



Spark Structured Streaming

Vybornov Artyom, avybornov@bigdatateam.org

Big Data Instructor, <http://bigdatateam.org/>

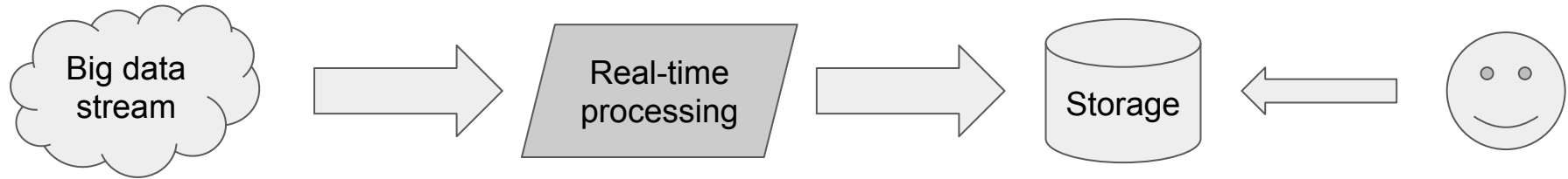
Head of Data Platform, Rambler Group

<https://www.linkedin.com/in/artvybor/>



Spark Structured Streaming

- ▶ **Intro**
- ▶ Application example
- ▶ Hints



- ▶ Apache Spark Streaming - classic
- ▶ Apache Spark Structured Streaming - new wave (:



Apache Spark Structured Streaming

Spark Structured Streaming provides real-time stream processing letting the user leave its details unattended

- ▶ DataFrame -> DataFrame
- ▶ Micro-batch approach
- ▶ Event-based approach (2.3.0+)



```
input_df = spark \  
    .read \  
    .format("kafka") \  
    .option("kafka.bootstrap.servers", "host:port") \  
    .option("subscribe", "topic") \  
    .load()  
  
result = input_df \  
    .select(  
        explode(split(lines.value, " ")) \  
    ).alias("word")  
  
result.write \  
    .format("parquet") \  
    .save("path/to/dst/dir")
```



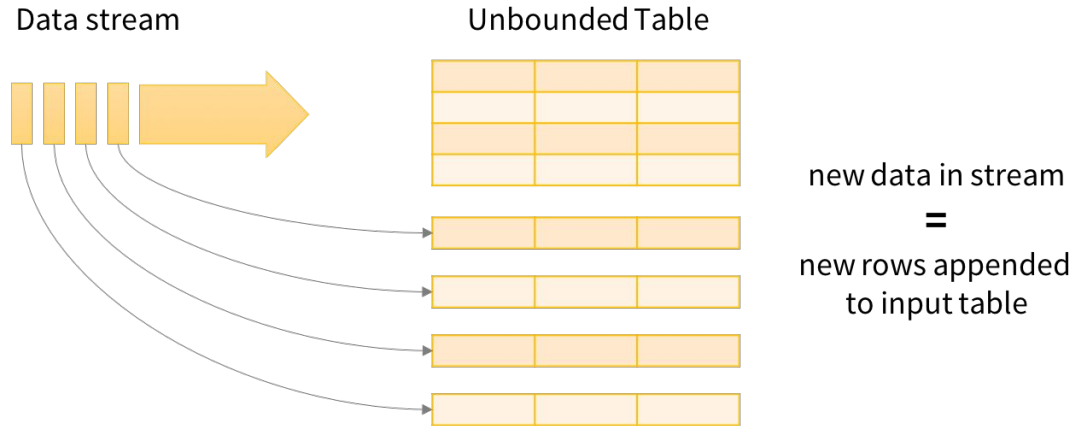
Spark Structured Streaming

```
input_df = spark \  
    .readStream \  
    .format("kafka") \  
    .option("kafka.bootstrap.servers", "host:port") \  
    .option("subscribe", "topic") \  
    .load()  
  
result = input_df \  
    .select(  
        explode(split(lines.value, " ")) \  
    .alias("word"))  
  
result.writeStream \  
    .format("parquet") \  
    .option("path", "path/to/dst/dir") \  
    .start()
```



Unbounded table

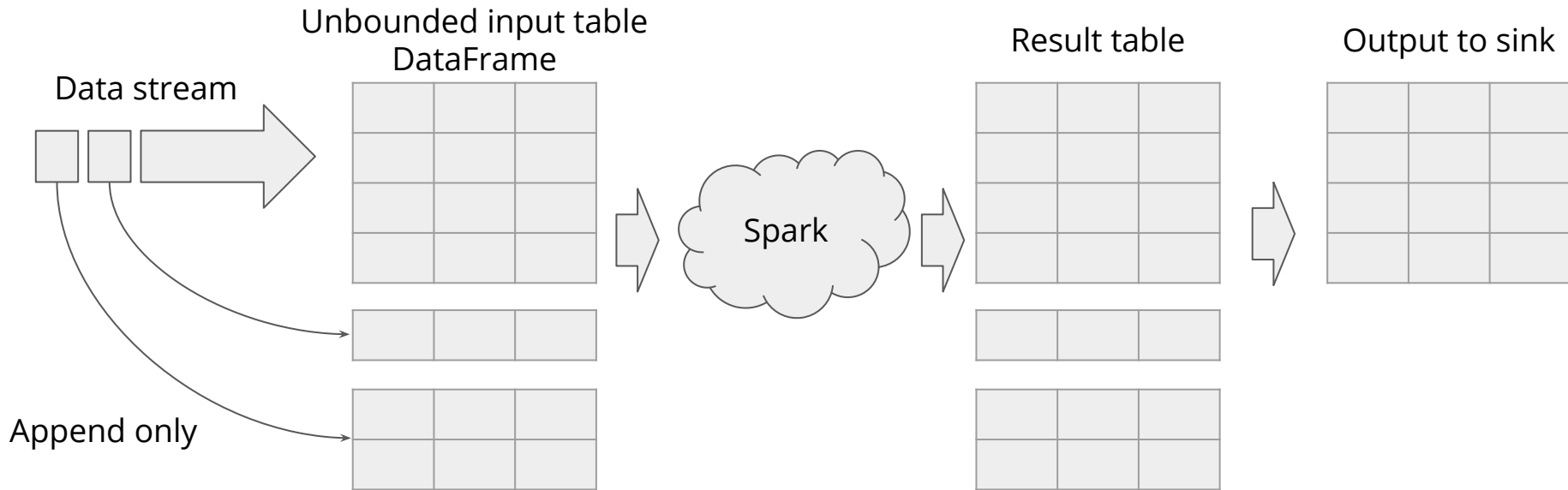
- Each new item in the stream is like a row appended to the input table



Data stream as an unbounded Input Table

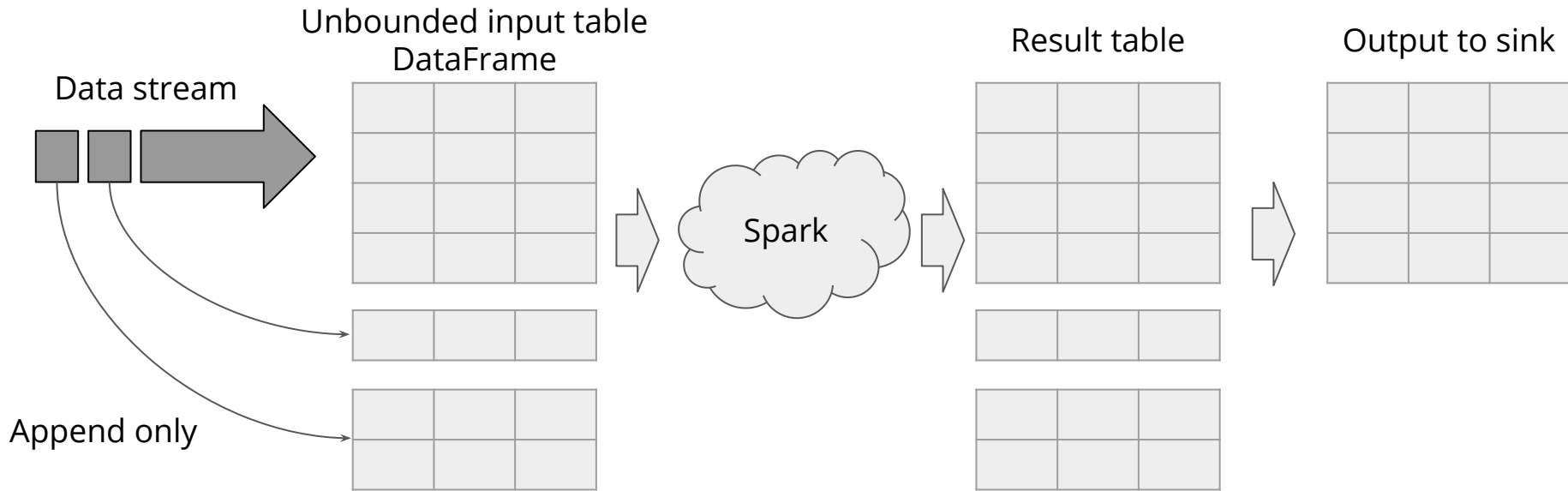


Structured Streaming Base





Input sources





- ▶ File source - reads files written in a directory as a stream of data
- ▶ Kafka source - reads data from Kafka
- ▶ Socket source - reads text data from a socket connection
- ▶ Rate source - generates data at the specified number of rows per second, each output row contains a timestamp and value
 - ▶ Useful for testing



Input sources examples

► Rate source

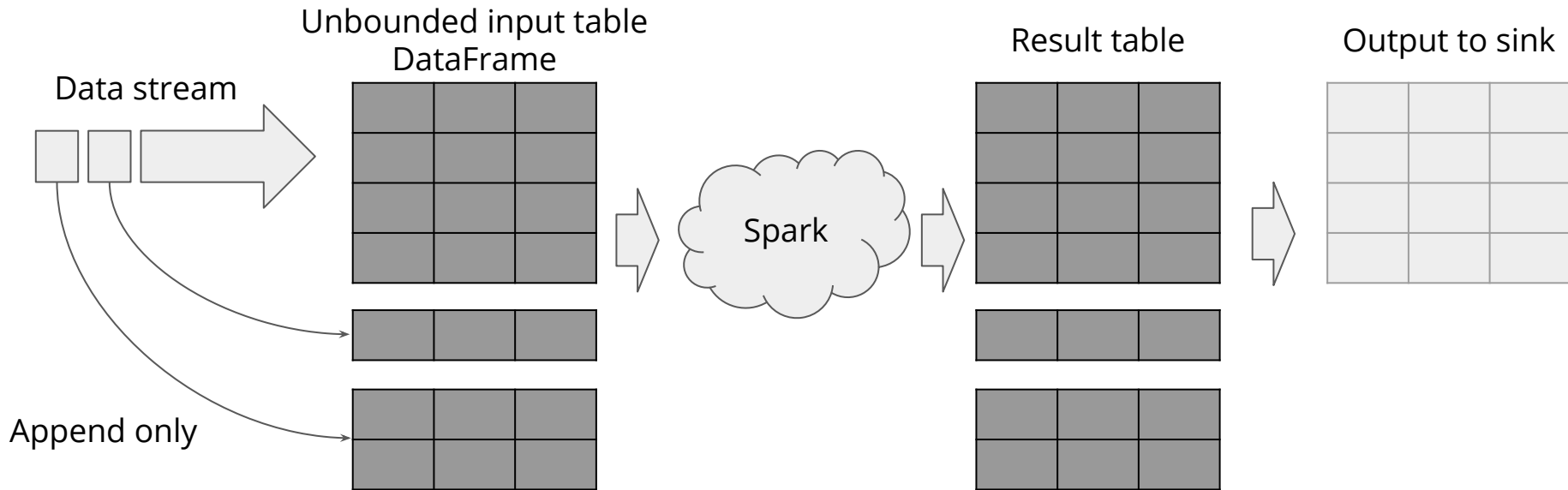
```
input_df = spark \  
    .readStream \  
    .format("rate") \  
    .option("rowsPerSecond", 100) \  
    .option("numPartitions", 3) \  
    .load()
```

► Kafka Source

```
input_df = spark \  
    .readStream \  
    .format("kafka") \  
    .option("kafka.bootstrap.servers", "kafka01.bigdatateam.ru:9092") \  
    .option("subscribe", "my_awesome_topic") \  
    .load()
```



Unbounded dataframe





Unbounded dataframe

```
In [29]: input_df = spark \  
         .readStream \  
         .format("rate") \  
         .option("rowsPerSecond", 100) \  
         .option("numPartitions", 3) \  
         .load()
```

```
In [30]: input_df.isStreaming
```

```
Out[30]: True
```

```
In [31]: input_df.printSchema()
```

```
root  
|-- timestamp: timestamp (nullable = true)  
|-- value: long (nullable = true)
```



```
In [33]: input_df.isStreaming
```

```
Out[33]: True
```

```
In [34]: input_df.createOrReplaceTempView("logs")  
count_df = spark.sql("select count(*) from logs")
```

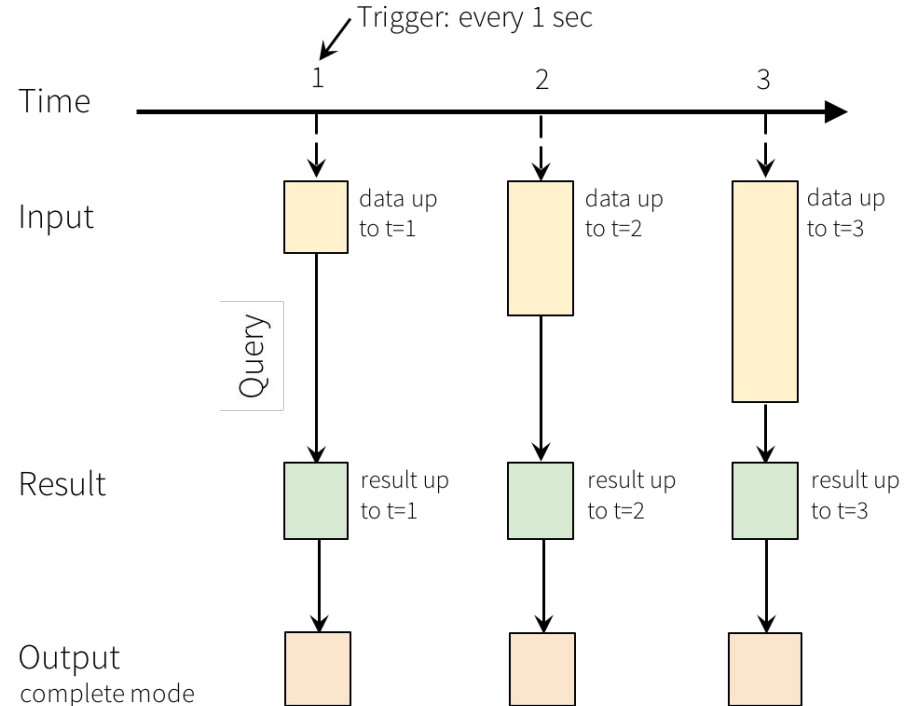
```
In [35]: count_df.isStreaming
```

```
Out[35]: True
```



Structured Streaming Trigger

- Trigger determine when the processing on the accumulated data is started





- ▶ Default - micro-batches will be generated as soon as the previous micro-batch has completed processing
- ▶ Fixed interval micro-batches - micro-batches will be kicked off at the user-specified intervals (similar to spark streaming)
- ▶ One-time micro-batch - only one batch will be executed to process all the available data
 - ▶ Reprocessing of historical data
 - ▶ Testing purposes
- ▶ Continuous with fixed checkpoint interval (alpha) - continuous processing mode (similar event-based approach)



► Fixed interval micro-batches

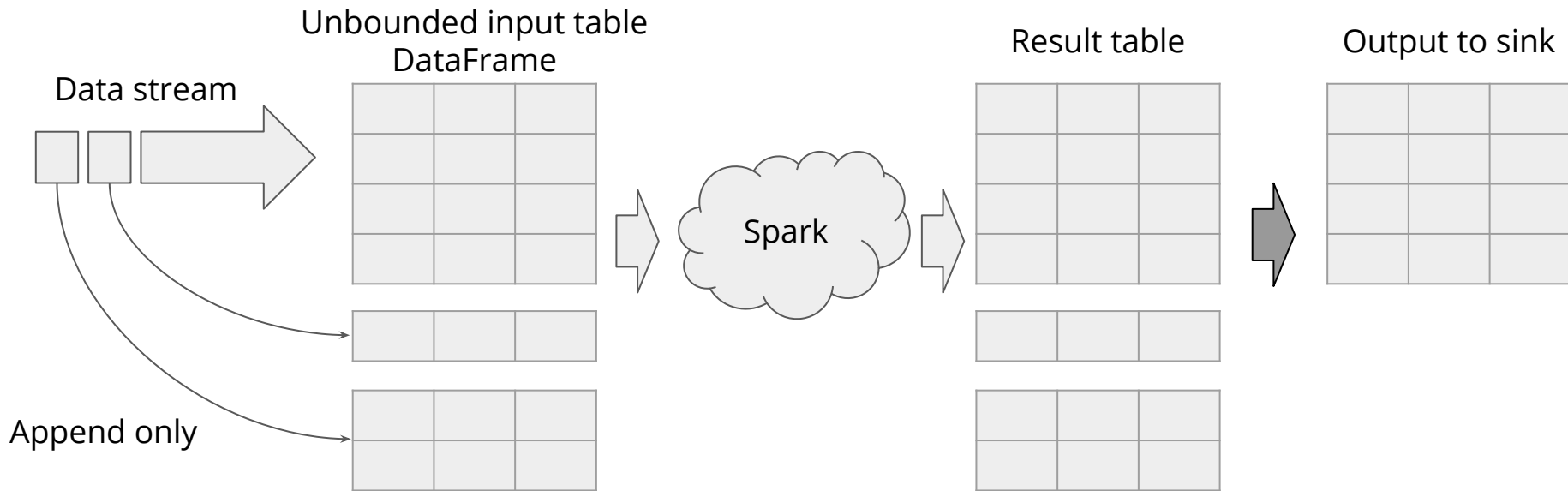
```
query = output_df \  
    .writeStream \  
    .format("console") \  
    .trigger(processingTime='2 seconds') \  
    .start()
```

► One-time micro-batch

```
query = output_df \  
    .writeStream \  
    .format("console") \  
    .trigger(once=True) \  
    .start()
```



The output mode

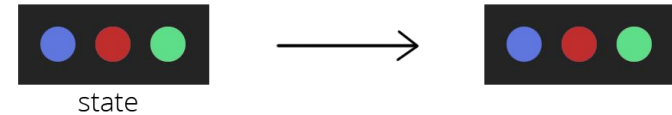




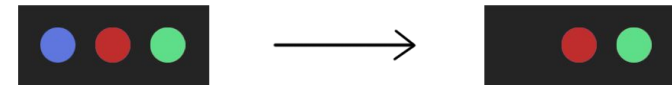
The output mode

- ▶ Result table => Output to sink
- ▶ **Complete Mode** - the whole result table will be outputted to the sink after every trigger
- ▶ **Update Mode** - only the rows in the result table that were updated since the last trigger will be outputted to the sink
- ▶ **Append Mode (default)** - only the new rows added to the result table since the last trigger will be outputted to the sink

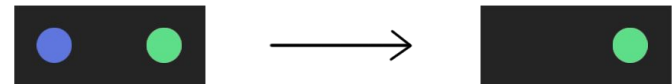
Complete Mode



Update Mode



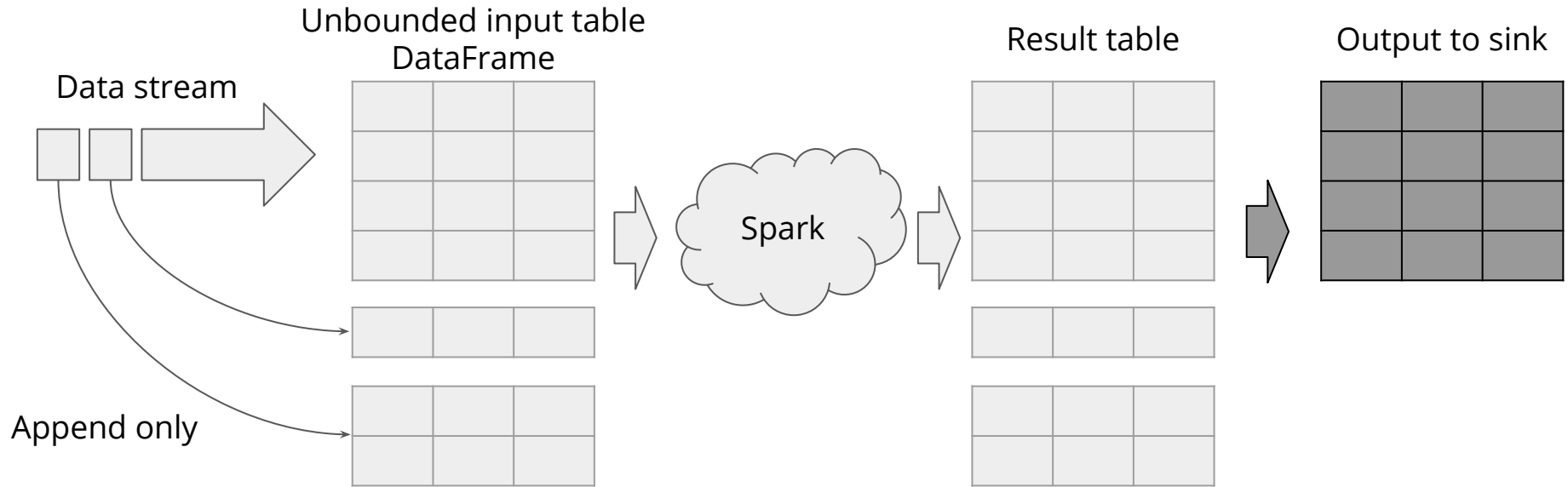
Append Mode



- from last batch result
- modified from last batch
- new in this batch



Output sinks





- ▶ File sink - stores the output to a directory
- ▶ Kafka sink - stores the output to one or more topics in Kafka
- ▶ Foreach sink - runs arbitrary computation on the records in the output (Python support in Spark 2.4.0+)
- ▶ Console sink - prints the output to the console/stdout every time there is a trigger
- ▶ Memory sink - the output is stored in memory as an in-memory table (for debugging)



► Console sink

```
query = output_df \  
    .writeStream \  
    .outputMode("append") \  
    .format("console") \  
    .option("truncate", "false") \  
    .start()
```

► Foreach Sink

```
def foreach_batch_function(df, epoch_id):  
    # Transform and write df  
    pass  
  
query = output_df \  
    .writeStream \  
    .foreachBatch(foreach_batch_function) \  
    .start()
```



Spark Structured Streaming

- ▶ Intro
- ▶ **Application example**
- ▶ Hints

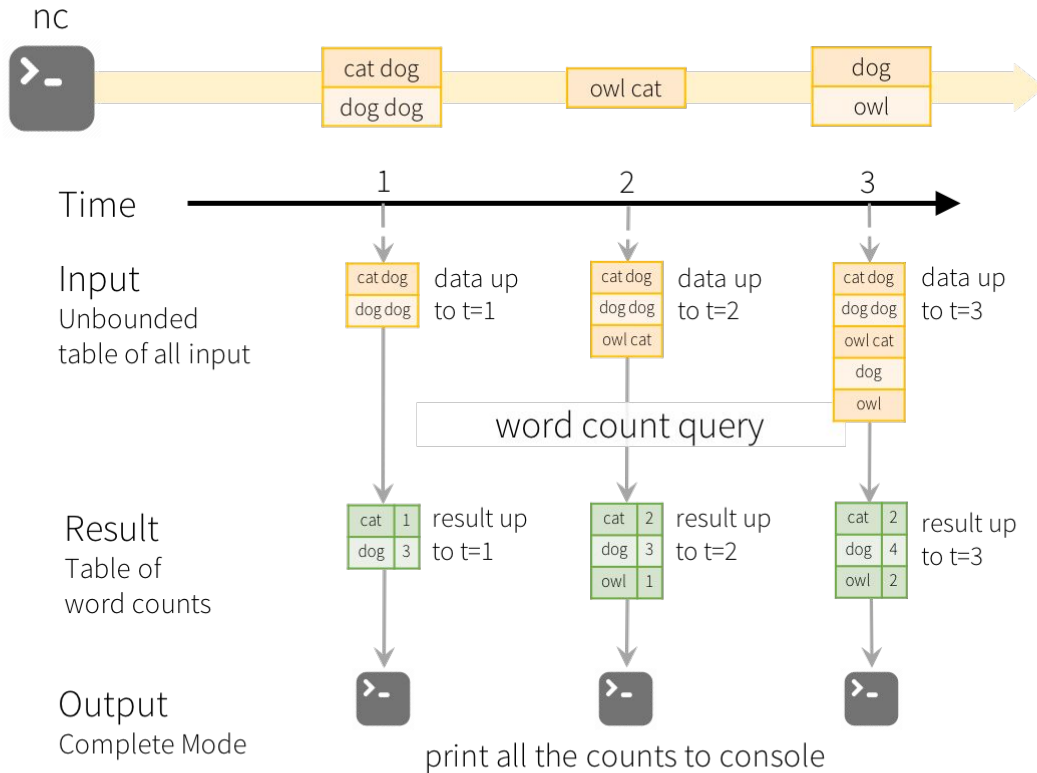


- ▶ Create Spark Session
- ▶ Define input source
- ▶ Transform DataFrame
- ▶ Write result to output
- ▶ Start query
- ▶ Wait for exit
(only for CLI)
- ▶ In Jupyter use query.stop()

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import explode
3 from pyspark.sql.functions import split
4
5 spark = SparkSession \
6     .builder \
7     .appName("word_count") \
8     .getOrCreate()
9
10 lines = spark \
11     .readStream \
12     .format("socket") \
13     .option("host", "localhost") \
14     .option("port", 9999) \
15     .load()
16
17 words = lines.select(explode(split(lines.value, " ")).alias("word"))
18 wordCounts = words.groupBy("word").count()
19
20 query = wordCounts \
21     .writeStream \
22     .outputMode("complete") \
23     .format("console") \
24     .start()
25
26 query.awaitTermination()
```




Example of App





Spark Structured Streaming

- ▶ Intro
- ▶ Application example
- ▶ **Hints**



Window operation

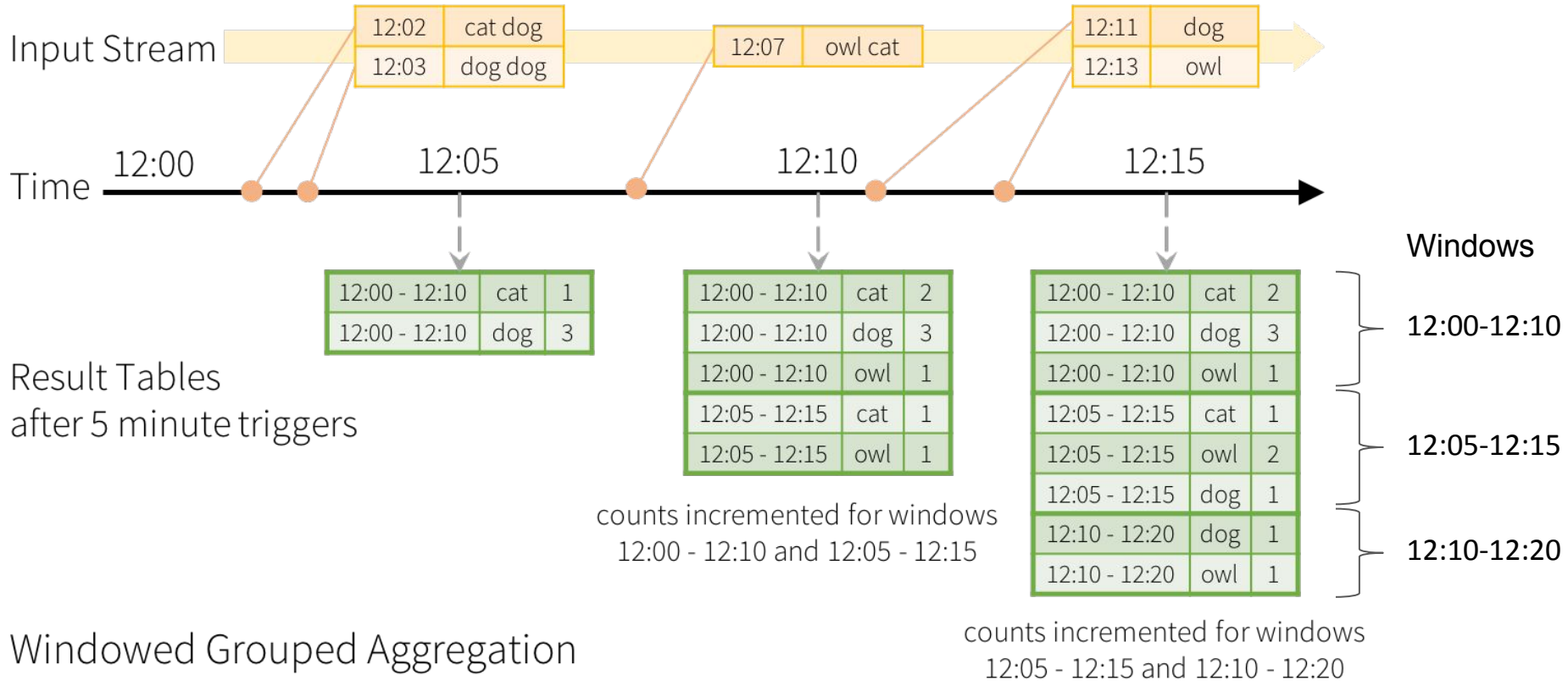
- ▶ Window operation provides a windowed approach, which allows you to apply transformations over a sliding window of data
- ▶ It is very similar to grouped aggregations

```
1 words = ... # streaming DataFrame of schema { timestamp: Timestamp, word: String }
2
3 windowedCounts = words.groupBy(
4     window(words.timestamp, "10 minutes", "5 minutes"),
5     words.word
6 ).count()
```

- ▶ Window size - 10 minutes
- ▶ Window slide - 5 minutes



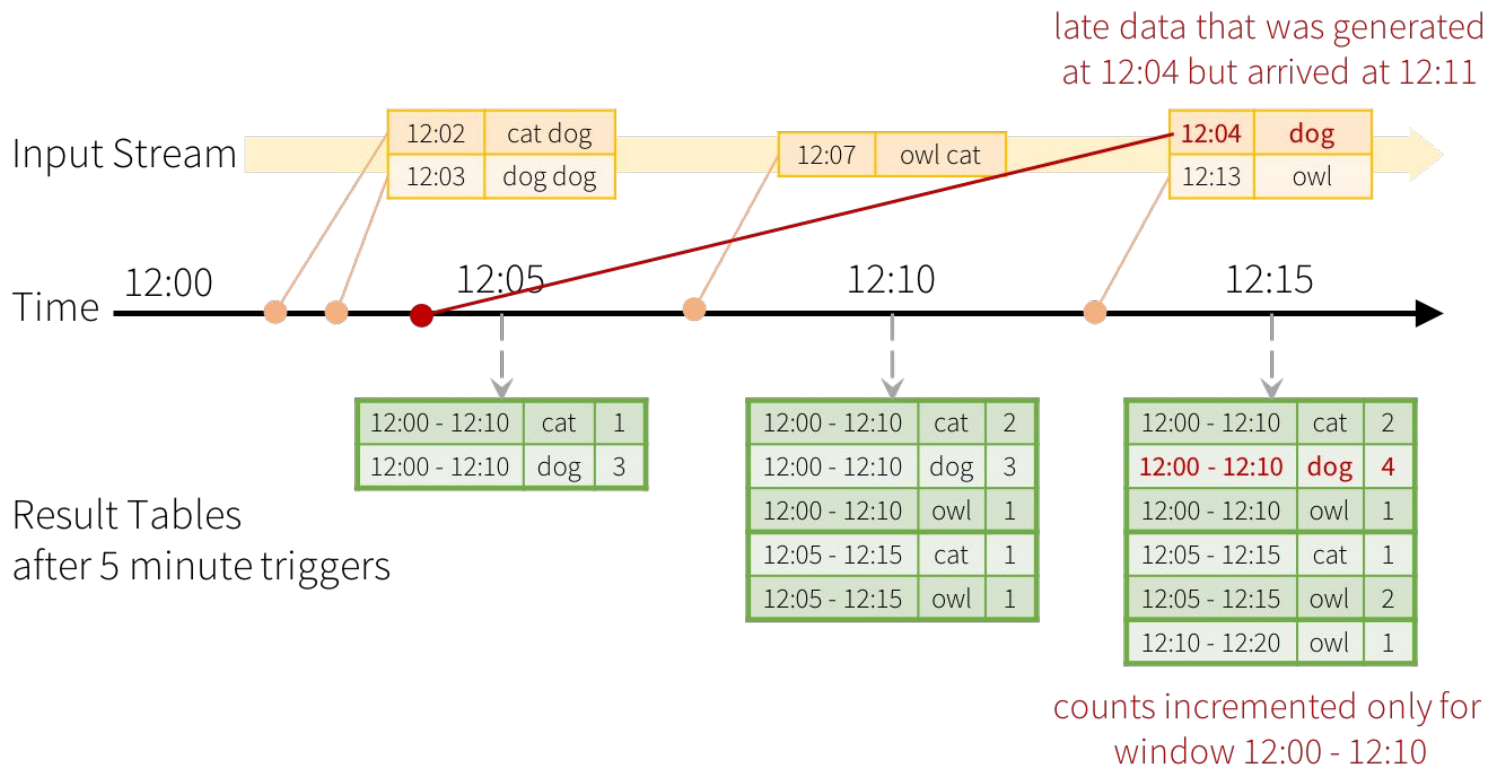
Window operation



Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins



Window operation and late data



Late data handling in
Windowed Grouped Aggregation

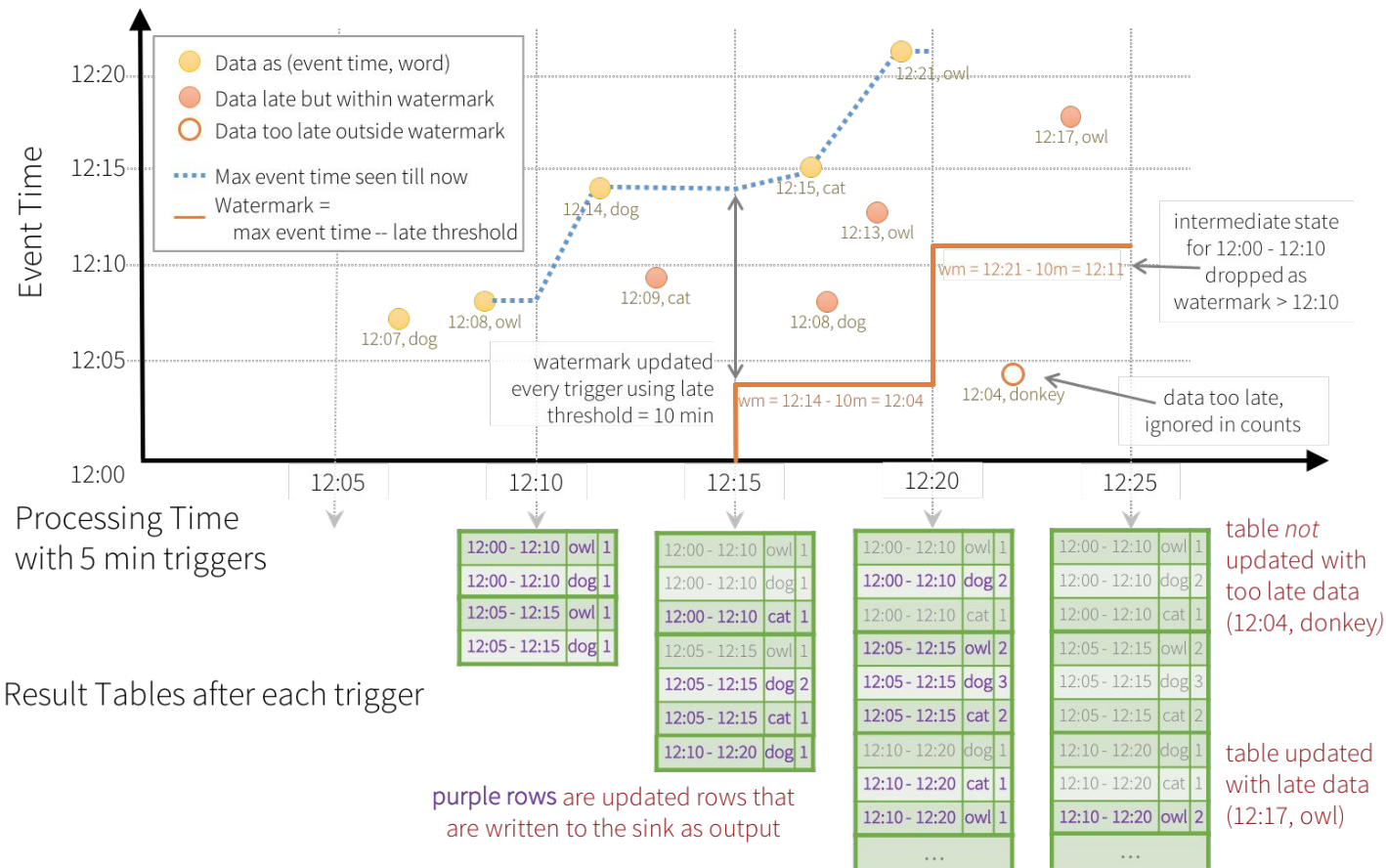


- ▶ Watermarking - lets the engine automatically track the current event time in the data and attempt to clean up old state accordingly.

```
words = ... # streaming DataFrame of schema { timestamp: Timestamp, word: String }  
  
# 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 (window & update)





Query Type VS Output Mode

Query type	Output modes
Without aggregations	Append, Update
With window aggregation and watermark	Append, Update, Complete
Other aggregation	Update, Complete



Spark Structured Streaming

- ▶ Intro
- ▶ Application example
- ▶ Hints



Thank you! Questions?

Vybornov Artyom, avybornov@bigdatateam.org

Big Data Instructor, <http://bigdatateam.org/>

Head of Data Platform, Rambler Group

<https://www.linkedin.com/in/artvybor/>