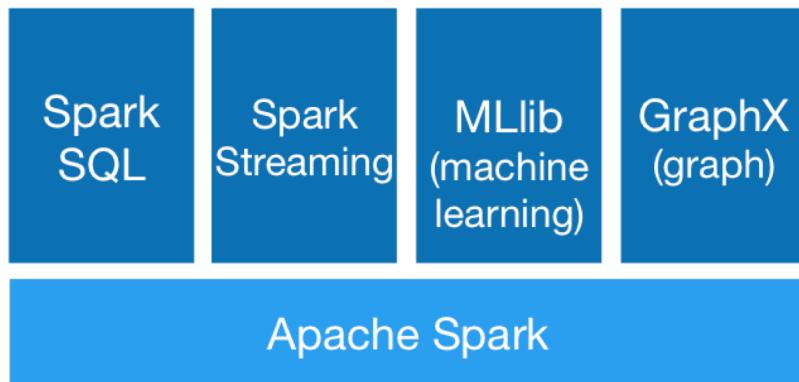


# AN INTRODUCTION TO SPARK AND TO ITS PROGRAMMING MODEL

---

# Apache Spark

Stack of libraries including



- Analytics engine for large-scale data processing
- High performance for batch and streaming data, using a DAG scheduler, a query optimizer, and an execution engine
- High-level APIs  
Java, Scala, Python and R

# Apache Spark

- Latest release - 2.4.0- <https://spark.apache.org/docs/latest/index.html>
- Quick start – <https://spark.apache.org/docs/latest/quick-start.html>
- Downloads - <https://spark.apache.org/downloads.html>

# Spark Ecosystem

Spark  
SQL

Spark  
Streaming

MLlib  
(machine  
learning)

GraphX  
(graph)

Apache Spark

# Spark Core

- Contains the basic functionality for
  - task scheduling,
  - memory management,
  - fault recovery,
  - interacting with storage systems,
  - and more.
- Defines the Resilient Distributed Data sets (RDDs)
  - main Spark programming abstraction.

# Spark SQL

- For working with structured data.
- View datasets as relational tables
- Define a schema of columns for a dataset
- Perform SQL queries
- Supports many sources of data
  - Hive tables, Parquet and JSON
- DataFrame



# Spark Streaming

- Data analysis of streaming data
  - e.g. log files generated by production web servers
- Aimed at high-throughput and fault-tolerant stream processing
- Dstream → Stream of datasets that contain data from certain interval

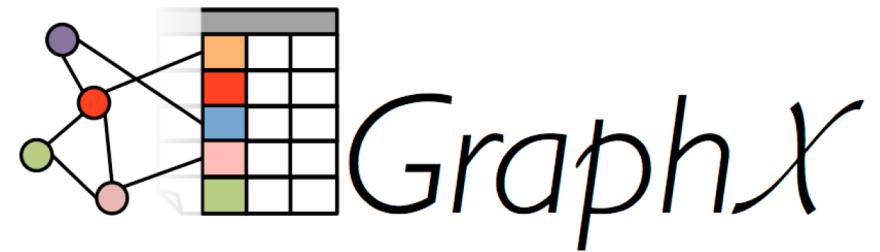


# Spark MLlib

- MLlib is a library that contains common Machine Learning (ML) functionality:
  - Basic statistics
  - Classification (Naïve Bayes, decision trees, LR)
  - Clustering (k-means, Gaussian mixture, ...)
  - And many others!
- All the methods are designed to scale out across a cluster.

# Spark GraphX

- Graph Processing Library
- Defines a graph abstraction
  - Directed multi-graph
  - Properties attached to each edge and vertex
  - RDDs for edges and vertices
- Provides various operators for manipulating graphs (e.g. subgraph and mapVertices)



# Programming with Python Spark (pySpark)

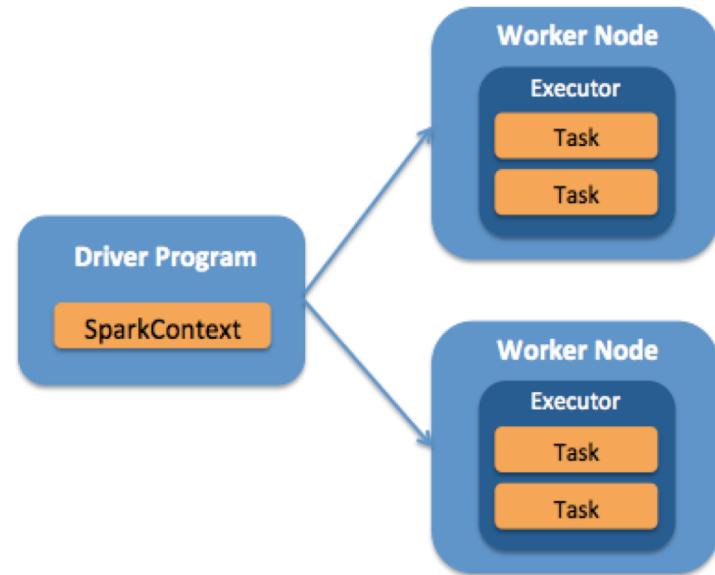
- We will use Python's interface to Spark called **pySpark**
- A **driver program** accesses the Spark environment through a **SparkContext** object
- The **key concept** in Spark are datasets called **RDDs** (**Resilient Distributed Dataset**)
- Basic idea: We load our data into RDDs and perform some **operations**

# Spark Deployment Options

- Interactive mode for testing and development
  - **On local machine using shared memory and one or more cores**
  - **Or interacting with cluster**
- Job submission to a cluster manager
  - **Spark Standalone cluster**
  - Hadoop YARN
  - Apache Mesos
  - Kubernetes (new!) - Spark 2.4 includes many enhancements for the Kubernetes integration

# Programming environment - Spark concepts

- Spark application execution involves runtime concepts such as:
  - **Driver**, executor, tasks, etc.
- Spark application maps to a **single driver process** and a set of **executor processes**.
- The **driver** manages the job flow and **schedules tasks**
- The driver is located:
  - Cluster mode – Client process
  - Interactive mode – The shell is the driver process



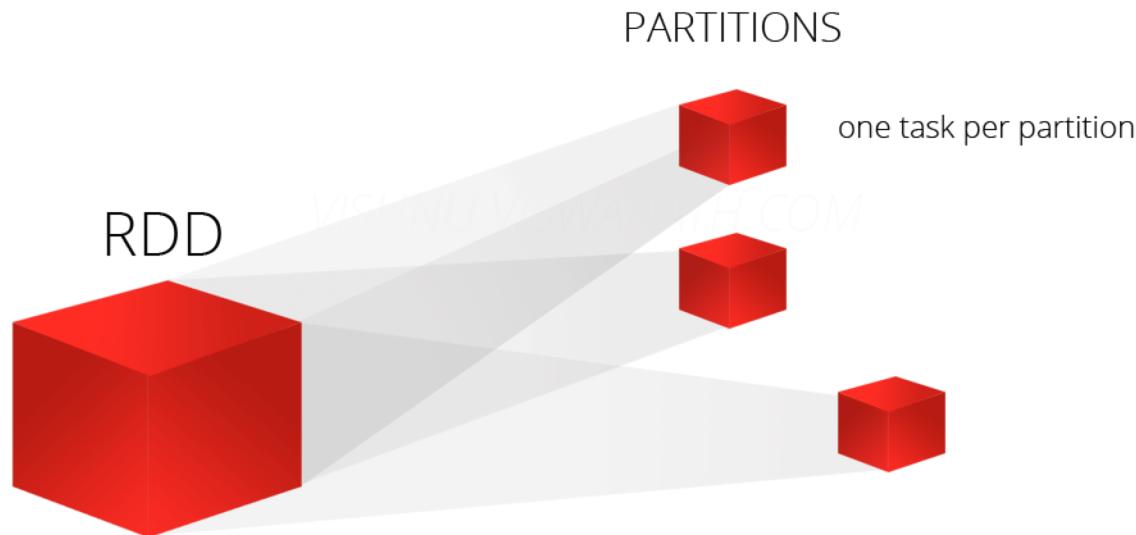
The **executors** are responsible for executing work and storing data in the cache.

# Programming environment - Spark concepts

- Driver programs access Spark through a **SparkContext** object → represents a connection to the computing cluster.
- In a *shell* the **SparkContext** is created for you (variable **sc**).
- **Spark context** is the **heart** of a Spark application:
  - Spark context sets up the internal services and establishes a connection to a Spark execution environment.
- Once a Spark Context is created you can used it to create:
  - **RDDs (Resilient Distributed Datasets )**, accumulators, etc.
  - Run jobs.

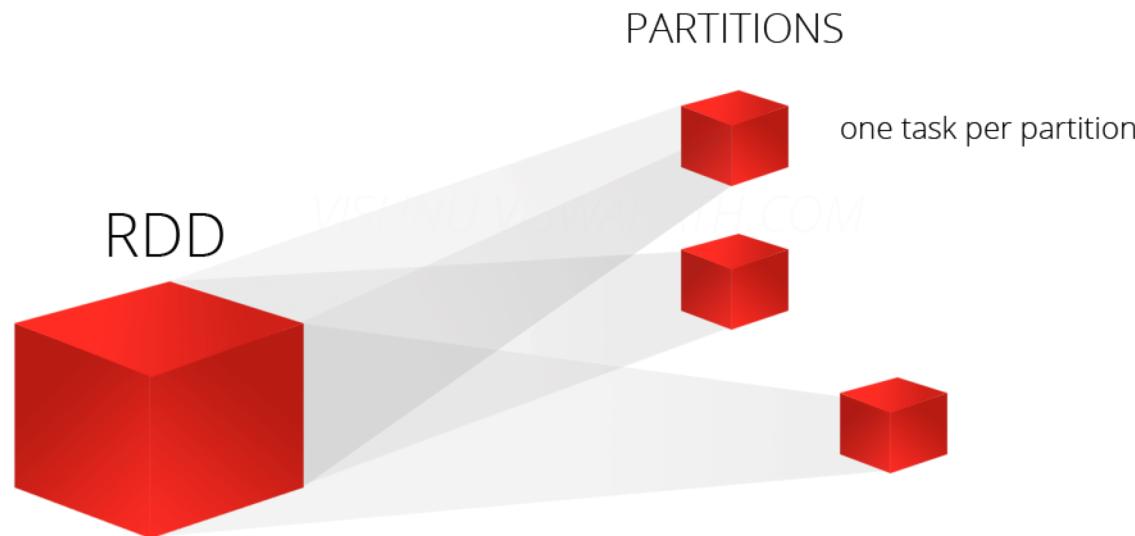
# Resilient Distributed Datasets (RDD)

- **RDD** is a fundamental data structure of **Spark**.
- It represents data or transformations on data
- It is distributed collection of items - partitions
- Read-only → they are immutable
- Enables operations to be performed in parallel



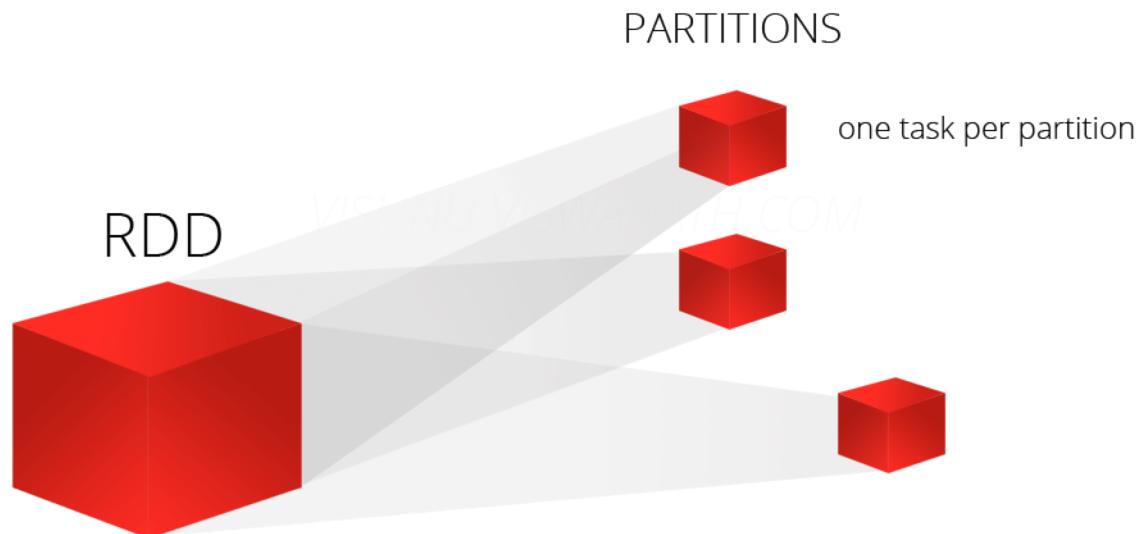
# Resilient Distributed Datasets (RDD)

- Fault tolerant:
  - Lineage of data is preserved, so data can be re-created on a new node at any time
- Caching dataset in memory
  - different storage levels available
  - fallback to disk possible



# Resilient Distributed Datasets (RDD)

- There are two ways to create RDDs:
  - *parallelizing* an existing collection in your driver program
  - *referencing* a dataset in :
    - an external storage system (e.g. shared filesystem, HDFS, Hbase)
    - data source offering a Hadoop InputFormat



# Resilient Distributed Datasets (RDD)

- Parallelize a collection

```
# Parallelize in Python  
wordsRDD = sc.parallelize(["fish", "cats", "dogs"])
```

- Take an existing in-memory collection and pass it to SparkContext's parallelize method
- Not generally used outside of prototyping and testing since it requires entire dataset in memory on one machine

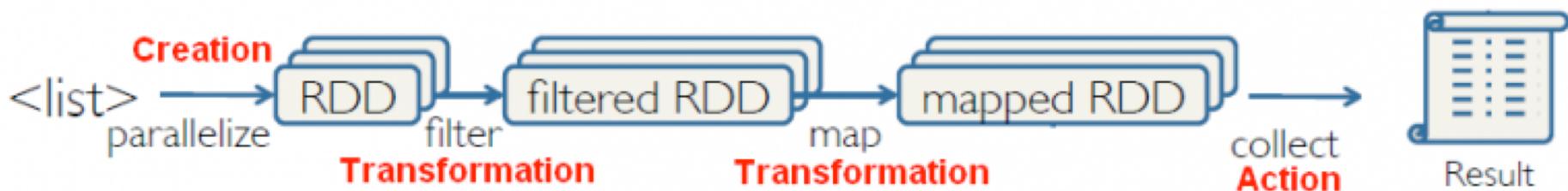
- Read from File

```
# Read a local txt file in Python  
linesRDD = sc.textFile("/path/to/README.md")
```

- There are other methods to read data from HDFS, C\*, S3, HBase, etc.

# Programming with RDDs

- All work is expressed as either:
  - creating new RDDs
  - transforming existing RDDs
  - calling operations on RDDs to compute a result.



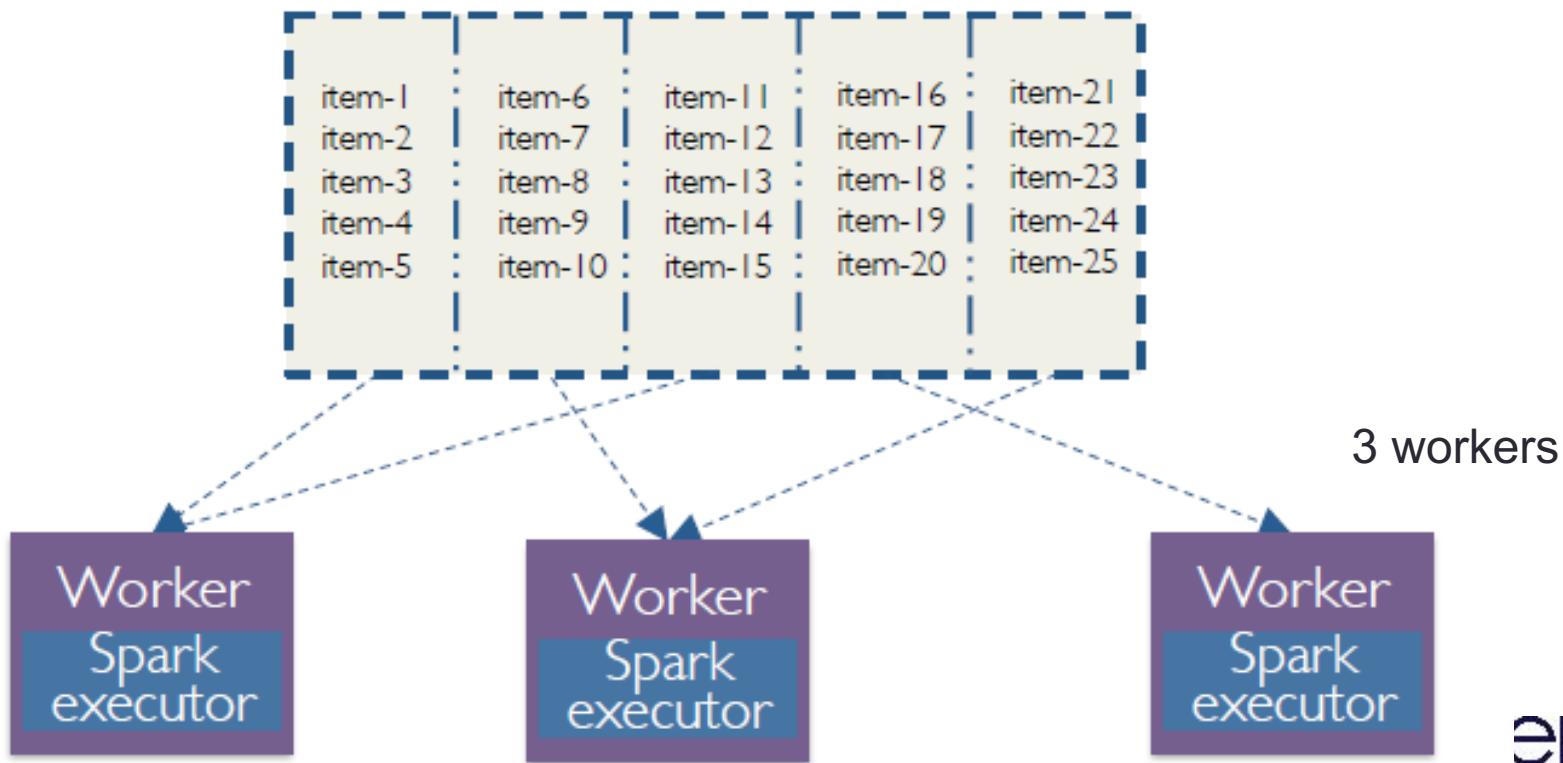
- Distributes the data contained in RDDs across the nodes (executors) in the cluster and parallelizes the operations.
- Each RDD is split into multiple **partitions**, which can be computed on different nodes of the cluster.

# Partitions

RDD are a collection of various data that could be so big in size, that they might not fit into a single node.

Apache Spark automatically partitions RDDs and distributes the partitions across different nodes.

RDD split into 5 partitions



## Note about partitions

- RDDs get partitioned automatically without programmer intervention.
- But they can be adjusted → Optimization techniques

<https://www.talend.com/blog/2018/03/05/intro-apache-spark-partitioning-need-know/>

# First Spark Program

```
sc = SparkContext(master="local[*]")
```

Context

```
lines = sc.textFile("README.md", 4)
```

RDD-1

```
lines.count()
```

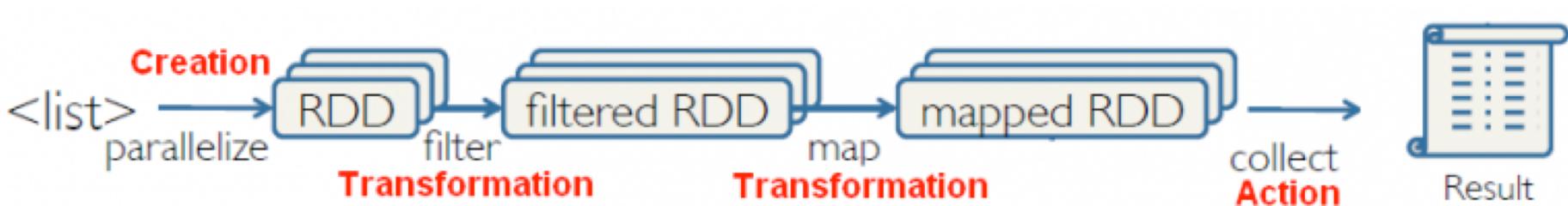
```
pythonLines = lines.filter(lambda line : "Python" in line)
```

```
pythonLines.first()
```

RDD-2

# RDD operations

- Once created, RDDs offer two types of operations:
  - Transformations**
    - They create a new dataset from an existing one
    - They include *map*, *filter*, *join*
    - Lazy operations → they do not compute their results right away
  - Actions**
    - actions include *count*, *collect*, *save*
    - return a result to the **driver** program or write it to storage



The transformations are only computed when an action requires a result to be returned to the driver program.

# Transformation vs Actions

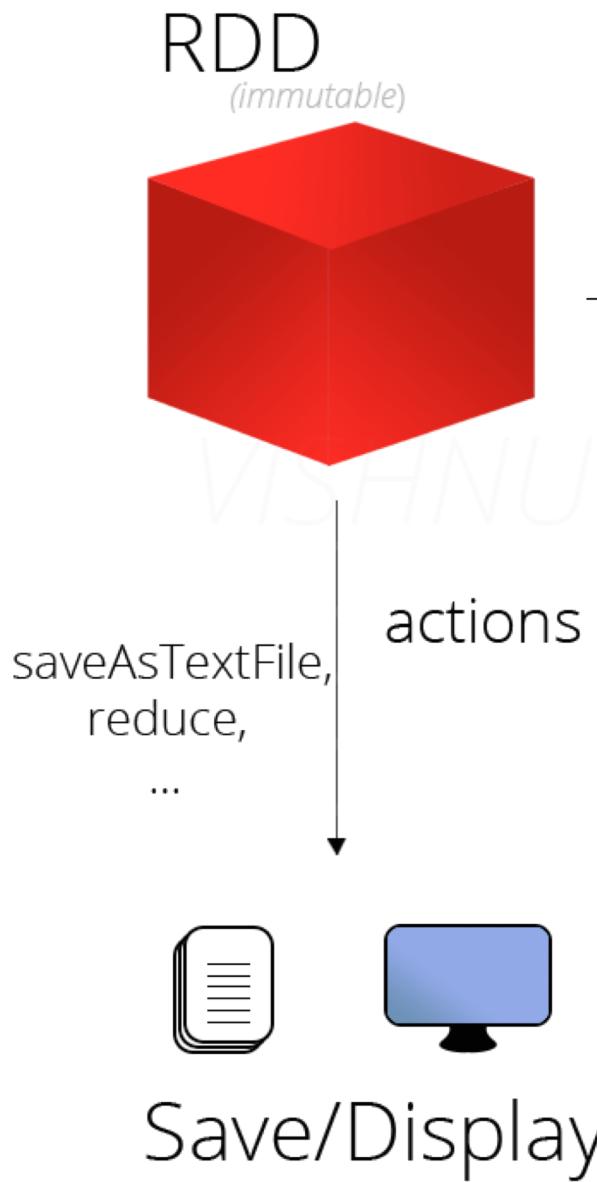
## Transformations

- map (func)
- flatMap(func)
- filter(func)
- groupByKey()
- reduceByKey(func)
- mapValues(func)
- sample(...)
- union(other)
- distinct()
- sortByKey()
- ...

## Actions

- reduce(func)
- collect()
- count()
- first()
- take(n)
- saveAsTextFile(path)
- countByKey()
- foreach(func)
- ...

<https://spark.apache.org/docs/latest/rdd-programming-guide.html>



**map** is a transformation that passes each dataset element through a function and returns a **new RDD representing the results**.

**reduce** is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program

# First Spark Program

```
sc = SparkContext(master="local[*]")
```

Context

```
lines = sc.textFile("README.md", 4)
```

RDD-1

```
lines.count()
```

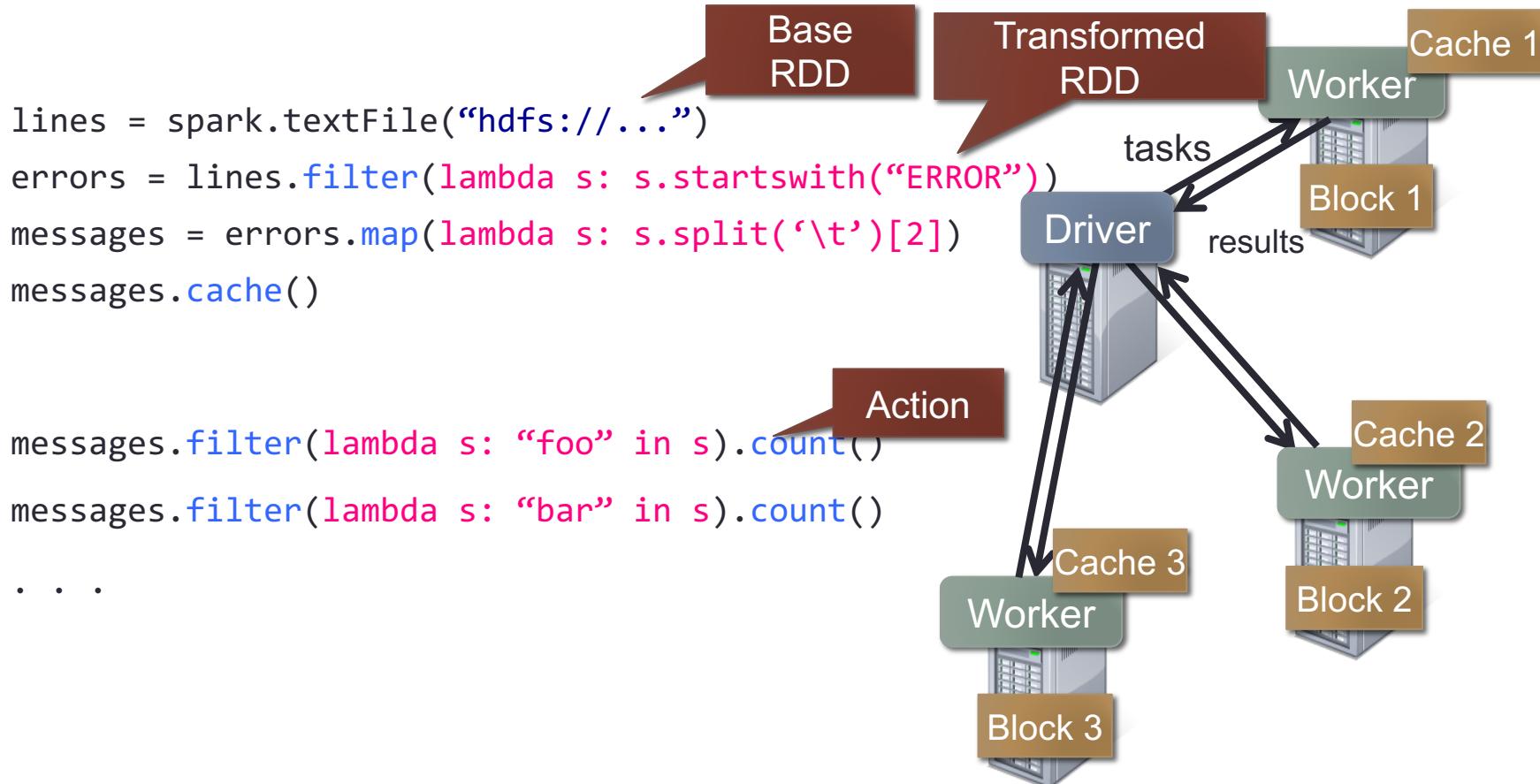
```
pythonLines = lines.filter(lambda line : "Python" in line)
```

```
pythonLines.first()
```

RDD-2

# Example: Mining Console Logs

- Load error messages from a log into memory, then interactively search for patterns



# Some Apache Spark tutorials

- <https://www.cloudera.com/documentation/enterprise/5-7-x/PDF/cloudera-spark.pdf>
- [https://stanford.edu/~rezab/sparkclass/slides/itas\\_workshop.pdf](https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf)
- <https://www.coursera.org/learn/big-data-essentials>
- <https://www.cloudera.com/documentation/enterprise/5-6-x/PDF/cloudera-spark.pdf>