Spark Streaming

(Scala Meetup – February, 2023)

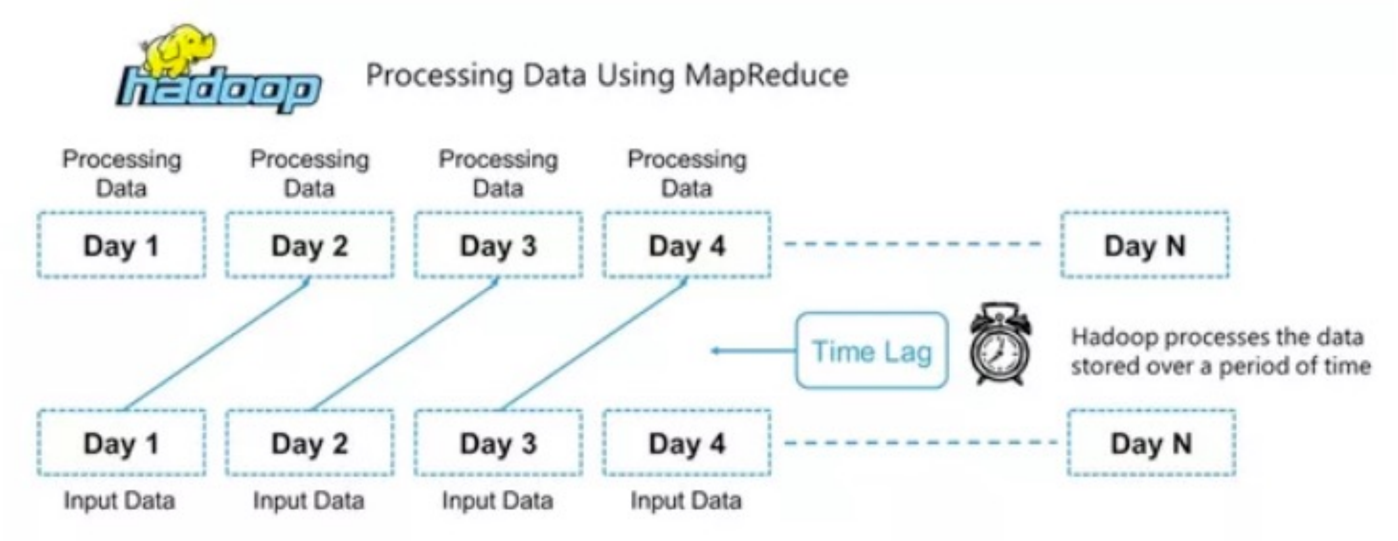
Attila Szűcs

attila_szucs@epam.com
big data, architecture

BATCH PROCESSING

Processing the data collected over a time period

- Traditional Big Data is batch processing
- Results are not real-time; we need to wait for the next “batch”

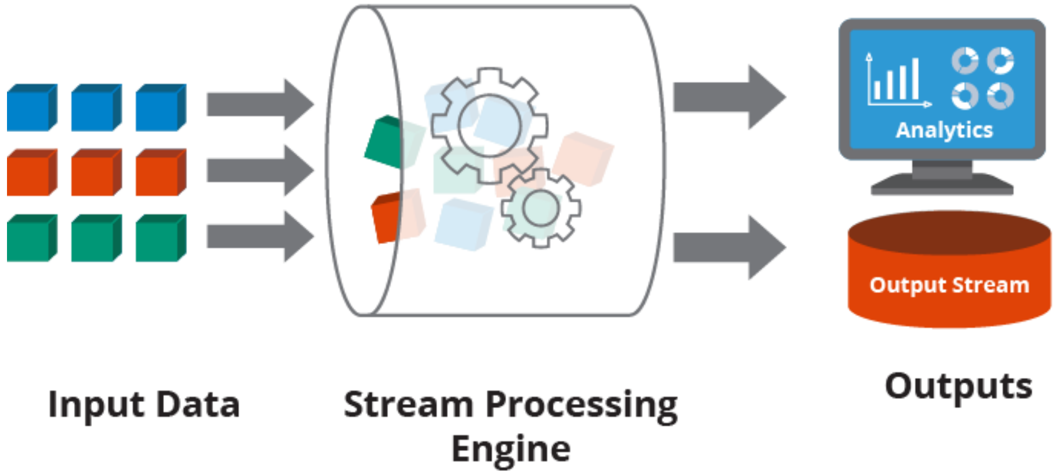


STREAM PROCESSING

STREAM PROCESSING

Definition:

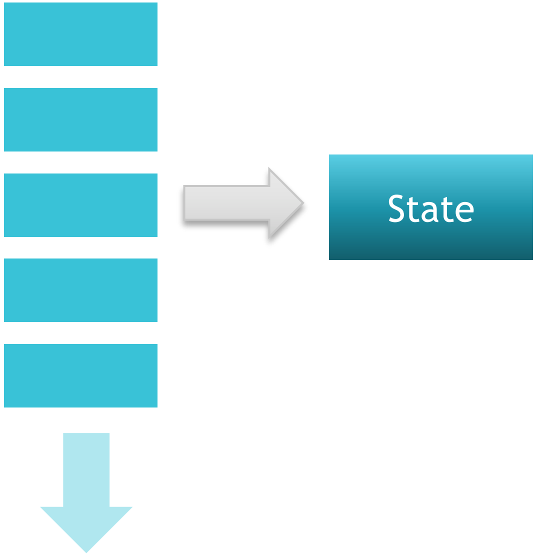
Stream processing is the practice of taking action on a series of data at the time the data is created.



An overview on Hazelcast

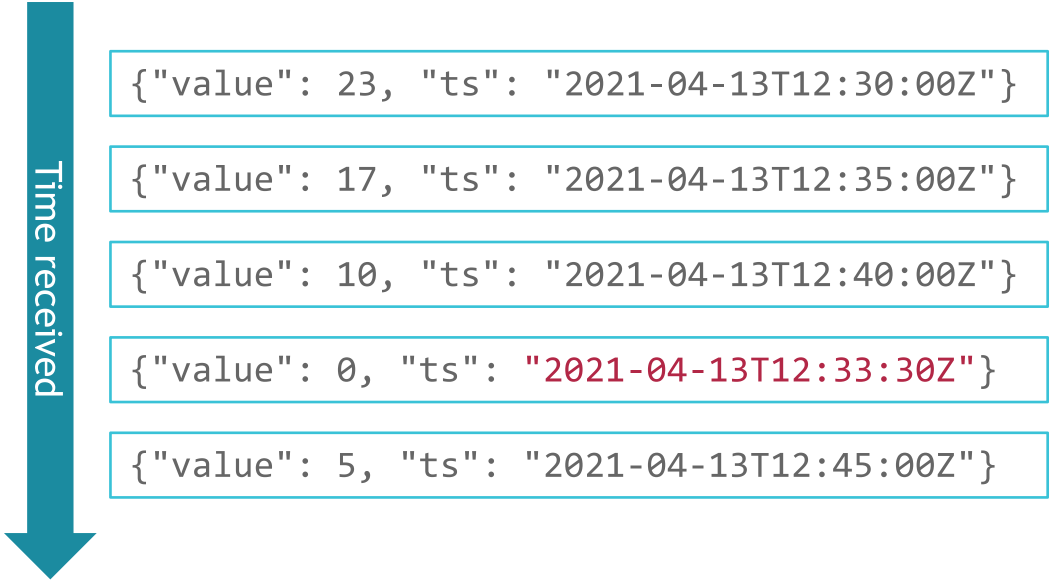
STREAMING CHALLENGES

State management



Traditionally it was considered in-accurate and unreliable

Event time vs processing time; late events

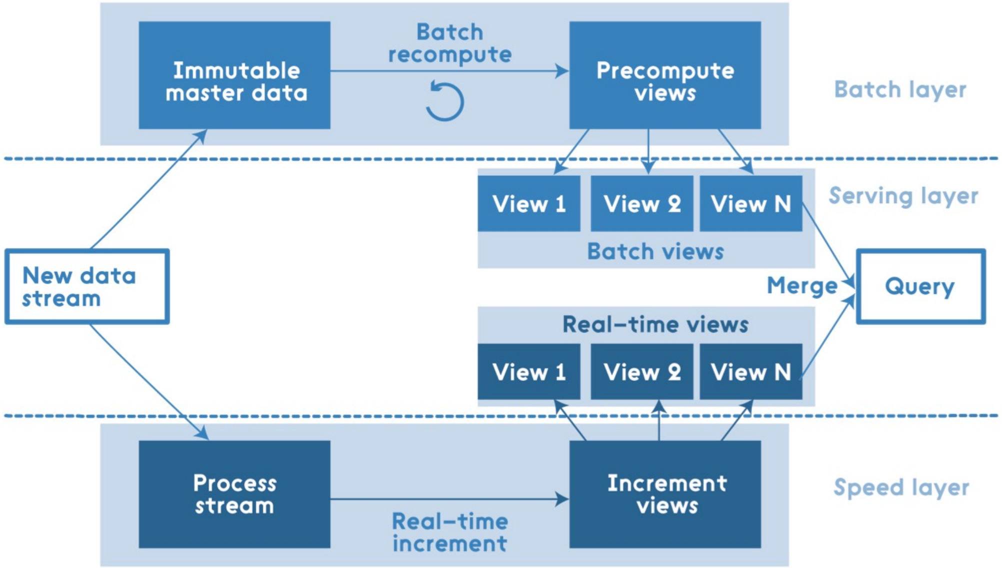


```
{"value": 23, "ts": "2021-04-13T12:30:00Z"}
{"value": 17, "ts": "2021-04-13T12:35:00Z"}
{"value": 10, "ts": "2021-04-13T12:40:00Z"}
{"value": 0, "ts": "2021-04-13T12:33:30Z"}
{"value": 5, "ts": "2021-04-13T12:45:00Z"}
```

LAMBDA ARCHITECTURE

Attempts to balance latency, throughput, and fault-tolerance by

- using **batch processing** to provide comprehensive and accurate views of batch data,
- while simultaneously using real-time **stream processing** to provide views of online data.



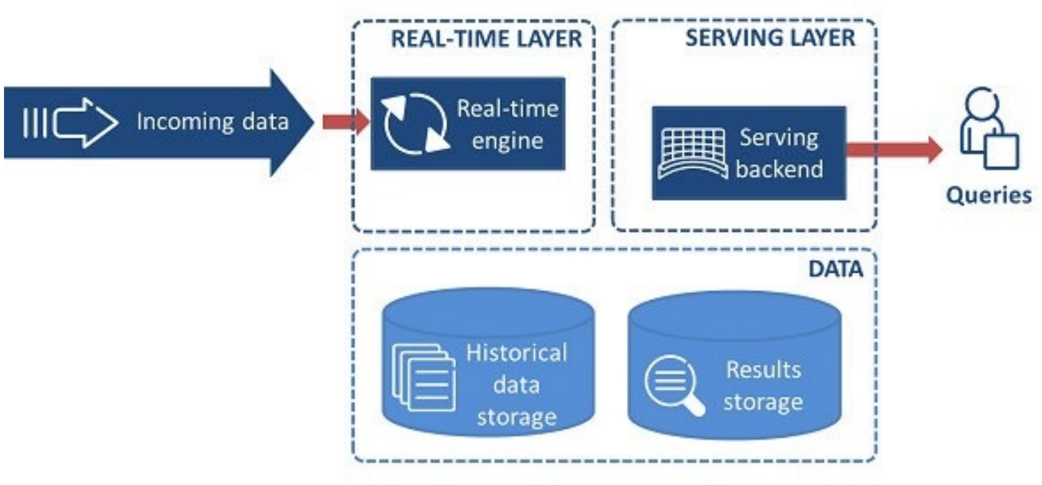
Disadvantages

- Two codebases, need to be in sync
- Complex architecture, specialized tools

KAPPA ARCHITECTURE

Is a software architecture used for processing streaming data. The main premise is that you can

- perform both real-time and batch processing with a single technology stack.



- **Everything is a stream**

  Batch operations become a subset of streaming

- **Keep it short and simple** (KISS) principle.

  A single analytics engine is required

  Coding and maintenance are simpler

- **Immutable data sources**

  Data source is persisted, and views are derived

  State can always be recomputed from source

- **Replay functionality**

  Computations and results can evolve by replaying the historical data from a stream

AGENDA

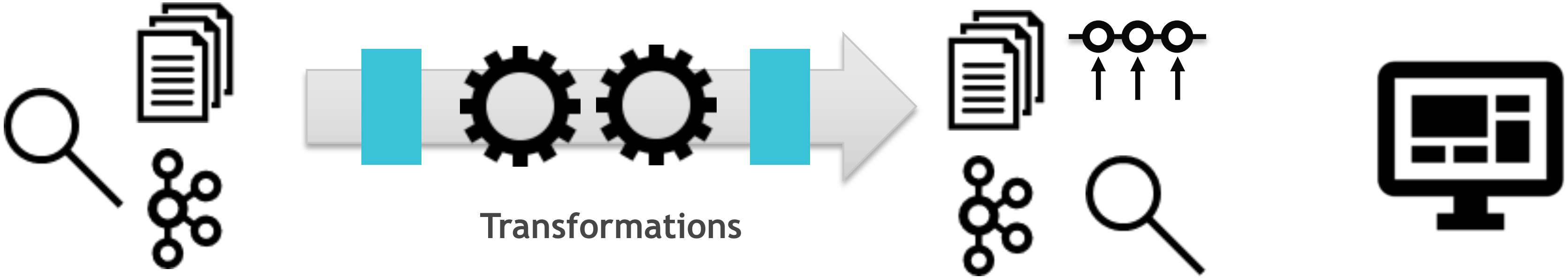
Handling events

- **Apache Kafka** is a distributed data store optimized for ingesting and processing streaming data in real-time.
- **Simplified definition:**
  Persistent & scalable messaging platform

Processing events

- **Spark streaming** is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
- **Spark streaming workshop**

STREAM PROCESSING – CONCEPTS



Source

- Files
- Kafka
- Socket (testing)
- Rate (testing)
- ...

Triggers

- Default (microbatch)
- Fixed interval
- Once
- Continuous (experimental)

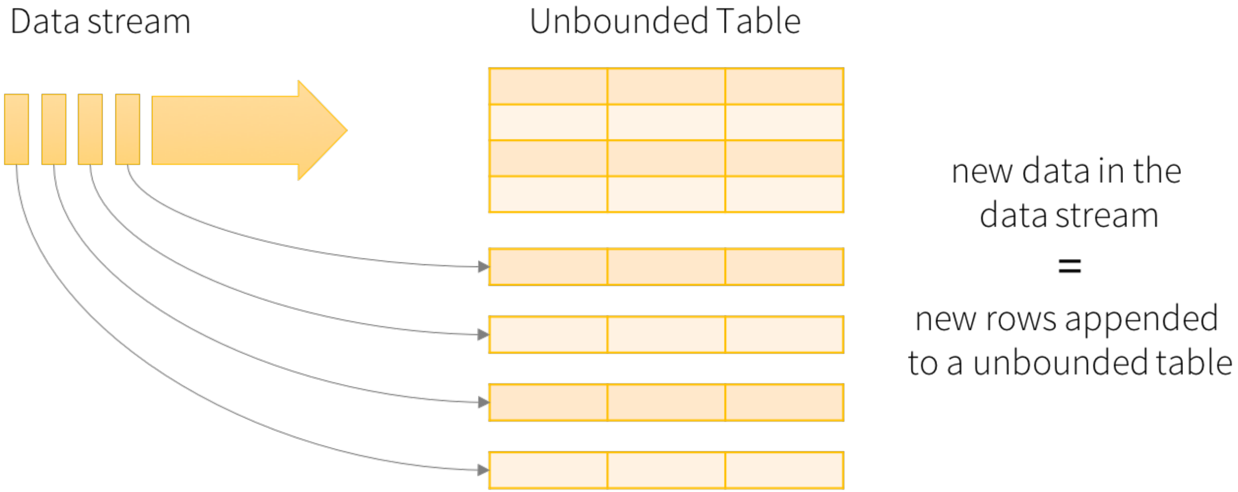
Sink

- Files
- Kafka
- Foreach
- Console (testing)
- ...

Output modes

- Append (default)
- Update
- Complete

SPARK – STRUCTURED STREAMING


Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table

Data stream as an unbounded table

- Spark's DataFrame API can be used
- Streams are (unbounded) data frames
- Operations on unbounded data frames result in unbounded data frames
- The same API can be used for both batch and stream processing

STRUCTURED STREAMING

Dataframe API

Transformations

- Simple transformations
- Aggregations
- Joins
  - Stream-Static
  - Stream-Stream

Limitations apply

```
val input = spark.readStream
    .schema(schema)
    .json("/path/to/folder")

val countStream = input
    .where("size > 10")
    .groupBy("category")
    .count()

countStream.writeStream
  .outputMode("complete")
  .format("console")
  .start()
```

DEMO – SPARK STRUCTURED STREAMING

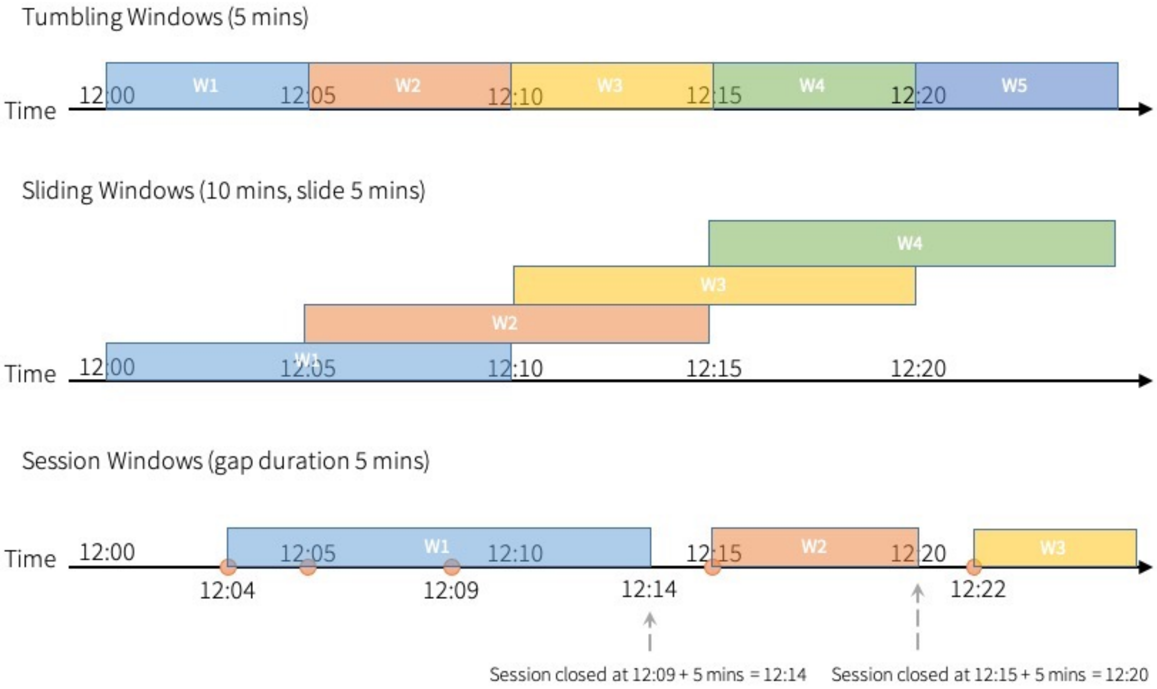
Source code available on

GitHub

medvekoma/streaming-workshop

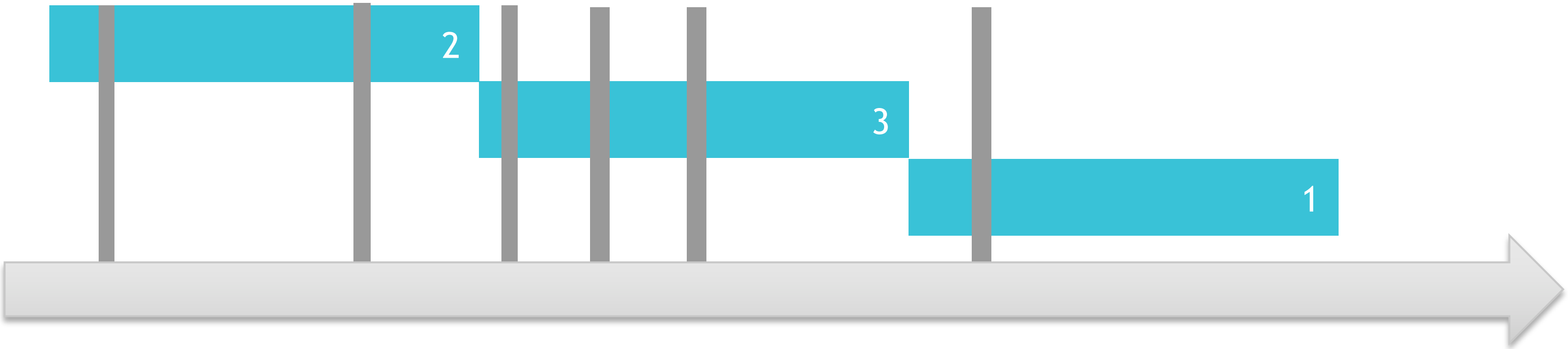
EVENT TIME & TIME WINDOWS

- Event time != processing time
- Event time is usually a field in the data
- Often, we group the data by event time windows
- Window types
  - Tumbling windows
  - Sliding windows
  - Session windows



Tumbling Windows (5 mins)
Time
12:00
W1
12:05
W2
12:10
W3
12:15
W4
12:20
W5
Sliding Windows (10 mins, slide 5 mins)
W4
W3
W2
W1
Time
12:00
12:05
12:10
12:15
12:20
Session Windows (gap duration 5 mins)
Time
12:00
12:05
W1
12:10
12:15
W2
12:20
W3
12:04
12:09
12:14
12:22
Session closed at 12:09 + 5 mins = 12:14
Session closed at 12:15 + 5 mins = 12:20

EVENT TIME WINDOWS – TUMBLING WINDOWS

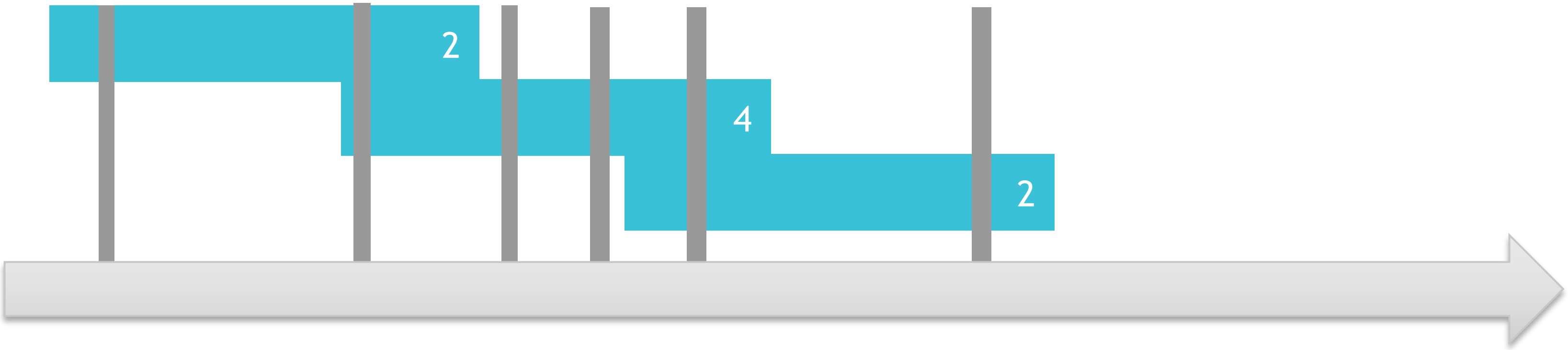


Window types

- Tumbling window

```
streamingDF
  .groupBy(
    window(col("event_time"), "15 minutes"))
  .count()
```

EVENT TIME WINDOWS – SLIDING WINDOWS

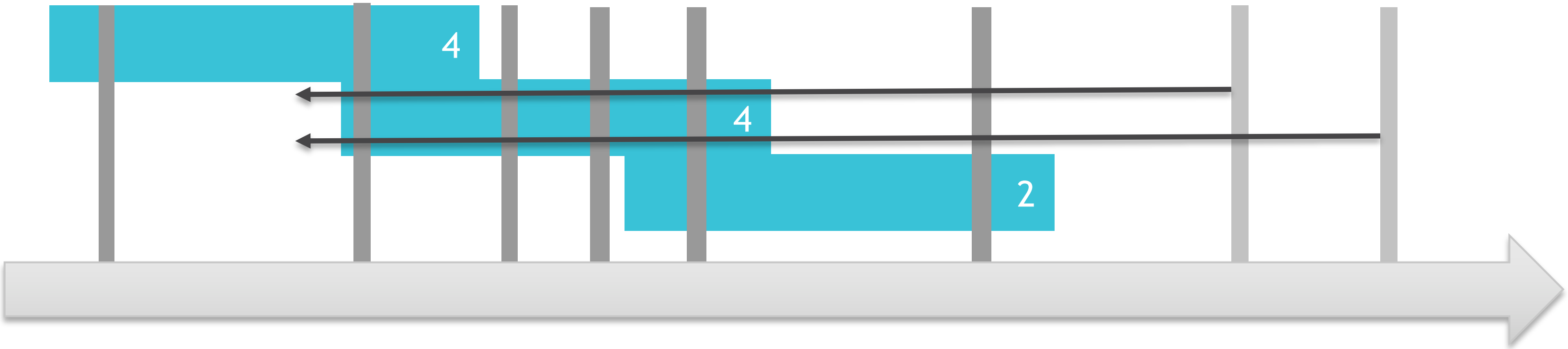


Window types

- Tumbling window
- Sliding window

```
streamingDF
  .groupBy(
    window(col("event_time"), "15 minutes", "10 minutes"))
  .count()
```

EVENT TIME WINDOWS – LATE EVENTS



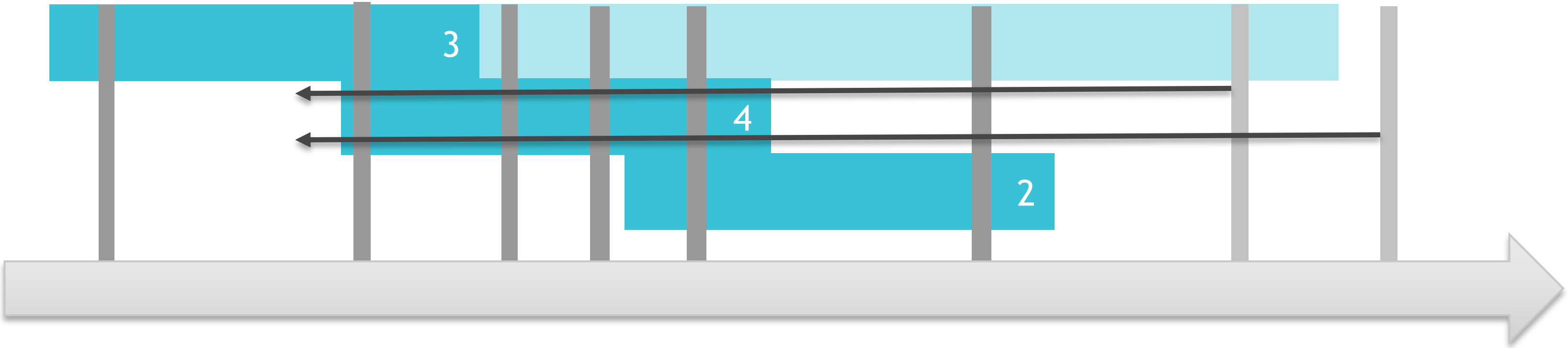
Window types

- Tumbling window
- Sliding window

Late events

```
streamingDF
  .groupBy(
    window(col("event_time"), "15 minutes", "10 minutes"))
  .count()
```

EVENT TIME WINDOWS – WATERMARKS



Window types

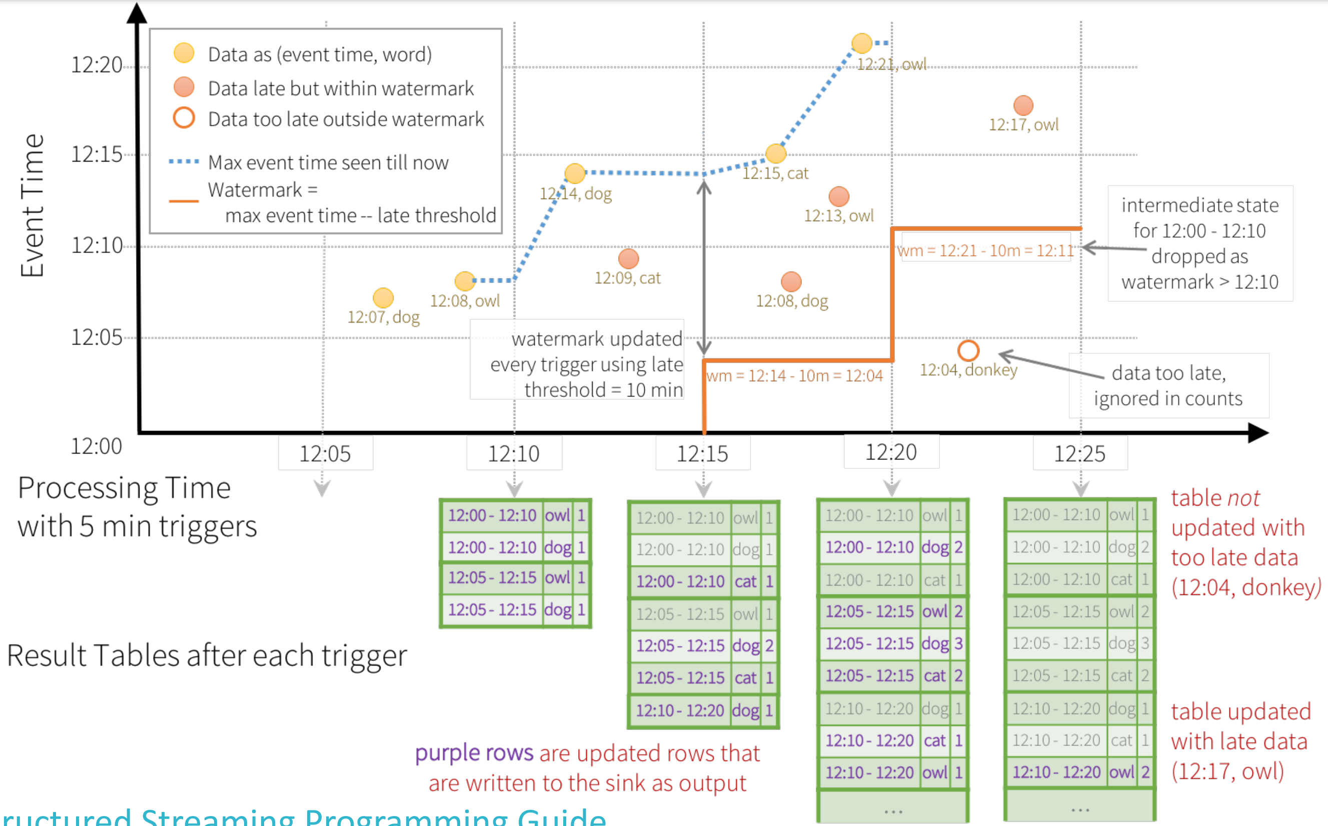
- Tumbling window
- Sliding window

Late events

- Watermarks

```
streamingDF
  .withWatermarks("event_time", "30 minutes")
  .groupBy(
    window(col("event_time"), "15 minutes", "10 minutes"))
  .count()
```

LATE EVENTS AND WATERMARKS


Data as (event time, word)
Data late but within watermark
Data too late outside watermark
Max event time seen till now
Watermark =
max event time -- late threshold
Event Time
12:20
12:15
12:10
12:05
12:00
12:05
12:10
12:15
12:20
12:25
12:07, dog
12:08, owl
12:14, dog
12:09, cat
12:15, cat
12:08, dog
12:13, owl
12:21, owl
12:17, owl
12:04, donkey
watermark updated every trigger using late threshold = 10 min
wm = 12:14 - 10m = 12:04
wm = 12:21 - 10m = 12:11
intermediate state for 12:00 - 12:10 dropped as watermark > 12:10
data too late, ignored in counts
Processing Time with 5 min triggers
Result Tables after each trigger
table *not* updated with too late data (12:04, donkey)
table updated with late data (12:17, owl)

12:10

12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	1

12:15

12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:00 - 12:10	cat	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	2
12:05 - 12:15	cat	1
12:10 - 12:20	dog	1

12:20

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	1
...		

12:25

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	2
...		

purple rows are updated rows that are written to the sink as output

More info: Structured Streaming Programming Guide

STREAMING – TAKE AWAY

- Apache Kafka is a persistent, scalable messaging platform
- Structured Streaming is „Dataframe but streaming”
- A stream is an unbounded table
- The same API for batch and streaming
- Event time != processing time
- Late events & watermarks

Questions?