Scaling up Machine Learning

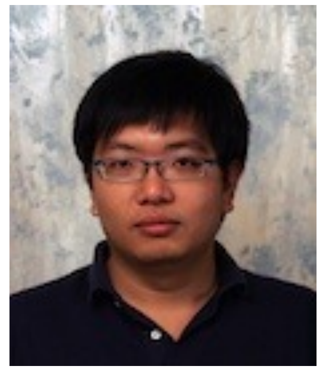
Alex Smola

Yahoo! Research
Santa Clara
alex.smola.org

Thanks

Amr
Ahmed

Joey
Gonzalez



Yucheng
Low

Qirong
Ho

Ziad
al Bawab

Sergiy
Matyusevich

Shravan
Narayanamurthy

Kilian
Weinberger

John
Langford

Vanja
Josifovski

Quoc
Le

Choon Hui
Teo

Eric
Xing

James
Petterson

Jake
Eisenstein

Shuang Hong
Yang

Vishy
Vishwanathan

Zhaohui
Zheng

Markus
Weimer

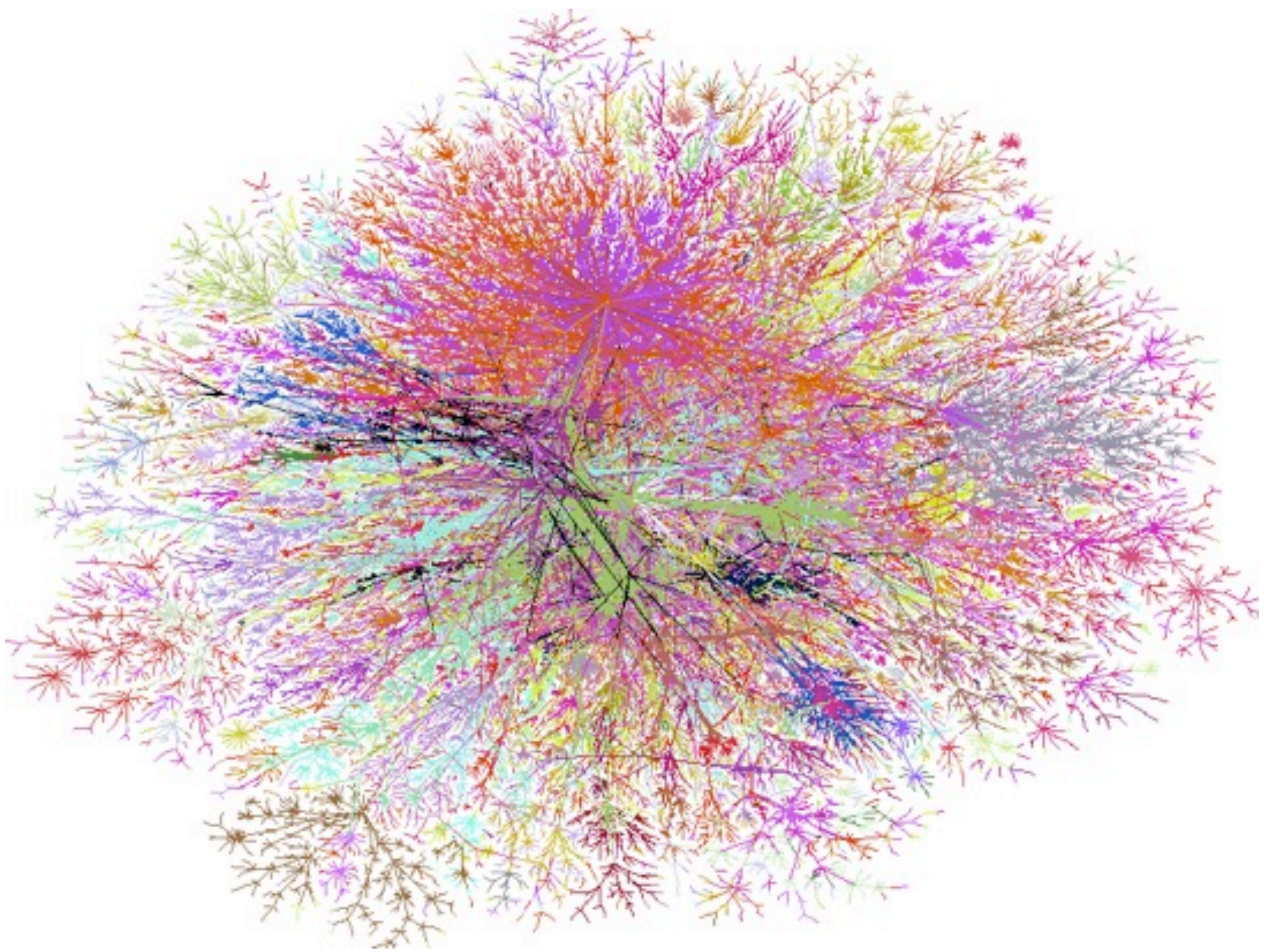
Alexandros
Karatzoglou

Martin
Zinkevich

Why

Data

- Webpages (content, graph)
- Clicks (ad, page, social)
- Users (OpenID, FB Connect)
- e-mails (Hotmail, Y!Mail, Gmail)
- Photos, Movies (Flickr, YouTube, Vimeo ...)
- Cookies / tracking info (see Ghostery)
- Installed apps (Android market etc.)
- Location (Latitude, Loopt, Foursquared)
- User generated content (Wikipedia & co)
- Ads (display, text, DoubleClick, Yahoo)
- Comments (Disqus, Facebook)
- Reviews (Yelp, Y!Local)
- Third party features (e.g. Experian)
- Social connections (LinkedIn, Facebook)
- Purchase decisions (Netflix, Amazon)
- Instant Messages (YIM, Skype, Gtalk)
- Search terms (Google, Bing)
- Timestamp (everything)
- News articles (BBC, NYTimes, Y!News)
- Blog posts (Tumblr, Wordpress)
- Microblogs (Twitter, Jaiku, Meme)



>10B useful webpages

Data - Identity & Graph

- Webpages (content, graph)
- Clicks (ad, page, social)
- Users (OpenID, FB Connect)
- e-mails (Hotmail, Y!Mail, Gmail)
- Photos, Movies (Flickr, YouTube, Vimeo ...)
- Cookies / tracking info (see Ghostery)
- Installed apps (Android market etc.)
- Location (Latitude, Loopt, Foursquared)
- User generated content (Wikipedia & co)
- Ads (display, text, DoubleClick, Yahoo)
- Comments (Disqus, Facebook)
- Reviews (Yelp, Y!Local)
- Third party features (e.g. Experian)
- Social connections (LinkedIn, Facebook)
- Purchase decisions (Netflix, Amazon)
- Instant Messages (YIM, Skype, Gtalk)
- Search terms (Google, Bing)
- Timestamp (everything)
- News articles (BBC, NYTimes, Y!News)
- Blog posts (Tumblr, Wordpress)
- Microblogs (Twitter, Jaiku, Meme)

100M-1B vertices

Data - User generated content

- Webpages (content, graph)
- Clicks (ad, page, social)
- Users (OpenID, FB Connect)
- e-mails (Hotmail, Y!Mail, Gmail)
- Photos, Movies (Flickr, YouTube, Vimeo ...)
- Cookies / tracking info (see Ghostery)
- Installed apps (Android market etc.)
- Location (Latitude, Loopt, Foursquared)
- User generated content (Wikipedia & co)
- Ads (display, text, DoubleClick, Yahoo)
- Comments (Disqus, Facebook)
- Reviews (Yelp, Y!Local)
- Third party features (e.g. Experian)
- Social connections (LinkedIn, Facebook)
- Purchase decisions (Netflix, Amazon)
- Instant Messages (YIM, Skype, Gtalk)
- Search terms (Google, Bing)
- Timestamp (everything)
- News articles (BBC, NYTimes, Y!News)
- Blog posts (Tumblr, Wordpress)
- Microblogs (Twitter, Jaiku, Meme)

>1B images, 40h video/minute

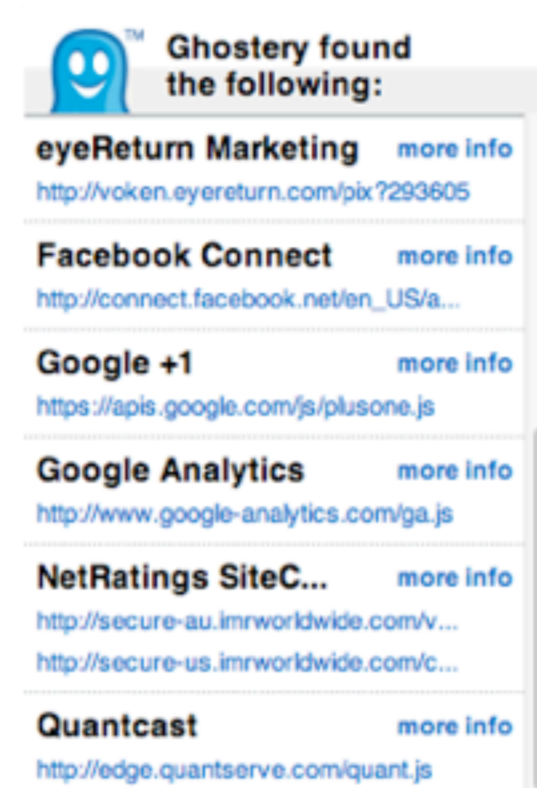
Data - Messages

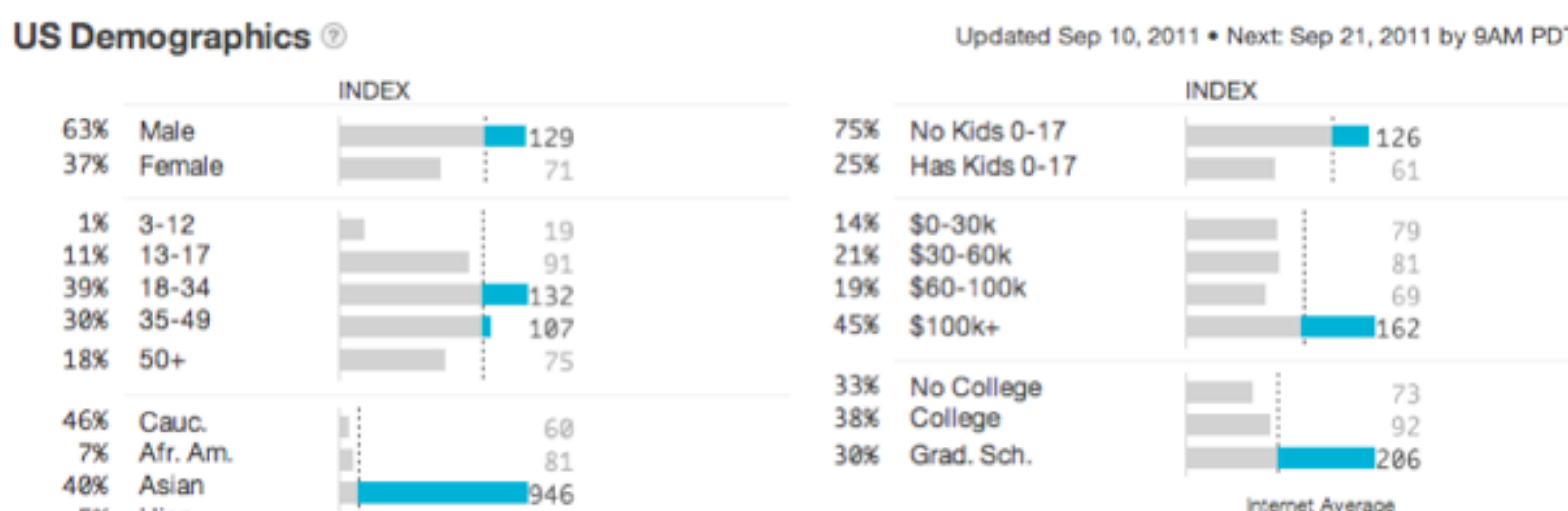
- Webpages (content, graph)
- Clicks (ad, page, social)
- Users (OpenID, FB Connect)
- e-mails (Hotmail, Y!Mail, Gmail)
- Photos, Movies (Flickr, YouTube, Vimeo ...)
- Cookies / tracking info (see Ghostery)
- Installed apps (Android market etc.)
- Location (Latitude, Loopt, Foursquared)
- User generated content (Wikipedia & co)
- Ads (display, text, DoubleClick, Yahoo)
- Comments (Disqus, Facebook)
- Reviews (Yelp, Y!Local)
- Third party features (e.g. Experian)
- Social connections (LinkedIn, Facebook)
- Purchase decisions (Netflix, Amazon)
- Instant Messages (YIM, Skype, Gtalk)
- Search terms (Google, Bing)
- Timestamp (everything)
- News articles (BBC, NYTimes, Y!News)
- Blog posts (Tumblr, Wordpress)
- Microblogs (Twitter, Jaiku, Meme)

>1B texts

Data - User Tracking

- Webpages (content, graph)
- Clicks (ad, page, social)
- Users (OpenID, FB Connect)
- e-mails (Hotmail, Y!Mail, Gmail)
- Photos, Movies (Flickr, YouTube, Vimeo ...)
- Cookies / tracking info (see Ghostery)
- Installed apps (Android market etc.)
- Location (Latitude, Loopt, Foursquared)
- User generated content (Wikipedia & co)
- Ads (display, text, DoubleClick, Yahoo)
- Comments (Disqus, Facebook)
- Reviews (Yelp, Y!Local)
- Third party features (e.g. Experian)
- Social connections (LinkedIn, Facebook)
- Purchase decisions (Netflix, Amazon)
- Instant Messages (YIM, Skype, Gtalk)
- Search terms (Google, Bing)
- Timestamp (everything)
- News articles (BBC, NYTimes, Y!News)
- Blog posts (Tumblr, Wordpress)
- Microblogs (Twitter, Jaiku, Meme)





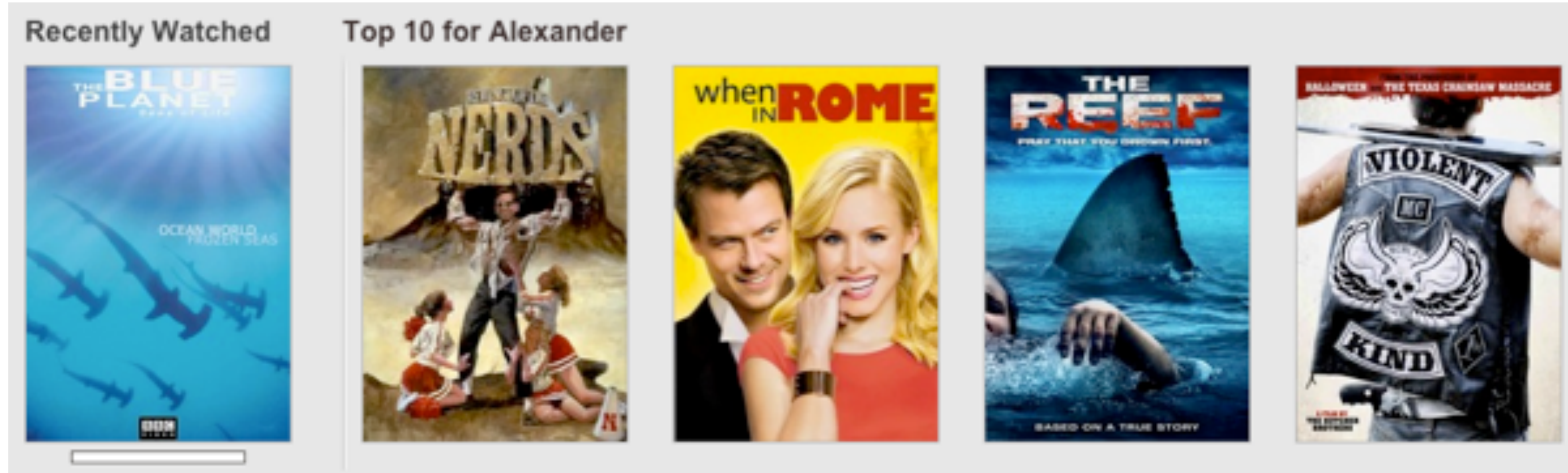
alex.smola.org

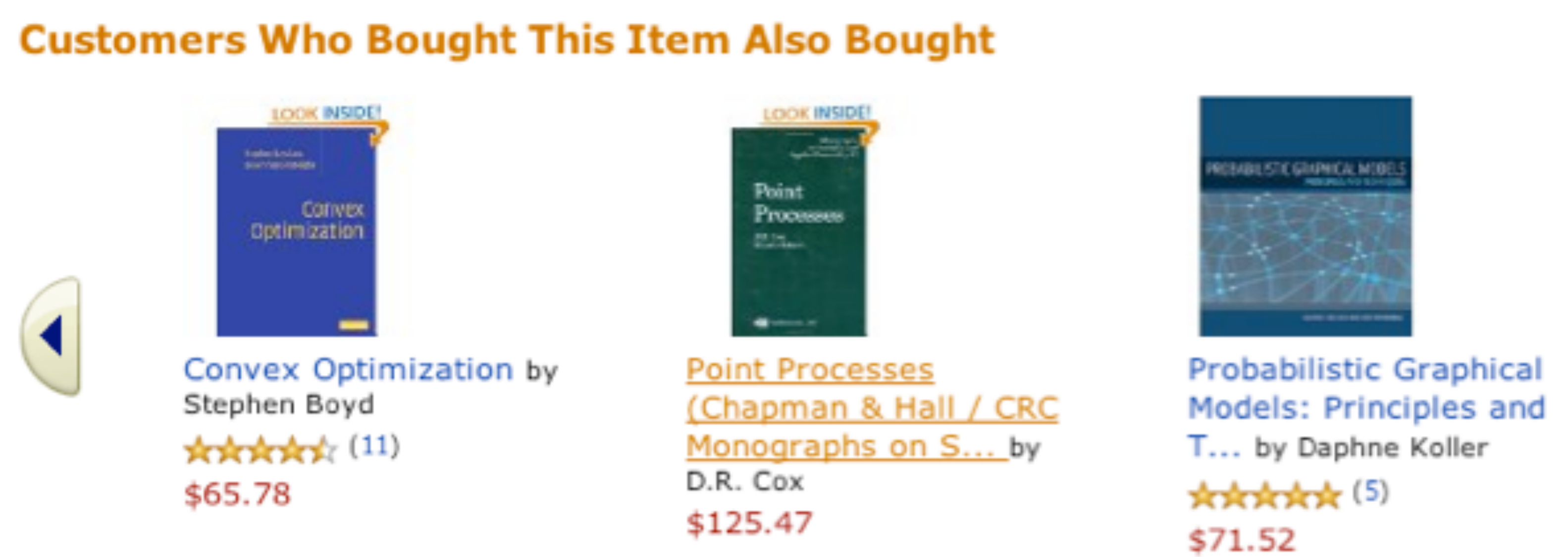
>1B 'identities'

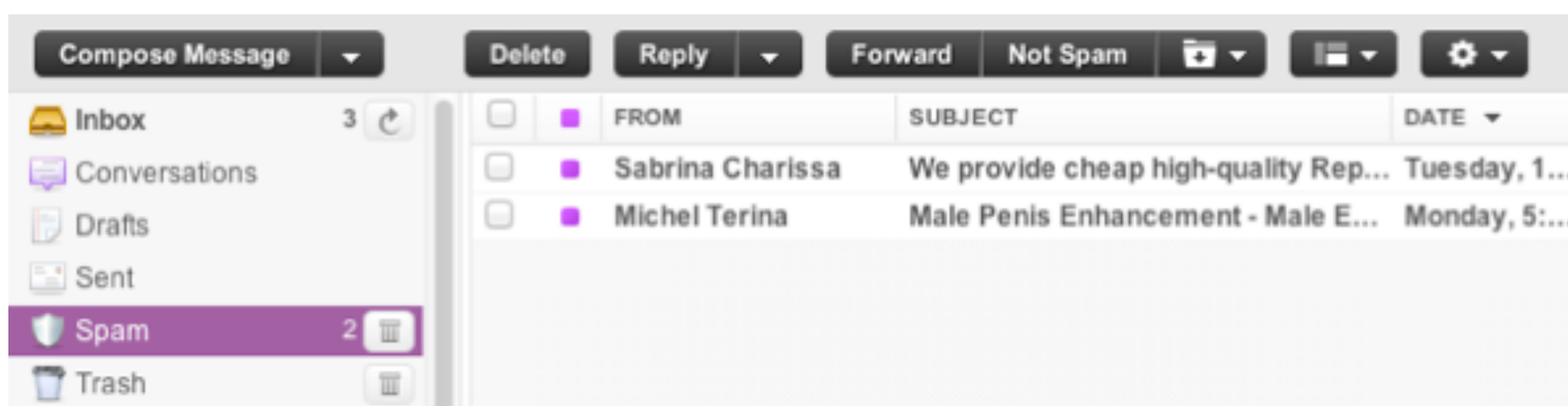
Personalization

- 100-1000M users
  - Spam filtering
  - Personalized targeting & collaborative filtering
  - News recommendation
  - Advertising

- Large parameter space (25 parameters = 100GB)
- Distributed storage (need it on every server)
- Distributed optimization
- Model synchronization



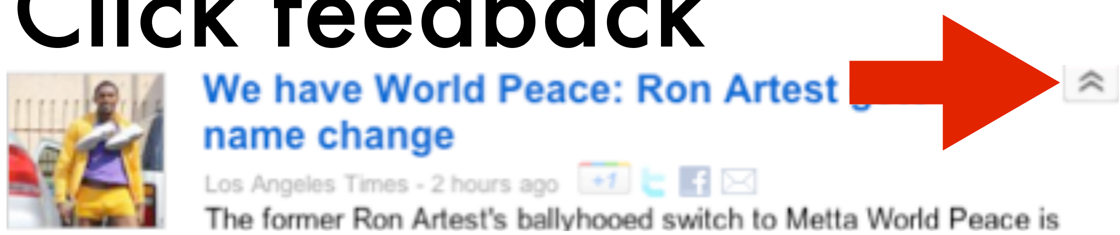




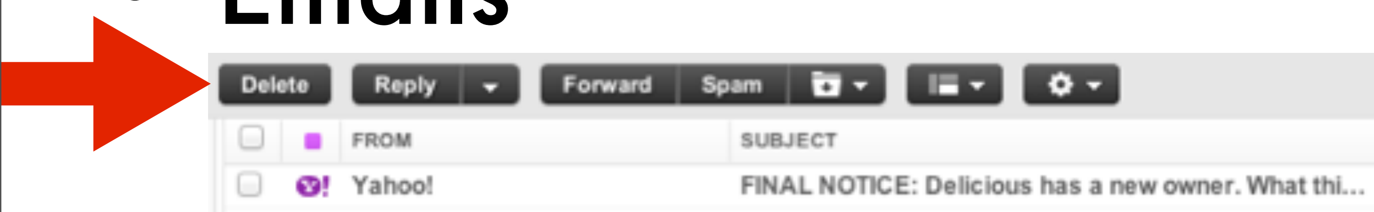
(implicit) Labels — no Labels

- Ads

- Click feedback



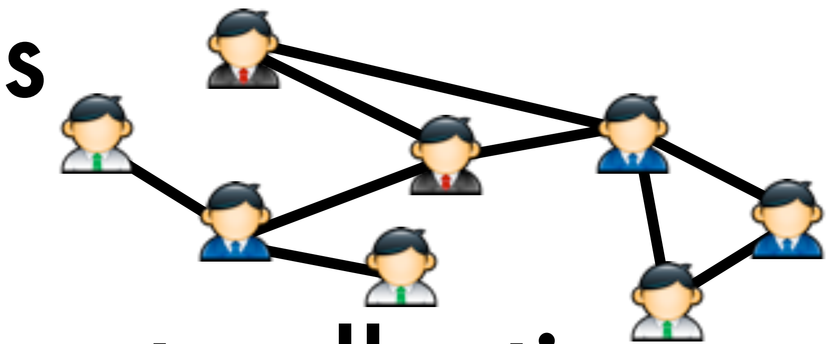
- Emails



- Tags

- Editorial data is very expensive! Do not use!

- Graphs



- Document collections

- Email/IM/Discussions

- Query stream

Hardware

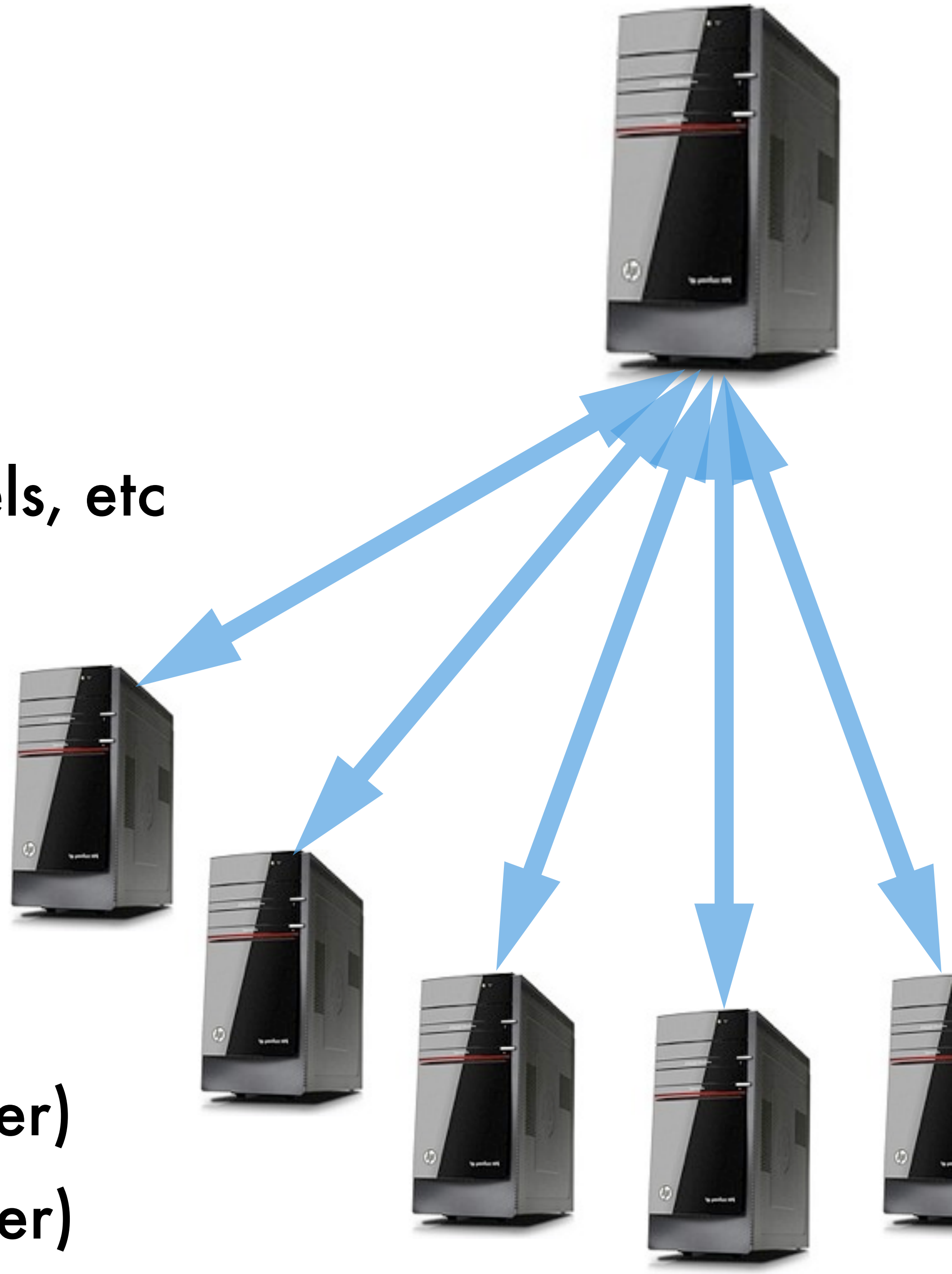
- Mostly commodity hardware
- Server
  - Multicore
  - Soft NUMA (e.g. 2-4 socket Xeons)
  - Plenty of disks
- Racks
  - Common switch per rack
  - 40 odd servers
- Server Center
  - Many racks
  - Big fat master switch(es)
- Faulty (1-100 years MTBF per machine)

What

modular strategy
simple components

1. Distributed Convex Optimization

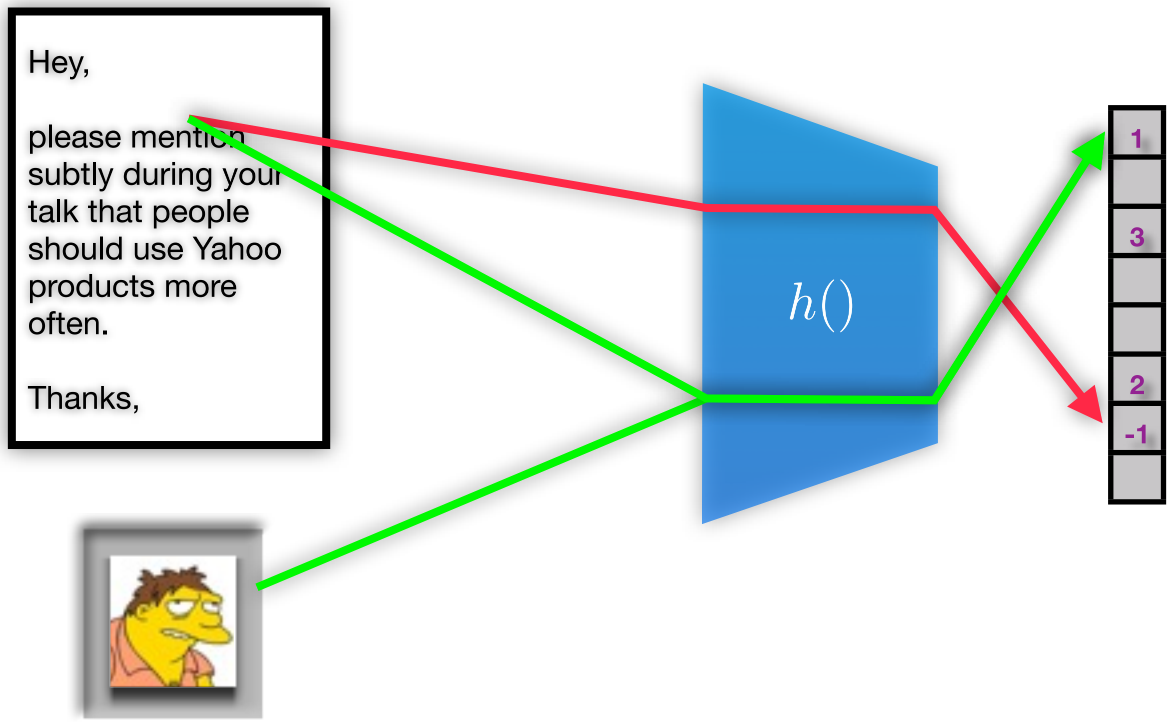
- Supervised learning
  - Classification, regression
  - CRFs, Max-Margin-Markov networks
  - Fully observed graphical models
  - Small modifications for aggregate labels, etc
- Works with MapReduce/Hadoop
- Small number of iterations
- Distributed file system
- Simple & theoretical guarantees
- Plenty of data
  - Parallel batch subgradient solver (cluster)
  - Parallel online solver (multicore & cluster)

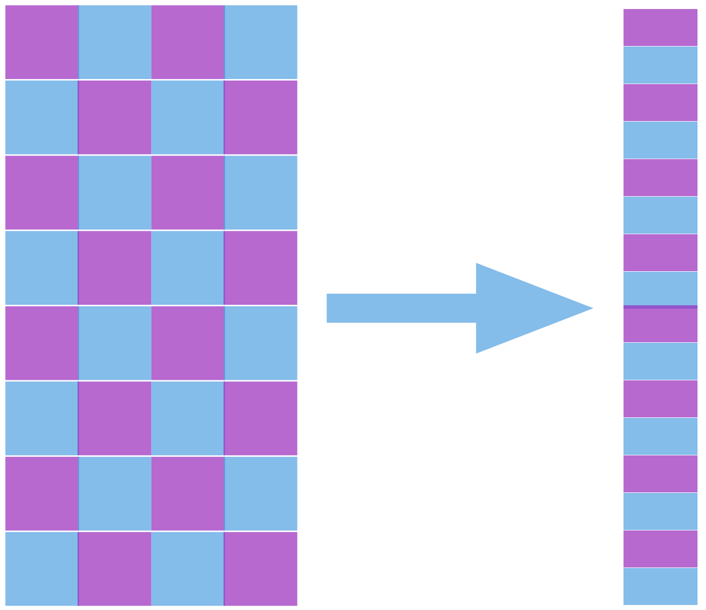


TLSV'07, ZSL'09, TVSL'10, ZWSL'10

2. Parameter Compression

- Personalization
  - Spam filtering
  - News recommendation
  - Collaborative filtering
- String kernels
  - Dictionary free
  - Arbitrary substrings
- Sparse high-dimensional data
- Structured data without pointers
- Fixed memory footprint
- Simple & theoretical guarantees

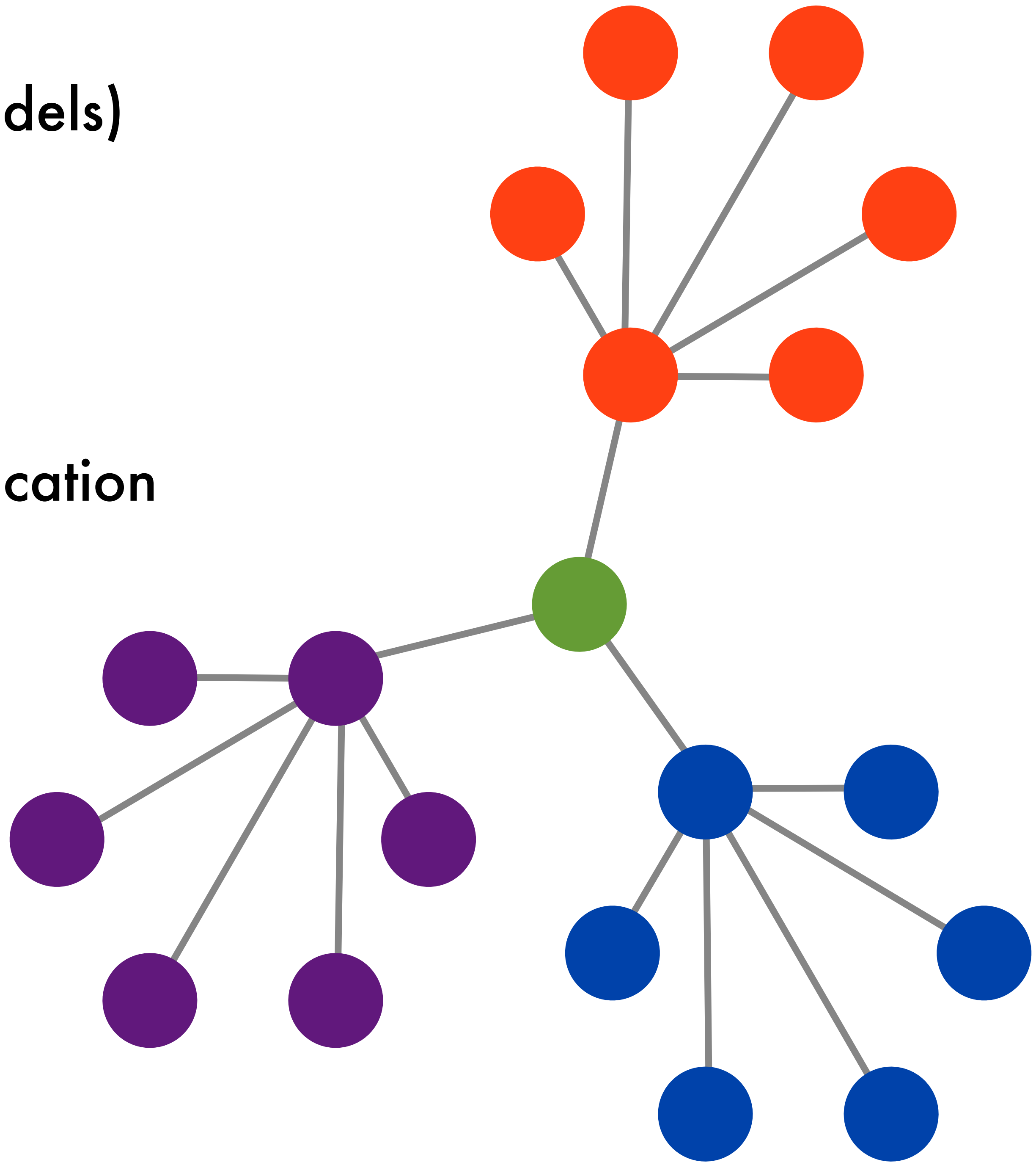




SPDLSSV'09, WDALS'09, KSW'10, PSCBN'10, YLSZZ'11, ASTV'12

3. Distributed Storage, Sampling and Synchronization

- Latent variable models with large state
  - Joint statistics (e.g. clustering, topic models)
  - Local state (attached to evidence)
  - Too big to store on a single machine
- Distributed Storage
  - Asynchronous computation & communication
  - Maps to network topology
  - Consistent hashing for scalability
  - Out of core storage of local state
- Distributed Gibbs sampler
  (10B latent variables, 1000 machines)



SN'10, AAJS'11, LAS'11, AAGS'12

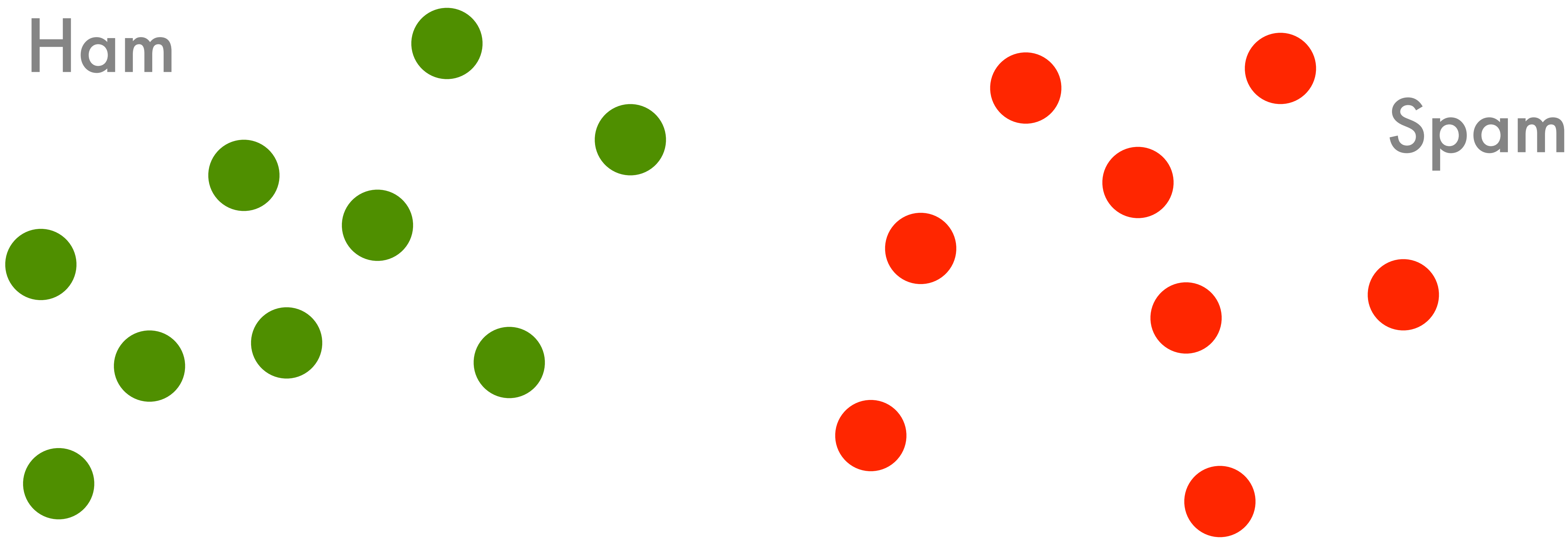
Design Principles

- Must scale (essentially linearly) with
  - Amount of data
  - Number of machines
  - Problem complexity (parameter space)
- Composable techniques
- Accommodate more complex model with more data
  - No 100 cluster model on 1B objects
  - Bayesian Nonparametrics
  - No 1000 parameter classifier on 1M data
  - Increased bit resolution for hashing
  - Throughput on simple models and 1CPU meaningless

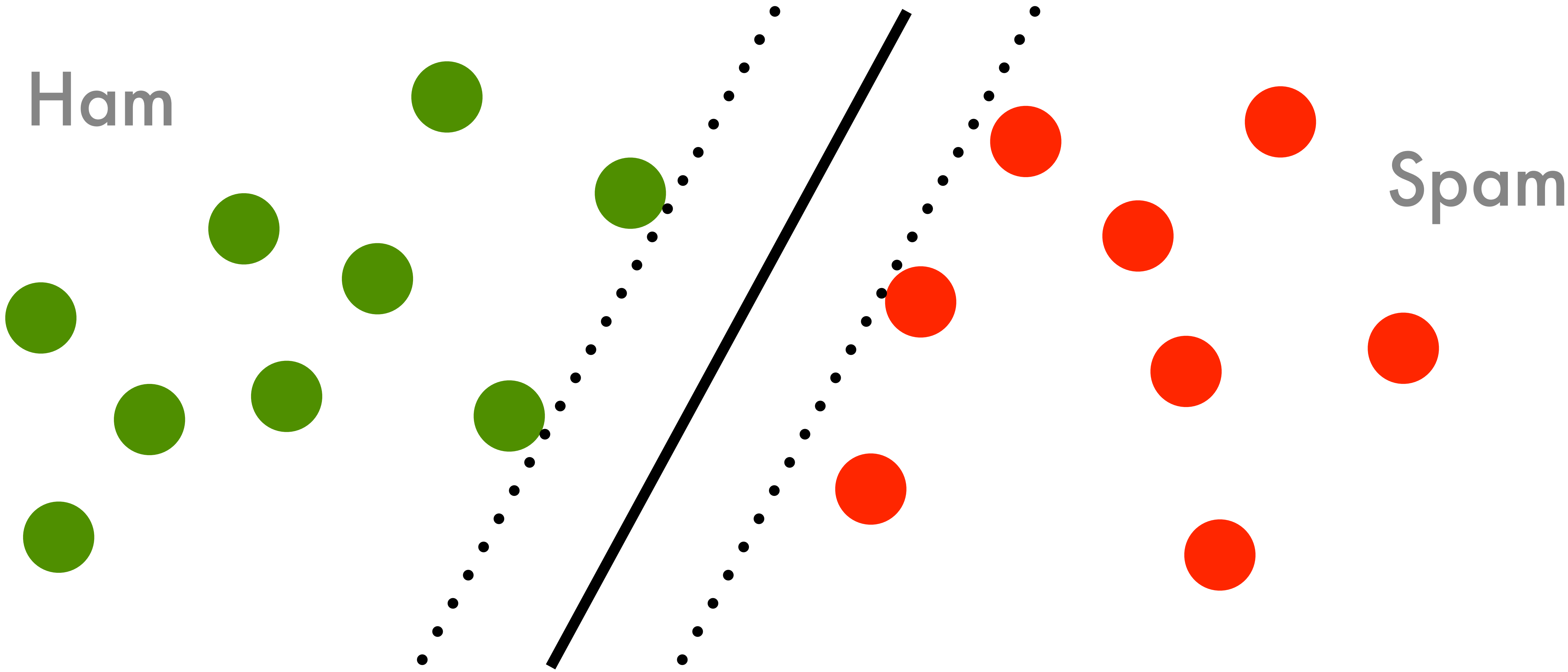
How

- Distributed Batch Convex Optimization
- Distributed Online Convex Optimization
- Parameter Compression
- Distributed Sampling and Synchronization

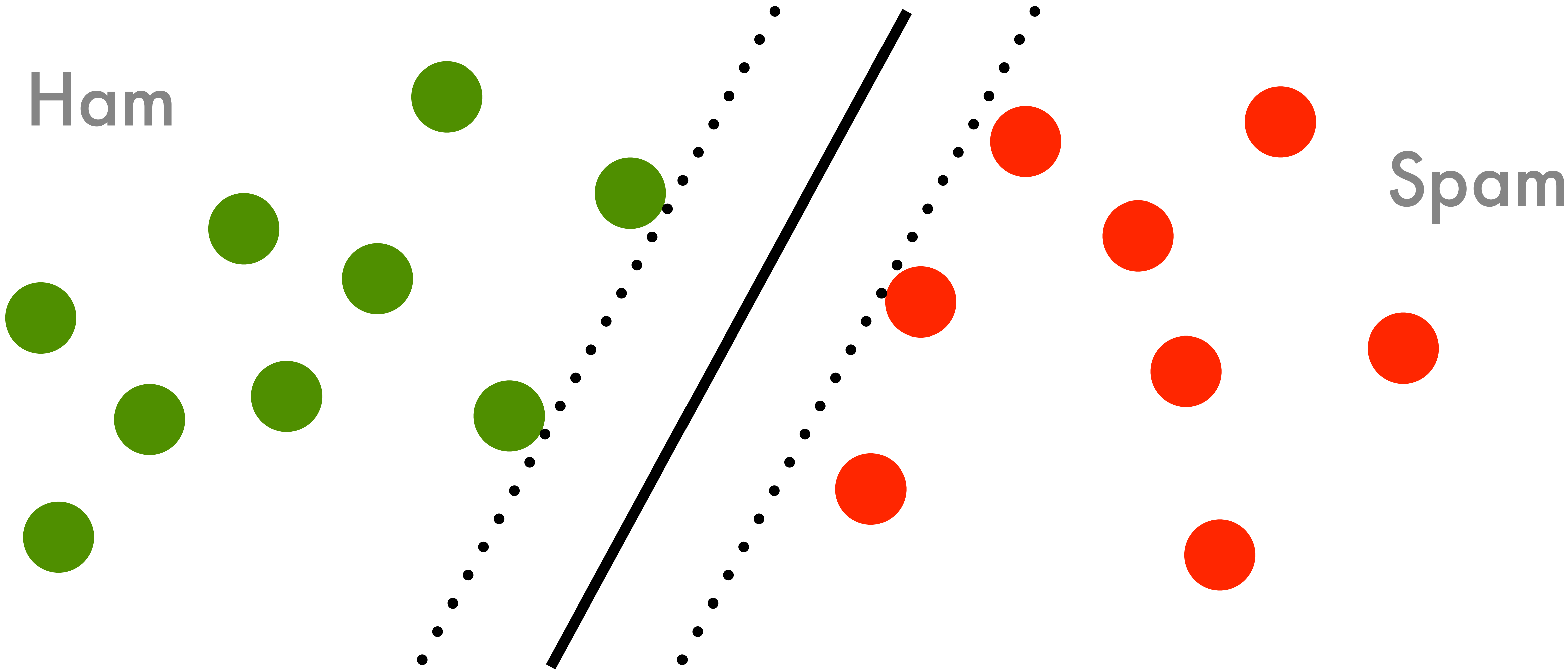
Large Margin Classification



Large Margin Classification



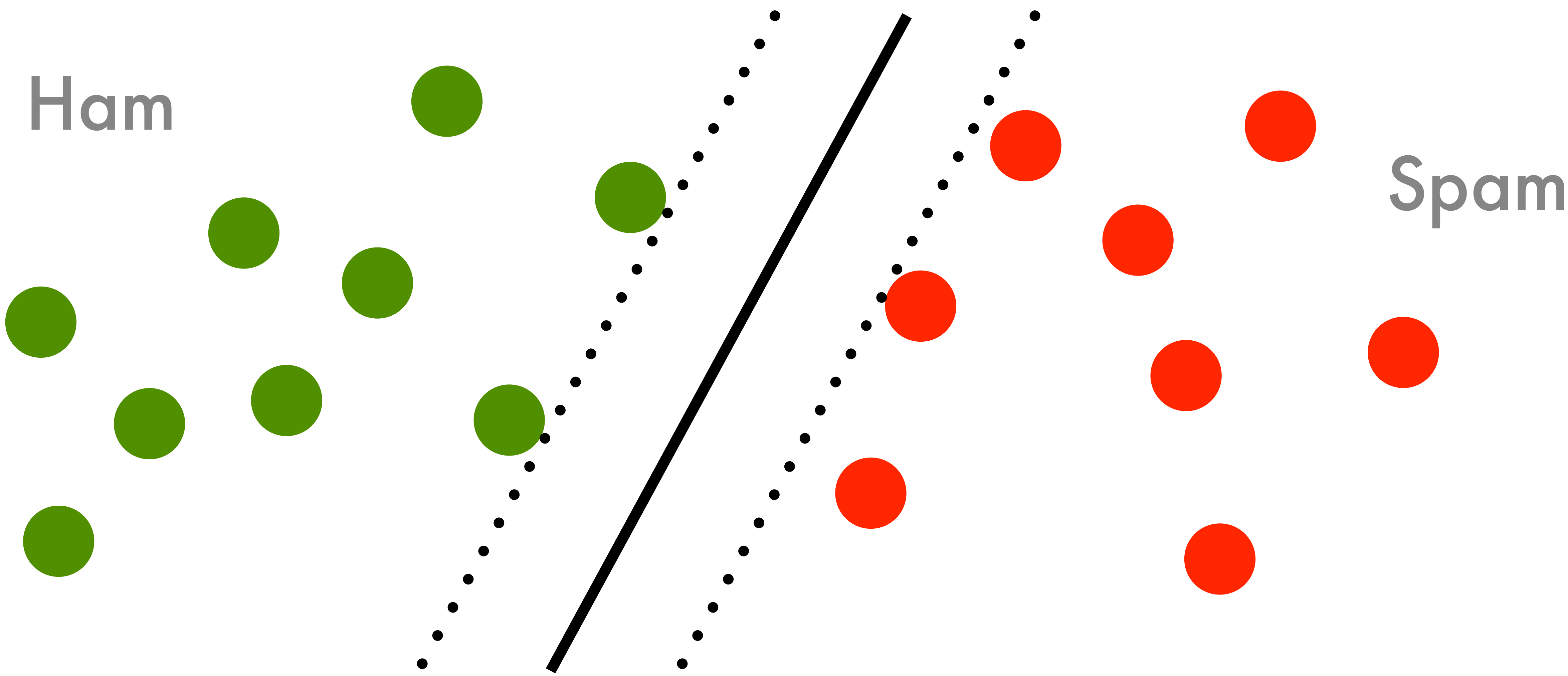
Large Margin Classification



$$\underset{w,b,\xi}{\text{minimize}} \;\; \frac{1}{m}\sum_{i=1}^{m}\xi_i + \frac{\lambda}{2}\left\|w\right\|^2$$

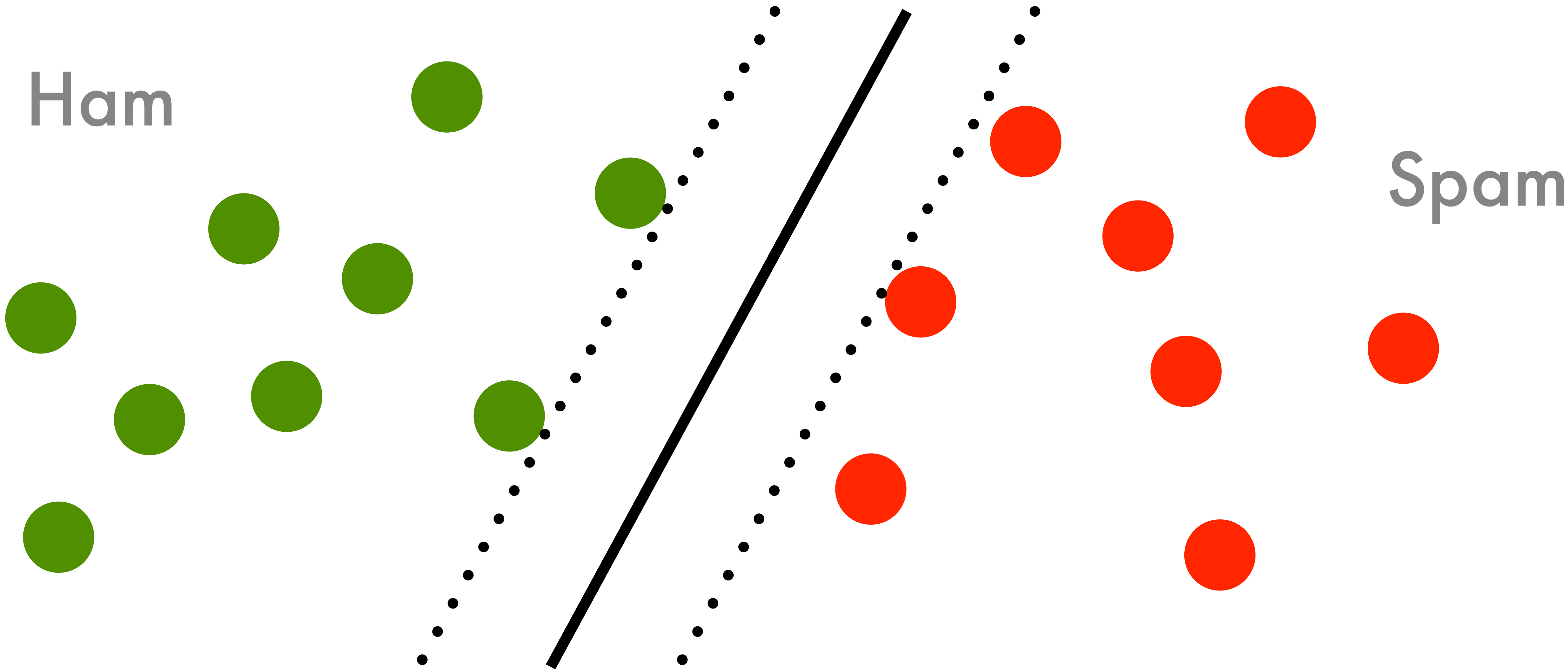
$$\text{subject to } y_i\left[\langle w, x_i\rangle + b\right] \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Large Margin Classification



$$\underset{w,b}{\text{minimize}}\ \frac{1}{m}\sum_{i=1}^{m}\max\left[0, 1 - y_i\left[\langle w, x_i\rangle + b\right]\right] + \frac{\lambda}{2}\left\|w\right\|^2$$

Large Margin Classification



$$\underset{w,b}{\text{minimize}} \; \frac{1}{m}\sum_{i=1}^{m} l(x_i, y_i, w) \qquad + \frac{\lambda}{2}\Omega[w]$$

Regularized Risk Functional

$$\underset{w}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} l(x_i, y_i, w) + \lambda \Omega[w]$$

decomposable

relatively simple

SVM, regression, sequence annotation, ranking and recommendation, image annotation, gene finding, face detection, density estimation, novelty detection

quadratic penalty (l2)

sparsity penalty (l1)

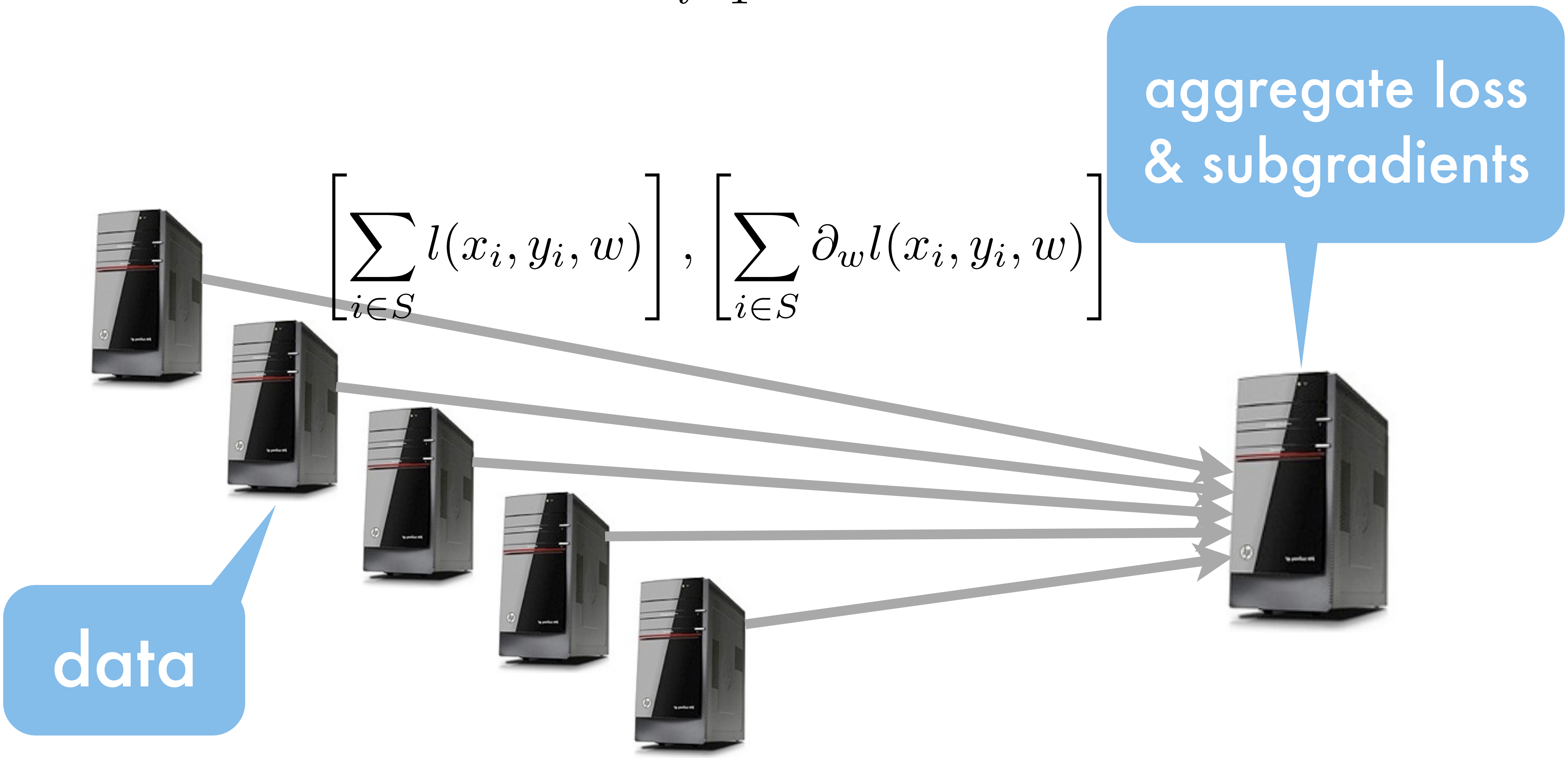
hyperkernels

group lasso

Regularized Risk Functional

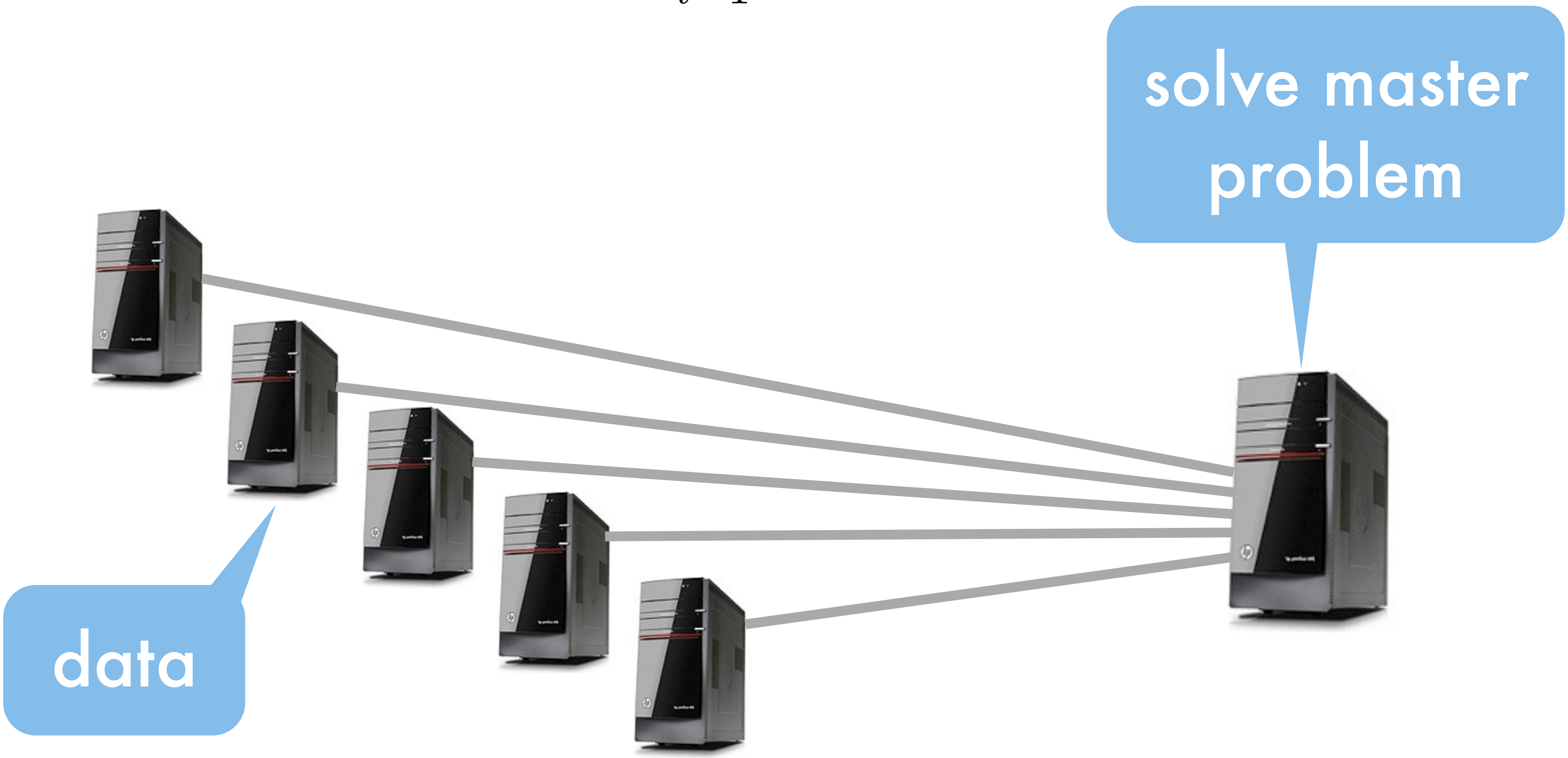
$$\underset{w}{\text{minimize}}\ \frac{1}{m}\sum_{i=1}^{m} l(x_i, y_i, w) + \lambda\Omega[w]$$

$$\left[\sum_{i\in S} l(x_i, y_i, w)\right], \left[\sum_{i\in S} \partial_w l(x_i, y_i, w)\right]$$



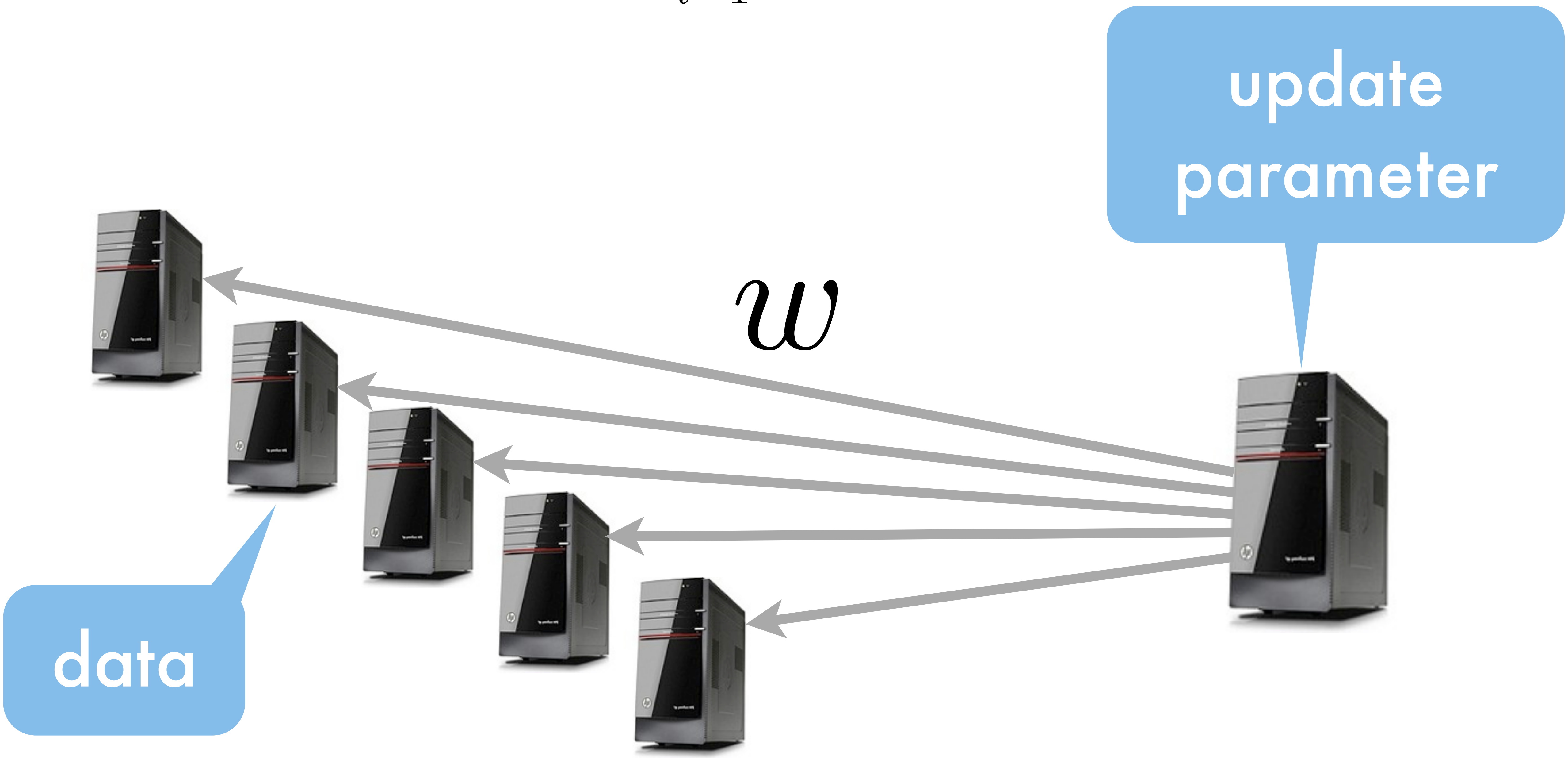
Regularized Risk Functional

$$\underset{w}{\text{minimize}}\ \frac{1}{m}\sum_{i=1}^{m} l(x_i, y_i, w) + \lambda\Omega[w]$$

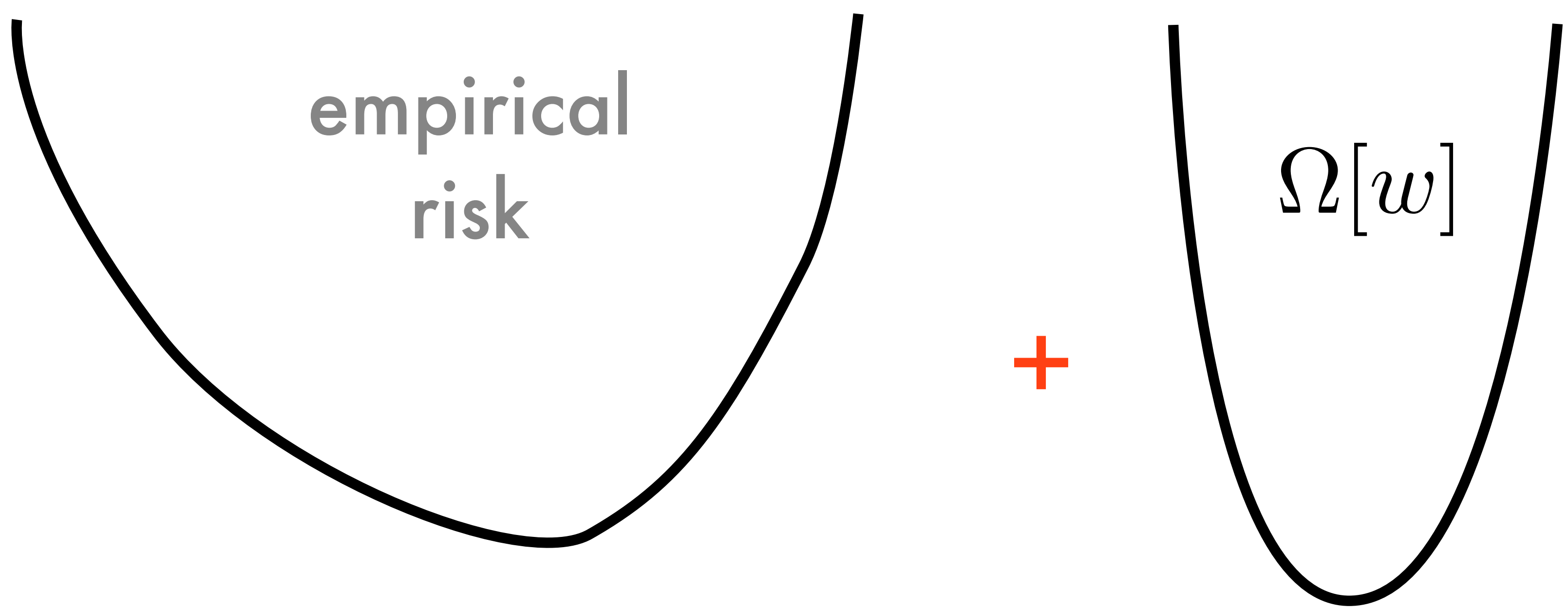


Regularized Risk Functional

$$\underset{w}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} l(x_i, y_i, w) + \lambda \Omega[w]$$

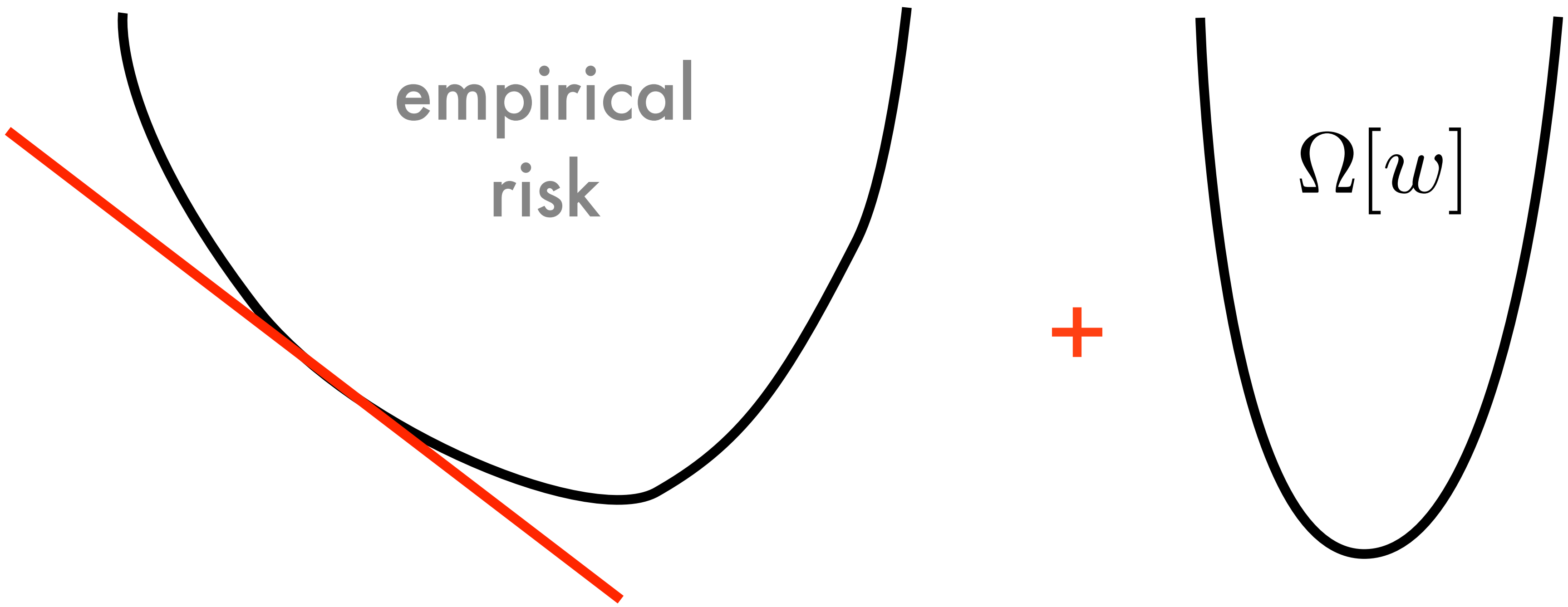


Bundle Method Solver



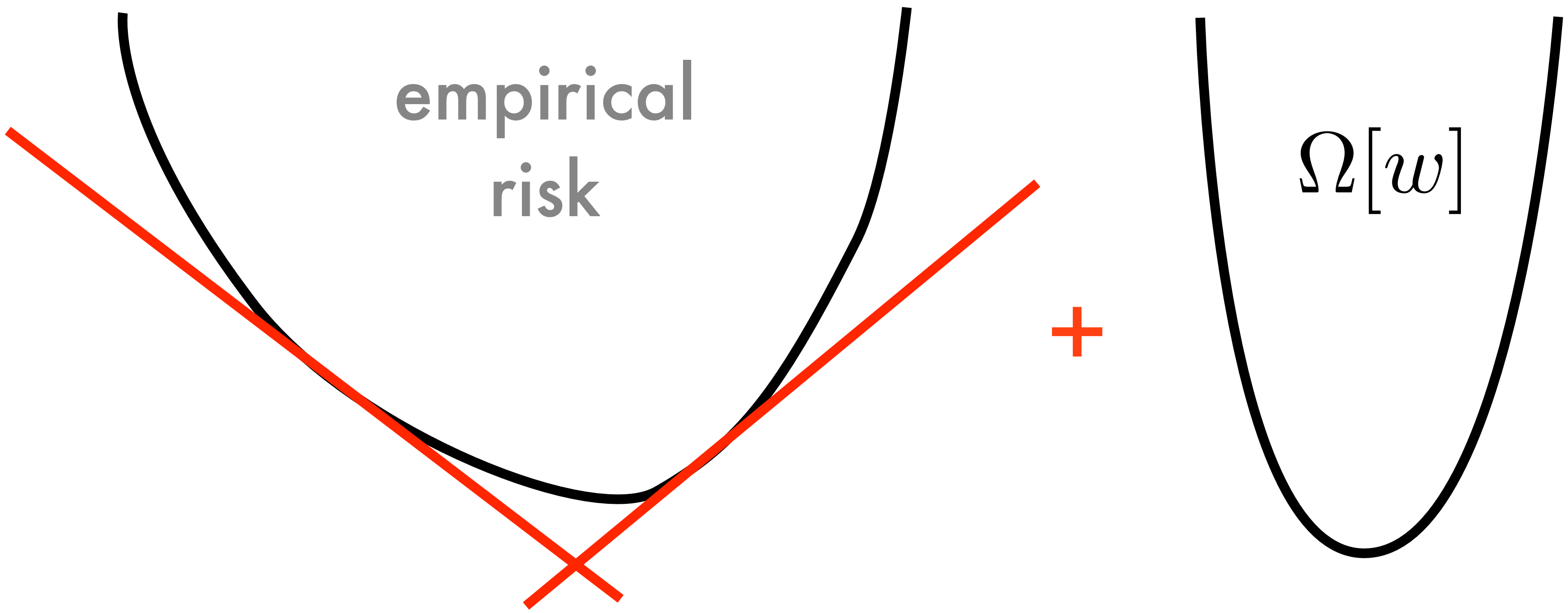
$$\underset{w}{\text{minimize}} \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

Bundle Method Solver



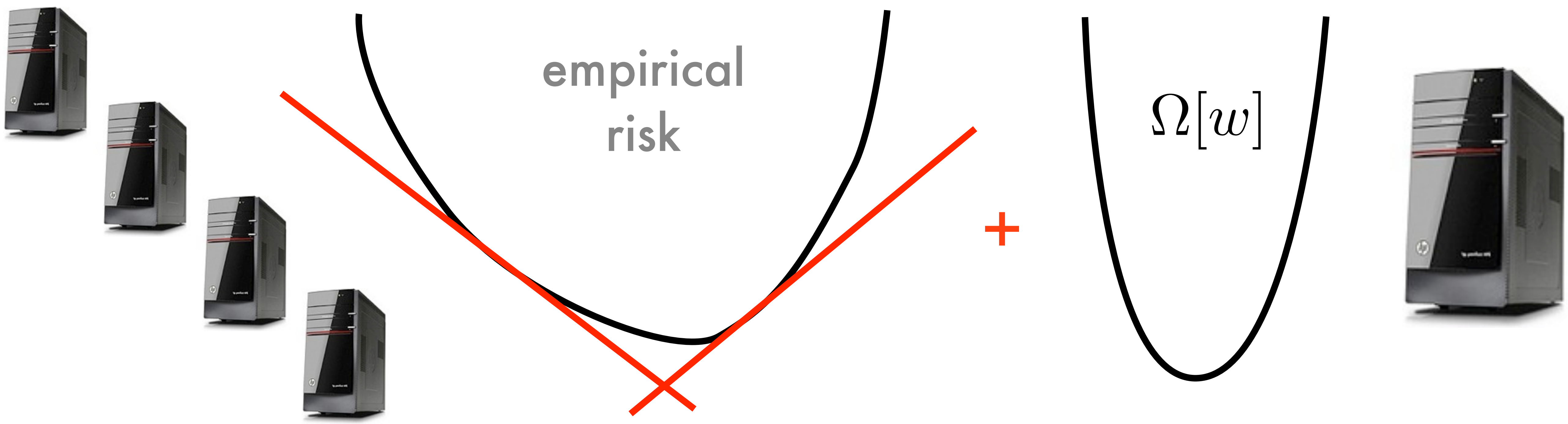
$$\underset{w}{\text{minimize}} \quad \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

Bundle Method Solver



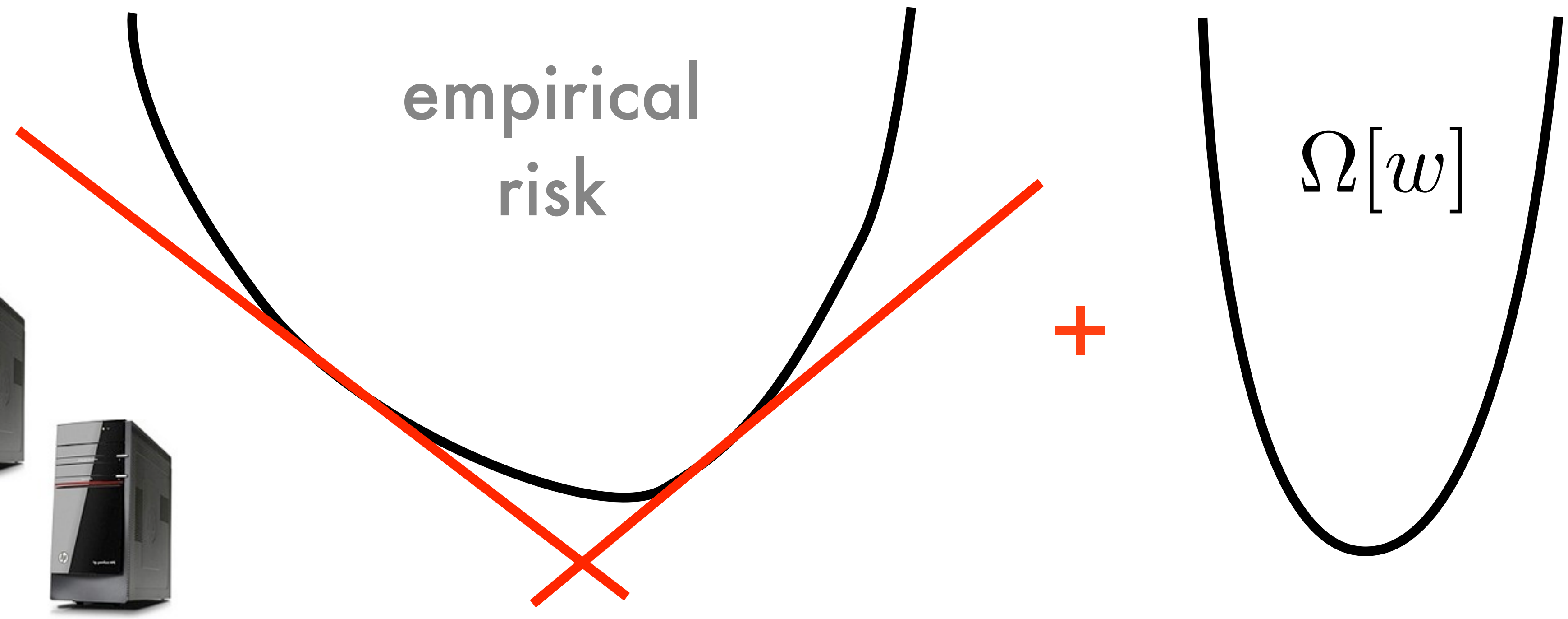
$$\underset{w}{\text{minimize}} \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

Bundle Method Solver



$$\underset{w}{\text{minimize}} \quad \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

Bundle Method Solver



$$\underset{w}{\text{minimize}} \quad \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

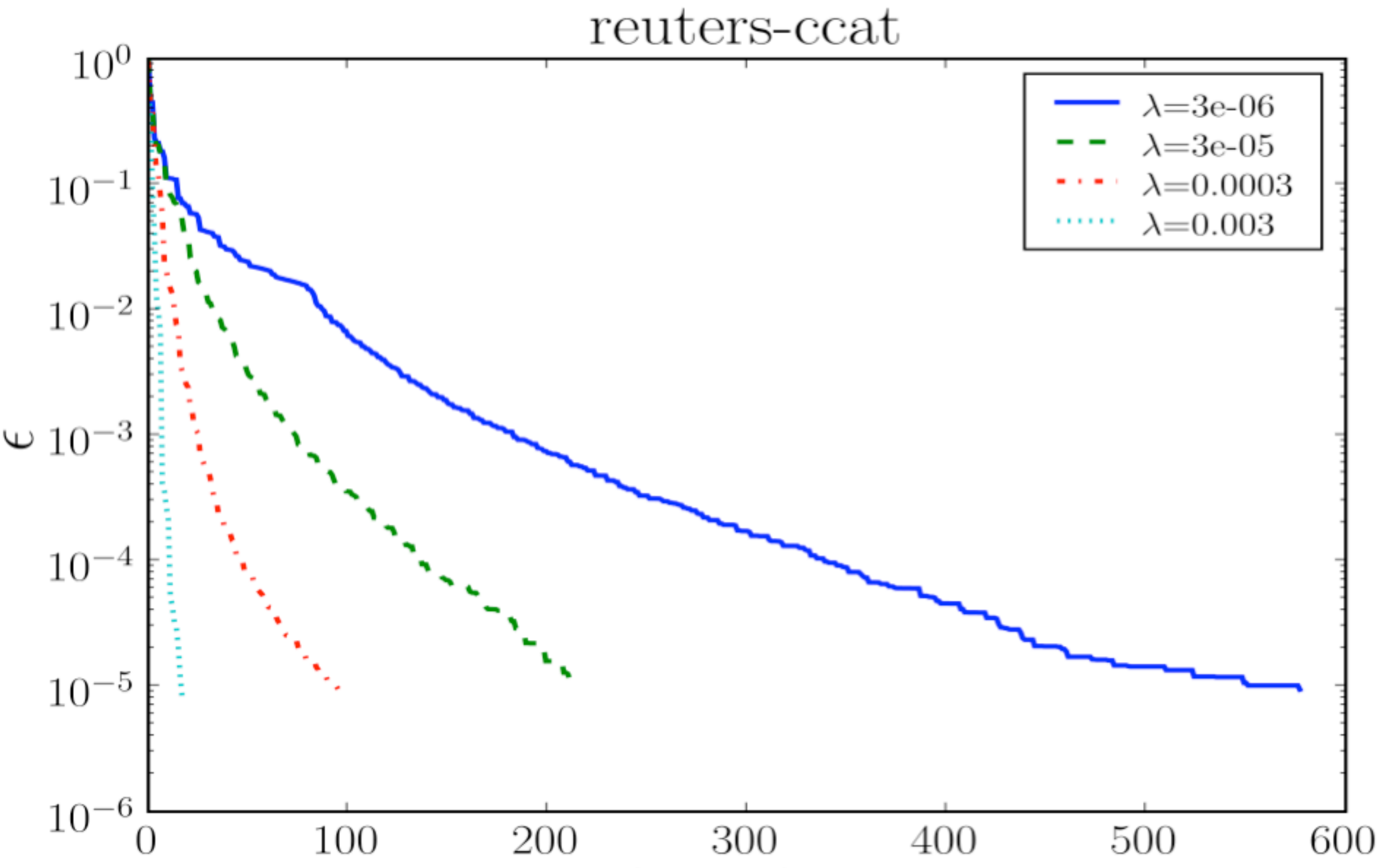
- starting point w_0
- compute first order Taylor approximation (g_i, b_i)
- solve optimization problem
- repeat

Bundle Method Solver

$$\underset{w}{\text{minimize}} \left[\max_i \langle g_i, w\rangle + b_i\right] + \frac{\lambda}{2}\Omega[w]$$

- Empirical risk certificates (at each iteration)
  - Upper bound on risk via first order Taylor approximation.
  - Lower bound on risk after solving optimization problem
- Convergence guarantees (worst case)
  (loss bound L, gradient bound G, Hessian bound H)
  - Generic iteration bound $\log \frac{\lambda L}{G^2} + \frac{8G^2}{\lambda\epsilon}$
  - For bounded Hessian $\log \frac{\lambda L}{G^2} + \frac{4}{\lambda}[1 + H \log 2\epsilon]$

Bundle Method Solver



Bundle Method Solver

- Alternatives
  - Use BFGS in outer loop
  - Gradient with line search
  - Dual Subgradient (Boyd et al.)
    - Theoretically elegant
    - Slow convergence due to dual gradient descent
  - FISTA (better for l_1 sparsity penalty)
- Problems with batch solvers
  - requires 50 passes through dataset
  - requires smooth regularizer for fast convergence

How

- Distributed Batch Convex Optimization
- Distributed Online Convex Optimization
- Parameter Compression
- Distributed Sampling and Synchronization

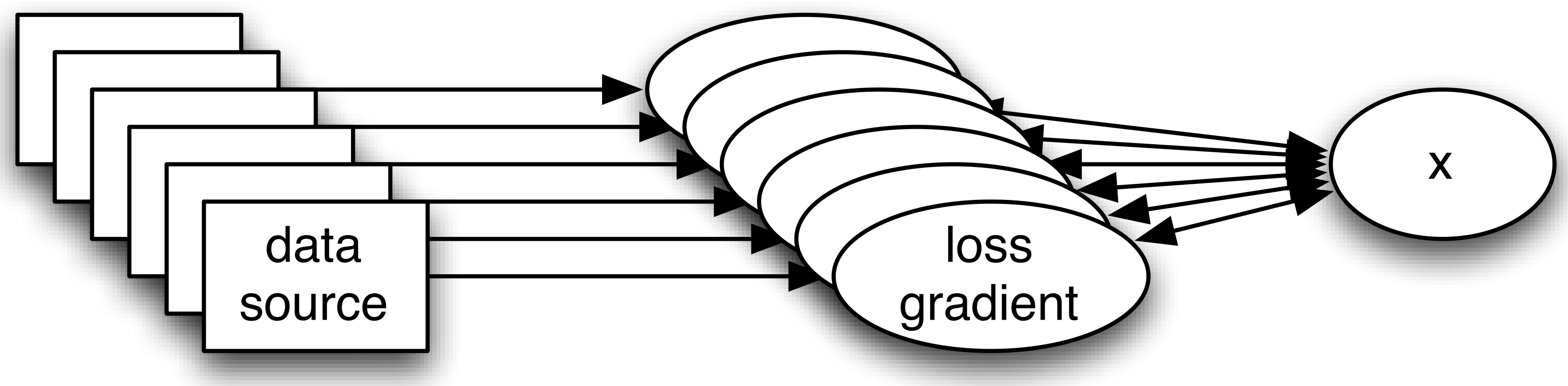
Multicore

Online Learning

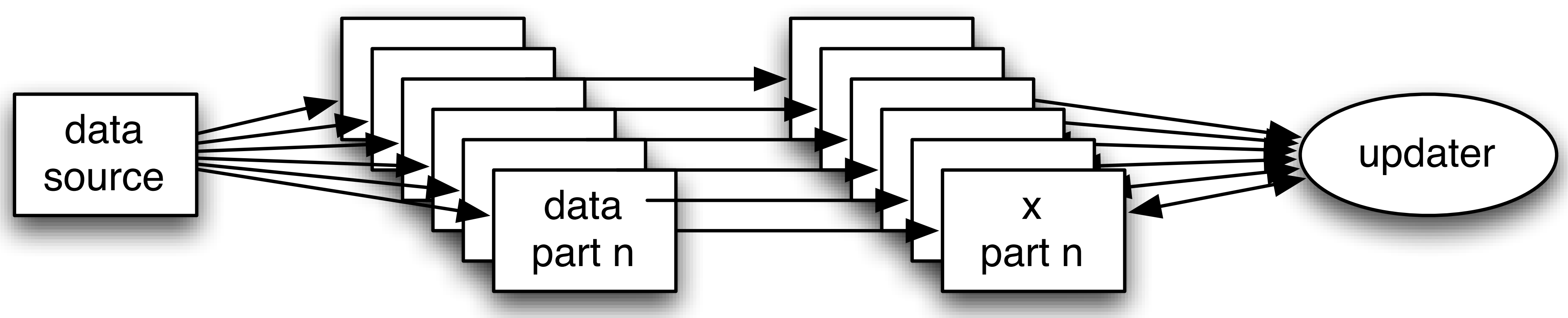
- **General Template**
  - Get instance
  - Compute instantaneous gradient
  - Update parameter vector
- **Problems**
  - Sequential execution (single **core**)
  - CPU **core** speed is no longer increasing
  - Disk/network bandwidth: 300GB/h
  - Does **not** scale to TBs of data

Parallel Online Templates

- **Data parallel**



- **Parameter parallel**



Delayed Updates

- **Data parallel**
  - n processors compute gradients
  - delay is n-1 between gradient computation and application
- **Parameter parallel**
  - delay between partial computation and feedback from joint loss
  - delay logarithmic in processors

Delayed Updates

- Optimization Problem

$$\underset{w}{\text{minimize}} \sum_i f_i(w)$$

- Algorithm

$$\begin{aligned}
&\textbf{Input: } \text{scalar } \sigma > 0 \text{ and delay } \tau \\
&\textbf{for } t = \tau + 1 \textbf{ to } T + \tau \textbf{ do} \\
&\quad \text{Obtain } f_t \text{ and incur loss } f_t(w_t) \\
&\quad \text{Compute } g_t := \nabla f_t(w_t) \text{ and set } \eta_t = \frac{1}{\sigma(t-\tau)} \\
&\quad \text{Update } w_{t+1} = w_t - \eta_t g_{t-\tau} \\
&\textbf{end for}
\end{aligned}$$

Adversarial Guarantees

- **Linear function classes**

$$\mathbf{E}[f_i(w)] \leq 4RL\sqrt{\tau T}$$

Algorithm converges no worse than with serial execution. Up to a factor of 4 as tight.

- **Strong convexity**

$$R[X] \leq \lambda \tau R + \left[\tfrac{1}{2} + \tau\right] \frac{L^2}{\lambda} (1 + \tau + \log T)$$

Each loss function is strongly convex with modulus λ. Constant offset depends on the degree of parallelism.

- **Bounds are tight**

Adversary sends same instance τ times

Nonadversarial Guarantees

- **Lipschitz continuous loss gradients**

$$\mathbf{E}[R[X]] \leq \left[28.3 R^2 H + \frac{2}{3} R L + \frac{4}{3} R^2 H \log T\right] \tau^2 + \frac{8}{3} R L \sqrt{T}.$$

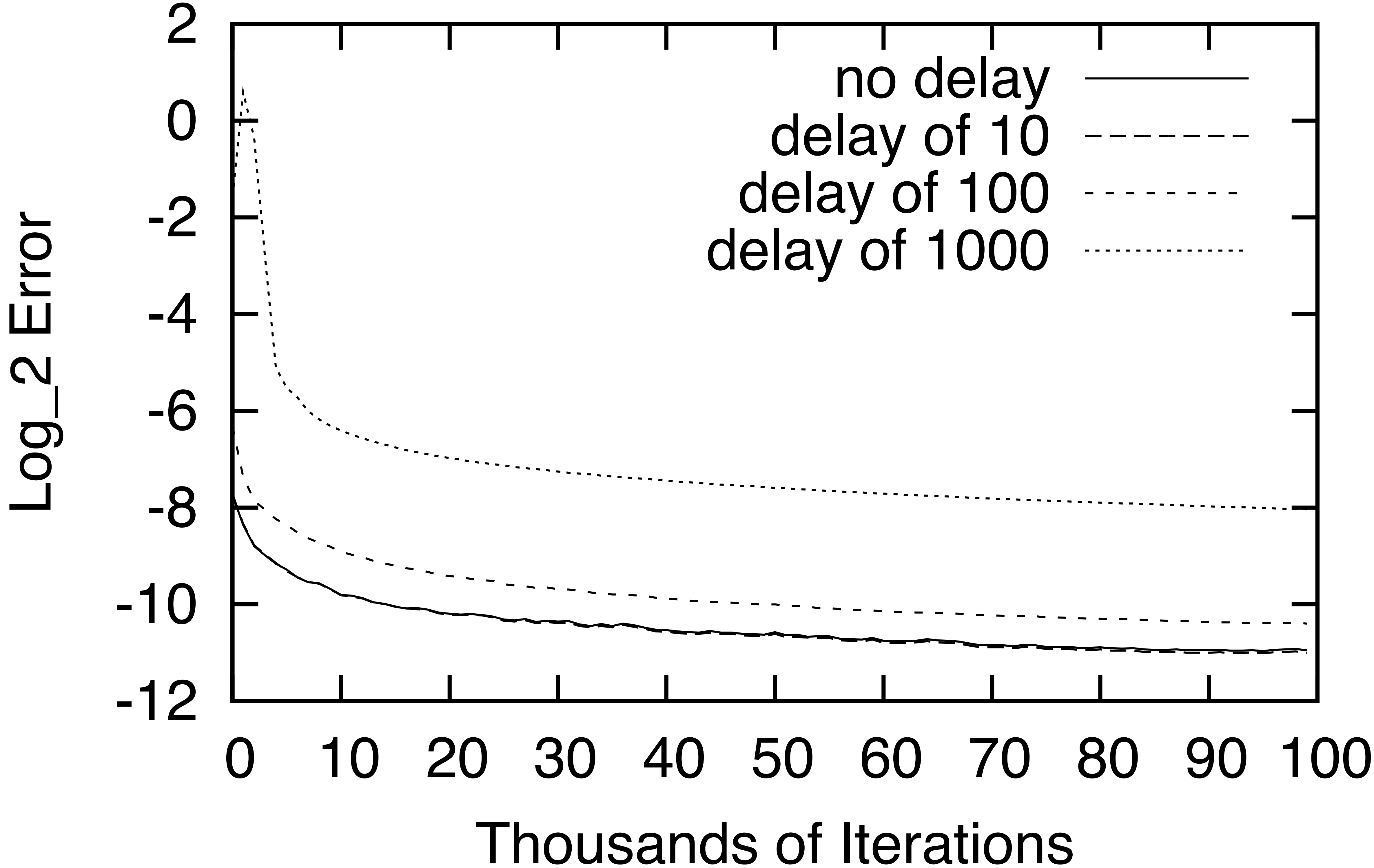
Rate **no longer** depends on amount of parallelism

- **Strong convexity and Lipschitz gradients**

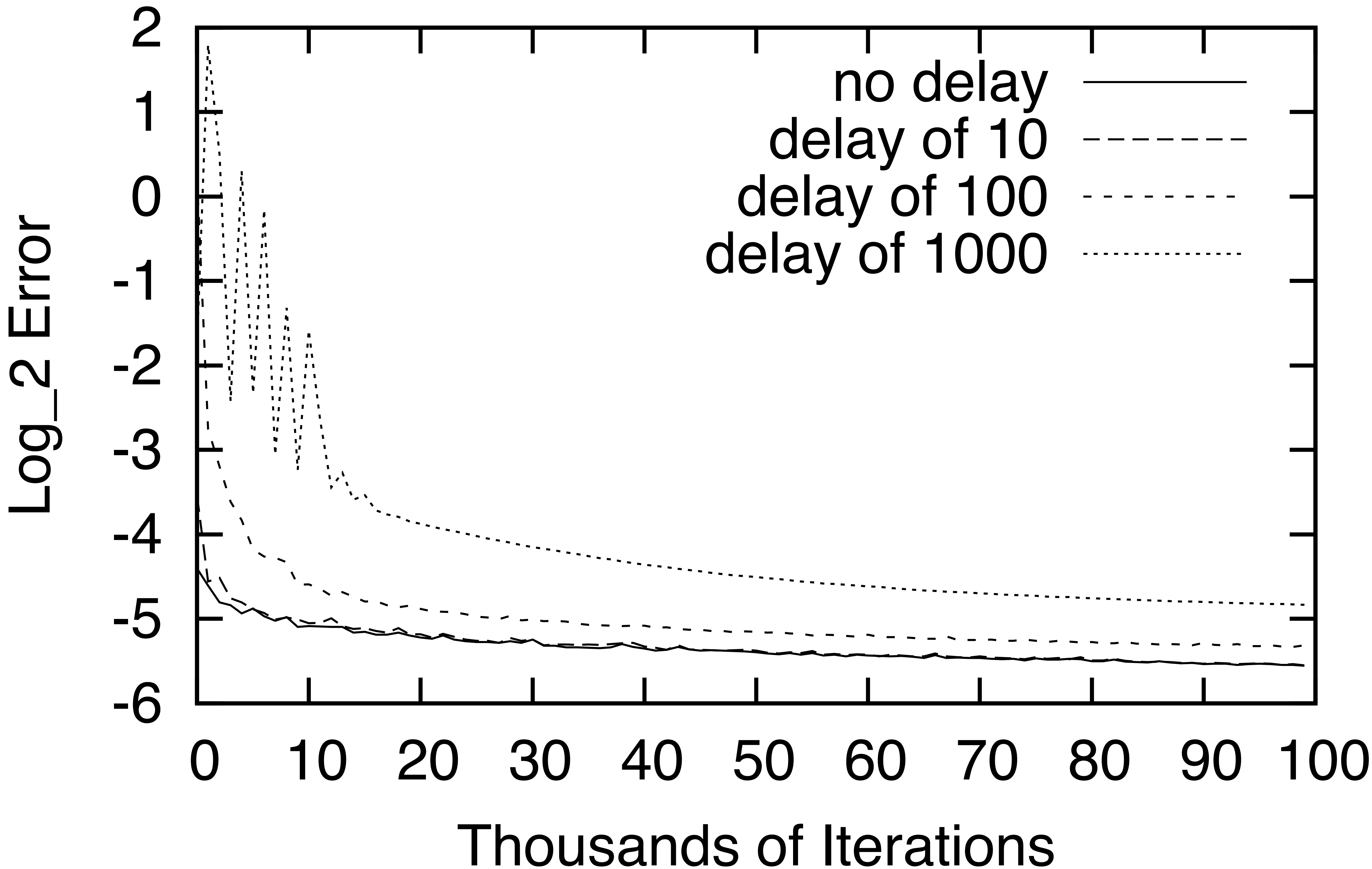
$$\mathbf{E}[R[X]] \leq O(\tau^2 + \log T)$$

This only works when the objective function is very close to a parabola (upper and lower bound)

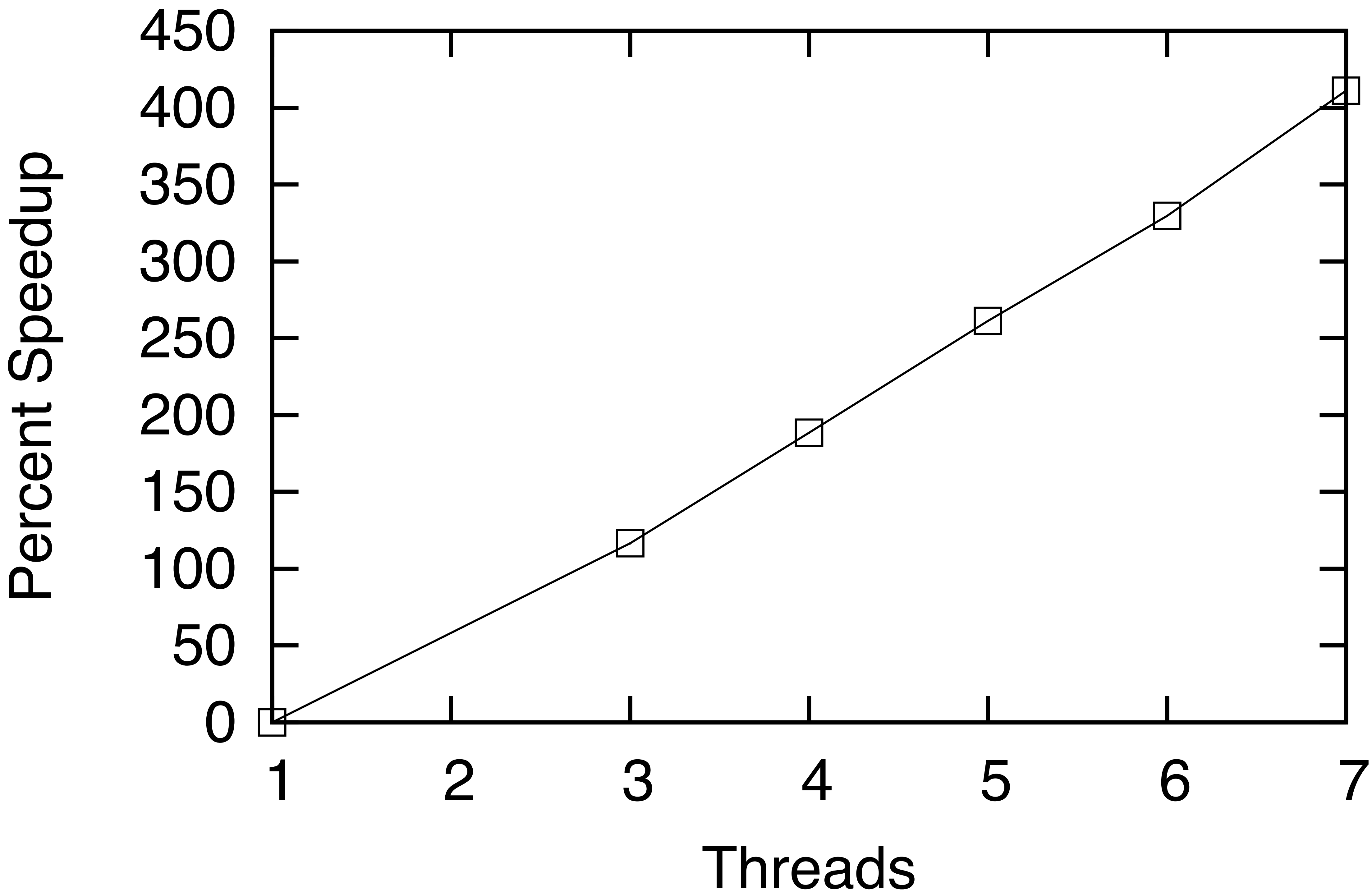
Convergence on TREC



Convergence on Y!Data



Speedup on TREC



Cluster

MapReduce variant

- **Idiot proof simple algorithm**
  - Perform stochastic gradient on each computer for a random subset of the data (drawn with replacement)
  - Average parameters
- **Benefits**
  - No communication during optimization
  - Single pass MapReduce
  - Latency is not a problem
  - Fault tolerant (we oversample anyway)

Guarantees

- **Requirements**
  - Strongly convex loss
  - Lipschitz continuous gradient
- **Theorem**

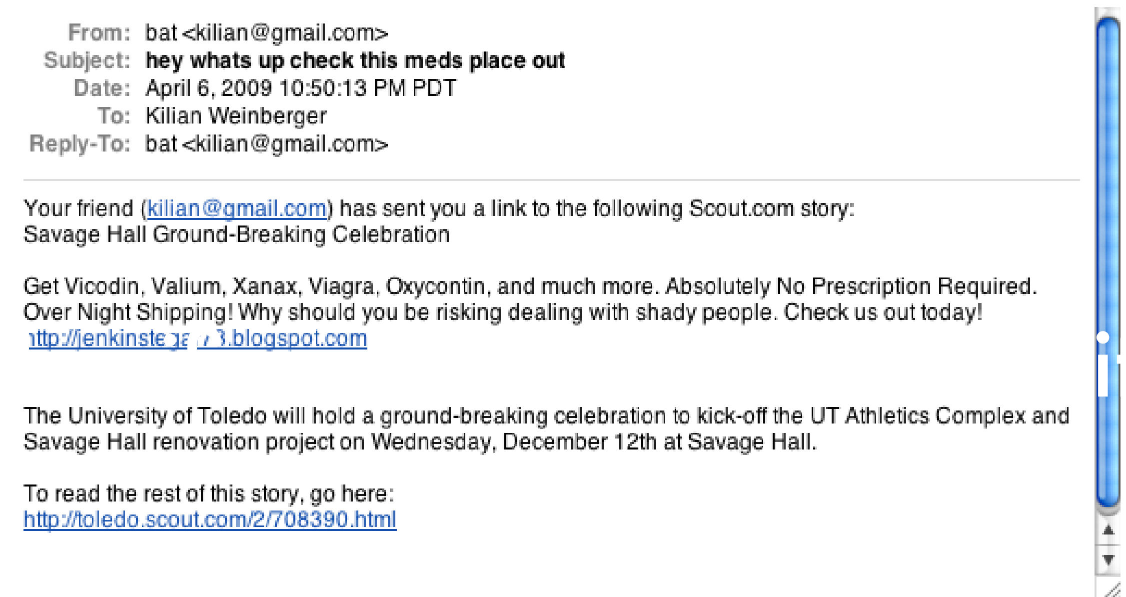
$$\mathbf{E}_{w \in D_\eta^{T,k}}[c(w)] - \min_w c(w) \leq \frac{8\eta G^2}{\sqrt{k\lambda}} \sqrt{\|\partial c\|_L} + \frac{8\eta G^2 \|\partial c\|_L}{k\lambda} + (2\eta G^2)$$

  - Not sample size dependent
  - Regularization limits parallelization
  - For runtime $T = \frac{\ln k - (\ln \eta + \ln \lambda)}{2\eta\lambda}$

How

- Distributed Batch Convex Optimization
- Distributed Online Convex Optimization
- Parameter Compression
- Distributed Sampling and Synchronization

Spam Classification



From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required.
Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
http://jenkinste[illegible].blogspot.com

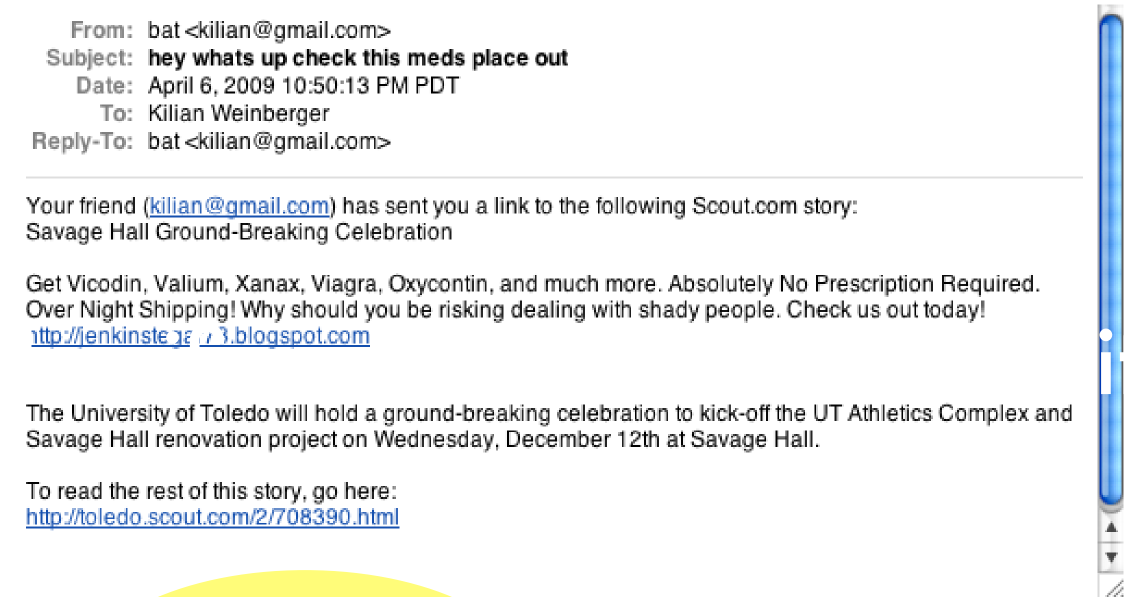
The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html





Spam Classification

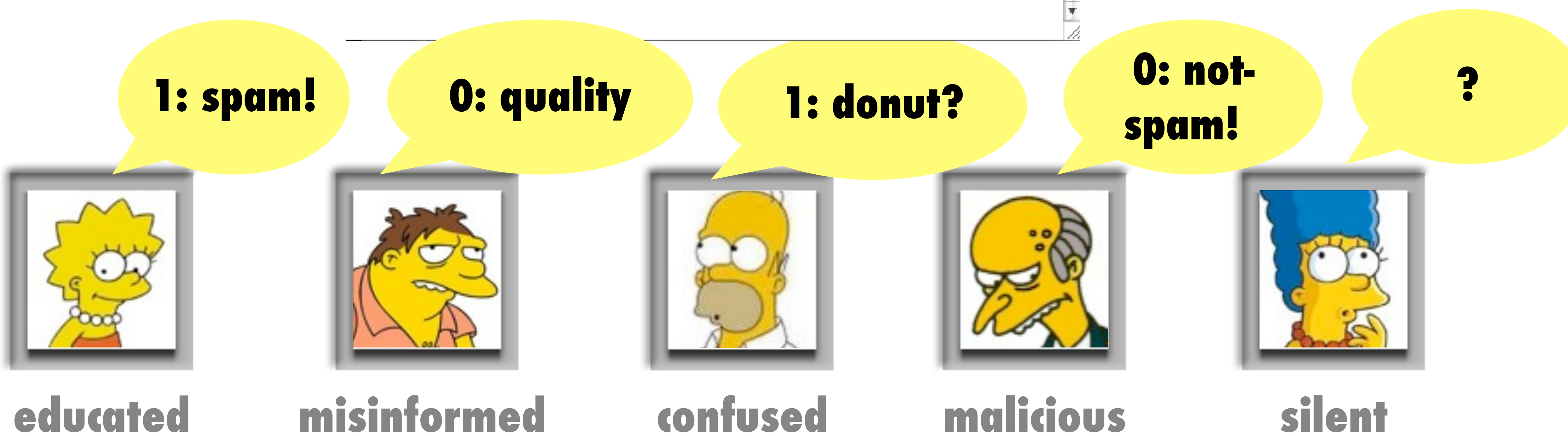

From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

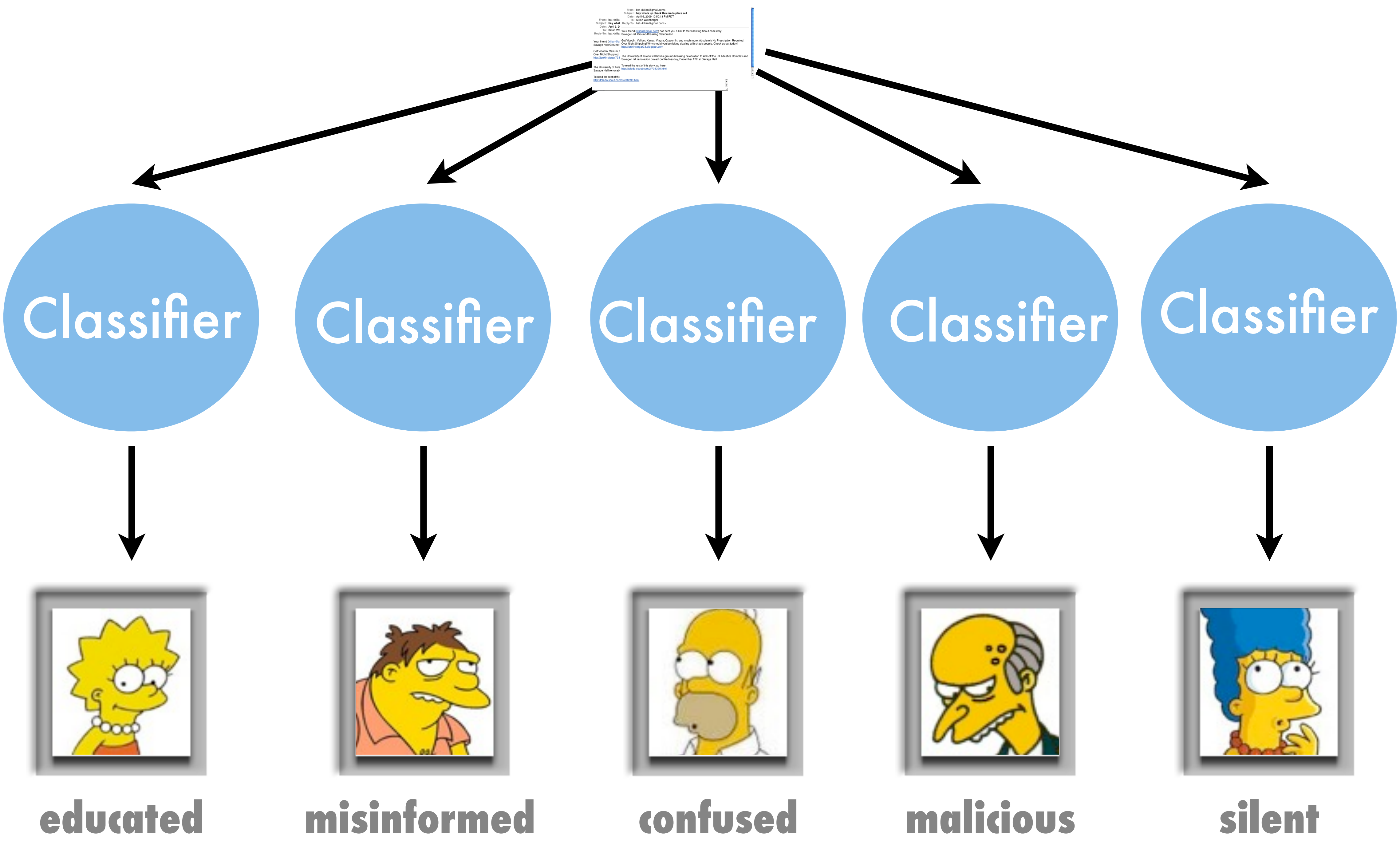
Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required.
Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
http://jenkinste [illegible] 3.blogspot.com

The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

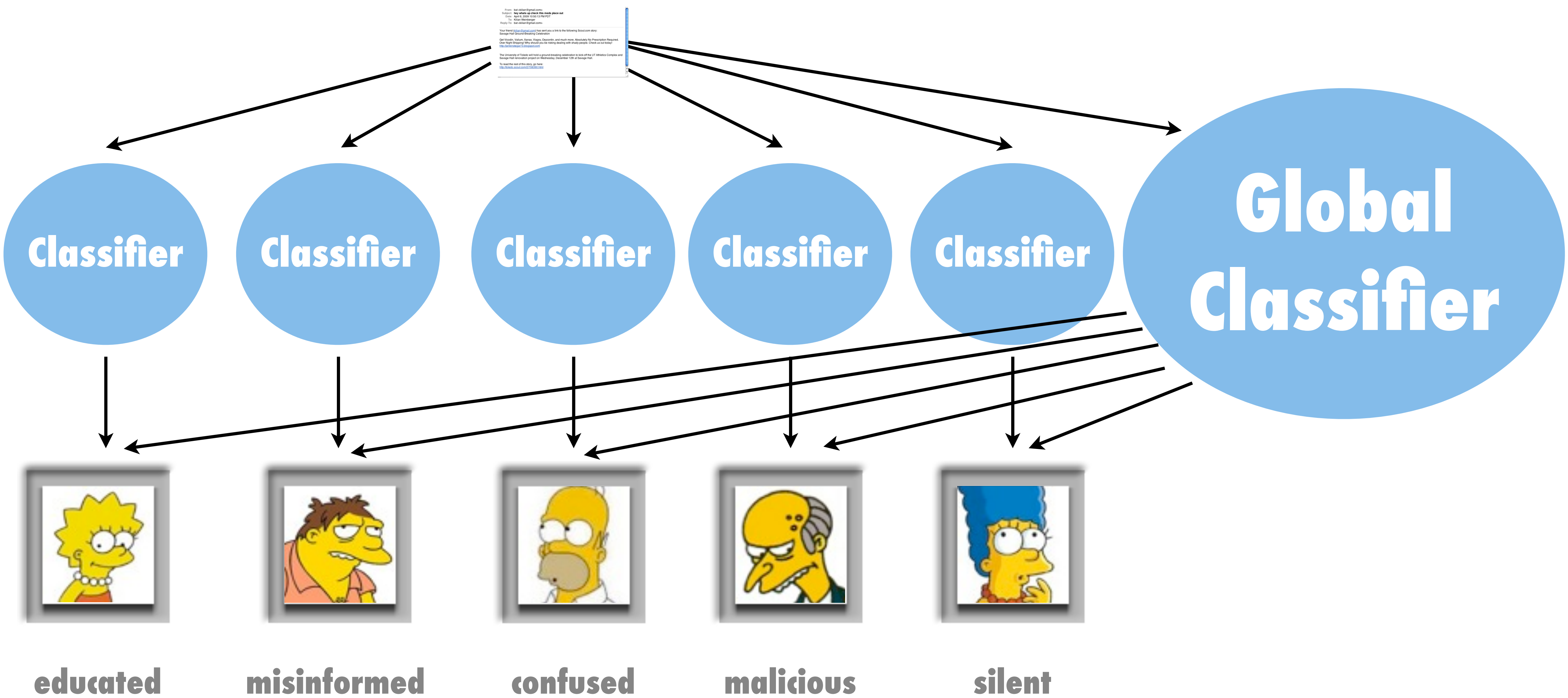
To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html



Spam Classification



Multitask Learning



Collaborative Classification

- **Primal representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

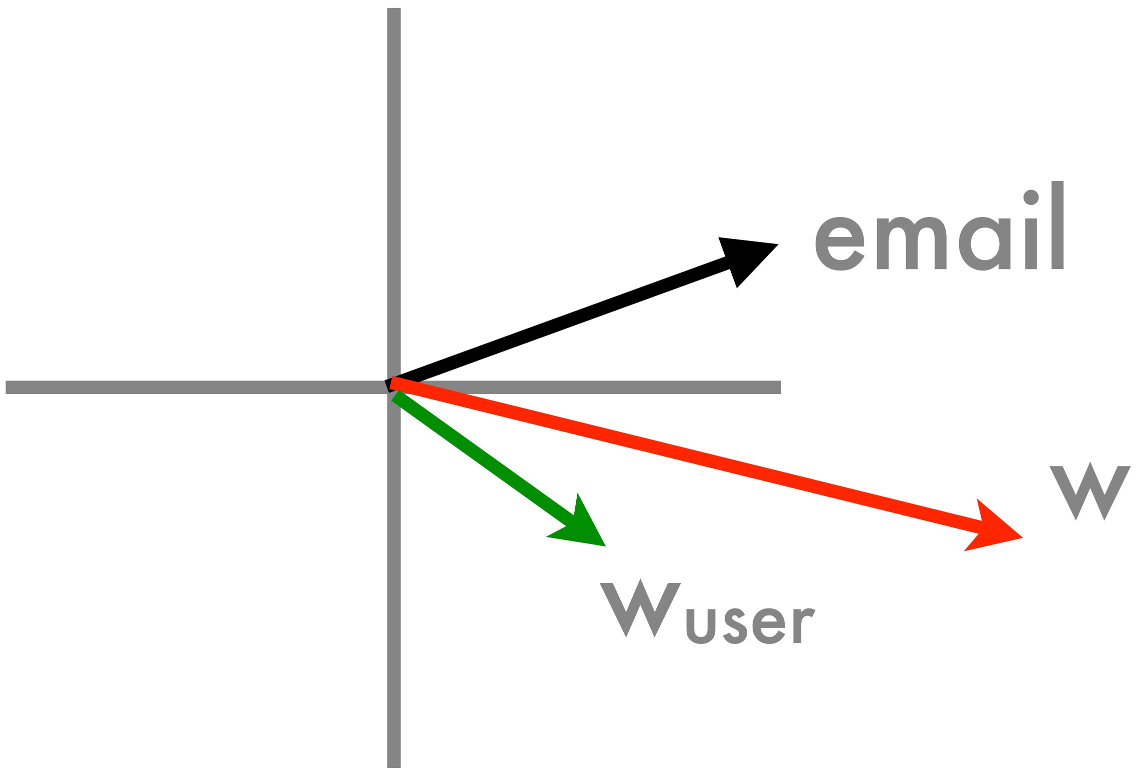
Kernel representation

$$k((x,u),(x',u')) = k(x,x')[1 + \delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

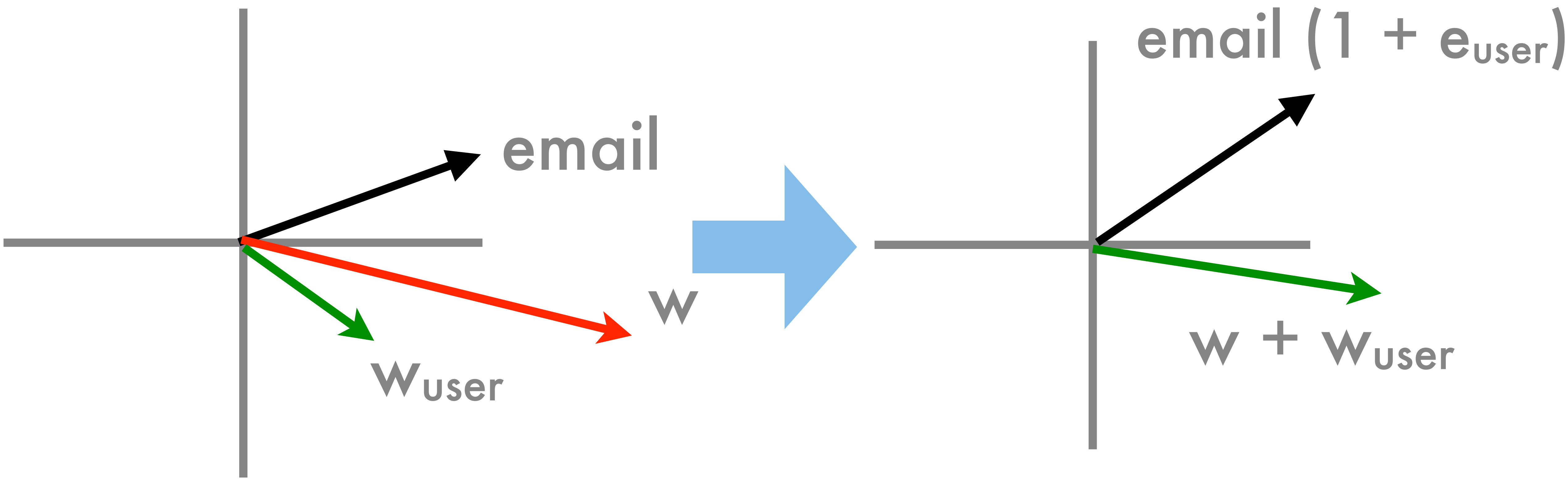
  Kernel representation

$$k((x,u),(x',u')) = k(x,x')[1 + \delta_{u,u'}]$$

  Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal representation**

$$f(x,u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

Kernel representation

$$k((x,u),(x',u')) = k(x,x')[1+\delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Hash Kernels

Hash Kernels

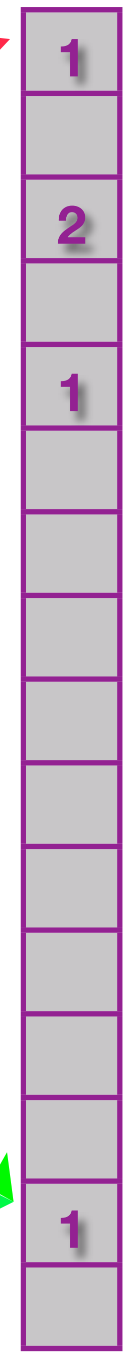
instance:

dictionary:

Hey,

please mention subtly during your talk that people should use Yahoo products more often.
Thanks,

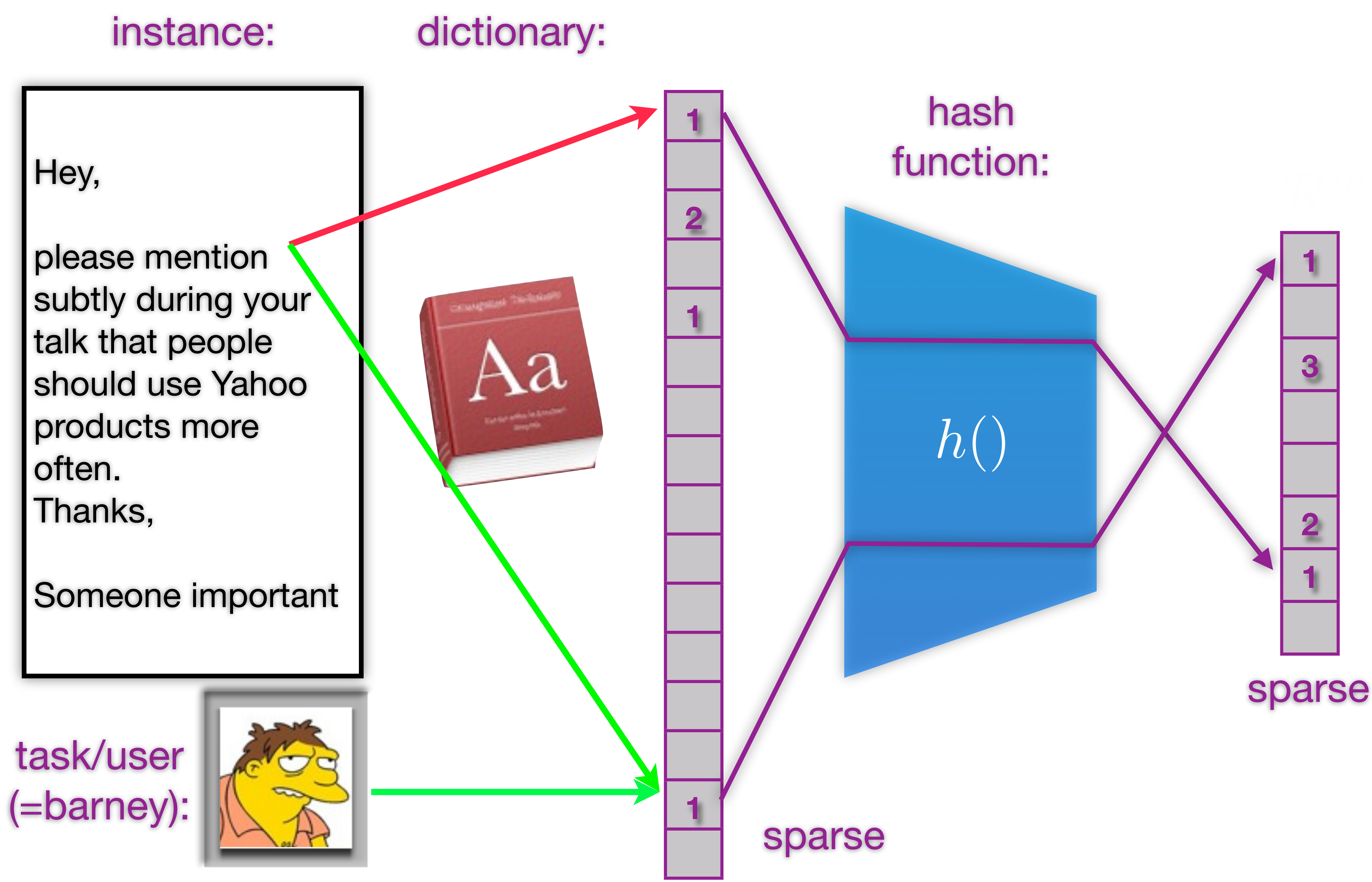
Someone important



sparse

task/user (=barney):

Hash Kernels

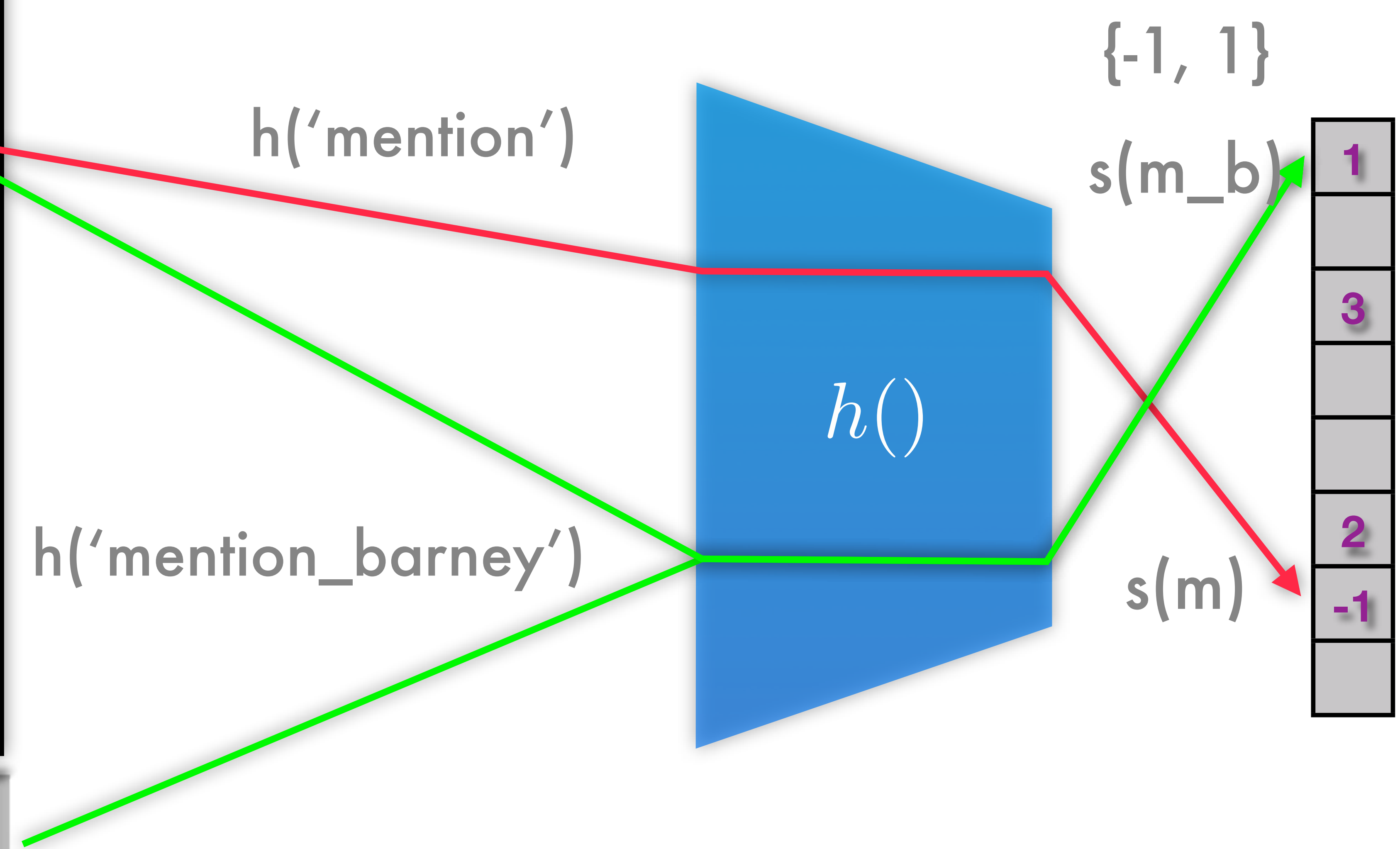


Hash Kernels

instance:

Hey,

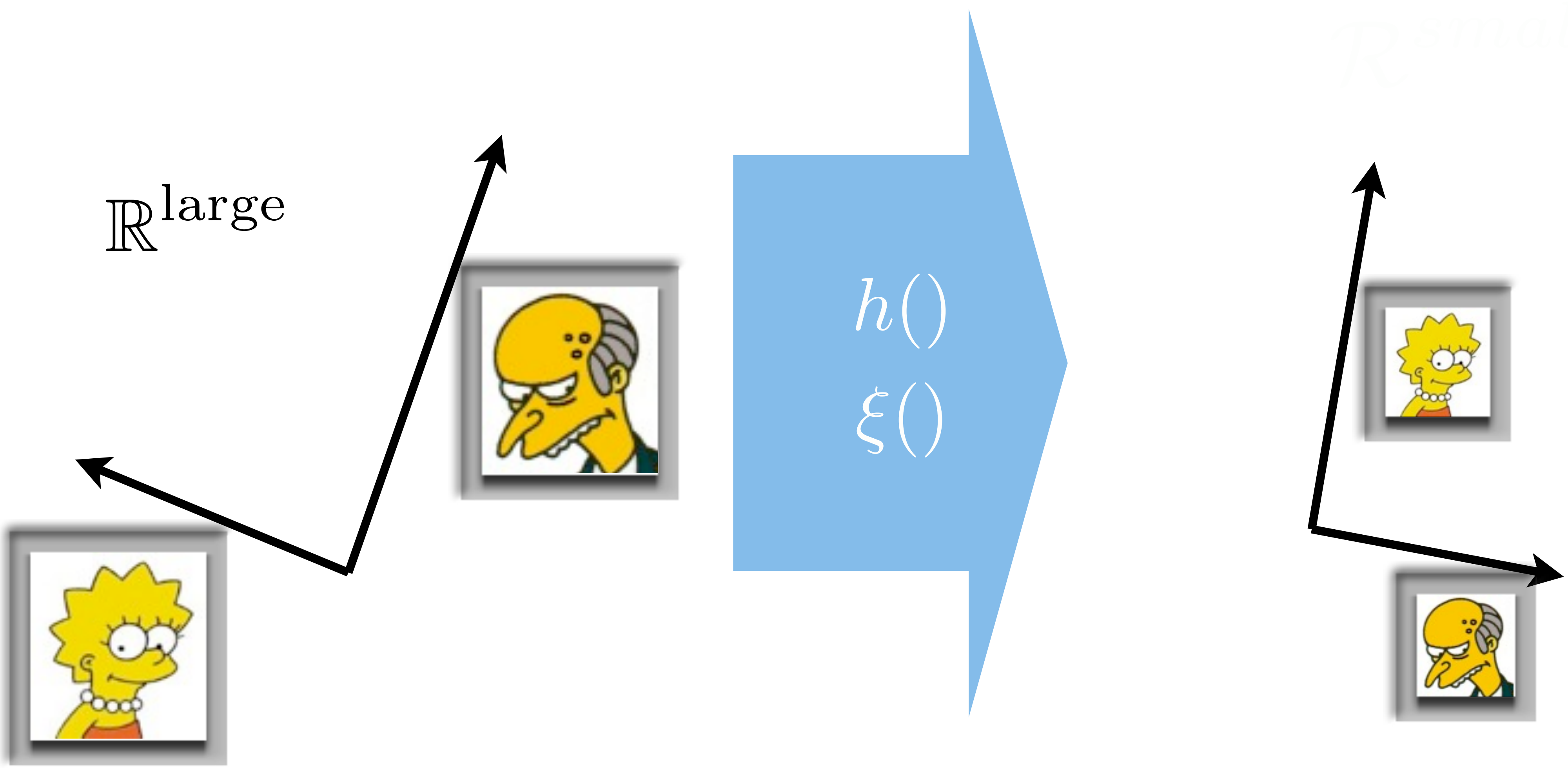
please mention subtly during your talk that people should use Yahoo search more often.
Thanks,



task/user (=barney):

Similar to count hash
(Charikar, Chen, Farrach-Colton, 2003)

Approximate Orthogonality



We can do multi-task learning!

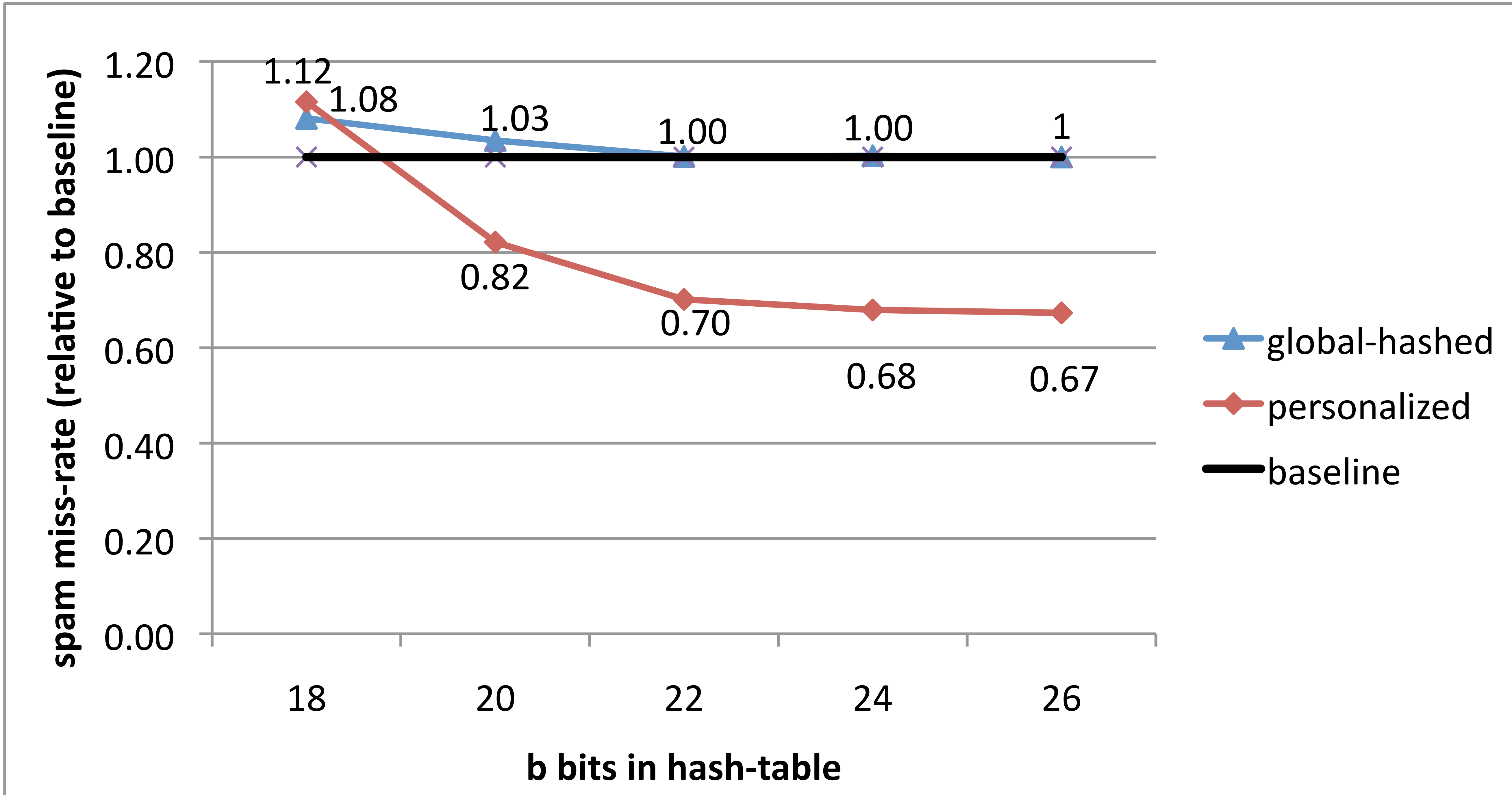
Guarantees

- For a random hash function the inner product vanishes with high probability via
$$\Pr\{|\langle w_v, h_u(x)\rangle| > \epsilon\} \leq 2e^{-C\epsilon^2 m}$$
- We can use this for multitask learning

Direct sum in Hilbert Space → **Sum** in Hash Space

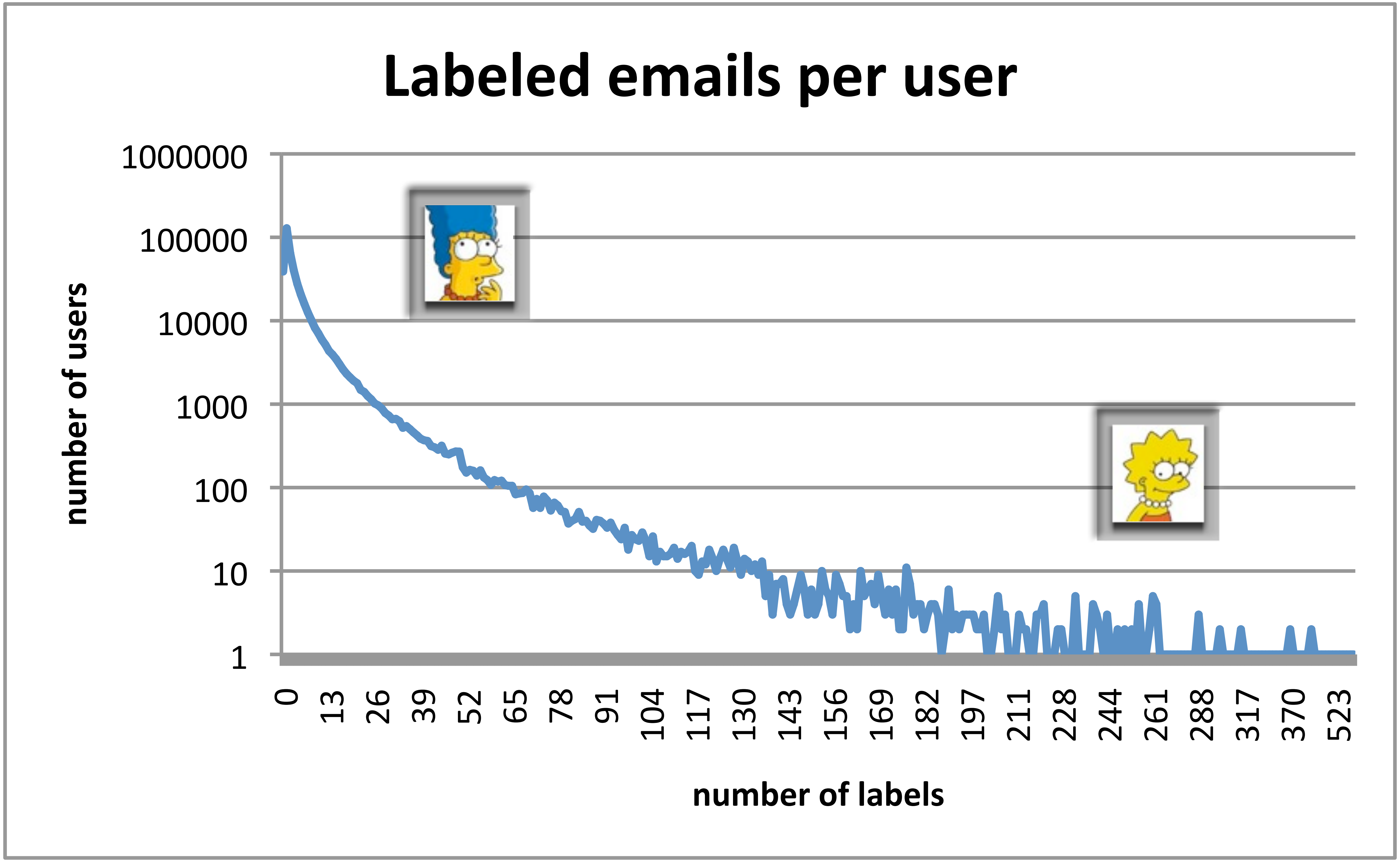
- **The hashed inner product is unbiased**
Proof: take expectation over random signs
- **The variance is O(1/n)**
Proof: brute force expansion
- Preserves sparsity
- No dictionary needed

Spam classification results



N=20M, U=400K

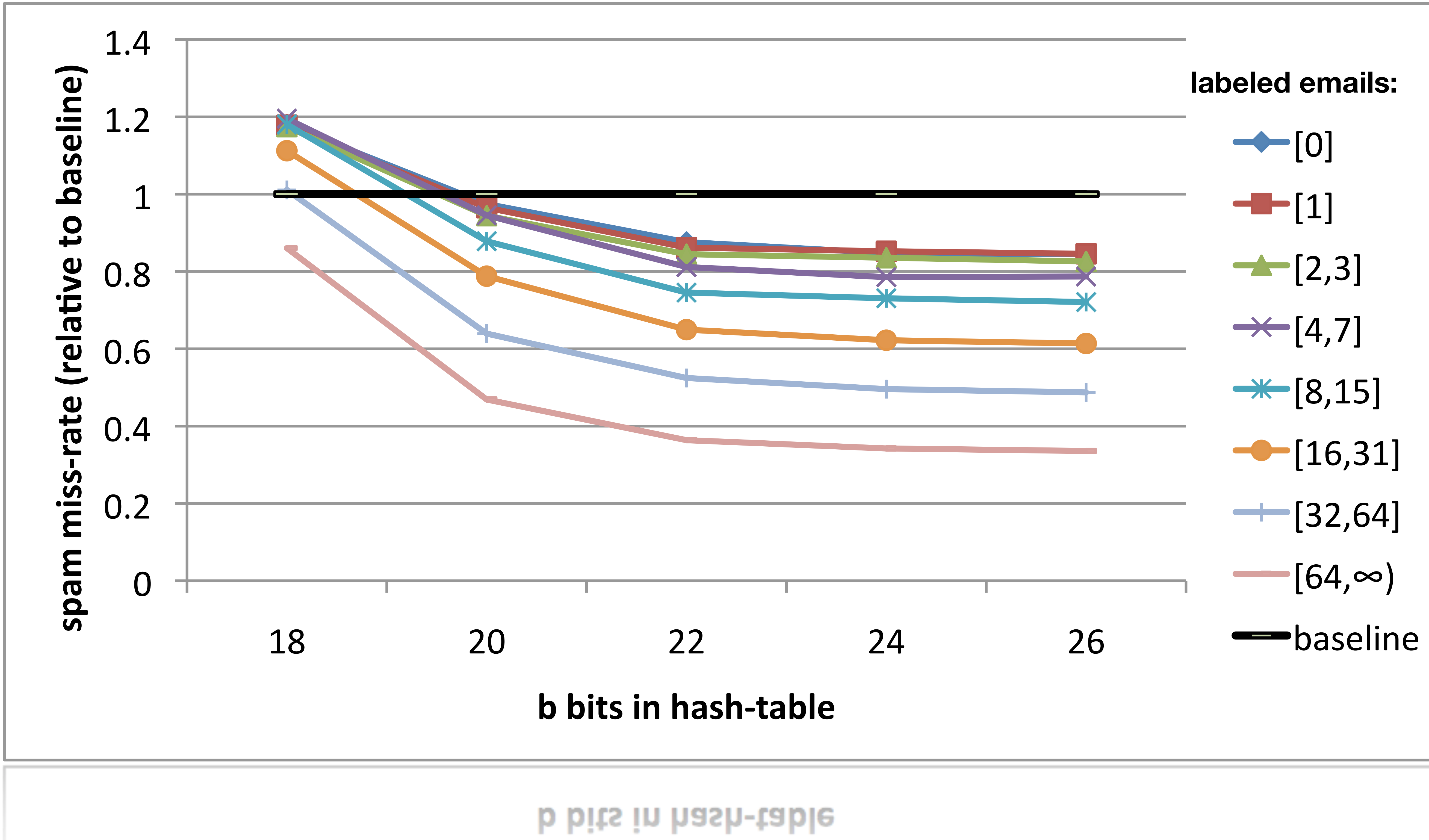
Lazy users ...



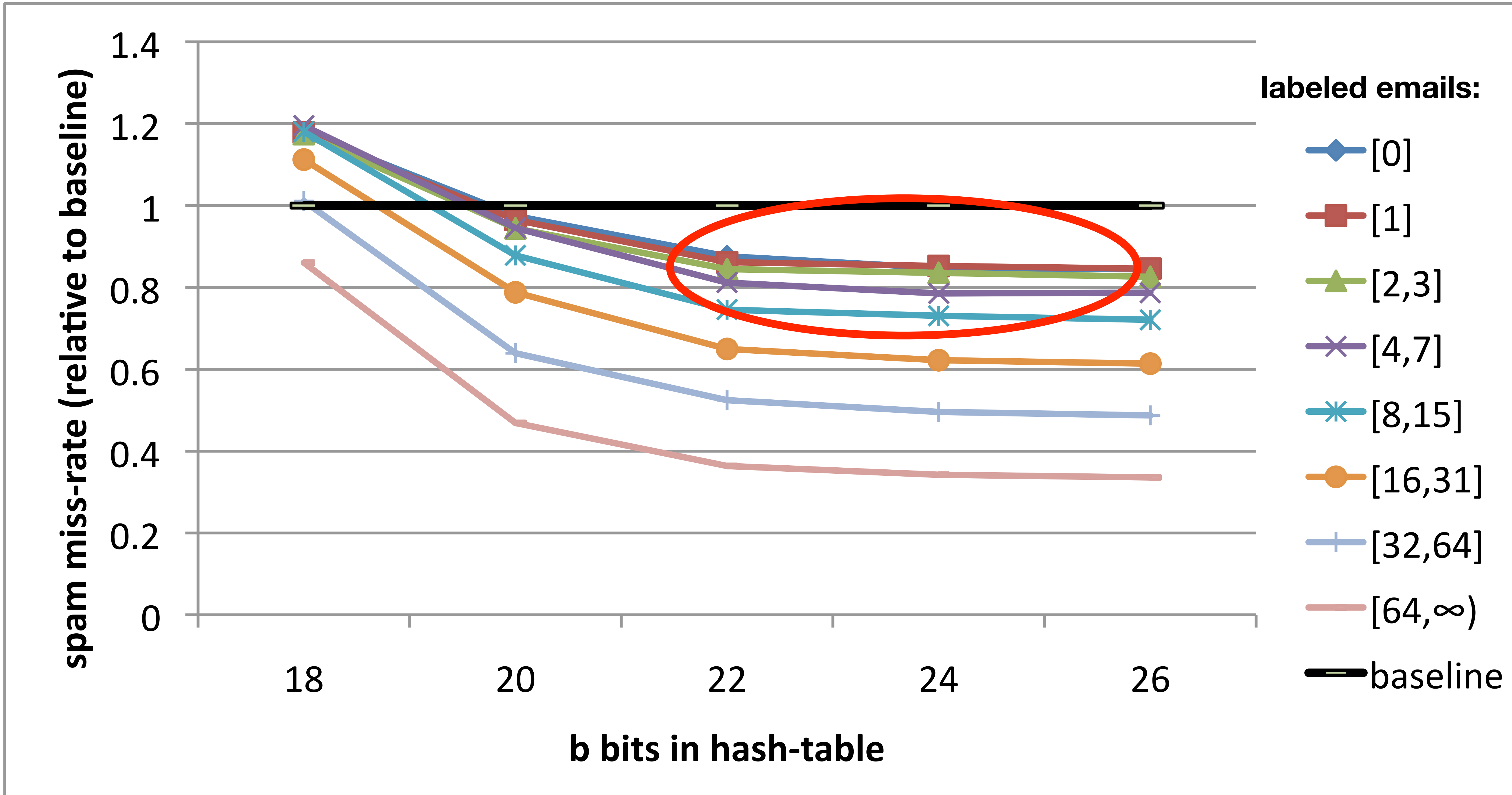
Labeled emails per user
number of users
1000000
100000
10000
1000
100
10
1
0
13
26
39
52
65
78
91
104
117
130
143
156
169
182
197
211
228
244
261
288
317
370
523
number of labels

Results by user group

Results by user group



Results by user group



Matrices

Collaborative Filtering

- **Netflix / Amazon / del.icio.us problem**
  - Many users, many products
  - Recommend product / news / friends
- **Matrix factorization**
  - Latent factor for users and movies each
  - Compatibility via
- **Factorization model**

$$X \approx U^\top V \text{ hence } X_{ij} \approx u_i^\top v_j$$

  - Optimization via stochastic gradient descent
  - Loss function depends on problem (regression, preference, ranking, quatile, novelty)

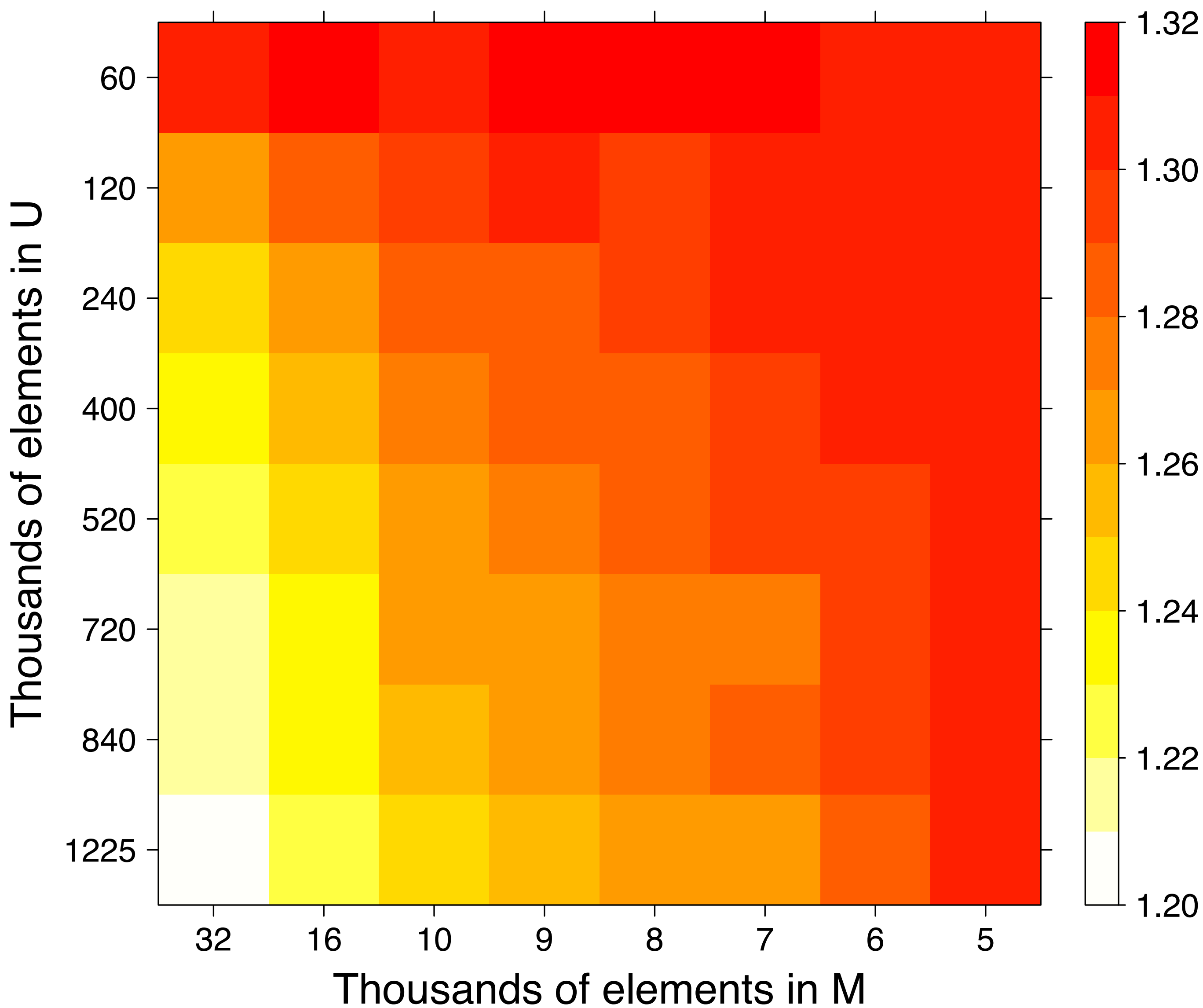
Collaborative Filtering

- **Big problem**
  - We have millions of users
  - We have millions of products
  - Storage - for 100 factors this is **800TB** of variables
  - We want a model that can be kept in **RAM** (<16GB)
- **Hashing compression**

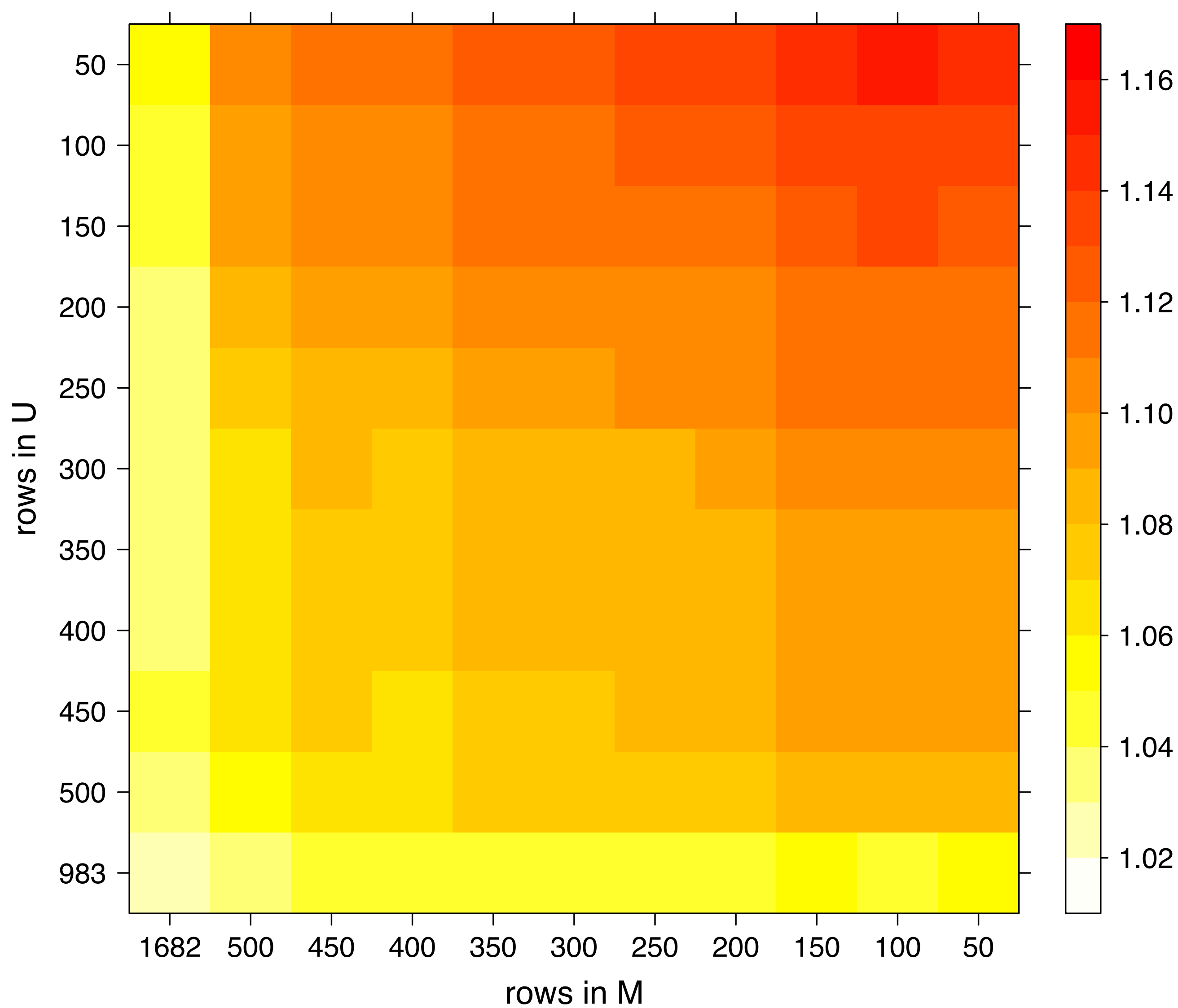
$$u_i = \sum_{j,k:h(j,k)=i} \xi(j,k) U_{jk} \text{ and } v_i = \sum_{j,k:h'(j,k)=i} \xi'(j,k) V_{jk}.$$

$$X_{ij} := \sum_k \xi(k,i)\xi'(k,j) u_{h(k,i)} v_{h'(k,j)}.$$

Examples



Eachmovie



MovieLens

Beyond

- String kernels
  - Hash substrings
  - Insert wildcards for approximate matching
- Data structures
  - Ontologies (hash class labels)
  - Hierarchical factorization (hash context)
- Feistel hash to reduce cache miss penalty

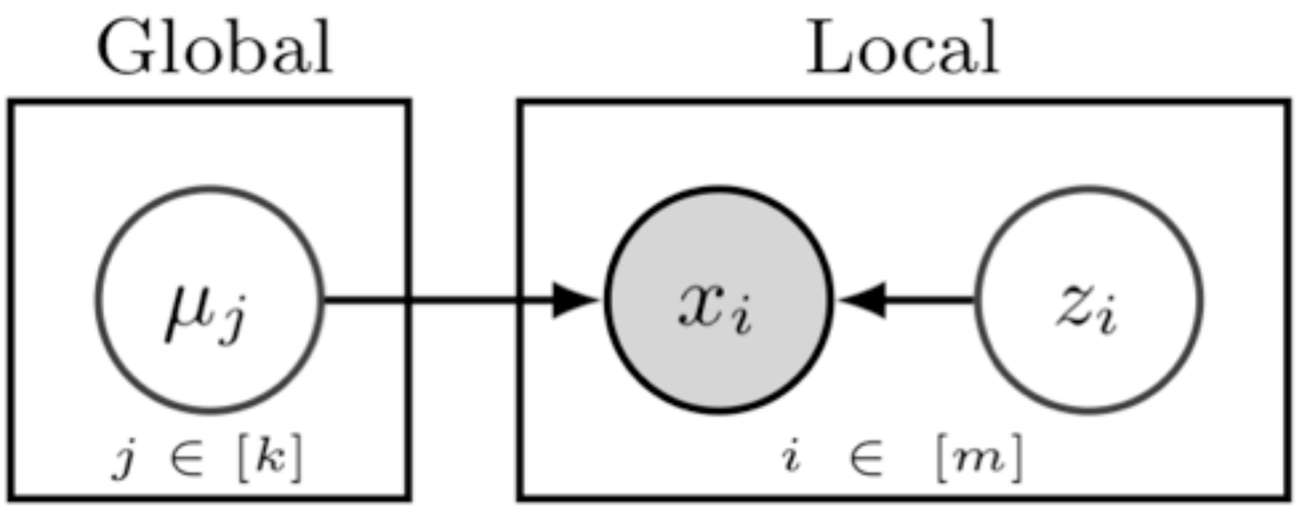
Beyond

- String kernels
  - Hash substrings
  - Insert wildcards for approximate matching
- Data structures
  - Ontologies (hash class labels)
  - Hierarchical factorization (hash context)
- Feistel hash to reduce cache miss penalty
- Better approximation guarantees in terms of risk
- Hashing does not satisfy RIP property
  (even breaks the Candes and Plan conditions)
- Dense function spaces
  (even Random Kitchen Sinks are too expensive)

How

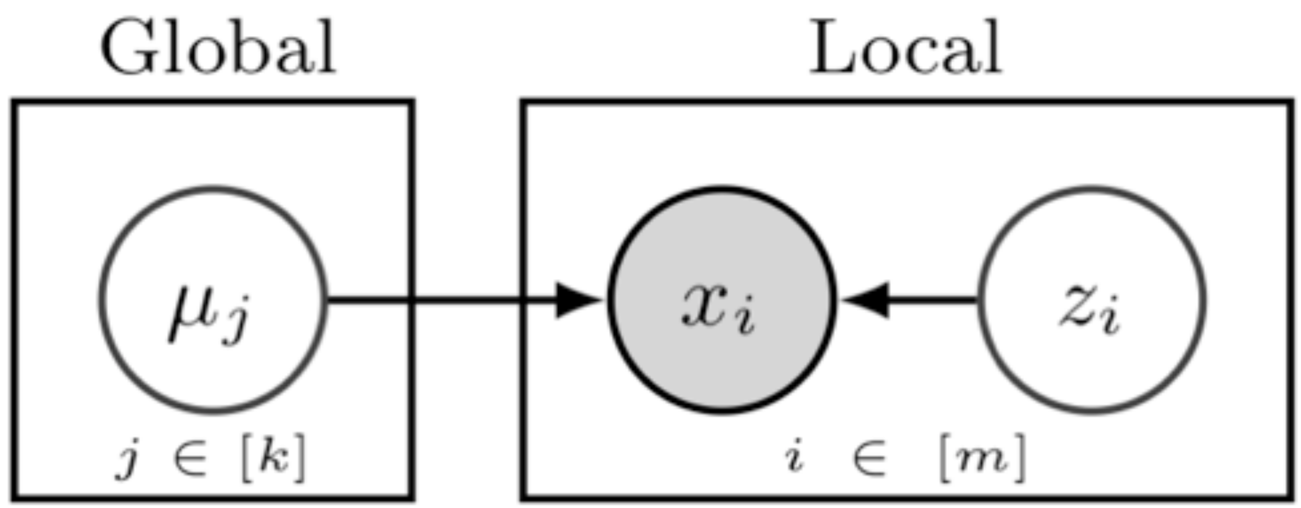
- Distributed Batch Convex Optimization
- Distributed Online Convex Optimization
- Parameter Compression
- **Distributed Sampling and Synchronization**

Latent Variable Models



- We don't observe everything
  - Poor engineering
  - Too intrusive
  - Too expensive
  - Machine failure
  - No editors
  - Forgot to measure it
  - Impossible to observe directly

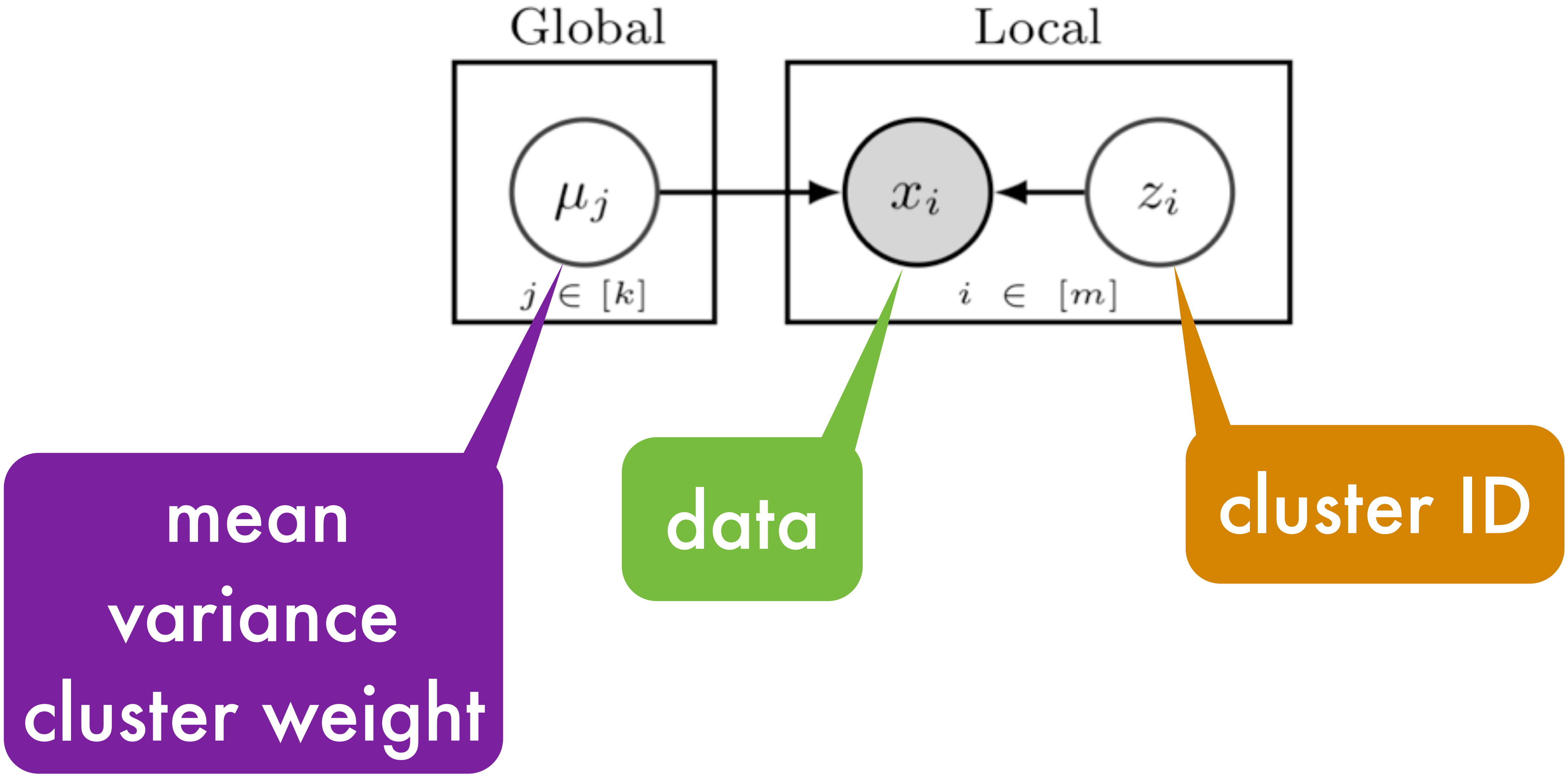
Latent Variable Models



- We don't observe everything
  - Poor engineering
  - Too intrusive
  - Too expensive
  - Machine failure
  - No editors
  - Forgot to measure it
  - Impossible to observe directly
- Local
  - Lots of evidence (data)
  - Lots of local state (parameters)
- Global
  - Large state (too large for single machine)
  - Depends on local state
  - Partitioning is difficult (e.g. natural graphs)

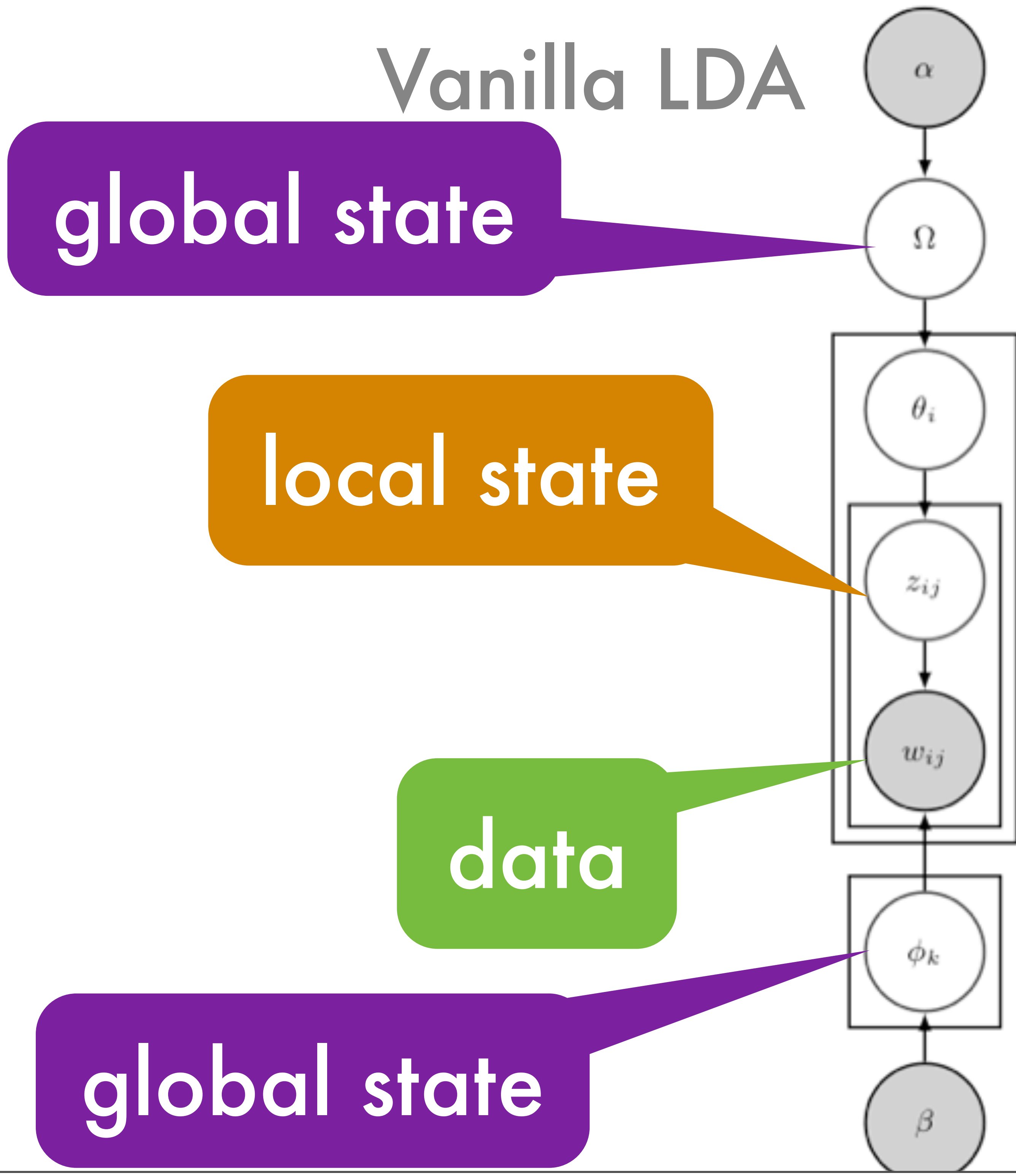
YAHOO!

Latent Variable Models

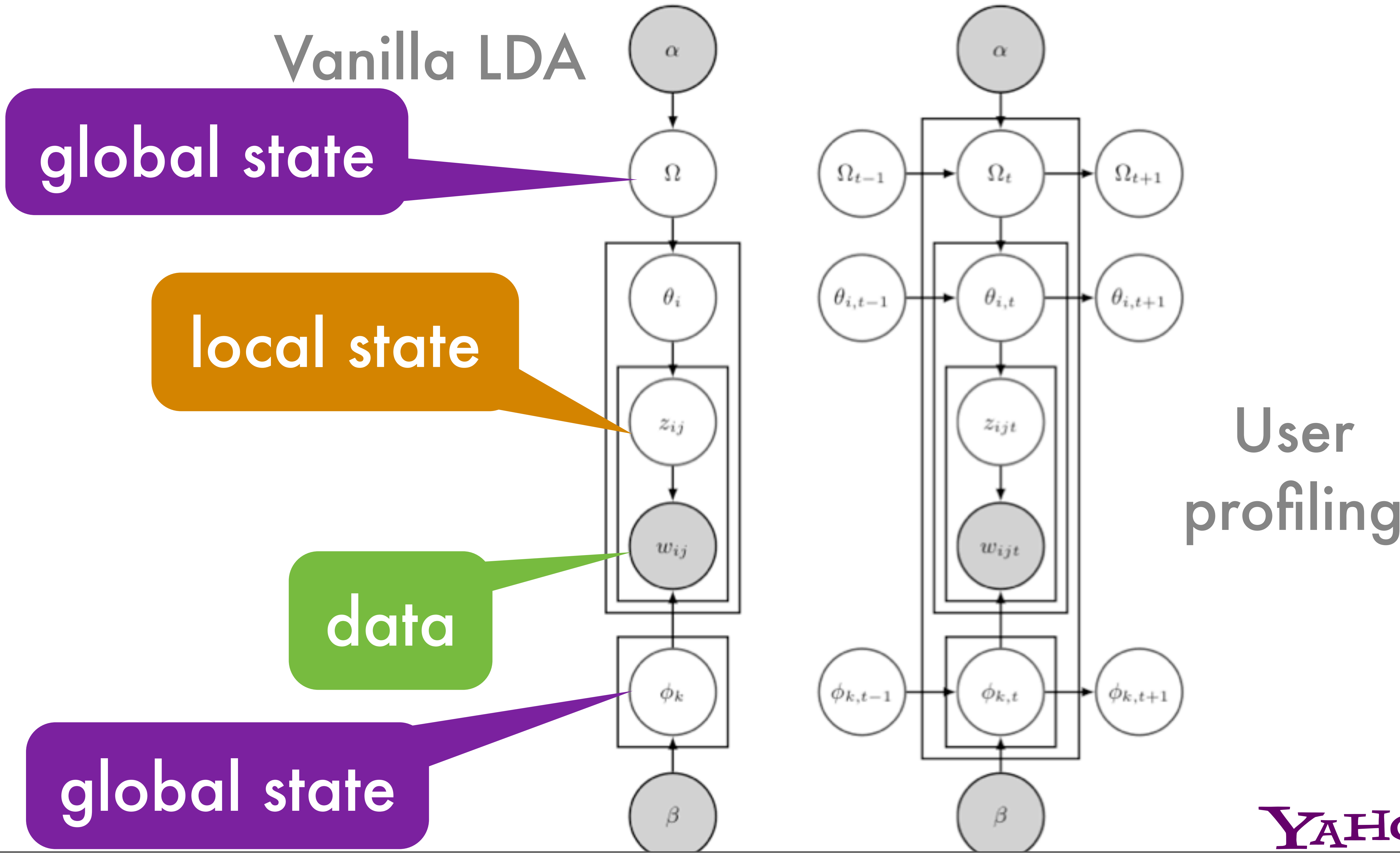


mixture of Gaussians clustering

Latent Variable Models

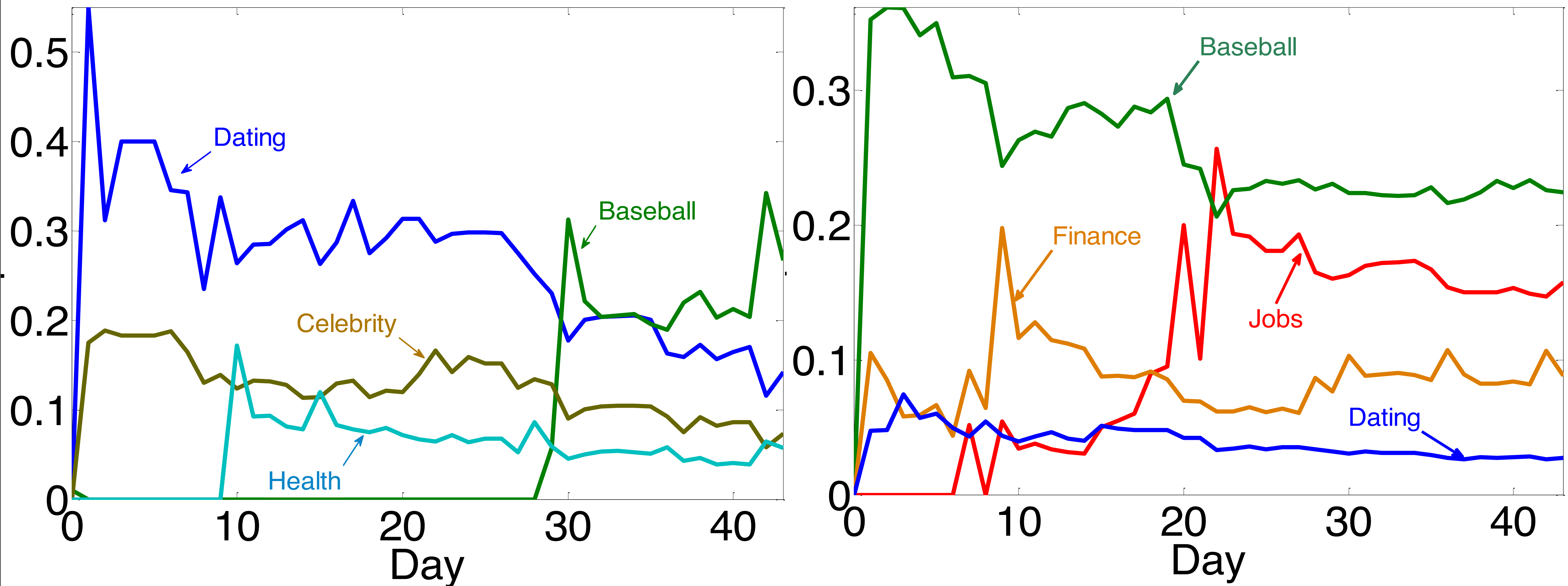


Latent Variable Models



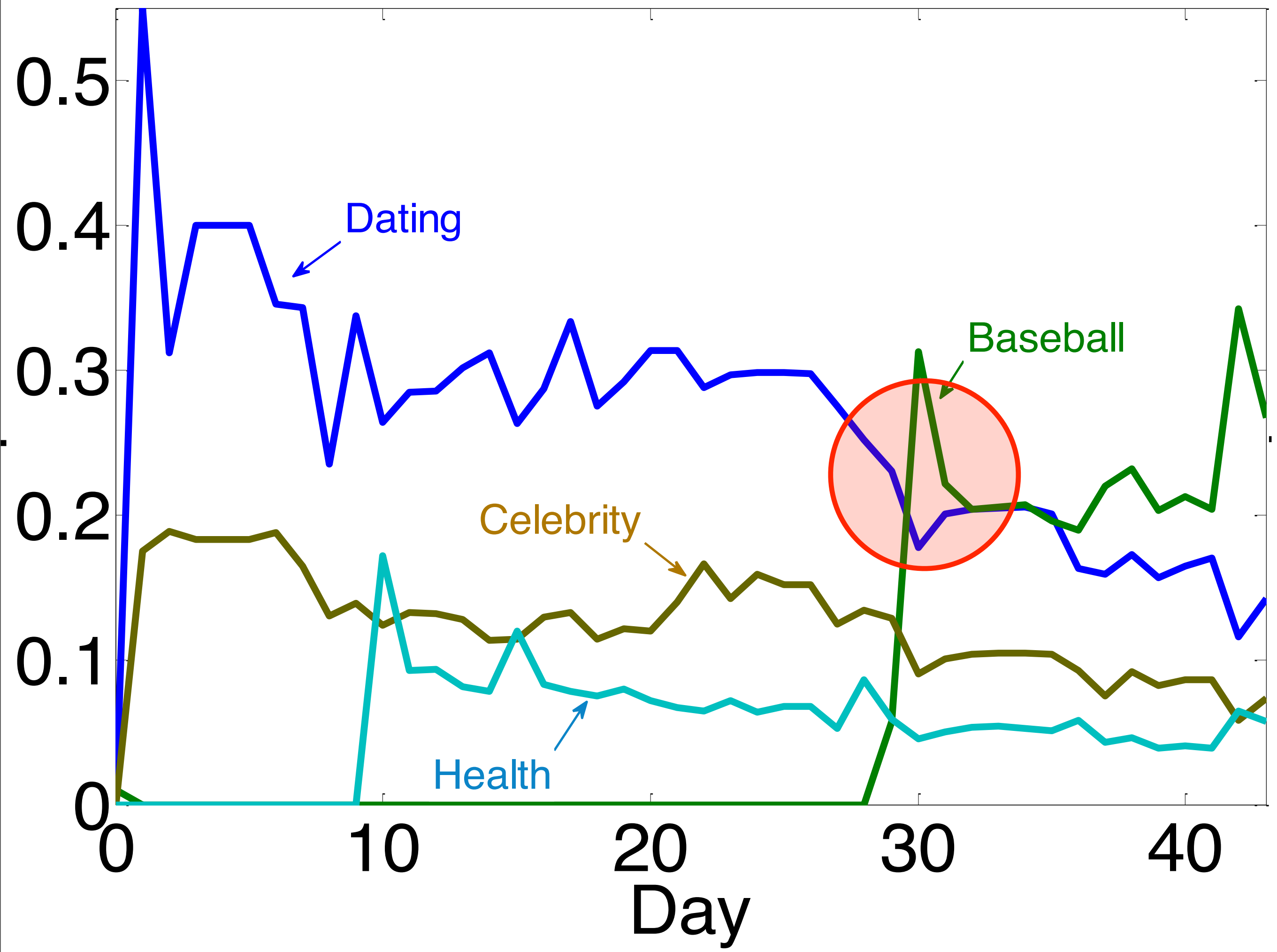
YAHOO!

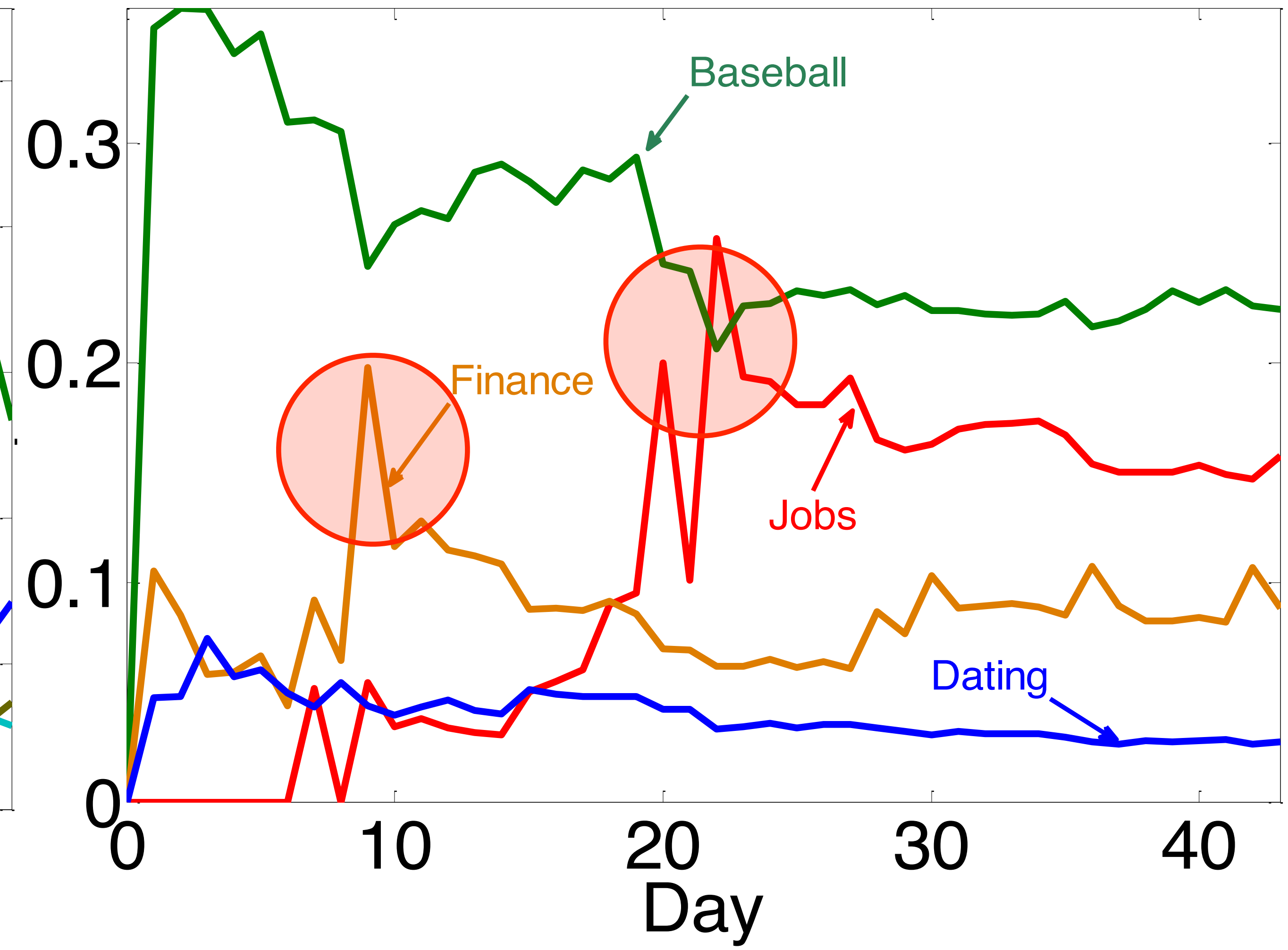
User profiling



Dating	Baseball	Celebrity	Health	Jobs	Finance
women men dating singles personals seeking match	League baseball basketball, doublehead Bergesen Griffey bullpen Greinke	Snooki Tom Cruise Katie Holmes Pinkett Kudrow Hollywood	skin body fingers cells toes wrinkle layers	job career business assistant hiring part-time receptionist	financial Thomson chart real Stock Trading currency

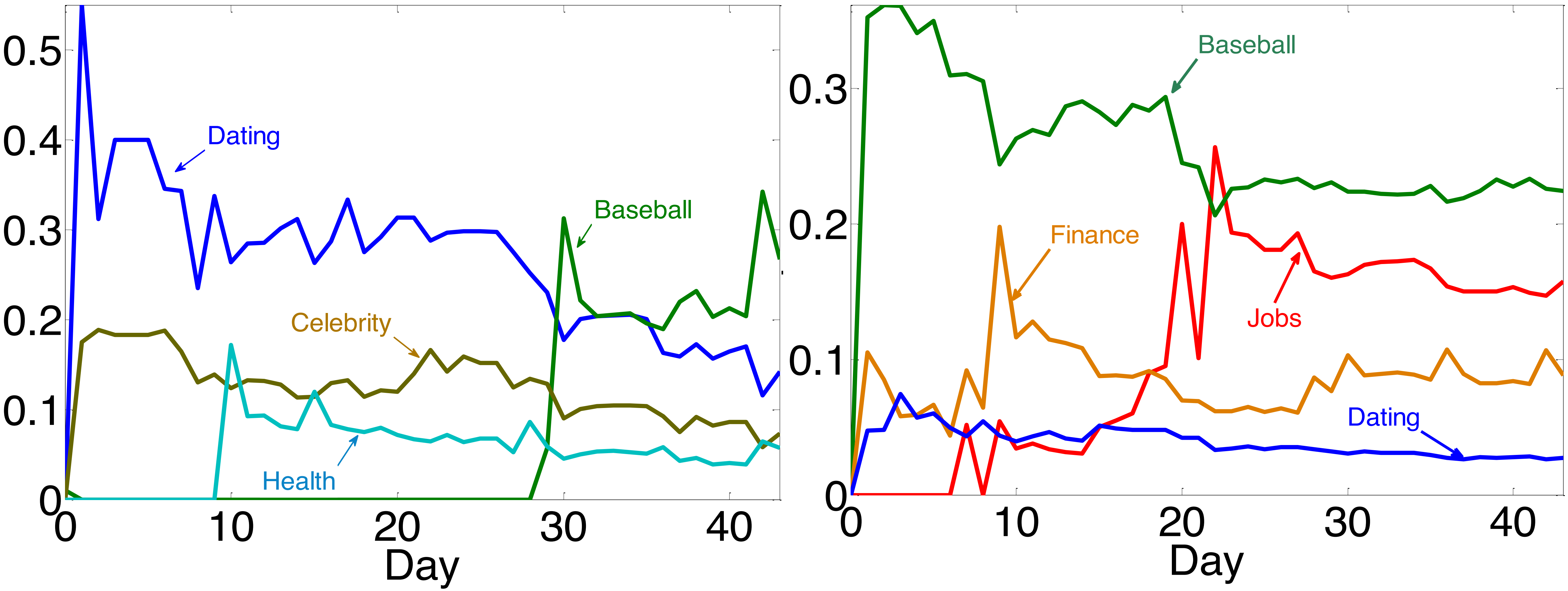
User profiling





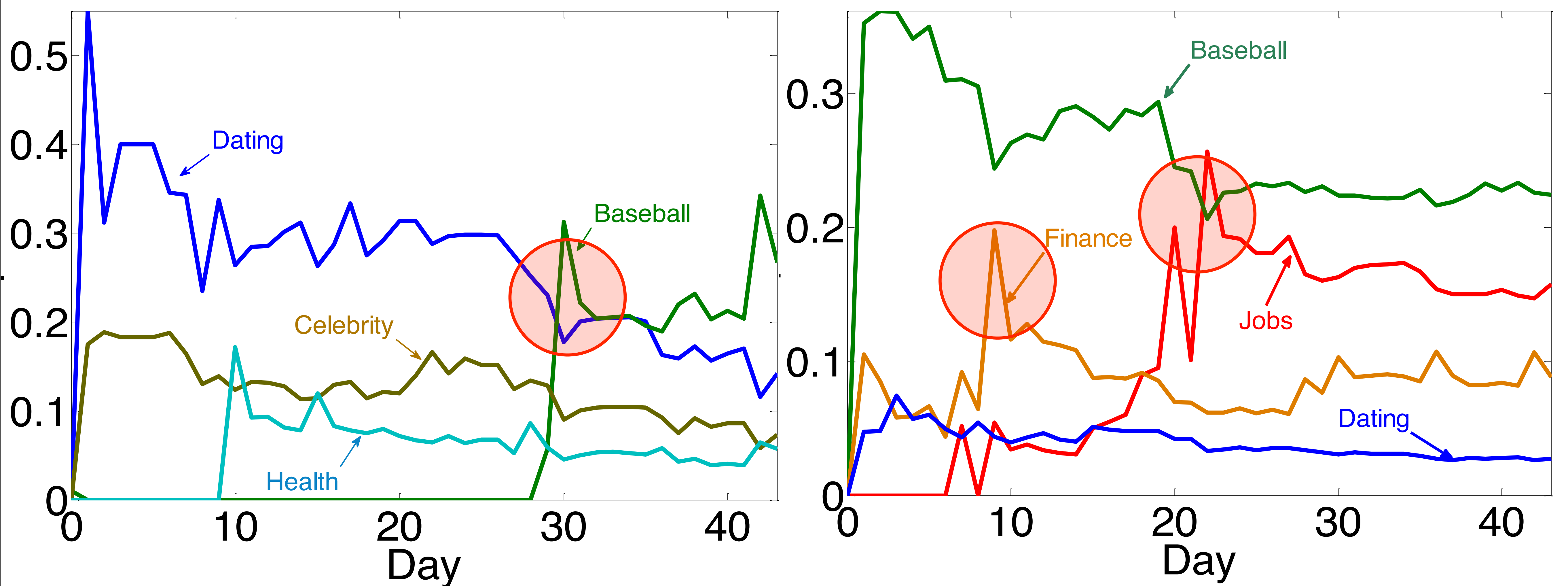
Dating	Baseball	Celebrity	Health	Jobs	Finance
women men dating singles personals seeking match	League baseball basketball, doublehead Bergesen Griffey bullpen Greinke	Snooki Tom Cruise Katie Holmes Pinkett Kudrow Hollywood	skin body fingers cells toes wrinkle layers	job career business assistant hiring part-time receptionist	financial Thomson chart real Stock Trading currency

User profiling



500 Million Users
100+ topics
full activity logs
1000 machines

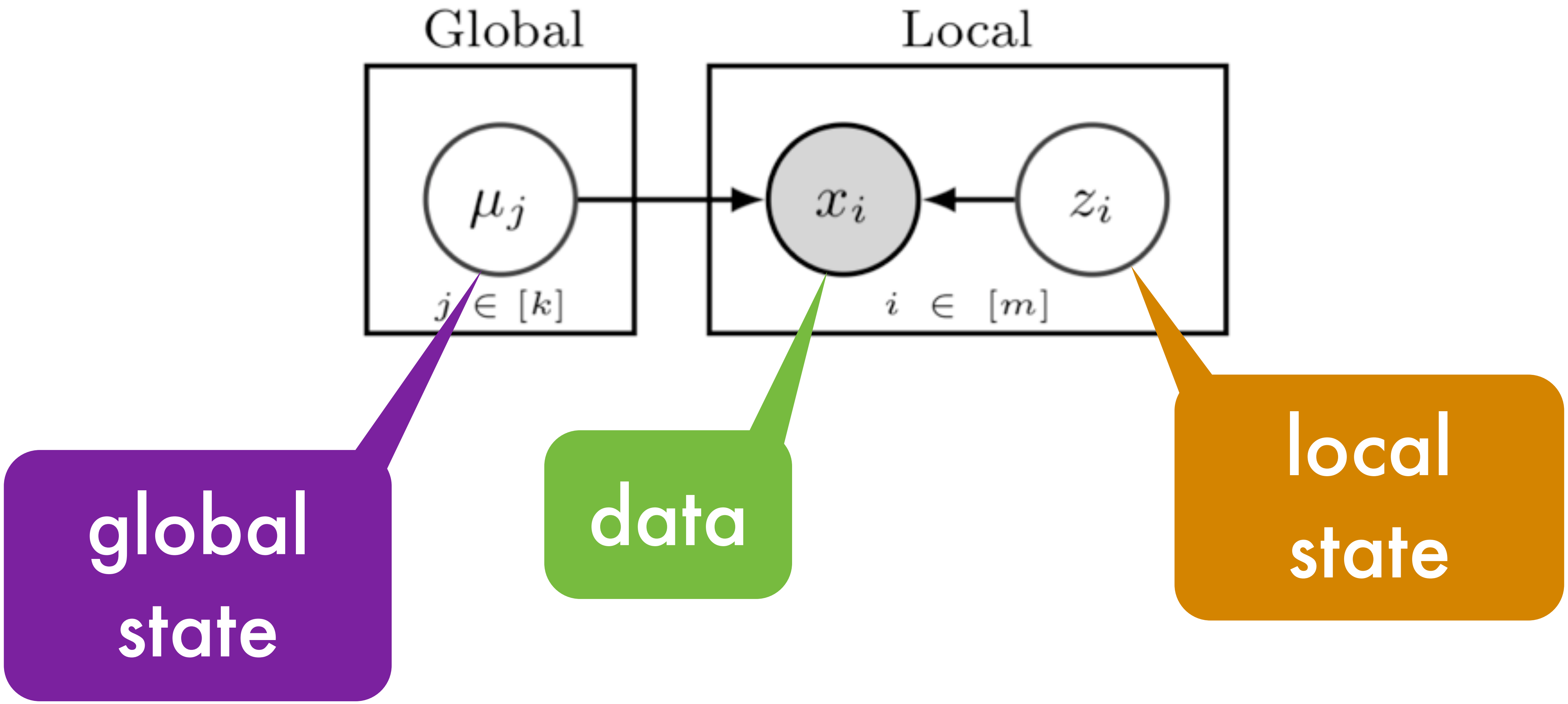
User profiling



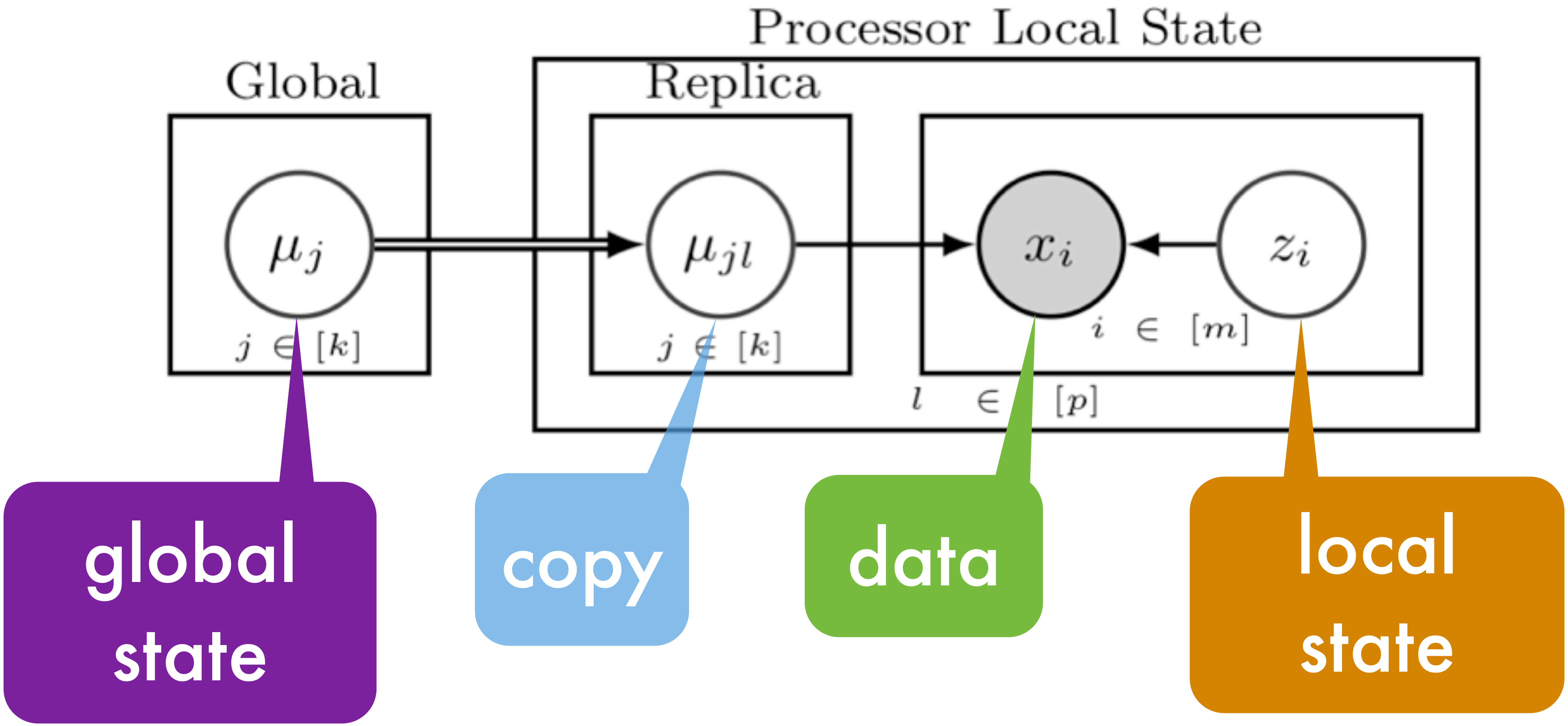
500 Million Users
100+ topics
full activity logs
1000 machines

Synchronization

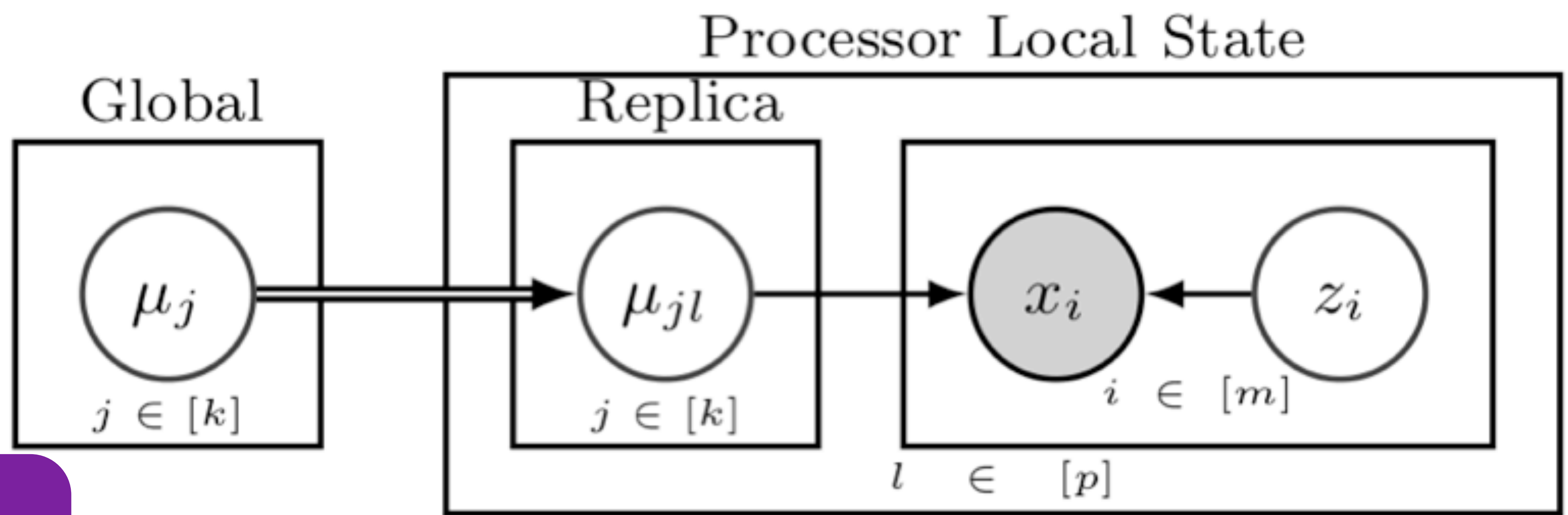
Variable Caching

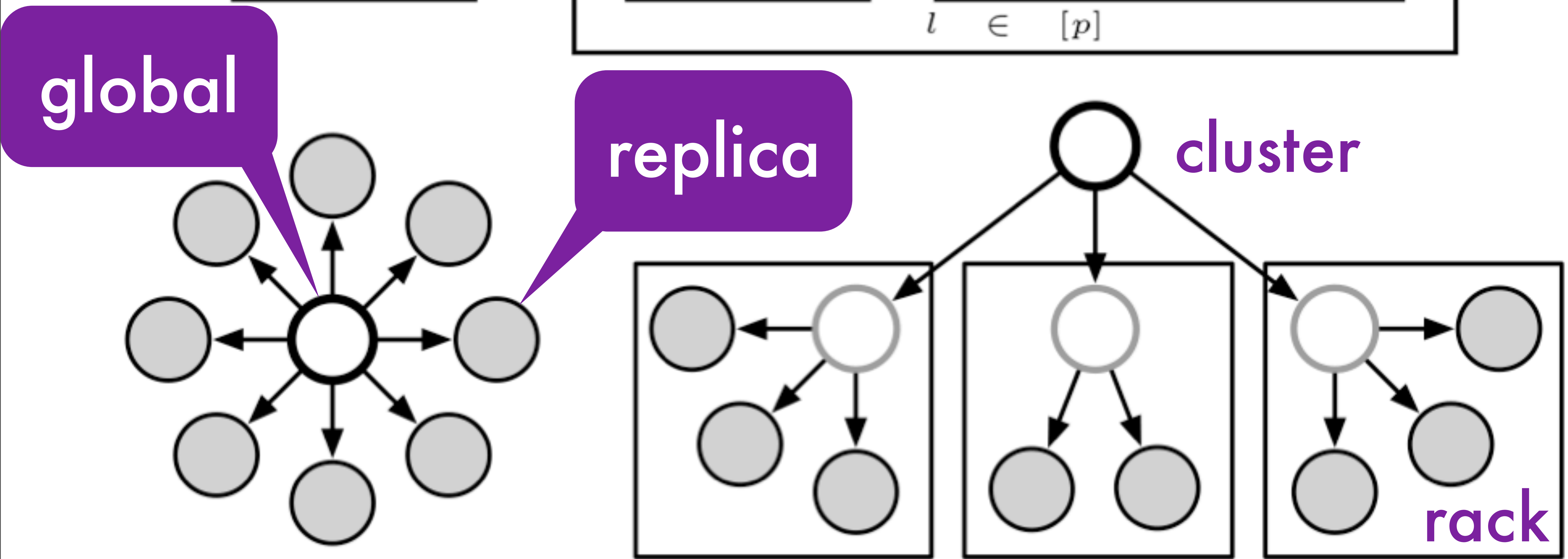


Variable Caching



Variable Caching





Message Passing

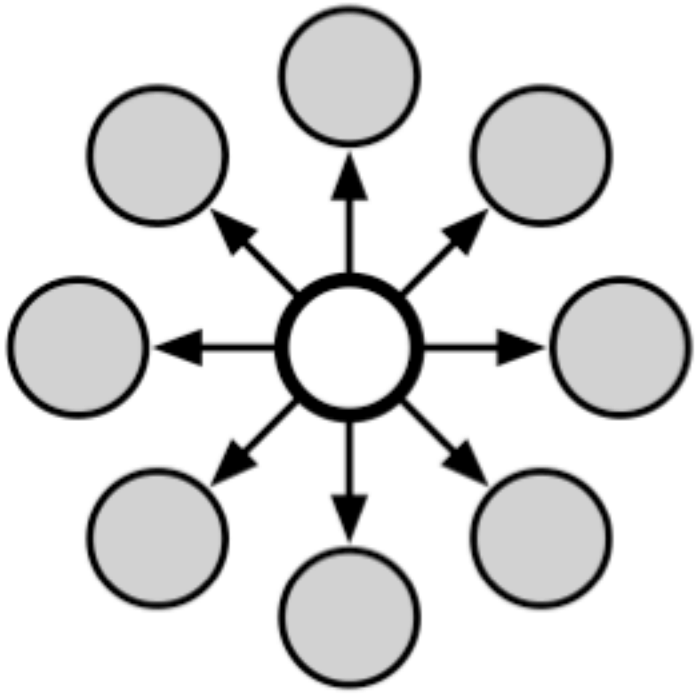
- Child performs updates (sampling, variational)
- Synchronization
  - Start with common state
  - Child stores old and new state
  - Parent keeps global state
  - Bandwidth limited
- Works for any abelian group (sum, log-sum, cyclic group)

local to global

$$\delta \leftarrow x - x^{\text{old}}$$

$$x^{\text{old}} \leftarrow x$$

$$x^{\text{global}} \leftarrow x^{\text{global}} + \delta$$



global to local

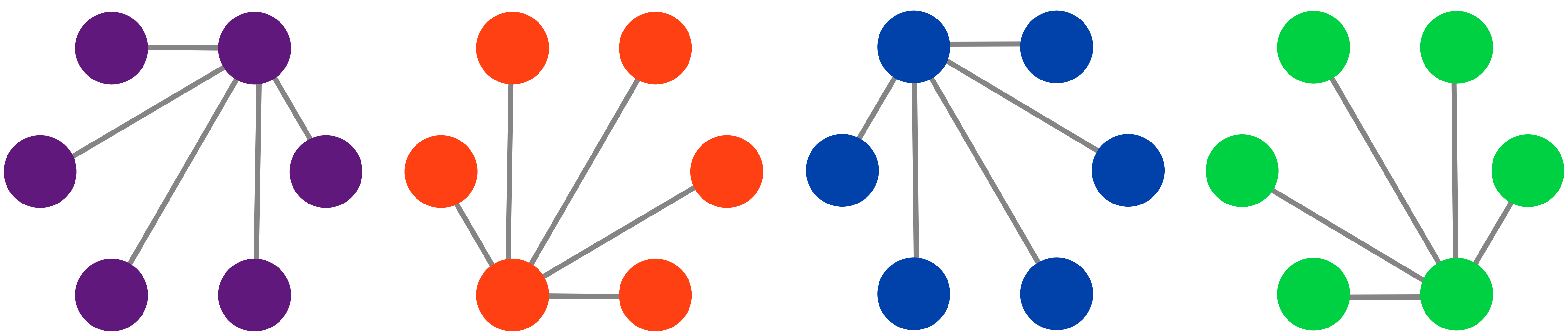
$$x \leftarrow x + \left(x^{\text{global}} - x^{\text{old}}\right)$$

$$x^{\text{old}} \leftarrow x^{\text{global}}$$

Consistent Hashing

- Dedicated server for variables
  - Insufficient bandwidth (hotspots)
  - Insufficient memory
- Select server via consistent hashing

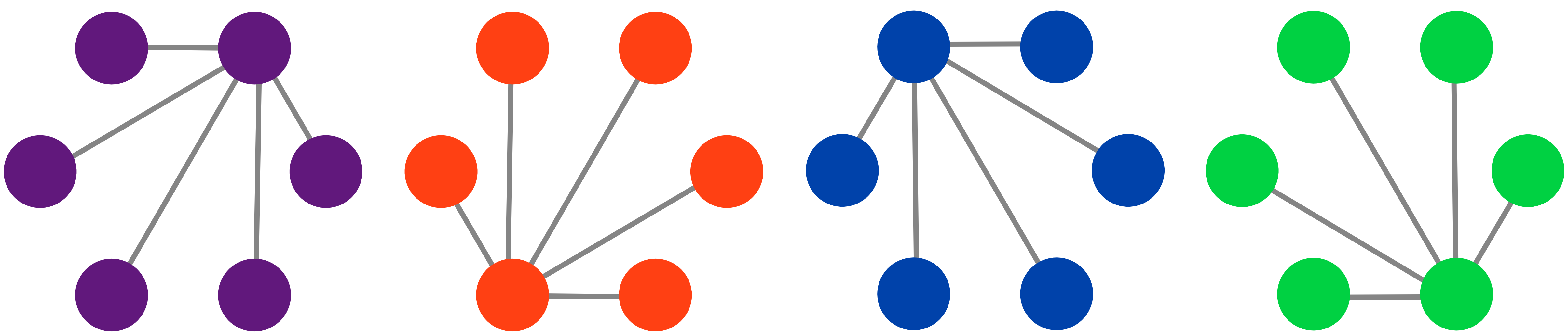
$$m(x) = \underset{m \in M}{\operatorname{argmin}}\, h(x, m)$$



Consistent Hashing

- Storage is O(1/k) per machine
- Communication is O(1) per machine
- Fast snapshots O(1/k) per machine
- O(k) open connections per machine
- O(1/k) throughput per machine

$$m(x) = \operatorname*{argmin}_{m \in M} h(x, m)$$

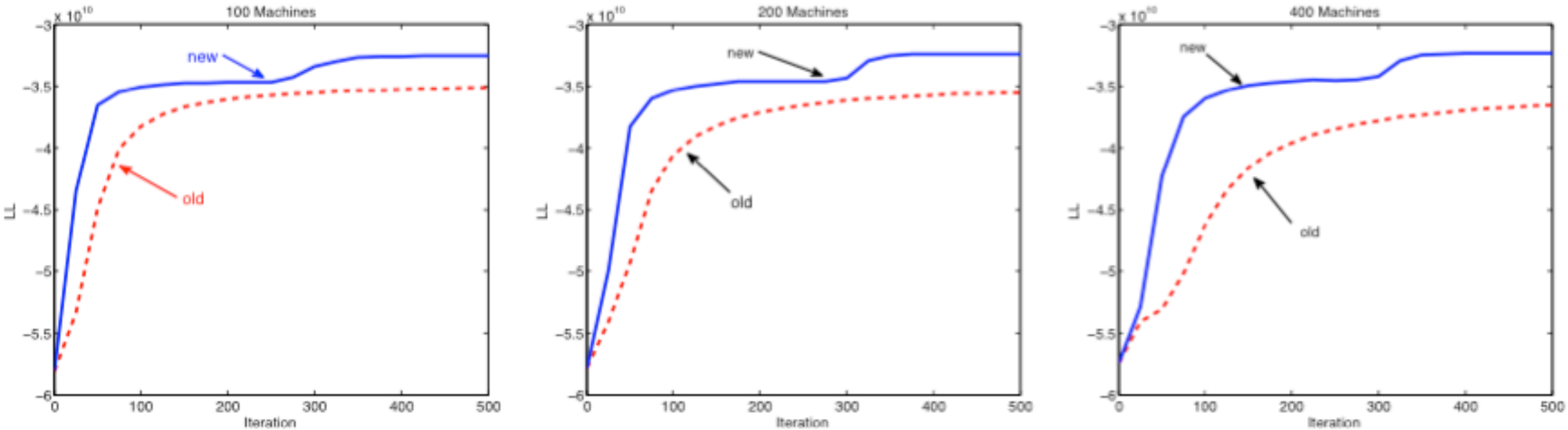


Communication Shaping

- Data rate between machines is O(1/k)
- Machines operate asynchronously (no barrier)
- Solution
  - Schedule message pair
  - Communicate with r machines simultaneously
  - Use Luby-Rackoff PRNG for load balancing
- Efficiency guarantee

$$1 - e^{-r} \sum_{i=0}^{r} \left[1 - \frac{i}{r}\right] \frac{r^i}{i!} \leq \text{Eff} \leq 1 - e^{-r}$$

Performance

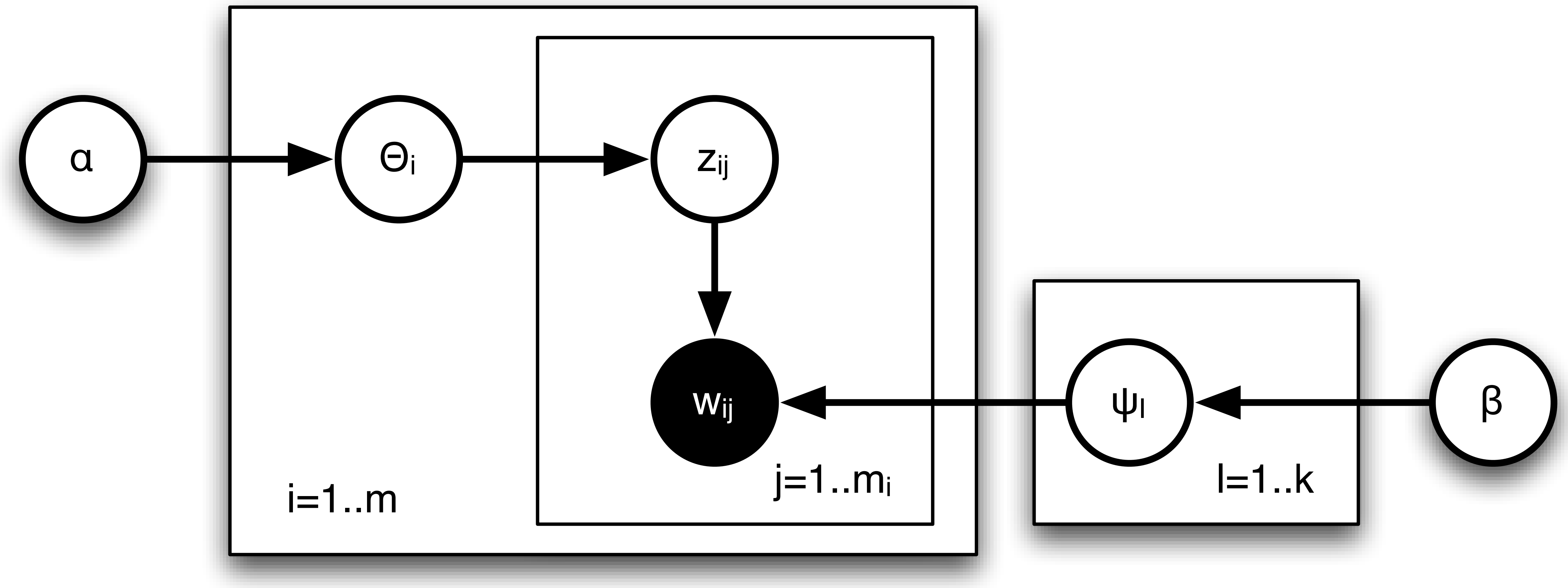


- 8 Million documents, 1000 topics, {100,200,400} machines, LDA
- Red (symmetric latency bound message passing)
- Blue (asynchronous bandwidth bound message passing & message scheduling)
  - 10x faster synchronization time
  - 10x faster snapshots
  - Scheduling improves 10% already on 150 machines

LDA - our Guinea Pig

https://**github**.com/shravanmn/**Yahoo_LDA**

Latent Dirichlet Allocation



Sequential Algorithm

- Collapsed Gibbs Sampler (Griffith & Steyvers 2005)
  - For 1000 iterations do
    - For each document do
      - For each word in the document do
        - Resample topic for the word
        - Update local (document, topic) table
        - Update global (word, topic) table

Sequential Algorithm

- Collapsed Gibbs Sampler (Griffith & Steyvers 2005)
  - For 1000 iterations do
    - For each document do
      - For each word in the document do
        - Resample topic for the word
        - Update local (document, topic) table
        - Update global (word, topic) table

this kills parallelism

YAHOO!

State of the art
UMass Mallet, UC Irvine, Google

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Update CPU local (word, topic) table
  - Update global (word, topic) table

$$p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \bar{\beta}} + \beta_w \frac{n(t, d = i)}{n(t) + \bar{\beta}} + \frac{n(t, w = w_{ij}) \left[n(t, d = i) + \alpha_t \right]}{n(t) + \bar{\beta}}$$

YAHOO!

State of the art
UMass Mallet, UC Irvine, Google

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Update CPU local (word, topic) table
  - Update global (word, topic) table

$$p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \bar{\beta}} + \beta_w \frac{n(t, d=i)}{n(t) + \bar{\beta}} + \frac{n(t, w=w_{ij}) \left[n(t, d=i) + \alpha_t\right]}{n(t) + \bar{\beta}}$$

slow

State of the art
UMass Mallet, UC Irvine, Google

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Update CPU local (word, topic) table
  - Update global (word, topic) table

changes rapidly

$$p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \bar{\beta}} + \beta_w \frac{n(t, d = i)}{n(t) + \bar{\beta}} + \frac{n(t, w = w_{ij}) \left[n(t, d = i) + \alpha_t \right]}{n(t) + \bar{\beta}}$$

slow

YAHOO!

State of the art
UMass Mallet, UC Irvine, Google

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Update CPU local (word, topic) table
  - Update global (word, topic) table

changes rapidly

$$p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \bar{\beta}} + \beta_w \frac{n(t, d=i)}{n(t) + \bar{\beta}} + \frac{n(t, w=w_{ij})\left[n(t, d=i) + \alpha_t\right]}{n(t) + \bar{\beta}}$$

slow

moderately fast

YAHOO!

State of the art
UMass Mallet, UC Irvine, Google

- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Update CPU local (word, topic) table
  - Update global (word, topic) table

table out of sync

memory inefficient

blocking

network bound

changes rapidly

$$p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \bar{\beta}} + \beta_w \frac{n(t, d = i)}{n(t) + \bar{\beta}} + \frac{n(t, w = w_{ij}) \left[n(t, d = i) + \alpha_t\right]}{n(t) + \bar{\beta}}$$

slow

moderately fast

YAHOO!

Distributed asynchronous sampler

- For 1000 iterations do (independently per computer)
  - For each thread/core do
    - For each document do
      - For each word in the document do
        - Resample topic for the word
        - Update local (document, topic) table
        - Generate computer local (word, topic) message
    - In parallel update local (word, topic) table
  - In parallel update global (word, topic) table

Distributed asynchronous sampler

- For 1000 iterations do (independently per computer)
  - For each thread/core do
    - For each document do
      - For each word in the document do
        - Resample topic for the word
        - Update local (document, topic) table
        - Generate computer local (word, topic) message
    - In parallel update local (word, topic) table
  - In parallel update global (word, topic) table

concurrent cpu hdd net

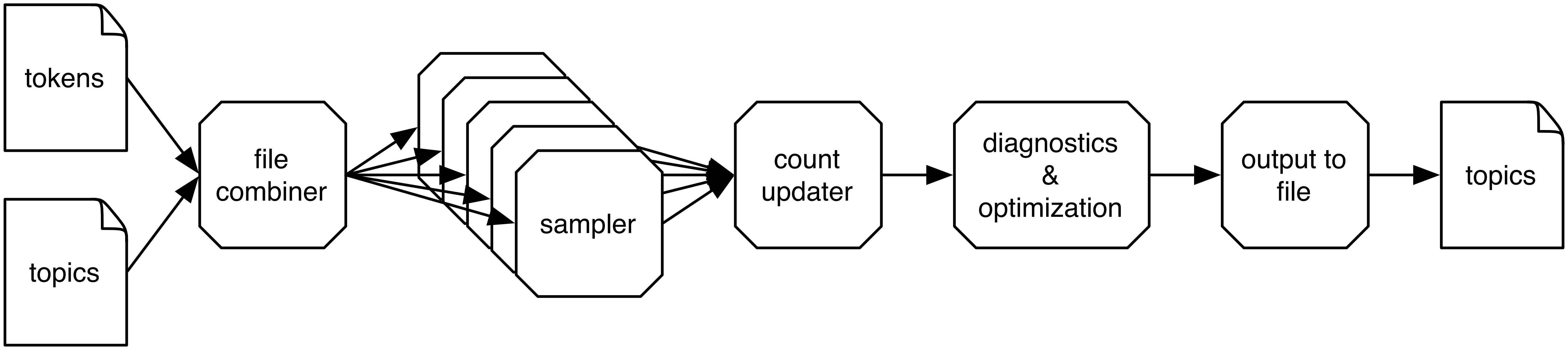
minimal view

continuous sync

barrier free

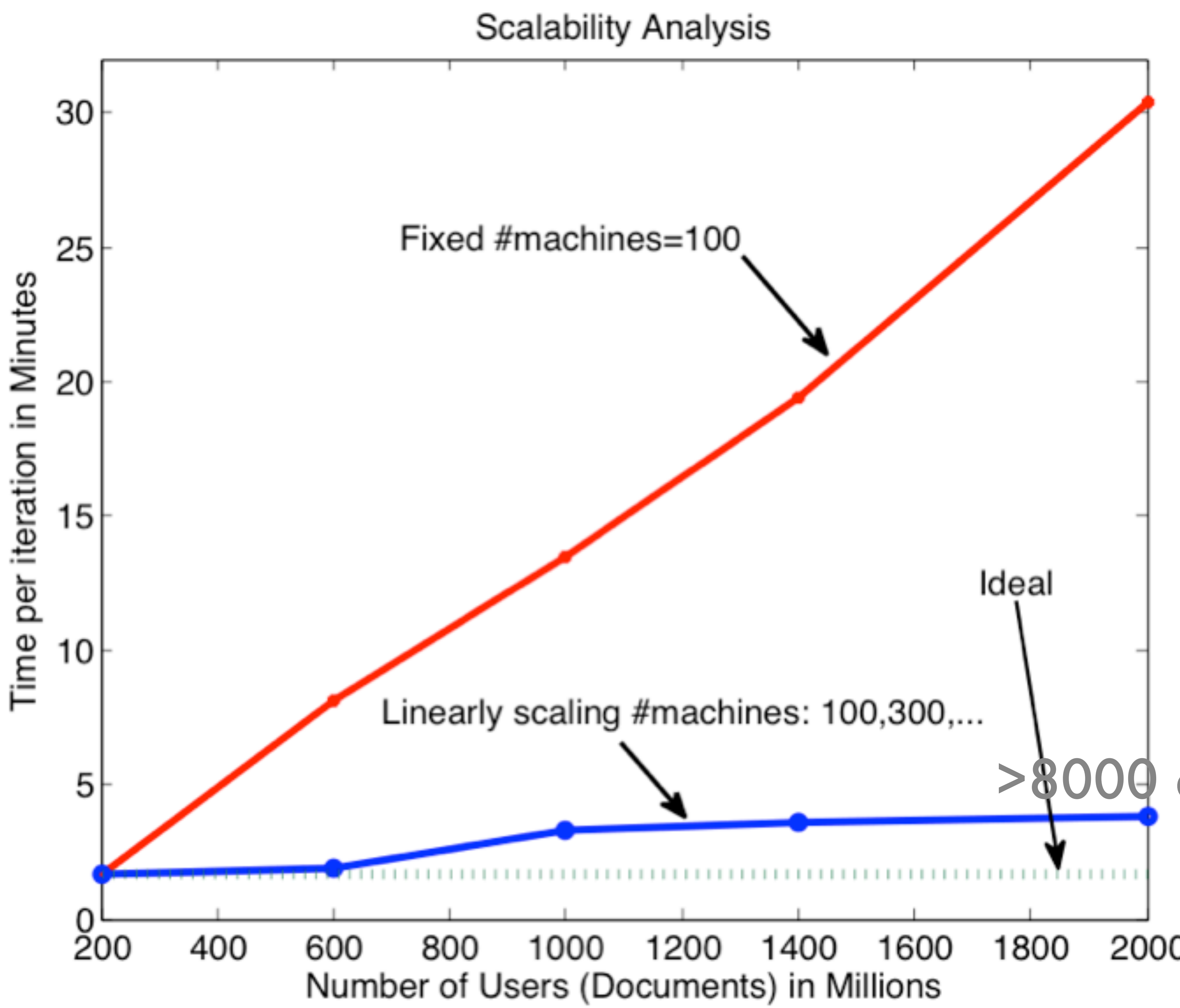
YAHOO!

Multicore Architecture



- Decouple multithreaded sampling and updating (almost) avoids stalling for locks in the sampler
- Joint state table
  - much less memory required
  - samplers syncronized (10s vs. m/proc delay)
- Hyperparameter update via stochastic gradient descent
- No need to keep documents in memory (streaming OK)

Scalability



>8000 documents/s

Outlook

- Convex optimization
- Parameter compression
- Distributed sampling
- **Fast** nonlinear function classes
- Data streams (sketches & statistics)

- Graphs, FAWN architectures, relational data, bandit-like settings, applications