Parallel Programming Map-Reduce

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

February 24, 2014

Needless to Say, We Need Machine Learning for Big Data

flickr

6 Billion
Flickr Photos

28 Million
Wikipedia Pages

facebook

1 Billion
Facebook Users

YouTube

72 Hours a Minute
YouTube

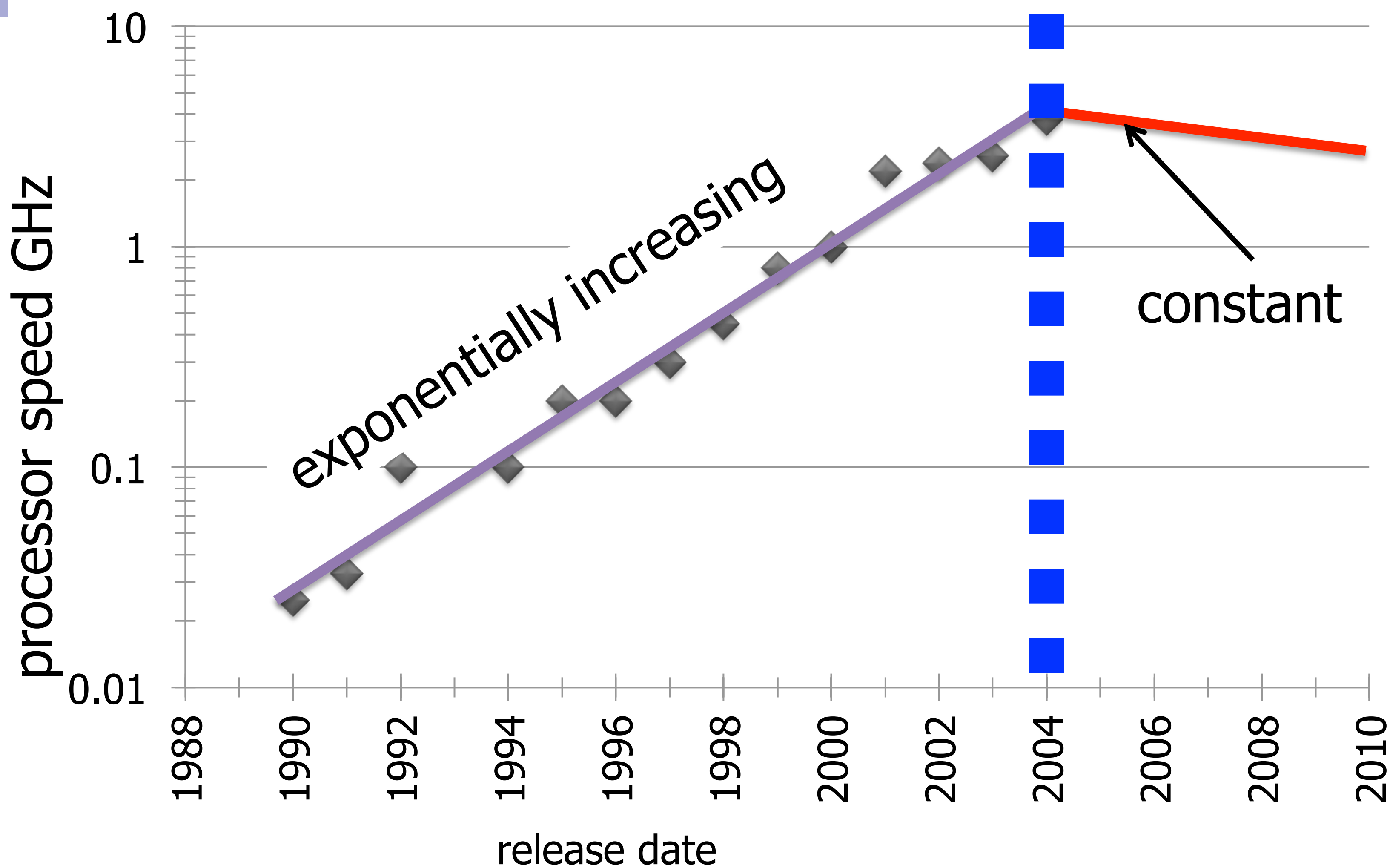
The New York Times
SundayReview

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

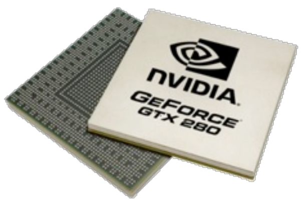
NEWS ANALYSIS
The Age of Big Data
By STEVE LOHR
Published: February 11, 2012

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

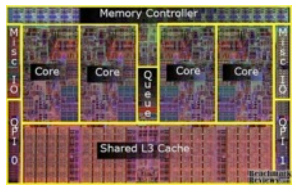
CPUs Stopped Getting Faster…



ML in the Context of Parallel Architectures



GPUs



Multicore

Clusters

amazon web services™

Clouds

Supercomputers

- But scalable ML in these systems is hard, especially in terms of:
  1. Programmability
  2. Data distribution
  3. Failures

Programmability Challenge 1: Designing Parallel programs

- SGD for LR:
  - For each data point $\mathbf{x}^{(t)}$:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + \phi_i(\mathbf{x}^{(t)})[y^{(t)} - P(Y=1|\phi(\mathbf{x}^{(t)}), \mathbf{w}^{(t)})] \right\}$$

Programmability Challenge 2: Race Conditions

- We are used to sequential programs:
  - Read data, think, write data, read data, think, write data, read data, think, write data, read data, think, write data, read data, think, write data, read data, think, write data…
- But, in parallel, you can have non-deterministic effects:
  - One machine reading data will other is writing
- Called a race-condition:
  - Very annoying
  - One of the hardest problems to debug in practice:
    - because of non-determinism, bugs are hard to reproduce

Data Distribution Challenge

- Accessing data:
  - Main memory reference: 100ns (10^{-7}s)
  - Round trip time within data center: 500,000ns ($5 * 10^{-4}$s)
  - Disk seek: 10,000,000ns (10^{-2}s)
- Reading 1MB sequentially:
  - Local memory: 250,000ns ($2.5 * 10^{-4}$s)
  - Network: 10,000,000ns (10^{-2}s)
  - Disk: 30,000,000ns ($3*10^{-2}$s)

- Conclusion: Reading data from local memory is **much** faster → Must have data locality:
  - Good data partitioning strategy fundamental!
  - "Bring computation to data" (rather than moving data around)

Robustness to Failures Challenge

- From Google's Jeff Dean, about their clusters of 1800 servers, in first year of operation:
  - 1,000 individual machine failures
  - thousands of hard drive failures
  - one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours
  - 20 racks will fail, each time causing 40 to 80 machines to vanish from the network
  - 5 racks will "go wonky," with half their network packets missing in action
  - the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span
  - 50% chance cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover

- How do we design distributed algorithms and systems robust to failures?
  - It's not enough to say: run, if there is a failure, do it again… because you may never finish

Move Towards Higher-Level Abstraction

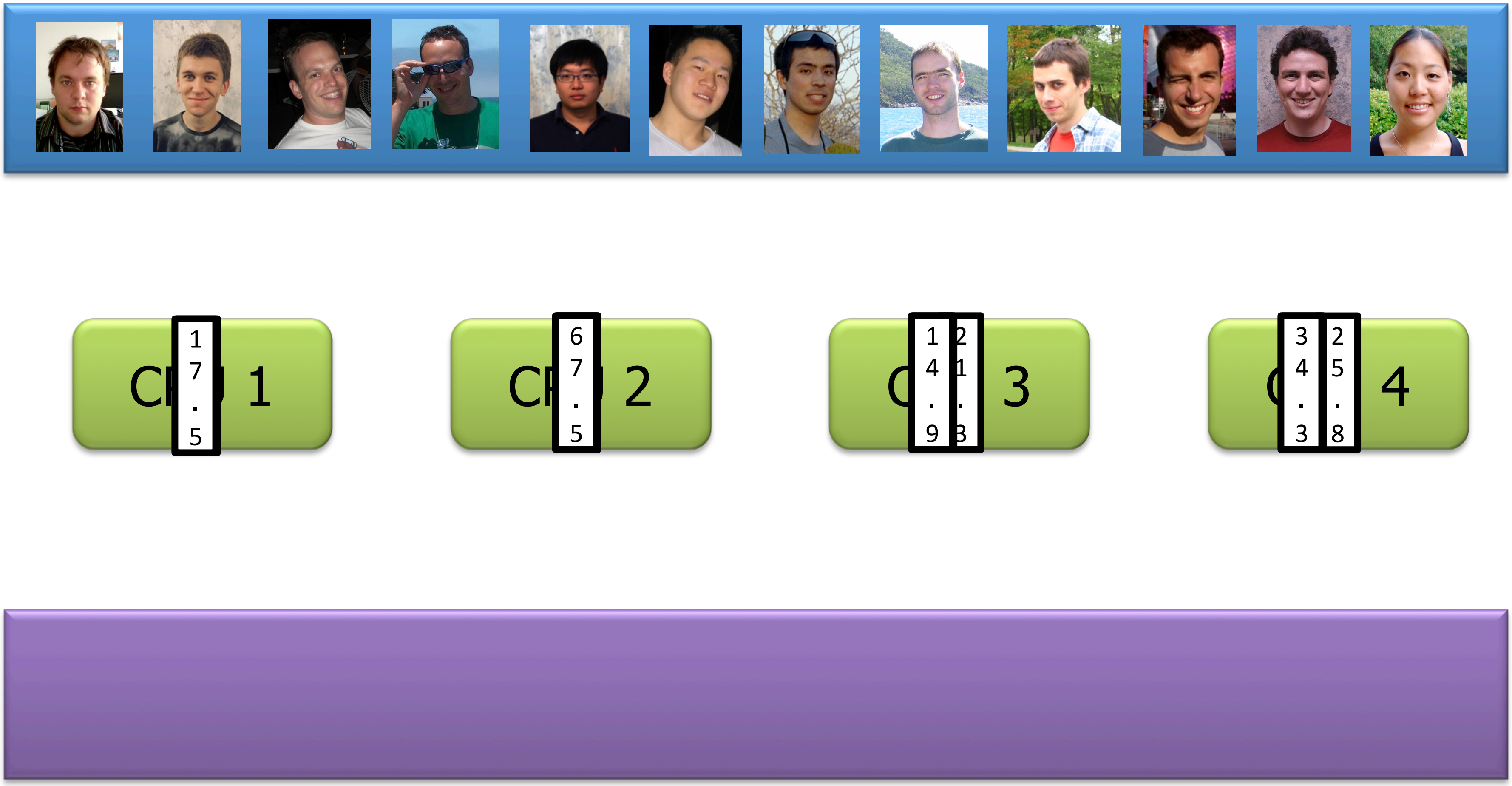
- Distributed computing challenges are hard and annoying!
  1. Programmability
  2. Data distribution
  3. Failures
- High-level abstractions try to simplify distributed programming by hiding challenges:
  - Provide different levels of robustness to failures, optimizing data movement and communication, protect against race conditions…
  - Generally, you are still on your own WRT designing parallel algorithms
- Some common parallel abstractions:
  - Lower-level:
    - Pthreads: abstraction for distributed threads on single machine
    - MPI: abstraction for distributed communication in a cluster of computers
  - Higher-level:
    - Map-Reduce (Hadoop: open-source version): mostly data-parallel problems
    - GraphLab: for graph-structured distributed problems

Simplest Type of Parallelism: Data Parallel Problems

- You have already learned a classifier
  - What's the test error?
- You have 10B labeled documents and 1000 machines

- Problems that can be broken into independent subproblems are called data-parallel (or embarrassingly parallel)
- Map-Reduce is a great tool for this…
  - Focus of today's lecture
  - but first a simple example

Data Parallelism (MapReduce)



Solve a huge number of ***independent*** *subproblems, e.g., extract features in images*

Counting Words on a Single Processor

- (This is the “Hello World!” of Map-Reduce)
- Suppose you have 10B documents and 1 machine
- You want to count the number of appearances of each word on this corpus
  - Similar ideas useful, e.g., for building Naïve Bayes classifiers and computing TF-IDF
- Code:

Naïve Parallel Word Counting

- Simple data parallelism approach:

- Merging hash tables: annoying, potentially not parallel ➔ no gain from parallelism???

Counting Words in Parallel & Merging Hash Tables in Parallel

- Generate pairs (word,count)
- Merge counts for each word in parallel
  - Thus parallel merging hash tables

Map-Reduce Abstraction

- Map:
  - Data-parallel over elements, e.g., documents
  - Generate (key,value) pairs
    - "value" can be any data type

- Reduce:
  - Aggregate values for each key
  - Must be commutative-associate operation
  - Data-parallel over keys
  - Generate (key,value) pairs

- Map-Reduce has long history in functional programming
  - But popularized by Google, and subsequently by open-source Hadoop implementation from Yahoo!

Map Code (Hadoop): Word Count

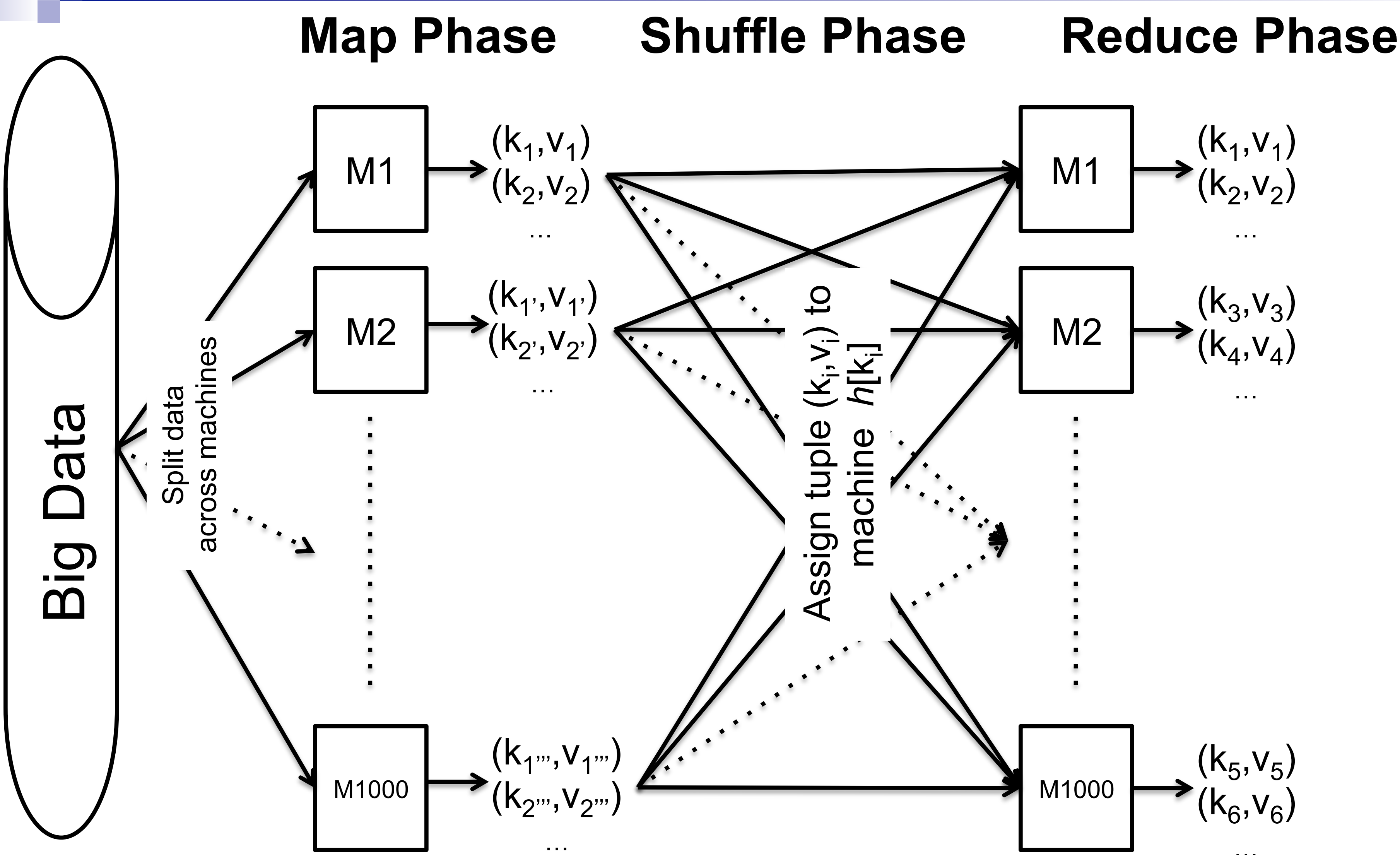
```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws <stuff>
     {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
     }
 }
```

Reduce Code (Hadoop): Word Count

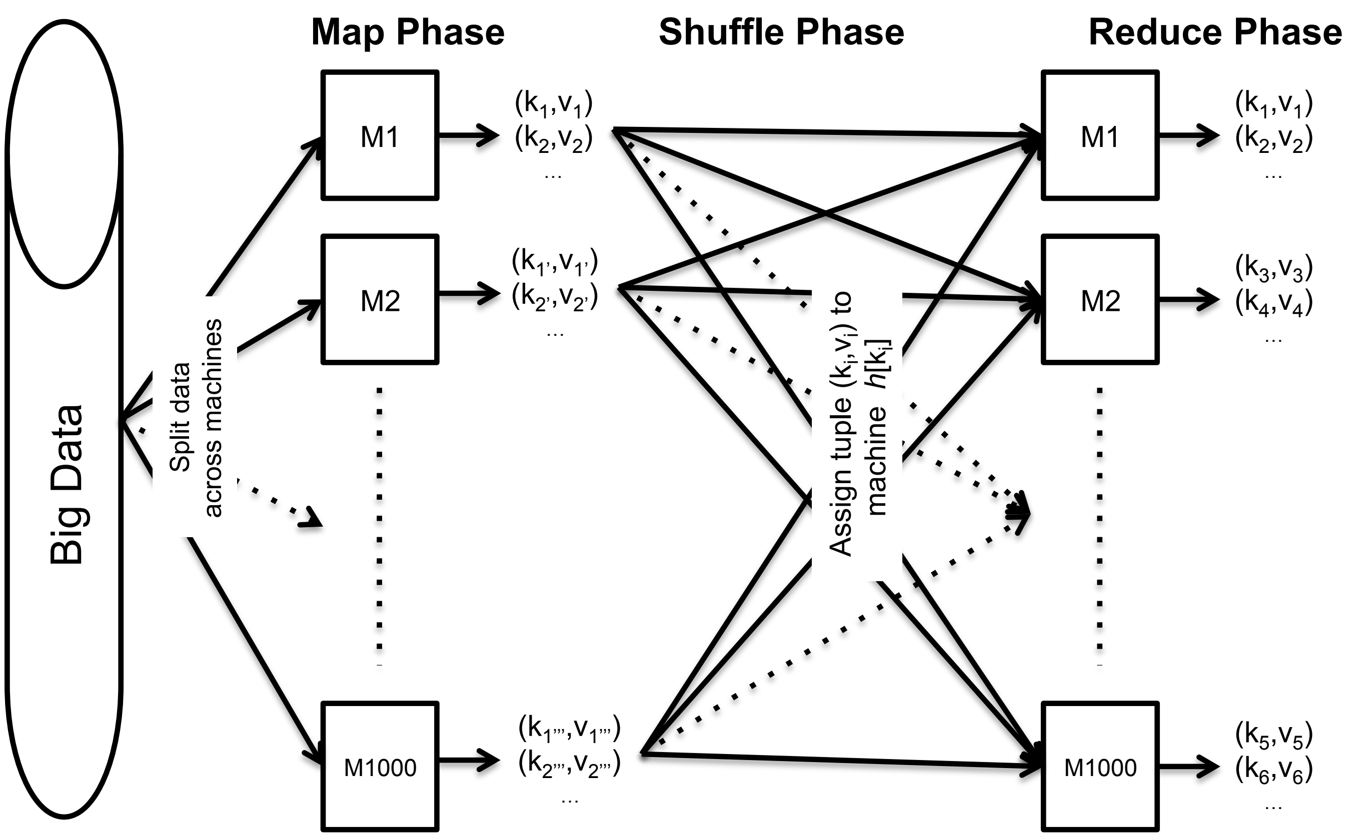
```
public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
                                 Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }
```

Map-Reduce Parallel Execution

Map-Reduce – Execution Overview



Map-Reduce – Robustness to Failures 1: Protecting Data: **Save To Disk Constantly**



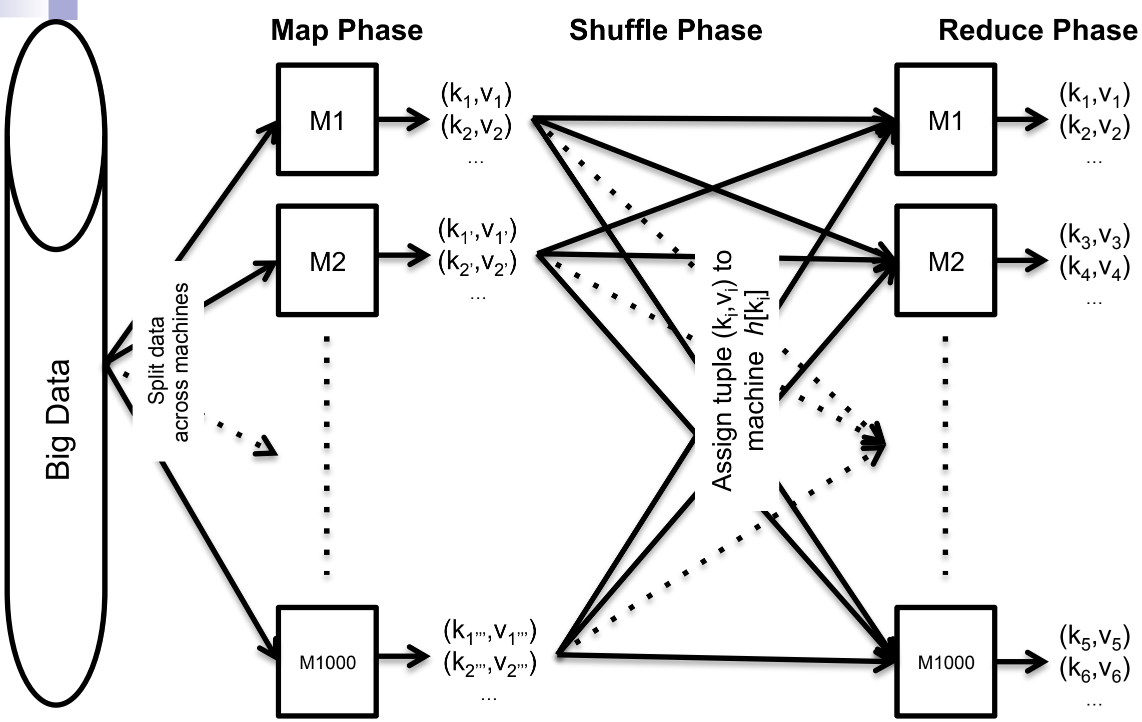
Distributed File Systems

- Saving to disk locally is not enough ➔ If disk or machine fails, all data is lost
- Replicate data among multiple machines!

- Distributed File System (DFS)
  - Write a file anywhere ➔ automatically replicated
  - Can read a file anywhere ➔ read from closest copy
    - If failure, try next closest copy

- Common implementations:
  - Google File System (GFS)
  - Hadoop File System (HDFS)
- Important practical considerations:
  - Write large files
    - Many small files ➔ becomes way too slow
  - Typically, files can't be "modified", just "replaced" ➔ makes robustness much simpler

Map-Reduce – Robustness to Failures 2: Recovering From Failures: **Read from DFS**



- Communication in initial distribution & shuffle phase "automatic"
  - Done by DFS
- If failure, don't restart everything
  - Otherwise, never finish
- Only restart Map/ Reduce jobs in dead machines

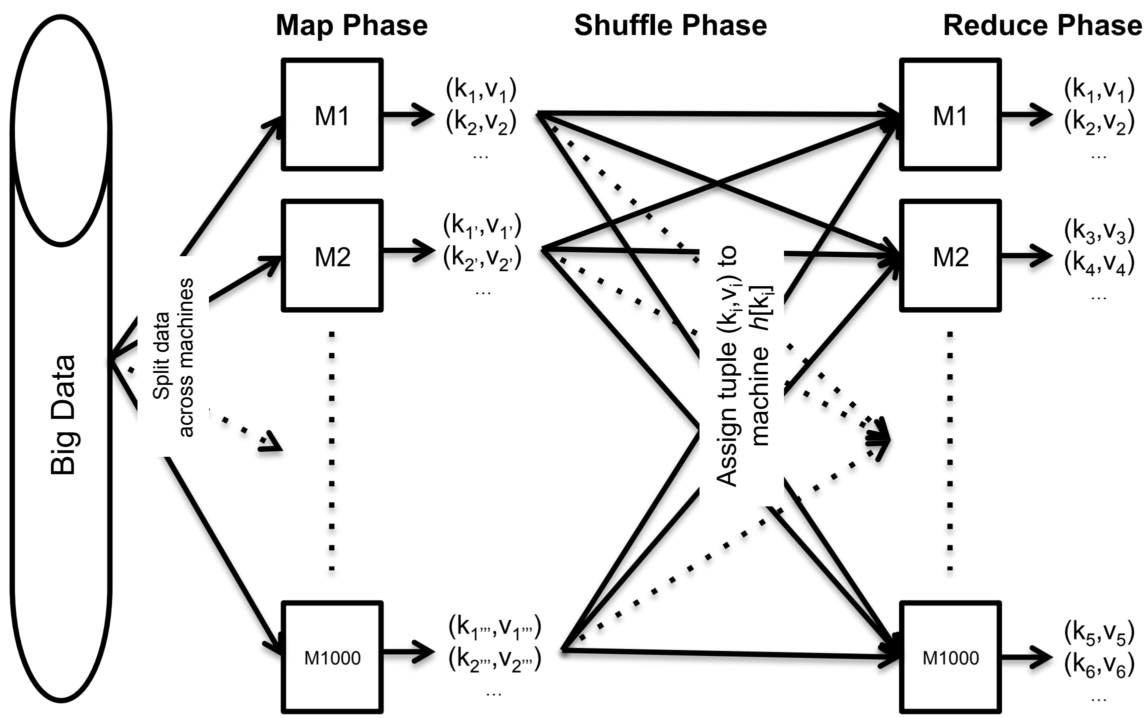
Improving Performance: Combiners

- Naïve implementation of M-R very wasteful in communication during shuffle:

- **Combiner**: Simple solution, perform reduce locally before communicating for global reduce
  - Works because reduce is commutative-associative

(A few of the) Limitations of Map-Reduce

- Too much synchrony
  - E.g., reducers don't start until all mappers are done
- "Too much" robustness
  - Writing to disk all the time
- Not all problems fit in Map-Reduce
  - E.g., you can't communicate between mappers
- Oblivious to structure in data
  - E.g., if data is a graph, can be much more efficient
    - For example, no need to shuffle nearly as much
- Nonetheless, extremely useful; industry standard for Big Data
  - Though many many companies are moving away from Map-Reduce (Hadoop)



What you need to know about Map-Reduce

- Distributed computing challenges are hard and annoying!
  1. Programmability
  2. Data distribution
  3. Failures
- High-level abstractions help a lot!
- Data-parallel problems & Map-Reduce
- Map:
  - Data-parallel transformation of data
    - Parallel over data points
- Reduce:
  - Data-parallel aggregation of data
    - Parallel over keys
- Combiner helps reduce communication
- Distributed execution of Map-Reduce:
  - Map, shuffle, reduce
  - Robustness to failure by writing to disk
  - Distributed File Systems

Parallel K-Means on Map-Reduce

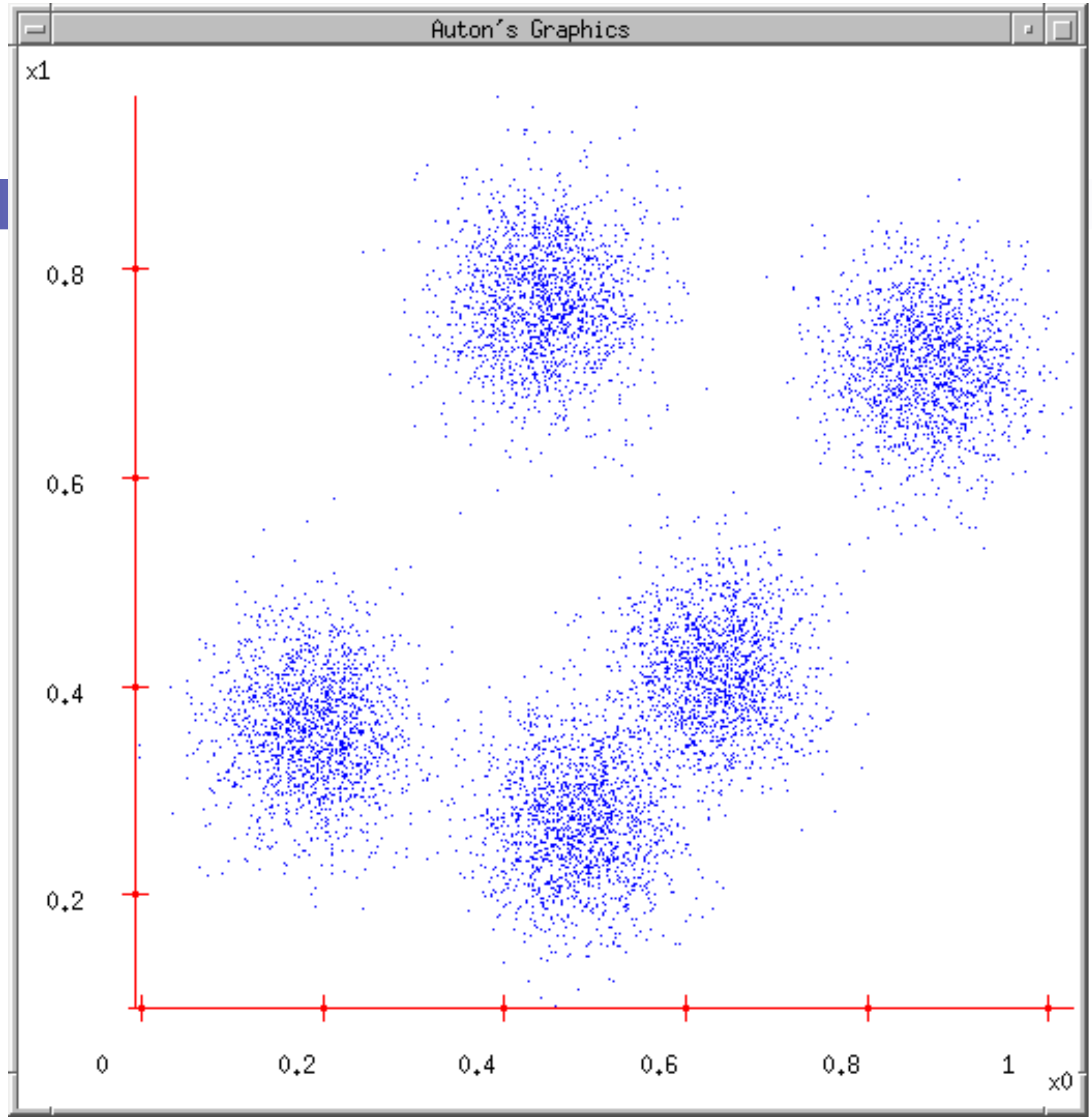
Machine Learning – CSEP546

Carlos Guestrin

University of Washington

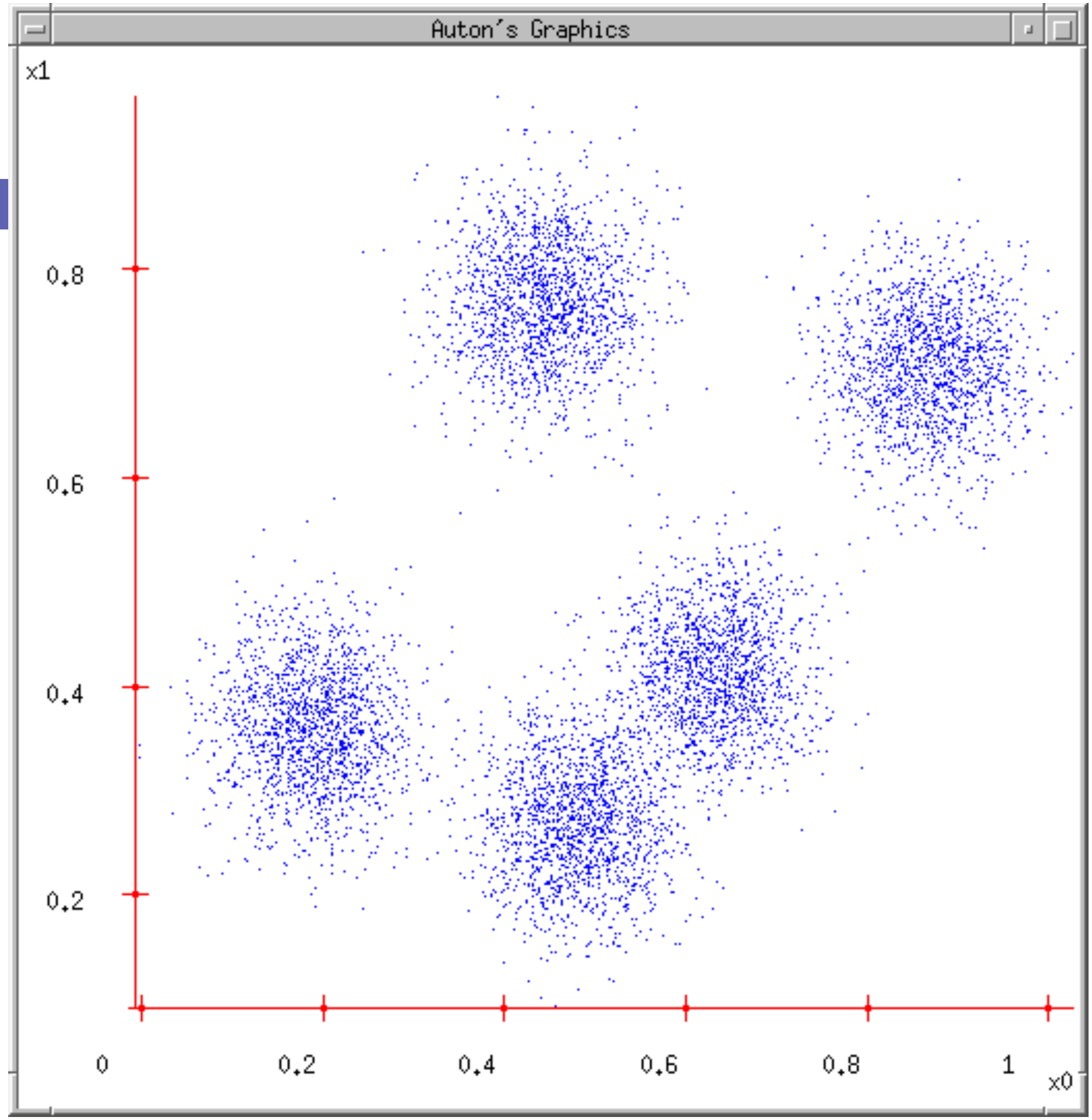
February 24, 2014

Some Data



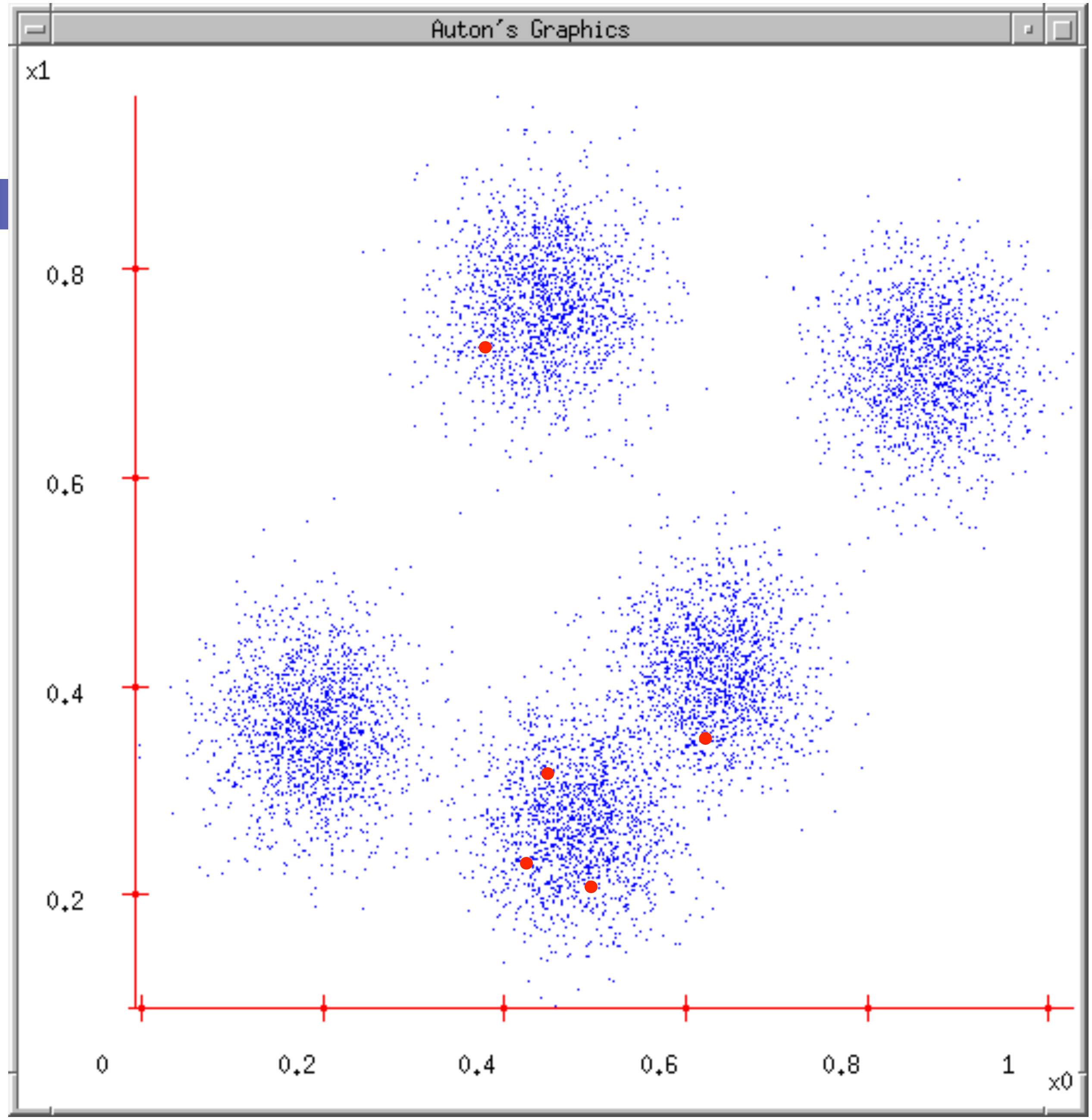
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*



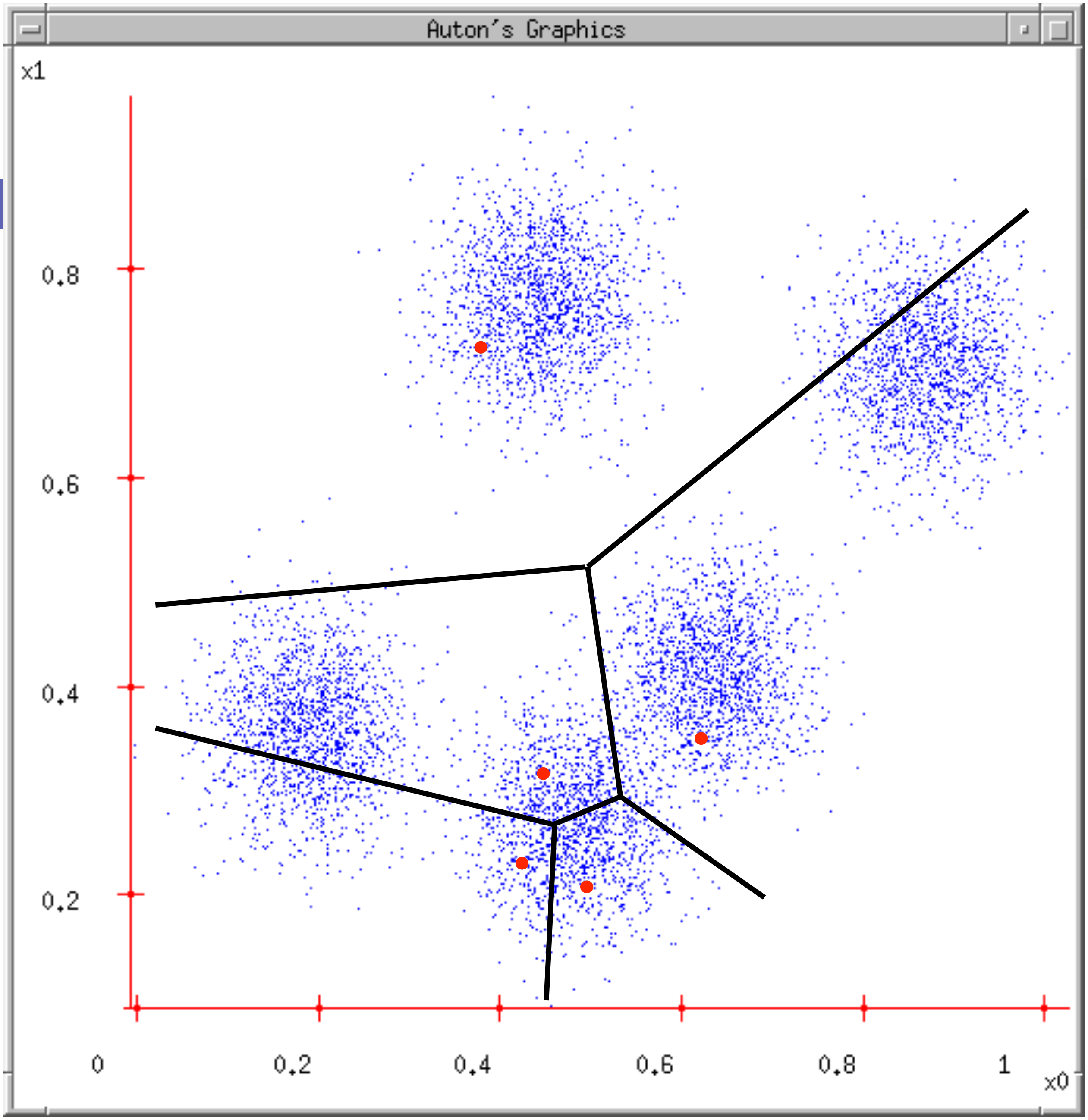
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



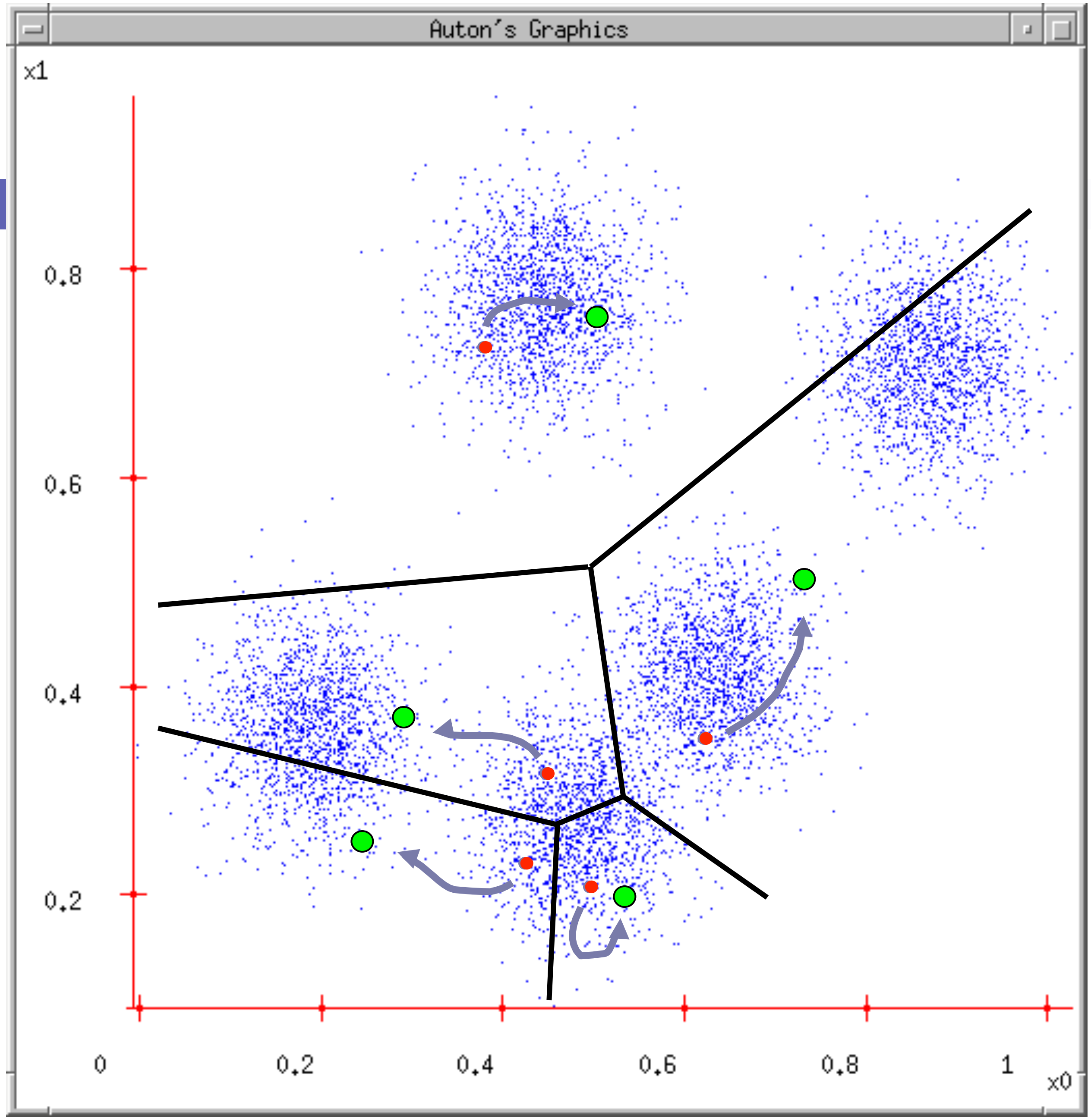
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



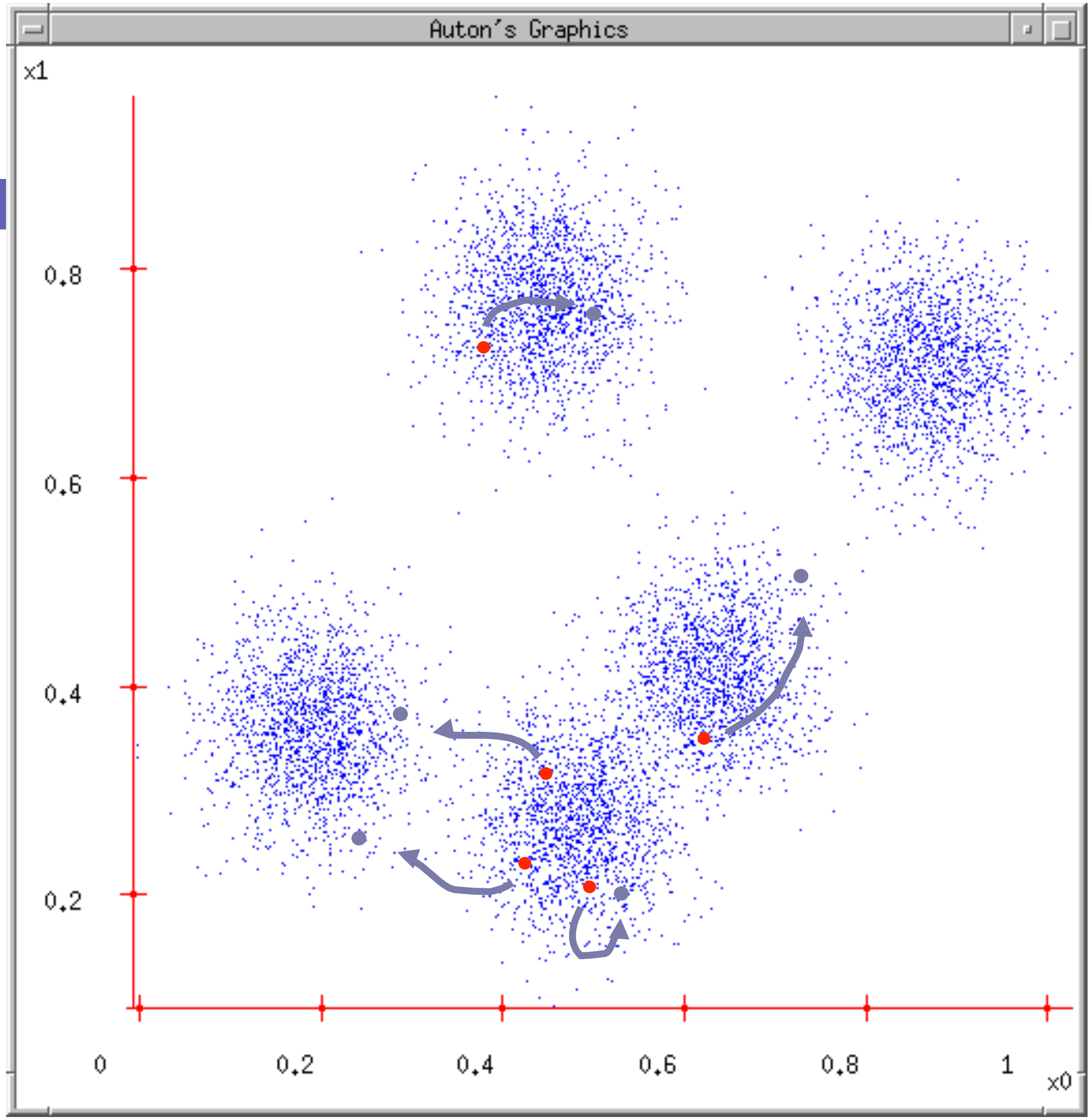
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



K-means

- Randomly initialize *k* centers
  - $\mu^{(0)} = \mu_1^{(0)}, \ldots, \mu_k^{(0)}$
- **Classify**: Assign each point j∈{1,…m} to nearest center:
  - $z^j \leftarrow \arg\min_i ||\mu_i - \mathbf{x}^j||_2^2$
- **Recenter**: μ_i becomes centroid of its point:
  - $$\mu_i^{(t+1)} \leftarrow \arg\min_{\mu} \sum_{j:z^j=i} ||\mu - \mathbf{x}^j||_2^2$$
  - Equivalent to $\mu_i \leftarrow$ average of its points!

Map-Reducing One Iteration of K-Means

- **Classify**: Assign each point j∈{1,…m} to nearest center:
  - $z^j \leftarrow \arg\min_i ||\mu_i - \mathbf{x}^j||_2^2$
- **Recenter**: μ_i becomes centroid of its point:
  - $\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:z^j=i} ||\mu - \mathbf{x}^j||_2^2$
  - Equivalent to μ_i ← average of its points!
- **Map**:
- **Reduce**:

Classification Step as Map

- **Classify**: Assign each point j∈{1,…m} to nearest center:
  - $z^j \leftarrow \arg\min_i ||\mu_i - \mathbf{x}^j||_2^2$

- **Map**:

Recenter Step as Reduce

- **Recenter**: μ_i becomes centroid of its point:
  - $\mu_i^{(t+1)} \leftarrow \arg\min_{\mu} \sum_{j:z^j=i} ||\mu - \mathbf{x}^j||_2^2$
  - Equivalent to $\mu_i \leftarrow$ average of its points!

- **Reduce**:

Some Practical Considerations

- K-Means needs an iterative version of Map-Reduce
  - Not standard formulation
- Mapper needs to get data point and all centers
  - A lot of data!
  - Better implementation: mapper gets many data points

What you need to know about Parallel K-Means on Map-Reduce

- K-Means = EM for mixtures of spherical Gaussians with hard assignments
- Map: classification step; data parallel over data point
- Reduce: recompute means; data parallel over centers

Graph-Parallel Problems

Synchronous v. Asynchronous Computation

Machine Learning – CSEP546
Carlos Guestrin
University of Washington
February 24, 2014

Issues with Map-Reduce Abstraction

- Often all data gets moved around cluster
  - Very bad for iterative settings

- Definition of Map & Reduce functions can be unintuitive in many apps
  - Graphs are challenging

- Computation is synchronous

SGD for Matrix Factorization in Map-Reduce?

$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1-\eta_t\lambda_u)L_u^{(t)} - \eta_t\epsilon_t R_v^{(t)} \\ (1-\eta_t\lambda_v)R_v^{(t)} - \eta_t\epsilon_t L_u^{(t)} \end{bmatrix}$$

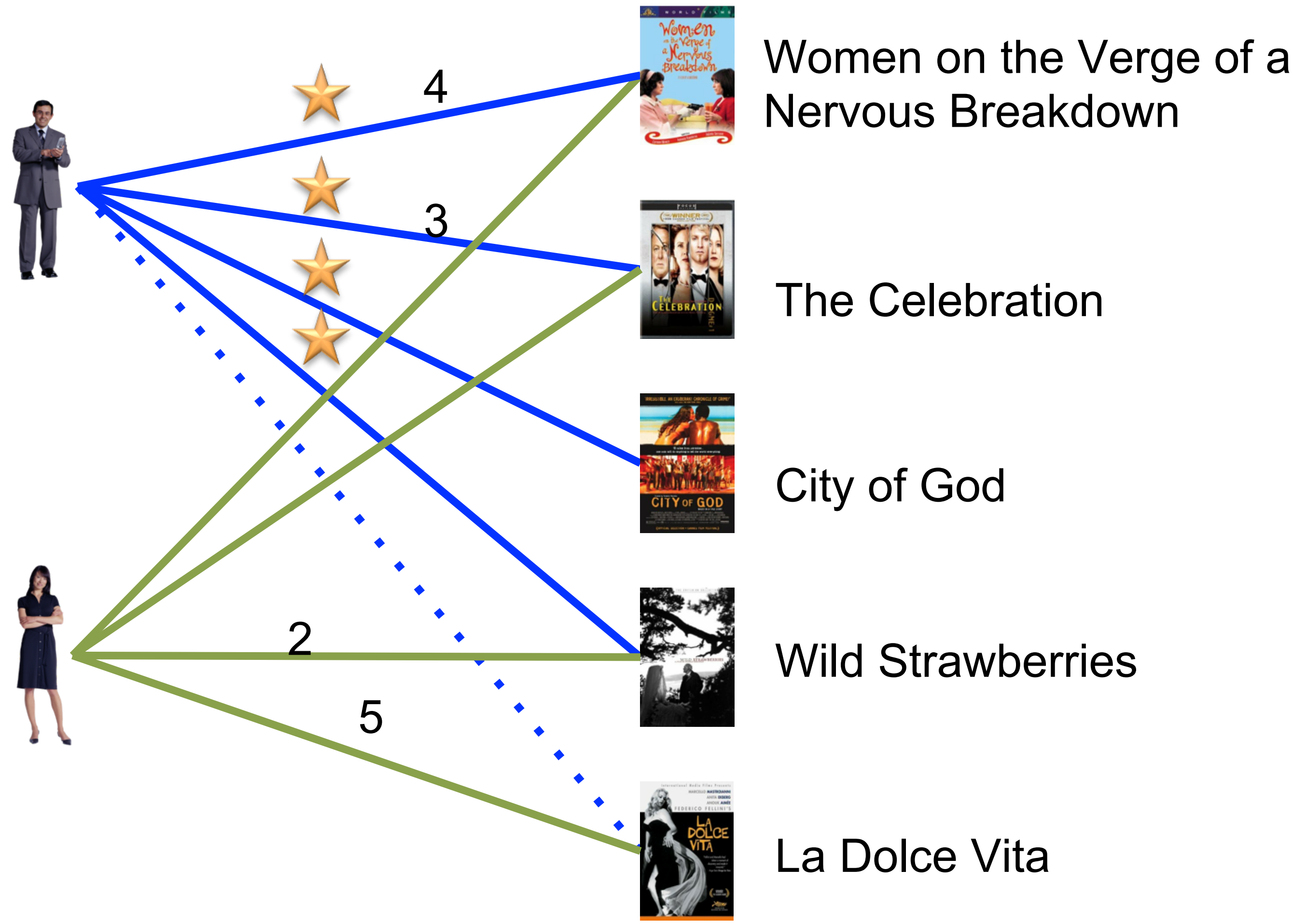
$$\epsilon_t = L_u^{(t)} \cdot R_v^{(t)} - r_{uv}$$

- Map and Reduce functions???

- Map-Reduce:
  - Data-parallel over all mappers
  - Data-parallel over reducers with same key

- Here, one update at a time!

Matrix Factorization as a Graph

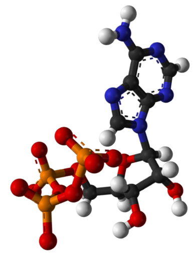


Flashback to 1998

Why?

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

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

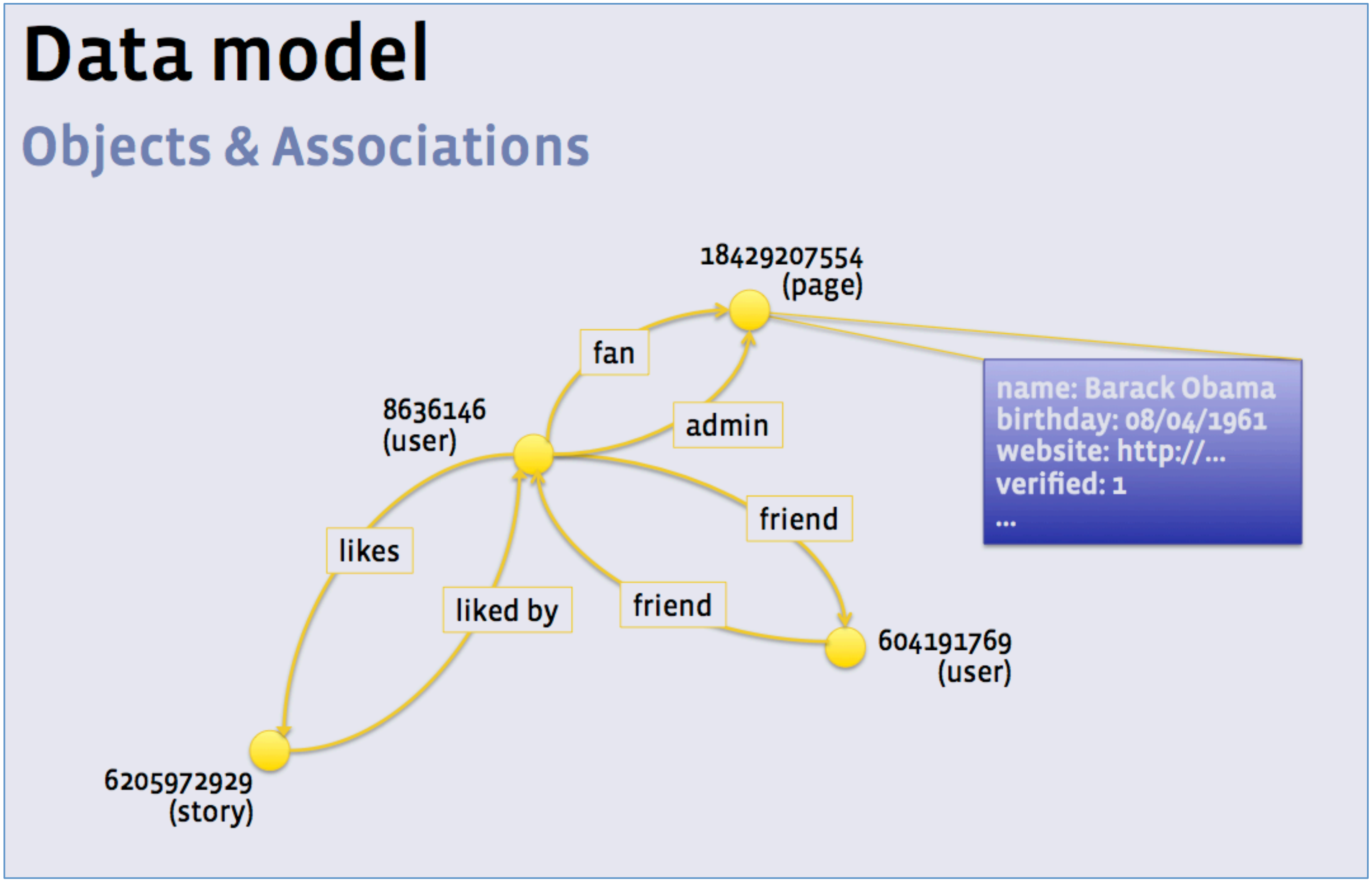


- **Graphs** encode the **relationships** between:

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

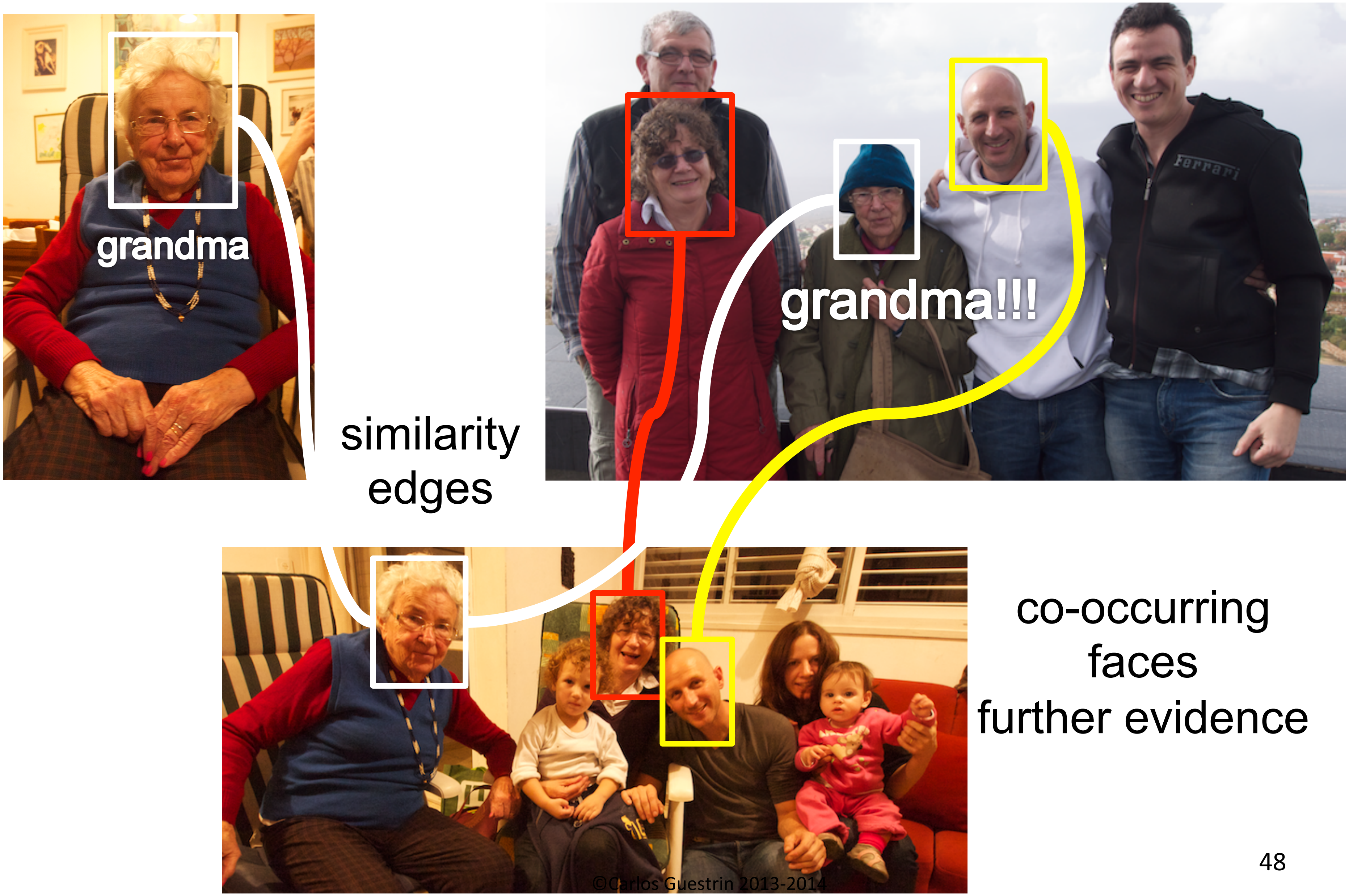


Slide from Facebook Engineering presentation

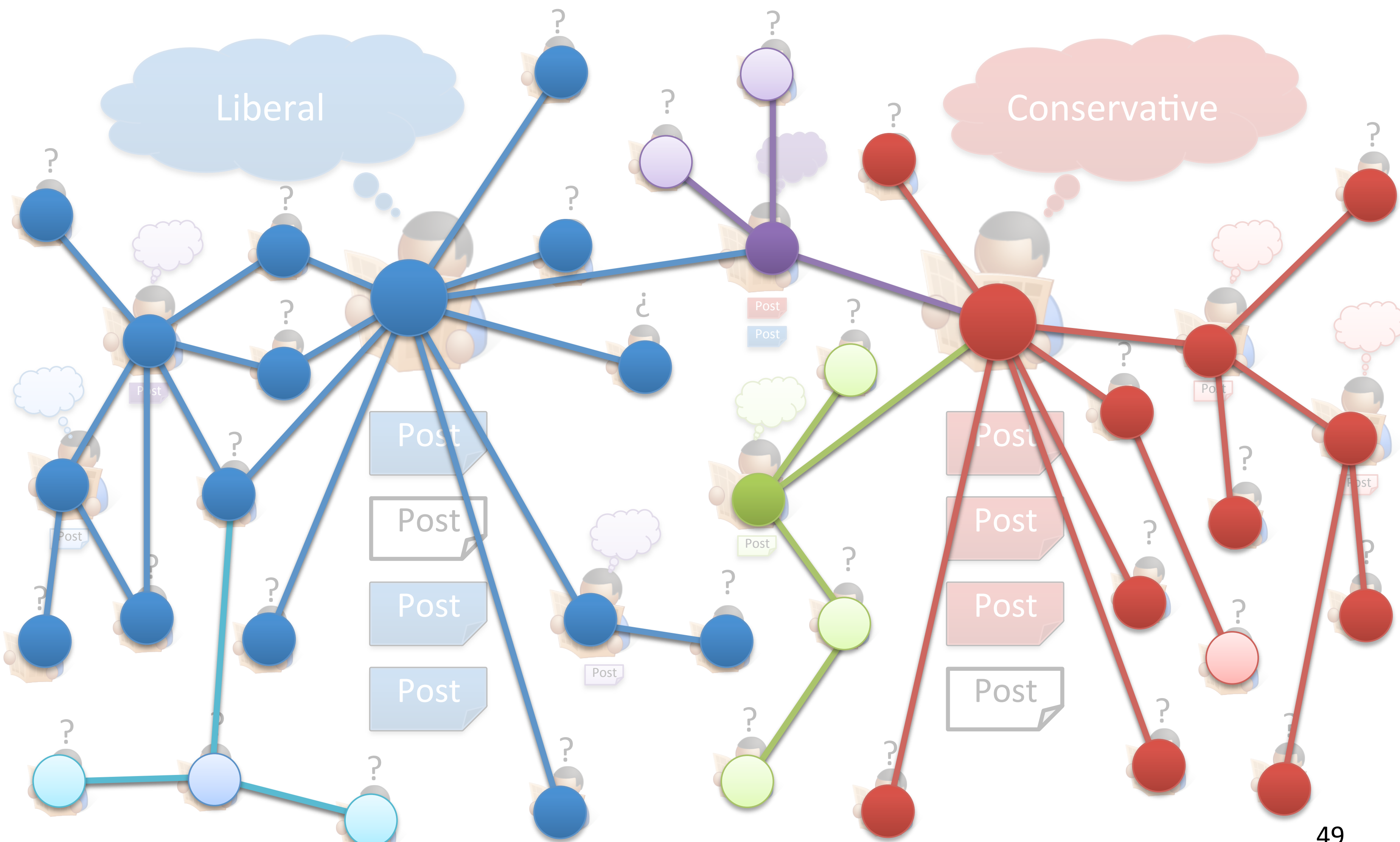
Label a Face and Propagate

Pairwise similarity not enough…

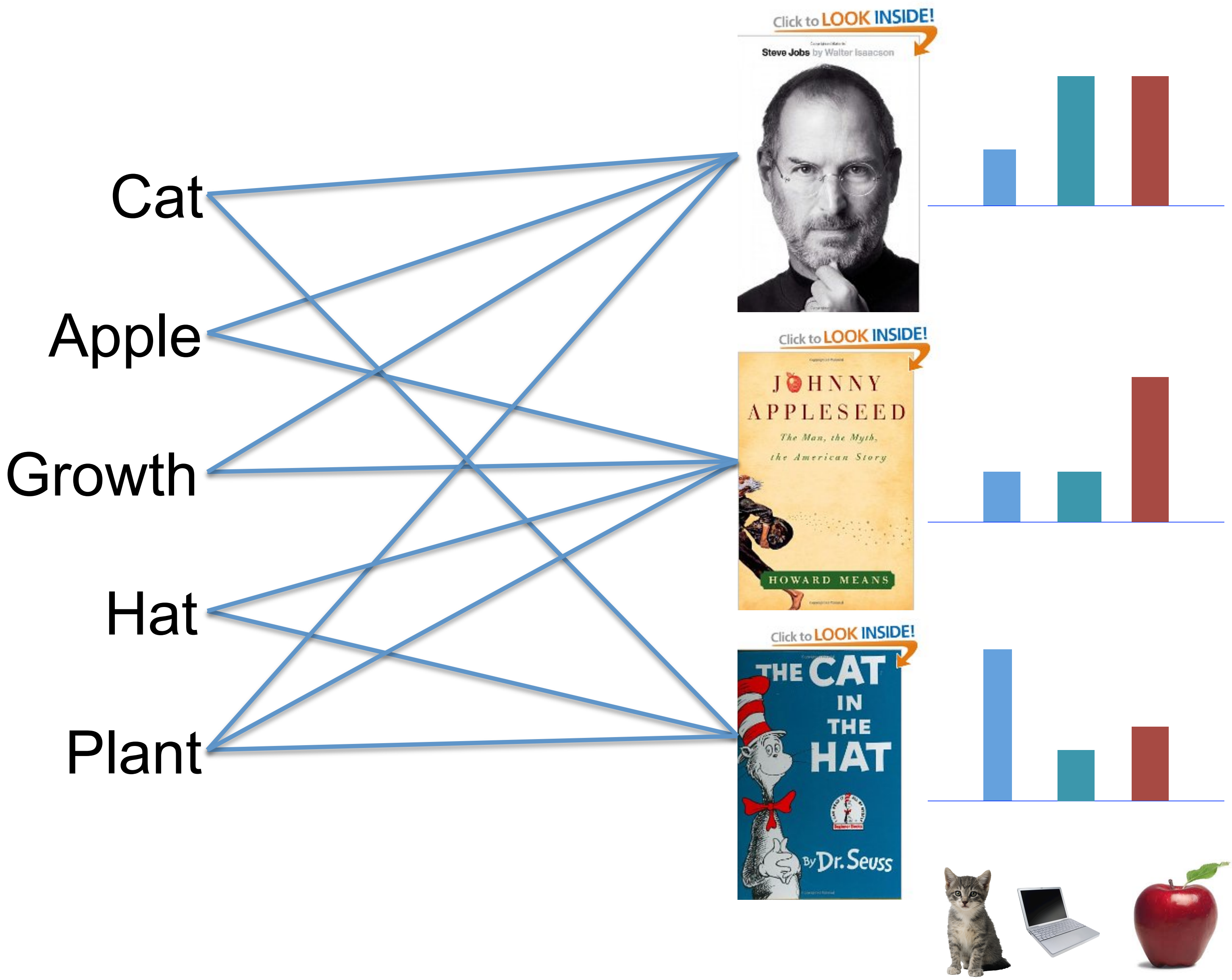
Propagate Similarities & Co-occurrences for Accurate Predictions



Example: *Estimate Political Bias*



Topic Modeling (e.g., LDA)



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.

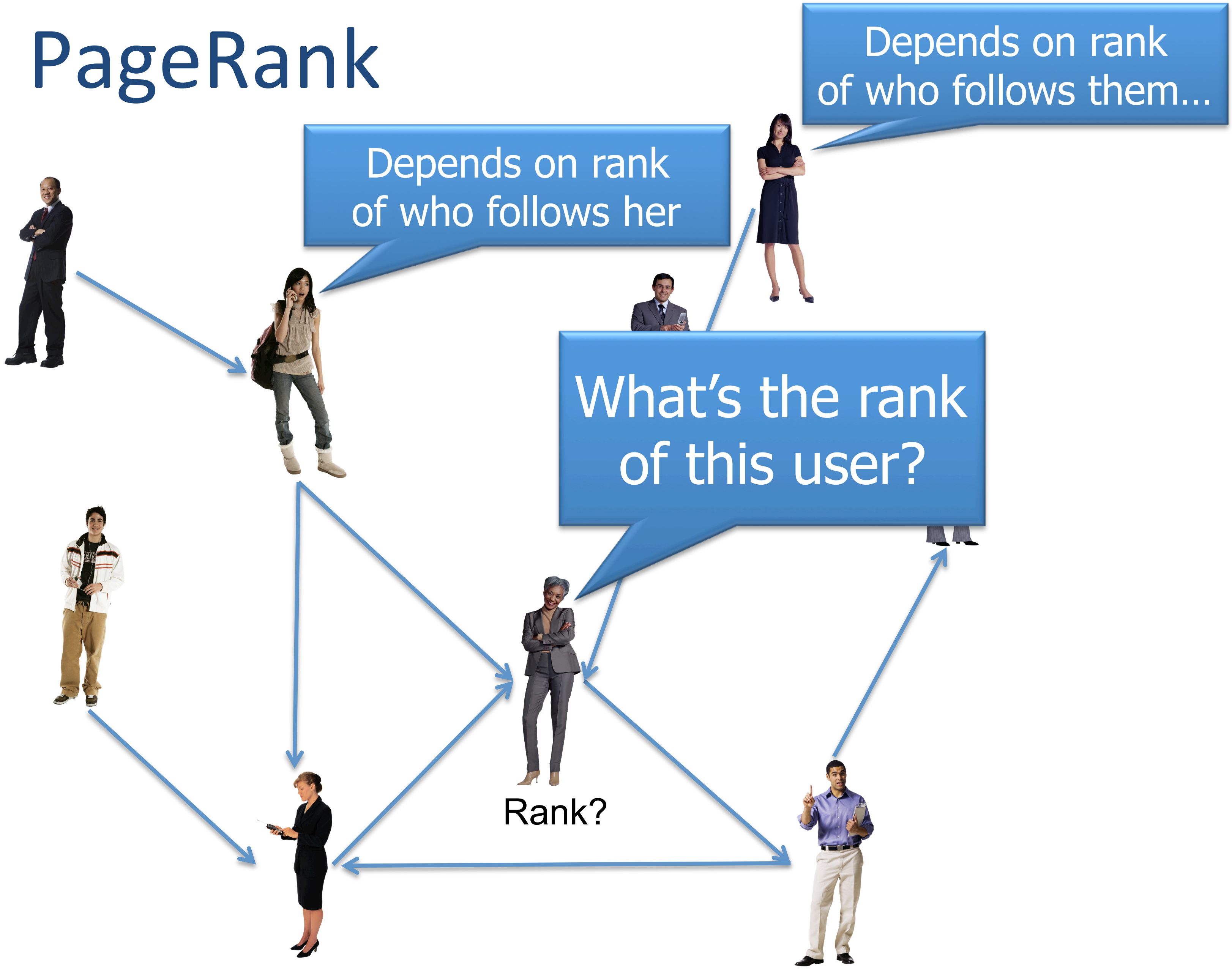
Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

Graph Analysis
PageRank
Triangle Counting

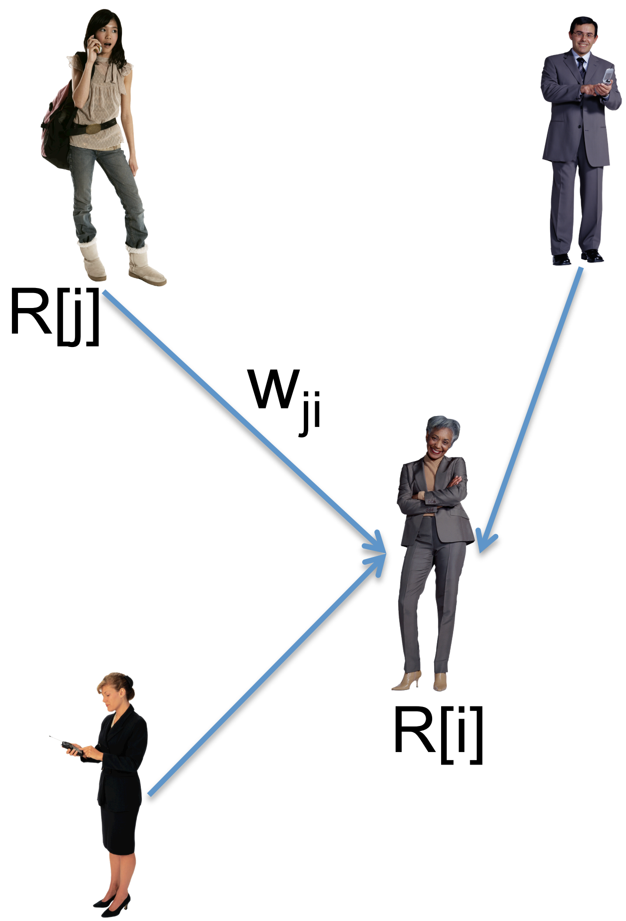
Example of a Graph-Parallel Algorithm

PageRank



Loops in graph → Must iterate!

PageRank Iteration

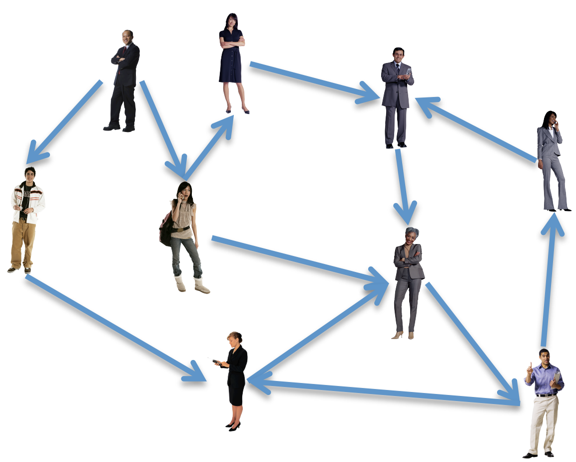


$$R[i] = \alpha + (1-\alpha) \sum_{(j,i)\in E} w_{ji} R[j]$$

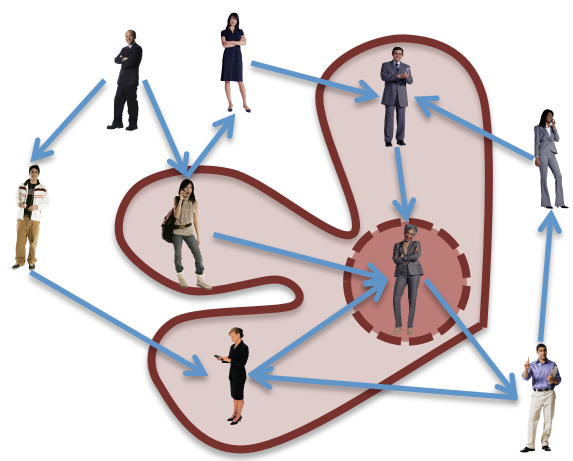
- α is the random reset probability
- w_{ji} is the prob. transitioning (similarity) from j to i

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:

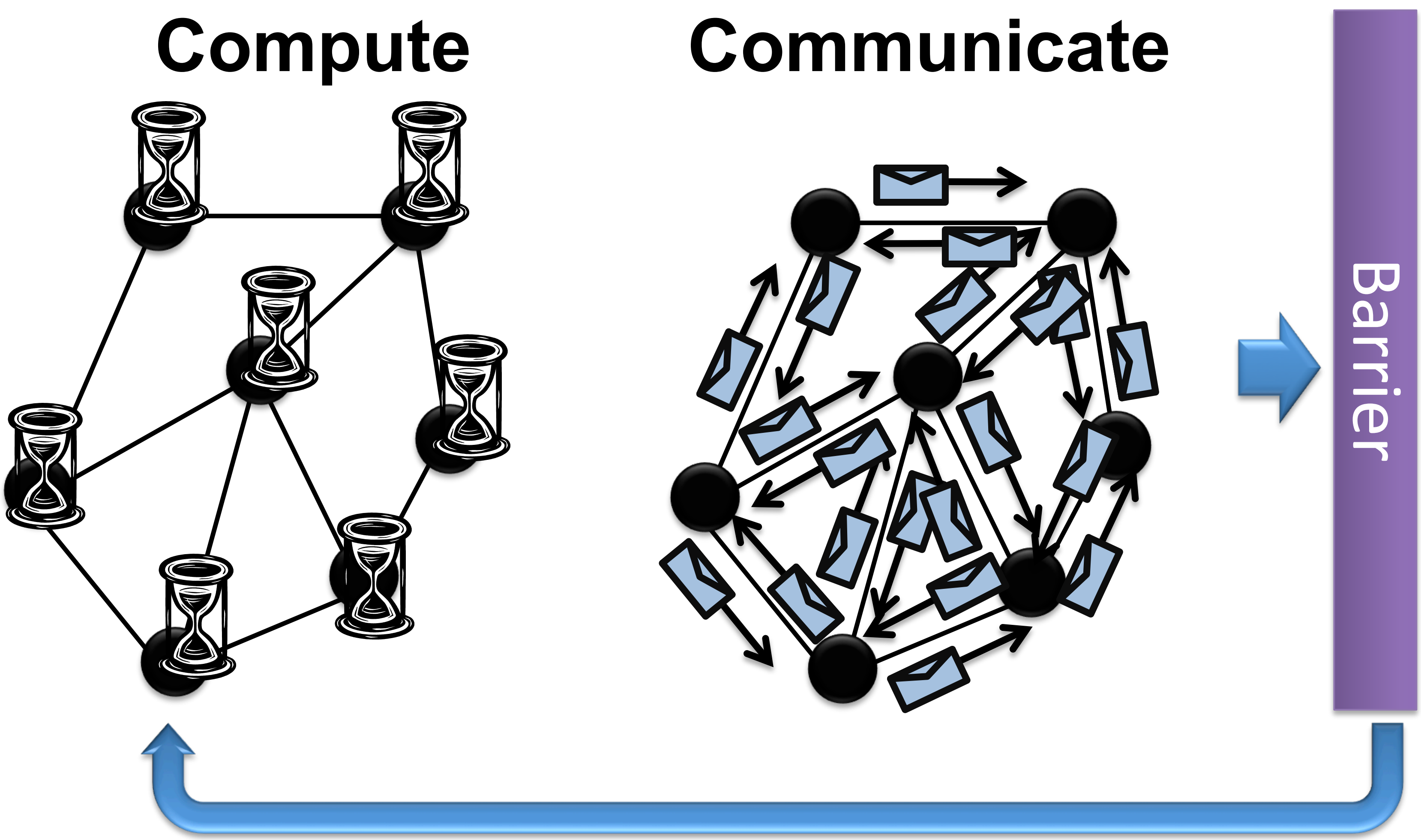
Synchronous

v.

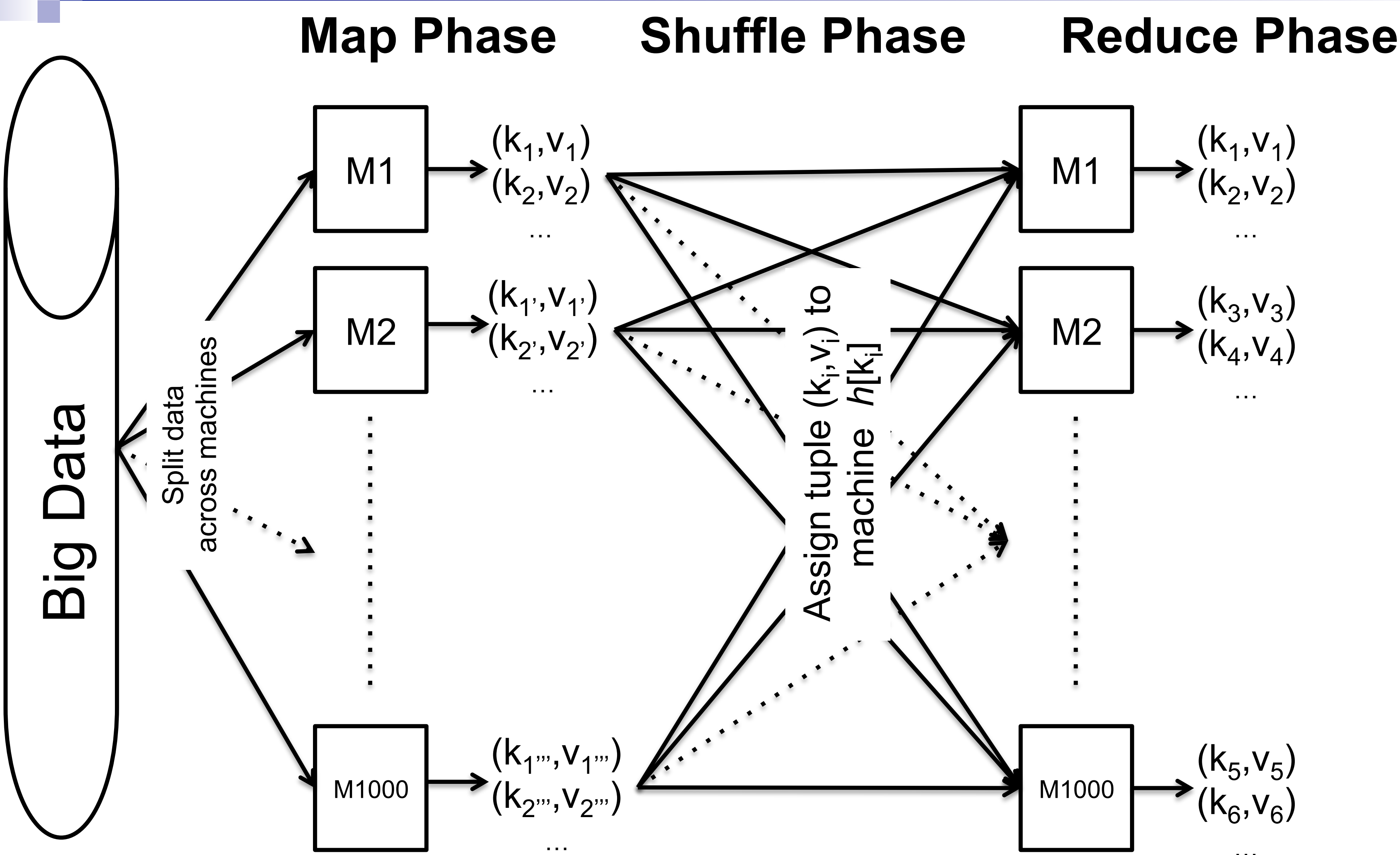
Asynchronous

Bulk Synchronous Parallel Model: Pregel (Giraph)

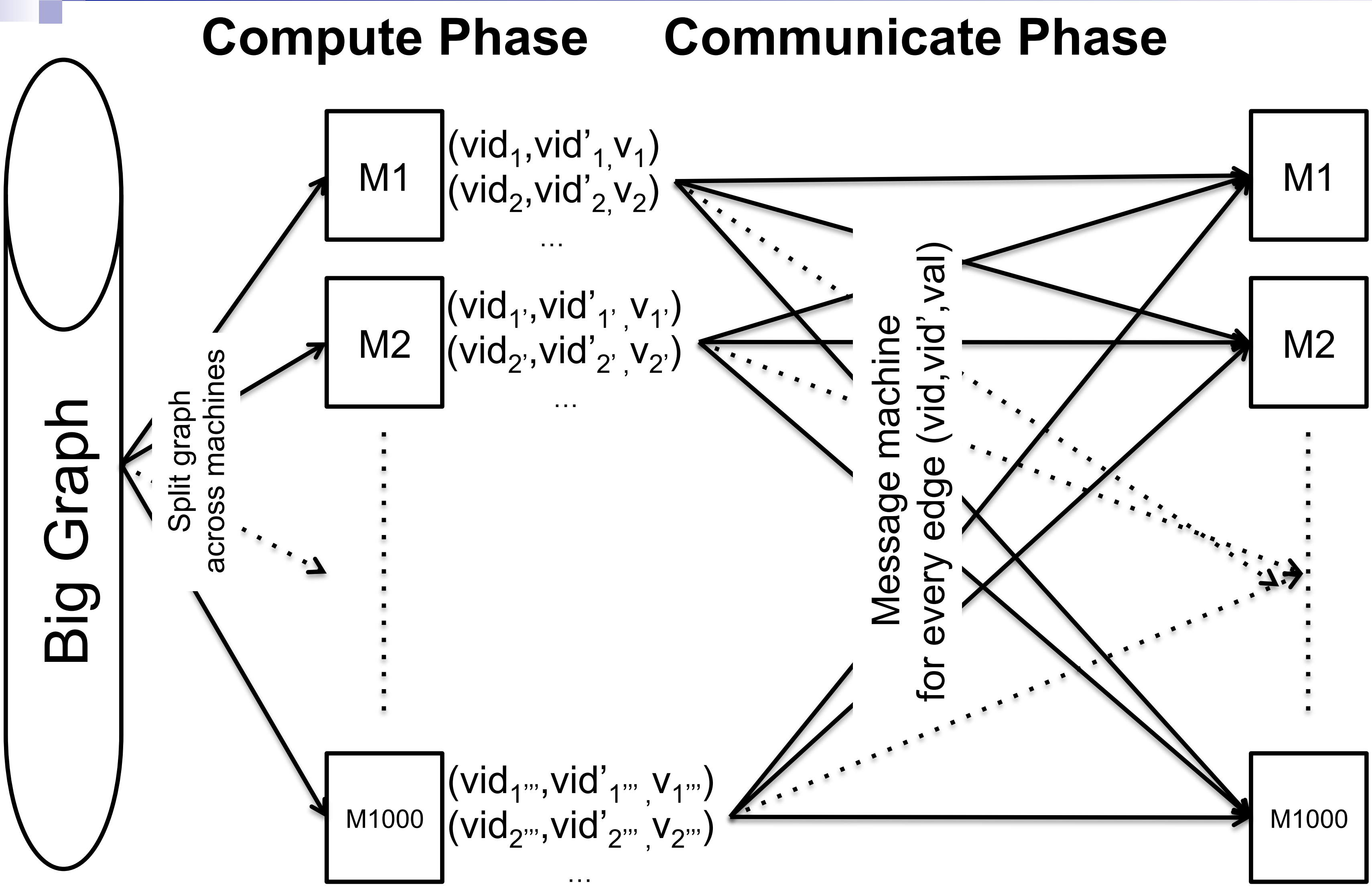
[Valiant '90]



Map-Reduce – Execution Overview



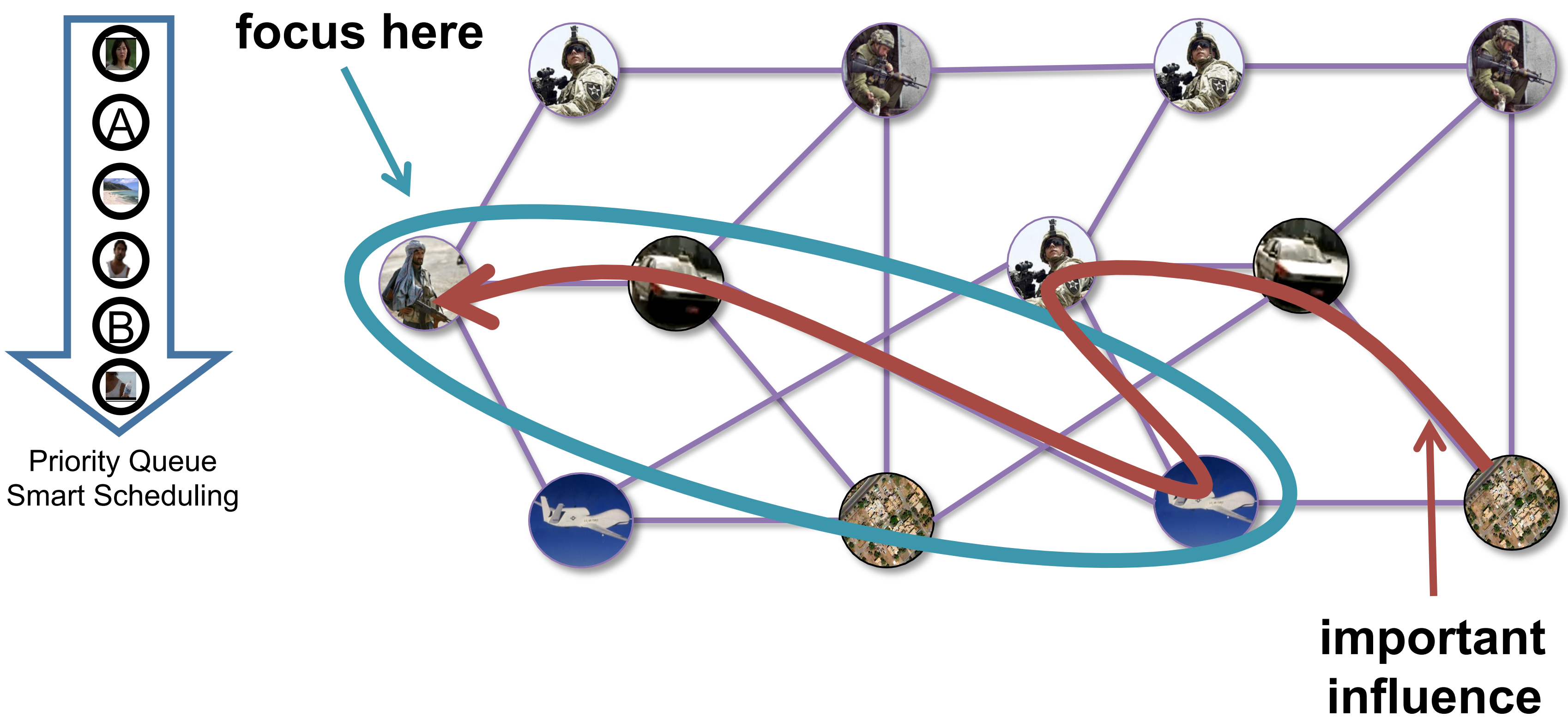
BSP – Execution Overview



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

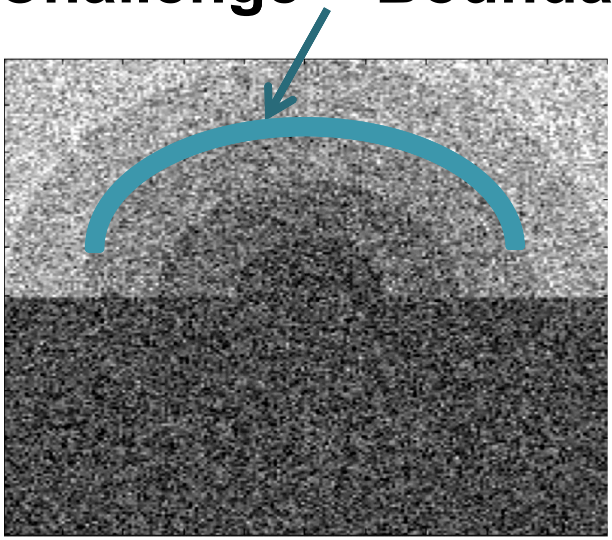
Analyzing Belief Propagation

[Gonzalez, Low, G. '09]

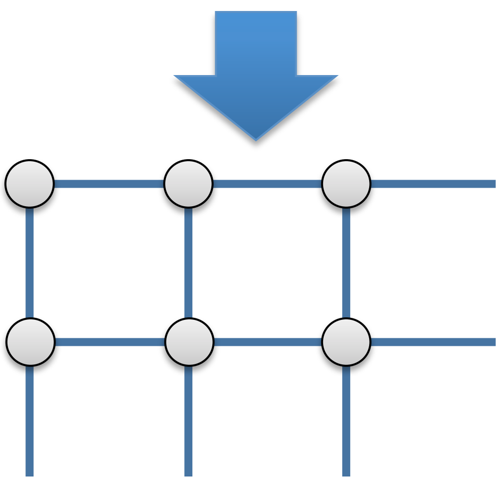


Asynchronous Belief Propagation

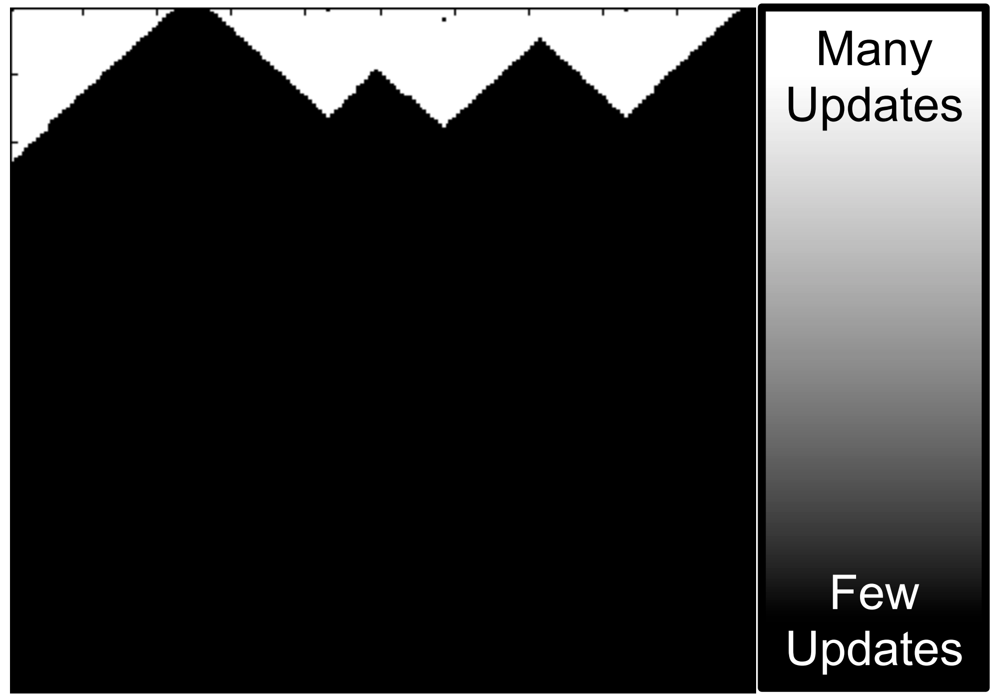
Challenge = Boundaries



Synthetic Noisy Image



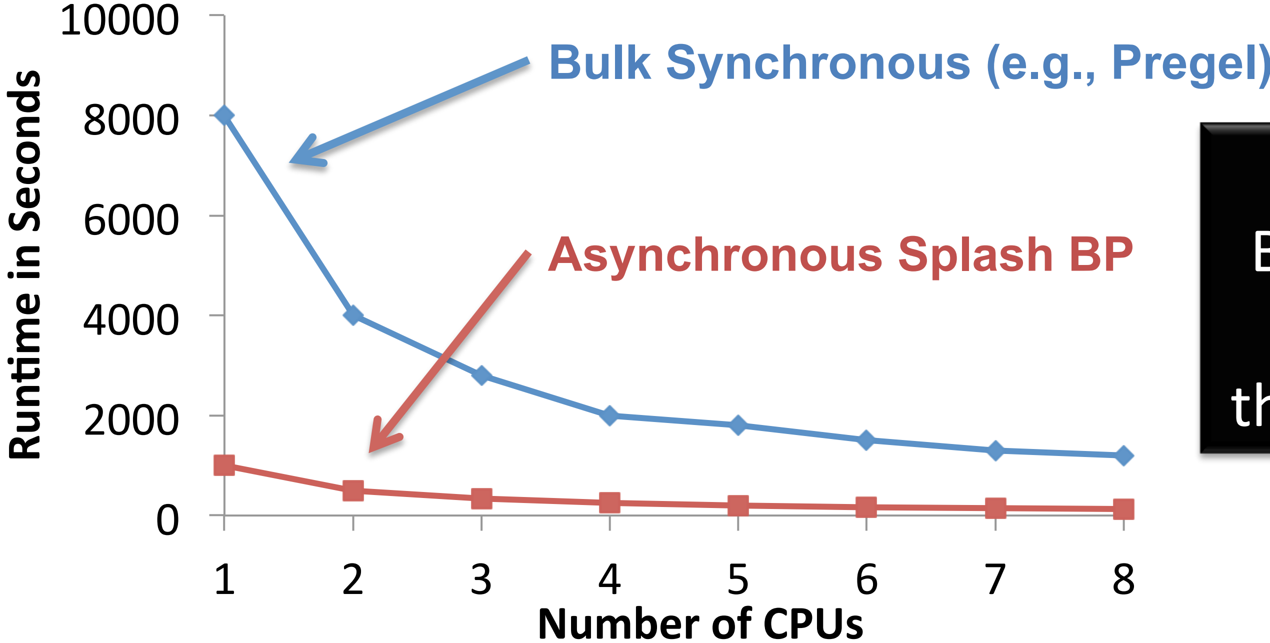
Graphical Model



Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

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



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

Synchronous v. Asynchronous

- Bulk synchronous processing:
  - Computation in phases
    - All vertices participate in a phase
      - Though OK to say no-op
    - All messages are sent
  - Simpler to build, like Map-Reduce
    - No worries about race conditions, barrier guarantees data consistency
    - Simpler to make fault-tolerant, save data on barrier
  - Slower convergence for many ML problems
  - In matrix-land, called Jacobi Iteration
  - Implemented by Google Pregel 2010
- Asynchronous processing:
  - Vertices see latest information from neighbors
    - Most closely related to sequential execution
  - Harder to build:
    - Race conditions can happen all the time
      - Must protect against this issue
    - More complex fault tolerance
    - When are you done?
    - Must implement scheduler over vertices
  - Faster convergence for many ML problems
  - In matrix-land, called Gauss-Seidel Iteration
  - Implemented by GraphLab 2010, 2012

GraphLab

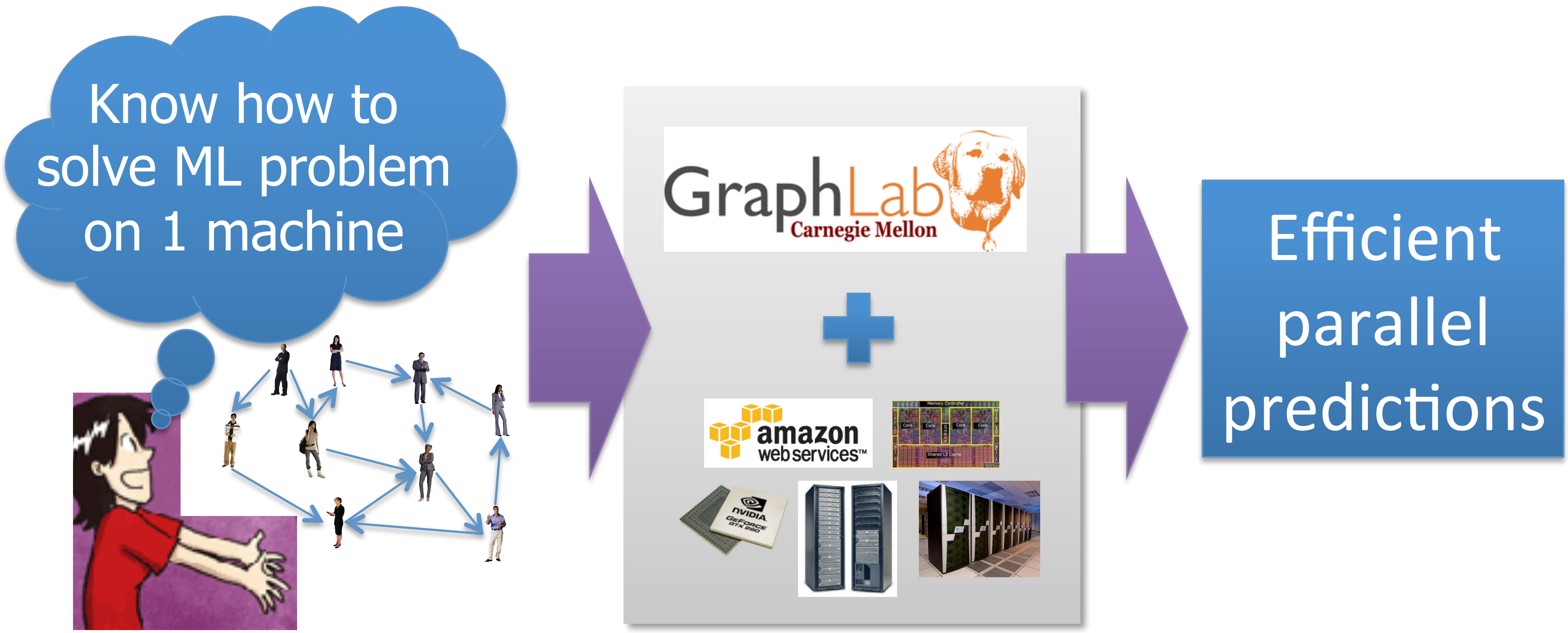
Machine Learning – CSEP546

Carlos Guestrin

University of Washington

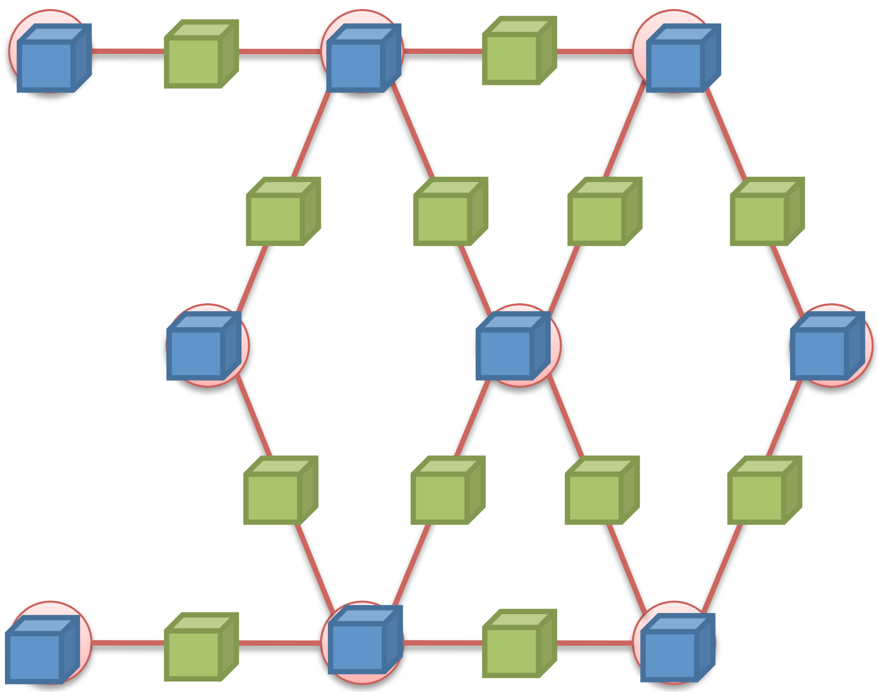
February 24, 2014

The **GraphLab** Goals



Data Graph

Data associated with vertices and edges



Graph:
- Social Network

Vertex Data:
- User profile text
- Current interests estimates

Edge Data:
- Similarity weights

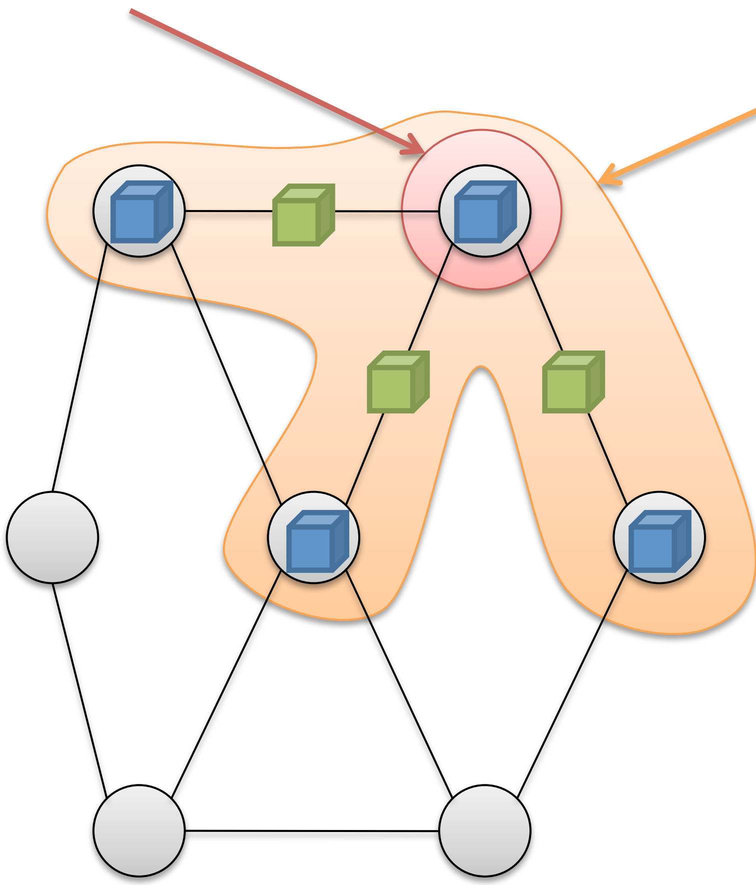
How do we *program* **graph** computation?

"Think like a Vertex."

-Malewicz et al. [SIGMOD'10]

Update Functions

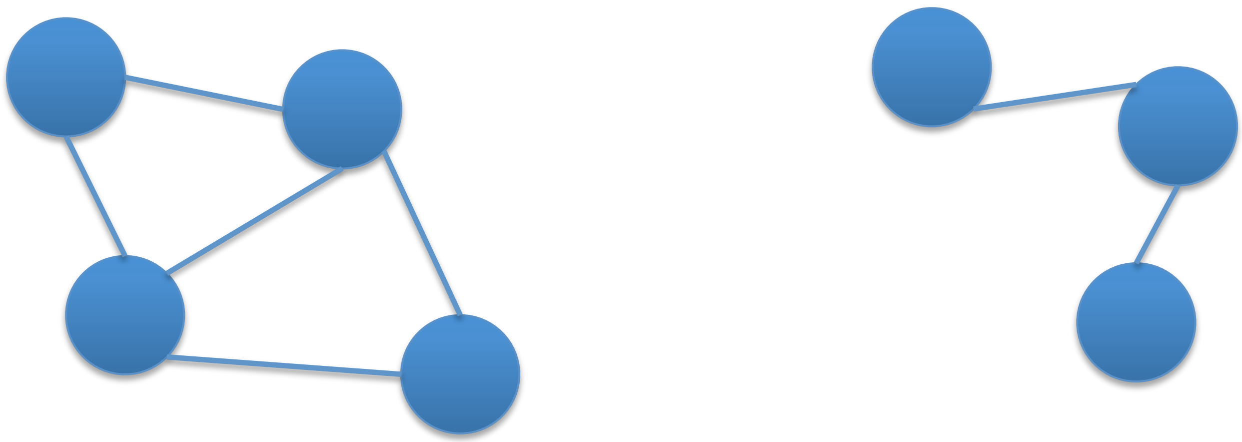
User-defined program: applied to **vertex** transforms data in **scope** of vertex



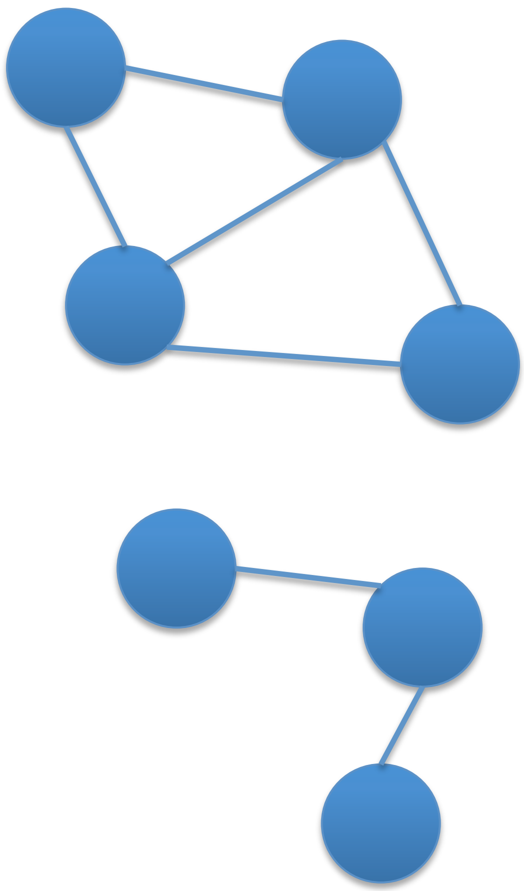
pagerank(i, scope){

}

Update Function Example: Connected Components

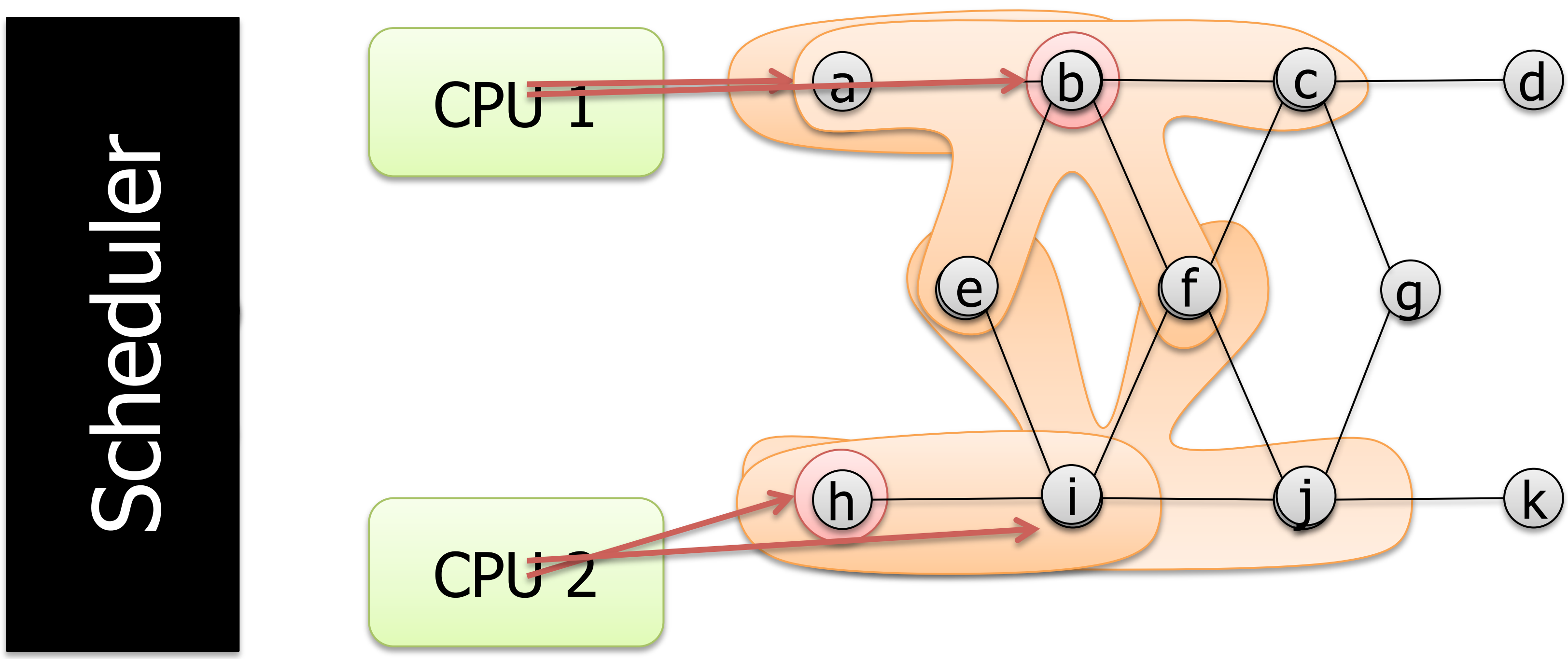


Update Function Example: Connected Components



The Scheduler

The **scheduler** determines order vertices are updated

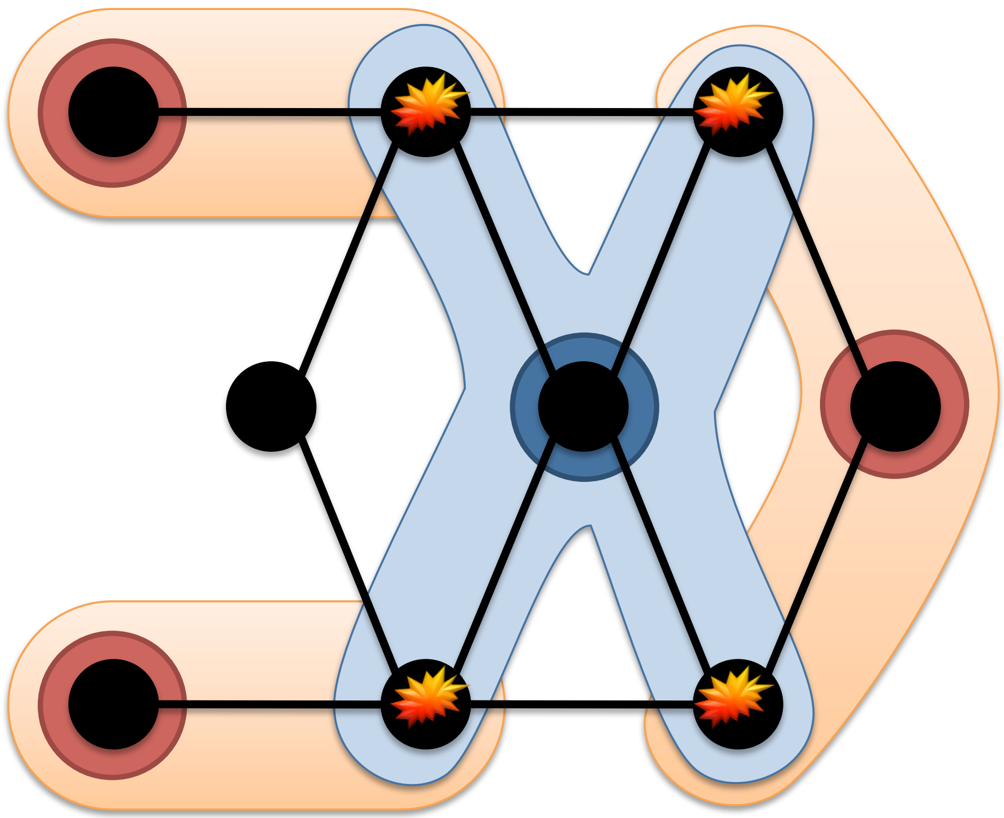


Example Schedulers

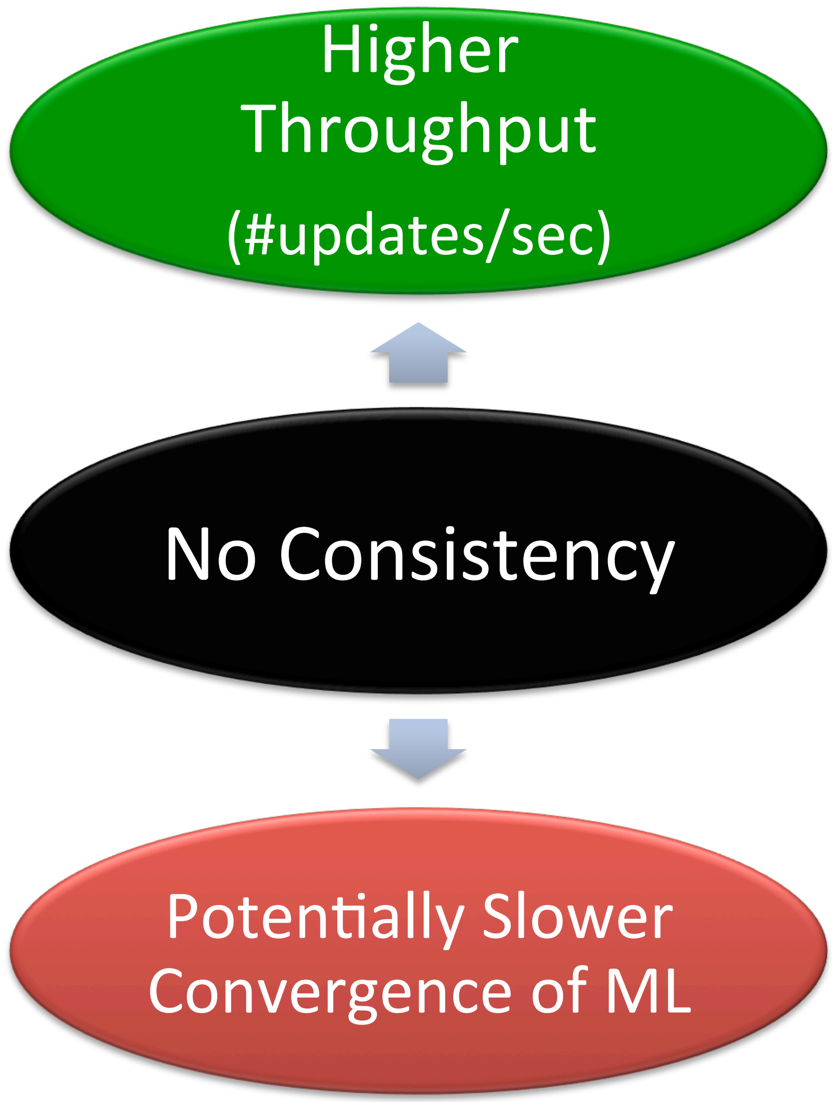
- Round-robin
- Selective scheduling (skipping):
  - round robin but jump over un-scheduled vertice
- FIFO
- Prioritize scheduling
  - Hard to implement in a distributed fashion
    - Approximations used (each machine has its own priority queue)

Ensuring Race-Free Code

How much can computation **overlap**?

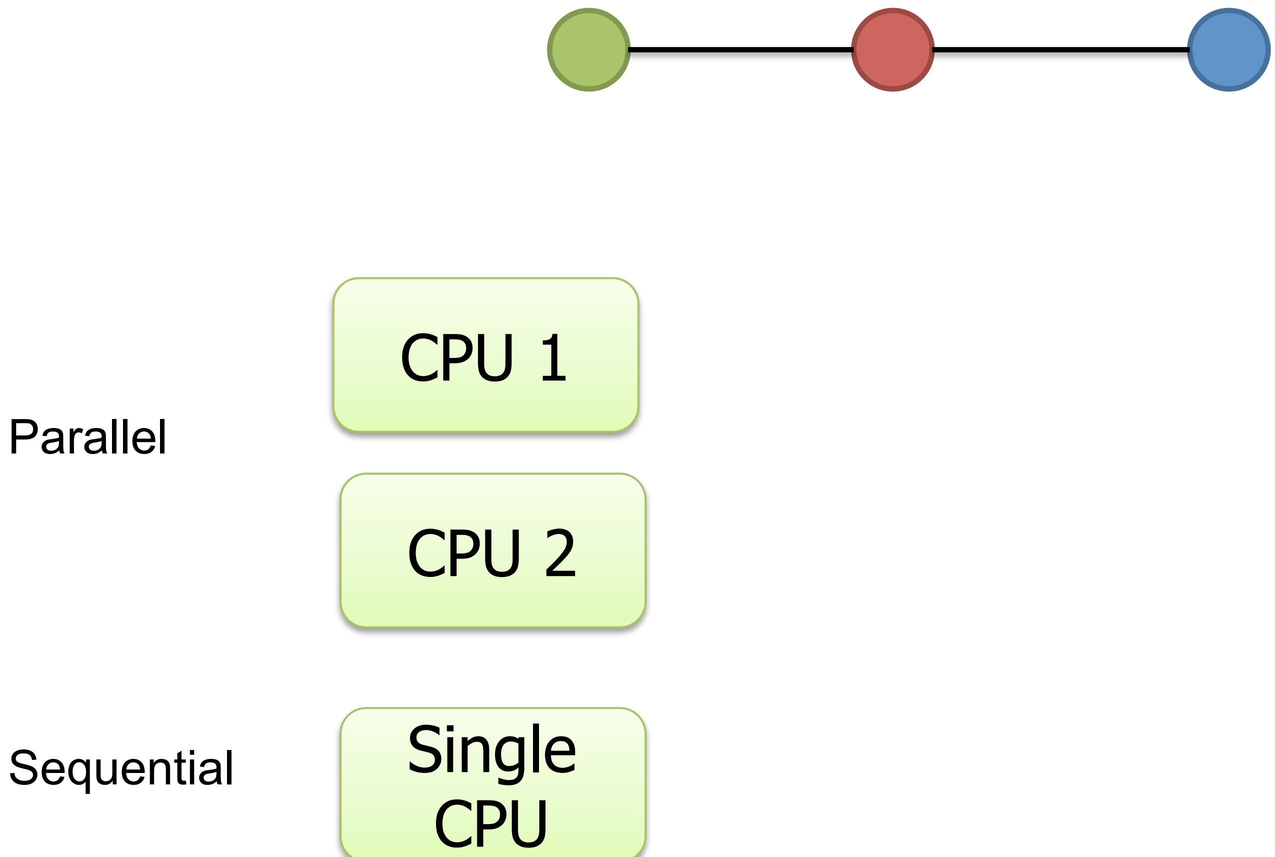


Need for Consistency?

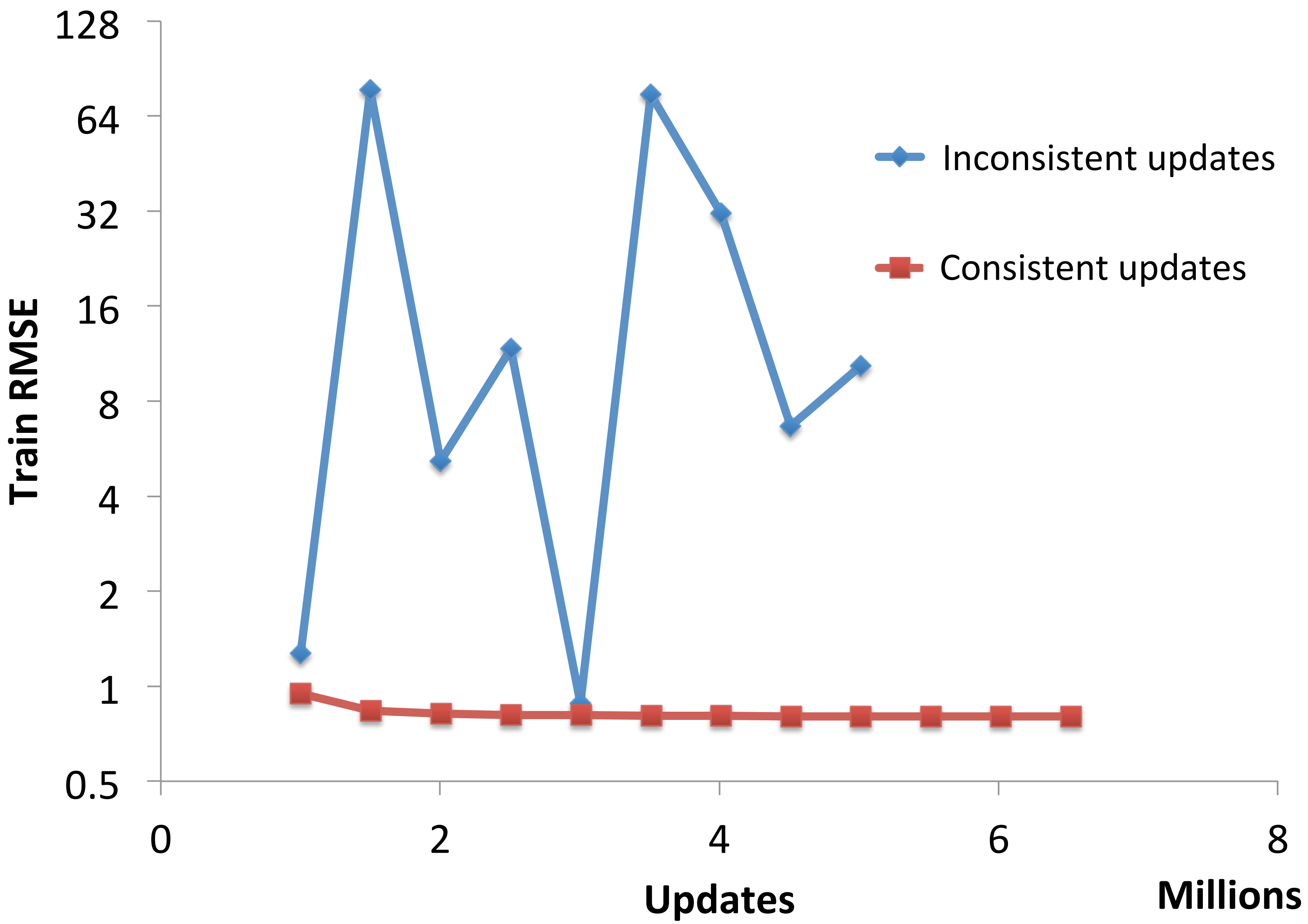


GraphLab Ensures **Sequential Consistency**

For **each parallel execution**, there exists a **sequential execution** of update functions which produces the same result



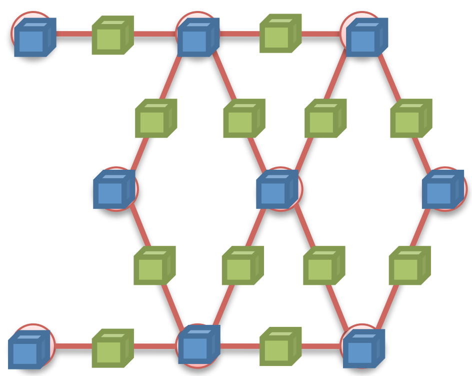
Consistency in Collaborative Filtering



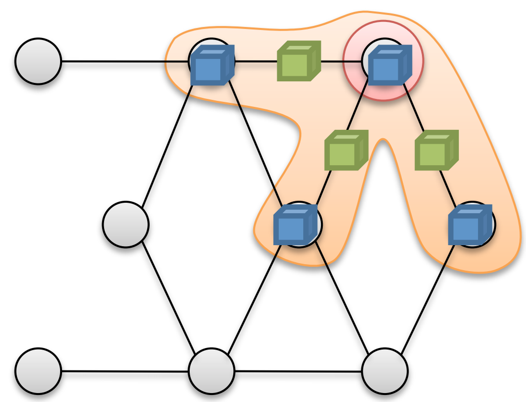
Netflix data, 8 cores

The GraphLab Framework

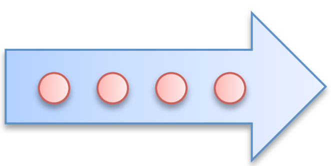
Graph Based
Data Representation



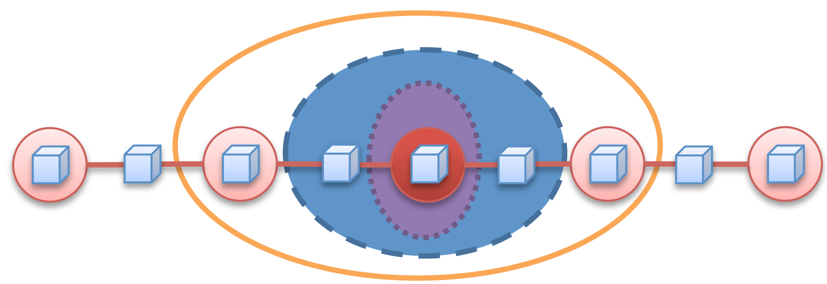
Update Functions
User Computation



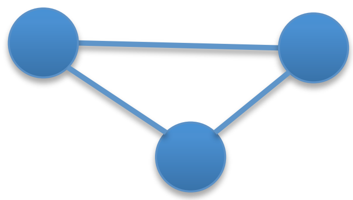
Scheduler



Consistency Model

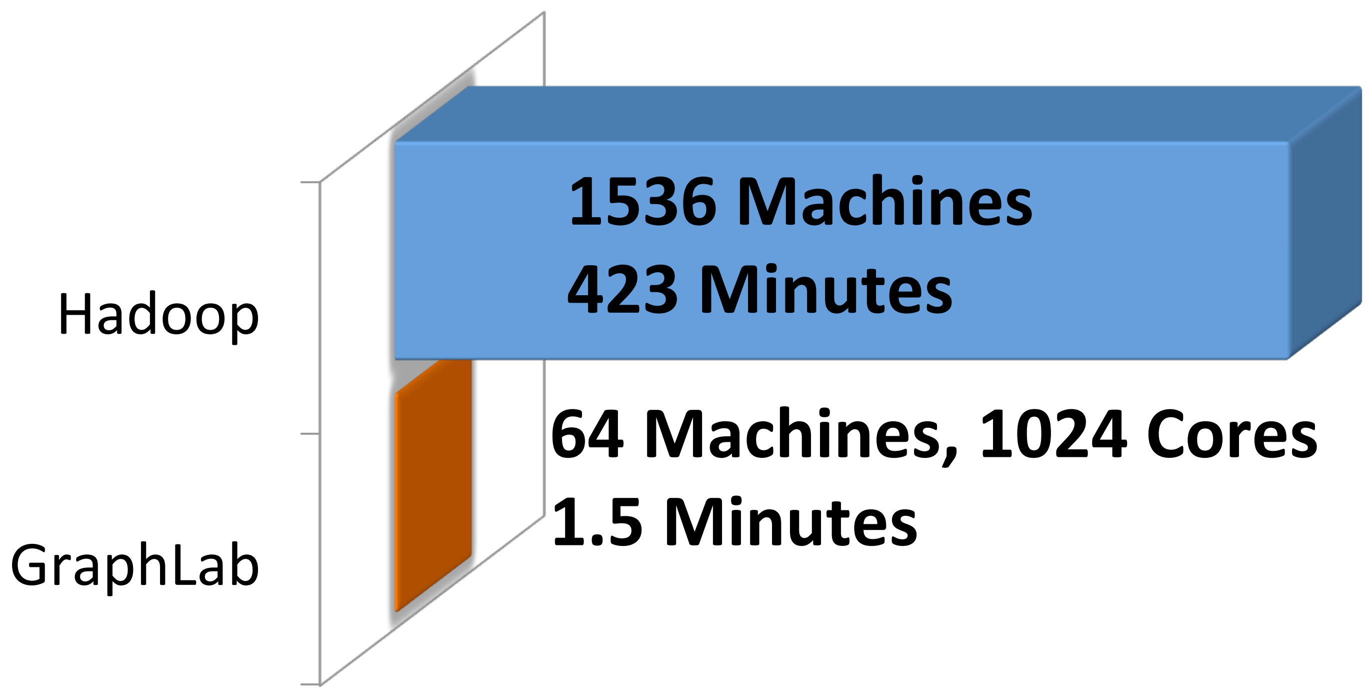


Triangle Counting in Twitter Graph



40M Users
1.2B Edges

Total:
34.8 Billion Triangles

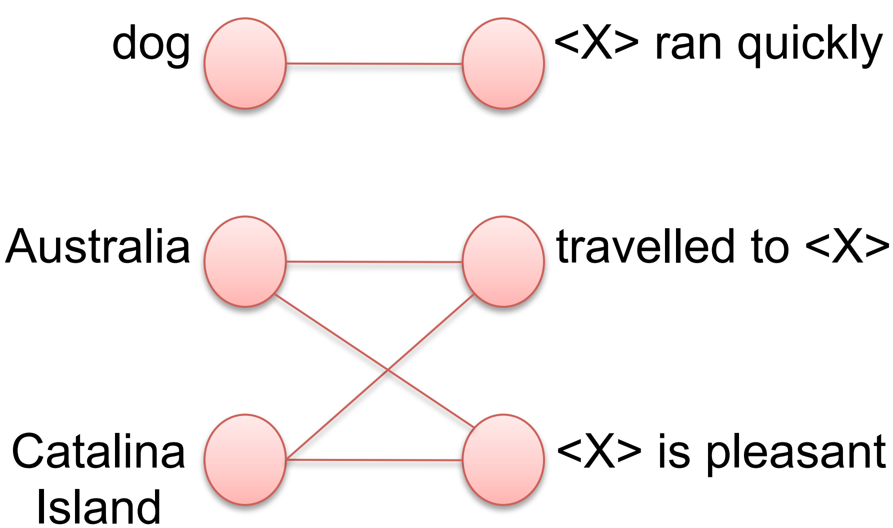


Hadoop results from [Suri & Vassilvitskii '11]

CoEM (Jones et al., 2005)

Named Entity Recognition Task

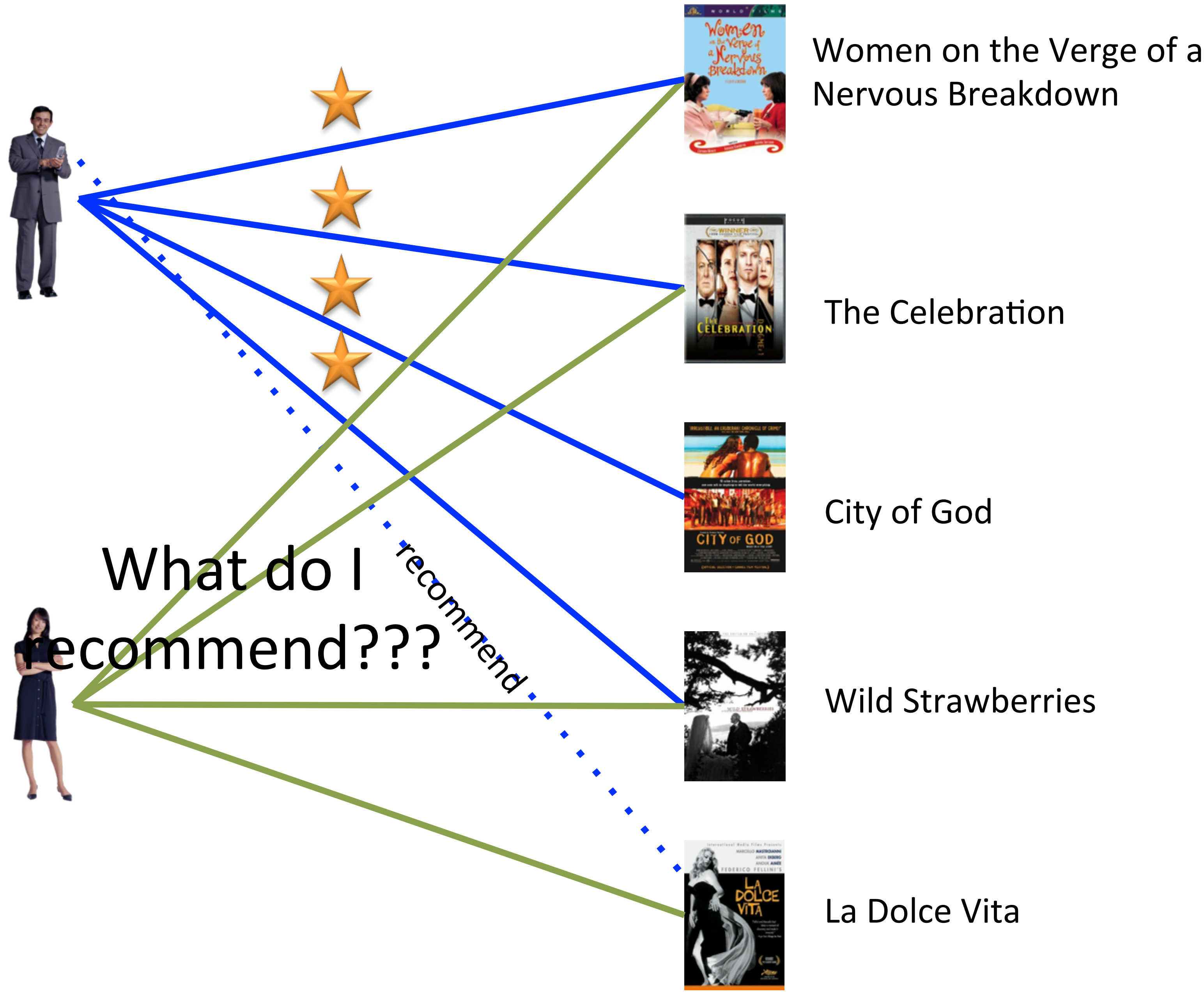
Is “Dog” an animal?
Is “Catalina” a place?



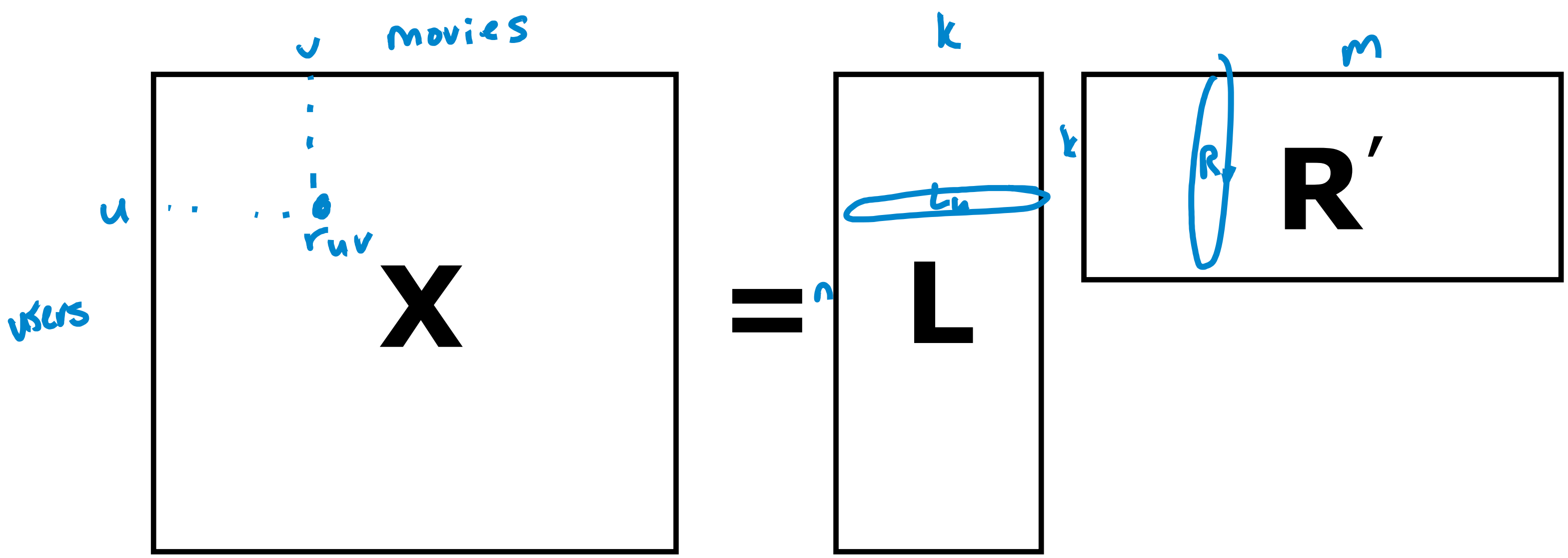
Never Ending Learner Project (CoEM)

Vertices: 2 Million
Edges: 200 Million

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


Women on the Verge of a Nervous Breakdown
The Celebration
City of God
Wild Strawberries
La Dolce Vita
What do I recommend???
recommend

Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)



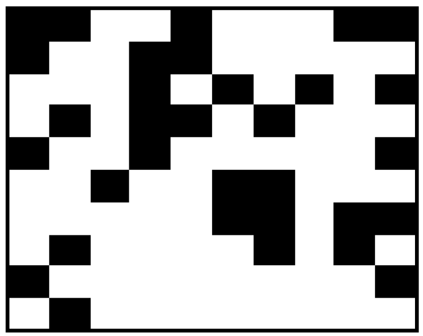
$r_{uv} \approx L_u \cdot R_v$

movie topic i "romance"

how "much" user u likes topic i

how "much" movie v is about topic i

Matrix Completion as a Graph

$\mathbf{X} =$ 

X_{ij} known for black cells

X_{ij} unknown for white cells

Rows index users

Columns index movies

Coordinate Descent for Matrix Factorization: Alternating Least-Squares

$$\min_{L,R} \sum_{(u,v):r_{uv}\neq ?} (L_u \cdot R_v - r_{uv})^2$$

- Fix movie factors, optimize for user factors
  - Independent least-squares over users

$$\min_{L_u} \sum_{v\in V_u} (L_u \cdot R_v - r_{uv})^2$$

- Fix user factors, optimize for movie factors
  - Independent least-squares over movies

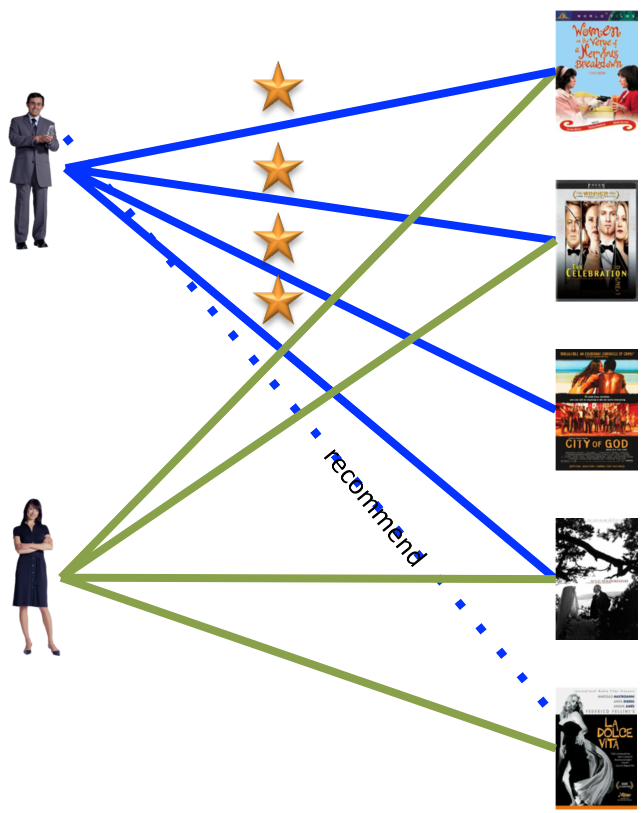
$$\min_{R_v} \sum_{u\in U_v} (L_u \cdot R_v - r_{uv})^2$$

- System may be underdetermined:

- Converges to

Alternating Least Squares Update Function

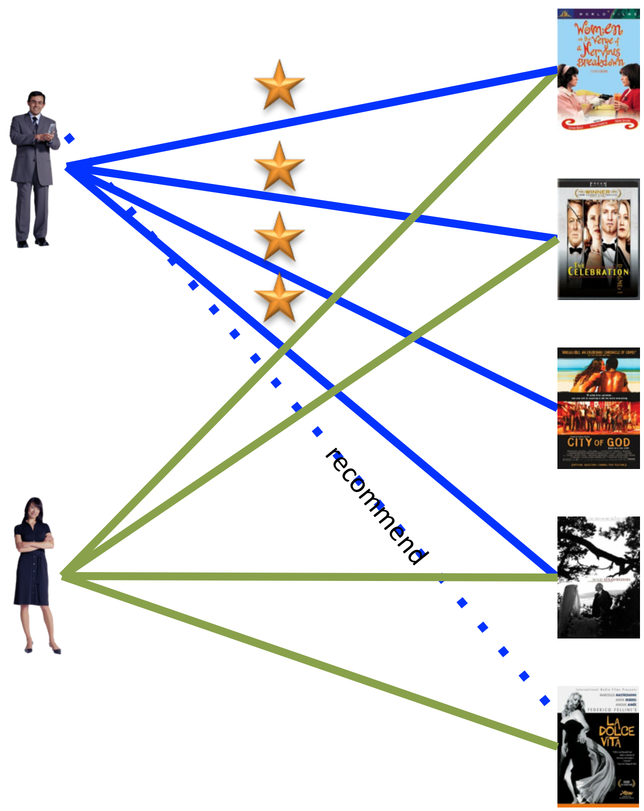
$$\min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 \qquad \min_{R_v} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2$$



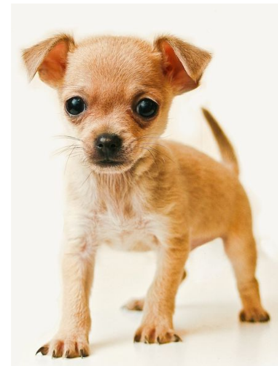
SGD for Matrix Factorization in Map-Reduce?

$$\epsilon_t = L_u^{(t)} \cdot R_v^{(t)} - r_{uv}$$

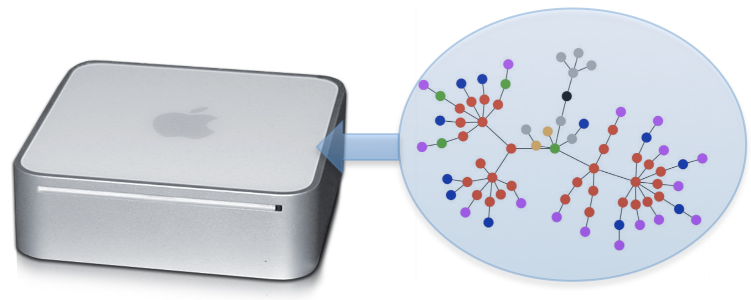
$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1 - \eta_t \lambda_u) L_u^{(t)} - \eta_t \epsilon_t R_v^{(t)} \\ (1 - \eta_t \lambda_v) R_v^{(t)} - \eta_t \epsilon_t L_u^{(t)} \end{bmatrix}$$



GraphChi: Going small with GraphLab



Solve huge problems on small or embedded devices?

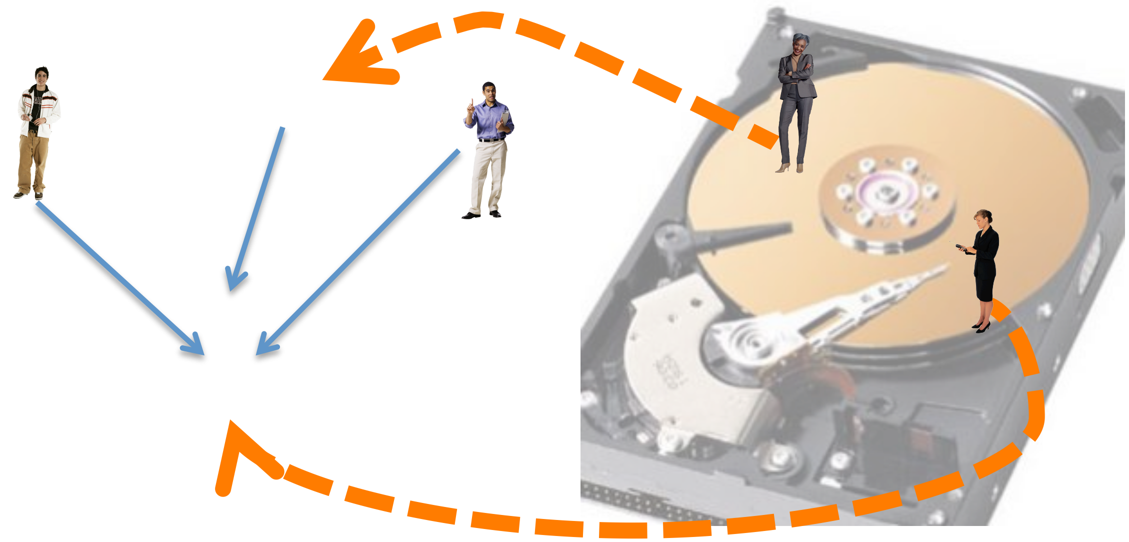


Key: Exploit non-volatile memory (starting with SSDs and HDs)

GraphChi – disk-based GraphLab

Challenge:

Random Accesses

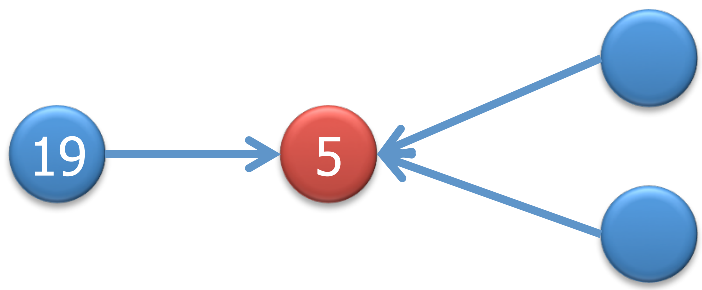


Novel GraphChi solution:

Parallel sliding windows method ➔

minimizes number of random accesses

Naive Graph Disk Layouts



- Symmetrized adjacency file with values,

vertex	in-neighbors	out-neighbors
5	**3**:2.3, **19**: 1.3, **49**: 0.65,…	**781**: 2.3, **881**: 4.2..
….		
19	**3**: 1.4, **9**: 12.1, …	**5**: 1.3, 28: 2.2, …

synchronize

Random write

- … or with file index pointers

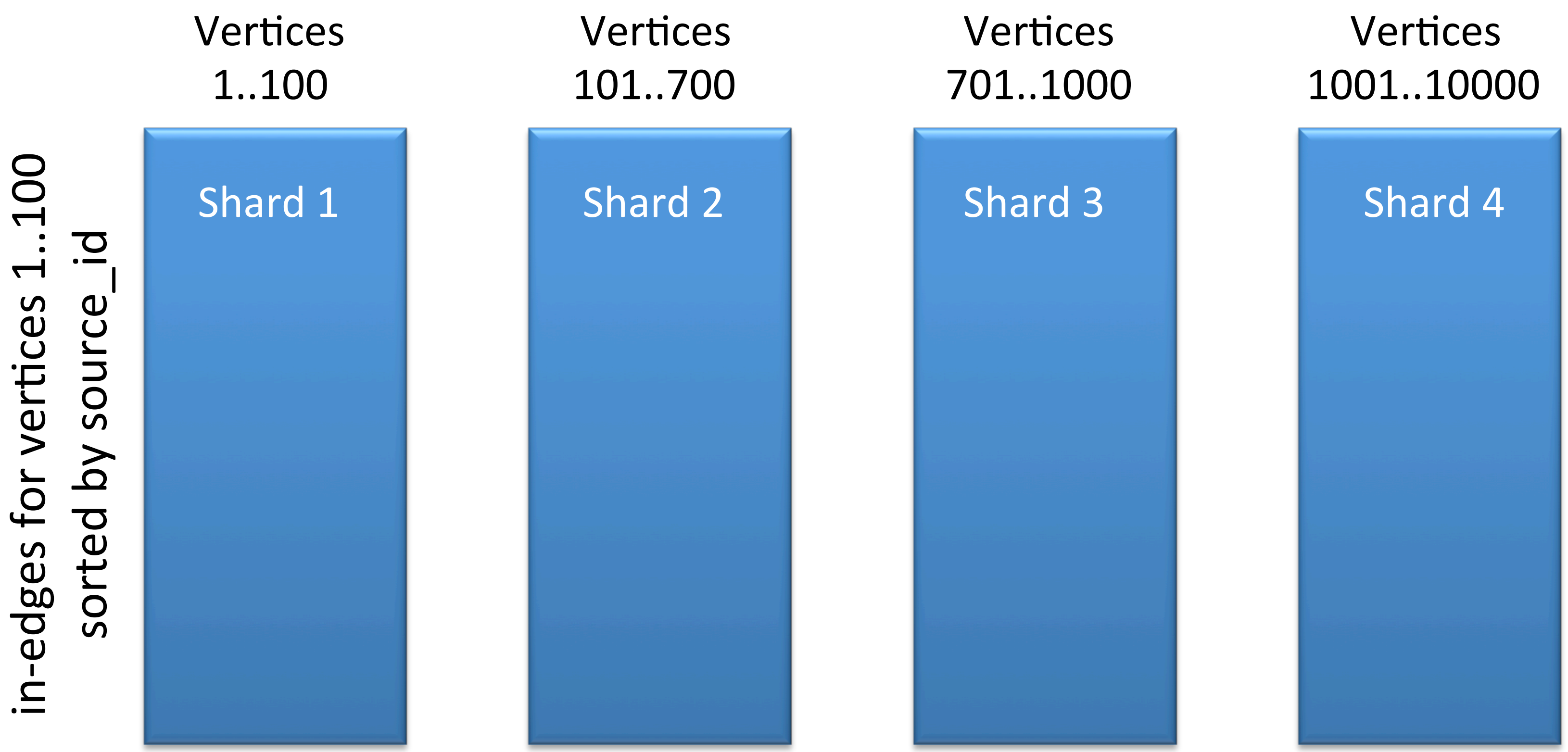
vertex	in-neighbor-ptr	out-neighbors
5	**3**: 881, **19**: 10092, **49**: 20763,…	**781**: 2.3, **881**: 4.2..
….		
19	**3**: 882, **9**: 2872, …	**5**: 1.3, 28: 2.2, …

read

Random read/write

Parallel Sliding Windows Layout

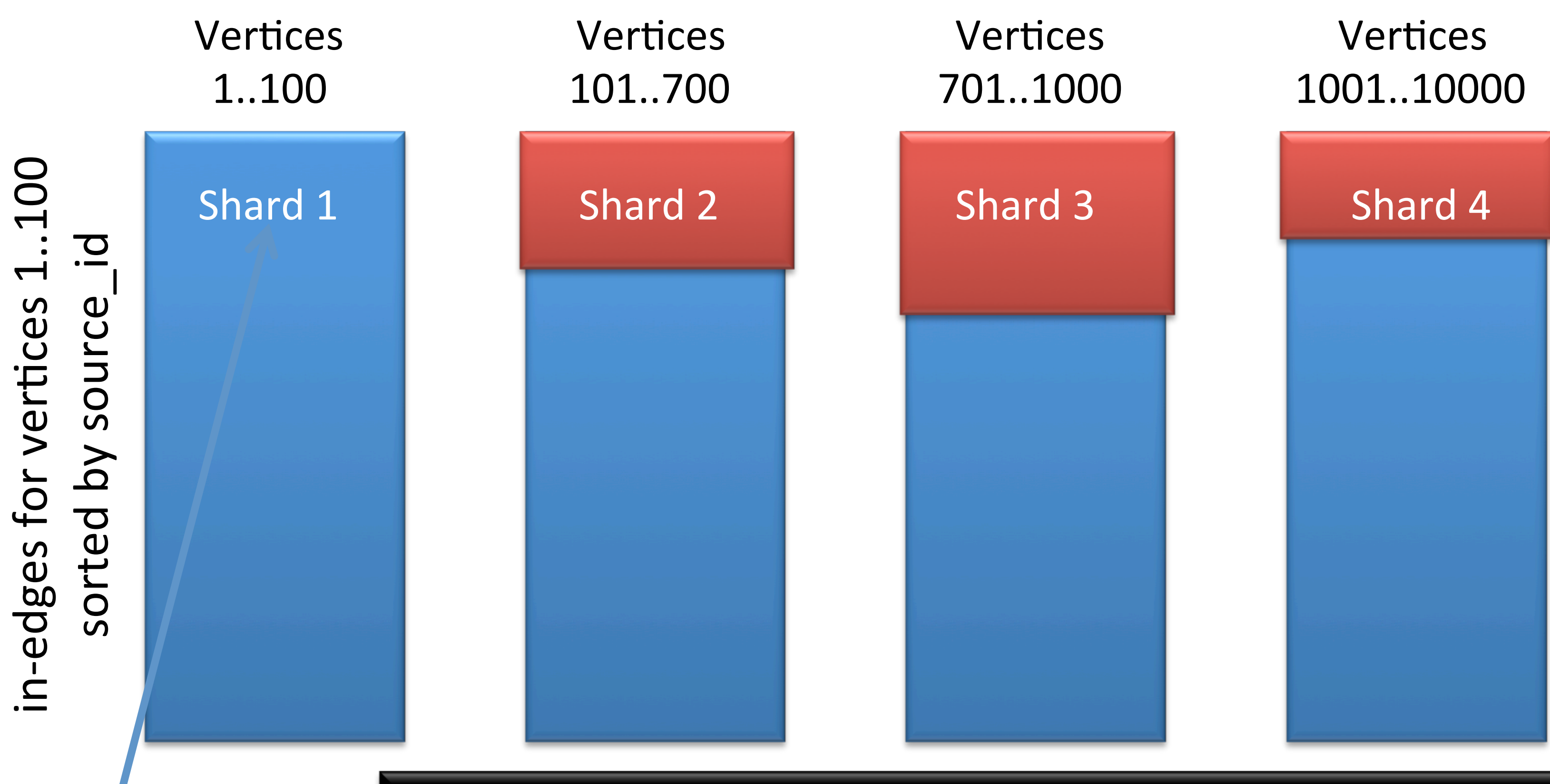
Shard: in-edges for subset of vertices; sorted by source_id



Shards small enough to fit in memory; balance size of shards

Parallel Sliding Windows Execution

Load subgraph for vertices 1..100



Load all in-edges in memory

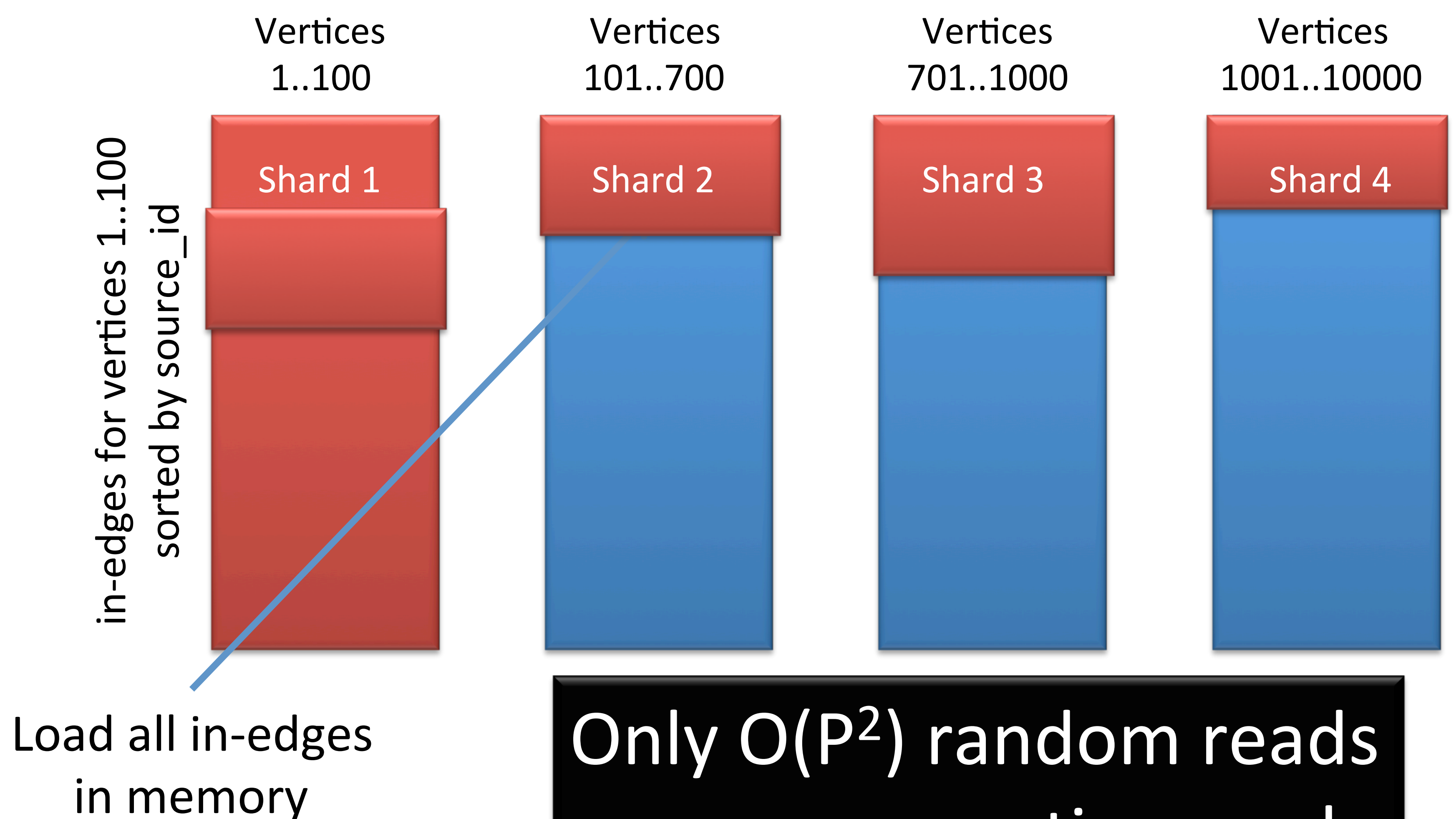
What about out-edges?

Arranged in sequence in other shards!

And sequential writes!

Parallel Sliding Windows Execution

Load subgraph for vertices 101..700

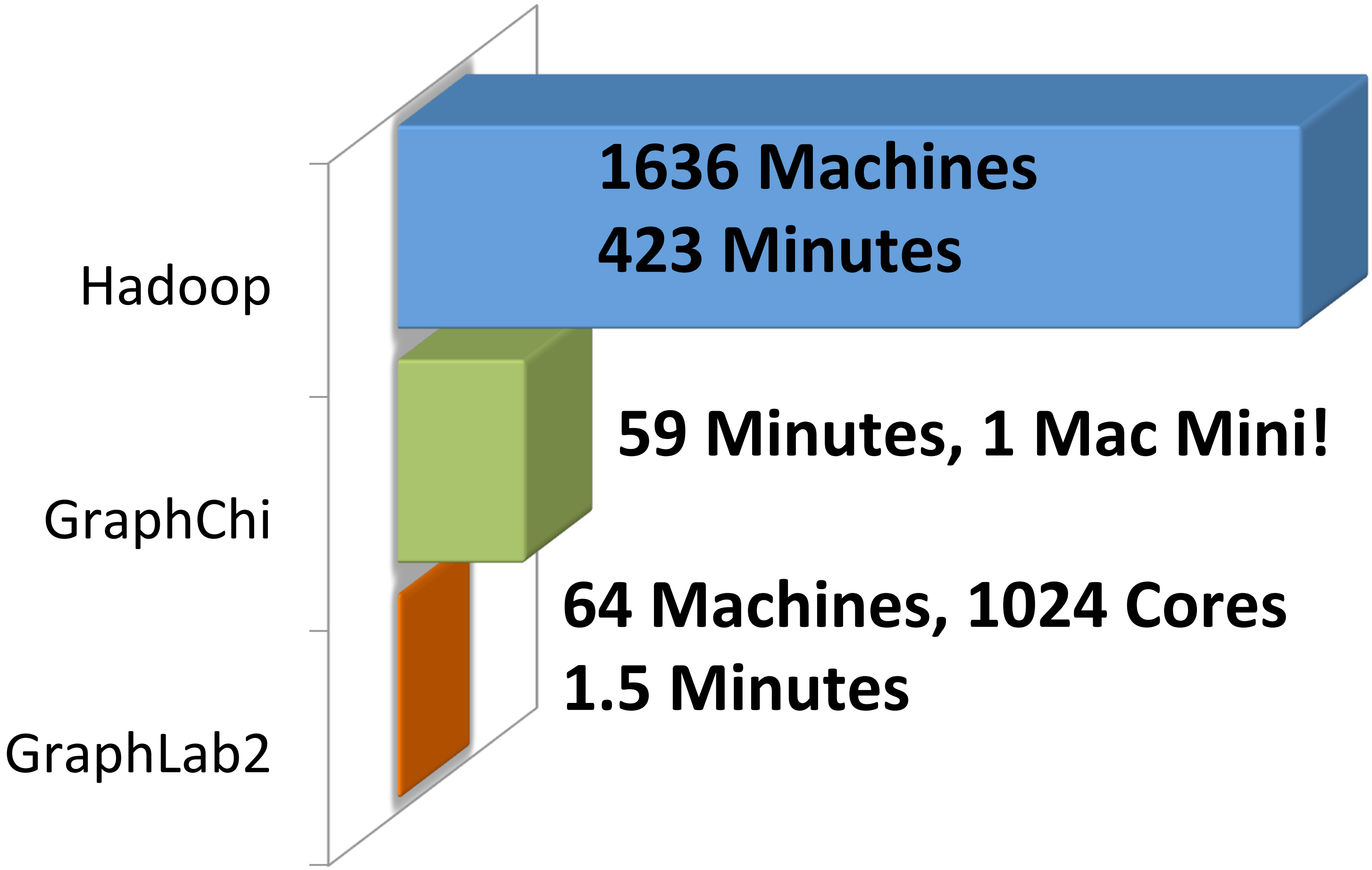


Only $O(P^2)$ random reads per pass on entire graph

Triangle Counting on Twitter Graph

40M Users
1.2B Edges

Total: 34.8 Billion Triangles



Hadoop results from [Suri & Vassilvitskii '11]

GraphLab

Release 2.2 available now

http://graphlab.org

Documentation… Code… Tutorials… (more on the way)

GraphChi 0.1 available now

http://graphchi.org

What you need to know…

- Data-parallel versus graph-parallel computation
- Bulk synchronous processing versus asynchronous processing
- GraphLab system for graph-parallel computation
  - Data representation
  - Update functions
  - Scheduling
  - Consistency model