Cristina Nita-Rotaru

7610 : Distributed Systems

AI.

Slides based on material by Prof. Ken Birman, for CS5412, and authors of TensorFlow and authors of GraphLab

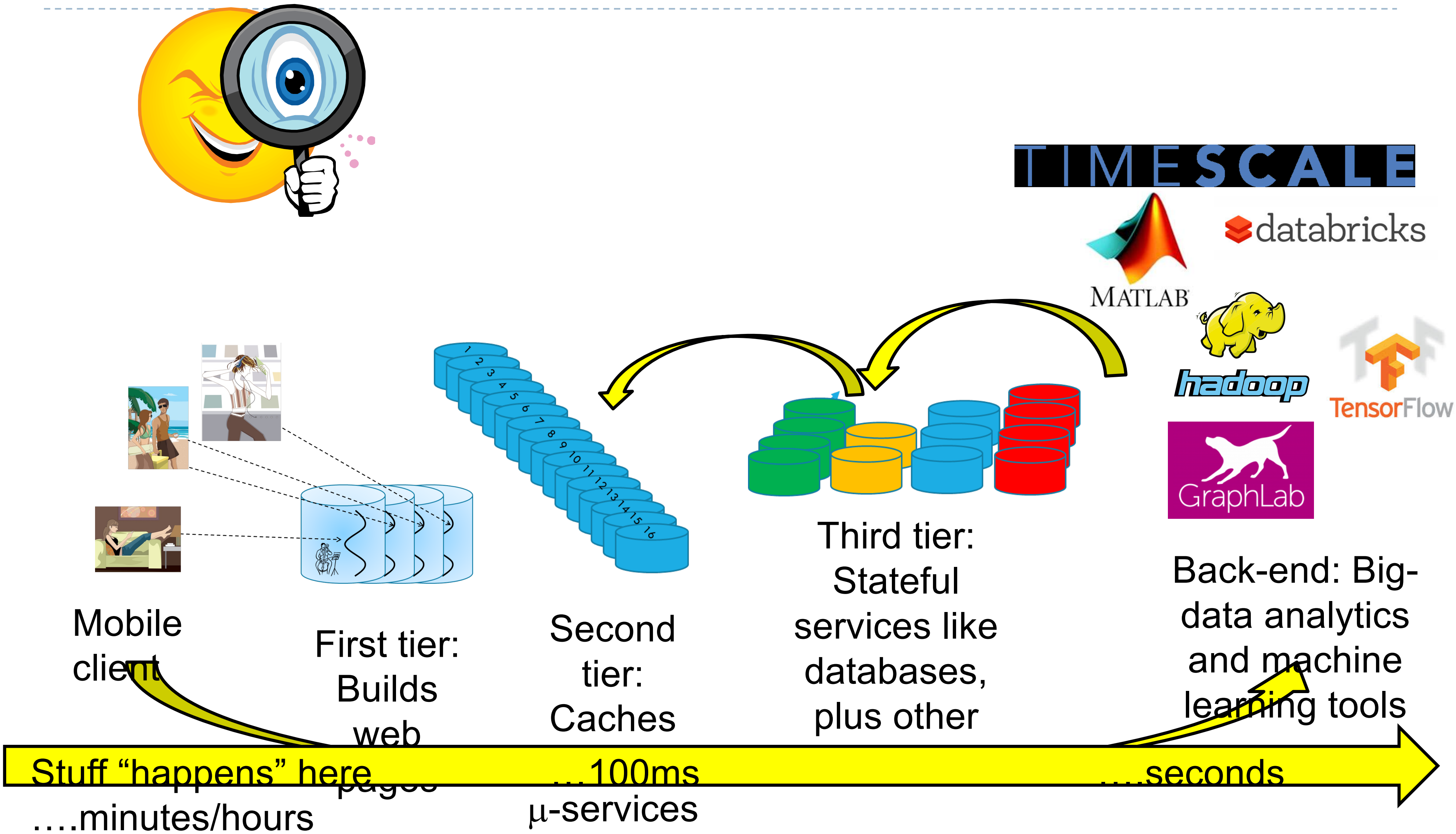
Required reading for this topic…

- Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud, VLDB 2012
- Pregel: A System for Large-Scale Graph Processing, SIGMOD 2010
- TensorFlow: A System for Large-Scale Machine Learning OSDI 2016
- Scaling Distributed Machine Learning with the Parameter Server, OSDI 2014

Clouds and machine learning tools

- Early cloud just served web pages and embedded ads
- However, individualized advertising gives far better results… (and they increase revenue)
- Better selection of ads gave rise to an AI revolution
  - Individual actions
  - Social networking "graphs"
- Today, the whole cloud is a massive scalable system for machine learning and associated actions

Where does the AI live?



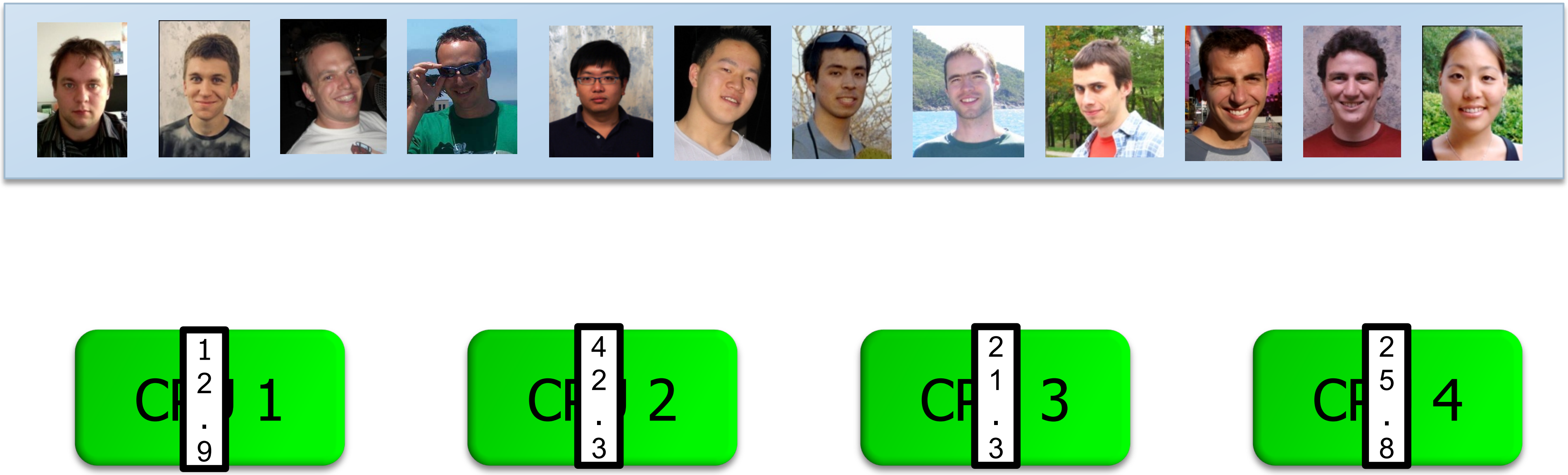
http://www.cs.cornell.edu/courses/cs5412/2018sp

How to support ML algorithms at scale

- Old approach:
  - threads, locks, messages
- Newer approach:
  - MapReduce, Spark
- When is MapReduce the right approach?
- When MapReduce does not work well?
- Design new abstractions and systems to support ML development and running at scale
  - GraphLab, created at CMU, eventually bought by Apple
  - TensorFlow, created by GoogleBrain

1:Why Map-Reduce is not the best approach for ML applications

MapReduce – Map Phase



Embarrassingly Parallel independent computation
No Communication needed

MapReduce – Map Phase

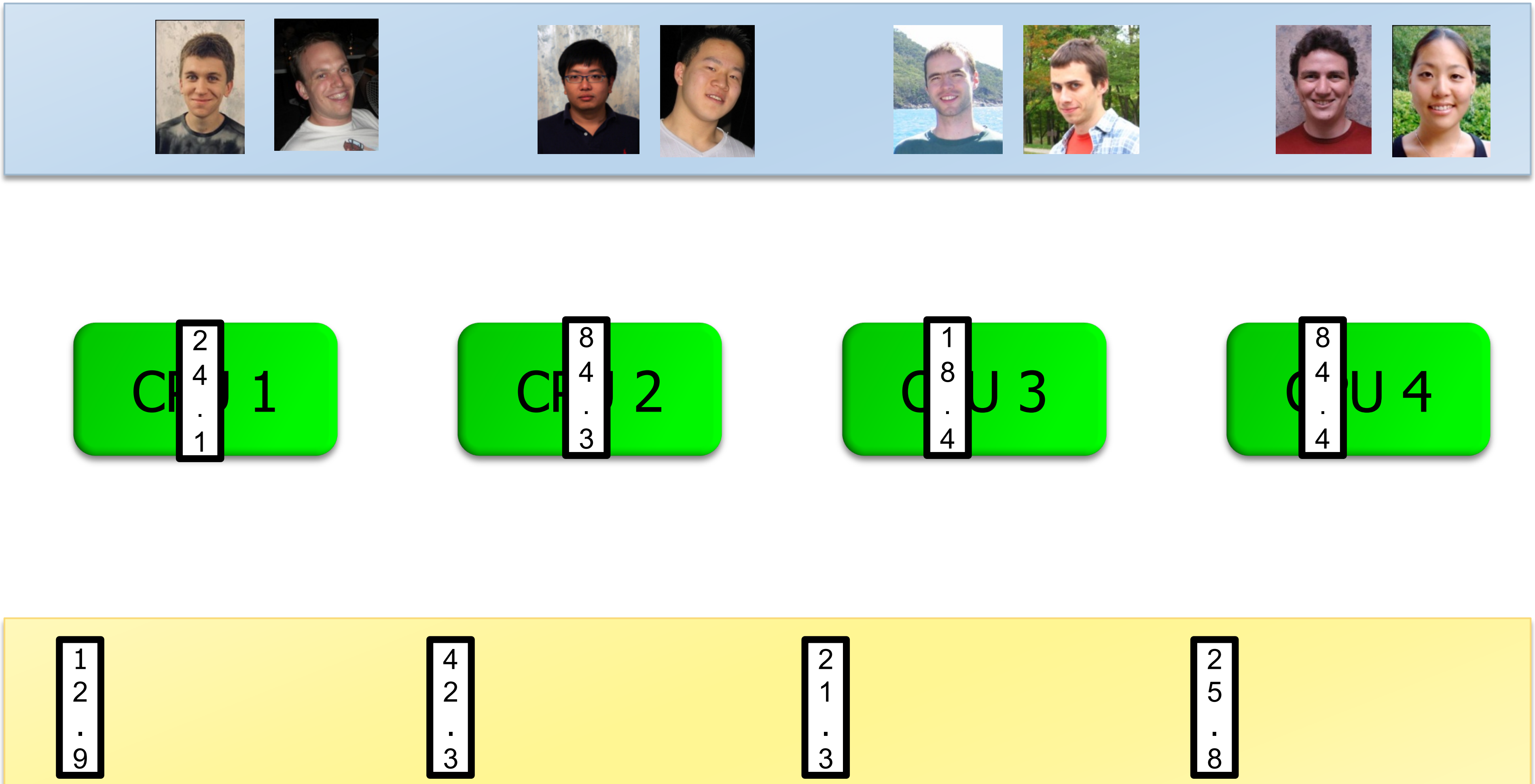
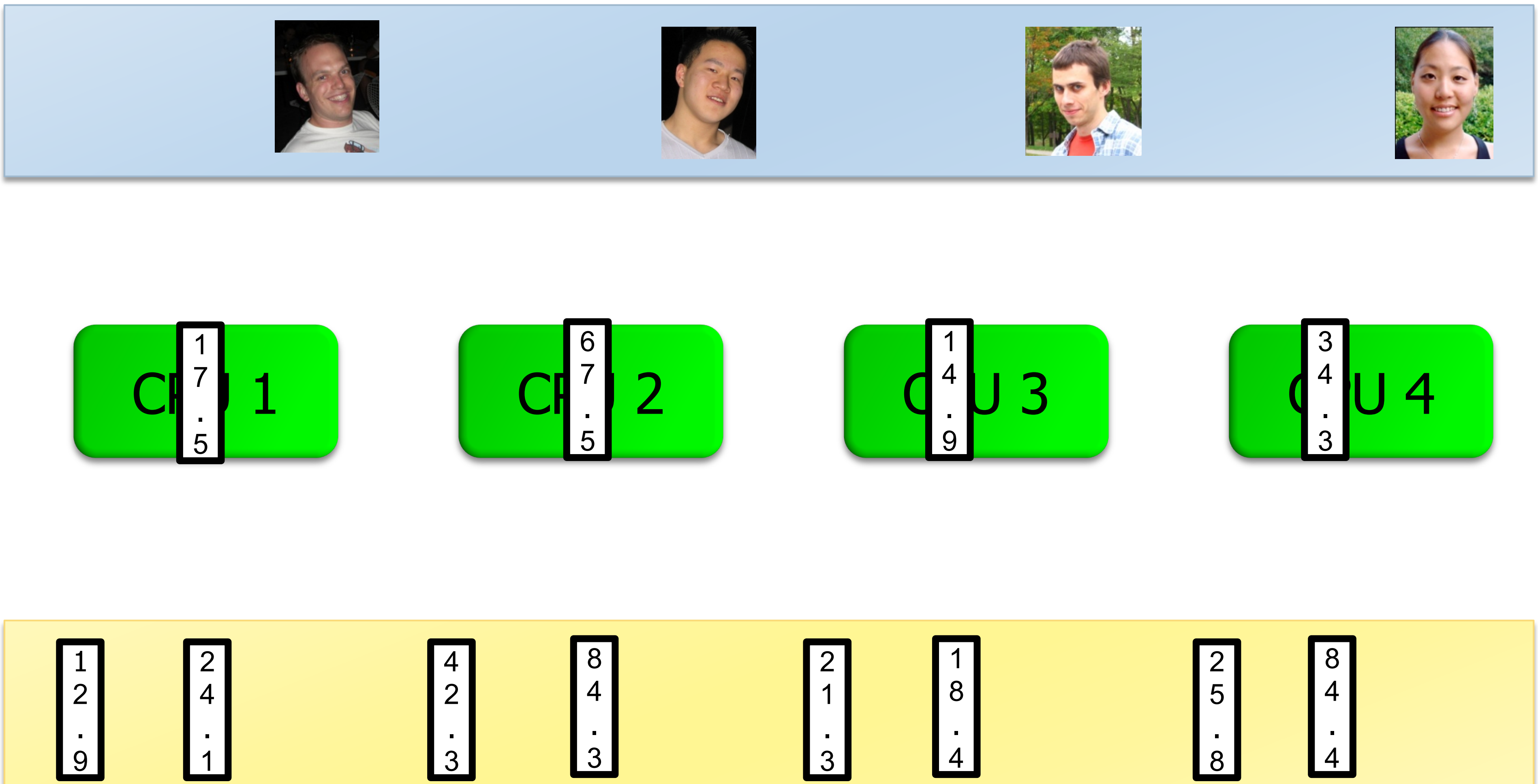


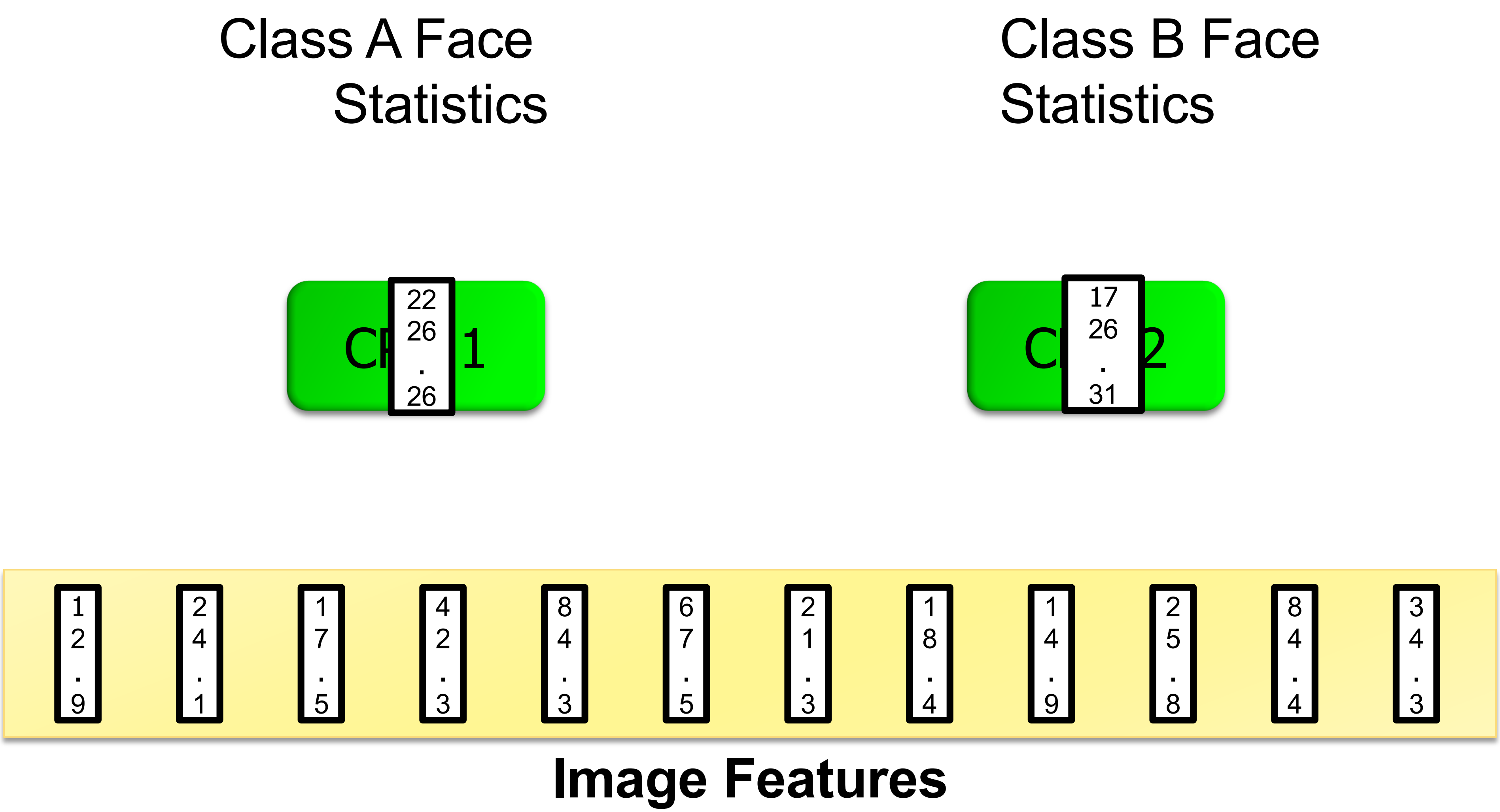
Image Features

MapReduce – Map Phase



Embarrassingly Parallel independent computation
No Communication needed

MapReduce – Reduce Phase



Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Data-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Is there more to Machine Learning

?

Label propagation algorithm

- Social Arithmetic:

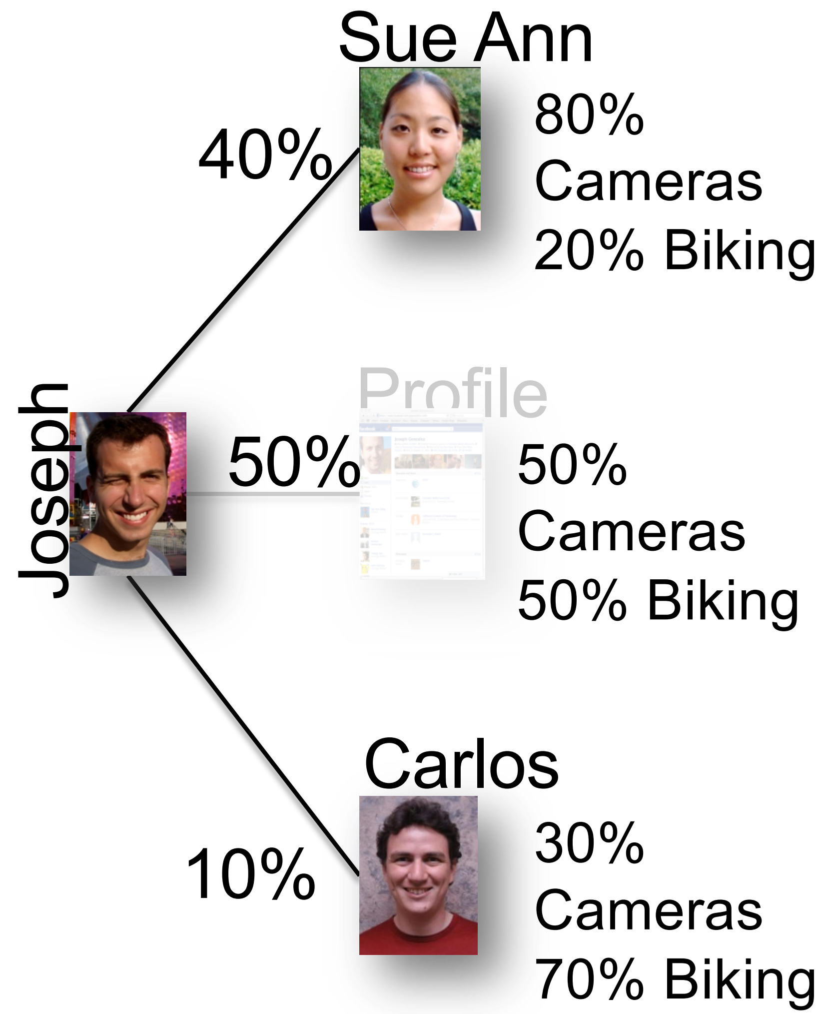
  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

  I Like: 60% Cameras, 40% Biking

- Recurrence Algorithm:

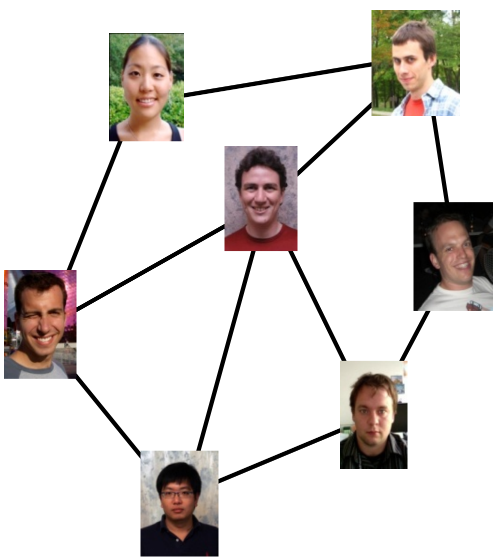
$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

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

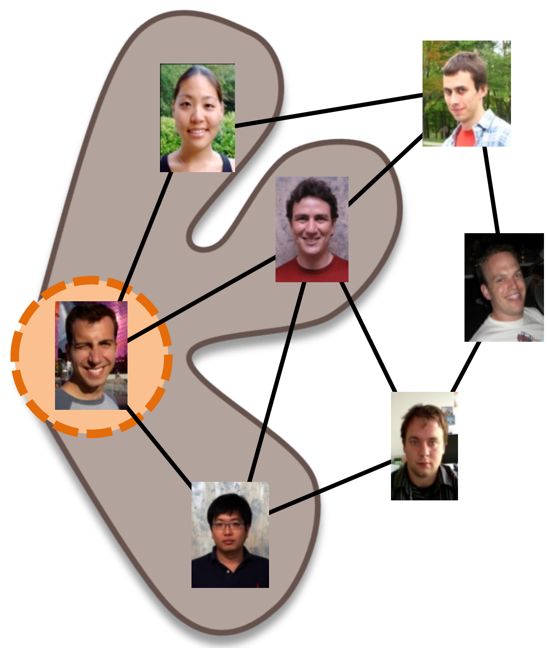


Properties of Graph Parallel Algorithms

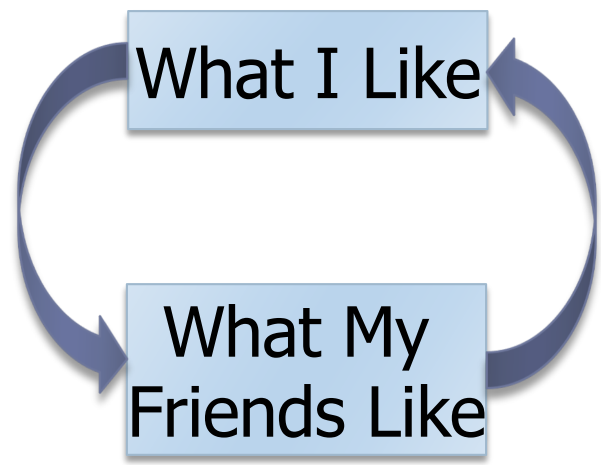
Dependency Graph



Factored Computation



Iterative Computation



Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Data-Parallel Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Map Reduce?

Lasso

Label Propagation

Kernel Methods

Belief Propagation

Tensor Factorization

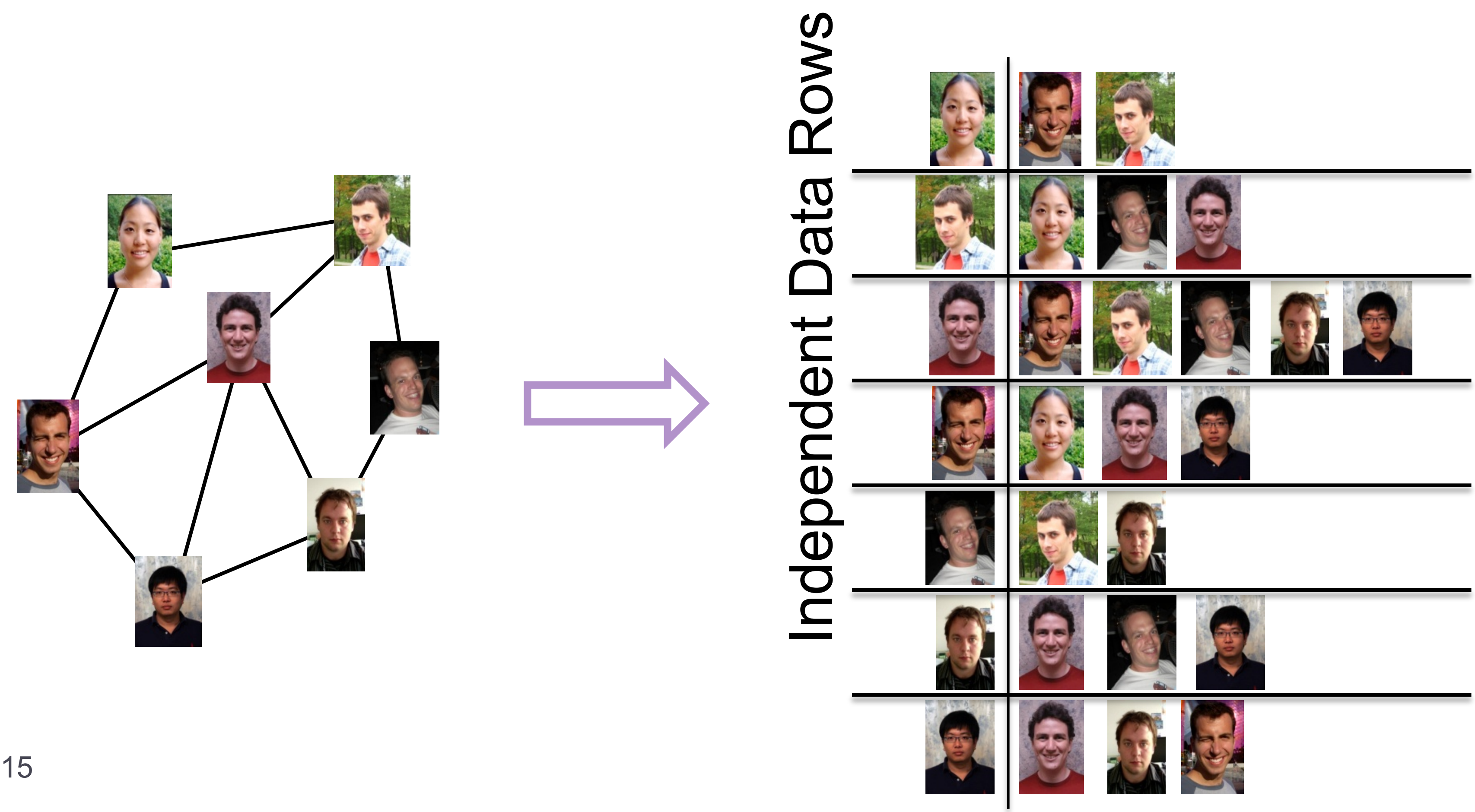
PageRank

Deep Belief Networks

Neural Networks

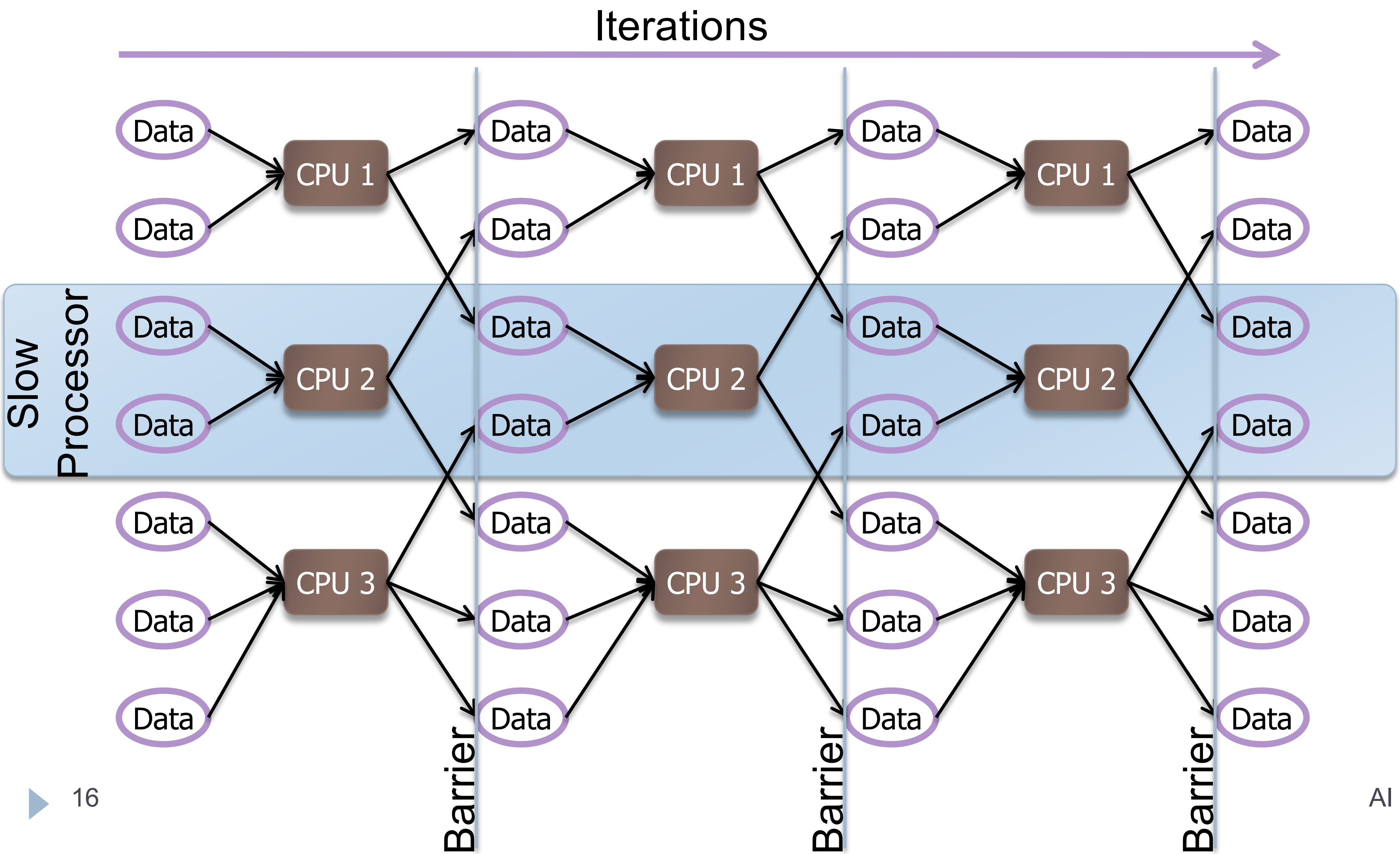
Limitations of MR: Data Dependencies

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



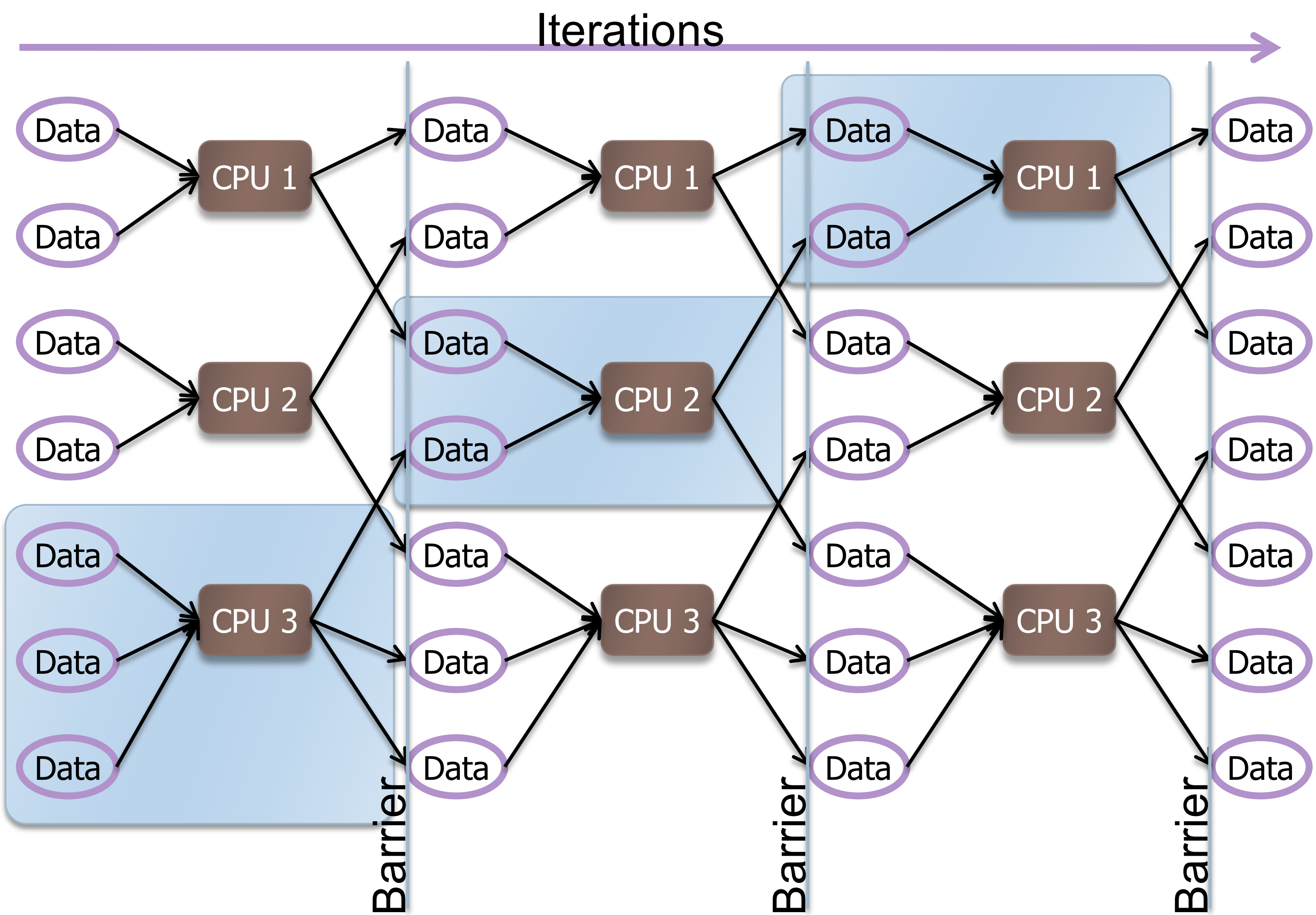
Limitations of MR: Iterative Algorithms

- Map-Reduce does not efficiently express iterative algorithms:



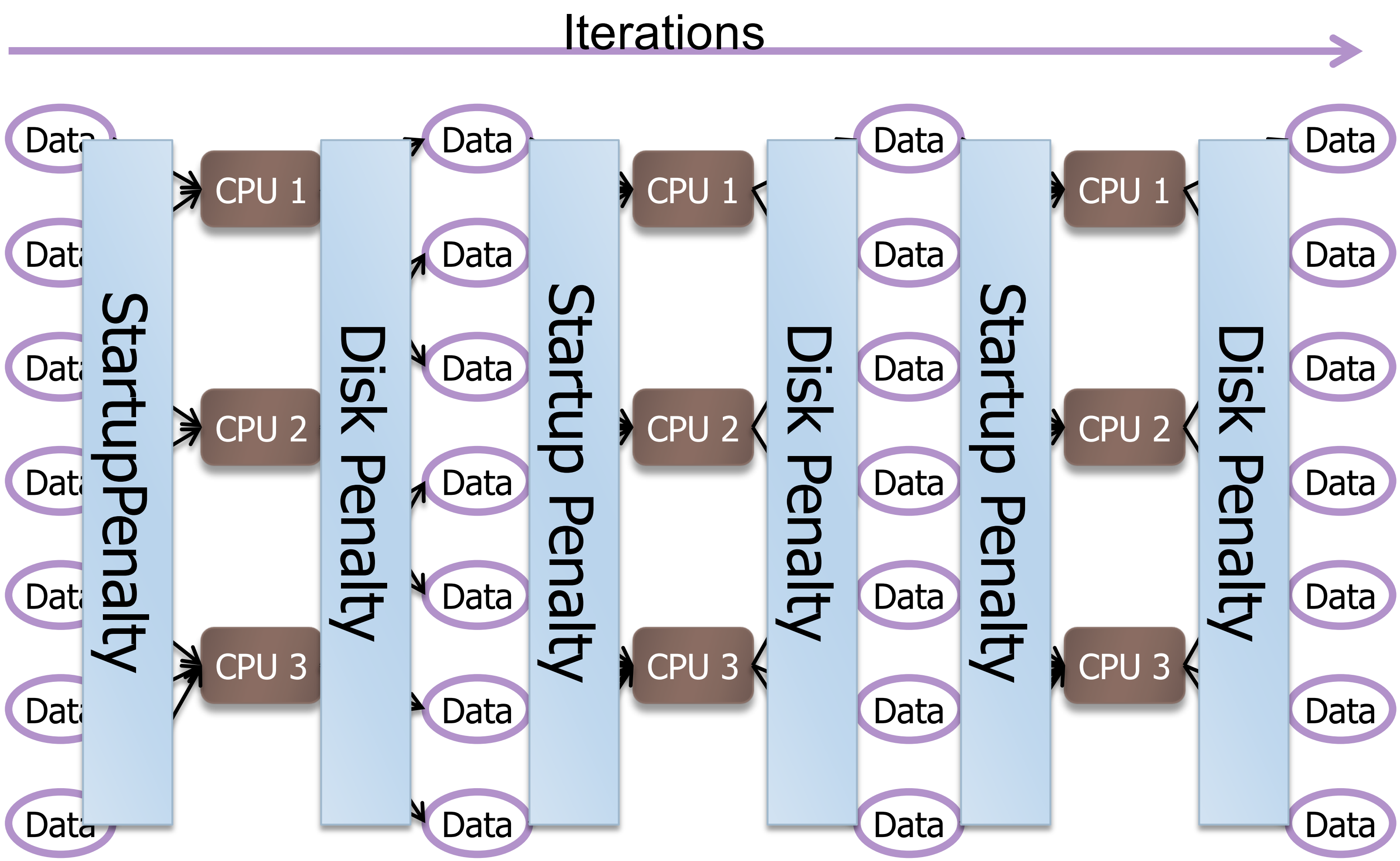
Iterative MapReduce

- Only a subset of data needs computation:



Iterative MapReduce

- System is not optimized for iteration:



Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Data-Parallel Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (Giraph)?

SVM

Lasso

Kernel Methods

Belief Propagation

Tensor Factorization

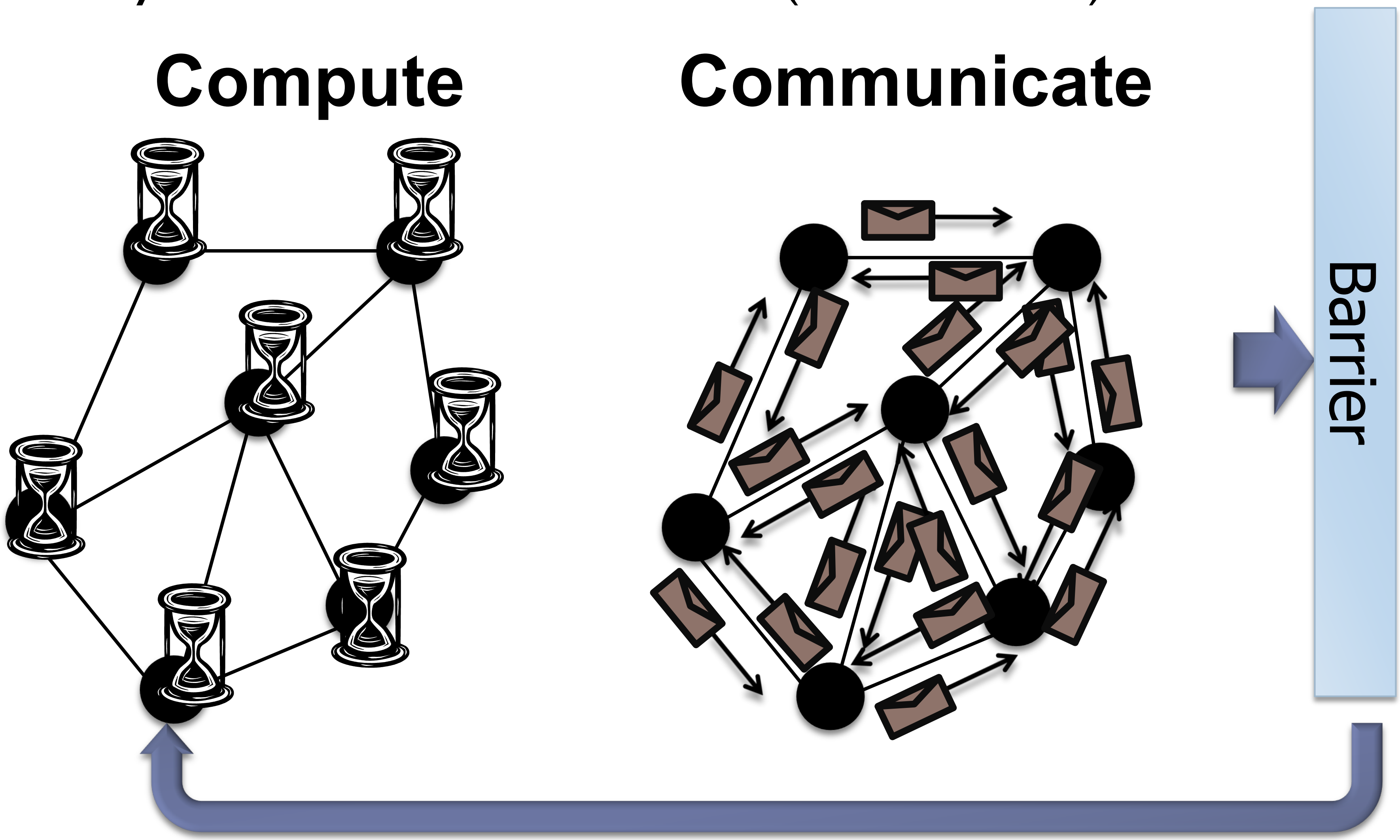
PageRank

Deep Belief Networks

Neural Networks

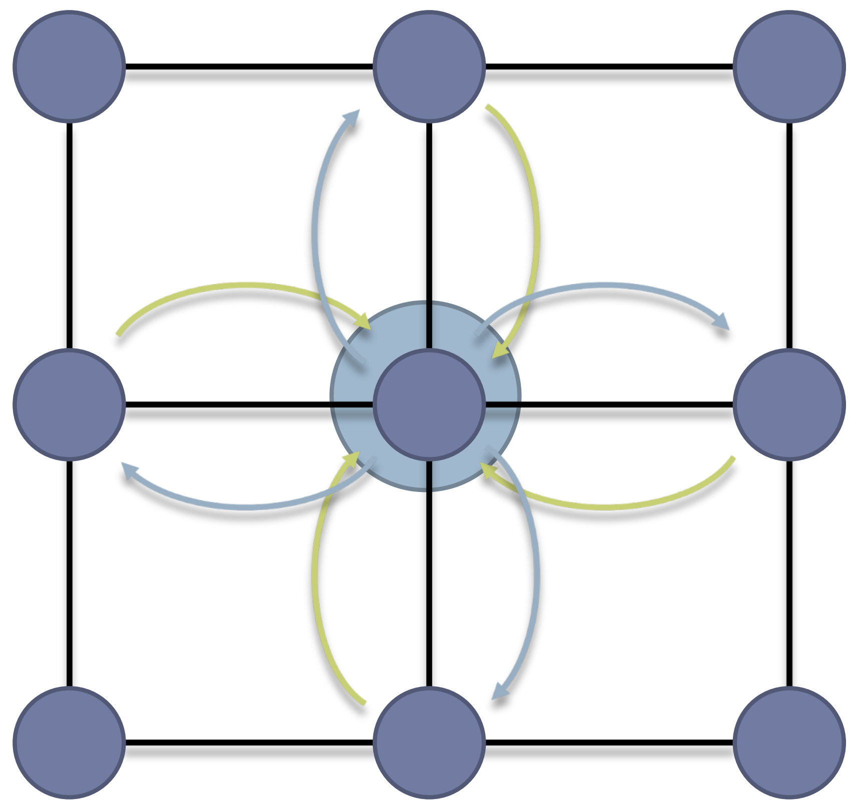
Pregel (Giraph)

- Bulk Synchronous Parallel Model (Valiant 1990):



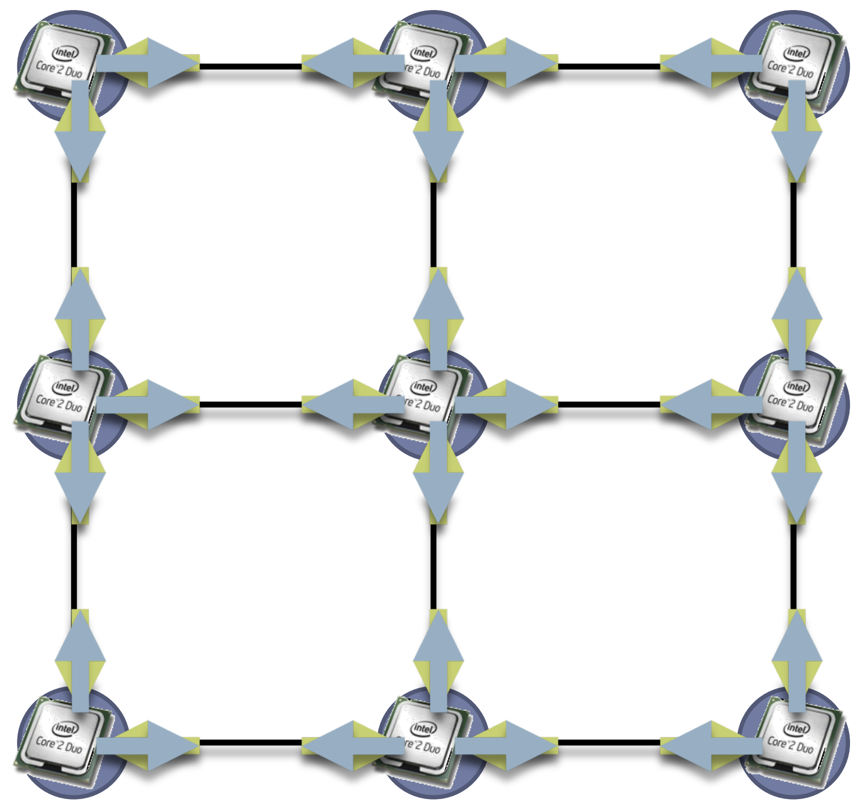
Loopy Belief Propagation (Loopy BP)

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

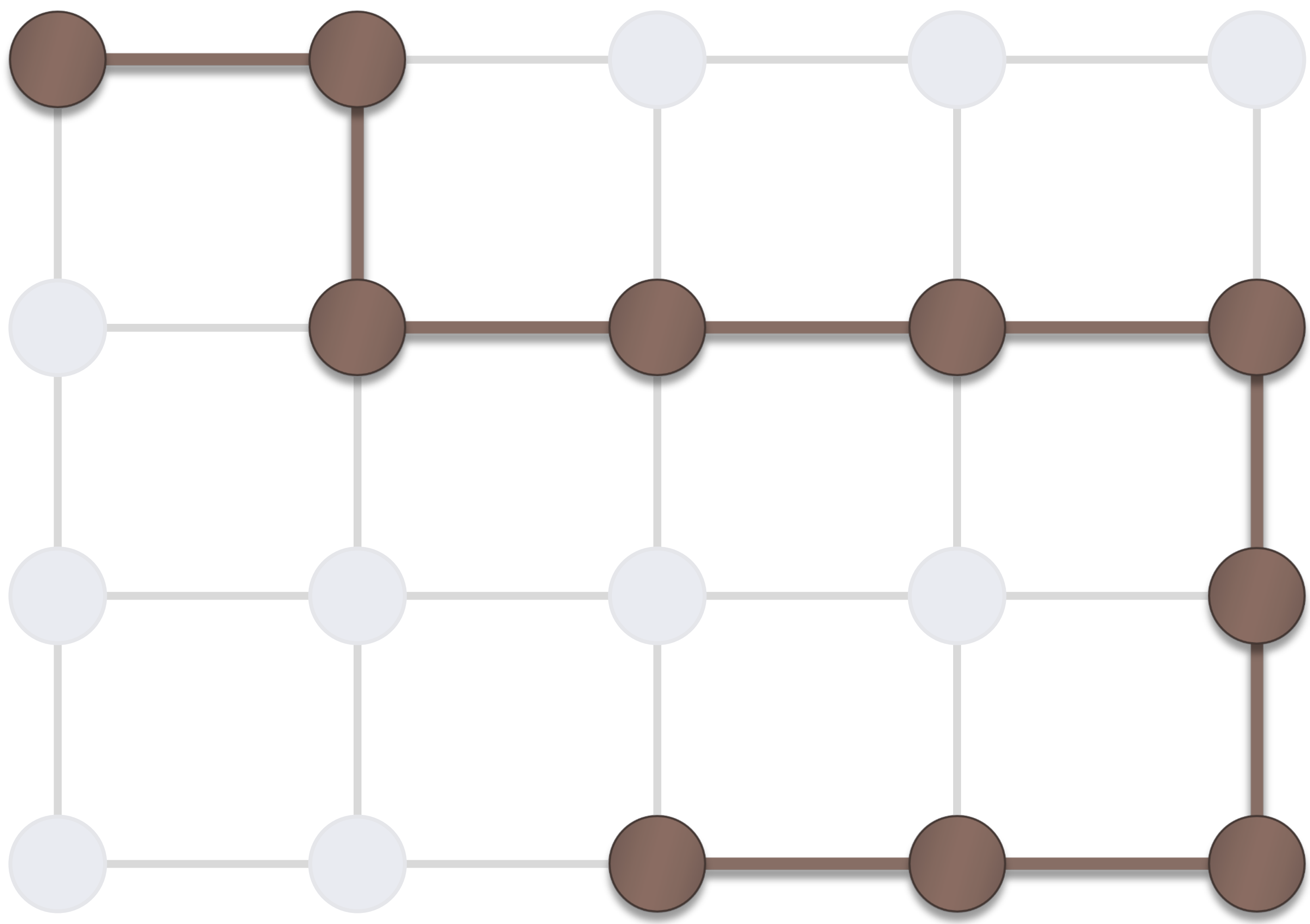


Bulk Synchronous Loopy BP

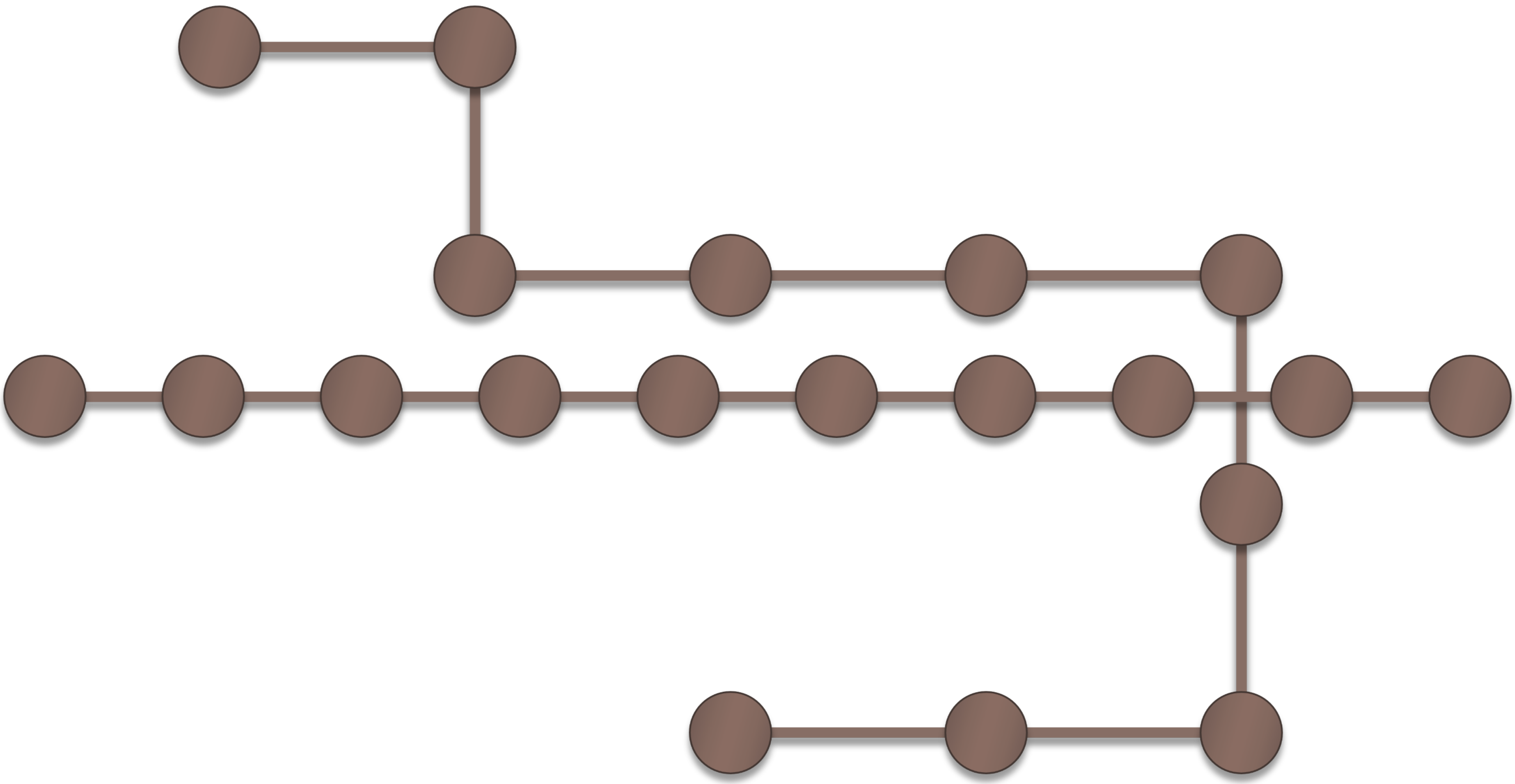
- Often considered embarrassingly parallel
  - Associate processor with each vertex
  - Receive all messages
  - Update all beliefs
  - Send all messages
- Proposed by:
  - Brunton et al. CRV'06
  - Mendiburu et al. GECC'07
  - Kang,et al. LDMTA'10
  - …



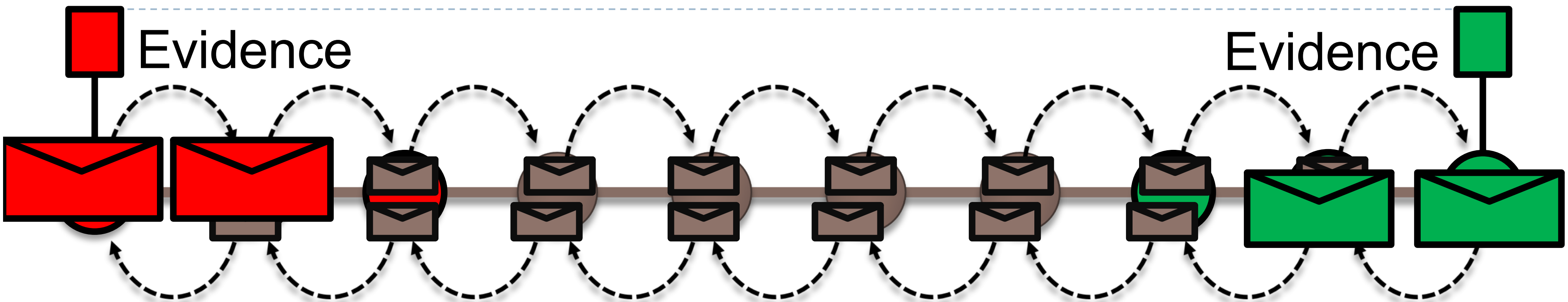
Sequential Computational Structure



Hidden Sequential Structure



Hidden Sequential Structure



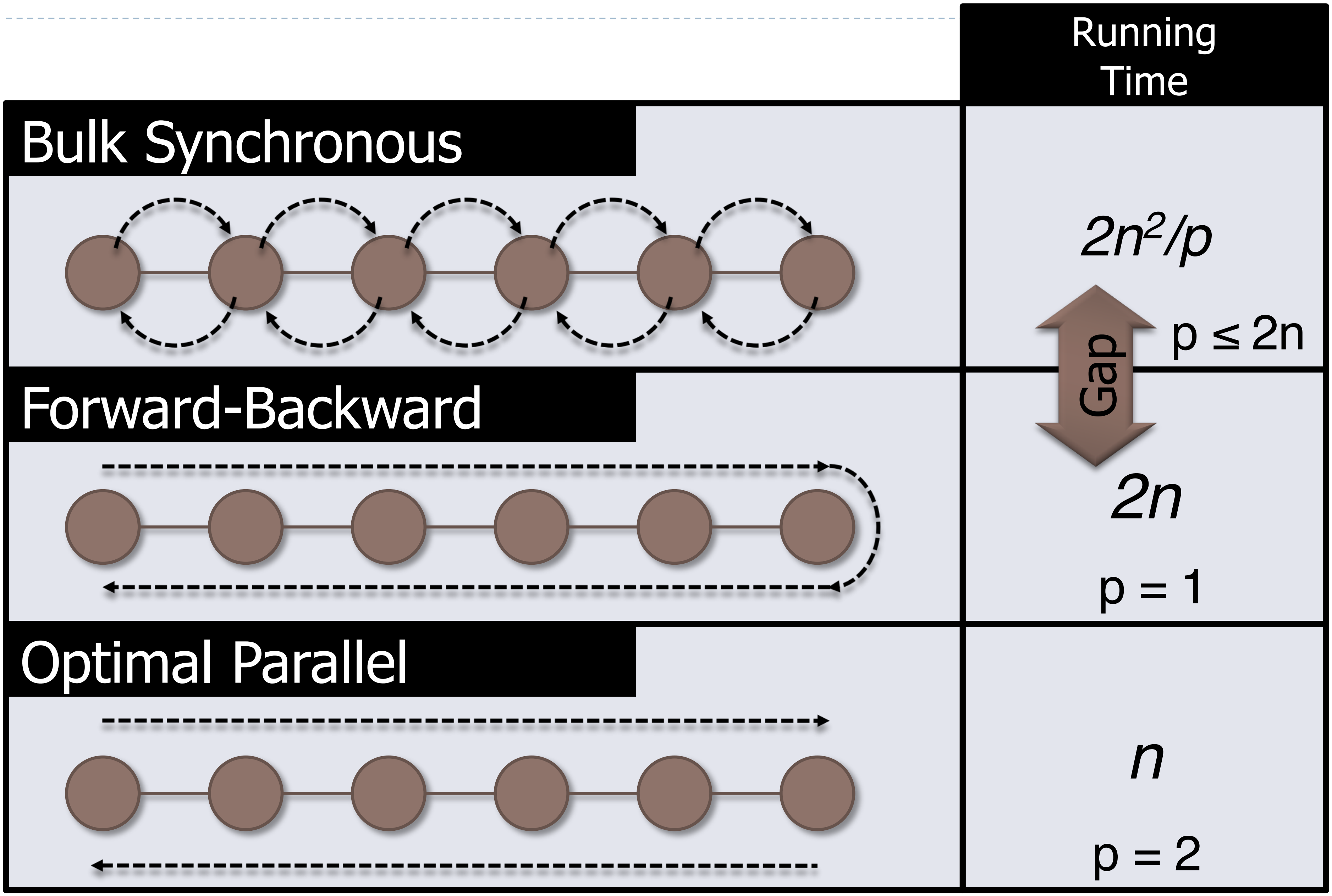
- Running Time:

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

Time for a single parallel iteration

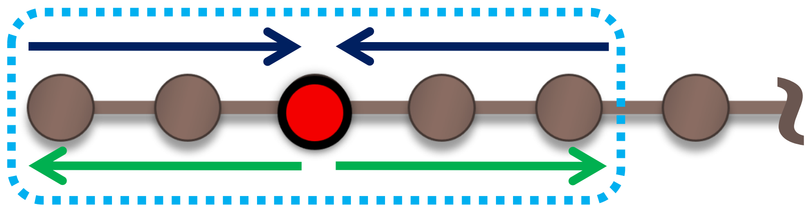
Number of Iterations

Optimal Sequential Algorithm



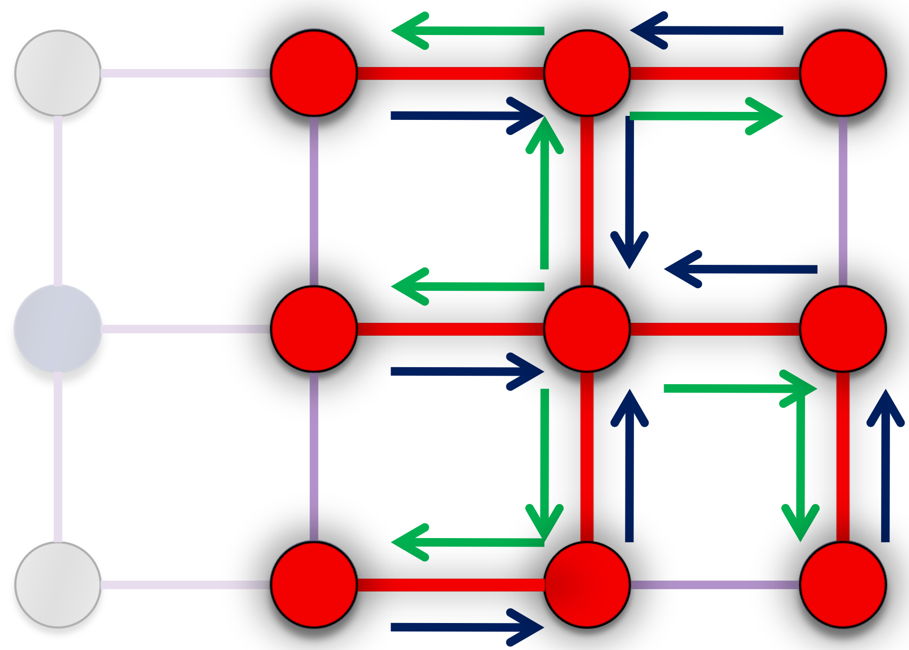
The Splash Operation

- 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



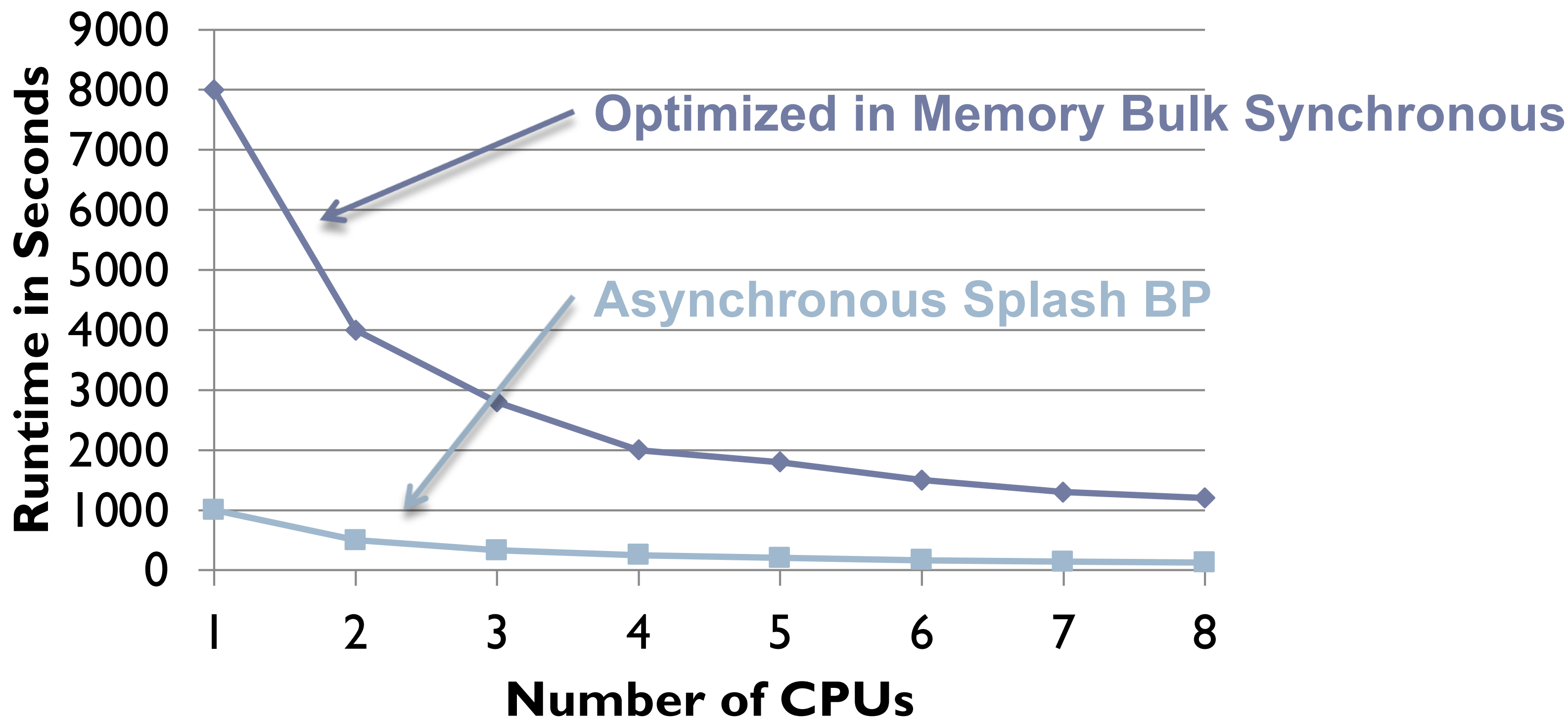
Data-Parallel algorithms can be inefficient

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.

Need a new abstraction

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

Data-Parallel Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Pregel (Giraph)

GraphLab
Carnegie Mellon

SVM

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

Lasso

Deep Belief Networks

Neural Networks

2:GraphLab

The GraphLab Framework

Graph Based
Data Representation

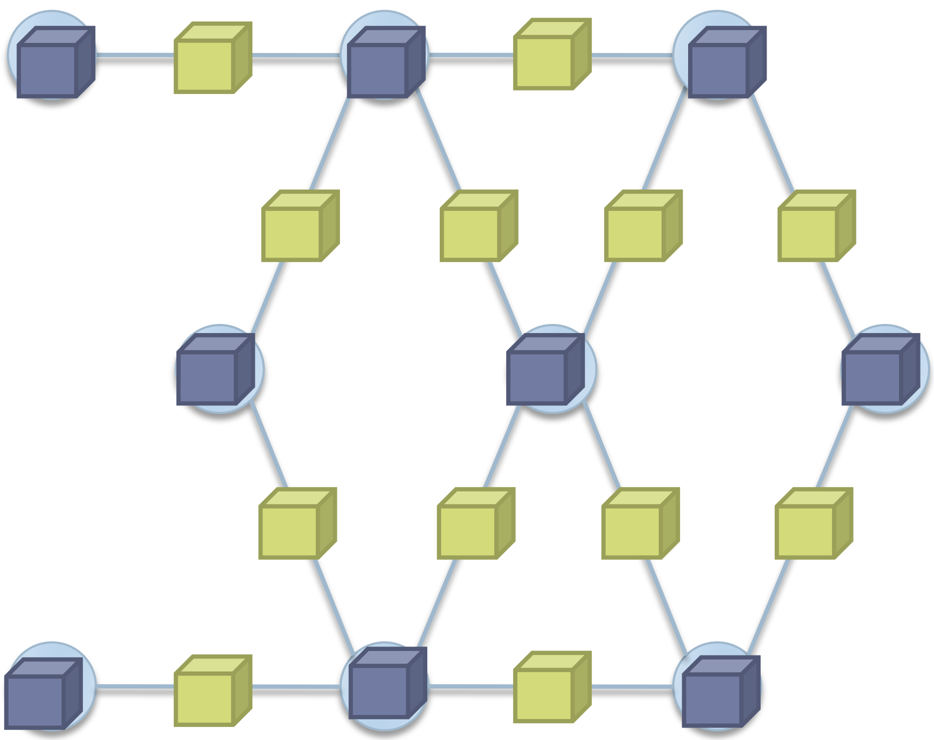
Update Functions
User Computation

Scheduler

Consistency

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

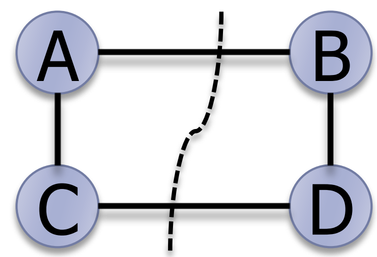
Implementing the Data Graph

Multicore Setting

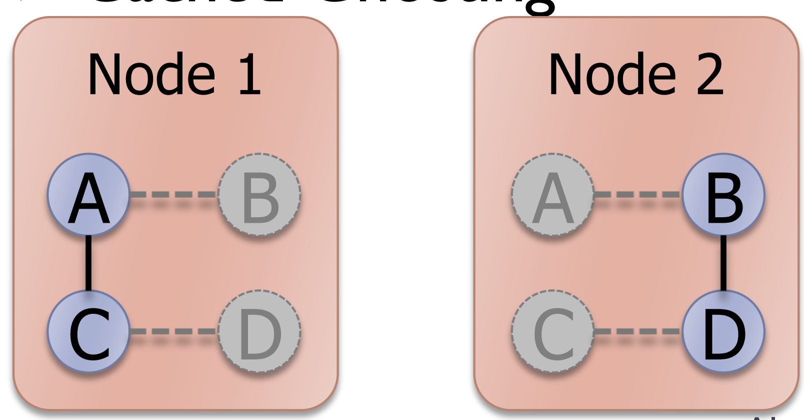
- **In Memory**
- Relatively Straight Forward
  - vertex_data(vid) → data
  - edge_data(vid,vid) → data
  - neighbors(vid) → vid_list
- Challenge:
  - Fast lookup, low overhead
- Solution:
  - Dense data-structures
  - Fixed Vdata&Edata types
  - Immutable graph structure

Cluster Setting

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



- Cached Ghosting



The GraphLab Framework

Graph Based
Data Representation

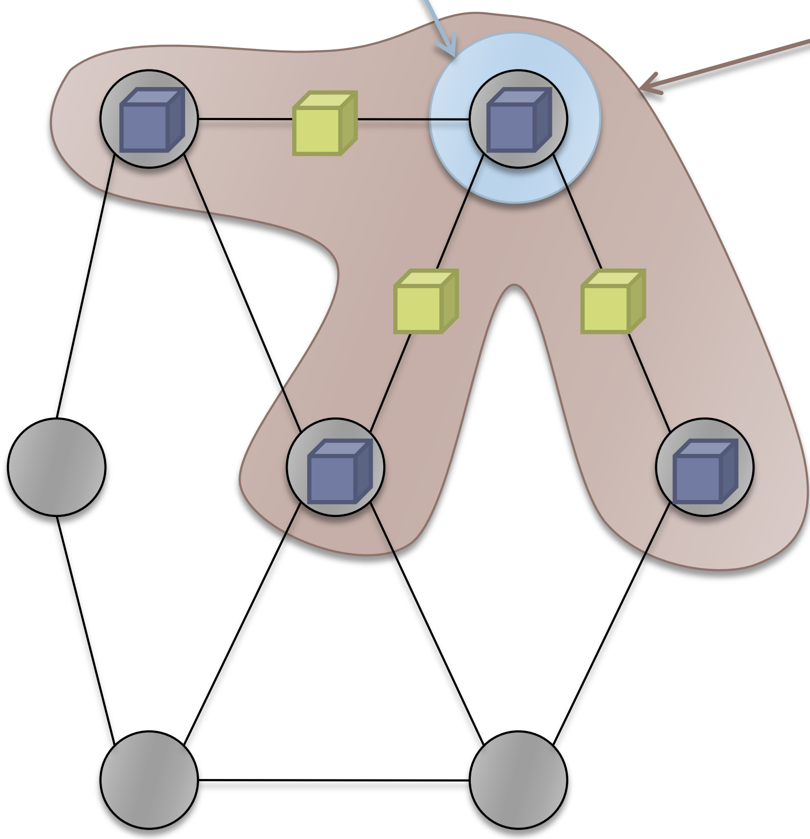
Update Functions
User Computation

Scheduler

Consistency

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], Wij, Likes[j]) ←scope

// Update the vertex data
```

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

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

The GraphLab Framework

Graph Based
Data Representation

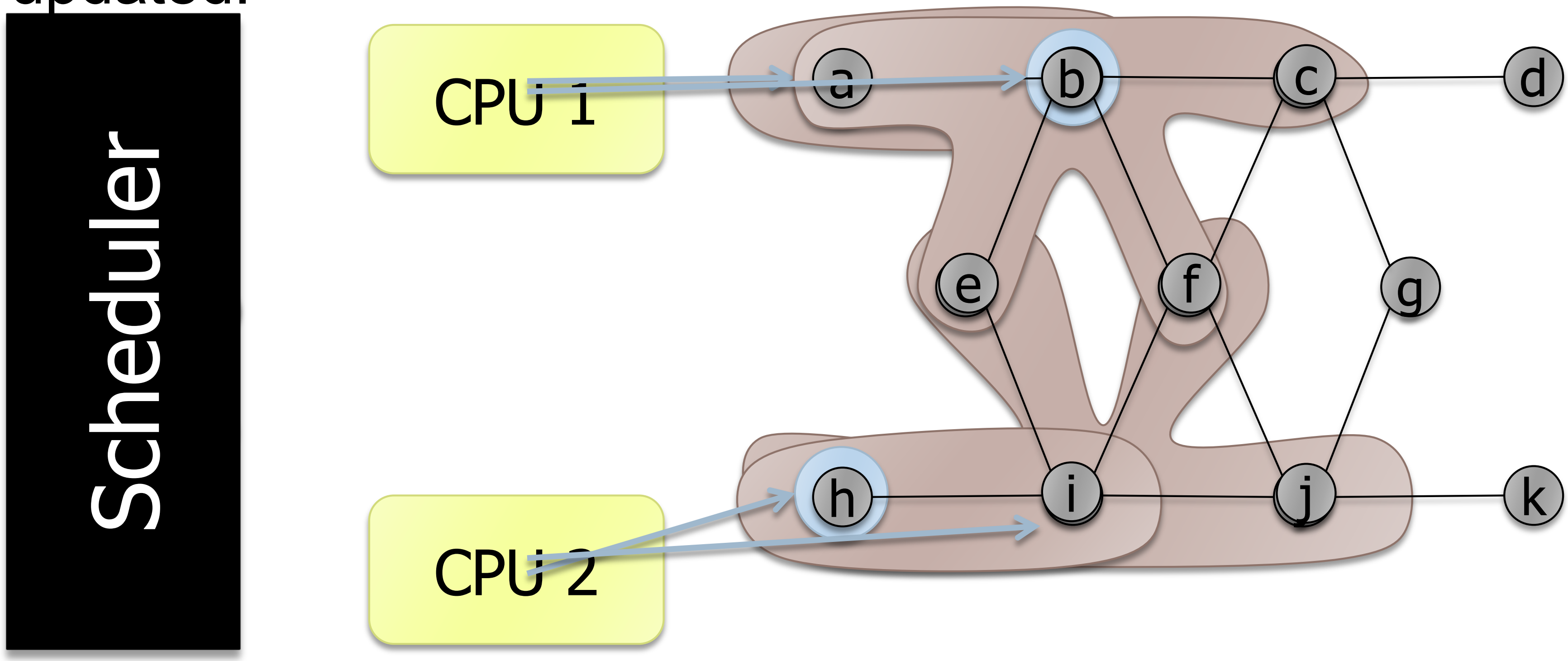
Update Functions
User Computation

Scheduler

Consistency

The Scheduler

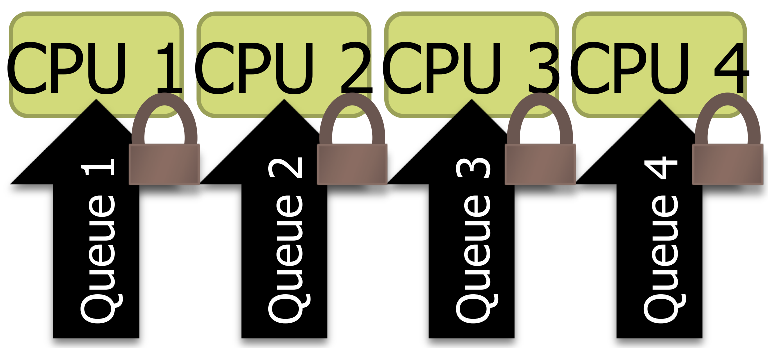
The **scheduler** determines the order that vertices are updated.



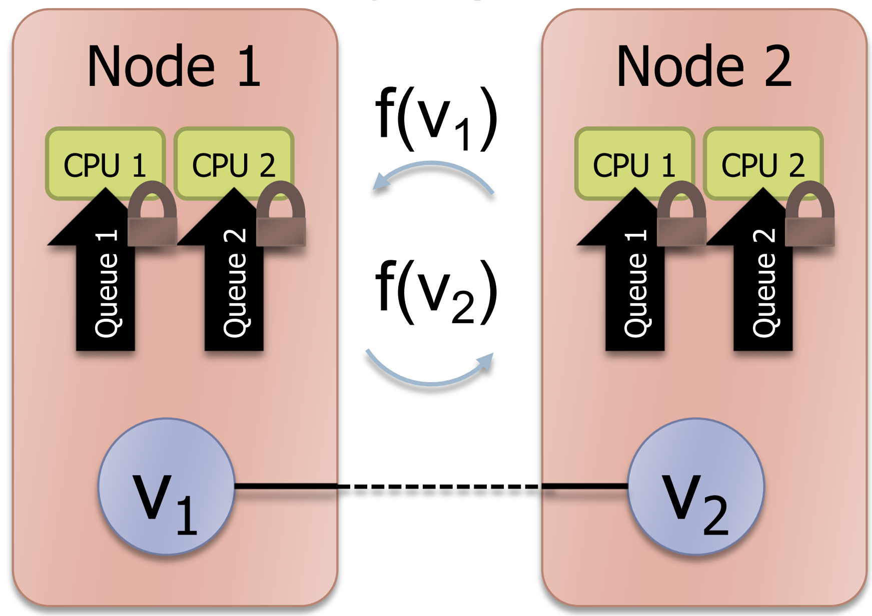
The process repeats until the scheduler is empty.

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

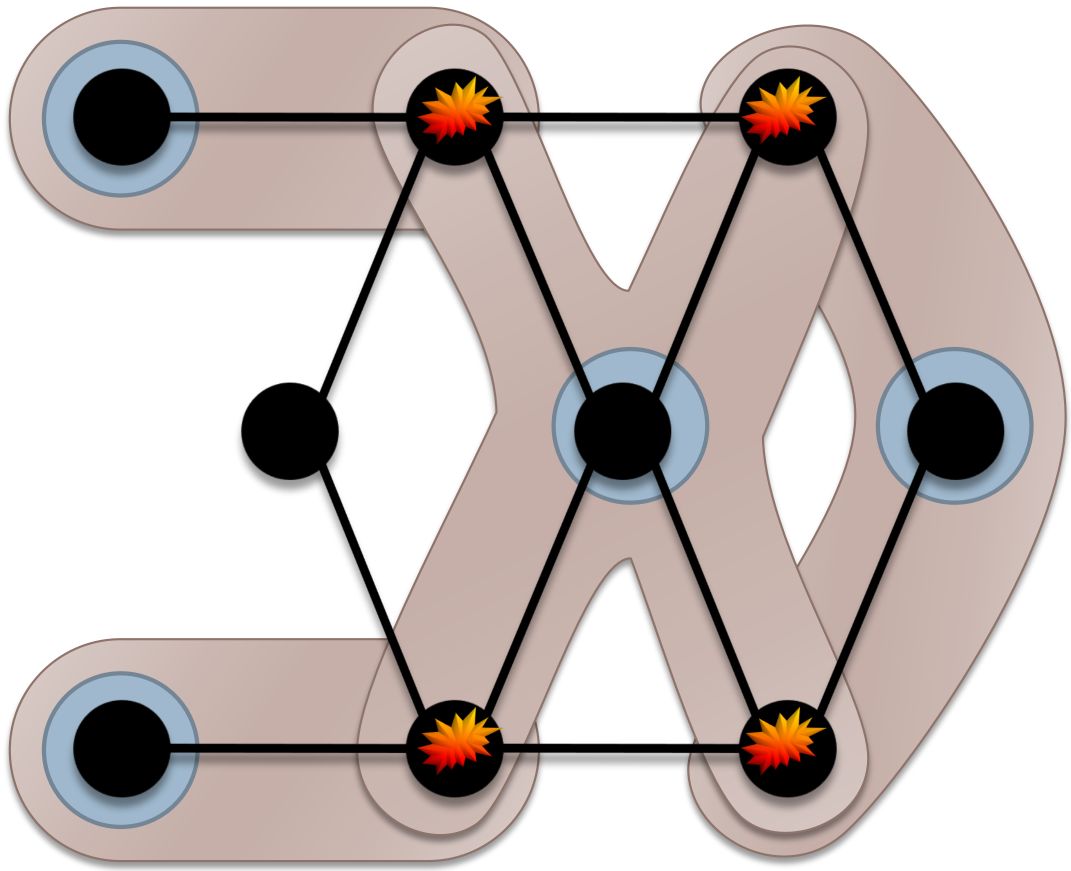
Update Functions
User Computation

Scheduler

Consistency

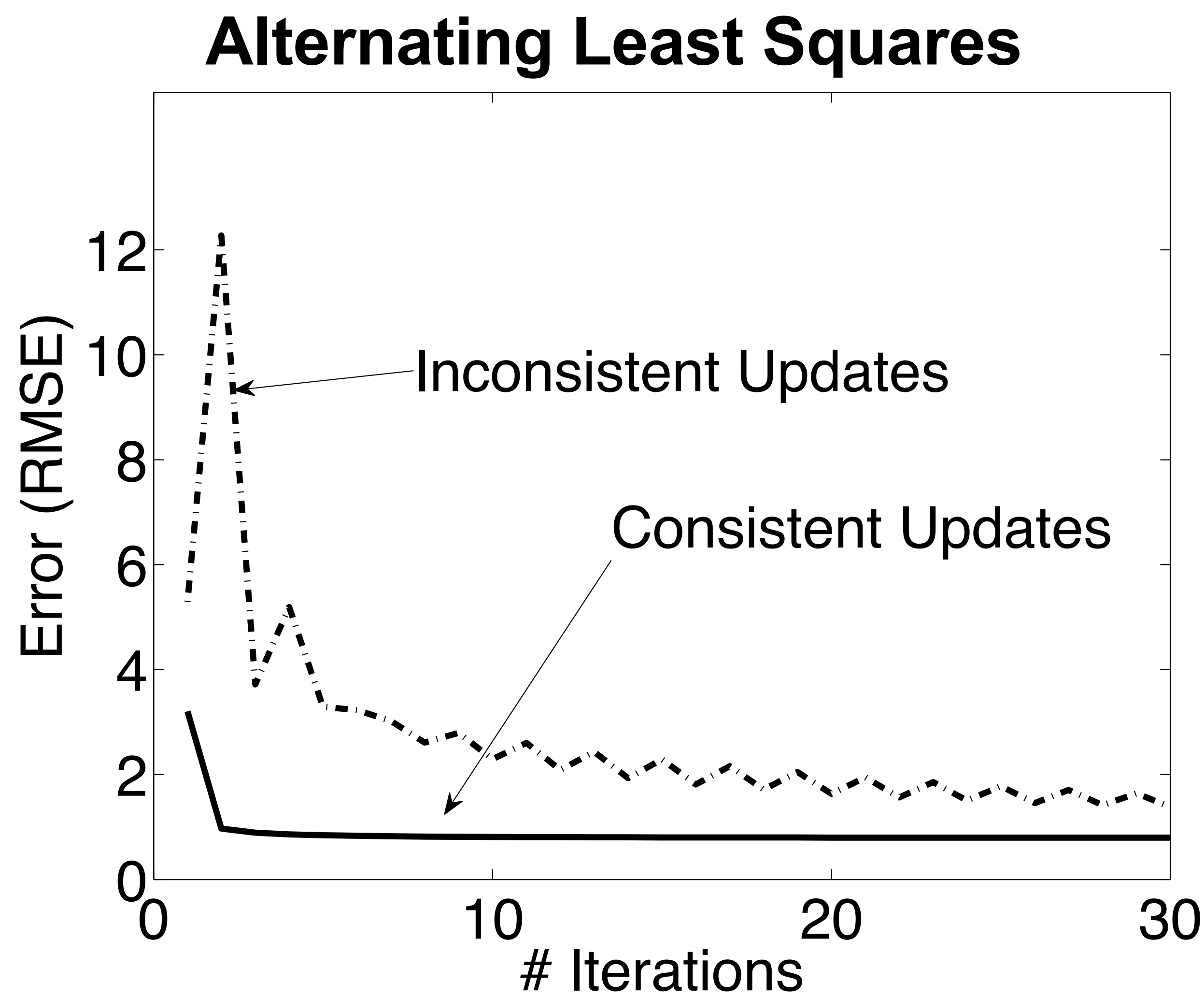
Ensuring Race-Free Code

- How much can computation **overlap**?



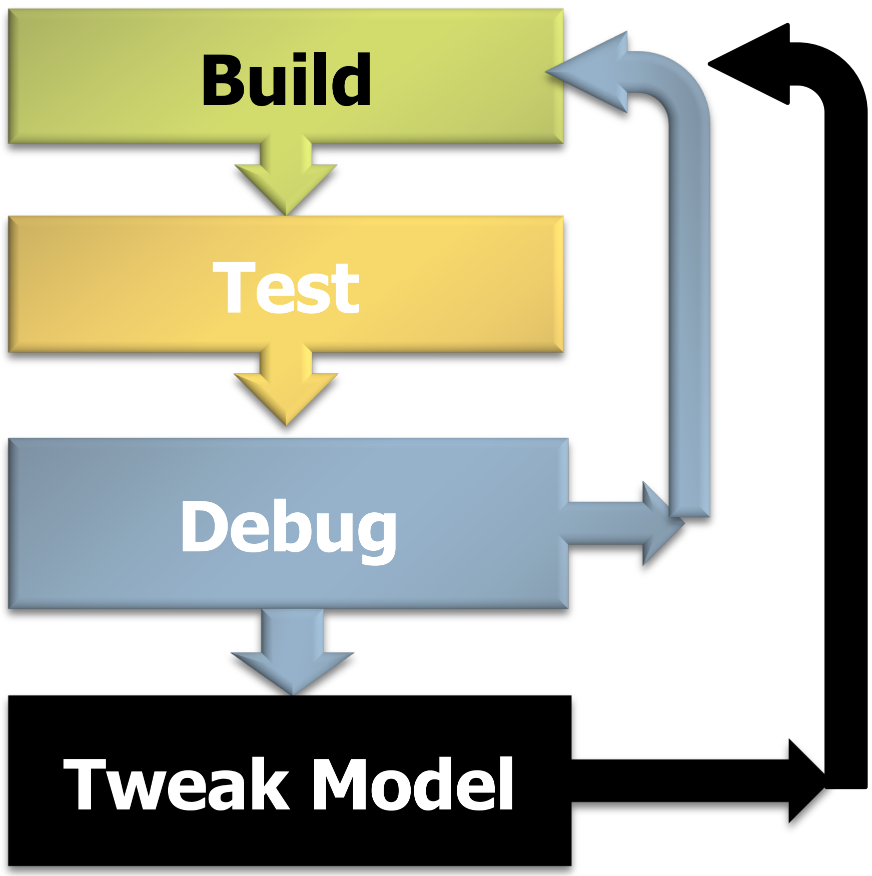
Importance of consistency

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



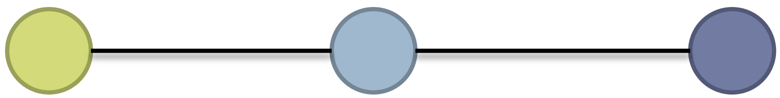
Importance of consistency

Machine learning algorithms require “model debugging”

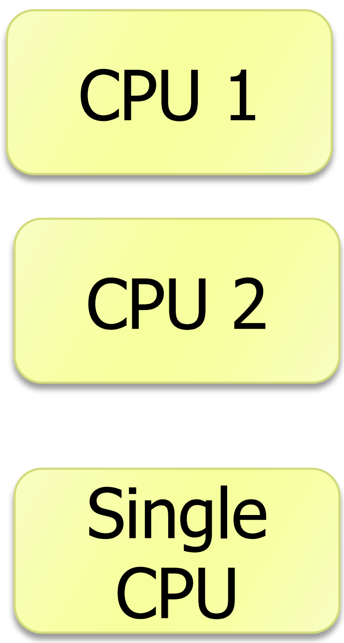


GraphLab Ensures Sequential Consistency

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

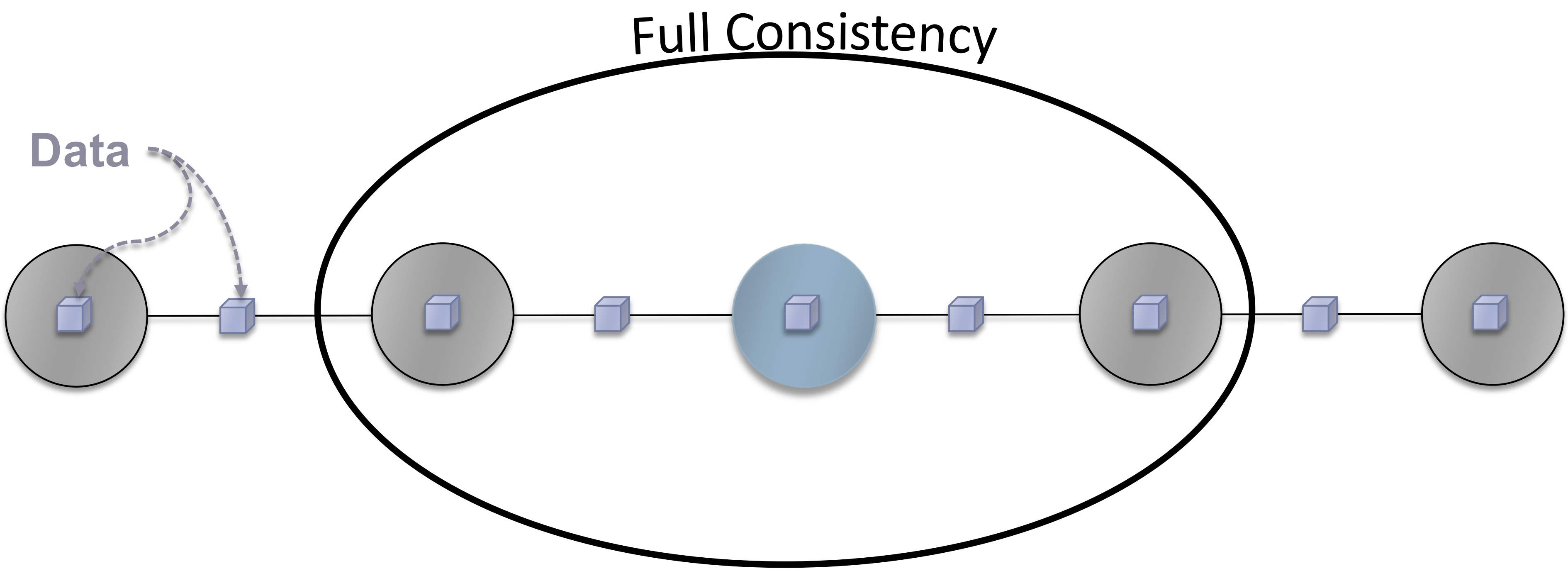


Parallel



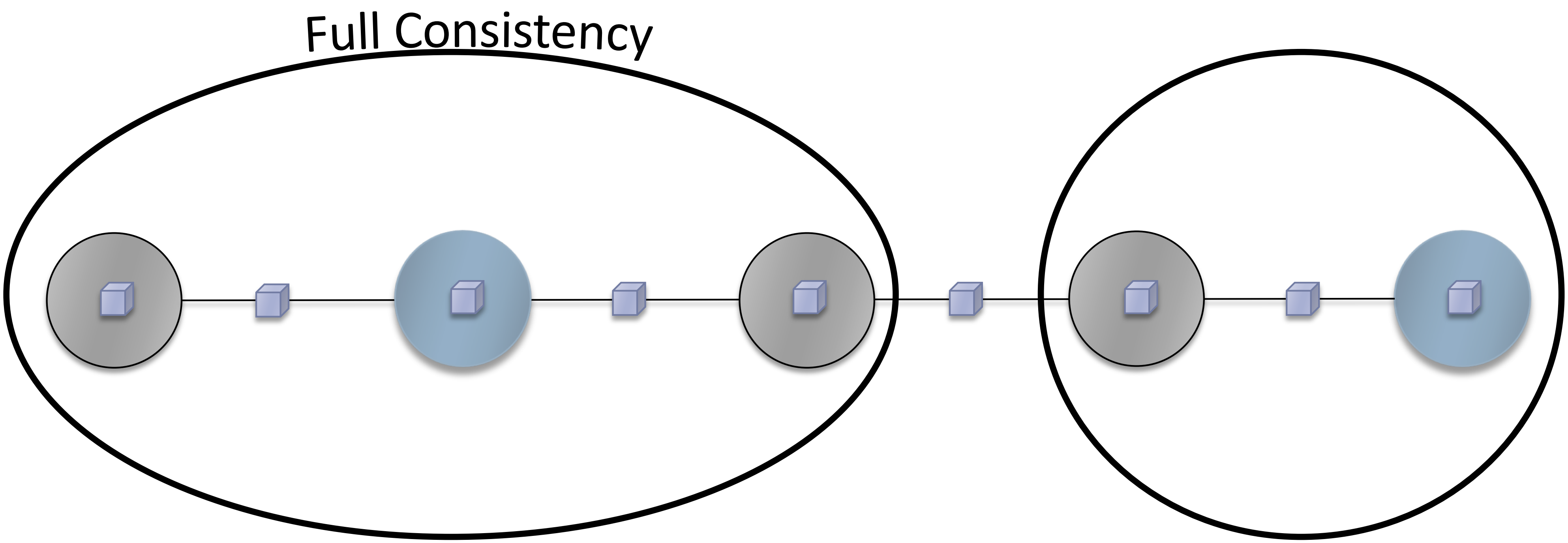
Sequential

Consistency Rules

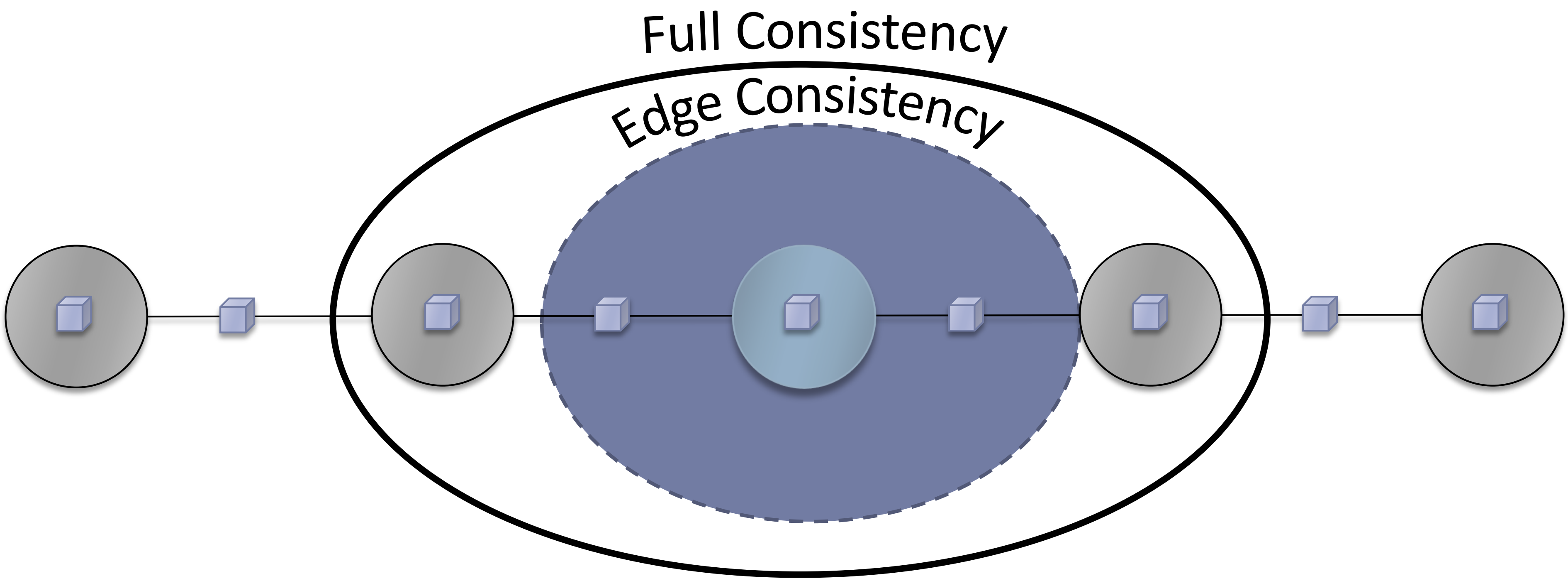


Guaranteed sequential consistency for all update functions

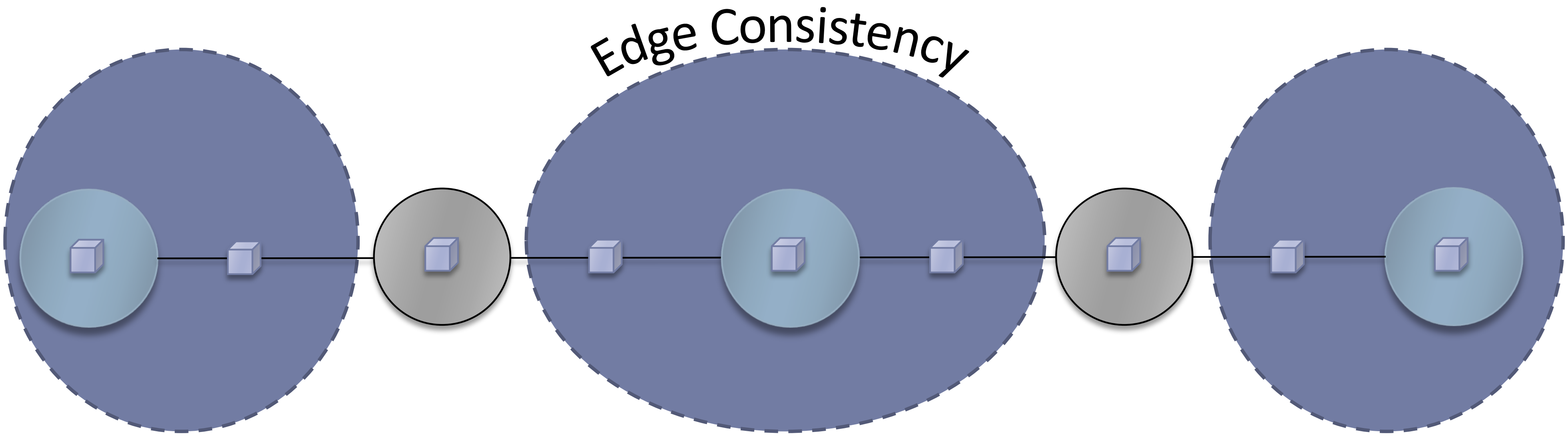
Full Consistency



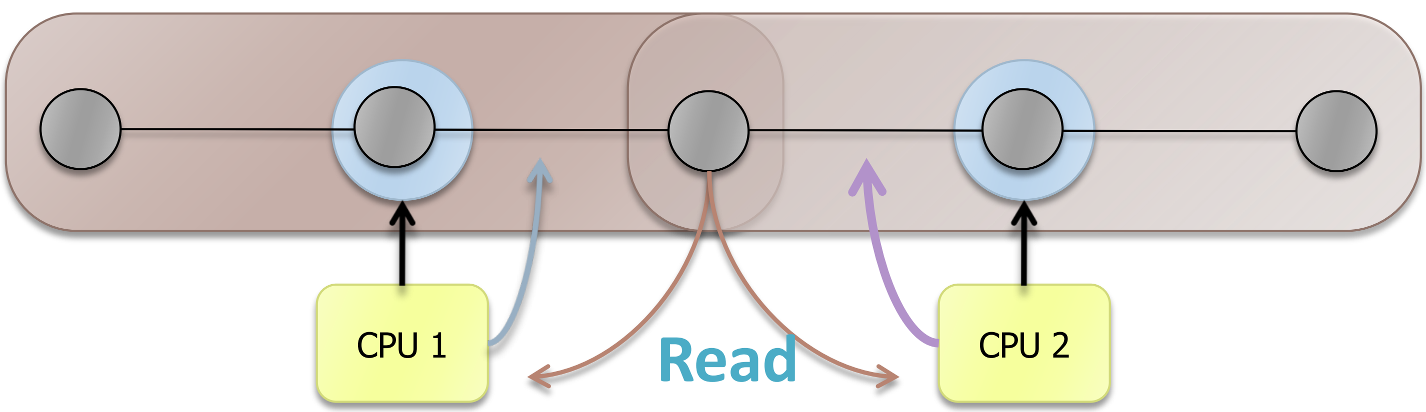
Obtaining More Parallelism



Edge Consistency

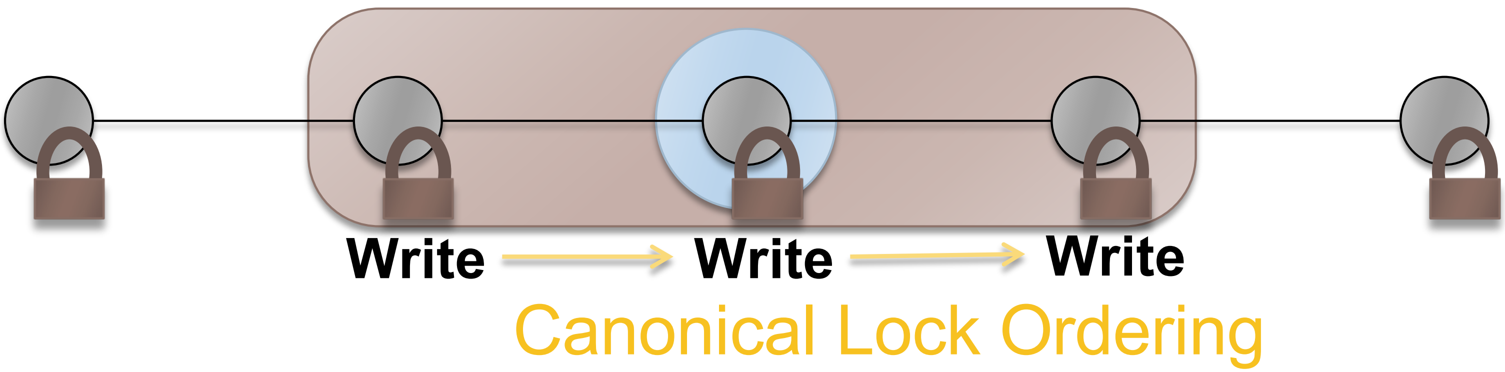


Safe

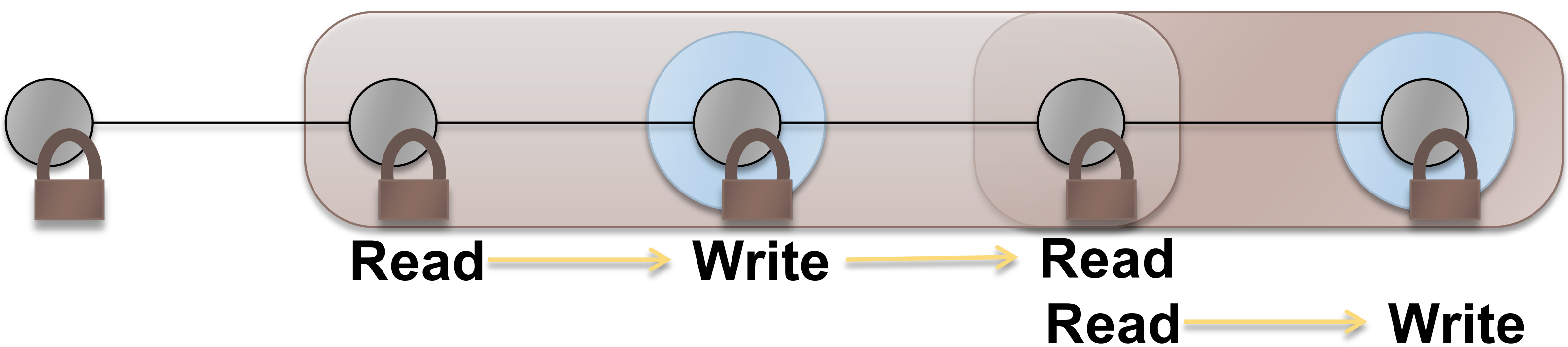


Consistency Through R/W Locks

- Read/Write locks:
  - Full Consistency



  - Edge Consistency



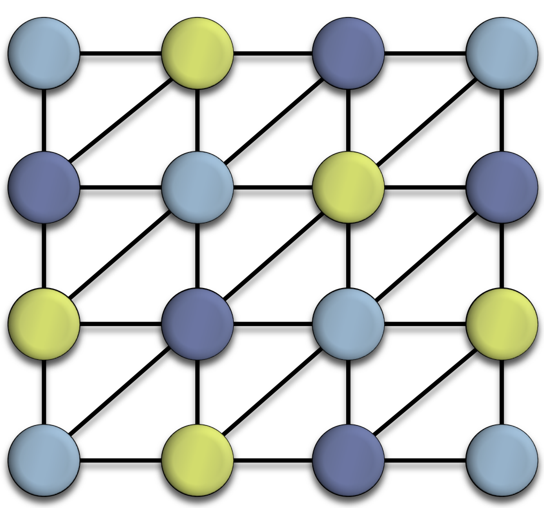
Consistency Through R/W Locks

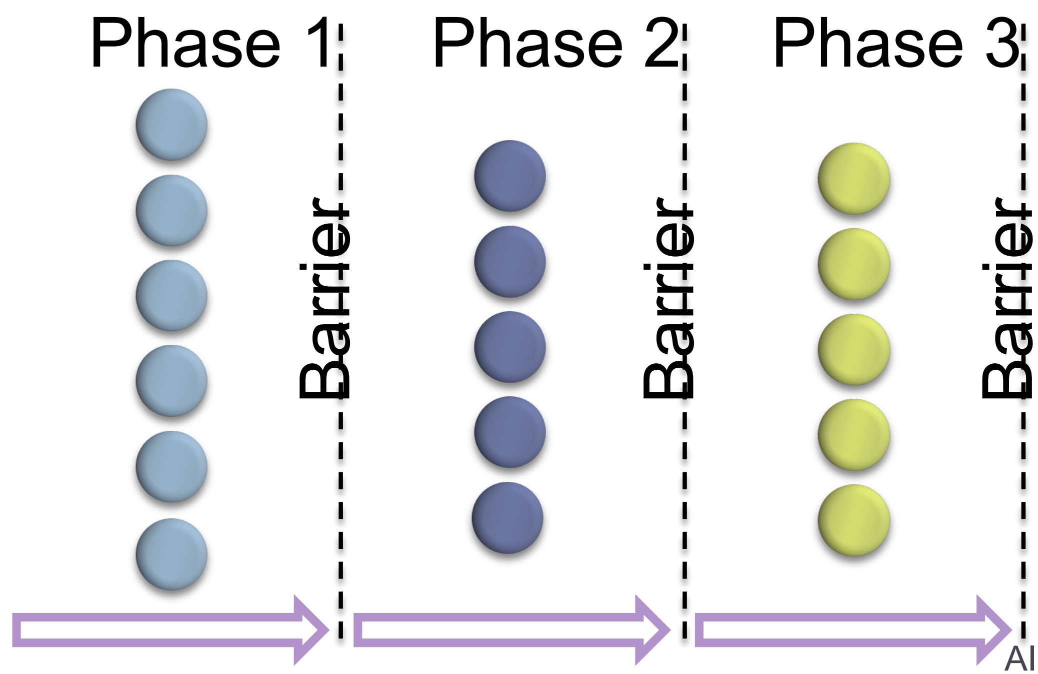
- Multicore Setting: Pthread R/W Locks
- Distributed Setting: *Distributed Locking*
  - Prefetch Locks and Data

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

Consistency through scheduling

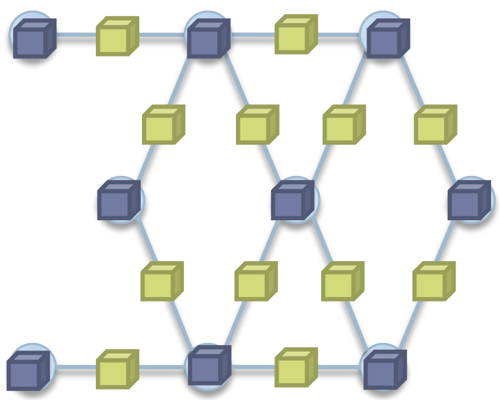
- Edge Consistency Model:
  - Two vertices can be **Updated** *simultaneously* if they do not share an edge.
- Graph Coloring:
  - Two vertices can be assigned the same color if they do not share an edge.



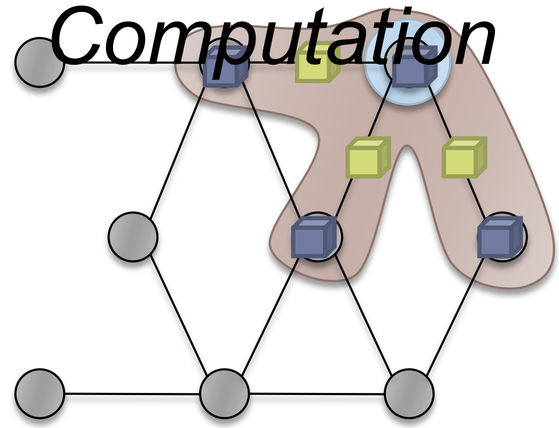


The GraphLab Framework

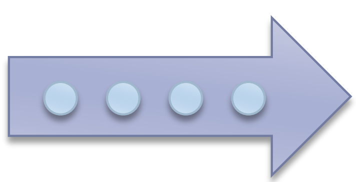
Graph Based
Data Representation



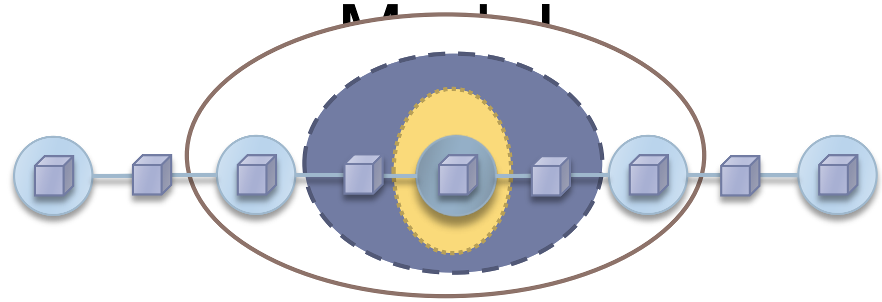
Update Functions
User Computation



Scheduler



Consistency

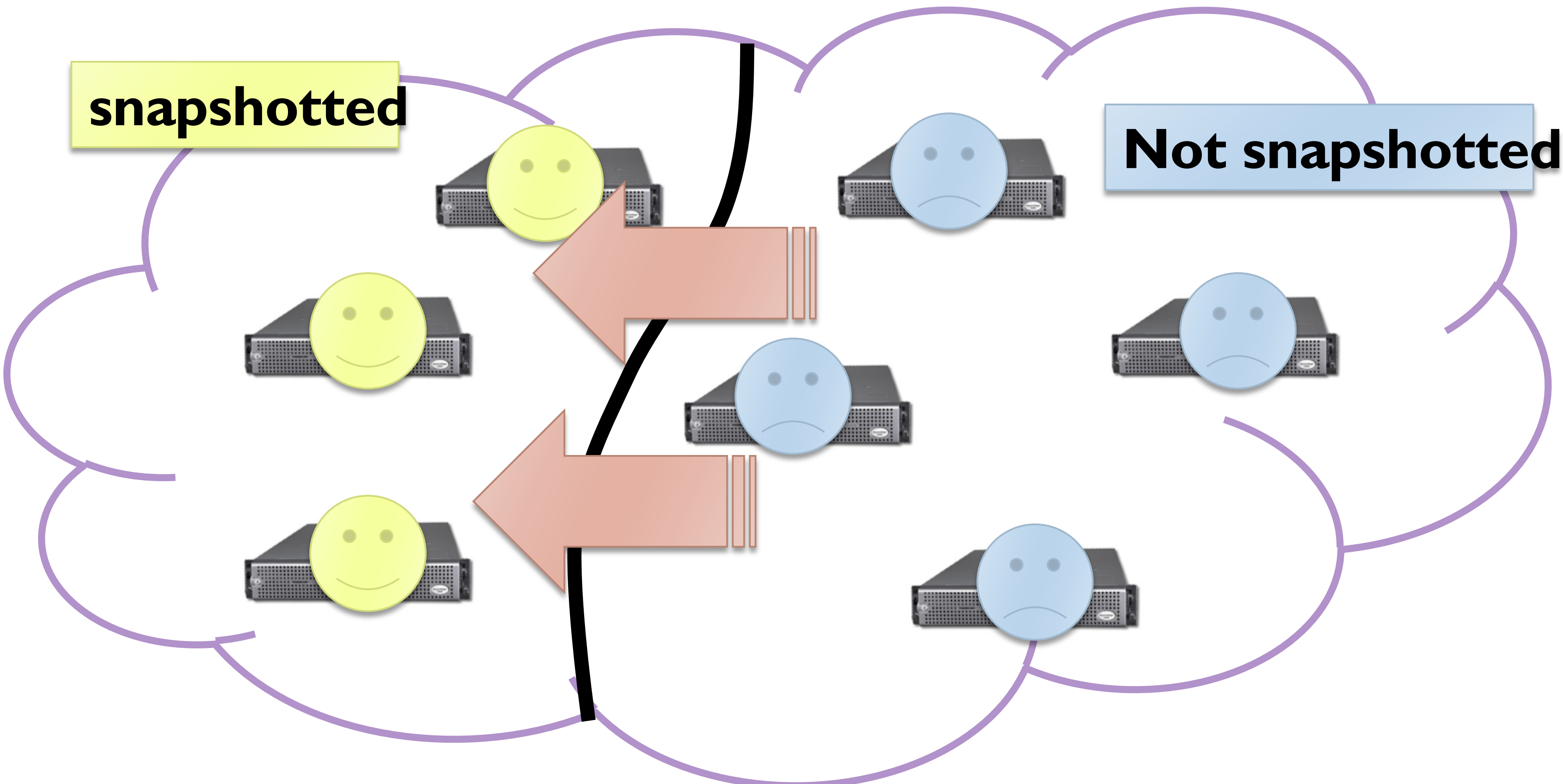


Algorithms Implemented

- 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
- …

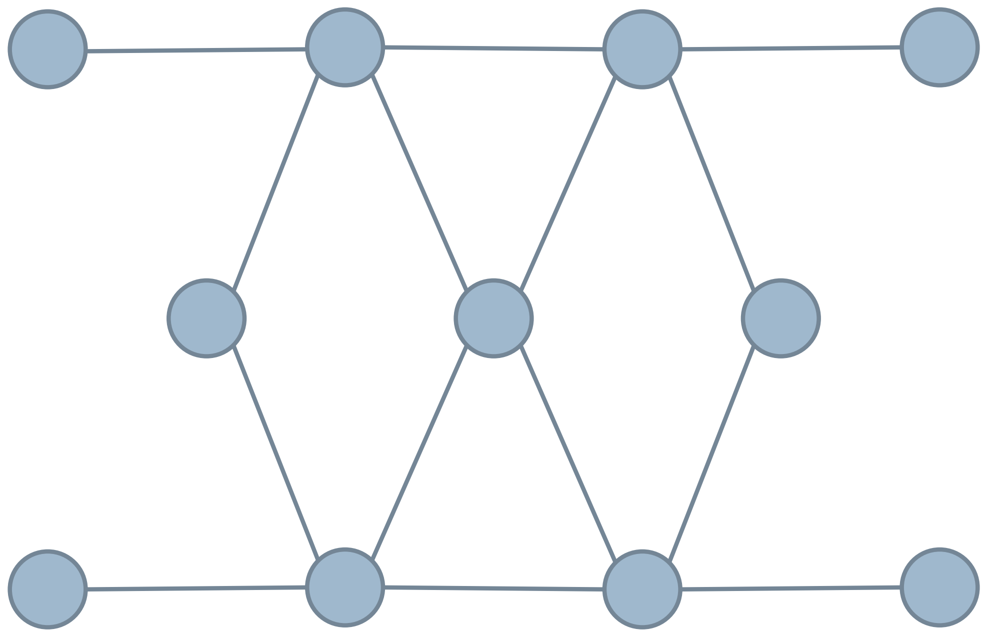
Fault-tolerance: Checkpointing

1985: Chandy-Lamport invented an asynchronous snapshotting algorithm for distributed systems.



Checkpointing

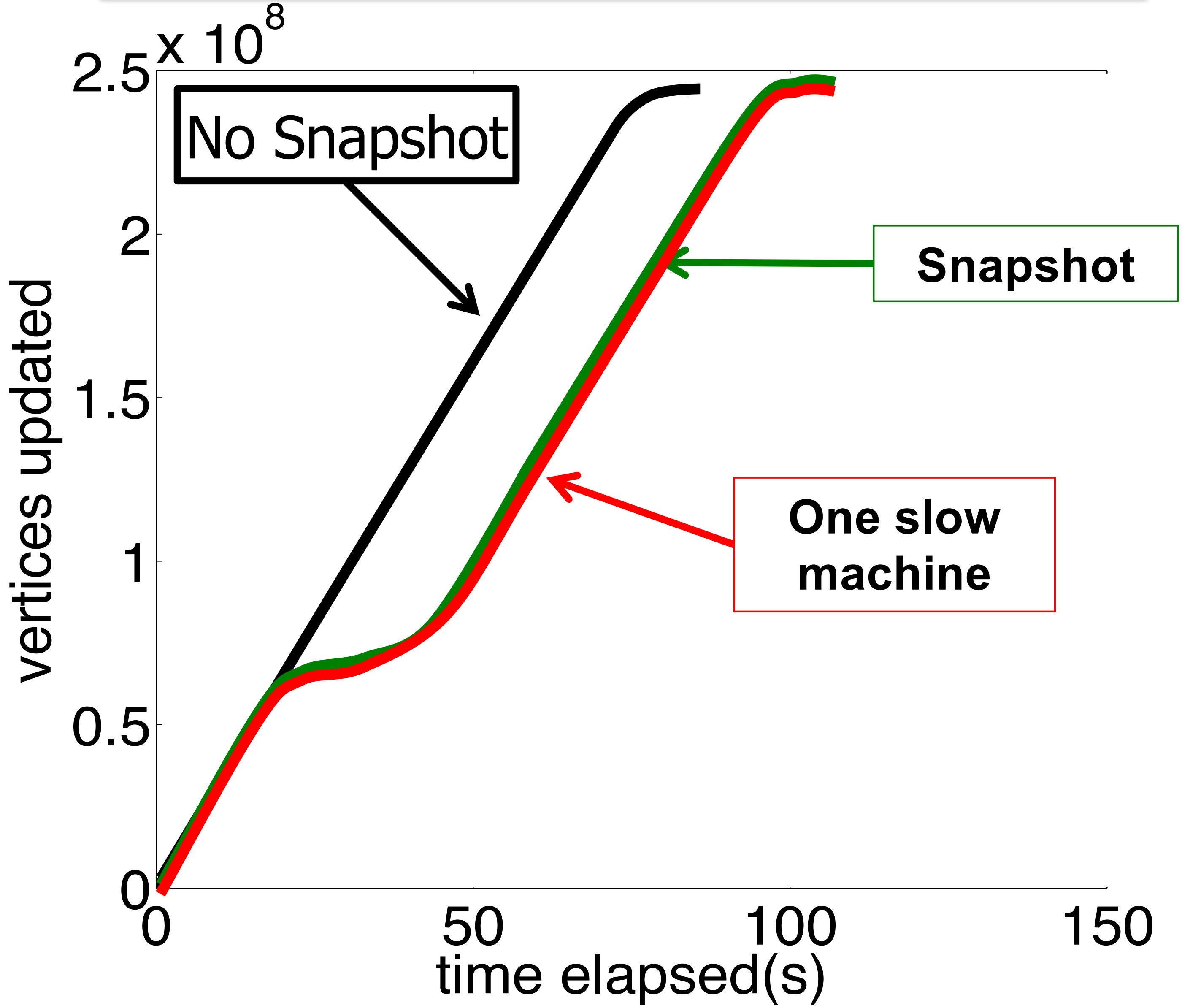
Fine Grained Chandy-Lamport.



Easily implemented within GraphLab as an Update Function!

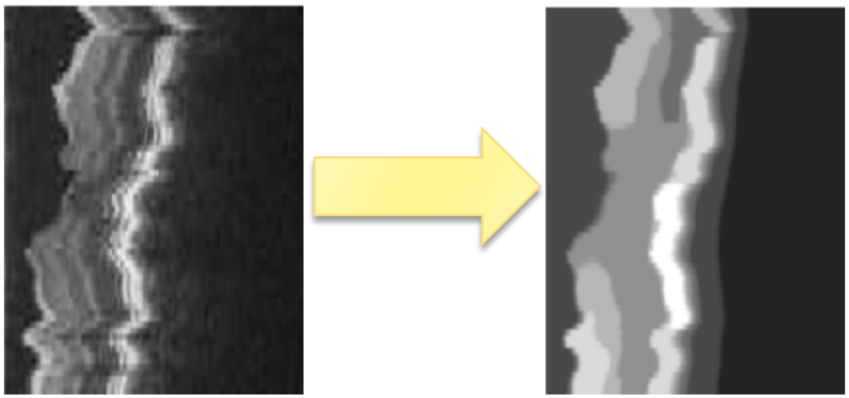
Async. Snapshot Performance

No penalty incurred by the slow machine!



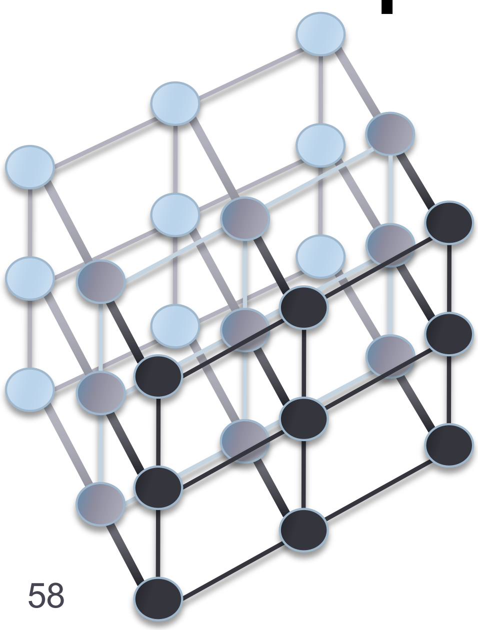
Loopy Belief Propagation

3D retinal image denoising



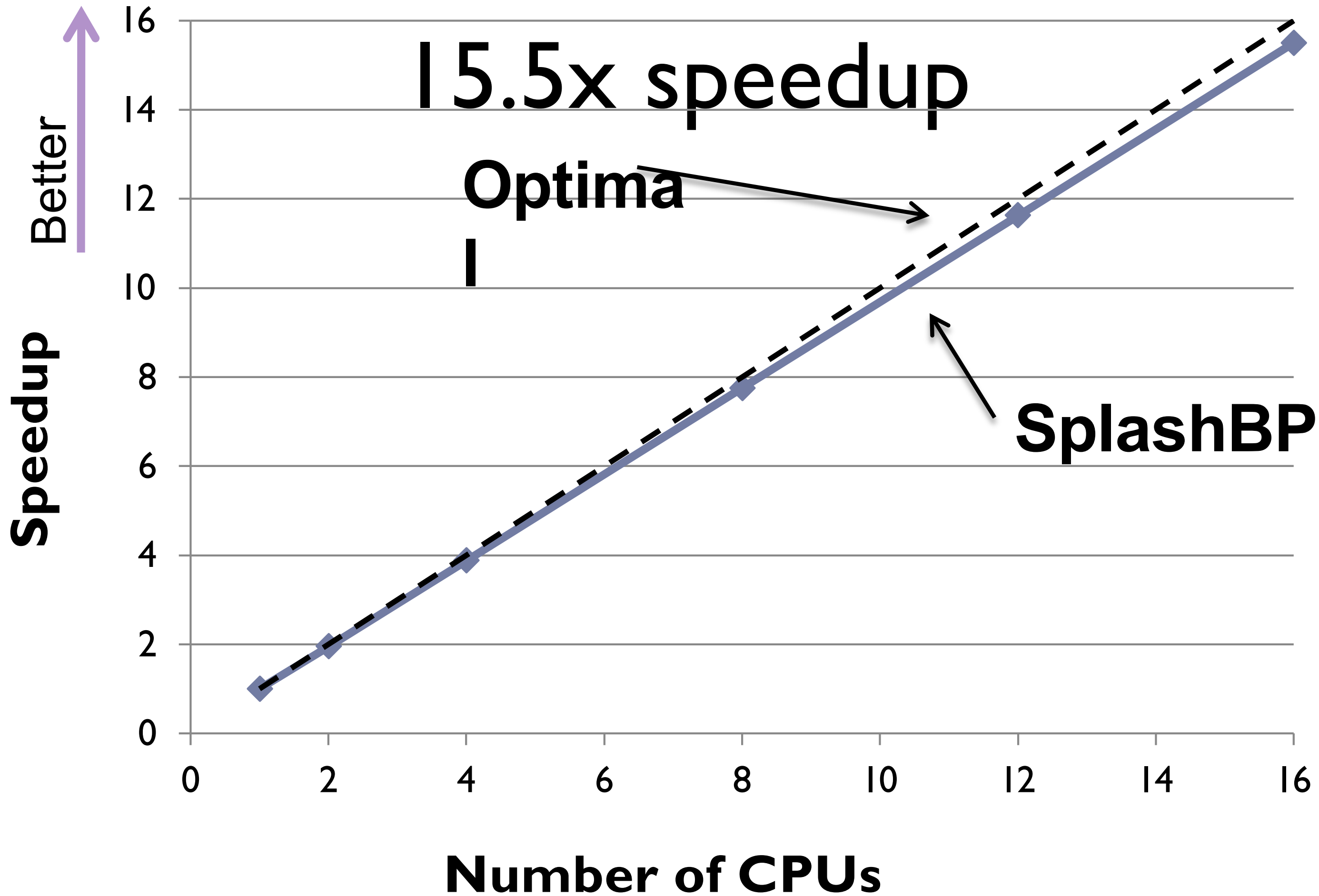
Vertices: 1 Million
Edges: 3 Million

Data Graph



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

Loopy Belief Propagation

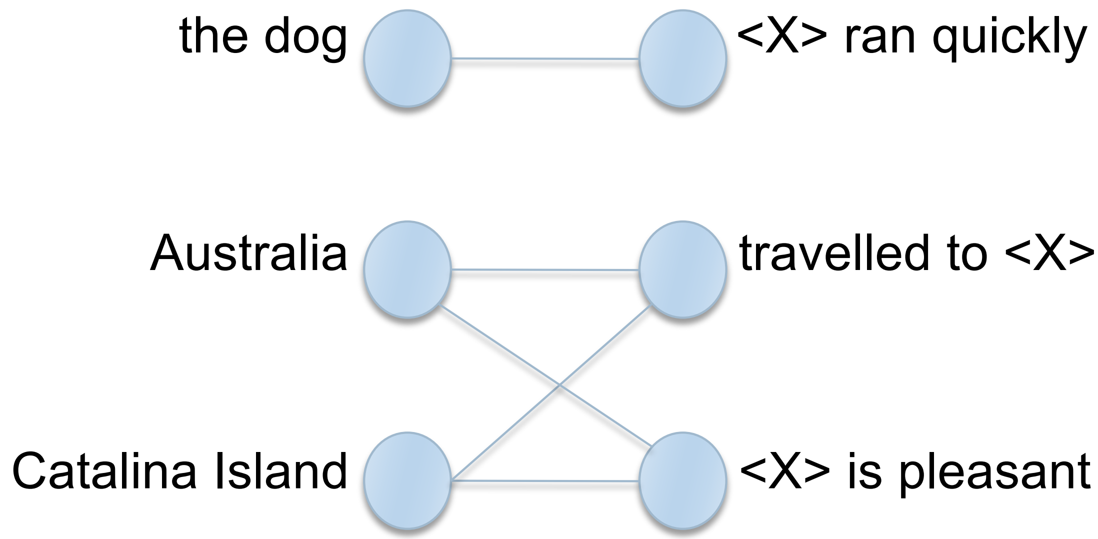


CoEM (Rosie Jones, 2005)

Named Entity Recognition Task

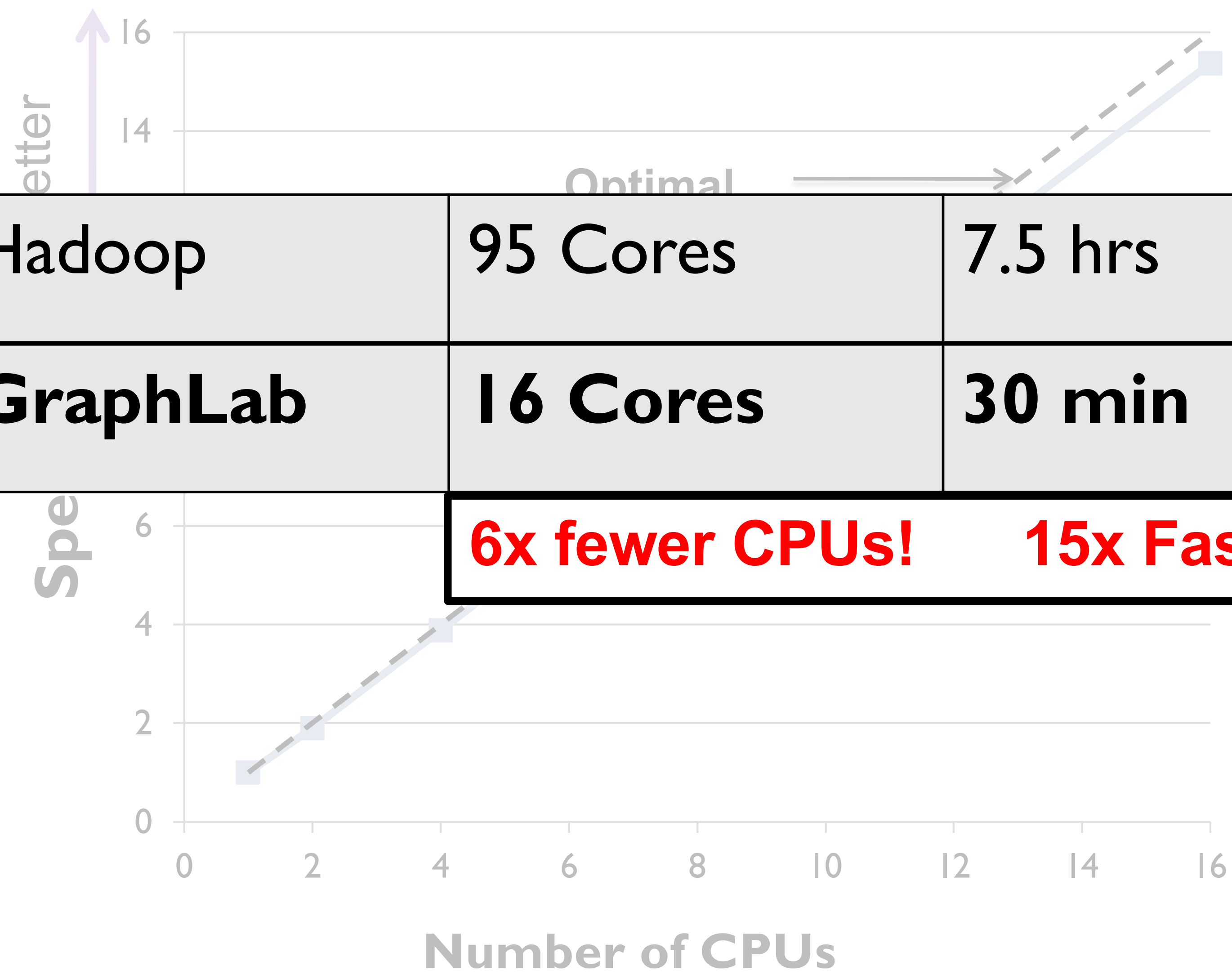
Is “Dog” an animal?
Is “Catalina” a place?

Vertices: 2 Million
Edges: 200 Million



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

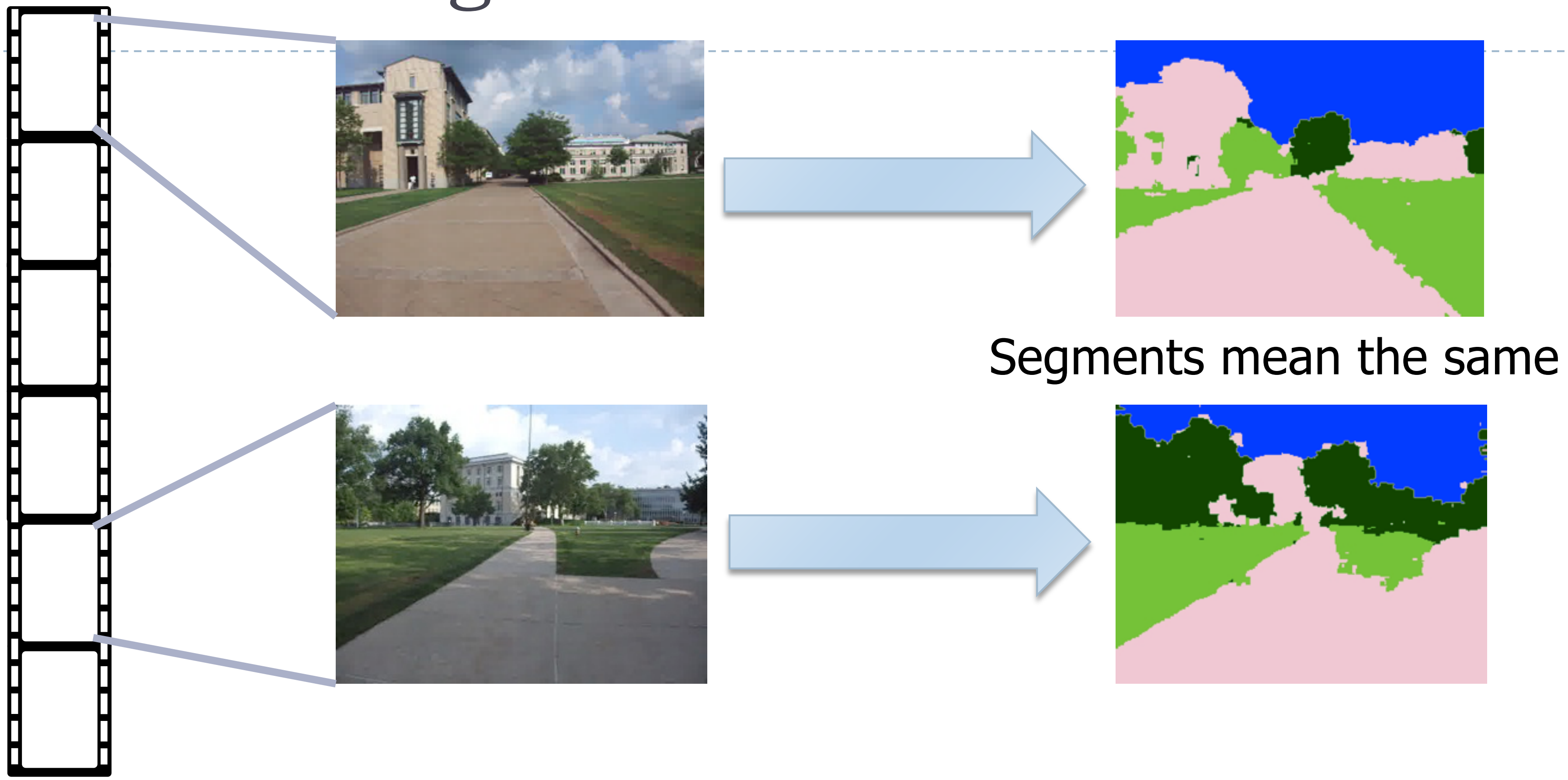
CoEM (Rosie Jones, 2005)



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

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

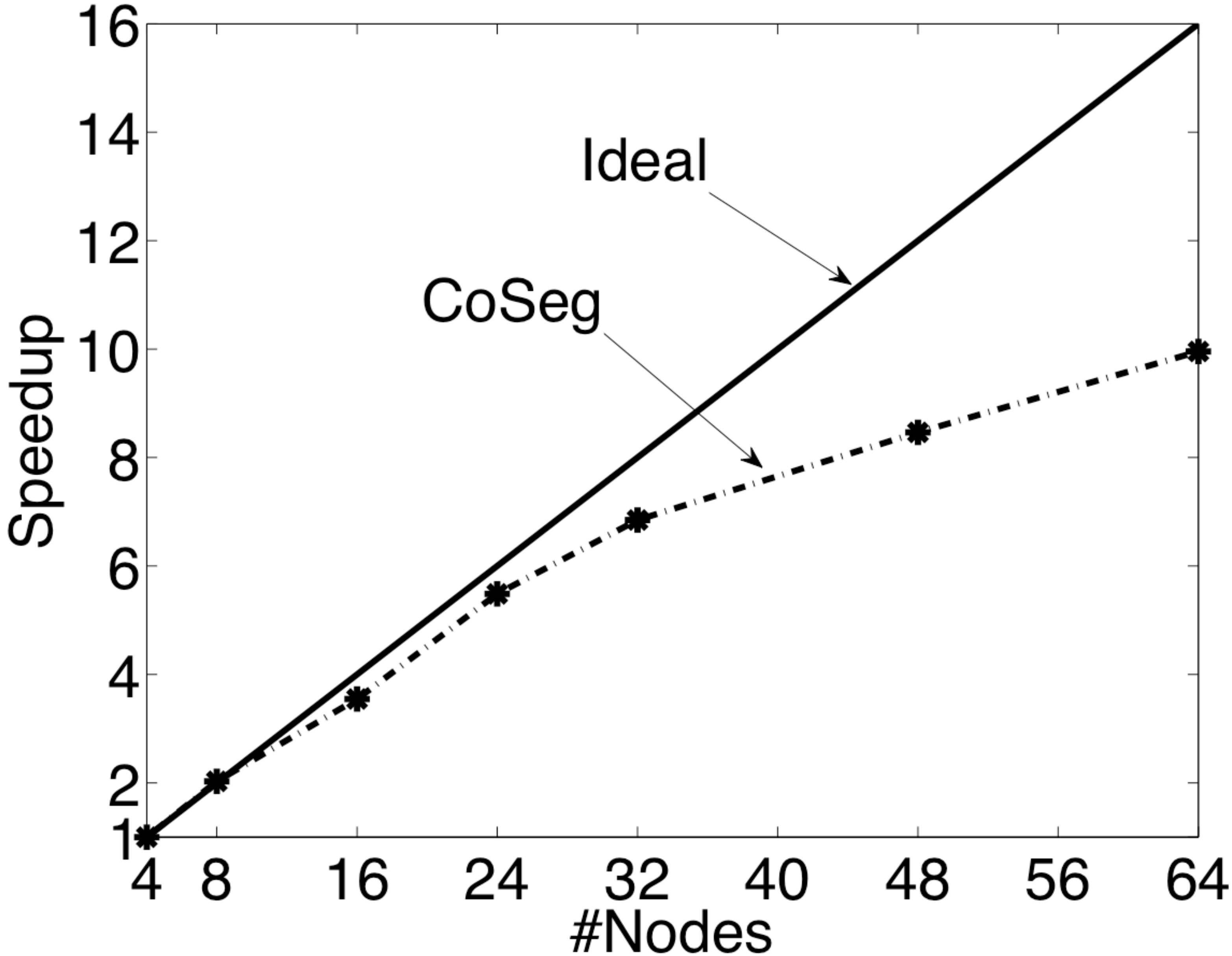
Video Cosegmentation



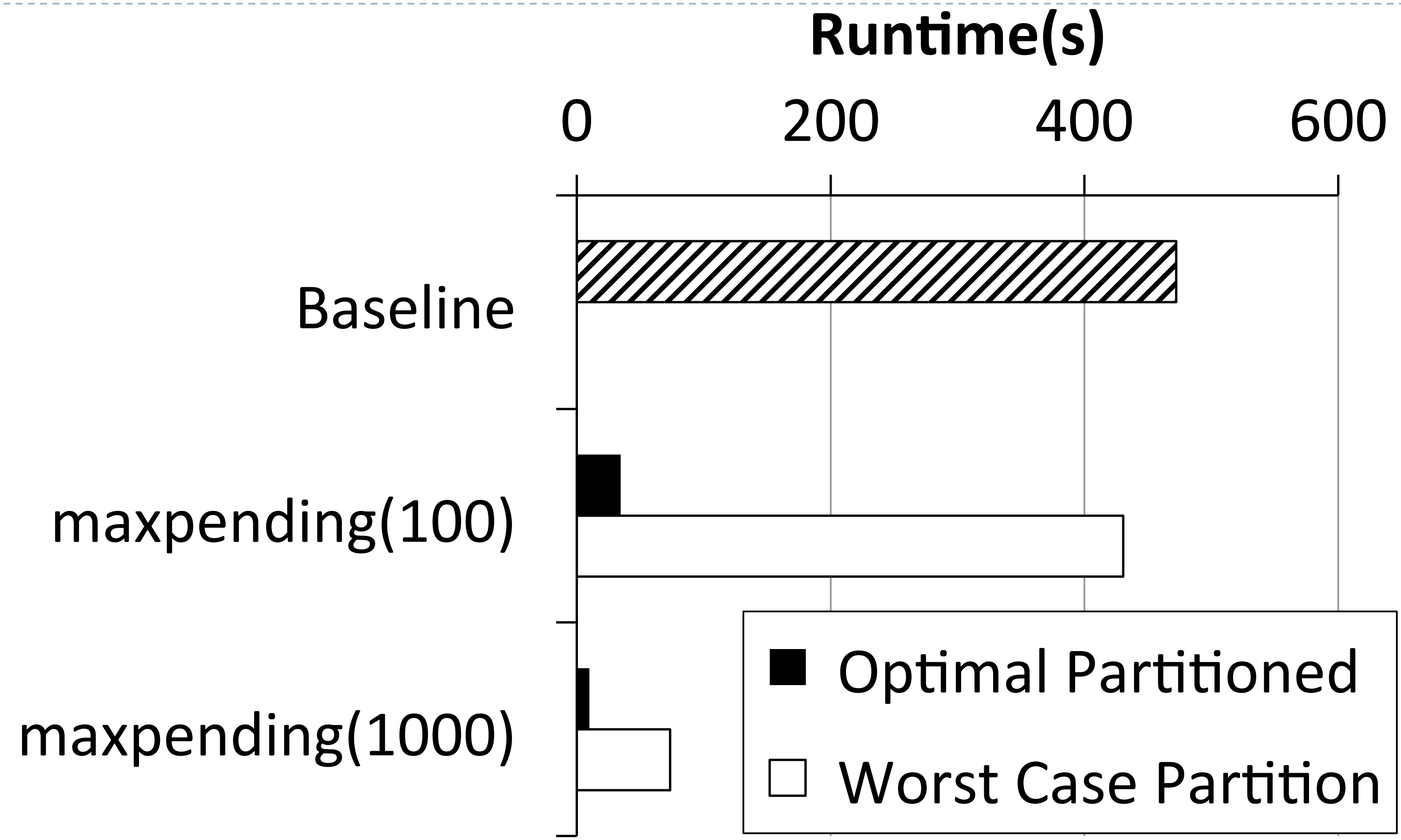
Gaussian EM clustering + BP on 3D grid

Model: 10.5 million nodes, 31 million edges

Video Coseg. Speedups



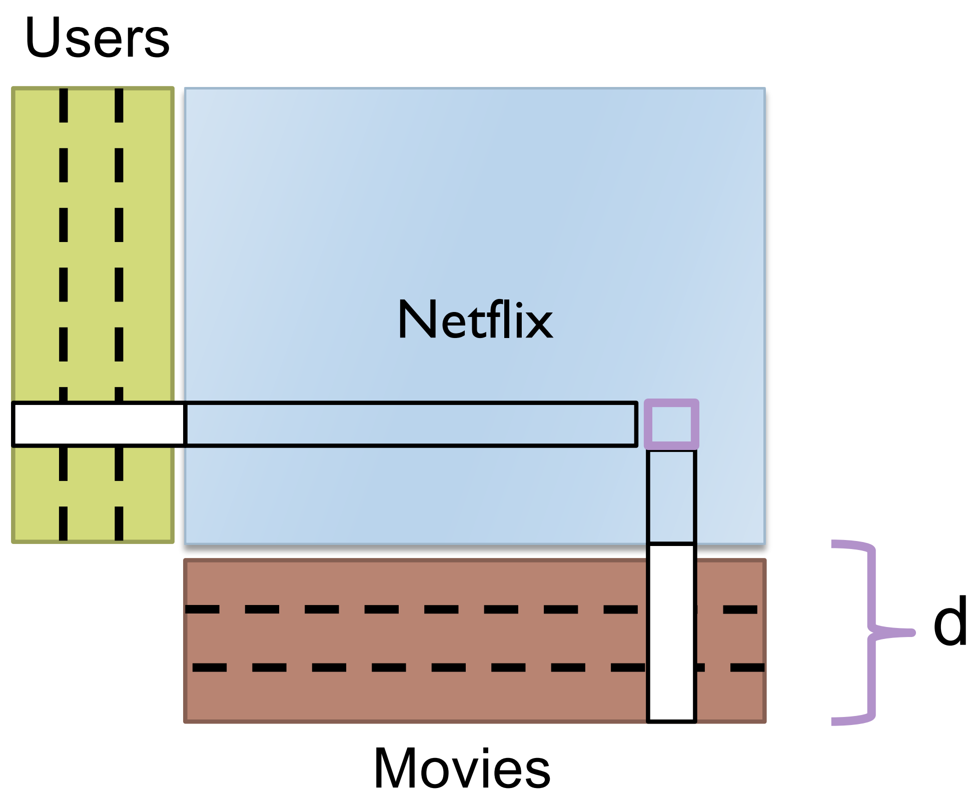
Prefetching Data & Locks



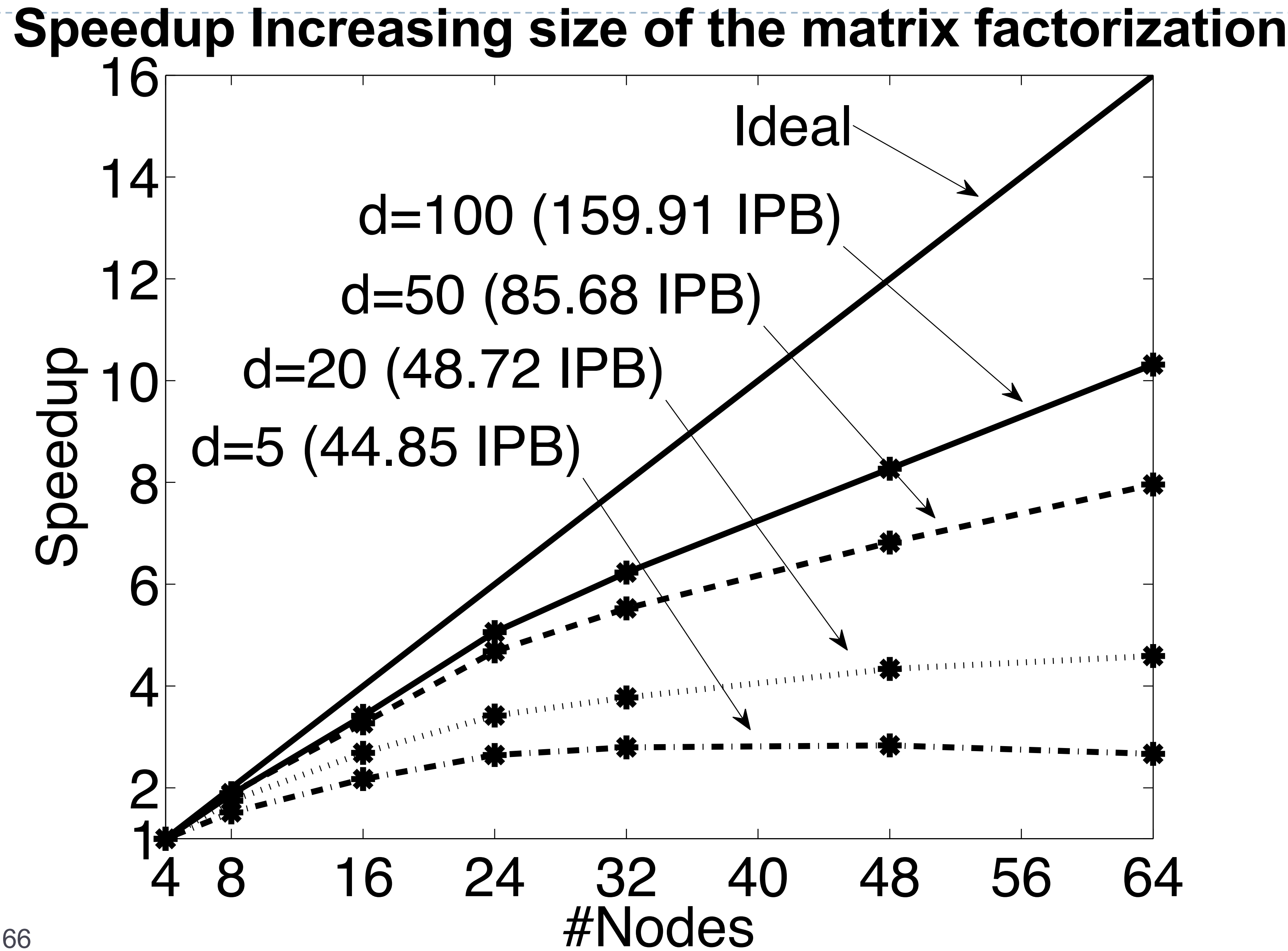
Matrix Factorization

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

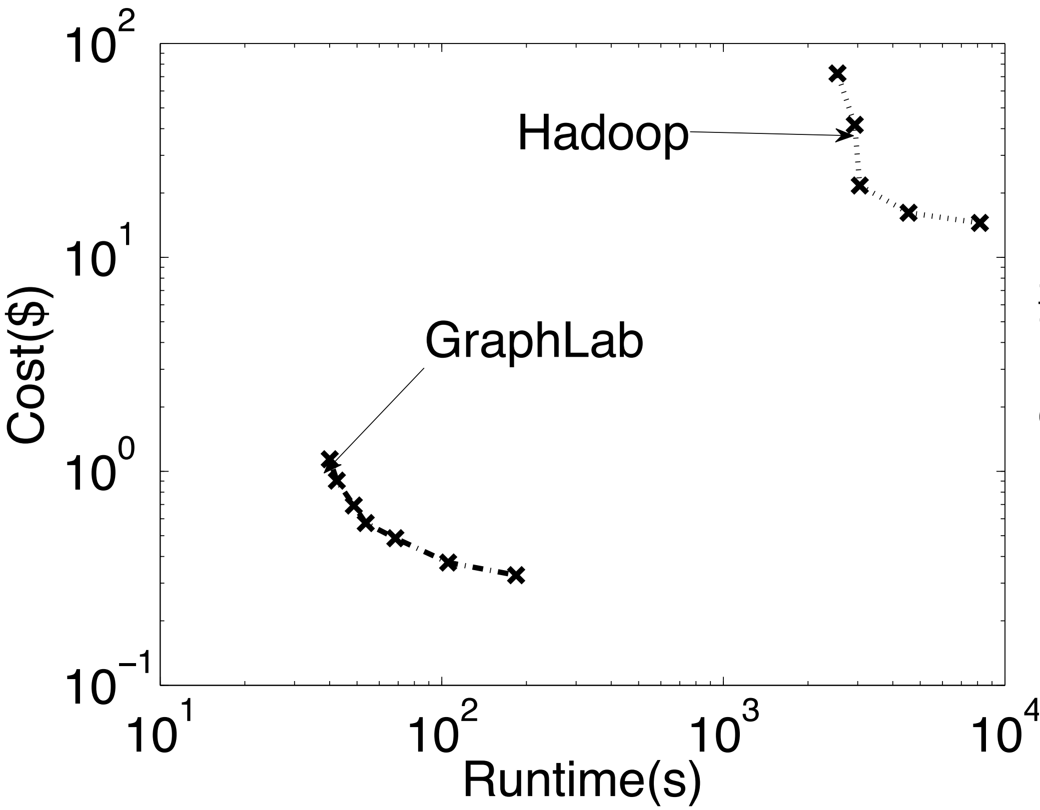
Model: 0.5 million nodes, 99 million edges

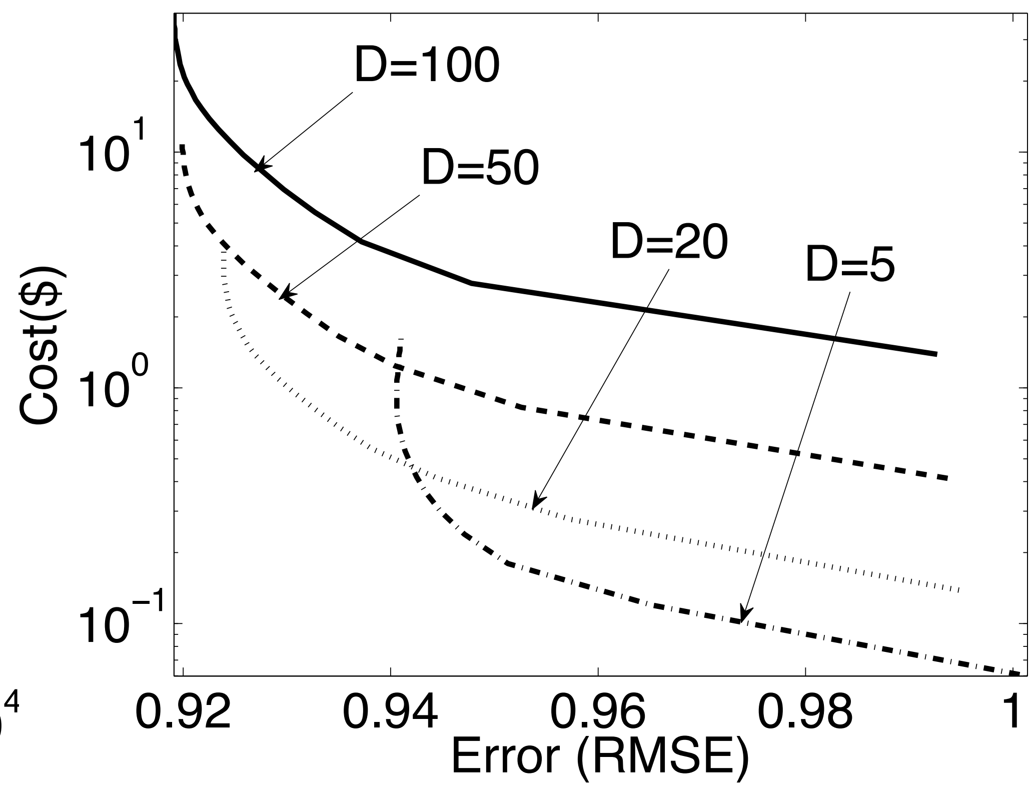


Netflix


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
16
14
12
10
8
6
4
2
1
4
8
16
24
32
40
48
56
64

The Cost of Hadoop





Summary

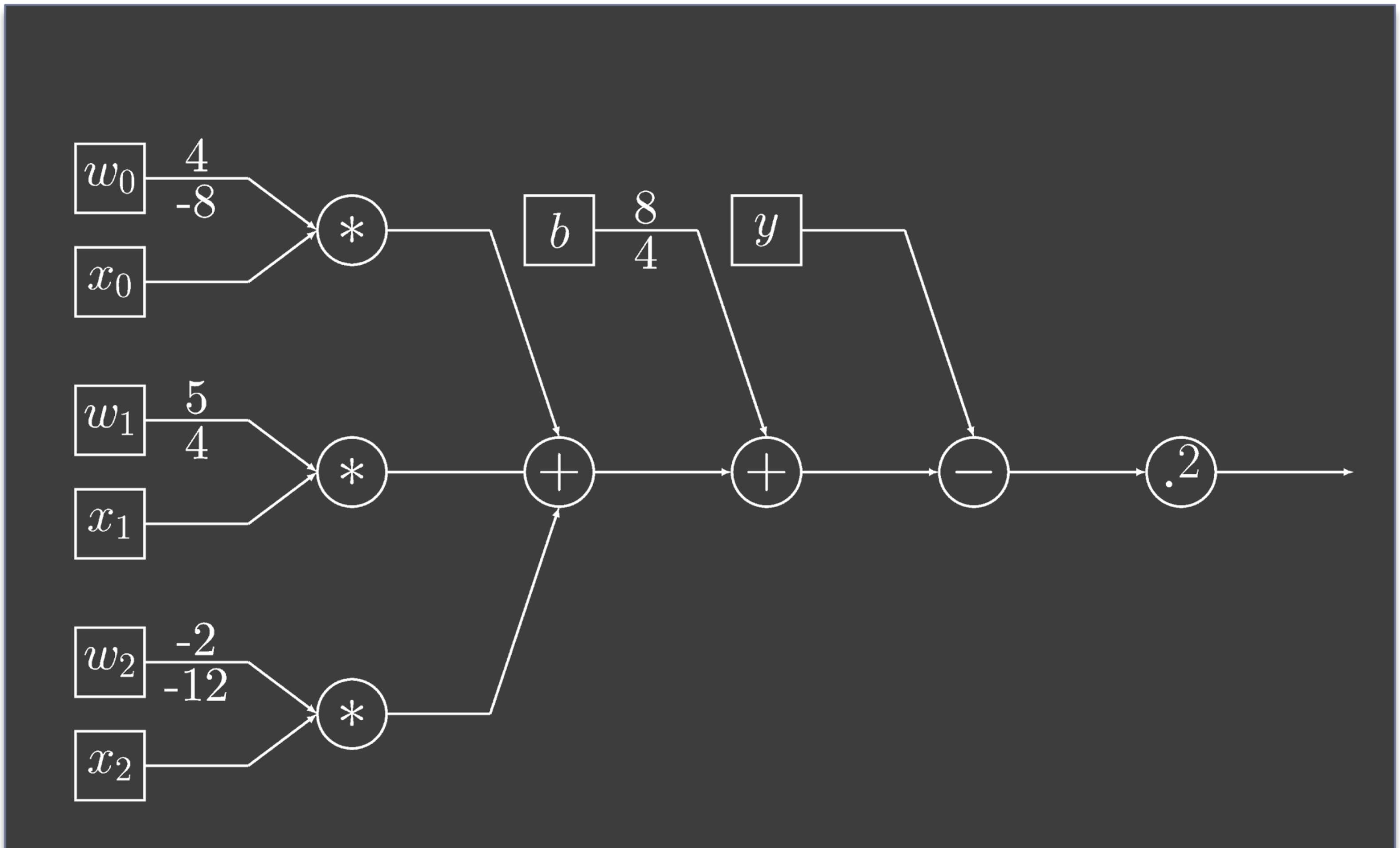
- An abstraction tailored to Machine Learning
  - Targets Graph-Parallel Algorithms
- Naturally expresses
  - Data/computational dependencies
  - Dynamic iterative computation
- Simplifies parallel algorithm design
- Automatically ensures data consistency
- Achieves state-of-the-art parallel performance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

3:TensorFlow

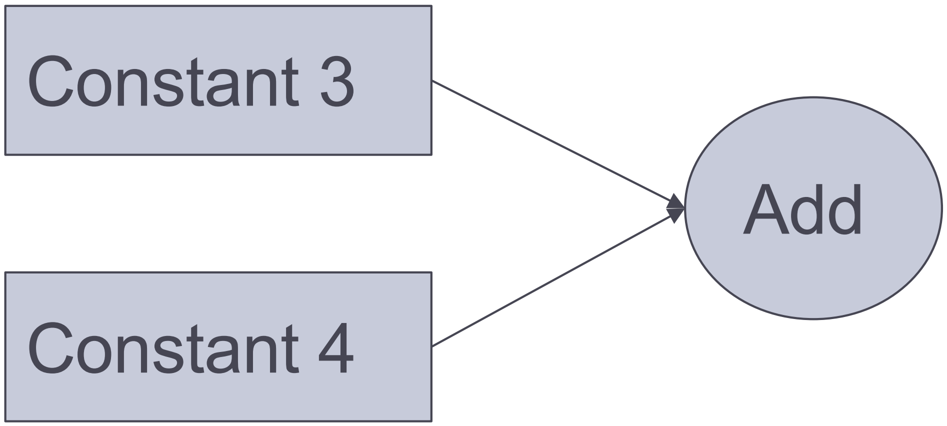
Context

- Huge need for high-productivity tools for building solutions to machine-learning problems
- Current infrastructures force people to reinvent the wheel
- Spark/RDD model illustrates power that better tools bring, but remains very low level: an RDD can deal with "anything" and is really just a small code applet
- TensorFlow builds off idea that ML applications are best understood by thinking about structured data: *tensors*

Python+Dataflow Programming



DataFlow Programming Example




```
node1 = tf.constant(3.0, dtype=tf.float32)

node2 = tf.constant(4.0, dtype=tf.float32)

node3 = tf.add(node1,node2)
```

Core TensorFlow Constructs

- **Dataflow Graphs**: entire computation
- **Data Nodes:** individual data or operations
- **Edges:** implicit dependencies between nodes;
  - TensorFlow transparently inserts the appropriate communication between distributed subcomputations.
- **Operations:** any computation
- **Constants:** single values (tensors)

Core TensorFlow constructs

- All nodes return **tensors**, or higher-dimensional matrices

- How a node computes is **indistinguishable to TensorFlow**

- **You are metaprogramming**. No computation occurs yet!

Running code

```
tf.Session().run(node3) #returns 7
```

Placeholders (inputs) and how to use them

```
node1 = tf.placeholder(tf.float32)
node2 = tf.placeholder(tf.float32)
node3 = tf.add(node1,node2)
tf.Session().run(node3, {node1 : 3, node2 : 4})
```

Variables (mutable state)

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b #Operator Overloading!
init = tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init)
  sess.run(linear_model)
```

Specifying devices using with blocks

```
with tf.device("/cpu:0"):
  W = tf.Variable(...)
  V = tf.Variable(...)
with tf.device("/gpu:0")
  output = tf.some_fancy_math(input, W) + b
```

CPU:0

GPU:0

Specifying devices using with blocks

```python
with tf.device("/task:0/cpu:0"):
  W = tf.Variable(...)
  V = tf.Variable(...)
with tf.device("/task:1/gpu:0")
  output = tf.some_fancy_math(input, W) + b
```

task:0/CPU:0

task:1/GPU:0

Starting remote TensorFlow nodes

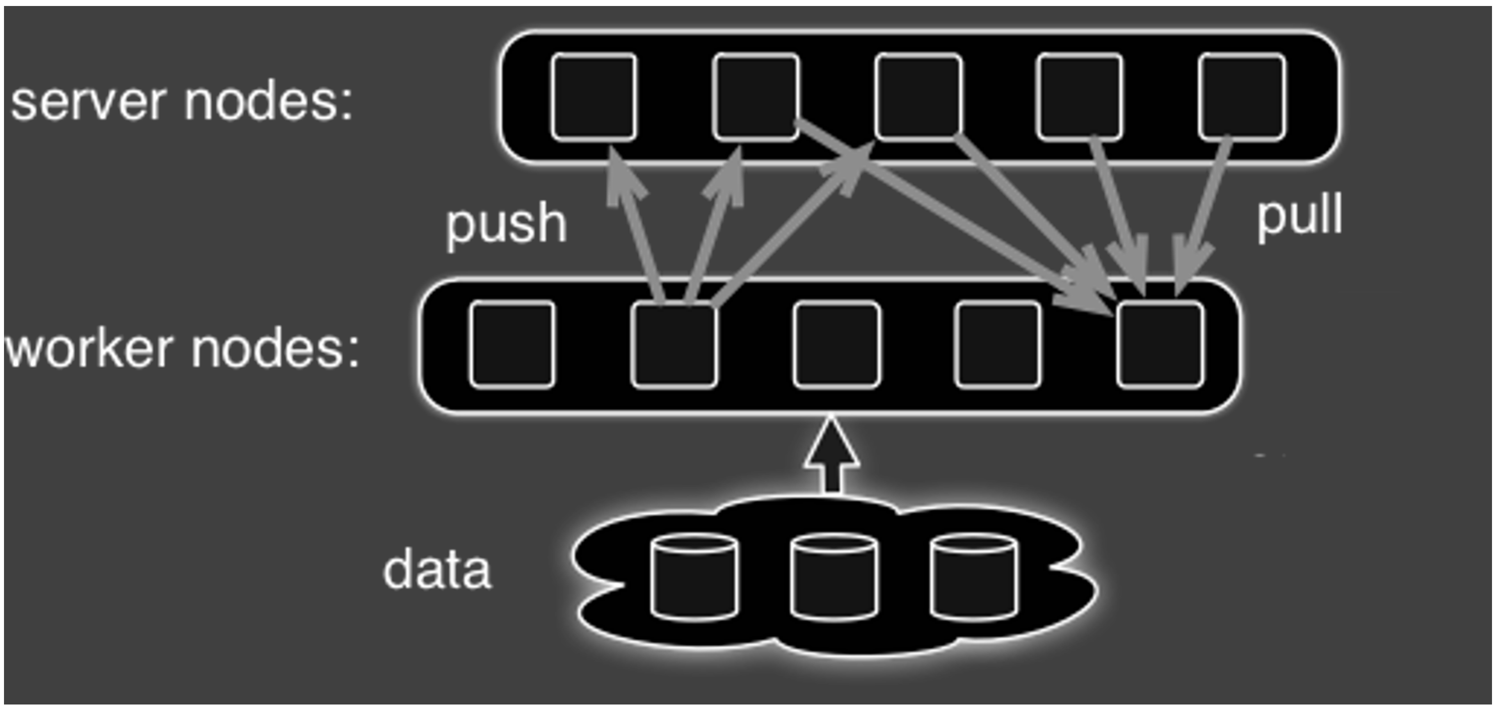
```
#all the machines mentioned in the dataflow graph
cluster = tf.train.ClusterSpec([ip1:p1,ip2:p2,...])
#task_index is set to my "id"
server = tf.train.Server(cluster,task_index=0)
#begin listening
server.join()
```

Server actions

Sessions run code on **subgraphs**; can parallelize by splitting input

```
with tf.device("/task:n"):
  half_input = tf.Variable(input[:len(input)/2])
  work = tf.CoolFeature(half_input)
cluster = tf.train.ClusterSpec(...)
server = tf.train.Server(cluster, task_index=n)
with tf.Session(server.target) as sess:
  sess.run(work)
```

Suggested Design: parameter server



Parameter server focus :

- Hold Mutable state
- Apply updates
- Maintain availability
- Group Name: **ps**

Worker focus:

- Perform "active" actions
- Checkpoint state to FS
- Mostly stateless; can be restarted
- Group name: **`worker`**

Parameter server example

```
with tf.device("/jobs:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0,num_workers,input)
outputs = []
for i in range (num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" % i):
    outputs.append(tf.matmul(input[i],W) + b)
```

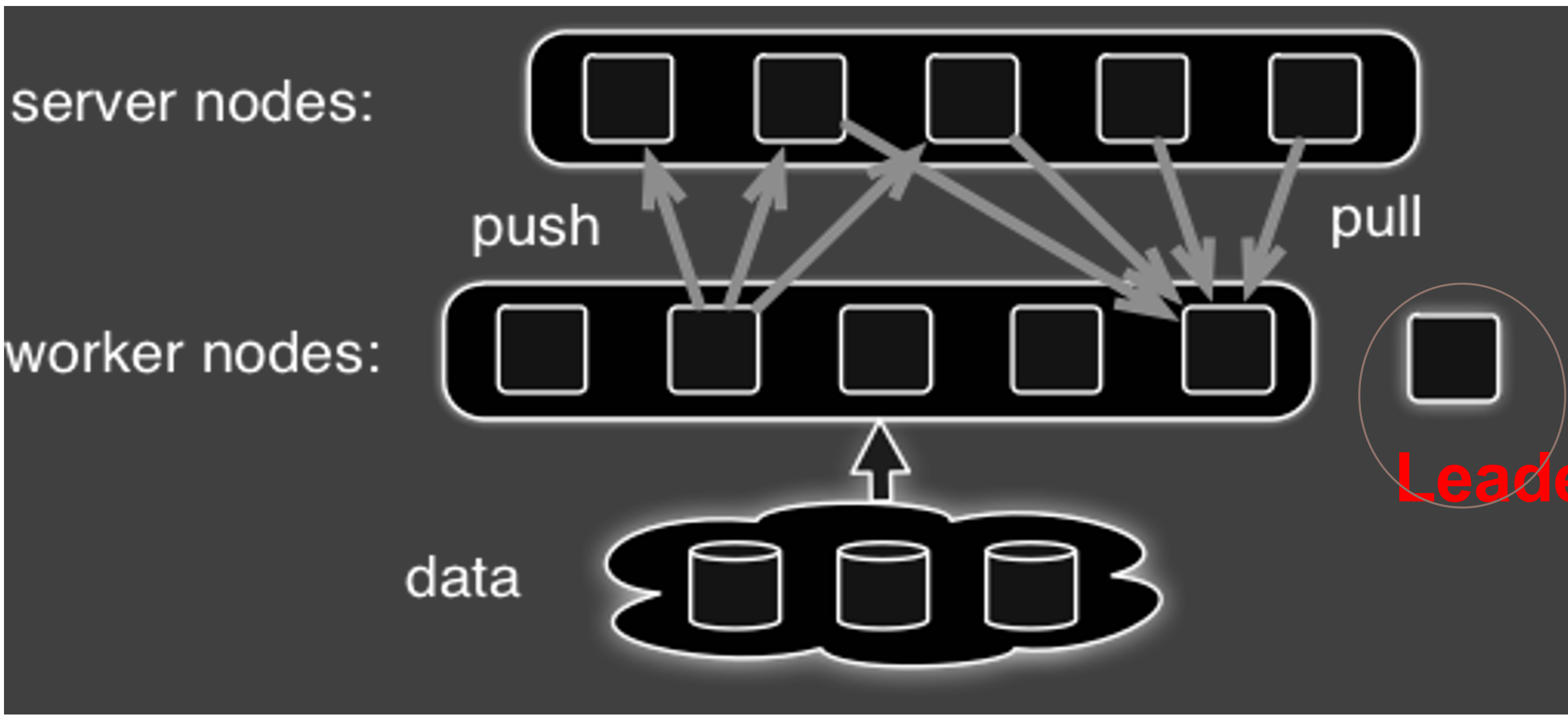
And that's it!

- For most TF applications, you don't need to know more.

- But this is because most TF runs are just a few steps, like a Spark job that performs a few actions on some RDDs

- What about using TF for long-term jobs that continuously process input, like events from a smart highway?
  - The model still makes sense, but now fault-tolerance would be an issue
  - Control of concurrency / consistency could begin to matter, too.

Adding Fault tolerance

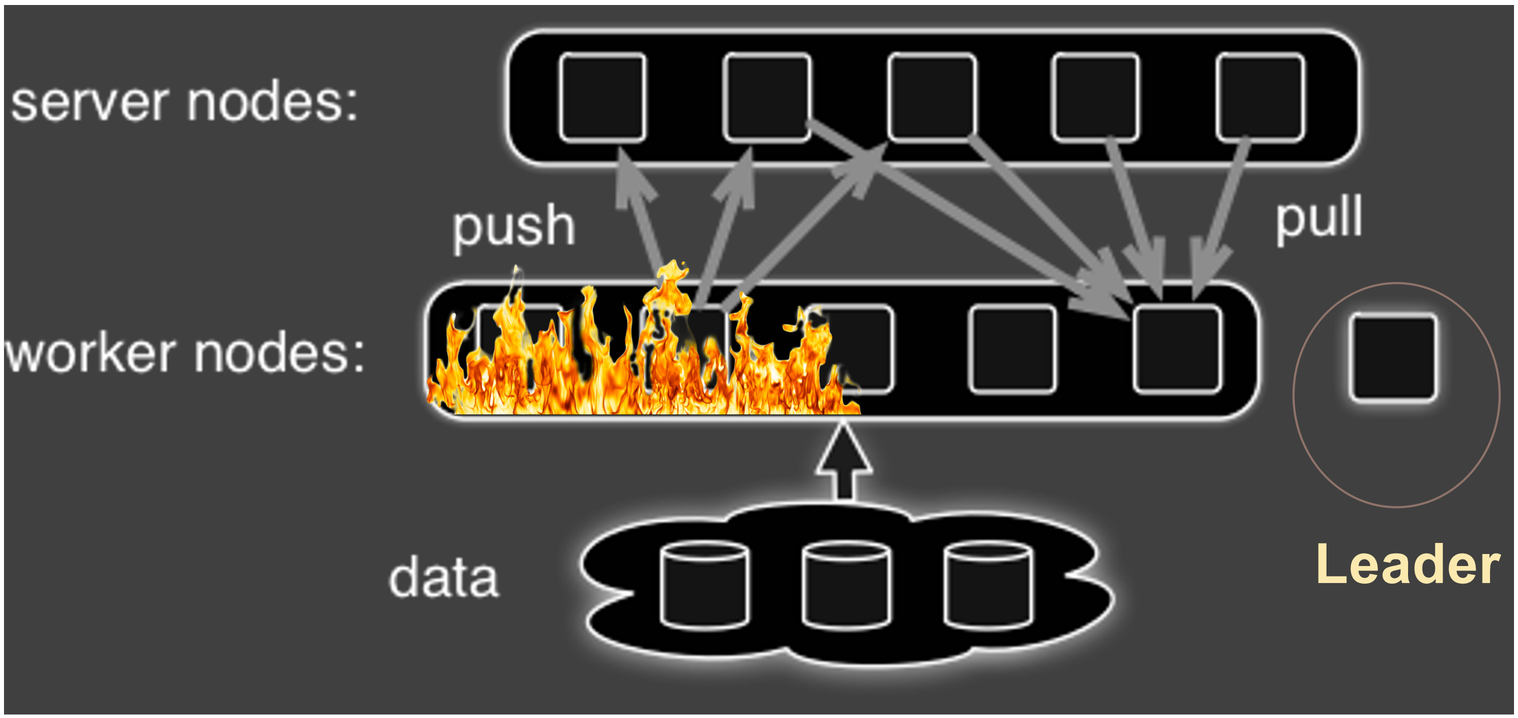


Distinguished Leader

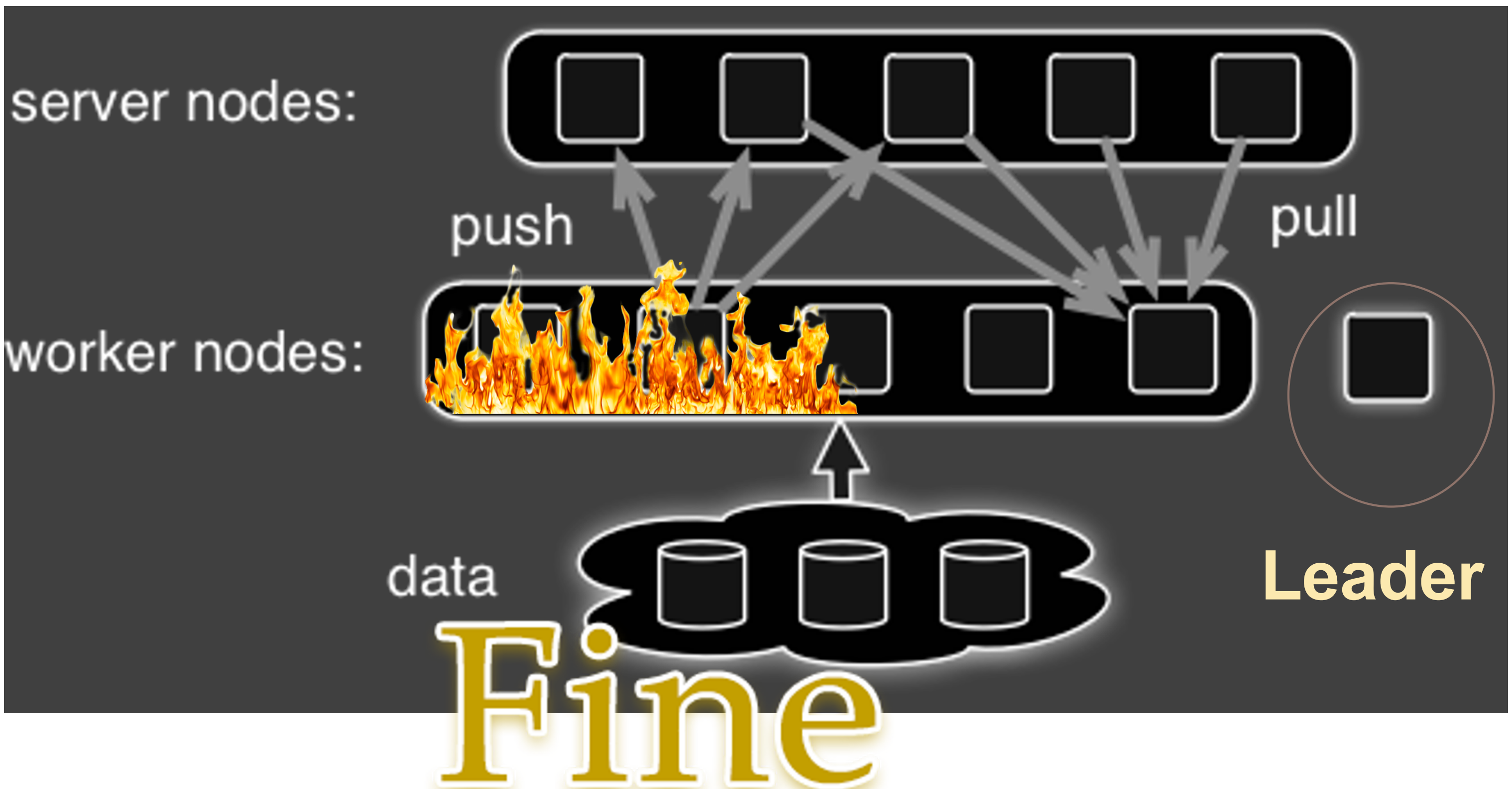
Hardcoded role. No worries about leader election, no consensus

```
saver = tf.train.Saver(sharded=True)
with tf.Session(server.target) as sess:
  while True:
    ... #sleep a bit
    saver.save(sess, "gs://path/to/dump")
    if (bad_thing_happens):
      saver.load(sess,"gs://path/to/dump")
```

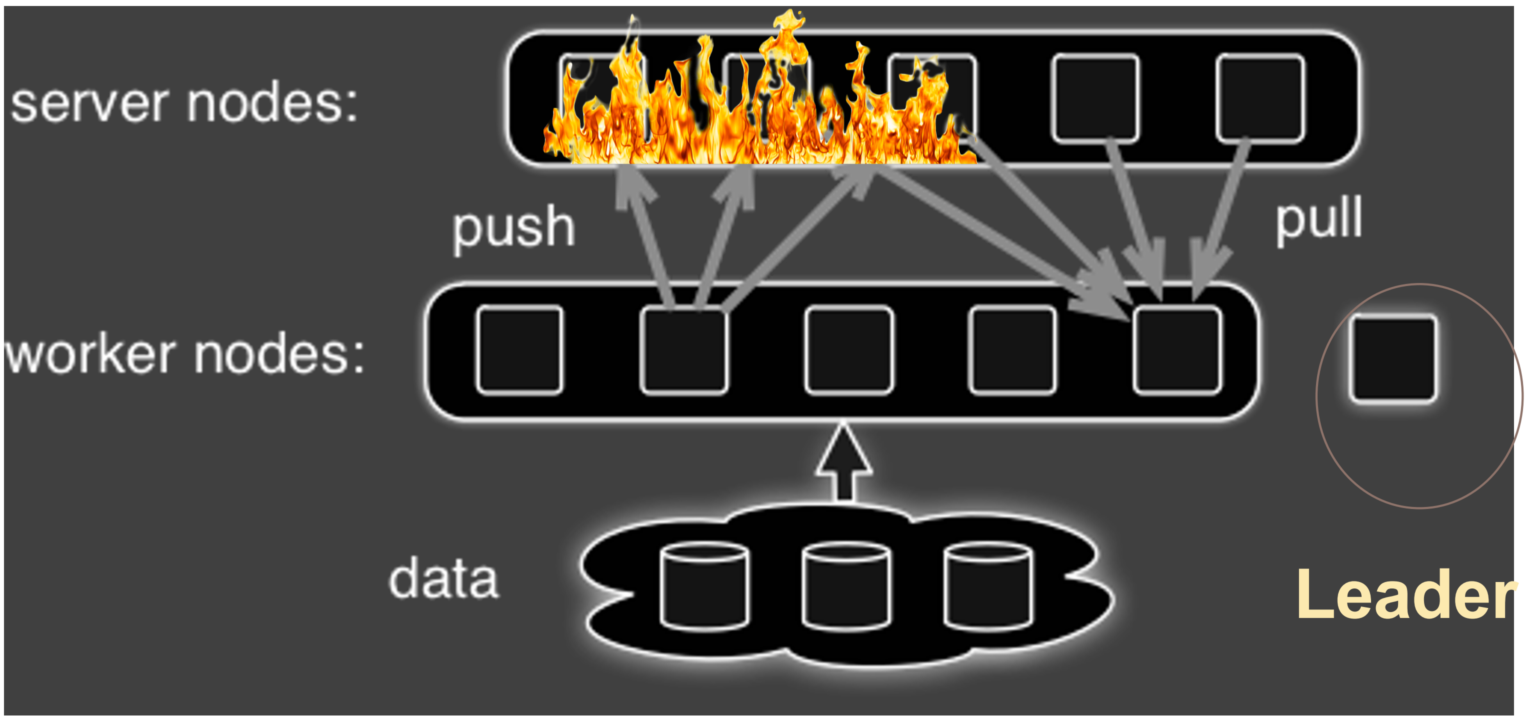
Adding Fault tolerance



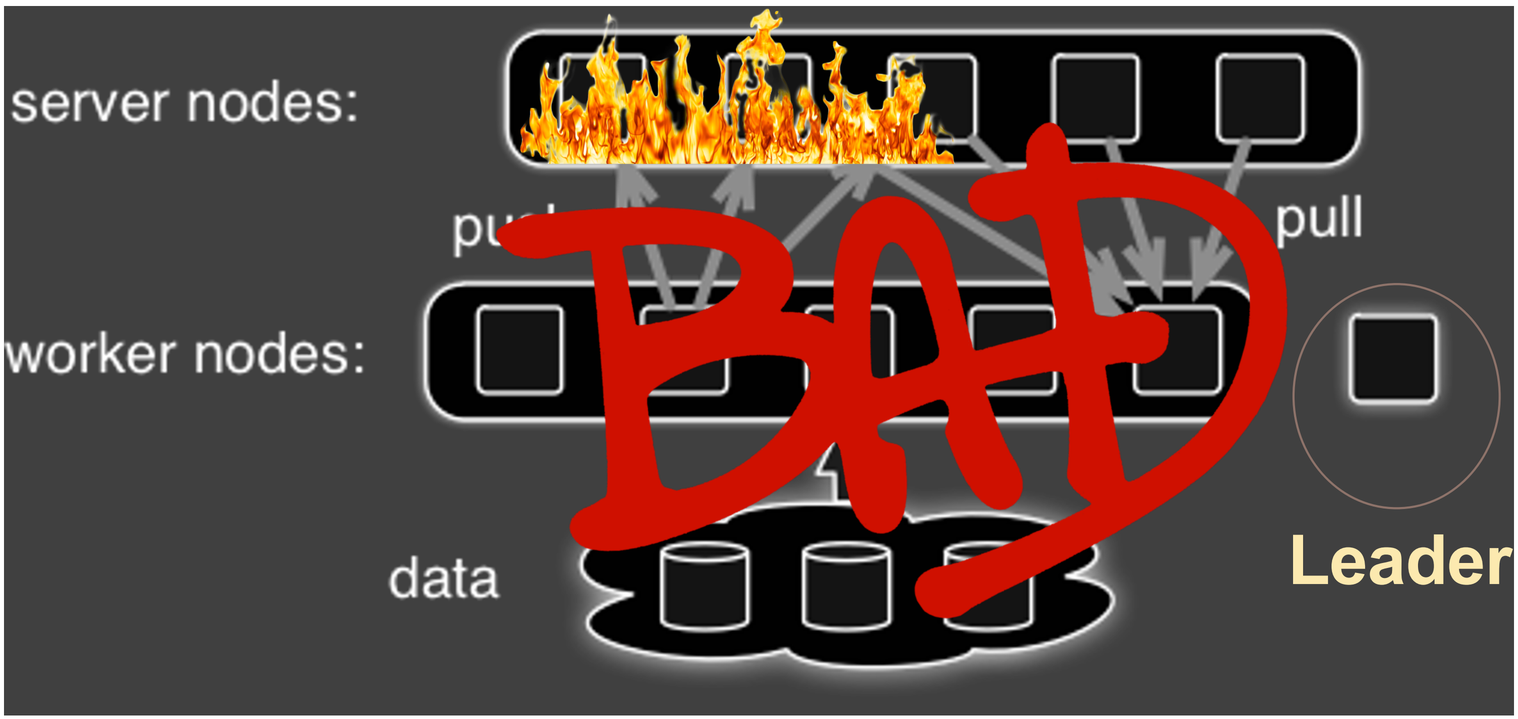
Adding Fault tolerance



Adding Fault tolerance

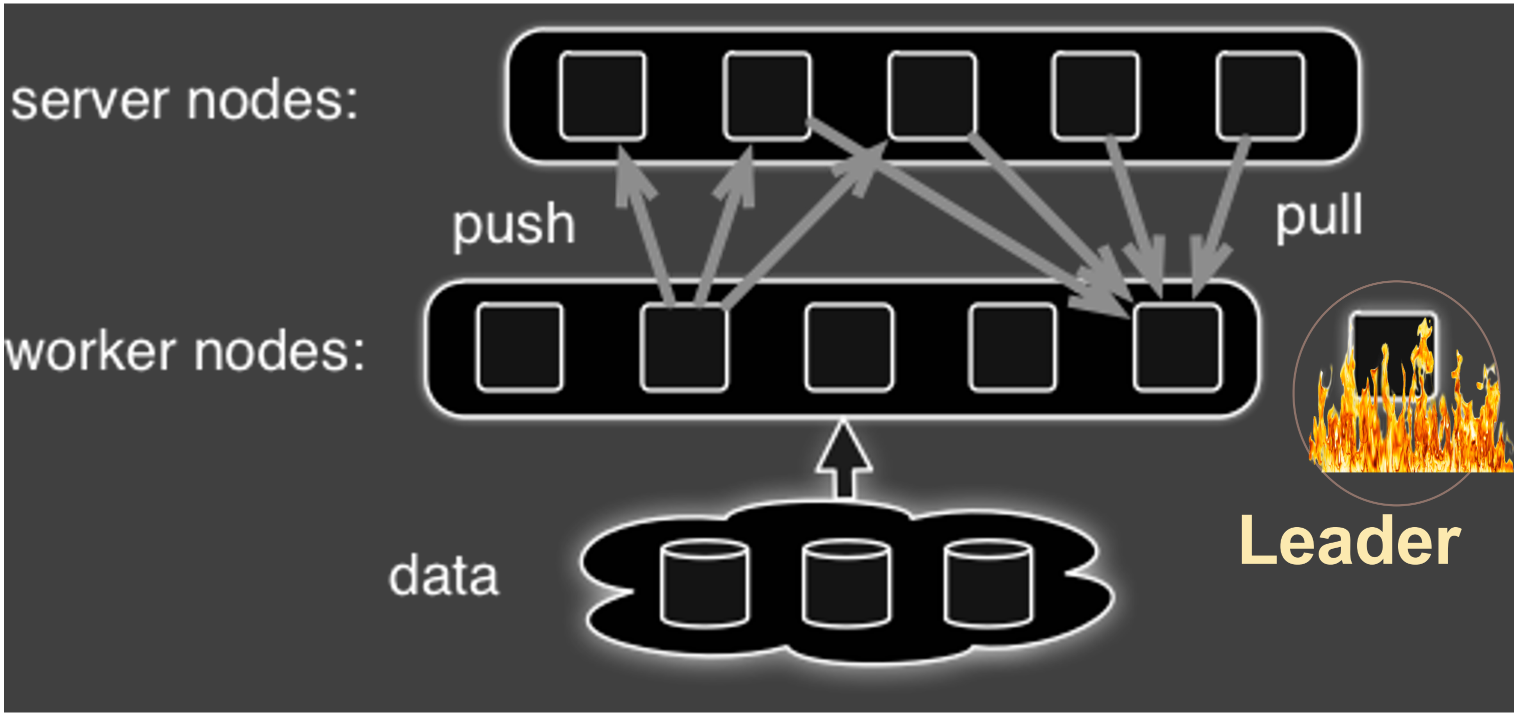


Adding Fault tolerance

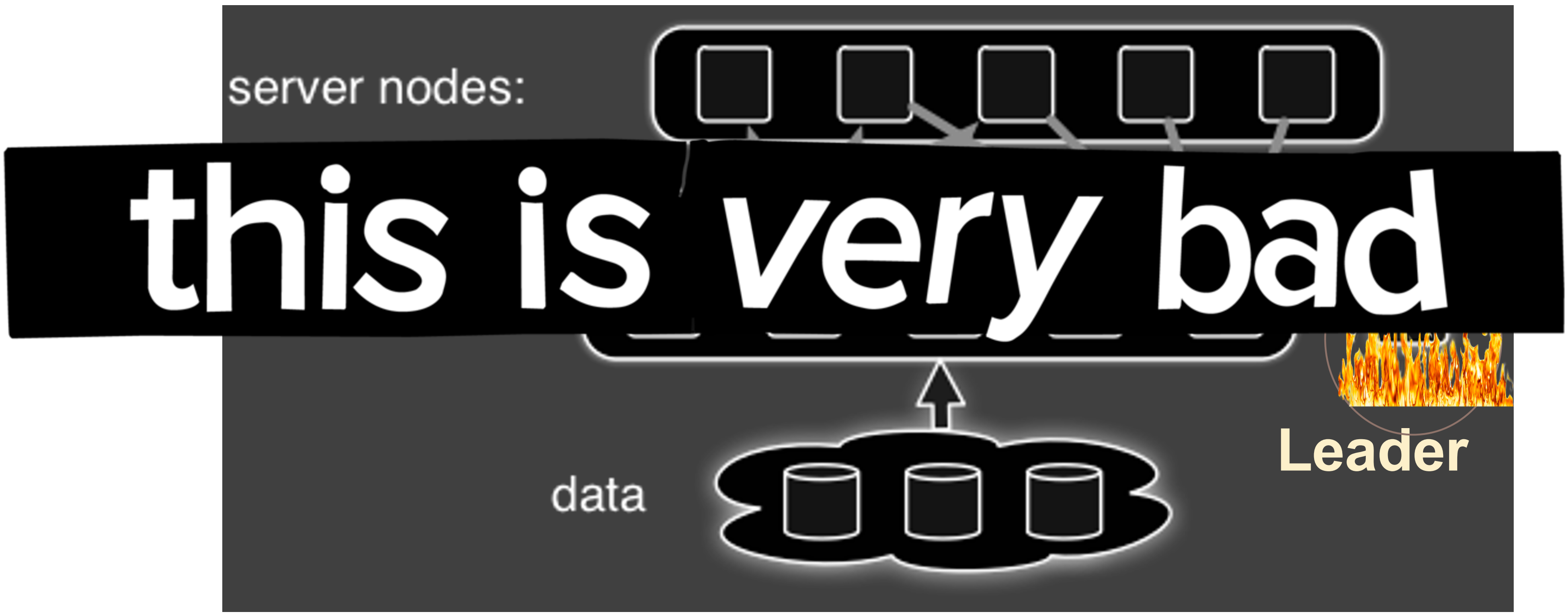


RESTART FROM CHECKPOINT!

Adding Fault tolerance



Adding Fault tolerance



CALL THE OPERATOR! MANUAL INTERVENTION!

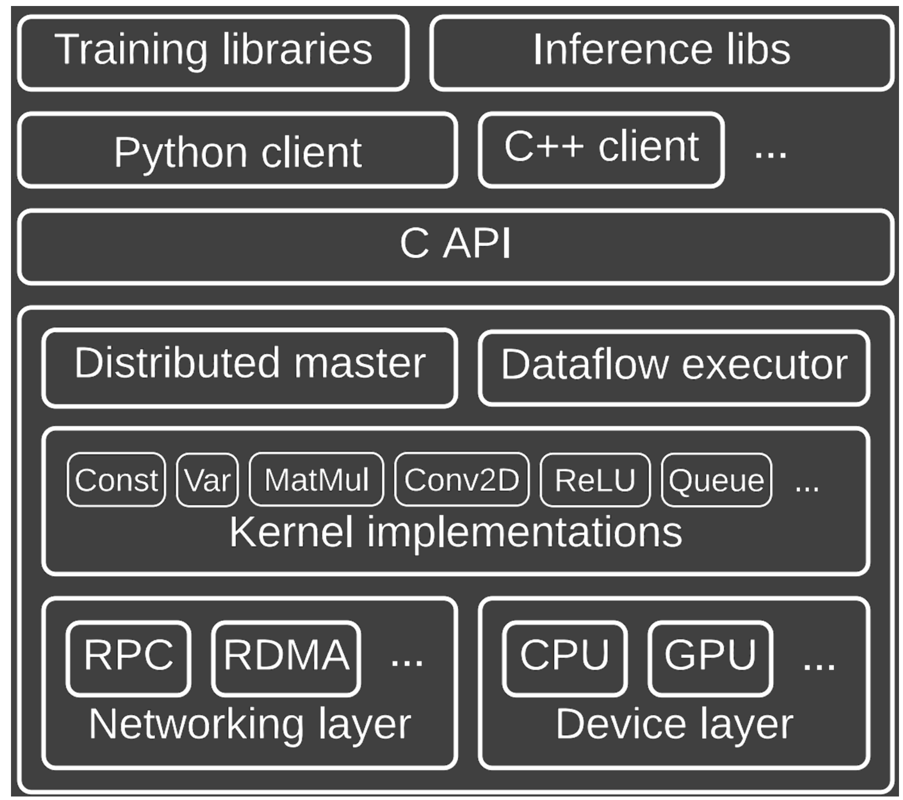
Notes

- There are libraries, but they are still a bit painful.
- Remember to create frequent checkpoints

  Bottom line is that by default, TF is not consistent and is good at restarting from a checkpoint. Recent events not in a checkpoint can be forgotten.

TensorFlow implementation

- Semi-interpreted
- Call to kernel per primitive operation
- Can batch operations with custom C++
- Basic type-safety within dataflow graph (error at graph construction time)
- Global Names: overlapping TF instances share variables!



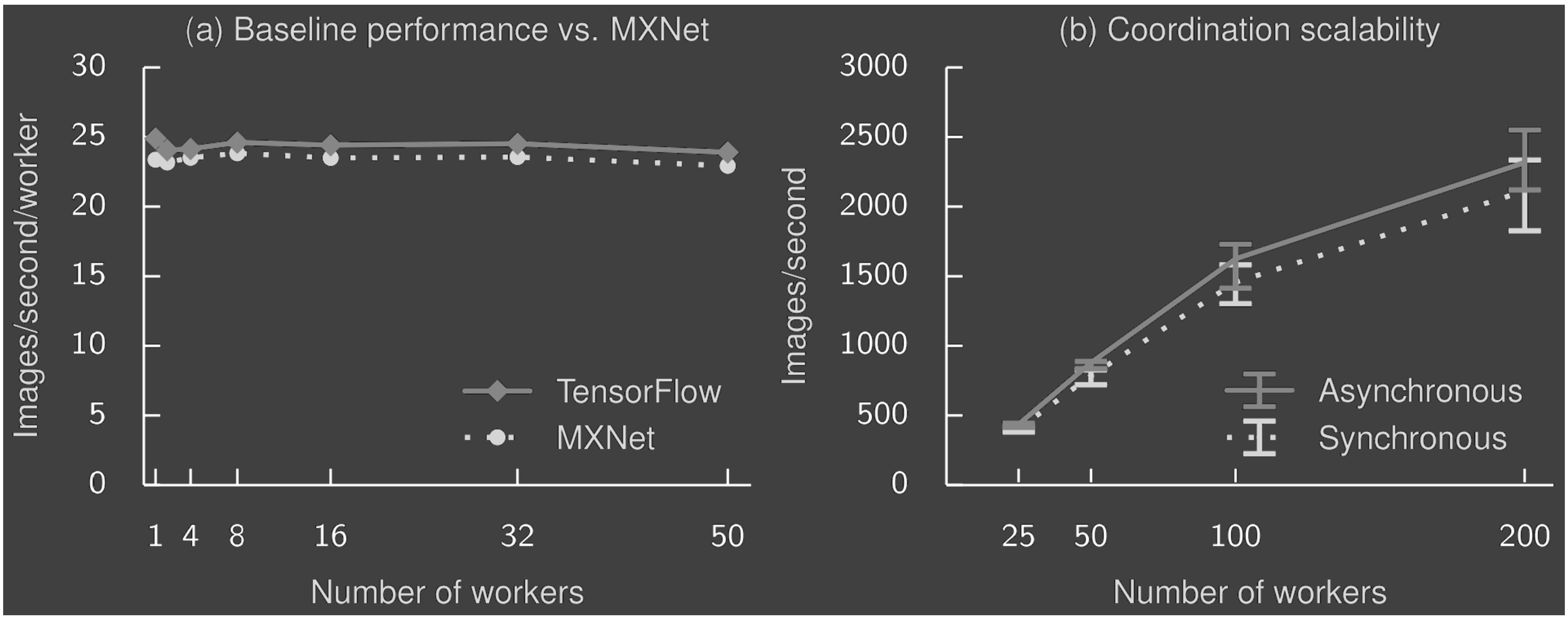
Synchronous vs Asynchronous

- Determined by node: Queue nodes used for barriers
- Synchronous nearly as fast as asynchronous
- Default model is asynchronous

Performance: Single Node

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

Performance: Distributed Throughput



Key Contributions

- Programmability
- Accessibility / ease of use
- Richness of Libraries
- Ready-made community