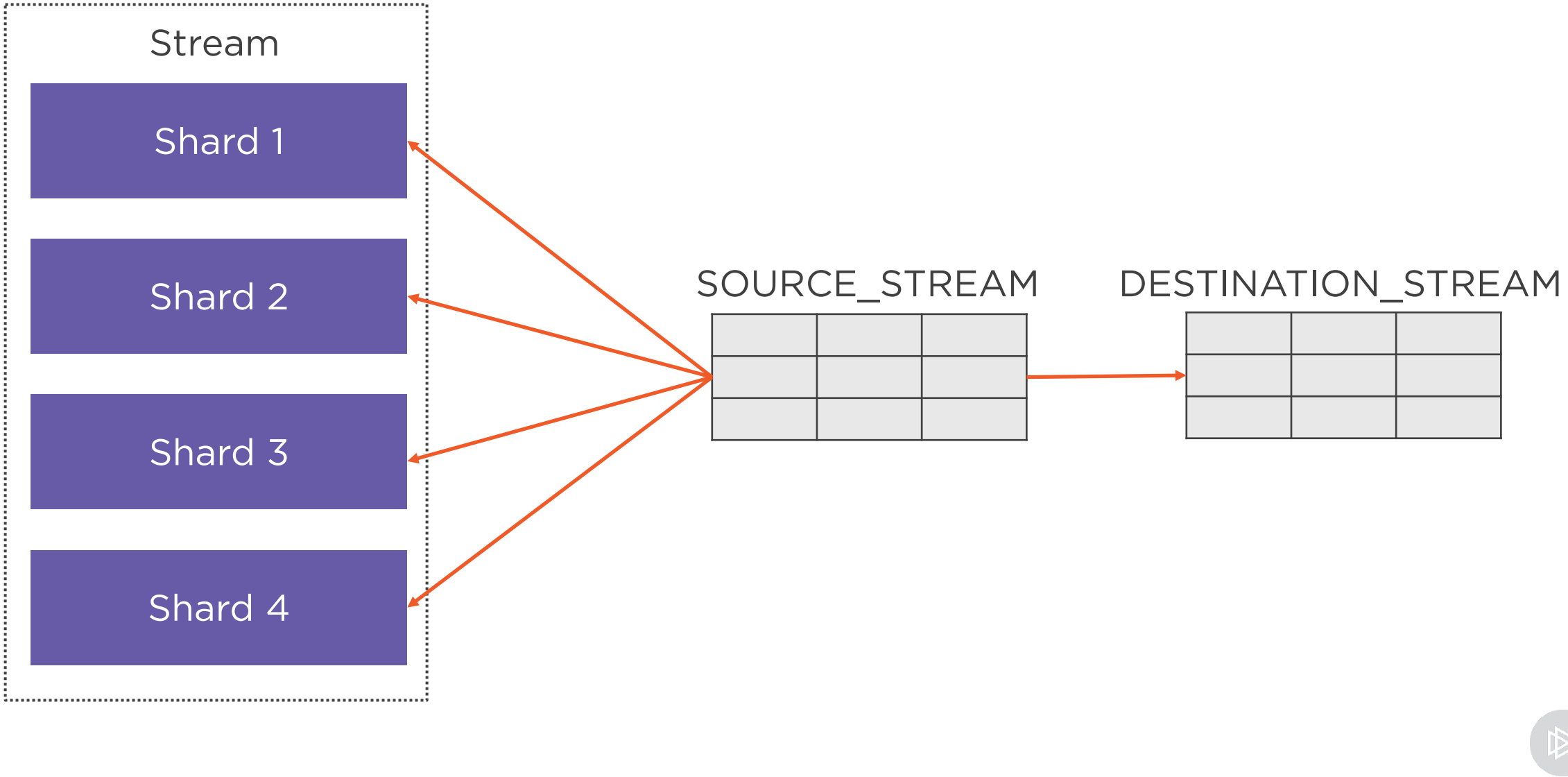
Kinesis Processing Units (KPUs)

- 1 vCPU
- 4 GB of memory
- $0.110 per hour

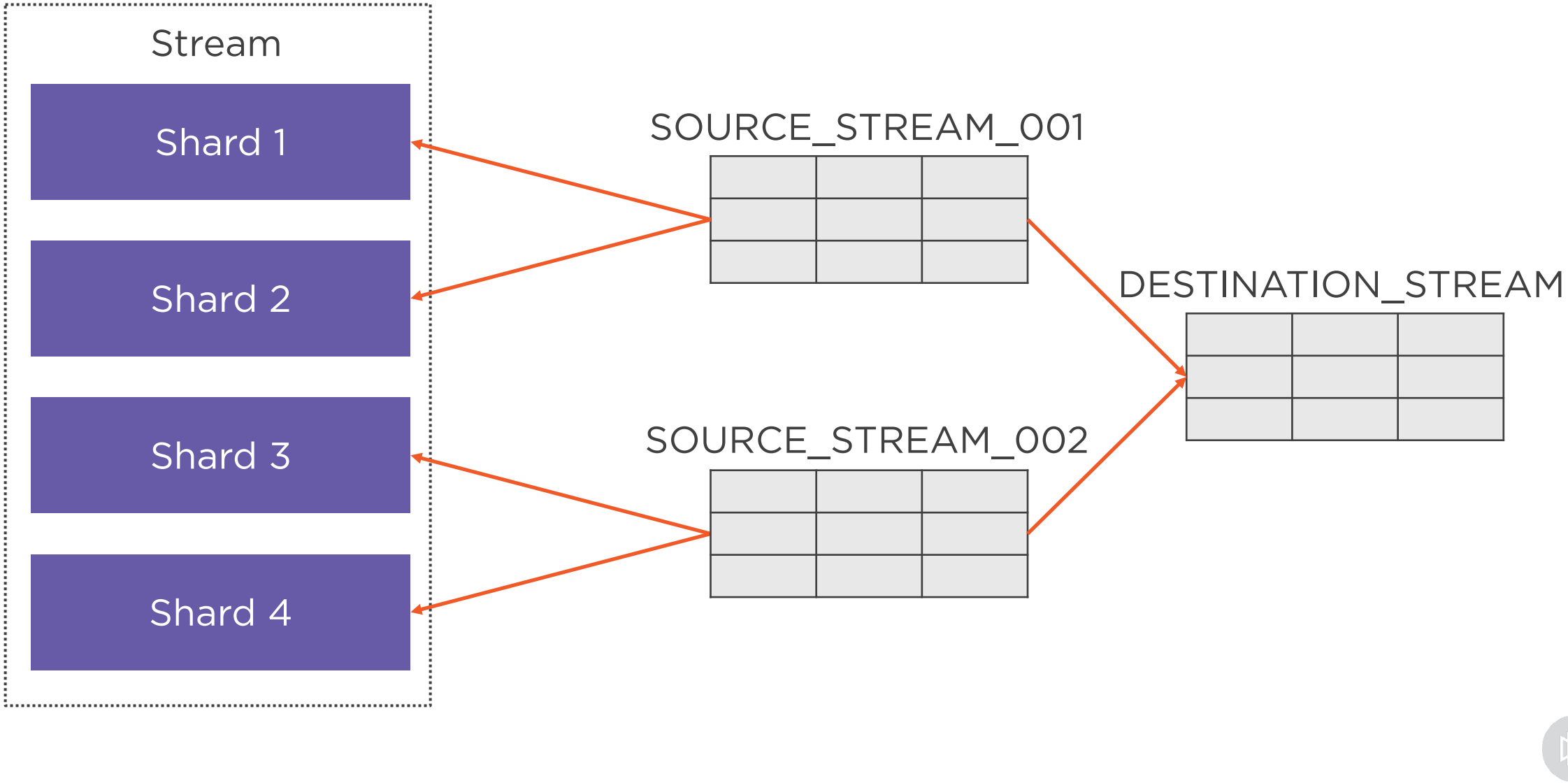
More windows -> more memory

More items -> more CPUs

Source Parallelism



Source Parallelism



When to Use Source Parallelism

Check MillisBehindLatest metric in CloudWatch

MillisBehindLatest is increasing

MillisBehindLatest is above 1 second

Use Kinesis Analytics API or AWS CLI to set parallelism

Source Parallelism Example

```
CREATE OR REPLACE STREAM RESULT ...;

CREATE OR REPLACE PUMP PUMP_1 AS INSERT INTO RESULT
SELECT STREAM * FROM SOURCE_SQL_STREAM_001
WHERE ...;

CREATE OR REPLACE PUMP PUMP_2 AS INSERT INTO RESULT
SELECT STREAM * FROM SOURCE_SQL_STREAM_002
WHERE ...;
```

Kinesis Limitations

Row in a stream limited to 50KB

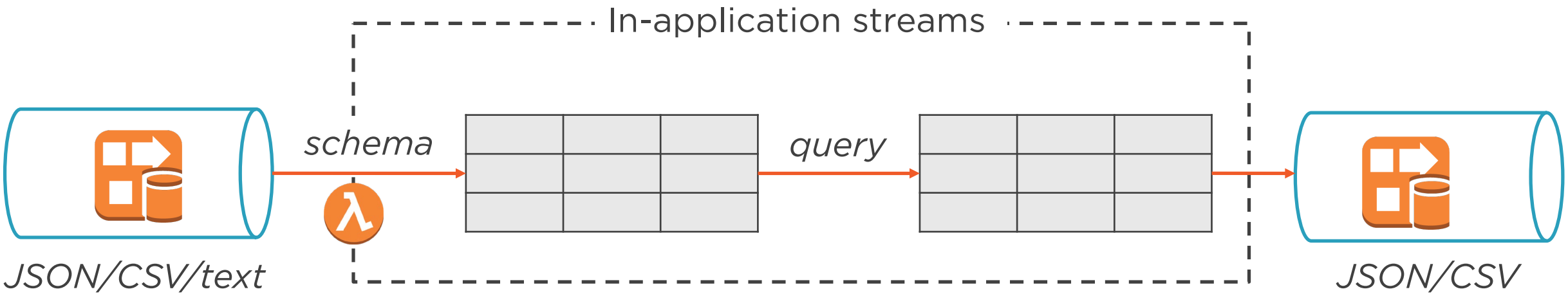
50 Kinesis Analytics applications

Inputs and outputs

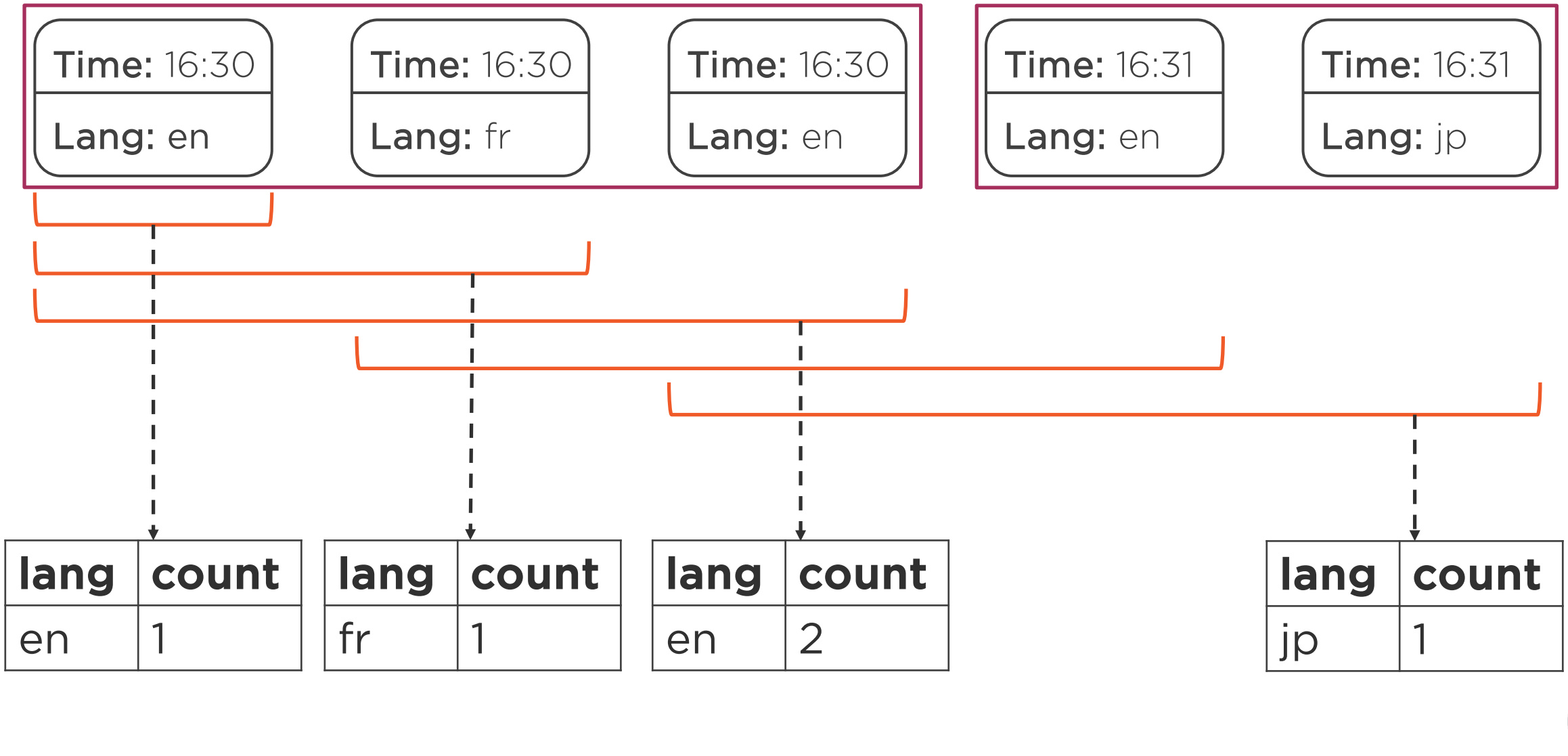
- One streaming source
- One reference source (up to 1 GB)
- Three destinations
- Maximum source parallelism: 64

Up to 8 KPUs (soft limit)

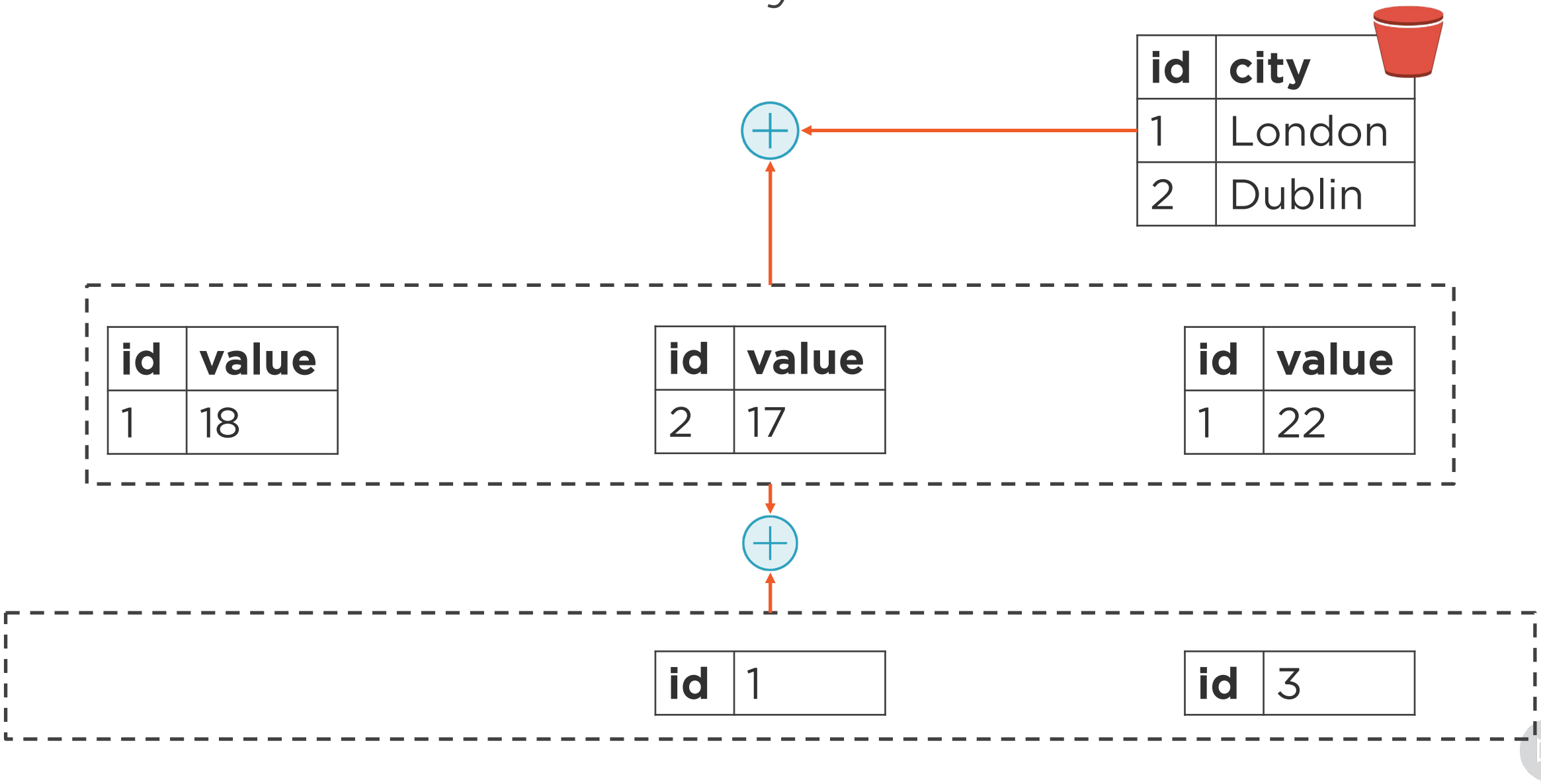
Summary - Kinesis Analytics



Summary - Windows



Summary - Joins



Thank You!

- Leave a comment
- Rate the course
- Share the course on social media
- Follow me on Twitter @mushketyk
- Follow me on Pluralsight