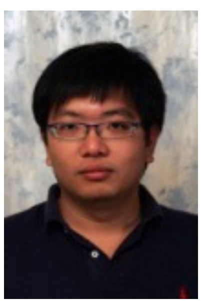
Algorithms and Systems for Scalable Graph-Parallel Inference

Joseph Gonzalez
Postdoc, UC Berkeley AMPLab
Co-Founder GraphLab Inc.
jegonzal@eecs.berkeley.edu

Joint work with:


Yucheng Low


Haijie Gu


Aapo Kyrola


Danny Bickson


Carlos Guestrin


Alex Smola


Guy Blelloch


Joe Hellerstein

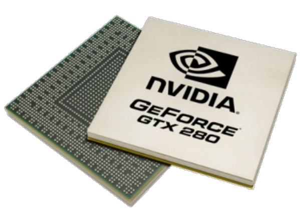
Massive **Structured** Problems

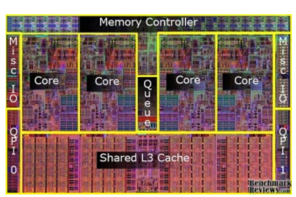
Graphical Model Representations

Parallel and **Distributed** Algorithms
for Probabilistic **Inference**

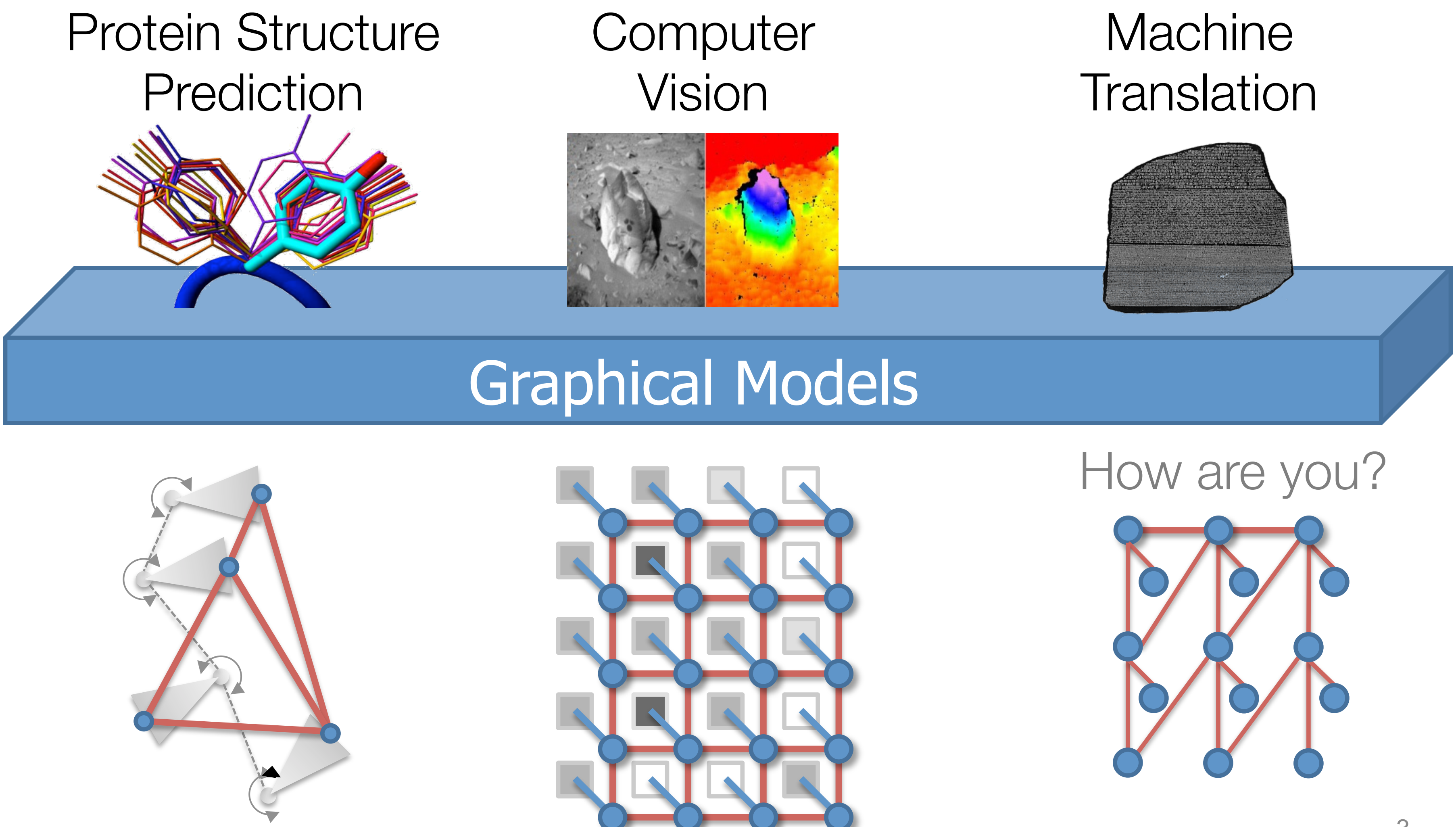
GraphLab: Graph-Parallel Systems

Advances Parallel Hardware





Graphical models provide a **common representation**



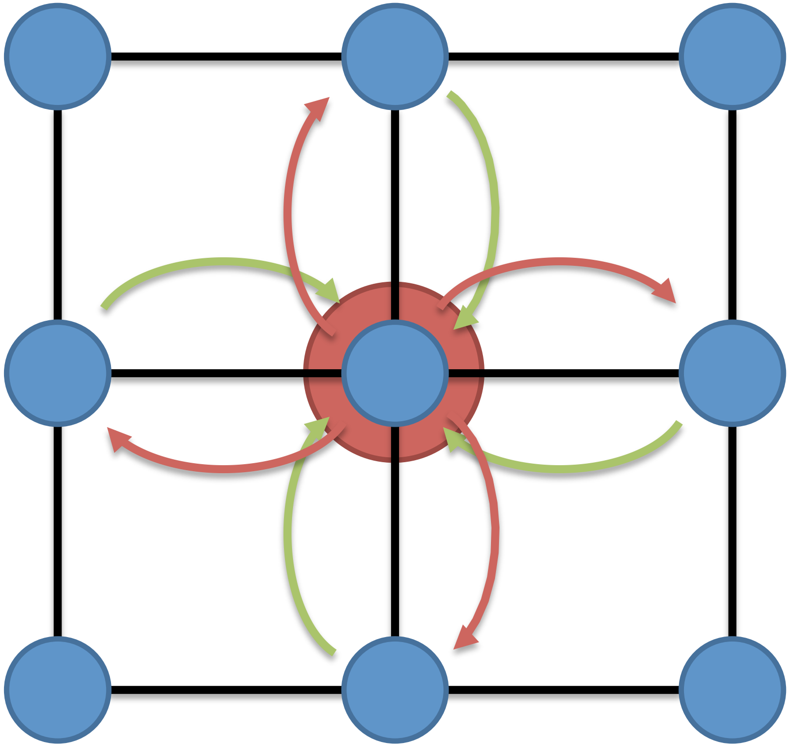
Parallel and **Distributed** Algorithms for Probabilistic **Inference**

Belief Propagation

Gibbs Sampling

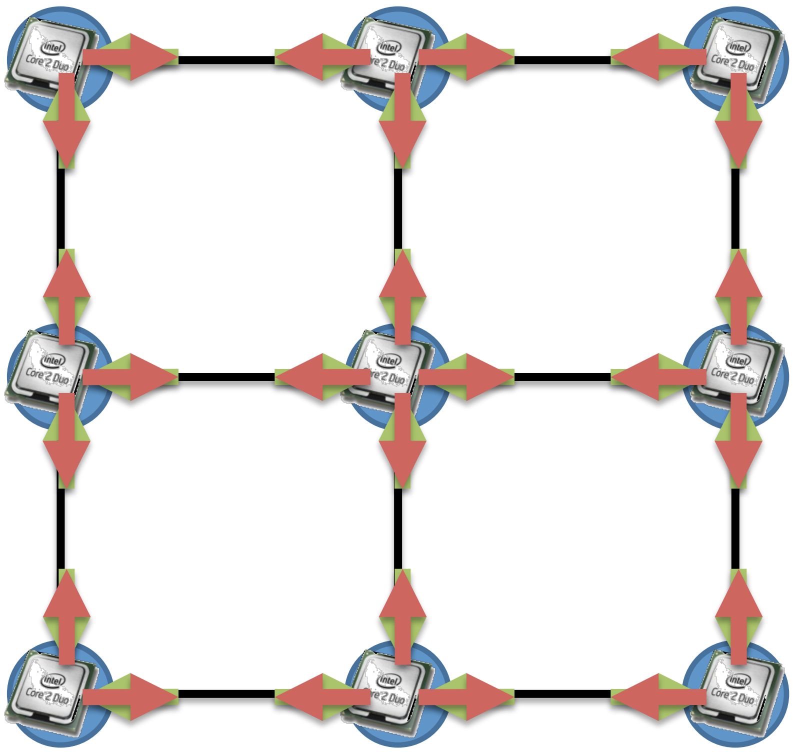
Loopy Belief Propagation (Loopy BP)

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



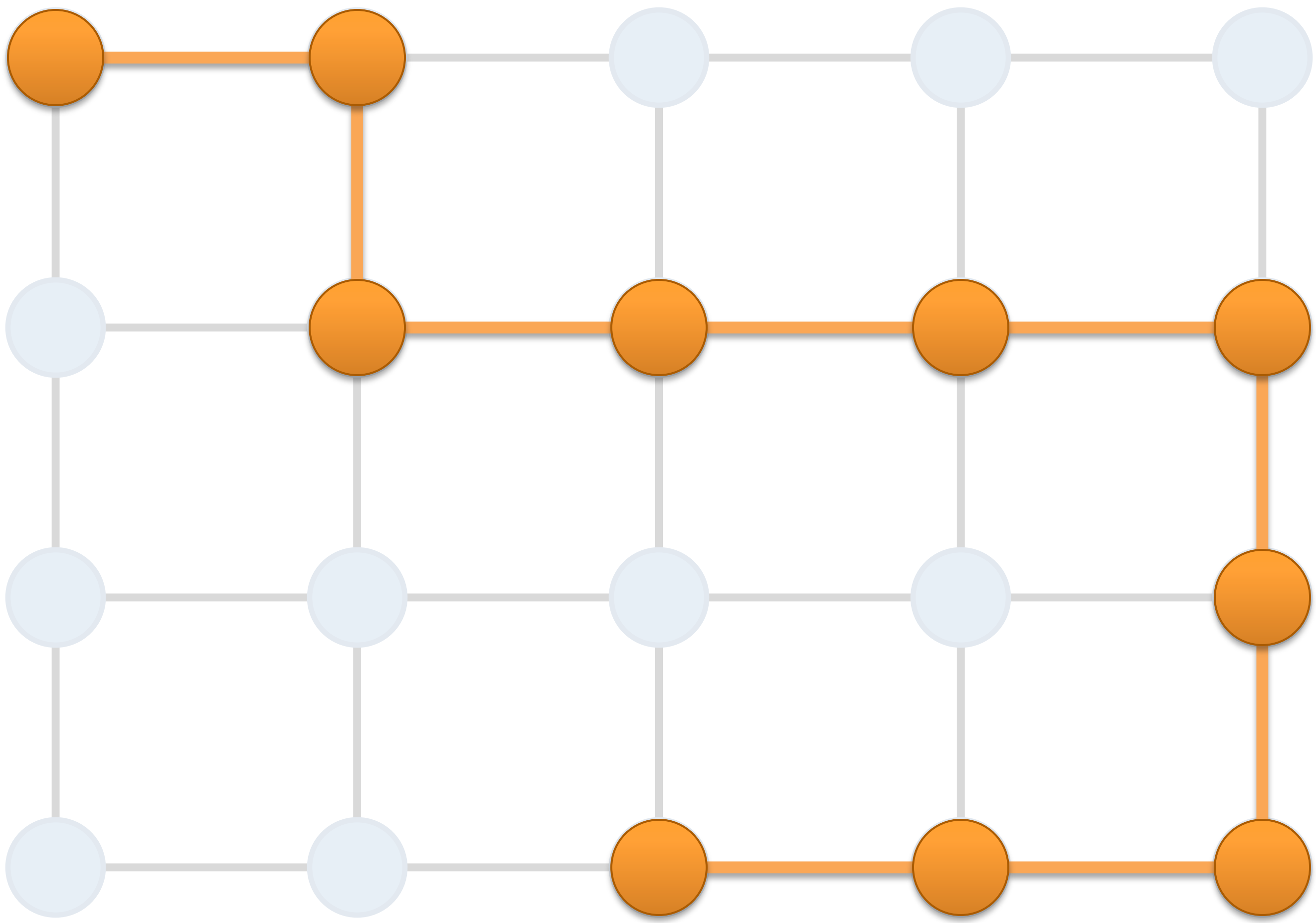
Synchronous Loopy BP

- 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
  - …

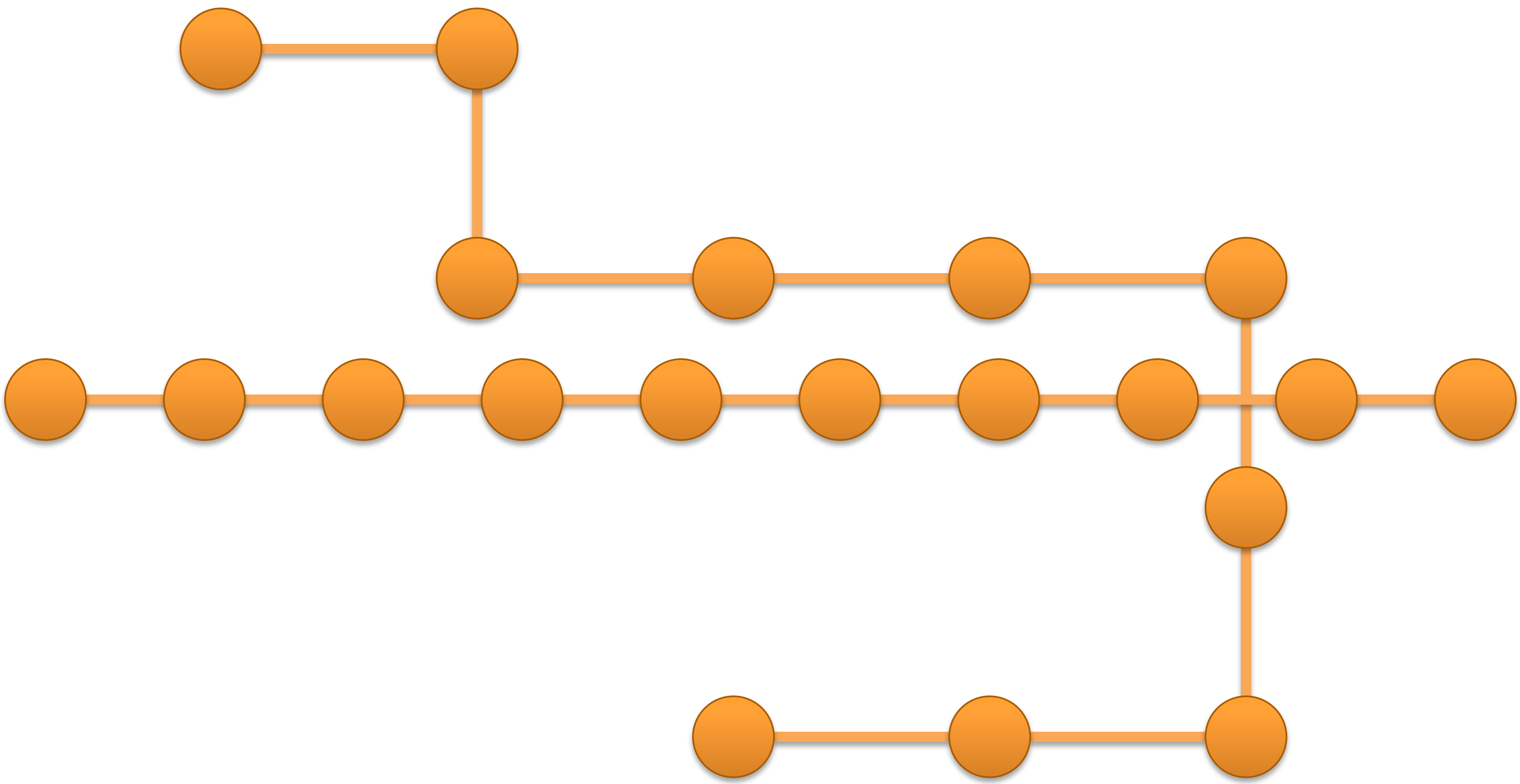


Is Synchronous Loopy BP an **efficient** parallel algorithm?

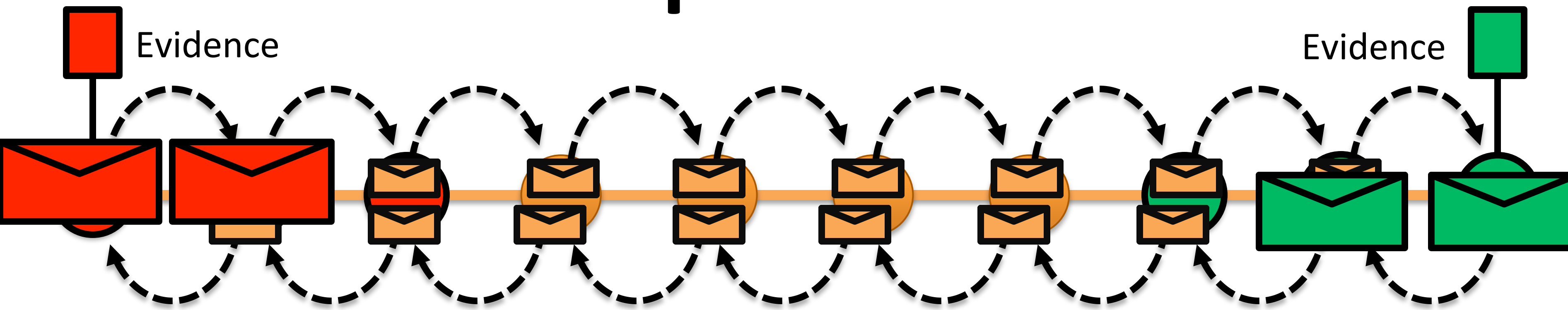
Sequential Computational Structure



Hidden Sequential Structure



Hidden **Sequential** Structure



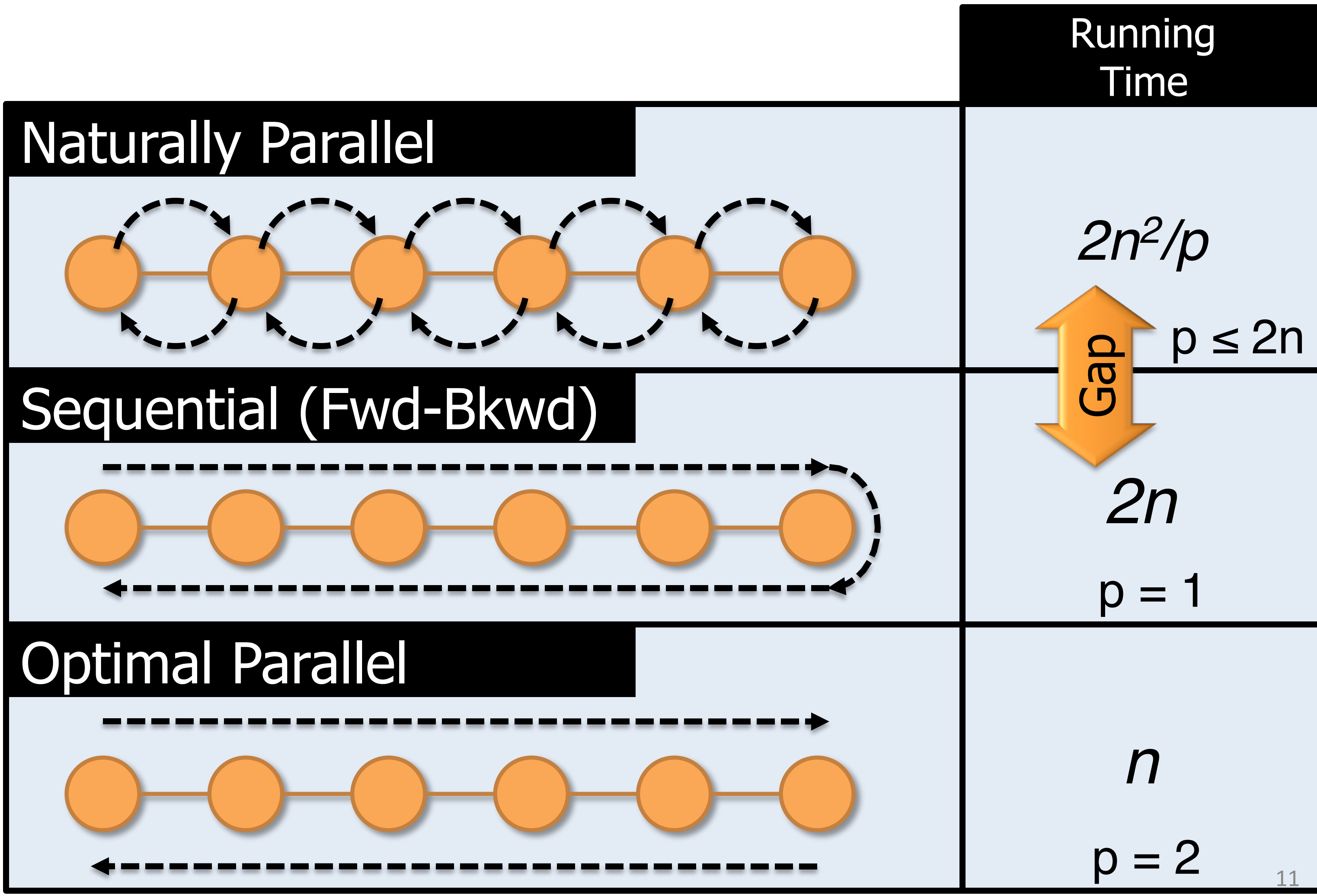
- Running Time:

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

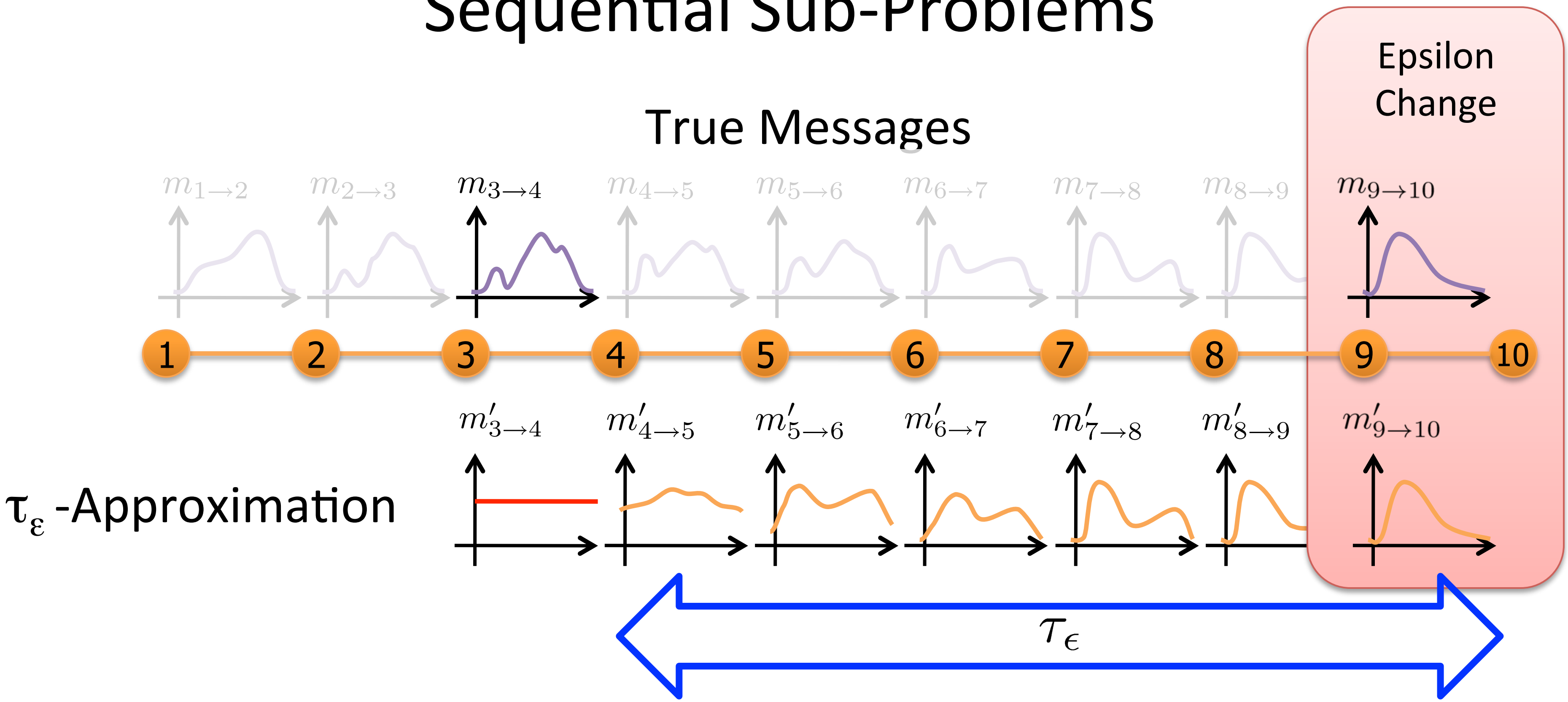
Time for a single parallel iteration

Number of Iterations

Optimal Sequential Algorithm

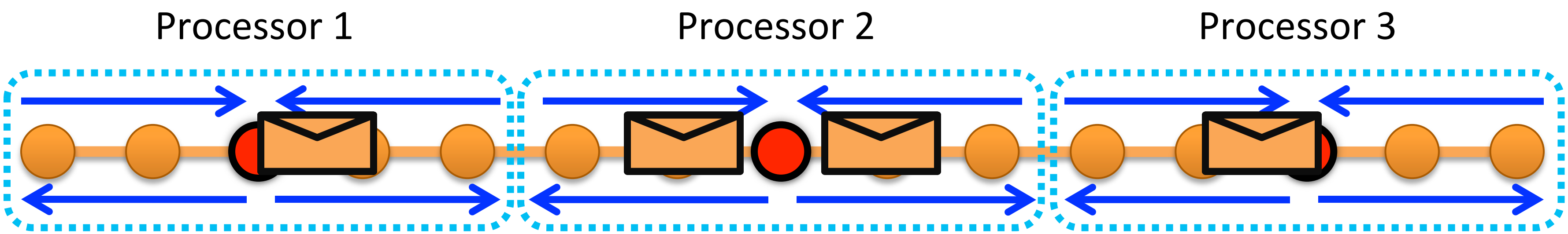


Role of model **Parameters** on Sequential Sub-Problems



Represents the minimal sequential sub-problem

Optimal Parallel Scheduling



Theorem: [AISTATS'09]
Using p processors this algorithm achieves a τ_ε approximation in time:

Parallel Component

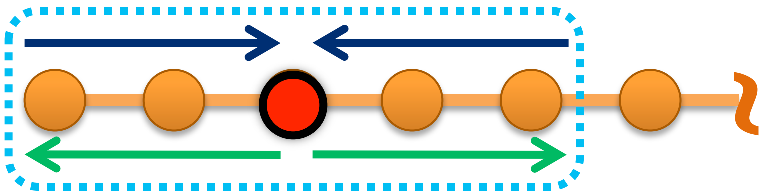
$$O\left(\frac{n}{p}+\tau_\epsilon\right)$$

Sequential Component

and is **optimal** for chain graphical models.

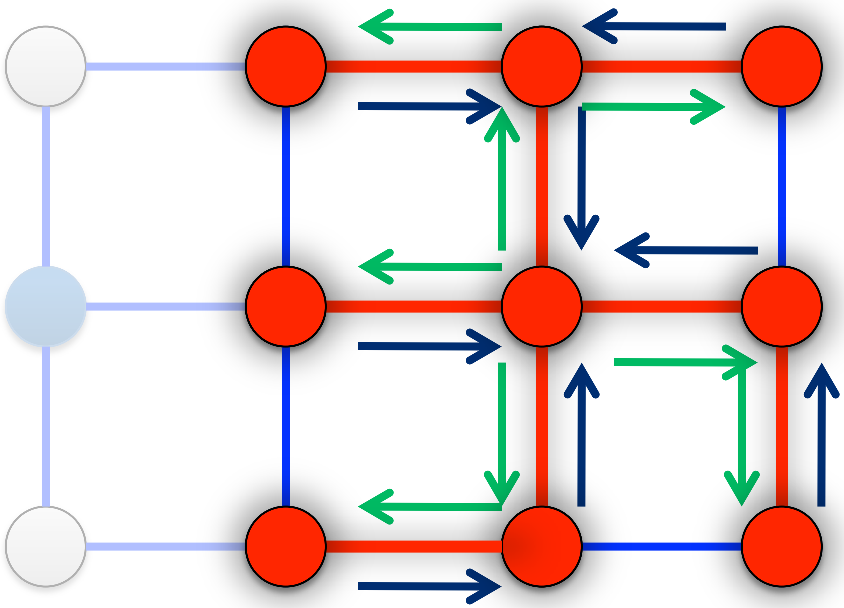
The Splash Operation

- Generalize the optimal chain algorithm:

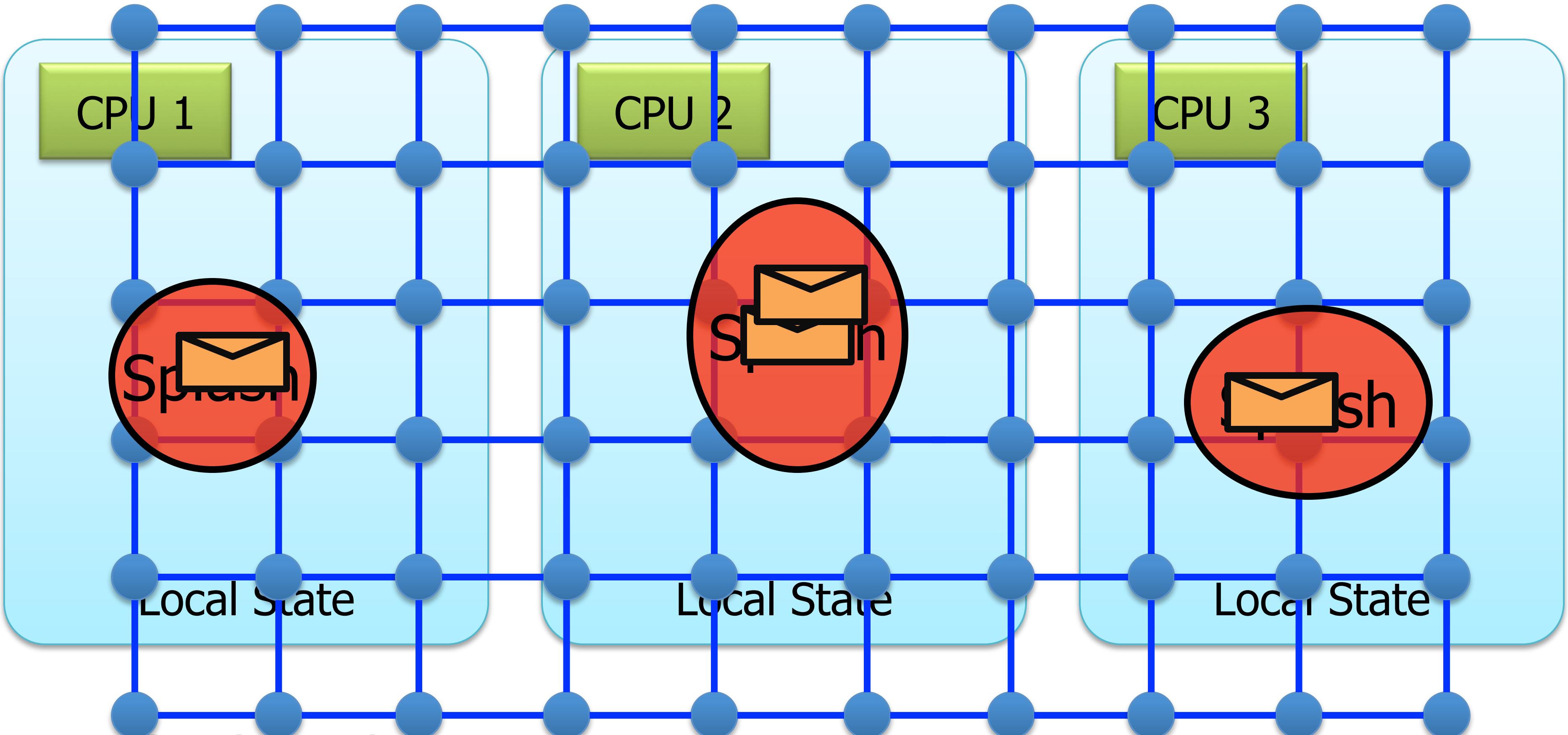


to arbitrary cyclic graphs:

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



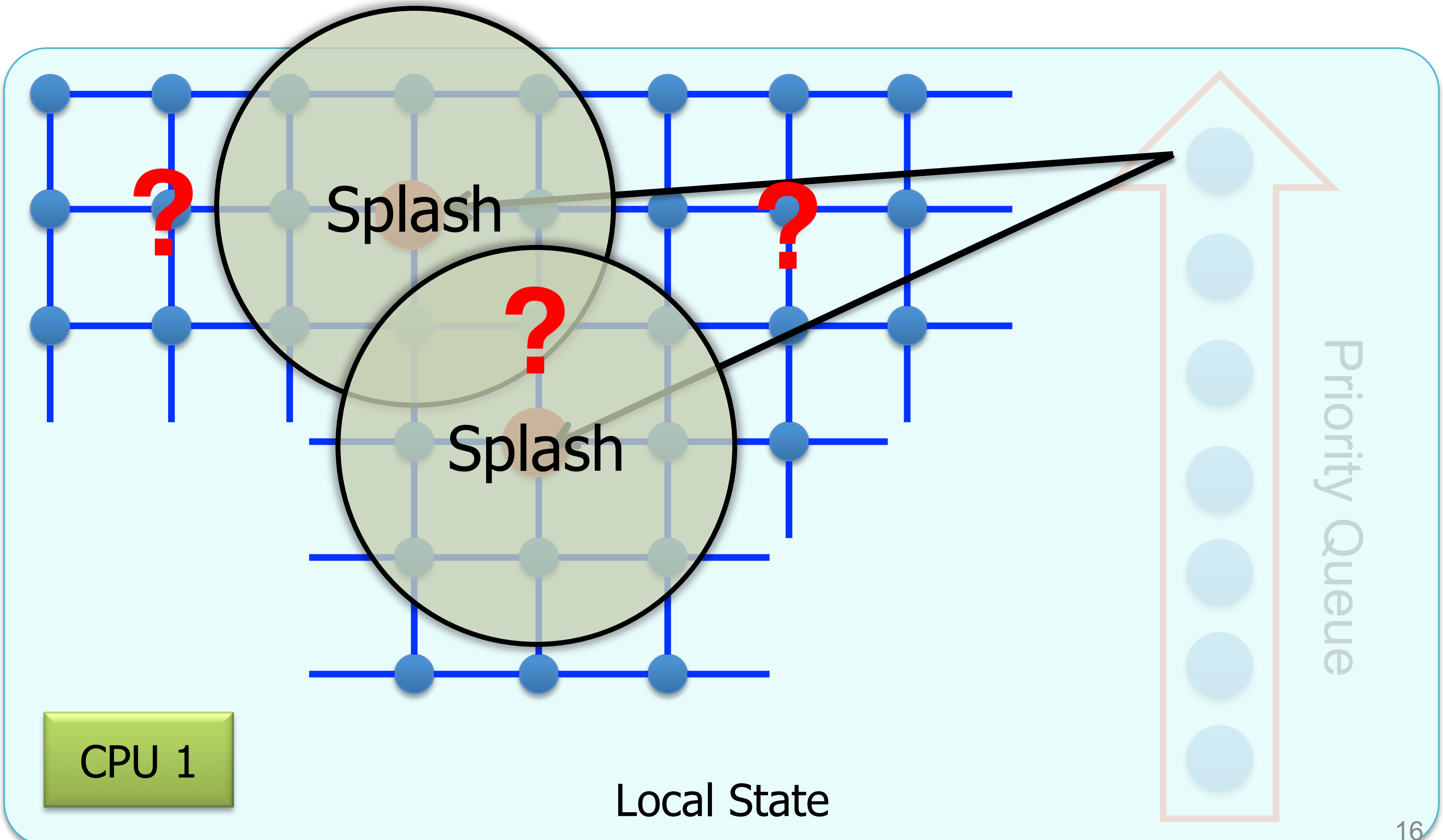
Distributed Splashes [UAI'09]



- Partition the graph
- Schedule Splashes locally
- Transmit the messages along the partition

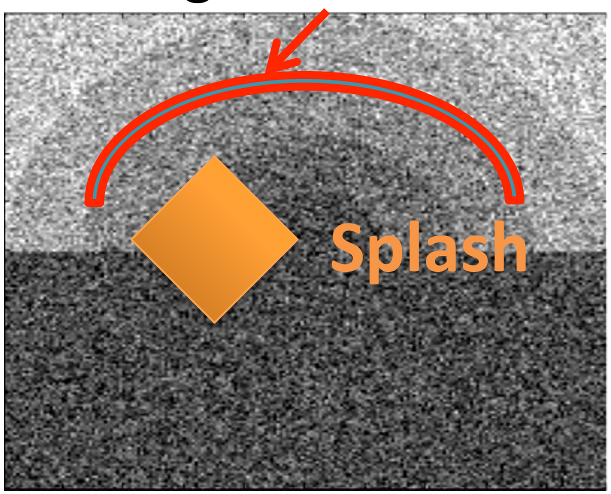
Priorities Determine the **Roots**

- Use a residual priority queue to select roots:

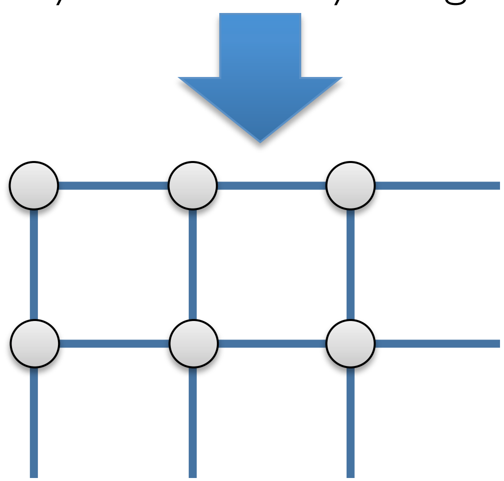


Adaptive Belief Propagation

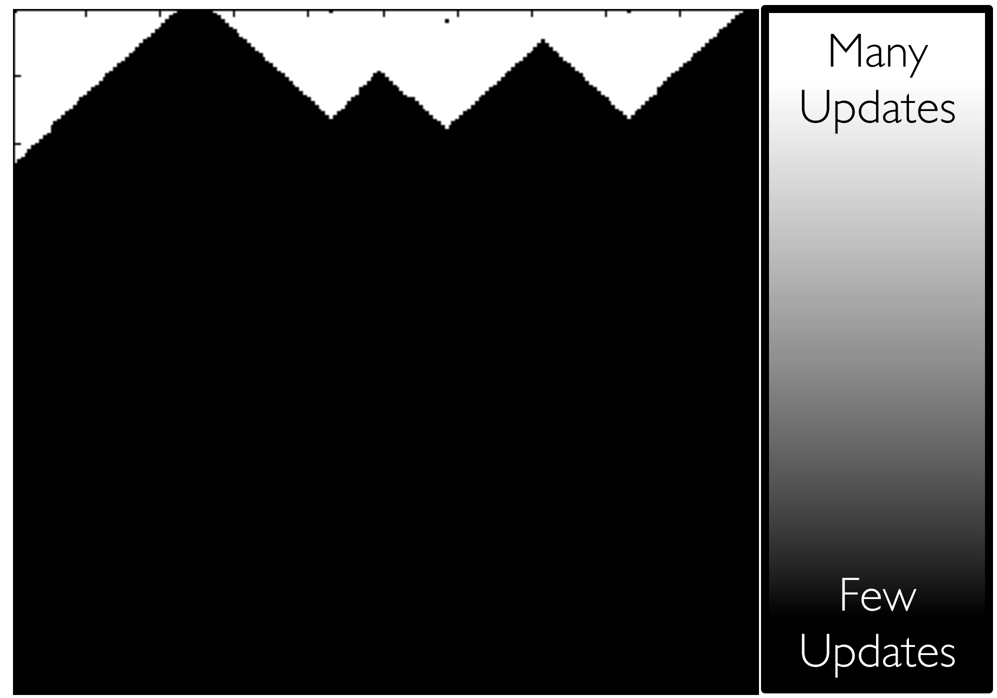
Challenge = Boundaries



Synthetic Noisy Image



Graphical Model

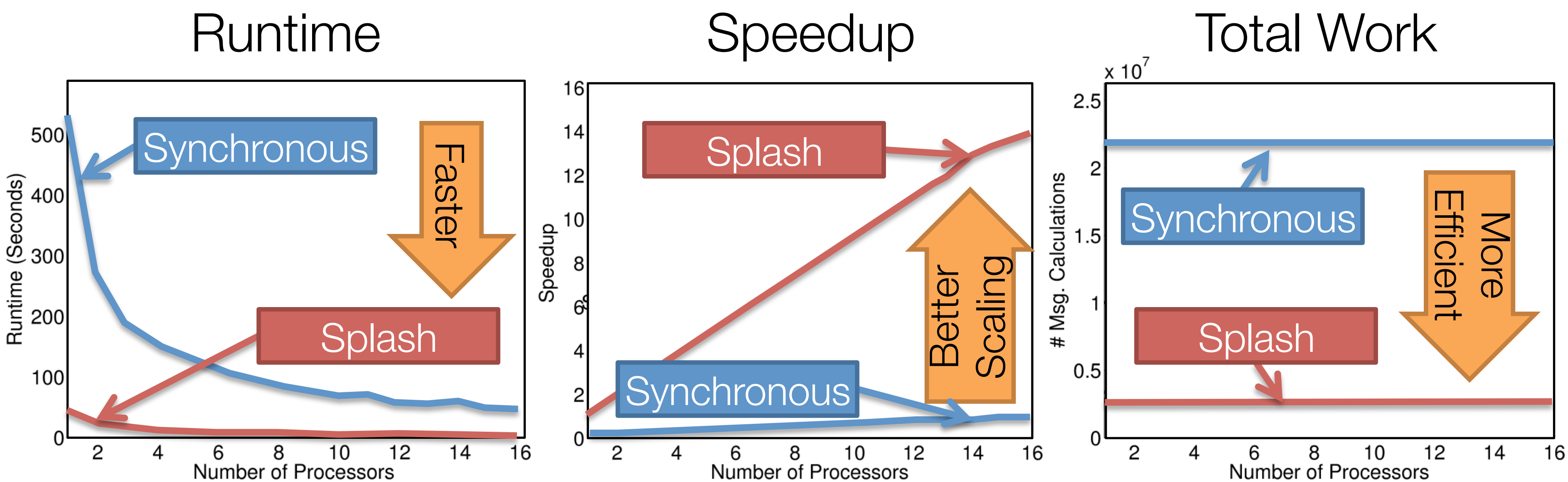


Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

Representative Results

Protein Interaction Models: 14K Vertices, 21K Factors

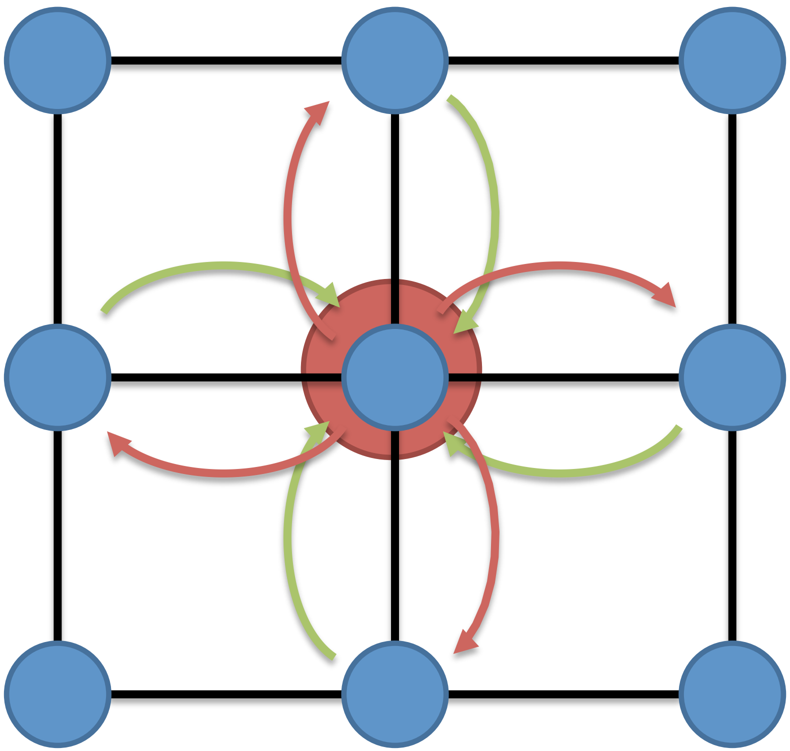


Dynamic Asynchronous (SplashBP)

- Faster and More Efficient
- Converges more often
- Achieves better prediction accuracy

Parallel and **Distributed** Algorithms for Probabilistic **Inference**

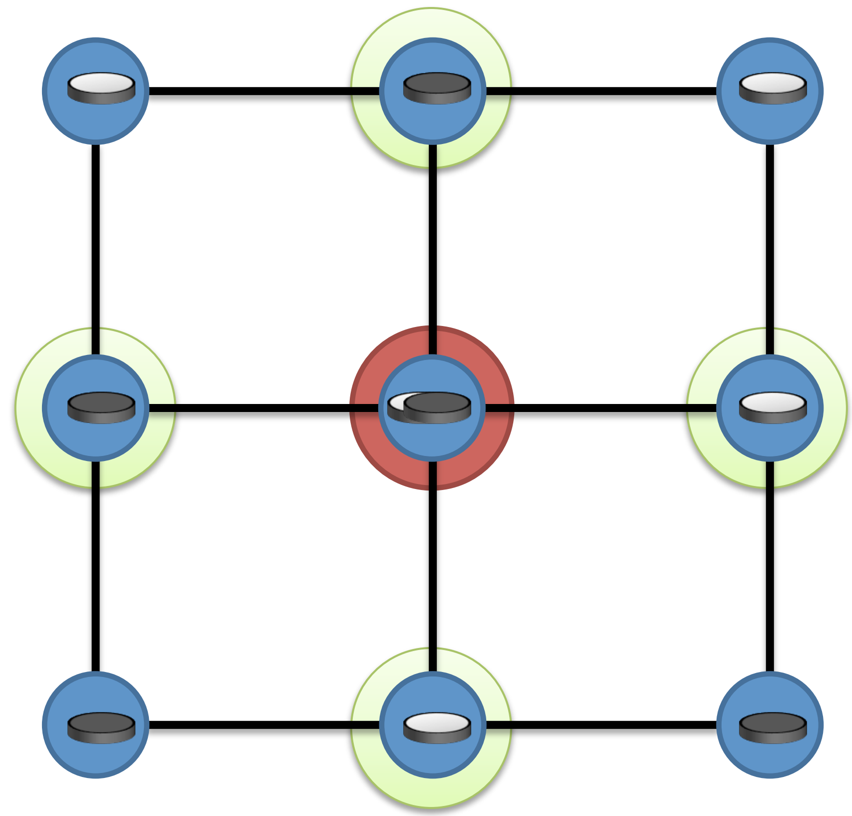
Belief Propagation

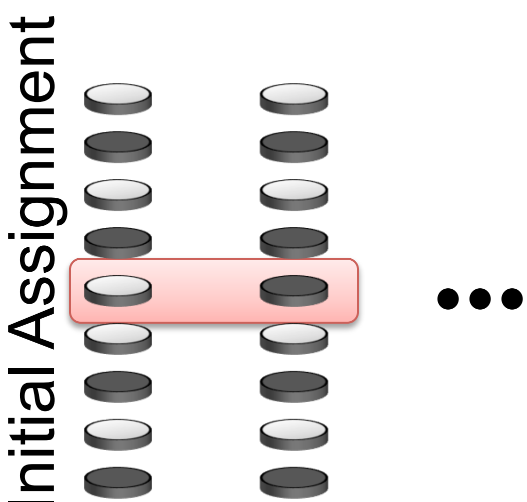


Gibbs Sampling

Gibbs Sampling [Geman & Geman, 1984]

- **Sequentially** for each variable in the model
  - Select variable
  - Construct condition using adjacent assignments
  - Sample from conditional



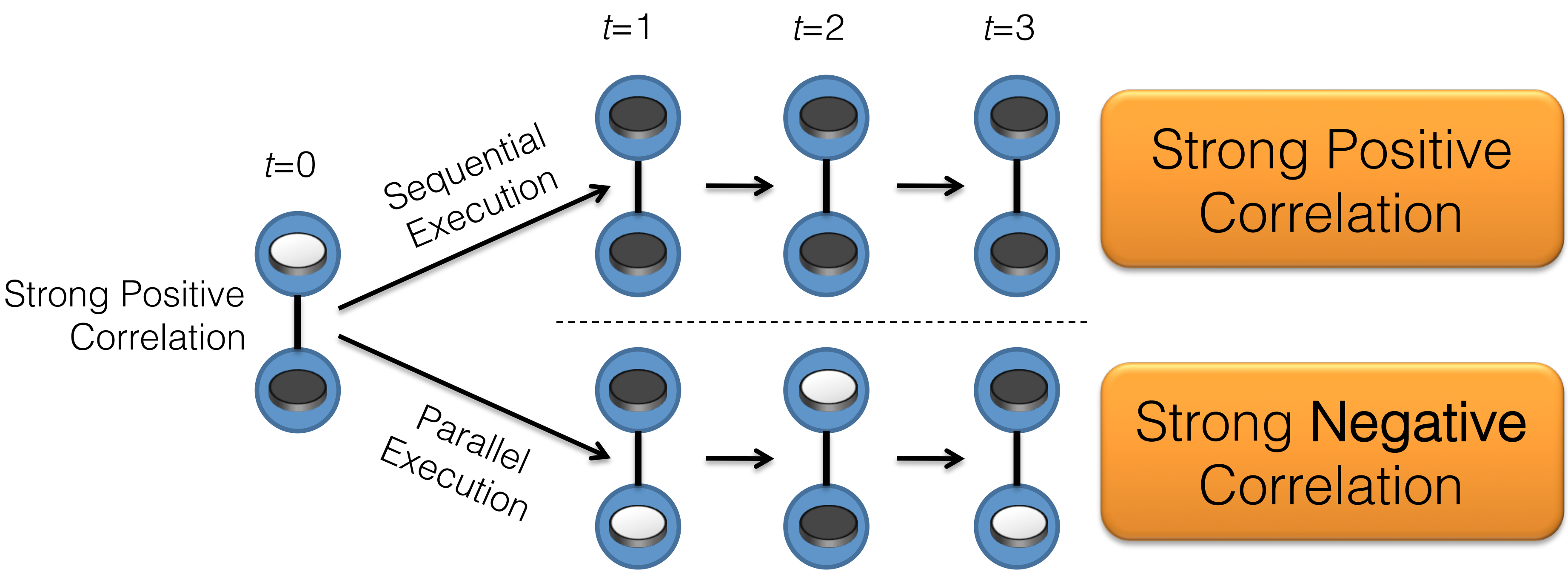


Synchronous Gibbs Sampling

Embarrassingly Parallel!

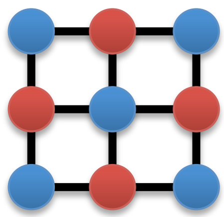
Converges to the **wrong** distribution!

The Problem with Synchronous Gibbs Sampling

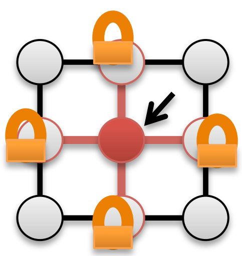


Adjacent variables ***cannot*** *be sampled* ***simultaneously****.*

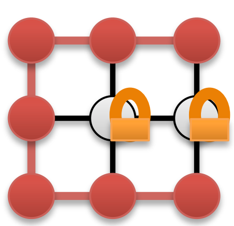
Three Convergent **Parallel** Samplers
[AISTATS'11]



Chromatic: Use graph coloring to *synchronously* sample independent sets



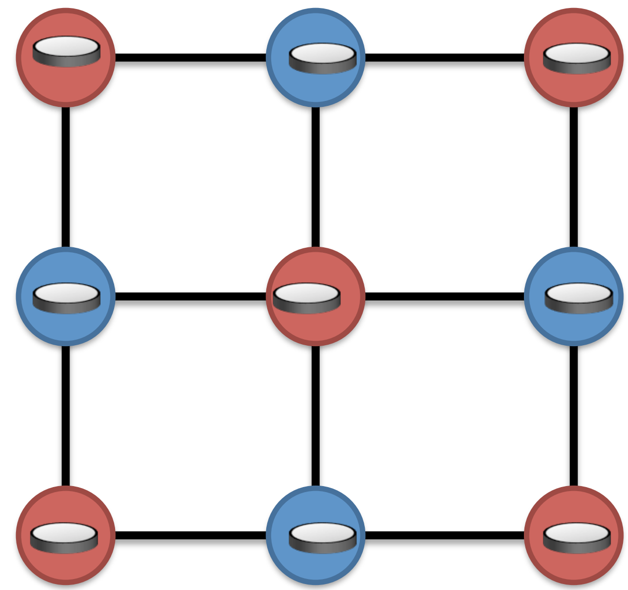
Asynchronous: Enable *prioritized scheduling* using Markov Blanket Locks to ensure serializable execution



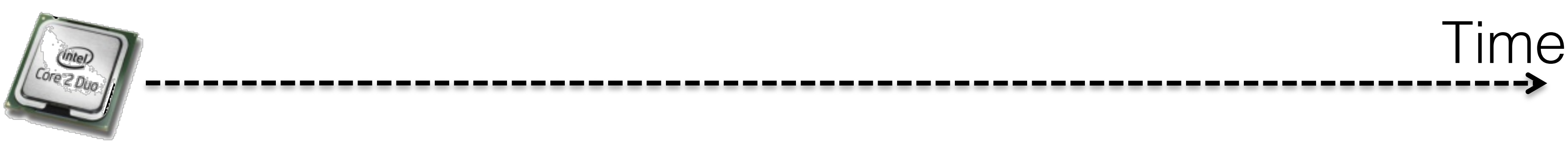
Splash: Address strong dependencies by adaptively constructing *thin junction tree blocks*

Chromatic Sampler

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel



- Serial Equivalence:



Theorem: *Chromatic* Sampler

- Converges to the correct distribution
  - Based on graph coloring of the Markov Random Field
- **Quantifiable** acceleration in **mixing**

Time for a single scan

$$O\left(\frac{n}{p} + k\right)$$

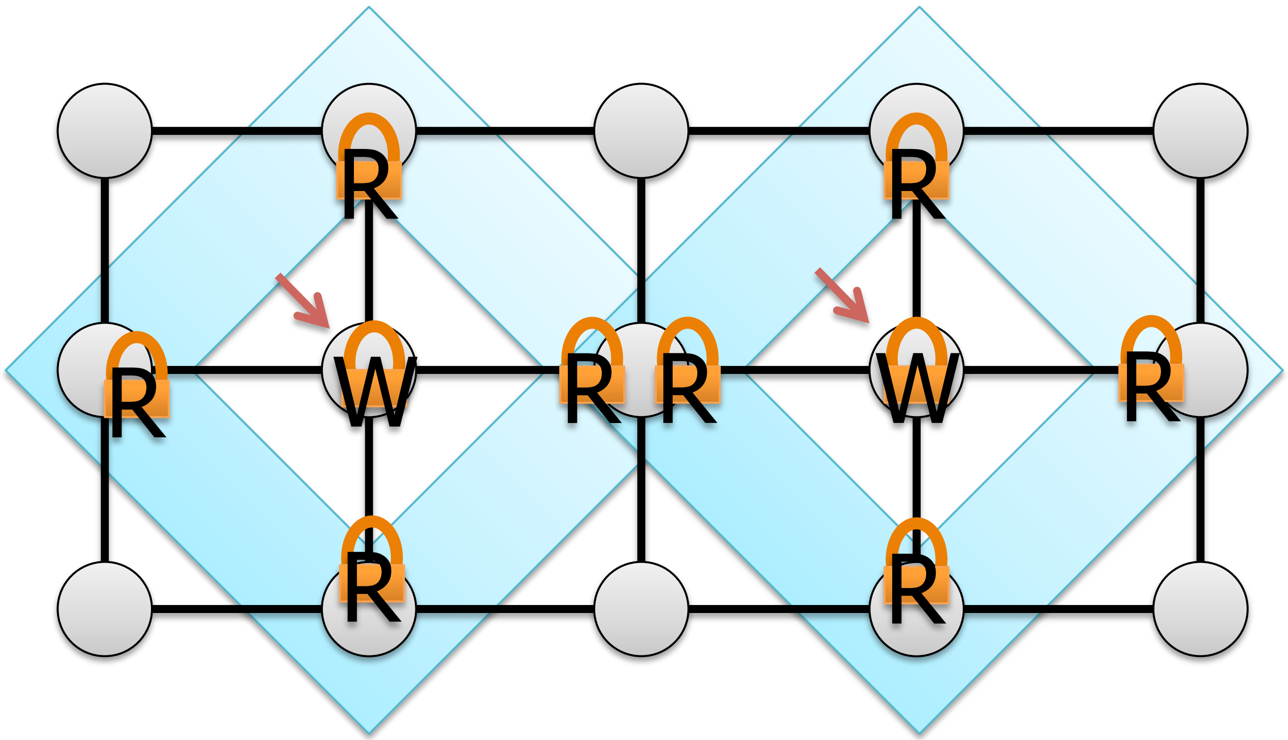
Variables

Colors

Processors

Asynchronous Gibbs Sampler:
Serial Equiv. through **Markov Blanket Locks**

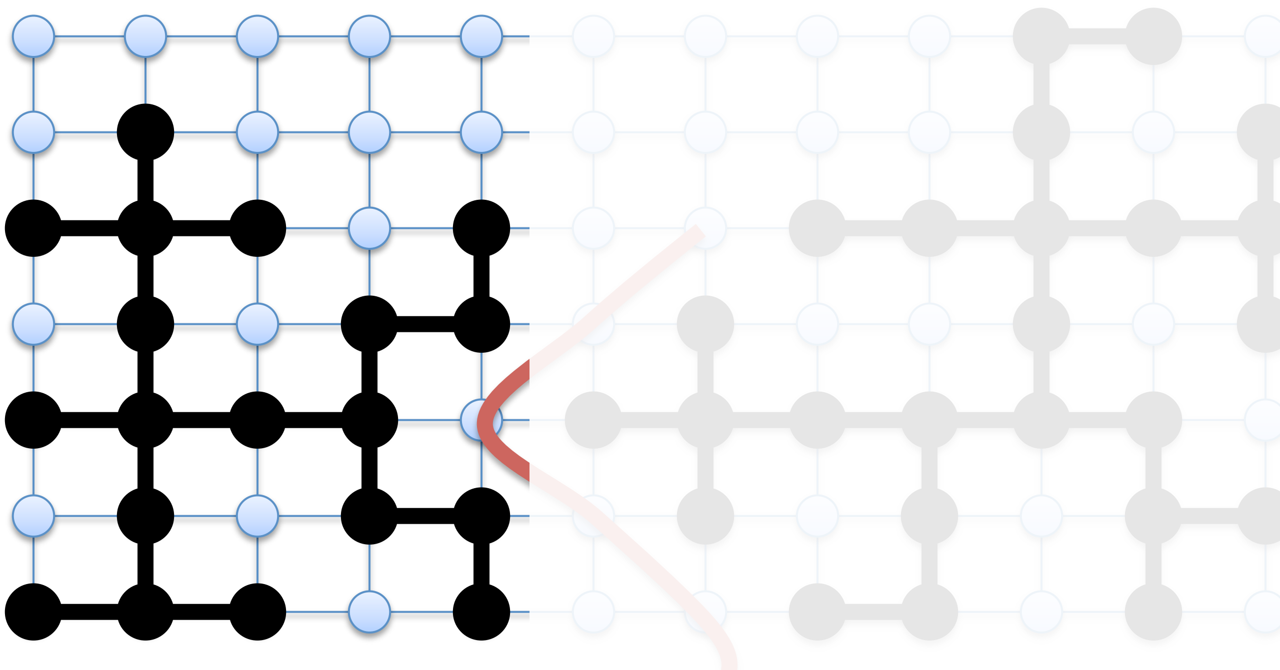
- Read/Write Locks:



- Enables asynchronous, prioritized sweeps

Splash Gibbs Sampler

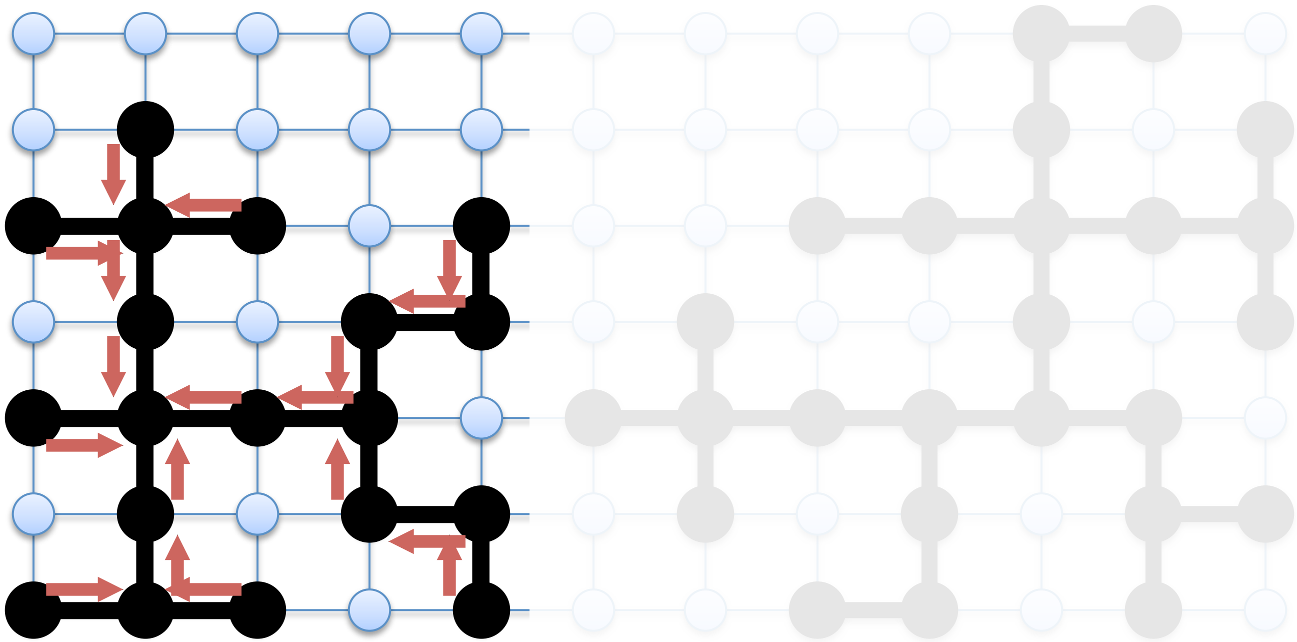
- Asynchronously grow bounded size Splashes:



Focus on a Single Splash

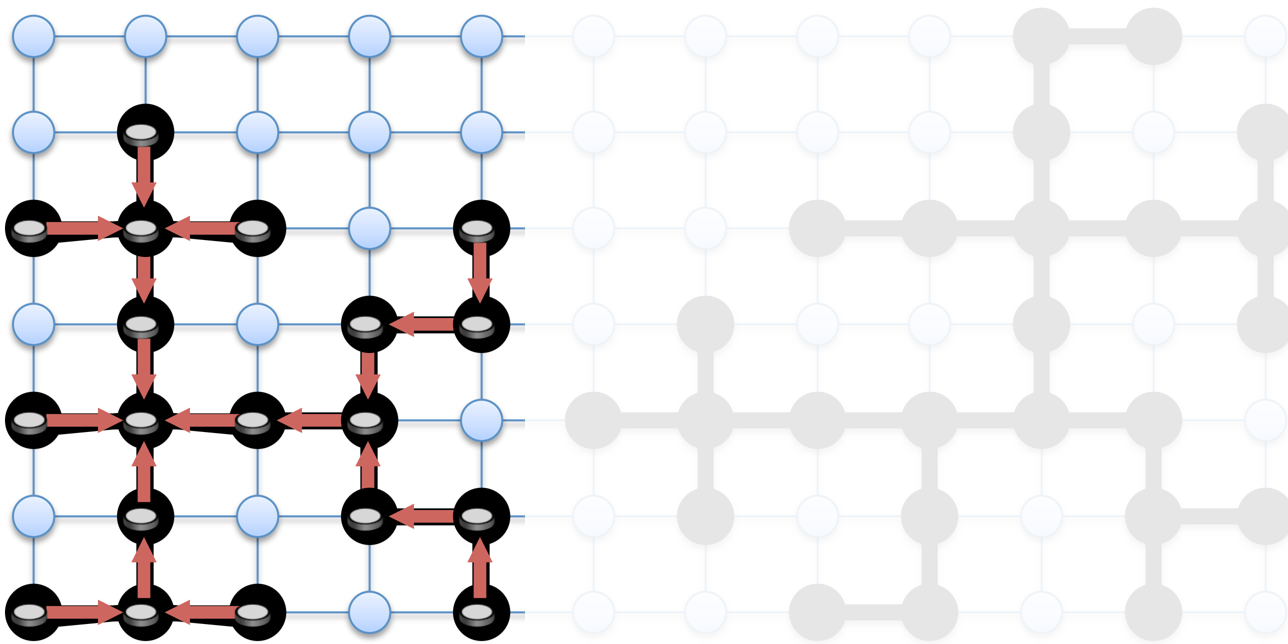
Splash Gibbs Sampler

- Pass BP messages up the tree in parallel



Splash Gibbs Sampler

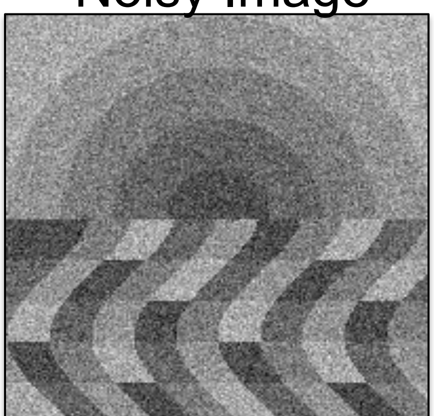
- Asynchronously sample outwards in parallel:



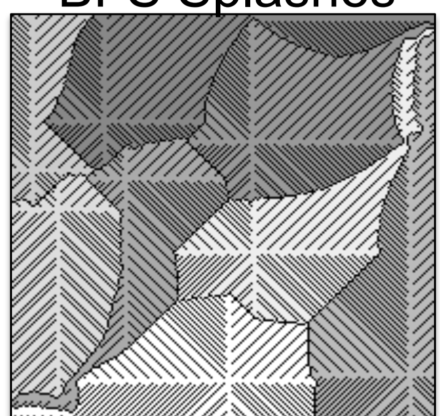
Dynamically Prioritized Sampling

- Prioritize Gibbs updates
- Adapt the **shape** of the junction tree to span strongly coupled variables:

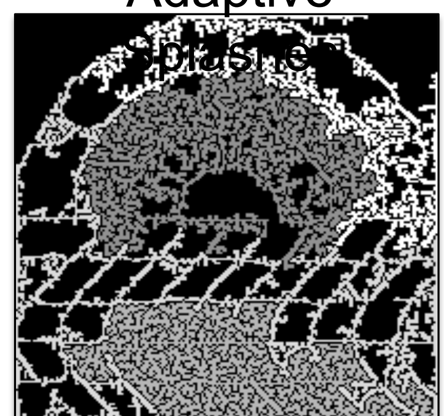
Noisy Image



BFS Splashes



Adaptive Splashes



Theorem
Asynchronous and *Splash Gibbs* Sampler

- **Ergodic:** converges to the correct distribution
  - Requires vanishing adaptation

- **Expected Parallelism:**

$$\mathbf{E}(\#\text{active processors}) \geq 1 + (p-1)\left(1 - (p-1)\left(\frac{d+1}{n}\right)\right)$$

Max Degree

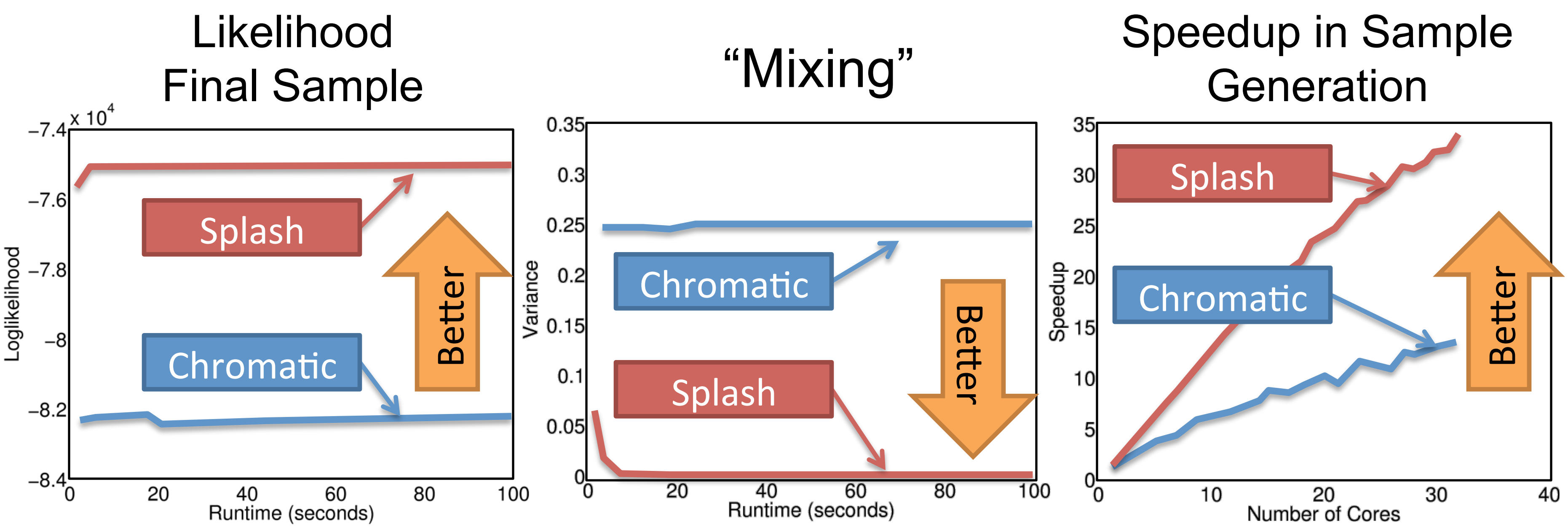
Processors

Variables

Representative Results

Markov logic network with **strong dependencies**

10K Variables *28K Factors*



The *Splash* sampler outperforms the *Chromatic* sampler on models with **strong** dependencies

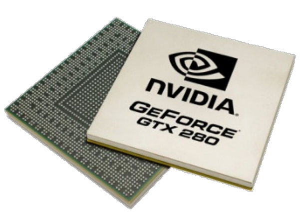
Massive Structured Problems

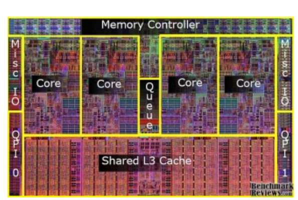
Graphical Representations

Parallel and **Distributed** Algorithms
for Probabilistic **Inference**

Graph-Parallel Systems: GraphLab

Advances Parallel Hardware

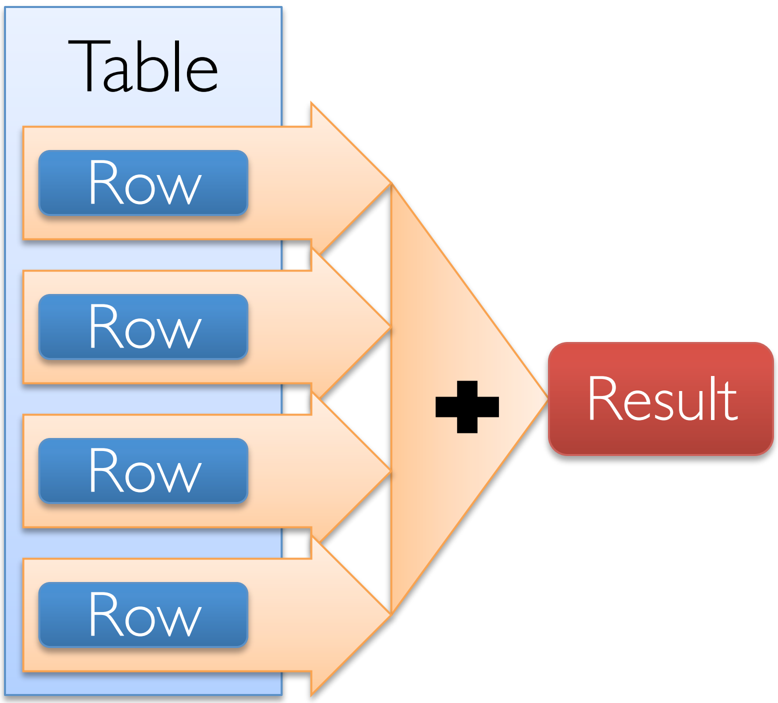




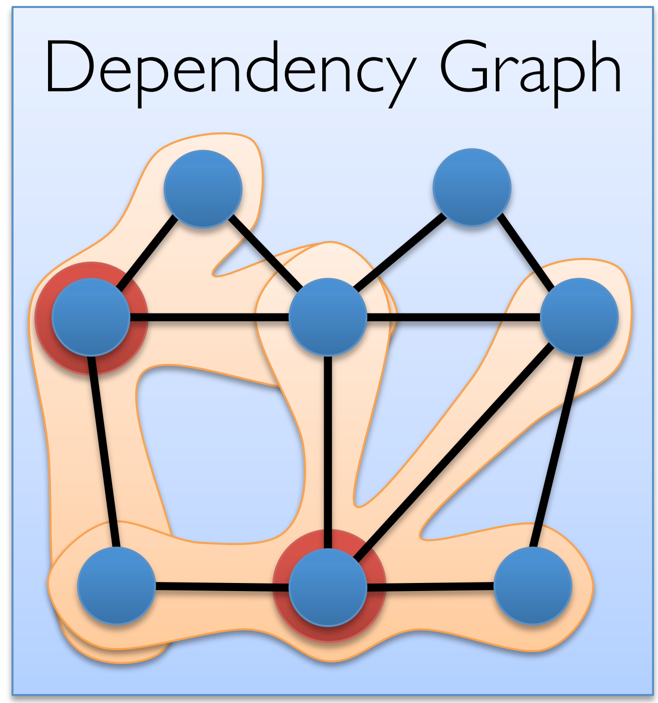
How do we **design** and **implement** **graph-parallel** inference algorithms?

Structure of Computation

Data-Parallel



Graph-Parallel



hadoop

Pregel

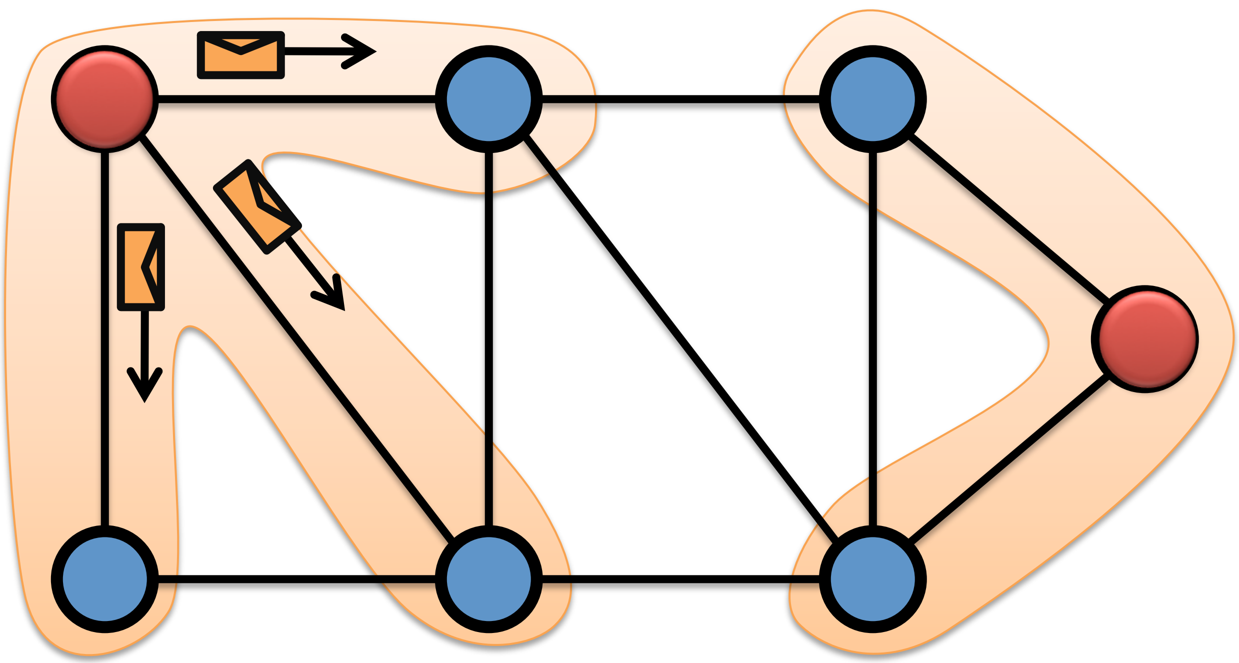
GraphLab

The **Graph-Parallel** Abstraction

A user-defined Vertex-Program runs on each vertex

Graph constrains **interaction** along edges

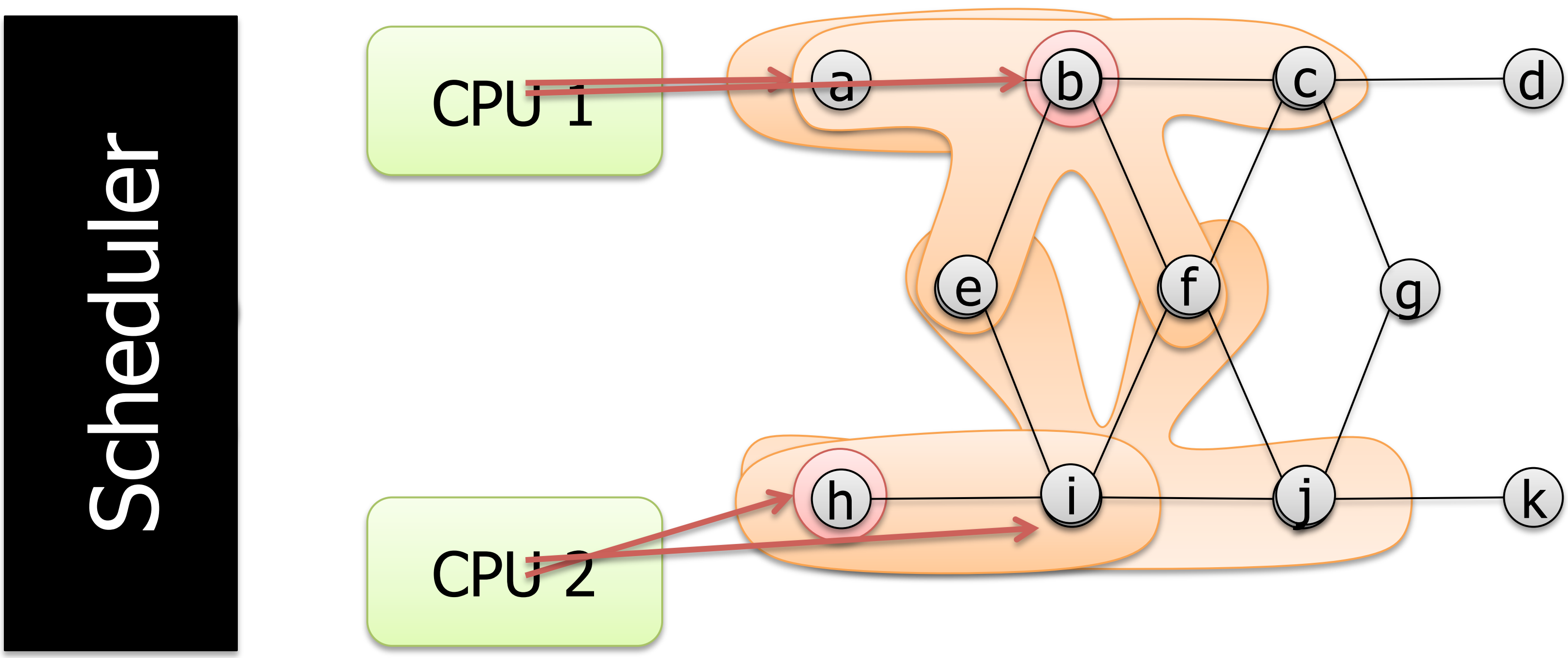
- Using **messages** (e.g. Pregel [PODC'09, SIGMOD'10])
- Through **shared state** (e.g., GraphLab [UAI'10, VLDB'12, OSDI'12])



Parallelism: run multiple vertex programs simultaneously

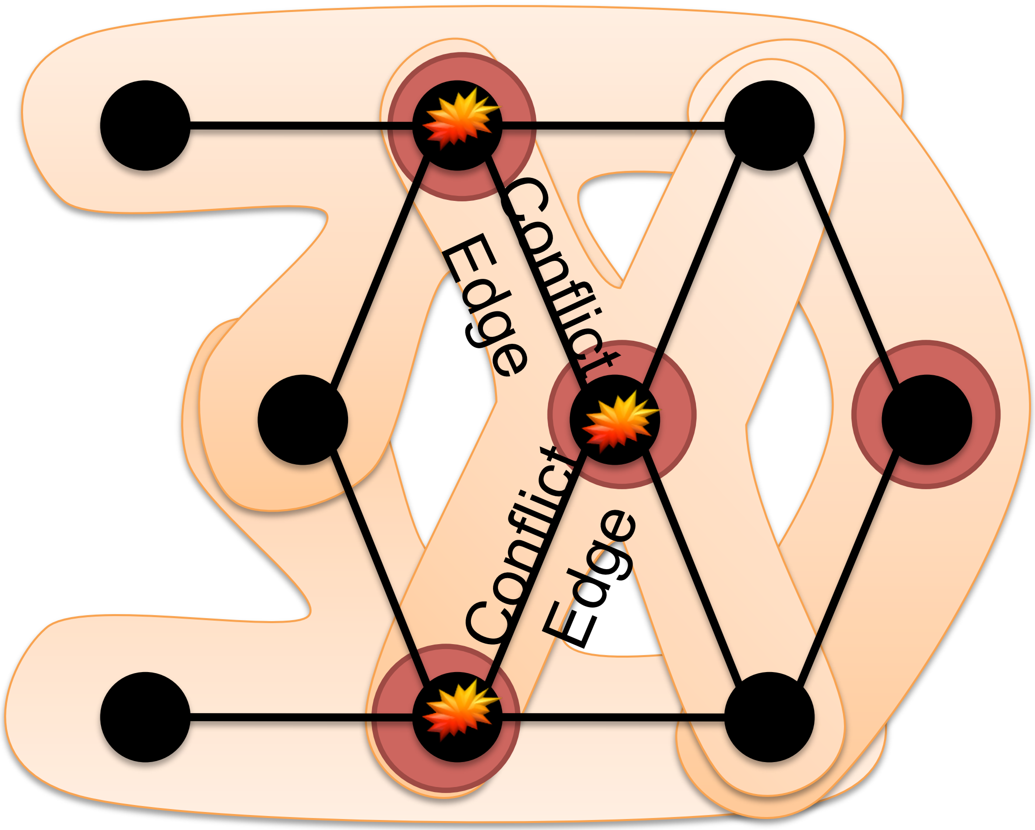
GraphLab **Asynchronous** Execution

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



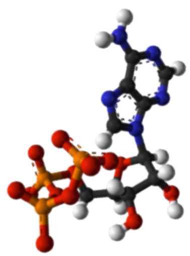
Scheduler can **prioritize** vertices.

GraphLab is **Serializable**



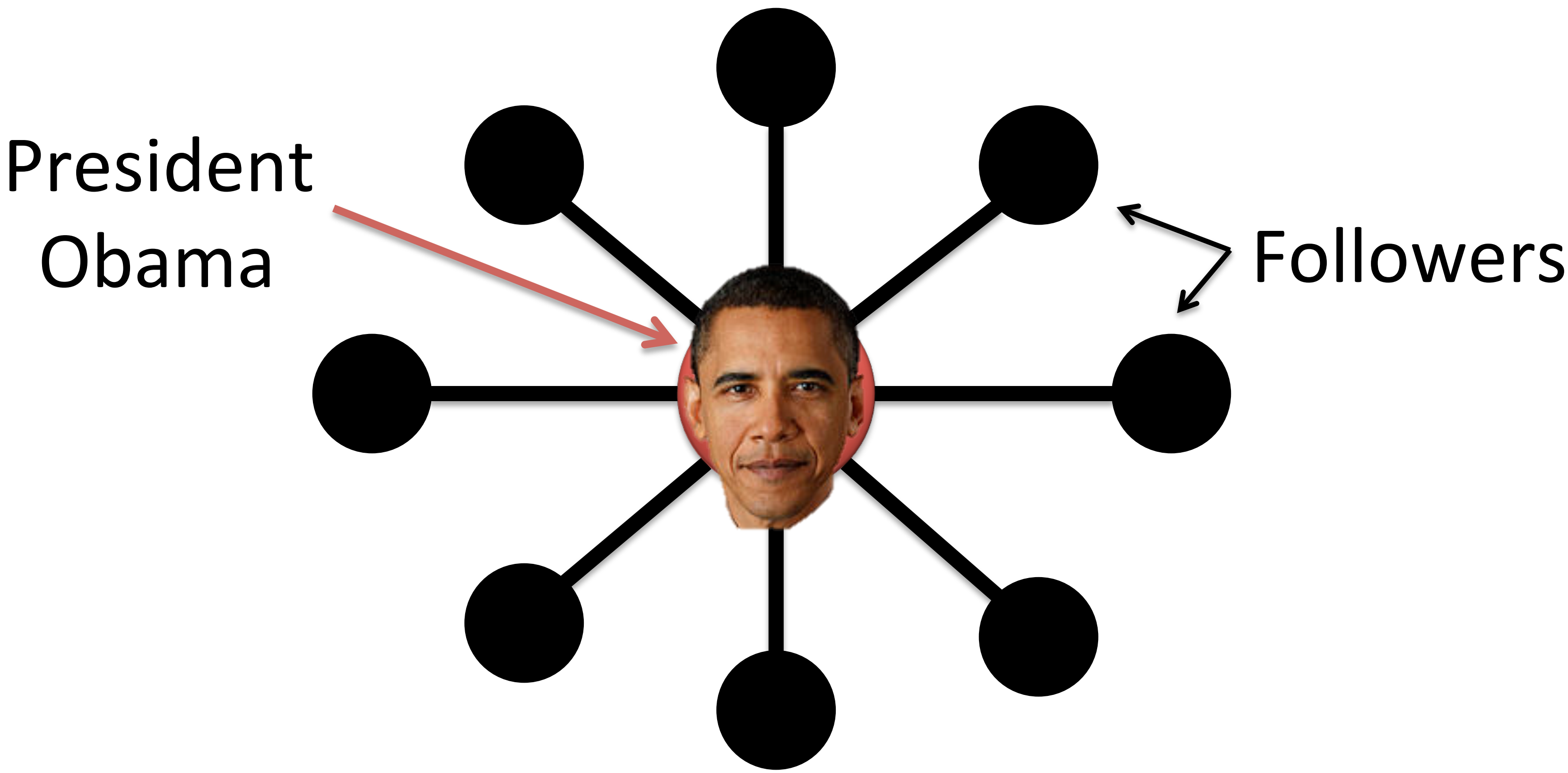
- Automatically ensures **serializable** executions

The Challenge of Power-Law Graphs

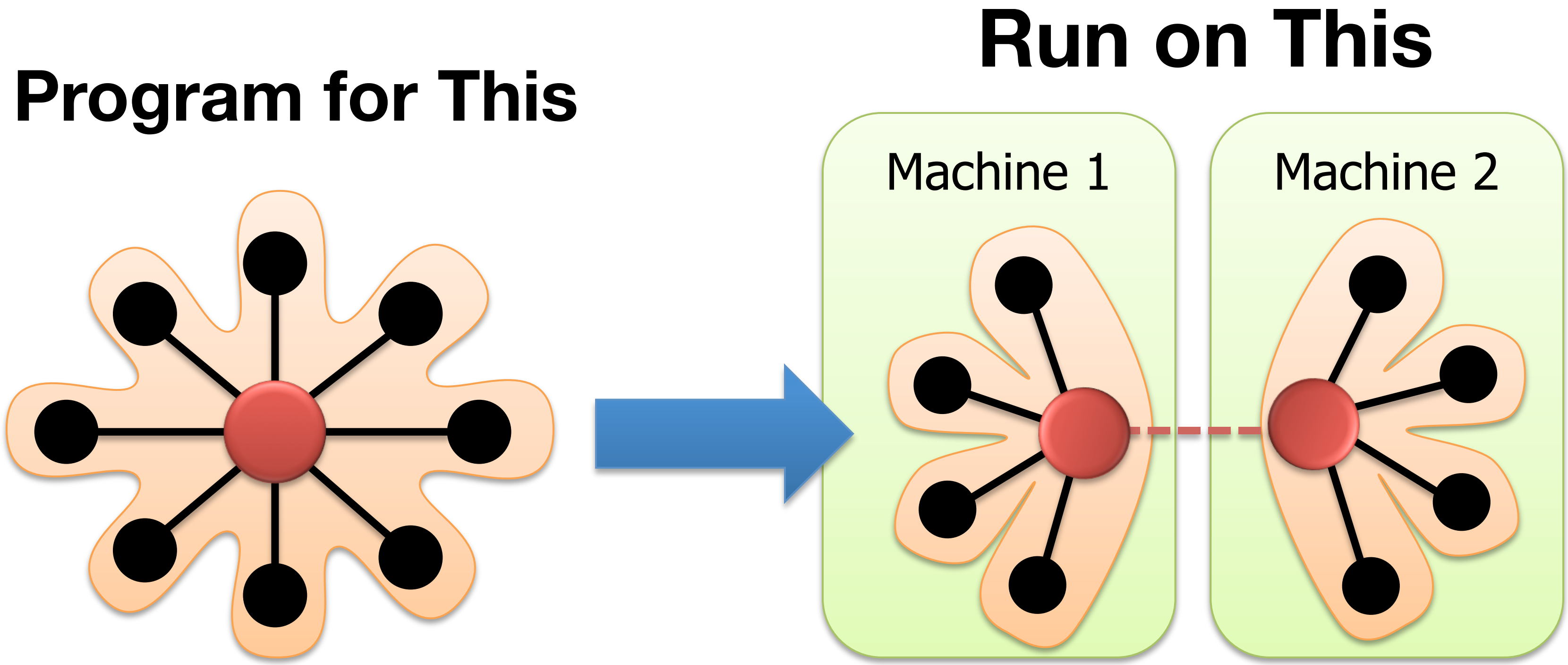


Power-Law Degree Distribution

“Star Like” Motif



GraphLab [OSDI'12]



Split **High-Degree** vertices

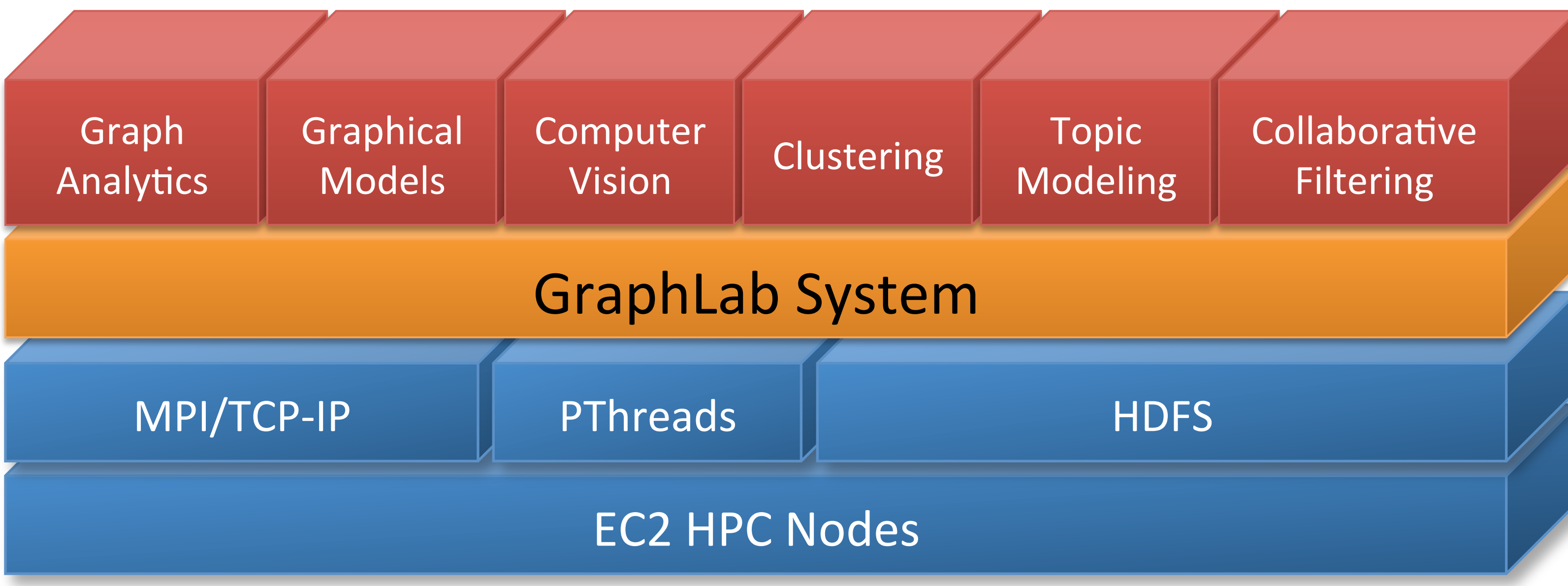
New Abstraction → *<u>Equivalence</u>* *on Split Vertices*

A **Common Pattern** for Vertex-Programs

```
GraphLab_Belief_Propagation( Vertex i )
```

Compute product of inbound messages	**Commutative Associative Agg.**
Update belief	**Vertex-Parallel**
Compute new outbound message	**Edge-Parallel Map Operation**

Machine Learning and **Data-Mining** Toolkits

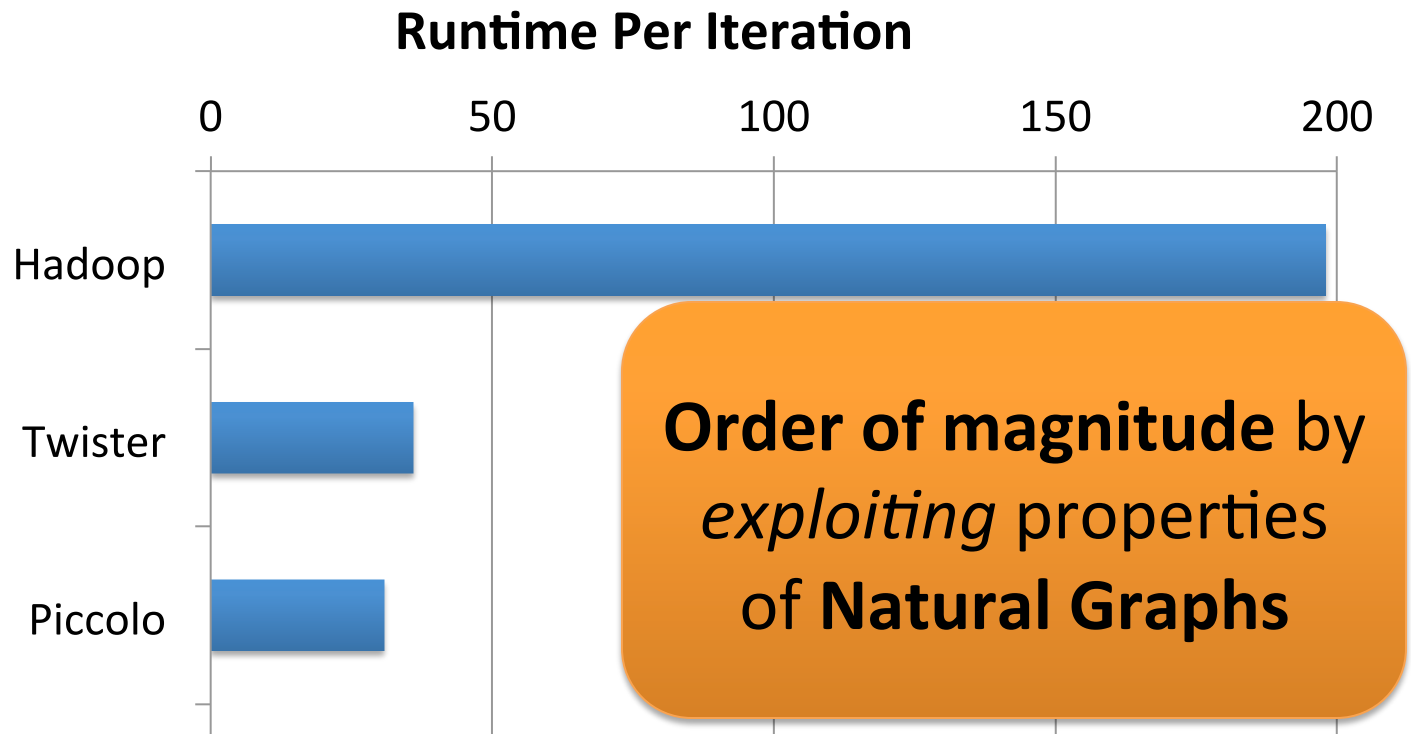


http://graphlab.org

Apache 2 License

PageRank on Twitter Follower Graph

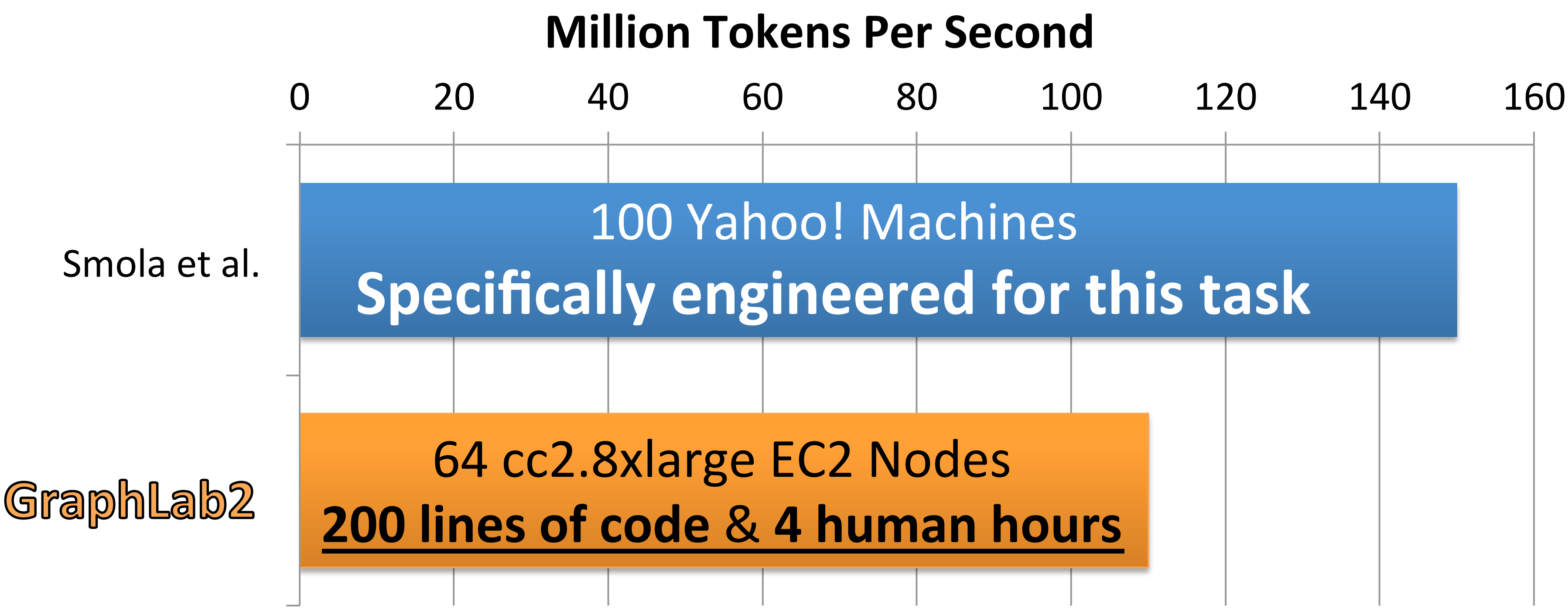
Natural Graph with 40M Users, 1.4 Billion Links



Hadoop results from [Kang et al. '11]
Twister (in-memory MapReduce) [Ekanayake et al. '10]

Gibbs Sampling for LDA

English language Wikipedia
- 2.6M Documents, 8.3M Words, 500M Tokens
- Computationally intensive algorithm



Triangle Counting on Twitter

40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles

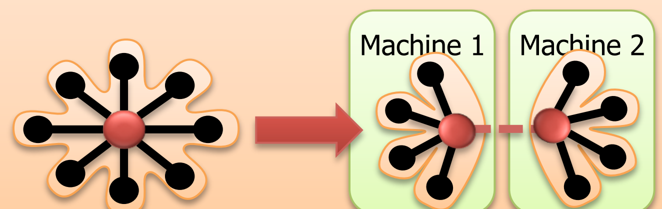
S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," WWW'11

GraphLab 2

By exploiting ***common patterns*** *in graph* ***data*** *and* ***computation****:*

New ways to **represent** real-world graphs

New ways **execute** graph algorithms



Orders of magnitude improvements over existing systems

Thank You

Joseph Gonzalez

Postdoc, UC Berkeley AMPLab
jegonzal@eecs.berkeley.edu

Co-Founder GraphLab Inc,
joseph@graphlab.com

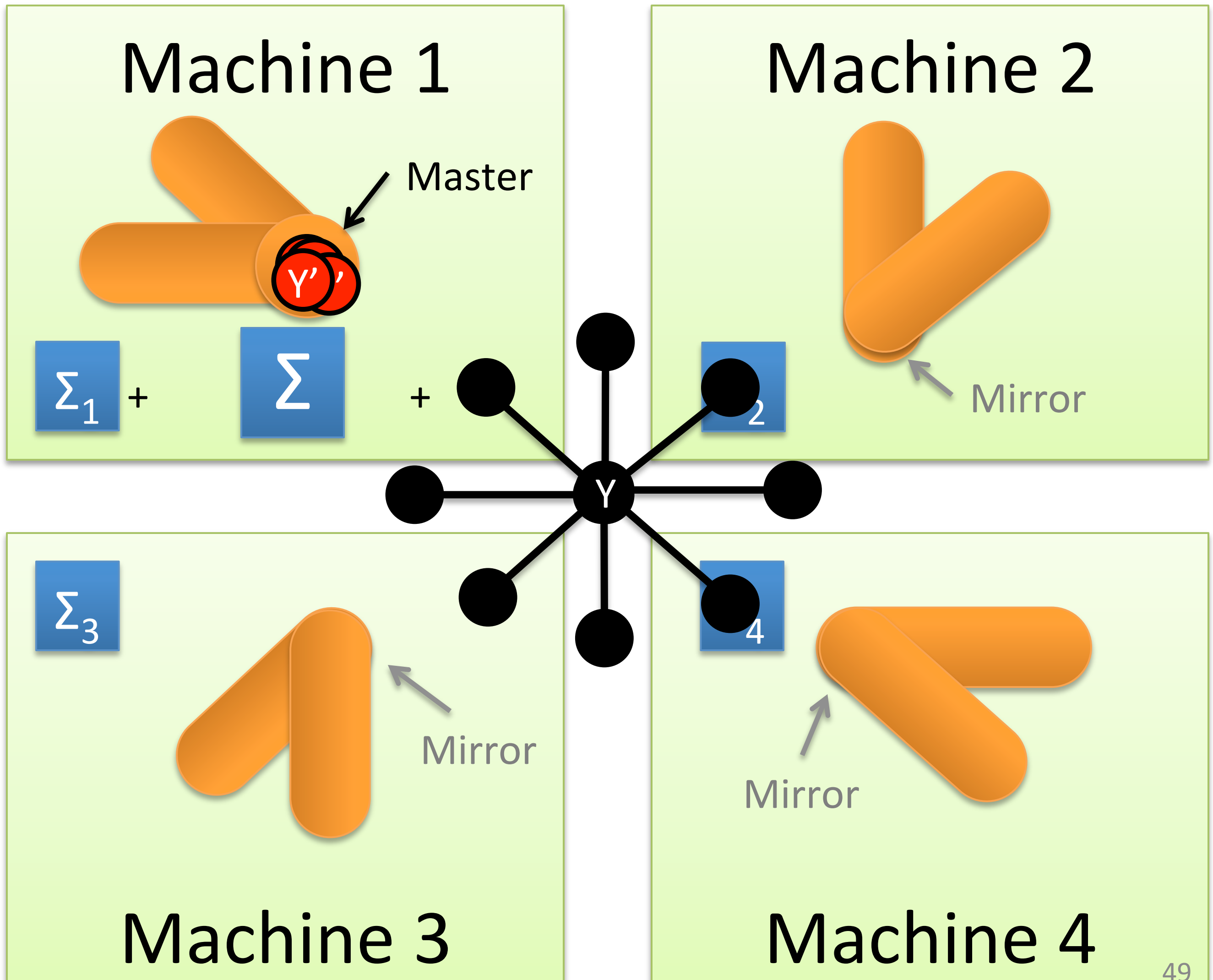
Checkout the NIPS http://biglearn.org Workshop on December 9th in Tahoe

GAS Decomposition

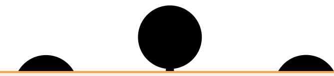
Gather

Apply

Scatter



Minimizing Communication in PowerGraph



New Theorem:

*For **any edge-cut** we can directly construct a vertex-cut which requires **strictly less** communication and storage.*

*Percolation theory suggests that power law graphs have **good vertex cuts**. [Albert et al. 2000]*

Constructing Vertex-Cuts

- **Evenly** assign **edges** to machines
  - Minimize machines spanned by each vertex
- Assign each edge **as it is loaded**
  - Touch each edge only once
- Propose two **distributed** approaches:
  - ***Random*** *Vertex Cut*
  - ***Greedy*** *Vertex Cut*

Random Vertex-Cut

- Randomly assign edges to machines

Machine 1

Machine 2

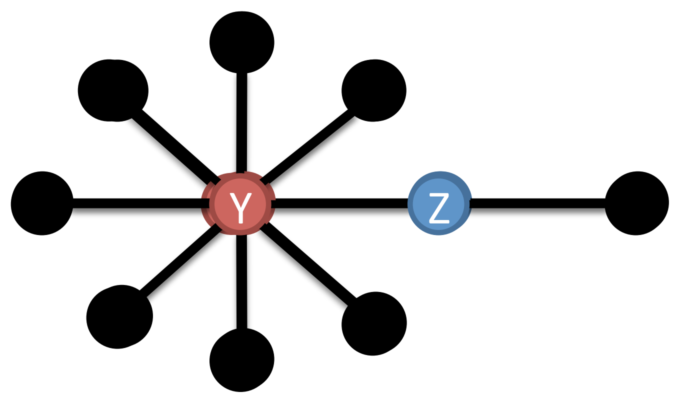
Machine 3

Balanced Vertex-Cut

Y Spans 3 Machines

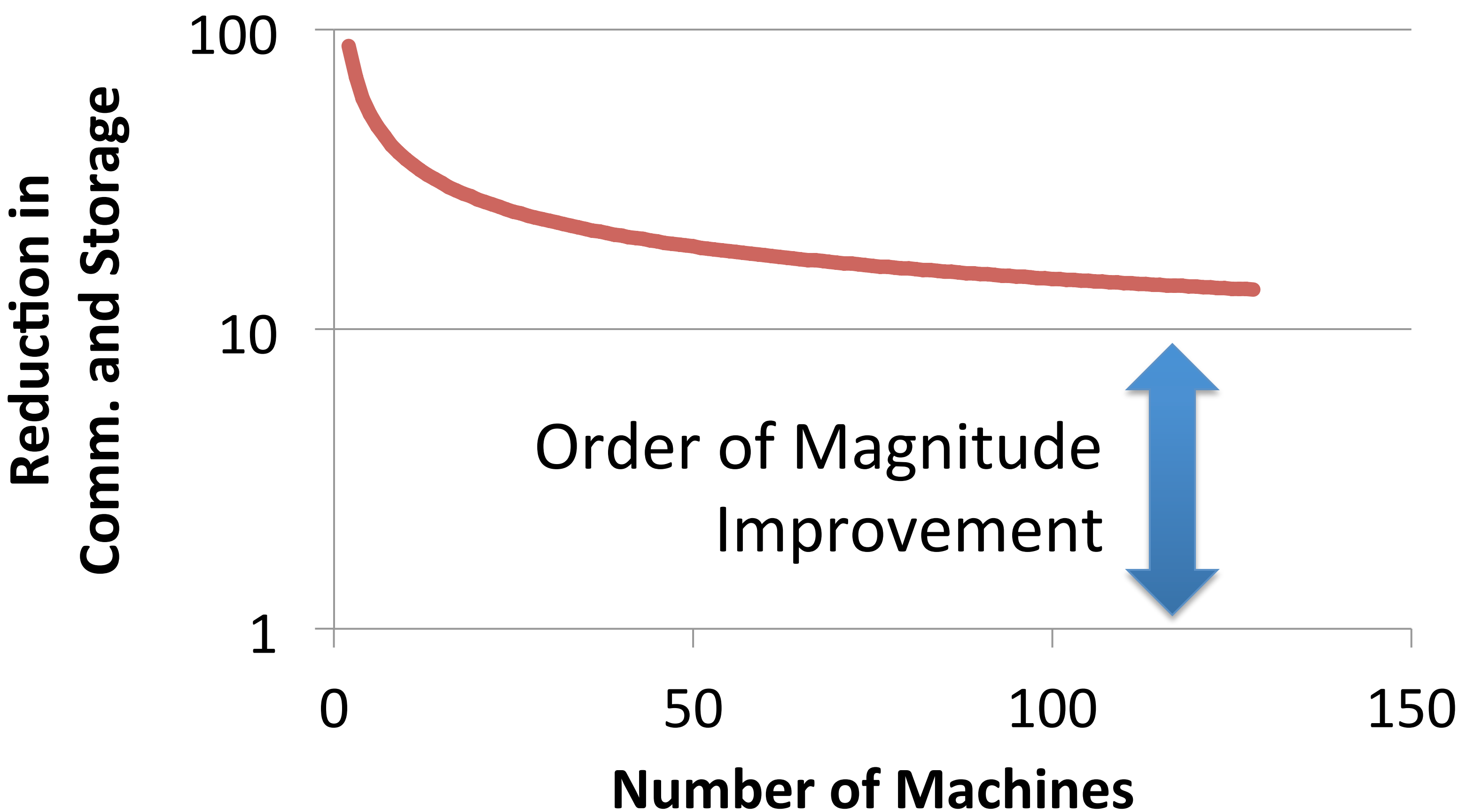
Z Spans 2 Machines

Not cut!



Random Vertex-Cuts vs. Edge-Cuts

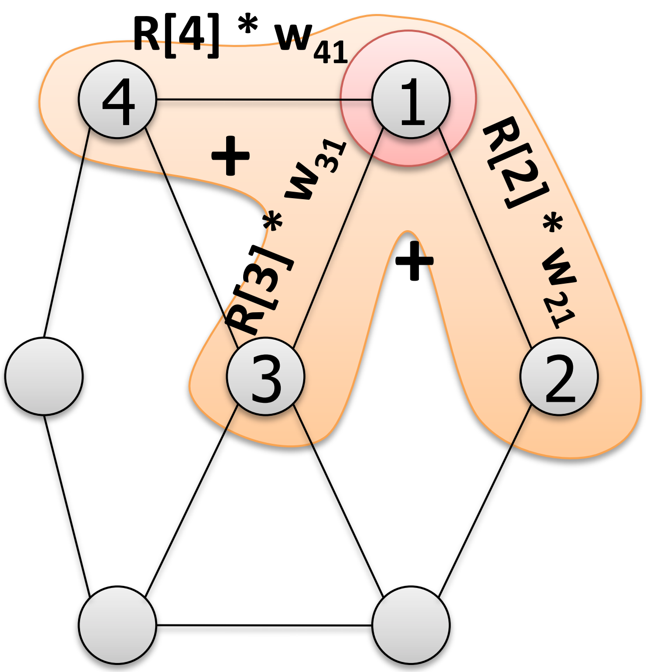
- Expected improvement from vertex-cuts:



The GraphLab Vertex Program

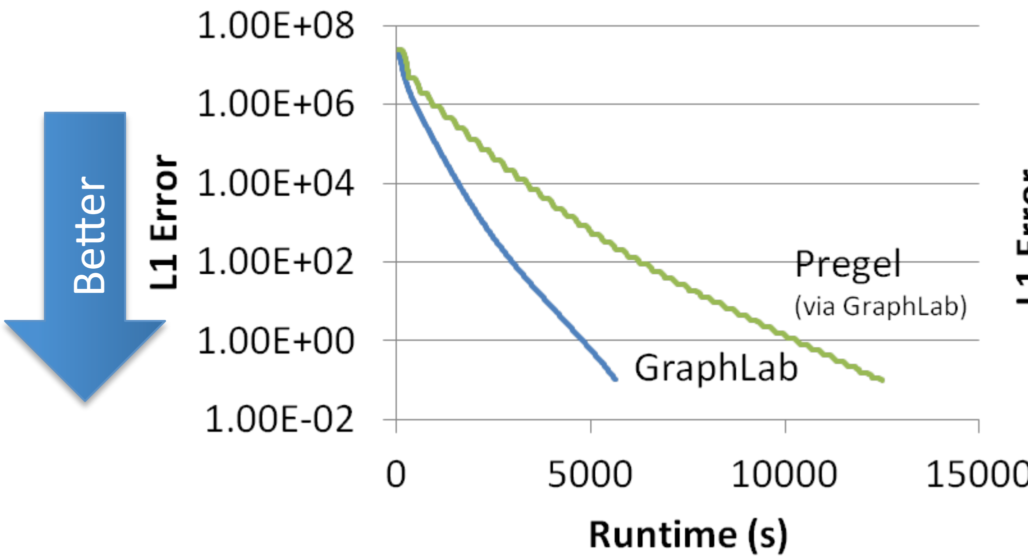
Vertex Programs directly **access** adjacent vertices and edges

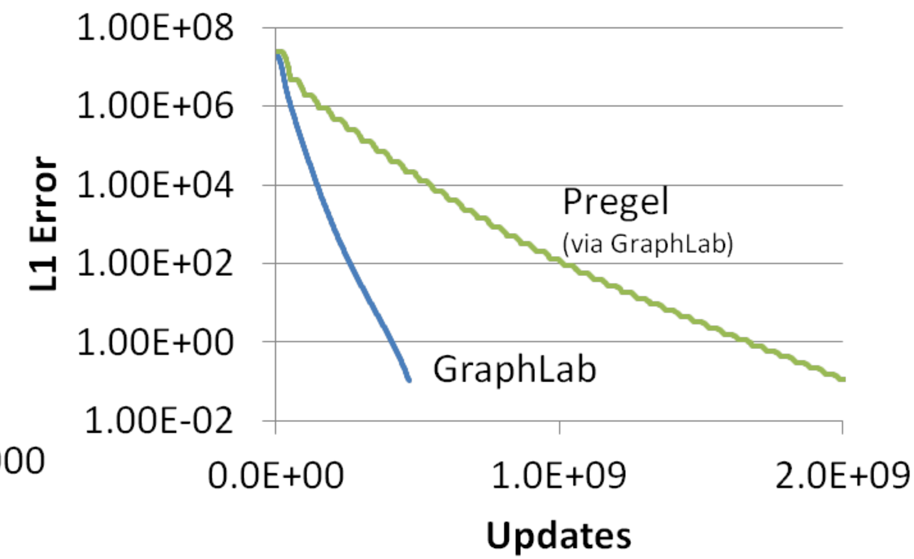
```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total

  // Trigger neighbors to run again
  if R[i] not converged then
   signal nbrsOf(i) to be recomputed
```

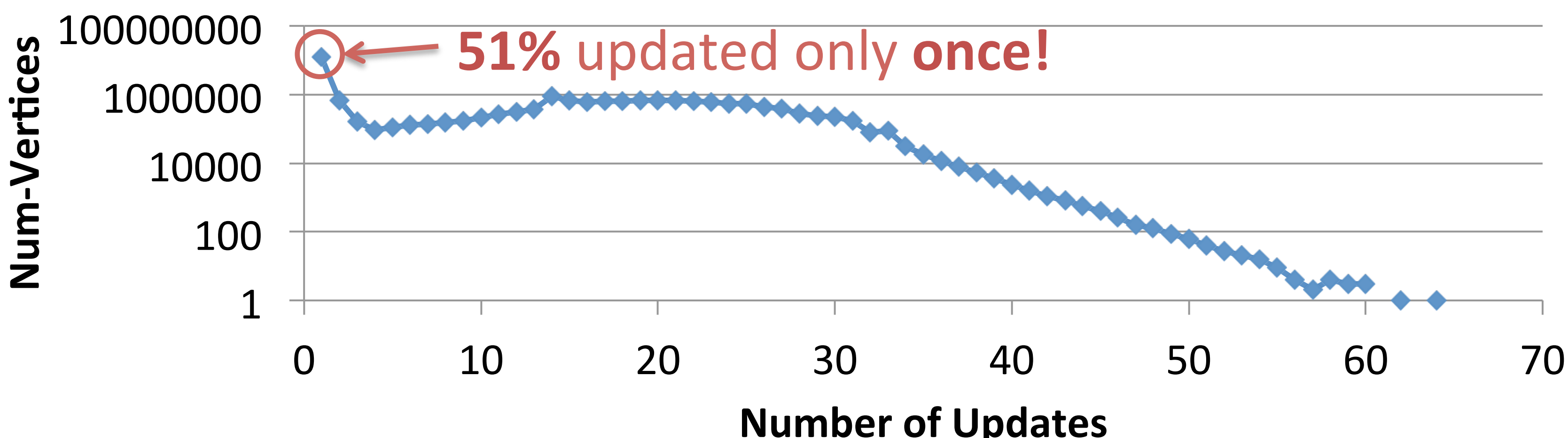


Signaled vertices are recomputed **eventually.**

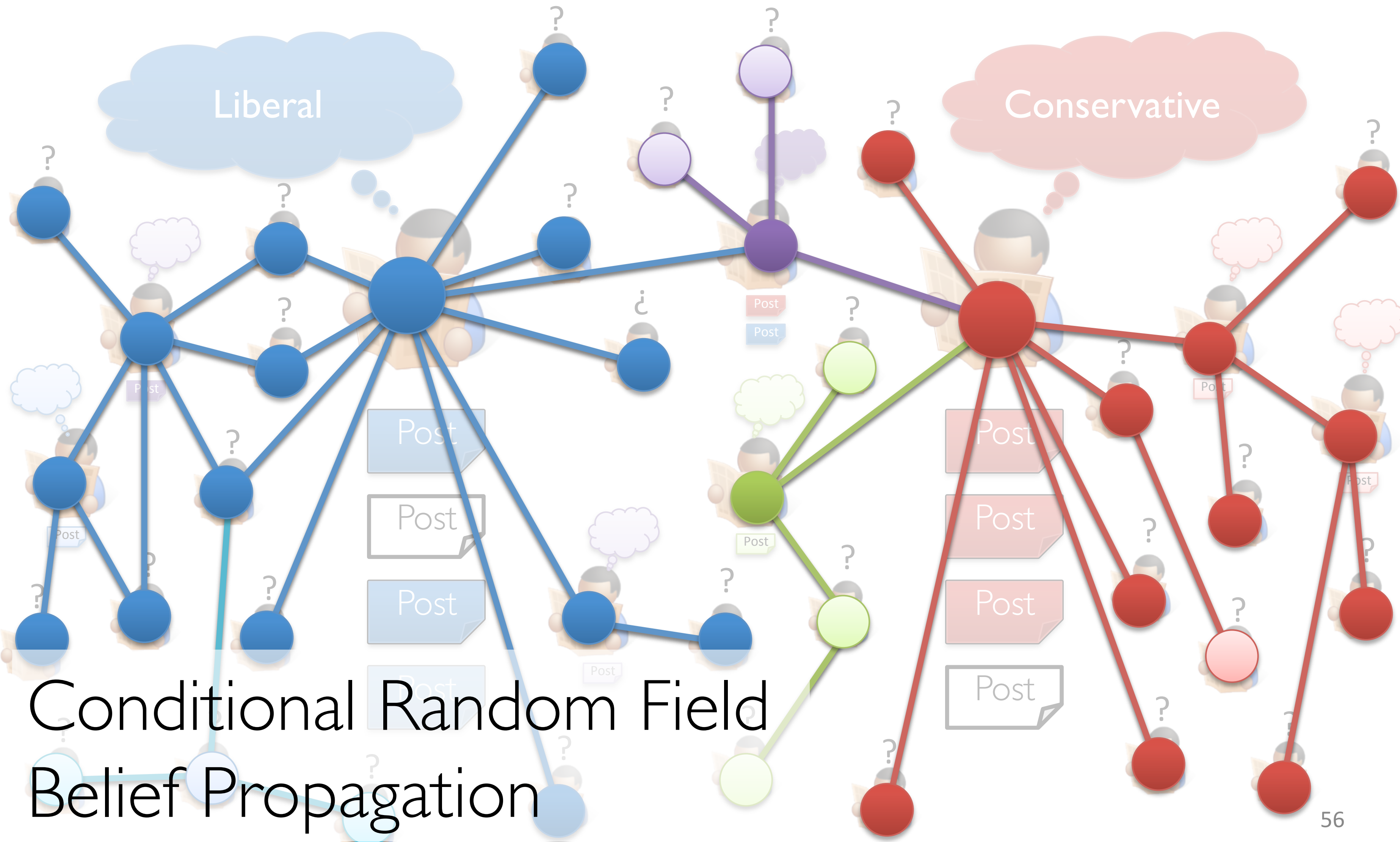
Convergence of Dynamic PageRank







Predicting Political Bias



Conditional Random Field
Belief Propagation