CS4225/CS5425 Big Data Systems for Data Science

PageRank and Graph Processing

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg

School of Computing

Announcements

- Next week is last lecture + tutorial (on stream processing)
- Practice test will be posted next week. For those who want to do the test in school, you can come to COM1-02-06 Seminar Room 1.
- HW2 is due 31 Oct 11.59pm

Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	19 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	
5	9 Sep	NoSQL Overview 1		Assignment 1 released
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 2 released
8	7 Oct	Apache Spark 2		Assignment 1 due (13 Oct)
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	Assignment 2 due (31 Oct)
12	4 Nov	Deepavali – No Class		
13	11 Nov	**Test**		

Recap: PageRank Flow Formulation

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i = number of out-links (or "out-degree") of node i

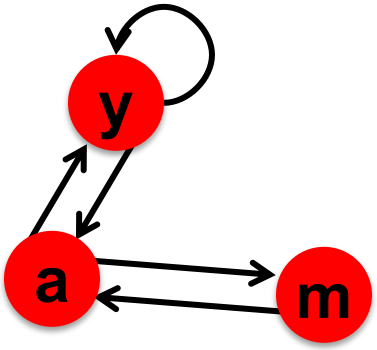
Importance of j

Sum of importances of pages linking to j, each divided by number of out-links

Matrix Form:

$$\mathbf{r} = \mathbf{M} \cdot \mathbf{r}$$

Example:



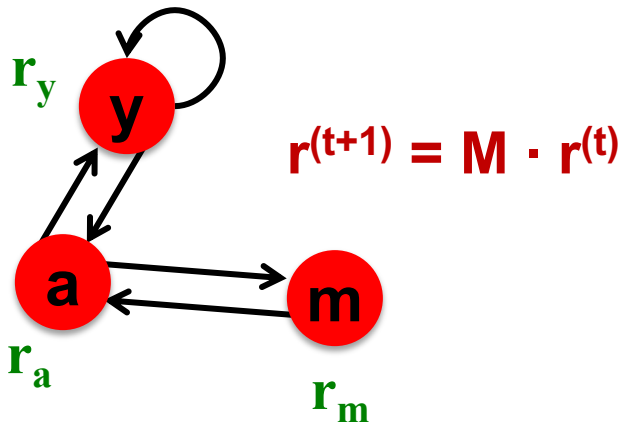
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r'_y \\ r'_a \\ r'_m \end{bmatrix}$$

Recap: Power Iteration

Power Iteration:

- Suppose there are *N* web pages
- Initialize: $\mathbf{r}^{(0)} = [1/N,\ldots,1/N]^T$
- Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \bullet \mathbf{r}^{(t)}$
- Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$



- **Example:**

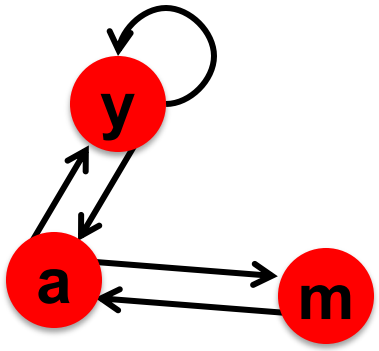
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 2/5 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 2/5 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 1/5 \end{matrix}$$

Iteration 0 Iteration 1 ⋯

Recap: Random Walk Interpretation

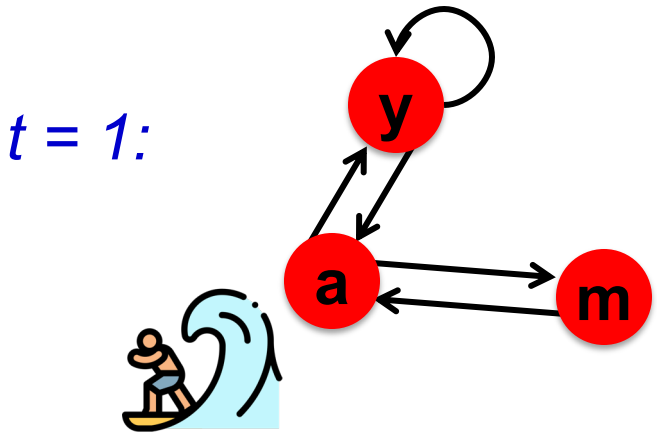
- Imagine a random web surfer:
  - At time t = 0, surfer starts on a random page
  - At any time t, surfer is on some page i
  - At time $t + 1$, the surfer follows an out-link from i uniformly at random
  - Process repeats indefinitely

t = 0:



Recap: Random Walk Interpretation

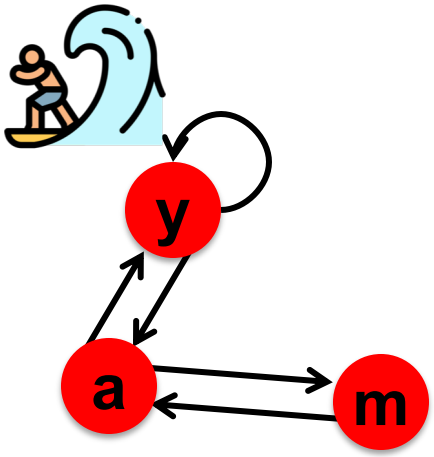
- Imagine a random web surfer:
  - At time t = 0, surfer starts on a random page
  - At any time t, surfer is on some page i
  - At time $t + 1$, the surfer follows an out-link from i uniformly at random
  - Process repeats indefinitely



Recap: Random Walk Interpretation

- Imagine a random web surfer:
  - At time t = 0, surfer starts on a random page
  - At any time t, surfer is on some page i
  - At time $t + 1$, the surfer follows an out-link from i uniformly at random
  - Process repeats indefinitely

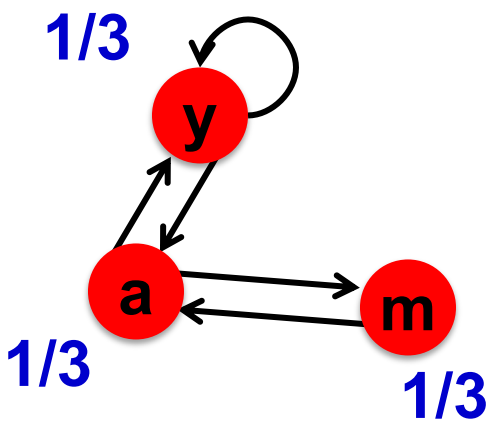
t = 2:



Recap: Random Walk Interpretation

- Let:
  - $p(t)$ … vector whose ith coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages

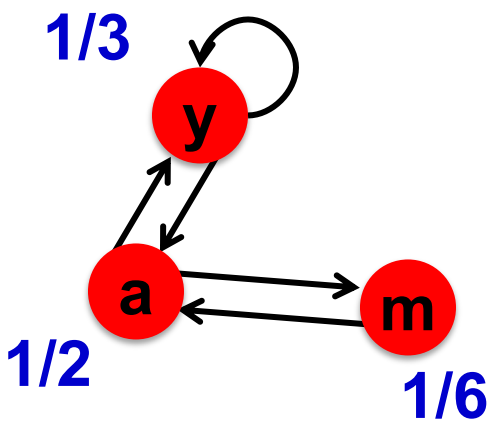
p(t) where t = 0:



Recap: Random Walk Interpretation

- Let:
  - $p(t)$ … vector whose ith coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages

p(t) where t = 1:



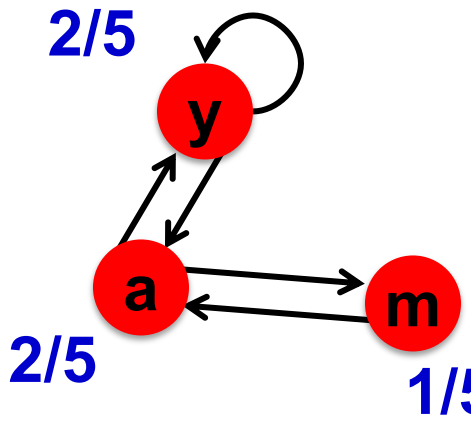
Recap: Random Walk Interpretation

- Let:
  - $p(t)$ … vector whose ith coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages
- **Stationary Distribution**: as $t \to \infty$, the probability distribution gradually converges to a 'steady state'
- PageRank equations are the same, except that r_j represents the (steady state) probability of the random walker being at node j:

Probability of random walker being at j

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

p(t) where $t \to \infty$:



Today's Plan

- **PageRank with Teleports**
- Topic Sensitive PageRank & TrustRank
- Graph Processing: Pregel and Giraph

PageRank: Three Questions

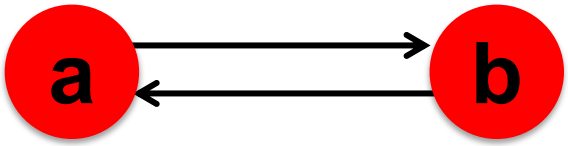
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

or equivalently

$$r = Mr$$

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

Does this converge?



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

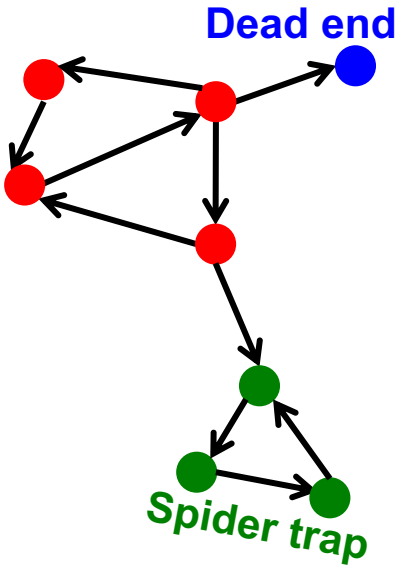
- **Answer: not always. Example:**

r_a	=	1	0	1	0
r_b		0	1	0	1

Iteration 0, 1, 2, …

Does it converge to what we want?

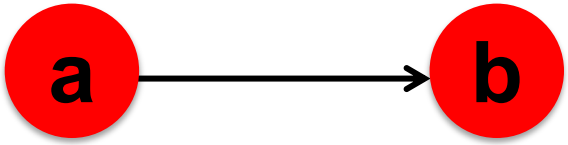
Answer: not always. 2 problems:



- **(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 walk gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Problem 1: Dead Ends



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

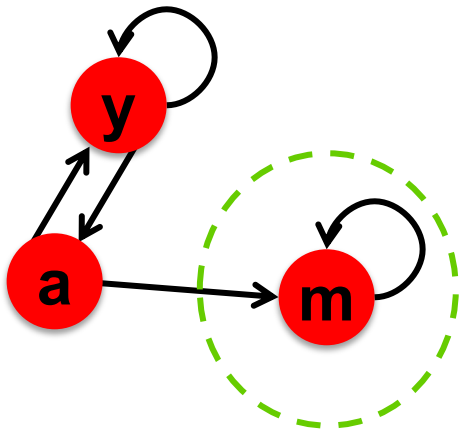
- **Example:**

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Problem 2: Spider Traps

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



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

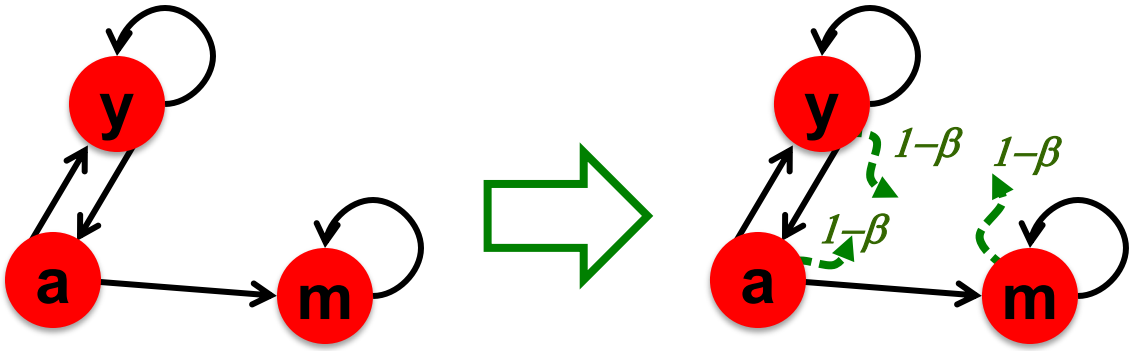
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

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

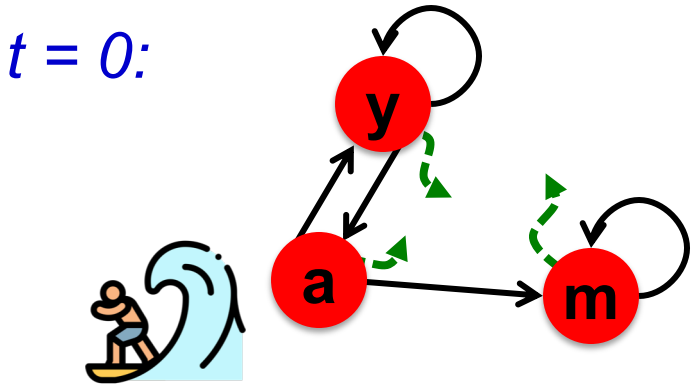
Solution: Teleports!

- **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



Solution: Teleports!

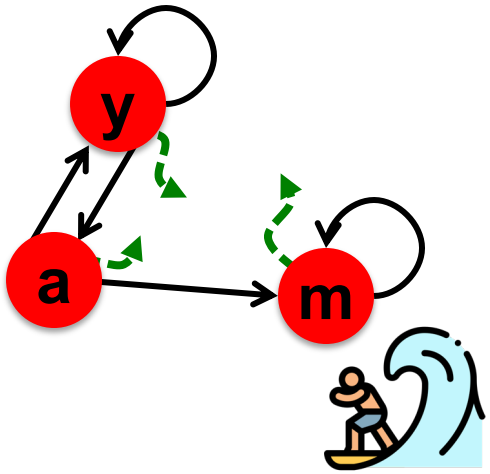
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9



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

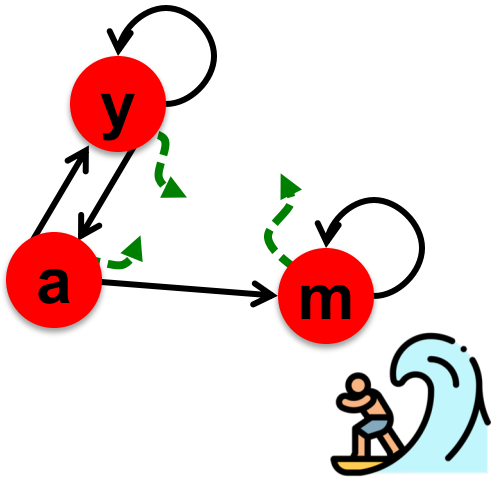
t = 1:



Solution: Teleports!

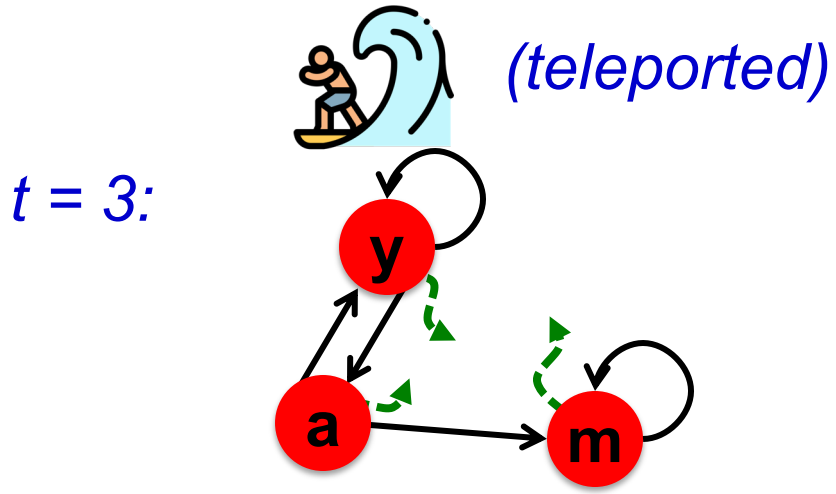
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9

t = 2:



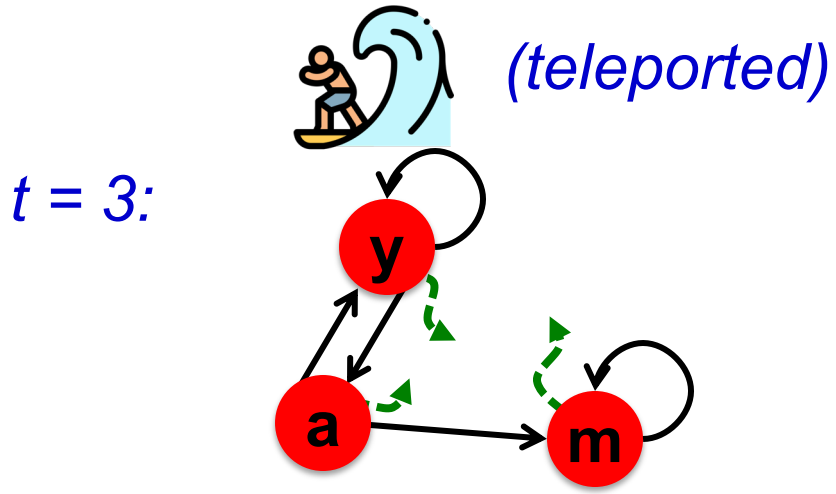
Solution: Teleports!

- **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



Solution: Teleports!

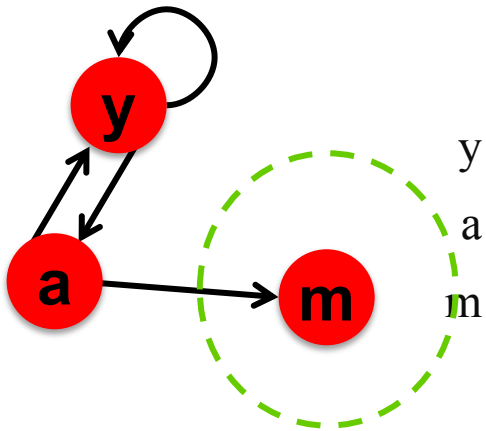
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9



- **Conclusion:** surfer will quickly teleport out of any spider trap

Problem: Dead Ends

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



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

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

- **Example:**

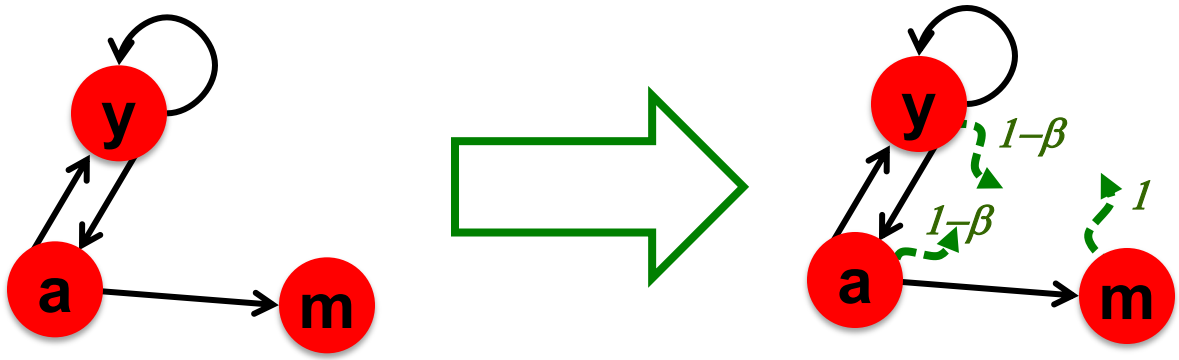
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

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

Solution: If at a Dead End, Always Teleport

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



Why Teleports Solve the Problem?

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

- **Spider-traps** cause random walker to get stuck in them, absorbing all importance
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** cause importance to "leak" out of the system
  - The matrix is not column stochastic
  - **Solution:** Make matrix column stochastic by always teleporting when at a dead end

Random Teleport: Equations

- At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1-\beta$, jump to some random page

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

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

Same as simplified PageRank

Teleport term

d_i … out-degree of node i

- *Compare to Simplified PageRank:*

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

(For now, assume that there are no dead ends. If there are, we can handle them - see next slide)

The Google Matrix

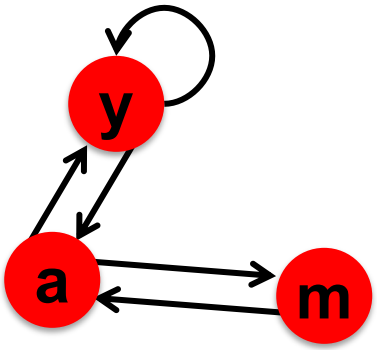
- **PageRank equation** [Brin-Page, '98]

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

- We can also write this as a matrix equation, by defining the **Google Matrix *A***:

$$A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

N by N matrix
where all entries are 1/N



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

	y	a	m
y	⅓	⅓	⅓
a	⅓	⅓	⅓
m	⅓	⅓	⅓

The Google Matrix

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

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

- **The Google Matrix *A*:**

y	a	m
½	½	0
½	0	1
0	½	0

$$A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

y	a	m
⅓	⅓	⅓
⅓	⅓	⅓
⅓	⅓	⅓

- **We have a recursive problem: $r = A \cdot r$**
  And the Power Iteration method still works!
- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (roughly, *5-10* steps on avg before a teleport)

Some Problems with Page Rank

- **Measures generic popularity of a page**
  - Doesn't consider popularity based on specific topics
  - **Solution:** Topic-Specific PageRank
- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank

Today's Plan

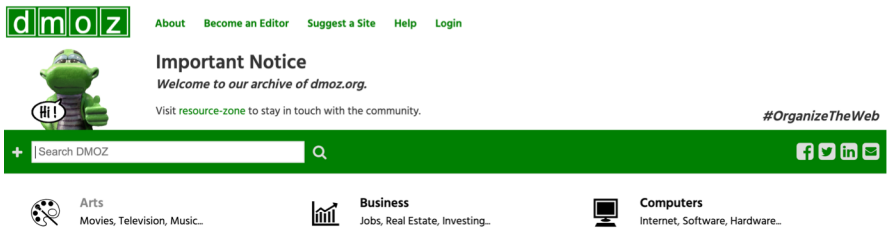
- PageRank with Teleports
- **Topic Sensitive PageRank & TrustRank**
- Graph Processing: Pregel and Giraph

Topic-Specific PageRank

- **Instead of generic popularity, can we measure popularity within a topic?**
- **Goal:** Evaluate Web pages not just according to their popularity, but by how close they are to a particular topic, e.g. "sports" or "history"
- **Allows search queries to be answered based on interests of the user**
  - **Example:** Query "Trojan" wants different pages depending on whether you are interested in sports, history and computer security

Topic-Specific PageRank

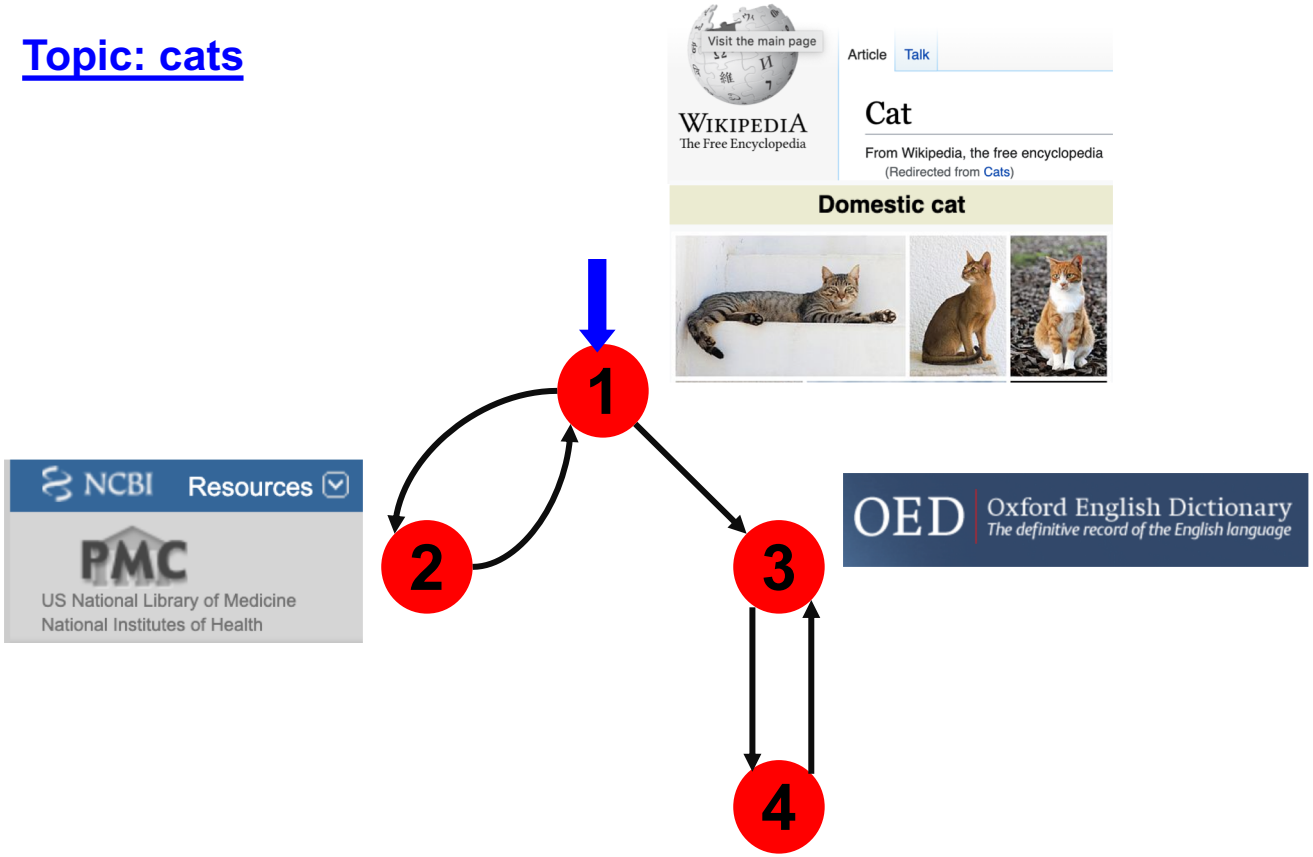
- Random walker has a small probability of teleporting at any step
- **Teleport can go to:**
  - **Standard PageRank: Any page with equal probability**
    - To avoid dead-end and spider-trap problems
  - **Topic Specific PageRank: A topic-specific set of "relevant" pages (teleport set)**
- **Idea: Bias the random walk**
  - When random walker teleports, it picks a page from a set ***S***
  - ***S*** contains only pages that are relevant to the topic
    - E.g., Open Directory (DMOZ) pages for a given topic/query
  - For each teleport set ***S***, we get a different vector r_S



Topic Specific PageRank

Example: S = Wikipedia page on cats

Topic: cats



- **Idea:** By setting the teleport set S as a small set of relevant pages for a topic, higher importance is assigned to pages in S and the web pages closely linked to S

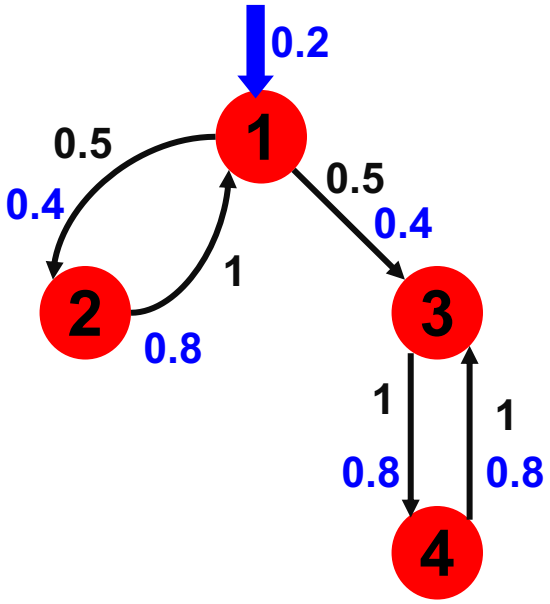
Matrix Formulation

- **To make this work all we need is to update the teleportation part of the PageRank formulation:**

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\ \beta M_{ij} + 0 & \text{otherwise} \end{cases}$$

  - ***A*** is stochastic!
- We weighted all pages in the teleport set ***S*** equally
- **Compute as for regular PageRank:**
  - Multiply by ***M***, then add a vector
  - Maintains sparseness

Example: Topic-Specific PageRank



Probabilities (no teleport)

Probabilities (with teleport)

$S = \{1\}$ $\beta = 0.8$

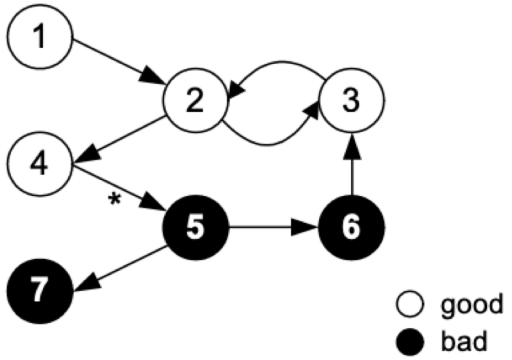
Node	Iteration				
	0	1	2	…	stable
1	0.25	0.4	0.28		0.294
2	0.25	0.1	0.16		0.118
3	0.25	0.3	0.32		0.327
4	0.25	0.2	0.24		0.261

S={1}, β=0.90:
r=[0.17, 0.07, 0.40, 0.36]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]
S={1}, β=0.70:
r=[0.39, 0.14, 0.27, 0.19]

S={1,2,3,4}, β=0.8:
r=[0.13, 0.10, 0.39, 0.36]
S={1,2,3} , β=0.8:
r=[0.17, 0.13, 0.38, 0.30]
S={1,2} , β=0.8:
r=[0.26, 0.20, 0.29, 0.23]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]

TrustRank

- **Goal**: identify trusted vs untrusted pages (e.g. spam pages). Start with a small set of trusted pages and propagate trust (assuming that "good" pages only rarely link to "bad" pages)



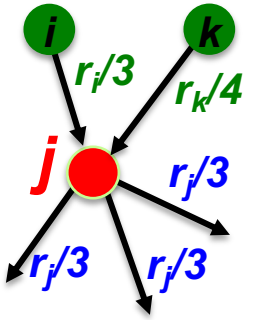
- **Approach:**
  - Sample a set of **seed pages** from the web
  - Have an **oracle** (**human**) to identify the good pages and the spam pages in the seed set
  - Perform a topic-sensitive PageRank with **teleport set = trusted pages**

Today's Plan

- PageRank with Teleports
- Topic Sensitive PageRank & TrustRank
- **Graph Processing: Pregel and Giraph**

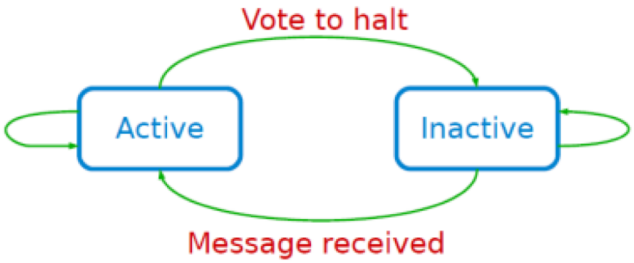
Characteristics of Graph Algorithms

- What are some common features of graph algorithms?
  - Local computations at each node
  - Passing messages to other nodes
- **Think like a vertex**: algorithms are implemented from the view of a single vertex, performing one iteration based on messages from its neighbor
  - Similar to MapReduce, the user only implements a simple function, `compute()`, that describes the algorithm's behavior at one vertex, in one step.
  - The framework abstracts away the scheduling / implementation details.



Pregel: Computational Model

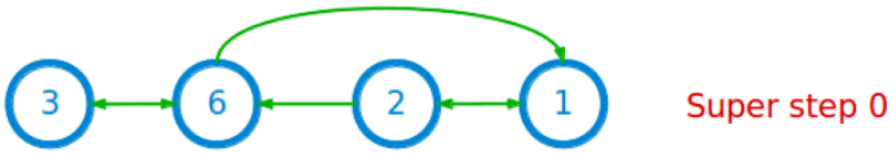
- Computation consists of a series of supersteps
- In each superstep, the framework invokes a user-defined function, `compute()`, for each vertex (conceptually in parallel)
- `compute()` specifies behavior at a single vertex *v* and a superstep *s*:
  - It can read messages sent to *v* in superstep *s* - 1;
  - It can send messages to other vertices that will be read in superstep *s* + 1;
  - It can read or write the value of *v* and the value of its outgoing edges (or even add or remove edges)
- Termination:
  - A vertex can choose to deactivate itself
  - Is "woken up" if new messages received
  - Computation halts when all vertices are inactive



Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

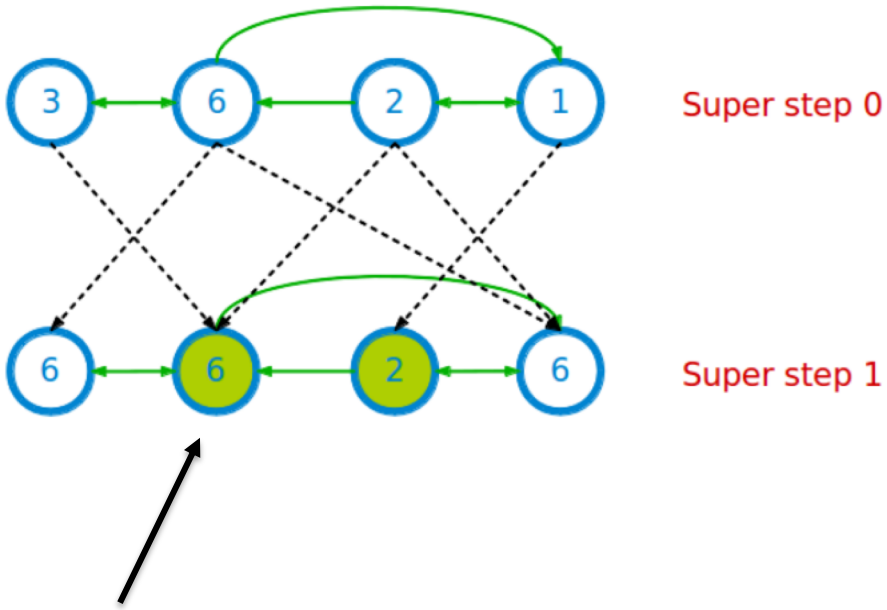
Example: Computing Max Value

```
Compute(v, messages):
  changed = False
  for m in messages:
    if v.getValue() < m:
      v.setValue(m)
      changed = True
  if changed:
    for each outneighbor w:
      sendMessage(w, v.getValue())
  else:
    voteToHalt()
```



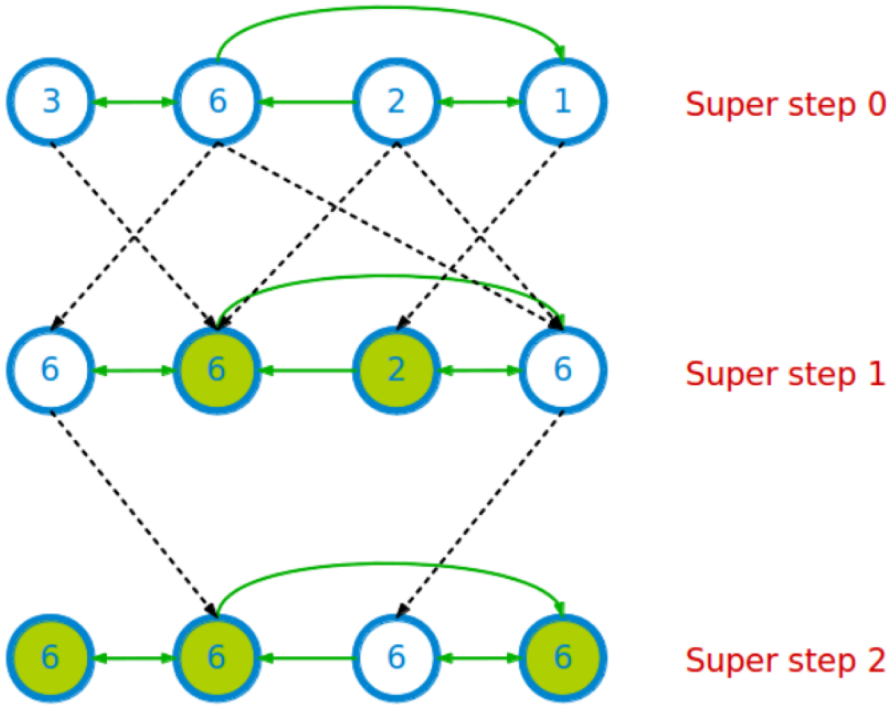
Example: Computing Max Value

```
Compute(v, messages):
  changed = False
  for m in messages:
    if v.getValue() < m:
      v.setValue(m)
      changed = True
  if changed:
    for each outneighbor w:
      sendMessage(w, v.getValue())
  else:
    voteToHalt()
```



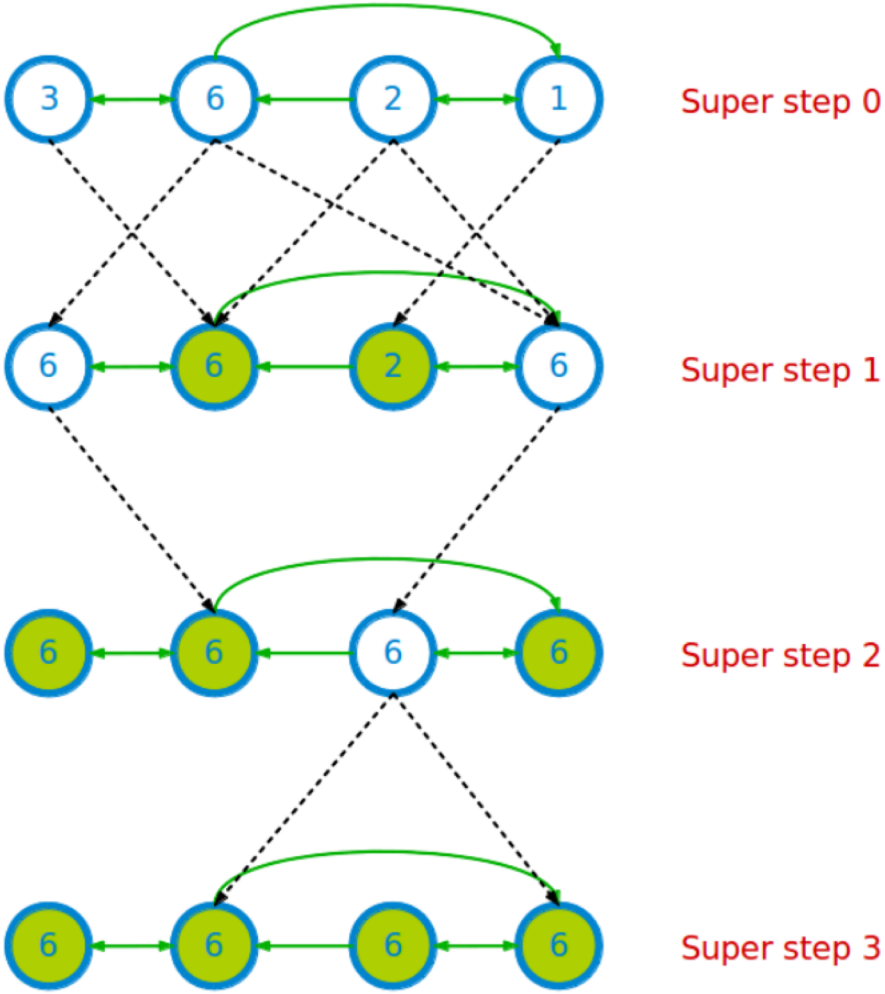
Example: Computing Max Value

```
Compute(v, messages):
  changed = False
  for m in messages:
    if v.getValue() < m:
      v.setValue(m)
      changed = True
  if changed:
    for each outneighbor w:
      sendMessage(w, v.getValue())
  else:
    voteToHalt()
```



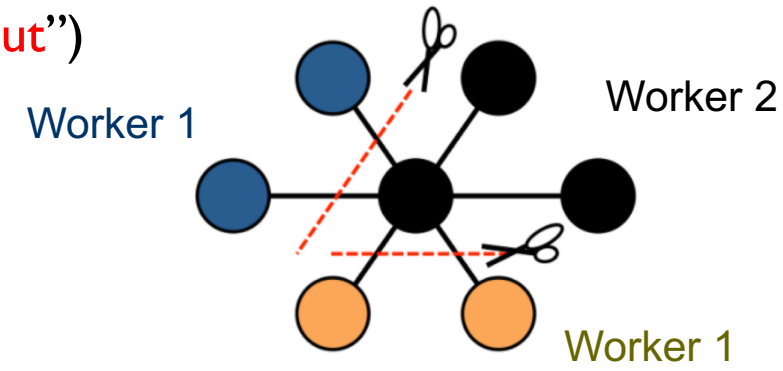
Example: Computing Max Value

```
Compute(v, messages):
  changed = False
  for m in messages:
    if v.getValue() < m:
      v.setValue(m)
      changed = True
  if changed:
    for each outneighbor w:
      sendMessage(w, v.getValue())
  else:
    voteToHalt()
```



Pregel: Implementation

- Master & workers architecture
  - Vertices are hash partitioned (by default) and assigned to workers ("edge cut")



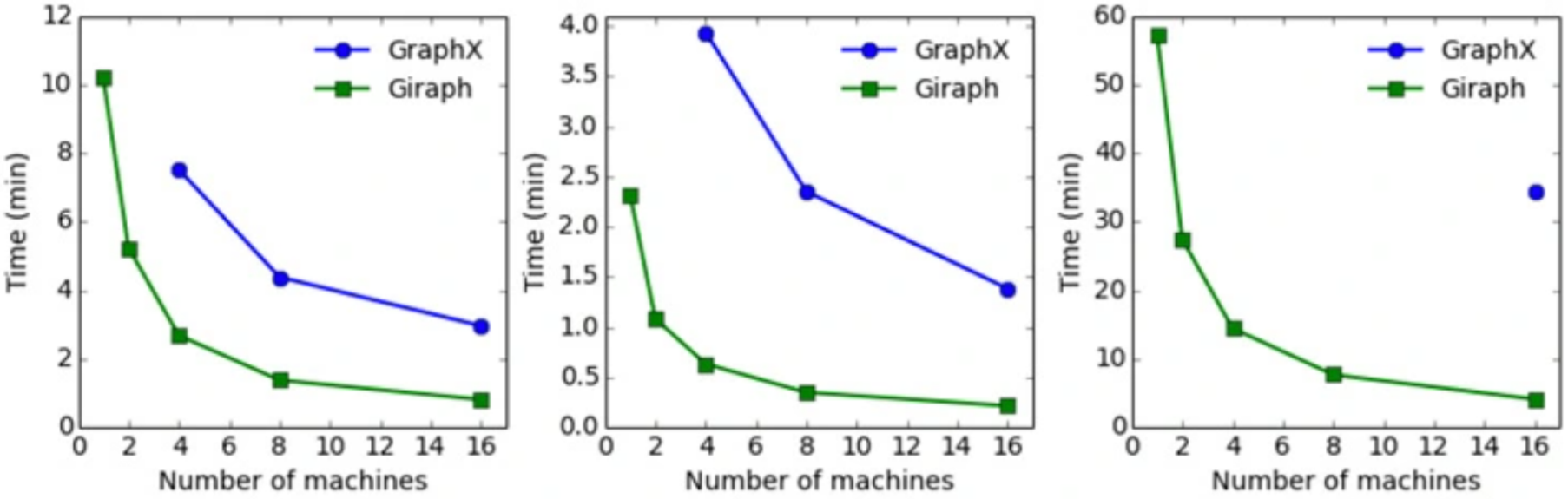
  - Each worker maintains the state of its portion of the graph in memory
  - Computations happen in memory
  - In each superstep, each worker loops through its vertices and executes `compute()`
  - Messages from vertices are sent, either to vertices on the same worker, or to vertices on different workers (buffered locally and sent as a batch to reduce network traffic)

Efficiency Comparison (on ~ trillion-scale Facebook social graph)

FACEBOOK Engineering

POSTED ON OCTOBER 19, 2016 TO CORE DATA, DATA INFRASTRUCTURE

A comparison of state-of-the-art graph processing systems



Giraph Architecture

- Giraph originated as the open-source counterpart to *Pregel*, with several features beyond the basic Pregel model.
- Employs Hadoop's Map phase to run computations
- Employs ZooKeeper (service that provides distributed synchronization) to enforce barrier waits

Example (Giraph): Max Value

```
package org.apache.giraph.examples;

public class MaxComputation extends BasicComputation<IntWritable, IntWritable,
NullWritable, IntWritable> {

 @Override
 public void compute(Vertex<IntWritable, IntWritable, NullWritable> vertex,
                     Iterable<IntWritable> messages) throws IOException {

   boolean changed = false;
   for (IntWritable message : messages) {
     if (vertex.getValue().get() < message.get()) {
       vertex.setValue(message);
       changed = true;
     }
   }
   if (getSuperstep() == 0 || changed) {
     sendMessageToAllEdges(vertex, vertex.getValue());
   }
   vertex.voteToHalt();
 }
}
```

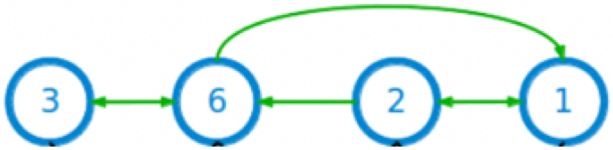
vertex id, vertex data
edge data, message type

process messages
from previous superstep

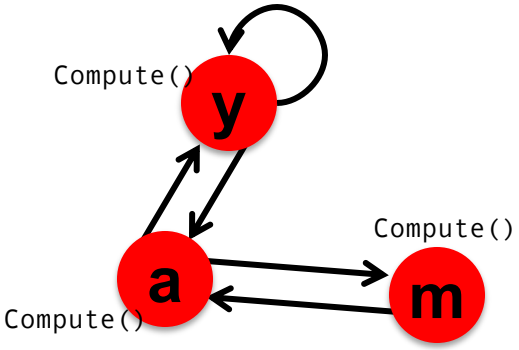
maximum changes

reactivation only
after incoming message

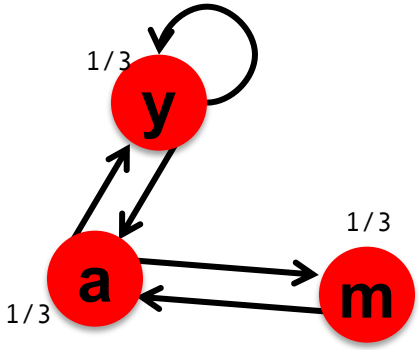
at start or after change,
message connected vertices



PageRank in Pregel

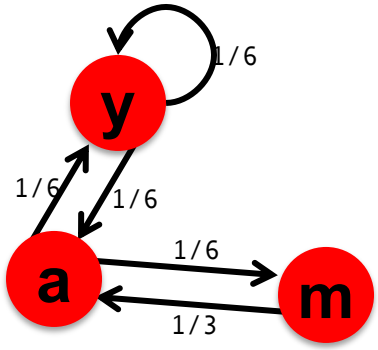


Superstep 1:
Invoke Compute()



Superstep 1:
Aggregate messages from neighbors to compute PageRank update

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



Superstep 1:
Send message to outgoing neighbors

Repeat this execution till Superstep reaches 30

PageRank in Pregel

```
class PageRankVertex : public Vertex<double, void, double> {
public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() = 0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Compute sum of incoming messages

PageRank update

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

Send outgoing messages

Stop after fixed no. of iterations

Source: Malewicz et al. (2010) Pregel: A System for Large-Scale Graph Processing. SIGMOD.

Acknowledgements

- Amir H. Payberah - Large Scale Graph Processing - Pregel, GraphLab, and Xstream
- https://engineering.fb.com/2016/10/19/core-data/a-comparison-of-state-of-the-art-graph-processing-systems/
- https://spark.apache.org/docs/latest/graphx-programming-guide.html
- Claudia Hauff - Big Data Processing (https://chauff.github.io/documents/bdp/graph_giraph.pdf)