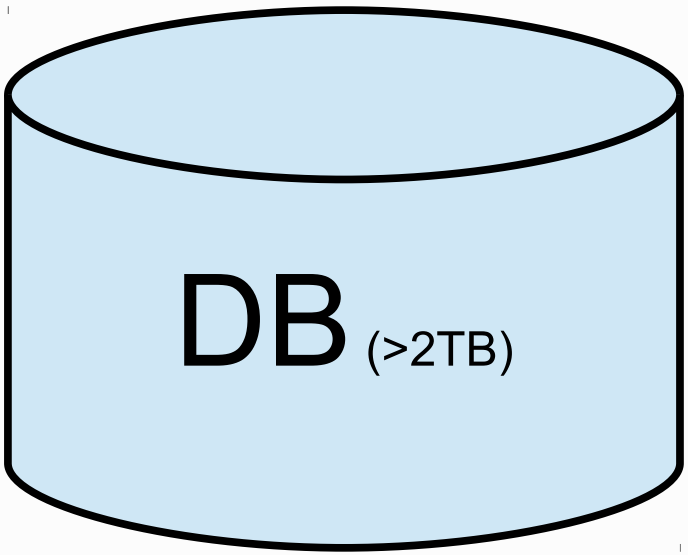
# Scalability challenges in Big Data Science

Mikio L. Braun
TU Berlin and TWIMPACT UG http://twimpact.com
mikiobraun on Twitter
mikio@twimpact.com

# Usually it starts like this




Let's

- cluster our user profiles
- classify our documents
- compute some nifty graph statistics

but how?

# First step: Scalable Database!

Add a dash of

Cassandra

http://cassandra.apache.org

mongoDB

{name: "mongo", type:"DB"}

http://mongodb.org

riak

http://wiki.basho.com/

MySQL

(sharded, of course)

http://www.mysql.com

→ But that won't scale your computations!

# Ok, some multi-threadding

Add

- Multithreading
- Actors
- Messaging Middleware

http://activemq.apache.org

http://www.zeromq.org/

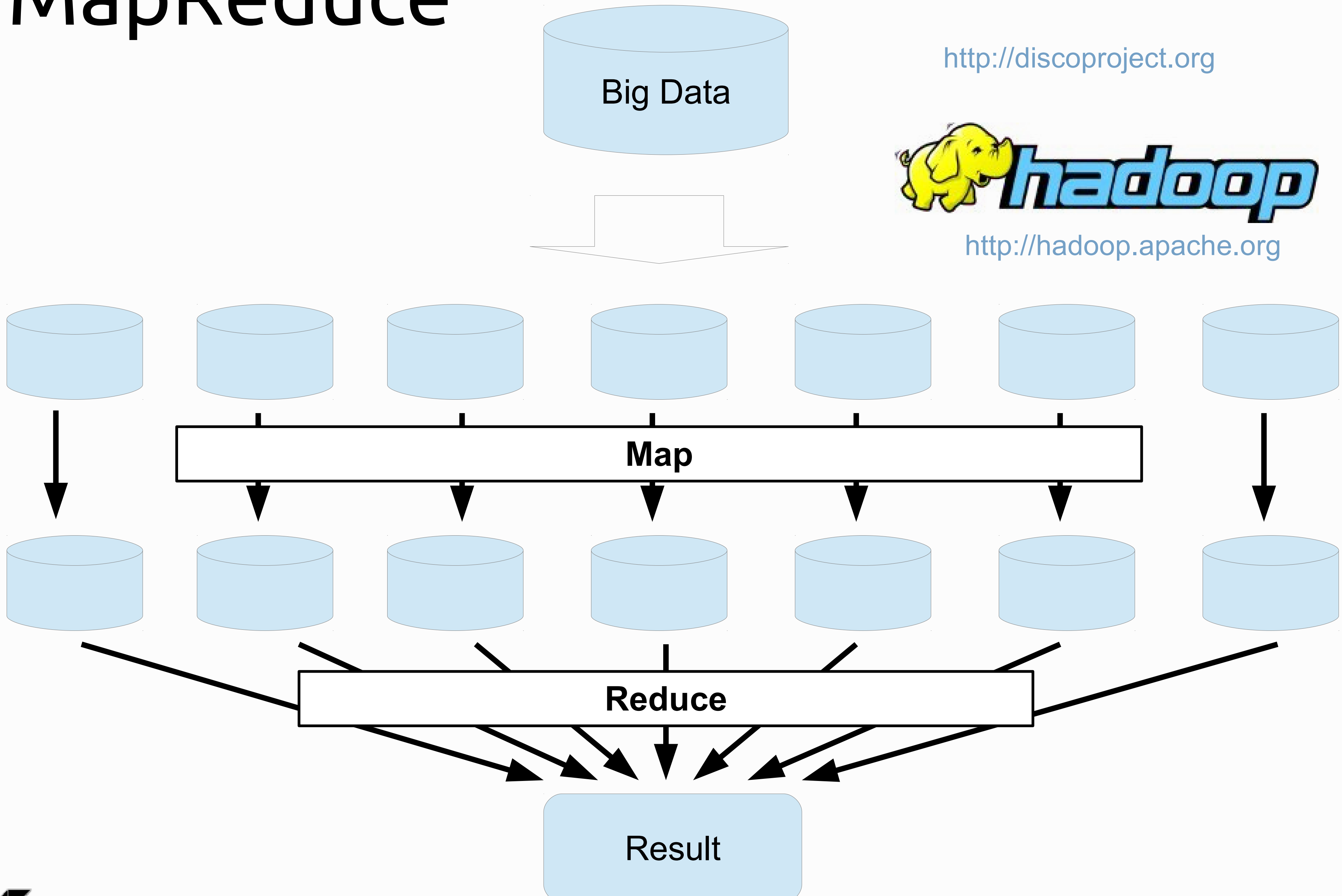
http://akka.io

But without transactions? central control?

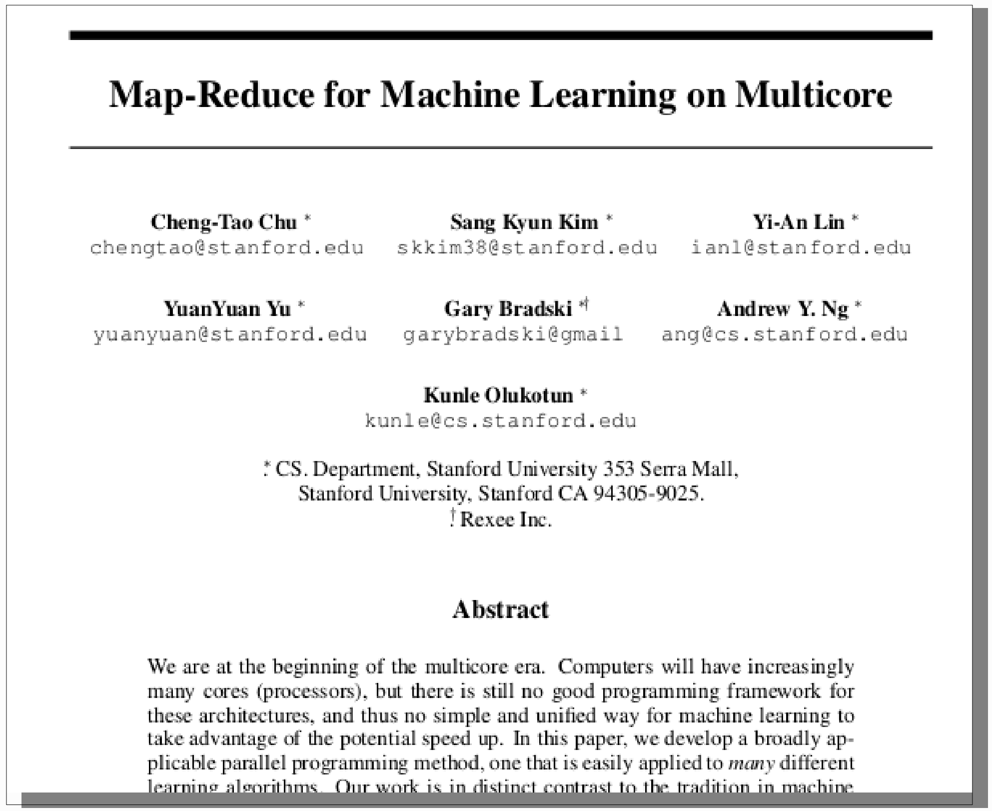
# MapReduce

http://discoproject.org

http://hadoop.apache.org



# The paper that started it all



## Map-Reduce for Machine Learning on Multicore

**Cheng-Tao Chu** [*] chengtao@stanford.edu
**Sang Kyun Kim** [*] skkim38@stanford.edu
**Yi-An Lin** [*] ianl@stanford.edu
**YuanYuan Yu** [*] yuanyuan@stanford.edu
**Gary Bradski** [*†] garybradski@gmail
**Andrew Y. Ng** [*] ang@cs.stanford.edu
**Kunle Olukotun** [*] kunle@cs.stanford.edu

[*] CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
[†] Rexee Inc.

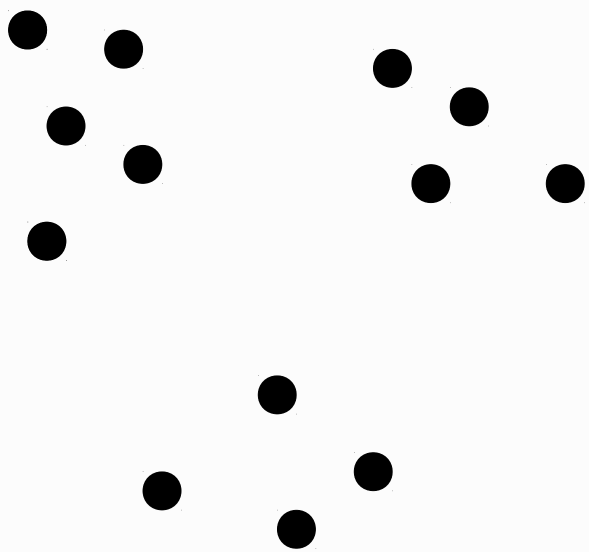
### Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine

Neural Information Processing Systems Conference, 2006

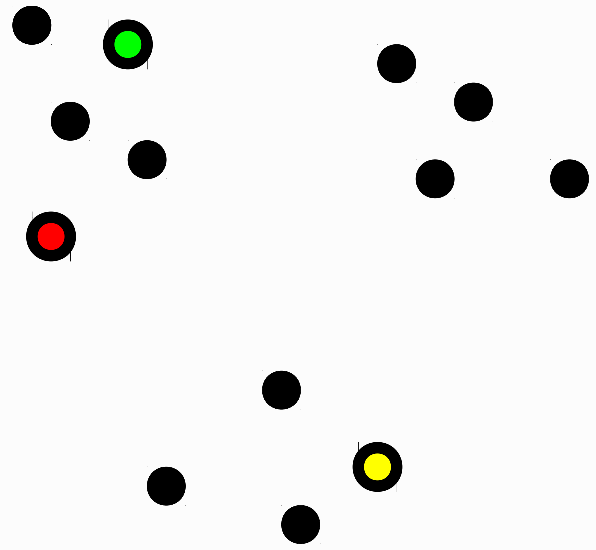
- Showed how to adapt ML algorithms to MapReduce
- Locally Weighted Linear Regression, Naive Bayes, Gaussian Discriminative Analysis, k-Means, Logistic Regression, Neural Networks, Principal Component Analysis, Independent Component Analysis, Expectation Maximization, Support Vector Machines

# Example: k-means Clustering



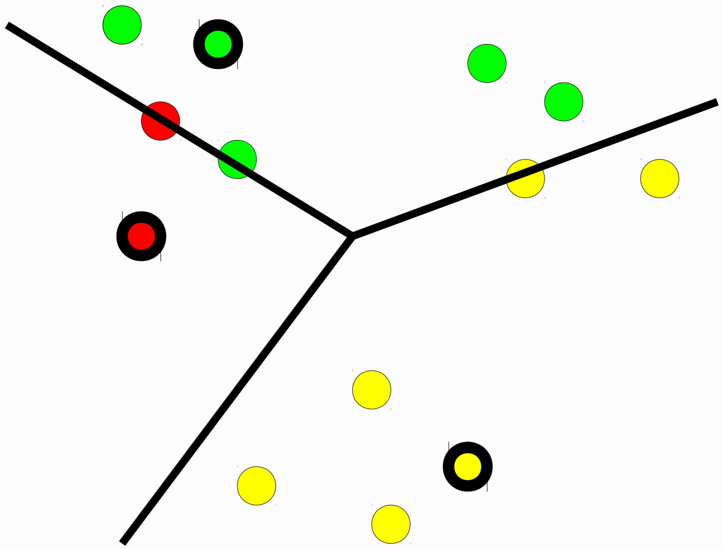
```
Input: points X1,...Xn, number k
Output: centroids µ1,...,µk

Initialize k centroids µ1,...,µk at
random

repeat until converged

   compute all distances between
     points and centroids

   map each point to closest
     centroid

   update centroids by computing
     average of all points in
     cluster
end
```

# Example: k-means Clustering



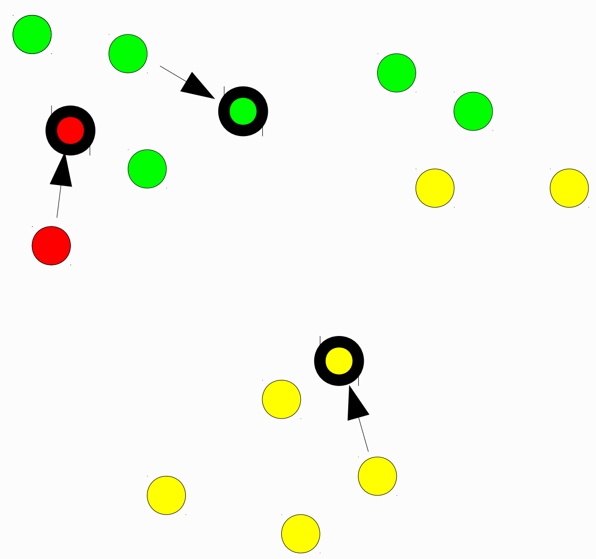
```
Input: points X1,...Xn, number k
Output: centroids µ1,...,µk

Initialize k centroids µ1,...,µk at
random

repeat until converged

   compute all distances between
     points and centroids

   map each point to closest
     centroid

   update centroids by computing
     average of all points in
     cluster
end
```

# Example: k-means Clustering



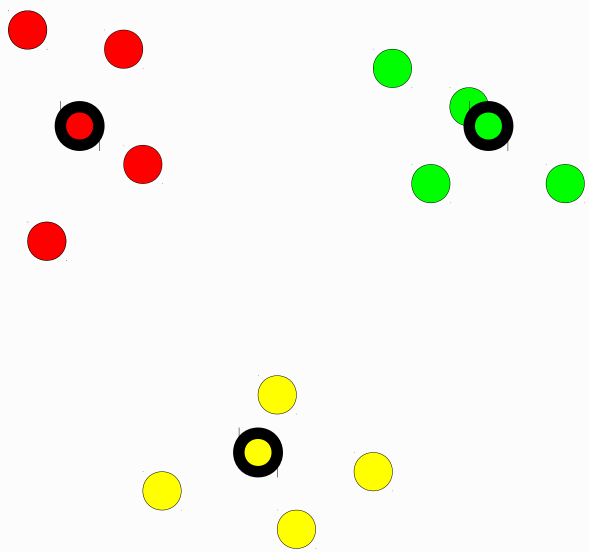
```
Input: points X1,...Xn, number k
Output: centroids µ1,...,µk

Initialize k centroids µ1,...,µk at
random

repeat until converged

    compute all distances between
      points and centroids

    map each point to closest
      centroid

    update centroids by computing
      average of all points in
      cluster
end
```

# Example: k-means Clustering



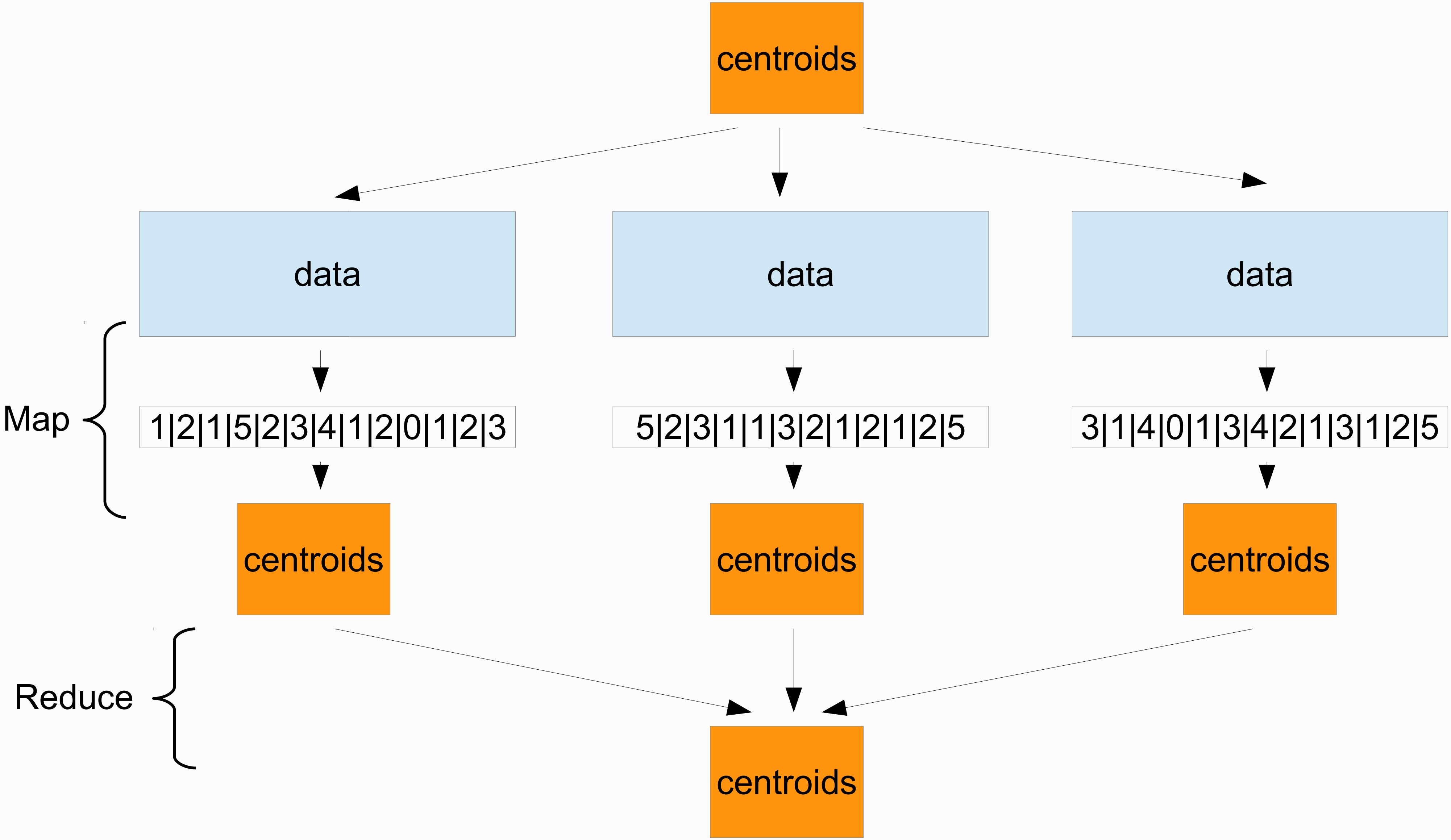
```
Input: points X1,...Xn, number k
Output: centroids µ1,...,µk

Initialize k centroids µ1,...,µk at
random

repeat until converged

    compute all distances between
      points and centroids

    map each point to closest
      centroid

    update centroids by computing
      average of all points in
      cluster
end
```

# Example: k-means Clustering



```
Input: points X1,...Xn, number k
Output: centroids µ1,...,µk

Initialize k centroids µ1,...,µk at
random

repeat until converged

    compute all distances between
      points and centroids

    map each point to closest
      centroid

    update centroids by computing
      average of all points in
      cluster
end
```

# Example: k-means Clustering



# k-means: Serial vs. Map Reduce

```
repeat until converged

   compute all distances between
     points and centroids

   map each point to closest
     centroid

   update centroids by computing
     average of all points in
     cluster
end
```

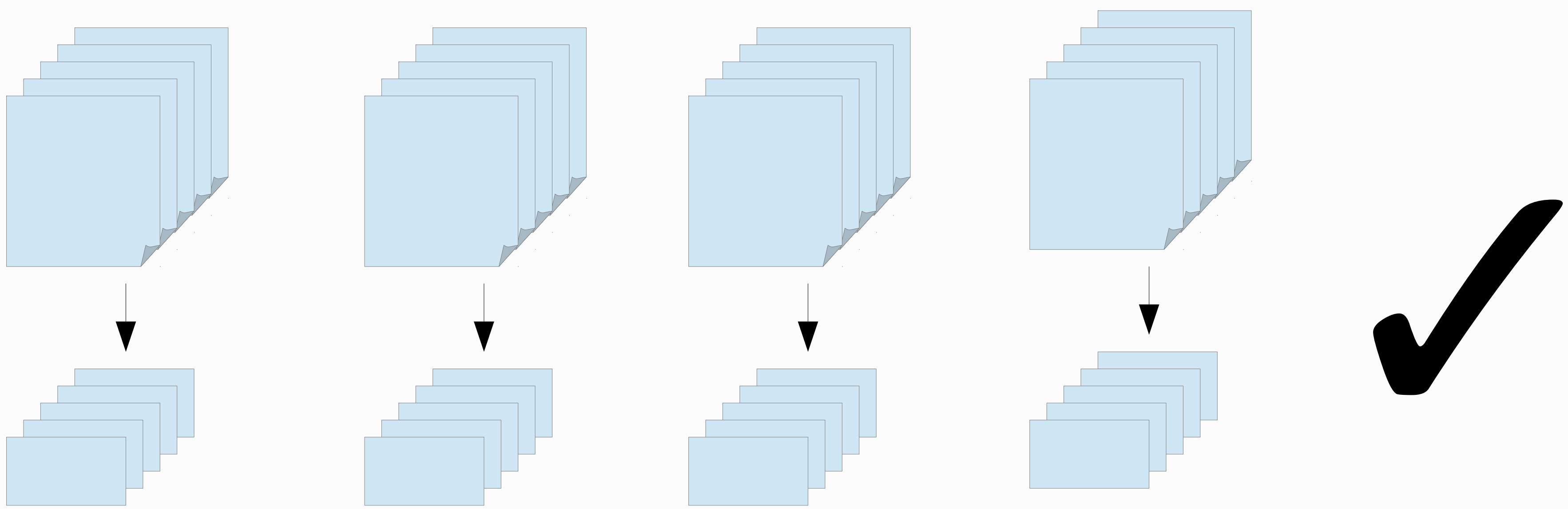
```
job k-means
    map:
      compute all distances to
         centroid

      map each point to closest
         centroid

      compute new cluster center

    reduce:
      average cluster centers
end job

repeat until converged

   send centroids to cluster

   run job k-means

   get centroids
end
```

Additional housekeeping

And k-means clustering is one of the simplest algorithms to parallelize!

# This works: classifying documents

- Parallel predictions on millions of objects
  - document classification
  - profile classification
  - media processing, etc.



# What about training?

- How to train your SVM/vowpal wabbit/Naive bayes/k-nearest neighbors on 2TB of data?
- You probably don't have to.
- But if, how do you train on heaps of data?

# Large-Scale learning.

- Large-scale means a linear model.

First of all, there is no such thing as “Data Science”. There is no scientific discipline called “data science”. You can’t go to an university to study data science. On the other hand, I agree that there is such a thing as a data scientist. Whenever I see someone calling himself a data scientist, I think that my own profile would probably also match that description. But what is it a data scientist does?

Document

a:4 agree:1 all:1 also:1 an:1
as:2 but:1 called:1 calling:1
can:1 data:6 description:1
discipline:1 does:1 first:1 go:1
hand:1 himself:1 i:3 is:4 it:1
match:1 my:1 no:2 of:1 on:1
other:1 own:1 probably:1
profile:1 science:3 scientific:1
scientist:3 see:1 someone:1
study:1 such:2 t:1 that:3 the:1
there:3 thing:2 think:1 to:2
university:1 what:1 whenever:1
would:1 you:1

Features (what you'll learn on)

- Then, learn weights for each of the words to predict between usually two classes.

# How to do large scale training?

- SVM-Training

$$\min_w \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(1 - y_i(w \cdot x_i + b))_+$$

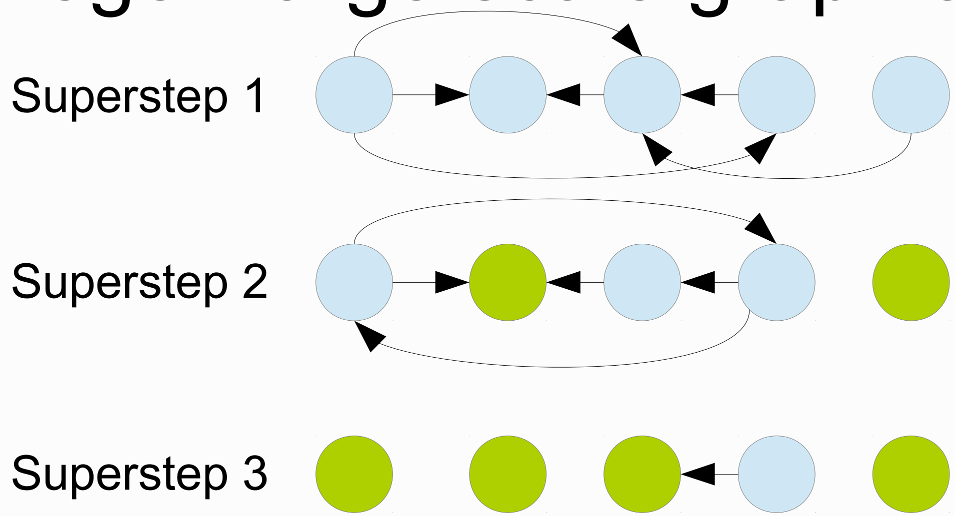
- Small-scale learning: Exact optimization
- Large-scale learning:
  - **Stochastic Gradient Descent** (one example at a time)
  - Other more complex methods (bundle methods, etc.)

# Stochastic Gradient Descent

- Do "gradient descent" on one point at a time
  - Take one point
  - Predict on that point
  - Update weights accordingly
- Model fits into memory, essentially IO bound
- Examples: vowpal wabbit http://hunch.net/~vw/
- Even the MapReduce paper only made "micro-batches"

# Other scaling concepts

- Pregel: large scale graph algorithms



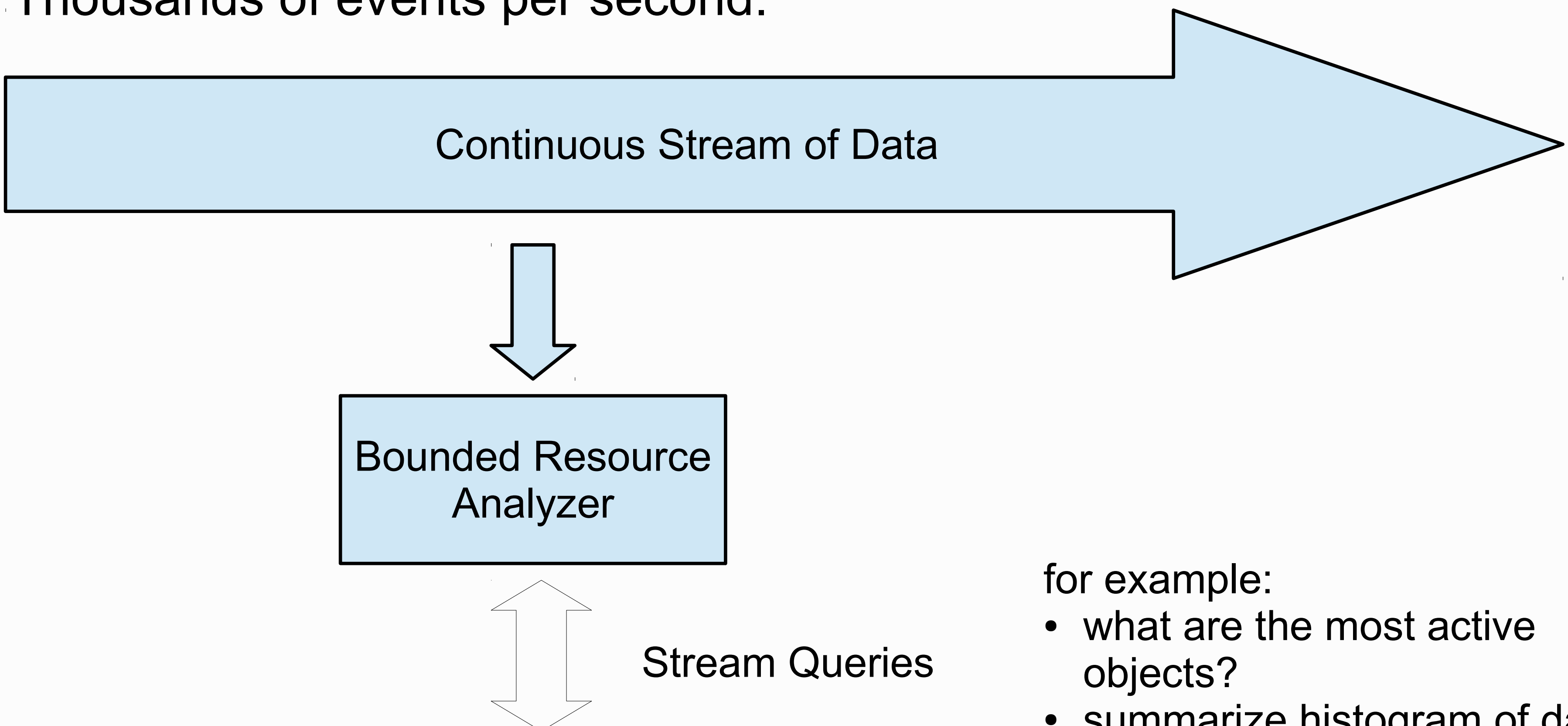

Actions:
- send messages
- read inbox
- change graph structure
- vote to halt

- Actor based / stream processing
  - Twitter's Storm https://github.com/nathanmarz/storm
  - Esper http://esper.codehaus.org/

Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski, *Pregel: A System for Large-Scale Graph Processing, SIGMOD'10*

# Stream Mining

- Large scale processing of event streams
- Very large domains (e.g. IP adresses, all users on Twitter)
- Thousands of events per second.




for example:
- what are the most active objects?
- summarize histogram of data
- range queries

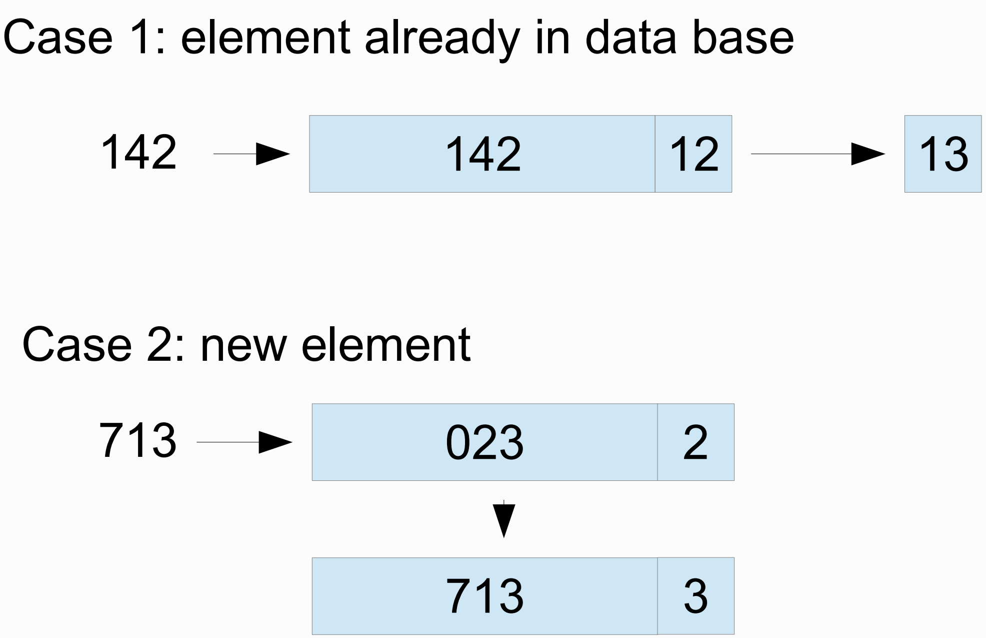
Excellent lecture by Alex Smola: http://alex.smola.org/teaching/berkeley2012/streams.html

# Heavy Hitters

- Count activities over large item sets (millions, even more, e.g. IP addresses, Twitter users)
- Interested in most active elements only.

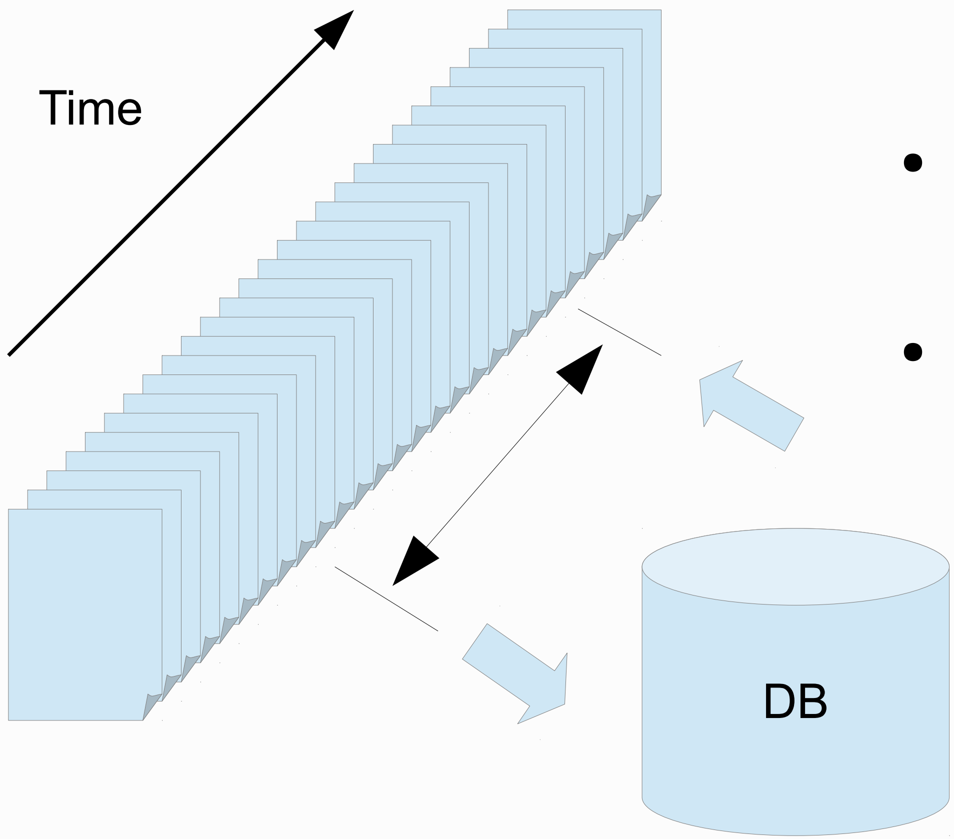
| | |
|---|---|
| 132 | 15 |
| 142 | 12 |
| 432 | 8 |
| 553 | 5 |
| 712 | 3 |
| 023 | 2 |

Fixed tables of counts

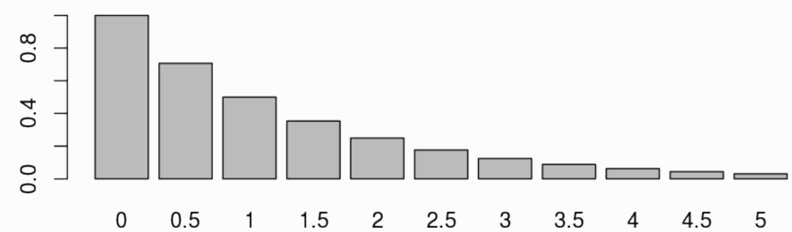



Metwally, Agrawal, Abbadi, *Efficient computation of Frequent and Top-k Elements in Data Streams, Internation Conference on Database Theory, 2005*

# Heavy Hitters over Time-Window




- Keep quite a big log (a month?)
- Constant write/erase in database
- Alternative: Exponential decay



# Hashing

- Compress large feature sets to smaller sets at random.
- On average, you make a very small error.

a:4 agree:1 all:1 also:1 an:1
as:2 but:1 called:1 calling:1
can:1 data:6 description:1
discipline:1 does:1 first:1 go:1
hand:1 himself:1 i:3 is:4 it:1
match:1 my:1 no:2 of:1 on:1
other:1 own:1 probably:1
profile:1 science:3 scientific:1
scientist:3 see:1 someone:1
study:1 such:2 t:1 that:3 the:1
there:3 thing:2 think:1 to:2
university:1 what:1 whenever:1
would:1 you:1

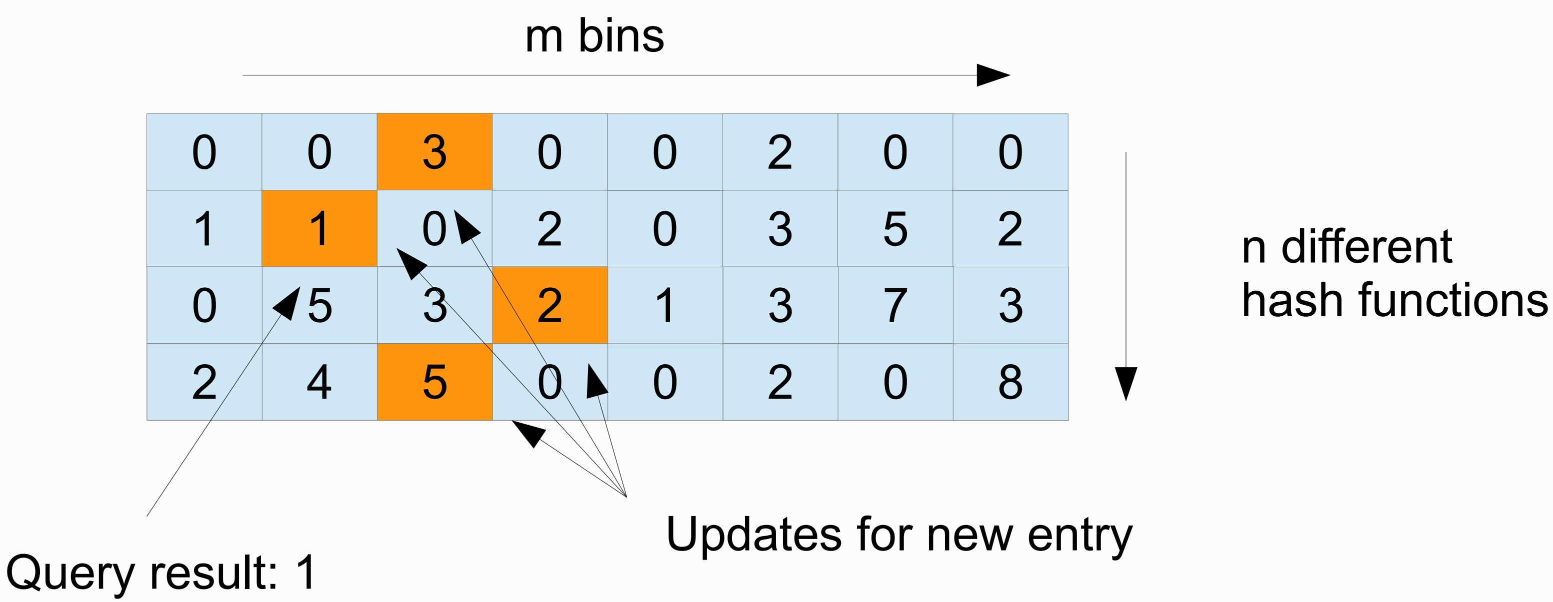
*-5:description,probably,profile*
*-3:someone,called*
*0:scientist,agree,data,as,i,go,my,no*
*1:would,also,university,does,other,of,the*
*2:hand,thing,such,that,science,on,an*
*3:is,own,all*
*4:calling,t,it,to,there,can,see*
*5:but,whenever,first,discipline,study*
*6:scientific,himself,what,a,match,you,think*

-5:3
-3:2
0:19
1:7
2:13
3:6
4:10
5:5
6:10

Weinberger, Dasgupta, Attenberg, Langford und Smola, *Feature Hashing for Large Scale Multitask Learning*, ICML, 2009

# Count-Min Sketches

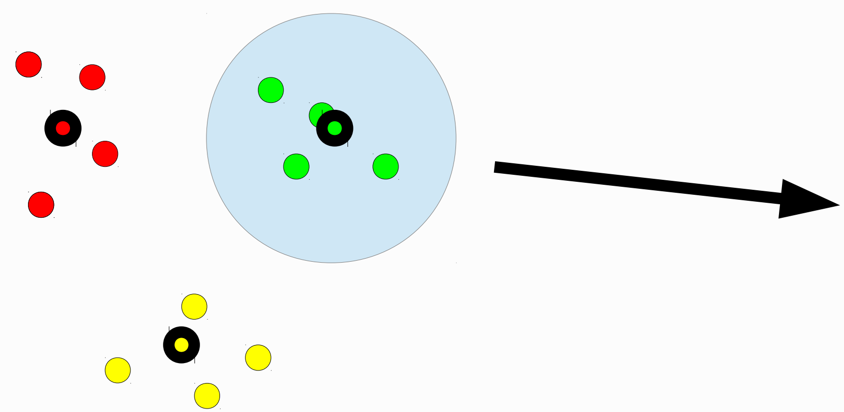
- Summarize histograms over large feature sets
- Like hashing, but better




- Query: Take minimum over all hash functions

# Clustering with count-min Sketches

- Online clustering
  - For each data point:
    - Map to closest centroid (=> compute distances)
    - Update centroid
  - count-min sketches to represent sum over all vectors in a class



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 0 | 2 | 0 | 0 |
| 1 | 1 | 0 | 2 | 0 | 3 | 5 | 2 |
| 0 | 5 | 3 | 2 | 1 | 3 | 7 | 3 |
| 2 | 4 | 5 | 0 | 0 | 2 | 0 | 8 |

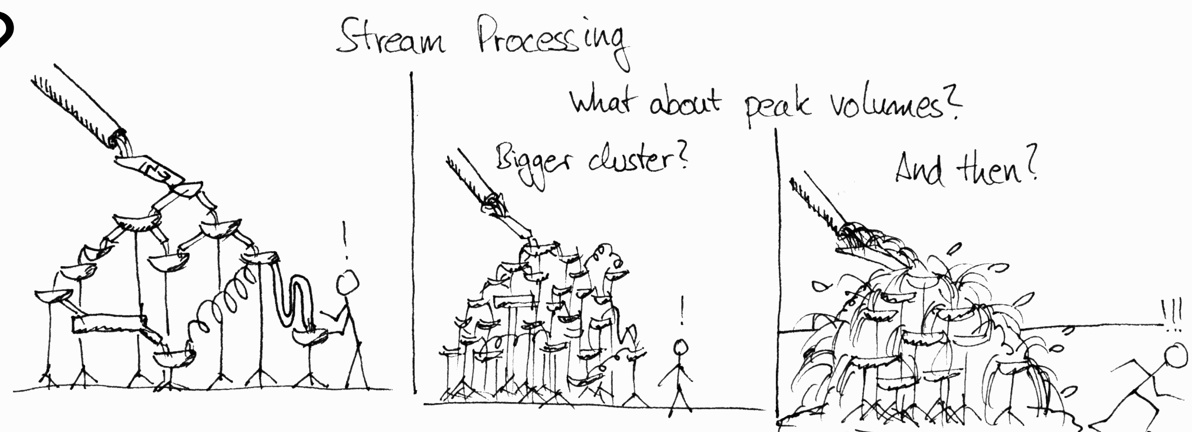
Aggarwal, *A Framework for Clustering Massive-Domain Data Streams, IEEE International Conference on Data Engineering , 2009*

# BUT, what about real-time?

# Scale into Real-Time?

- Putting everything in a data base and running a query.

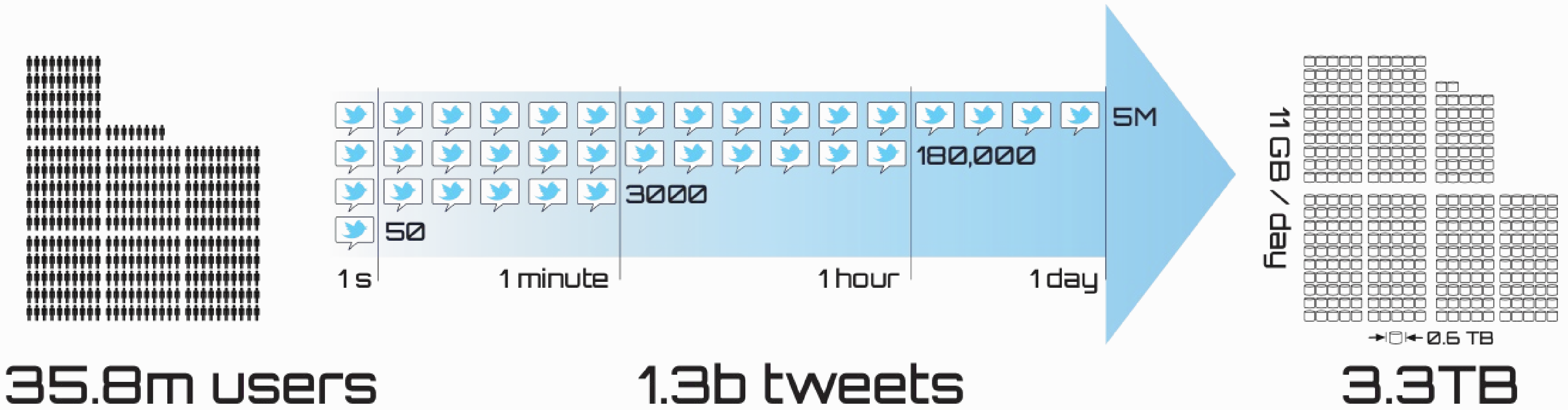

- What is the maximum throughput for stream processing?



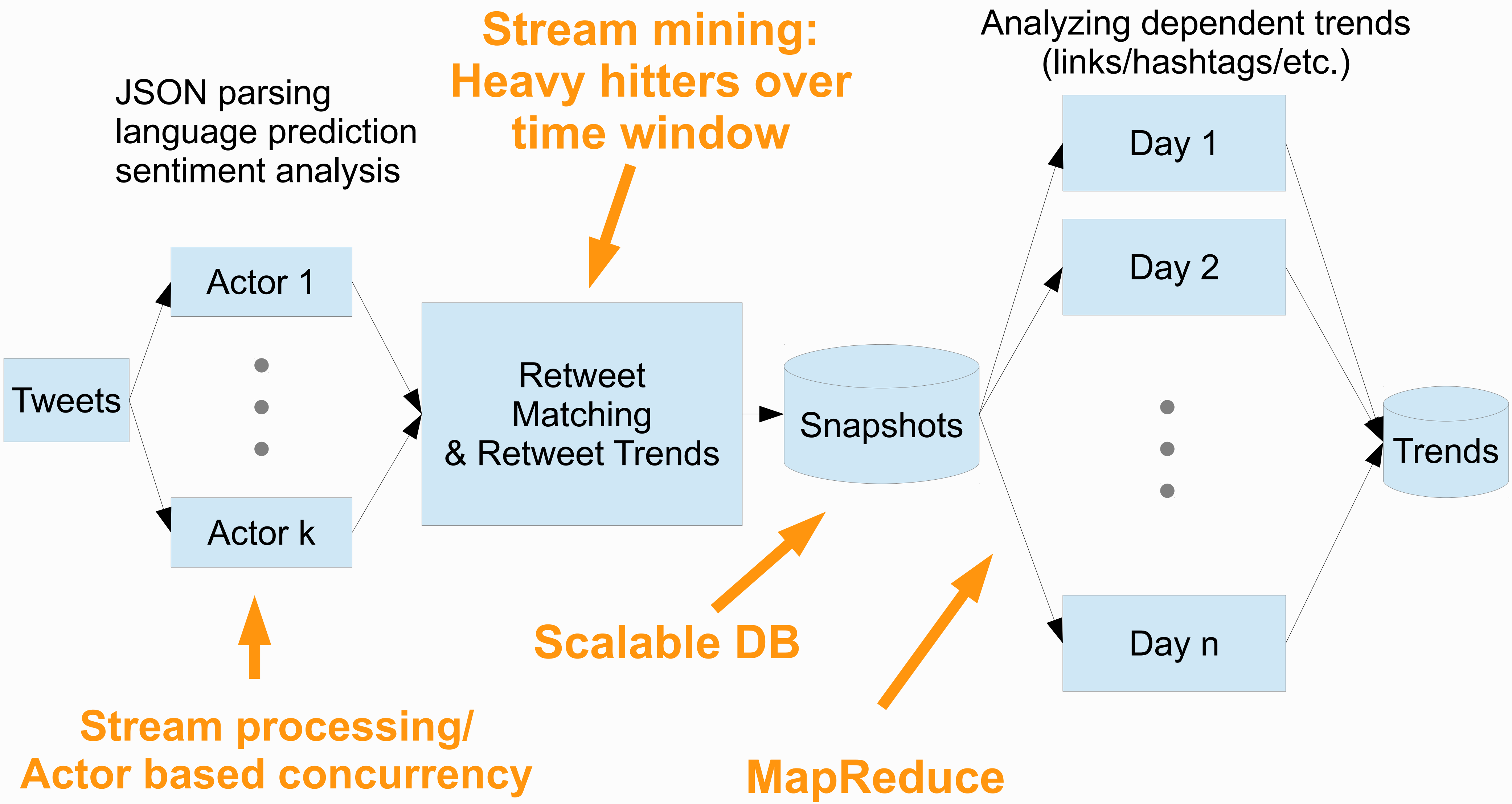
# Real-Time Requirements

- What do we need for real-time:
  - Guaranteed constant processing time per event.
  - Resilience against volume peaks.
- Our recipe for real-time:
  - Stream-mining method (heavy hitters, etc.)
  - Keep “hot data” in memory
  - Add scalable technology as needed for persistence, etc.

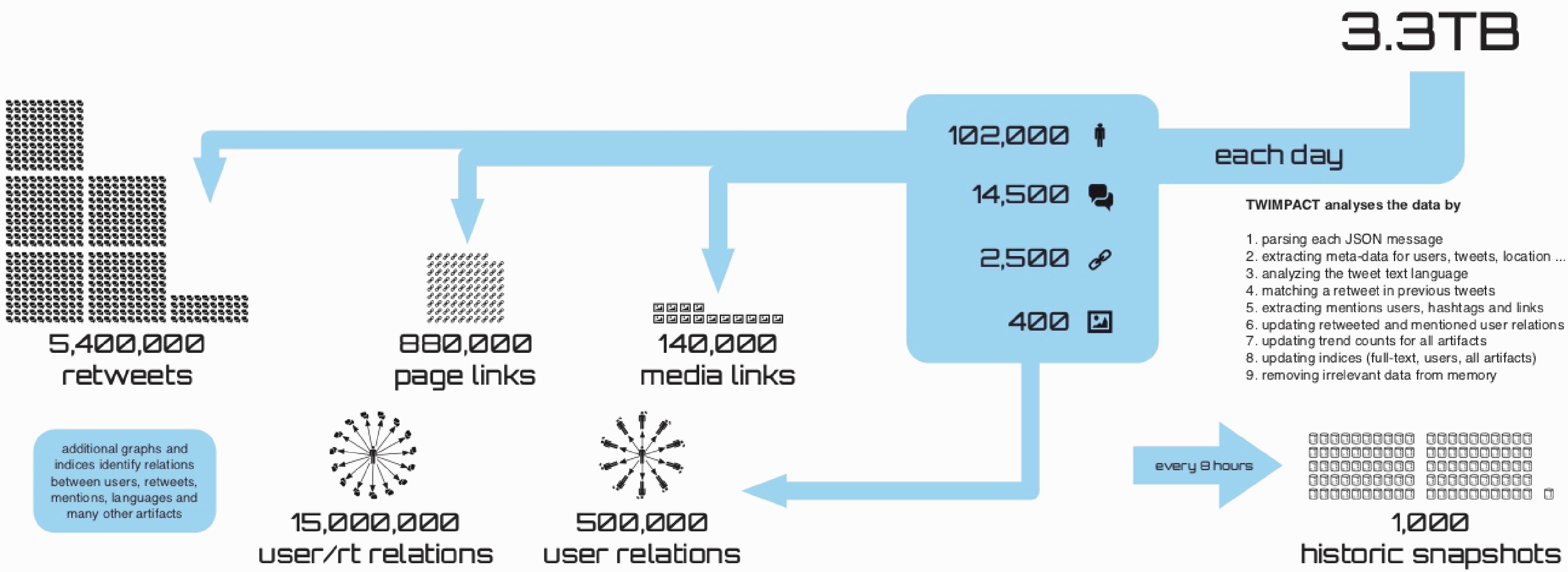
# 2011 in Retweets



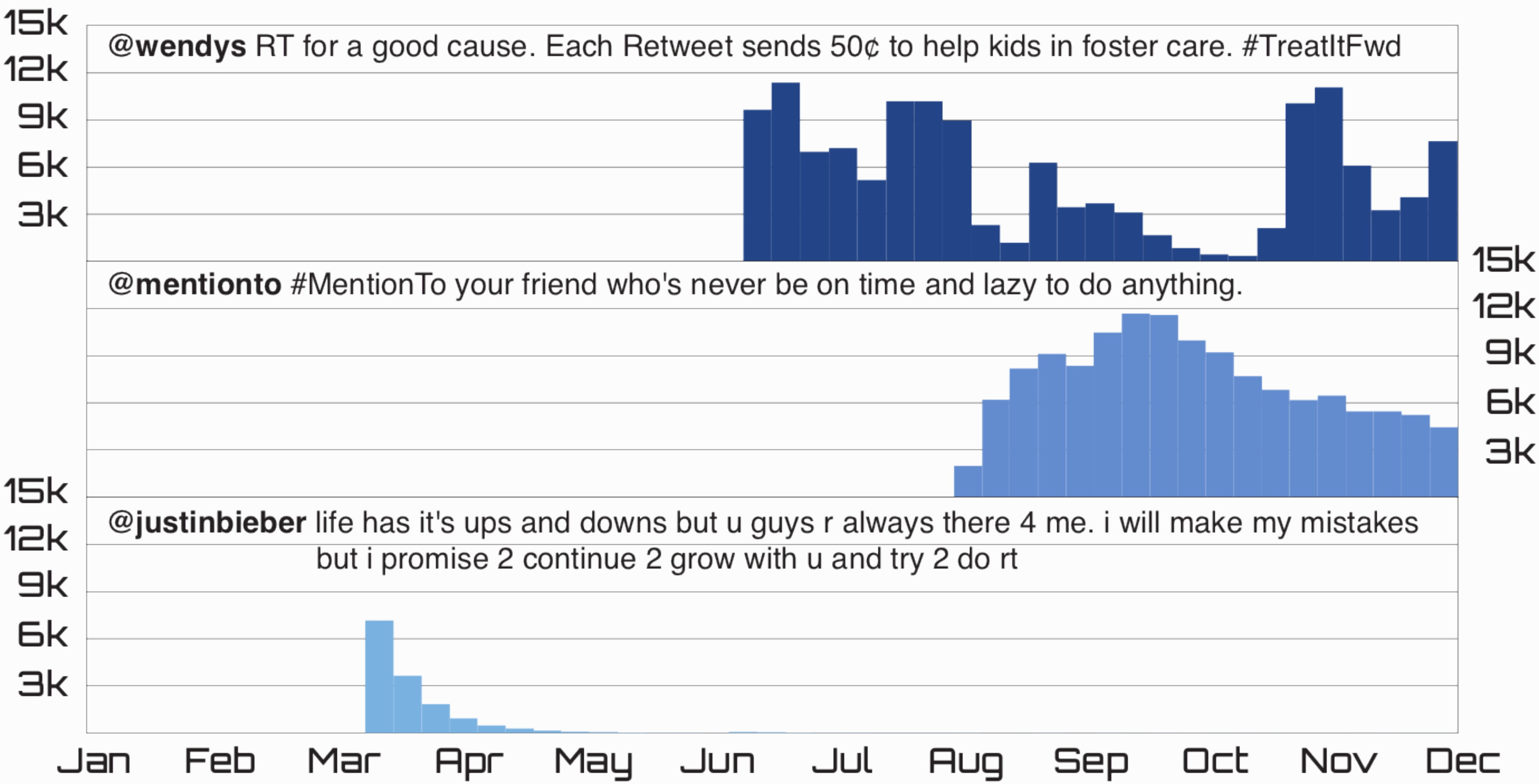
# TWIMPACT Analysis Pipeline



# 2011 in Retweets



# Most retweeted tweets



# Summary

- **Big Data Science is not just a scaling problem.**
- To scale, you need to scale data & computation
- Roll Your Own, or use an existing framework
- Computation models might be unnatural
- Large-scale learning: **linear models & one example at a time**
- **Stream mining:** heavy hitters, hashing, count-min sketches
- **You don't scale into real-time.**

- DataScience seminars: datascience-berlin.org
- serienradar.de: real-time TV trends from Twitter

# So what are the challenges?

- Non-locality of learning algorithms.
- Dealing with large amounts of writes.
- Maximum through-put of stream processing.
- Real-time.