# Week 11

## FIT5202 Big Data Processing

Data Streaming using Apache Kafka and Spark

Spark Structured Streaming

Aggregations on Windows over event-time

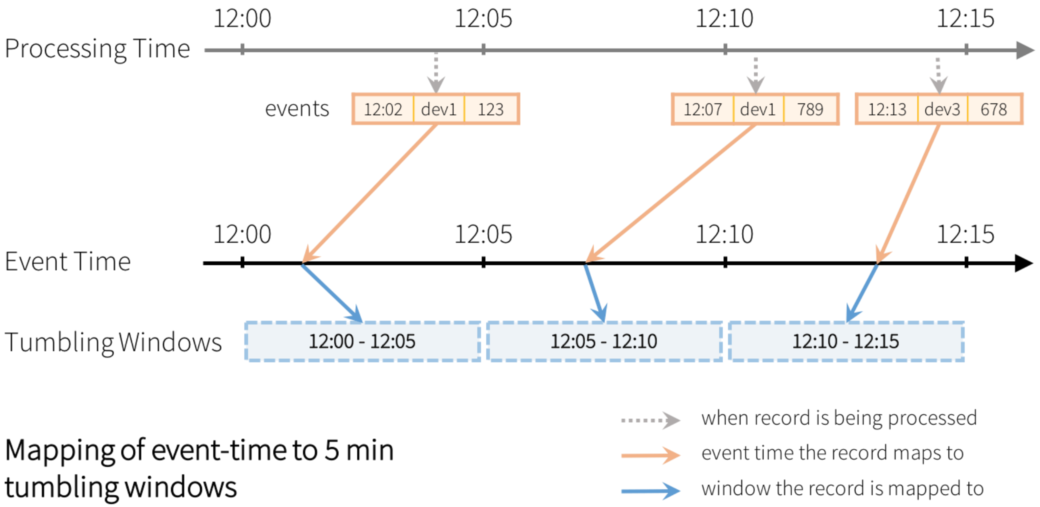
Handling Late events with water marking

# Week 11 Agenda

- Week 10 Review
    - Structured Streaming
    - Integration with Kafka
    - DEMO
- This Week:
    - Aggregations on Windows over Event Time
    - Handling late data with watermarking
    - “Parquet” sink
    - Checkpointing

# Aggregations on Windows over Event Time

- In many cases (e.g., moving averaging), we want aggregations over data bucketed by time windows (e.g., every 5 minutes) rather than over entire streams
- Bucketing data into windows based on event-time (e.g. the time data generated in the producer) – grouping using window function



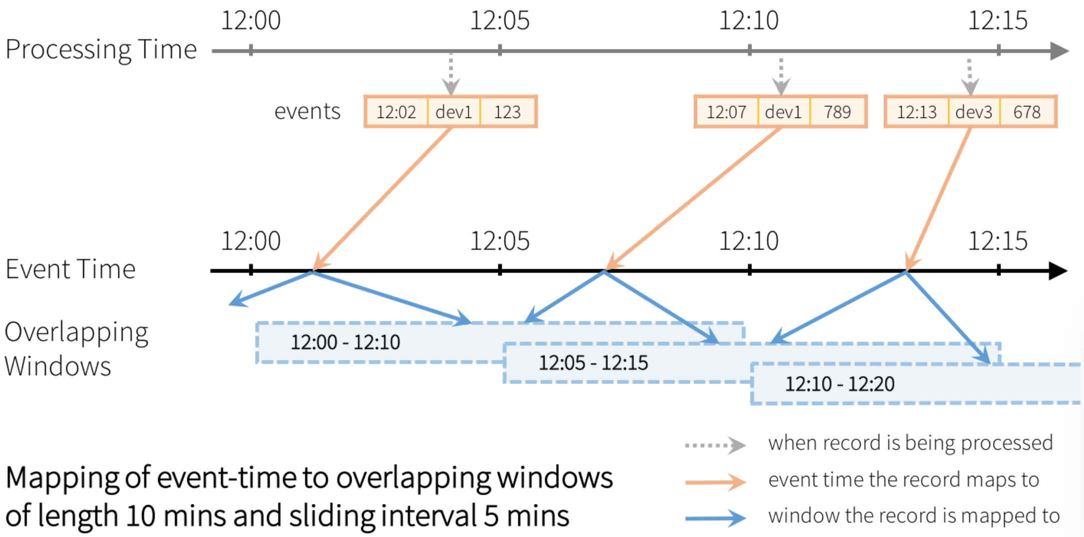

Mapping of event-time to 5 min tumbling windows

Non-overlapping windows

```
from pyspark.sql.functions import *

windowedAvgSignalDF = \
  eventsDF \
    .groupBy(window("eventTime", "5 minute")) \
    .count()
```

# Aggregations on Windows over Event Time

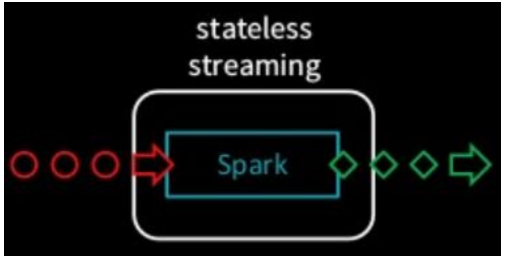


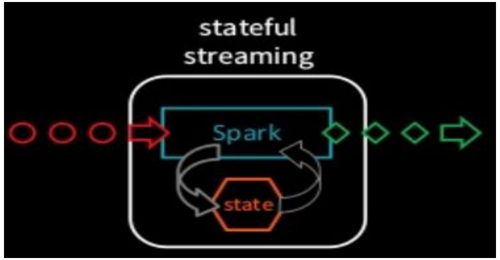

Overlapping windows

```
from pyspark.sql.functions import *

windowedAvgSignalDF = \
  eventsDF \
    .groupBy(window("eventTime", "10 minutes", "5 minutes")) \
    .count()
```

# Stateless vs Stateful Stream Processing




## Stateless

- Each record is processed independently of other records
- e.g. operations like map, filter, join with static data

## Stateful

- Processing of records depends upon the result of previously processed records.
- Need to maintain "intermediate information" for processing – called "state"
- E.g., operations like aggregating count of records (e.g., intermediate count)

## State of Progress

- keeping track of data that has been processed in streaming so far.
- Called checkpointing/saving of offsets of incoming data.

## State of Data

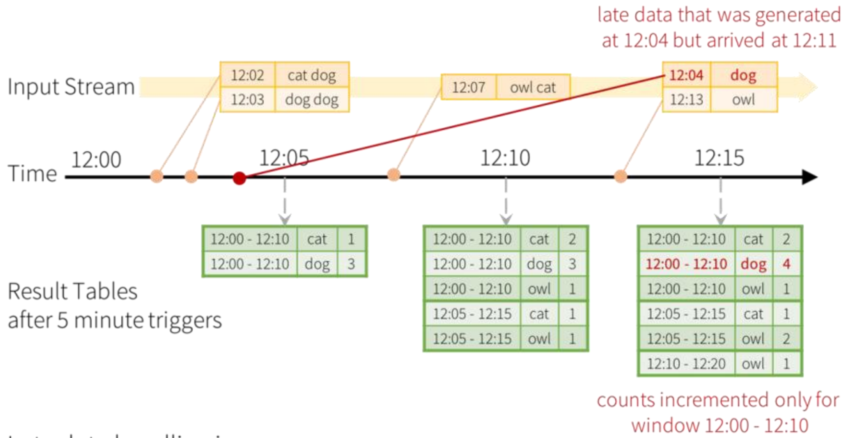
- intermediate information derived from data (processed so far).

https://medium.com/@chandanbaranwal/state-management-in-spark-structured-streaming-aaa87b6c9d31

# Watermarking

Structured Streaming can maintain the intermediate state for partial aggregates (e.g. intermediate count) for a period of time such that late data can update aggregates of old windows correctly

- To handle events that arrive late to the application
- E.g. a word is generated at 12:04 (event time) but received at 12:11 by the application.
- The application should use the time 12:04 instead of 12:11 to update the older counts for a window 12:00 - 12:10.
- Watermarking lets engine automatically track the current event time in data and clean up/update old state accordingly




Late data handling in Windowed Grouped Aggregation

# Watermarking

Two parameters to define the watermark of a query

**(1) event time column**

**(2) Threshold** specify for how late data should be processed (in event time)

```
withWatermark(eventTime: String, delayThreshold: String)
```

late data within the threshold will be aggregated, but data later than the threshold will start getting dropped

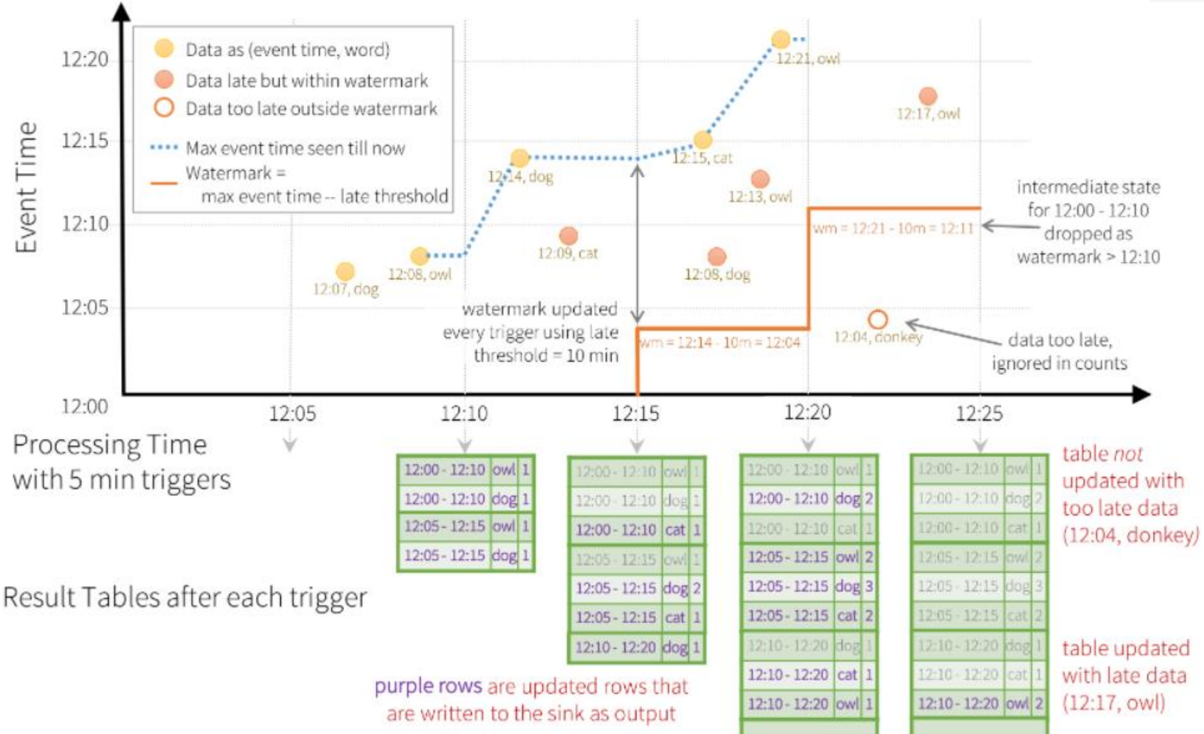
```
# Group the data by window and word and compute the count of each group
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```


**max event time** - is the latest event time seen by the engine

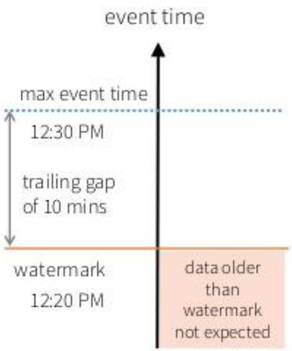
# Watermarking

Window-based aggregation based on event time –
Window size = 10 mins
Slide = 5 mins

```
# Group the data by window and word and compute the count of each group
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```




Watermarking in Windowed Grouped Aggregation with Update Mode




- Engine will keep updating counts of a window in the Result Table until the window is older than the watermark, which lags behind the current event time by 10 minutes.

# Recovering from Failures with Checkpointing

- In case of failure, can recover the previous progress and state of previous query and continue where it left off.
- Can enable checkpointing using the option **checkpointLocation** on the query.
- To save all the progress information (i.e. range of offsets processed in each trigger) and the running aggregates (‘states’) to the checkpoint location

```
aggDF \
    .writeStream \
    .outputMode("complete") \
    .option("checkpointLocation", "path/to/HDFS/dir") \
    .format("memory") \
    .start()
```

# Clickstream Watermarking

```
schema = StructType([
    StructField('Clicks', IntegerType(), True),
    StructField('Impressions', IntegerType(), True),
    StructField('ts', TimestampType(), True)
])
```

Using the schema, we convert the data to a Spark DataFrame

```
df=df.select(F.from_json(F.col("value").cast("string"), schema).alias('parsed_value'))
```

```
df = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Kafka producer
Late events

```
------------------------
2021-05-20 23:46:15 impressions: 7
2021-05-20 23:46:15 impressions: 9
2021-05-20 23:46:15 impressions: 4
2021-05-20 23:46:15 impressions: 2
2021-05-20 23:46:15 impressions: 4
2021-05-20 23:46:15 impressions: 1
2021-05-20 23:46:15 impressions: 7
2021-05-20 23:46:15 impressions: 3
2021-05-20 23:46:15 impressions: 7
2021-05-20 23:46:15 impressions: 4
2021-05-20 23:46:10 impressions: 8
2021-05-20 23:46:10 impressions: 2
2021-05-20 23:46:10 impressions: 5
------------------------
2021-05-20 23:46:20 impressions: 5
2021-05-20 23:46:20 impressions: 1
2021-05-20 23:46:20 impressions: 4
```

Cast to String

Parse the string data in json format according to the schema

```
df_formatted.printSchema()
root
 |-- Clicks: integer (nullable = true)
 |-- Impressions: integer (nullable = true)
 |-- ts: timestamp (nullable = true)
```

| key | value | topic | partition | offset | timestamp |
|---|---|---|---|---|---|
| [binary] | [binary] | "topic" | 0 | 345 | 1486087873 |
| [binary] | [binary] | "topic" | 3 | 2890 | 1486086721 |

Spark subscribe to topic & read data

```
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

| Clicks | Impressions | ts |
|---|---|---|
| 0 | 6 | 1602944650 |
| 0 | 10 | 1602944650 |
| 0 | 5 | 1602944650 |

Aggregations on Windows over Event-time

```
windowedCounts = df_formatted \
    .withWatermark("ts", "10 seconds") \
    .groupBy(window(df_formatted.ts, "10 seconds"))\
    .agg(F.sum("Impressions").alias("total"))\
    .select("window","total")
```

Output result table

```
+------------------------------------------+-----+
|window                                    |total|
+------------------------------------------+-----+
|[2021-05-20 23:52:00, 2021-05-20 23:52:10]|124  |
|[2021-05-20 23:51:40, 2021-05-20 23:51:50]|138  |
|[2021-05-20 23:52:20, 2021-05-20 23:52:30]|40   |
|[2021-05-20 23:51:30, 2021-05-20 23:51:40]|85   |
|[2021-05-20 23:51:50, 2021-05-20 23:52:00]|134  |
|[2021-05-20 23:52:10, 2021-05-20 23:52:20]|133  |
+------------------------------------------+-----+
```

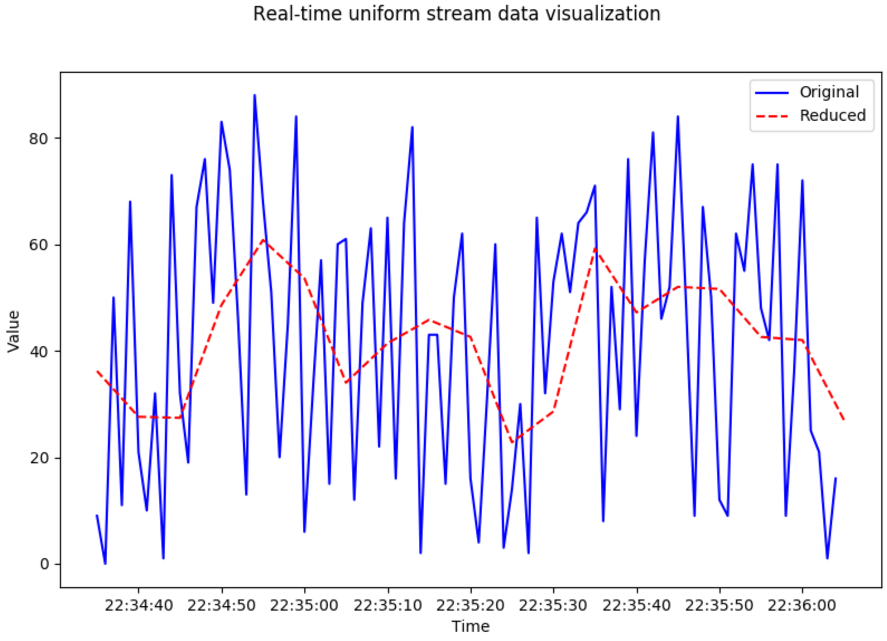
```
query = windowedCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .trigger(processingTime='5 seconds') \
    .option("truncate","false")\
    .start()
```

For aggregation query, use 'complete' mode

# Granularity Reduction DEMO

(Refer to Lecture)

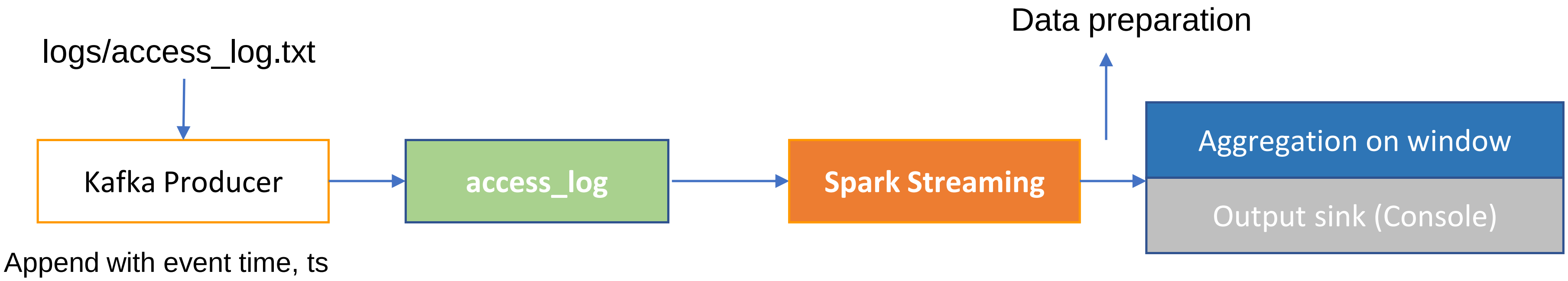
The Aggregation on Windows over Event-time is another way of understanding the concept of granularity reduction.



# Lab Task: Access Log – Window-based Aggregation

**Aggregations on window over event-time**

tsExp=r'(\d{10})\s'
- Extract event timestamp




Each line contains some valuable information such as:

1. Host
2. Timestamp
3. HTTP method
4. URL endpoint
5. Status code
6. Protocol
7. Content Size

Task 1:

- ❑ Using the Window function, find the number of logs for each status in a window of 30 seconds. Set the window slide to 10 seconds
- ❑ Write the output to console sink.

# References


- https://databricks.com/blog/2017/05/08/event-time-aggregation-watermarking-apache-sparks-structured-streaming.html
- https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#quick-example
- https://docs.databricks.com/spark/latest/structured-streaming/production.html
- http://blog.madhukaraphatak.com/introduction-to-spark-structured-streaming-part-7/