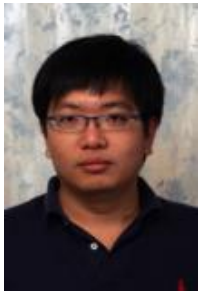
GraphLab
Carnegie Mellon

A New Parallel Framework for Machine Learning

Joseph Gonzalez

Joint work with


Yucheng Low


Aapo Kyrola

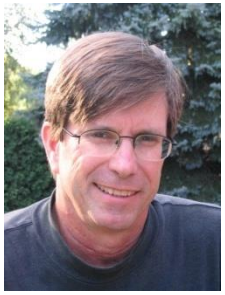

Danny Bickson


Carlos Guestrin


Alex Smola


Guy Blelloch


Joe Hellerstein


David O'Hallaron

Select Lab

Carnegie Mellon

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

We could use ….

Threads, Locks, & Messages

“low level parallel primitives”

sense learn act

Threads, Locks, and Messages

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

"We implemented _____ in parallel."

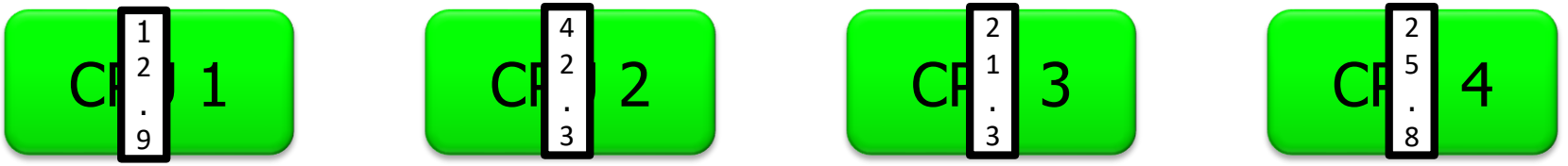
- The resulting code:
  - is difficult to maintain
  - is difficult to extend
  - couples learning model to parallel implementation

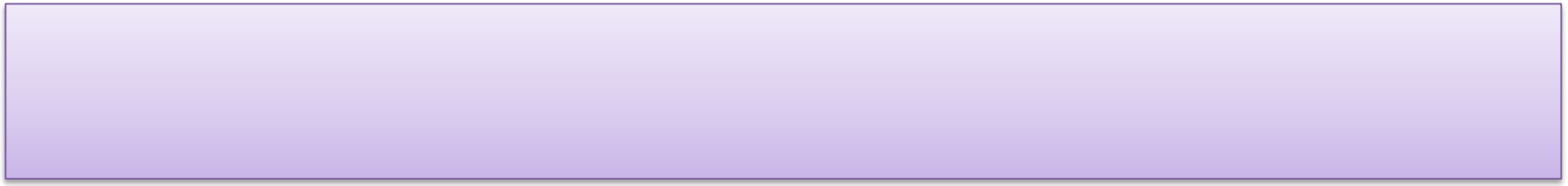
... a better answer:

Map-Reduce / Hadoop

Build learning algorithms on-top of
high-level parallel abstractions

MapReduce – Map Phase

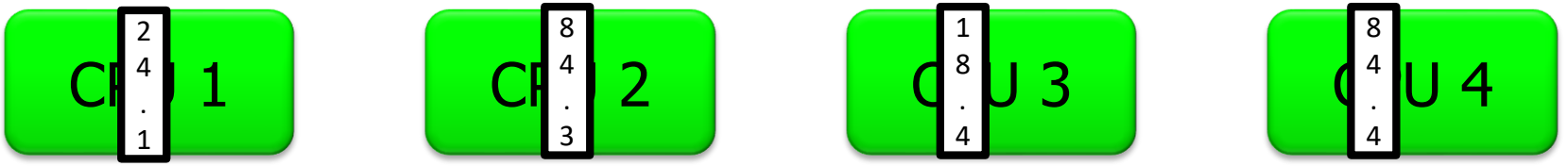




Embarrassingly Parallel independent computation

No Communication needed

MapReduce – Map Phase



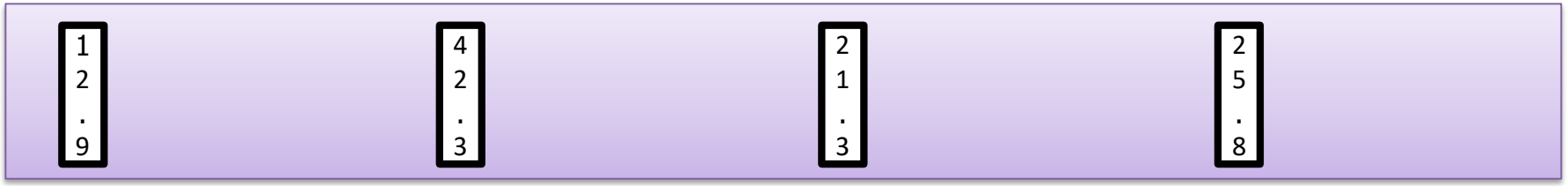
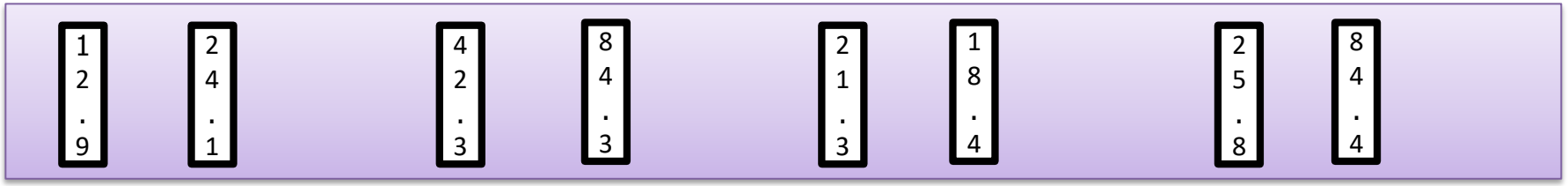


Image Features

MapReduce – Map Phase



Embarrassingly Parallel independent computation

No Communication needed

MapReduce – Reduce Phase

Attractive Face Statistics

Ugly Face Statistics

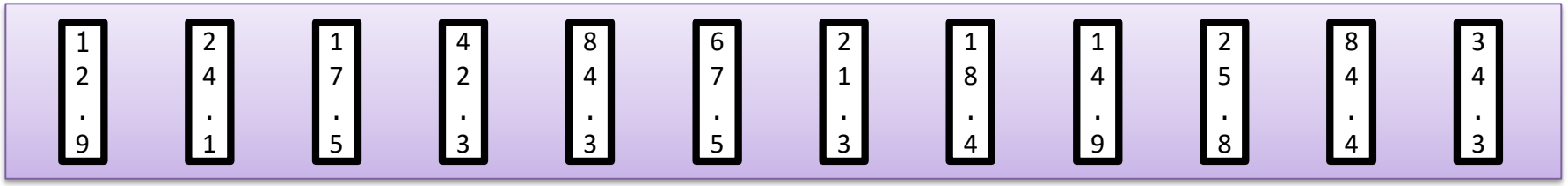


Image Features

Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Is there more to Machine Learning

?

Concrete Example

Label Propagation

Label Propagation Algorithm

- Social Arithmetic:

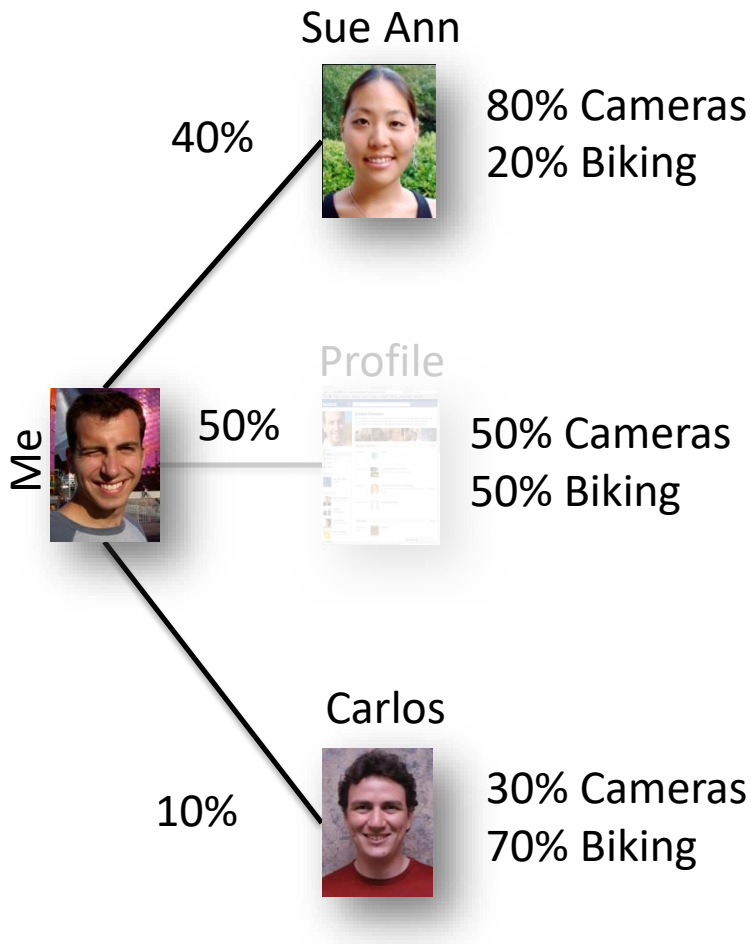
  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

  I Like: 60% Cameras, 40% Biking

- Recurrence Algorithm:

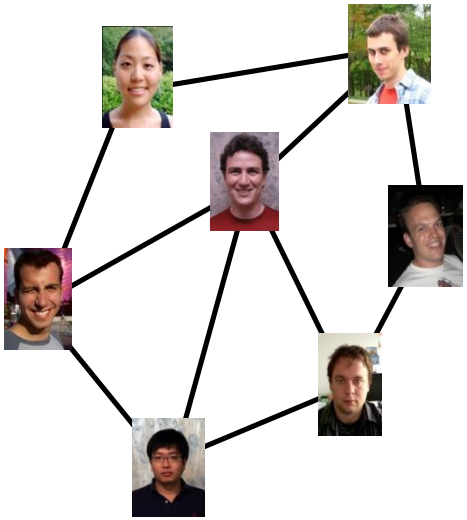
$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

  - iterate until convergence
- Parallelism:
  - Compute all *Likes[i]* in parallel

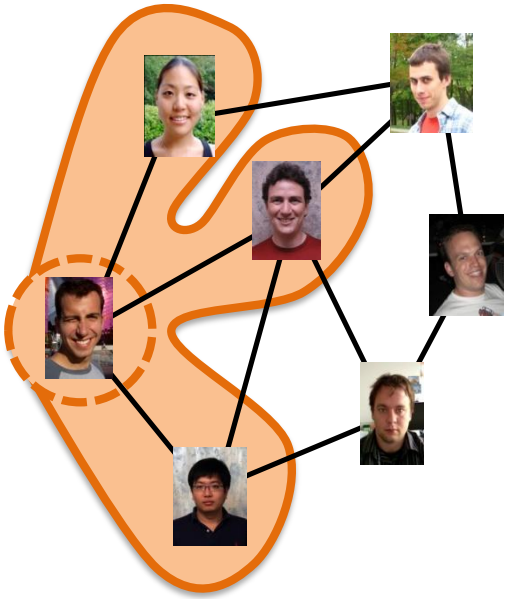


Properties of Graph Parallel Algorithms

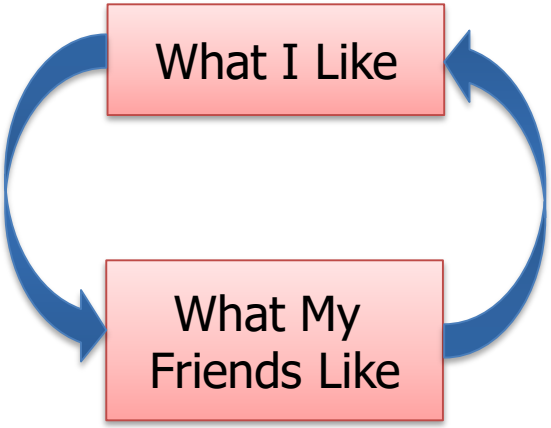
Dependency Graph



Factored Computation

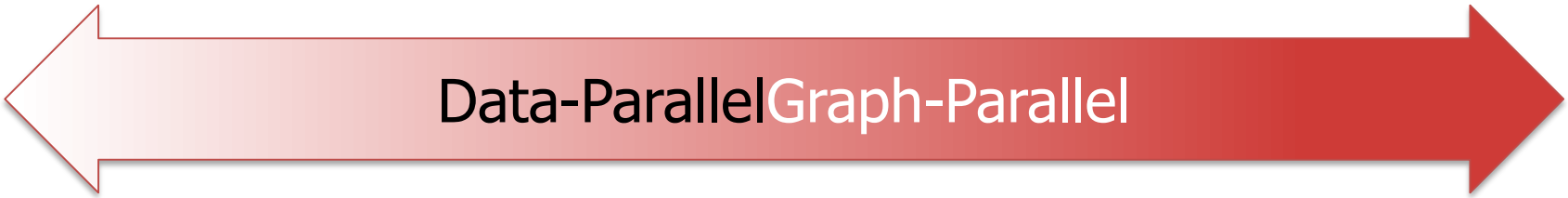


Iterative Computation



Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!



Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Map Reduce?

Lasso

Label Propagation

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

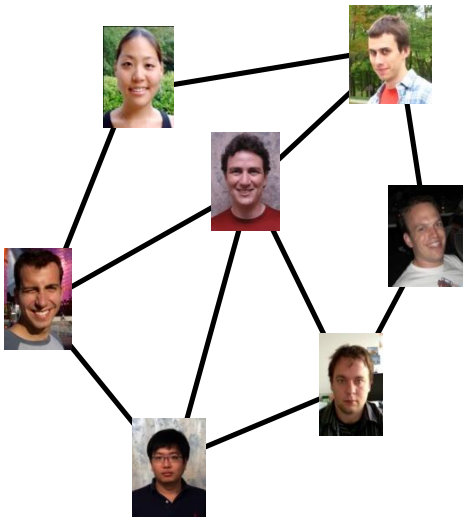
Deep Belief Networks

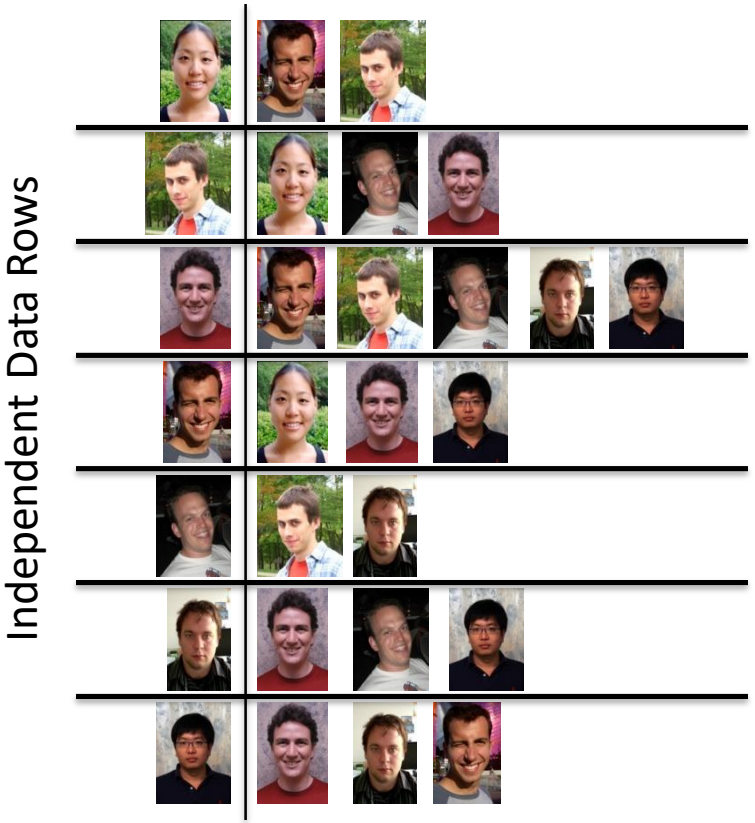
Neural Networks

Why not use Map-Reduce for Graph Parallel Algorithms?

Data Dependencies

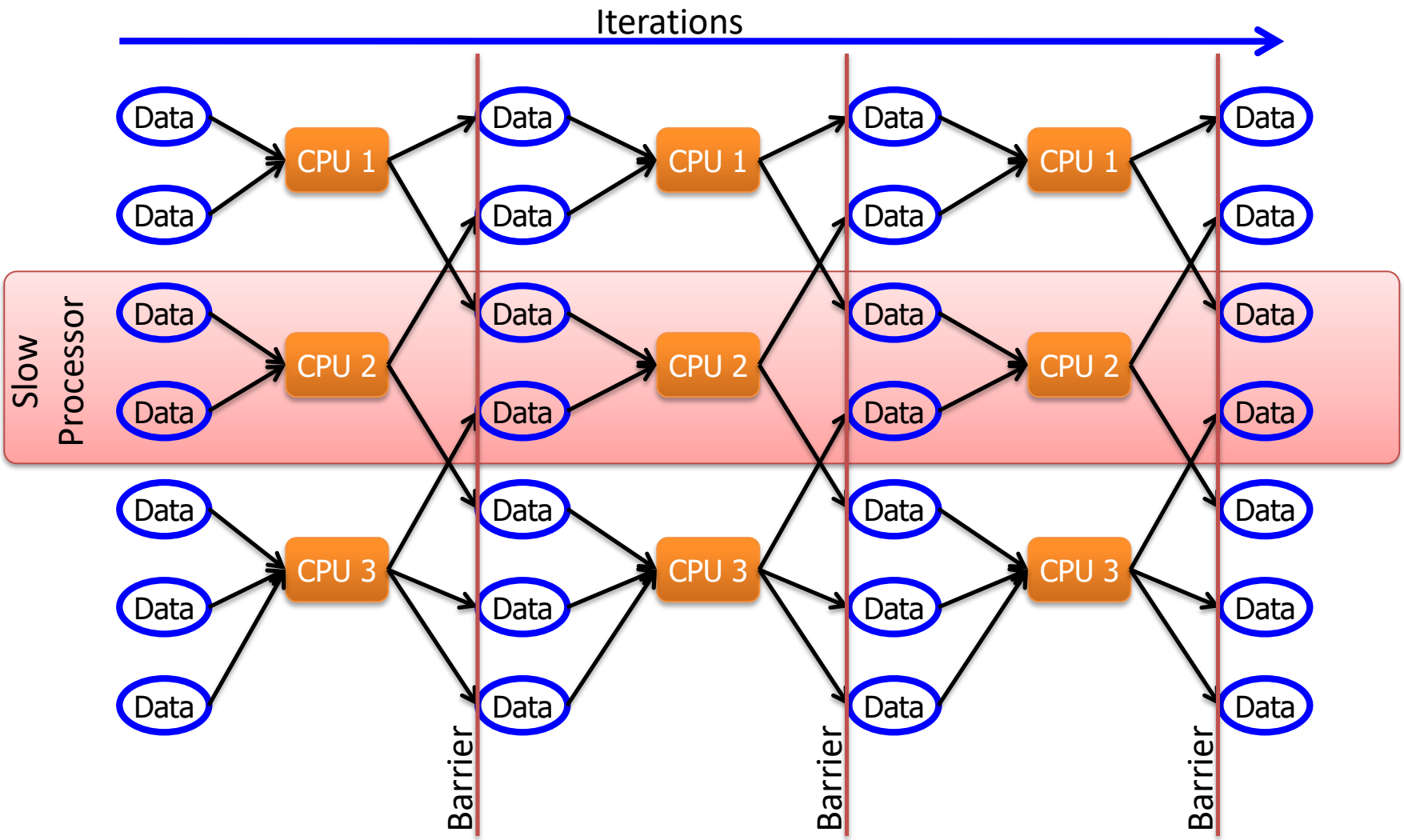
- Map-Reduce does not efficiently express dependent data
  - User must code substantial data transformations
  - Costly data replication





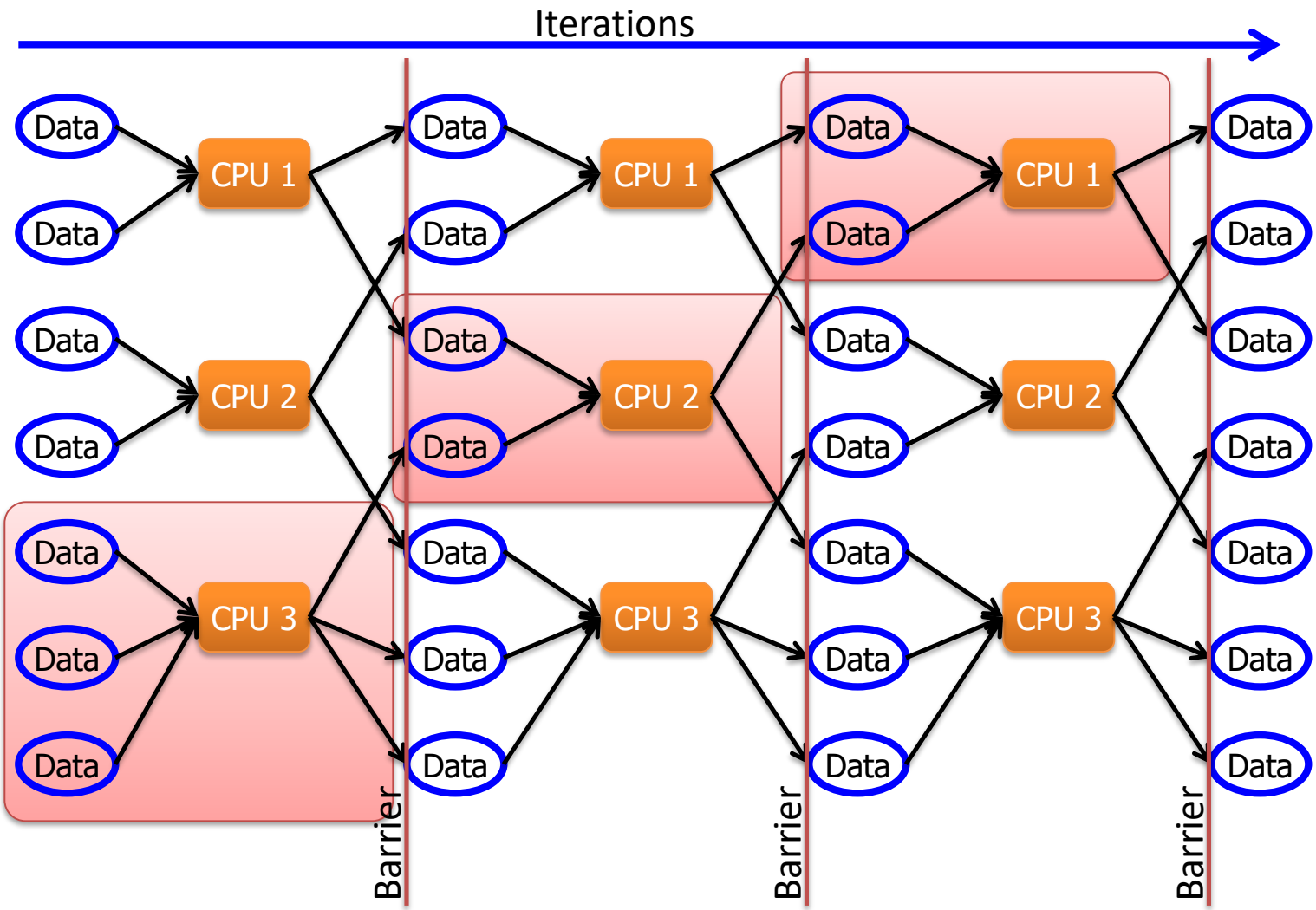
Iterative Algorithms

- Map-Reduce not efficiently express iterative algorithms:



MapAbuse: Iterative MapReduce

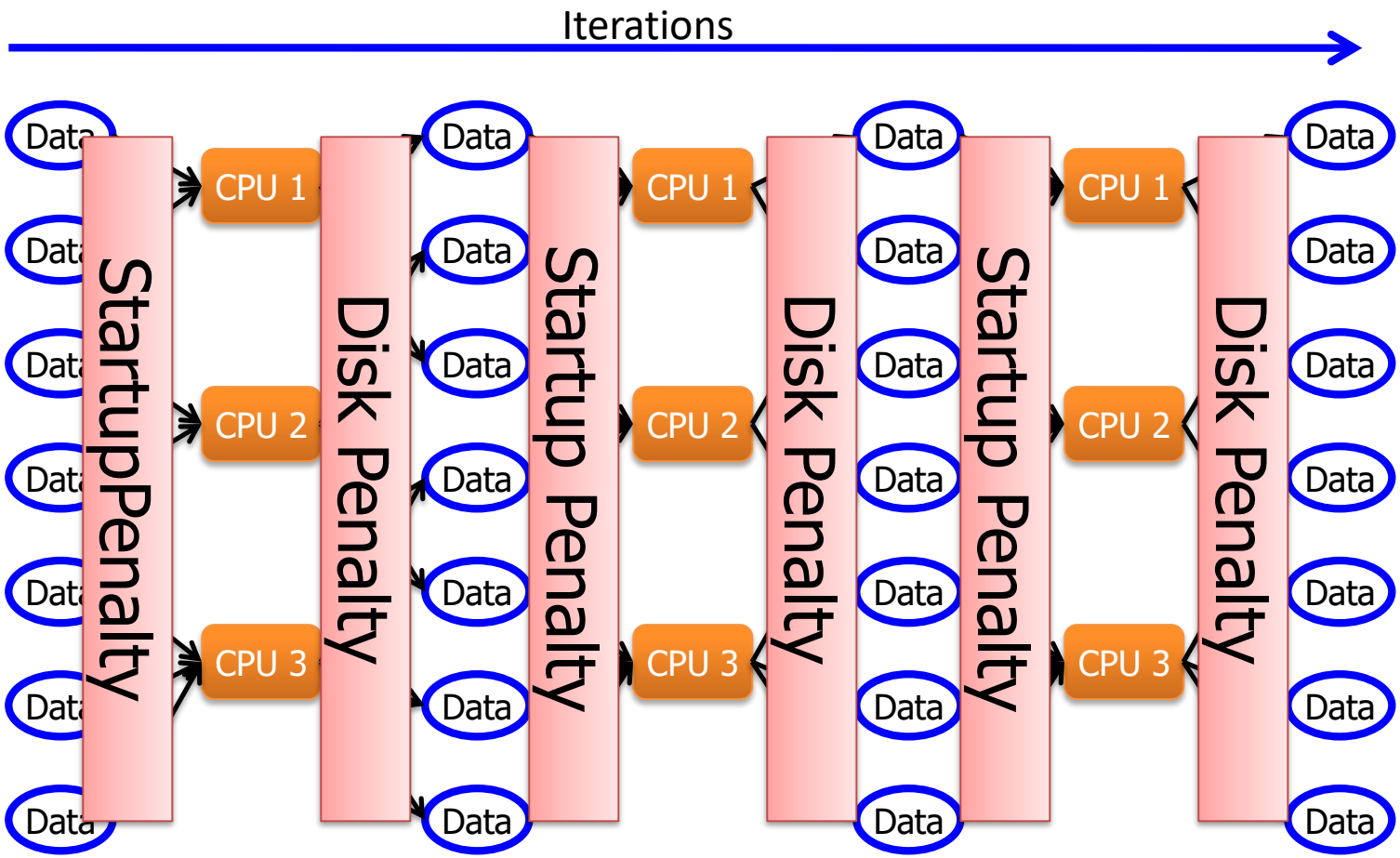
- Only a subset of data needs computation:



sense learn act

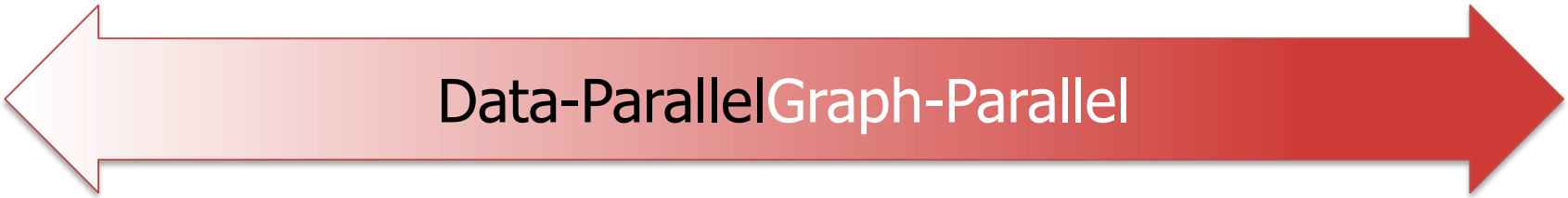
MapAbuse: Iterative MapReduce

- 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

Pregel (Giraph)?

Lasso

SVM

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

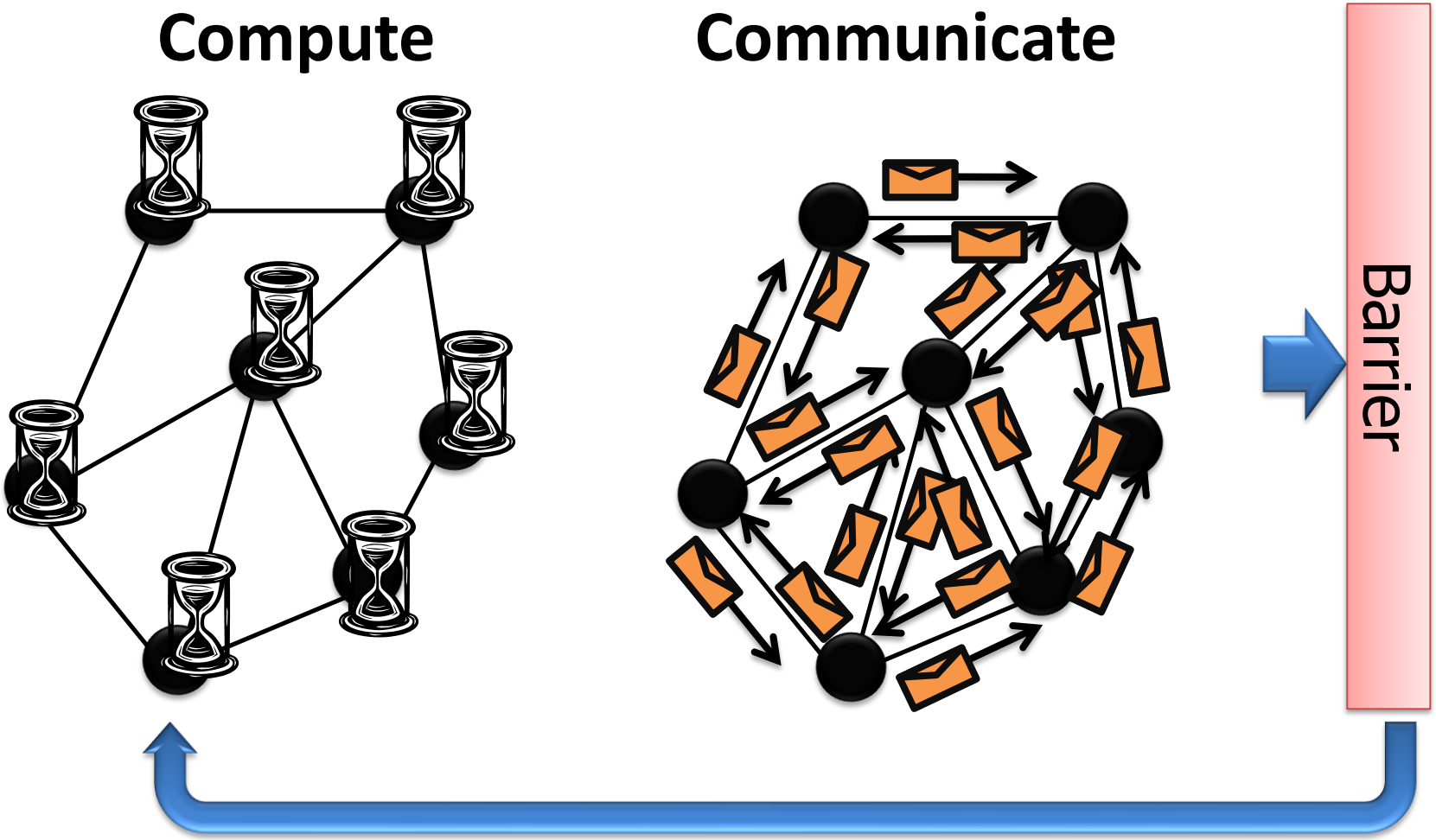
Deep Belief Networks

Neural Networks

sense learn act

Pregel (Giraph)

- Bulk Synchronous Parallel Model:



Problem

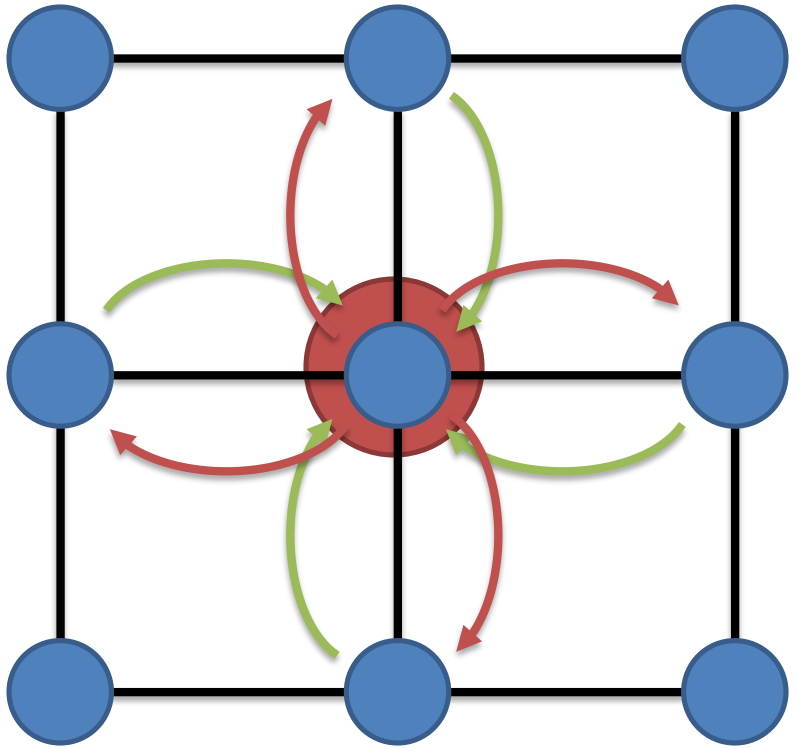
Bulk synchronous computation can be highly inefficient.

Example:

Loopy Belief Propagation

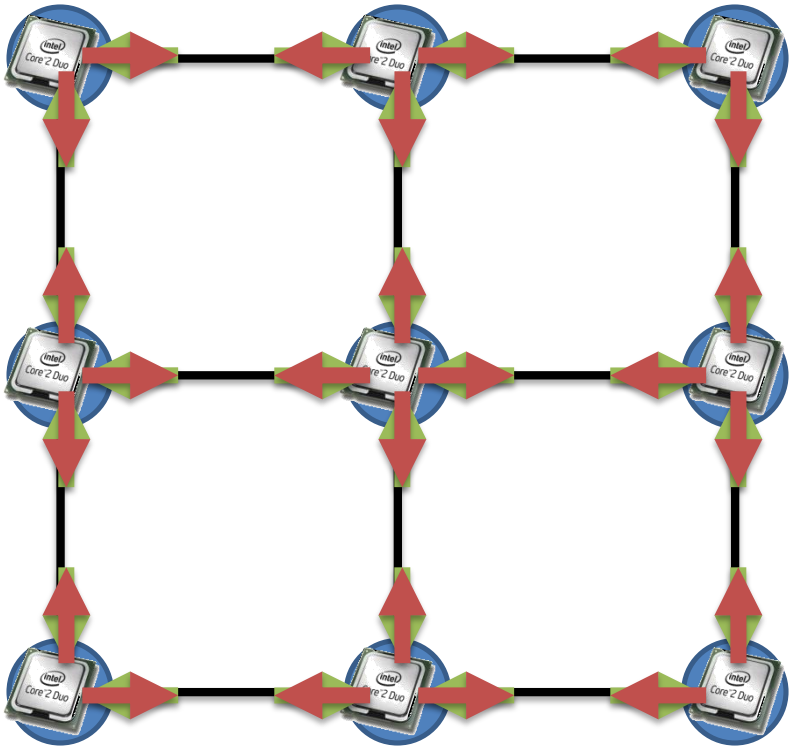
Loopy Belief Propagation (Loopy BP)

- Iteratively estimate the "beliefs" about vertices
  - Read in messages
  - Updates marginal estimate (**belief**)
  - Send updated out messages
- Repeat for all variables until convergence

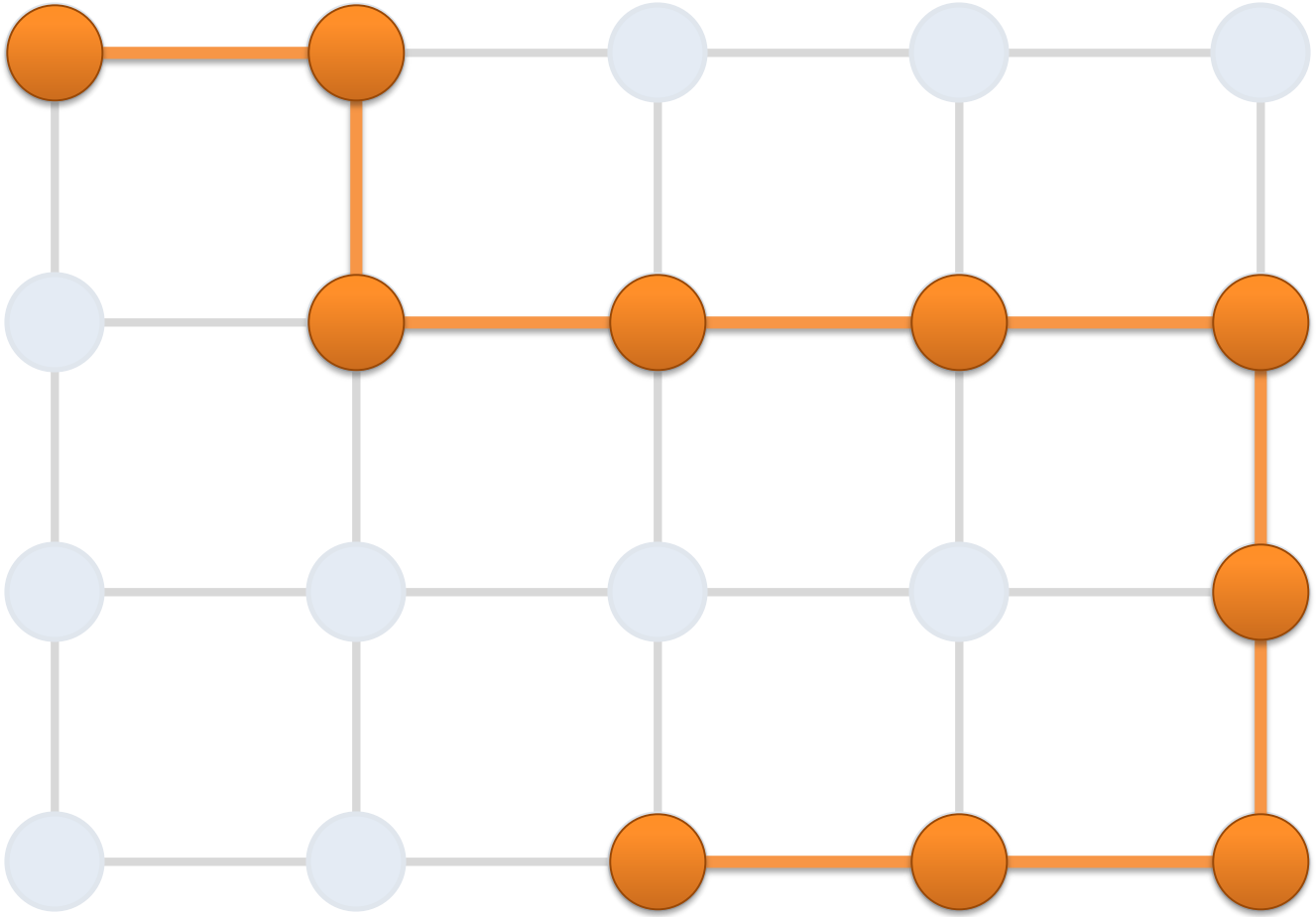


Bulk Synchronous Loopy BP

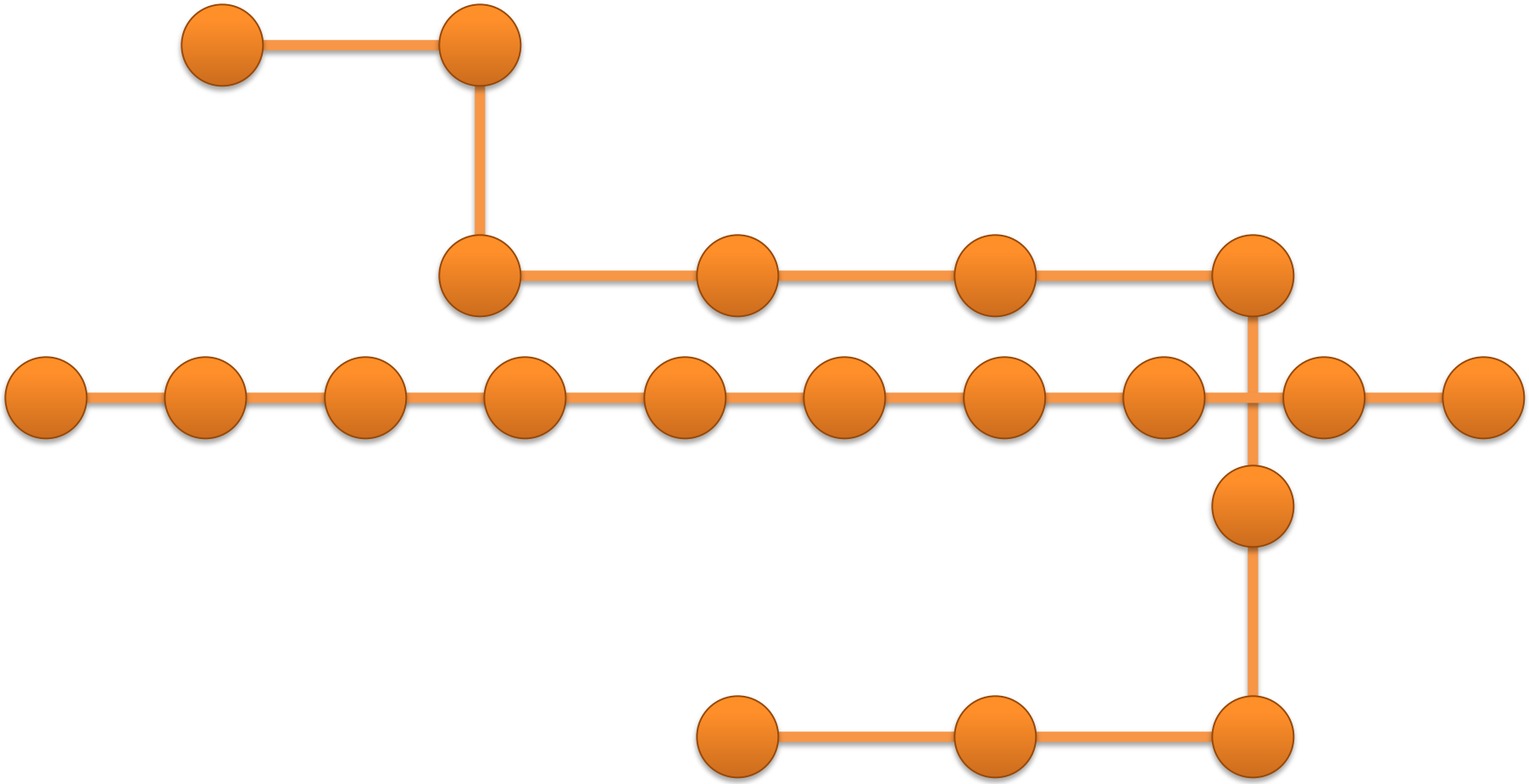
- Often considered embarrassingly parallel
  - Associate processor with each vertex
  - Receive all messages
  - Update all beliefs
  - Send all messages
- Proposed by:
  - Brunton et al. CRV'06
  - Mendiburu et al. GECC'07
  - Kang,et al. LDMTA'10
  - …



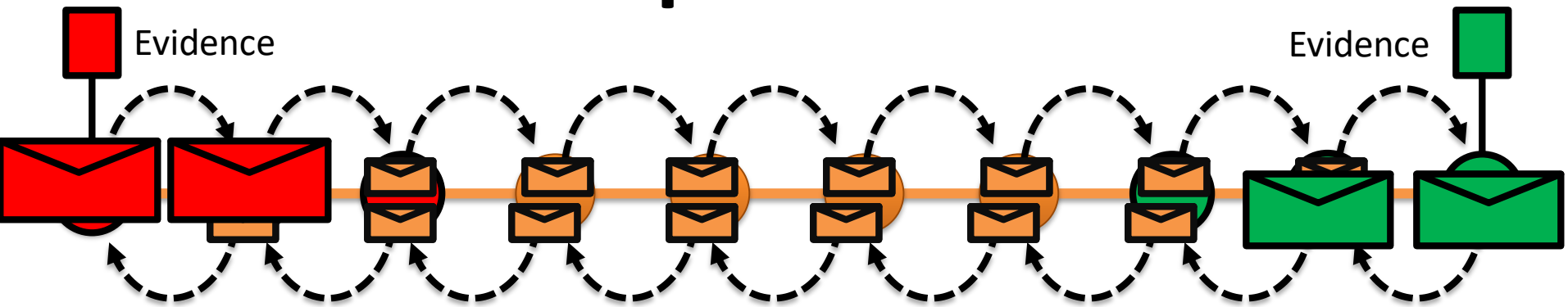
Sequential Computational Structure



Hidden Sequential Structure



Hidden **Sequential** Structure



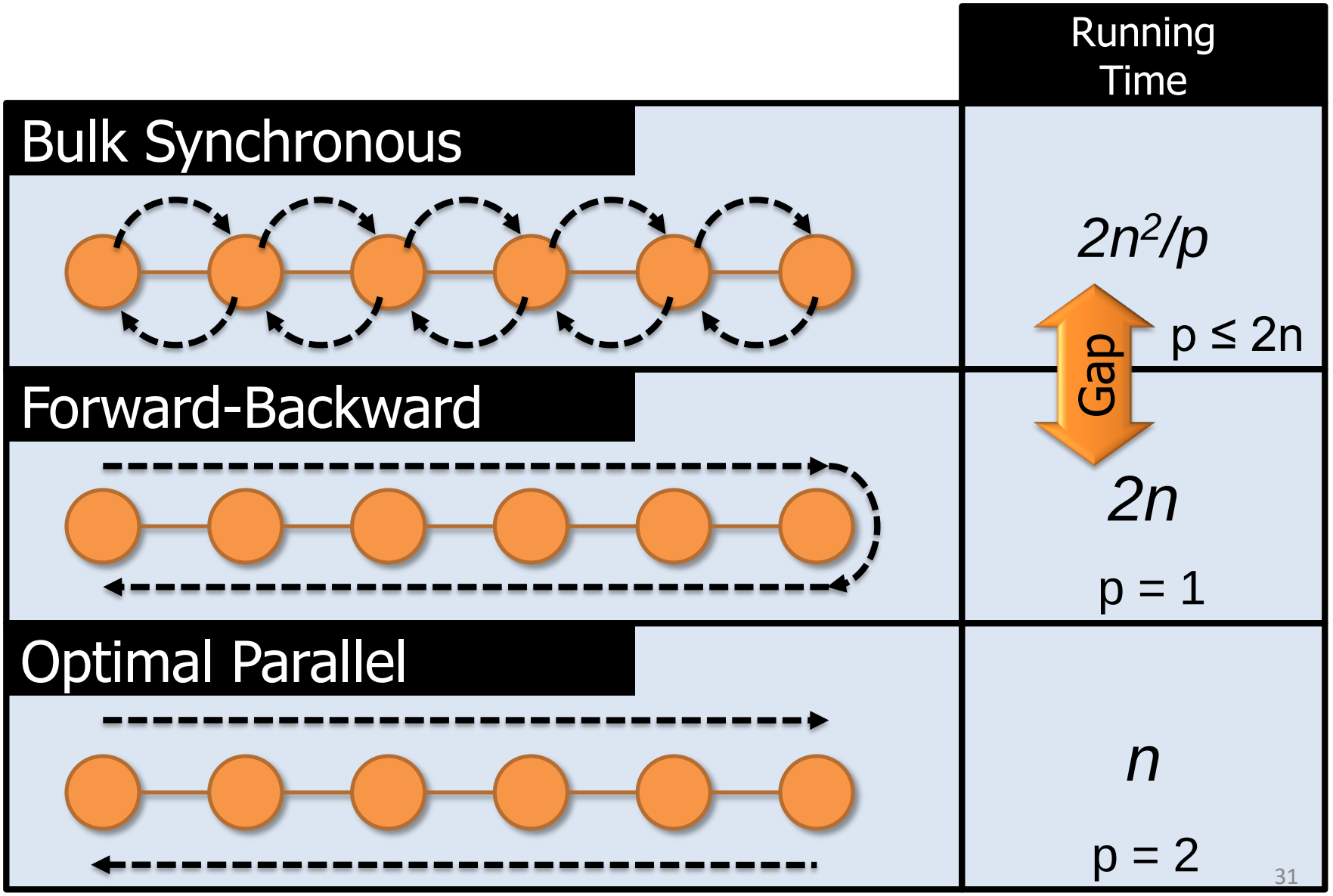
- Running Time:

$$\frac{2n \text{ Messages Calculations}}{p \text{ Processors}} \times (n \text{ Iterations to Converge}) = \frac{2n^2}{p}$$

Time for a single parallel iteration

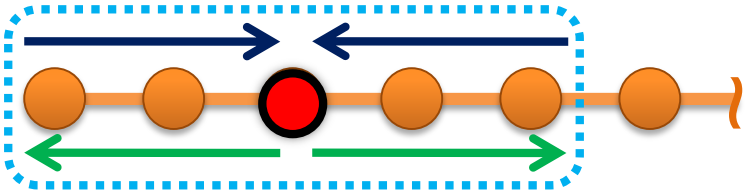
Number of Iterations

Optimal Sequential Algorithm



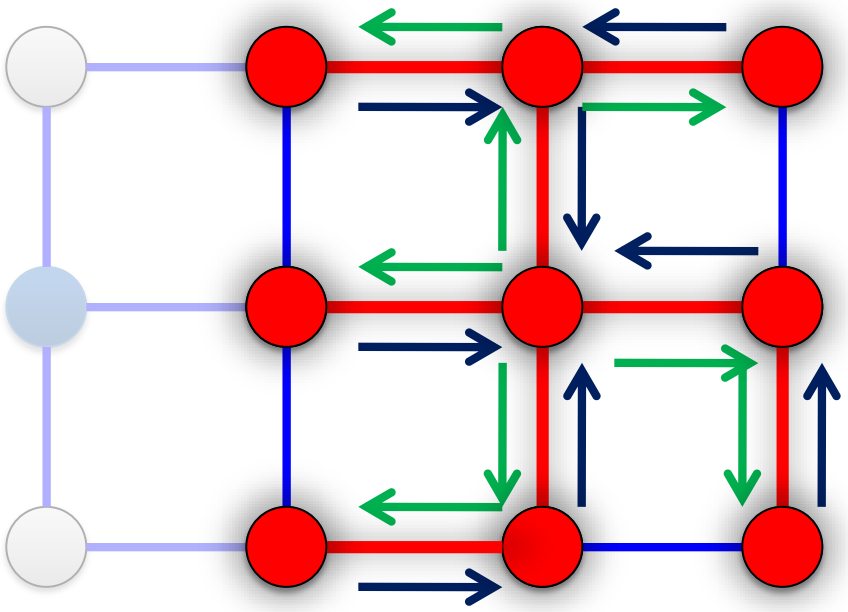
The Splash Operation

- Generalize the optimal chain algorithm:



to arbitrary cyclic graphs:

1) Grow a BFS Spanning tree with fixed size
2) Forward Pass computing all messages at each vertex
3) Backward Pass computing all messages at each vertex



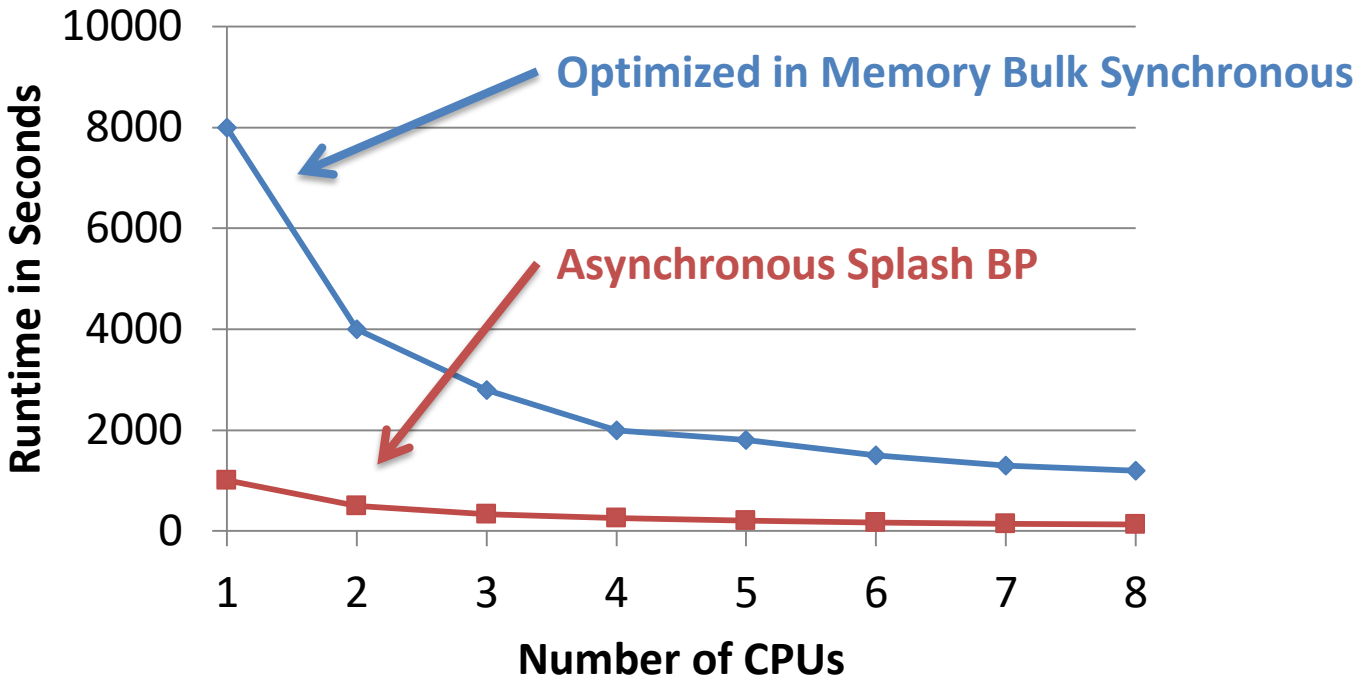
Data-Parallel Algorithms can be **Inefficient**

Residual Splash for Optimally Parallelizing Belief Propagation

Joseph E. Gonzalez
Carnegie Mellon University

Yucheng Low
Carnegie Mellon University

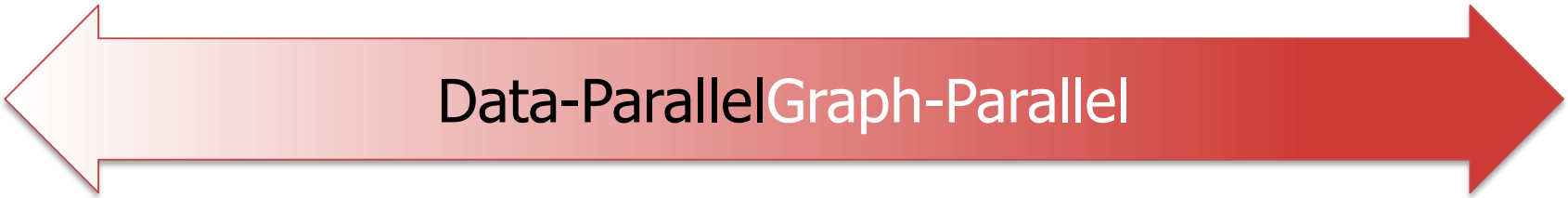
Carlos Guestrin
Carnegie Mellon University



The limitations of the Map-Reduce abstraction can lead to inefficient parallel algorithms.

Map-Reduce for Data-Parallel ML

- Map-Reduce is not well suited for Graph-Parallelism



Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (Giraph)?

Lasso

SVM

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

Deep Belief Networks

Neural Networks

What is GraphLab?

The GraphLab Framework

Graph Based

Data Representation

Update Functions

User Computation

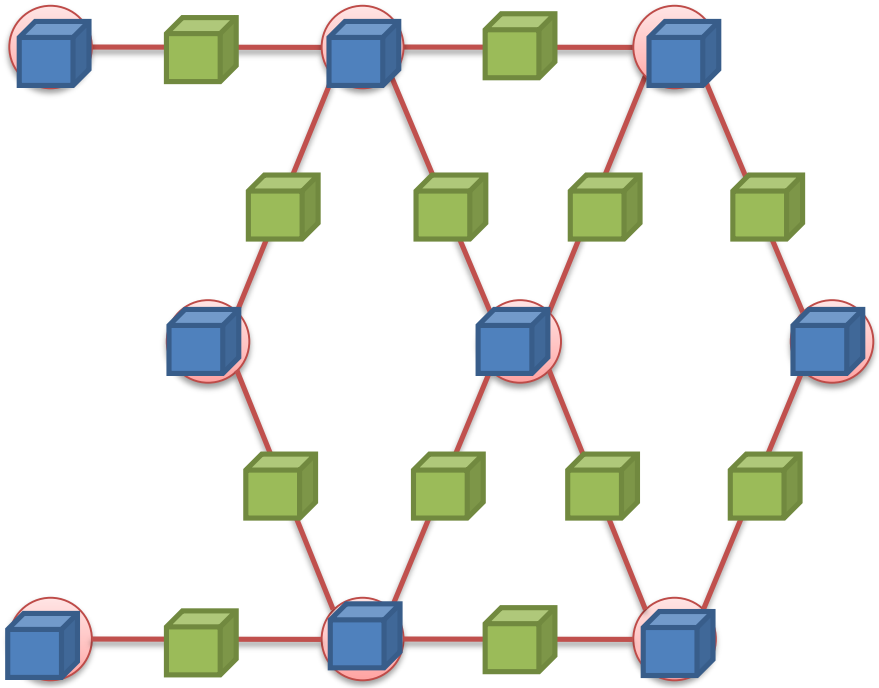
Scheduler

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

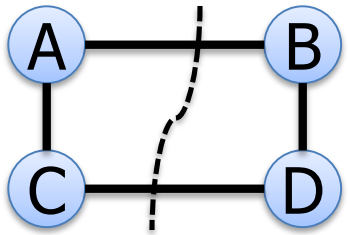
Implementing the Data Graph

Multicore Setting

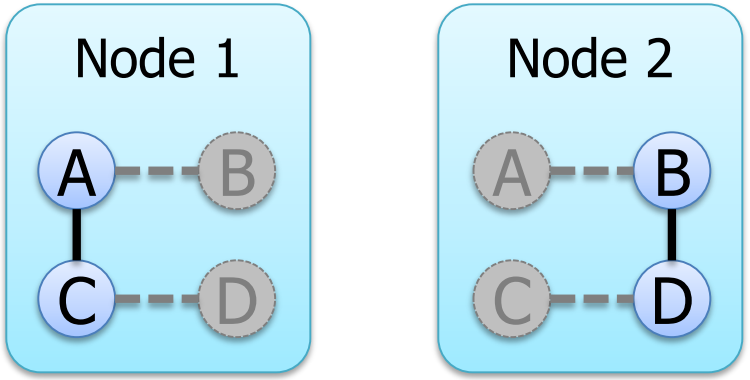
- **In Memory**
- Relatively Straight Forward
  - vertex_data(vid) → data
  - edge_data(vid,vid) → data
  - neighbors(vid) → vid_list
- Challenge:
  - Fast lookup, low overhead
- Solution:
  - Dense data-structures
  - Fixed Vdata&Edata types
  - Immutable graph structure

Cluster Setting

- **In Memory**
- Partition Graph:
  - ParMETIS or Random Cuts



- Cached Ghosting



The GraphLab Framework

Graph Based

Data Representation

Update Functions

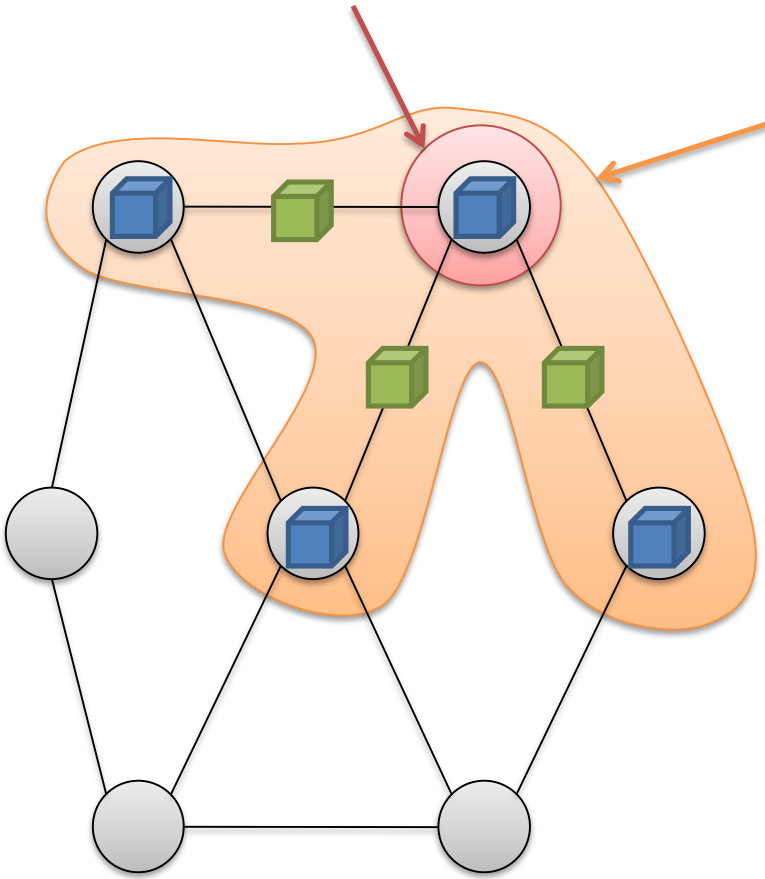
User Computation

Scheduler

Consistency Model

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



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

// Update the vertex data

  Likes[i]¬  å  W_ij ´ Likes[j];
          jî Friends[i]

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

The GraphLab Framework

Graph Based

Data Representation

Update Functions

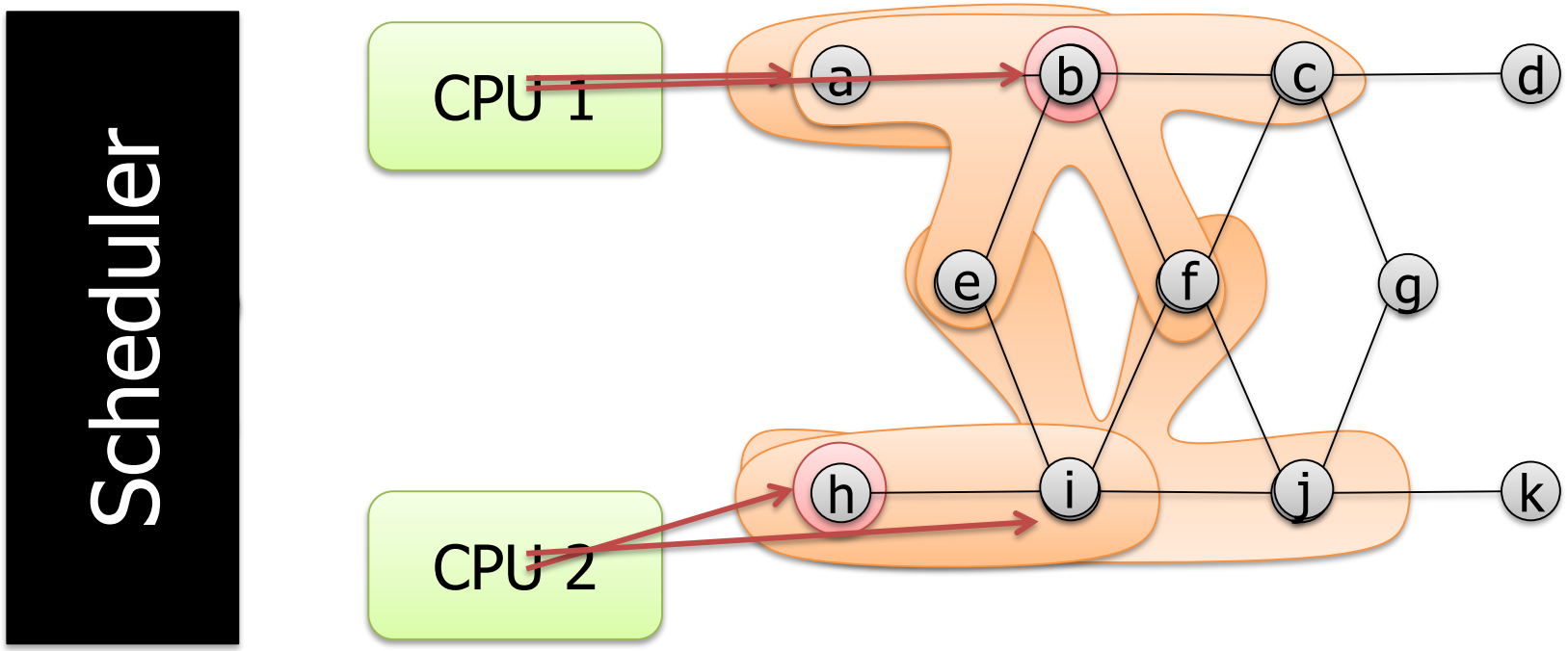
User Computation

Scheduler

Consistency Model

The Scheduler

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

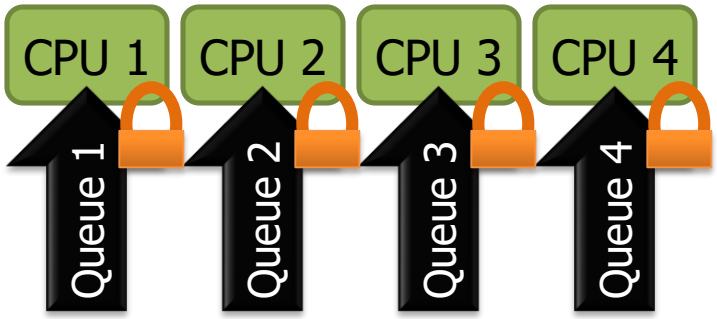


The process repeats until the scheduler is empty.

Implementing the Schedulers

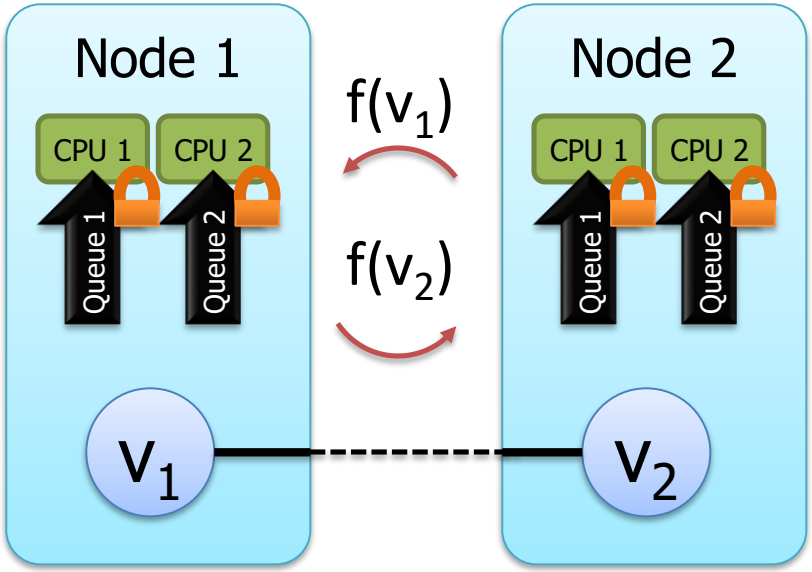
Multicore Setting

- Challenging!
  - Fine-grained locking
  - Atomic operations
- Approximate FiFo/Priority
  - Random placement
  - Work stealing



Cluster Setting

- Multicore scheduler on each node
  - Schedules only “local” vertices
  - Exchange update functions



The GraphLab Framework

Graph Based

Data Representation

Update Functions

User Computation

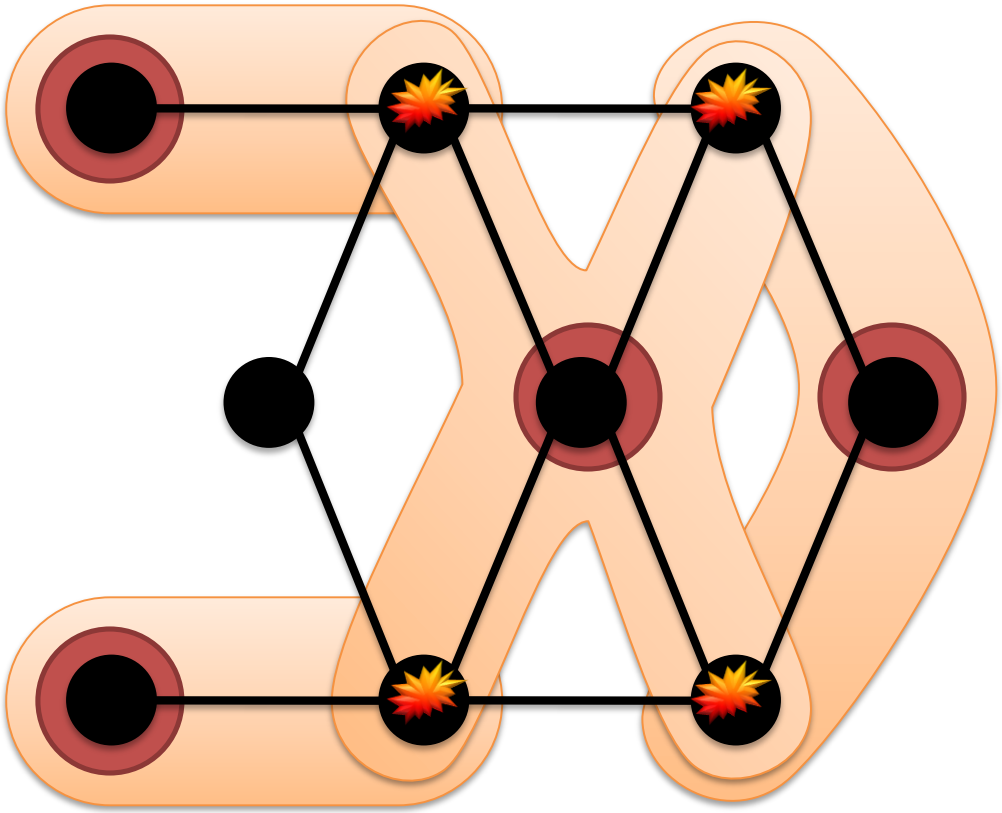
Scheduler

Consistency Model

sense learn act

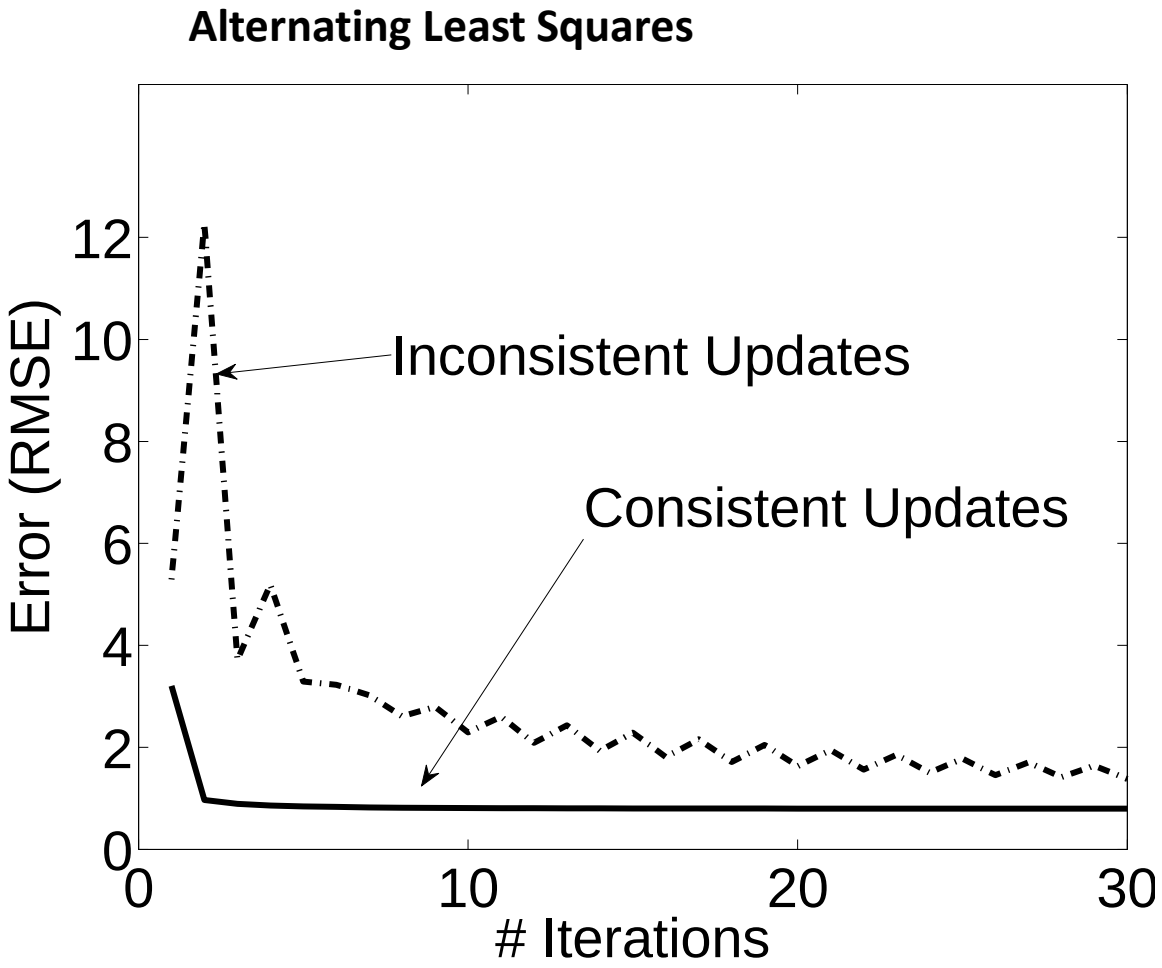
Ensuring Race-Free Code

- How much can computation **overlap**?



Importance of Consistency

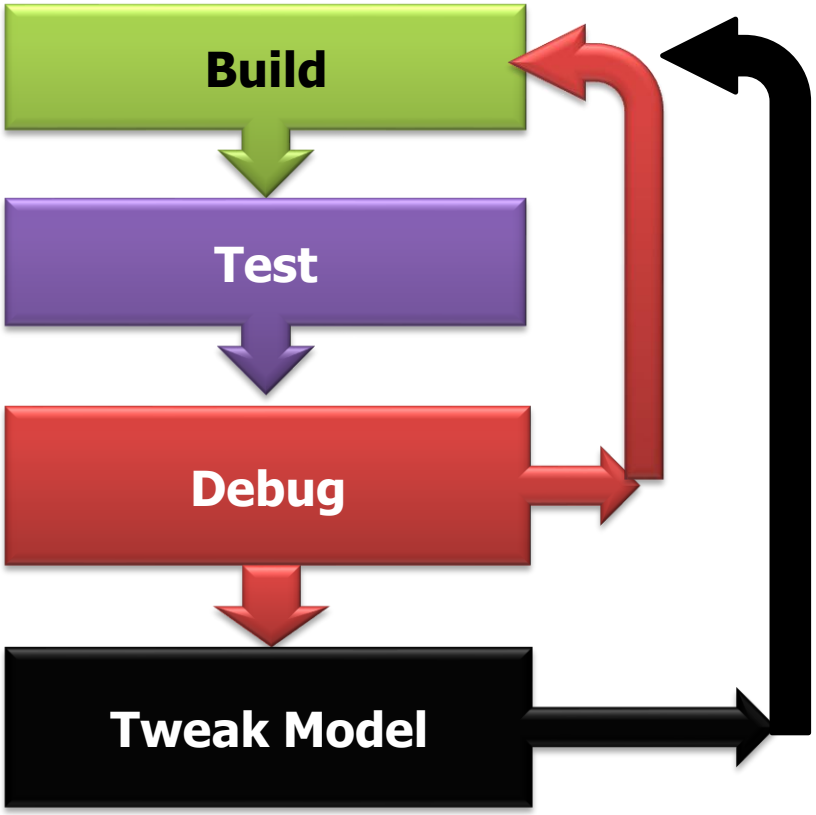
Many algorithms require strict consistency, or performs significantly better under strict consistency.



sense learn act

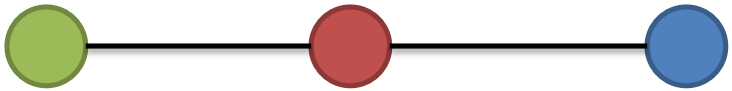
Importance of Consistency

Machine learning algorithms require “model debugging”



GraphLab Ensures **Sequential Consistency**

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



Parallel

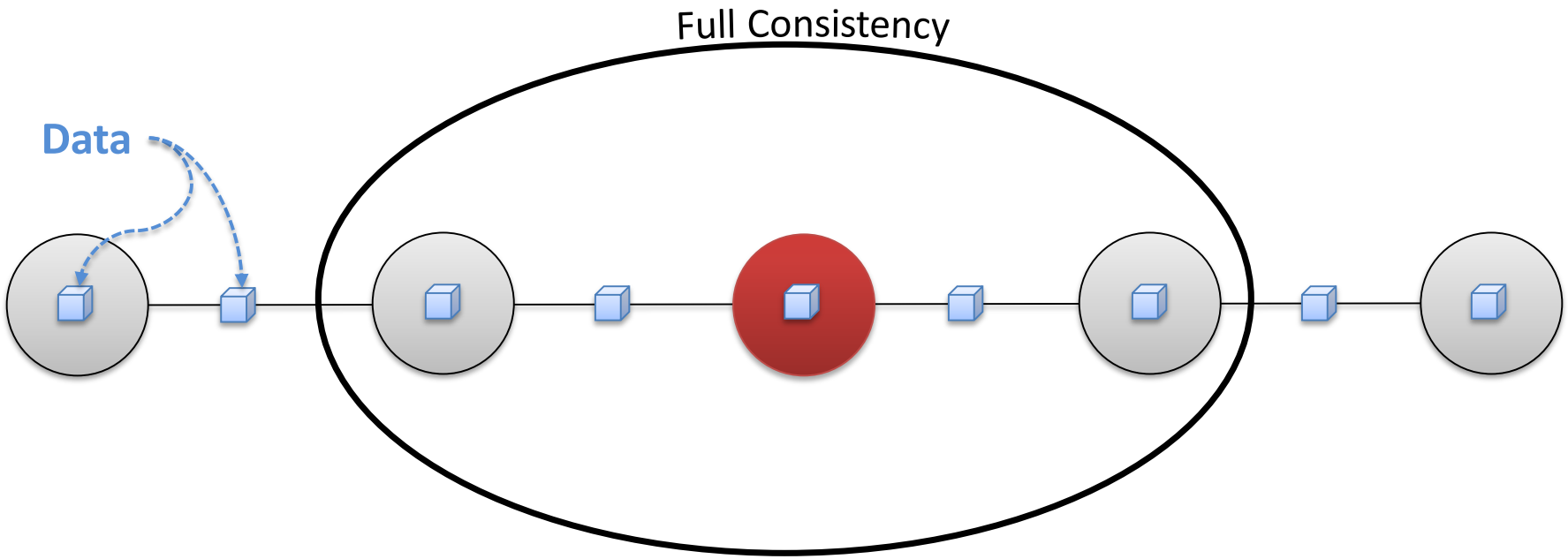
CPU 1

CPU 2

Sequential

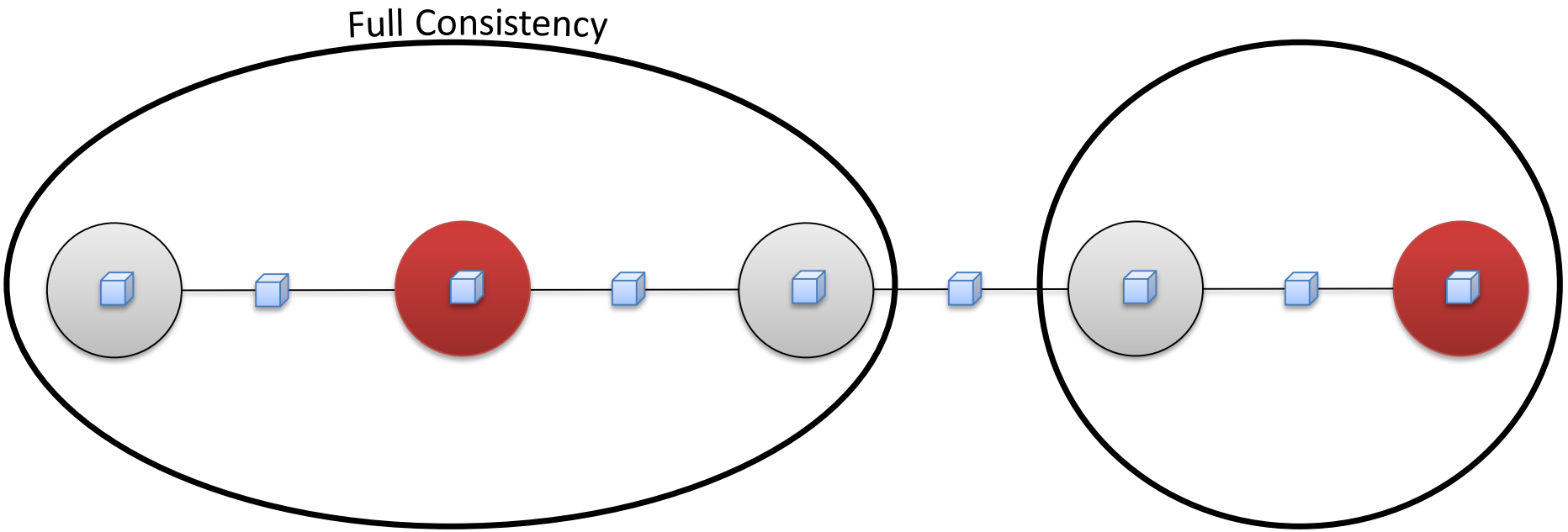
Single CPU

Consistency Rules

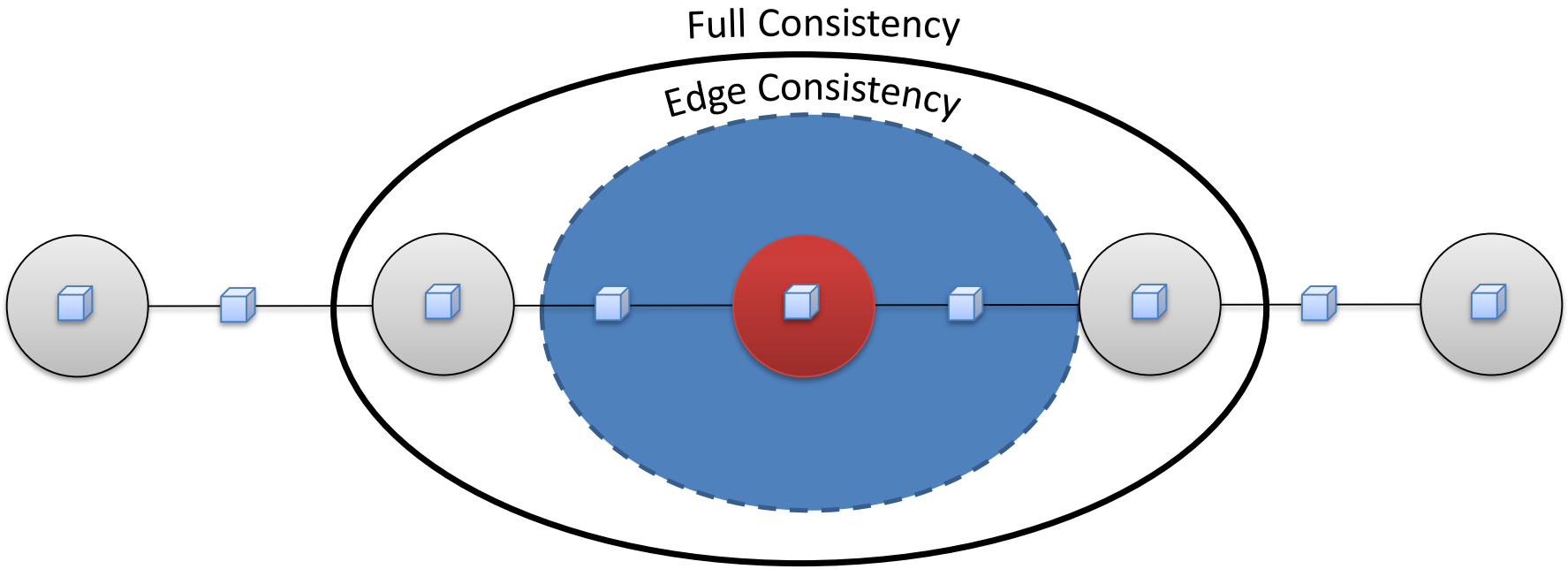


Guaranteed sequential consistency for all update functions

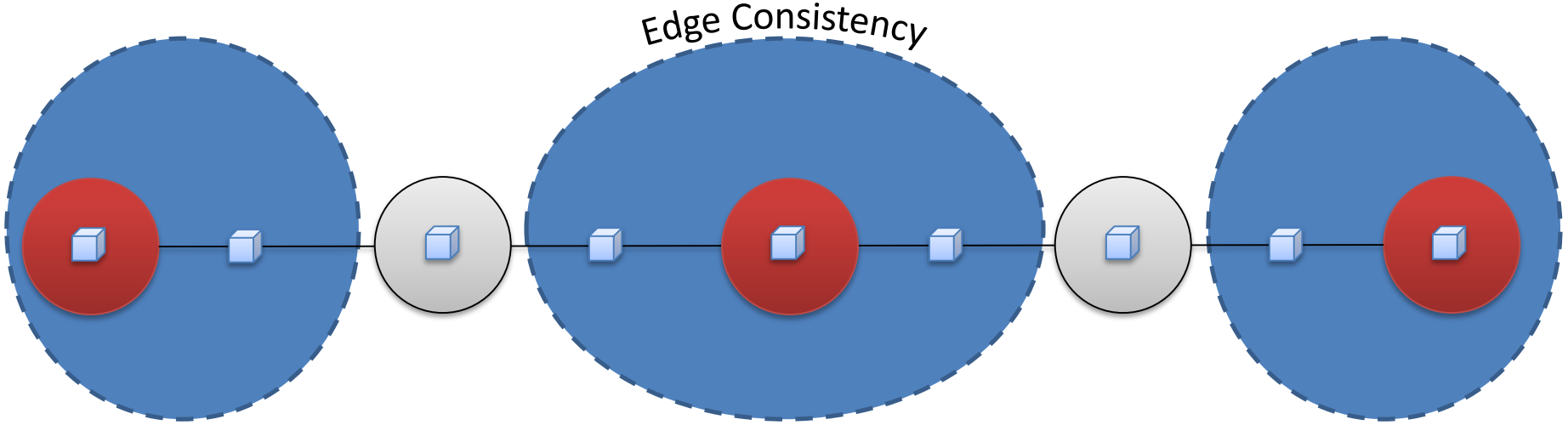
Full Consistency



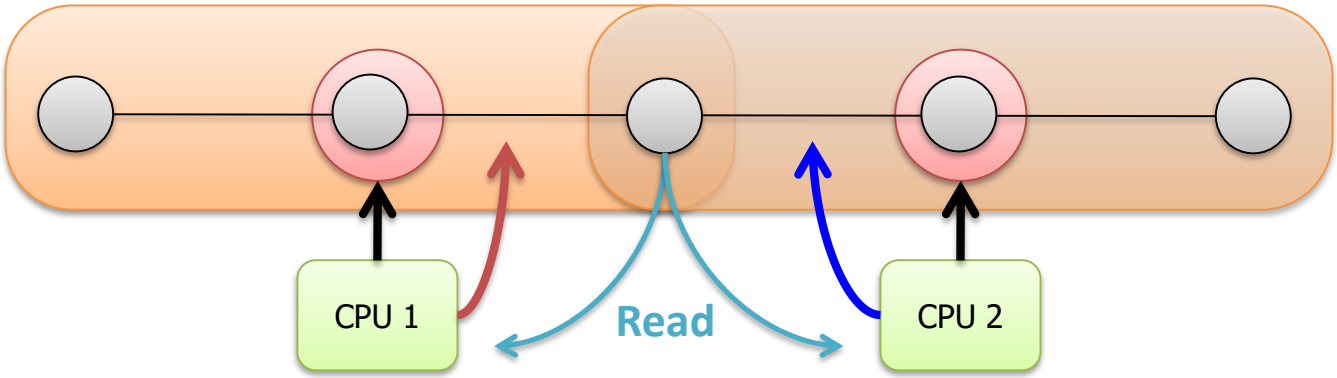
Obtaining More Parallelism

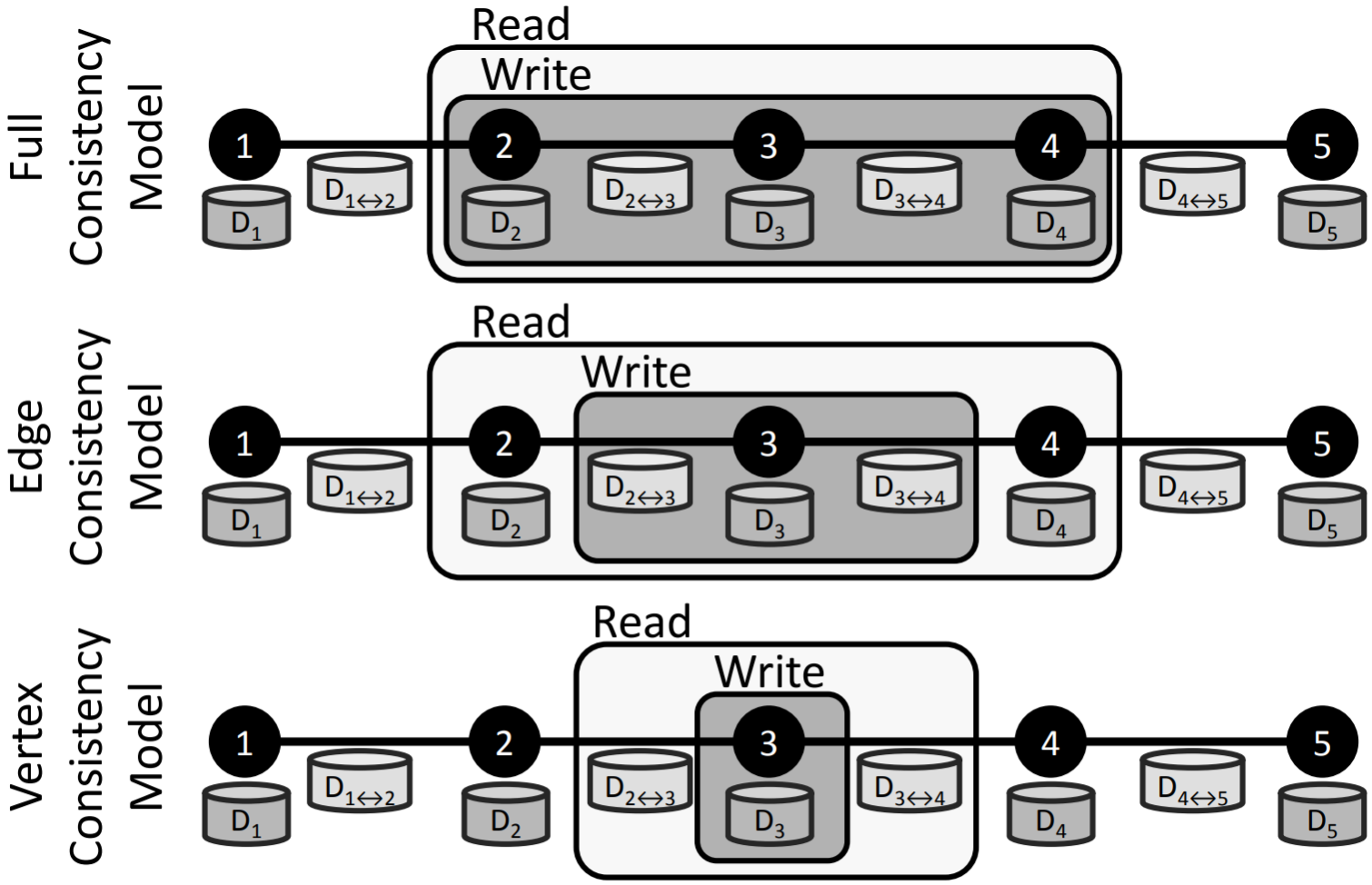


Edge Consistency



Safe





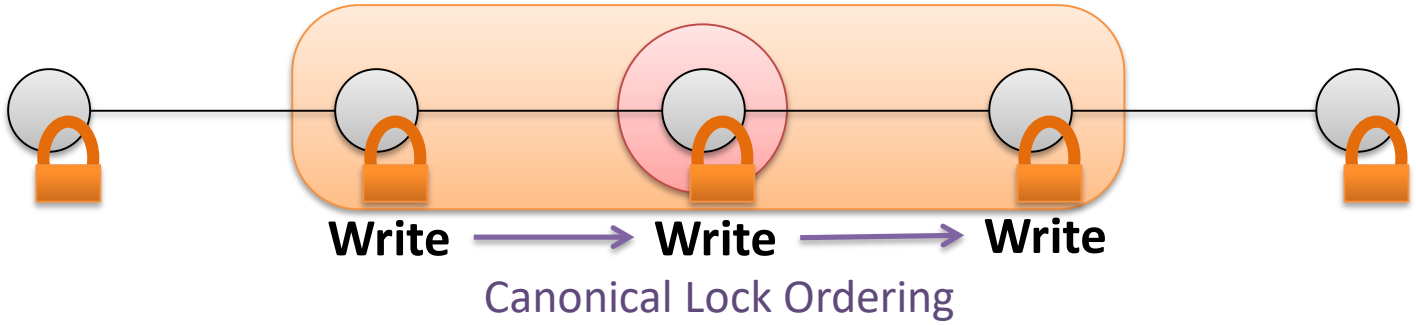
(b) Consistency Models

Added by Kesden from: https://arxiv.org/pdf/1204.6078.pdf

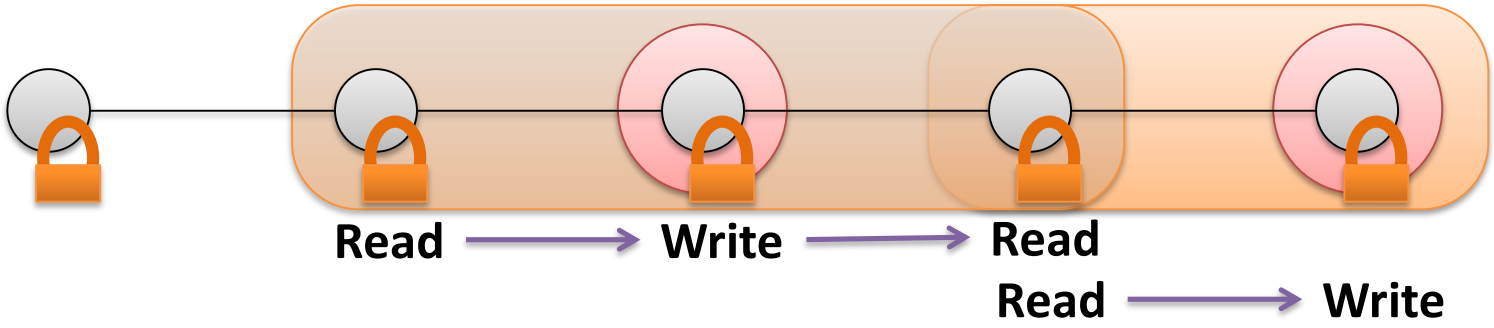
sense learn act

Consistency Through R/W Locks

- Read/Write locks:
  - Full Consistency



  - Edge Consistency



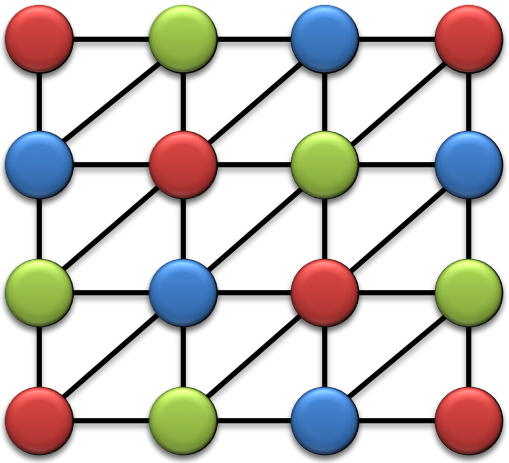
Consistency Through R/W Locks

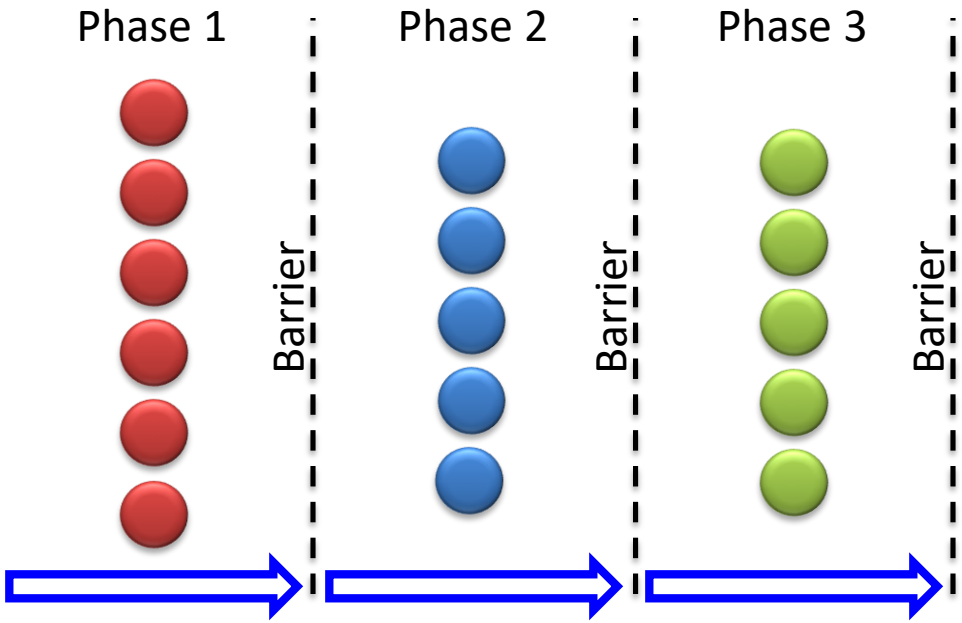
- Multicore Setting: Pthread R/W Locks
- Distributed Setting: *Distributed Locking*
  - Prefetch Locks and Data

  - Allow computation to proceed while locks/data are requested.

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.



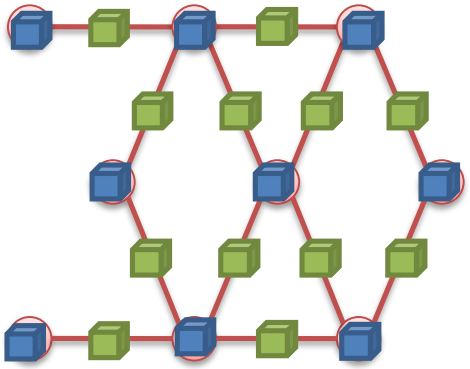


sense learn act

The GraphLab Framework

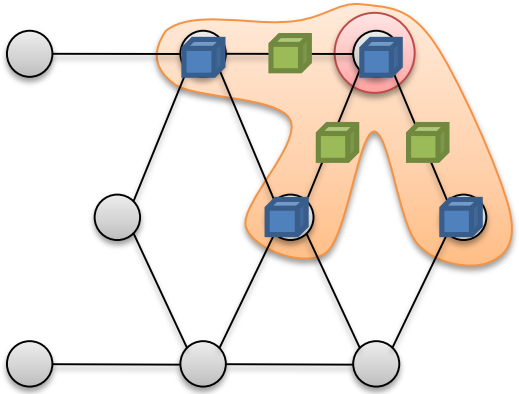
Graph Based

Data Representation

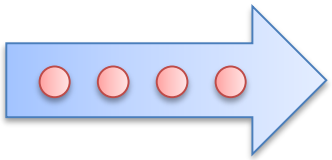


Update Functions

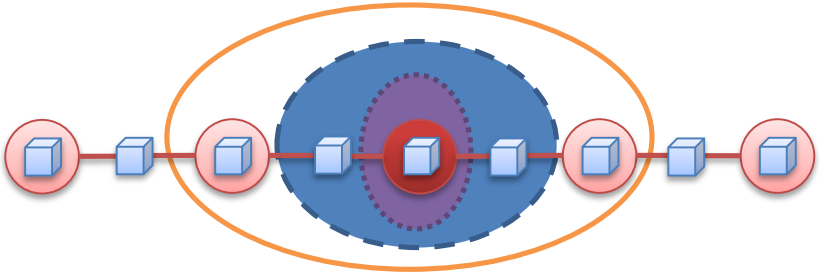
User Computation



Scheduler



Consistency Model



sense learn act

Algorithms Implemented

- PageRank
- Loopy Belief Propagation
- Gibbs Sampling
- CoEM
- Graphical Model Parameter Learning
- Probabilistic Matrix/Tensor Factorization
- Alternating Least Squares
- Lasso with Sparse Features
- Support Vector Machines with Sparse Features
- Label-Propagation
- …

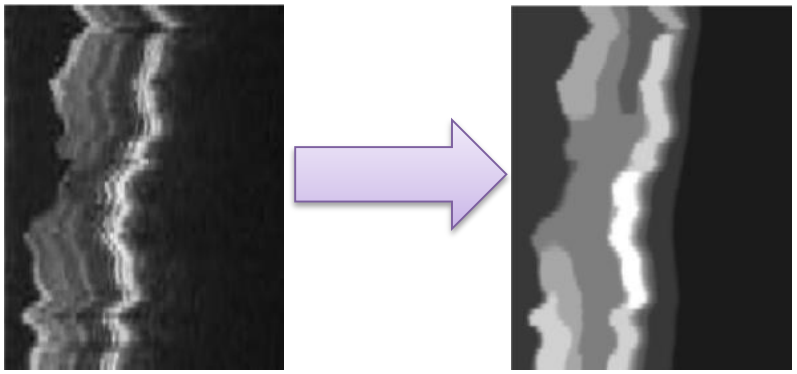
Shared Memory Experiments

Shared Memory Setting

16 Core Workstation

Loopy Belief Propagation

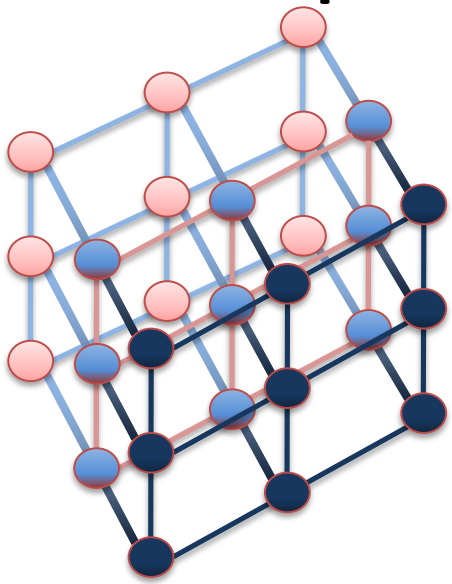
3D retinal image denoising



Vertices: 1 Million

Edges: 3 Million

Data Graph



Update Function:

Loopy BP Update Equation

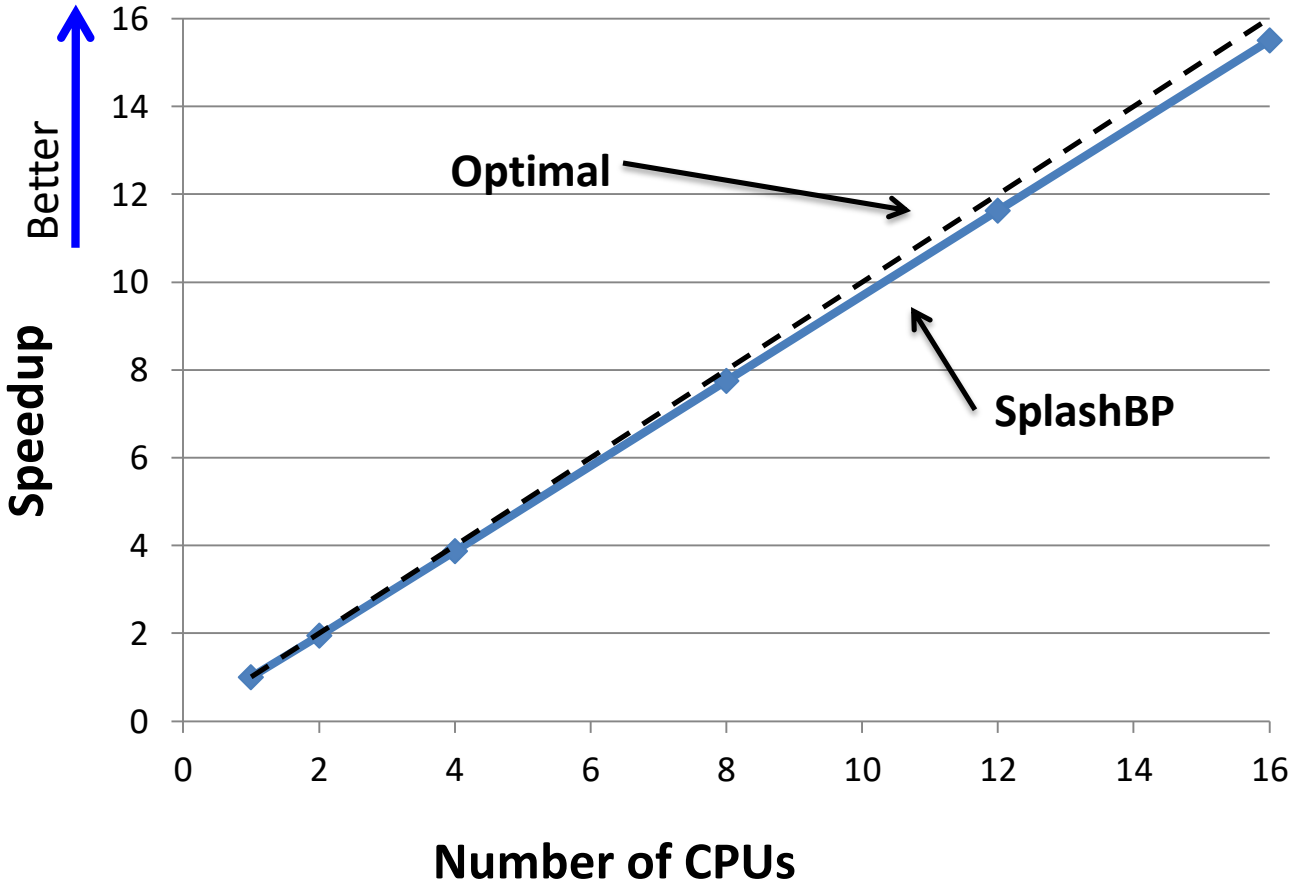
Scheduler:

Approximate Priority

Consistency Model:

Edge Consistency

Loopy Belief Propagation



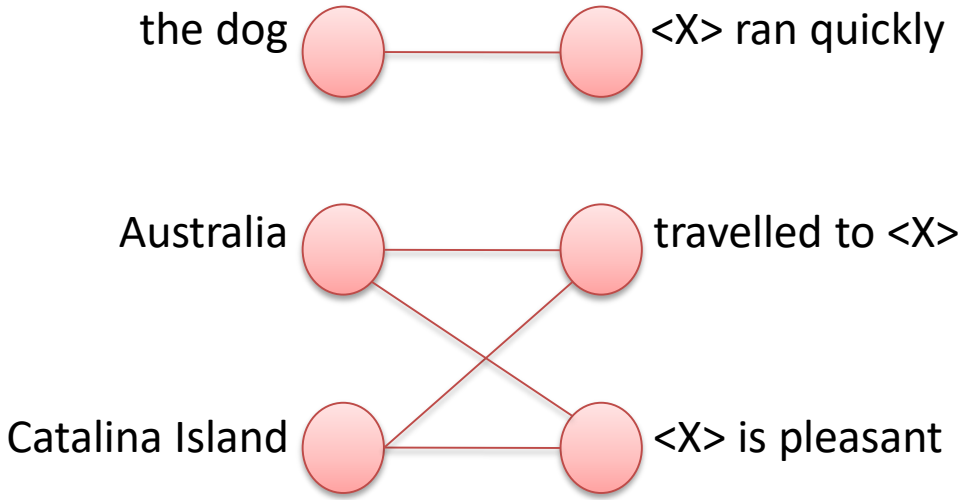
15.5x speedup

CoEM (Rosie Jones, 2005)

Named Entity Recognition Task

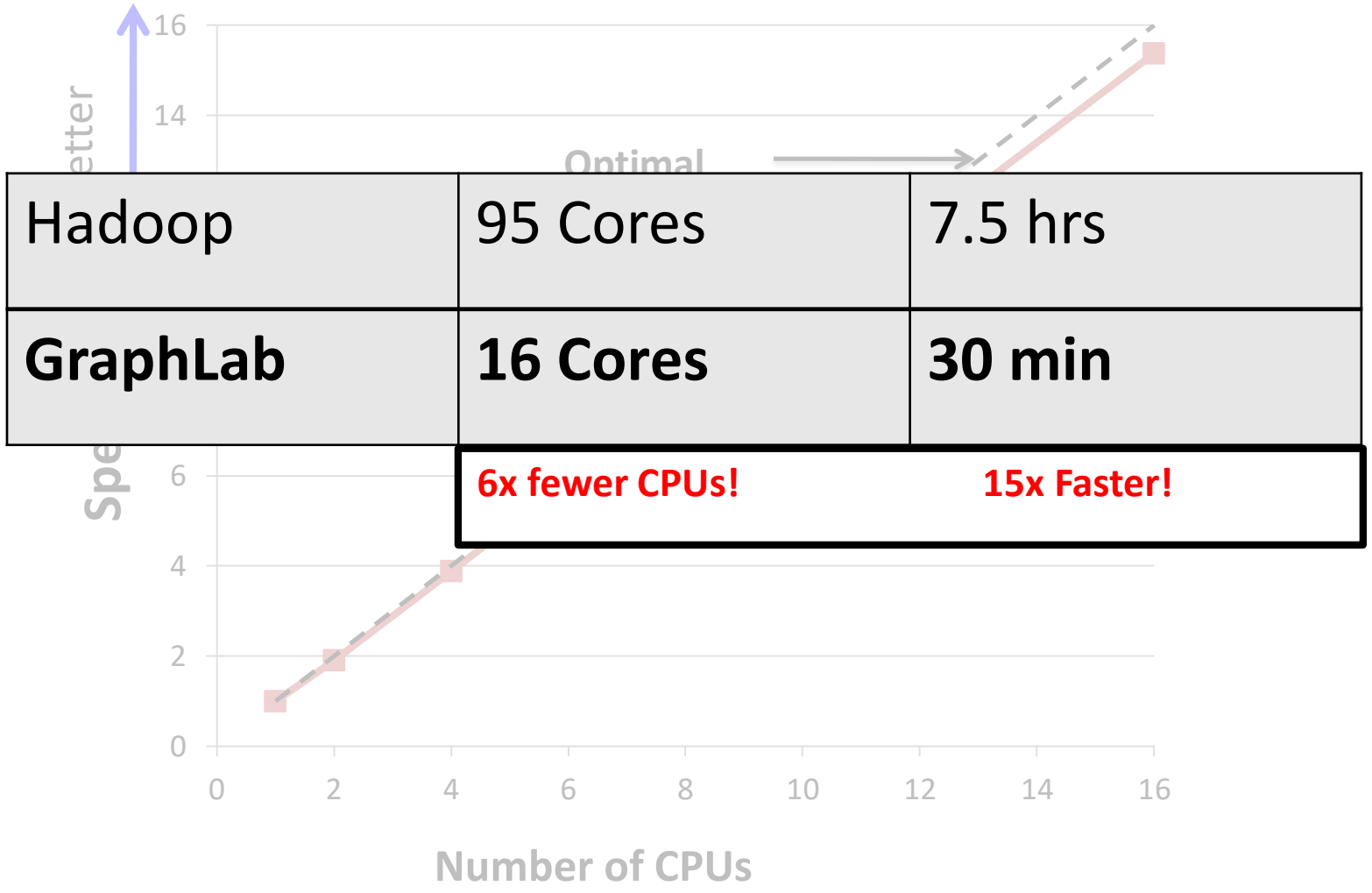
Is “Dog” an animal?
Is “Catalina” a place?

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**

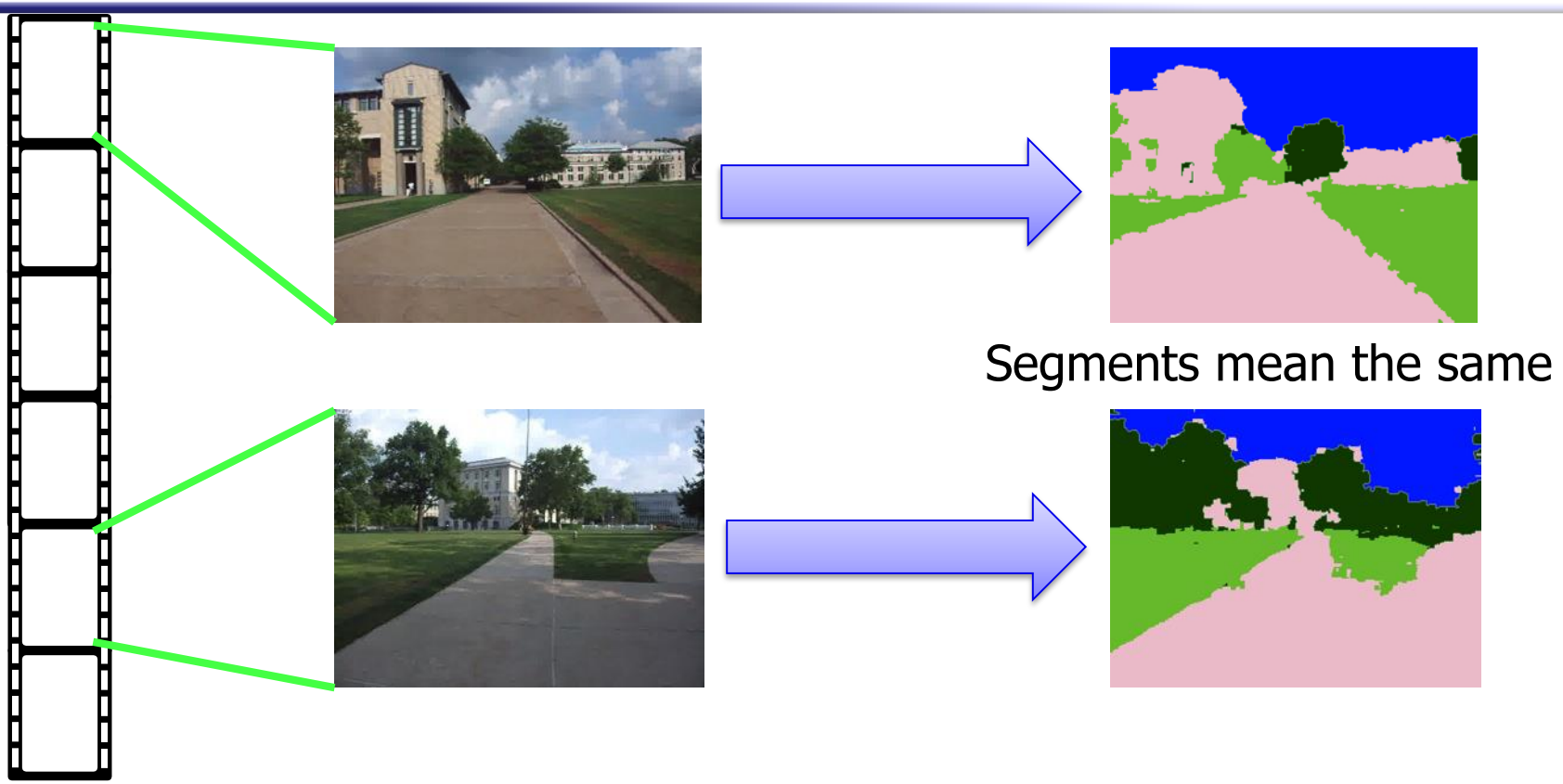
6x fewer CPUs! **15x Faster!**

Experiments

Amazon EC2
High-Performance Nodes

sense learn act

Video Cosegmentation



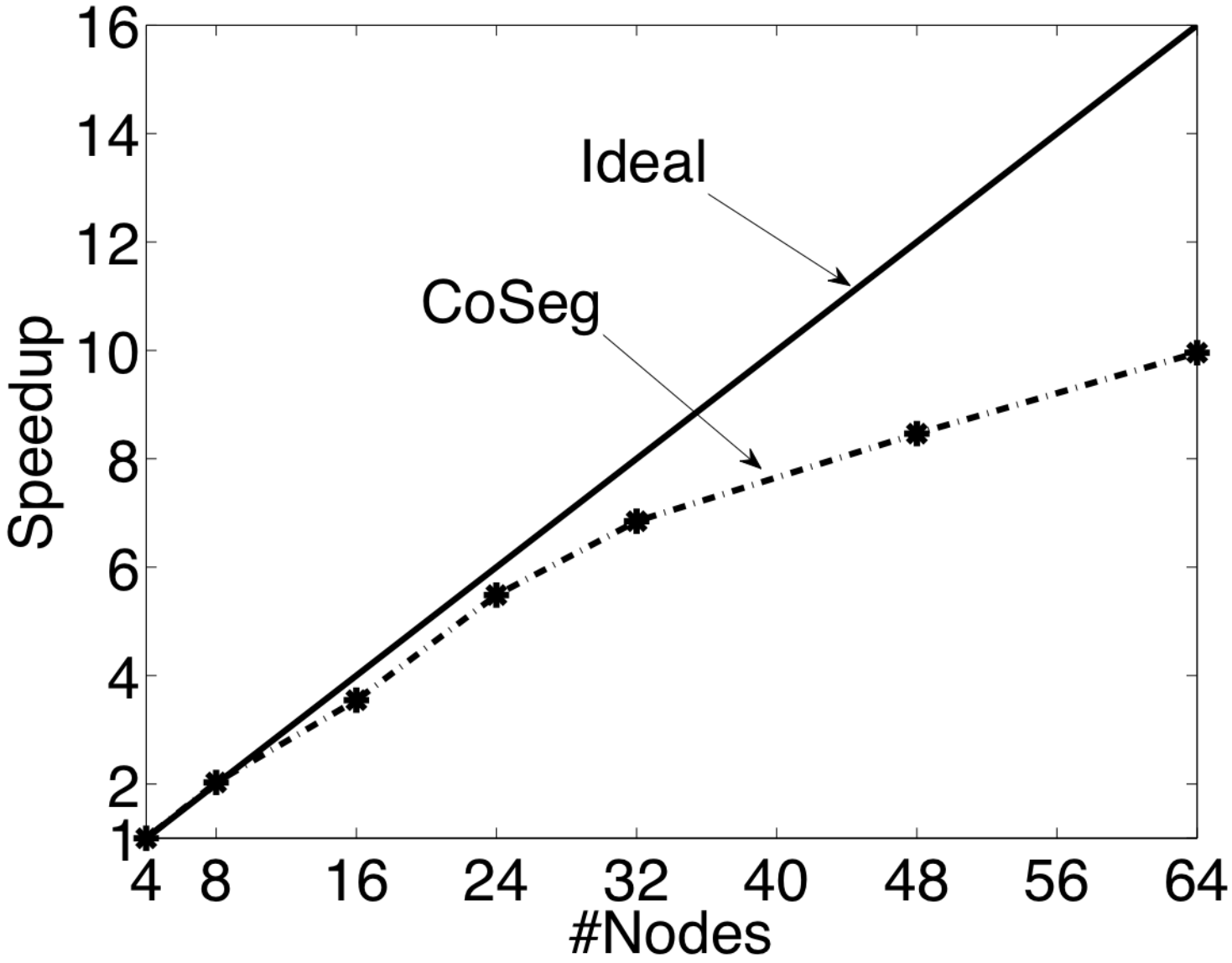
Segments mean the same

Gaussian EM clustering + BP on 3D grid

Model: 10.5 million nodes, 31 million edges

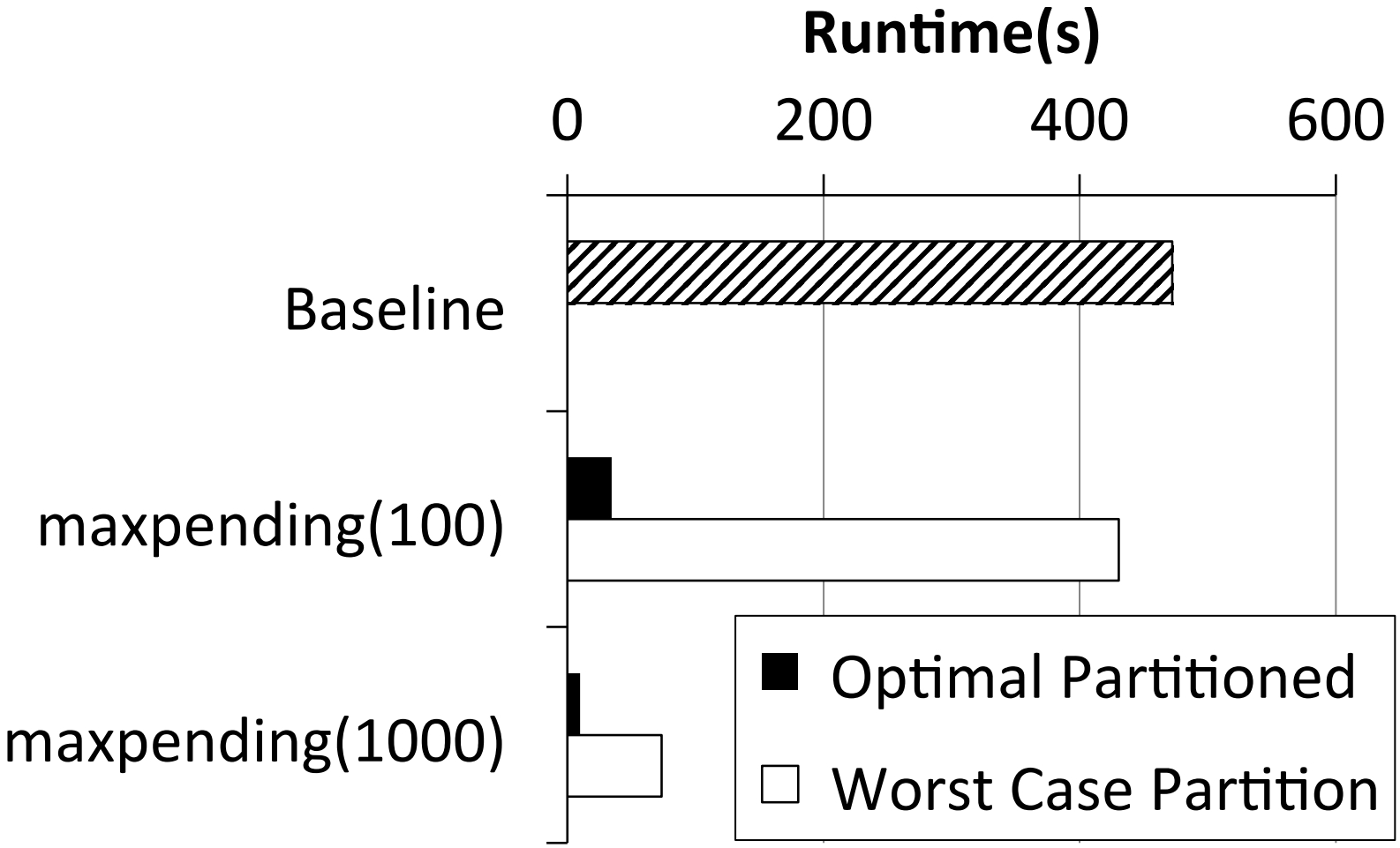
sense learn act

Video Coseg. Speedups



sense learn act

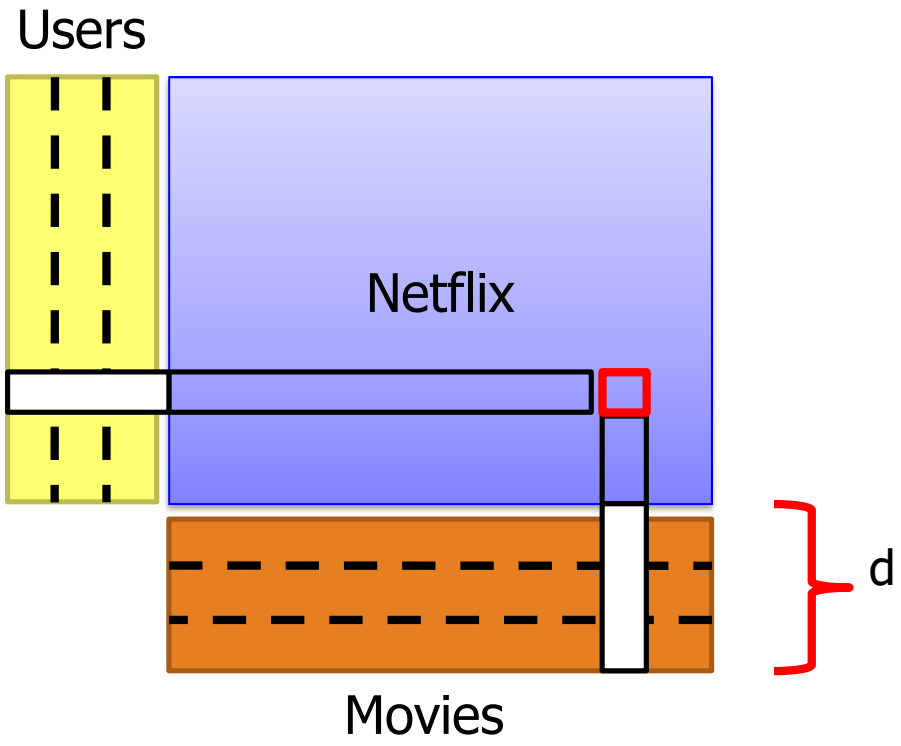
Prefetching Data & Locks



Matrix Factorization

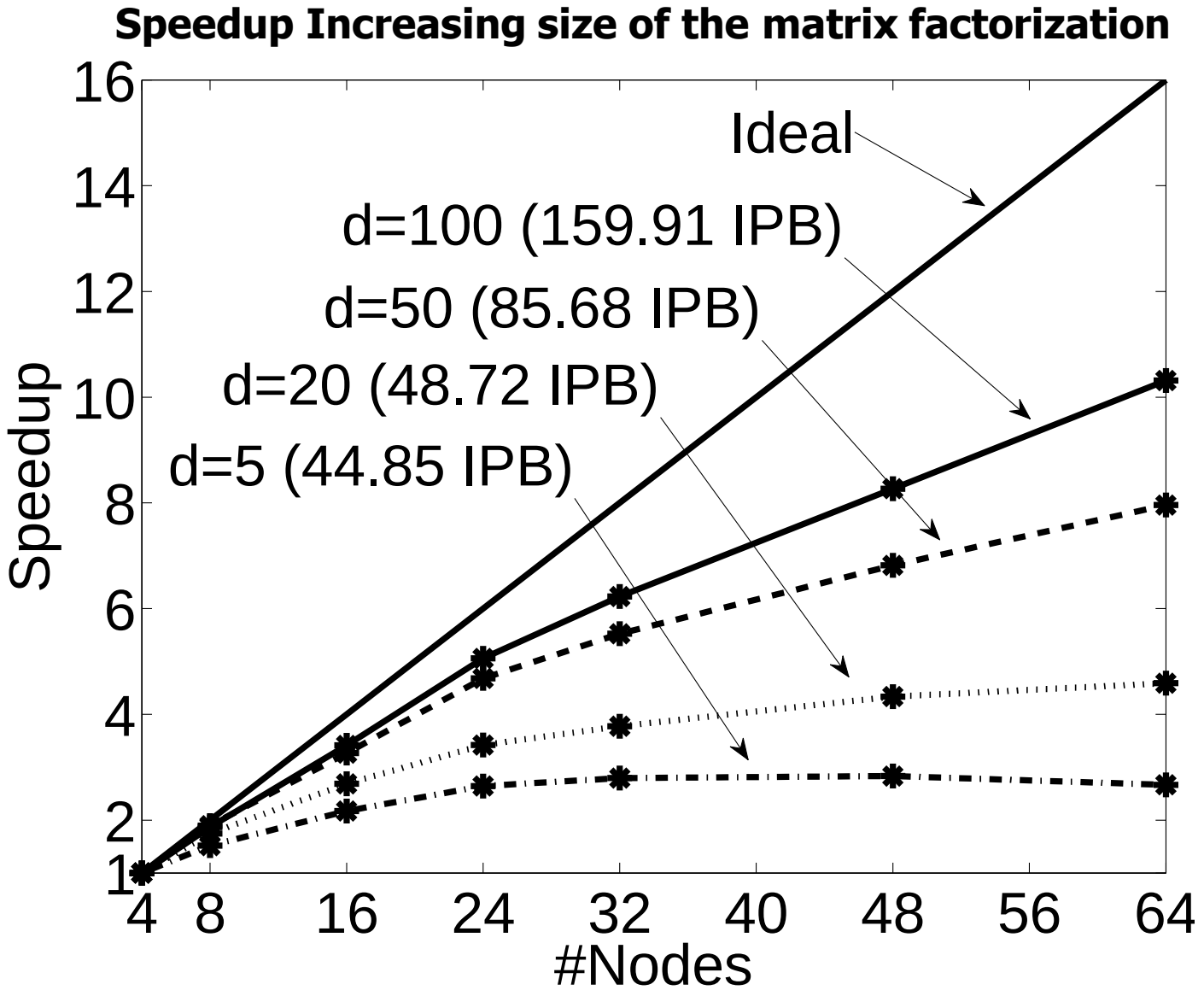
- **Netflix Collaborative Filtering**
  - Alternating Least Squares Matrix Factorization

Model: 0.5 million nodes, 99 million edges

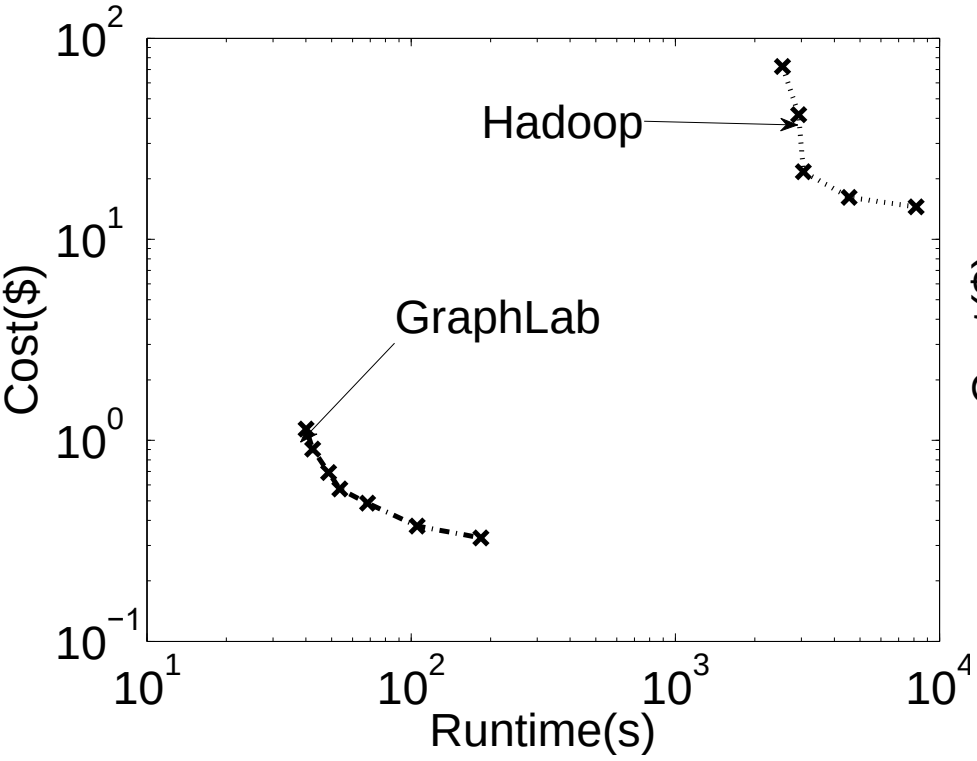


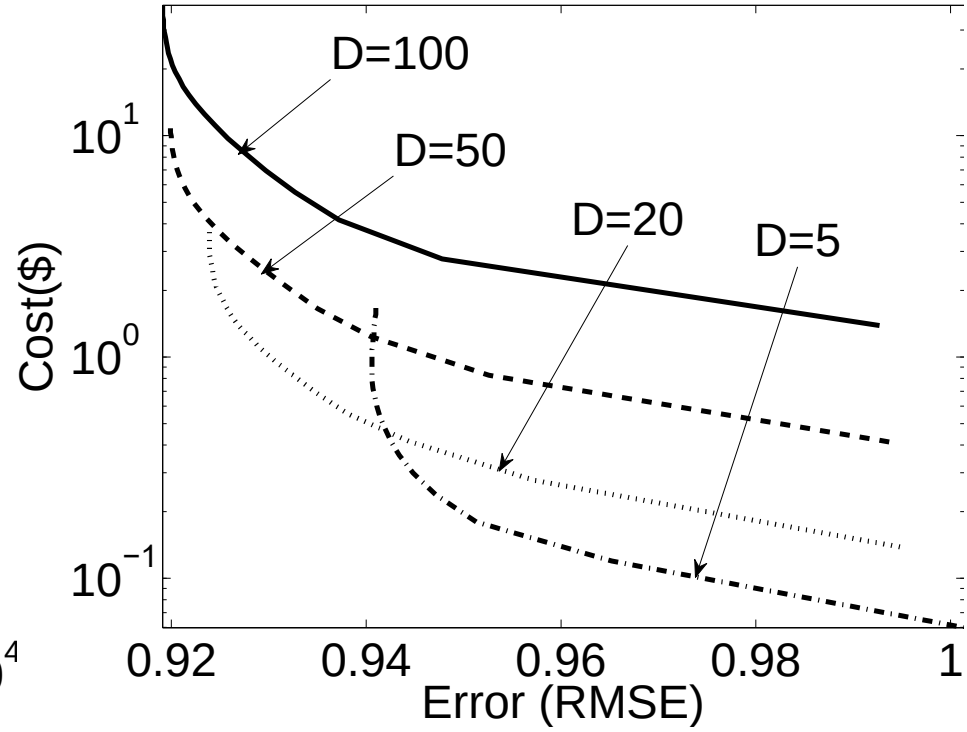
sense learn act

Netflix



The Cost of Hadoop





Summary

- An abstraction tailored to Machine Learning
  - Targets Graph-Parallel Algorithms
- Naturally expresses
  - Data/computational dependencies
  - Dynamic iterative computation
- Simplifies parallel algorithm design
- Automatically ensures data consistency
- Achieves state-of-the-art parallel performance on a variety of problems

Checkout **GraphLab**

Documentation… Code… Tutorials…

http://graphlab.org

Questions & Feedback

jegonzal@cs.cmu.edu

sense learn act

Current/Future Work

- Out-of-core Storage
- Hadoop/HDFS Integration
  - Graph Construction
  - Graph Storage
  - Launching GraphLab from Hadoop
  - Fault Tolerance through HDFS Checkpoints
- Sub-scope parallelism
  - Address the challenge of very high degree nodes
- Improved graph partitioning
- Support for dynamic graph structure