Introduction to Spark

Dr. Rajiv Misra
Associate Professor
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 some of its applications such as: **Page rank** and **GraphX.**

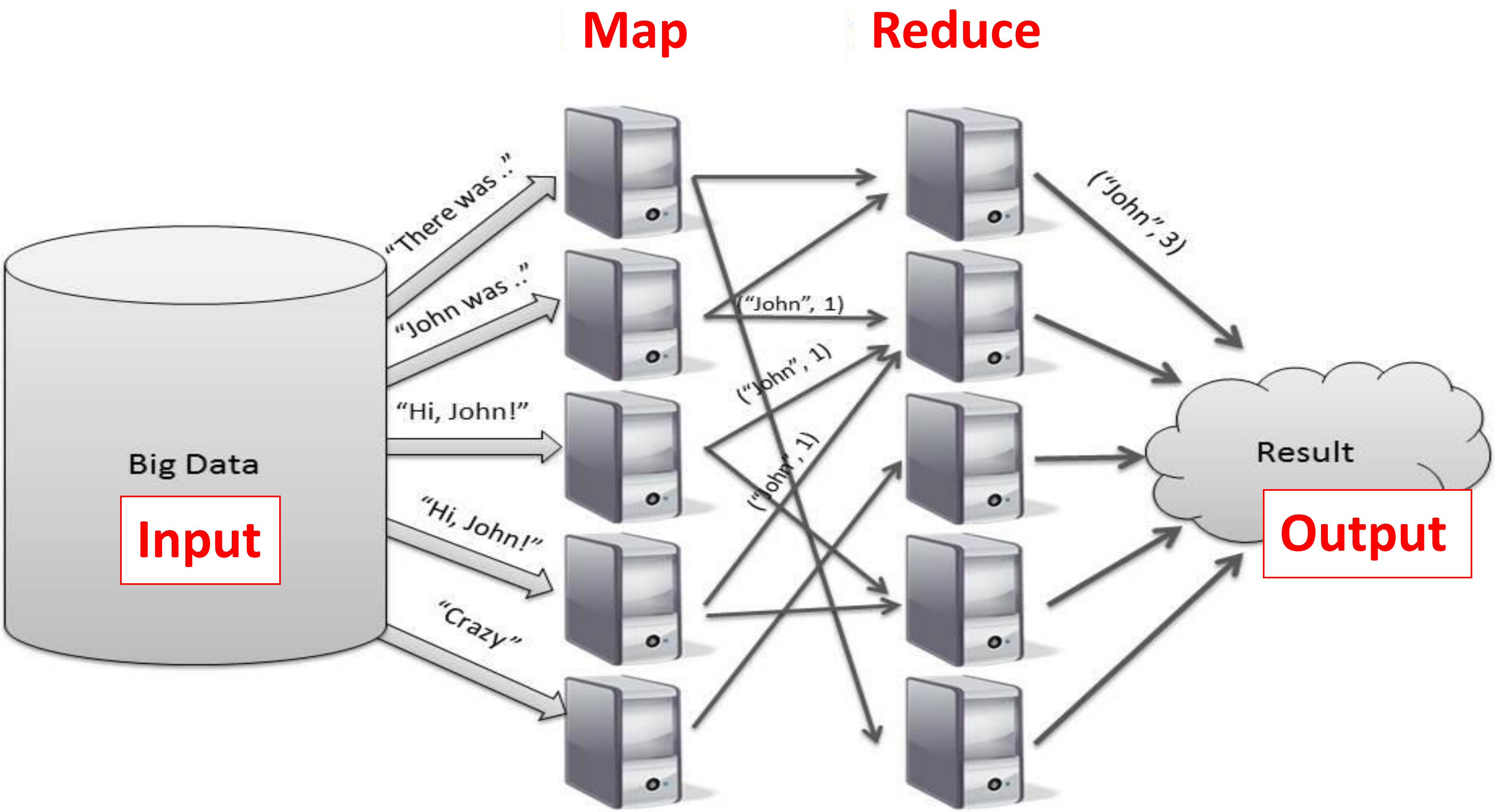
Need of Spark

- **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.

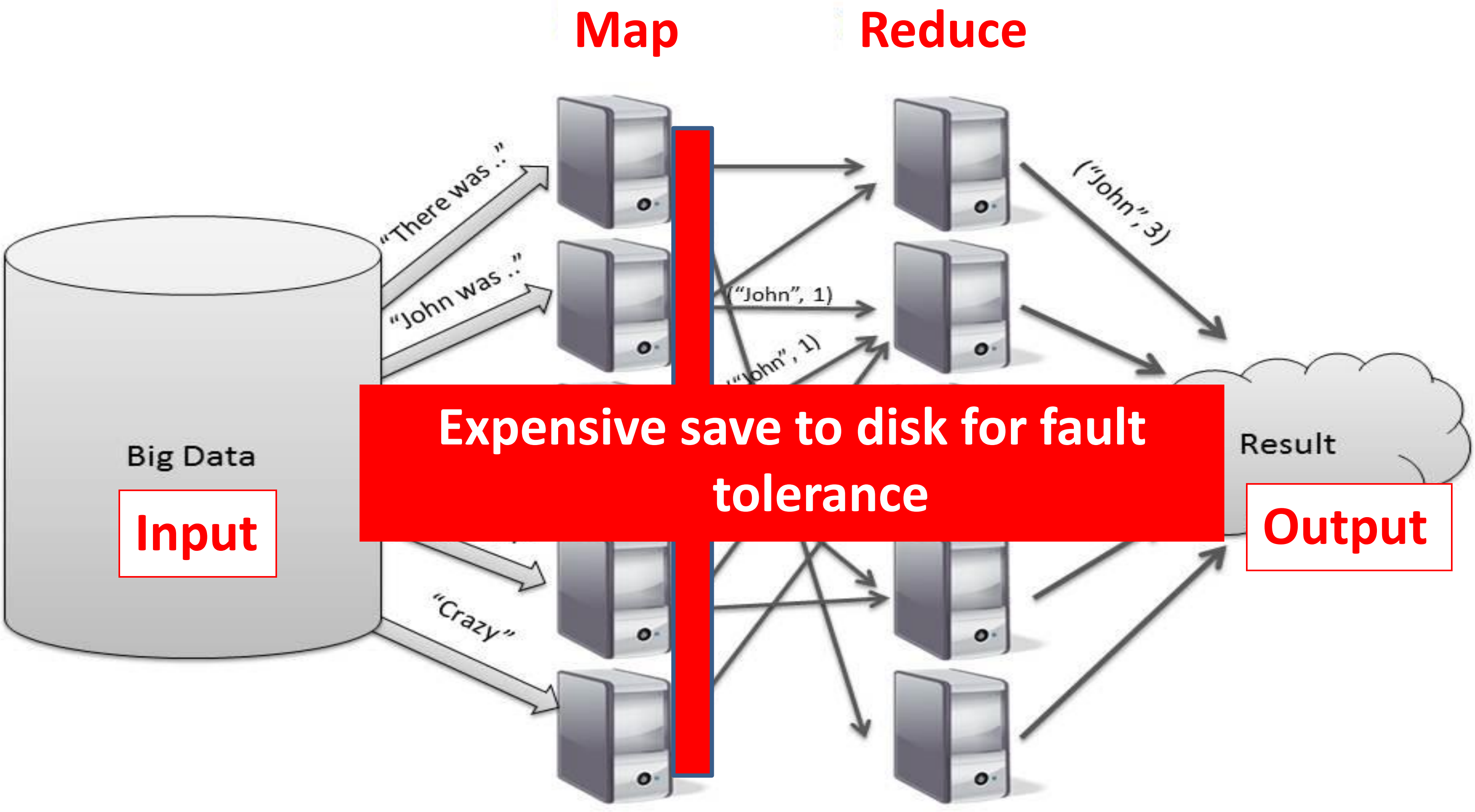
- It is an another system for big data analytics

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

Need of Spark



Need of Spark



Need of Spark

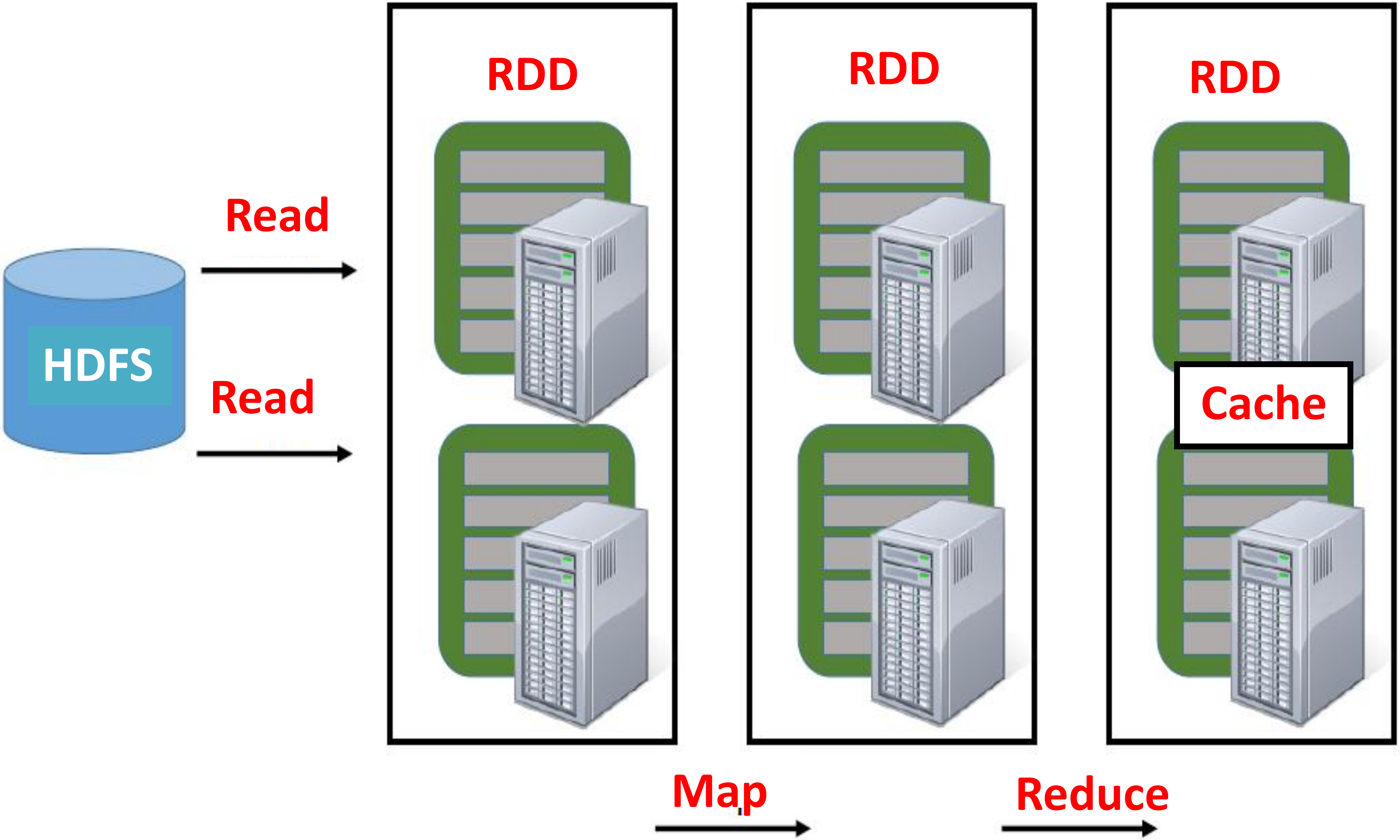
- MapReduce can be expensive for some applications e.g.,
    - **Iterative**
    - **Interactive**
- Lacks efficient data sharing
- Specialized frameworks did evolve for different programming models
    - **Bulk Synchronous Processing (Pregel)**
    - **Iterative MapReduce (Hadoop) ….**

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



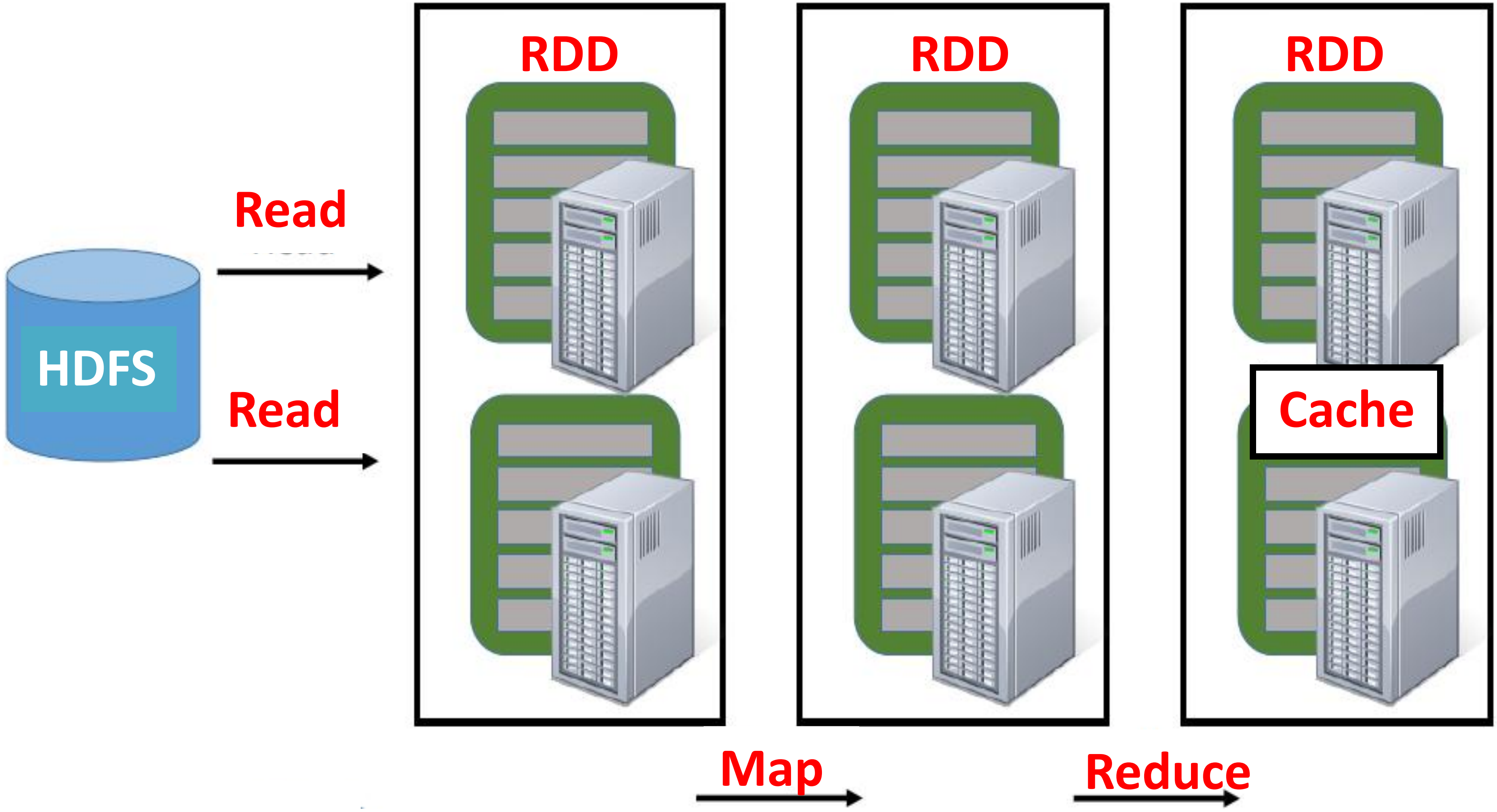
Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

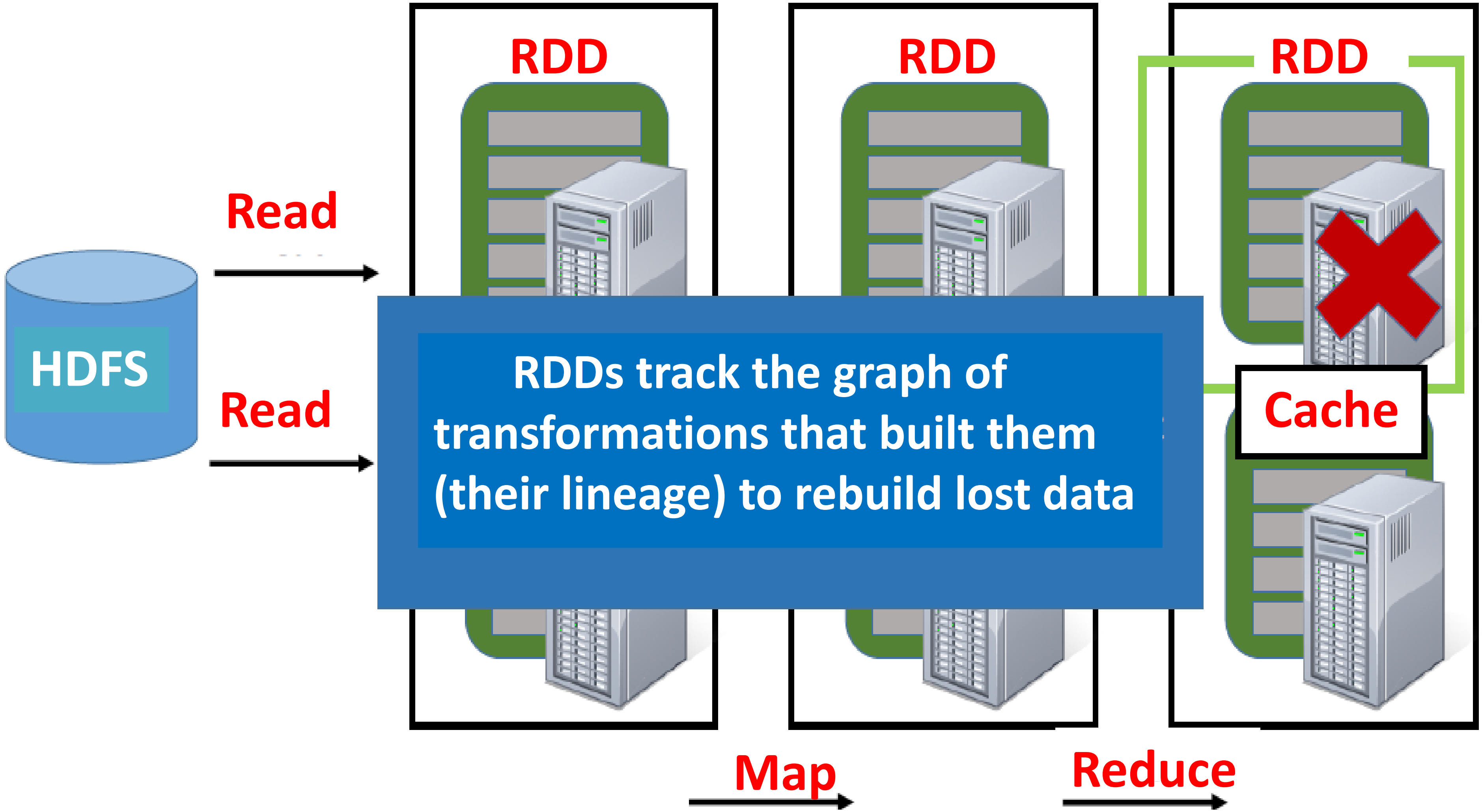
- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)

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
HDFS
Read
Cache
Map
Reduce


Lineage
HDFS
Read
RDD
Map
RDD
Reduce
RDD

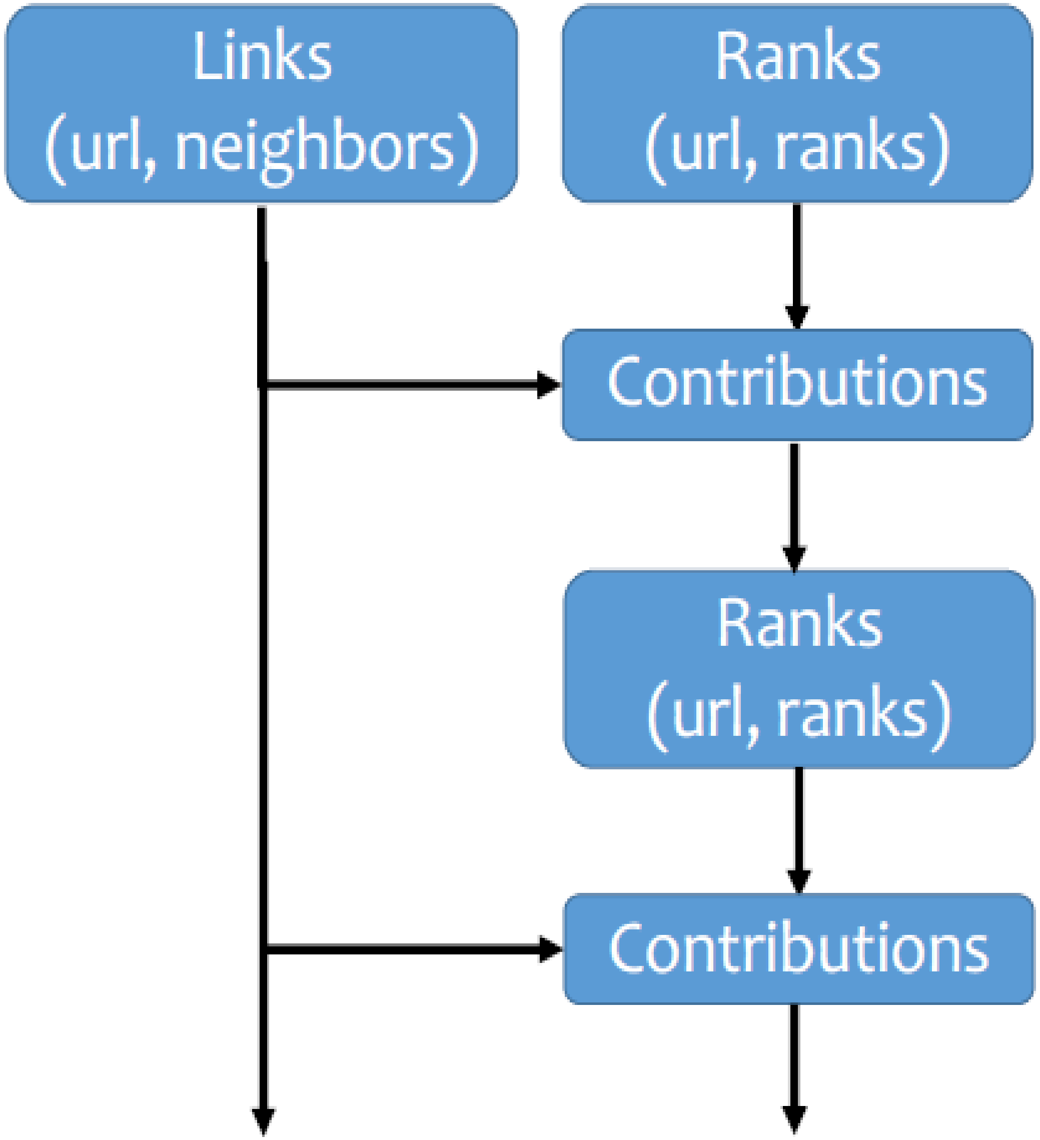

RDD
RDD
RDD
Read
HDFS
Read
RDDs track the graph of transformations that built them (their lineage) to rebuild lost data
Cache
Map
Reduce

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.

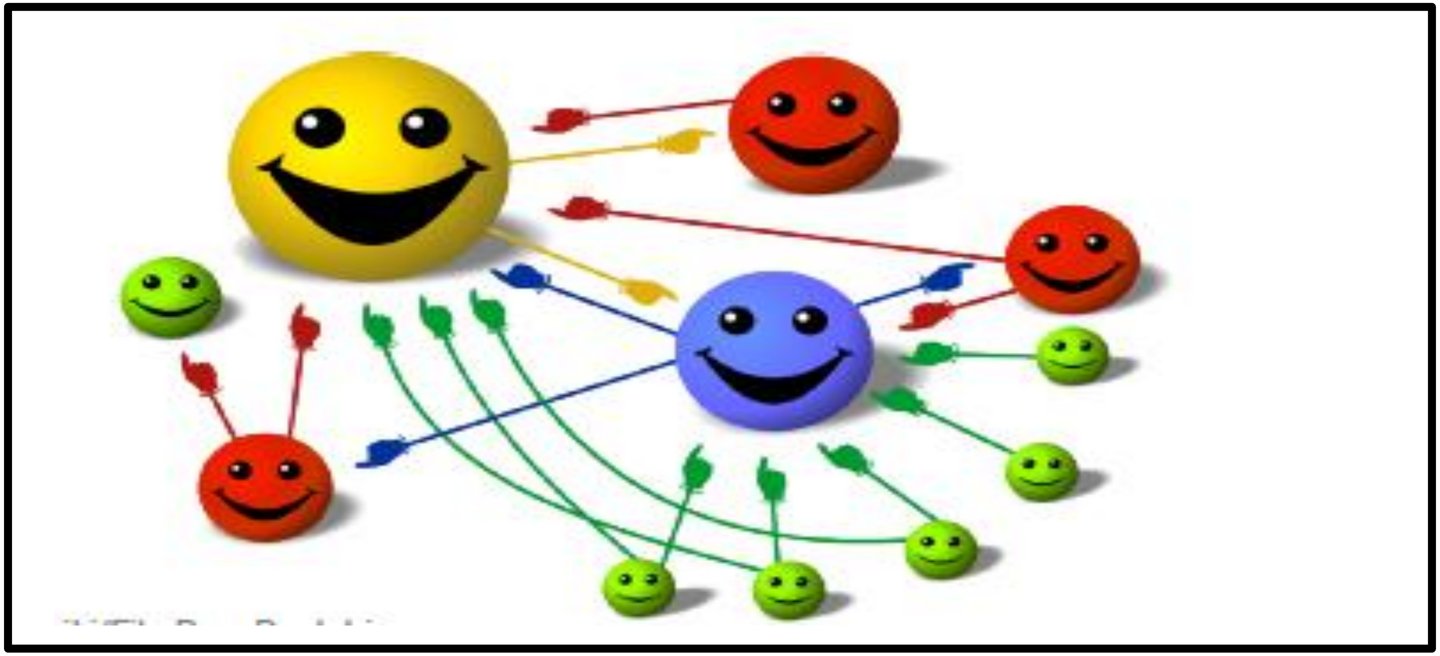
Partitioning: PageRank



- Joins take place repeatedly
- Good partitioning reduces shuffles

Example: PageRank

- Give pages ranks (scores) based on links to them
- Links from many pages → high rank
- Links from a high-rank page → high rank

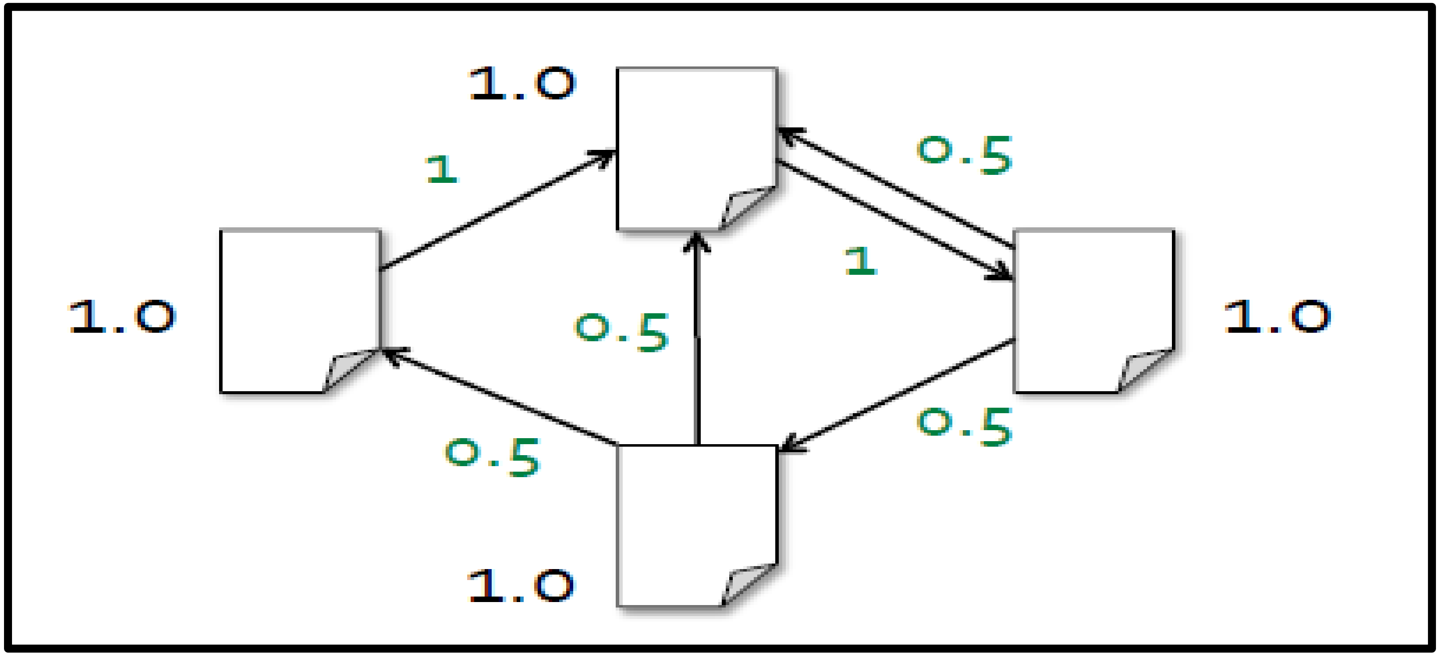


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ |$neighbors_p$|** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

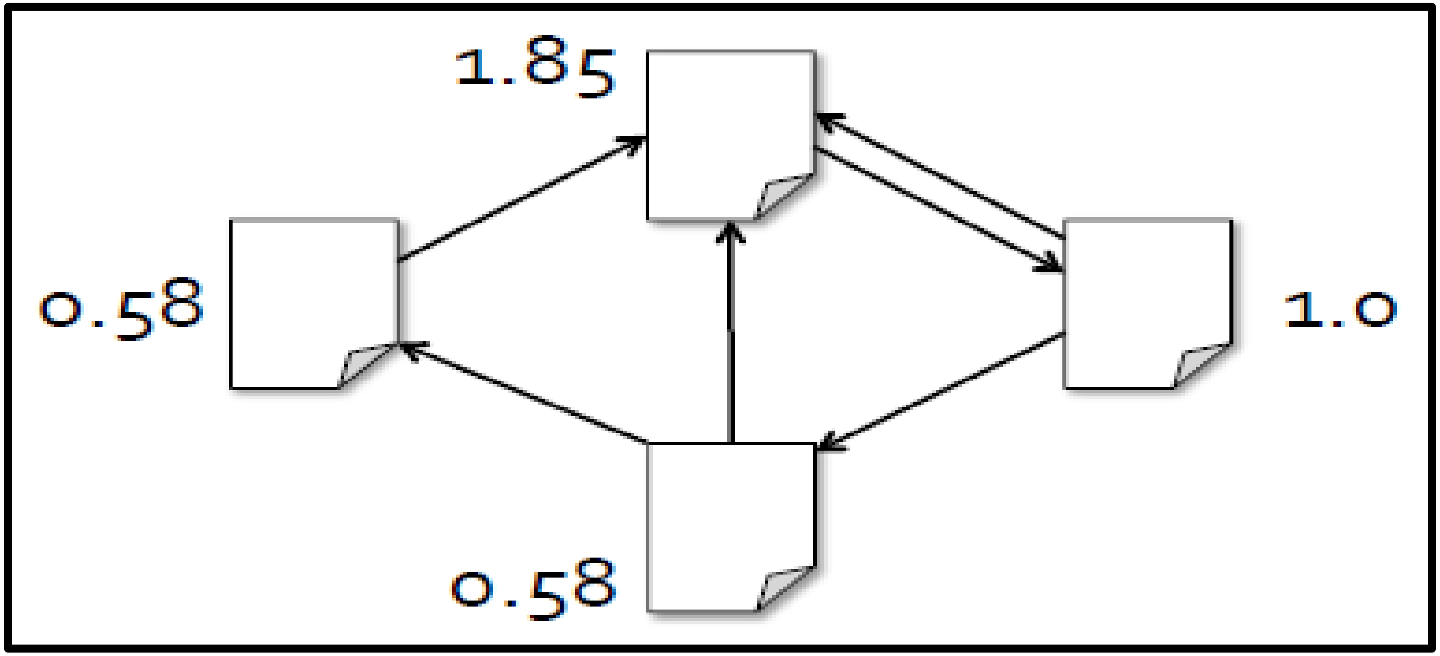


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ |$neighbors_p$|** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

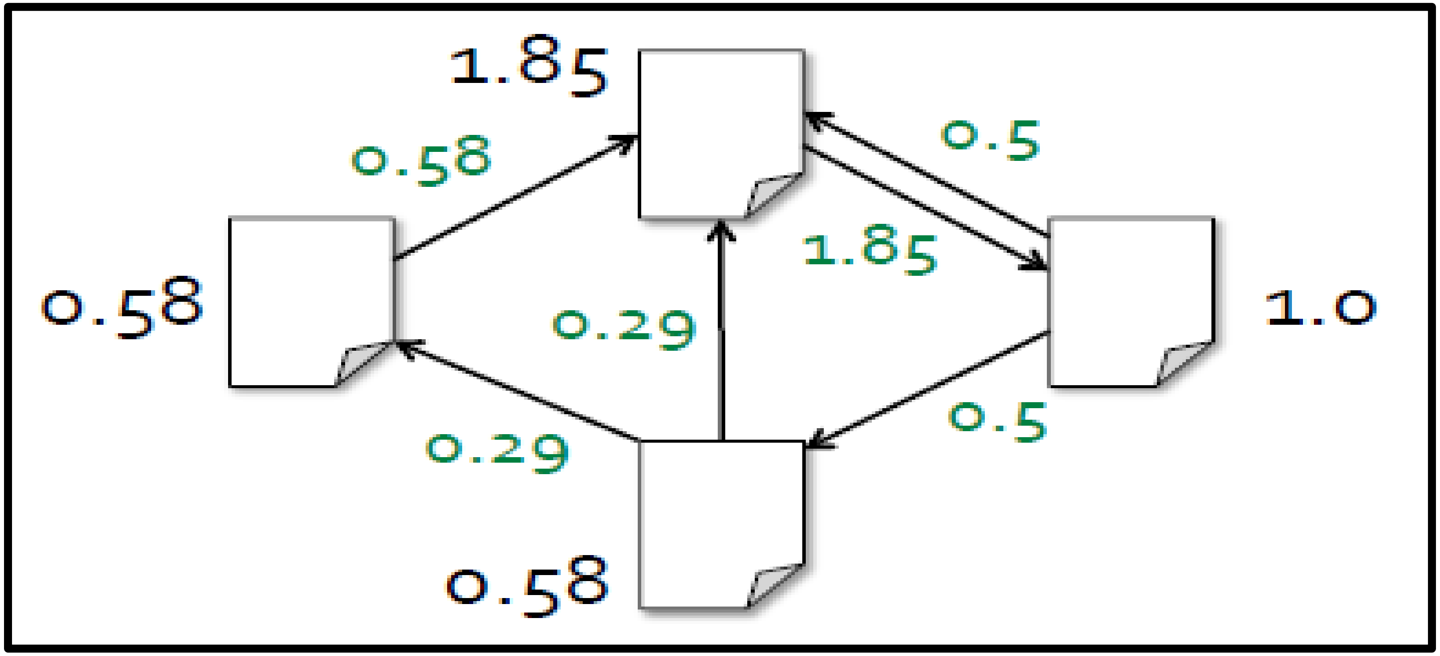


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ |$neighbors_p$|** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

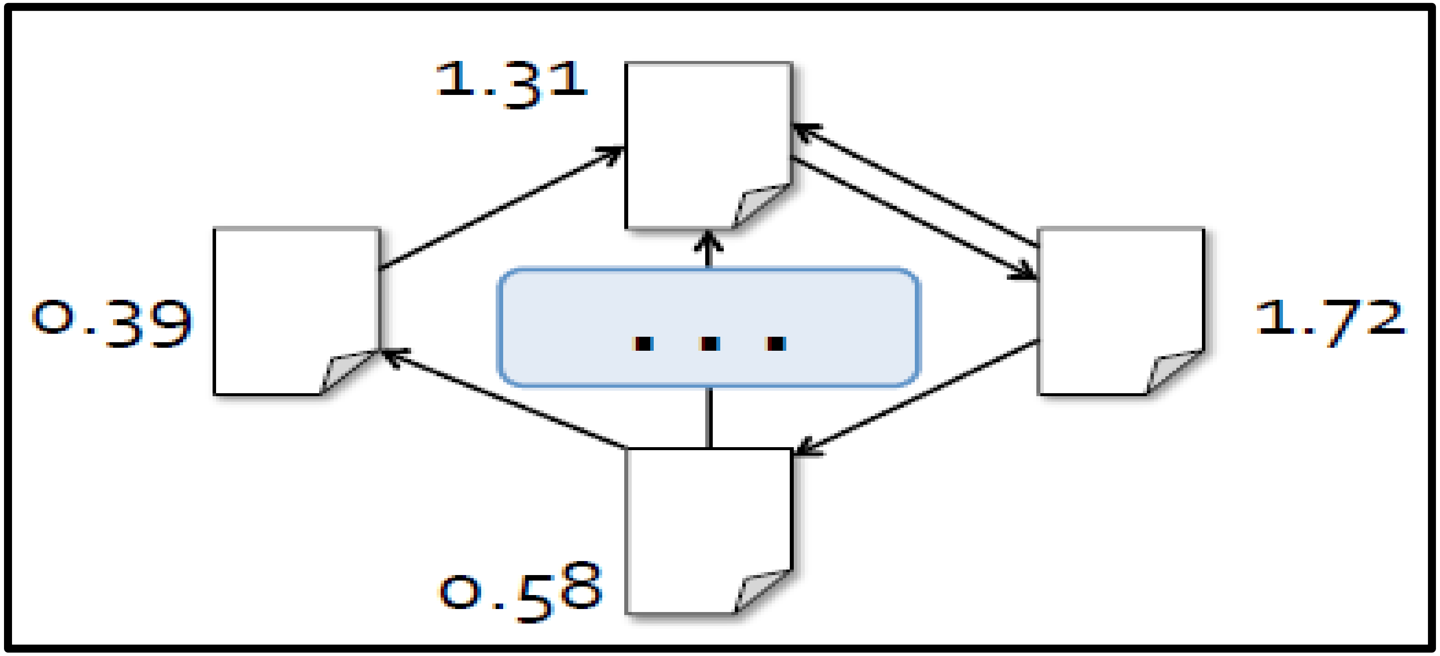


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ $|neighbors_p|$** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

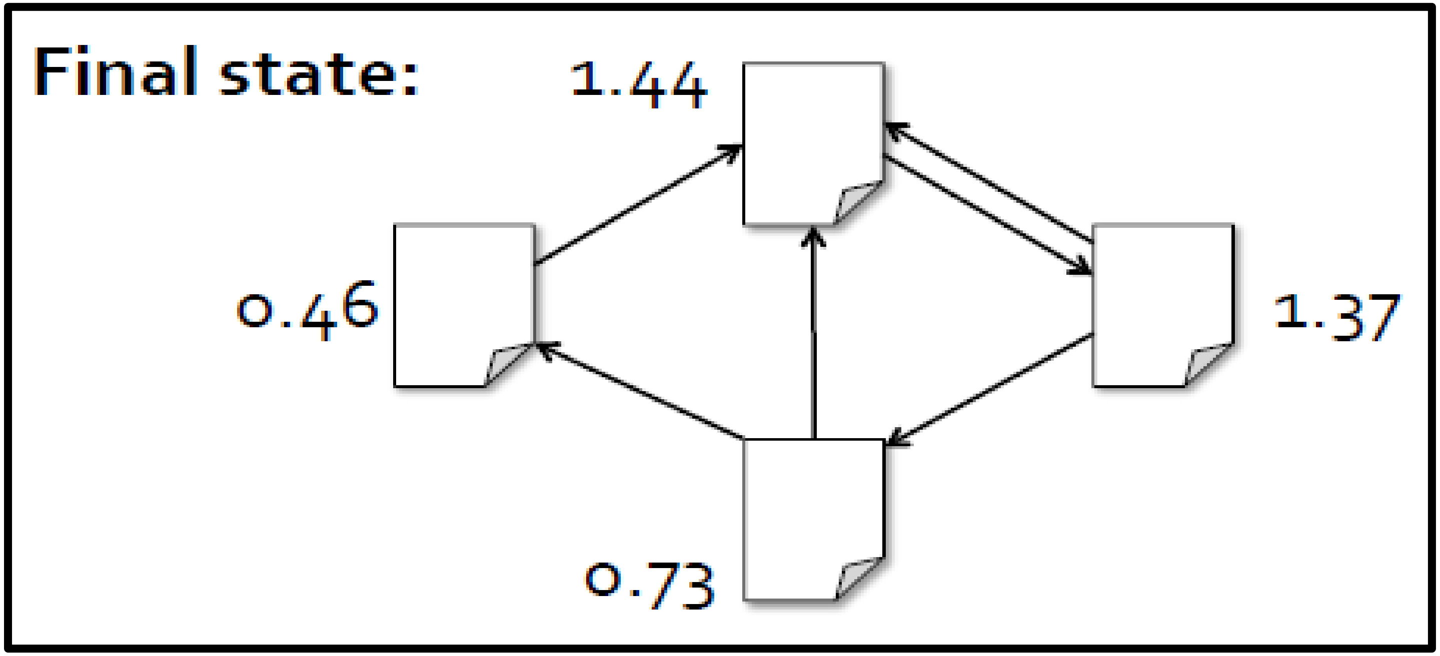


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ $|neighbors_p|$** to its neighbors

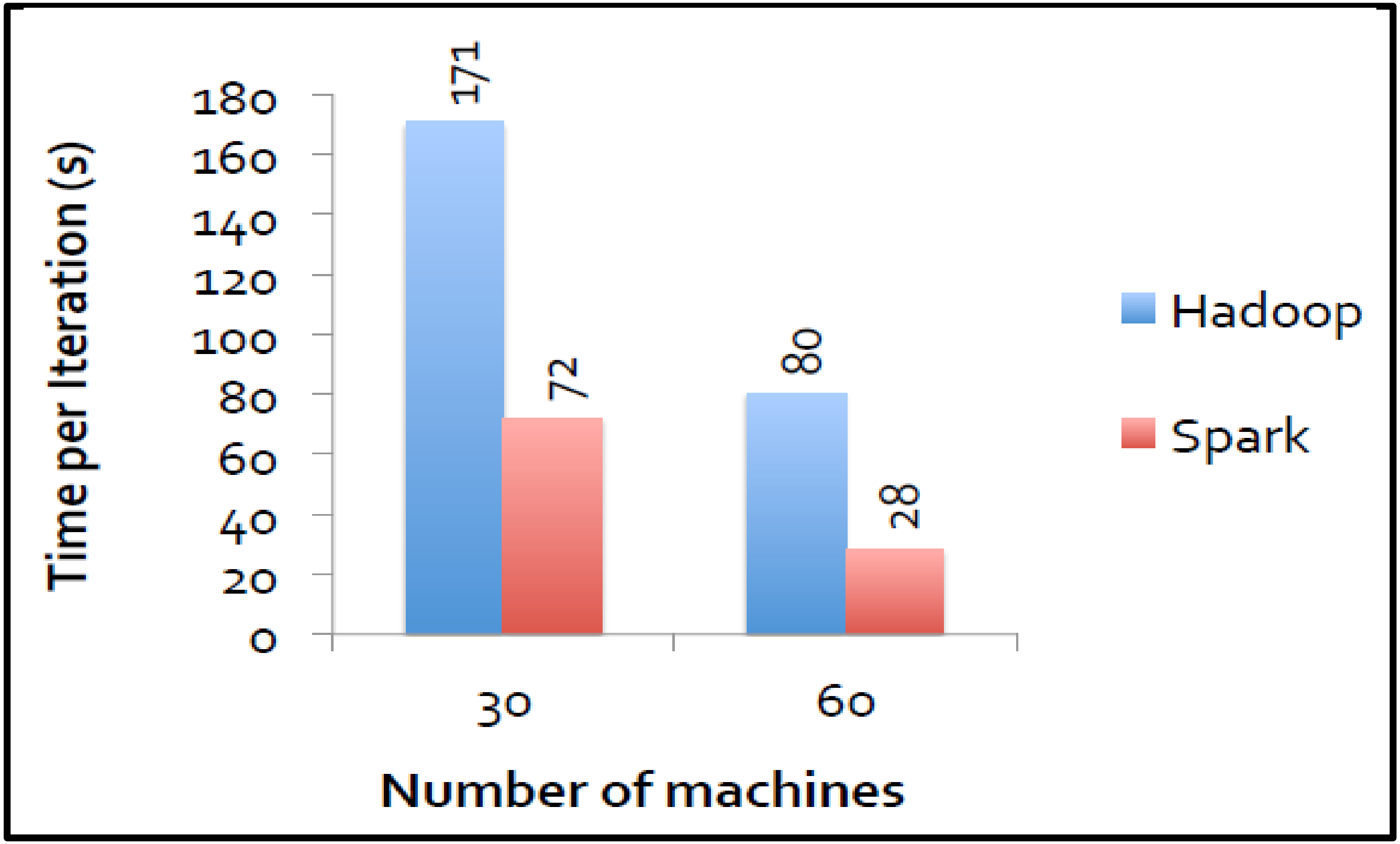
Step-3 Set each page's rank to **0.15 + 0.85 x contributions**



Spark Program

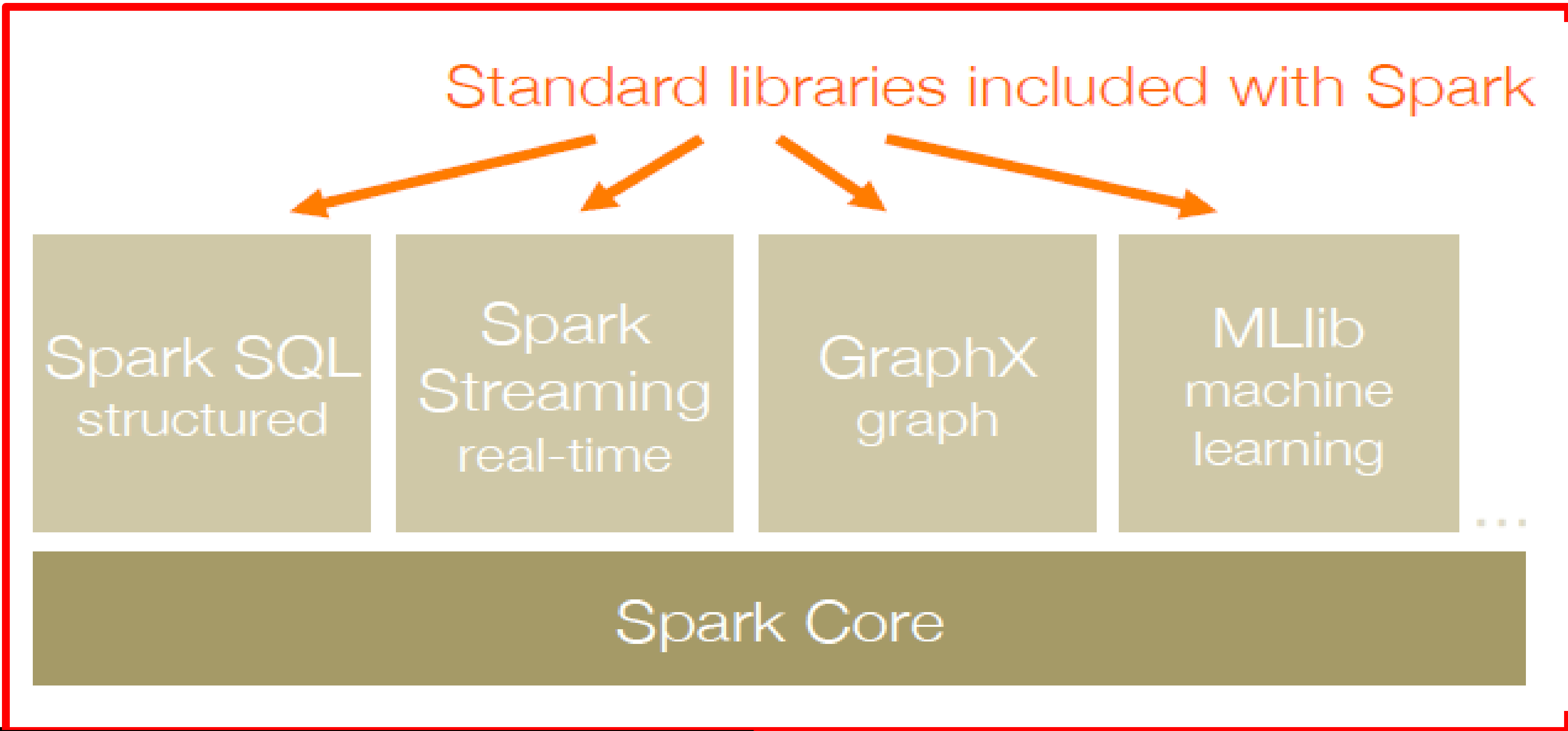
```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs
for (i <- 1 to ITERATIONS) {
    val contribs = links.join(ranks).flatMap {
        case (url, (links, rank)) =>
            links.map(dest => (dest, rank/links.size))
    }
    ranks = contribs.reduceByKey (_ + _)
                        .mapValues (0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

PageRank Performance

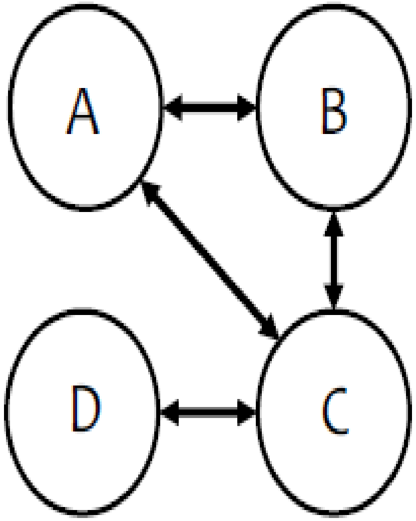


Generality

- RDDs allow unification of different programming models
    - Stream Processing
    - Graph Processing
    - Machine Learning

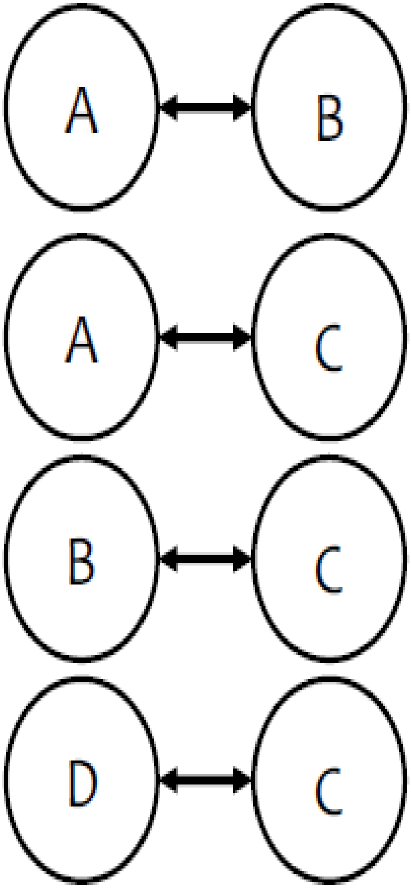


Gather-Apply-Scatter on GraphX



Vertices	Neighbors
A	B
A	C
B	C
D	C

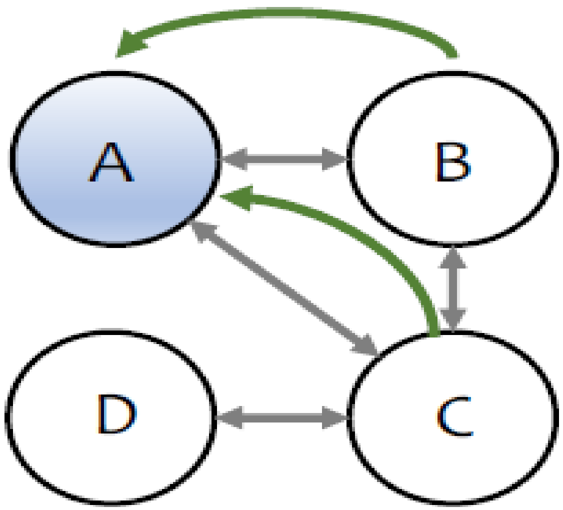
Triplets

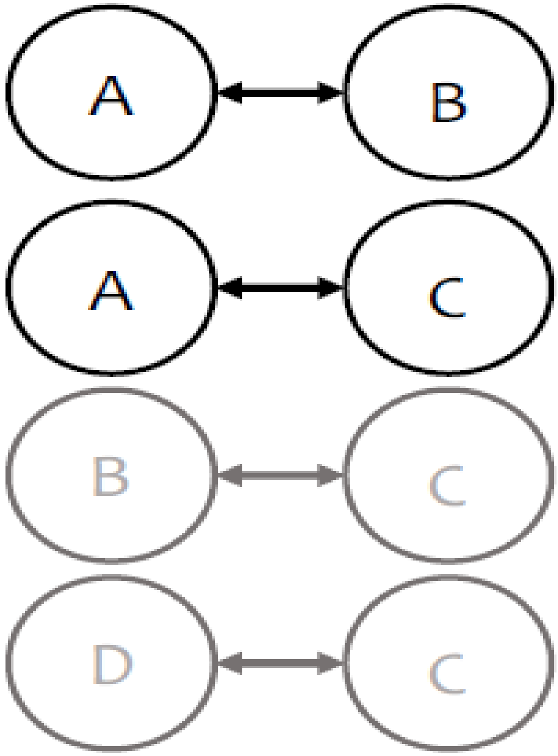


Graph Represented In a Table

Triplets

Gather-Apply-Scatter on GraphX

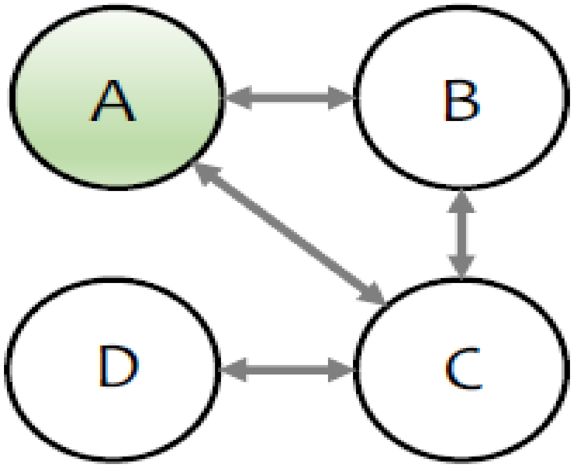




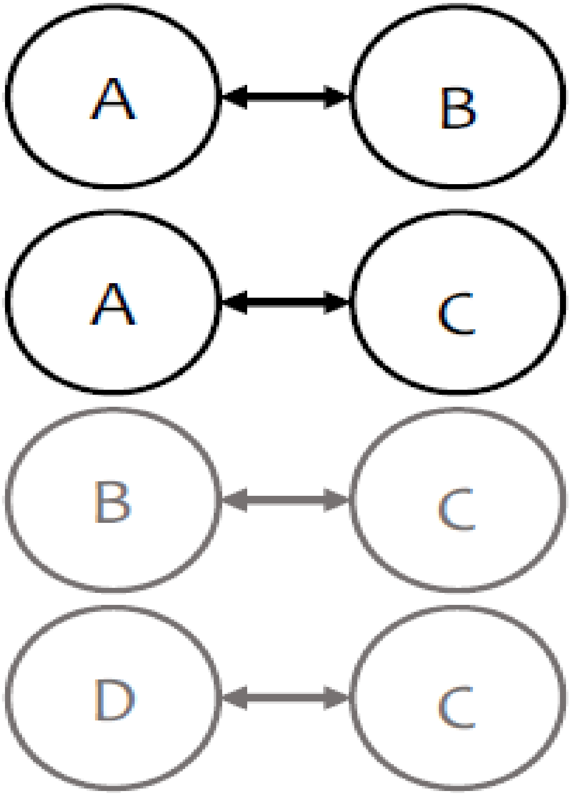
Gather at A

Group-By A

Gather-Apply-Scatter on GraphX

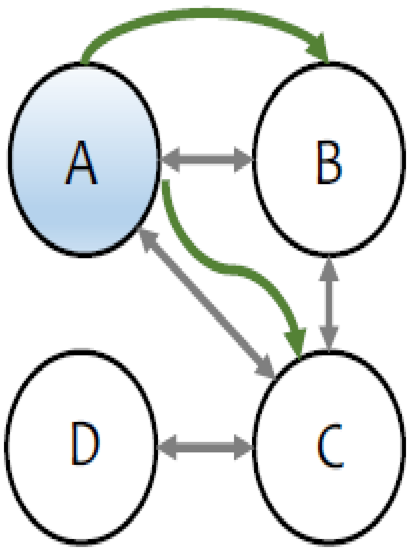


Apply

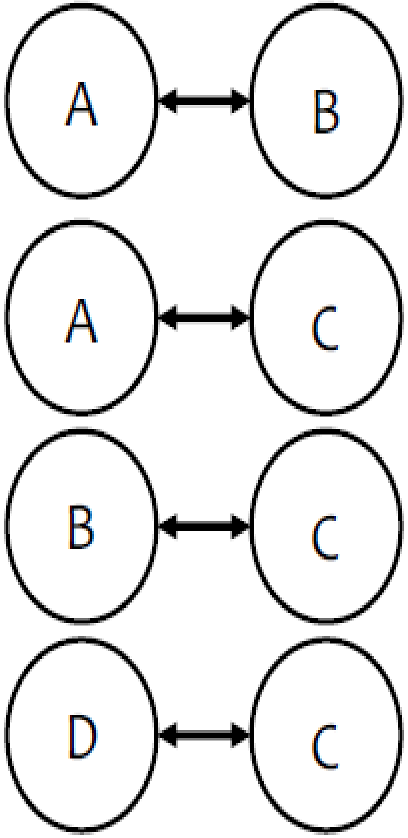


Map

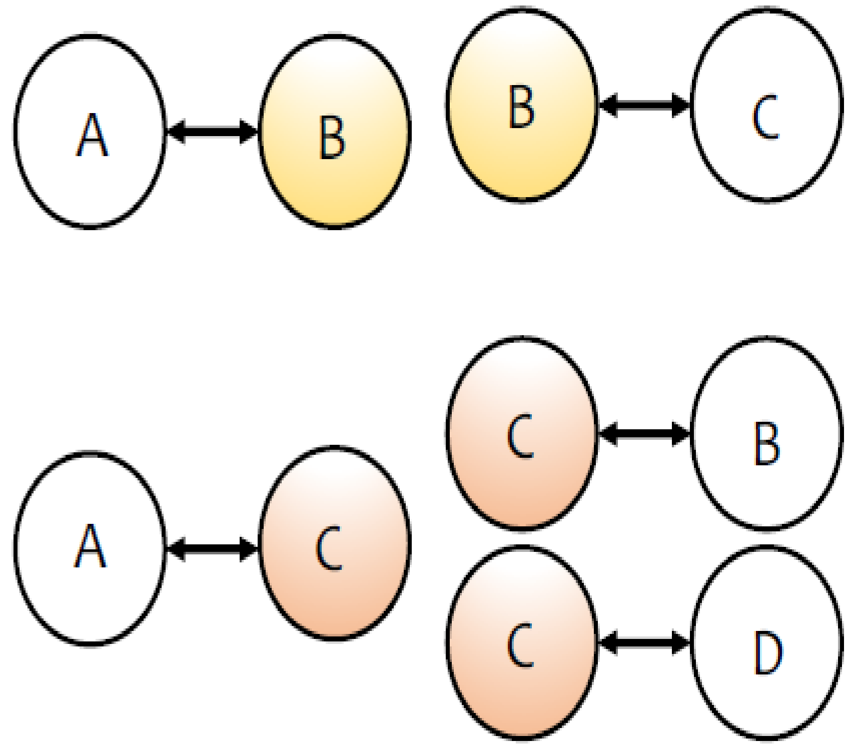
Gather-Apply-Scatter on GraphX



Scatter



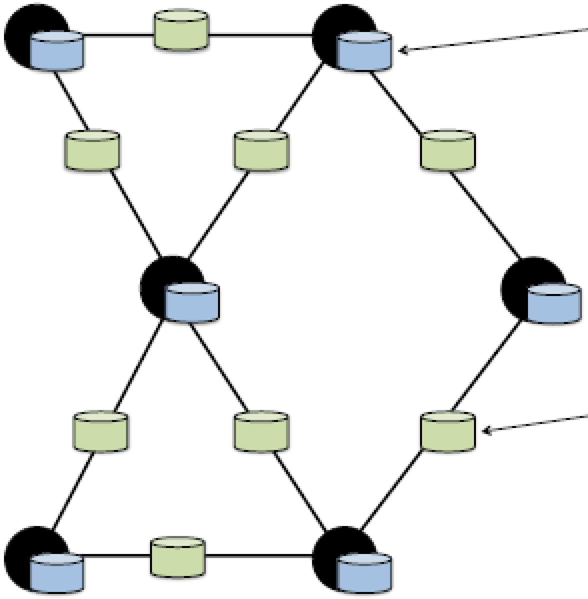
Triplets



Join

The GraphX API

Graphs Property



Vertex Property:

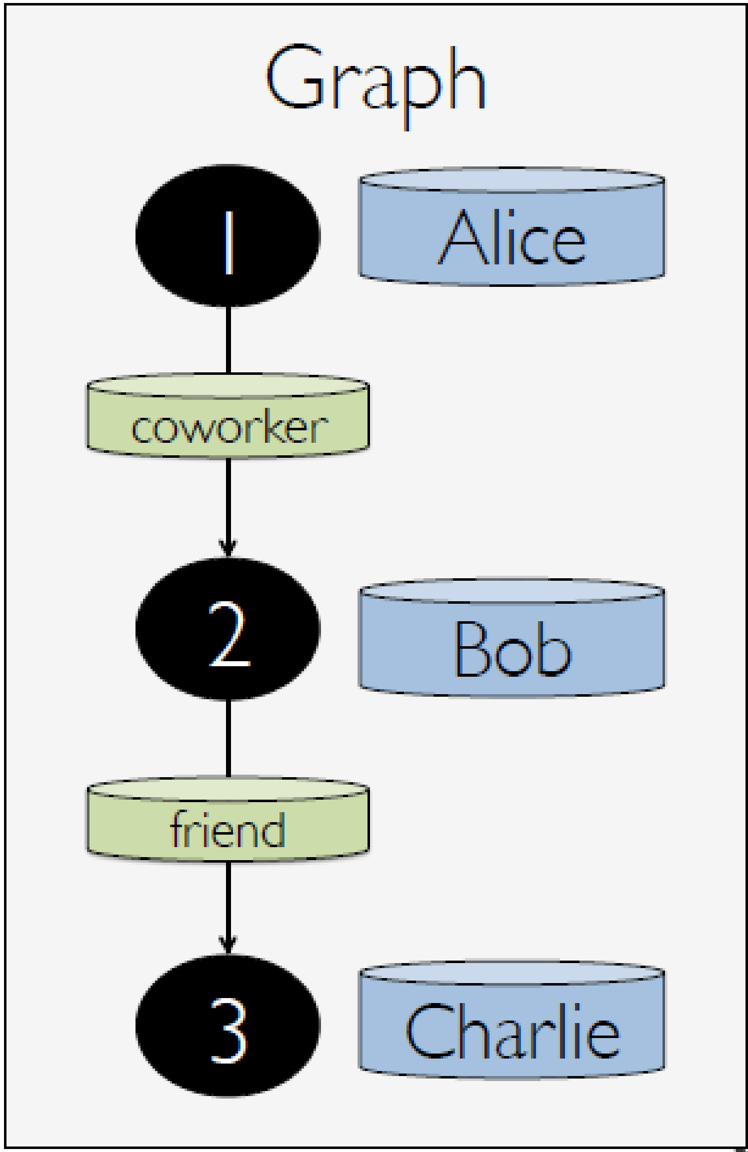
- User Profile
- Current PageRank Value

Edge Property:

- Weights
- Relationships
- Timestamps

Creating a Graph (Scala)

```
type VertexId = Long

val vertices: RDD[(VertexId, String)] =
  sc.parallelize(List(
    (1L, "Alice"),
    (2L, "Bob"),
    (3L, "Charlie")))

class Edge[ED](
  val srcId: VertexId,
  val dstId: VertexId,
  val attr: ED)

val edges: RDD[Edge[String]] =
  sc.parallelize(List(
    Edge(1L, 2L, "coworker"),
    Edge(2L, 3L, "friend")))

val graph = Graph(vertices, edges)
```



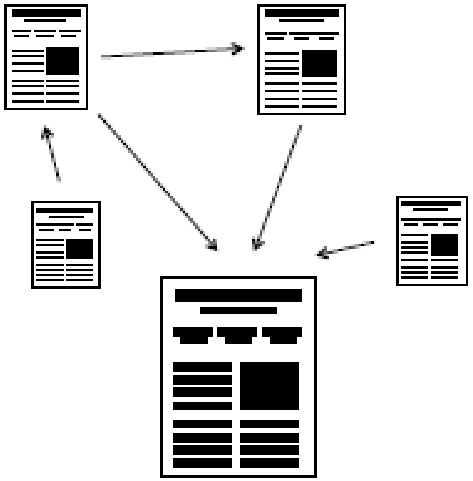
Graph Operations (Scala)

```
class Graph[VD, ED] {
    // Table Views -----------------------
    def vertices: RDD[(VertexId, VD)]
    def edges: RDD[Edge[ED]]
    def triplets: RDD[EdgeTriplet[VD, ED]]
    // Transformations ------------------------------------
    def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
    def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD2, ED]
    def reverse: Graph[VD, ED]
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
    // Joins ----------------------------------------
    def outerJoinVertices[U, VD2]
        (tbl: RDD[(VertexId, U)])
        (f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
    // Computation ----------------------------------
    def aggregateMessages[A](
        sendMsg: EdgeContext[VD, ED, A] => Unit,
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]
```

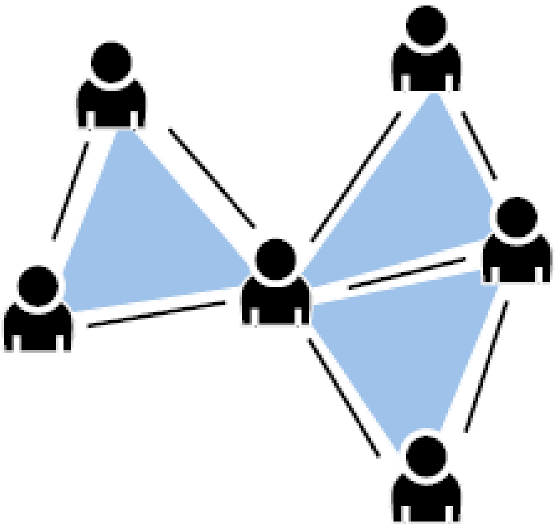
Built-in Algorithms (Scala)

```scala
    // Continued from previous slide
    def pageRank(tol: Double): Graph[Double, Double]
    def triangleCount(): Graph[Int, ED]
    def connectedComponents(): Graph[VertexId, ED]
    // ...and more: org.apache.spark.graphx.lib
}
```

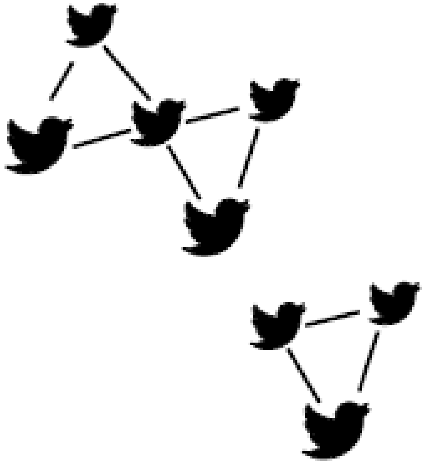
PageRank



Triangle Count

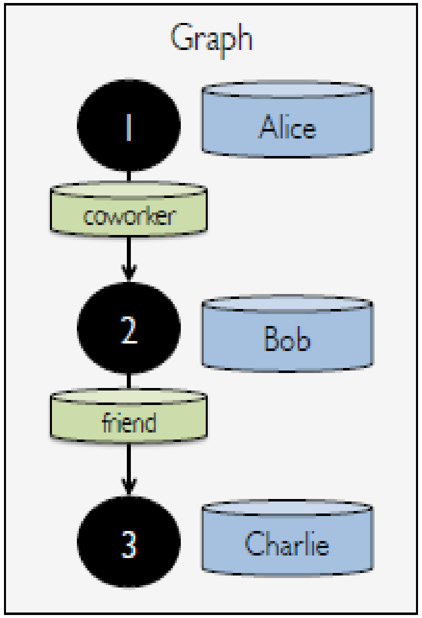


Connected Components



The triplets view

```scala
class Graph[VD, ED] {
  def triplets: RDD[EdgeTriplet[VD, ED]]
}

class EdgeTriplet[VD, ED](
  val srcId: VertexId, val dstId: VertexId, val attr: ED,
  val srcAttr: VD, val dstAttr: VD)
```



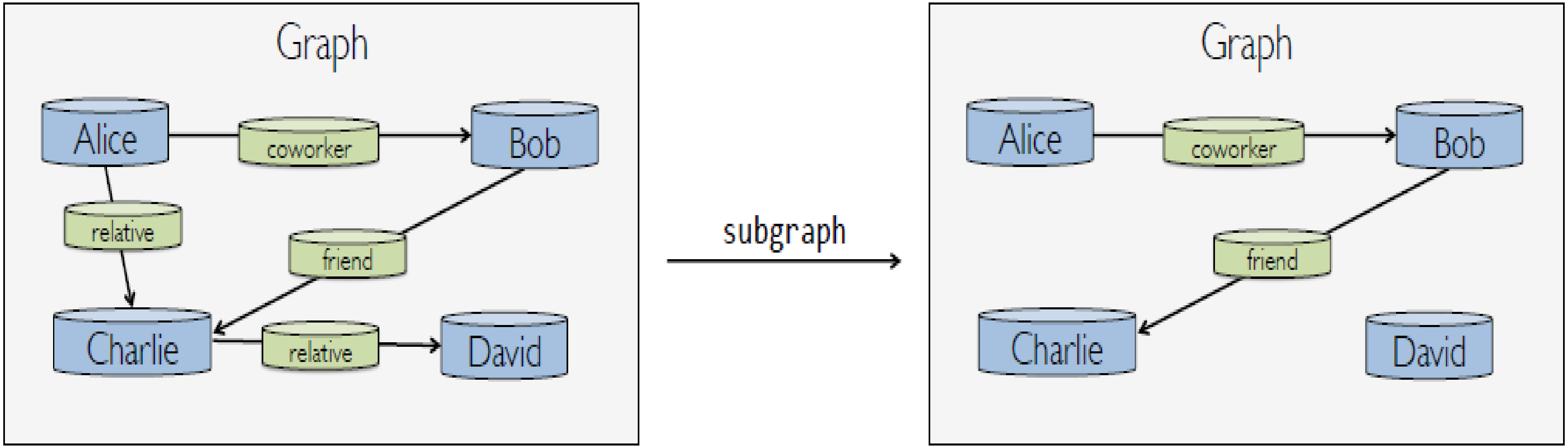
triplets →

RDD

srcAttr	dstAttr	attr
Alice	coworker	Bob
Bob	friend	Charlie

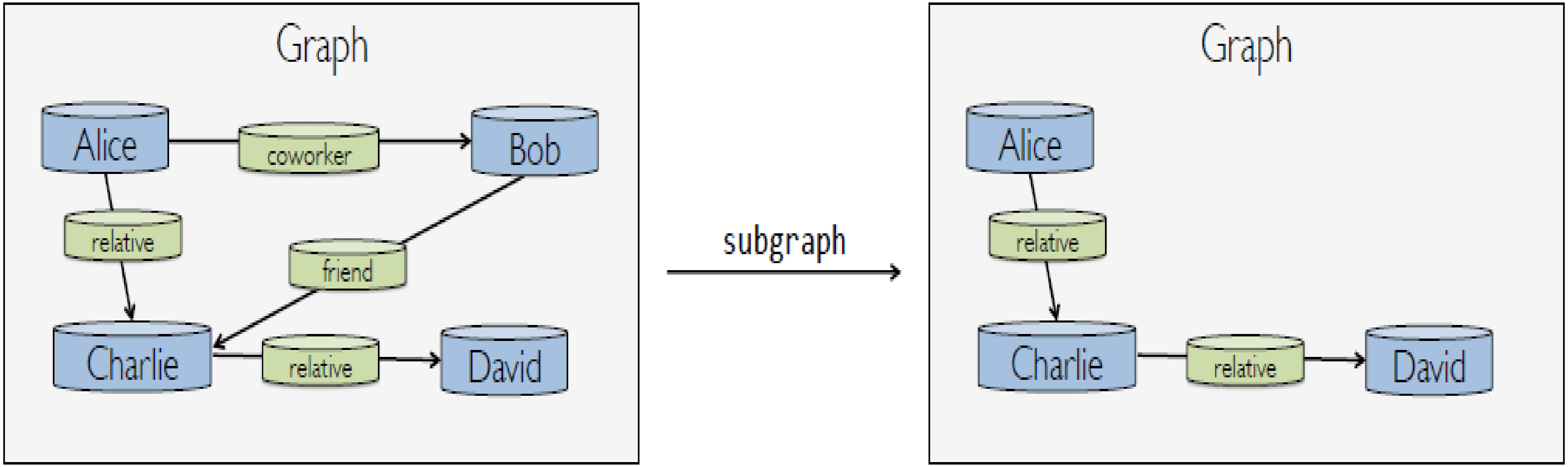
The subgraph transformation

```
class Graph[VD, ED] {
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(epred = (edge) => edge.attr != "relative")
```



The subgraph transformation

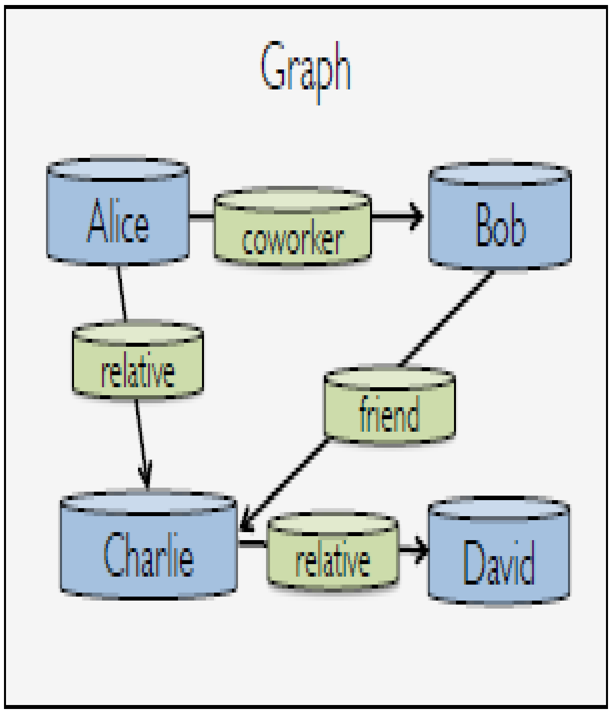
```
class Graph[VD, ED] {
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(vpred = (id, name) => name != "Bob")
```



Computation with aggregateMessages

```
class Graph[VD, ED] {
  def aggregateMessages[A](
    sendMsg: EdgeContext[VD, ED, A] => Unit,
    mergeMsg: (A, A) => A): RDD[(VertexId, A)]
}

class EdgeContext[VD, ED, A](
    val srcId: VertexId, val dstId: VertexId, val attr: ED,
    val srcAttr: VD, val dstAttr: VD) {
  def sendToSrc(msg: A)
  def sendToDst(msg: A)
}

graph.aggregateMessages(
  ctx => {
    ctx.sendToSrc(1)
    ctx.sendToDst(1)
  },
  _ + _)
```

Computation with aggregateMessages

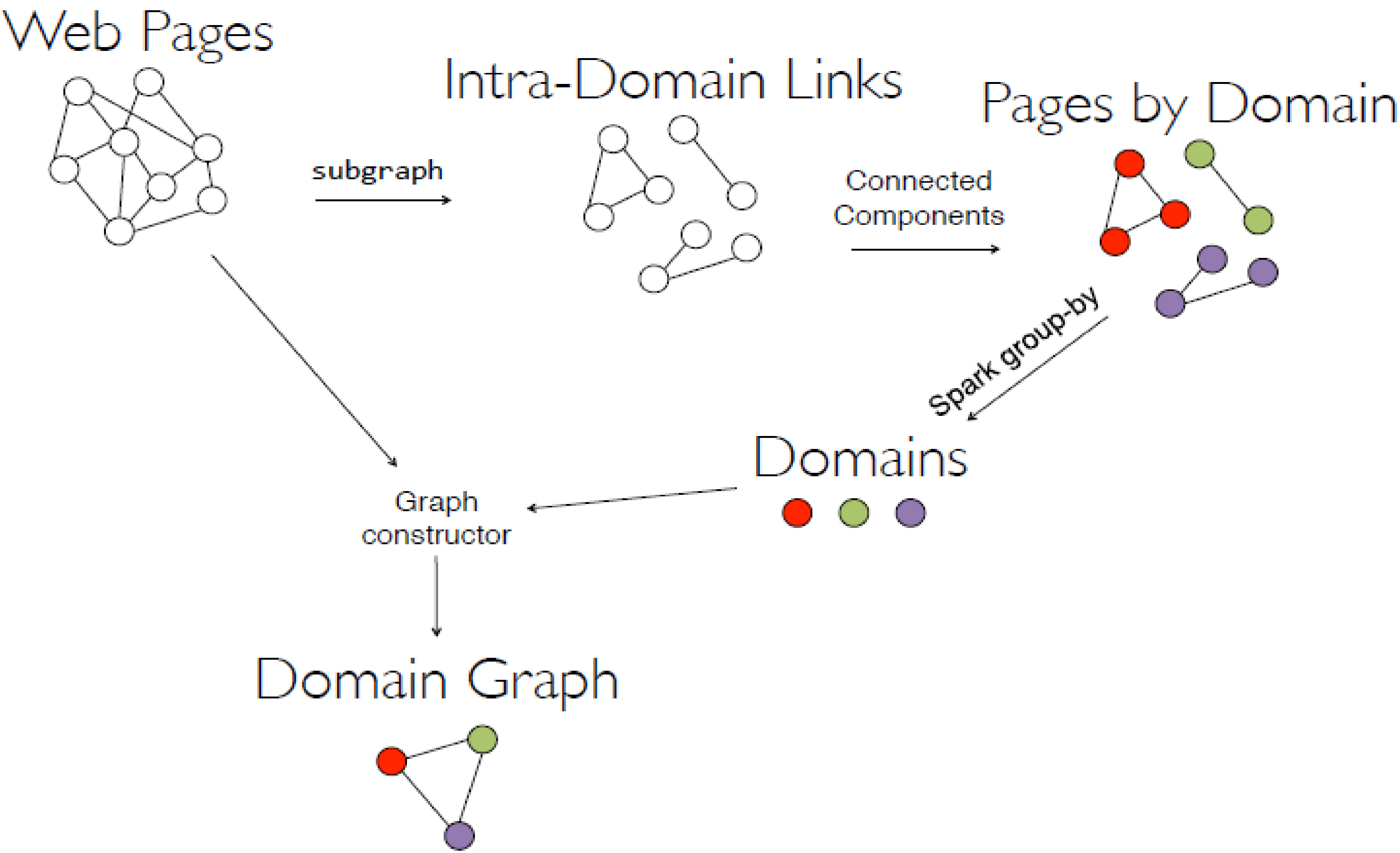


aggregateMessages →

RDD

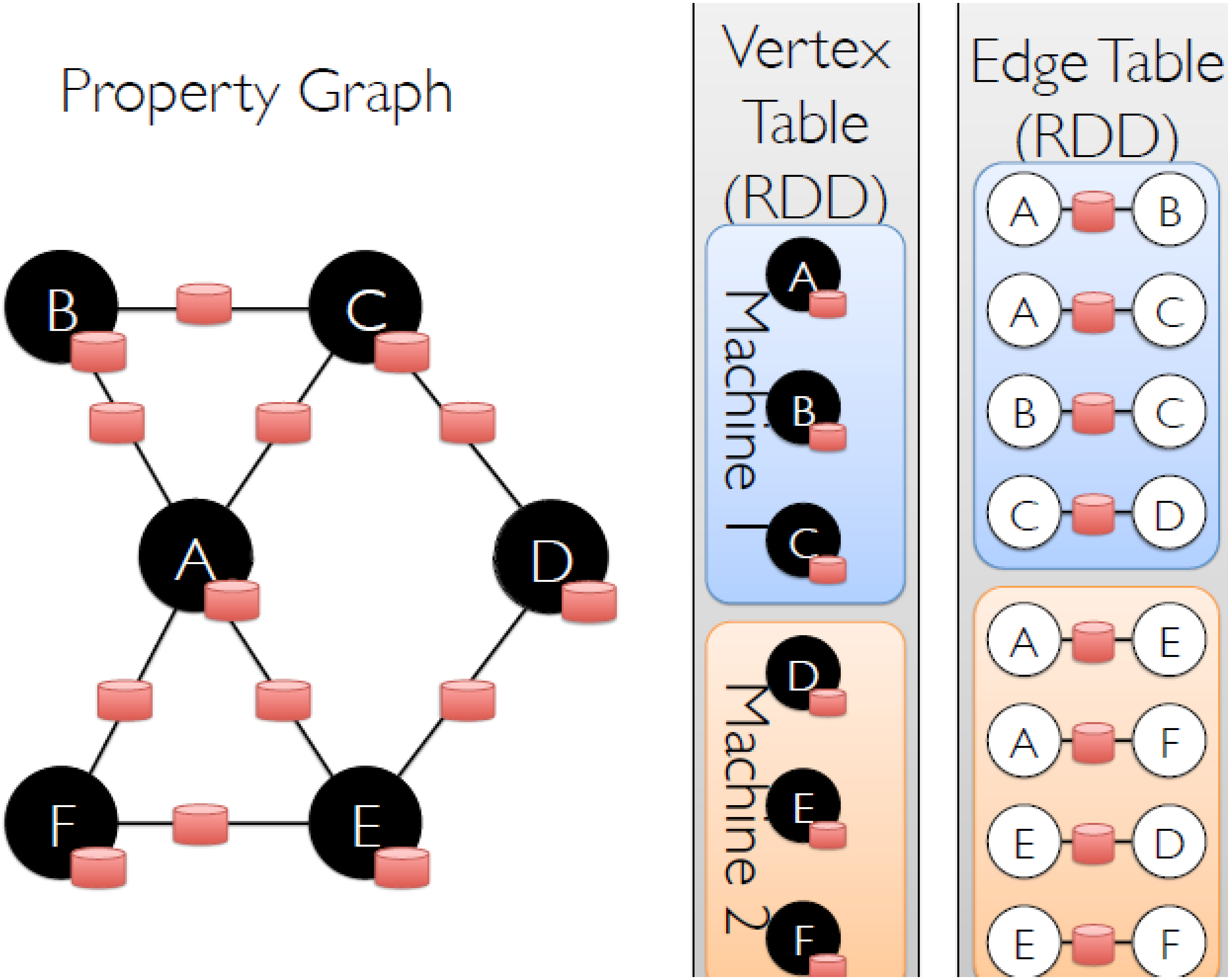
vertex id	degree
Alice	2
Bob	2
Charlie	3
David	1

Example: Graph Coarsening



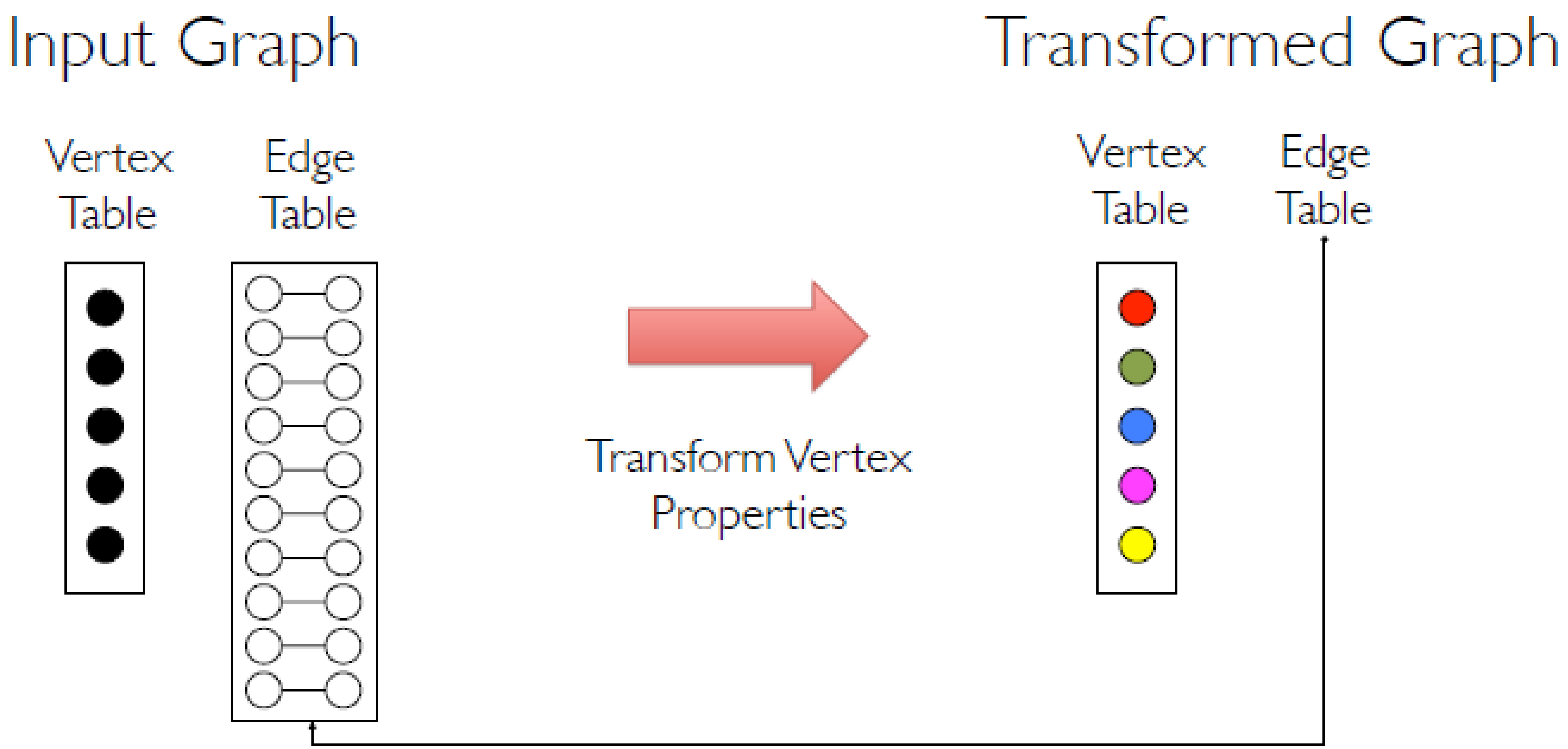
How GraphX Works

Storing Graphs as Tables

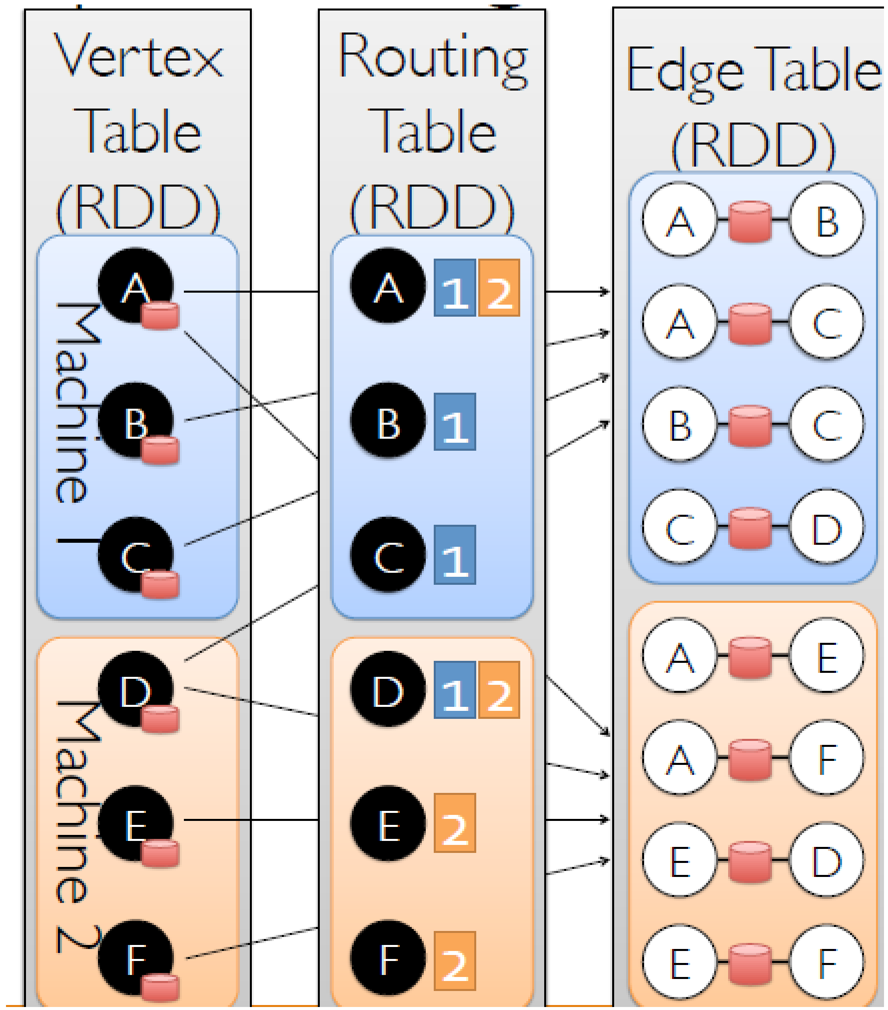


Simple Operations

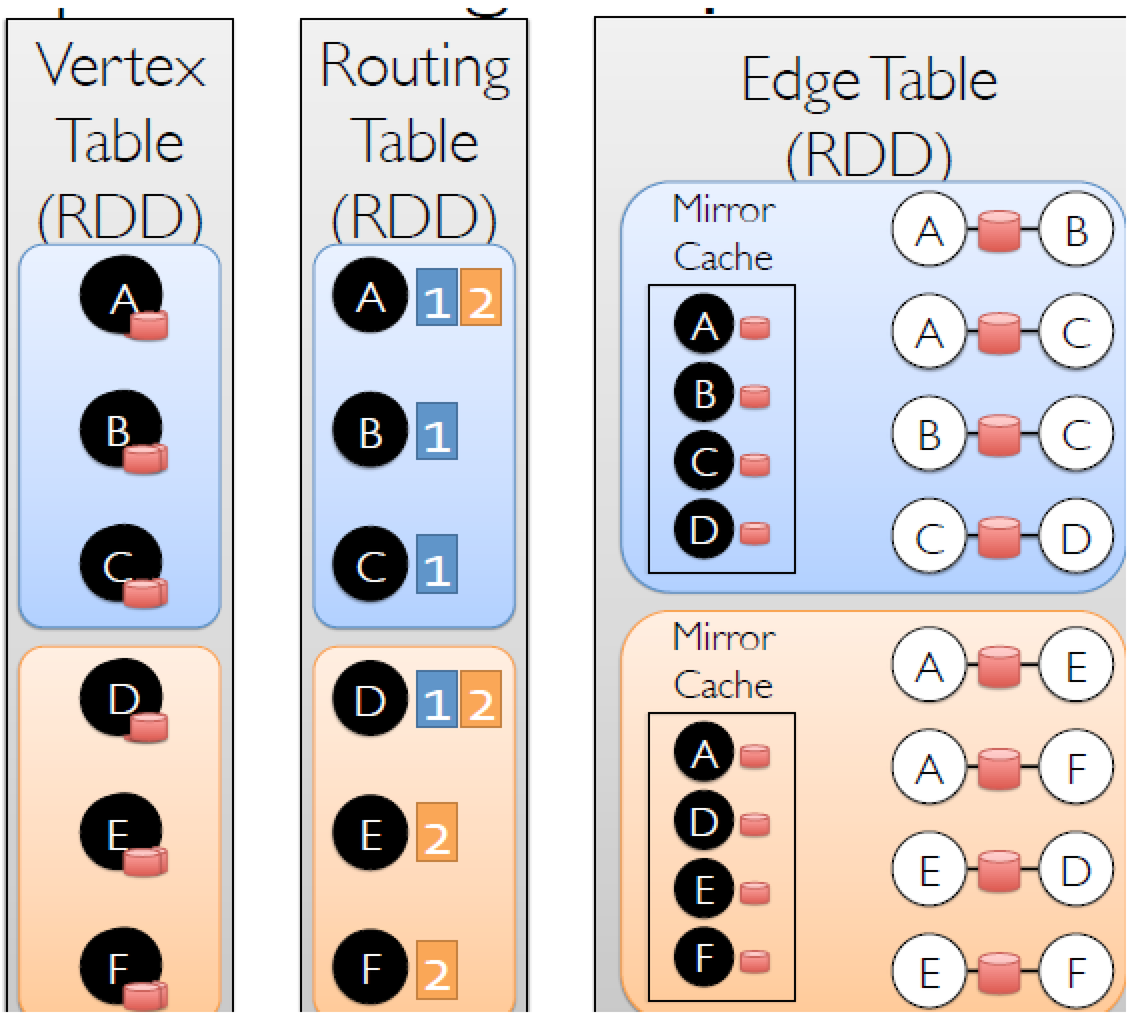
Reuse vertices or edges across multiple graphs



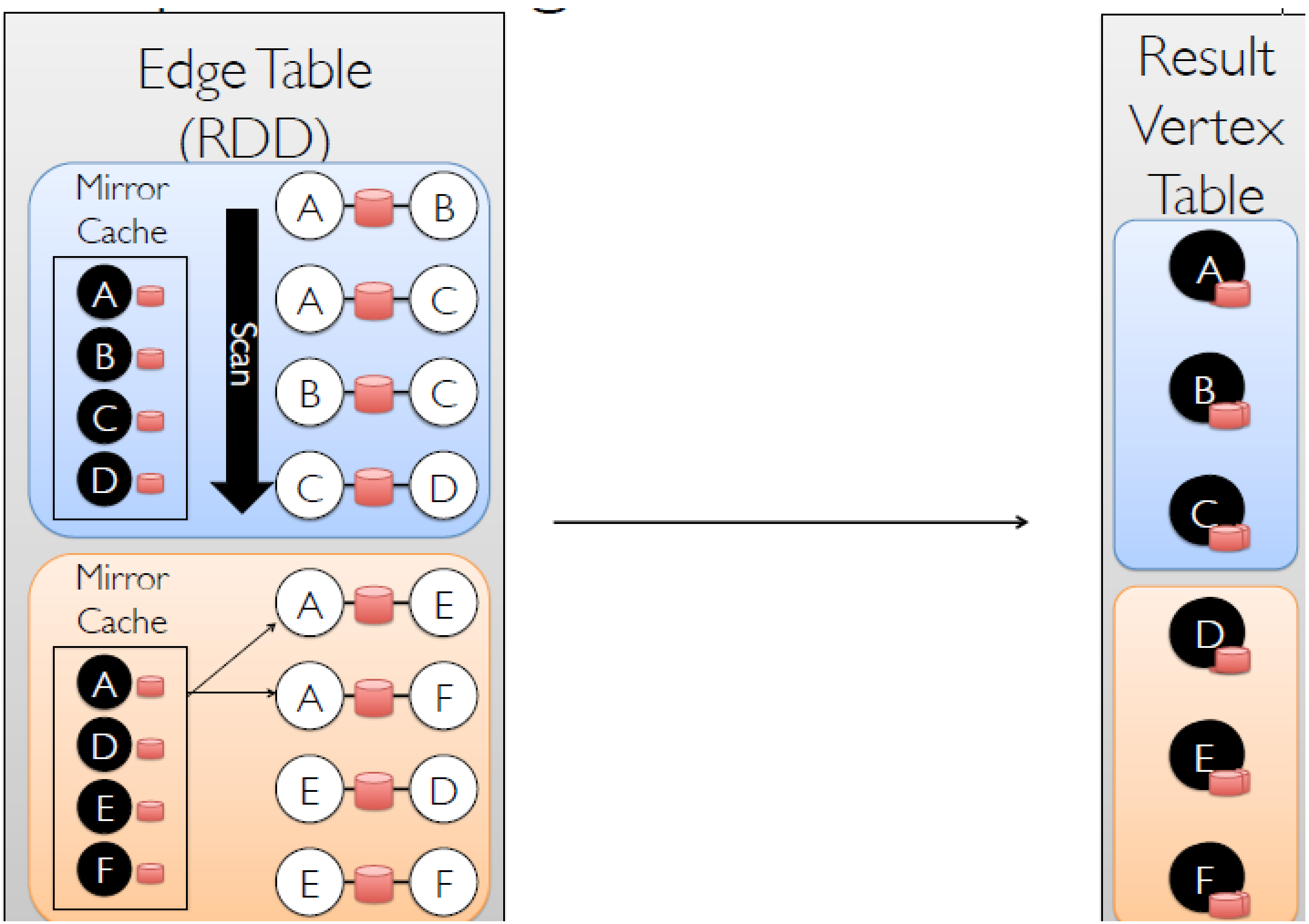
Implementing triplets



Implementing triplets



Implementing aggregateMessages



Future of GraphX

1. Language support

a) Java API

b) Python API: collaborating with Intel, SPARK-3789

2. More algorithms

a) LDA (topic modeling)

b) Correlation clustering

3. Research

a) Local graphs

b) Streaming/time-varying graphs

c) Graph database–like queries

Other 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/

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