Big Data Hadoop and Spark Developer

simplilearn

Stream Processing Frameworks and Spark Streaming

Learning Objectives

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

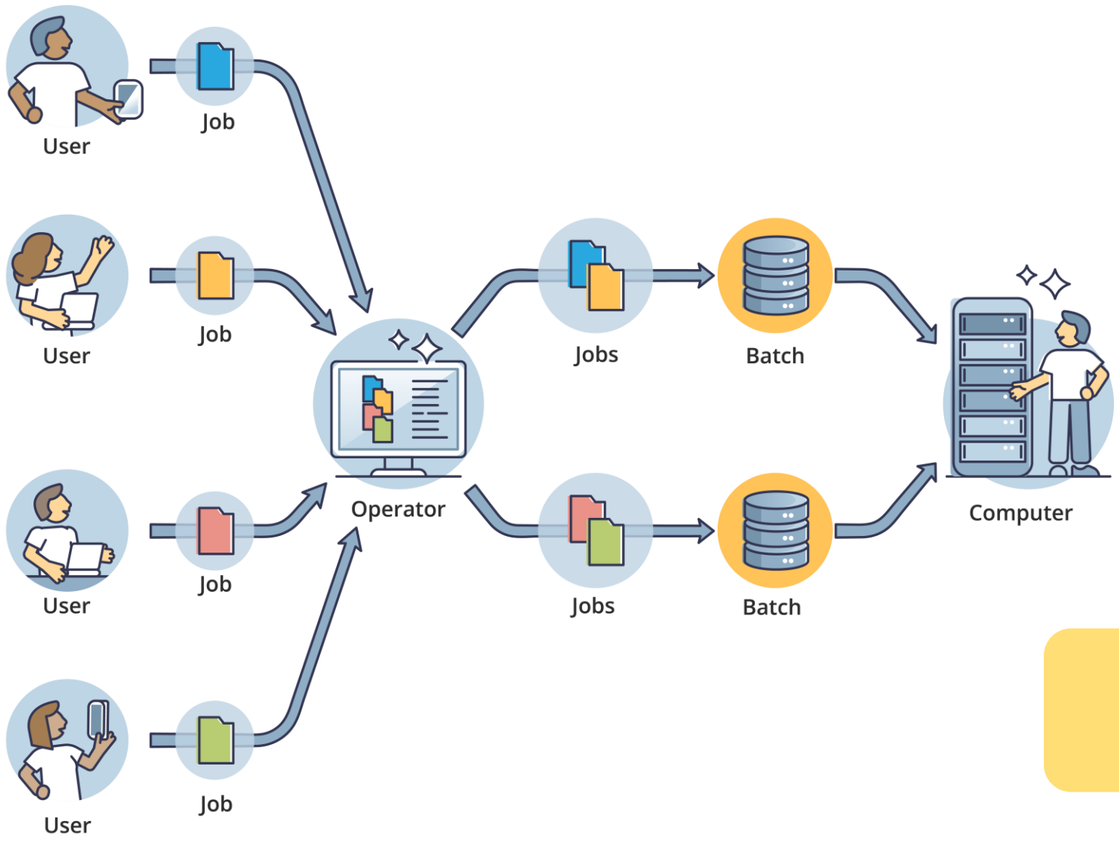
- Illustrate the concept of Spark Streaming
- Summarize the Streaming context and DStreams
- Specify the transformations on DStreams
- Work with window operators
- Design the slice, window, and ReduceByWindow operators

DATA AND
ARTIFICIAL INTELLIGENCE

Traditional Computing Methods and Its Drawbacks

Traditional Computing Methods

Batch processing is a computing method used to perform computing in the following ways:



The analysis of data is done in the form of batches.

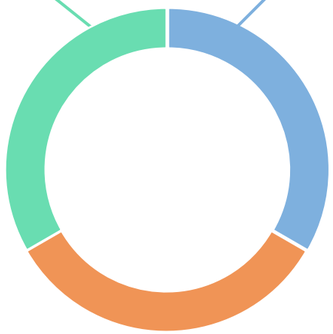
A processor must be active when a job is assigned to it.

The large jobs are divided into smaller parts to ensure efficiency during debugging.

Drawbacks of Traditional Computing Methods

The drawbacks of the traditional computing methods are:

The data is processed in batches, which makes this process ineffective for data tracking in real time.



The cost of batch processing is higher since a large amount of storage is required.

Solution to Drawbacks of Traditional Methods

Spark streaming is the solution to overcome the drawbacks of traditional methods.

It is a feature of the Spark API that allows data stream processing.

It is scalable and provides high throughput.

It provides fault-tolerant live data stream processing.

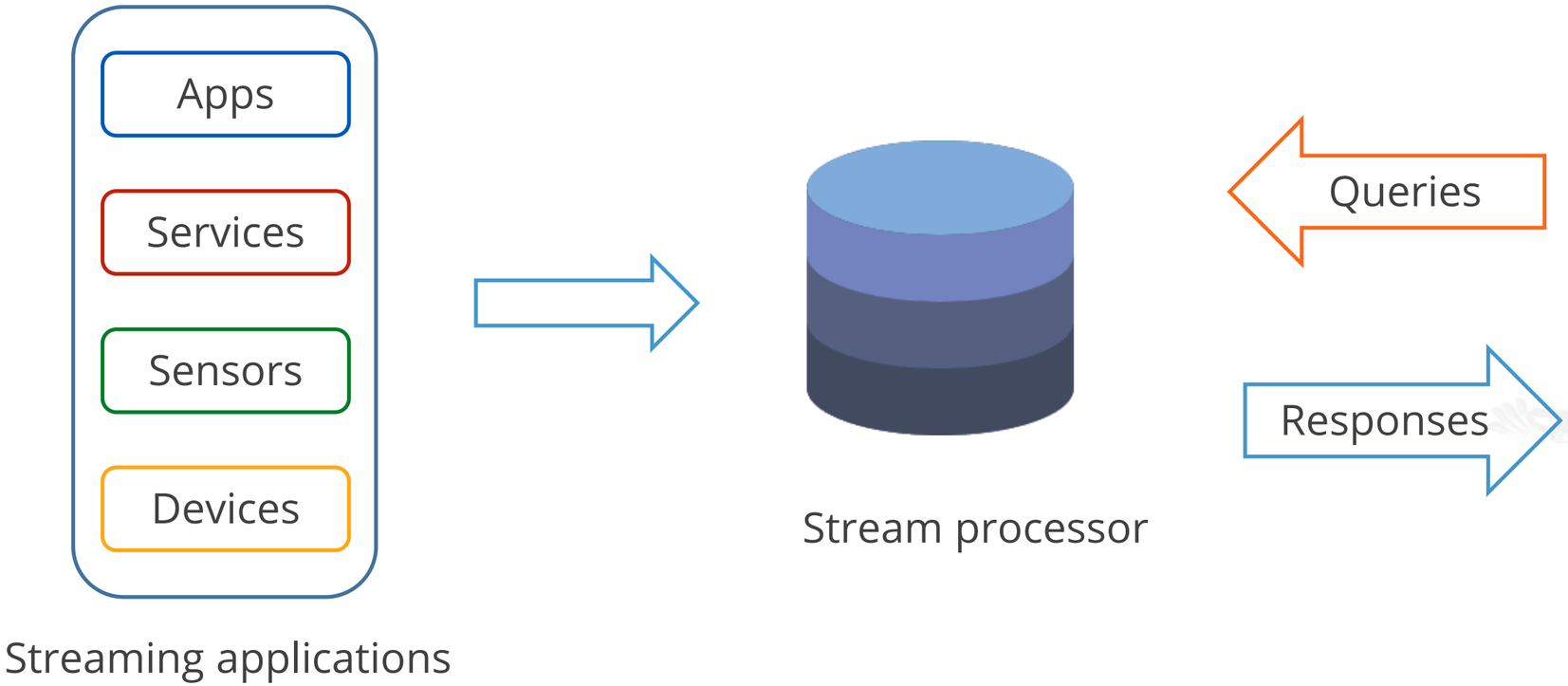
It achieves better resource management by balancing the load across workers.

DATA AND
ARTIFICIAL INTELLIGENCE

Spark Streaming Introduction

What Is Streaming?

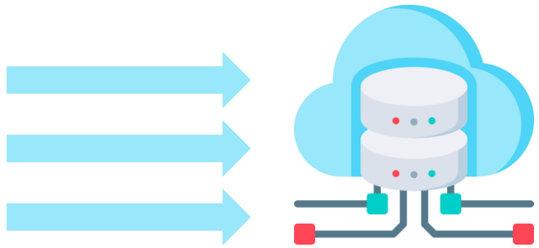
The continuous flow of data at a high-speed rate is known as streaming.



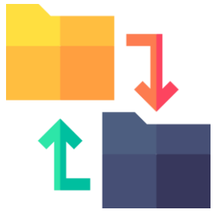
The diagram illustrates the processing of continuous data streams to extract real-time insights.

What Is a Data Stream?

A data stream is an unbounded series of data that arrives continuously.



Data streaming

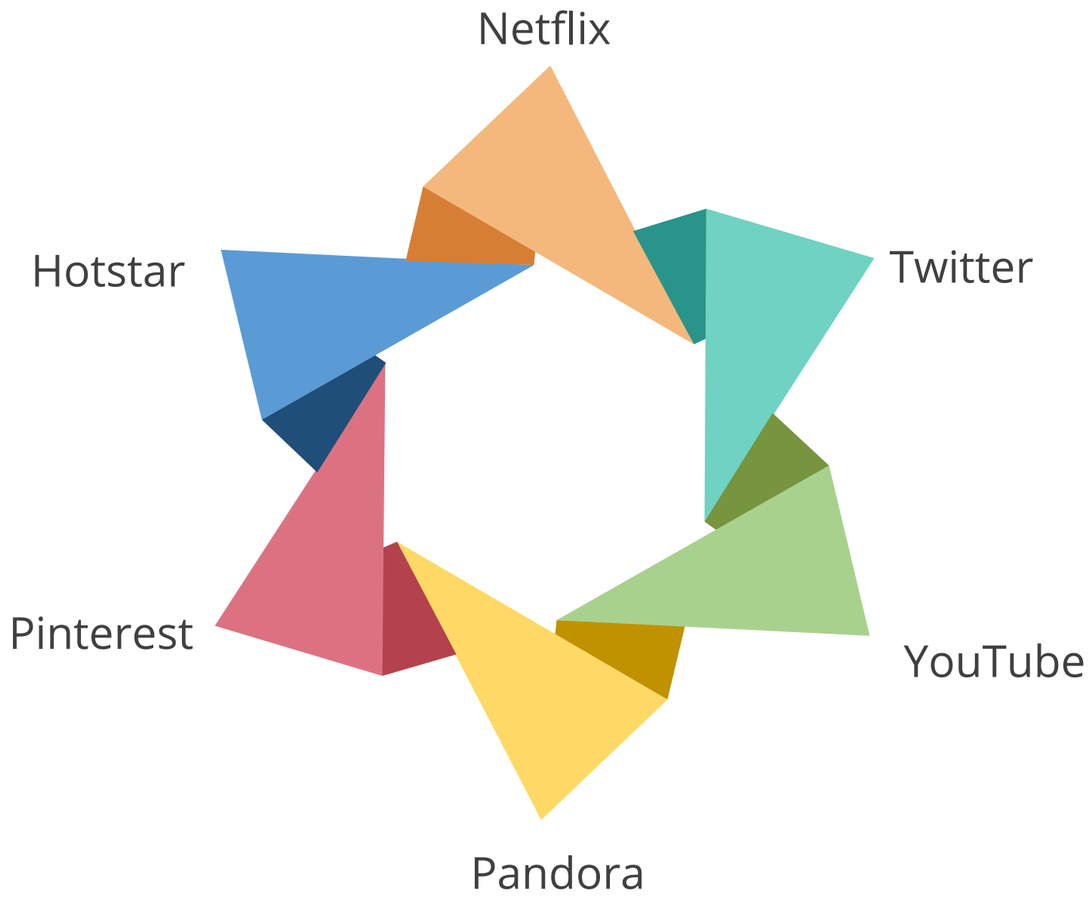


Data transfer

Streaming is the process of transferring data in a continuous stream.

Examples of Streaming

The following are some examples of streaming sources:



Need for Real-Time Processing

Real-time processing of data is extremely beneficial in the following scenarios:

Tsunami warnings

The data should be tracked in real-time because a delay in warning could result in lives being lost.

Google maps

It predicts traffic in real-time, which is useful for determining the most time-efficient route.

Marketing

The advertisements must be properly targeted to the audience as there is a chance for the product to lose its popularity.

Stream Processing Techniques

Stream processing techniques require the use of NoSQL databases to solve problems, such as:

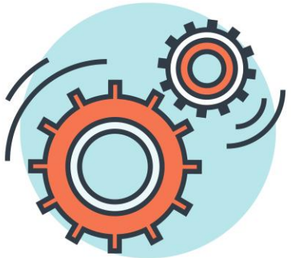
DATA AND
ARTIFICIAL INTELLIGENCE

Real-Time Processing of Big Data

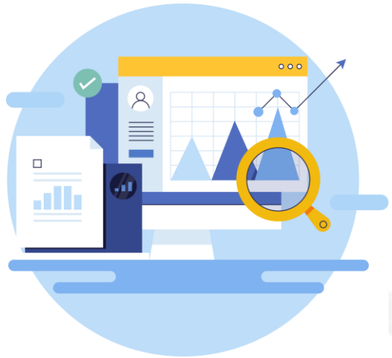
Real-Time Processing of Big Data

Real-time processing consists of:

Continuous input



Processing data

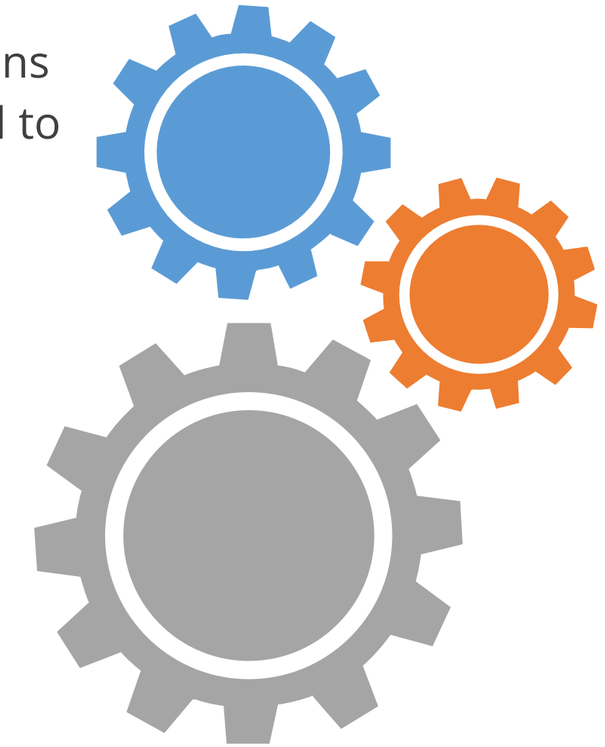


Analysis of reporting data

Real-Time Processing of Big Data

Real-time processing of data has certain characteristics, such as:

It contains a set of repeated operations in which data streams are transferred to memory.

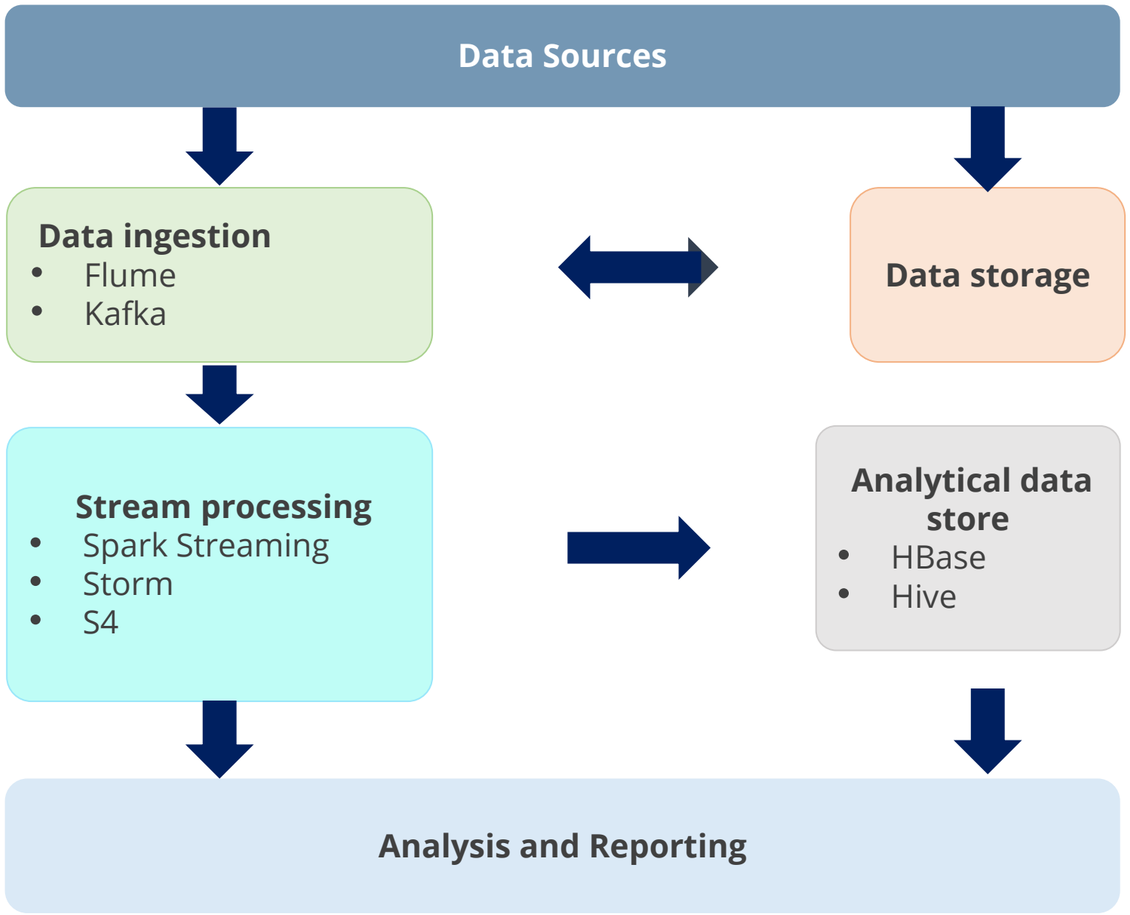


It is essential for automated systems with large data streams and complex data structures to maintain a high level of functionality.

Bank ATMs, radar systems, disaster management systems, the internet of things, and social media are a few examples.

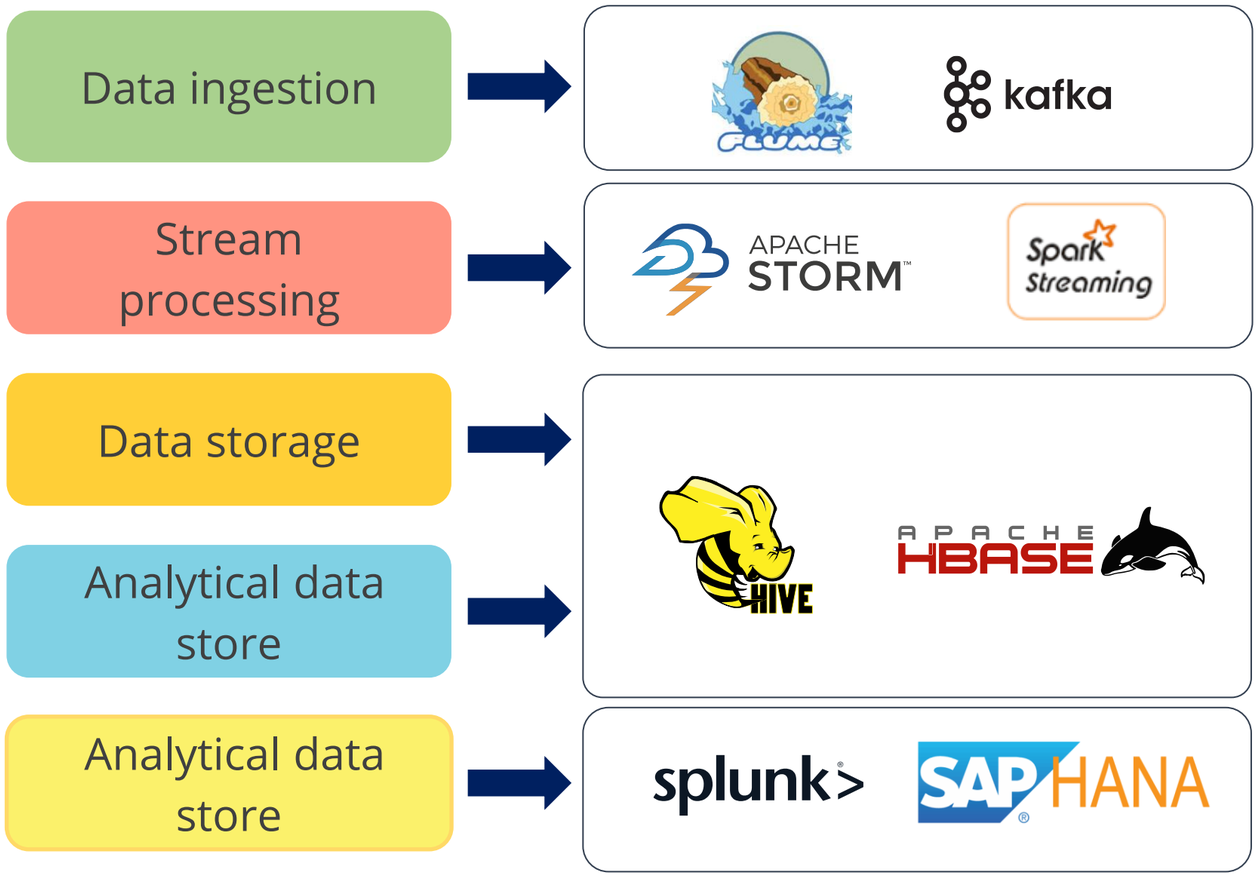
Real-Time Big Data Processing Life Cycle

The following diagram shows the life cycle of real-time big data processing.



Real-Time Big Data Processing Tools

Some of the most common tools for real-time big data processing are:



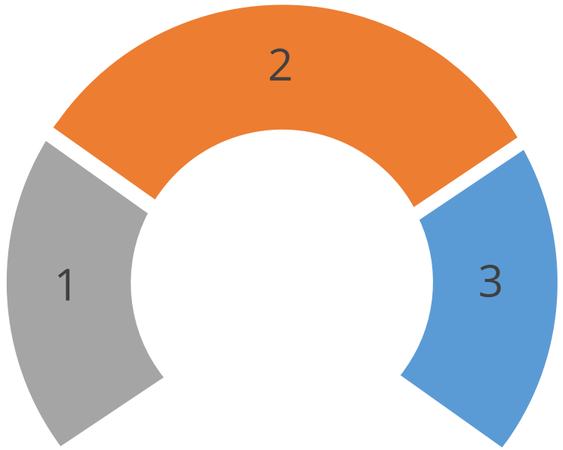
DATA AND
ARTIFICIAL INTELLIGENCE

Data Processing Architecture

Data Processing Architecture

The following are the characteristics of a good architecture for real-time processing:

Supportive of batch and incremental updates

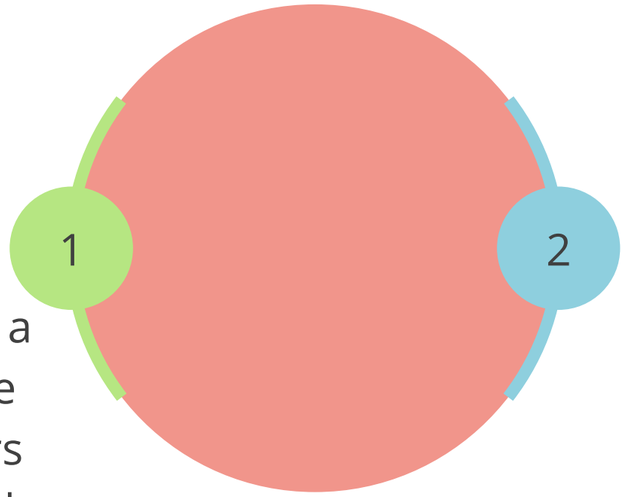


Fault-tolerant and scalable

Extensible

Data Processing Architecture

The two data processing architectures are as follows:



Lambda architecture

The lambda architecture is a method for analyzing large amounts of data that offers both batch-processing and streaming.

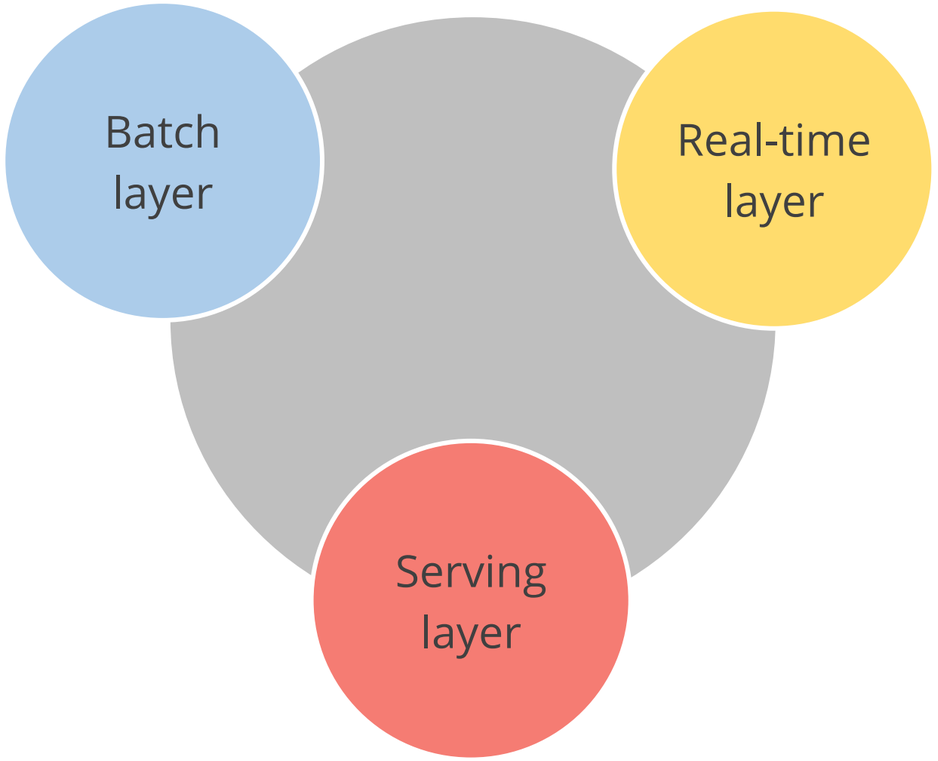
Kappa architecture

Kappa Architecture is a software architecture for processing streaming data.

The Lambda Architecture

The Lambda architecture has three layers:

It saves all the data that enters the system as batch views in preparation for indexing.

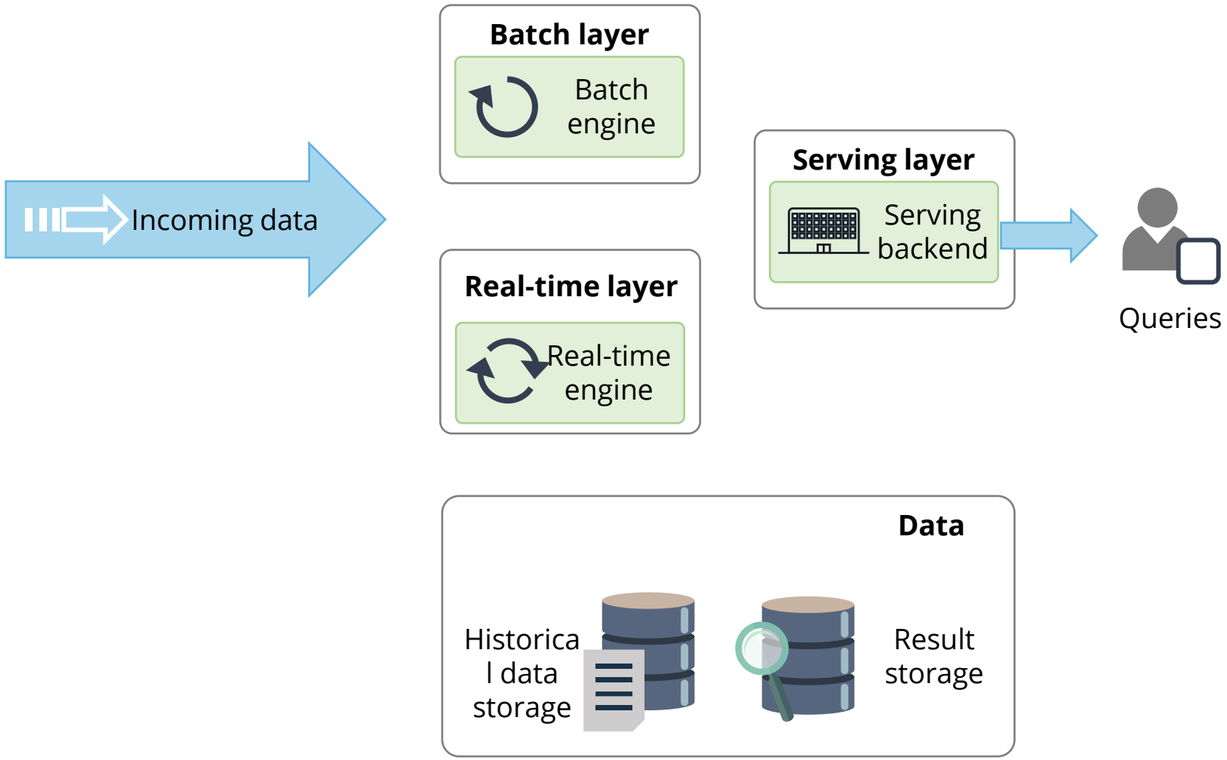


It complements the serving layer by indexing the most recent additions that the serving layer has not yet indexed fully.

It incrementally indexes the latest batch views so that they can be accessed by end-users.

Batch Layer

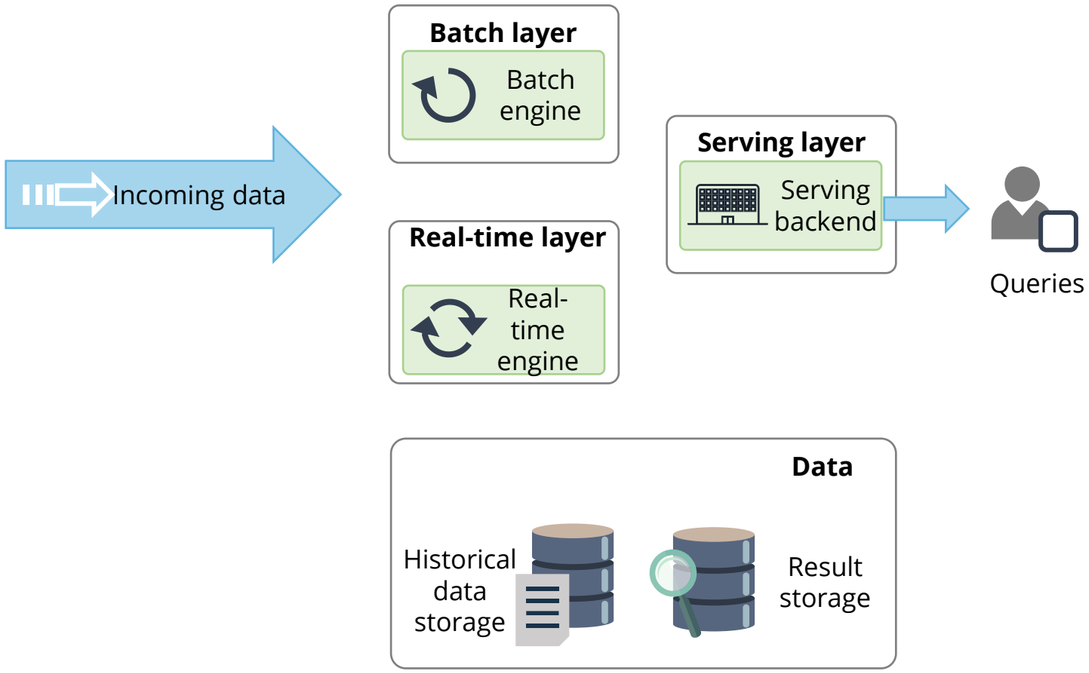
The batch layer has the following features:



- Stores the raw data as it arrives
- Computes the batch views for consumption
- Manages historical data
- Recomputes results
- Operates on full data
- Produces most accurate results
- Has a high cost of high latency

Real-Time Layer

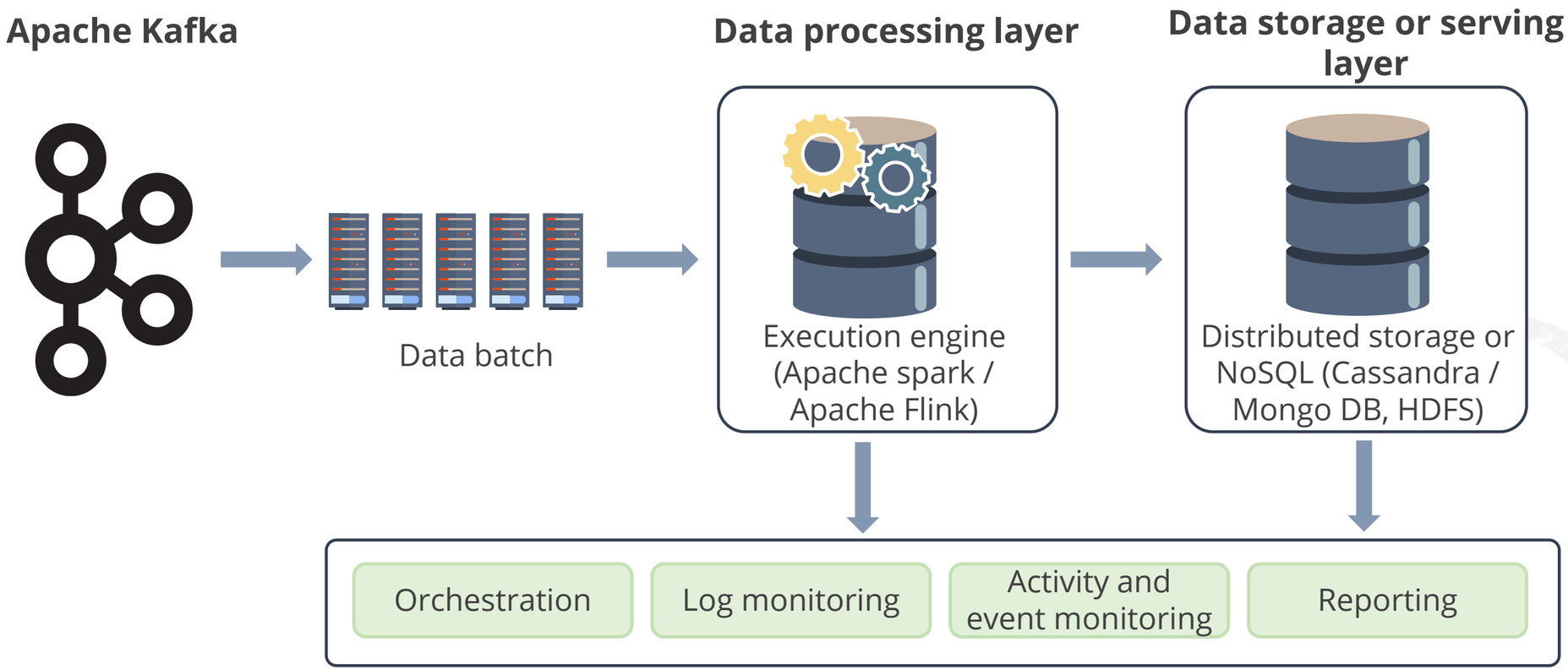
The real-time layer has the following features:



- 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 processes data as a stream.

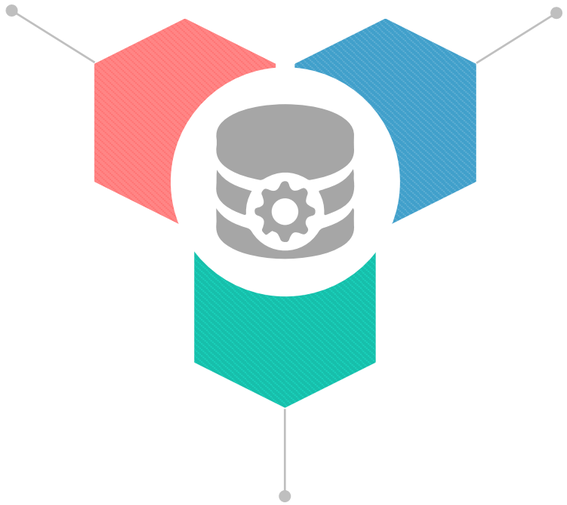


The diagram illustrates the work of Kappa Architecture.

Applications of Kappa Architecture

The Kappa architecture can be deployed for data processing enterprise models in which:

Multiple data events or queries are queued and catered against a distributed storage system.



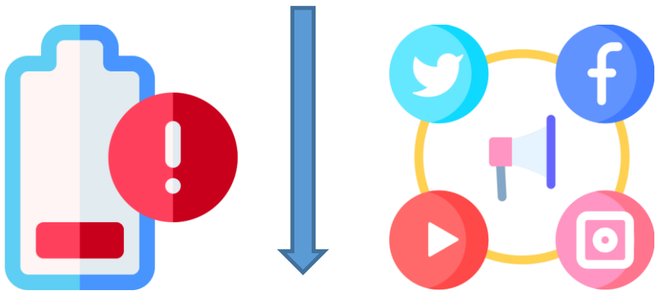
There is no predetermined order for the events and queries, and the platforms can interact with the databases at any time.

Replication is effective as terabytes of storage are required for each node of the system.

Use Case

Problem statement: Twitter wants to improve the mobile experience for its users. It wants to eliminate a complex mobile system that receives events, archives them, performs real-time computations, and combines the results into coherent information.

Goal



Reduce battery and network usage

Ensure data reliability

Solution: Analytics events can be compressed, and data can be sent in batches to reduce the impact on the device.

Use Case

Process: Twitter utilized the Lambda architecture to achieve an improved mobile experience for its users. The architecture consists of the following components:

Result: It has enabled app developers to get reliable, real-time, and actionable insights into their mobile applications.

DATA AND
ARTIFICIAL INTELLIGENCE

Spark Streaming

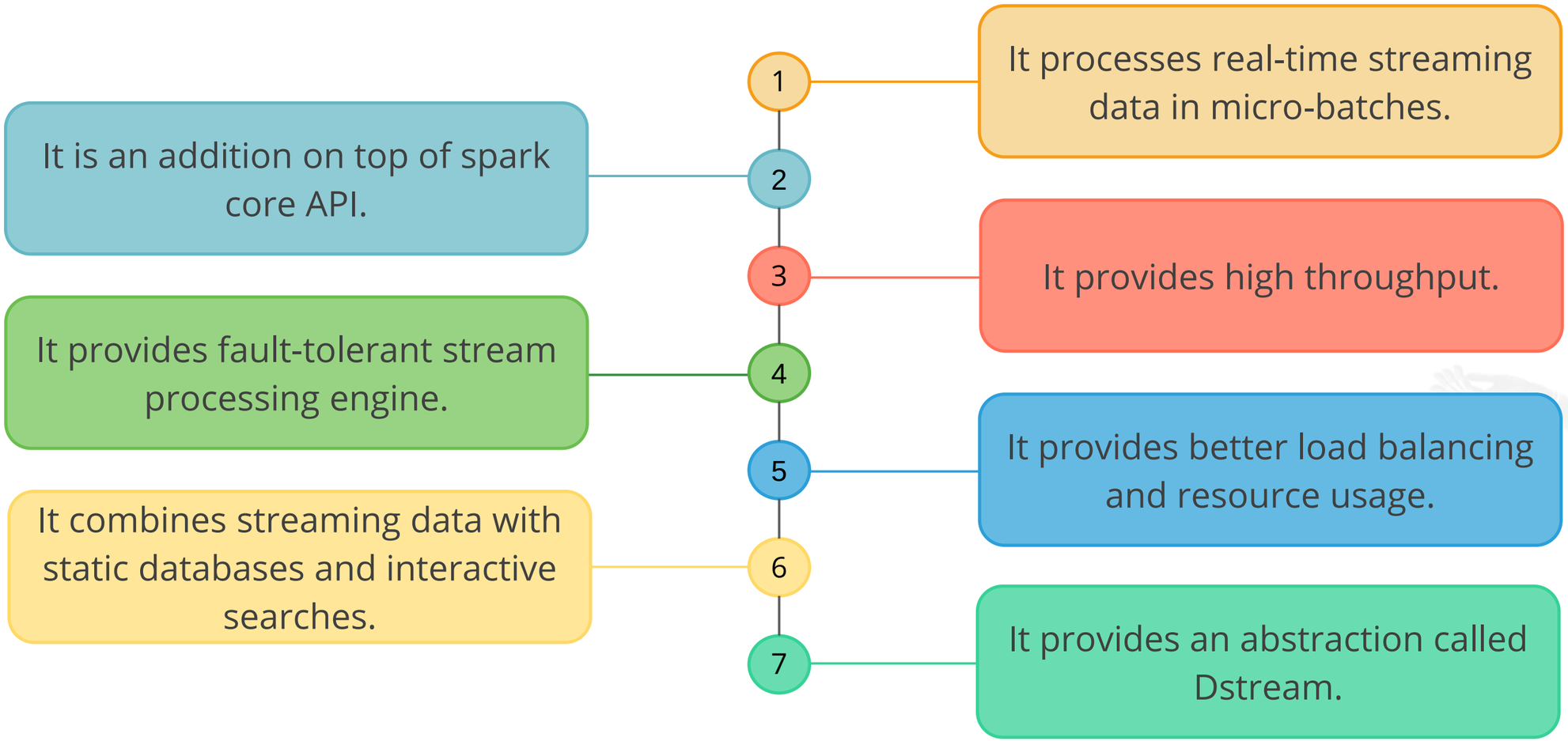
Introduction to Spark Streaming

Spark Streaming is an extension of the core Spark API.

The diagram illustrates the workflow of Spark streaming.

Spark Streaming Overview

Spark streaming offers the following features:



Features of Spark Streaming

Some more features of Spark streaming are:

Scaling	It scales to hundreds of nodes with ease.
Speed	It achieves low latency.
Fault Tolerance	It can efficiently recover from failures.
Integration	It works with both batch and real-time data.
Business Analysis	It is used to track customer behavior for business insights.

Features of Spark Streaming

Extensive Support

- It has algorithms for machine learning and graph processing.
- Scala, Java, and Python

Discretized Stream (DStream)

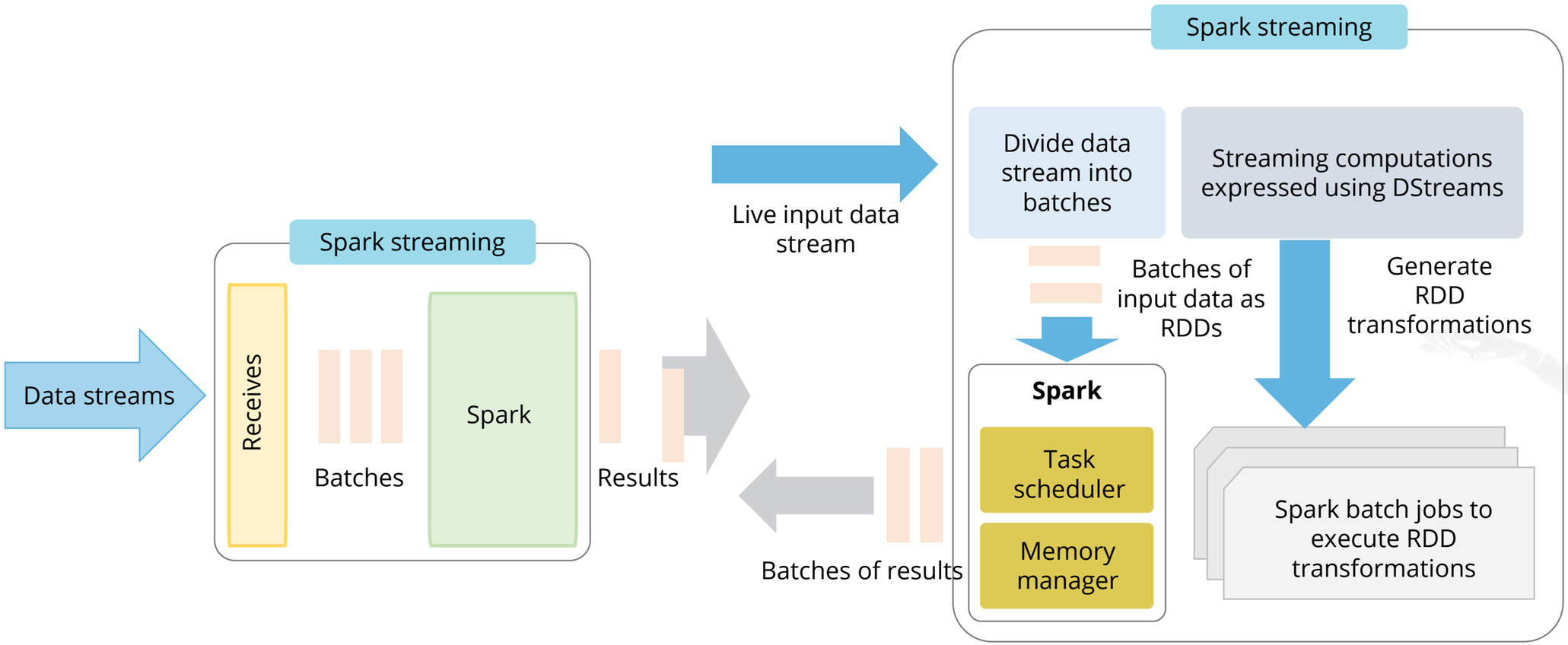
- Flume, Kafka, and Kinesis

ZooKeeper and HDFS

- It is used for State storage and leader election support.

Workflow of Spark Streaming

Spark Streaming is an extension of the Spark API that enables scalable, high-throughput, fault-tolerant live data stream processing.



The diagram illustrates the working of spark streaming.

Streaming Word Count program

The following steps count the number of words in a program:

Step 1: Import required packages

training@localhost:~

```
import random
import time
from socket import *
from threading import Thread

import pyspark
from pyspark.streaming import StreamingContext
```

Streaming Word Count program

Step 2: Create a SparkContext with two threads in local mode

training@localhost:~

```
sc = pyspark.SparkContext("local[2]")
```

Streaming Word Count program

Step 3: Create a class streamer that reads from the specified socket and performs the word count

training@localhost:~

```
class Streamer(Thread):
 def __init__(self, sc):
 Thread.__init__(self)
 self.sc = sc

 def run(self):
 print("starting Streaming thread...")

 # Using the spark context, create a streaming context with a batch interval of 1 second
 ssc = StreamingContext(self.sc, 1)

 # Create a socket DStream reading from localhost at port 9999
 socketDstream = ssc.socketTextStream("localhost", 9999)

 # WordCount
 wordcounts = socketDstream.flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda a, b: a + b)

 # Print first 50 words counted in the last one second
 wordcounts.pprint(50)

 # Start the execution
 ssc.start()

 # Stop the streaming context after 35 seconds
 time.sleep(35)
 ssc.stop()
```

Streaming Word Count program

Step 4: Create a class tweeter that writes to a specified socket

training@localhost:~

```
class Tweeter(Thread):
 def run(self):
 print('starting Socket writing thread')
 HOST = 'localhost'
 PORT = 4444
 ADDR = (HOST, PORT)
 # Create an INET, Streaming socket
 serversocket = socket(AF_INET, SOCK_STREAM)
 # Bind the socket to localhost and port 4444
 serversocket.bind(ADDR)
 # Become a server socket
 serversocket.listen(SOMAXCONN)
 # Accept connections
 cSocket, addr = serversocket.accept()
 print("connection ready.Sending data")

 cSocket.send("When you and your best friend.\n\r".encode())
 time.sleep(random.randint(0, 3))
 cSocket.send("Behind every strong woman.\n\r".encode())
 time.sleep(random.randint(0, 3))
 cSocket.send("When you and your best friend.\n\r".encode())
 time.sleep(random.randint(0, 3))
 cSocket.send("Mushrooms are rich in carbohydrates and proteins.\n\r".encode())
 time.sleep(random.randint(0, 3))
 cSocket.send("Its distribution encompasses Asia.\n\r".encode())
 time.sleep(random.randint(0, 3))
 cSocket.send("The cap cuticle is thin and can be easily peeled off .\n\r".encode())
 print("Tweeting DONE :)")
```

Streaming Word Count program

Step 5: Start the streamer and tweeter thread

training@localhost:~

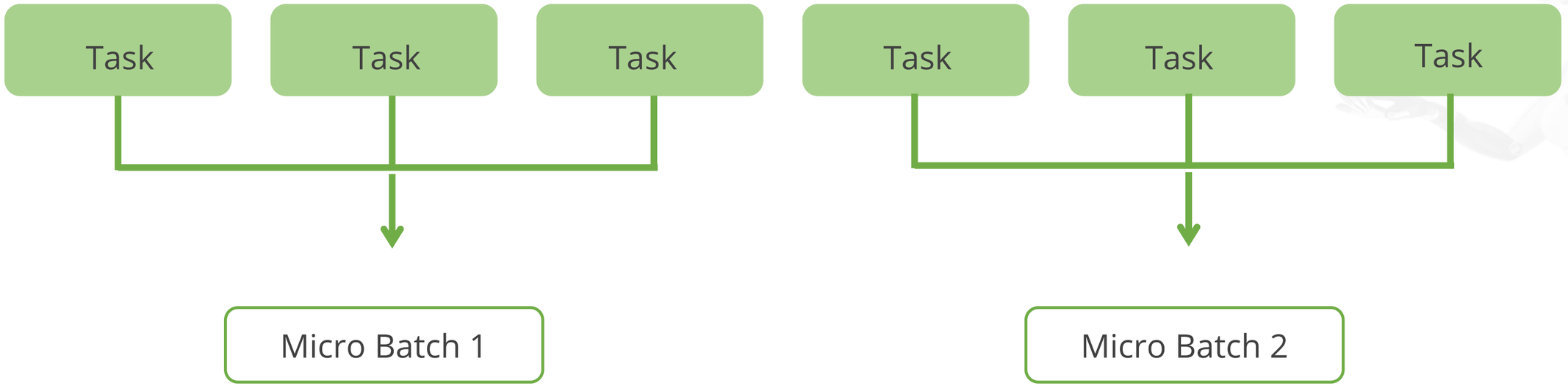
```
Streamer(sc).start()

Tweeter().start()
```

Output:

```
-------------------------------------------
Time: 2022-03-03 19:34:05
-------------------------------------------
('When', 1)
('best', 1)
('start', 1)
('talking', 1)
('love', 1)
('', 1)
('you', 1)
```

Micro Batch

A micro-batch process handles streams as a sequence containing chunks of data by assigning each task to a specific job or process.

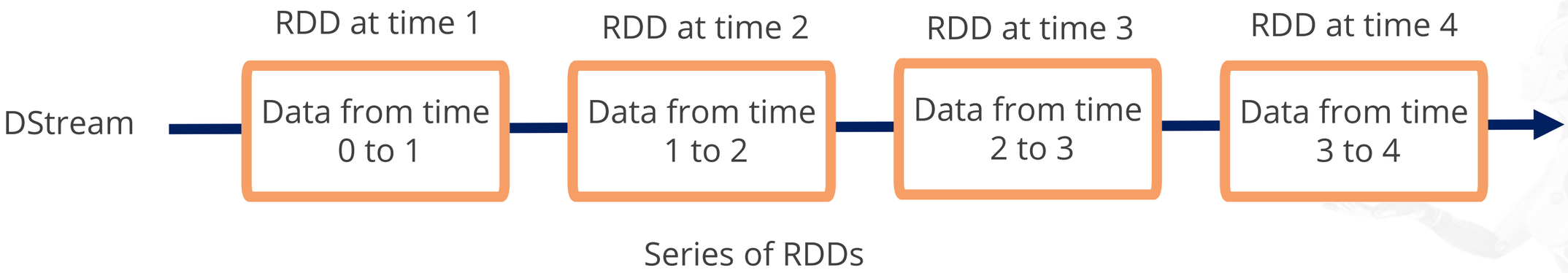


DATA AND
ARTIFICIAL INTELLIGENCE

Introduction to DStreams

Introduction to DStreams

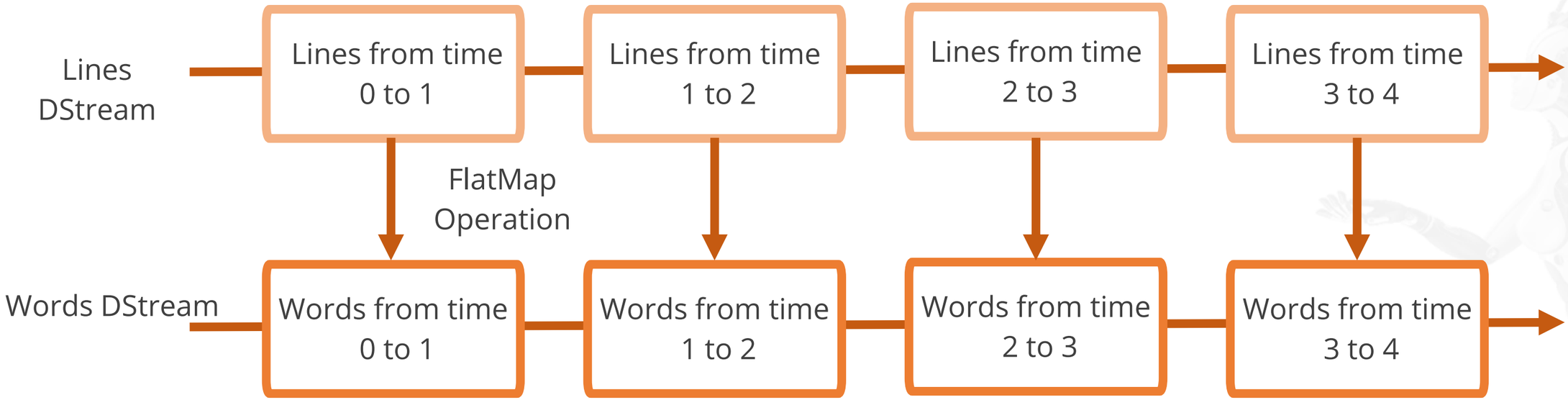
The fundamental abstraction in spark streaming is the Discretized Stream (DStream).



A series of RDDs characterizes it.

Introduction to DStreams

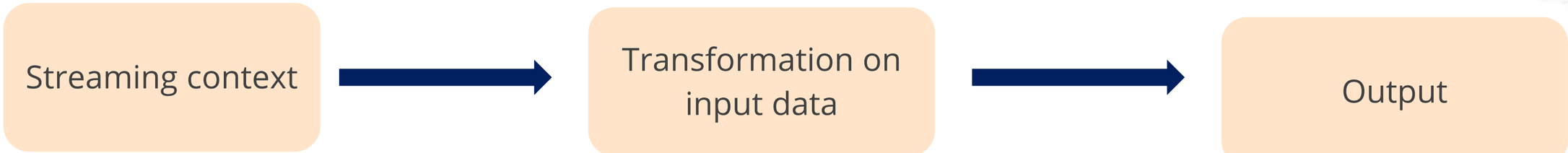
The operations applied to a DStream are translated into those applicable to the underlying RDDs.



The diagram illustrates the workflow of DStreams.

Input DStreams and Receivers

The input data streams represent the streams of data received from streaming sources.



Streaming Context

Streaming context is the main entry point of streaming functionality.

It provides implementation for various streaming sources like:

- Twitter
- Akka Actor
- ZeroMQ

Twitter real-time data is pulled from an API and then processed by Apache-Spark.

Akka actor provides a higher level of abstraction for writing concurrent and distributed applications.

The ZeroMQ message queue library allows picking a queue type flexibly.

Streaming Context

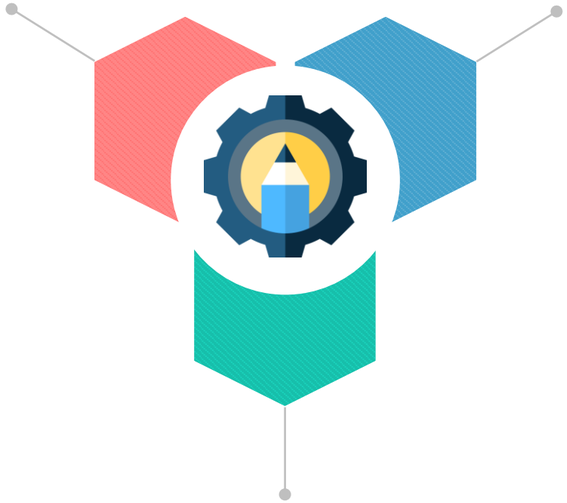
The features of a streaming context are as follows:

- Consumes a stream of data
- Registers an input DStream to produce receiver objects
- Provides methods used to create DStream from various input sources

Streaming Context: Initialization

Streaming Context can be created in the following ways:

By providing spark master URL and app name



By using SparkConf objects

Based on the existing SparkContext

Transformations on DStreams

Transformations on DStreams are similar to those of RDDs.

The map function

It returns a new DStream by passing each element of the source DStream through the function.

Syntax: map(f, preservesPartitioning=False)

The flatMap function

The input items are mapped to zero or more output items, similar to the map.

Syntax: flatMap(f, preservesPartitioning=False)

The filter function

It returns a new DStream by selecting only the records of the source DStream on which the function returns true.

Syntax: filter(condition)

Transformations on DStreams

Repartition	Union	count()
It changes the level of parallelism in the DStream by creating more or fewer partitions. **Syntax:** c = b.rdd.repartition(5) c.getNumPartitions()	It returns a new DStream that contains the union of the elements in the source DStream and other DStream. **Syntax:** dataFrame1.union(dataFrame2)	It returns a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream. **Syntax:** b.count()

Transformations on DStreams

reduce

It returns a new DStream of single-element RDDs by aggregating the elements in every RDD of the source DStream using a function func.

Syntax: def reduce(f: (T, T) => T): T

countByValue

It returns a new DStream of pairs where the value of each key is its frequency in each RDD of the source DStream.

Syntax: words.countByValue()

reduceByKey

It returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function.

Syntax: reduceByKey(func, numPartitions=None, partitionFunc=<function portable_hash>)

Transformations on DStreams

join

It returns a DStream of pairs (K, (V, W)) with all elements for each key when being called on two DStreams of (K, V) and (K, W) pairs.

Syntax: join(self, other, on=None, how=None)

cogroup

It returns a new DStream of (K, Seq[V], Seq[W]) tuples when called on (K, V) and (K, W) pairs.

Syntax: val result = myrdd1.cogroup(myrdd2)

transform

It returns a new DStream by applying an RDD-to-RDD function to every RDD of the source DStream.

Syntax: map(f, preservesPartitioning=False)

Transformations on DStreams

updateStateByKey

It returns a new “state” DStream where the state for each key is updated by applying the given function to the previous state and new values for the key.

Syntax:

DStream.updateStateByKey(updateFunc, numPartitions=None, initialRDD=None)

Output Operations on DStreams

DStreams data can be pushed to external systems using output methods.

- print()
- saveAsTextFiles(prefix, [suffix])
- saveAsObjectFiles(prefix, [suffix])
- saveAsObjectFiles(prefix, [suffix])
- saveAsObjectFiles(prefix, [suffix])

Design Patterns for Using foreachRDD

DStream.foreachRDD is a powerful primitive for transmitting data to other systems.

Example:

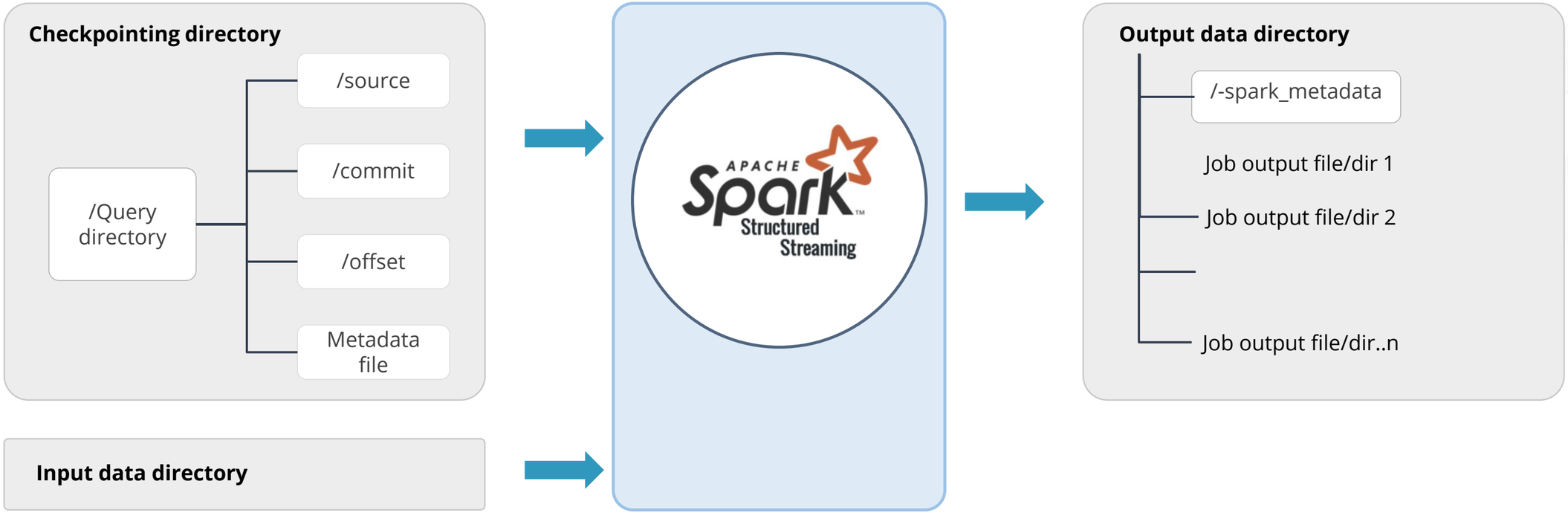
```
DStream.foreachRDD { rdd => val connection = createNewConnection()
// executed at the driver
rdd.foreach { record => connection.send(record)
// executed at the worker } }
```

DATA AND
ARTIFICIAL INTELLIGENCE

Checkpointing

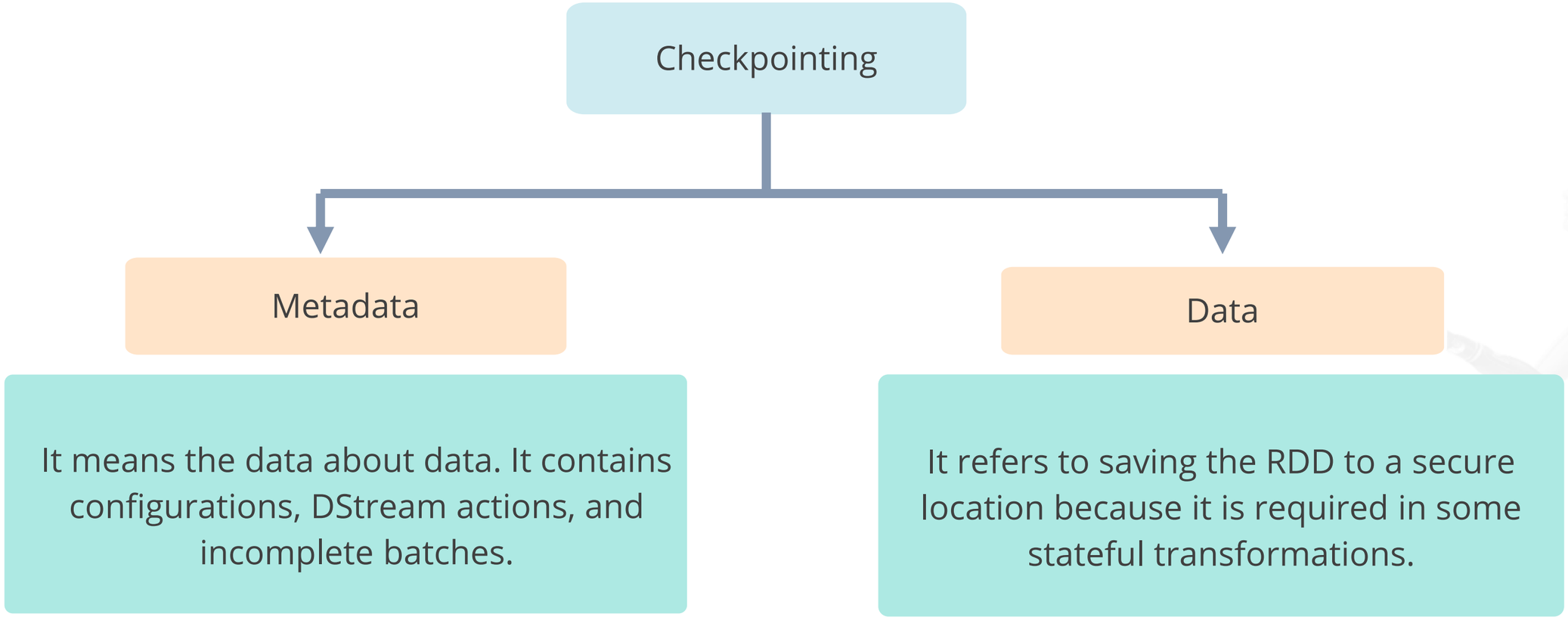
Checkpointing

The checkpointing process truncates the RDD lineage graph and saves the application state timely to reliable storage (HDFS.)



Types of Checkpointing

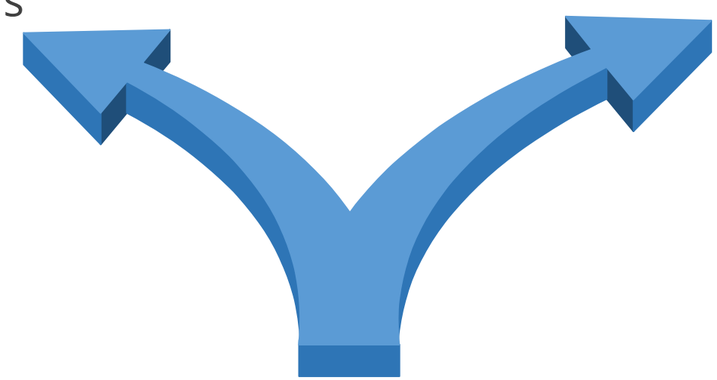
There are two types of data for which checkpointing is used:



Enabling Checkpoints

The following are the requirements to enable checkpointing:

Use of stateful transformations



Recover from failures of the driver running the applications

Types of Stream

There are two types of stream:

File stream

Socket stream

Socket Stream

A socket is built on the driver's machine that provides bidirectional, dependable, sequential, and unduplicated data flow with no record boundaries.

Example:

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

File Stream

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

Example:

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

DATA AND
ARTIFICIAL INTELLIGENCE

State Operations

State Management

The two primary conditions in state management are:

Stateless

The service is in a stateless state when it is active but not processing anything.

Stateful

A stateful service is actively processing and retaining state data.

State Operations

Three stateful operations are as follows:

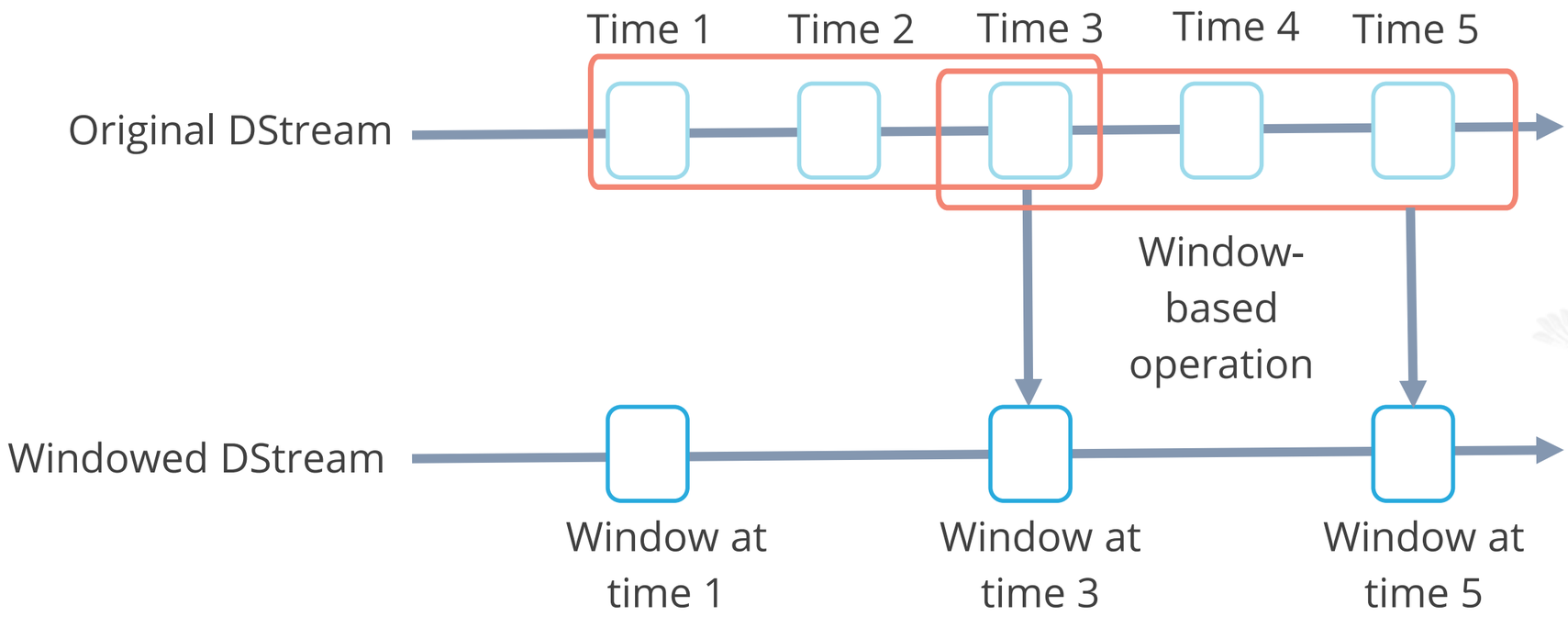
01	It operates on various data batches.
02	It includes all window-based activities and the updateStateByKey operation.
03	It is dependent on previous data batches.

DATA AND
ARTIFICIAL INTELLIGENCE

Windowing Operation

Window Operations

Window operations implement transformations over a sliding window of data.



The diagram shows the working of window operations.

Window Operations

The following example generates word counts over the previous 30 seconds of data every 10 seconds.

Example:

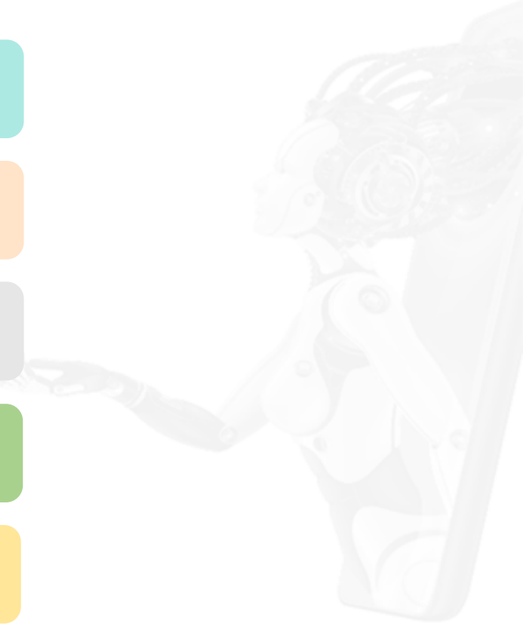
```
windowedWordCounts =
pairs.reduceByKeyAndWindow(lambda a,b: (a + b),
Seconds(30), Seconds(10)))
```


It uses the operation reduceByKeyAndWindow.

Types of Window Operations

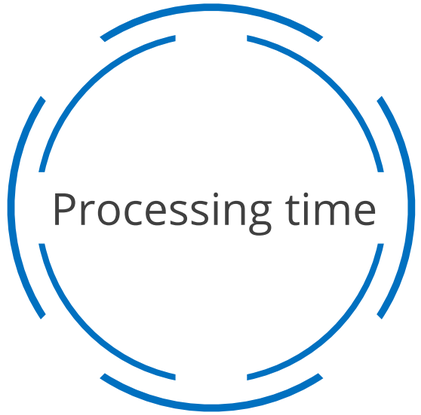
The following operations use window length and slide interval as parameters:

- 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])

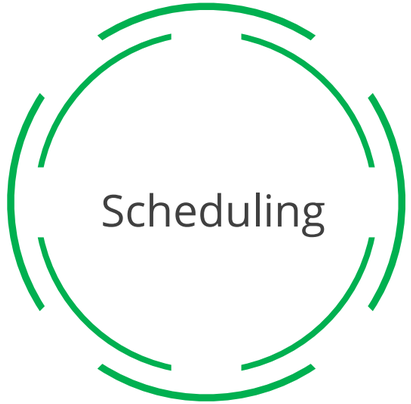


Monitoring Spark Streaming Applications

The key parameters to monitoring the spark streaming application are:



The amount of time it takes to process every data batch



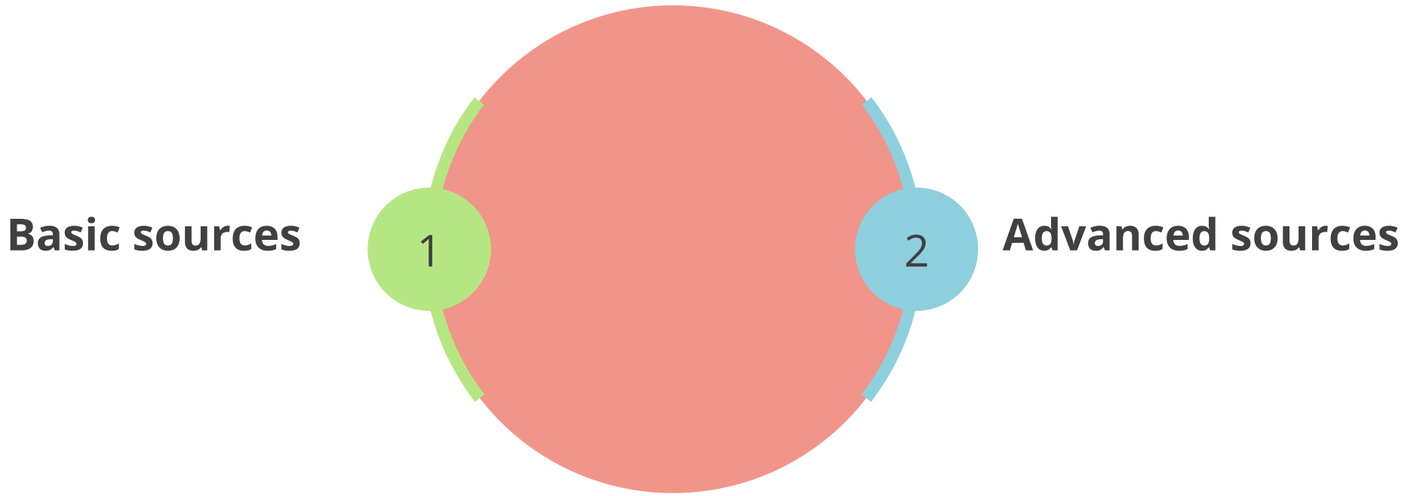
The amount of time a batch spends in a queue waiting for earlier batches to finish processing

DATA AND
ARTIFICIAL INTELLIGENCE

Spark Streaming Sources

Types of Spark Streaming Sources

There are two types of sources in spark streaming:



Basic Sources

Spark streaming monitors and processes all files created in the data directory for basic sources.

Streams based on custom actors

Queue of RDDs as a stream

Categories

Basic Sources

The syntax for basic sources is given below:

Syntax:

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

Advanced Sources

Some of the advanced streaming sources include:

Twitter

Apache Kafka

Apache Flume

Kinesis

Advanced Sources: Twitter

Follow the instructions below to construct a DStream using data from Twitter's stream of tweets:

Linking

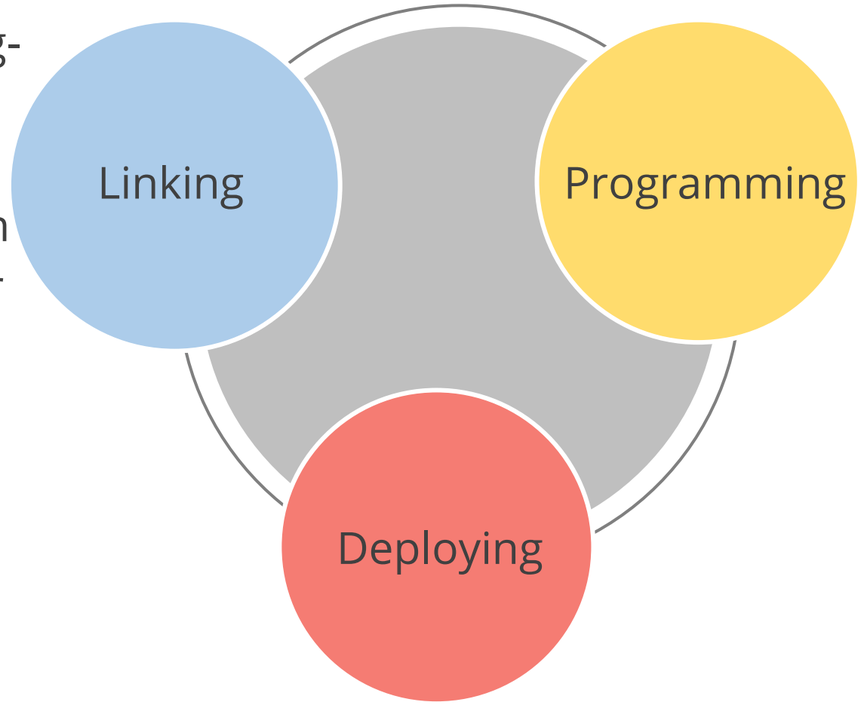
Programming

Deploying

Advanced Sources: Twitter

The three steps to construct a DStream are explained below:

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



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

An uber JAR needs to be generated with all dependencies

Assisted Practice: Apache Spark Streaming

Duration: 15 mins

Problem Statement: Create a real-time streaming application with the data provided to see the streaming output at different timestamps

Objective: In this demonstration, you will learn how to create a real-time Spark streaming application.

Tasks to Perform:

Step 1: Create a Streaming context
Step 2: Create a text socket stream; the method socketTextStream returns a DStream object which represents a deserialized stream
Step 3: Use the flatMap() method on DStream to break the line into words
Step 4: Convert each word into a tuple containing (word,1)
Step 5: Create a word count program using reduceByKey

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

ASSISTED PRACTICE

Key Takeaways

- Big data streaming involves processing continuous streams of data in order to extract real-time insights.
- The components of real-time processing involve continuous input, processing data, and analysis of reporting data.
- Window operations let you implement transformations over a sliding window of data.
- Discretized Stream (DStream) is the fundamental abstraction available in spark streaming.
- Spark Streaming is an extension of the core Spark API.

DATA AND
ARTIFICIAL INTELLIGENCE

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 Twitter, Kafka, and Flume 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.

Lesson-End Project

Retail Business Analysis Using Spark Streaming

Problem Statement: An organization is collecting the IoT (Internet of Things) device data to perform analysis. The data contains information about the actions performed by a user on an IoT device, such as “power off” and “power on.”

Objective:
The objective is to analyze the data that is coming from the IoT devices to perform data analysis.

Lesson-End Project

Tasks to be performed:

Step 1: Create a Python script in Webconsole to perform analytics
Step 2: In the Python script:
2.1 Create a class to write log messages to a host with port
2.2 Create a class that creates a streaming context, reads data from the socket, and creates a word count program
Step 3: Execute the Python script using the spark-submit command

DATA AND
ARTIFICIAL INTELLIGENCE

Thank You

simplilearn