MAP REDUCE - GRAPHS

Adapted from the slides by Dr. Zareen Alamgir

Content obtained from many sources notably Jimmy Lin, Jeff Ullman, Jerome Simeon, Juliana Freire notes

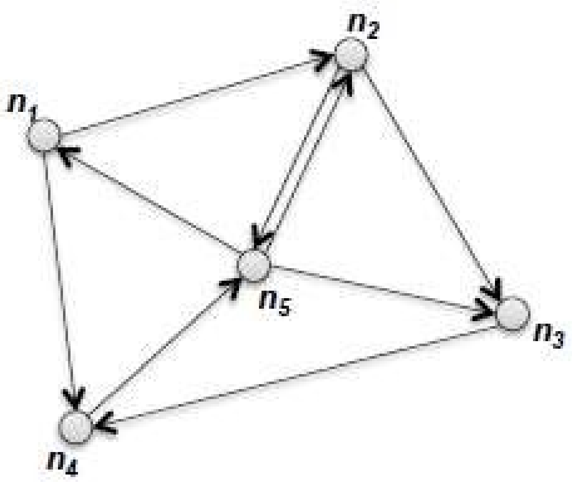
Graphs and MapReduce

- A large class of graph algorithms involve:
  - Performing computations at each node: based on node features, edge features, and local link structure
  - Propagating computations: "traversing" the graph

- Key questions:
  - How do you represent graph data in MapReduce?
  - How do you traverse a graph in MapReduce?

Representing Graphs

- G = (V, E)
- Two common representations
  - Adjacency matrix
  - Adjacency list



	n_1	n_2	n_3	n_4	n_5
n_1	0	1	0	1	0
n_2	0	0	1	0	1
n_3	0	0	0	1	0
n_4	0	0	0	0	1
n_5	1	1	1	0	0

adjacency matrix

n_1	$[n_2, n_4]$
n_2	$[n_3, n_5]$
n_3	$[n_4]$
n_4	$[n_5]$
n_5	$[n_1, n_2, n_3]$

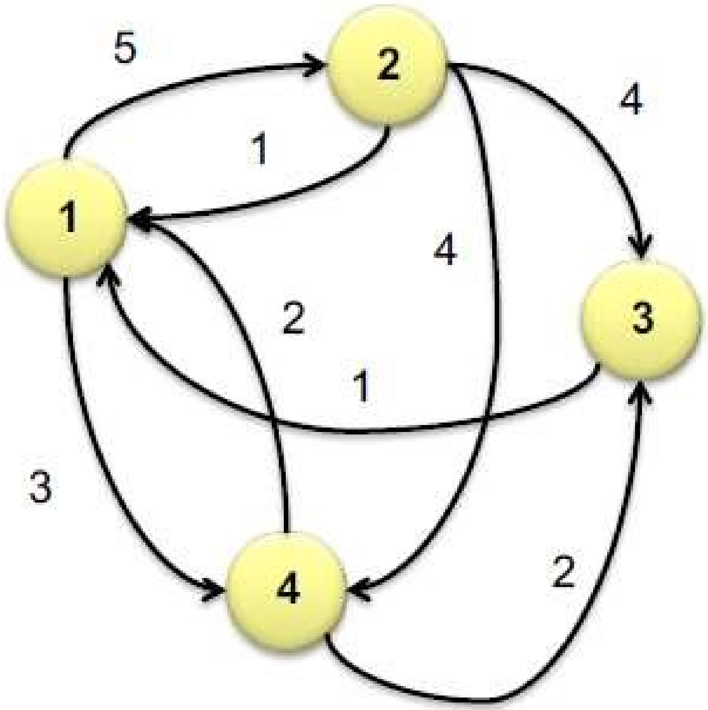
adjacency lists

Adjacency Matrices

Represent a graph as an *n* x *n* square matrix *M*

- $n = |V|$
- M_{ij} = the weight of link from node *i* to *j*

	1	2	3	4
1	0	5	0	3
2	1	0	4	4
3	1	0	0	0
4	3	0	2	0



Adjacency Matrices: Critique

- Advantages:
  - Amenable to mathematical manipulation
  - Iteration over rows and columns corresponds to computations on outlinks and inlinks
- Disadvantages:
  - Lots of zeros for sparse matrices
  - Lots of wasted space

Adjacency Lists

- Take adjacency matrices… and throw away all the zeros

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

→

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

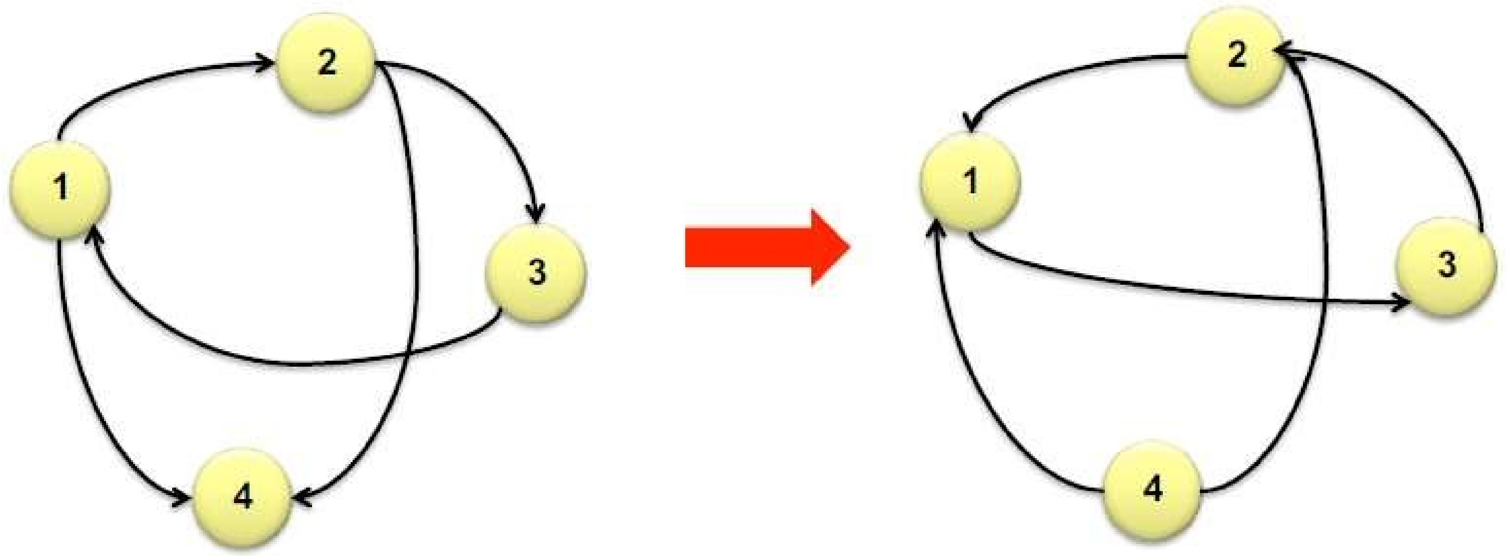
Adjacency Lists: Critique

- Advantages:
  - Much more compact representation
  - Easy to compute over outlinks

- Disadvantages:
  - Much more difficult to compute over inlinks

Challenge Question

- How would you invert a graph in MapReduce?



Reverse Web-Link Graph

- For each URL, find all pages (URLs) pointing to it (incoming links)
- **Problem:** Web page has only outgoing links
- Need all (anySource, P) links for each page P
  - Suggests Reduce with P as the key, source as value

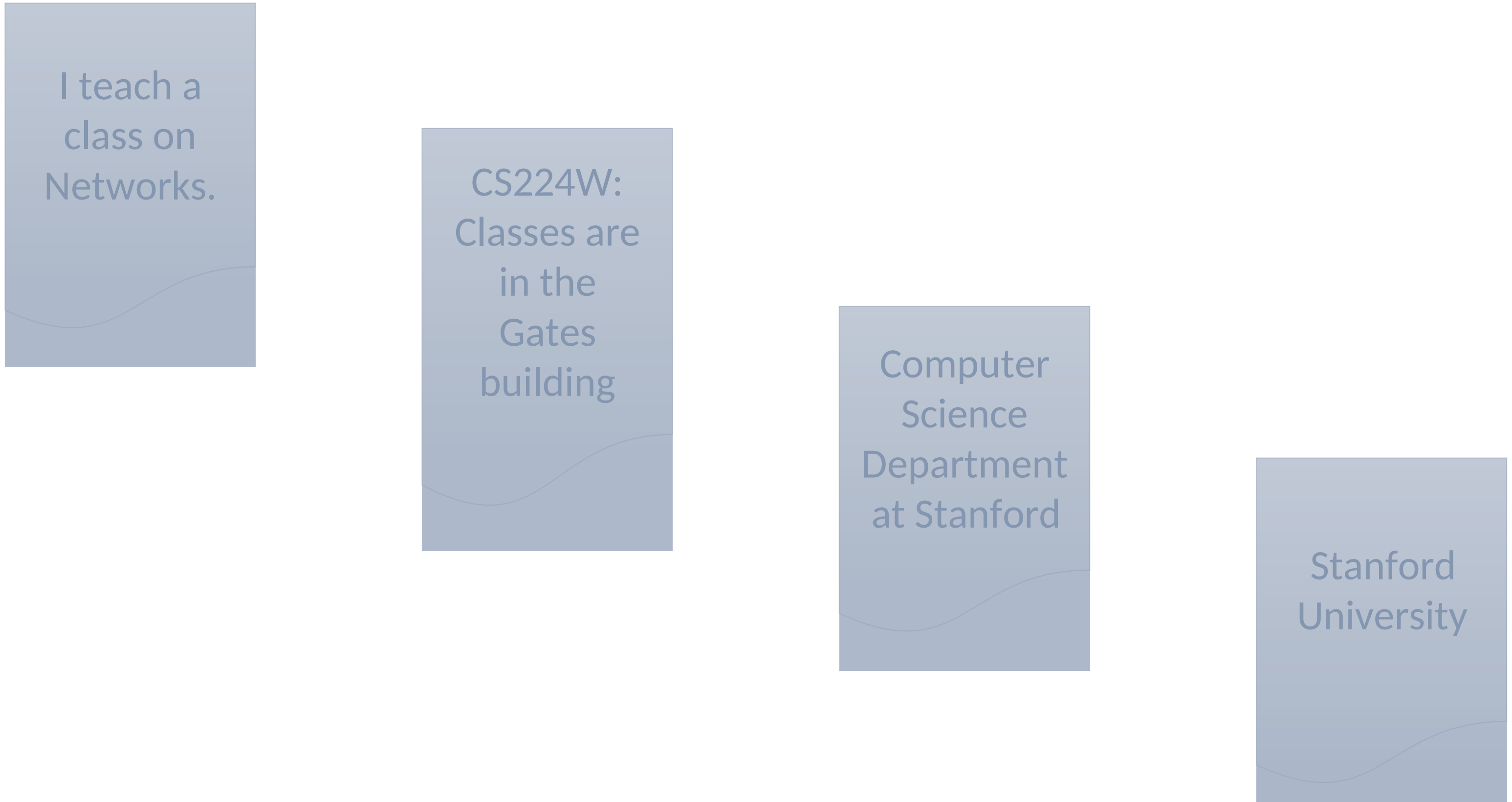
- **Map:**
  - for page source, create all (target, source) pairs for each link to a target found in page
- **Reduce:**
  - since target is key, will receive all sources pointing to that target

Graphs and MapReduce

- A large class of graph algorithms involve:
  - Performing computations at each node: based on node features, edge features, and local link structure
  - Propagating computations: “traversing” the graph
- Generic recipe:
  - Represent graphs as adjacency lists
  - Perform local computations in mapper
  - Pass along partial results via out-links, keyed by destination node
  - Perform aggregation in reducer on in-links to a node
  - Iterate until convergence: controlled by external “driver”
  - Don’t forget to pass the graph structure between iterations

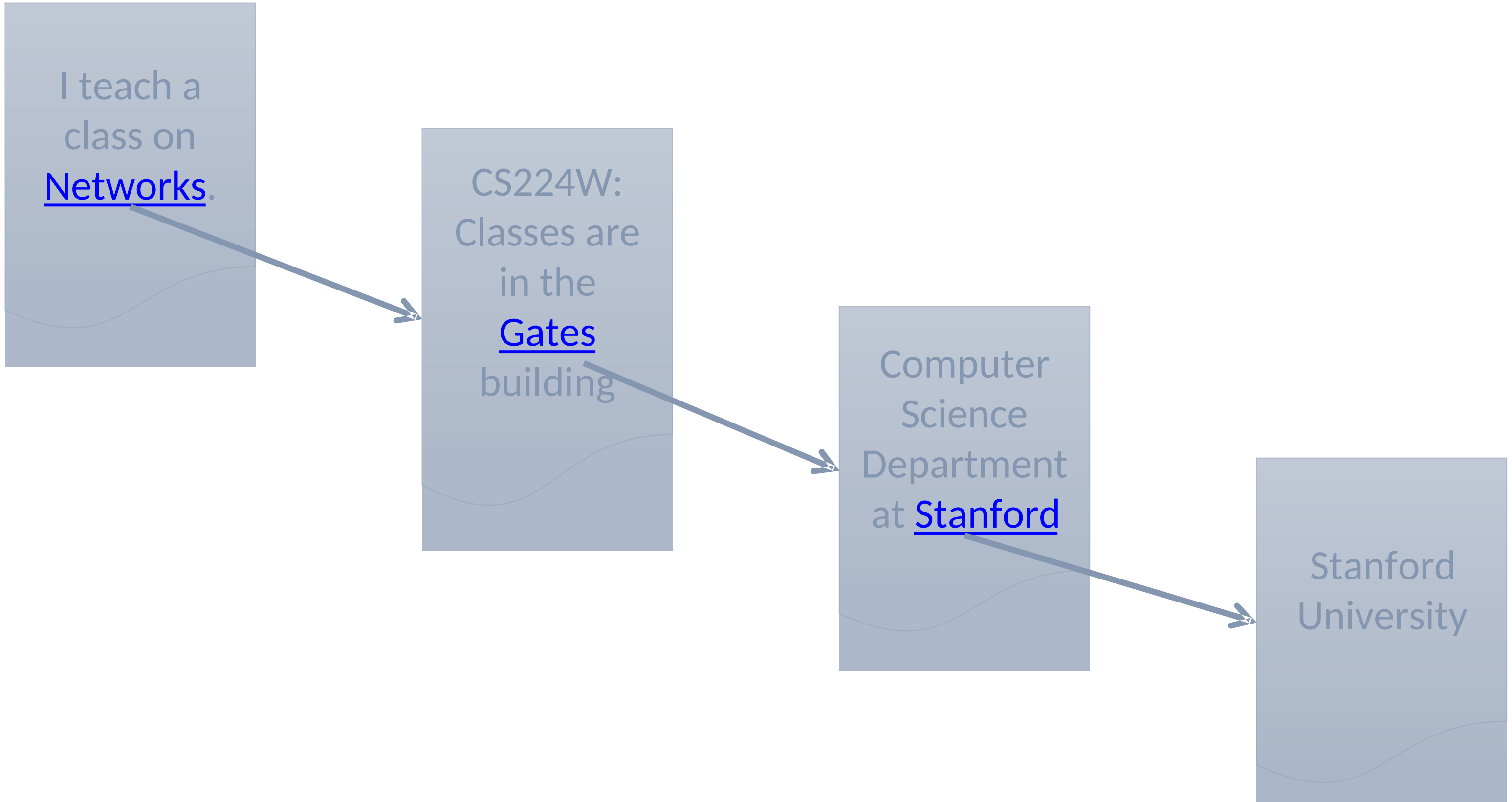
Web as a Graph

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



Web as a Graph

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

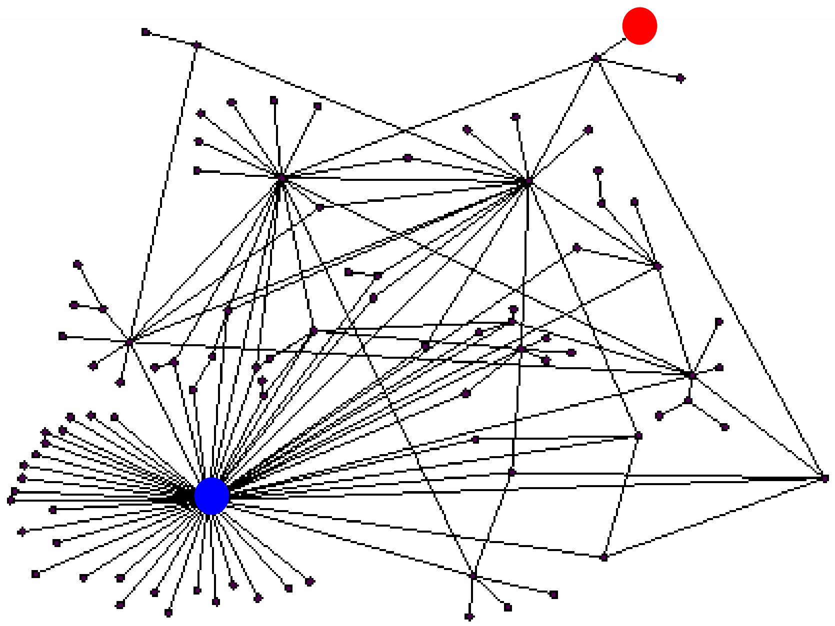


Ranking Nodes on the Graph

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

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

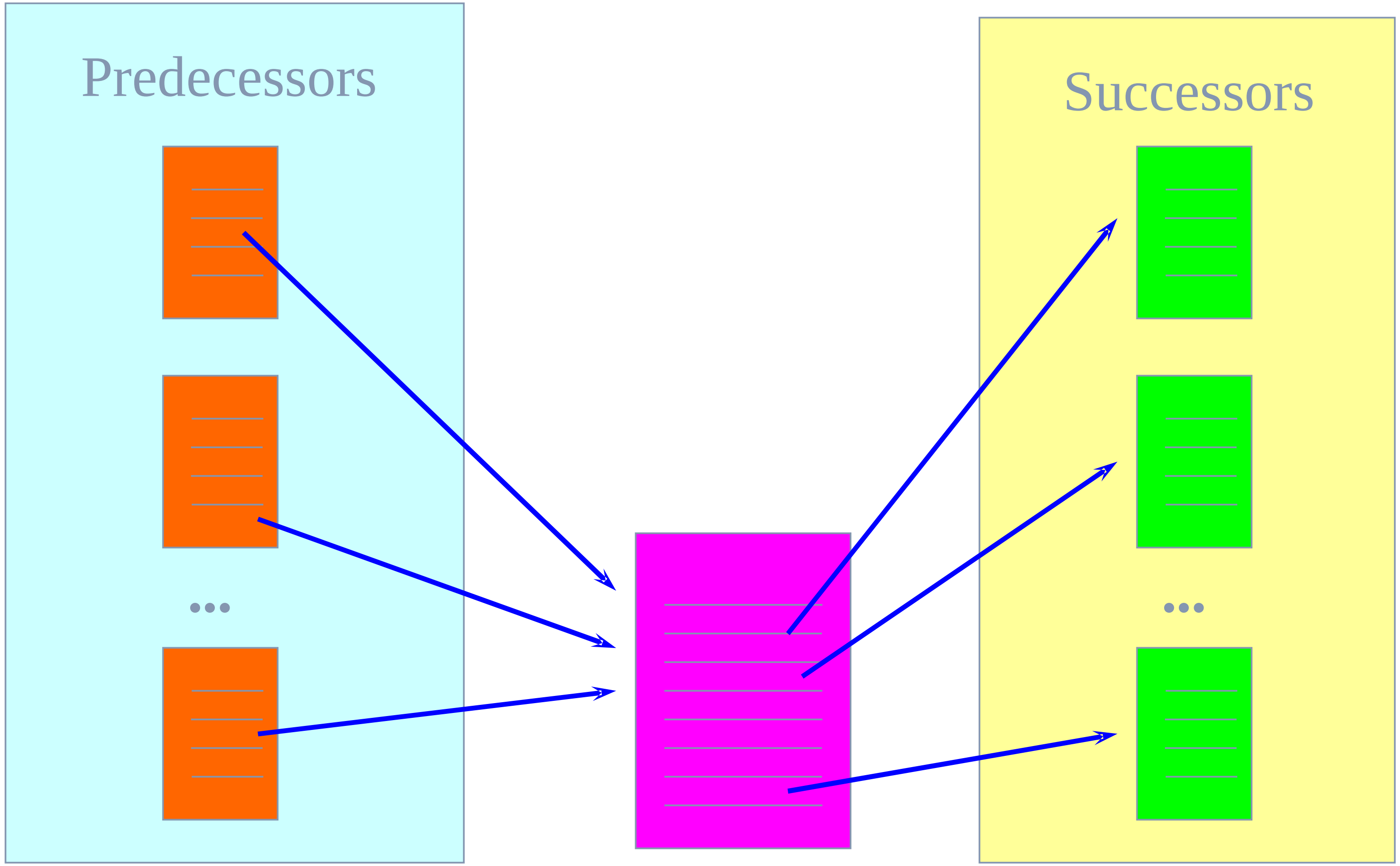
- There is large diversity in the web-graph node connectivity.
  Let's rank the pages by the link structure!



PageRank: Ranking web pages (Brin & Page'98)

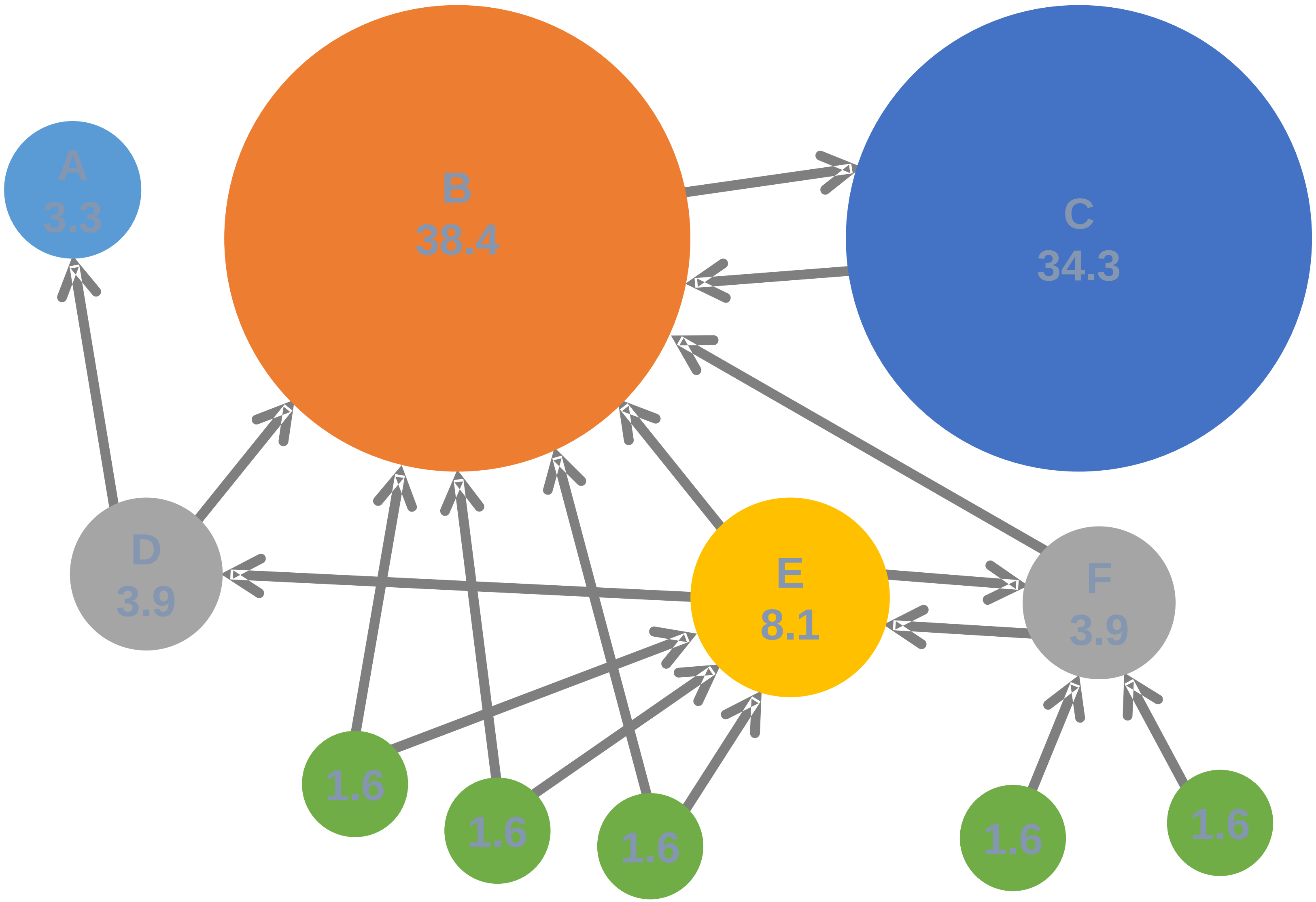
- Intuition
  - Web pages are not equally "important"
    - www.james.com v www.stanford.edu
  - Links as citations: a page cited often is more important
    - www.stanford.edu has 23,400 inlinks
    - www.james.com has 1 inlink
  - Are all links equal?
    - Links from important pages count more
  - Recursive model: being cited by a highly cited paper counts a lot…

Predecessors and Successors of a Web Page



PageRank:
The “Flow” Formulation

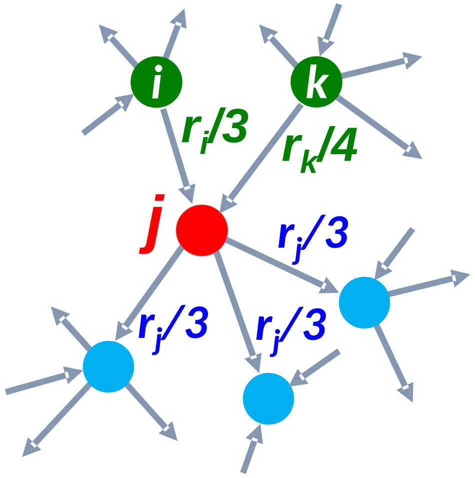
Example: PageRank Scores



Simple Recursive Formulation

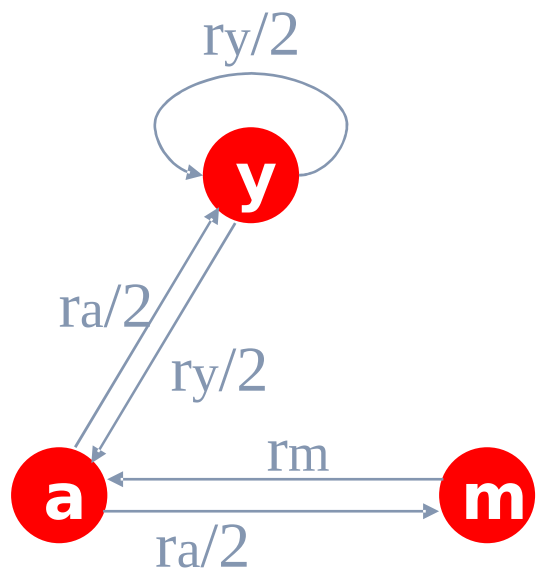
- Each link's vote is proportional to the **importance** of its source page
- If page ***j*** with importance r_j has ***n*** out-links, each link gets r_j / n votes
- Page ***j***'s own importance is the sum of the votes on its in-links

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



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



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

d_i … out-degree of node i

"Flow" equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: Matrix Formulation

- **Stochastic adjacency matrix**
  - Let page i has $_i$ out-links
  - If $i \to j$ then $_{ji} = \frac{1}{_i}$ else $_{ji} = 0$
    - ㅊ is a **column stochastic matrix**
      - ㅊColumns sum to 1
- **Rank vector** : vector with an entry per page
  - $_i$ is the importance score of page i
  - σ $_i$ $_i$ $= 1$
- **The flow equations can be written**

$$= \cdot$$

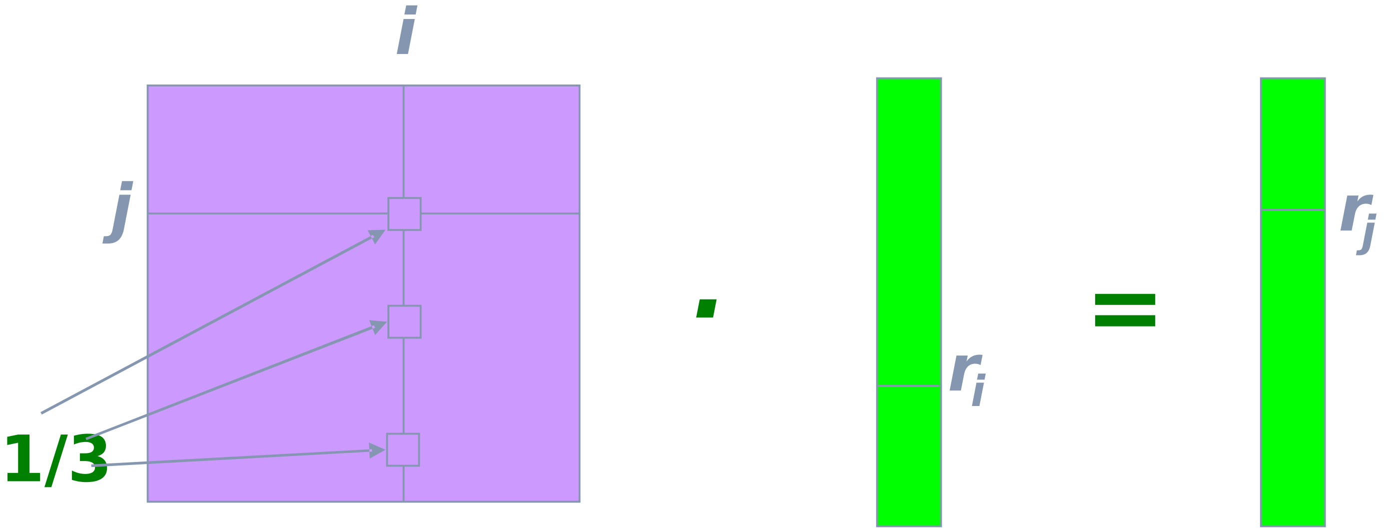
$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$$

Example

- Remember the flow equation: $$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$$
- Flow equation in the matrix form

$\cdot \quad =$

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

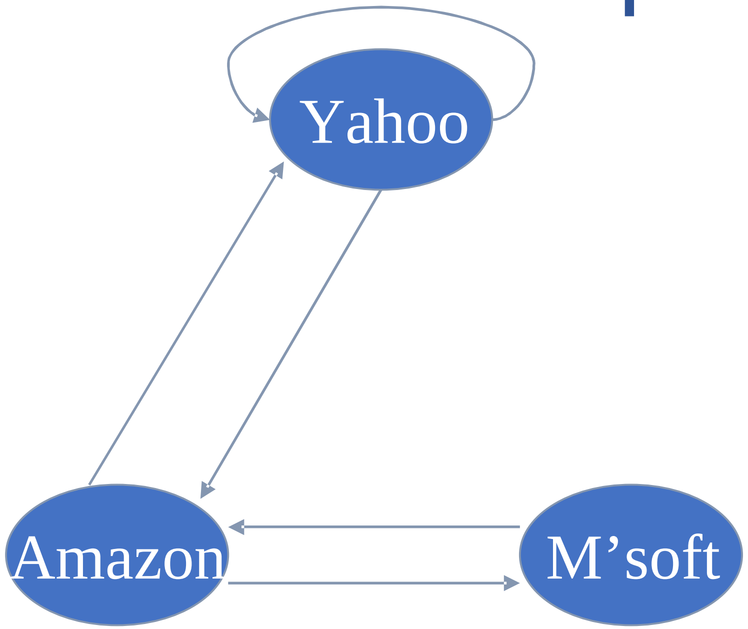


$$M \cdot r = r$$

Power Iteration method

- Simple iterative scheme (aka relaxation)
- Suppose there are N web pages
- Initialize: $\mathbf{r}^0 = [1/N,....,1/N]^T$
- Iterate: $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$
- Stop when $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \varepsilon$
  - $|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
  - Can use any other vector norm e.g., Euclidean

Power Iteration Example



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 1/3 \\ 1/2 \\ 1/6 \end{array} \quad \begin{array}{c} 5/12 \\ 1/3 \\ 1/4 \end{array} \quad \begin{array}{c} 3/8 \\ 11/24 \\ 1/6 \end{array} \quad \ldots \quad \begin{array}{c} 2/5 \\ 2/5 \\ 1/5 \end{array}$$

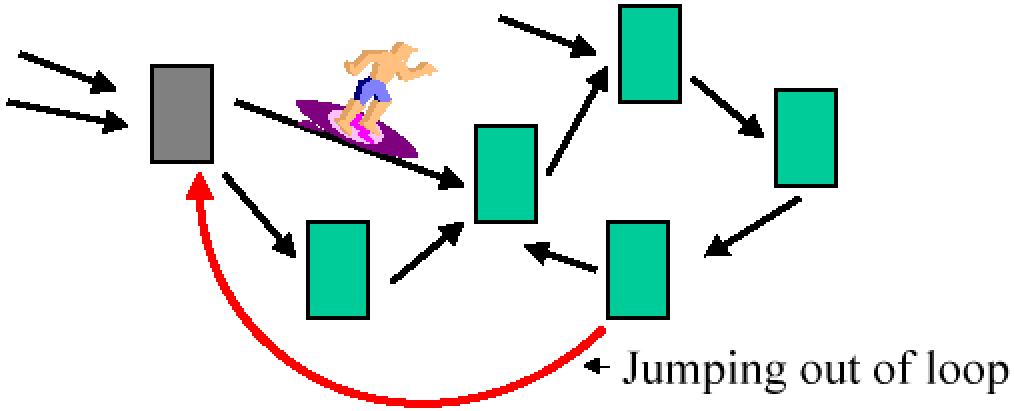
Page Rank

The intuition behind this matrix:

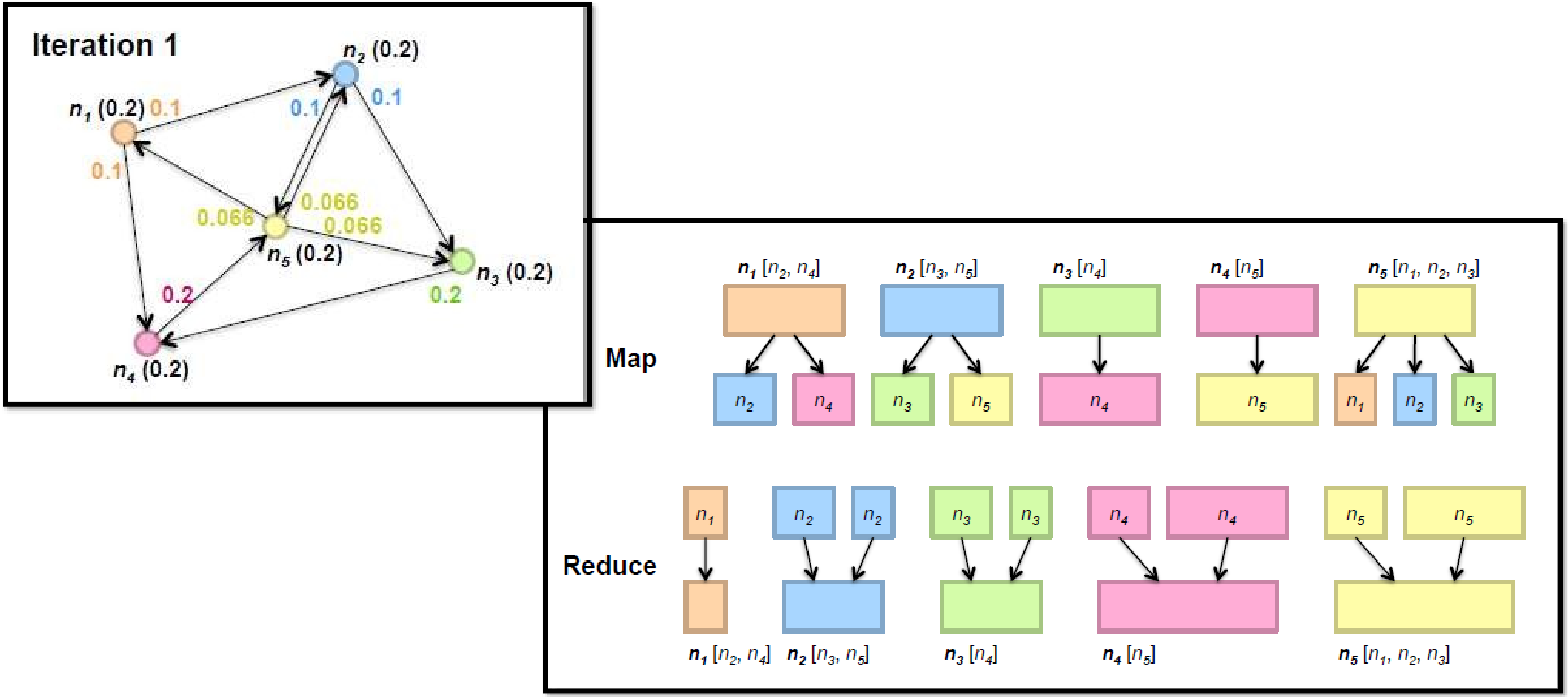
- Initially each page has 1 unit of importance.
- At each round, each page shares importance it has among its successors, and receives new importance from its predecessors.
- The importance of each page reaches a limit after some steps
- That importance is also the probability that a Web surfer, starting at a random page, and following random links from each page will be at the page in question after a long series of links.

Random Walk Interpretation

- Imagine a random web surfer
  - At any time t, surfer is on some page P
  - At time t+1, the surfer follows an outlink from P uniformly at random
  - Ends up on some page Q linked from P
  - Process repeats indefinitely
- **p**(t) is the probability distribution whose i[th] component is the probability that the surfer is at page i at time t



Page Rank in Map Reduce



PageRank Pseudo-code

```
class Mapper
    method Map(nid n, node N)
        p ← N.PageRank/|N.AdjacencyList|
        Emit(nid n, N)                                ▷ Pass along graph structure
        for all nodeid m ∈ N.AdjacencyList do
            Emit(nid m, p)                            ▷ Pass PageRank mass to neighbors
```

```
class Reducer
    method Reduce(nid m, [p1, p2, . . .])
        M ← ∅
        for all p ∈ counts [p1, p2, . . .] do
            if IsNode(p) then
                M ← p                                 ▷ Recover graph structure
            else
                s ← s + p                             ▷ Sums incoming PageRank contributions
        M.PageRank ← s
        Emit(nid m, node M)
```

PageRank:
The Google Formulation

PageRank: Three Questions

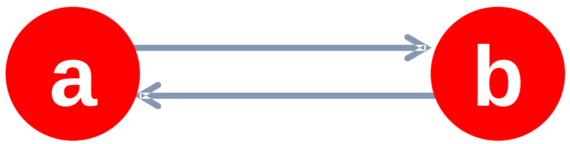
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

or equivalently

$$r = Mr$$

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

Does this converge?



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

- **Example:**

$$\begin{array}{lcccc} \mathrm{r_a} & 1 & 0 & 1 & 0 \\ \mathrm{r_b} & 0 & 1 & 0 & 1 \end{array}$$

where "=" stands between the first and second columns of values.

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:**

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

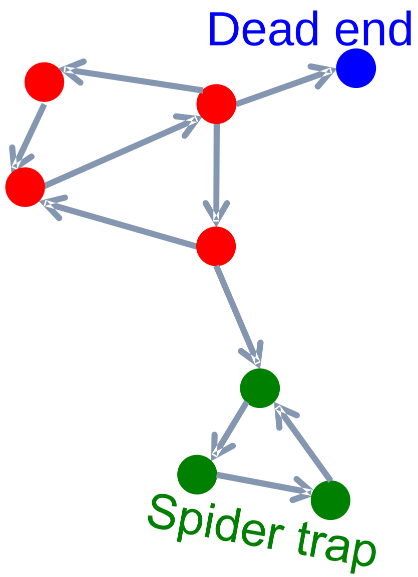
Iteration 0, 1, 2, …

PageRank: Problems

2 problems:

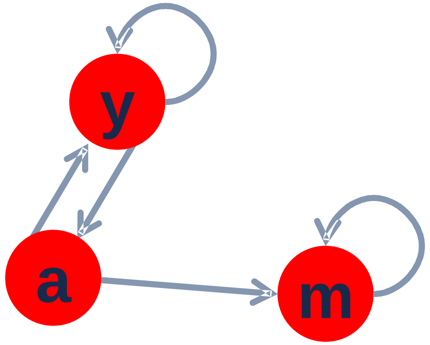
- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"

- **(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 $_j = 1$
  - $_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

m is a spider trap

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

- **Example:**

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

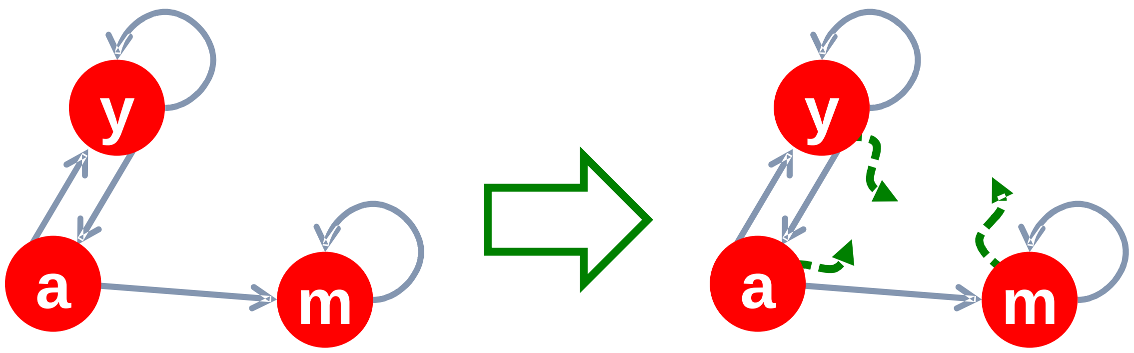
Iteration 0, 1, 2, …

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

Solution: Teleports!

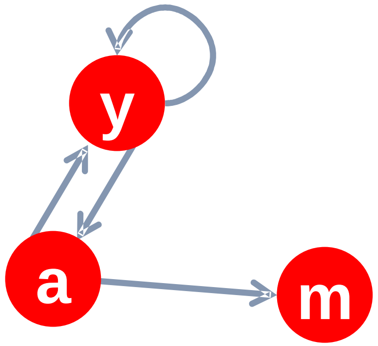
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-** β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9

- **Surfer will teleport out of spider trap within a few time steps**



Problem: Dead Ends

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

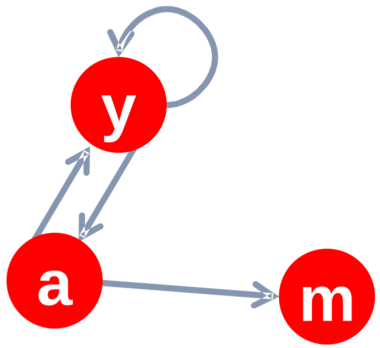
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

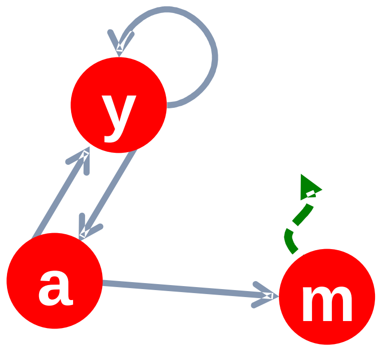
Here the PageRank “leaks” out since the matrix is not stochastic.

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

Solution: Random Teleports

- **<u>Google's solution that does it all:</u>**
  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]

$$= \sum_{i \rightarrow} \frac{i}{i} + (1 - \ ) \frac{1}{}$$

d_i … out-degree of node i

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

The Google Matrix

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

$$_j = \sum_{i \to j} \frac{_i}{_i} + (1 - \ ) \frac{1}{\ }$$

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

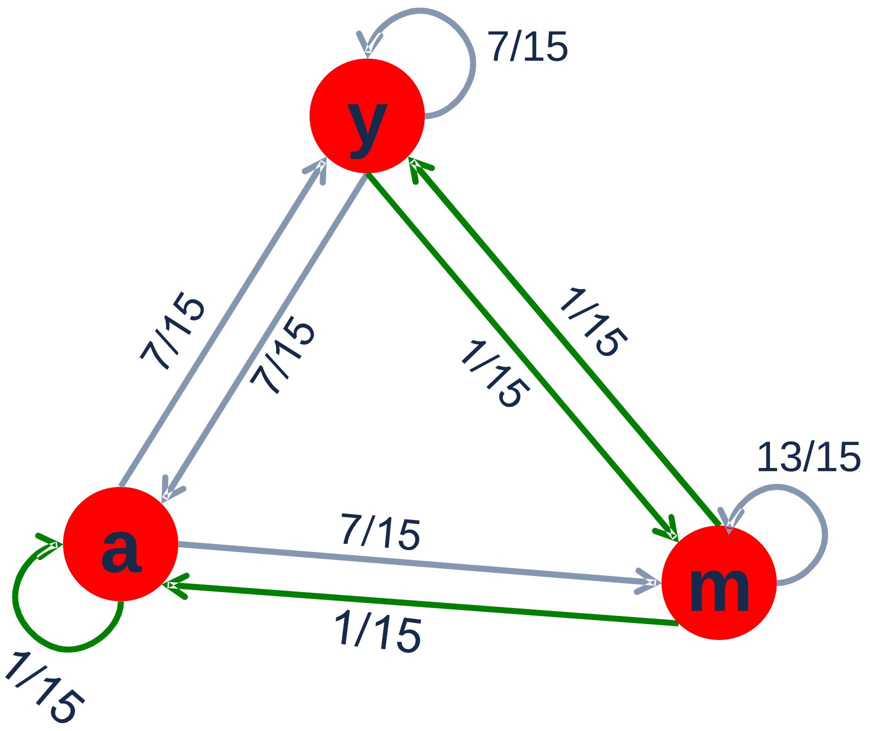
$$= + 1 - \quad (\qquad) \left[\frac{1}{\ }\right]_{\times}$$

[1/N]$_{NxN}$…N by N matrix where all entries are 1/N

- **We have a recursive problem:** $= \ \cdot$

  And the Power method still works!
- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleports ($\beta = 0.8$)



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

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

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