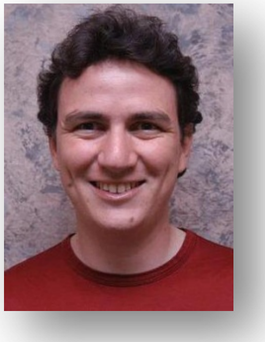
Thesis Defense

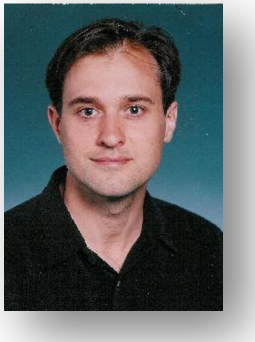
Parallel and Distributed Systems for Probabilistic Reasoning

Joseph E. Gonzalez

Thesis Committee:



Carlos Guestrin
University of Washington & CMU



Guy Blelloch
CMU

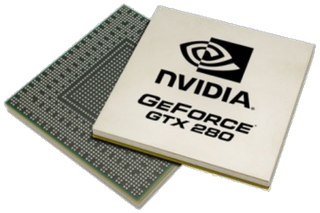
David O'Hallaron
CMU

Alex Smola
CMU & Google

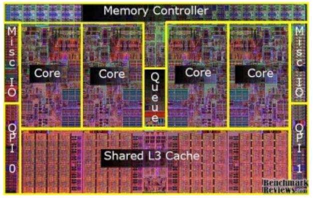
Jeff Bilmes
University of Washington

*The foundations of **computation** have changed …*

New Parallel and Distributed Platforms



GPUs



Multicore

Clusters

Single Chip
Cloud Computers

Clouds

- New Opportunities
  - Increased processing and storage
- New Challenges
  - Parallel algorithm design and implementation

The ***scale*** *of*

machine learning problems

is ***exploding*** *…*

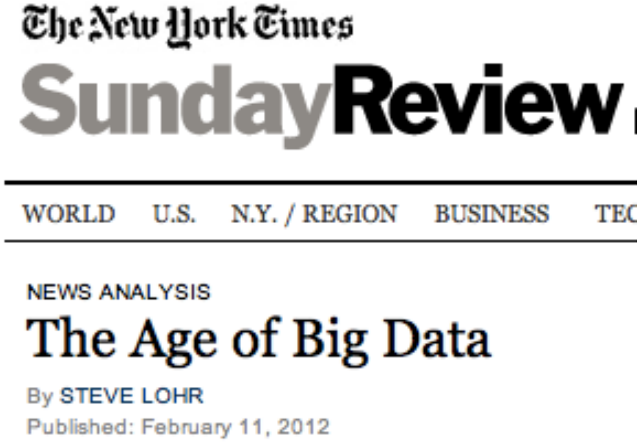
The Age of Big Data

28 Million
Wikipedia Pages

1 Billion
Facebook Users

6 Billion
Flickr Photos

72 Hours a Minute
YouTube



The New York Times

SundayReview

WORLD U.S. N.Y. / REGION BUSINESS TEC

NEWS ANALYSIS

The Age of Big Data

By STEVE LOHR

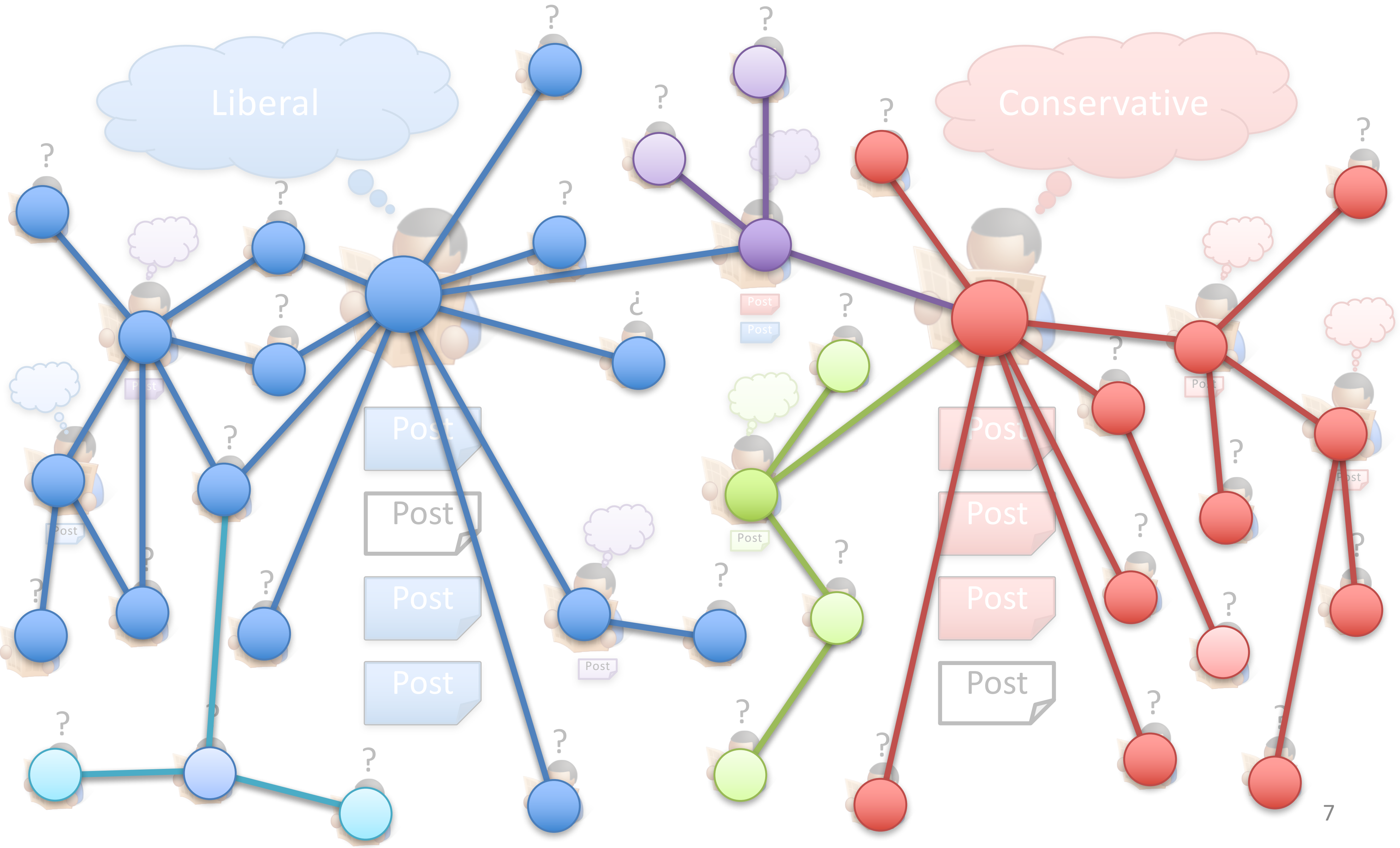
Published: February 11, 2012

"…growing at 50 percent a year…"

"… data a new class of economic asset, like currency or gold."

Massive data provides opportunities for ***structured models****…*

Example: *Estimate Political Bias*



facebook flickr YouTube

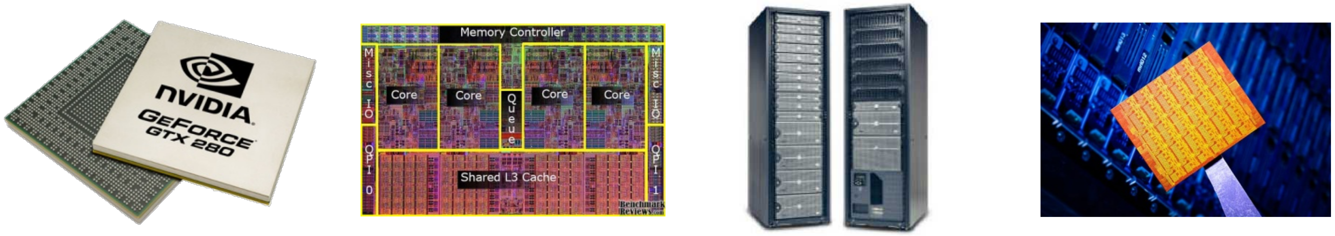
Massive Structured Problems

Thesis:

?

Parallel and Distributed Systems for Probabilistic Reasoning

Advances Parallel Hardware



amazon web services™

Thesis Statement: **GrAD Methodology**

*Efficient **parallel** and **distributed** systems for probabilistic reasoning:*

1. **Graphically** decompose *computational* and *statistical* dependencies

2. **Asynchronously** schedule computation

3. **Dynamically** identify and *prioritize* computation along critical paths

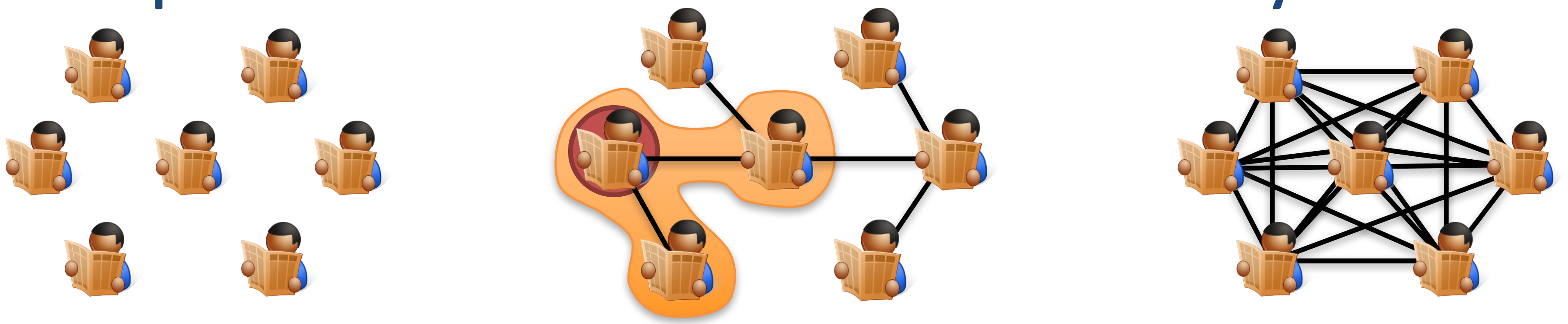
GrAD Methodology: **Graphical**

- Factor **statistical** and **computational** dependencies

Independent

Sparse Graph

Fully Connected



Overly Simplistic

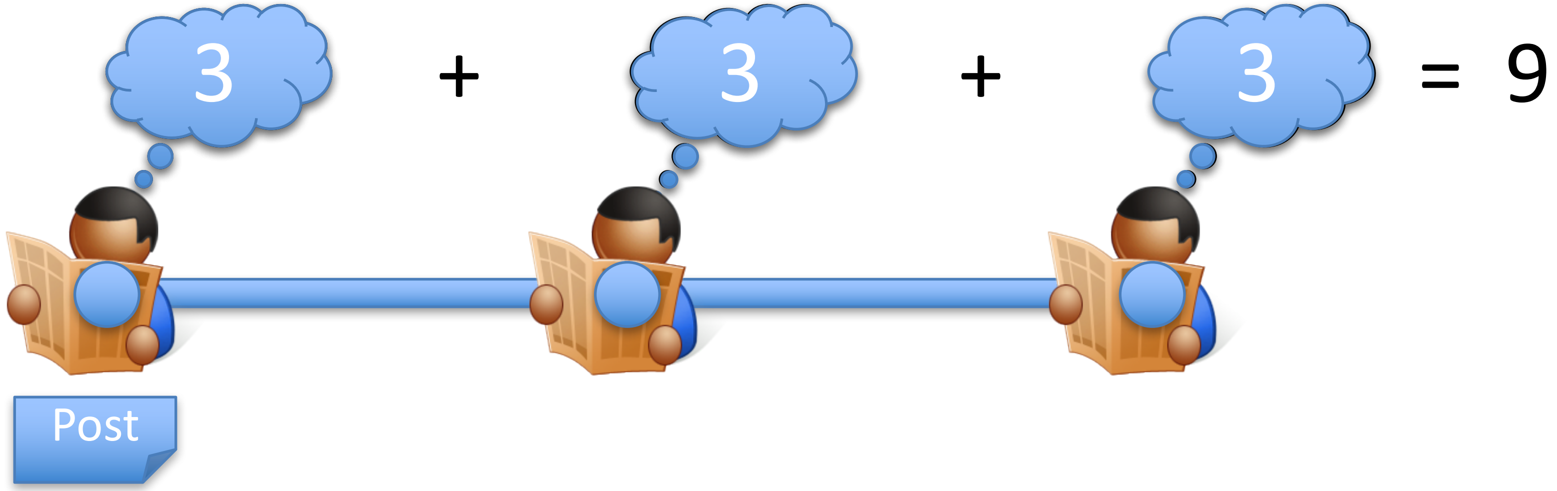
Expressive Tractable Models & Algorithms

Intractable

- Improves **computational** and **statistical** efficiency
- Increases **parallelism**

Synchronous vs. Asynchronous

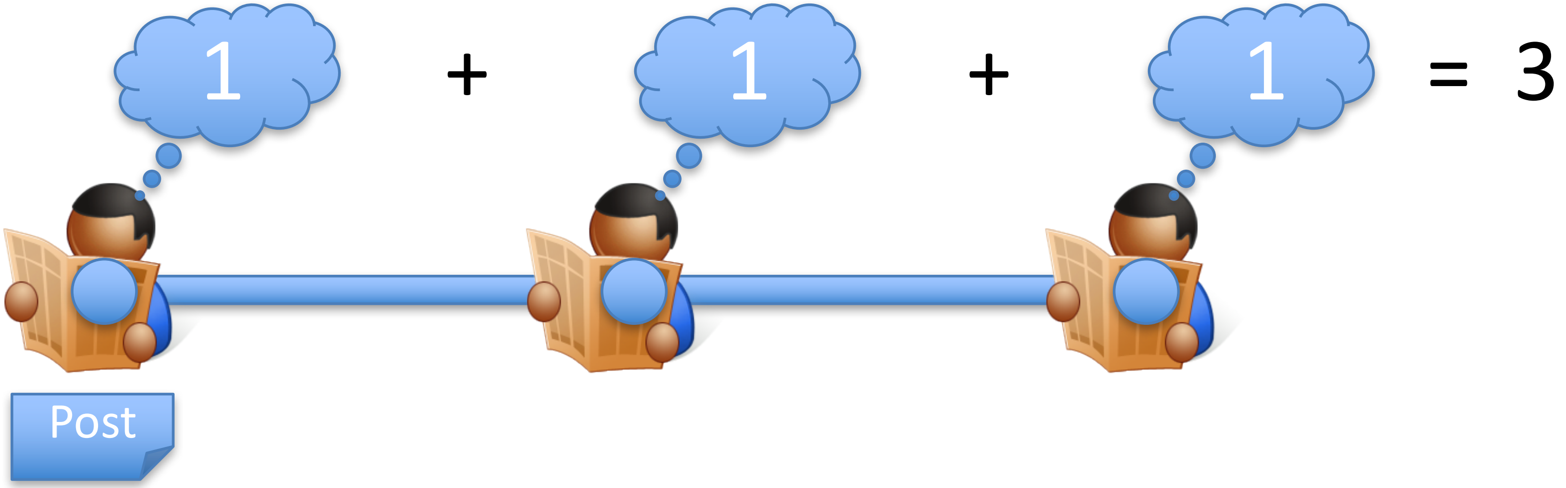
- **Synchronous:** *compute everything in parallel*



- Highly **parallel** – Maximum independent work
- Highly **inefficient** – Many wasted cycles

GrAD Methodology: **Asynchronous**

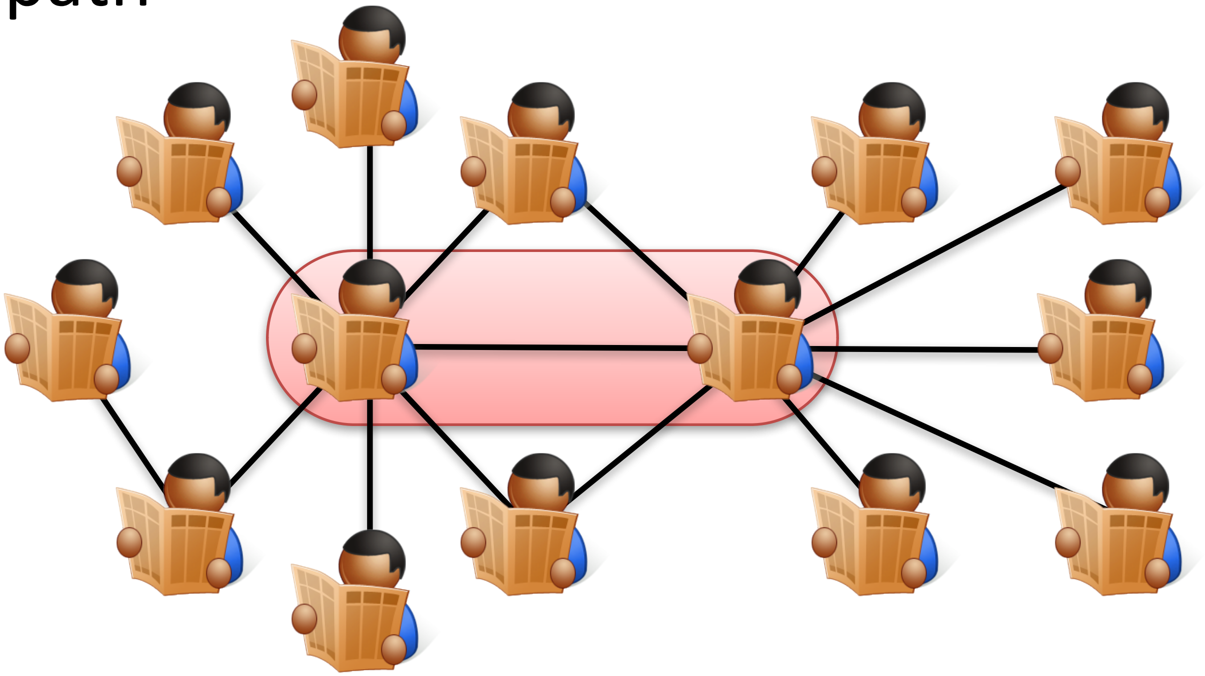
- Trigger computation as **new information arrives**



- Capture the flow of information:
  - More efficiently use network and processor resources
  - Guarantee algorithm correctness

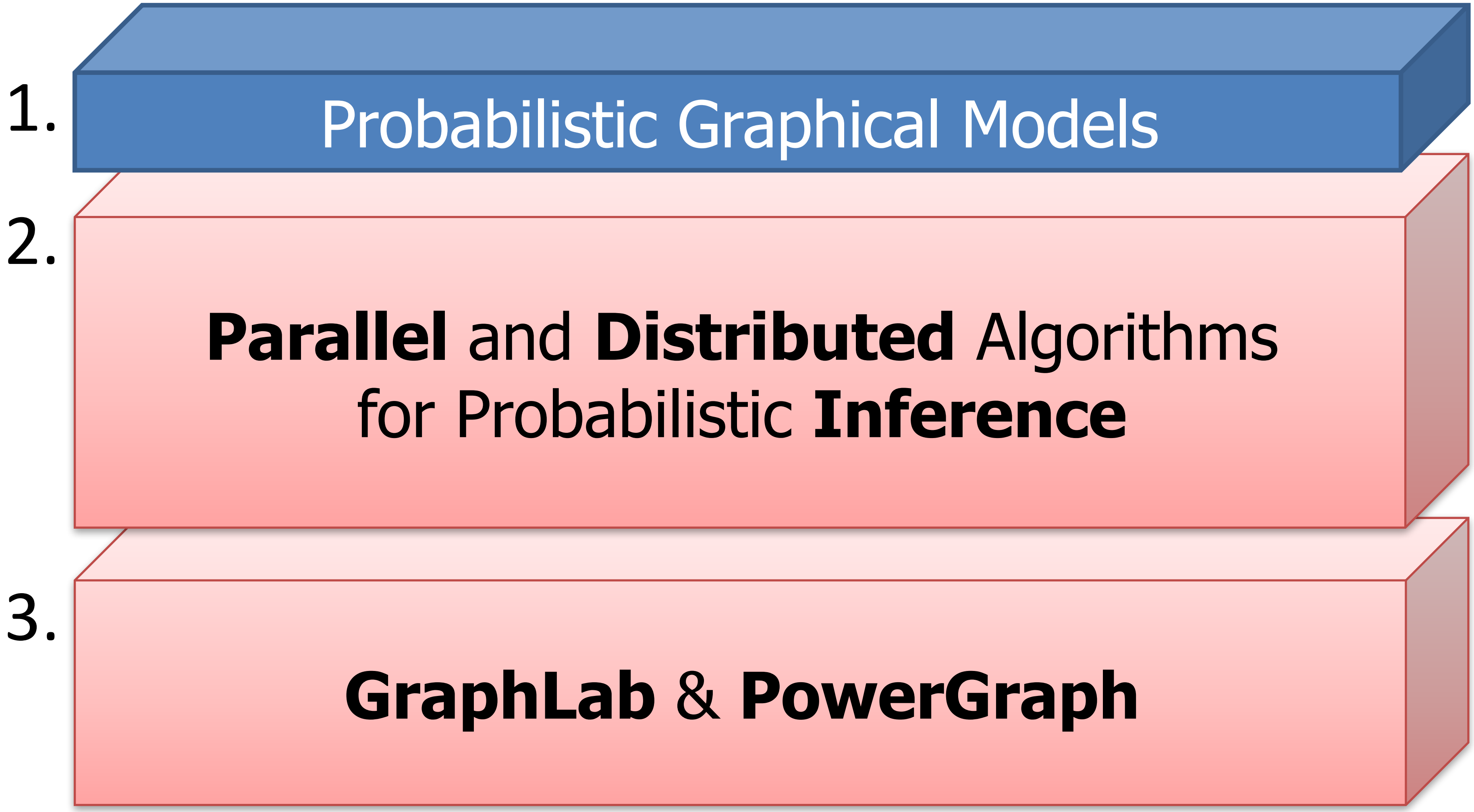
GrAD Methodology: **Dynamic**

- Dynamically **identify** and **prioritize** computation along the critical path



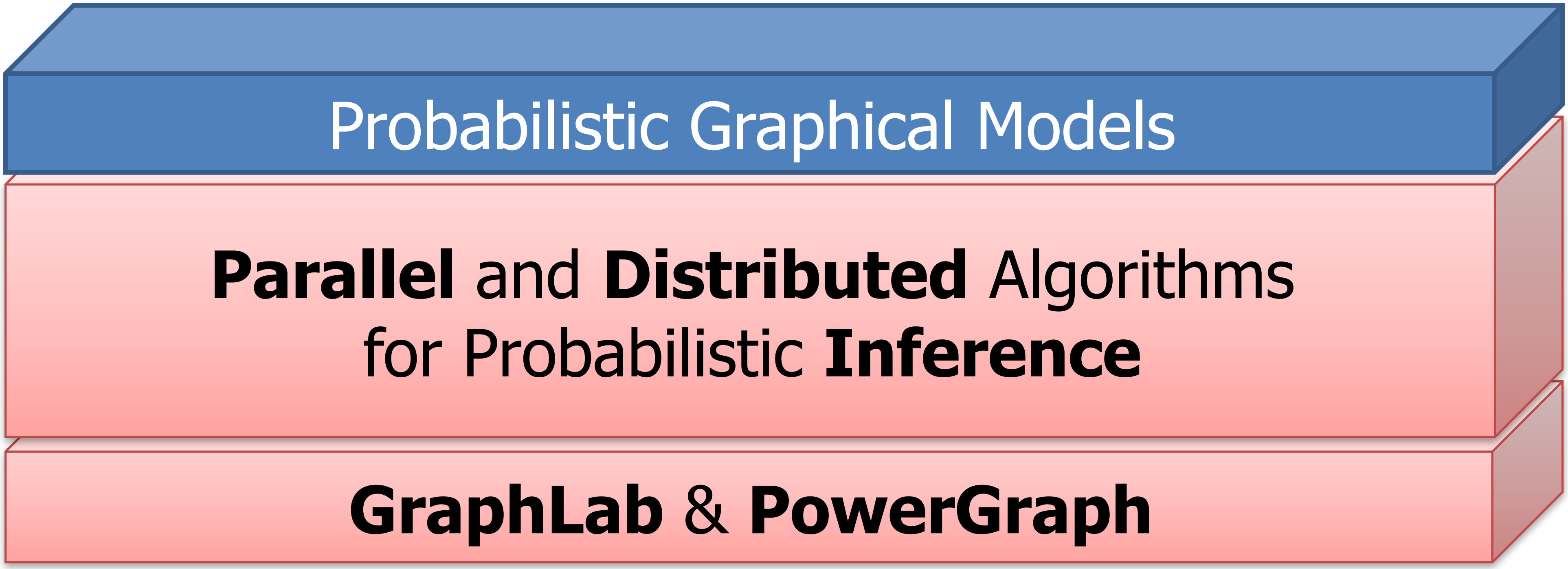
- Focus computational resources where most effective:
  - Accelerated convergence
  - Increased work efficiency

We apply the GrAD methodology to



1. Probabilistic Graphical Models
2. **Parallel** and **Distributed** Algorithms for Probabilistic **Inference**
3. **GraphLab** & **PowerGraph**

Massive Structured Problems

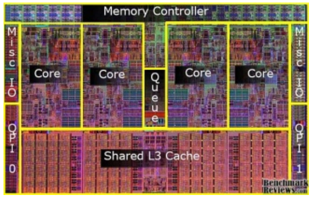


Probabilistic Graphical Models

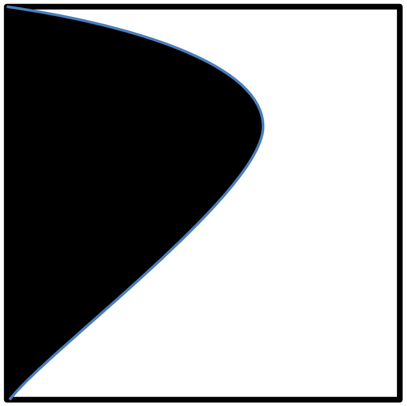
Parallel and **Distributed** Algorithms for Probabilistic **Inference**

GraphLab & **PowerGraph**

Advances Parallel Hardware

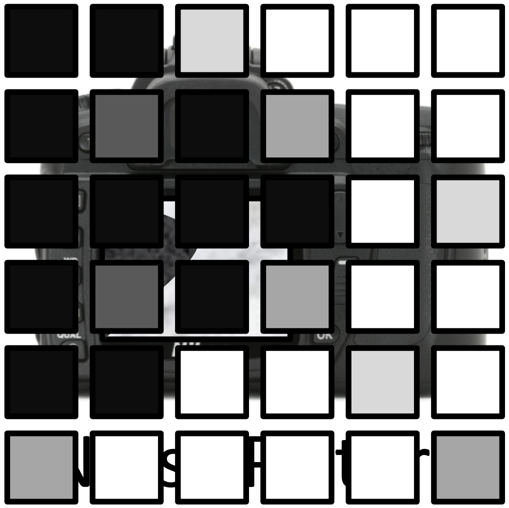


Encode **Probabilistic Structure**



True Image

Noisy Pixels



Random Variables

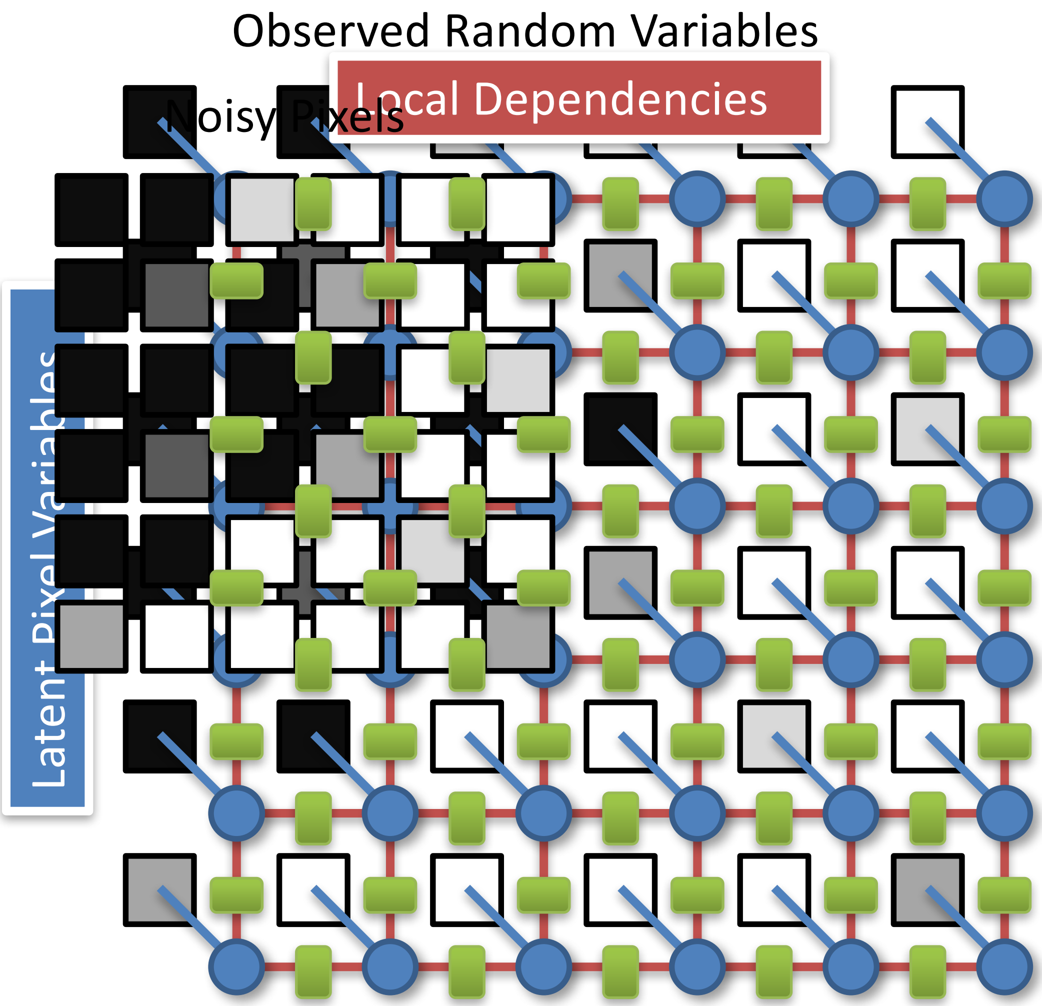
True *unobserved* values

Dependency Graph:

Represent dependencies

Parameters:

Characterize probabilities



Observed Random Variables

Noisy Pixels

Local Dependencies

Latent Pixel Variables

$$\mathbf{P}(\underbrace{X_1,\ldots,X_n}_{\text{Joint Probability}};\theta) \propto \underbrace{\prod_{(u,v)\in E}}_{\text{Graph}} f(X_u,X_v;\underbrace{\theta_{u,v}}_{\text{Factors}})$$

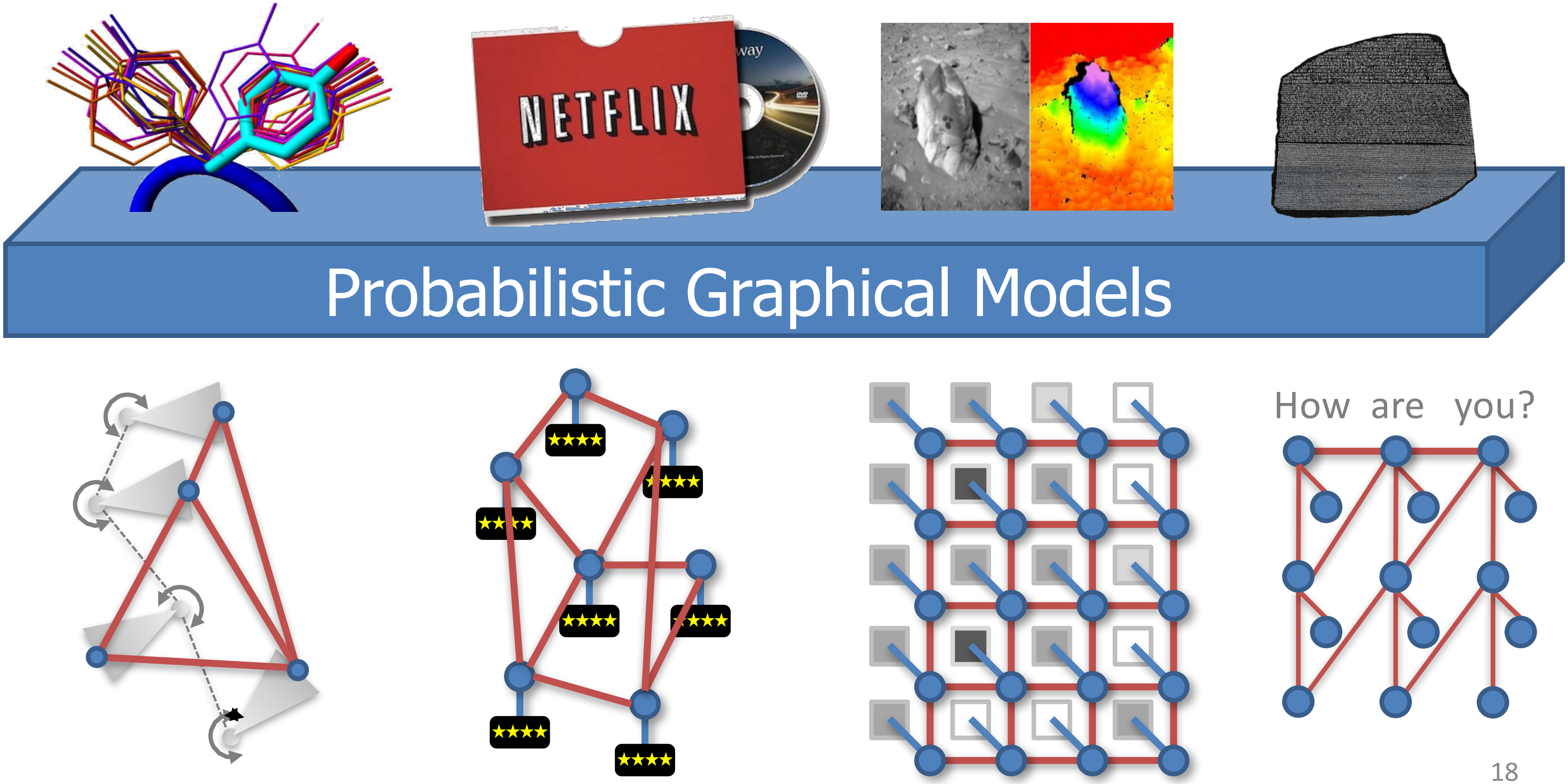
Graphical models provide a **common representation**

Protein Structure Prediction

Movie Recommendation

Computer Vision

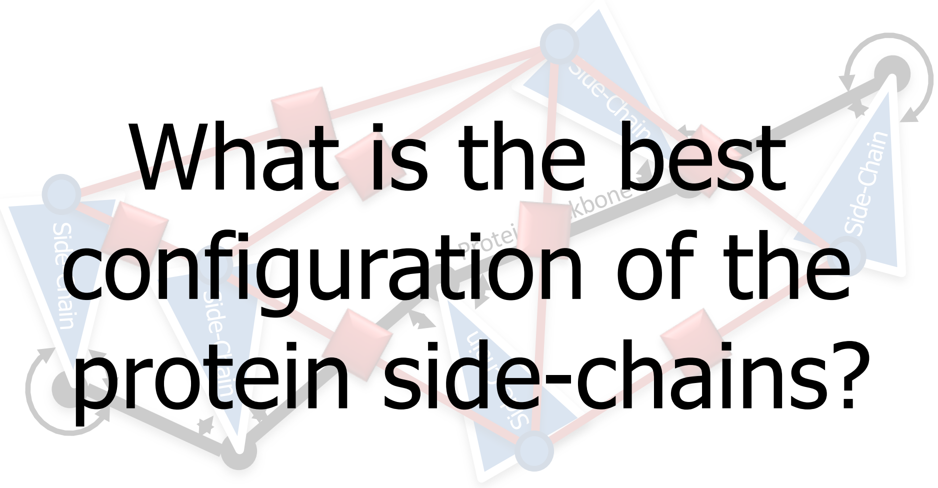
Machine Translation



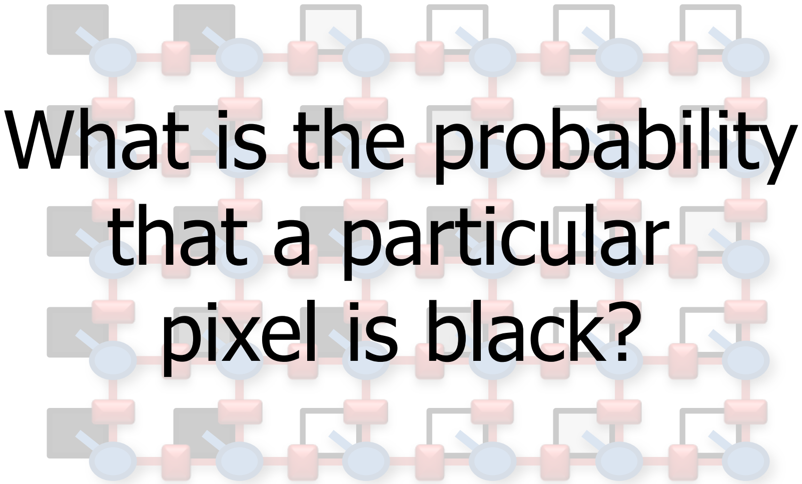
Probabilistic Graphical Models

Probabilistic **Inference**

*Making **predictions** given the model structure and parameters*



What is the best configuration of the protein side-chains?

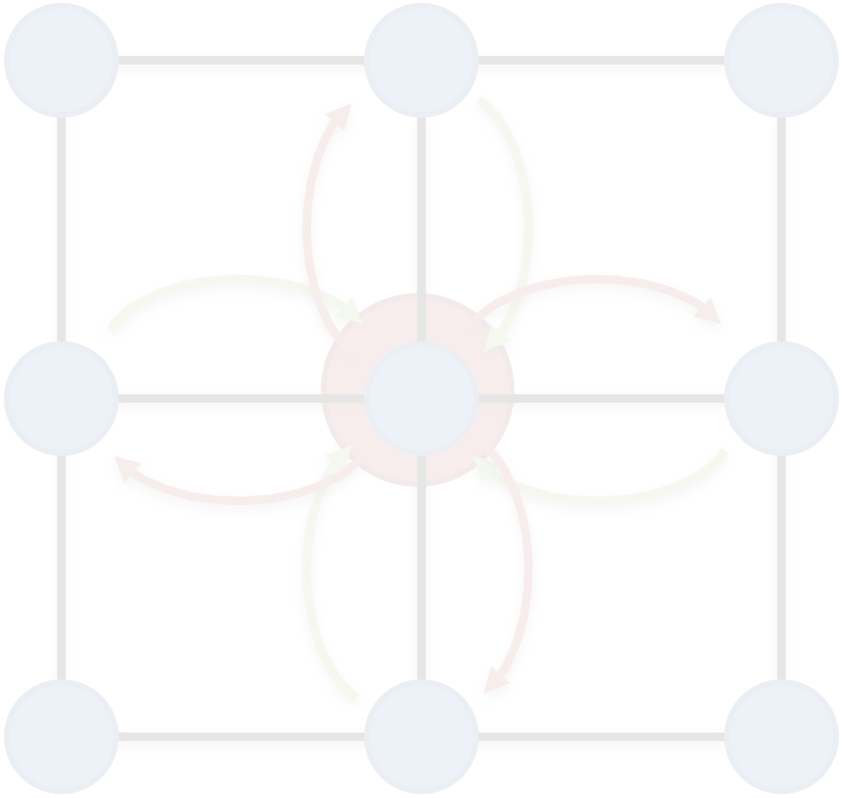


What is the probability that a particular pixel is black?

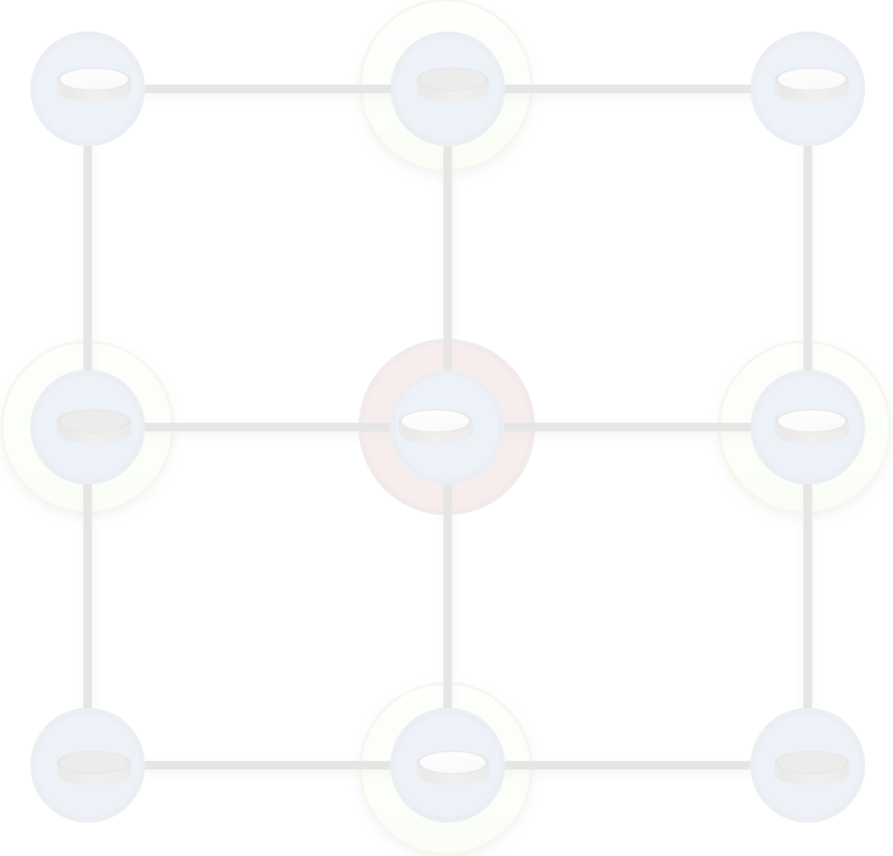
- **NP-complete** in general
  - Focus on *approximate* methods

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

Belief Propagation



Gibbs Sampling



Parallel Belief Propagation

Joint Work With:

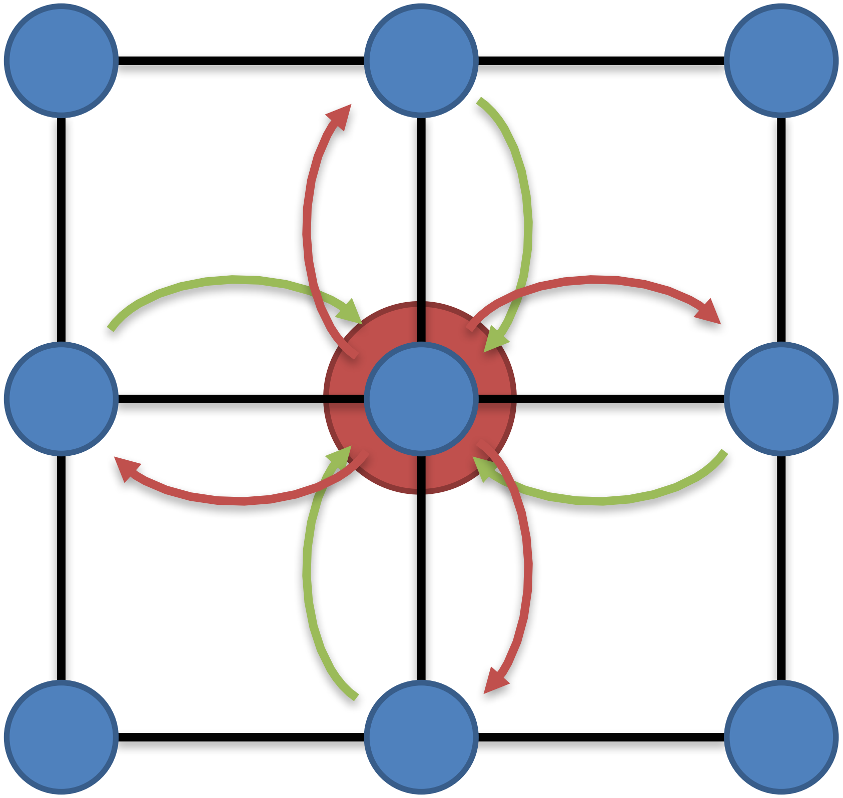
Yucheng Low ***Carlos Guestrin*** ***David O'Hallaron***

Published Results

AISTATS'09 **UAI'09** **Chapter in SUML'10**

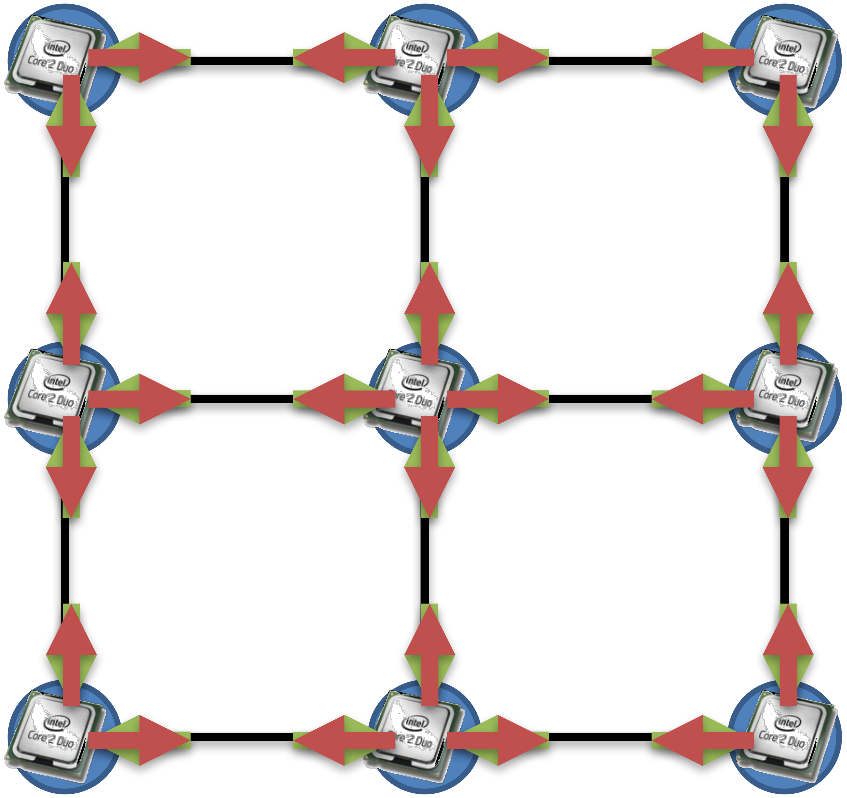
Loopy Belief Propagation (Loopy BP)

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



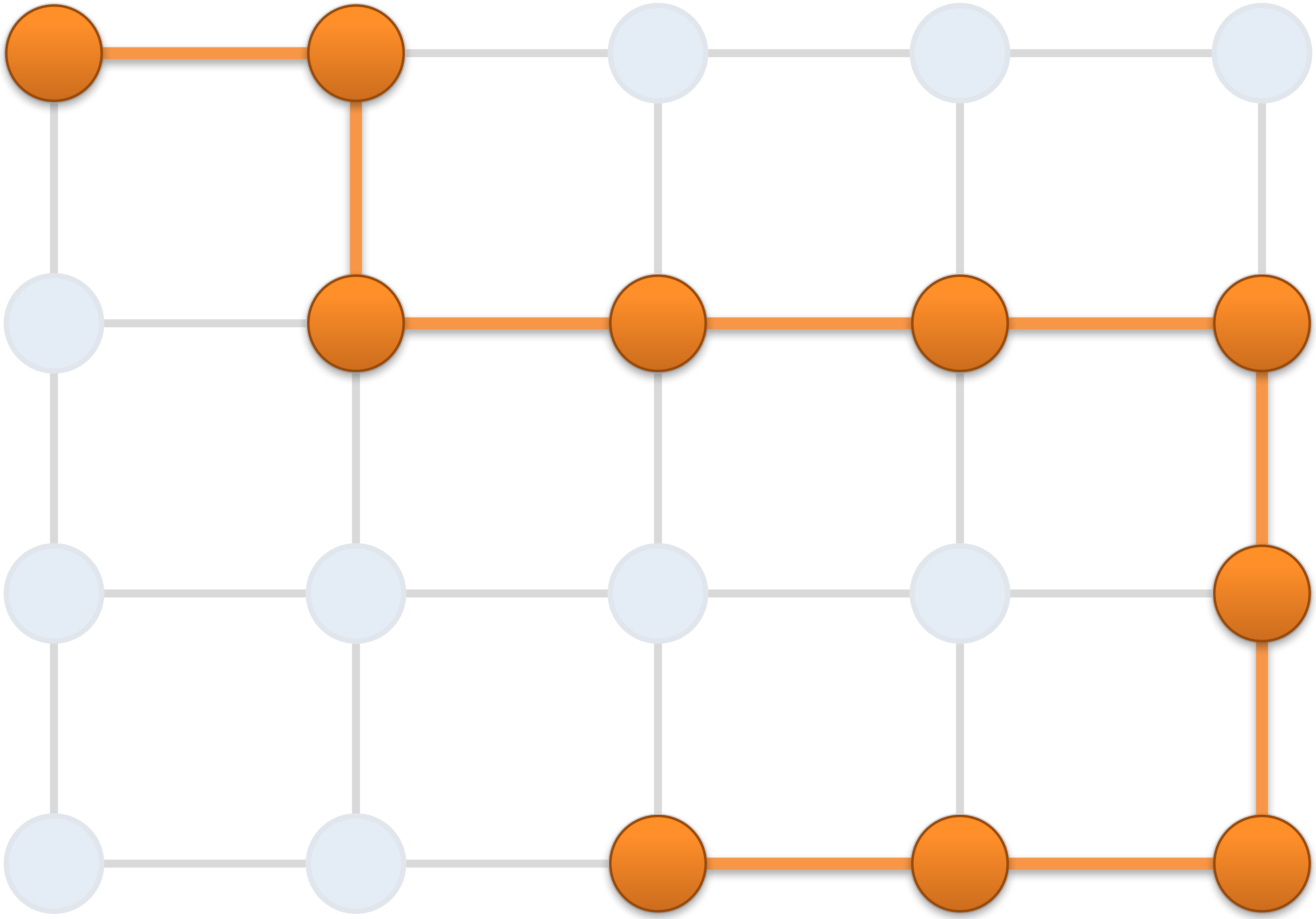
Synchronous Loopy BP

- 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
  - Kang,et al. LDMTA'10
  - …

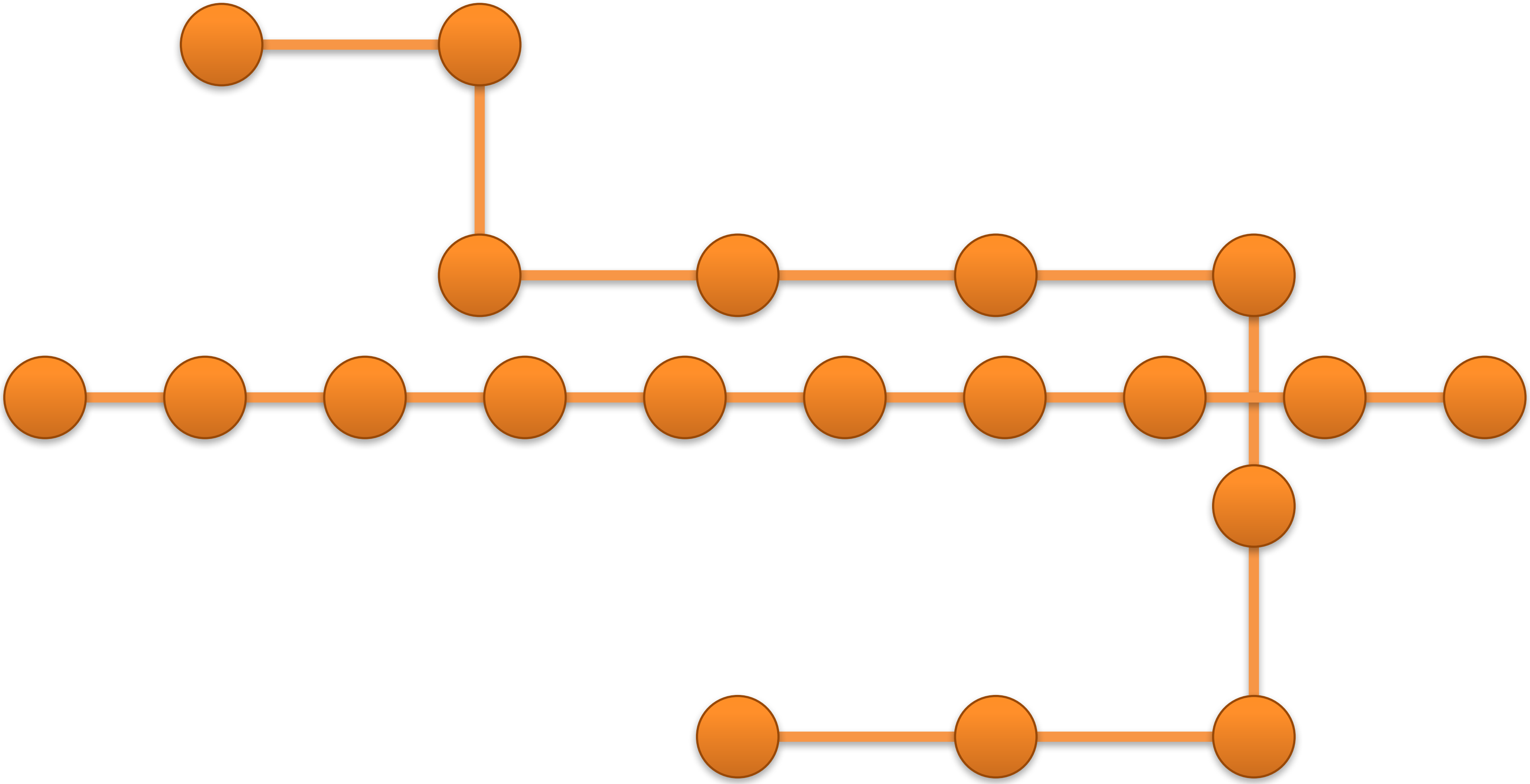


Is Synchronous Loopy BP an ***efficient*** *parallel algorithm?*

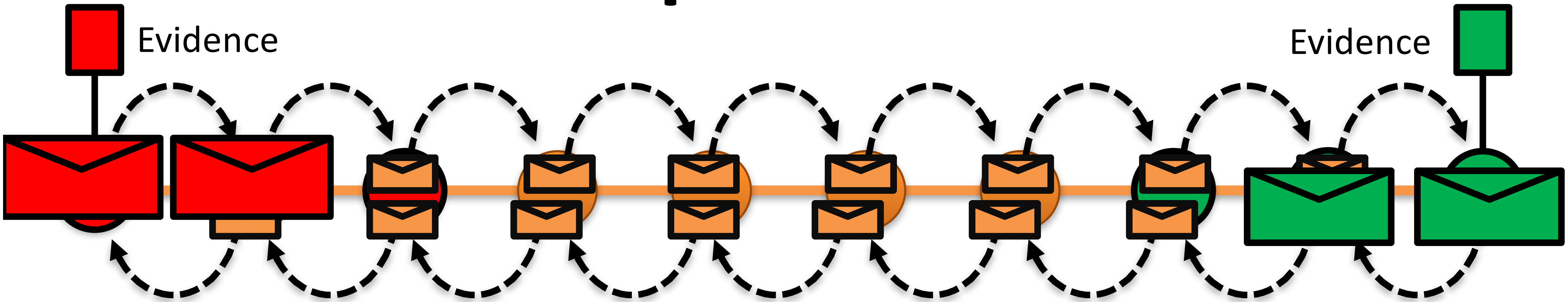
Sequential Computational Structure



Hidden Sequential Structure



Hidden **Sequential** Structure



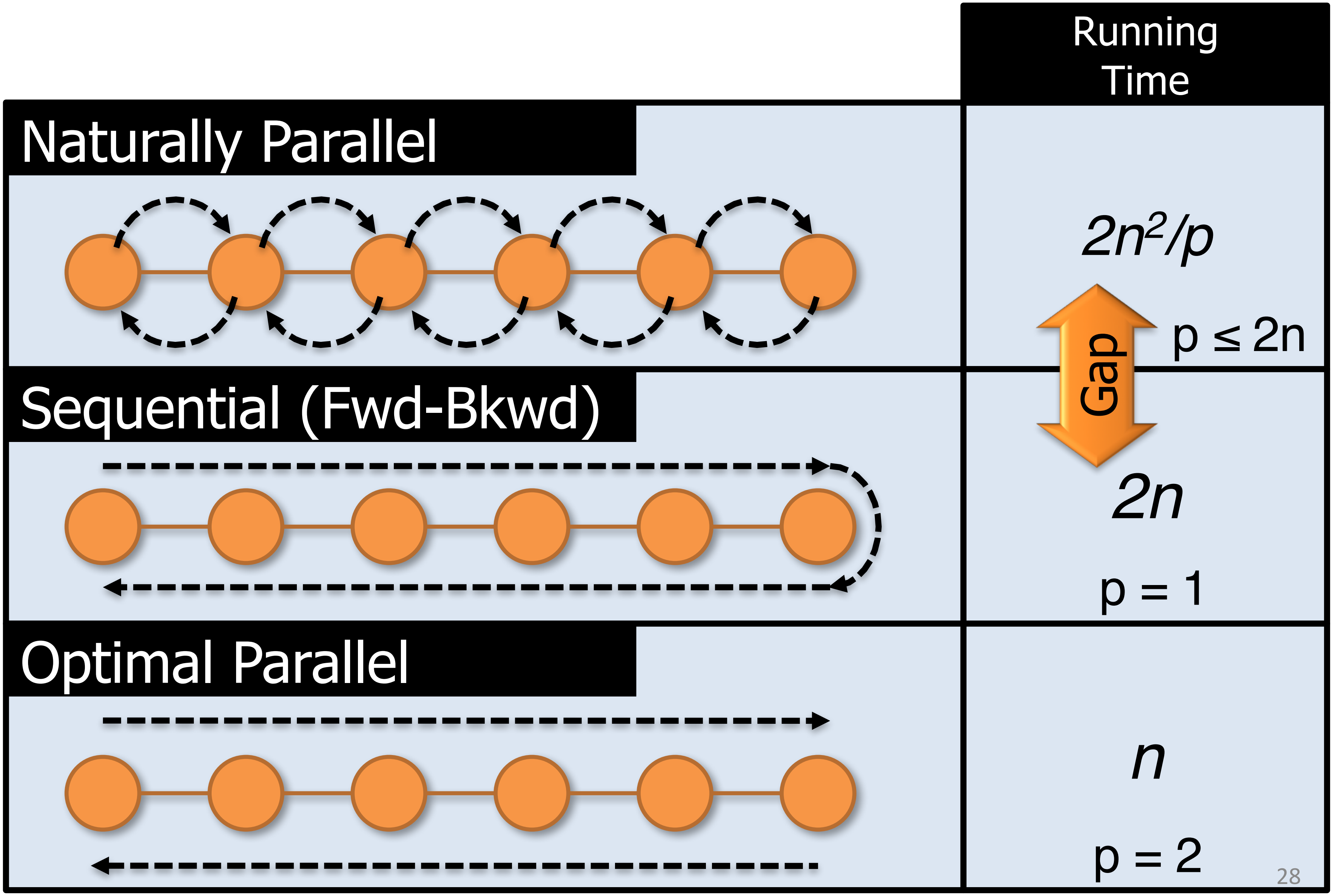
- Running Time:

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

Time for a single parallel iteration

Number of Iterations

Optimal Sequential Algorithm



Running Time

Naturally Parallel

$2n^2/p$

$p \le 2n$

Gap

Sequential (Fwd-Bkwd)

$2n$

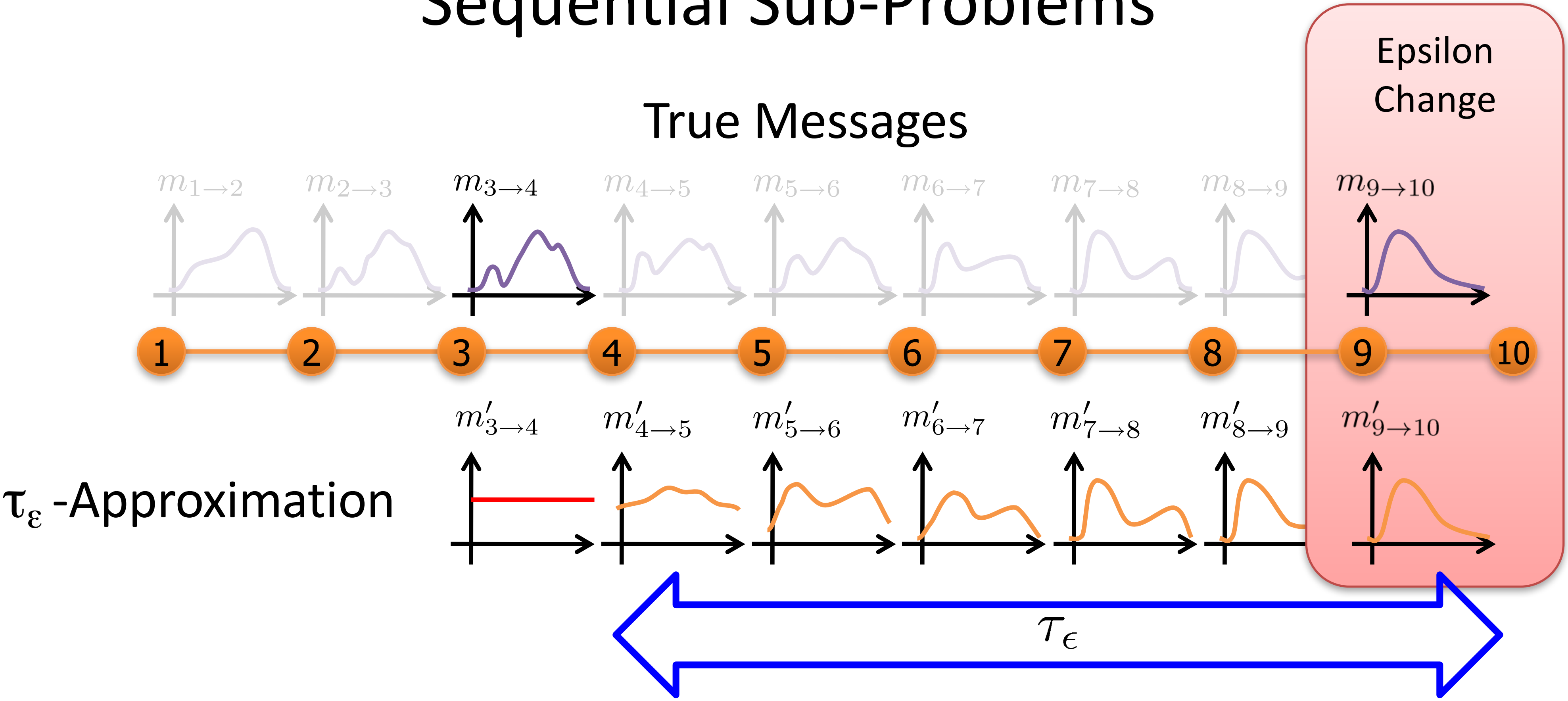
$p = 1$

Optimal Parallel

n

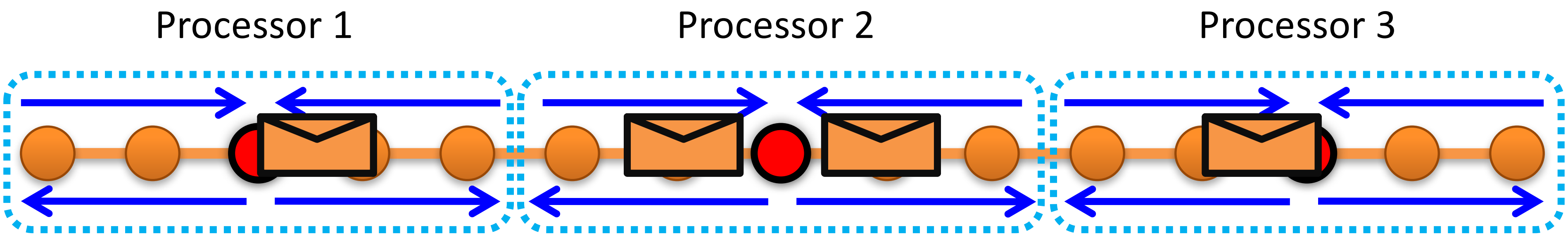
$p = 2$

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



- τ_ε represents the minimal sequential sub-problem
- Captures dependence on **model parameters**

Optimal Parallel Scheduling



Theorem:
Using p processors this algorithm achieves a τ_ε approximation in time:

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

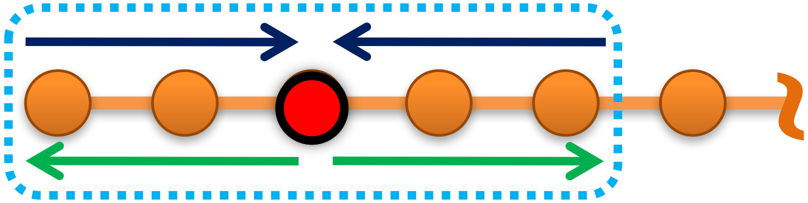
Parallel Component

Sequential Component

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

The Splash Operation

- Generalize the optimal chain algorithm:

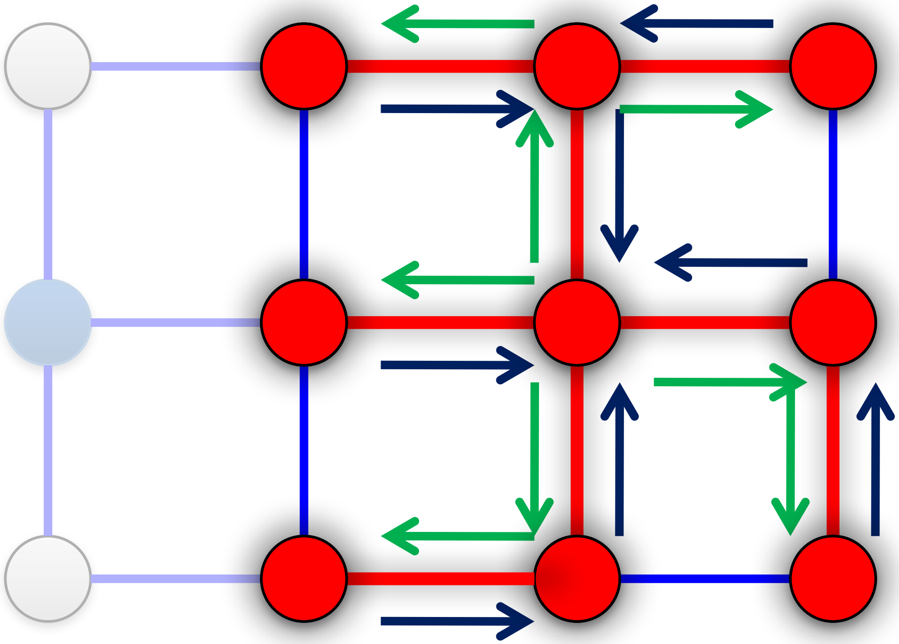


to arbitrary cyclic graphs:

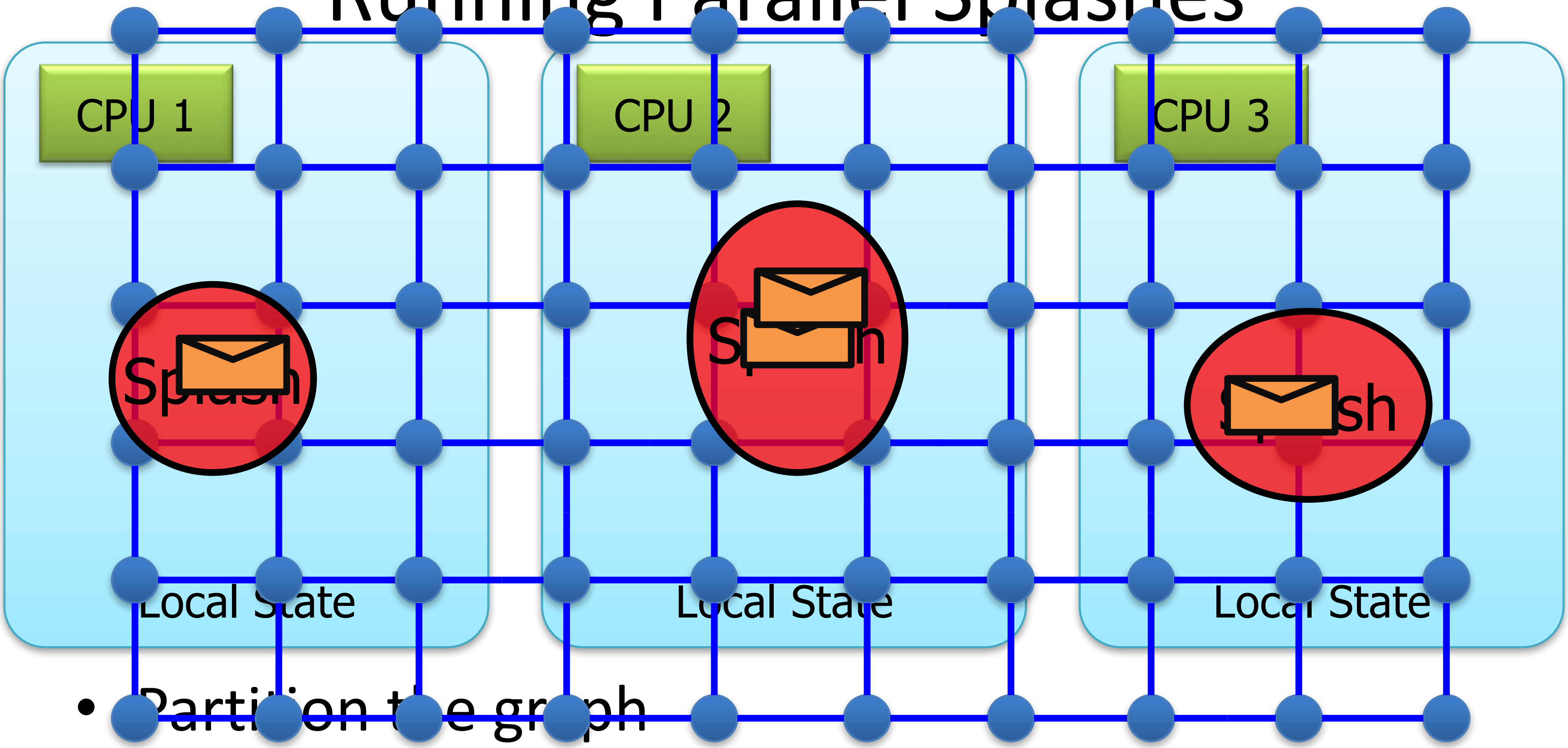
1) Grow a BFS Spanning tree with fixed size

2) Forward Pass computing all messages at each vertex

3) Backward Pass computing all messages at each vertex



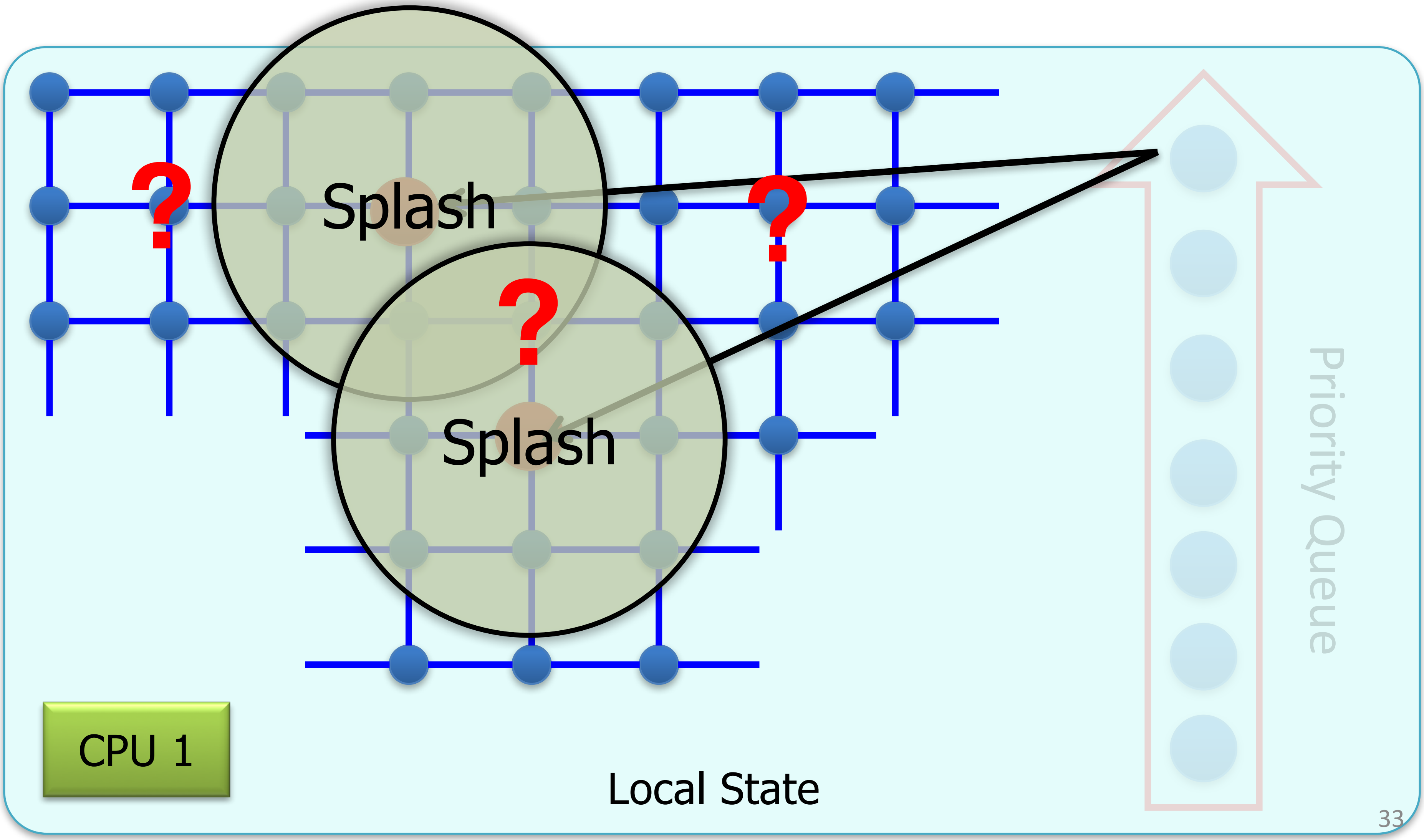
Running Parallel Splashes



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

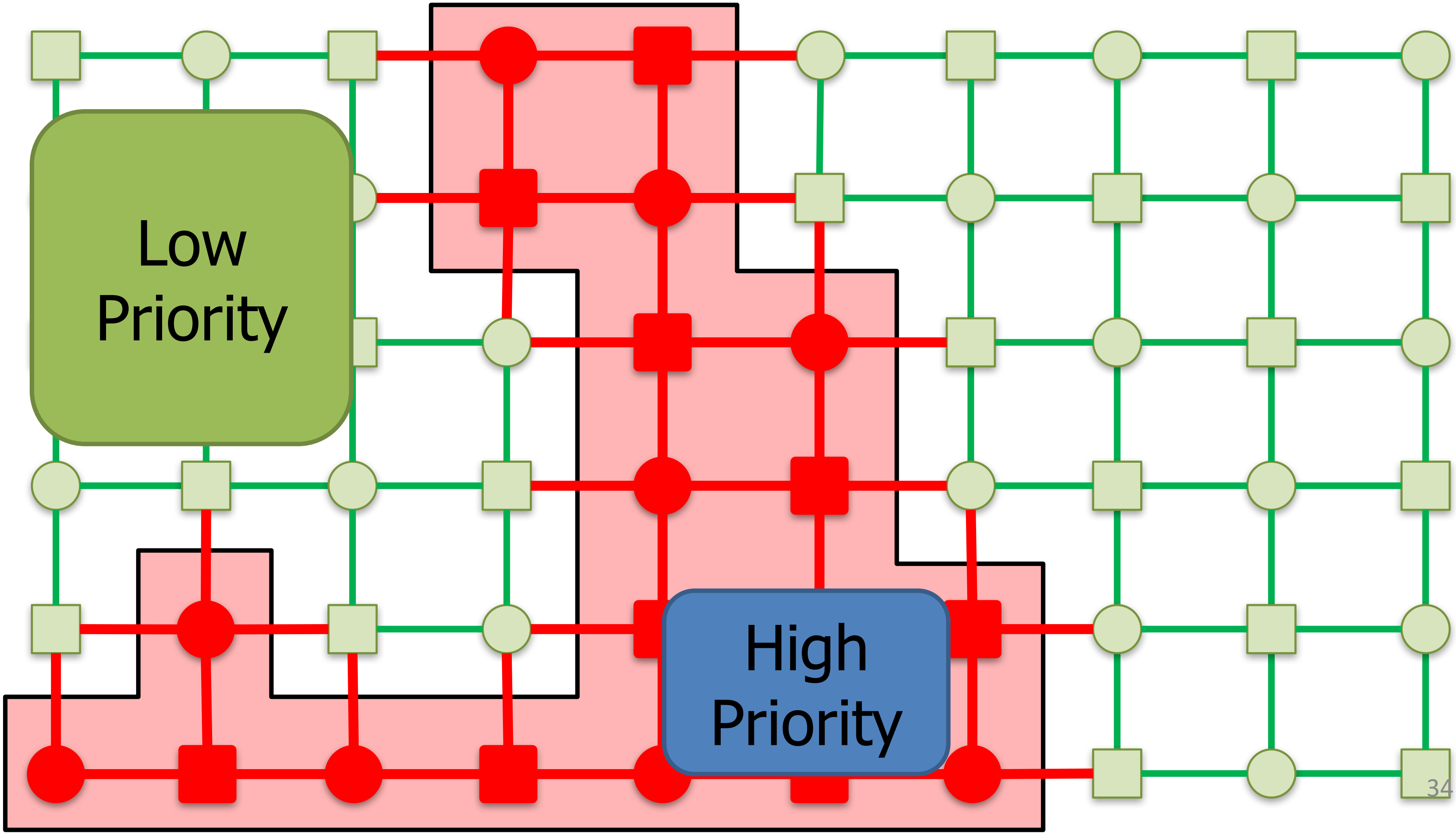
Priorities Determine the **Roots**

- Use a residual priority queue to select roots:

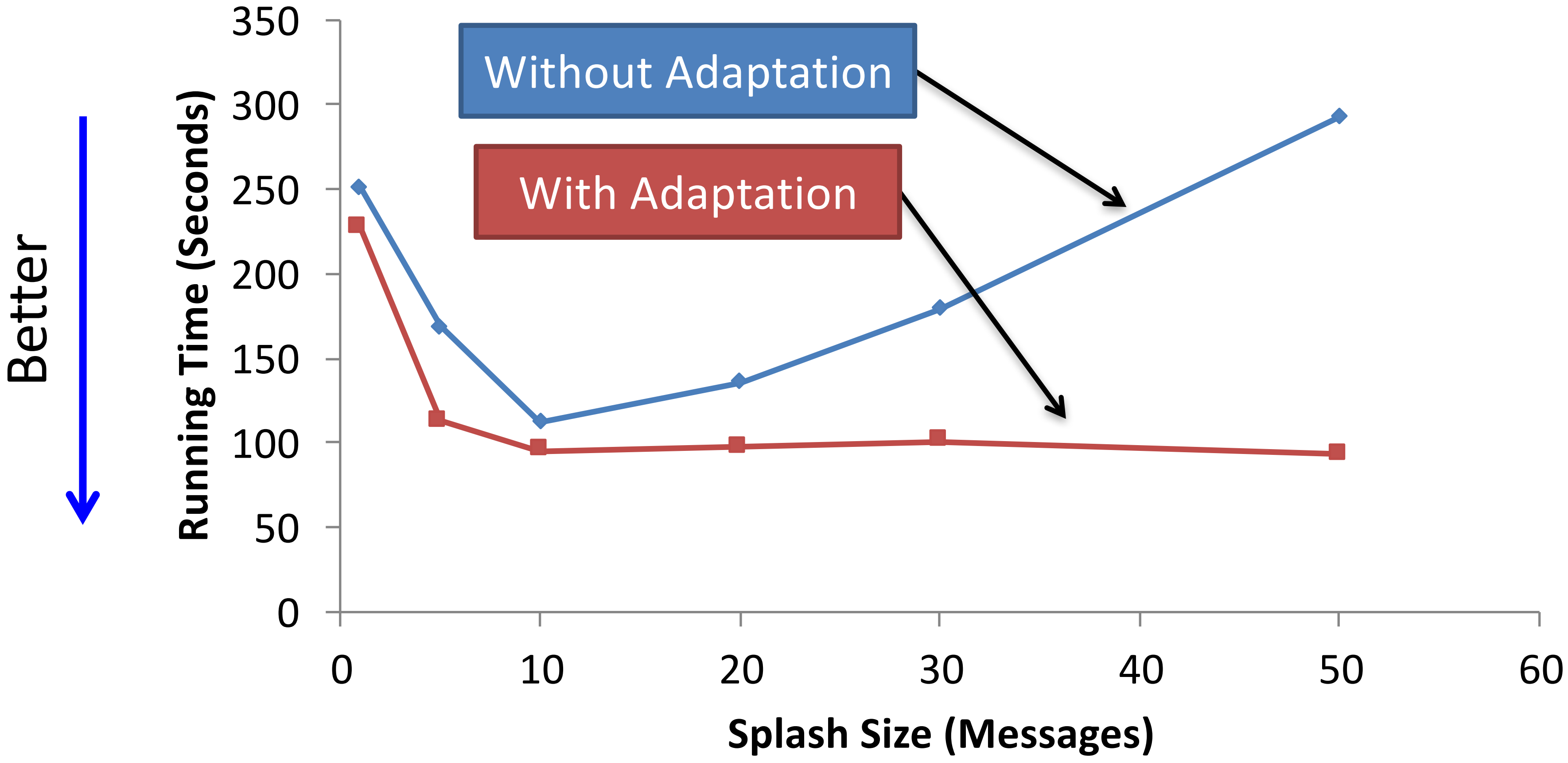


Dynamic Splashes

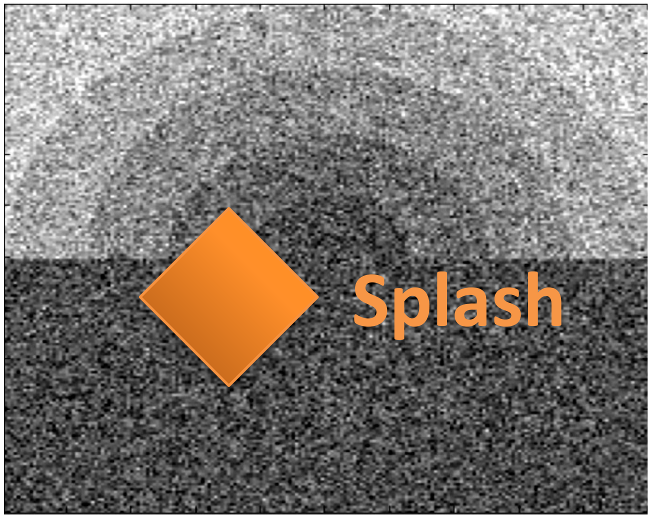
Priorities **adaptively** focus computation by determining the **shape** and **size** of each Splash



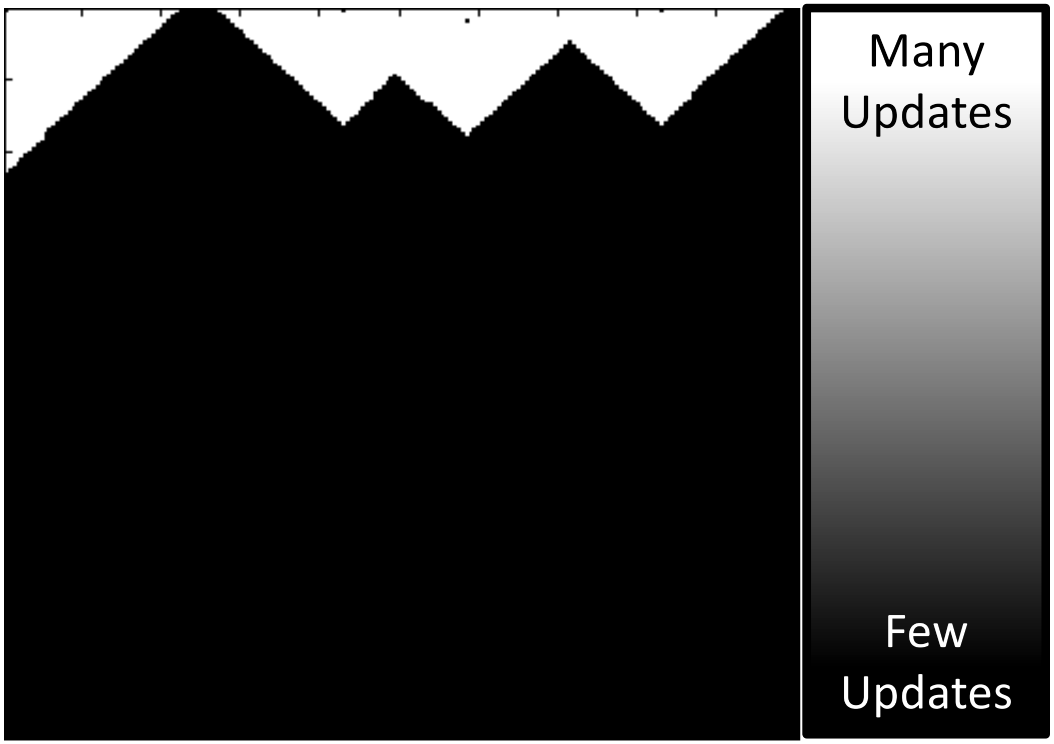
Dynamic Splashes automatically identify the **optimal** splash size



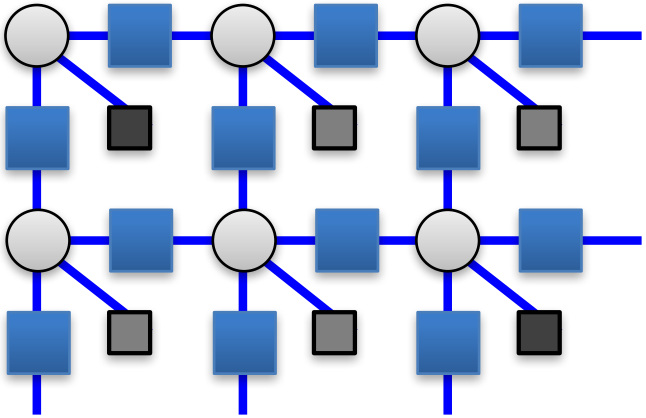
Splash Belief Propagation



Synthetic Noisy Image



Vertex Updates



Factor Graph

Algorithm identifies and focuses on hidden sequential structure

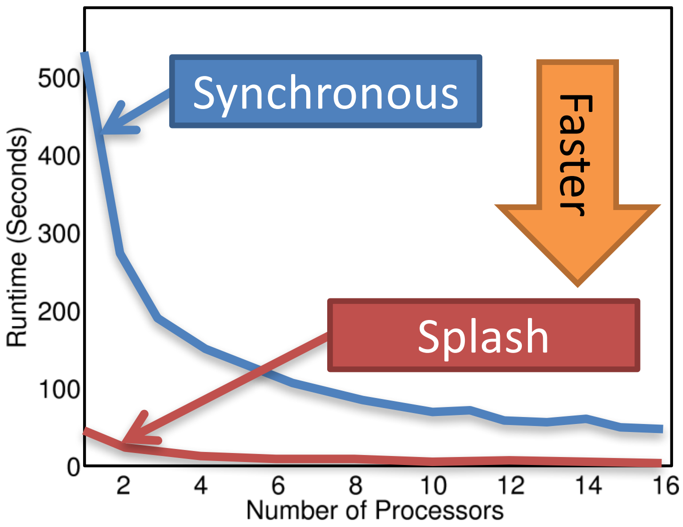
Evaluation

- System Design
  - Multicore and distributed implementations
  - Development was **time consuming**
- Evaluated on several real-world problems
  - Protein interaction and structure prediction
  - Markov Logic Networks
- Compared against several other variants
  - Faster, more efficient, more stable

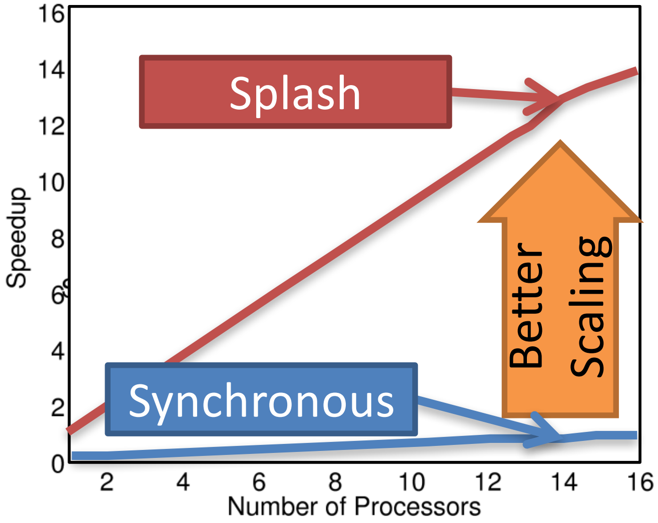
Representative Results

Protein Interaction Models: 14K Vertices, 21K Factors

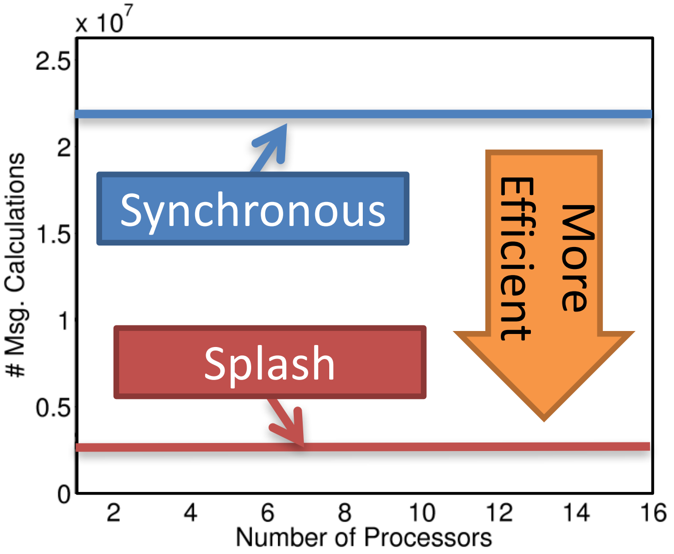
Runtime



Speedup



Total Work



- SplashBP converges more often
- Achieves better prediction accuracy

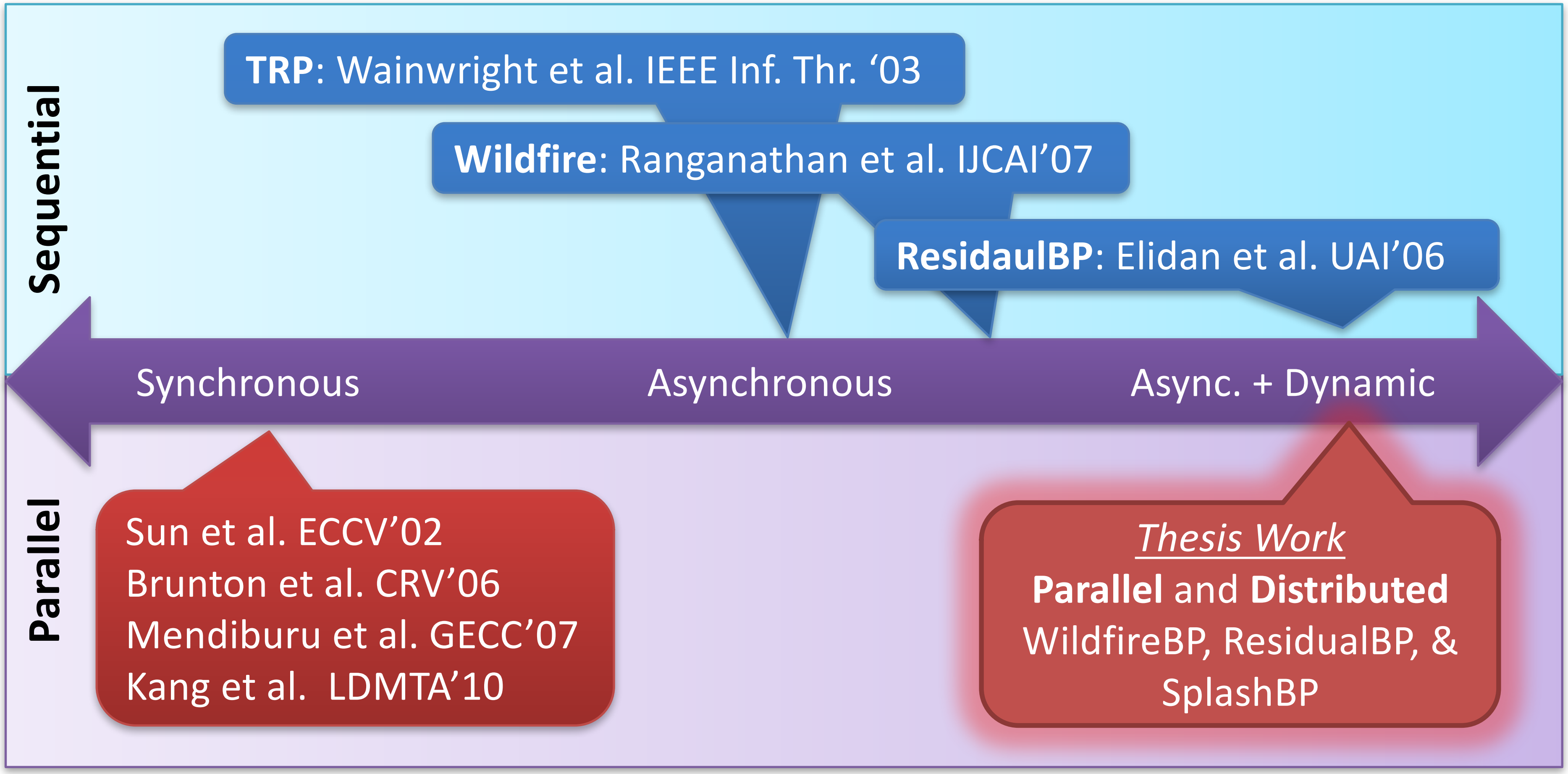
Summary: **Belief Propagation**

- **Asynchronous + Dynamic** → more efficient
  - *Theoretically* and *experimentally*
  - *Insight:* parallelize optimal **sequential** algorithm
  - *Tradeoff:* **Parallelism** & **Convergence**
- *Approximation* → Increased Parallelism
  - Exploit *weak* interactions (τ_{ε} – approximation)
- Key Contributions:
  - Demonstrate the importance of dynamic asynchronous scheduling in parallel inference
  - Theoretical analysis of work efficiency and relationship to model structure and parameters

GrAD Methodology

- **Graphical**
  - BP updates only depend on adjacent vertices
- **Asynchronous**
  - Compute messages sequentially within Splash
- **Dynamic**
  - Priority scheduling and adaptive Splashes

Additional Related Work



Sequential

TRP: Wainwright et al. IEEE Inf. Thr. ‘03

Wildfire: Ranganathan et al. IJCAI’07

ResidaulBP: Elidan et al. UAI’06

Synchronous

Asynchronous

Async. + Dynamic

Parallel

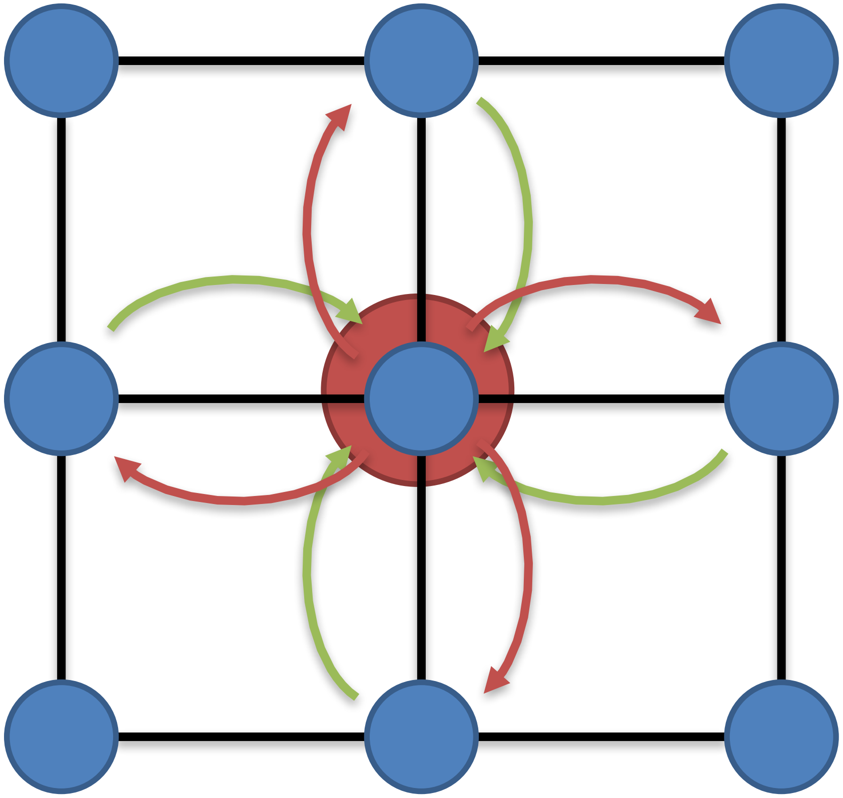
Sun et al. ECCV’02
Brunton et al. CRV’06
Mendiburu et al. GECC’07
Kang et al. LDMTA’10

Thesis Work
Parallel and **Distributed**
WildfireBP, ResidualBP, &
SplashBP

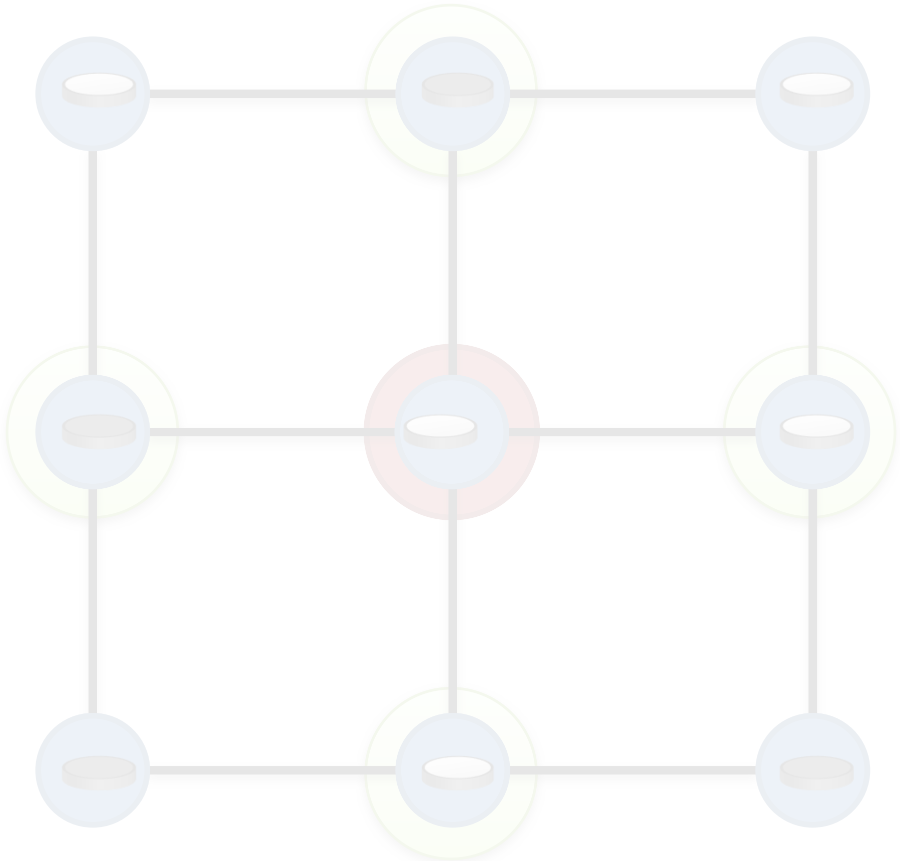
- **Parallel Exact Inference:** Pennock et al. UAI’98
- **Approximate Messages:** Ihler et al. JMLR’05

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

Belief Propagation



Gibbs Sampling



Parallel Gibbs Sampling

An ***asynchronous*** *Gibbs Sampler that* ***dynamically*** *addresses* ***strong dependencies****.*

Joint Work With

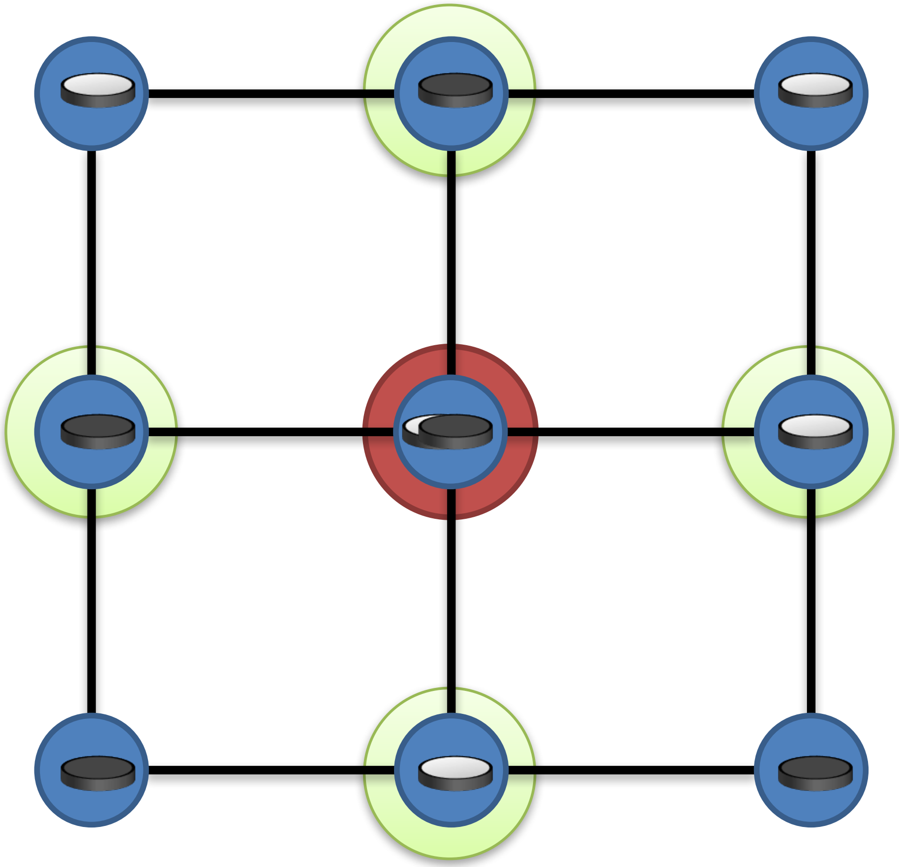
Yucheng Low ***Arthur Gretton*** ***Carlos Guestrin***

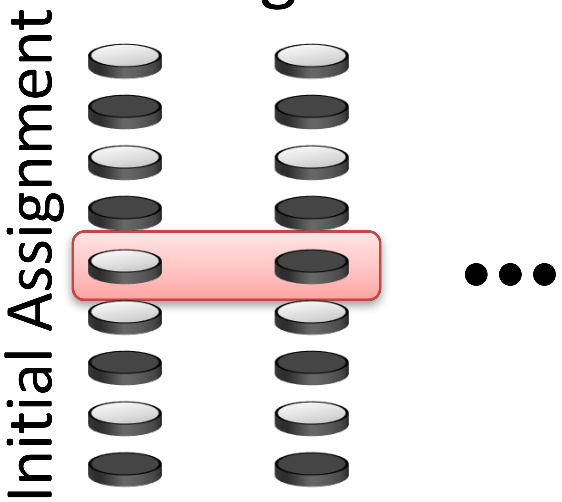
Published

AISTATS'11 (Related to work in WSDM'12)

Gibbs Sampling [Geman & Geman, 1984]

- **Sequentially** for each variable in the model
  - Select variable
  - Use adjacent assignments to construct a biased coin
  - Flip coin and update assignment to variable



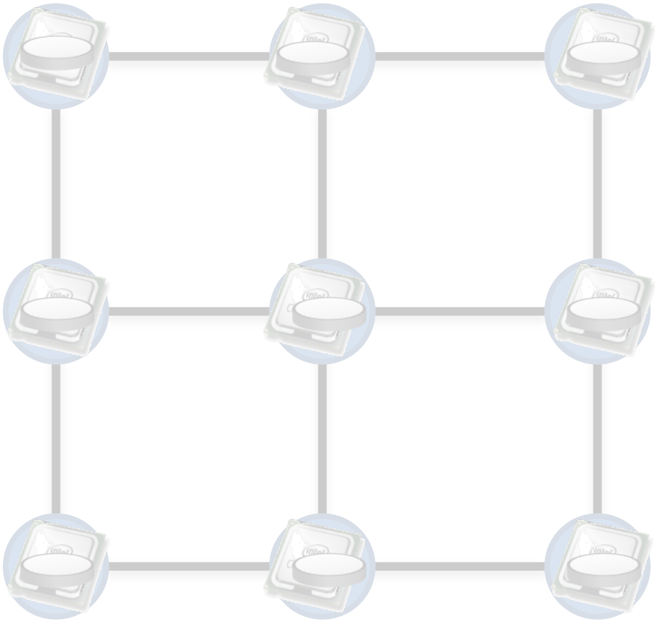


*Can we sample multiple variables in **parallel**?*

From the original paper on Gibbs Sampling:

> *"…the MRF can be divided into collections of [variables] with each collection assigned to an **independently** running **asynchronous processor**."*

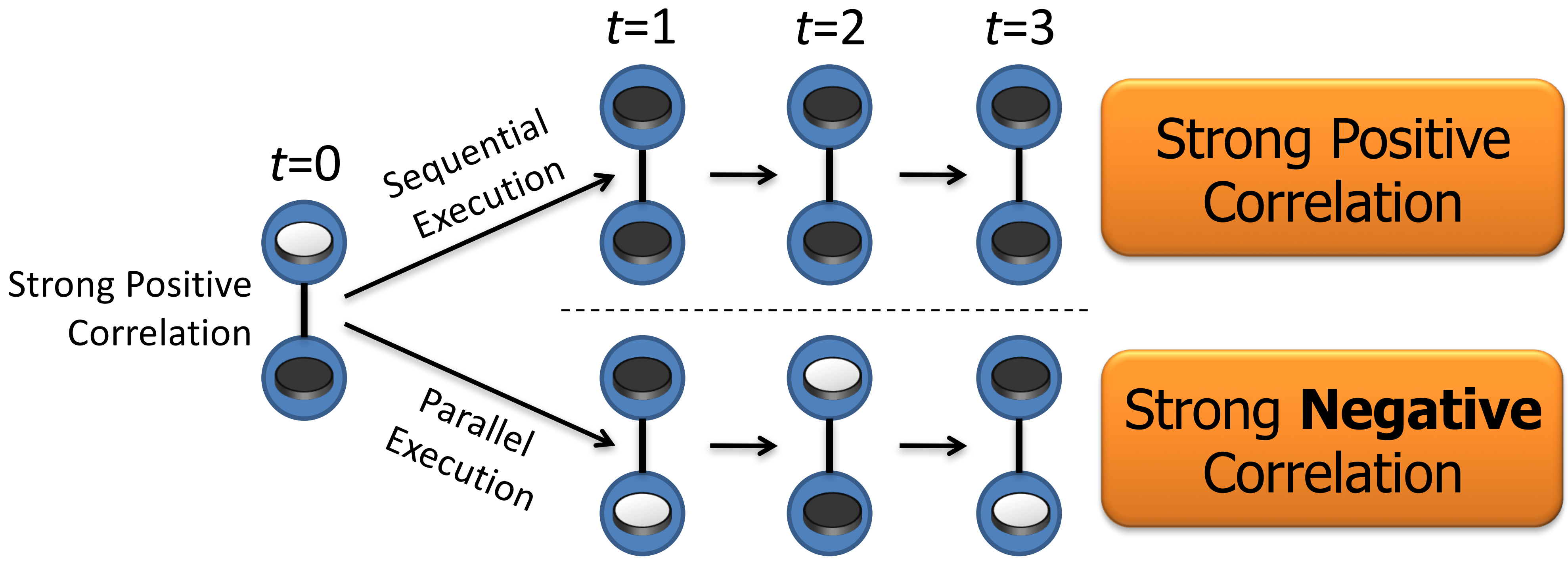
-- Stuart and Donald Geman, 1984.



Embarrassingly Parallel!

Converges to the **wrong** distribution!

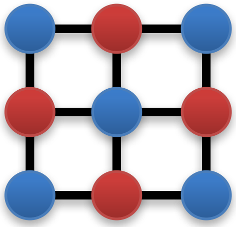
The problem with **Synchronous** Gibbs sampling



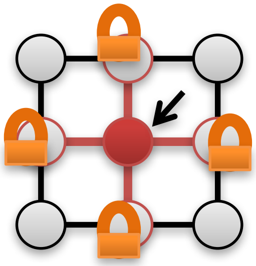
t=0
Strong Positive Correlation
Sequential Execution
Parallel Execution
t=1
t=2
t=3
Strong Positive Correlation
Strong **Negative** Correlation

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

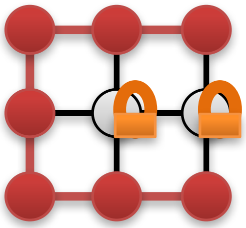
Introduced Three Convergent Samplers



Chromatic: Use graph coloring to synchronously sample independent sets



Asynchronous: Markov Blanket Locks ensure serializable execution

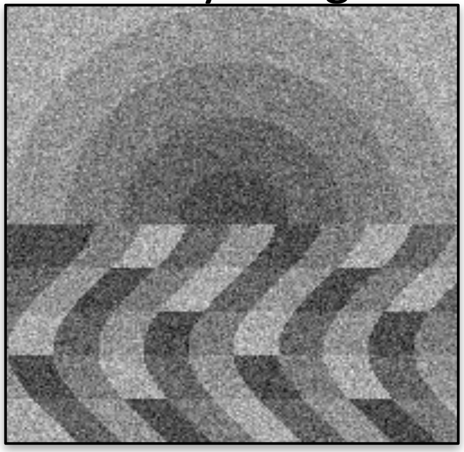


Splash: Adaptively constructs thin junction tree blocks

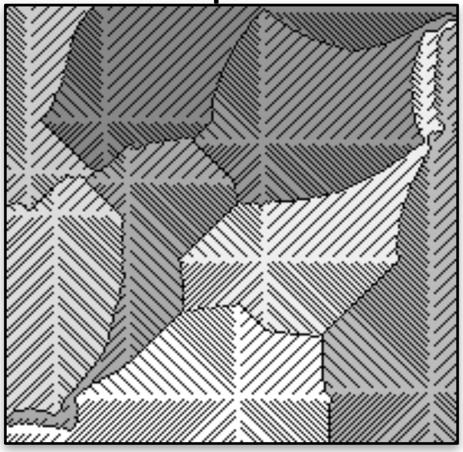
Dynamically Prioritized Sampling

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

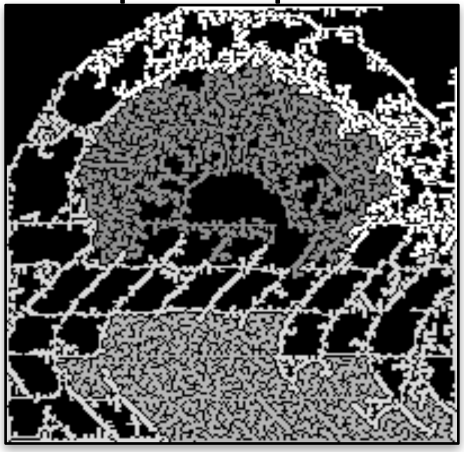
Noisy Image



BFS Splashes



Adaptive Splashes



Theorem: *Chromatic* Sampler

- **Ergodic:** converges to the correct distribution
  - – Based on graph coloring of the Markov Random Field
- **Quantifiable** acceleration in **mixing**

Time to update
all variables once

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

# Variables

# Colors

# Processors

Theorem
Asynchronous and *Splash Gibbs* Sampler

- **Ergodic:** converges to the correct distribution
  - Requires vanishing adaptation
  - Corrected an error in a result by Levin & Casella *J. Multivar. Anal. '06*
- **Expected Parallelism:**

Max Degree

$$\mathbf{E}(\#\text{active processors})$$

$$\geq 1 + (p-1)\left(1 - (p-1)\left(\frac{d+1}{n}\right)\right)$$

# Processors

Variables

Evaluation

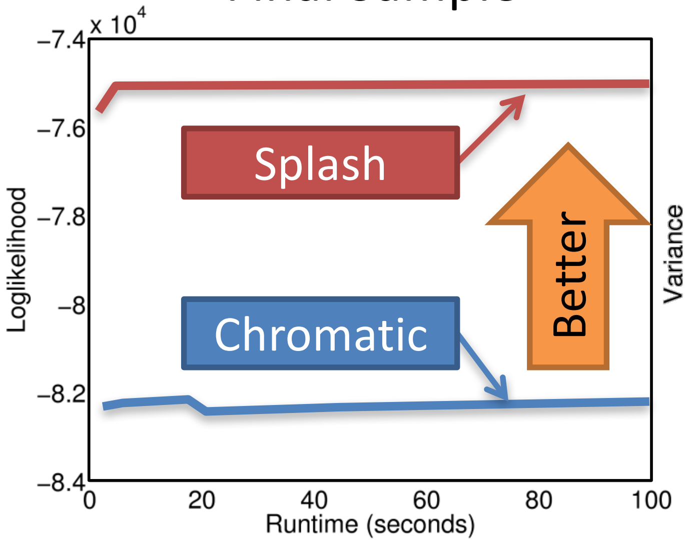
- Implemented multicore version:
  - Built using a GraphLab prototype
    - Substantially shorter development time
  - Novel **junction tree** construction algorithm
  - Markov blanket locking protocol
- Evaluated on large real-world problems

Experimental Results

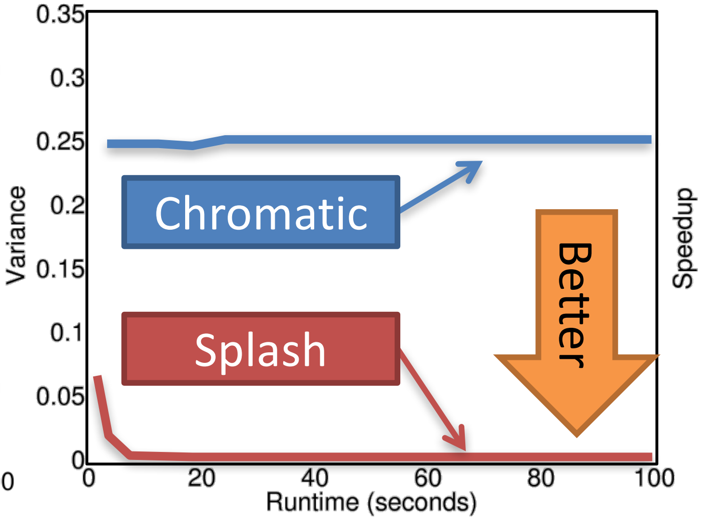
- Markov logic network with **strong dependencies**

10K Variables *28K Factors*

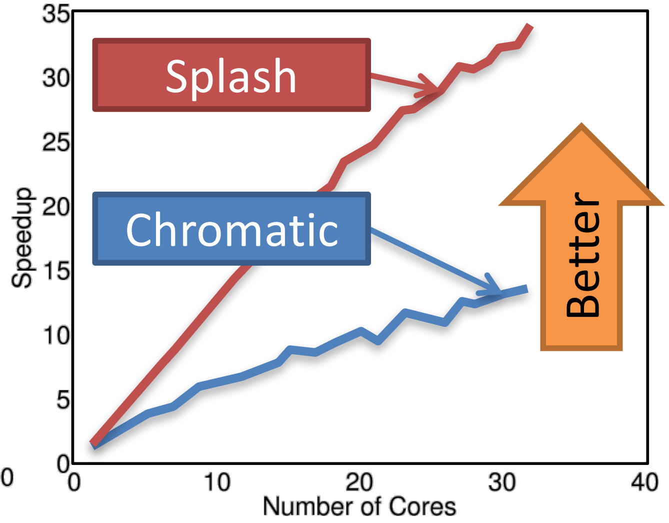
Likelihood Final Sample



“Mixing”



Speedup in Sample Generation



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

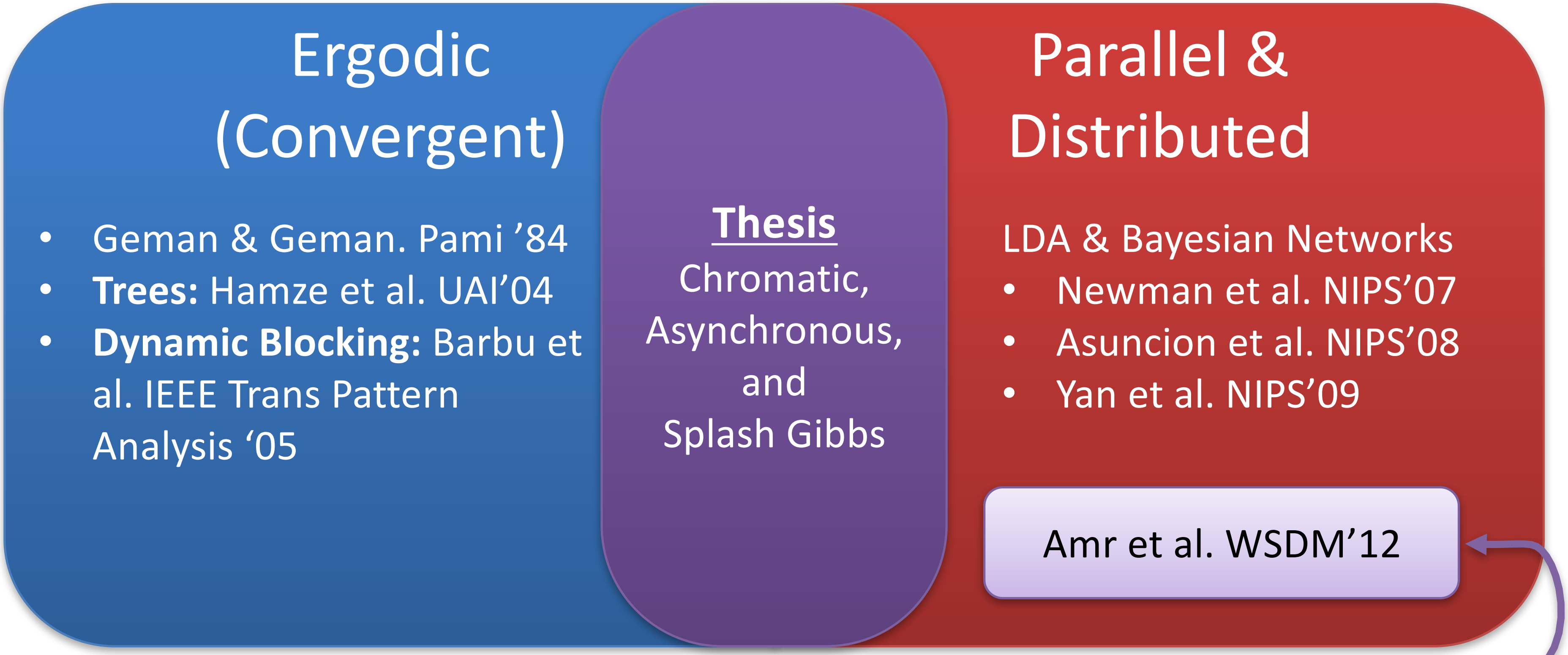
Contributions: Gibbs Sampling

- Proposed **three** *convergent* Gibbs samplers
  - Chromatic, Asynchronous, Splash
  - Spectrum partially synchronous to asynchronous
  - New algorithms for junction tree construction
- Theoretical analysis of parallel Gibbs sampling
  - Convergence of asynchronous blocking
  - Relate parallelism to model structure
  - Stationary distribution of synchronous sampler
- Experimental analysis on real-world problems and systems

GrAD Methodology

- **Graphical**
  - Gibbs updates depend only on neighbors in MRF
- **Asynchronous**
  - Graph *Coloring* and *Markov Blanket Locks*
- **Dynamic**
  - Prioritized updates and adaptive Splash

Related Work



Ergodic (Convergent)

- Geman & Geman. Pami ’84
- **Trees:** Hamze et al. UAI’04
- **Dynamic Blocking:** Barbu et al. IEEE Trans Pattern Analysis ‘05

Thesis
Chromatic,
Asynchronous,
and
Splash Gibbs

Parallel & Distributed

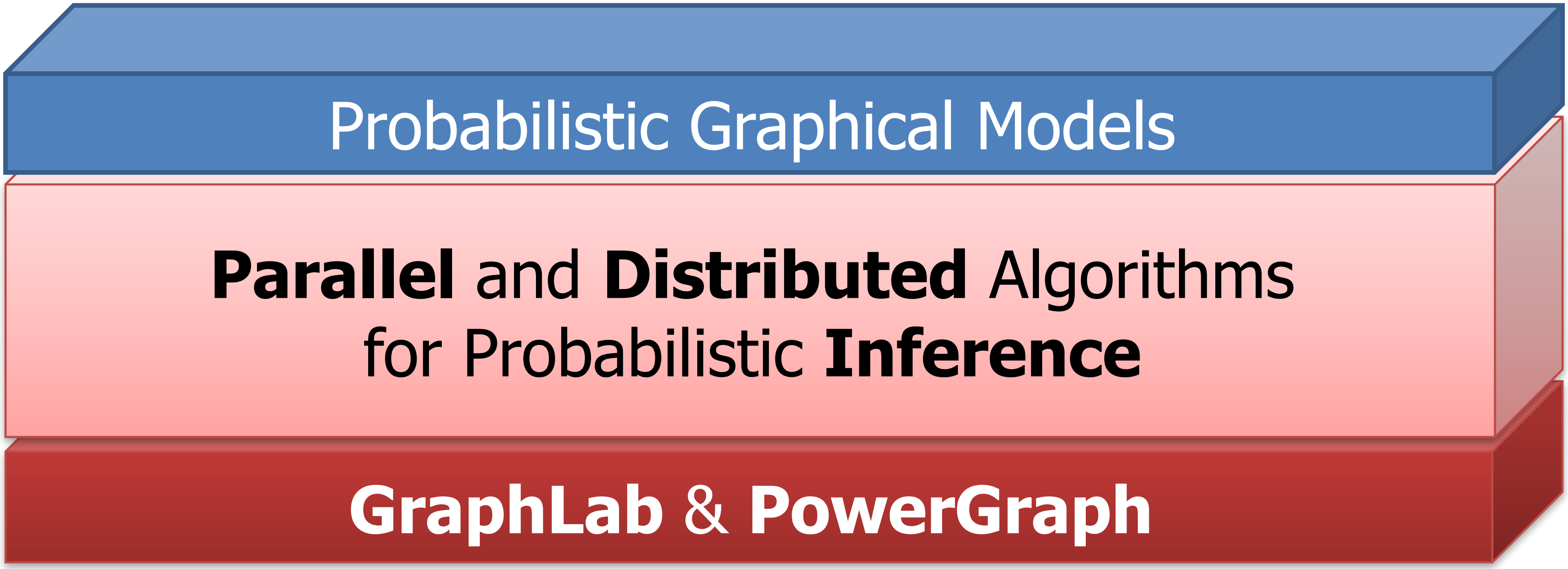
LDA & Bayesian Networks

- Newman et al. NIPS’07
- Asuncion et al. NIPS’08
- Yan et al. NIPS’09

Amr et al. WSDM’12

- Asynchronous approximations empirically perform well

Massive Structured Problems

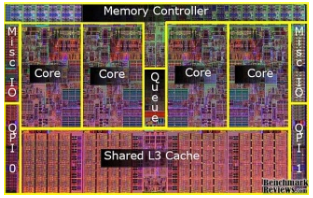


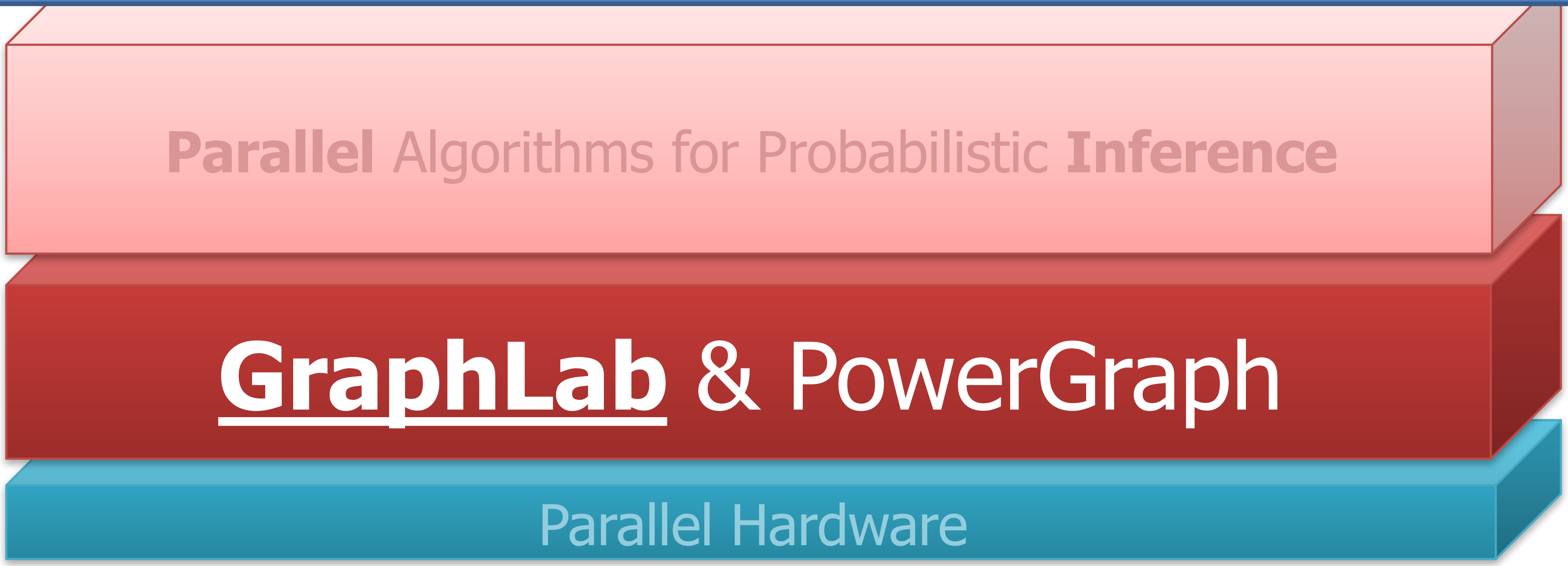
Probabilistic Graphical Models

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

GraphLab & **PowerGraph**

Advances Parallel Hardware





Parallel Algorithms for Probabilistic **Inference**

GraphLab & PowerGraph

Parallel Hardware

Joint Work With

Yucheng Low Aapo Kyrola Haijie Gu Danny Bickson
Carlos Guestrin Joe Hellerstein Guy Blelloch David O'Hallaron

Published Results

UAI'10 VLDB'12

How do we **design** and **implement** **GrAD Algorithms**

We could:

- *design* and *implement* for each architecture?
  - **Time consuming**
  - Repeatedly solving the same system problems
- use high-level abstractions like **MapReduce**?
  - Unable to express:
    - **Graphical**
    - **Asynchronous**
    - **Dynamic**

GrAD Methodology

Solution: GraphLab

GraphLab is a **Graph-Parallel** Abstraction

Data-Parallel

Graph-Parallel

Map Reduce

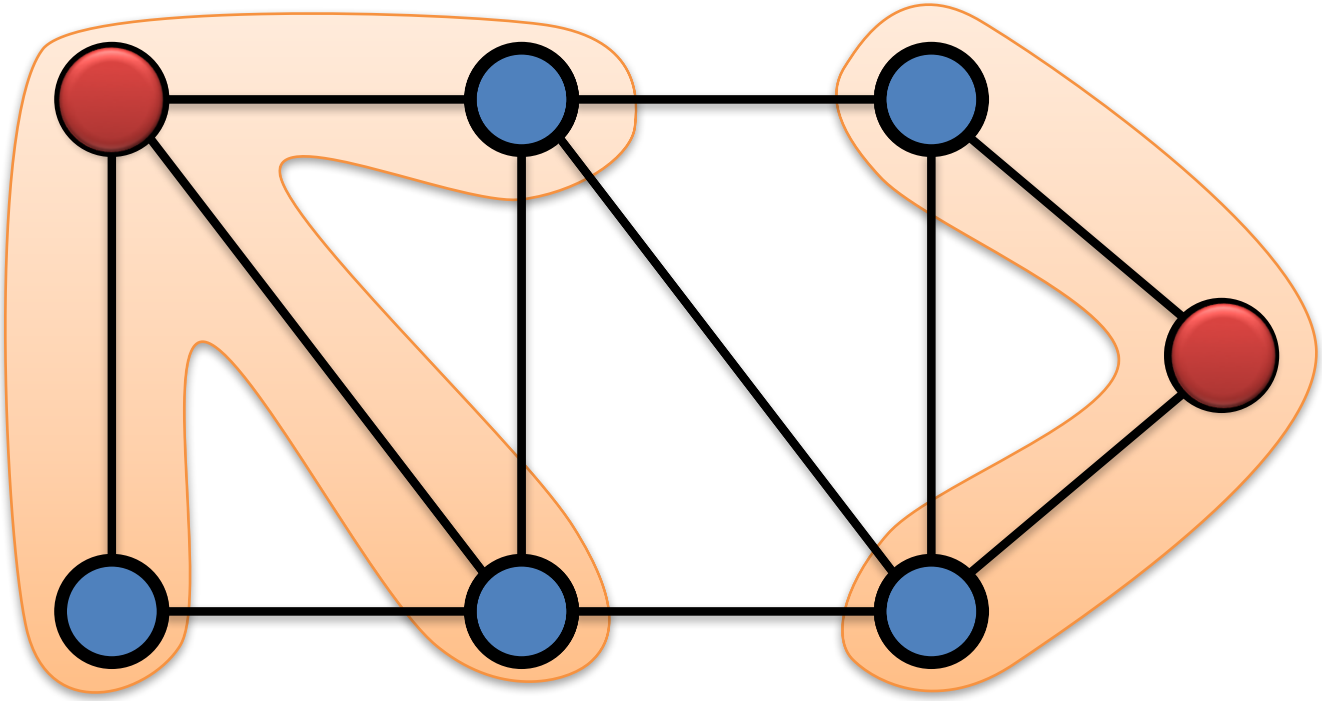
- *Independent* Data
- *Single Pass*
- *Synchronous*

GraphLab

- **Graph** Structured Data
- **Iterative** Computation
- **Dynamic + Asynchronous**

The **GraphLab** Abstraction

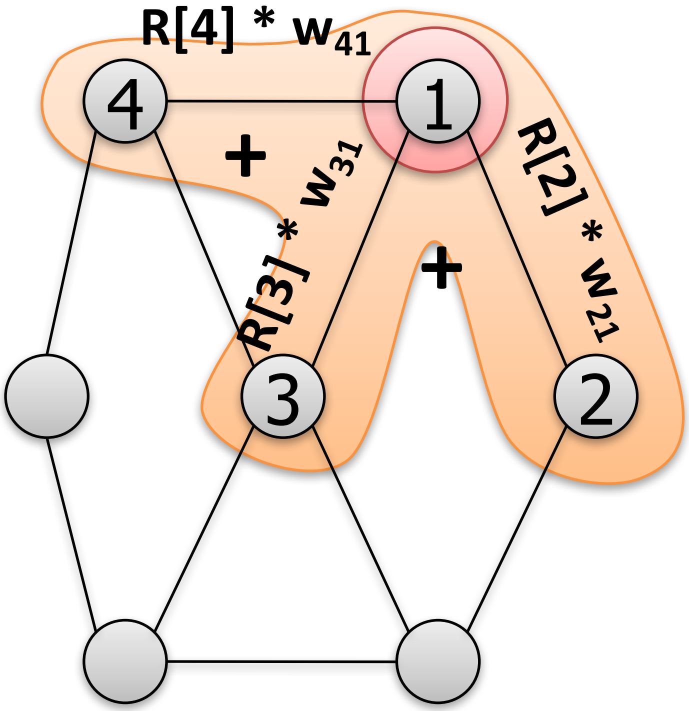
- A user-defined **Vertex Program** runs on each vertex
- **Graph** constrains **interaction** along edges
  - Directly **read** and **modify** the state of adjacent vertices and edges
- **Parallelism**: run multiple vertex programs simultaneously



The GraphLab Vertex Program

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

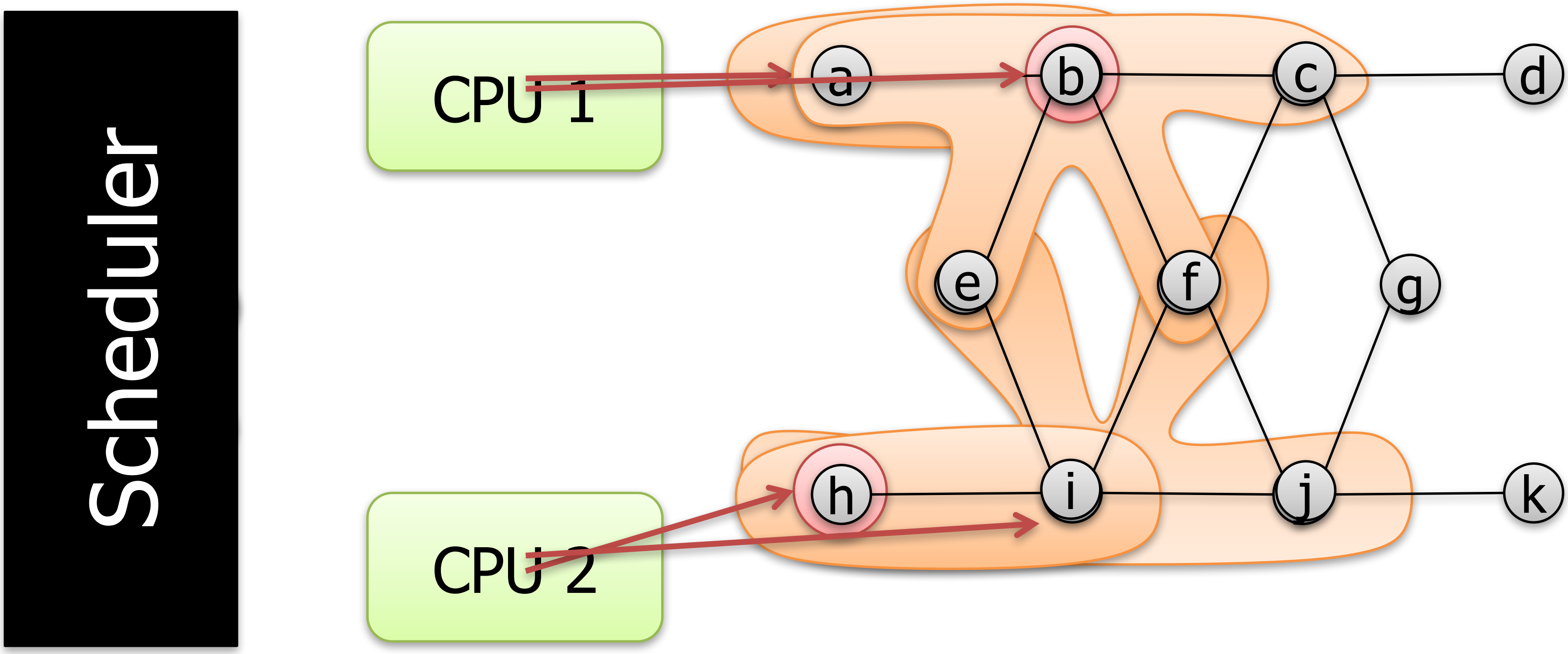
```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * $w_{ji}$

  // Update the PageRank
  R[i] = total

  // Trigger neighbors to run again
  priority = |R[i] – oldR[i]|
  if R[i] not converged then
    signal neighbors(i) with priority
```



Dynamics

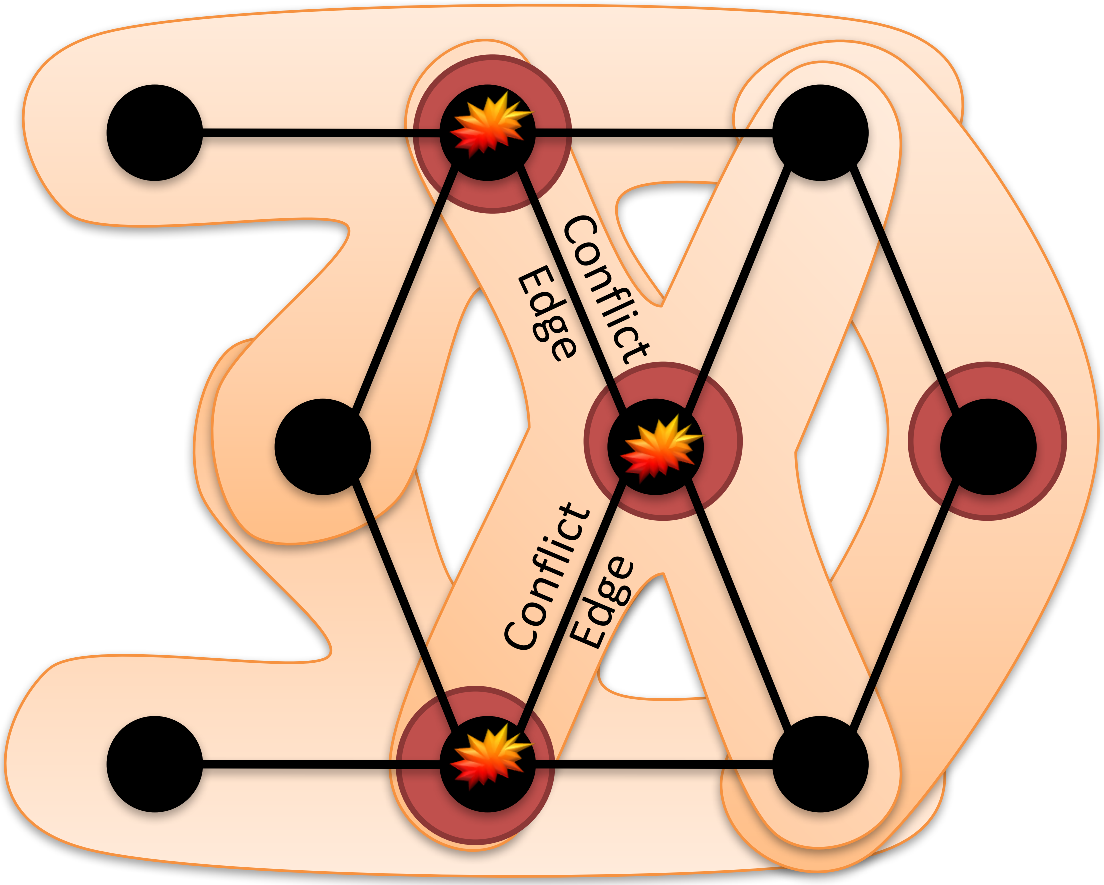
GraphLab is **Asynchronous**

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



Scheduler can **prioritize** vertices.

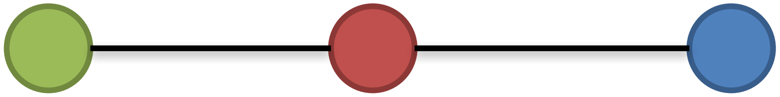
GraphLab is **Serializable**

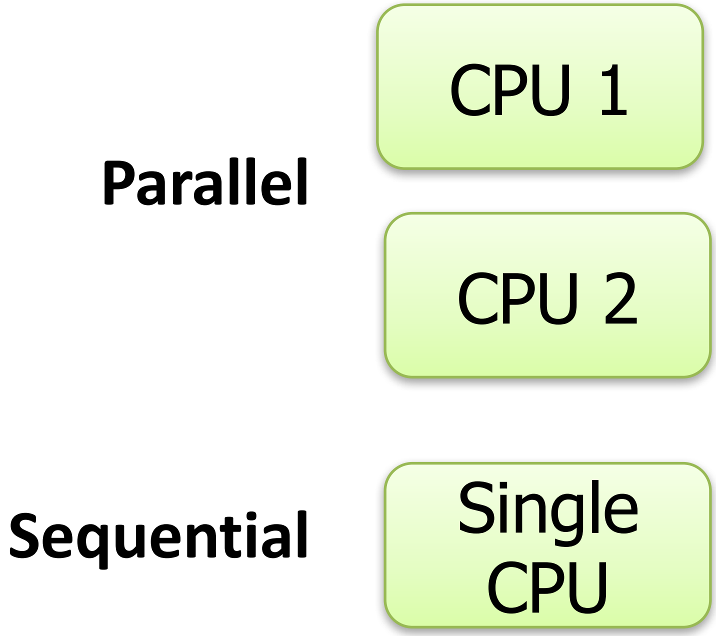


- Automatically ensures **serializable** executions

Serializable Execution

For **each parallel execution**, there exists a **sequential execution** of vertex-programs which produces the same result.





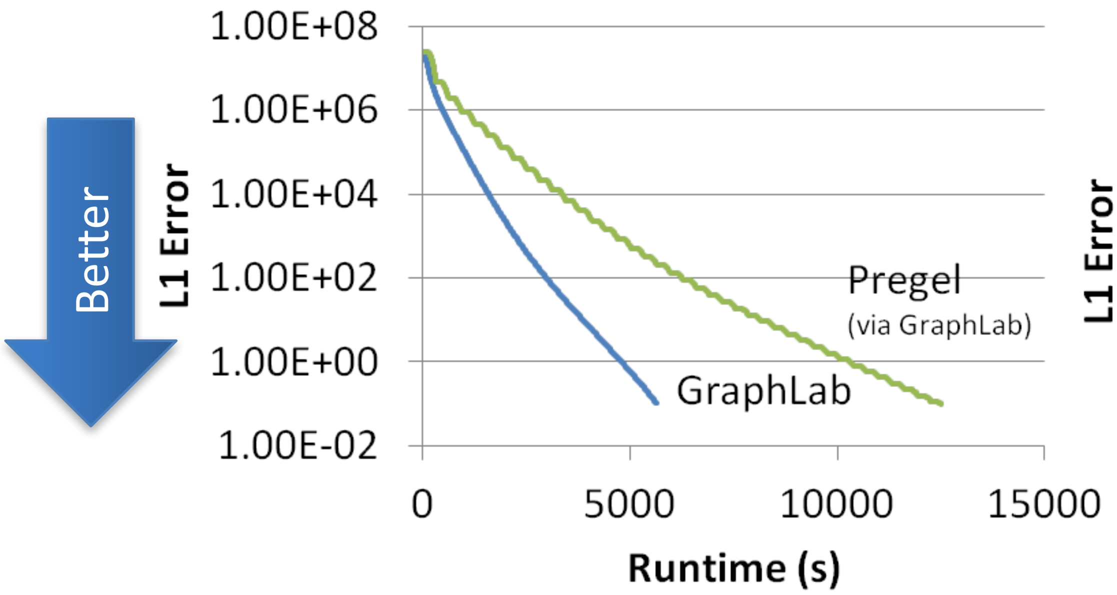
The GraphLab System

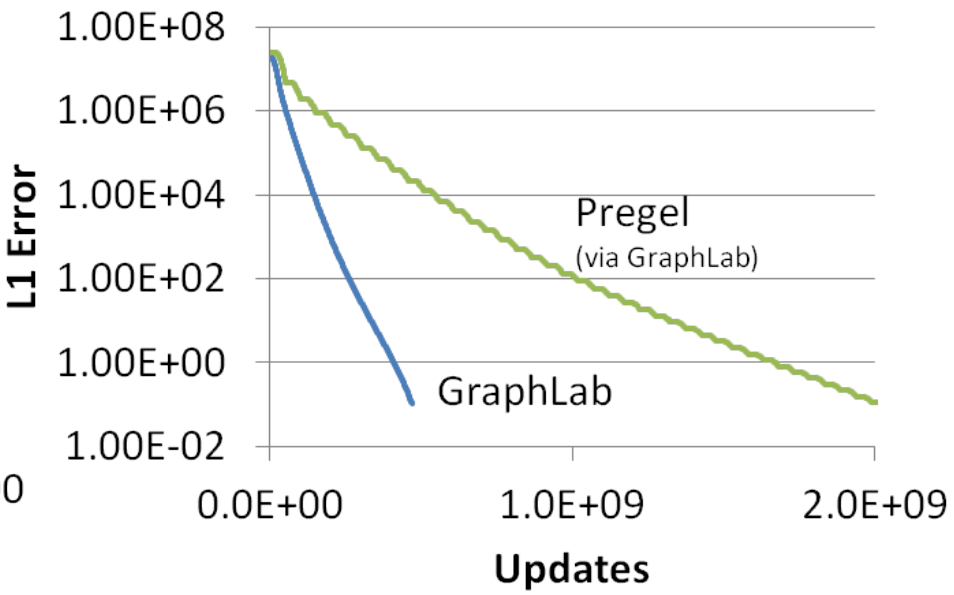
- Implemented as a C++ API
  - Widely downloaded open-source project

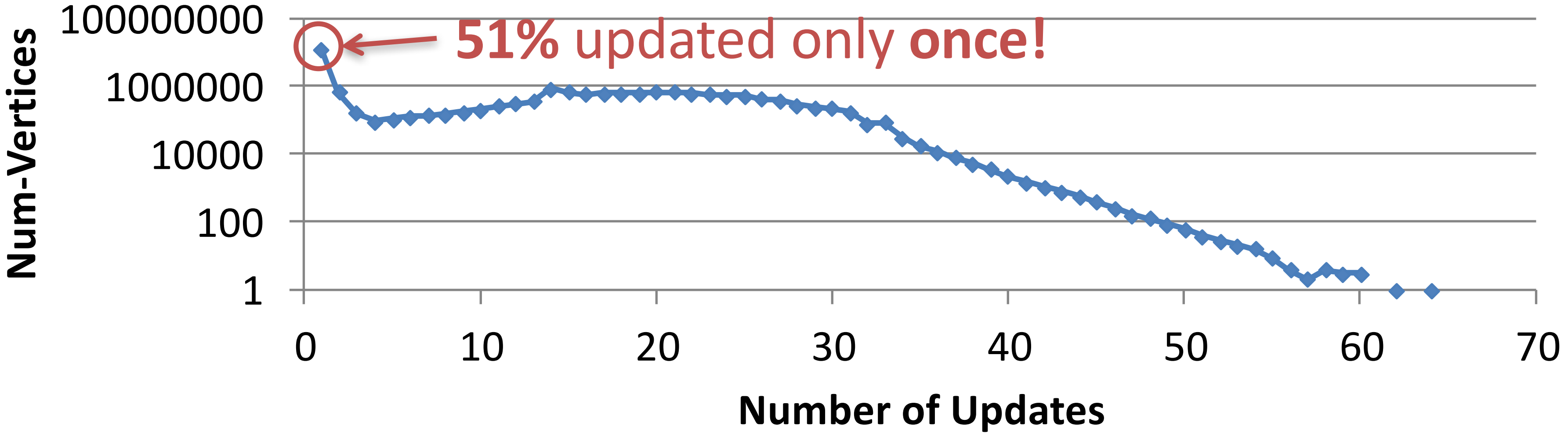
- **Multicore** and **distributed** versions:
  - **Hide Latency:** Pipelined locking
  - **Fault Tolerance:** Chandy-Lamport Snapshot

- Tested on a wide range of ML algorithms
  - ALS, BP, Gibbs, Lasso, CoEM, SVM, LDA, …

GraphLab vs. Pregel (BSP)



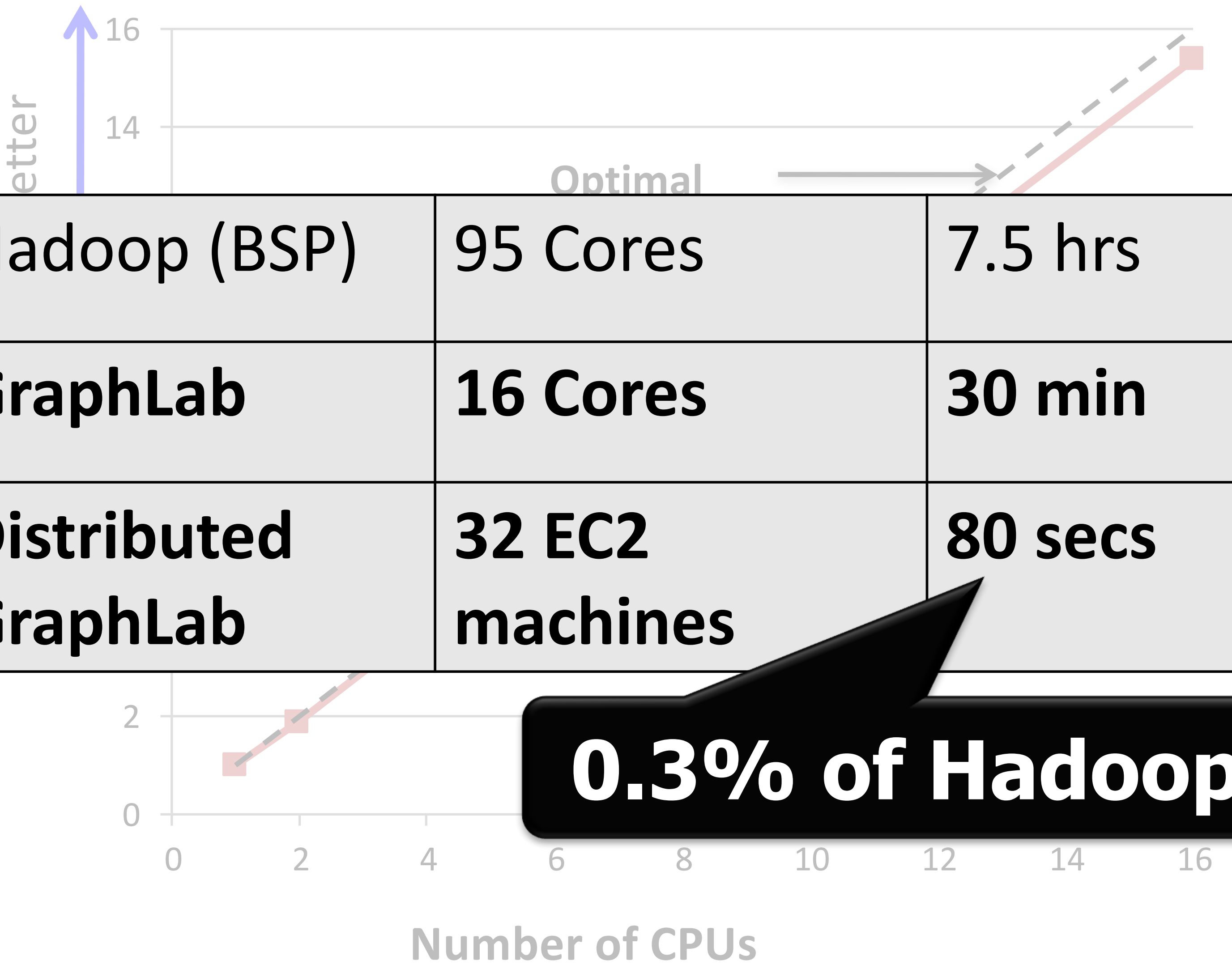




51% updated only **once!**

PageRank (25M Vertices, 355M Edges)

Never Ending Learner Project (CoEM)



Hadoop (BSP)	95 Cores	7.5 hrs
GraphLab	**16 Cores**	**30 min**
Distributed GraphLab	**32 EC2 machines**	**80 secs**

0.3% of Hadoop time

Summary: **GraphLab**

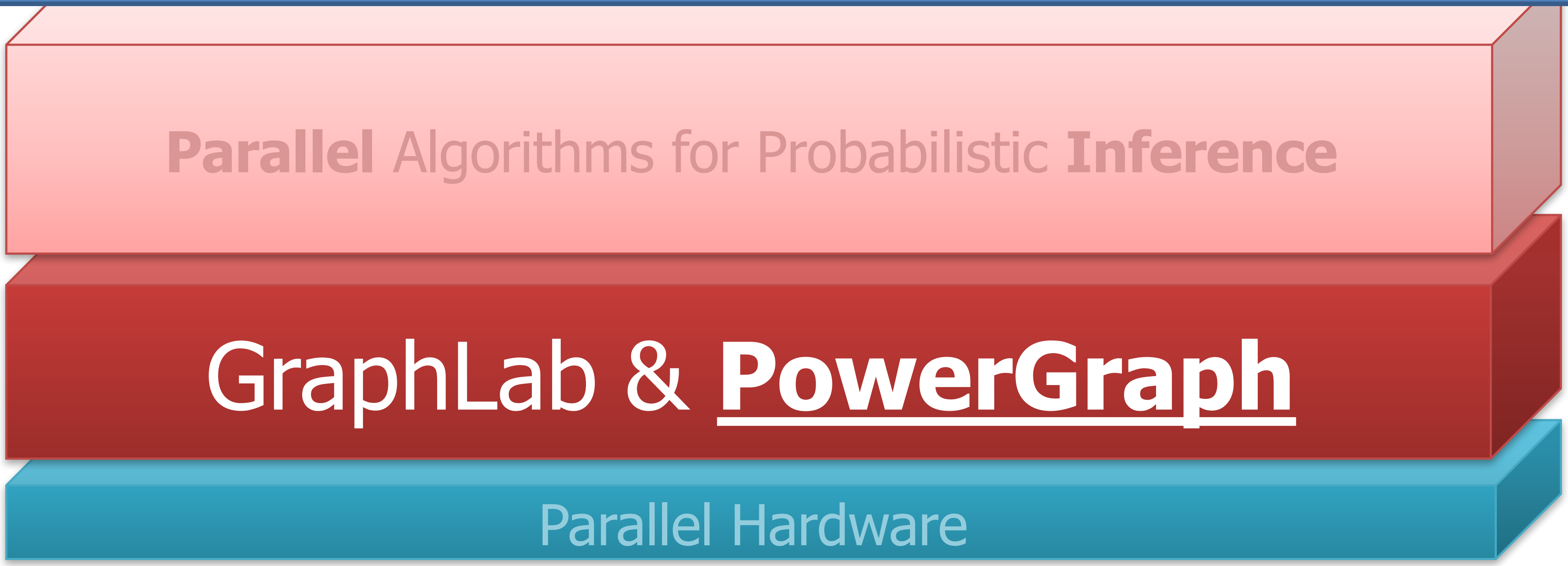
- **Generalizes** the GrAD Methodology
  - ALS, BP, Gibbs, Lasso, CoEM, SVM, PageRank, LDA, …
- **Simplifies** the *design* and *implementation* of GrAD Algorithms
- Substantially outperforms existing systems
- Key Contributions:
  - Formalized the graph-parallel setting
  - Isolates computation from movement of data
  - Strong serializability guarantees
  - Evaluation on a wide range of algorithms

Thus far…

GraphLab provided exciting scaling performance

But…

We couldn’t scale up to Altavista Webgraph from 2002 1.4B vertices, 6.6B edges



Parallel Algorithms for Probabilistic **Inference**

GraphLab & **<u>PowerGraph</u>**

Parallel Hardware

Joint Work With

Yucheng Low Aapo Kyrola Haijie Gu Danny Bickson

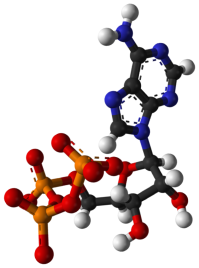
Carlos Guestrin Joe Hellerstein Guy Blelloch David O'Hallaron

Published Results

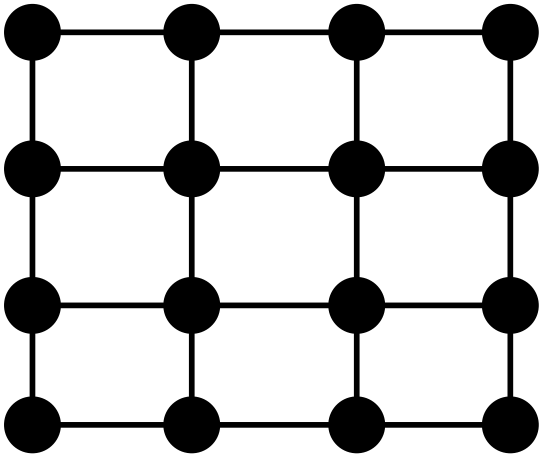
OSDI'12

Natural Graphs

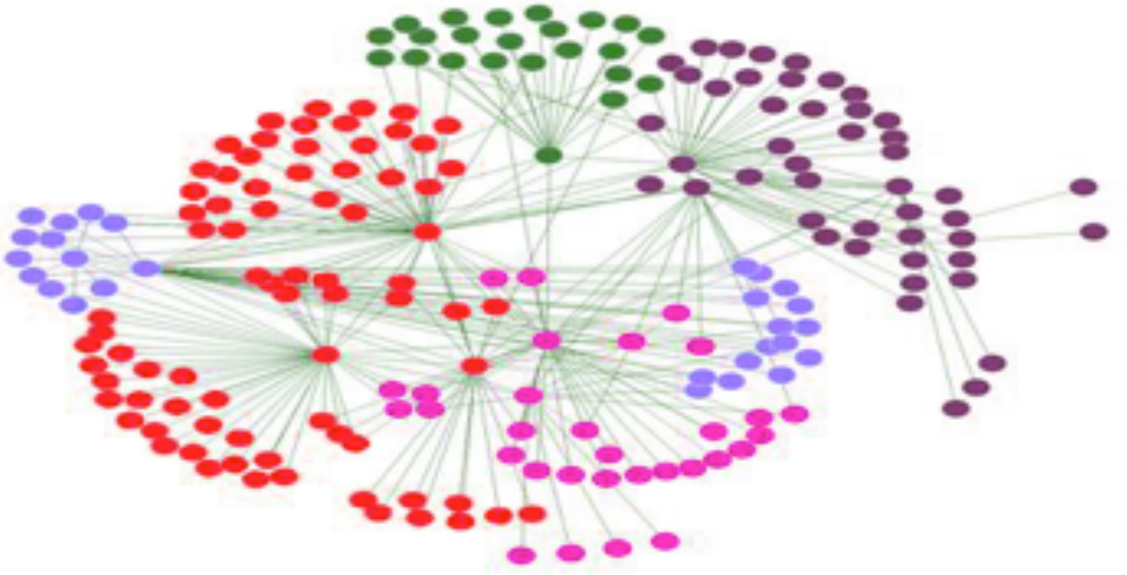
Graphs derived from natural phenomena



Properties of Natural Graphs



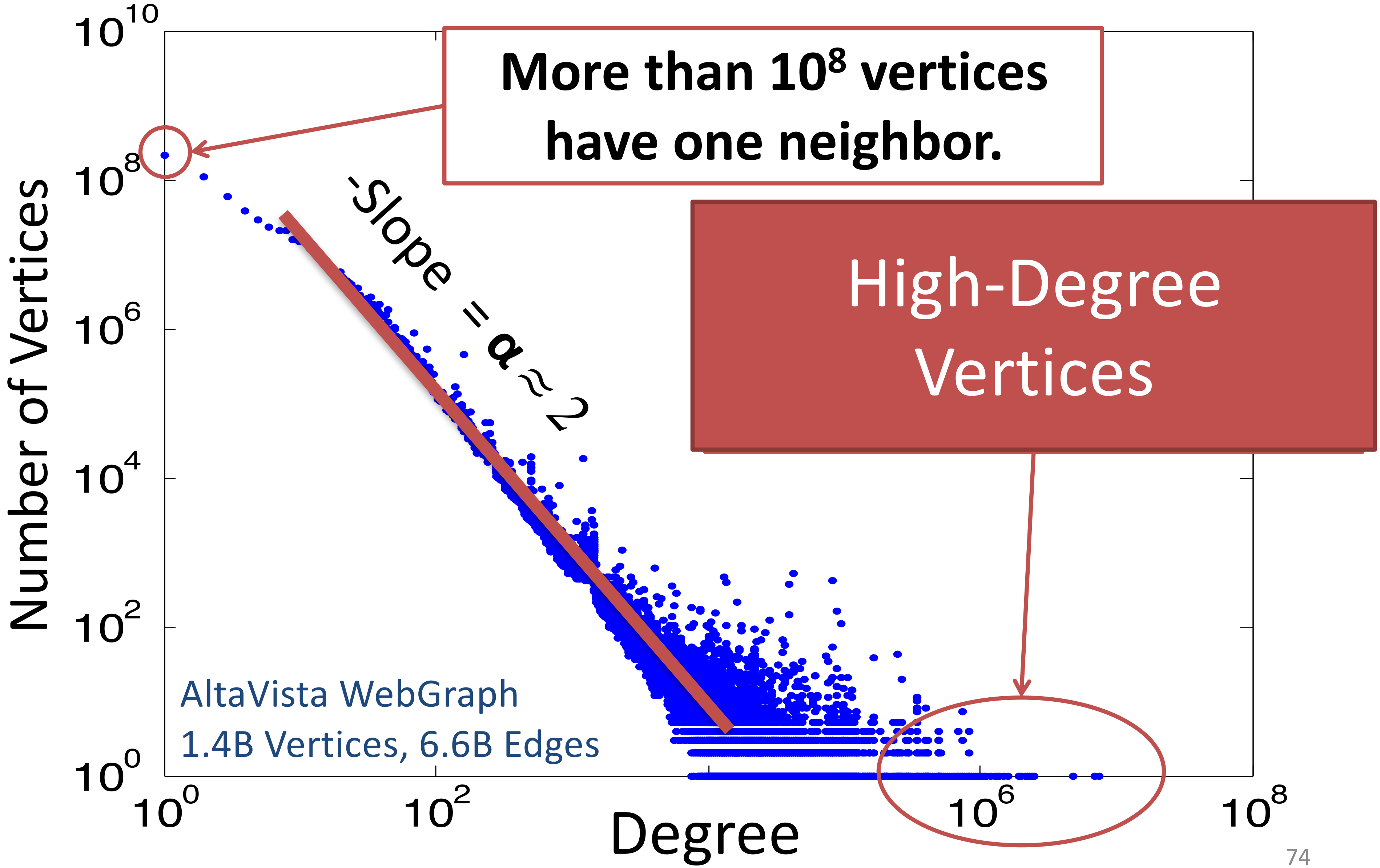
Regular Mesh



Natural Graph

Power-Law Degree Distribution

Power-Law Degree Distribution



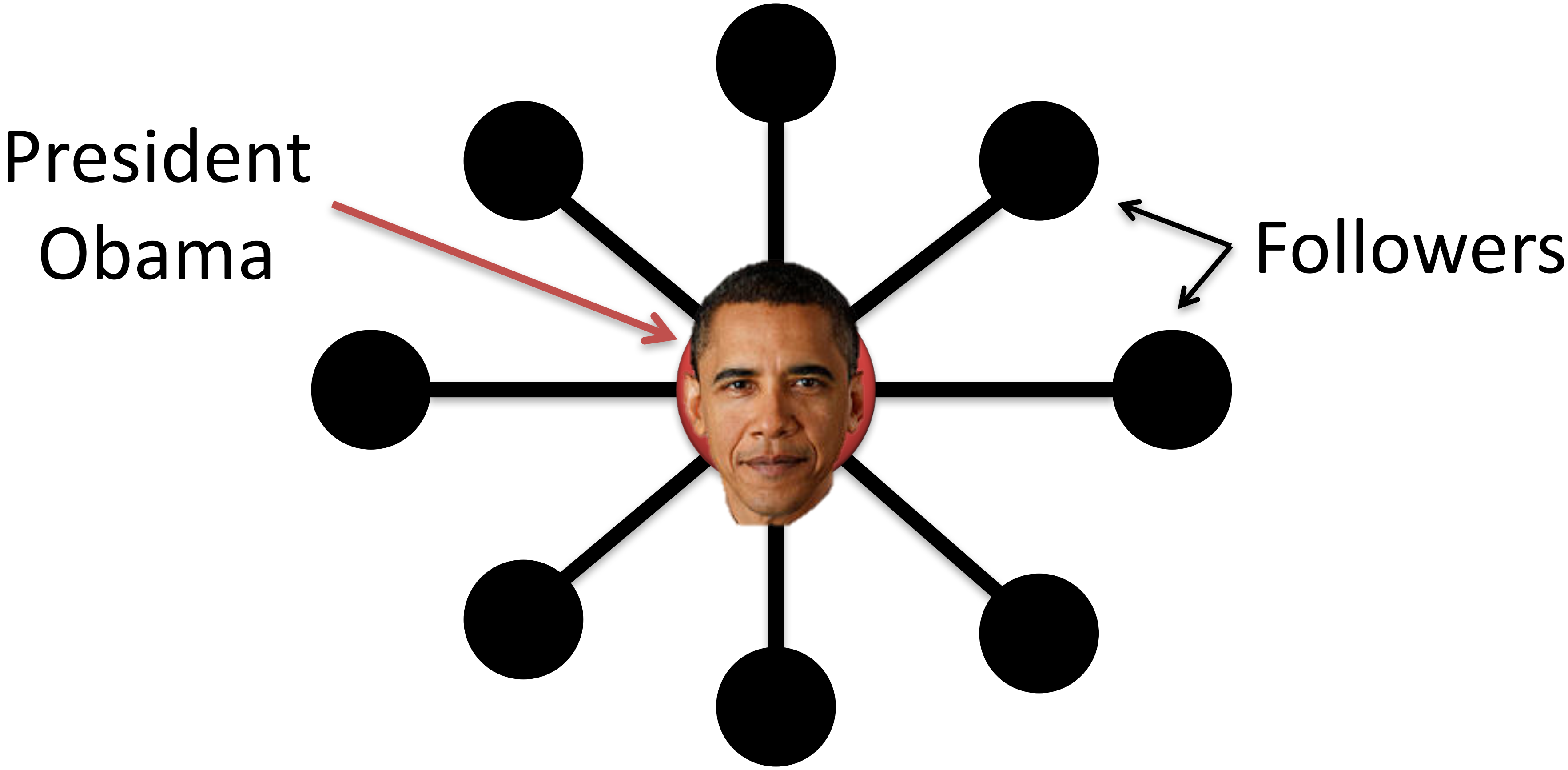
More than 10^8 vertices have one neighbor.

High-Degree Vertices

AltaVista WebGraph
1.4B Vertices, 6.6B Edges

Power-Law Degree Distribution

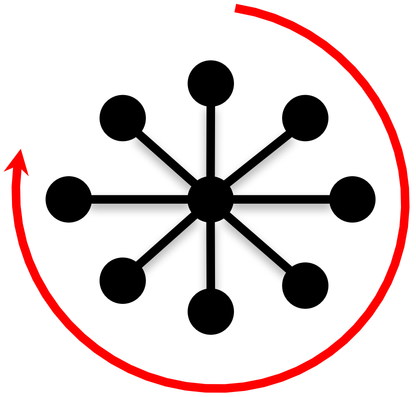
“**Star Like**” Motif



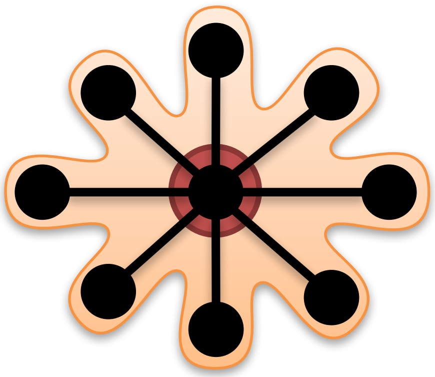
President Obama

Followers

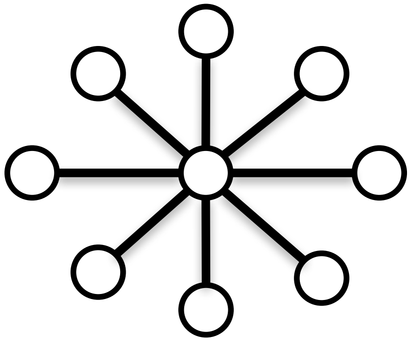
Challenges of **High-Degree** Vertices



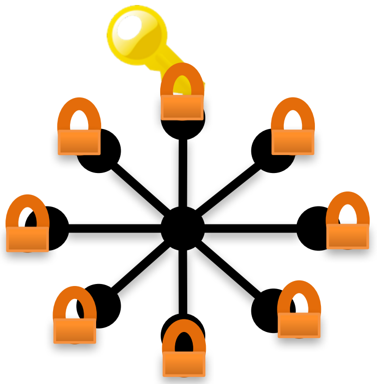
Sequentially process edges



Touches a large fraction of graph



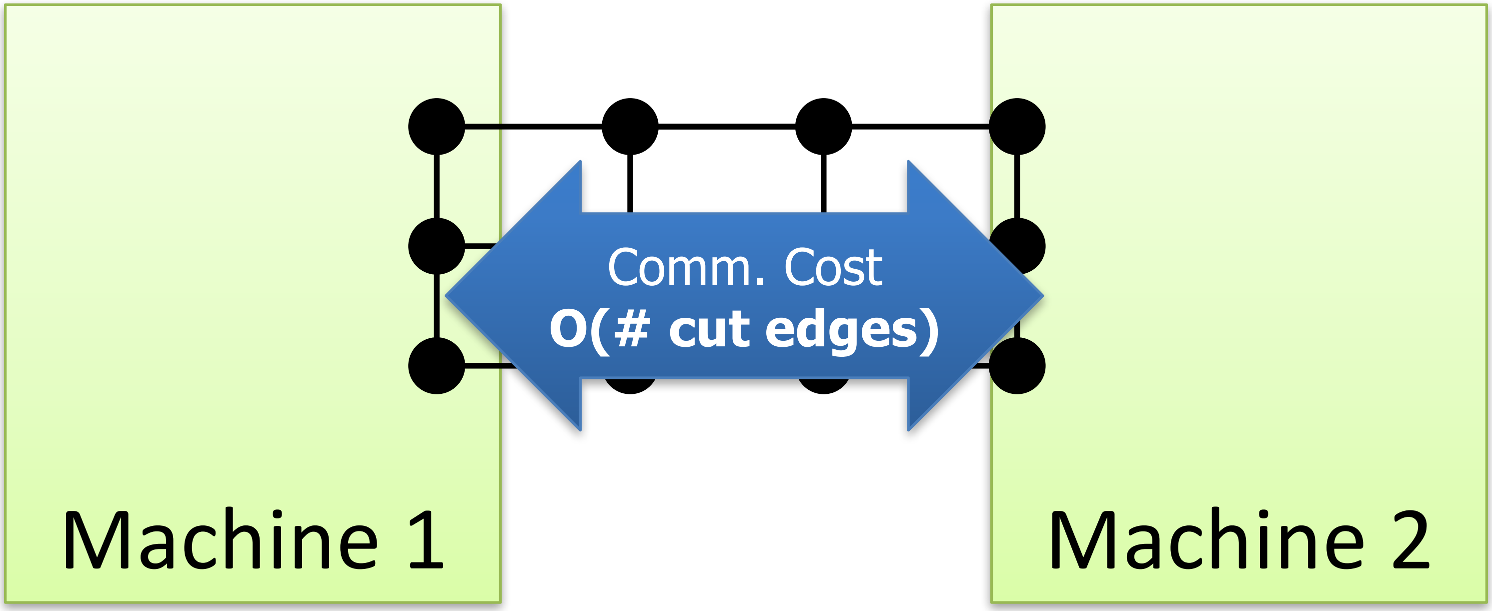
Edge meta-data too large for single machine



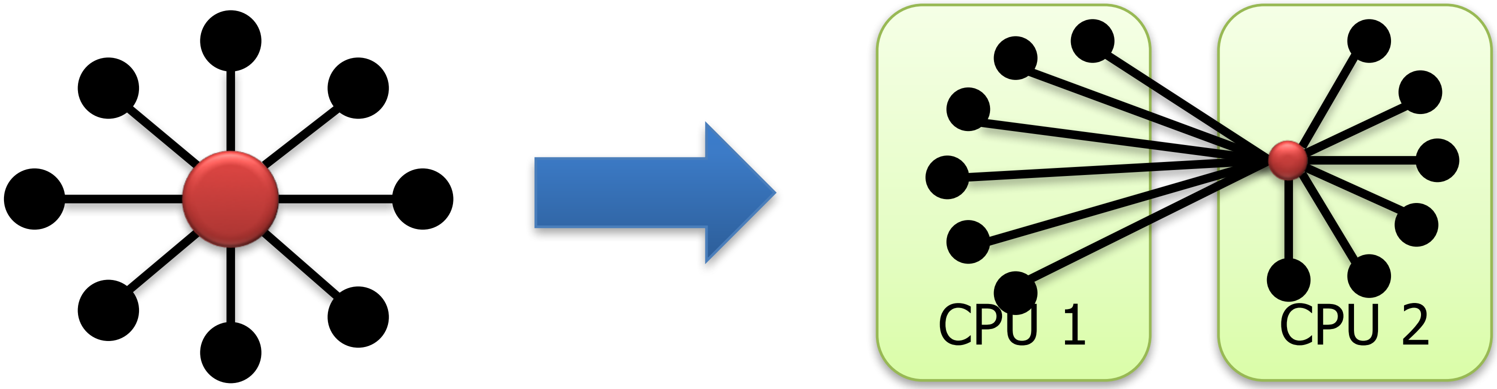
Serializability Requires **Heavy Locking**

Graph Partitioning

- Graph parallel abstractions rely on partitioning:
  - Minimize communication
  - Balance computation and storage



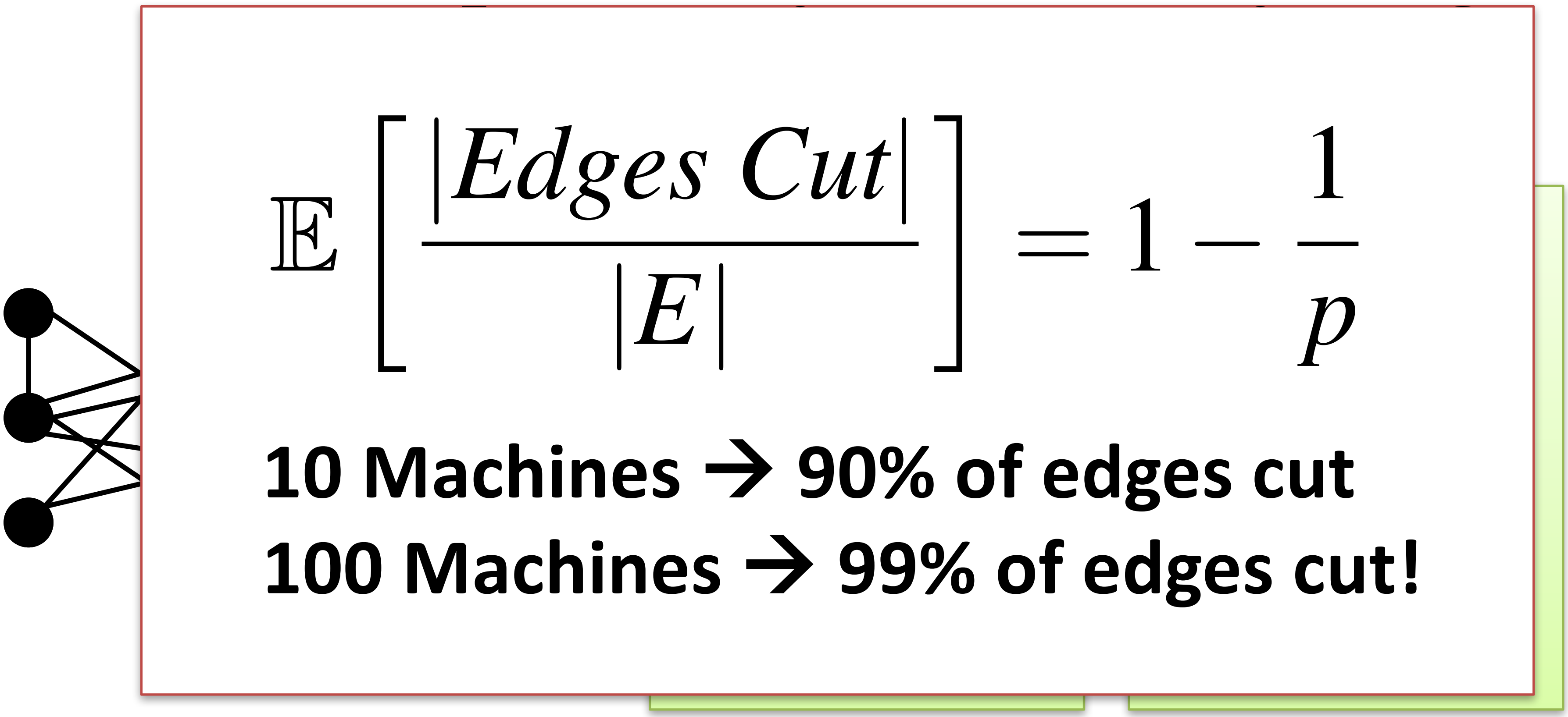
Power-Law Graphs are **Difficult to Partition**



- Power-Law graphs do not have **low-cost** balanced cuts *[Leskovec et al. 08, Lang 04]*
- Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs. *[Abou-Rjeili et al. 06]*

Random Partitioning

- GraphLab resorts to **random** (hashed) partitioning on **natural graphs**



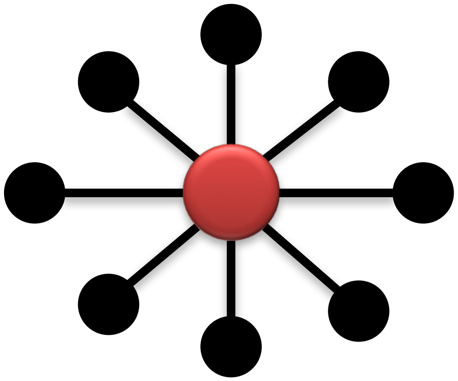
$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|E|}\right] = 1 - \frac{1}{p}$$

10 Machines → 90% of edges cut

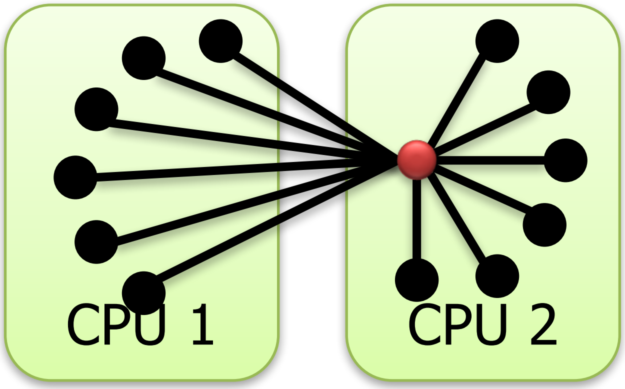
100 Machines → 99% of edges cut!

In Summary

GraphLab is not well suited for **natural graphs**



Challenges of **high-degree vertices**

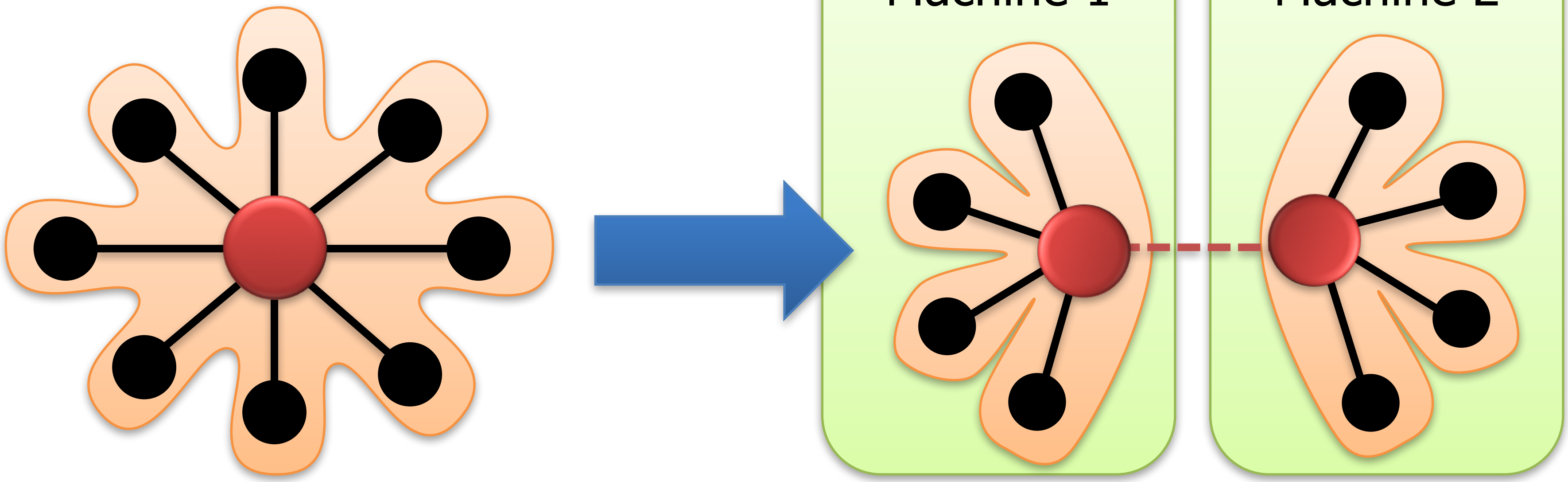


Low quality **partitioning**

PowerGraph

Program For This

Run on This



- Split **High-Degree** vertices
- **New Abstraction →** ***Equivalence*** *on Split Vertices*

A **Common Pattern** for Vertex-Programs

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * $w_{ji}$
```

Gather Information About Neighborhood

```
  // Update the PageRank
  R[i] = total
```

Update Vertex

```
  // Trigger neighbors to run again
  priority = |R[i] – oldR[i]|
  if R[i] not converged then
    signal neighbors(i) with priority
```

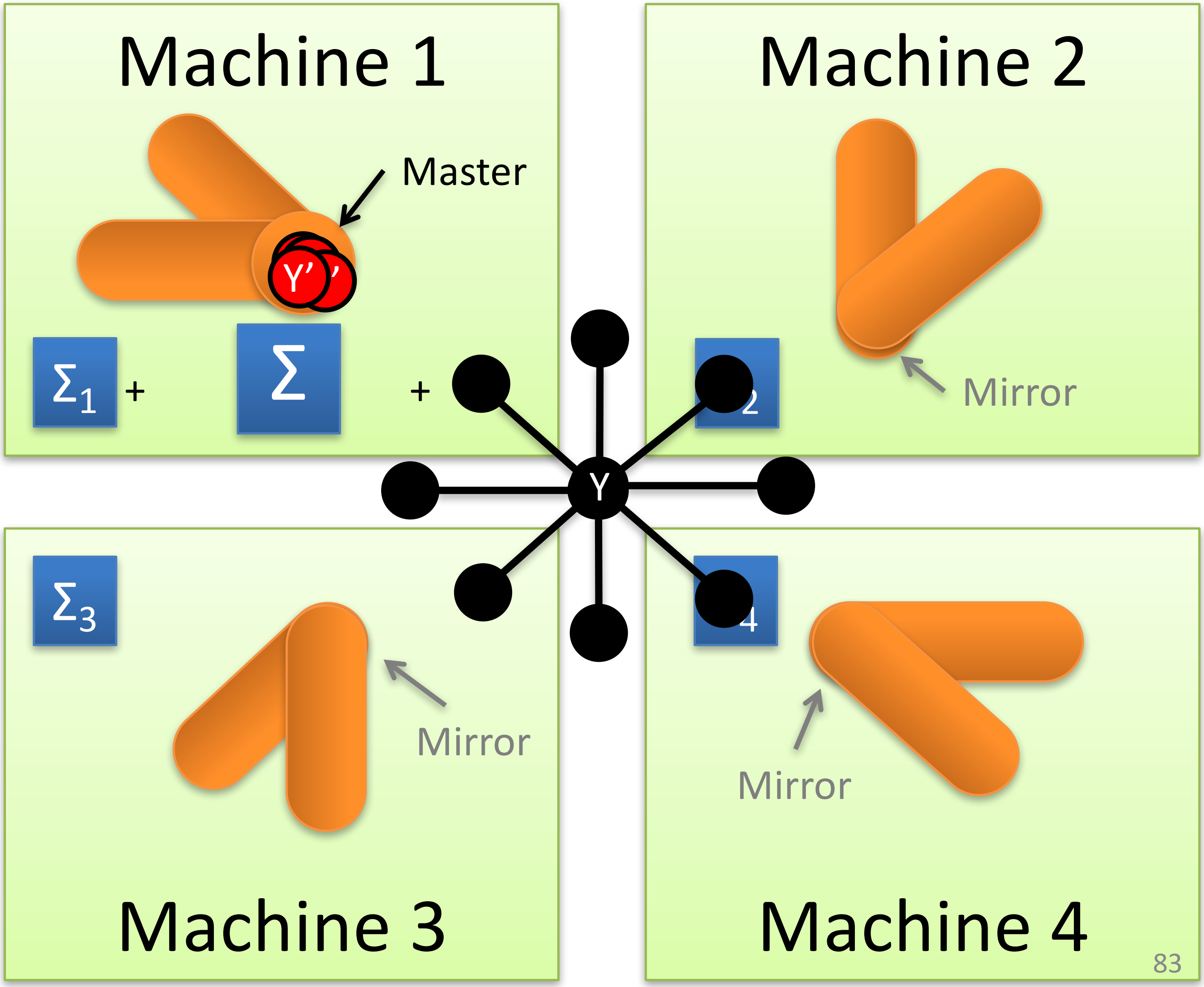
Signal Neighbors & Modify Edge Data

GAS Decomposition

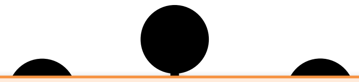
Gather

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

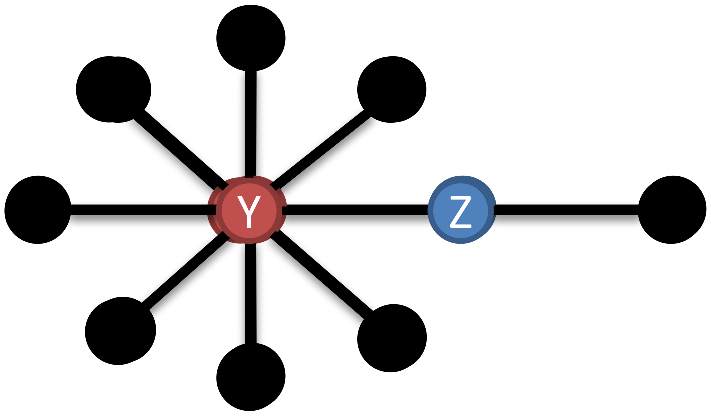
Machine 3

Balanced Vertex-Cut

Y Spans 3 Machines

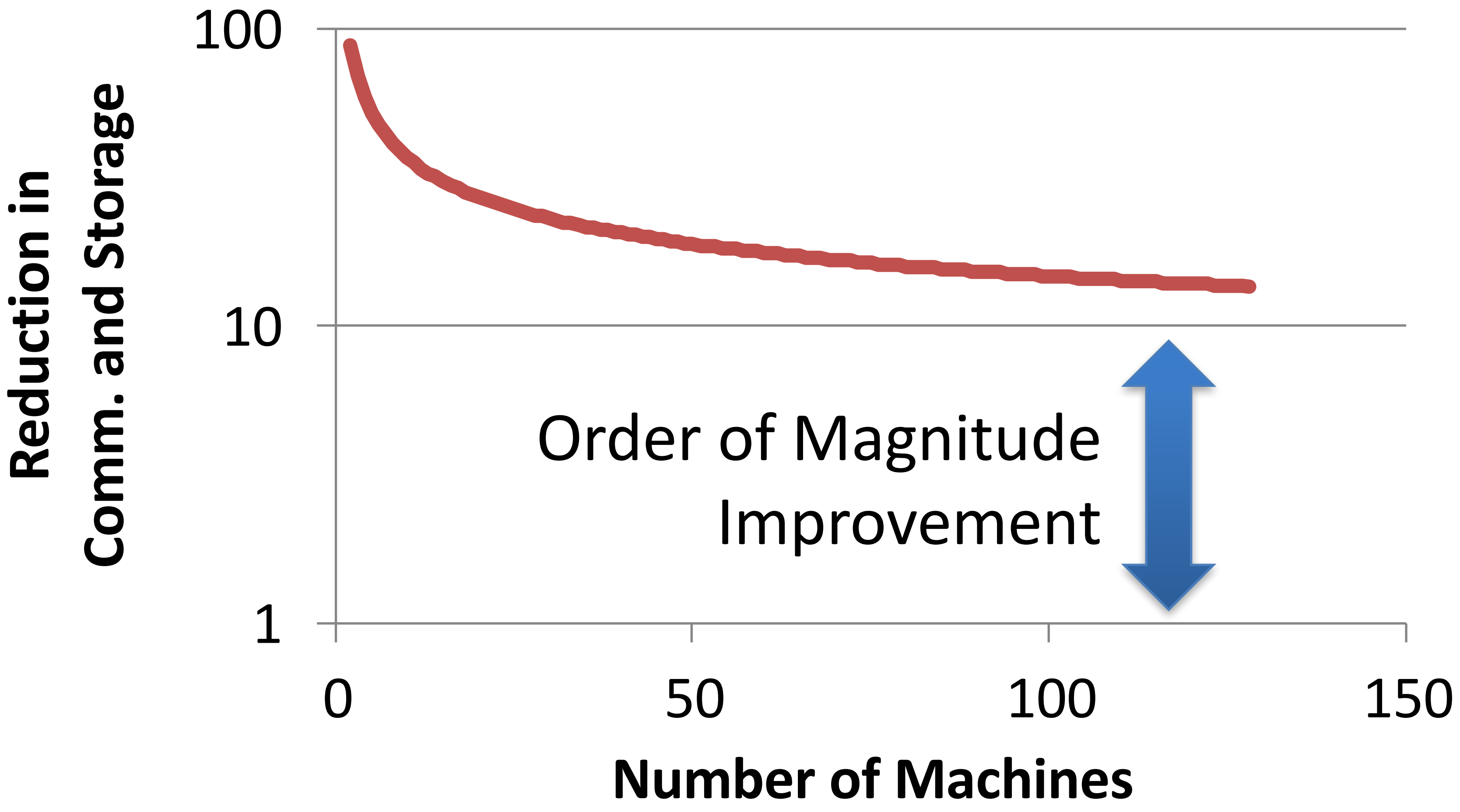
Z Spans 2 Machines

● **Not cut!**



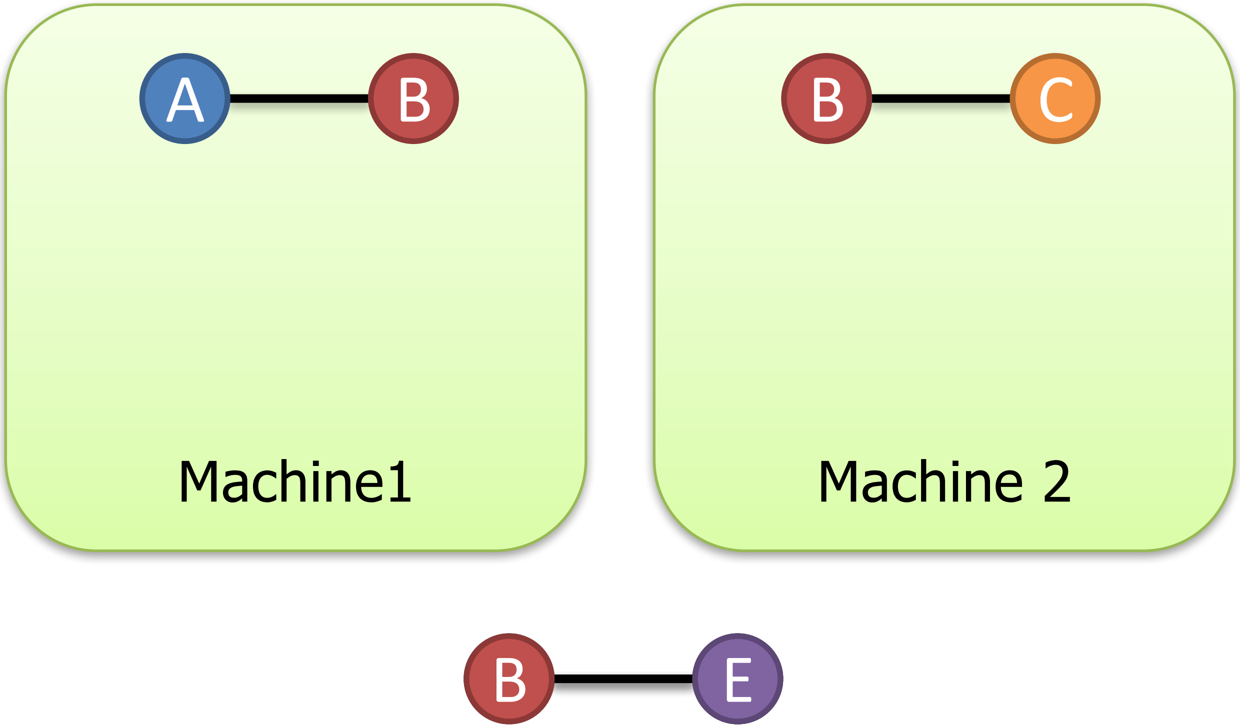
Random Vertex-Cuts vs. Edge-Cuts

- Expected improvement from vertex-cuts:

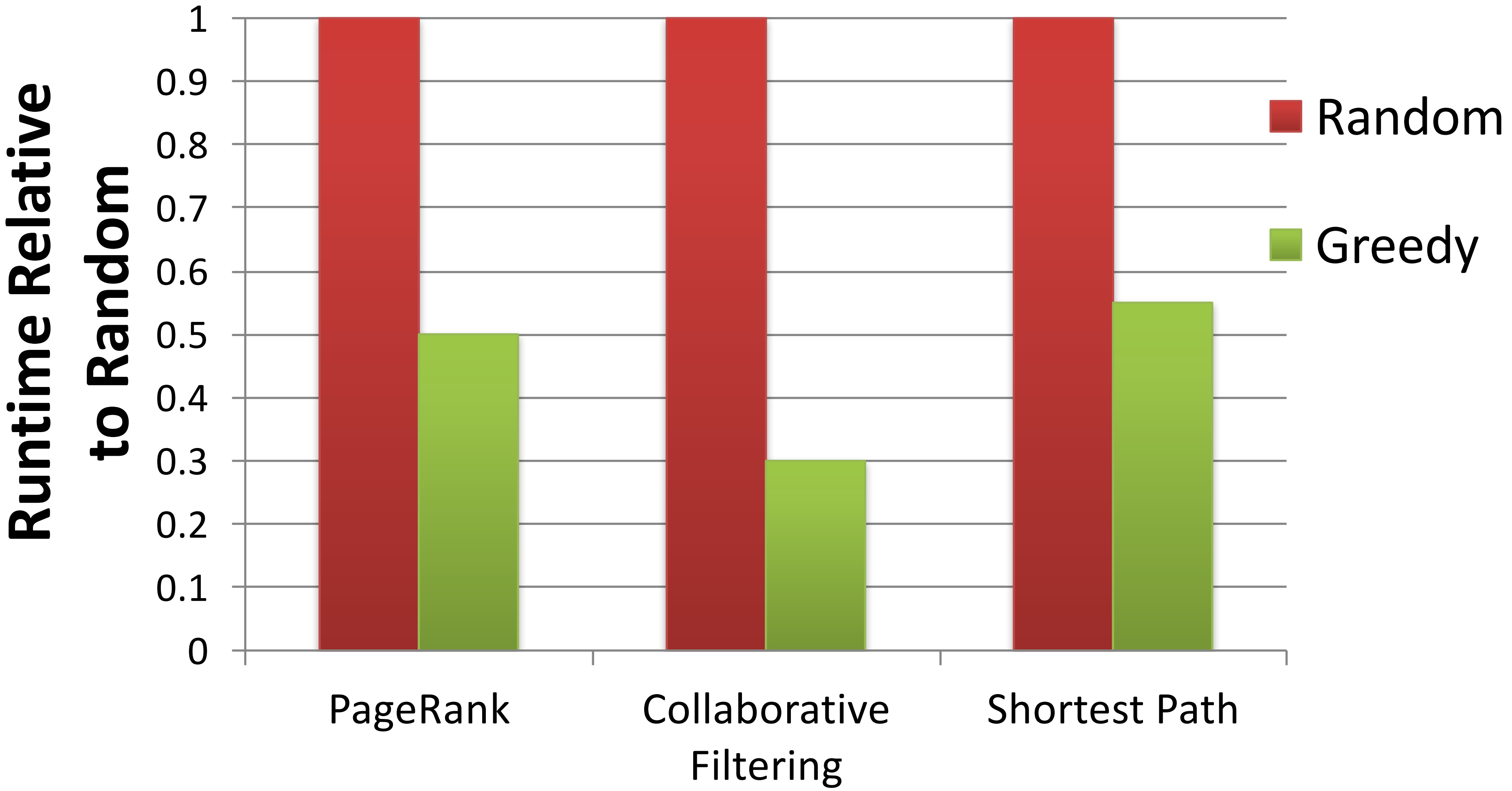


Greedy Vertex-Cuts

- Place edges on machines which already have the vertices in that edge.

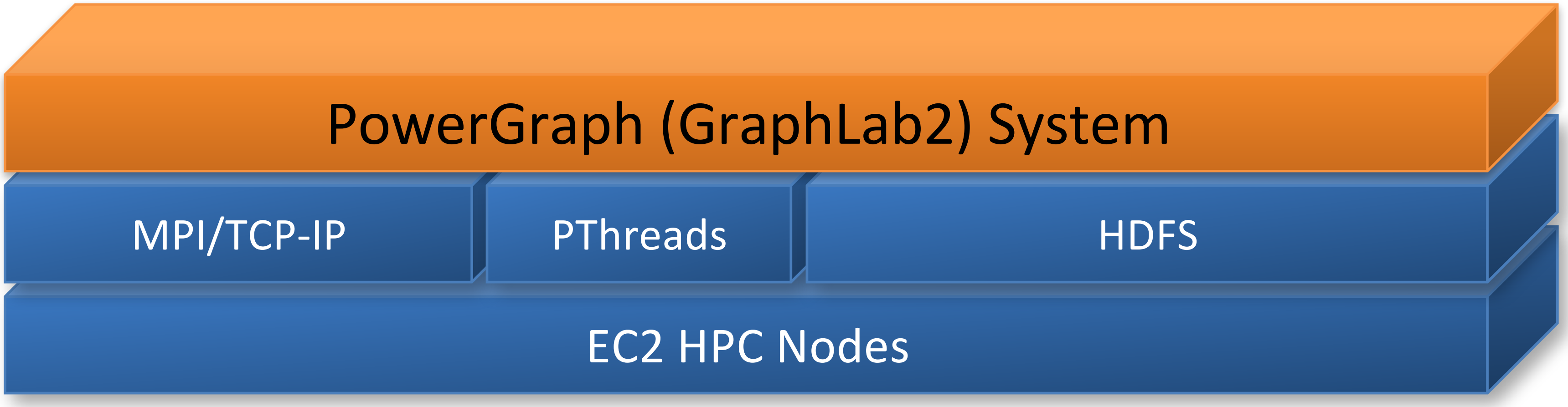


Greedy Vertex-Cuts Improve Performance



Greedy partitioning improves computation performance.

System Design

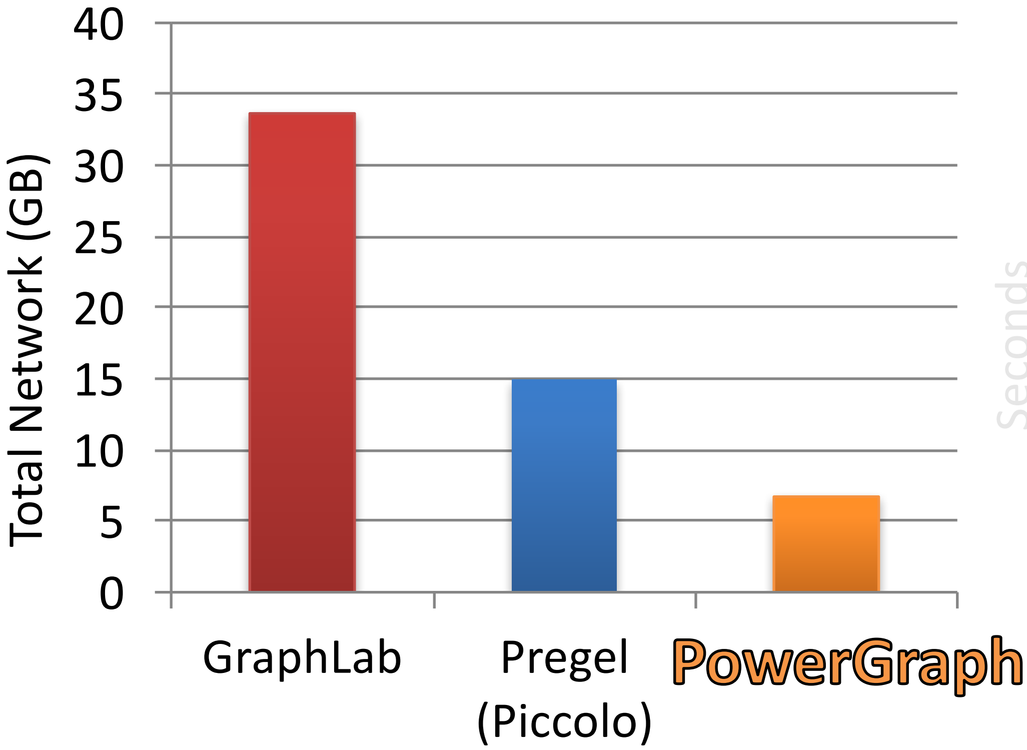


- Implemented as C++ API
- Uses HDFS for Graph Input and Output
- Fault-tolerance is achieved by check-pointing
  - Snapshot time < 5 seconds for twitter network

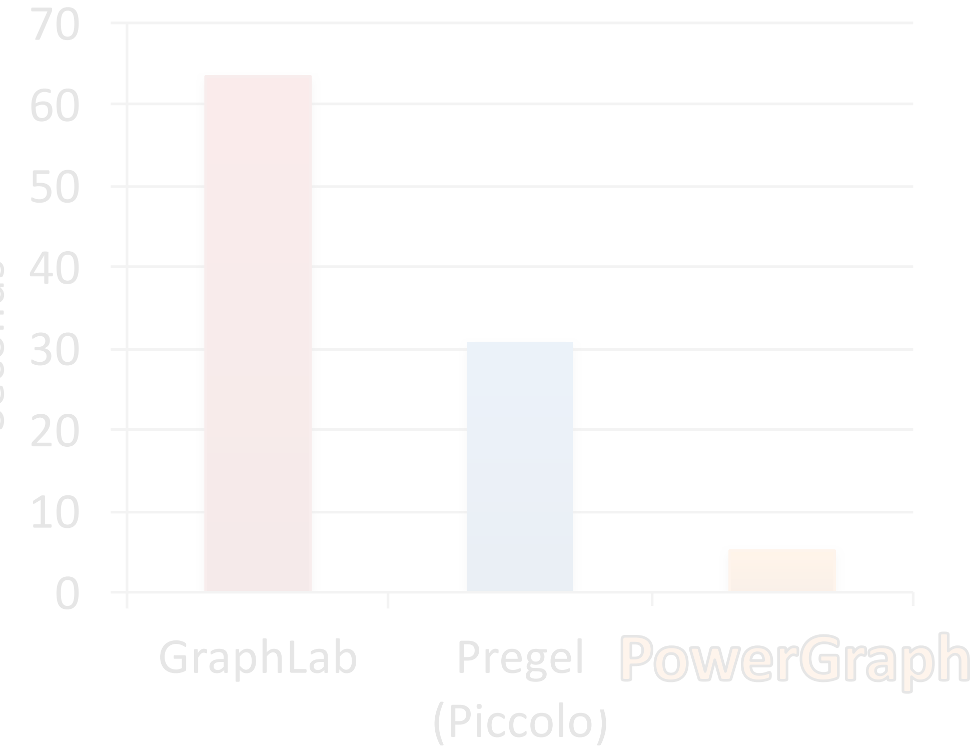
PageRank on the Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

Communication



Runtime



Reduces Communication

Runs Faster

32 Nodes x 8 Cores (EC2 HPC cc1.4x)

PowerGraph is Scalable

Yahoo Altavista Web Graph (2002):

One of the largest publicly available web graphs

1.4 Billion Webpages, 6.6 Billion Links

7 Seconds per Iter.

1B links processed per second

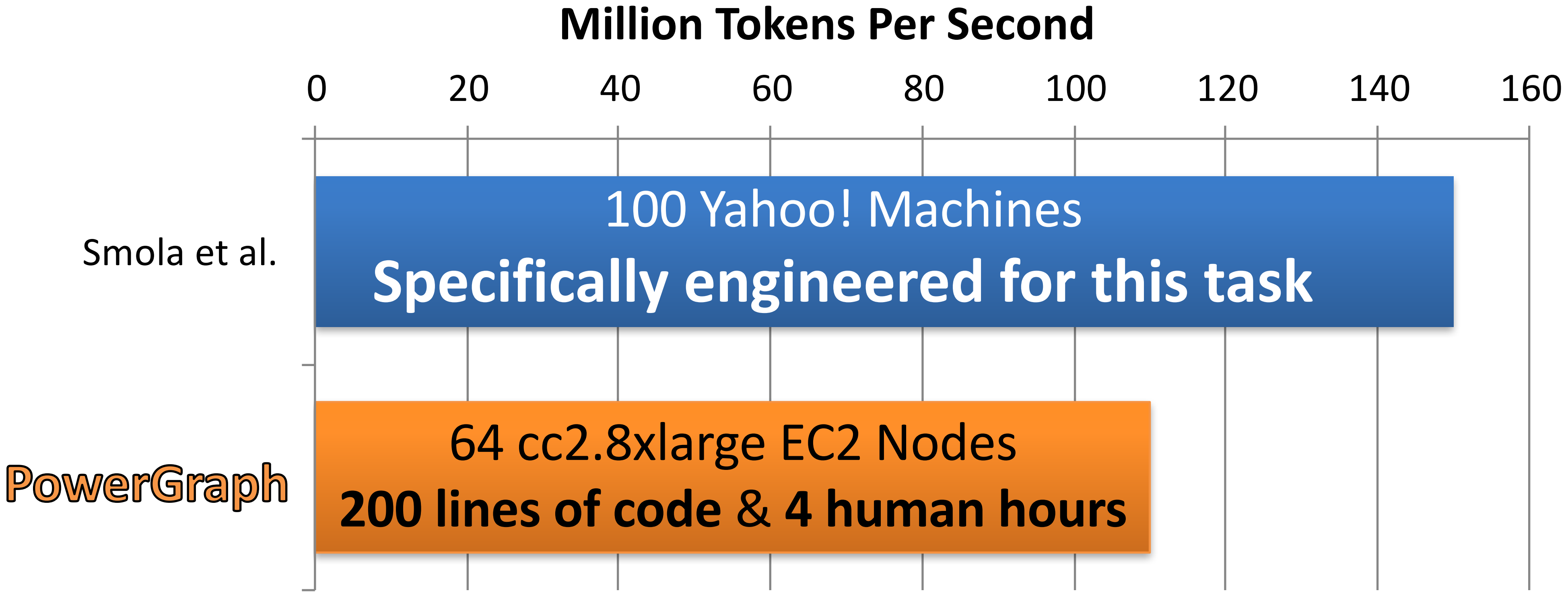
30 lines of user code

64 HPC Nodes

(1024 Cores (2048 HT)

Topic Modeling

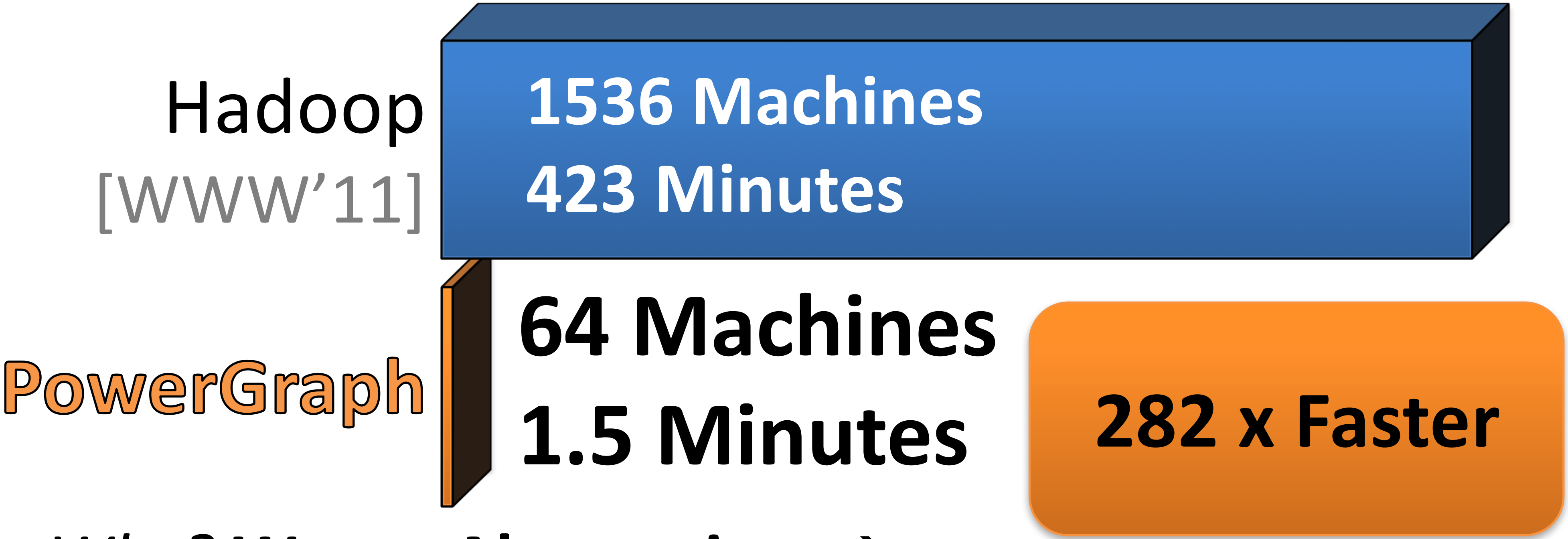
- English language Wikipedia
  - 2.6M Documents, 8.3M Words, 500M Tokens
  - Computationally intensive algorithm



Triangle Counting on The Twitter Graph

Identify individuals with **strong communities.**

Counted: 34.8 Billion Triangles



Hadoop
[WWW'11]

1536 Machines
423 Minutes

PowerGraph

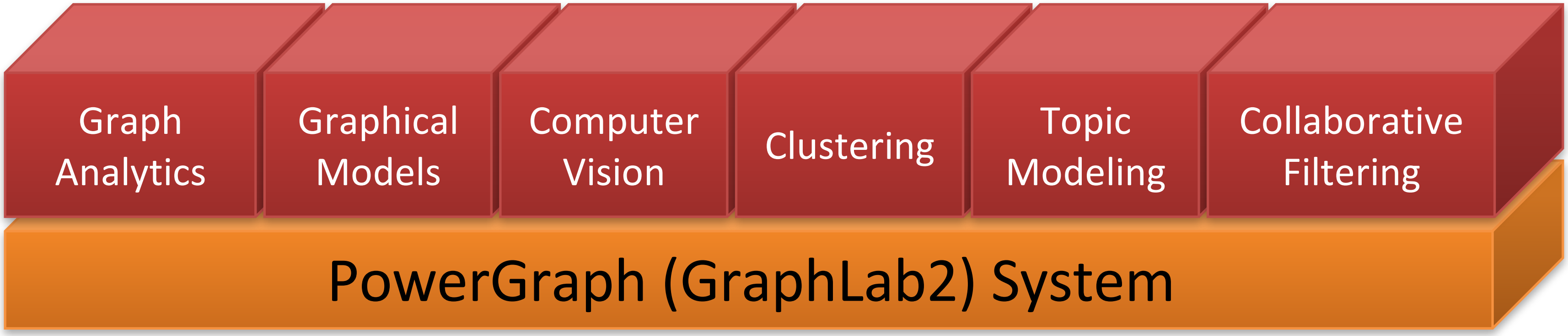
64 Machines
1.5 Minutes

282 x Faster

Why? **Wrong Abstraction** →
Broadcast O(degree^2) messages per Vertex

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

Machine Learning and **Data-Mining** Toolkits



Graph Analytics

Graphical Models

Computer Vision

Clustering

Topic Modeling

Collaborative Filtering

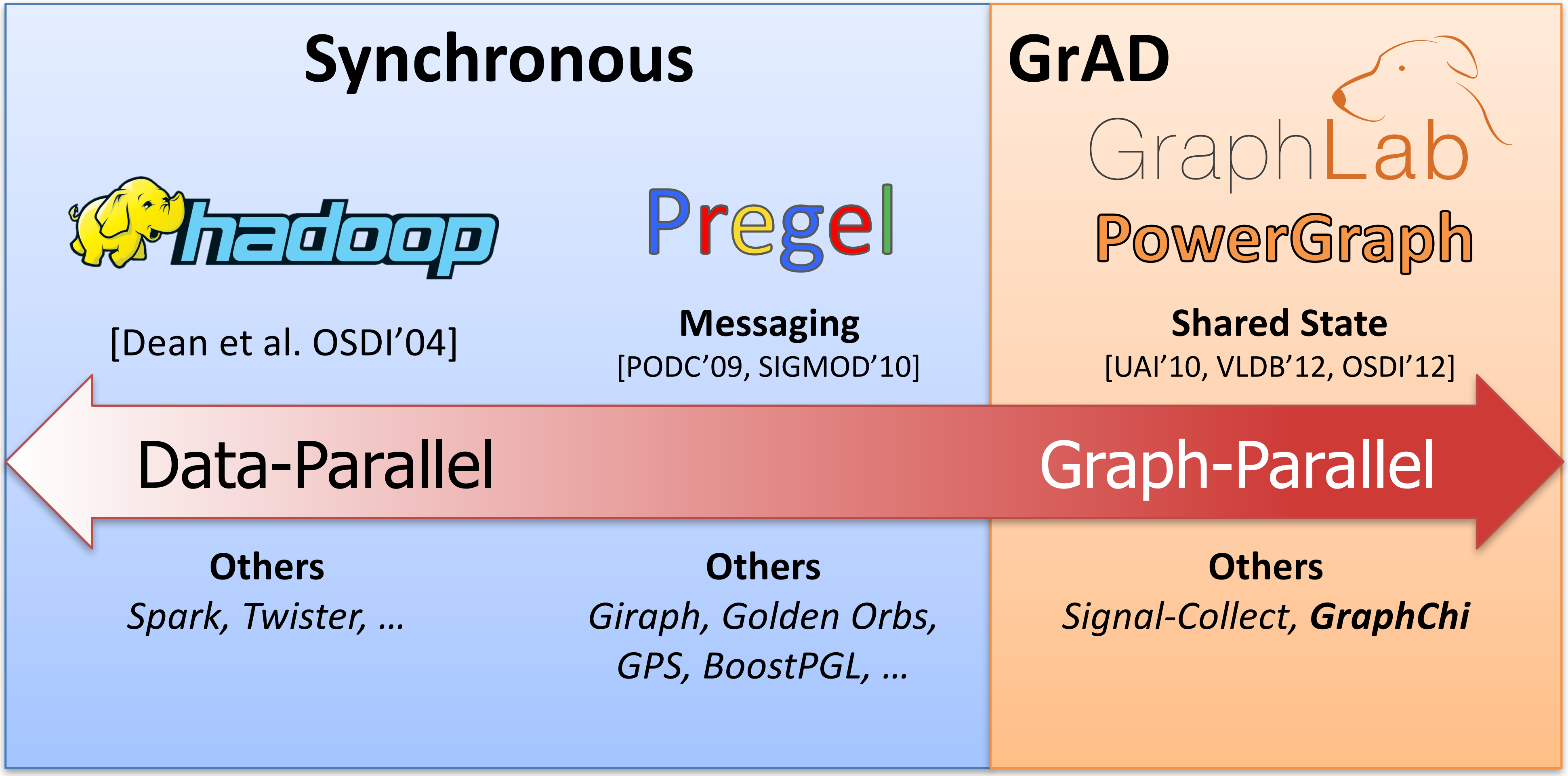
PowerGraph (GraphLab2) System

Demonstrates the Applicability of the GrAD Methodology

Summary: **PowerGraph**

- Identify the **challenges** of Natural Graphs
  - High-degree vertices, Low-quality edge-cuts
- Solution **PowerGraph** System
  - **GAS Decomposition**: split vertex programs
  - **Vertex-partitioning**: distribute natural graphs
- PowerGraph **theoretically** and **experimentally** outperforms existing graph-parallel systems.

*Related **High-Level** Abstractions*



Synchronous

hadoop

[Dean et al. OSDI'04]

Pregel

Messaging
[PODC'09, SIGMOD'10]

GrAD

GraphLab

PowerGraph

Shared State
[UAI'10, VLDB'12, OSDI'12]

Data-Parallel

Graph-Parallel

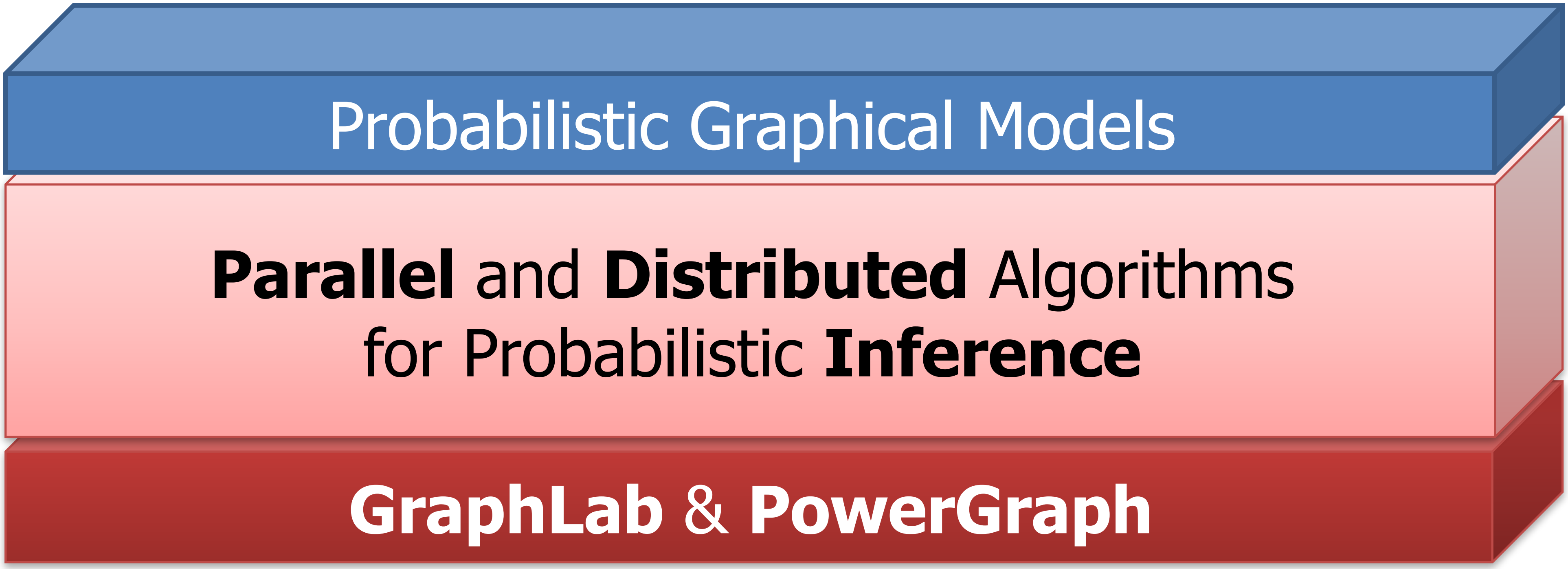
Others
Spark, Twister, …

Others
Giraph, Golden Orbs, GPS, BoostPGL, …

Others
*Signal-Collect, **GraphChi***

facebook flickr YouTube

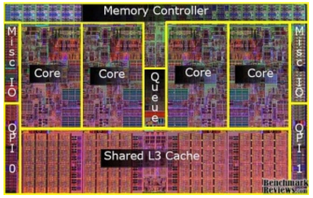
Massive Structured Problems


Probabilistic Graphical Models

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

GraphLab & **PowerGraph**

Advances Parallel Hardware



amazon webservices™

Thesis Statment

*Efficient **parallel** and **distributed** systems for probabilistic reasoning follow the **GrAD** **Methodology***

1. **Graphically** decomposition:
   - Expose parallelism and distribute state
2. **Asynchronous** scheduling
   - Improved convergence and correctness
3. **Dynamic** prioritization
   - Eliminated wasted work

Observations

- *Graphical models* encode **statistical**, **computational**, and **parallel** structure
- *Tradeoff:* **Convergence** and **Parallelism**
  - Many things can be computed in parallel
  - Not all parallel computation is productive
- **Approximation** →*Increased* **Parallelism**
  - τ_{ε}-approximation, approximate sampling
- Power of high-level abstractions
  - Enables the exploration of GrAD methodology

Future: Declarative Models

- Models are *recursive relationships*
  - BP, Gibbs Sampling, PageRank, …

My Interests

Sum of my friends interests

$$A[x_i] = a \left(\sum_{j \in \mathcal{N}[i]} g(A[x_i], A[x_i, x_j], A[x_j]) \right)$$

"Closeness"

number of overlapping posts

$$A[x_i, x_j] = s(A[x_i], A[x_i, x_j], A[x_j])$$

- System determines the optimal schedule

Future: **Online** Probabilistic Reasoning

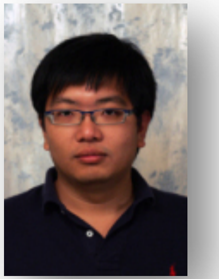
- The world is rapidly evolving:
  - Make friends and rate movies in real-time
- How do we define and maintain models?
  - **Declarative specification:** *time invariant*
  - τ_{ε}*-approximation:* **small** change → **local** effect
- Exploit ***Power-Law*** structure in change
  - Popular items are rated more frequently
  - Exploit burstiness for better caching

Contributions & Broader Impact

- Theoretically and experimentally characterized
  - Importance of **dynamic asynchronous** scheduling
  - Effect of model **structure** and **parameters** on parallelism
  - Effect of **approximation accuracy** on parallelism
  - Tradeoff between **parallelism** and **convergence**
- Developed two **graph-parallel** abstractions
  - **GraphLab:** vertex-centric view of computation
  - **PowerGraph:** *Distributed* vertex-centric view of computation
- Fostered a community around GraphLab/PowerGraph
  - Substantial industry and academic interest
- Built a foundation for the future design of scalable systems for probabilistic reasoning

Thank You!

Sue Ann
Hong


Yucheng
Low

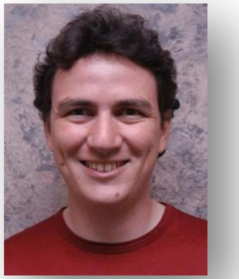
Aapo
Kyrola

Haijie
Gu

Danny
Bickson

Arthur
Gretto
n

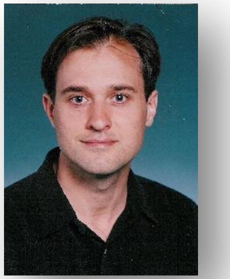
Andreas
Krause


Carlos
Guestrin

Alex
Smola

Jeff
Bilmes

David
O’Hallaron


Guy
Blelloch

Joe
Hellerstein

The Select Lab & My Family