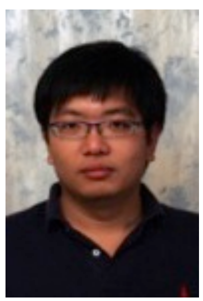
GraphLab

Large-Scale Distributed Machine Learning

Carlos Guestrin

Joseph Gonzalez



Yucheng Low

Aapo Kyrola

Haijie Gu

Joseph Bradley

Danny Bickson

Needless to Say, We Need Machine Learning for Big Data

6 Billion
Flickr Photos

28 Million
Wikipedia Pages

1 Billion
Facebook Users

72 Hours a Minute
YouTube

The New York Times
SundayReview

WORLD U.S. N.Y. / REGION BUSINESS TEC

NEWS ANALYSIS

The Age of Big Data

By STEVE LOHR
Published: February 11, 2012

"… data a new class of economic asset, like currency or gold."

Big Learning

How will we
design and **implement**
parallel learning systems?

Part 1

ASYNCHRONOUS DATA-PARALLEL ALGORITHMS

Sparse Regression

[Tibshirani, 1996]

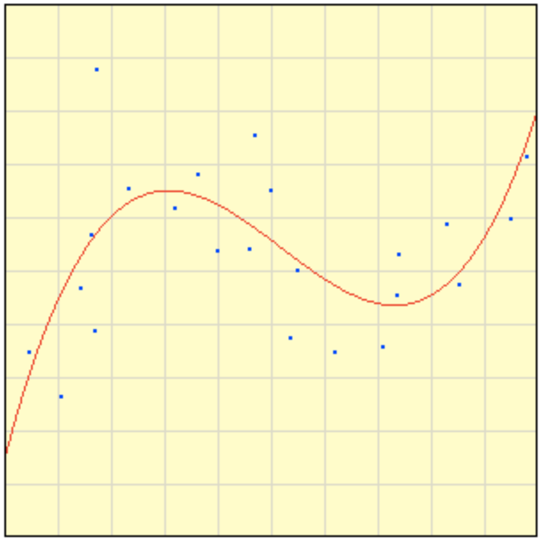
$$y \approx w_0 + a w_1 + a^2 w_2 + a^3 w_3 + \dots$$

$$= Aw$$

target

weights

basis functions



LASSO: **find sparse weight vector w***

$$\min_w F(w)$$

$$F(w) = ||y - Aw||_2^2 + \lambda ||w||_1$$

least squares

sparsity inducing regularizer

- Fundamental machine learning task
- Huge number of applications (many thousands of papers)
  - Computational biology, computer vision, compressed sensing…

Shooting: Stochastic Coordinate Descent (SCD)

[e.g., Shalev-Shwartz & Tewari '09]

While not converged

- Choose random coordinate j
- Optimize w_j
  (closed-form minimization)

$F(w)$ contour

Lasso: $\min_{w} F(w)$ where $F(w) = \frac{1}{2} \| \mathbf{A}w - \mathbf{y} \|_2^2 + \lambda \| w \|_1$

Coordinate Descent for LASSO (aka Shooting Algorithm)

- Repeat until convergence
  - Pick a coordinate *j* at (random or sequentially)
    - Set:

$$\hat{w}_\ell = \begin{cases} (c_\ell + \lambda)/a_\ell & c_\ell < -\lambda \\ 0 & c_\ell \in [-\lambda, \lambda] \\ (c_\ell - \lambda)/a_\ell & c_\ell > \lambda \end{cases}$$

    - Where:

$$a_\ell = 2\sum_{j=1}^{N}(h_\ell(\mathbf{x}_j))^2$$

$$c_\ell = 2\sum_{j=1}^{N} h_\ell(\mathbf{x}_j)\left(t(\mathbf{x}_j) - (w_0 + \sum_{i\neq\ell} w_i h_i(\mathbf{x}_j))\right)$$

Analysis of SCD [Shalev-Shwartz, Tewari '09/'11]

- Theorem: With iterations T, expected error decreases as:

$$\mathcal{O}\left(\frac{d\ \gamma\ ||w^*||^2}{T}\right)$$

  - For d dimensions, and optimum w*
  - For (coordinate-wise) strongly convex functions ($\Delta w = \delta_{wj}\, e_j$):

$$F(w+\Delta w) \leq F(w) + |\Delta w|(\nabla F(w))_j + \frac{\gamma|\Delta w|^2}{2}$$

  - For LASSO γ=1, for Logistic Regression γ=1/4

Great rate…
but gets expensive in high dimensions

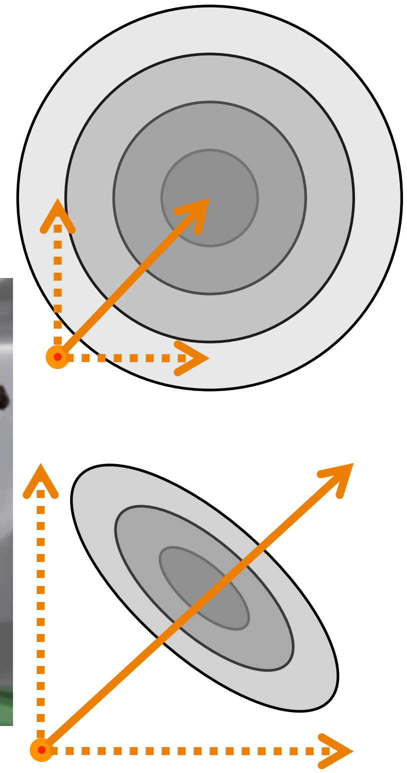
Shotgun: Data-Parallel SCD

[Bradley, Kyrola, Bickson, G. '11]

While not converged

- On each of *P* processors
  - Choose random coordinate j
  - Optimize w_j (as in Shooting)

Is coordinate descent inherently sequential?



Nice case: Uncorrelated features

Bad case: Correlated features

Lasso: $\min_{w} F(w)$ where $F(w) = \frac{1}{2} \| \mathbf{A}w - \mathbf{y} \|_2^2 + \lambda \| w \|_1$

Is SCD inherently sequential?

Lasso: $\min_{w} F(w)$ where $F(w) = \| Xw - \mathbf{y} \|_2^2 + \lambda \| w \|_1$

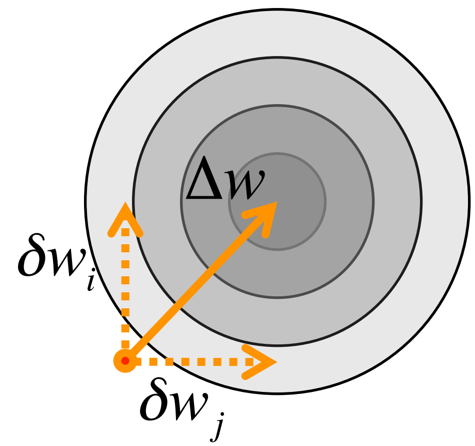
Coordinate update:

$$w_j \leftarrow w_j + \delta w_j$$

(closed-form minimization)

Collective update:

$$\Delta w = \begin{pmatrix} \delta w_i \\ 0 \\ 0 \\ \delta w_j \\ 0 \end{pmatrix}$$



Is SCD inherently sequential?

Lasso: $\min_w F(w)$ where $F(w) = \| Xw - \mathbf{y} \|_2^2 + \lambda \| w \|_1$

Lemma: If X is normalized s.t. diag($X^T X$)=1,

$$F(w + \Delta w) - F(w)$$

$$\leq -\sum_{i_j \in \mathcal{P}} \left(\delta w_{i_j}\right)^2 + \sum_{\substack{i_j, i_k \in \mathcal{P}, \\ j \neq k}} \left(X^{\mathrm{T}} X\right)_{i_j, i_k} \delta w_{i_j} \delta w_{i_k}$$

"Positive" progress

Can be positive or negative ☹

Key term!

(Measures "correlation" between features…)

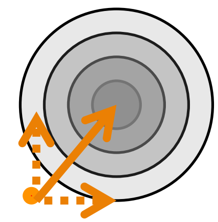
"interference" between updates

Theorem: Shotgun Convergence

Assume $P < d/\rho + 1$

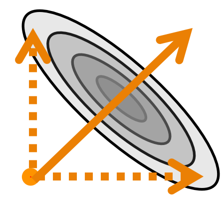
where ρ = largest eigenvalue of $\mathbf{A}^T\mathbf{A}$

Nice case:
Uncorrelated
features



$\rho = 1 \Rightarrow P_{max} = d$

Bad case:
Correlated
features



$\rho = d \Rightarrow P_{max} = 1$ (at worst)

Then: can achieve linear speed ups with up to *P* processors

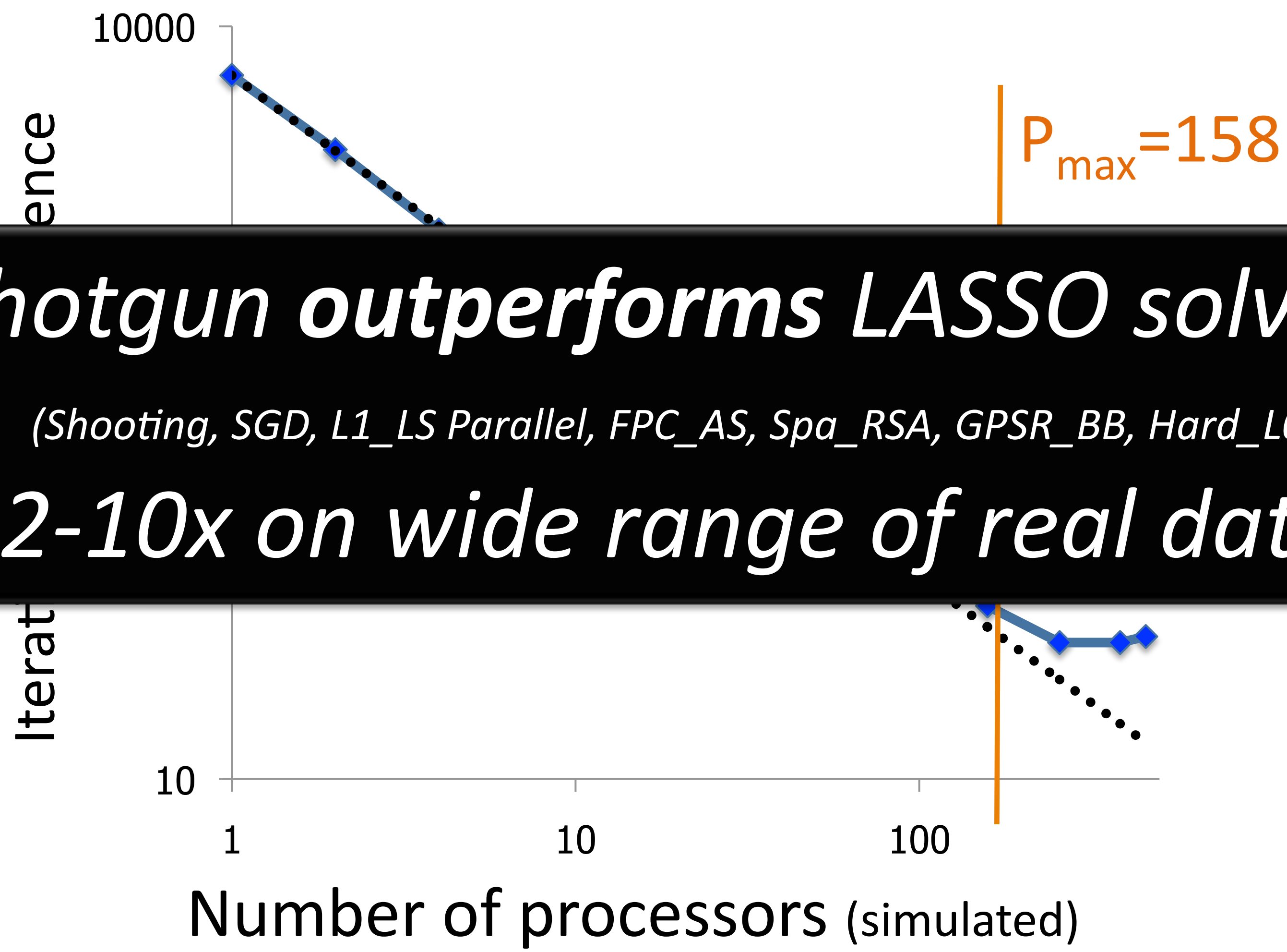
final - opt
objective

$$E\left[F(w^{(T)})\right] - F(w^*) \le \frac{d\left(\frac{1}{2}\|w^*\|_2^2 + F(w^{(0)})\right)}{TP}$$

iterations

parallel updates

Experiments Match Theory!



Shotgun* outperforms *LASSO solvers

(Shooting, SGD, L1_LS Parallel, FPC_AS, Spa_RSA, GPSR_BB, Hard_L0)

2-10x on wide range of real data

Mug32, single pixel camera dataset

Key Proof Technique

- Parallel optimization problem
  - Potential interference between updates
    - Guarantee based on bounding magnitude of interference

Stepping Back…

- Stochastic coordinate ascent (SCD)
  - Optimization: Pick a coordinate *j*, find $argmin_{wj}\ F(w)$
  - Parallel SCD: Pick *p* coordinates and update at once
  - Issue: Updates may interfere on *p* coordinates
  - Solution: Bound possible interference using spectral norm

- Natural counterpart: Stochastic gradient descent (SGD)
  - Optimization: Pick a data point and take a small gradient step on all coordinates
  - Parallel: Pick *p* data points and update at once
  - Issue: Updates may interfere on ***all*** coordinates
  - Solution: Bound interference using sparsity of data points

Stochastic Gradient Descent

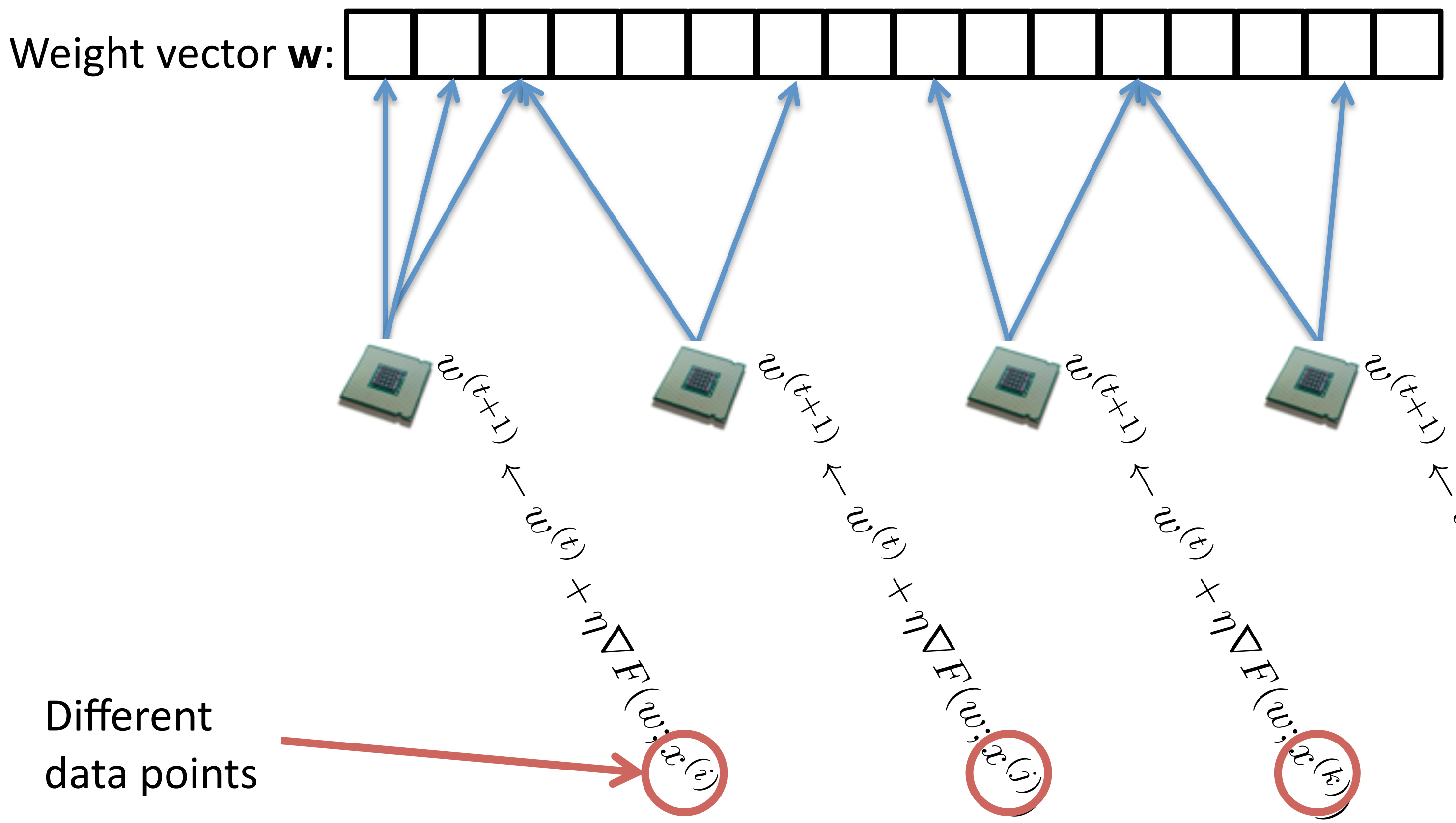
- Coordinate descent updates one coordinate w_j, using all data points

- Stochastic gradient descent updates all coordinates, using one data point $x^{(i)}$:

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \nabla F(w; x^{(i)})$$

Parallel Stochastic Gradient Descent

- Each processor does update using a different data point



Risk versus coordinate descent:
SGD could interfere on all coordinates simultaneously

Parallel SGD with No Locks

[e.g., Hogwild!, Niu et al. '11]

- Each processor in parallel:
  - Pick data point i at random
  - For j = 1…d:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left(\nabla F(w; x^{(i)}) \right)_j$$

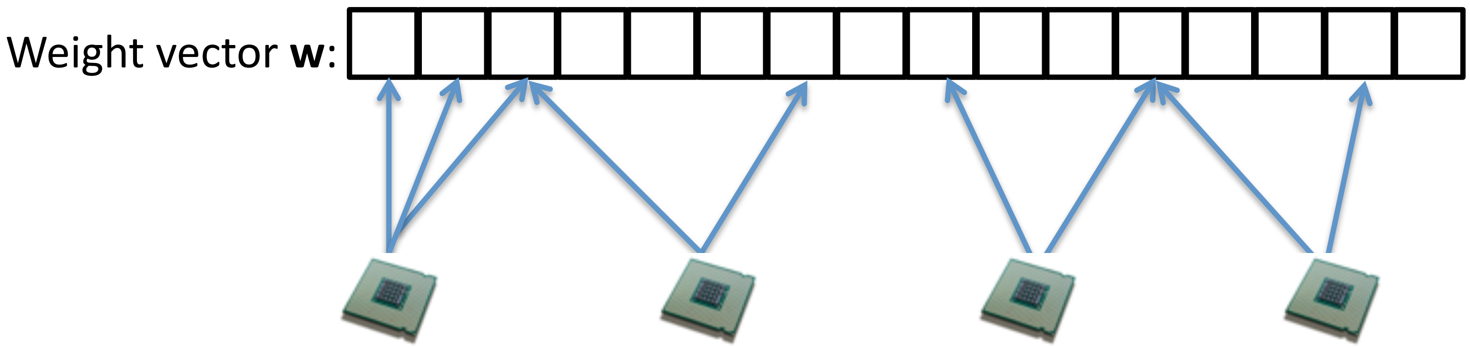
- Assume atomicity of sum operation for a coordinate:

$$w_j \leftarrow w_j + a$$

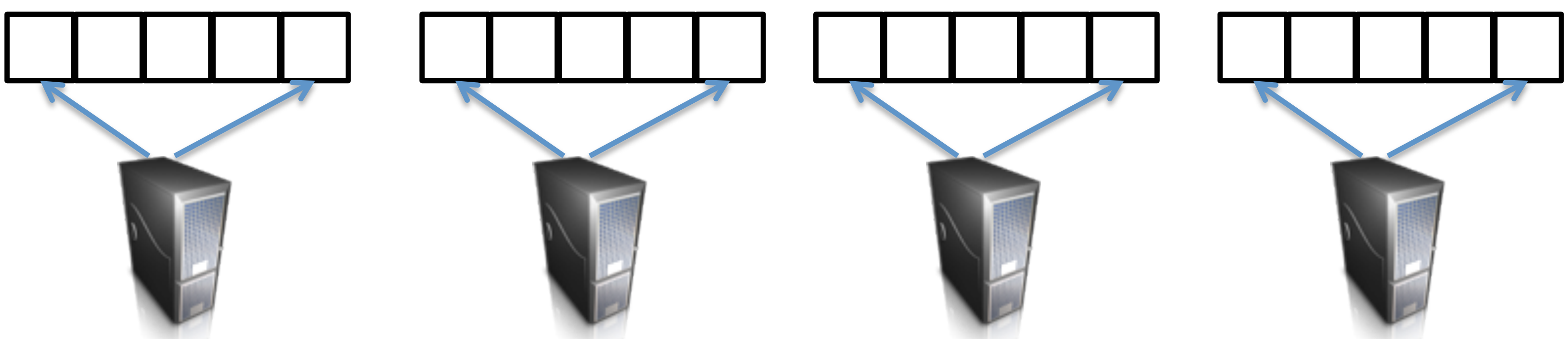
Key to proof of bounded interference:
Assume data points are sparse →
update interferes at most on a few coordinates

Shared Memory versus Distributed Memory

- **Shared memory**: all machines can access same memory space

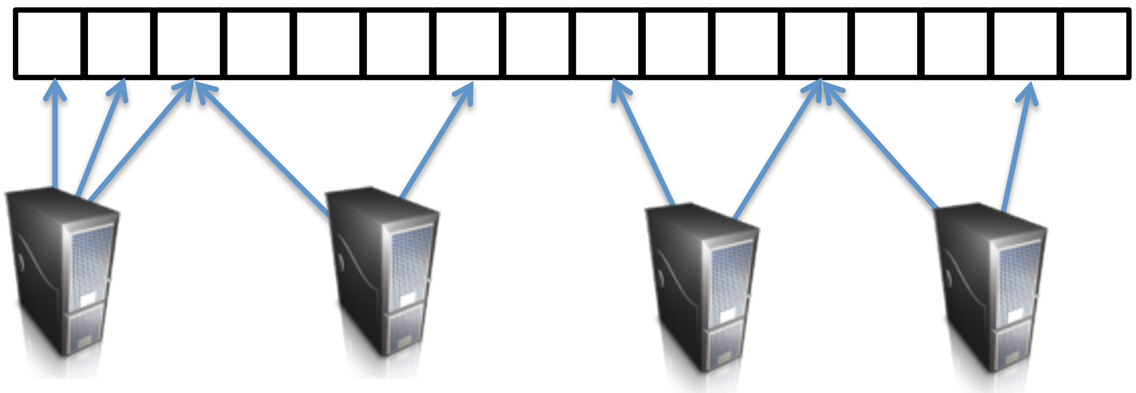


- **Distributed memory**: machines can only access local memory



- Much harder to implement Shotgun or Hogwild!, because of need to synchronize parameters across machines
  - Synchronization can be extremely slow

Distributed Hash Tables (DHTs)

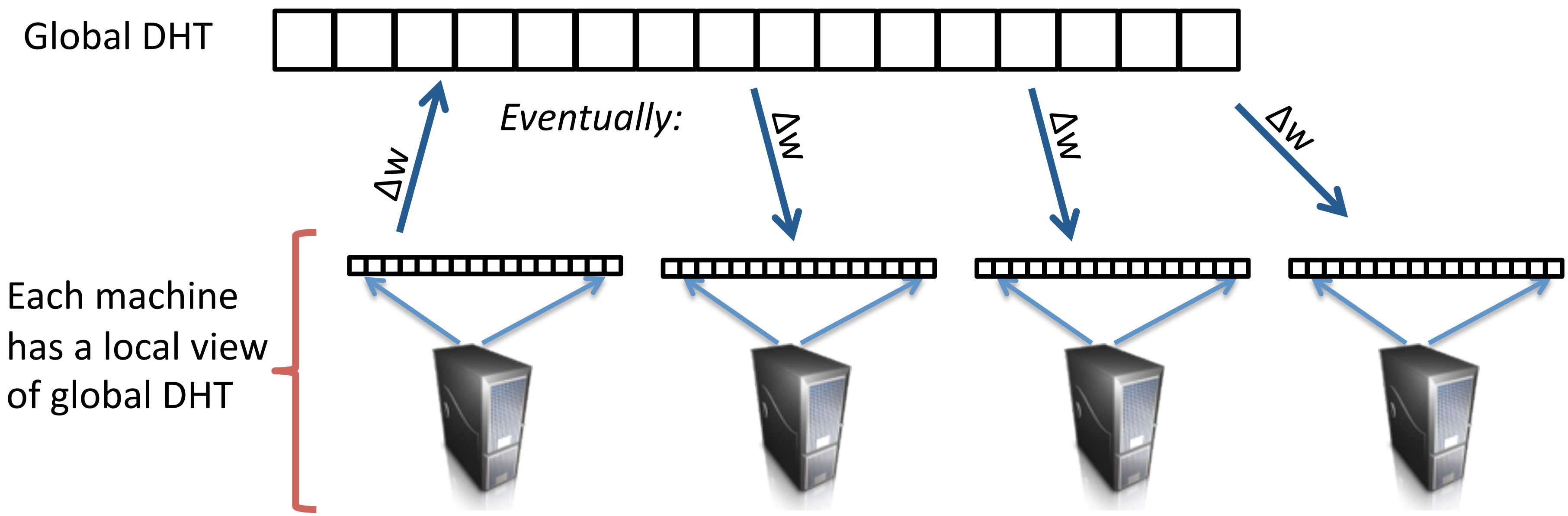


- **DHT**: Distributed memory that looks like shared memory from the programmer's perspective

+	−
Easy to program Guarantees consistency of values read/written	Only really efficient when "large" objects are written/read In ML, an "object" is a parameter, just a double ➔ standard DHTs are too slow

Parameter Servers (e.g., Smola et al.)

- A parameter server is a ***Lazy DHT*** with ***commutative-associative operations***, e.g., $w_j \leftarrow w_j + a$



- Parameter servers only guarantee eventual consistency
- But, often good enough for many distributed learning procedures

Summary of Part 1

- Shotgun/Hogwild! solve distributed optimization by ignoring dependencies in problem
- Key proof method: bound interference in updates
- Implement in distributed settings using parameter servers

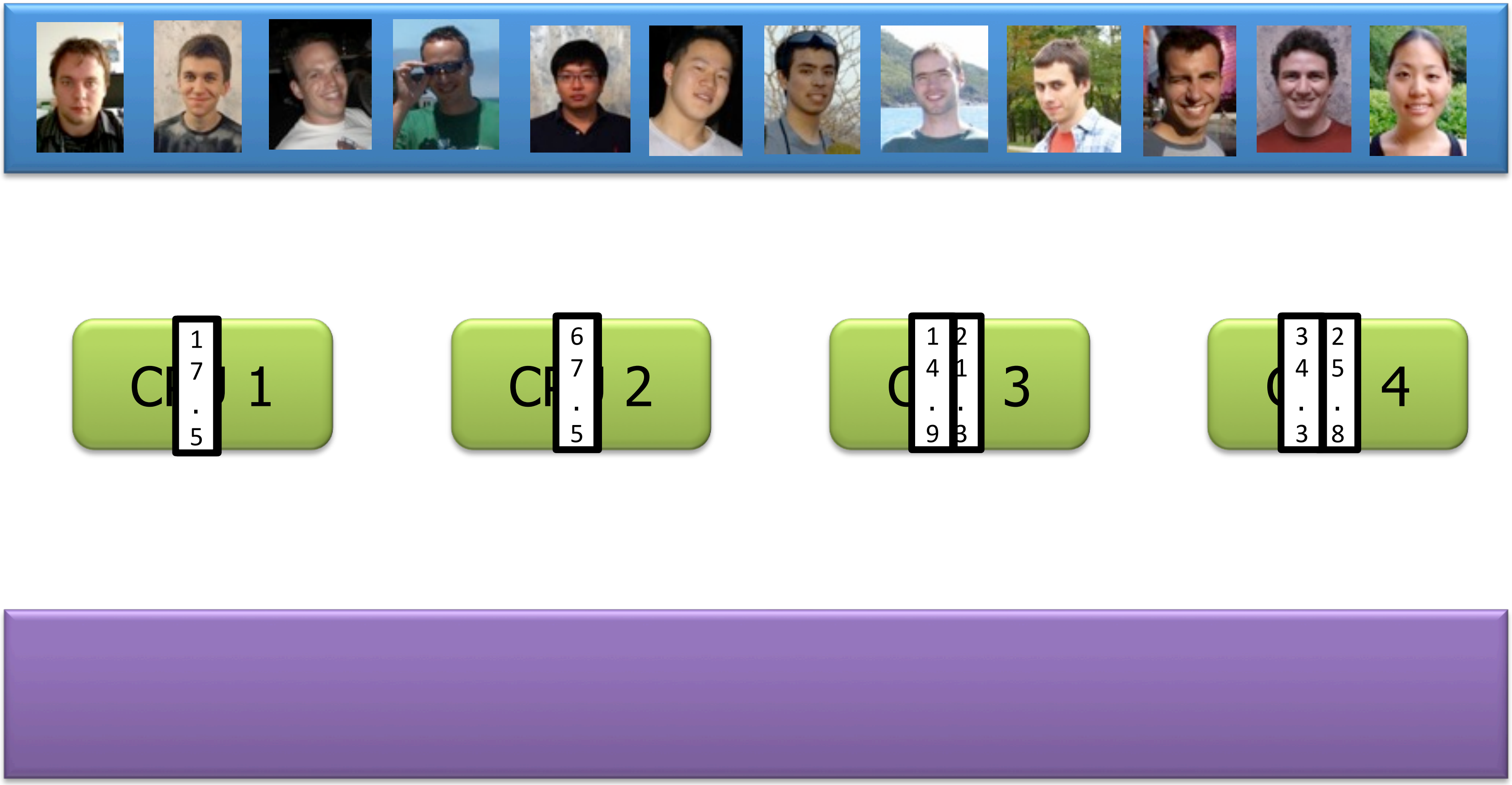
Part 2

ASYNCHRONOUS GRAPH-PARALLEL ALGORITHMS

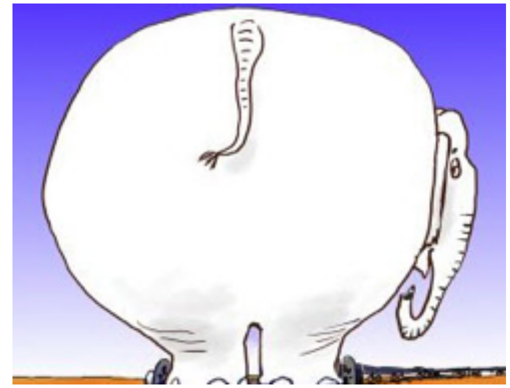
DATA PARALLEL
versus
GRAPH PARALLEL

abstractions

Data Parallelism (MapReduce)



*Solve a huge number of **independent** subproblems*



"A white elephant is a valuable but burdensome possession of which its owner cannot dispose and **whose cost (particularly cost of upkeep) is out of proportion to its usefulness** or worth." *Wikipedia*

Everyone knows has limitations,
nobody happy,
but **what to do next**???

MapReduce for Data-Parallel ML

Excellent for large data-parallel tasks!

Data-Parallel

MapReduce

Feature Extraction

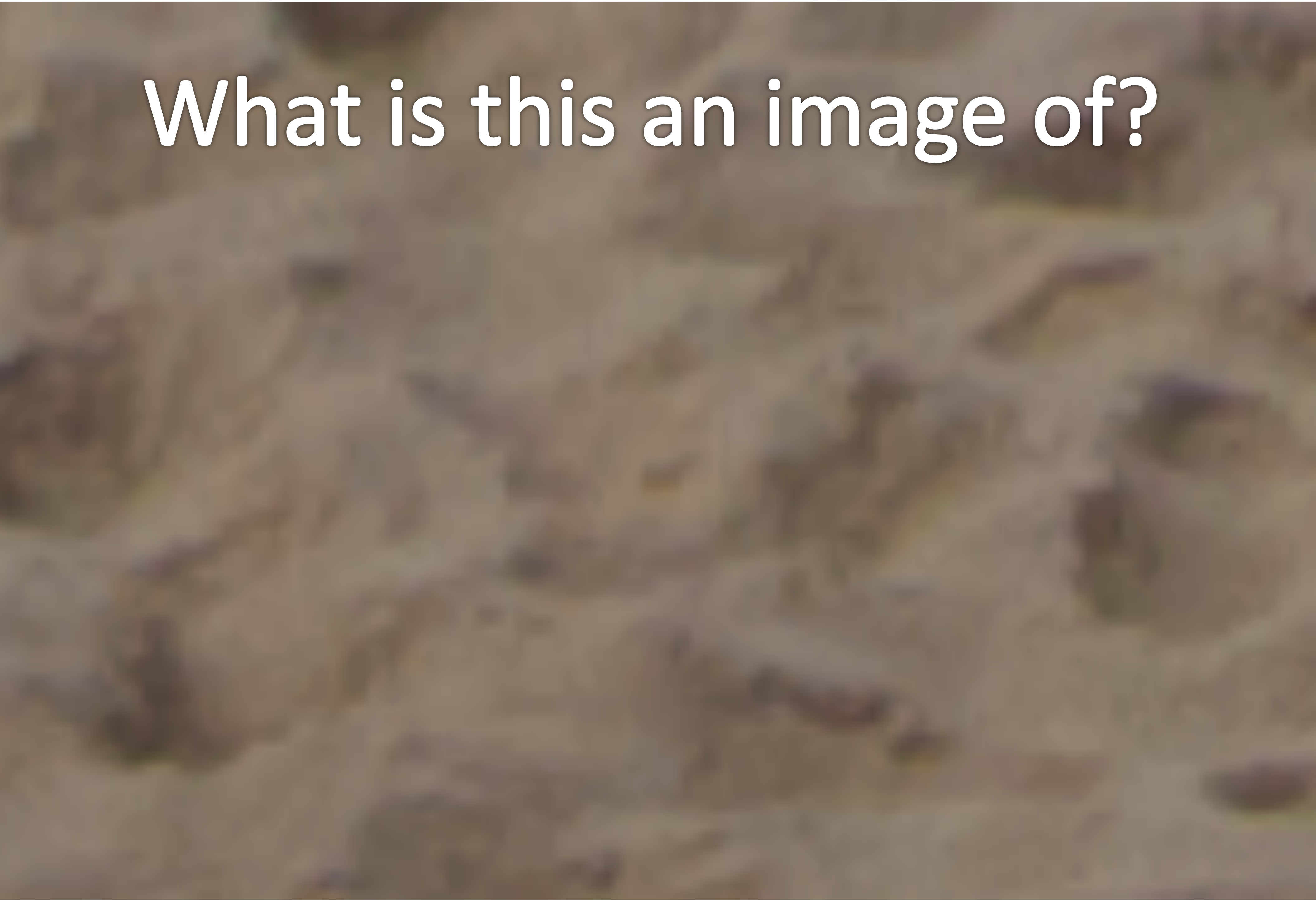
Cross Validation

Computing Sufficient Statistics

Is there more to Machine Learning

?

What is this an image of?



It’s next to this…

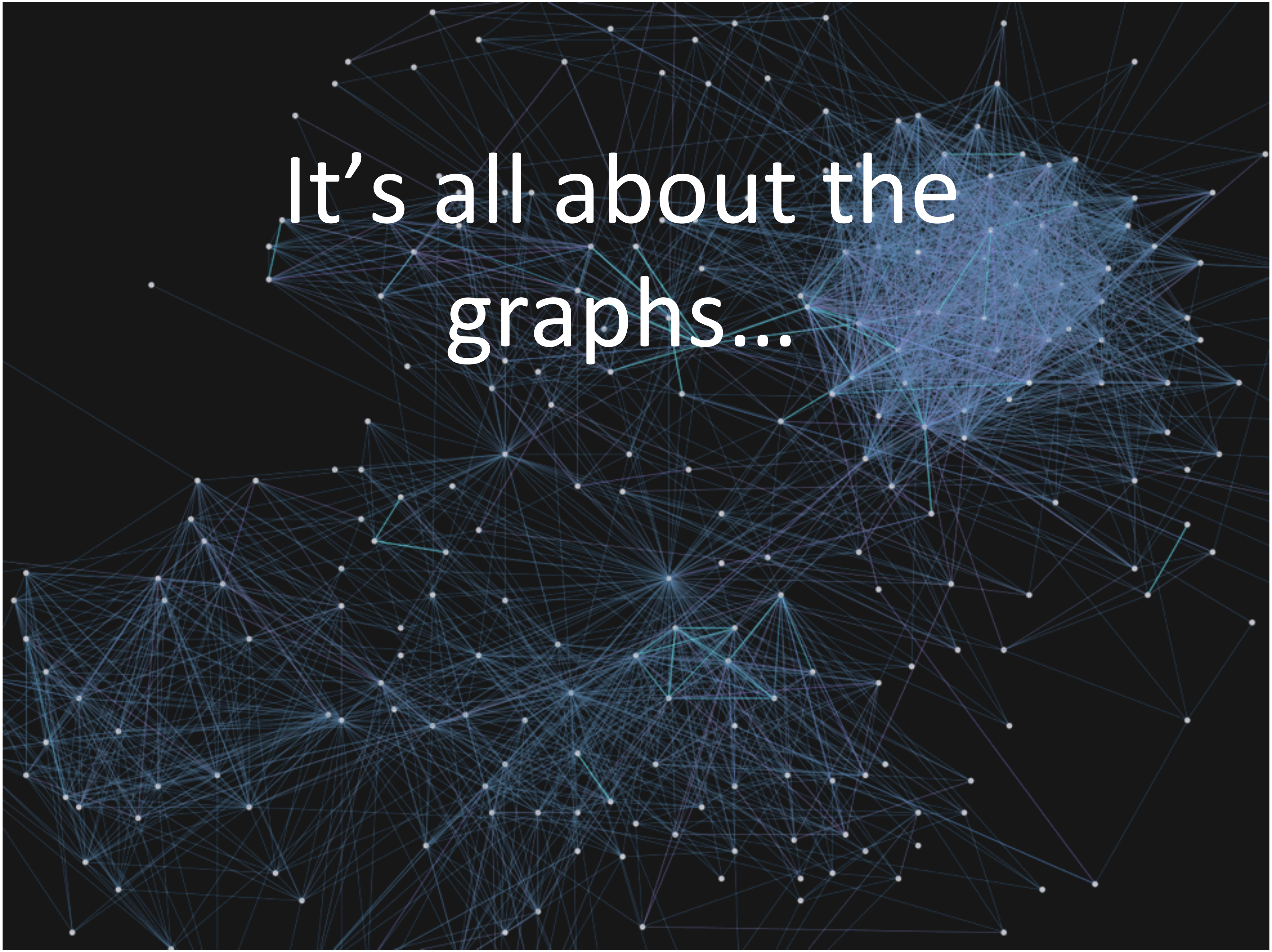
The Power of Dependencies

where the value is!

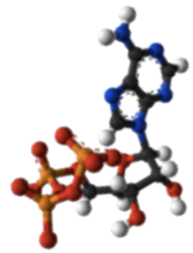
Flashback to 1998

Why?

First Google advantage:
a **Graph Algorithm** & a **System to Support it**!


It’s all about the graphs…

Social Media **Science** **Advertising** **Web**

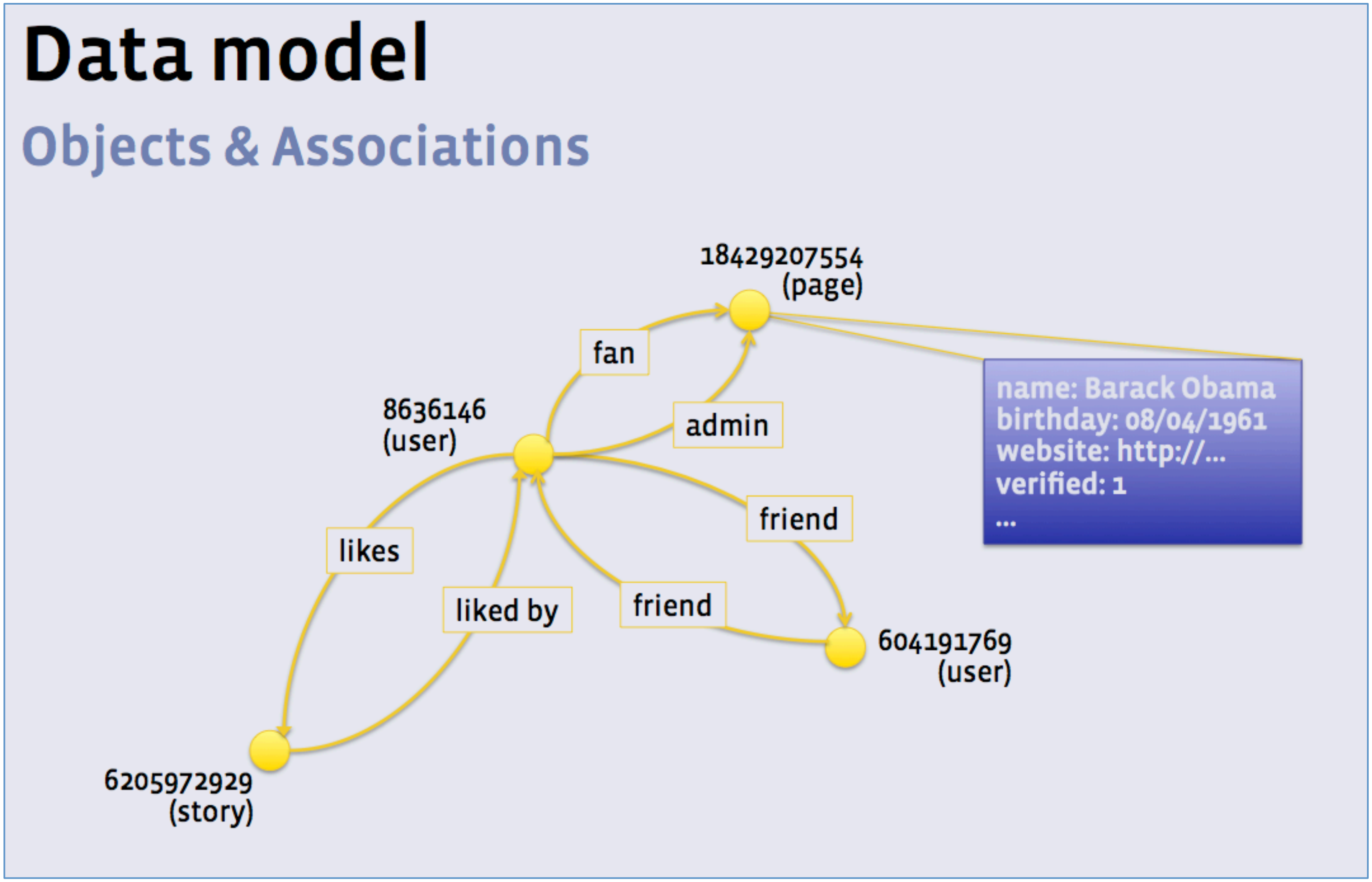


- **Graphs** encode the **relationships** between:

People Products Ideas
Facts Interests

- **Big: 100 billions** of **vertices** and **edges** and rich metadata
  - Facebook (10/2012): 1B users, 144B friendships
  - Twitter (2011): 15B follower edges

Facebook Graph



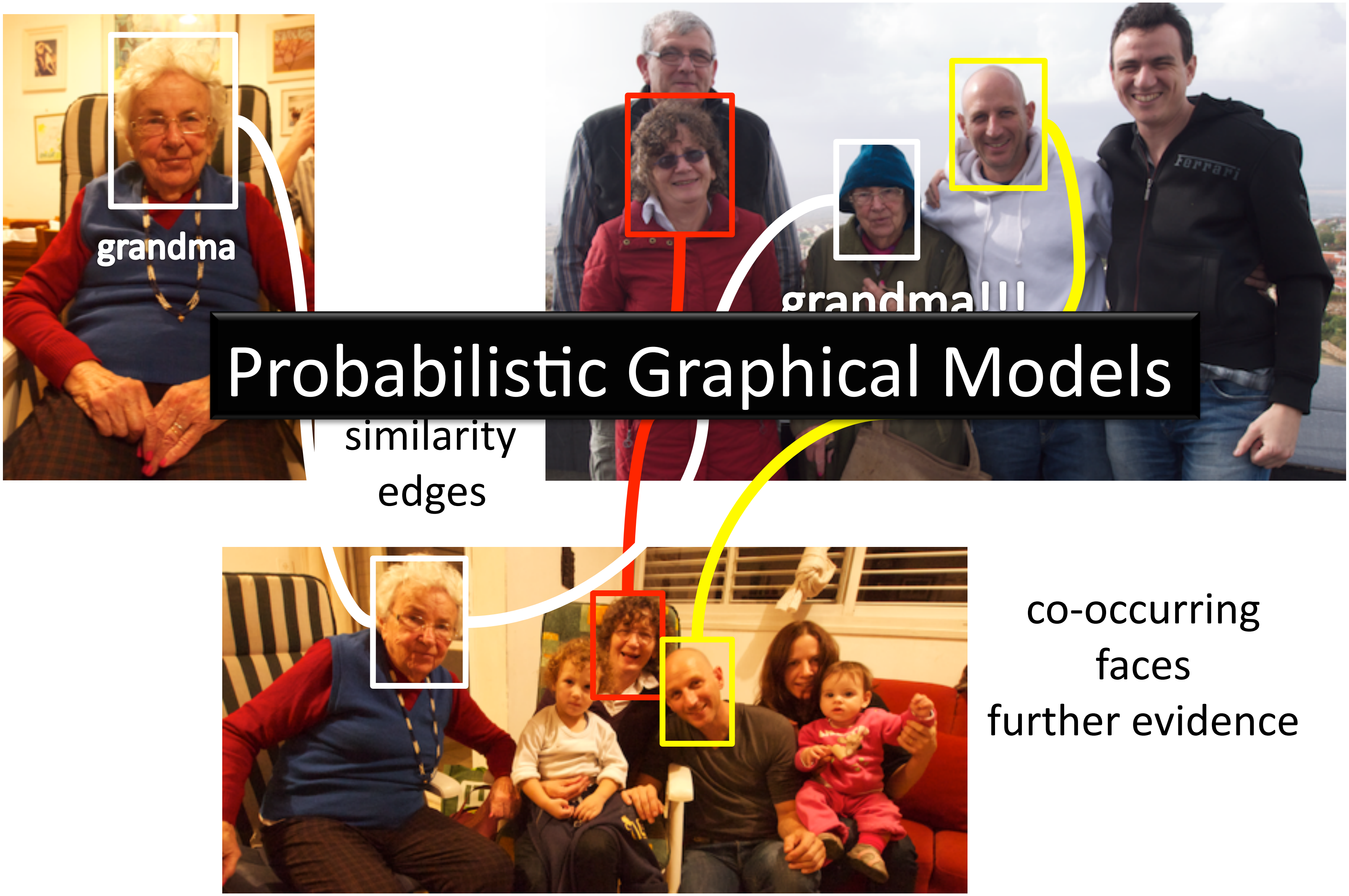
Slide from Facebook Engineering presentation

Examples of Graphs in Machine Learning

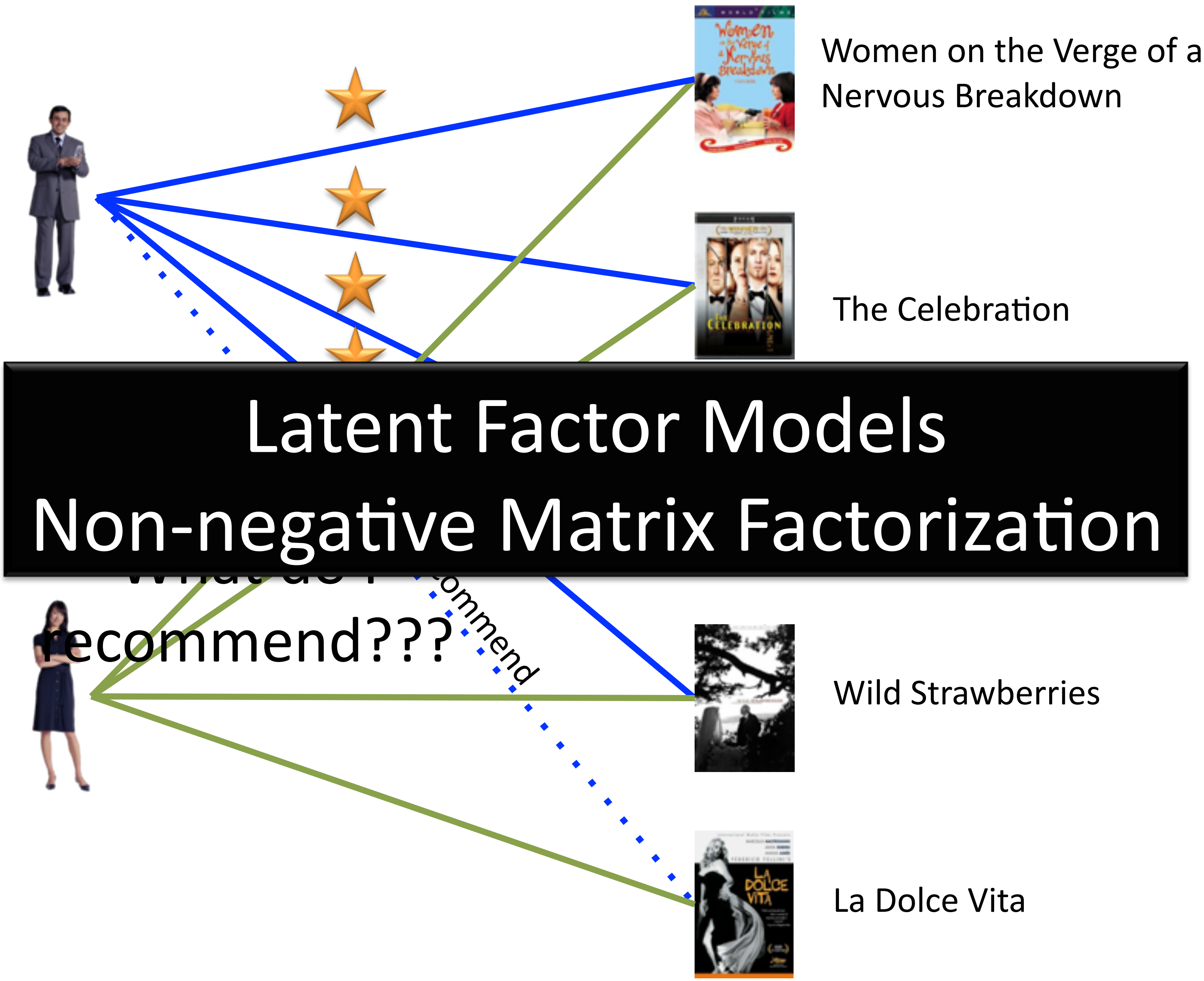
Label a Face and Propagate

Pairwise similarity not enough…

Propagate Similarities & Co-occurrences for Accurate Predictions



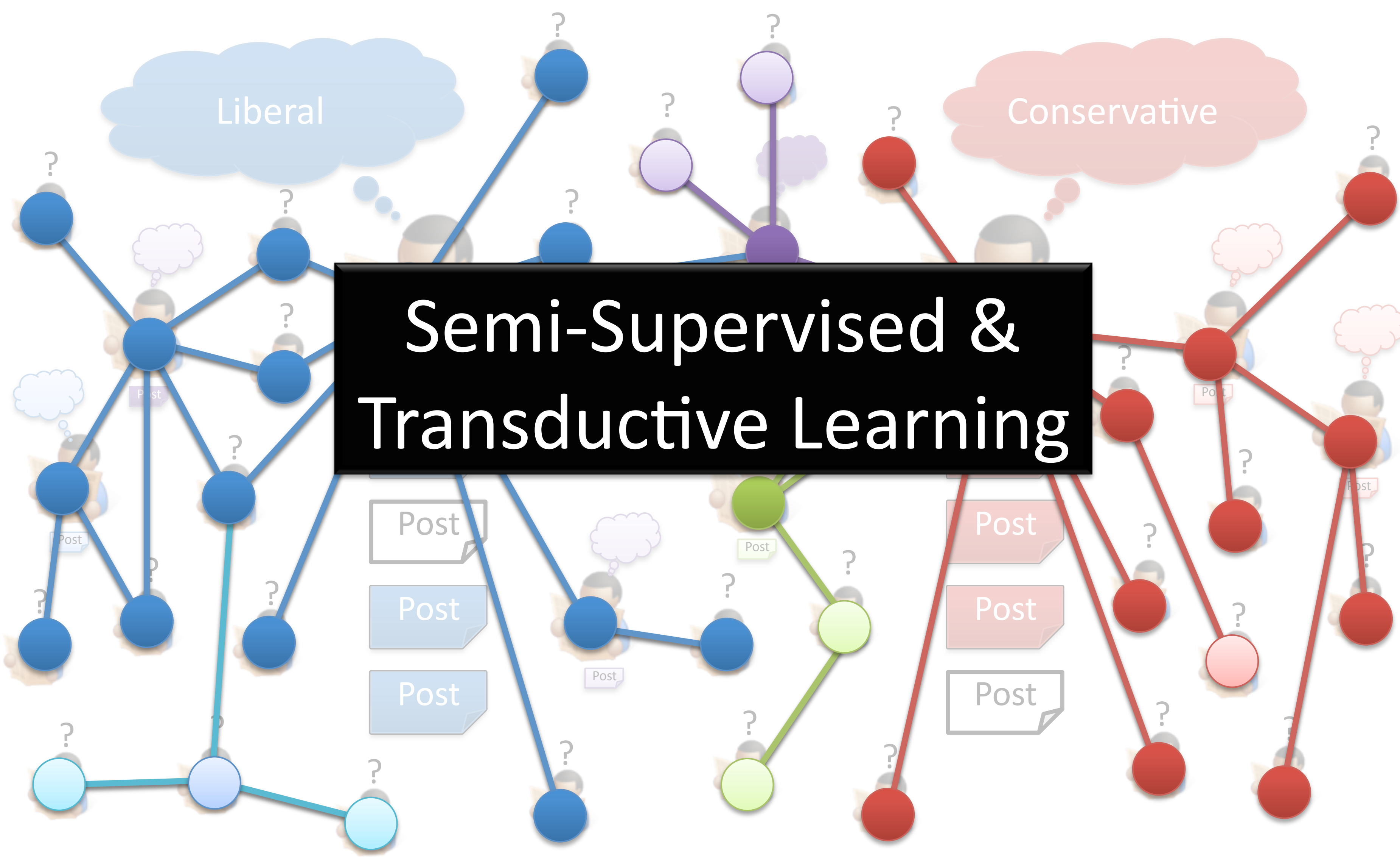
Collaborative Filtering: Exploiting Dependencies



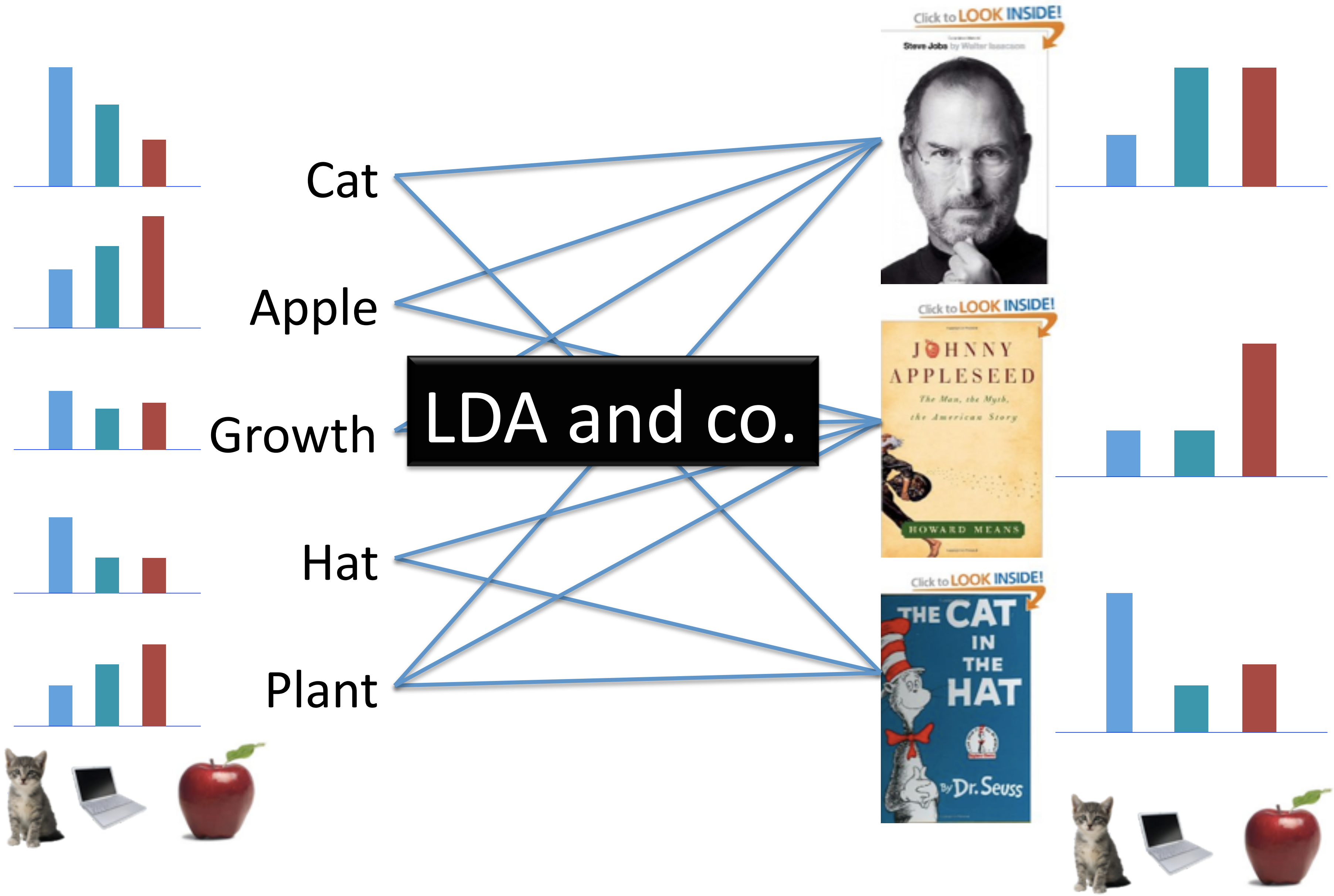
Latent Factor Models

Non-negative Matrix Factorization

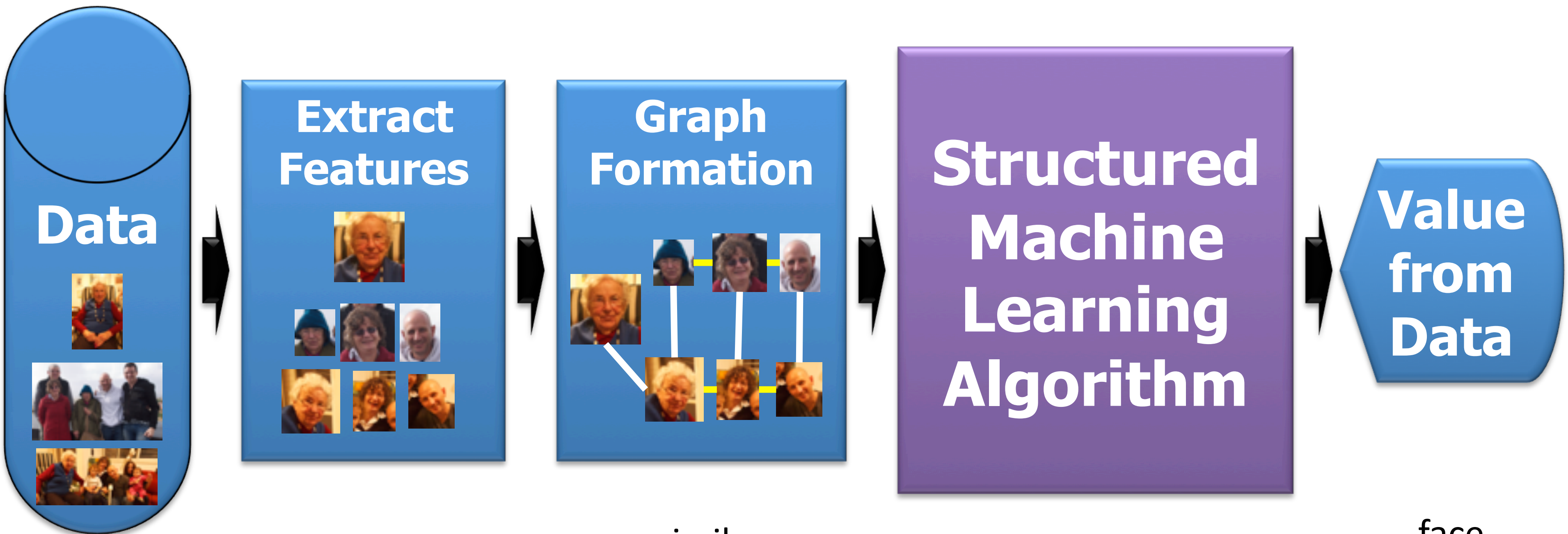
Estimate Political Bias



Topic Modeling



Machine Learning Pipeline



images
docs
movie ratings

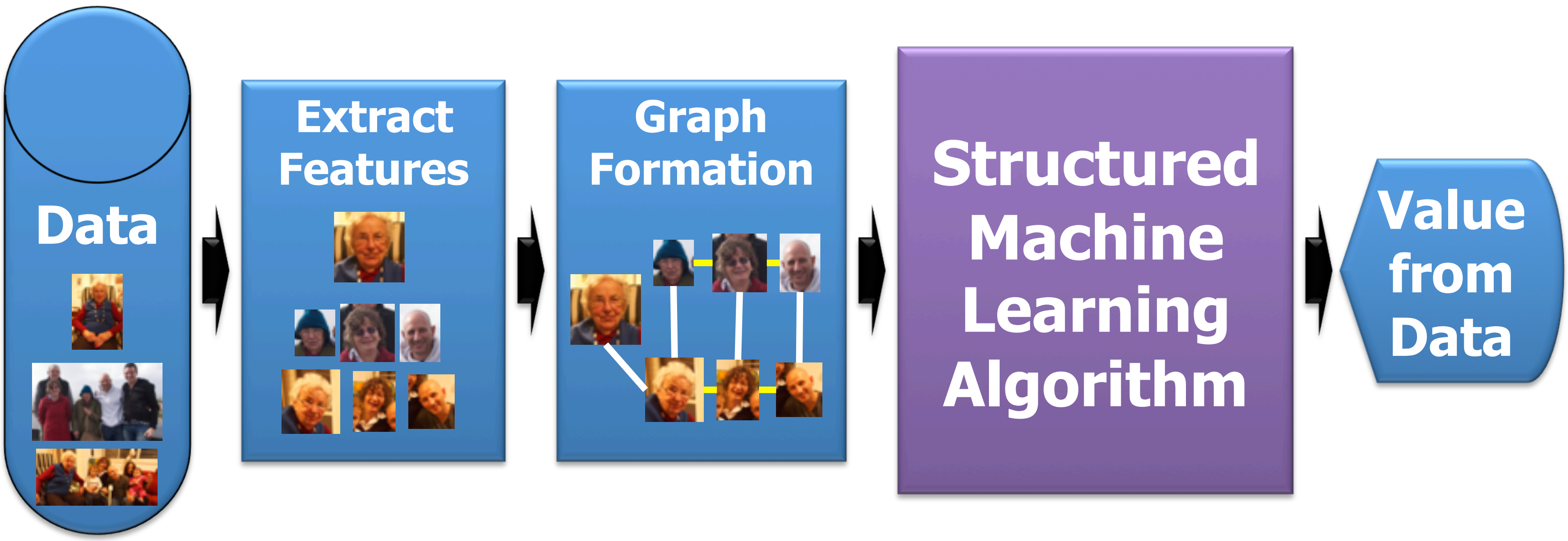
faces
important words
side info

similar faces
shared words
rated movies

belief propagation
LDA
collaborative filtering

face labels
doc topics
movie recommend.

Parallelizing Machine Learning



Graph Ingress

mostly data-parallel

Graph-Structured Computation

graph-parallel

GraphLab

ML Tasks Beyond Data-Parallelism

Data-Parallel

Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

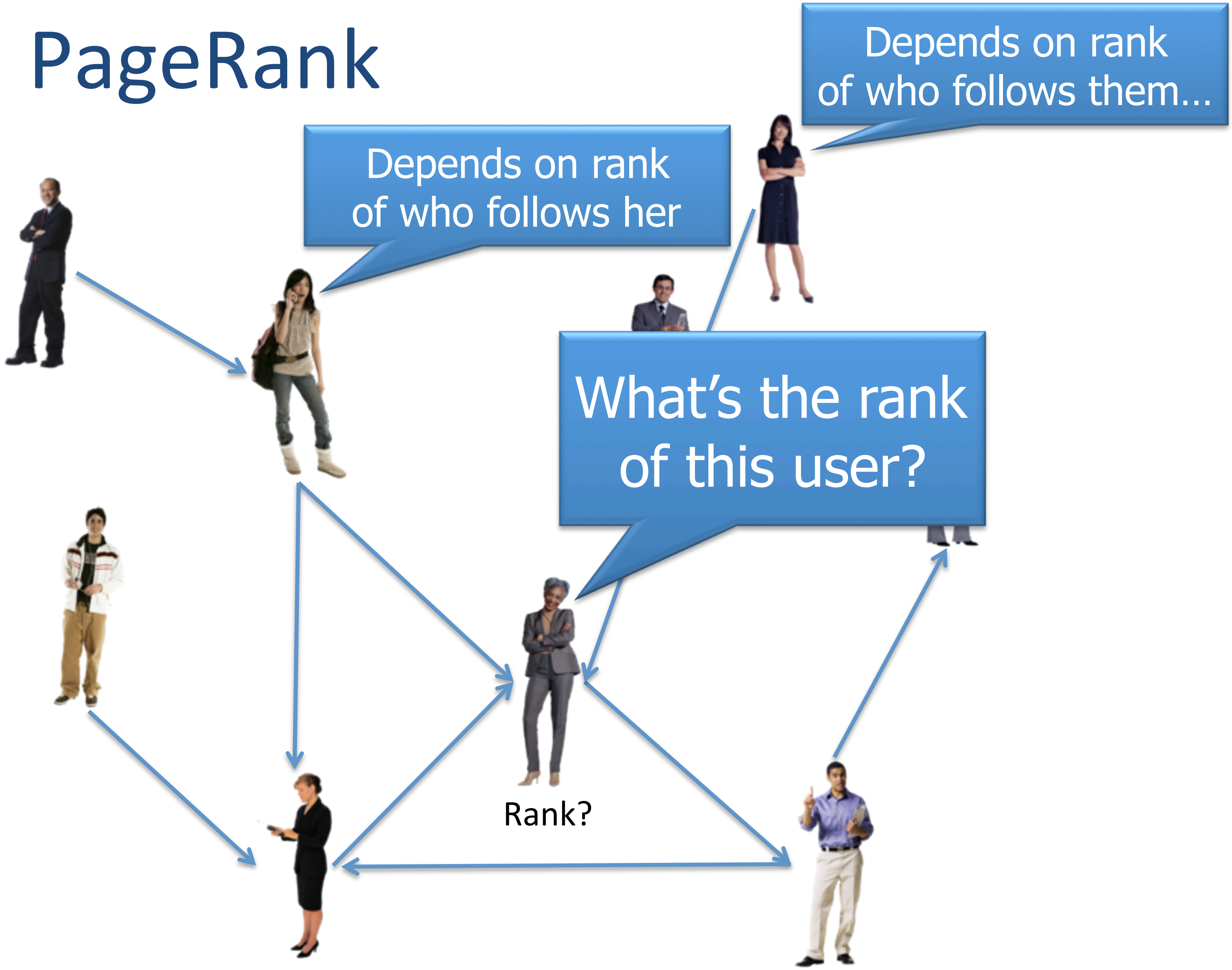
Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

Graph Analysis
PageRank
Triangle Counting

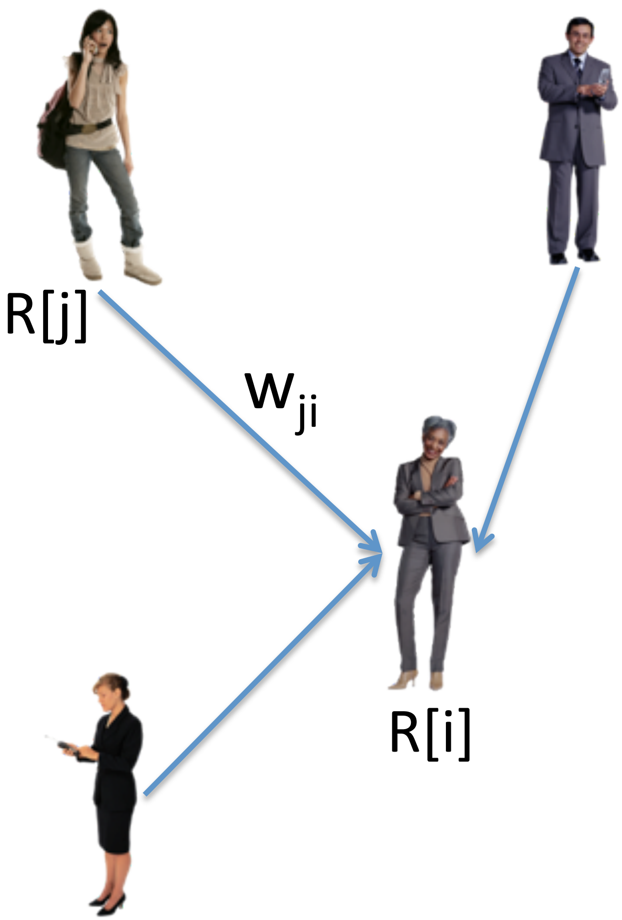
Example of a Graph-Parallel Algorithm

PageRank



Loops in graph → Must iterate!

PageRank Iteration



Iterate until convergence:

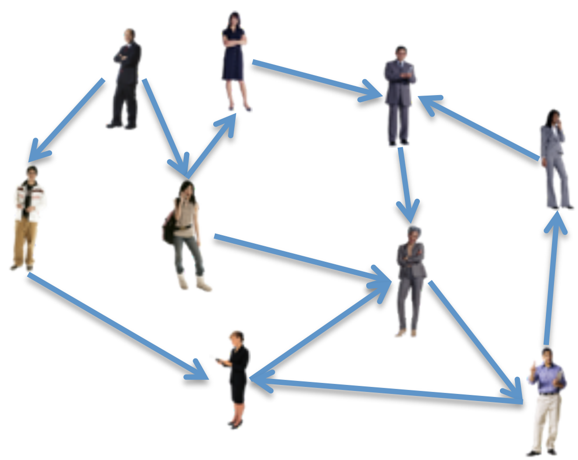
"My rank is weighted average of my friends' ranks"

$$R[i] = \alpha + (1-\alpha) \sum_{(j,i)\in E} w_{ji} R[j]$$

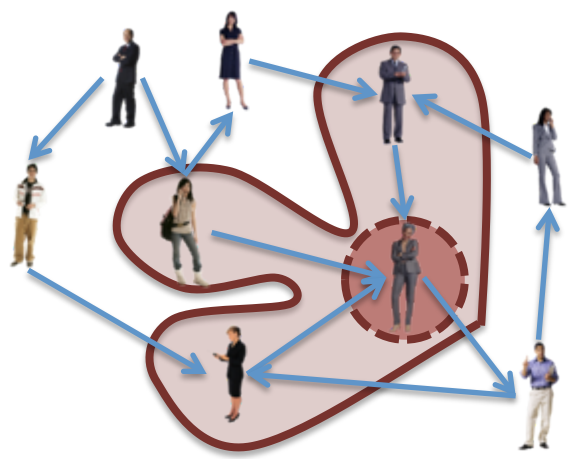
- α is the random reset probability
- w_{ji} is the prob. transitioning (similarity) from j to i

Properties of Graph Parallel Algorithms

Dependency Graph



Local Updates



Iterative Computation

Addressing Graph-Parallel ML

Data-Parallel ⟷ Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graph-Parallel Abstraction

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

Data-Mining
PageRank
Triangle Counting

Graph Computation:

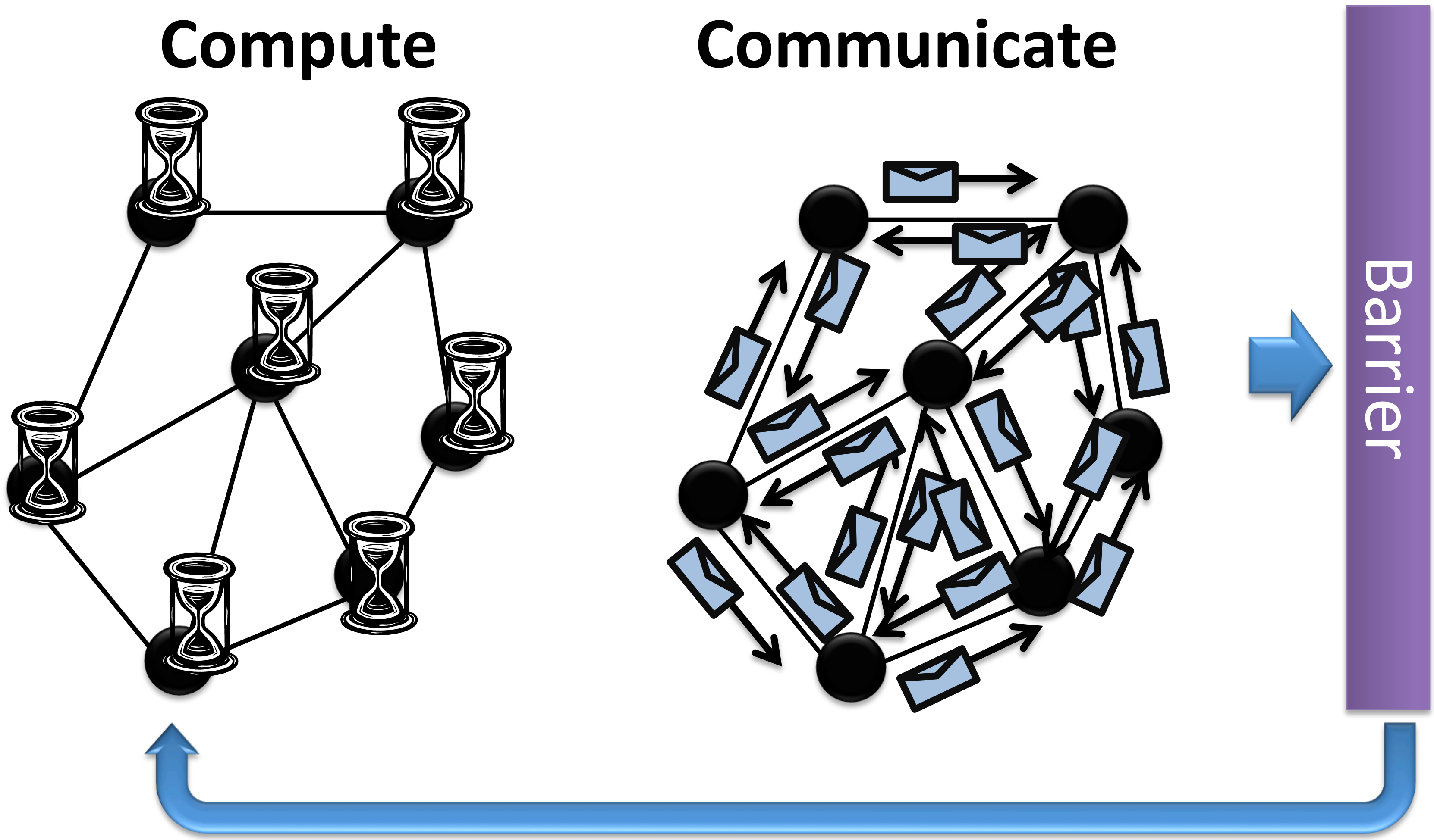
Synchronous

v.

Asynchronous

Bulk Synchronous Parallel Model: Pregel (Giraph)

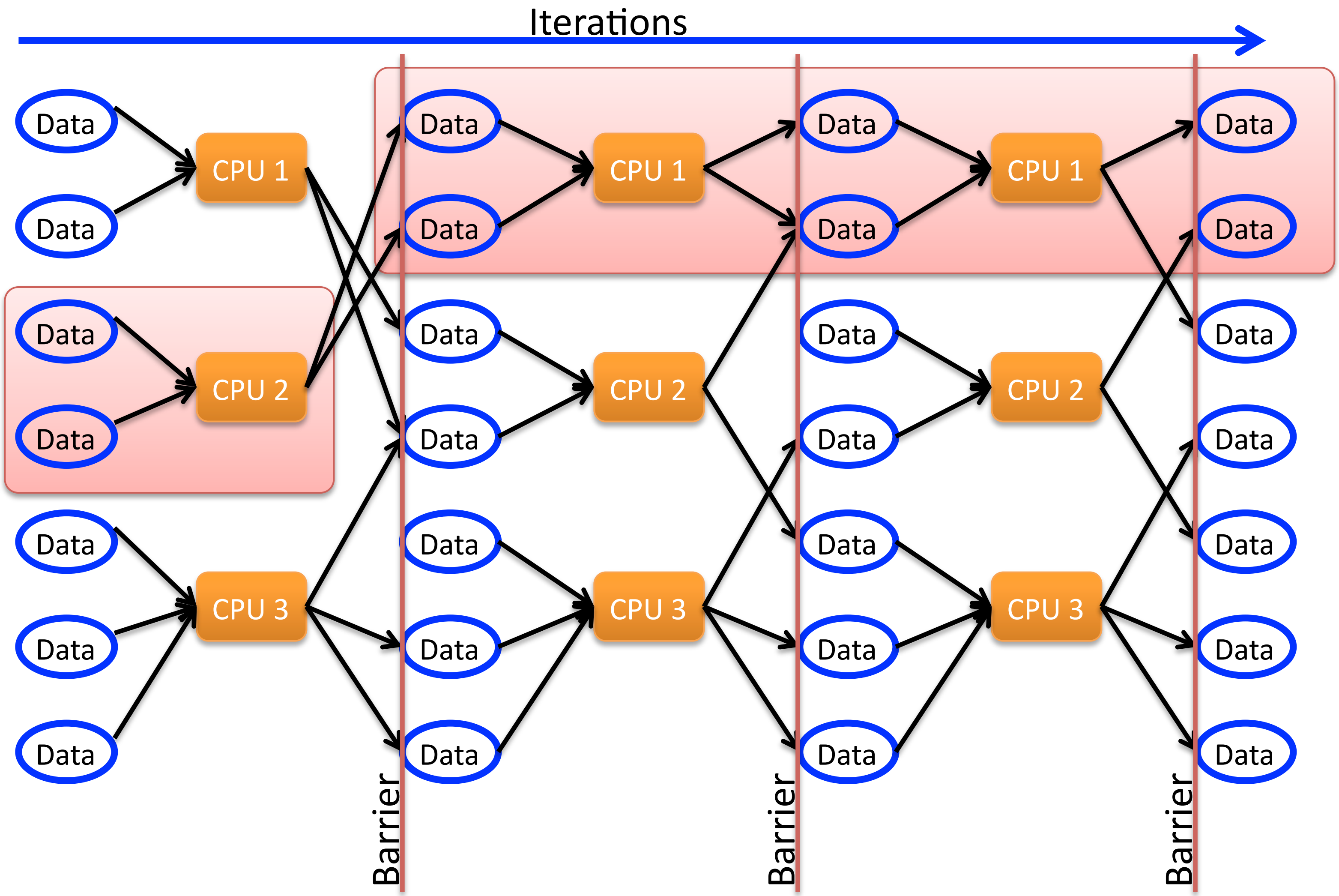
[Valiant '90]



Problem:

Bulk synchronous parallel systems can be ***highly inefficient***

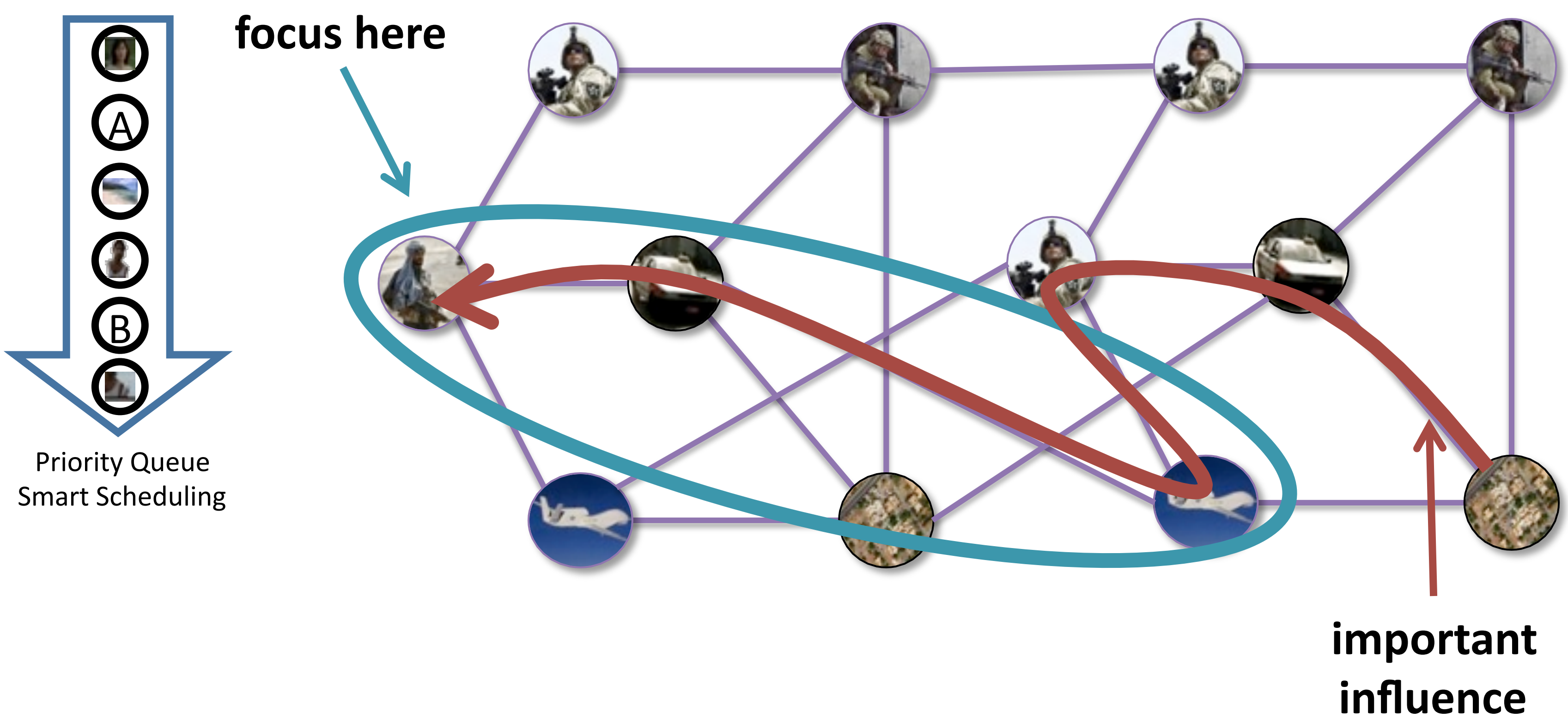
BSP Systems Problem: Curse of the Slow Job



Bulk synchronous parallel model ***provably inefficient*** *for some ML tasks*

Analyzing Belief Propagation

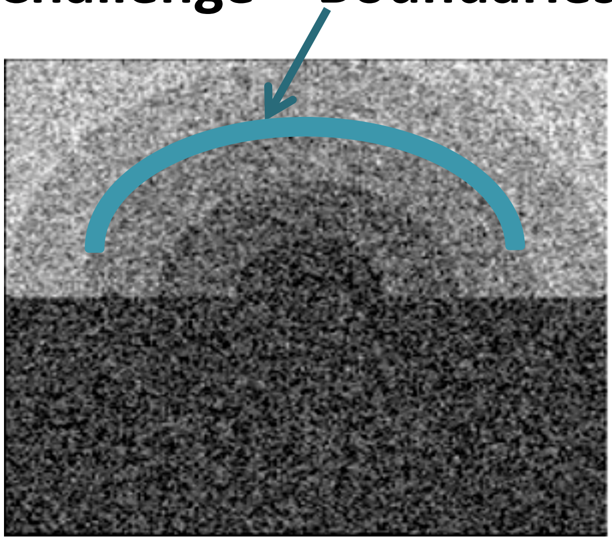
[Gonzalez, Low, G. '09]



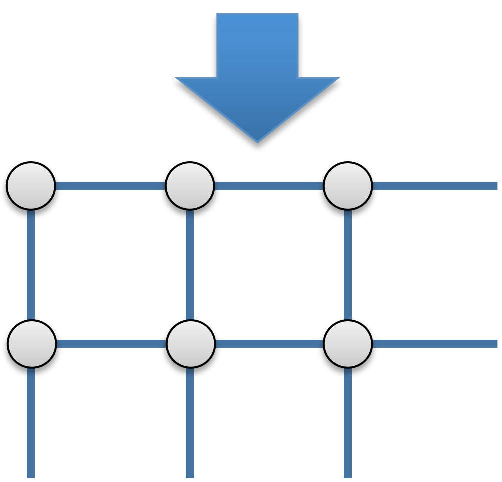
Asynchronous Parallel Model (rather than BSP) fundamental for efficiency

Asynchronous Belief Propagation

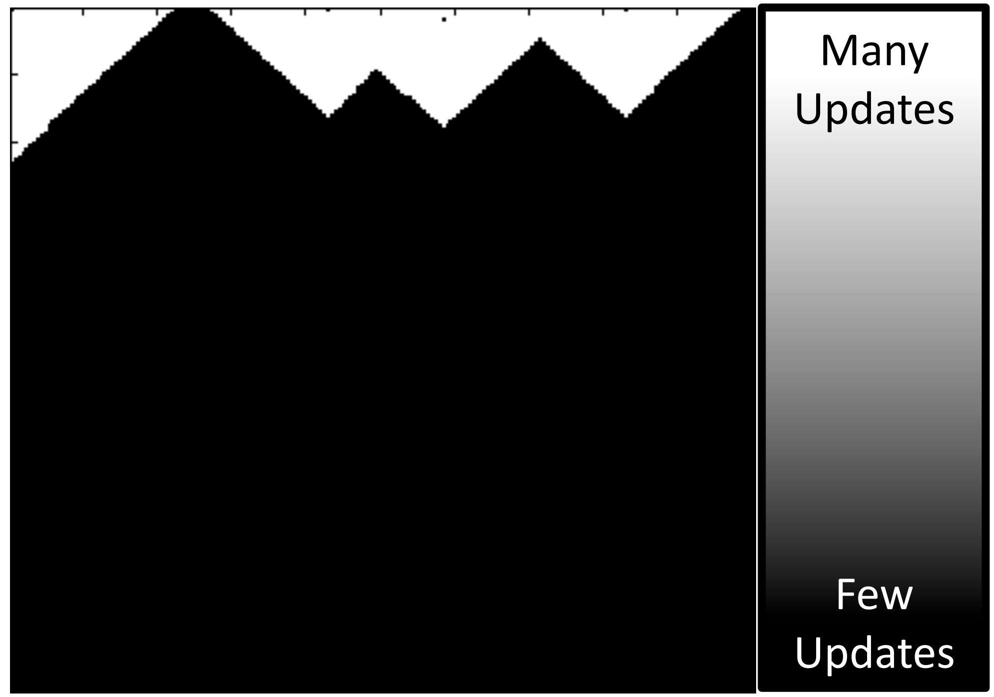
Challenge = Boundaries



Synthetic Noisy Image



Graphical Model

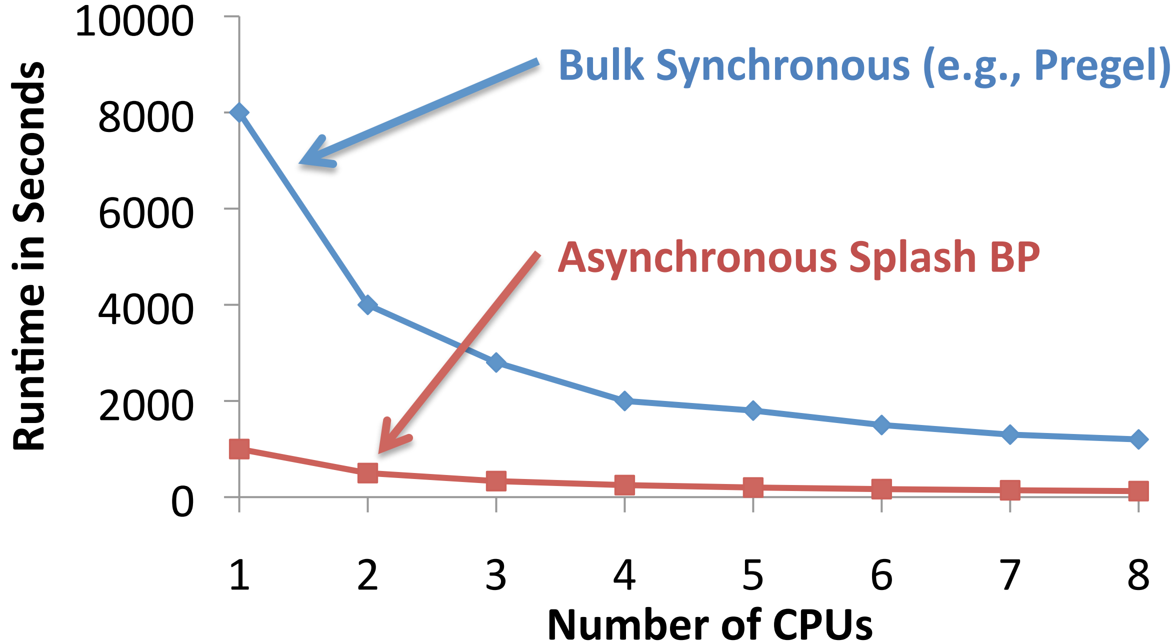


Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

BSP ML Problem:
Synchronous Algorithms can be **Inefficient**

[Gonzalez, Low, G. '09]



Theorem:
Bulk Synchronous BP
O(#vertices) slower
than Asynchronous BP

Efficient parallel implementation was painful, painful, painful…

The Need for a New Abstraction

- Need: Asynchronous, Dynamic Parallel Computations

Data-Parallel ⟷ Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

GraphLab

Carnegie Mellon

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

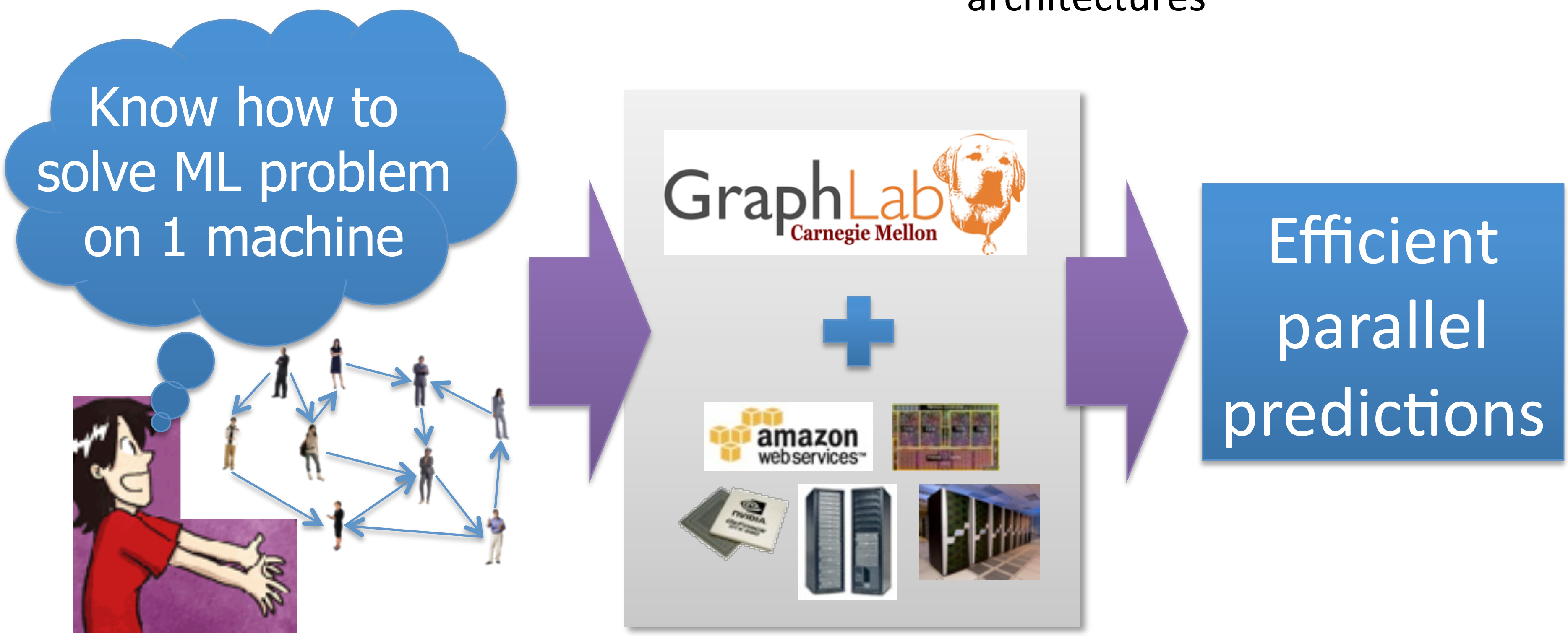
Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

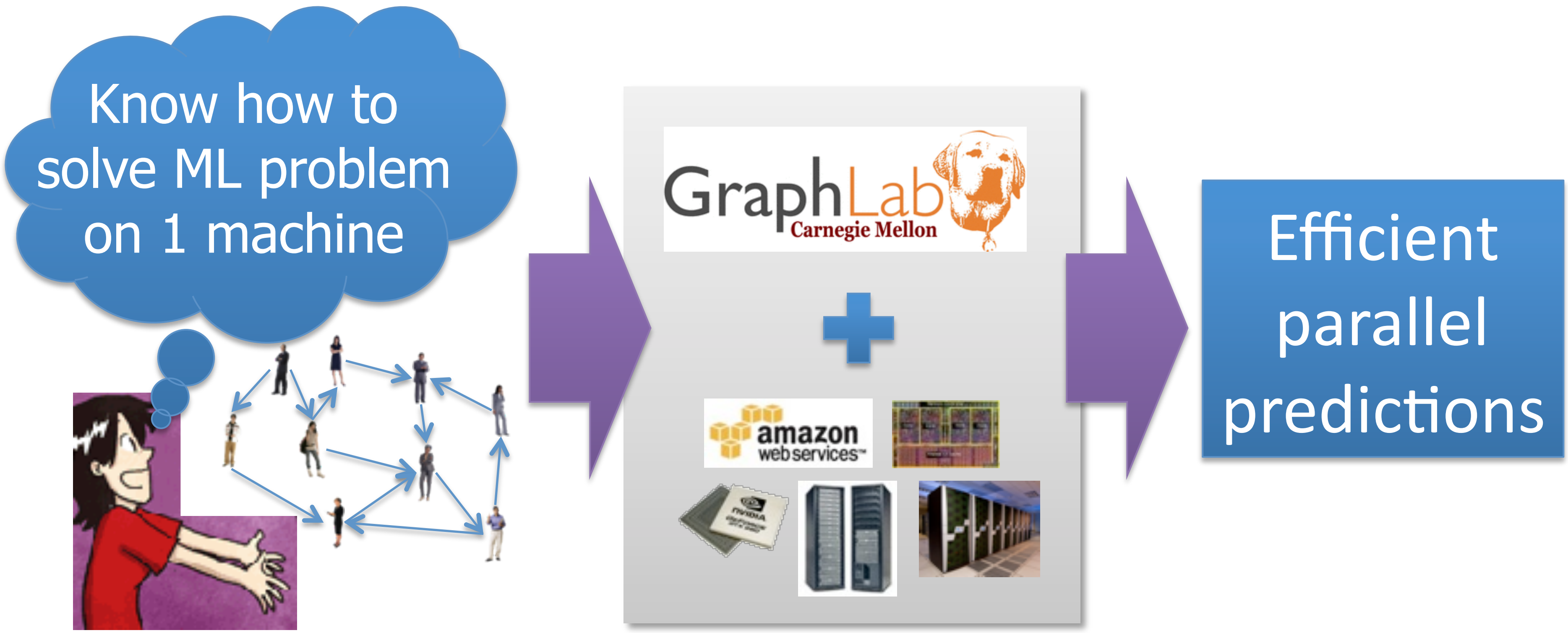
Data-Mining
PageRank
Triangle Counting

The **GraphLab** Goals

- Designed specifically for ML
  - Graph dependencies
  - Iterative
  - Asynchronous
  - Dynamic
- Simplifies design of parallel programs:
  - Abstract away hardware issues
  - Automatic data synchronization
  - Addresses multiple hardware architectures



The **GraphLab** Goals



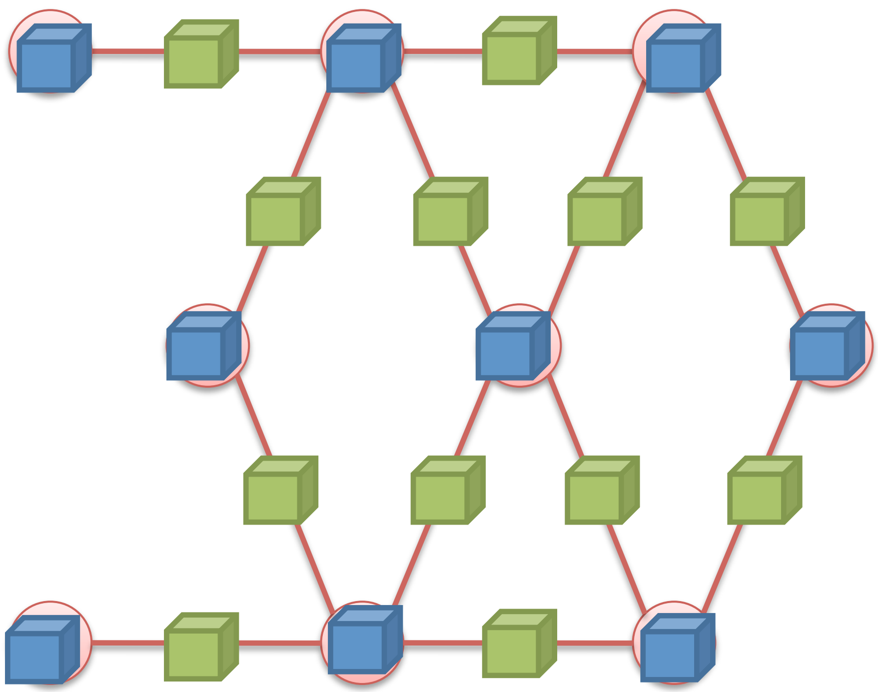
GraphLab 1

Carnegie Mellon

POSSIBILITY

Data Graph

Data associated with vertices and edges



Graph:
- Social Network

Vertex Data:
- User profile text
- Current interests estimates

Edge Data:
- Similarity weights

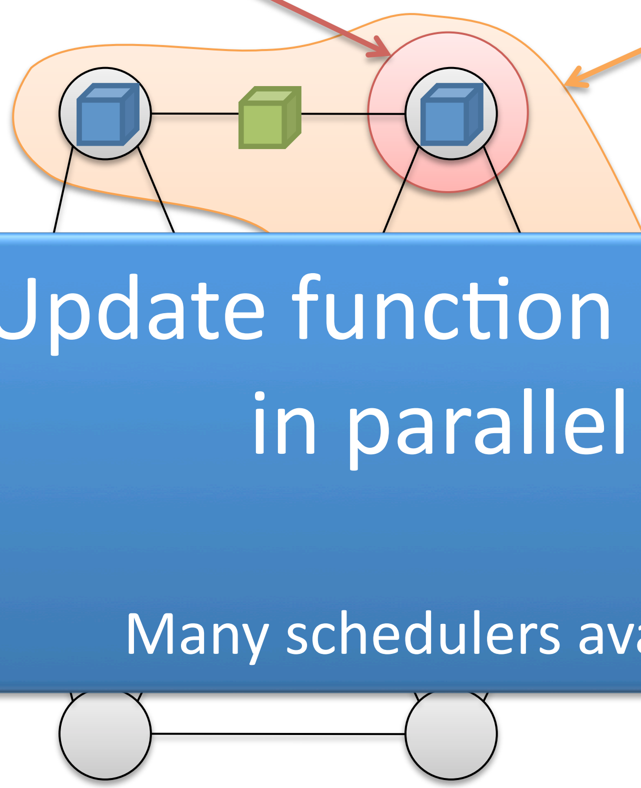
How do we *program* **graph** computation?

“Think like a Vertex.”

-Malewicz et al. [SIGMOD’10]

Update Functions

User-defined program: applied to **vertex** transforms data in **scope** of vertex



```
pagerank(i, scope){
  // Get Neighborhood data
  (R[i], w_ij, R[j]) ←scope;
```

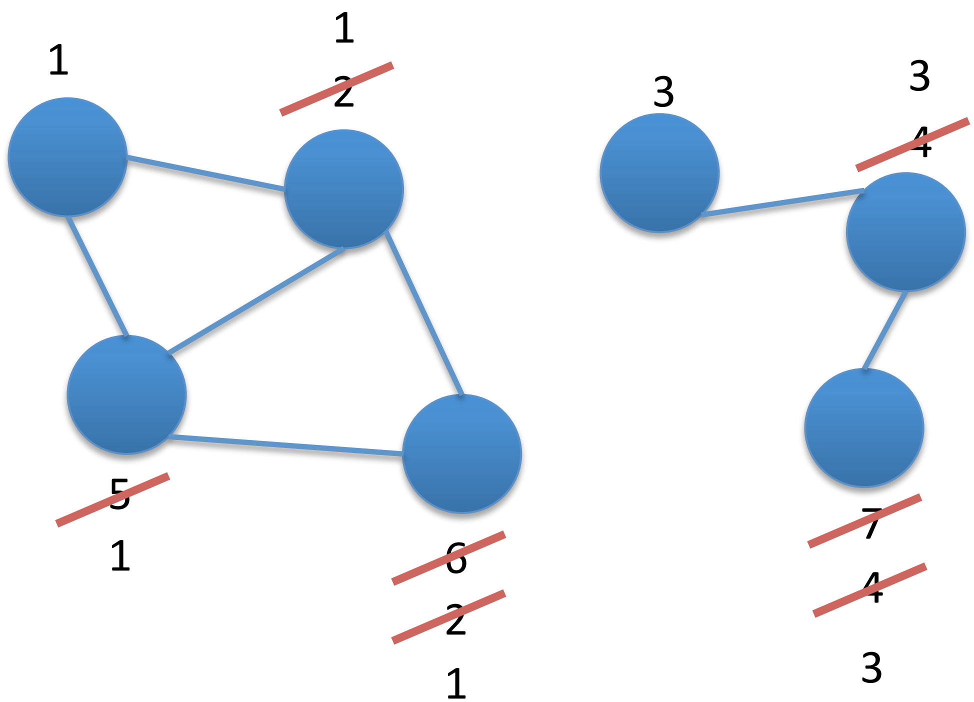
Update function applied (asynchronously) in parallel until convergence

Many schedulers available to prioritize computation

```
}
```

Dynamic computation

Update Function Example: Connected Components



Initialize:

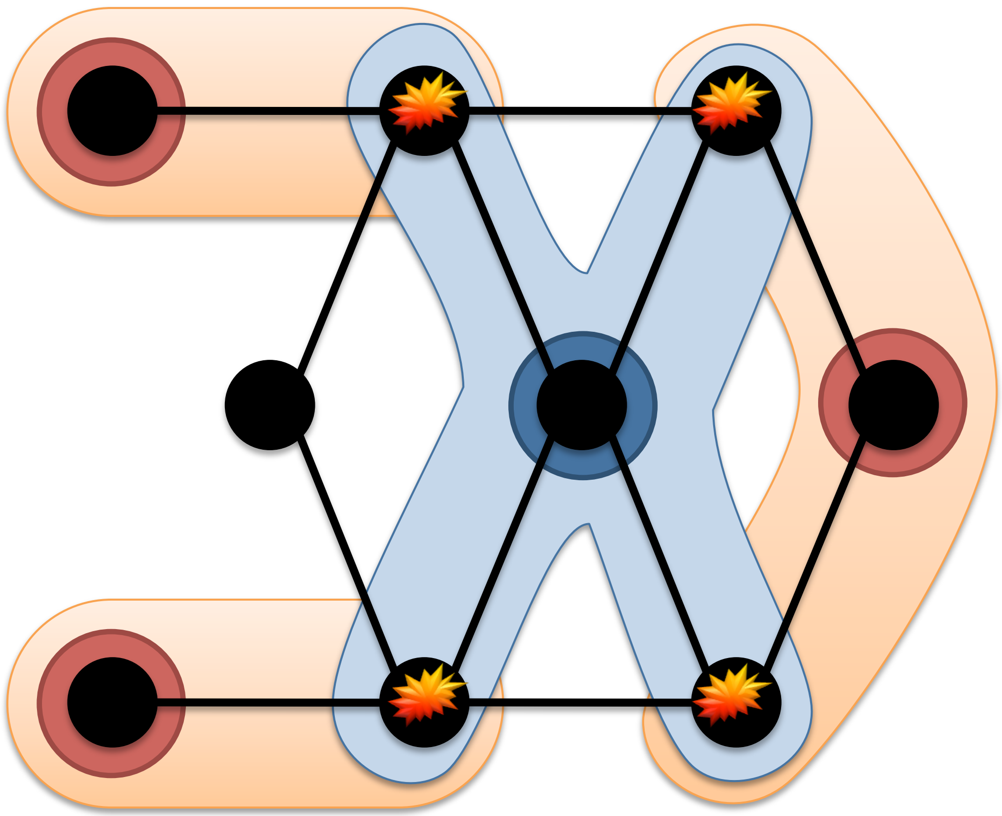
Assign component id to vertex id

Update(v):

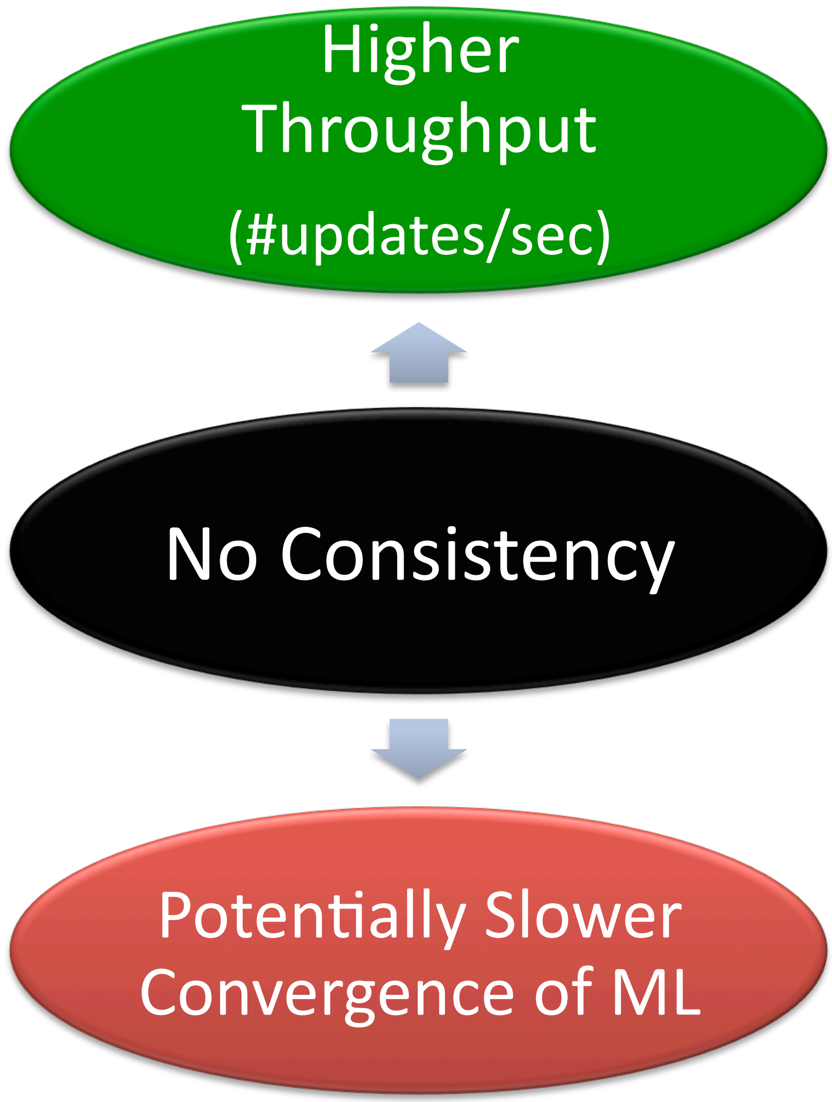
v.component = min(self & neighbor components)

Ensuring Race-Free Code

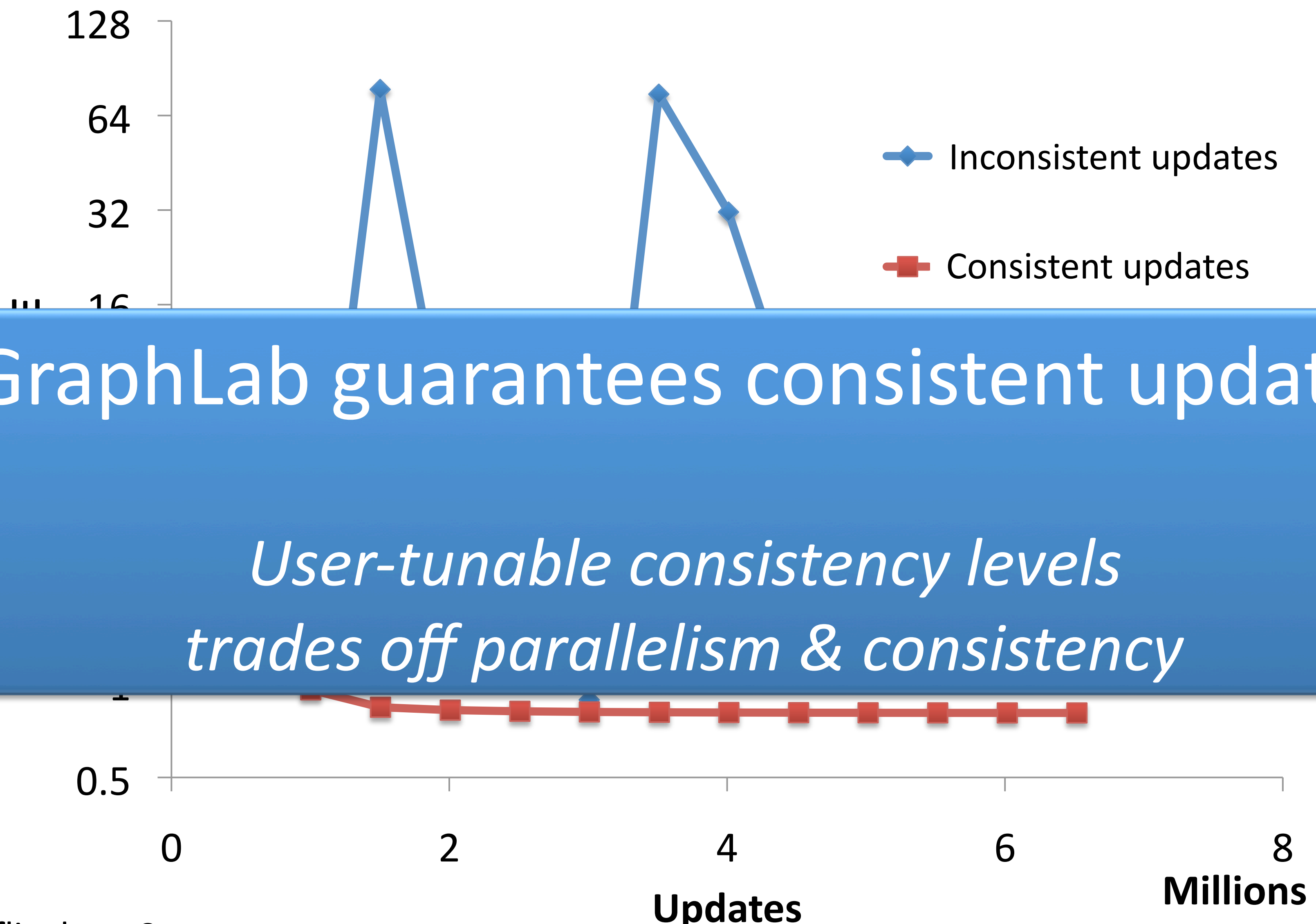
How much can computation **overlap**?



Need for Consistency?



Consistency in Collaborative Filtering



GraphLab guarantees consistent updates

User-tunable consistency levels
trades off parallelism & consistency

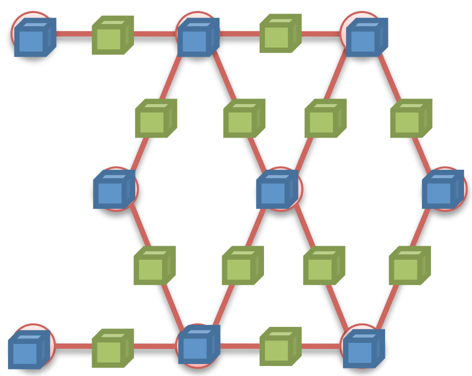
Netflix data, 8 cores

Fix text

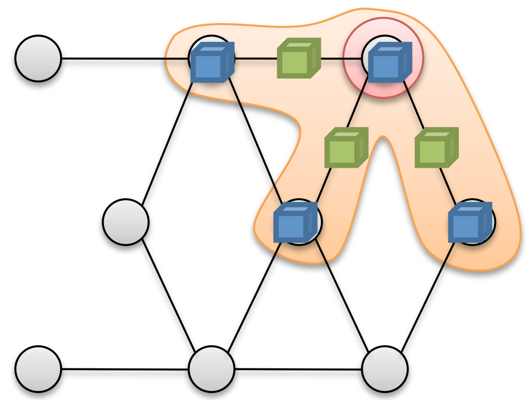
MORE SLIDES ABOUT CONSISTENCY???

The GraphLab Framework

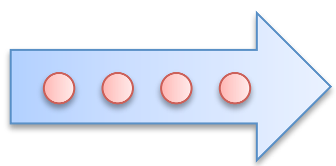
Graph Based
Data Representation



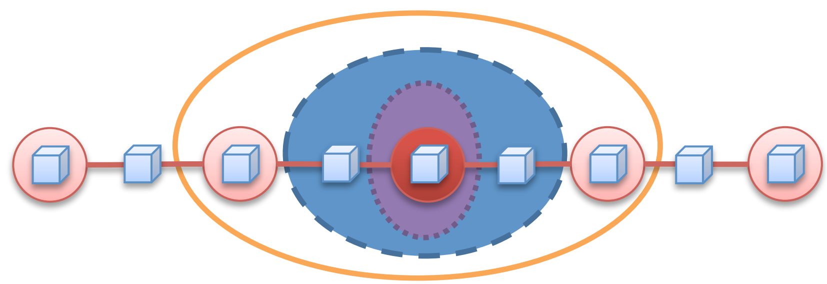
Update Functions
User Computation



Scheduler



Consistency Model



Alternating Least Squares

SVD

Splash Sampler

CoEM

Bayesian Tensor Factorization

Lasso

Belief Propagation

PageRank

LDA

GraphLab
Carnegie Mellon

SVM

Gibbs Sampling

Dynamic Block Gibbs Sampling

K-Means

...Many others...

Matrix Factorization

Linear Solvers

Never Ending Learner Project (CoEM)

Hadoop	95 Cores	7.5 hrs
Distributed GraphLab	**32 EC2 machines**	**80 secs**

0.3% of Hadoop time

2 orders of mag faster → 2 orders of mag cheaper

GraphLab Carnegie Mellon 1

- ML algorithms as vertex programs
- Asynchronous execution and consistency models

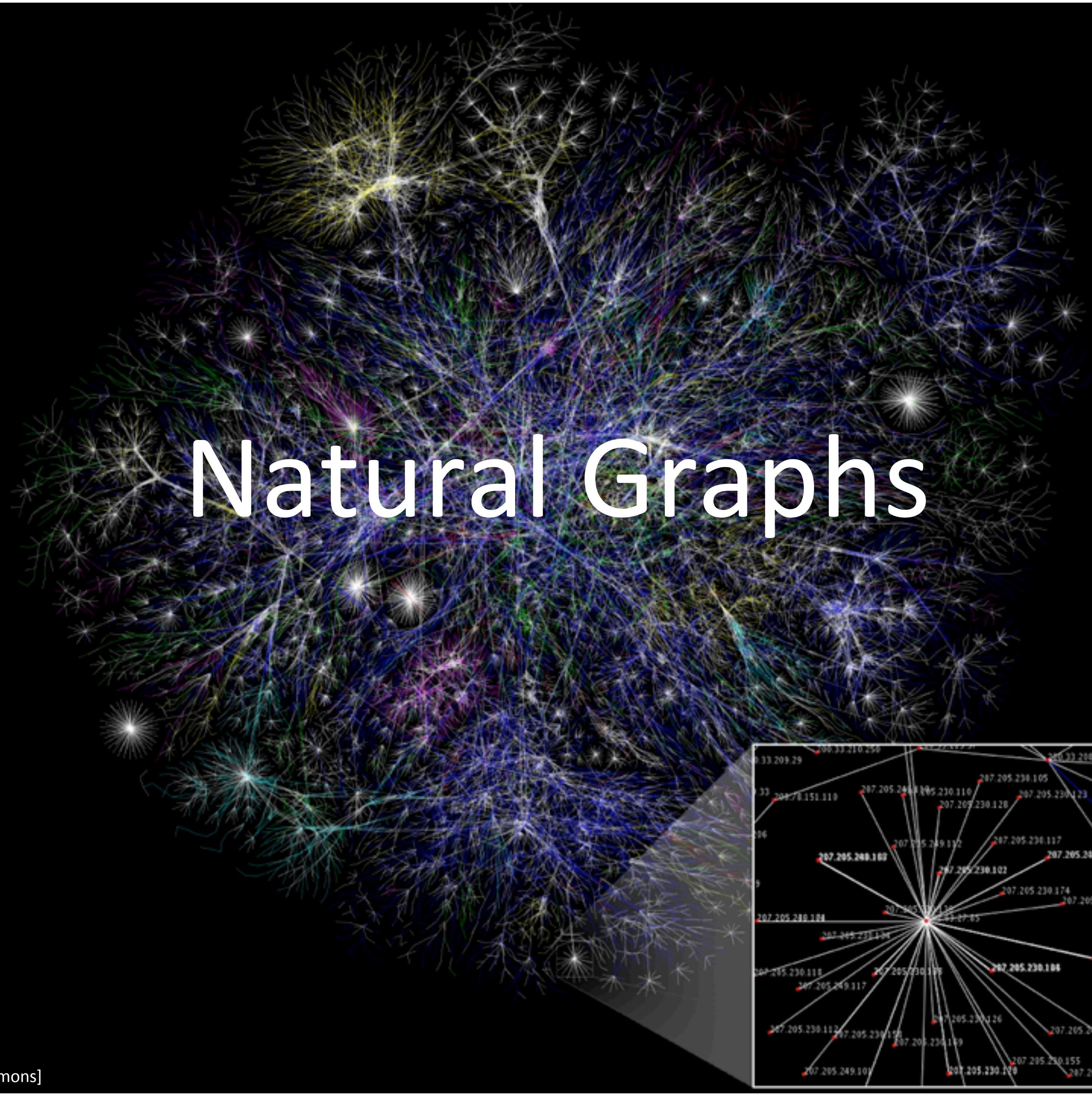
Thus far…

GraphLab 1 provided exciting scaling performance

But…

We couldn’t scale up to Altavista Webgraph 2002 1.4B vertices, 6.7B edges

Natural Graphs



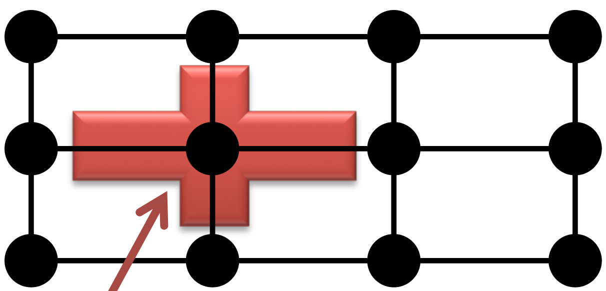
[Image from WikiCommons]

Problem:

Existing ***distributed*** graph computation systems perform poorly on **Natural Graphs**

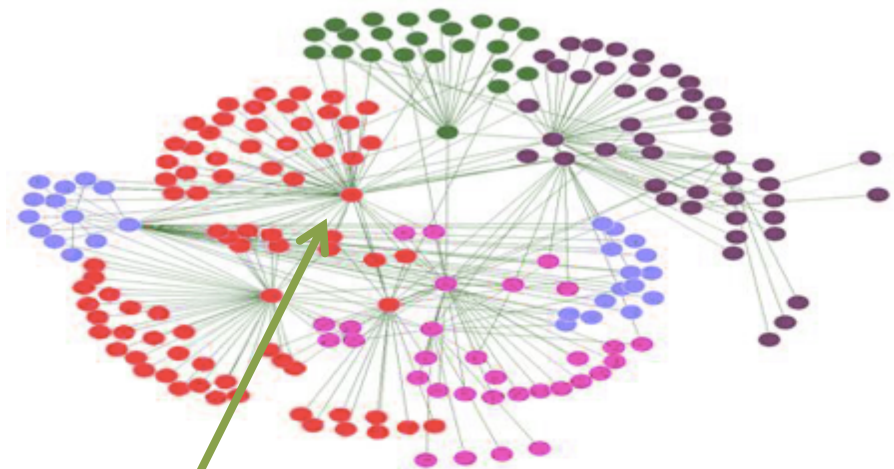
Achilles Heel: Idealized Graph Assumption

Assumed…



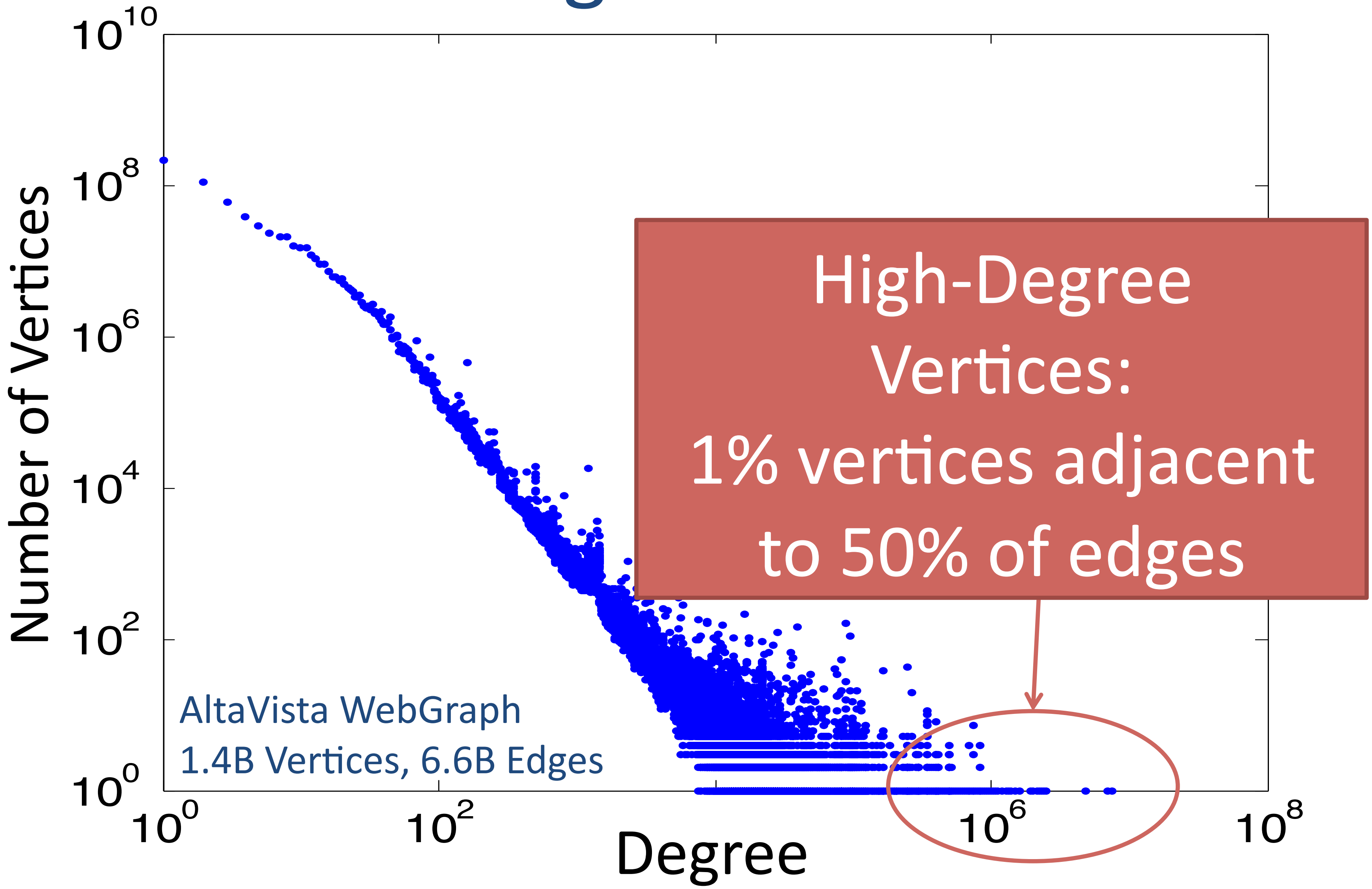
Small degree → Easy to partition

But, Natural Graphs…



Many high degree vertices (power-law degree distribution) → Very hard to partition

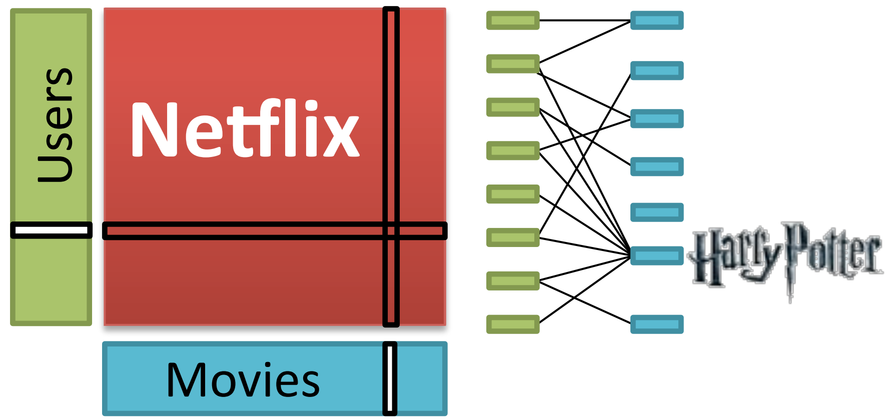
Power-Law Degree Distribution



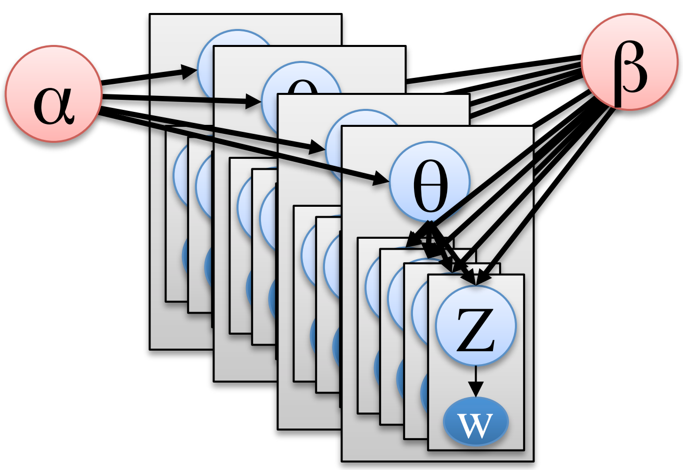
High Degree Vertices are Common

“Social” People

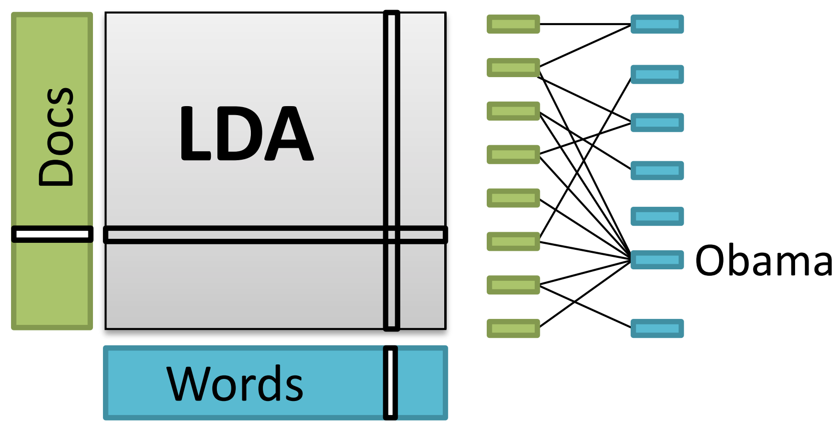
Popular Movies



Hyper Parameters

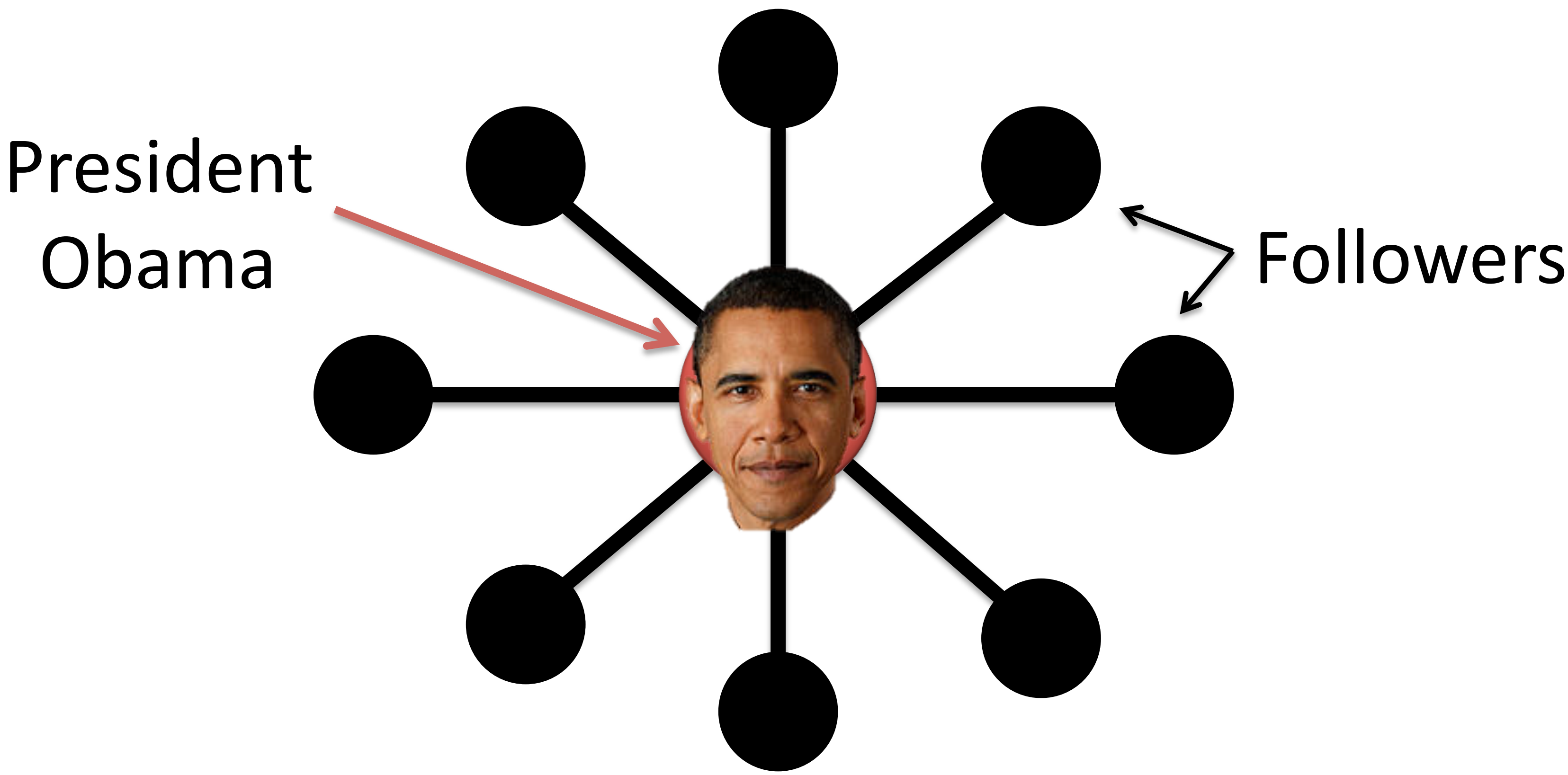


Common Words



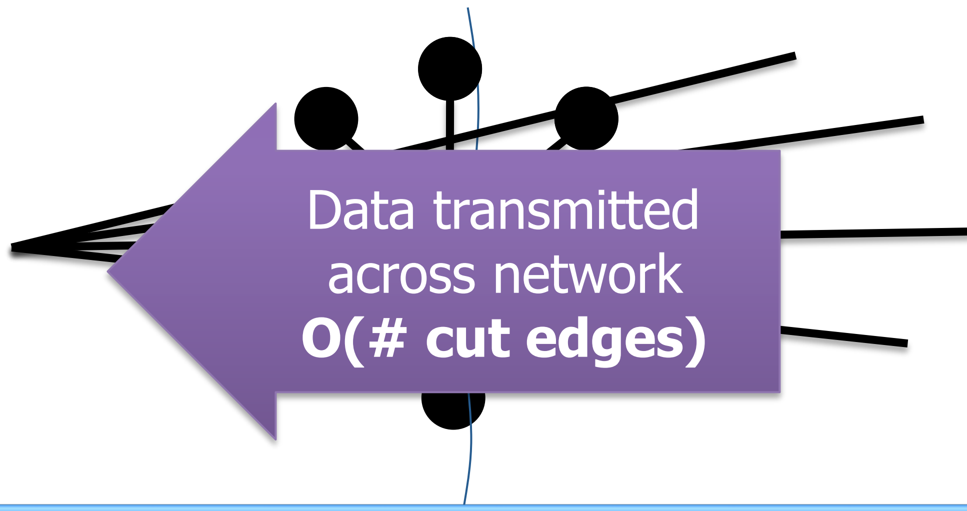
Power-Law Degree Distribution

"Star Like" Motif



Problem:

High Degree Vertices → High Communication for Distributed Updates



Natural graphs do not have low-cost balanced cuts

[Leskovec et al. 08, Lang 04]

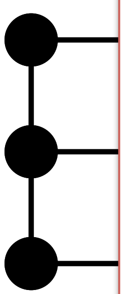
Popular partitioning tools (Metis, Chaco,…) perform poorly

[Abou-Rjeili et al. 06]

Extremely slow and require substantial memory

Random Partitioning

- Both GraphLab 1, Pregel, Twitter, Facebook,… rely on Random (hashed) partitioning for Natural Graphs



For *p* Machines:

$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|E|}\right] = 1 - \frac{1}{p}$$

10 Machines → 90% of edges cut

100 Machines → 99% of edges cut!

All data is communicated… Little advantage over MapReduce

In Summary

GraphLab 1 and Pregel are not well suited for natural graphs

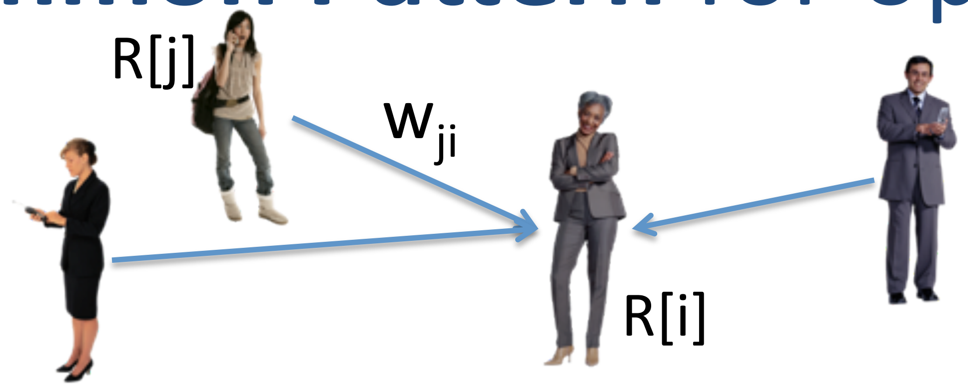
- Poor performance on high-degree vertices
- Low Quality Partitioning

GraphLab2

PowerGraph

SCALABILITY

Common Pattern for Update Fncs.



GraphLab_PageRank(i)

```
// Compute sum over neighbors
total = 0
foreach( j in in_neighbors(i)):
  total = total + R[j] * w_ji
```

Gather **Information About Neighborhood**

```
// Update the PageRank
R[i] = 0.1 + total
```

Apply **Update to Vertex**

```
// Trigger neighbors to run again
if R[i] not converged then
  foreach( j in out_neighbors(i))
    signal vertex-program on j
```

Scatter **Signal to Neighbors & Modify Edge Data**

GAS Decomposition

Gather (Reduce)

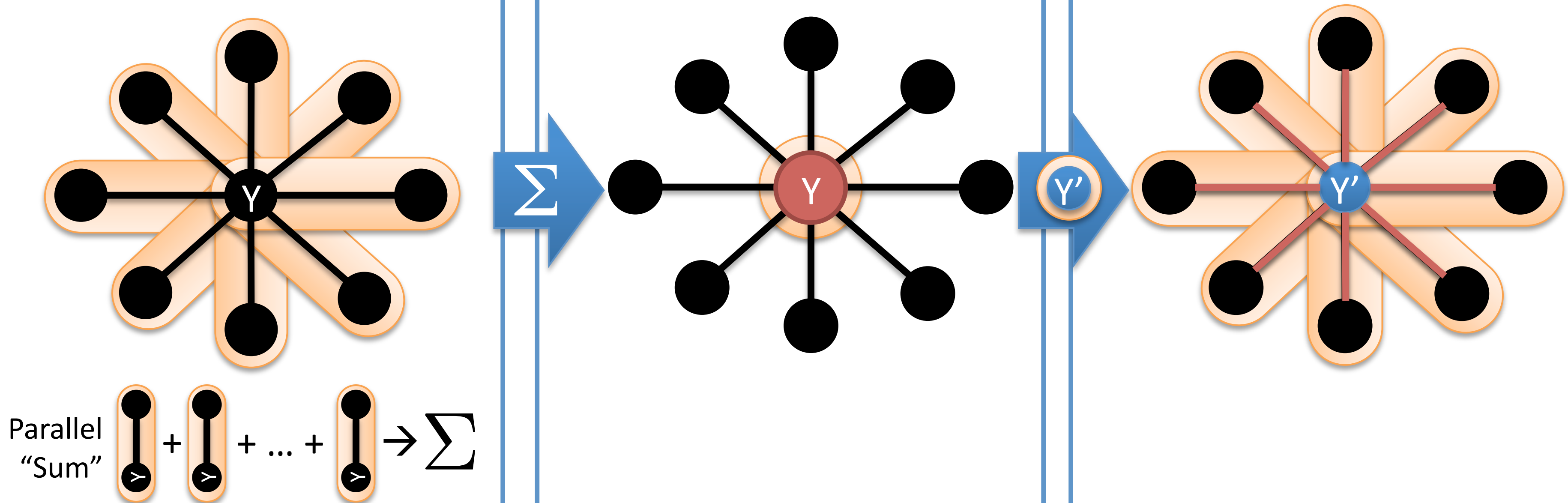
Accumulate information about neighborhood

Apply

Apply the accumulated value to center vertex

Scatter

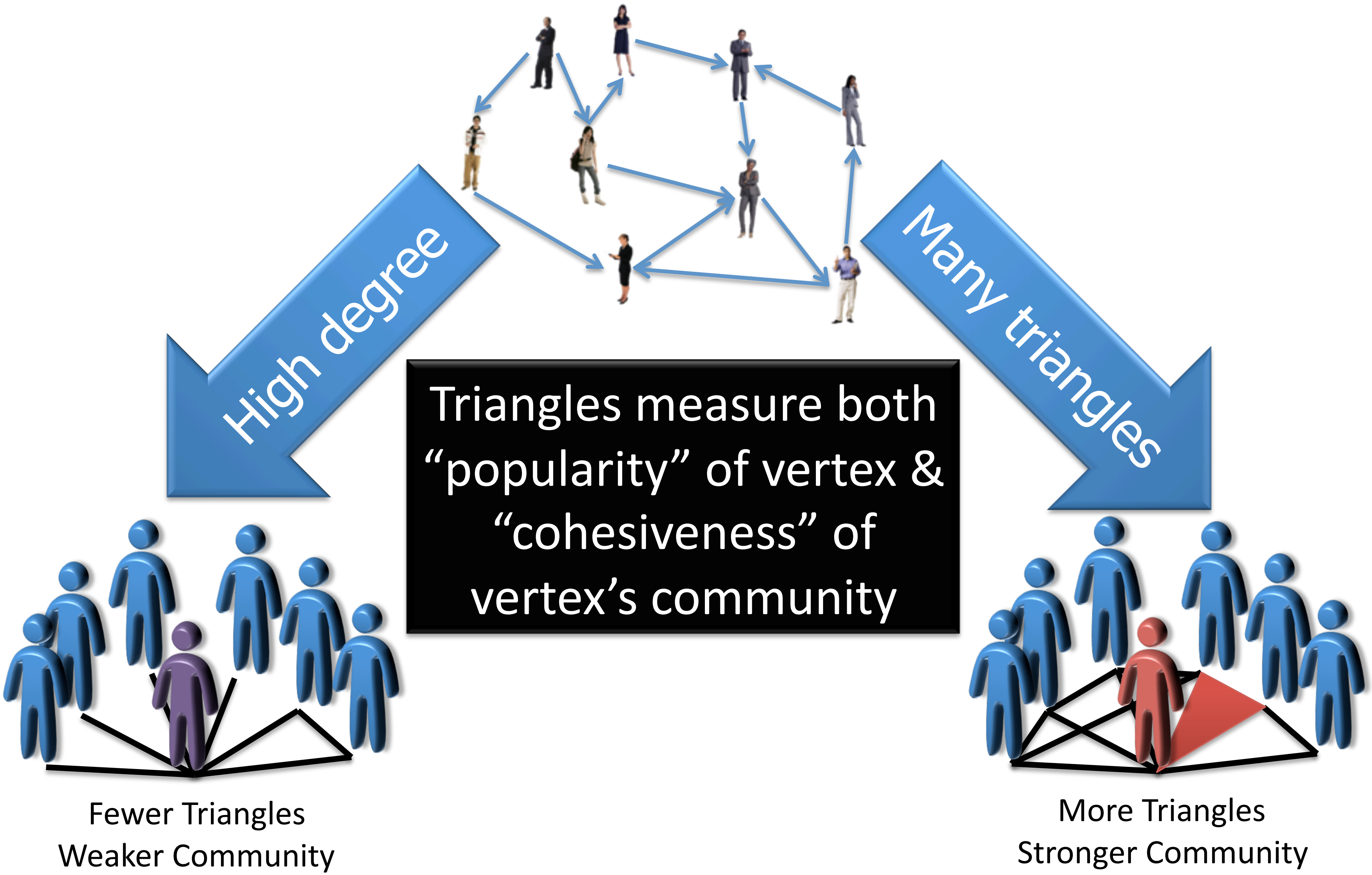
Update adjacent edges and vertices.



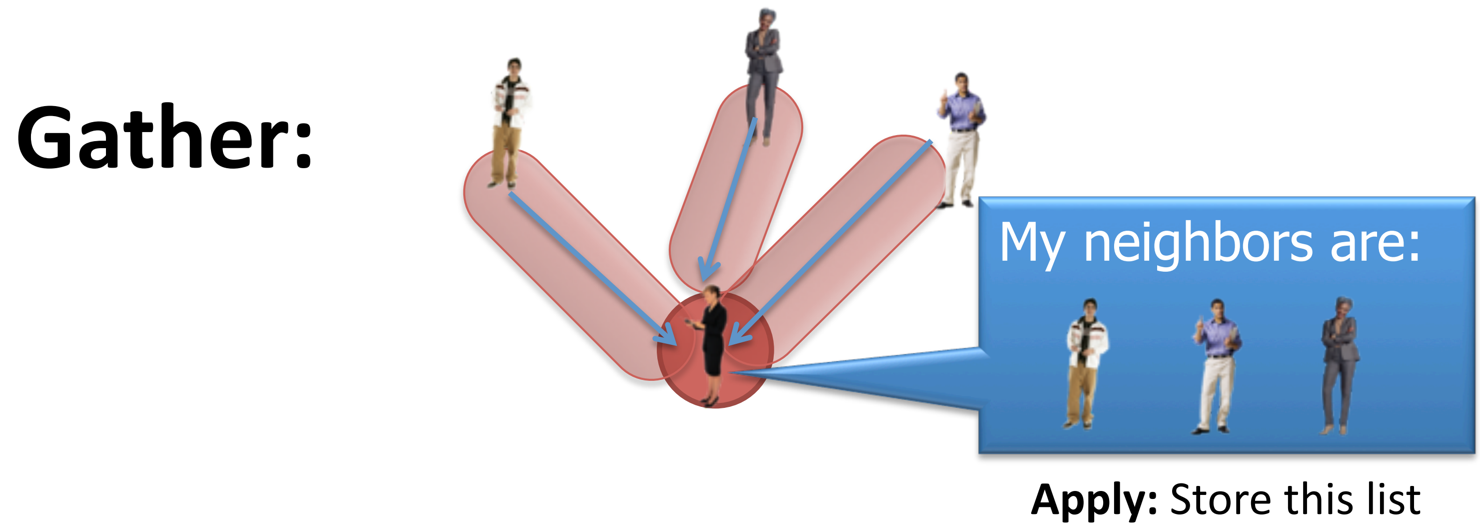
Many ML Algorithms fit into GAS Model

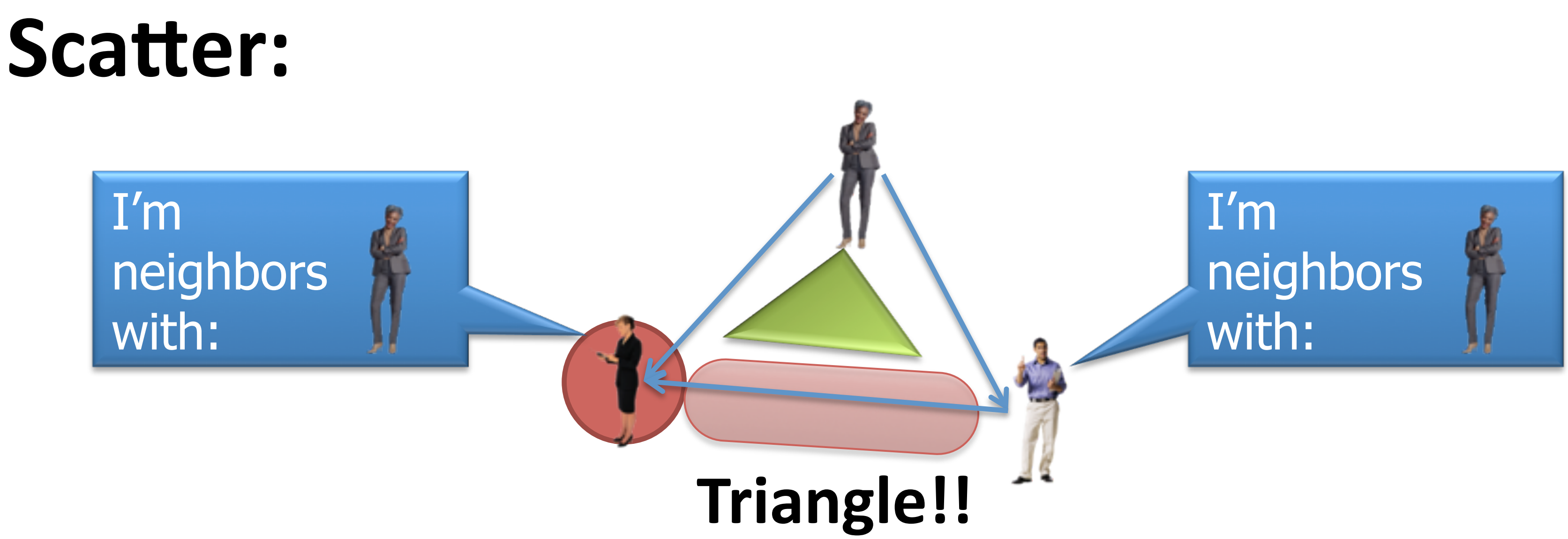
graph analytics, inference in graphical models, matrix factorization, collaborative filtering, clustering, LDA, …

Discovering *Influencers* in Social Networks



Gather/Apply/Scatter Triangle Counting





Triangle Counting on Twitter (2010)

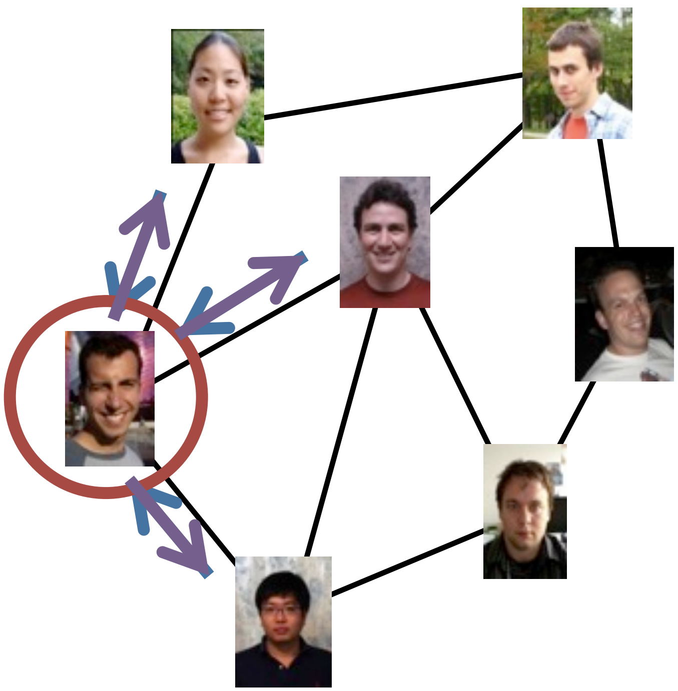
Popular People

Degree	
Britney Spears	3081108
ashton kutcher	2997653
Ellen DeGeneres	2679666
Barack Obama	2653045
CNN Breaking News	2450768
Oprah Winfrey	1994945
Twitter	1959765
Ryan Seacrest	1885917
SHAQ	1844123

Triangles / Following	
Women's Wear Daily	6449985
wefollow	5.962666e+06
Stephen Colbert	5607368
Jonas Brothers	5272692
Rev Run	4483823
Defamer.com	3564740
You Look Great	3207562
Oprah Winfrey	2.936561e+06
Al Gore	2.488950e+06
CNN Breaking News	2.474015e+06

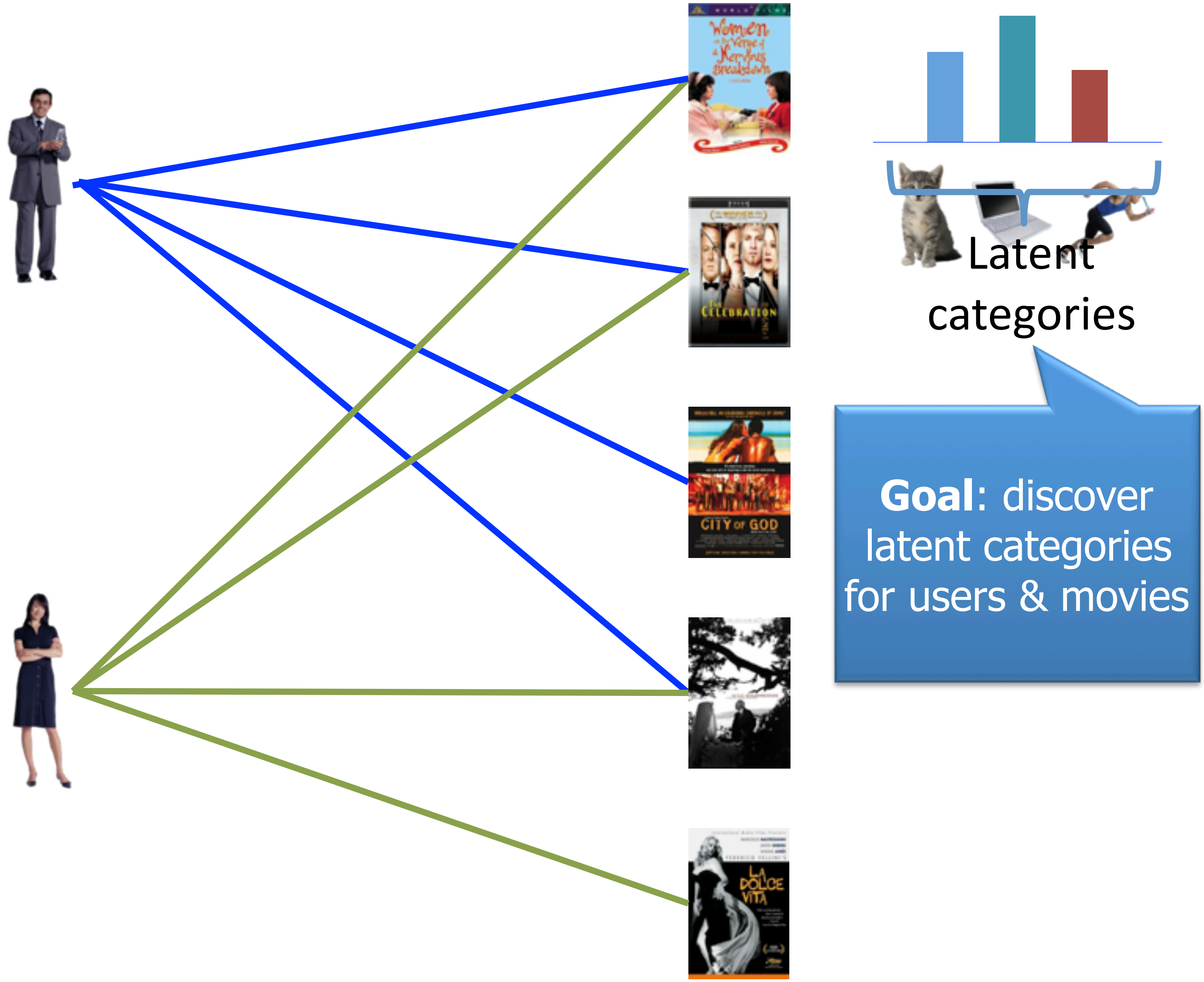
Popular People With Strong Communities

Factorized Belief Propagation

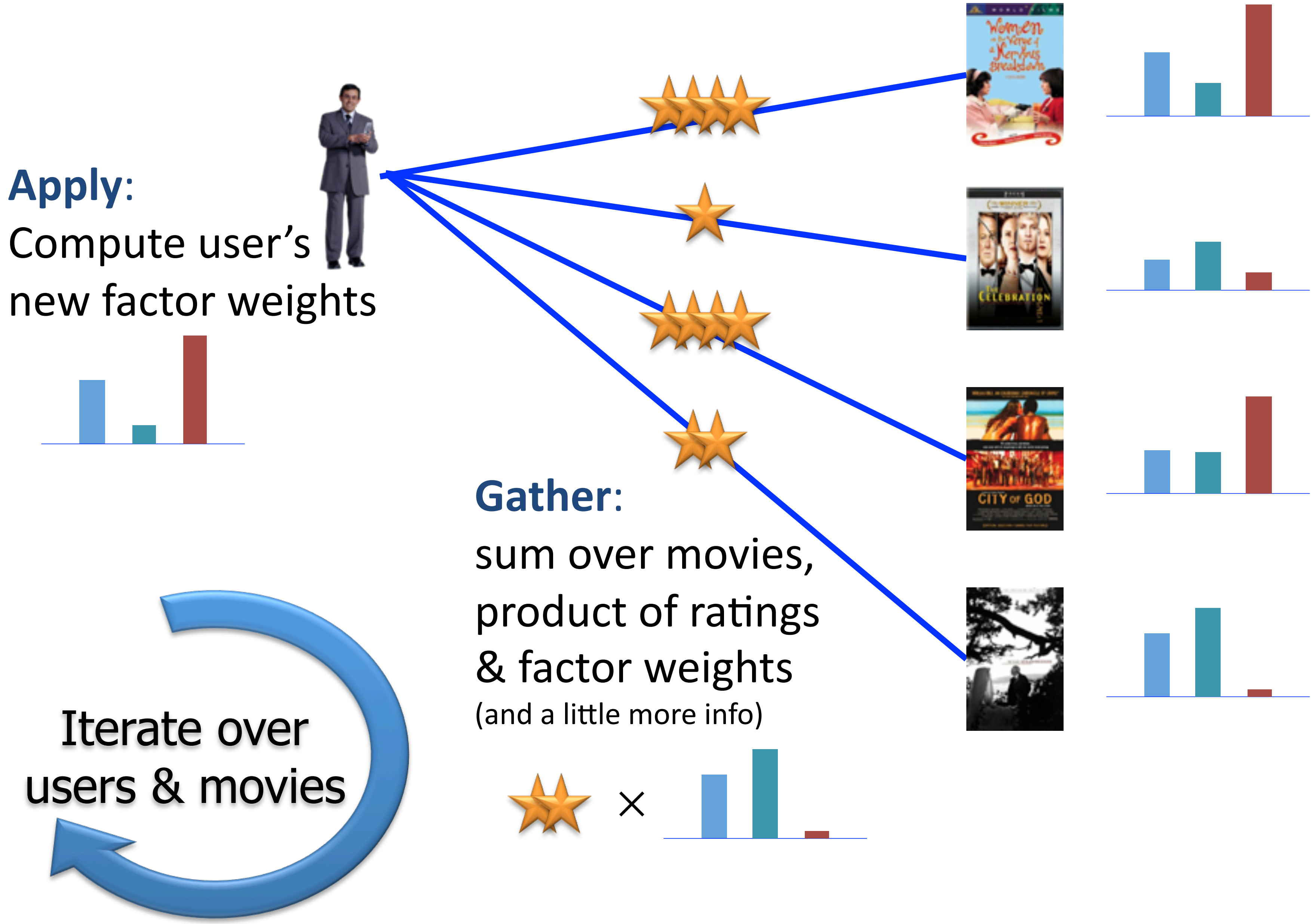


- **Gather**: Accumulates product of *in messages*
- **Apply**: Updates central belief
- **Scatter**: Computes out messages & schedules neighbors as needed

Collaborative Filtering (via Alternating Least Squares)



Factorized Collaborative Filtering Updates

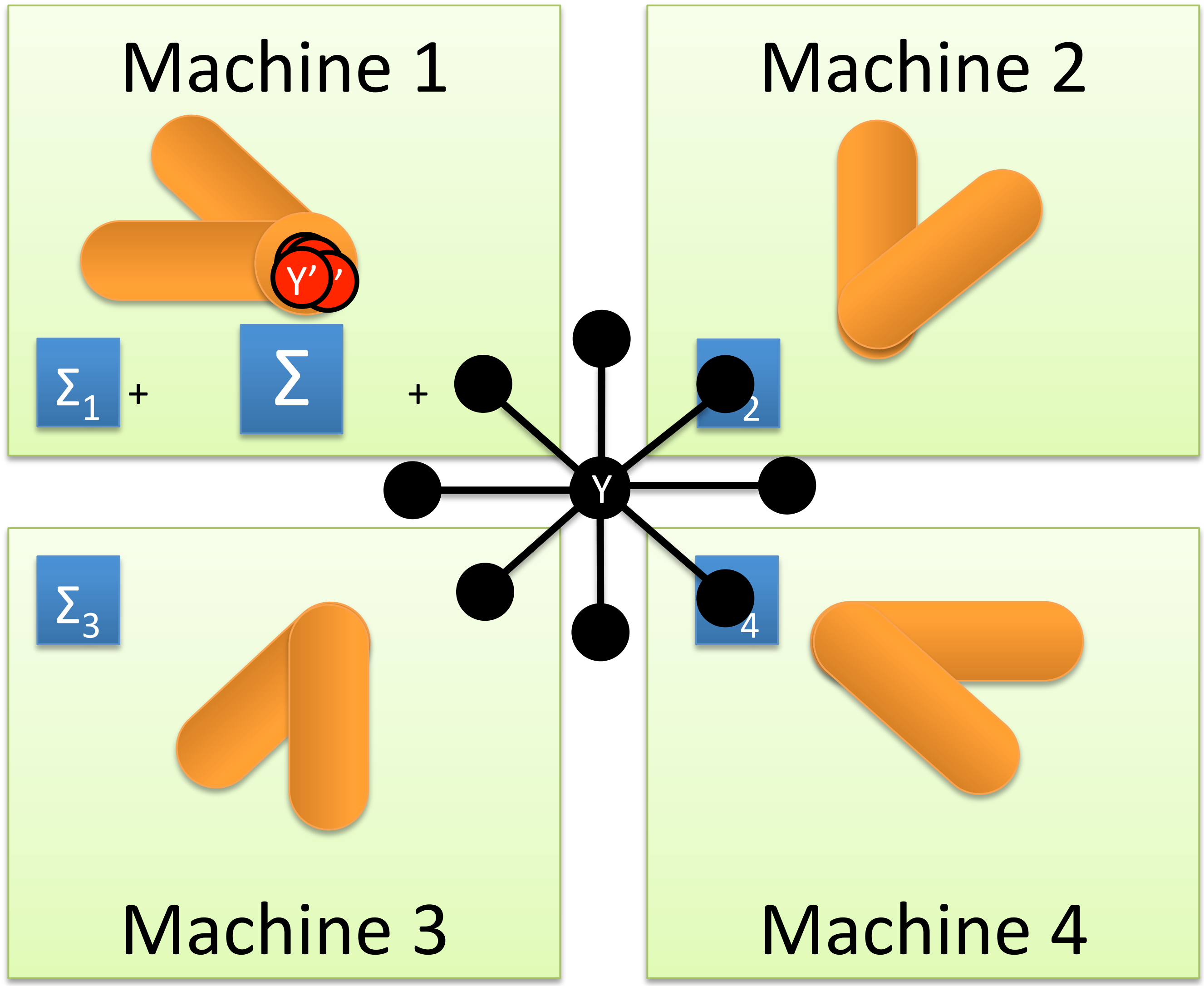


Distributed Execution of a GL2 PowerGraph Vertex-Program

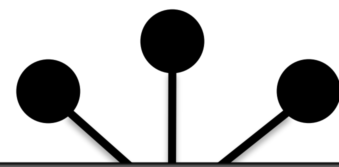
Gather

Apply

Scatter



Minimizing Communication in GL2 PowerGraph: **Vertex Cuts**



Communication linear
in # spanned machines

GL2 PowerGraph includes novel vertex cut algorithms

↓

Provides order of magnitude gains in performance

machines per vertex

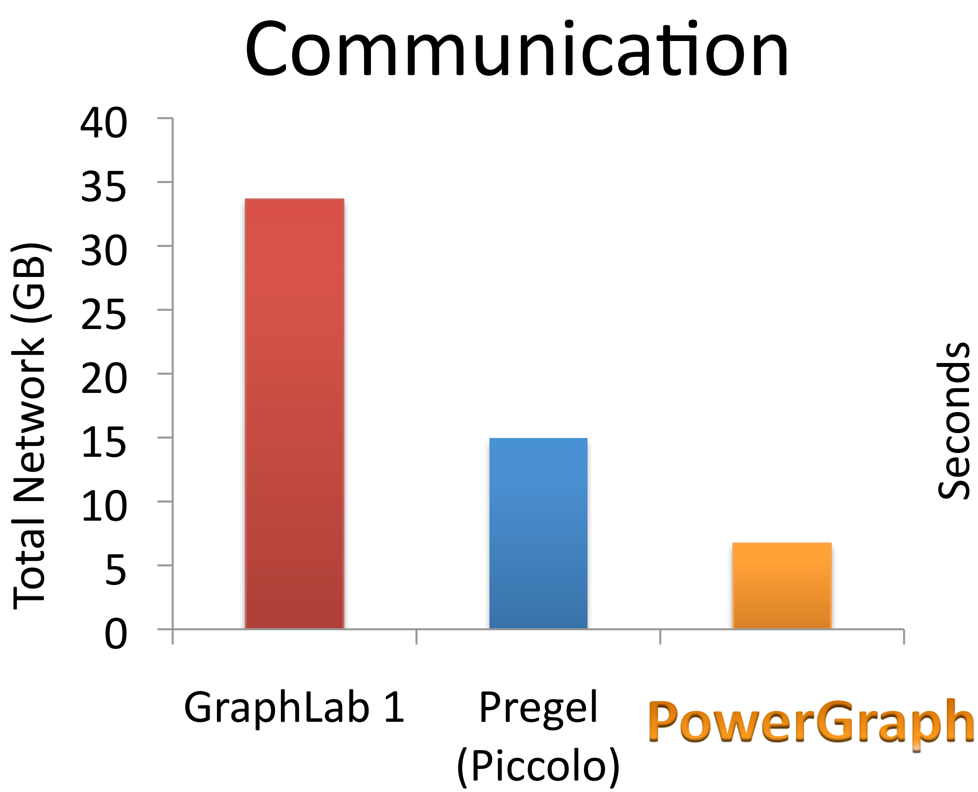
Percolation theory suggests Power Law graphs can be split by removing only a small set of vertices [Albert et al. 2000]

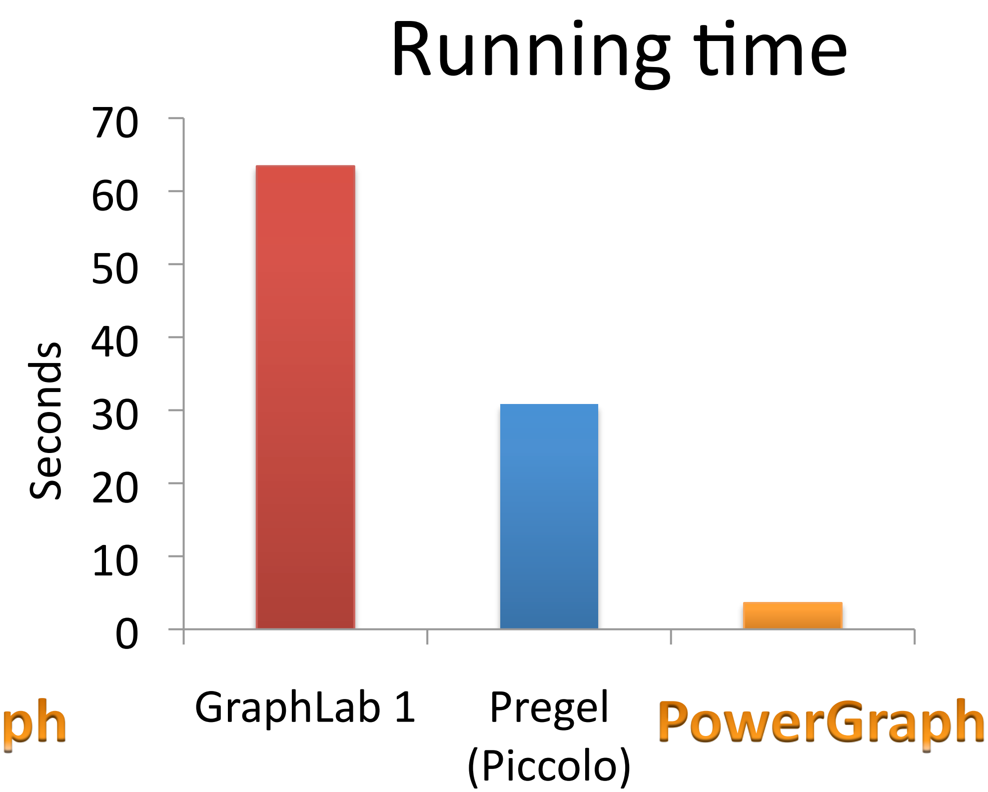
➔

Small vertex cuts possible!

PageRank on the Twitter Follower Graph

Natural Graph with 41M Users, 1.4 Billion Links





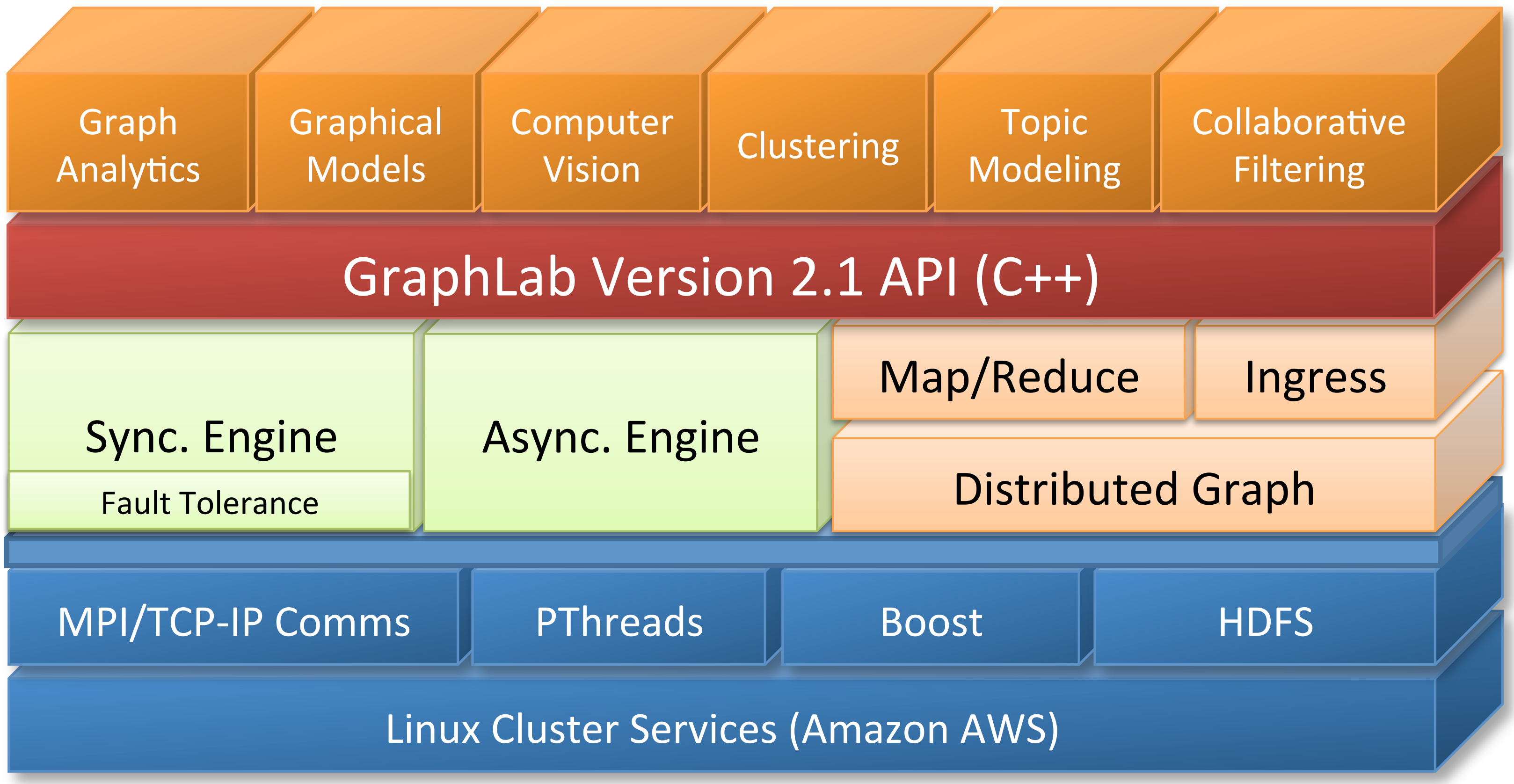
Reduces Communication

Order of Magnitude Faster

32 Nodes x 8 Cores (EC2 HPC cc1.4x)

GraphLab2

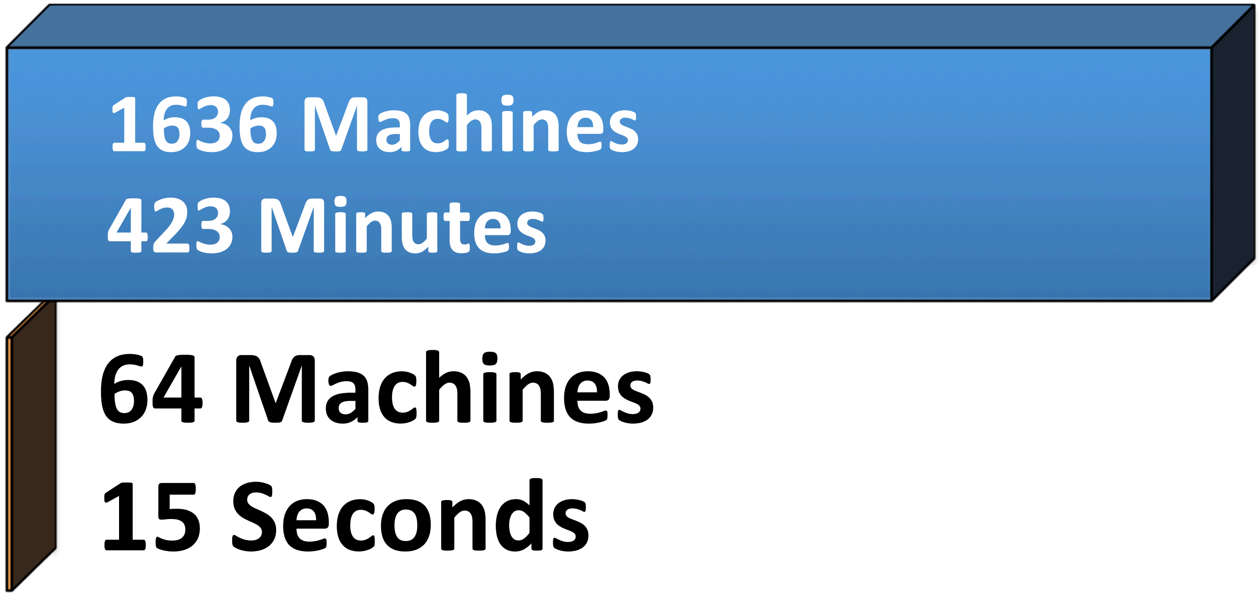
From the Abstraction to a System


Graph Analytics
Graphical Models
Computer Vision
Clustering
Topic Modeling
Collaborative Filtering
GraphLab Version 2.1 API (C++)
Sync. Engine
Fault Tolerance
Async. Engine
Map/Reduce
Ingress
Distributed Graph
MPI/TCP-IP Comms
PThreads
Boost
HDFS
Linux Cluster Services (Amazon AWS)

Triangle Counting on Twitter Graph

34.8 Billion Triangles

Hadoop
[WWW'11]



GL2 PowerGraph

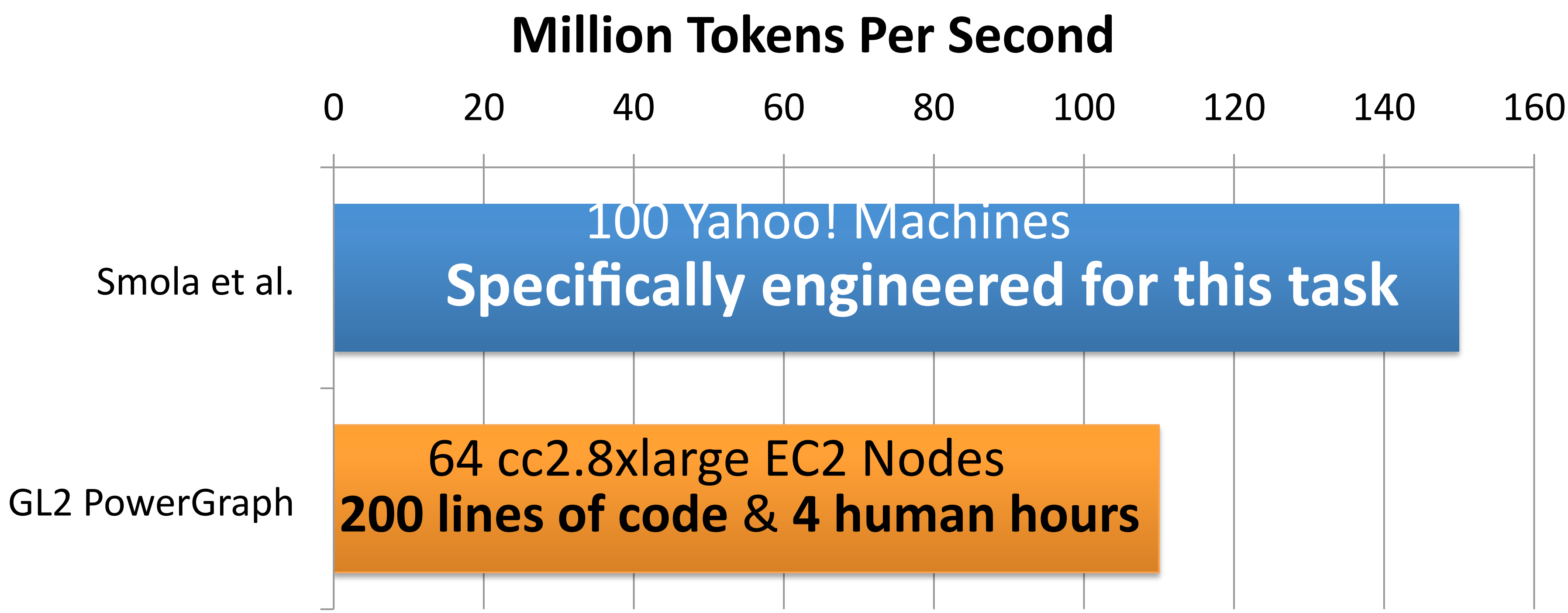
64 Machines
15 Seconds

Why? **Wrong Abstraction** →
Broadcast O(degree^2) messages per Vertex

S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," WWW'11

Topic Modeling (LDA)

- English language Wikipedia
  - 2.6M Documents, 8.3M Words, 500M Tokens
  - Computationally intensive algorithm



How well does GraphLab scale?

Yahoo Altavista Web Graph (2002):

One of the largest publicly available webgraphs

1.4B Webpages, 6.7 Billion Links

7 seconds per iter.

64 HPC Nodes

1B links processed per second

30 lines of user code

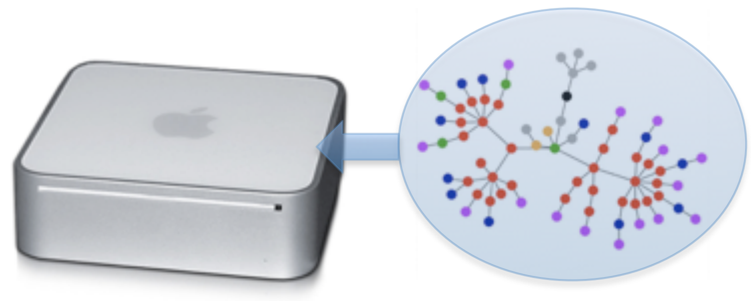
1024 Cores (2048 HT)

4.4 TB RAM

GraphChi: Going small with GraphLab

GraphLab

Solve huge problems on small or embedded devices?

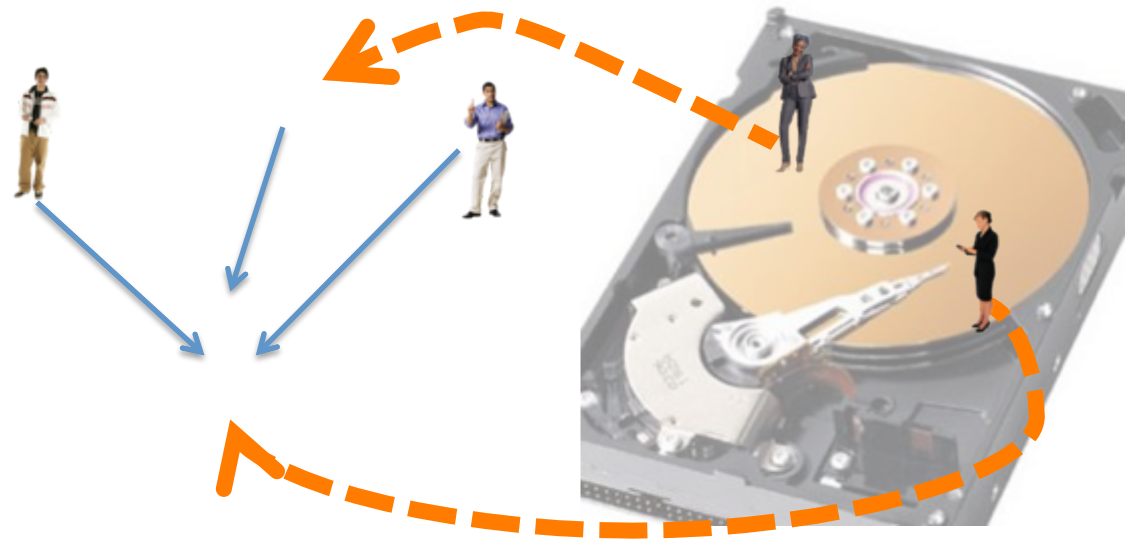


Key: Exploit non-volatile memory (starting with SSDs and HDs)

GraphChi – disk-based GraphLab

Challenge:

Random Accesses



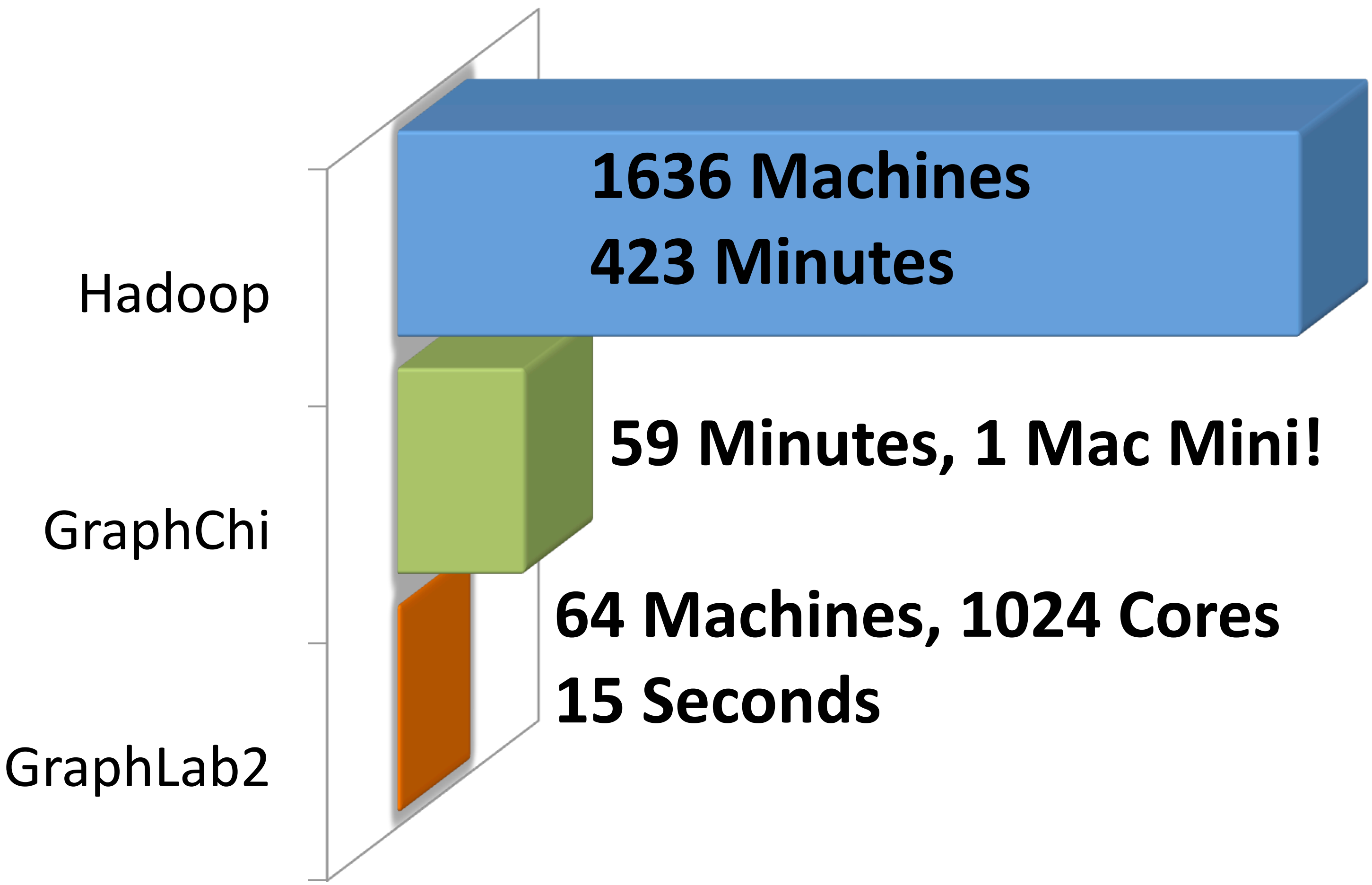
Novel GraphChi solution:

Parallel sliding windows method ➔ *minimizes number of random accesses*

Triangle Counting on Twitter Graph

40M Users
1.2B Edges

Total: 34.8 Billion Triangles



Hadoop results from [Suri & Vassilvitskii '11]

GraphLab 1 (Carnegie Mellon)

- ML algorithms as vertex programs
- Asynchronous execution and consistency models

GraphLab 2 (PowerGraph)

- Natural graphs change the nature of computation
- Vertex cuts and gather/apply/scatter model

GL2 PowerGraph
focused on
Scalability

at the loss of
Usability

GraphLab 1

```
PageRank(i, scope){
  acc = 0
  for (j in InNeighbors) {
    acc += pr[j] * edge[j].weight
  }
  pr[i] = 0.15 + 0.85 * acc
}
```

Explicitly described operations

Code is intuitive

GraphLab 1

```
PageRank(i, scope){
  acc = 0
  for (j in InNeighbors) {
    acc += pr[j] * edge[j].weight
  }
  pr[i] = 0.15 + 0.85 * acc
}
```

Explicitly described operations

Code is intuitive

GL2 PowerGraph

Implicit operation

```
gather(edge) {
  return edge.source.value *
         edge.weight
}
```

```
merge(acc1, acc2) {
       return accum1 + accum2
}
```

Implicit aggregation

```
apply(v, accum) {
  v.pr = 0.15 + 0.85 * acc
}
```

Need to understand engine to understand code

GraphLab 1
Carnegie Mellon

Great flexibility,
but hit scalability wall

GraphLab 2
PowerGraph

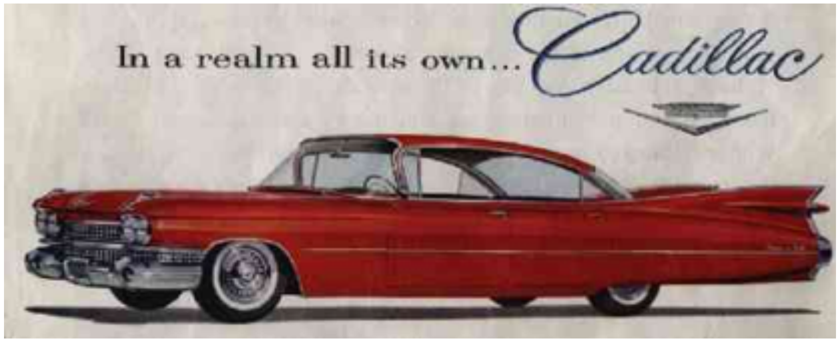
Scalability,
but very rigid abstraction
(many contortions needed to implement
SVD++, Restricted Boltzmann Machines)

What now?

GraphLab 3

WarpGraph

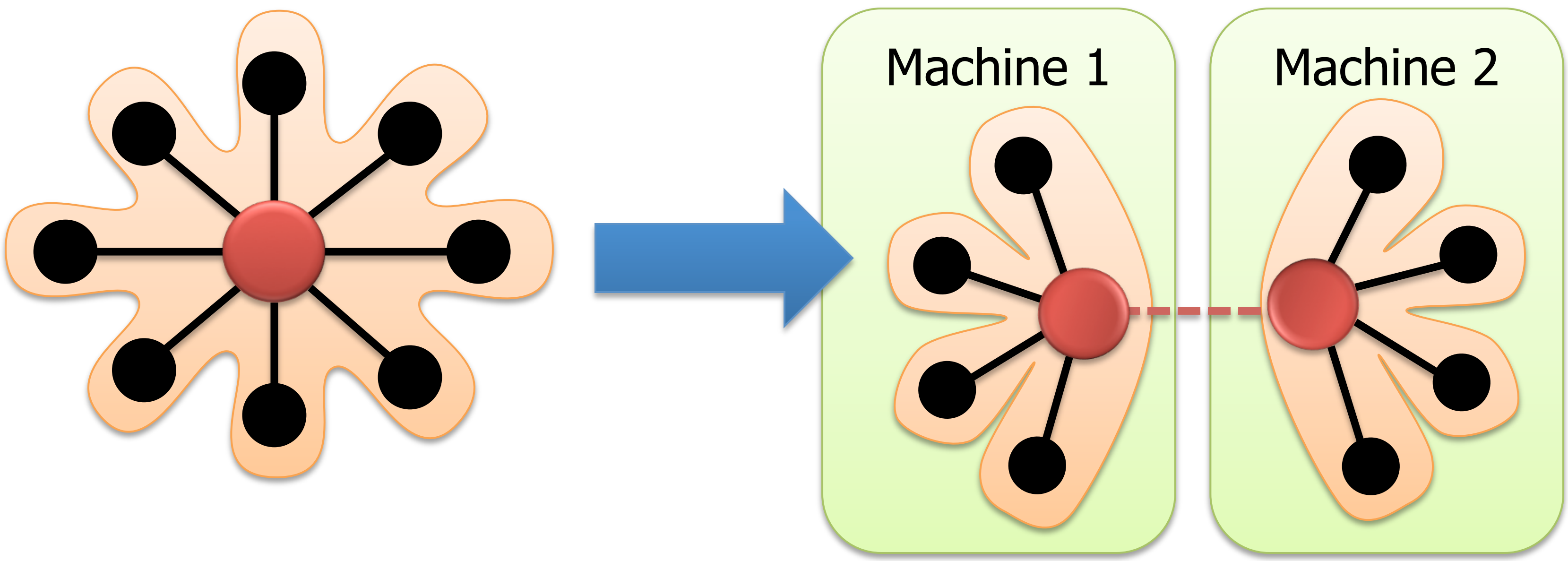
USABILITY



GL3 WarpGraph Goals

Program Like GraphLab 1

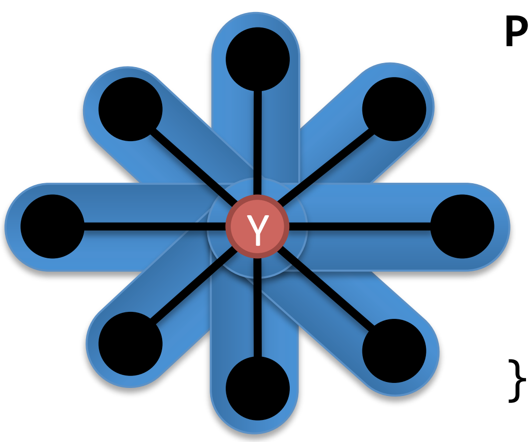
Run Like GraphLab 2



Fine-Grained Primitives

Expose Neighborhood Operations through Parallelizable Iterators

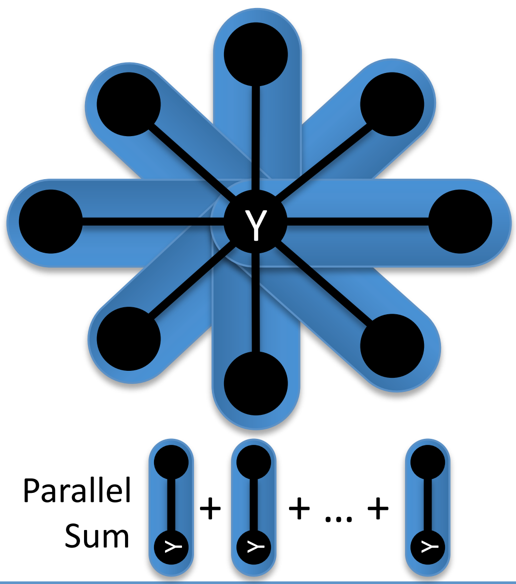
$$R[i] = 0.15 + 0.85 \sum_{(j,i) \in E} \mathrm{w}[j,i] * R[j]$$



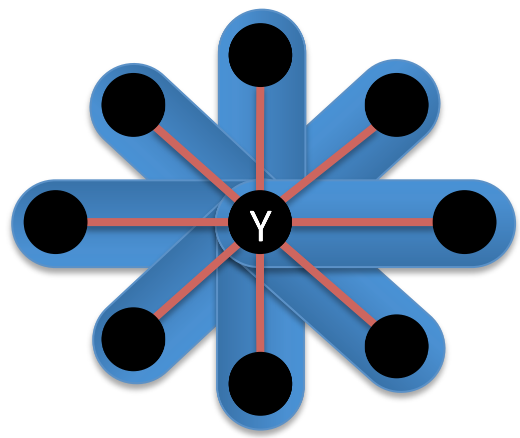

```
PageRankUpdateFunction(Y) {
    Y.pagerank = 0.15 + 0.85 *



}
```

Expressive, Extensible Neighborhood API

MapReduce over Neighbors

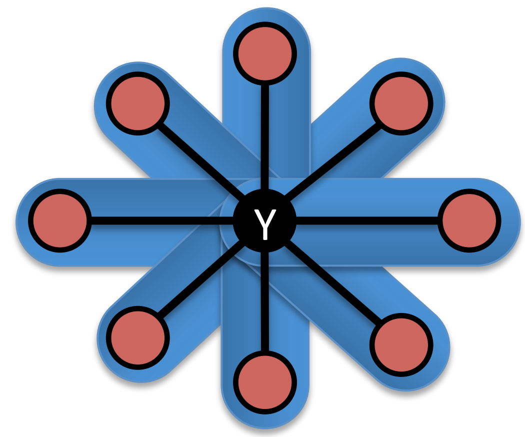


Parallel Transform Adjacent Edges



Modify adjacent edges

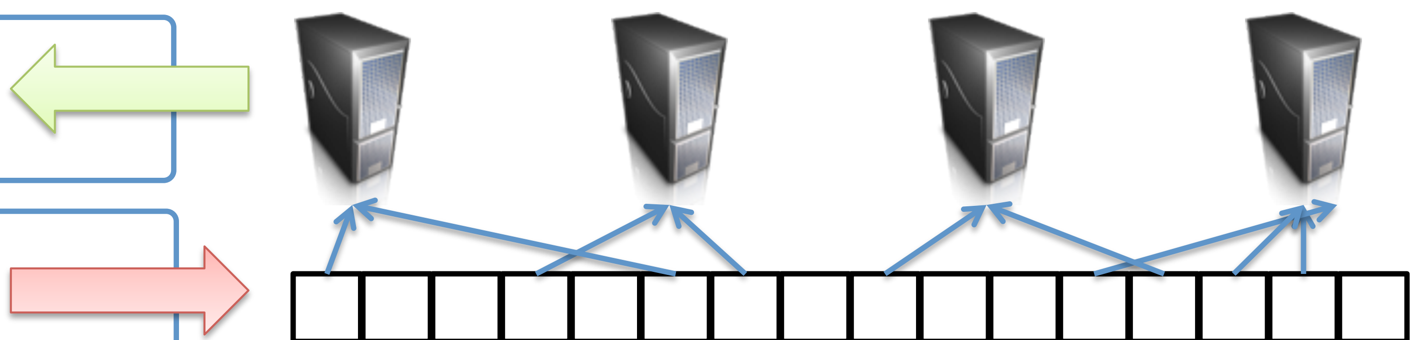
Broadcast



Schedule a selected subset of adjacent vertices

DHT Get Keys

DHT Update Keys



Can express every GL2 PowerGraph program (more easily) in GL3 WarpGraph

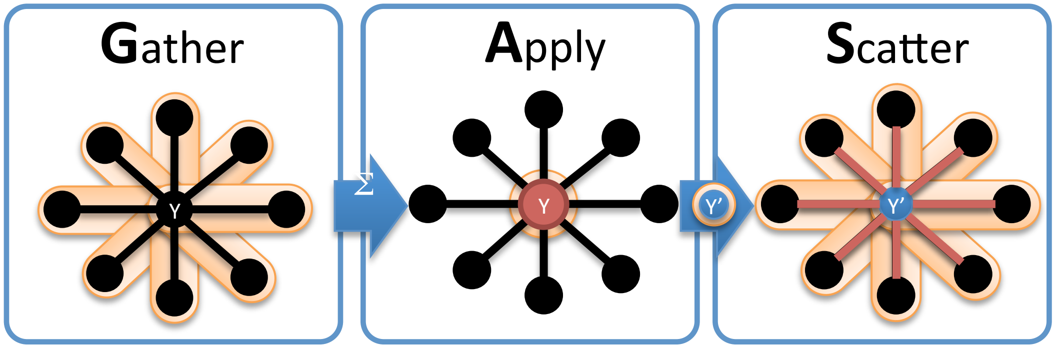
But GL3 is more expressive

```
UpdateFunction(v) {
  if (v.data == 1)
    accum = MapReduceNeighs(g,m)
  else ...
}
```

- Multiple gathers
- Scatter before gather
- Conditional execution

GL2 PowerGraph:

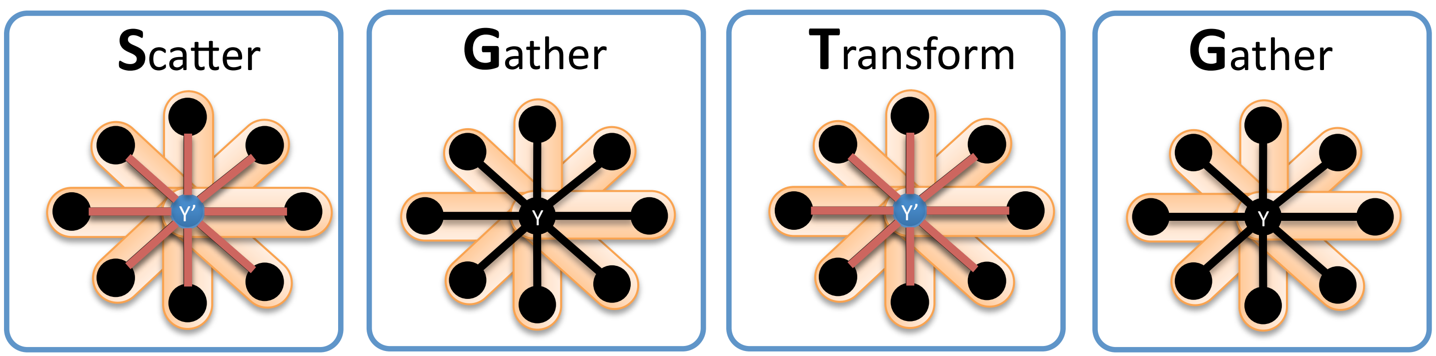
Fast because **communication** phases are very **predictable**



… repeat

GL3 WarpGraph:

Communication highly **unpredictable**



…

Risk: **High Latency**

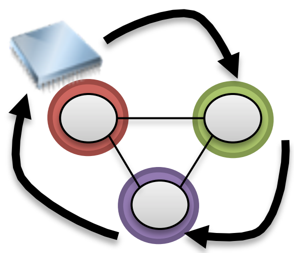
(spend all our time waiting for a reply…)

Hide Latency

Do Something Else while Waiting

Create 1000s of threads, each running an update function on a different vertex

Performance Bottleneck: Context Switching



Every cycle used in context switching is wasted

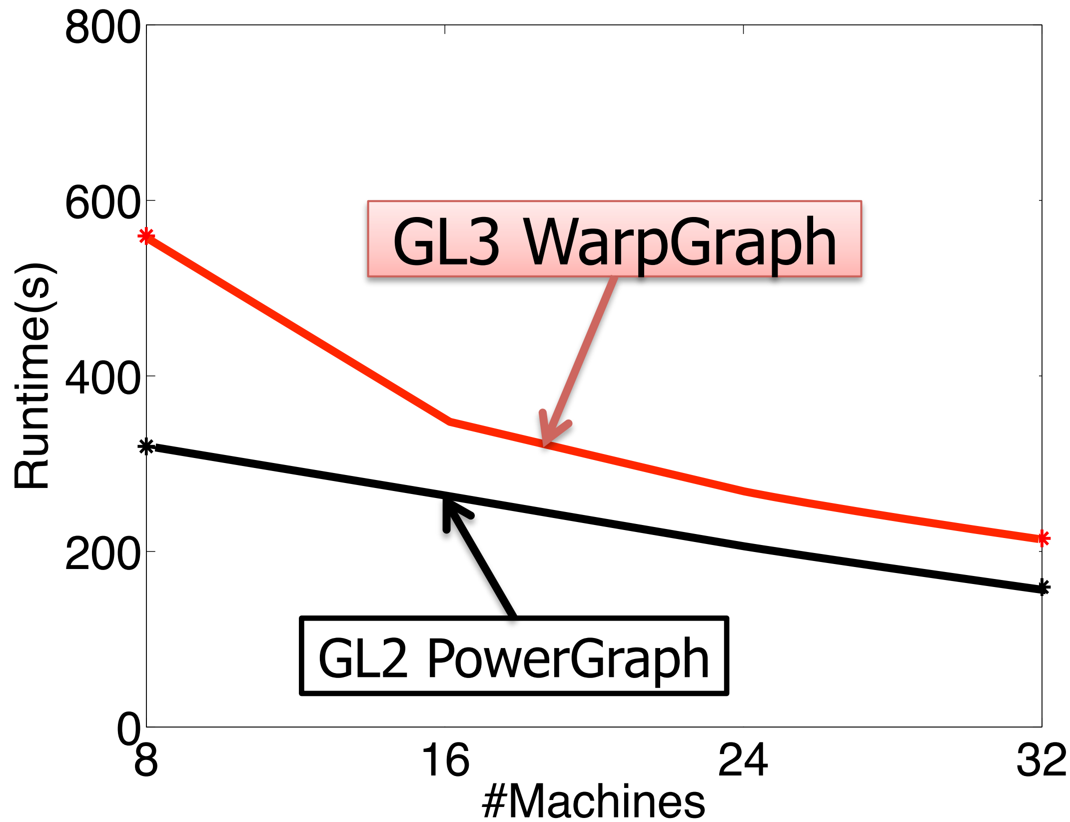
(OS context switch is slow requiring 10K-100k cycles)

GL3 WarpGraph: Novel user-mode threading

8M context switches per second

100x faster than OS

PageRank Twitter Graph: 41M Vertices 1.4B Edges

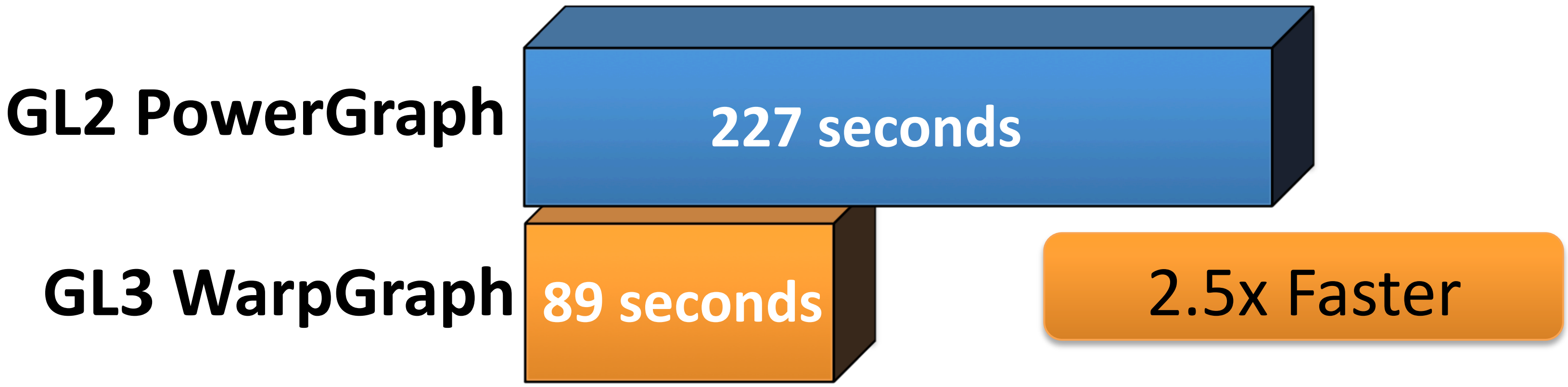


WarpGraph only 25% slower, with much improved programmability
But, here, asynchrony not fundamental for performance

32 Nodes x 16 Cores (EC2 HPC cc2.8x)

Graph Coloring

Twitter Graph: 41M Vertices 1.4B Edges



Asynchrony fundamental here →
WarpGraph outperforms PowerGraph with simpler code

32 Nodes x 16 Cores (EC2 HPC cc2.8x)

Usability

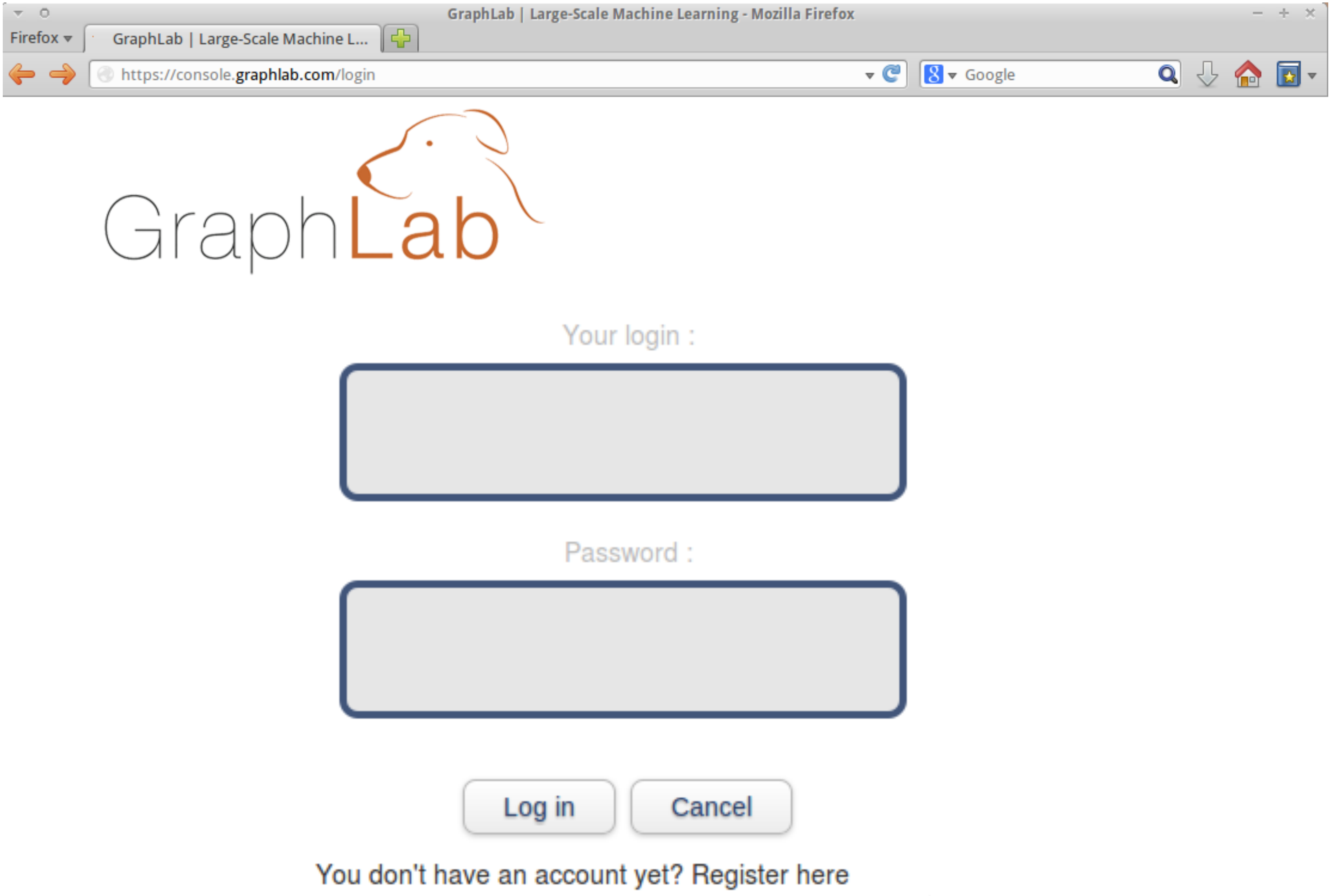
Consensus that WarpGraph is much easier to use than PowerGraph

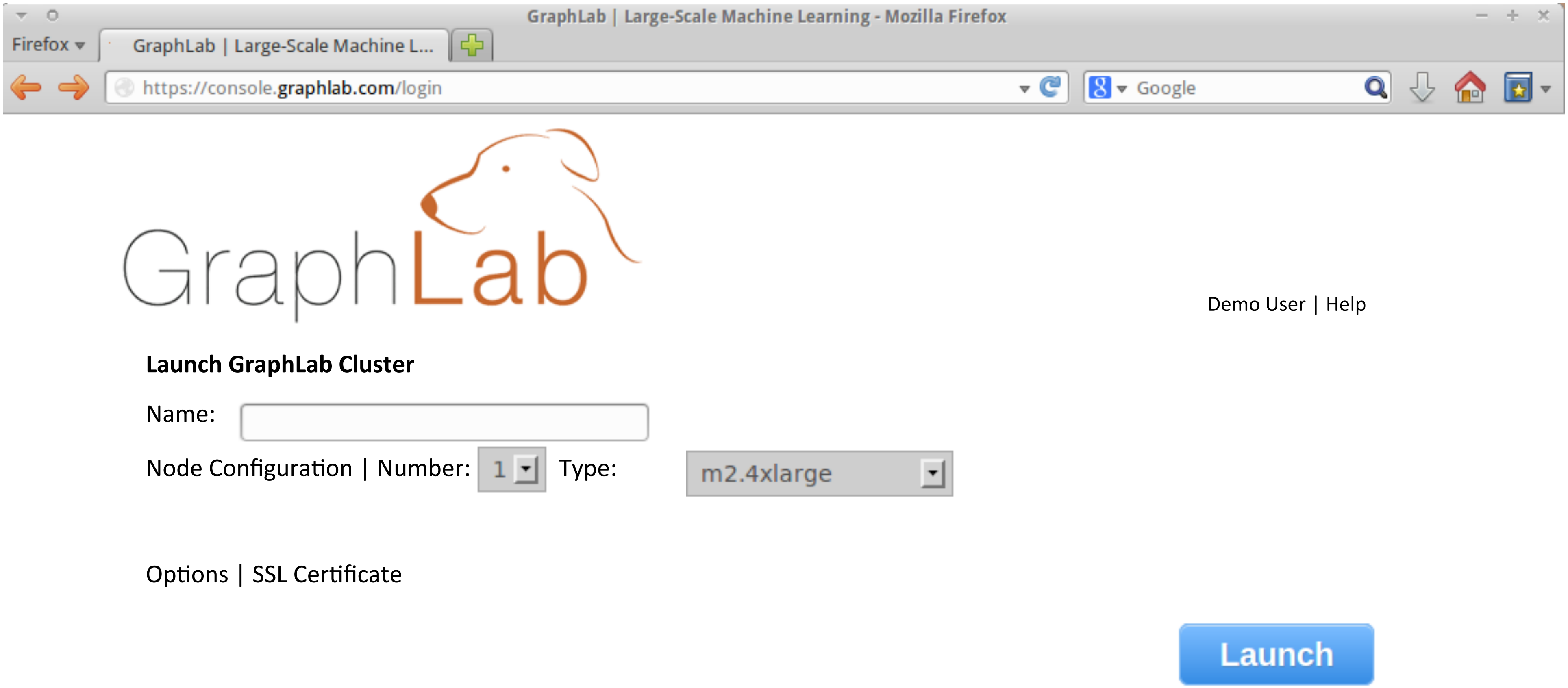
User study size = 2 :-)

Bigger + Real User Study in Progress, as we release new open-source version of GraphLab

New abstraction simplifies writing programs in GraphLab

But you still need to get a cluster, install GraphLab, configure system…


GraphLab | Large-Scale Machine Learning - Mozilla Firefox
Firefox
GraphLab | Large-Scale Machine L...
https://console.graphlab.com/login
Google
GraphLab
Your login :
Password :
Log in
Cancel
You don't have an account yet? Register here


GraphLab | Large-Scale Machine Learning - Mozilla Firefox
Firefox
GraphLab | Large-Scale Machine L...
https://console.graphlab.com/login
Google
GraphLab
Demo User | Help
Launch GraphLab Cluster
Name:
Node Configuration | Number:
1
Type:
m2.4xlarge
Options | SSL Certificate
Launch

GraphLab | Large-Scale Machine Learning - Mozilla Firefox

Firefox | GraphLab | Large-Scale Machine L...

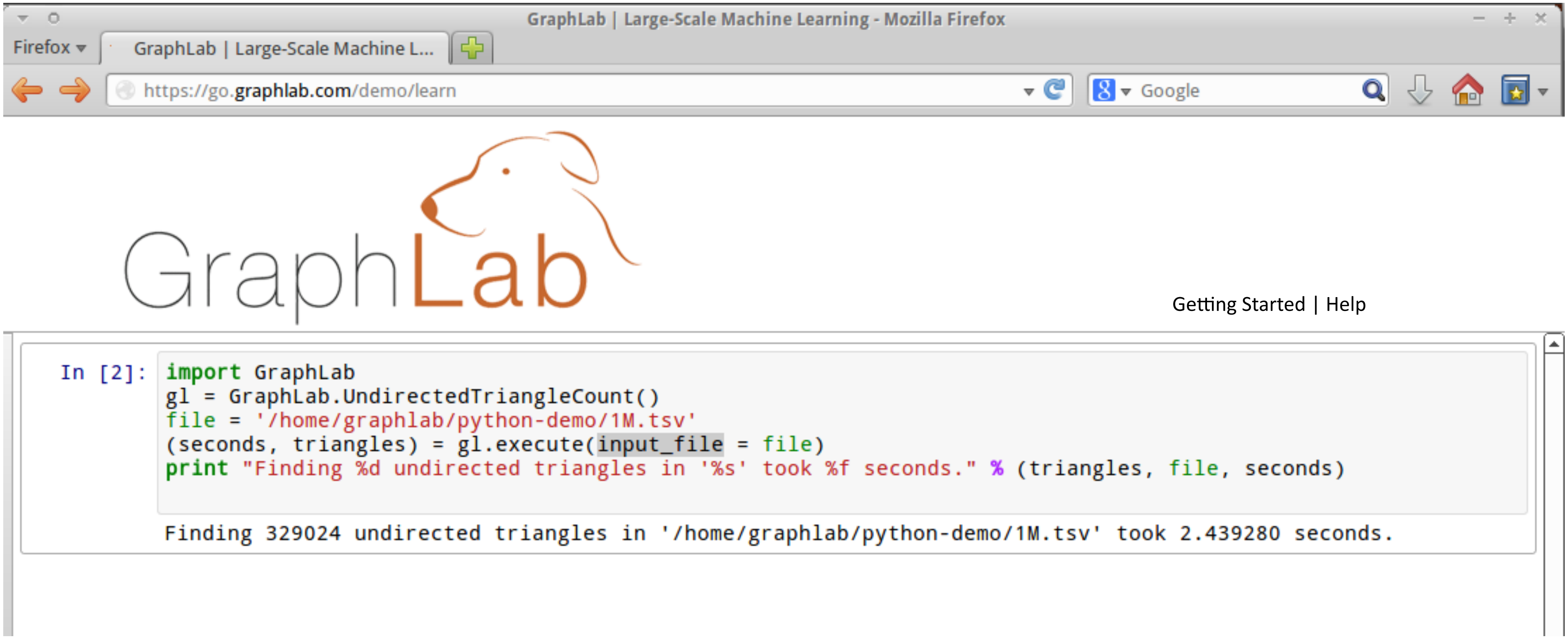
https://console.graphlab.com/login | Google

GraphLab

Demo User | Help

Clusters

Name	Connect URL	Configuration	Status	Last
demo	https://go.graphlab.com/user/demo/learn Connect	1 Node (m2.4xlarge)	*Running*	today


GraphLab | Large-Scale Machine Learning - Mozilla Firefox
Firefox
GraphLab | Large-Scale Machine L...
https://go.graphlab.com/demo/learn
Google
GraphLab
Getting Started | Help
In [2]:
import GraphLab
gl = GraphLab.UndirectedTriangleCount()
file = '/home/graphlab/python-demo/1M.tsv'
(seconds, triangles) = gl.execute(input_file = file)
print "Finding %d undirected triangles in '%s' took %f seconds." % (triangles, file, seconds)
Finding 329024 undirected triangles in '/home/graphlab/python-demo/1M.tsv' took 2.439280 seconds.

GraphLab 1
Carnegie Mellon

- ML algorithms as vertex programs
- Asynchronous execution and consistency models

GraphLab 2
PowerGraph

- Natural graphs change the nature of computation
- Vertex cuts and gather/apply/scatter model

GraphLab 3
WarpGraph

- Usability is key
- Access neighborhood through parallelizable iterators and latency hiding

Usability for Whom???

GL2
PowerGraph

GL3
WarpGraph

...

Anyone in my lab

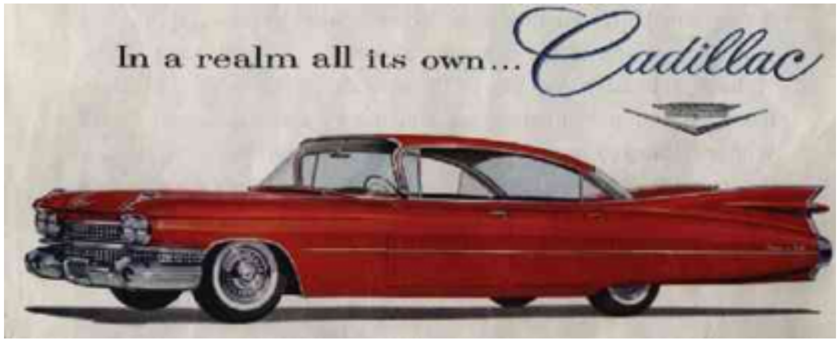
Anyone in this hall

Any "Data Scientist"

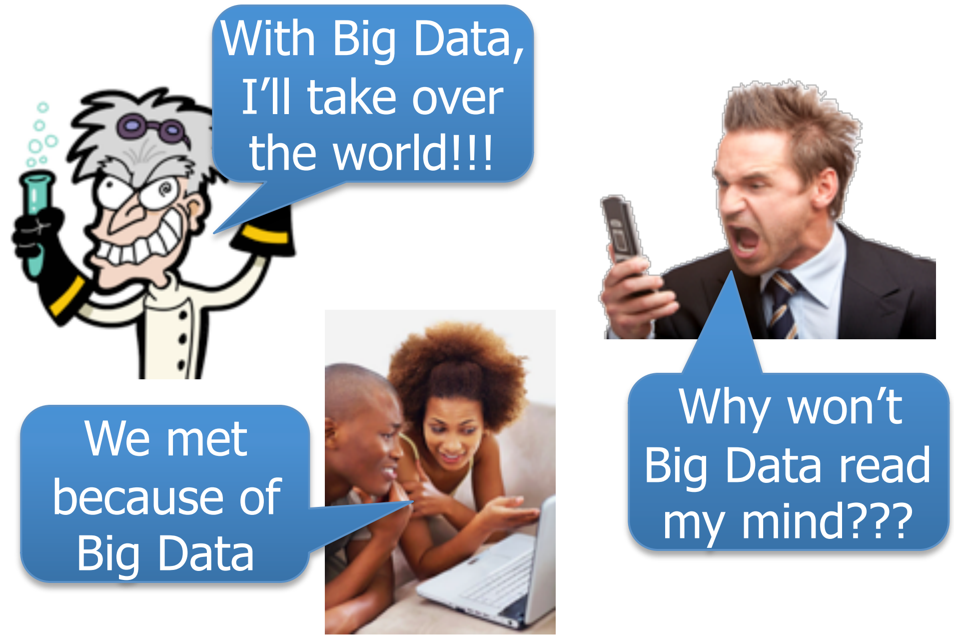
Anyone with Big Data

Machine Learning
PHASE 3

USABILITY



Exciting Time to Work in ML



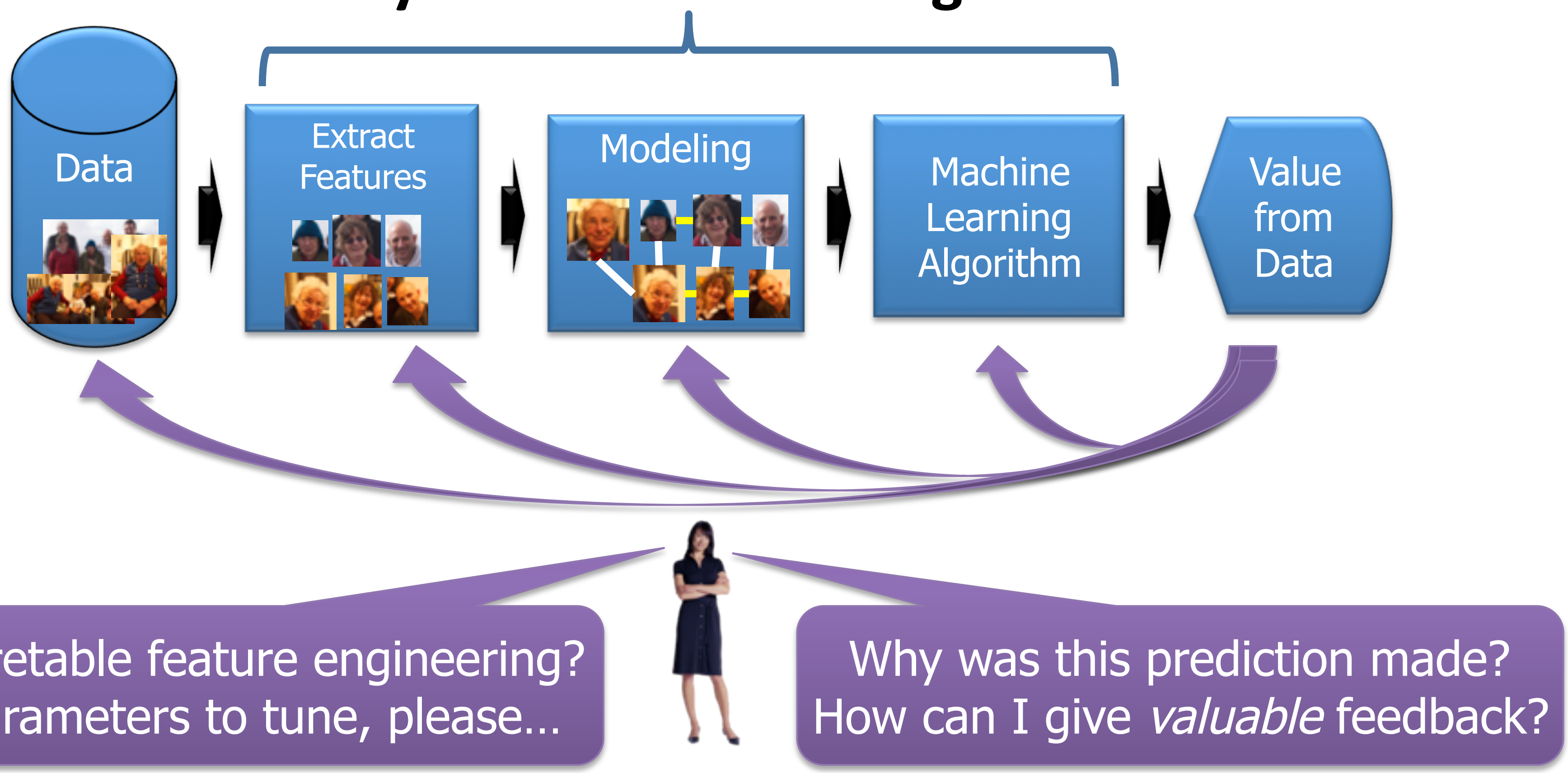
Unique opportunities to change the world!! ☺

But, **every deployed system** is an one-off solution, and **requires PhDs** to make work… ☹

And, Usability for ML is not just "Engineering" – Must Be Easy to Iterate through Models to Solve Task

But, when ML doesn't work, need a PhD to understand why…

GraphLab **One kind of ML usability: Fast and easy iterations over huge datasets**



GraphLab

v1 Possibility

v2 Scalability

v3 Usability

GraphLab 2.2 available now: **graphlab.com**