

Activity Feeds Architecture

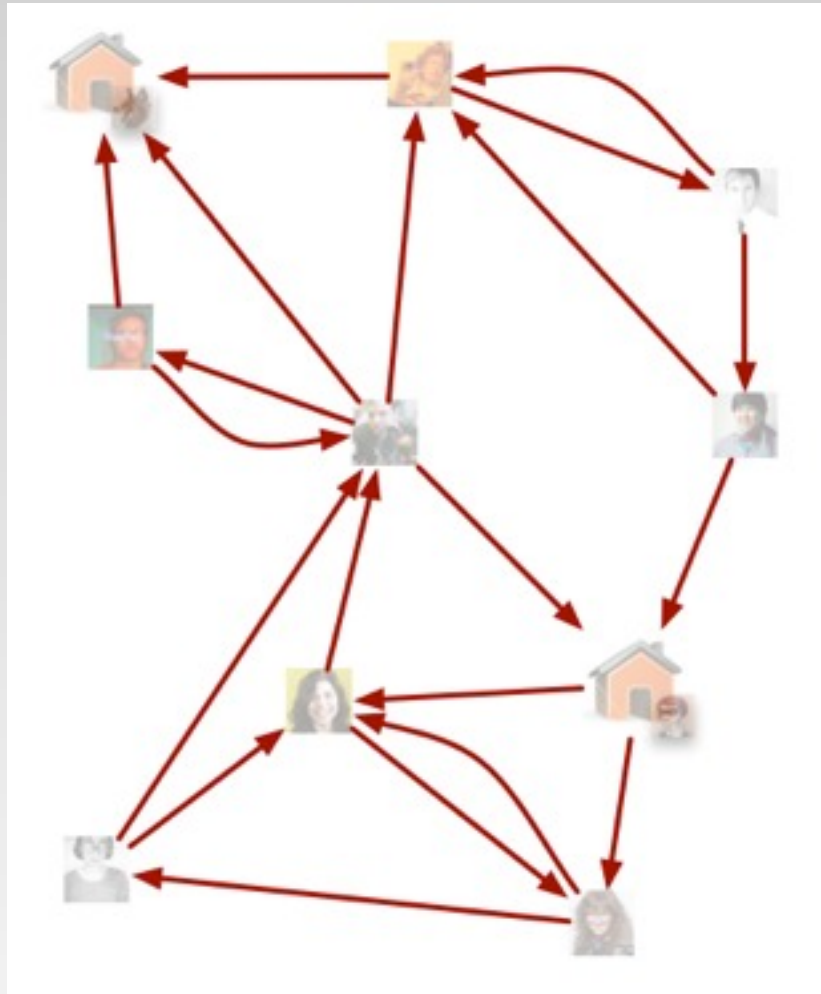
January, 2011

To Be Covered:

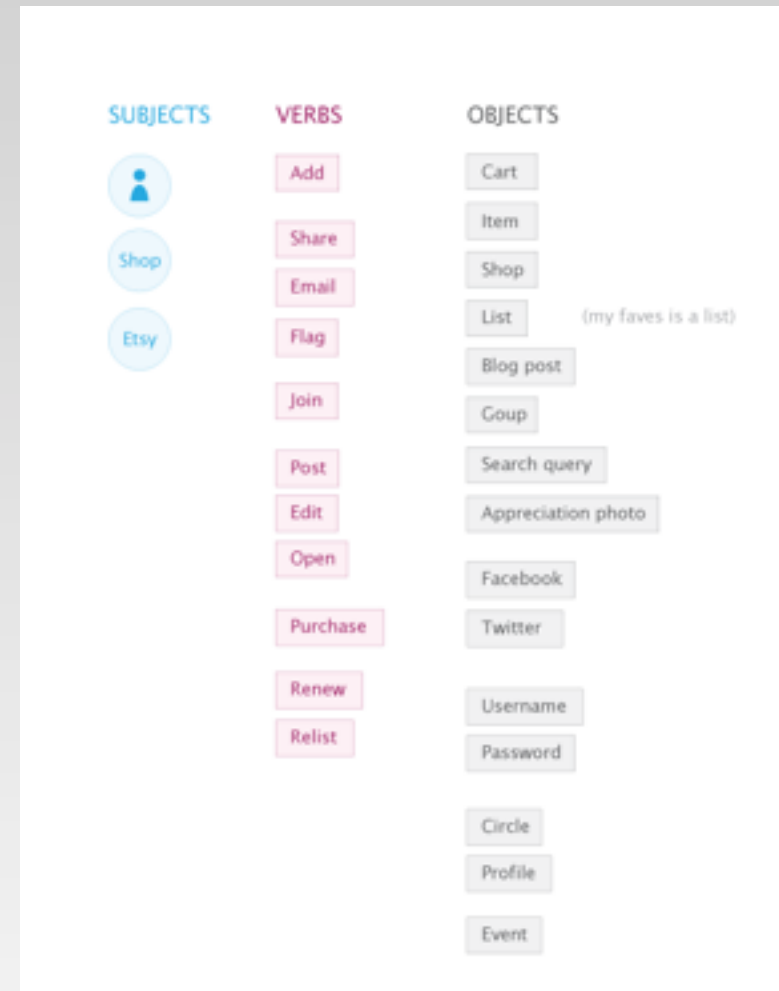
- Data model
- Where feeds come from
- How feeds are displayed
- Optimizations

Fundamental Entities

Connections



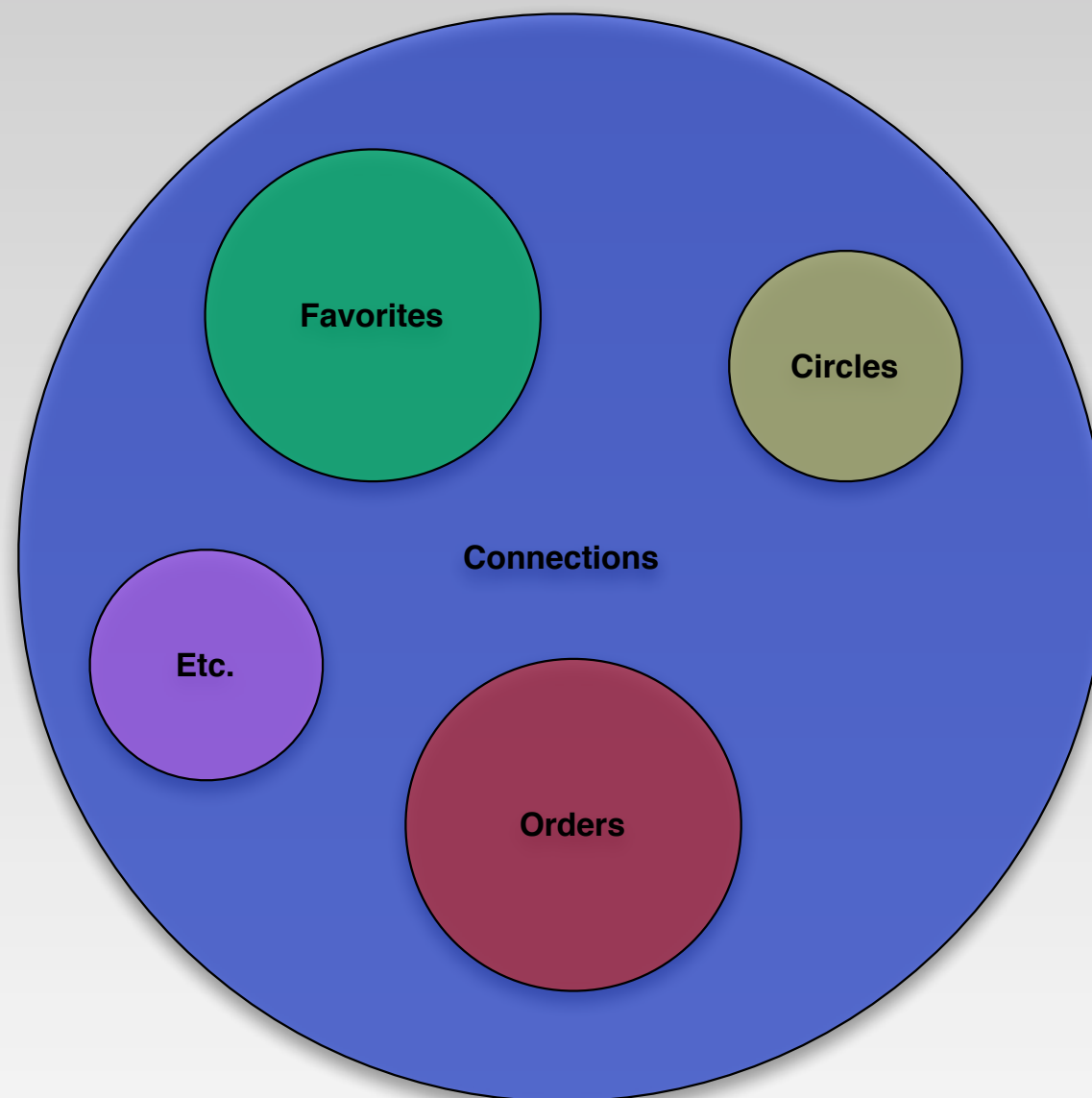
Activities



Friday, January 14, 2011

There are two fundamental building blocks for feeds: connections and activities. Activities form a log of what some entity on the site has done, or had done to it. Connections express relationships between entities. I will explain the data model for connections first.

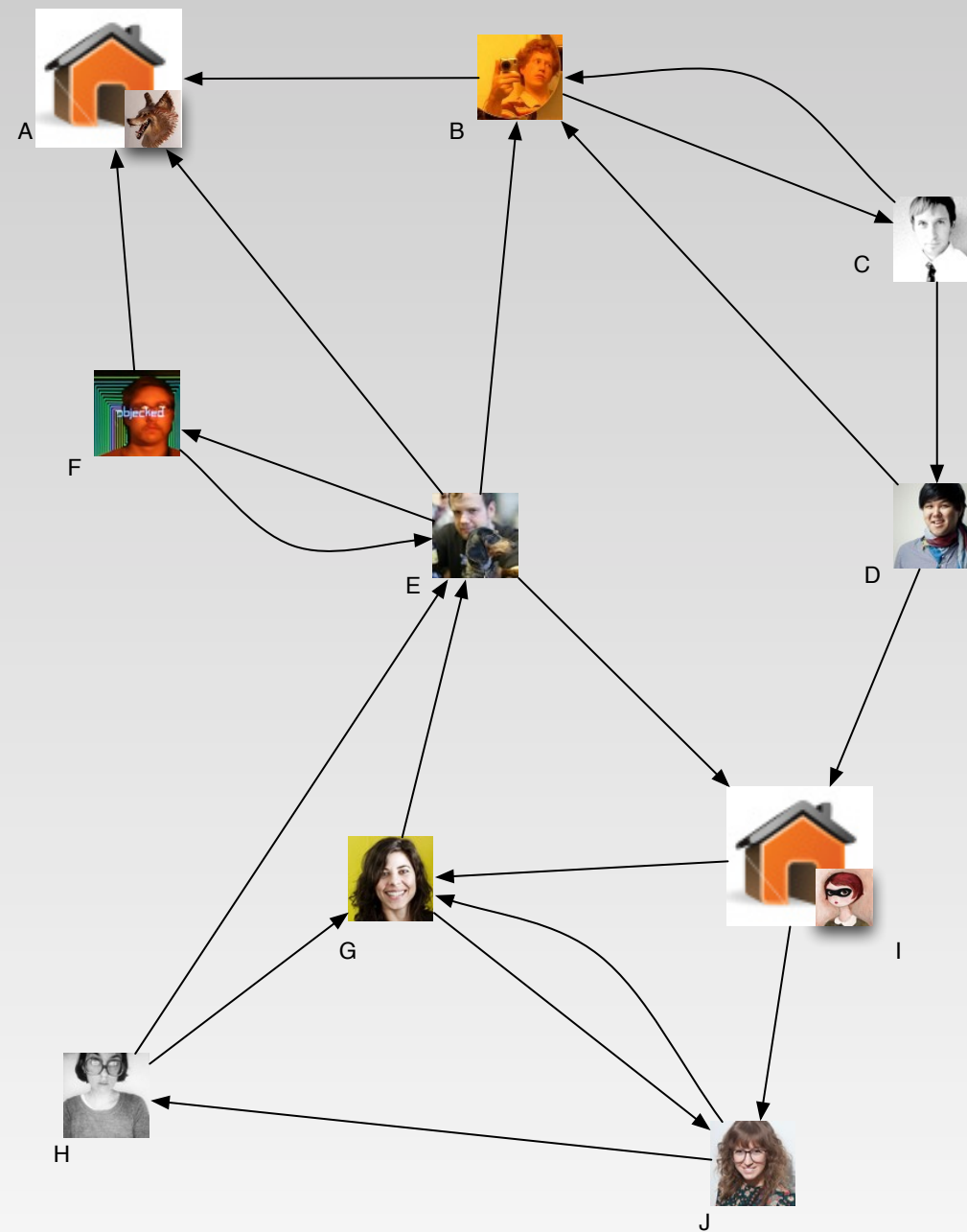
Connections



Friday, January 14, 2011

Connections are a superset of Circles, Favorites, Orders, and other relationships between entities on the site.

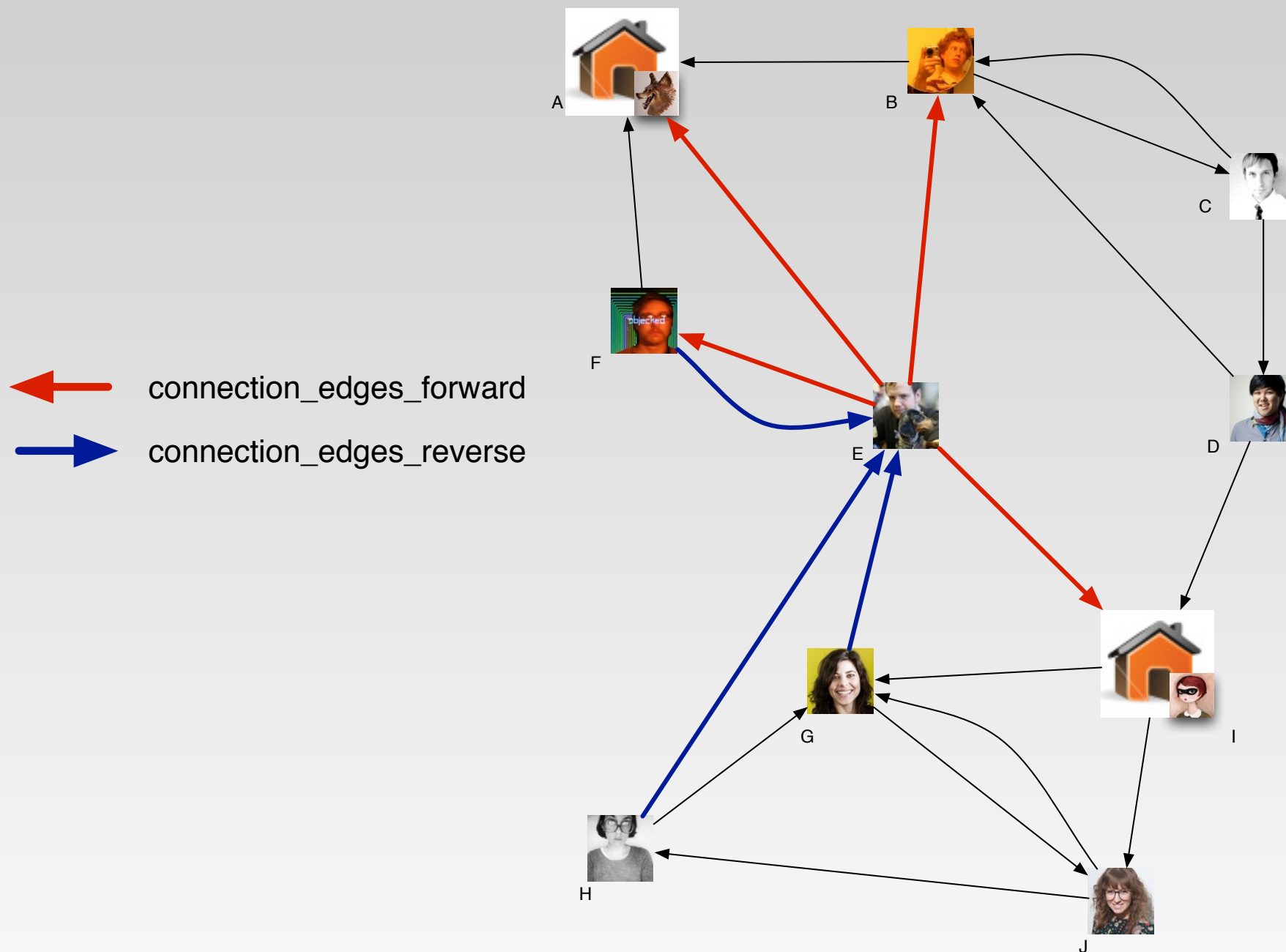
Connections



Friday, January 14, 2011

Connections are implemented as a directed graph.
Currently, the nodes can be people or shops. (In principle they can be other objects.)

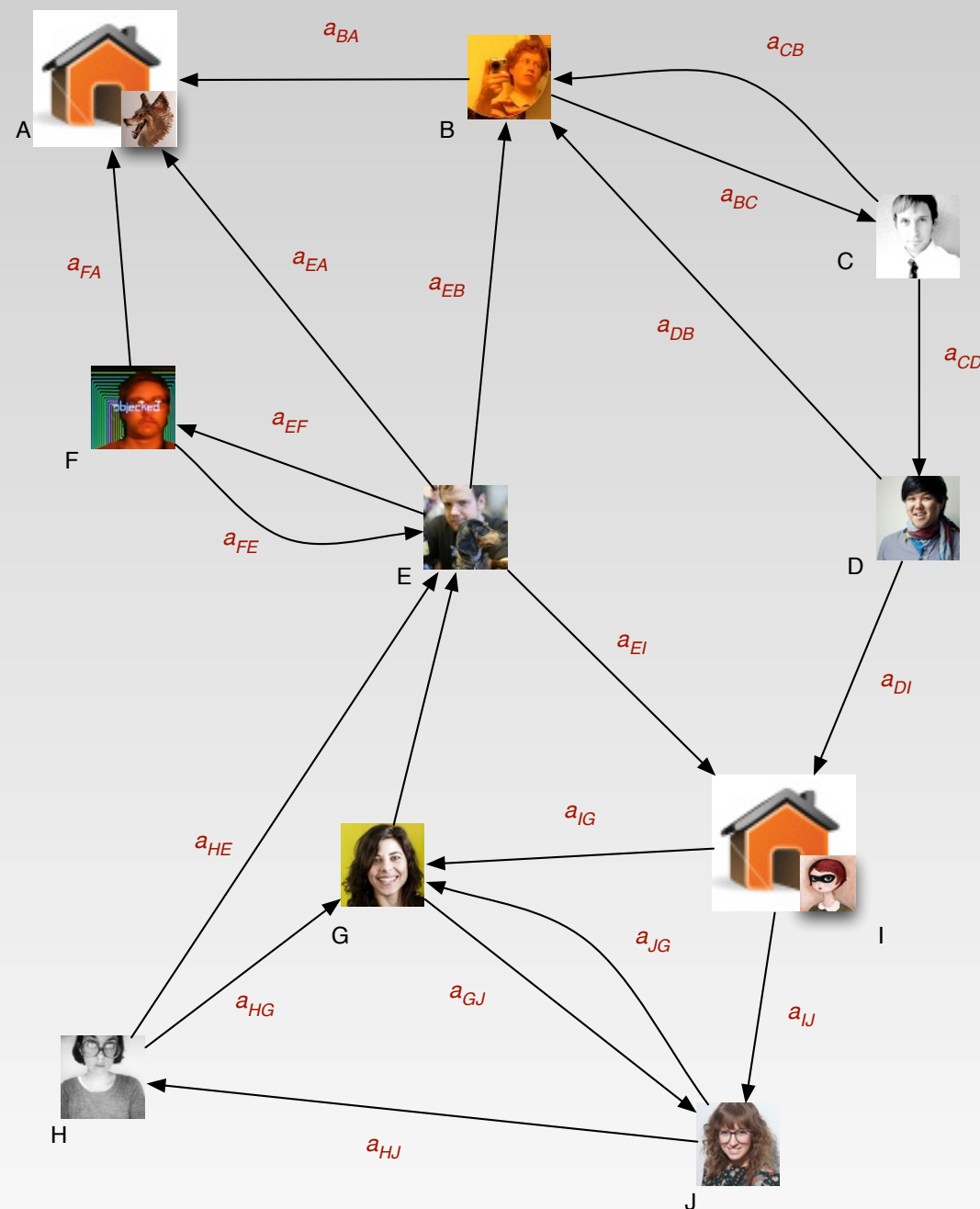
Connections



Friday, January 14, 2011

The edges of the graph are stored in two tables.
For any node, `connection_edges_forward` lists outgoing edges and `connection_edges_reverse` lists the incoming edges.
In other words, we store each edge twice.

Connections



Friday, January 14, 2011

We also assign each edge a weight, known as affinity.

Connections

On *H*'s shard

connection_edges_forward

from	to	affinity
H	E	0.3
H	G	0.7

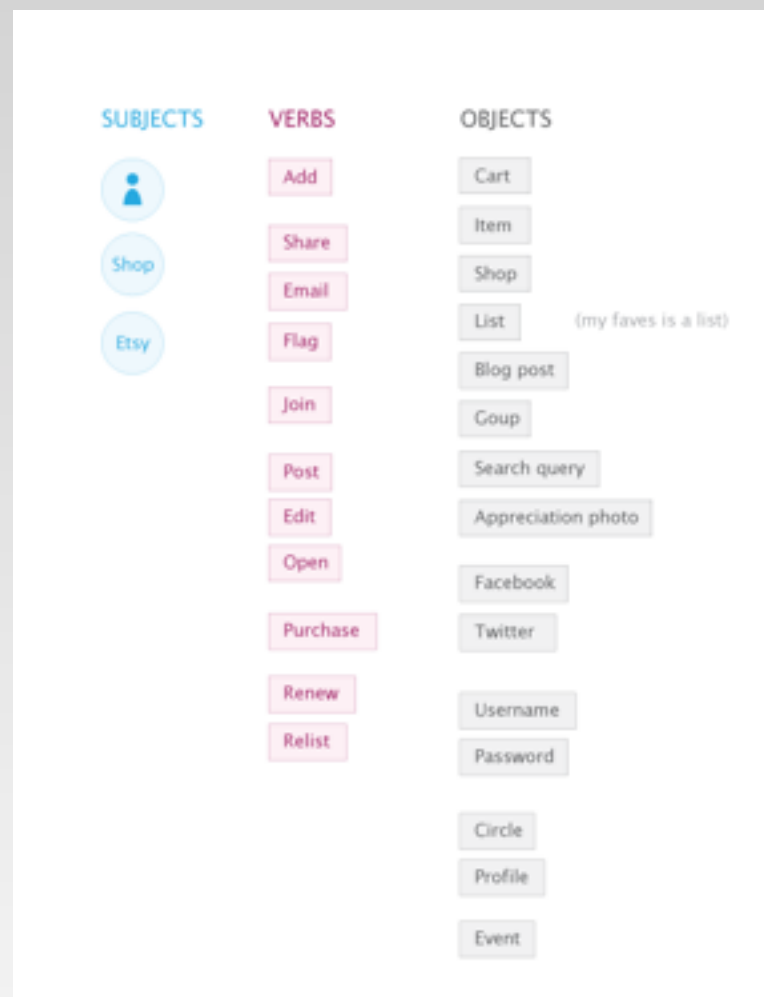
connection_edges_reverse

from	to	affinity
J	H	0.75

Friday, January 14, 2011

Here we see the data for Anda's connections on her shard. She has two entries in the forward connections table for the people in her circle. She has one entry in the reverse connections so that she can see everyone following her.

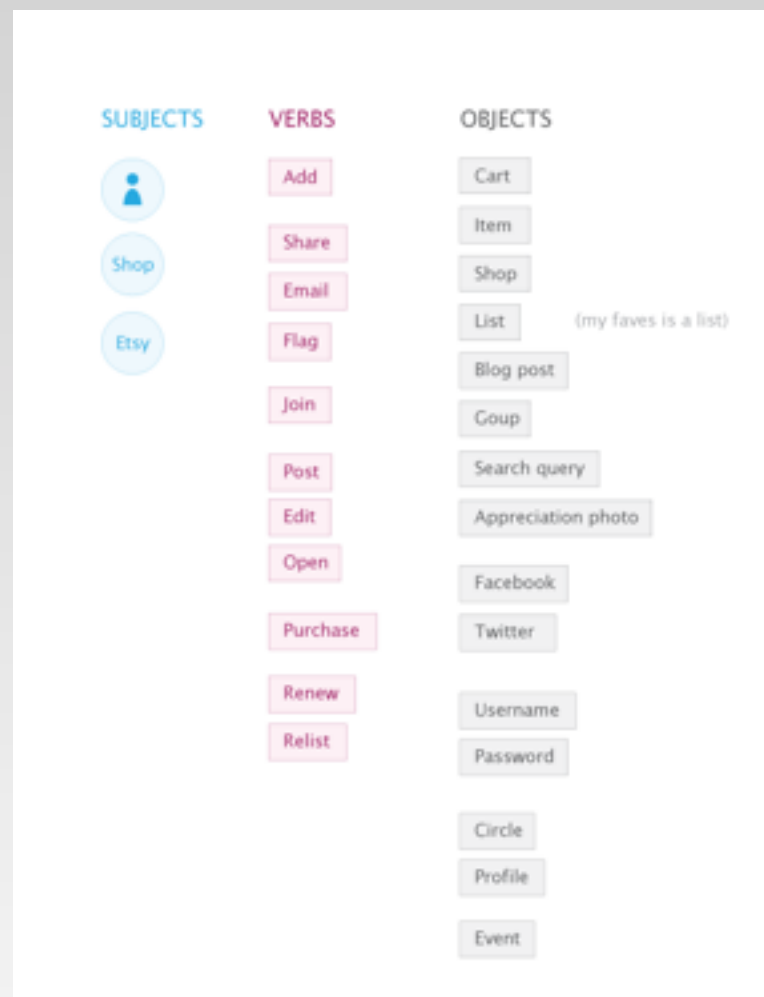
Activities



Friday, January 14, 2011

Activities are the other database entity important to activity feeds.

activity := (subject, verb, object)



Friday, January 14, 2011

As you can see in Rob's magnetic poetry diagram, activities are a description of an event on Etsy boiled down to a subject ("who did it"), a verb ("what they did"), and an object ("what they did it to").

activity := (subject, verb, object)

(Steve, connected, Kyle)

(Kyle, favorited, brief jerky)

Friday, January 14, 2011

Here are some examples of activities.

The first one describes Steve adding Kyle to his circle.

The second one describes Kyle favoriting an item.

In each of these cases note that there are probably several parties interested in these events [examples]. The problem (the main one we're trying to solve with activity feeds) is how to notify all of them about it. In order to achieve that goal, as usual we copy the data all over the place.

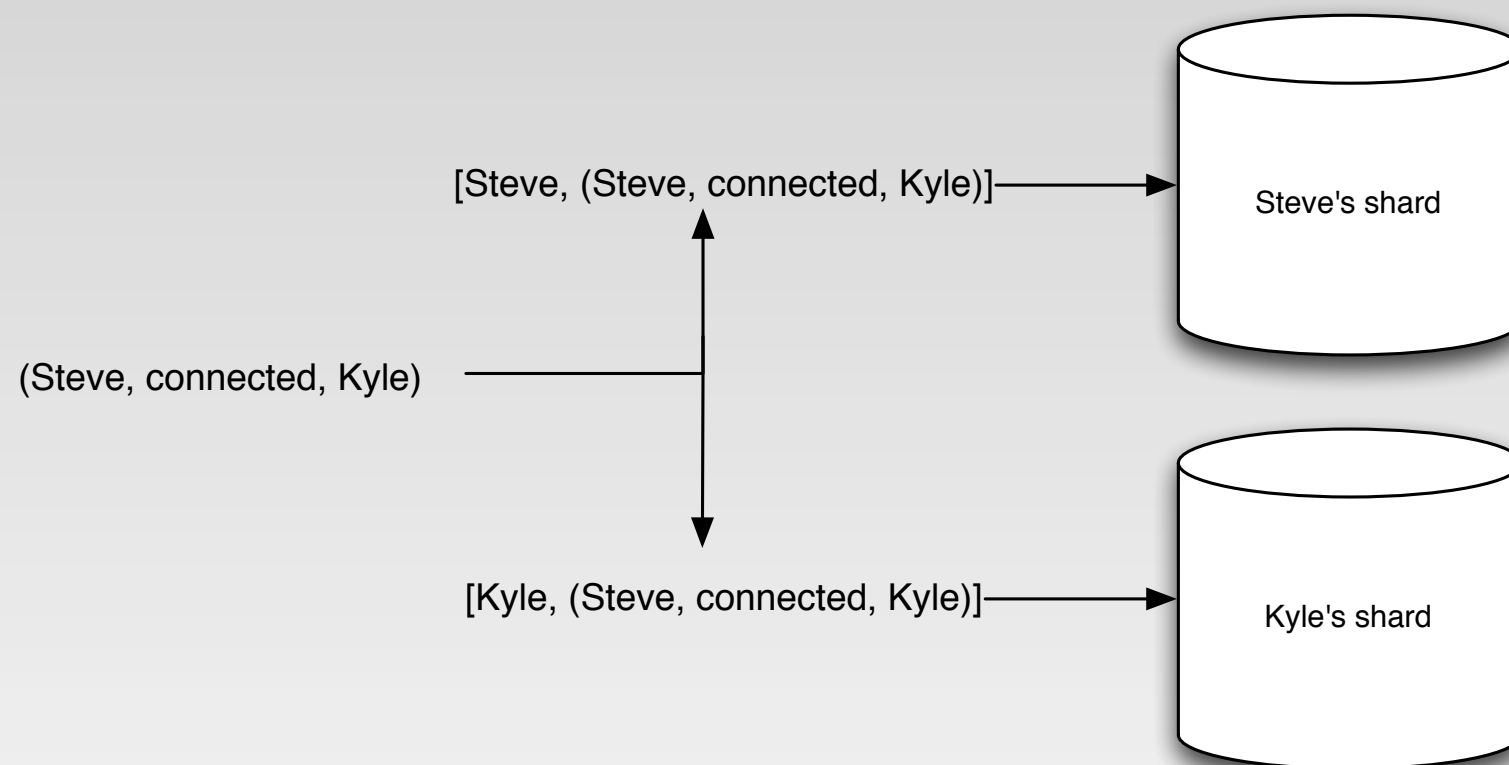
~~activity := (subject, verb, object)~~

activity := [owner, (subject, verb, object)]

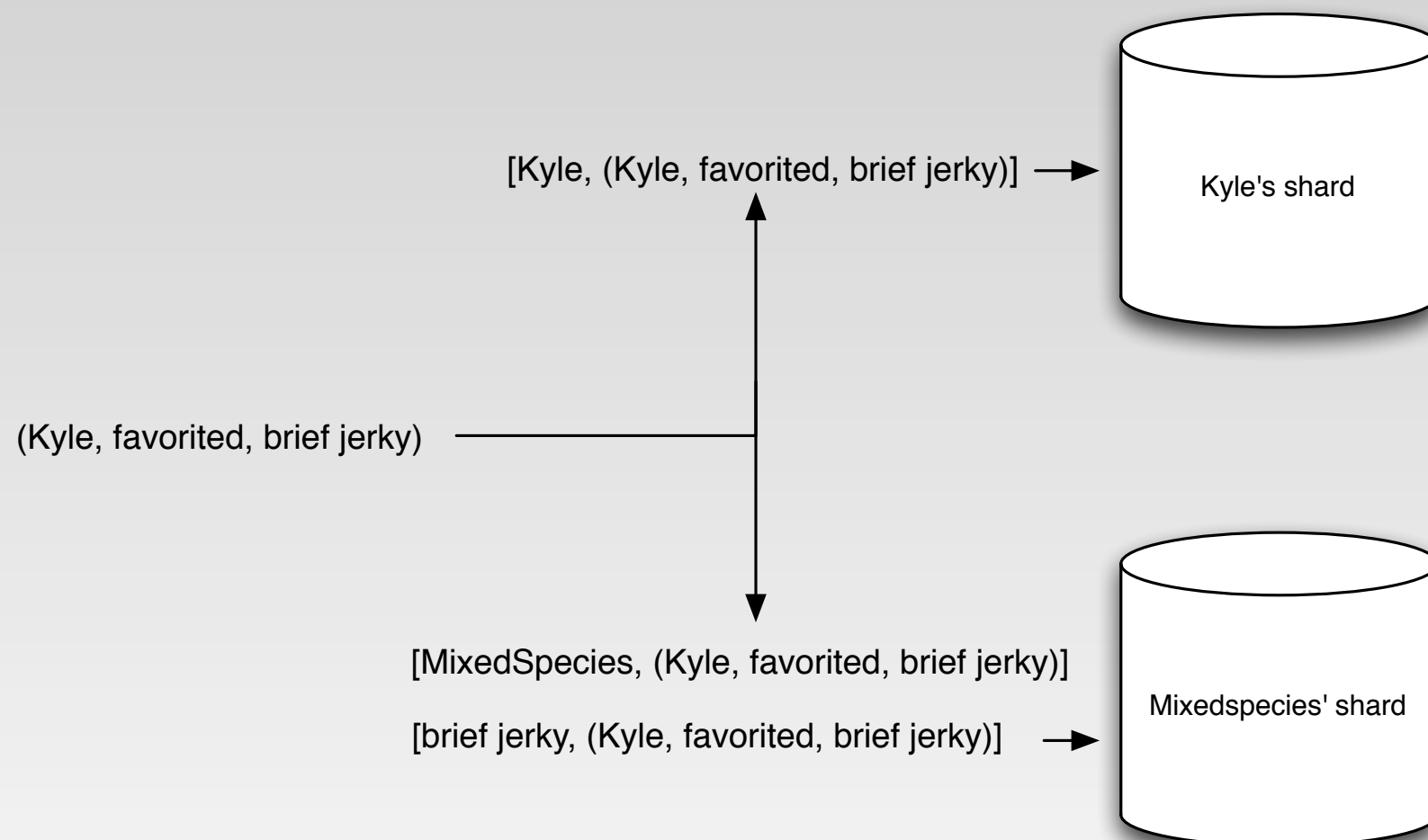
Friday, January 14, 2011

So what we do is duplicate the S,V,O combinations with different owners. Steve will have his record that he connected to Kyle, and Kyle will be given his own record that Steve connected to him.

activity := [owner, (subject, verb, object)]



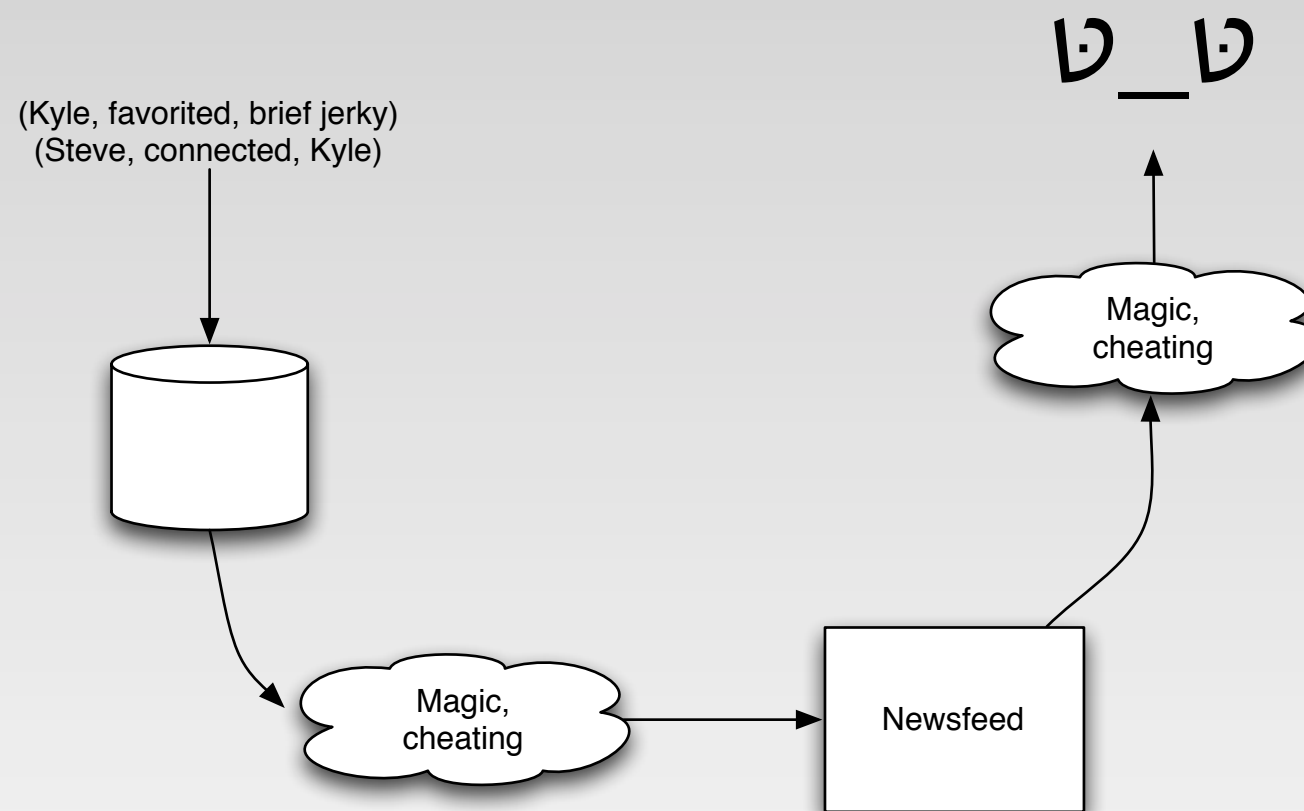
activity := [owner, (subject, verb, object)]



Friday, January 14, 2011

In more complicated examples there could be more than two owners. You could envision people being interested in Kyle, people being interested in MixedSpecies, or people being interested in brief jerky. In cases where there are this many writes, we will generally perform them with Gearman. Again, in order for interested parties to find the activities, we copy the activities all over the place.

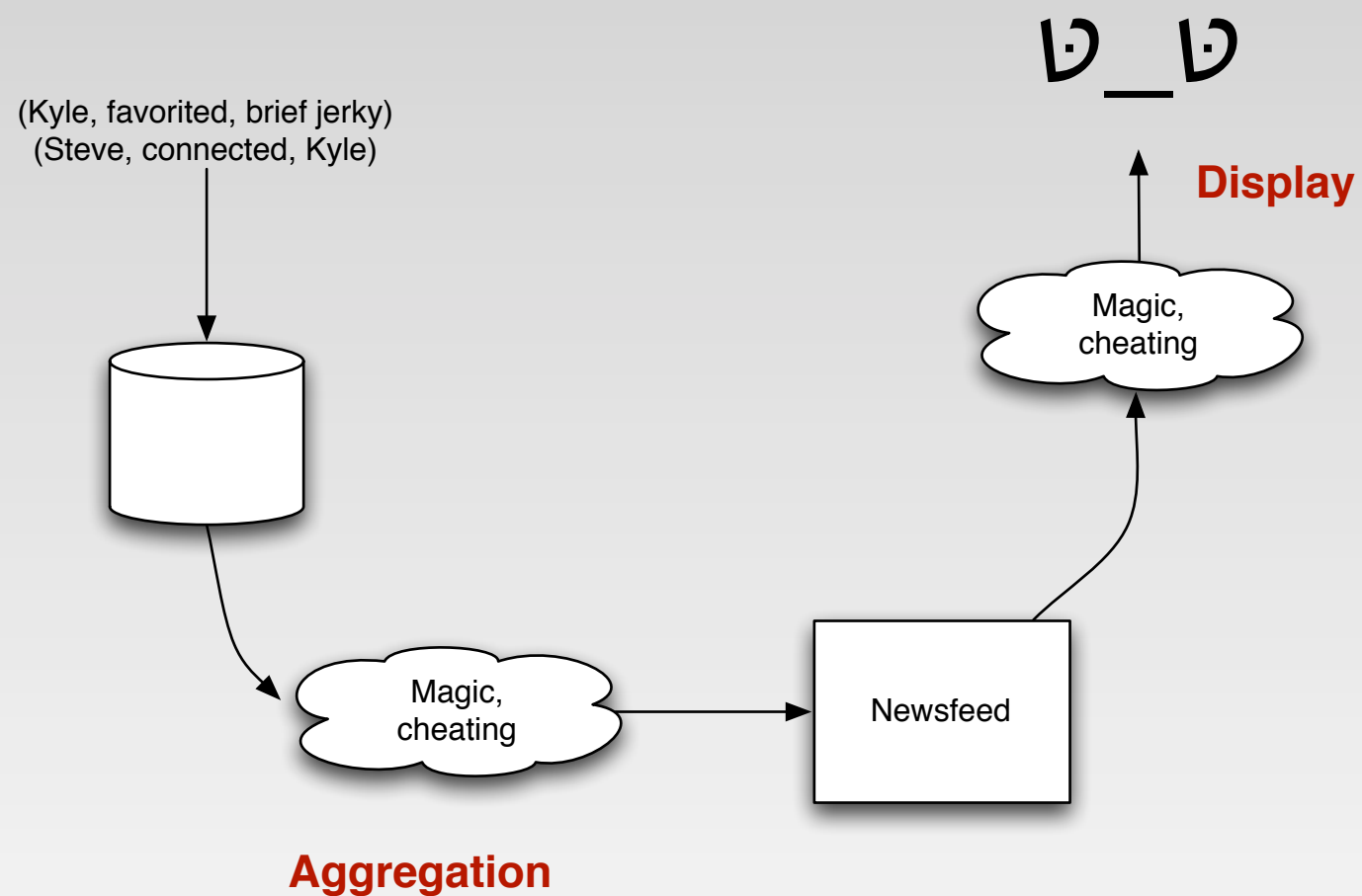
Building a Feed



Friday, January 14, 2011

Now that we know about connections and activities, we can talk about how activities are turned into Newsfeeds and how those wind up being displayed to end users.

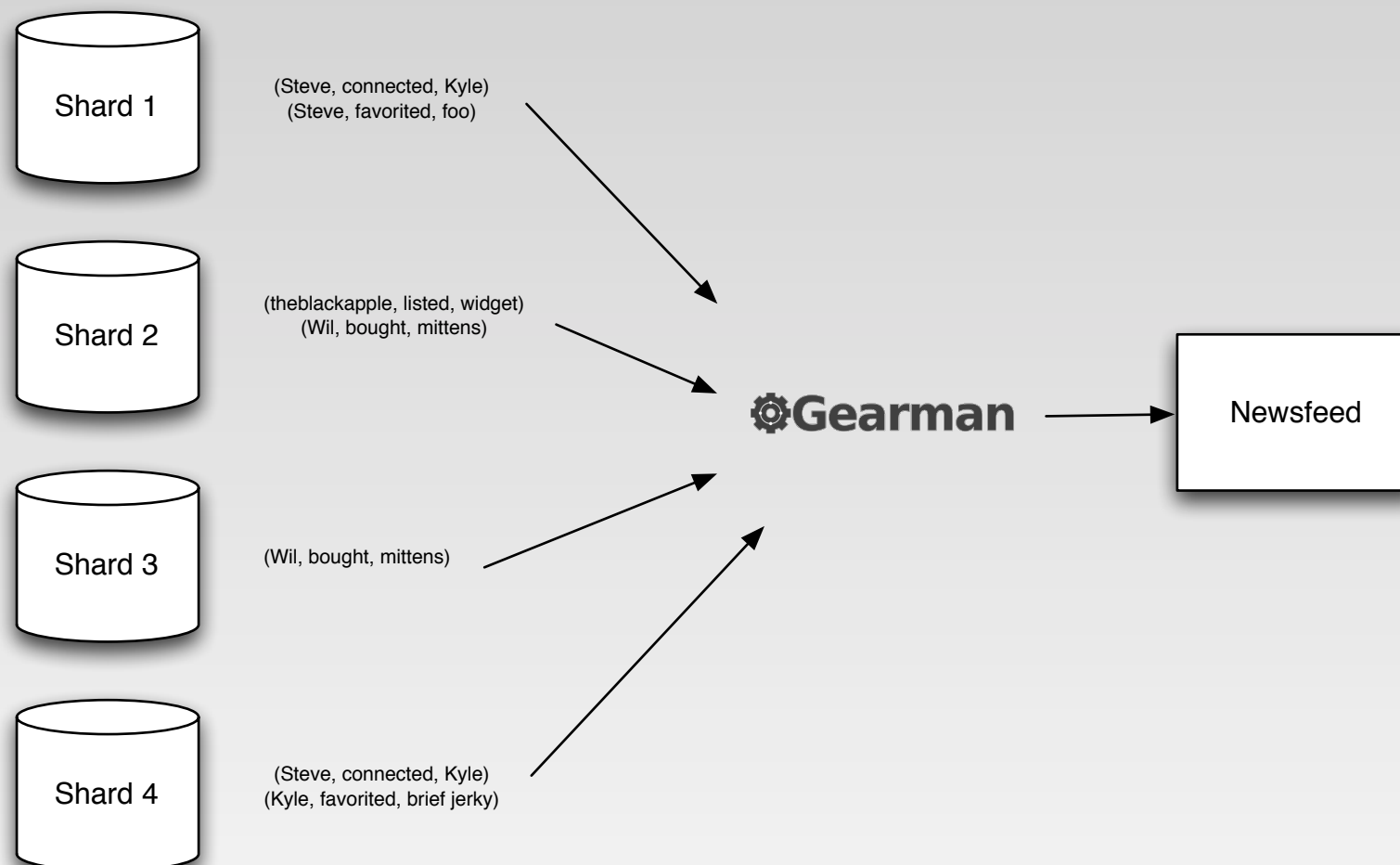
Building a Feed



Friday, January 14, 2011

Getting to the end result (the activity feed page) has two distinct phases: aggregation and display.

Aggregation



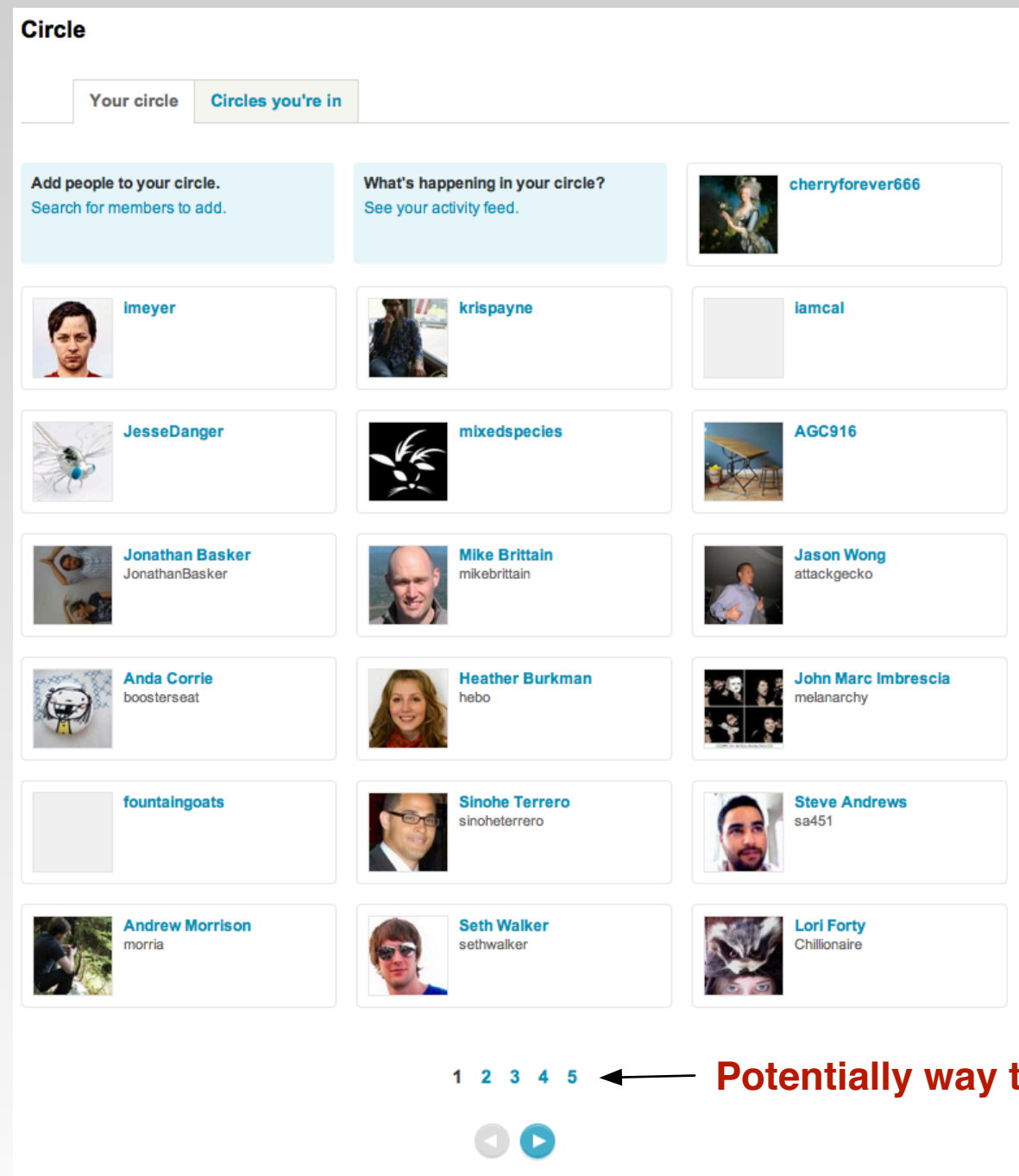
Friday, January 14, 2011

I am going to talk about aggregation first.

Aggregation turns activities (in the database) into a Newsfeed (in memcache).

Aggregation typically occurs offline, with Gearman.

Aggregation, Step 1: Choosing Connections

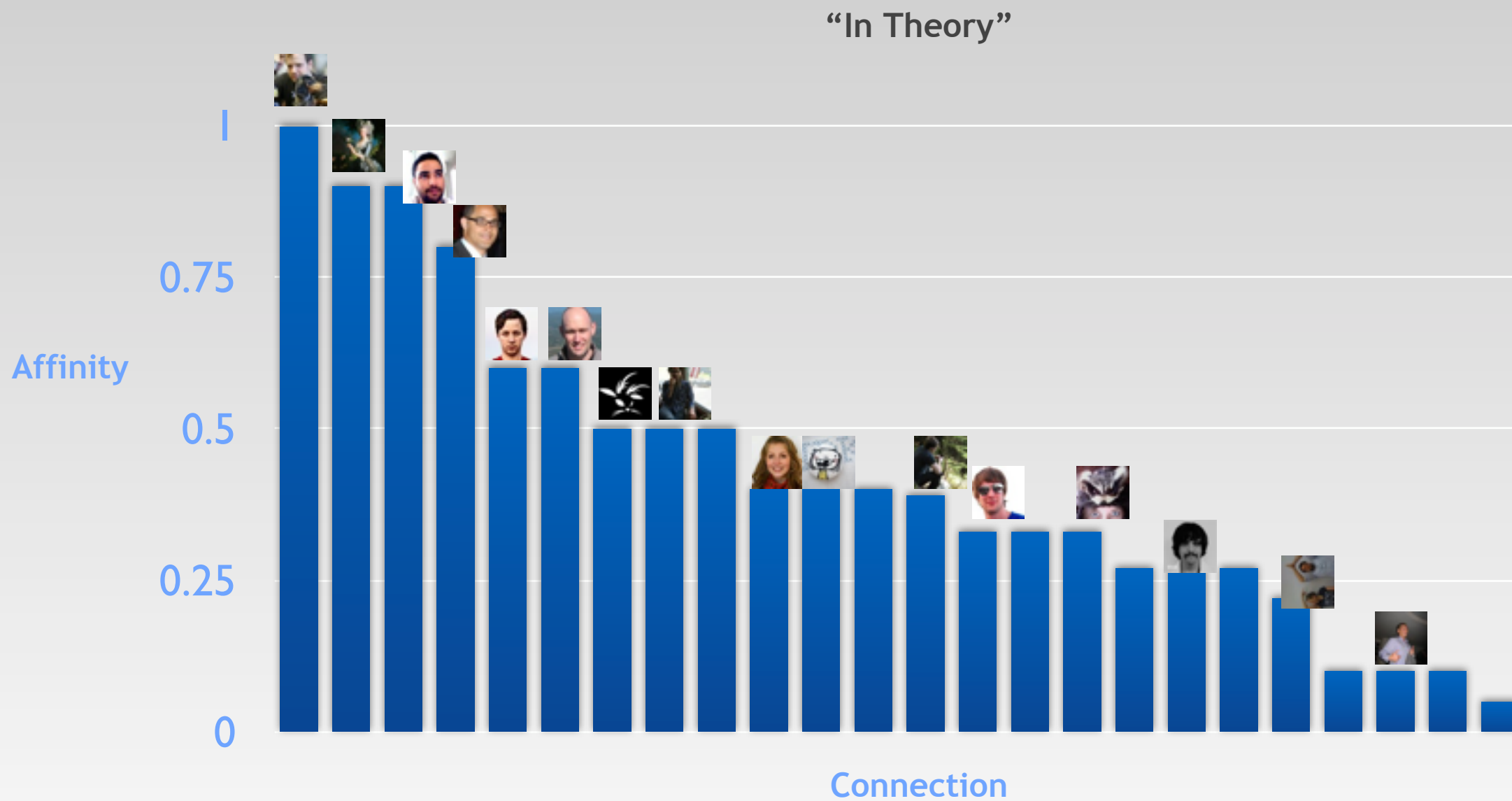


Friday, January 14, 2011

We already allow people to have more connections than would make sense on a single feed, or could be practically aggregated all at once.

The first step in aggregation is to turn the list of people you are connected to into the list of people we're actually going to go seek out activities for.

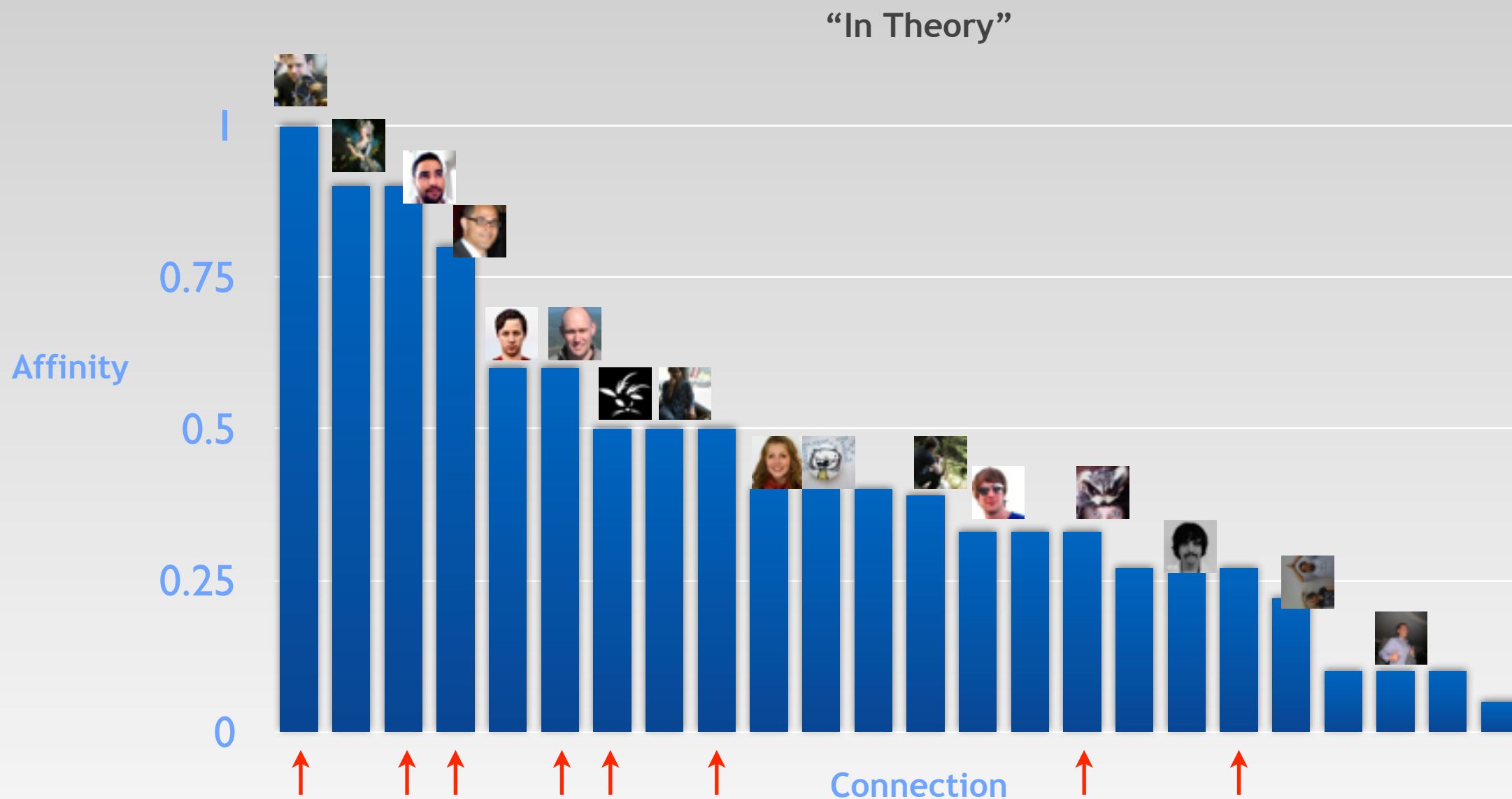
Aggregation, Step 1: Choosing Connections



Friday, January 14, 2011

In theory, the way we would do this is rank the connections by affinity and then treat the affinity as the probability that we'll pick it.

Aggregation, Step 1: Choosing Connections



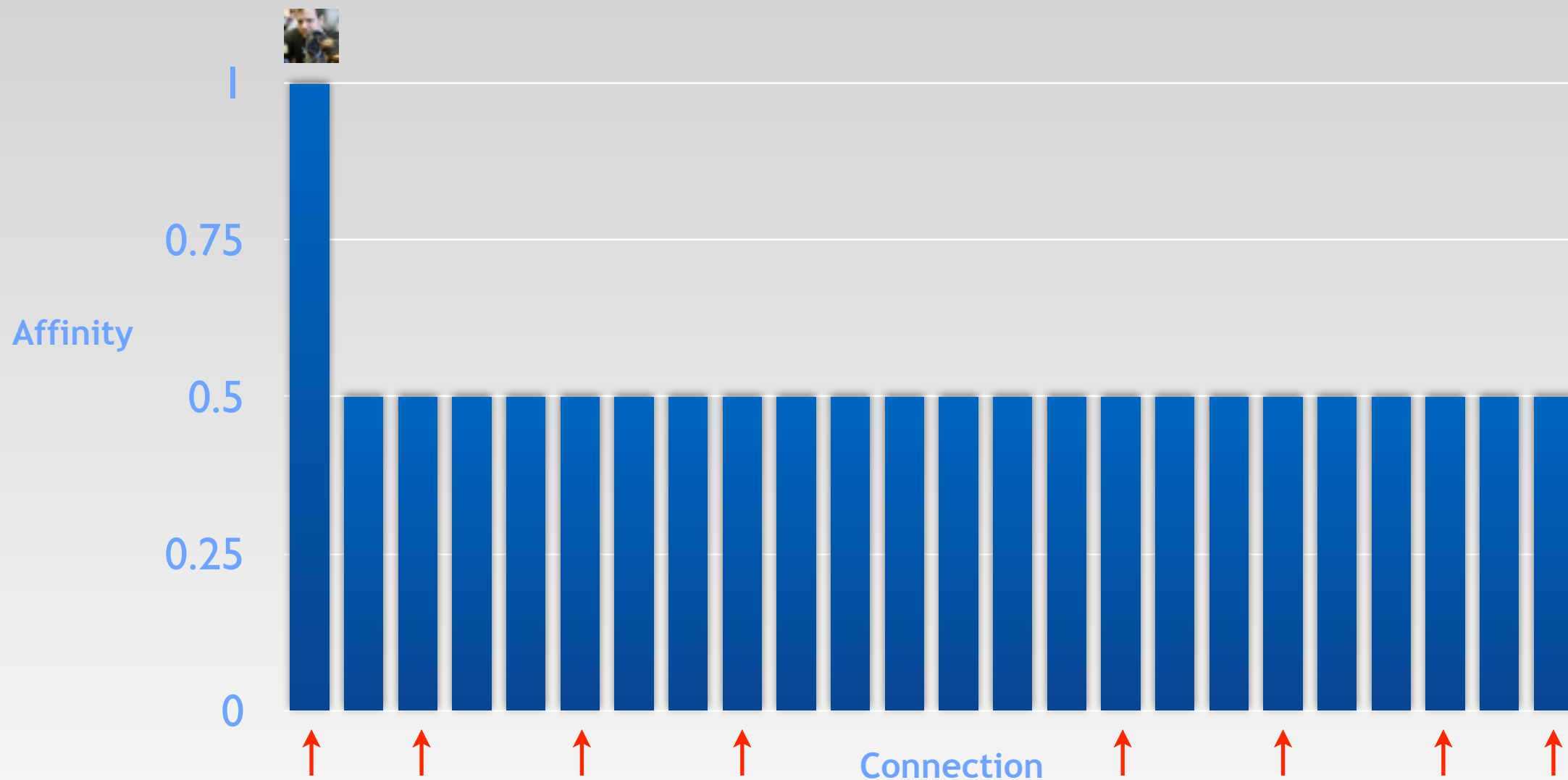
`$choose_connection = mt_rand() < $affinity;`

Friday, January 14, 2011

So then we'd be more likely to pick the close connections, but leaving the possibility that we will pick the distant ones.

Aggregation, Step 1: Choosing Connections

“In Practice”

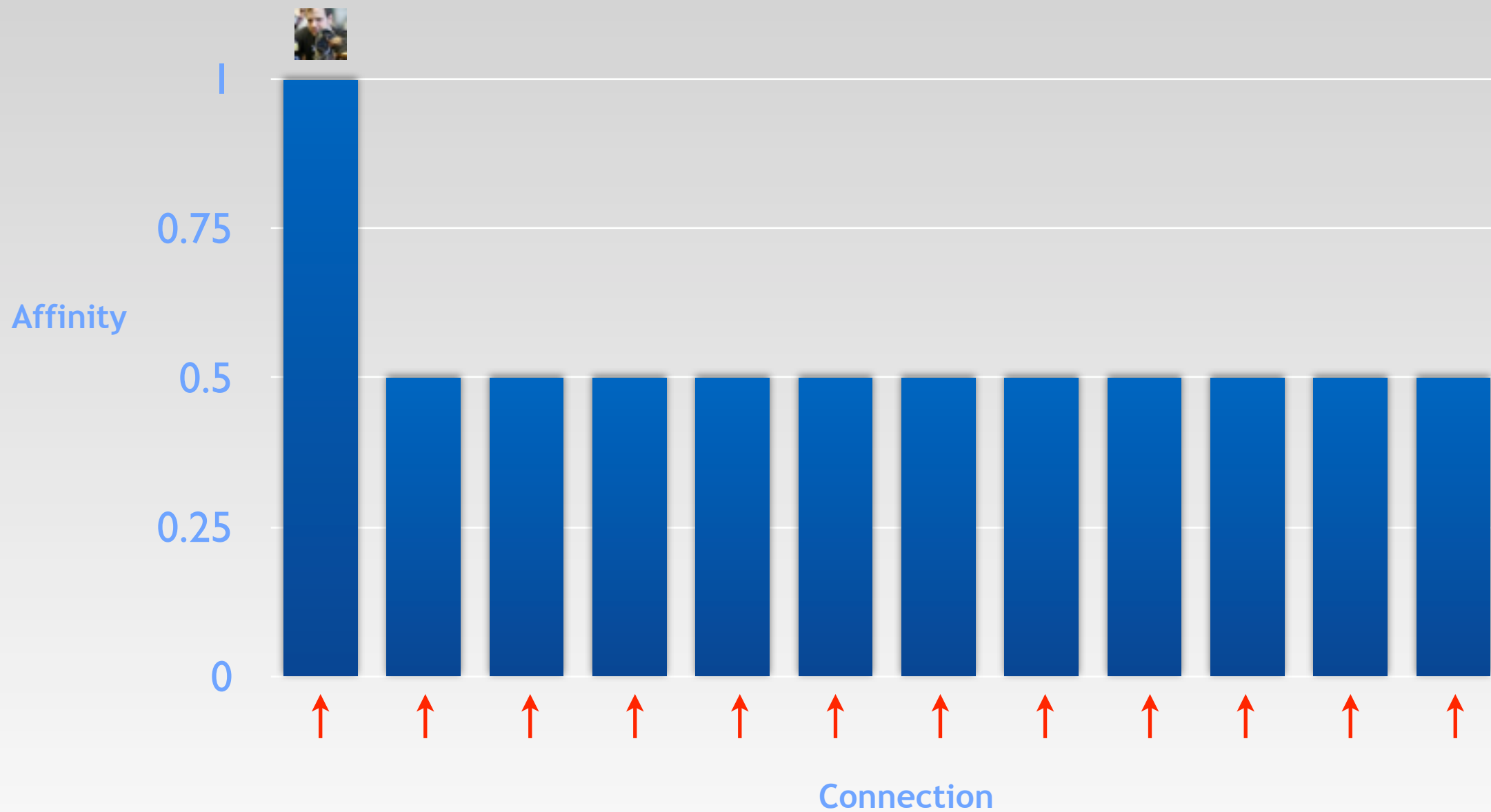


Friday, January 14, 2011

In practice we don't really handle affinity yet.

Aggregation, Step 1: Choosing Connections

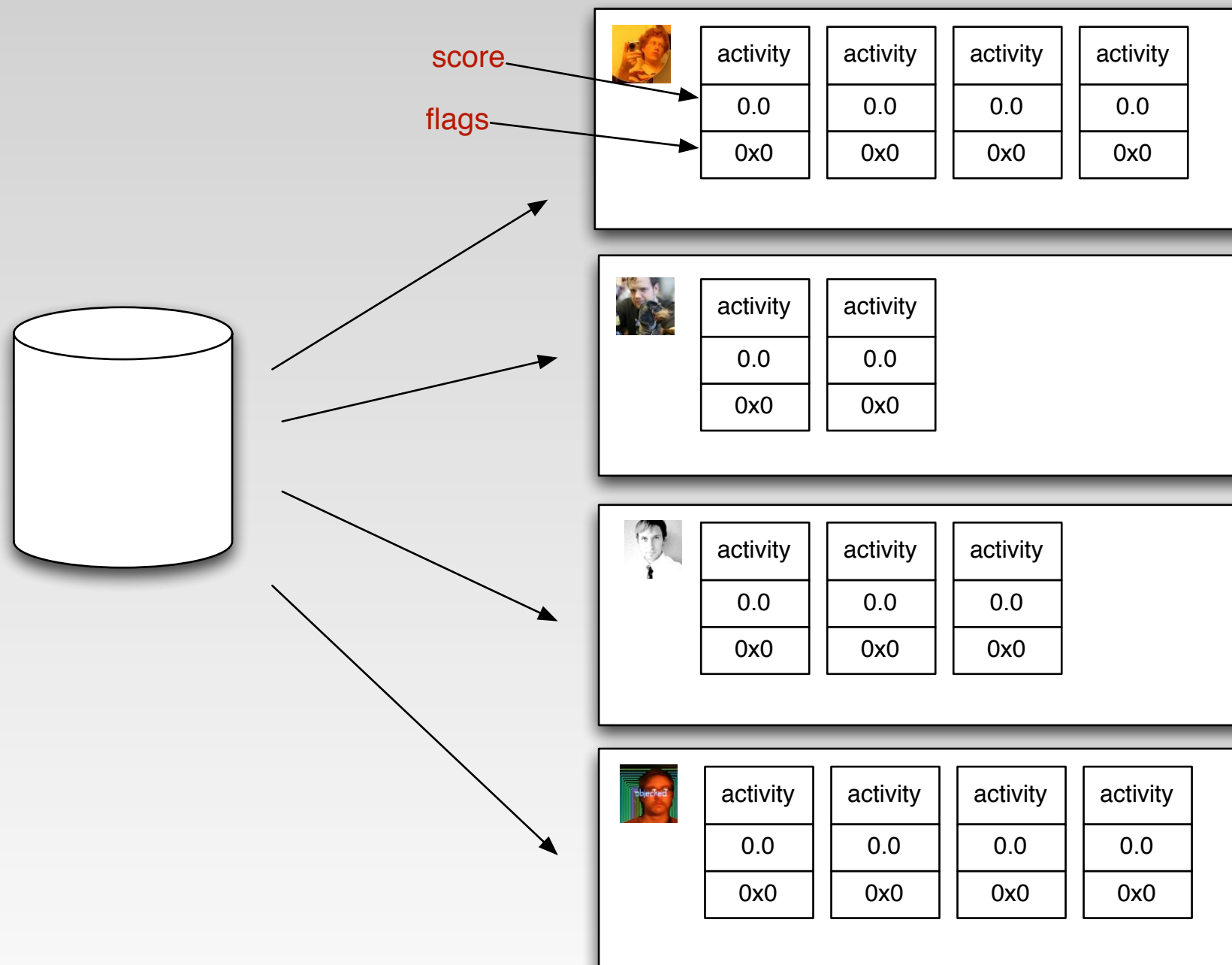
“Even More In Practice”



Friday, January 14, 2011

And most people don't currently have enough connections for this to matter at all. (Mean connections is around a dozen.)

Aggregation, Step 2: Making Activity Sets




Friday, January 14, 2011


Once the connections are chosen, we then select historical activity for them and convert them into in-memory structures called activity sets.

These are just the activities grouped by connection, with a score and flags field for each. The next few phases of aggregation operate on these.


Aggregation, Step 3: Classification




activity	activity	activity	activity
0.0	0.0	0.0	0.0
0x11	0x3	0x20c1	0x10004



activity	activity
0.0	0.0
0x20c1	0x4



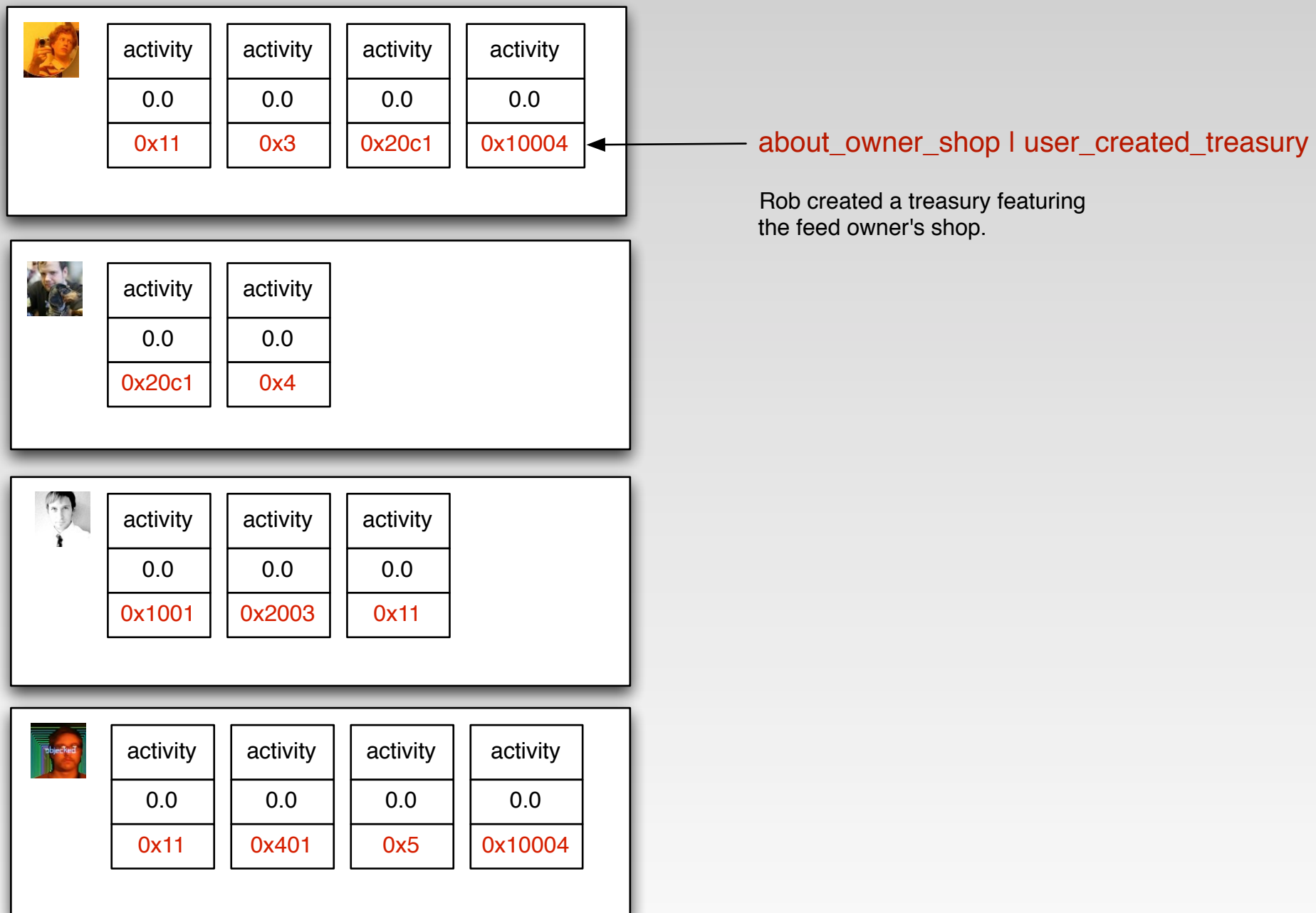
activity	activity	activity
0.0	0.0	0.0
0x1001	0x2003	0x11



activity	activity	activity	activity
0.0	0.0	0.0	0.0
0x11	0x401	0x5	0x10004

The next thing that happens is that we iterate through all of the activities in all of the sets and classify them.

Aggregation, Step 3: Classification



Friday, January 14, 2011

The flags are a bit field.
They are all from the point of view of the feed owner.
So the same activities on another person's feed would be assigned different flags.

Aggregation, Step 4: Scoring

activity
0.8
0x11

activity
0.77
0x3

activity
0.9
0x20c1

activity
0.1
0x10004

activity
0.6
0x20c1

activity
0.47
0x4

activity
0.8
0x1001

activity
0.55
0x2003

activity
0.8
0x11

activity
0.3
0x11

activity
0.25
0x401

activity
0.74
0x5

activity
0.9
0x10004

Friday, January 14, 2011

Next we fill in the score fields.

At this point the score is just a simple time decay function (older activities always score lower than new ones).

Aggregation, Step 5: Pruning



Friday, January 14, 2011

As we noted before it's possible to wind up seeing the same event as two or more activities. The next stage of aggregation detects these situations.


Aggregation, Step 5: Pruning





Friday, January 14, 2011


We iterate through the activity sets and remove the duplicates. Right now we can just cross off the second instance of the SVO pair; once we have comments this will be more complicated.

Aggregation, Step 6: Sort & Merge

	activity	activity	activity	activity
	0.8	0.77	0.9	0.1
	0x11	0x3	0x20c1	0x10004

	activity	activity		
	0.6	0.47		
	0x20c1	0x4		

	activity	activity		
	0.8	0.55		
	0x1001	0x2003		

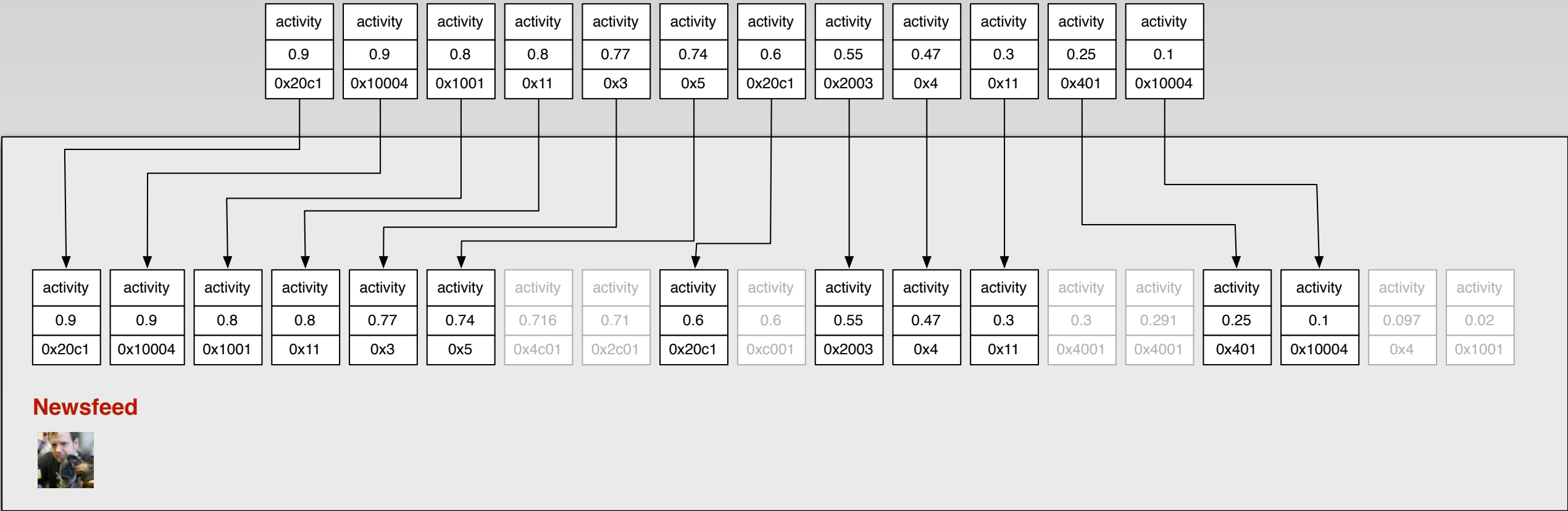
	activity	activity	activity	activity
	0.3	0.25	0.74	0.9
	0x11	0x401	0x5	0x10004



activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity
0.9	0.9	0.8	0.8	0.77	0.74	0.6	0.55	0.47	0.3	0.25	0.1
0x20c1	0x10004	0x1001	0x11	0x3	0x5	0x20c1	0x2003	0x4	0x11	0x401	0x10004

Once everything is scored, classified, and de-duped we can flatten the whole thing and sort it by score.

Aggregation, Step 6: Sort & Merge



Then we take the final set of activities and merge it on to the owner’s existing newsfeed. (Or we create a new newsfeed if they don’t have one.)


Aggregation: Cleaning Up

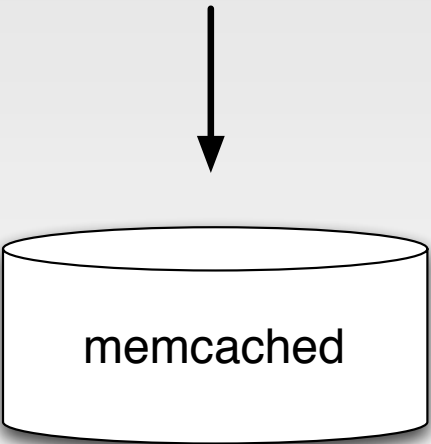


Aggregation: Cleaning Up

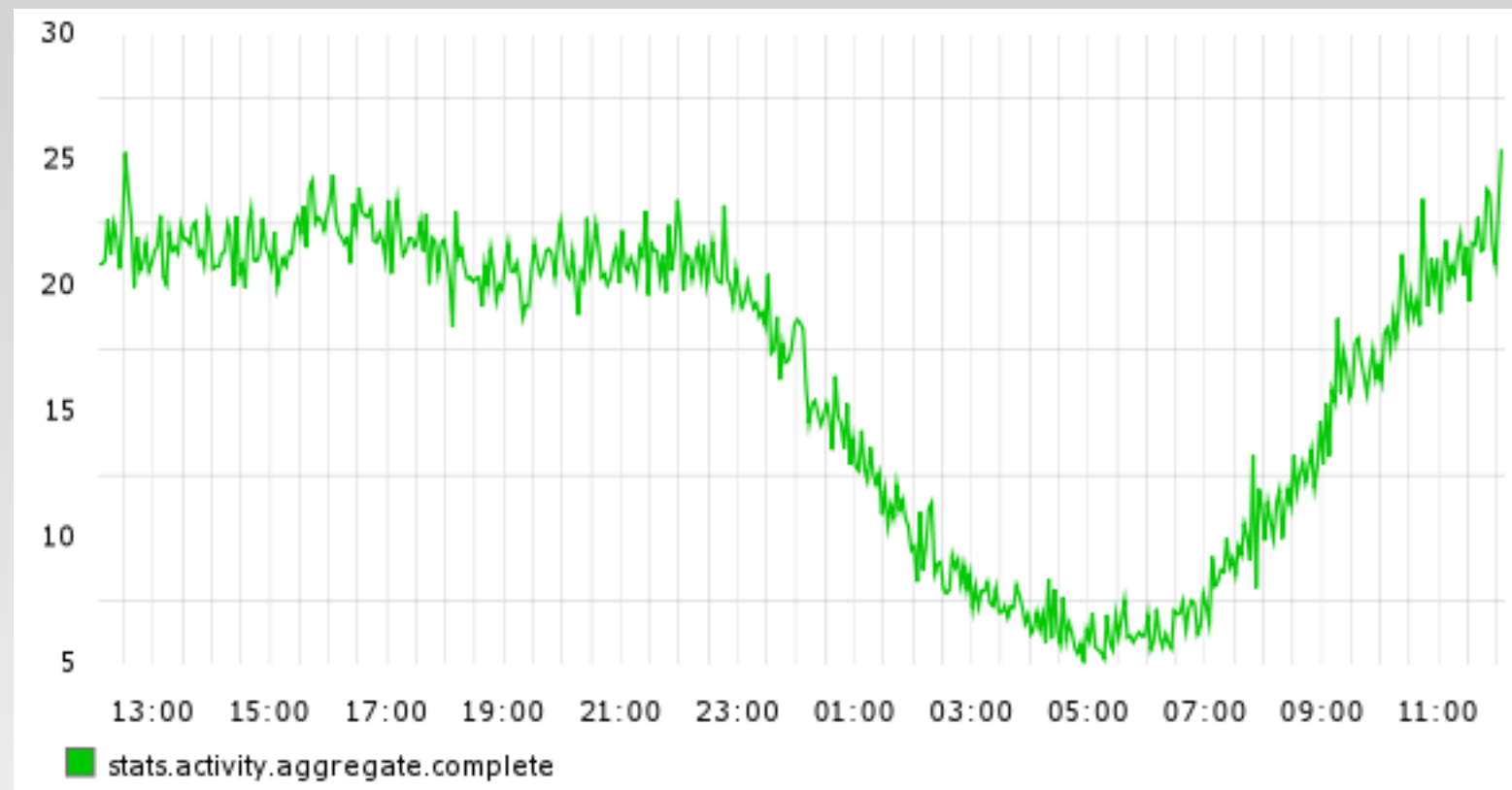
activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity	activity
0.9	0.9	0.8	0.8	0.77	0.74	0.716	0.71	0.6	0.6	0.55	0.47	0.3	0.3
0x20c1	0x10004	0x1001	0x11	0x3	0x5	0x4c01	0x2c01	0x20c1	0xc001	0x2003	0x4	0x11	0x4001

Newsfeed





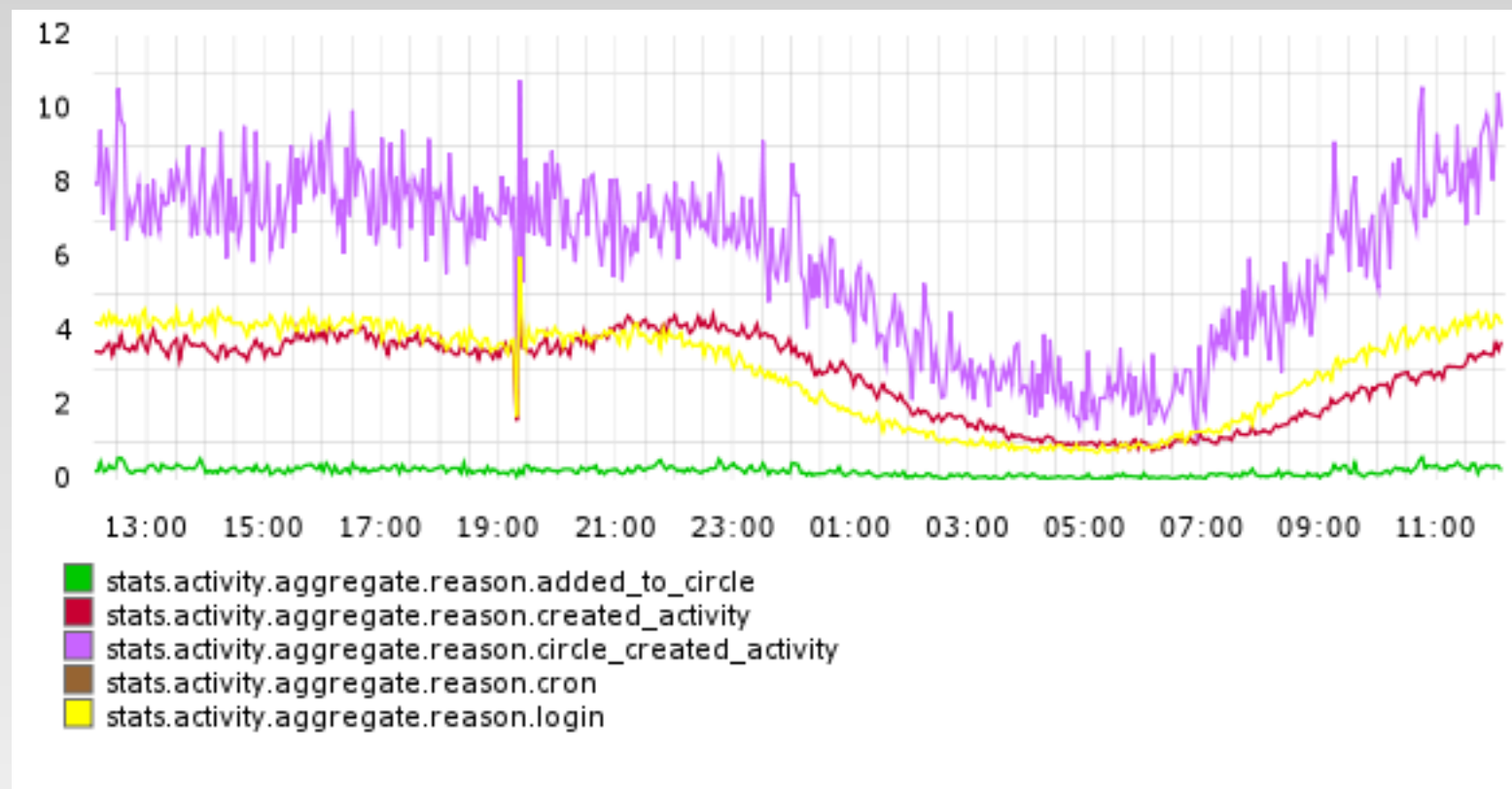
Aggregation



Friday, January 14, 2011

Currently we peak at doing this about 25 times per second.

Aggregation

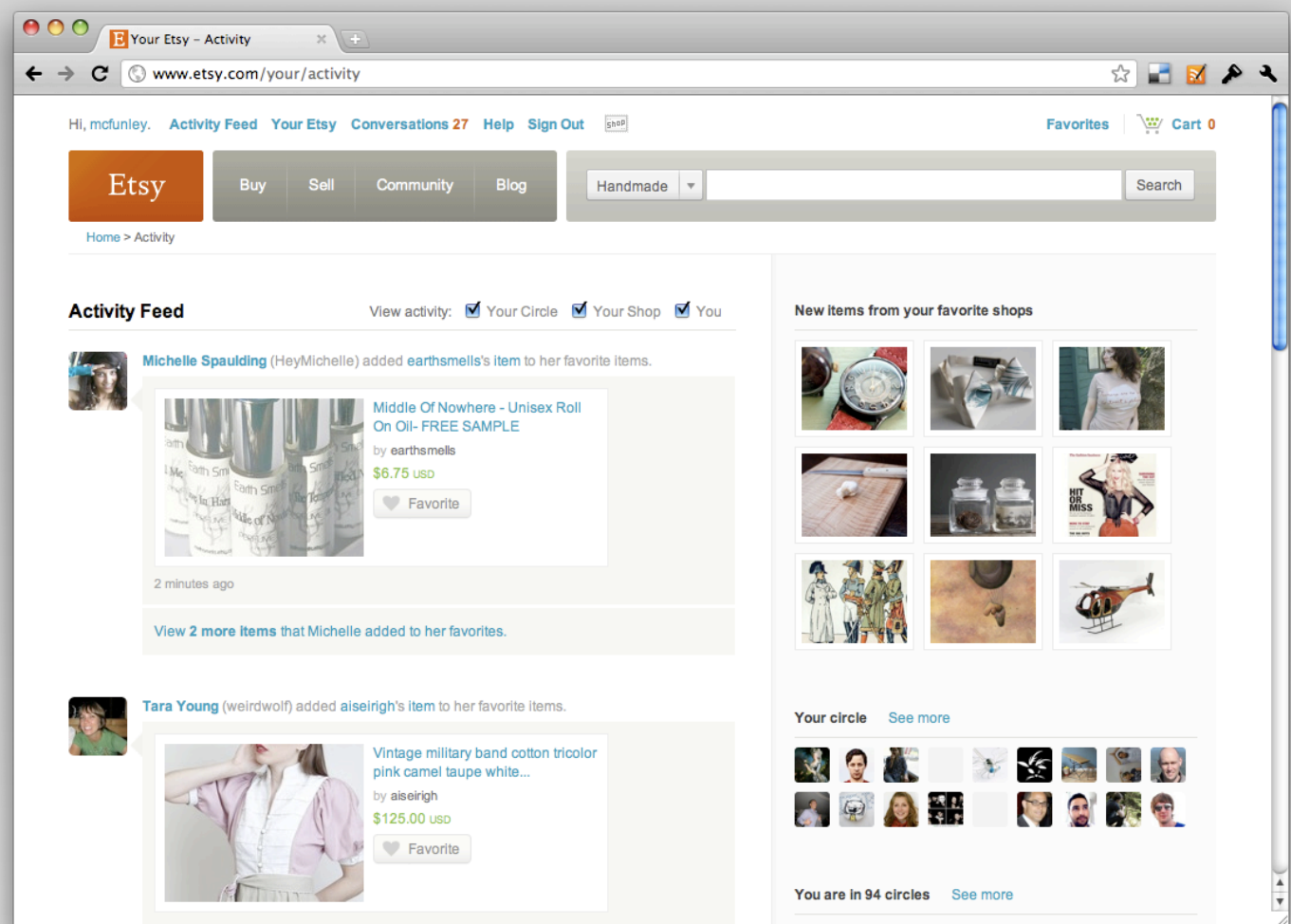
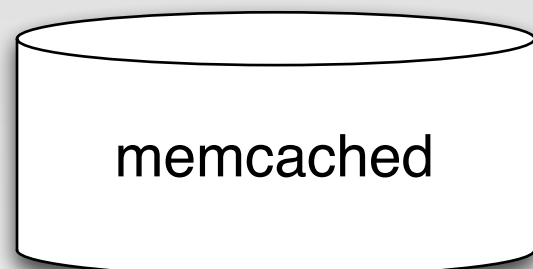


Friday, January 14, 2011

We do it for a lot of different reasons:

- The feed owner has done something, or logged in.
- On a schedule, with cron.
- We also aggregate for your connections when you do something (purple). This is a hack and won't scale.

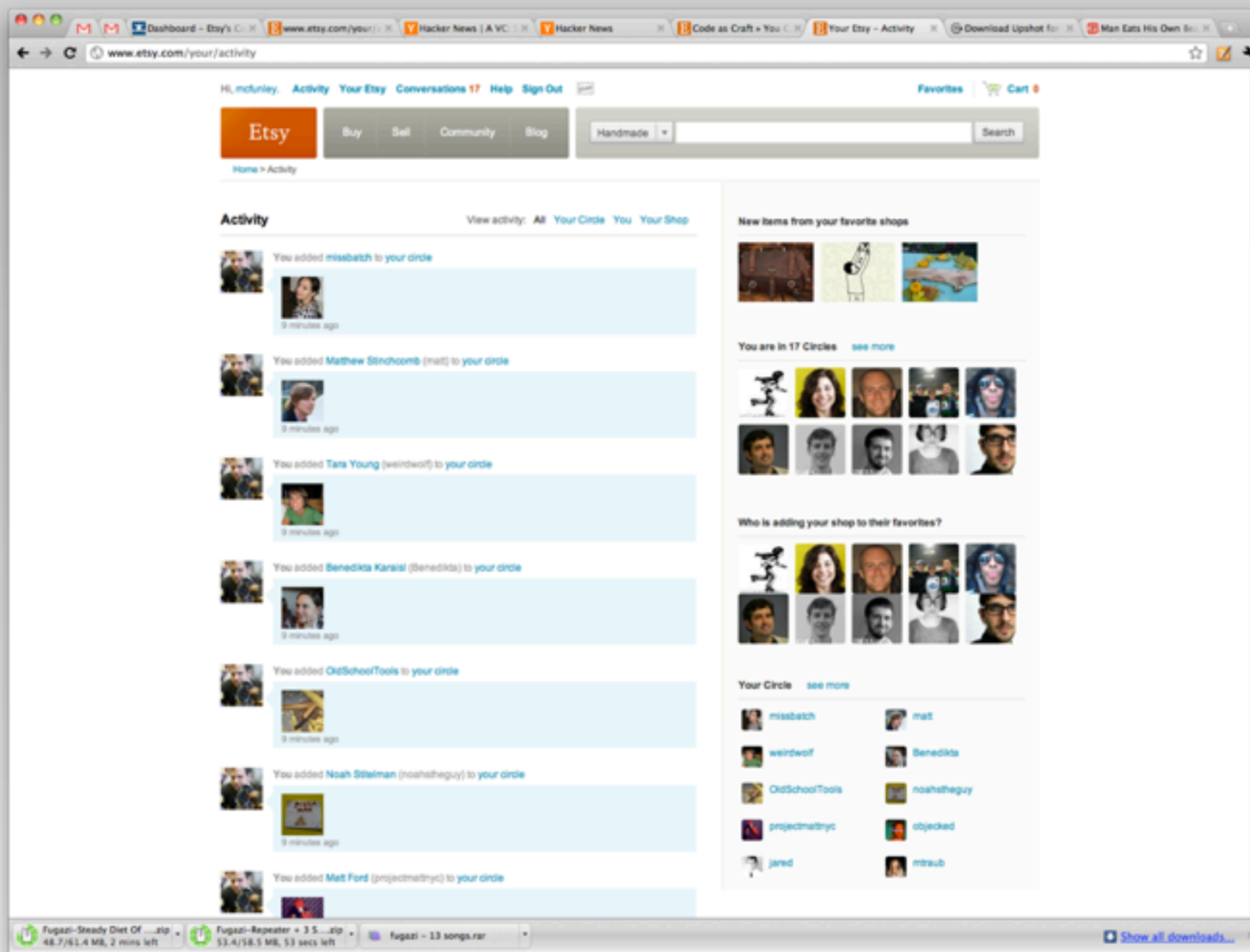
Display



Friday, January 14, 2011

Next I'm going to talk about how we get from the memcached newsfeed to the final product.

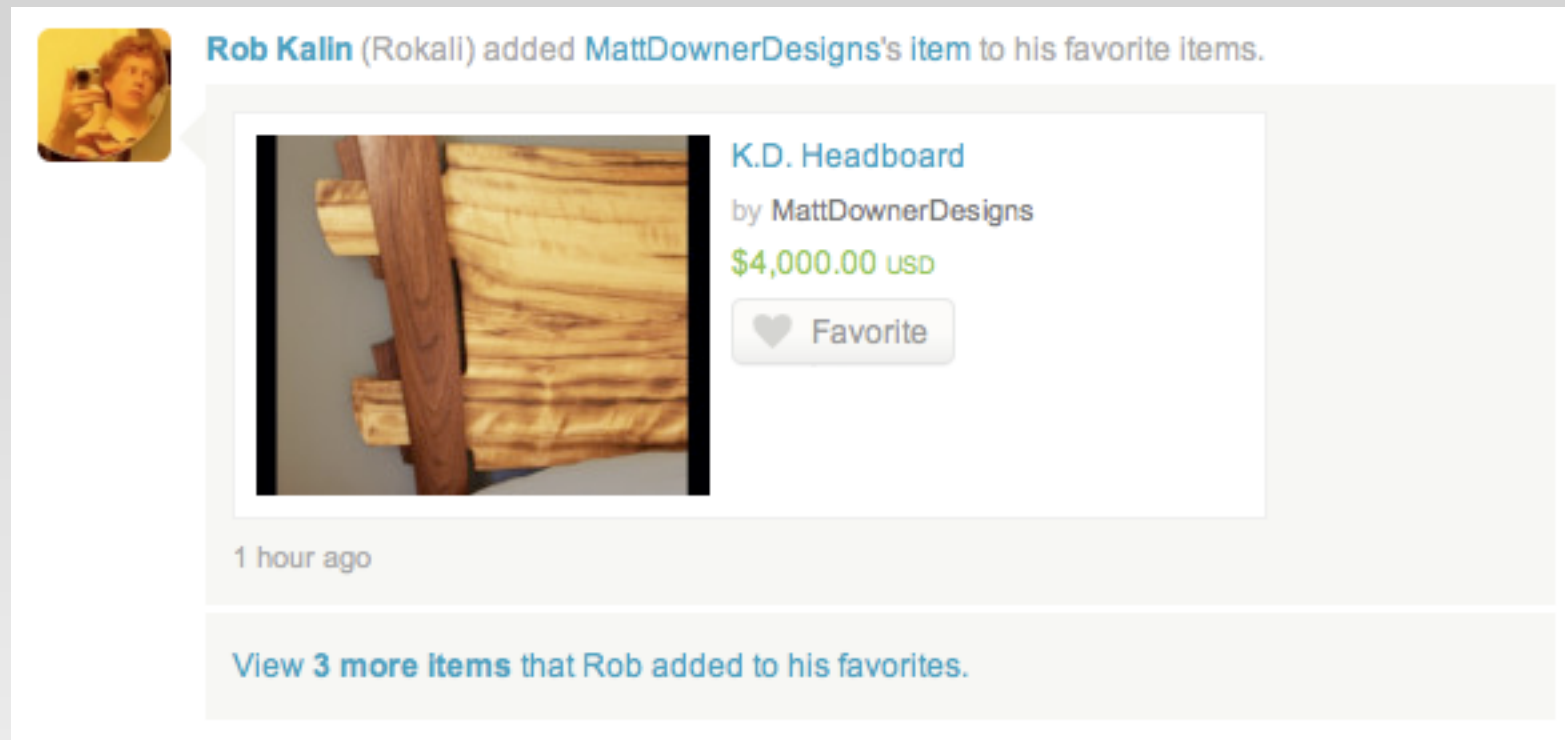
Display: done naively, sucks



Friday, January 14, 2011

If we just showed the activities in the order that they're in on the newsfeed, it would look like this.

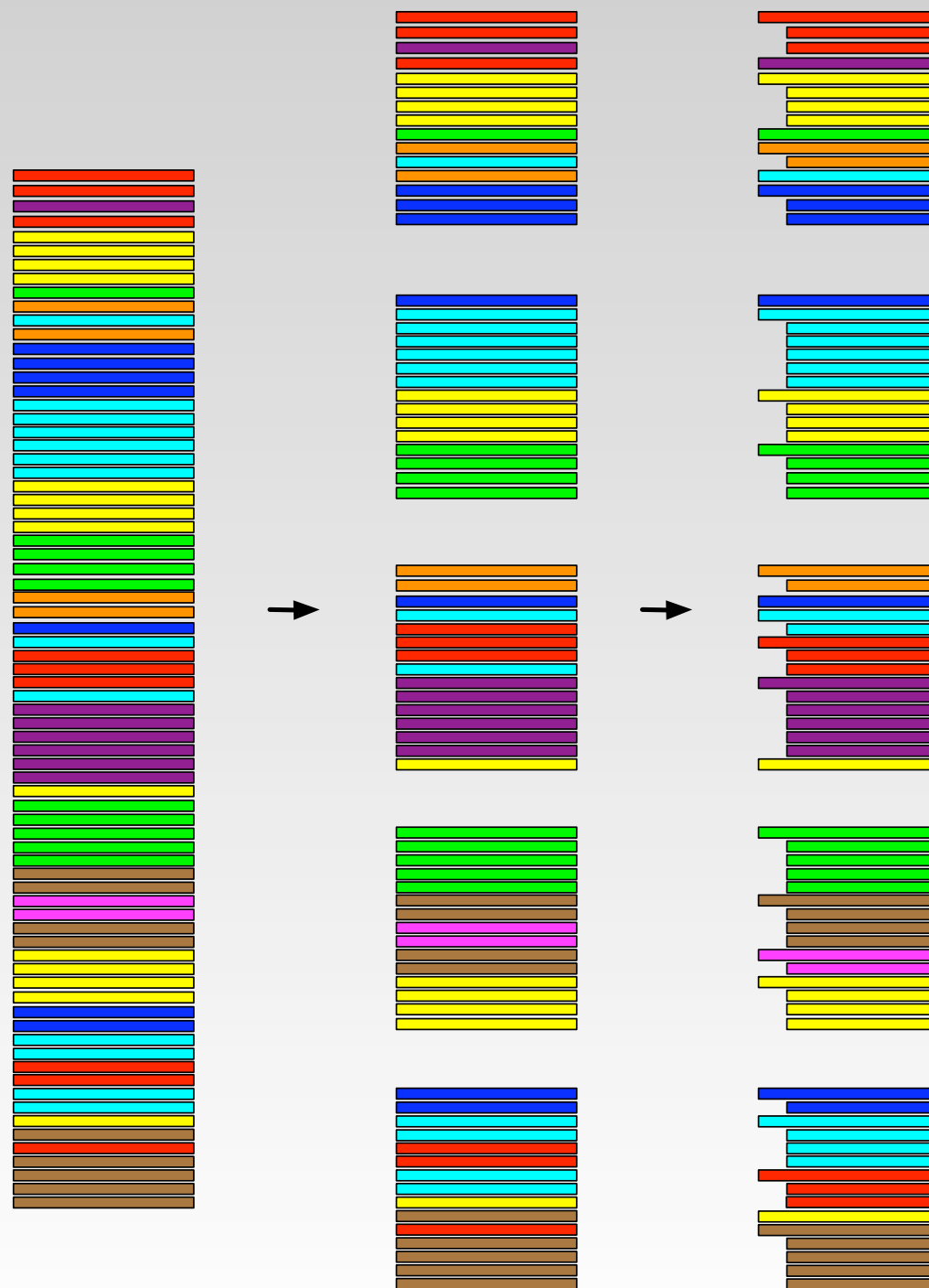
Display: Enter Rollups



Friday, January 14, 2011

To solve this problem we combine similar stories into rollups.

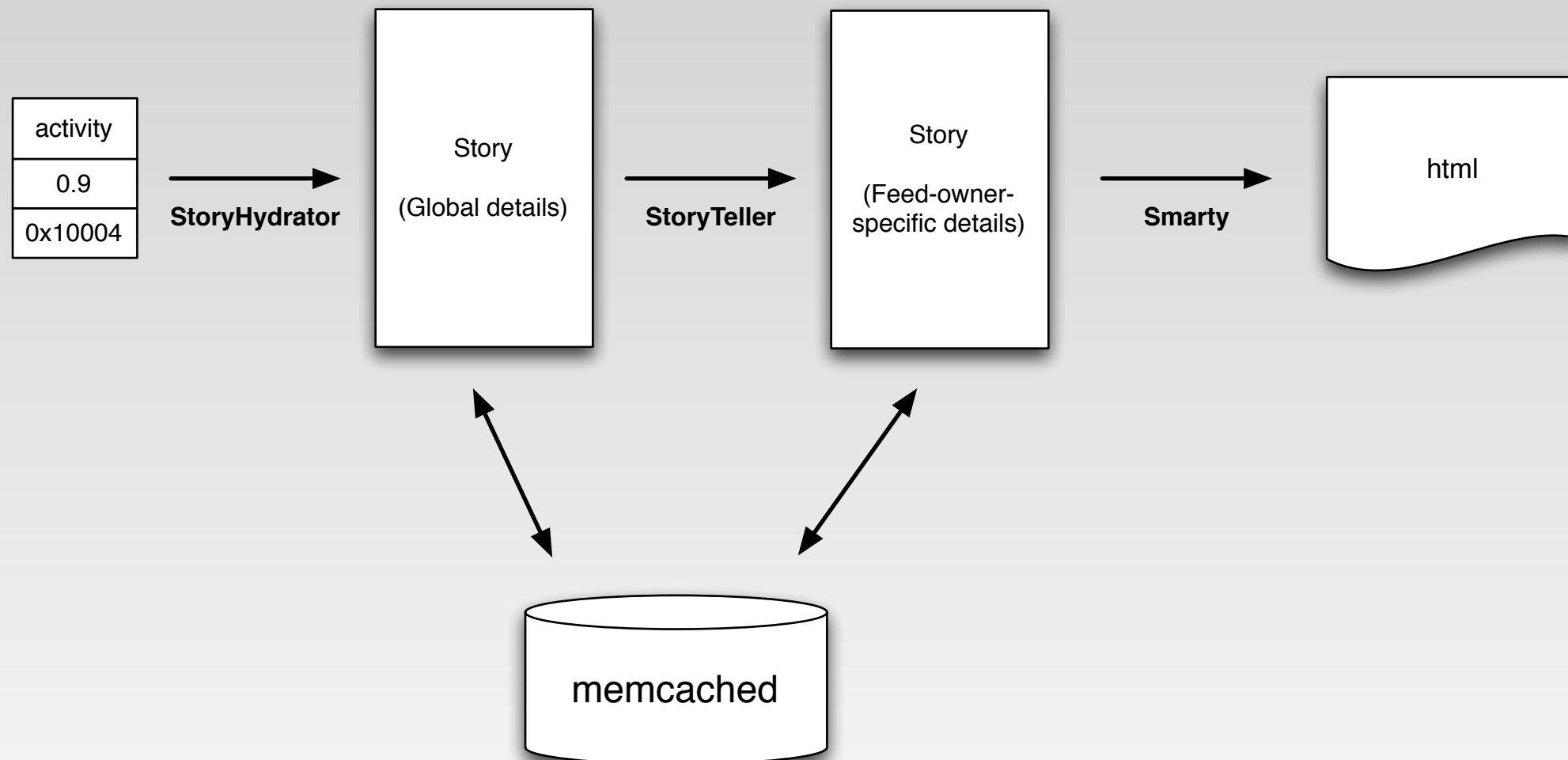
Display: Computing Rollups



Friday, January 14, 2011

Here's an attempt at depicting how rollups are created. The feed is divided up into sections, so that you don't wind up seeing all of the reds, greens, etc. on the entire feed in just a few very large rollups. Then the similar stories are grouped together within the sections.

Display: Filling in Stories



Friday, January 14, 2011

Once that's done, we can go through the rest of the display pipeline for the root story in each rollup.

There are multiple layers of caching here. Things that are global (like the shop associated with a favorited listing) are cached separately from things that are unique to the person looking at the feed (like the exact way the story is phrased).

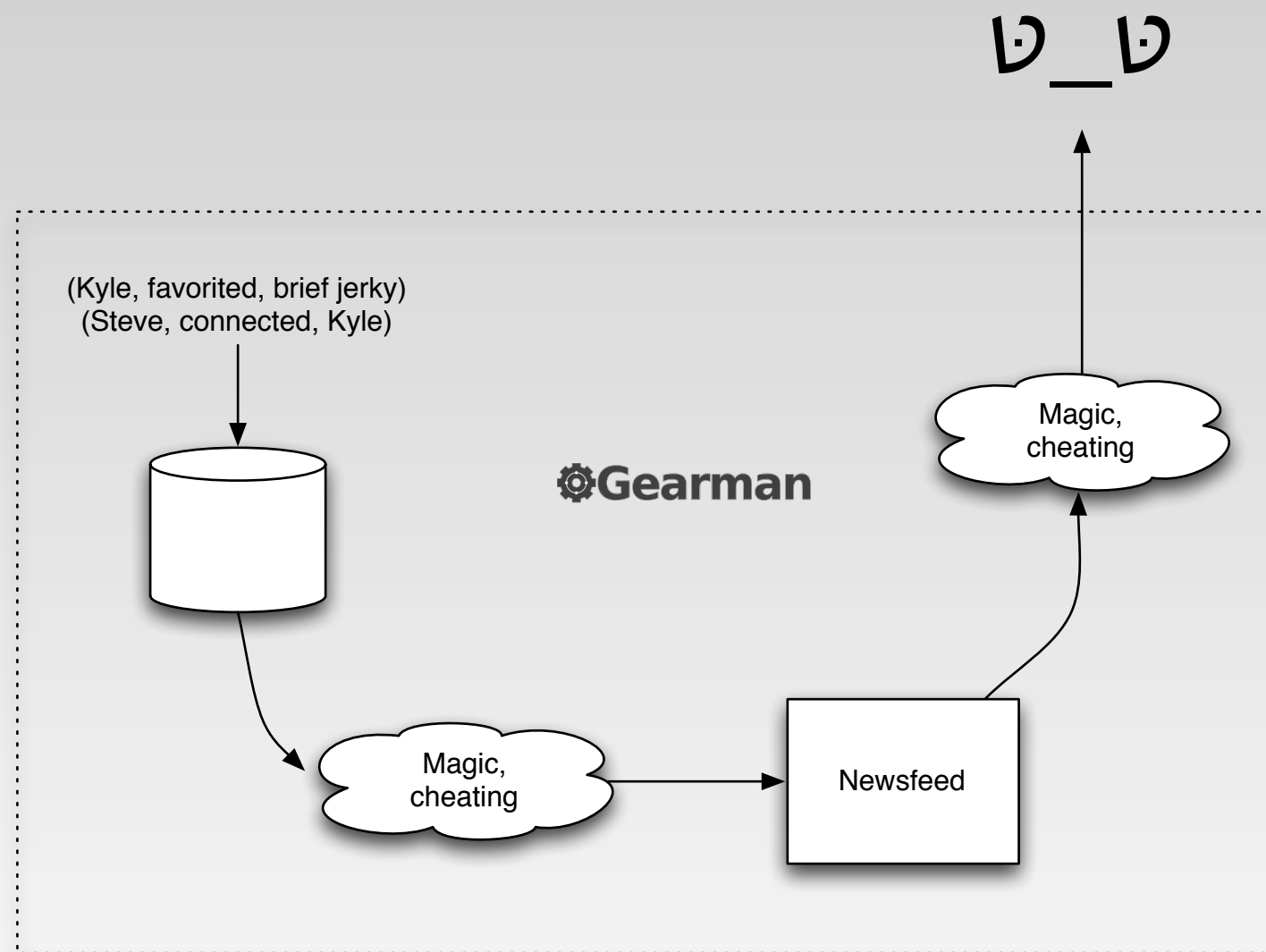
Making it Fast



Friday, January 14, 2011

Finally I'm going to go through a few ways that we've sped up activity, to the point where it's one of the faster pages on the site (despite being pretty complicated).

Hack #1: Cache Warming



Friday, January 14, 2011

The first thing we do to speed things up is run almost the entire pipeline proactively using gearman.

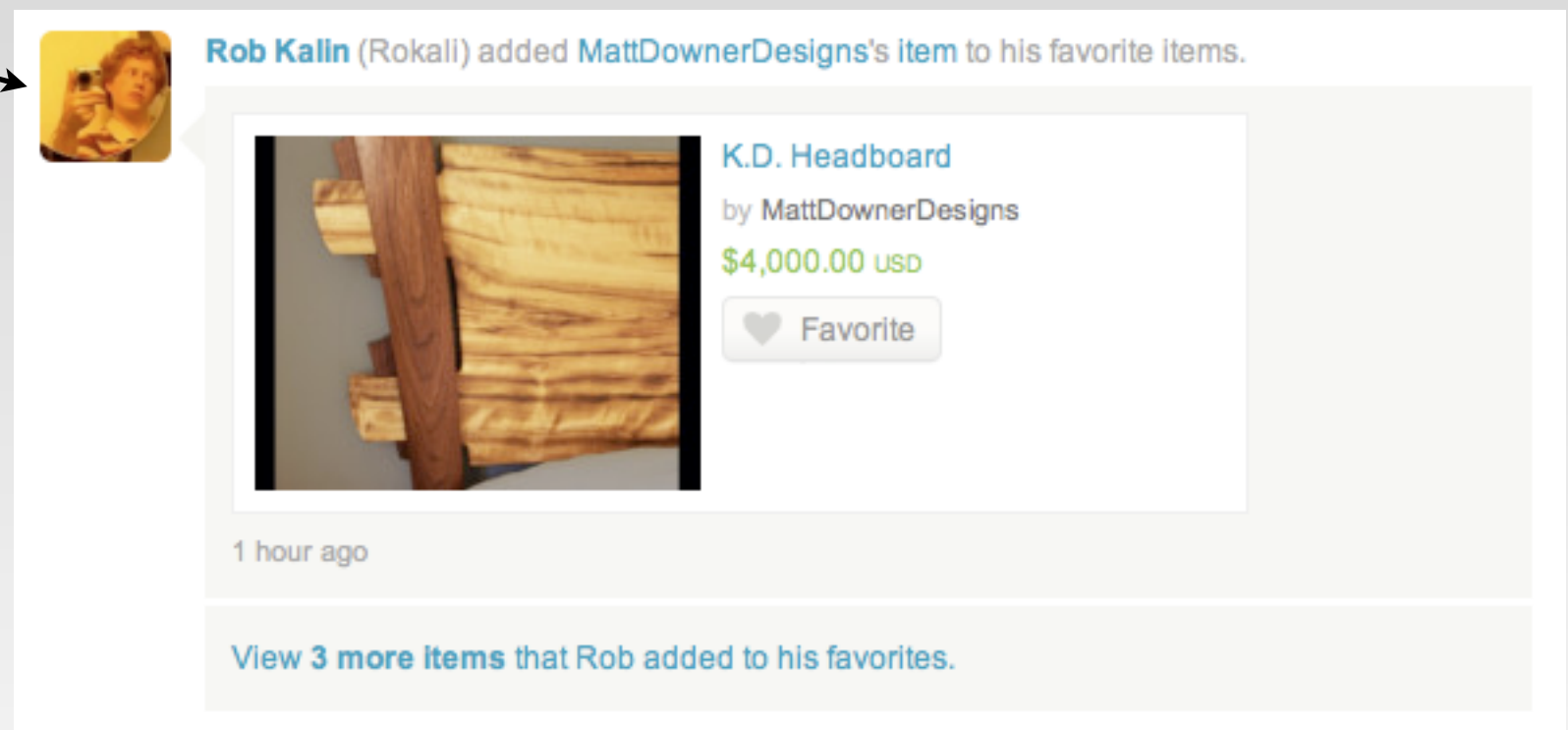
So after aggregation we trigger a display run, even though nobody is there to look at the html.

The end result is that almost every pageview is against a hot cache.

Hack #2: TTL Caching

May be his avatar
from 5 minutes
ago.

Big f'ing deal.



Friday, January 14, 2011

The second thing we do is add bits of TTL caching where few people will notice. Straightforward but not done in many places on the site.

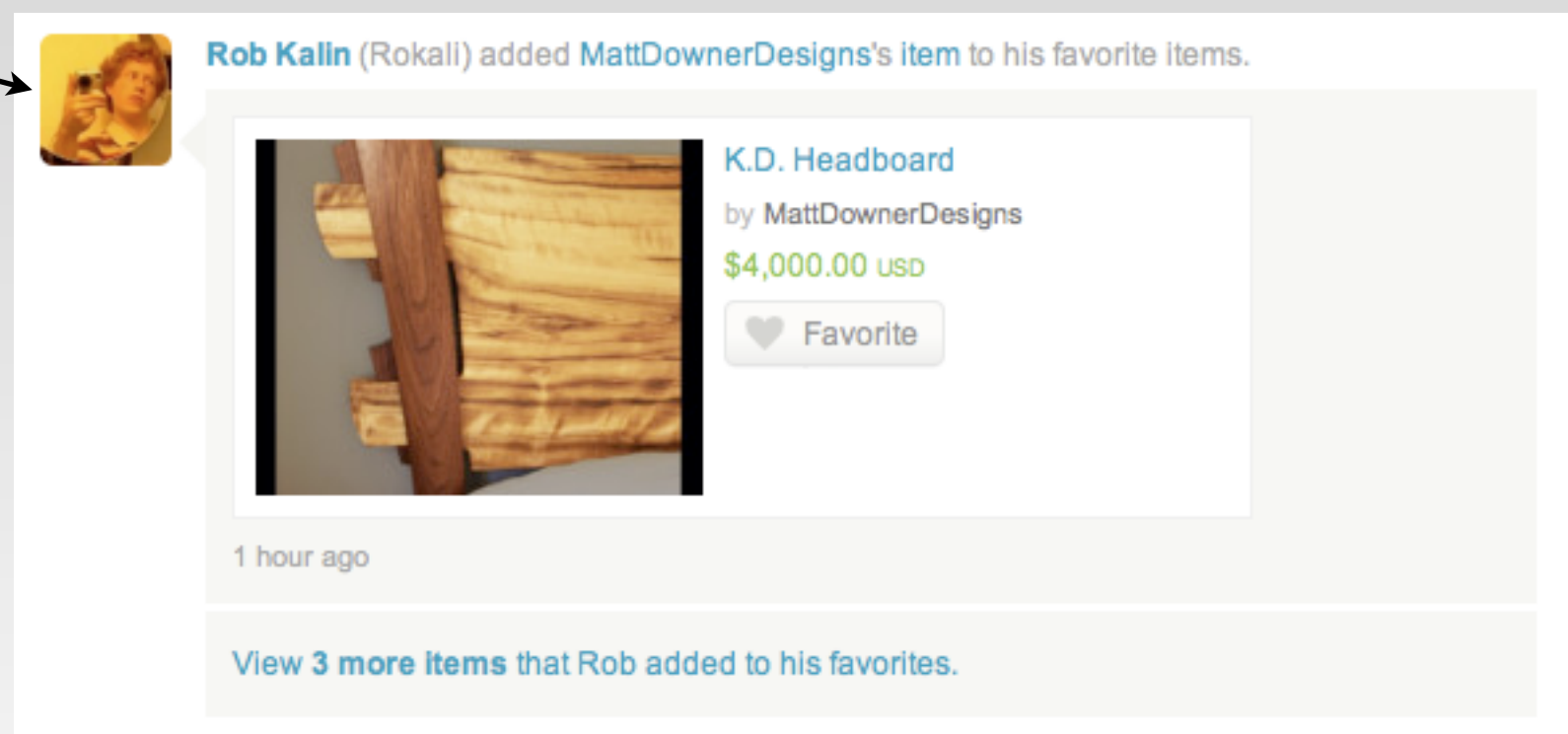
Note that his avatar here is tied to the story. If he generates new activity he'll see his new avatar.

Hack #3: Judicious Associations

```
getFinder("UserProfile")->find  
(...)
```

not

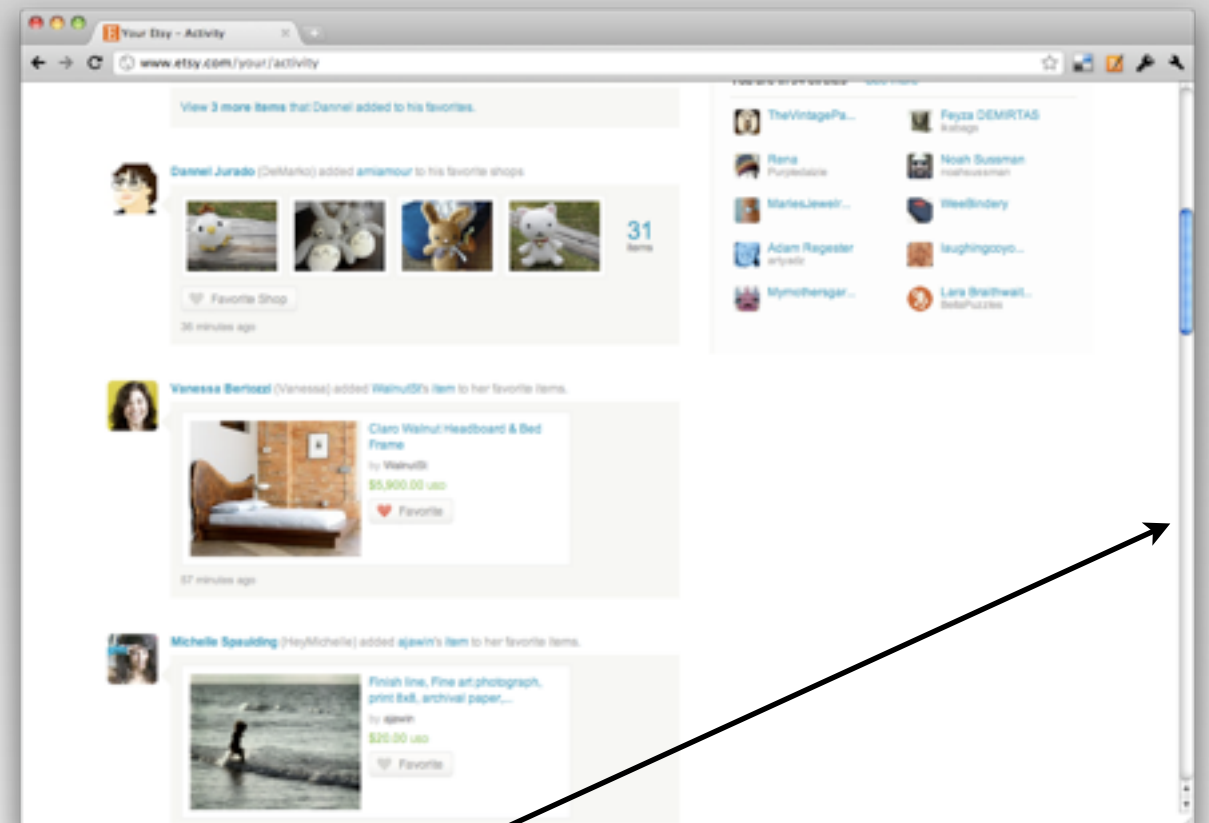
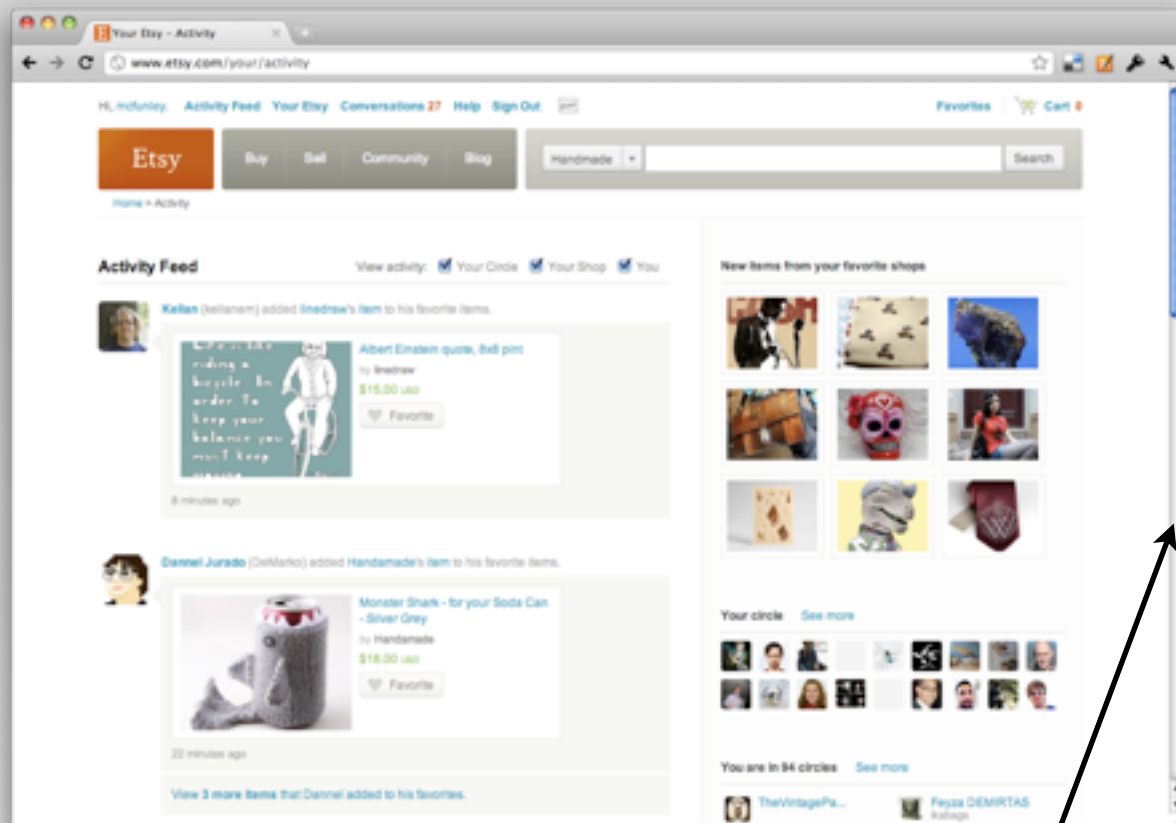
```
getFinder("User")->find(...)-  
>Profile
```



Friday, January 14, 2011

We also profiled the pages and meticulously simplified ORM usage. Again this sounds obvious but it's really easy to lose track of what you're doing as you hand the user off to the template. Lots of ORM calls were originally actually being made by the template.

Hack #4: Lazy Below the Fold



We don't load much at the outset.
You get more as you scroll down
(finite scrolling).

The End