Parallel Machine Learning

System on data science

Kaicheng Yu

2019/03/25

Content

- Two papers to do parallel ML system.

- Hogwild!
  - Asynchronized stochastic gradient descent
- Graph Lab:
  - A graph abstraction of machine learning problem
  - Enable parallelization via serialization
  - Provide real-world examples

Hogwild!

HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent

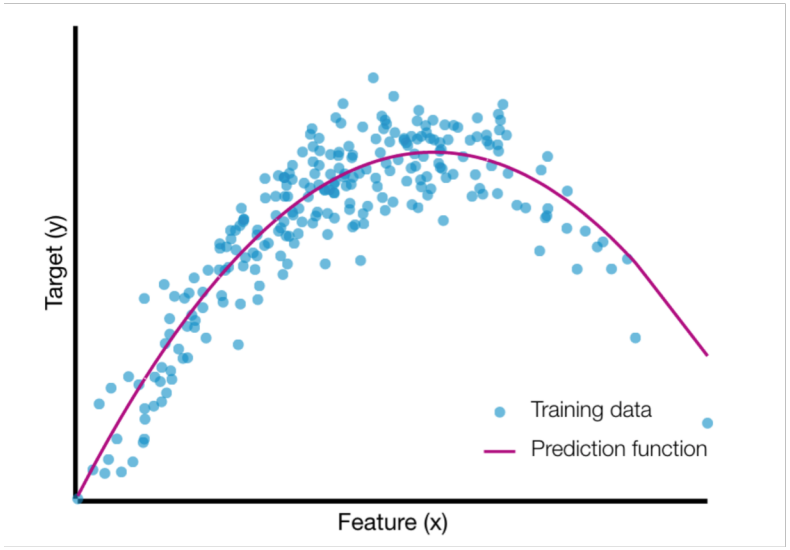
Feng Niu, Benjamin Recht, Christopher Ré and Stephen J. Wright
Computer Sciences Department, University of Wisconsin-Madison
1210 W Dayton St, Madison, WI 53706

June 2011

- Asynchronized stochastic gradient descent
- Implemented in famous ML platform
  - Pytorch
  - Tensorflow
  - Node2vec
  - Cited over 1200 times

Before Hogwild

- What's Stochastic Gradient Descent?
- Gradient descent is an optimization method for
  - Logistic regression
  - Support vector machine
  - Matrix factorization
  - Neural networks
- Goal is to learn:
  - Best fit of prediction f (Pink)
  - Given based data



Loss function

- Loss function
  - is to capture the quality of model
  - based on an error metric (or multiple ones)
  - usually, is formulated as continuous function
  - map from observation data (x)
    to predicted labels (y)
- Statistics

$$\theta^* = \arg\min \sum_{i=1}^{N} L(f_\theta(x_i), y_i)$$

Best model

Loss function

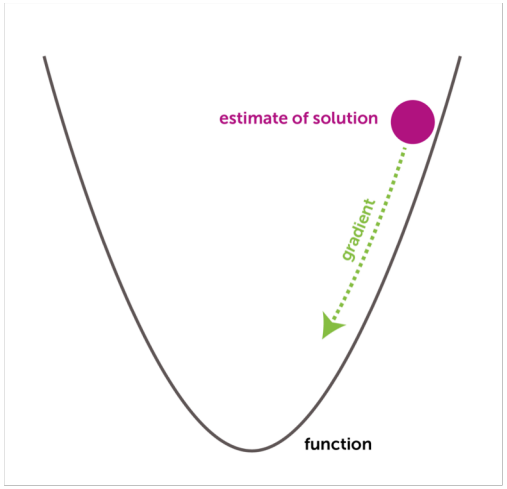
Model fitting

How to minimize loss function?

- Many solutions, but we focus on Gradient descent
- A greedy math algorithm to find minima.
  - Use the gradient direction as estimation of func.
  - Decrease
  - Repeat until convergence
- Mathematically,
  - Update function

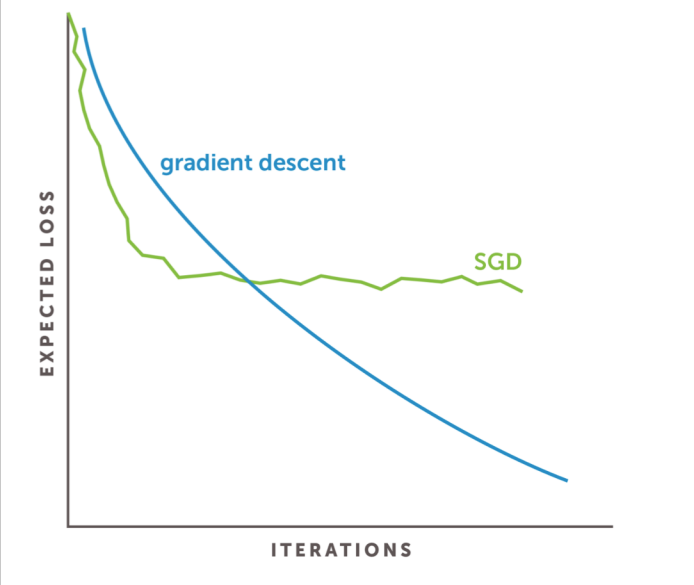
$$\theta_{t+1} = (\theta_t - \alpha \nabla L(f_\theta(x_i), y_i))$$

  - α - Step size, define the speed
  - $\nabla L = \sum \nabla L(x_i, y_i)$ over entire dataset



Stochastic gradient descent

- Gradient is computed from entire dataset
  - On small dataset, is doable and accurate
  - On larger dataset, is very tricky
- Stochastic approach:
  - Only estimate on a small subset
  - Very small memory footprint
  - Get a reasonable solution quickly



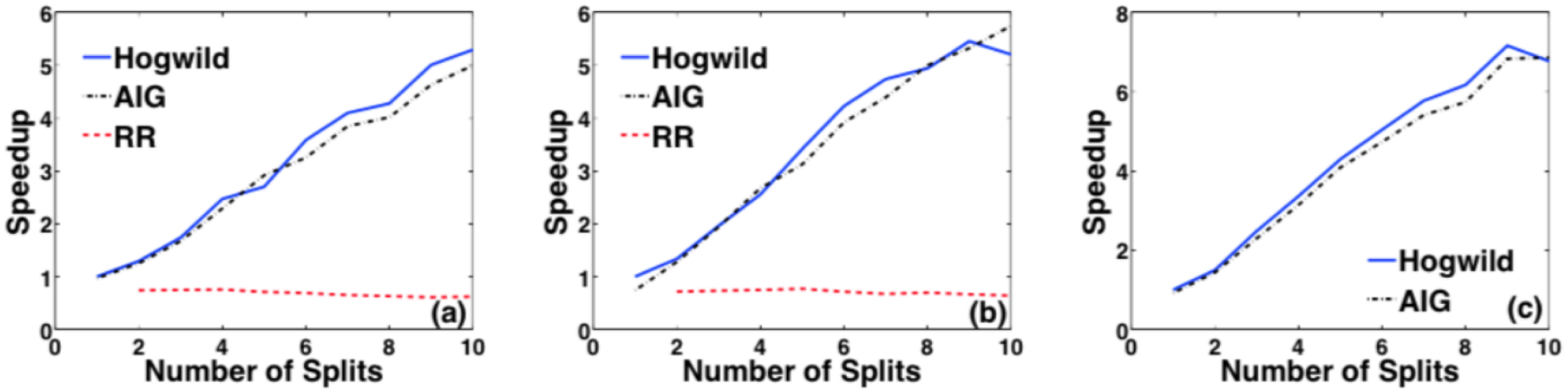
SGD on multiple threads

- Multi CPU with shared memory:
  - Partition the data into multiple sets
  - For each thread:
    - Take a partition
    - Compute the gradient
    - Acquire a lock on current state of θ
    - Read and update θ
    - Release lock
- Bottleneck on time:
  - Update: ~1us
  - Locking acquisition: ~1ms (i.e., 1000x of Update time)

Hogwild!

- The main idea is very simple
  - **Removing lock for all read and update!**
- Too simple to be true?
  - Authors noticed an interesting fact
  - Gradient is usually **sparse**, i.e., lots of 0s in one vector
  - Update them without locking will not easily overlap

  - Performance shown:



Hogwild!

Algorithm 1 HOGWILD! update for individual processors

loop
  Sample e uniformly at random from E
  Read current state w_e and evaluate $G_e(w)$
  for $v \in e$ **do** $w_v = w_v - \gamma b_v^T G_e(w)$
end loop

- Parameters:
  - step size γ
  - Indicator variable b equals to 1 for v-th element and 0 elsewise.
- Only update parameter:
  - With non-zero data element associated

Sparse separable cost function

- Assumption: update vector is sparse
  - i.e., for a training example e
  - Corresponding w_e is very small

$$f(w) = \sum_{e \in E} f_e(w_e)$$

- Sparse Support Vector Machine
  - Data: $E = \{(x_1, y_1), (x_2, y_2), (x_{|E|}, y_{|E|})\}$
  - Loss:

$$\texttt{minimize}_w \sum_{\alpha \in E} \max(1 - y_\alpha w^T x_\alpha, 0) + \lambda ||w||_2^2)$$

  - In final form

$$\texttt{minimize}_w \sum_{\alpha \in E} \max(1 - y_\alpha w^T x_\alpha, 0) + \lambda \sum_{u \in e_\alpha} \frac{w_u^2}{d_u})$$

$$e_\alpha = \texttt{nonzero}(x_\alpha)$$

$$d_u = \#\text{ of training example in } x_\alpha$$

For project:

- Original Hogwild!
  - a single-machine implementation
  - Relies on shared-memory
- The project requires multiple machine
- How will this be different?

GraphLab: A New Framework For Parallel Machine Learning

Yucheng Low
Carnegie Mellon University
ylow@cs.cmu.edu

Joseph Gonzalez
Carnegie Mellon University
jegonzal@cs.cmu.edu

Aapo Kyrola
Carnegie Mellon University
akyrola@cs.cmu.edu

Danny Bickson
Carnegie Mellon University
bickson@cs.cmu.edu

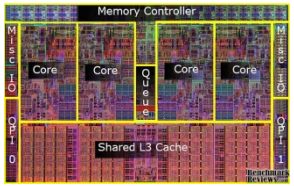
Carlos Guestrin
Carnegie Mellon University
guestrin@cs.cmu.edu

Joseph Hellerstein
UC Berkeley
hellerstein@cs.berkeley.edu

GraphLab

- A new parallel framework for machine learning
- But what's the motivation behind?
- Wide array of different parallel architectures:

GPUs



Multicore

Clusters

Mini Clouds

Clouds

Reference: Low et al., GraphLab: A new framework for parallel machine learning, arxiv.org
https://arxiv.org/abs/1408.2041
http://cloud.berkeley.edu/data/graphlab.pptx

A typical approach:

- 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

Map-reduce / Hadoop, solution?

- Hadoop and Map-reduce is designed for
  - Parallel programming
  - Real world solution for
    - Search engine, e.g. Google
    - Recognition system
- Can this generalize to machine learning problem?
  - (Answer is no, otherwise why bother to propose GraphLab …)
  - but we will see why :)

MapReduce – Map Phase

CPU 1 — 12.9

CPU 2 — 42.3

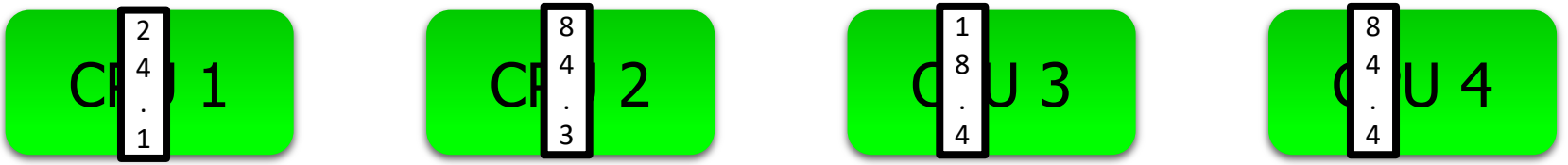
CPU 3 — 21.3

CPU 4 — 25.8

Embarrassingly Parallel independent computation

No Communication needed

MapReduce – Map Phase



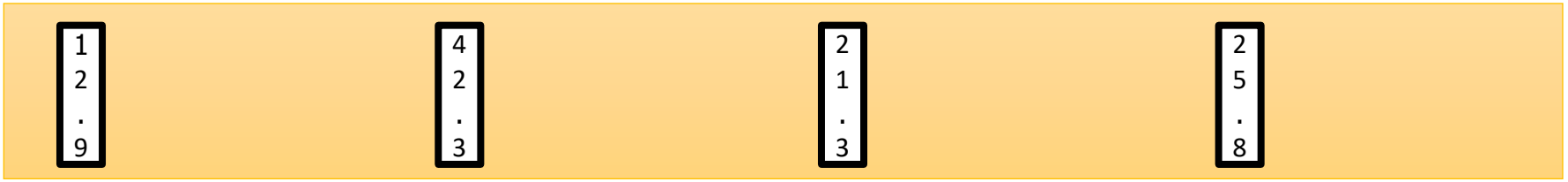
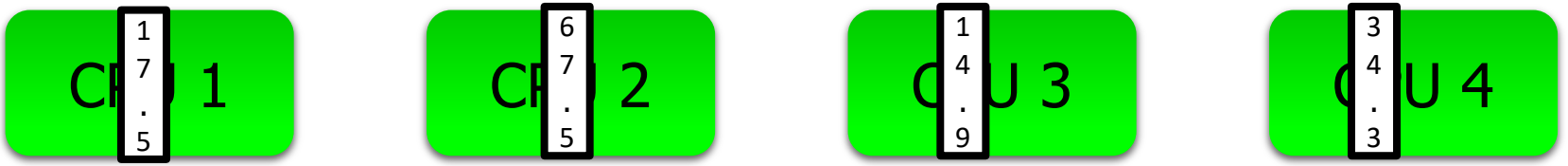
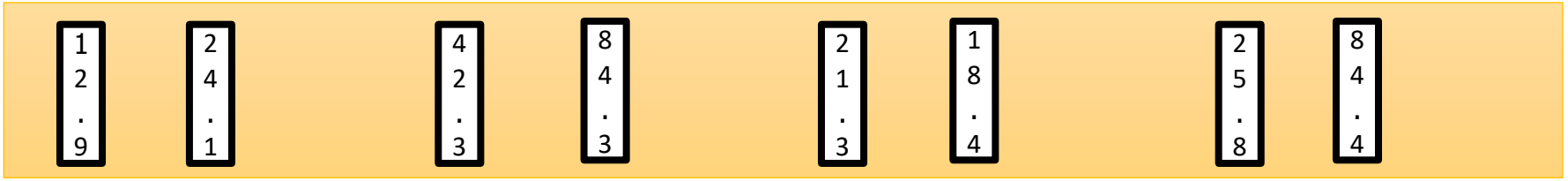


Image Features

MapReduce – Map Phase





Embarrassingly Parallel independent computation

No Communication needed

MapReduce – Reduce Phase

Attractive Face Statistics

Ugly Face Statistics

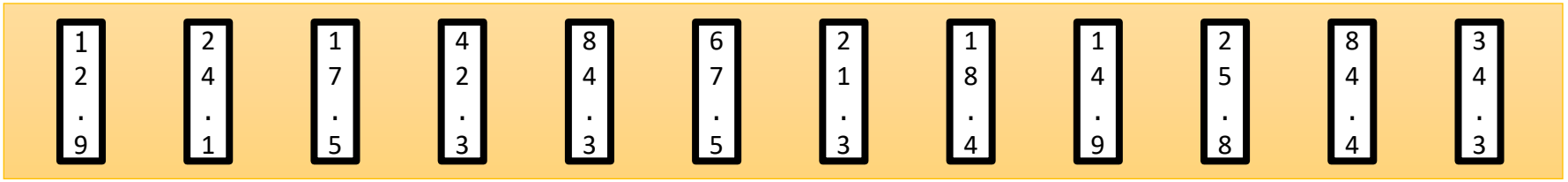


Image Features

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:

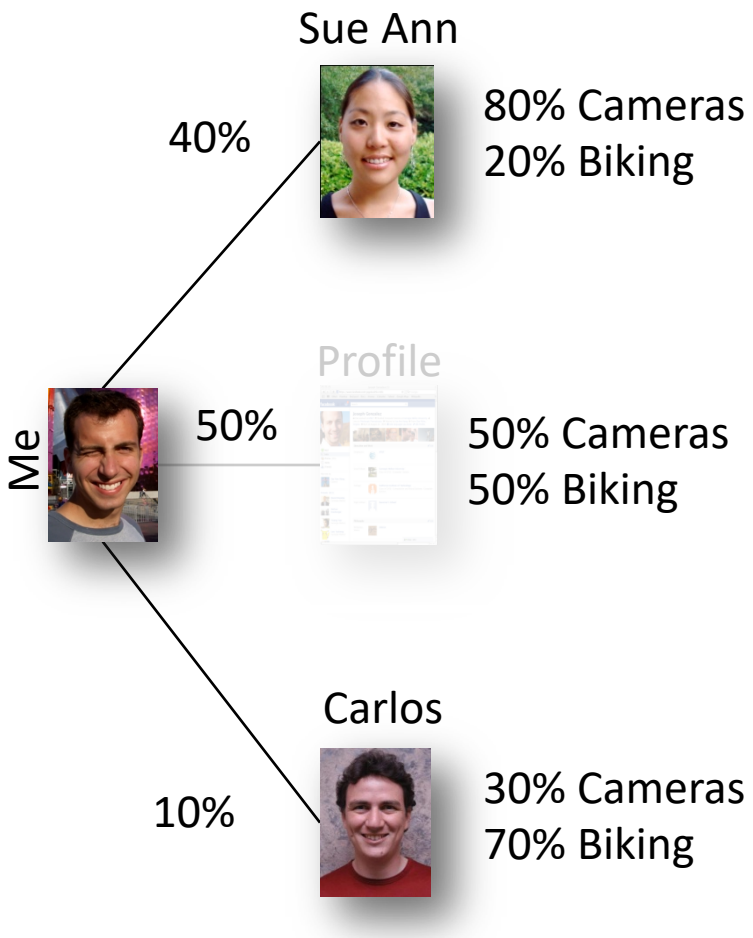
50% What I list on my profile
40% Sue Ann Likes
\+ 10% Carlos Like

I Like: 60% Cameras, 40% Biking

- Recurrence Algorithm:

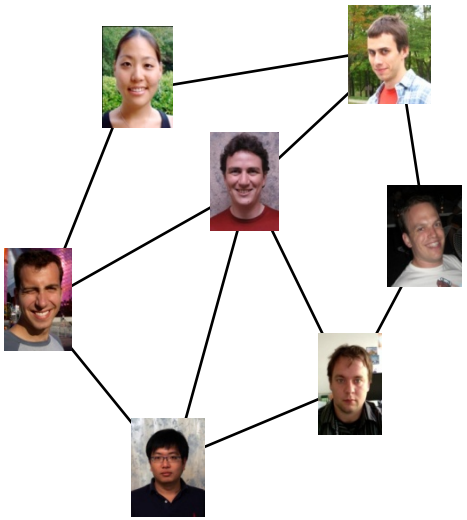
$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

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

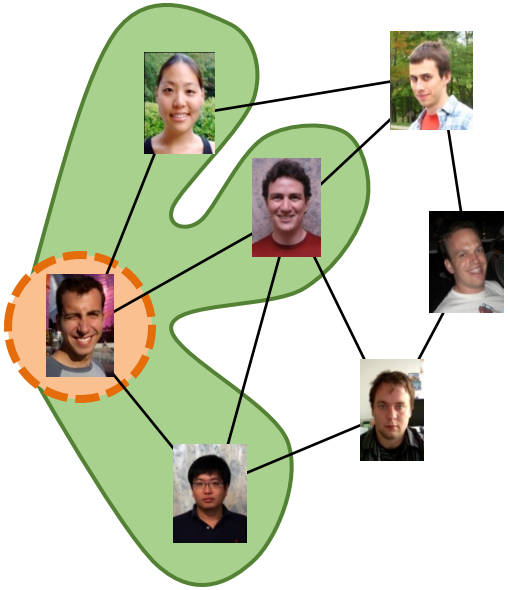


Properties of Graph Parallel Algorithms

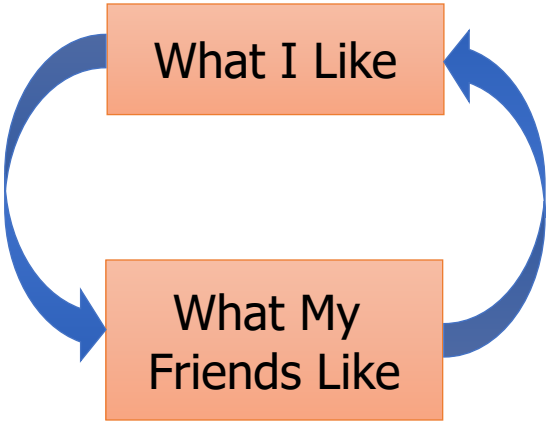
Dependency Graph



Factored Computation

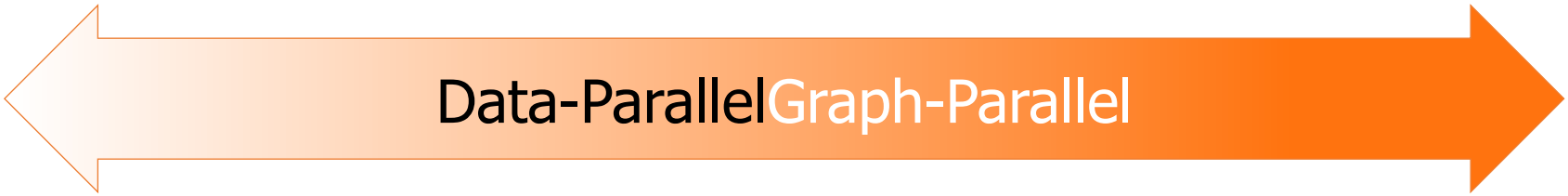


Iterative Computation



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

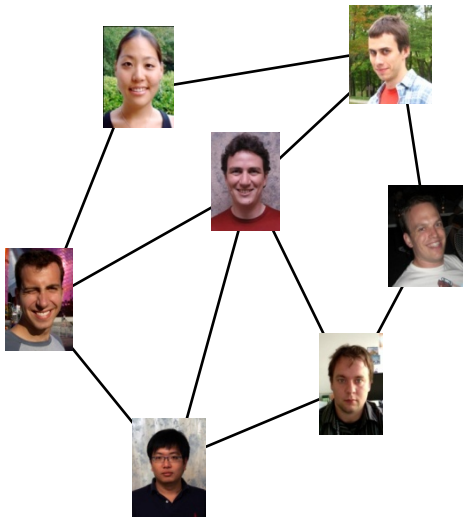
Deep Belief Networks

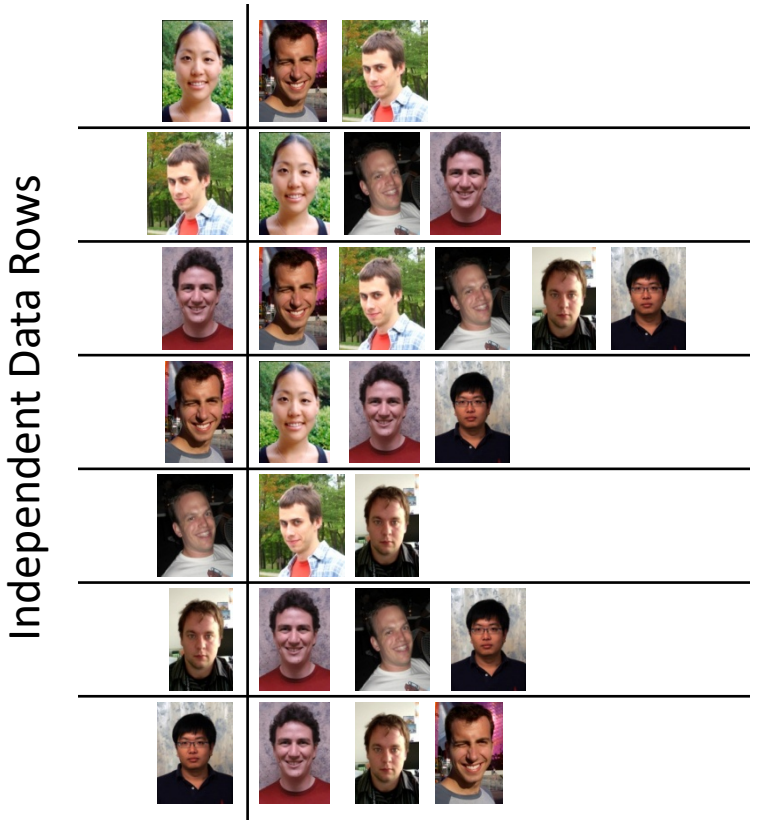
Neural Networks

Why not use Map-Reduce for Graph Parallel Algorithms?

Data Dependencies

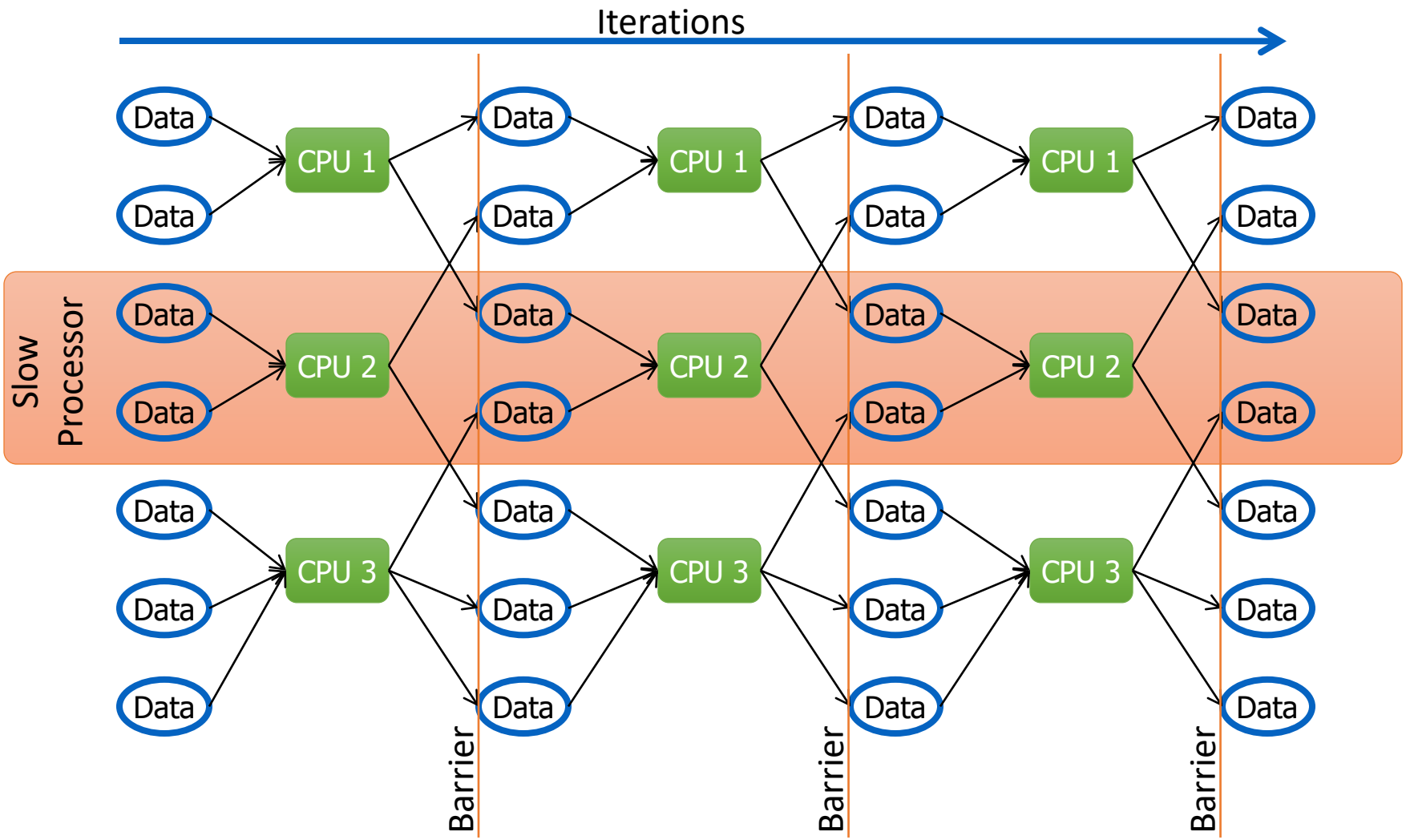
- Map-Reduce does not efficiently express dependent data
  - User must code substantial data transformations
  - Costly data replication





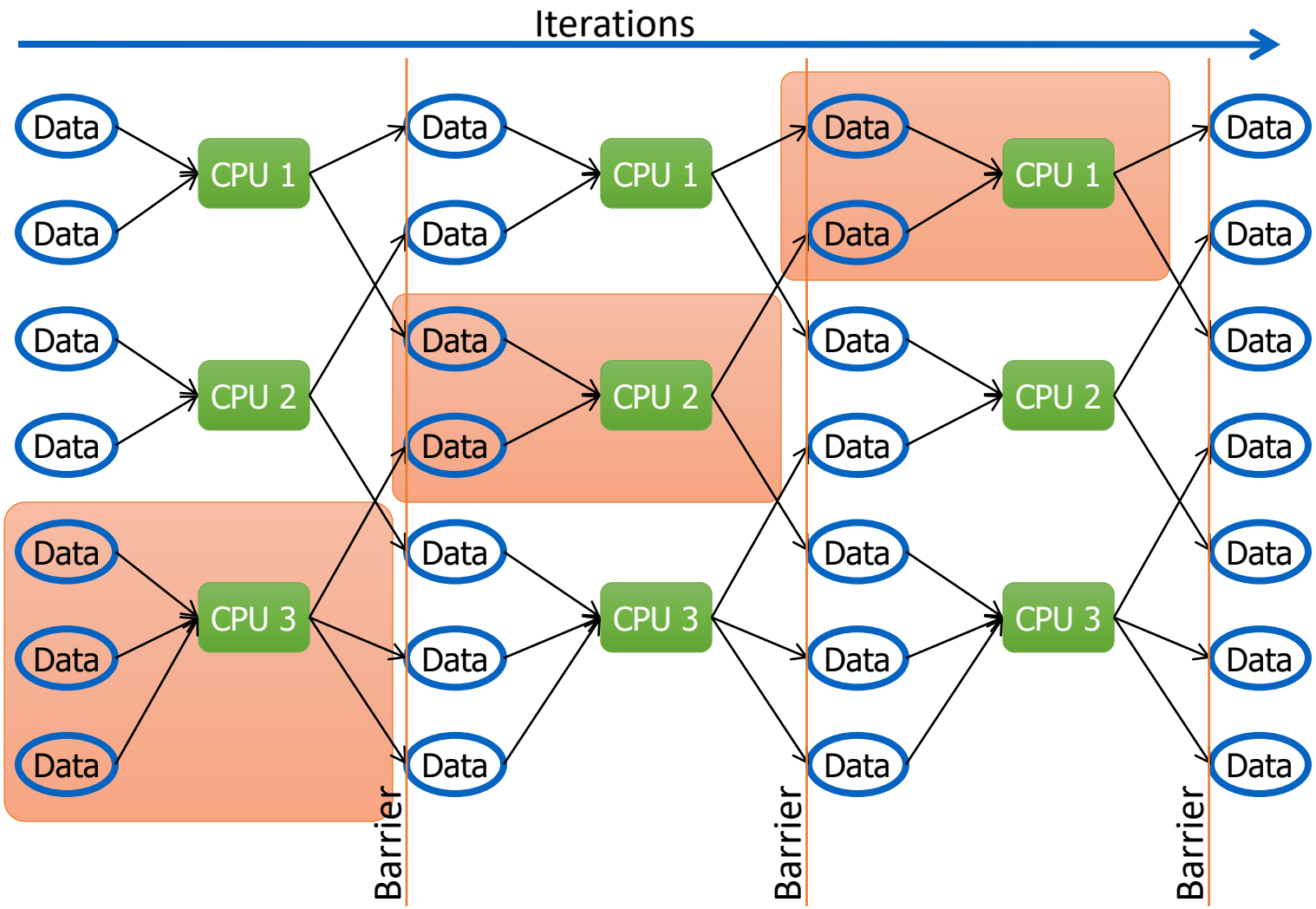
Iterative Algorithms

- Map-Reduce not efficiently express iterative algorithms:



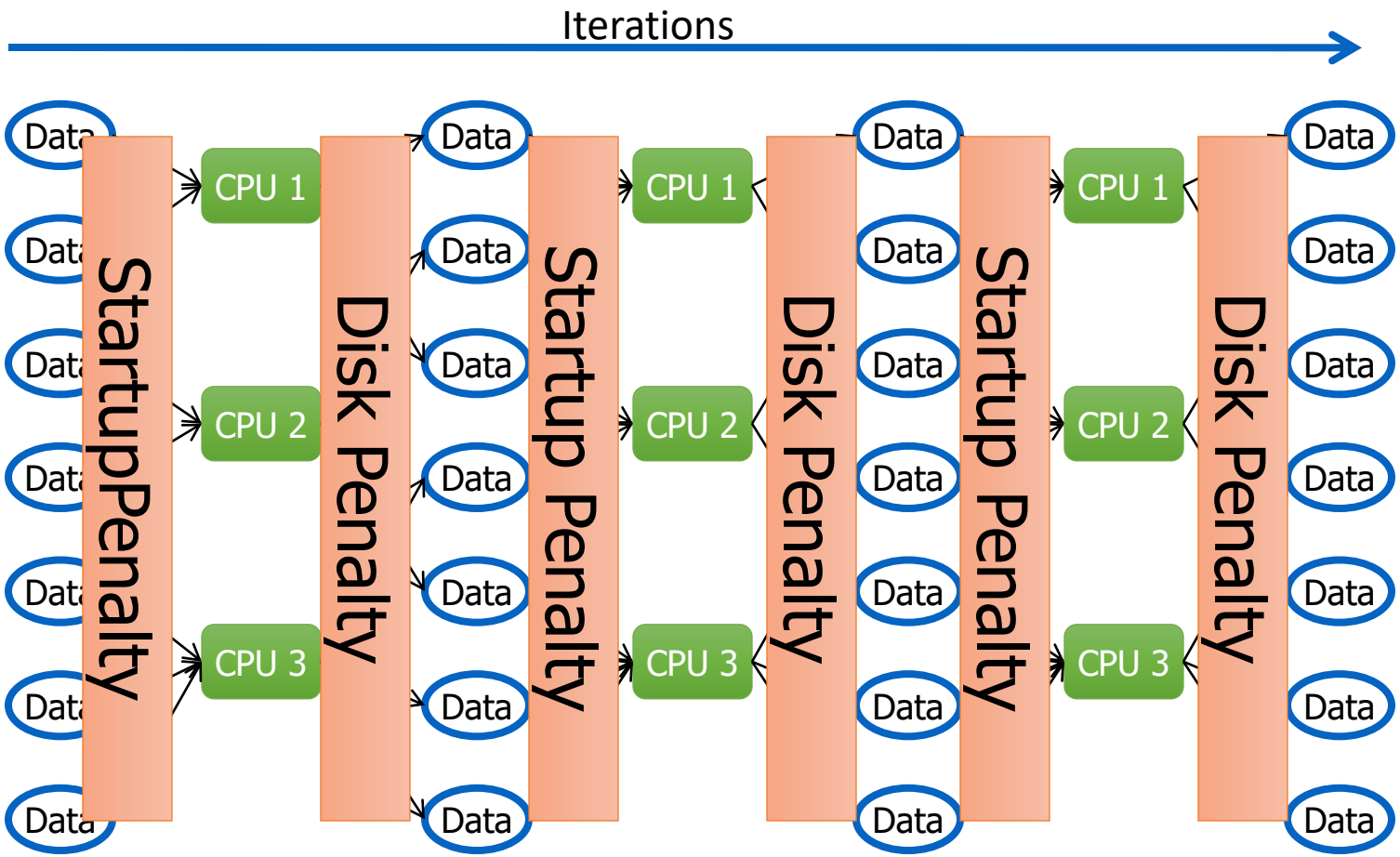
MapAbuse: Iterative MapReduce

- Only a subset of data needs computation:



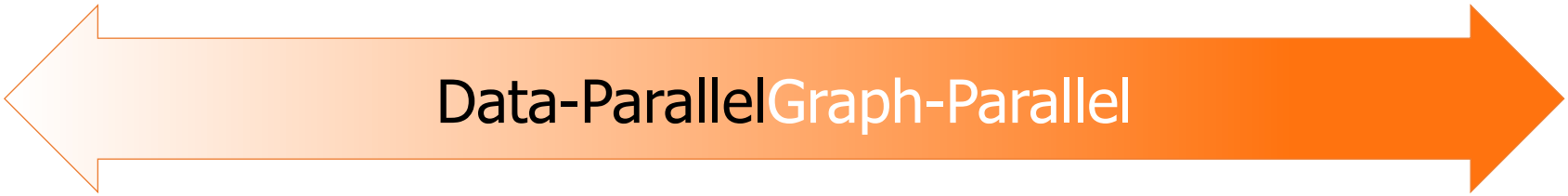
MapAbuse: Iterative MapReduce

- 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

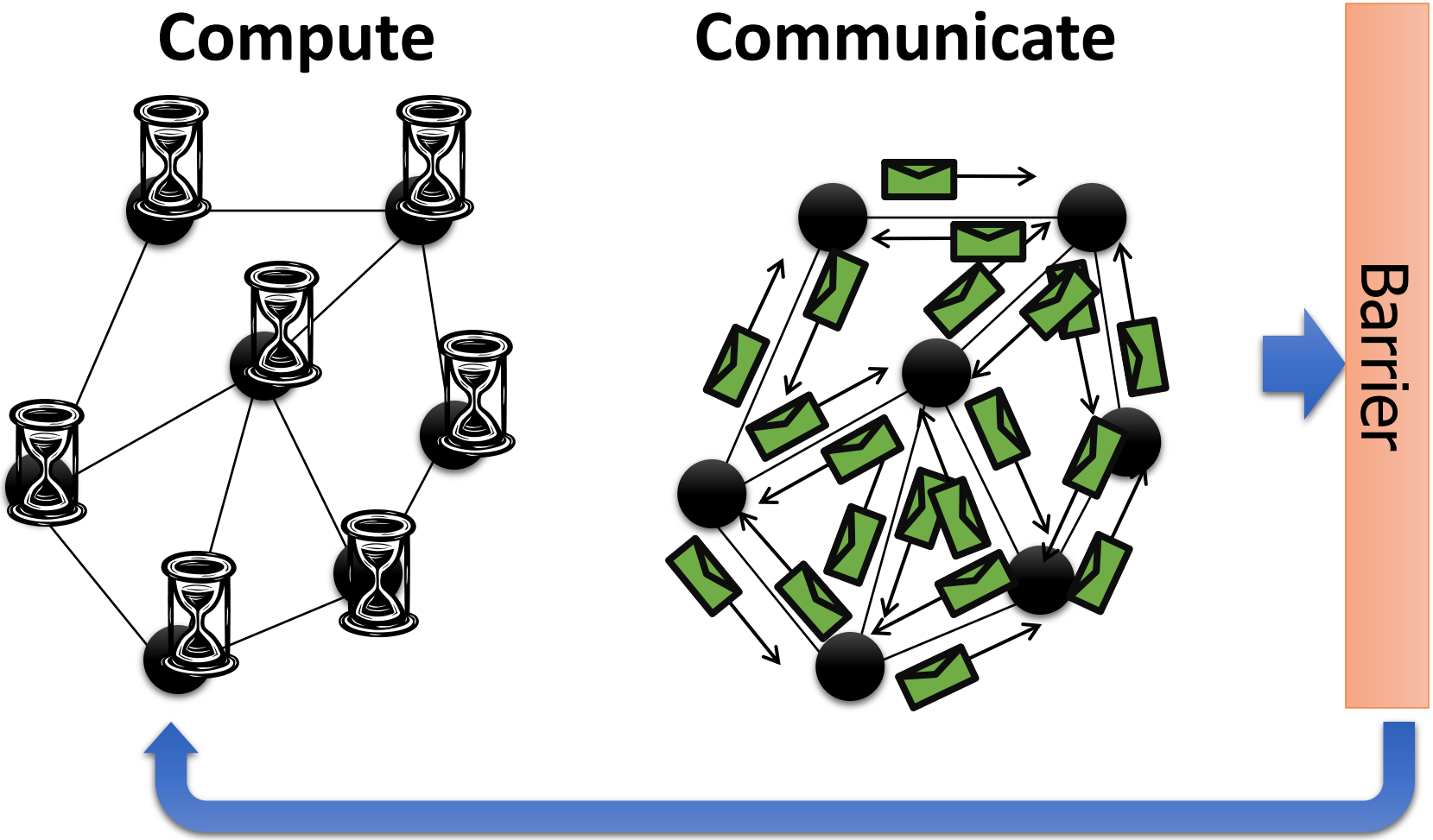
PageRank

Deep Belief Networks

Neural Networks

Pregel (Giraph)

- Bulk Synchronous Parallel Model:



Problem

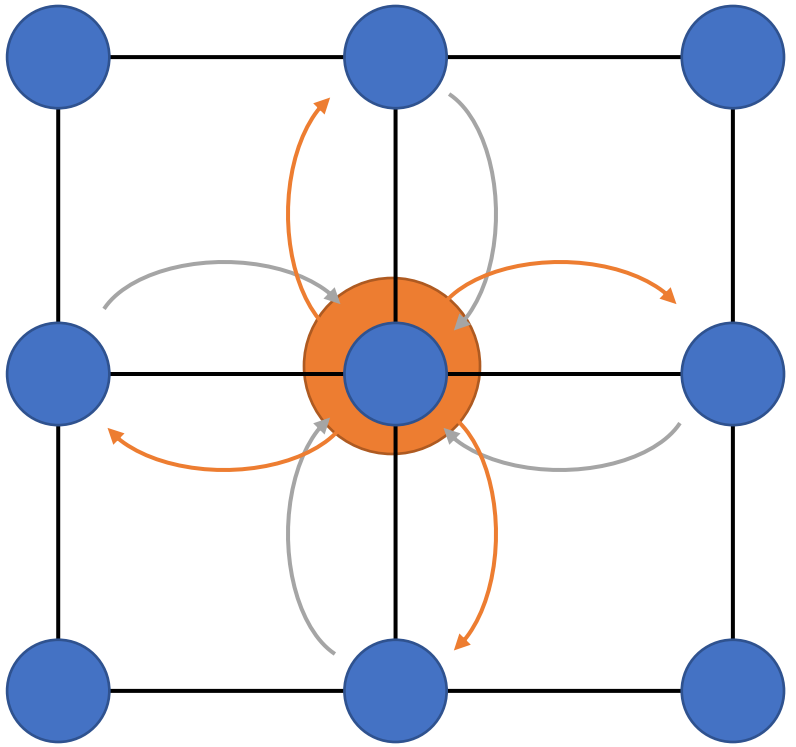
Bulk synchronous computation can be highly inefficient.

Example:

Loopy Belief Propagation

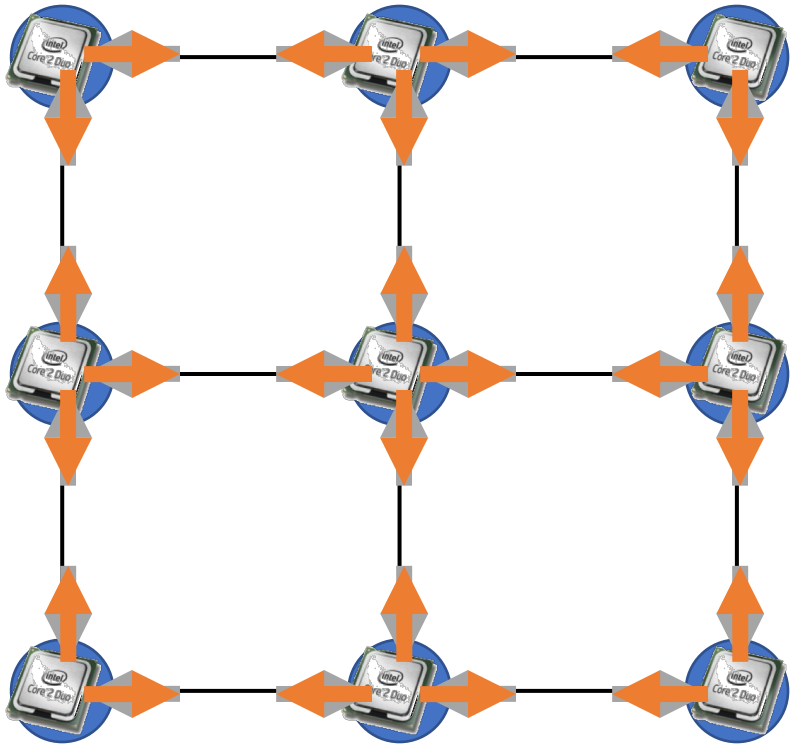
Loopy Belief Propagation (Loopy BP)

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

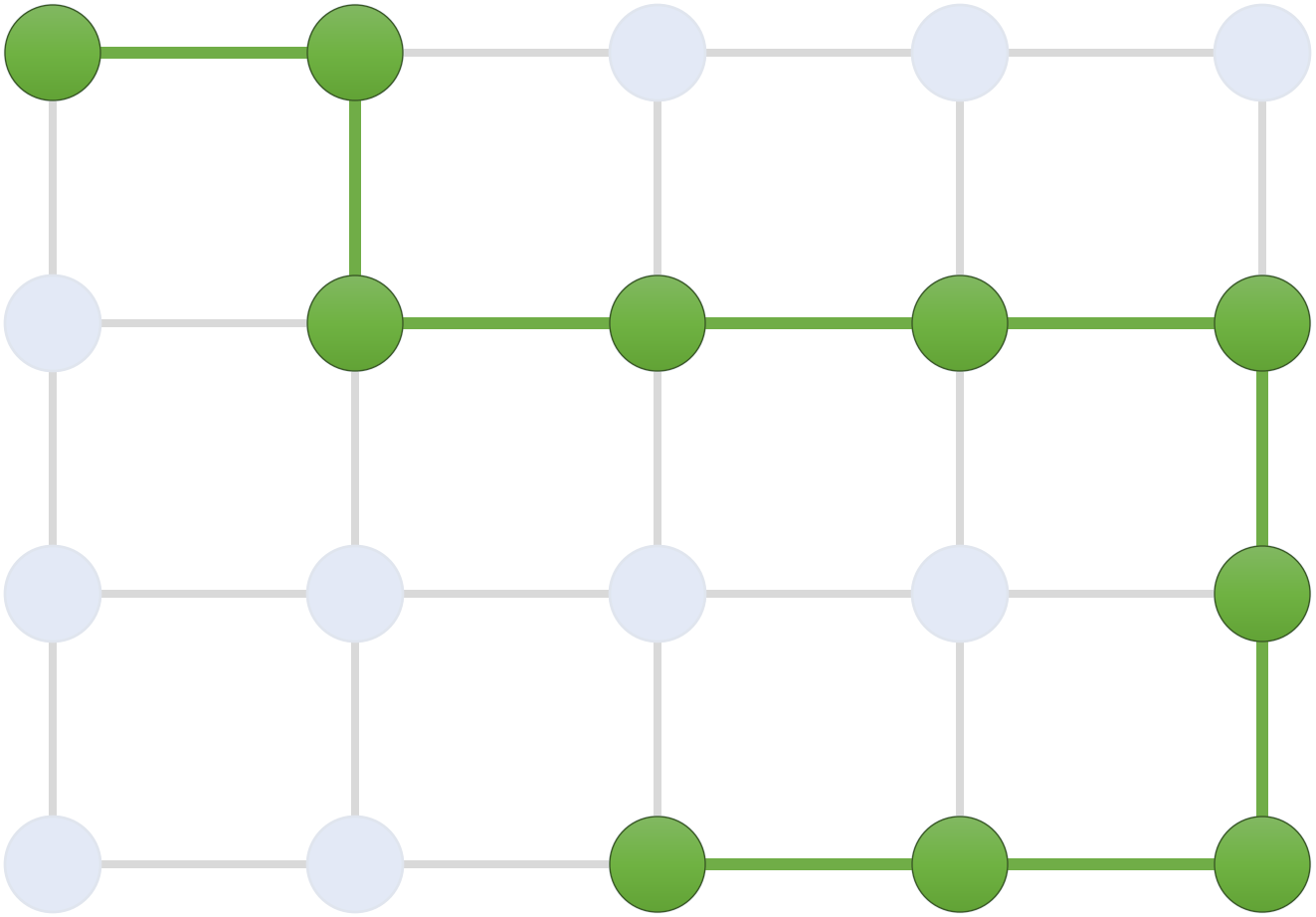


Bulk Synchronous Loopy BP

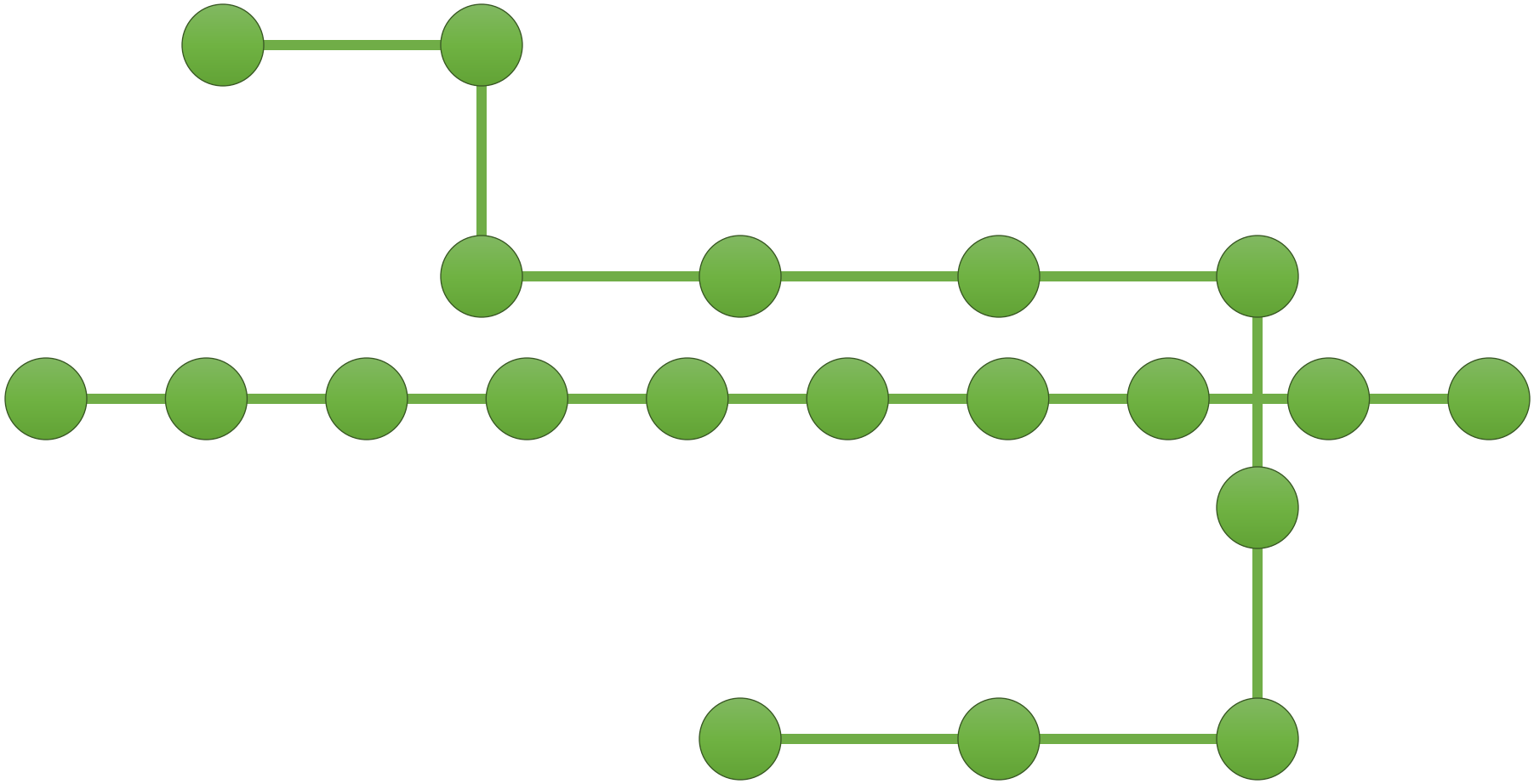
- Often considered embarrassingly parallel
  - Associate processor with each vertex
  - Receive all messages
  - Update all beliefs
  - Send all messages
- Proposed by:
  - Brunton et al. CRV'06
  - Mendiburu et al. GECC'07
  - Kang,et al. LDMTA'10
  - …



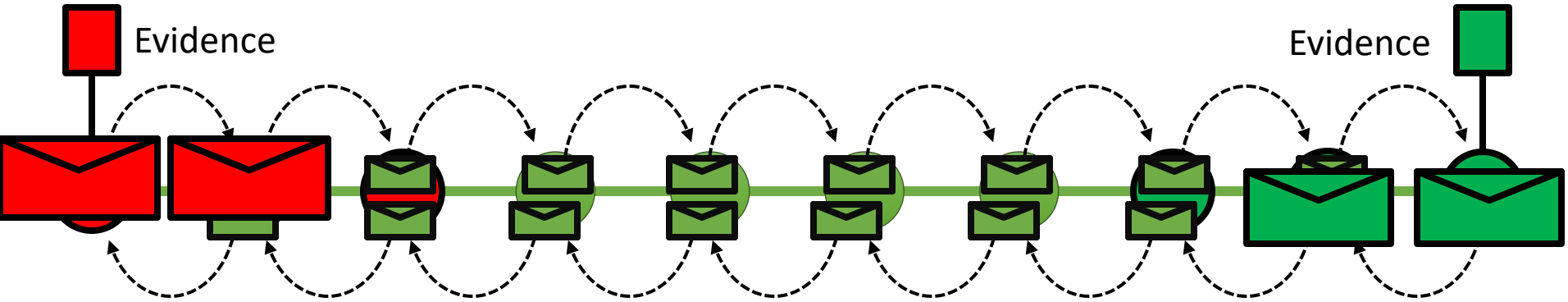
Sequential Computational Structure



Hidden Sequential Structure



Hidden **Sequential** Structure



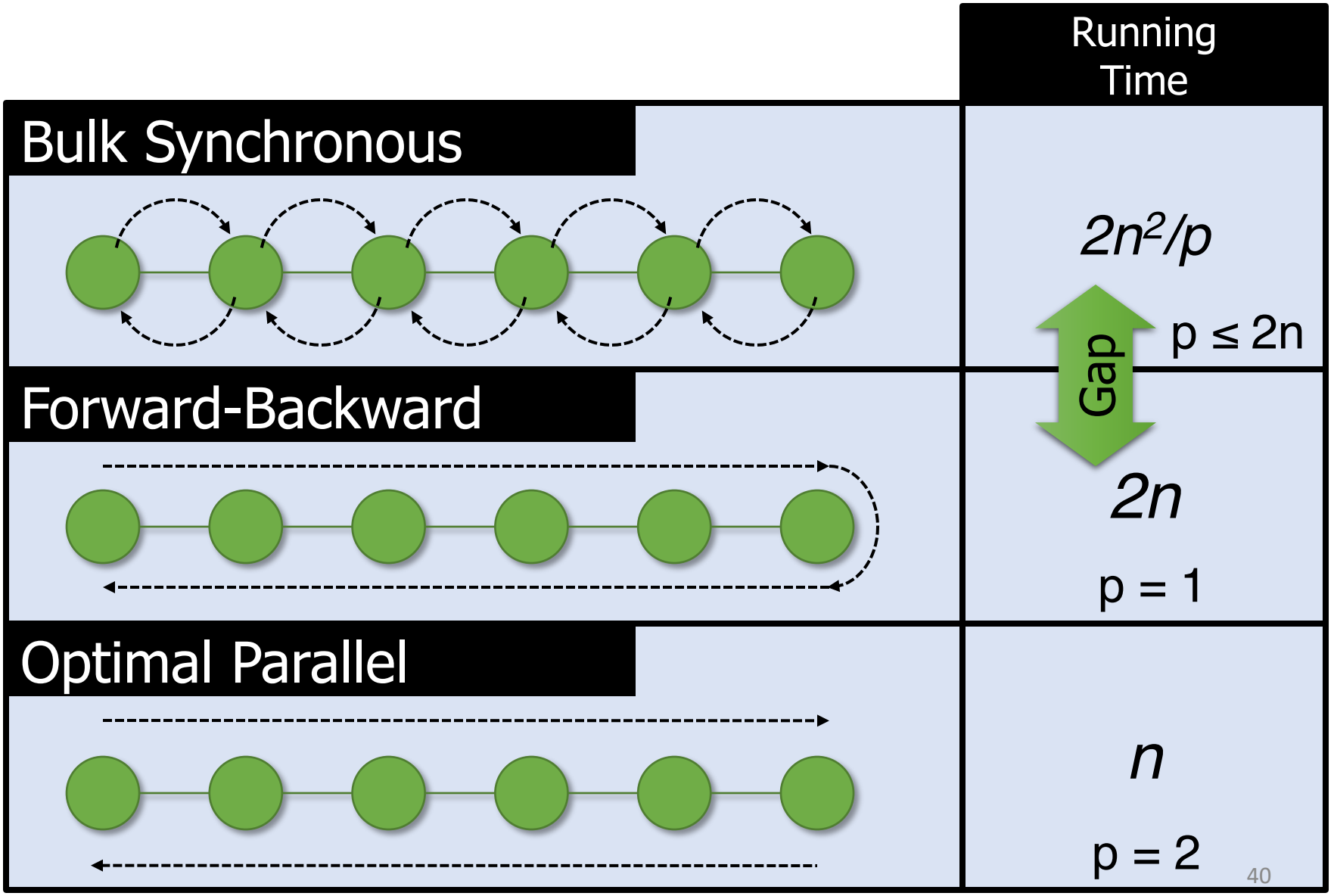
- Running Time:

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

Time for a single parallel iteration

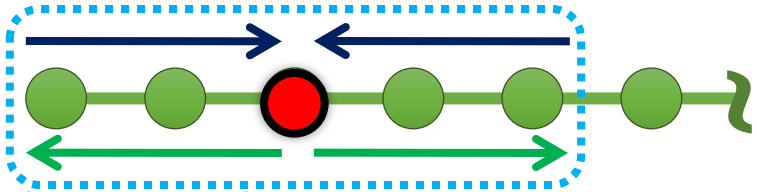
Number of Iterations

Optimal Sequential Algorithm



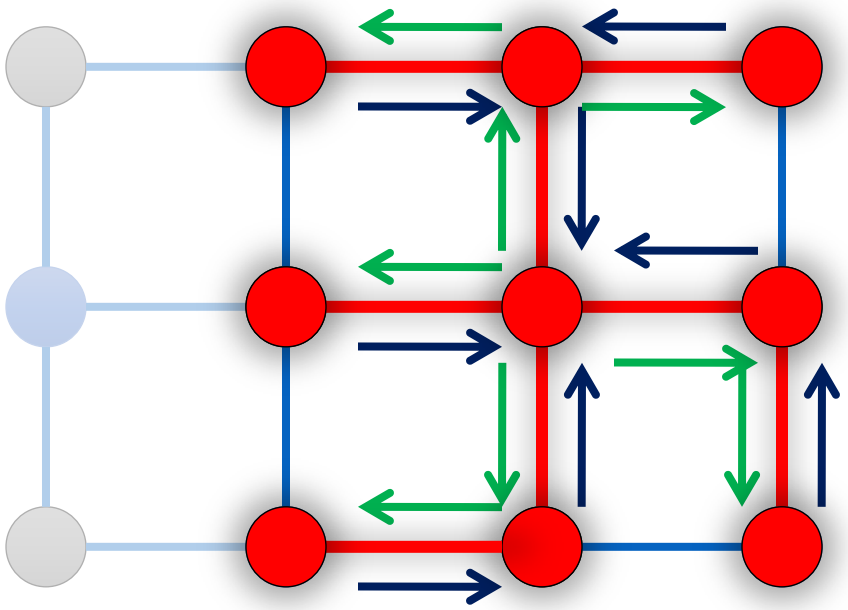
The Splash Operation

- 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



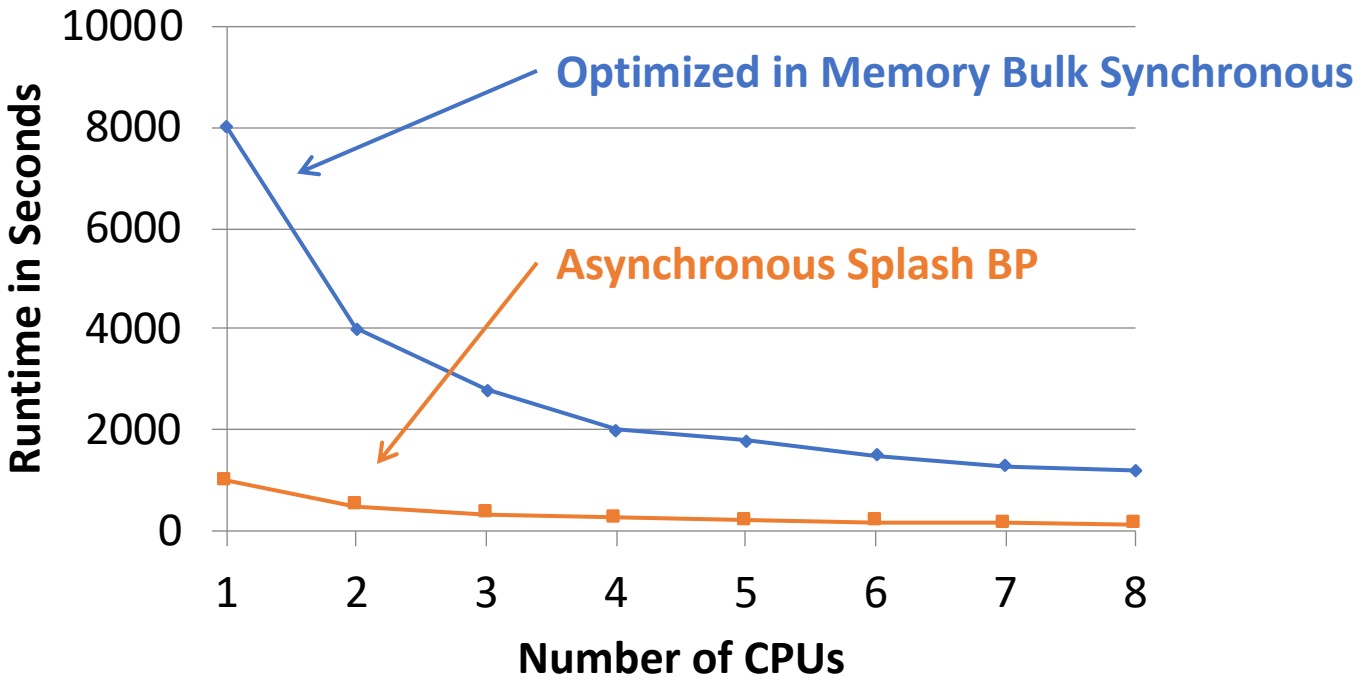
Data-Parallel Algorithms can be **Inefficient**

Residual Splash for Optimally Parallelizing Belief Propagation

Joseph E. Gonzalez
Carnegie Mellon University

Yucheng Low
Carnegie Mellon University

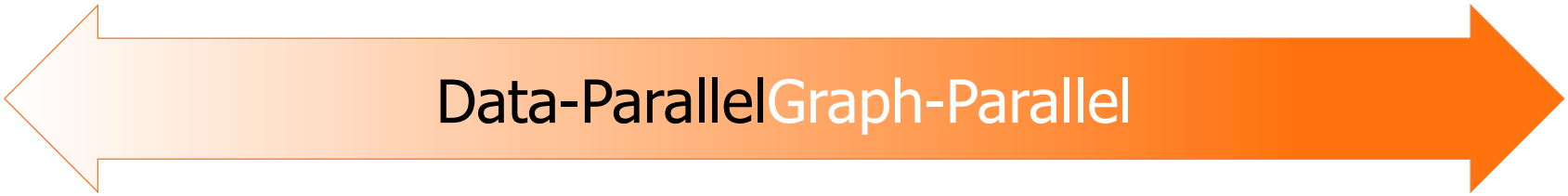
Carlos Guestrin
Carnegie Mellon University



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

The Need for a New Abstraction

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



Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (Giraph)

Carnegie Mellon

SVM

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

Deep Belief Networks

Neural Networks

Lasso

The GraphLab Framework

Graph Based

Data Representation

Update Functions

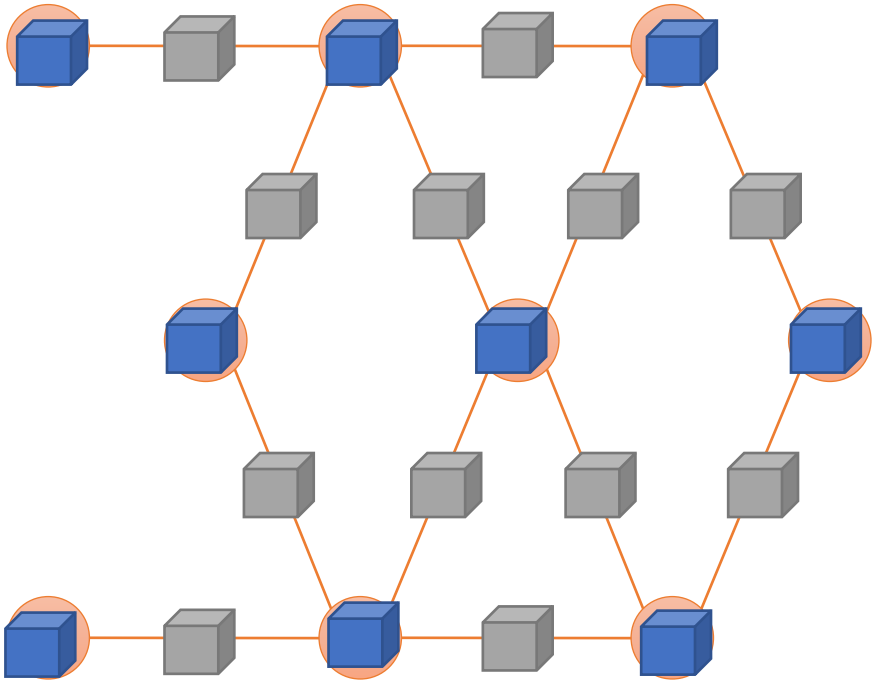
User Computation

Scheduler

Consistency Model

Data Graph

A **graph** with arbitrary data associated with each vertex and edge.



Graph:

• Social Network

Vertex Data:

•User profile text
• Current interests estimates

Edge Data:

• Similarity weights

The GraphLab Framework

Graph Based

Data Representation

Update Functions

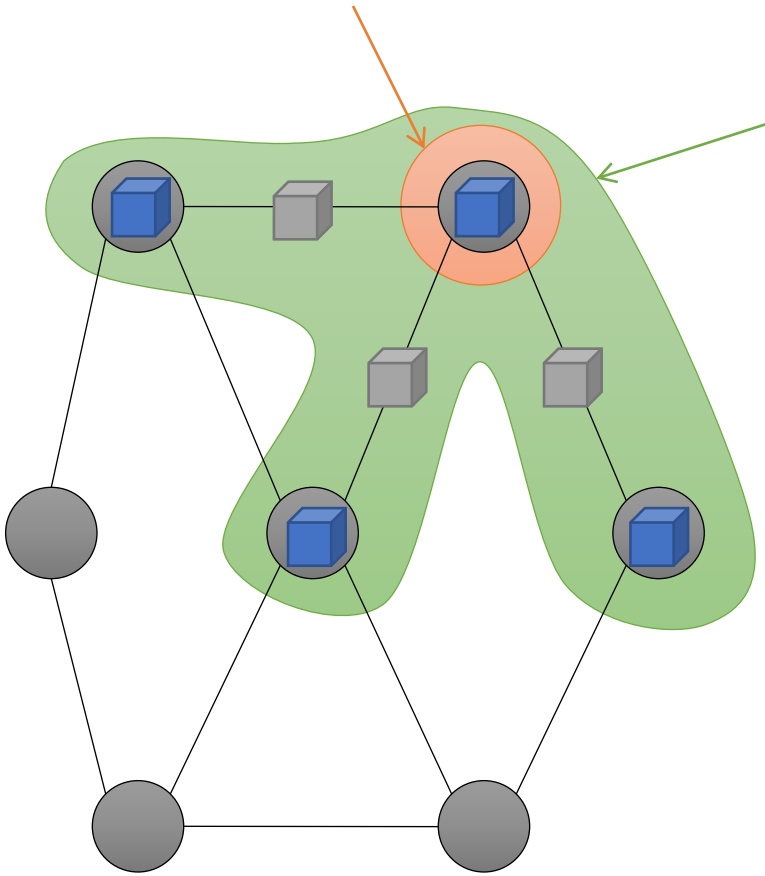
User Computation

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] ← Σ_{j∈Friends[i]} W_ij × Likes[j];

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

The GraphLab Framework

Graph Based

Data Representation

Update Functions

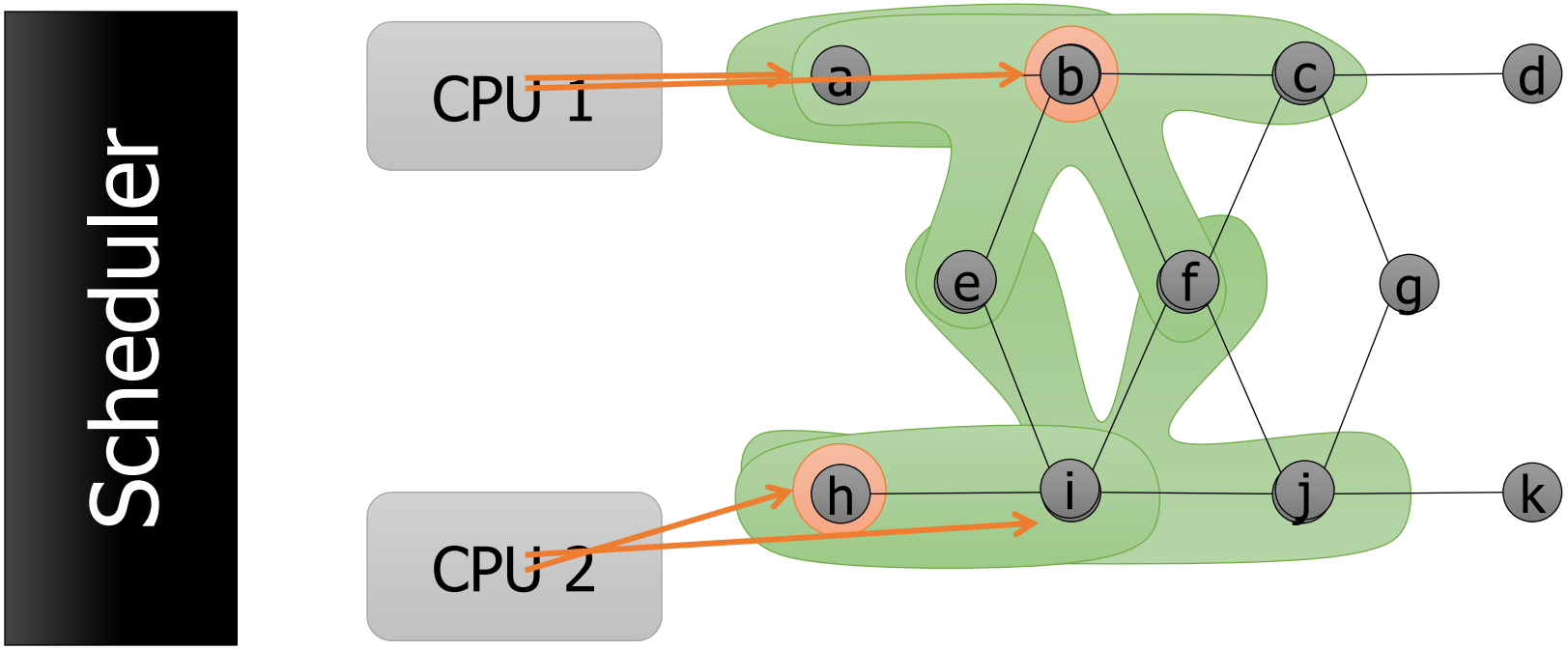
User Computation

Scheduler

Consistency Model

The Scheduler

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

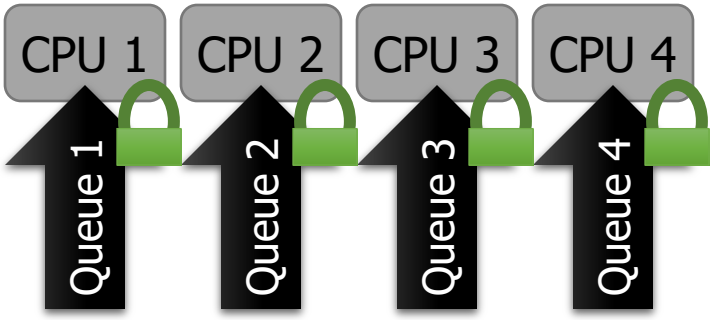


The process repeats until the scheduler is empty.

Implementing the Schedulers

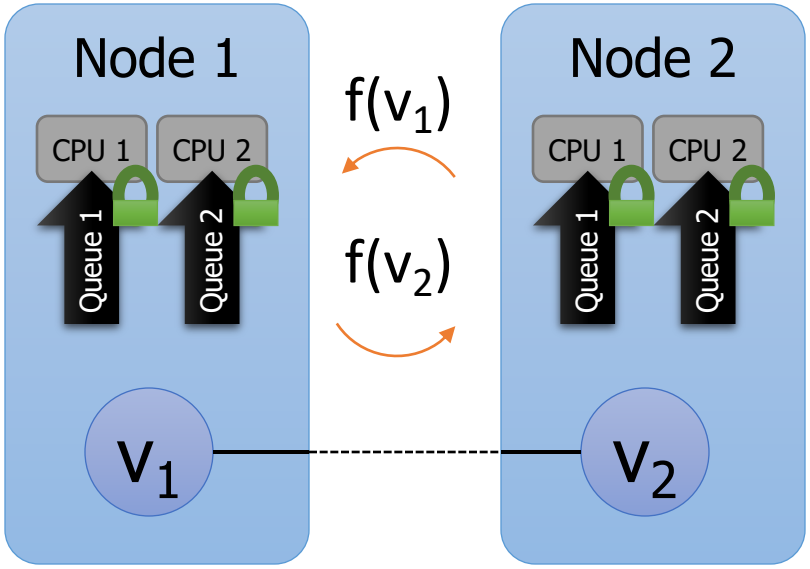
Multicore Setting

- 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

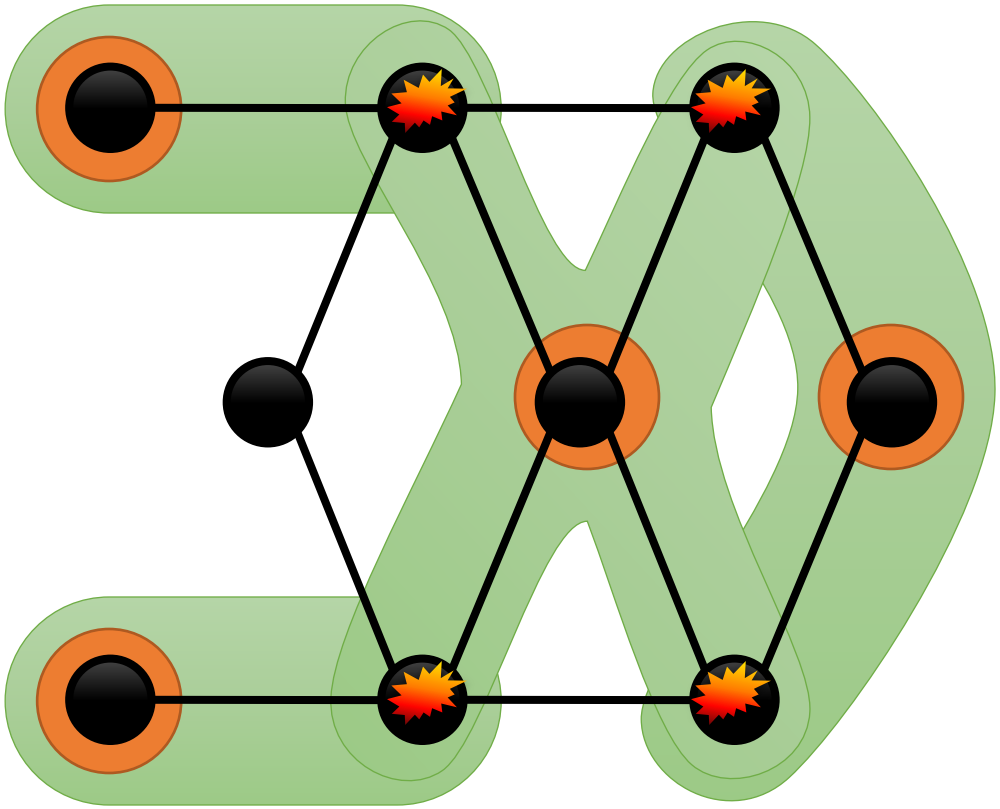
User Computation

Scheduler

Consistency Model

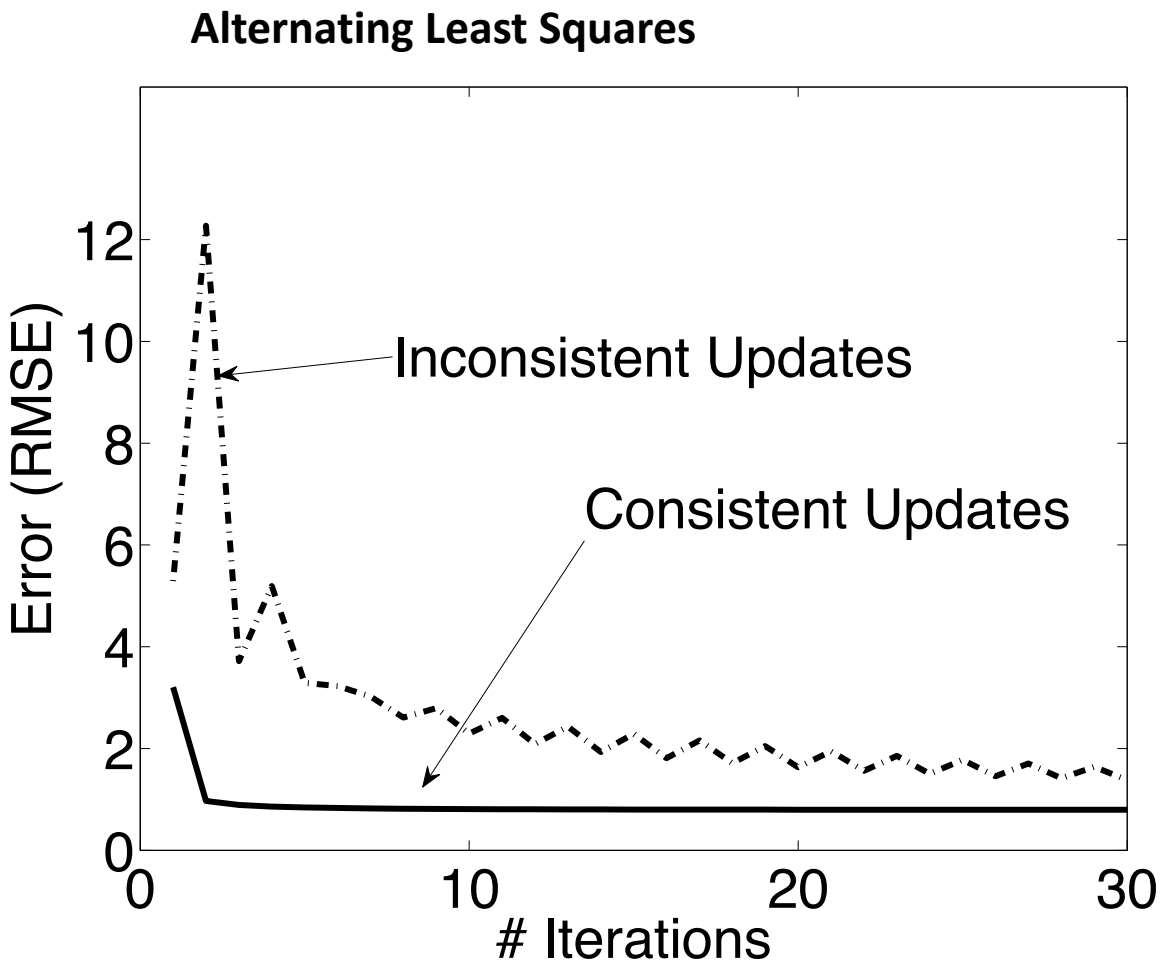
Ensuring Race-Free Code

- How much can computation **overlap**?



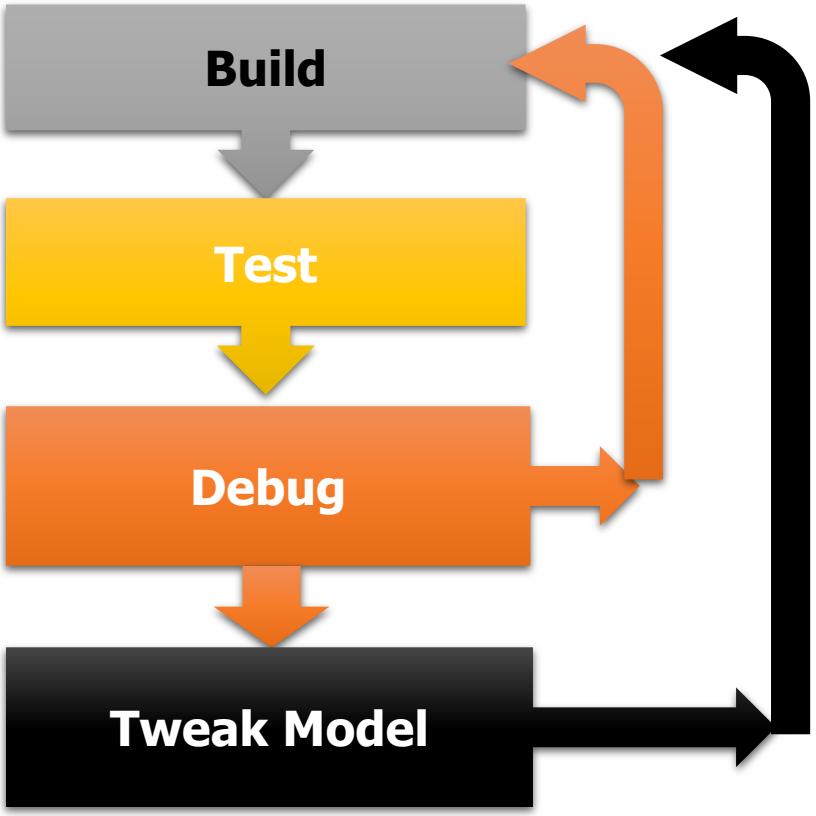
Importance of Consistency

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



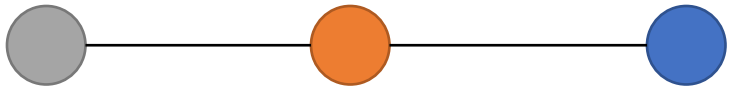
Importance of Consistency

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

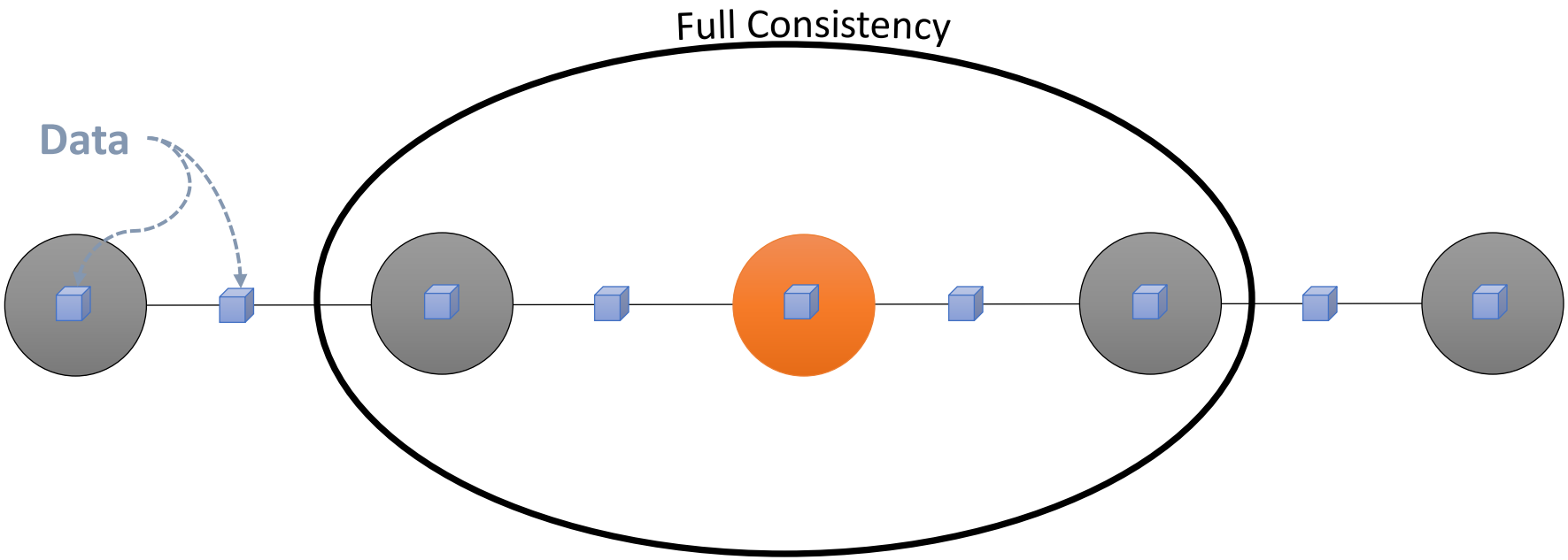
CPU 1

CPU 2

Sequential

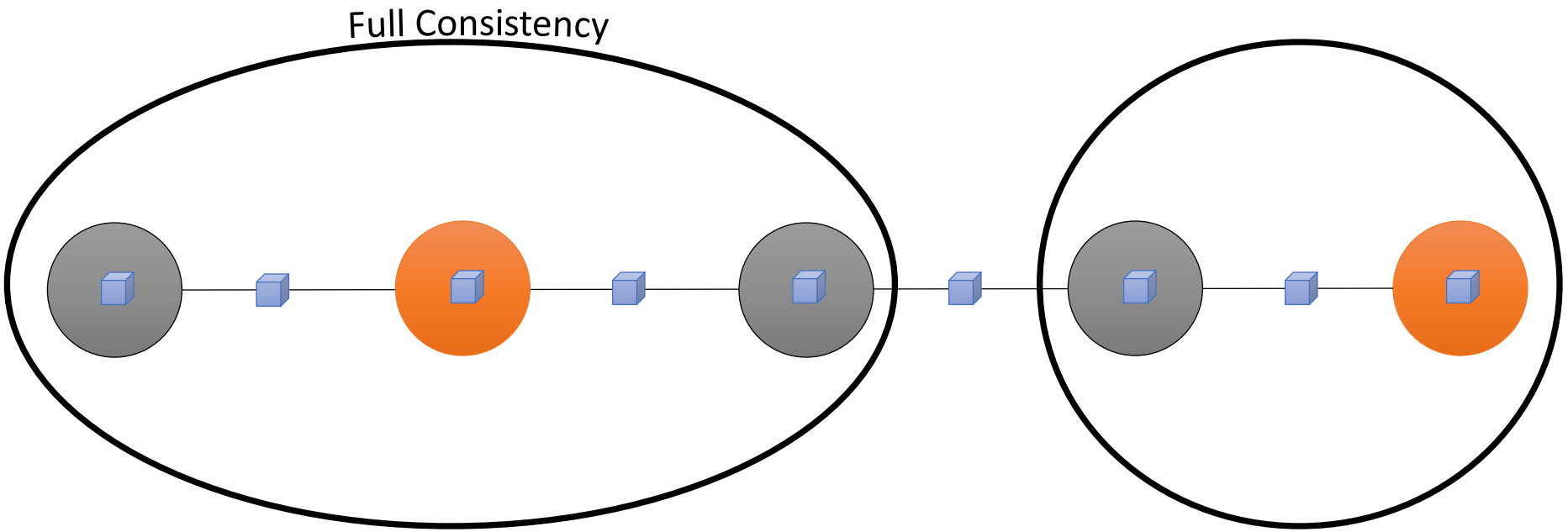
Single CPU

Consistency Rules

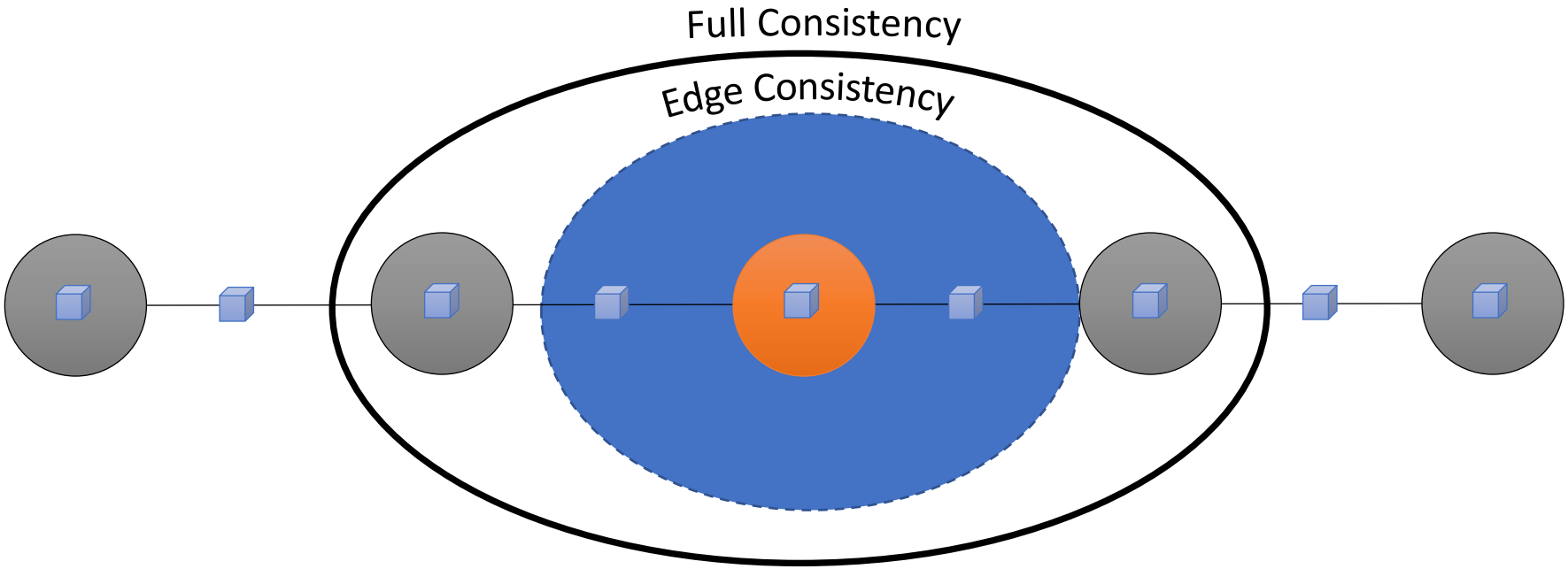


Guaranteed sequential consistency for all update functions

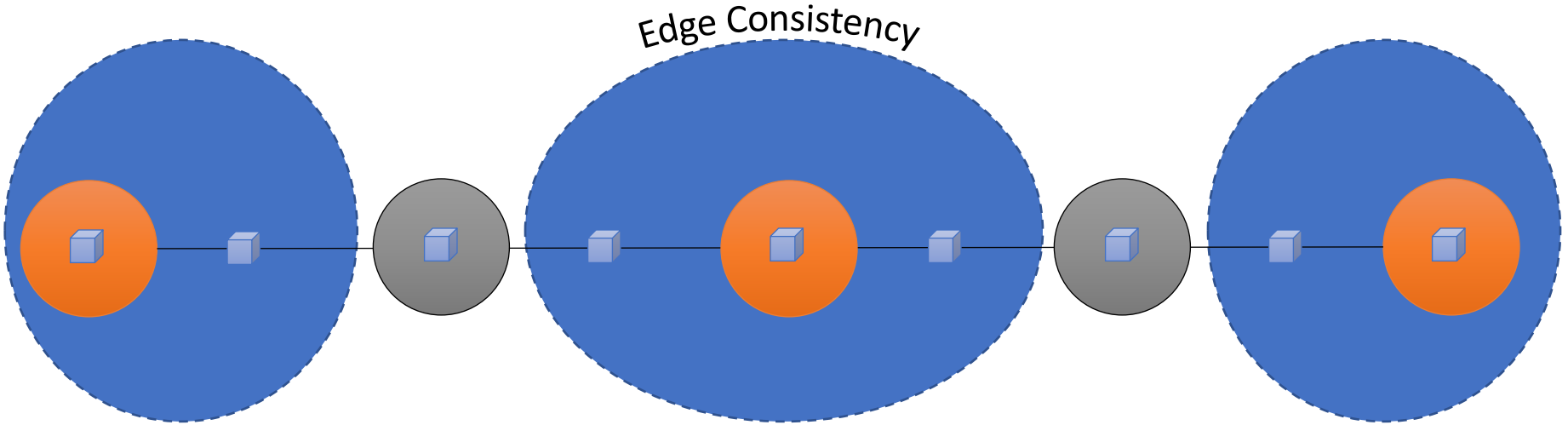
Full Consistency



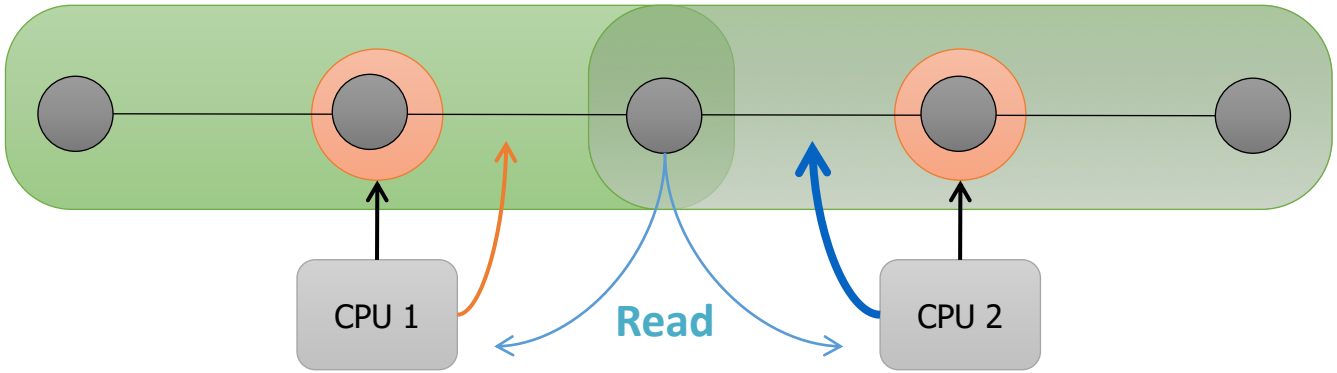
Obtaining More Parallelism



Edge Consistency

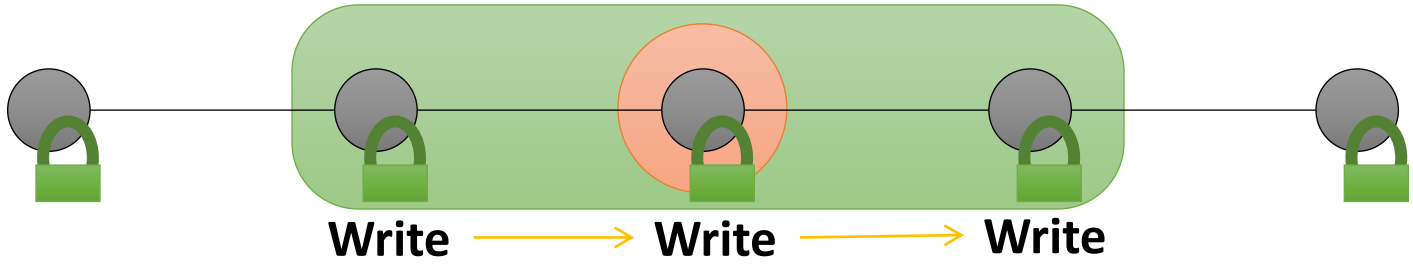


Safe

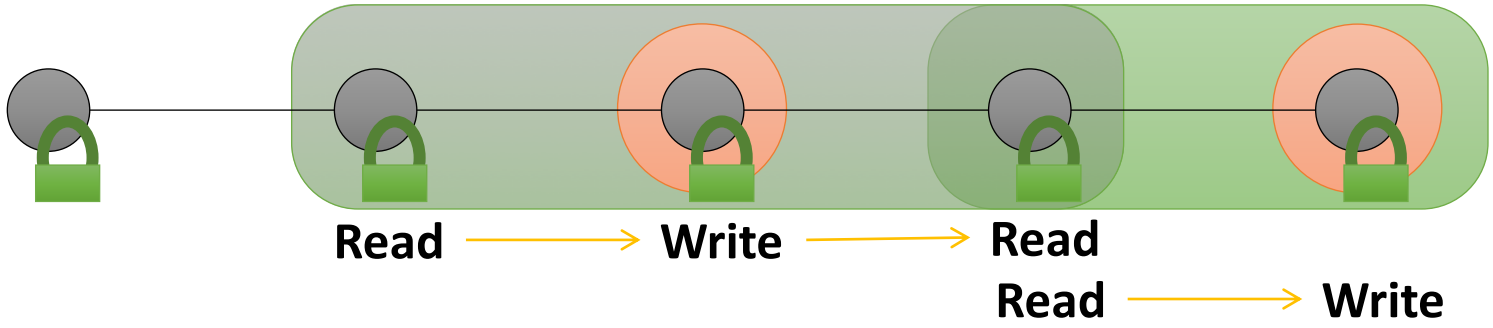


Consistency Through R/W Locks

- Read/Write locks:
  - Full Consistency



  - Edge Consistency Canonical Lock Ordering



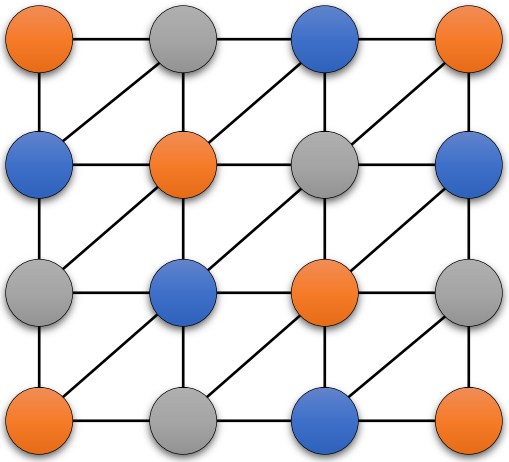
Consistency Through R/W Locks

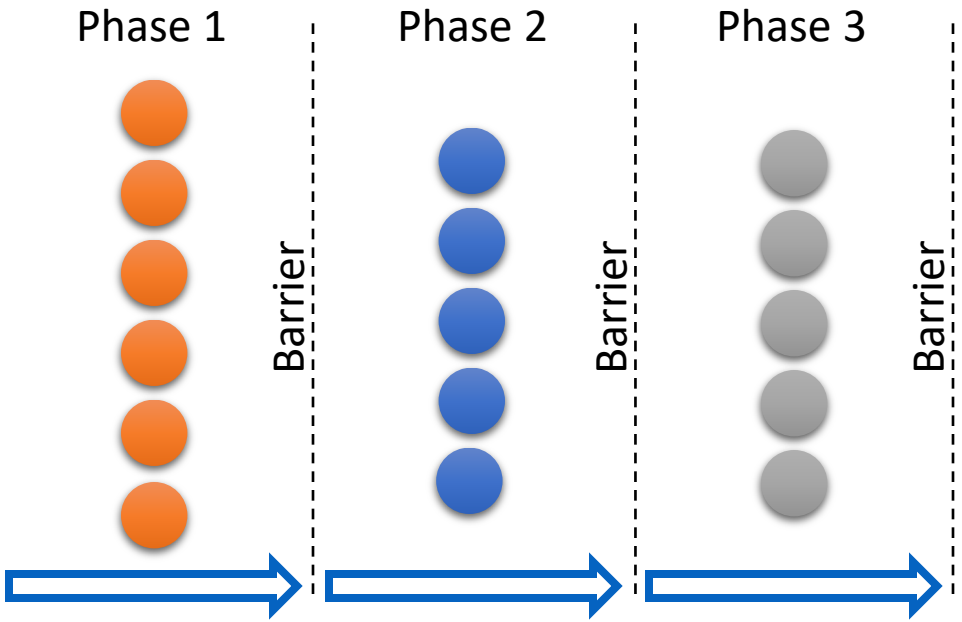
- Multicore Setting: Pthread R/W Locks
- Distributed Setting: *Distributed Locking*
  - Prefetch Locks and Data

  - 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.

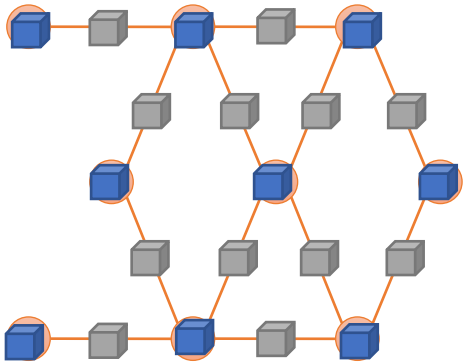




The GraphLab Framework

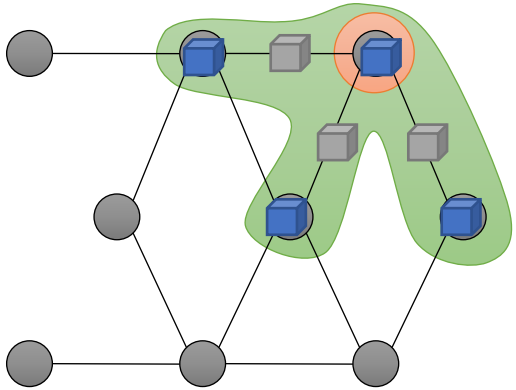
Graph Based

Data Representation

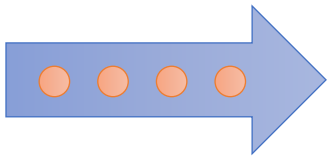


Update Functions

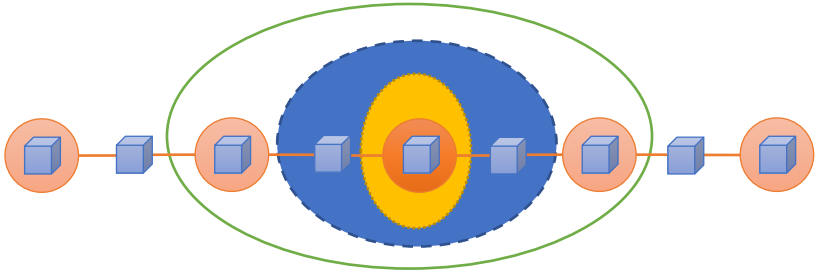
User Computation



Scheduler



Consistency Model



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
- …

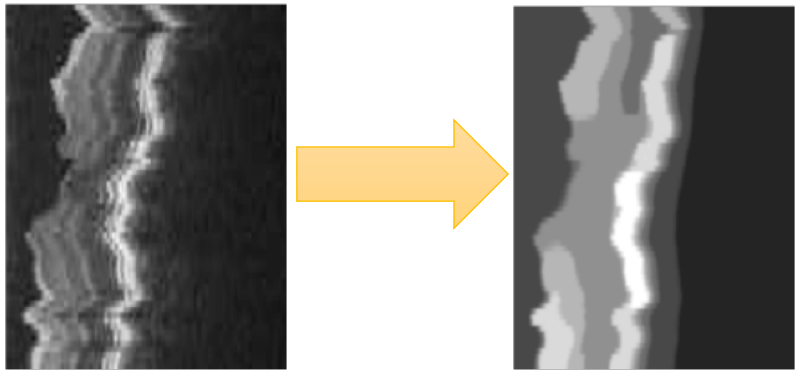
Shared Memory Experiments

Shared Memory Setting

16 Core Workstation

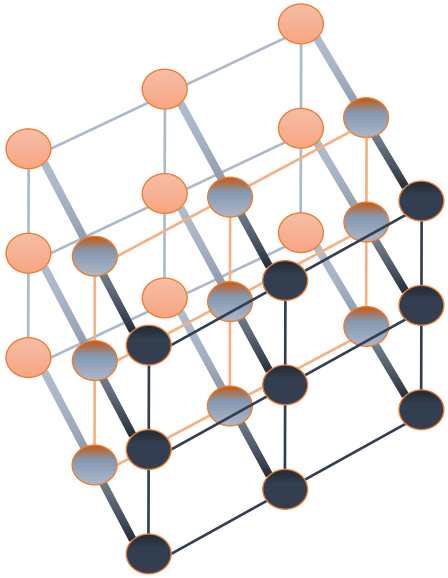
Loopy Belief Propagation

3D retinal image denoising



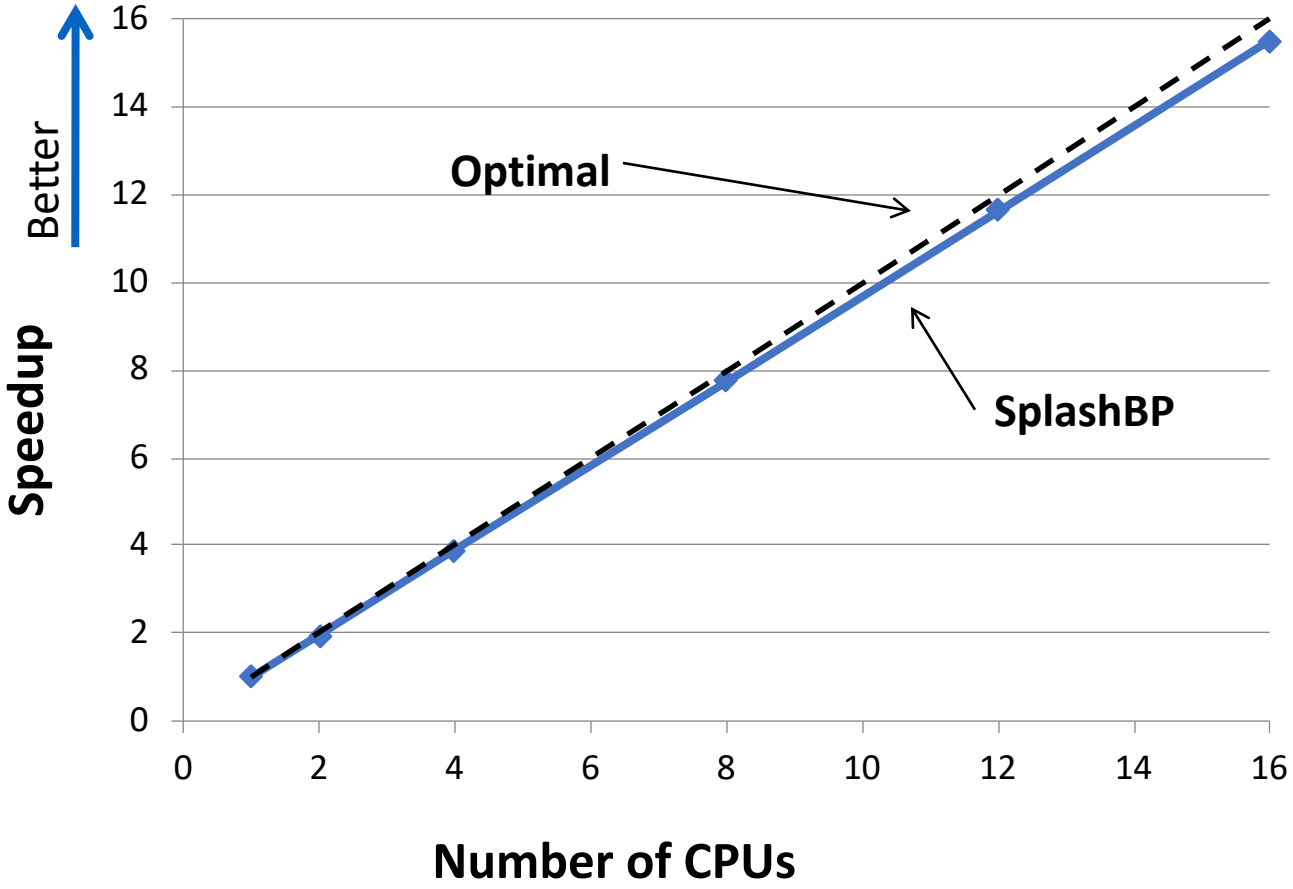
Vertices: 1 Million
Edges: 3 Million

Data Graph



Update Function:
Loopy BP Update Equation
Scheduler:
Approximate Priority
Consistency Model:
Edge Consistency

Loopy Belief Propagation

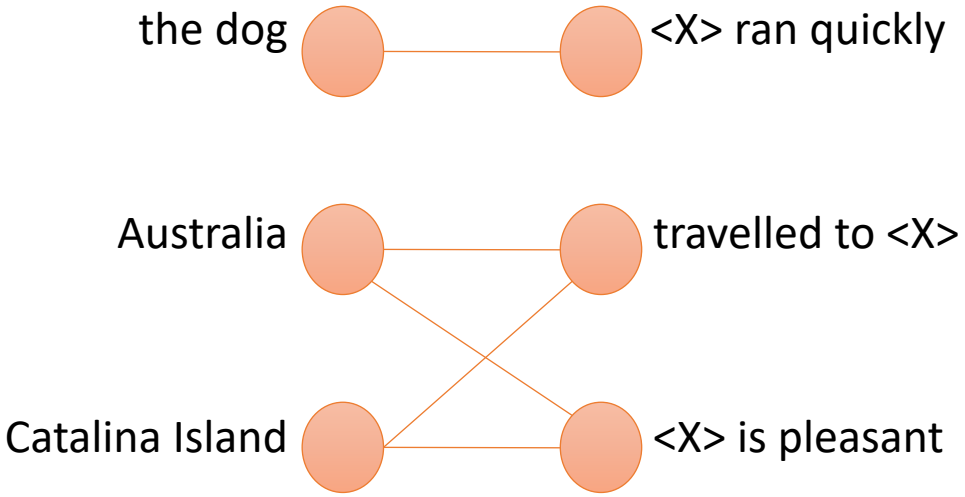


15.5x speedup

CoEM (Rosie Jones, 2005)

Named Entity Recognition Task

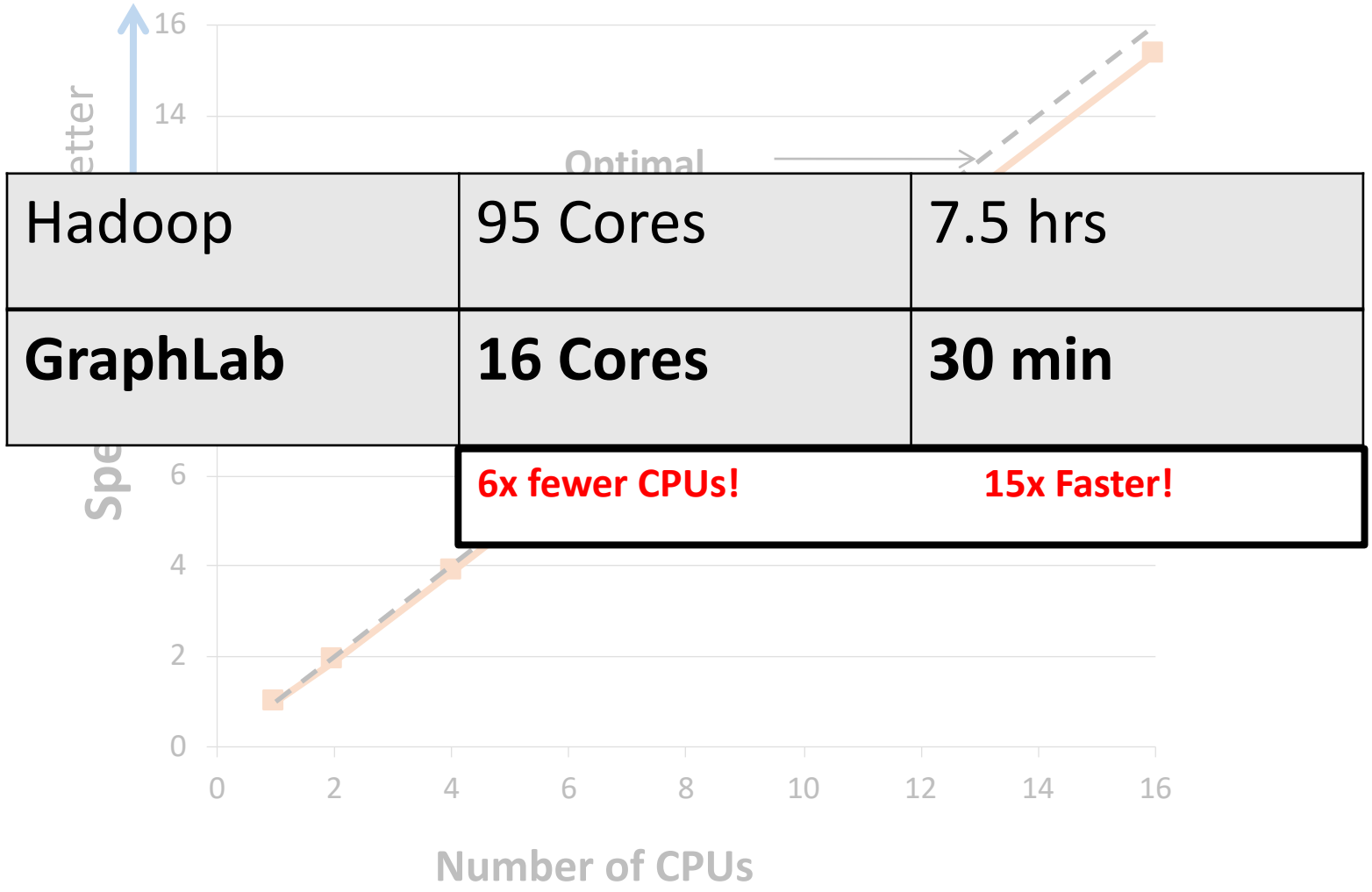
Is “Dog” an animal?
Is “Catalina” a place?



Vertices: 2 Million
Edges: 200 Million

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

CoEM (Rosie Jones, 2005)



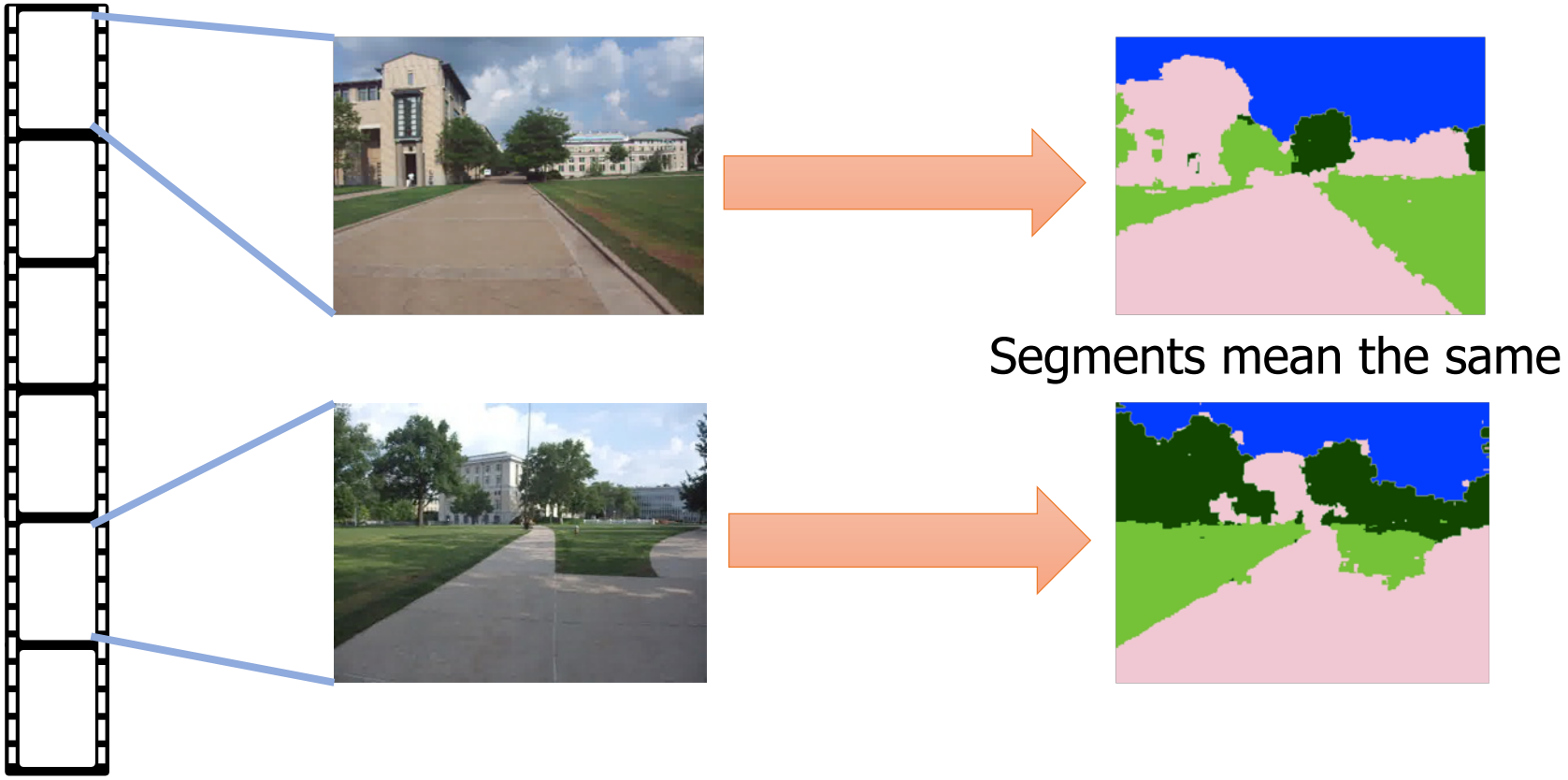
Hadoop	95 Cores	7.5 hrs
GraphLab	**16 Cores**	**30 min**

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

Experiments

Amazon EC2
High-Performance Nodes

Video Cosegmentation

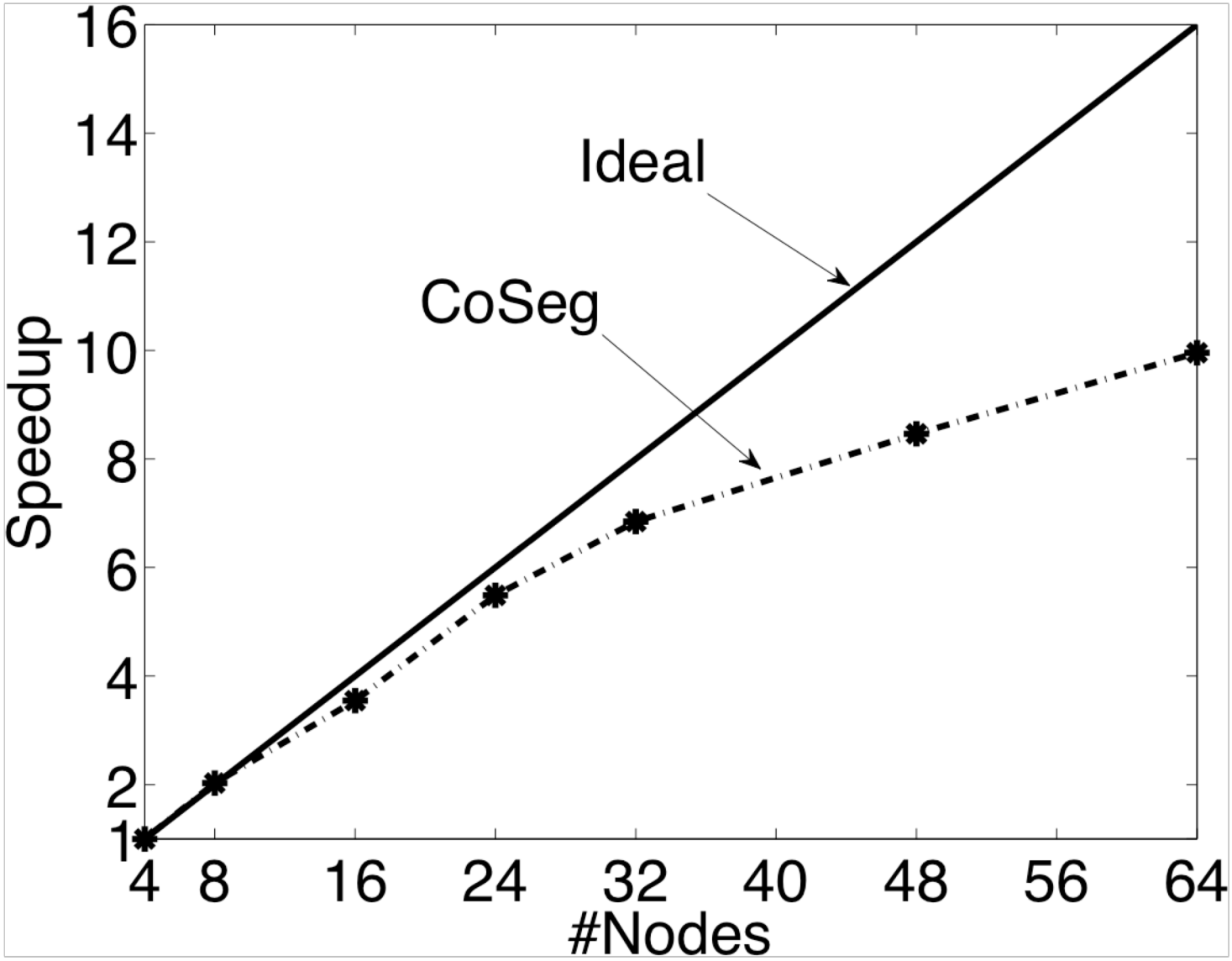


Segments mean the same

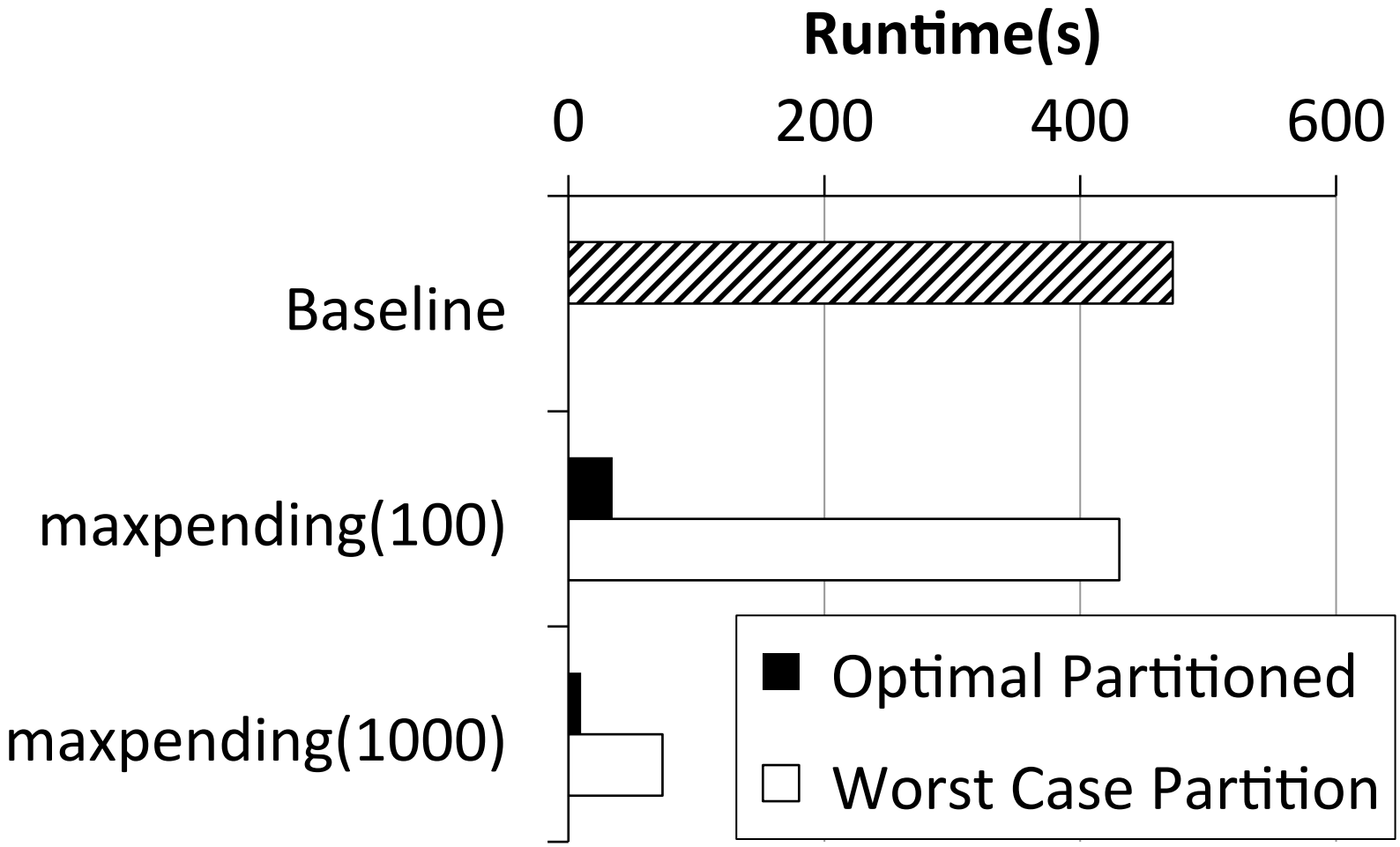
Gaussian EM clustering + BP on 3D grid

Model: 10.5 million nodes, 31 million edges

Video Coseg. Speedups



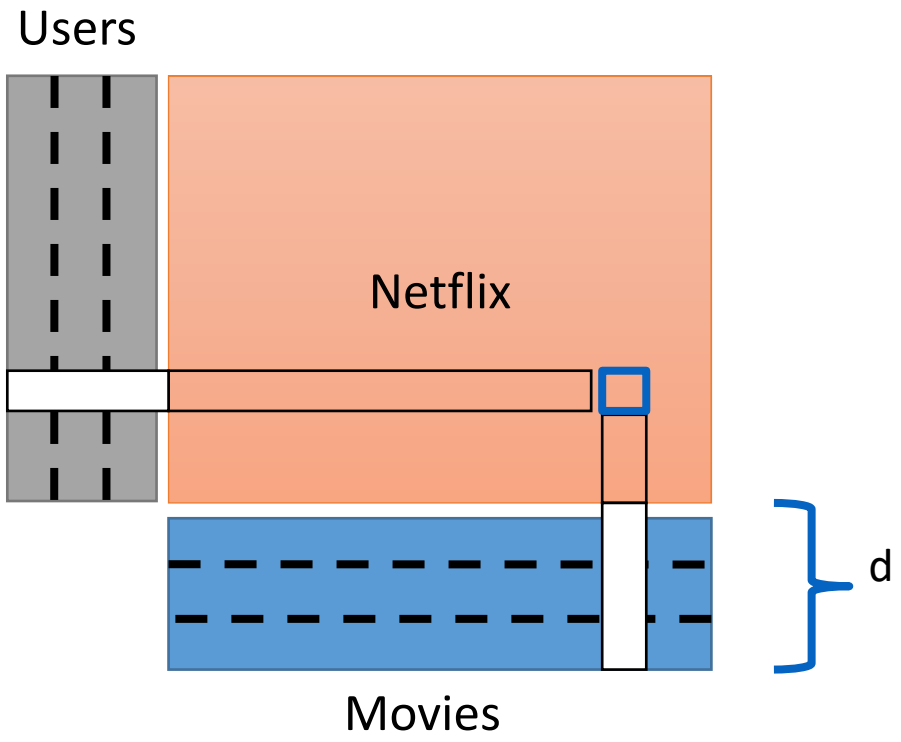
Prefetching Data & Locks



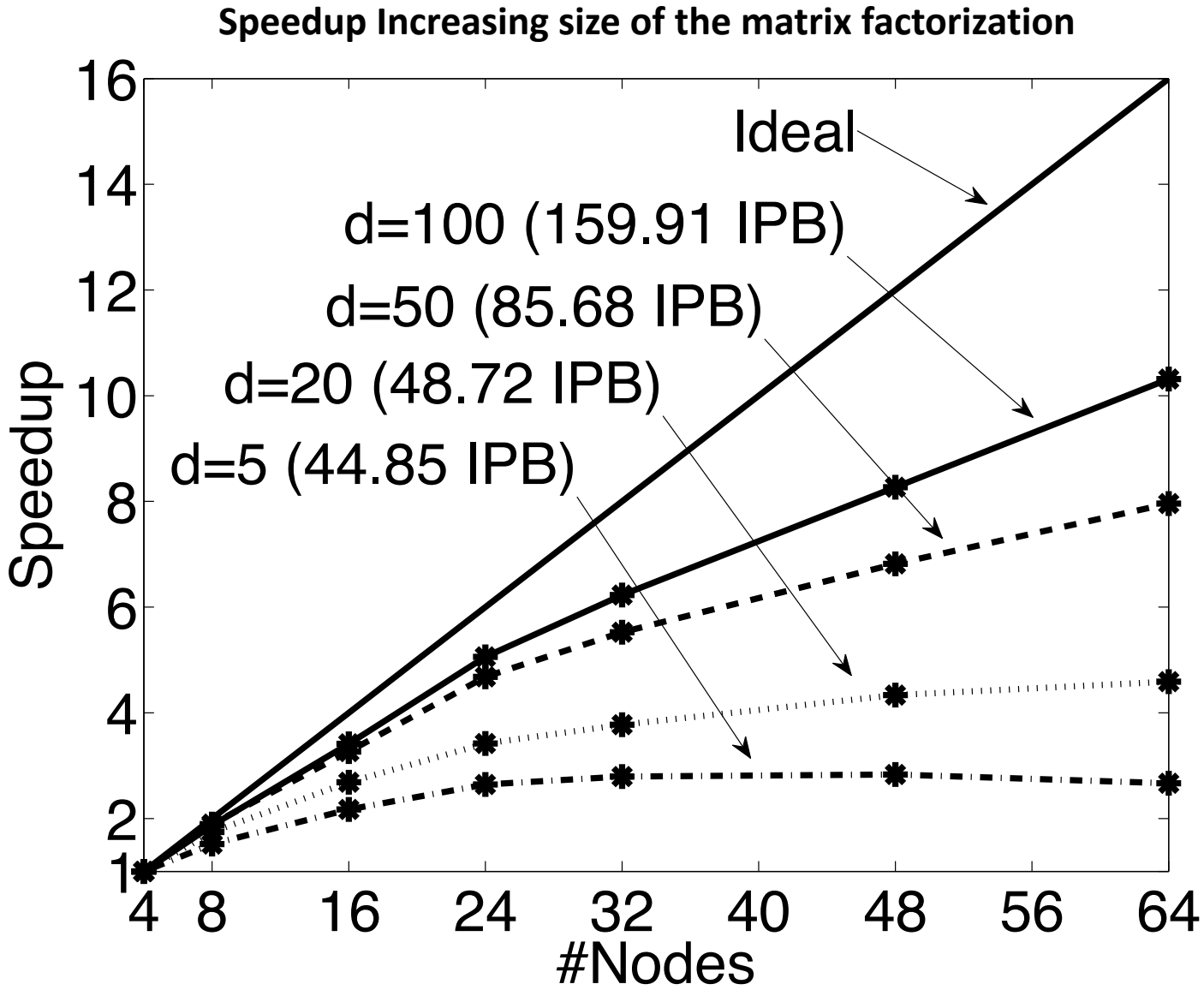
Matrix Factorization

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

Model: 0.5 million nodes, 99 million edges



Netflix


Speedup Increasing size of the matrix factorization
Ideal
d=100 (159.91 IPB)
d=50 (85.68 IPB)
d=20 (48.72 IPB)
d=5 (44.85 IPB)
Speedup
#Nodes
1
2
4
6
8
10
12
14
16
4
8
16
24
32
40
48
56
64

The Cost of Hadoop