Data Schools

Day 3 - Module 2:
Spark Streaming

Introduction

Learning Objectives

Upon successful completion of this lecture, you will have a good understanding of Spark Streaming, motivation and programming model.

Spark Streaming

Real-time Data Streaming

- Real-time data from sensors, IoT devices, log files, social networks, etc. needs to be closely monitored and immediately processed.
- Therefore, for real-time data analytics, we need a highly scalable, reliable, and fault-tolerant data streaming engine.

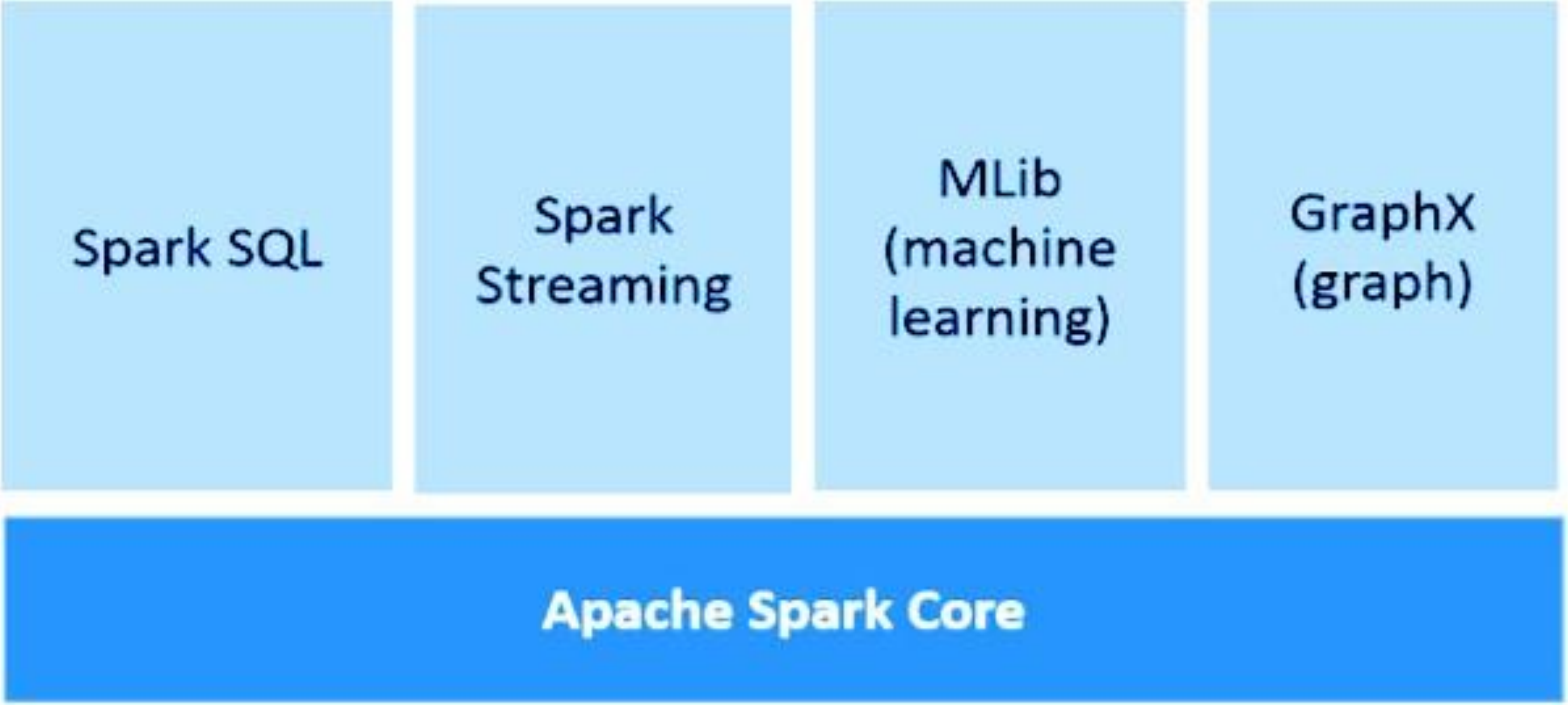
Data Streaming

- Data streaming is a way of collecting data continuously in real-time from multiple data sources in the form of data streams. Datastream can be thought of as a table that is continuously being appended.

- Data streaming is essential for handling massive amounts of live data. Such data can be from a variety of sources like online transactions, log files, sensors, in-game player activities, etc.

- There are various real-time data streaming techniques like Apache Kafka, Spark Streaming, Apache Flume etc. We will discuss data streaming using Spark Streaming

Spark Streaming



Spark SQL

Spark Streaming

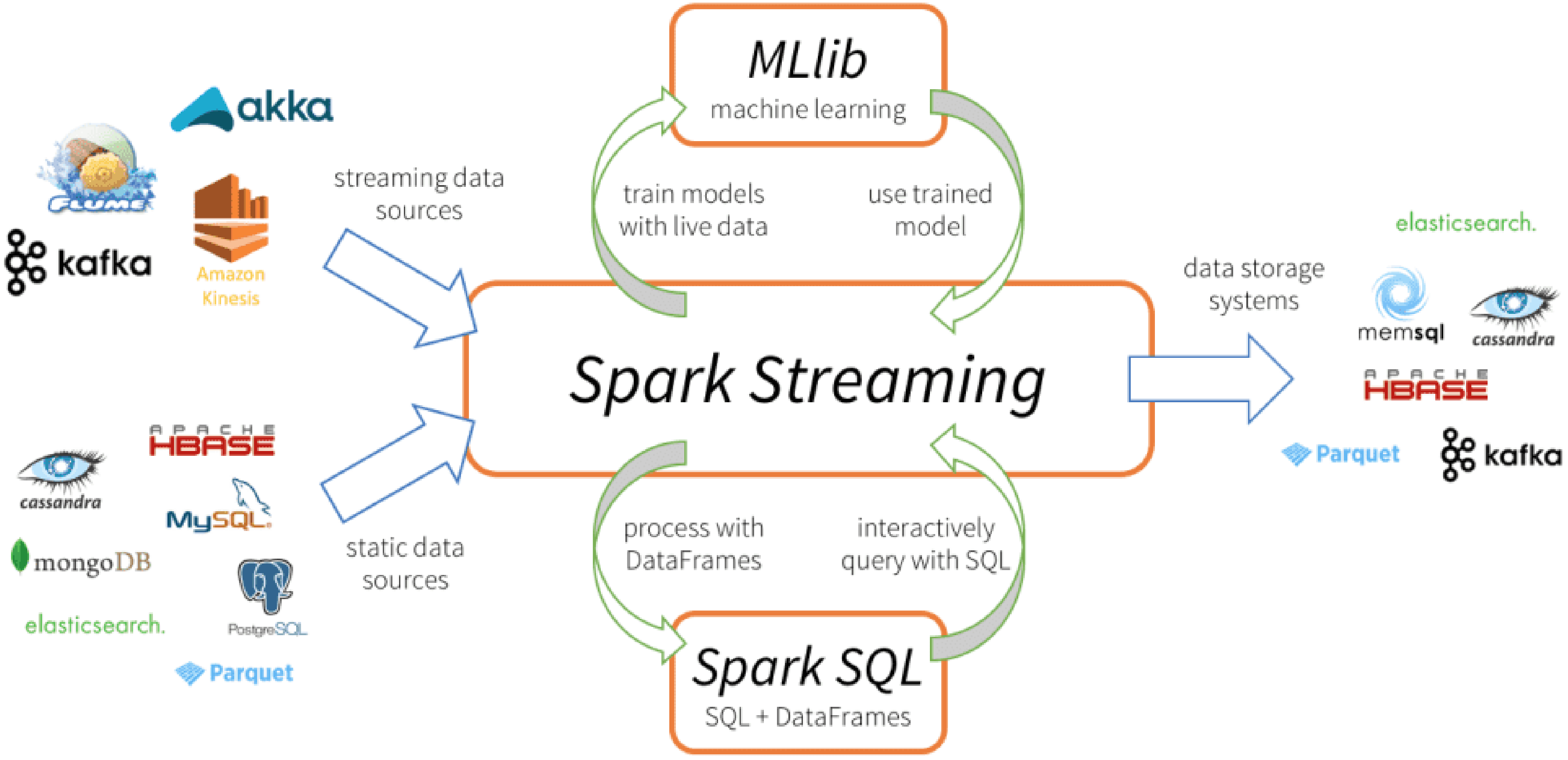
MLib (machine learning)

GraphX (graph)

Apache Spark Core

Spark Streaming

1. Data Sources
2. Streaming Engine
3. Output Sinks



Advantages of Spark Streaming

- Unified streaming framework for all data processing tasks(including machine learning, graph processing, SQL operations) on live data streams.

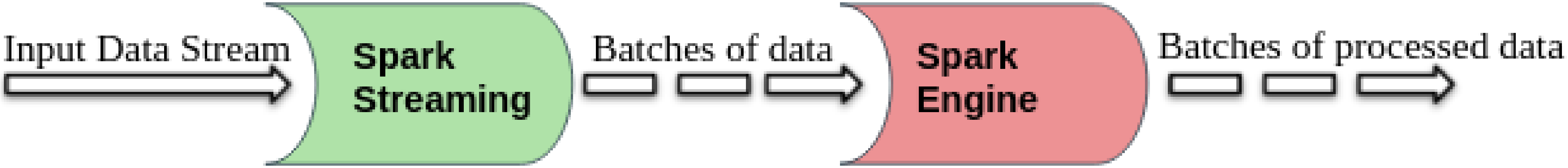
- Dynamic load balancing and better resource management by efficiently balancing the workload across the workers and launching the task in parallel.

- Deeply integrated with advanced processing libraries like Spark SQL, MLlib, GraphX.

- Faster recovery from failures by re-launching the failed tasks in parallel on other free nodes.

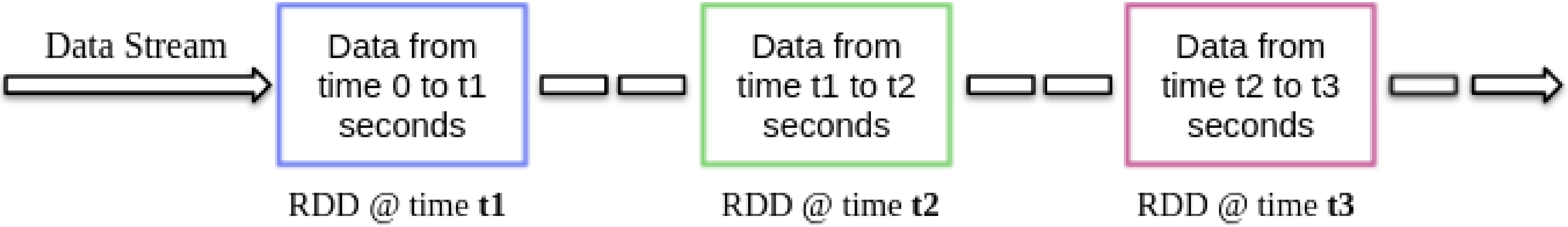
Spark Streaming Fundamentals

- Spark Streaming divides the live input data streams into batches which are further processed by Spark engine



DStream (Discretized Stream)

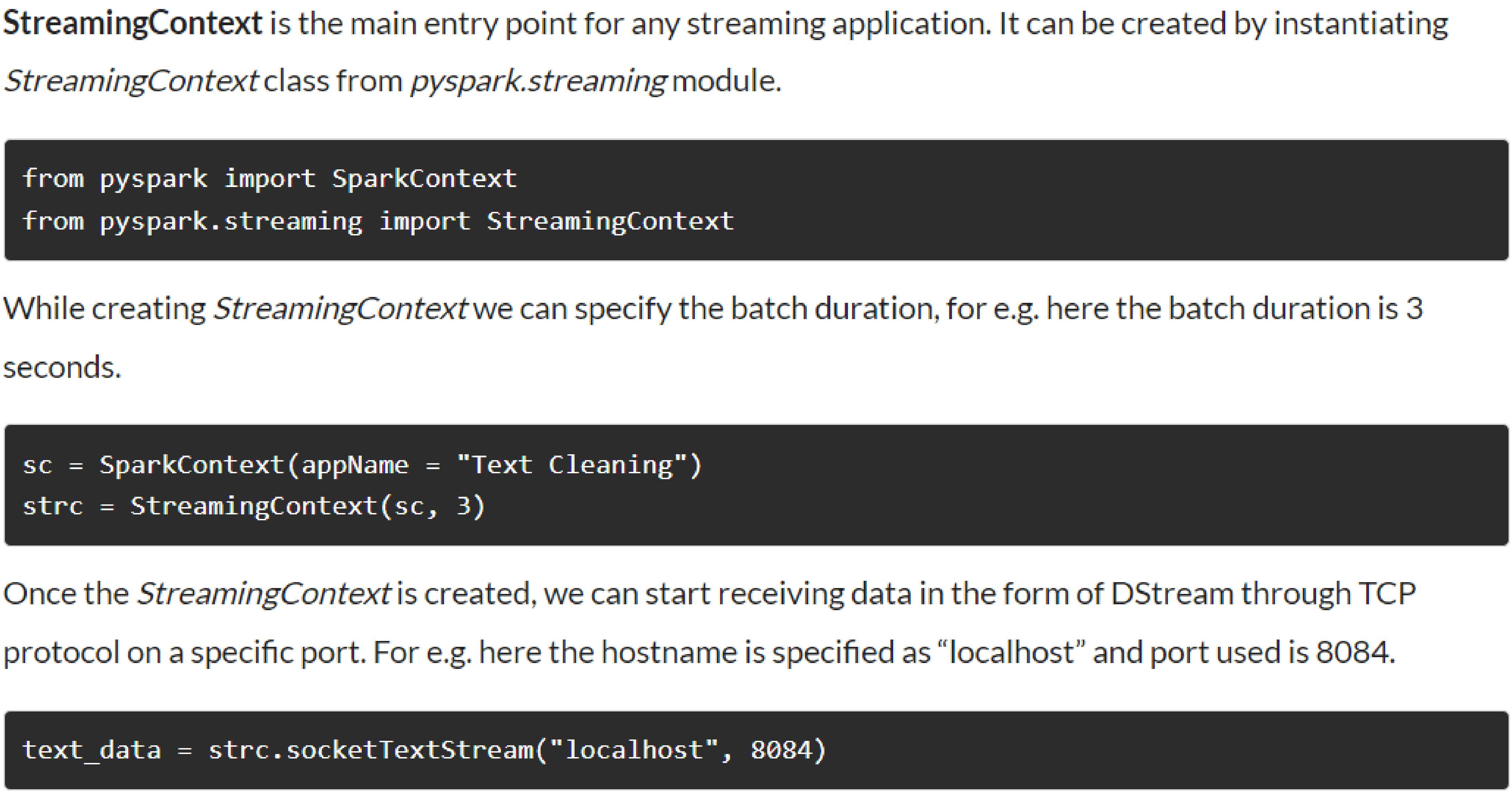
- DStream is a high-level abstraction provided by Spark Streaming, basically, it signifies the continuous stream of data.
- Internally DStream is a sequence of RDDs


Data Stream
Data from time 0 to t1 seconds
RDD @ time t1
Data from time t1 to t2 seconds
RDD @ time t2
Data from time t2 to t3 seconds
RDD @ time t3

Sample Application

- As we discussed earlier, Spark Streaming also allows receiving data streams using TCP sockets.
- So let’s write a simple streaming program to receive text data streams on a particular port, perform basic text cleaning (like white space removal, stop words removal, lemmatization, etc.), and print the cleaned text on the screen.

1. Creating Streaming Context and Receiving data streams



StreamingContext is the main entry point for any streaming application. It can be created by instantiating *StreamingContext* class from *pyspark.streaming* module.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
```

While creating *StreamingContext* we can specify the batch duration, for e.g. here the batch duration is 3 seconds.

```
sc = SparkContext(appName = "Text Cleaning")
strc = StreamingContext(sc, 3)
```

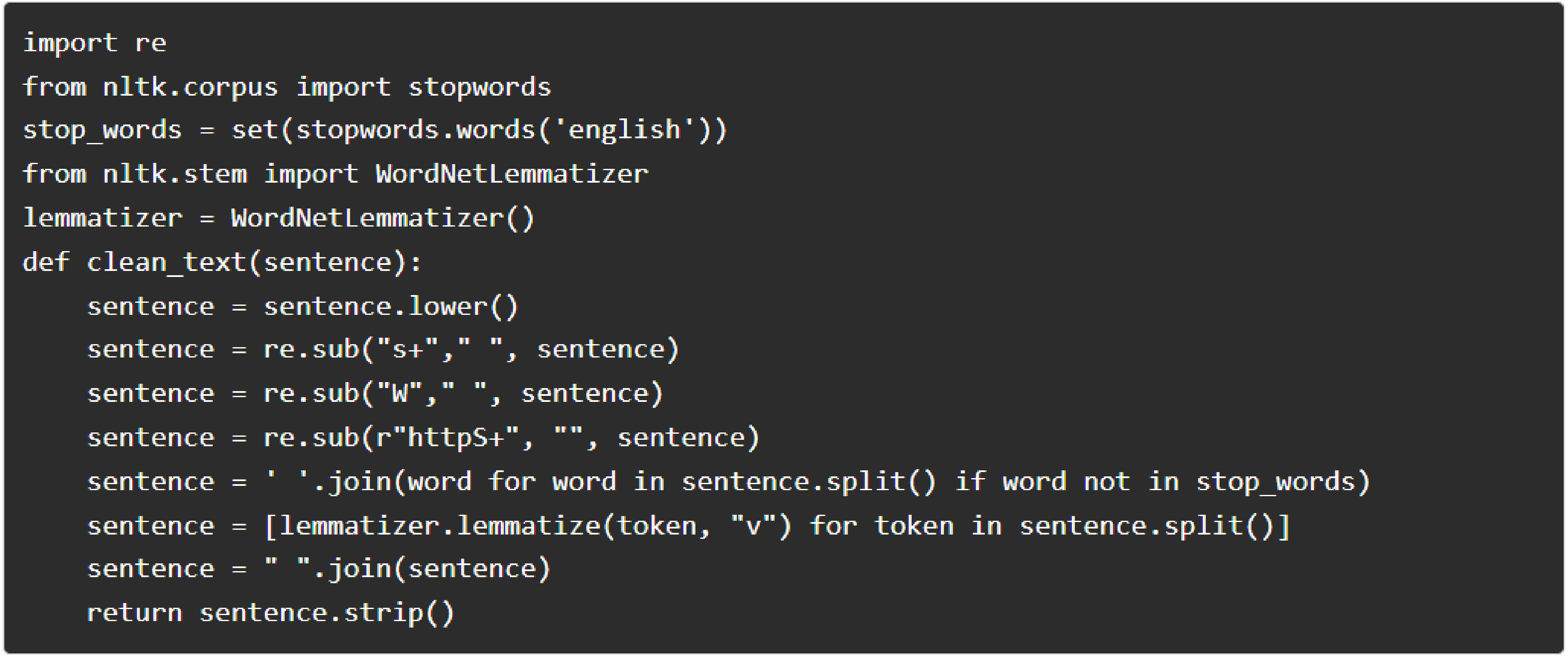
Once the *StreamingContext* is created, we can start receiving data in the form of DStream through TCP protocol on a specific port. For e.g. here the hostname is specified as "localhost" and port used is 8084.

```
text_data = strc.socketTextStream("localhost", 8084)
```

2. Performing operations on data streams

After creating a *DStream* object, we can perform operations on it as per the requirement. Here, we wrote a custom text cleaning function.

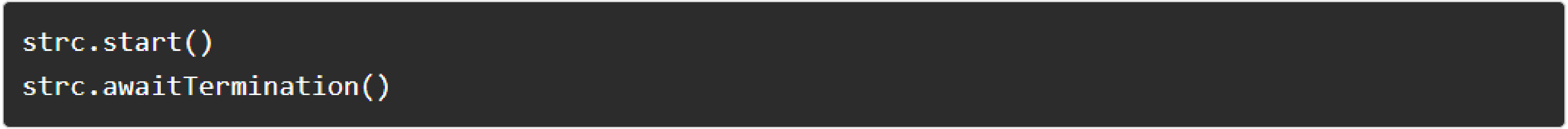
This function first converts the input text into lower case, then removes extra spaces, non-alphanumeric characters, links/URLs, stop words, and then further lemmatizes the text using the NLTK library.



```
import re
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def clean_text(sentence):
    sentence = sentence.lower()
    sentence = re.sub("s+"," ", sentence)
    sentence = re.sub("W"," ", sentence)
    sentence = re.sub(r"httpS+", "", sentence)
    sentence = ' '.join(word for word in sentence.split() if word not in stop_words)
    sentence = [lemmatizer.lemmatize(token, "v") for token in sentence.split()]
    sentence = " ".join(sentence)
    return sentence.strip()
```

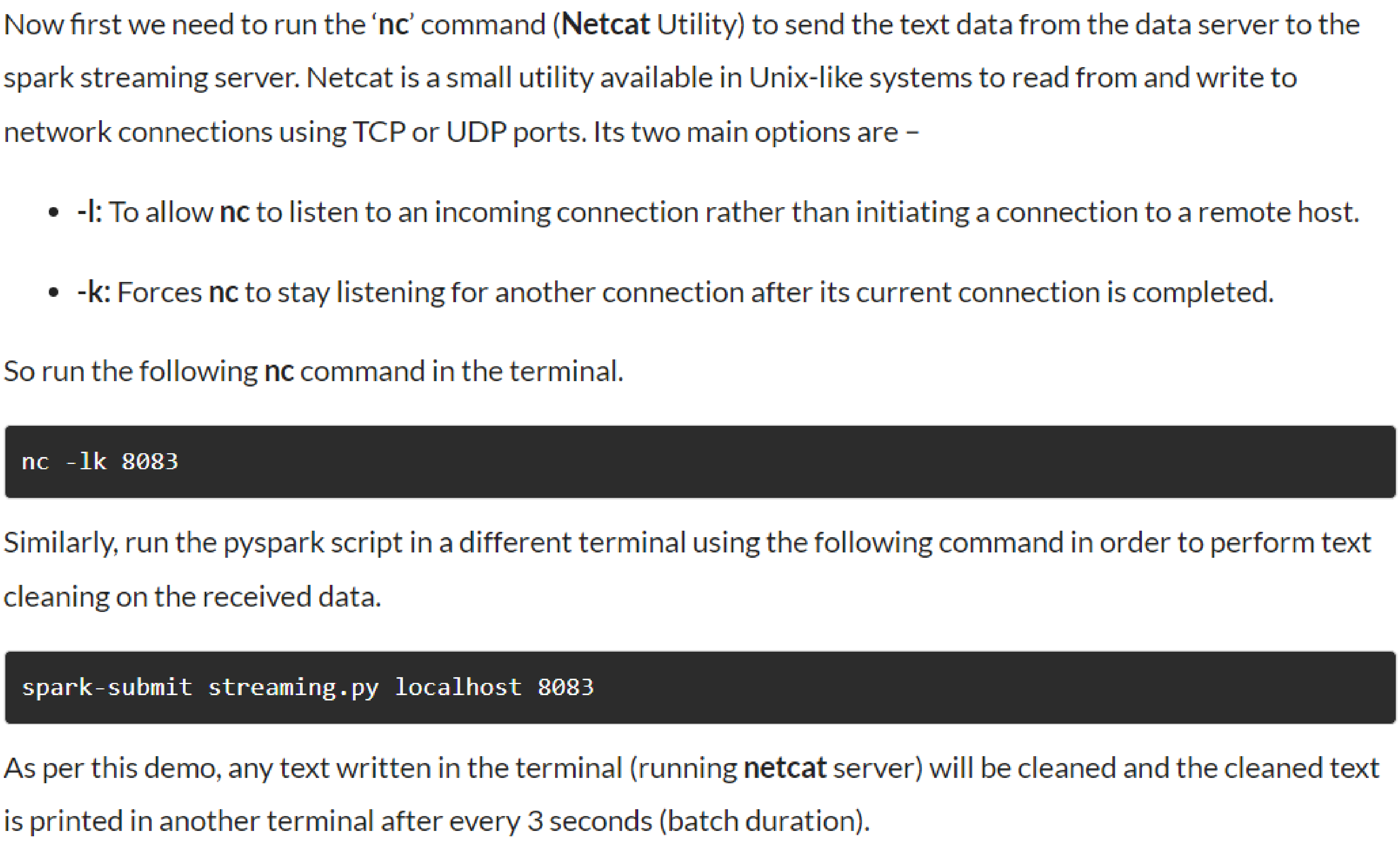
3. Starting the Streaming service

The streaming service has not started yet. Use the *start()* function on top of the *StreamingContext* object to start it and keep on receiving streaming data until the termination command (Ctrl + C or Ctrl + Z) is not received by *awaitTermination()* function.



```
strc.start()
strc.awaitTermination()
```

Running the Application



Now first we need to run the ‘**nc**’ command (**Netcat** Utility) to send the text data from the data server to the spark streaming server. Netcat is a small utility available in Unix-like systems to read from and write to network connections using TCP or UDP ports. Its two main options are –

- **-l:** To allow **nc** to listen to an incoming connection rather than initiating a connection to a remote host.
- **-k:** Forces **nc** to stay listening for another connection after its current connection is completed.

So run the following **nc** command in the terminal.

```
nc -lk 8083
```

Similarly, run the pyspark script in a different terminal using the following command in order to perform text cleaning on the received data.

```
spark-submit streaming.py localhost 8083
```

As per this demo, any text written in the terminal (running **netcat** server) will be cleaned and the cleaned text is printed in another terminal after every 3 seconds (batch duration).

Code

```
#!/usr/bin/env python
# coding: utf-8

from pyspark import SparkContext
from pyspark.streaming import StreamingContext


sc = SparkContext(appName = "Text Cleaning")
strc = StreamingContext(sc, 3)


text_data = strc.socketTextStream("localhost", 8083)


import re
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()


def clean_text(sentence):
    sentence = sentence.lower()
    sentence = re.sub("\s+"," ", sentence)
    sentence = re.sub("\W"," ", sentence)
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = ' '.join(word for word in sentence.split() if word not in stop_words)
    sentence = [lemmatizer.lemmatize(token, "v") for token in sentence.split()]
    sentence = " ".join(sentence)
    return sentence.strip()


cleaned_text = text_data.map(lambda line: clean_text(line))
cleaned_text.pprint()

strc.start()
strc.awaitTermination()
```

ANY
QUESTIONS