

# Parallel Programming using Resilient Distributed Datasets

---

# Objectives

---

- Define Resilient Distributed Datasets (RDDs)
- Define Parallel Programming
- Explain Resilience in Apache Spark
- Relate RDD and Parallel Programming with Apache Spark

**PySpark**

**Spark With Python**

# Overview of PySpark

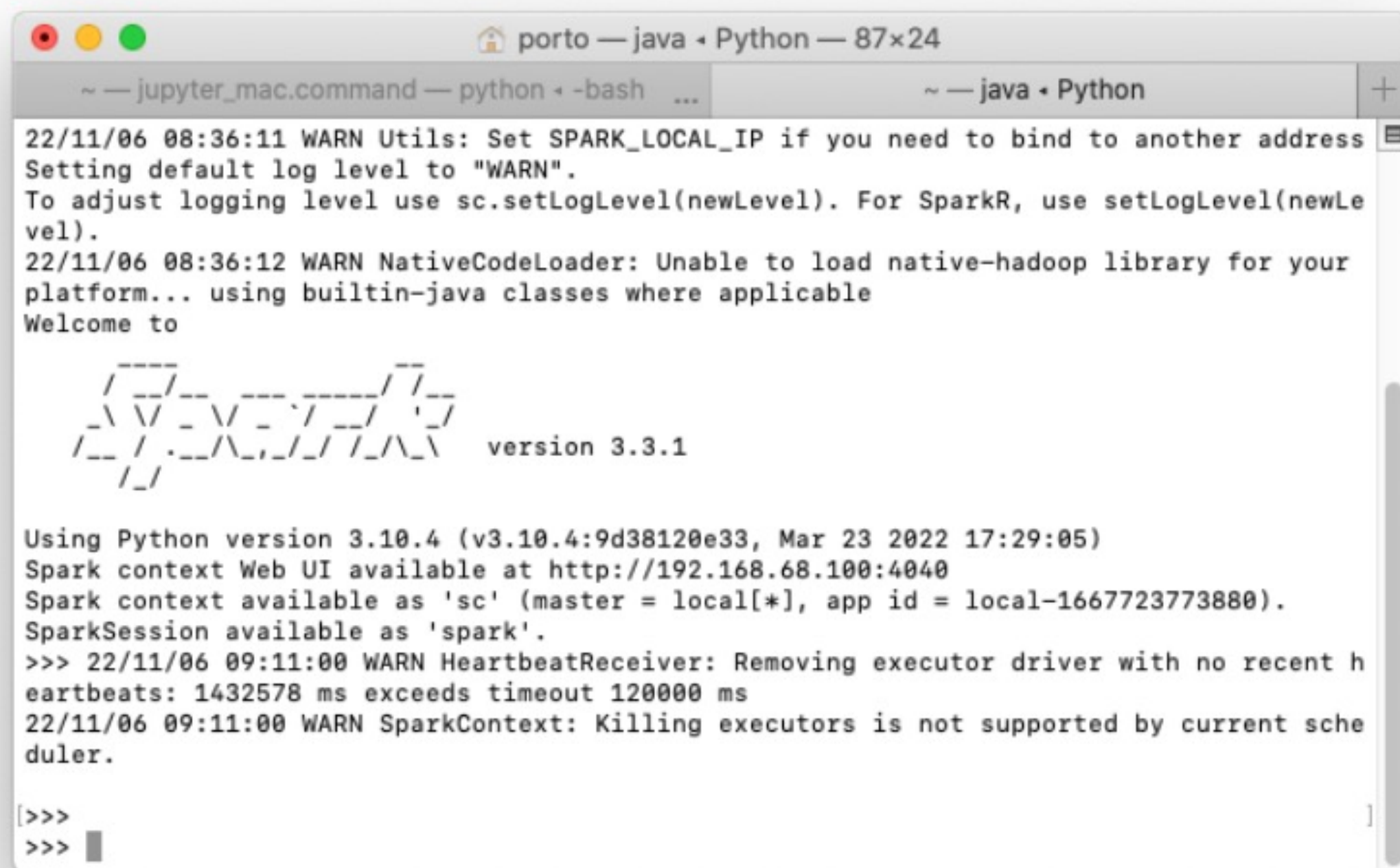
- Apache Spark is written in Scala
- To support Python with Spark, Apache Spark Community released PySpark
- Similar computation speed and power as Scala
- PySpark APIs are similar to Pandas and Scikit-learn

# PySpark

Apache Spark is written in Scala programming language. To support Python with Spark, Apache Spark Community released a tool, PySpark. Using PySpark, you can work with RDDs in Python programming language also. It is because of a library called Py4j that they are able to achieve this. PySpark offers PySpark Shell which links the Python API to the spark core and initializes the Spark context. Majority of data scientists and analytics experts today use Python because of its rich library set. Integrating Python with Spark is a boon to them.

# PySpark shell

PySpark shell is referred as REPL (Read Eval Print Loop) which is used to quickly test PySpark statements. Spark shell is available for Scala, Python and R. The pyspark command is used to launch Spark with Python shell also call PySpark.



```
22/11/06 08:36:11 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/11/06 08:36:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to

      /--
     /  \
    /    \
   /      \
  /        \
 /          \
/            \
 \          /
  \        /
   \      /
    \    /
     \  /
      --

version 3.3.1

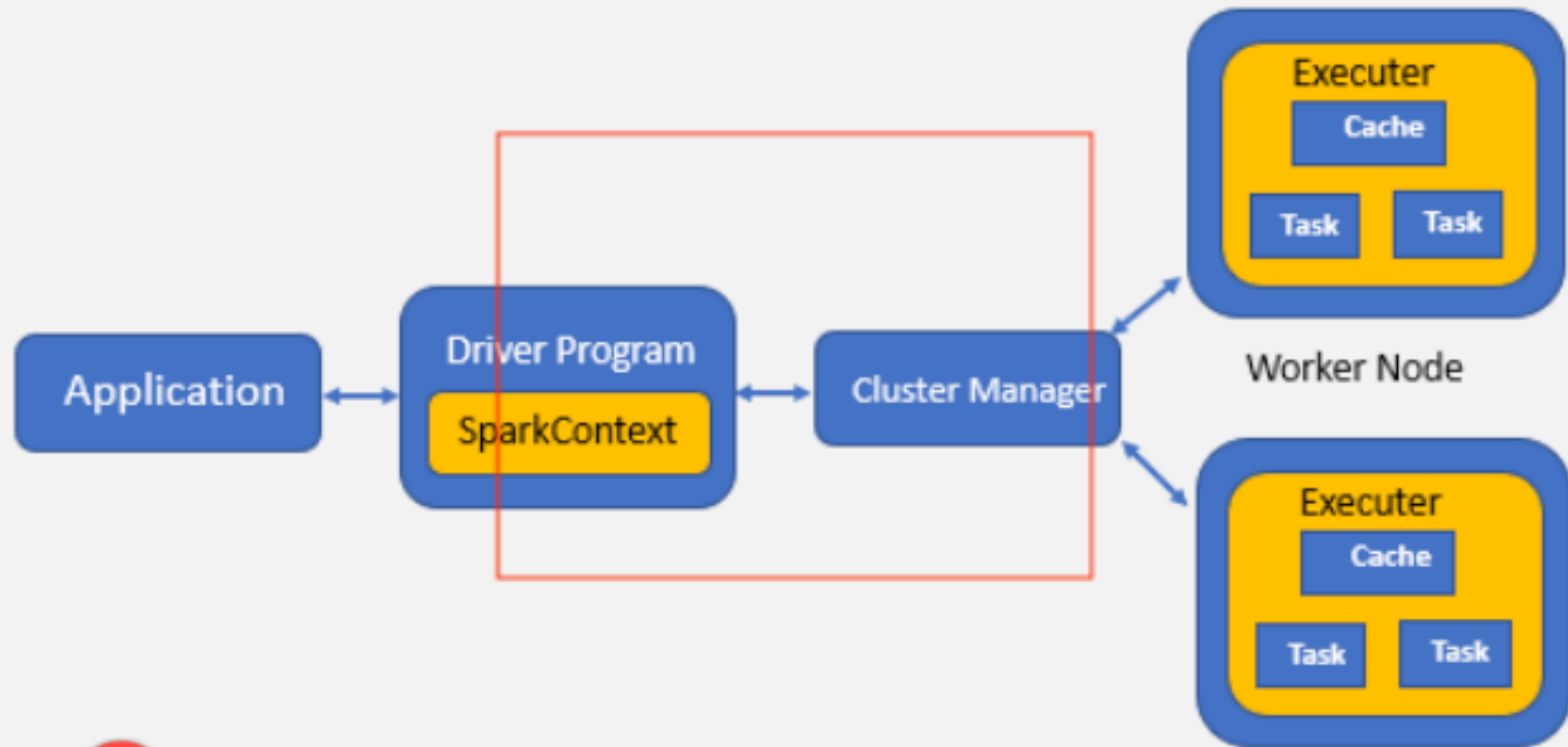
Using Python version 3.10.4 (v3.10.4:9d38120e33, Mar 23 2022 17:29:05)
Spark context Web UI available at http://192.168.68.100:4040
Spark context available as 'sc' (master = local[*], app id = local-1667723773880).
SparkSession available as 'spark'.
>>> 22/11/06 09:11:00 WARN HeartbeatReceiver: Removing executor driver with no recent h
eartbeats: 1432578 ms exceeds timeout 120000 ms
22/11/06 09:11:00 WARN SparkContext: Killing executors is not supported by current sche
duler.

[>>>
>>>
```

## Using Spark in Python - Spark Context

The first step in using Spark is connecting to a Spark cluster. A `SparkContext` represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster. `SparkContext` is the entry point to any spark functionality. When we run any Spark application, a driver program starts, which has the main function and your `SparkContext` gets initiated here. The driver program then runs the operations inside the executors on worker nodes. In practice, the cluster will be hosted on a remote machine that's connected to all other nodes. There will be one computer, called the master that manages splitting up the data and the computations. The master is connected to the rest of the computers in the cluster, which are called worker. The master sends the workers data and calculations to run, and they send their results back to the master.

# SparkContext in Apache Spark





## Method 2: Automatic Installation – The Easy Way

The second method of installing PySpark on Google Colab is to use pip install.

```
# Install pyspark  
!pip install pyspark
```



```
from pyspark import SparkConf
from pyspark.context import SparkContext

spark = SparkContext.getOrCreate(SparkConf() \
    .setMaster("local[*]") \
    .setAppName("Intro pyspark"))

print (spark.version)

print(spark.pythonVer)

print(spark.master)
```

3.3.1

3.8

local[\*]

# Inspecting SparkContext

- Version: To retrieve SparkContext version

```
spark.version
```

```
3.3.1
```

- Python Version: To retrieve Python version of SparkContext

```
spark.pythonVer
```

```
3.8
```

- Master: URL of the cluster or “local” string to run in local mode of SparkContext

```
spark.master
```

```
local[*]
```

# Resilient Distributed Datasets

# What are RDDs

---

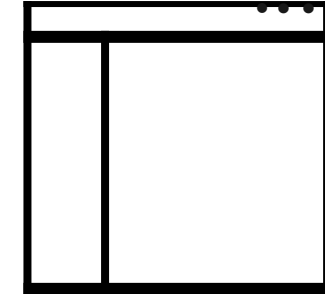
A resilient distributed dataset is:

- Spark's primary data abstraction
- A fault-tolerant collection of elements
- Partitioned across the nodes of the cluster
- Capable of accepting parallel operations

# Spark applications

---

Consist of a driver program that runs  
The user's main functions  
And multiple *parallel operations*  
on a cluster.



# RDD supported files

---

## Supported File types

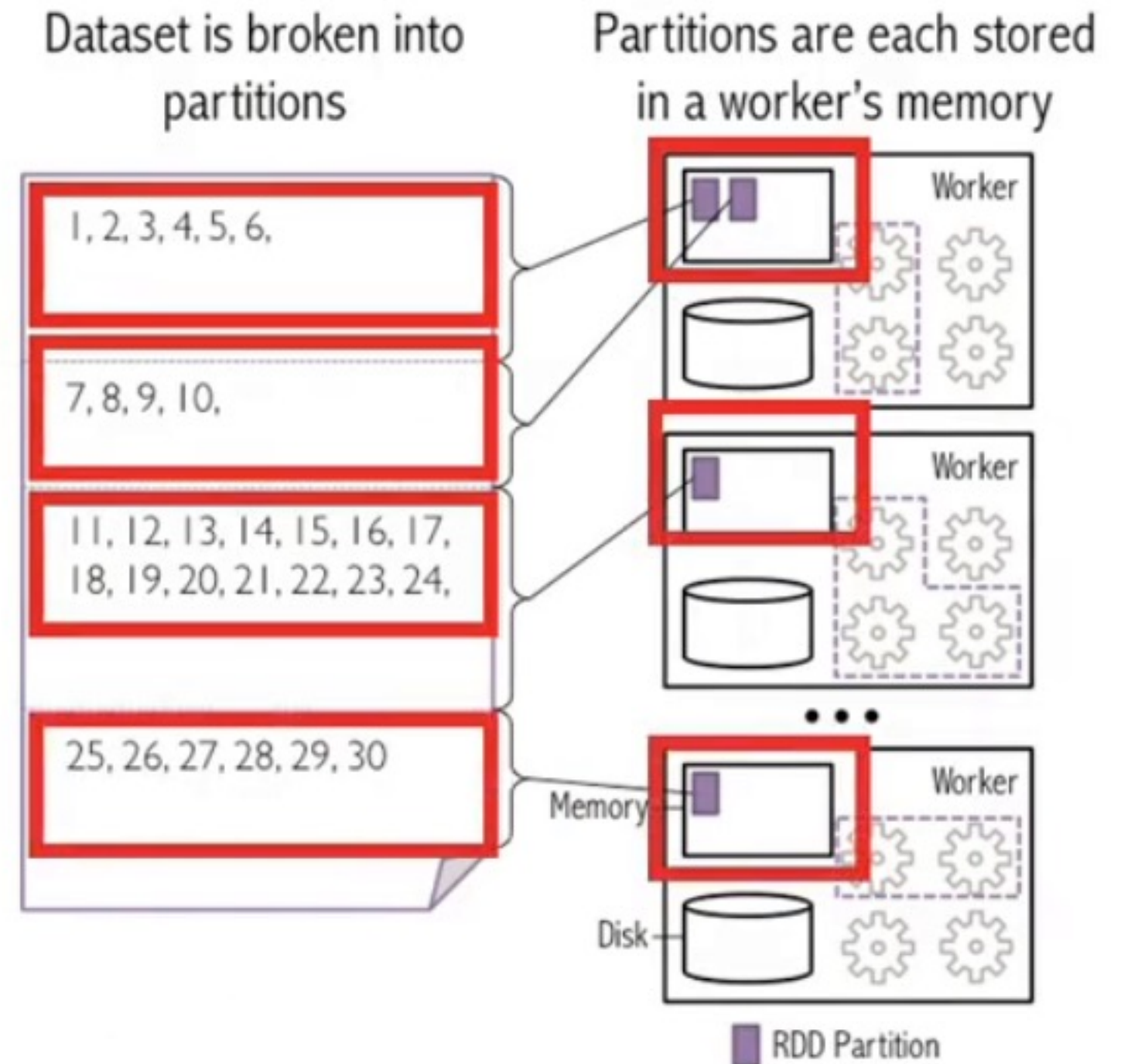
- Text
- SequenceFiles
- Avro
- Parquet
- Hadoop input formats

## Supported File formats

- Local
- Cassandra
- HBase
- HDFS
- Amazon S3
- and others
- SQL and NoSQL

# Creating an RDD in Spark

Use an external or local file from Hadoop-supported file system such as:





# Create an RDD from a collection

---

Simple examples of creating a RDD from a list in Scala and Python

```
// Scala example  
val data= Array(1, 2, 3, 4,  
5)  
val distData =  
sc.parallelize(data)
```

```
// Python example  
data= [1, 2, 3, 4, 5]  
distData =  
sc.parallelize(data)
```

# Create an RDD from a collection

---

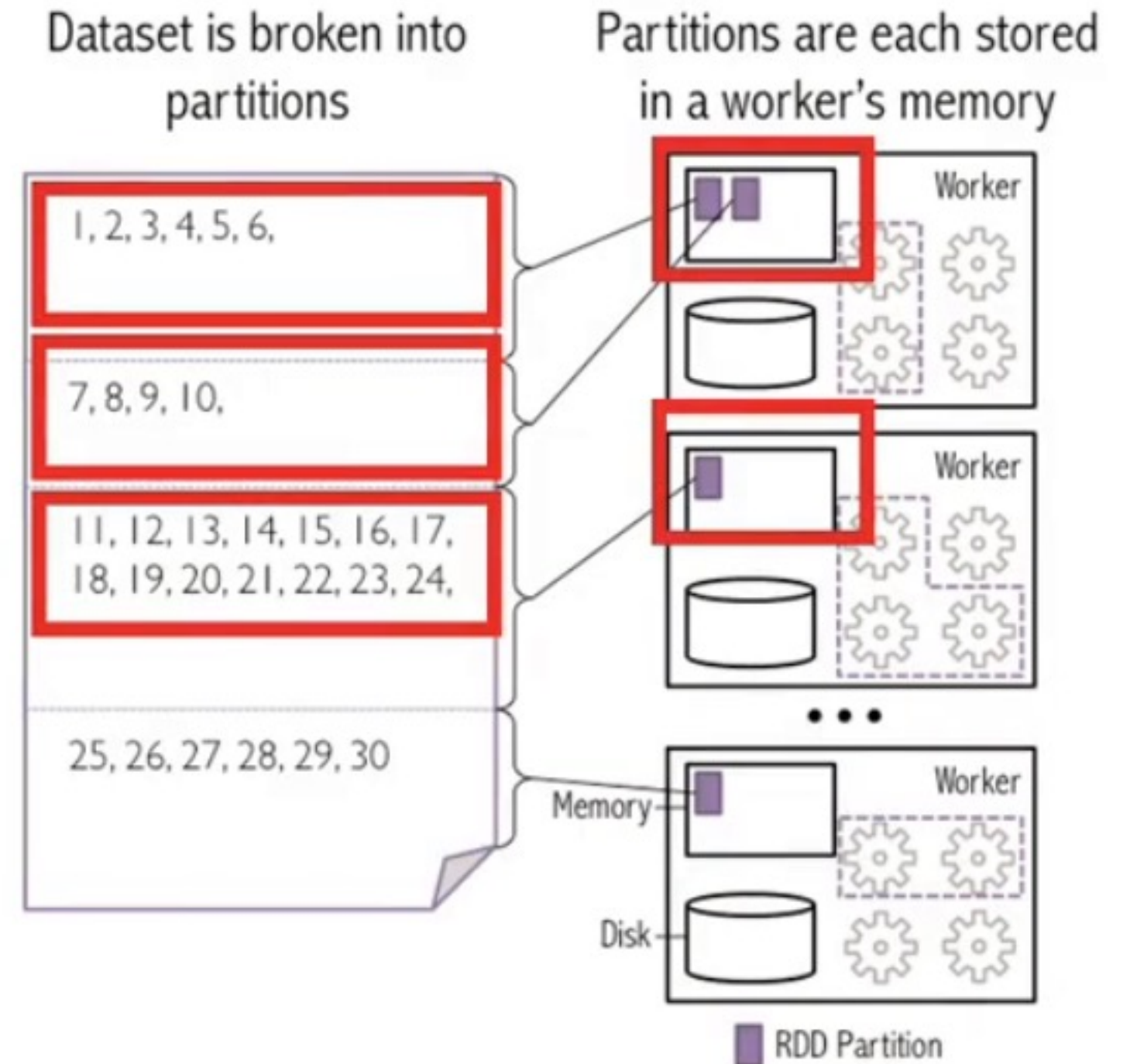
Simple examples of creating a RDD from a list in Scala and Python

```
// Scala example  
val data = Array(1, 2, 3, 4, 5)  
val distData =  
sc.parallelize(data)
```

```
// Python example  
data = [1, 2, 3, 4, 5]  
distData =  
sc.parallelize(data)
```

# Creating an RDD in Spark

Apply a transformation on an existing RDD to create a new RDD



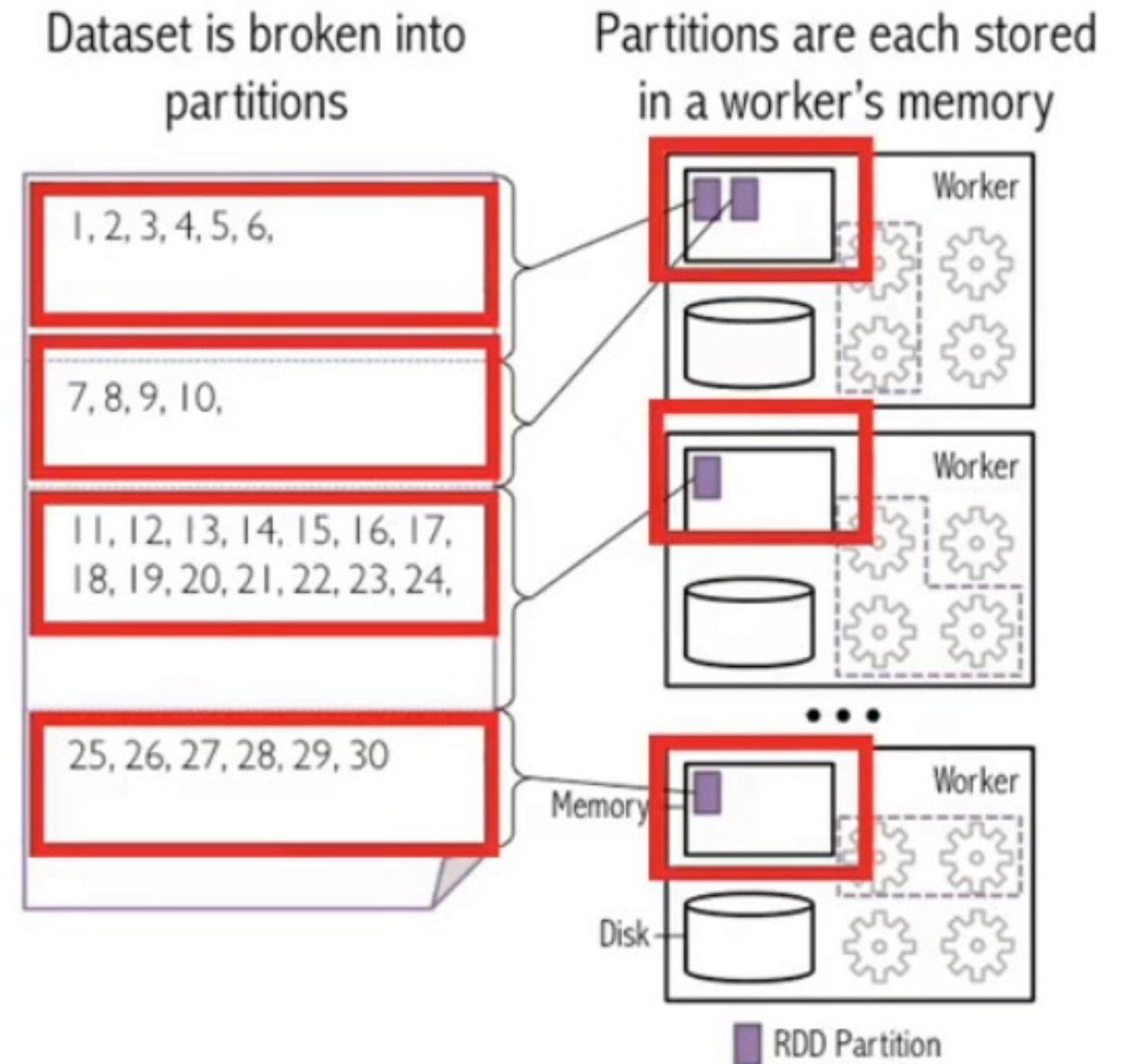
# What is Parallel Programming

---

- Parallel programming:
- Is the simultaneous use of multiple compute resources to solve a computational problem
- Breaks problems into discrete parts that can be solved concurrently
- Runs simultaneous instructions on multiple processors

# RDDs & Parallel Programming

- You can create an RDD by parallelizing an array of objects, or by splitting a dataset into partitions
- Spark runs one task for each partition of the cluster



Once created, the distributed dataset (distData) can be operated on in parallel. For example, we can call `distData.reduce(lambda a, b: a + b)` to add up the elements of the list.

In [4]:

```
distData = sc.parallelize([1,2,3])  
distData.reduce(lambda a, b: a + b)
```

Out [4]:

6

# Summary

---

- You can create an RDD using an external or local file from Hadoop-supported file, from a collection or from another RDD
- RDDs are immutable and always recoverable
- RDDs can persist or cache datasets in memory across operations, which speeds iterative operations