Large Graph Processing

Institute of Parallel and Distributed Systems

Rong Chen

*Slides adapted from Joseph Gonzalez

What we've learnt so far

- Data-parallel Programming Model
  - MapReduce & Dryad
    - Parallelization
    - Load balance
    - Locality and Data Distribution
    - Fault Tolerance
    - …
- This class:
  - Large graph processing

Big Data already Happened

1 Billion Tweets
Per Week

750 Million
Facebook Users

6 Billion
Flickr Photos

48 Hours of Video
Uploaded every Minute

How do we
understand and **use**
Big Data?

Big Learning

Big Learning Today: **Simple Models**

- Pros:
  - Easy to understand/predictable
  - Easy to train in parallel
  - Supports **Feature Engineering**
  - Versatile: classification, ranking, density estimation

Philosophy of Big Data and Simple Models

*"Invariably, **simple models** and **a lot of data** trump more elaborate models based on less data."*

Alon Halevy, Peter Norvig, and
Fernando Pereira, Google

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/35179.pdf

*"Invariably, **simple models** and **a lot of data** trump more elaborate models **based on less data**."*

Alon Halevy, Peter Norvig, and
Fernando Pereira, Google

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/35179.pdf

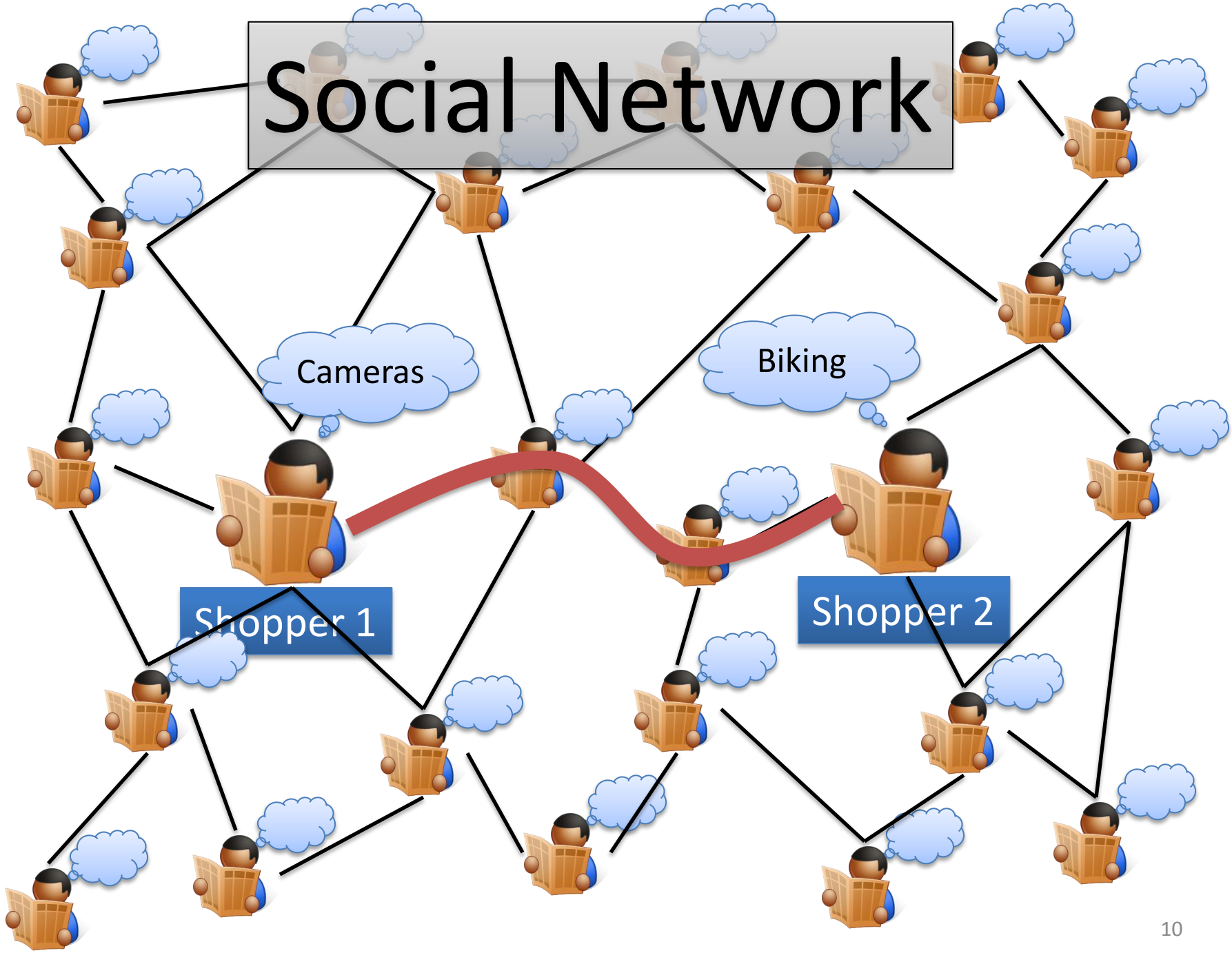
Why not build **elaborate models** with **lots of data**?

Difficult

Computationally Intensive

Big Learning Today: **Simple Models**

- Pros:
  - Easy to understand/predictable
  - Easy to train in parallel
  - Supports **Feature Engineering**
  - Versatile: classification, ranking, density estimation
- Cons:
  - Favors **bias** in the presence of **Big Data**
  - Strong **independence assumptions**


Social Network
Cameras
Biking
Shopper 1
Shopper 2

Big Data exposes the opportunity for ***structured*** *machine learning*

Examples

Label Propagation

- Social Arithmetic:

  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

  I Like: 60% Cameras, 40% Biking

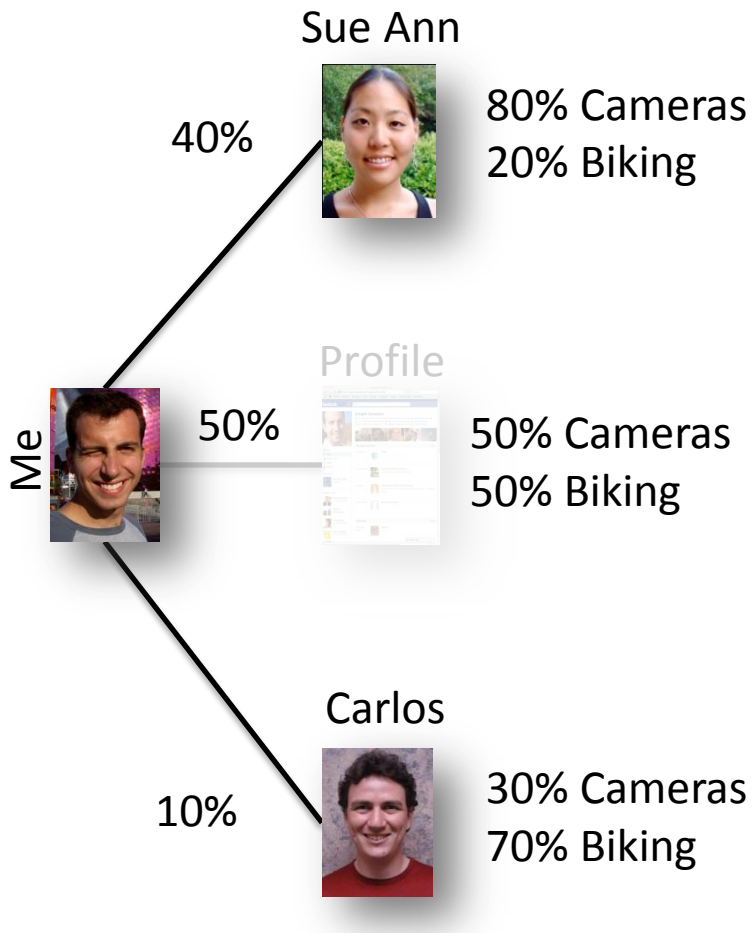
- Recurrence Algorithm:

$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

  – iterate until convergence

- Parallelism:

  – Compute all *Likes[i]* in parallel



http://www.cs.cmu.edu/~zhuxj/pub/CMU-CALD-02-107.pdf

PageRank (Centrality Measures)

- Iterate:

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

- Where:
  - α is the random reset probability
  - $L[j]$ is the number of links on page j

$$R[5] = \alpha + (1 - \alpha) \left(\frac{1}{3} R[1] + \frac{1}{1} R[4] \right)$$

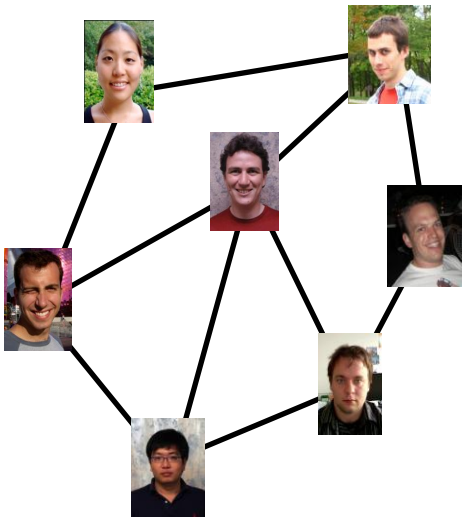
http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

Other Examples

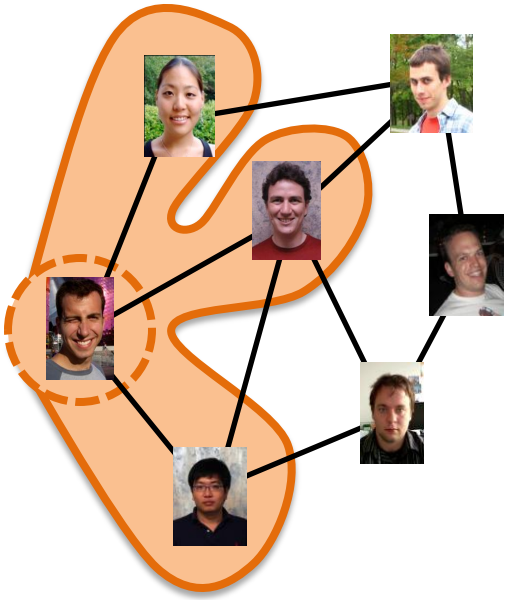
- Statistical Inference in Relational Models
  - Belief Propagation
  - Gibbs Sampling
- Network Analysis
  - Centrality Measures
  - Triangle Counting
- Natural Language Processing
  - CoEM
  - Topic Modeling

Graph Parallel Algorithms

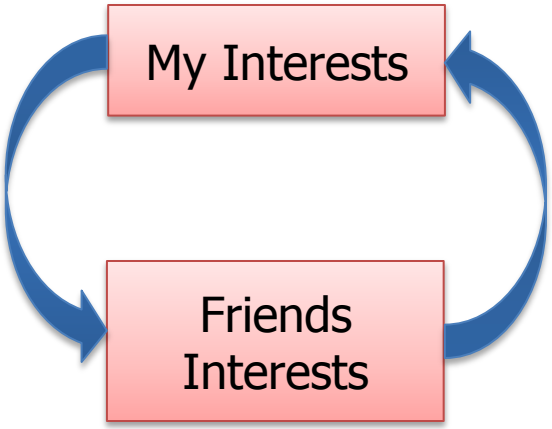
Dependency **Graph**



Local Updates



Iterative Computation



What is the right tool for Graph-Parallel ML

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Map Reduce?

Lasso

Label Propagation

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

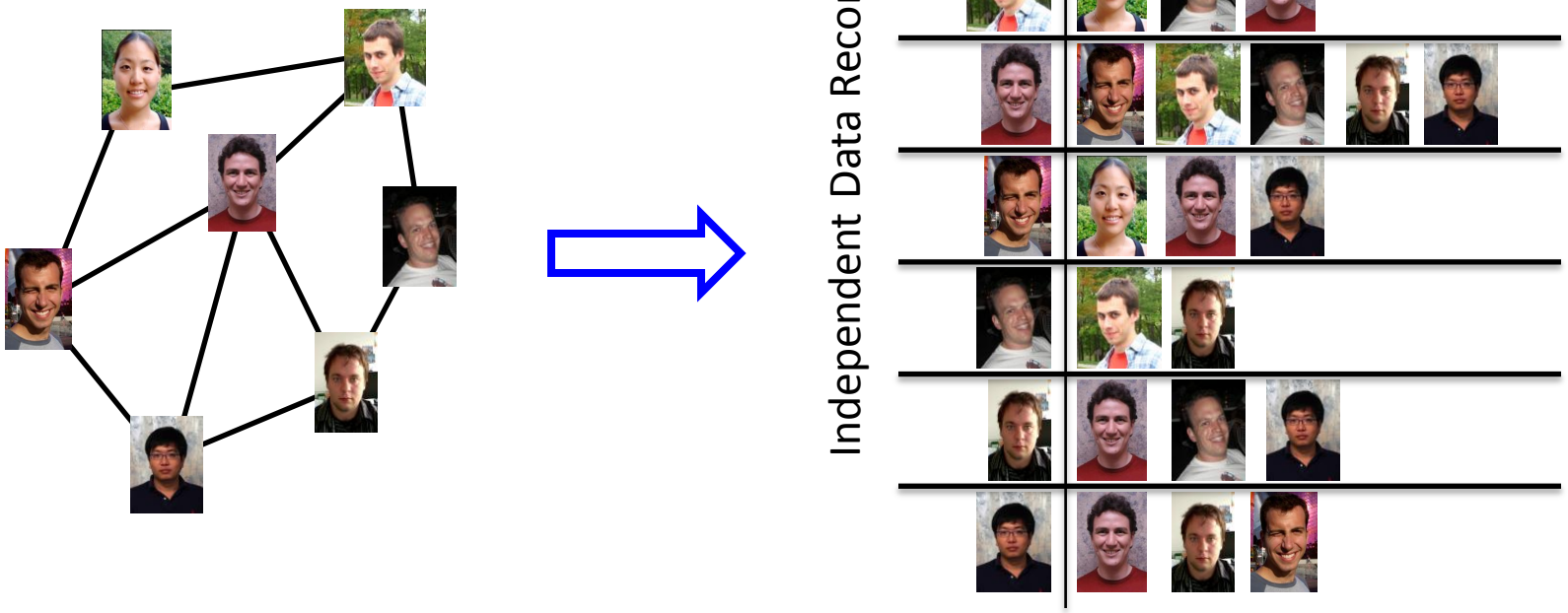
Deep Belief Networks

Neural Networks

Why not use *Map-Reduce*
for
Graph Parallel algorithms?

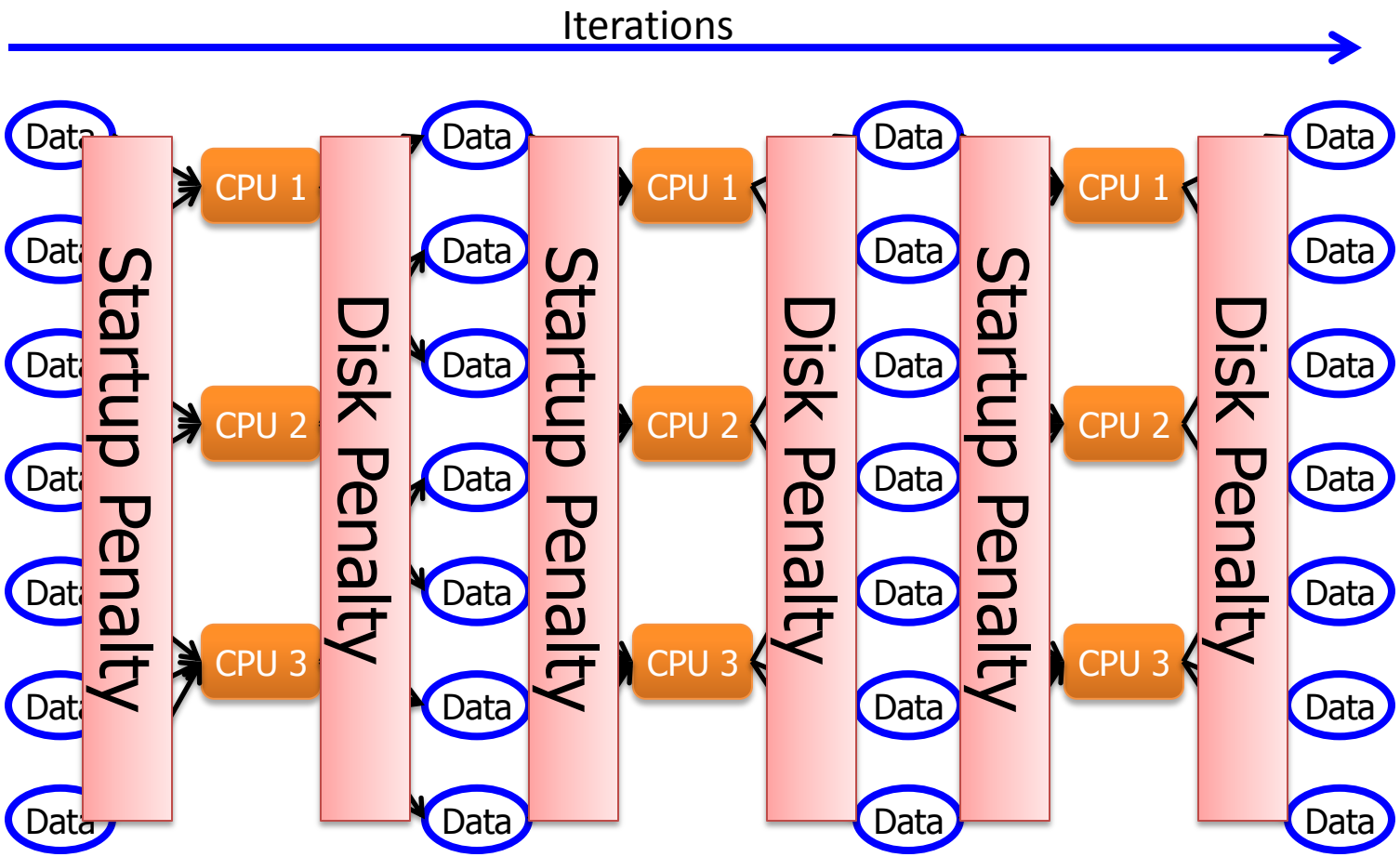
Data Dependencies are Difficult

- Difficult to express dependent data in MR
  - Substantial data transformations
  - User managed graph structure
  - Costly data replication



Iterative Computation is Difficult

- System is not optimized for iteration:



Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

MPI/Pthreads

Lasso

SVM

Belief Propagation

Kernel Methods

Tensor Factorization

PageRank

Deep Belief Networks

Neural Networks

We could use ….

Threads, Locks, & Messages

“low level parallel primitives”

Threads, Locks, and Messages

- Graduate students **repeatedly** solve the same parallel design challenges:
  - Implement and debug complex parallel system
  - Tune for a specific parallel platform
  - Six months later the conference paper contains:

"We implemented _____ in parallel."

- The resulting code:
  - is difficult to maintain
  - is difficult to extend
    - couples learning model to parallel implementation

Addressing Graph-Parallel ML

- We need alternatives to Map-Reduce

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (BSP)

Lasso

SVM

Kernel Methods

Belief Propagation

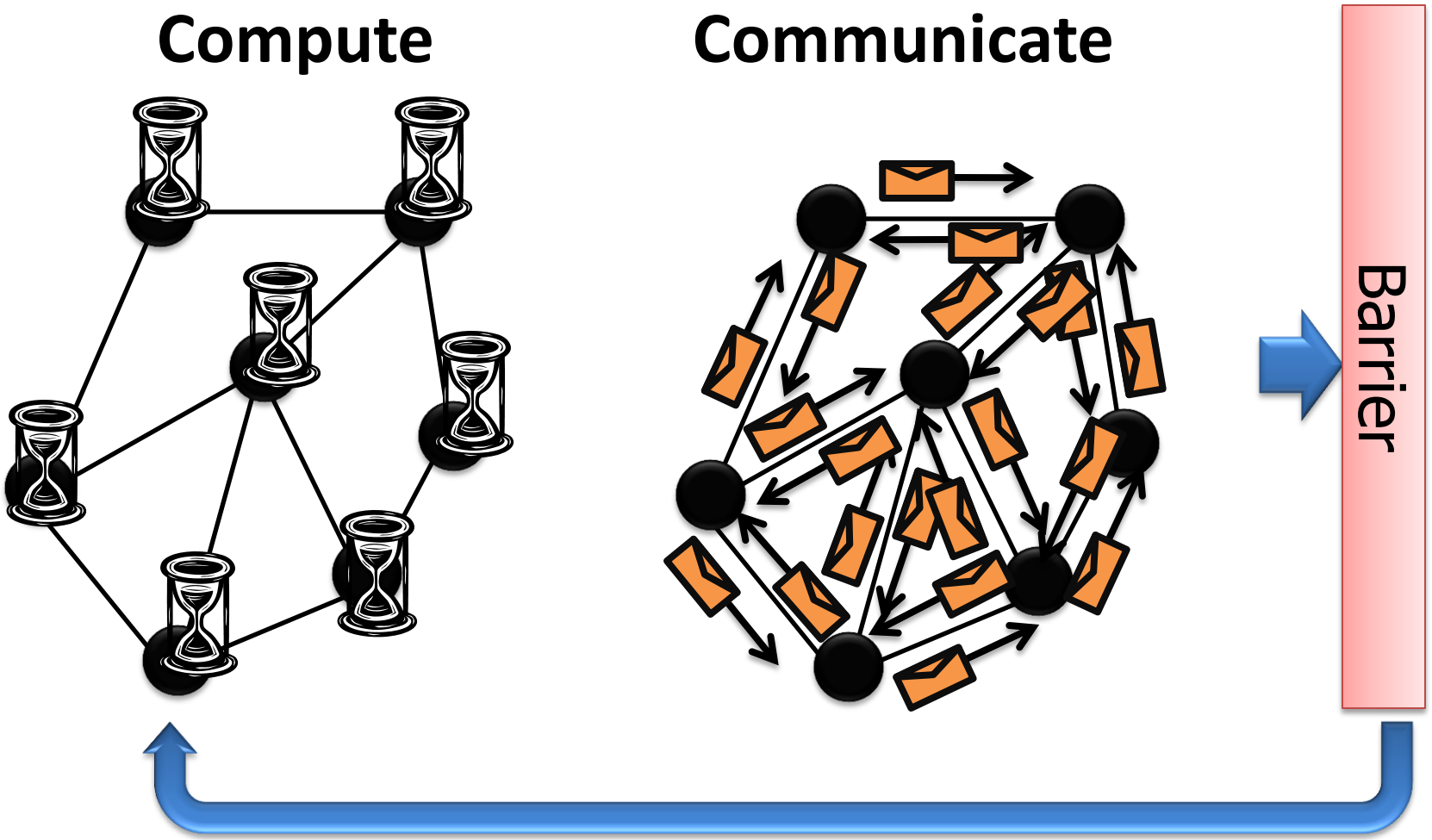
Tensor Factorization

PageRank

Deep Belief Networks

Neural Networks

Pregel: Bulk Synchronous Parallel



http://dl.acm.org/citation.cfm?id=1807184

Open Source Implementations

- Giraph: http://incubator.apache.org/giraph/
- Golden Orb: http://goldenorbos.org/

An asynchronous variant:

- GraphLab: http://graphlab.org/

PageRank in Giraph (Pregel)

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

```
public void compute(Iterator<DoubleWritable> msgIterator) {
  double sum = 0;
  while (msgIterator.hasNext())
    sum += msgIterator.next().get();
  DoubleWritable vertexValue =
    new DoubleWritable(0.15 + 0.85 * sum);
  setVertexValue(vertexValue);

  if (getSuperstep() < getConf().getInt(MAX_STEPS, -1)) {
    long edges = getOutEdgeMap().size();
    sentMsgToAllEdges(
      new DoubleWritable(getVertexValue().get() / edges));
  } else voteToHalt();
}
```

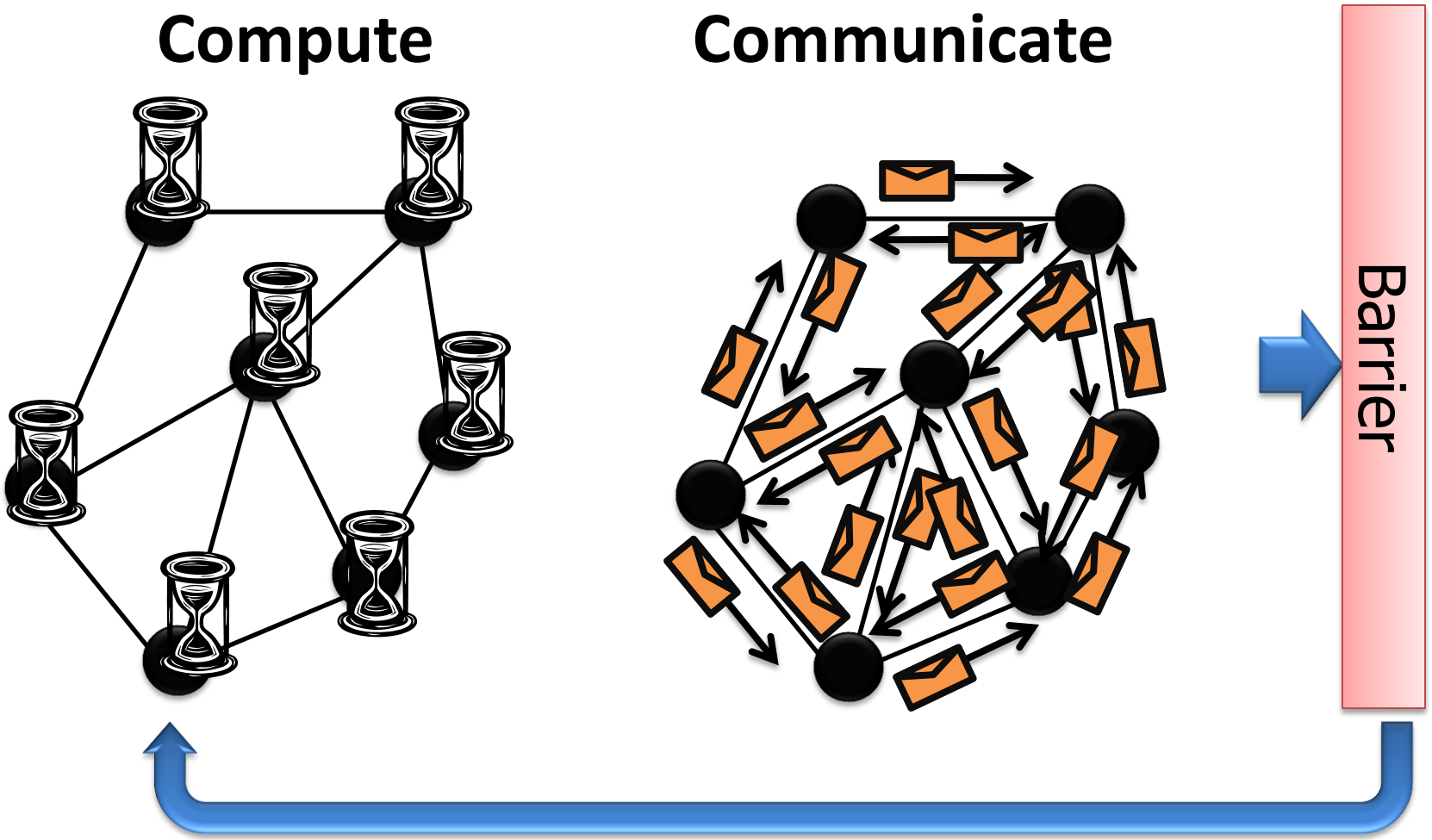
Sum PageRank over incoming messages

http://incubator.apache.org/giraph/

Tradeoffs of the BSP Model

- Pros:
  - *Graph Parallel*
  - Relatively easy to implement and reason about
  - **Deterministic execution**

Embarrassingly Parallel Phases

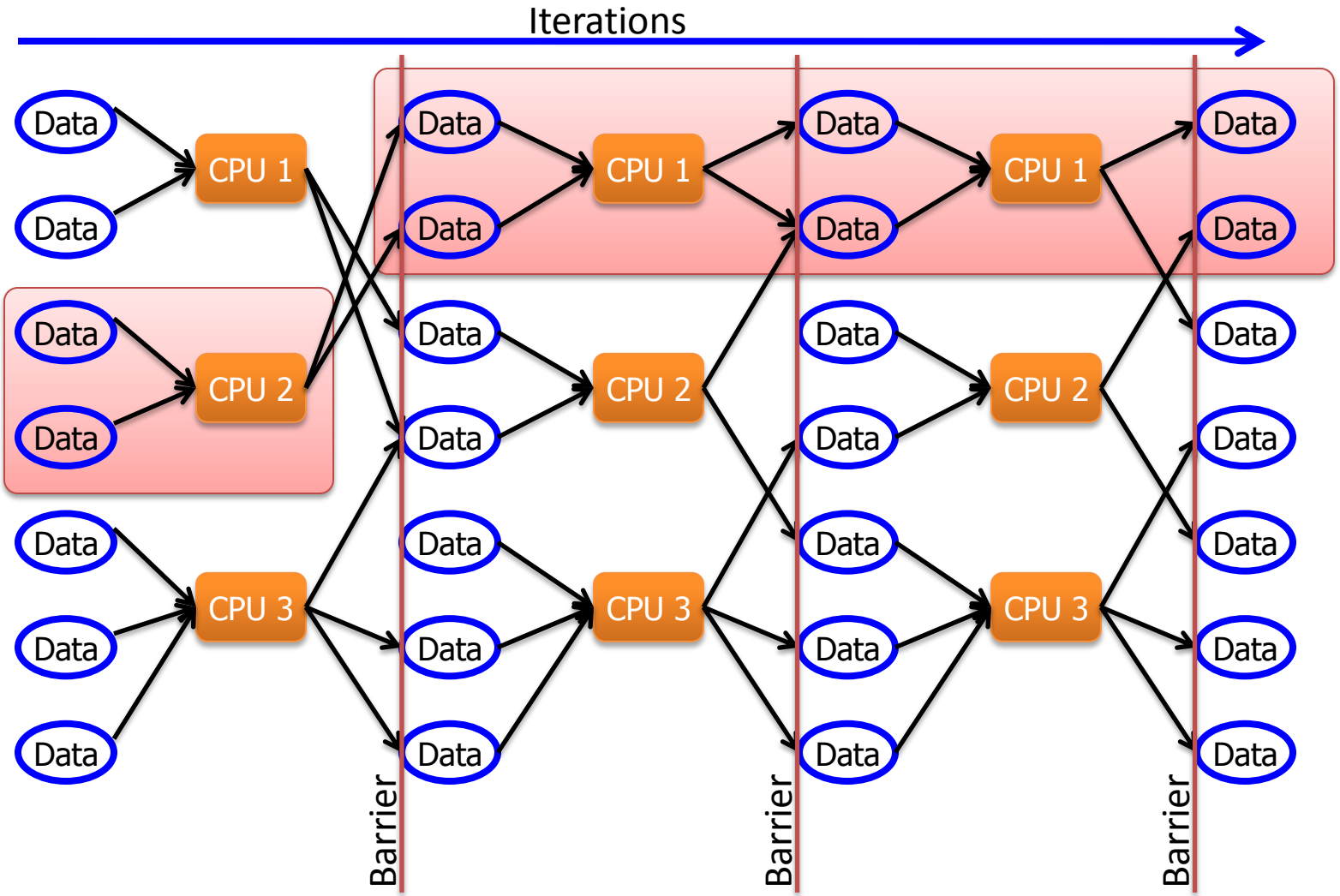


http://dl.acm.org/citation.cfm?id=1807184

Tradeoffs of the BSP Model

- Pros:
  - *Graph Parallel*
  - Relatively easy to build
  - Deterministic execution
- Cons:
  - Doesn’t exploit the graph structure
  - Can lead to inefficient systems

Curse of the Slow Job



http://www.www2011india.com/proceeding/proceedings/p607.pdf

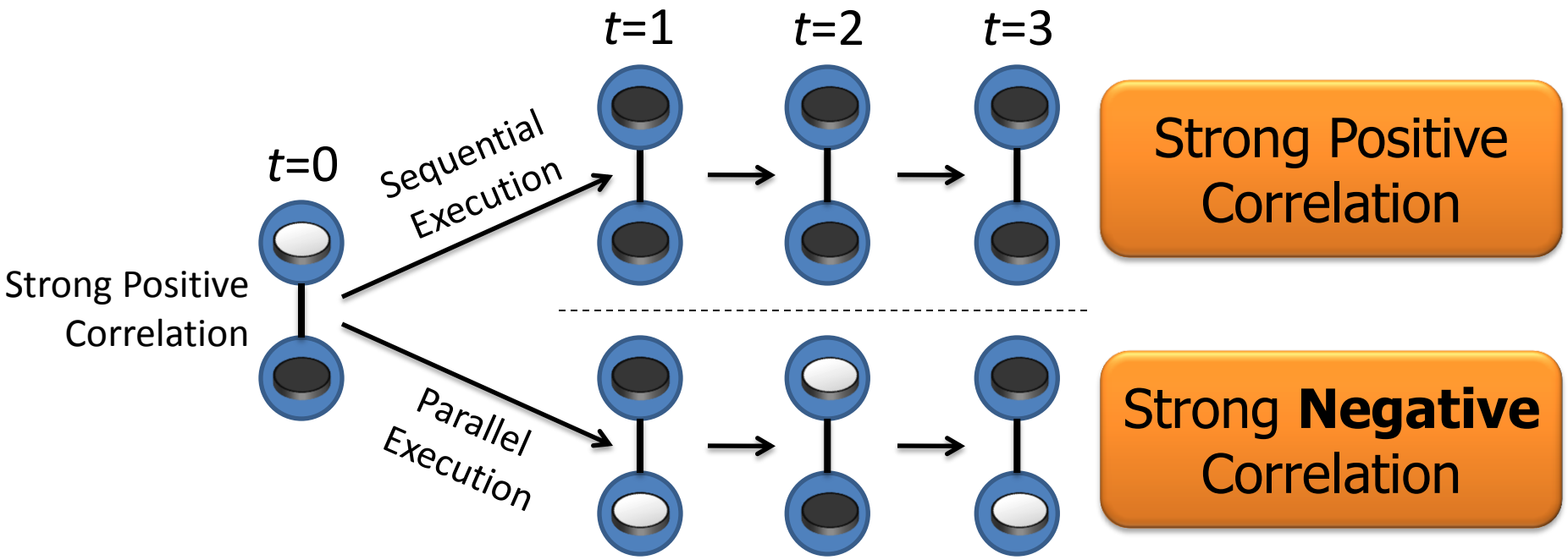
Tradeoffs of the BSP Model

- Pros:
  - *Graph Parallel*
  - Relatively easy to build
  - Deterministic execution
- Cons:
  - Doesn't exploit the graph structure
  - Can lead to inefficient systems
  - Can lead to inefficient computation

Tradeoffs of the BSP Model

- Pros:
  - *Graph Parallel*
  - Relatively easy to build
  - Deterministic execution
- Cons:
  - Doesn’t exploit the graph structure
  - Can lead to inefficient systems
  - Can lead to invalid computation

The problem with **Bulk Synchronous** Gibbs Sampling



- *Adjacent variables **cannot** be sampled **simultaneously**.*

The Need for a New Abstraction

- If not Pregel, then what?

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (Giraph)

Carnegie Mellon

SVM

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

Lasso

Deep Belief Networks

Neural Networks

GraphLab Addresses the Limitations of the BSP Model

- Use graph structure
  - Automatically manage the movement of data
- Focus on **Asynchrony**
  - Computation runs as resources become available
  - Use the most recent information
- Support Adaptive/Intelligent Scheduling
  - Focus computation to where it is needed
- Preserve Serializability
  - Provide the illusion of a sequential execution
  - Eliminate “race-conditions”

What is GraphLab?

http://graphlab.org/ Checkout Version 2

The GraphLab Framework

Graph Based

Data Representation

Update Functions

User Computation

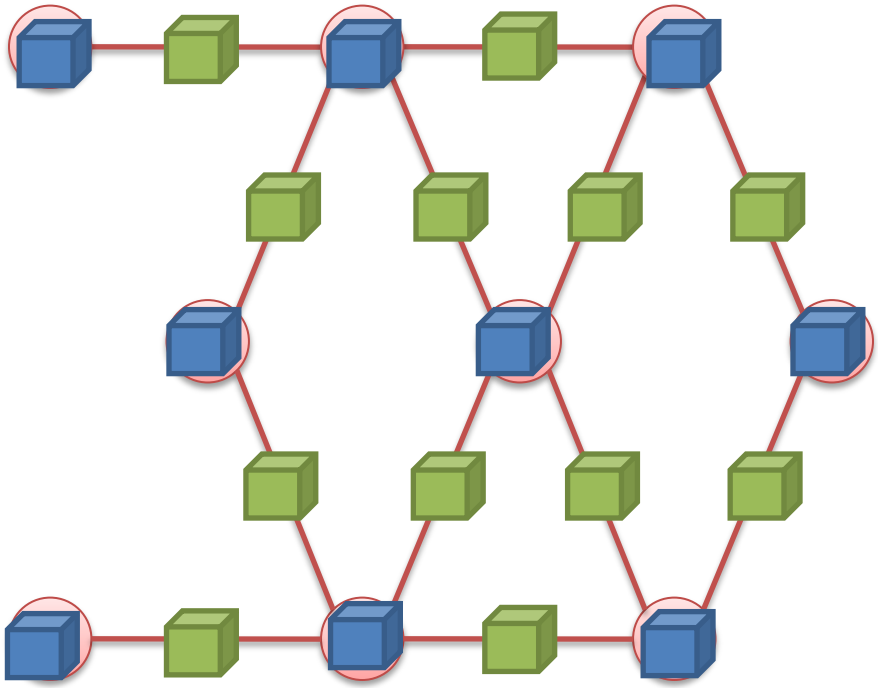
Scheduler

Consistency Model

sense learn act

Data Graph

A **graph** with arbitrary data (C++ Objects) associated with each vertex and edge.



Graph:

- Social Network

Vertex Data:

- User profile text
- Current interests estimates

Edge Data:

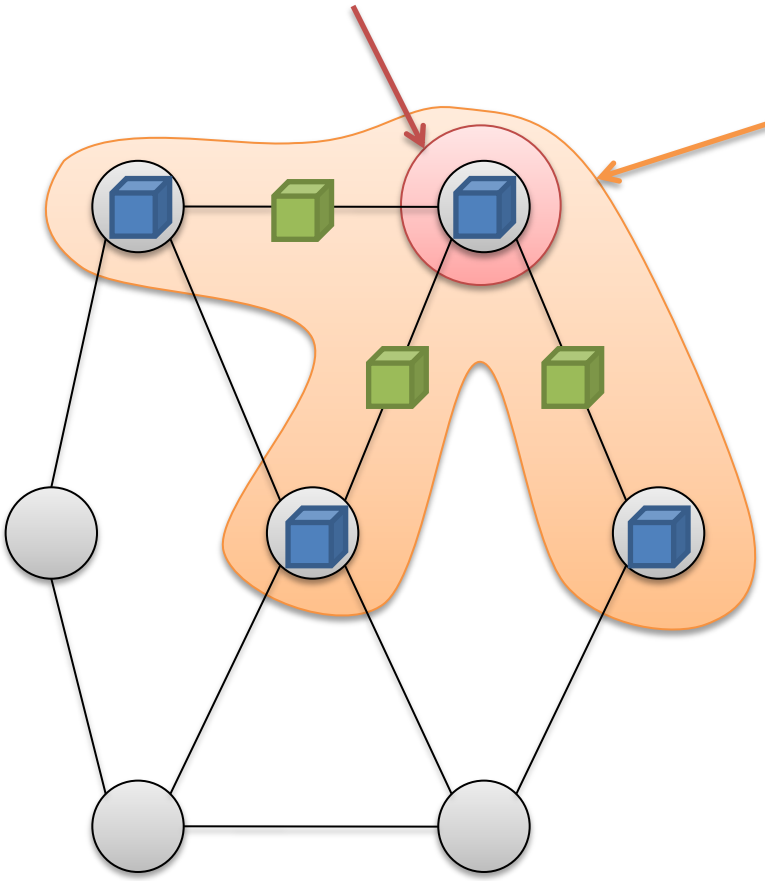
- Similarity weights

Implementing the Data Graph

- All data and structure is stored in memory
  - Supports fast random lookup needed for dynamic computation
- **Multicore Setting:**
  - *Challenge:* Fast lookup, low overhead
  - *Solution:* dense data-structures
- **Distributed Setting:**
  - *Challenge:* Graph partitioning
  - *Solutions:* ParMETIS and Random placement

Update Functions

An **update function** is a user defined program which when applied to a **vertex** transforms the data in the **scope** of the vertex



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

  // Update the vertex data

  R[i] ← α + (1−α) Σ_{j∈N[i]} W_ji × R[j];

  // Reschedule Neighbors if needed
  if R[i] changes then
    reschedule_neighbors_of(i);
}
```

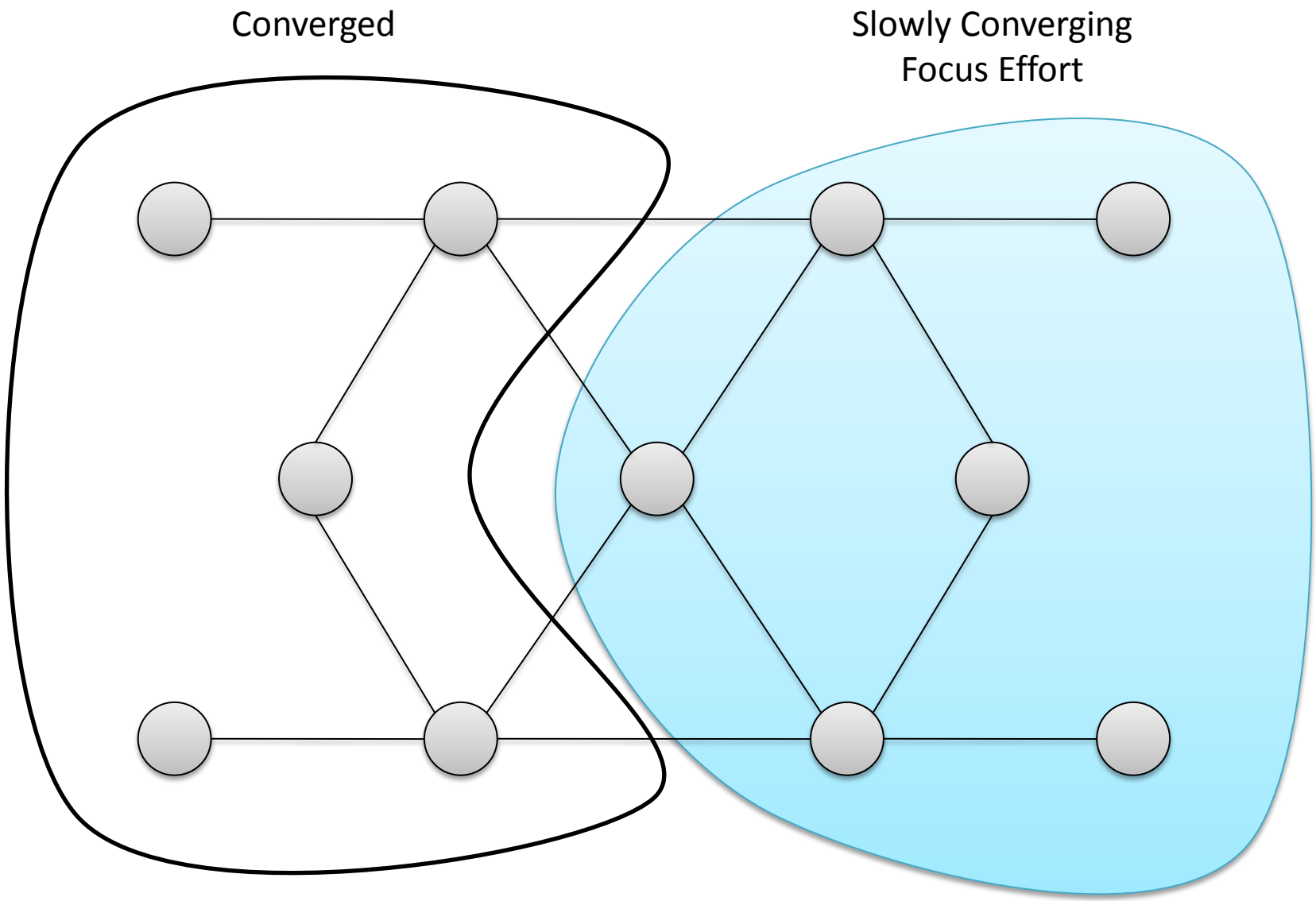
PageRank in GraphLab (V2)

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

```
struct pagerank : public iupdate_functor<graph, pagerank> {
  void operator()(icontext_type& context) {
    double sum = 0;
    foreach ( edge_type edge, context.in_edges() )
      sum += 1/context.num_out_edges(edge.source()) *
        context.vertex_data(edge.source());
    double& rank = context.vertex_data();
    double old_rank = rank;
    rank = RESET_PROB + (1-RESET_PROB) * sum;
    double residual = abs(rank – old_rank)
    if (residual > EPSILON)
      context.reschedule_out_neighbors(pagerank());
  }
};
```

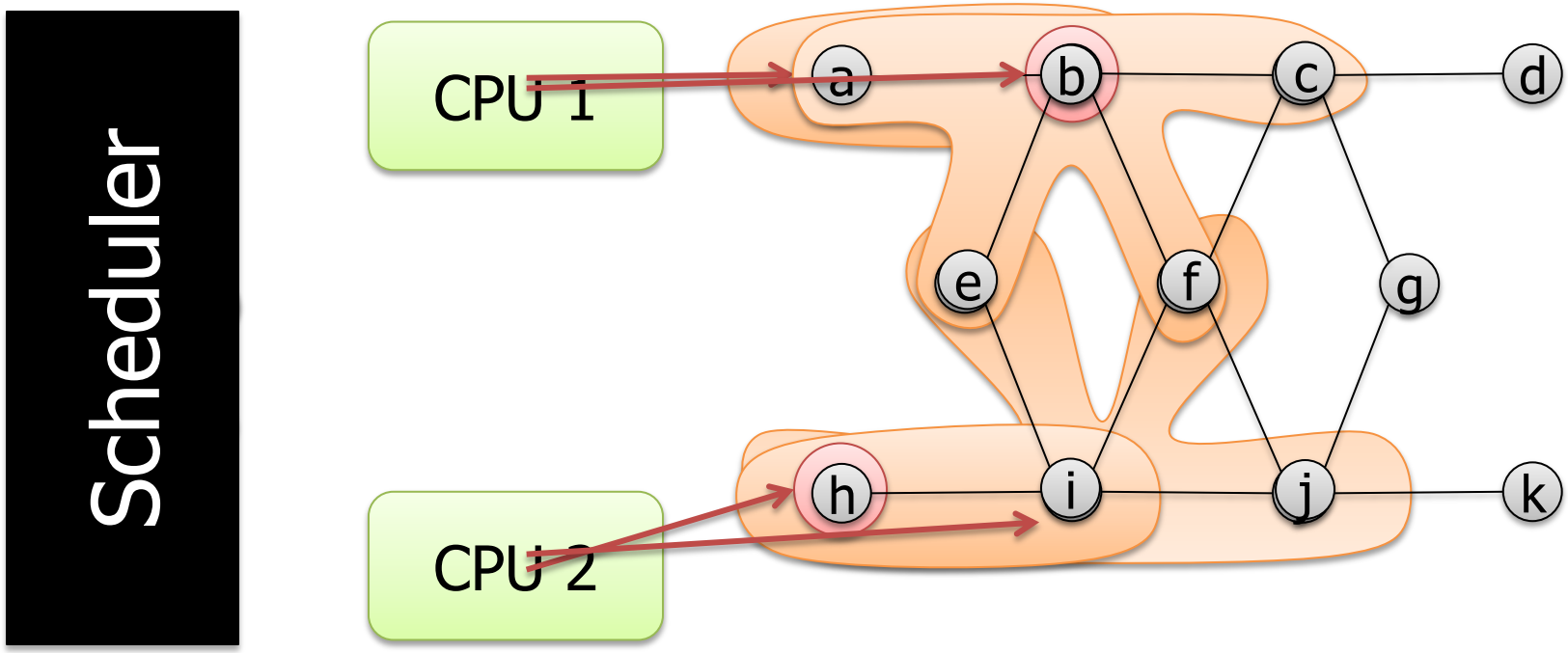
sense
learn
act

Dynamic Computation



The Scheduler

The **scheduler** determines the order that vertices are updated



The process repeats until the scheduler is empty

Choosing a Schedule

The choice of schedule affects the correctness and parallel performance of the algorithm

- GraphLab provides several different schedulers
  - Round Robin: vertices are updated in a fixed order
  - FIFO: Vertices are updated in the order they are added
  - Priority: Vertices are updated in priority order

Obtain different algorithms by simply changing a flag!

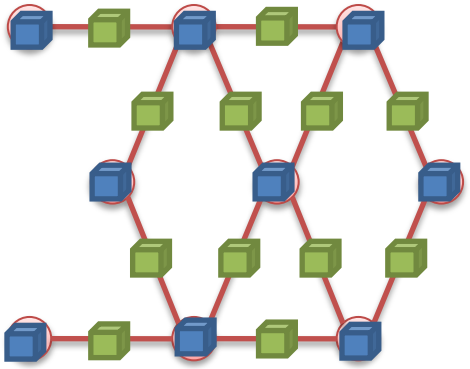
```
--scheduler=sweep
      --scheduler=fifo
            --scheduler=priority
```

sense learn act

The GraphLab Framework

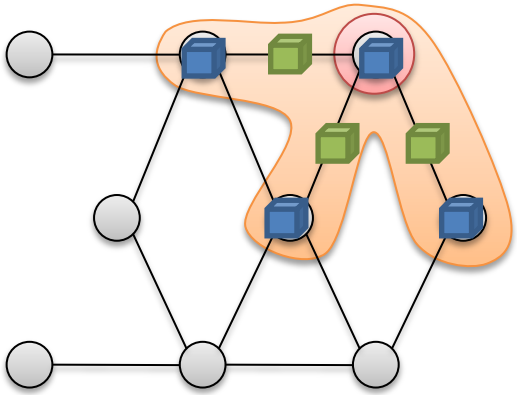
Graph Based

Data Representation

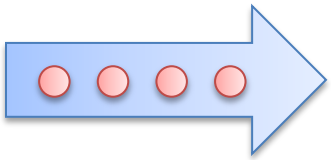


Update Functions

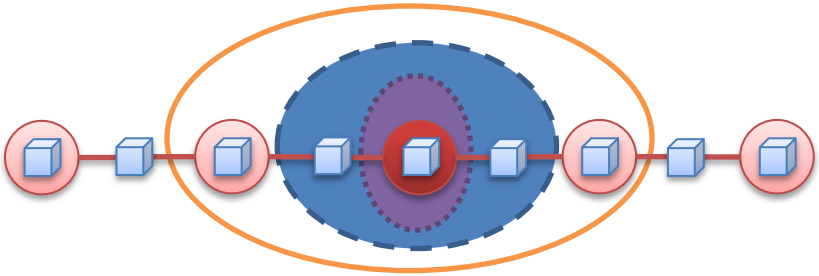
User Computation



Scheduler



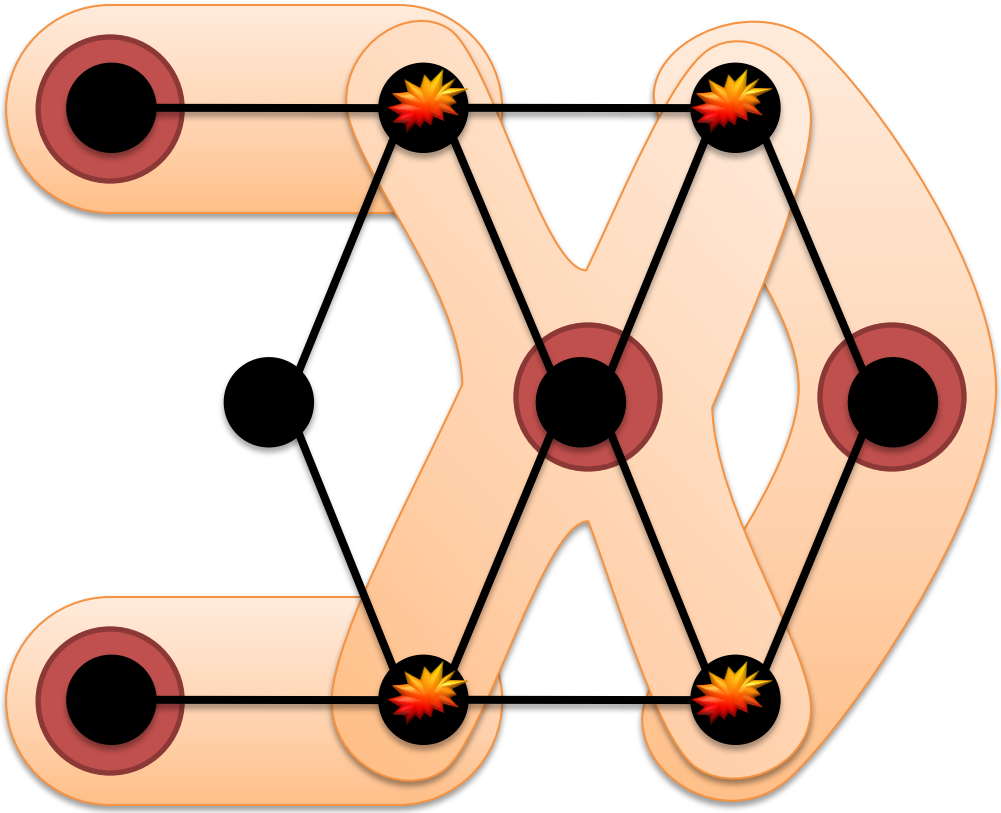
Consistency Model



sense learn act

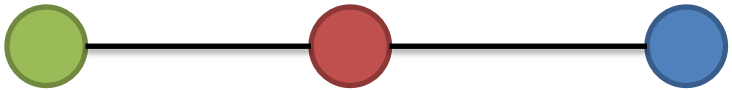
Ensuring Race-Free Execution

- How much can computation **overlap**?



GraphLab Ensures **Sequential Consistency**

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



Parallel

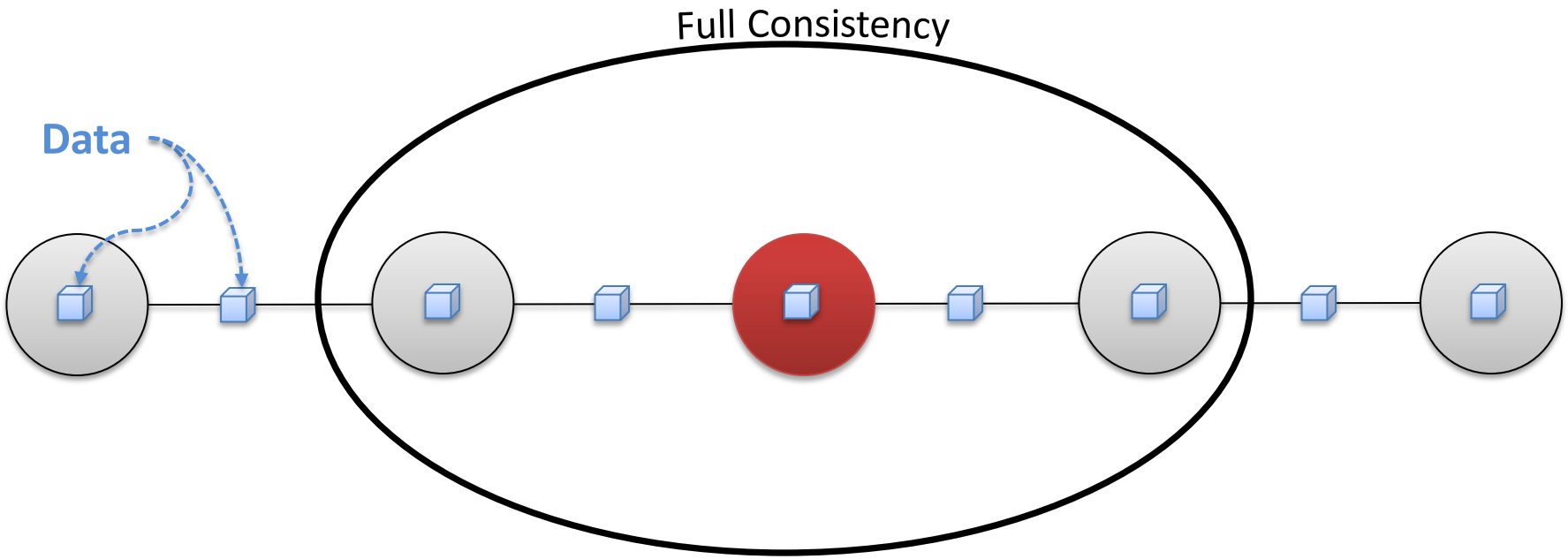
CPU 1

CPU 2

Sequential

Single CPU

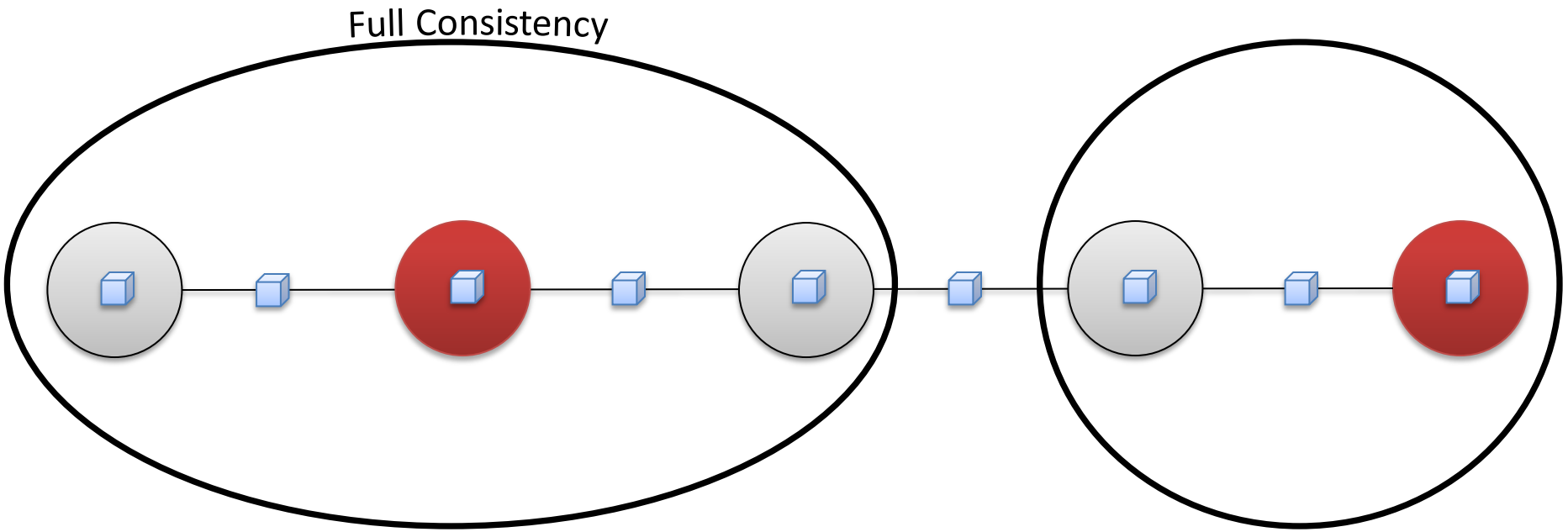
Consistency Rules



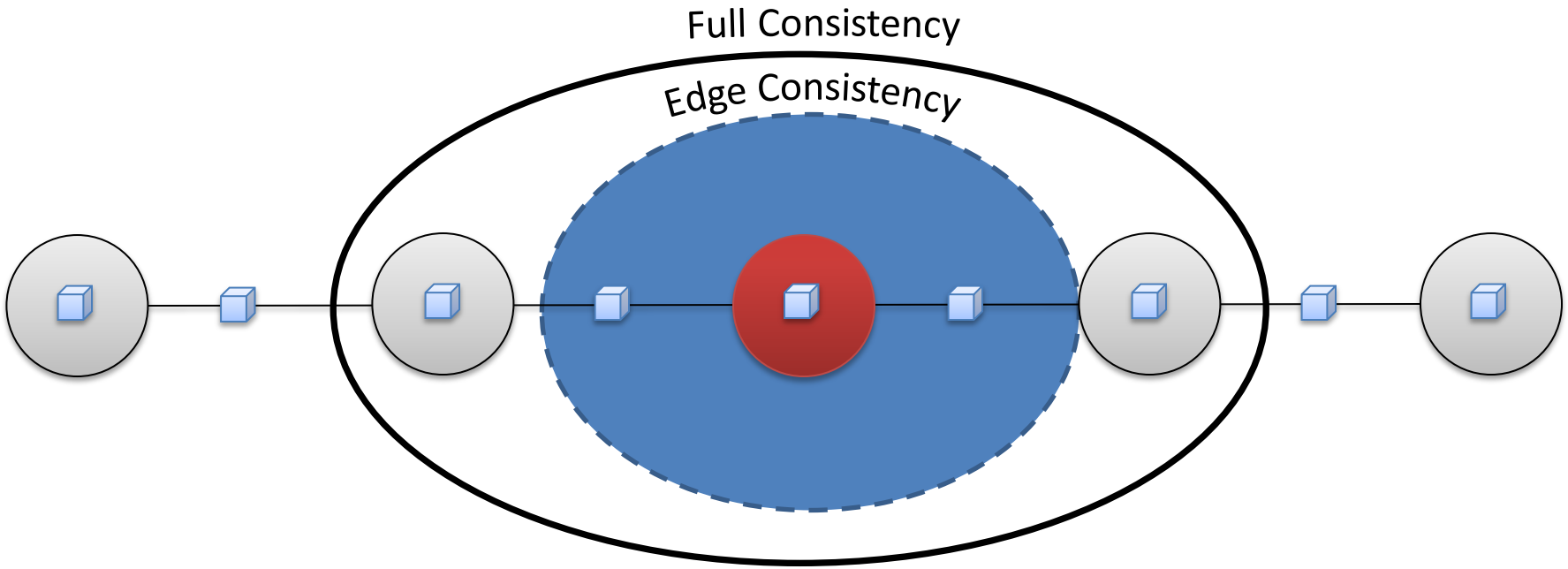
Guaranteed sequential consistency for all update functions

sense learn act

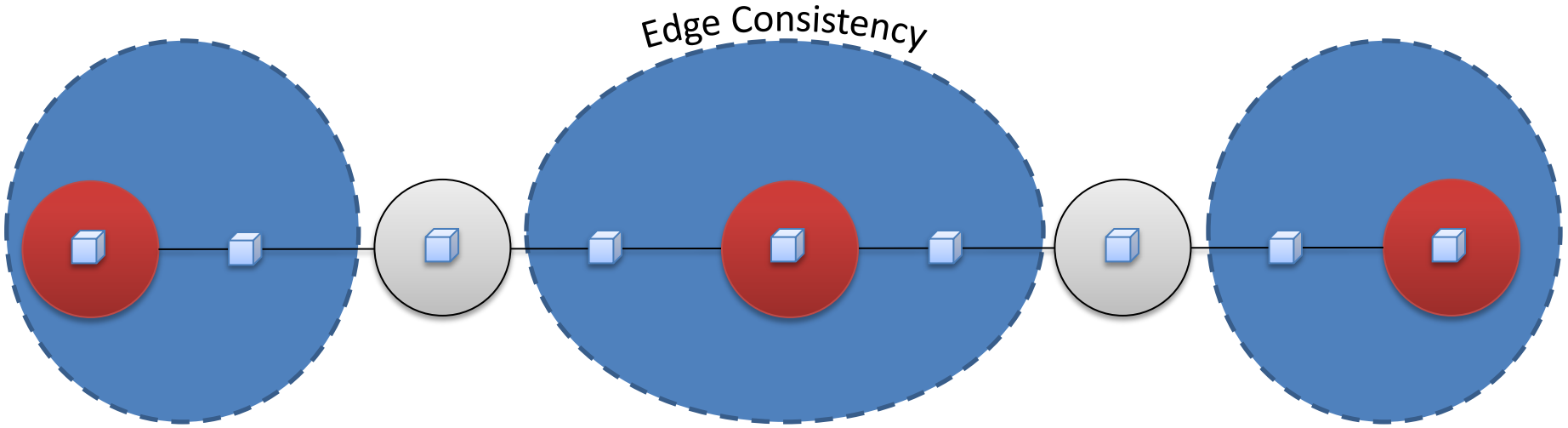
Full Consistency



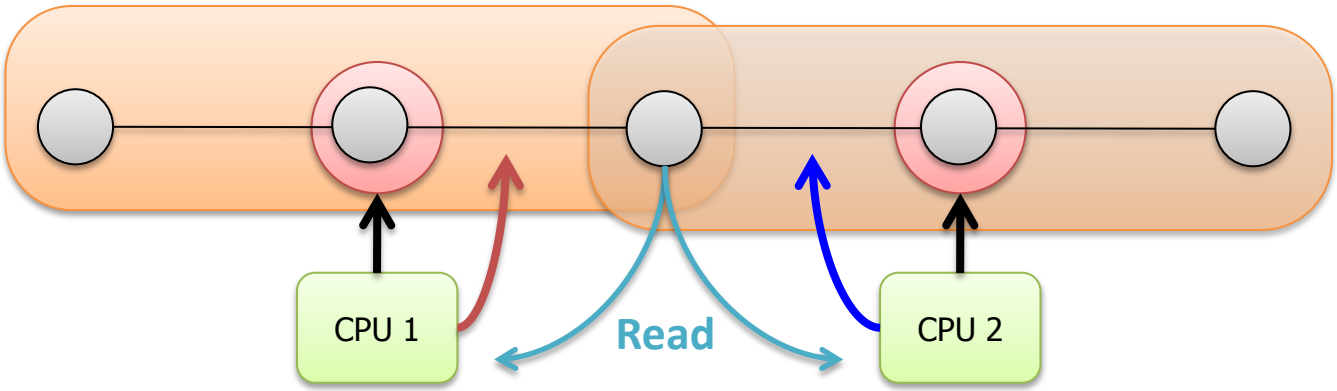
Obtaining More Parallelism



Edge Consistency

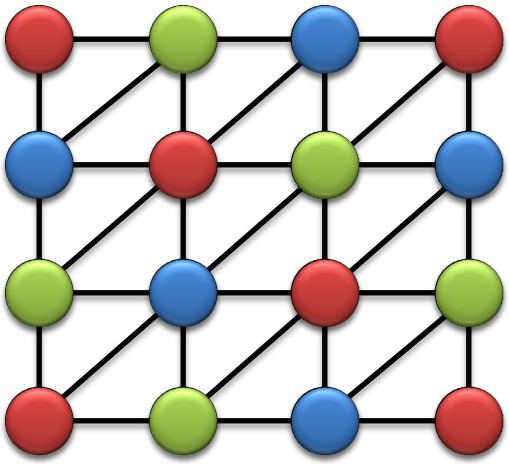


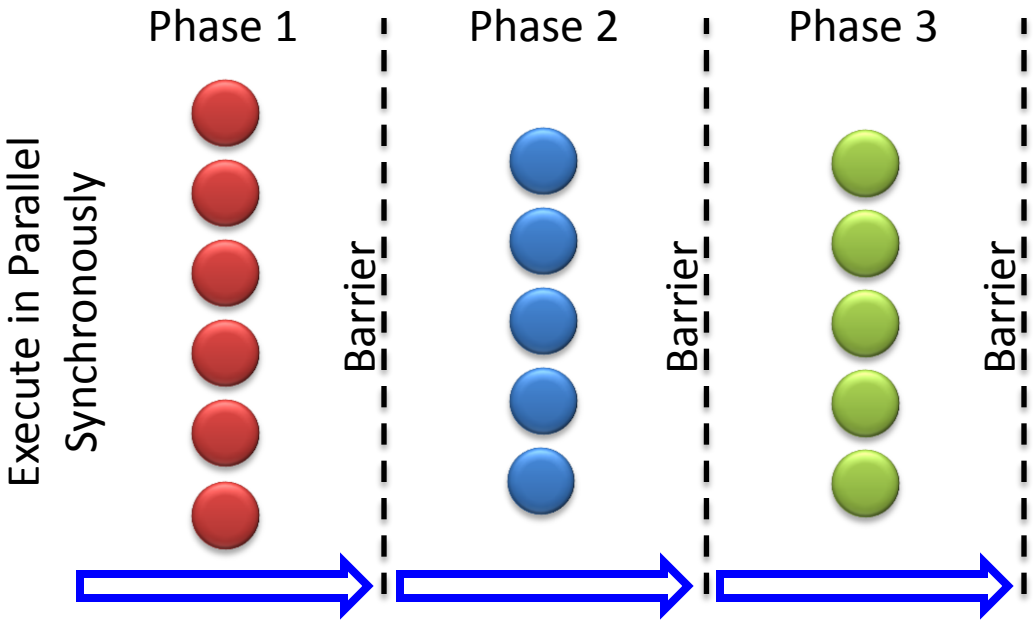
Safe



Consistency Through Scheduling

- Edge Consistency Model:
  - Two vertices can be **Updated** *simultaneously* if they do not share an edge.
- Graph Coloring:
  - Two vertices can be assigned the same color if they do not share an edge.

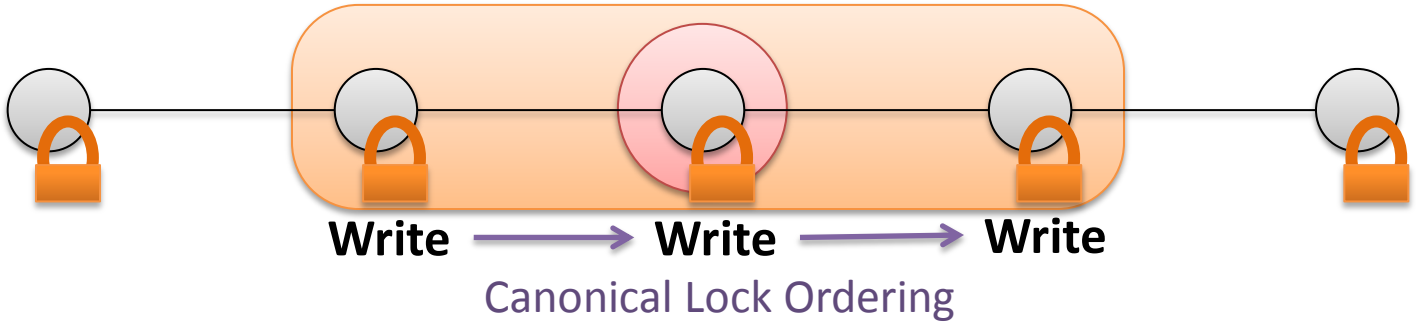




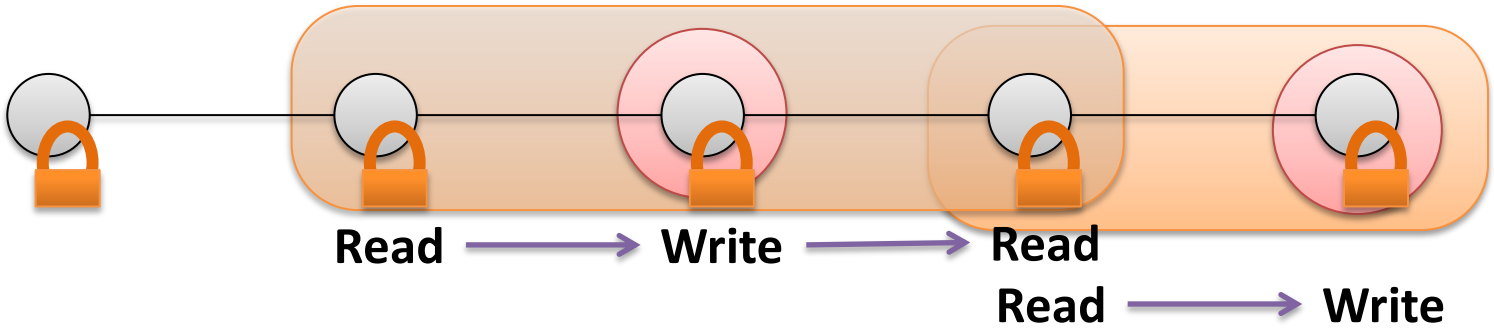
sense learn act

Consistency Through R/W Locks

- Read/Write locks:
  - Full Consistency



  - Edge Consistency

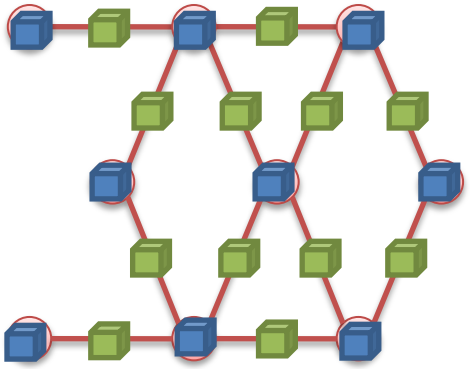


sense learn act

The GraphLab Framework

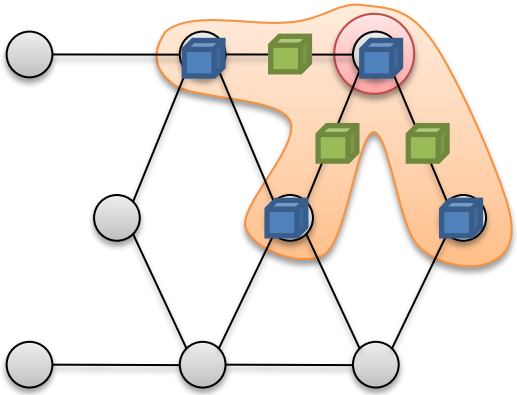
Graph Based

Data Representation

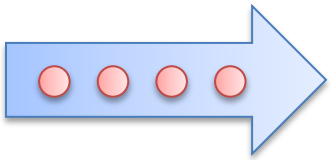


Update Functions

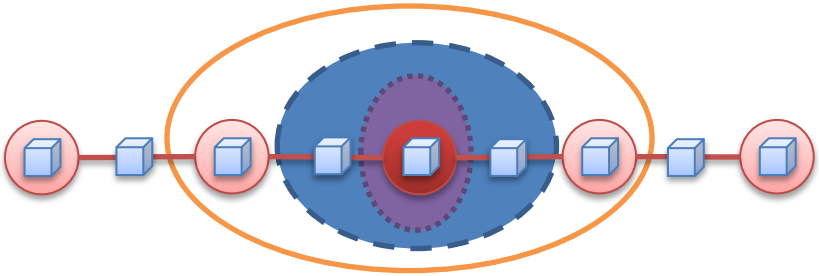
User Computation



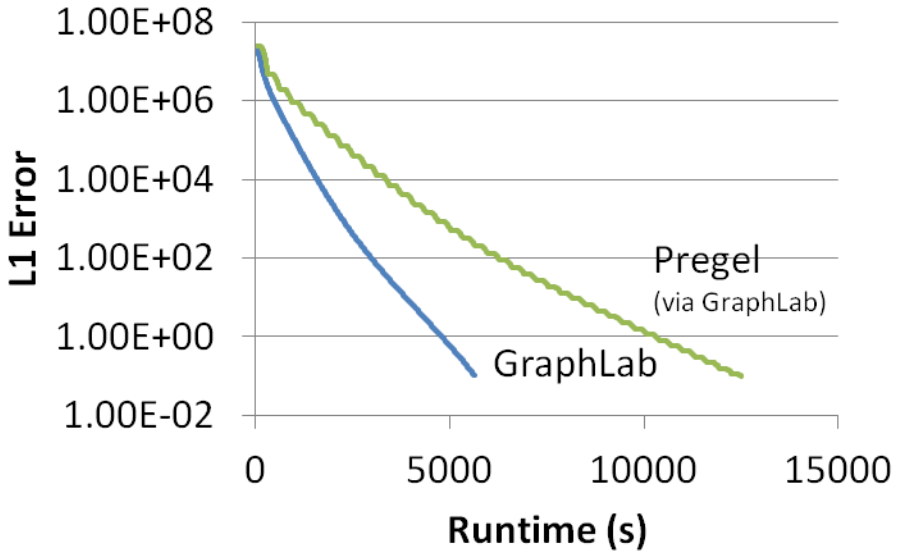
Scheduler

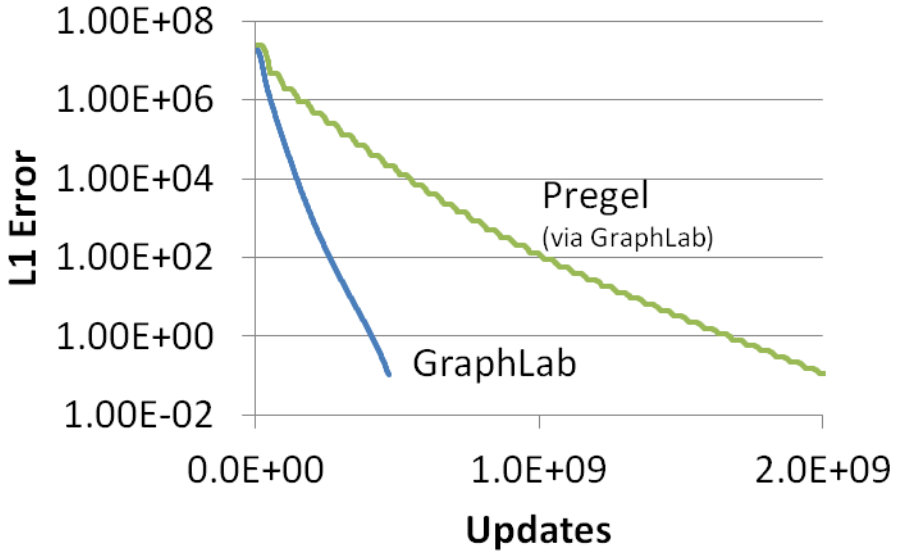


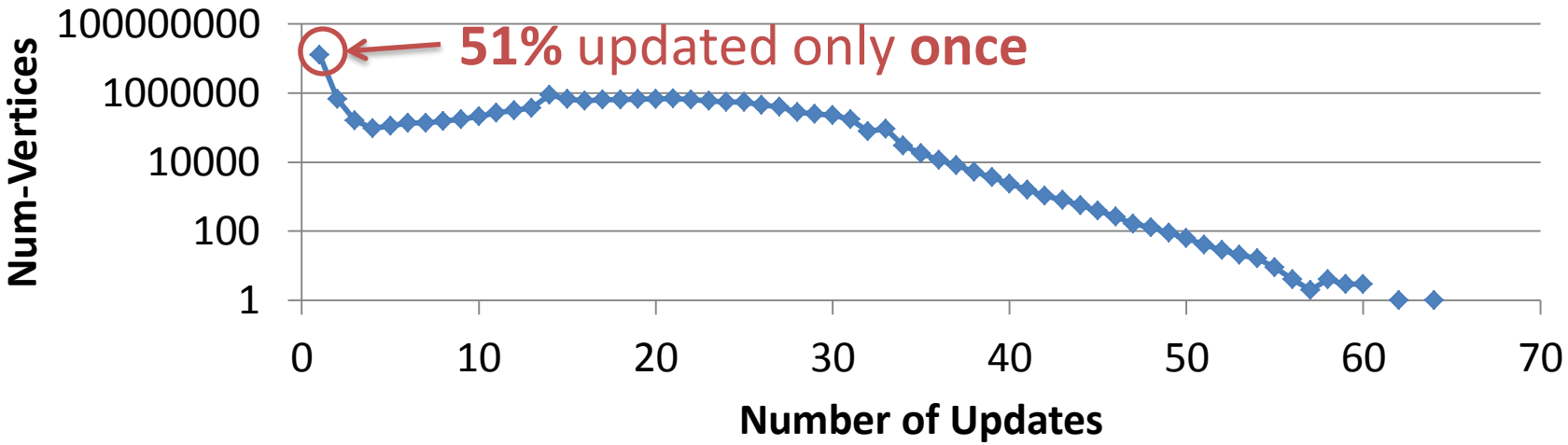
Consistency Model



GraphLab vs. Pregel (BSP)







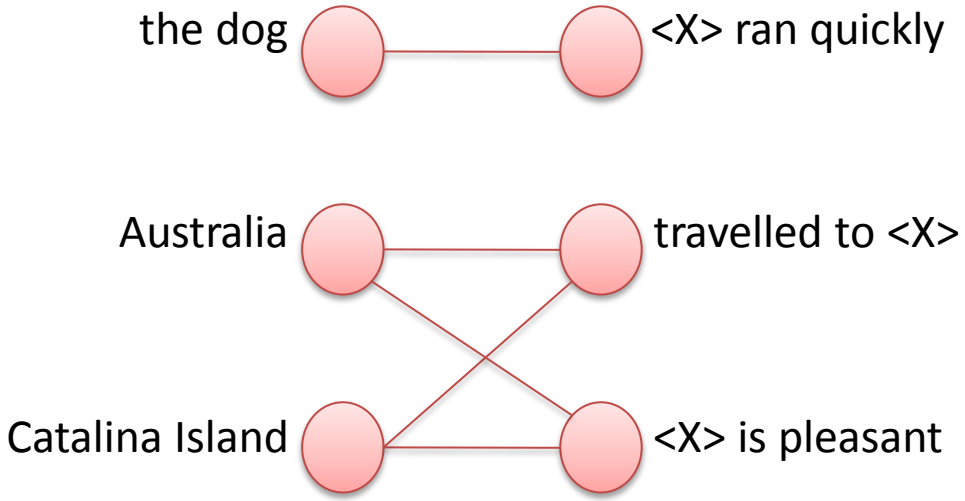
- PageRank (25M Vertices, 355M Edges)

CoEM (Rosie Jones, 2005)

Named Entity Recognition Task

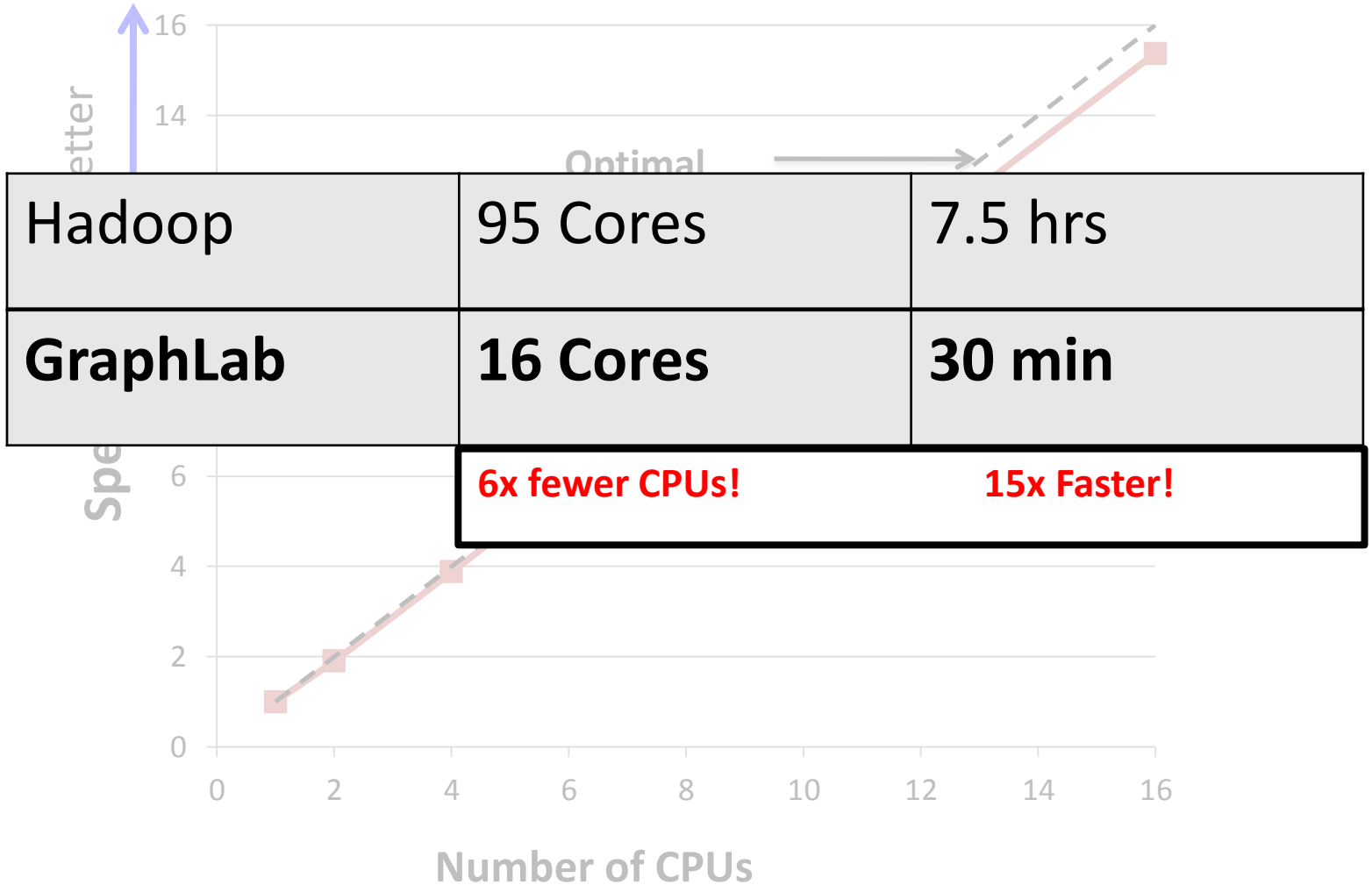
Is “Dog” an animal?
Is “Catalina” a place?

Vertices: 2 Million
Edges: 200 Million



Hadoop	**95 Cores**	**7.5 hrs**

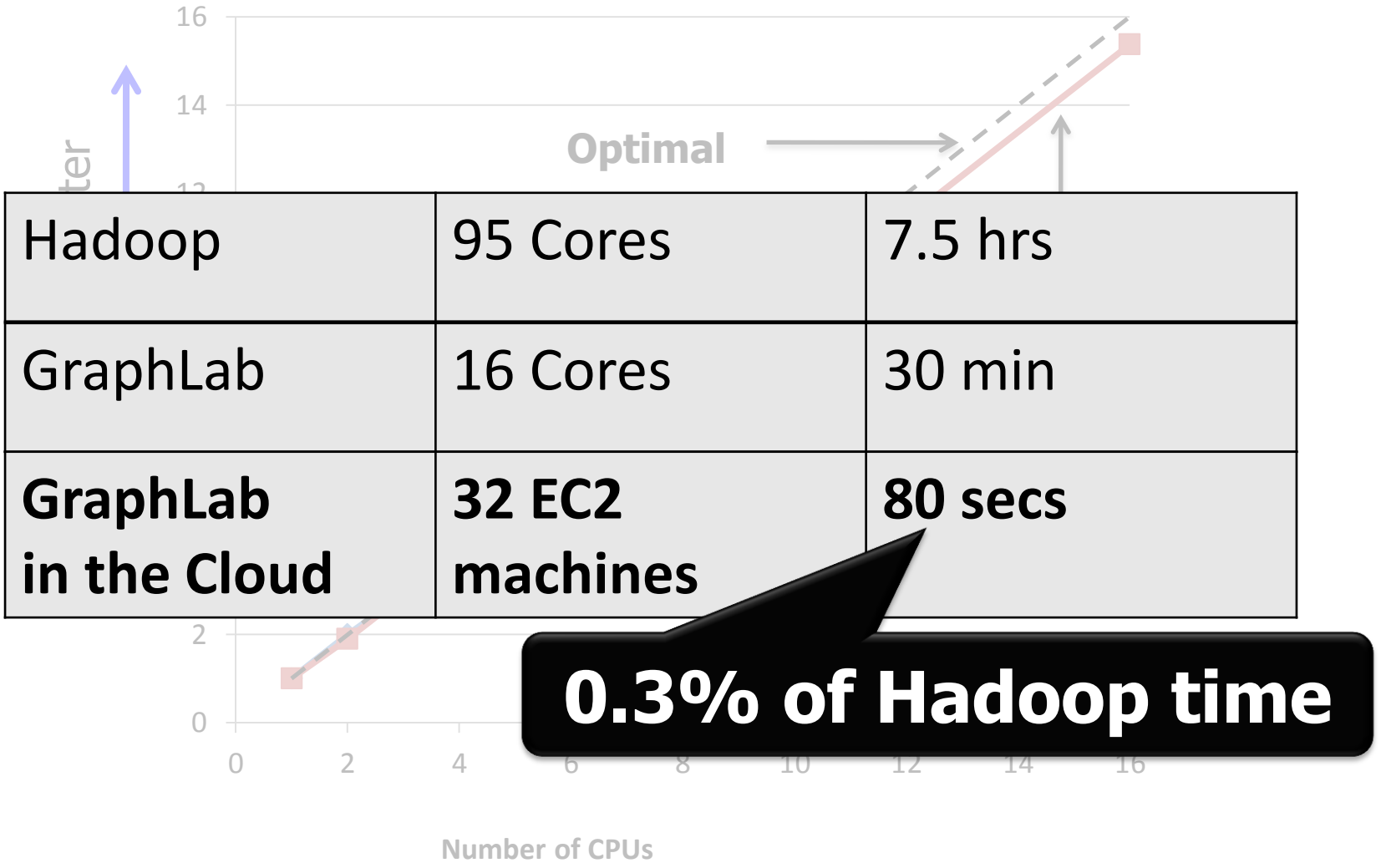
CoEM (Rosie Jones, 2005)



Hadoop	95 Cores	7.5 hrs
GraphLab	**16 Cores**	**30 min**

6x fewer CPUs! **15x Faster!**

CoEM (Rosie Jones, 2005)

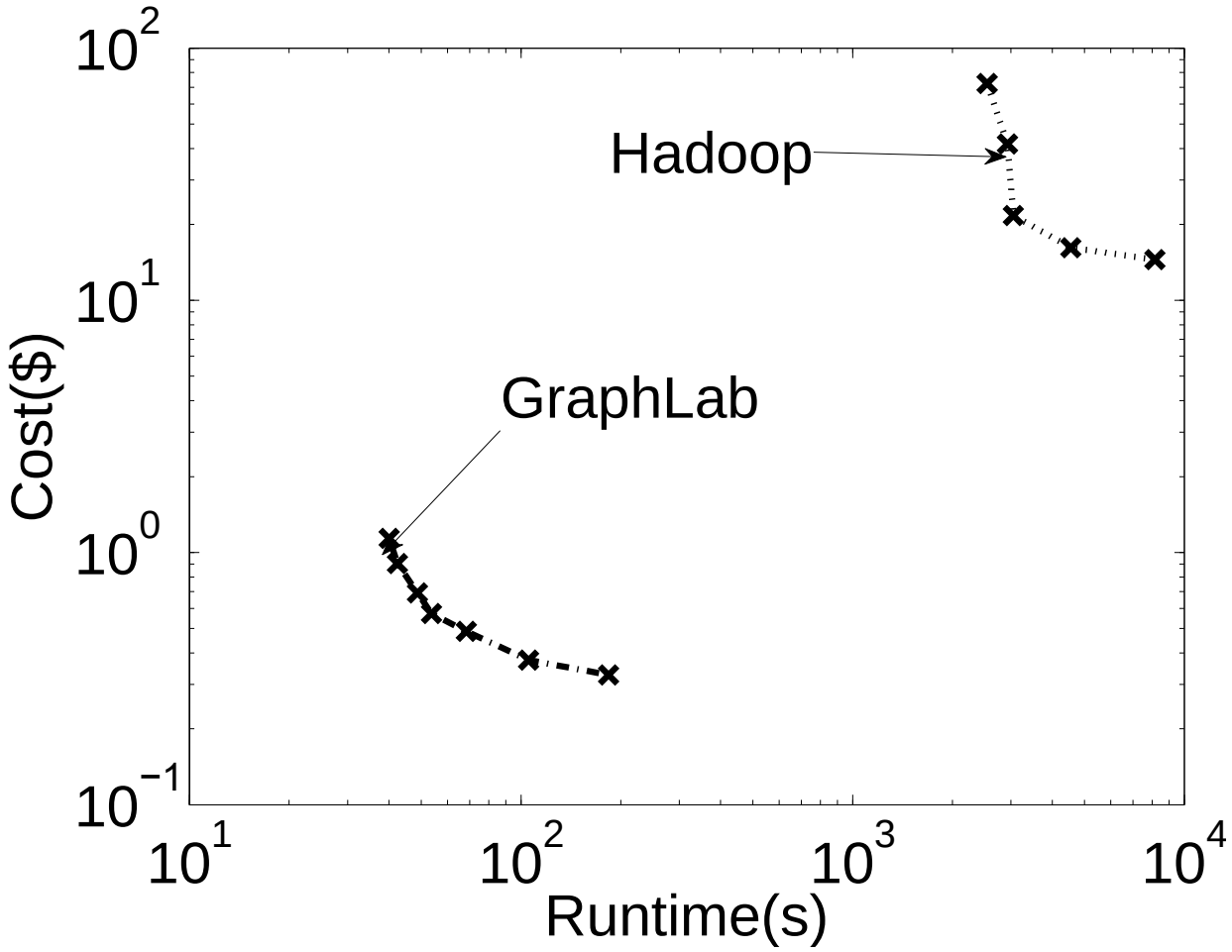


Hadoop	95 Cores	7.5 hrs
GraphLab	16 Cores	30 min
GraphLab in the Cloud	**32 EC2 machines**	**80 secs**

0.3% of Hadoop time

The Cost of the Wrong Abstraction

Log-Scale!



Tradeoffs of GraphLab

- Pros:
  - Separates algorithm from movement of data
  - Permits dynamic asynchronous scheduling
  - More expressive consistency model
  - Faster and more efficient runtime performance
- Cons:
  - **Non-deterministic execution**
  - Defining “residual” can be tricky
  - Substantially more complicated to implement

Thank you !

Any questions?