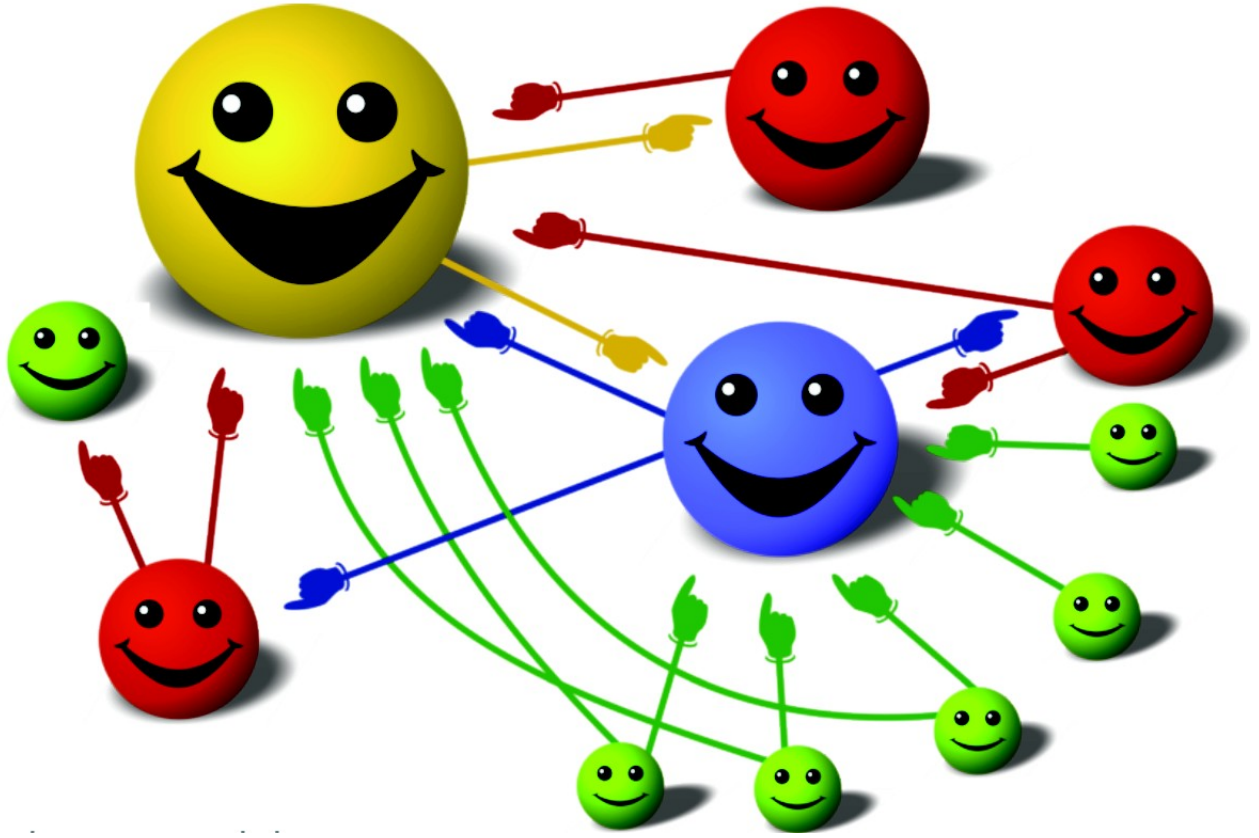
Optimizing Joins-1
By Mohit Kumar

Spark Internals:Pagerank

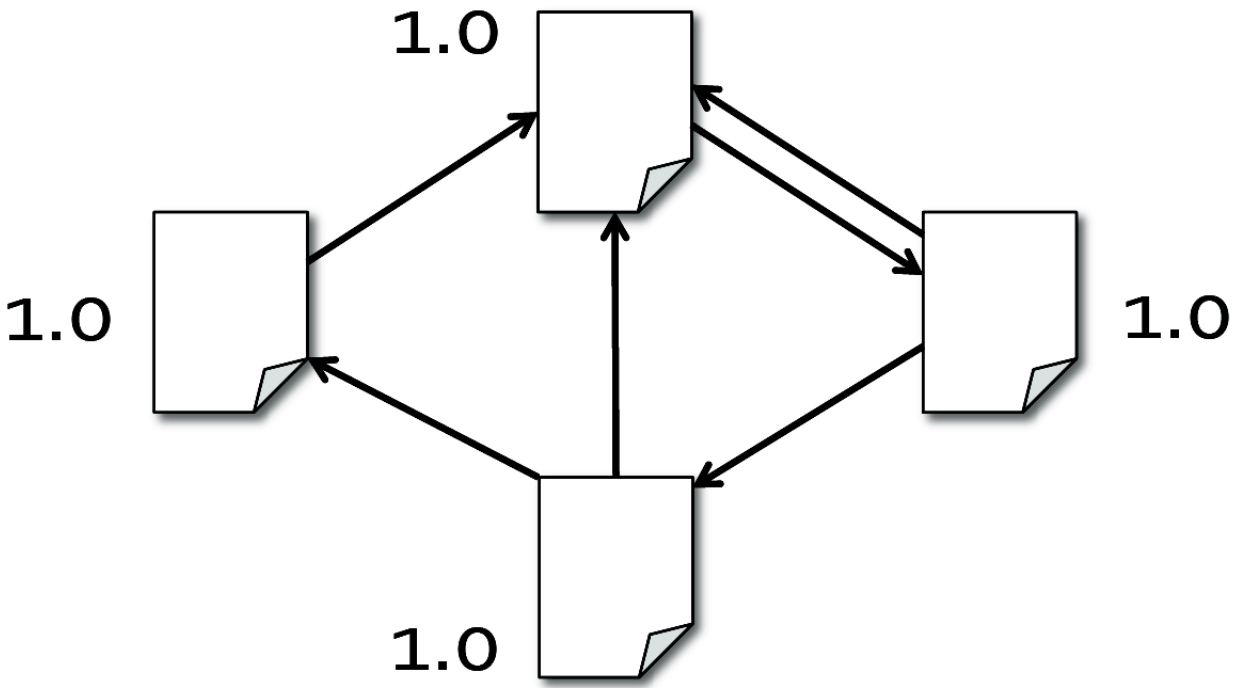
Give pages ranks (scores) based on links to them

- » Links from many pages ➔ high rank
- » Link from a high-rank page ➔ high rank



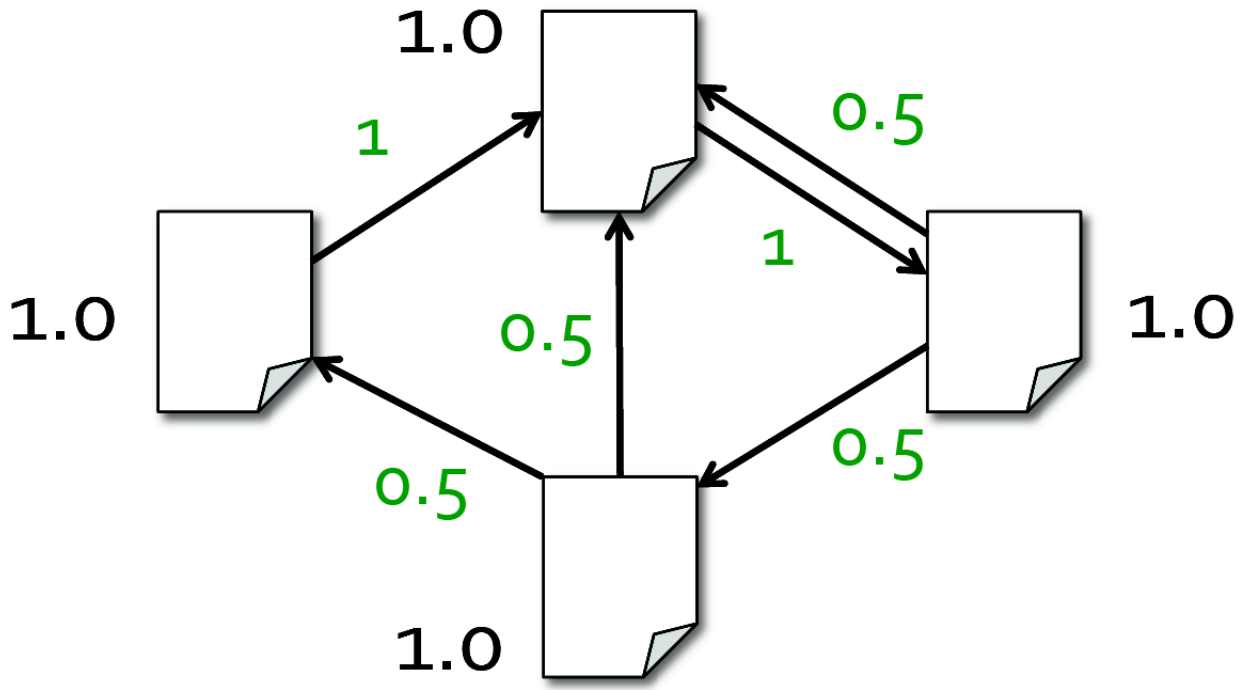
Spark Internals:Pagerank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



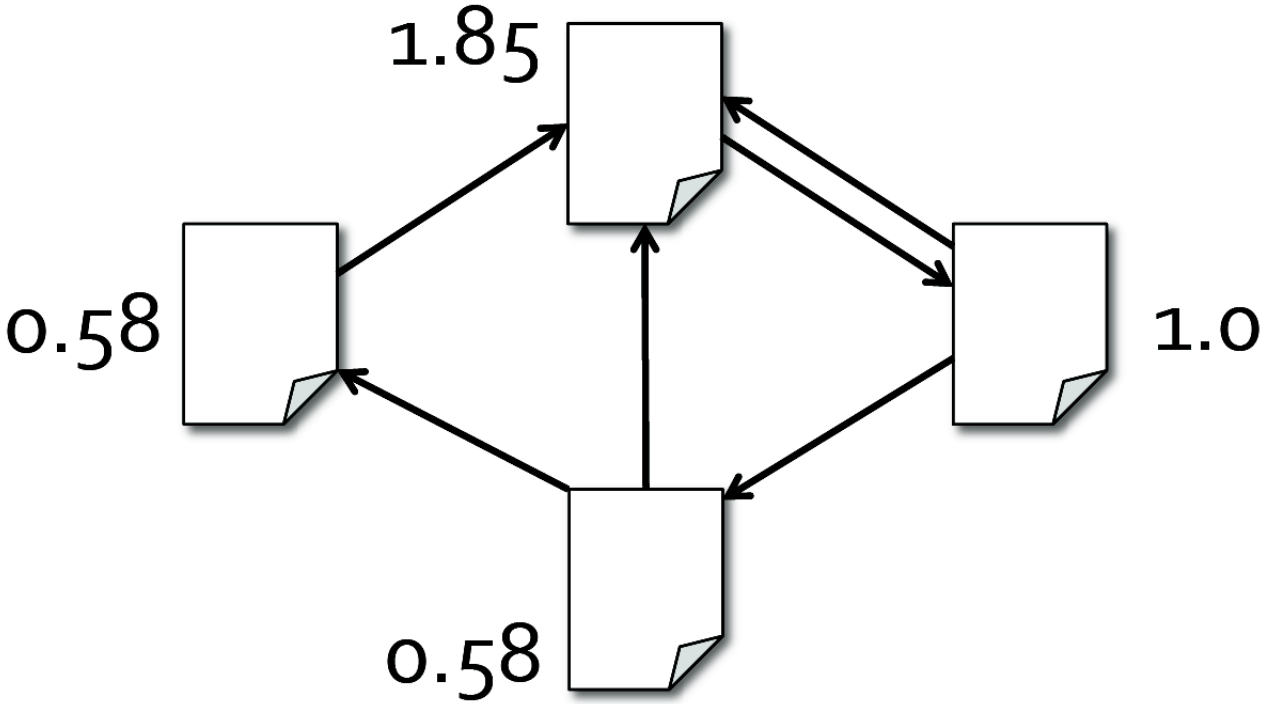
Spark Internals:Pagerank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



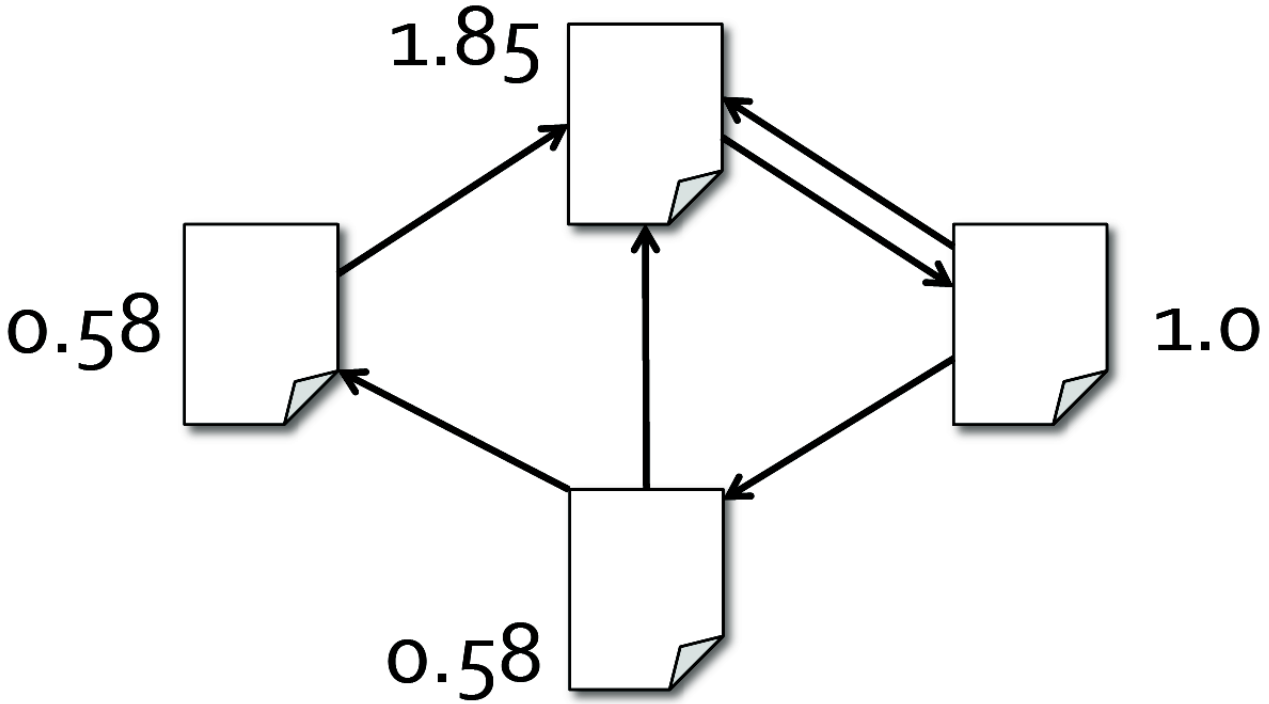
Spark Internals:Pagerank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / |$neighbors_p$| to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



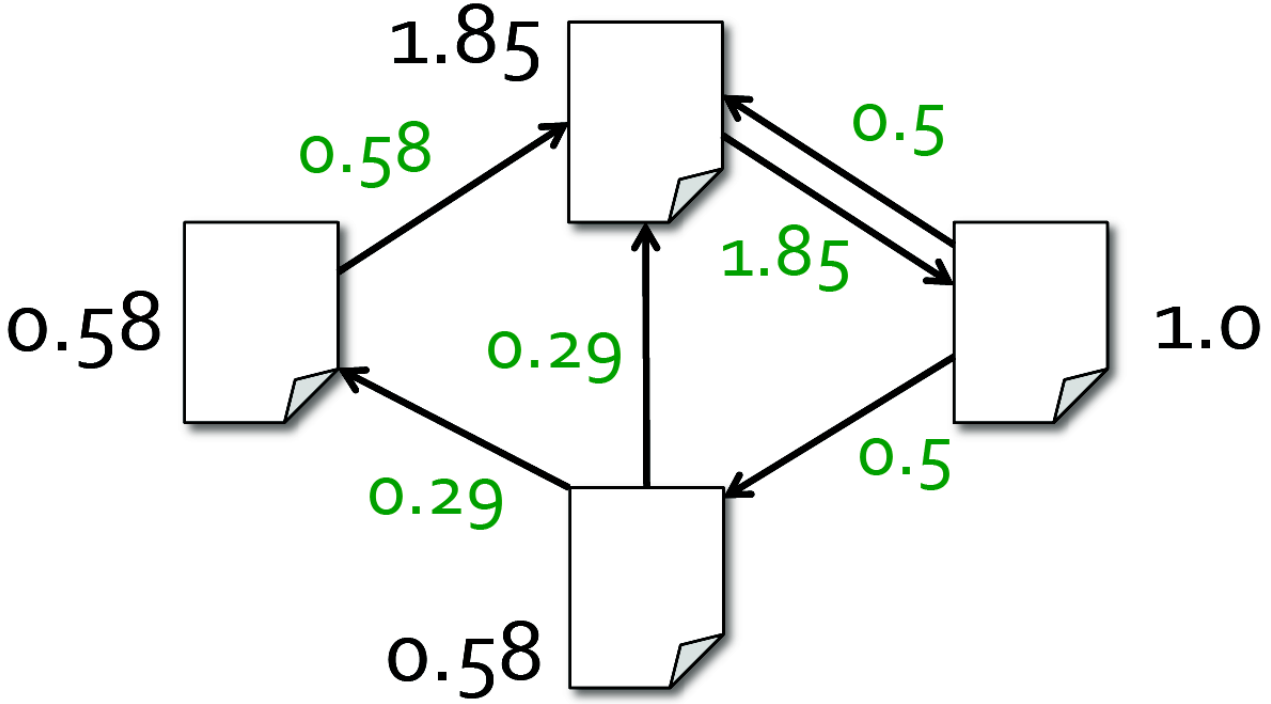
Spark Internals:Pagerank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



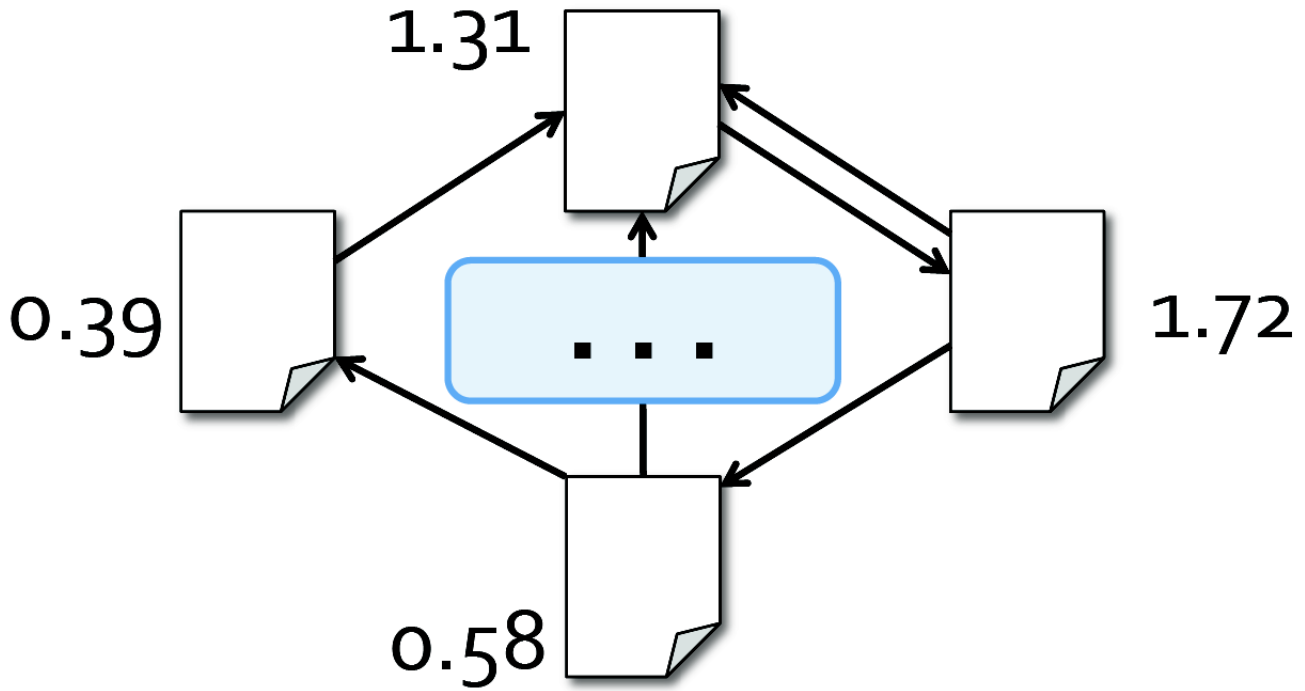
Spark Internals:Pagerank

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Spark Internals:Pagerank

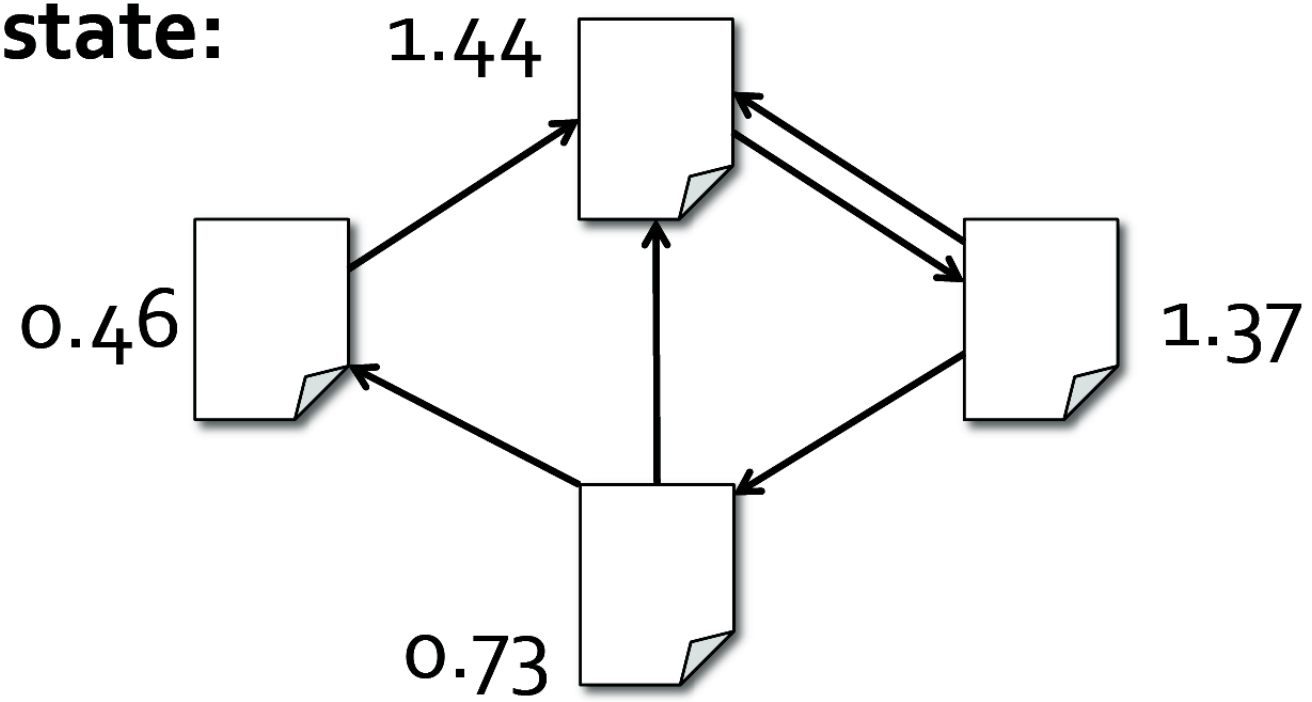
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Spark Internals:Pagerank

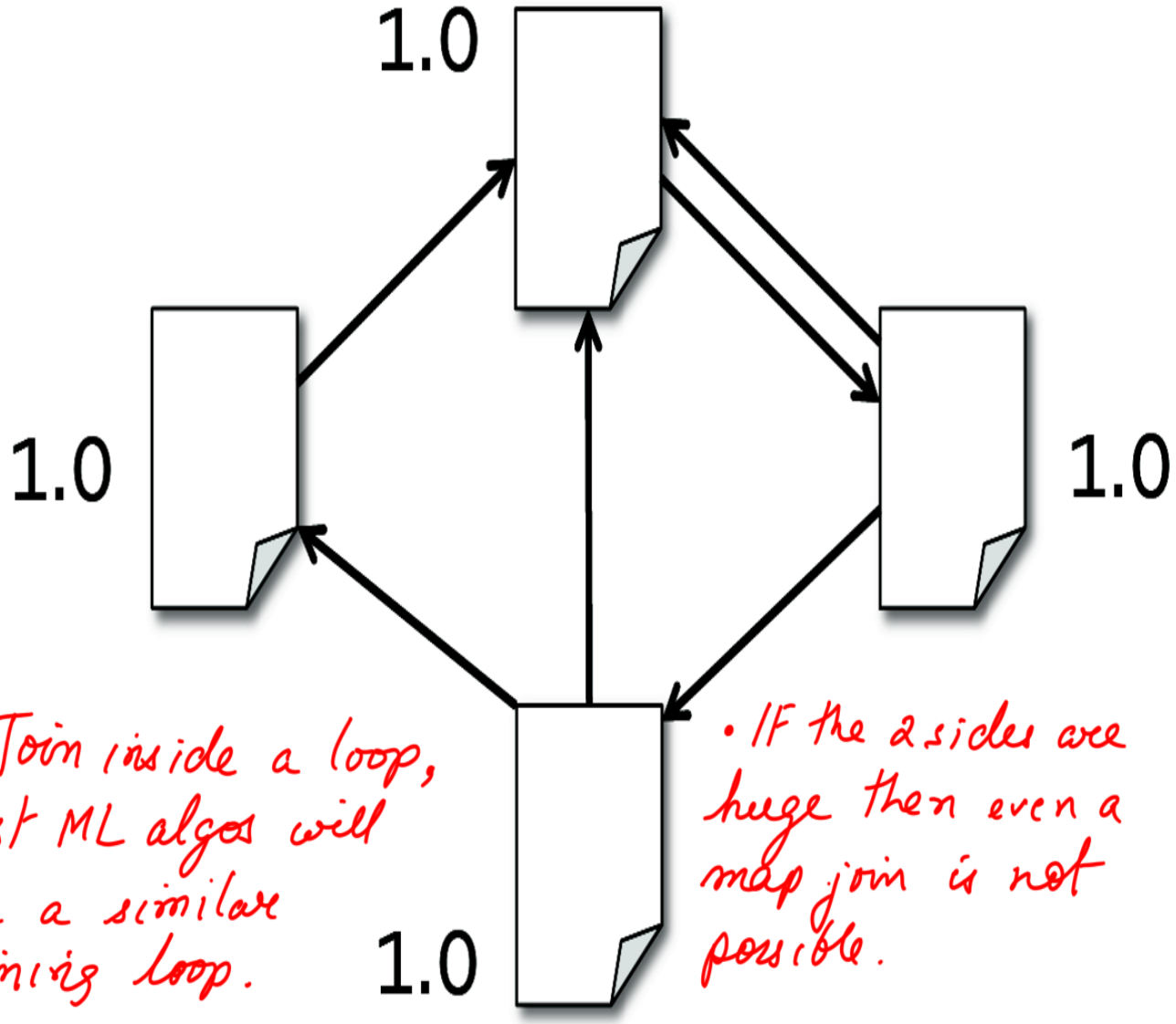
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

Final state:



Spark Internals:Pagerank

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.map(lambda k: (k[0],1))

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .map(lambda x: (x[0],0.15 + x[1]*.85))
```

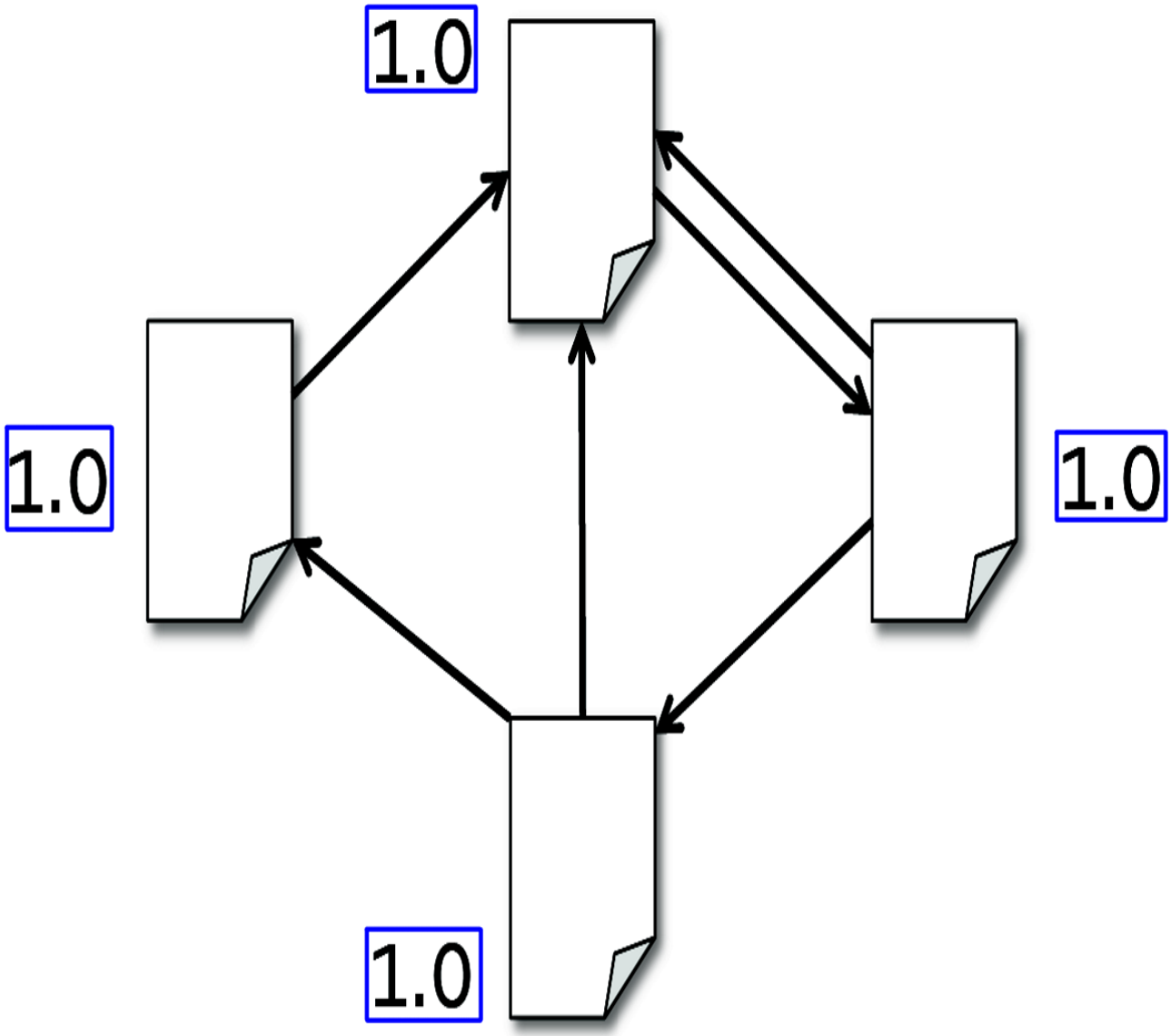


- Join inside a loop, most ML algos will have a similar training loop.

- If the 2 sides are huge then even a map join is not possible.

Spark Internals:Pagerank

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.map(lambda k: (k[0],1))

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .map(lambda x: (x[0],0.15 + x[1]*.85))
```

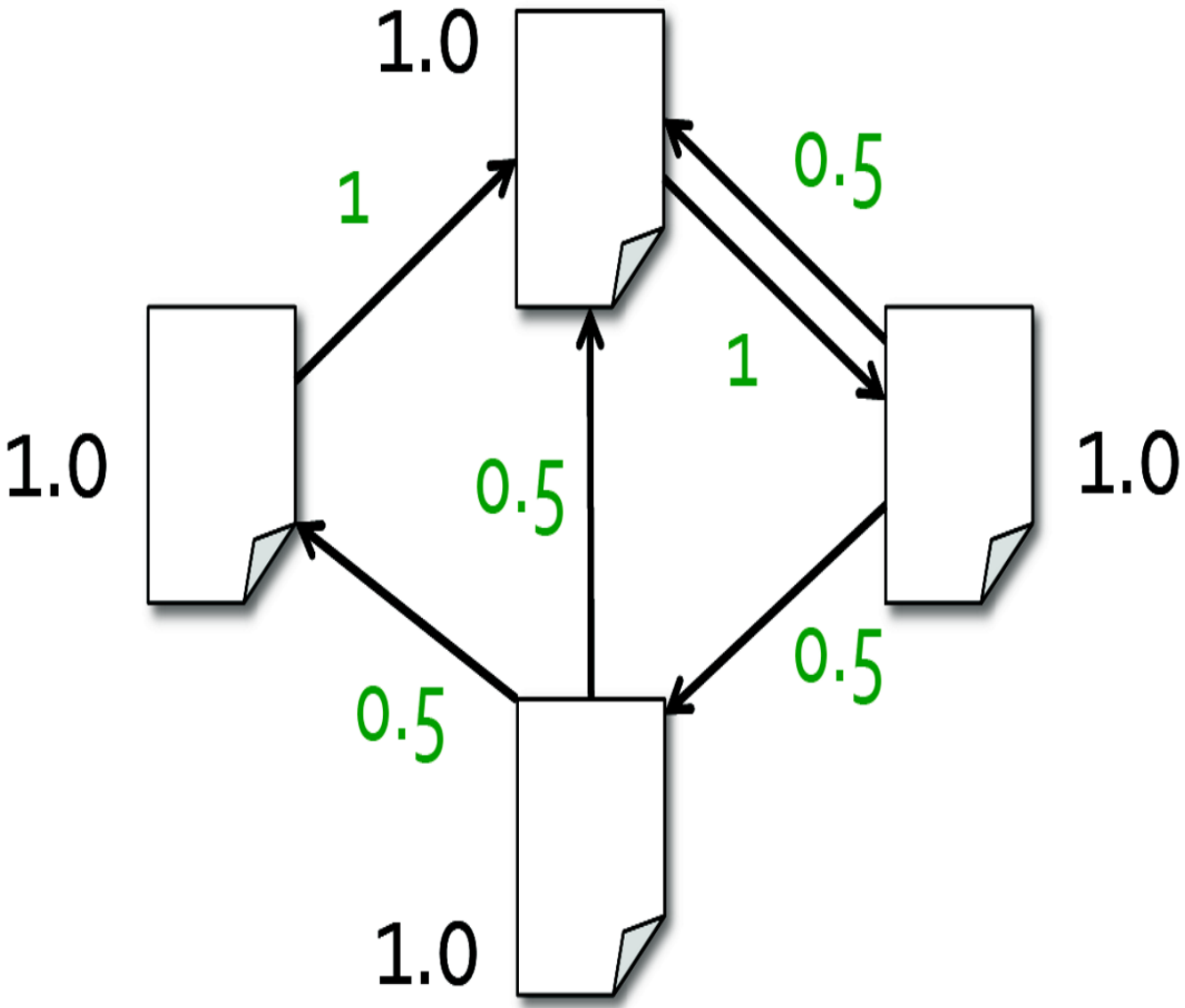


Spark Internals:Pagerank

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.map(lambda k: (k[0],1))
```

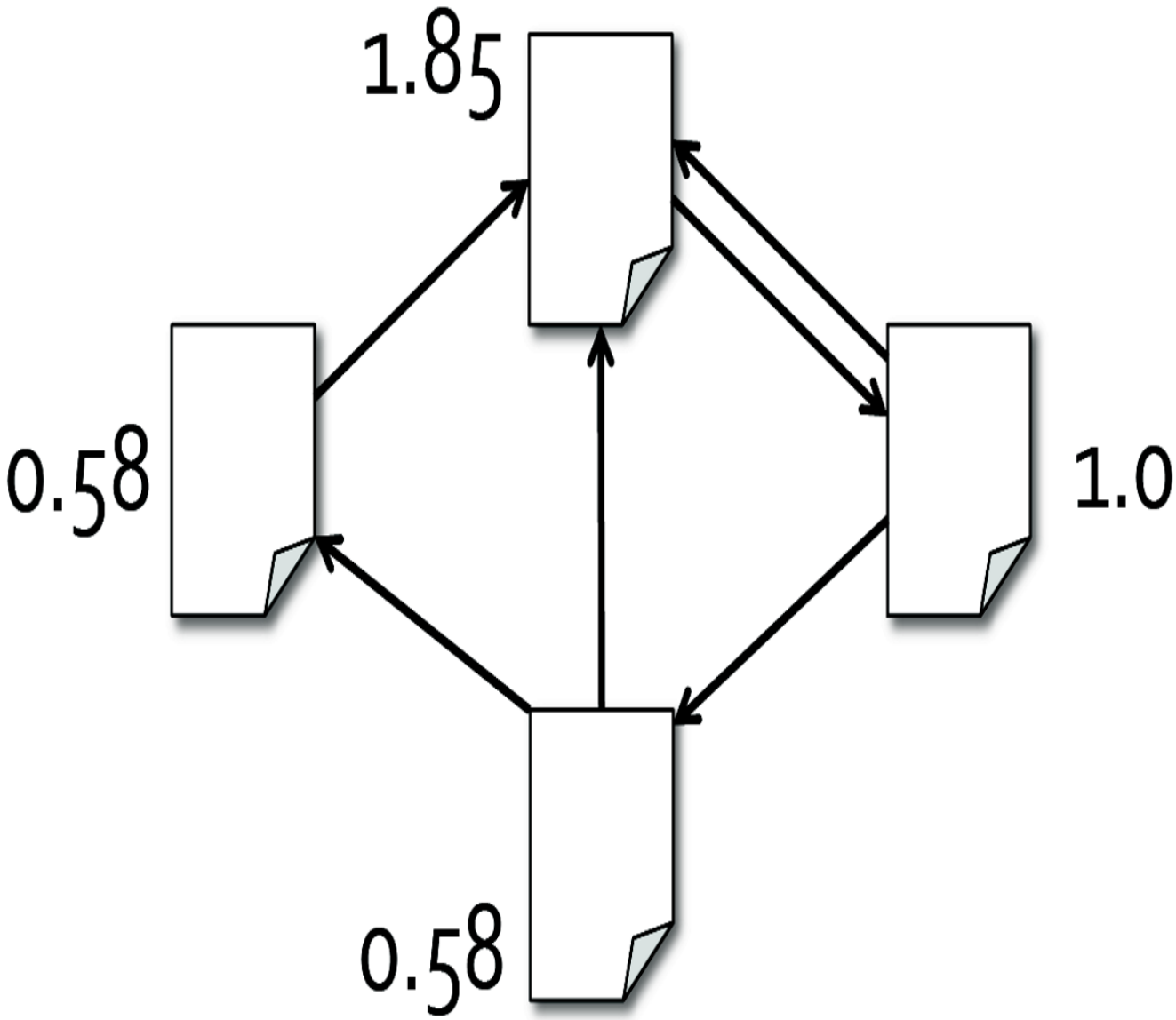
```
def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result
```

```
for i in range(num_iter) :
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .map(lambda x: (x[0],0.15 + x[1]*.85))
```

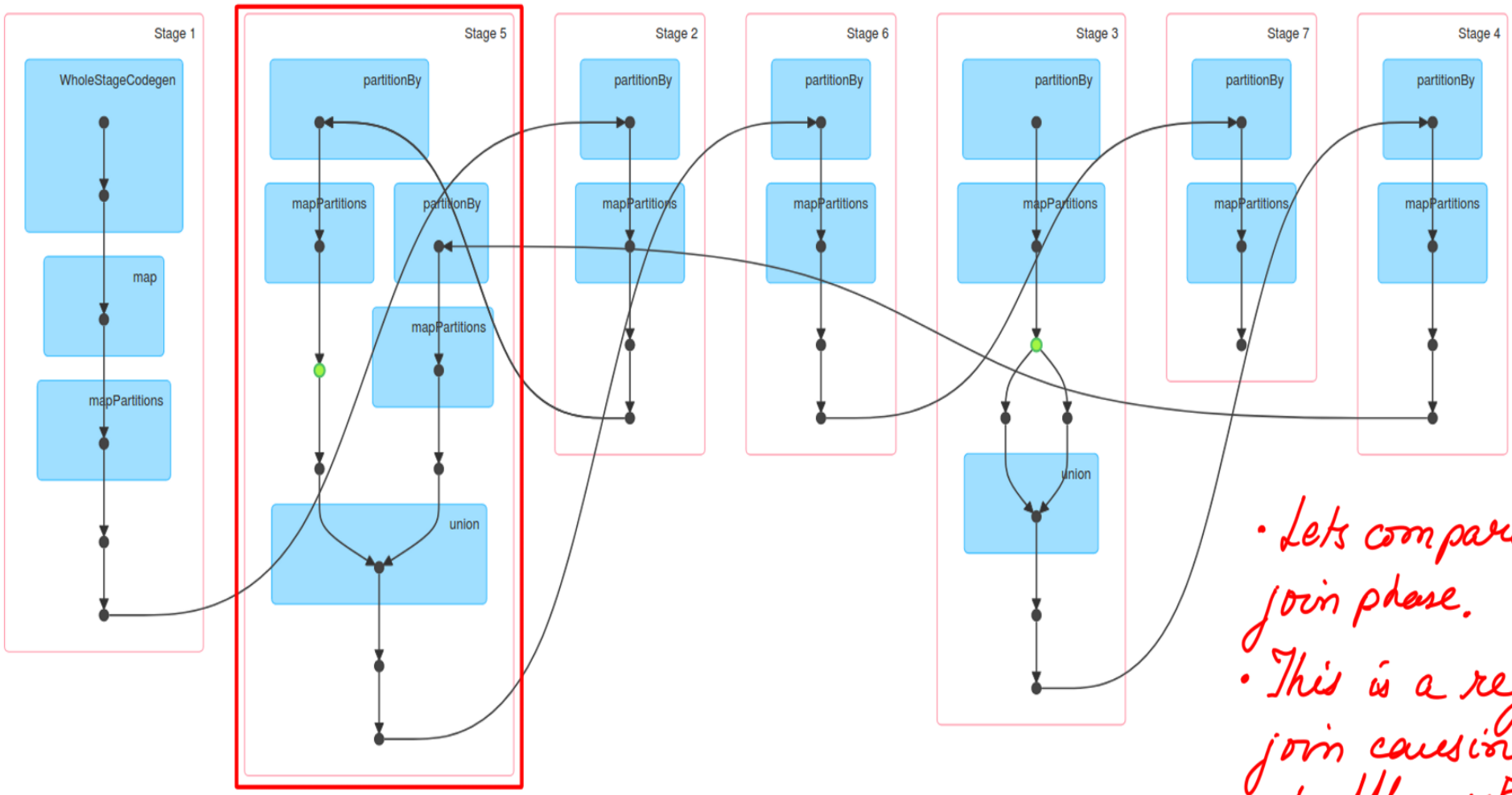


Spark Internals:Pagerank

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.map(lambda k: (k[0],1))

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .map(lambda x: (x[0],0.15 + x[1]*.85))
```



Spark Internals:Pagerank

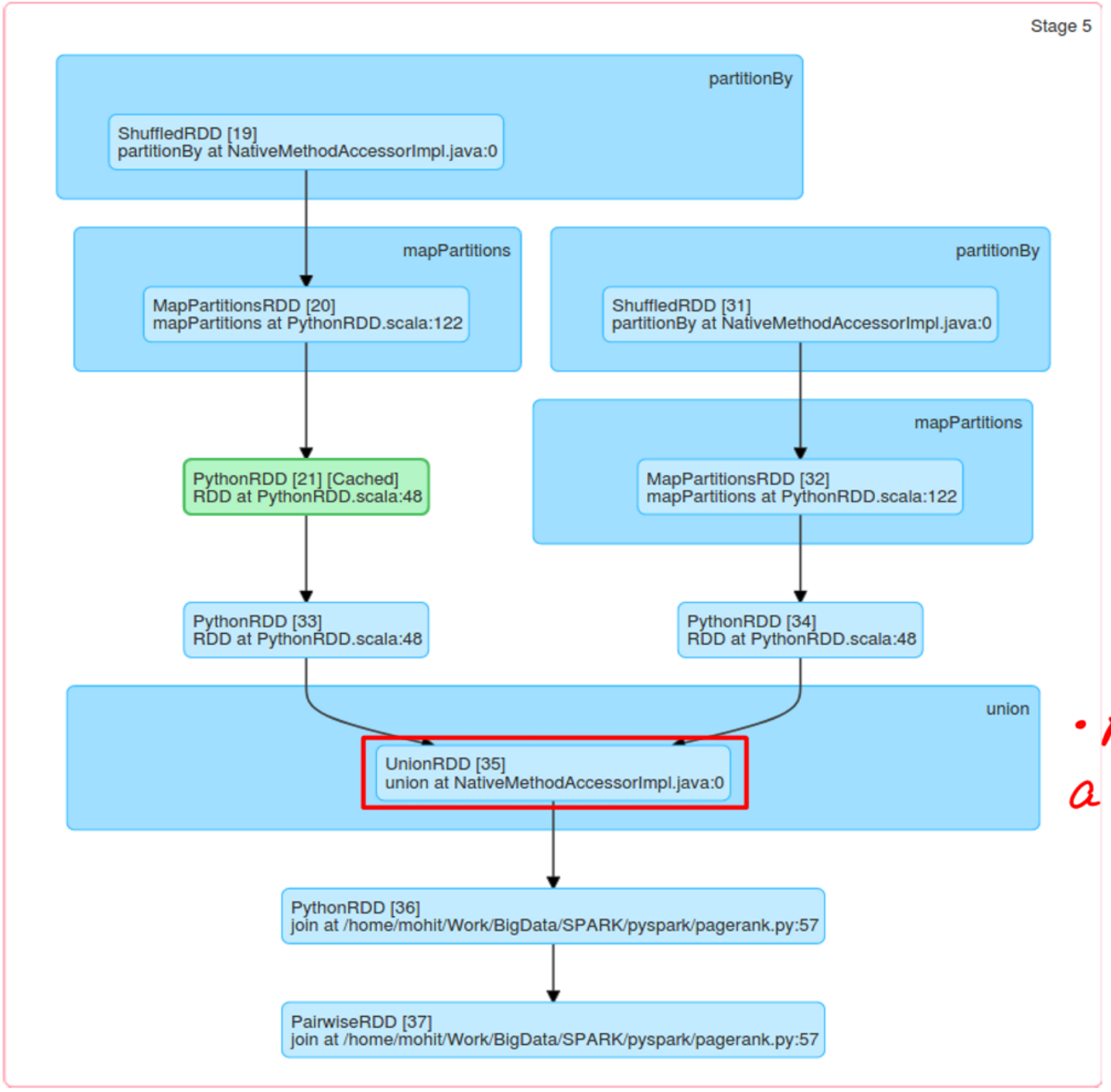


- Lets compare the join phase.
- This is a regular join causing shuffle-sort.

Completed Stages (7)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	collect at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:64	+details	2021/03/15 10:15:04	55 ms	3/3			506.0 B	
6	reduceByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:58	+details	2021/03/15 10:15:04	63 ms	3/3			1017.0 B	506.0 B
5	join at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:57	+details	2021/03/15 10:15:04	68 ms	3/3	310.0 B		334.0 B	1017.0 B
4	reduceByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:58	+details	2021/03/15 10:15:04	64 ms	2/2			768.0 B	334.0 B
3	join at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:57	+details	2021/03/15 10:15:04	0.1 s	2/2	310.0 B		185.0 B	768.0 B
2	groupByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:33	+details	2021/03/15 10:15:04	75 ms	1/1			209.0 B	185.0 B
1	distinct at /home/mohit/Work/BigData/SPARK/pyspark/pagerank.py:33	+details	2021/03/15 10:15:03	0.5 s	1/1				209.0 B

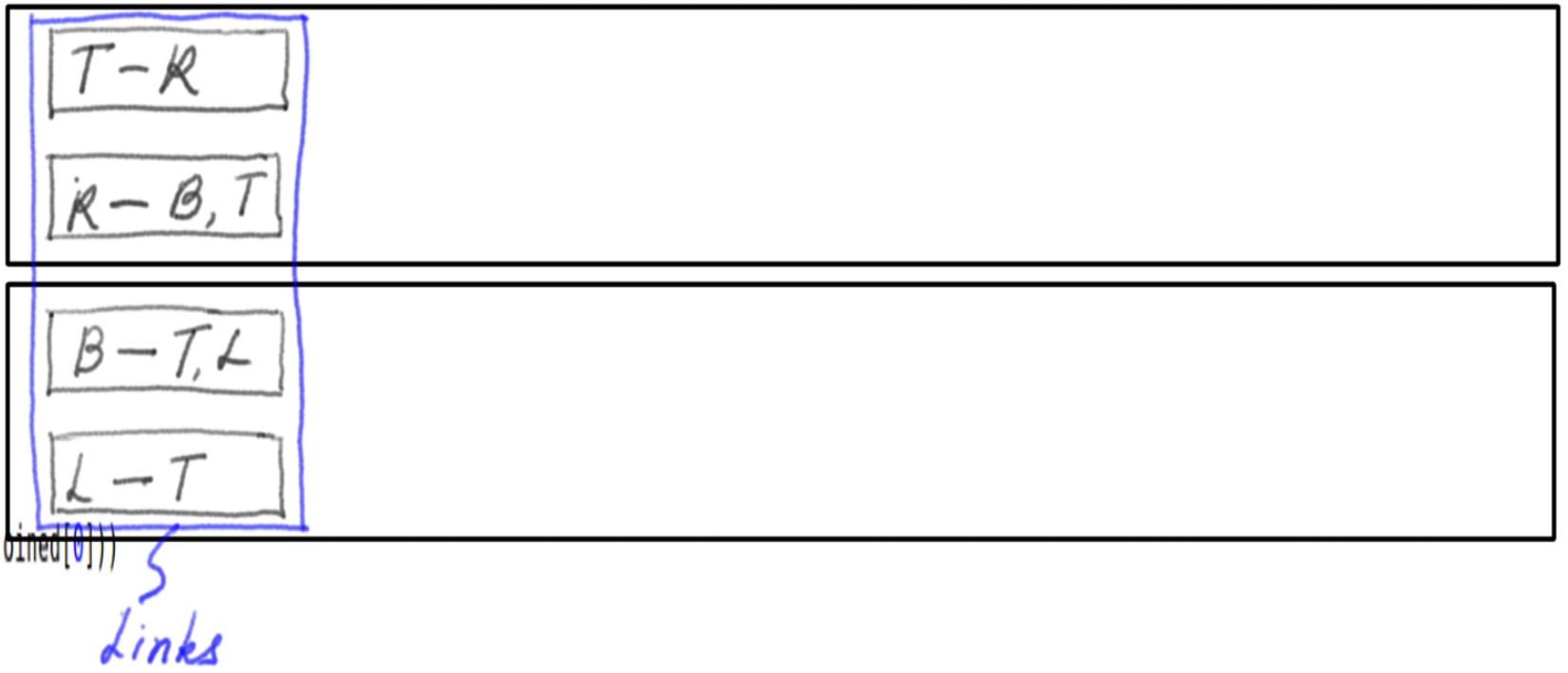
Spark Internals:Pagerank

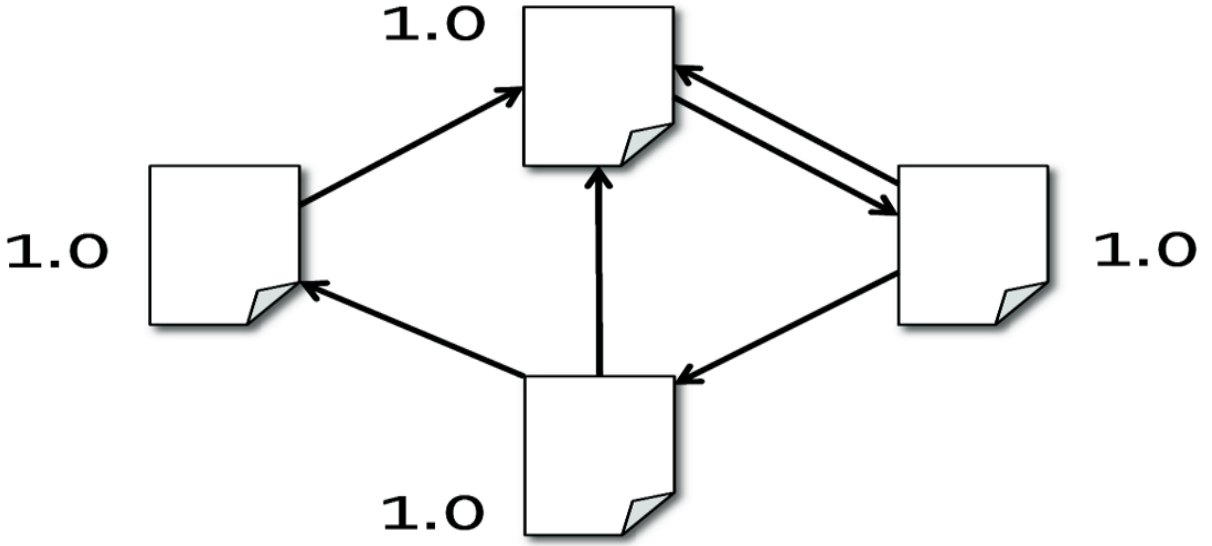


- Regular SMJoin with a UnionRDD.

Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.mapValues(lambda x:1)

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```

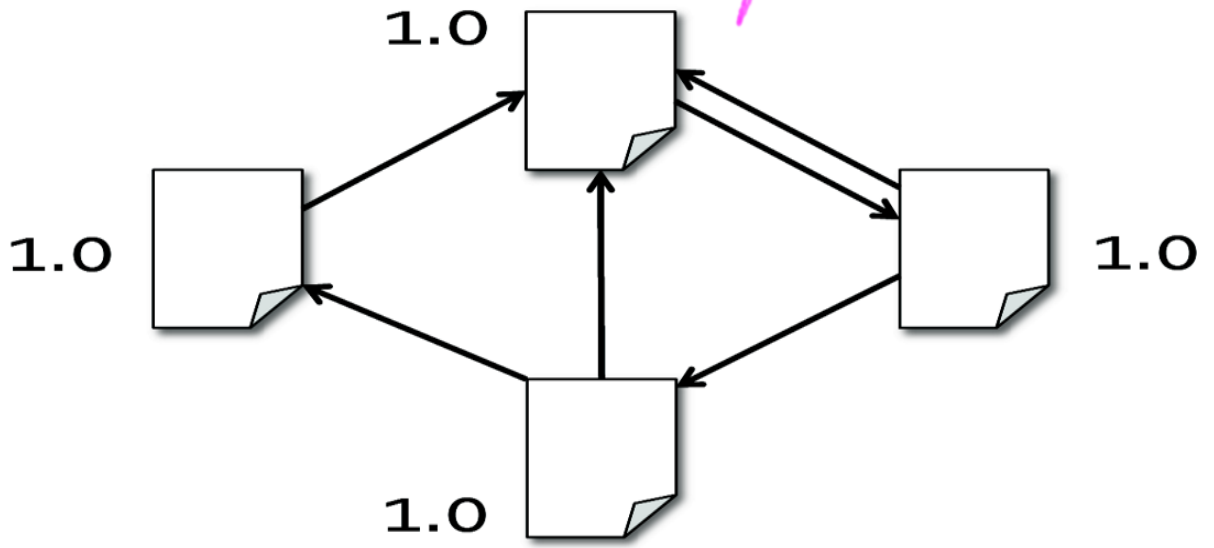




Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.mapValues(lambda x:1)

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```

• Do you notice anything special??

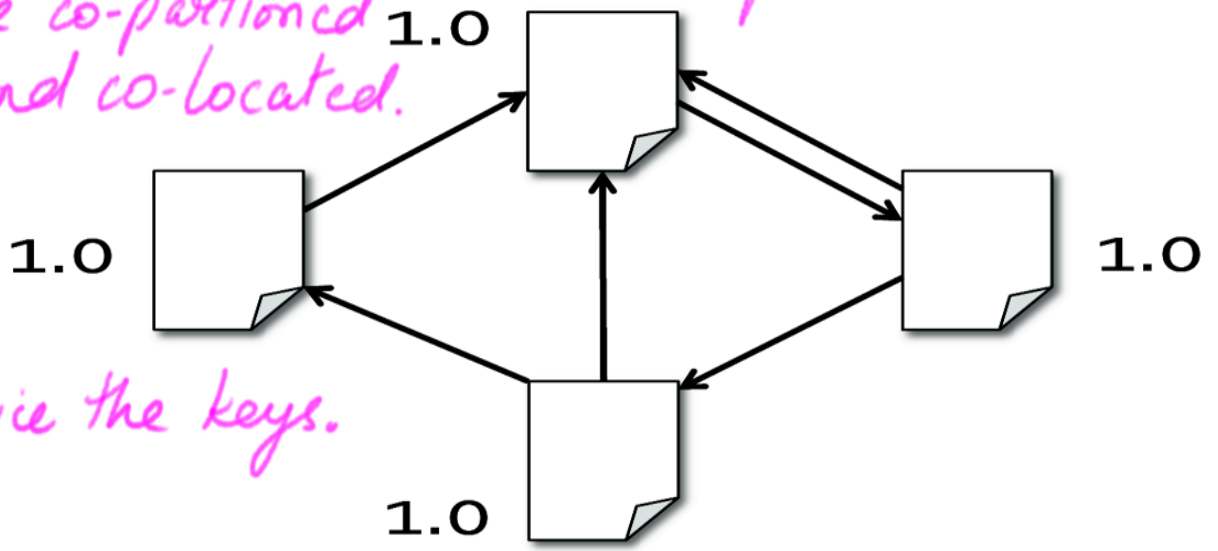


Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.mapValues(lambda x:1)

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```

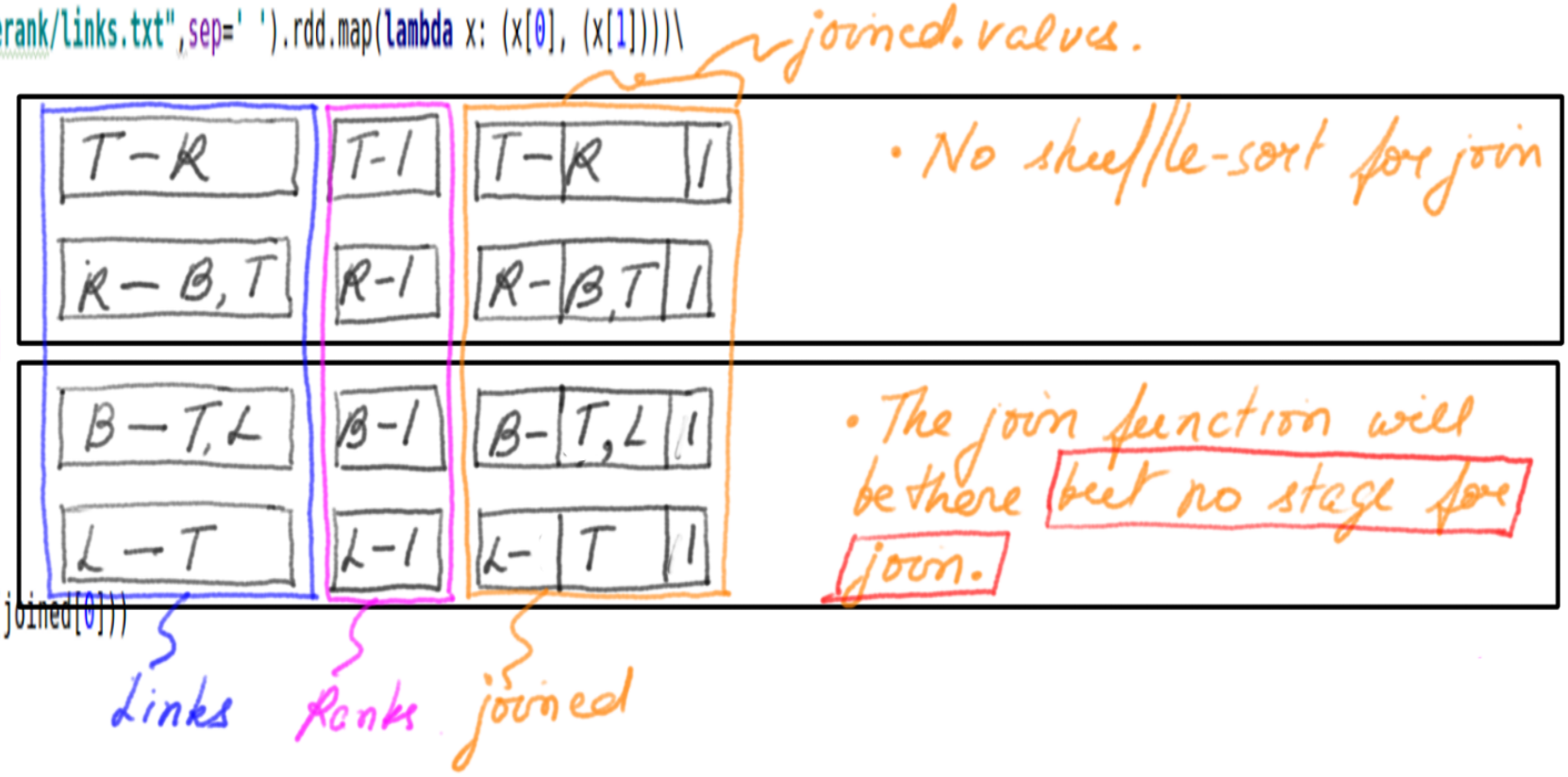
Links Ranks

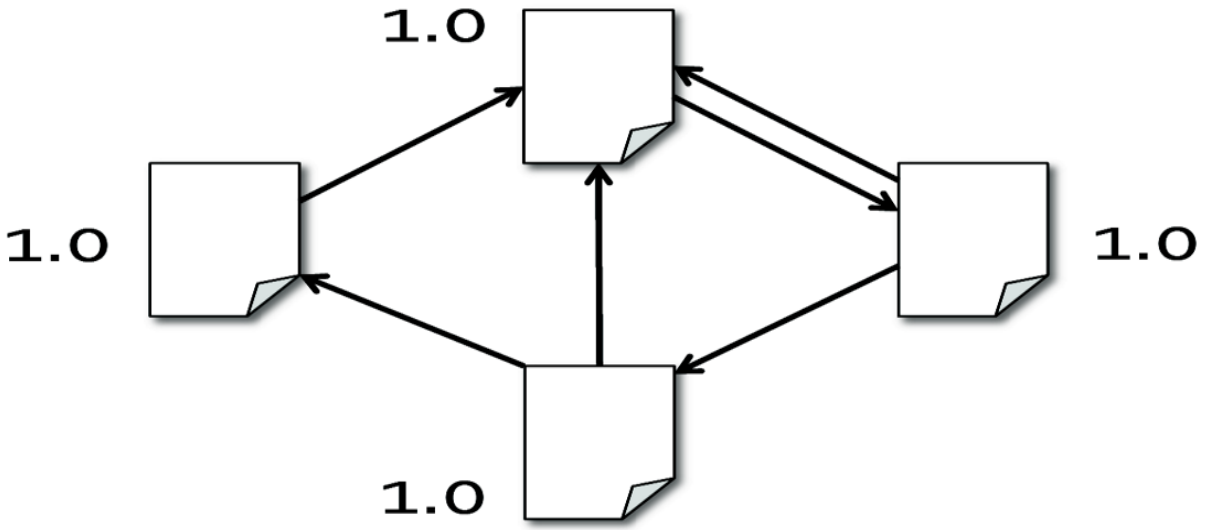
- Do you notice anything special??
- The 2 rdds are co-partioned and co-located.
- Notice the keys.



Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.mapValues(lambda x:1)

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```



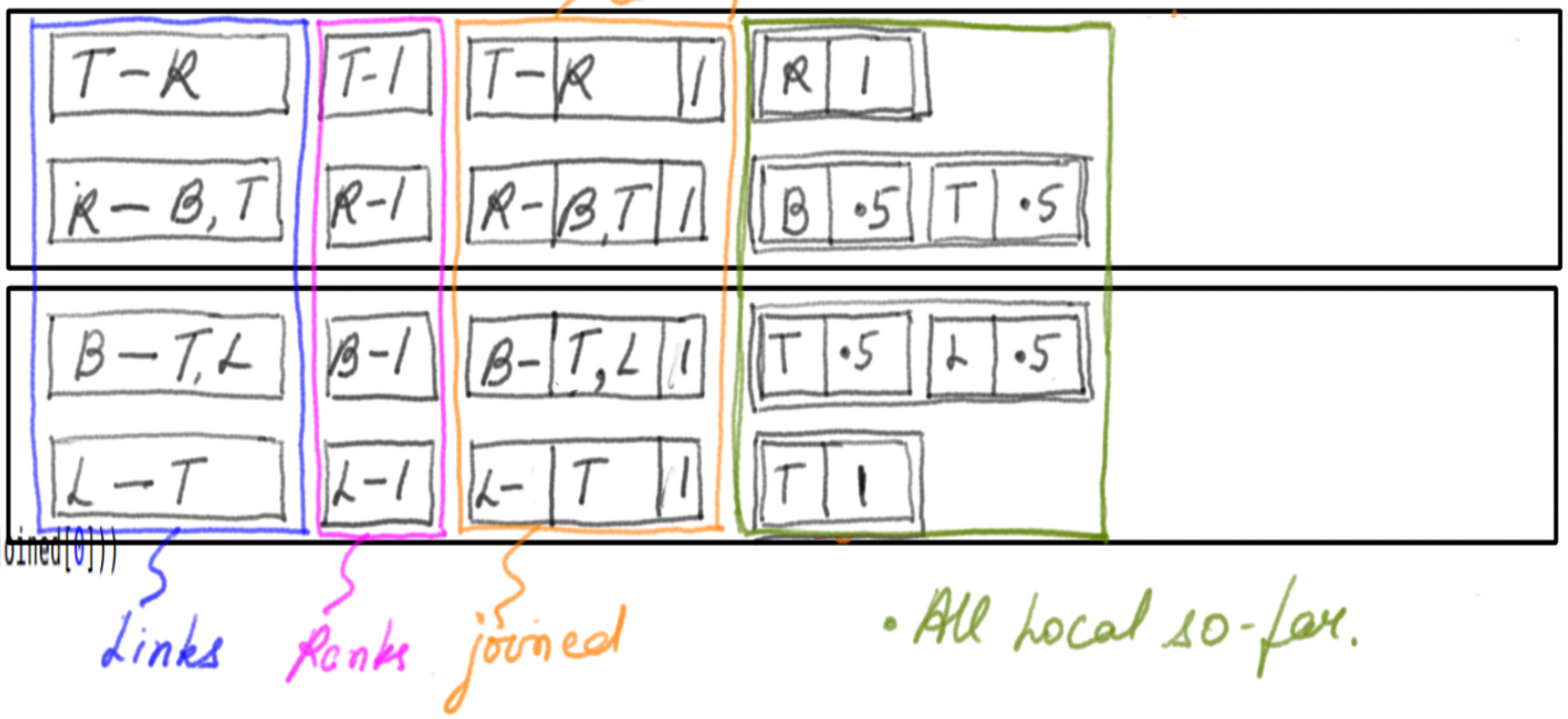


Spark Internals:Pagerank:Optimized

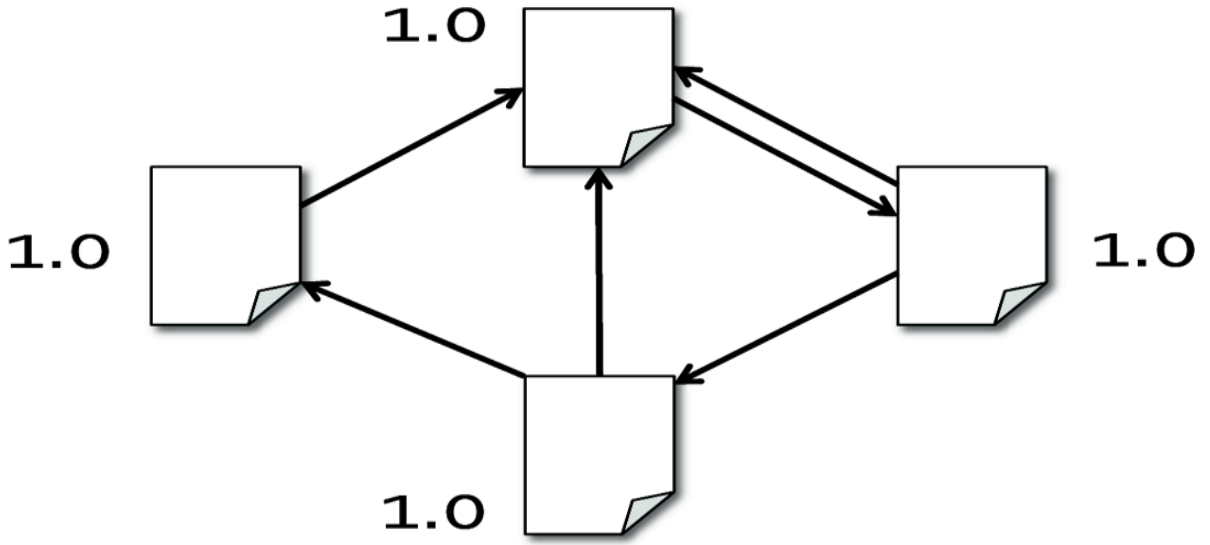
```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();
```

```
ranks=links.mapValues(lambda x:1)
```

```
def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result
```

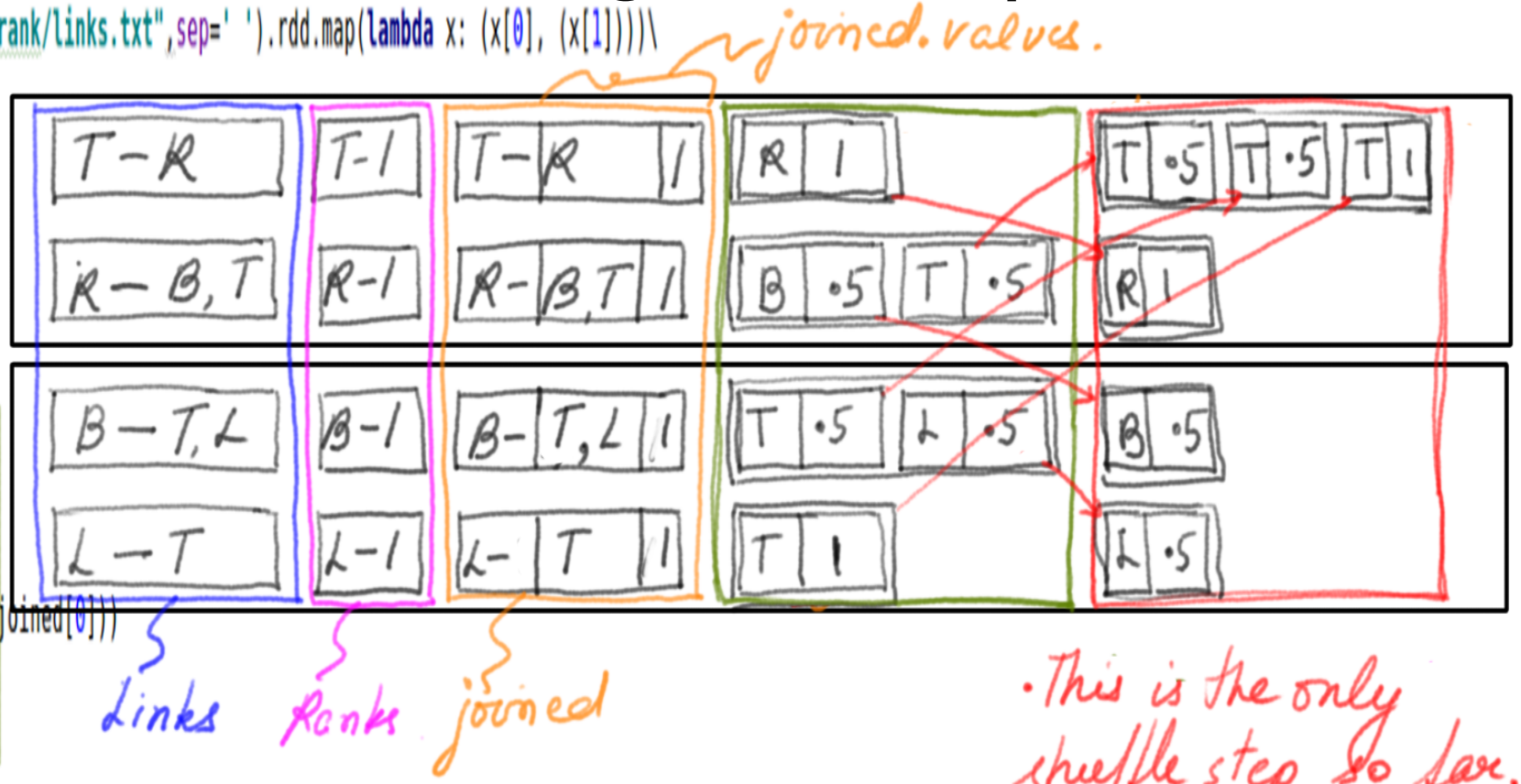



```
for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        .reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```

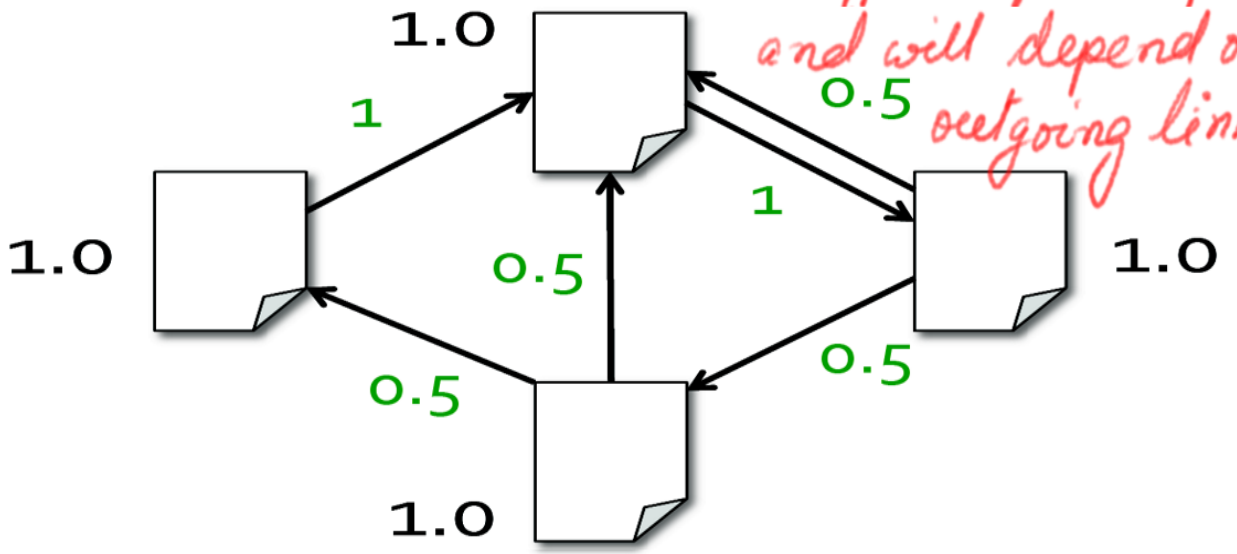


Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();

ranks=links.mapValues(lambda x:1)

def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result

for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        reduceByKey(add)\
        .mapValues(lambda x: 0.15 + x * 0.85)
```



- This is the only shuffle step so far, and will depend on outgoing links.



Spark Internals:Pagerank:Optimized

```
links = spark.read.csv("input/pagerank/links.txt",sep=' ').rdd.map(lambda x: (x[0], (x[1])))\
 .distinct()\
 .groupByKey()\
 .cache();
```

```
ranks=links.mapValues(lambda x:1)
```

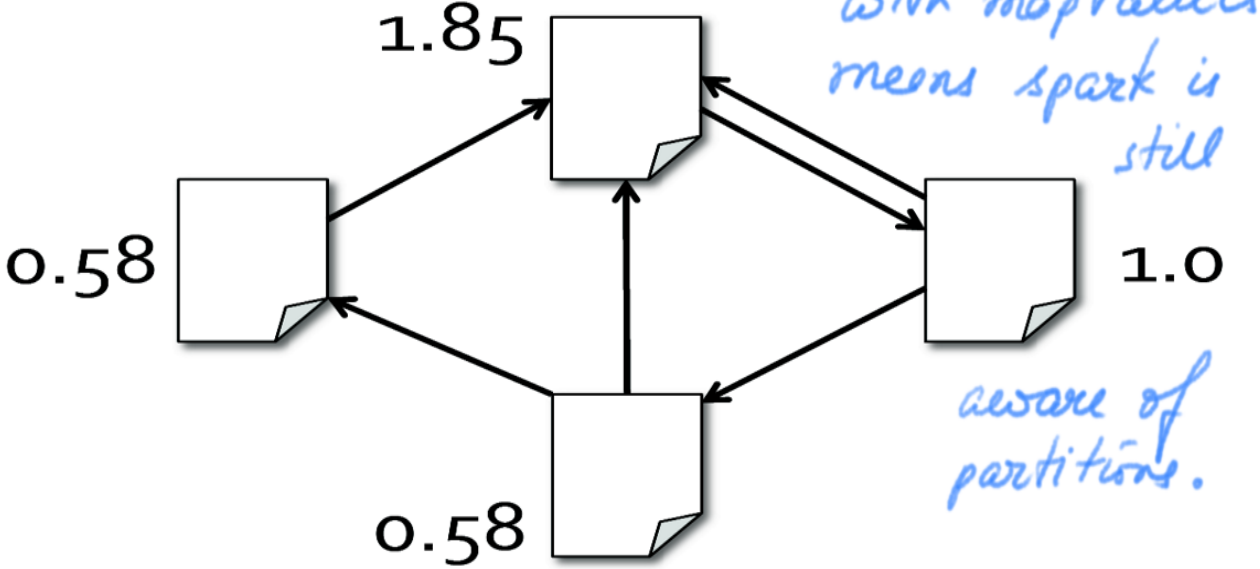
```
def processJoined(joined):
    result=[]
    for tolink in joined[0]:
        tup=tolink,(joined[1]/len(joined[0]))
        result.append(tup)
    return result
```

joined.values.

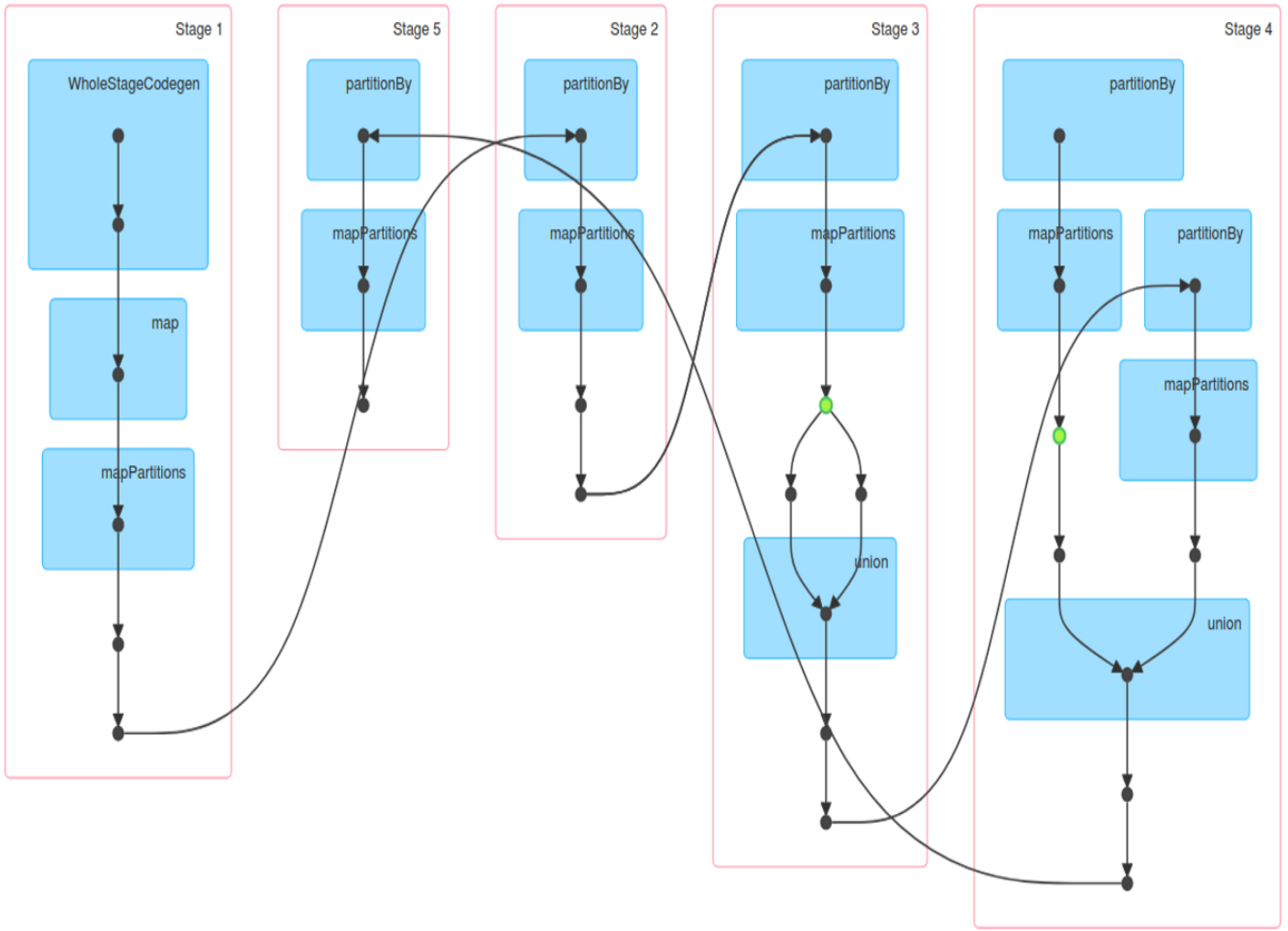
Links Ranks joined

Update ranks with mapValues means spark is still aware of partitions.

```
for i in range(num_iter) :
    #calculates URL contributions to the rank of other URLs.
    ranks = links.join(ranks).values()\
        .flatMap(lambda joined: processJoined(joined))\
        reduceByKey(add)\
        mapValues(lambda x: 0.15 + x * 0.85)
```



Spark Internals:Pagerank:Optimized

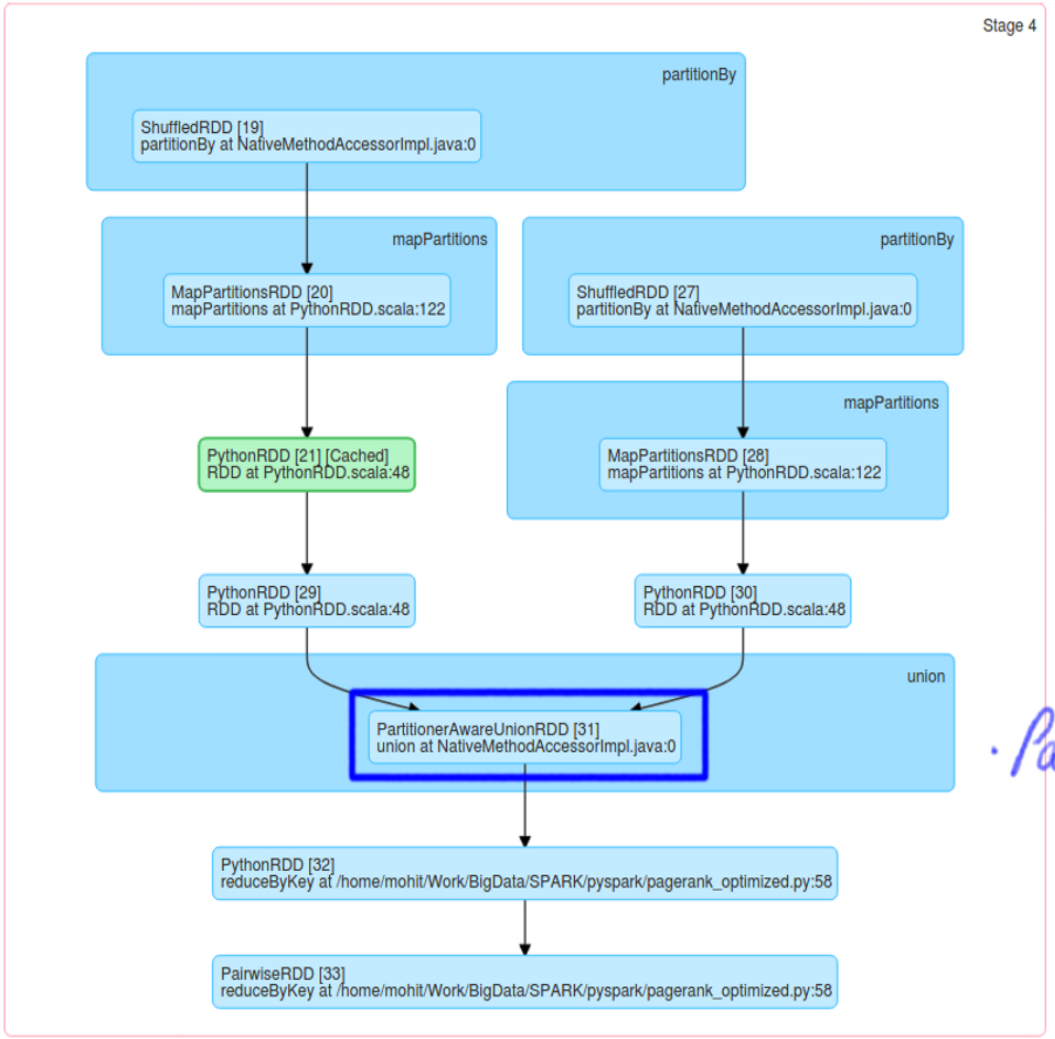


- No join stage, join function is there but not join stage as there is no shuffle.

Completed Stages (5)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	collect at /home/mohit/Work/BigData/SPARK/pyspark/pagerank_optimized.py:64	+details	2021/03/15 10:24:09	59 ms	1/1			169.0 B	
4	reduceByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank_optimized.py:58	+details	2021/03/15 10:24:09	0.1 s	1/1	310.0 B		163.0 B	169.0 B
3	reduceByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank_optimized.py:58	+details	2021/03/15 10:24:08	0.2 s	1/1	310.0 B		185.0 B	163.0 B
2	groupByKey at /home/mohit/Work/BigData/SPARK/pyspark/pagerank_optimized.py:33	+details	2021/03/15 10:24:08	69 ms	1/1			209.0 B	185.0 B
1	distinct at /home/mohit/Work/BigData/SPARK/pyspark/pagerank_optimized.py:33	+details	2021/03/15 10:24:08	0.4 s	1/1				209.0 B

Spark Internals:Pagerank:Optimized



· Partition aware UnionRDD.

▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.1 s	0.1 s	0.1 s	0.1 s	0.1 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	310.0 B / 3	310.0 B / 3	310.0 B / 3	310.0 B / 3	310.0 B / 3
Shuffle Read Size / Records	163.0 B / 1	163.0 B / 1	163.0 B / 1	163.0 B / 1	163.0 B / 1
Shuffle Write Size / Records	169.0 B / 1	169.0 B / 1	169.0 B / 1	169.0 B / 1	169.0 B / 1