Resilient Distributed Datasets

A fault tolerant abstraction for in-memory cluster computing

By: Vinod Dalavai
Date: Feb 16, 2022

Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology, NY

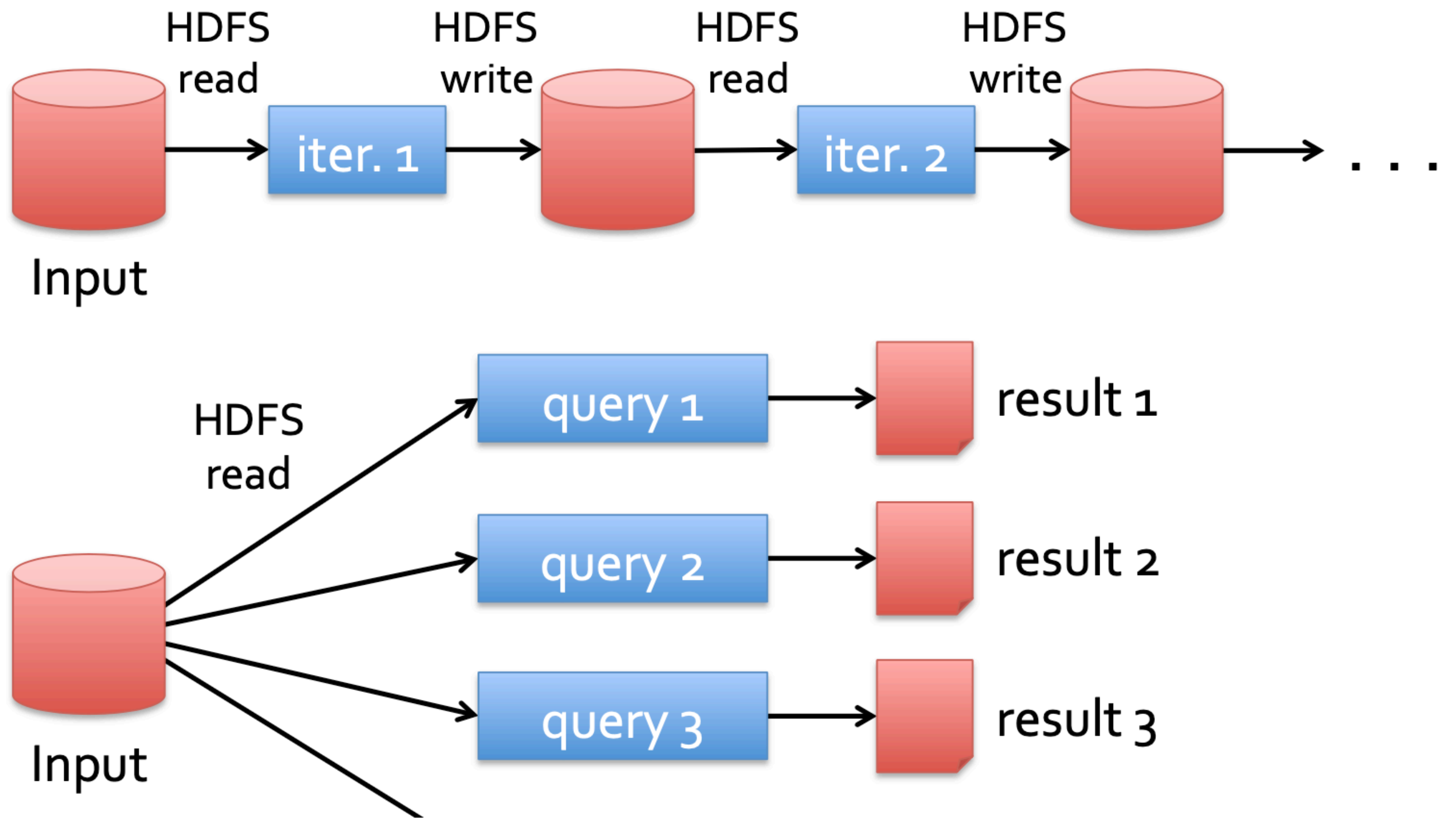
What is RDD

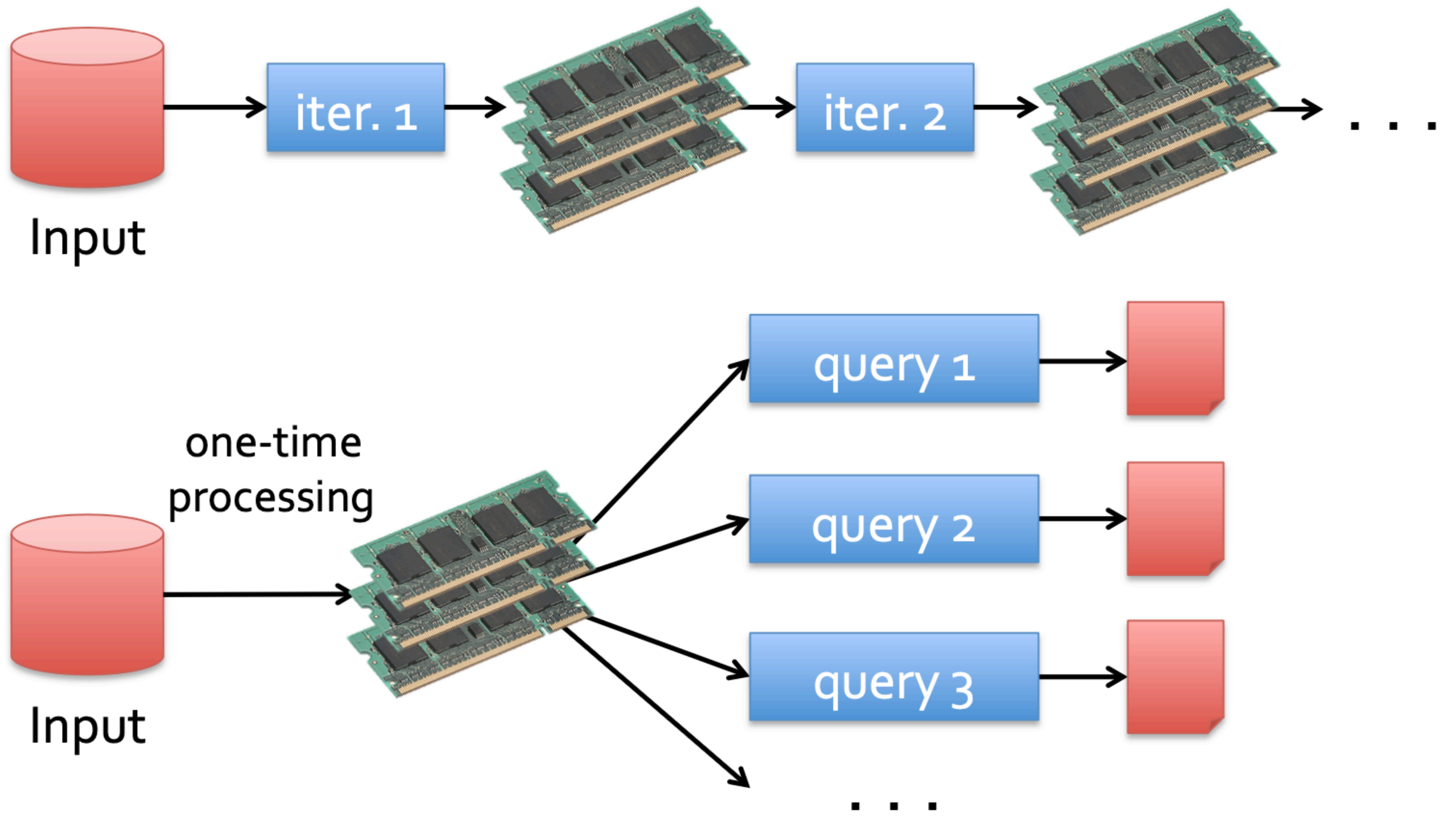
- ***Resilient -*** meaning, fault tolerant. Can recompute in the event of a network partition
- ***Distributed -*** meaning, it resides in multiple nodes
- ***Dataset -*** meaning, records of data with which programmers will work

- Solves the problem of iterative algorithms and interactive data mining tools that current computing frameworks handle inefficiently by keeping data in-memory

Limitations of existing approaches

- Hadoop’s MapReduce simplified “big data” analysis by performing parallel computations on data while being fault tolerant.
- But, users wanted more. More complex iterative algorithms and interactive ad-hoc queries
- Hence, specialized frameworks such as Pregel and HaLoop were introduced that kept intermediate data in-memory.
- But they were narrow focussed and not for more general reuse of data. eg) let a user load several datasets and run ad-hoc queries on the same subset of data
- There was a need for efficient primitives for data sharing.





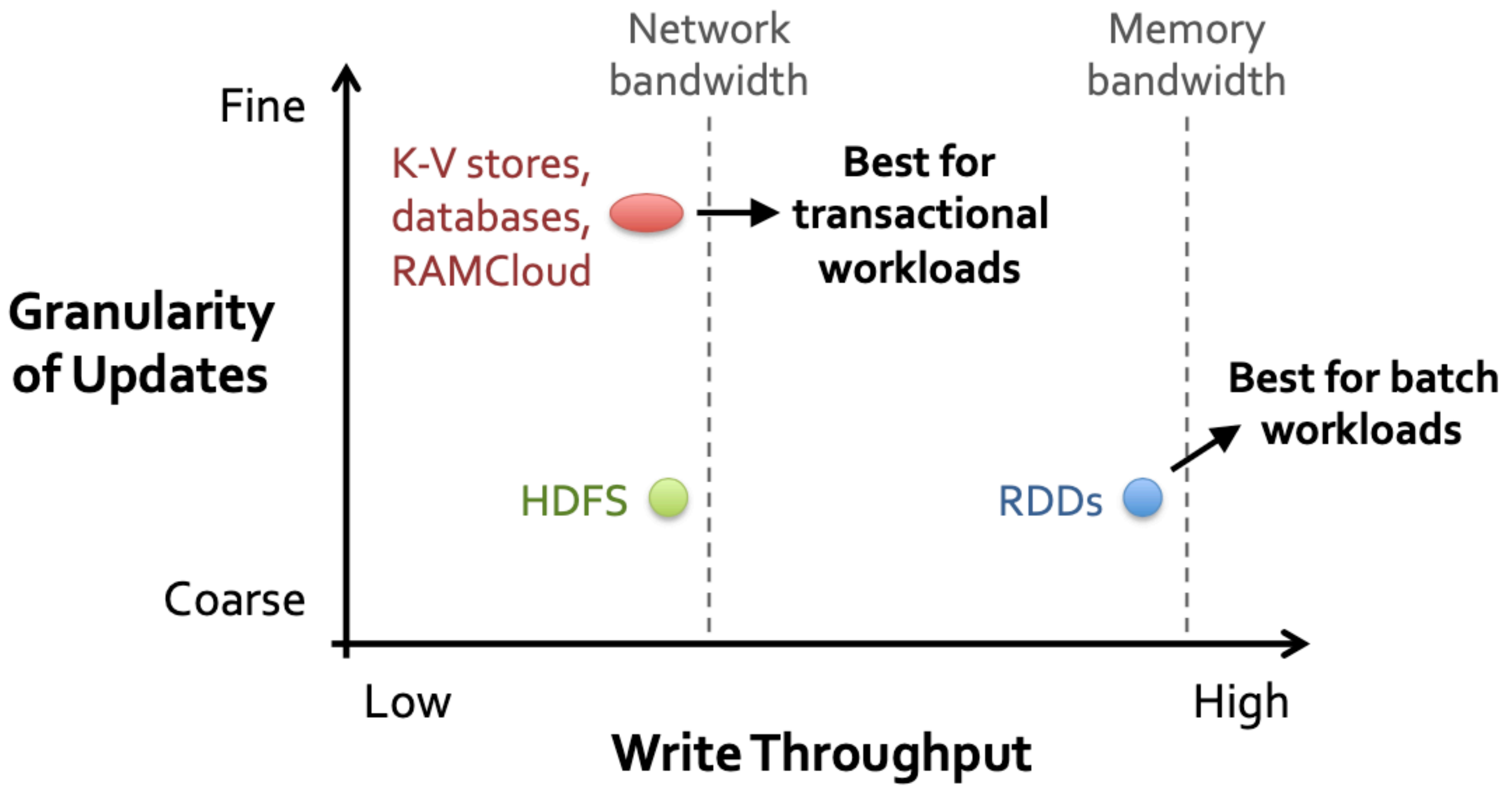
Challenges

- Existing storage abstractions have interfaces based on fine-grained updates to mutable state. For instance, DSM, Piccolo, key-value stores
- Require replicating data or logs across multiple nodes for fault tolerance
  - Expensive operations to write large amounts of data or logs
  - Limited by network bandwidth
  - Much slower compared to memory write

Solution: RDDs

- Restricted form of DSM
- Immutable and partitioned collection of records
- Can only be built through coarse-grained deterministic transformations
- Uses lineage for efficient fault recovery
- In case of a failure, recompute only the lost partitions
- Can be used to apply the same operation to many items

Trade-off Space



RDDs and Spark

- The concept of RDD is implemented in Spark as a language-integrated API
- Dataset is represented as an object and transformations are invoked on these objects
- Usable interactively from Scala interpreter
- Provides:
    - Operations on RDDs: ***transformations***(create new RDDs) and ***actions***(compute and output results on the RDDs)
    - Control of each RDD's ***partitioning*** (layout across multiple nodes) and ***persistence*** (storage in RAM, on disk, etc)

Implementation

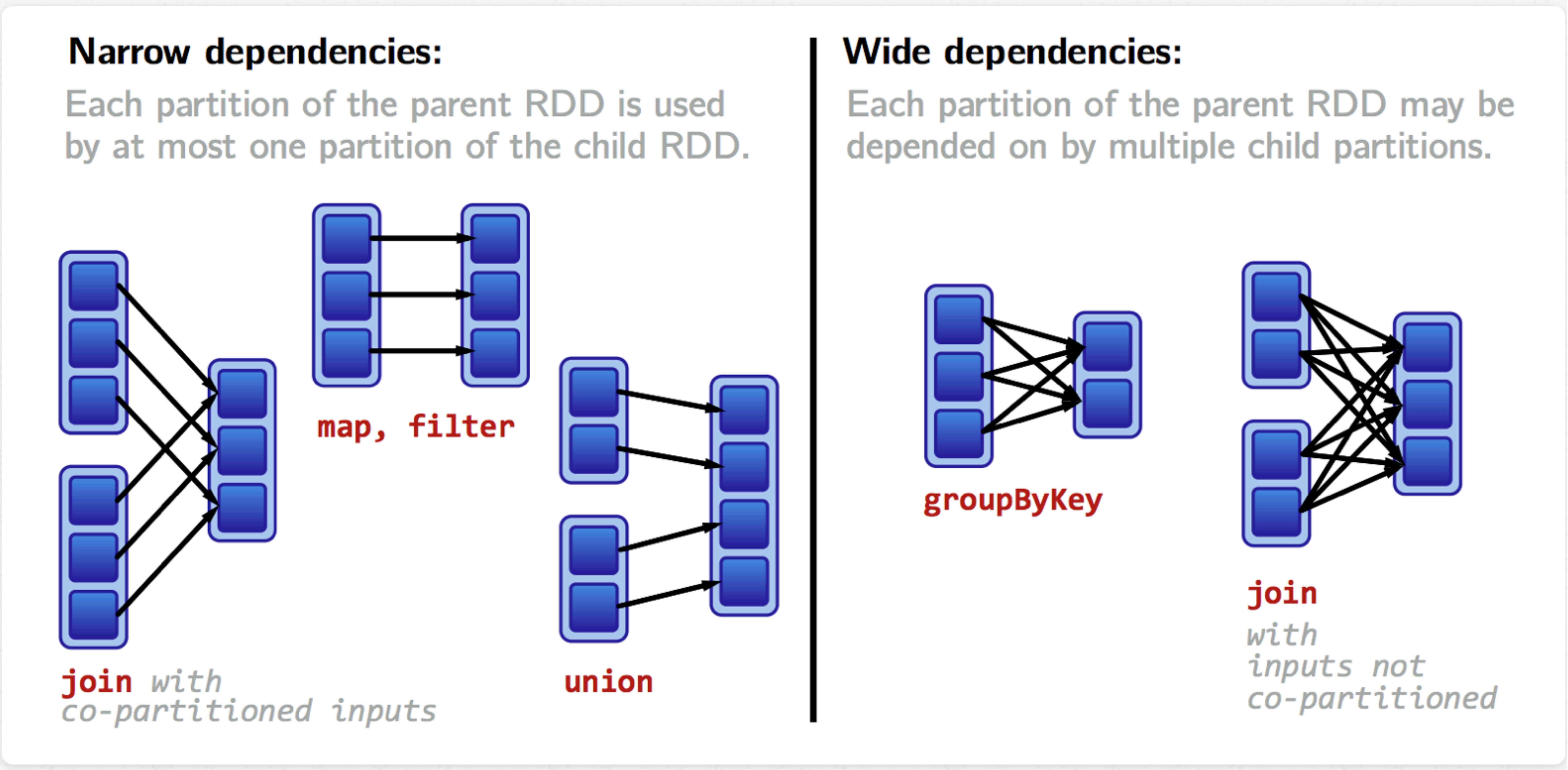
- Implemented Spark in 14000 lines of Scala.
- System runs over Mesos cluster manager and shares resources with Hadoop and other applications
- Each Spark application runs as a separate Mesos application with its own driver and workers
- Spark can read data from any DFS such as HBase, Hadoop etc.

Representing RDDs

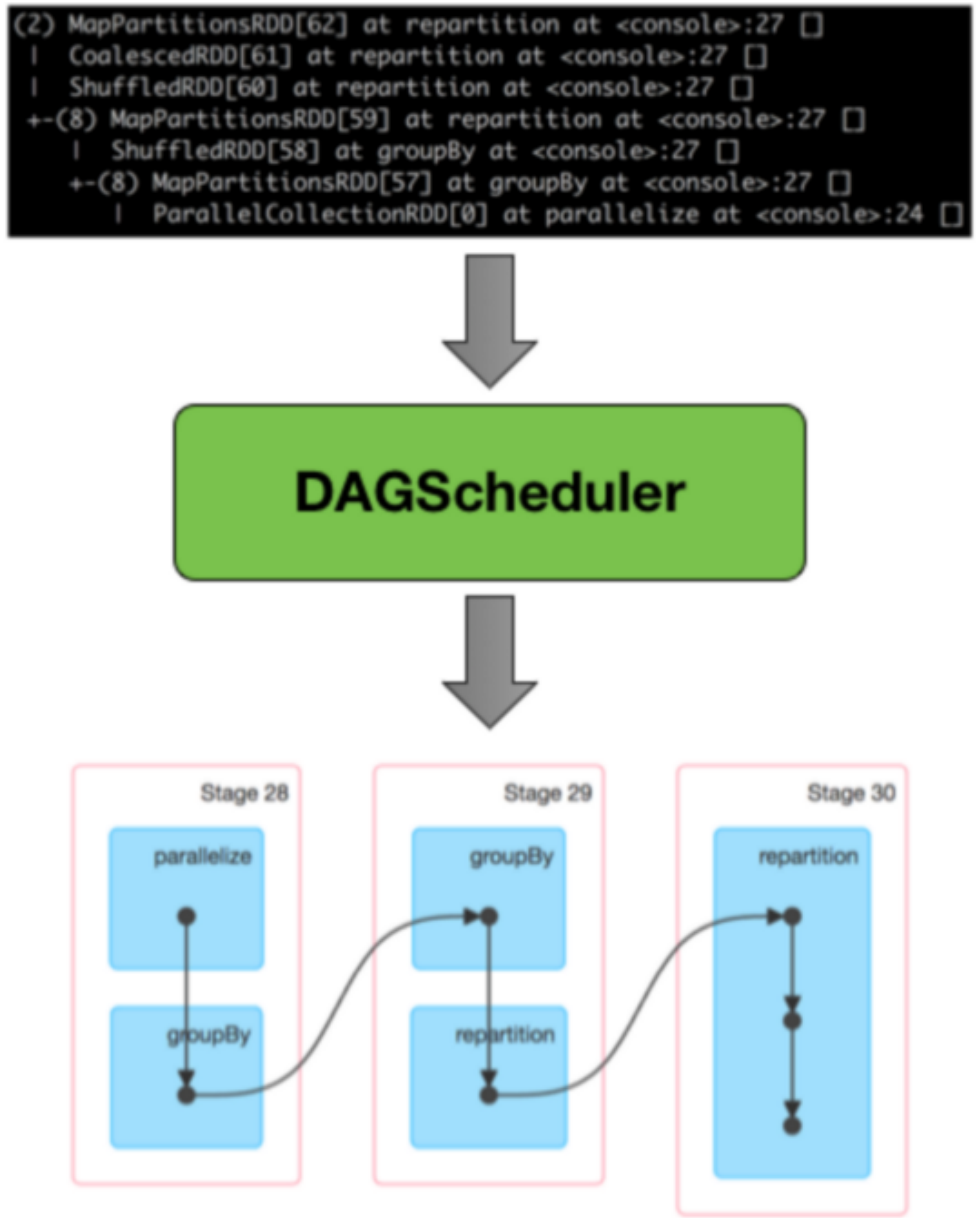
- RDD can be represented through a common interface that exposes five pieces of information:

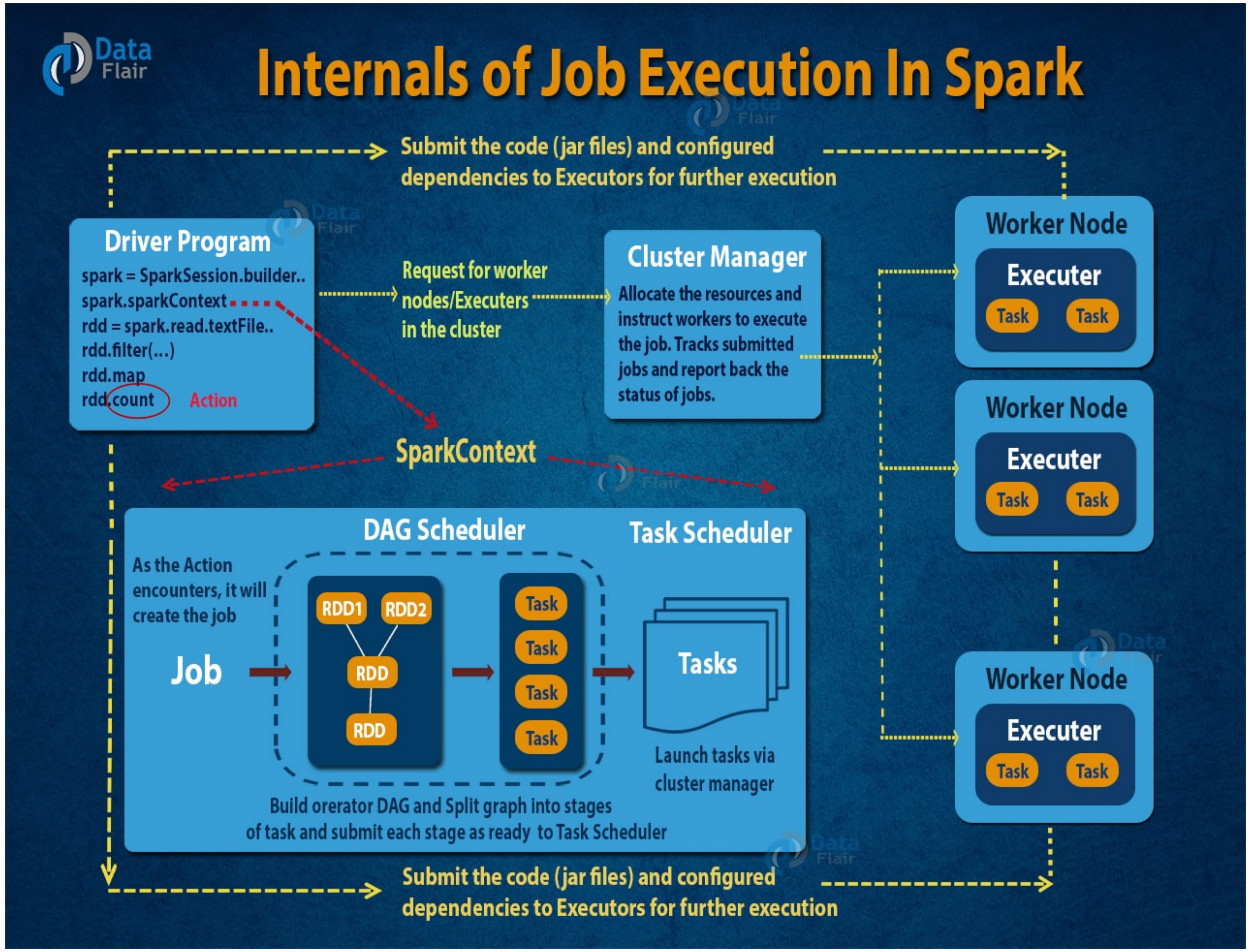
partitions()	atomic pieces of the dataset
preferredLocations(p)	List nodes where partition p can be accessed faster
dependencies()	set of dependencies on the parent RDD
iterator(p, parentIters)	function for computing the dataset of p based on parents
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

Types of dependencies



Lineage & Directed Acyclic Graph (DAG)

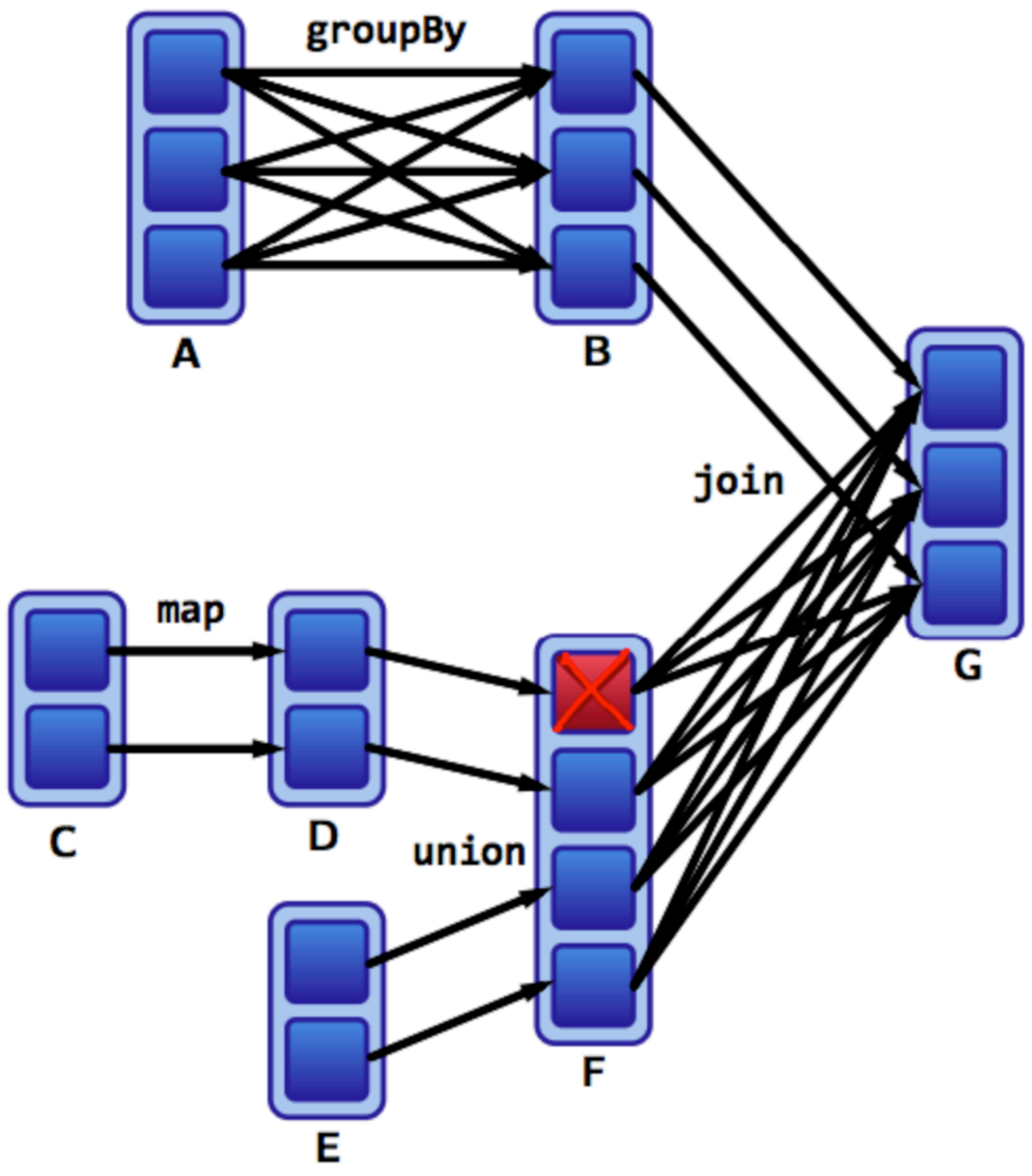


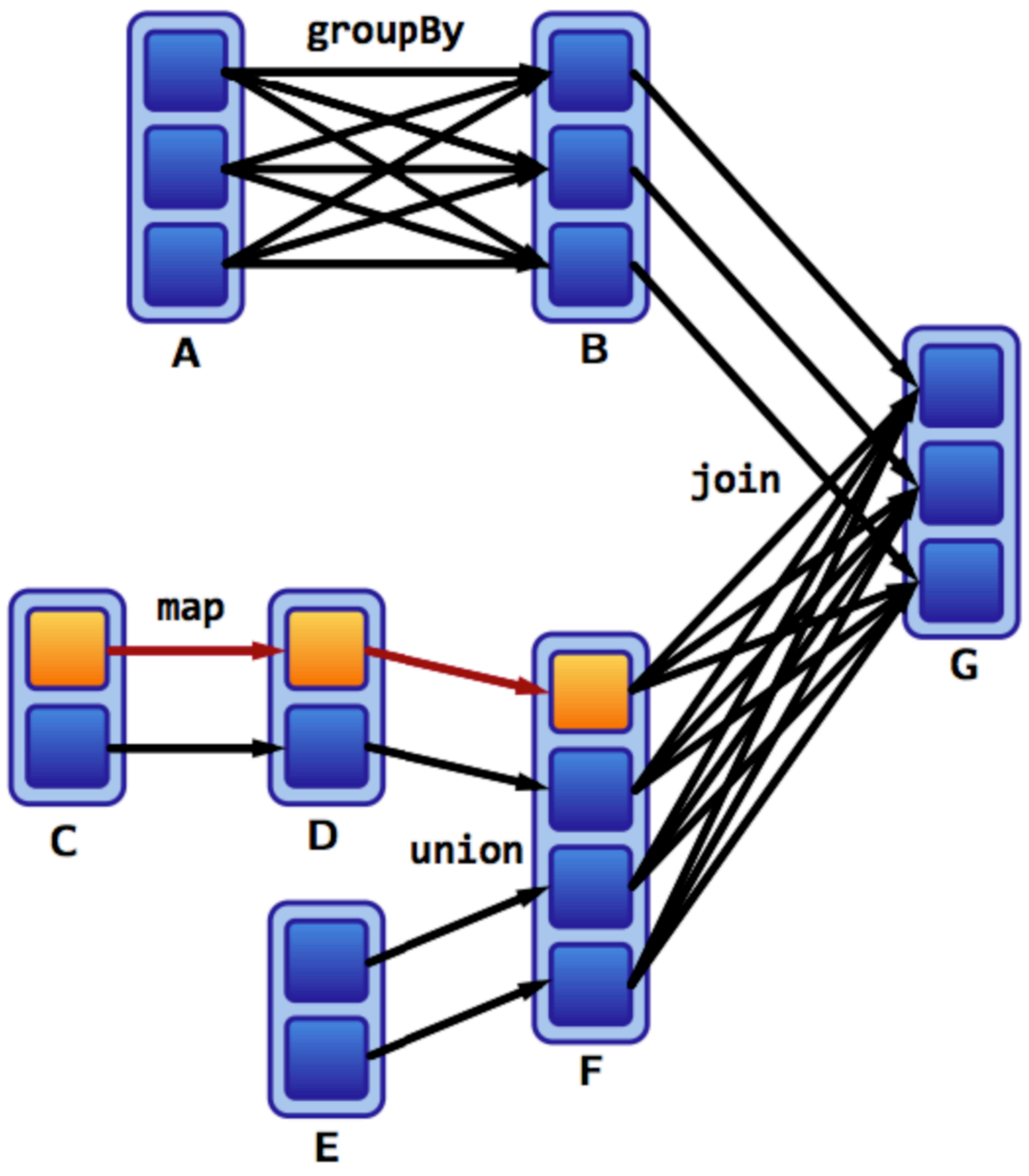


Fault Tolerance

- Lineages are key to fault tolerance in Spark
- Three other properties required to deliver fault tolerance:
  - Immutability of RDDs
  - Usage of higher order functions such as map, filter, flatMap to perform functional transformations on this immutable data
  - Function for computing the dataset based on parent RDD
- Along with keeping track of dependency information between partitions and the three properties mentioned above allow us to *recover from failures by recomputing lost partitions from lineage graphs*

Fault Tolerance (Contd..)

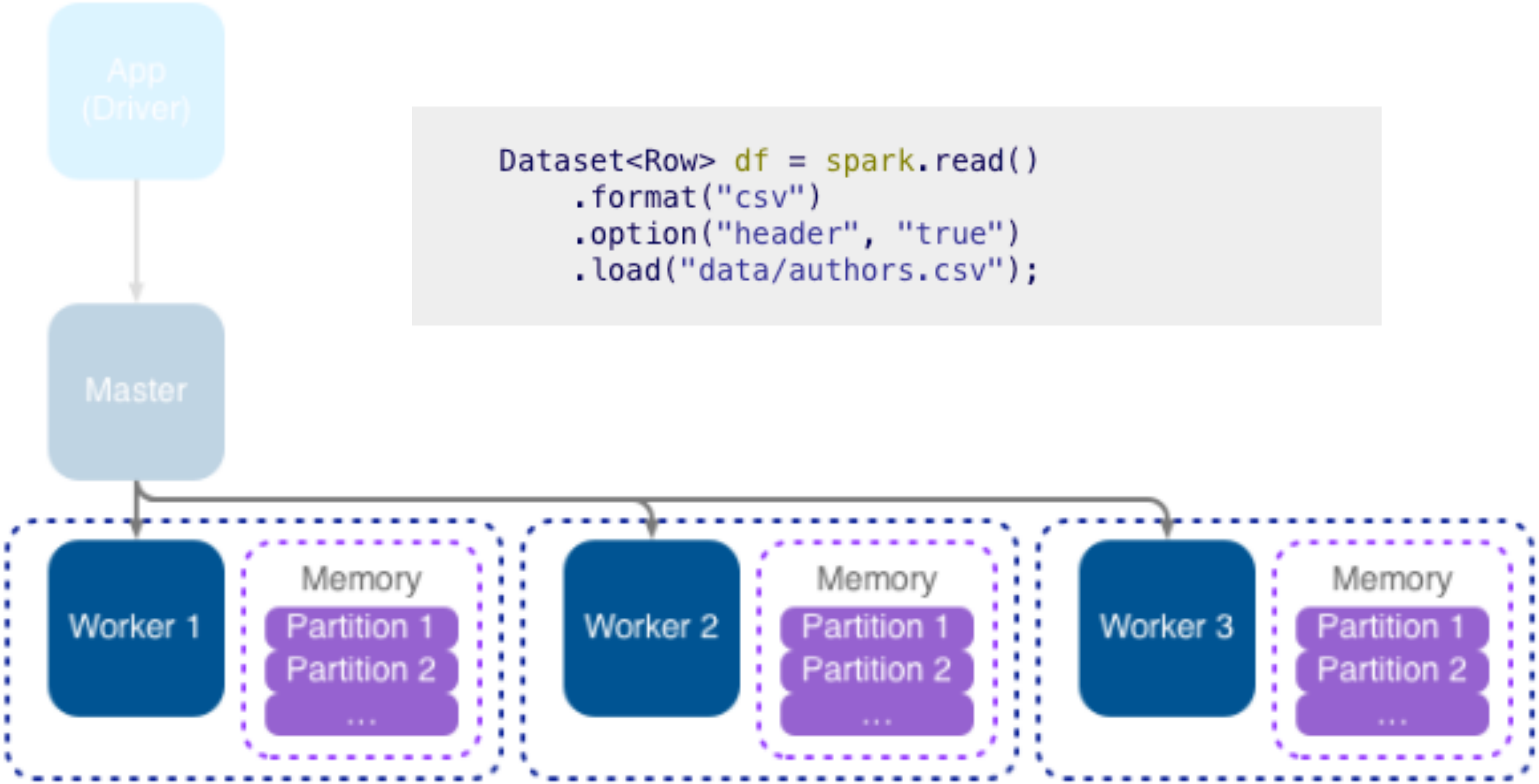


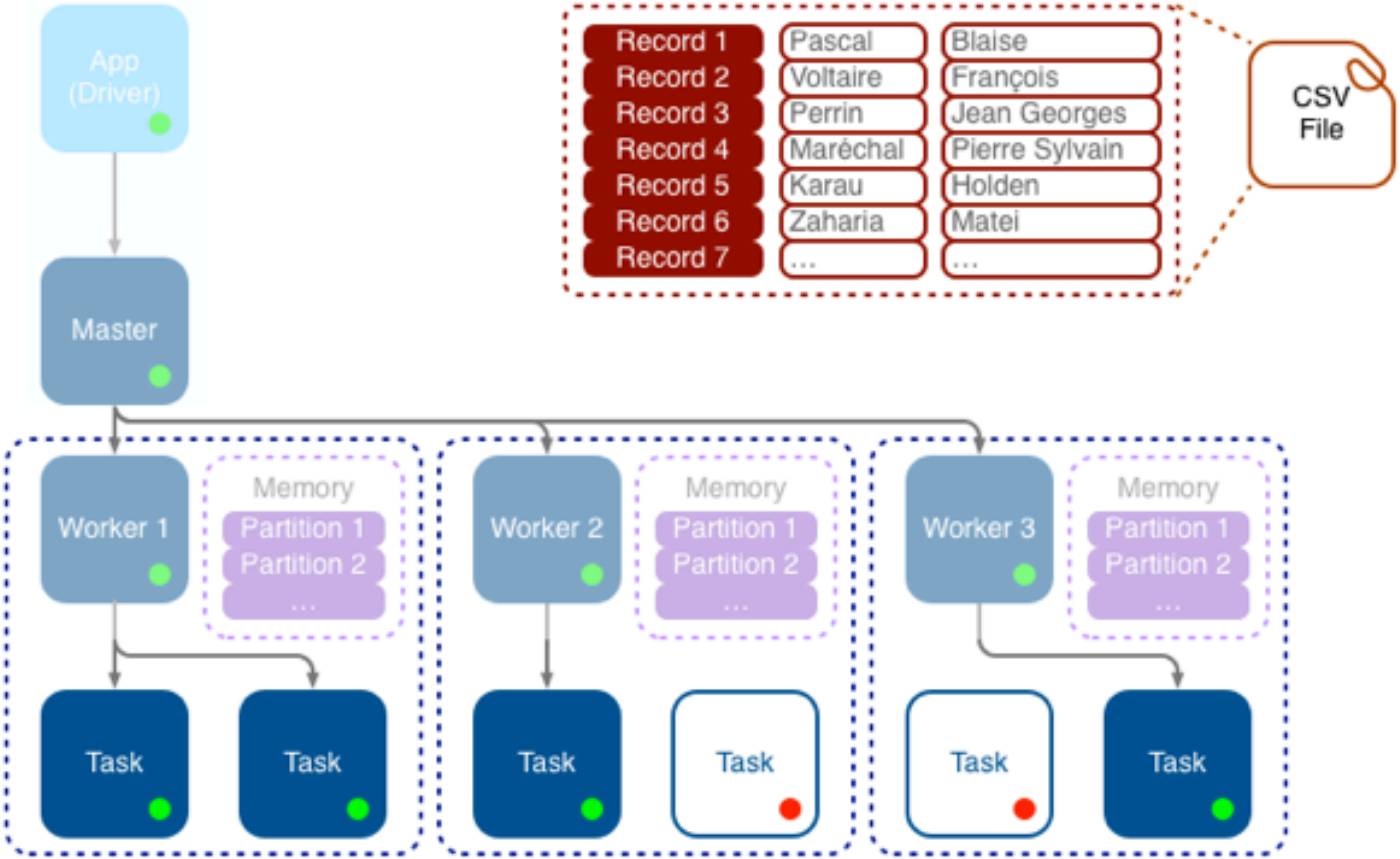


End To End Flow of Data in Spark

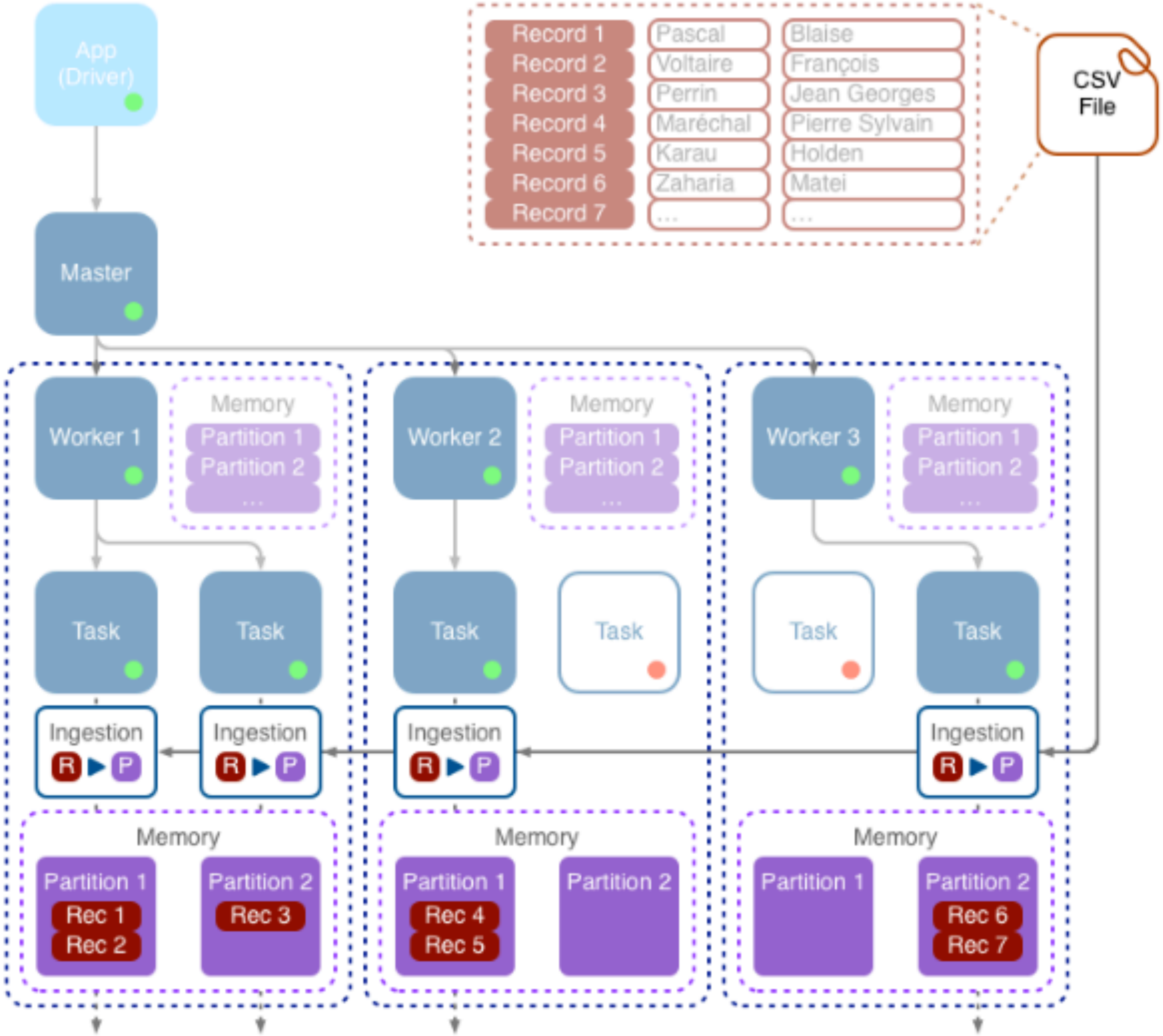

```
SparkSession spark = SparkSession.builder()
      .appName("CSV to DB")
      .master("local")
      .getOrCreate();
```

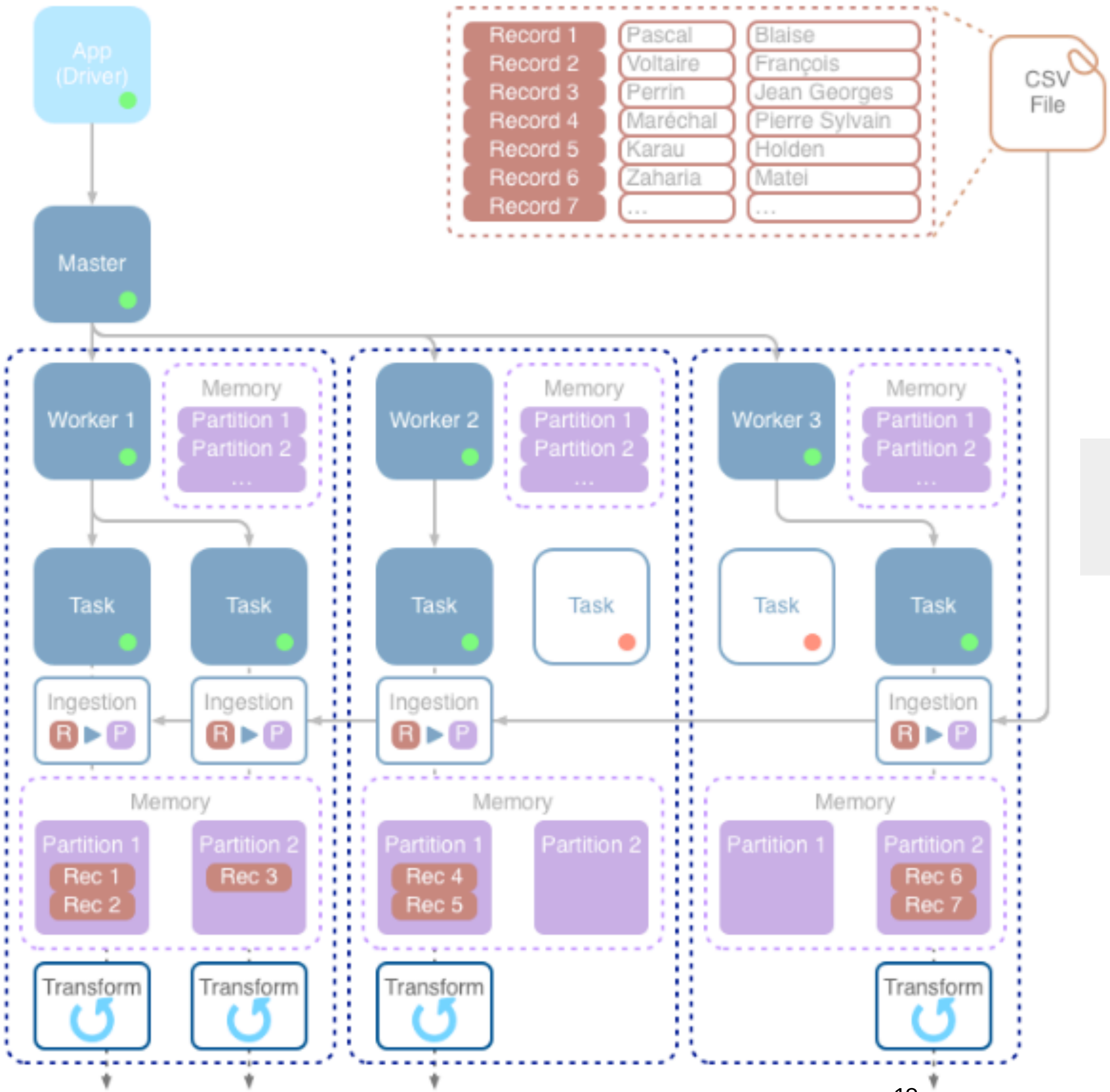
```java
Dataset<Row> df = spark.read()
    .format("csv")
    .option("header", "true")
    .load("data/authors.csv");
```



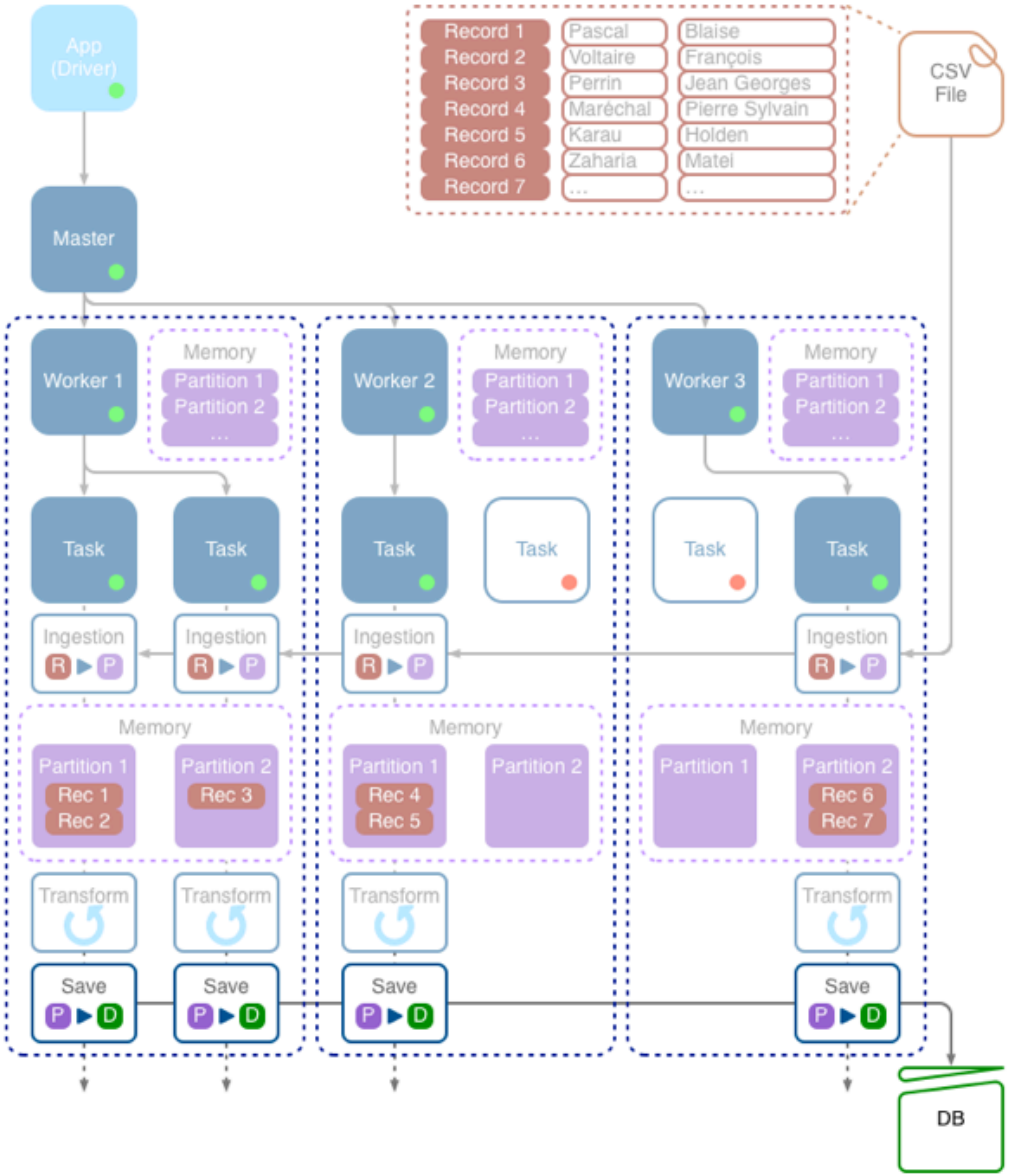


App (Driver)
Master
Record 1
Pascal
Blaise
Record 2
Voltaire
François
Record 3
Perrin
Jean Georges
Record 4
Maréchal
Pierre Sylvain
Record 5
Karau
Holden
Record 6
Zaharia
Matei
Record 7
…
…
CSV File
Worker 1
Memory
Partition 1
Partition 2
…
Task
Task
Worker 2
Memory
Partition 1
Partition 2
…
Task
Task
Worker 3
Memory
Partition 1
Partition 2
…
Task
Task


App (Driver)
Master
Record 1
Record 2
Record 3
Record 4
Record 5
Record 6
Record 7
Pascal
Voltaire
Perrin
Maréchal
Karau
Zaharia
...
Blaise
François
Jean Georges
Pierre Sylvain
Holden
Matei
...
CSV File
Worker 1
Worker 2
Worker 3
Memory
Partition 1
Partition 2
...
Task
Ingestion
R
P
Rec 1
Rec 2
Rec 3
Rec 4
Rec 5
Rec 6
Rec 7




```
df = df.withColumn(
    "name",
    concat(df.col("lname"), lit(", "), df.col("fname")));
```




```java
String dbConnectionUrl = "jdbc:postgresql://localhost/spark_labs";

Properties prop = new Properties();
prop.setProperty("driver", "org.postgresql.Driver");
prop.setProperty("user", "jgp");
prop.setProperty("password", "Spark<3Java");

df.write()
    .mode(SaveMode.Overwrite)
    .jdbc(dbConnectionUrl, "ch02", prop);
```

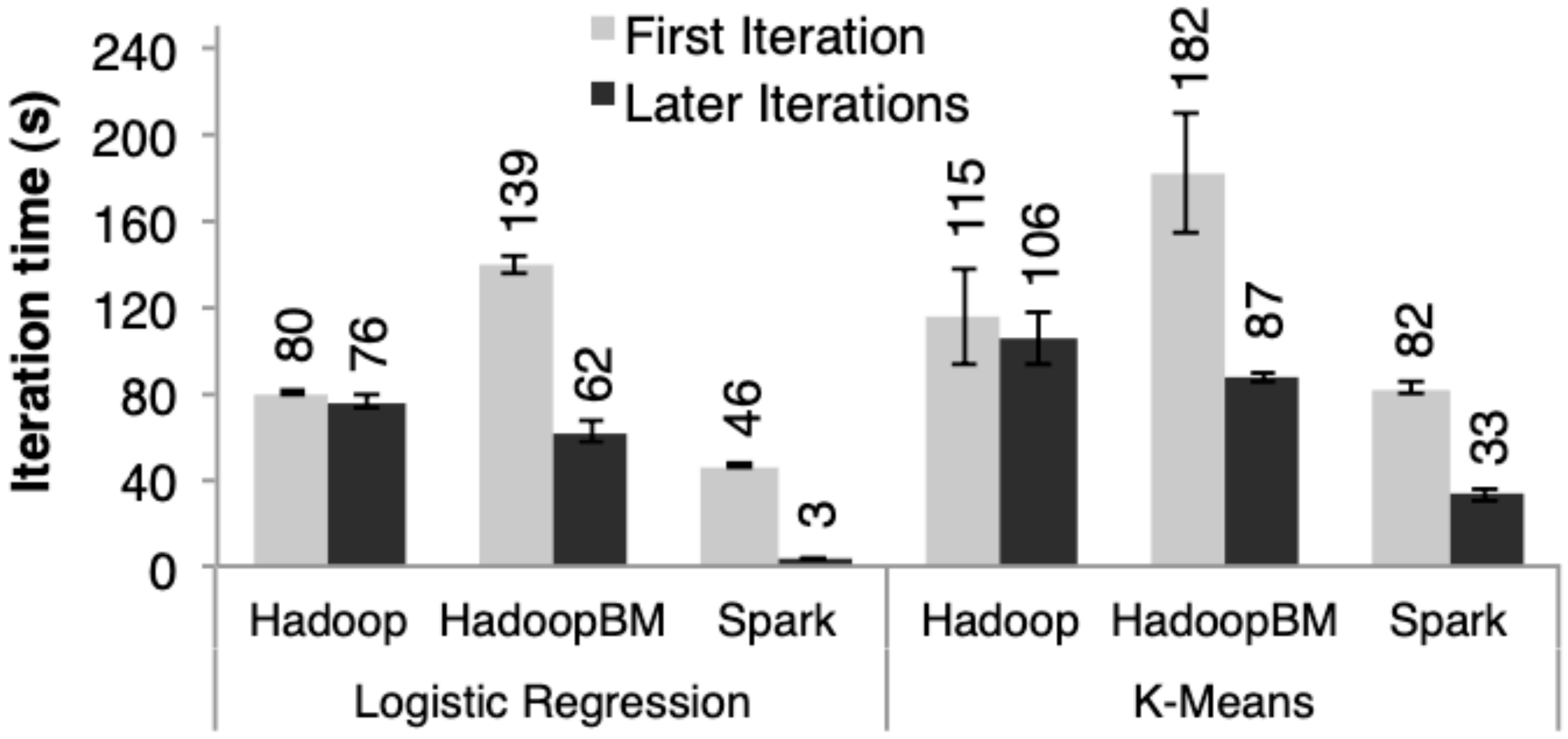
Evaluation

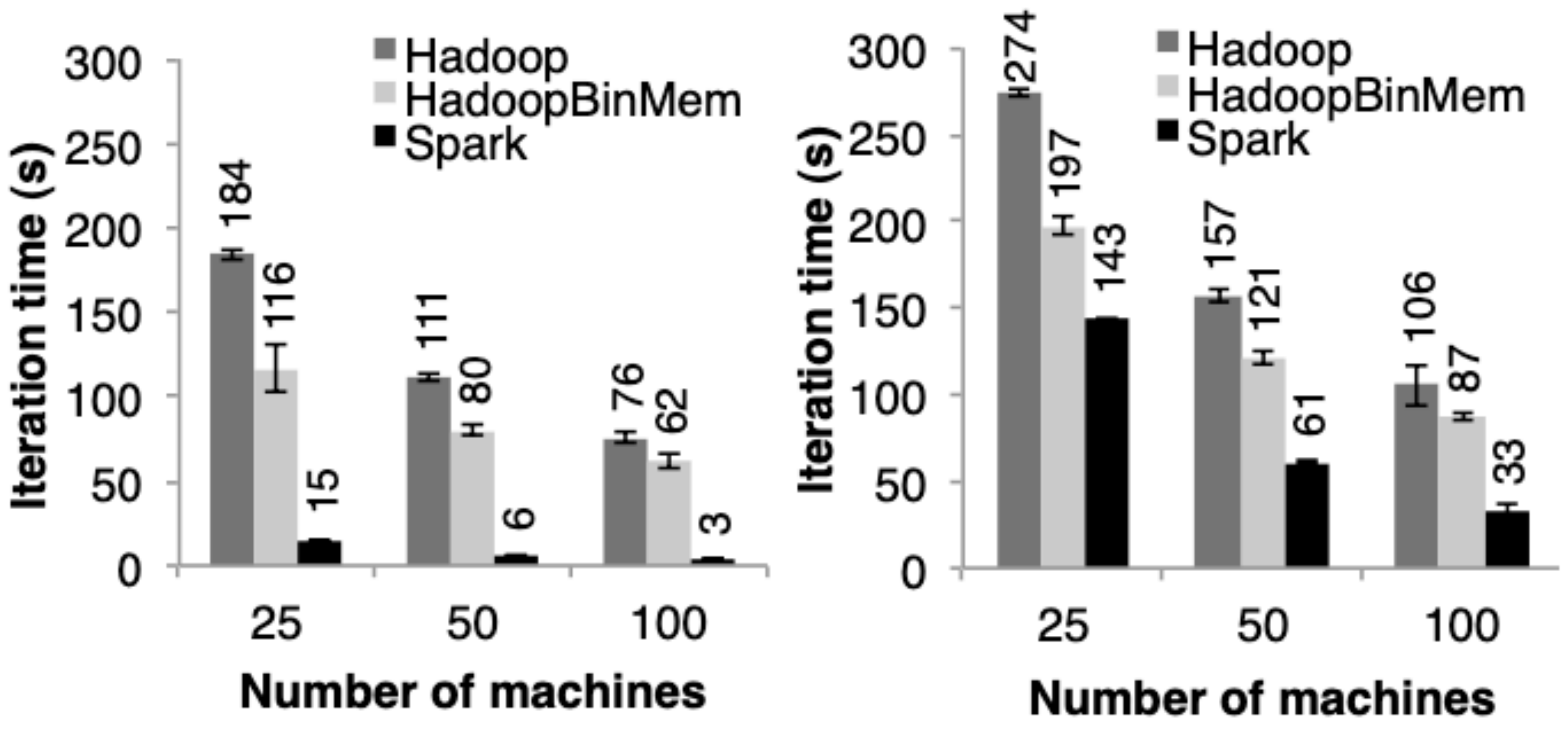
Iterative Machine Learning Applications

- Implemented *logistic regression* and *k-means* to compare performance of *Hadoop, HadoopBinMem and Spark*
- Both algorithms were run for 10 iterations on 100GB datasets using 25-100 machines.
- Key difference between logistic regression and k-means is the amount of computation performed per byte of data
- Logistic regression is less compute intensive compared to k-means and requires more time in deserialization and I/O

Evaluation (contd..)

Iterative Machine Learning Applications



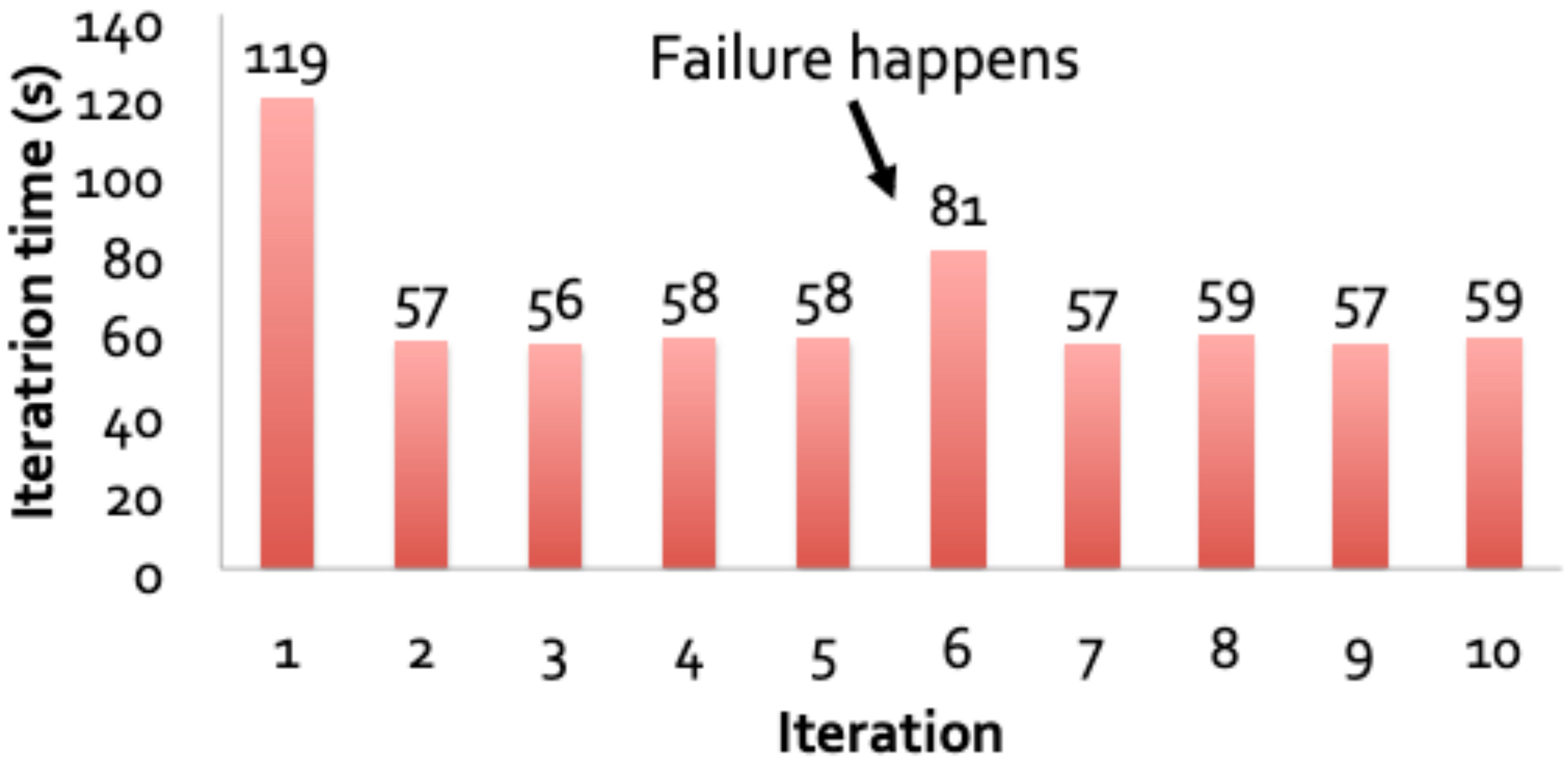


(a) Logistic Regression (b) K-Means

Evaluation (Contd..)

Fault Recovery for K-means application

- Compares running times for 10 iterations of k-means on a 75 node cluster
- 400 tasks working on 100GB data per iteration
- One node fails in the 6th iteration
- RDD was re-created with input data and lineage
- Average time resumes to be 58s from the next iteration



Evaluation (Contd..)

Page Rank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

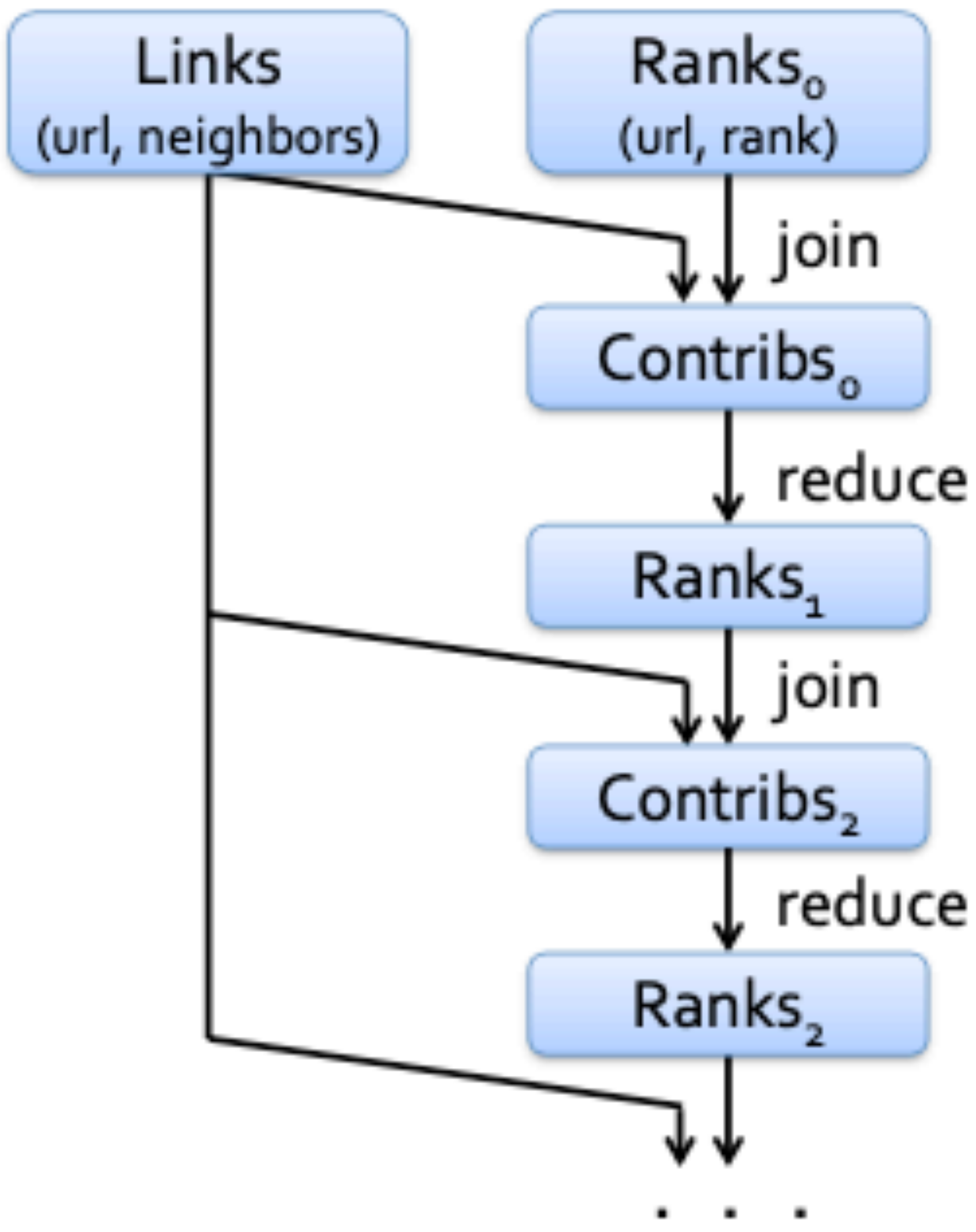
$$\Sigma_{i \in neighbors} \, rank_i \, / \, |neighbors_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```

Evaluation (Contd..)

Page Rank



`links` & `ranks` repeatedly joined

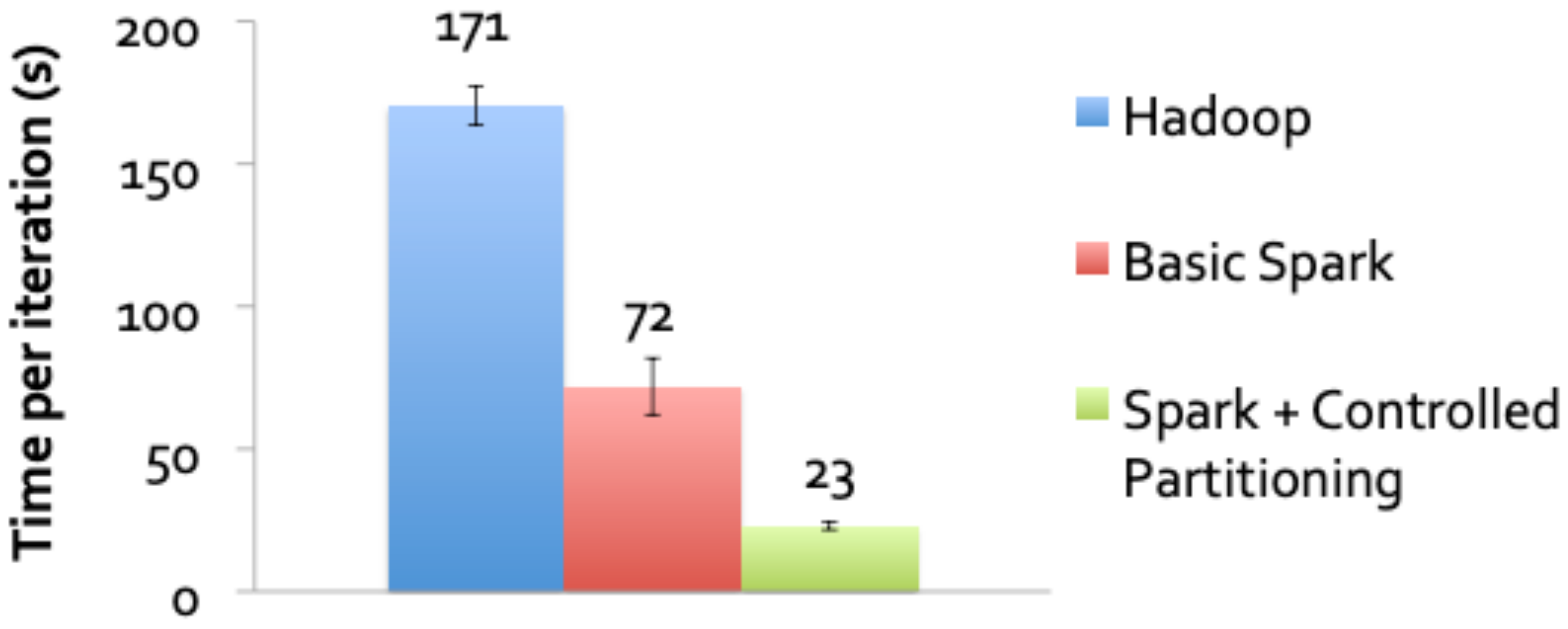
Can *co-partition* them (e.g. hash both on URL) to avoid shuffles

Can also use app knowledge, e.g., hash on DNS name

```
links = links.partitionBy(
        new URLPartitioner())
```

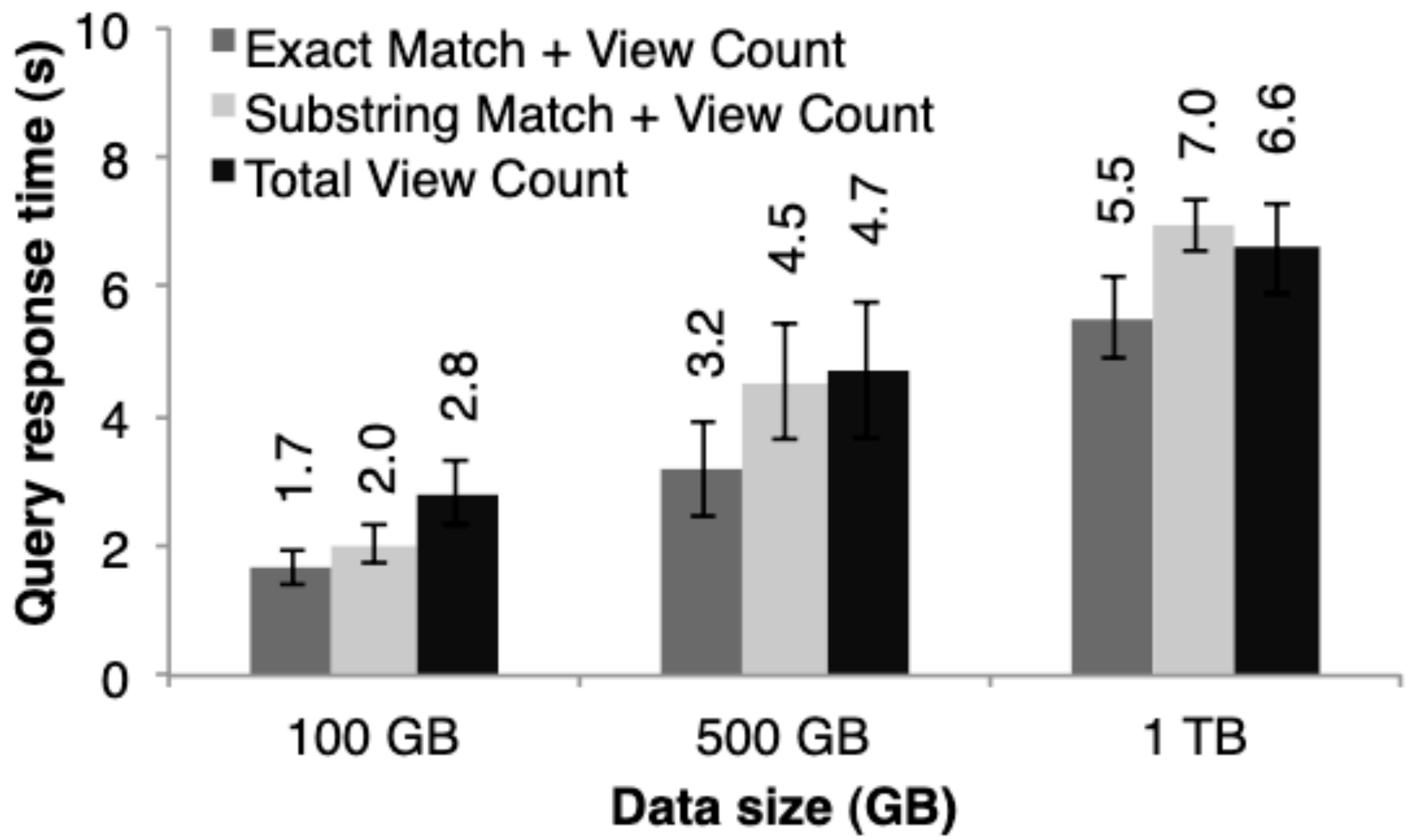
Evaluation (Contd..)

Page Rank



Evaluation (Contd..)
Interactive Data Mining

- Analyze 1TB of Wikipedia page view logs(~2years of data)
- Used 100 large EC2 instances with 8cores and 68GB RAM each
- Ran queries to find total view of:
  - all pages
  - pages with titles exactly matching a given word
  - pages with titles partially matching a word



Querying from disk for 1TB data: 170s 😐

Conclusion

- RDDs offer a simple and efficient programming model for a broad range of applications
- They are particularly useful for batch processing where they leverage the coarse-grained nature of many parallel transformations for low-overhead recovery
- Some existing programming models expressible using RDDs:
  - MapReduce
  - DryadLINQ
  - SQL
  - Pregel
  - Batch Stream Processing
  - Iterative MapReduce

References

- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing *by* Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica *UC, Berkeley*
- Blog about DAG: https://data-flair.training/blogs/dag-in-apache-spark/
- Medium Article on RDD basics: https://medium.com/@goyalsaurabh66/spark-basics-rdds-stages-tasks-and-dag-8da0f52f0454
- Official Spark Documentation: https://spark.apache.org/docs/latest/rdd-programming-guide.html

Thank You