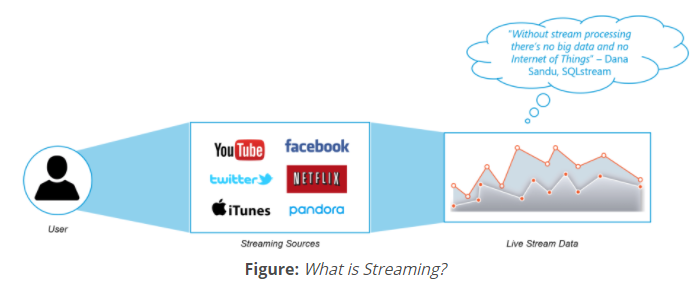
**What is Streaming?**

Data Streaming is a technique for transferring data so that it can be processed as a steady and continuous stream. Streaming technologies are becoming increasingly important with the growth of the Internet.

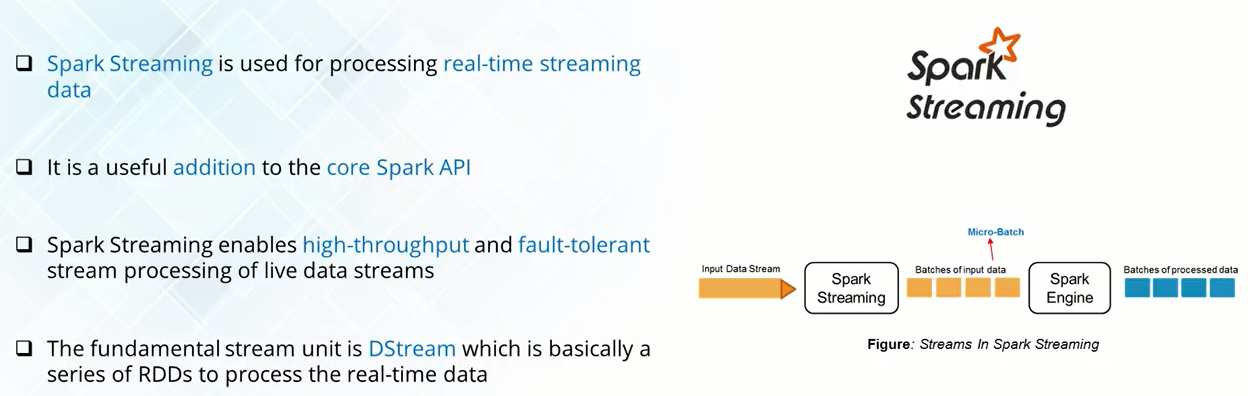


**Spark Streaming:**

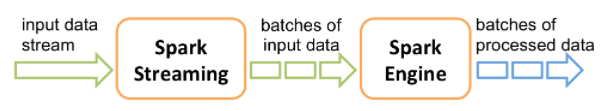
* Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
* Spark Streaming can be used to stream live data and processing can happen in real time.
* Data can be ingested from many sources like Twitter, Stock Market, Geographical Systems, Kafka, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window.
* Finally, processed data can be pushed out to file systems, databases, and live dashboards.
* Spark Streaming’s ever-growing user base consists of household names like Uber, Netflix and Pinterest.

****

**Spark Streaming Overview:**

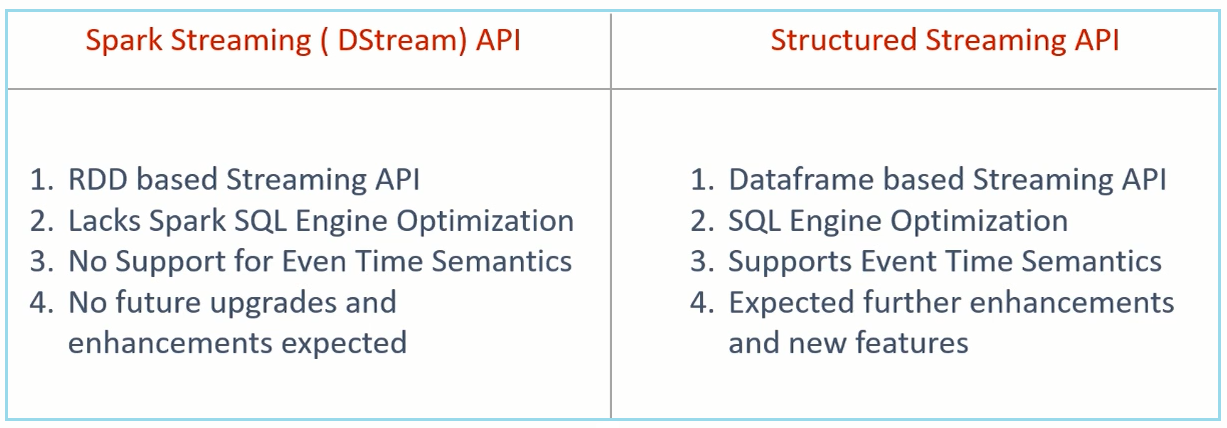


Internally, it works as follows. Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

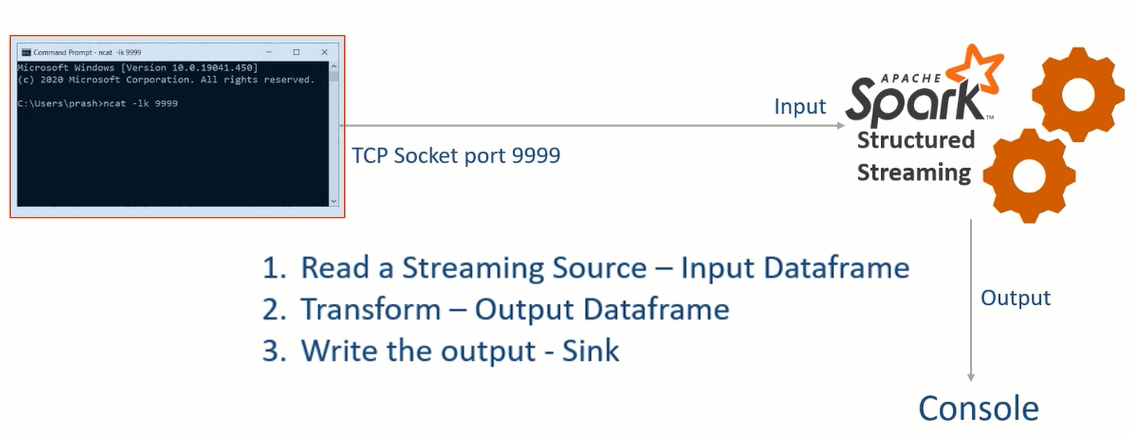


Spark Streaming provides a high-level abstraction called ***discretized stream*** or ***DStream***, which represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka and Kinesis or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of [RDDs](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/rdd/RDD.html).

**DStreams VS Dataframe API:**



**Streaming Word Count:**



from pyspark.sql import SparkSession

from pyspark.sql.functions import split, explode

if \_\_name\_\_ == "\_\_main\_\_":

spark = SparkSession.builder \

.appName("Streaming Word Count") \

.master("local[3]") \

.config("spark.streaming.stopGracefullyOnShutdown", "true") \

.config("spark.sql.shuffle.partitions", 3) \

.getOrCreate()

lines\_df = spark.readStream \

.format("socket") \

.option("host", "localhost") \

.option("port", "9999") \

.load()

#lines\_df.printSchema()

words\_df = lines\_df.select(explode(split("value", " ")).alias("word"))

#words\_df = lines\_df.select(expr("explode(split(value,' ')) as word"))

counts\_df = words\_df.groupBy("word").count()

word\_count\_query = counts\_df.writeStream \

.format("console") \

.outputMode("complete") \

.option("checkpointLocation", "/home/saif/Desktop/checkpoint/dir1") \

.start()

word\_count\_query.awaitTermination()

**Stateless Vs Stateful Transformations:**

**1) Stateless**means that the logic of handling the new data is independent of the previous data.

**E.g.** select ( ), filter ( ), map ( ), flatMap ( ), explode ( ) etc.

**Drawback:** Stateless transformation will not support **COMPLETE** output mode.



**2) Stateful** in contrast to stateless, it means that you need somehow combine the data with old records or previous batches.

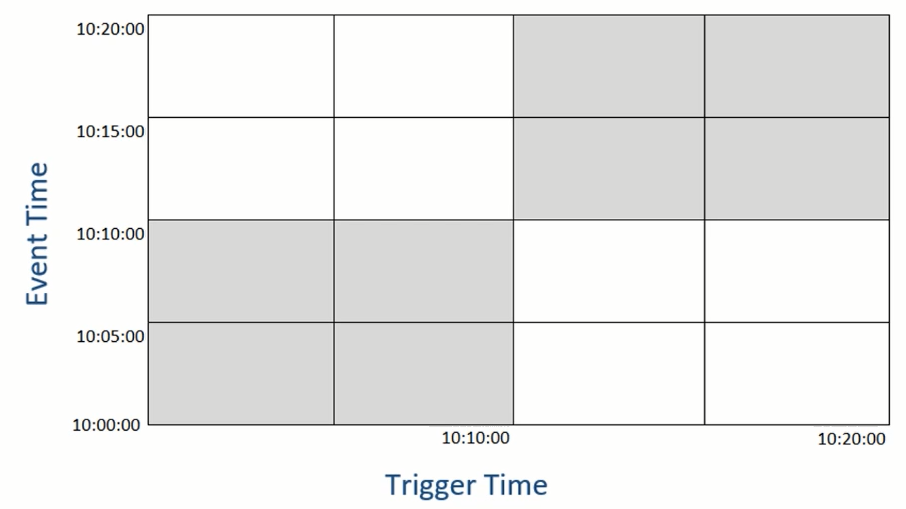
**E.g.** Grouping, Aggregations, Windowing & Joins.

**Drawbacks:** Excessive state causes out of memory

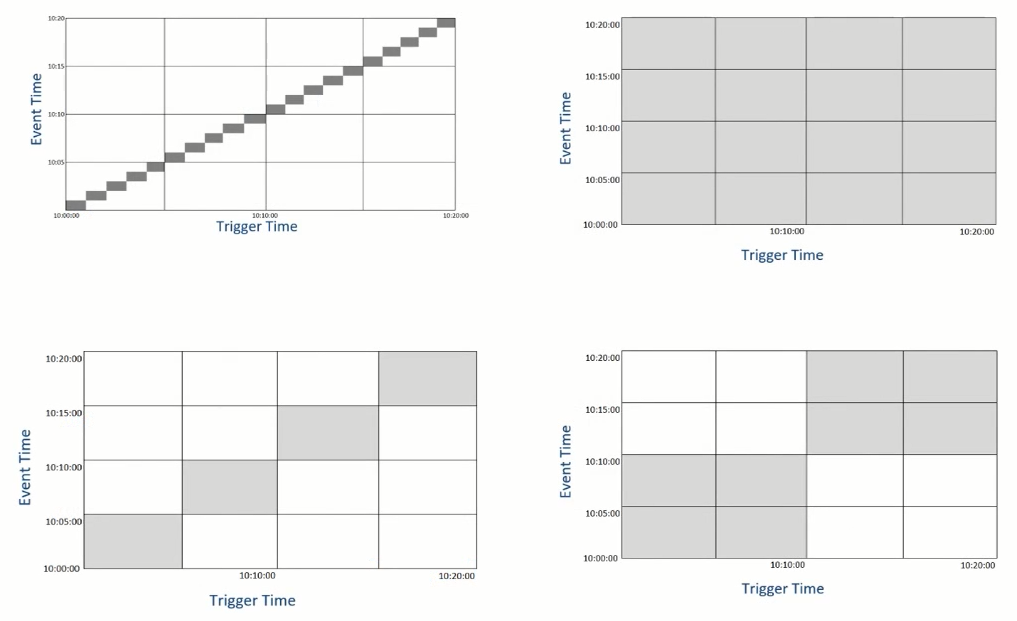
**Window Aggregates:**

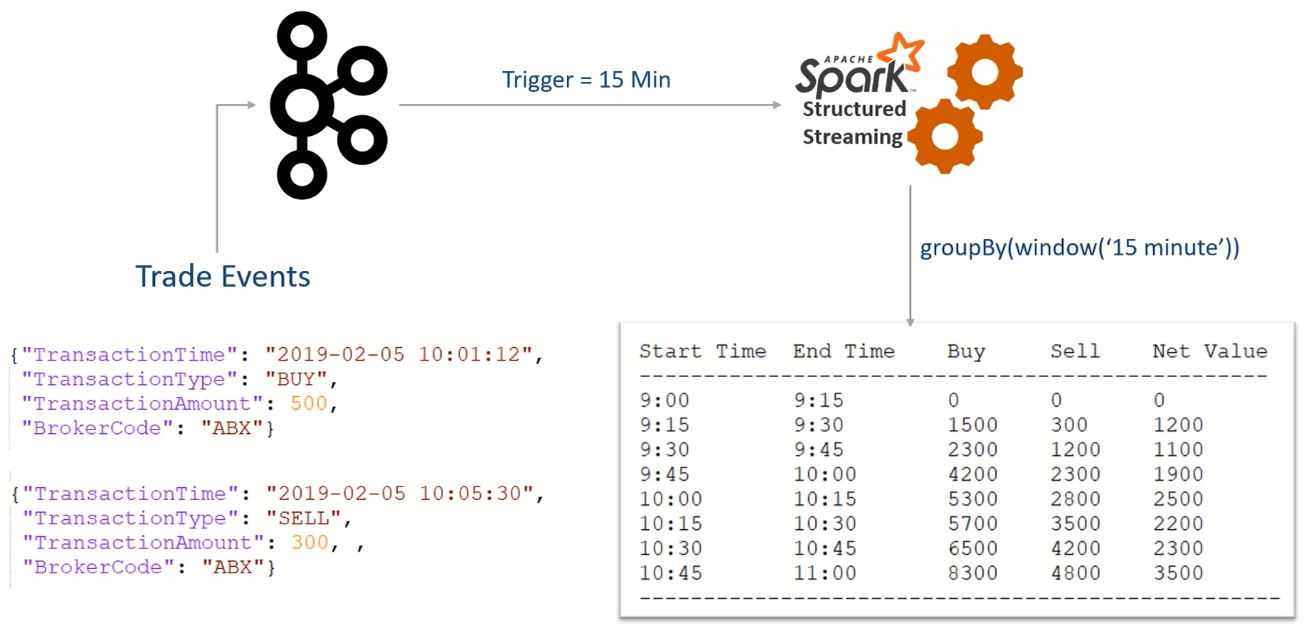
1) Tumbling Time Window

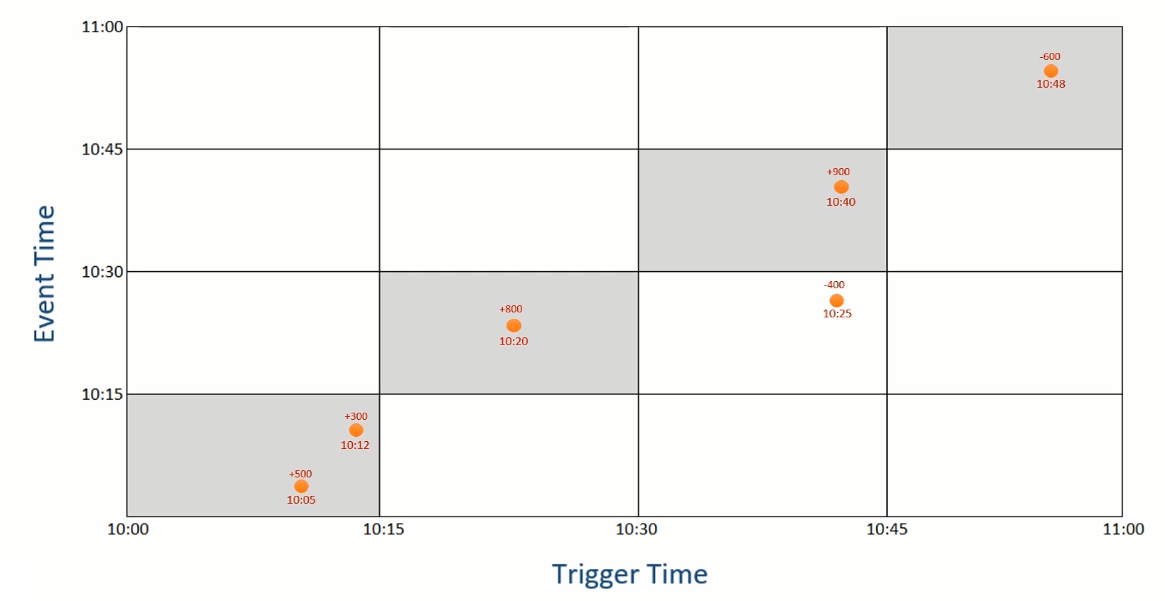
2) Sliding Time Window



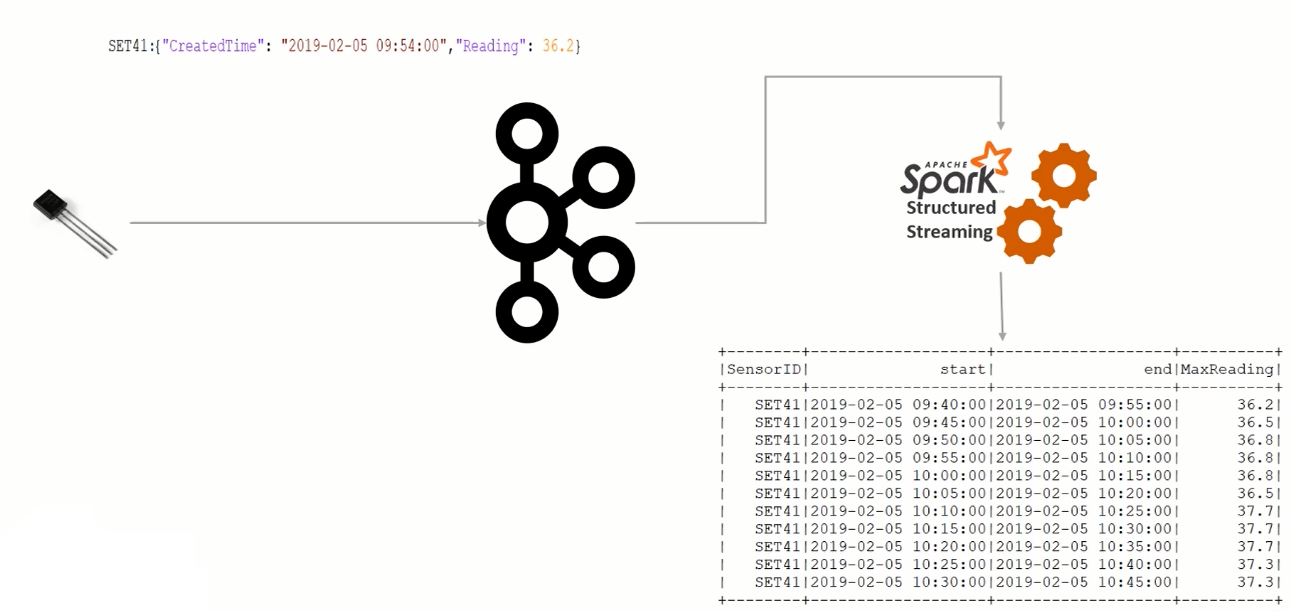


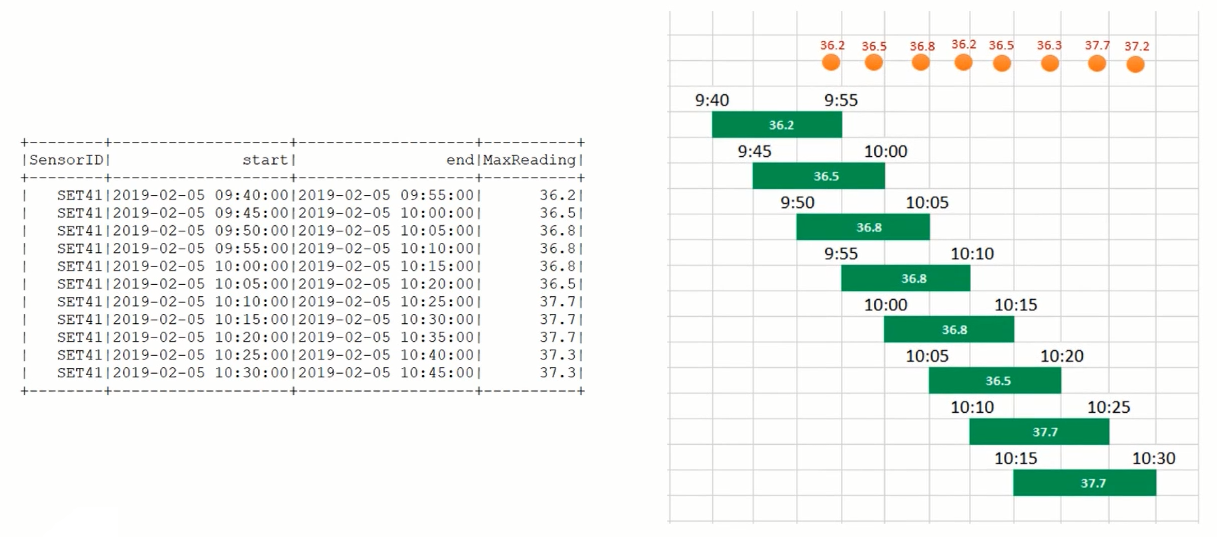






**Sliding Window also called Hopping Window i.e. overlapping:**





**Sliding Window:**

**a) Start the Zookeeper:**

cd $ZOOKEEPER\_HOME

bin/zkServer.sh start

**a) Start the Kafka:**

cd $KAFKA\_HOME

bin/kafka-server-start.sh config/server.properties

bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic sensor --partitions 1 --replication-factor 1

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic sensor

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic sensor --property "parse.key=true" --property "key.separator=:"

from pyspark.sql import SparkSession

from pyspark.sql.functions import from\_json, col, to\_timestamp, window, max

from pyspark.sql.types import StructType, StructField, StringType, DoubleType

if \_\_name\_\_ == "\_\_main\_\_":

spark = SparkSession \

.builder \

.master("local[3]") \

.appName("Sliding Window Demo") \

.config("spark.streaming.stopGracefullyOnShutdown", "true") \

.config("spark.sql.shuffle.partitions", 1) \

.getOrCreate()

schema = StructType([

StructField("CreatedTime", StringType()),

StructField("Reading", DoubleType())

])

kafka\_source\_df = spark \

.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "sensor") \

.option("startingOffsets", "earliest") \

.load()

value\_df = kafka\_source\_df.select(col("key").cast("string").alias("SensorID"),

from\_json(col("value").cast("string"), schema).alias("value"))

sensor\_df = value\_df.select("SensorID", "value.\*") \

.withColumn("CreatedTime", to\_timestamp(col("CreatedTime"), "yyyy-MM-dd HH:mm:ss"))

agg\_df = sensor\_df \

.withWatermark("CreatedTime", "30 minute") \

.groupBy(col("SensorID"),

window(col("CreatedTime"), "15 minute", "5 minute")) \

.agg(max("Reading").alias("MaxReading"))

output\_df = agg\_df.select("SensorID", "window.start", "window.end", "MaxReading")

window\_query = output\_df.writeStream \

.format("console") \

.outputMode("update") \

.option("checkpointLocation", "file:///home/saif/Desktop/checkpoint/dir1") \

.trigger(processingTime="1 minute") \

.start()

window\_query.awaitTermination()