

# Data Analysis with



## 1. Introduction & RDDs

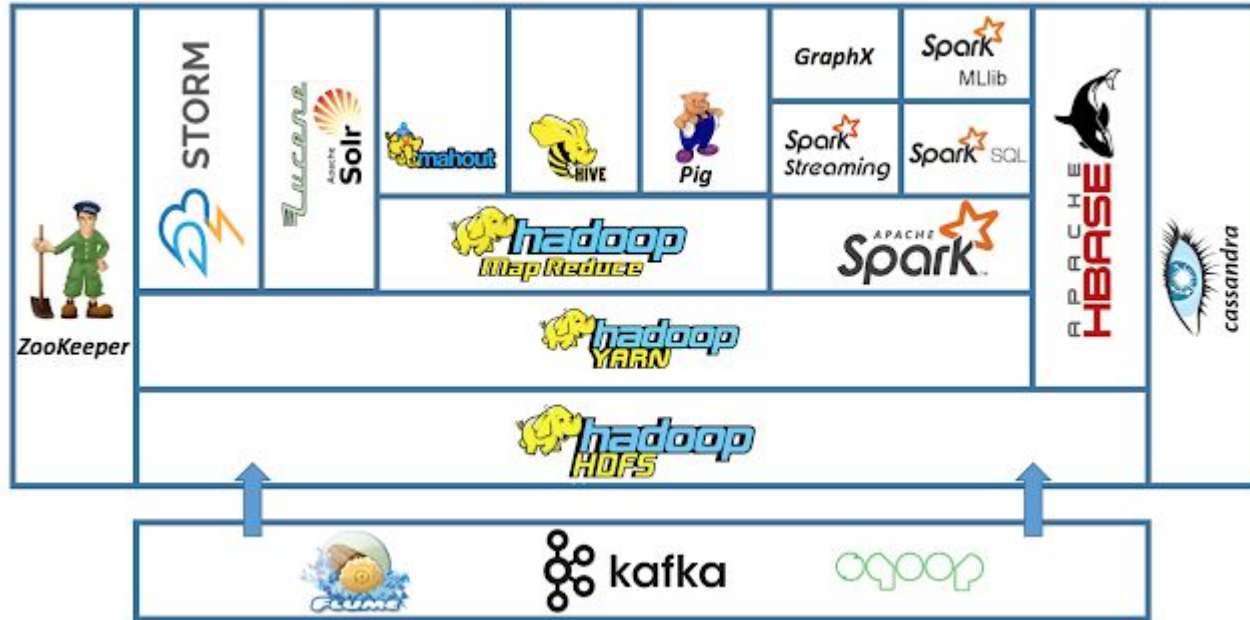
# What is Apache Spark?

- Fast (in-memory), distributed (parallel), general-purpose **cluster computing system** - [spark.apache.org](https://spark.apache.org)
- **Open Source** project ([Apache Software Foundation](https://www.apache.org/))
- Strongly tied to the **Hadoop** ecosystem

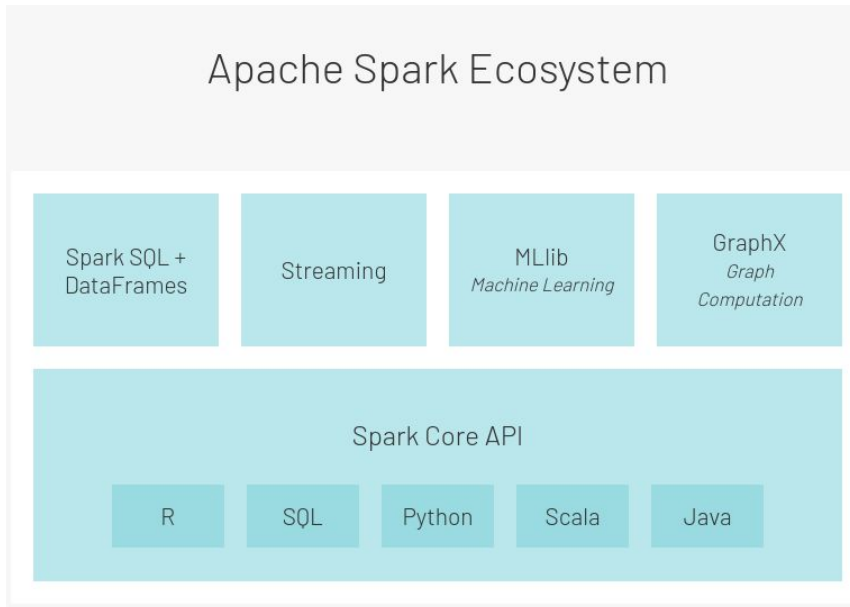
# What is Apache Spark?

- Written in **Scala** → runs in the JVM (Java Virtual Machine)
- Pick your language: **Scala, Python, R, SQL, Java** (not in the notebook environment)
- Sparks transforms your code into **tasks** to run on the **cluster nodes**

# Hadoop Ecosystem



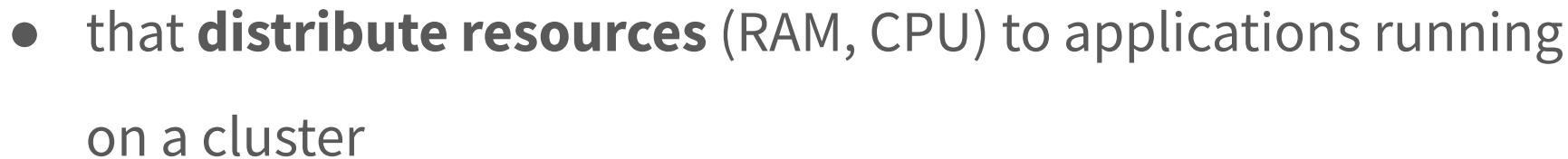
# Spark Ecosystem



# Use cases

- Analyze / transform / apply ML models on:
  - Very **large datasets** (Extract, Transform and Load)
  - **Streaming** data (in near-real-time)
  - **Graphs** (network analysis)
- of structured (tables), semi-structured (JSON) or unstructured (text) data

- Hadoop **YARN**
- Apache Mesos
- Kubernetes
- Spark standalone



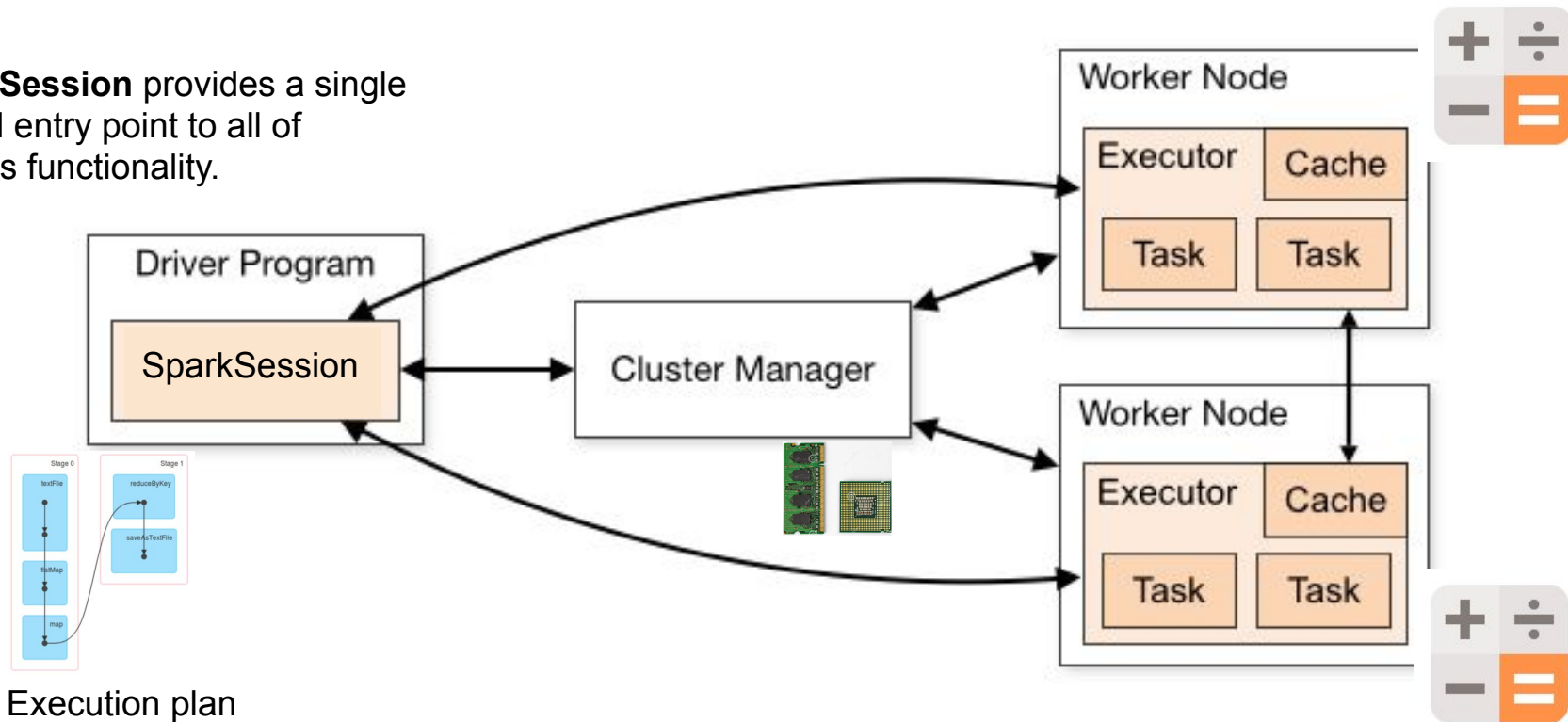
# Spark Internals

- You: Write the **code** and **submit** it
- Spark:
  1. Asks for resources to **create driver + executors**
  2. Transforms the **code** into **tasks**
  3. **Driver** sends **tasks** to **executors**
  4. **Executors** sends **results** to **driver**




# Spark Internals

**SparkSession** provides a single unified entry point to all of Spark's functionality.



# YARN (cluster manager)

← → ↻ 149.165.158.138:8088/cluster ★ Logg



Cluster

[About](#)  
[Nodes](#)  
[Node Labels](#)  
[Applications](#)  

NEW  
NEW SAVING  
SUBMITTED  
ACCEPTED  
RUNNING  
FINISHED  
FAILED  
KILLED

[Scheduler](#)

Tools

## All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
7	0	0	7	0	0 B	32 GB	0 B	0	32	0	4	0	0	0

Scheduler Metrics

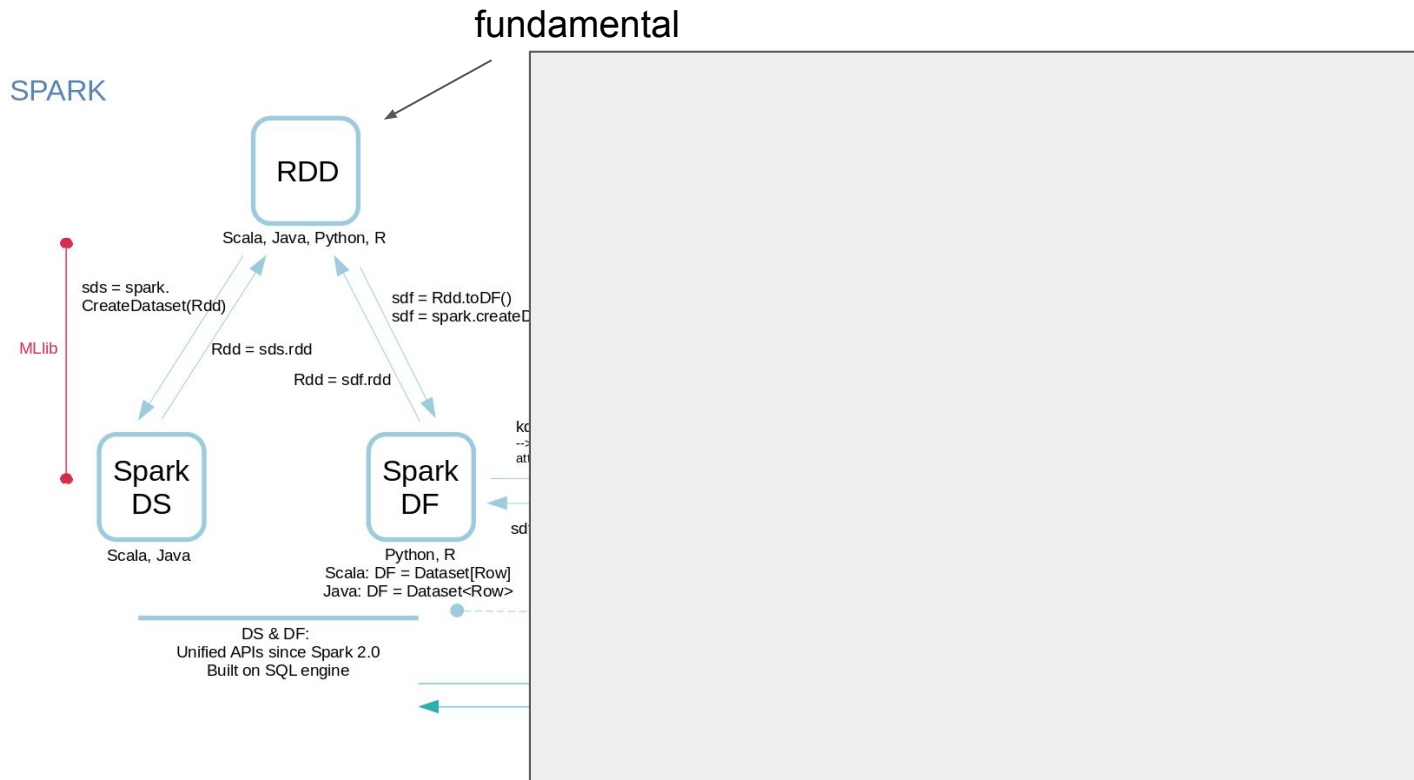
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries Search:

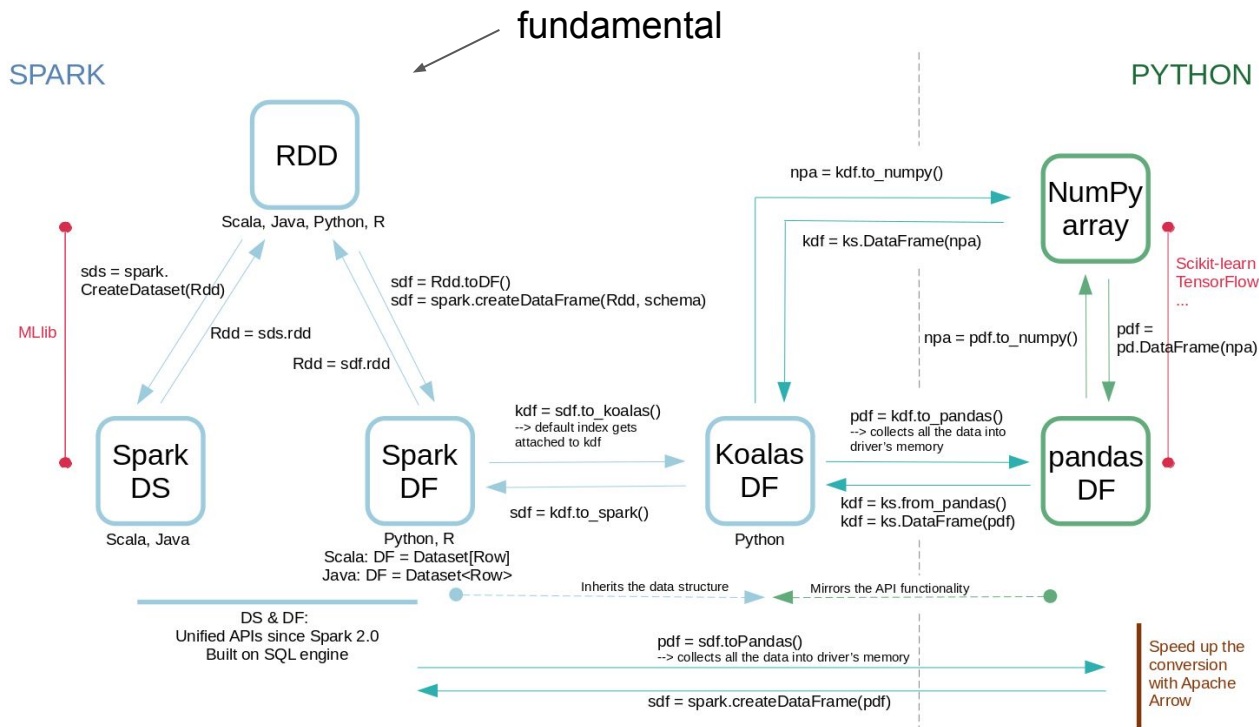
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress
application_1430437177775_0007	hadoop	QuasiMonteCarlo	MAPREDUCE	default	Thu Apr 30 21:39:01 -0400 2015	Thu Apr 30 21:39:56 -0400 2015	FINISHED	SUCCEEDED	<div></div>
application_1430437177775_0006	hadoop	grep-sort	MAPREDUCE	default	Thu Apr 30 20:37:57 -0400 2015	Thu Apr 30 20:38:12 -0400 2015	FINISHED	SUCCEEDED	<div></div>
application_1430437177775_0005	hadoop	grep-search	MAPREDUCE	default	Thu Apr 30 20:36:45 -0400 2015	Thu Apr 30 20:37:55 -0400 2015	FINISHED	FAILED	<div></div>
application_1430437177775_0004	hadoop	QuasiMonteCarlo	MAPREDUCE	default	Thu Apr 30 20:34:57 -0400 2015	Thu Apr 30 20:35:54 -0400 2015	FINISHED	SUCCEEDED	<div></div>
application_1430437177775_0003	hadoop	QuasiMonteCarlo	MAPREDUCE	default	Thu Apr 30 19:57:25 -0400 2015	Thu Apr 30 19:58:23 -0400 2015	FINISHED	SUCCEEDED	<div></div>
application_1430437177775_0002	hadoop	grep-sort	MAPREDUCE	default	Thu Apr 30 19:43:20 -0400 2015	Thu Apr 30 19:43:37 -0400 2015	FINISHED	SUCCEEDED	<div></div>
application_1430437177775_0001	hadoop	grep-search	MAPREDUCE	default	Thu Apr 30 19:41:58 -0400 2015	Thu Apr 30 19:43:18 -0400 2015	FINISHED	FAILED	<div></div>

Showing 1 to 7 of 7 entries First Previous 1

# Spark Data Structures



# Spark Data Structures



# Spark Operations

2 types of **operations**:

- **Transformations:** transform a Spark DataFrame/RDD into a new DataFrame/RDD without altering the original data
- **Actions:** get the result

Transformations	Actions
orderBy()	show()
groupBy()	take()
filter()	count()
select()	collect()
join()	save()

**Lazy evaluation:** transformations triggered when action is called.

# RDDs: Resilient Distributed Datasets

- A **fault-tolerant collection** of elements partitioned **across the nodes** of the cluster (parallelism)
- An element can be: string, array, dictionary, etc.
- An RDD is **immutable**
- Transformations: lambda expressions on **key-value pairs**
- An RDD can be **persisted** in memory for reuse (avoid recomputing)

# RDDs: Resilient Distributed Datasets

- Mostly load data from **HDFS** (or Hadoop-like file system)
- RDDs are **partitioned**:
  - **1 partition** = **1 block** = 128 MB in HDFS
  - **1 task** runs on **1 partition**
  - Default = 1 partition per CPU core

# Spark RDD API

Chain transformations and use the result with an action :

```
rdd1 = sc.wholeTextFiles('hdfs://text/file/path') \
    .map(lambda x: x.split(',')) \      ← transformation
    .flatMap(...) \                    ← transformation
    .groupByKey(...)                   ← transformation

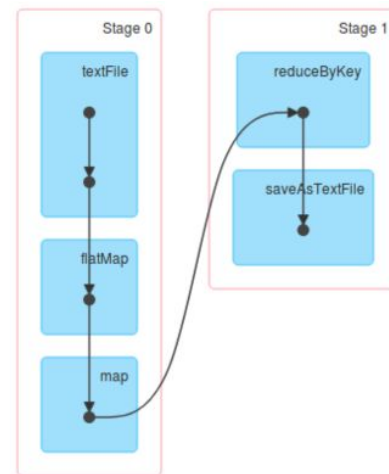
rdd1.take(10)      ← action
```



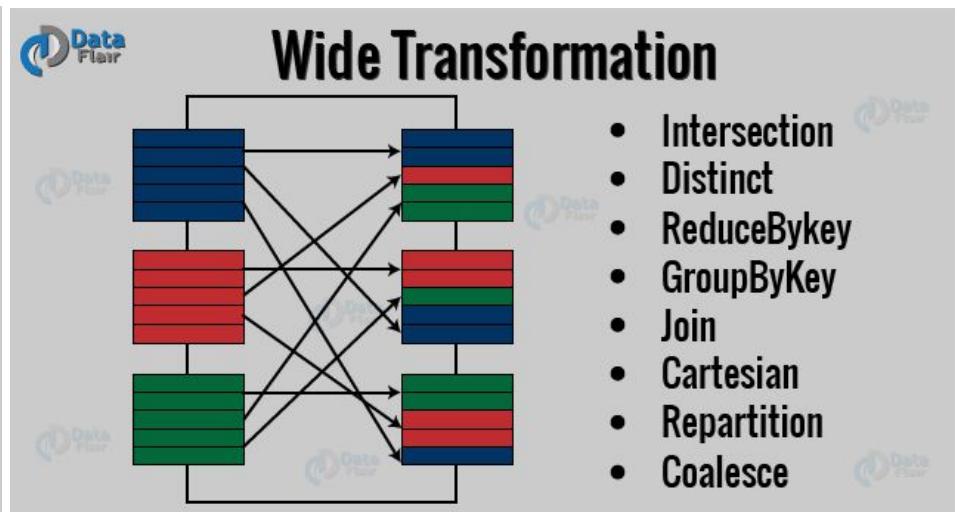
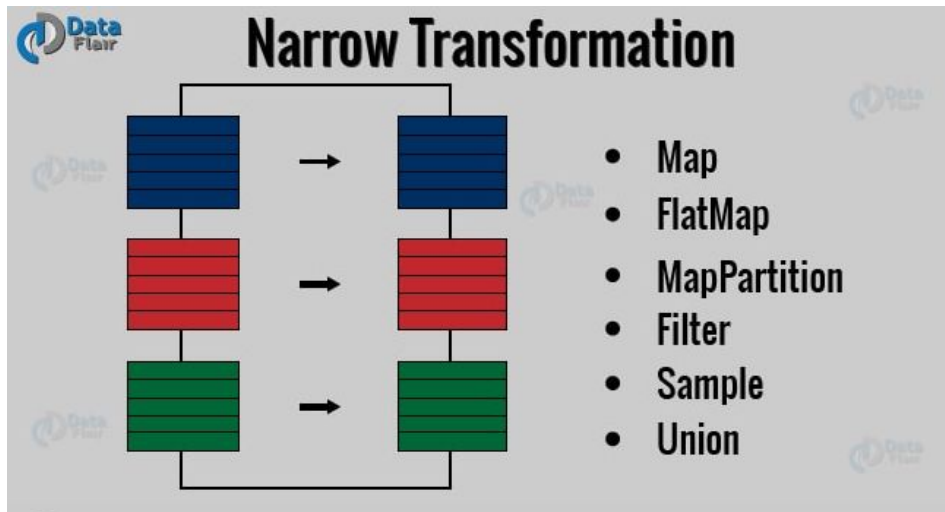
# Spark RDD API

When an **action** is run:

- Spark builds a **Directed Acyclic Graph** (DAG) of **stages**
- **1 stage** = **X tasks** (1 by RDD partition)
- Tasks are sent to **executors**
- The end of one stage is conditioned by a **shuffle**



# Narrow and Wide Transformations



# Spark + RDD: full recap

1. Spark **creates driver + executors**
2. Spark transforms your **code** into **stages** (DAG)
3. Each **executor** gets **partitions** of the RDD
4. For each **stage**, the **driver** sends a **task** to each **executor** to run on each **partition**
5. Some transformations require **data shuffling**, some don't