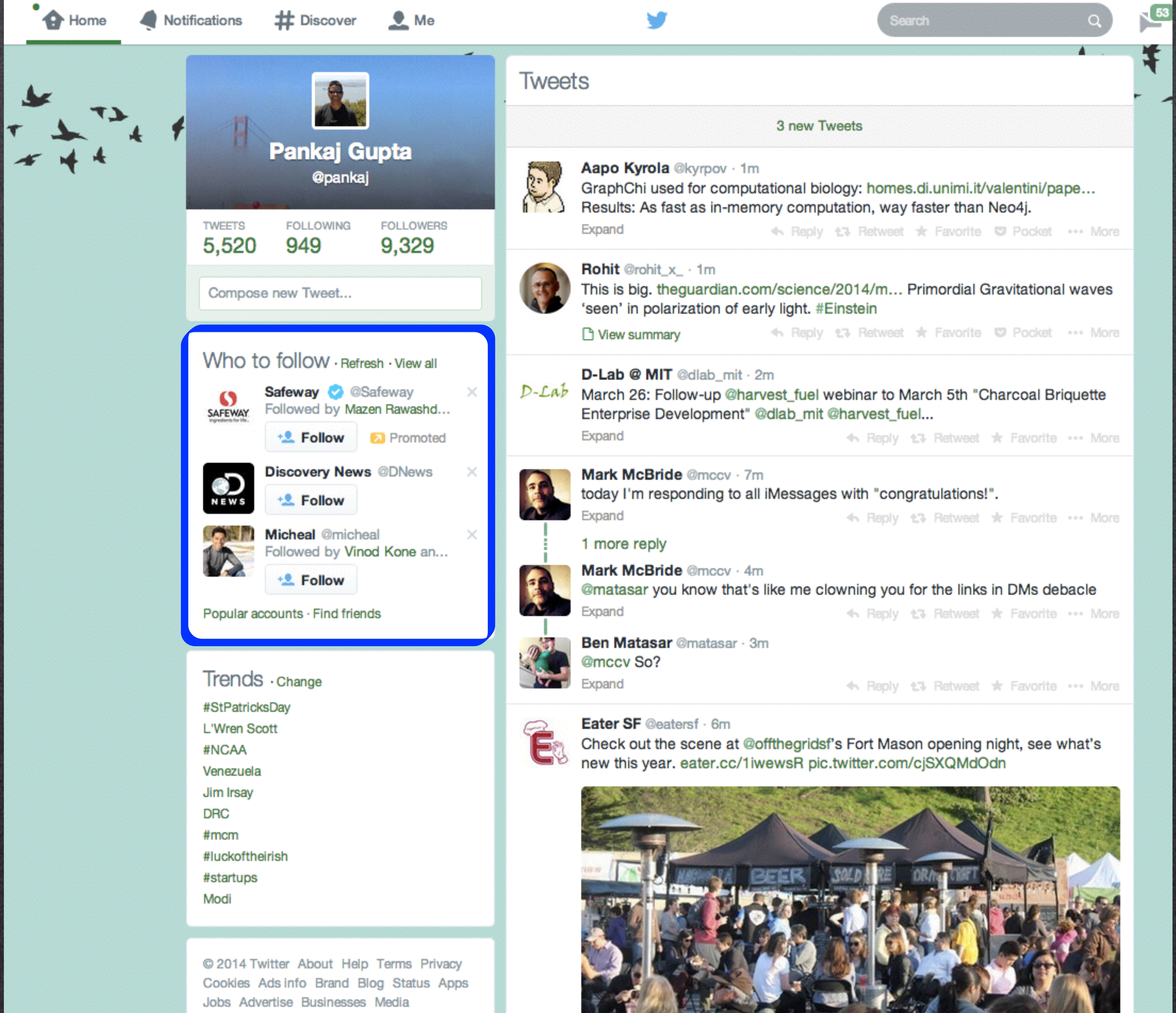
Two Random Walks that Surprise

Ashish Goel
Stanford University

PageRank

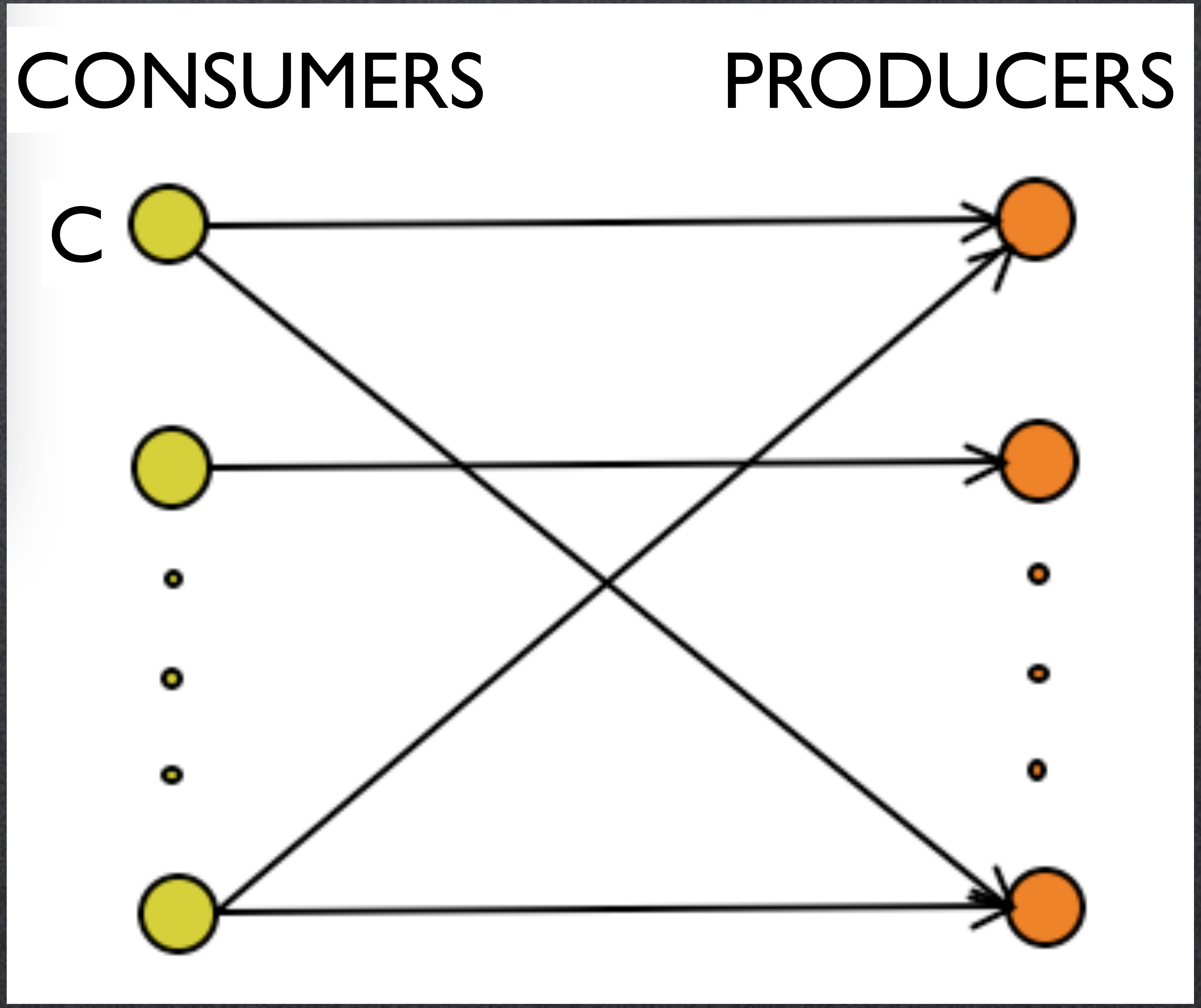

Home
Notifications
Discover
Me
Search
53
Pankaj Gupta
@pankaj
TWEETS
5,520
FOLLOWING
949
FOLLOWERS
9,329
Compose new Tweet...
Who to follow · Refresh · View all
Safeway @Safeway
Followed by Mazen Rawashd...
Follow
Promoted
Discovery News @DNews
Follow
Micheal @micheal
Followed by Vinod Kone an...
Follow
Popular accounts · Find friends
Trends · Change
#StPatricksDay
L'Wren Scott
#NCAA
Venezuela
Jim Irsay
DRC
#mcm
#luckoftheirish
#startups
Modi
© 2014 Twitter About Help Terms Privacy
Cookies Ads info Brand Blog Status Apps
Jobs Advertise Businesses Media
Tweets
3 new Tweets
Aapo Kyrola @kyrpov · 1m
GraphChi used for computational biology: homes.di.unimi.it/valentini/pape...
Results: As fast as in-memory computation, way faster than Neo4j.
Expand
Reply Retweet Favorite Pocket More
Rohit @rohit_x_ · 1m
This is big. theguardian.com/science/2014/m... Primordial Gravitational waves 'seen' in polarization of early light. #Einstein
View summary
Reply Retweet Favorite Pocket More
D-Lab @ MIT @dlab_mit · 2m
March 26: Follow-up @harvest_fuel webinar to March 5th "Charcoal Briquette Enterprise Development" @dlab_mit @harvest_fuel...
Expand
Reply Retweet Favorite More
Mark McBride @mccv · 7m
today I'm responding to all iMessages with "congratulations!".
Expand
Reply Retweet Favorite More
1 more reply
Mark McBride @mccv · 4m
@matasar you know that's like me clowning you for the links in DMs debacle
Expand
Reply Retweet Favorite More
Ben Matasar @matasar · 3m
@mccv So?
Expand
Reply Retweet Favorite More
Eater SF @eatersf · 6m
Check out the scene at @offthegridsf's Fort Mason opening night, see what's new this year. eater.cc/1iwewsR pic.twitter.com/cjSXQMdOdn
BEER

Collaborative Filtering

To get recommendations for C, compute similarity scores for all consumers, and relevance scores for all producers, with respect to C

1. Start with sim(C) = 1

2. Propagate similarity scores along graph edges to compute relevance scores, and vice-versa



Many propagation methods; Often, a linear system of equations

Collaborative Filter: Love or Money

How should we do this propagation? Two extremes:

LOVE: All the similarity score of a consumer X gets transferred to each producer that X follows, and the same in the reverse direction

- Analogous to Singular Value Decompositions in the dense graph limit (HITS)

MONEY: If X follows d producers, then a fraction 1/d of the similarity score of X gets transferred to each producer that X follows (SALSA)

Personalized PageRank

Given a consumer C, perform a random walk on the Follow graph. If the walk is at node v, then the walk:

- Jumps back to node C with probability α
- Follows a random edge out of v with probability 1 – α

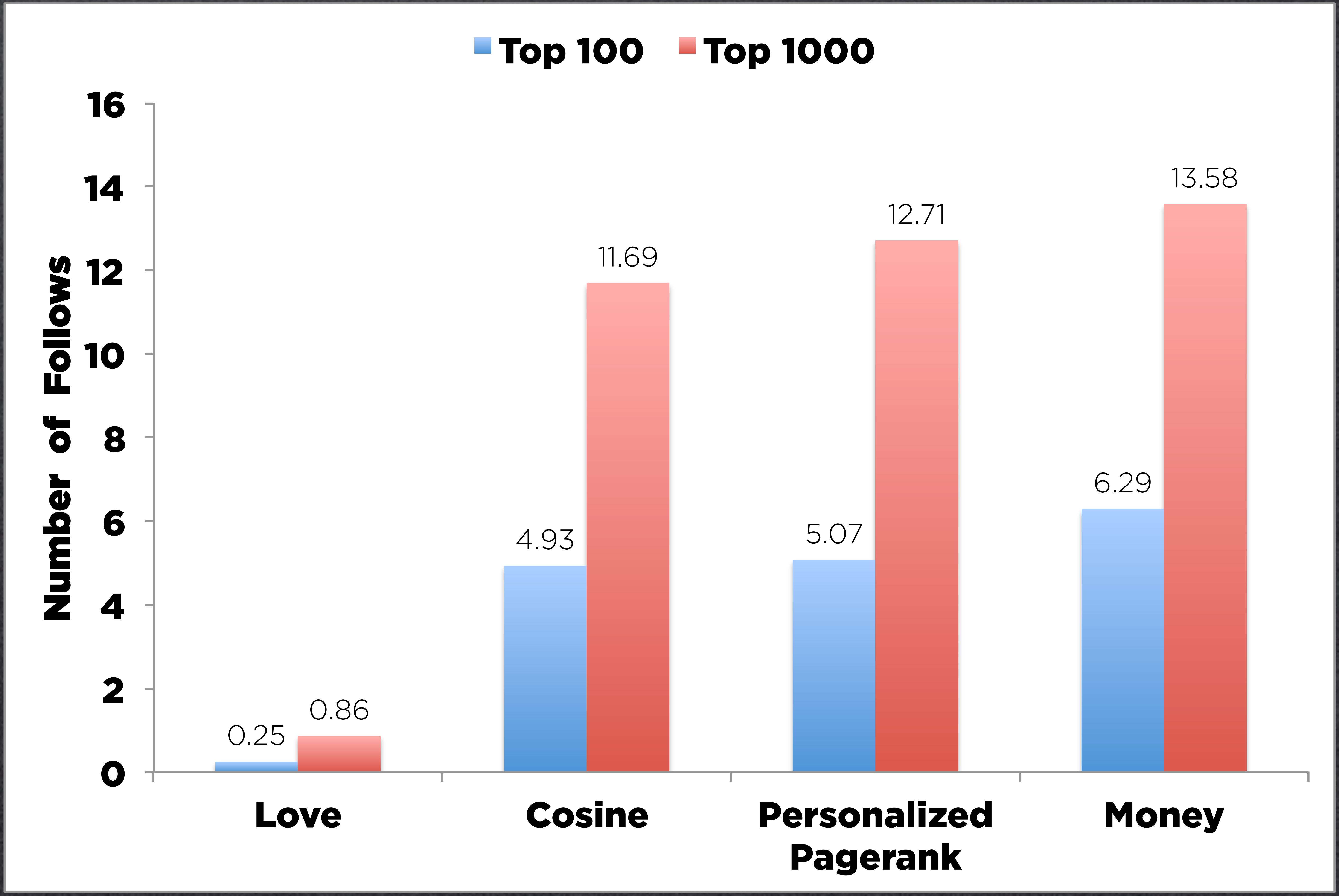
The Personalized PageRank of node Y is the weight of Y in the stationary distribution of this random walk

SALSA/Money is just Personalized PageRank run on the undirected consumer-producer graph

A Dark Test

Run various algorithms to predict follows, but don't display the results. Instead, just observe how many of the top predictions get followed organically

[Bahmani, Chowdhury, Goel; 2010]

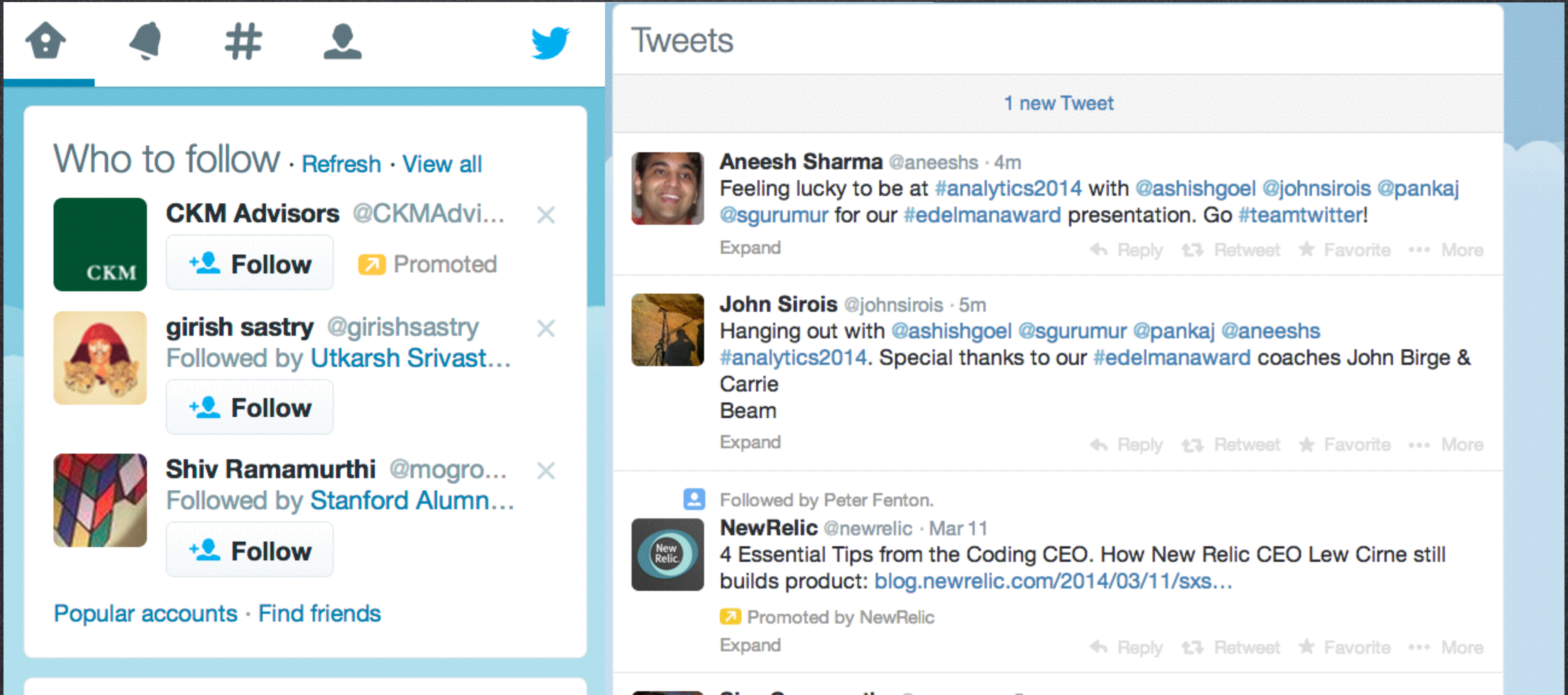


Strategic Impact

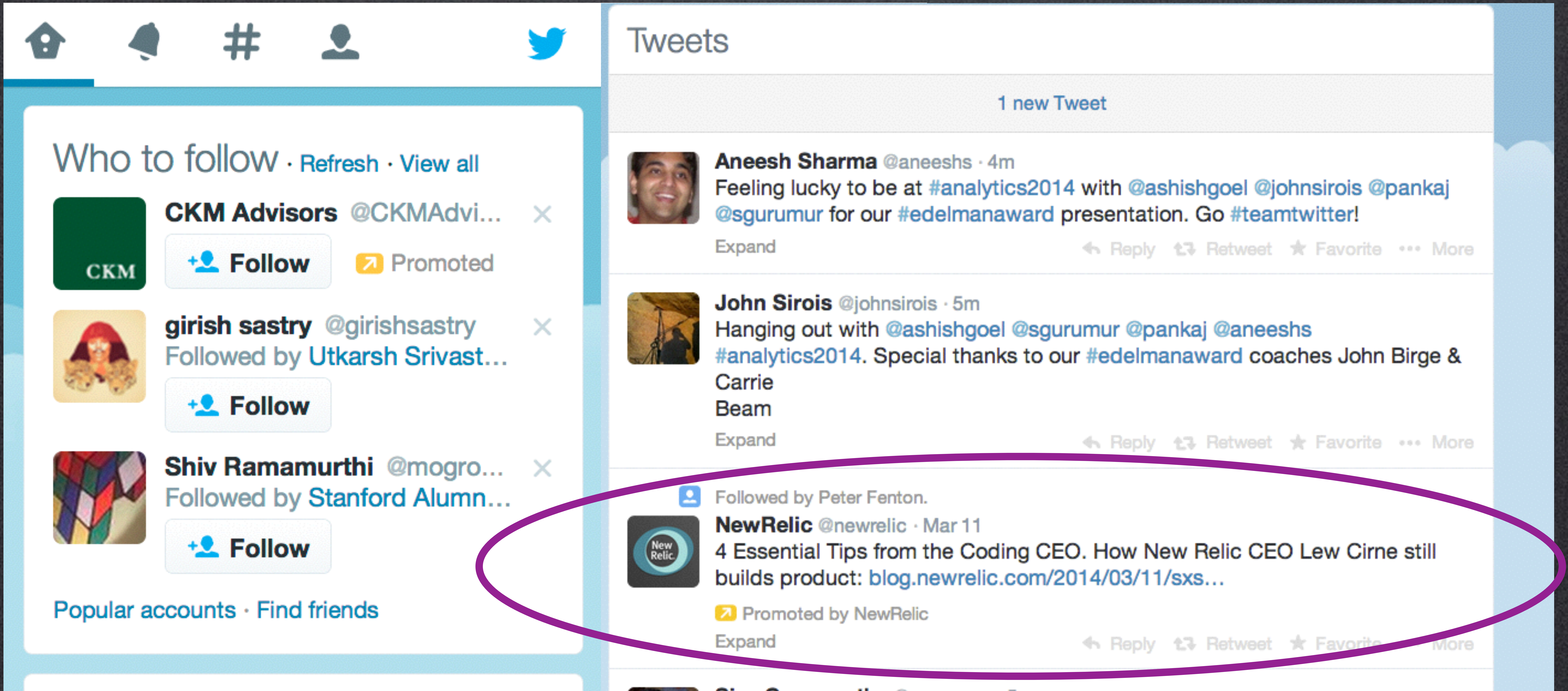
Creates billions of new follows every year

- More than 1/8 of new follows are directly via the Who-to-Follow module
- More than 15% of active users (> 36 Million users) make at least one follow every month via this module

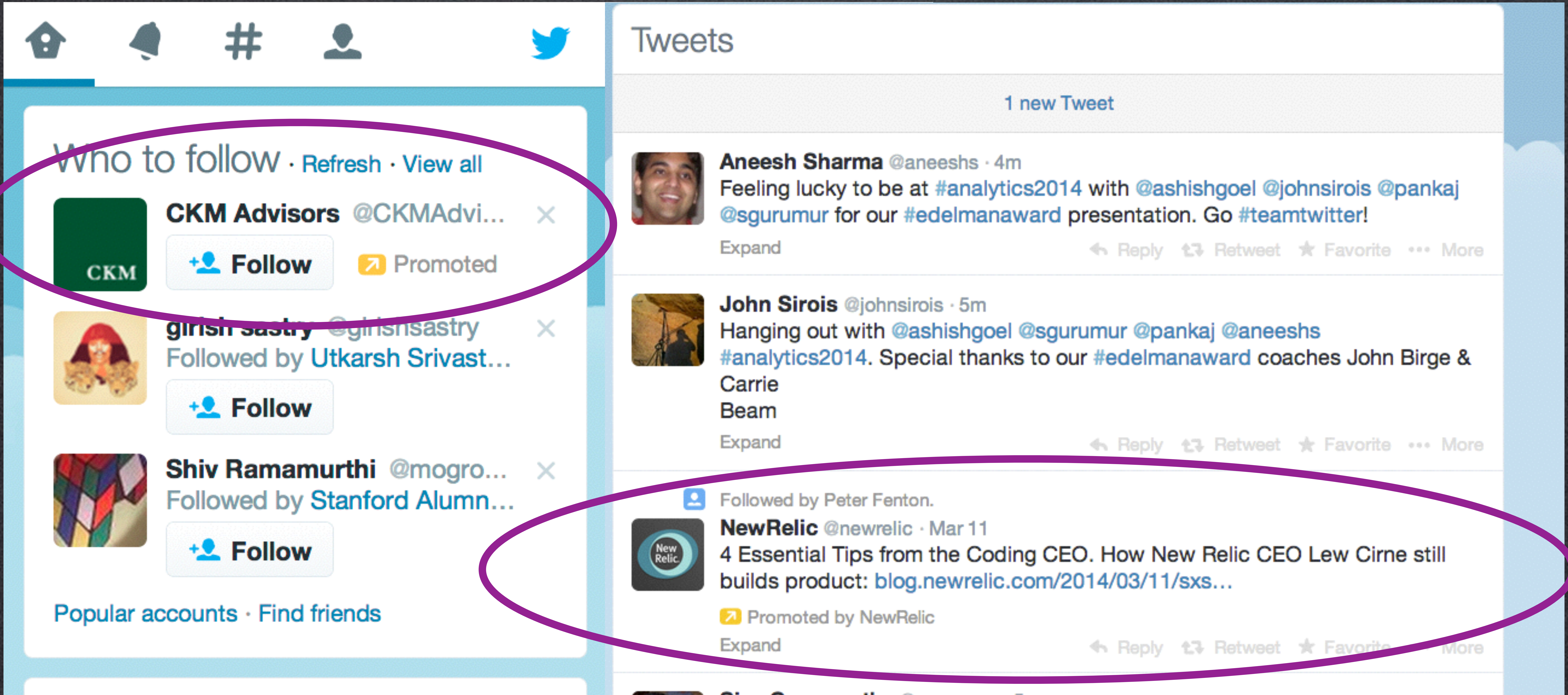
Promoted Tweets and Promoted Accounts



Promoted Tweets and Promoted Accounts



Promoted Tweets and Promoted Accounts



Impact on Revenue

"The Who-To-Follow system was crucial, in a fundamental way, for the Promoted Accounts product, and the Promoted Tweets product also initially used the Who-To-Follow system's targeting"

– Alex Roetter (VP of Engineering, Revenue)

Scientific Questions

1. Fast Incremental PageRank
2. Fast Personalized PageRank

Incremental PageRank

Updates to social graph are made in real-time

- As opposed to a batched crawl process for web search
- Real-time updates to PageRank are important to capture trending events

Goal: Design an algorithm to update PageRank incrementally (i.e. upon an edge arrival)

- t-th edge arrival: Let (u_t, v_t) denote the arriving edge, $d_t(v)$ denote the out-degree of node v, and $\pi_t(v)$ its PageRank

Incremental PageRank via Monte Carlo

Start with $R = O(\log N)$ random walks from every node

At time t, for every random walk through node u_t, re-route it to use the new edge (u_t, v_t) with probability $1/d_t(u_t)$

- Time/number of network-calls for each re-routing: $O(1/\alpha)$

Claim: This faithfully maintains R random walks after arbitrary edge arrivals

Need the graph and the stored random walks in fast distributed memory

Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order

Theorem: # of re-routings per arrival goes to 0

- t-th arrival: # of reroutes = $O(N\,R/(\alpha\,t))$
- Total time over M arrivals = $O((N\,R \log N)/\alpha^2)$
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]

Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order

Theorem: # of r per arrival goes to 0

- t-th arrival: # of N R/(α t))
- Total time over M arrivals = O((N R log N)/α^2)
- Comparable to doing power iteration/Monte Carlo just once!

> Monte Carlo takes time NR/α

[Bahmani, Goel, Chowdhury, VLDB 2010]

Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order

Theorem: # of r … per arrival goes to 0

Monte Carlo takes time NR/α

- t-th arrival: # of … N R/(α t))
- Total time over M arrivals = O((N R log N)/α²)
- Comparable to doing power iteration/Monte Carlo just once!

Only an extra log N/α

[Bahmani, Goel, Chowdhury, VLDB 2010]

Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order

Theorem: # of r … per arrival goes to 0

Power iteration takes time $M\ R/\alpha$

- t-th arrival: # of … $N\ R/(\alpha\ t))$
- Total time over M arrivals = $O((N\ R \log N)/\alpha^2)$
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]

Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order

Theorem: # of r

Power iteration takes time $M\ R/\alpha$

per arrival goes to 0

- t-th arrival: # of N $R/(\alpha\ t)$)
- Total time over M arrivals = $O((N\ R \log N)/\alpha^2)$
- Comparable to doing power iteration/Monte Carlo just once!

$N \log N/\alpha$ vs M

[Bahmani, Goel, Chowdhury, VLDB 2010]

Personalized PageRank

Network-based Personalized Search is not yet mature

Missing technical piece: Efficient algorithms for Personalized PageRank Queries

- Given source s and target t, estimate the Personalized PageRank of t for s with high accuracy, if it is greater than δ

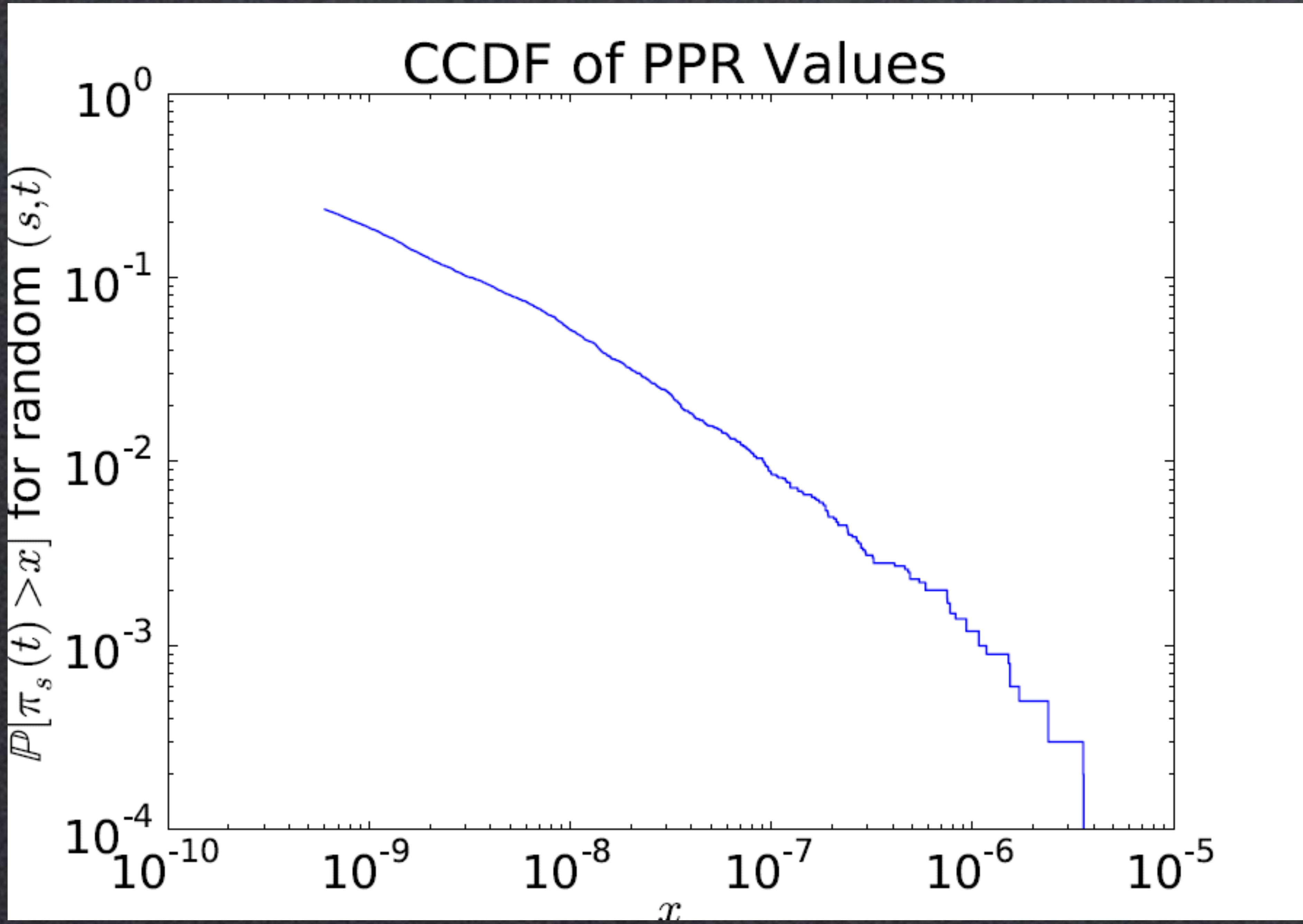
Personalized PageRank

Given a consumer C, perform a random walk on the Follow graph. If the walk is at node v, then the walk:

- Jumps back to node C with probability α
- Follows a random edge out of v with probability 1 – α

The Personalized PageRank of node Y is the weight of Y in the stationary distribution of this random walk

Existing Methods for PPR Queries



Monte Carlo uses time > 1/δ

"Local Update" uses time d/δ

[d = M/N is the average degree]

On Twitter-2010, if $\delta = \frac{4}{n} \approx 10^{-7}$, then

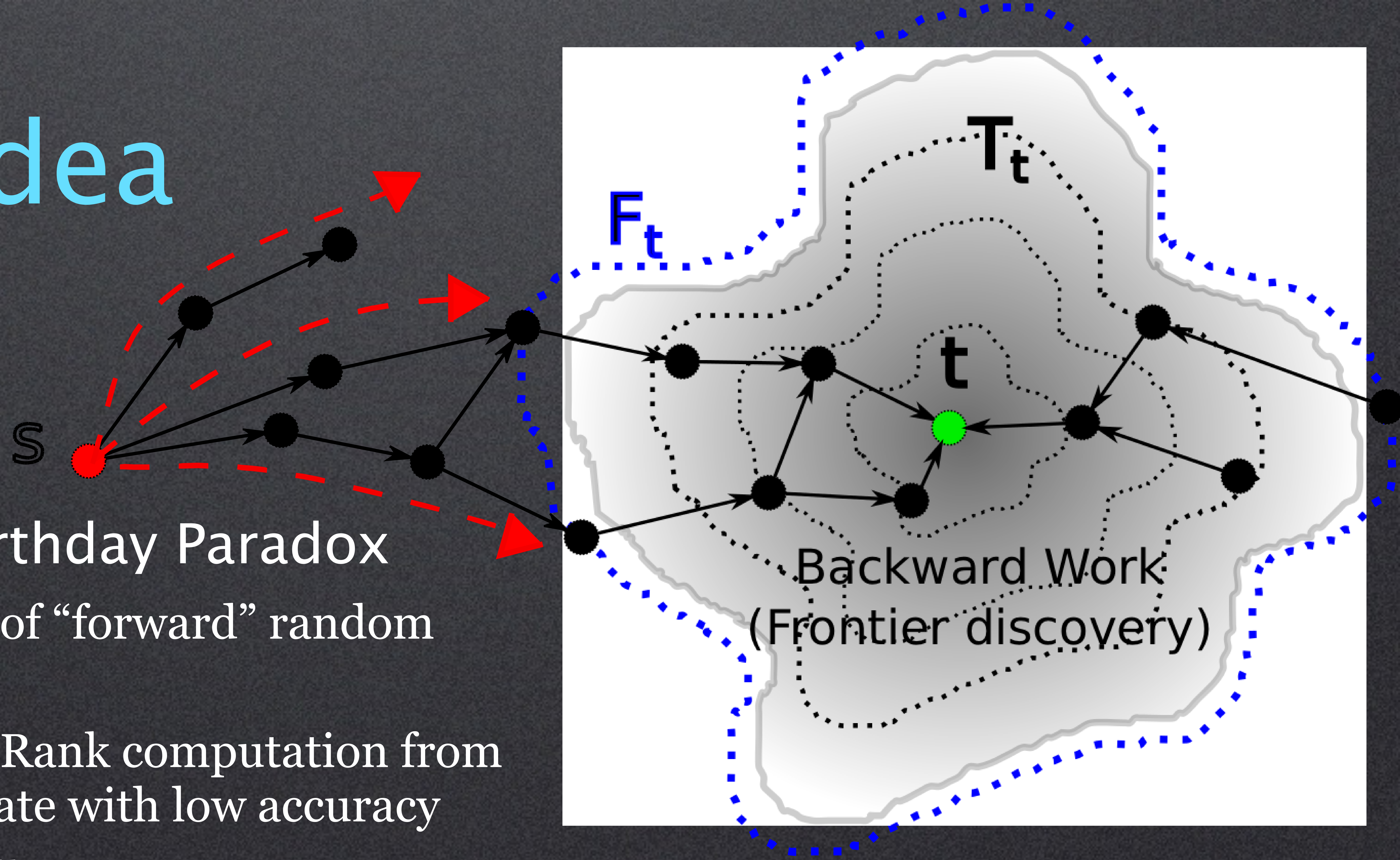
$$\Pr\left[\pi(s,t) > \delta\right] = 1\%$$

FAST PPR

We can answer PPR queries in either

- Average time $\tilde{O}(\sqrt{(d/\delta)})$
- Worst case time $\tilde{O}(\sqrt{(d/\delta)})$ with $\tilde{O}(\sqrt{(d/\delta)})$ storage and pre-processing time per node
- Typical values: $\delta \sim 10^{-8}$, $d \sim 100$; results in a > 100-fold decrease

Basic Idea



Intuition: The Birthday Paradox

- Do small number of “forward” random walks from s
- Do “reverse” PageRank computation from t using Local Update with low accuracy
- Use number of collisions as an estimator
- Need to “catch” a collision just before it happens

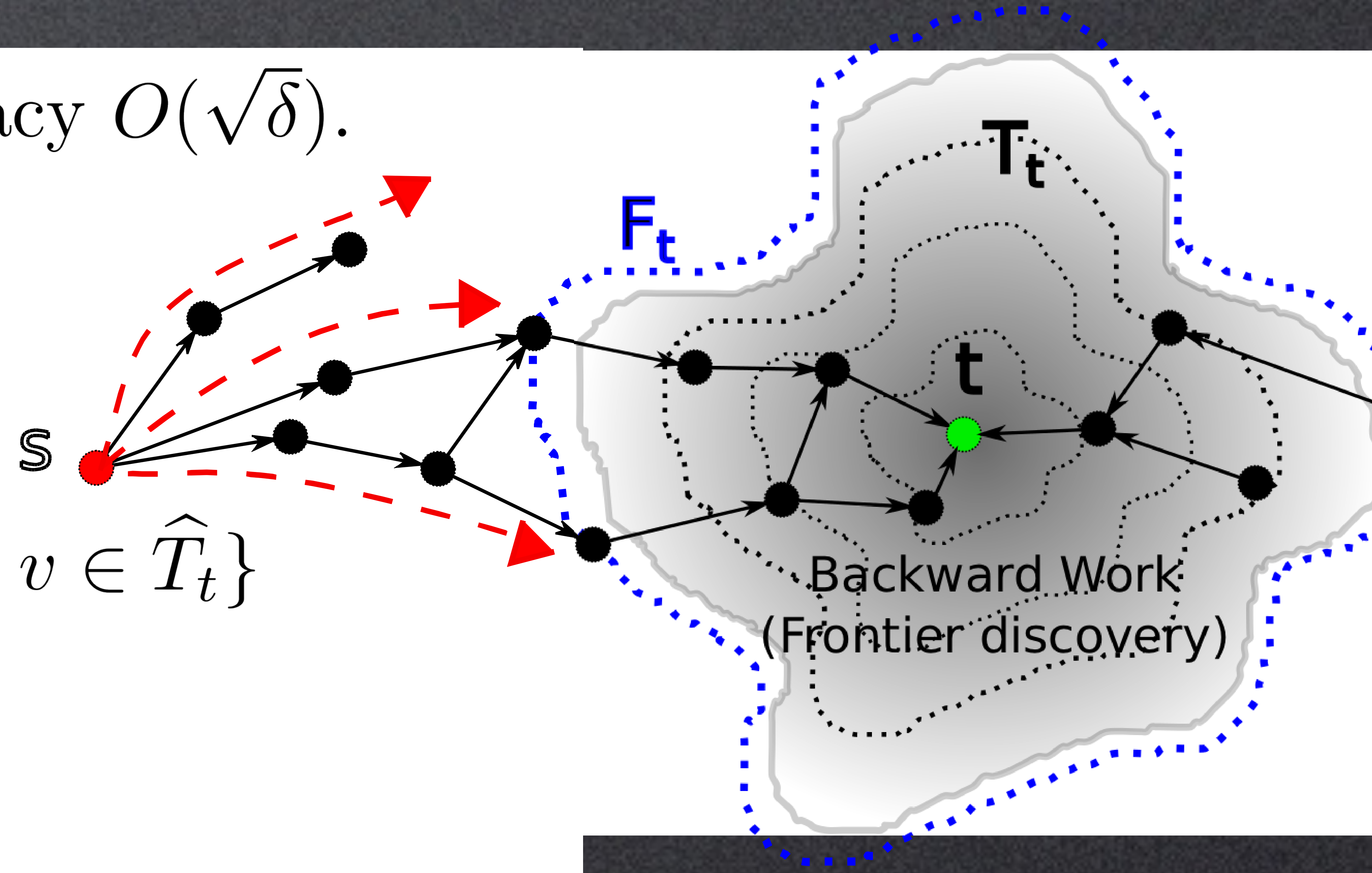
Simple Version of FAST PPR

1. Use Local Update to compute estimates $\hat{\pi}(v,t)$ to accuracy $O(\sqrt{\delta})$.

2. Define

$$\text{Target Set } \widehat{T_t} = \{v \in V : \hat{\pi}(v,t) > \sqrt{\delta}\}$$

$$\text{Frontier } \widehat{F_t} = \{u \in V \backslash \widehat{T_t} : (u,v) \in E \text{ for some } v \in \widehat{T_t}\}$$



3. Take $O\left(\frac{\log(n)}{\sqrt{\delta}}\right)$ Random Walks $\{W_i\}$, terminating each early if it hits $\widehat{F_t}$. Define

$$X_i = \begin{cases} \hat{\pi}(u,t), & W_i \text{ hits } u \in \widehat{F_t} \\ 0, & W_i \text{ does not hit } \widehat{F_t} \end{cases}$$

4. Return empirical mean$\{X_i\}$.

Running Time for Simple Version

For a uniformly random target node t, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}}\left(\bar{d} + \log(n)\right)\right).$$

Running Time for Simple Version

For a uniformly random target node t, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}}\left(\bar{d} + \log(n)\right)\right).$$

Reverse work
(Local Update)

Running Time for Simple Version

For a uniformly random target node t, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}}\left(\bar{d} + \log(n)\right)\right).$$

Reverse work
(Local Update)

Forward work
(Monte Carlo)

Running Time for Simple Version

For a uniformly random target node t, the average per-query running time is

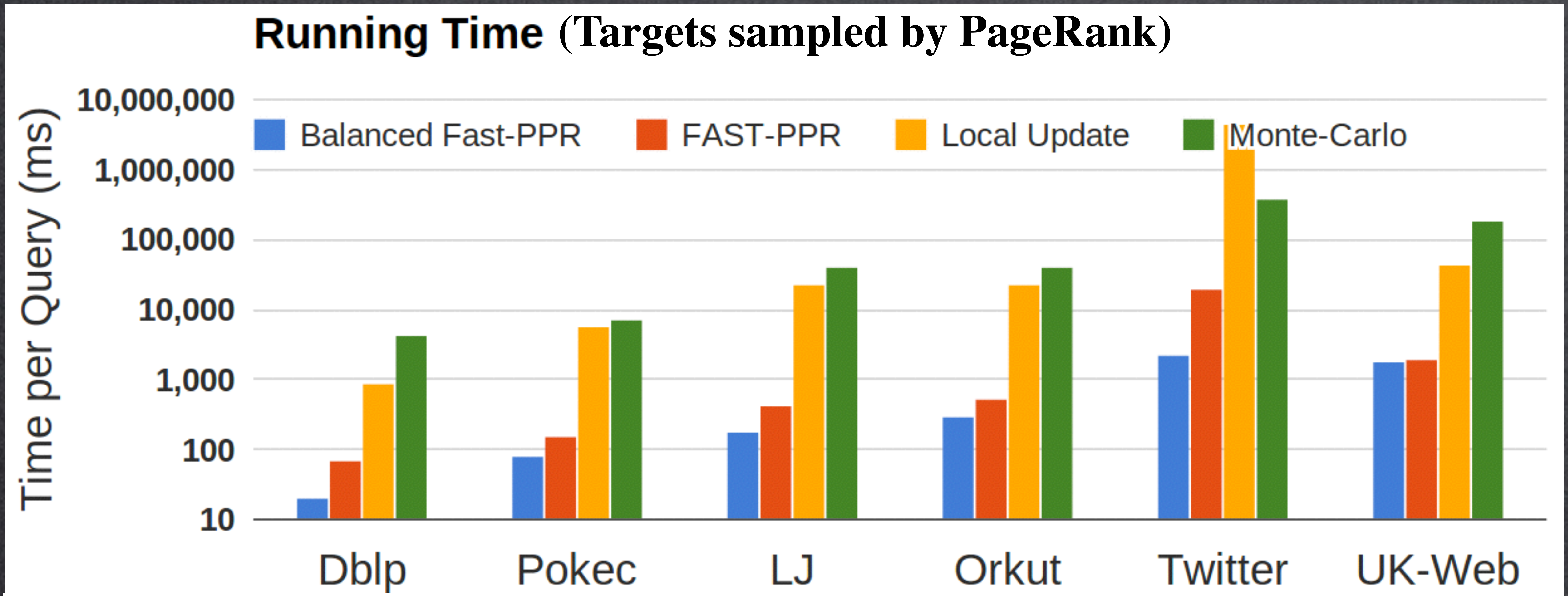
$$O\left(\frac{1}{\sqrt{\delta}}\left(\bar{d} + \log(n)\right)\right).$$

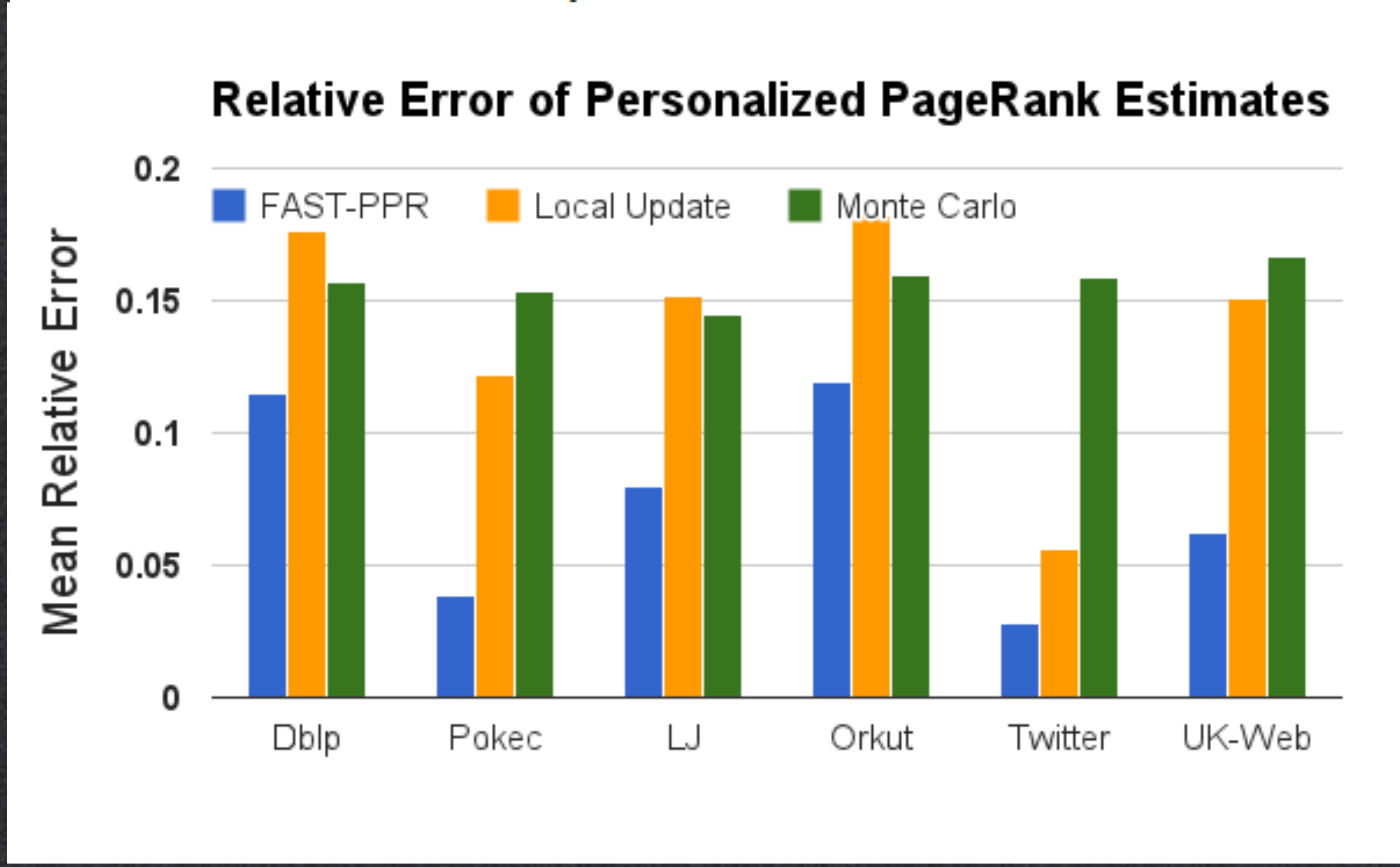
Reverse work (Local Update)

Forward work (Monte Carlo)

We get final running time of $\tilde{O}(\sqrt{(d/\delta)})$ by using different accuracies in forward and reverse computation

We use $\tilde{O}(\sqrt{(d/\delta)})$ pre-processing/space to go from average to worst case running time





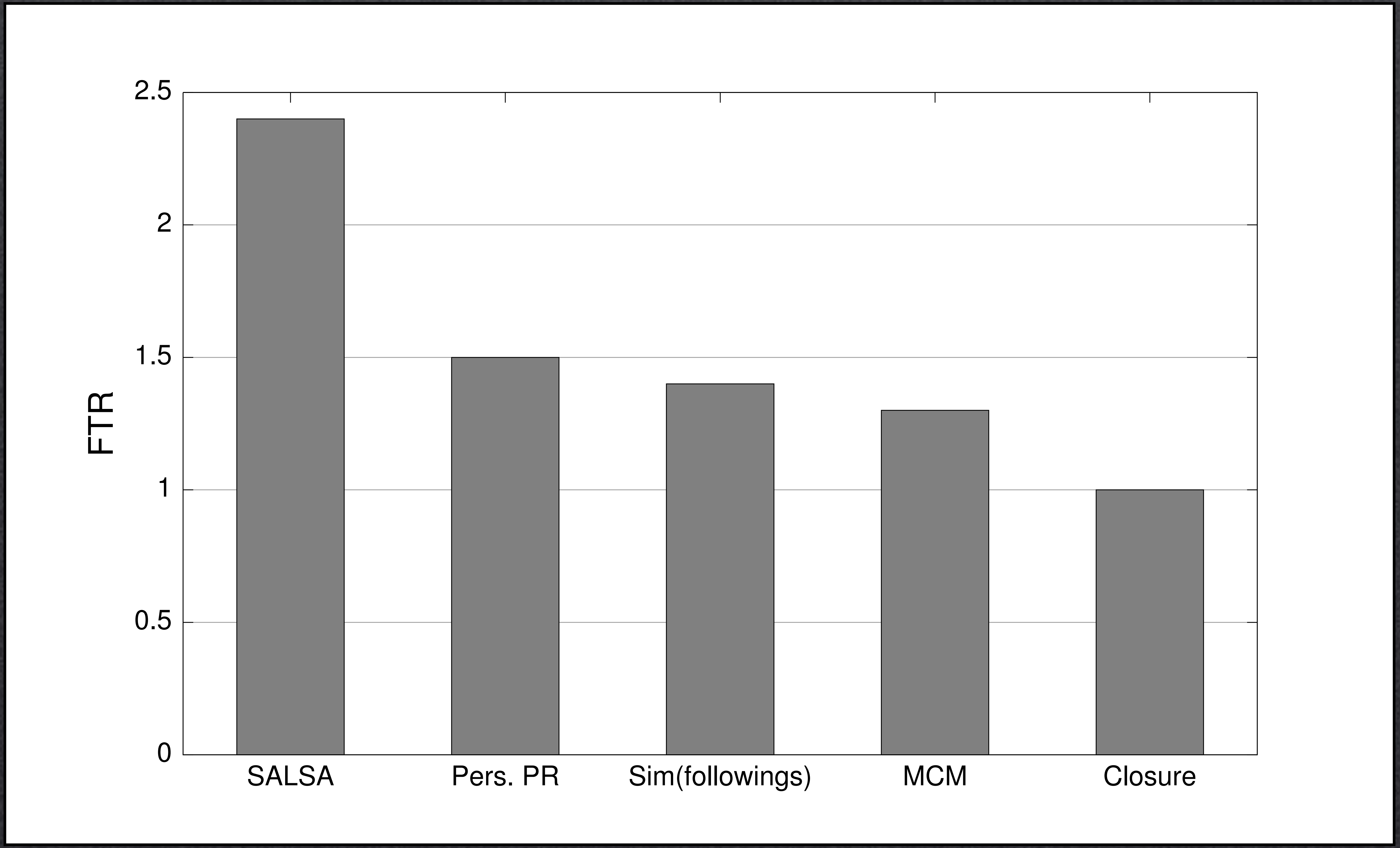
Experiments

- Admits Distributed Implementation
- Works when source is a set of nodes
- Lower bound of $1/\sqrt{\delta}$
- Open problem: do we need the $\sqrt{d}$?

[Lofgren, Banerjee, Goel, Seshadhri, KDD, 2014]

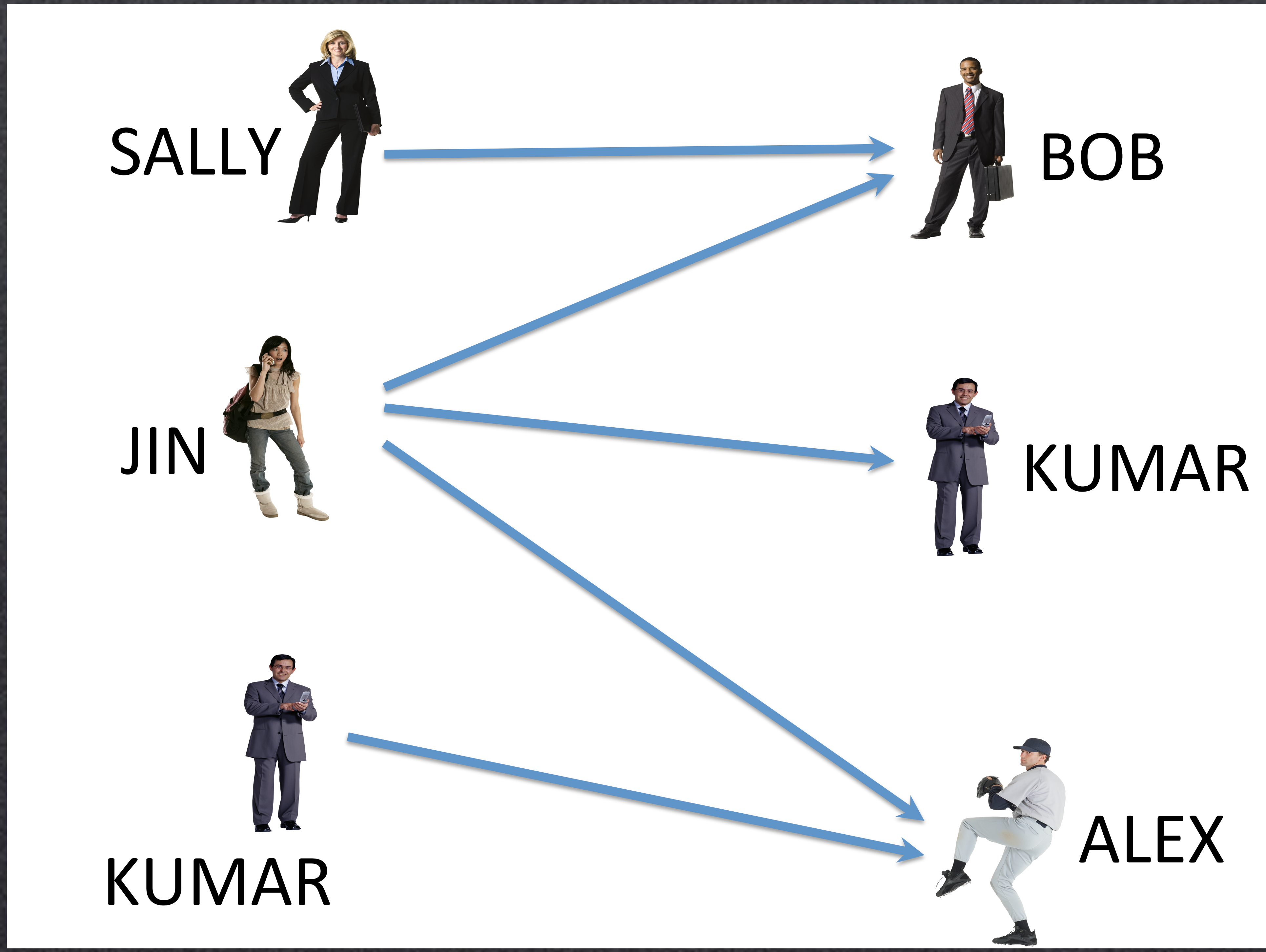
BACKUP SLIDES

Ongoing evaluation



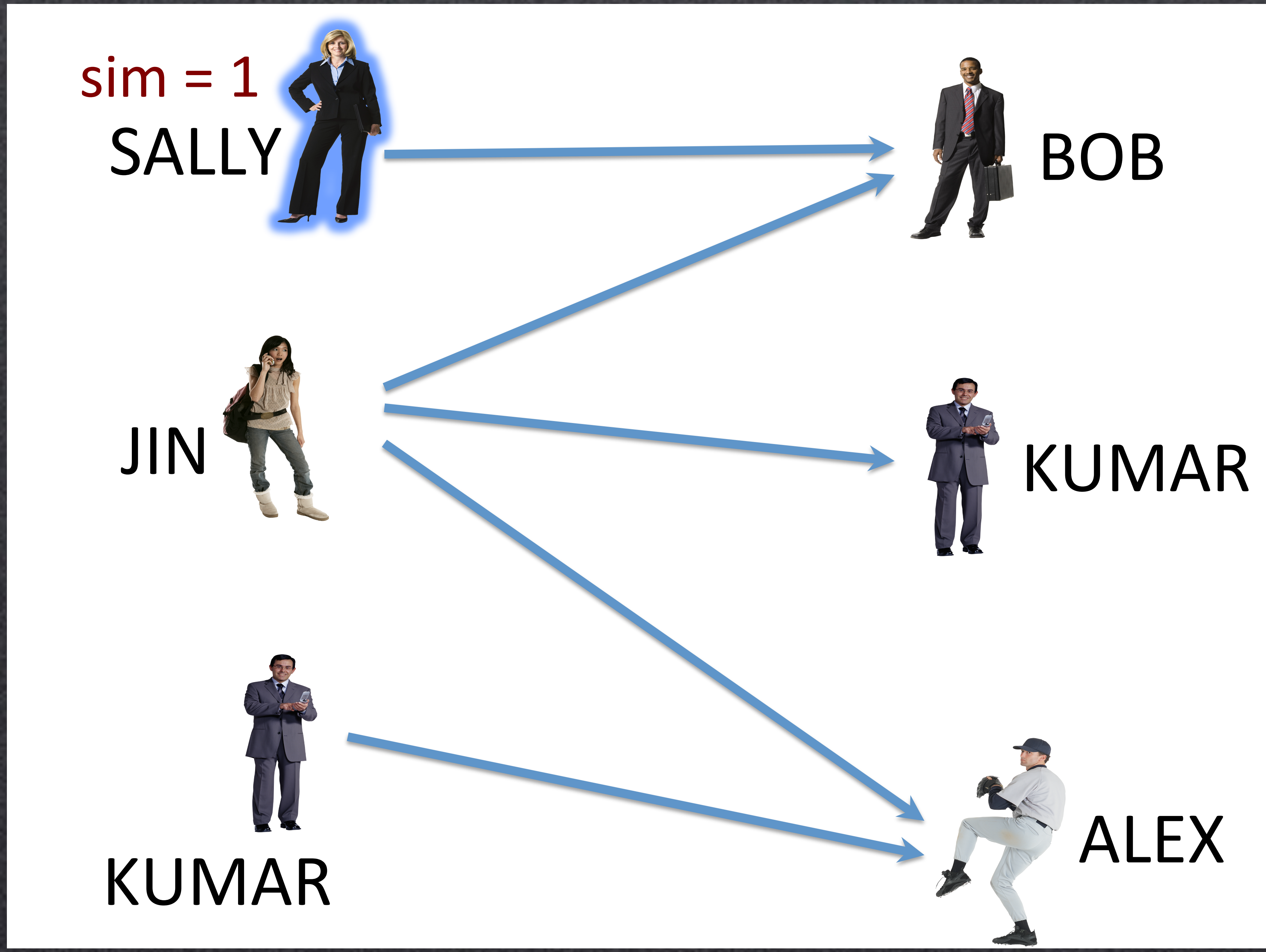
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)



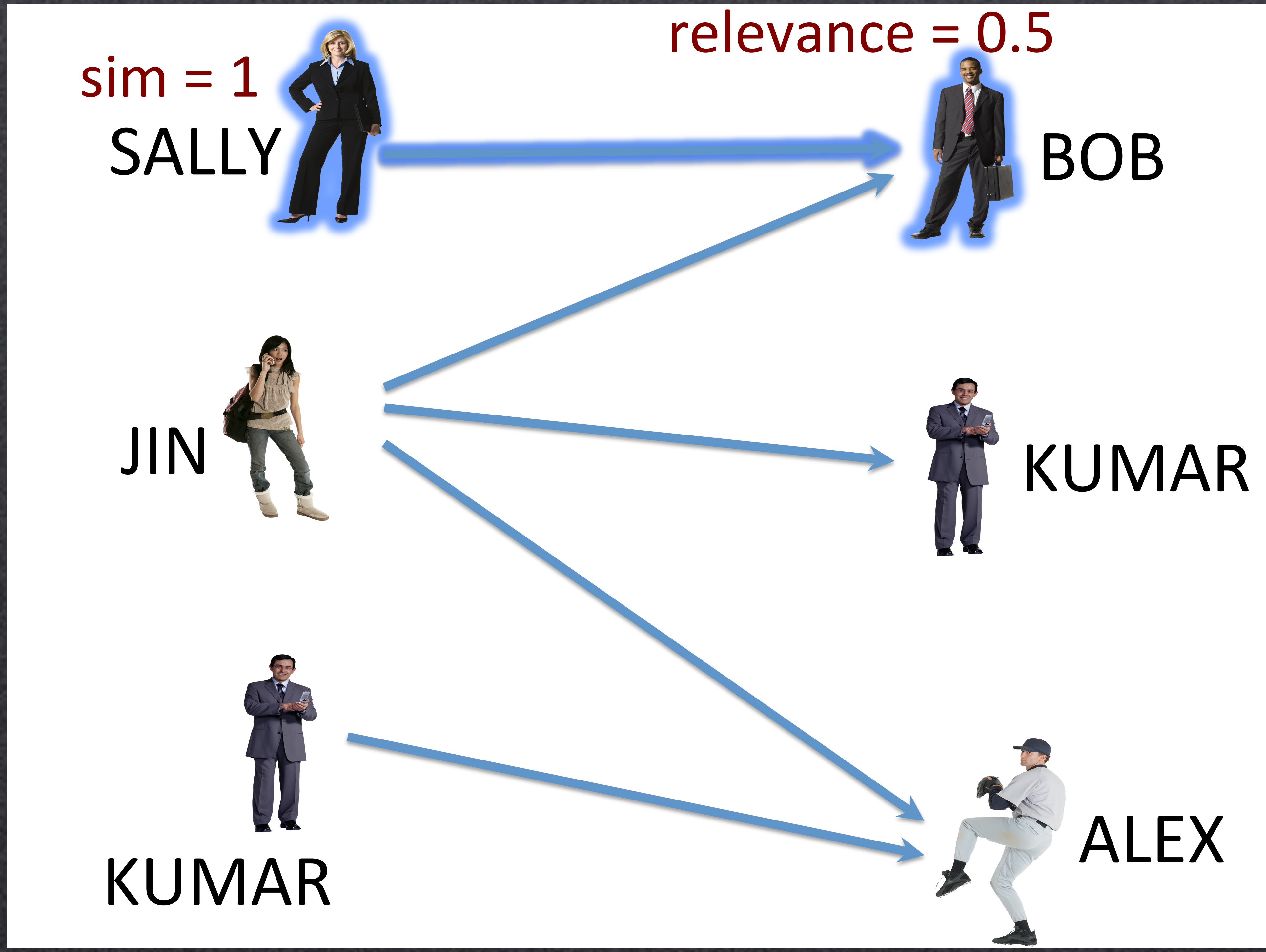
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)



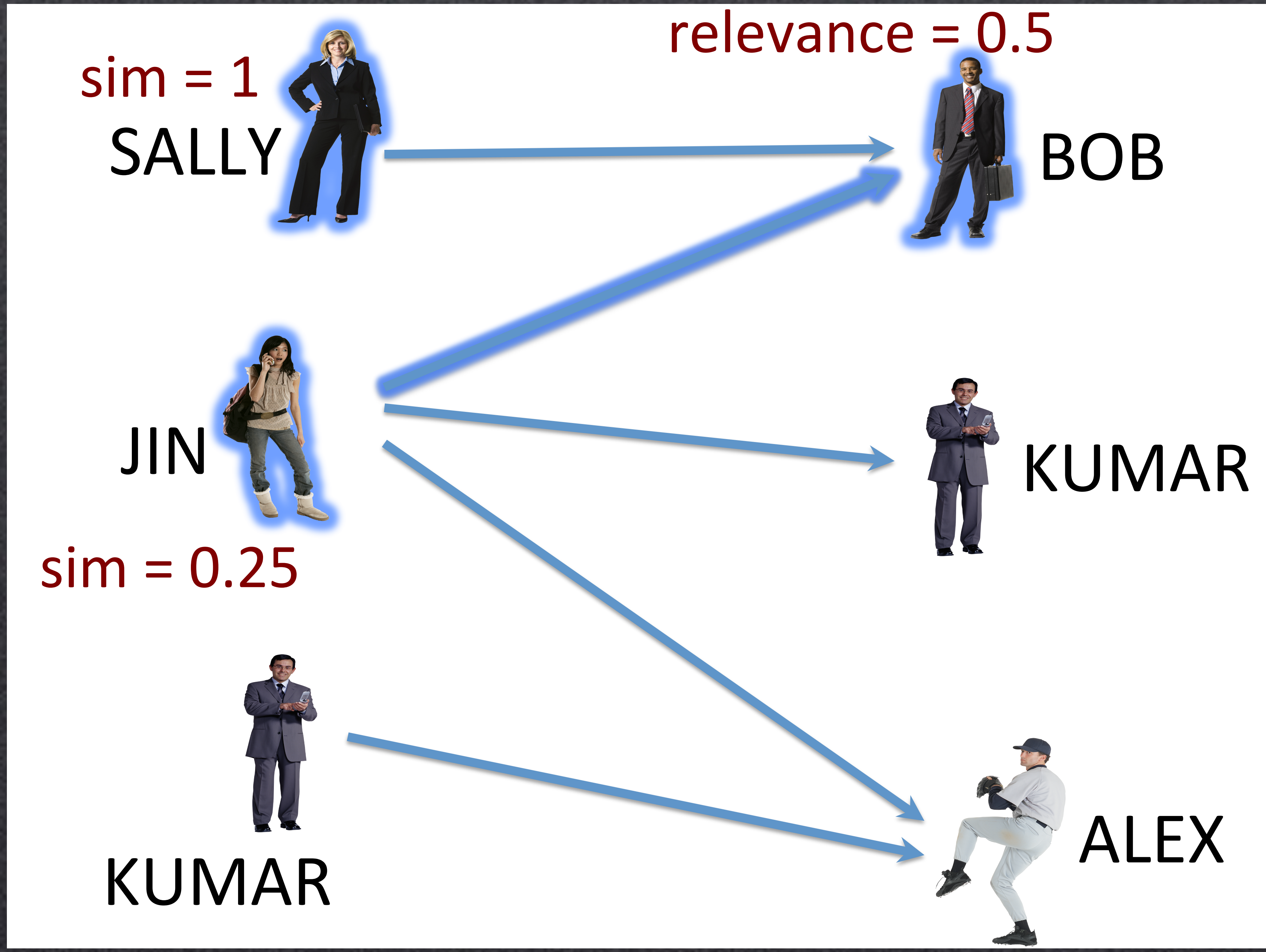
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)



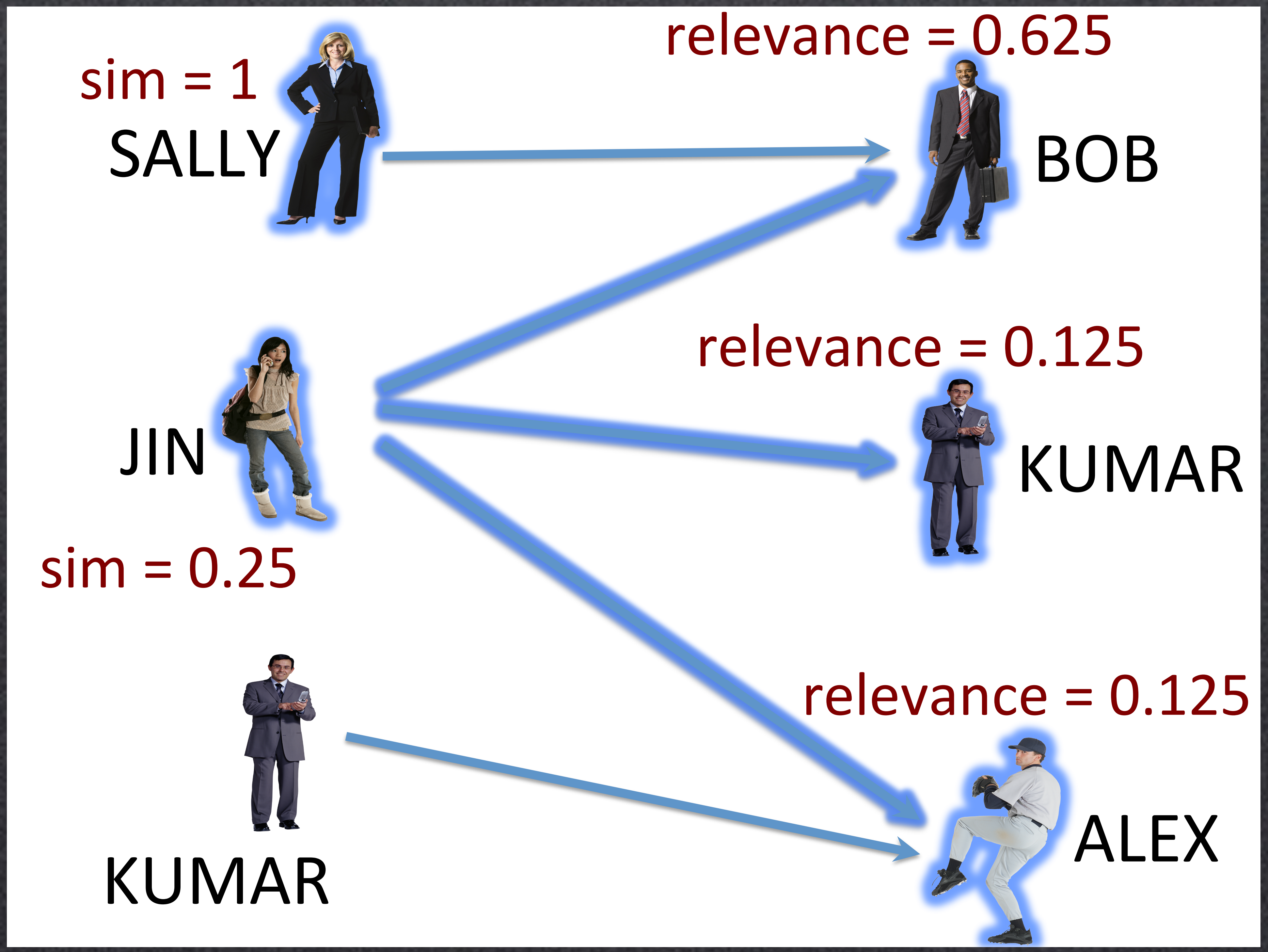
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)



Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)



Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)