CS4225/CS5425 Big Data Systems for Data Science

Graphs and PageRank

Ai Xin
School of Computing
National University of Singapore
aixin@comp.nus.edu.sg

School of Computing

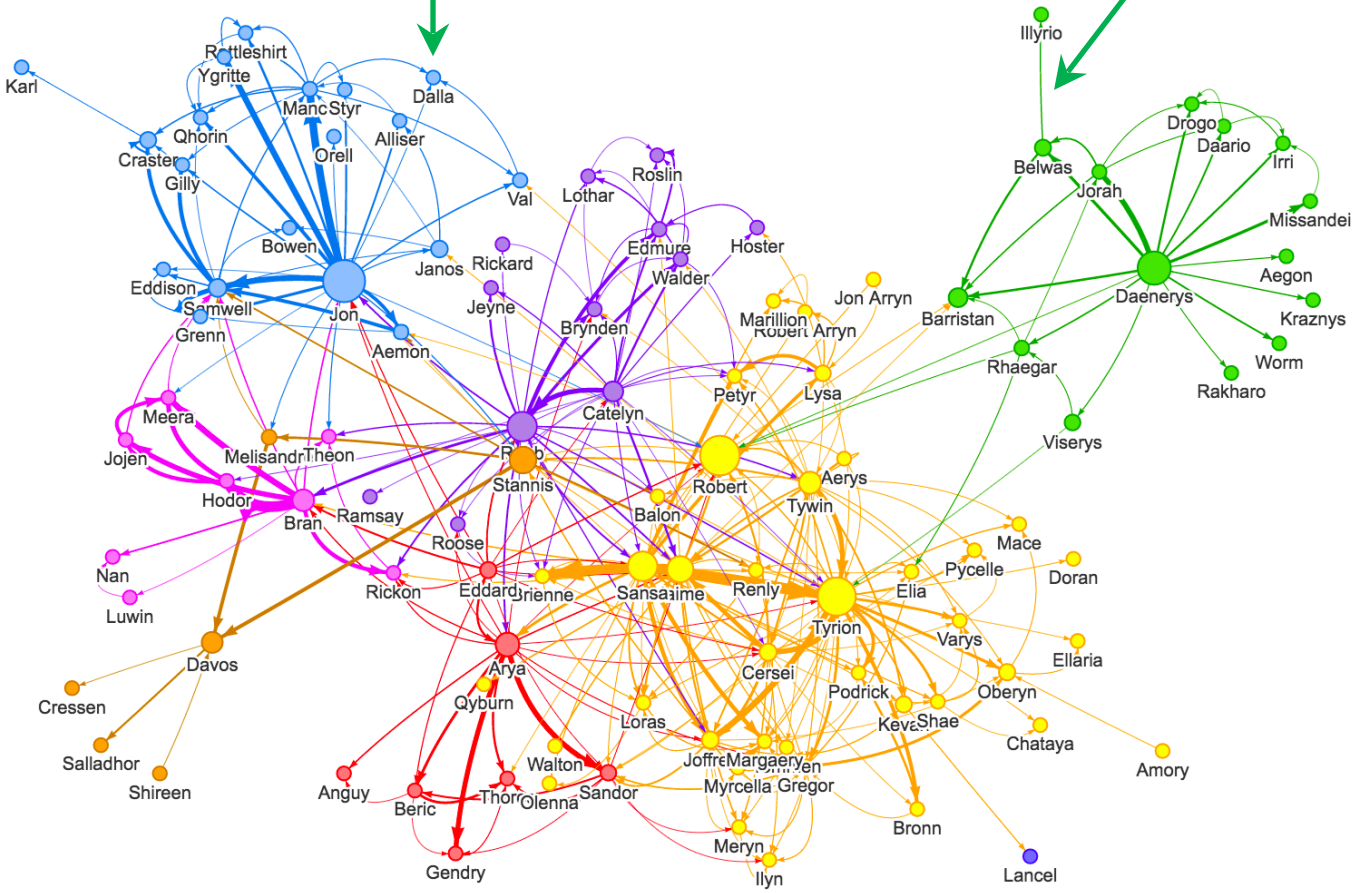
Today's Plan

- Graphs: Introduction
- Simplified PageRank
  - Flow Formulation
  - Random Walk Formulation
- PageRank with Teleports
- Topic Sensitive PageRank
- PageRank Implementation

Graph Data

Nodes (e.g. people)

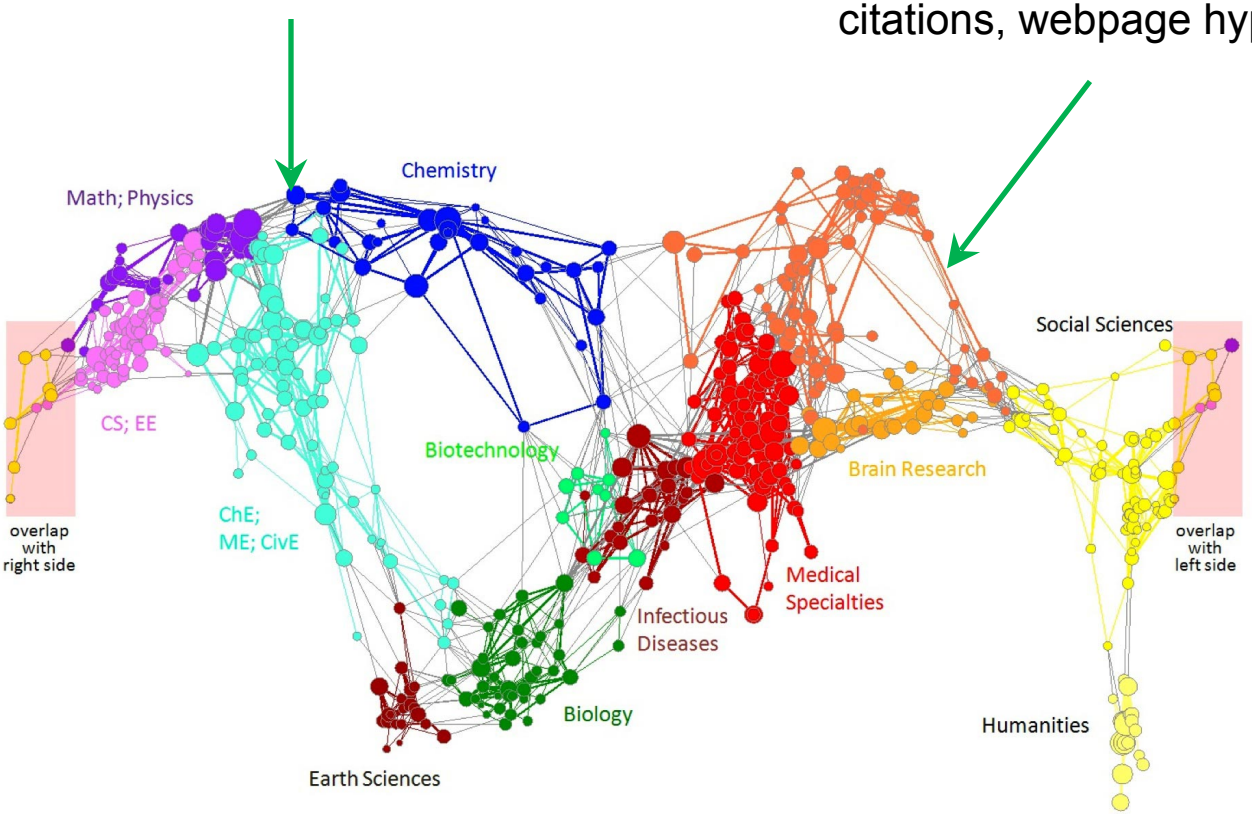
Edges represent relationships



Graph Data: Information Networks

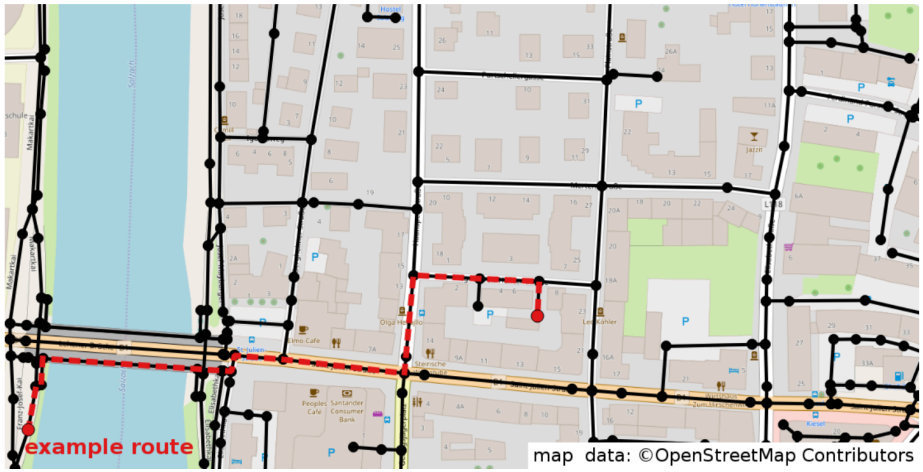
Nodes represent objects
(in this case, journals)

Edges represent relationships
(in this case, citations)
- Can be *undirected* (e.g. friendship) or *directed* (e.g. citations, webpage hyperlinks)



Citation networks and Maps of science
[Börner et al., 2012]

Graph Data: Traffic Networks



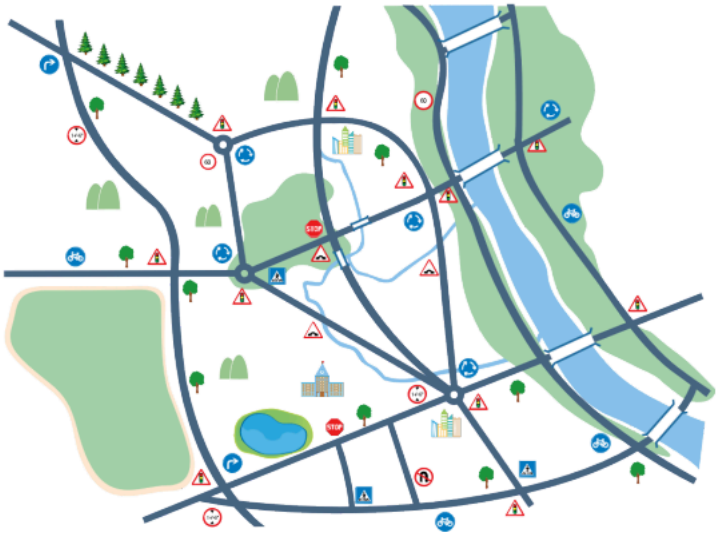


Figure 1: How can we predict how risky each road intersection is, based on nearby road features?

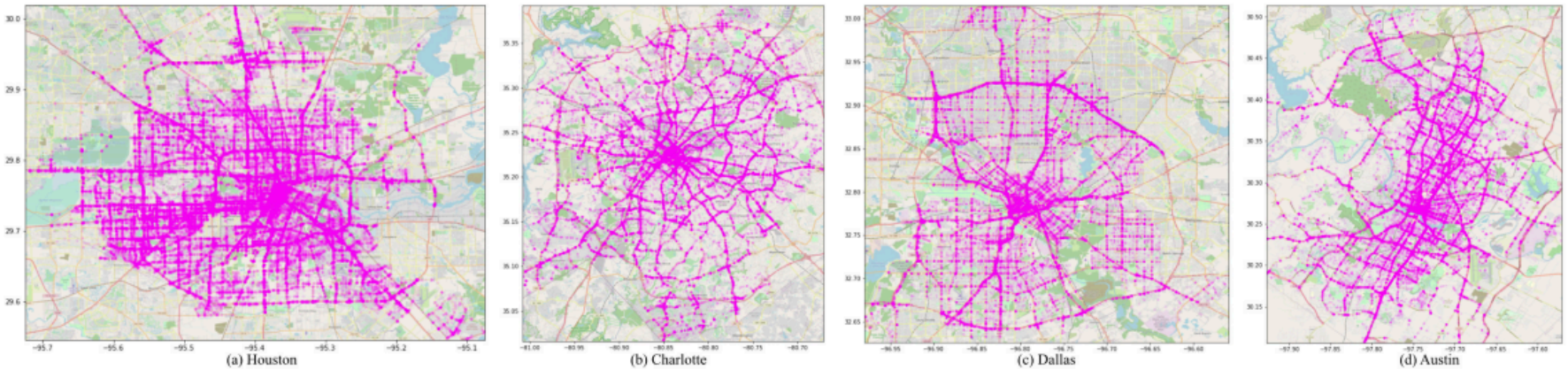
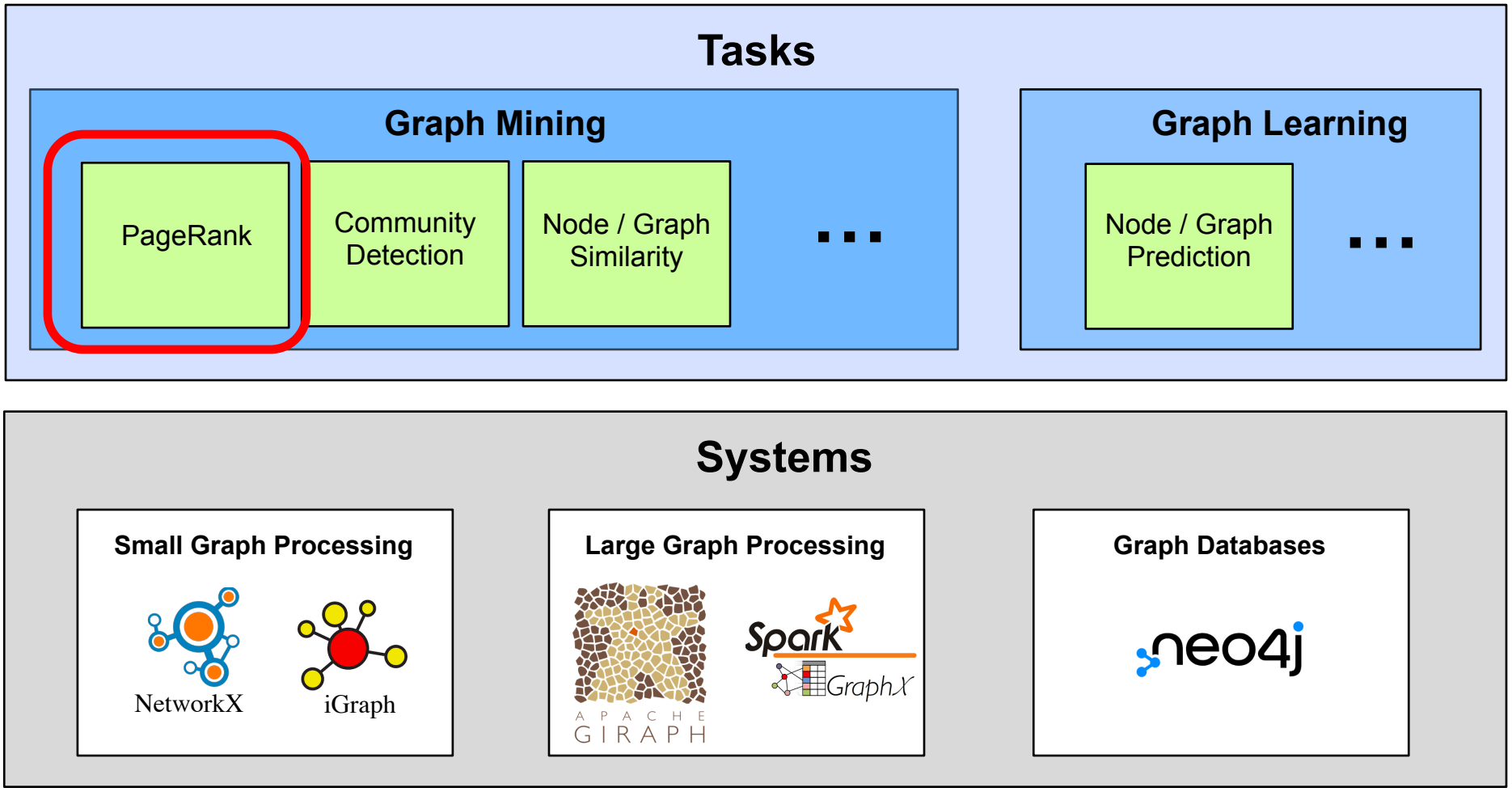


Figure 3: Traffic accident locations of Houston, Charlotte, Dallas, and Austin.

Overview: Graph Processing

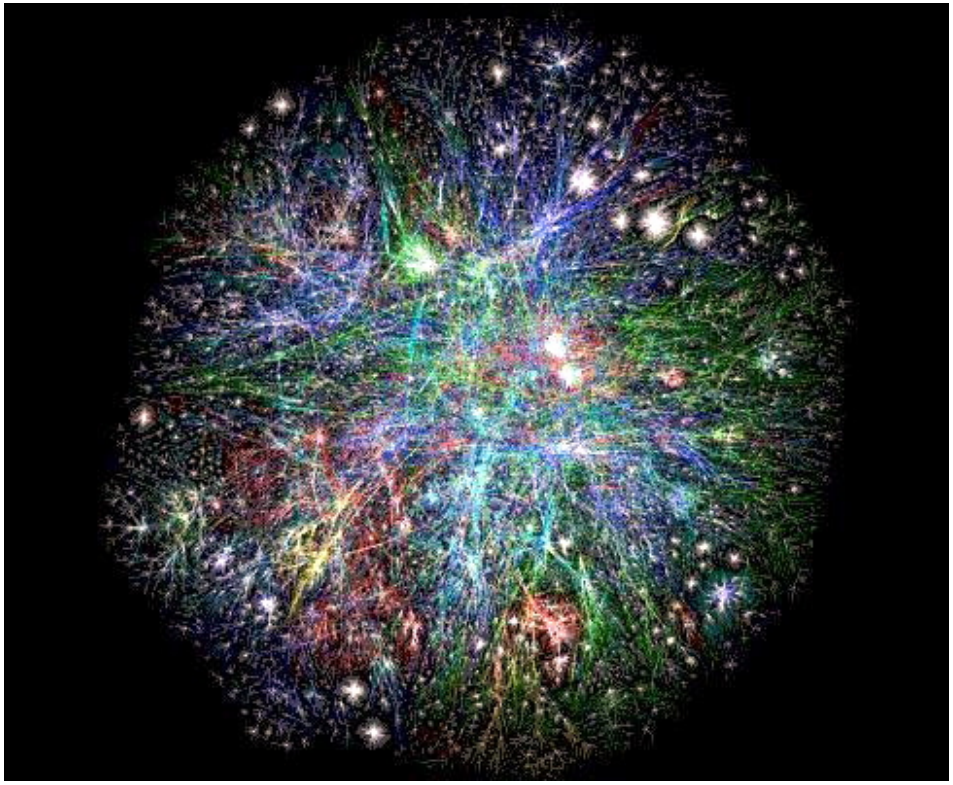


Today's Plan

- Graphs: Introduction
- **Simplified PageRank**
  - **Flow Formulation**
  - ~~Random Walk Formulation~~
- PageRank (with Teleports)
- Topic Sensitive PageRank
- PageRank Implementation

Web as a Graph

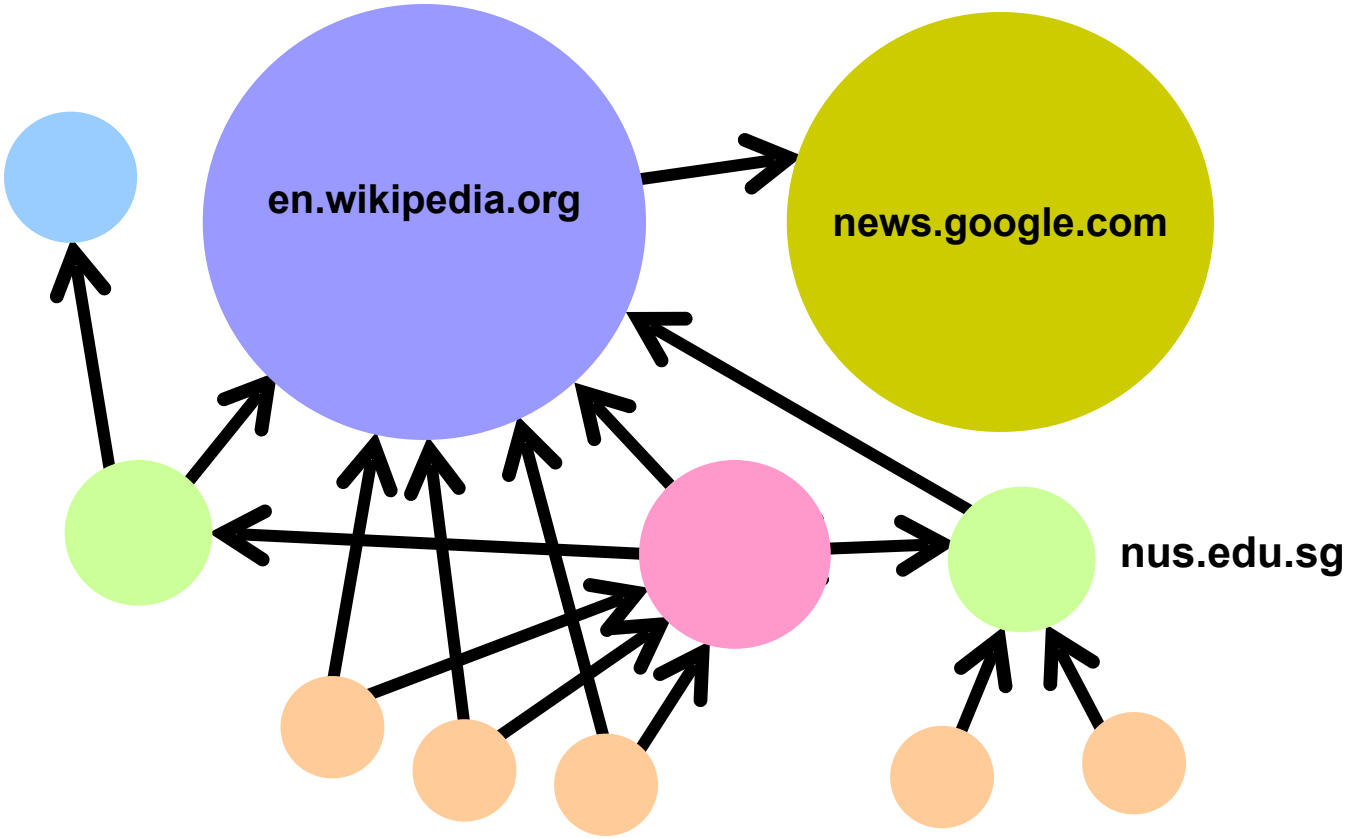
- **Web as a directed graph:**
  - **Nodes: Webpages**
  - **Edges: Hyperlinks**



Graph of the World Wide Web (http://www.bordalierinstitute.com/ (The Bordalier Institute))

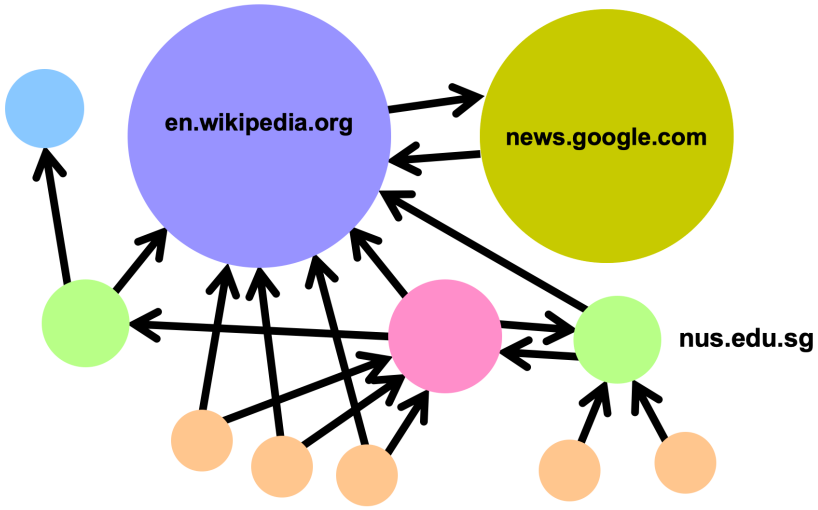
PageRank: Ranking Pages on the Web

- **All web pages are not equally "important":** www.joe-schmoe.com vs. en.wikipedia.org
  - Measuring the **importance** of pages is necessary for many web-related tasks (e.g. search, recommendation)
  - PageRank-like methods are also used in many other applications (bioinformatics, etc.)



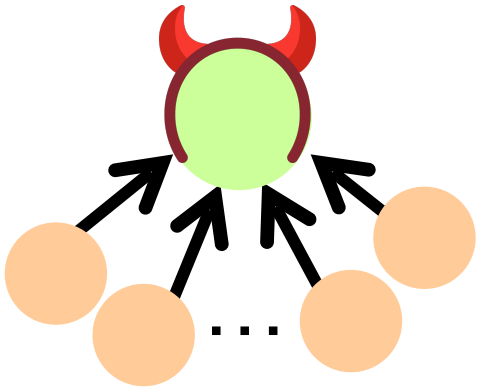
Links as Votes

- **Idea: Links as votes**
  - Page is more important if it has **more in-links**
    - Assume that incoming links are harder to manipulate. For example, anyone can create an out-link from their page to en.wikipedia.edu, but it is hard to get en.wikipedia.edu to create a link to their page
- **Think of in-links as votes**



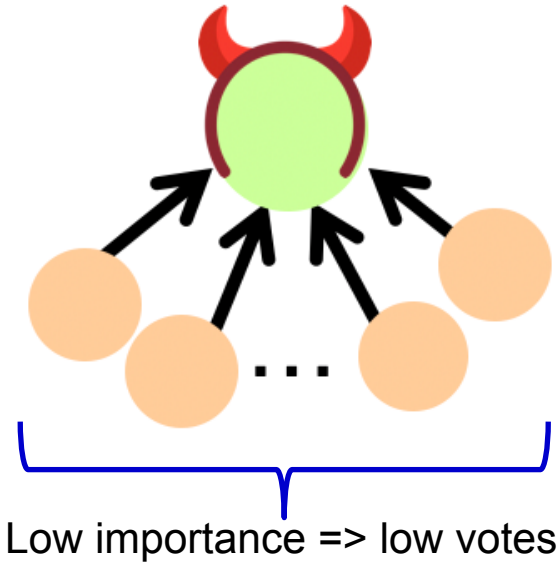
Links as Votes

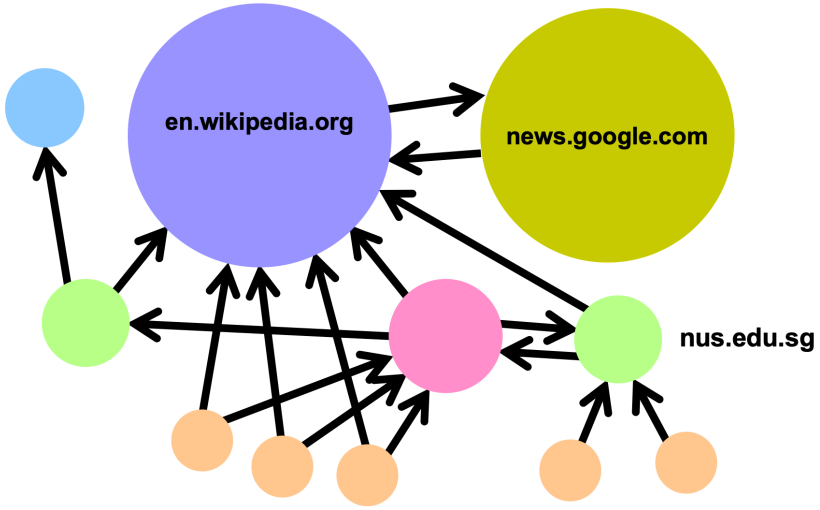
- **Think of in-links as votes**
- Naïve solution: What if we rank each page based on its *number* of in-links?
- **Problem:** malicious user can create a huge number of 'dummy' web pages, to link to their one page, to drive up its rank!



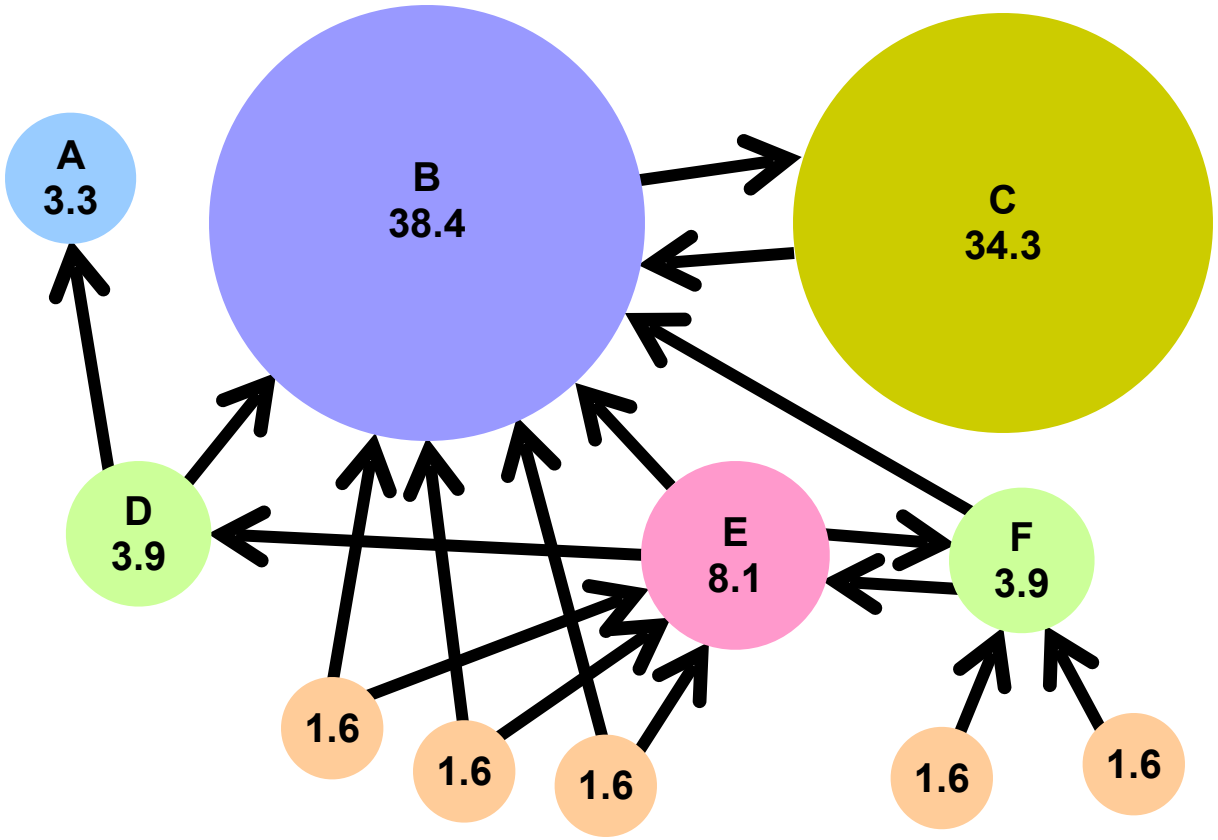
Links as Votes

- **Solution:** make the number of 'votes' that a page has proportional to its own importance. Then, as long as the 'dummy' pages themselves have low importance, they will contribute little votes as well.
  - Links from important pages count more – recursive definition!
  - This is the main idea of PageRank, which recursively defines the importance of a page based on the importance of the pages linking to it



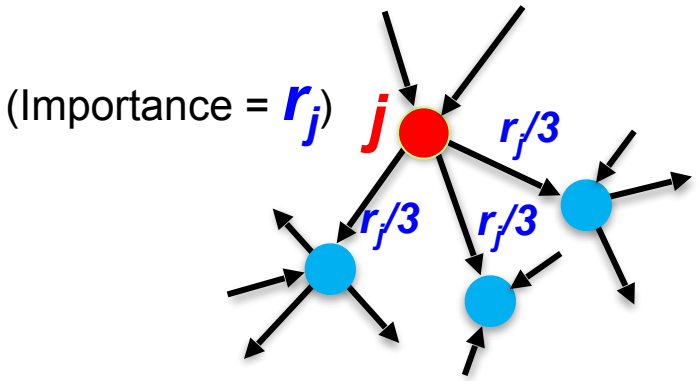


Example: PageRank Scores



‘Voting’ Formulation

- Each link’s vote is proportional to the **importance** of its source page
- For each page ***j***, define its “importance” (or rank) as r_j
- If page ***j*** with importance r_j has ***n*** out-links, each link gets r_j / n votes



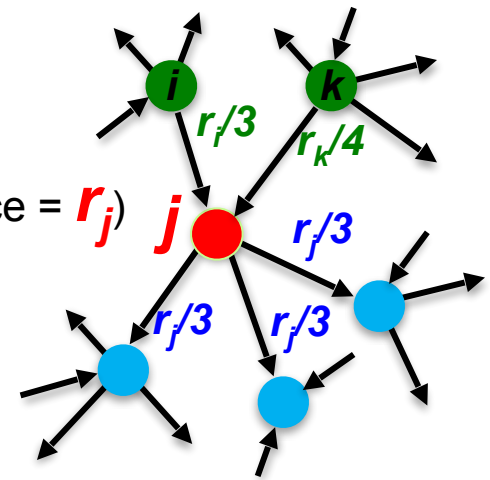
‘Voting’ Formulation

- Each link’s vote is proportional to the **importance** of its source page
- For each page ***j***, define its “importance” (or rank) as r_j
- 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
  - Analogy: each page receives a certain amount of candies from its incoming neighbors. It distributes these candies evenly to its outgoing neighbors.

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

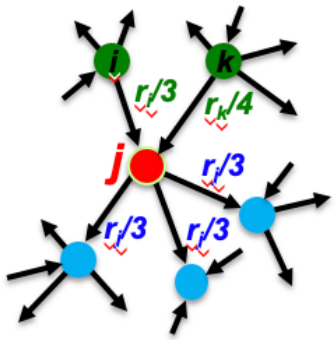
Importance of node ***j***

Importance node ***j*** receives from its incoming edges



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" or importance r_j for page j



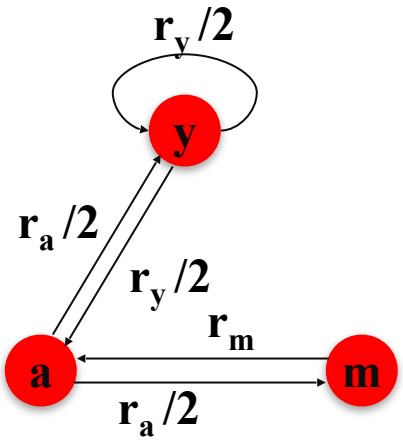
$r_j = r_i/3 + r_k/4$

Importance of j

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

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

The web in 1839

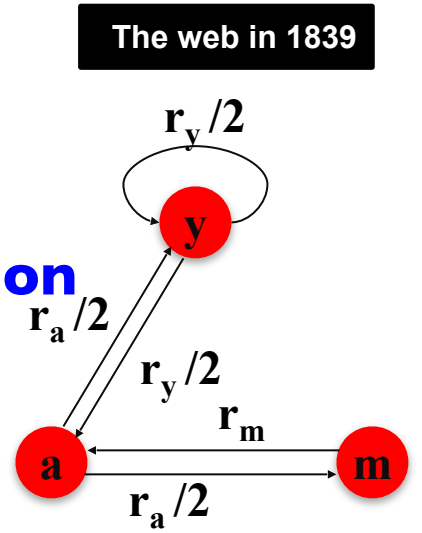


"Flow" equations:

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

"Simplified PageRank"

Solving the Flow Equations

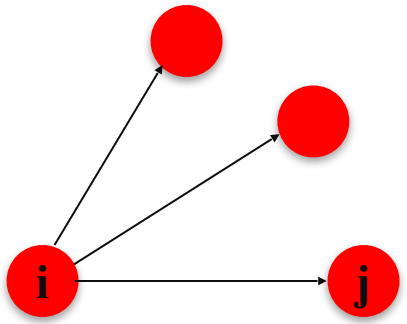


- **3 equations, 3 unknowns, 1 redundant equation**
  - No unique solution
  - All solutions are rescalings of each other
- **Additional constraint forces uniqueness:**
  - $r_y + r_a + r_m = 1$
  - **Solution** (via substitution, calculations are skipped)**:**

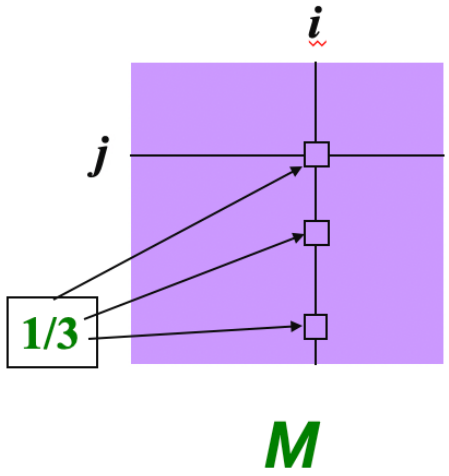
$$r_y = \frac{2}{5},\quad r_a = \frac{2}{5},\quad r_m = \frac{1}{5}$$

- Solving the equations through substitution or 'Gaussian elimination' works for small examples, but we need a better method for large web-size graphs
- **We need a new formulation!**

PageRank: Matrix Formulation



- **Stochastic adjacency matrix M**
  - Let page i has di out-links
  - If $i \rightarrow j$, then $Mji = \frac{1}{di}$ else $Mji = 0$
    - M is a **column stochastic matrix**
      - Columns sum to 1
- **Rank vector r:** vector with an entry per page
  - ri is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**



$$r = M \cdot r$$

This is equivalent to our earlier formulation:

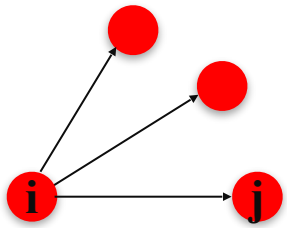
For all nodes j :

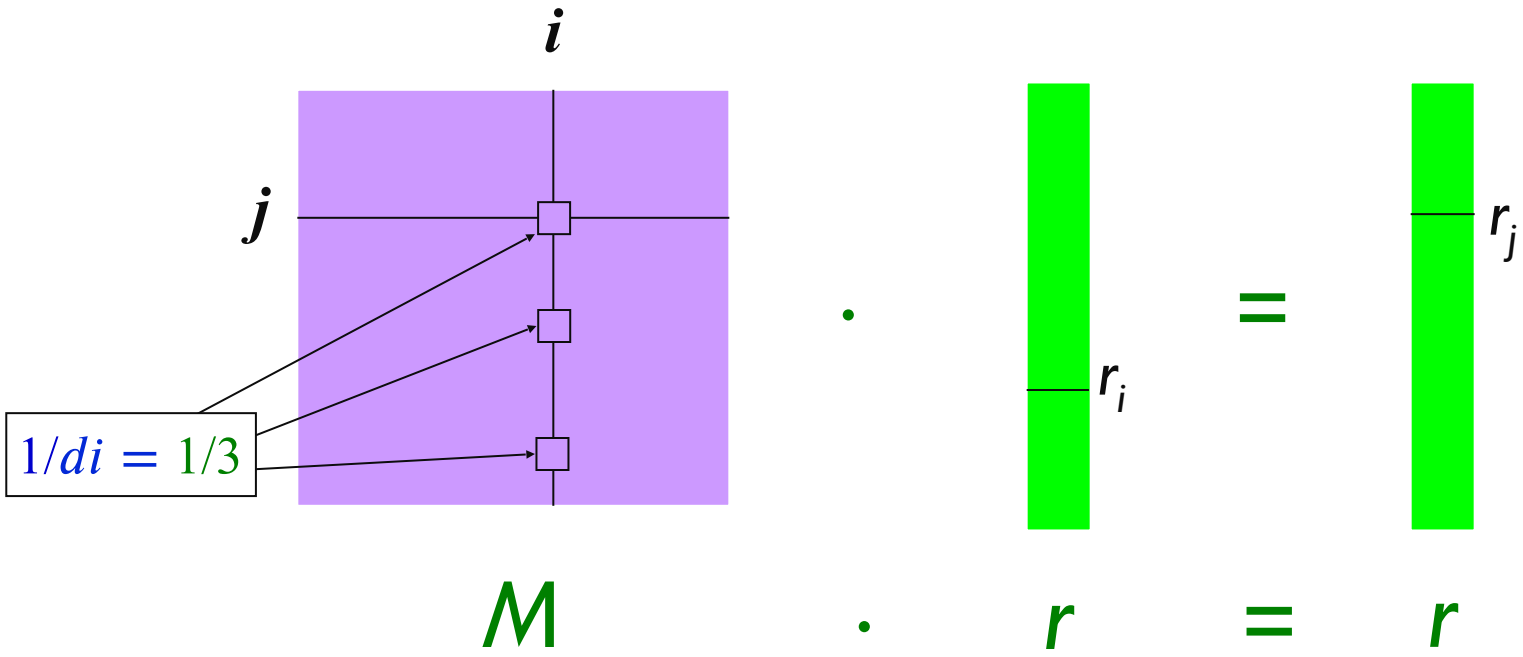
Why are the 2 formulations equivalent?

- **Remember the flow equation:**
- **Flow equation in the matrix form**

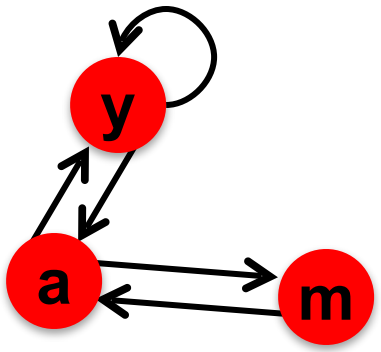
$$M \cdot r = r$$

- **Suppose page *i* links to 3 pages, including *j***





Example: Flow Equations & M



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r = M \cdot r$

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

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

Solving the Flow Equations

- **Given the flow equation:**

$$r = M \bullet r$$

- **We can efficiently solve for *r*!**
  The method is called Power iteration

Power Iteration Method

- **Given a web graph with *n* nodes, where the nodes are pages and edges are hyperlinks**

Power iteration: a simple iterative scheme

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

Equivalent!

d_i …. out-degree of node i

$|x|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm

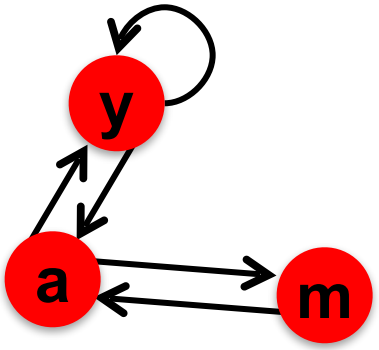
Can use any other vector norm, e.g., Euclidean

- **Intuitive interpretation of power iteration**: each node starts with equal importance (of 1/N). During each step, each node passes its current importance along its outgoing edges, to its neighbors.

Power Iteration: Example

Power Iteration:

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



$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

- **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 …

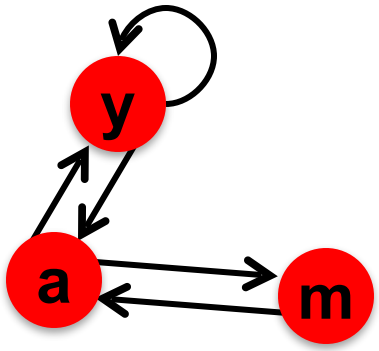
Today's Plan

- Graphs: Introduction
- **Simplified PageRank**
  - Flow Formulation
  - **Random Walk Formulation**
- PageRank (with Teleports)
- Topic Sensitive PageRank
- PageRank Implementation

Random Walk Interpretation

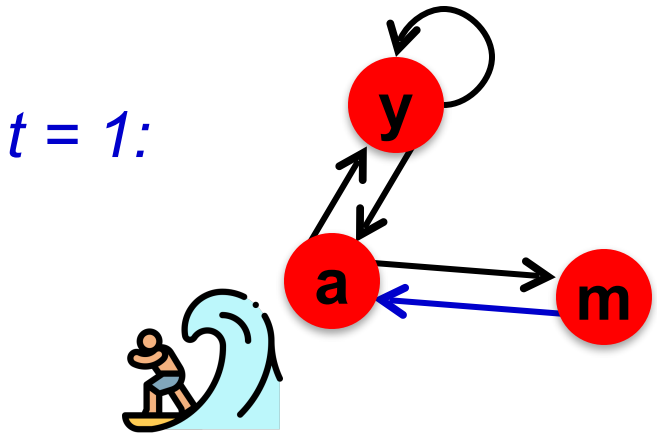
- Imagine a random web surfer:
  - At time t = 0, surfer starts on a random page
  - At any time t, surfer is on some page i
  - At time $t + 1$, the surfer follows an out-link from i uniformly at random
  - Process repeats indefinitely

t = 0:



Random Walk Interpretation

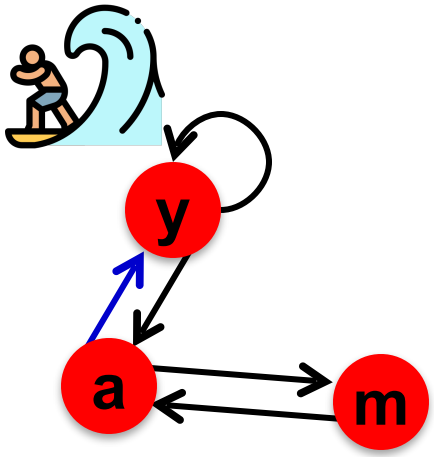
- Imagine a random web surfer:
  - At time $t = 0$, surfer starts on a random page
  - At any time t, surfer is on some page i
  - At time $t + 1$, the surfer follows an out-link from i uniformly at random
  - Process repeats indefinitely



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

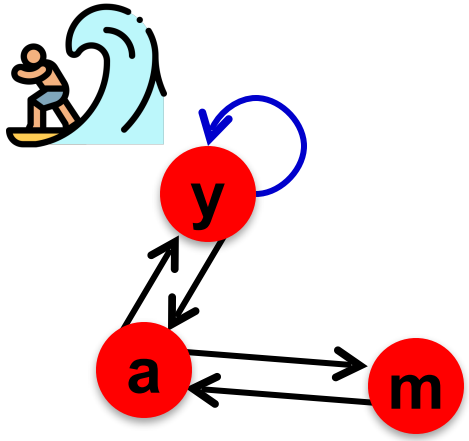
t = 2:



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

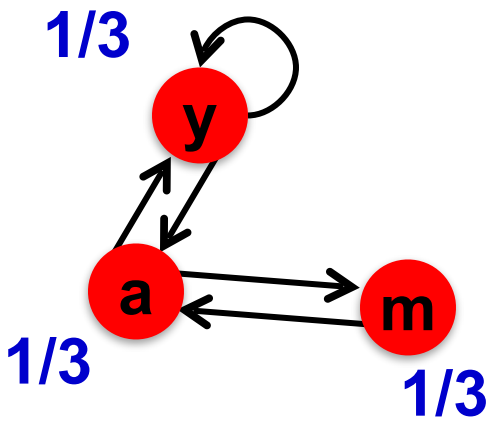
t = 3:



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
- 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

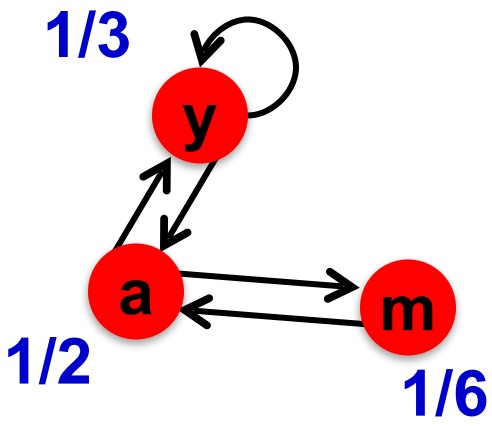
p(t) where t = 0:



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
- 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

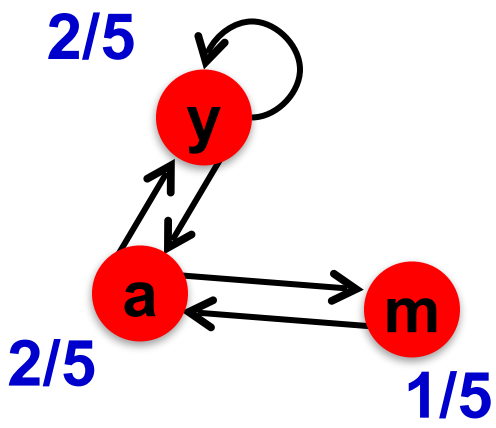
p(t) where t = 1:



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
- 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 $t \rightarrow \infty$, the probability distribution approaches a 'steady state' representing the long term probability that the random walker is at each node, which are the PageRank scores

p(t) where $t \rightarrow \infty$

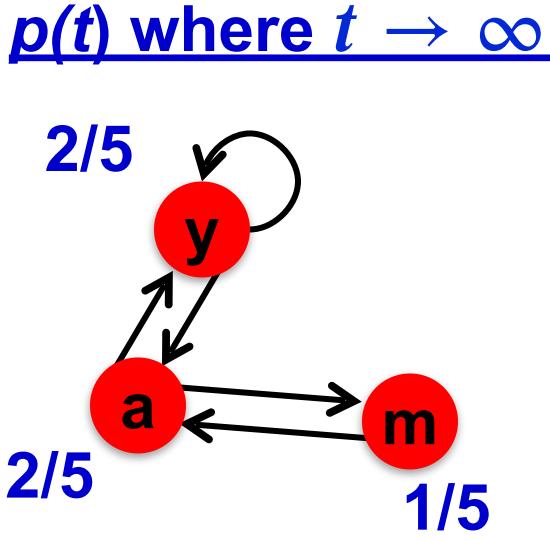


Equivalence between Random Walk and Flow Formulations

- Where is the surfer at time t+1?
  - Follows a link uniformly at random

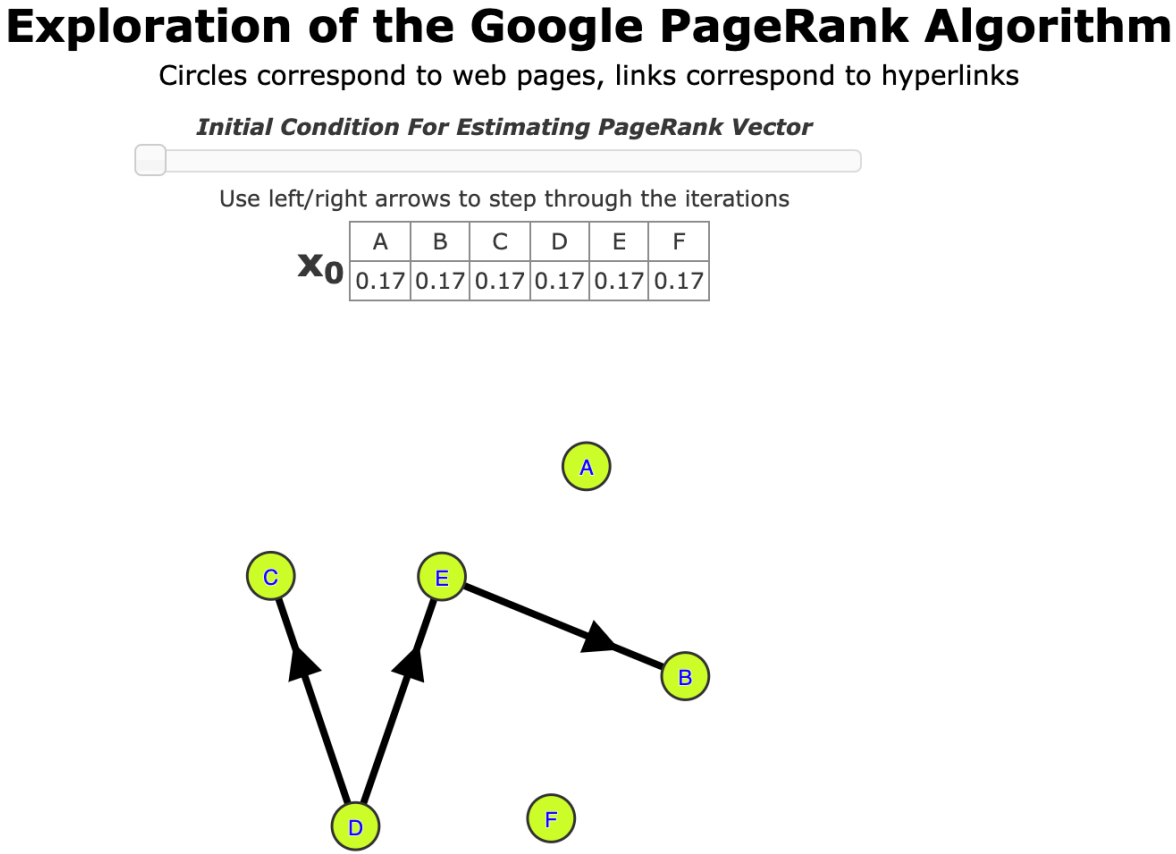
$$p(t+1) = M \cdot p(t)$$

- Suppose the random walk reaches a stationary state p_s: then $p_s = M \cdot p_s$
- In the previous 'flow' formulation of PageRank, the rank vector r was also defined by $r = M \cdot r$
  - Thus, the two (recursive) definitions are the same!



PageRank: Animation

- https://www.learnforeverlearn.com/pagerank/



Today's Plan

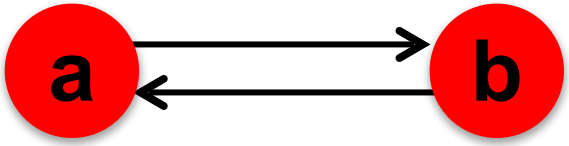
- Graphs: Introduction
- Simplified PageRank
  - Flow Formulation
  - Random Walk Formulation
- **PageRank (with Teleports)**
- Topic Sensitive PageRank
- PageRank Implementation

PageRank: Three Questions

or
equivalently

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

Does this converge?



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

r_a	1	0	1	0
r_b	0	1	0	1

=

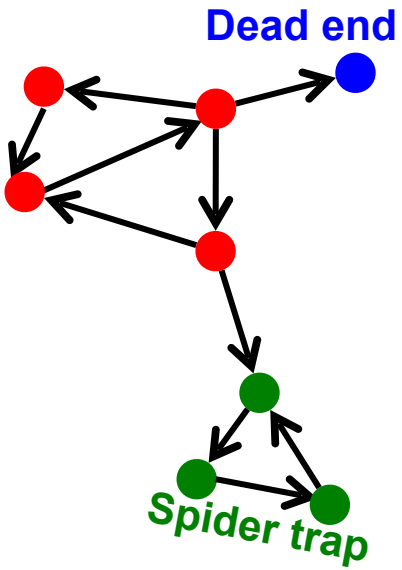
Iteration 0, 1, 2, …

Does it converge to what we want?

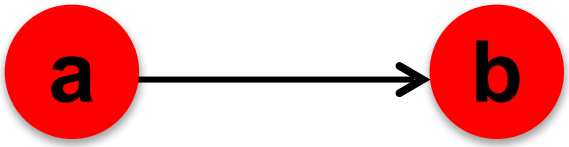
Answer: not always. 2 problems:

- **(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



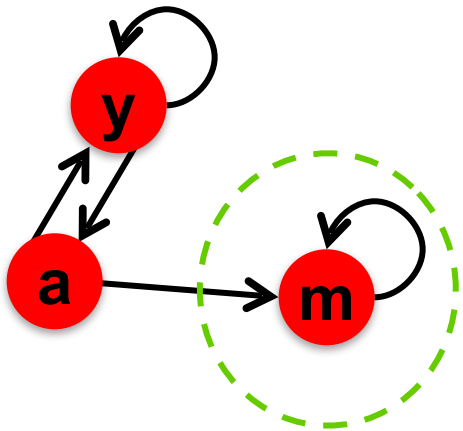
- **Example:**

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

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:**



$r_y = r_y/2 + r_a/2$

$r_a = r_y/2$

$r_m = r_a/2 + r_m$

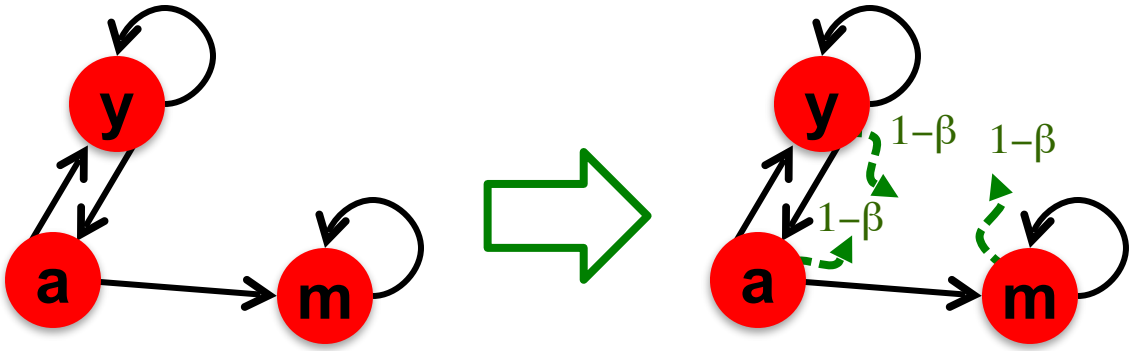
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

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

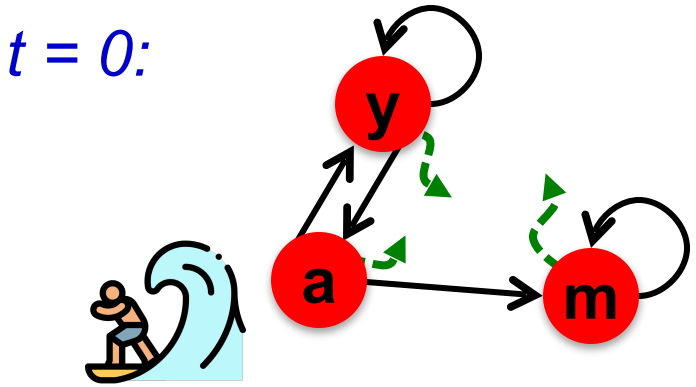
Solution: Teleports!

- **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!

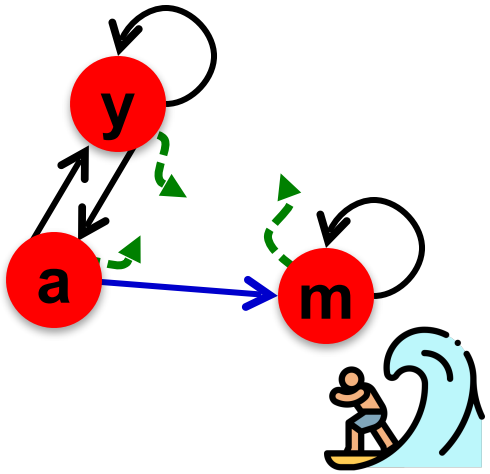
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9



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

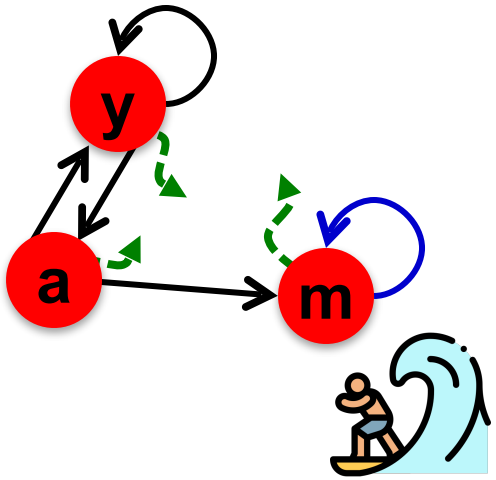
t = 1:



Solution: Teleports!

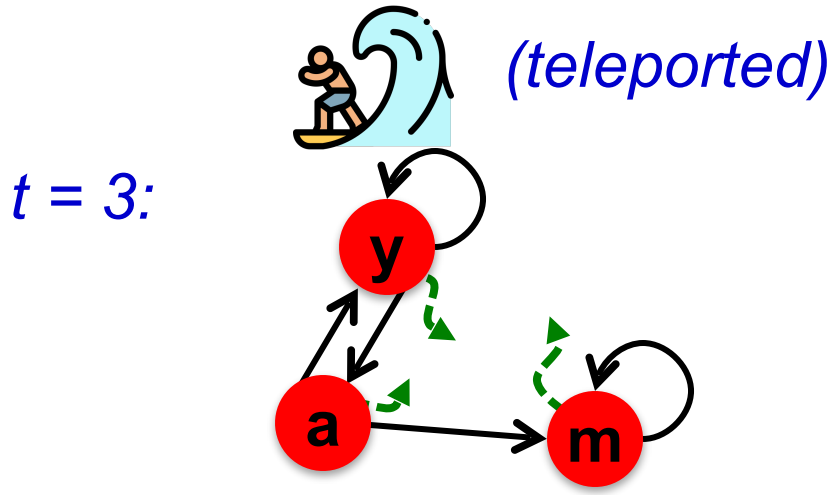
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9

t = 2:



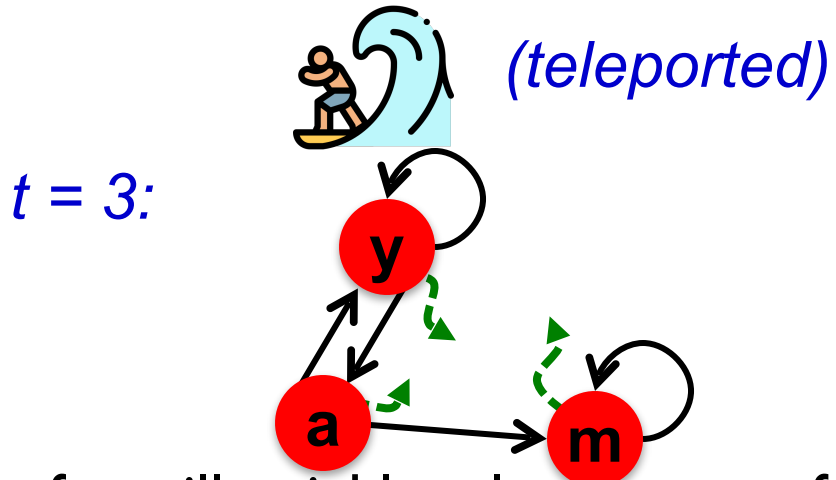
Solution: Teleports!

- **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!

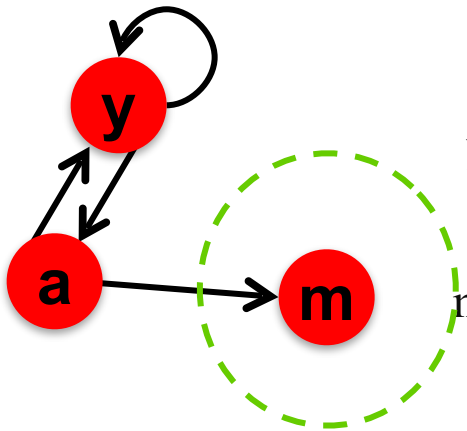
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9



- **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:**

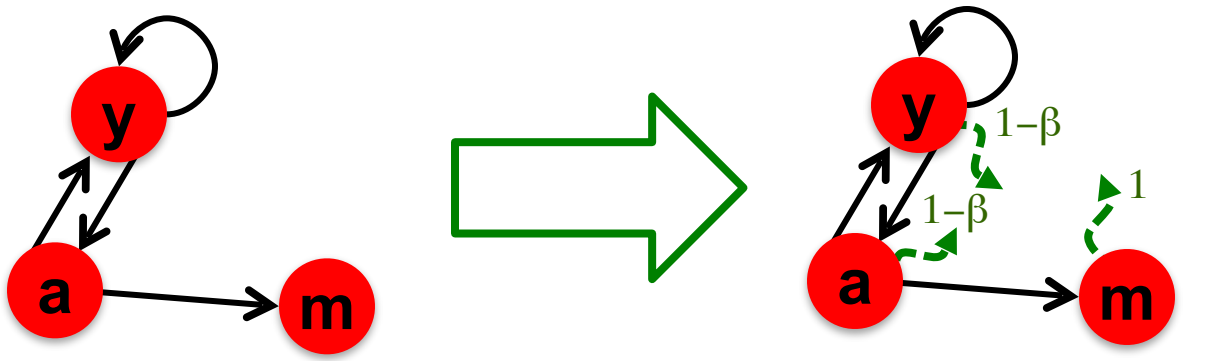
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

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
  - Equivalently, for each dead end **m** we can preprocess the random walk matrix **M** by making **m** connected to ***every node*** (including itself).



M =

	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** 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-β, jump to some random page

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

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

- *Compare to Simplified PageRank:*

Same as simplified PageRank

Teleport term

d_i … out-degree of node i

(If any dead ends exist, we assume that we have preprocessed by adding connections from them to every other node)

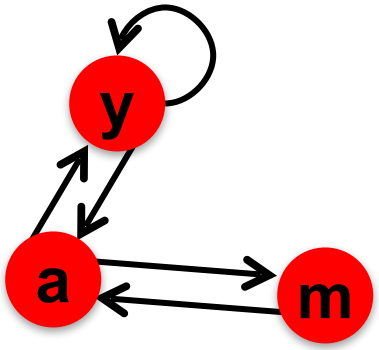
The Google Matrix

- **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***:

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



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

N by N matrix
where all entries are 1/N

	y	a	m
y	⅓	⅓	⅓
a	⅓	⅓	⅓
m	⅓	⅓	⅓

- **PageRank equation (matrix form): $r = A \cdot r$**
- In practice β = 0.8, 0.9 (5-10 steps on average before 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

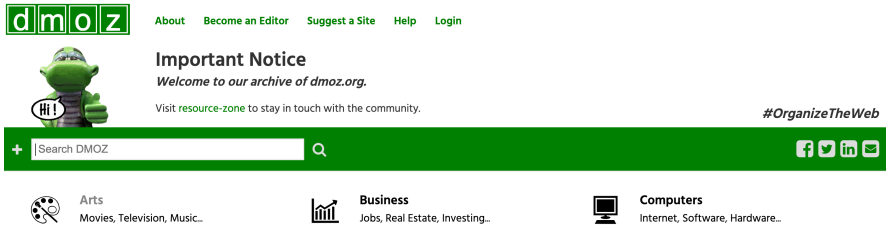
- Graphs: Introduction
- Simplified PageRank
  - Flow Formulation
  - Random Walk Formulation
- PageRank (with Teleports)
- **Topic Specific PageRank**
- PageRank Implementation

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



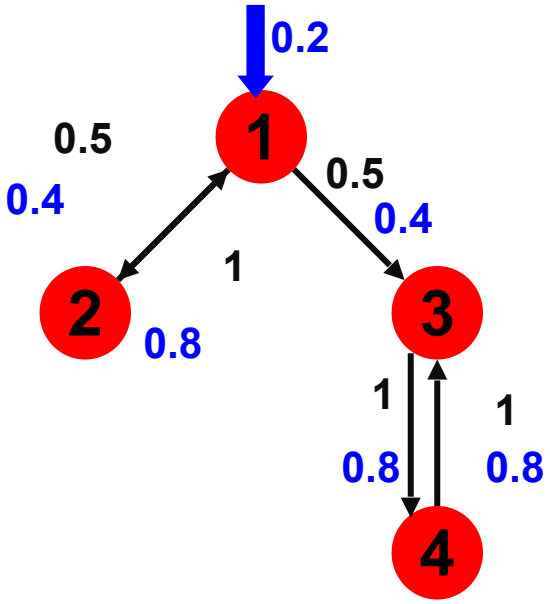
Matrix Formulation

- **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
  - **Could also assign different weights to pages!**
- **Compute as for regular PageRank:**
  - Multiply by ***M***, then add a vector
  - Maintains sparseness

Example: Topic-Specific PageRank



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

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

Probabilities (no teleport)

Probabilities (with teleport)

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]

Discovering the Topic Vector S

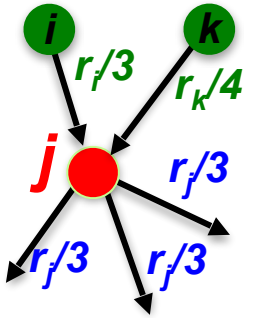
- **Create different PageRanks for different topics**
  - The 16 DMOZ top-level categories:
    - arts, business, sports,…
- **Which topic ranking to use?**
  - User can pick from a menu
  - Classify query into a topic
  - Can use the **context** of the query
    - E.g., query is launched from a web page talking about a known topic
    - History of queries e.g., "basketball" followed by "Jordan"
  - User context, e.g., user's bookmarks, …

Today's Plan

- Graphs: Introduction
- Simplified PageRank
  - Flow Formulation
  - Random Walk Formulation
- PageRank with Teleports
- Topic Sensitive PageRank
- PageRank Implementation

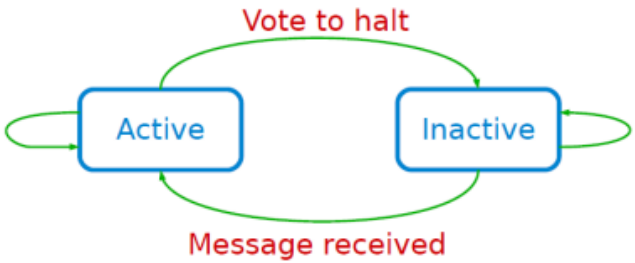
Characteristics of Graph Algorithms

- What are some common features of graph algorithms?
  - Local computations at each vertex
  - Passing messages to other vertex
- **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

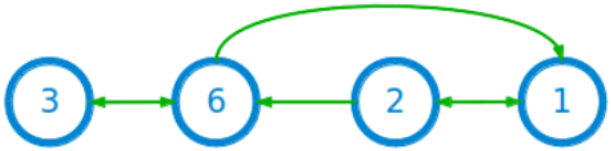
- Computation consists of a series of supersteps
- In each superstep, the framework invokes a user-defined function, `compute()`, for each vertex (conceptually in parallel)
- `compute()` specifies behavior at a single vertex *v* and a superstep *s*:
  - It can read messages sent to *v* in superstep *s* - 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

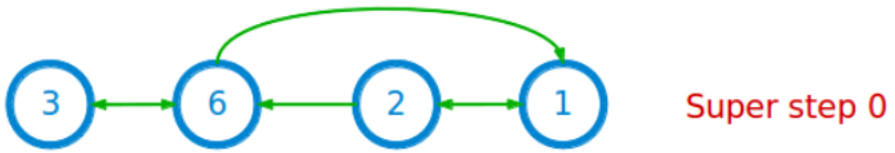
- **Task**: compute the maximum over the values in the vertices in this graph:



- **Summary of approach:**
  - Each vertex repeatedly sends its current value to its neighbors (as "messages"). Then each vertex updates its value to the maximum over its own value, and all the messages it receives. This process continues until all vertex values stop changing.

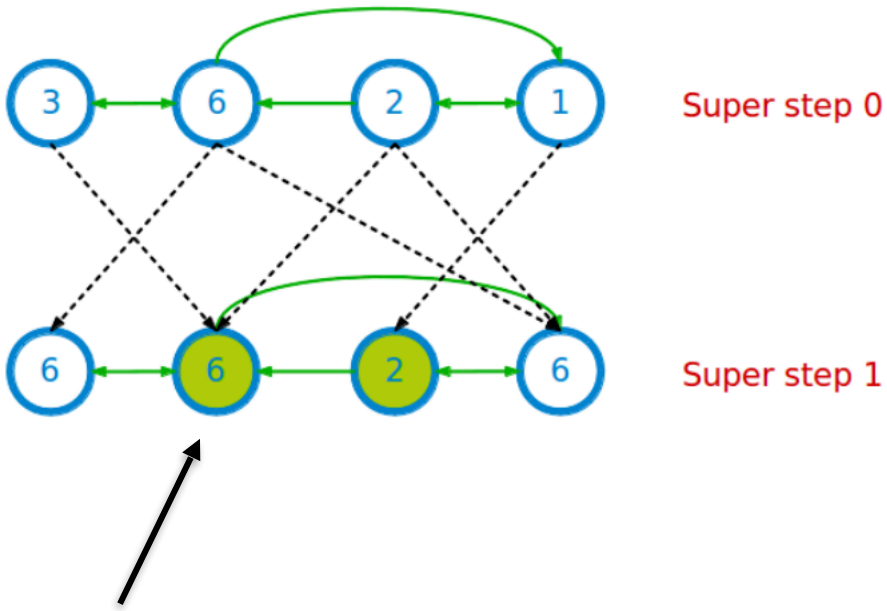
Example: Computing Max Value

```
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()
```



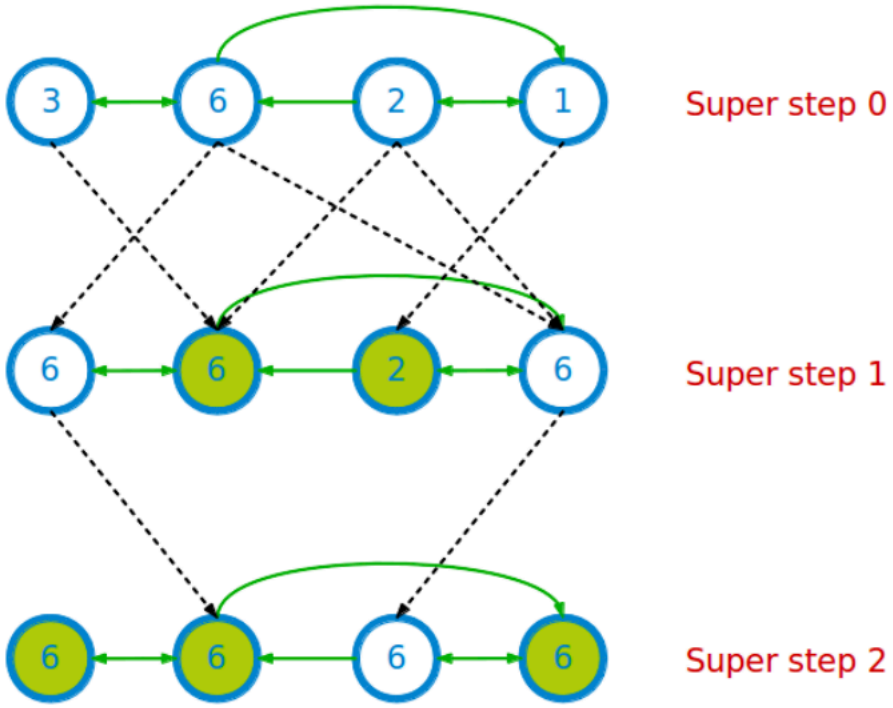
Example: Computing Max Value

```
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()
```



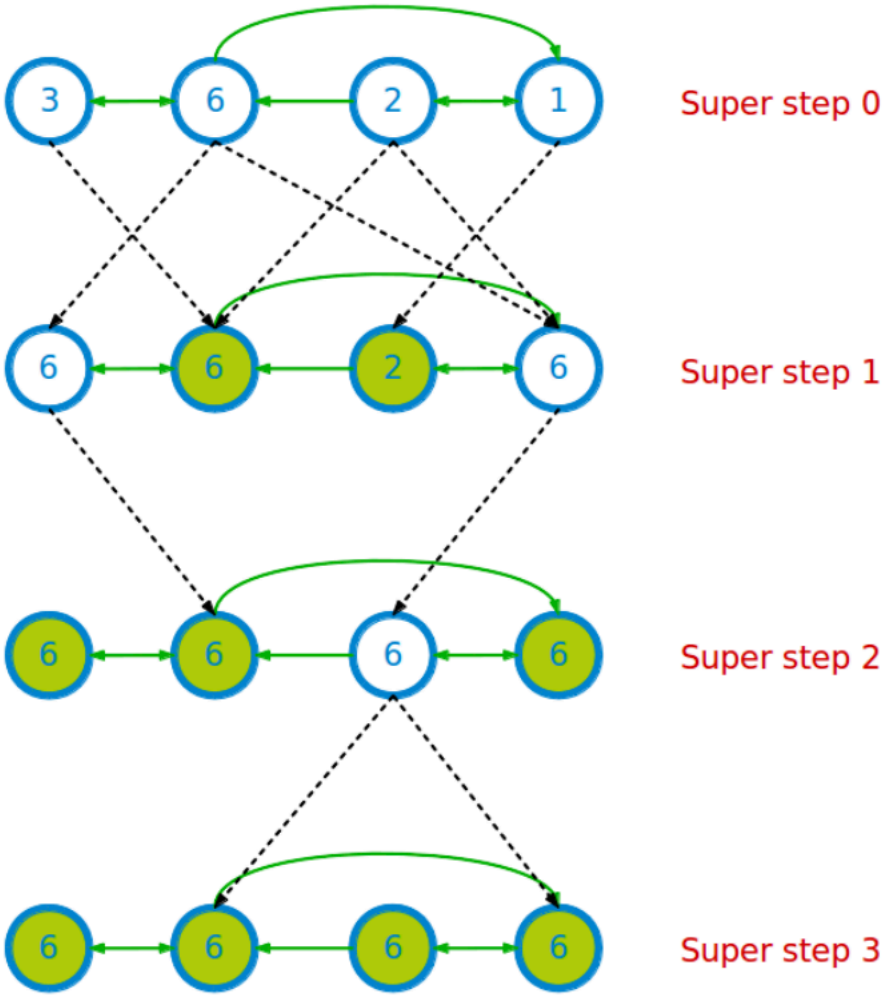
Example: Computing Max Value

```
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()
```



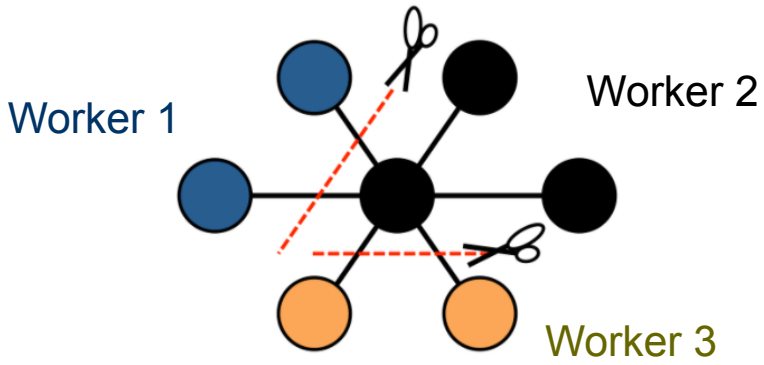
Example: Computing Max Value

```
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)

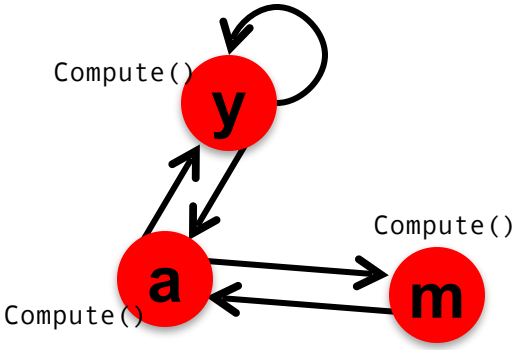
Pregel: Implementation

- Fault tolerance
  - Checkpointing to persistent storage
  - Failure detected through heartbeats
  - Corrupt workers are reassigned and reloaded from checkpoints

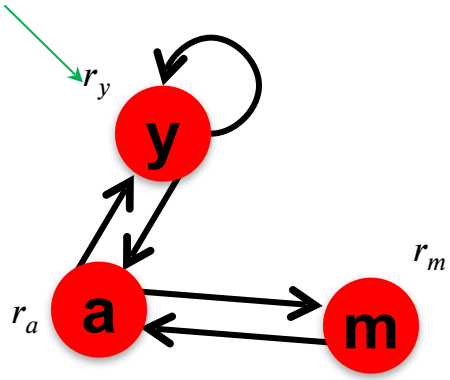
PageRank in Pregel

Node's current values, computed by summing messages from the previous Superstep and computing the PageRank update equation below.

Outgoing messages are computed by dividing each node's value equally among its outgoing neighbours (thus obtaining $\frac{r_i}{d_i}$)

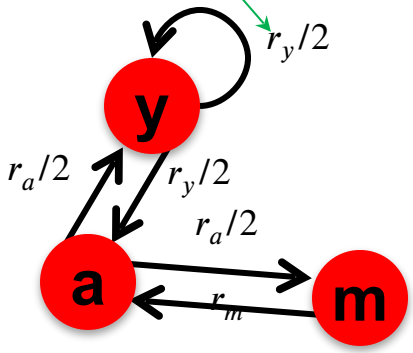


Superstep 1:
Invoke Compute()



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

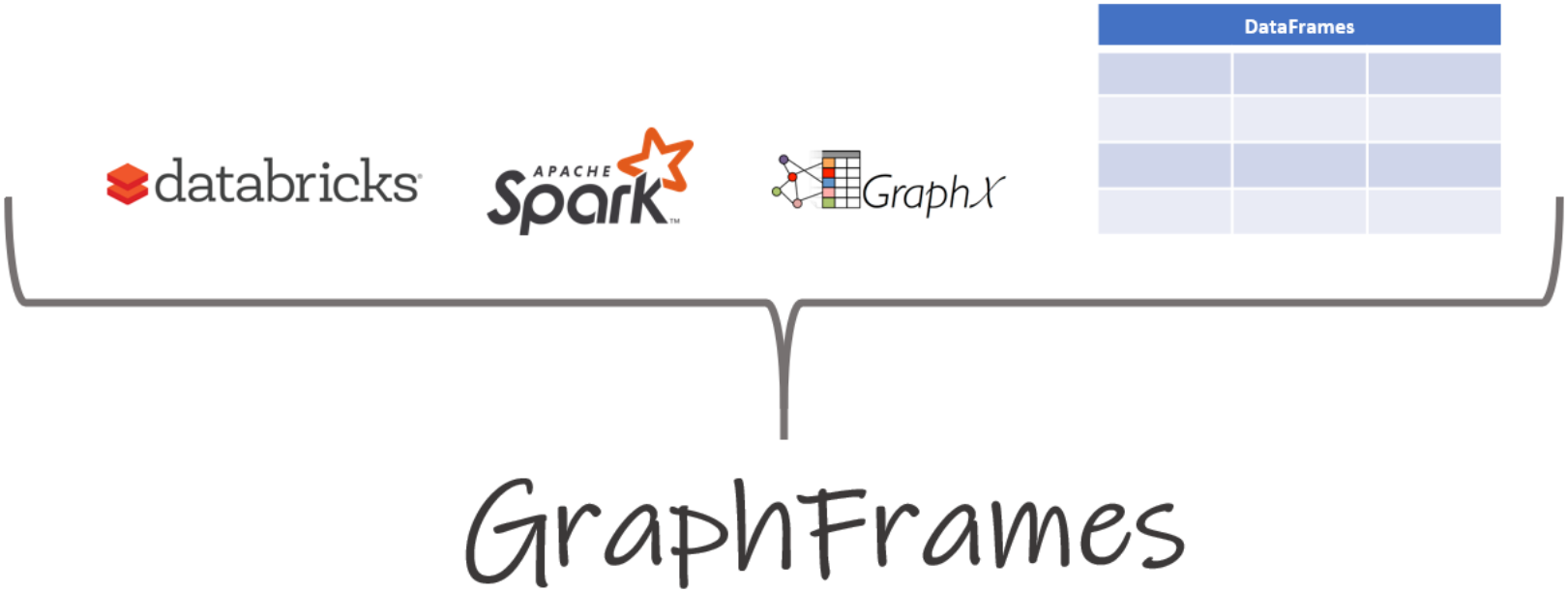
$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta)\frac{1}{N}$$

Send outgoing messages

Stop after fixed no. of iterations

Note: this algorithm implicitly assumes that the nodes' values (*MutableValue()) have been correctly initialized based on the initialization scheme that the user wants. In practice, you would generally do the initialization during superstep 0.

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


databricks
APACHE Spark
GraphX
DataFrames
GraphFrames

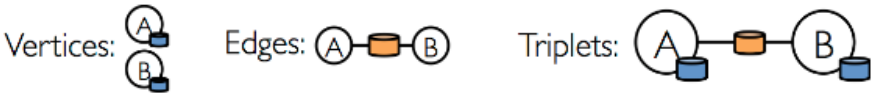
Spark for Graphs

- Spark is more efficient in iterative computation than Hadoop
- Integration of record-oriented and graph-oriented processing
- GraphX: Extends RDDs to Resilient Distributed Property Graphs
- Property Graphs:
  - Present different views of the graph (vertices, edges, triplets)
  - Support map-like operations
  - Support distributed Pregel-like aggregations
- Graphframe: Extends GraphX to provide a DataFrame API and support for Spark's different language bindings

Spark Implementation on weighted pagerank

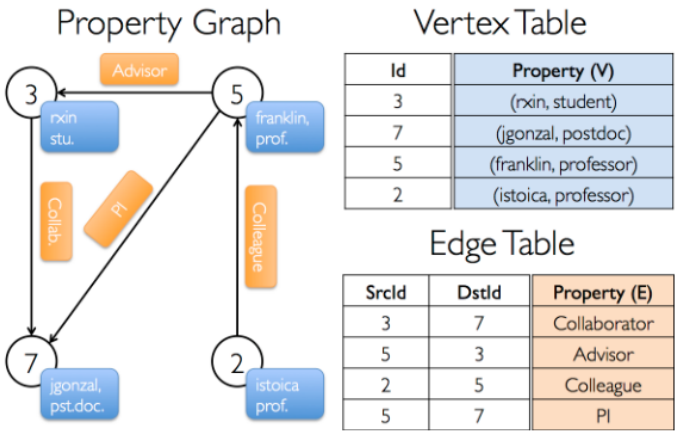
- weighted page rank algorithm: PR = 0.15+0.85*sum(PR*weight)
- Stage 1: Join

```
CREATE VIEW triplets AS
SELECT s.Id, d.Id, s.P, e.P, d.P
FROM edges AS e
JOIN vertices AS s JOIN vertices AS d
ON e.srcId = s.Id AND e.dstId = d.Id
```



- Stage 2: Group-By

```
SELECT t.dstId, 0.15+0.85*sum(t.srcP*t.eP)
FROM triplets AS t GROUP BY t.dstId
```



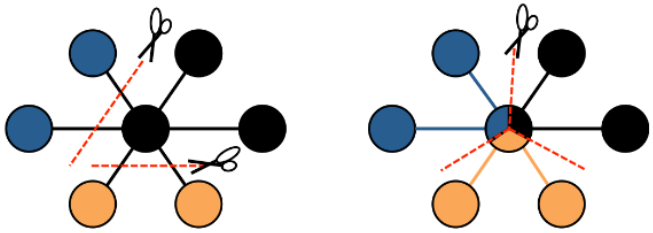
Vertex Table

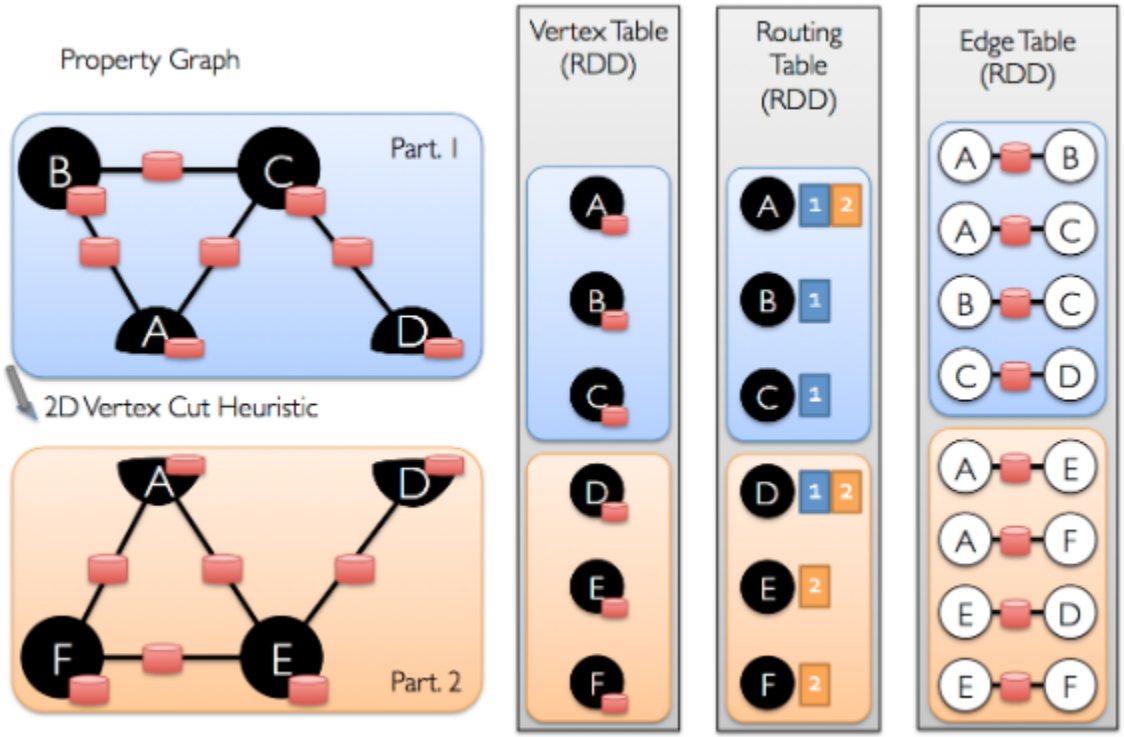
Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

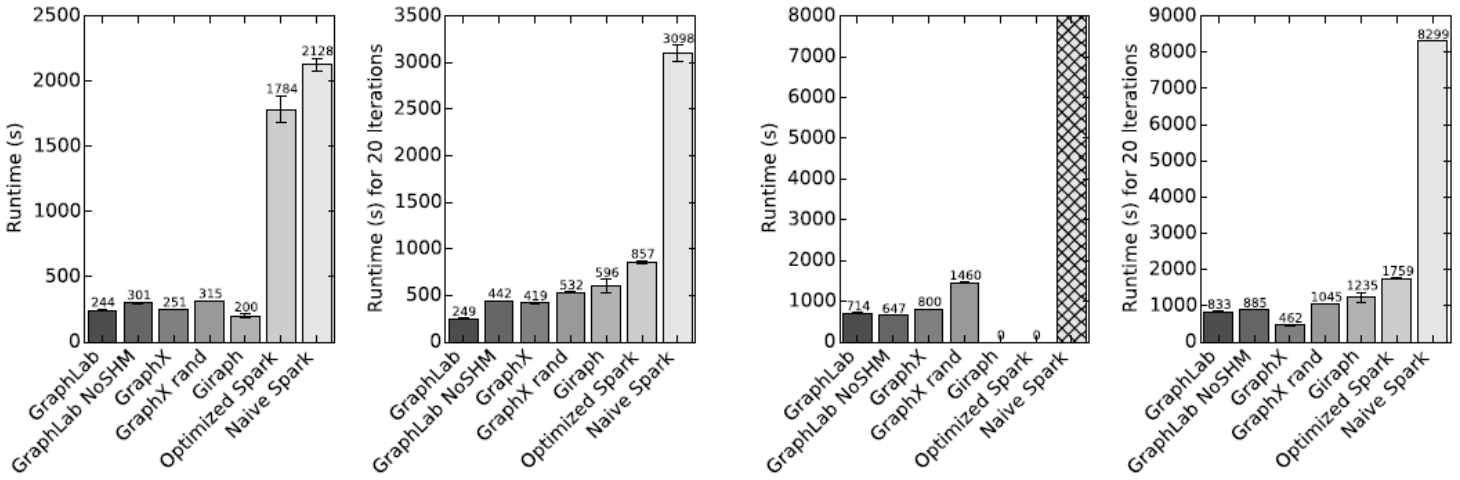
Spark uses Vertex Cut to partition the graph





Performance Comparison

- GraphX Paper 2015
  - GraphX can achieve performance parity compared with specialized graph systems



(a) Conn. Comp. Twitter (b) PageRank Twitter (c) Conn. Comp. uk-2007-05* (d) PageRank uk-2007-05

Figure 7: **System Performance Comparison.** (c) Spark did not finish within 8000 seconds, Giraph and Spark + Part. ran out of memory.

GraphX: Graph Processing in a Distributed Dataflow Framework, 2015

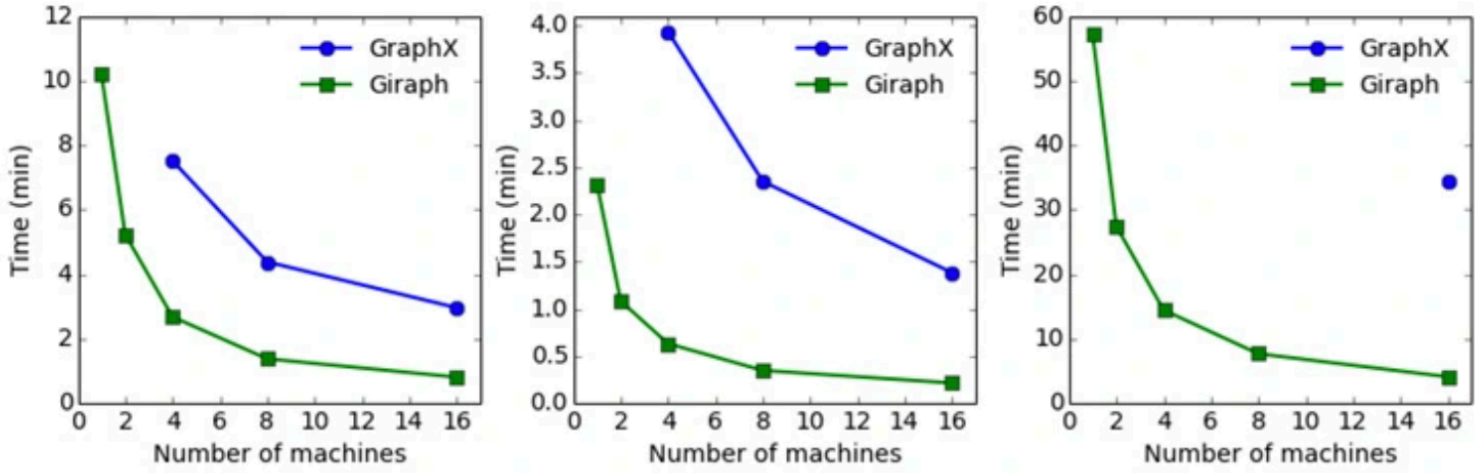
Performance Comparison

- Facebook 2016
  - Giraph: an open source implementation of Pregel by Facebook
  - 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



Acknowledgements

- CS4225 slides by He Bingsheng and Bryan Hooi
- Claudia Hauff - Big Data Processing (https://chauff.github.io/documents/bdp/graph_giraph.pdf) mmds.org – Mining of Massive Datasets - Jure Leskovec, Anand Rajaraman, Jeff Ullman
- https://jakobmiksch.eu/post/openstreetmap_routing/
- https://www.edrawmax.com/online-map-maker.html
- https://learnforeverlearn.com/pagerank/
- https://neo4j.com/