大数据计算及应用(六)

Link Analysis-1

Agenda

| High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	**PageRank,** SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Graph Data: Social Networks
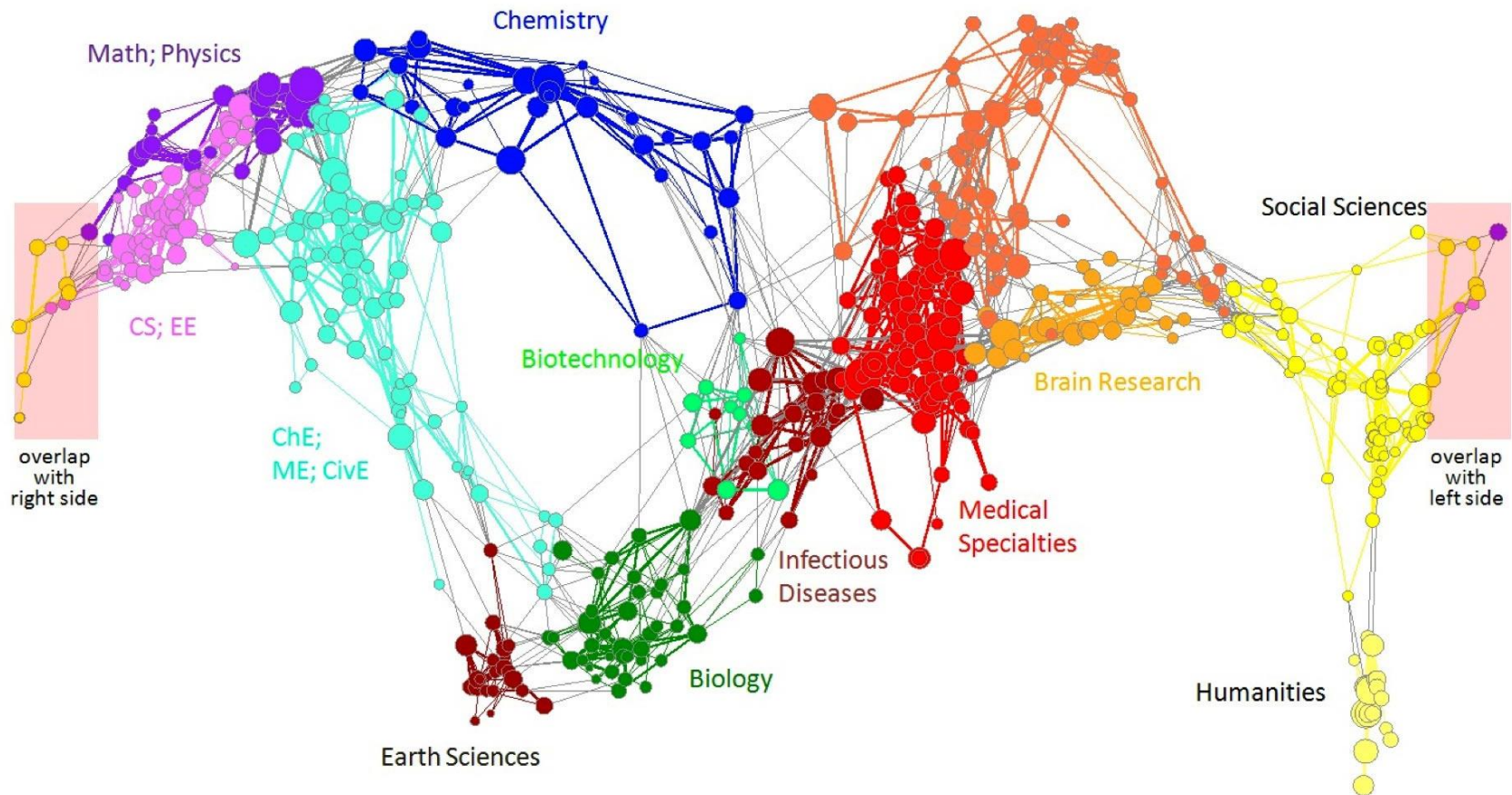

Facebook social graph
4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]
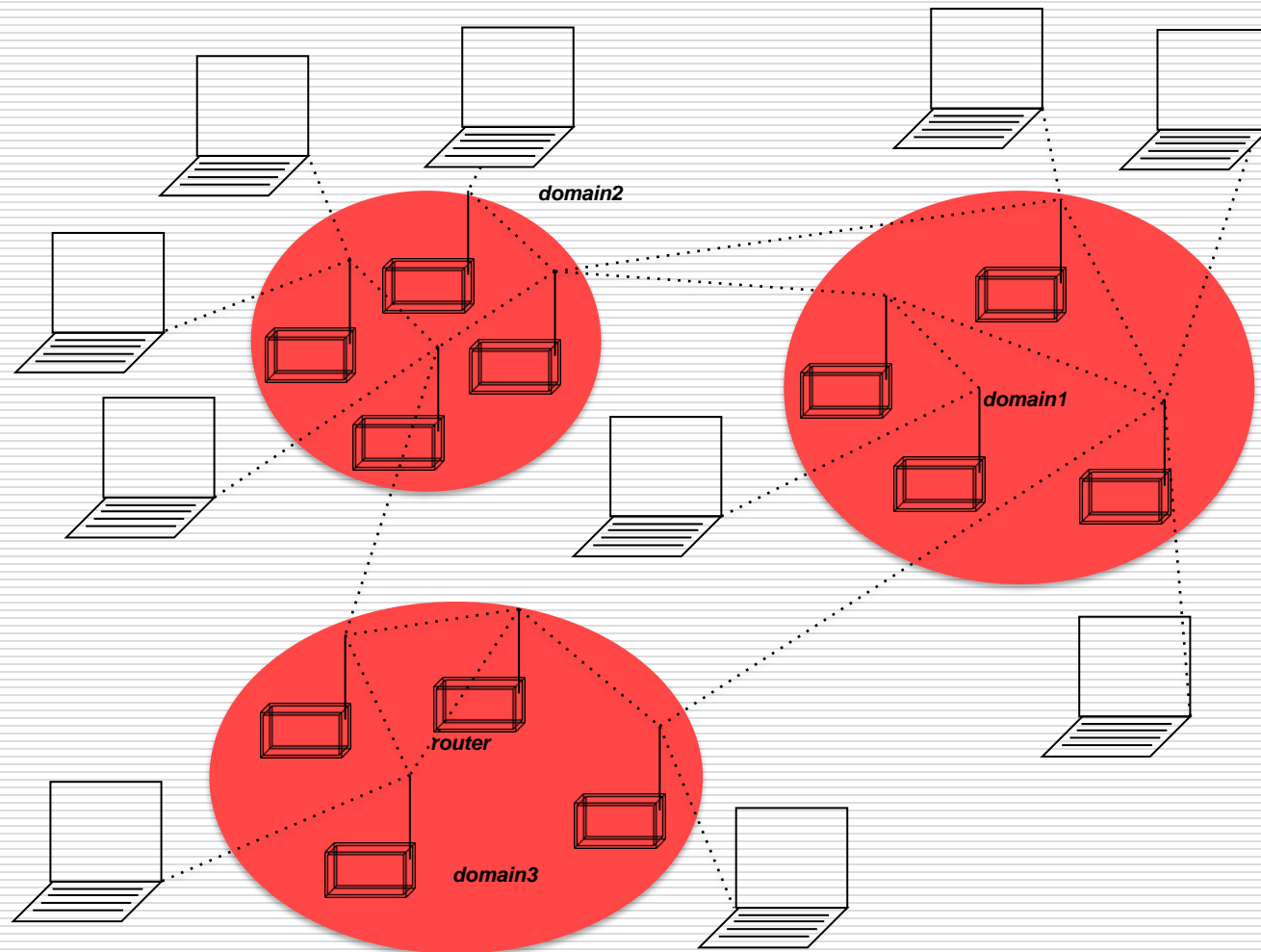
Graph Data: Media Networks
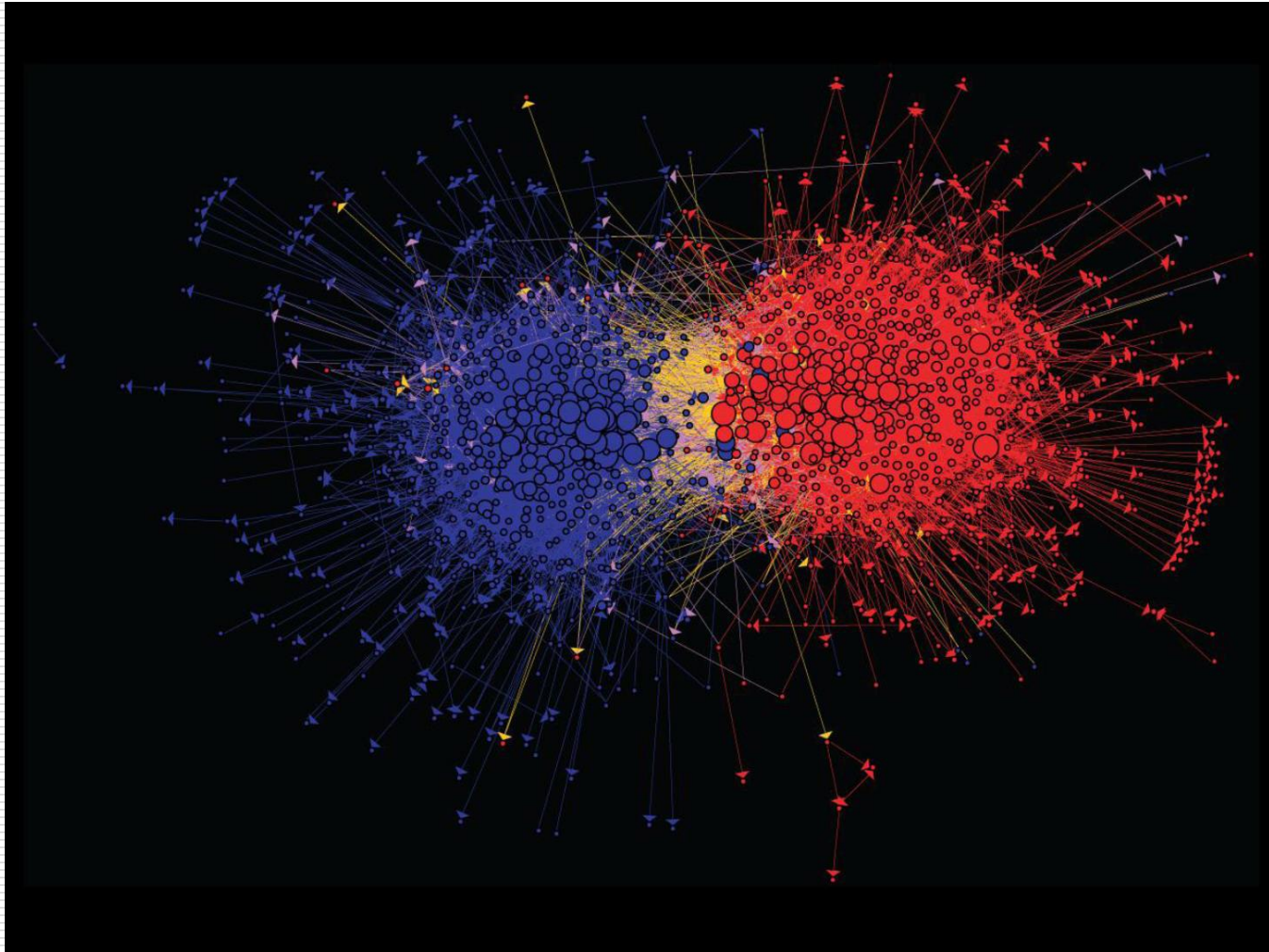
Connections between political blogs

Polarization of the network [Adamic-Glance, 2005]
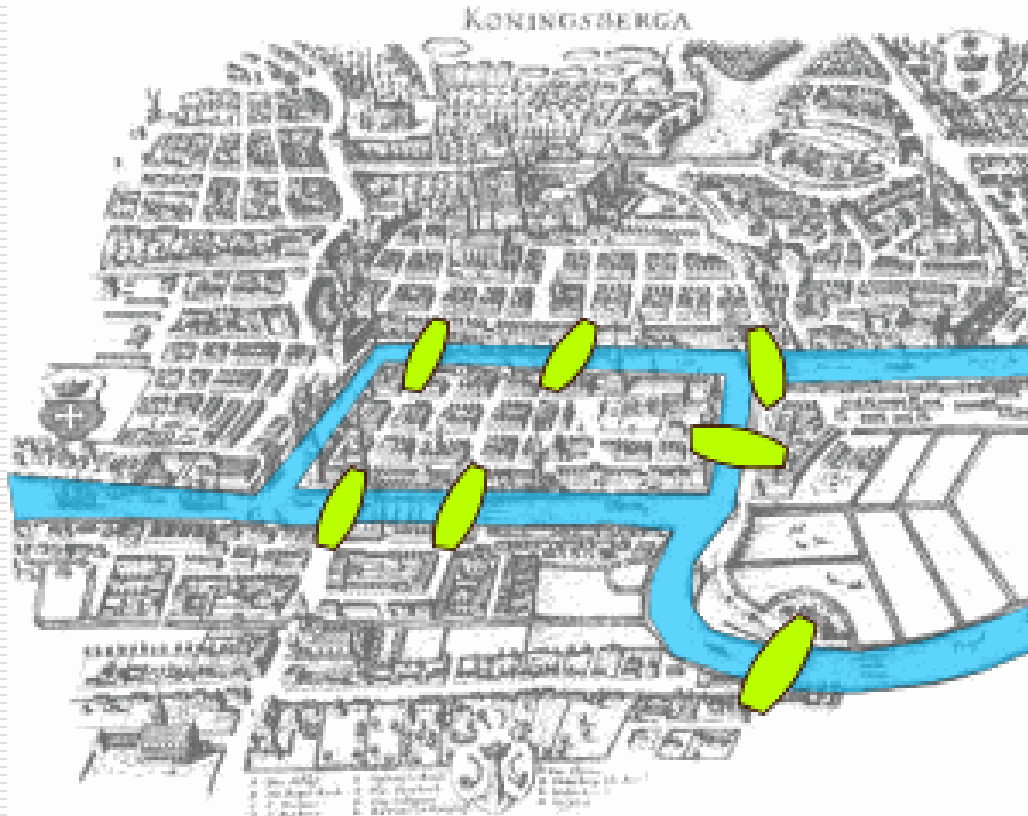
Graph Data: Information Nets

Citation networks and Maps of science

[Börner et al., 2012]

Graph Data: Communication Nets

domain2
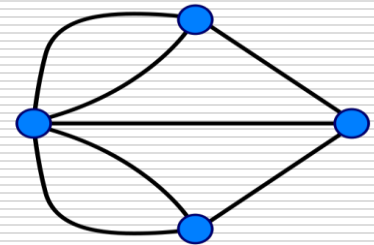
domain1

router

domain3

Internet

Graph Data: Technological Networks

Seven Bridges of Königsberg

[Euler, 1735]

Return to the starting point by traveling each link of the graph once and only once.

Web as a Graph

Web as a directed graph:

- **Nodes: Webpages**
- **Edges: Hyperlinks**

I teach a class on Networks.

CS224W: Classes are in the Gates building

Computer Science Department at Stanford

Stanford University

Web as a Graph

☐ **Web as a directed graph:**

■ **Nodes: Webpages**

■ **Edges: Hyperlinks**

I teach a class on **Networks**.

CS224W: Classes are in the **Gates** building
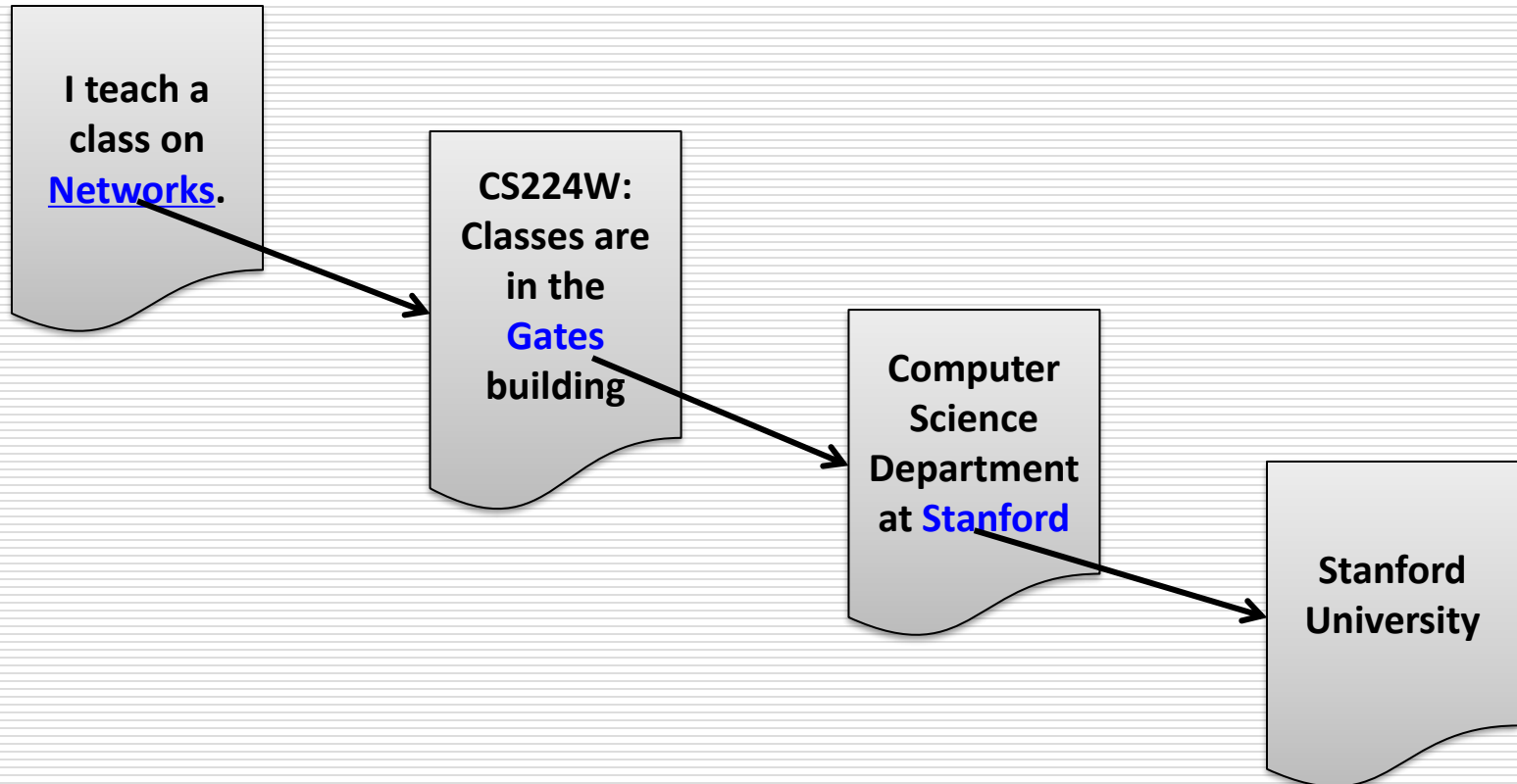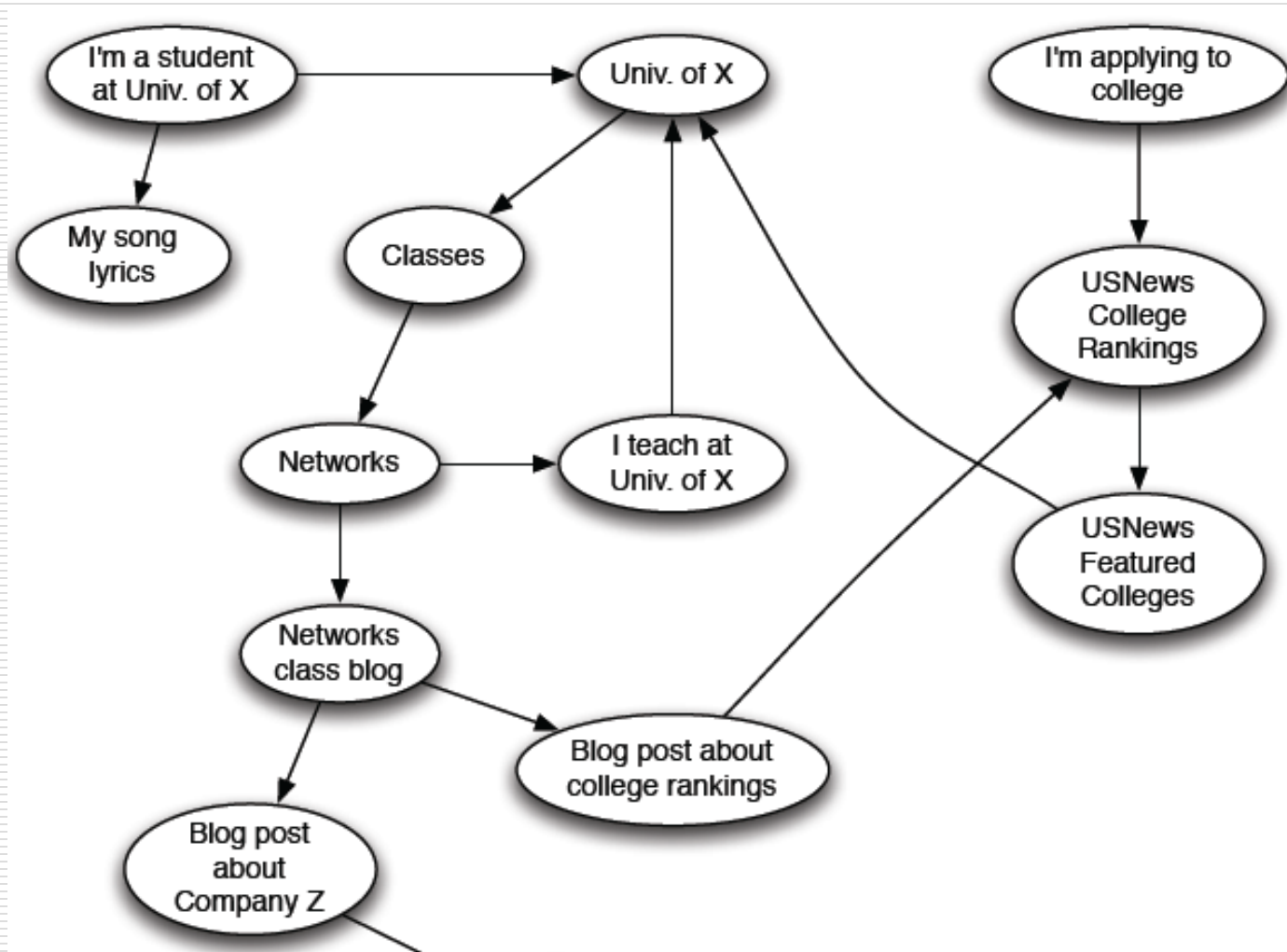
Computer Science Department at **Stanford**

Stanford University

Web as a Directed Graph

Broad Question

☐ **How to organize the Web?**

☐ **First try:** Human curated
Web directories

■ Yahoo, DMOZ, LookSmart

☐ **Second try: Web Search**

■ **Information Retrieval** investigates
Find relevant docs in a small
and trusted set

☐ Newspaper articles, Patents, etc.

■ **But:** Web is **huge**, full of untrusted documents,
random things, web spam, etc.

Web Search: 2 Challenges

**2 challenges of web search:**

- **(1) Web contains many sources of information Who to "trust"?**
    - **Trick:** Trustworthy pages may point to each other!

- **(2) What is the "best" answer to query "newspaper"?**
    - No single right answer
    - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

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

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

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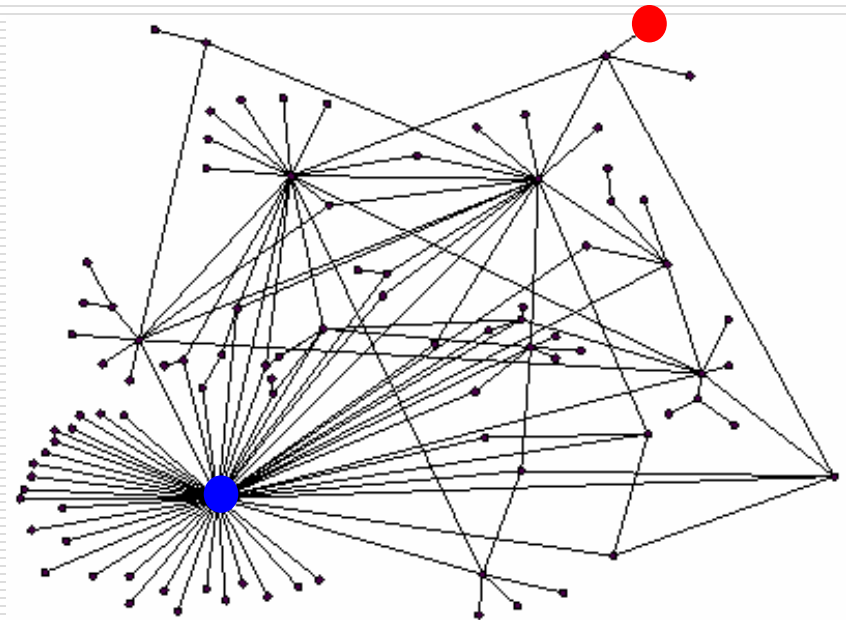
Link Analysis Algorithms

☐ We will cover the following **Link Analysis approaches** for computing **importances** of nodes in a graph:

- ■ Page Rank
- ■ Topic-Specific (Personalized) Page Rank
- ■ Web Spam Detection Algorithms

PageRank:
The "Flow" Formulation

Links as Votes

☐ **Idea: Links as votes**

  ■ **Page is more important if it has more links**

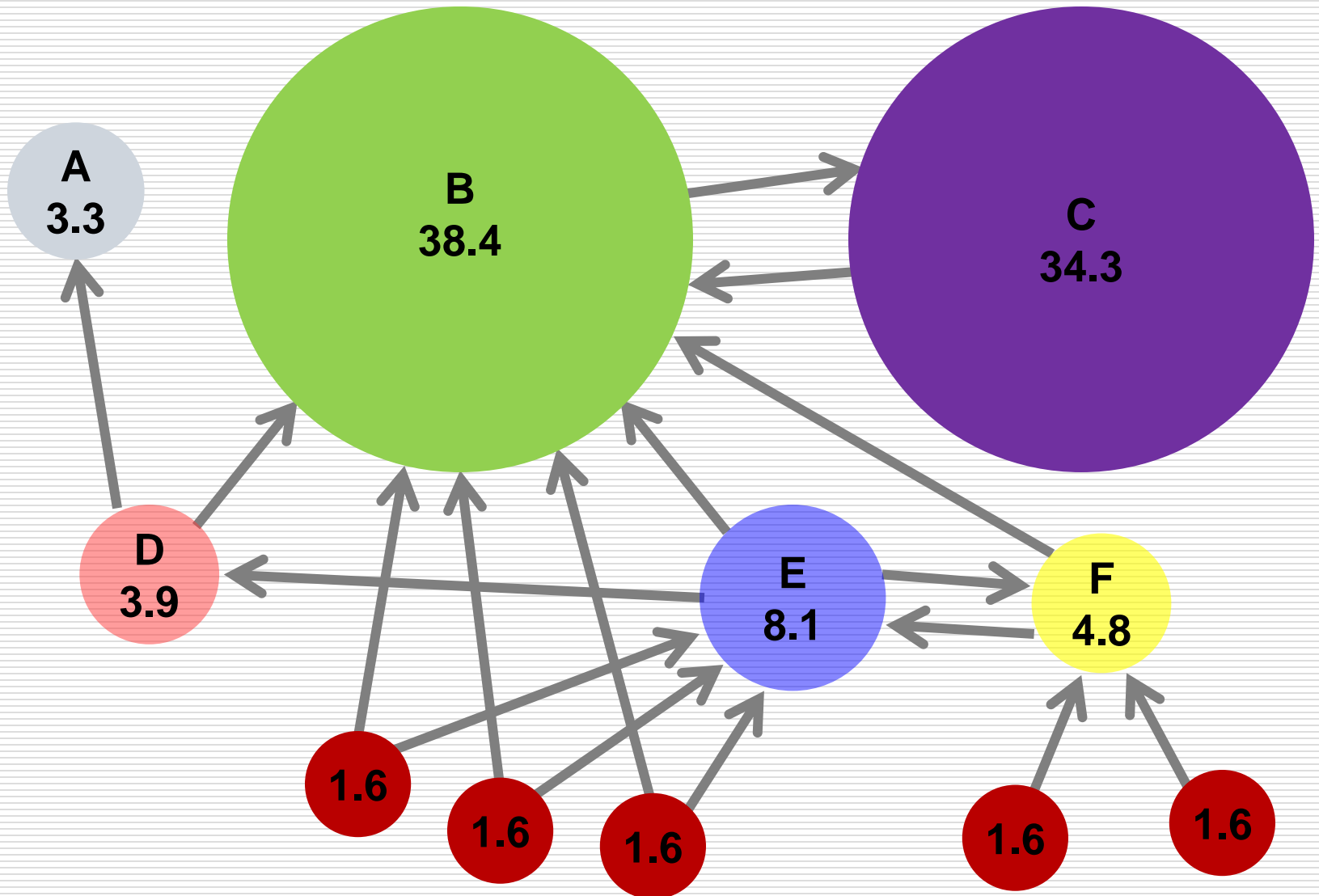   ☐ In-coming links? Out-going links?

☐ **Think of in-links as votes:**

  ■ www.stanford.edu has 23,400 in-links

  ■ www.joe-schmoe.com has 1 in-link

☐ **Are all in-links are equal?**

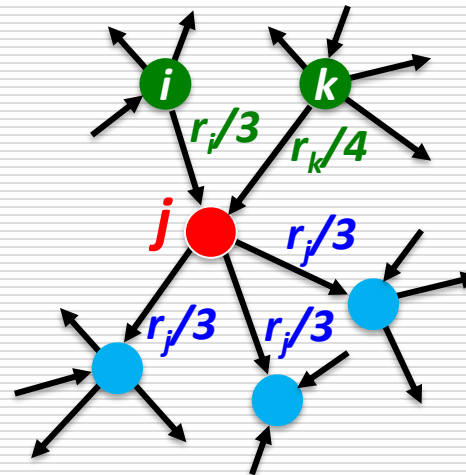 ■ **Links from important pages count more**

  ■ Recursive question!

Example: PageRank Scores

Simple Recursive Formulation

☐ Each link's vote is proportional to the **importance** of its source page

☐ If page *j* with importance r_j has *n* out-links, each link gets r_j / n votes

☐ Page *j*'s own importance is the sum of the votes on its in-links

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

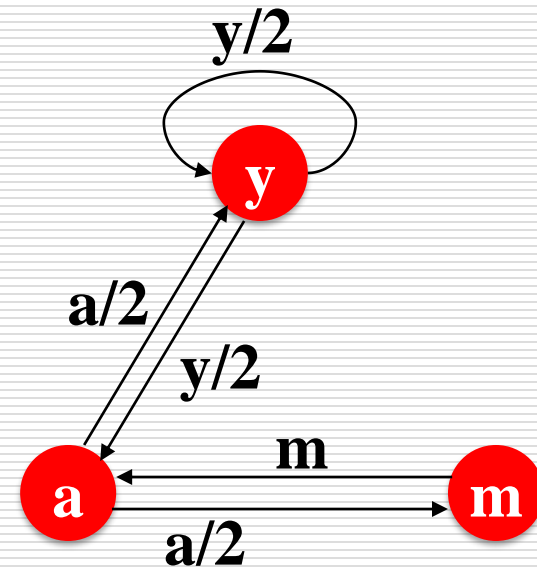# PageRank: The "Flow" Model

- **A "vote" from an important page is worth more**

- **A page is important if it is pointed to by other important pages**

- **Define a "rank" r_j for page j**

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

d_i ... **out-degree of node i**

The web in 1839

"Flow" equations:

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

Solving the Flow Equations

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

PageRank: Matrix Formulation

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

$$r = M \cdot r$$

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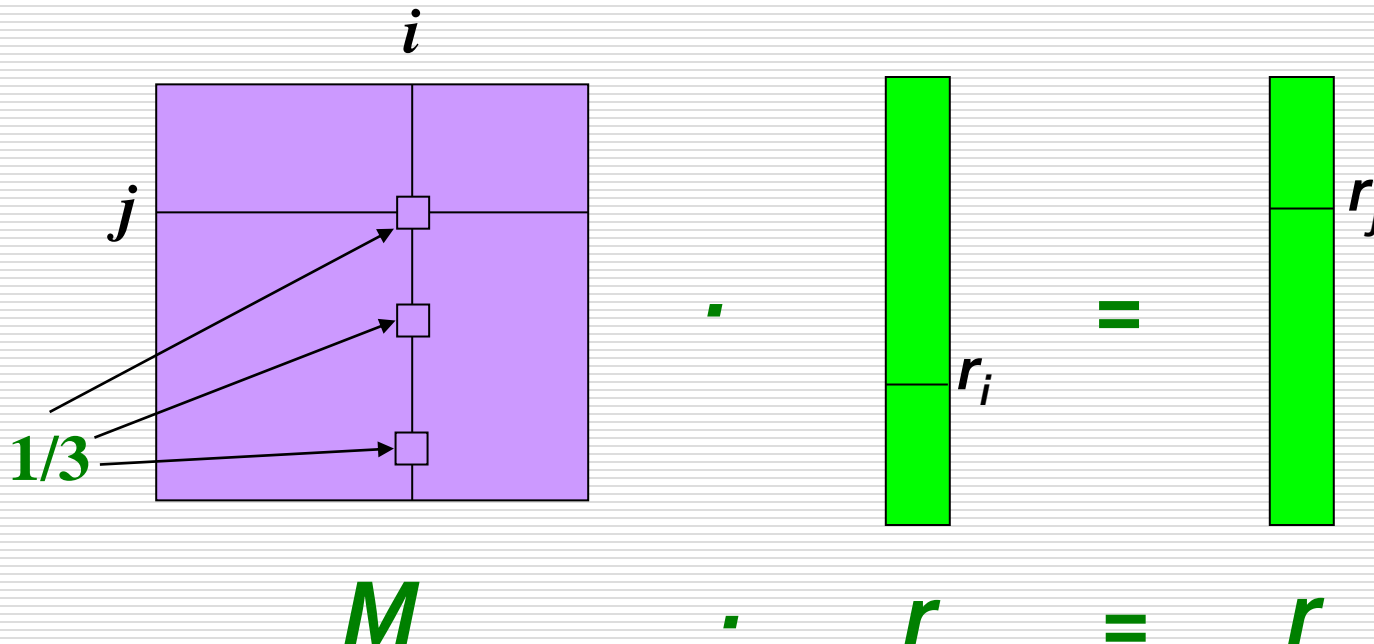
Example

☐ **Remember the flow equation:**

☐ **Flow equation in the matrix form**
$$M \cdot r = r$$

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

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

$$M \qquad \cdot \quad r \quad = \quad r$$

Eigenvector Formulation

☐ **The flow equations can be written**
$$r \;=\; M \cdot r$$

☐ So the **rank vector r** is an **eigenvector** of the stochastic web matrix **M**

   ■ In fact, its first or principal eigenvector, with corresponding eigenvalue **1**

      ☐ Largest eigenvalue of **M** is **1** since **M** is column stochastic (with non-negative entries)

         ■ *We know r is unit length and each column of M sums to one, so $Mr \leq 1$*

NOTE: x is an eigenvector with the corresponding eigenvalue λ if:
$$Ax = \lambda x$$

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

Example: Flow Equations & M
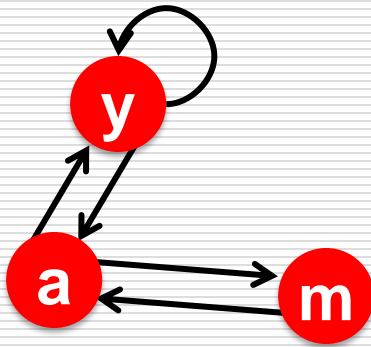
	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$r = M \cdot r$$

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

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

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: $\mathbf{r}^{(0)} = [1/N,\ldots,1/N]^{\mathsf{T}}$

■ Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$

■ Stop when $\left| \mathbf{r}^{(t+1)} - \mathbf{r}^{(t)} \right|_1 < \varepsilon$

$|x|_1 = \sum_{1 \leq i \leq N} |x_i|$ is the L₁ norm
Can use any other vector norm, e.g., Euclidean

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

d_i …. out-degree of node

PageRank: How to solve?

☐ **Power Iteration:**

  ■ Set $r_j = 1/N$

  ■ **1:** $r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$

  ■ **2:** $r = r'$

  ■ Goto **1**

☐ **Example:**

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

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

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

Iteration 0

26

PageRank: How to solve?

☐ **Power Iteration:**

▪ Set $r_j = 1/N$

▪ **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$

▪ **2:** $r = r'$

▪ Goto **1**

☐ **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{array}{cc} 1/3 & 1/3 \\ 1/3 & 3/6 \\ 1/3 & 1/6 \end{array}$$

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Iteration 0, 1

PageRank: How to solve?
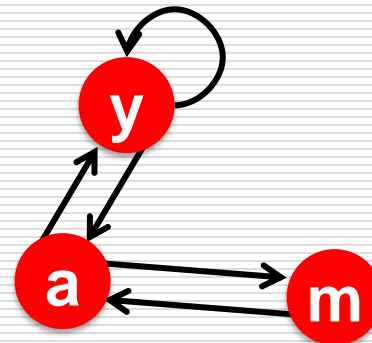
☐ **Power Iteration:**

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

☐ **Example:**

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

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

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

Iteration 0, 1, 2

PageRank: How to solve?
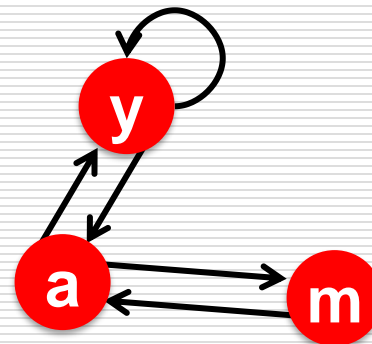
☐ **Power Iteration:**

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

☐ **Example:**

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

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

Iteration 0, 1, 2, 3 …

Why Power Iteration works? (1)

☐ **Power iteration:**

A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)

- $r^{(1)} = M \cdot r^{(0)}$

- $r^{(2)} = M \cdot r^{(1)} = M(Mr^{(0)}) = M^2 \cdot r^{(0)}$

- $r^{(3)} = M \cdot r^{(2)} = M(M^2 r^{(0)}) = M^3 \cdot r^{(0)}$

☐ **Claim:**

Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \dots M^k \cdot r^{(0)}, \dots$ approaches the dominant eigenvector of M

Why Power Iteration works? (2)

☐ **Claim:** Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \dots M^k \cdot r^{(0)}, \dots$ approaches the dominant eigenvector of M

☐ **Proof:**

■ Assume M has n linearly independent eigenvectors, $x_1, x_2, \dots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$

■ Vectors $x_1, x_2, \dots, x_n$ form a basis and thus we can write:
$r^{(0)} = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$

■ $M r^{(0)} = M(c_1 x_1 + c_2 x_2 + \cdots + c_n x_n)$
$= c_1(M x_1) + c_2(M x_2) + \cdots + c_n(M x_n)$
$= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \cdots + c_n(\lambda_n x_n)$

■ **Repeated multiplication on both sides produces**
$M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$

Why Power Iteration works? (3)

Details!

- ☐ **Claim:** Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \dots M^k \cdot r^{(0)}, \dots$ approaches the dominant eigenvector of M
- ☐ **Proof (continued):**
  - ■ Repeated multiplication on both sides produces

    $$M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$$

  - ■ $M^k r^{(0)} = \lambda_1^k \left[c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right]$

  - ■ Since $\lambda_1 > \lambda_2$ then fractions $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \dots < 1$

    and so $\left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$ as $k \to \infty$ (for all $i = 2 \dots n$).

  - ■ **Thus:** $M^k r^{(0)} \approx c_1\left(\lambda_1^k x_1\right)$
    - ☐ Note if $c_1 = 0$ then the method won't converge

Random Walk Interpretation

- **Imagine a random web surfer:**
  - 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
  - Ends up on some page j linked from i
  - Process repeats indefinitely
- **Let:**
  - $p(t)$ … vector whose i^{th} coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages

$$r_j = \sum_{i \to j} \frac{r_i}{d_{\text{out}}(i)}$$
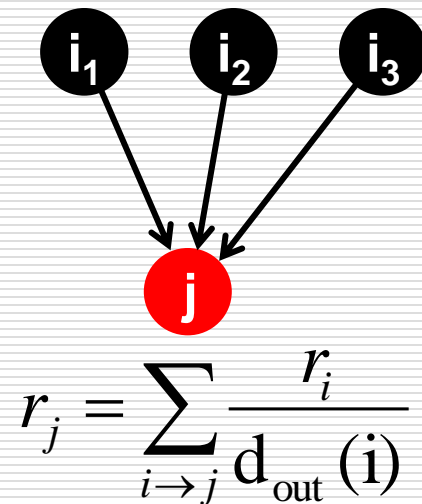
The Stationary Distribution

☐ **Where is the surfer at time *t+1*?**

■ Follows a link uniformly at random

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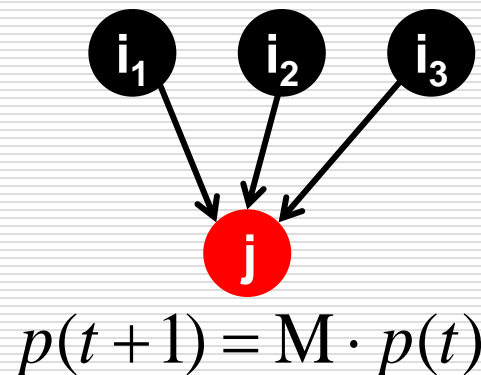
$$p(t+1) = \mathrm{M} \cdot p(t)$$

■ Suppose the random walk reaches a state

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

then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk

■ **Our original rank vector $\boldsymbol{r}$** satisfies $\boldsymbol{r = M \cdot r}$

■ **So, $\boldsymbol{r}$ is a stationary distribution for the random walk**

Existence and Uniqueness

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

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

PageRank:
The Google Formulation

PageRank: Three Questions

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

or equivalently

$$r = Mr$$

□ **Does this converge?**

□ **Does it converge to what we want?**

□ **Are results reasonable?**

Does this converge?

a ⇄ b

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

☐ **Example:**

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

Iteration 0, 1, 2, …

Does it converge to what we want?

$$a \longrightarrow b \qquad r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

☐ **Example:**

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

Iteration 0, 1, 2, ...

PageRank: Problems

2 problems:

Dead end

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

Spider trap

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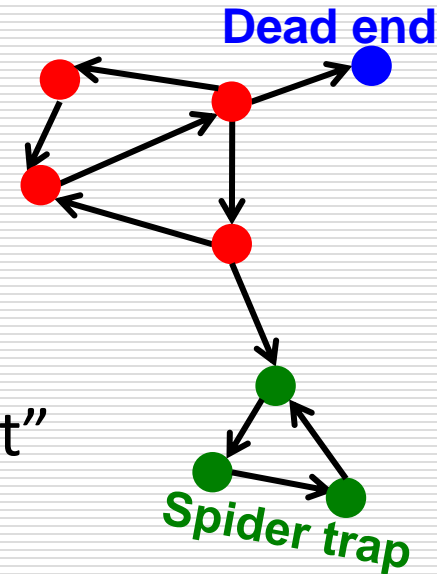
Problem: Spider Traps

☐ **Power Iteration:**

■ Set $r_j = 1$

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

☐ And iterate

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1
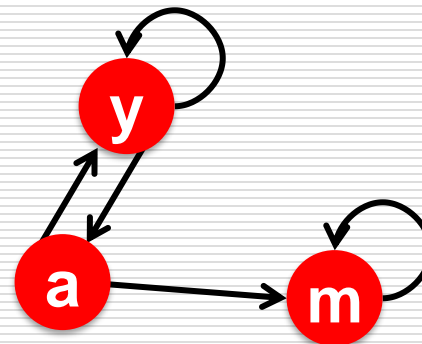
m is a spider trap

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2$

$r_m = r_a/2 + r_m$

☐ **Example:**

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

Iteration 0, 1, 2, …

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

41

Solution: Teleports!

☐ **The Google solution for spider traps:** At each time step, the random surfer has two options

- ■ With prob. β, follow a link at random
- ■ With prob. **1-β**, jump to some random page
- ■ Common values for β are in the range 0.8 to 0.9

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

Problem: Dead Ends

☐ **Power Iteration:**

   ■ Set $r_j = 1$

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

      ☐ And iterate

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0
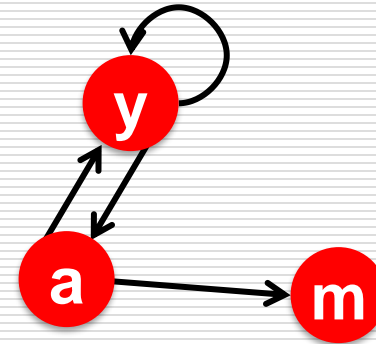
$r_y = r_y/2 + r_a/2$
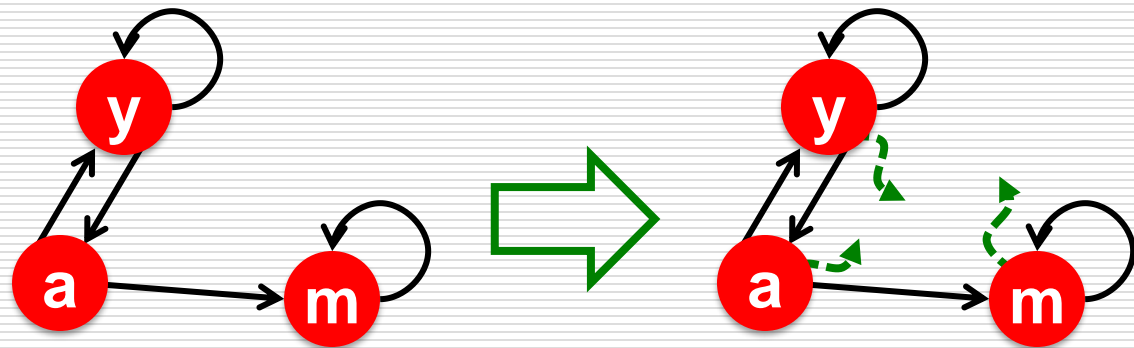
$r_a = r_y/2$

$r_m = r_a/2$

☐ **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: Always Teleport!

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

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓
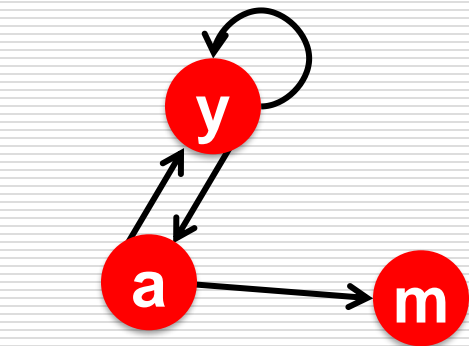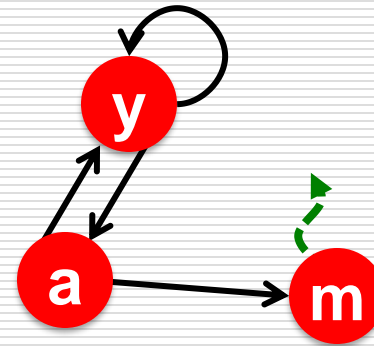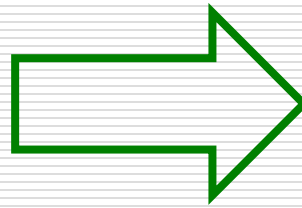
Why Teleports Solve the Problem?

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

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

Solution: Random Teleports

☐ **Google's solution that does it all:**

   At each step, random surfer has two options:

   ◼ With probability β, follow a link at random

   ◼ With probability $1-\beta$, jump to some random page

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

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

d_i ... out-degree of node i

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

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:**

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

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

☐ **We have a recursive problem: $r = A \cdot r$**
   And the Power method still works!

☐ **What is β?**

   ■ In practice β =0.8,0.9 (make *5* steps on avg., jump)

Random Teleports (β = 0.8)
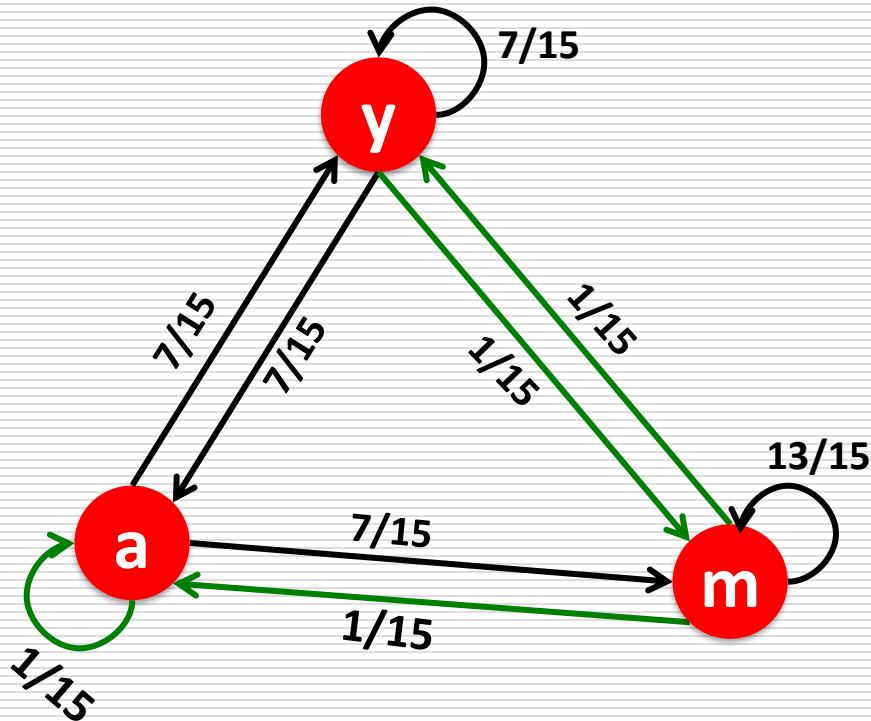
M

$$0.8 \begin{array}{|ccc|} \hline 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \\ \hline \end{array}$$

$\mathbf{[1/N]_{NxN}}$

$$+ 0.2 \begin{array}{|ccc|} \hline 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

y	7/15	7/15	1/15
a	7/15	1/15	1/15
m	1/15	7/15	13/15

A

y		1/3	0.33	0.24	0.26		7/33
a	=	1/3	0.20	0.20	0.18	. . .	5/33
m		1/3	0.46	0.52	0.56		21/33

How do we actually compute the PageRank?

Computing Page Rank

☐ **Key step is matrix-vector multiplication**

- $r^{new} = A \cdot r^{old}$

☐ Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$

☐ **Say N = 1 billion pages**

$A = \beta \cdot M + (1-\beta) [1/N]_{NxN}$

- We need 4 bytes for each entry (say)

$$A = 0.8 \begin{array}{ccc} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{array} + 0.2 \begin{array}{ccc} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{array}$$

- 2 billion entries for vectors, approx 8GB

- Matrix **A** has N^2 entries
  - ☐ 10^{18} is a large number!

$$= \begin{array}{ccc} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{array}$$

Matrix Formulation

☐ Suppose there are **_N_** pages

☐ Consider page **_i_**, with **d_i** out-links

☐ We have **$M_{ji} = 1/|d_i|$** when **_i → j_**
    and **$M_{ji} = 0$** otherwise

☐ **The random teleport is equivalent to:**

- ■ Adding a **teleport link** from **_i_** to every other page and setting transition probability to **$(1-\beta)/N$**

- ■ Reducing the probability of following each out-link from **$1/|d_i|$** to **$\beta/|d_i|$**

- ■ **Equivalent:** Tax each page a fraction **$(1-\beta)$** of its score and redistribute evenly

Rearranging the Equation

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r},$ **where** $\boldsymbol{A_{ji}} = \boldsymbol{\beta}\, \boldsymbol{M_{ji}} + \dfrac{1-\beta}{N}$

- $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$

- $r_j = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \dfrac{1-\beta}{N} \right] \cdot r_i$

$$= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \dfrac{1-\beta}{N} \sum_{i=1}^{N} r_i$$

$$= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \dfrac{1-\beta}{N} \qquad \text{since } \sum r_i = 1$$

- **So we get:** $\boldsymbol{r} = \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + \left[\dfrac{1-\beta}{N} \right]_N$

Note: Here we assumed M has no dead-ends

$[x]_N$ **… a vector of length _N_ with all entries _x_**

Sparse Matrix Formulation

☐ We just rearranged the **PageRank equation**

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

☐ where **[(1-β)/N]$_N$** is a vector with all **N** entries **(1-β)/N**

☐ **M** is a **sparse matrix!** (with no dead-ends)

■ 10 links per node, approx 10N entries

☐ So in each iteration, we need to:

■ Compute $r^{new} = \beta M \cdot r^{old}$

■ Add a constant value **(1-β)/N** to each entry in **r**^{new}

☐ **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

PageRank: The Complete Algorithm

☐ **<u>Input:</u> Graph G and parameter β**

  ▪ Directed graph G (can have **spider traps** and **dead ends**)

  ▪ Parameter β

☐ **<u>Output:</u> PageRank vector r^{new}**

  ▪ **Set:** $r_j^{old} = \dfrac{1}{N}$

  ▪ **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$

    ☐ $\forall j: \ \boldsymbol{r'}_{\boldsymbol{j}}^{\boldsymbol{new}} = \sum_{i \to j} \boldsymbol{\beta} \ \dfrac{r_i^{old}}{d_i}$

      $\boldsymbol{r'}_{\boldsymbol{j}}^{\boldsymbol{new}} = \boldsymbol{0}$ if in-degree of j is **0**

    ☐ **Now re-insert the leaked PageRank:**

    $\forall \boldsymbol{j}: \ \boldsymbol{r}_{\boldsymbol{j}}^{\boldsymbol{new}} = \boldsymbol{r'}_{\boldsymbol{j}}^{\boldsymbol{new}} + \dfrac{1-S}{N}$ **where:** $S = \sum_j r'_j^{new}$

    ☐ $\boldsymbol{r}^{\boldsymbol{old}} = \boldsymbol{r}^{\boldsymbol{new}}$

If the graph has no dead-ends then the amount of leaked PageRank is 1-β. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing S.

Sparse Matrix Encoding

☐ **Encode sparse matrix using only nonzero entries**

- ■ Space proportional roughly to number of links
- ■ Say 10N, or 4*10*1 billion = 40GB
- ■ **Still won't fit in memory, but will fit on disk**

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm: Update Step

- ☐ **Assume enough RAM to fit r^{new} into memory**
  - ■ Store r^{old} and matrix **M** on disk

- ☐ **1 step of power-iteration is:**

  Initialize all entries of $r^{new} = (1-\beta) / N$
  For each page i (of out-degree d_i):
    Read into memory: $i, d_i, dest_1, ..., dest_{d_i}, r^{old}(i)$
    For $j = 1...d_i$
      $r^{new}(dest_j) += \beta \, r^{old}(i) / d_i$

source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23

r^{new} indices: 0, 1, 2, 3, 4, 5, 6

r^{old} indices: 0, 1, 2, 3, 4, 5, 6

Analysis

- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix M on disk
- **In each iteration, we have to:**
  - Read r^{old} and M
  - Write r^{new} back to disk
  - **Cost per iteration of Power method:**
    $= 2|r| + |M|$

- **Question:**
  - What if we could not even fit r^{new} in memory?

Block-based Update Algorithm

	r^{new}		src	degree	destination		r^{old}	
0			0	4	0, 1, 3, 5			**0**
1								**1**
			1	2	0, 5			**2**
2								**3**
3			2	2	3, 4			**4**
				M				**5**
4								
5								
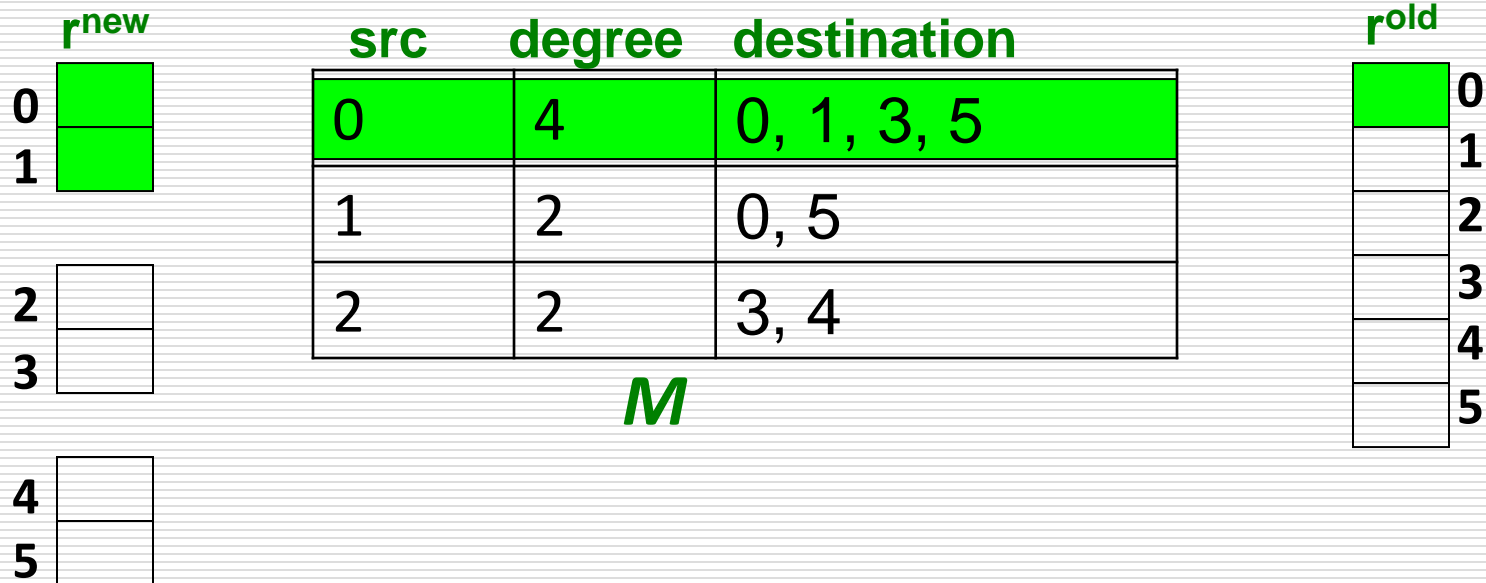
- Break ***r***^{new} into ***k*** blocks that fit in memory
- Scan ***M*** and ***r***^{old} once for each block

Analysis of Block Update

☐ **Similar to nested-loop join in databases**
  - ■ Break r^{new} into k blocks that fit in memory
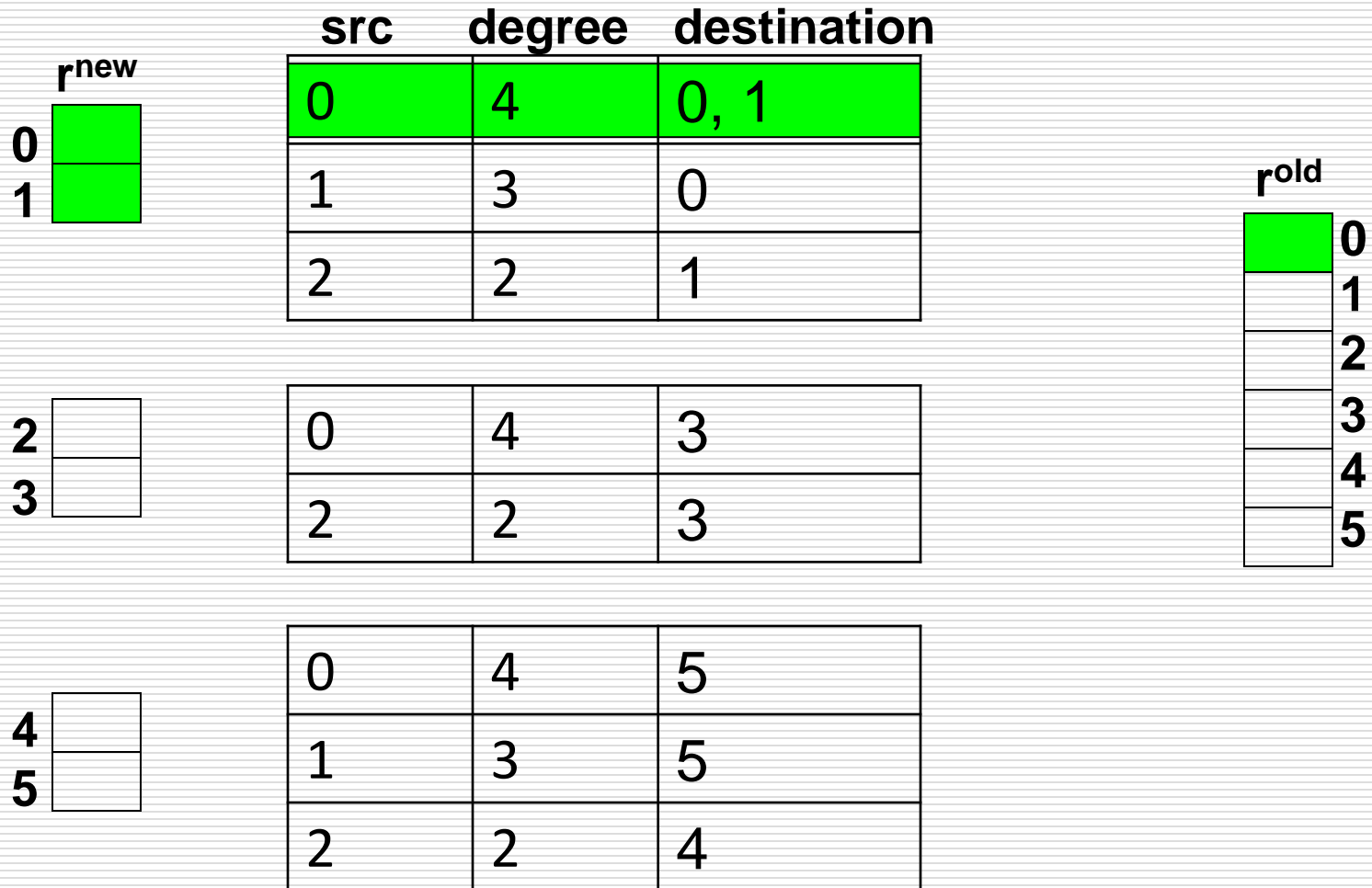 - ■ Scan M and r^{old} once for each block

☐ **Total cost:**
  - ■ k scans of M and r^{old}
  - ■ **Cost per iteration of Power method:**
    $k(|M| + |r|) + |r| = k|M| + (k+1)|r|$

☐ **Can we do better?**
  - ■ **Hint:** M is much bigger than r (approx 10-20x), so we must avoid reading it k times per iteration

Block-Stripe Update Algorithm

r^new

0
1

src	degree	destination
0	4	0, 1
1	3	0
2	2	1

r^old

	0
	1
	2
	3
	4
	5

2
3

0	4	3
2	2	3

4
5

0	4	5
1	3	5
2	2	4

Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of *r*^new

Block-Stripe Analysis

- ☐ Break **_M_** into stripes

    - ■ Each stripe contains only destination nodes in the corresponding block of **_r_**^{new}

- ☐ Some additional overhead per stripe

    - ■ But it is usually worth it

- ☐ **Cost per iteration of Power method:**
  $=|M|(1+\varepsilon) + (k+1)|r|$

Some Problems with Page Rank

- ☐ **Measures generic popularity of a page**
  - ■ Biased against topic-specific authorities
  - ■ **Solution:** Topic-Specific PageRank (**next**)
- ☐ **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

Acknowledgement

- Slides are adapted from:
  - Prof. Jeffrey D. Ullman
  - Dr. Anand Rajaraman
  - Dr. Jure Leskovec

编程大作业-分组说明

- 分组 2023.3.25-2023.3.28（人数与最终成绩无相关性，同组成员成绩相同）

  1~3人一组

  3月27日(周一)24点前将本组组员信息(学号+姓名)发送至

  邮箱: bigdatacomputing@163.com

  逾期未发送分组情况则视为单人一组

  3月30日(周四)发布最终分组情况

- 发布作业 2023.3.30

- 提交作业 2023.4.30

  （分组情况和作业要求会在微信群里发布）