





Intro to Apache Spark

EECS 4415
Big Data Systems

Tilemachos Pechlivanoglou
tipech@eecs.yorku.ca



Apache Spark

- Spark is a cluster computing engine.
- Provides high-level API in Scala, Java, Python and R.
- Provides high level tools:
 - *Spark SQL.*
 - *MLib.*
 - *GraphX.*
 - *Spark Streaming.*





RDDs

- The basic abstraction in Spark is the RDD.
- Stands for: **R**esilient **D**istributed **D**ataset.
- A **collection of items**, with source:
 - *Hadoop (HDFS).*
 - *JDBC.*
 - *ElasticSearch.*
 - *others...*



RDD concepts



Main concepts regarding RDD:

- Partitions.
- Dependencies.
- Lazy computation



RDD partitions



- An RDD is partitioned.
- A partition is usually computed on a different process
 - *(usually on a different machine).*
- This is the implementation of the distributed part of the RDD.

RDD dependency

- RDDs can depend on other RDDs.
- RDD calculations are lazy
 - *map operation on RDD gives new RDD which depends on original*
 - *new RDD only contains meta-data (i.e., the computing function)*
- Flow is only computed on a specific command
 - *i.e. when we calculate something final (reduce)*

Lazy RDDs & dependency

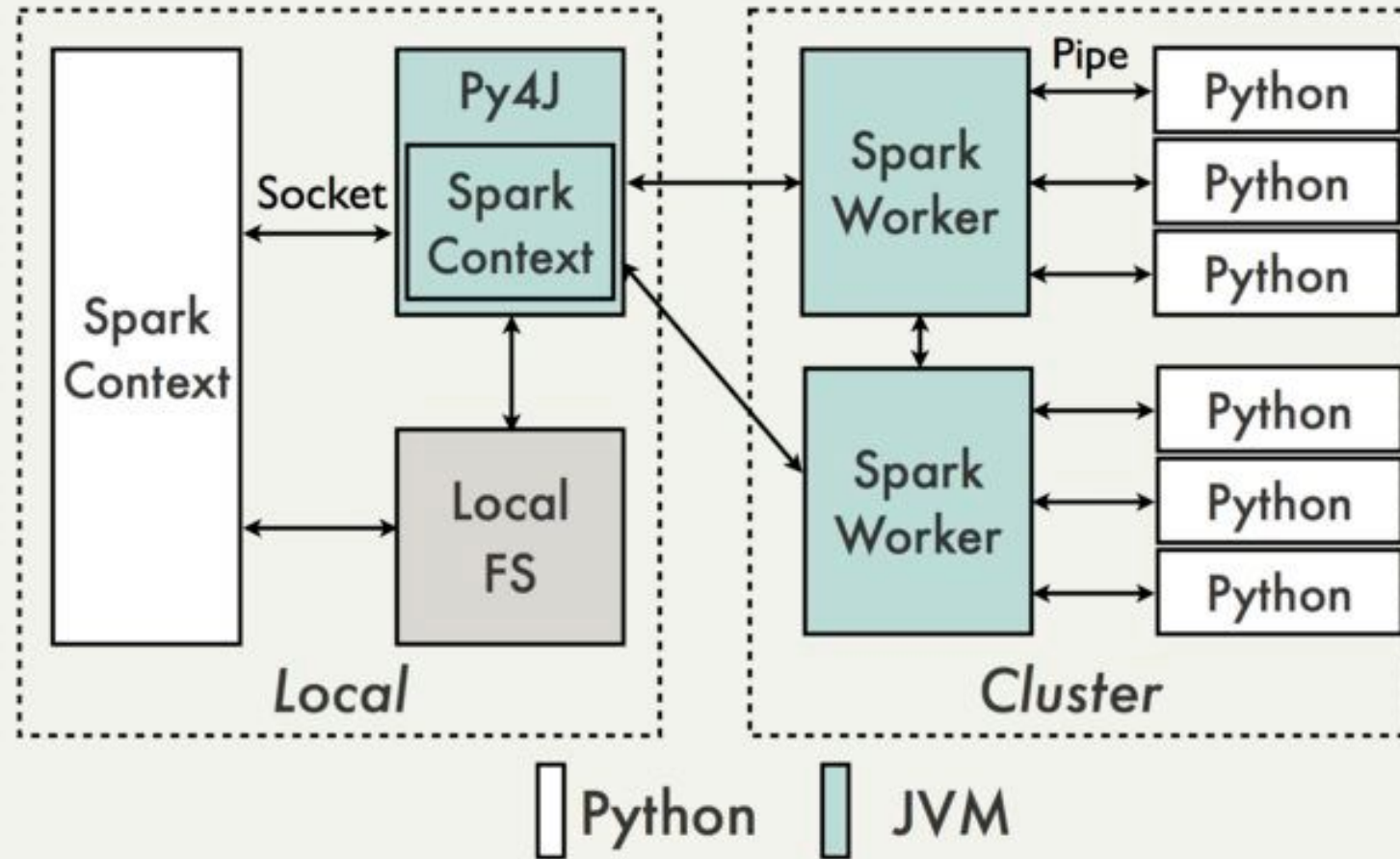
- RDDs can depend on other RDDs.
- RDD calculations are lazy
 - *map operation on RDD gives new RDD which depends on original*
 - *new RDD only contains meta-data (i.e., the computing function)*
- Flow is only computed on a specific command
 - *i.e. when we calculate something final (reduce)*



Spark structure

- Driver:
 - *Executes the main program*
 - *Creates the RDDs*
 - *Collects the results*
- Executors:
 - *Execute the RDD operations*
 - *Participate in the shuffle*

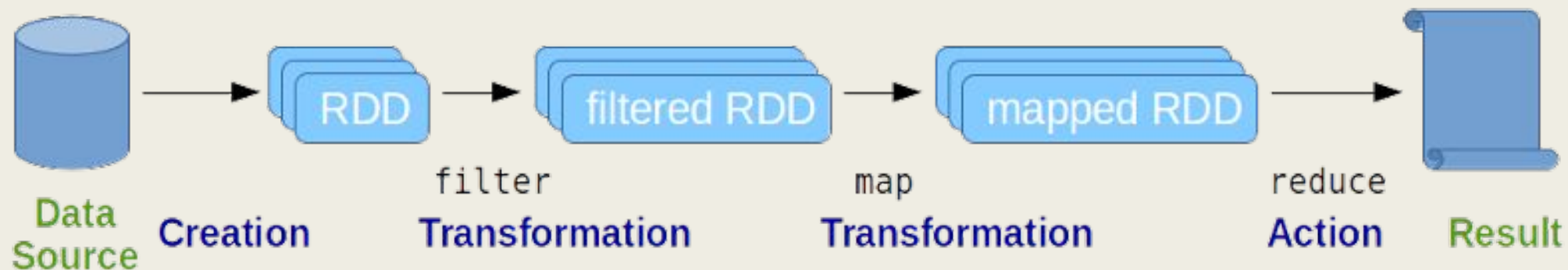
Data Flow



Spark flow

■ Normal process:

- *Data ingestion:* turn any source of data to RDDs
- *Transformations:* modify the RDDs in some way
- *Final actions:* evaluate the RDDs and return some result



RDD creation

- Spark supports reading files, directories, streams, etc.
- Some out-of-the-box methods:
 - `textFile` - *retrieving an* `RDD[String]`
 - `sequenceFile` - *Hadoop sequence files* `RDD[(K, V)]`
 - `socketTextStream` - *text stream* `RDD[String]`



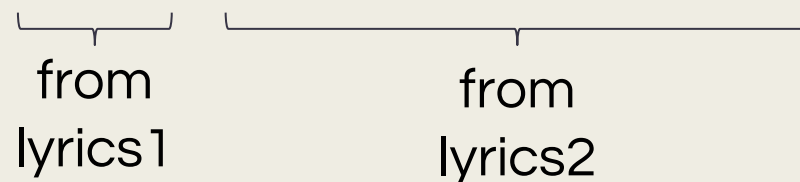
RDD transformation



- Transformations are divided to two main types:
 - *Those who shuffle*
 - *Those who don't*
- Remember these are lazy operations!

RDD transformations, no shuffle

- **map(func):**
 - return new RDD by passing each element through a function
- **filter(func):**
 - return new RDD by selecting elements on which func returns true
- **flatMap(func):**
 - similar to map, but each input item is mapped to 0 or more output items
 - (so func should return a Seq rather than a single item)
 - e.g. (lyrics1, lyrics2) -> flatmap -> (word1, word2, word3, word4)



RDD transformations, shuffle

- Shuffle operations repartition the data across the network.
- Can be very expensive operations in Spark.
- You must be aware where and why shuffle happens.
- Order is not guaranteed inside a partition.

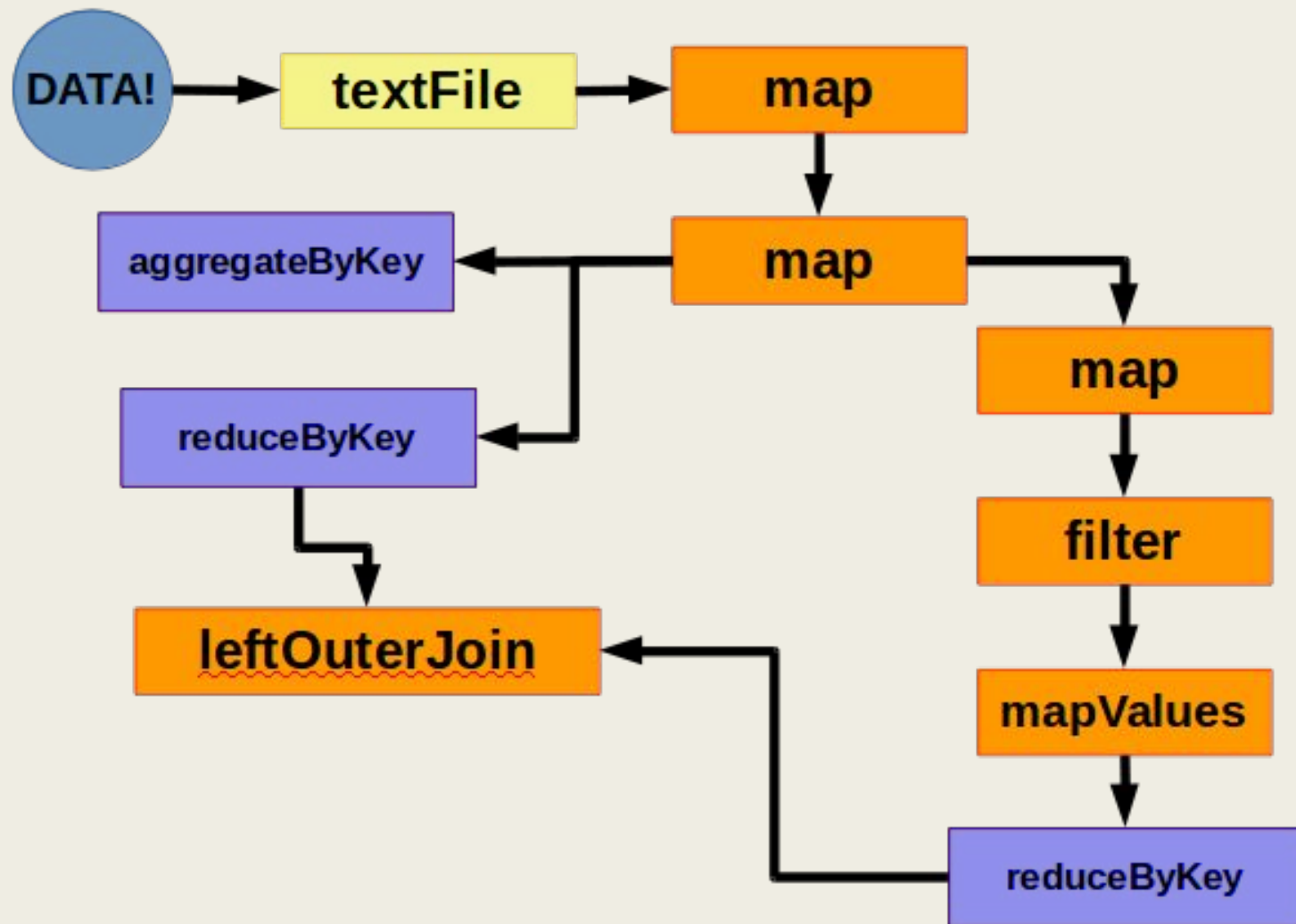
- Popular operations that cause shuffle are:
 - *groupBy*, reduceBy*, sort*, aggregateBy* and join/intersect RDDs*

Final actions (1)

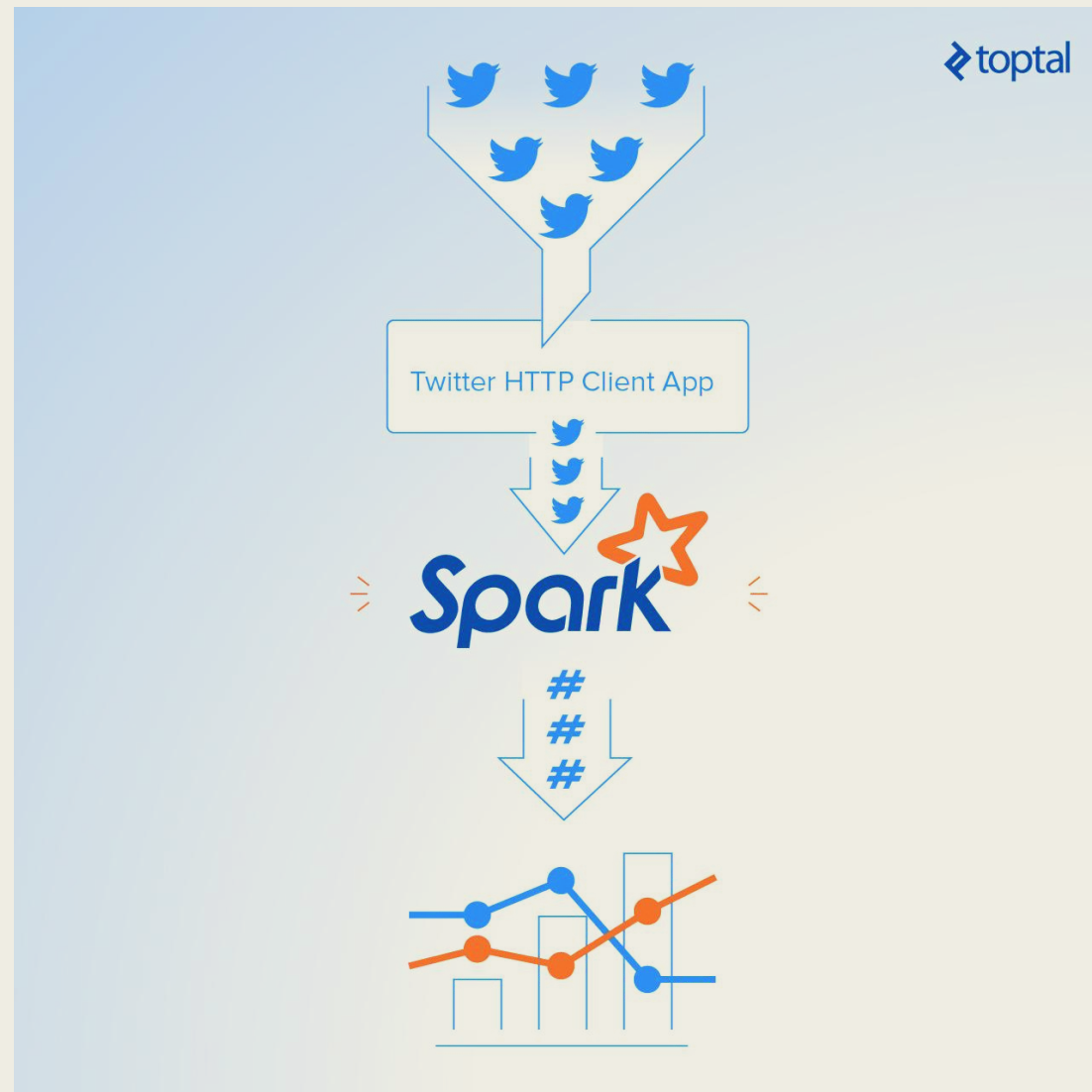
- The following (selected) methods evaluate the RDD (not lazy):
 - **collect()** – *returns a list containing all the elements of the RDD*
main RDD evaluation method
 - **count()** – *returns the number of the elements in the RDD*
 - **first()** – *returns the first element of the RDD*
 - **foreach(f)** – *performs a function on each element of the RDD*
 - **isEmpty**
 - **max/min**
 - **reduce((T, T) => T)** – *parallel reduction.*

Final actions (2)

- More evaluating methods
 - **take (n)** – *returns the first n elements*
 - **takeSample ()**
 - **takeOrdered (n)** – *returns the first (smallest) n elements*
 - **top (n)** – *returns the first (largest) n elements*
 - **countByKey** – *for pair RDDs*
 - **save*File**



An example workflow



Demo streaming Twitter app

Running the demo Twitter app

- Demo is executed in two different Docker containers
 - *one responsible for connecting to Twitter stream and forwarding it locally*
 - *one responsible for getting the local stream and processing it in Spark*
 - *we make them talk to each other by “linking” them*
- Running `twitter_app.py`
 - `docker run -it -v $PWD:/app --name twitter -w /app python bash`
 - `pip install -U git+https://github.com/tweepy/tweepy.git`
 - `python twitter_app.py`
- Running `spark_app.py`
 - `docker run -it -v $PWD:/app --link twitter:twitter eecsyorku/eecs4415`
 - `spark-submit spark_app.py`

Installs latest version,
previous one has a bug

Twitter app credentials

- Twitter requires app developer account for access to stream.
 - *Normally requires applying for it*
 - *This is the best option*
- If that isn't possible, you can use credentials below:
 - *May cause limiting issues with too many people running at the same time*

```
ACCESS_TOKEN = '2591998746-Mx8ZHsXJHzIxAaD2IxYfmzYuL3pYNVnvWoHZgR5'
```

```
ACCESS_SECRET = 'LJDvEa0jL7QJXxq10NVrULTAniLobe2TAAlnBdXRfm1xF'
```

```
CONSUMER_KEY = 'ZAPfZLcBhYEBCeRSAK5PqkTT7'
```

```
CONSUMER_SECRET = 'M81KvgaiicyJIaQegdGXcdKDeZrSsJz4AVrGv3yoFwuItQQPMay'
```

Thank you!

Based on:

<http://trainologic.com/wp-content/uploads/2017/06/SparkForDataScienceMeetup1.pptx>
<https://www.toptal.com/apache/apache-spark-streaming-twitter>