GOOGLE PAGERANK ALGORITHM

Li Zhang
Metis SF16_ds4 Investigation 1

Generic search methodology

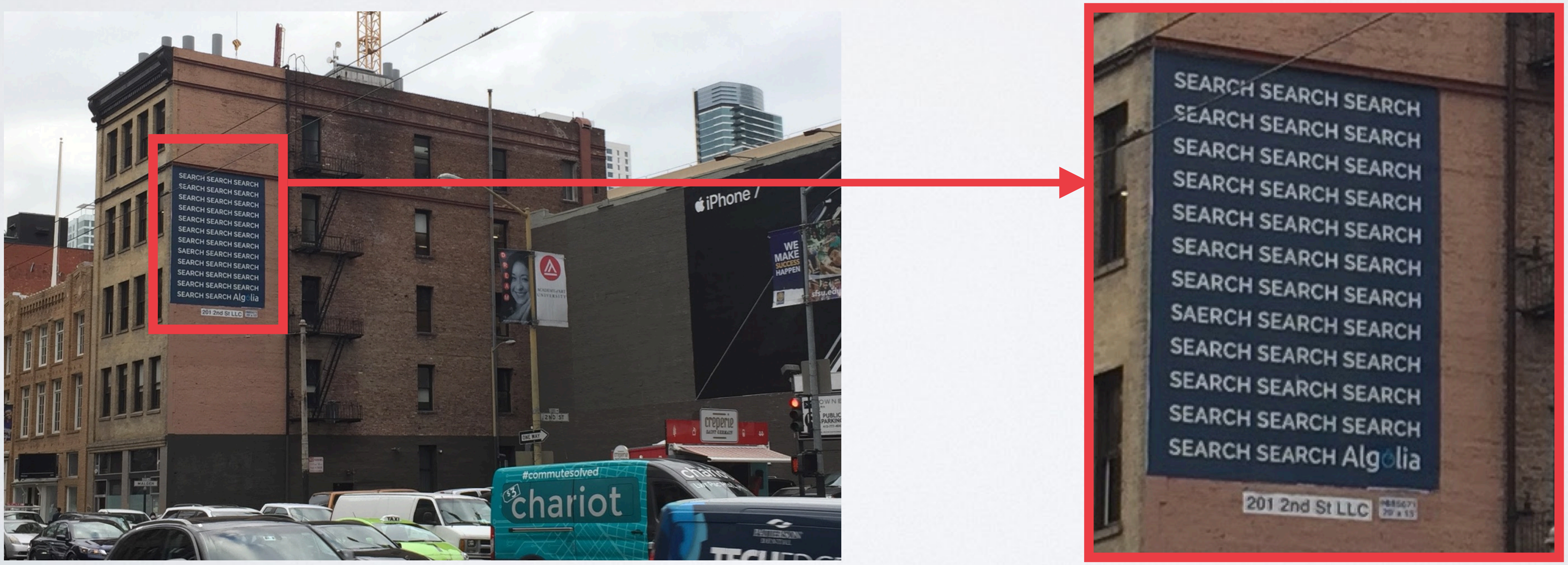
1. Crawl the web and create a repository of pages
2. Receive the search term from the user
3. Locate pages containing the search term
4. Order the page importance with **Metric X**
5. Return the top k results

The pursuit of **X**

Before google: content based metric (e.g. word counts)

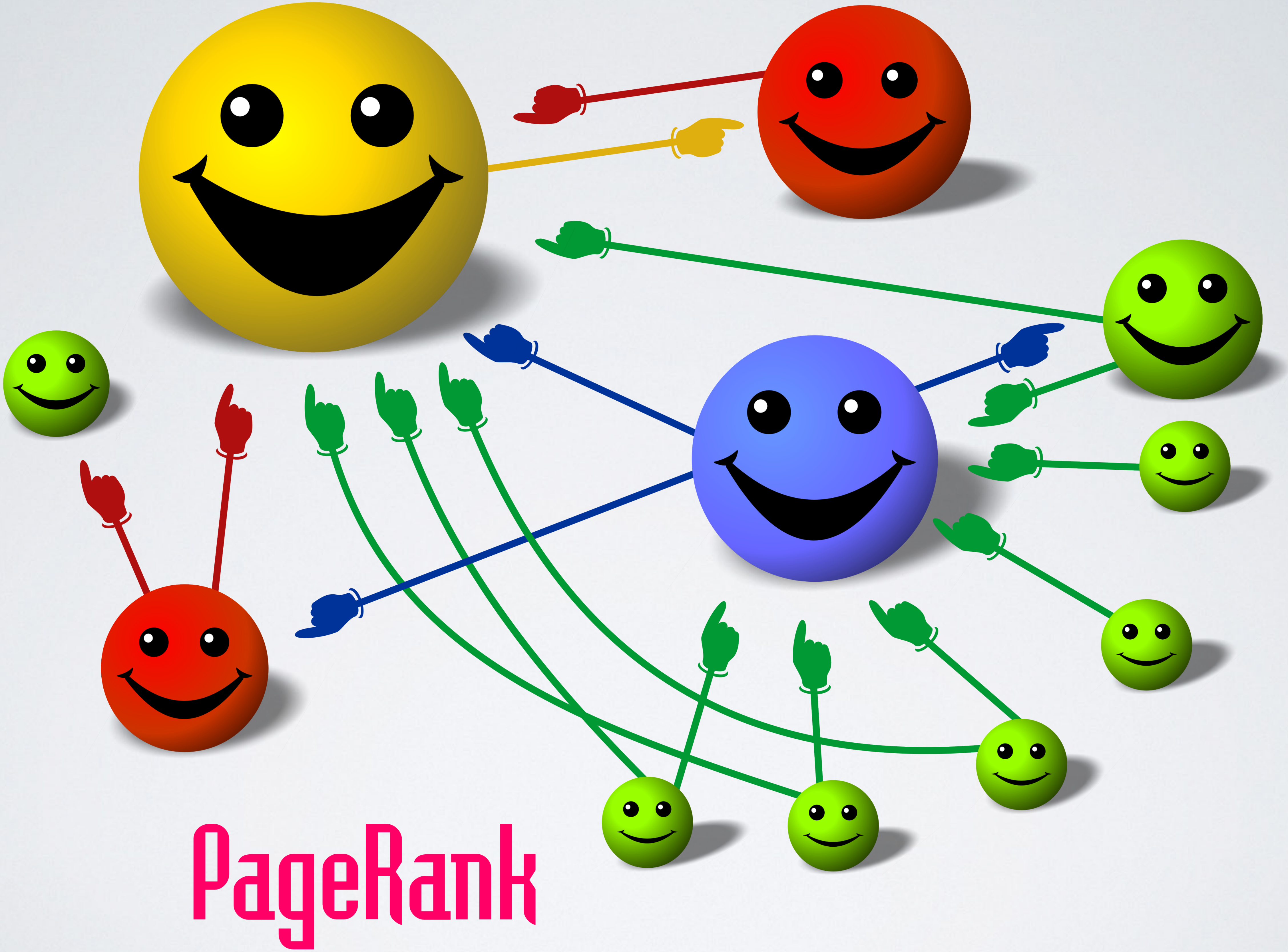
Problem: “junk results”(e.g. search for “search”…)

Example: “…as of November 1997, only one of the top four commercial search engines finds itself…”(1)



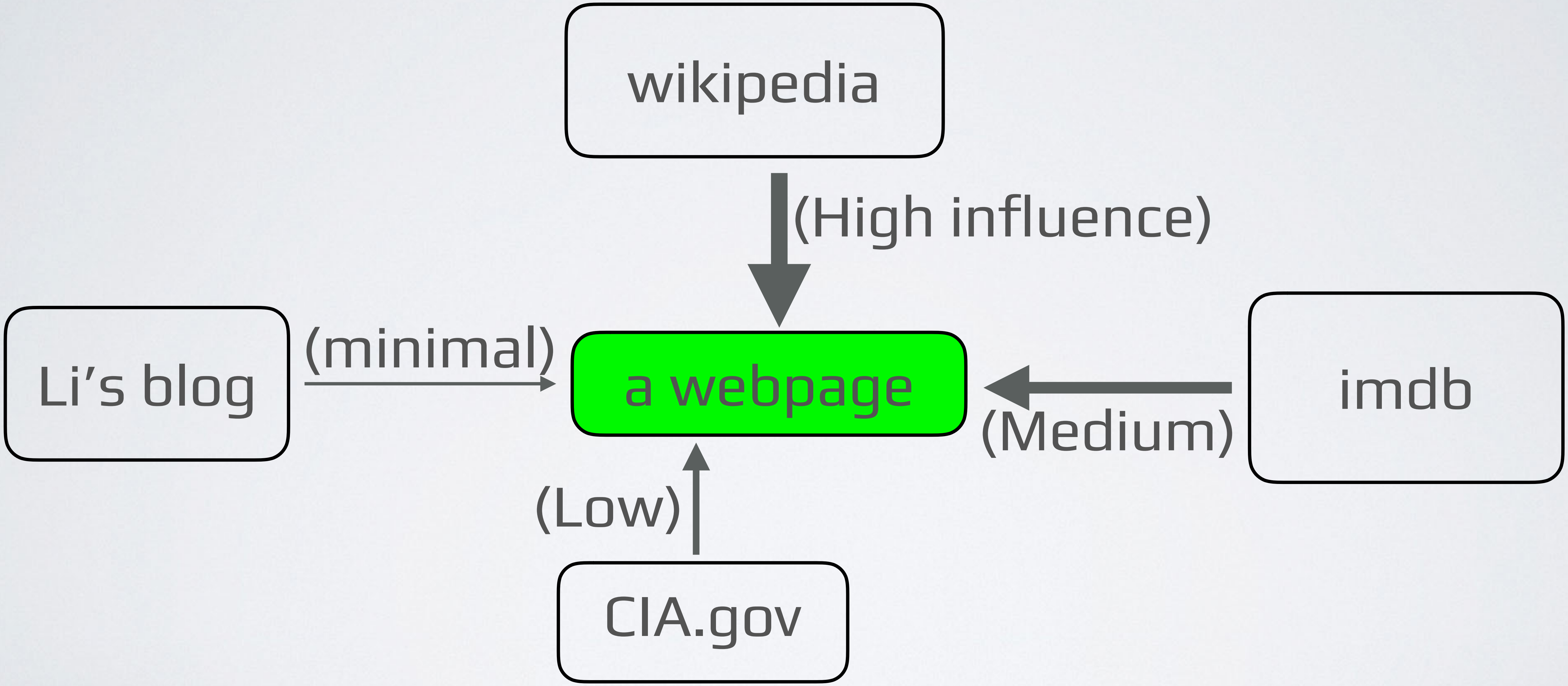
Enter the PageRank* algorithm…

* Inventor: Larry **Page** and Sergey Brin


PageRank

PageRank algorithm

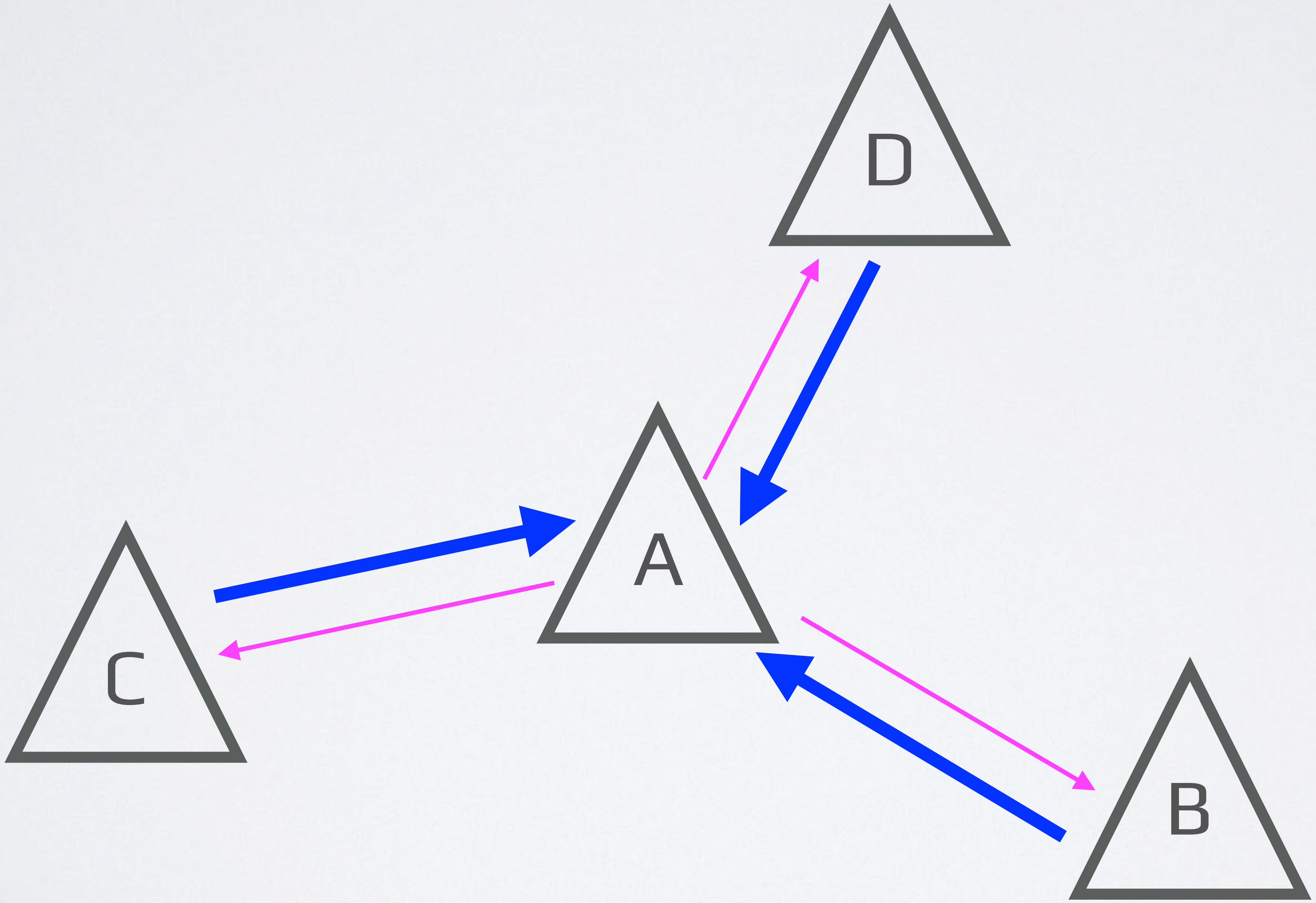
Metric X: **Number** and **quality** of links to a certain page, instead of the content.



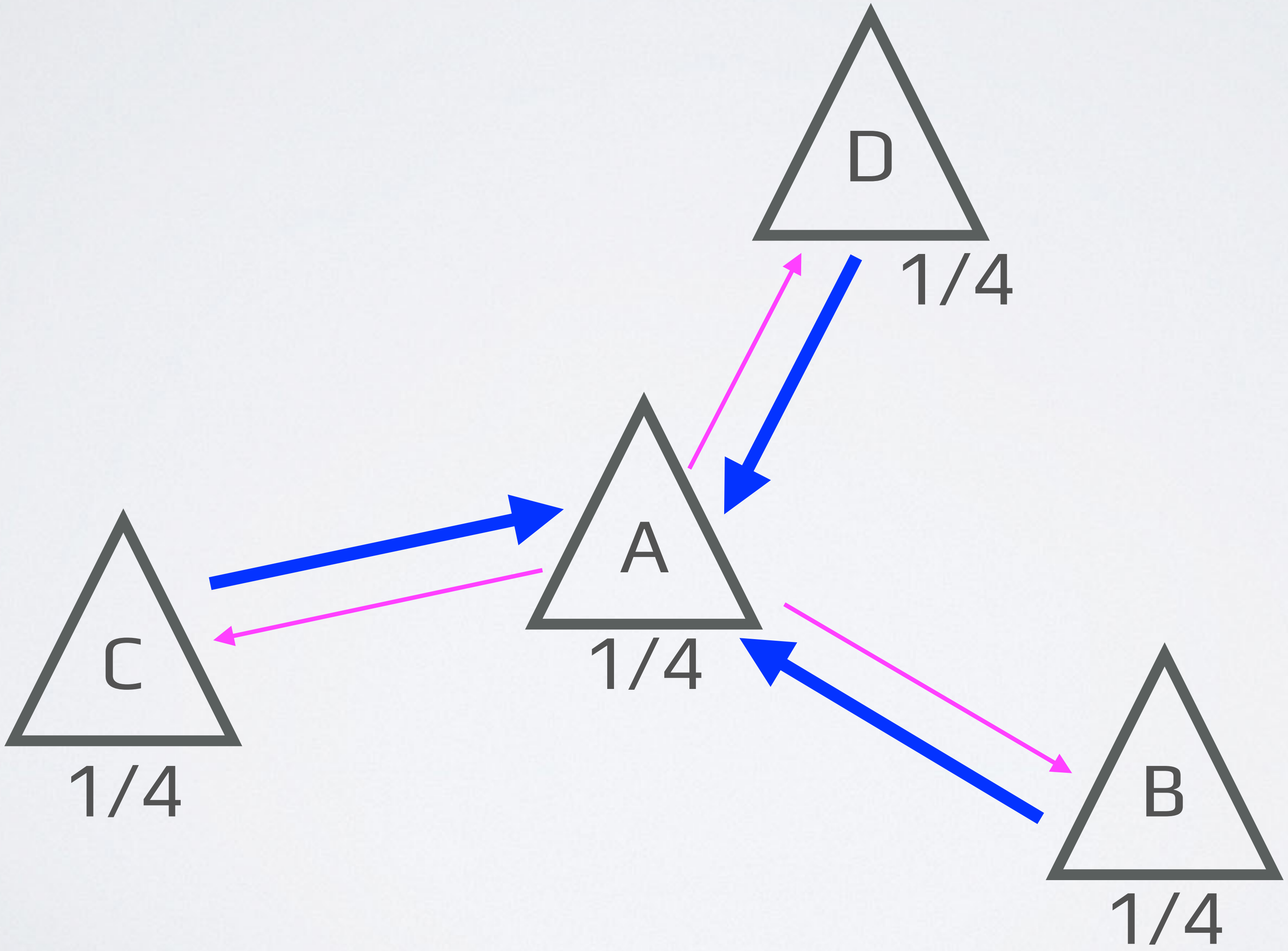
Calculate PageRank/Importance

1. Importance transfers over a link

2. Amount of transfer divided among outgoing links

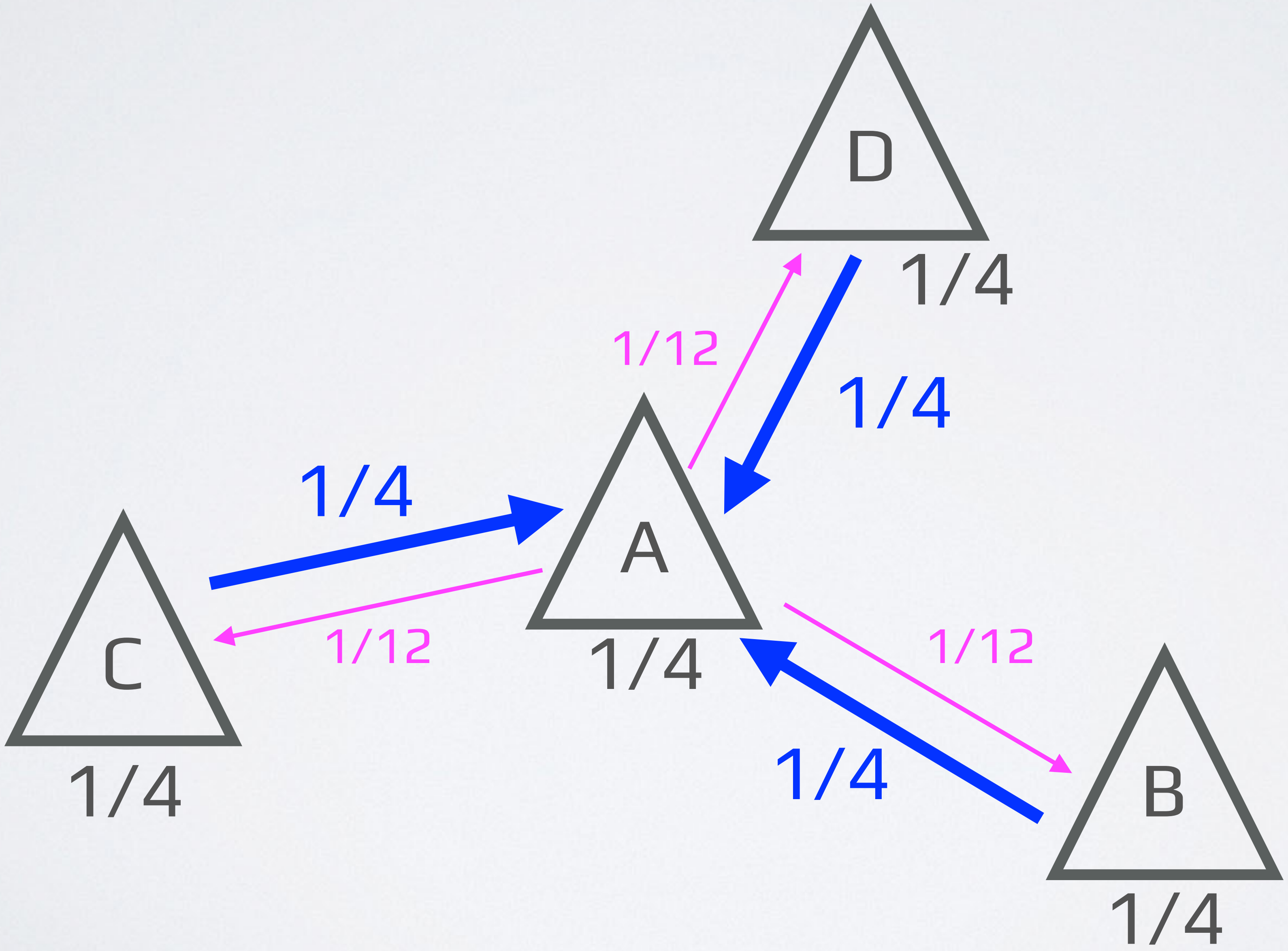


In a mini web galaxy far, far away...
The PageRank is being transferred...



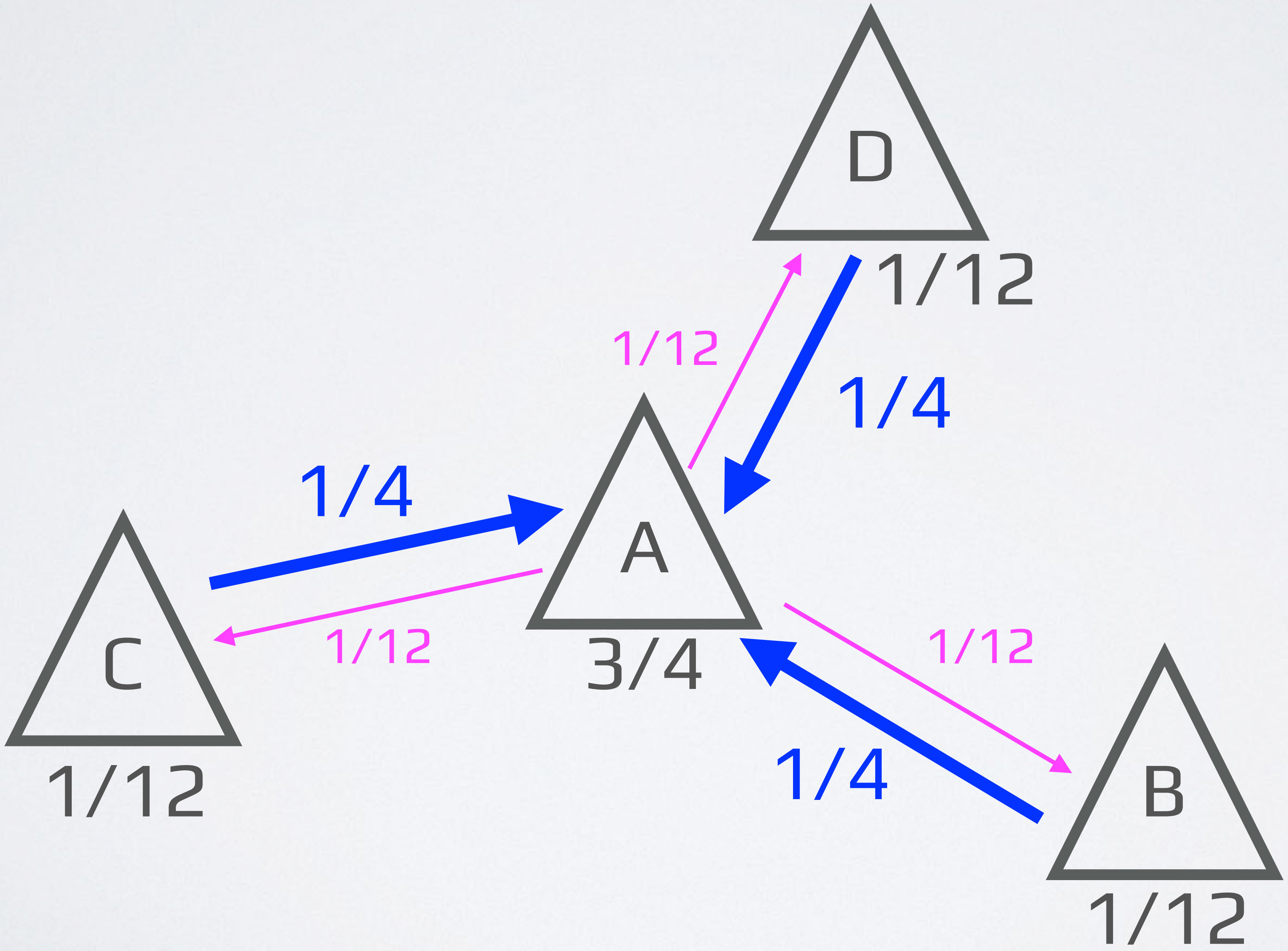
Simplified algorithm

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$



Simplified algorithm

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$



Recalculate using matrix multiplication

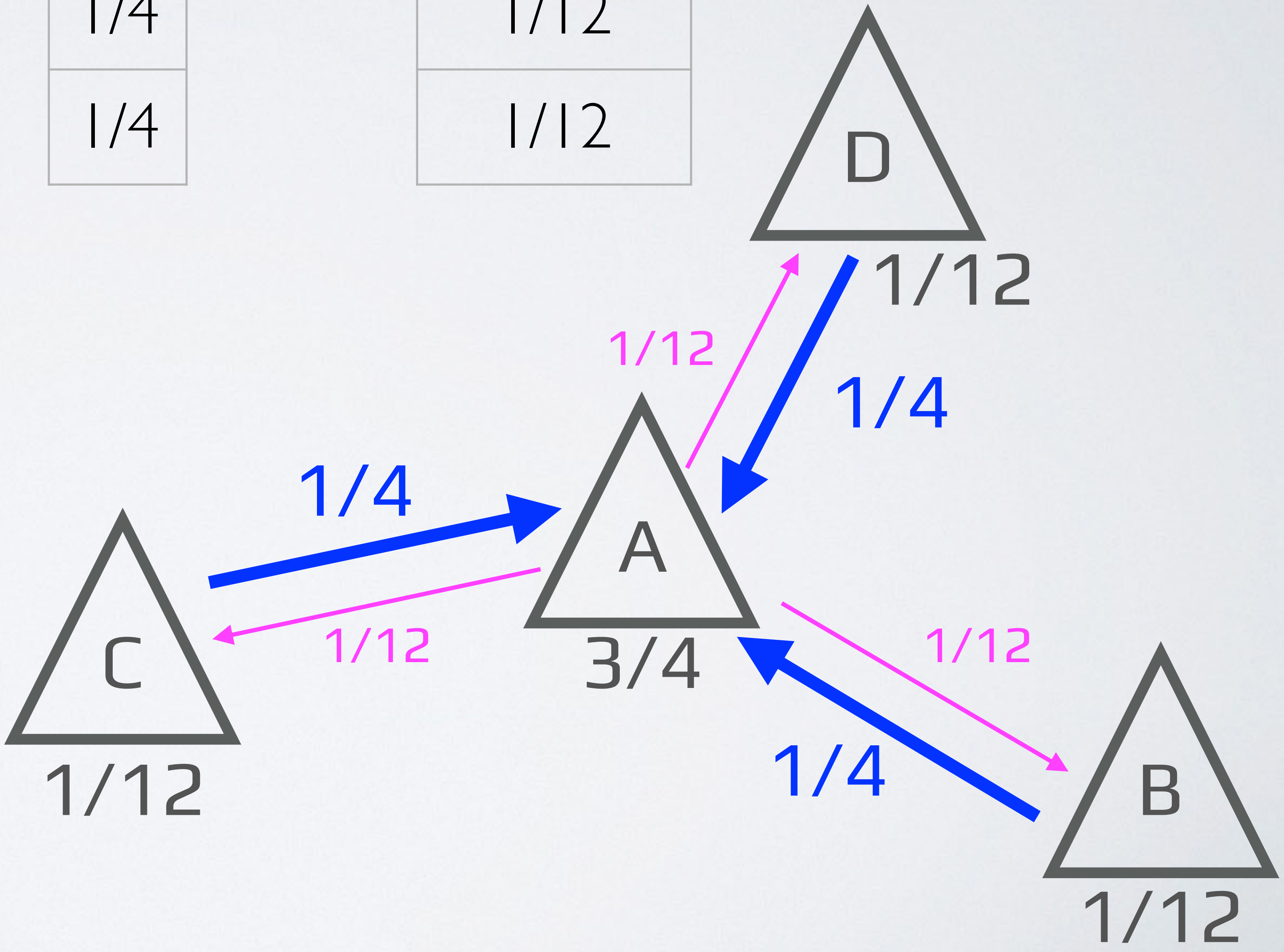
	A	B	C	D
A	0	1	1	1
B	1/3	0	0	0
C	1/3	0	0	0
D	1/3	0	0	0

x

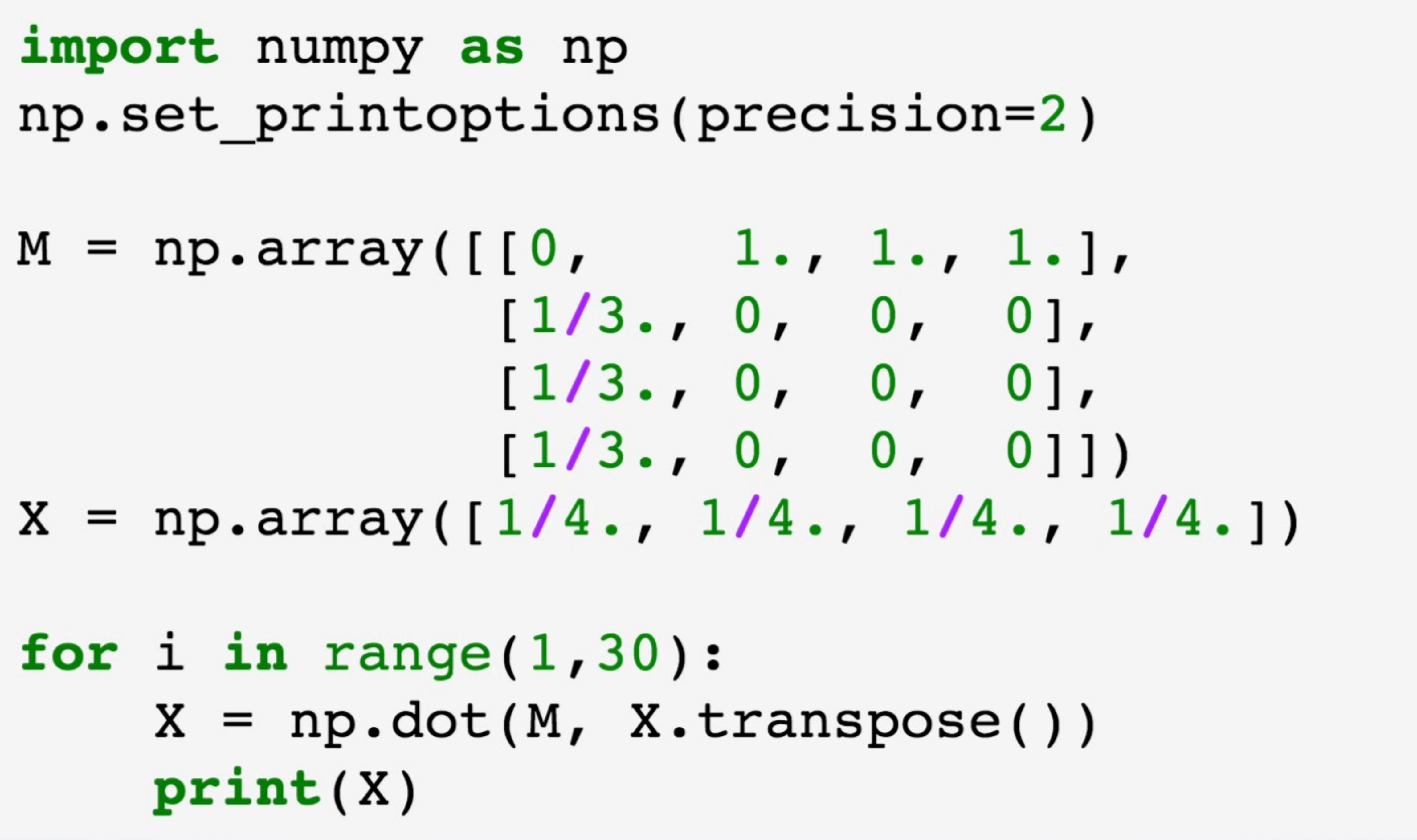
PR
1/4
1/4
1/4
1/4

=

New_PR
3/4
1/12
1/12
1/12



Simulation in Python

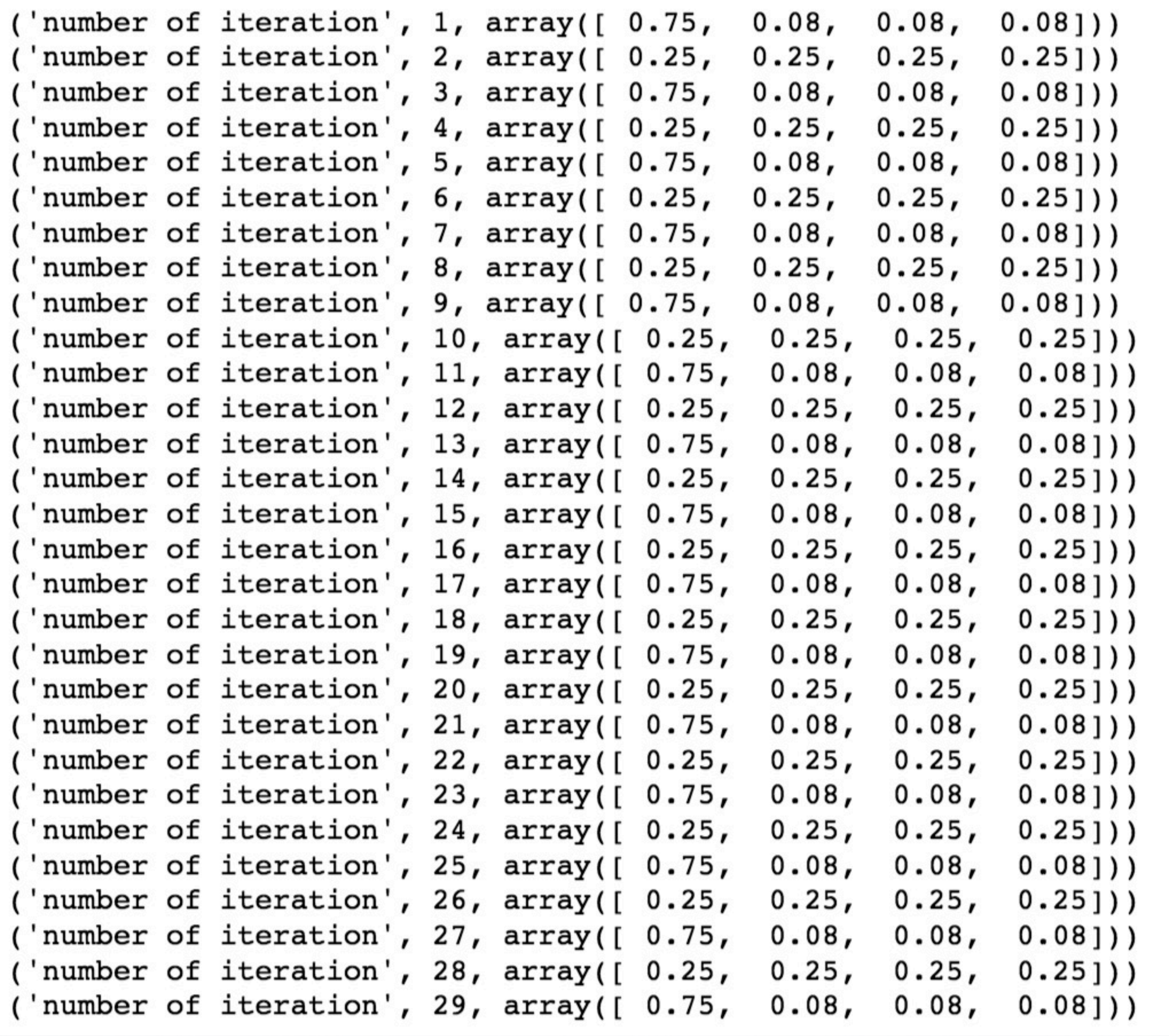


```
import numpy as np
np.set_printoptions(precision=2)

M = np.array([[0,    1., 1., 1.],
              [1/3., 0,  0,  0],
              [1/3., 0,  0,  0],
              [1/3., 0,  0,  0]])
X = np.array([1/4., 1/4., 1/4., 1/4.])

for i in range(1,30):
    X = np.dot(M, X.transpose())
    print(X)
```

It does NOT converge...



```
('number of iteration', 1, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 2, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 3, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 4, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 5, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 6, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 7, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 8, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 9, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 10, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 11, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 12, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 13, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 14, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 15, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 16, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 17, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 18, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 19, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 20, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 21, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 22, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 23, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 24, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 25, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 26, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 27, array([ 0.75,  0.08,  0.08,  0.08]))
('number of iteration', 28, array([ 0.25,  0.25,  0.25,  0.25]))
('number of iteration', 29, array([ 0.75,  0.08,  0.08,  0.08]))
```

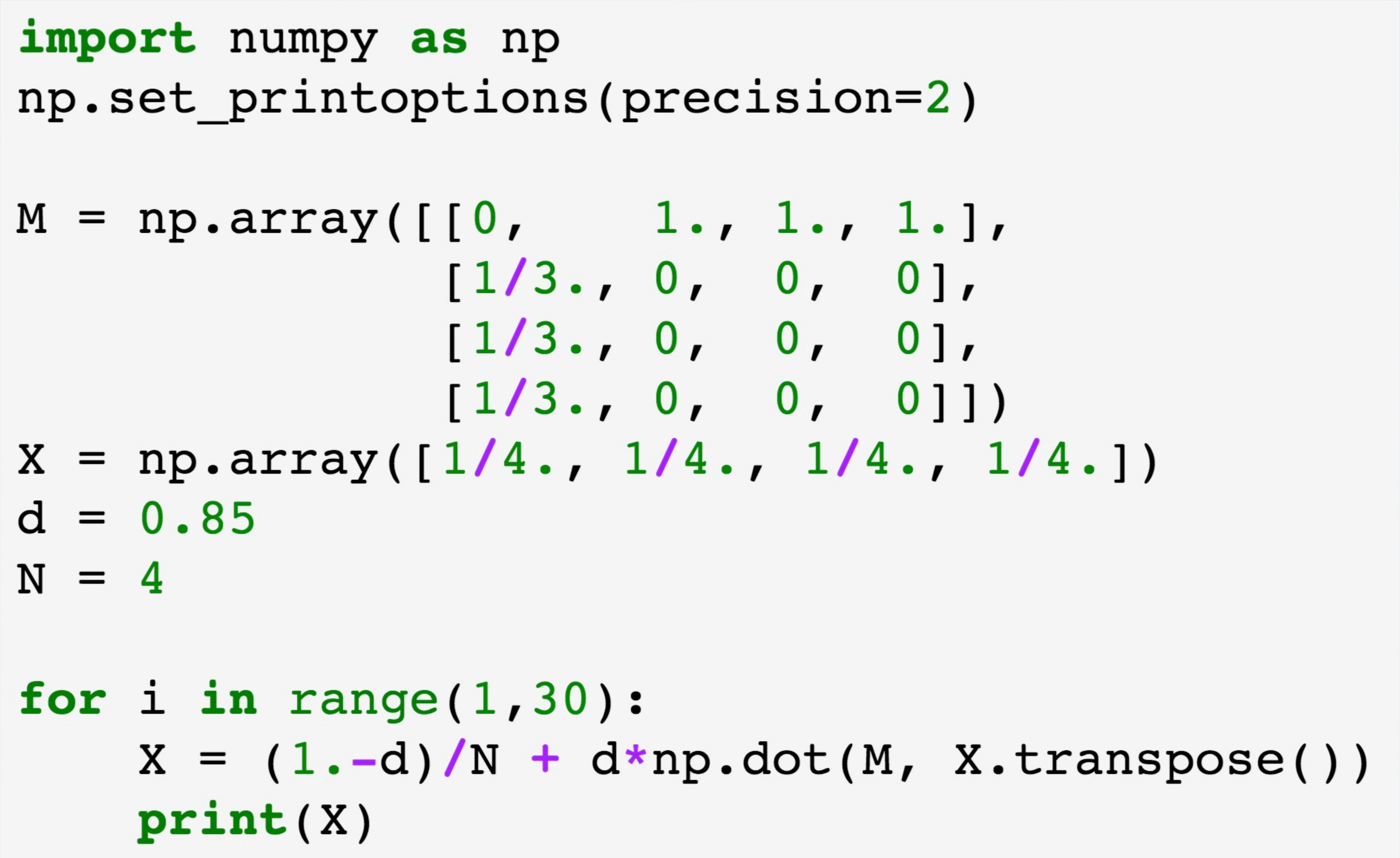
It does NOT converge…

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Introducing the dampening factor **$0 < d < 1$**

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Simulation in Python with dampening factor



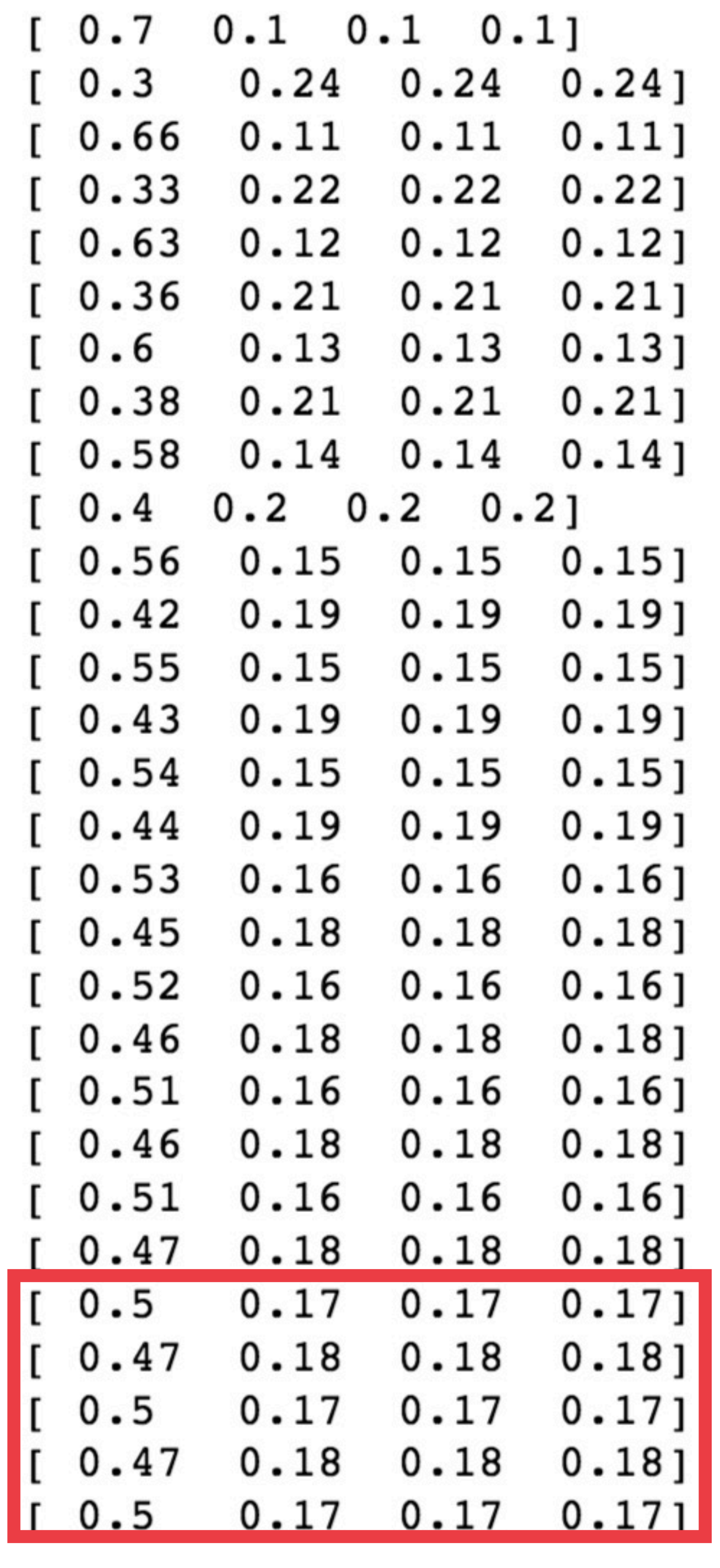
```
import numpy as np
np.set_printoptions(precision=2)

M = np.array([[0,    1., 1., 1.],
              [1/3., 0,  0,  0],
              [1/3., 0,  0,  0],
              [1/3., 0,  0,  0]])
X = np.array([1/4., 1/4., 1/4., 1/4.])
d = 0.85
N = 4

for i in range(1,30):
    X = (1.-d)/N + d*np.dot(M, X.transpose())
    print(X)
```

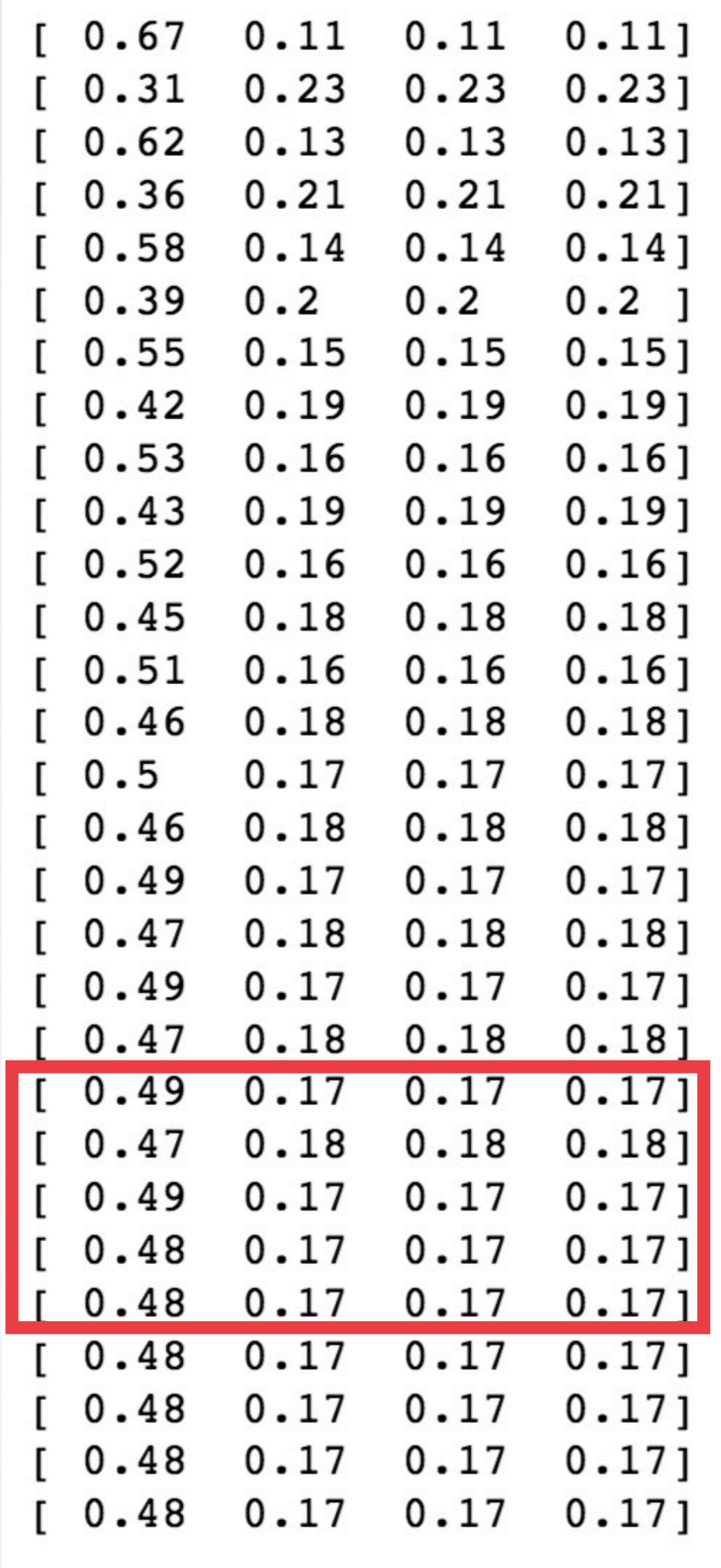
PageRank converges faster with smaller d

d = 0.9



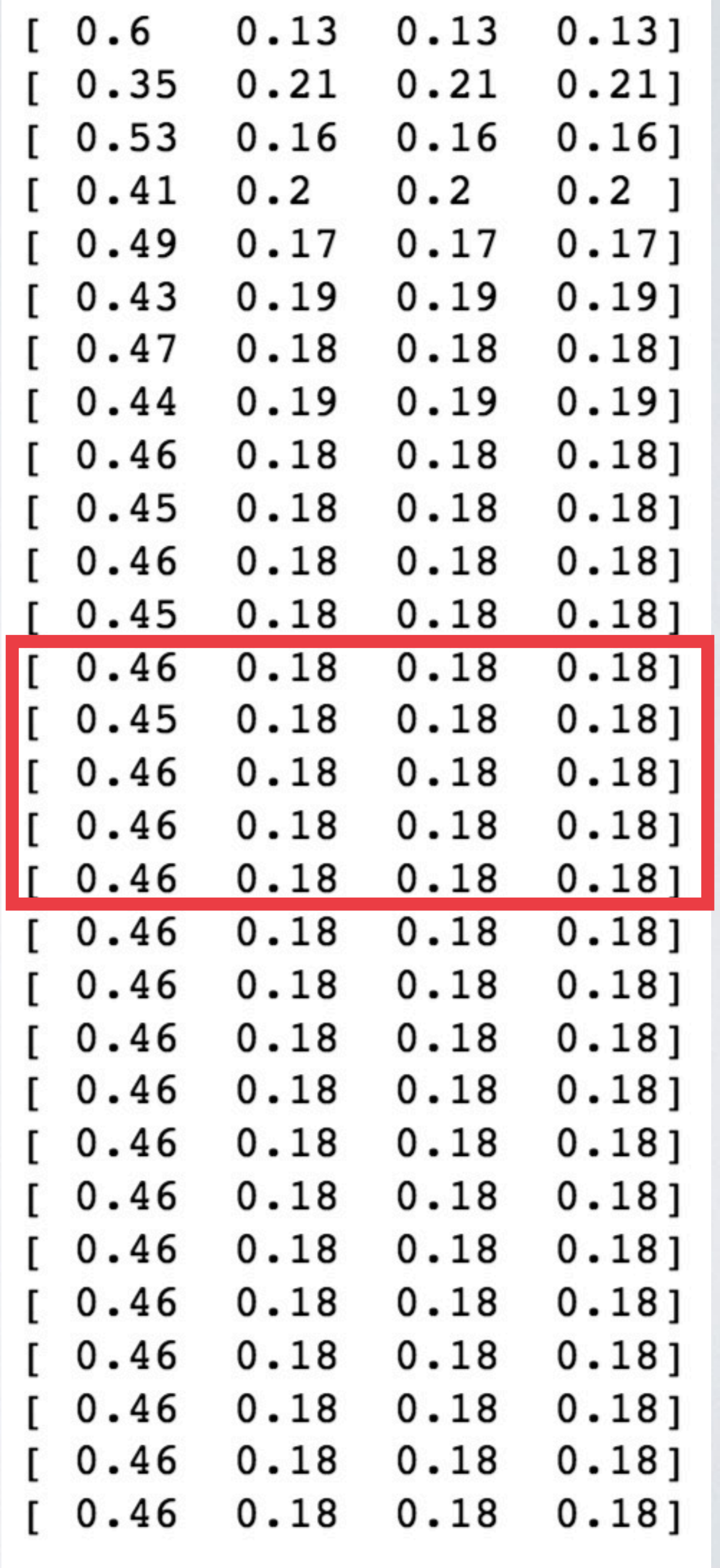
```
[ 0.7   0.1   0.1   0.1]
[ 0.3    0.24   0.24   0.24]
[ 0.66   0.11   0.11   0.11]
[ 0.33   0.22   0.22   0.22]
[ 0.63   0.12   0.12   0.12]
[ 0.36   0.21   0.21   0.21]
[ 0.6    0.13   0.13   0.13]
[ 0.38   0.21   0.21   0.21]
[ 0.58   0.14   0.14   0.14]
[ 0.4   0.2   0.2   0.2]
[ 0.56   0.15   0.15   0.15]
[ 0.42   0.19   0.19   0.19]
[ 0.55   0.15   0.15   0.15]
[ 0.43   0.19   0.19   0.19]
[ 0.54   0.15   0.15   0.15]
[ 0.44   0.19   0.19   0.19]
[ 0.53   0.16   0.16   0.16]
[ 0.45   0.18   0.18   0.18]
[ 0.52   0.16   0.16   0.16]
[ 0.46   0.18   0.18   0.18]
[ 0.51   0.16   0.16   0.16]
[ 0.46   0.18   0.18   0.18]
[ 0.51   0.16   0.16   0.16]
[ 0.47   0.18   0.18   0.18]
[ 0.5    0.17   0.17   0.17]
[ 0.47   0.18   0.18   0.18]
[ 0.5    0.17   0.17   0.17]
[ 0.47   0.18   0.18   0.18]
[ 0.5    0.17   0.17   0.17]
```

d = 0.85



```
[ 0.67   0.11   0.11   0.11]
[ 0.31   0.23   0.23   0.23]
[ 0.62   0.13   0.13   0.13]
[ 0.36   0.21   0.21   0.21]
[ 0.58   0.14   0.14   0.14]
[ 0.39   0.2    0.2    0.2 ]
[ 0.55   0.15   0.15   0.15]
[ 0.42   0.19   0.19   0.19]
[ 0.53   0.16   0.16   0.16]
[ 0.43   0.19   0.19   0.19]
[ 0.52   0.16   0.16   0.16]
[ 0.45   0.18   0.18   0.18]
[ 0.51   0.16   0.16   0.16]
[ 0.46   0.18   0.18   0.18]
[ 0.5    0.17   0.17   0.17]
[ 0.46   0.18   0.18   0.18]
[ 0.49   0.17   0.17   0.17]
[ 0.47   0.18   0.18   0.18]
[ 0.49   0.17   0.17   0.17]
[ 0.47   0.18   0.18   0.18]
[ 0.49   0.17   0.17   0.17]
[ 0.47   0.18   0.18   0.18]
[ 0.49   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
[ 0.48   0.17   0.17   0.17]
```

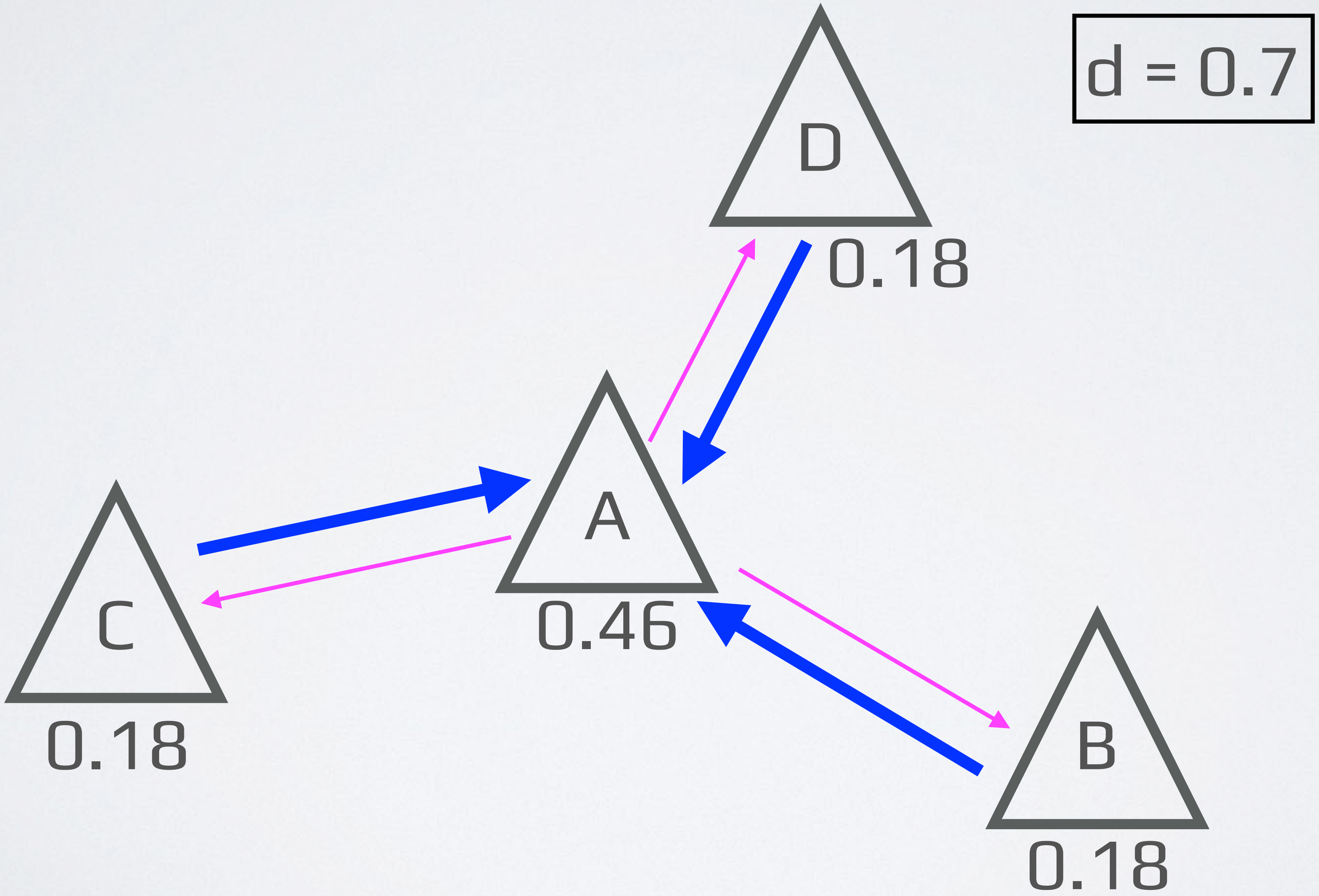
d = 0.7



```
[ 0.6    0.13   0.13   0.13]
[ 0.35   0.21   0.21   0.21]
[ 0.53   0.16   0.16   0.16]
[ 0.41   0.2    0.2    0.2 ]
[ 0.49   0.17   0.17   0.17]
[ 0.43   0.19   0.19   0.19]
[ 0.47   0.18   0.18   0.18]
[ 0.44   0.19   0.19   0.19]
[ 0.46   0.18   0.18   0.18]
[ 0.45   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.45   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.45   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
[ 0.46   0.18   0.18   0.18]
```

Page A has the highest PR value in the equilibrium

Expected: Page A is the “hub”



Modeling web surfing behavior

1. a random surfer given random page
2. surfer keeps clicking on links
3. surfer gets bored
4. surfer requests a random page (prob. = 1 - d)

Modeling web surfing behavior

1. a random surfer given random page
2. surfer keeps clicking on links
3. surfer gets bored
4. surfer requests a random page (prob. = 1 - d)

PageRank is the **probability of arriving at a page** after a large number of clicks

Extreme case: d = 0 (1-d = 1)
The surfer never hits any link and always requests another page. All pages have equal PR = 1/N

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

The pursuit of **X**

For Google search, **X = PageRank value with d = 0.85**

1. Crawl the web and create a repository of pages
2. Receive the search term from the user
3. Locate pages containing the search term
4. Order the page importance using **PageRank(d = 0.85)**
5. Return the top k results

The pros and cons of PageRank

Pros:

-Avoid Junk results

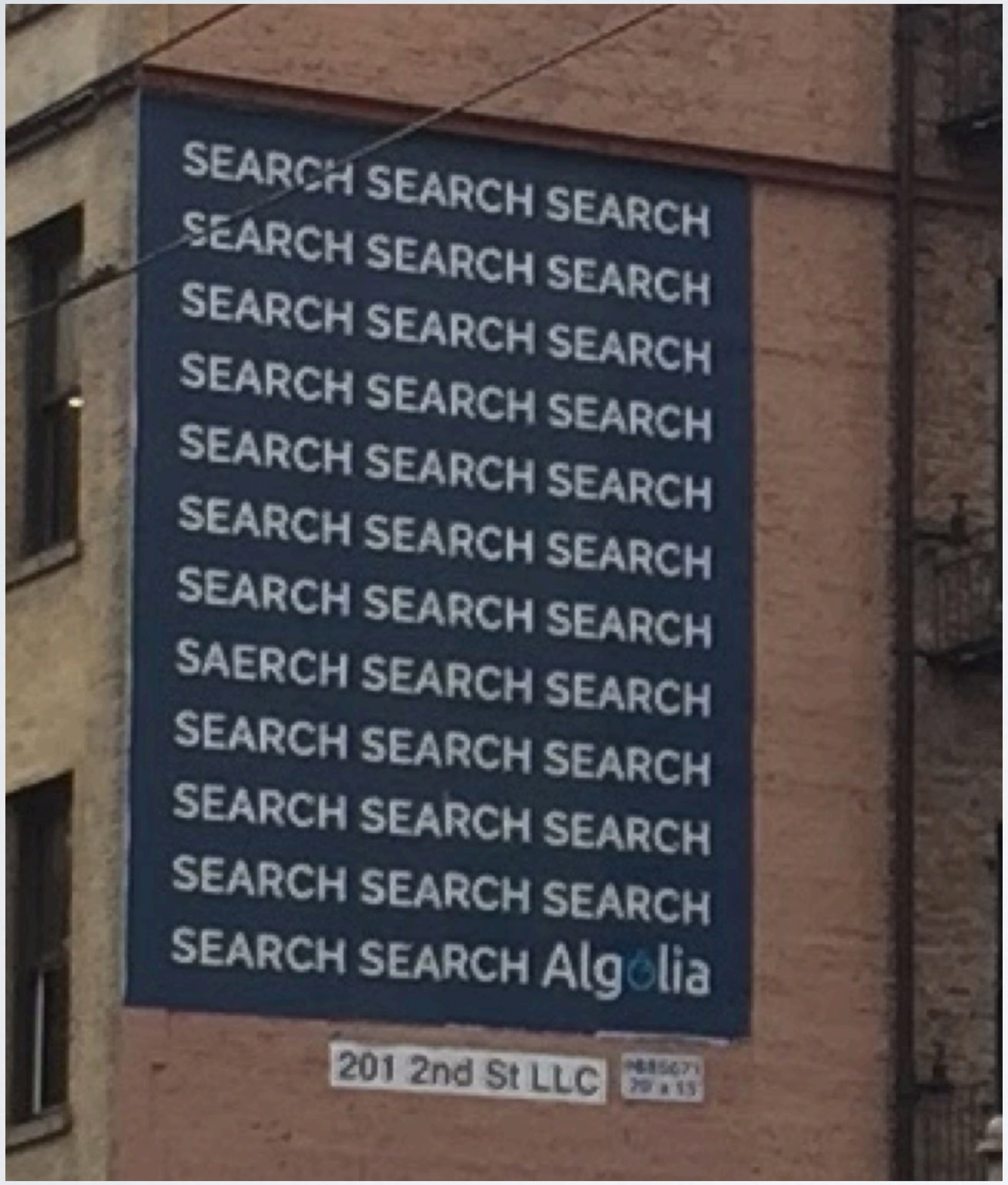
-Great scalability: complexity O(log N)

322 million links converge with 52 iterations[1]

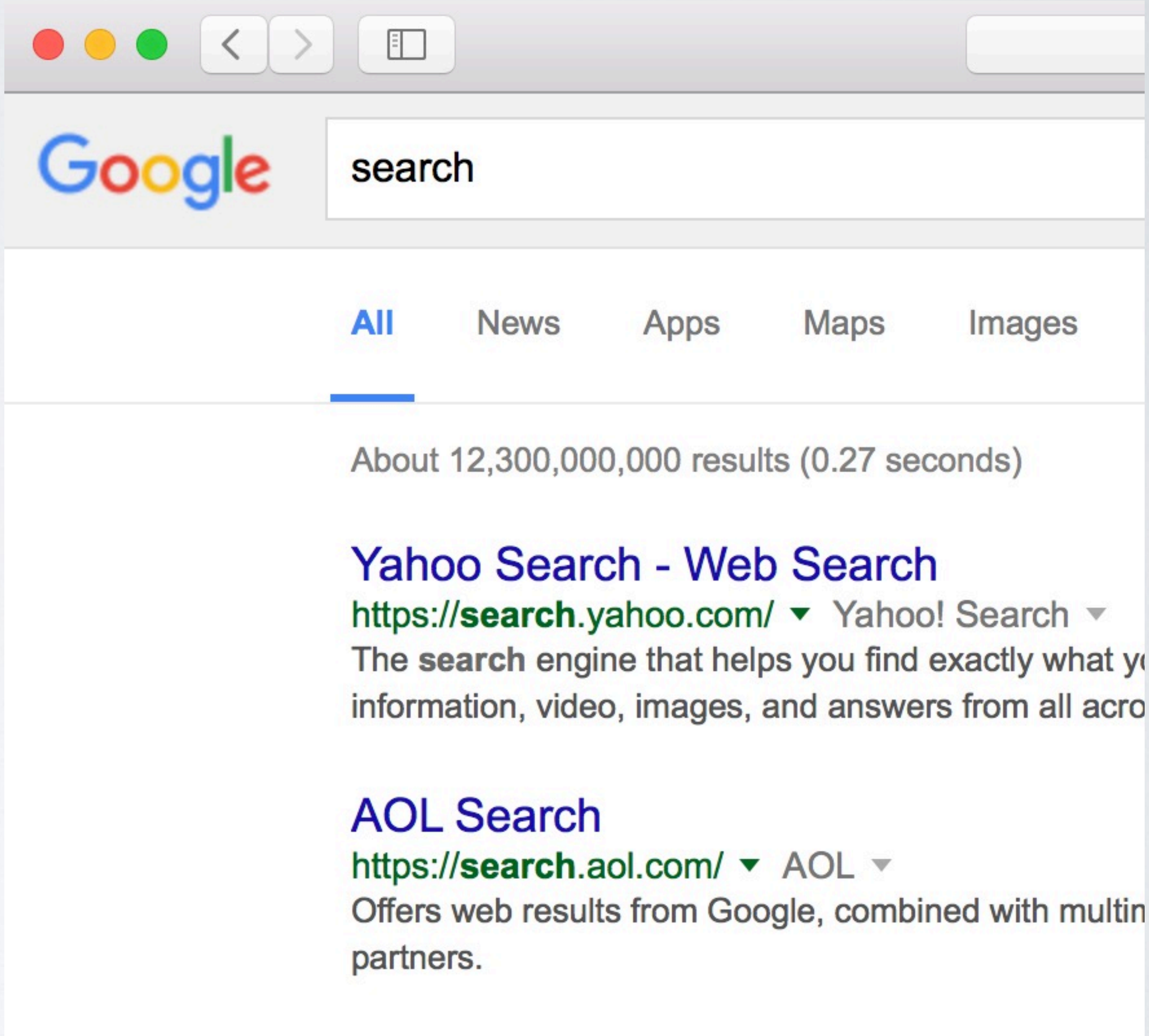
Cons:

-Favors older pages[2]

Search for word “search” on google yields…



Search for word "search" on google yields...



search

About 12,300,000,000 results (0.27 seconds)

Yahoo Search - Web Search
https://search.yahoo.com/ ▾ Yahoo! Search ▾
The **search** engine that helps you find exactly what y
information, video, images, and answers from all acro

AOL Search
https://search.aol.com/ ▾ AOL ▾
Offers web results from Google, combined with multin
partners.

References

1. http://infolab.stanford.edu/~backrub/google.html

2. https://en.wikipedia.org/wiki/PageRank

3. http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

4. http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm

5. http://www.pagerank.dk/Pagerank-formula/Damping-factor.htm