UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

CS 431/631 451/651 (Fall 2019)

Part 4: Analyzing Graphs (2/2)

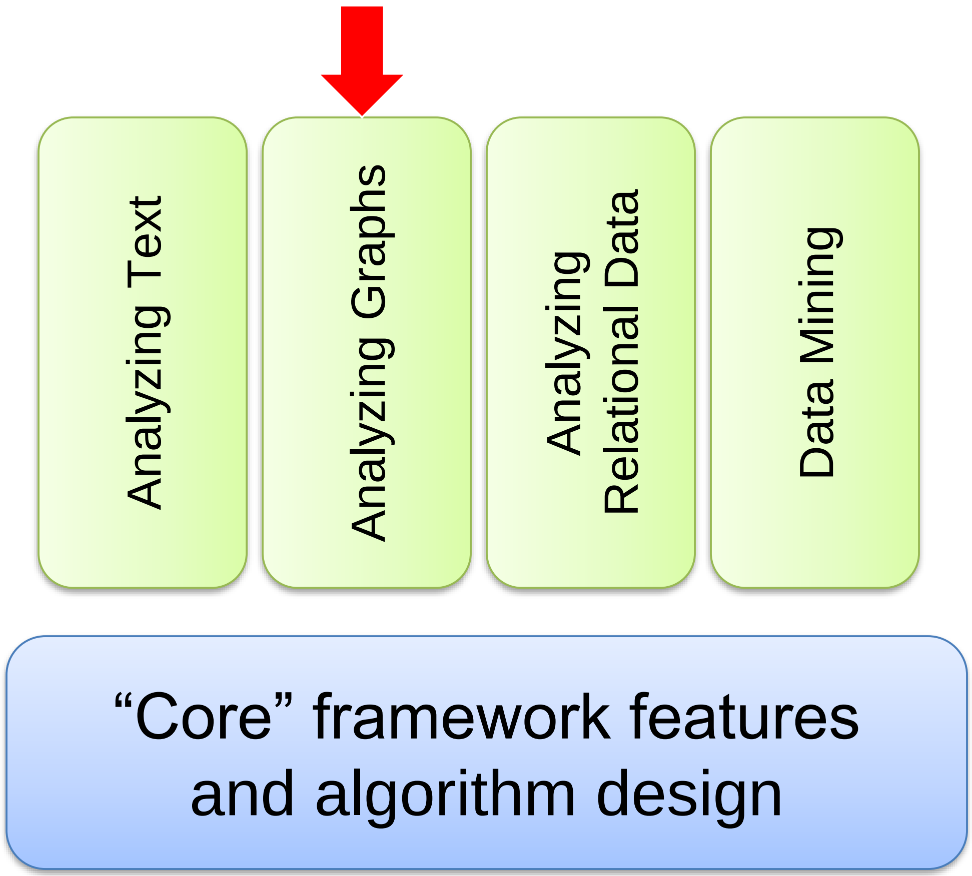
October 8, 2019

Ali Abedi

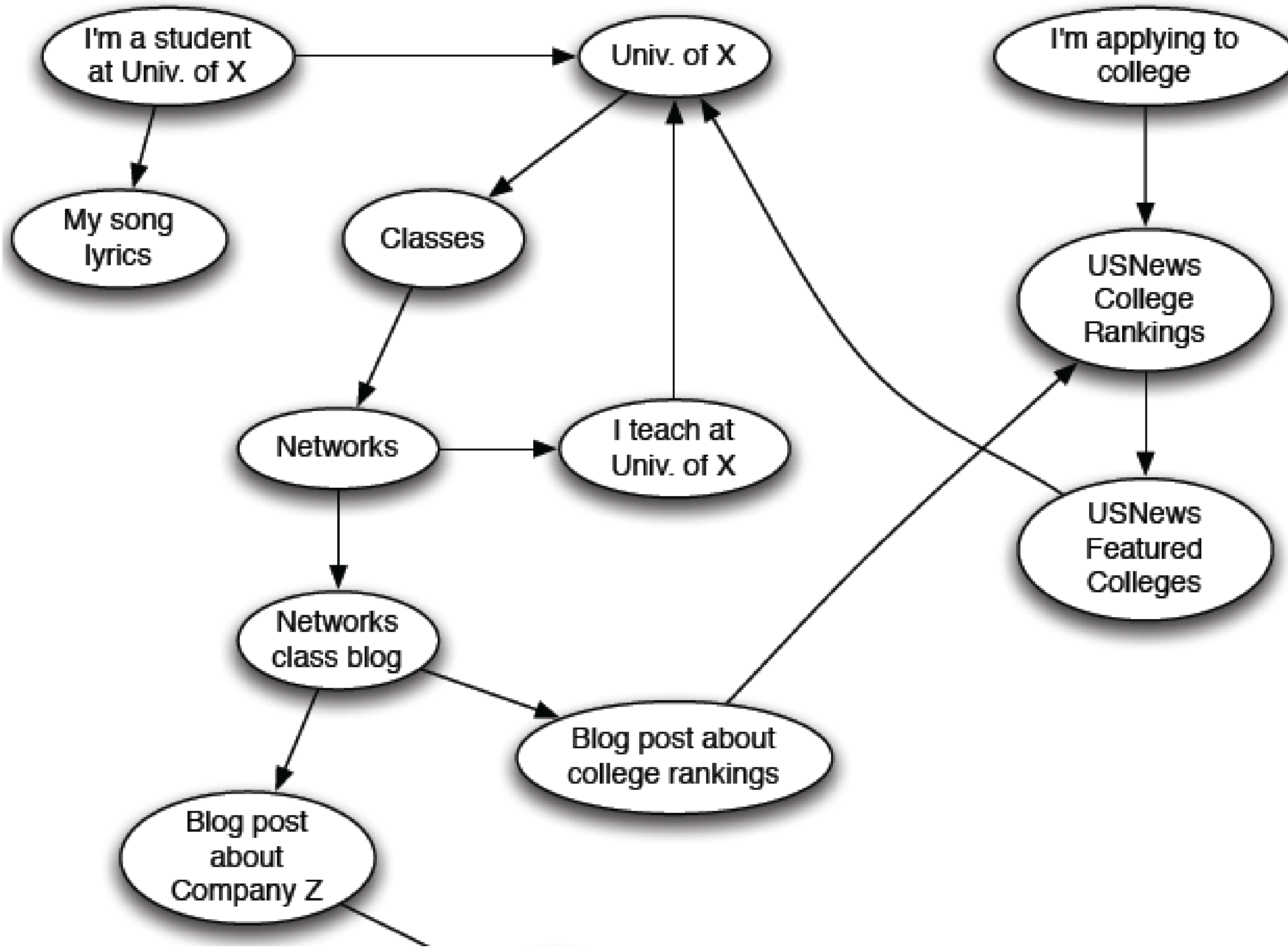
Thanks to Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University)

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451/

Structure of the Course



Web as a Directed Graph



Broad Question

- **How to organize the Web?**
- **First try:** Human curated **Web directories**
  - Yahoo, DMOZ, LookSmart
- **Second try: Web Search**
  - **Information Retrieval** investigates: Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - **But:** Web is **huge**, full of untrusted documents, random things, web spam, etc.

Web Search: 2 Challenges

2 challenges of web search:

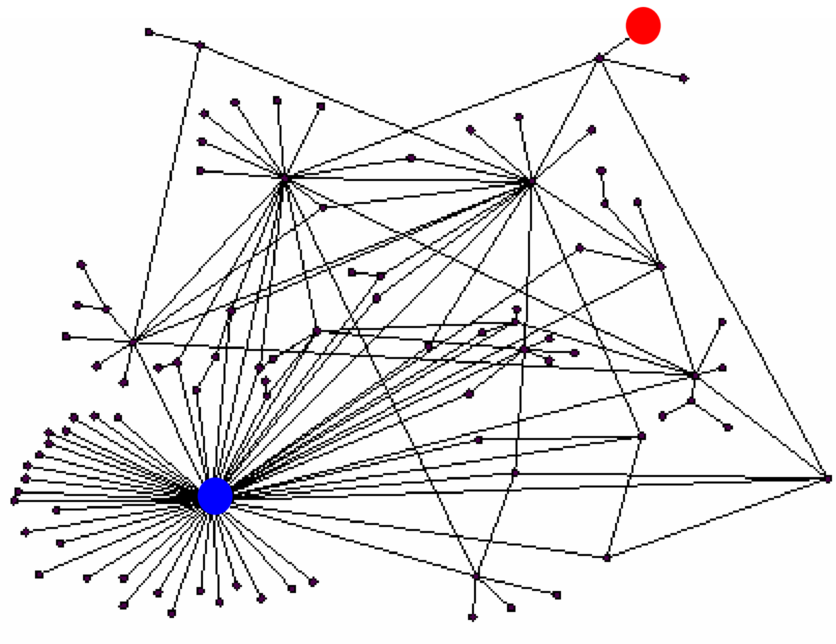
- **(1) Web contains many sources of information Who to "trust"?**
  - **Trick:** Trustworthy pages may point to each other!
- **(2) What is the "best" answer to query "newspaper"?**
  - No single right answer
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

- **All web pages are not equally "important"**

  www.joe-schmoe.com vs. www.stanford.edu

- There is large diversity in the web-graph node connectivity.
  Let's rank the pages by the link structure!

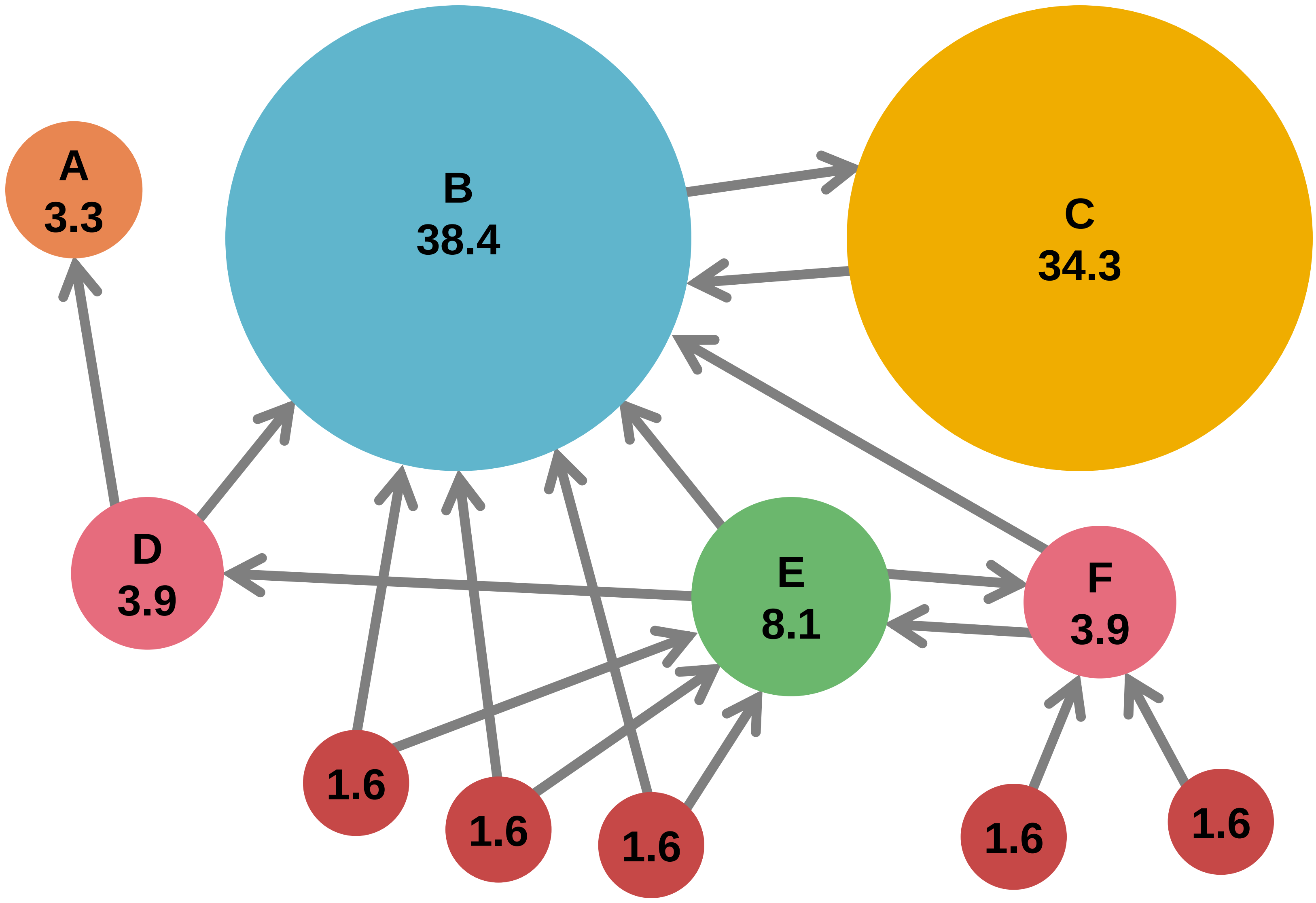


PageRank: The “Flow” Formulation

Links as Votes

- **Idea: Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - www.stanford.edu has 23,400 in-links
  - www.joe-schmoe.com has 1 in-link
- **Are all in-links are equal?**
  - **Links from important pages count more**
  - Recursive question!

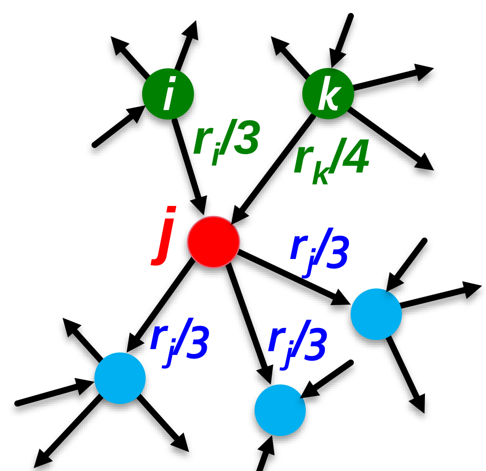
Example: PageRank Scores



Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page j with importance r_j has n out-links, each link gets r_j / n votes
- Page j's own importance is the sum of the votes on its in-links

$$r_j = r_i/3 + r_k/4$$

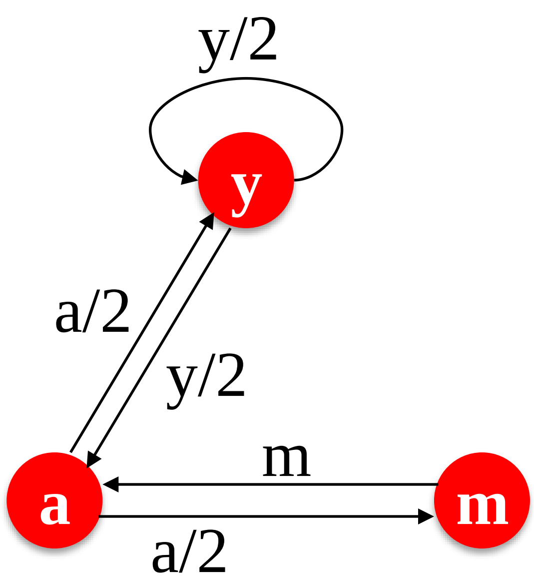


PageRank: The "Flow" Model

- A "vote" from an important page is worth more
- A page is important if it is pointed to by other important pages
- Define a "rank" r_j for page j

$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_i}$$

d_i … out-degree of node i



"Flow" equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Solving the Flow Equations

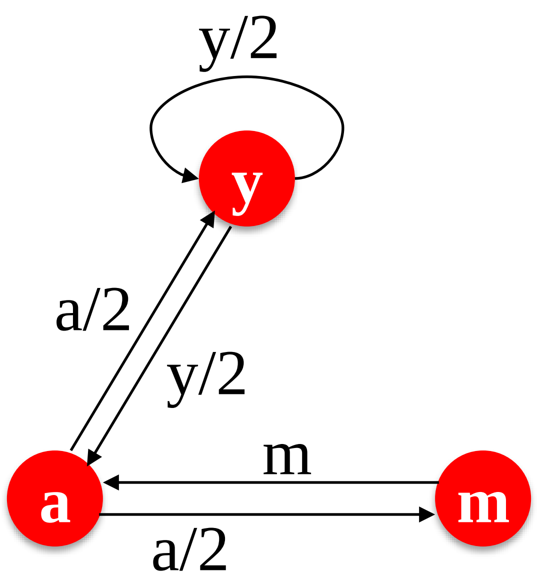
- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - All solutions equivalent modulo the scale factor
- **Additional constraint forces uniqueness:**
  - $r_y + r_a + r_m = 1$
  - **Solution:** $r_y = \frac{2}{5},\ r_a = \frac{2}{5},\ r_m = \frac{1}{5}$
- **Gaussian elimination method works for small examples, but we need a better method for large web-size graphs**
- **We need a new formulation!**

Flow equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

PageRank: Matrix Formulation

- **Stochastic adjacency matrix *M***
  - Let page i has d_i out-links
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - ***M*** is a **column stochastic matrix**
      - Columns sum to 1



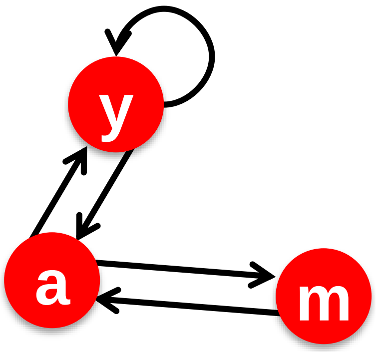
	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

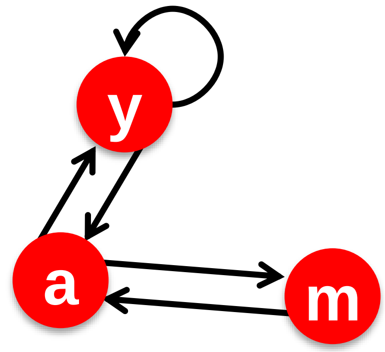
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

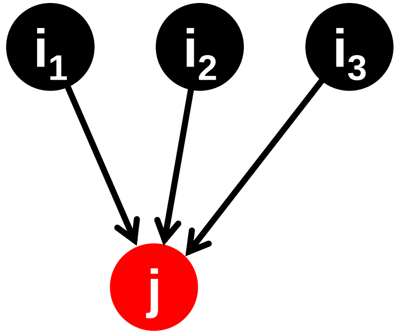
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

Random Walk Interpretation

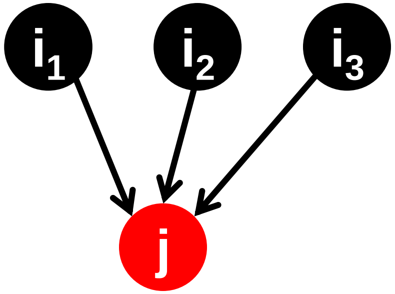


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

- **Imagine a random web surfer:**
  - At any time $\boldsymbol{t}$, surfer is on some page $\boldsymbol{i}$
  - At time $\boldsymbol{t+1}$, the surfer follows an out-link from $\boldsymbol{i}$ uniformly at random
  - Ends up on some page $\boldsymbol{j}$ linked from $\boldsymbol{i}$
  - Process repeats indefinitely
- **Let:**
  - $\boldsymbol{p(t)}$ … vector whose $\boldsymbol{i}^{\text{th}}$ coordinate is the prob. that the surfer is at page $\boldsymbol{i}$ at time $\boldsymbol{t}$
  - So, $\boldsymbol{p(t)}$ is a probability distribution over pages

The Stationary Distribution

- **Where is the surfer at time *t+1*?**
  - Follows a link uniformly at random
  $$\boldsymbol{p}(\boldsymbol{t}+1) = \boldsymbol{M} \cdot \boldsymbol{p}(\boldsymbol{t})$$



$$p(t+1) = \mathrm{M} \cdot p(t)$$

- Suppose the random walk reaches a state
$$\boldsymbol{p}(\boldsymbol{t}+1) = \boldsymbol{M} \cdot \boldsymbol{p}(\boldsymbol{t}) = \boldsymbol{p}(\boldsymbol{t})$$
then $\boldsymbol{p}(\boldsymbol{t})$ is **stationary distribution** of a random walk

Existence and Uniqueness

- **A central result from the theory of random walks (a.k.a. Markov processes):**

For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time **t = 0**

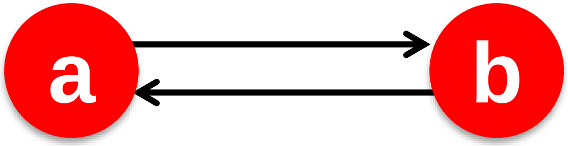
PageRank:
The Google Formulation

PageRank: Three Questions

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Does this converge?**
- **Does it converge to what we want?**
- **Are results reasonable?**

Does this converge?



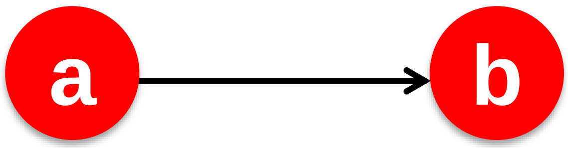
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

r_a	=	1	0	1	0
r_b		0	1	0	1

Iteration 0, 1, 2, …

Does it converge to what we want?



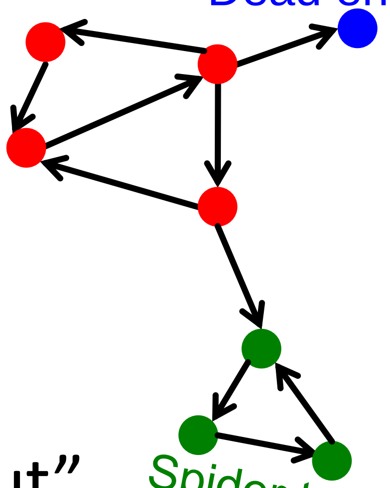
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

$$\begin{array}{l} \mathrm{r_a} \\ \mathrm{r_b} \end{array} = \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}$$

Iteration 0, 1, 2, …

PageRank: Problems



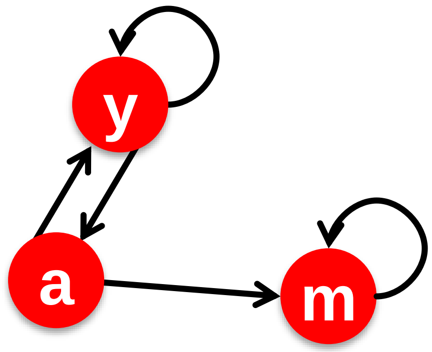
2 problems:

- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"

- **(2) Spider traps:** (all out-links are within the group)
  - Random walked gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2$$
$$r_m = r_a/2 + r_m$$

- **Example:**

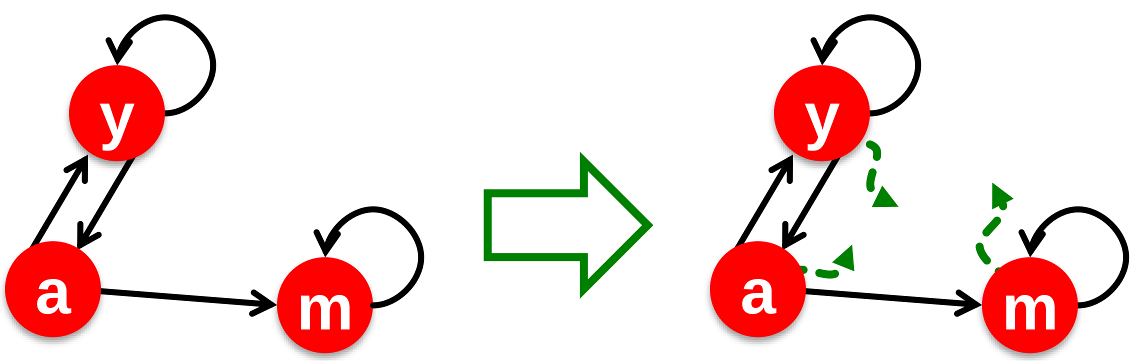
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{array}{cccccc} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{array}$$

Iteration 0, 1, 2, …

All the PageRank score gets "trapped" in node m.

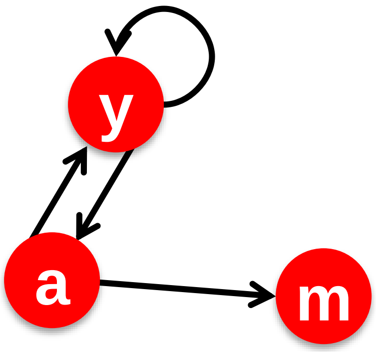
Solution: Teleports!

- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

$\mathbf{r_y = r_y/2 + r_a/2}$

$\mathbf{r_a = r_y/2}$

$\mathbf{r_m = r_a/2}$

- **Example:**

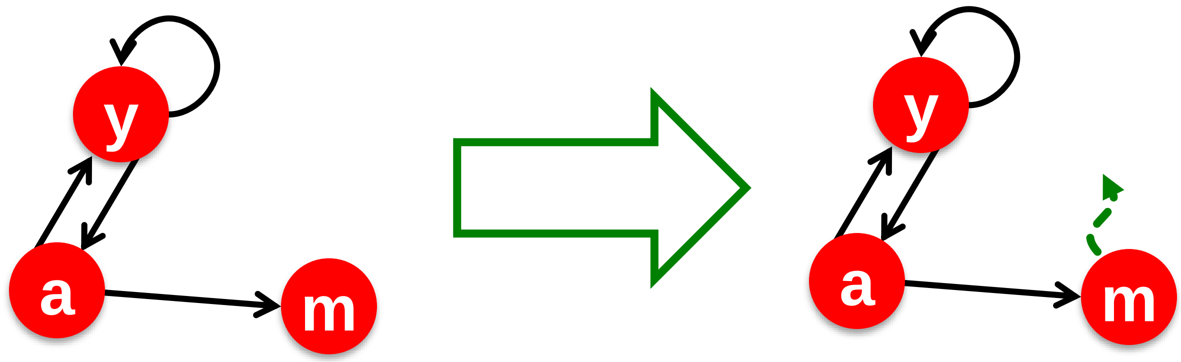
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Here the PageRank "leaks" out since the matrix is not stochastic.

Solution: Always Teleport!

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why Teleports Solve the Problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic, so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page

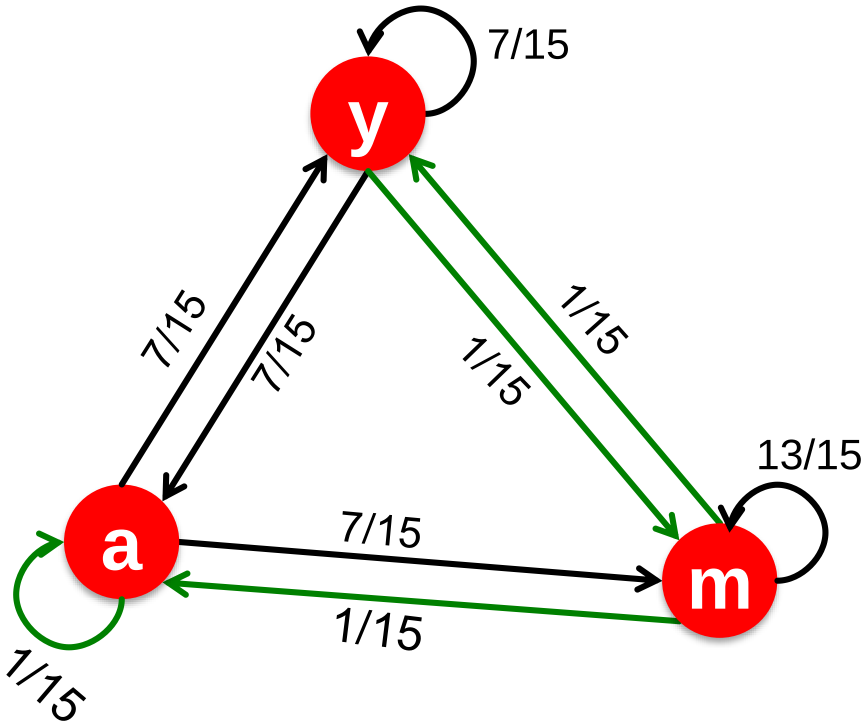
- **PageRank equation** [Brin-Page, 98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

Random Teleports ($\beta = 0.8$)



$$0.8 \underbrace{\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}}_{\mathbf{M}} + 0.2 \underbrace{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}_{[1/N]_{N\times N}}$$

$$\mathbf{A}: \begin{array}{c} y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 0.33 \\ 0.20 \\ 0.46 \end{array} \quad \begin{array}{c} 0.24 \\ 0.20 \\ 0.52 \end{array} \quad \begin{array}{c} 0.26 \\ 0.18 \\ 0.56 \end{array} \quad \cdots \quad \begin{array}{c} 7/33 \\ 5/33 \\ 21/33 \end{array}$$

PageRank MapReduce Implementation

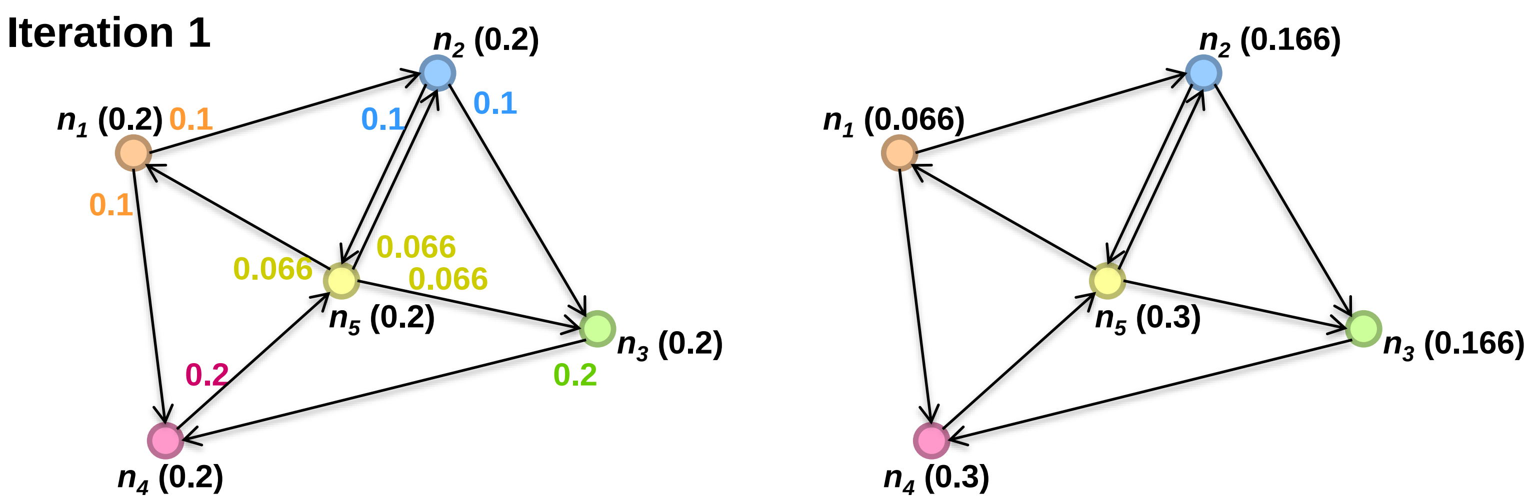
Simplified PageRank

First, tackle the simple case:

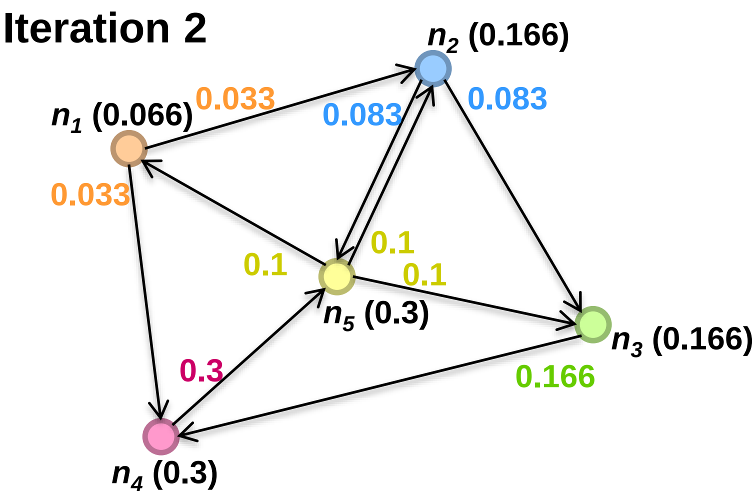
No random jump factor

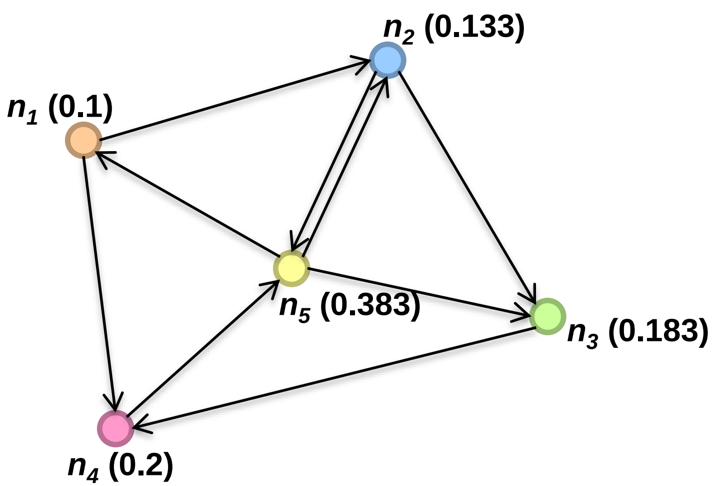
No dangling (dead-end) nodes

Sample PageRank Iteration (1)

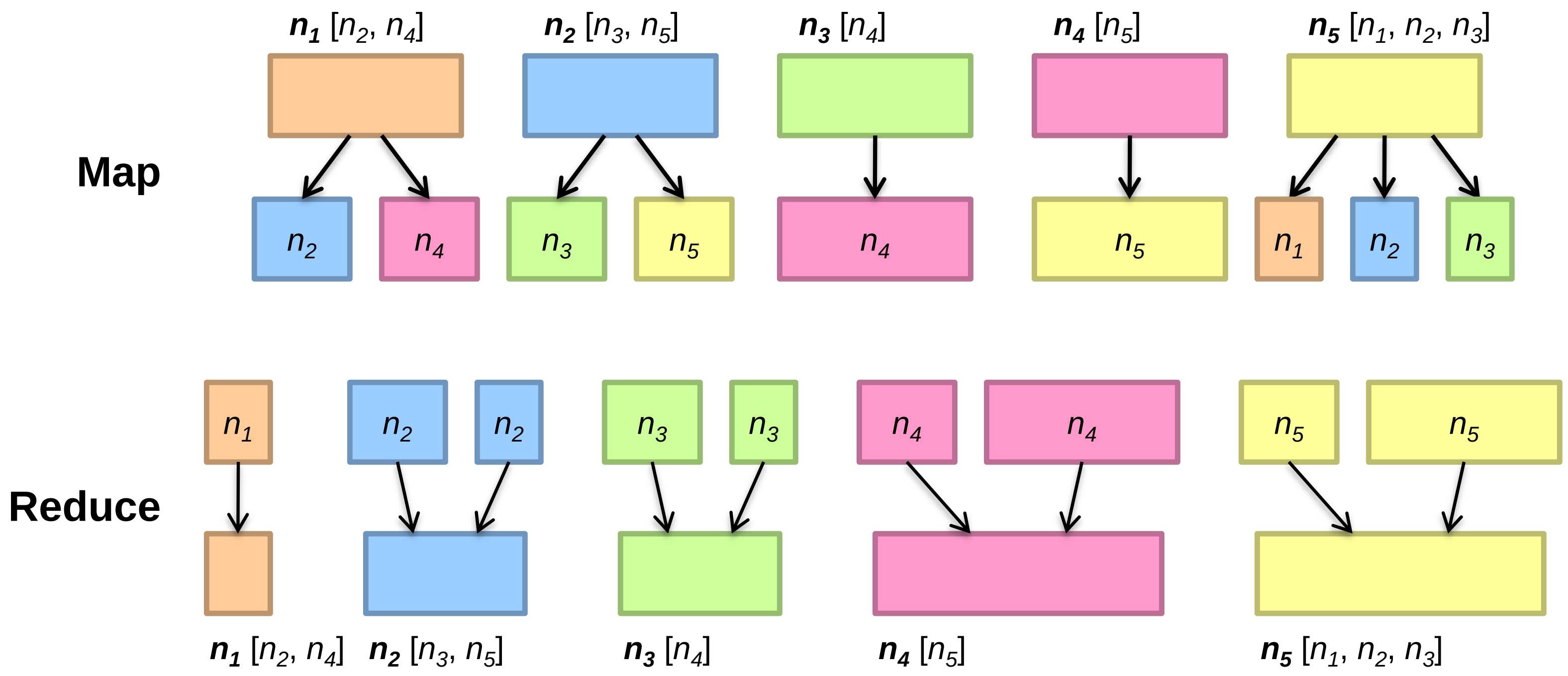


Sample PageRank Iteration (2)





PageRank in MapReduce



PageRank Pseudo-Code

```
class Mapper {
  def map(id: Long, n: Node) = {
    emit(id, n)
    p = n.PageRank / n.adjacenyList.length
    for (m <- n.adjacenyList) {
      emit(m, p)
    }
}

class Reducer {
  def reduce(id: Long, objects: Iterable[Object]) = {
    var s = 0
    var n = null
    for (p <- objects) {
      if (isNode(p))    n = p
      else              s += p
    }
    n.PageRank = s
    emit(id, n)
  }
}
```

PageRank vs. BFS

	PageRank	BFS
Map	PR/N	d+1
Reduce	sum	min

A large class of graph algorithms involve:

Local computations at each node

Propagating results: “traversing” the graph

Complete PageRank

Two additional complexities

What is the proper treatment of dangling nodes?

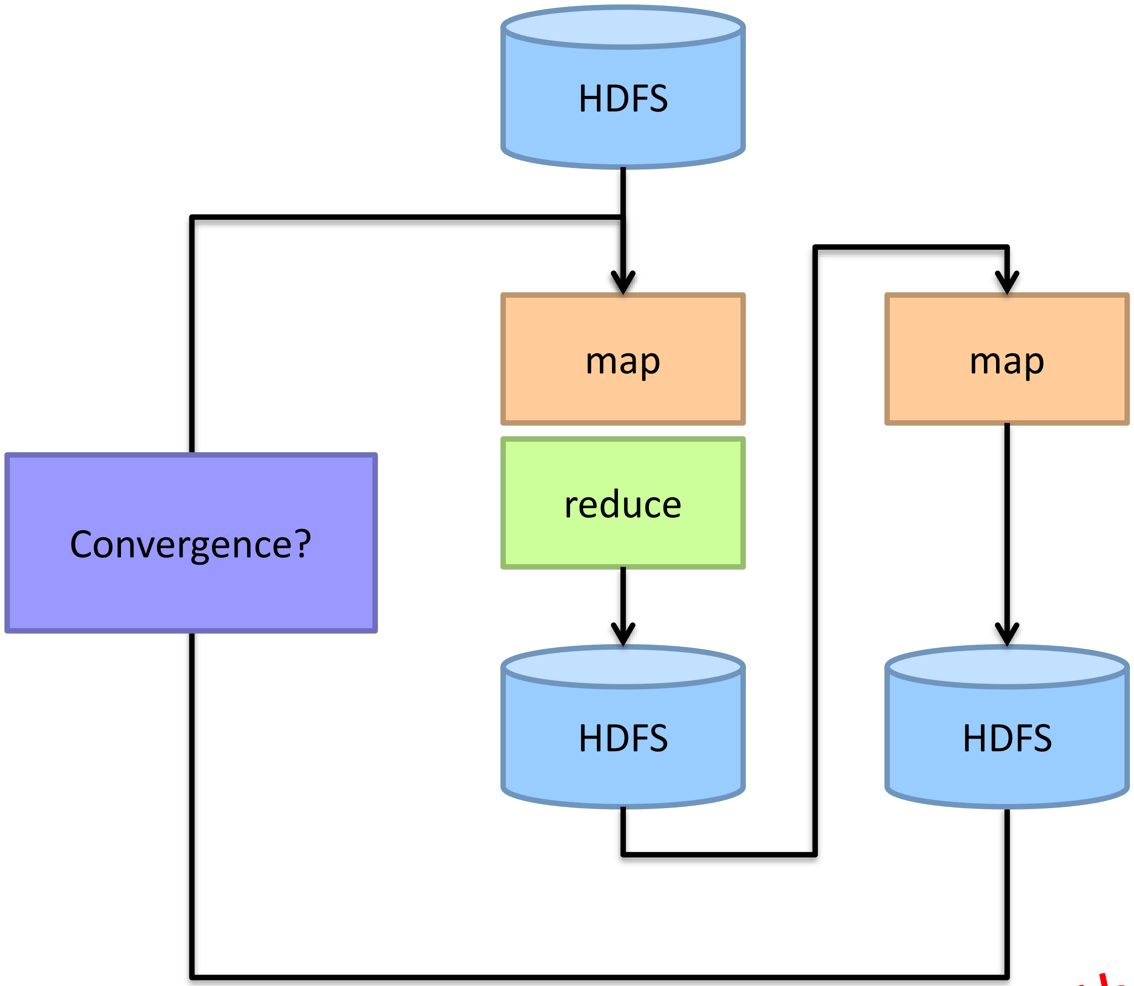
How do we factor in the random jump factor?

Solution: second pass to redistribute "missing PageRank mass" and account for random jumps

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

One final optimization: fold into a single MR job

Implementation Practicalities



What's the optimization?

PageRank Convergence

Alternative convergence criteria

Iterate until PageRank values don’t change

Iterate until PageRank rankings don’t change

Fixed number of iterations

Log Probs

PageRank values are *really* small…

Solution?

Product of probabilities = Addition of log probs

Addition of probabilities?

$$a \oplus b = \begin{cases} b + \log(1 + e^{a-b}) & a < b \\ a + \log(1 + e^{b-a}) & a \geq b \end{cases}$$

Beyond PageRank

Variations of PageRank

Weighted edges

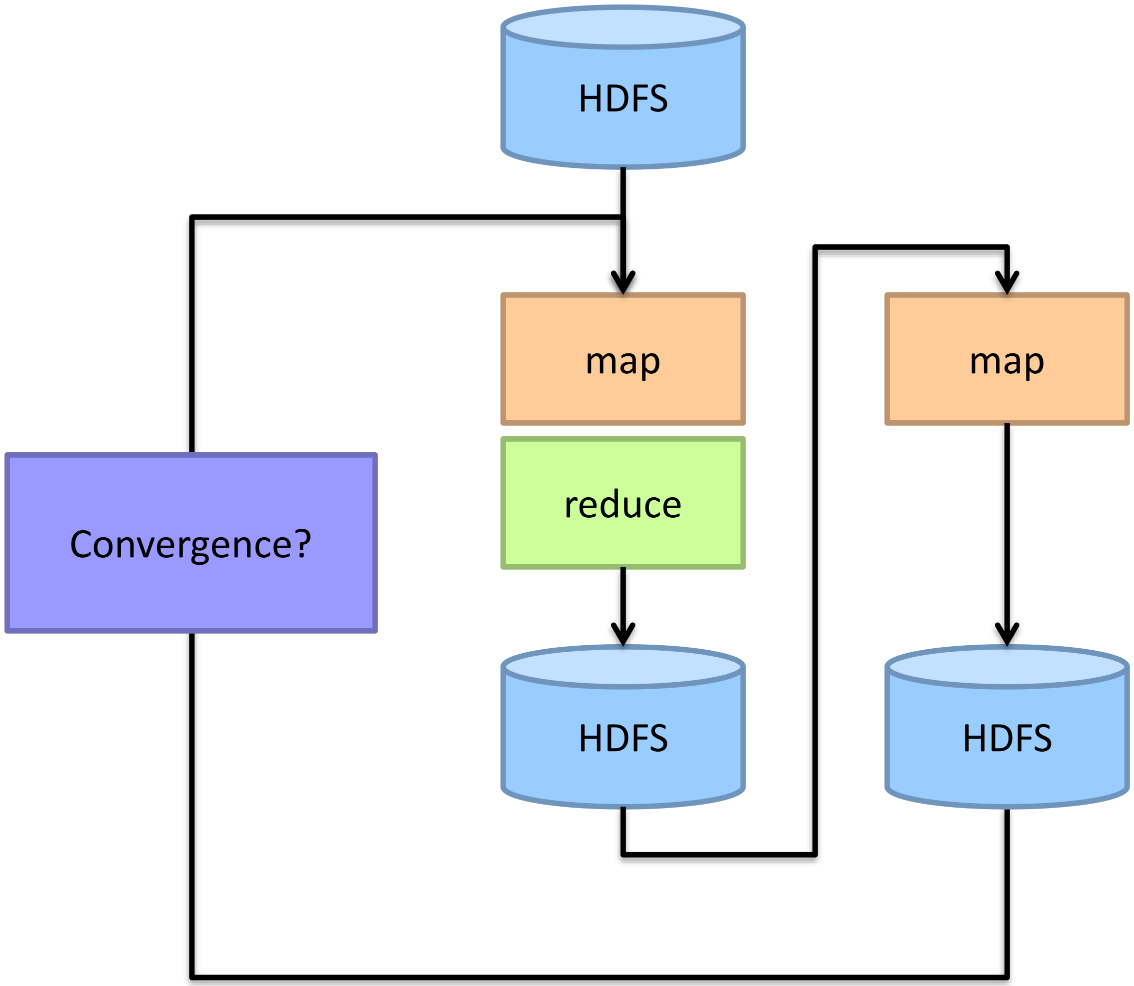
Personalized PageRank (A4 ☺)

Variants on graph random walks

Hubs and authorities (HITS)

SALSA

Implementation Practicalities



MapReduce Sucks

Java verbosity

Hadoop task startup time

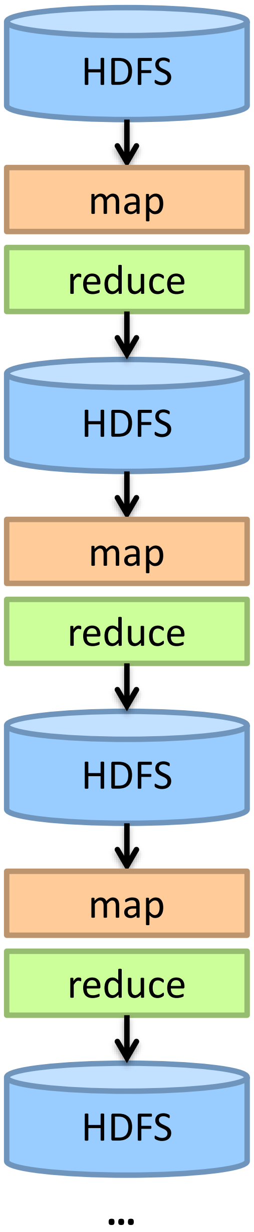
Stragglers

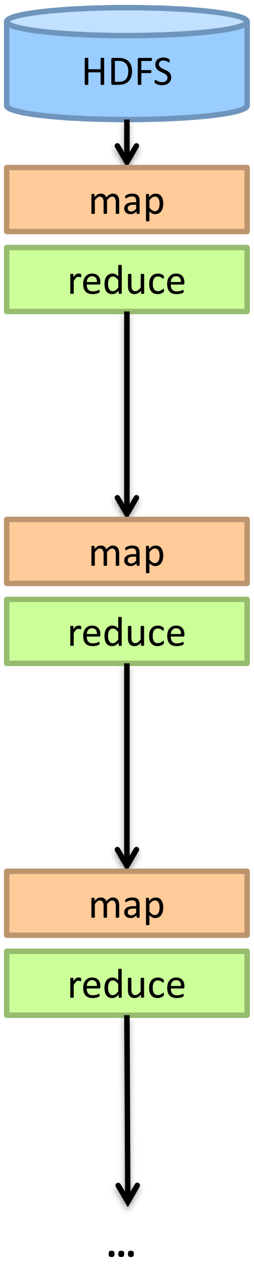
Needless graph shuffling

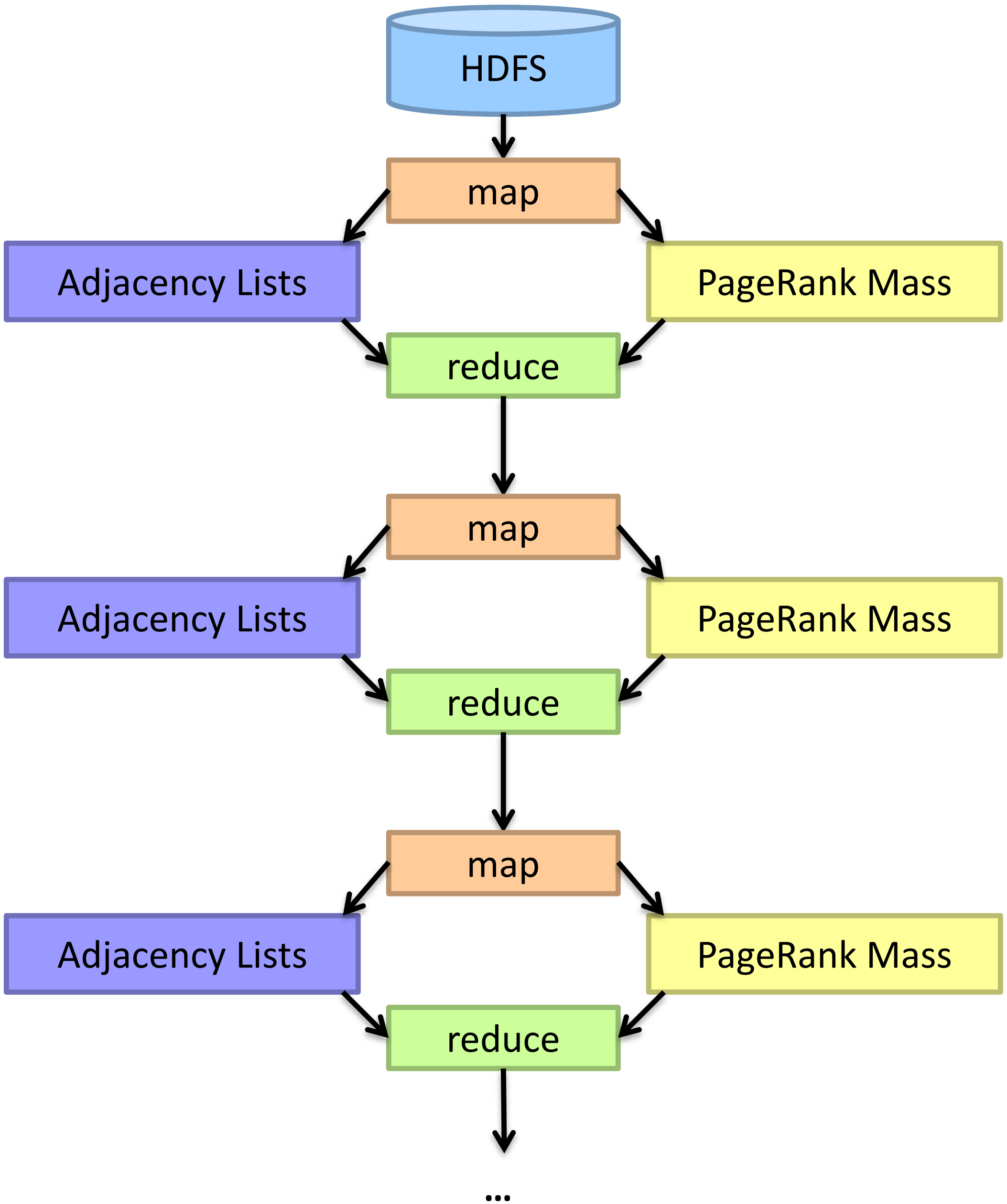
Checkpointing at each iteration

Spark to the rescue?

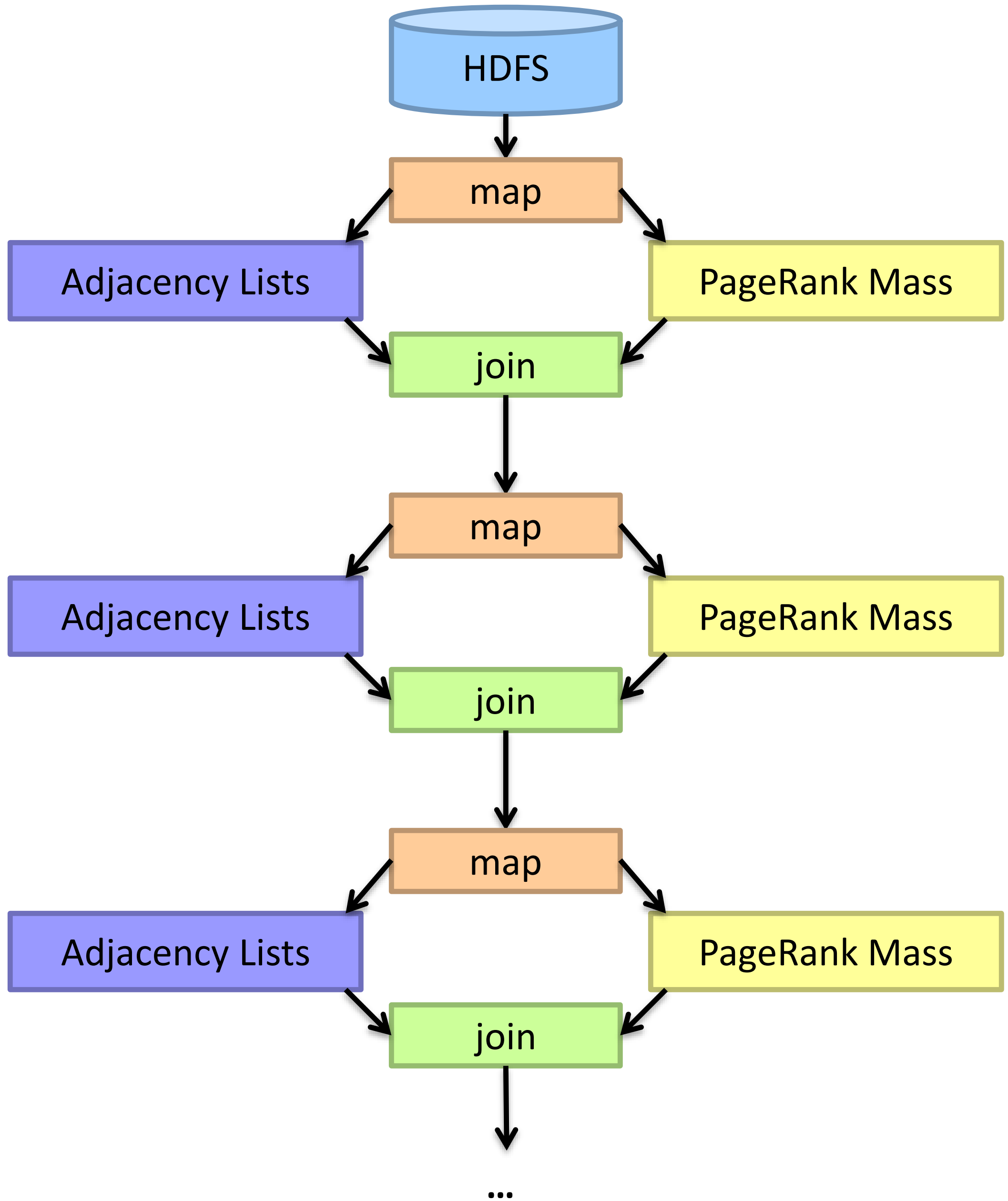
Let’s Spark!

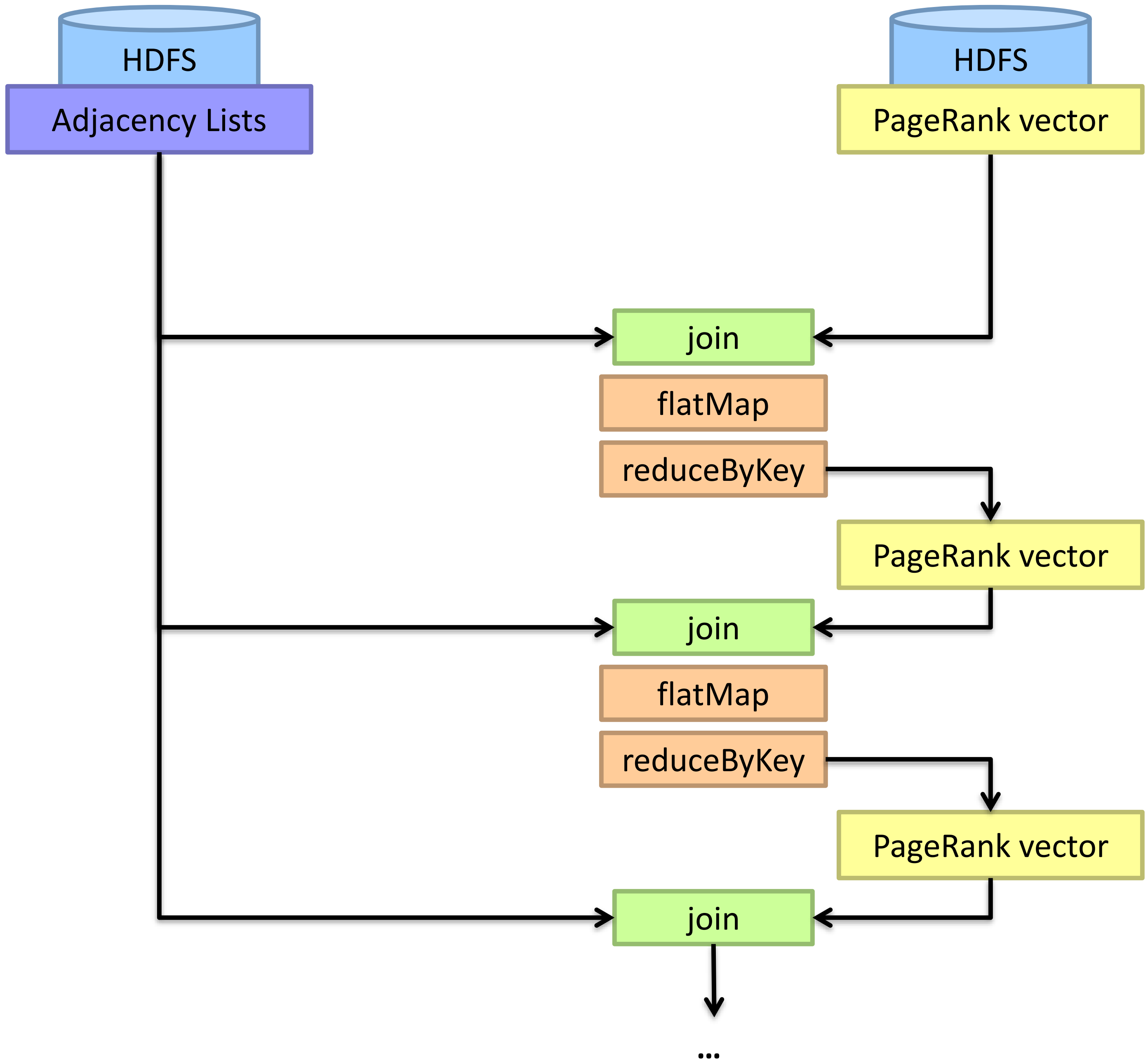



HDFS
map
reduce
map
reduce
map
reduce
...

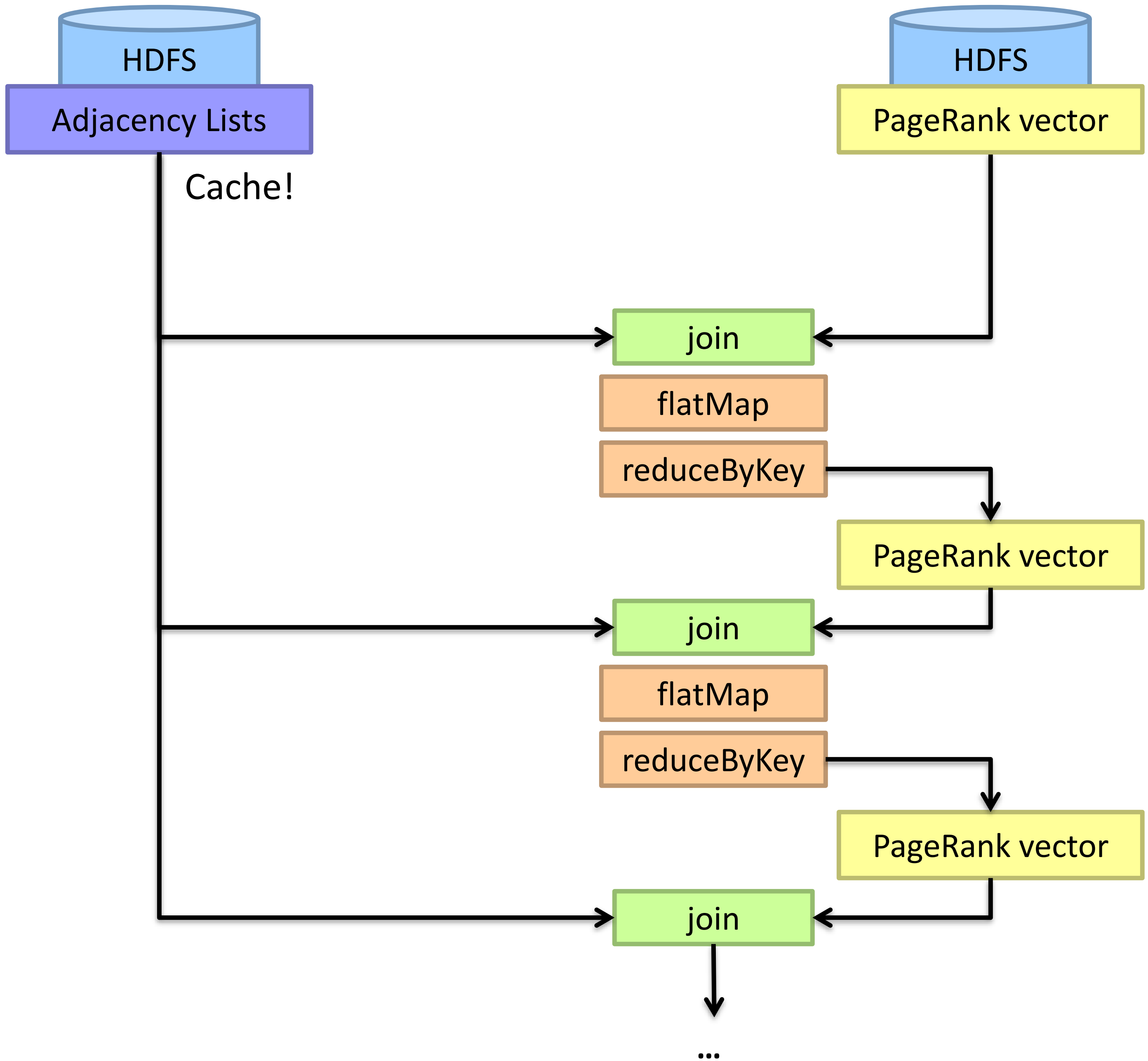


HDFS
map
Adjacency Lists
PageRank Mass
reduce
map
Adjacency Lists
PageRank Mass
reduce
map
Adjacency Lists
PageRank Mass
reduce
…

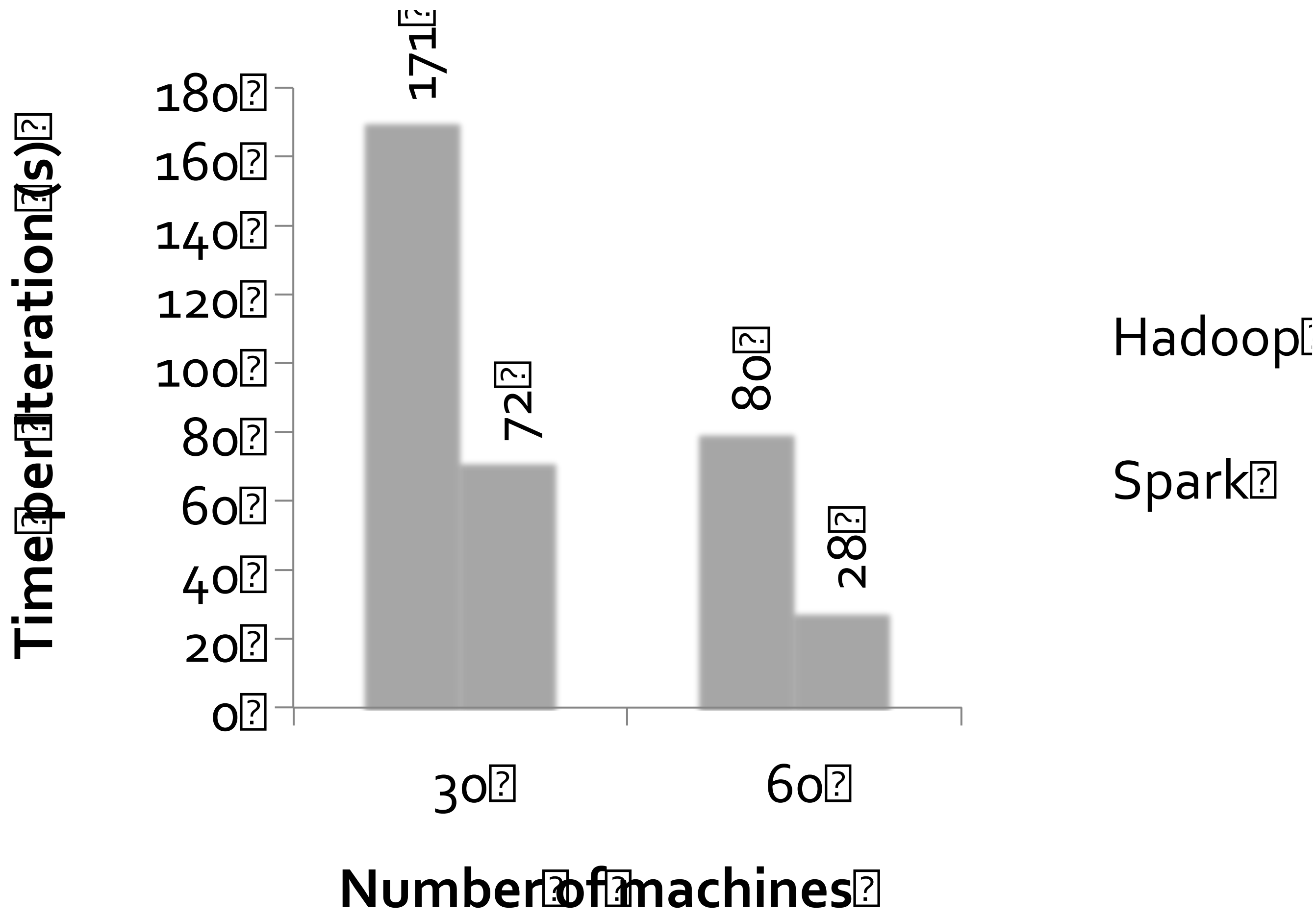

HDFS
map
Adjacency Lists
PageRank Mass
join
map
Adjacency Lists
PageRank Mass
join
map
Adjacency Lists
PageRank Mass
join
…



HDFS
Adjacency Lists
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…


HDFS
Adjacency Lists
Cache!
HDFS
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
flatMap
reduceByKey
PageRank vector
join
…

MapReduce vs. Spark



Source: http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-part-2-amp-camp-2012-standalone-programs.pdf