



# Deep Dive into Concepts and Tools for Analyzing Streaming Data

Dr. Steffen Hausmann

Sr. Solutions Architect, Amazon Web Services

# Data originates in real-time



Photo by mountainamoeba

<https://www.flickr.com/photos/mountainamoeba/2527300028/>

# Analytics is done in batches



Photo by PracticalHacks

<https://www.flickr.com/photos/29225844@N05/2828724211>



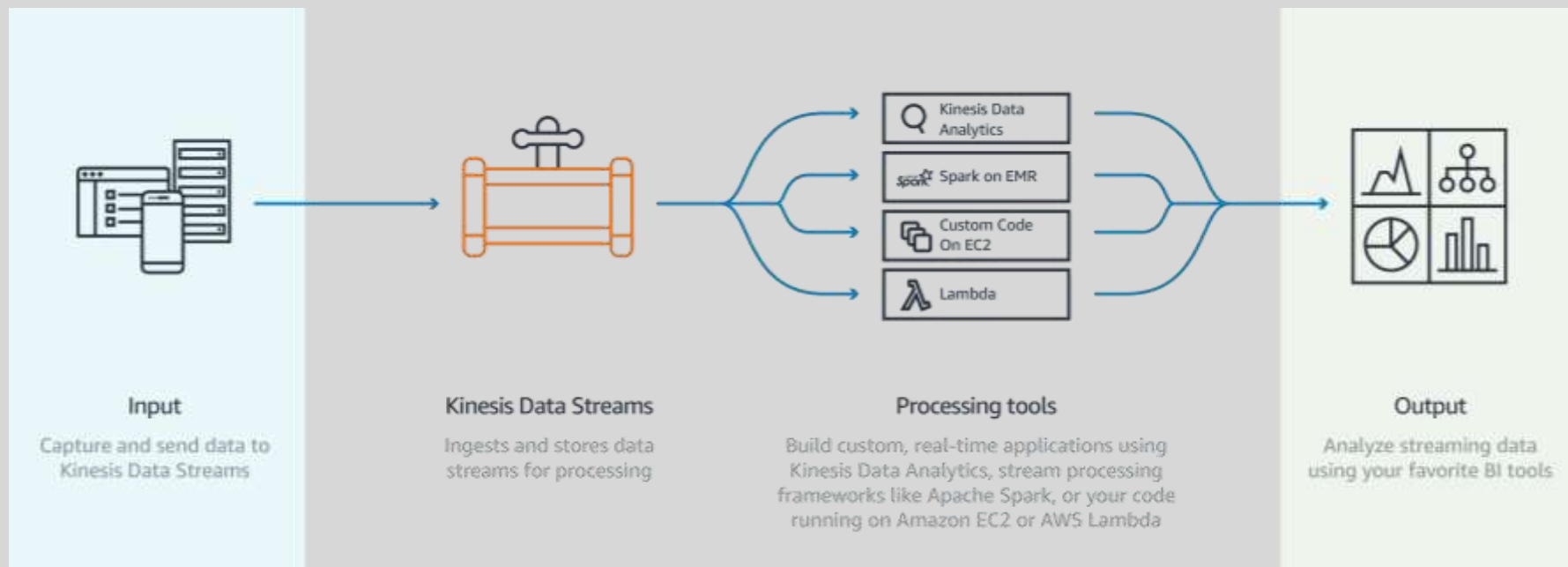
# Insights are Perishable



Photo by Lucas Cobb

<https://www.flickr.com/photos/cobblucas/4780005097/>

# Analyzing Streaming Data on AWS







# Challenges of Stream Processing

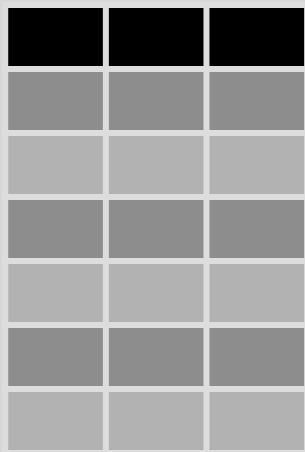
Photo by Follow your Nose

<https://www.flickr.com/photos/laprimadonna/3294167673>

# Comparing Streams and Relations

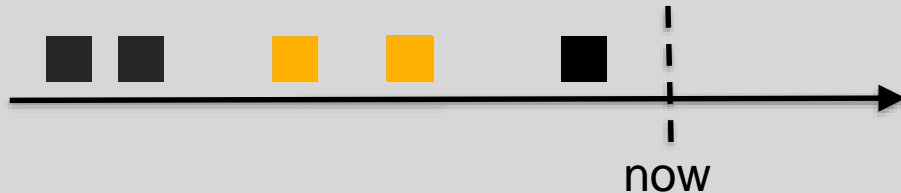
**Relation**

$$R \subseteq Id \times Color$$



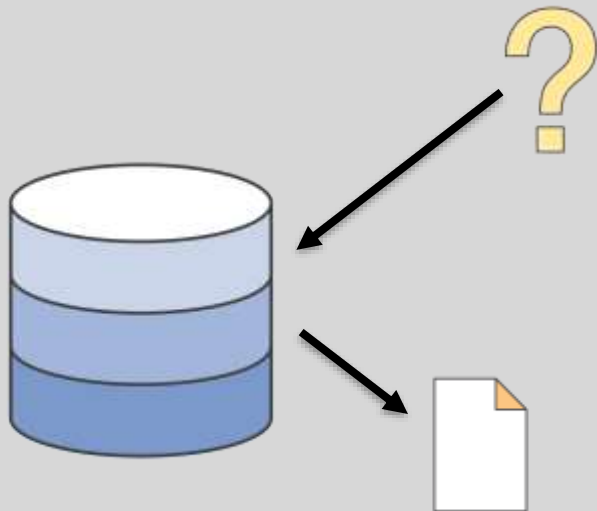
**Stream**

$$S \subseteq Id \times Color \times Time$$



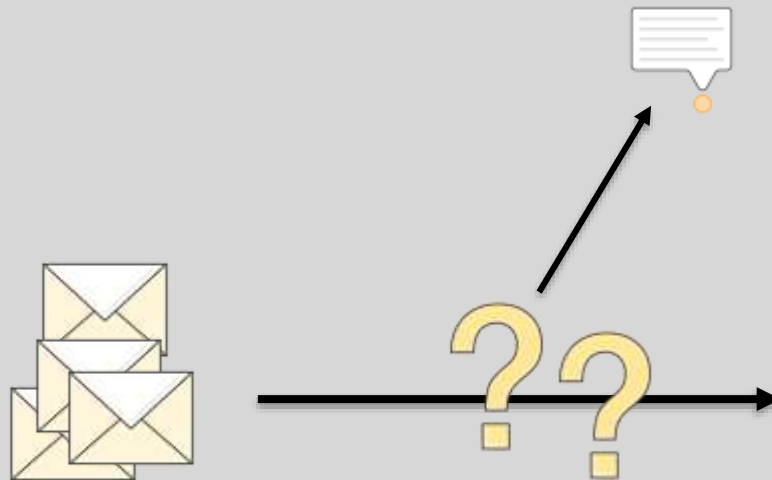
# Querying Streams and Relations

## Relation



Fixed data and ad-hoc queries

## Stream



Fixed queries and  
continuously ingested data



# Challenges of Querying Infinite Streams

```
SELECT * FROM S WHERE color = 'black'
```

```
SELECT * FROM S JOIN S'
```

```
SELECT color, COUNT(1) FROM S GROUP BY color
```

```
SELECT * FROM S WHERE color = 'red')
```

**Unbounded Query**

**Non-monotonic Operators**



**Mathias Verraes**

@mathiasverraes

Follow



There are only two hard problems in distributed systems:  
1. Guaranteed order of messages  
2. Exactly-once delivery

11:40 AM - 14 Aug 2015

7,103 Retweets 5,283 Likes



69



7.1K



5.3K

# Analyzing Streaming Data on AWS

## Amazon Kinesis Analytics

- Runs standard SQL queries on top of streaming data
- Fully managed and scales automatically
- Only pay for the resources your queries consume



## Apache Flink

- Open-source stream processing framework
- Included in Amazon Elastic Map Reduce (EMR)
- Flexible APIs with Java and Scalar, SQL, and CEP support







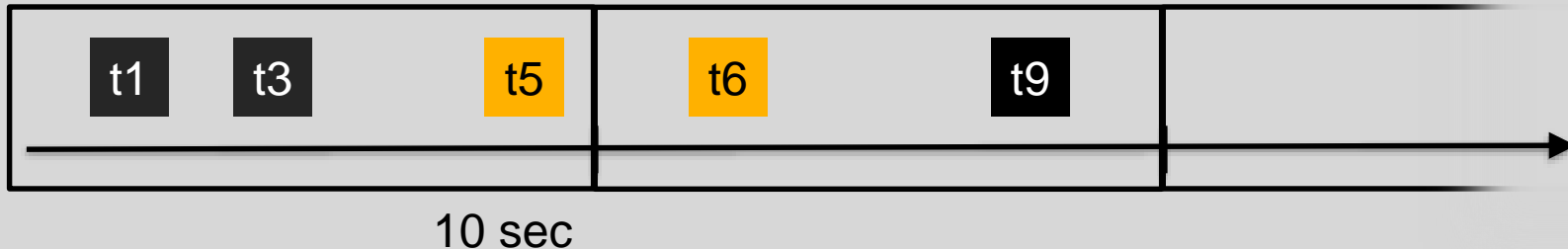
# Evaluating Queries over Streams

Photo by Brad Greenlee

<https://www.flickr.com/photos/bgreenlee/91309374/>

# Evaluating Non-monotonic Operators

## Tumbling Windows



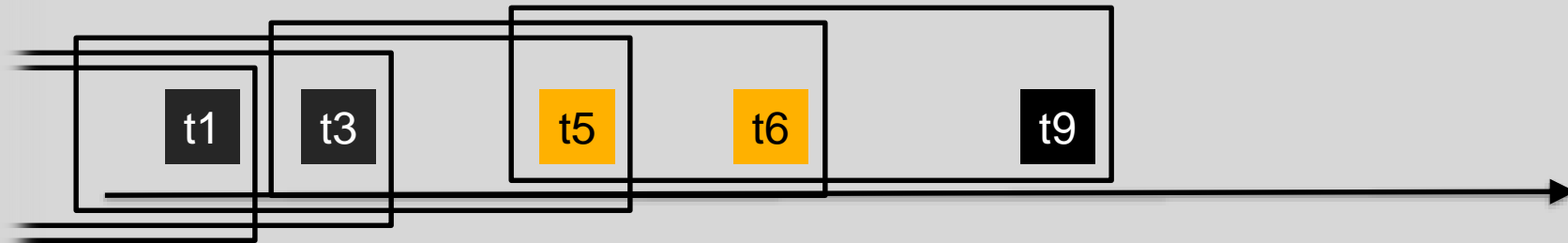
```
SELECT STREAM color, COUNT(1)
```

```
FROM ...
```

```
GROUP BY STEP(rowtime BY INTERVAL '10' SECOND), color;
```

# Evaluating Non-monotonic Operators

## Sliding Windows

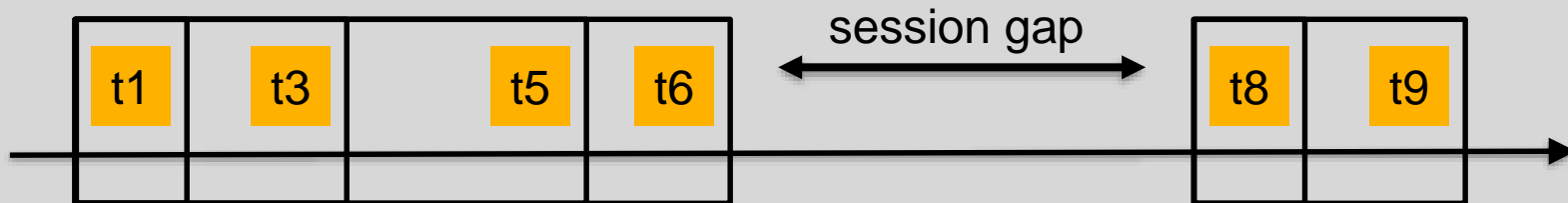


```
SELECT STREAM color, COUNT(1) OVER w
FROM ...
GROUP BY color
WINDOW w AS (RANGE INTERVAL '10' SECOND PRECEDING);
```



# Evaluating Non-monotonic Operators

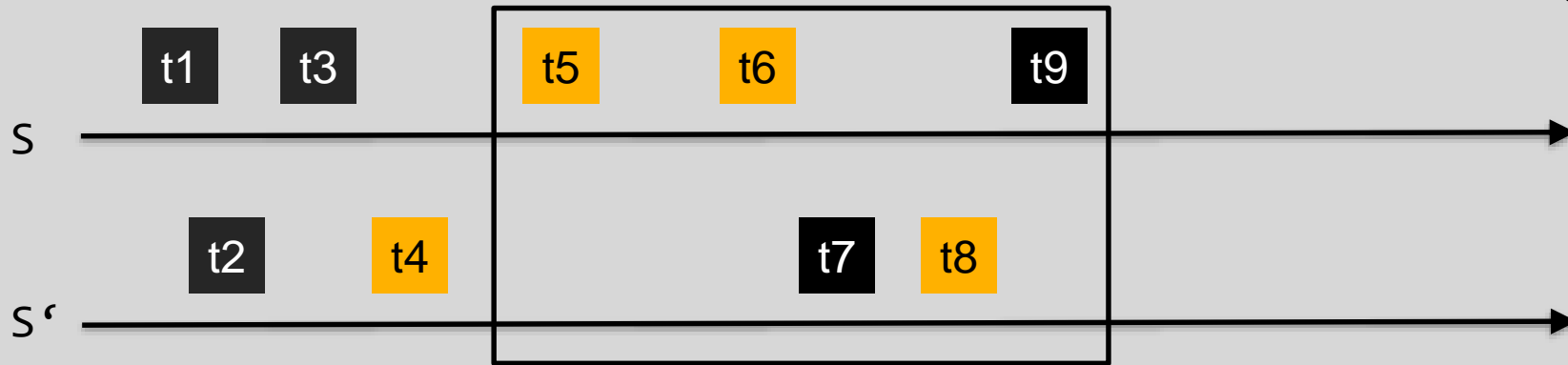
## Session Windows



stream

```
.keyBy(<key selector>)  
.window(EventTimeSessionWindows.withGap(Time.minutes(10)))  
.<windowed transformation>(<window function>);
```

# Evaluating Unbounded Queries



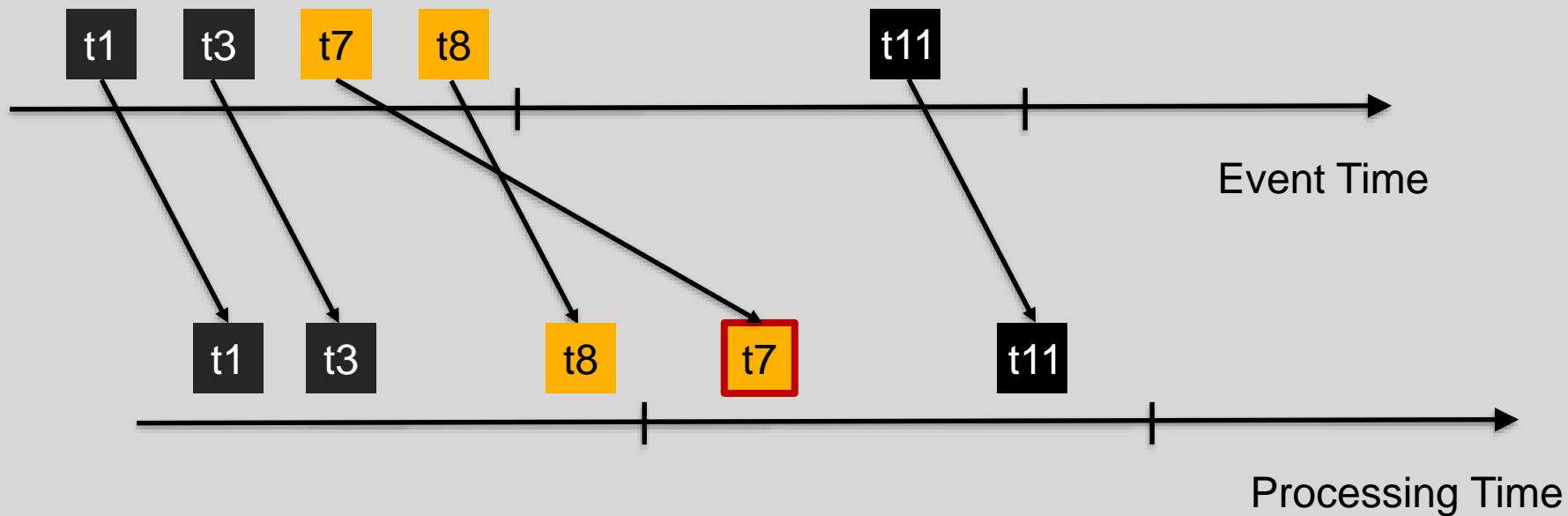
```
SELECT STREAM *  
FROM S OVER w AS s JOIN S' OVER w AS t  
ON s.color = t.color  
WINDOW w AS (RANGE INTERVAL '10' SECOND PRECEDING);
```



# **Different Time Semantics**

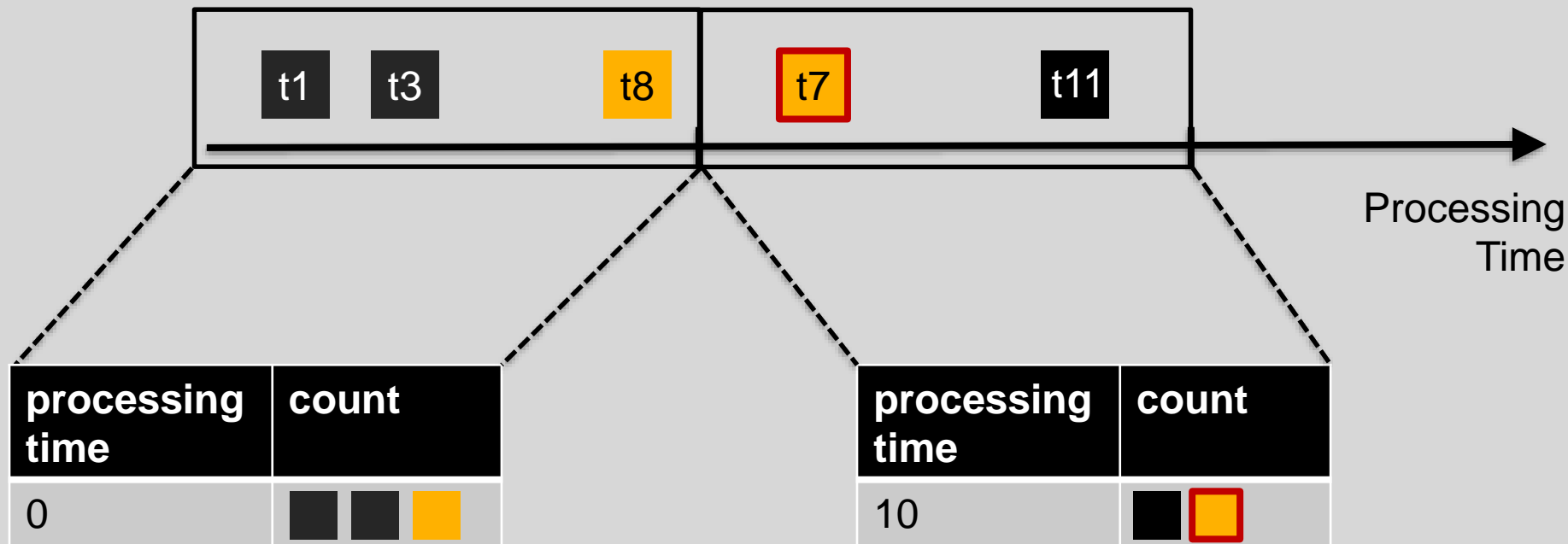


# Maintaining Order of Events



# Maintaining Order of Events

## Using processing time based windows



# Maintaining Order of Events

## Using multiple time-windows



```
SELECT STREAM
```

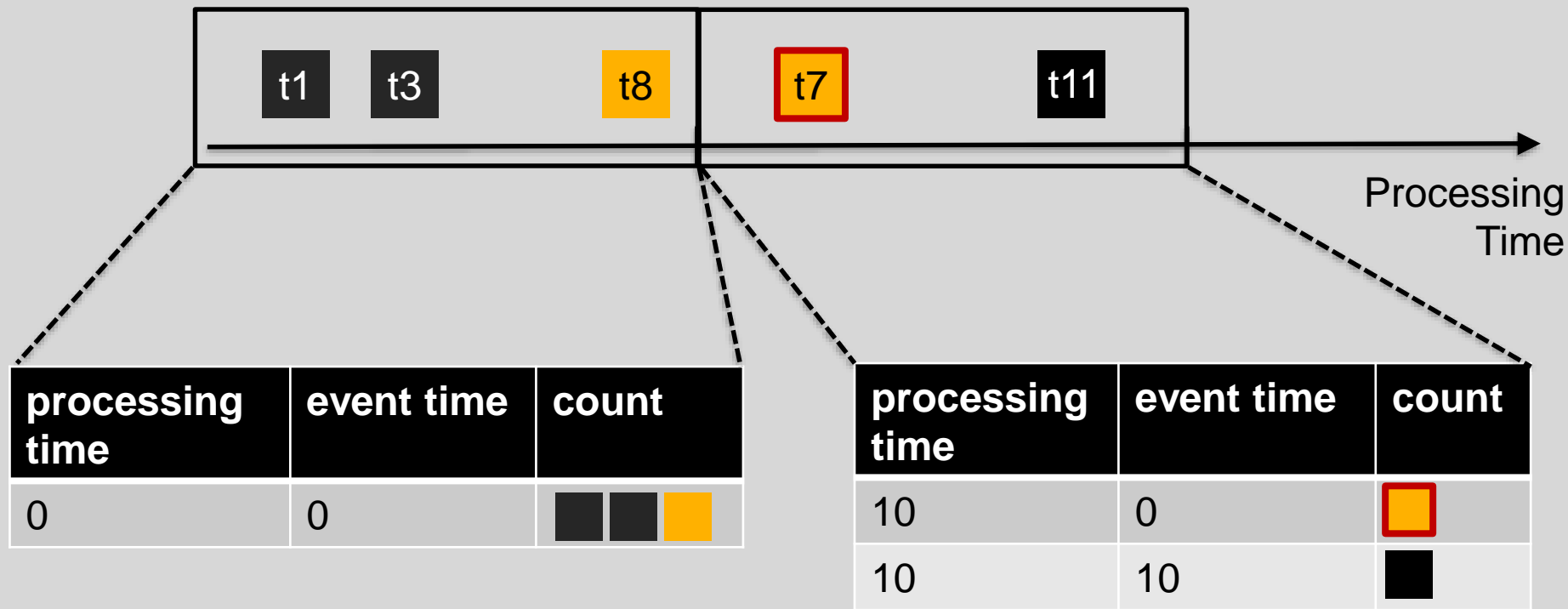
```
    STEP(rowtime BY INTERVAL '10' SECOND) AS processing_time,  
    STEP(event_time BY INTERVAL '10' SECOND) AS event_time,  
    color,  
    COUNT(1)
```

```
FROM ...
```

```
GROUP BY processing_time, event_time, color;
```

# Maintaining Order of Events

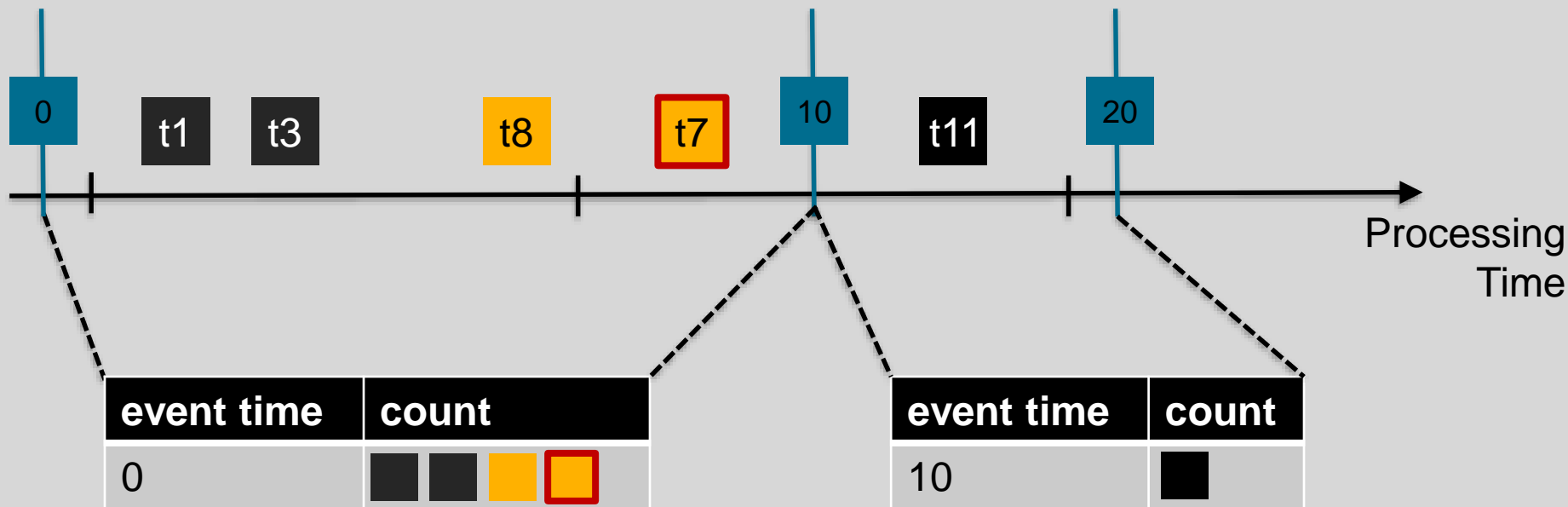
## Using multiple time-windows





# Maintaining Order of Events

## Using event time and watermarks



# Adding Watermarks to a Stream



- Periodic watermarks
  - Assuming ascending timestamps
- Punctuated watermarks

```
stream.assignTimestampsAndWatermarks(  
    new AscendingTimestampExtractor<MyEvent>() {  
  
        @Override  
        public long extractAscendingTimestamp(MyEvent element) {  
            return element.getCreationTime();  
        }  
    }  
));
```

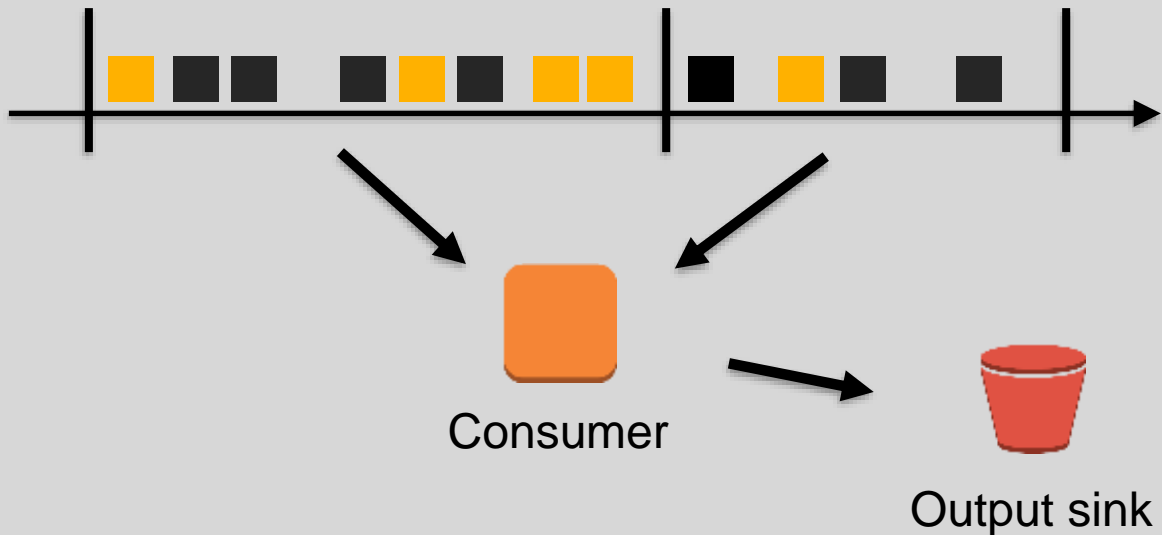


# Different Processing Semantics

Photo by Dominic Alves

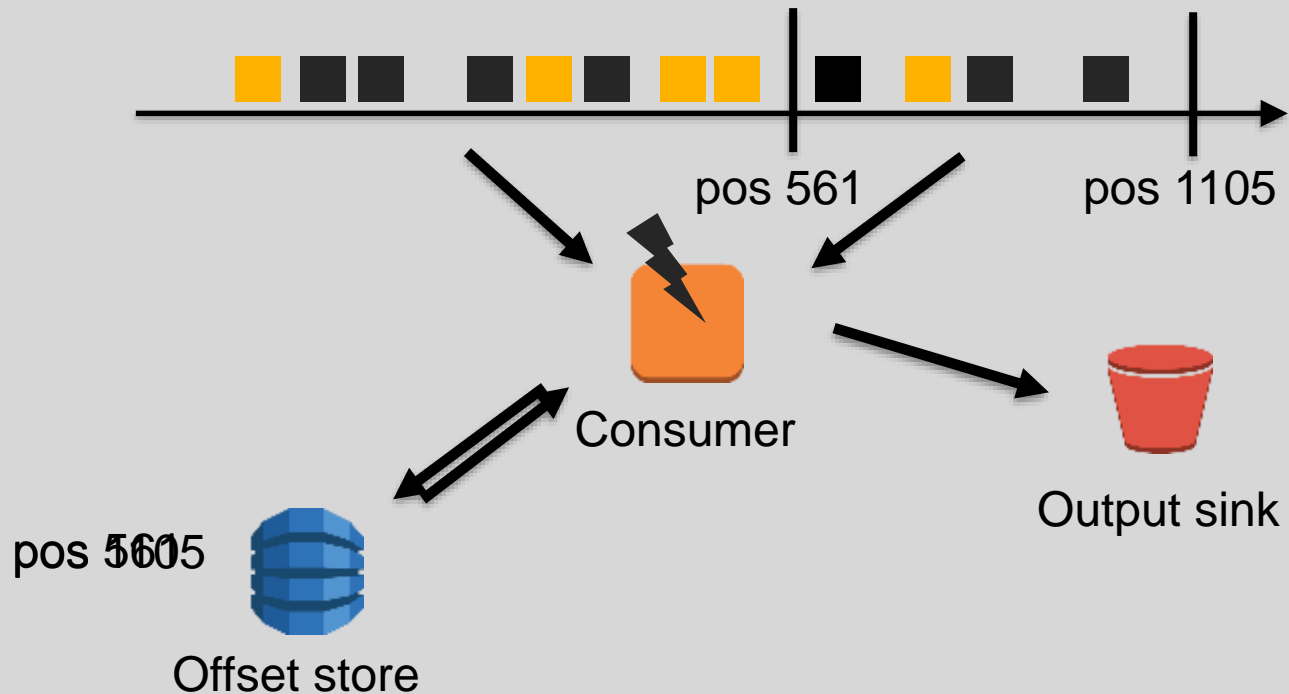
<https://www.flickr.com/photos/dominicpics/6854063597/>

# Consuming Data from a Stream



# Different Processing Semantics

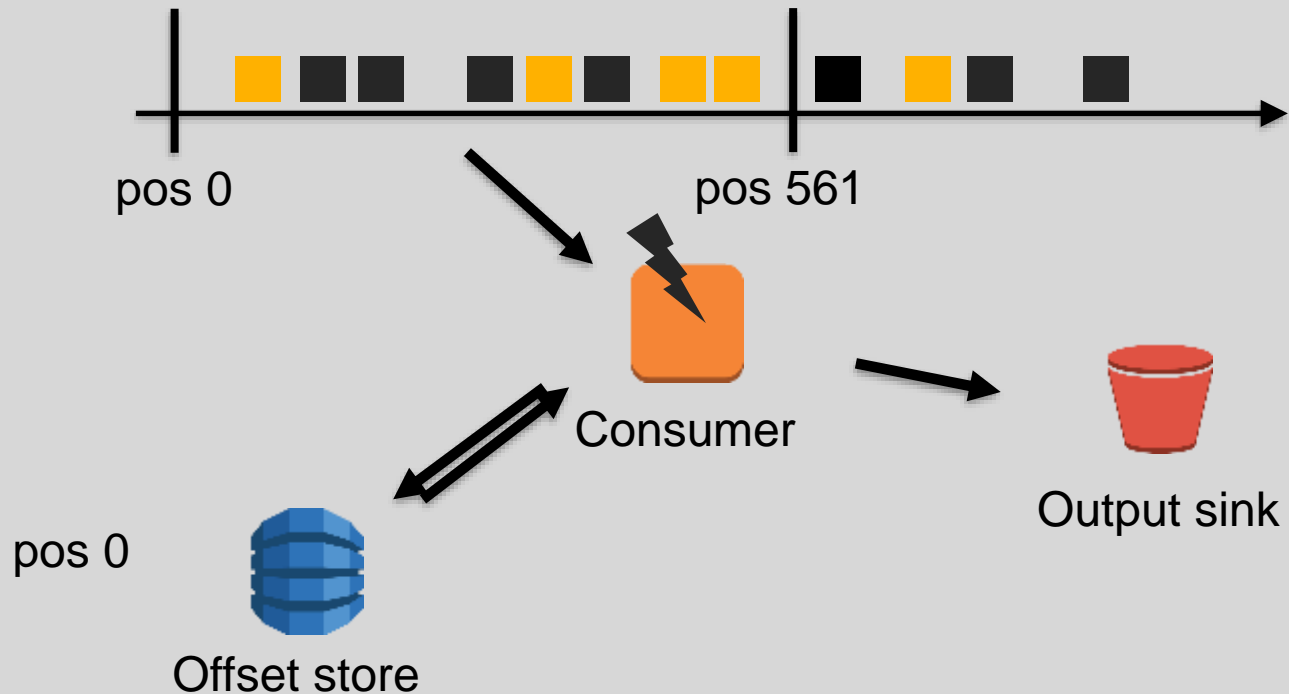
## At-most Once Semantics





# Different Processing Semantics

## At-least Once Semantics



# Different Processing Semantics

## Exactly-once Semantics

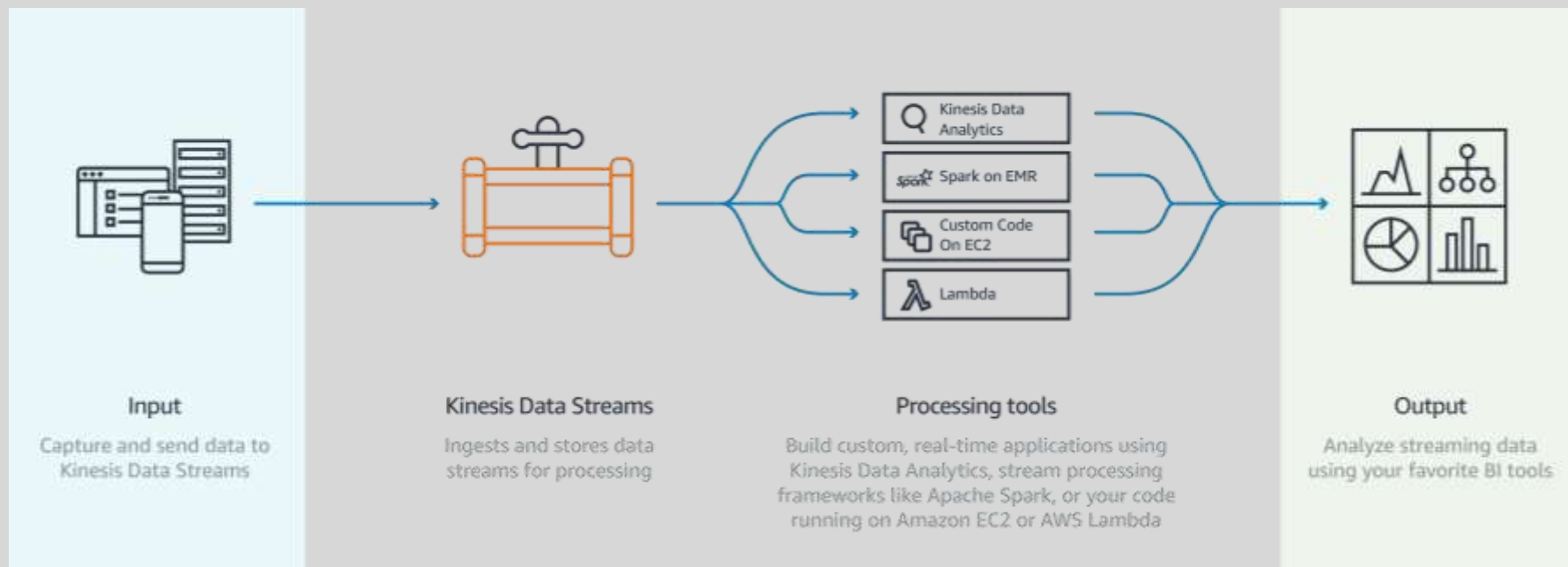
### Message Deduplication

- At-least-once event delivery plus message deduplication
- Keep a transaction log of processed messages
- On failure, replay events and remove duplicated events for every operator

### Distributed Snapshots

- State for each operator is periodically checkpointed
- On failure, rewind operator to the previous consistent state

# Go Build!

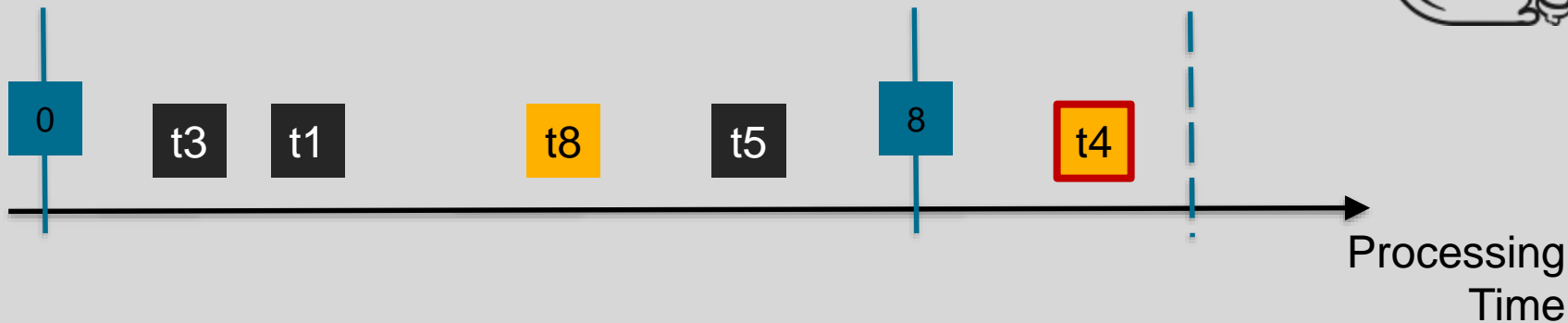


**Please complete the session  
survey in the summit mobile app.**

# Thank you!



# Watermarks and Allowed Lateness



stream

```
.keyBy(<key selector>)  
.window(<window assigner>)  
.allowedLateness(<time>)  
.sideOutputLateData(LateOutputTag)
```