Big Data Hadoop and Spark Developer

DATA AND
ARTIFICIAL INTELLIGENCE

Stream Processing Frameworks and Spark Streaming

Learning Objectives

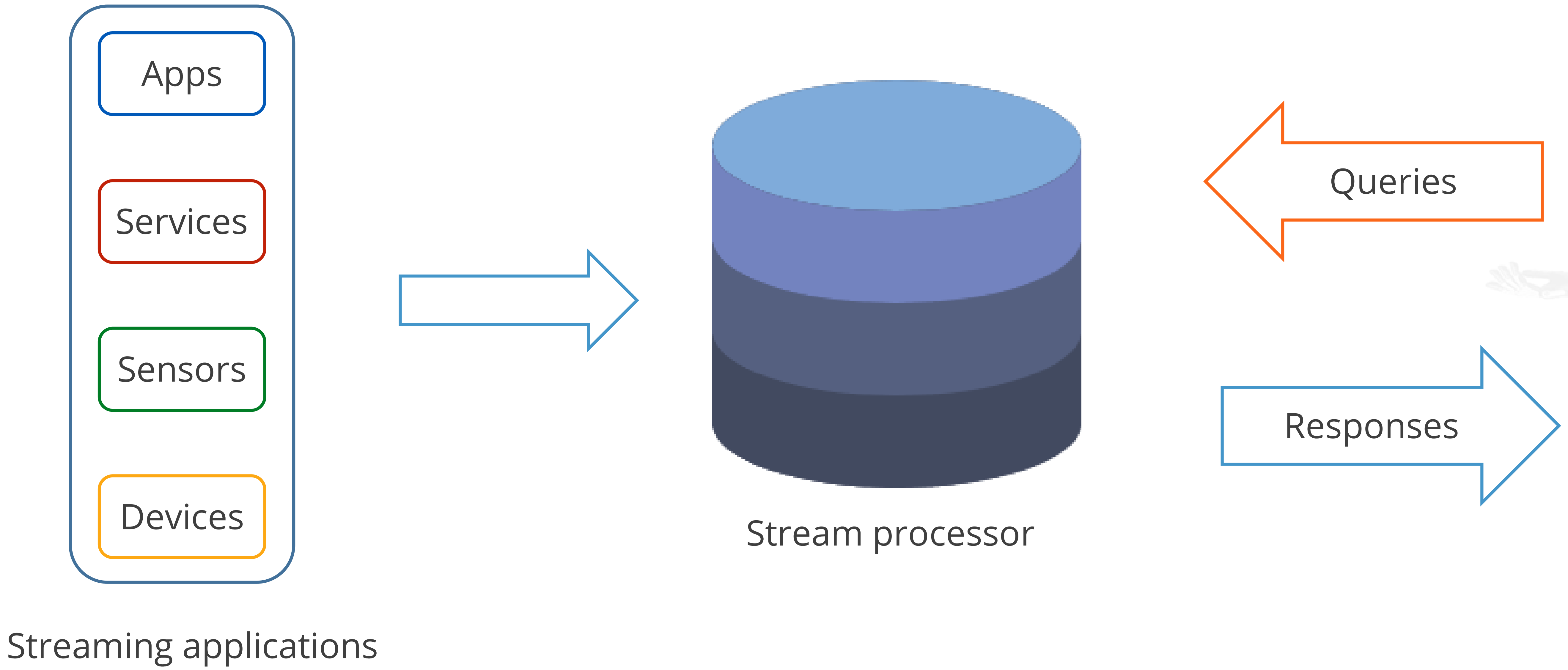
By the end of this lesson, you will be able to:

- Explain concepts of Spark Streaming
- Understand the Lambda and Kappa architecture
- Explain the concepts of Spark Structured Streaming
- Explain Join and Window operations

Streaming Overview

What Is Streaming?

Big data streaming involves processing continuous streams of data in order to extract Real-time insights.



Need for Real-Time Processing

Certain tasks require big data processing as quickly as possible. For example:

- Delays in tsunami prediction can cost people’s lives
- Delays in traffic jam prediction cost extra time
- Advertisements can lose popularity, if not correctly targeted

NoSQL Popularity

NoSQL databases are commonly used to solve challenges posed by stream processing techniques.

Minimal Latency

Unstructured Data

High Storage

Scalability

Real-Time Processing of Big Data

Real-time Processing of Big Data

Real-time processing consists of continuous input, processing, and analysis of reporting data.

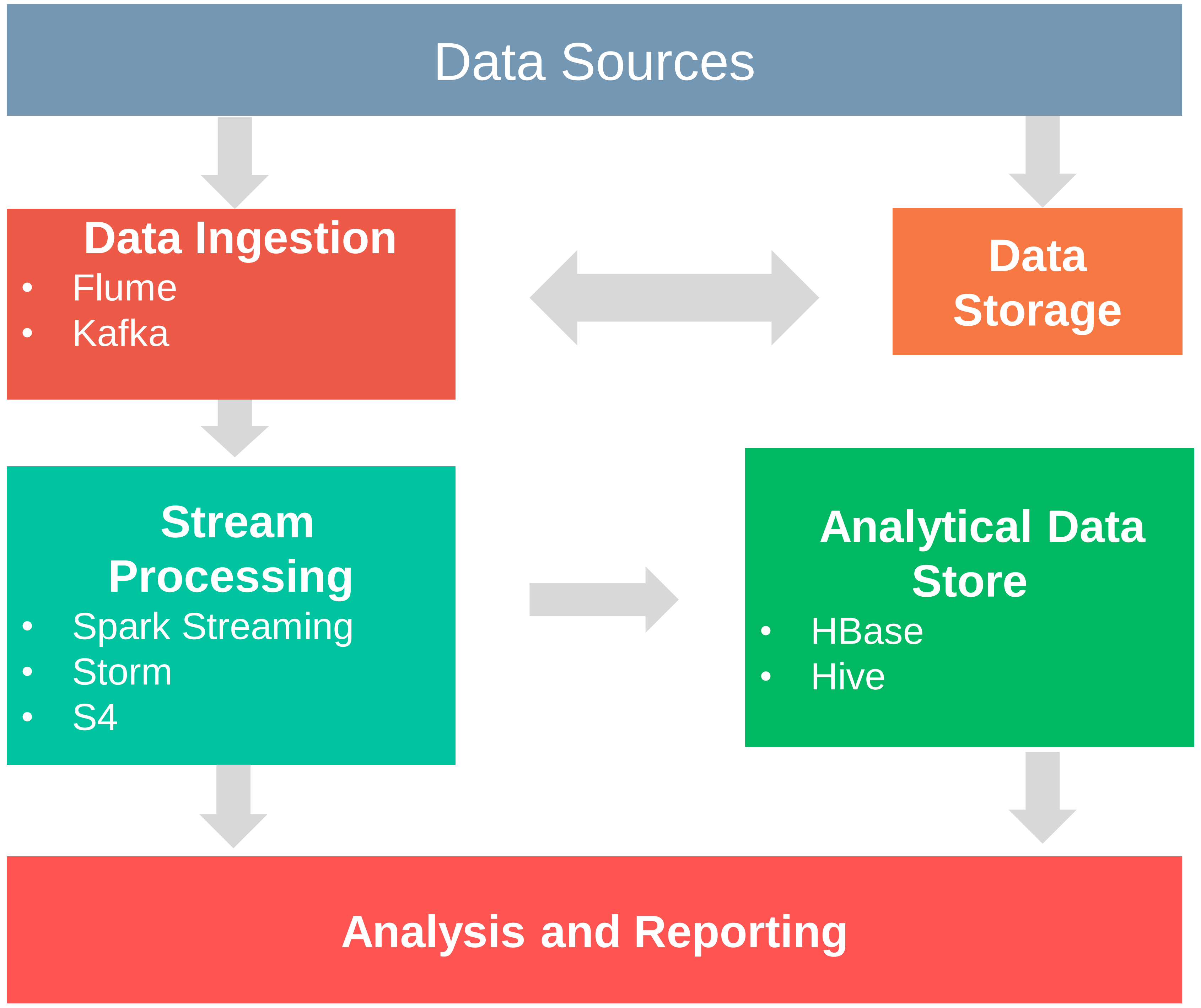
01 The process consists of a sequence of repeated operations, in which the data streams are transferred to the memory.

02 Real-time processing is crucial in order to continue the high-level functionality of automated systems having intensive data streams and different data structures.

03 For example: Bank ATMs, radar systems, disaster management systems, internet of things, and social networks.

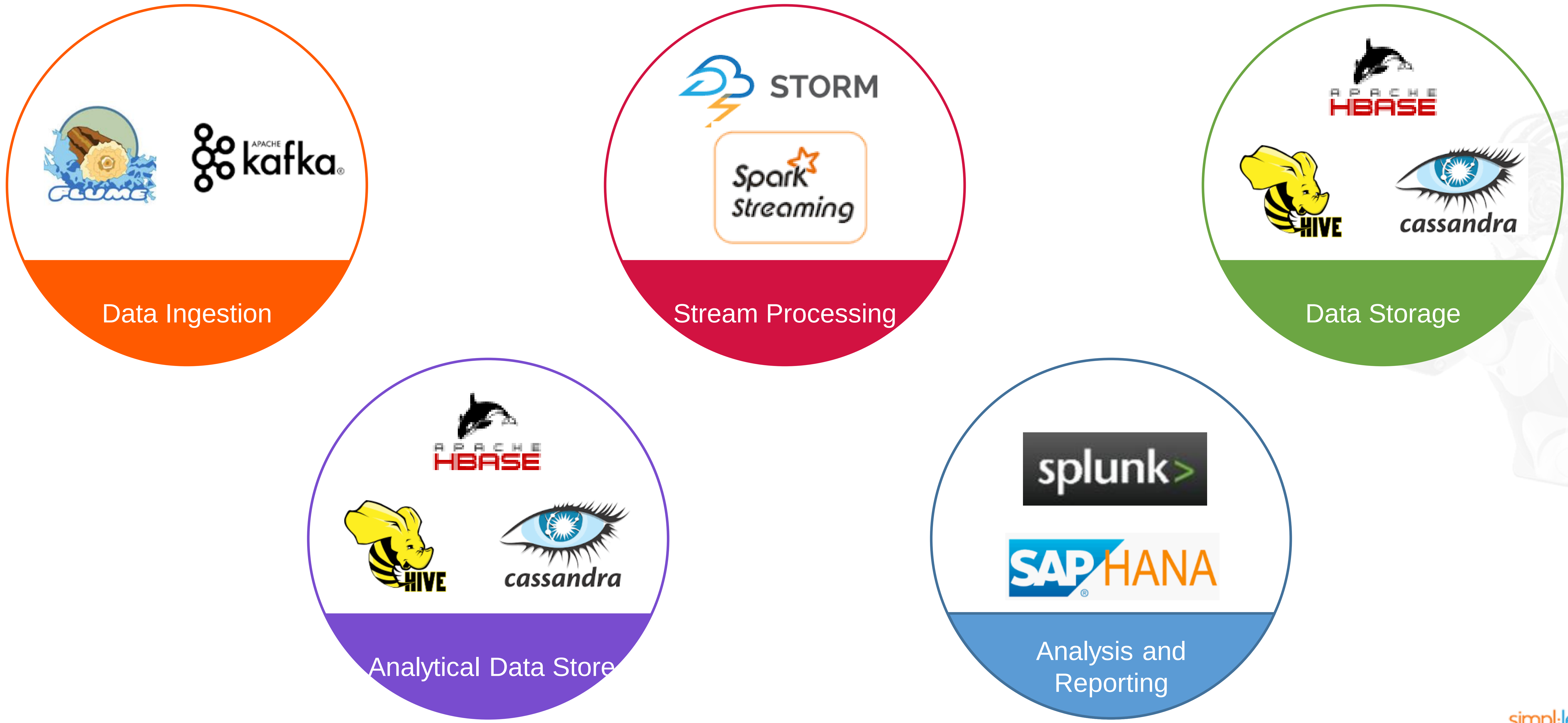
Real-Time Big Data Processing Lifecycle

The below diagram shows the lifecycle of real-time big data processing.



Real-Time Big Data Processing Tools

Below are some of the popular tools for Real-time Big data processing.



Data Processing Architectures

Data Processing Architecture

A good architecture for Real-time processing should have the following properties.

1

Fault-tolerant and scalable

2

Supportive of batch and incremental updates

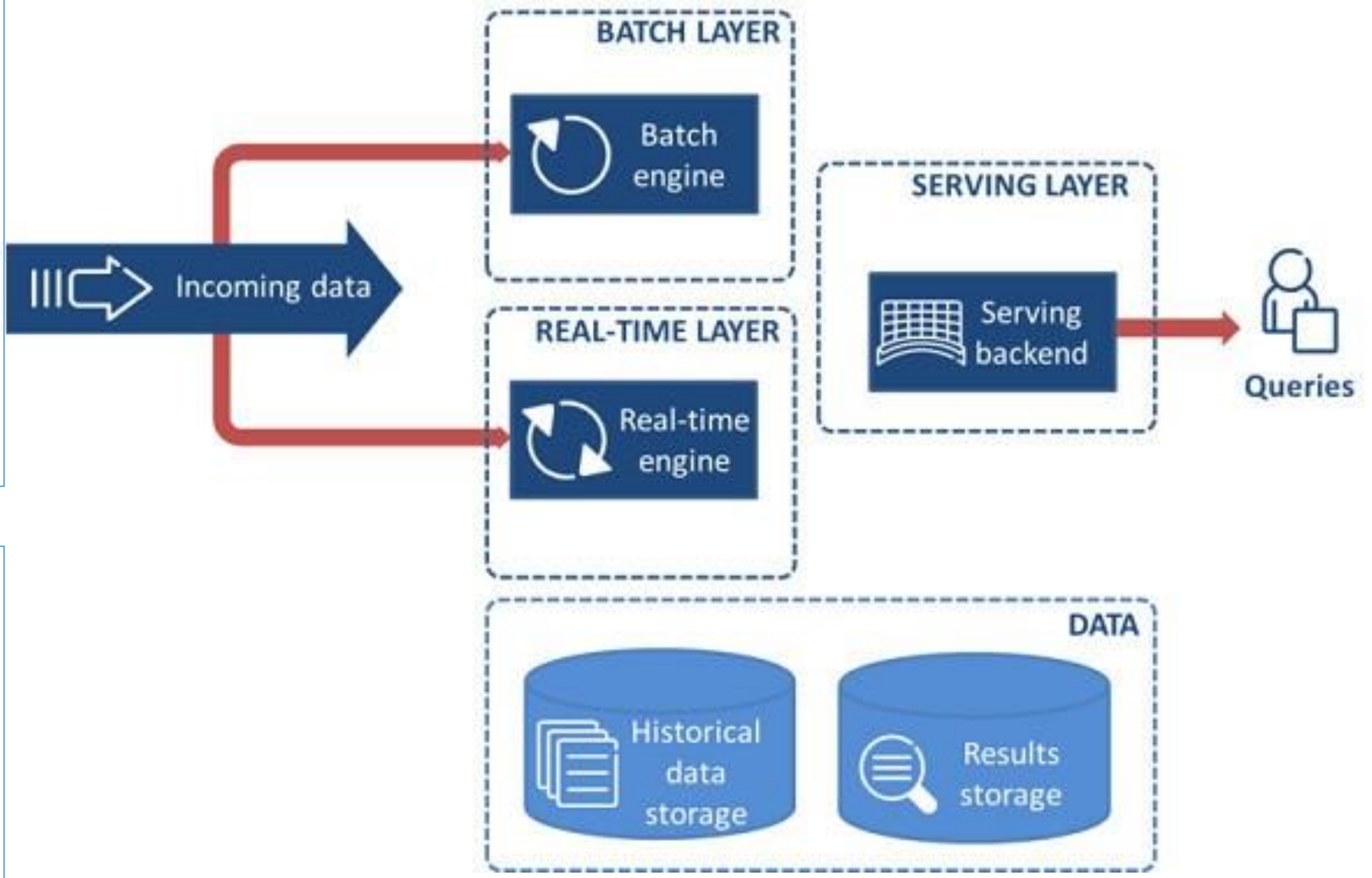
3

Extensible

The Lambda Architecture

The Lambda Architecture is composed of three layers: Batch, Real-Time, and Serving

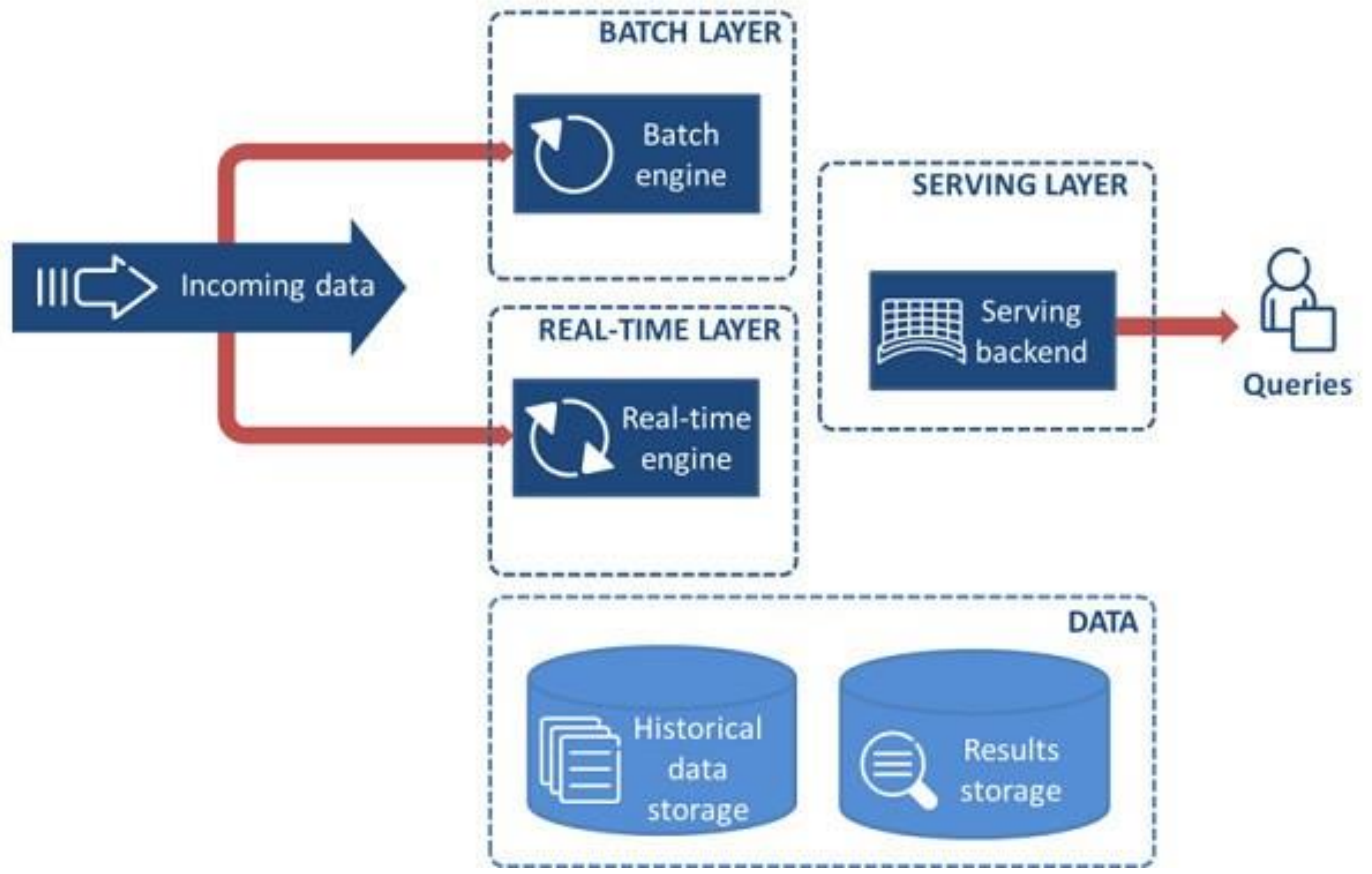
Batch Layer
- Stores the raw data as it arrives
- Computes the batch views for consumption
- Manages historical data
- Re-computes results such as machine learning models
- Operates on full data
- Produces most accurate results
- Has a high cost of high latency due to high computation time



Real-Time Layer
- Receives the arriving data
- Performs incremental updates to the batch layer results
- Has incremental algorithms implemented at the speed layer
- Has a significantly reduced computation cost

The Kappa Architecture

The Kappa Architecture only processes data as a stream.



Use Case

Case Scenario: Twitter wanted to improve mobile experience for its users.

Problem: A complex system that receives events, archives them, performs offline and real-time computations, and merges the results of those computations into coherent information

Goal: To reduce impact on battery and network usage; ensuring data reliability and getting the data over as close to real time as possible

Solution: To reduce impact on the device, analytics events are compressed and sent in batches.

Process: Twitter utilized the Lambda architecture to achieve this. The architecture consists of four major components: event reception, event archival, speed computation, and batch computation.

Result: This has helped provide app developers with reliable, real-time and actionable insights into their mobile applications.

Assisted Practice

Real-Time Data Processing

Duration: 15 mins

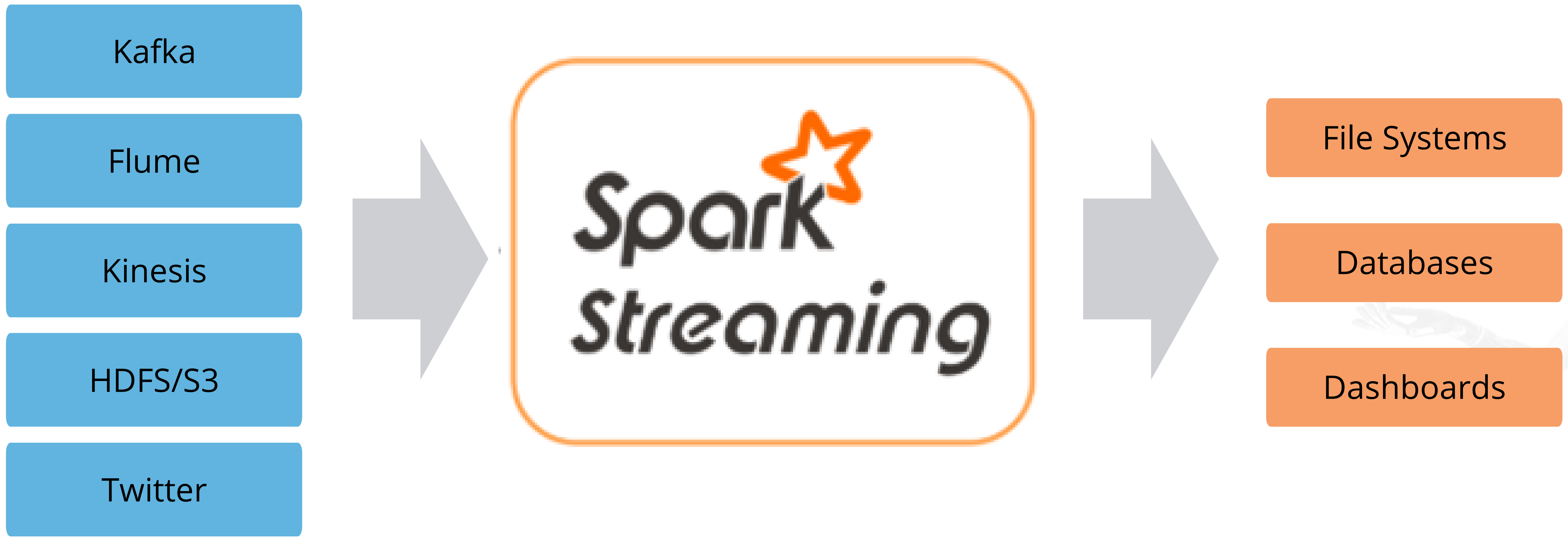
Problem Statement: In this demonstration, you will learn the basics of real-time processing.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

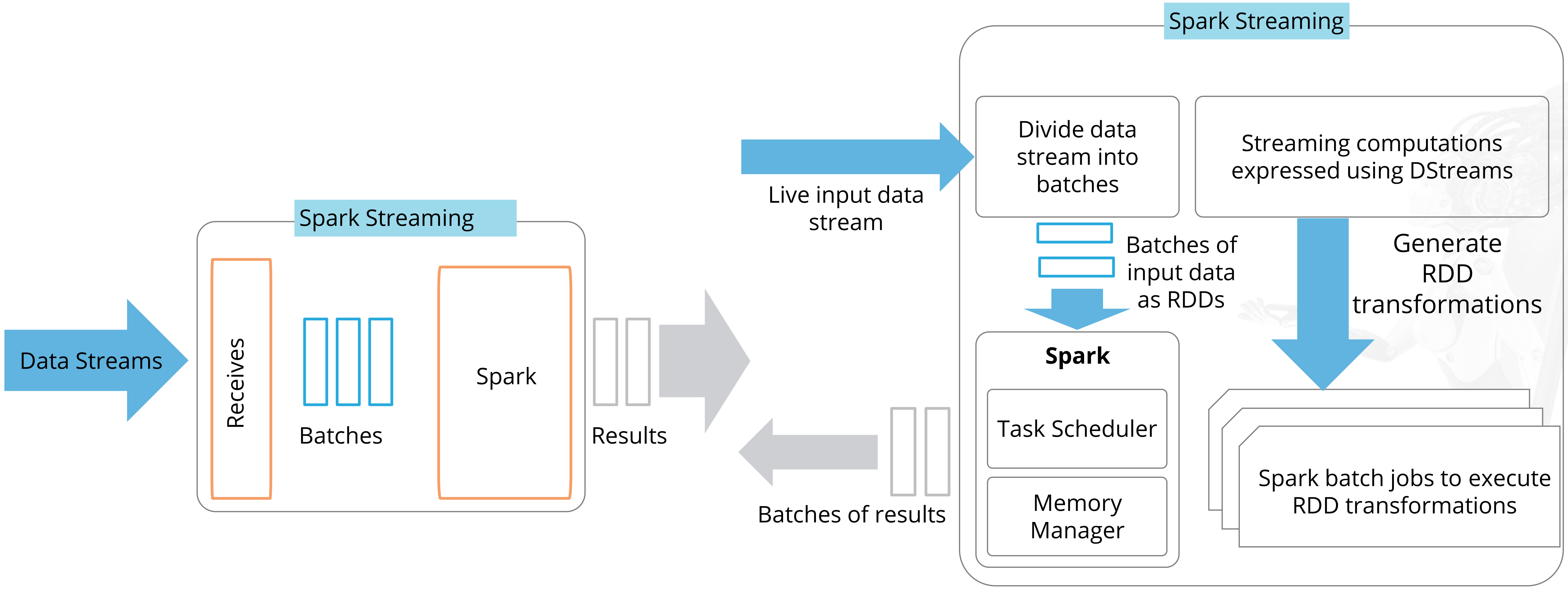
Spark Streaming

Introduction to Spark Streaming

Spark Streaming is an extension of the core Spark API.



Working of Spark Streaming


Data Streams
Spark Streaming
Receives
Batches
Spark
Results
Live input data stream
Batches of results
Spark Streaming
Divide data stream into batches
Streaming computations expressed using DStreams
Batches of input data as RDDs
Generate RDD transformations
Spark
Task Scheduler
Memory Manager
Spark batch jobs to execute RDD transformations

Features of Spark Streaming

01

Discretized Stream (DStream)

Flume, Kafka, and Kinesis

02

Extensive Support

Scala, Java, and Python

Machine learning and graph processing algorithms

03

ZooKeeper and HDFS

State storage and leader election support

Streaming Word Count

Sample code to execute Spark Streaming:

Example:

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()
```

Writing Spark Streaming Application

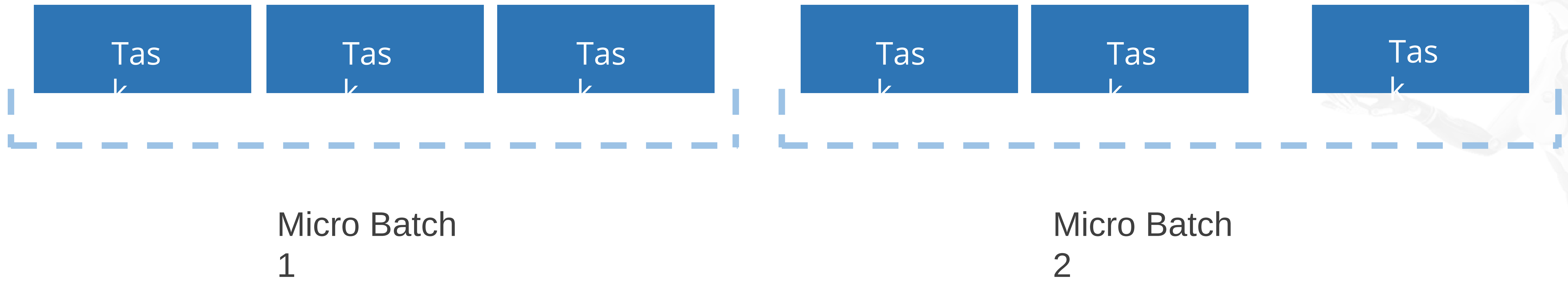
Duration: 15 mins

Problem Statement: In this demonstration, you will learn to write a Spark streaming application.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Micro Batch

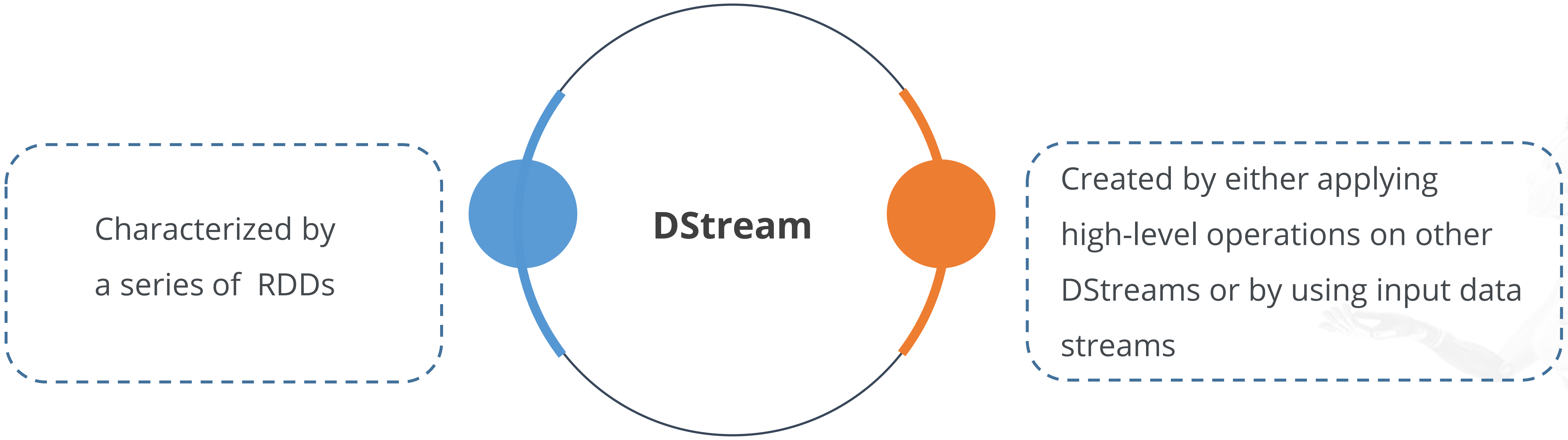
Micro batching handles a stream by a task or process as a sequence that contains data chunks.

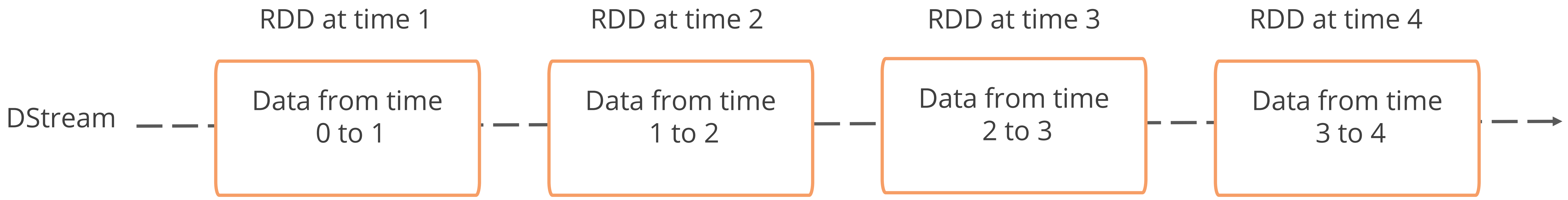


Introduction to DStreams

Introduction to DStreams

Discretized Stream (DStream) is the fundamental abstraction available in Spark Streaming.

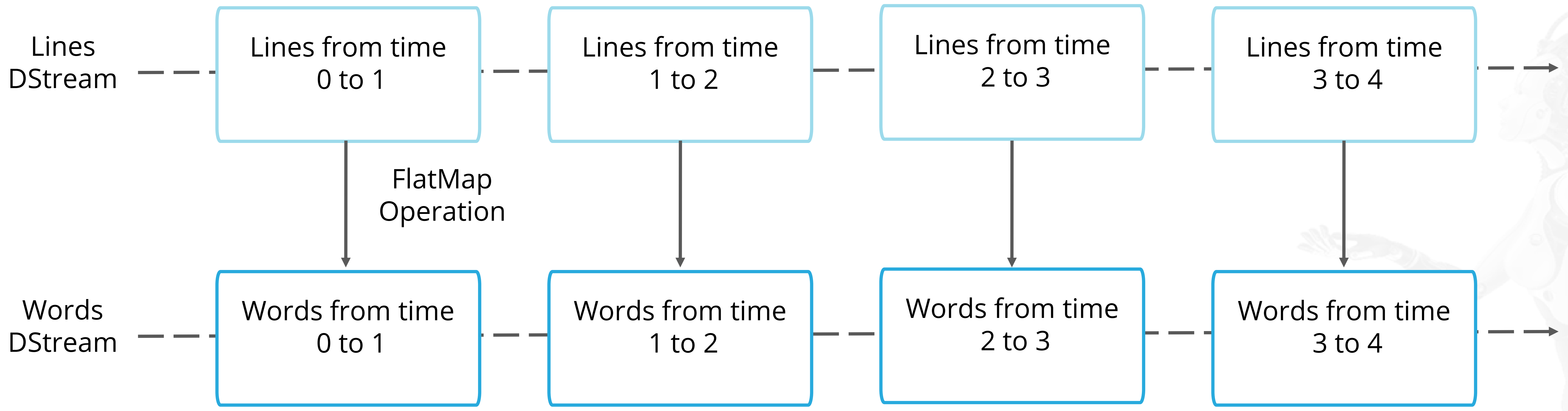

Characterized by a series of RDDs
DStream
Created by either applying high-level operations on other DStreams or by using input data streams



Series of RDDs

Introduction to DStreams

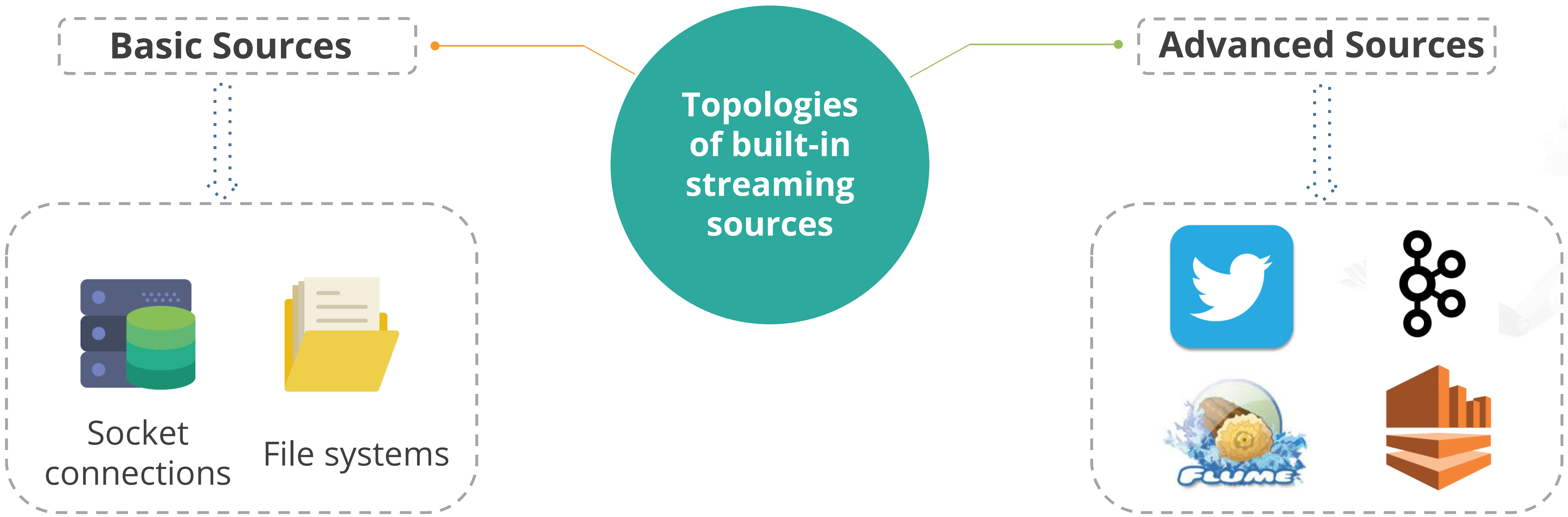
All operations applied on a DStream get translated to operations applicable on the underlying RDDs.



Input DStreams and Receivers

Input DStreams represent the input data stream received from streaming sources.

Except file stream, each input DStream is linked with a receiver object that stores the data received from a source.



Transformations on DStreams

Transformations on DStreams

Transformations on DStreams are similar to those of RDDs.

- A few of the common transformations on DStreams are given in the table below:

Transformations on DStreams

- map(func)
- flatMap(func)
- filter(func)
- repartition(numPartitions)
- union(otherStream)
- countByValue()
- reduce(func)
- reduceByKey(func, [numTasks])
- join(otherStream, [numTasks])

Output Operations on DStreams

- Output operations let the data of DStreams be pushed to external systems.
- They trigger the real execution of all the DStream transformations.

print()

saveAsTextFiles(prefix, [suffix])

saveAsObjectFiles(prefix, [suffix])

saveAsHadoopFiles(prefix, [suffix])

foreachRDD(func)

Design Patterns for Using ForeachRDD

- DStream.foreachRDD is a powerful primitive that lets the data to be sent to external systems.

Example:

```
DStream.foreachRDD { rdd => val connection = createNewConnection()      //
executed at the driver
rdd.foreach { record => connection.send(record)  // executed at the worker } }
```

DataFrame and SQL Operations

To use DataFrames and SQL operations	Create an SQLContext using the SparkContext that the StreamingContext uses
To allow restarting in case of driver failures	Create a lazily instantiated singleton instance of SQLContext

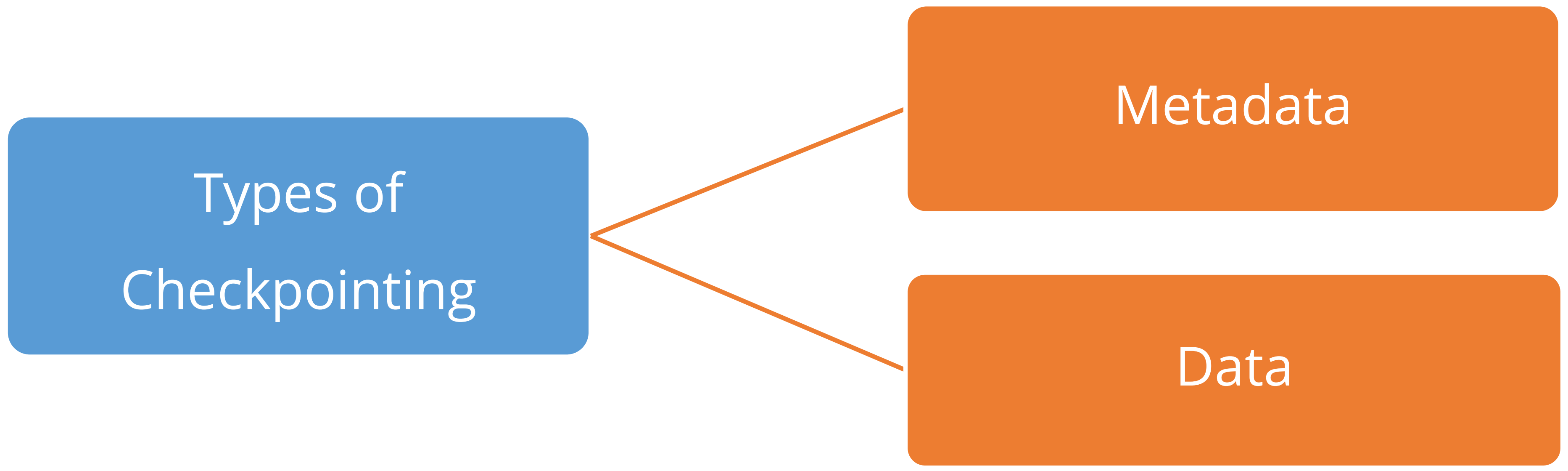
Example:

```
val words: DStream[String] = ...
words.foreachRDD { rdd =>
 // Get the singleton instance of SQLContext
val sqlContext = SQLContext.getOrCreate(rdd.SparkContext)
import sqlContext.implicits._
val wordsDataFrame = rdd.toDF("word")
// Register as table  and Do word count on DataFrame using SQL and print it
wordsDataFrame.registerTempTable("words")
val wordCountsDataFrame =       sqlContext.sql("select word, count(*) as total
from words group by word")
wordCountsDataFrame.show() }
```

Checkpointing

A streaming application must be:

- Resilient to failures
- Fault-tolerant storage system



Enabling Checkpointing

Requirements

- Using stateful transformations
- Recovering from failures of the driver running the applications

Socket Stream

- A socket is created on the driver's machine. The code residing outside the closure of the DStream is implemented in the driver, while the rdd.foreach method is implemented on every distributed RDD partition.
- The socket and computation are performed in the same host, which makes it effective.

Example:

```
crowd.foreachRDD(rdd =>
{rdd.collect.foreach(record=>{
out.println(record)
})
})
```

File Stream

- A DStream can be created to read data from files on any file system that is compatible with the HDFS API such as HDFS, S3, and NFS.

Example:

```
streamingContext.fileStream[KeyClass,
ValueClass, InputFormatClass](dataDirectory)
streamingContext.fileStream<KeyClass,
ValueClass, InputFormatClass>(dataDirectory);
streamingContext.textFileStream(dataDirectory)
```

Unassisted Practice

Spark Streaming

Duration: 15 mins

Problem Statement: Perform the below tasks:

- Build a network wordcount program using Spark streaming
- Make sure you run netcat; for example, nc -lk 9999
- Submit the jar file with wordcount program
- Type something on nc and make sure word count is done as part of the job

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Unassisted Practice

Steps to Perform

- **Word count program code**

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{ Seconds, StreamingContext }
import StreamingContext._
import org.apache.hadoop.conf._
import org.apache.hadoop.fs._


object Streaming {

def main(args: Array[String]) {

val sparkConf = new SparkConf().setAppName("HdfsWordCount").setMaster("local[4]")

val ssc = new StreamingContext(sparkConf, Seconds(2))
```

Unassisted Practice

Steps to Perform

- **Word count program code**

```
// create the FileInputDStream on the directory and use the stream to count words in newly created files

   val lines = ssc.socketTextStream("localhost", 1234)
   val words = lines.flatMap(_.split(" "))
   val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
   wordCounts.print()
   ssc.start()
   ssc.awaitTermination()
  }

}
```

Unassisted Practice

Steps to Perform

- **Spark submit command**

```
spark-submit --class "Streaming" scala-spark-training_2.10-1.0.jar
```

- **Netcat input**

```
$ nc -lk 1234
   Hi there, this is John
   I am learning Big data from Simplilearn
   Hi there, this is Alice
   I am learning Apache Spark from Simplilearn
```

DATA AND ARTIFICIAL INTELLIGENCE

State Operations

State Management

There are two primary conditions in state management:

Stateless

When a service is active but is not engaged in processing, it is said to be in a stateless condition

Stateful

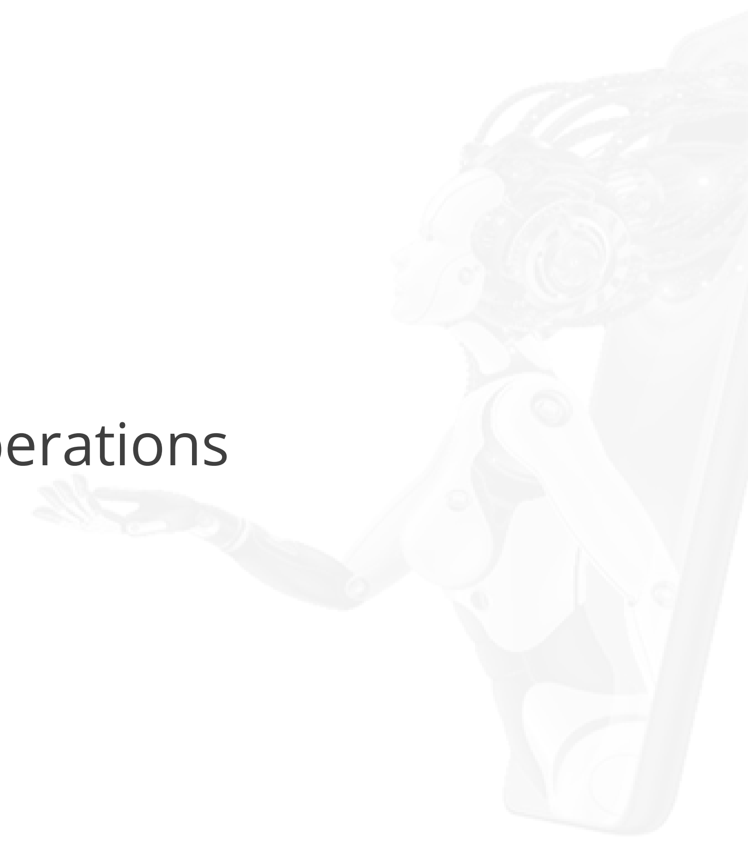
A service that is processing and retaining state data actively is in a stateful condition

Stateful Operations

01 Operate over various data batches

02 Include the updateStateByKey operation and all window-based operations

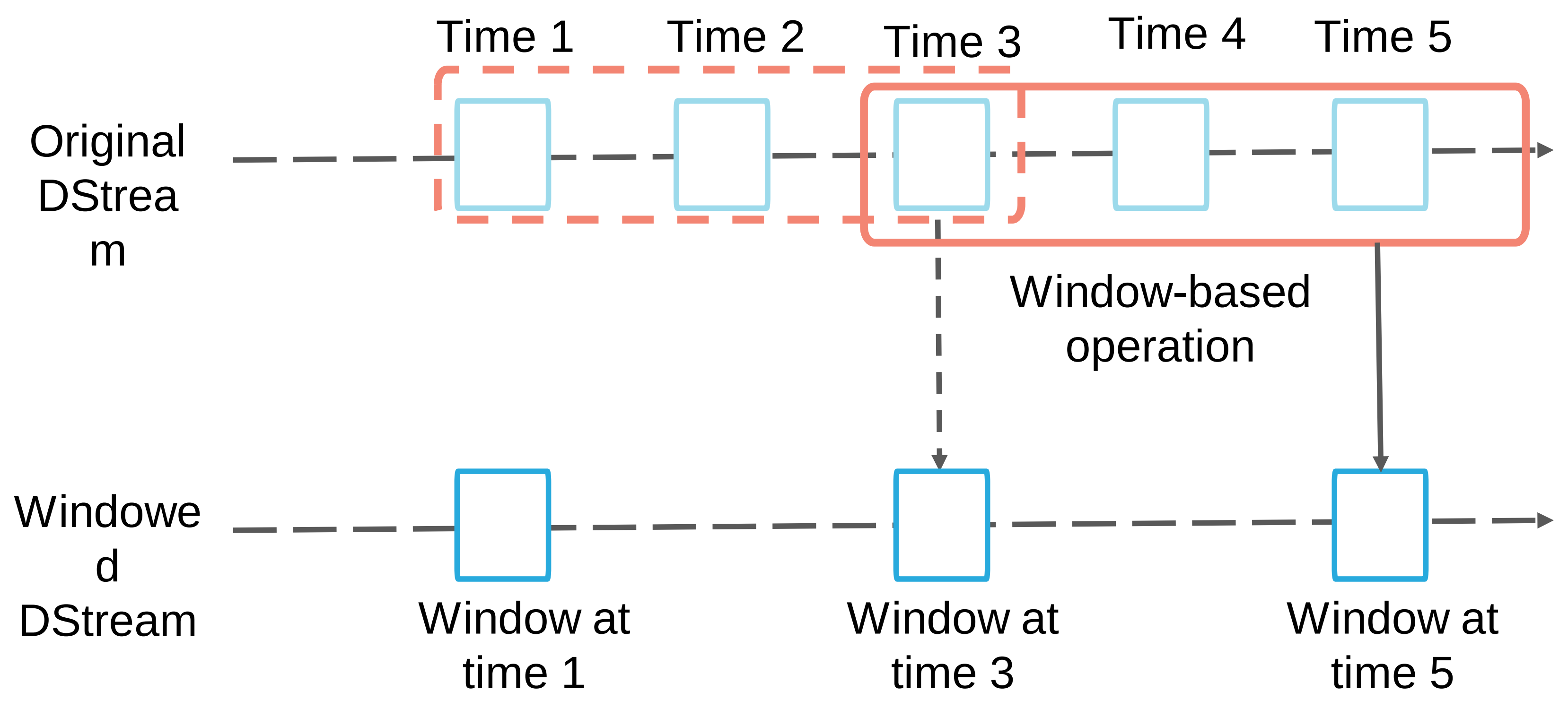
03 Dependent upon the earlier data batches



Windowing Operation

Window Operations

- Window operations let you implement transformations over a sliding window of data.



Example:

```
val windowedWordCounts =

pairs.reduceByKeyAndWindow((a:Int,b:Int)

=> (a + b), Seconds(30), Seconds(10)) })
```

Types of Window Operations

- Operations that take window length and slide interval as parameters are the following:

```
window(windowLength, slideInterval)
```

```
countByWindow(windowLength,slideInterval)
```

```
reduceByWindow(func, windowLength,slideInterval)
```

```
reduceByKeyAndWindow(func,windowLength, slideInterval, [numTasks])
```

```
reduceByKeyAndWindow(func, invFunc,windowLength, slideInterval, [numTasks])
```

```
countByValueAndWindow(windowLength,slideInterval,[numTasks])
```

Join Operations: stream-stream Join

- The first type, stream-stream joins, allows to join streams with other streams.

Example 1:

```
val stream1: DStream[String, String] = ...
val stream2: DStream[String, String] = ...
val joineDStream = stream1.join(stream2)
```

Example 2 (joining over windows of the streams):

```
val windoweDStream1 = stream1.window(Seconds(20))
val windoweDStream2 = stream2.window(Minutes(1))
val joineDStream = windoweDStream1.join(windoweDStream2)
```

Join Operations: stream-dataset Join

- The second type, stream-dataset joins, allows to join a stream and a dataset.

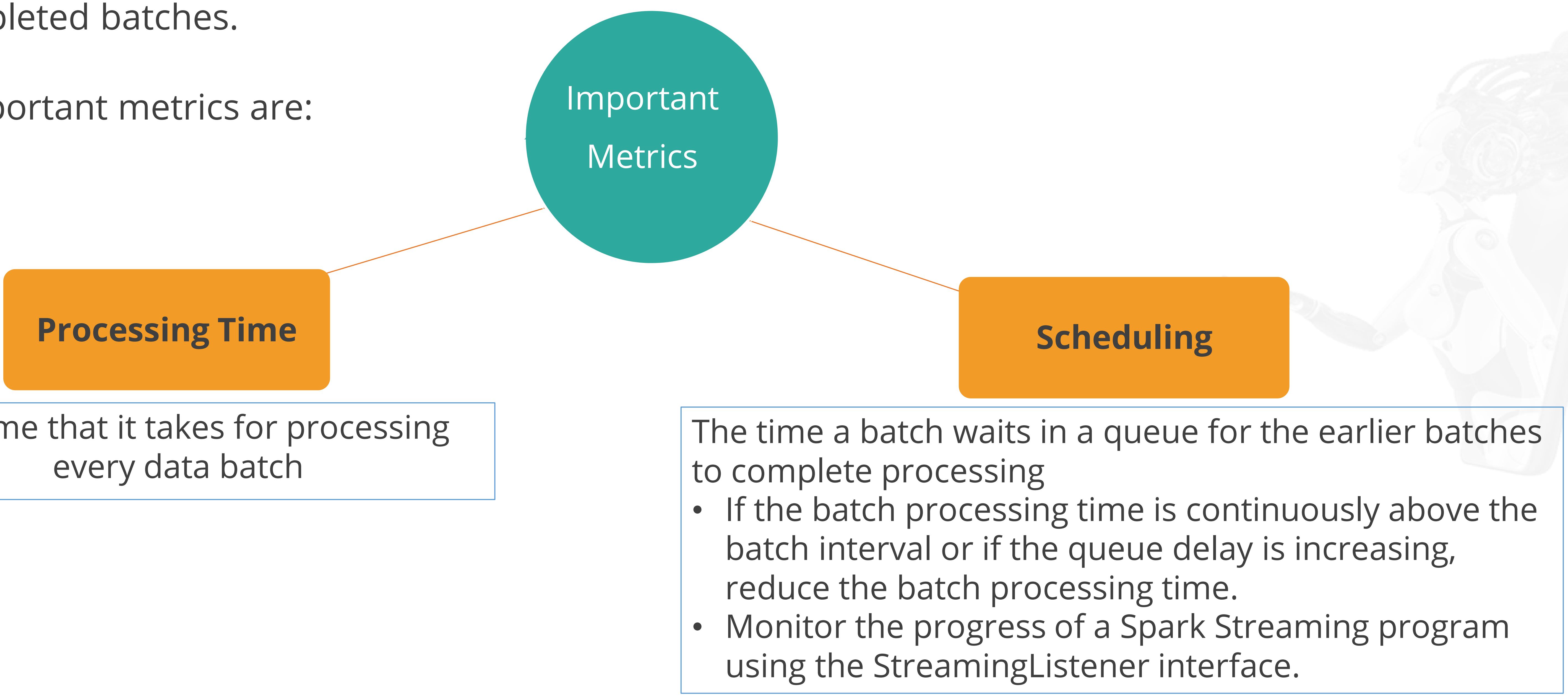
Example:

```
val dataset: RDD[String, String] = ...
val windoweDStream = stream.window(Seconds(20))...
val joineDStream = windoweDStream.transform { rdd => rdd.join(dataset) }
```

Monitoring Spark Streaming Application

- Spark Web UI displays a streaming tab that shows the statistics of the running receivers and details of completed batches.

The important metrics are:



The time that it takes for processing every data batch

The time a batch waits in a queue for the earlier batches to complete processing

- If the batch processing time is continuously above the batch interval or if the queue delay is increasing, reduce the batch processing time.
- Monitor the progress of a Spark Streaming program using the StreamingListener interface.

Assisted Practice

Windowing of Real-Time Data Processing

Duration: 15 mins

Problem Statement: In this demonstration, you will learn windowing of real-time data processing.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

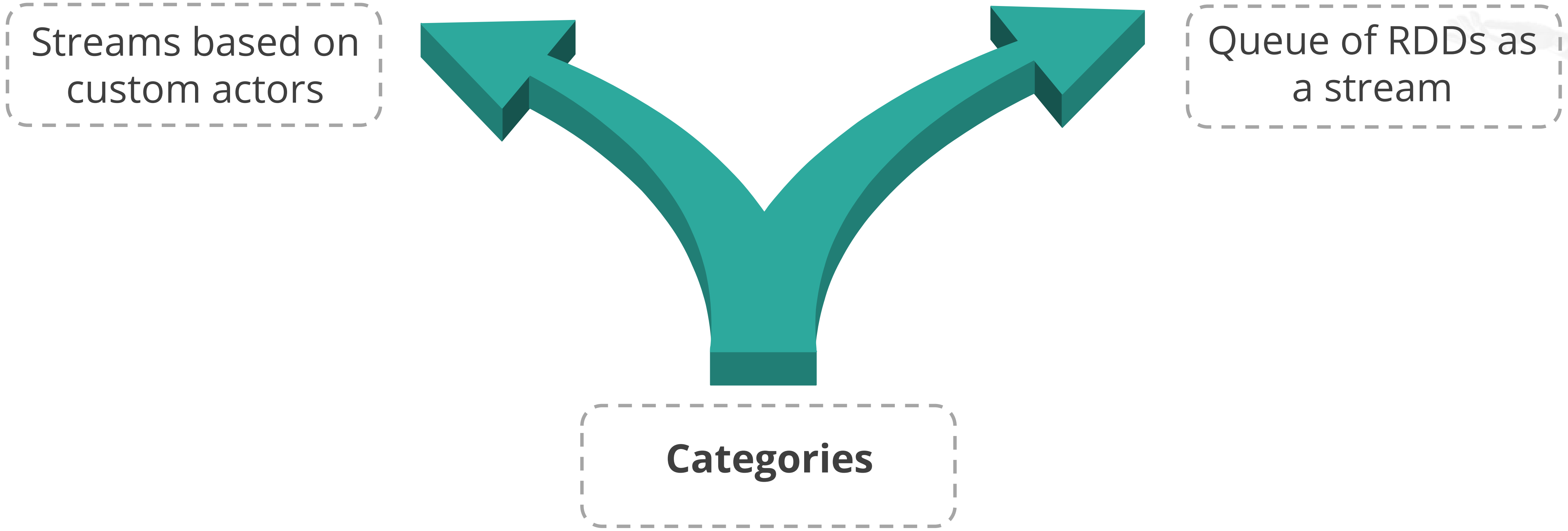
Spark Streaming Sources

Basic Sources

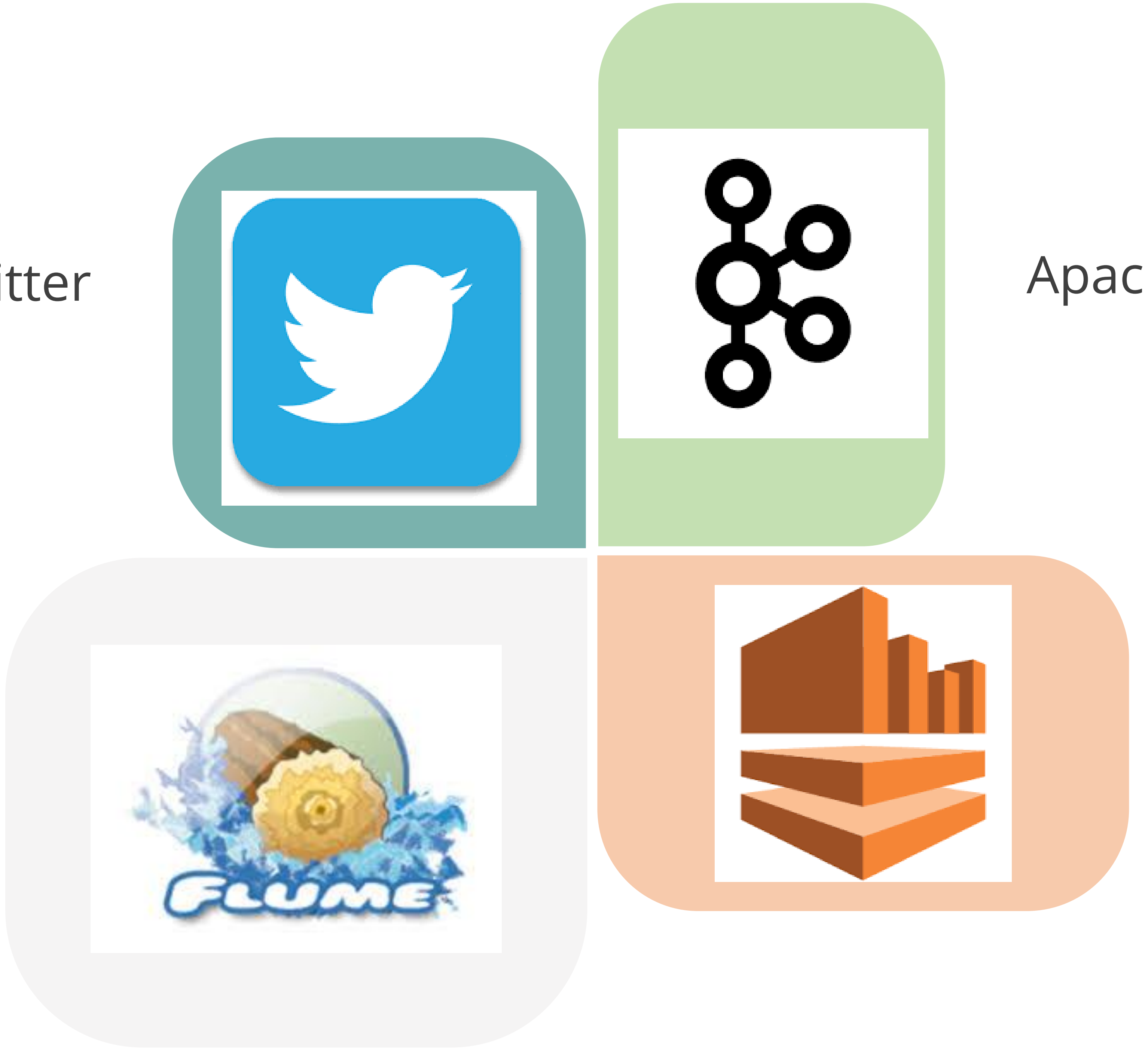
For basic sources, Spark streaming monitors the data directory and processes all files created in it.

Syntax:

```
streamingContext.fileStream[KeyClass, ValueClass, InputFormatClass](dataDirectory)
```



Advanced Sources



Advanced Sources: Twitter

To create a DStream using data from Twitter’s stream of tweets, follow the steps listed below:

01

Linking

The artifact Spark-streaming-twitter_2.11 needs to be added to the SBT/Maven project dependencies which is under org.apache.bahir

02

Programing

The TwitterUtils class needs to be imported and a DStream needs to be created: import org.apache.Spark.streaming.twitter._TwitterUtils.createStream(ssc, None

03

Deploying

An uber JAR needs to be generated with all dependencies

Assisted Practice

Processing Twitter Streaming Data

Duration: 15 mins

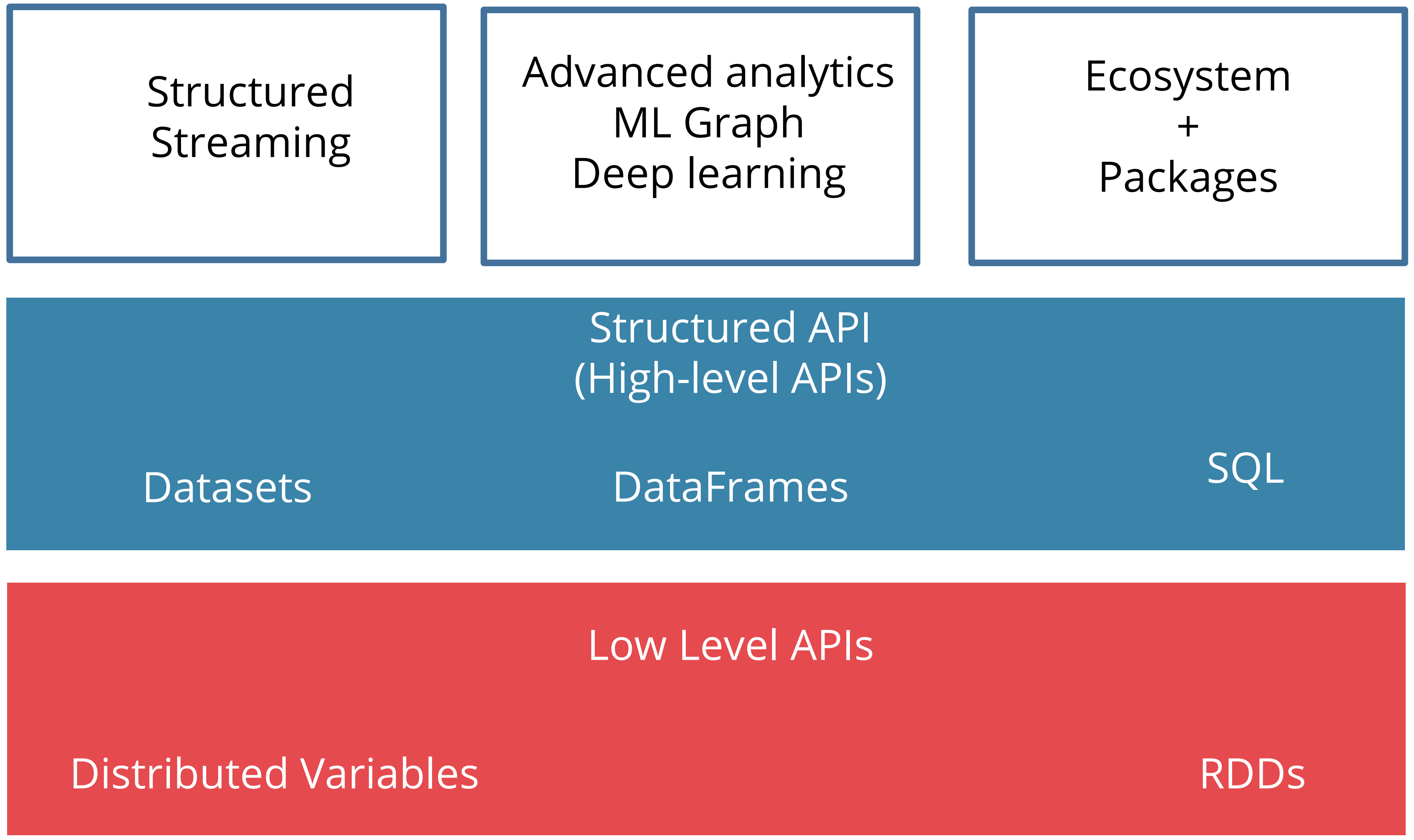
Problem Statement: In this demonstration, you will learn how to process Twitter streaming data and perform sentimental analysis.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Structured Spark Streaming

Introduction to Spark Structured Streaming

Structured Streaming is a high-level streaming API that is built on Spark SQL engine.



Why Spark Structured Streaming?

Spark Structured Streaming is introduced to overcome the shortcoming of DStreams.

Limitations of DStreams

Works only with the batch time

Difficult to deal with delayed data

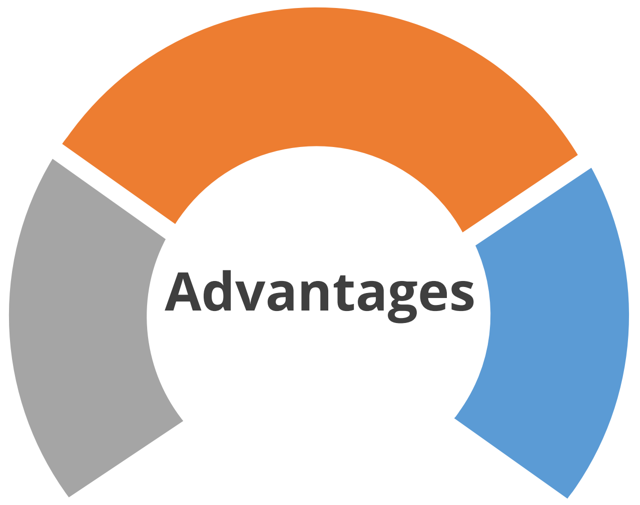
DStreams API is different from RDD API

Unreliable streaming

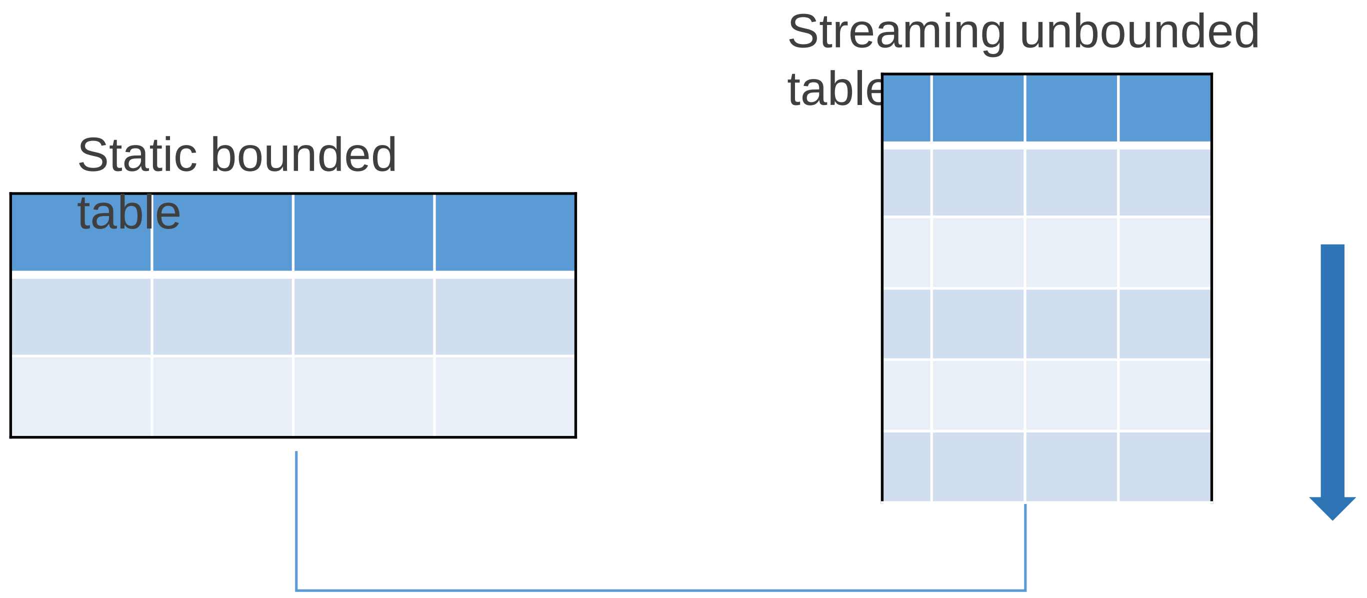
Advantages of Spark Structured Streaming

Easy to use

Better performance through SQL optimizations



One unified API for both batch and streaming sources



The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended.

Batch vs. Streaming

● Sample code

The new Structured Streaming API enables you to easily adapt the batch jobs that you have already written to deal with a stream of data.

Batch

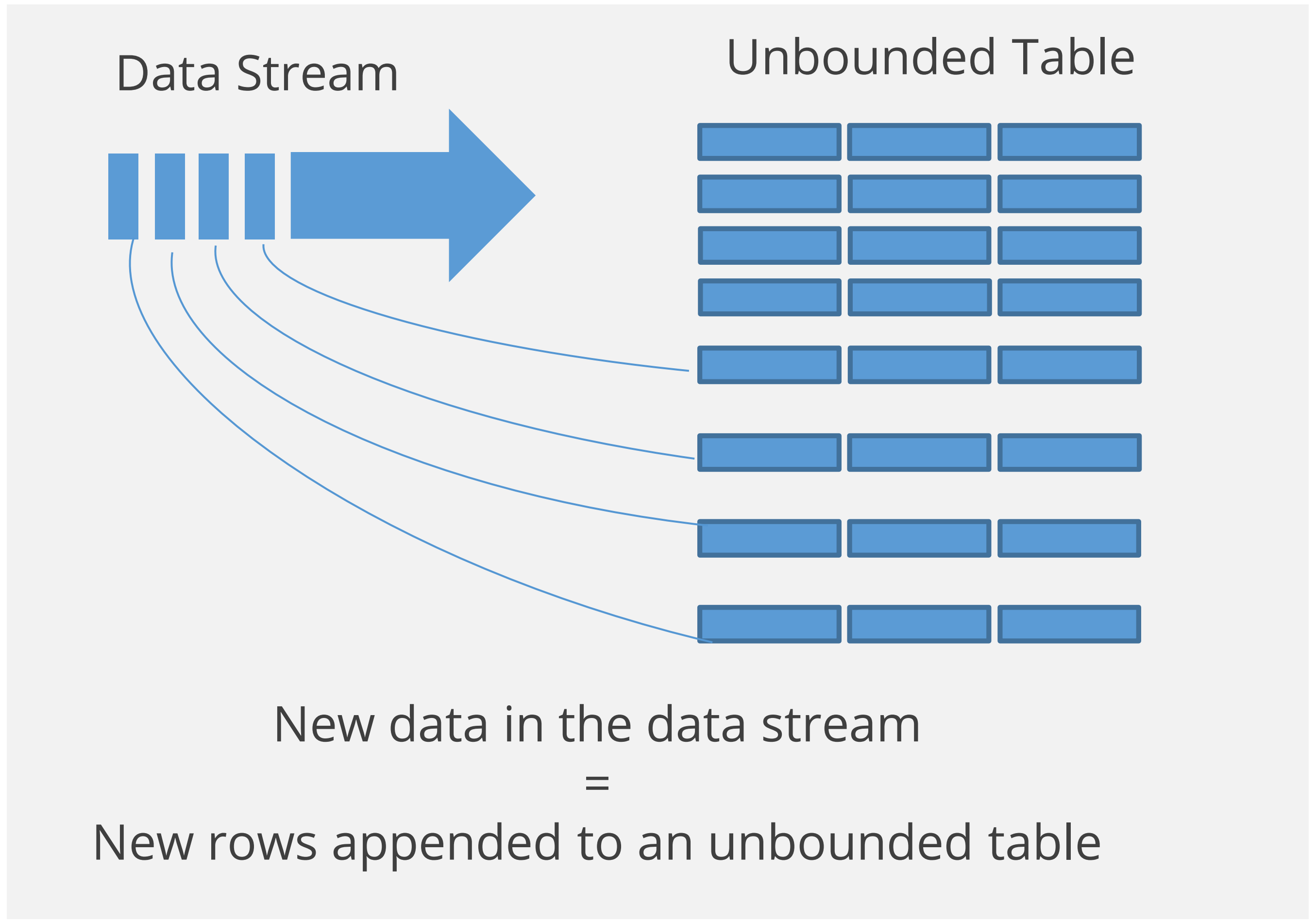
```
val callSchema = new
StructType().add(“Name”,”String").add("City","
String").add("Country","String").add("CallTs",
"String").add("CallCharge","integer")
val personDF =
Spark.read.schema(callSchema).json(“/sampledat
a/json/call.json")
val peopleParis =
personDF.select("City","Callcharge").where("Ci
ty = ‘Paris'").where("CallCharge > 500")
peopleParis.write.format("parquet").save("/sam
pledata/streaming/output")
```

Continuous Stream

```
val personSchema = new
StructType().add(“Name”,”String”).add("City","
String").add("Country","String").add("CallTs",
"String").add("CallCharge","integer")
val personDF =
Spark.reaDStream.schema(callSchema).json(“/sam
pledata/json/call.json")
val peopleParis =
personDF.select("City","Callcharge").where("Ci
ty = ‘Paris'").where("CallCharge > 500")
peopleParis.write.format("parquet").save("/sam
pledata/streaming/output")
```

Use Case: Banking Transactions

- **Scenario**: Banking transaction records containing the account number and transaction amount are coming in a stream.



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

Advanatges

- Allows whatever data processing that is possible using a DataFrame to be possible with the stream data as well
- Reduces the burden on application developers
- Allows them to focus on the business logic of the application rather than the infrastructure related aspects

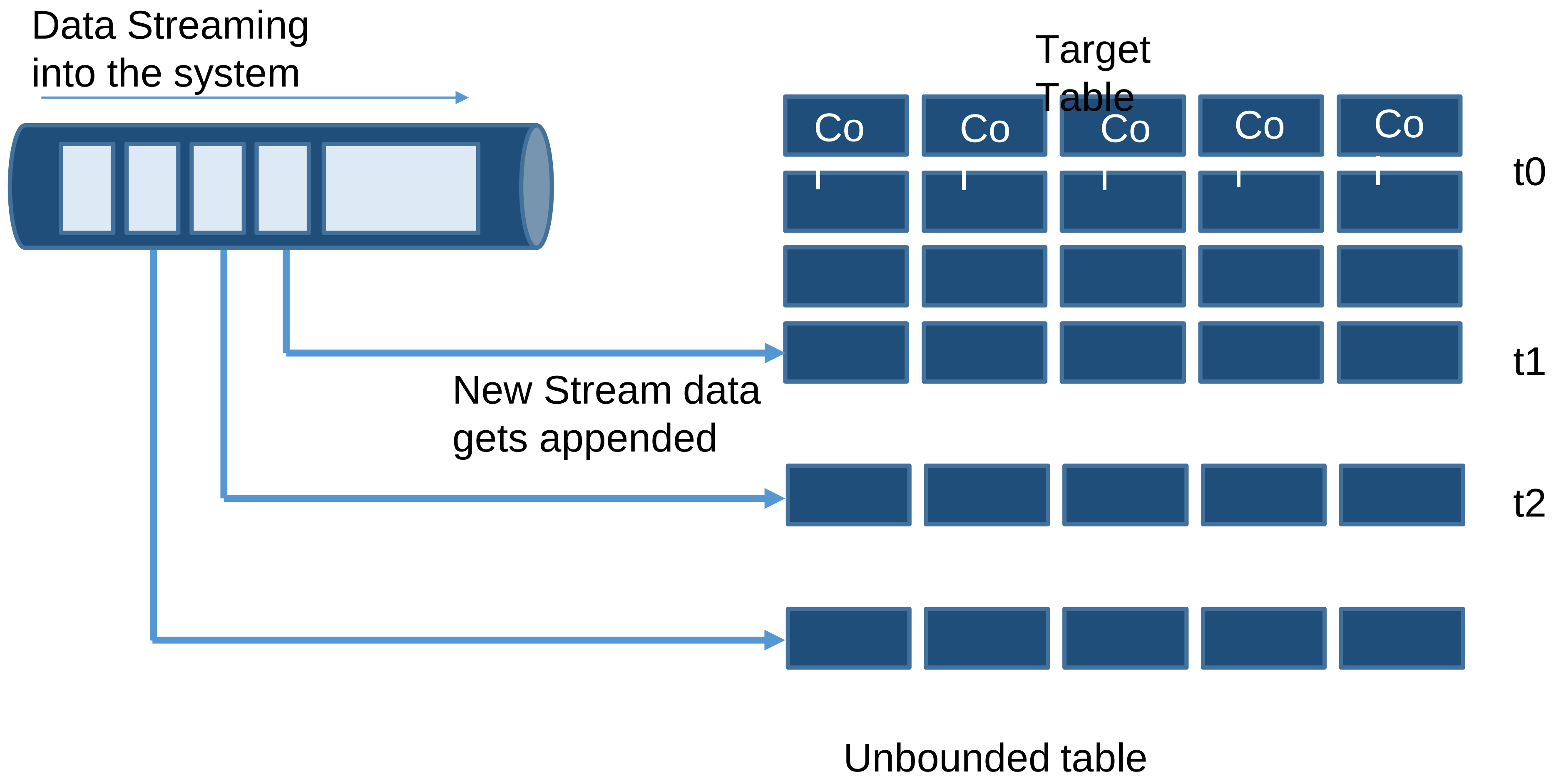
Spark Streaming vs. Spark Structured Streaming

Spark Streaming	Spark Structured Streaming
• Dstream only uses RDD API	• Uses DataFrame/Dataset API
• Tracking of state between batch times for cumulative statistics is complex in DStream	• User just needs to take care of business logic
• No guarantee of data integrity	• Automatically handles consistency and reliability
• The API works only with batch	• The API is same so you can write to the same data destination and can also read it back.

Structured Streaming Architecture, Model, and Its Components

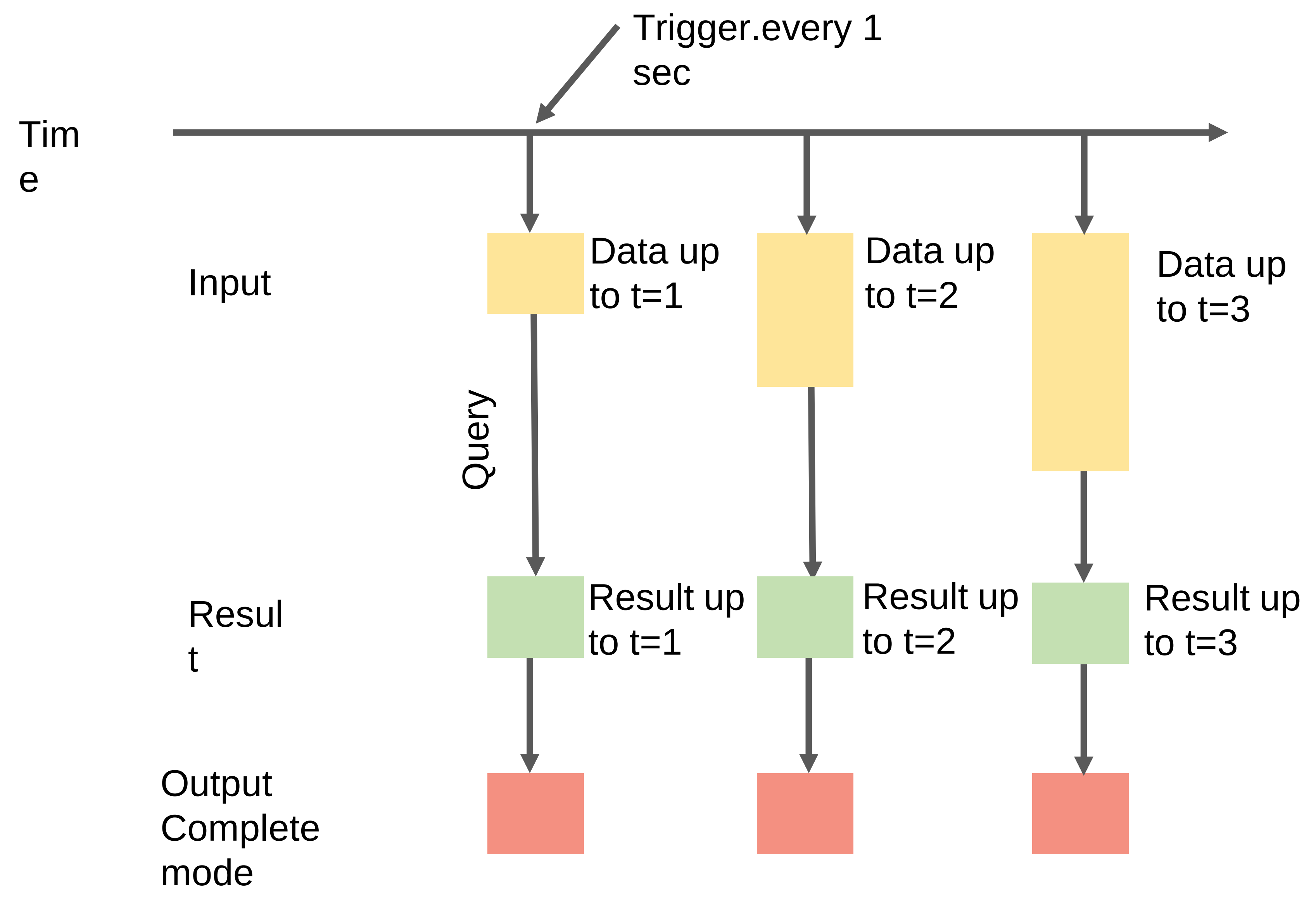
Structured Streaming Architecture

- Structured Streaming treats all the arriving data as an **unbounded input table**.
- Every time there is a new item in the stream, it gets appended as a row in the input table at the bottom.

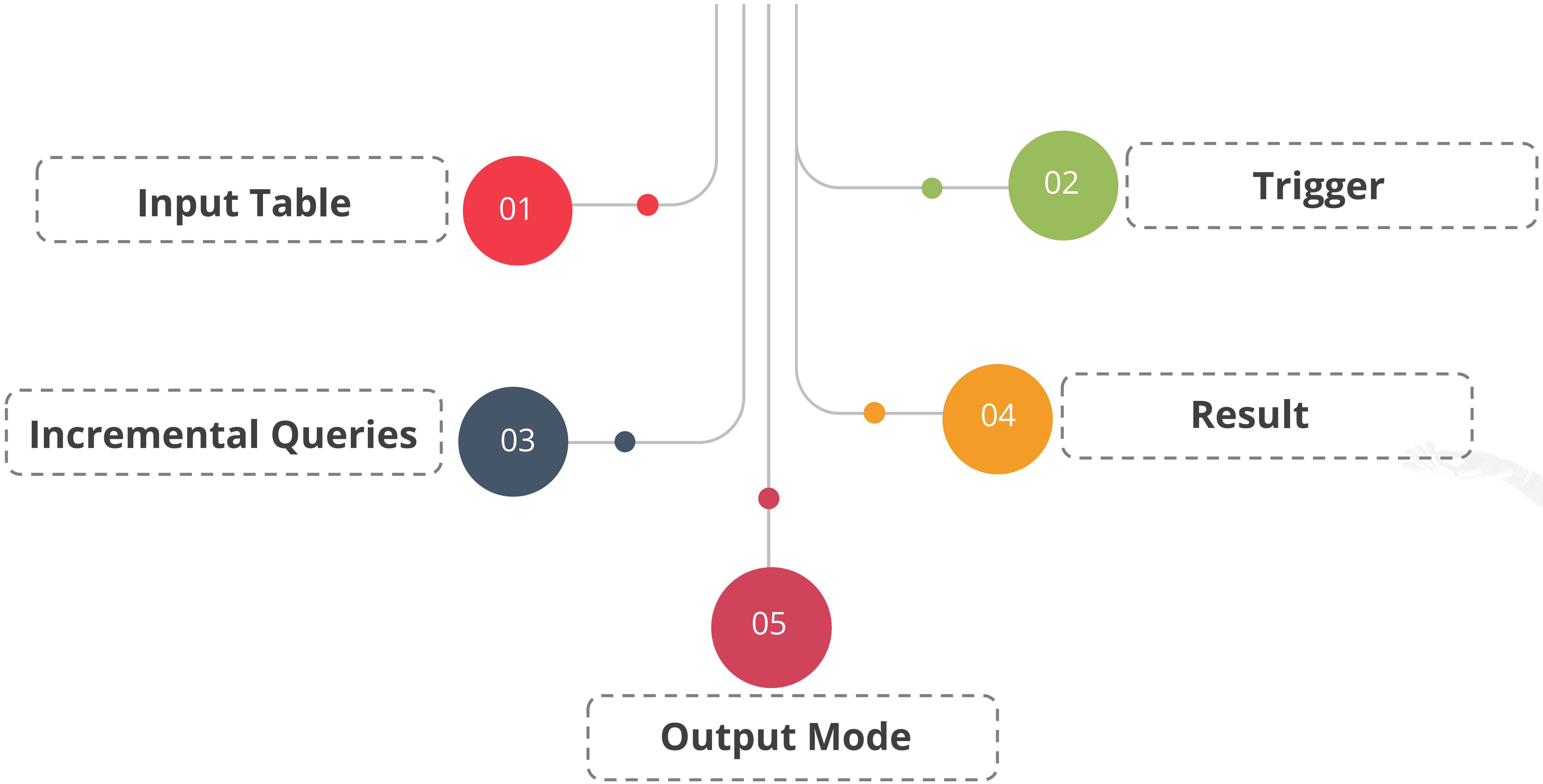


Structured Streaming Model

Incremental execution on streaming data

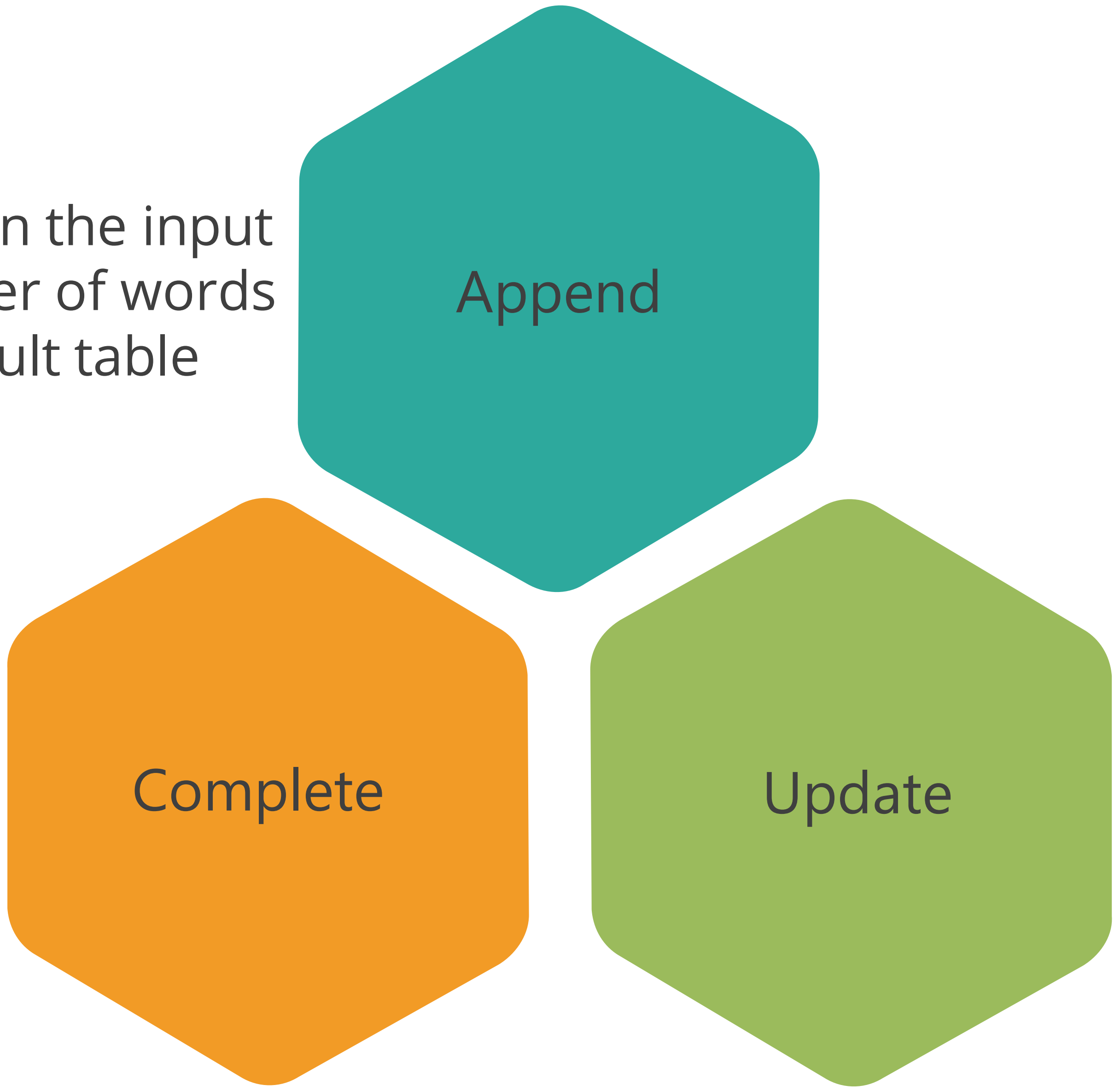


Components of Structured Streaming Model



Output Modes

A query defined by the developer on the input table and count to count the number of words that in turn computes a final result table



A query defined by the developer on the input table and count to count the number of words that in turn computes a final result table

A query defined by the developer on the input table and count to count the number of words that in turn computes a final result table

Output Sinks

File sink: Stores the output to a directory

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

Foreach sink: Runs arbitrary computation on the records in the output

```
writeStream

.foreach(...)

.start()
```

Output Sinks

Console sink (for debugging):

```
 writeStream

.format("console")

.start()
```

Memory sink (for debugging):

```
writeStream

.format("memory")

.queryName("tableName")

.start()
```

Structured Streaming APIs

Features of Structured Streaming APIs

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

Example

```
val socketDF = Spark
  .reaDStream
  .format(“socket")// Reading Data from socket(Socket Datasource)
  .option("host", "localhost")
  .option("port", 9999)
  .load()
```

Data Sources

File source

Socket source (for testing)

Kafka source

Example

```
val inputDF = Spark.reaDStream.json("s3://logs")
```

Reading data from JSON file

Operations on Streaming DataFrames and Datasets

- With Structured Streaming API, we can perform:
  - Untyped, SQL-like operations
  - Typed RDD-like operations

Example

```
// Select the persons which have age more than 60
df.select("name").where("age > 60") // using untyped APIs
ds.filter(_.age > 60).map(_.name)   // using typed APIs
// Running count of the number of counts for each value
df.groupBy("value").count()         // using untyped API
```

Parsing Data with Schema Inference

Code to read a CSV file using schema inference

Example

```
import org.apache.Spark.sql.types._
import org.apache.Spark.sql.catalyst.ScalaReflection
import org.apache.Spark.sql.functions._
case class Employee(
name:String,
city:String,
country:String,
age:Option[Int]
)
//Step 1:-Create schema for parsing data
val caseSchema =
(ScalaReflection.schemaFor[Employee].dataType.asInstanceOf[StructType])
//Step2:-Schema is passed to the stream
val empStream =
(Spark.reaDStream.schema(caseSchema).option("header",true).option("maxfilespertrigger
",1).csv("data/people.*").as[Employee])
//Step 3:Write the results to the screen
(empStream.writeStream.outputMode("append").format("console").start)
```

Constructing Columns in Structured Streaming

- Structured Streaming makes use of column objects for manipulating data
- A column can also be constructed from other columns using binary operator

Example

```
 (empStream.select($"country" === "France" as "in_France", $"age" <35 as “under_35”,
 'country startsWith "U" as
"U_country").writeStream.outputMode("append").format("console").start)
```

groupby and Aggregation

- Spark structured stream uses a groupby operator
- You can then perform different aggregation operations to this group

Example

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

- For complex aggregations, "agg" function is used

Example

```
(empStream.groupBy('country).agg(first("country") as "country",
count  ( "age")) .writeStream.outputMode("complete").format("console").start)
```

Joining Structured Stream with Datasets

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

Example

```
val staticDf = Spark.read. ...
val streamingDf = Spark.reaDStream. ...

streamingDf.join(staticDf, "type")  // inner equi-join with a static DF
streamingDf.join(staticDf, "type", "right_join")  // right outer join with a static
DF
```

SQL Query in Spark Structured Streaming

- Writing SQL queries directly on stream:

1. Create a temporary Table

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

2. Write a SQL query on created temp table

```
val query = Spark.sql("Select country,avg(age) from empTable group by country")
```

3. Write the results to the console

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

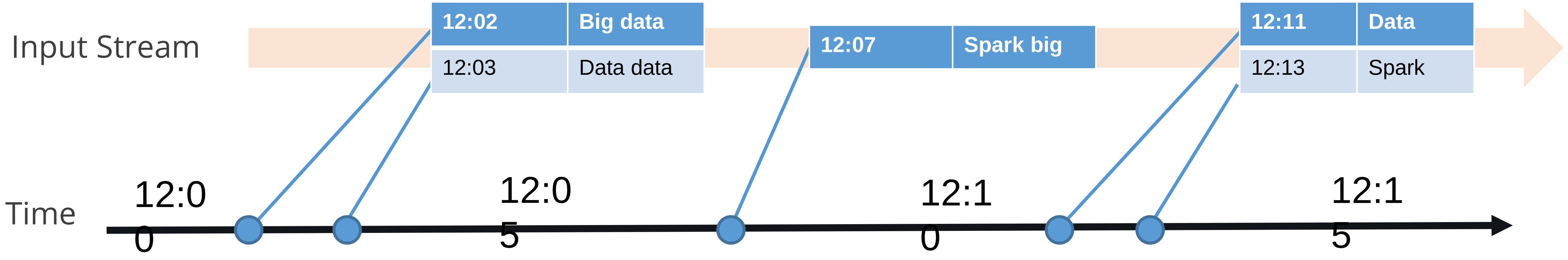
Windowed Operations on Event-Time

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

WordCount Example

```
// Split the lines into words, retaining timestamps
 val words = lines.as[(String, Timestamp)].flatMap(line =>line._1.split(" ").map(word =>
(word, line._2))).toDF(“word", "timestamp")
// Group the data by window and word and compute the count of each group
  val windowedCounts = words.groupBy(window($"timestamp", windowDuration, slideDuration),
     $”word").count().orderBy("window")
```

Windowed Grouped Aggregations



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

With Structured Streaming, expressing such windows on event-time is simply performing a special grouping using the window() function.

Failure Recovery And Checkpointing

- Structured Streaming allows recovery from failures, which is achieved by using checkpointing and WAL.
- Configure a query with a checkpoint location, and the query will save all the progress information and the running aggregates for the checkpoint location.
- The checkpoint location should be a path in an HDFS compatible file system, and can be set as an option in the DataStreamWriter when starting a query.

Example

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

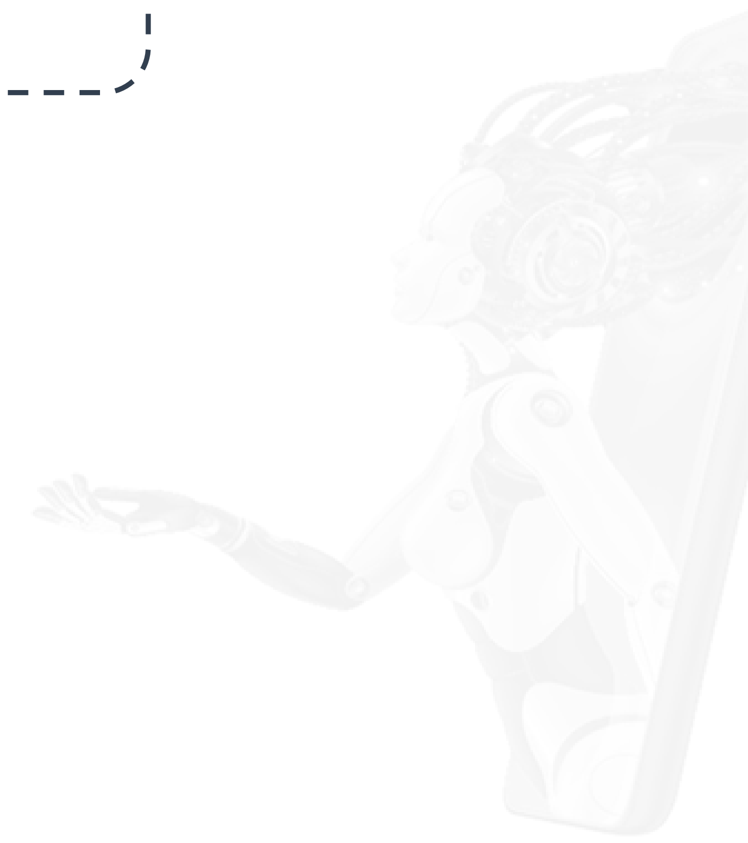
Use Cases

Real-world scenarios where Spark streaming is used

Analyze streams of real-time sensor data

Analyze streams of web log data

Streaming ETL pipelines prepare raw, unstructured data into a form that can be queried easily and efficiently

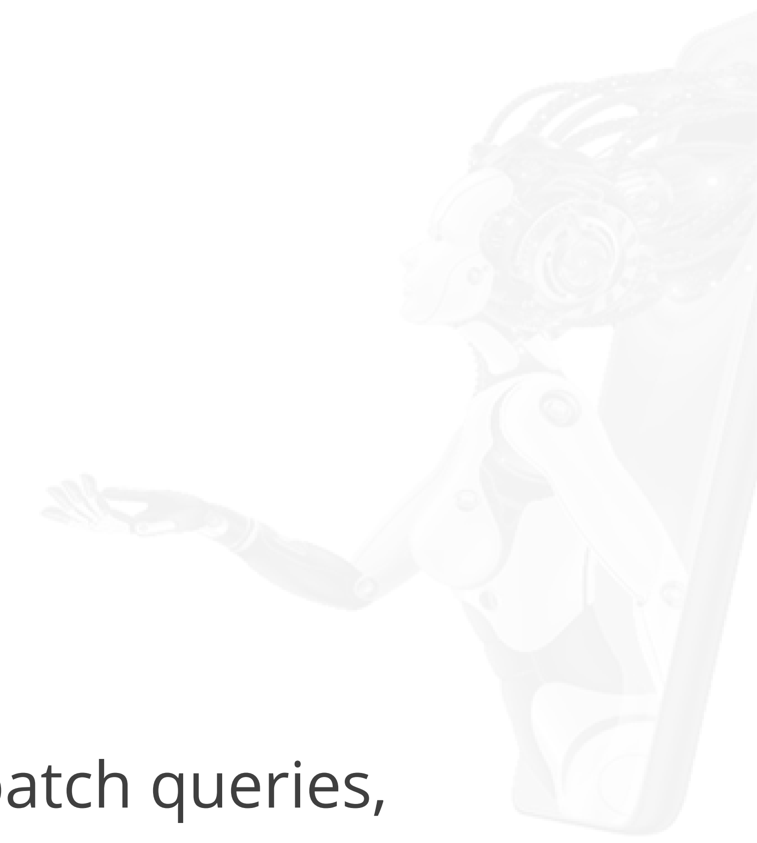


When to Use Structured Streaming

01 To create a streaming application

02 To provide data consistencies and exactly-once semantics

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



Assisted Practice

Streaming Pipeline

Duration: 15 mins

Problem Statement: In this demonstration, you will learn how to create a streaming pipeline.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Key Takeaways

You are now able to:

- Explain concepts of Spark Streaming
- Understand the Lambda and Kappa architecture
- Explain the concepts of Spark Structured Streaming
- Explain Join and Window operations

Knowledge Check

Knowledge Check

1

_________ attempts to balance high-throughput MapReduce frameworks with low-latency real-time processing.

a. Lambda architecture

b. Kappa architecture

c. Both A and B

d. None of the above

Knowledge Check

1

_________ attempts to balance high-throughput MapReduce frameworks with low-latency Real-time processing.

a. Lambda architecture

b. Kappa architecture

c. Both A and B

d. None of the above

The correct answer is **a.**

Lambda Architecture (LA) attempts to balance high-throughput MapReduce frameworks with low-latency real-time processing.

Knowledge Check

2

Which of the following data sources is supported in Spark Streaming?

a. Twitter

b. Kafka

c. Flume

d. All of the above

Knowledge Check

2

Which of the following data sources is supported in Spark Streaming?

a. Twitter

b. Kafka

c. Flume

d. All of the above

The correct answer is **d.**

Spark Streaming supports all these advance data sources.

Knowledge Check

3

Which of the following statements is true 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, func, to each RDD generated from the stream

Knowledge Check

3

Which of the following statements is true 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, func, 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

4

When should you use Structured Streaming?

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

4

When should you use Structured Streaming?

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.**

You can use Structured Streaming for all the mentioned cases.

Lesson-End Project

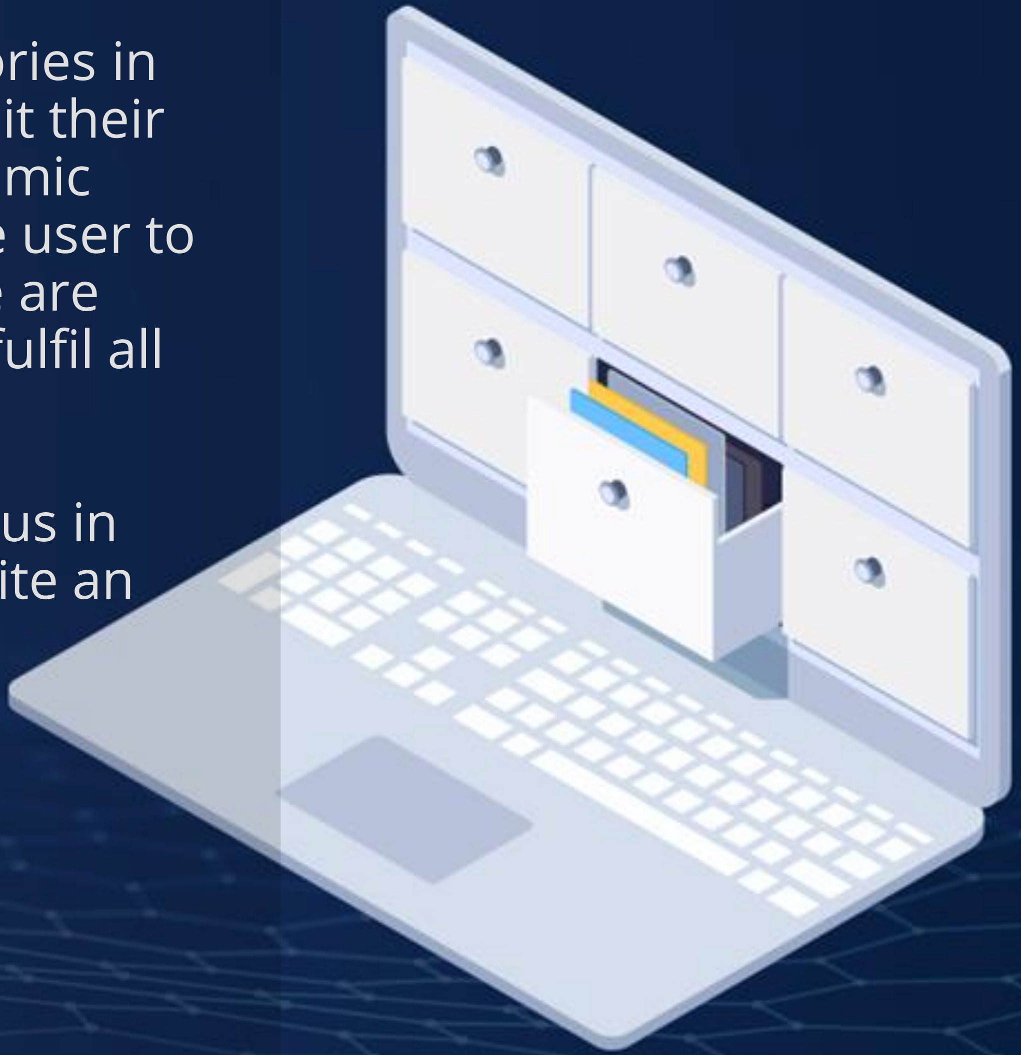
Problem Statement:

Alibaba is an e-commerce website that sells products online across different categories in different countries. Festivals are coming up and they want to make sure that they hit their target revenue. To make sure of that, they have decided to provide a trending dynamic banner where users can see the trending categories and brands which can help the user to decide which brand's product to purchase and to see from which categories people are purchasing the most. This will also help the company to keep enough inventory to fulfil all orders.

Currently, their system is using Hadoop MR which is not providing the trending status in real-time. They hired you as a big data engineer to modify the existing code and write an optimized code that will work for any time duration.
For example, trending brands in the last 5 minutes.

You have been given transactions.csv file which contains the below fields:

1. Product Code
2. Description
3. Brand
4. Category
5. Sub Category



Lesson-End-Project

You have to write Spark Streaming jobs to find out:

1. The top 5 trending categories in the last 5 minutes which has the maximum number of orders
2. The bottom 5 brands in the last 10 minutes which has the least number of orders
3. Product units sold in the last 10 minutes

Note:

You have to send the above data from CSV to Kafka.
Make sure you have some delay while sending the data as real-time scenario orders take time and they are pushed whenever the end-user purchases something.
Also, avoid pushing all the data at once, in Kafka.

Thank You