Class 8: Spark Algorithms

New York University

Summer 2017

Agenda

1. Common Spark Use Cases
2. Iterative Algorithms in Spark
3. Graph Processing and Analysis
4. Machine Learning

- **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

- **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?"

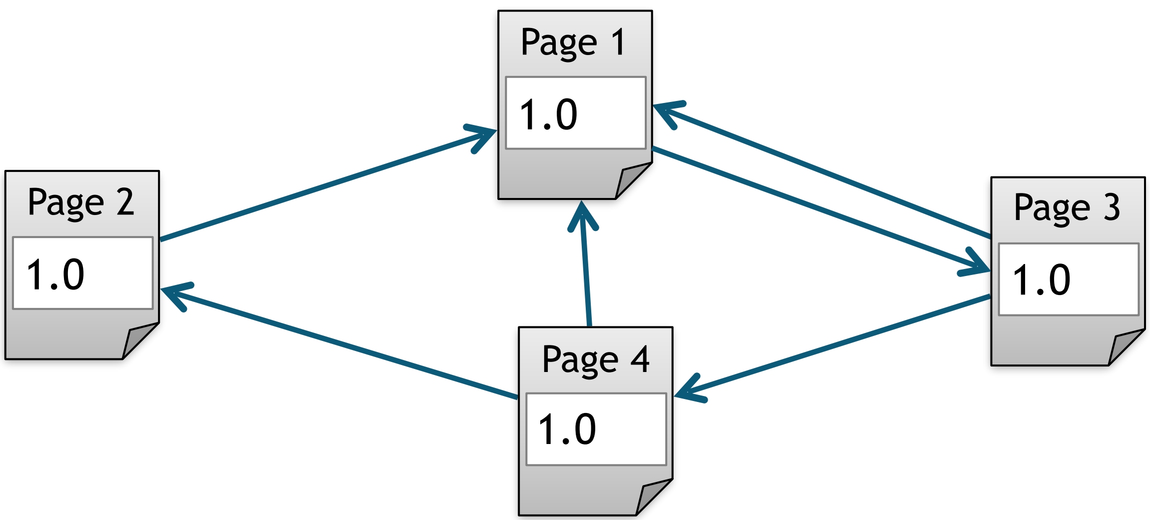
- **Spark includes many example programs that demonstrate some common Spark programming patterns and algorithms**
  - k-means
  - Logistic regression
  - Calculate pi
  - Alternating least squares (ALS)
  - Querying Apache web logs
  - Processing Twitter feeds
- **Examples**
  - **`$SPARK_HOME/examples/lib`**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

Agenda

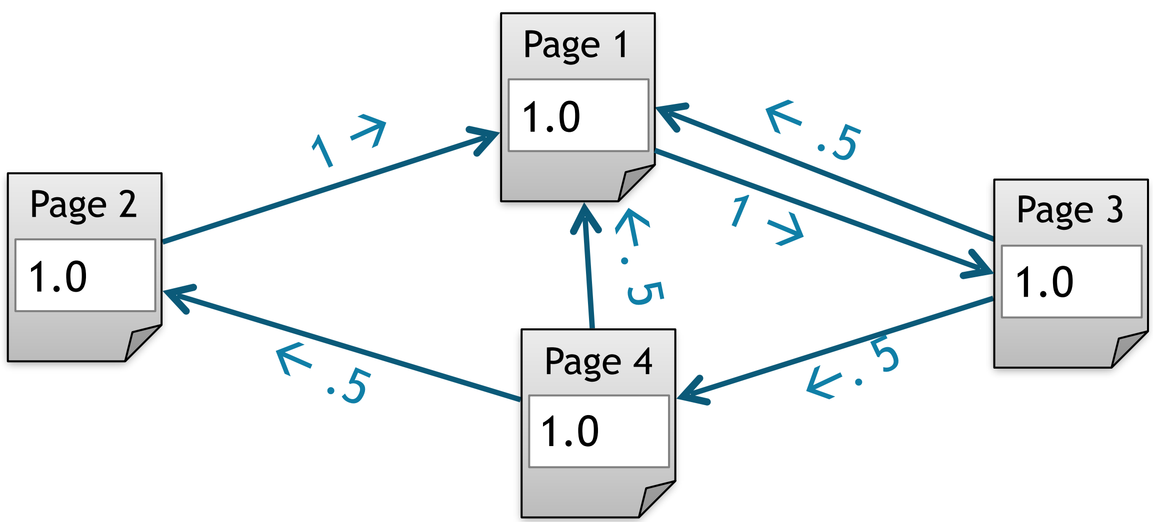
1. Common Spark Use Cases
2. Iterative Algorithms in Spark
3. Graph Processing and Analysis
4. Machine Learning

- **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 the answer

1. Start each page with a rank of 1.0

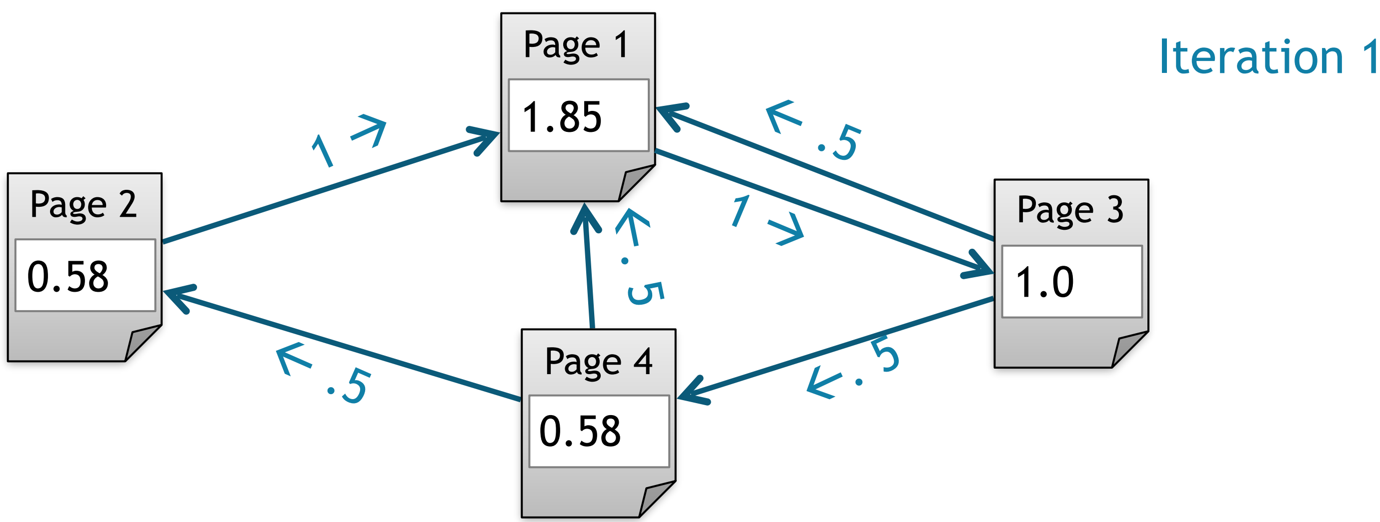


1. **Start each page with a rank of 1.0**
2. **On each iteration:**
   1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p \,/\, neighbors_p$

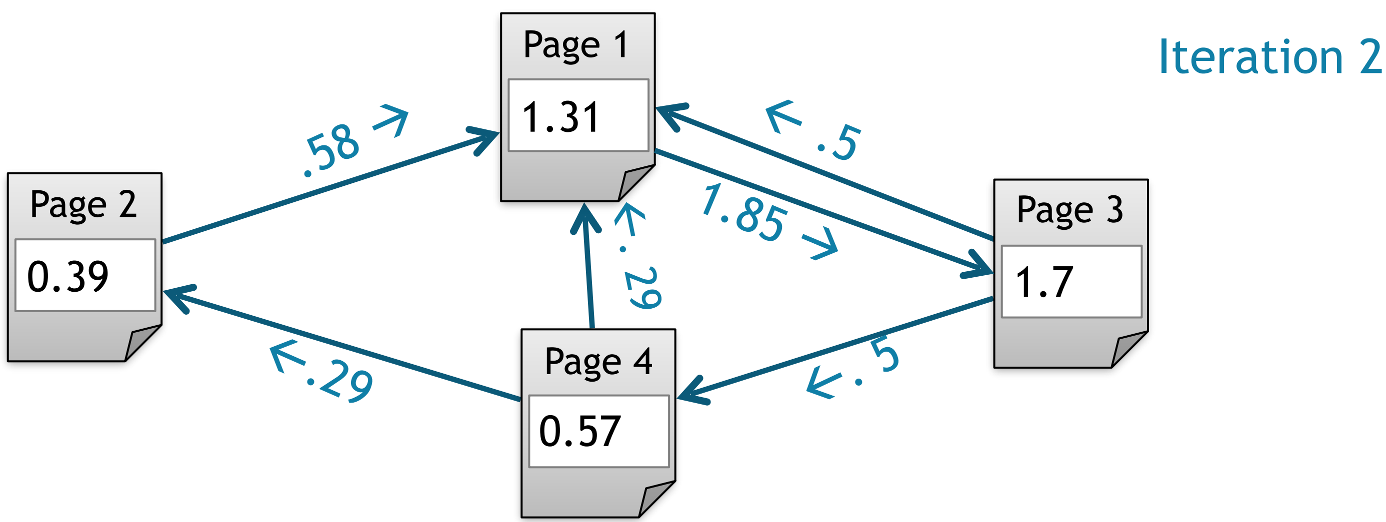


1. **Start each page with a rank of 1.0**

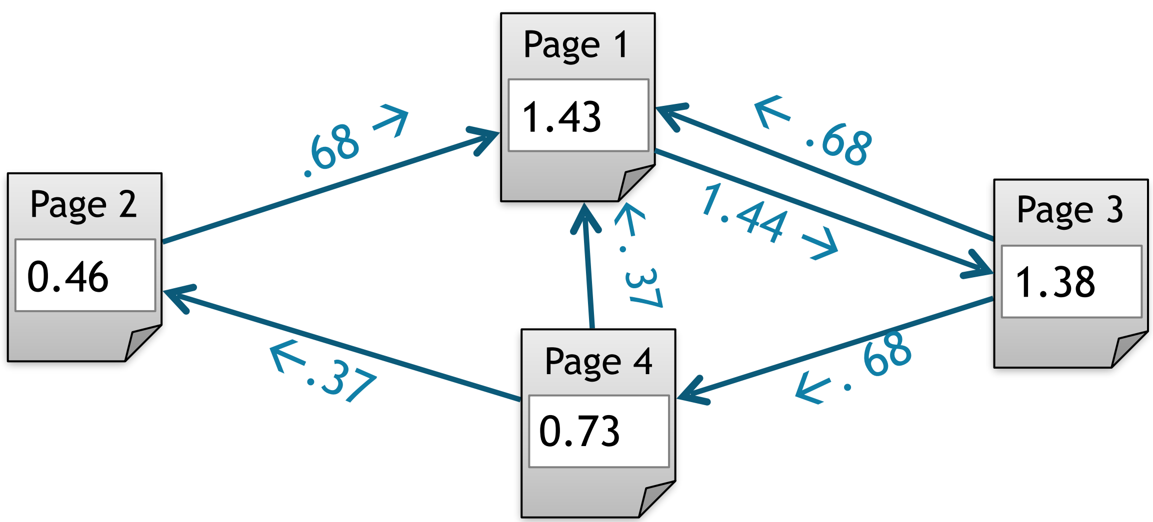
2. **On each iteration:**
   1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p \;/\; neighbors_p$
   2. Set each page's new rank based on the sum of its neighbors contribution: $\text{new-rank} = \Sigma\text{contribs} * .85 + .15$



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: $contrib_p = rank_p / neighbors_p$
   2. Set each page's new rank based on the sum of its neighbors contribution: new-rank = Σcontribs * .85 + .15
3. **Each iteration incrementally improves the page ranking**



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: $contrib_p = rank_p / neighbors_p$
   2. Set each page's new rank based on the sum of its neighbors contribution: $new\text{-}rank = \Sigma contribs * .85 + .15$
3. **Each iteration incrementally improves the page ranking**

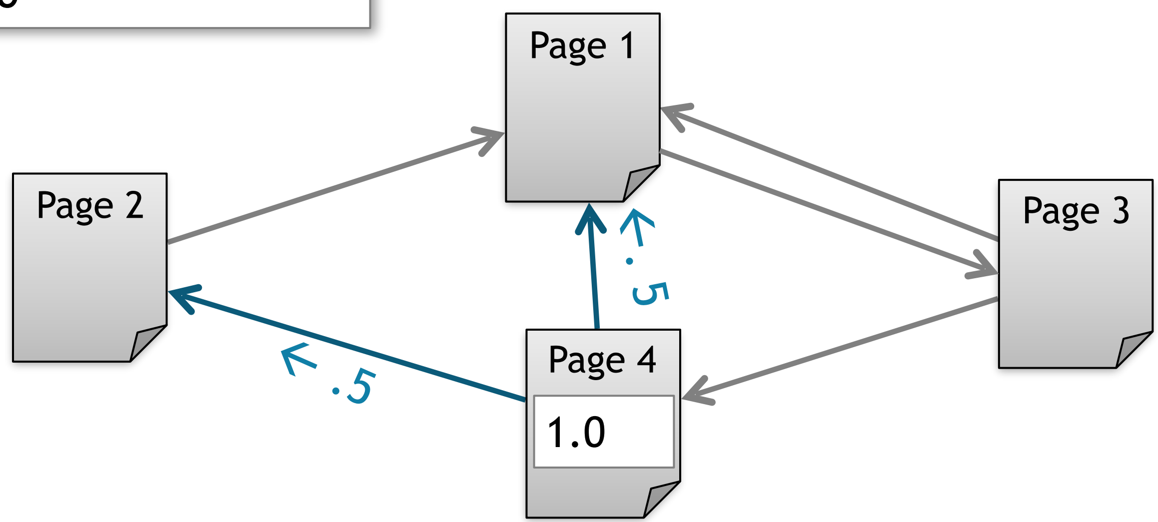


Iteration 10
(Final)

```
Define a function, computeContribs(neighbors, rank)
    For each neighbor, output: (neighbor, rank/len(neighbors))
```

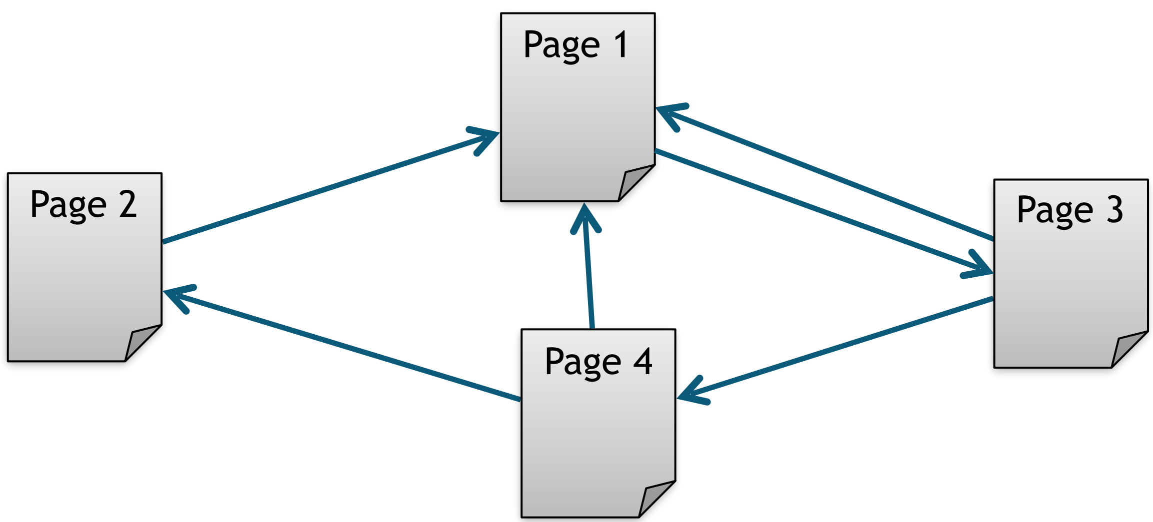
neighbors:
[page1,page2]
rank: 1.0

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



Data Format:
source-page destination-page
...

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



```
Define a function, computeContribs(neighbors, rank)
    For each neighbor, output: (neighbor, rank/len(neighbors)

Read in the file and use map and distinct to generate tuples.
```

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

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

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

Define a function, computeContribs(neighbors, rank)
    For each neighbor, output: (neighbor, rank/len(neighbors)

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

Read in the file and use map and distinct to generate tuples.

Use groupByKey to group pages related to a given page.

links

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

```
Define a function, computeContribs(neighbors, rank)
    For each neighbor, output: (neighbor, rank/len(neighbors)

Read in the file and use map and distinct to generate tuples.

Use groupByKey() to group pages related to a given page.

Use persist() to persist the cache the links information.
```

```
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])

```
Define a function, computeContribs(neighbors, rank)
    For each neighbor, output: (neighbor, rank/len(neighbors))

Read in the file and use map and distinct to generate tuples.

Use groupByKey() to group pages related to a given page.

Use persist() to persist the cache the links information.

Create an RDD, ranks, that initializes rank to 1.0 for each page.
```

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

ranks
(page4, 1.0)
(page2, 1.0)
(page3, 1.0)
(page1, 1.0)

`Use join() to join links and rank data.`

links

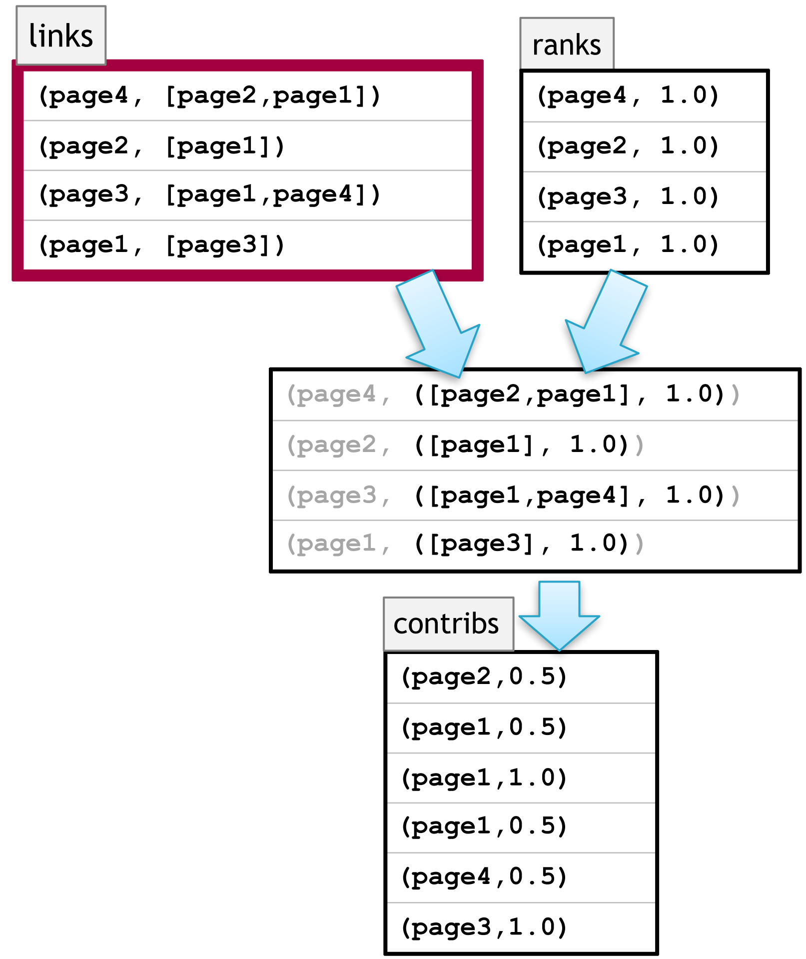
```
(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])
```

ranks

```
(page4, 1.0)
(page2, 1.0)
(page3, 1.0)
(page1, 1.0)
```

```
(page4, ([page2,page1], 1.0))
(page2, ([page1], 1.0))
(page3, ([page1,page4], 1.0))
(page1, ([page3], 1.0))
```

`After join(), use flatMap and call computeContribs to compute the contributions.`



Use reduceByKey to produce the results by summing partial ranks.

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)

`After reduceByKey, compute the rank by multiplying by .85 and adding .15`

contribs

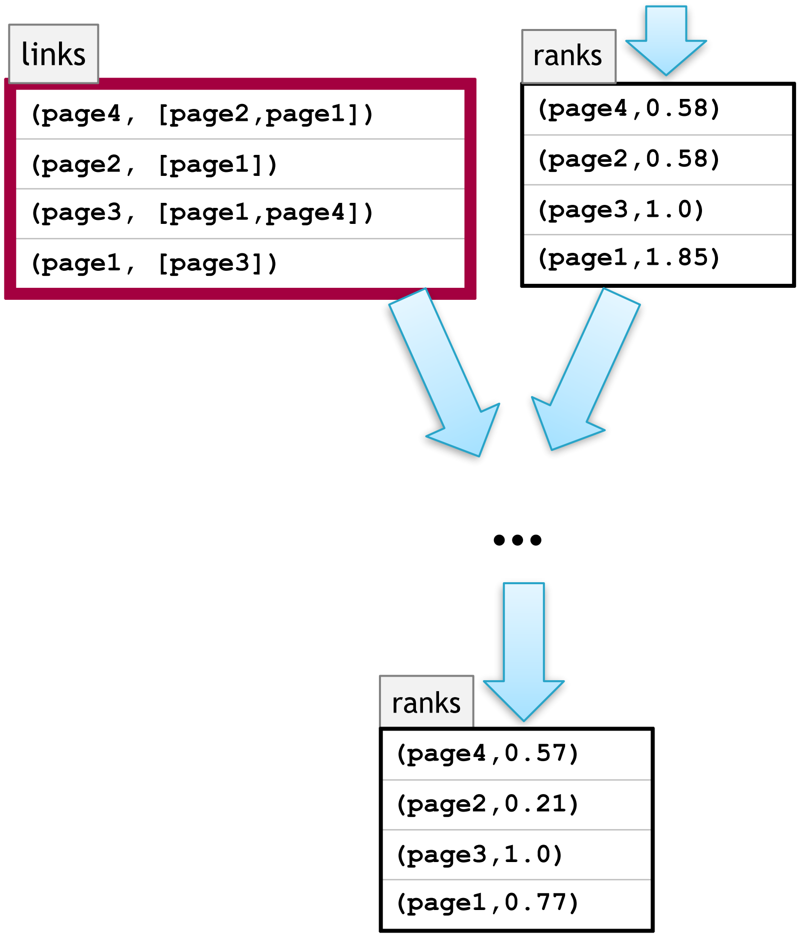
(page2,0.5)
(page1,0.5)
(page1,1.0)
(page1,0.5)
(page4,0.5)
(page3,1.0)

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

ranks

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

Finally, collect() the results.



Agenda

1. Common Spark Use Cases
2. Iterative Algorithms in Spark
3. Graph Processing and Analysis
4. Machine Learning

- **Many data analytics problems work with "data parallel" algorithms**
  - Records can be processed independently of each other
  - Very well suited to parallelizing
- **Some problems focus on the relationships between the individual data items. For example:**
  - Social networks
  - Web page hyperlinks
  - Roadmaps
- **These relationships can be represented by graphs**
  - Requires "graph parallel" algorithms

- **Graph Creation**
  - Extracting relationship information from a data source
    - For example, extracting links from web pages
- **Graph Representation**
  - e.g., adjacency lists in a table
- **Graph Analysis**
  - Inherently iterative, hard to parallelize
- **Post-analysis processing**
  - e.g., incorporating product recommendations into a retail site

- **Spark is very well suited to graph parallel algorithms**
- **Apache GraphX**
  - UC Berkeley AMPLab project on top of Spark
  - Unifies optimized graph computation with Spark's fast data parallelism and interactive abilities
  - Supersedes predecessor Bagel (Pregel on Spark)

Agenda

1. Common Spark Use Cases
2. Iterative Algorithms in Spark
3. Graph Processing and Analysis
4. Machine Learning

- **Most programs tell computers exactly what to do**
  - Database transactions and queries
  - Controllers
    - Phone systems, manufacturing processes, transport, weaponry, etc.
  - Media delivery
  - Simple search
  - Social systems
    - Chat, blogs, email, etc.
- **An alternative technique is to have computers *learn* what to do**
- **Machine Learning programs leverage collected data to drive future program behavior**

- **Machine Learning is an active area of research and new applications**
- **There are three well-established categories of techniques for exploiting data**
  - Collaborative filtering (recommendations)
  - Clustering
  - Classification
- **Highly computation intensive and iterative**

- **Collaborative Filtering is a technique for recommendations**
- **Example application: given people who each like certain books, learn to suggest what someone may like in the future based on what they already like**
- **Helps users navigate data by expanding to topics that have affinity with their established interests**
- **Collaborative Filtering algorithms are agnostic to the different types of data items involved**
  - Useful in many different domains

- **Clustering algorithms discover structure in collections of data**
  - Where no formal structure previously existed
- **They discover what clusters, or groupings, naturally occur in data**
- **Examples**
  - Finding related news articles
  - Computer vision

- **The previous two techniques are considered *unsupervised* learning**
  - The algorithm discovers groups or recommendations itself
- **Classification is a form of *supervised* learning**
- **A classification system accepts as input a set of records with labels**
  - Learns how to label new records based on that information
- **Examples**
  - Given a set of e-mails identified as spam/not spam, label new e-mails as spam/not spam
  - Given tumors identified as benign or malignant, classify new tumors

- **MLlib is part of Apache Spark**
- **Includes many common ML functions**
  - ALS (alternating least squares)
  - k-means
  - Logistic Regression
  - Linear Regression
  - Gradient Descent

Homework

See the homework packet for details.