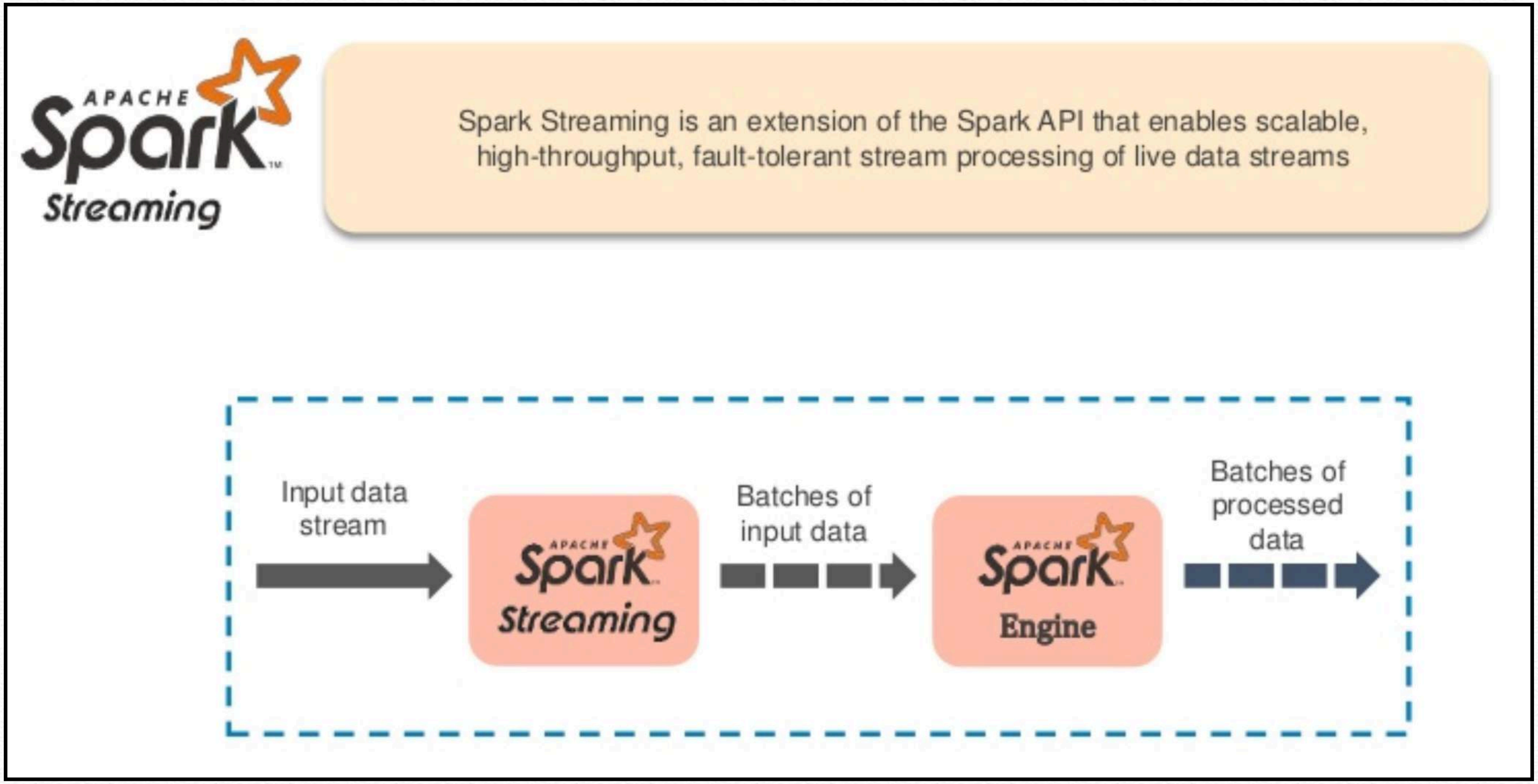
Spark

Lightning-Fast Cluster Computing

Spark Streaming

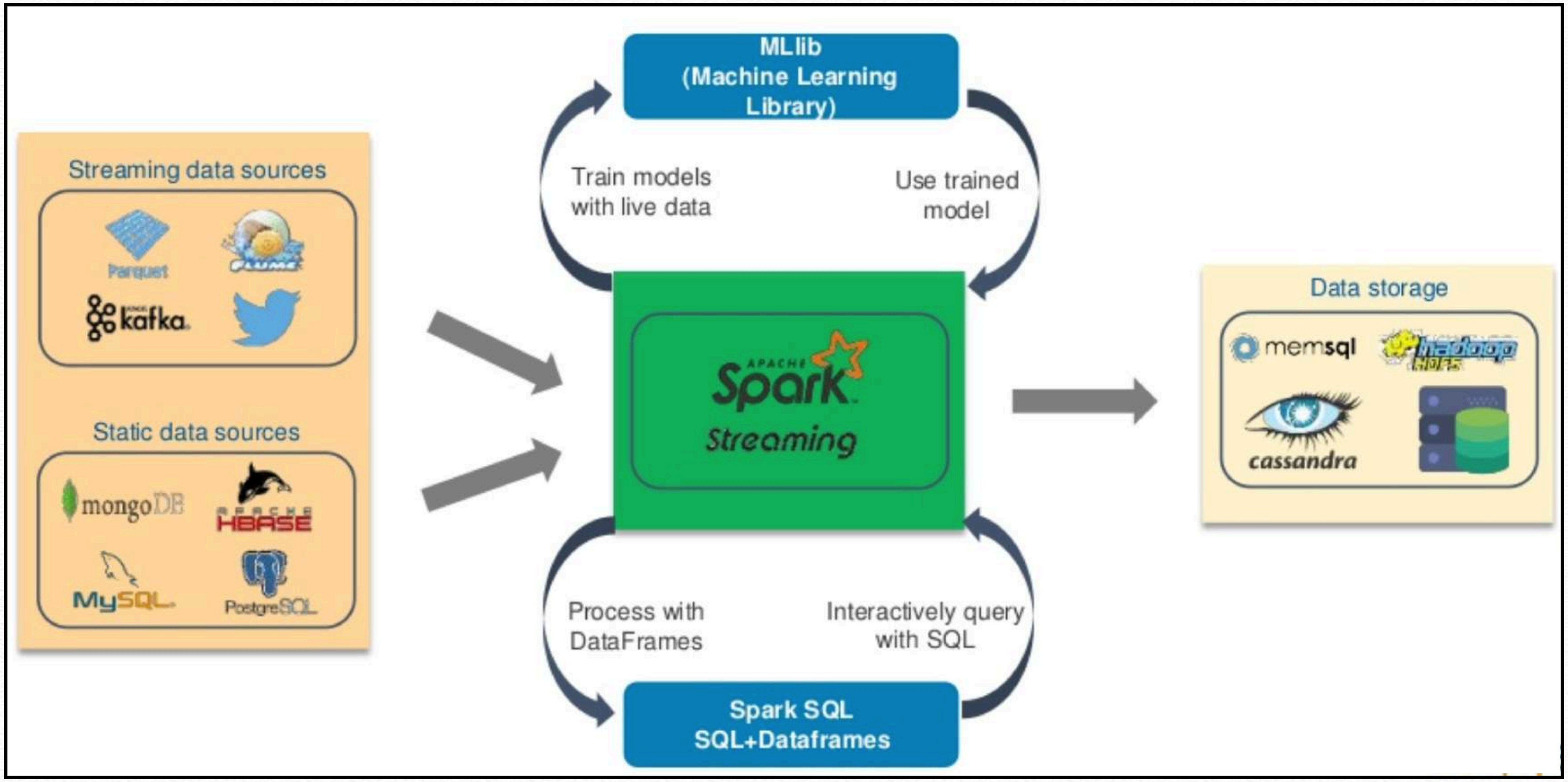
04/05/2020 - Big Data

What is Spark Streaming

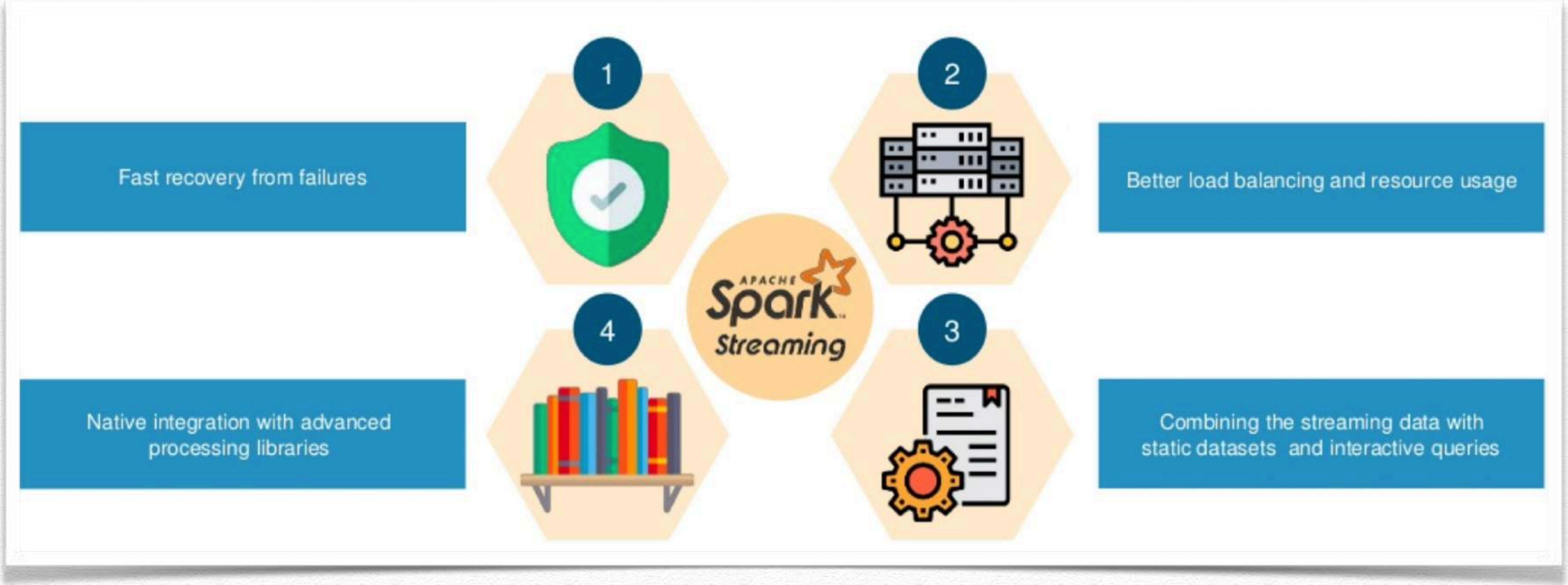


Spark Streaming is an extension of the Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams

What is

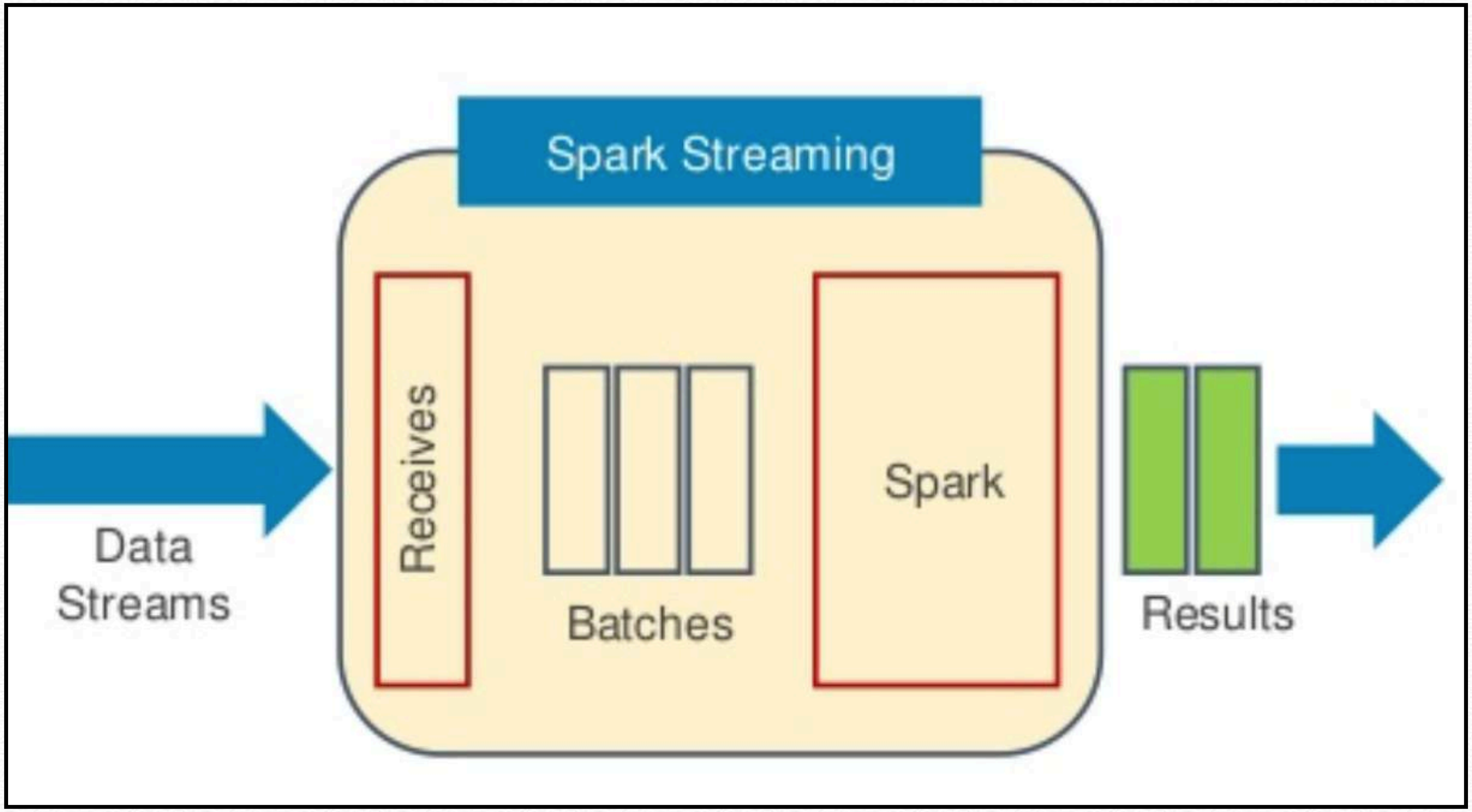

MLlib
(Machine Learning Library)
Streaming data sources
Parquet
Flume
kafka
Static data sources
mongoDB
Apache HBase
MySQL
PostgreSQL
Train models with live data
Use trained model
Apache Spark Streaming
Data storage
memsql
hadoop HDFS
cassandra
Process with DataFrames
Interactively query with SQL
Spark SQL
SQL+Dataframes

Features

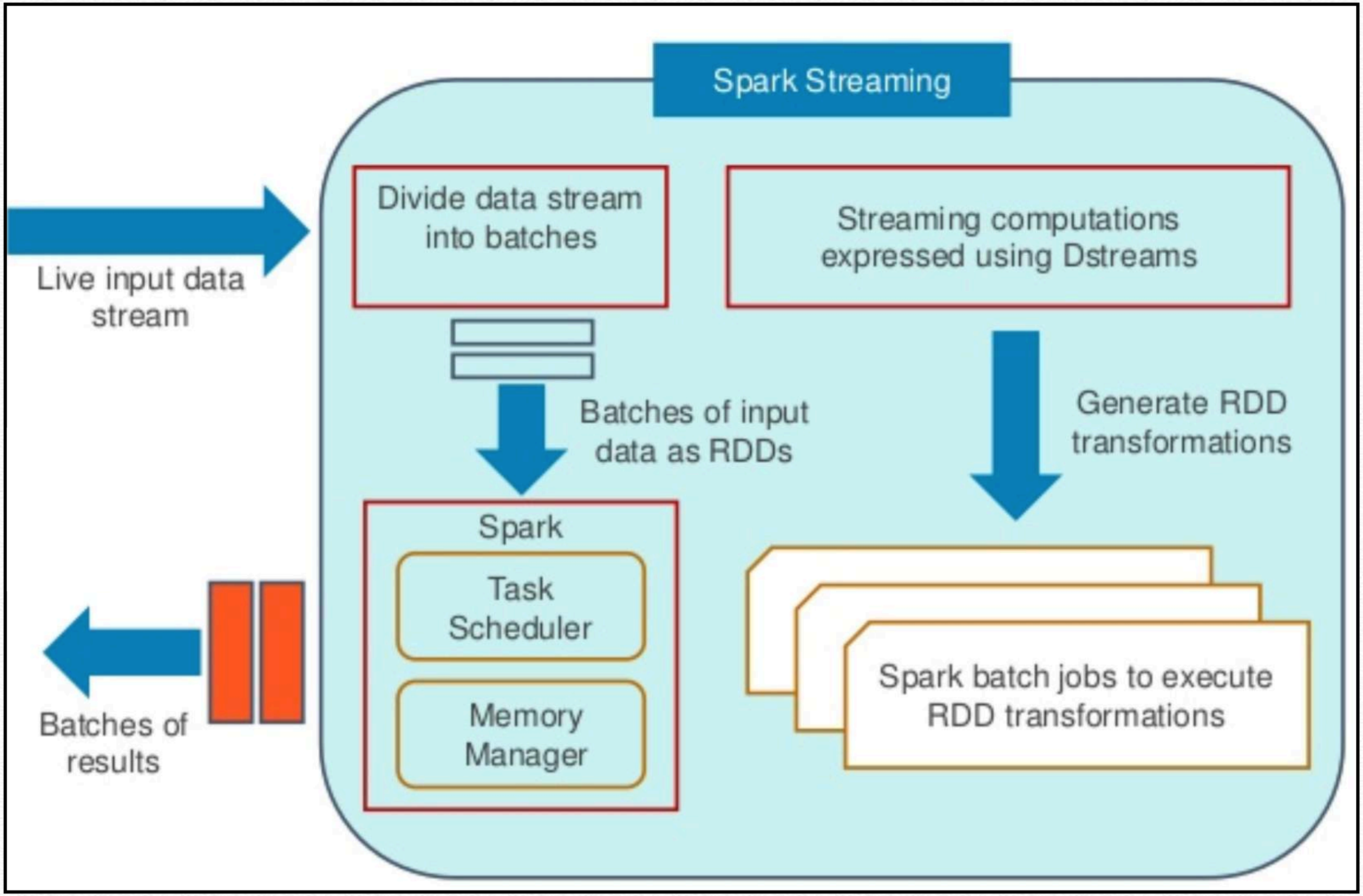


1
2
Fast recovery from failures
Better load balancing and resource usage
APACHE Spark Streaming
4
3
Native integration with advanced processing libraries
Combining the streaming data with static datasets and interactive queries

Working

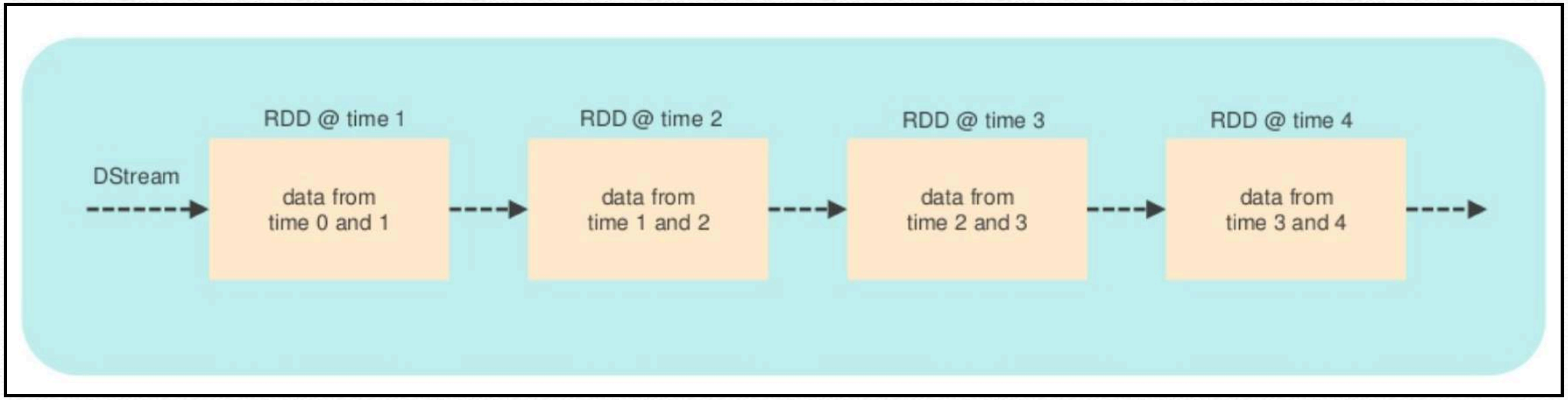

Spark Streaming
Data Streams
Receives
Batches
Spark
Results

Working

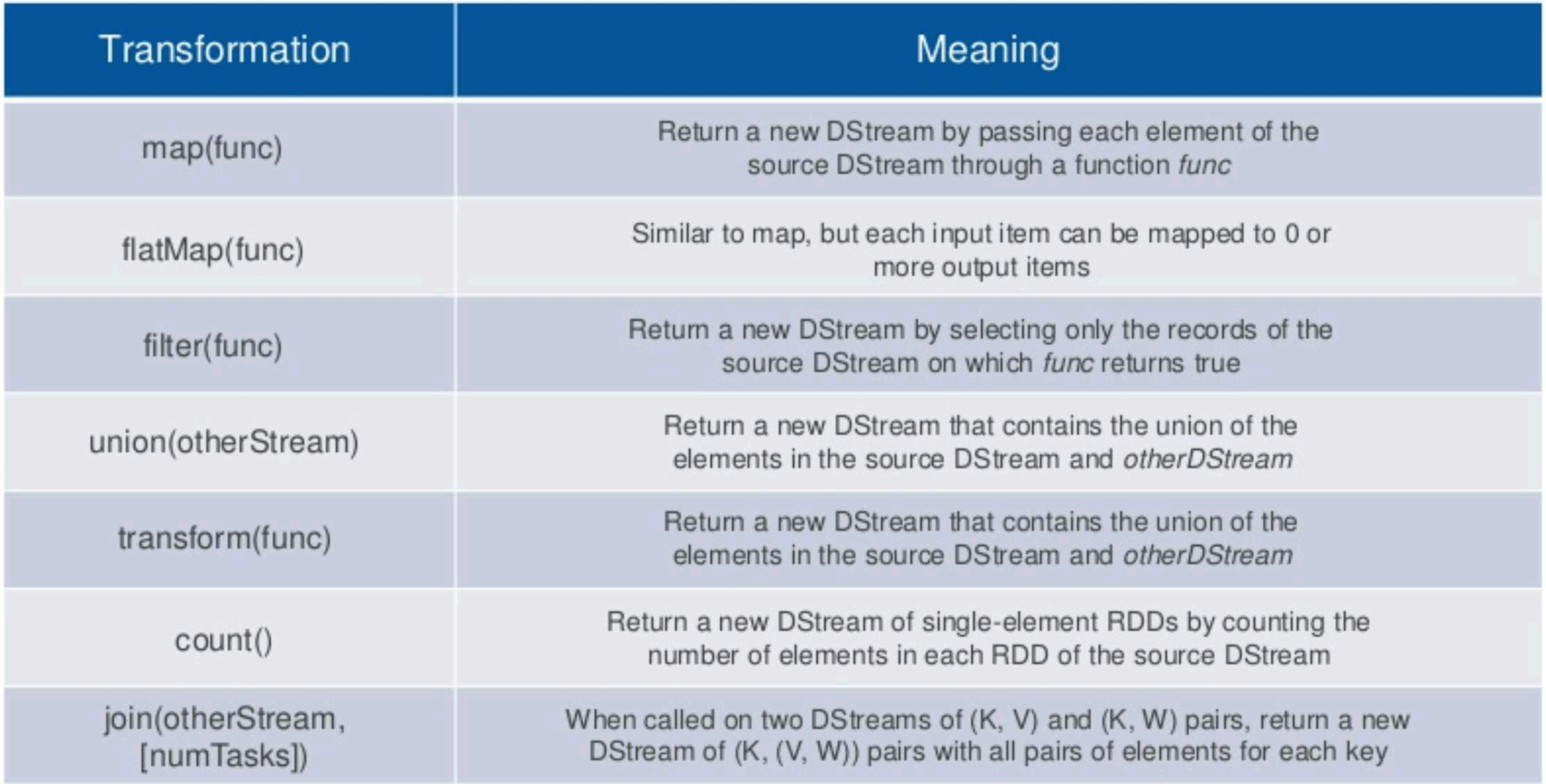

Spark Streaming
Live input data stream
Divide data stream into batches
Streaming computations expressed using Dstreams
Batches of input data as RDDs
Generate RDD transformations
Spark
Task Scheduler
Memory Manager
Spark batch jobs to execute RDD transformations
Batches of results

Discretized Streams (Dstream)

- **Discretized Stream is the basic abstraction provided by Spark Streaming. It represents a continuous stream of data.**



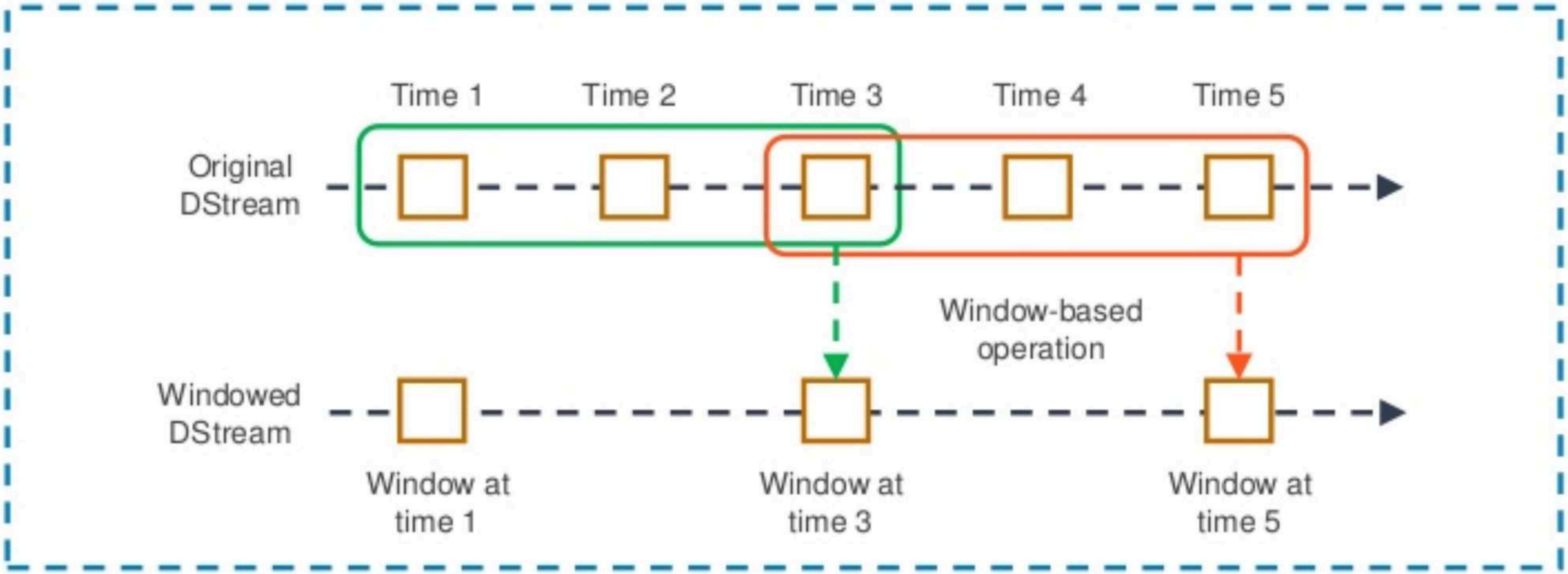
DStream Transformation



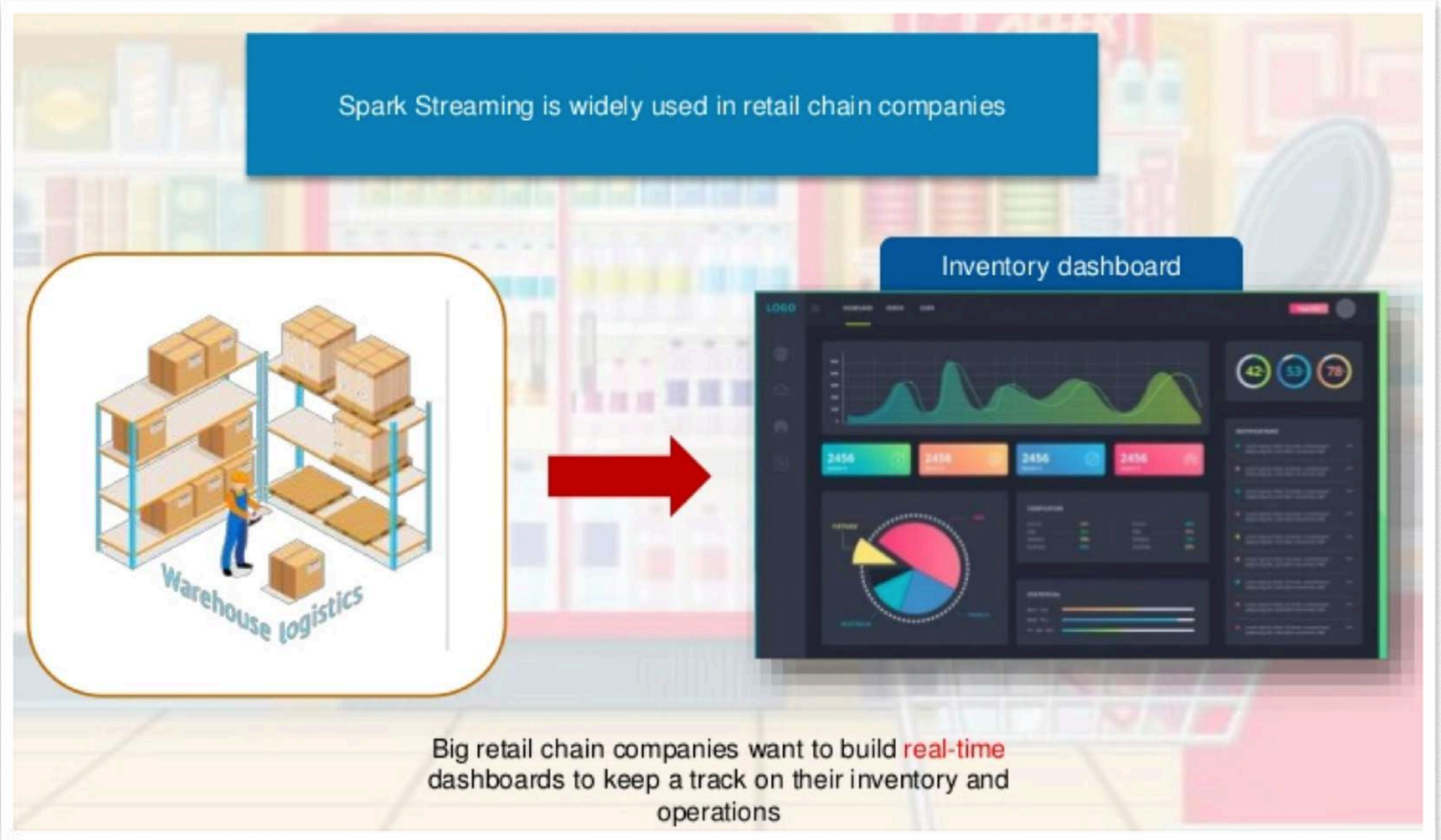
Transformation	Meaning
map(func)	Return a new DStream by passing each element of the source DStream through a function *func*
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items
filter(func)	Return a new DStream by selecting only the records of the source DStream on which *func* returns true
union(otherStream)	Return a new DStream that contains the union of the elements in the source DStream and *otherDStream*
transform(func)	Return a new DStream that contains the union of the elements in the source DStream and *otherDStream*
count()	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream
join(otherStream, [numTasks])	When called on two DStreams of (K, V) and (K, W) pairs, return a new DStream of (K, (V, W)) pairs with all pairs of elements for each key

Windowed Stream Processing

Spark Streaming allows you to apply transformations over a sliding window of data. This operation is called as ***windowed computation***



Spark Streaming Applications



Spark Streaming is widely used in retail chain companies

Inventory dashboard

Big retail chain companies want to build real-time dashboards to keep a track on their inventory and operations

Spark Streaming Applications

Spark Streaming is widely used in retail chain companies

Inventory dashboard

Using these interactive dashboards, retail companies can draw insights about their business

How many products are being purchased

Products that have been shipped

How many products have been delivered to customers

42
53
78
2456
2456
2456
2456

Spark Streaming Applications

Spark Streaming is widely used in retail chain companies

Inventory dashboard

Using these interactive dashboards, retail companies can draw insights about their business

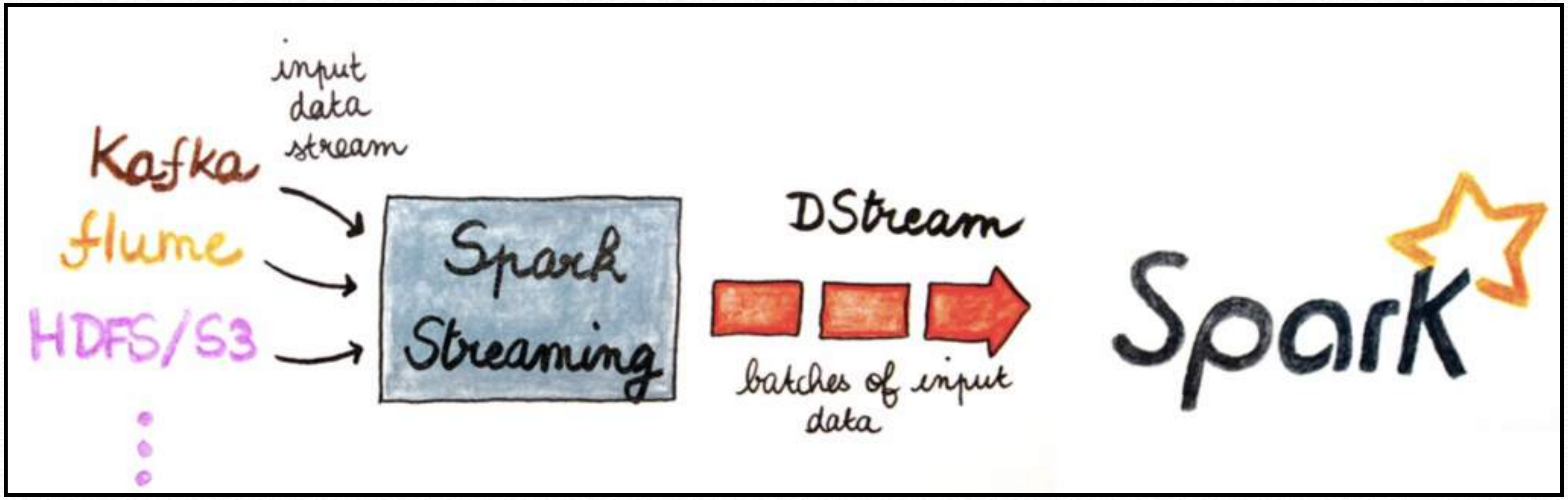
How many products are being purchased

Products that have been shipped

How many products have been delivered to customers

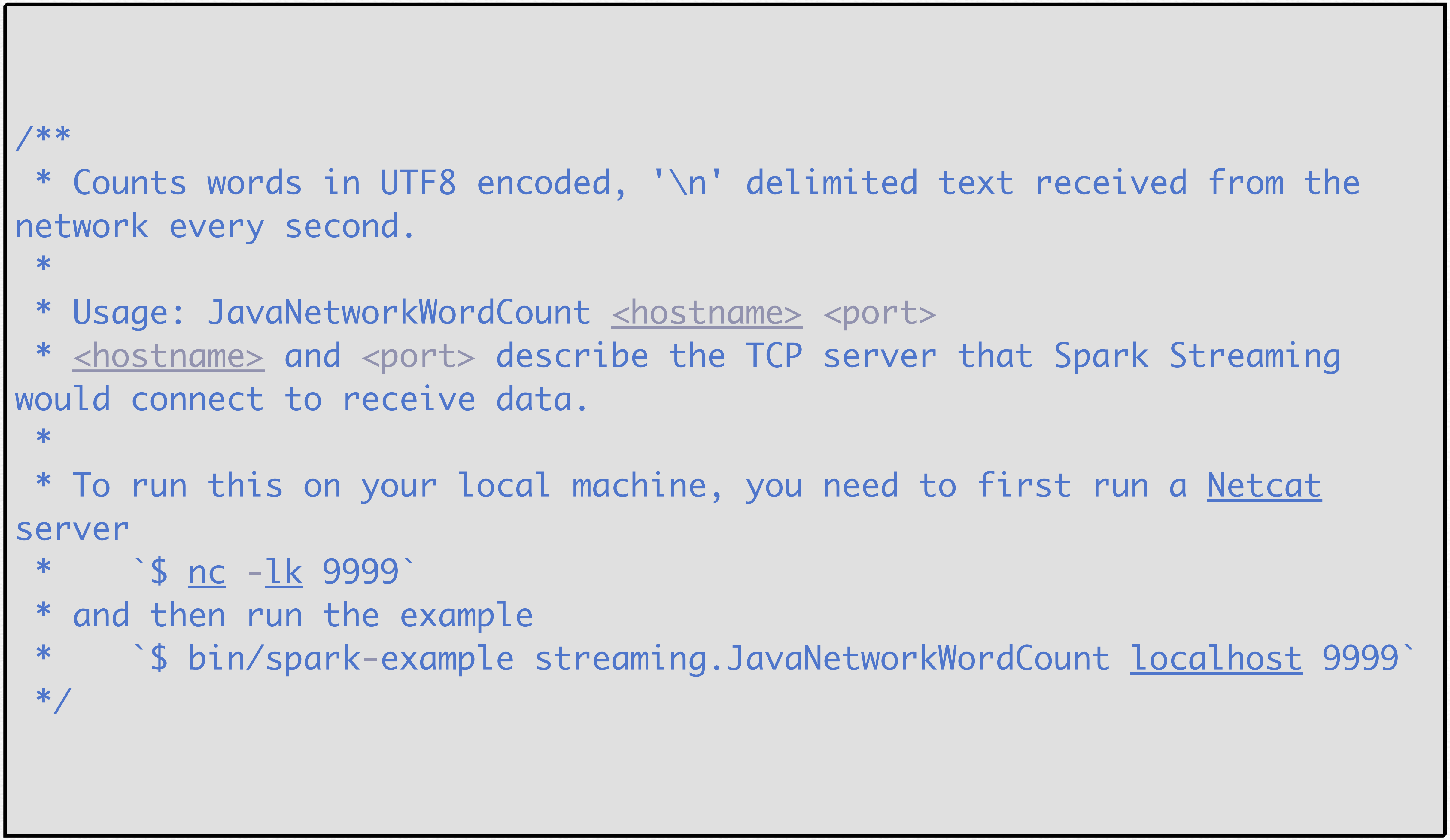
Exercises

- **SPARK Streaming**



Exercises

SPARK Streaming: JavaNetworkWordCount



```
/**
 * Counts words in UTF8 encoded, '\n' delimited text received from the
network every second.
 *
 * Usage: JavaNetworkWordCount <hostname> <port>
 * <hostname> and <port> describe the TCP server that Spark Streaming
would connect to receive data.
 *
 * To run this on your local machine, you need to first run a Netcat
server
 *    `$ nc -lk 9999`
 * and then run the example
 *    `$ bin/spark-example streaming.JavaNetworkWordCount localhost 9999`
 */
```

Exercises

SPARK Streaming: JavaNetworkWordCount

```
public final class JavaNetworkWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.println("Usage: JavaNetworkWordCount <hostname> <port>");
      System.exit(1);
    }

    // Create the context with a 1 second batch size
    SparkConf sparkConf = new SparkConf().setAppName("JavaNetworkWordCount");
    JavaStreamingContext ssc =
                  new JavaStreamingContext(sparkConf, Durations.seconds(1));
```

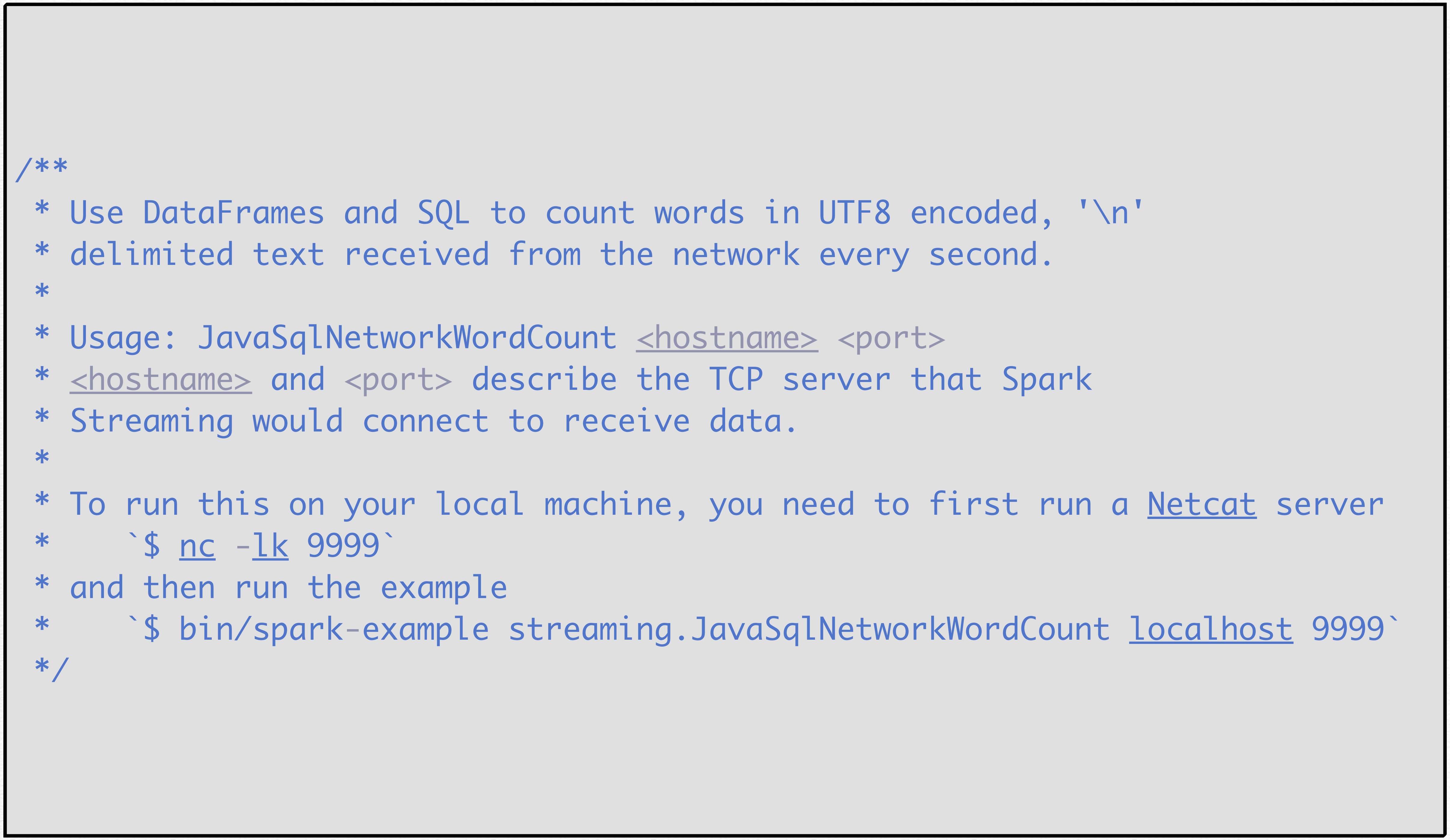
Exercises

SPARK Streaming: JavaNetworkWordCount

```
// Create a JavaReceiverInputDStream on target ip:port and count the
// words in input stream of \n delimited text (eg. generated by 'nc')
// Note that no duplication in storage level only for running locally.
// Replication necessary in distributed scenario for fault tolerance.
    JavaReceiverInputDStream<String> lines =
       ssc.socketTextStream(
        args[0], Integer.parseInt(args[1]), StorageLevels.MEMORY_AND_DISK_SER);

    JavaDStream<String> words = lines
          .flatMap(x -> Arrays.asList(SPACE.split(x)).iterator());

    JavaPairDStream<String, Integer> wordCounts =
                     words.mapToPair(s -> new Tuple2<>(s, 1))
                          .reduceByKey((i1, i2) -> i1 + i2);

    wordCounts.print();
    ssc.start();
    ssc.awaitTermination();
  }
}
```

Exercises

SPARK Streaming: JavaSqlNetworkWordCount



```
/**
 * Use DataFrames and SQL to count words in UTF8 encoded, '\n'
 * delimited text received from the network every second.
 *
 * Usage: JavaSqlNetworkWordCount <hostname> <port>
 * <hostname> and <port> describe the TCP server that Spark
 * Streaming would connect to receive data.
 *
 * To run this on your local machine, you need to first run a Netcat server
 *    `$ nc -lk 9999`
 * and then run the example
 *    `$ bin/spark-example streaming.JavaSqlNetworkWordCount localhost 9999`
 */
```

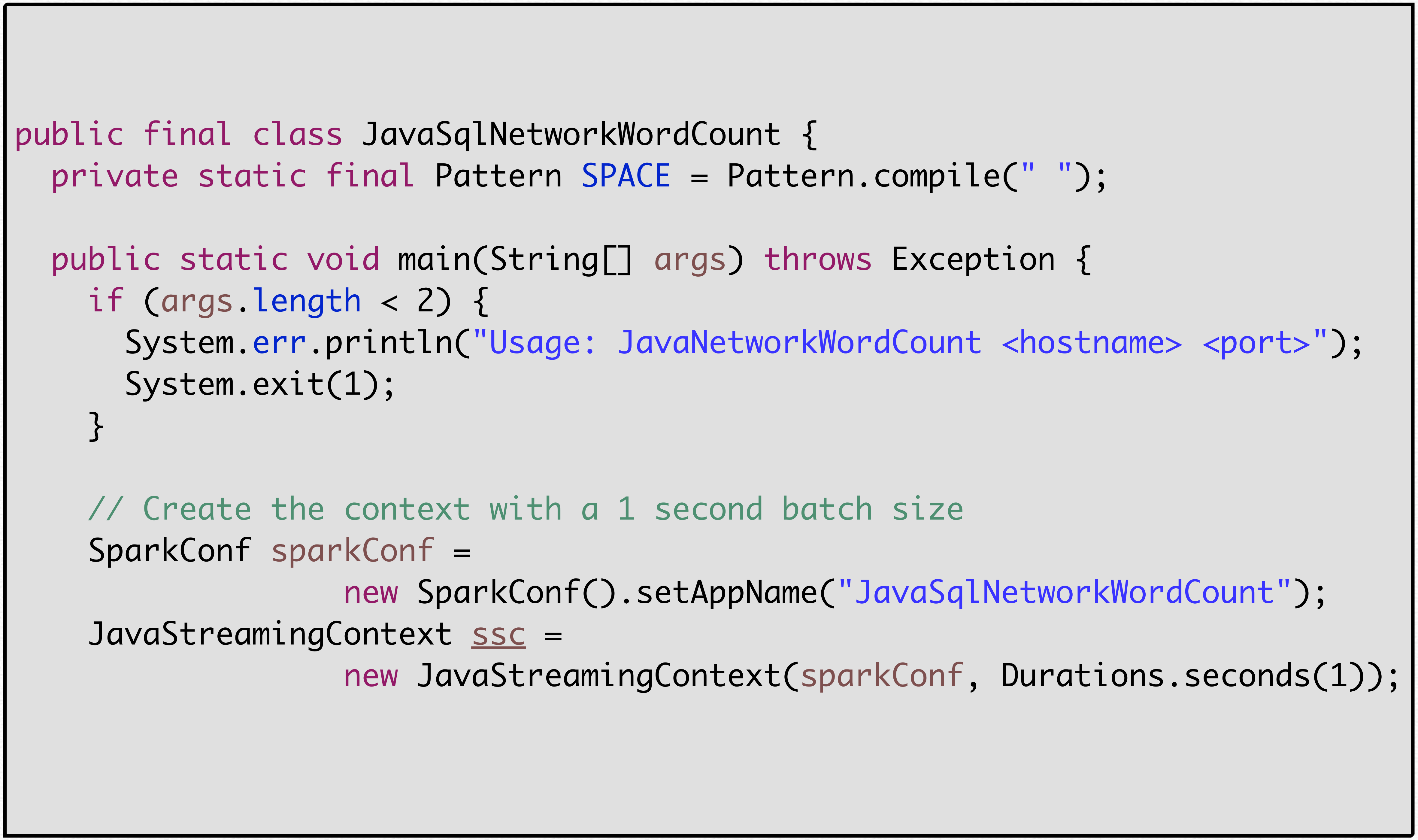
Exercises

SPARK Streaming: JavaSqlNetworkWordCount

```
/** Java Bean class to be used with the example JavaSqlNetworkWordCount. */
public class JavaRecord implements java.io.Serializable {
  private String word;

  public String getWord() {
    return word;
  }

  public void setWord(String word) {
    this.word = word;
  }
}
```

Exercises

SPARK Streaming: JavaSqlNetworkWordCount



```java
public final class JavaSqlNetworkWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.println("Usage: JavaNetworkWordCount <hostname> <port>");
      System.exit(1);
    }

    // Create the context with a 1 second batch size
    SparkConf sparkConf =
                  new SparkConf().setAppName("JavaSqlNetworkWordCount");
    JavaStreamingContext ssc =
                  new JavaStreamingContext(sparkConf, Durations.seconds(1));
```

Exercises

SPARK Streaming: JavaSqlNetworkWordCount

```
// Create a JavaReceiverInputDStream on target ip:port and count the
// words in input stream of \n delimited text (eg. generated by 'nc')
// Note that no duplication in storage level only for running locally.
// Replication necessary in distributed scenario for fault tolerance.

JavaReceiverInputDStream<String> lines =
   ssc.socketTextStream(
    args[0], Integer.parseInt(args[1]), StorageLevels.MEMORY_AND_DISK_SER);

JavaDStream<String> words = lines
                   .flatMap(x -> Arrays.asList(SPACE.split(x)).iterator());
```

Exercises

SPARK Streaming: JavaSqlNetworkWordCount

```
// Convert RDDs of the words DStream to DataFrame and run SQL query
    words.foreachRDD((rdd, time) -> {
      SparkSession spark =
        JavaSparkSessionSingleton.getInstance(rdd.context().getConf());

// Convert JavaRDD[String] to JavaRDD[bean class] to DataFrame
      JavaRDD<JavaRecord> rowRDD = rdd.map(word -> {
        JavaRecord record = new JavaRecord();
        record.setWord(word);
        return record;
      });
      Dataset<Row> wordsDataFrame =
                     spark.createDataFrame(rowRDD, JavaRecord.class);
```

Exercises

SPARK Streaming: JavaSqlNetworkWordCount

```
// Creates a temporary view using the DataFrame
      wordsDataFrame.createOrReplaceTempView("words");

// Do word count on table using SQL and print it
      Dataset<Row> wordCountsDataFrame =
      spark.sql("select word, count(*) as total from words group by word");
      System.out.println("========= " + time + "=========");
      wordCountsDataFrame.show();
    });

    ssc.start();
    ssc.awaitTermination();
  }
}
```

Exercises

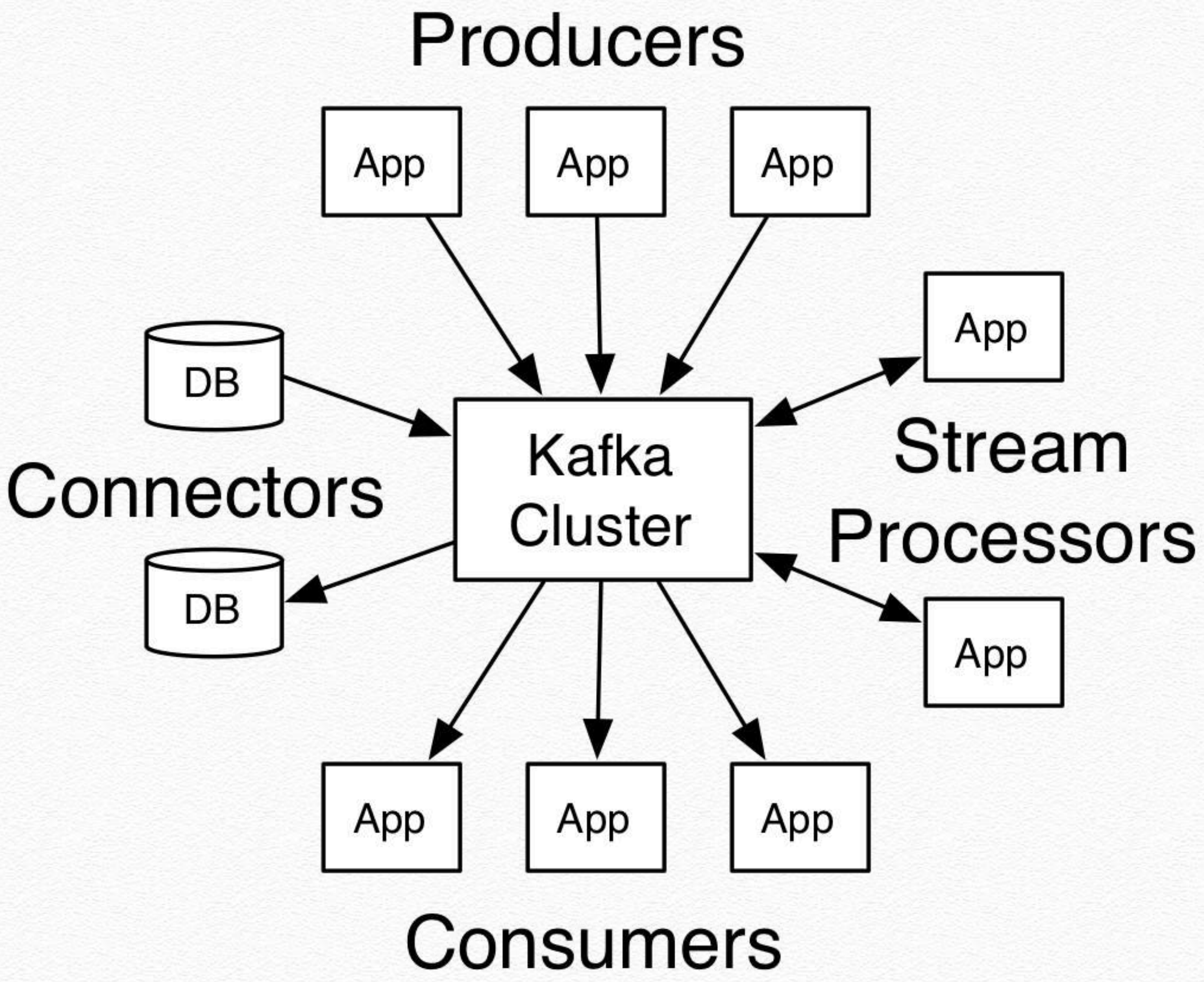
SPARK Streaming: JavaSqlNetworkWordCount

```
/** Lazily instantiated singleton instance of SparkSession */
class JavaSparkSessionSingleton {
  private static transient SparkSession instance = null;
  public static SparkSession getInstance(SparkConf sparkConf) {
    if (instance == null) {
      instance = SparkSession
        .builder()
        .config(sparkConf)
        .getOrCreate();
    }
    return instance;
  }
}
```

Apache Spark Streaming

Apache kafka®

A distributed streaming platform



Producers

App

App

App

DB

Connectors

DB

Kafka Cluster

App

Stream Processors

App

App

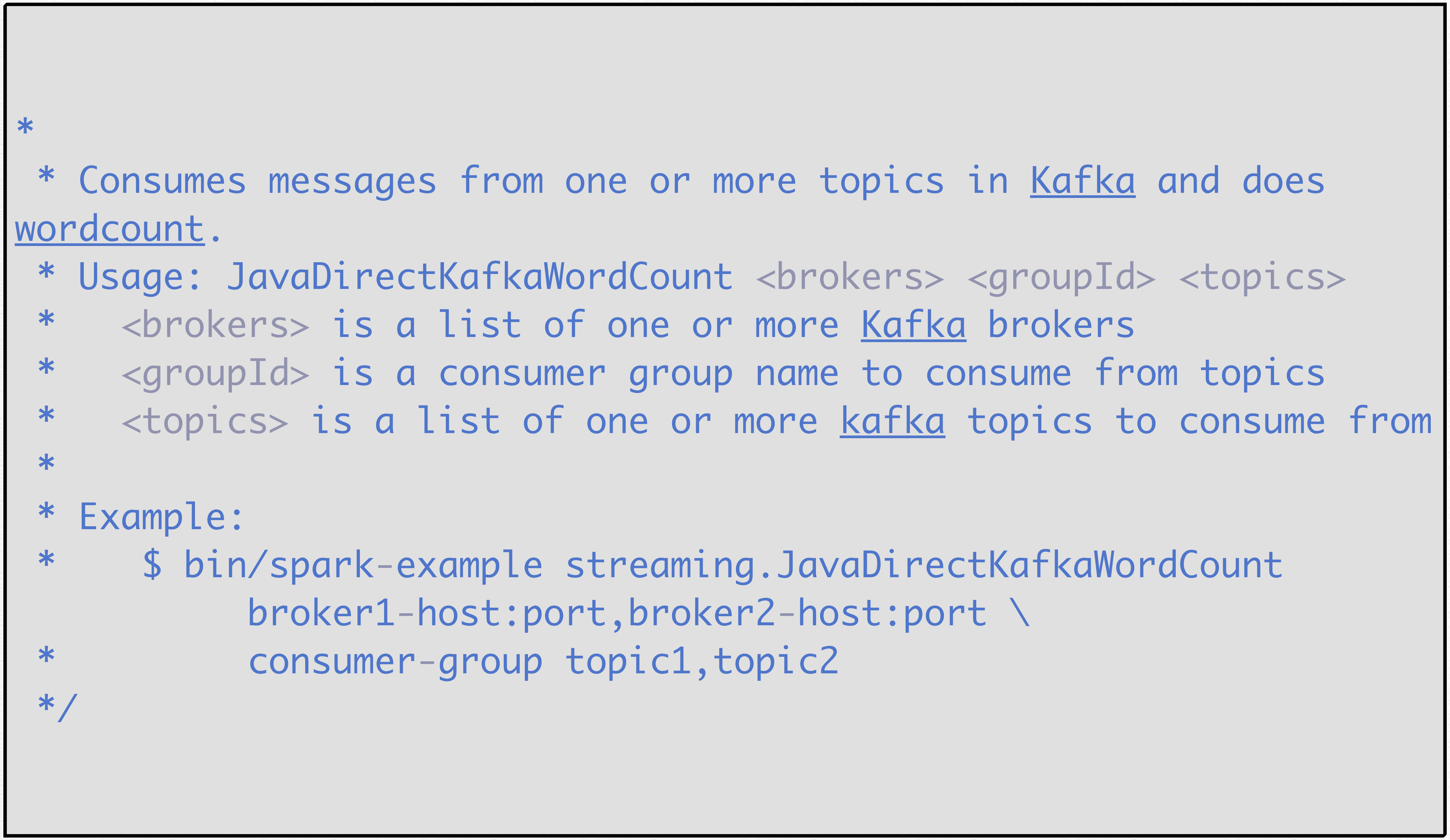
App

App

Consumers

Exercises

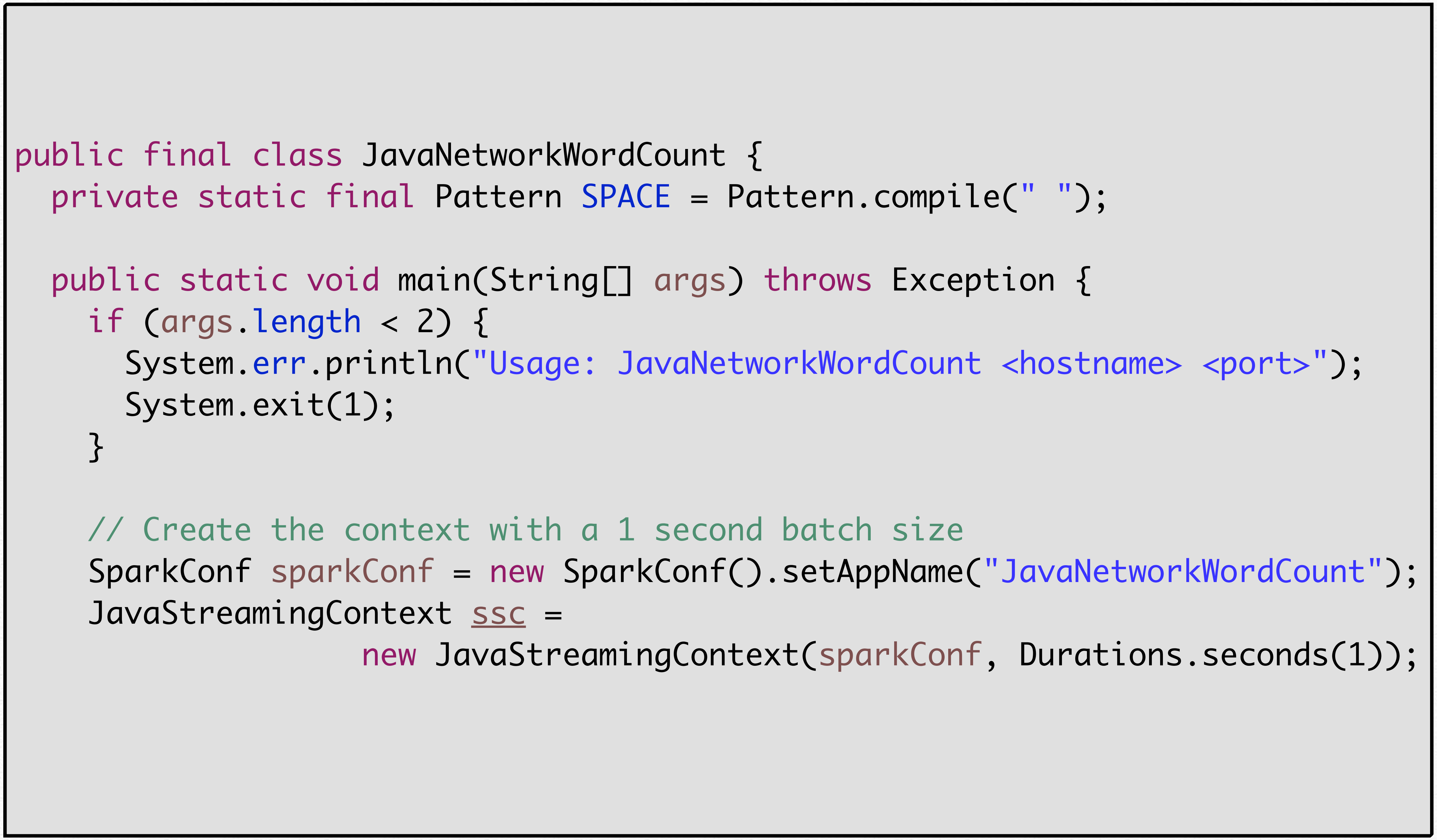
SPARK Streaming: JavaDirectKafkaWordCount



```
*
 * Consumes messages from one or more topics in Kafka and does
wordcount.
 * Usage: JavaDirectKafkaWordCount <brokers> <groupId> <topics>
 *   <brokers> is a list of one or more Kafka brokers
 *   <groupId> is a consumer group name to consume from topics
 *   <topics> is a list of one or more kafka topics to consume from
 *
 * Example:
 *    $ bin/spark-example streaming.JavaDirectKafkaWordCount
           broker1-host:port,broker2-host:port \
 *         consumer-group topic1,topic2
 */
```

Exercises

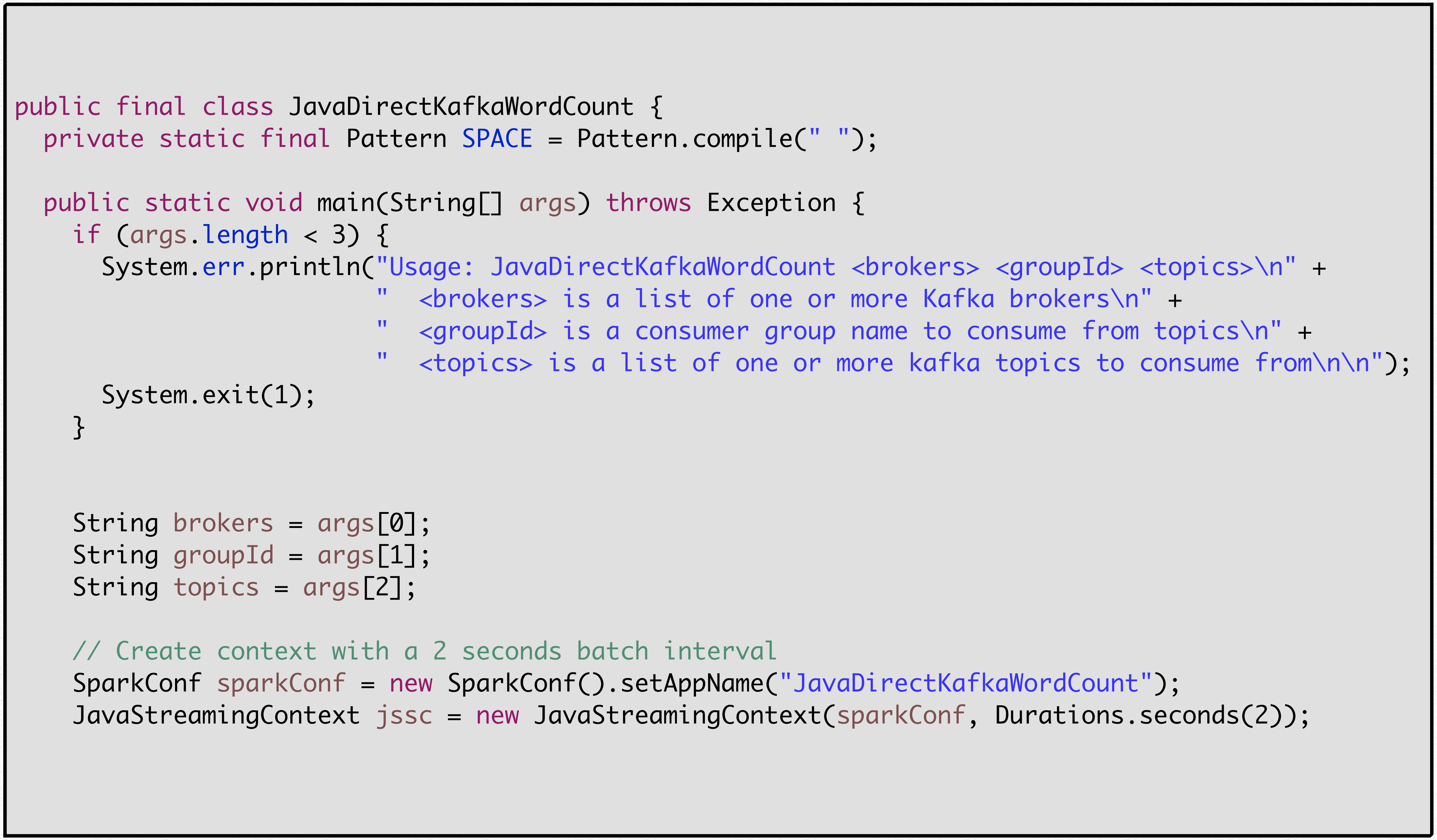
- **SPARK Streaming: JavaDirectKafkaWordCount**



```java
public final class JavaNetworkWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.println("Usage: JavaNetworkWordCount <hostname> <port>");
      System.exit(1);
    }

    // Create the context with a 1 second batch size
    SparkConf sparkConf = new SparkConf().setAppName("JavaNetworkWordCount");
    JavaStreamingContext ssc =
                  new JavaStreamingContext(sparkConf, Durations.seconds(1));
```

Exercises

SPARK Streaming: JavaDirectKafkaWordCount



```java
public final class JavaDirectKafkaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {
    if (args.length < 3) {
      System.err.println("Usage: JavaDirectKafkaWordCount <brokers> <groupId> <topics>\n" +
                         "  <brokers> is a list of one or more Kafka brokers\n" +
                         "  <groupId> is a consumer group name to consume from topics\n" +
                         "  <topics> is a list of one or more kafka topics to consume from\n\n");
      System.exit(1);
    }


    String brokers = args[0];
    String groupId = args[1];
    String topics = args[2];

    // Create context with a 2 seconds batch interval
    SparkConf sparkConf = new SparkConf().setAppName("JavaDirectKafkaWordCount");
    JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));
```

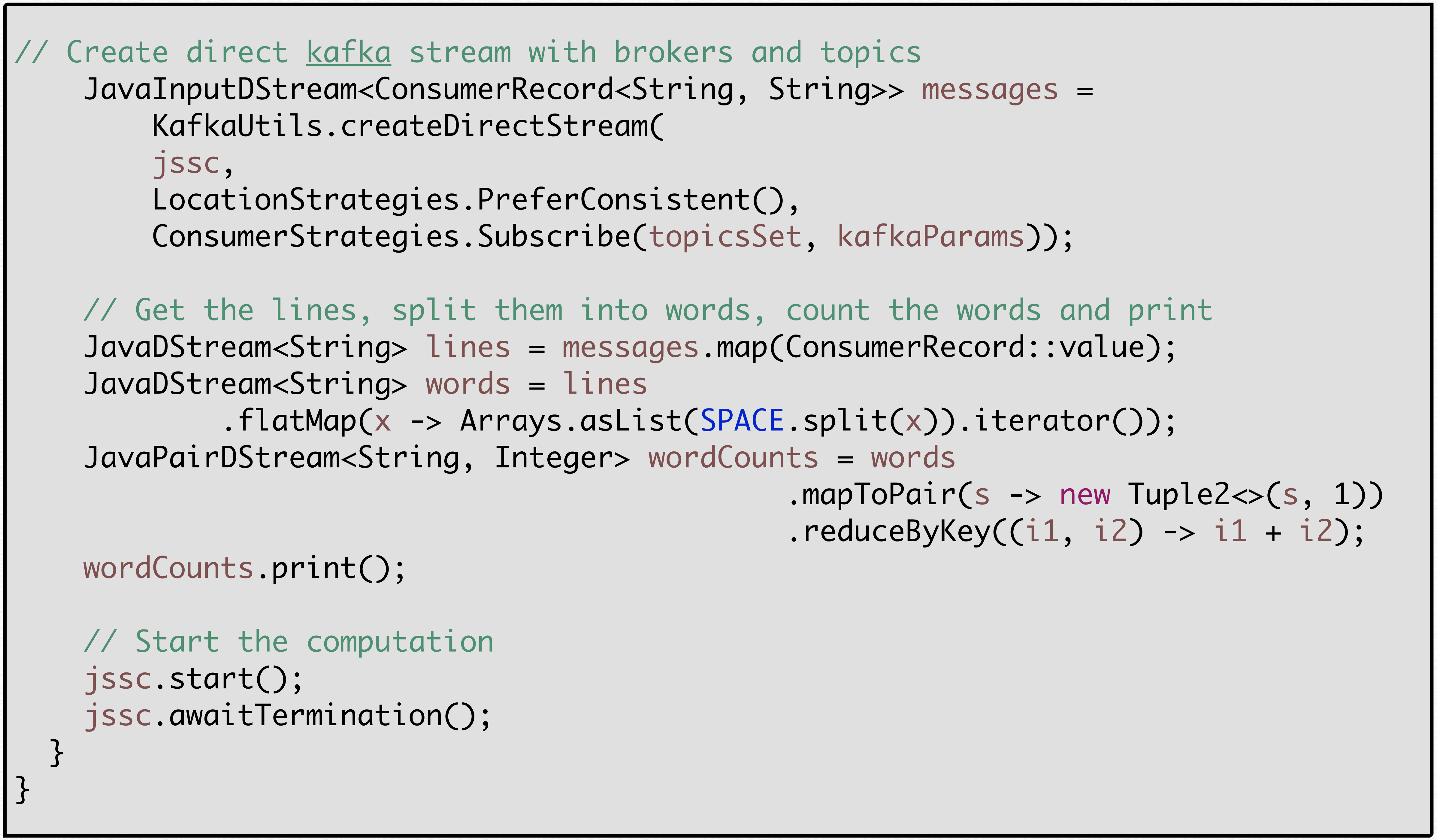
Exercises

SPARK Streaming: JavaDirectKafkaWordCount

```
Set<String> topicsSet = new HashSet<>(Arrays.asList(topics.split(",")));
    Map<String, Object> kafkaParams = new HashMap<>();
    kafkaParams.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers);
    kafkaParams.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
    kafkaParams.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                                             StringDeserializer.class);
    kafkaParams.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                                             StringDeserializer.class);
```

Exercises

SPARK Streaming: JavaDirectKafkaWordCount



```
// Create direct kafka stream with brokers and topics
    JavaInputDStream<ConsumerRecord<String, String>> messages =
        KafkaUtils.createDirectStream(
        jssc,
        LocationStrategies.PreferConsistent(),
        ConsumerStrategies.Subscribe(topicsSet, kafkaParams));

    // Get the lines, split them into words, count the words and print
    JavaDStream<String> lines = messages.map(ConsumerRecord::value);
    JavaDStream<String> words = lines
            .flatMap(x -> Arrays.asList(SPACE.split(x)).iterator());
    JavaPairDStream<String, Integer> wordCounts = words
                                             .mapToPair(s -> new Tuple2<>(s, 1))
                                             .reduceByKey((i1, i2) -> i1 + i2);
    wordCounts.print();

    // Start the computation
    jssc.start();
    jssc.awaitTermination();
  }
}
```

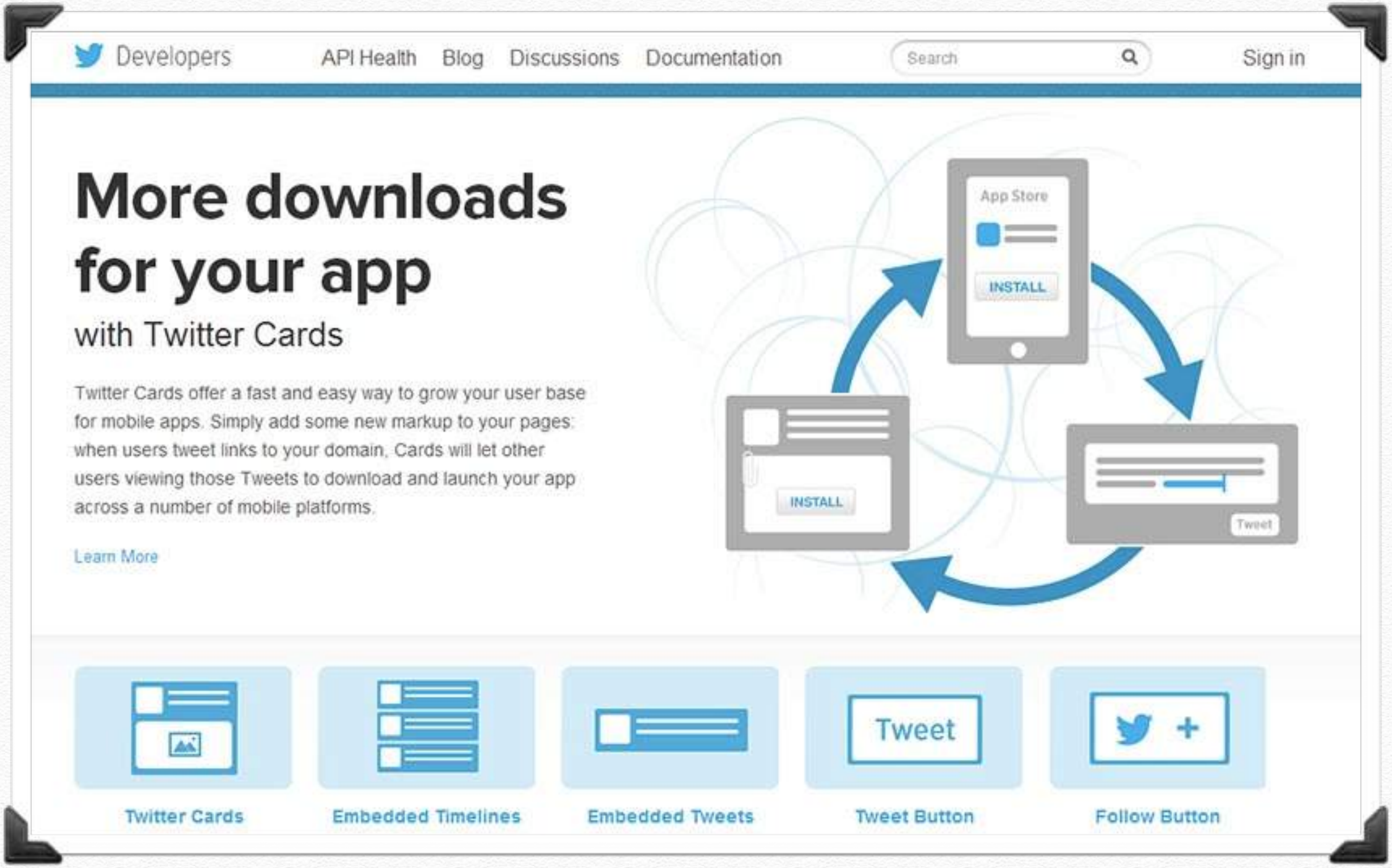
https://www.youtube.com/watch?v=65lHphtrfo0

- Web video to **configure KAFKA**

```
if path:
    self.file = open(os.pat
    self.file.seek(0)
    self.fingerprints.upda

@classmethod
def from_settings(cls, se
    debug = settings.get
    return cls(job_dir(s

def request_seen(self, reque
    fp = self.request_finger
    if fp in self.fingerpri
        return True
    self.fingerprints.add(f
    if self.file:
        self.file.write(fp

def request_fingerprint(se
```

How to Register a Twitter App in 8 Easy Steps

#1 Visit the Twitter Developers' Site



More downloads for your app

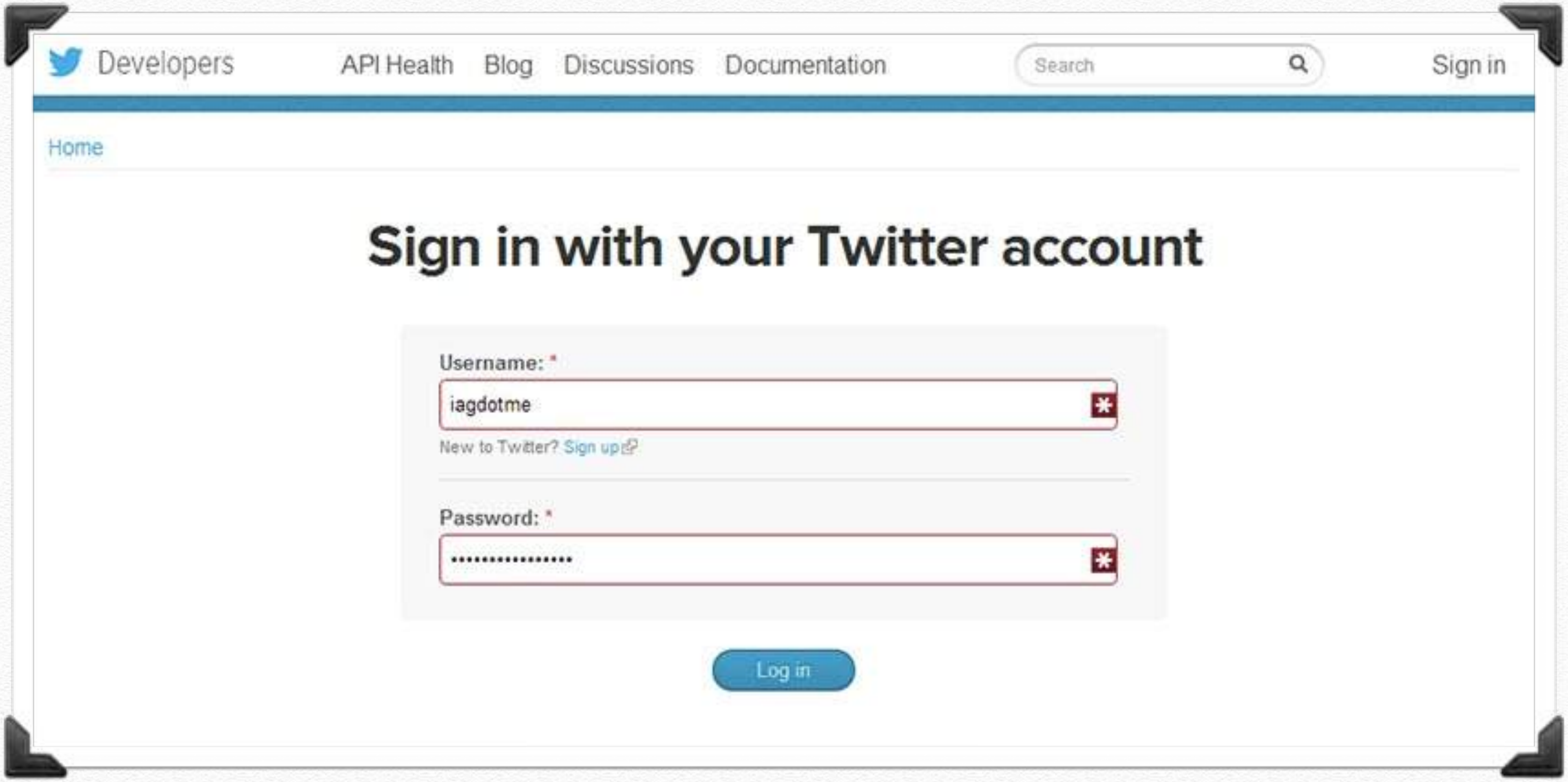
with Twitter Cards

Twitter Cards offer a fast and easy way to grow your user base for mobile apps. Simply add some new markup to your pages: when users tweet links to your domain, Cards will let other users viewing those Tweets to download and launch your app across a number of mobile platforms.

Learn More

How to Register a Twitter App in 8 Easy Steps

#2 Sign in with your Twitter Account



Sign in with your Twitter account

Username: *

iagdotme

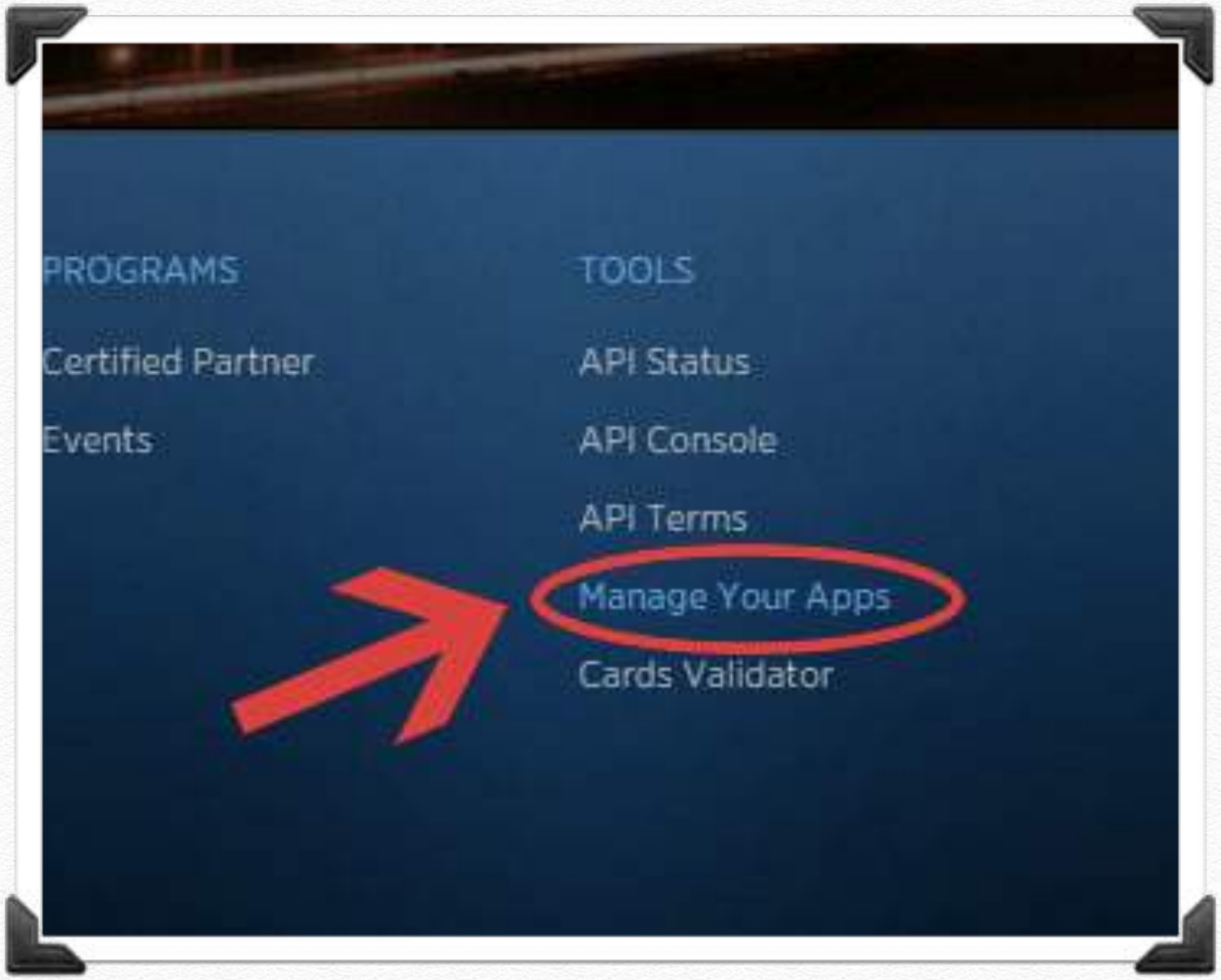
New to Twitter? Sign up

Password: *

•••••••••••••••••

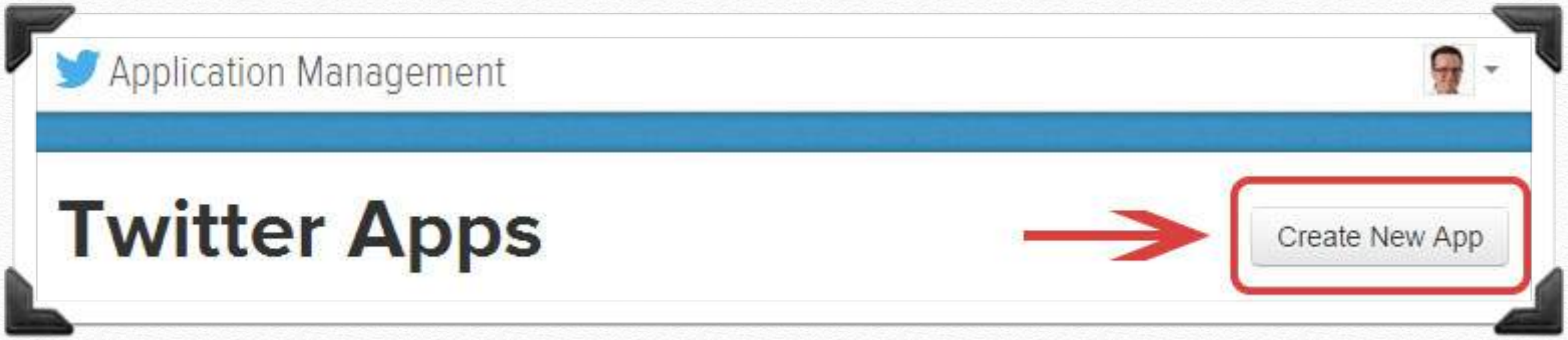
How to Register a Twitter App in 8 Easy Steps

- **#3 Go to apps.twitter.com**



How to Register a Twitter App in 8 Easy Steps

#4 Create a New Application

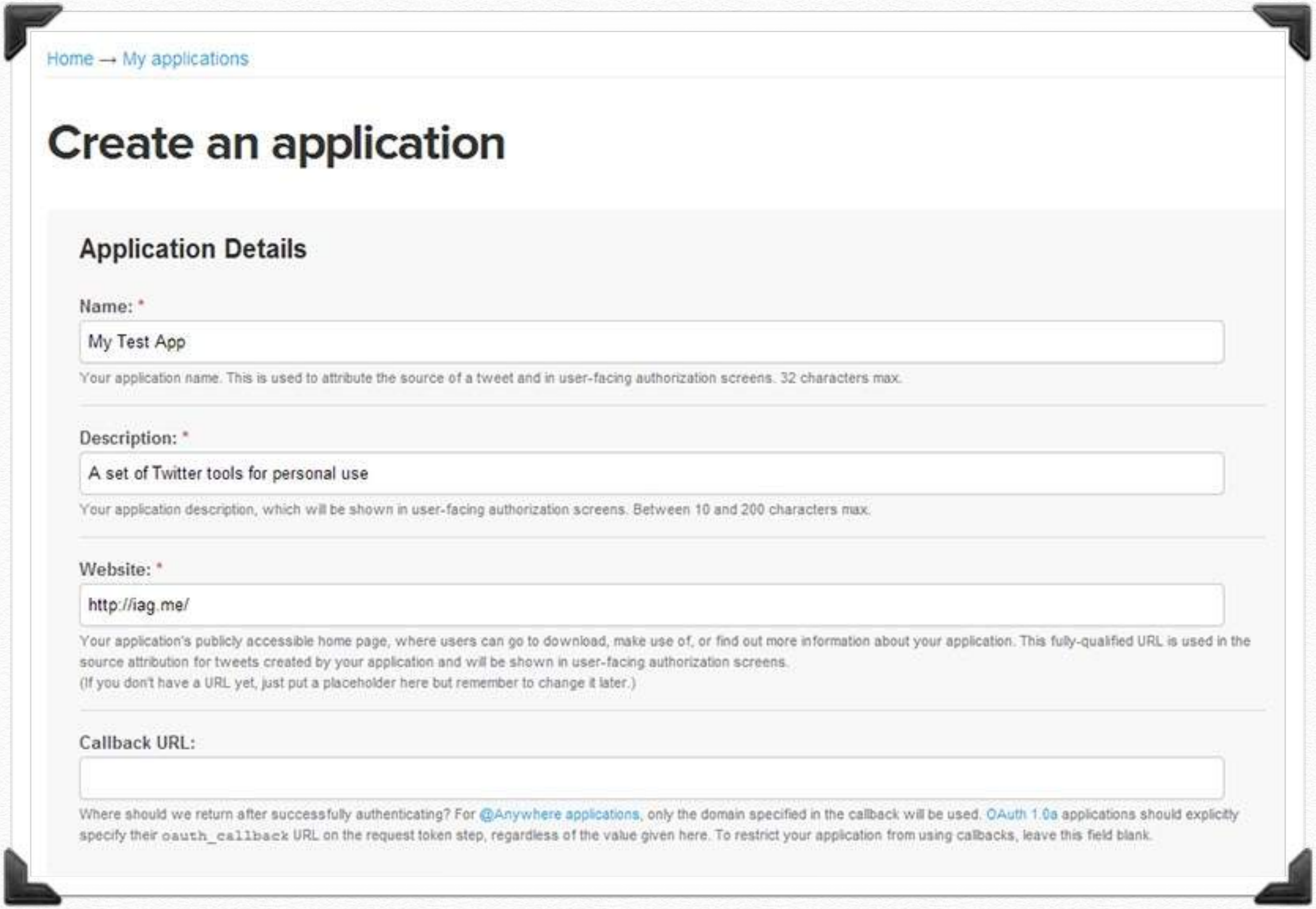


Application Management

Twitter Apps

How to Register a Twitter App in 8 Easy Steps

#5 Fill in your Application Details



Home → My applications

Create an application

Application Details

Name: *

My Test App

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description: *

A set of Twitter tools for personal use

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website: *

http://iag.me/

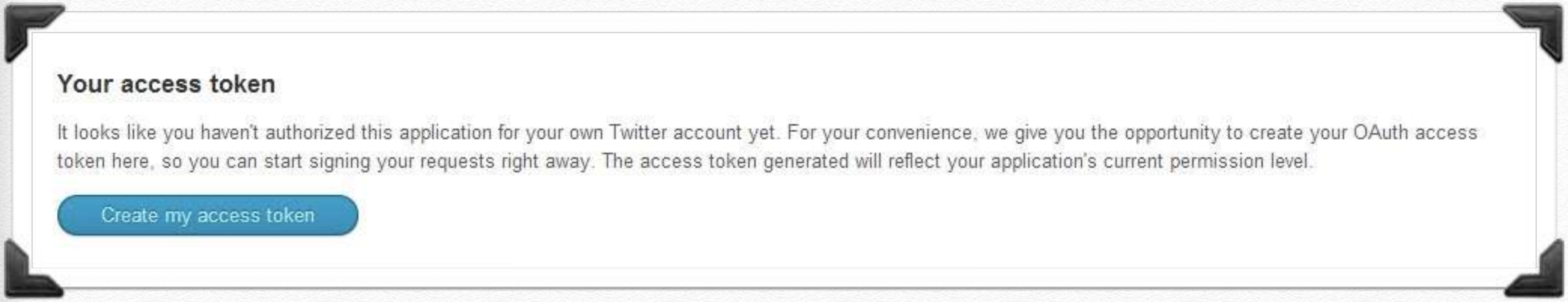
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL:

Where should we return after successfully authenticating? For @Anywhere applications, only the domain specified in the callback will be used. OAuth 1.0a applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

How to Register a Twitter App in 8 Easy Steps

#6 Create Your Access Token

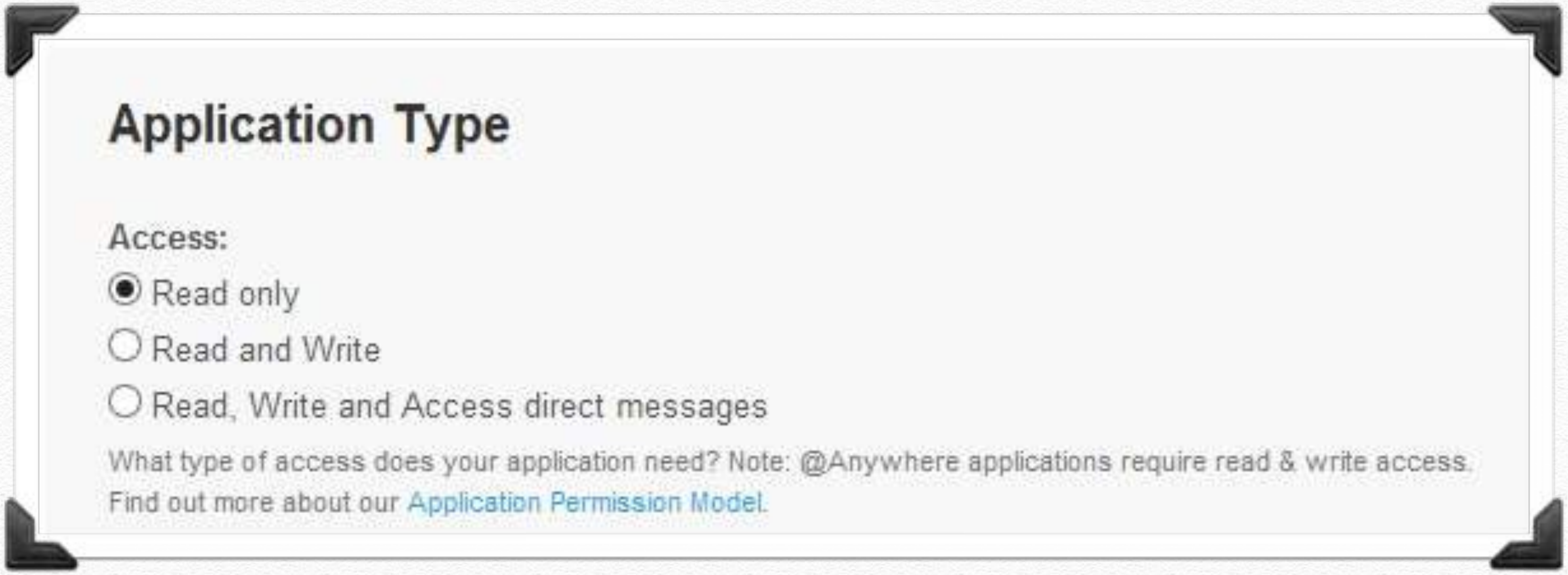


Your access token

It looks like you haven't authorized this application for your own Twitter account yet. For your convenience, we give you the opportunity to create your OAuth access token here, so you can start signing your requests right away. The access token generated will reflect your application's current permission level.

How to Register a Twitter App in 8 Easy Steps

#7 Choose what Access Type You Need



Application Type

Access:

● Read only

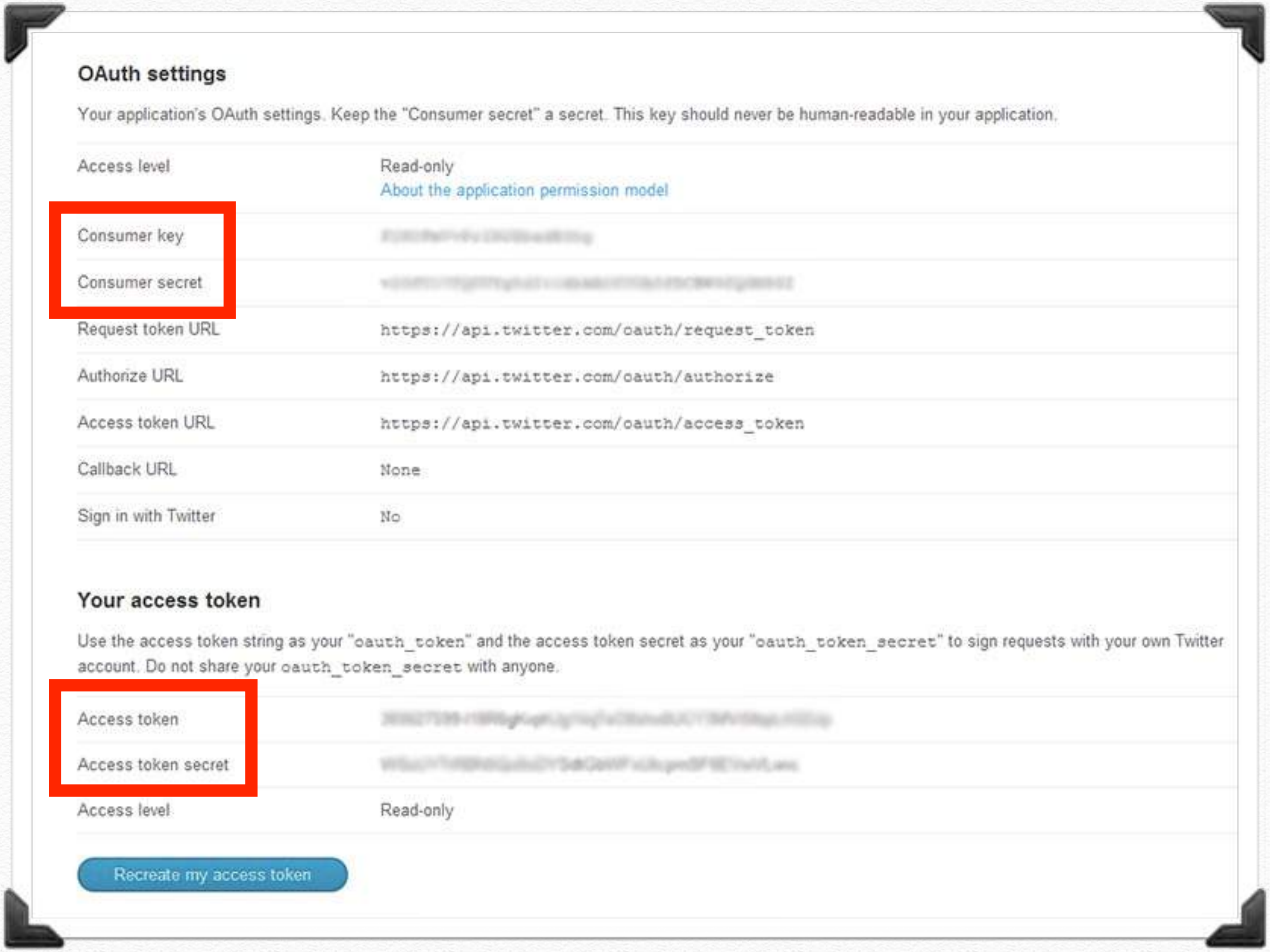
○ Read and Write

○ Read, Write and Access direct messages

What type of access does your application need? Note: @Anywhere applications require read & write access.
Find out more about our Application Permission Model.

How to Register a Twitter App in 8 Easy Steps

#8 Make a note of your OAuth Settings



OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

Access level	Read-only About the application permission model
Consumer key	[redacted]
Consumer secret	[redacted]
Request token URL	`https://api.twitter.com/oauth/request_token`
Authorize URL	`https://api.twitter.com/oauth/authorize`
Access token URL	`https://api.twitter.com/oauth/access_token`
Callback URL	None
Sign in with Twitter	No

Your access token

Use the access token string as your "`oauth_token`" and the access token secret as your "`oauth_token_secret`" to sign requests with your own Twitter account. Do not share your `oauth_token_secret` with anyone.

Access token	[redacted]
Access token secret	[redacted]
Access level	Read-only

Exercises

SPARK Streaming: StreamUtils

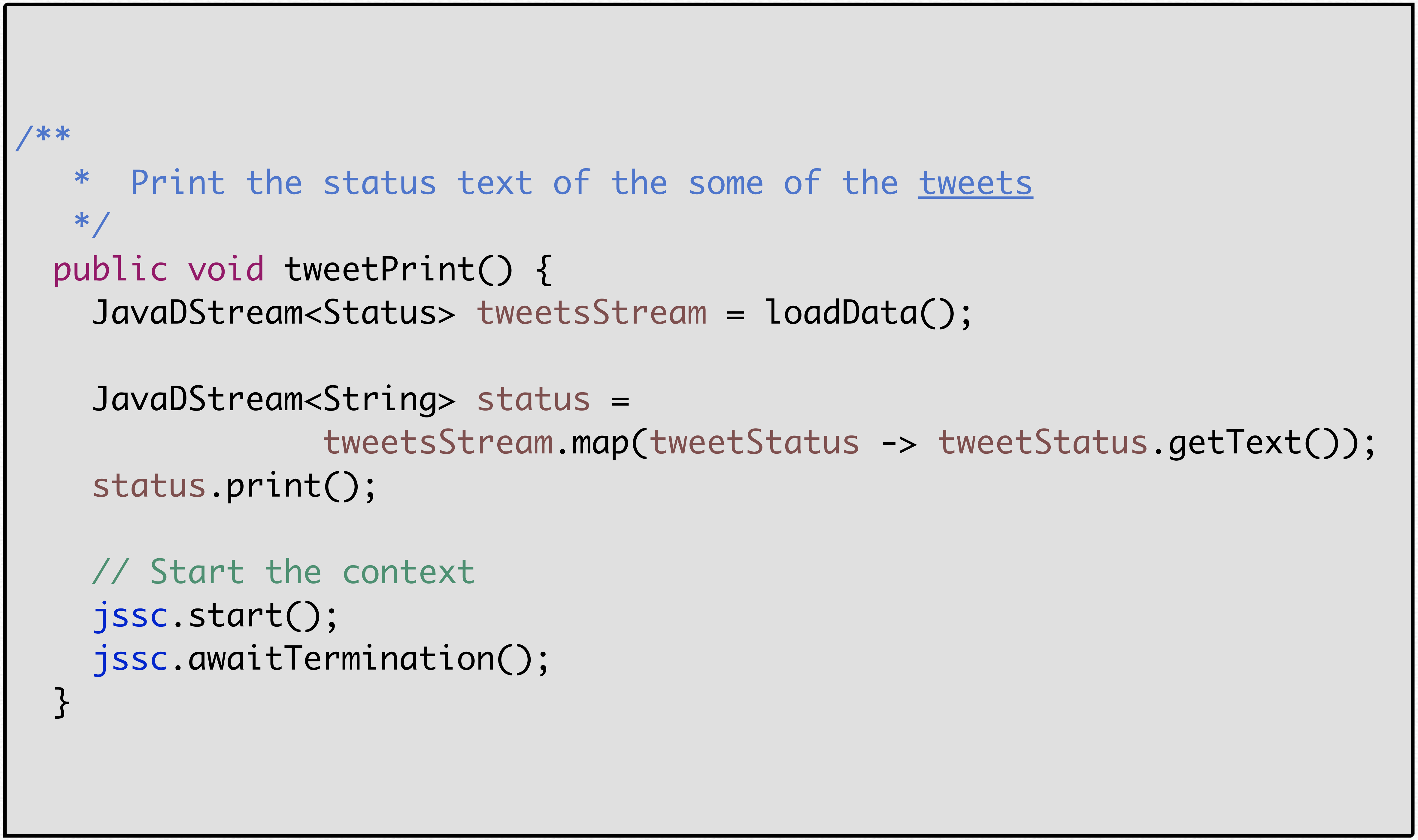
```
public class StreamUtils {

  private static String CONSUMER_KEY = "AFiNCb8vxYZfhPls2DXyDpF";
  private static String CONSUMER_SECRET = "JRg7SyVFkXEESWbzFzC1xaIGRC3xNdTvrekMvMFk6tjKooOR";
  private static String ACCESS_TOKEN = "493498548-HCt6LCposCb3Ij7Ygt7ssTxTBPwGoPrnkkDQoaN";
  private static String ACCESS_TOKEN_SECRET = "3px3rnBzWa9bmOmOQPWNMpYc4qdOrOdxGFgp6XiCkEKH";

  public static OAuthAuthorization getAuth() {

    return new OAuthAuthorization(
        new ConfigurationBuilder().setOAuthConsumerKey(CONSUMER_KEY)
            .setOAuthConsumerSecret(CONSUMER_SECRET)
            .setOAuthAccessToken(ACCESS_TOKEN)
            .setOAuthAccessTokenSecret(ACCESS_TOKEN_SECRET)
            .build());
  }
}
```

SPARK Streaming: StreamingOnTweets

```
public class StreamingOnTweets {

  JavaStreamingContext jssc;

  public JavaDStream<Status> loadData() {
    SparkConf conf = new SparkConf()
        .setAppName("Play with Spark Streaming”);

    // create a java streaming context and define the window (2 seconds batch)
    jssc = new JavaStreamingContext(conf, Durations.seconds(2));

    System.out.println("Initializing Twitter stream...");

    // create a DStream (sequence of RDD). The object tweetsStream is a
    // DStream of tweet statuses:
    // - the Status class contains all information of a tweet
    // See http://twitter4j.org/javadoc/twitter4j/Status.html
    JavaDStream<Status> tweetsStream =
                    TwitterUtils.createStream(jssc, StreamUtils.getAuth());

    return tweetsStream;

  }
```

Exercises

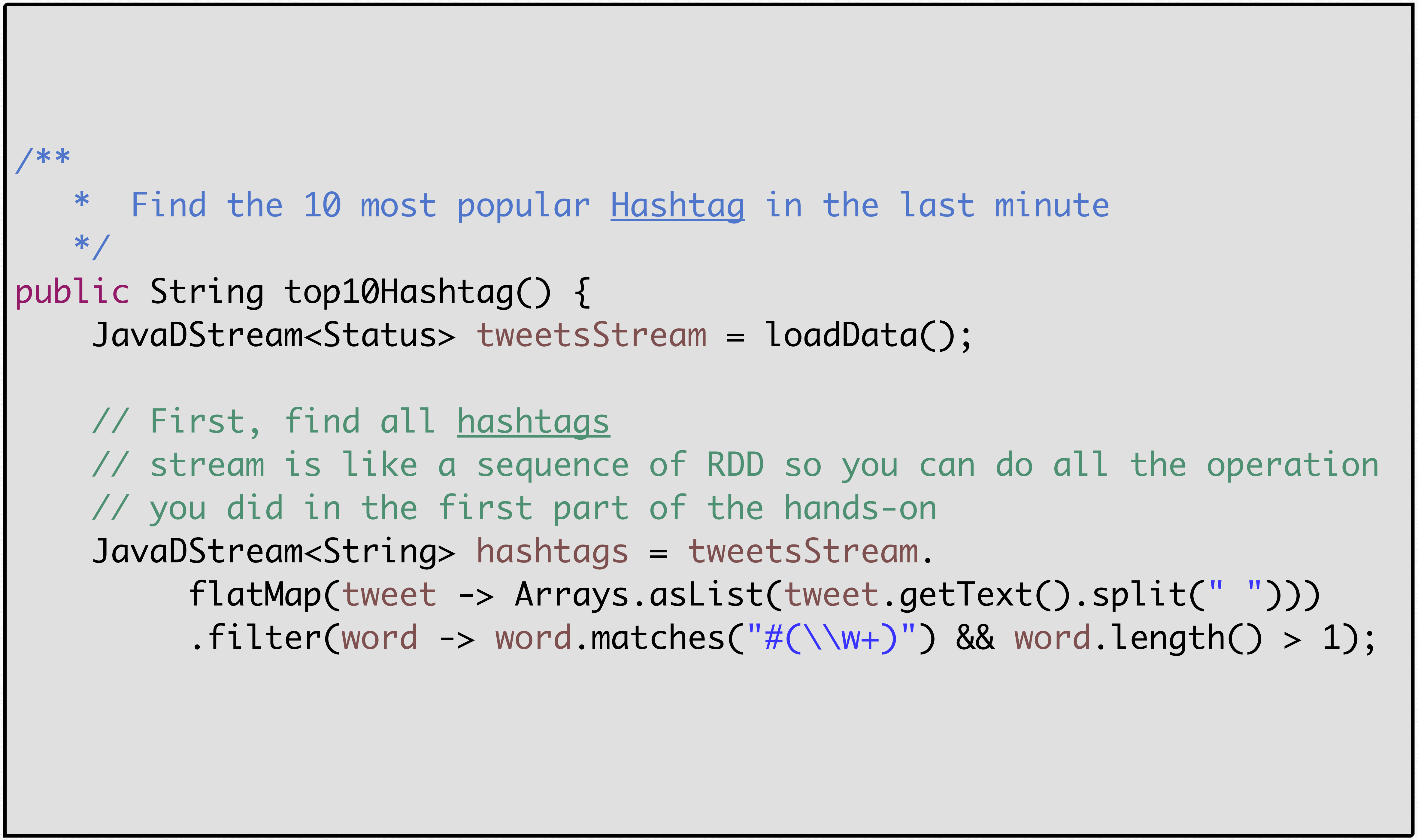
SPARK Streaming: StreamingOnTweets



```
/**
   *  Print the status text of the some of the tweets
   */
  public void tweetPrint() {
    JavaDStream<Status> tweetsStream = loadData();

    JavaDStream<String> status =
                tweetsStream.map(tweetStatus -> tweetStatus.getText());
    status.print();

    // Start the context
    jssc.start();
    jssc.awaitTermination();
  }
```

Exercises

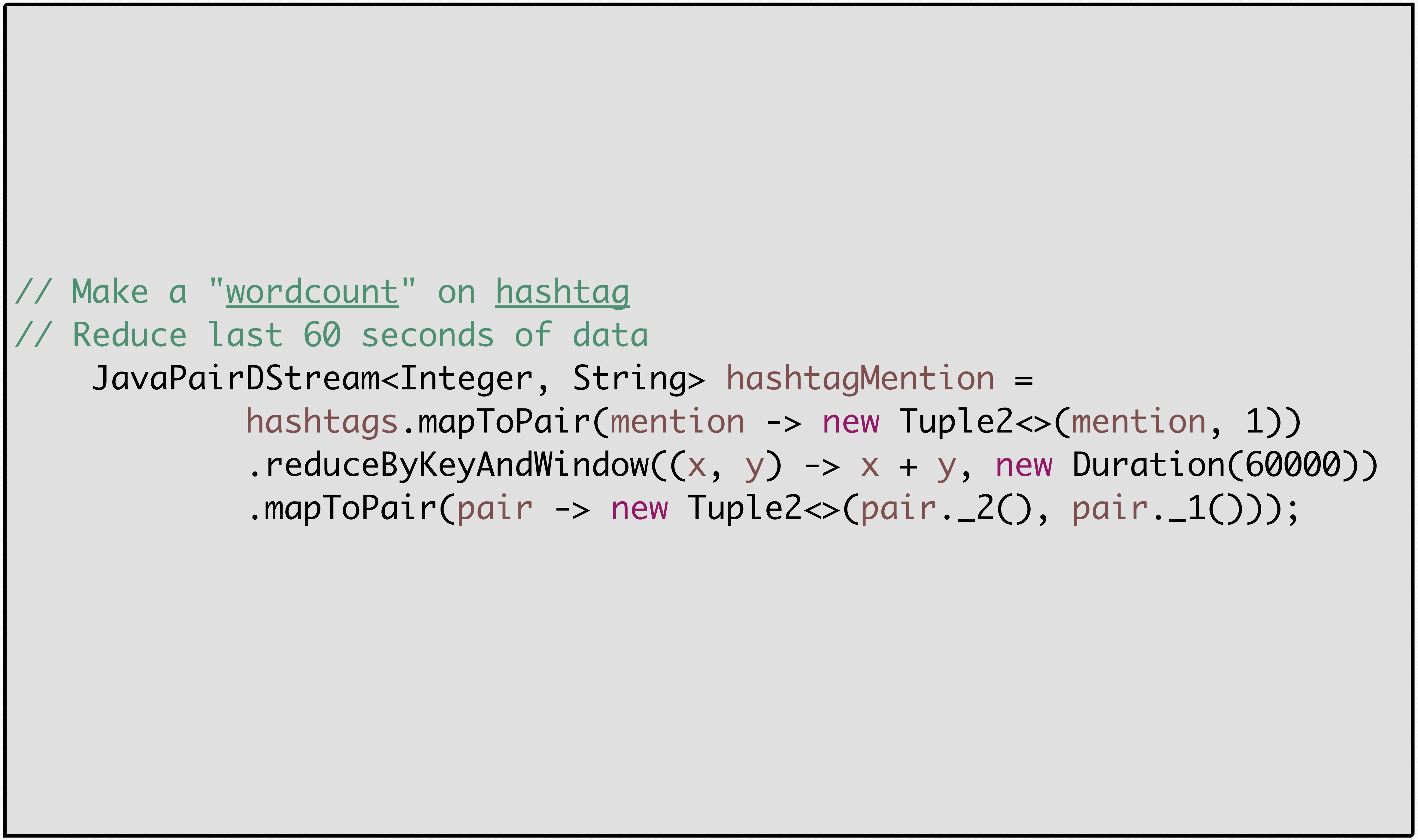
SPARK Streaming: StreamingOnTweets



```
/**
   *  Find the 10 most popular Hashtag in the last minute
   */
public String top10Hashtag() {
    JavaDStream<Status> tweetsStream = loadData();

    // First, find all hashtags
    // stream is like a sequence of RDD so you can do all the operation
    // you did in the first part of the hands-on
    JavaDStream<String> hashtags = tweetsStream.
         flatMap(tweet -> Arrays.asList(tweet.getText().split(" ")))
         .filter(word -> word.matches("#(\\w+)") && word.length() > 1);
```

Exercises

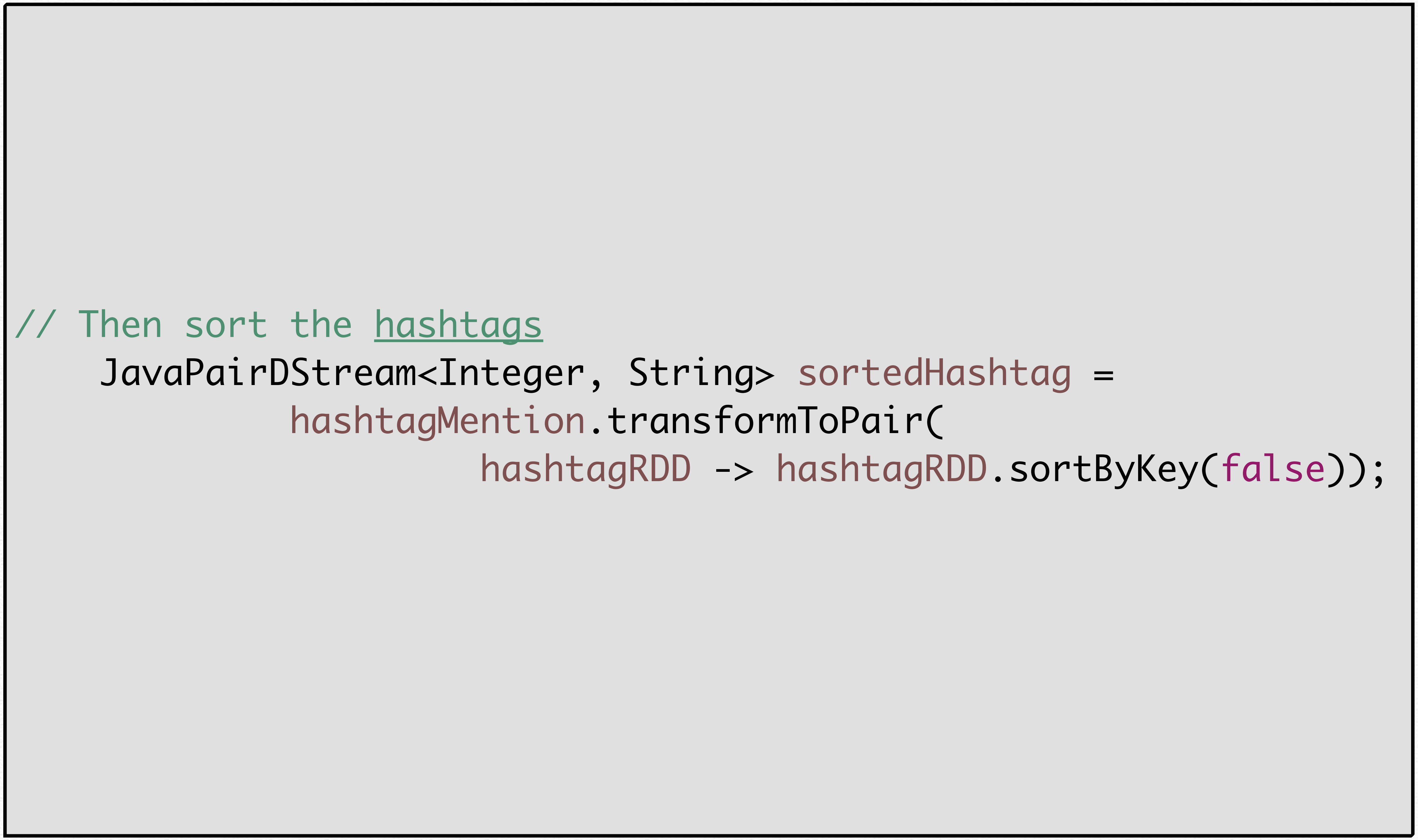
SPARK Streaming: StreamingOnTweets



```
// Make a "wordcount" on hashtag
// Reduce last 60 seconds of data
    JavaPairDStream<Integer, String> hashtagMention =
            hashtags.mapToPair(mention -> new Tuple2<>(mention, 1))
            .reduceByKeyAndWindow((x, y) -> x + y, new Duration(60000))
            .mapToPair(pair -> new Tuple2<>(pair._2(), pair._1()));
```

Exercises

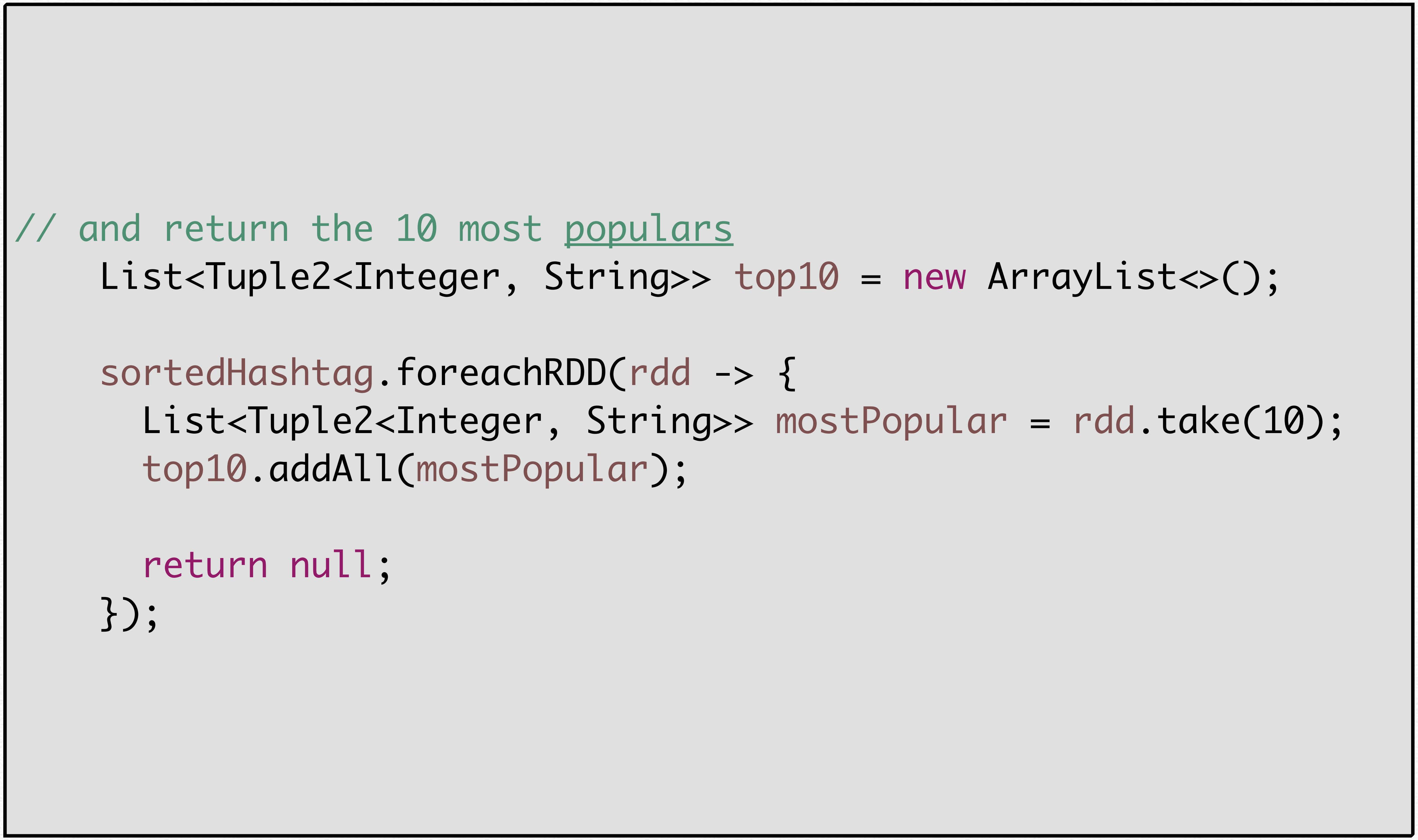
SPARK Streaming: StreamingOnTweets



```
// Then sort the hashtags
    JavaPairDStream<Integer, String> sortedHashtag =
            hashtagMention.transformToPair(
                    hashtagRDD -> hashtagRDD.sortByKey(false));
```

Exercises

SPARK Streaming: StreamingOnTweets



```
// and return the 10 most populars
    List<Tuple2<Integer, String>> top10 = new ArrayList<>();

    sortedHashtag.foreachRDD(rdd -> {
      List<Tuple2<Integer, String>> mostPopular = rdd.take(10);
      top10.addAll(mostPopular);

      return null;
    });
```

Exercises

SPARK Streaming: StreamingOnTweets

```
// we need to tell the context to start running the computation we
// have setup. It won't work if you don't add this!
    jssc.start();
    jssc.awaitTermination();

    return "Most popular hashtag :" + top10;
  }
```

Exercises

SPARK Streaming: execution

```
$/bin/spark-submit
                   --class "streaming.StreamingOnTweets"
                   --master local[4]
                   --packages "org.apache.spark:spark-streaming-twitter_2.10:1.5.1"
                   --jars $HOME/spark-in-practice-1.0.jar
                          $HOME/twitter4j-core-3.0.3.jar
```

Spark Streaming

04/05/2020 - Big Data