
Common Patterns in Spark Data Processing

Prof. Hyuk-Yoon Kwon

Common Patterns in Spark Programming

In this chapter you will learn

- What kinds of processing and analysis Spark is best at
- How to implement an iterative algorithm in Spark

Common Spark Use Cases (1)

- **Spark is especially useful when working with any combination of:**
 - Large amounts of data
 - Distributed storage
 - Intensive computations
 - Distributed computing
 - Iterative algorithms
 - In-memory processing and pipelining

Common Spark Use Cases (2)

- Examples

- Risk analysis
 - “How likely is this borrower to pay back a loan?”
 - Recommendations
 - “Which products will this customer enjoy?”
 - Predictions
 - “How can we prevent service outages instead of simply reacting to them?”
 - Classification
 - “How can we tell which mail is spam and which is legitimate?”

Spark Examples

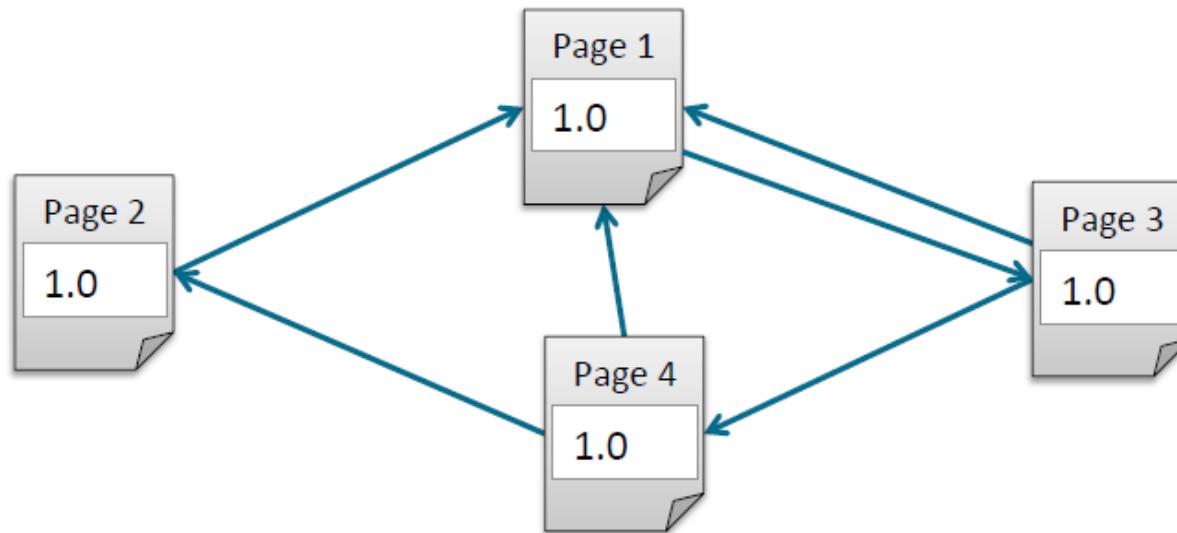
- Spark includes many example programs that demonstrate some common Spark programming patterns and algorithms
 - k-means
 - Logistic regression
 - Calculate pi
 - Alternating least squares (ALS)
 - Querying Apache web logs
 - Processing Twitter feeds
- Examples
 - `$SPARK_HOME/examples/lib`
 - `spark-examples-version.jar` – Java and Scala examples
 - `python.tar.gz` – Pyspark examples

Example: PageRank

- **PageRank gives web pages a ranking score based on links from other pages**
 - Higher scores given for more links, and links from other high ranking pages
- **Why do we care?**
 - PageRank is a classic example of big data analysis (like WordCount)
 - Lots of data – needs an algorithm that is distributable and scalable
 - Iterative – the more iterations, the better than answer

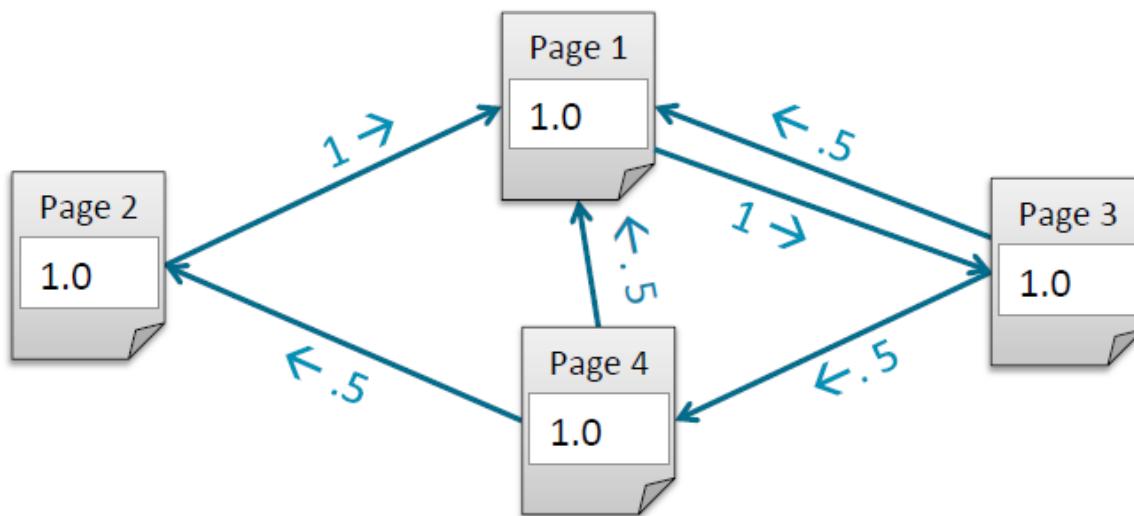
PageRank Algorithm (1)

1. Start each page with a rank of 1.0



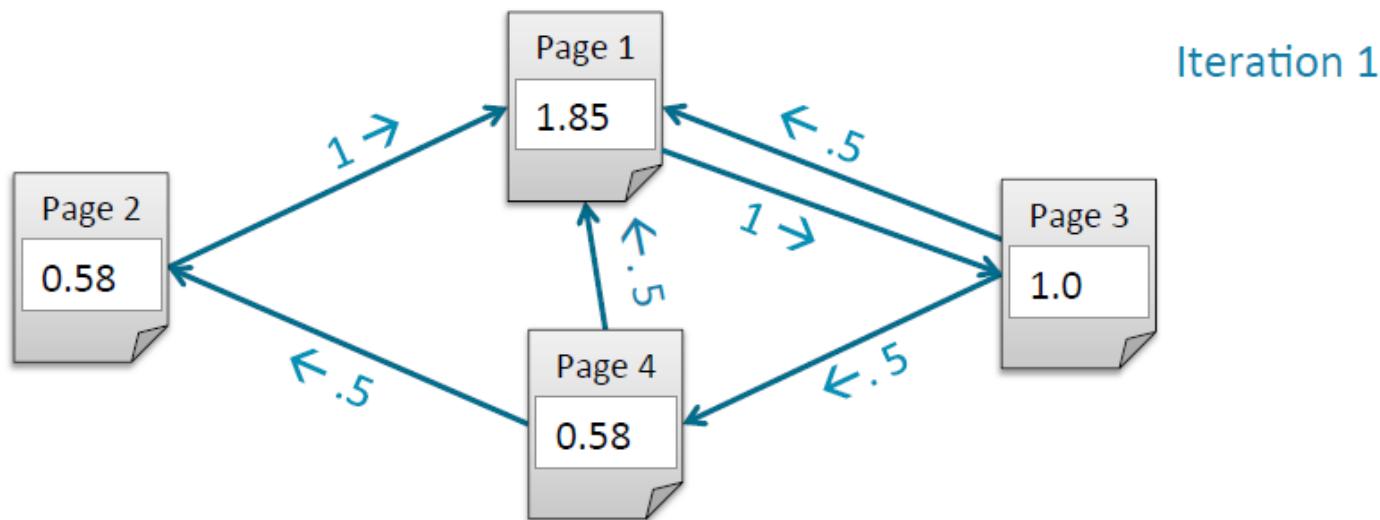
PageRank Algorithm (2)

1. Start each page with a rank of 1.0
2. On each iteration:
 1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $\text{contrib}_p = \text{rank}_p / \text{neighbors}_p$



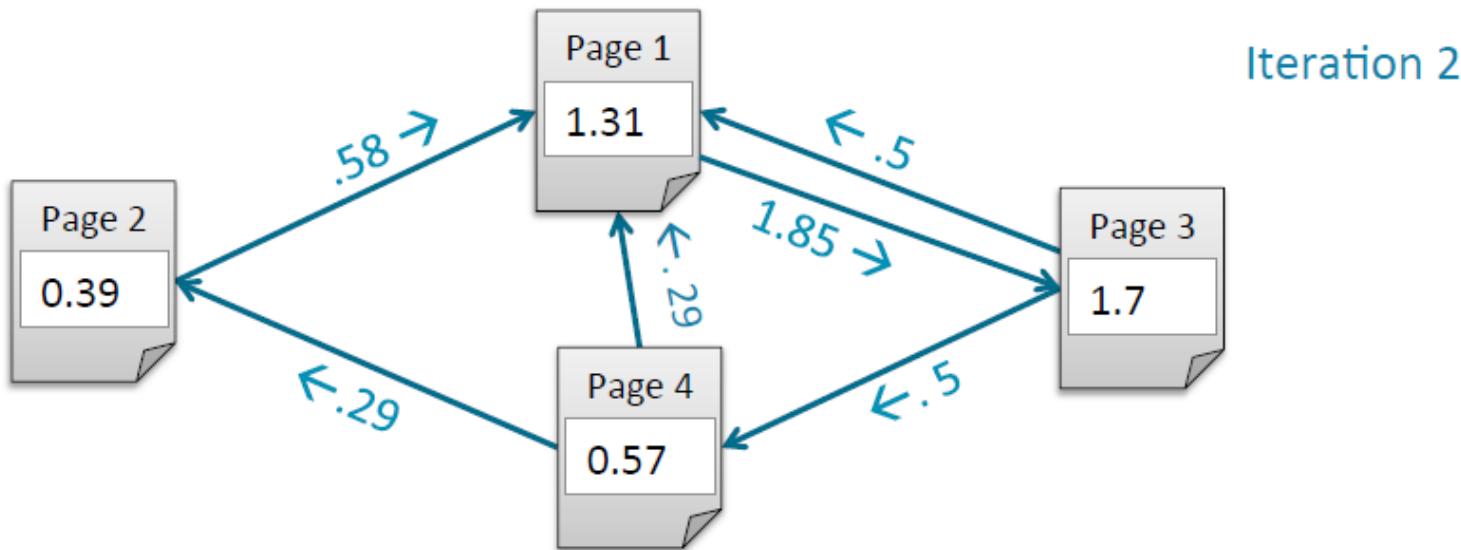
PageRank Algorithm (3)

1. Start each page with a rank of 1.0
2. On each iteration:
 1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $\text{contrib}_p = \text{rank}_p / \text{neighbors}_p$
 2. Set each page's new rank based on the sum of its neighbors contribution: $\text{new-rank} = \sum \text{contribs} * .85 + .15$



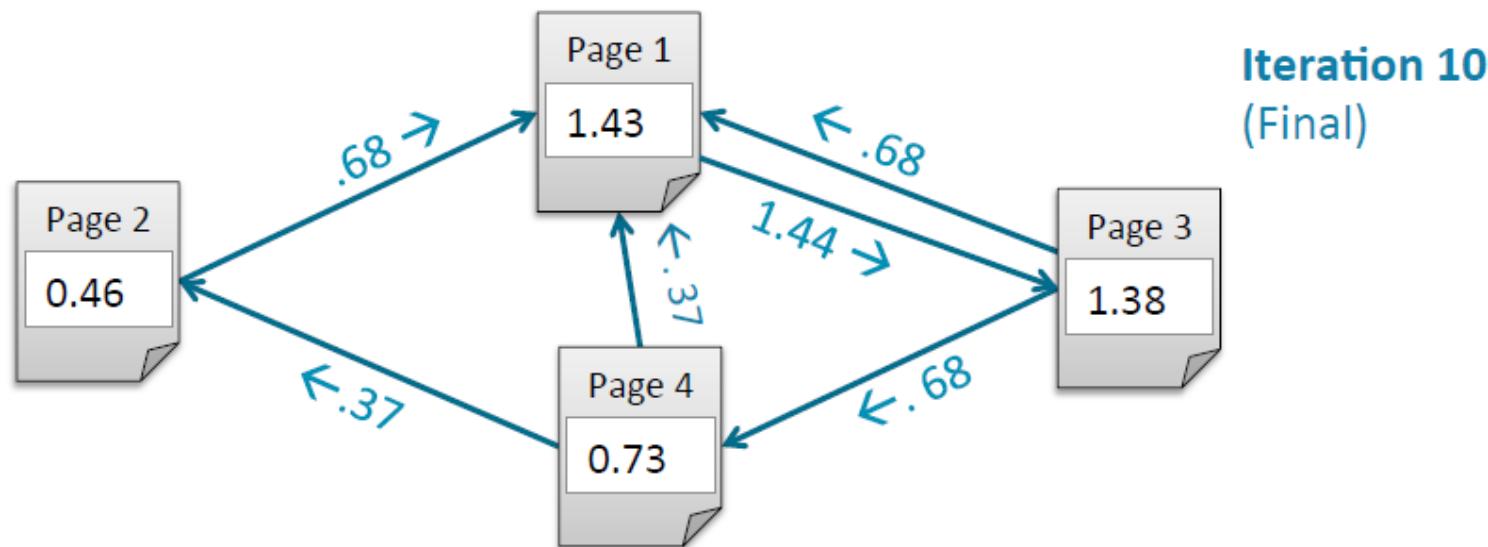
PageRank Algorithm (4)

1. Start each page with a rank of 1.0
2. On each iteration:
 1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $\text{contrib}_p = \text{rank}_p / \text{neighbors}_p$
 2. Set each page's new rank based on the sum of its neighbors contribution: $\text{new-rank} = \sum \text{contris} * .85 + .15$
3. Each iteration incrementally improves the page ranking



PageRank Algorithm (5)

1. Start each page with a rank of 1.0
2. On each iteration:
 1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $\text{contrib}_p = \text{rank}_p / \text{neighbors}_p$
 2. Set each page's new rank based on the sum of its neighbors contribution: $\text{new-rank} = \sum \text{contris} * .85 + .15$
3. Each iteration incrementally improves the page ranking



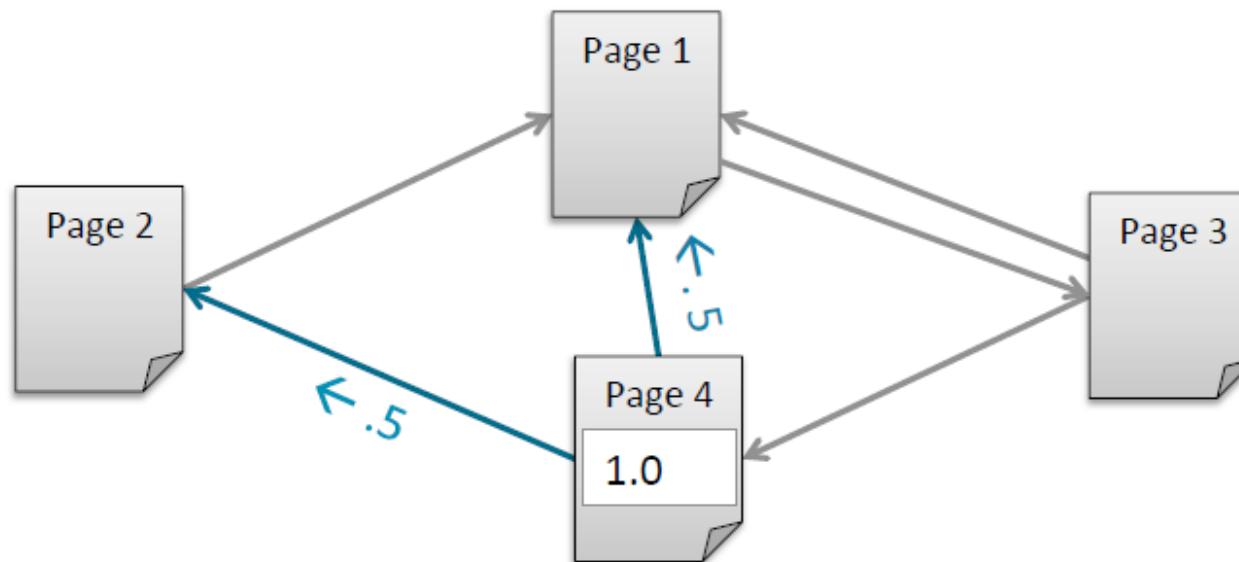
PageRank in Spark: Neighbor Contribution Function

```
def computeContribs(neighbors, rank):
    for neighbor in neighbors: yield(neighbor, rank/len(neighbors))
```

neighbors: [page1,page2]
rank: 1.0



(page1,.5)
(page2,.5)



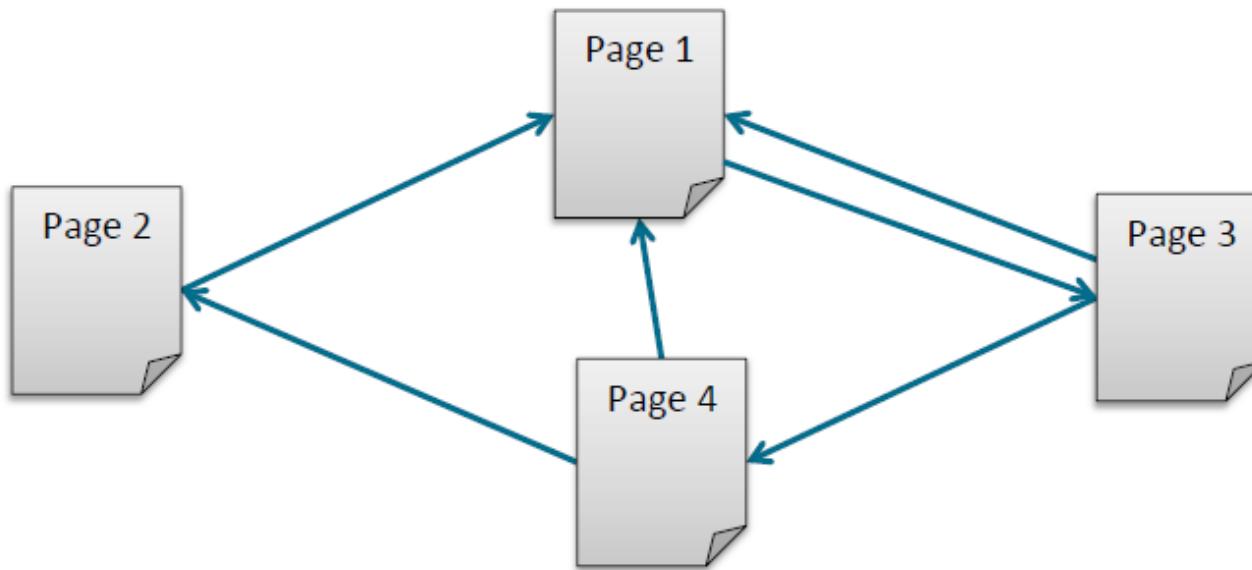
PageRank in Spark: Example Data

Data Format:

source-page destination-page

...

page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4



PageRank in Spark: Pairs of Page Links

```
def computeContribs(neighbors, rank):...  
  
links = sc.textFile(file) \  
.map(lambda line: line.split()) \  
.map(lambda pages: (pages[0], pages[1])) \  
.distinct()
```

page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4

(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)

PageRank in Spark: Page Links Grouped by Source Page

```
def computeContribs(neighbors, rank):...  
  
links = sc.textFile(file)\n    .map(lambda line: line.split())\n    .map(lambda pages: (pages[0], pages[1]))\n    .distinct()\n    .groupByKey()
```

page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4

(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)

links
(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

PageRank in Spark: Persisting the Link Pair RDD

```
def computeContribs(neighbors, rank):...  
  
links = sc.textFile(file) \  
.map(lambda line: line.split()) \  
.map(lambda pages: (pages[0], pages[1])) \  
.distinct() \  
.groupByKey() \  
.persist()
```

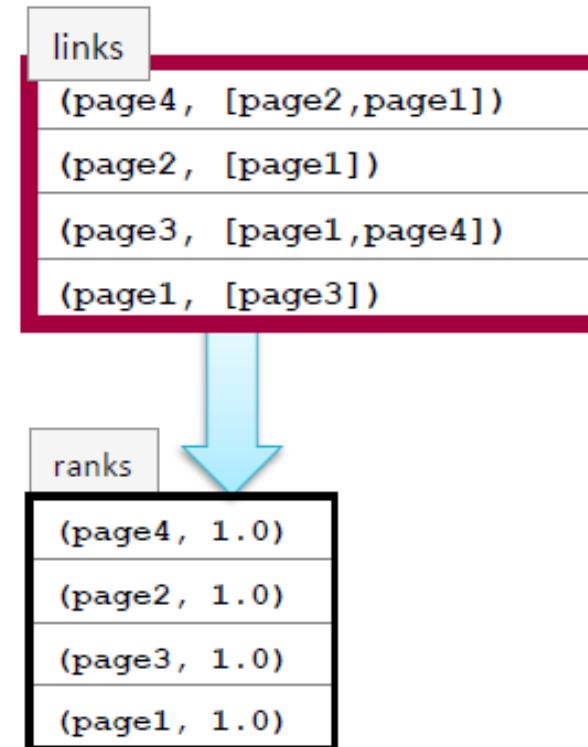
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4

(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)

links
(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

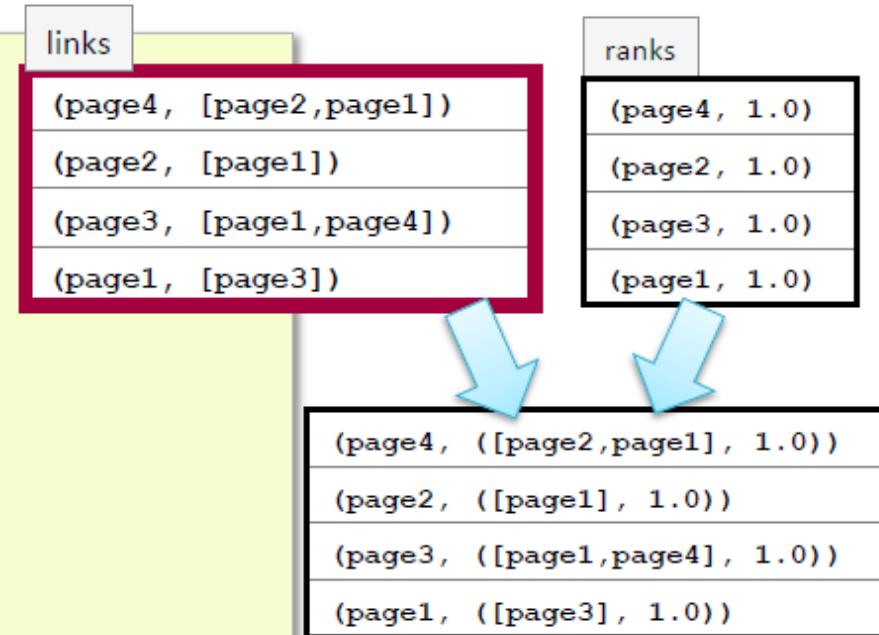
PageRank in Spark: Set Initial Ranks

```
def computeContribs(neighbors, rank):...  
  
links = sc.textFile(file) \  
.map(lambda line: line.split()) \  
.map(lambda pages: (pages[0], pages[1])) \  
.distinct() \  
.groupByKey() \  
.persist()  
  
ranks=links.map(lambda (page,neighbors): (page,1.0))
```



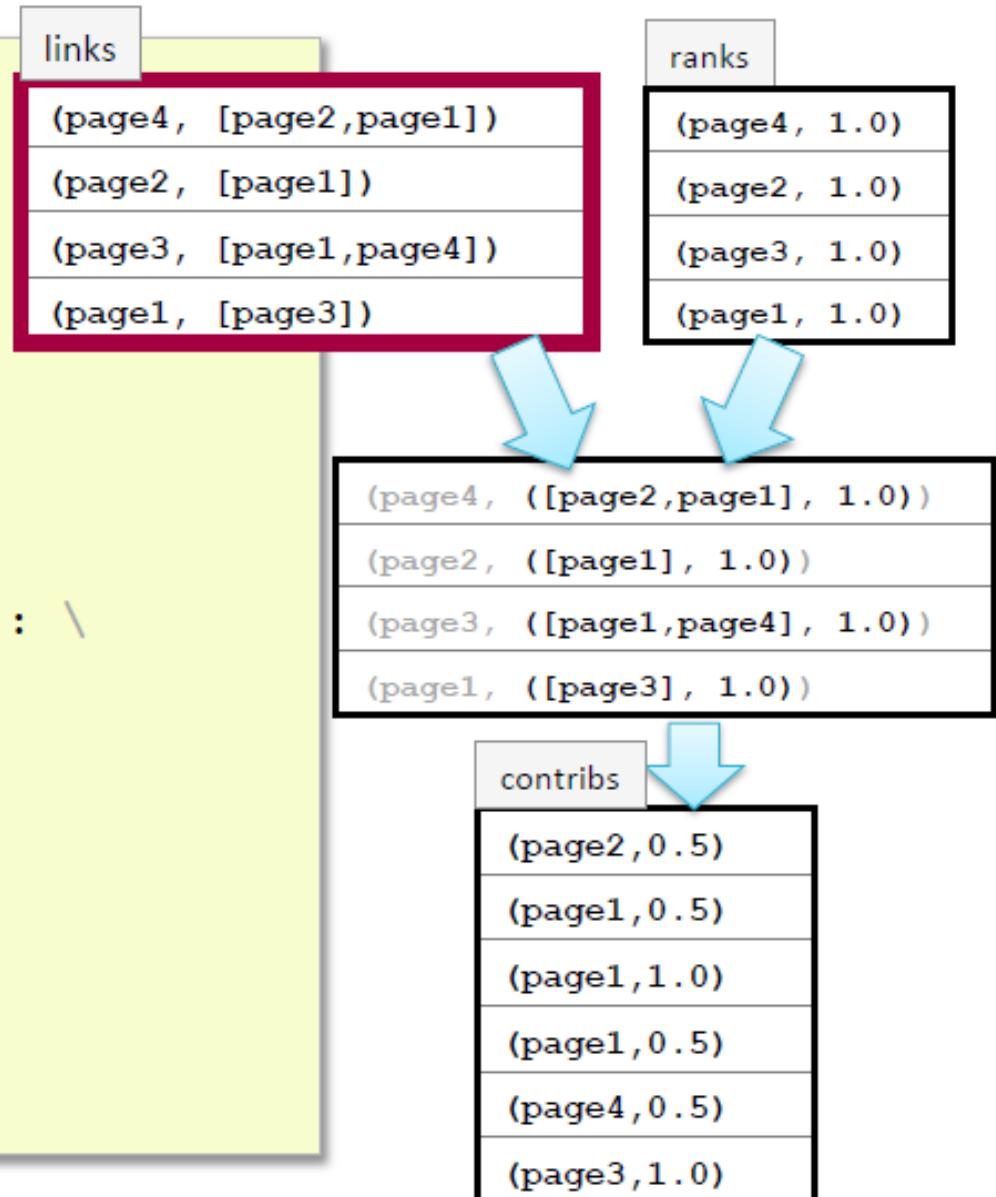
PageRank in Spark: First Iteration (1)

```
def computeContribs(neighbors, rank):...  
  
links = ...  
  
ranks = ...  
  
for x in xrange(10):  
    contribs=links\  
        .join(ranks)
```



PageRank in Spark: First Iteration (2)

```
def computeContribs(neighbors, rank):...  
  
links = ...  
  
ranks = ...  
  
for x in xrange(10):  
    contribs=links\  
        .join(ranks)\  
        .flatMap(lambda (page, (neighbors,rank)): \  
            computeContribs(neighbors,rank))
```



PageRank in Spark: First Iteration (3)

```
def computeContribs(neighbors, rank):...  
  
links = ...  
  
ranks = ...  
  
for x in xrange(10):  
    contribs=links\  
        .join(ranks)\  
        .flatMap(lambda (page, (neighbors,rank)): \  
            computeContribs(neighbors,rank))  
    ranks=contribs\  
        .reduceByKey(lambda v1,v2: v1+v2)
```

contribs
(page2,0.5)
(page1,0.5)
(page1,1.0)
(page1,0.5)
(page4,0.5)
(page3,1.0)

↓

(page4, 0.5)
(page2, 0.5)
(page3, 1.0)
(page1, 2.0)

PageRank in Spark: First Iteration (4)

```
def computeContribs(neighbors, rank):...  
  
links = ...  
  
ranks = ...  
  
for x in xrange(10):  
    contribs=links\  
        .join(ranks)\  
        .flatMap(lambda (page, (neighbors, rank)): \  
            computeContribs(neighbors, rank))  
    ranks=contribs\  
        .reduceByKey(lambda v1,v2: v1+v2)\  
        .map(lambda (page,contrib): \  
            (page,contrib * 0.85 + 0.15))
```

contribs
(page2, 0.5)
(page1, 0.5)
(page1, 1.0)
(page1, 0.5)
(page4, 0.5)
(page3, 1.0)



(page4, 0.5)
(page2, 0.5)
(page3, 1.0)
(page1, 2.0)

ranks
(page4, .58)
(page2, .58)
(page3, 1.0)
(page1, 1.85)

PageRank in Spark: Second Iteration

```
def computeContribs(neighbors, rank):...  
  
links = ...  
  
ranks = ...  
  
for x in xrange(10):  
    contribs=links\  
        .join(ranks)\  
        .flatMap(lambda (page,(neighbors,rank)): \  
            computeContribs(neighbors,rank))  
    ranks=contribs\  
        .reduceByKey(lambda v1,v2: v1+v2)\  
        .map(lambda (page,contrib): \  
            (page,contrib * 0.85 + 0.15))  
  
for rank in ranks.collect(): print rank
```

