AN INTRODUCTION TO SPARK AND TO ITS PROGRAMMING MODEL

epcc

Introduction to Apache Spark

- Fast, expressive cluster computing system
- Compatible with Apache Hadoop
- Faster and much easier than Hadoop MapReduce to use -- rich APIs
- Large community
- Goes far beyond batch applications to support a variety of workloads:
  - including interactive queries, streaming, machine learning, and graph processing

Figure: *Real Time Processing In Spark*

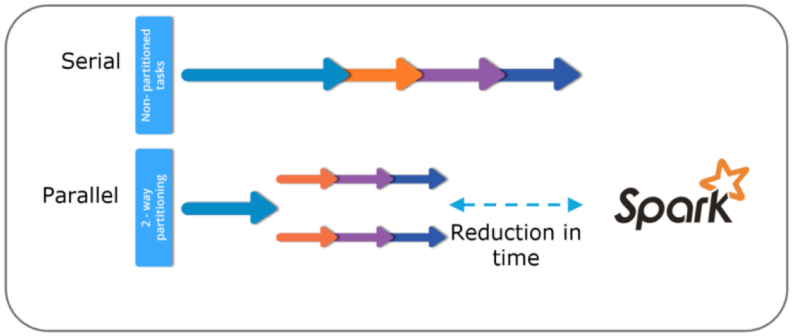
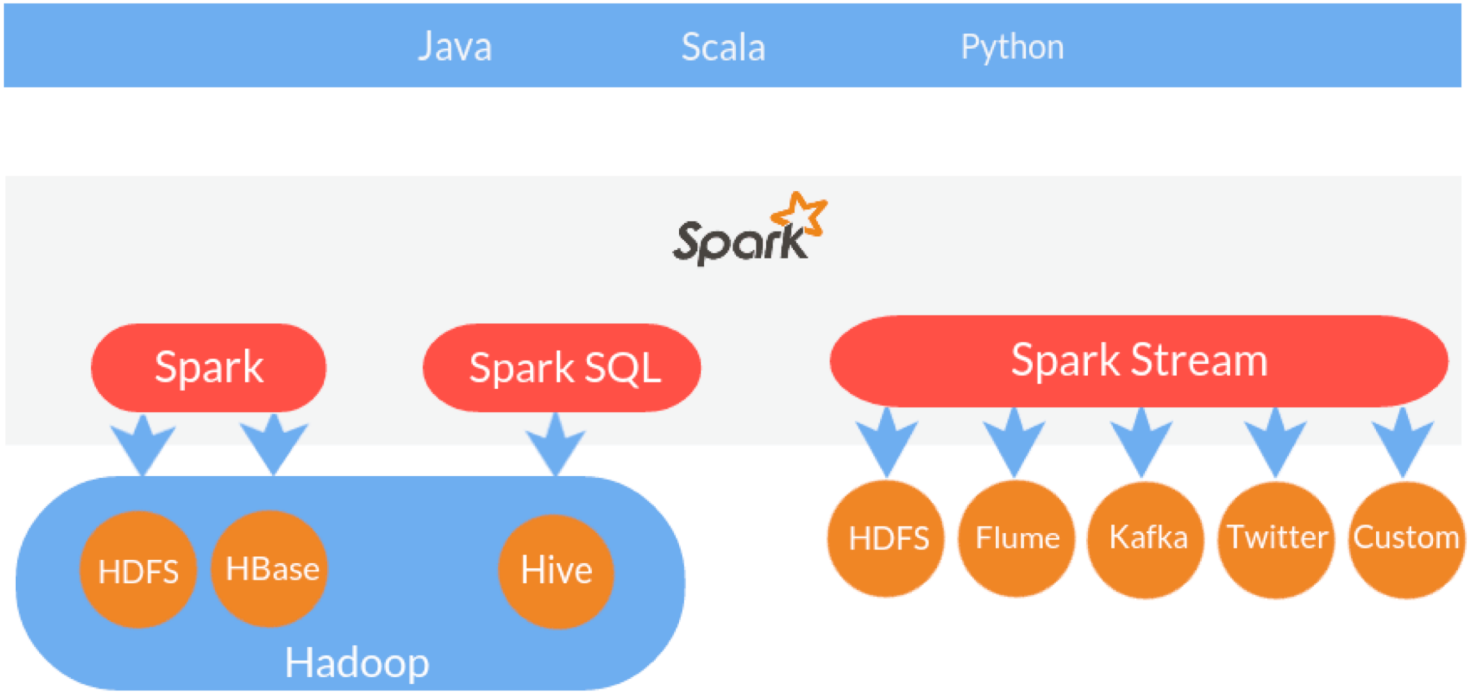


Figure: *Data Parallelism In Spark*

Introduction to Apache Spark

- General-purpose cluster in-memory computing system
- Provides high-level APIs in Java, Scala, python

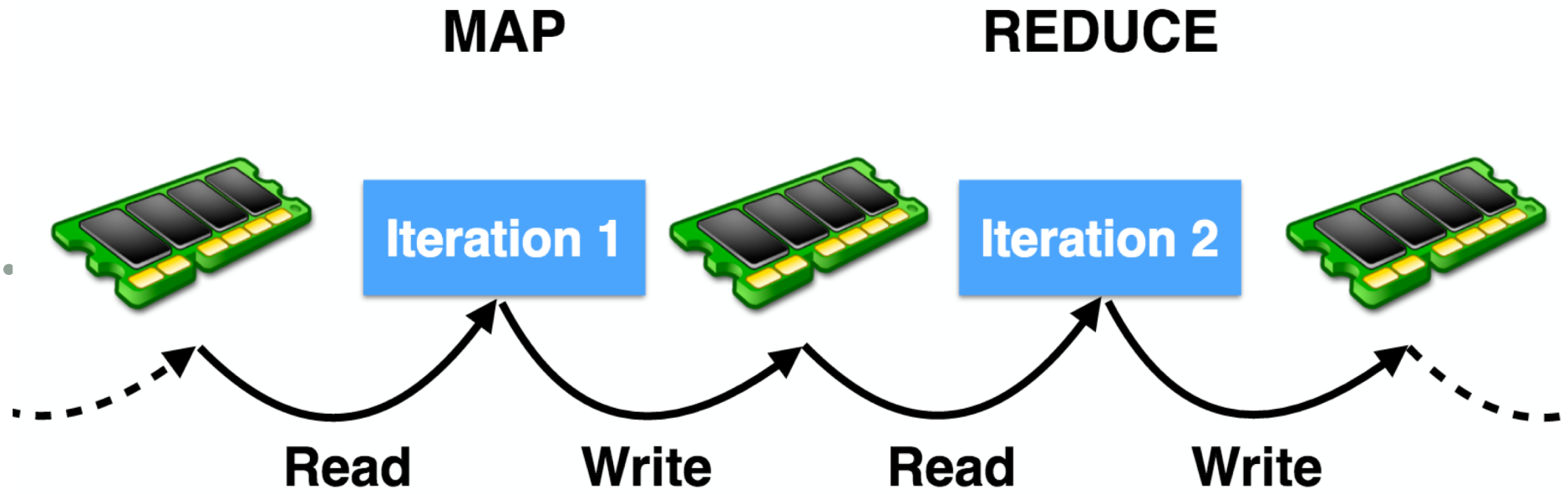


Introduction to Apache Spark

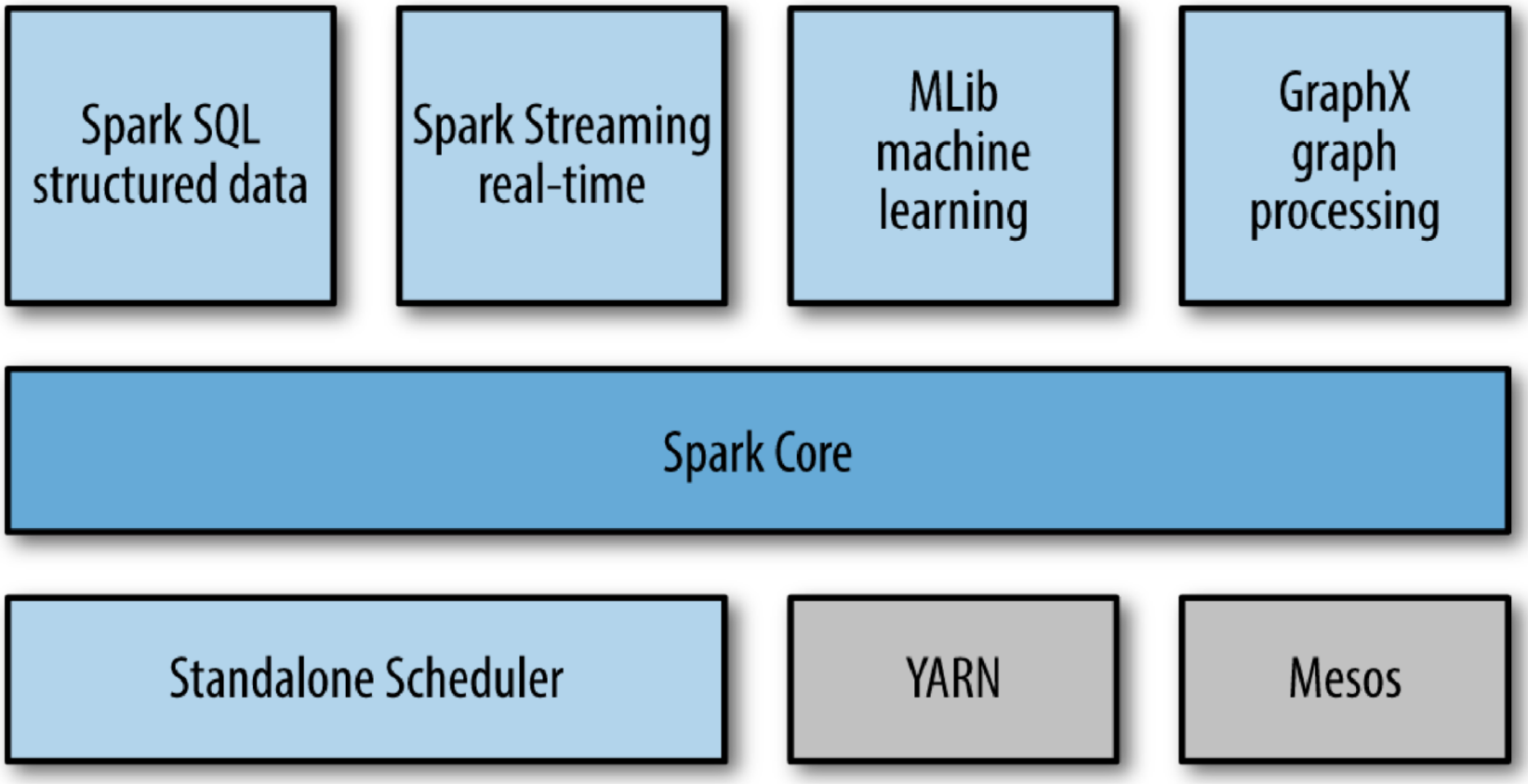
- Latest release - 2.4.0- https://spark.apache.org/docs/latest/index.html
- Quick start – https://spark.apache.org/docs/latest/quick-start.html
- Downloads - https://spark.apache.org/downloads.html

Introduction to Apache Spark

- Spark



Spark Ecosystem



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

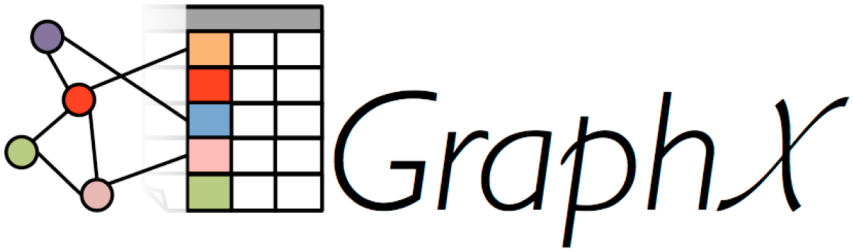
- Data analysis of streaming data
  - e.g. log files generated by production web servers
- Aimed at hight-throughput and fault-tolerant stream processing
- Dstream → Stream of datasets that contain data from certal interval

Spark MLlib

- MLlib is a library that contains common Machine Learning (ML) functionality:
  - Basic statistics
  - Classification (Naïve Bayes, decision tress, 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)

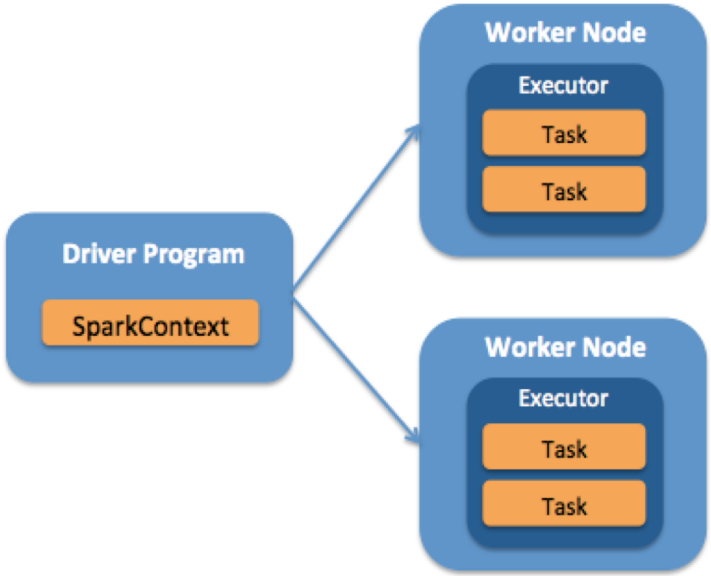
- We will use Python's interface to Spark called **pySpark**
- **A driver program** accesses the Spark environment through a **SparkContext** objet
- They **key concept** in Spark are datasets called **RDD**s (**R**esilient **D**istributed Dateset)
- Basic idea: We load our data into RDDs and perform 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.

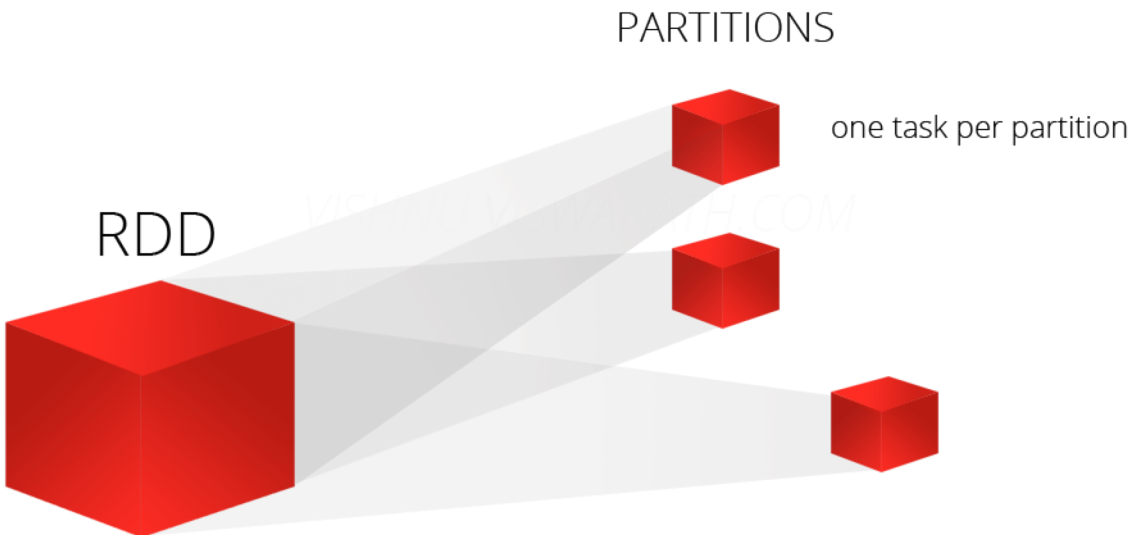
- In a *shell* the **SparkContext** is created for you (variable **sc).**

- **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.

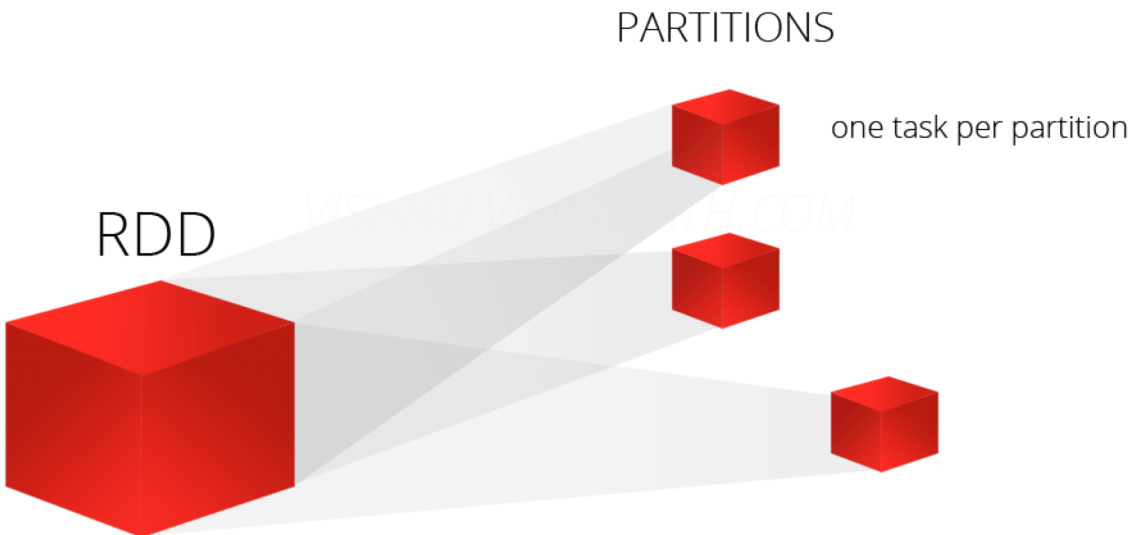
Resilient Distributed Datasets (**RDD**)

- **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



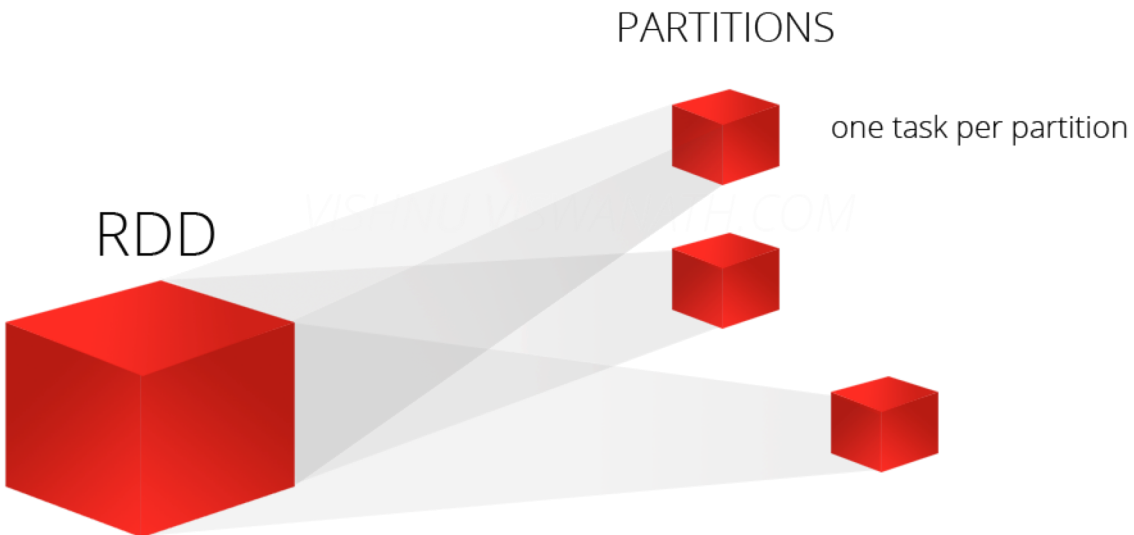
Resilient Distributed Datasets (**RDD**)

- 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

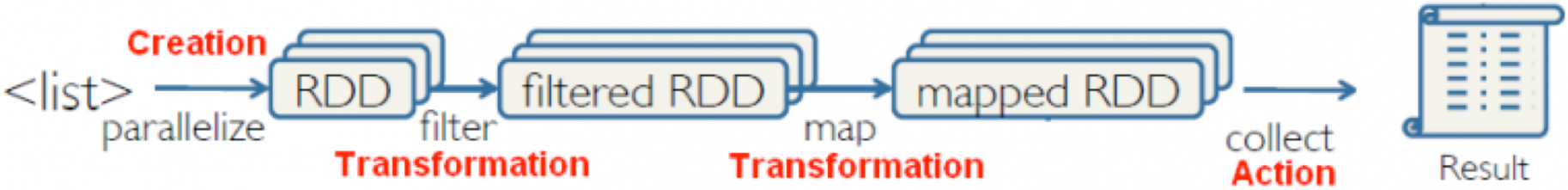
- Read from File

```
# 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

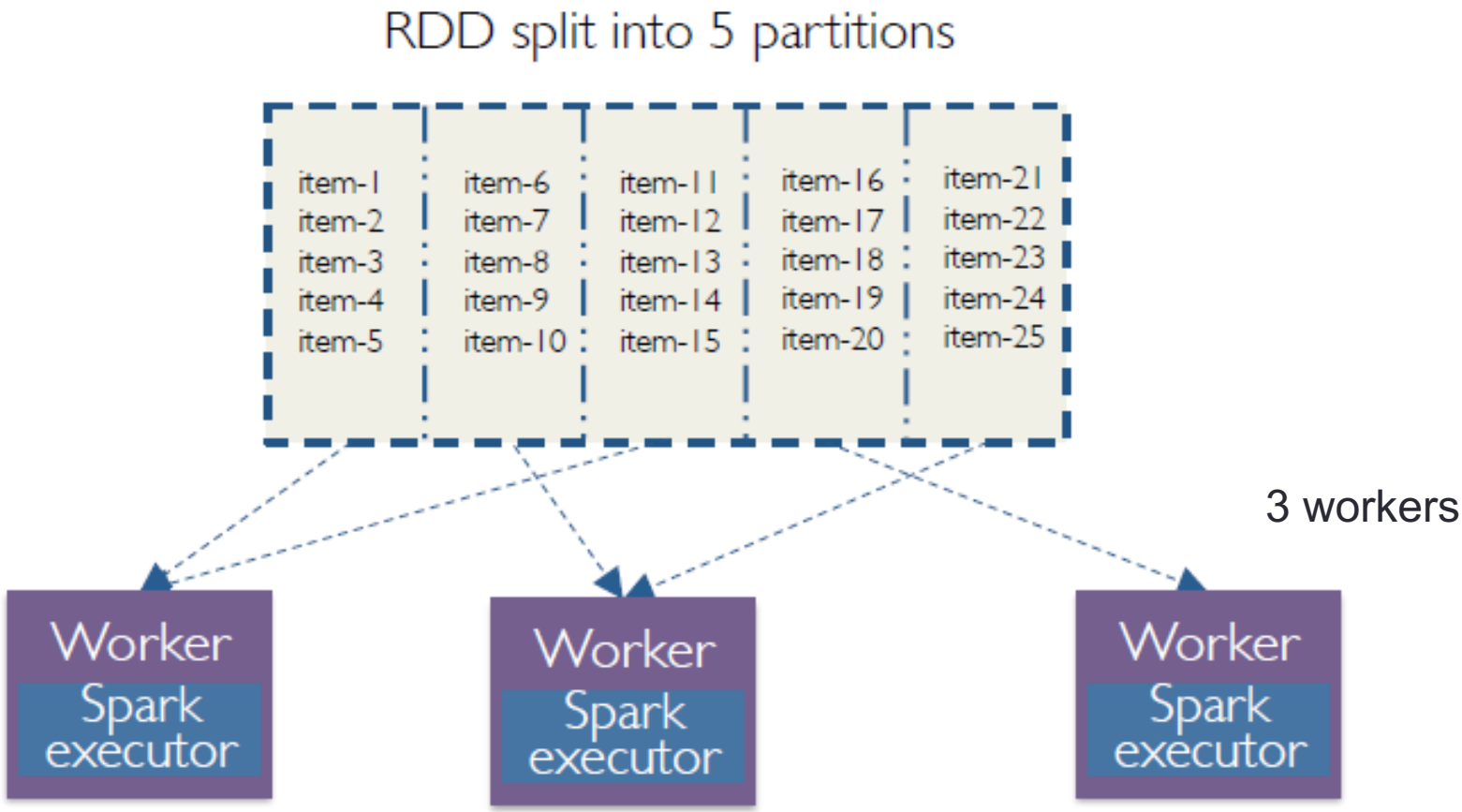
- All work is expressed as either:
  - creating new RDDs
  - transforming existing RDDs
  - calling operations on RDDs to compute a result.



- 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.

Partition


RDD split into 5 partitions
item-1
item-2
item-3
item-4
item-5
item-6
item-7
item-8
item-9
item-10
item-11
item-12
item-13
item-14
item-15
item-16
item-17
item-18
item-19
item-20
item-21
item-22
item-23
item-24
item-25
3 workers
Worker
Spark executor
Worker
Spark executor
Worker
Spark executor

Note about partitions

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

First Program!

```
sc = SparkContext(master="local[*]")
lines = sc.textFile("README.md", 4)
lines.count()
pythonLines = lines.filter(lambda line : "Python" in line)
pythonLines.first()
```

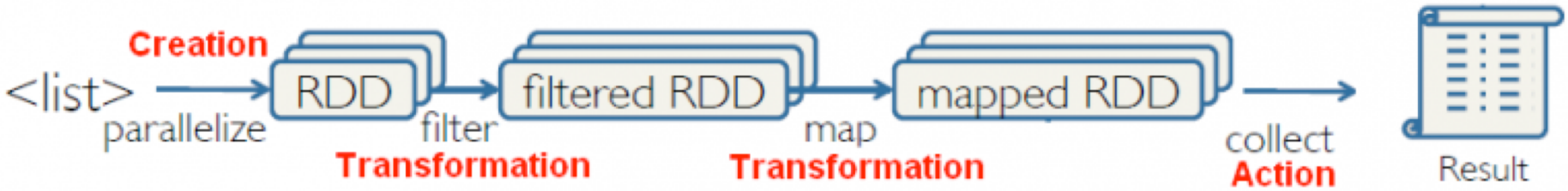
Context

RDD-1

RDD-2

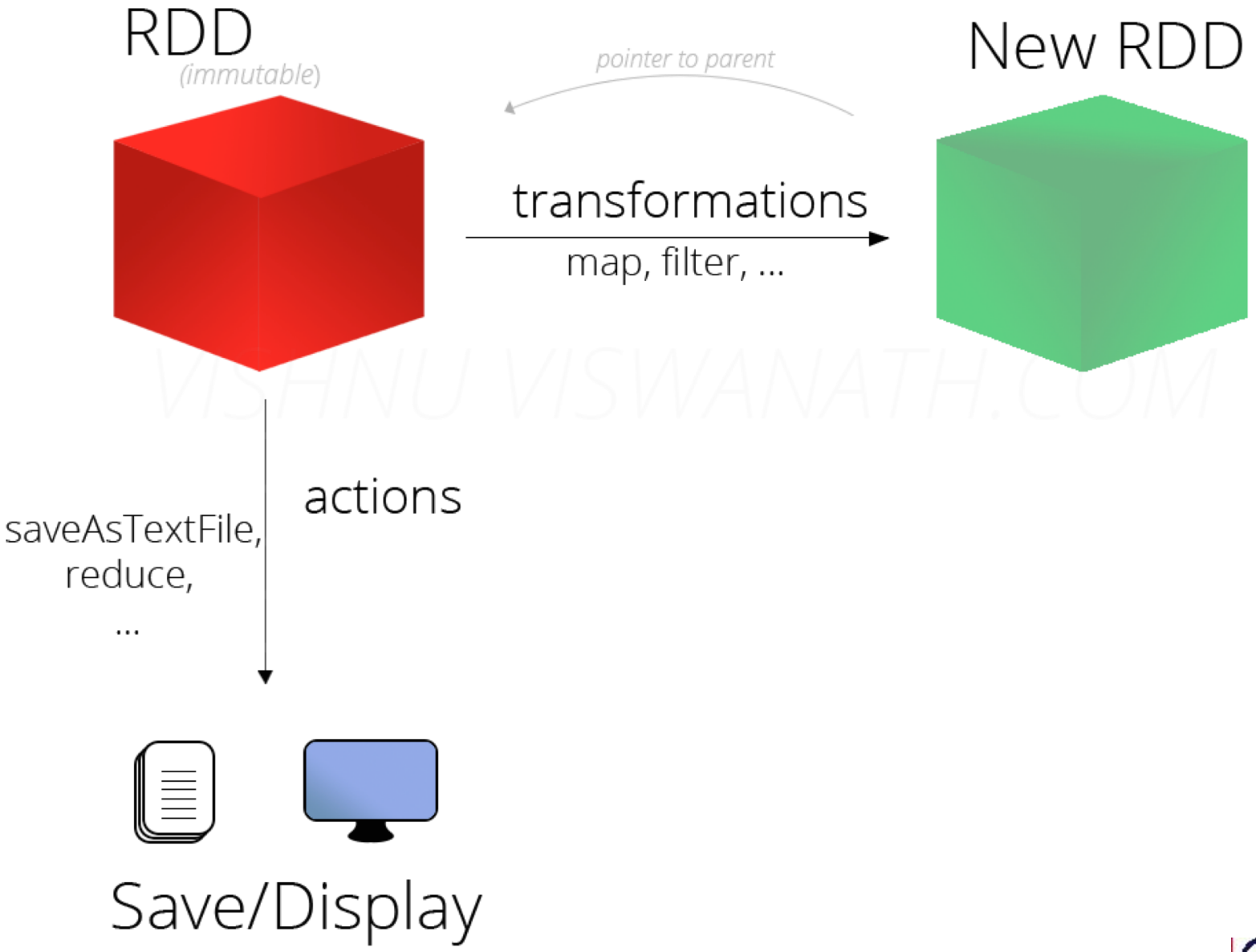
RDD operations

- Once created, RDDs offer two types of operations:
  - **transformations**
    - transformations include *map*, *filter*, *join*
    - lazy operation to build RDDs from other RDDs
  - **actions**
    - actions include *count*, *collect*, *save*
    - return a result or write it to storage



Transformation vs Actions

Transformations	Actions
map (func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey(func)	take(n)
mapValues(func)	saveAsTextFile(path)
sample(...)	countByKey()
union(other)	foreach(func)
distinct()	...
sortByKey()	
...	

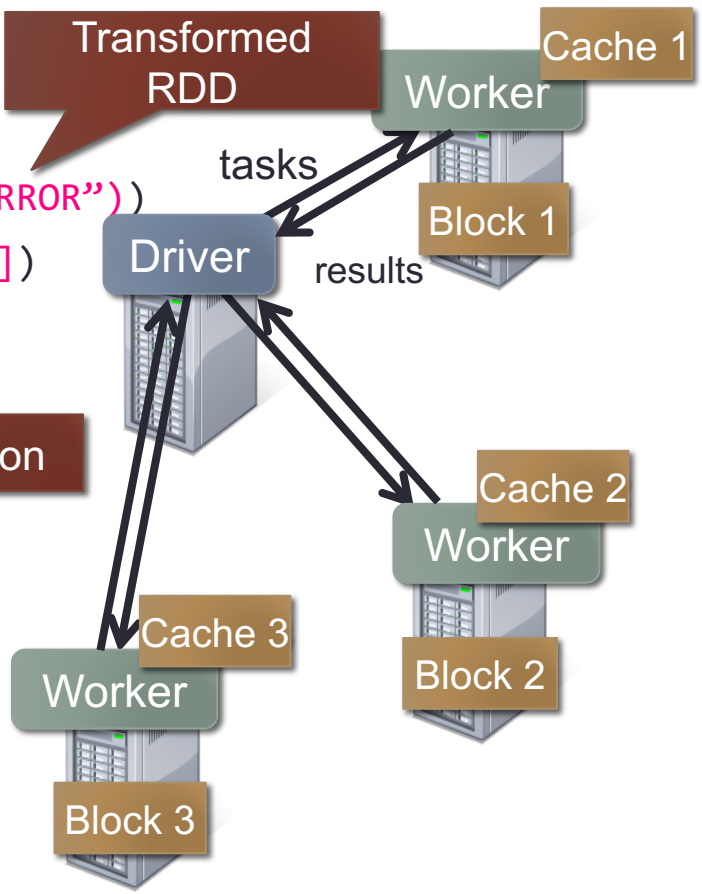


Example: Mining Console Logs

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()


messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
. . .
```



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://www.coursera.org/learn/big-data-essentials
- https://www.cloudera.com/documentation/enterprise/5-6-x/PDF/cloudera-spark.pdf