aws SUMMIT

Deep Dive into Concepts and Tools for Analyzing Streaming Data

Dr. Steffen Hausmann

Sr. Solutions Architect, Amazon Web Services

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Data originates in real-time

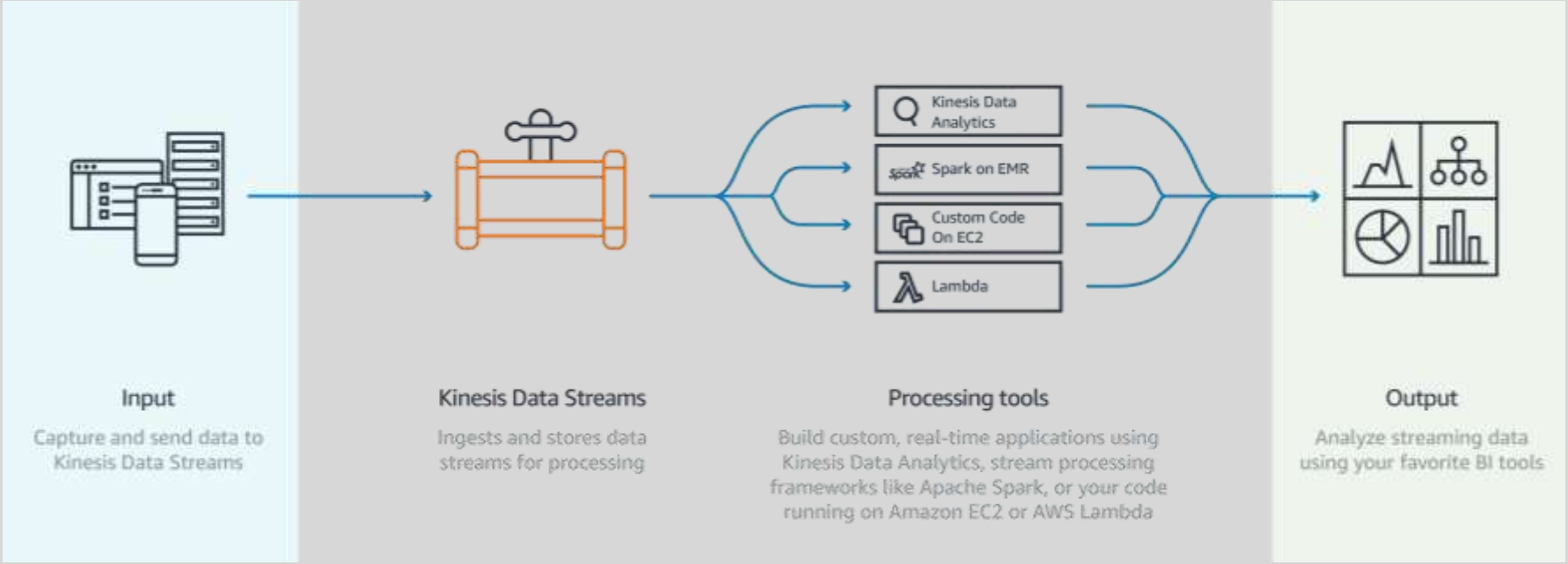
Photo by mountainamoeba
https://www.flickr.com/photos/mountainamoeba/2527300028/

Analytics is done in batches

Photo by PracticalHacks
https://www.flickr.com/photos/29225844@N05/2828724211

Insights are Perishable

Analyzing Streaming Data on AWS



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

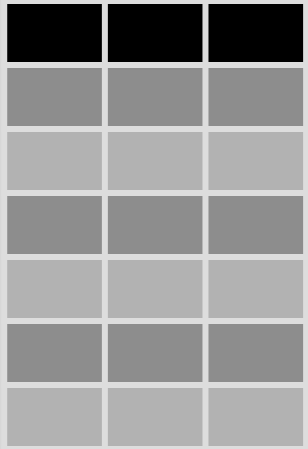
Challenges of Stream Processing

Photo by Follow Your Nose
https://www.flickr.com/photos/laprimadc/3294167673

Comparing Streams and Relations

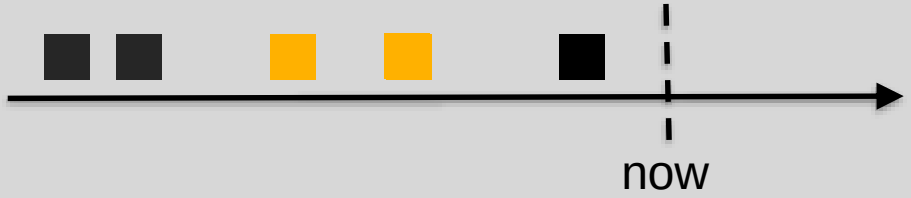
Relation

$$R \subseteq Id \times Color$$



Stream

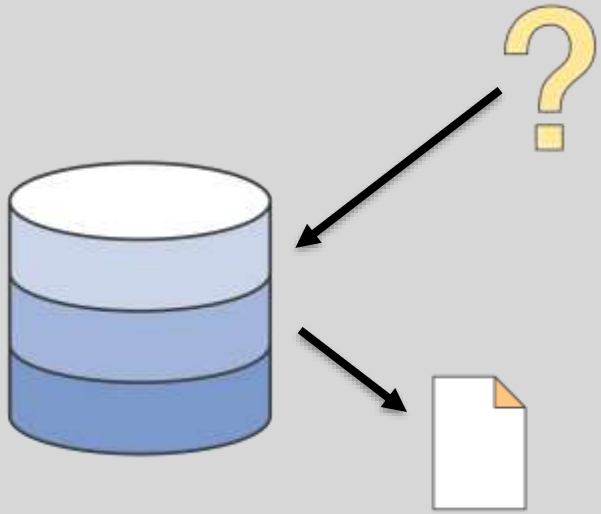
$$S \subseteq Id \times Color \times Time$$



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

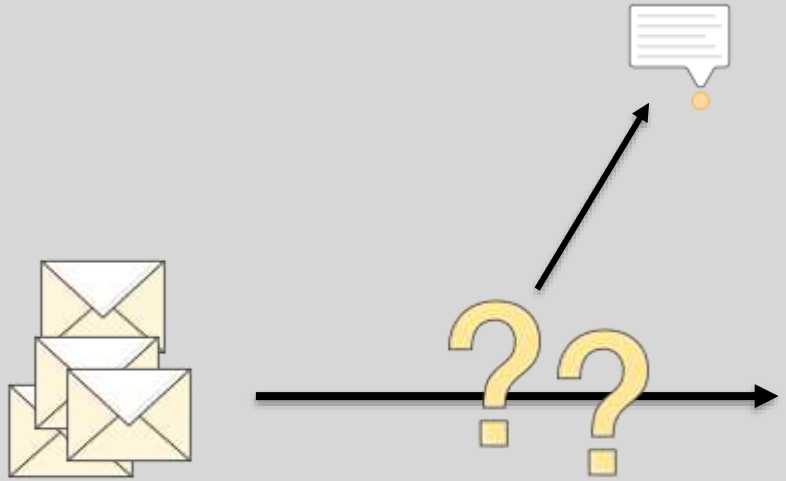
Querying Streams and Relations

Relation



Fixed data and ad-hoc queries

Stream



Fixed queries and
continuously ingested data

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

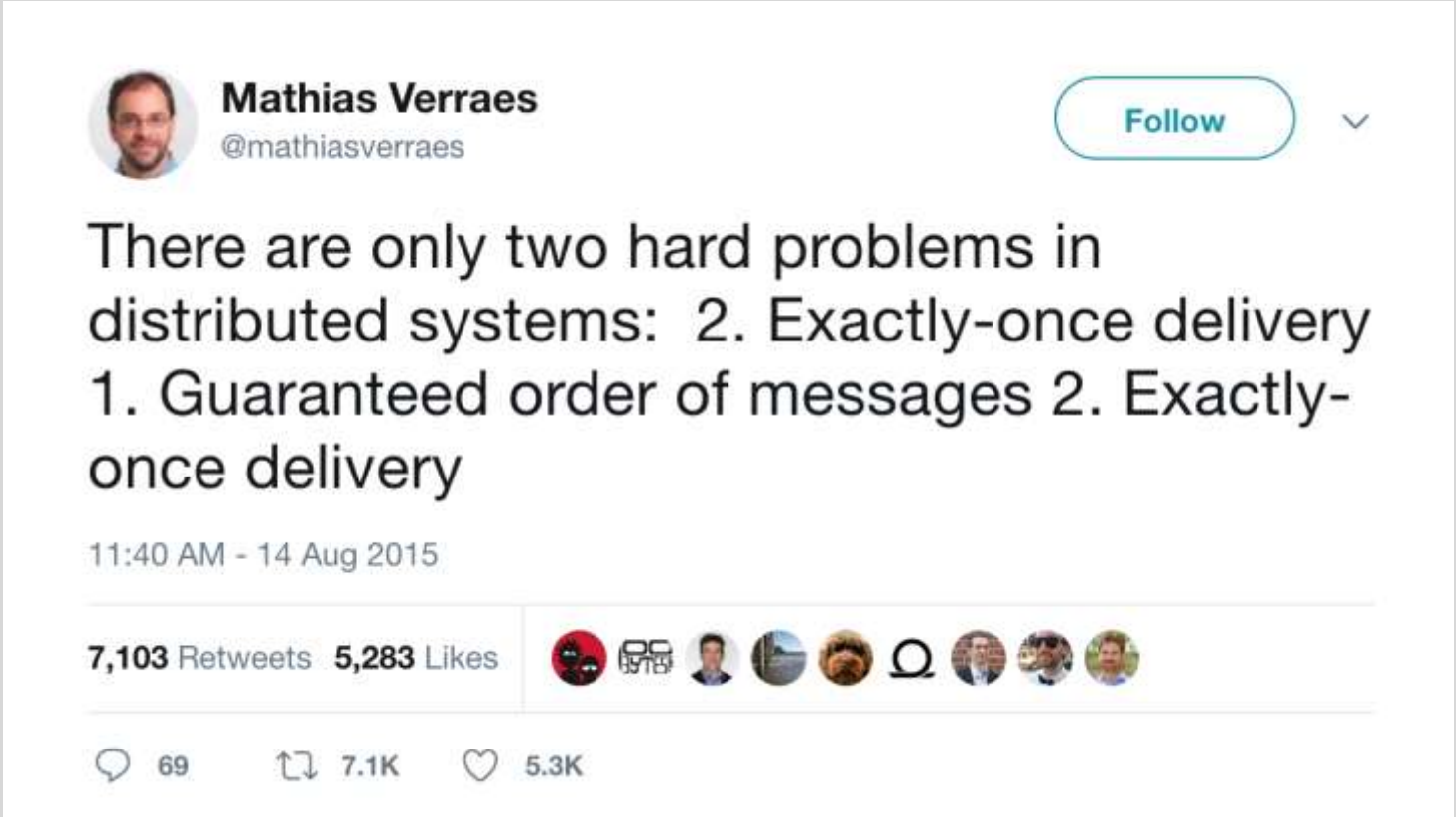
Challenges of Querying Infinite Streams

```
SELECT * FROM S WHERE color = ‘black’

SELECT * FROM S JOIN S’

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

... NOT EXISTS (SELECT * FROM S WHERE color = ‘red’)
```

Unbounded Query

Non-monotonic Operators

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

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

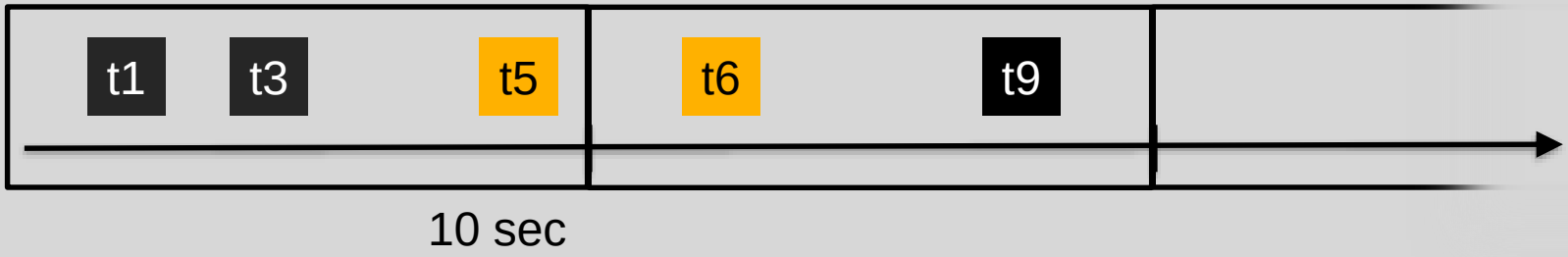
© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Evaluating Queries over Streams

Photo by Brad Greenlee
https://www.flickr.com/photos/bgreenlee/91309374/

Evaluating Non-monotonic Operators

Tumbling Windows

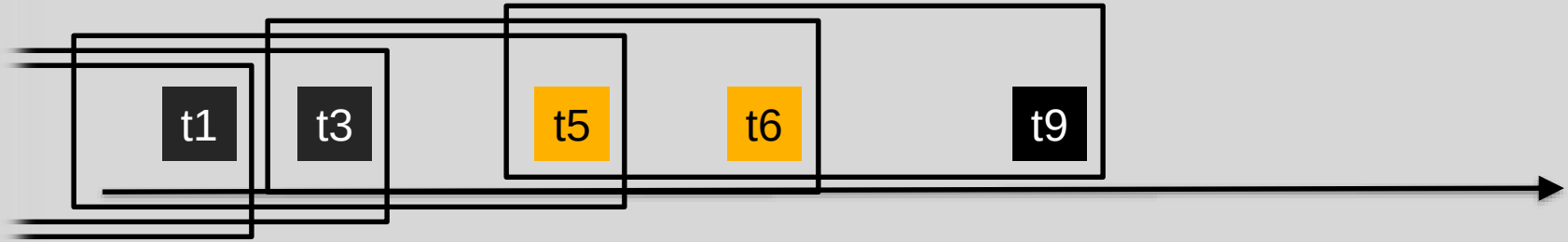



```
SELECT STREAM color, COUNT(1)
FROM ...
GROUP BY STEP(rowtime BY INTERVAL ‘10’ SECOND), color;
```

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Evaluating Non-monotonic Operators

Sliding Windows

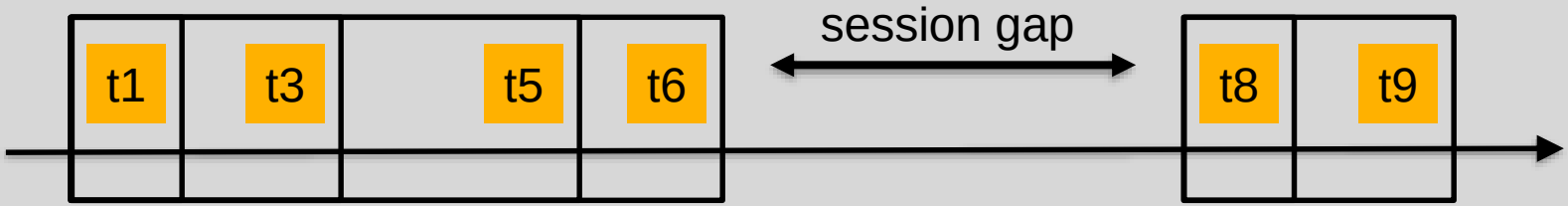



```
SELECT STREAM color, COUNT(1) OVER w
FROM ...
GROUP BY color
WINDOW w AS (RANGE INTERVAL ’10’ SECOND PRECEDING);
```

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Evaluating Non-monotonic Operators

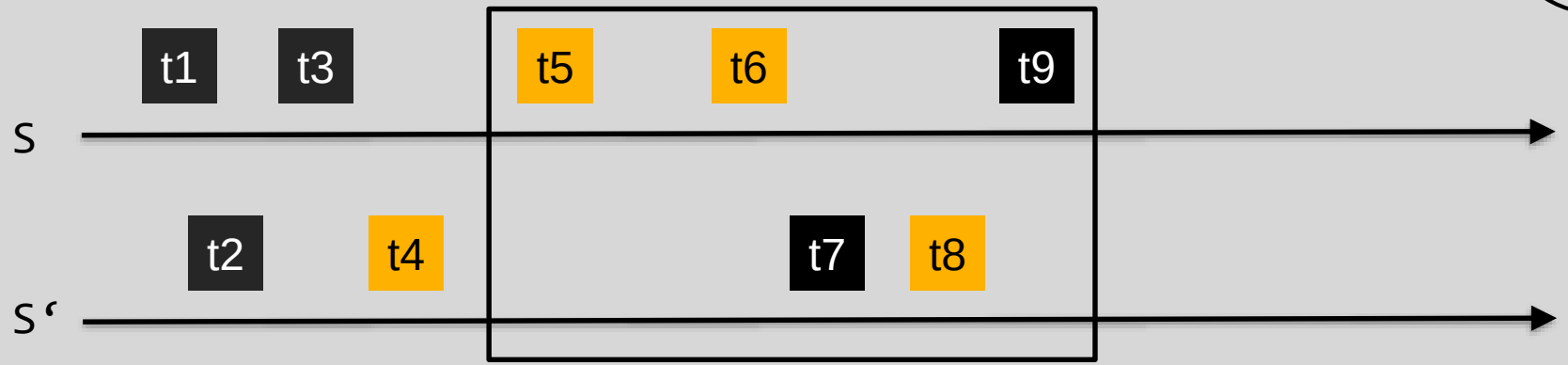
Session Windows




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

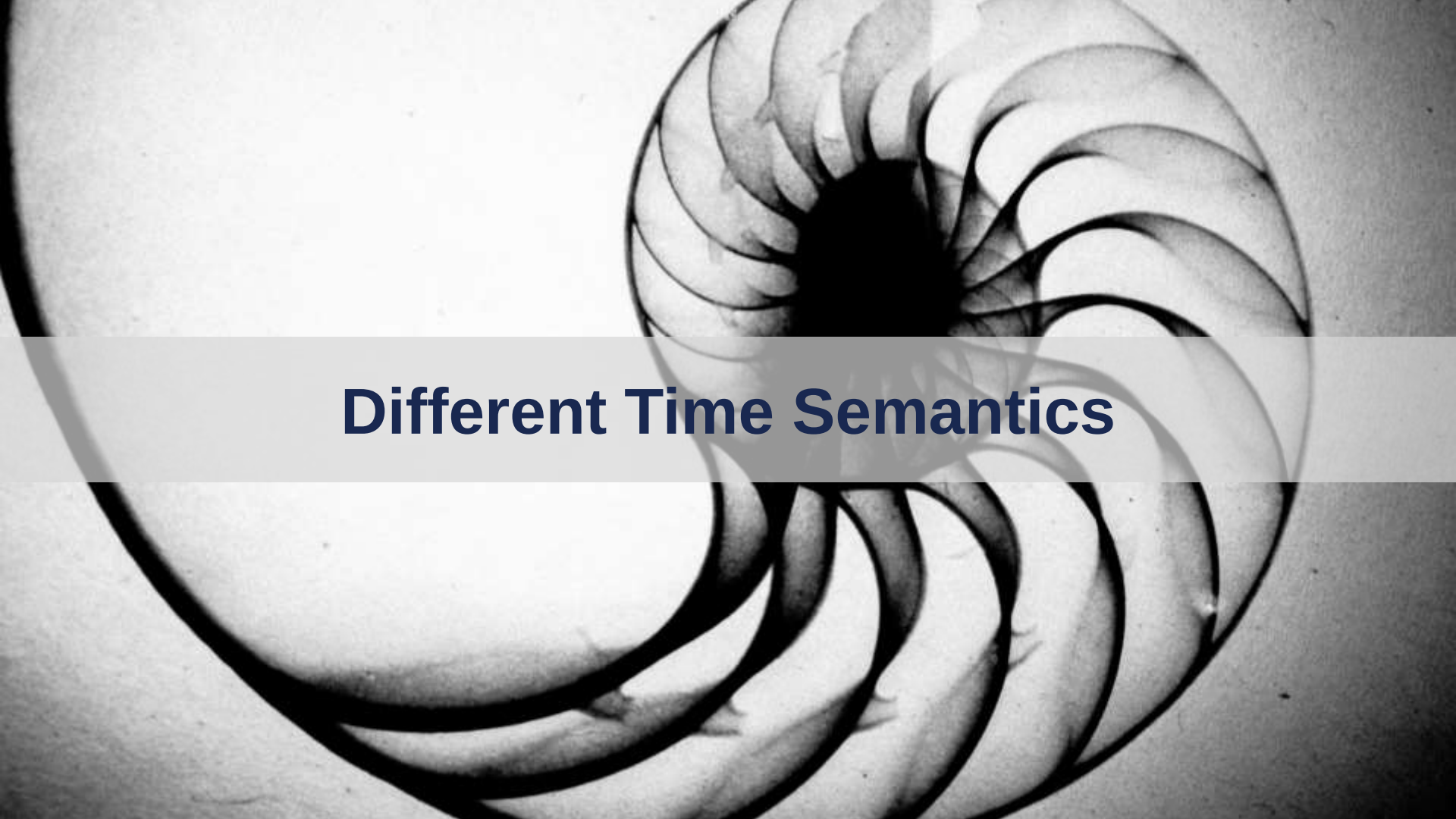
© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Evaluating Unbounded Queries



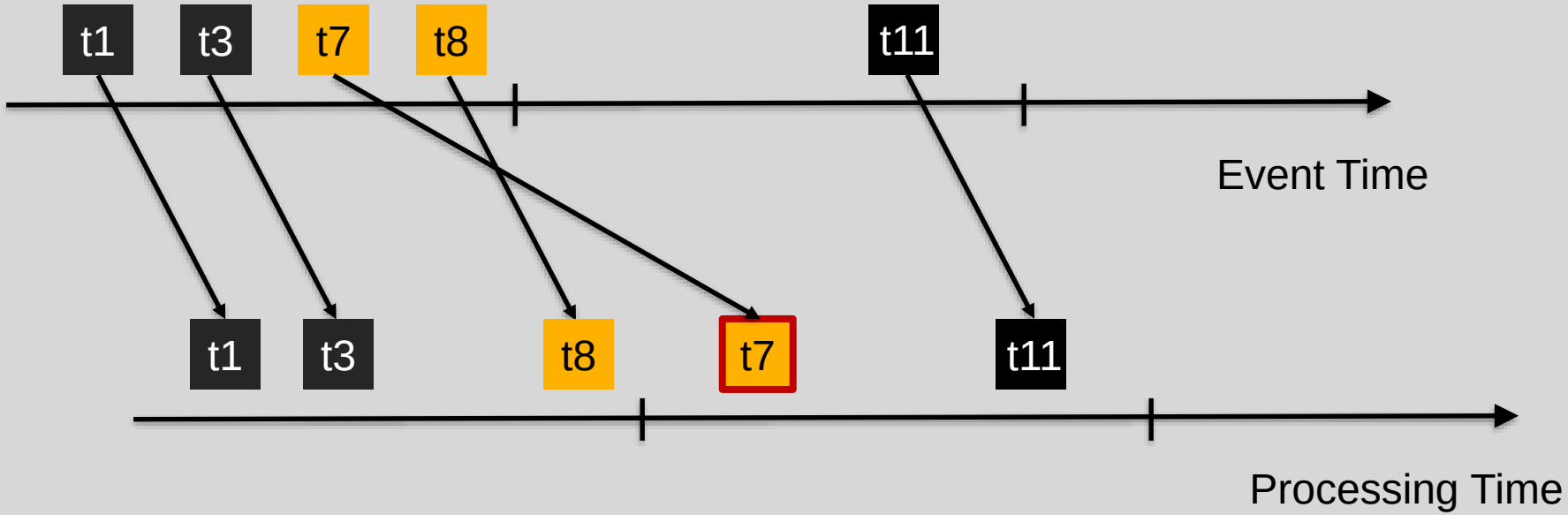

```
SELECT STREAM *
FROM S OVER w AS s JOIN S’ OVER w AS t
ON s.color = t.color
WINDOW w AS (RANGE INTERVAL ‘10’ SECOND PRECEDING);
```

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Different Time Semantics

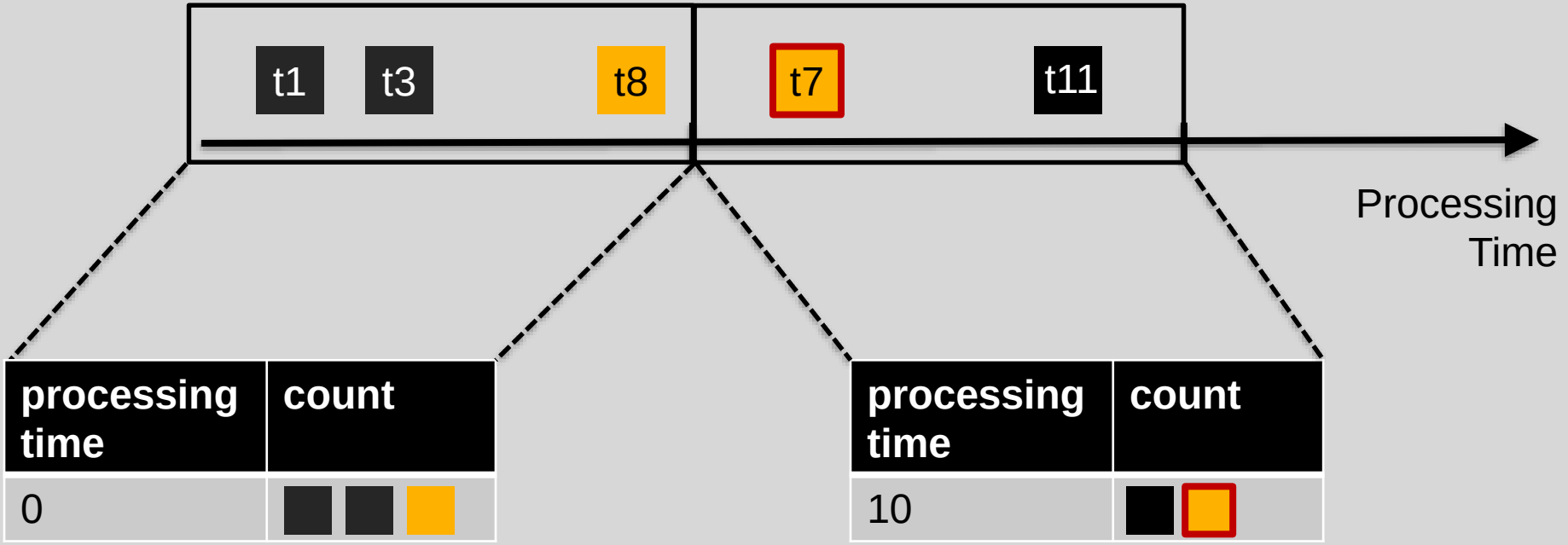
Maintaining Order of Events



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Maintaining Order of Events

Using processing time based windows



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

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;
```

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Maintaining Order of Events

Using multiple time-windows

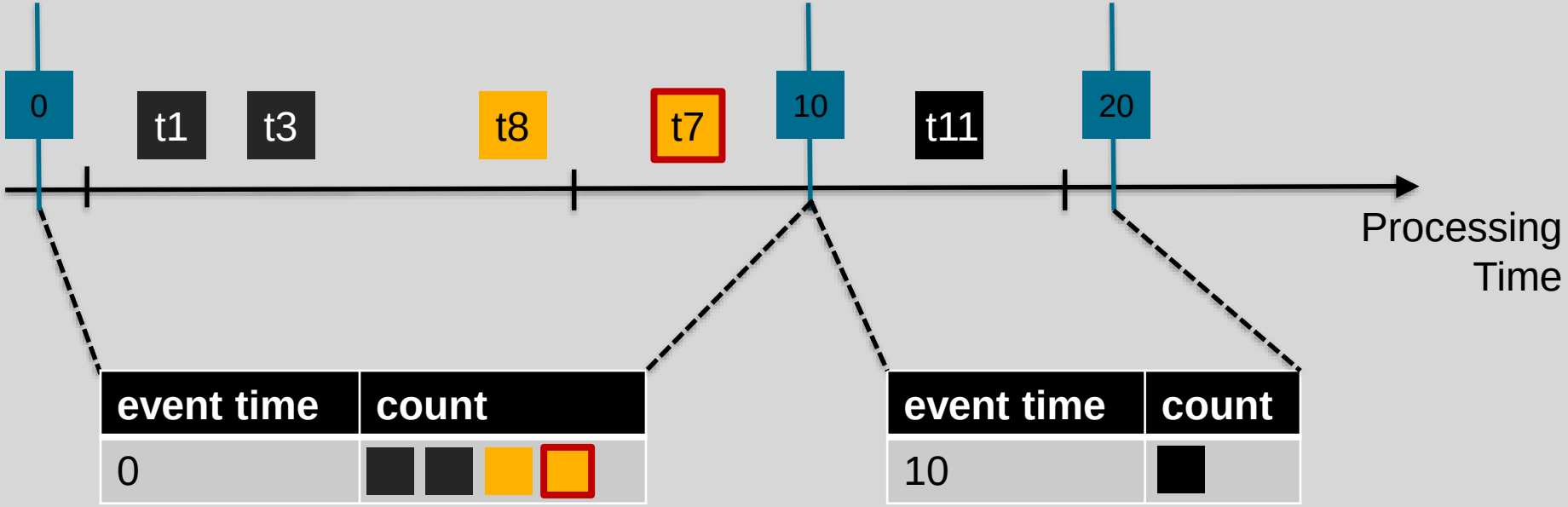
processing time	event time	count
0	0	

processing time	event time	count
10	0	
10	10	

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Maintaining Order of Events

Using event time and watermarks



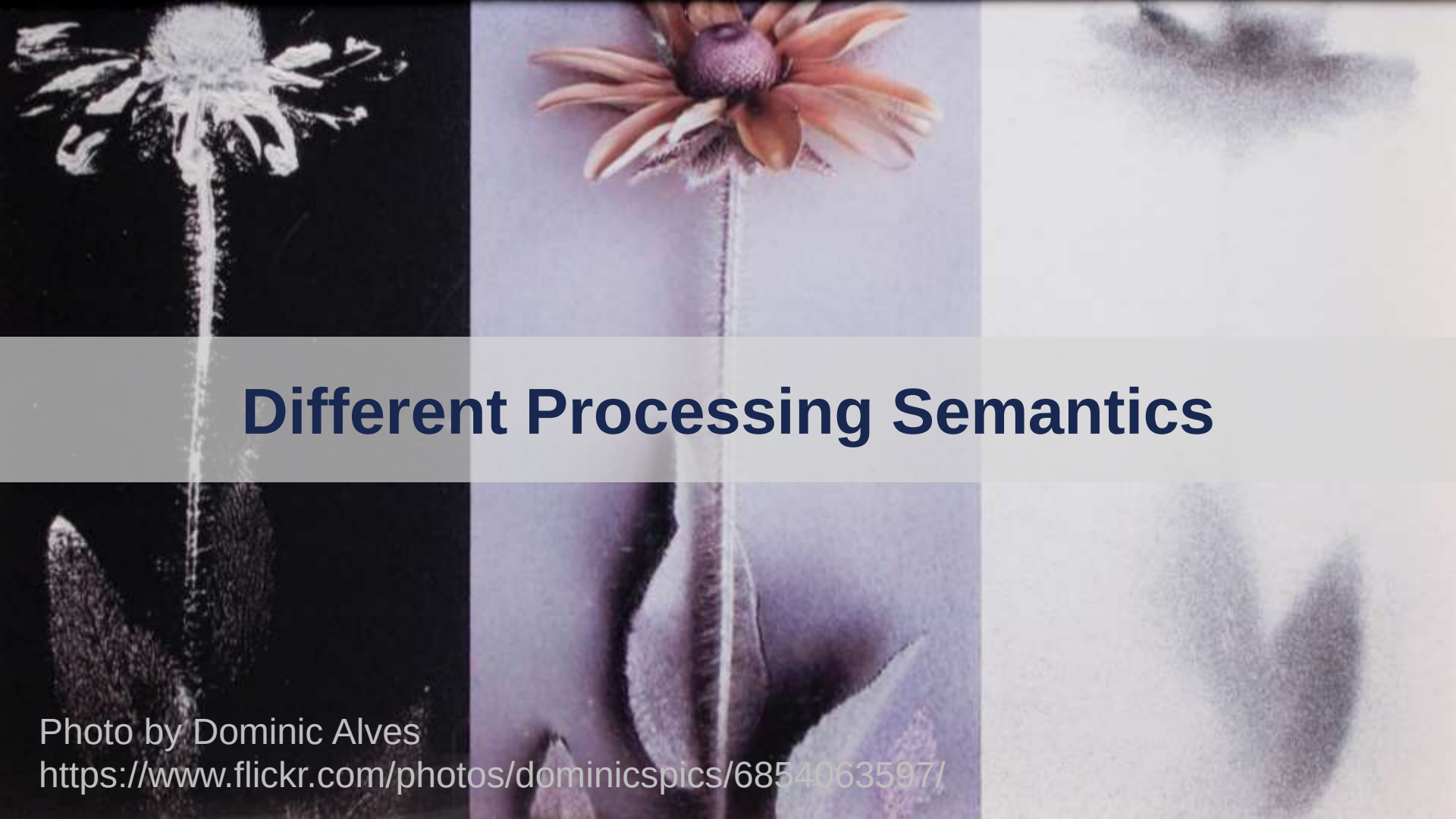
© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

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();
        }
    });
```

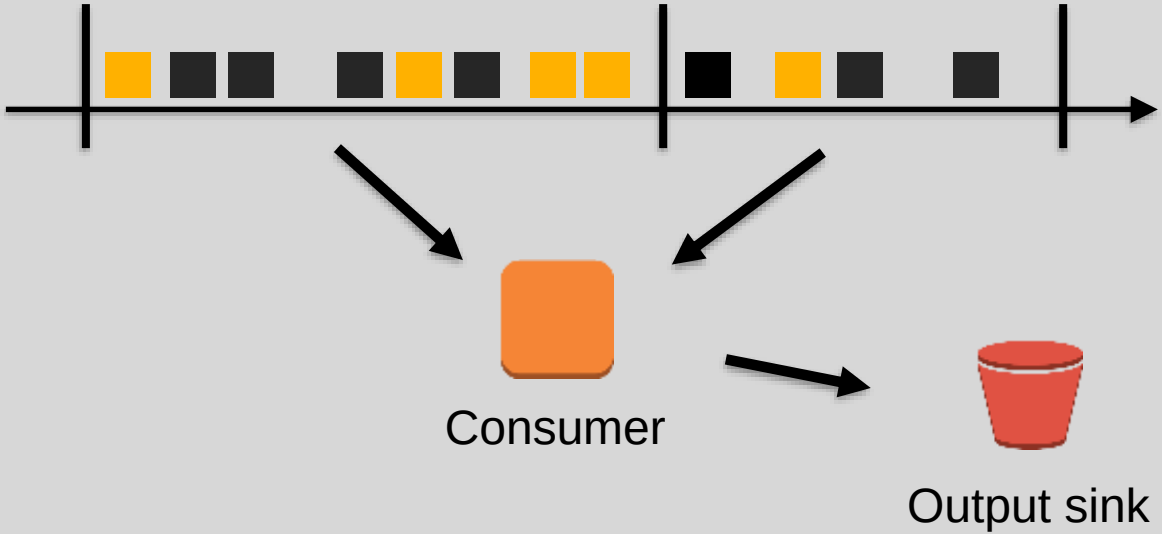
© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Different Processing Semantics

Photo by Dominic Alves
https://www.flickr.com/photos/dominicspics/68540635971

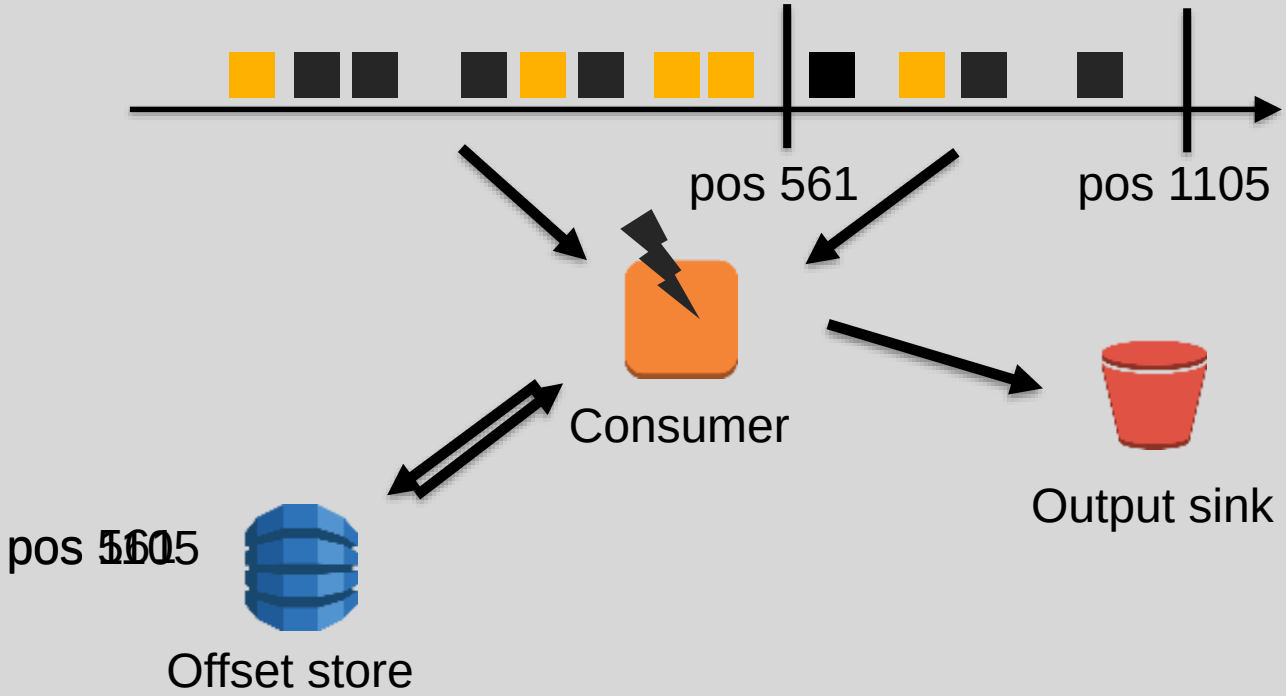
Consuming Data from a Stream



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Different Processing Semantics

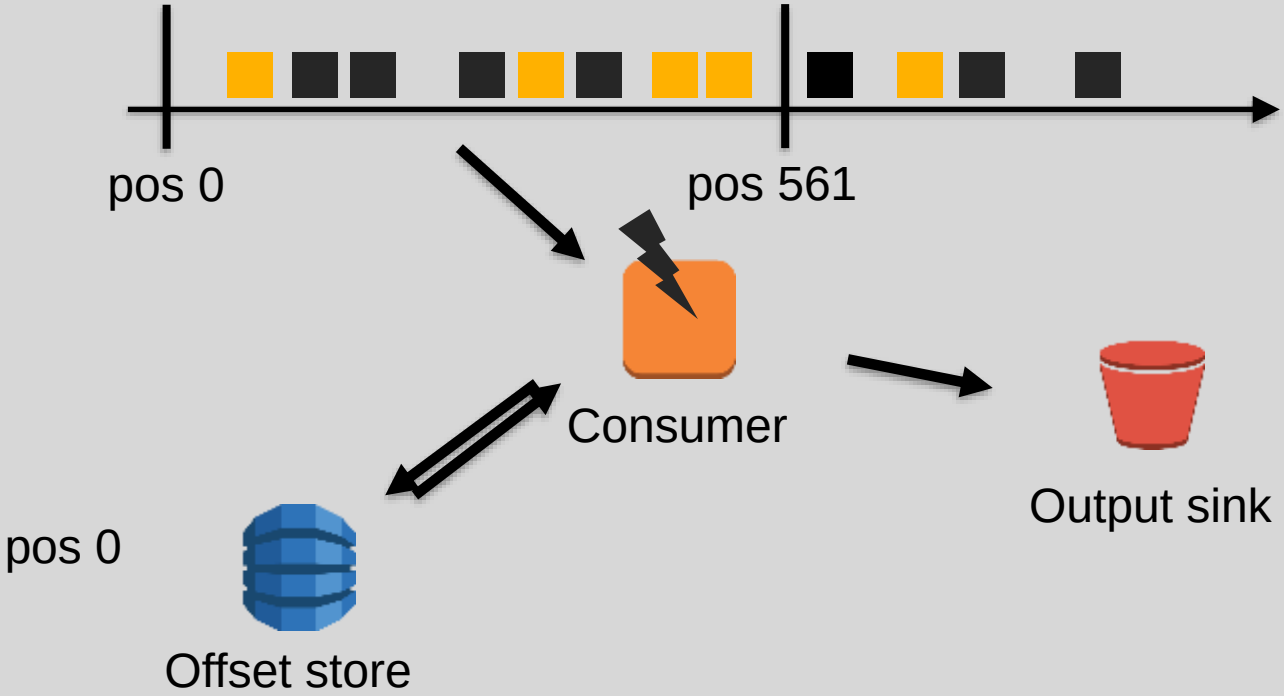
At-most Once Semantics



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Different Processing Semantics

At-least Once Semantics



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Different Processing Semantics

Exactly-once Semantics

Message Deduplication

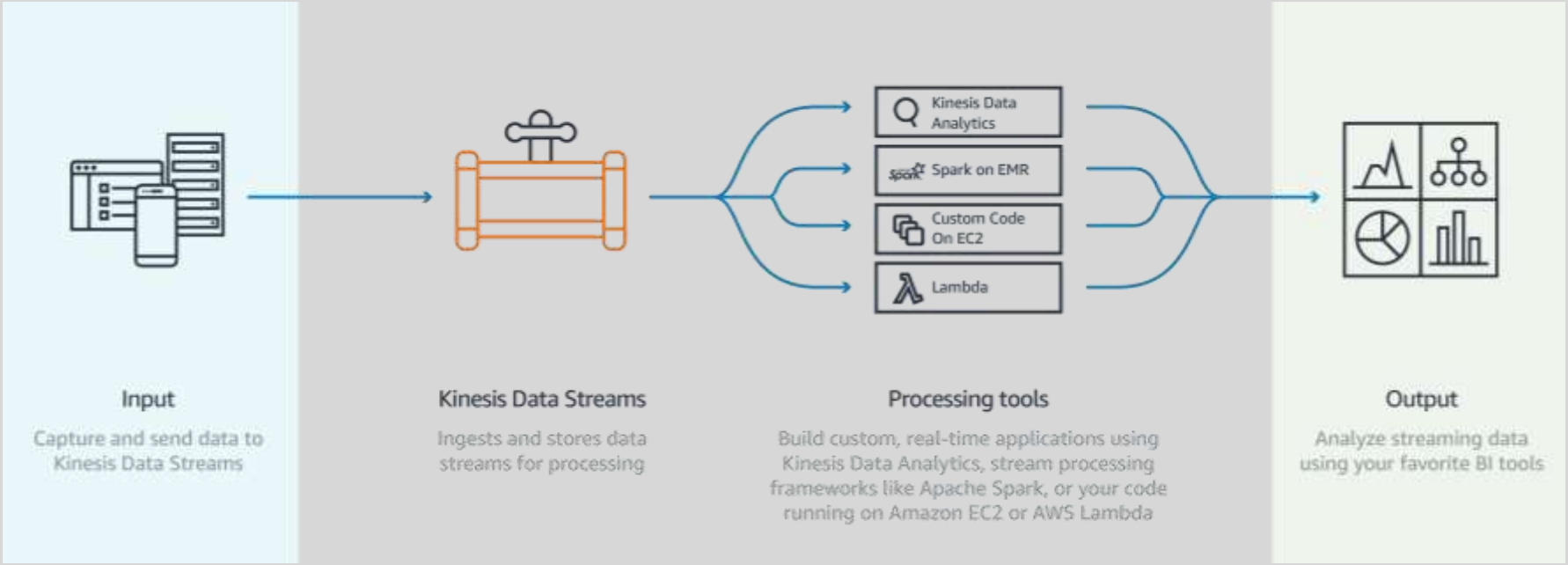
- At-least-once event delivery plus message deduplication
- Keep a transaction log of processed messages
- On failure, replay events and remove duplicated events for every operator

Distributed Snapshots

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

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Go Build!



© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Please complete the session survey in the summit mobile app.

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws SUMMIT

Thank you!

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Watermarks and Allowed Lateness


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

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.