Introduction to Spark

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**framework of spark**', Resilient Distributed Datasets (RDDs) and also discuss Spark execution.

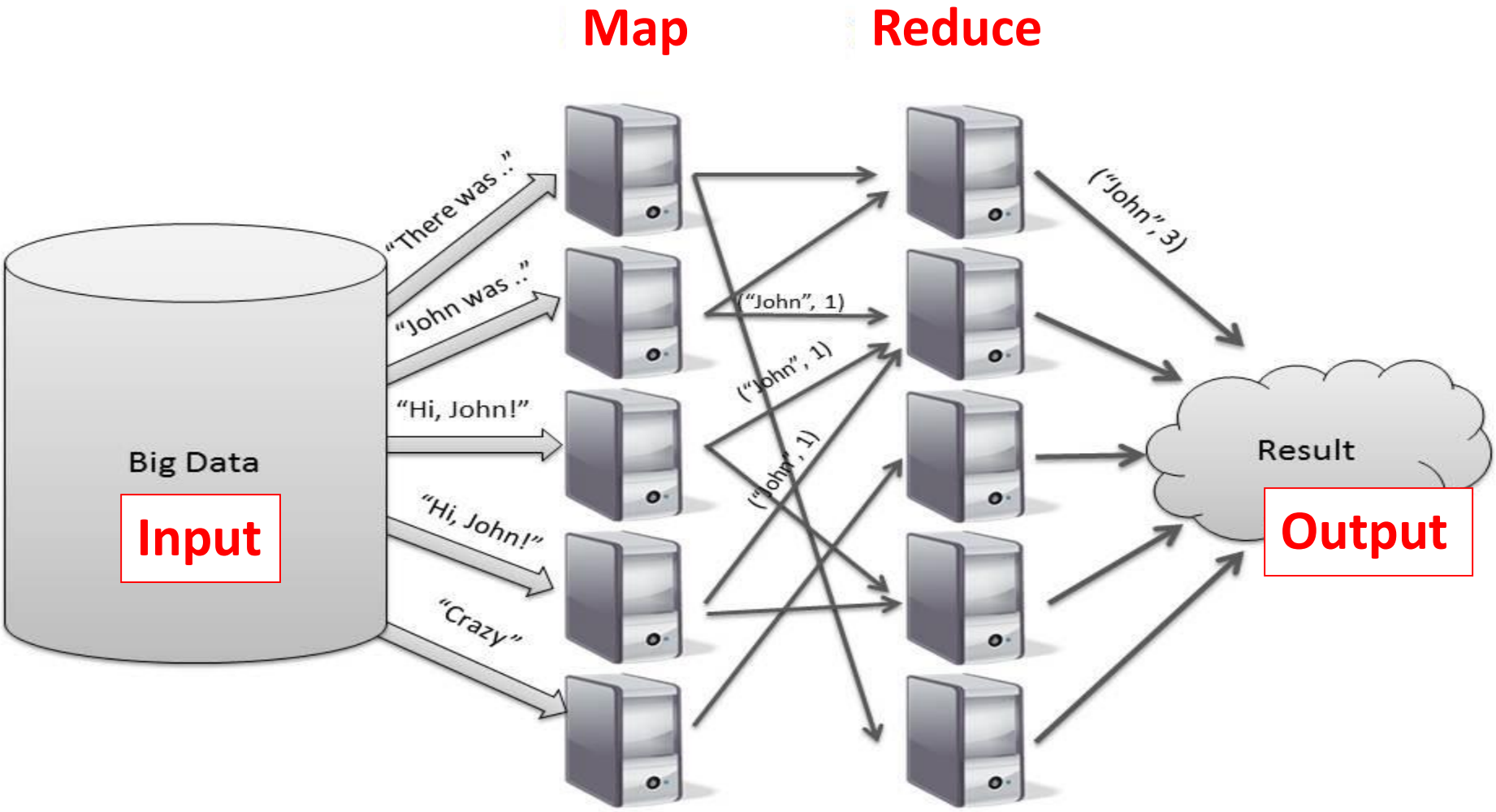
Need of Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.

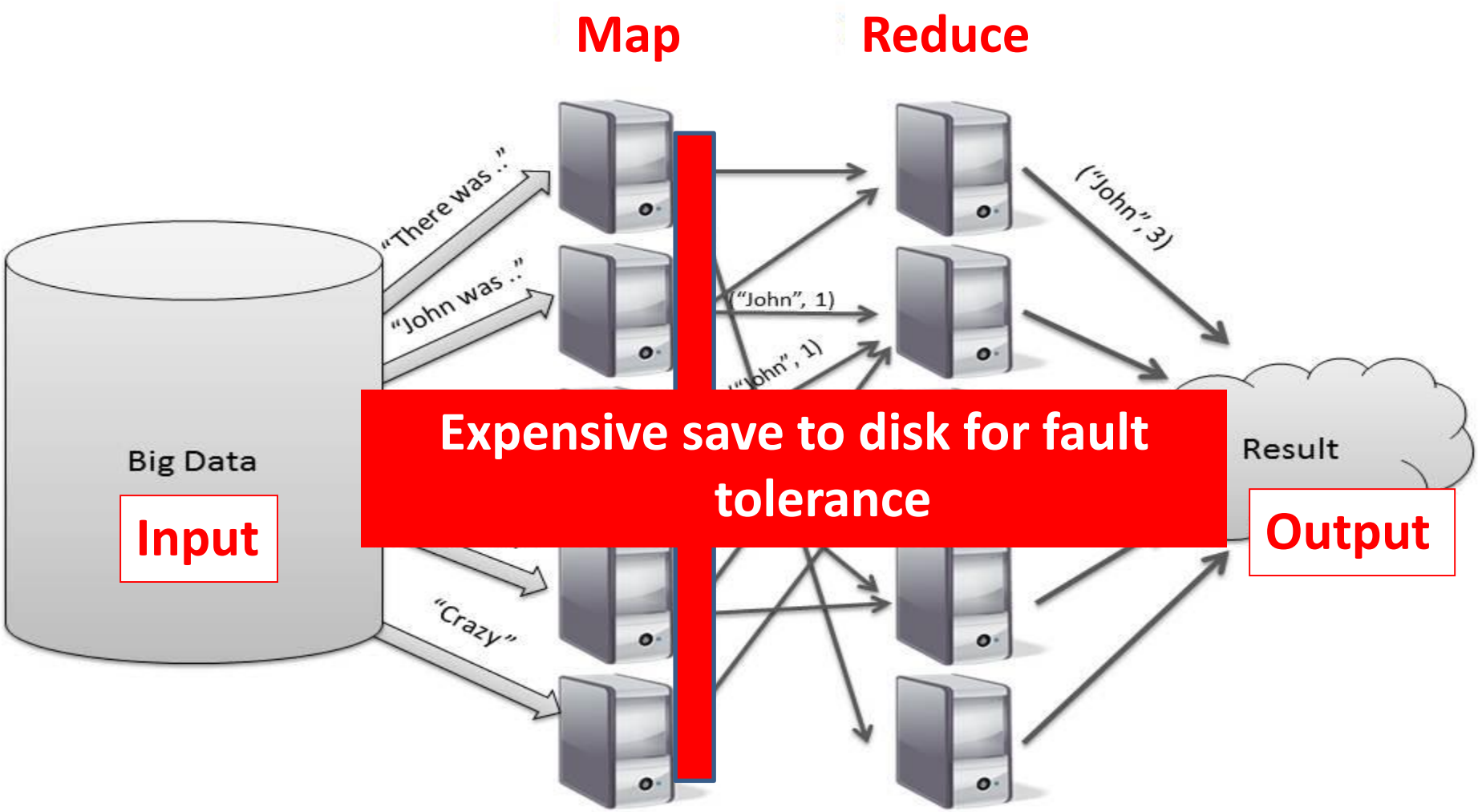
- It is an another system for big data analytics

- **Isn't MapReduce good enough?**
  - **Simplifies batch processing on large commodity clusters**

Need of Spark



Need of Spark



Need of Spark

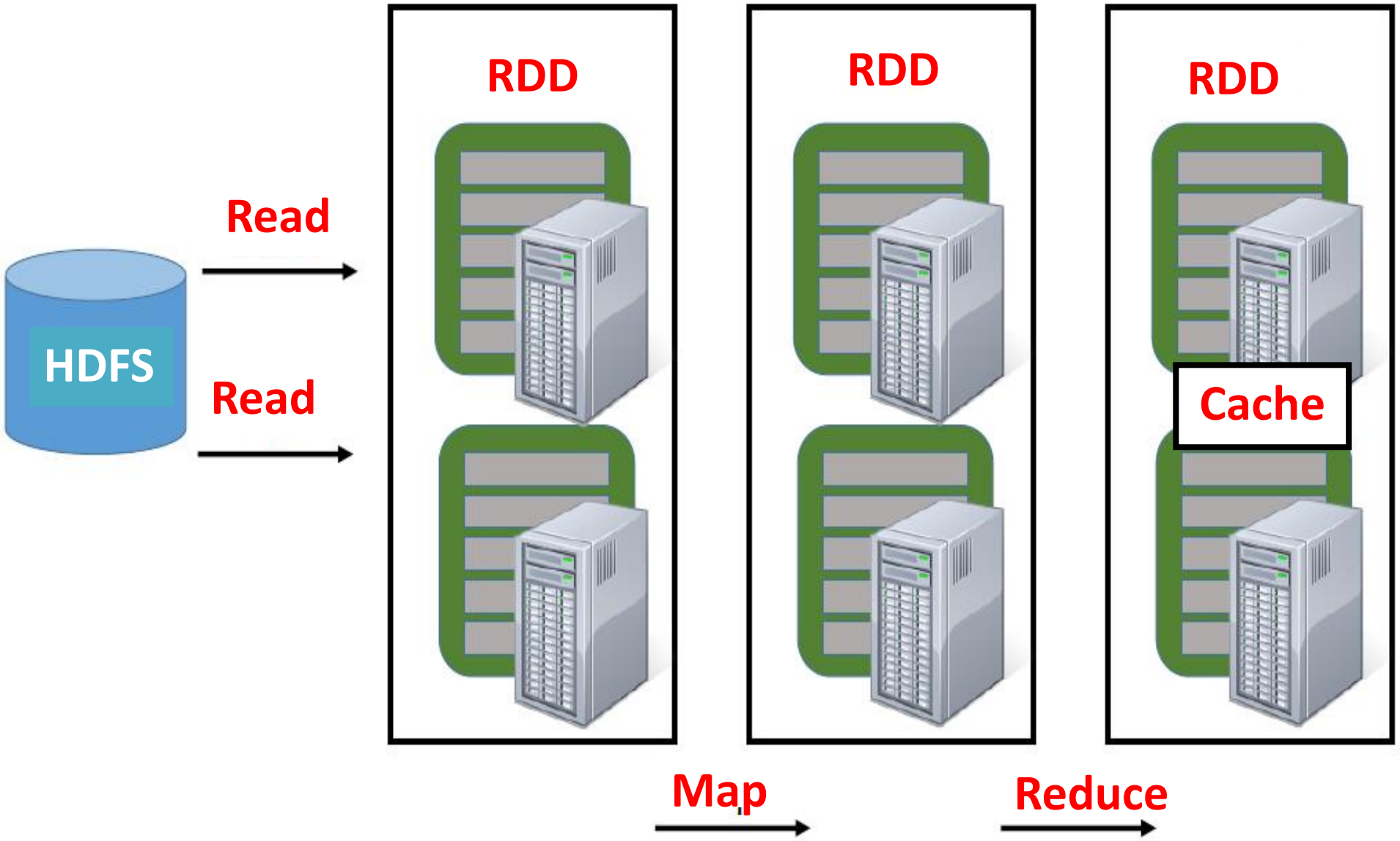
- MapReduce can be expensive for some applications e.g.,
  - **Iterative**
  - **Interactive**
- Lacks efficient data sharing
- Specialized frameworks did evolve for different programming models
  - **Bulk Synchronous Processing (Pregel)**
  - **Iterative MapReduce (Hadoop) ….**

Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)
- Can be cached for efficient reuse

Need of Spark



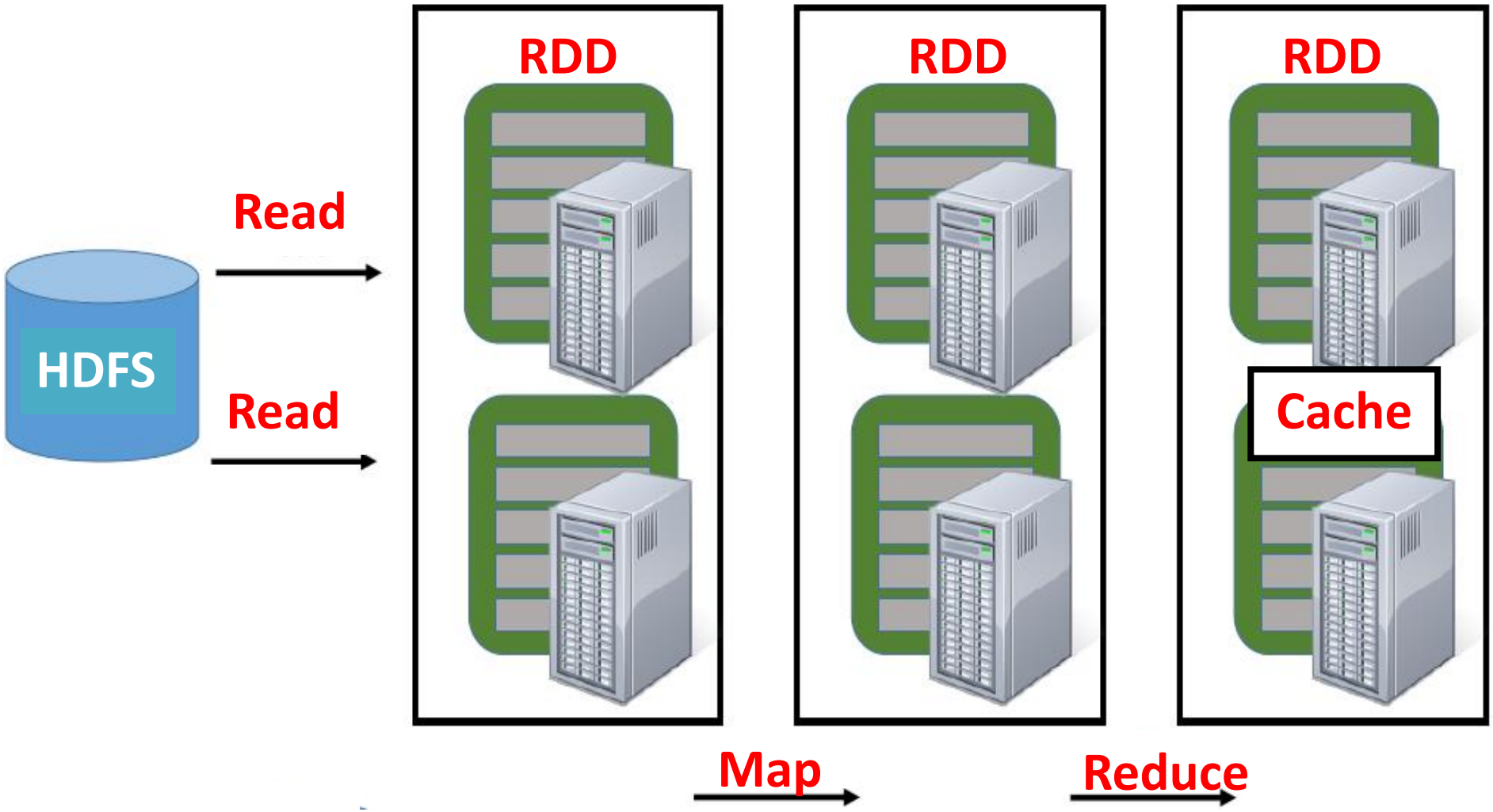
Solution: Resilient Distributed Datasets (RDDs)

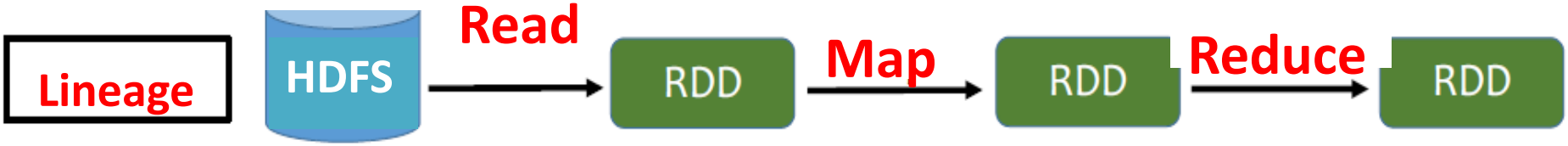
Resilient Distributed Datasets (RDDs)

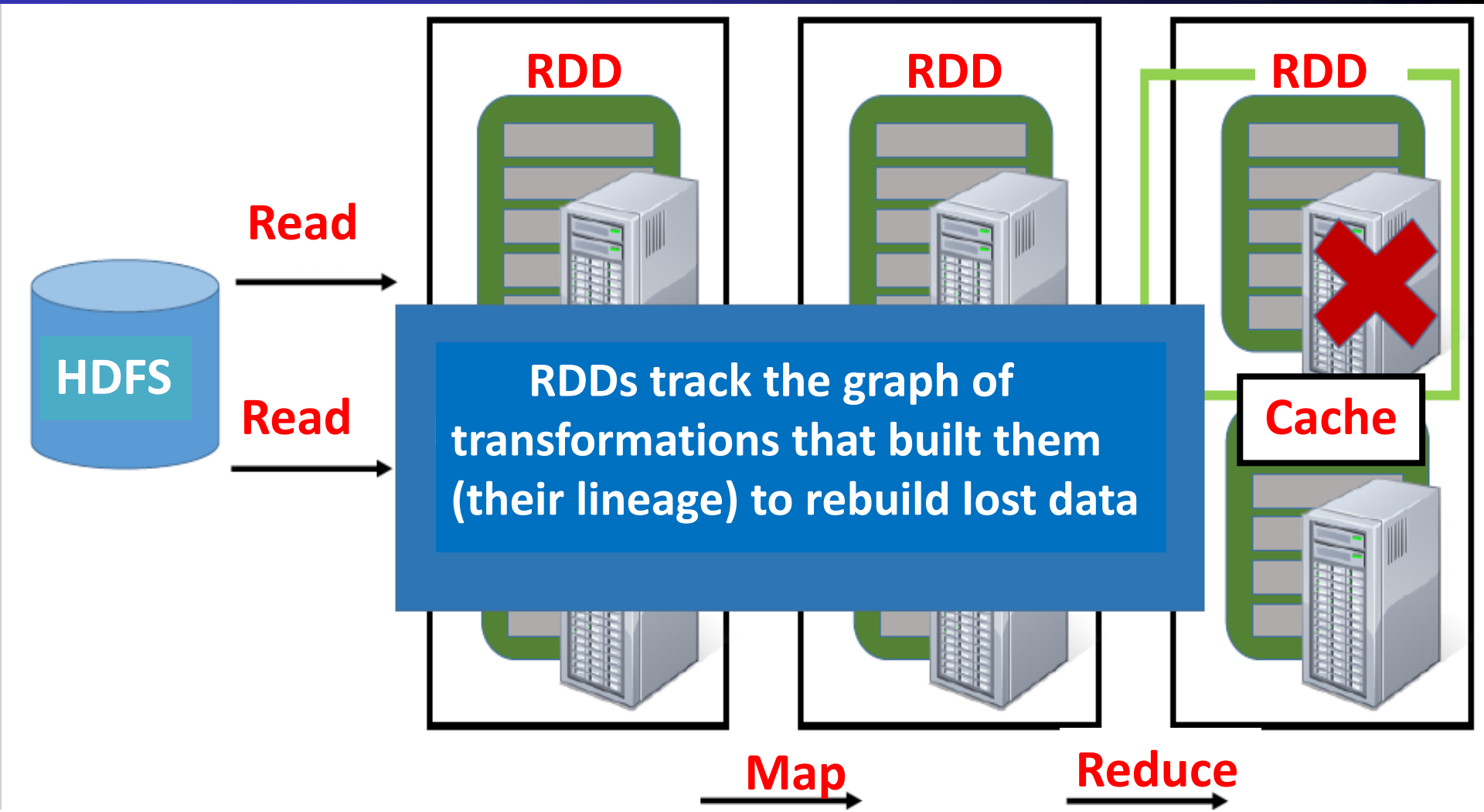
- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)

Fault Recovery?

- Lineage!
  - Log the coarse grained operation applied to a partitioned dataset
  - Simply recompute the lost partition if failure occurs!
  - No cost if no failure


RDD
RDD
RDD
Read
Read
HDFS
Cache
Map
Reduce


Lineage
HDFS
Read
RDD
Map
RDD
Reduce
RDD


RDD
RDD
RDD
Read
HDFS
Read
RDDs track the graph of transformations that built them (their lineage) to rebuild lost data
Cache
Map
Reduce

What can you do with Spark?

- **RDD operations**
  - Transformations e.g., filter, join, map, group-by …
  - Actions e.g., count, print …
- **Control**
  - **Partitioning:** Spark also gives you control over how you can partition your RDDs.
  - **Persistence:** Allows you to choose whether you want to persist RDD onto disk or not.

Spark Applications

i. Twitter spam classification

ii. EM algorithm for traffic prediction

iii. K-means clustering

iv. Alternating Least Squares matrix factorization

v. In-memory OLAP aggregation on Hive data

vi. SQL on Spark

Reading Material

- Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica

"Spark: Cluster Computing with Working Sets"

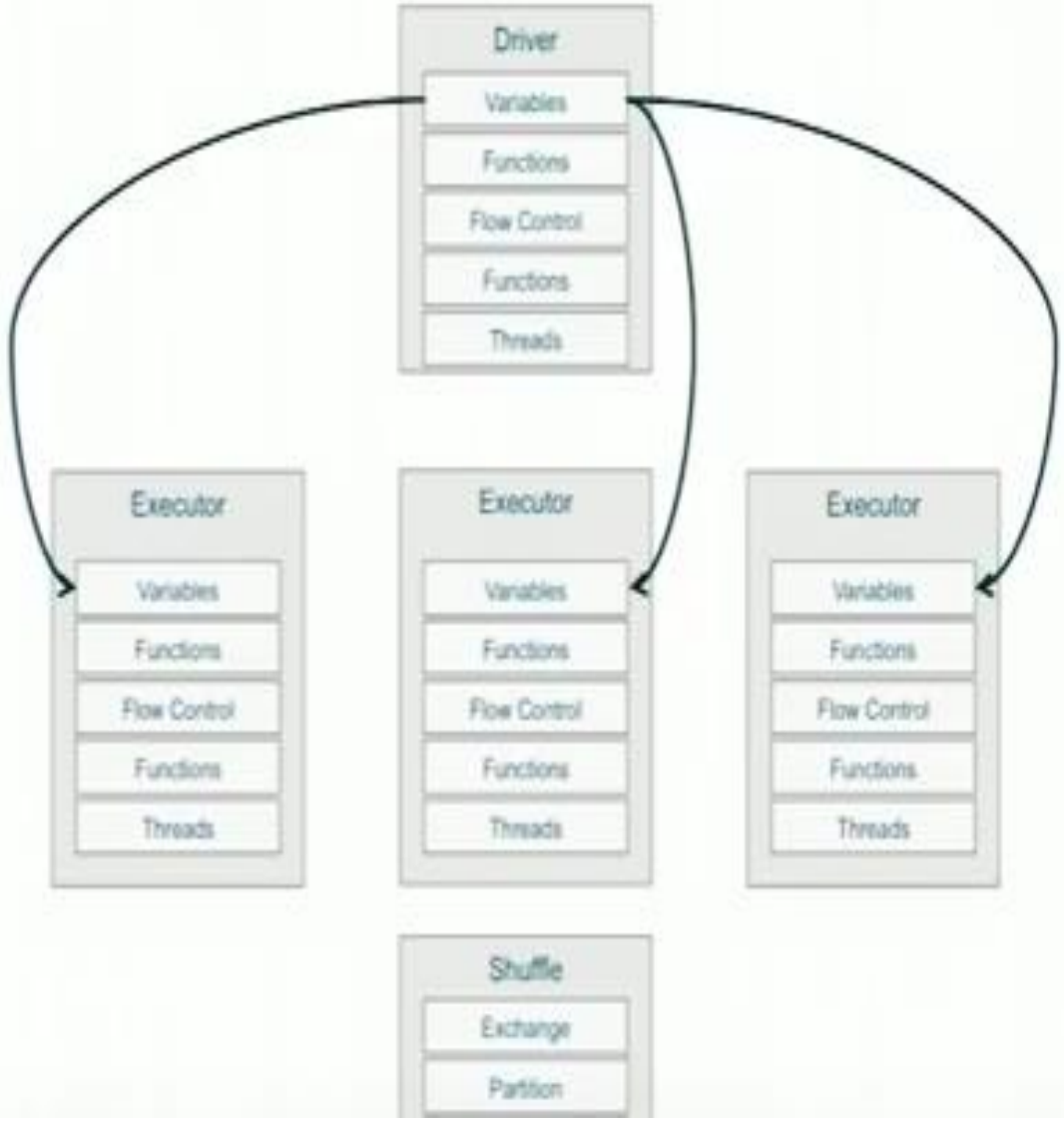
- Matei Zaharia, Mosharaf Chowdhury et al.

"Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"

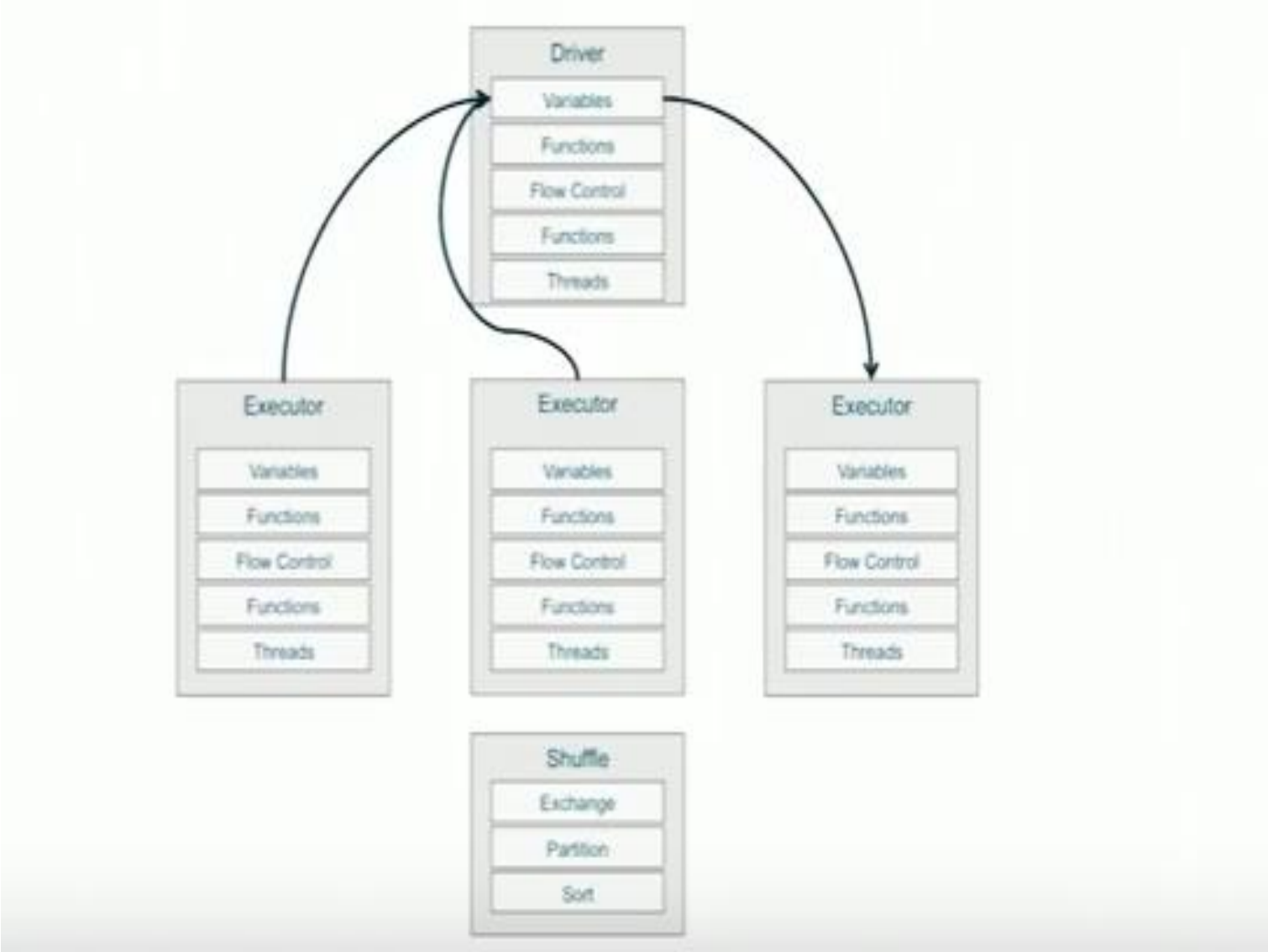
https://spark.apache.org/

Spark Execution

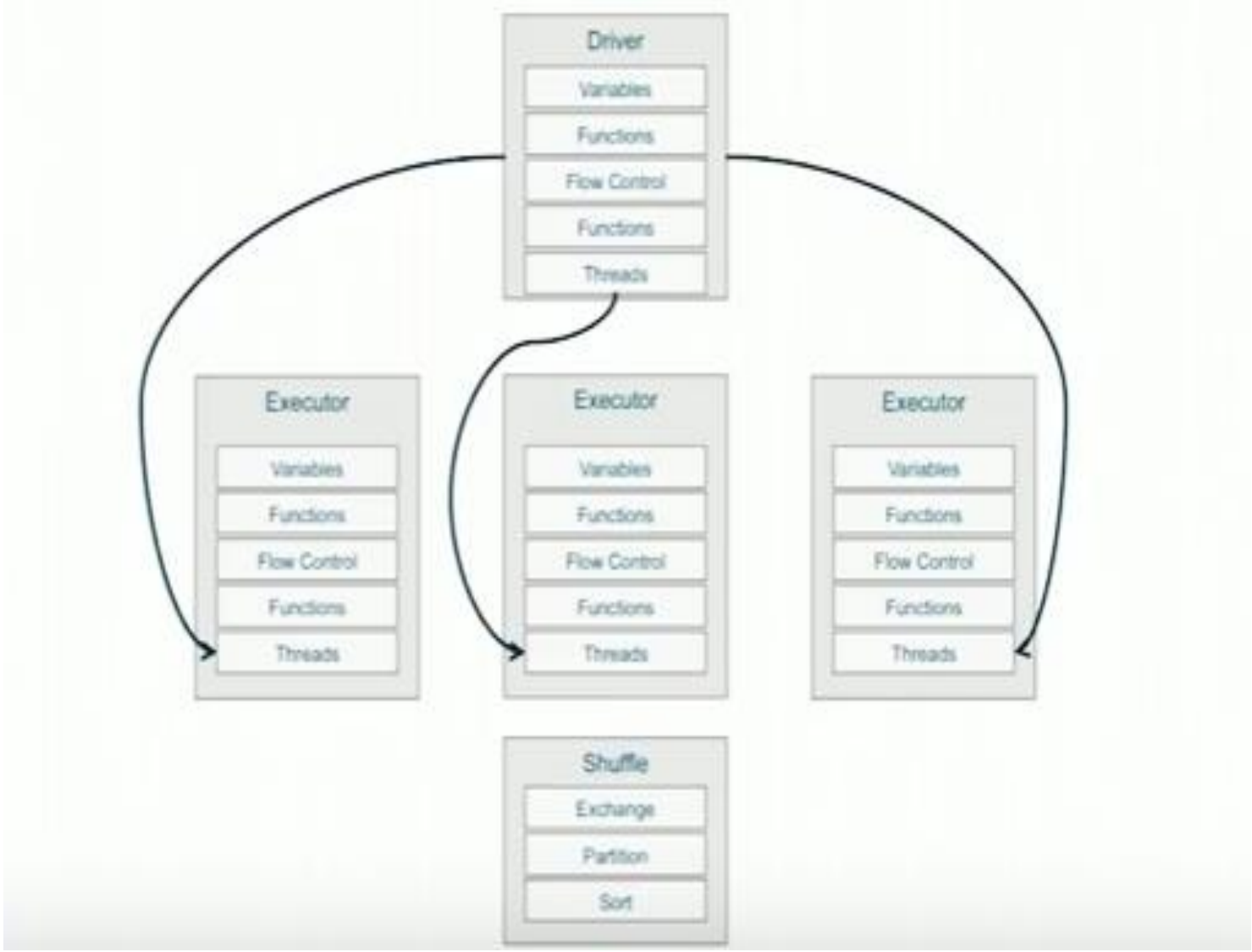
Distributed Programming (Broadcast)



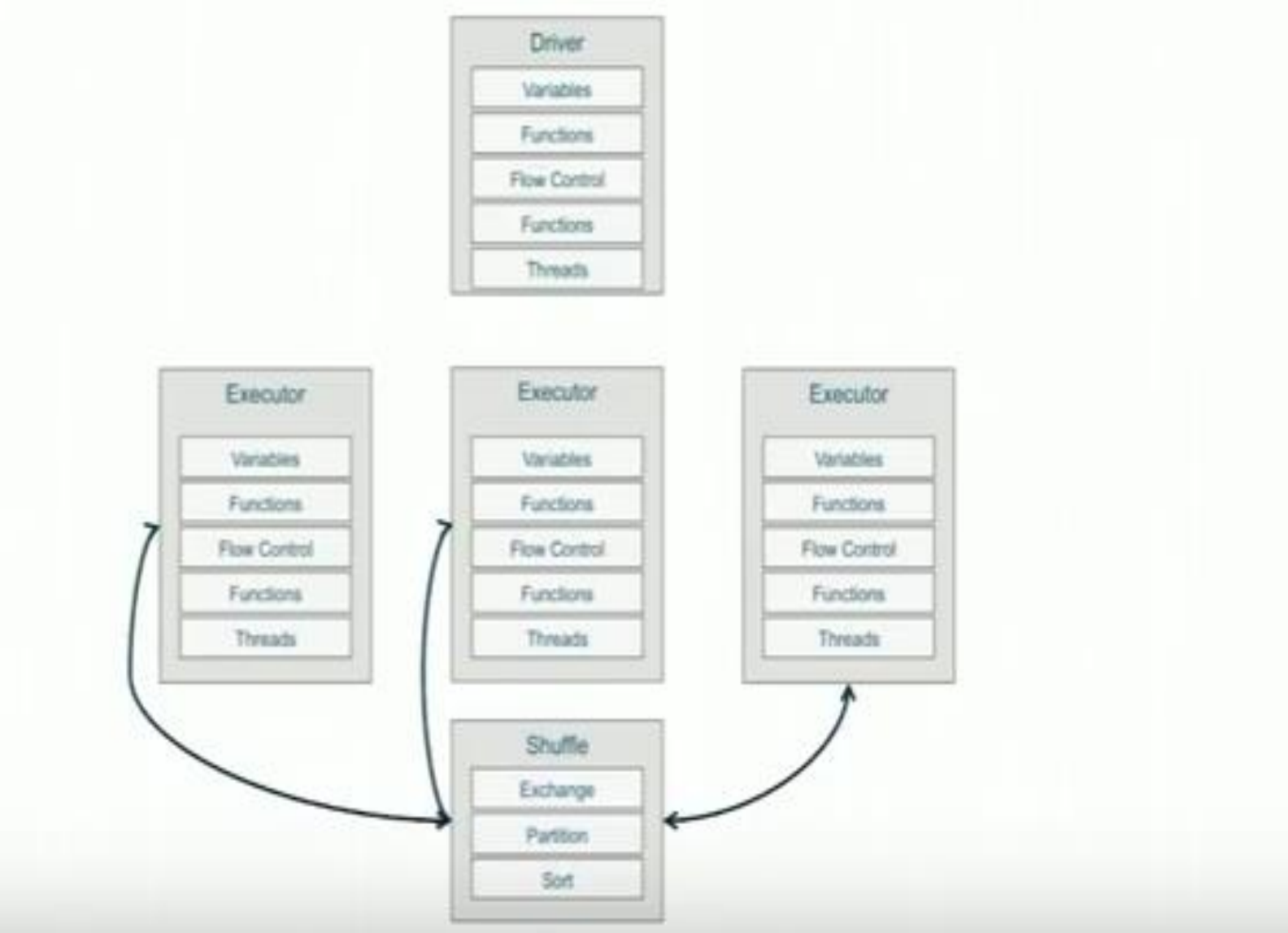
Distributed Programming (Take)



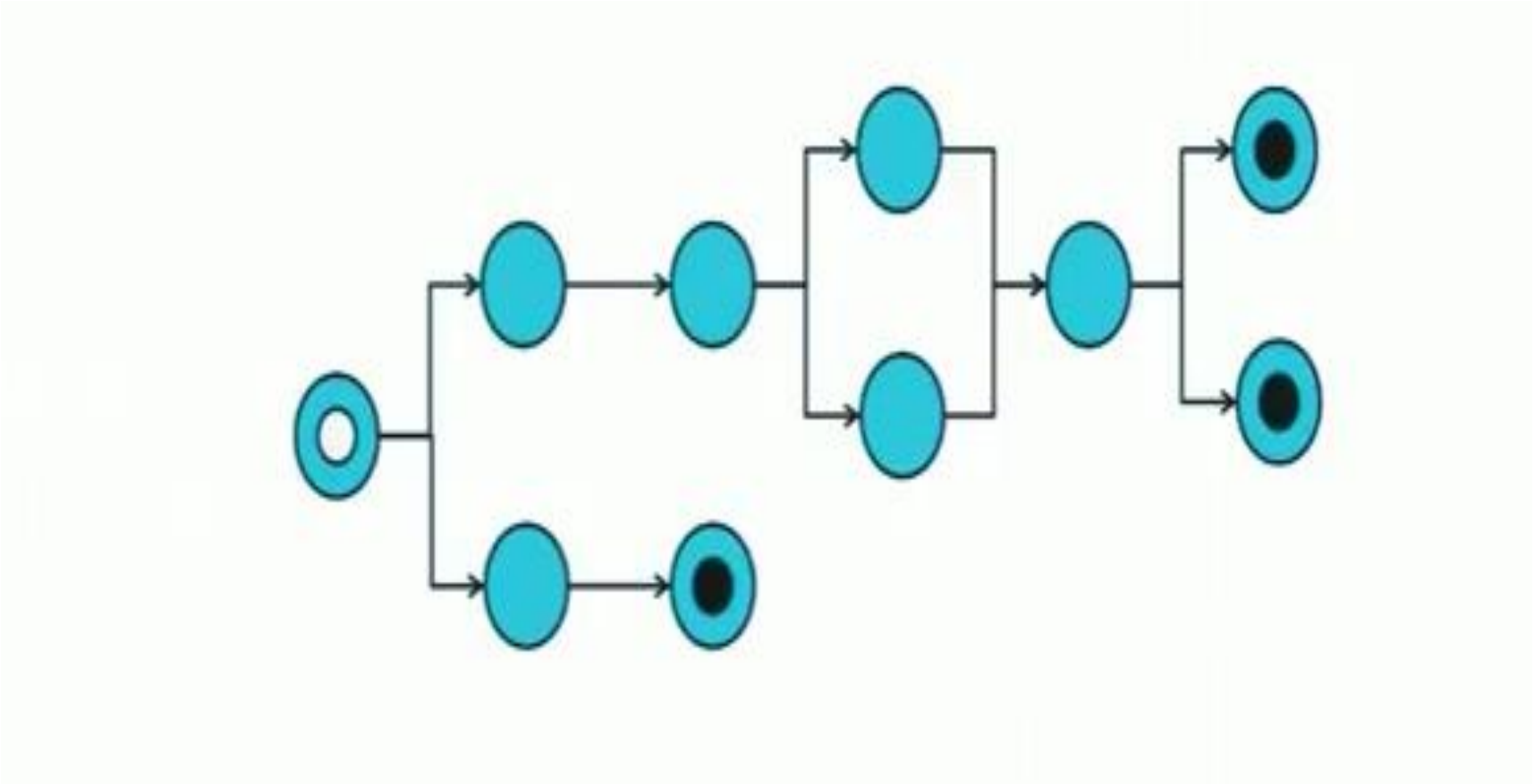
Distributed Programming (DAG Action)



Distributed Programming (Shuffle)



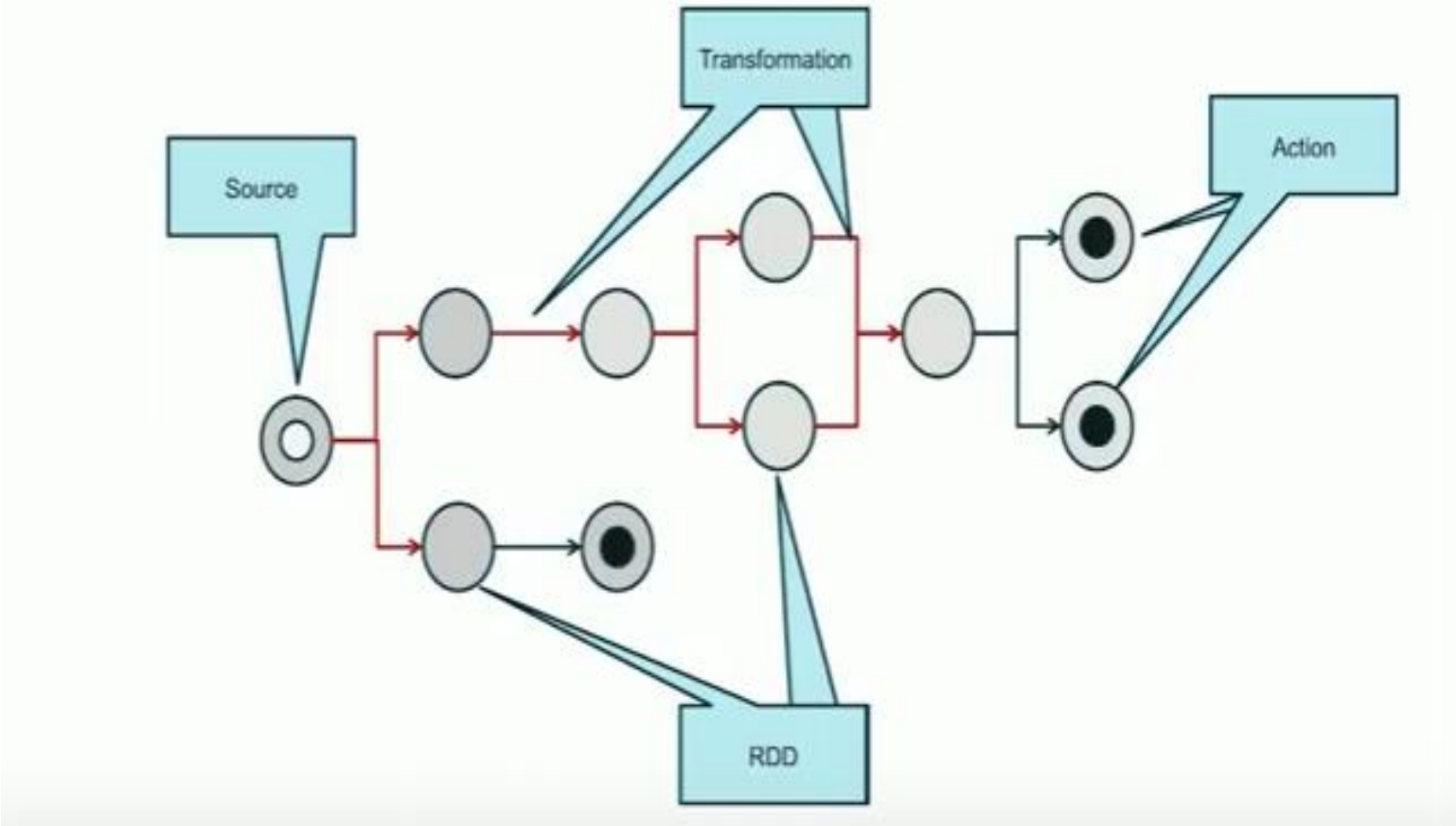
DAG (Directed Acyclic Graph)



DAG (Directed Acyclic Graph)

- **Action**
  - Count
  - Take
  - Foreach
- **Transformation**
  - Map
  - ReduceByKey
  - GroupByKey
  - JoinByKey

DAG (Directed Acyclic Graph)



Flume Java

```
1. val conf = new SparkConf().setMaster("local[2]")
2. val sc = new SparkContext(conf)
3. val lines = sc.textFile(path, 2)
4. val words = lines.flatMap(_.split(" "))
5. val pairs = words.map(word => (word, 1))
6. val wordCounts = pairs.reduceByKey(_ + _)
7. val localValues = wordCounts.take(100)
8. localValues.foreach(r => println(r))
```

Spark Implementation

Spark ideas

- Expressive computing system, not limited to map-reduce model
- Facilitate system memory
  - avoid saving intermediate results to disk
  - cache data for repetitive queries (e.g. for machine learning)
- Compatible with Hadoop

RDD abstraction

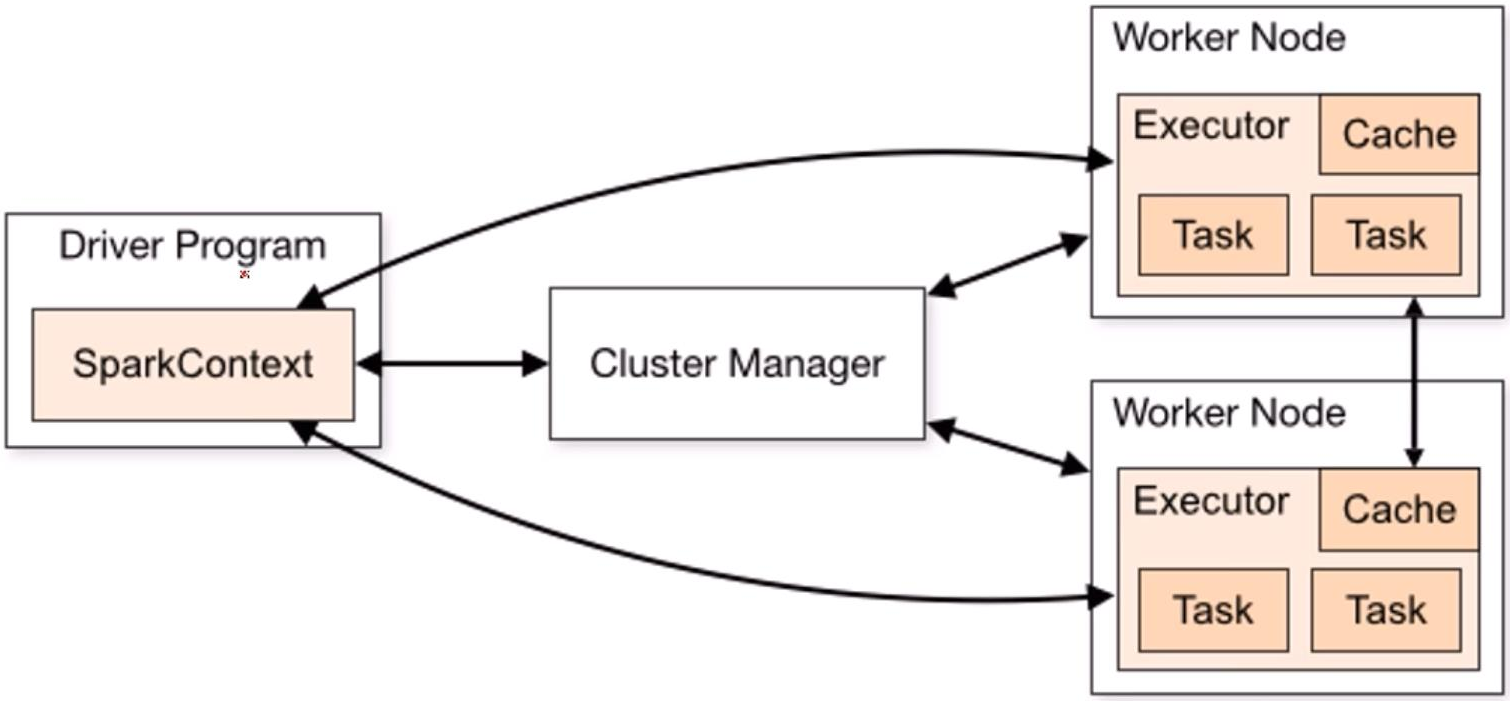
- Resilient Distributed Datasets
- Partitioned collection of records
- Spread across the cluster
- Read-only
- Caching dataset in memory
  - different storage levels available
  - fallback to disk possible

RDD operations

- *Transformations* to build RDDs through deterministic operations on other RDDs
  - transformations include *map*, *filter*, *join*
  - lazy operation

- *Actions* to return value or export data
  - actions include *count*, *collect*, *save*
  - triggers execution

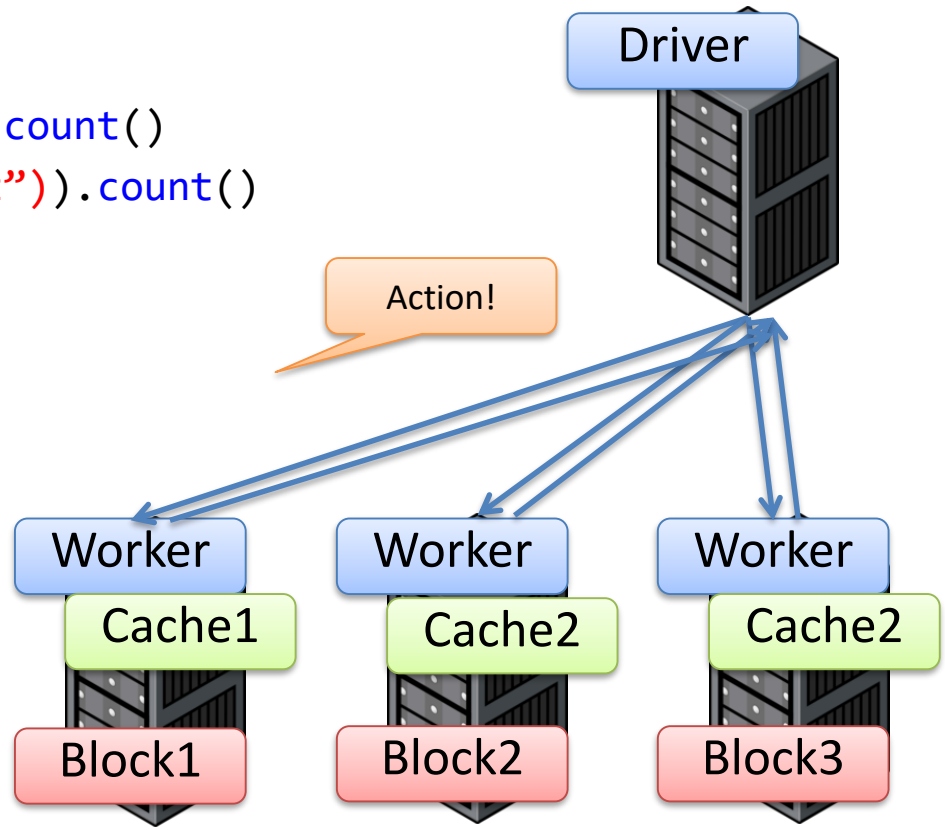
Spark Components



Job example

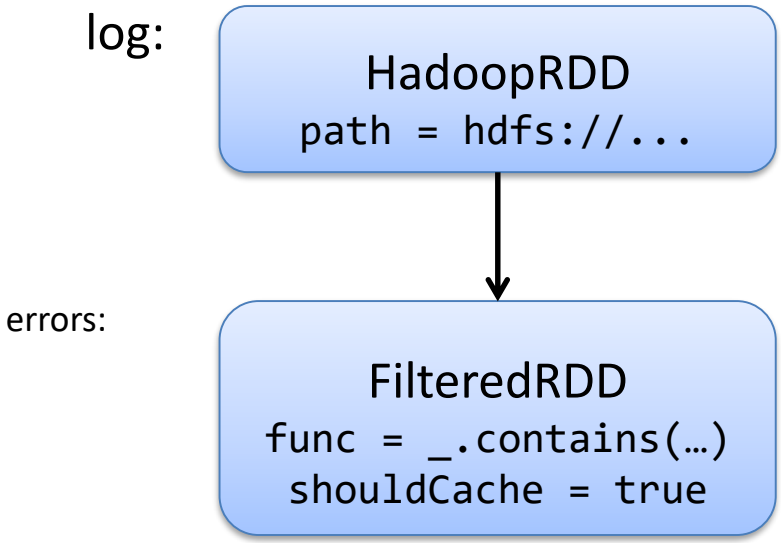
```
val log = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()

errors.filter(_.contains("I/O")).count()
errors.filter(_.contains("timeout")).count()
```

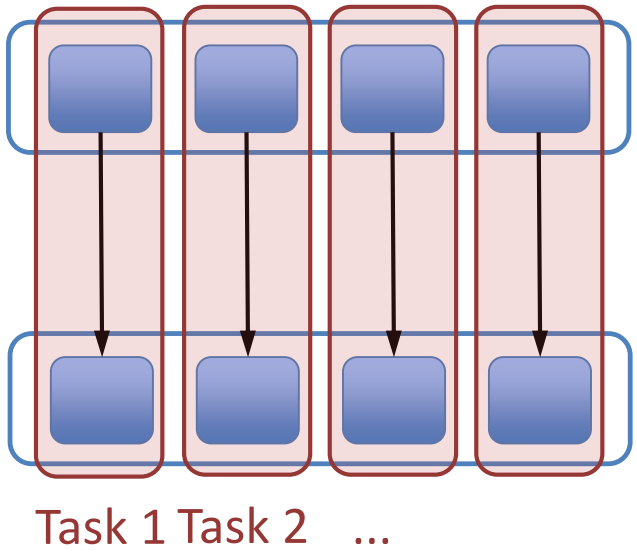


RDD partition-level view

Dataset-level view:

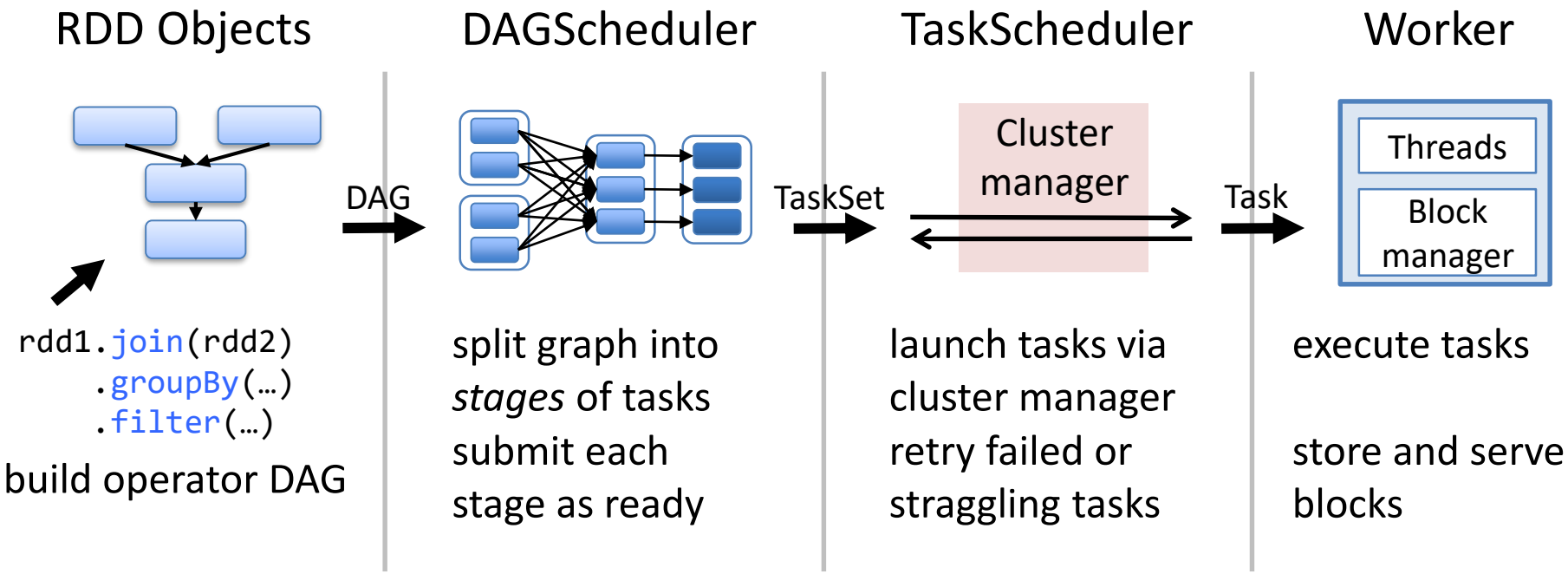


Partition-level view:



source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

Job scheduling



source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

Available APIs

- You can write in Java, Scala or Python
- Interactive interpreter: Scala & Python only
- Standalone applications: any
- Performance: Java & Scala are faster thanks to static typing

Hand on - interpreter

- script

http://cern.ch/kacper/spark.txt

- run scala spark interpreter

```
$ spark-shell
```

- or python interpreter

```
$ pyspark
```

Hand on – build and submission

- download and unpack source code

```
wget http://cern.ch/kacper/GvaWeather.tar.gz; tar -xzf GvaWeather.tar.gz
```

- build definition in

```
GvaWeather/gvaweather.sbt
```

- source code

```
GvaWeather/src/main/scala/GvaWeather.scala
```

- building

```
cd GvaWeather
sbt package
```

- job submission

```
spark-submit --master local --class GvaWeather \
        target/scala-2.10/gva-weather_2.10-1.0.jar
```

Summary

- Concept not limited to single pass map-reduce
- Avoid sorting intermediate results on disk or HDFS
- Speedup computations when reusing datasets

Conclusion

- RDDs (Resilient Distributed Datasets (RDDs) provide a simple and efficient programming model
- Generalized to a broad set of applications
- Leverages coarse-grained nature of parallel algorithms for failure recovery