# Big Data Hadoop and Spark Developer

simplilearn

## Spark Structured Streaming

## Learning Objectives

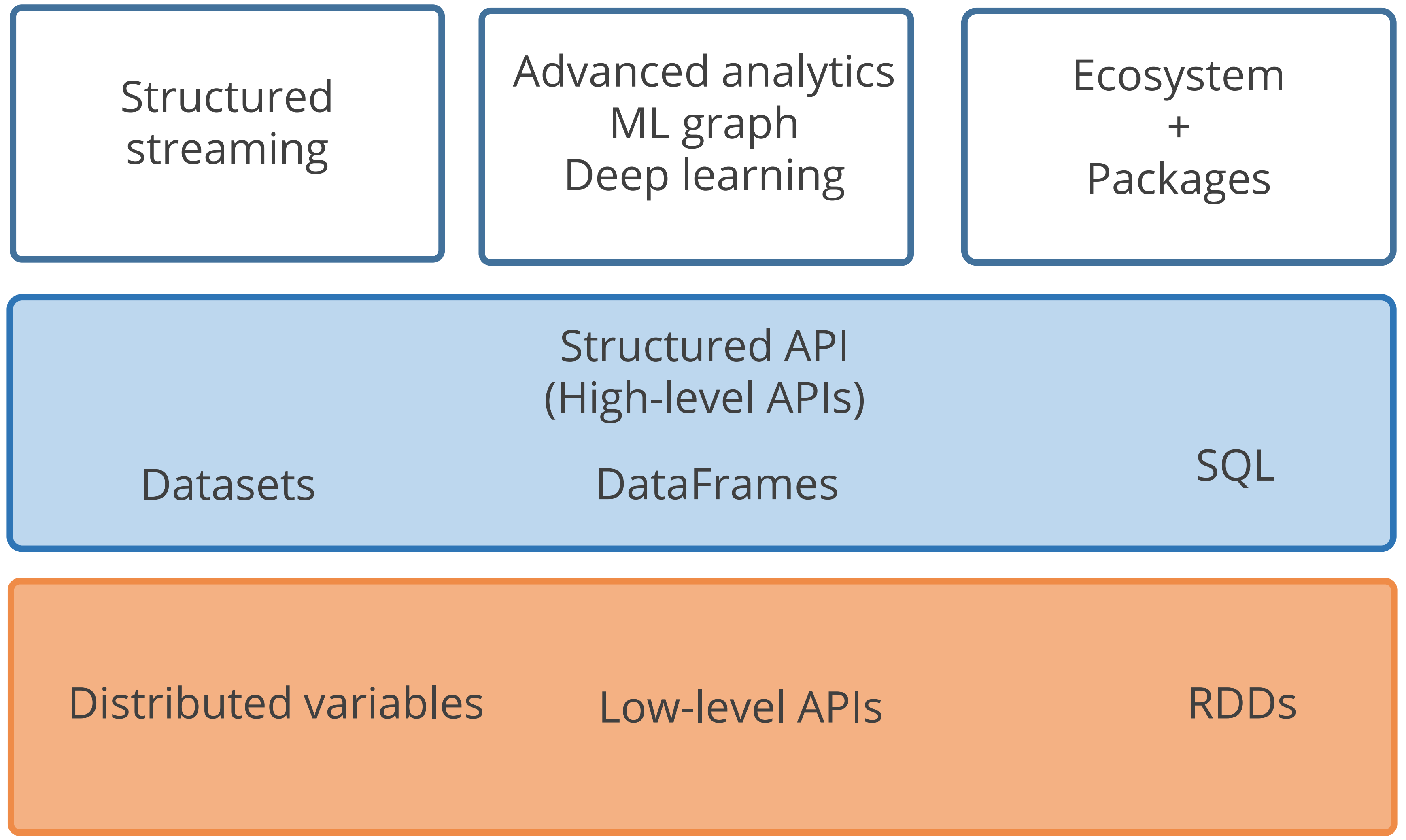
By the end of this lesson, you will be able to:

- Illustrate Structured streaming
- Create streaming DataFrames
- Execute streaming queries
- Demonstrate data sources in Apache Spark streaming
- Summarize Apache Flume and Apache Kafka data sources

## Introduction to Spark Structured Streaming

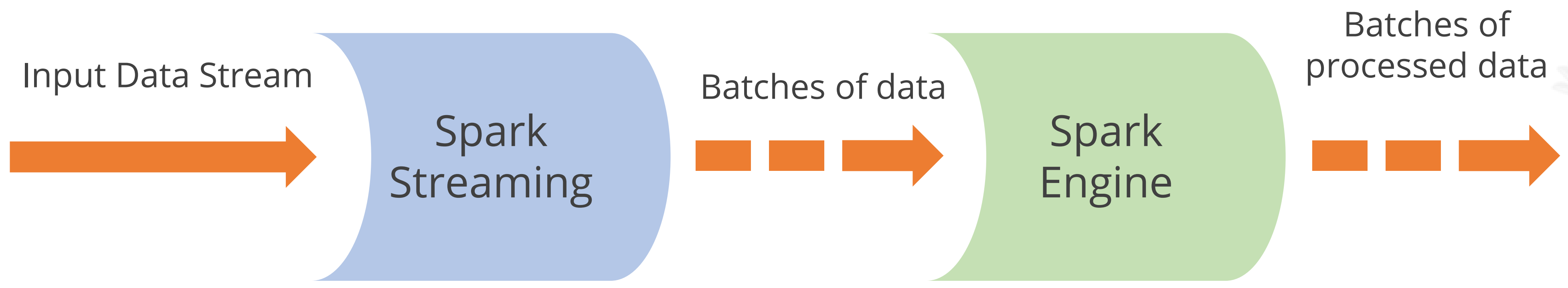
## Spark Structured Streaming

It is a high-level streaming API that is built on the Spark SQL engine. It is also an efficient way to ingest large quantities of data from a variety of sources.



## Spark Structured Streaming

The main concept behind Structured streaming is to consider a live data stream that is being continuously appended. This results in a new stream processing model that is very similar to a batch processing model.



## Unified Batch and Streaming APIs

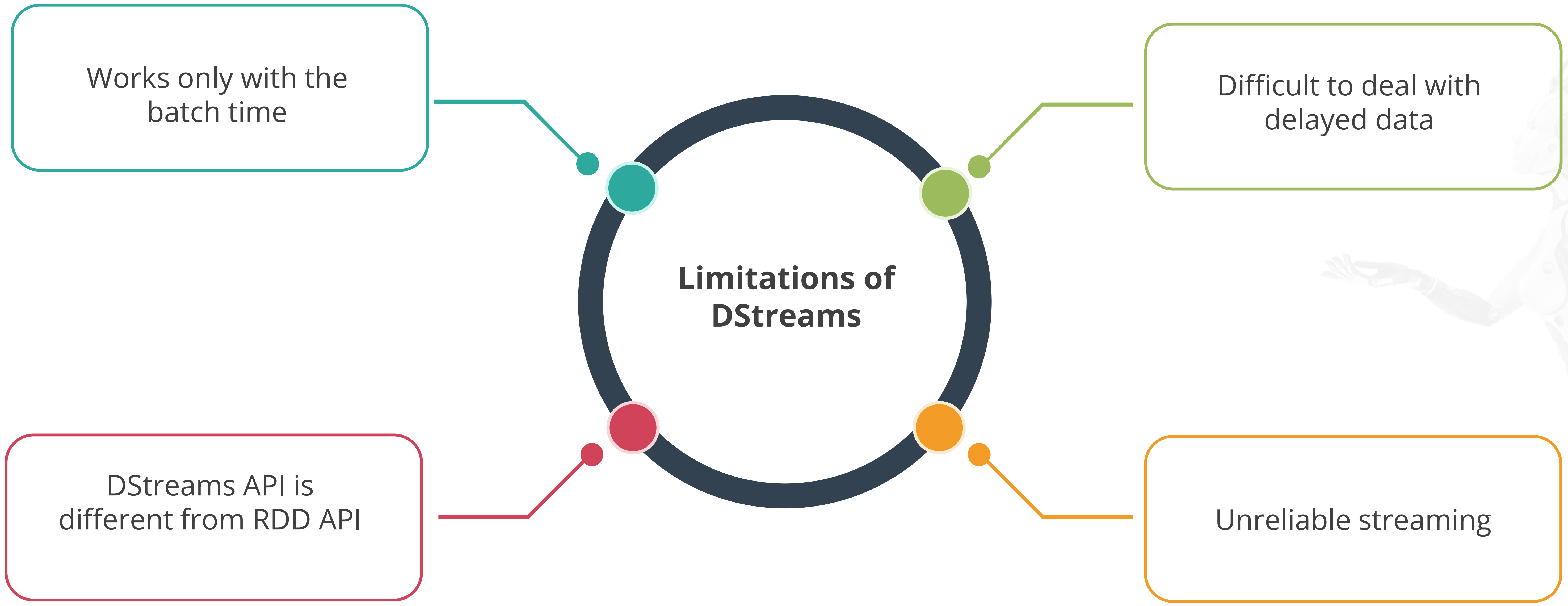
Spark Structured streaming provides the same structured APIs as Spark, so users do not need to develop or maintain two different technology stacks for batch and streaming. In addition, unified APIs make it easy to migrate user's existing batch Spark jobs to streaming jobs.

Batch

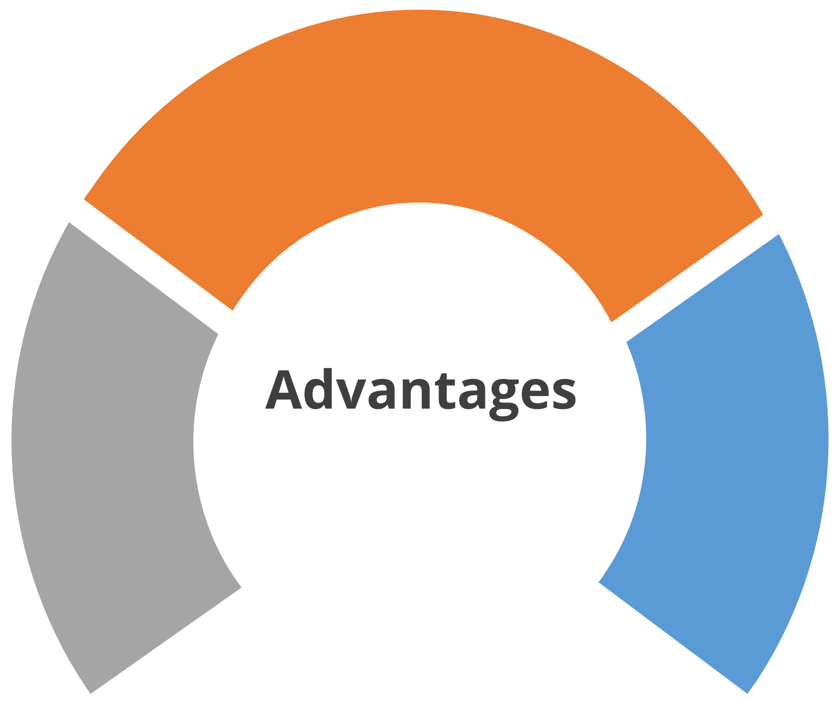
Streaming

## Why Spark Structured Streaming?

Spark Structured streaming is introduced to overcome the shortcomings of DStreams.


Limitations of DStreams
Works only with the batch time
Difficult to deal with delayed data
DStreams API is different from RDD API
Unreliable streaming

## Advantages of Spark Structured Streaming



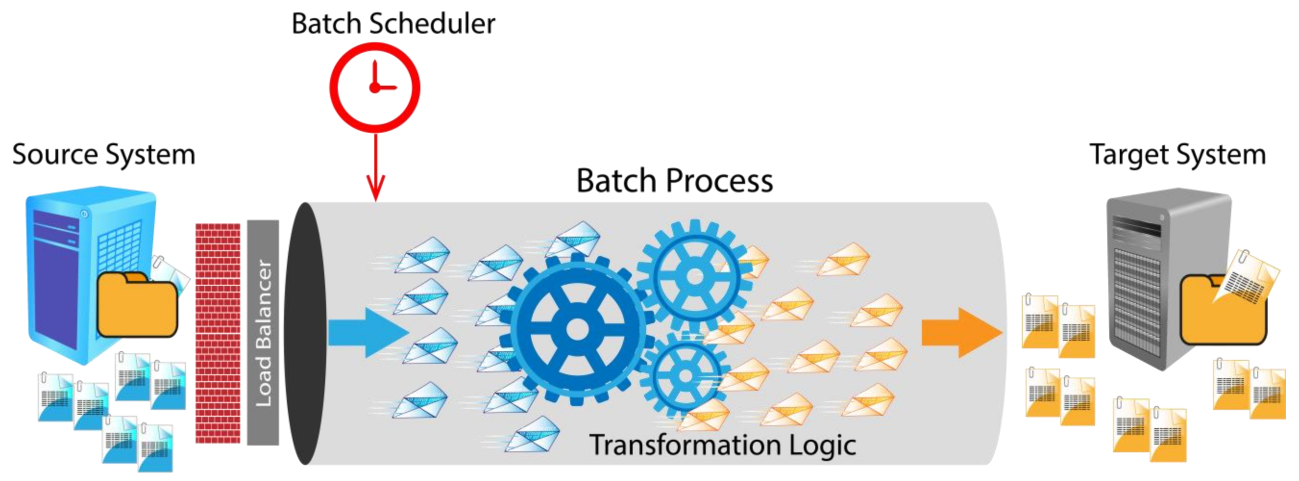
Easy to use

Low latency and cost effective

One unified API for both batch and streaming sources

## Batch vs. Stream Processing

## Batch Processing



- Batch processing refers to the processing of large amounts of data in a single batch over a specified period.
- When the data size is known and finite, batch processing is performed.
- Data is processed in many passes by a batch processor. Batch processing is utilized when data is collected over time and similar data is grouped together.

## Batch Processing

The sample code for batch processing to read data from JSON files.

cmd:

```
personDF = spark.read.schema(callSchema)            // Specify schema using schema() method
                 .json("/sampledata/json/call.json") // Specify the file HDFS path
                 .select("City","Callcharge")             // Selecting specific columns from Dataframe
                 .where("City = 'Paris'")                   // Filtering data based on values
```

spark.read returns a DataFrameReader

# Stream Processing

- Stream processing refers to the processing of real-time streaming data.
- When the data size is unknown, as well as limitless and continuous, stream processing is performed.
- The data output rate in stream processing is the same as the data intake rate.
- Stream processing is utilized when a data stream is continuous and demands an immediate reaction.

## Stream Processing

The sample code for stream processing to read data from JSON files.

cmd:

```
personDF = spark.readStream // readStream returns a DataStreamReader for streaming data
                .schema(callSchema) // Specify schema using schema() method
                .json("/sampledata/json/call.json") // Specify the file HDFS path
```

spark.readStream returns a DataStreamReader

## Advantages of Streaming

Allows data processing with stream data in the same way as with a DataFrame

Allows to focus on the business logic of the application rather than the infrastructure

Reduces the burden on application developers

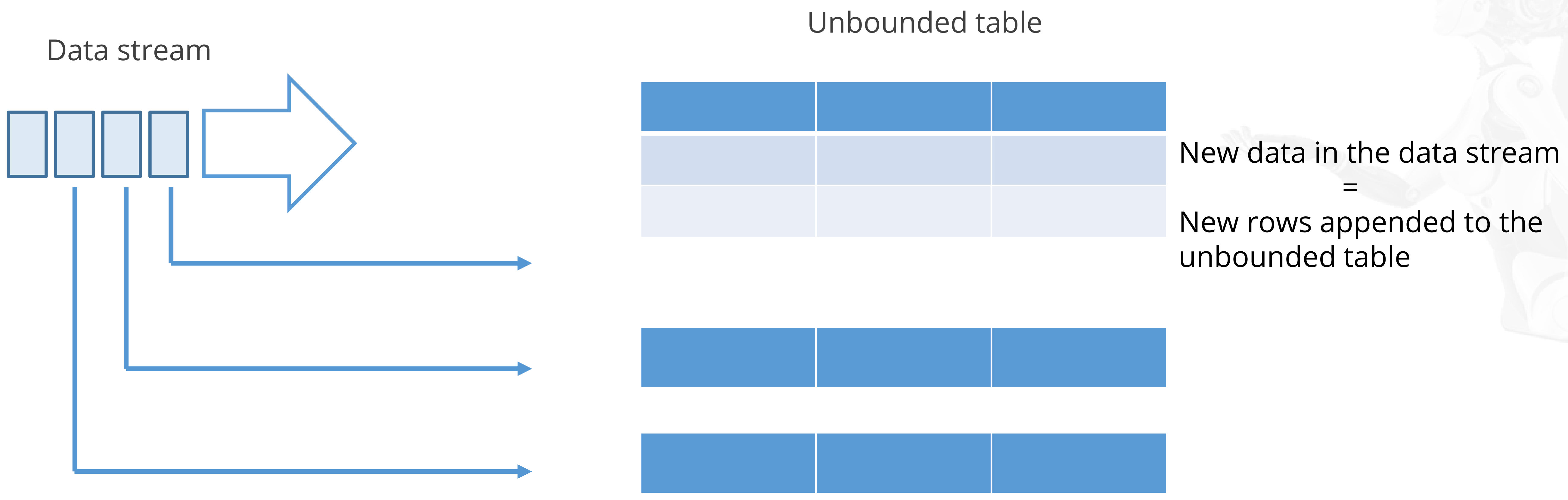
## Spark Streaming vs. Spark Structured Streaming

| Spark Streaming | Spark Structured Streaming |
|---|---|
| Uses DStream abstraction, which is based on RDD | Uses DataFrame or Dataset API |
| Provides low-level RDD operations | User needs to take care of business logic |
| Tracking of state between batch times for cumulative statistics is complex in DStream | Automatically handles consistency and reliability |
| No guarantee of data integrity | Provides guarantee for data integrity |
| API works in micro-batch model | Works in both micro-batch model and continuous processing model |

## Structured Streaming Architecture

## Structured Streaming Architecture

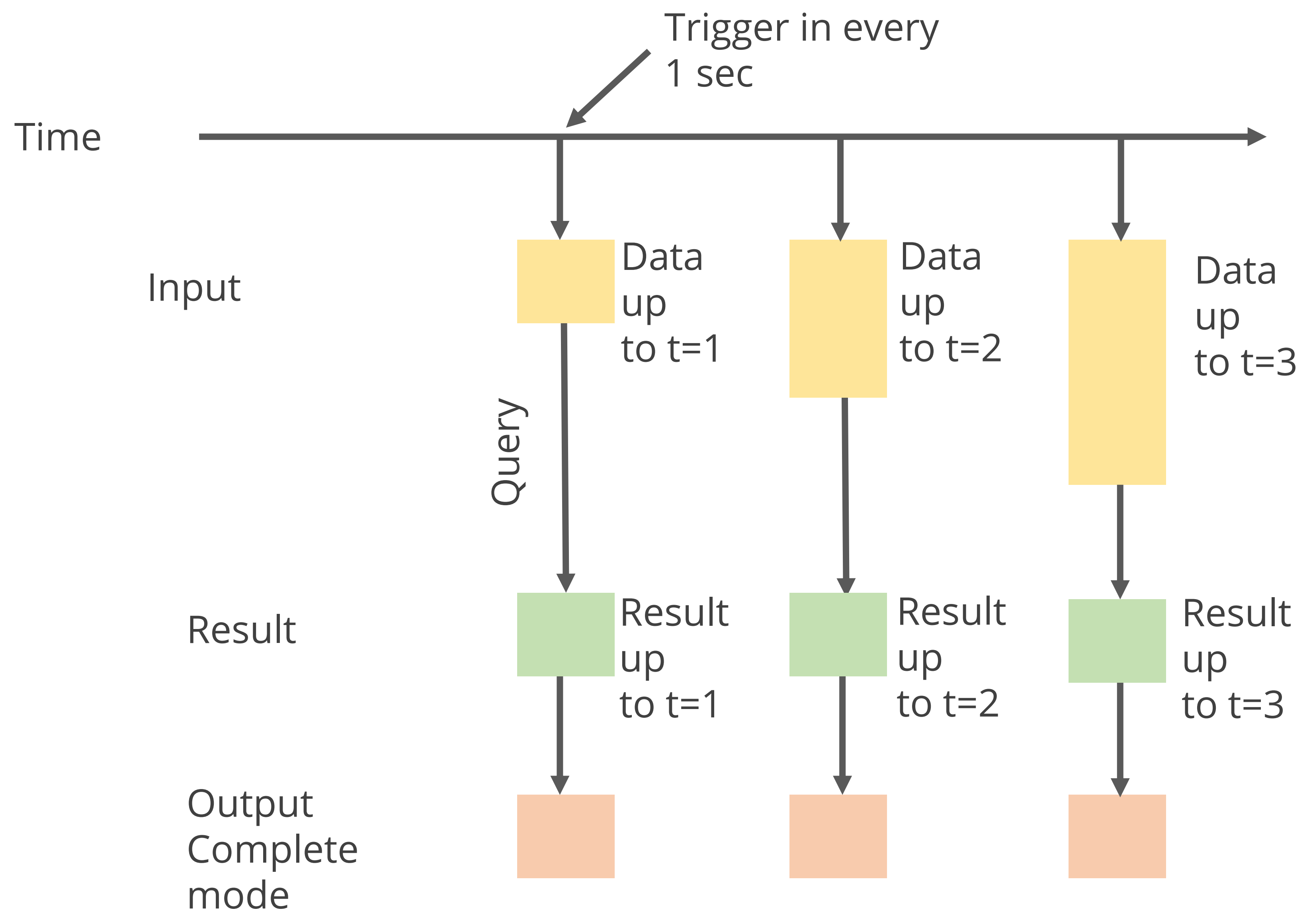
- Structured streaming treats all the arriving data as an **unbounded input table**.
- Every time a new item appears in the stream, it is added to the bottom of the input table as a row.


Data stream
Unbounded table
New data in the data stream
=
New rows appended to the unbounded table

Data stream as an unbounded table

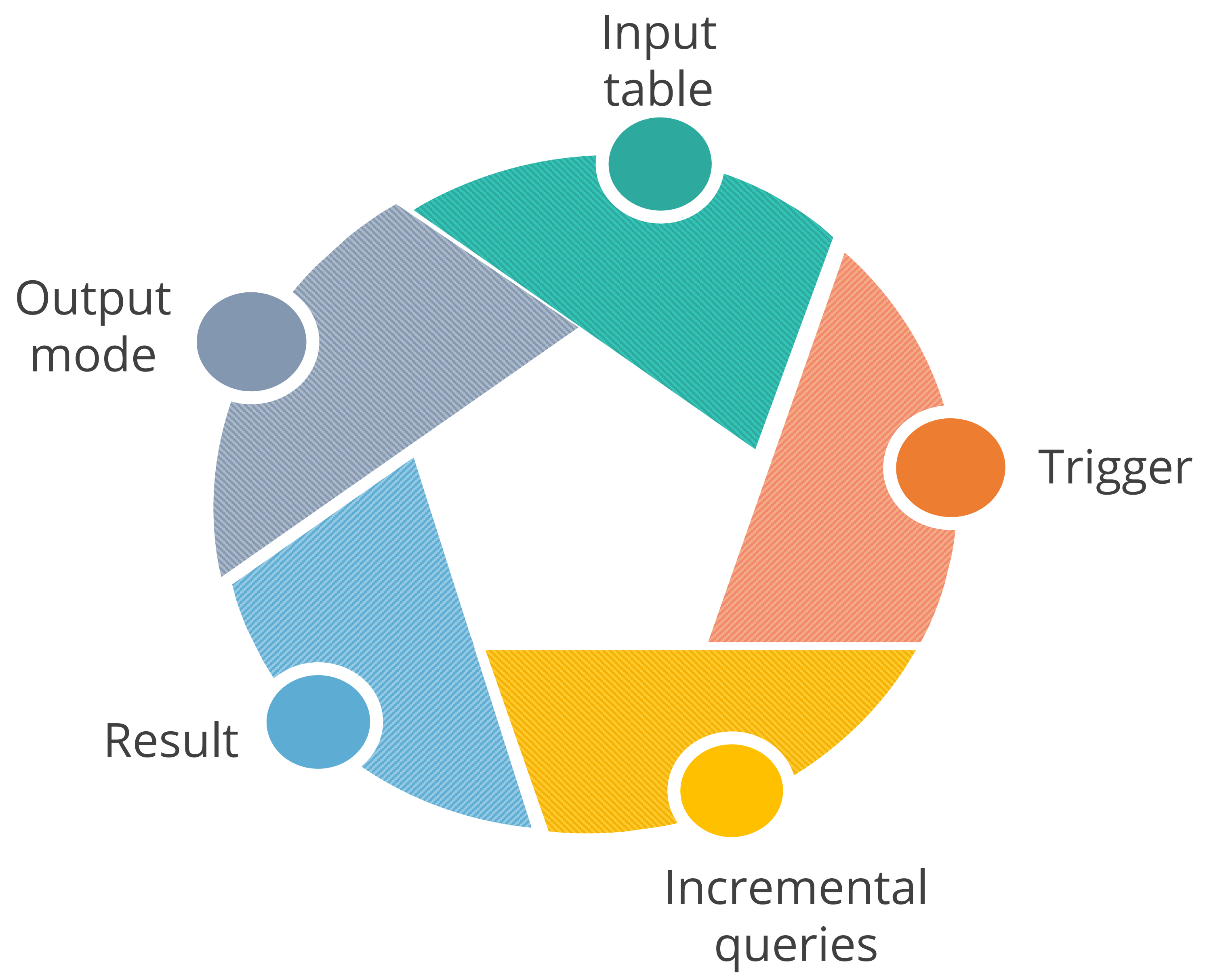
## Structured Streaming Model

Spark Structured streaming enables the user to create low-latency streaming applications and pipelines at a minimal cost. The below diagram illustrates the incremental execution of streaming data:



## Components of Structured Streaming Model

The components of the Structured streaming model are as below:



## Input Sources

For data ingestion, Spark supports a variety of input sources.

| | |
|---|---|
| File Source | The DataStreamReader uses a file source to stream data from a directory. The supported formats are Text, CSV, JSON, and Parquet. |
| Apache Kafka | Apache Kafka is used to read data. This only works with Kafka versions 0.10.0 and onwards. |

## Input Sources

Rate Source

It generates random data with two columns: *timestamp* and *value,* and it is recommended to be used for testing.

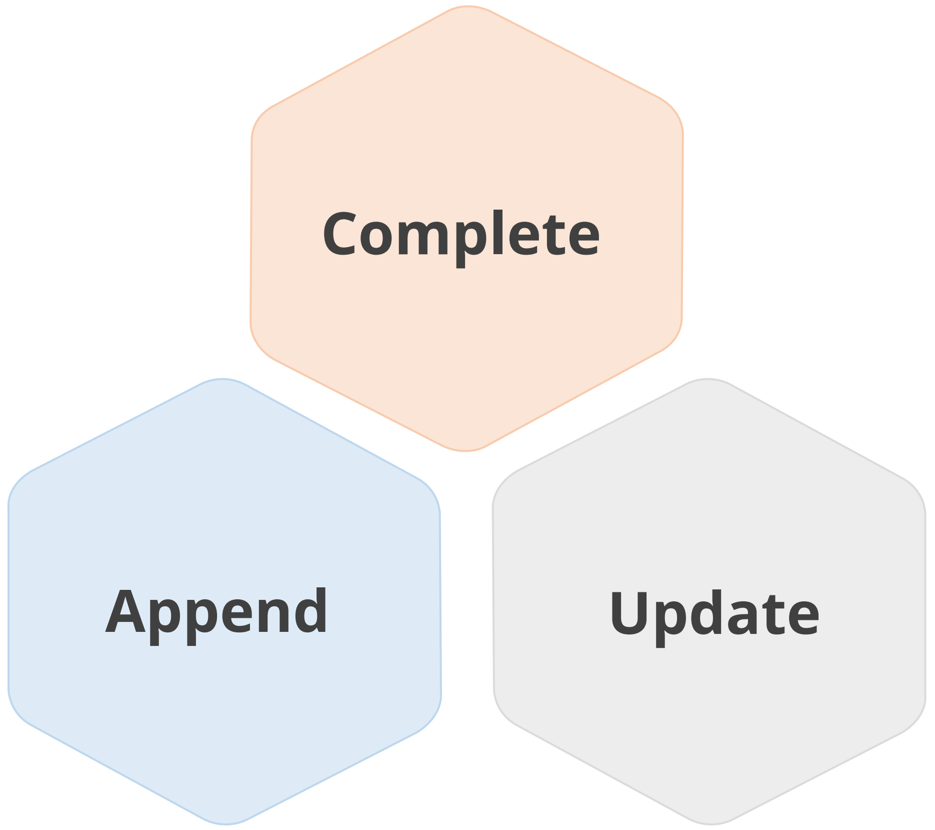
Socket Source

It reads data from a socket connection. As it does not support fault-tolerance assurances, it is mostly used for testing.

## Output Modes

The output mode defines what is written to the external storage device. The different output modes are defined below:

Entire updated result table is written to the disk

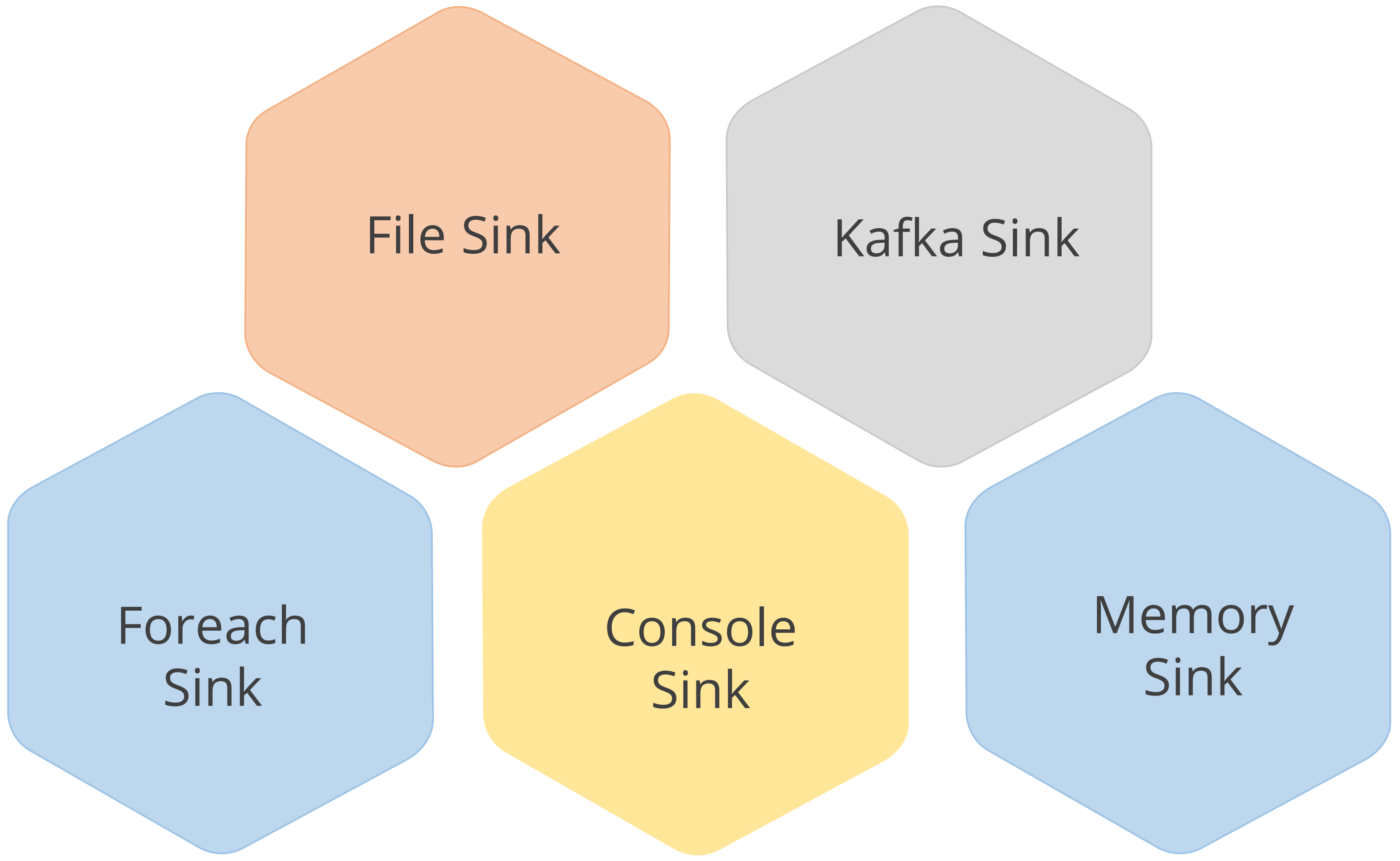


Only new rows are added, since the last trigger will be written to the external storage

Only updated rows in the result table are written, since the last trigger will be written to the external storage

## Output Sinks

Sinks store the output to external systems once all the computations on the data have been completed. They use DataStreamWriter via writeStream. Spark Structured streaming provides several built-in output sinks.



## Output Sinks

Spark Structured streaming provides several built-in output sinks.

**File sink:** Store the output into a directory

Syntax:

```
df.writeStream
   .format("parquet") // can be "orc", "json", "csv", etc.
   .option("path", "path/to/destination/dir")
   .start()
```

**Foreach sink:** Run arbitrary computation on the records in the output

Syntax:

```
df.writeStream
.foreach(...)
.start()
```

## Output Sinks

Spark Structured streaming provides several built-in output sinks.

**Console sink:** For debugging

Syntax:

```
df.writeStream
  .format("console")
  .start()
```

**Memory sink:** For debugging

Syntax:

```
 df.writeStream
  .format("memory")
  .queryName("tableName")
  .start()
```

## Output Sinks

Spark Structured streaming provides several built-in output sinks.

**Kafka sink:** Stores the output to one or more topics in Kafka

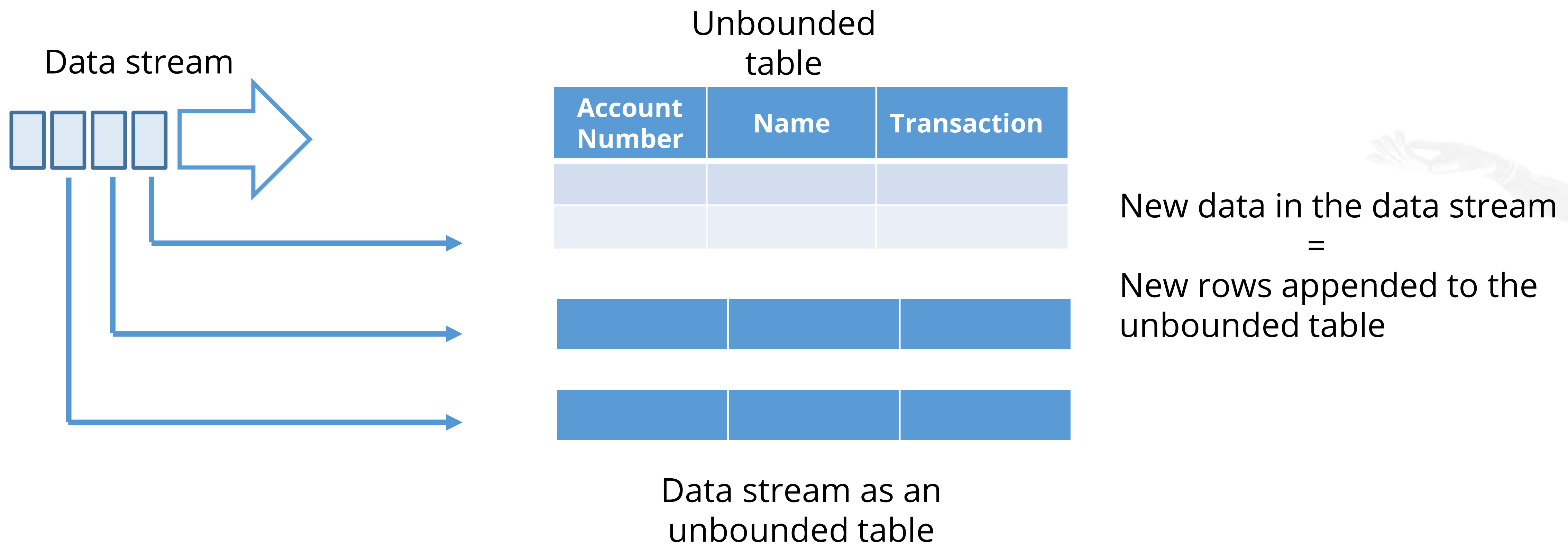
Syntax:

```
writeStream
     .format("kafka")
     .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
     .option("topic", "updates")
     .start()
```

## Use Case: Banking Transactions

## Use Case: Banking Transactions

Consider a scenario in which a continuous stream of banking transaction records with the account number and the transaction amount is arriving. Every second, a query on the input creates the *Result Table,* and new rows are added to the input table, which finally changes the result table. When the result table is updated, the changed result rows are written to an external sink.


Data stream
Unbounded table
Account Number
Name
Transaction
New data in the data stream
=
New rows appended to the unbounded table
Data stream as an unbounded table

## Structured Streaming API

## Structured Streaming API

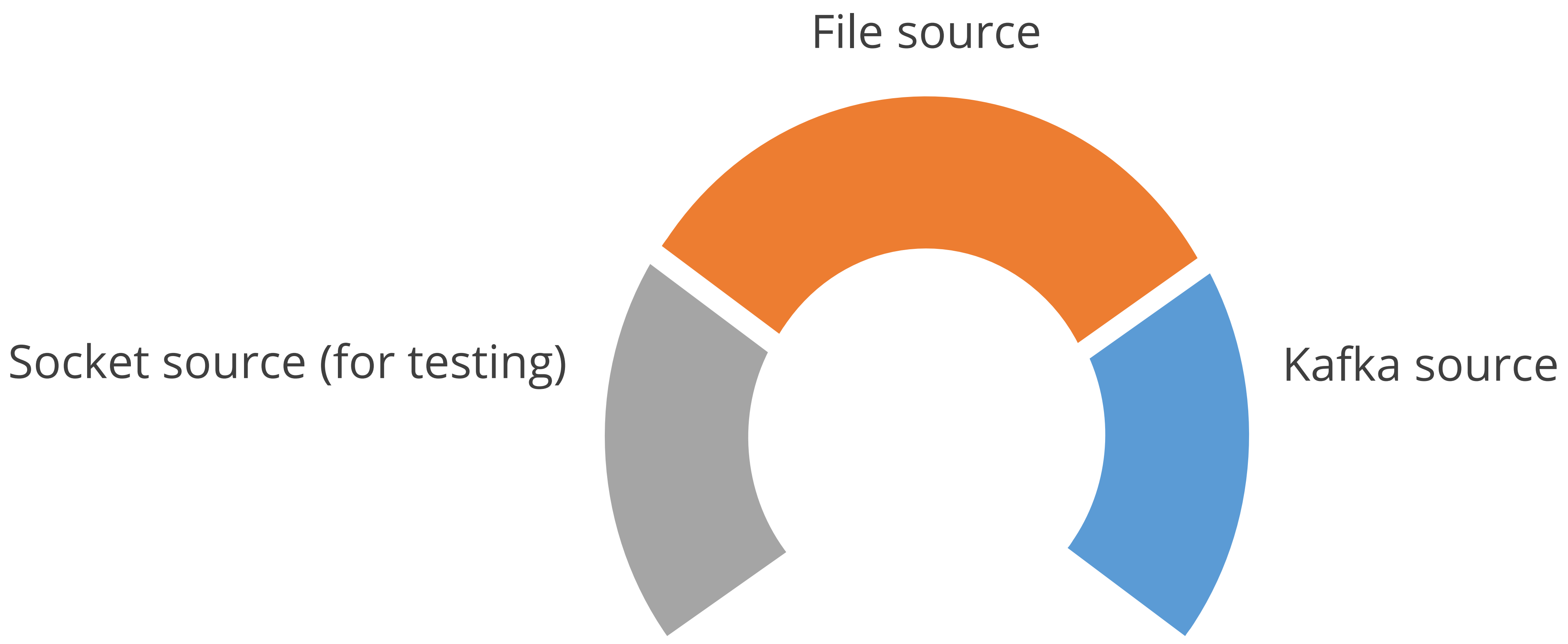
Structured streaming offers a high-level declarative streaming API built on top of datasets and DataFrames.

Syntax of reading data from a socket:

```
socketDF = spark \
  .readStream \  // Returns a DataStreamReader
  .format(“socket") \  // Reading data from a socket
  .option("host", "localhost") \
  .option("port", 9999) \
  .load()
```

## Data Sources

There are a few built-in sources in Spark 2.0 and the later versions.



## Reading Data from JSON File

The below syntax is used to read data from a JSON file.

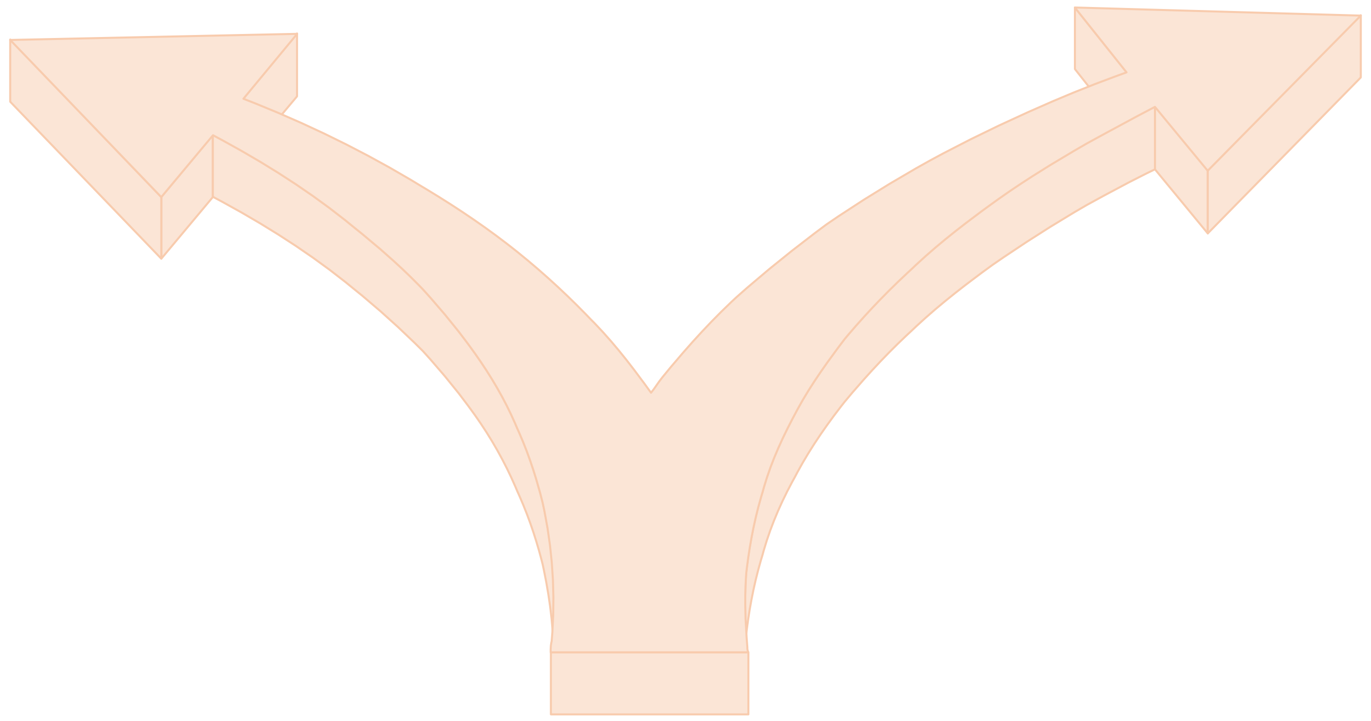
Syntax:

```
inputDF = spark.read.json("/user/simplilearnuser/data/people.json")
```

## Operations on Streaming DataFrames and Datasets

With the Structured Streaming API, users can perform all kinds of operations on streaming DataFrames and datasets, ranging from untyped SQL-like operations to typed RDD-like operations.

**Untyped SQL-like operations**
(e.g., select, where, and groupBy)



**Typed RDD-like operations**
(e.g., map,filter, and flatMap)

## Untyped SQL-like Operations: Example

The untyped SQL-like operations is explained as below:

Code Snippet:

```
# Select the people who are more than 60 years
df1.select(“First Name”).where(F.col(“Age”) > 60) // using untyped API

# Running count of the number for each value
df1.groupBy(“Age").count()         // using untyped API
```

## groupby and Aggregate

Code Snippet:

```
empStream.groupBy(F.col("country")) \
     .mean(F.col("age")) \
     .writeStream \
     .outputMode("complete") \
     .format("console") \
     .start()
```

- Spark Structured streaming uses a groupby operator that allows to group the rows based on a column value.
- Once the groupBy operation is performed, an aggregate function can be used to aggregate multiple rows of data into a single output.
- Spark can perform different aggregation operations on the same input group.

## Parsing Data with Schema Inference

The below example is used to read a CSV file using schema inference.

Example:

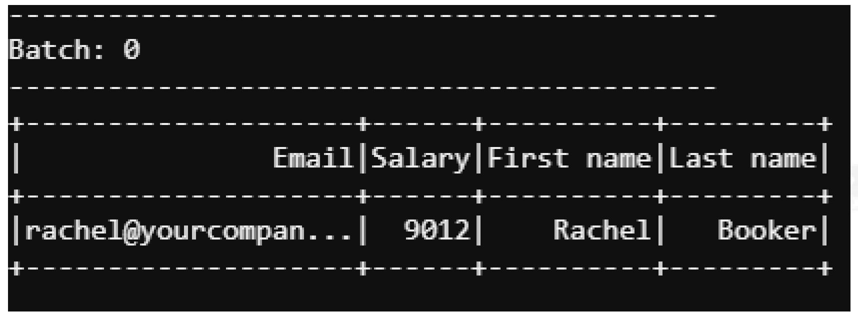
```
# Import required packages
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType
from pyspark.sql import functions as F

# Create SparkSession
spark = SparkSession\
    .builder\
    .appName("Demo")\
    .getOrCreate()

# Define Schema
userSchema = StructType()\
               .add("name", "string")\
               .add("salary", "integer")

# Read CSV from provided path
df = spark.readStream
        .option(“sep”, “;”)
        .option(“header”, “false”)
        .schema(userSchema)
        .csv(“/path/to/CSV”) //HDFS path to csv file
```

## Constructing Columns in Structured Streaming

Structured streaming uses column objects for manipulating data. A column can also be constructed from other columns using a binary operator.

Code Snippet:

```
df.filter(F.col("First name") == "Rachel" ) \
    .writeStream.queryName("counts2") \
    .outputMode("append") \
    .format("console") \
    .start().awaitTermination()
```



```
-------------------------------------------
Batch: 0
-------------------------------------------
+--------------------+------+----------+---------+
|               Email|Salary|First name|Last name|
+--------------------+------+----------+---------+
|rachel@yourcompan...|  9012|    Rachel|   Booker|
+--------------------+------+----------+---------+
```

## Complex Types: Aggregations

For complex aggregations, the *agg* function is used.

Code Snippet:

```
tempStream.groupBy(F.col(“country”)) \
         .agg(F.col(“country“),count(F.col(“age"))) \
         .writeStream \
         .outputMode("complete") \
         .format("console") \
         .start()
```

## Joining Datasets with Structured Streams

Streaming DataFrame can be joined with static DataFrame to create a new streaming DataFrame.

Syntax:

```
# Create a static Dataframe
staticDf = spark.read. ...

# Streaming Dataframe
streamingDf = Spark.readStream. ...

# inner equi-join with a static DF
streamingDf.join(staticDf, "type")

# right outer join with a static DF
streamingDf.join(staticDf, "type", "right_join")
```

## SQL Query in Spark Structured Streaming

SQL queries can be directly written on stream after registering them as a temp view. Below are the steps required to perform the activity.

Step 1: Create a temporary Table

```
df1.createOrReplaceTempView(“empTable")
```

Step 2: Write a SQL query on created temp table

```
query = spark.sql(“select country,avg(age) from empTable group by country")
```

Step 3: Write the results to the console

```
query.writeStream.outputMode(“complete").format("console").start()
```

## Windowed Operations on Event-Time

Windowed operations are running aggregations over data bucketed by time windows.

Code Snippet:

```
# Split the lines into words
words = lines.select(
    F.explode(
        F.split(lines.value, " ")
    ).alias("word")
)
```

# Windowed Operations on Event-Time

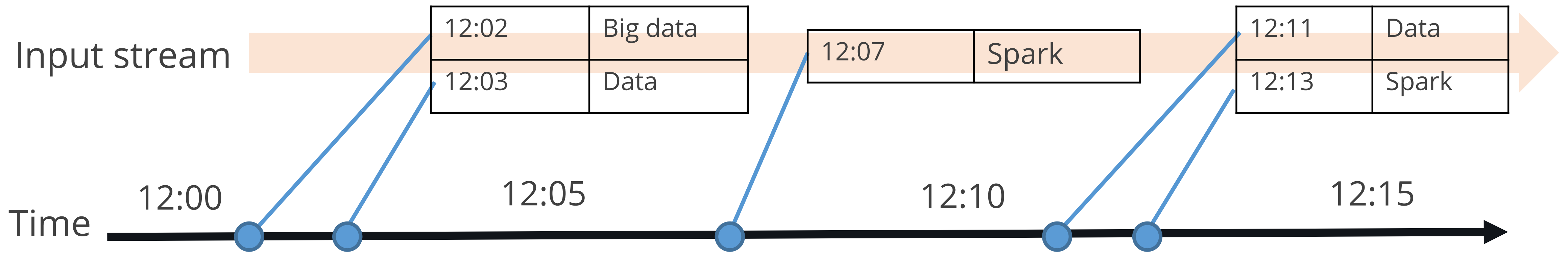
Windowed operations are running aggregations over data bucketed by time windows.

Code Snippet:

```
# Group the data by window and word and compute the count of each group
windowedCounts = words
 .groupBy(window(F.col("timestamp“),windowDuration,
slideDuration),F.col(”word")) \
 .count() \
 .orderBy("window")
```

# Windowed Grouped Aggregations

The window function can be used with Structured streaming to express windows on the event time to perform grouping.



Resulting tables after 5 minutes of triggers

Windowed Grouped Aggregation with 10 minutes windows, sliding every 5 minutes

| | | |
|---|---|---|
| 12:00-12:10 | Big | 1 |
| 12:00-12:10 | Data | 3 |

| | | |
|---|---|---|
| 12:00-12:10 | Big | 2 |
| 12:00-12:10 | Data | 3 |
| 12:00-12:10 | Spark | 1 |
| 12:05-12:15 | Spark | 1 |
| 12:05-12:15 | Big | 1 |

Counts incremented for windows 12:00 -12:10 and 12:05 -12:15

| | | |
|---|---|---|
| 12:00-12:10 | Big | 2 |
| 12:00-12:10 | Data | 3 |
| 12:00-12:10 | Spark | 1 |
| 12:05-12:15 | Big | 1 |
| 12:05-12:15 | Spark | 2 |
| 12:05-12:15 | Data | 1 |
| 12:05-12:20 | Data | 1 |
| 12:05-12:20 | Spark | 1 |

Counts incremented for windows 12:05 -12:15 and 12:10 -12:1220

## Failure Recovery and Checkpointing

- Structured streaming allows recovery from failures, which is achieved by using checkpointing and WAL.
- A query can be configured to save all progress information and run aggregates for a checkpoint location.
- When launching a query, the checkpoint location should be a path in an HDFS compatible file system that may be set as an option in the DataStreamWriter.

Code Snippet:

```
callsFromParis \
     .writeStream \
     .format(“parquet") \
     .option(“checkpointlocation","hdfs://nn:8020/mycheckloc") \
     .start("/home/Spark/streaming/output")
```

## When to Use Structured Streaming?

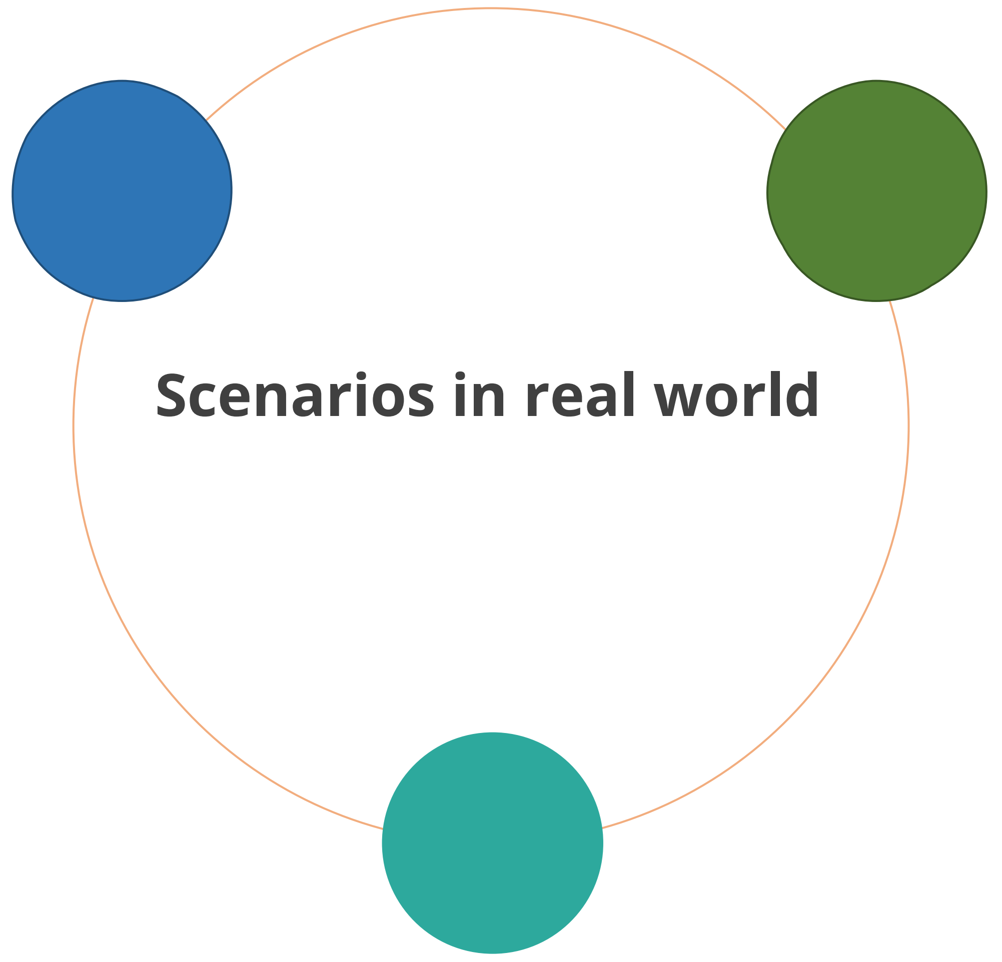
1. To create a streaming application

2. To provide data consistency semantics that occurs exactly once

3. To create continuous applications that are integrated with batch queries, streaming, and machine learning

## Usecase: Spark Structured Streaming

## Usecase: Spark Structured Streaming



Analyze streams of real-time sensor data

Analyze streams of web log data

Transform unstructured data into a format that can be easily and efficiently queried

## Assisted Practice: Working with Spark Structured Application

**Duration**: 15 mins

**Problem Scenario:** Create a Spark Structured streaming application to work with real-time data

**Objective:** The objective is to create a real-time Spark Structured streaming application.

**Tasks to Perform:**

Step 1: Open the **"Webconsole"** and start the **“netcat“** with any port number

Step 2: Login to the PySpark shell in another **“Webconsole”**

Step 3: Create a **“DataStreamReader”** by importing the necessary packages

Step 4: Write a program to split each line with space and execute the code

Step 5: Calculate the network word count using the groupby and count functions of real-time streaming data

**Note: The solution to this assisted practice is provided under the course resources section**.

## Key Takeaways

- Spark Structured streaming is a high-level streaming API that is built on the Spark SQL engine.
- Spark Structured streaming works in both micro-batch model and continuous processing model.
- Spark supports a variety of input sources for data ingestion.
- Structured streaming offers a high-level declarative streaming API built on top of datasets and DataFrames.
- Spark can perform different aggregation operations on the same input group.

## Knowledge Check

Knowledge Check

1

## Which of the following data sources are supported in Spark streaming?

A. Twitter

B. Kafka

C. Flume

D. All of the above

**Knowledge Check 1**

## Which of the following data sources are supported in Spark streaming?

A. Twitter

B. Kafka

C. Flume

D. All of the above

The correct answer is **D.**

Spark Streaming supports Twitter, Kafka, and Flume data sources.

Knowledge Check

2

## Which of the following statements is correct about the saveAsObjectFile method?

A. Saves a DStream's contents as a SequenceFile of serialized Java objects

B. Saves a DStream's contents as text files

C. Saves a DStream's contents as an Avro file in Hadoop

D. Applies a function to each RDD generated from the stream

Knowledge Check

2

## Which of the following statements is correct about the saveAsObjectFile method?

A. Saves a DStream's contents as a SequenceFile of serialized Java objects

B. Saves a DStream's contents as text files

C. Saves a DStream's contents as an Avro file in Hadoop

D. Applies a function to each RDD generated from the stream

The correct answer is **A.**

The saveAsObjectFile method saves a DStream's contents as a SequenceFile of serialized Java objects.

Knowledge Check 3

## When should Structured streaming be used?

A. To create a streaming application using Dataset and DataFrame APIs

B. When providing data consistencies and exactly-once semantics even in case of delays and failures at multiple levels

C. For creating continuous applications that are integrated with batch queries, streaming, and machine learning

D. All of the above

Knowledge Check

3

## When should Structured streaming be used?

A. To create a streaming application using Dataset and DataFrame APIs

B. When providing data consistencies and exactly-once semantics even in case of delays and failures at multiple levels

C. For creating continuous applications that are integrated with batch queries, streaming, and machine learning

D. All of the above

The correct answer is **D.**

Structured Streaming can be used for all the cases.

## Lesson-End Project: Retail Business Analysis Using Structured Streaming

**Problem Scenario:**
An organization is collecting data from IoT (Internet of Things) devices to perform analysis. The data contains information about the action performed by a user on an IoT device like “power off” and “power on”.

**The structure of the data is shown below:**

**time, String**
**customer, String**
**action, String**
**device, String**.

**Objective:** The objective is to analyze the data that is coming from the IoT devices to perform a data analysis using structured streaming.

# Lesson-End Project: Retail Business Analysis Using Structured Streaming

**Tasks to Perform:**

1. Download the folder “data“ from the course resources section to perform analytics
2. Create a directory in HDFS named “data-files” and upload the dataset into the directory
3. Open the PySpark shell in the Webconsole
4. Define a proper schema containing four fields:
   time
   customer
   action
   device
5. Read the input JSON data as a DataFrame and the schema created
6. Write the streaming DataFrame created in the previous step (5) to memory in append output mode

## Thank You