Data-Intensive Distributed Computing

CS 451/651 431/631 (Winter 2018)

Part 8: Analyzing Graphs, Redux (2/2)

March 22, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018w/

Theme for Today:

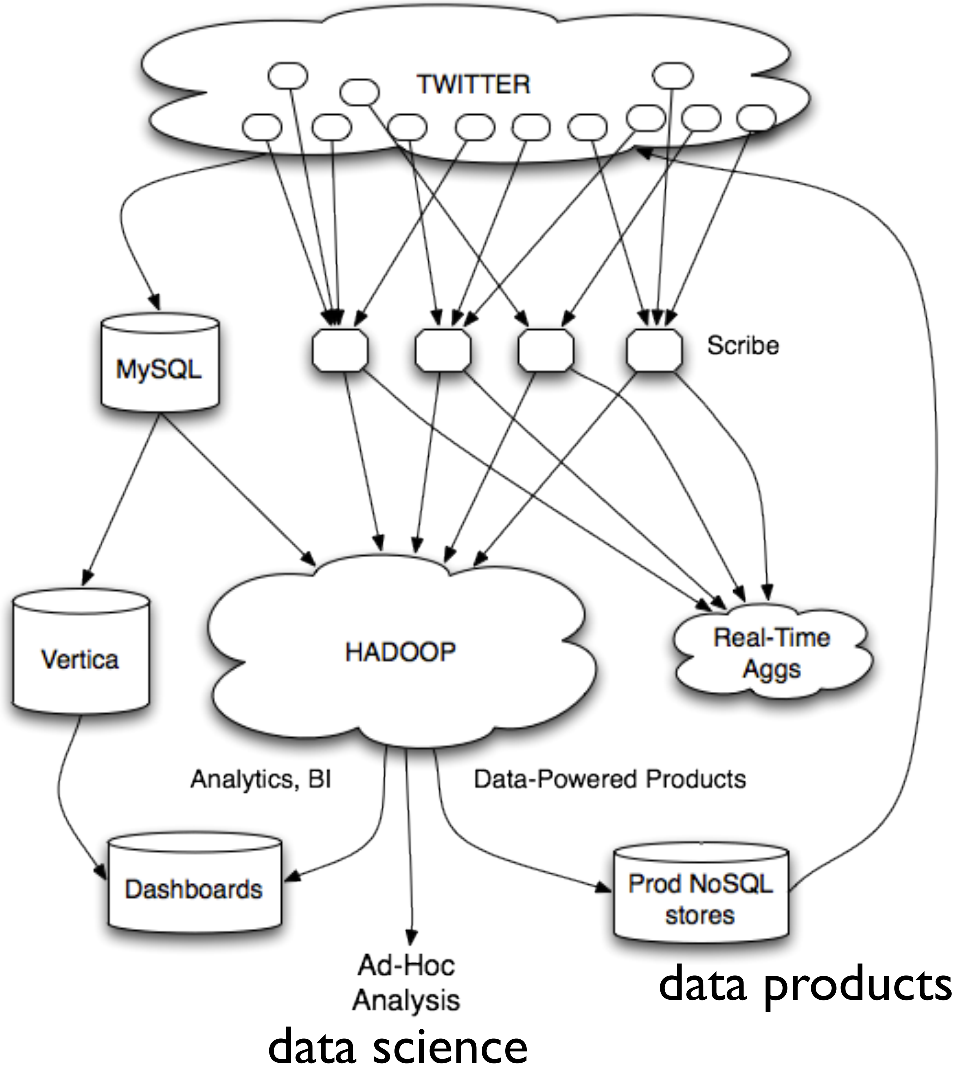
How things work in the real world

(forget everything I told you…)

Source: Wikipedia (All Souls College, Oxford)

Source: Wikipedia (Factory)

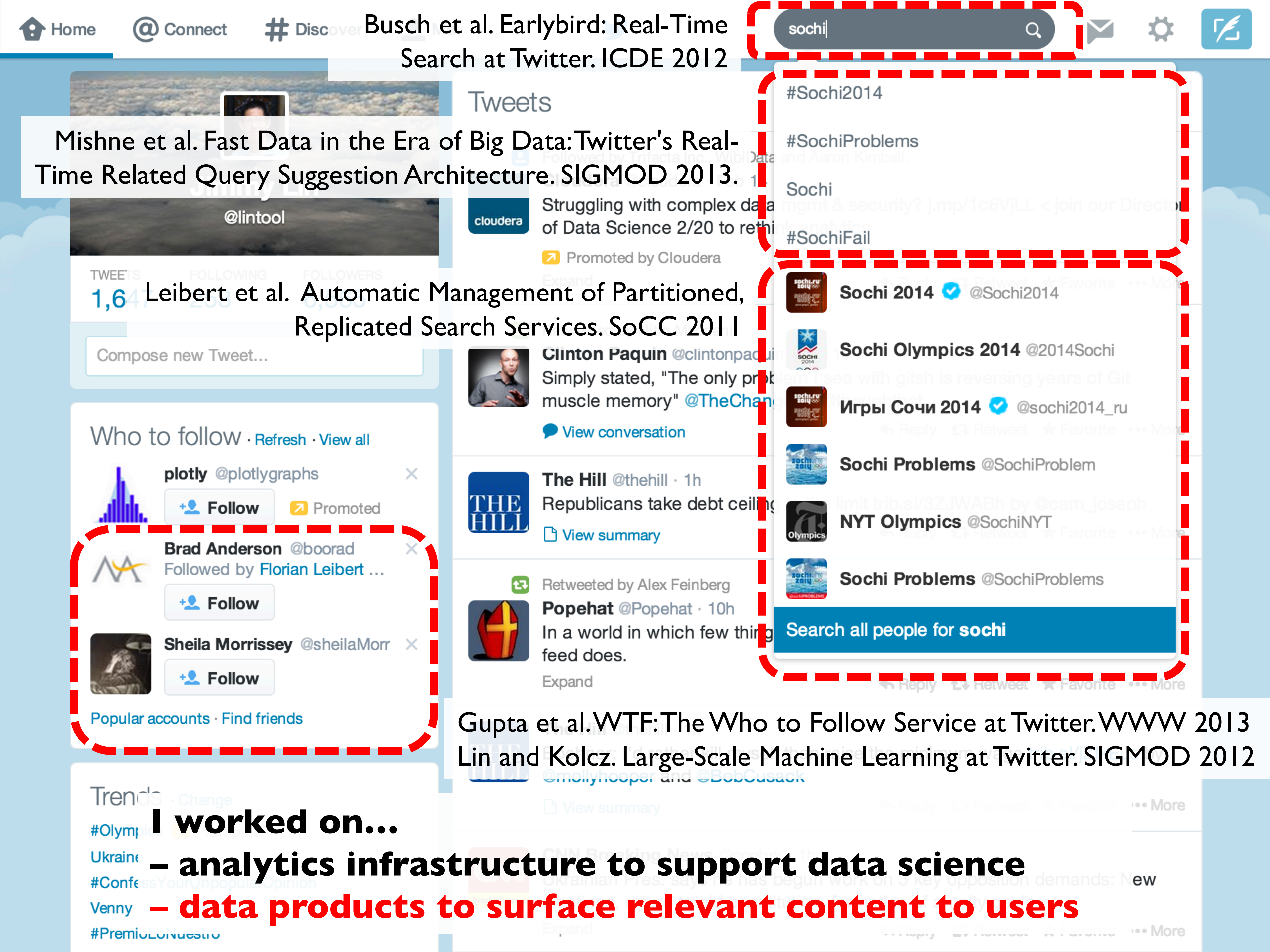
What exactly did I do at Twitter?



I worked on…

– analytics infrastructure to support data science

– data products to surface relevant content to users



Busch et al. Earlybird: Real-Time Search at Twitter. ICDE 2012

Mishne et al. Fast Data in the Era of Big Data: Twitter's Real-Time Related Query Suggestion Architecture. SIGMOD 2013.

Leibert et al. Automatic Management of Partitioned, Replicated Search Services. SoCC 2011

Gupta et al. WTF: The Who to Follow Service at Twitter. WWW 2013

Lin and Kolcz. Large-Scale Machine Learning at Twitter. SIGMOD 2012

I worked on…

– analytics infrastructure to support data science

– data products to surface relevant content to users


Source: https://www.flickr.com/photos/bongtongol/3491316758/

circa ~2010

~150 people total

~60 Hadoop nodes

~6 people use analytics stack daily

circa ~2012

~1400 people total

10s of Ks of Hadoop nodes, multiple DCs

10s of PBs total Hadoop DW capacity

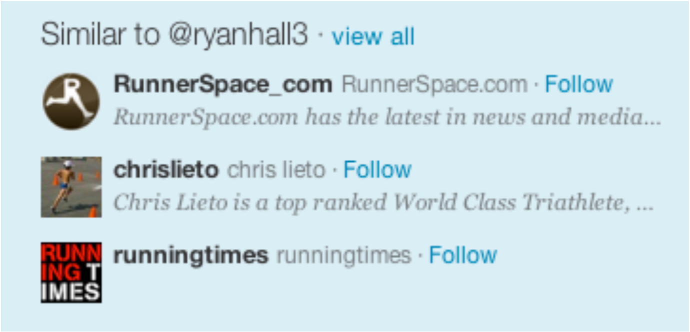
~100 TB ingest daily

dozens of teams use Hadoop daily

10s of Ks of Hadoop jobs daily

WTF

(whom to follow)
(who to follow)



#numbers

(Second half of 2012)

~175 million active users

~20 billion edges

42% edges bidirectional

Avg shortest path length: 4.05

40% as many unfollows as follows daily

WTF responsible for ~1/8 of the edges

Myers, Sharma, Gupta, Lin. Information Network or Social Network? The Structure of the Twitter Follow Graph. WWW 2014.

Graphs are core to Twitter

"structural" property

Twitter "verbs" = interactions

Graph-based recommendation systems

Why? Increase engagement!

The Journey

From the static follower graph for account recommendations…
… to the real-time interaction graph for content recommendations

In Four Acts...

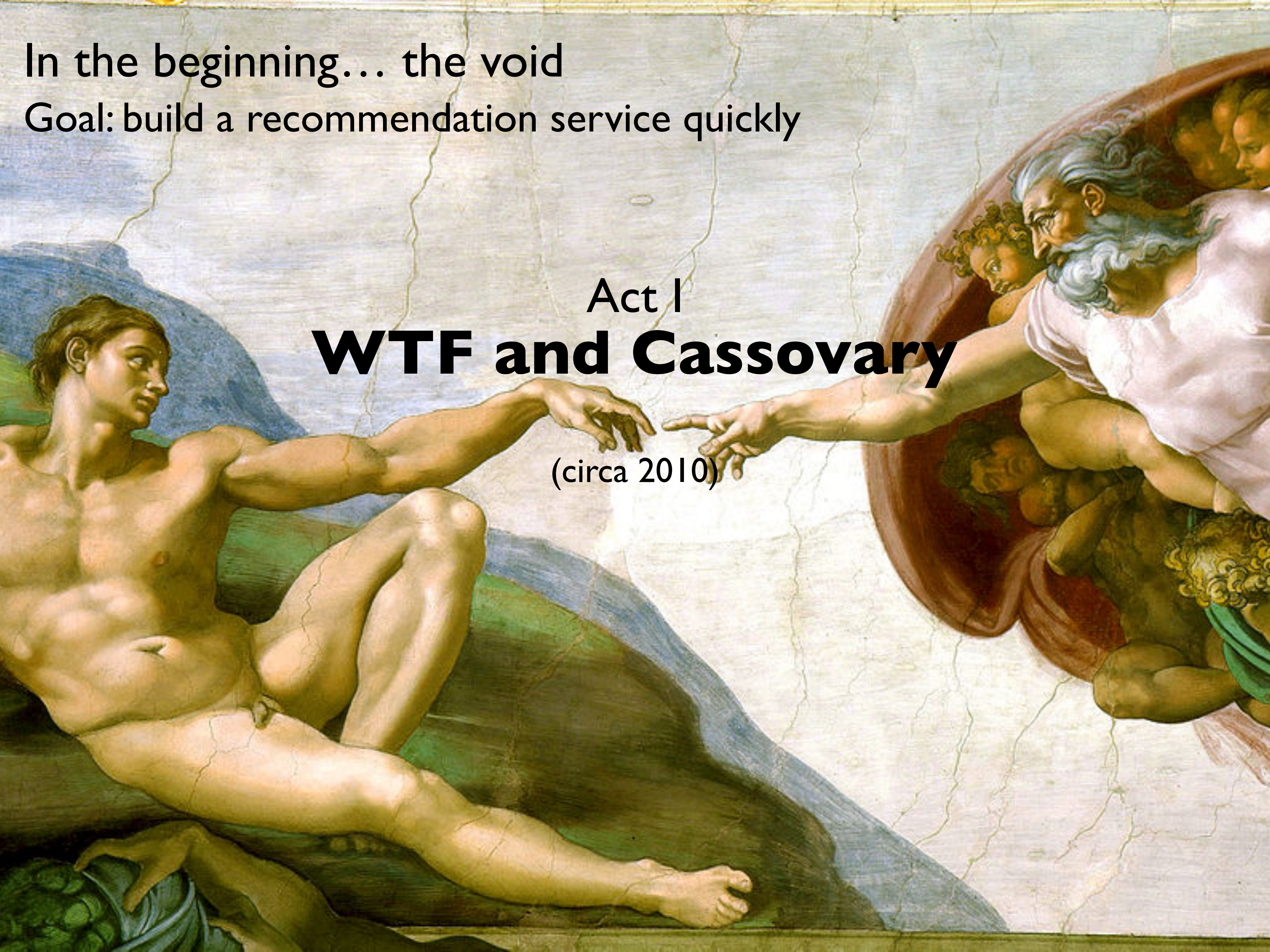
Source: flickr (https://www.flickr.com/photos/39414578@N03/16042029002)

In the beginning… the void

Act I

WTF and Cassovary

(circa 2010)



In the beginning… the void

Goal: build a recommendation service quickly

Act I

WTF and Cassovary

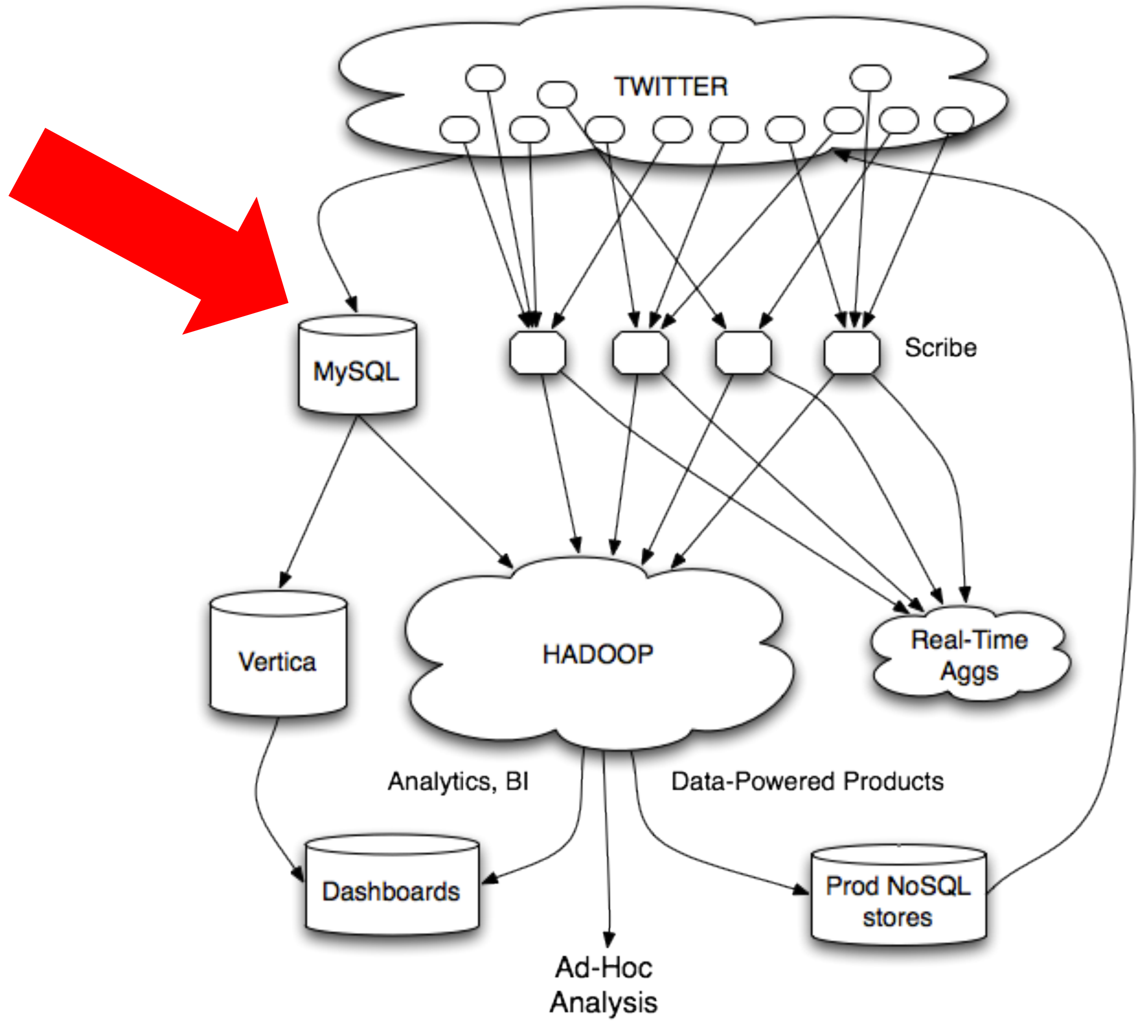
(circa 2010)

flockDB

(graph database)

Simple graph operations
Set intersection operations

Not appropriate for graph algorithms!


TWITTER
MySQL
Scribe
Vertica
HADOOP
Real-Time Aggs
Analytics, BI
Data-Powered Products
Dashboards
Prod NoSQL stores
Ad-Hoc Analysis

Okay, let’s use MapReduce!

But MapReduce sucks for graphs!

What about…?

HaLoop (VLDB 2010)
Twister (MapReduce Workshop 2010)
Pregel/Giraph (SIGMOD 2010)
Graphlab (UAI 2010)
PrIter (SoCC 2011)
Datalog on Hyracks (Tech report, 2012)
Spark/GraphX (NSDI 2012, arXiv 2014)
PowerGraph (OSDI 2012)
GRACE (CIDR 2013)
Mizan (EuroSys 2013)
…

MapReduce sucks for graph algorithms…

Let's build our own system!

Key design decision:

Keep entire graph in memory… on a single machine!

Nuts!

Why?

Because we can!

Graph partitioning is hard… so don't do it

Simple architecture

Right choice at the time!

Source: Wikipedia (Pistachio)

The runway argument

Source: Wikipedia (Heathrow)

Suppose: 10×10^9 edges
(src, dest) pairs: ~80 GB

18 × 8 GB DIMMS = 144 GB
18 × 16 GB DIMMS = 288 GB
12 × 16 GB DIMMS = 192 GB
12 × 32 GB DIMMS = 384 GB

Cassovary

In-memory graph engine

- Implemented in Scala
- Compact in-memory representations
- But no compression
- Avoid JVM object overhead!
- Open-source

Source: Wikipedia (Cassowary)

PageRank

"Semi-streaming" algorithm

Keep vertex state in memory, stream over edges

Each pass = one PageRank iteration

Bottlenecked by memory bandwidth

Convergence?

Don't run from scratch… use previous values

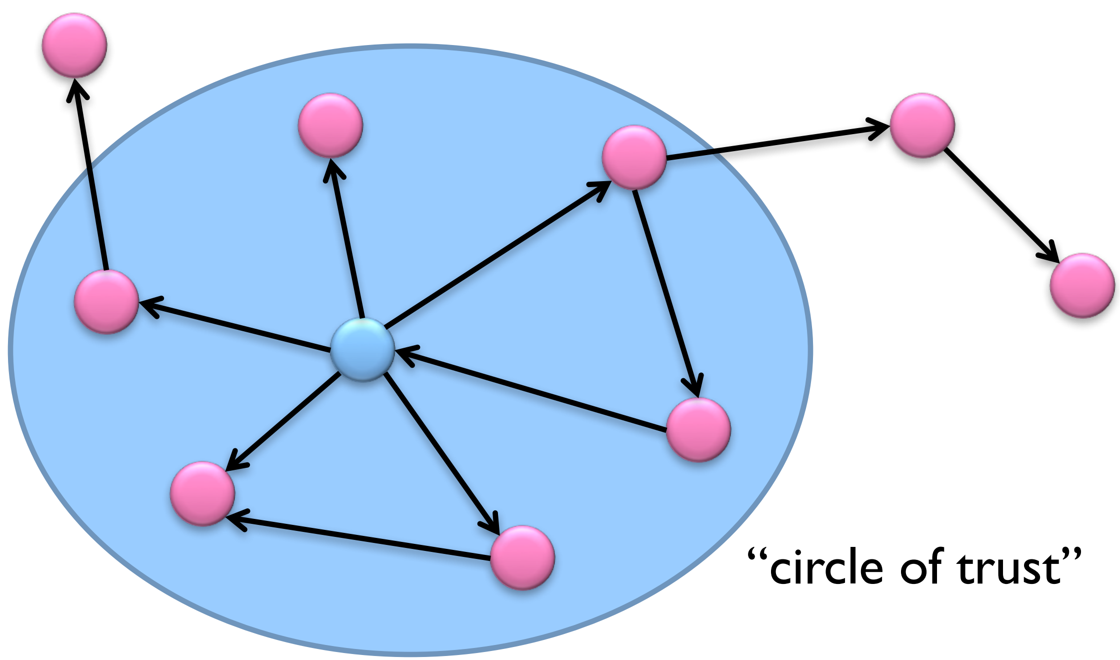
A few passes are sufficient

“Circle of Trust”

Ordered set of important neighbors for a user

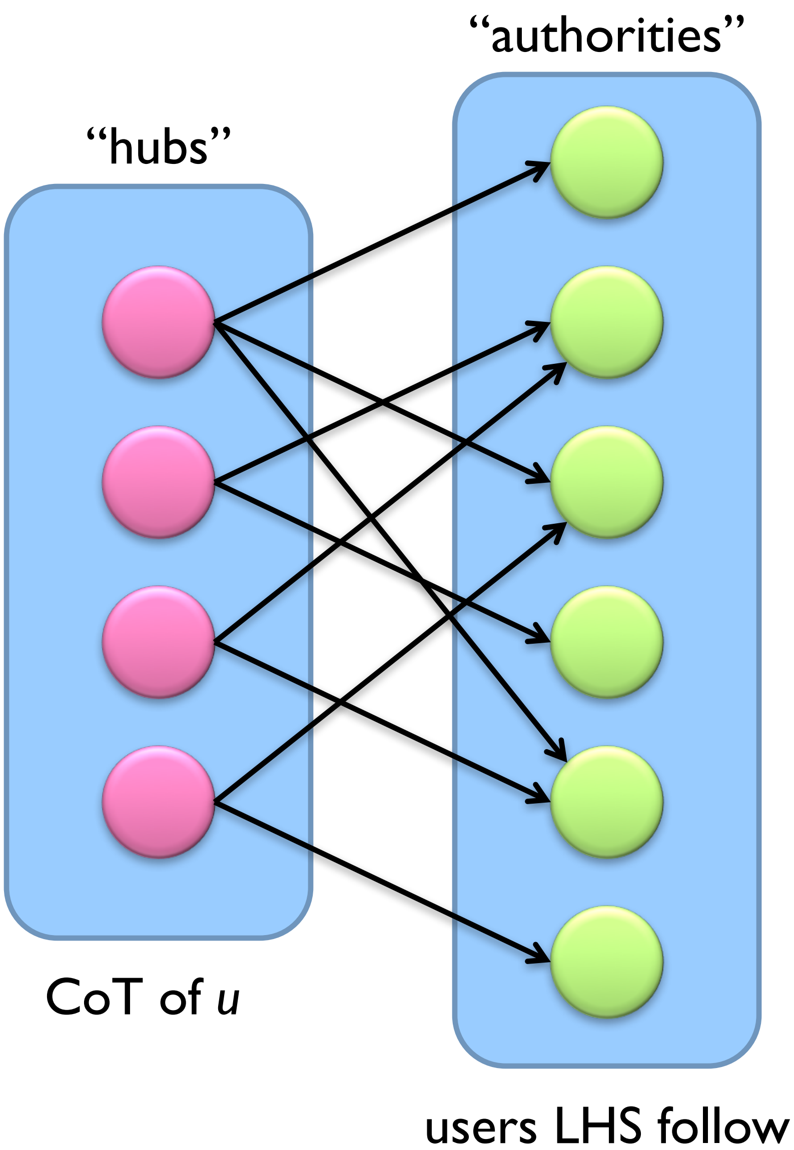
Result of egocentric random walk: Personalized PageRank!

Computed online based on various input parameters



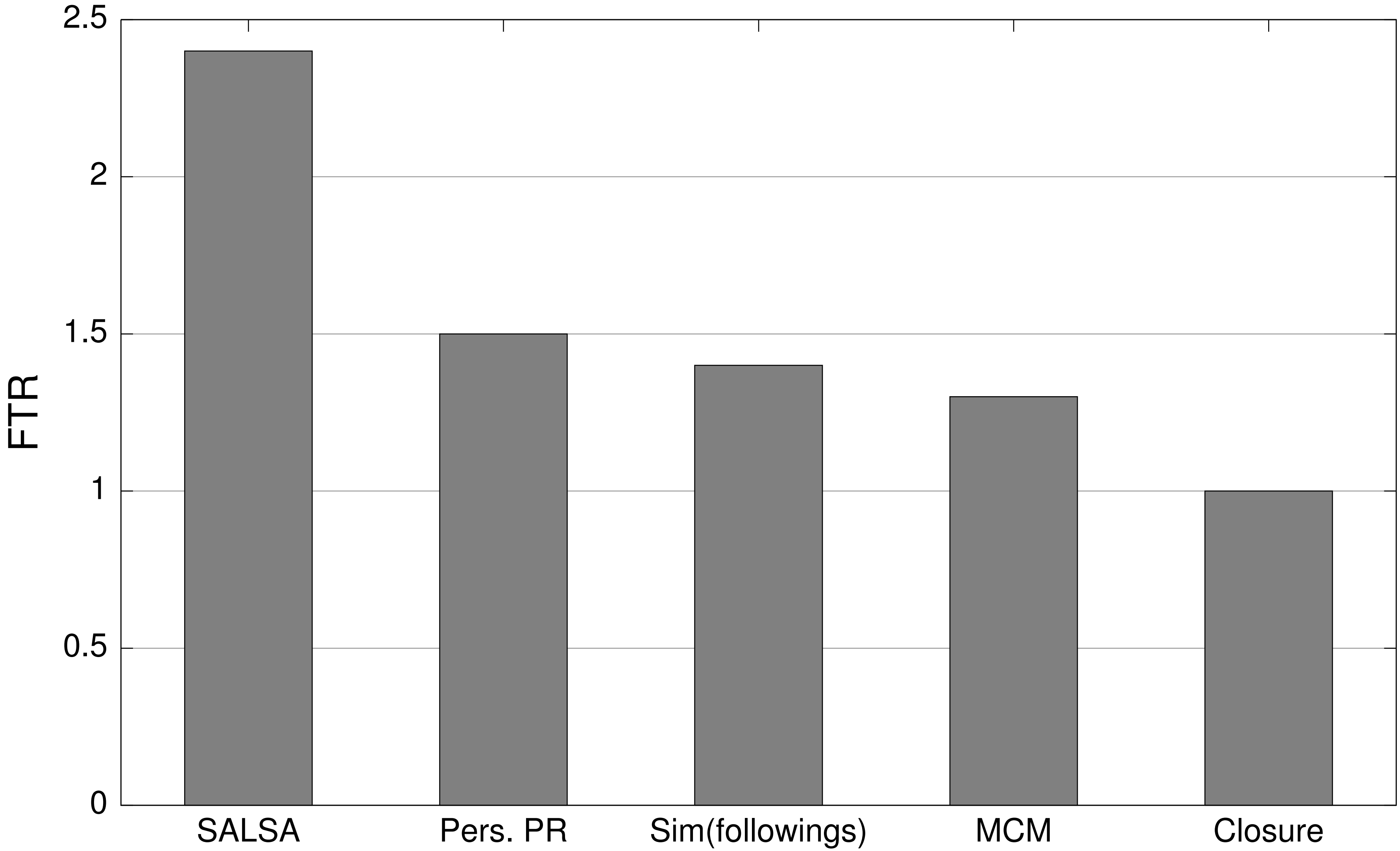
One of the features used in search

SALSA for Recommendations

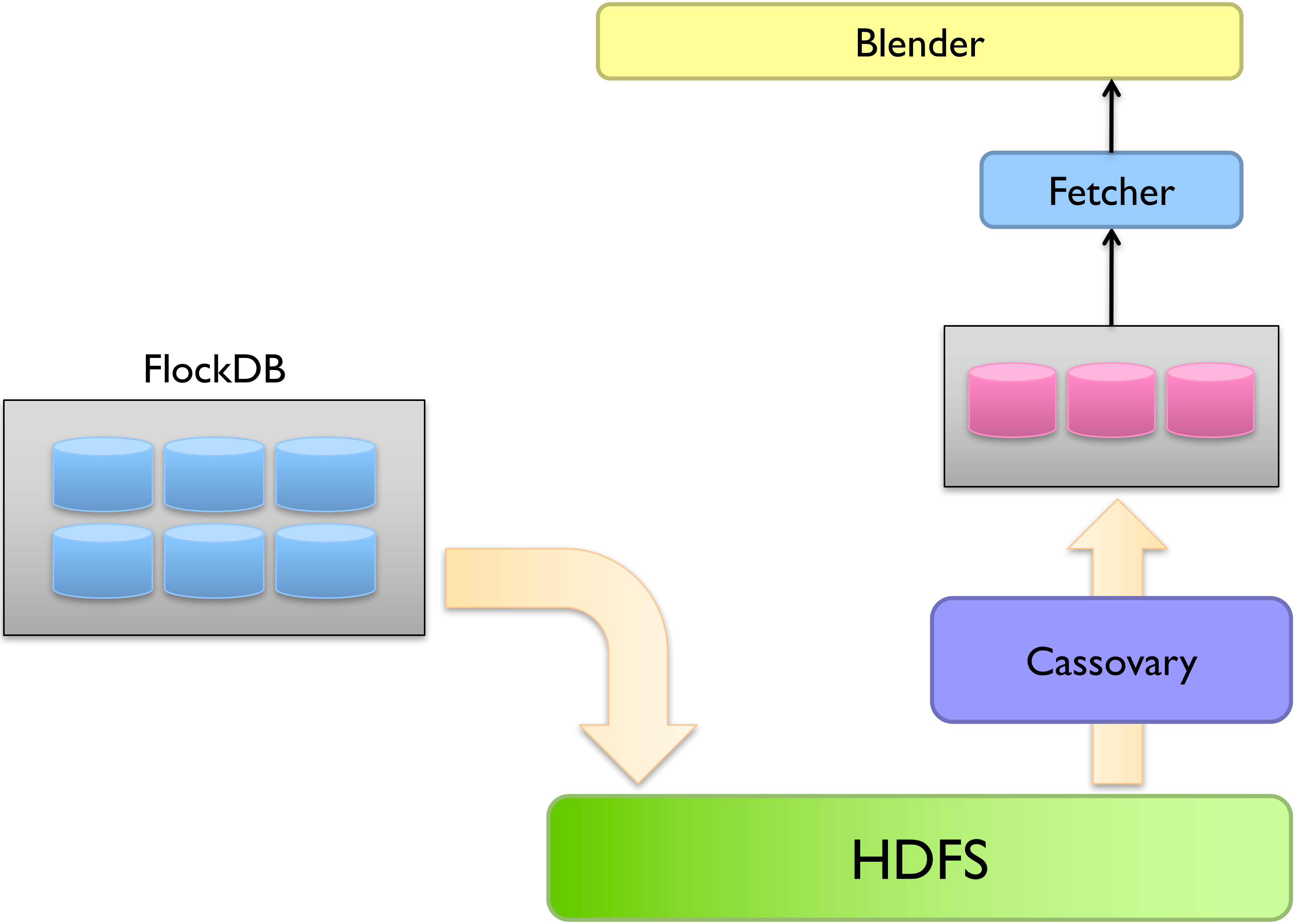


hubs scores:
similarity scores to *u*

authority scores:
recommendation scores for *u*

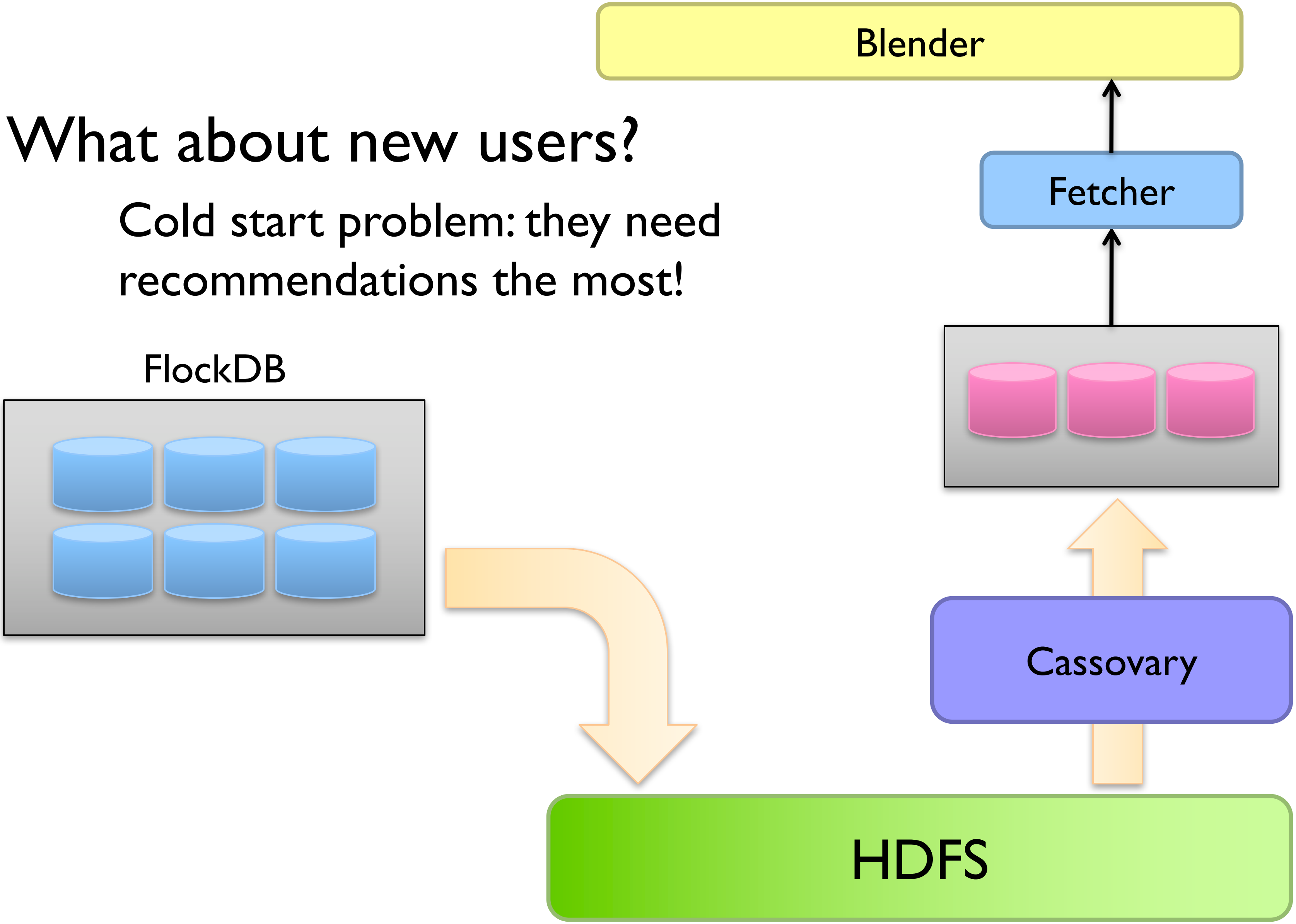


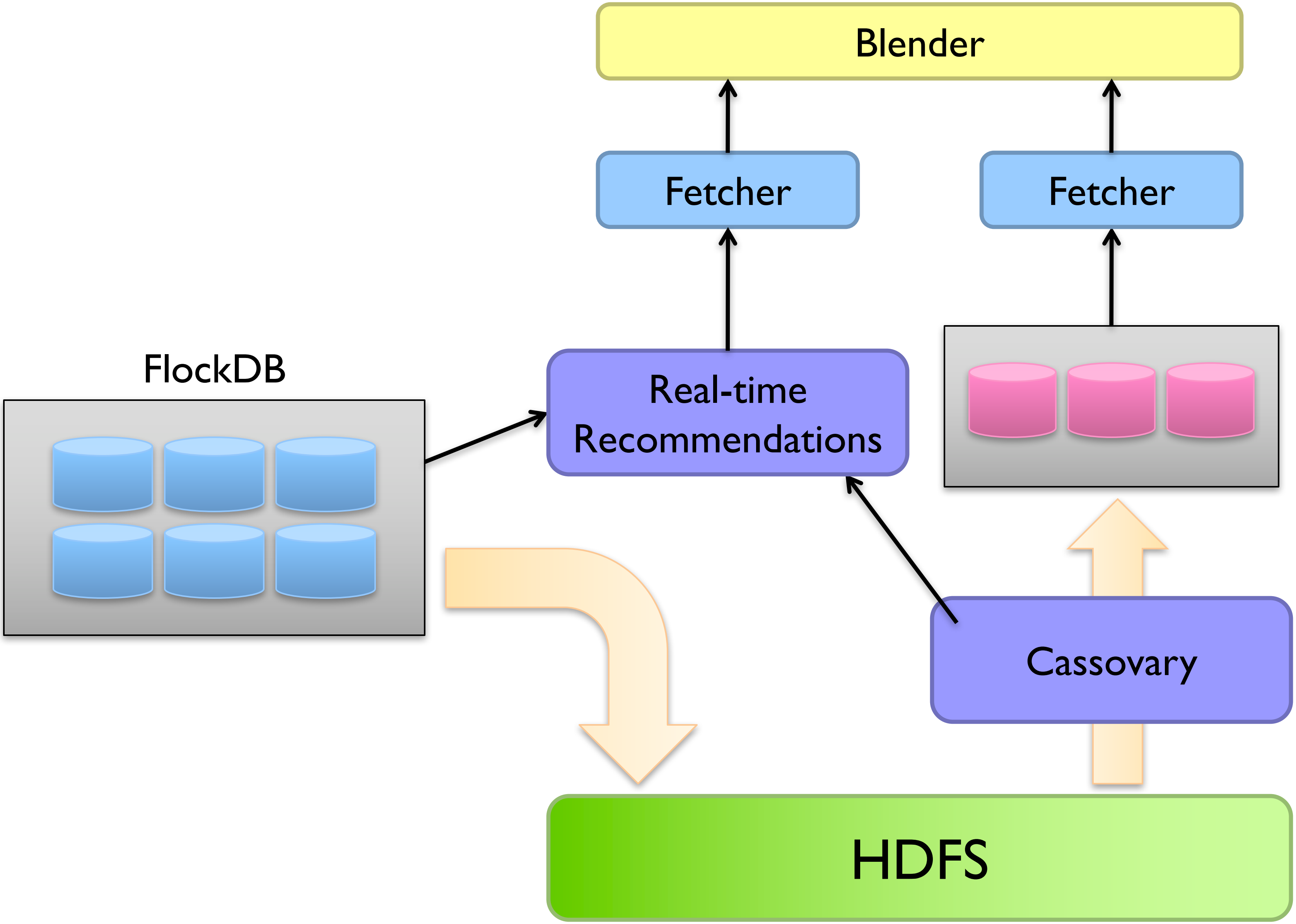
Goel, Lin, Sharma, Wang, and Zadeh. WTF: The Who to Follow Service at Twitter. WWW 2013


Blender
Fetcher
FlockDB
Cassovary
HDFS

What about new users?

Cold start problem: they need recommendations the most!

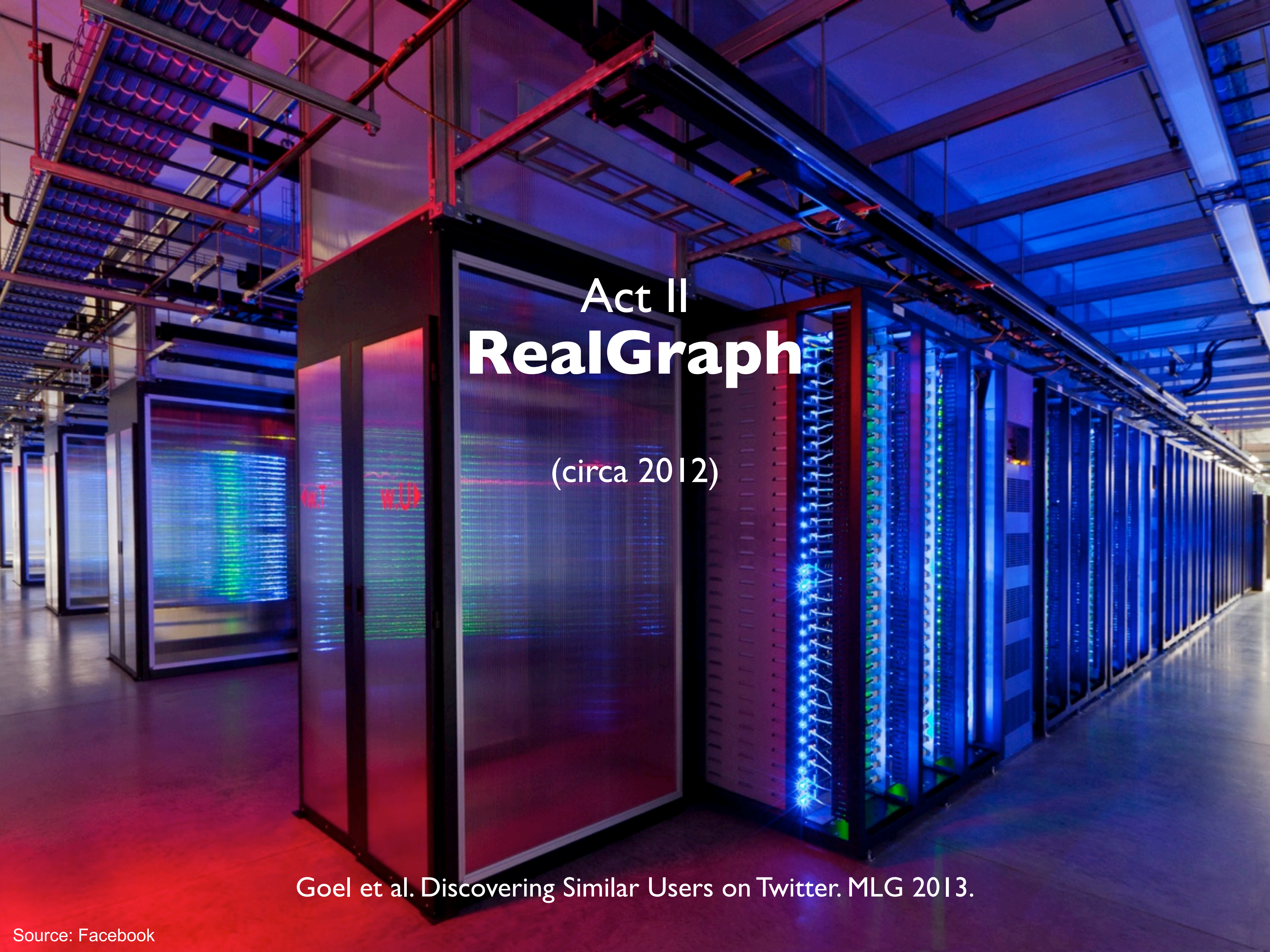



Blender
Fetcher
Fetcher
FlockDB
Real-time
Recommendations
Cassovary
HDFS

Spring 2010: no WTF

seriously, WTF?

Summer 2010: WTF launched



Act II

RealGraph

(circa 2012)

Goel et al. Discovering Similar Users on Twitter. MLG 2013.

Source: Facebook

Another "interesting" design choice:

We migrated from Cassovary back to Hadoop!

Source: Wikipedia (Pistachio)

Whaaaaaa?

Cassovary was a stopgap!

Hadoop provides:

Richer graph structure

Simplified production infrastructure

Scaling and fault-tolerance "for free"

Right choice at the time!

Wait, didn't you say MapReduce sucks?

What exactly is the issue?

Random walks on egocentric 2-hop neighborhood

Naïve approach: self-joins to materialize, then run algorithm

The shuffle is what kills you!

Graph algorithms in MapReduce

Tackle the shuffling problem!

Key insights:

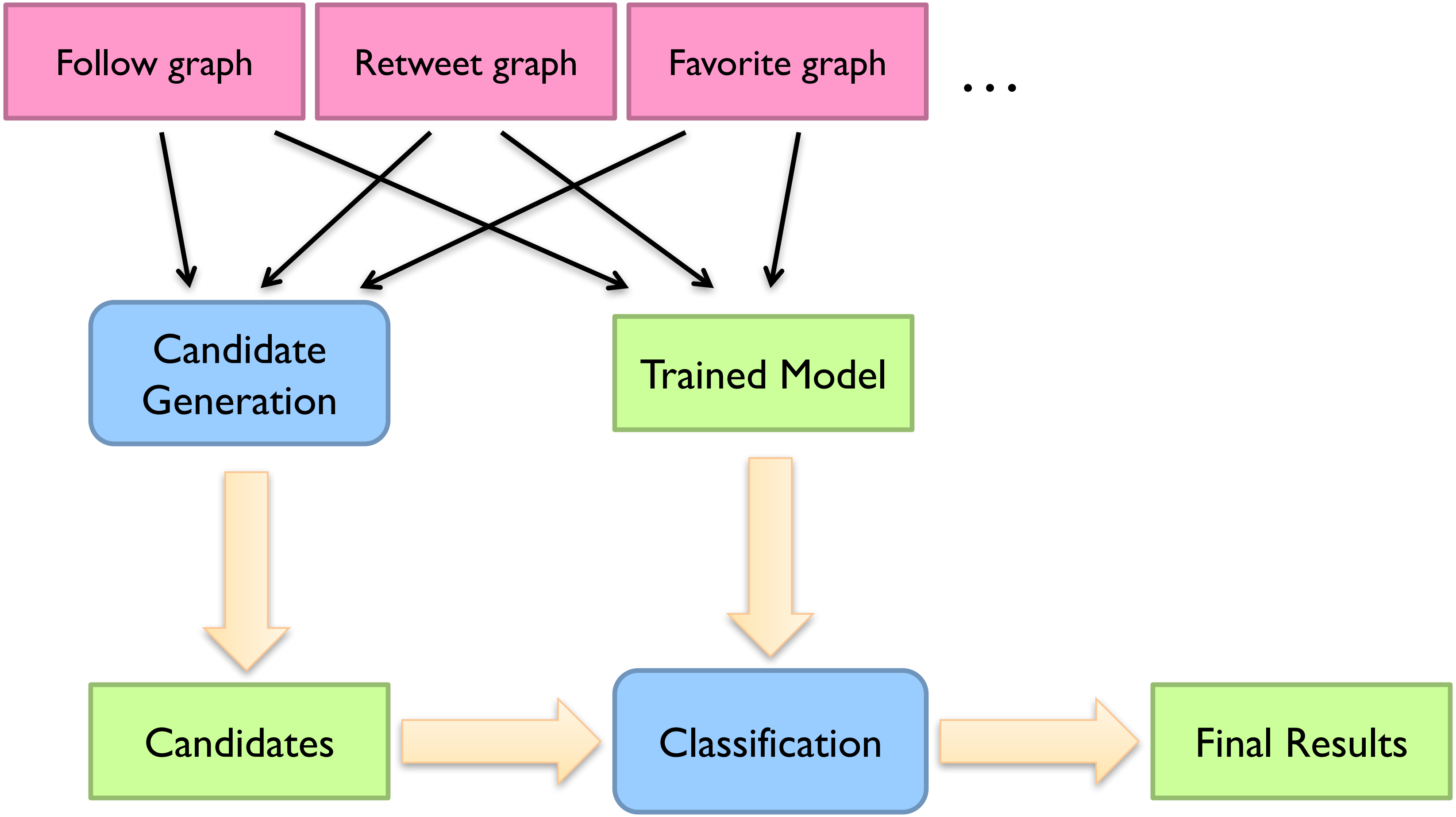
Batch and "stich together" partial random walks*

Clever sampling to avoid full materialization

* Sarma et al. Estimating PageRank on Graph Streams. PODS 2008

Bahmani et al. Fast Personalized PageRank on MapReduce. SIGMOD 2011.

Throw in ML while we're at it…



Lin and Kolcz. Large-Scale Machine Learning at Twitter. SIGMOD 2012.

Act III

MagicRecs

(circa 2013)

Source: Wikipedia (Fire hose)

Isn't the point of Twitter real-time?

So why is WTF still dominated by batch processing?

Source: Wikipedia (Motion Blur)

Observation: fresh recommendations get better engagement

Logical conclusion: generate recommendations in real time!

From batch to real-time recommendations:

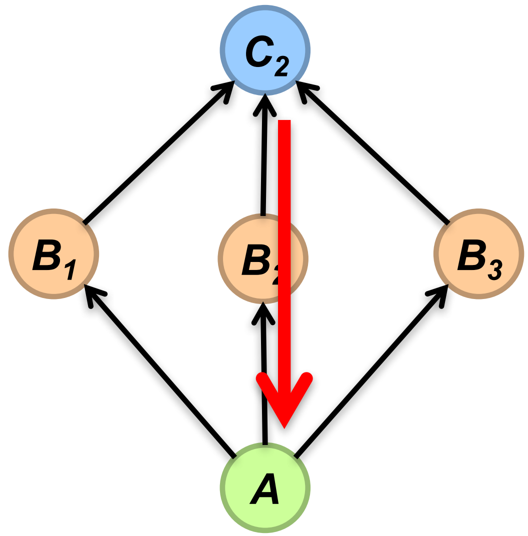
Recommendations based on recent activity

"Trending in your network"

Inverts the WTF problem:

For this user, what recommendations to generate?

Given this new edge, which user to make recommendations to?



Why does this work?

A follows B's because they're interesting

B's following C's because "something's happening"

(generalizes to any activity)

Gupta, Satuluri, Grewal, Gurumurthy, Zhabiuk, Li, and Lin. Real-Time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs. VLDB 2014

Scale of the Problem

$O(10^8)$ vertices, $O(10^{10})$ edges

Designed for $O(10^4)$ events per second

Naïve solutions:

Poll each vertex periodically

Materialize everyone's two-hop neighborhood, intersect

Production solution:

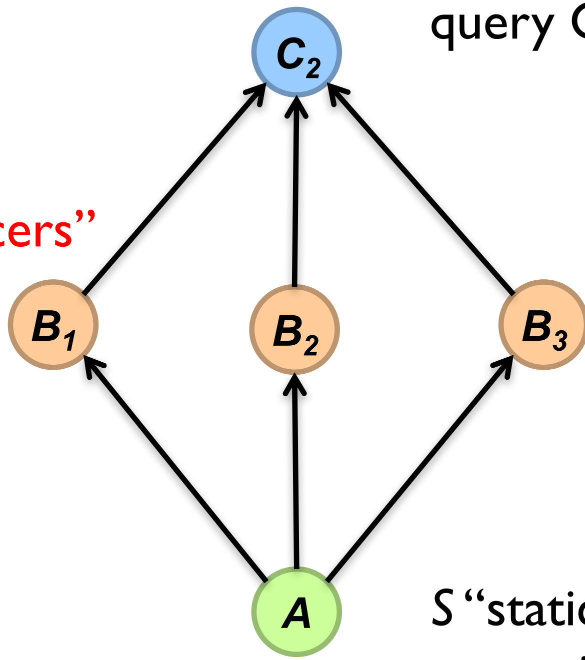
Idea #1: Convert problem into adjacency list intersection

Idea #2: Partition graph to eliminate non-local intersections

Gupta, Satuluri, Grewal, Gurumurthy, Zhabiuk, Li, and Lin. Real-Time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs. VLDB 2014

Single Node Solution

Who we're recommending

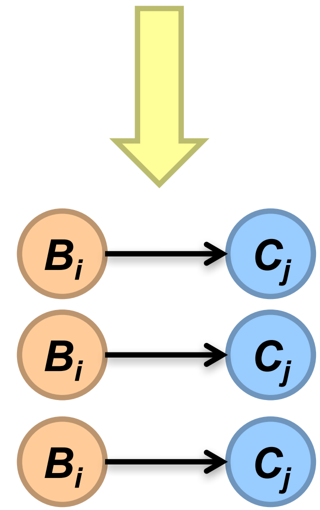


"influencers"

Who we're making the recommendations to

D "dynamic" structure:
stores inverted adjacency lists

query C, return all B's that link to it



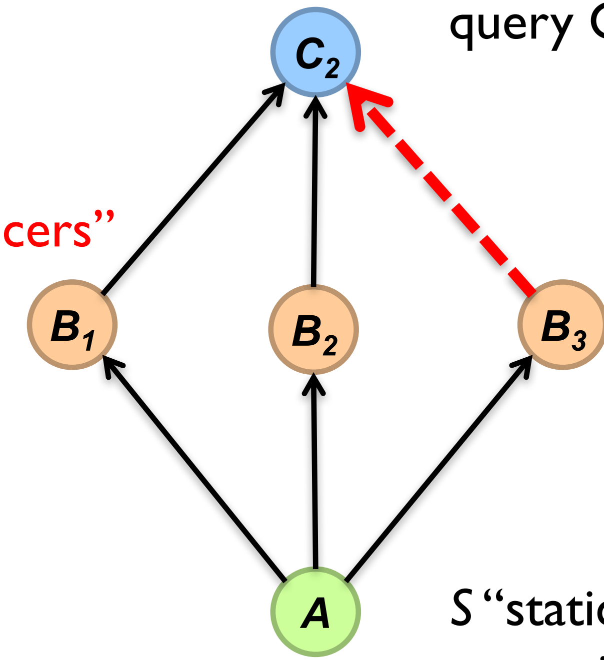
S "static" structure:
stores inverted adjacency lists

query B, return all A's that link to it

Algorithm

Who we're recommending

"influencers"



Who we're making the recommendations to

D "dynamic" structure:
stores inverted adjacency lists

query C, return all B's that link to it

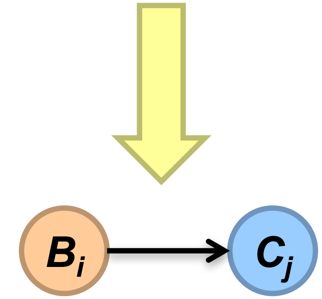
1. Receive B_3 to C_2
2. Query D for C_2, get B_1, B_2, B_3
3. For each B_1, B_2, B_3, query S
4. Intersect lists to compute A's

S "static" structure:
stores inverted adjacency lists

query B, return all A's that link to it

Idea #1: Convert problem into adjacency list intersection

Distributed Solution

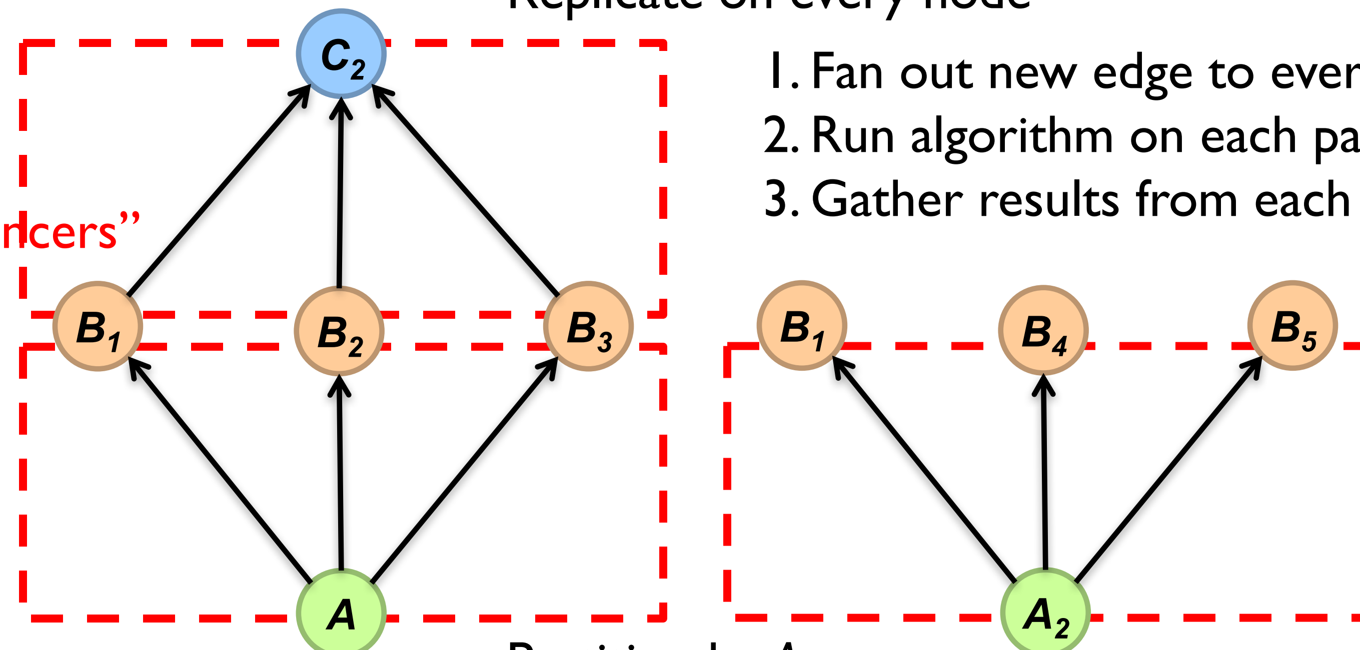


Who we're recommending

Replicate on every node

1. Fan out new edge to every node
2. Run algorithm on each partition
3. Gather results from each partition

"influencers"



Partition by *A*

Who we're making the recommendations to

Idea #2: Partition graph to eliminate non-local intersections

Production Status

Launched September 2013

Usage Statistics (Circa 2014)

Push recommendations to Twitter mobile users

Billions of raw candidates, millions of push notifications daily

Performance

End-to-end latency (from edge creation to delivery):

median 7s, p99 15s

Gupta, Satuluri, Grewal, Gurumurthy, Zhabiuk, Li, and Lin. Real-Time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs. VLDB 2014

Act IV
GraphJet

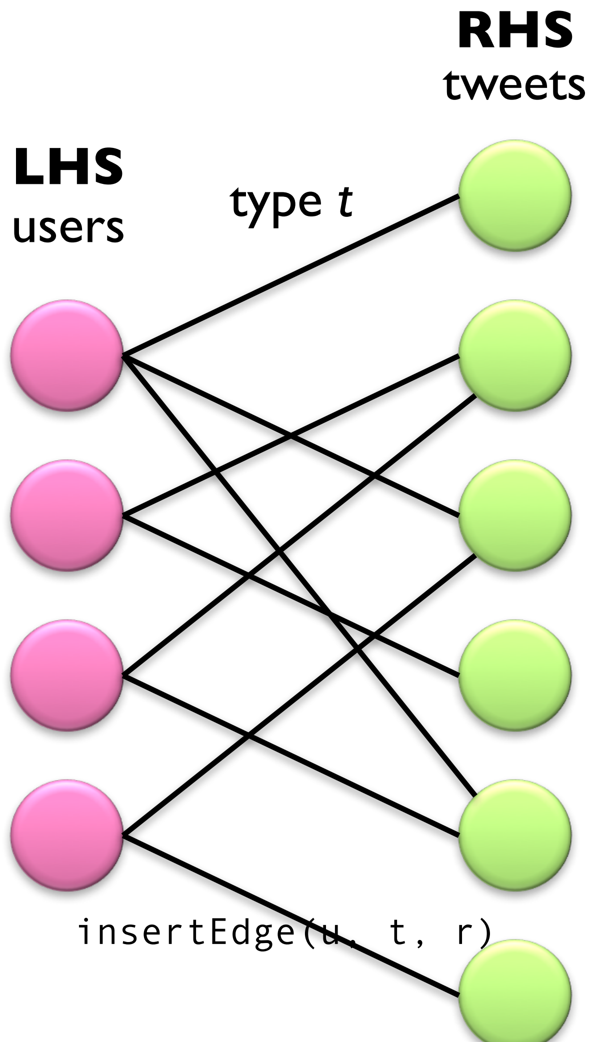
(circa 2014)

Fully bought into the potential of real-time…
but needed something more general

Focused specifically on the interaction graph

Source: flickr (https://www.flickr.com/photos/martinsfoto/6432093025/)

Data Model



`getLeftVertexEdges(u)`
`getLeftVertexRandomEdges(u, k)`

`getRightVertexEdges(t)`
`getRightVertexRandomEdges(t, k)`

Noteworthy design decisions

Make it simple, make it fast!

No partitioning

Focus on recent data, fits on a single machine

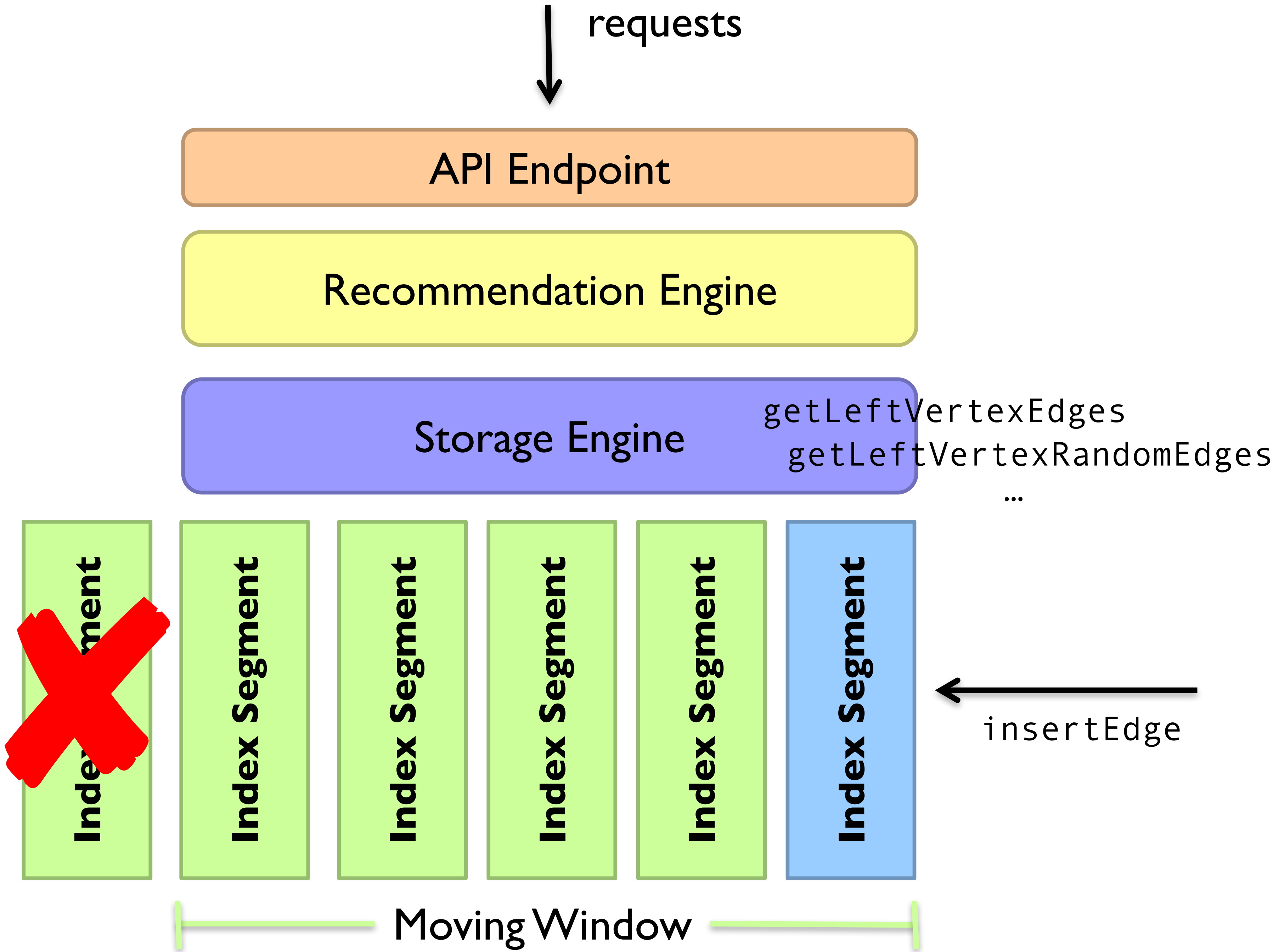
No deletes

Not meaningful w/ interaction data

No arbitrary edge metadata

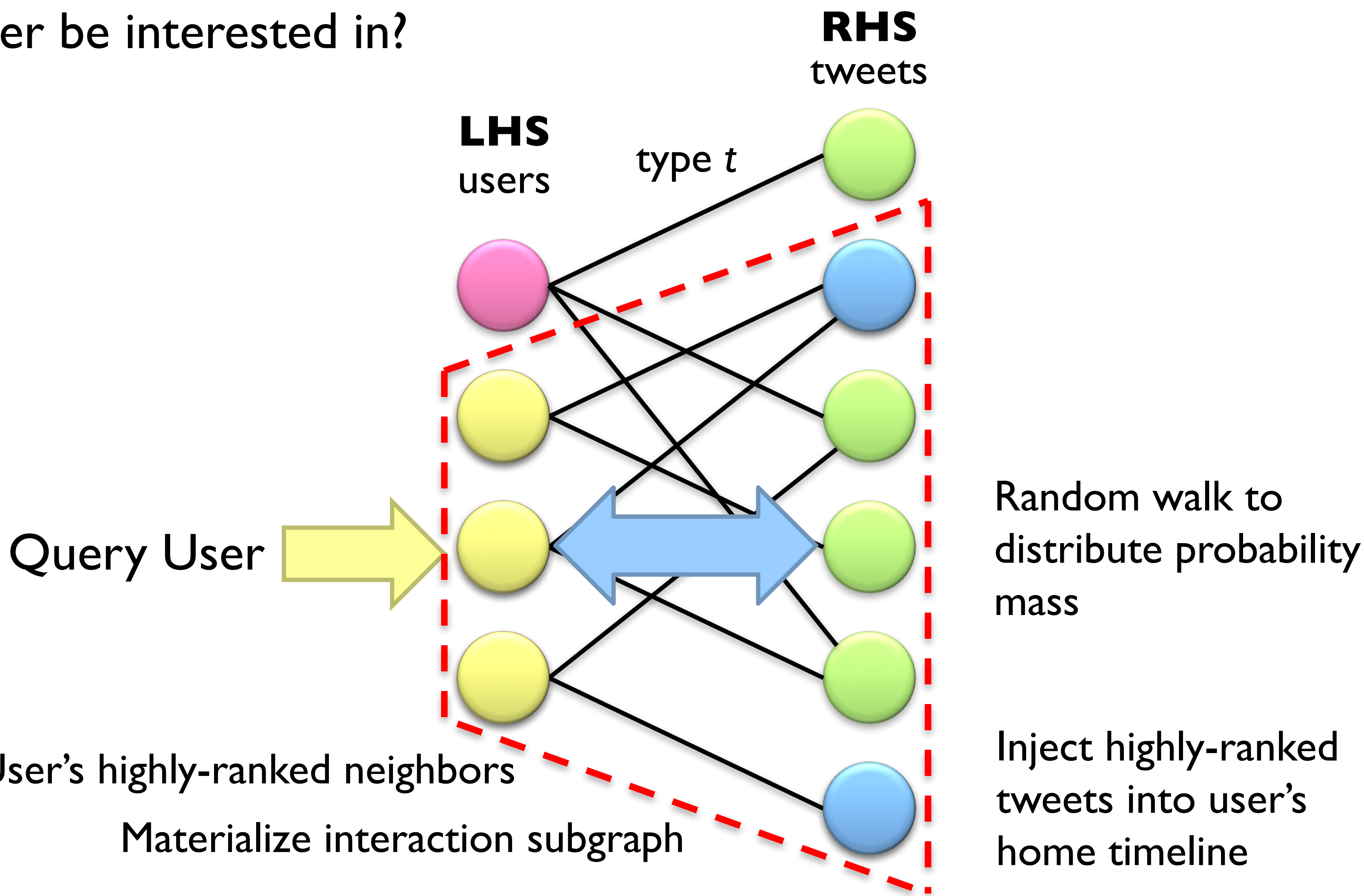
Marginally better results at the cost of space – not worthwhile

Note: design supports revisiting these choices


requests
API Endpoint
Recommendation Engine
Storage Engine
getLeftVertexEdges
getLeftVertexRandomEdges
...
Index Segment
Index Segment
Index Segment
Index Segment
Index Segment
Index Segment
insertEdge
Moving Window

Recommendation Algorithm: Subgraph SALSA

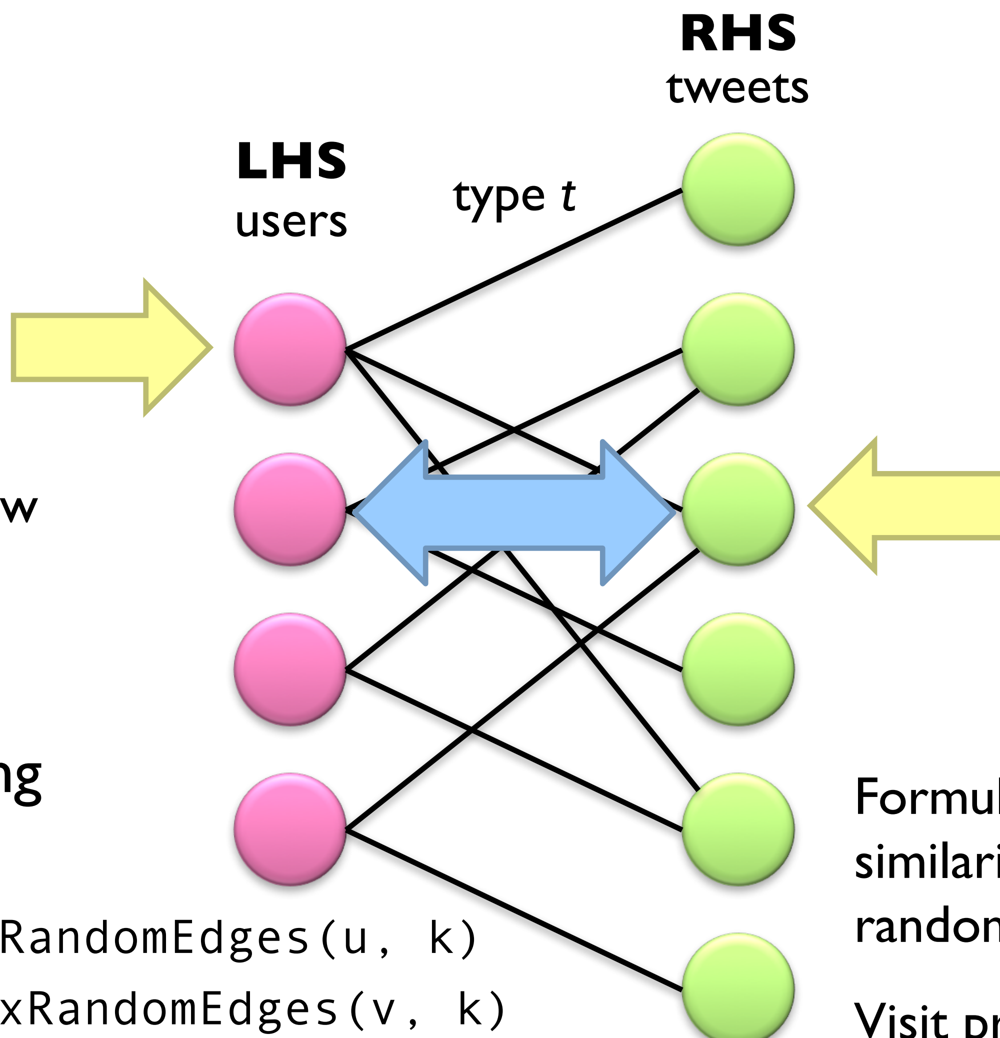
What tweets might a user be interested in?



Recommendation Algorithm: Similarity Query

Query User

Recommend users to follow



Query Tweet

Recommend Content

Efficient sampling API is critical!

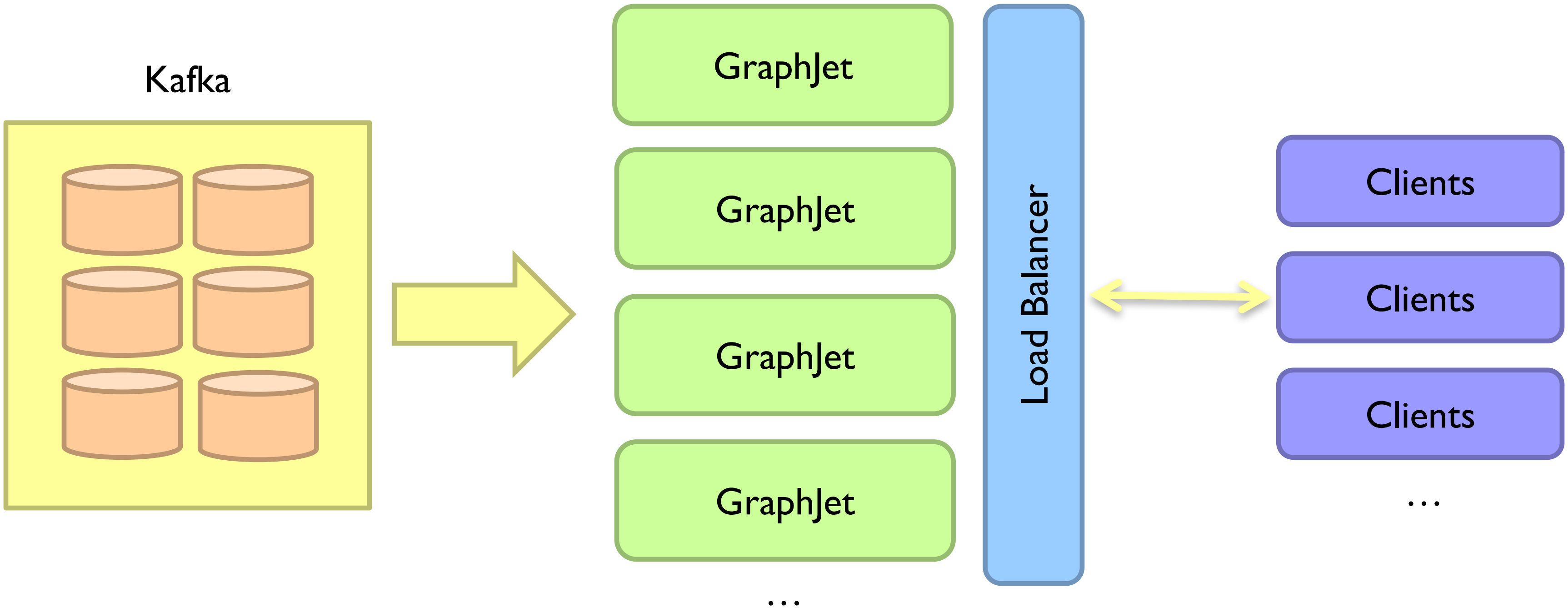
```
getLeftVertexRandomEdges(u, k)
getRightVertexRandomEdges(v, k)
```

Formulation of cosine similarity in terms of random walks

Visit prob. ~ sim

Goel et al. Discovering Similar Users on Twitter. MLG 2013.

Deployment Architecture



Production Status

Started serving production traffic early 2014

Dual Intel Xeon 6-cores (E5-2620 v2) at 2.1 GHz

Cold startup: ingestion at $O(10^6)$ edges per sec from Kafka

Steady state: ingestion at $O(10^4)$ edges per sec

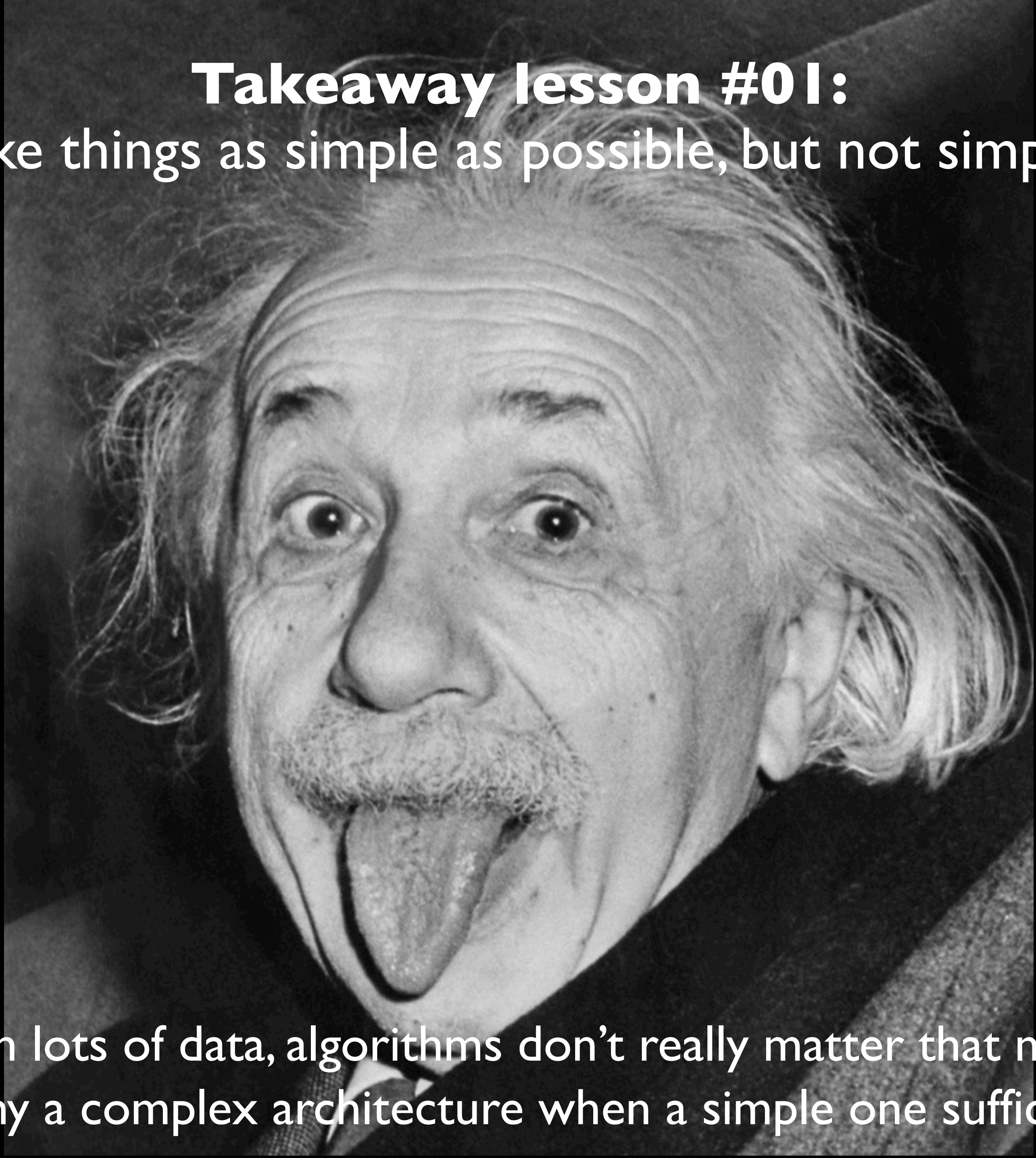
Space usage: $O(10^9)$ edges in < 30 GB

Sample recommendation algorithm: subgraph SALSA

500 QPS, p50 = 19ms, p99 = 33ms

Takeaway lesson #01:

Make things as simple as possible, but not simpler.

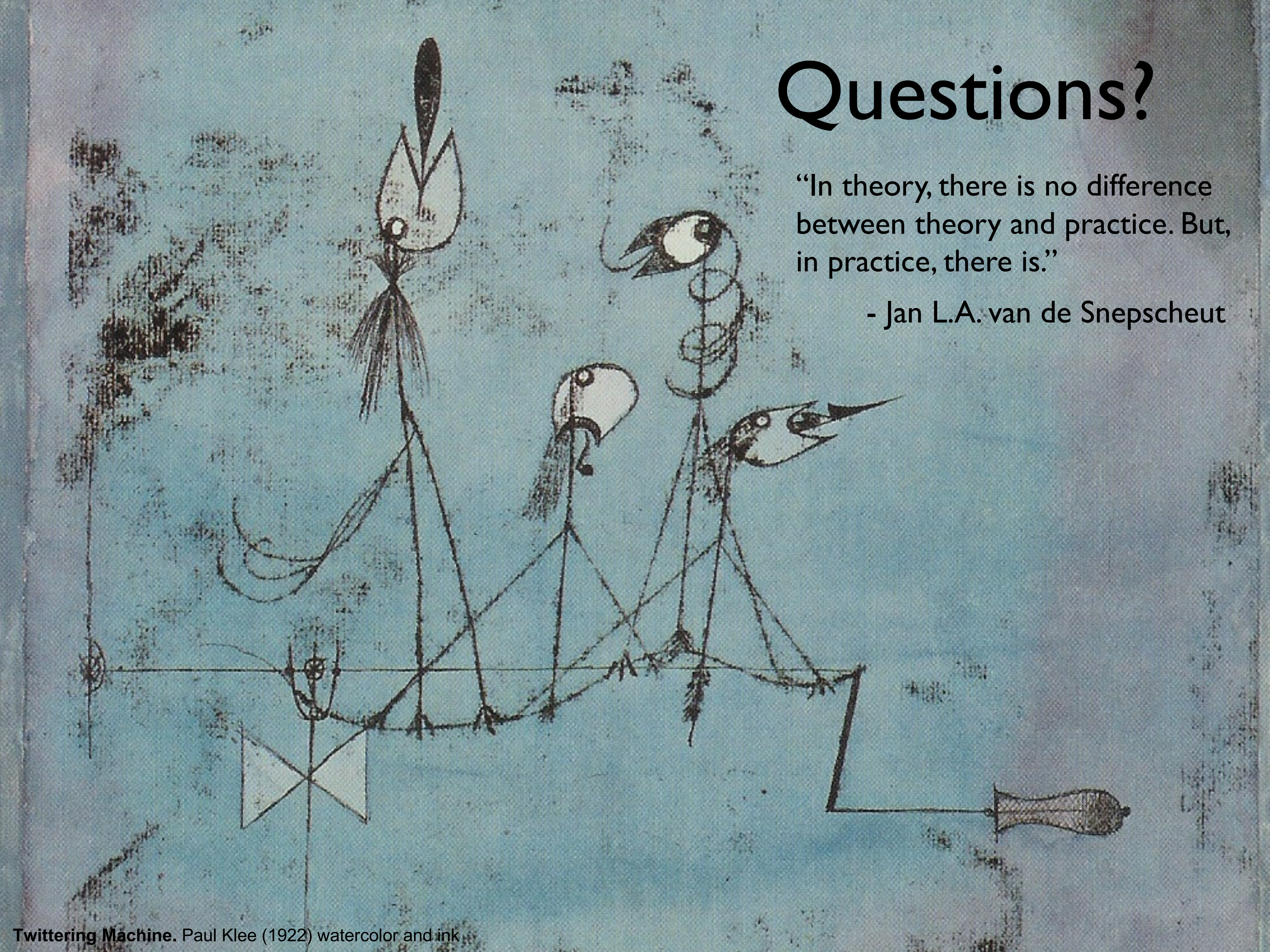


With lots of data, algorithms don't really matter that much

Why a complex architecture when a simple one suffices?

Source: https://www.flickr.com/photos/43677780@N07/6240710770/


Takeaway lesson #11:
Visiting and revisiting design decisions
Source: https://www.flickr.com/photos/exmachina/8186754683/



Questions?

"In theory, there is no difference between theory and practice. But, in practice, there is."

- Jan L.A. van de Snepscheut

Twittering Machine. Paul Klee (1922) watercolor and ink