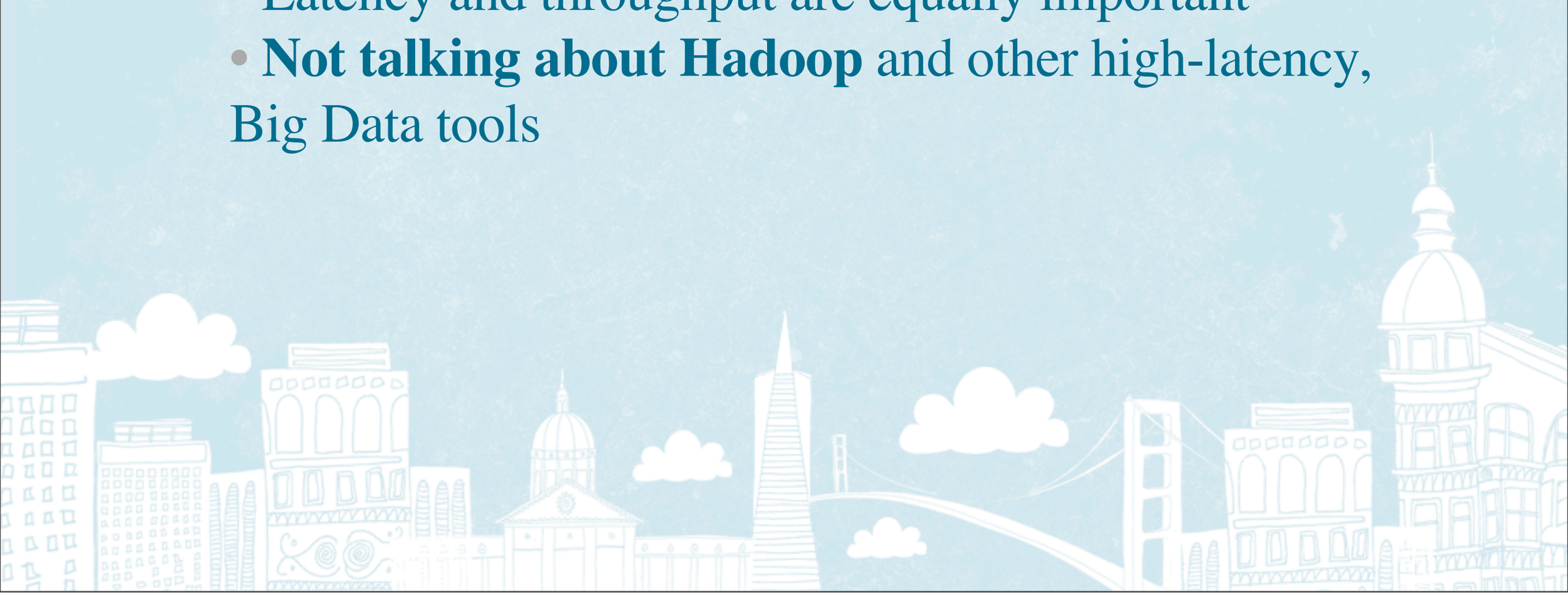
Big Data in Real-Time *at Twitter*

by Nick Kallen (@nk)

What is Real-Time Data?

- On-line queries for a single **web request**
- **Off-line** computations with *very* **low latency**
- Latency and throughput are equally important
- **Not talking about Hadoop** and other high-latency, Big Data tools



The three data problems

- **Tweets**
- Timelines
- Social graphs

Your facial hair tells me you revel in celibacy.

11:33 AM Sep 25th, 2009 via web

Reply Retweet

sween
Jason Sweeney

What is a Tweet?

- 140 character message, plus some metadata
- Query patterns:
  - by **id**
  - by **author**
  - (also @replies, but not discussed here)
- Row Storage

Find by primary key: 43761677936

Your facial hair tells me you revel in celibacy.

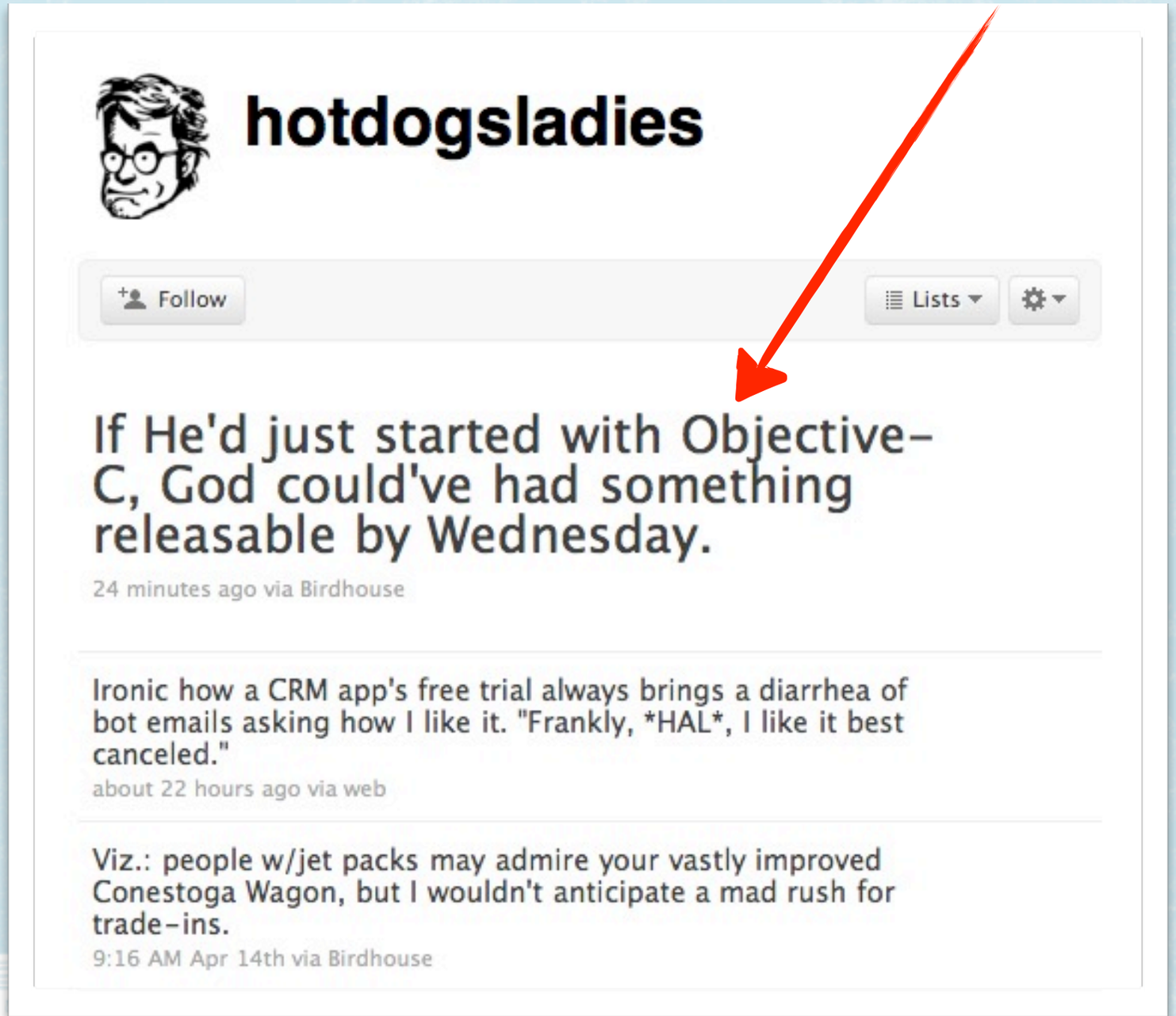
11:33 AM Sep 25th, 2009 via web

Reply Retweet

sween

Jason Sweeney

Find all by user_id: 749863



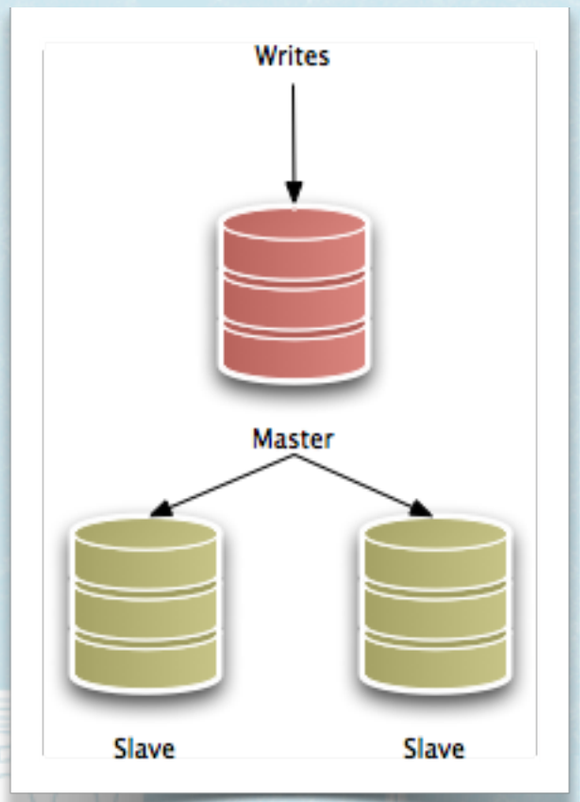
Original Implementation

id	user_id	text	created_at
20	12	just setting up my twttr	2006-03-21 20:50:14
29	12	inviting coworkers	2006-03-21 21:02:56
34	16	Oh shit, I just twittered a little.	2006-03-21 21:08:09

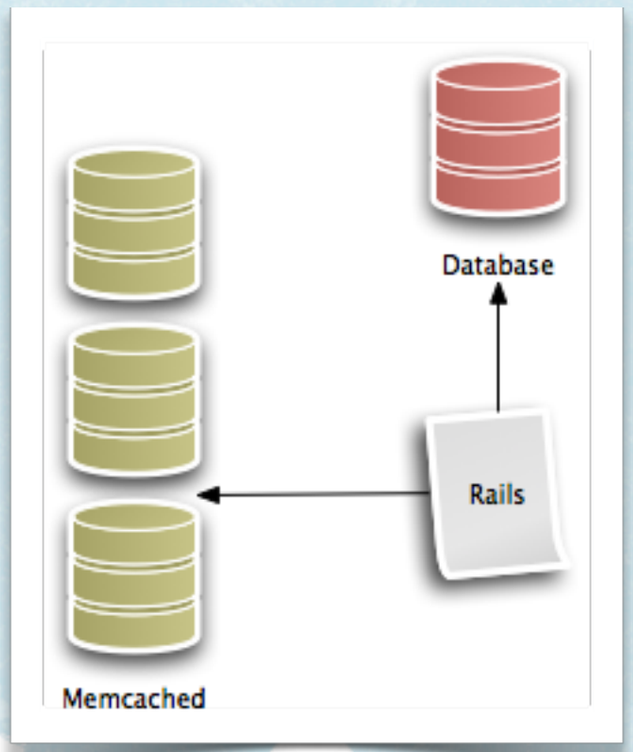
- **Relational**
- **Single table**, vertically scaled
- **Master-Slave** replication and **Memcached** for read throughput.

Original Implementation

Master-Slave Replication



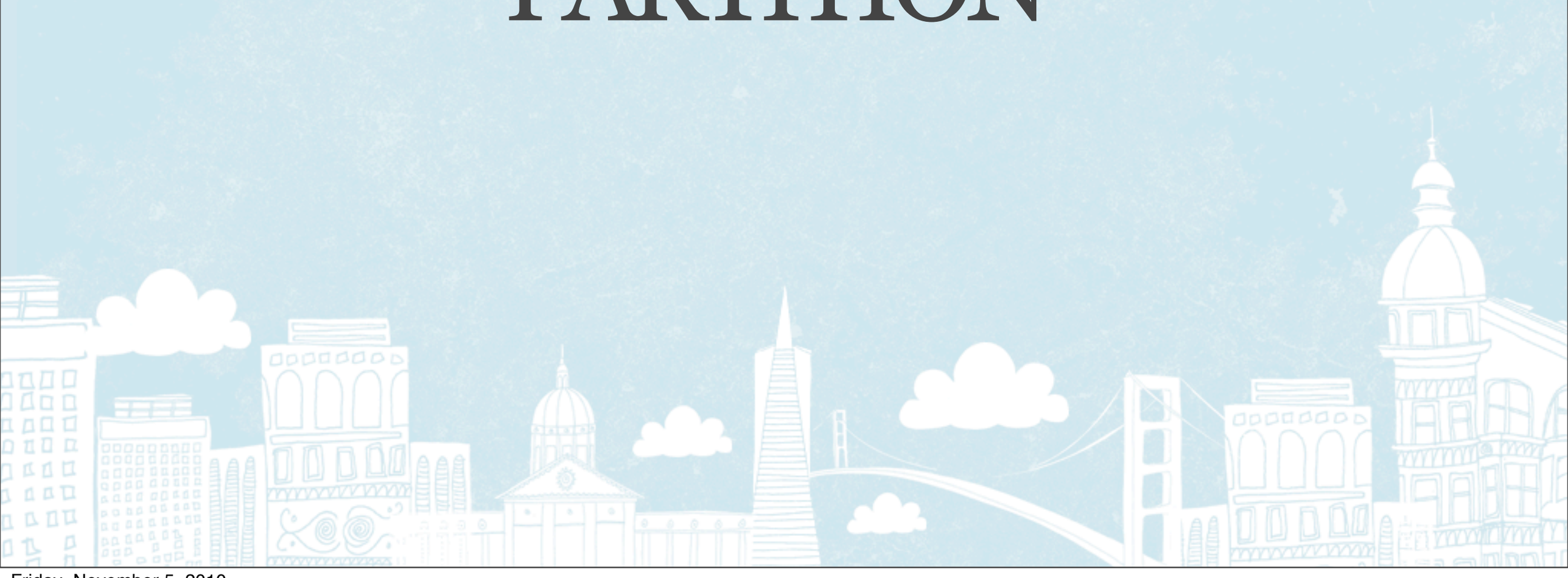
Memcached for reads



Problems w/ solution

- **Disk space**: did not want to support disk arrays larger than 800GB
- At 2,954,291,678 tweets, disk was over 90% utilized.

PARTITION



Dirt-Goose Implementation

Queries try each partition in order until enough data is accumulated

Partition by time

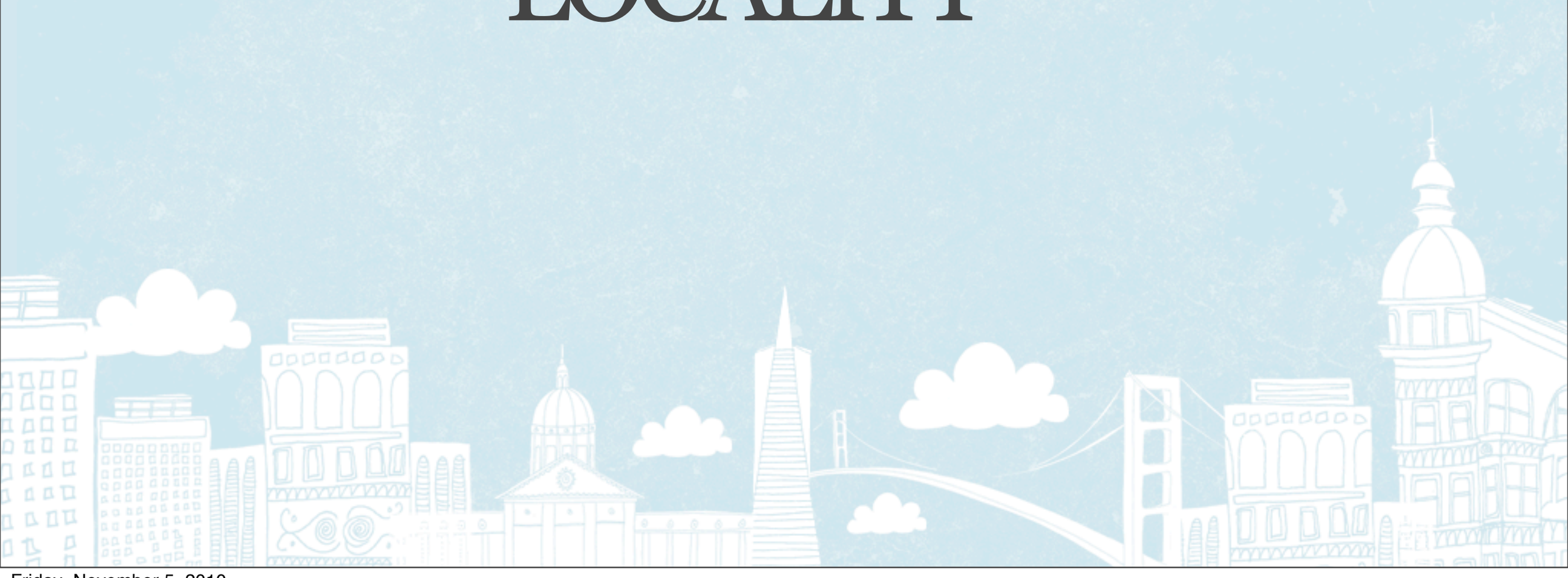
Partition 2

id	user_id
24	...
23	...

Partition 1

id	user_id
22	...
21	...

LOCALITY



Problems w/ solution

- Write throughput

T-Bird Implementation

Partition by primary key

Partition 1

id	text
20	...
22	...
24	...

Partition 2

id	text
21	...
23	...
25	...

Finding recent tweets by user_id queries N partitions

T-Flock

Partition user_id index by user id

Partition 1

user_id	id
1	1
3	58
3	99

Partition 2

user_id	id
2	21
2	22
2	27

Low Latency

	PK Lookup
Memcached	1ms
T-Bird	5ms

Principles

- Partition and index
- Index and partition
- Exploit **locality** (in this case, **temporal** locality)
  - New tweets are requested most frequently, so usually only 1 partition is checked

The three data problems

- ~~Tweets~~
- **Timelines**
- Social graphs

What's happening? 140

Latest: Man I need a Tecate 2 minutes ago

Tweet

Home

missionhipster Man I need a Tecate
2 minutes ago via web

NeonIndian Well even if it was an LP after all.. still stoked on that iphone love.
1:54 PM Apr 14th via web

NeonIndian psychic chasms EP?
12:14 PM Apr 14th via mobile web

NeonIndian Overheard band name of the night: Demon Semen.
11:40 PM Apr 13th via mobile web

NeonIndian kareoke to the max! time to take vengance with some jessie's girl...
10:39 PM Apr 13th via mobile web

Sightglass No sweat, big guy--really. RT @therealjoshcook: sorry about your siphon @sighglass :(
6:49 PM Apr 12th via Birdfeed Reply Retweet

What is a Timeline?

- Sequence of tweet ids
- Query pattern: get by user_id
- High-velocity bounded vector
- RAM-only storage

Tweets from 3 different people

What's happening? 140

Latest: Man I need a Tecate 2 minutes ago

Tweet

Home

missionhipster Man I need a Tecate
2 minutes ago via web

NeonIndian Well even if it was an LP after all.. still stoked on that iphone love.
1:54 PM Apr 14th via web

NeonIndian psychic chasms EP?
12:14 PM Apr 14th via mobile web

NeonIndian Overheard band name of the night: Demon Semen.
11:40 PM Apr 13th via mobile web

NeonIndian kareoke to the max! time to take vengance with some jessie's girl...
10:39 PM Apr 13th via mobile web

Sightglass No sweat, big guy--really. RT @therealjoshcook: sorry about your siphon @sighglass :(
6:49 PM Apr 12th via Birdfeed Reply Retweet

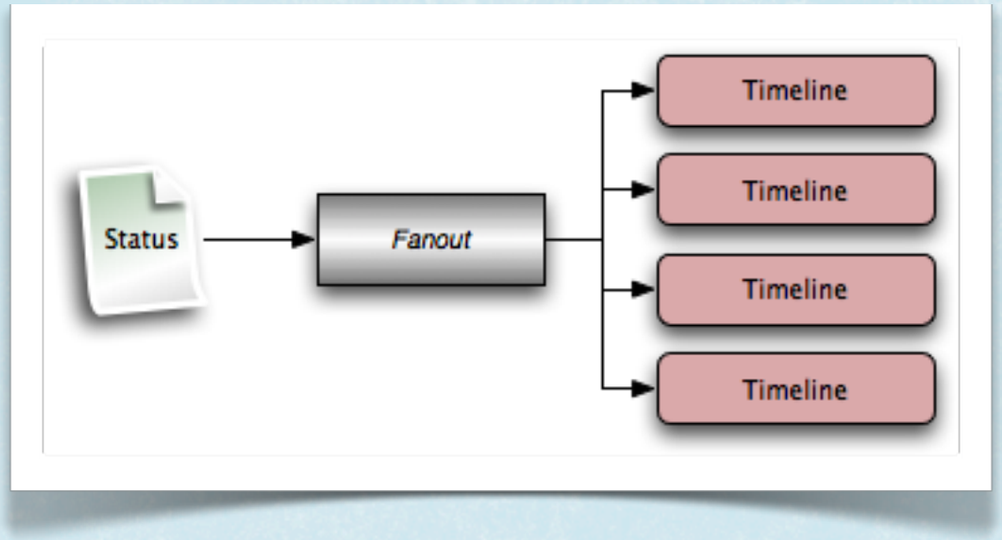
Original Implementation

```
SELECT * FROM tweets
WHERE user_id IN
  (SELECT source_id
   FROM followers
   WHERE destination_id = ?)
ORDER BY created_at DESC
LIMIT 20
```

Crazy slow if you have lots of friends or indices can't be kept in RAM

OFF-LINE *VS.* ONLINE COMPUTATION

Current Implementation



- Sequences stored in **Memcached**
- Fanout off-line, but has a **low latency SLA**
- Truncate at random intervals to ensure bounded length
- On **cache miss**, merge user timelines

Throughput Statistics

date	daily pk tps	all-time pk tps	fanout ratio	deliveries
10/7/2008	30	120	175:1	21,000
11/1/2010	1500	3,000	700:1	2,100,000

2.1m

Deliveries per second

MEMORY HIERARCHY

Possible implementations

- Fanout to **disk**
  - Ridonculous number of IOPS required, even with fancy buffering techniques
  - Cost of rebuilding data from other durable stores not *too* expensive
- Fanout to **memory**
  - Good if cardinality of corpus * bytes/datum not too many GB

Low Latency

get	append	fanout
1ms	1ms	<1s*

* Depends on the number of followers of the tweeter

Principles

- Off-line vs. Online computation
  - The answer to some problems can be **pre-computed** if the amount of work is **bounded** and the query pattern is very limited
- Keep the memory hierarchy in mind

The three data problems

- ~~Tweets~~
- ~~Timelines~~
- **Social graphs**

Your 15 followers

User / Name

staykreative ✔ Following
Stay Kreative
I'm at Stay Kreative (164 Stanton St, Clinton St, New York). http://4sq.com/ajRPQ1
about 6 hours ago

thebowlingrobot
Alison Dale | 415, 504,505
I subscribed to jamiliya13's channel on YouTube http://www.youtube.com/user/jamiliya13?feature=autoshare_twitter 1:04 PM Apr 14th

immergent ✔ Following
Immergent | Los Angeles
#NameThatSong "Please allow me to introduce myself I'm a man of wealth and taste..." http://ow.ly/1yWA1 about 3 hours ago

thenightatx ✔ Following
The Night | Austin, TX
that was fun thanks y'all :) EP coming out in May, with Radio, Summertime, The Night, Twisted and more http://fb.me/vUx3pyRs 4 days ago

Verified Account

Name ashton kutcher
Location here
Web http://www.facebo...
Bio I make stuff, actually I make up stuff, stories mostly, collaborations of thoughts, dreams, and actions. Thats me.

405	4,764,815	35,887
following	followers	listed

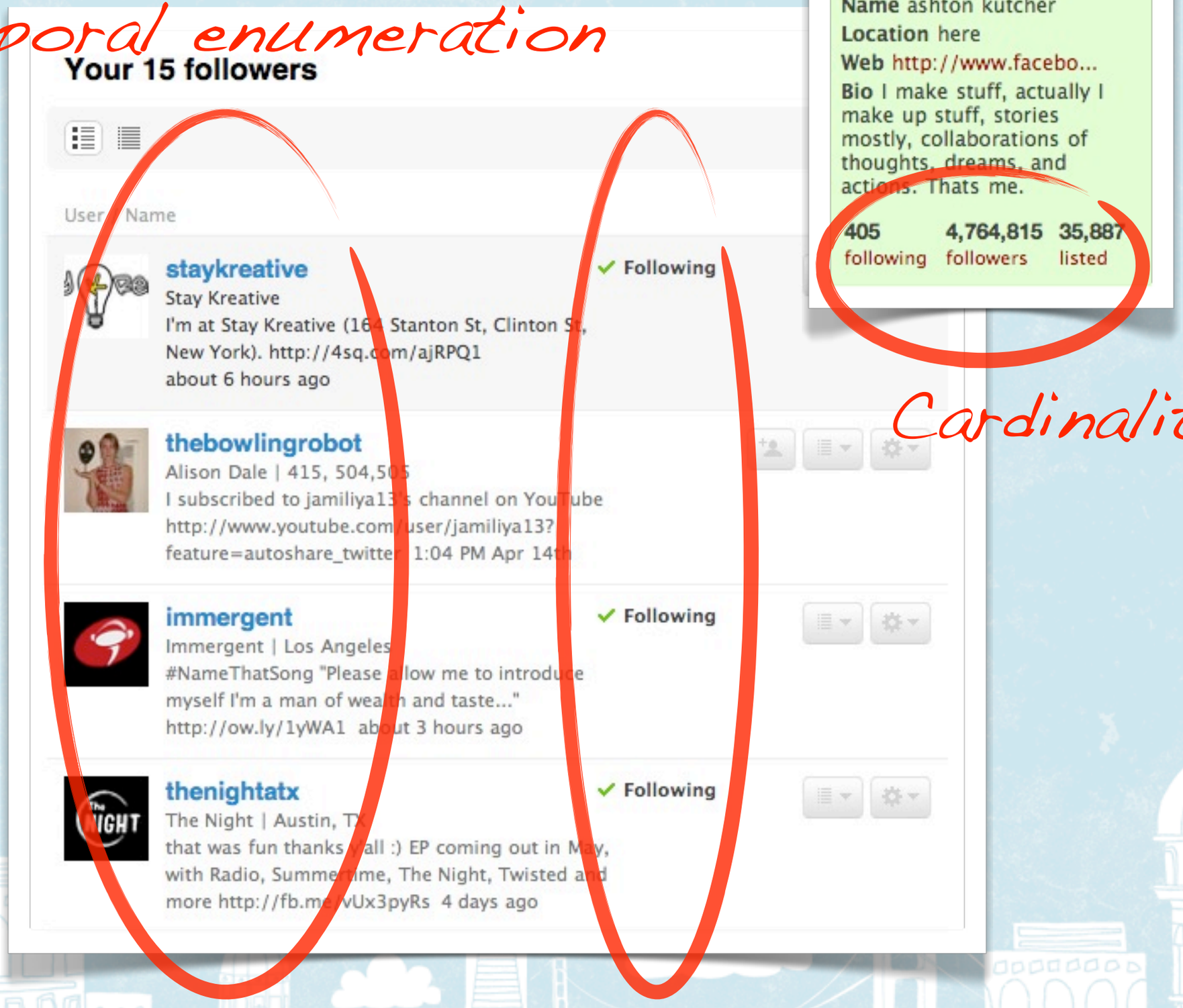
What is a Social Graph?

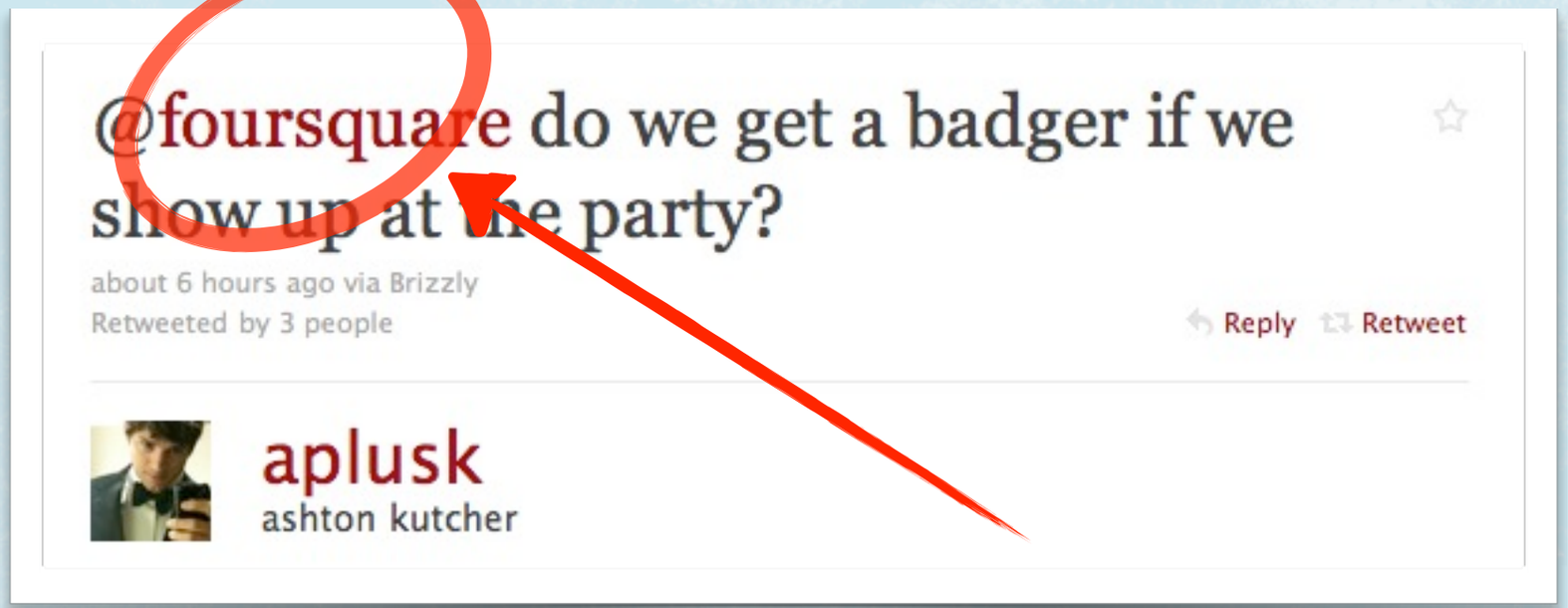
- List of who follows whom, who blocks whom, etc.
- Operations:
  - Enumerate by time
  - Intersection, Union, Difference
  - Inclusion
  - Cardinality
  - Mass-deletes for spam
- Medium-velocity unbounded vectors
- Complex, predetermined queries

Inclusion

Temporal enumeration

Cardinality





Intersection: Deliver to people who follow both @aplusk and @foursquare

Original Implementation

Index

Index

source_id	destination_id
20	12
29	12
34	16

- **Single table**, vertically scaled
- **Master-Slave** replication

Problems w/ solution

- Write throughput
- Indices couldn't be kept in RAM

Edges stored in both directions

Current solution

Forward

source_id	destination_id	updated_at	x
20	12	20:50:14	x
20	13	20:51:32	
20	16		

Backward

destination_id	source_id	updated_at	x
12	20	20:50:14	x
12	32	20:51:32	
12	16		

- Partitioned by user id
- Edges stored in "forward" and "backward" directions
- **Indexed** by time
- **Indexed** by element (for **set algebra**)
- Denormalized cardinality

Partitioned by user

Challenges

- Data consistency in the presence of failures
- Write operations are **idempotent**: retry until success
- **Last-Write Wins** for edges
  - (with an **ordering relation on State** for time conflicts)
- Other **commutative** strategies for mass-writes

Low Latency

cardinality	iteration	write ack	write materialize	inclusion
1ms	100edges/ms*	1ms	16ms	1ms

* 2ms lower bound

Principles

- It is not possible to pre-compute set algebra queries
- **Partition, replicate, index**. Many efficiency and scalability problems are solved the same way

The three data problems

- ~~Tweets~~
- ~~Timelines~~
- ~~Social graphs~~

Summary Statistics

	reads/second	writes/ second	cardinality	bytes/item	durability
Tweets	100k	1100	30b	300b	durable
Timelines	80k	2.1m	a lot	3.2k	volatile
Graphs	100k	20k	20b	110	durable

Principles

- All **engineering solutions are transient**
- Nothing's perfect but some solutions are good enough for a while
- Scalability solutions aren't magic. They involve **partitioning, indexing, and replication**
- All data for real-time queries **MUST** be in memory. **Disk is for writes only**.
- Some problems can be solved with **pre-computation**, but a lot can't
- Exploit locality where possible