Parallel Splash Belief Propagation

Joseph E. Gonzalez

Yucheng Low

Carlos Guestrin

David O'Hallaron

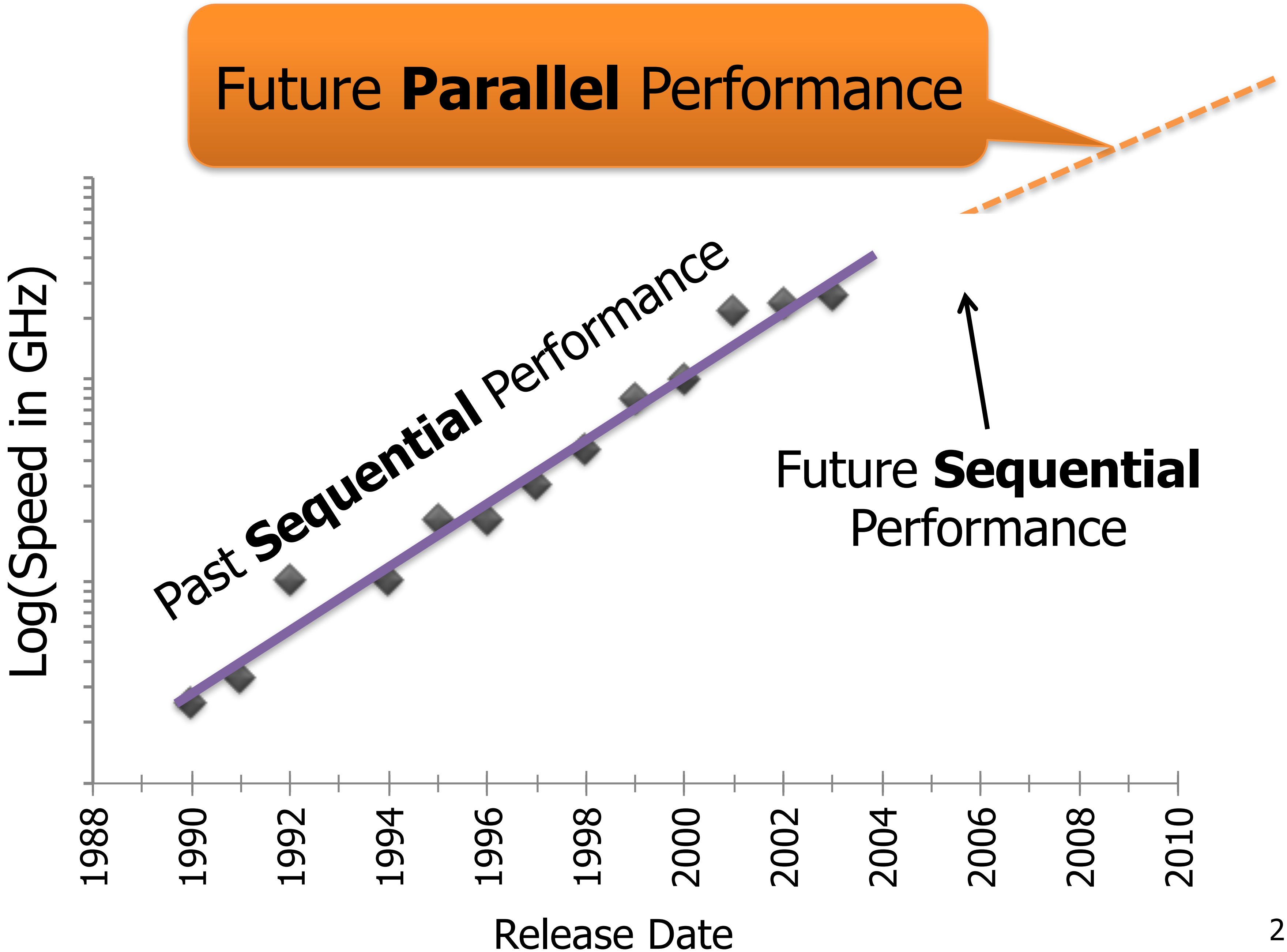
Computers which worked on this project:

BigBro1, BigBro2, BigBro3, BigBro4, BigBro5, BigBro6, BiggerBro, BigBroFS
Tashish01, Tashi02, Tashi03, Tashi04, Tashi05, Tashi06, ..., Tashi30,
parallel, gs6167, koobcam (helped with writing)

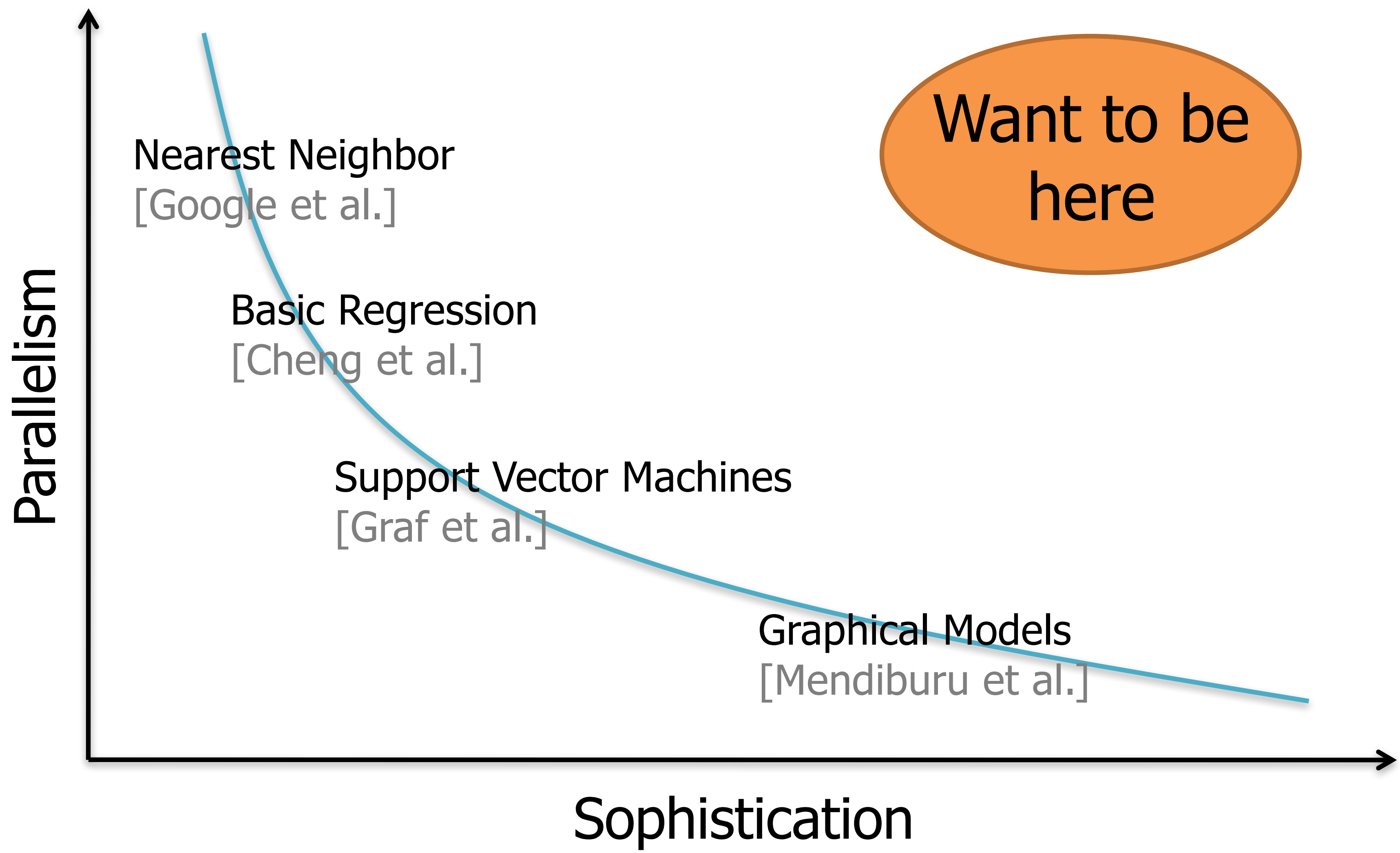
Select Lab

Carnegie Mellon

Change in the Foundation of ML



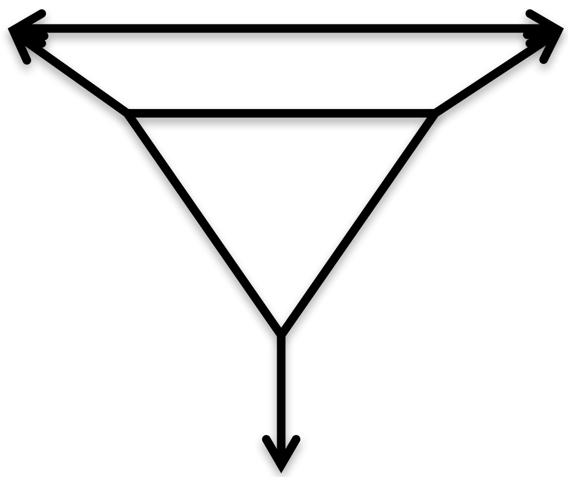
Why is this a Problem?



Why is it hard?

Algorithmic Efficiency

Eliminate wasted computation



Parallel Efficiency

Expose independent computation

Implementation Efficiency

Map computation to real hardware

sense learn act

The Key Insight

Statistical Structure

••Graphical Model Structure
••Graphical Model Parameters

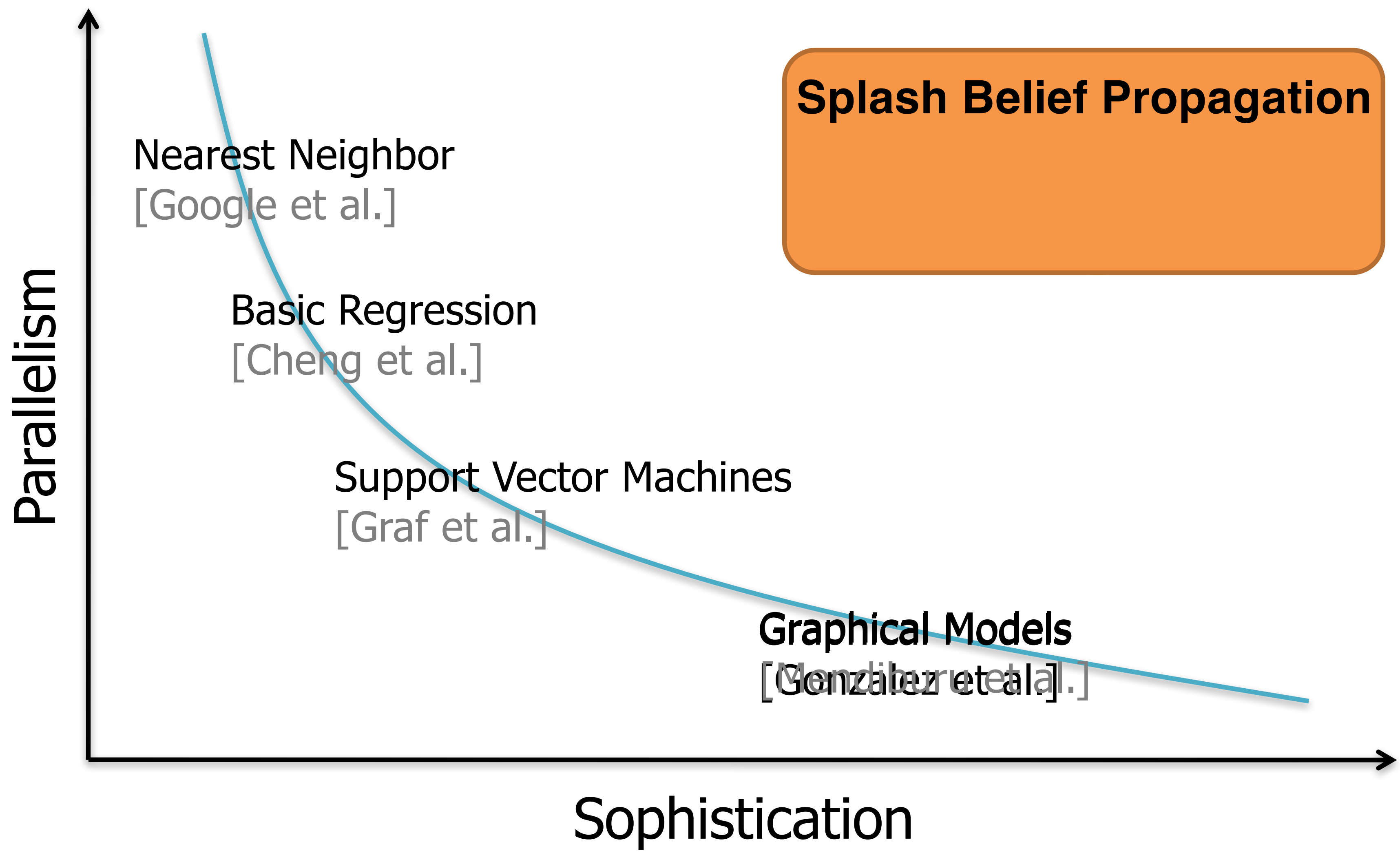
Computational Structure

••Chains of Computational Dependences
••Decay of Influence

Parallel Structure

••Parallel Dynamic Scheduling
••State Partitioning for Distributed Computation

The Result



Splash Belief Propagation
Nearest Neighbor
[Google et al.]
Basic Regression
[Cheng et al.]
Support Vector Machines
[Graf et al.]
Graphical Models
[Gonzalez et al.]
[Mendiburu et al.]
Parallelism
Sophistication

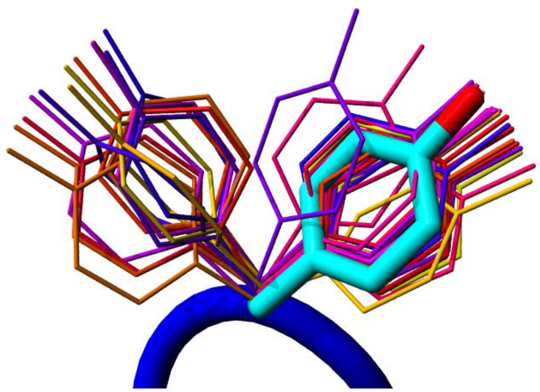
Outline

- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_ε - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

Graphical Models and Parallelism

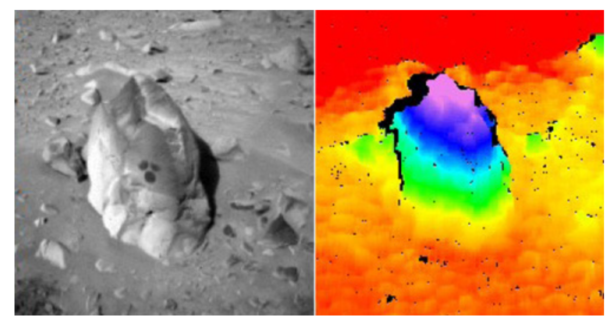
*Graphical models provide a common language for **general purpose** parallel algorithms in machine learning*

- A parallel **inference algorithm** would improve:



Protein Structure Prediction

Movie Recommendation



Computer Vision

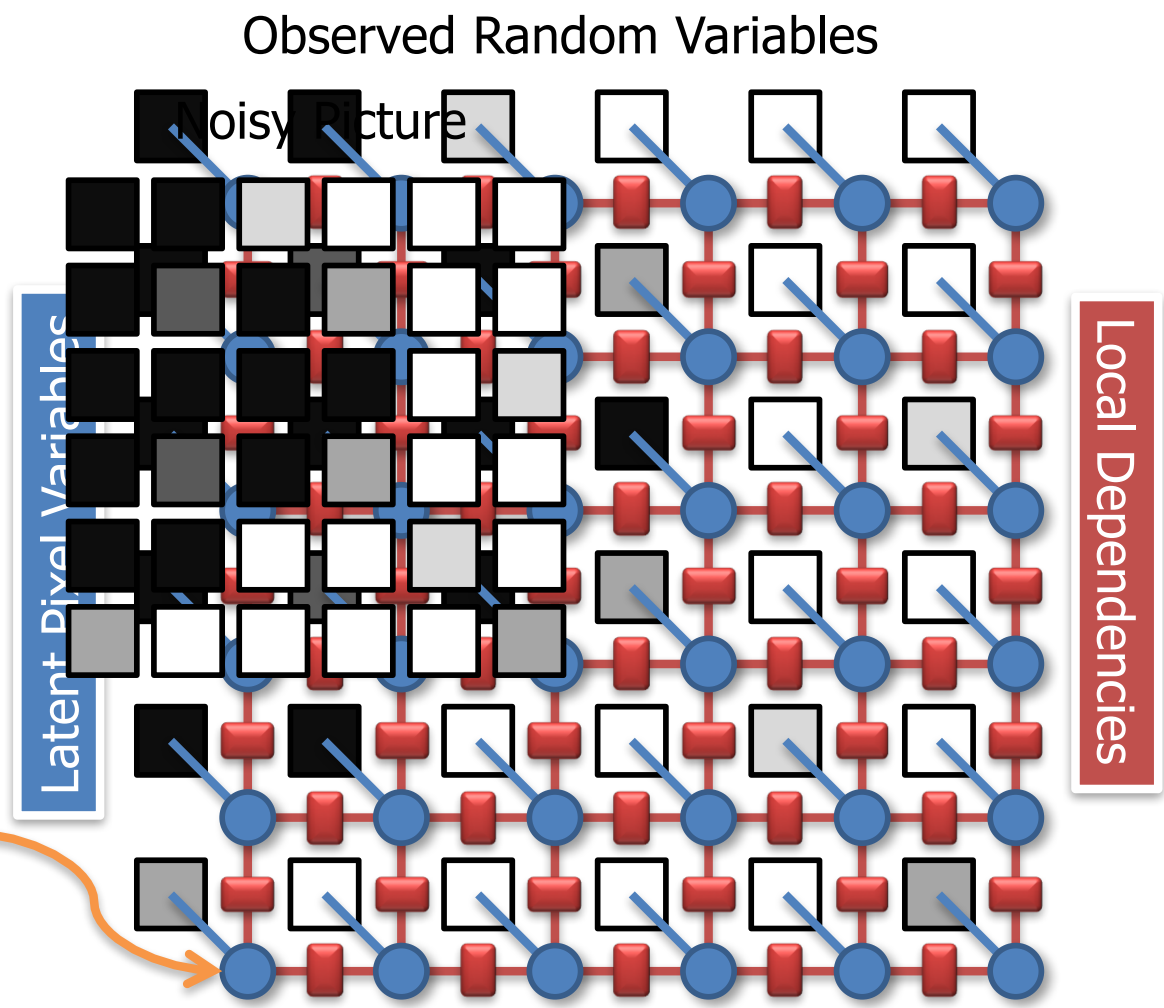
Inference is a key step in **Learning** Graphical Models

Overview of Graphical Models

- Graphical represent of local statistical dependencies

“True” Pixel Values

Continuity Assumptions

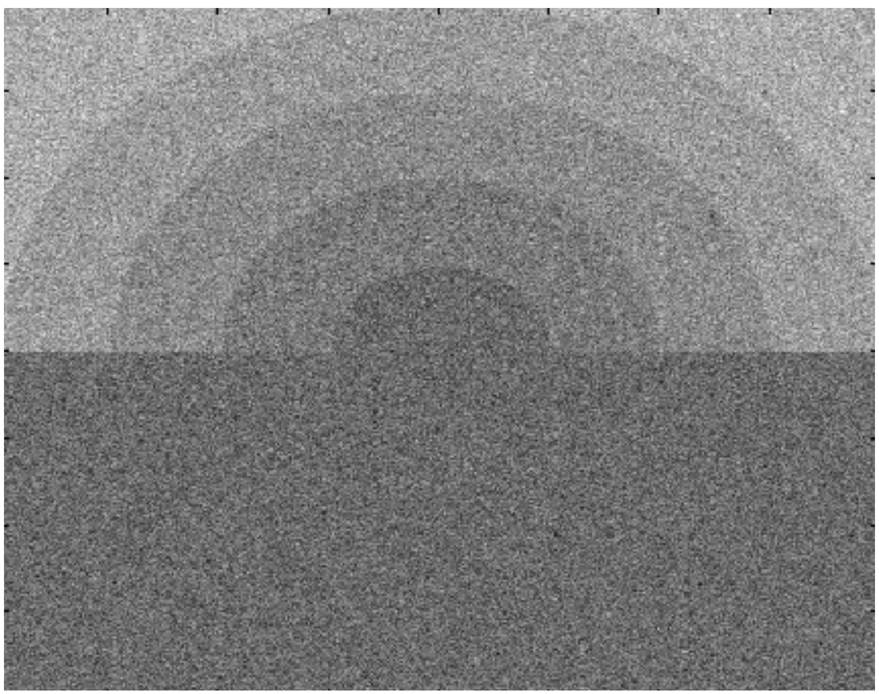


Inference

What is the probability that this pixel is black?

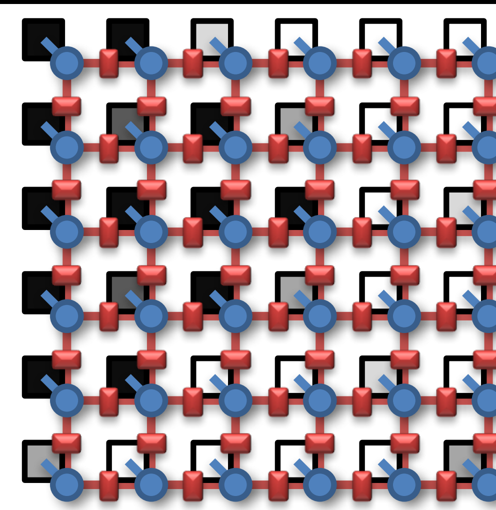
Synthetic Noisy Image Problem

Noisy Image



Predicted Image

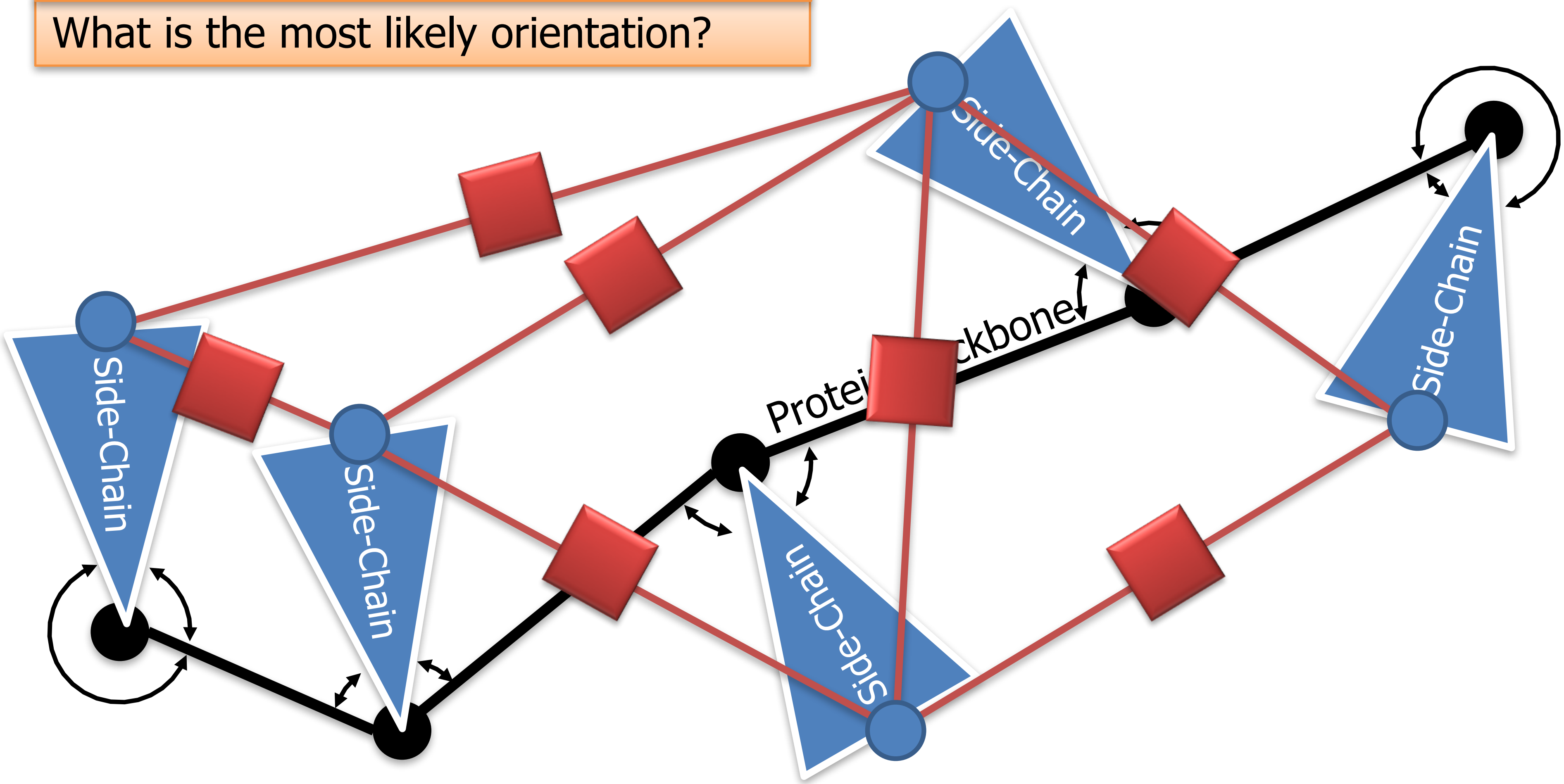
- Overlapping Gaussian noise
- Assess convergence and accuracy



Protein Side-Chain Prediction

- Model side-chain interactions as a graphical model

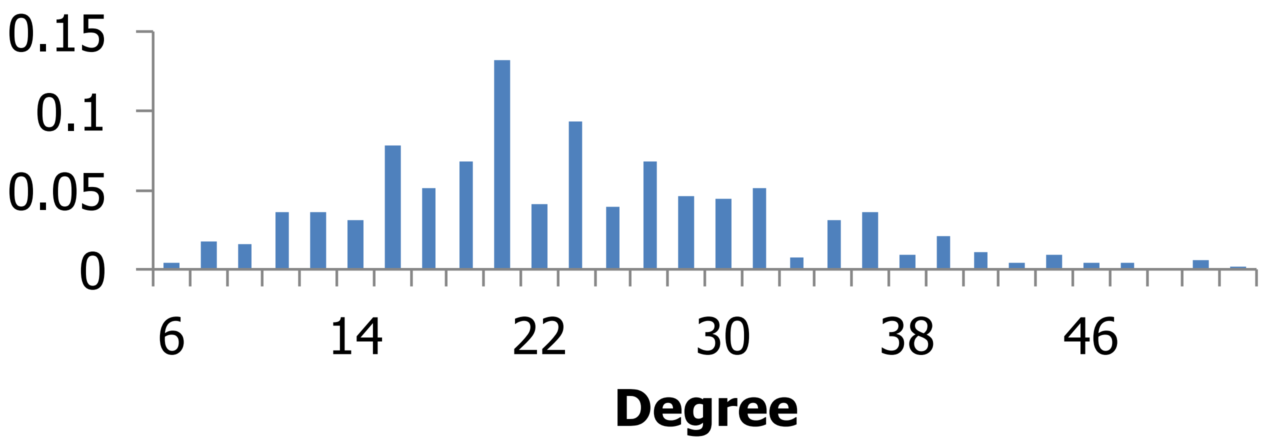
Inference
What is the most likely orientation?

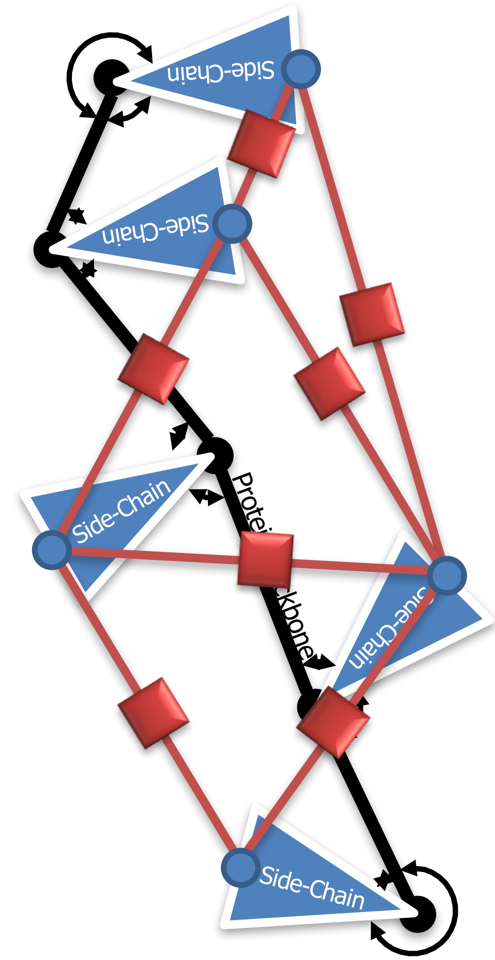


Protein Side-Chain Prediction

- 276 Protein Networks:
- Approximately:
  - 700 Variables
  - 1600 Factors
  - 70 Discrete orientations
- Strong Factors

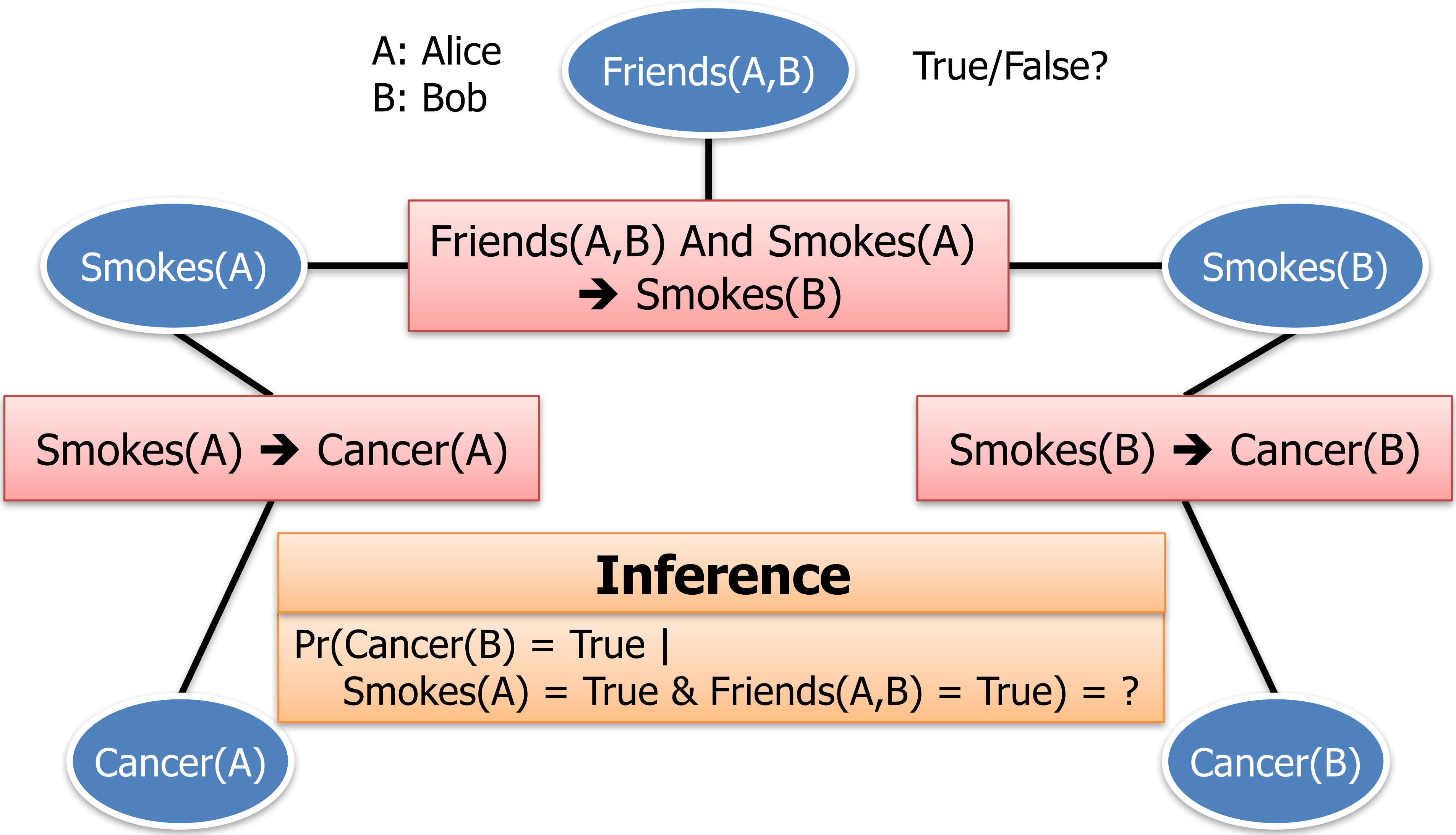
Example Degree Distribution





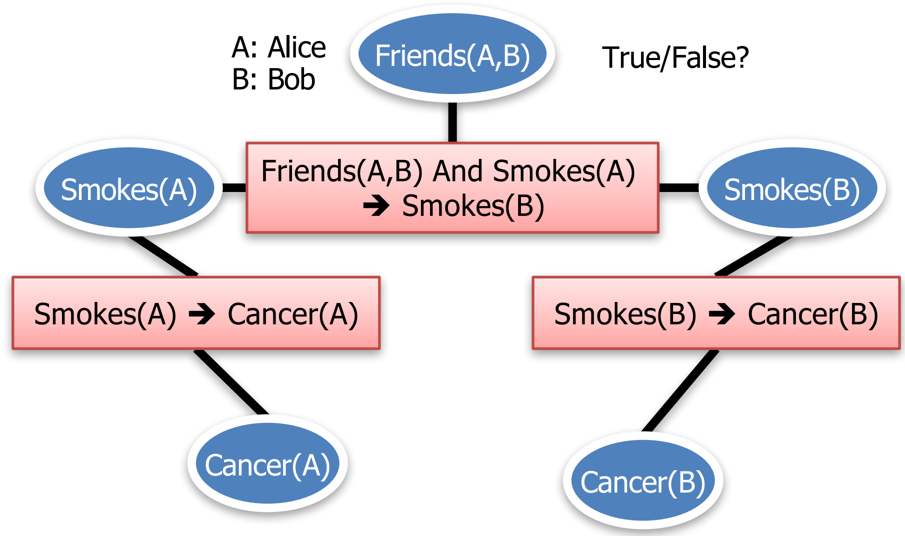
Markov Logic Networks

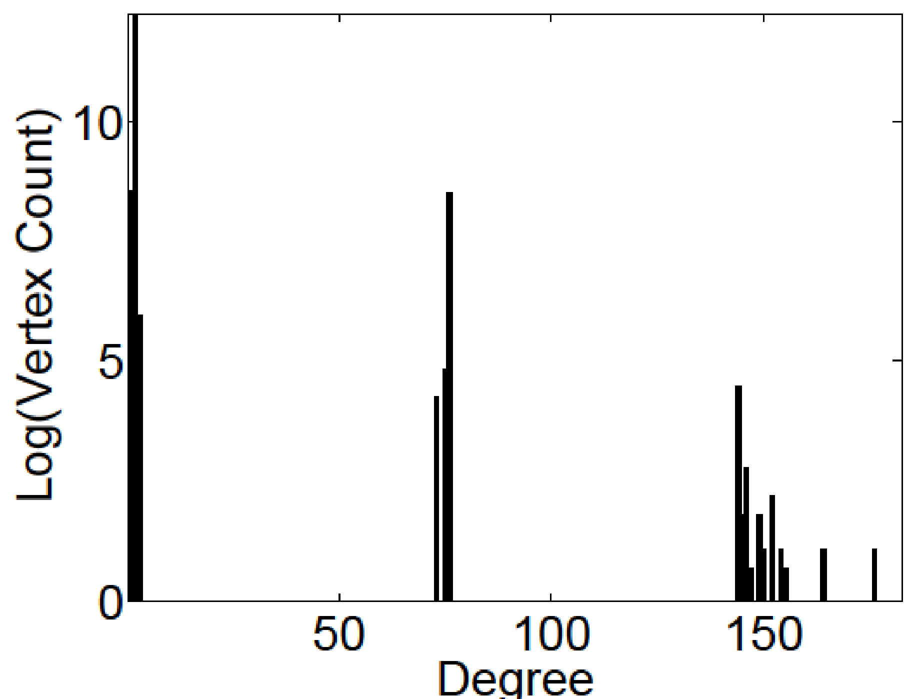
- Represent Logic as a graphical model



Markov Logic Networks

- UW-Systems Model
  - 8K Binary Variables
  - 406K Factors
- Irregular degree distribution:
  - Some vertices with high degree

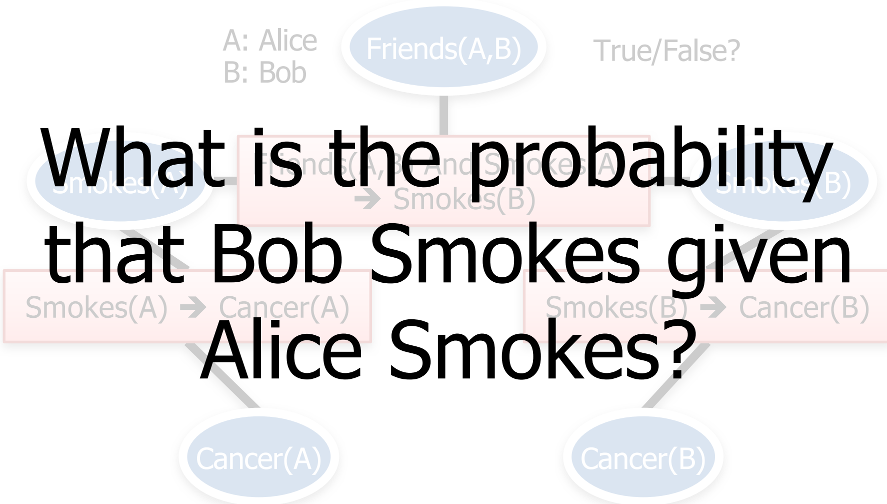




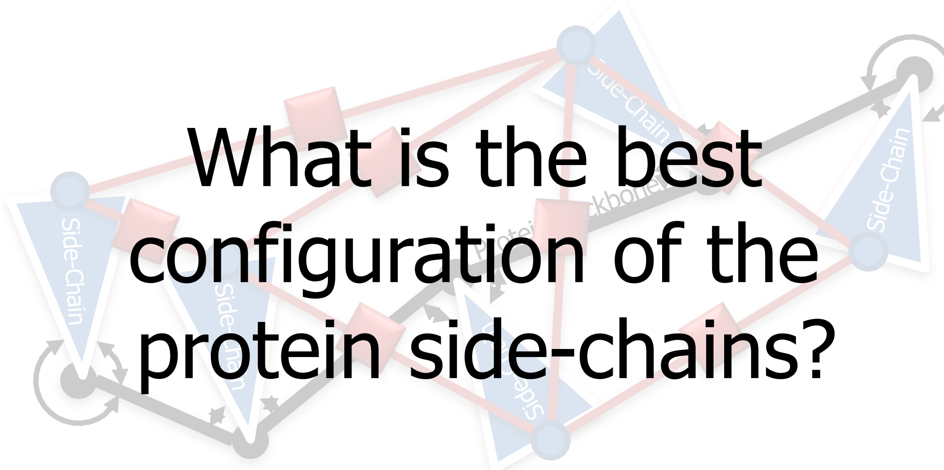
Outline

- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_ε - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

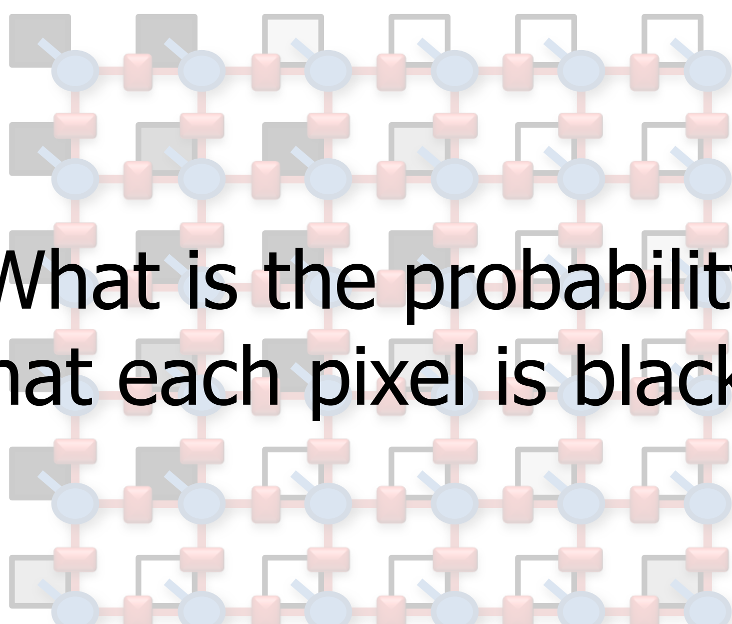
The Inference Problem



What is the probability that Bob Smokes given Alice Smokes?



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

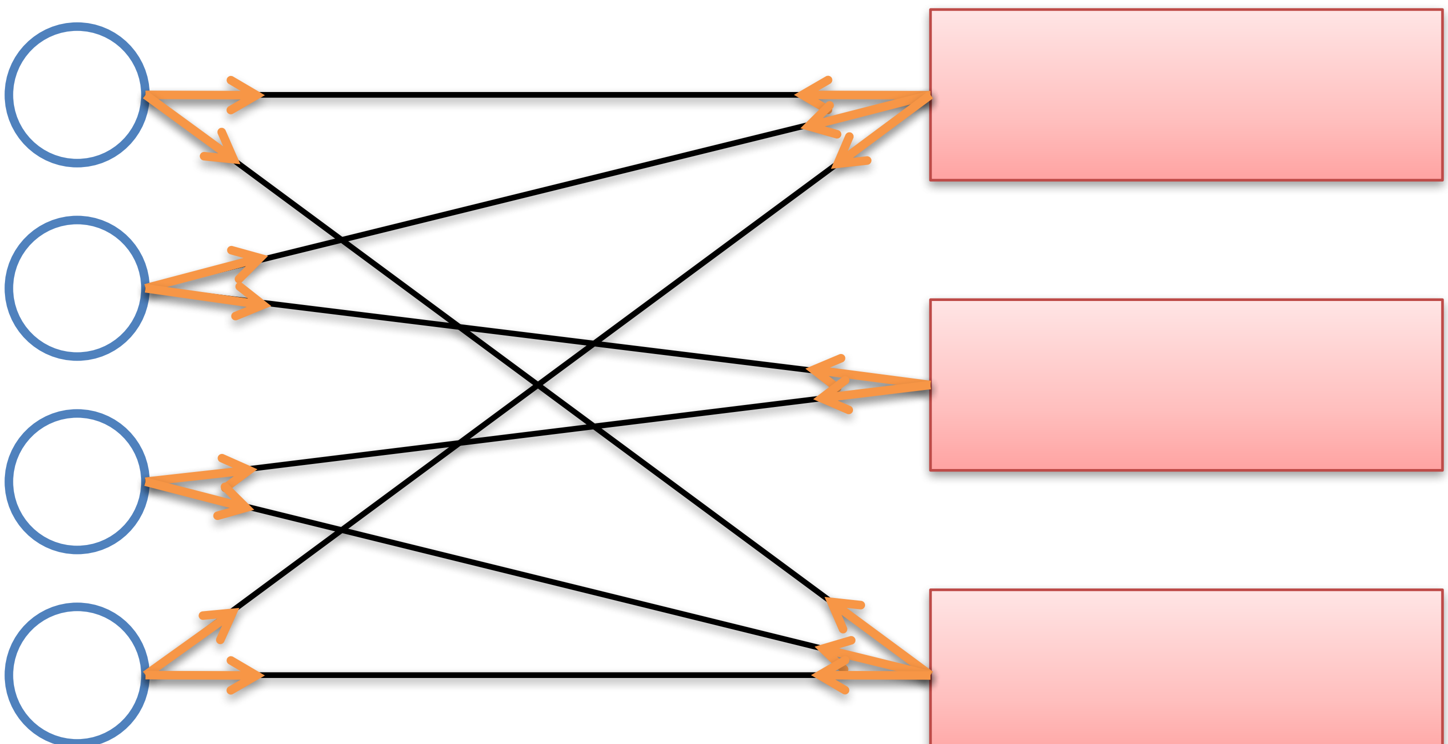


What is the probability that each pixel is black?

- NP-Hard in General
- Approximate Inference:
  - Belief Propagation

Belief Propagation (BP)

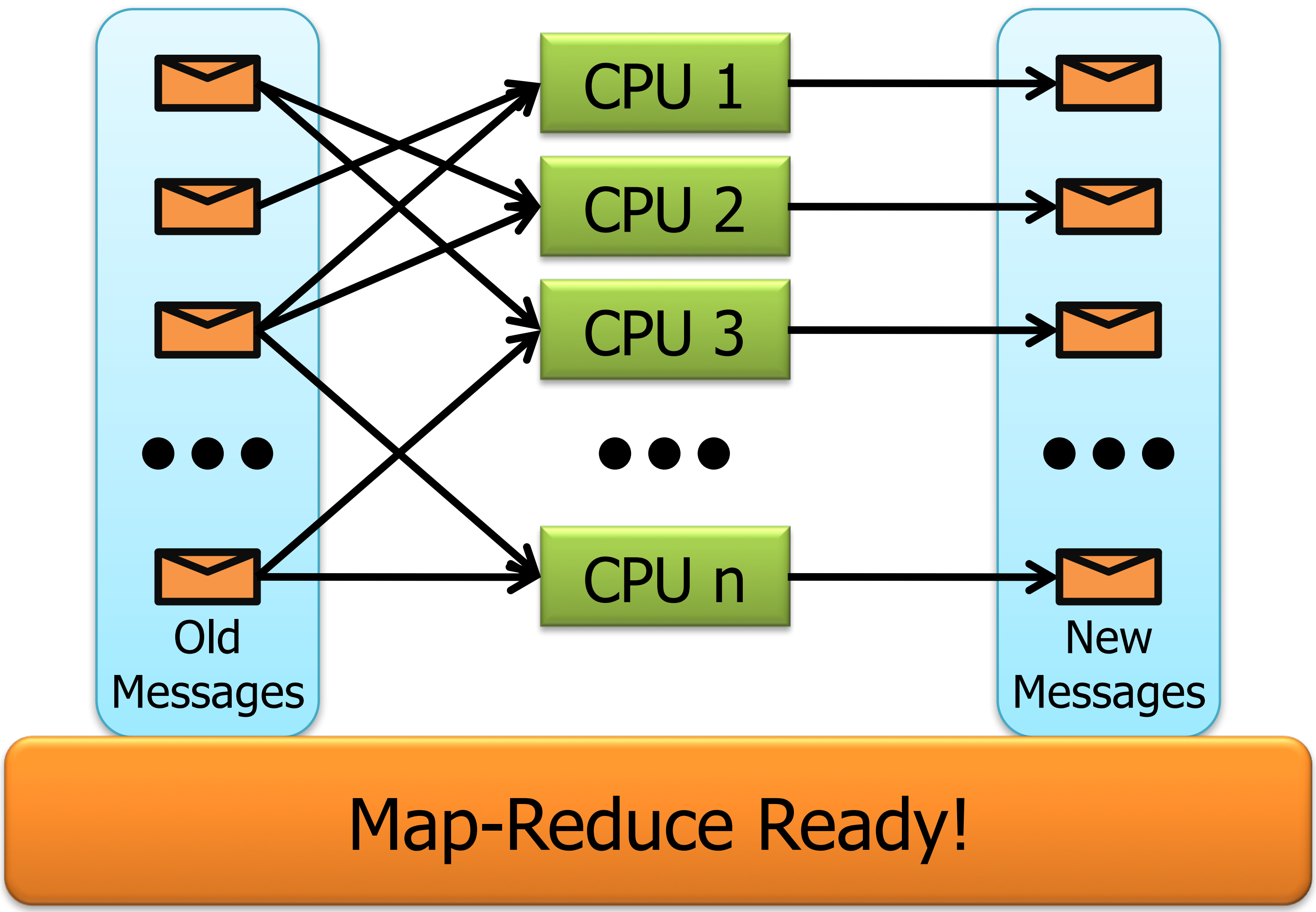
- Iterative message passing algorithm



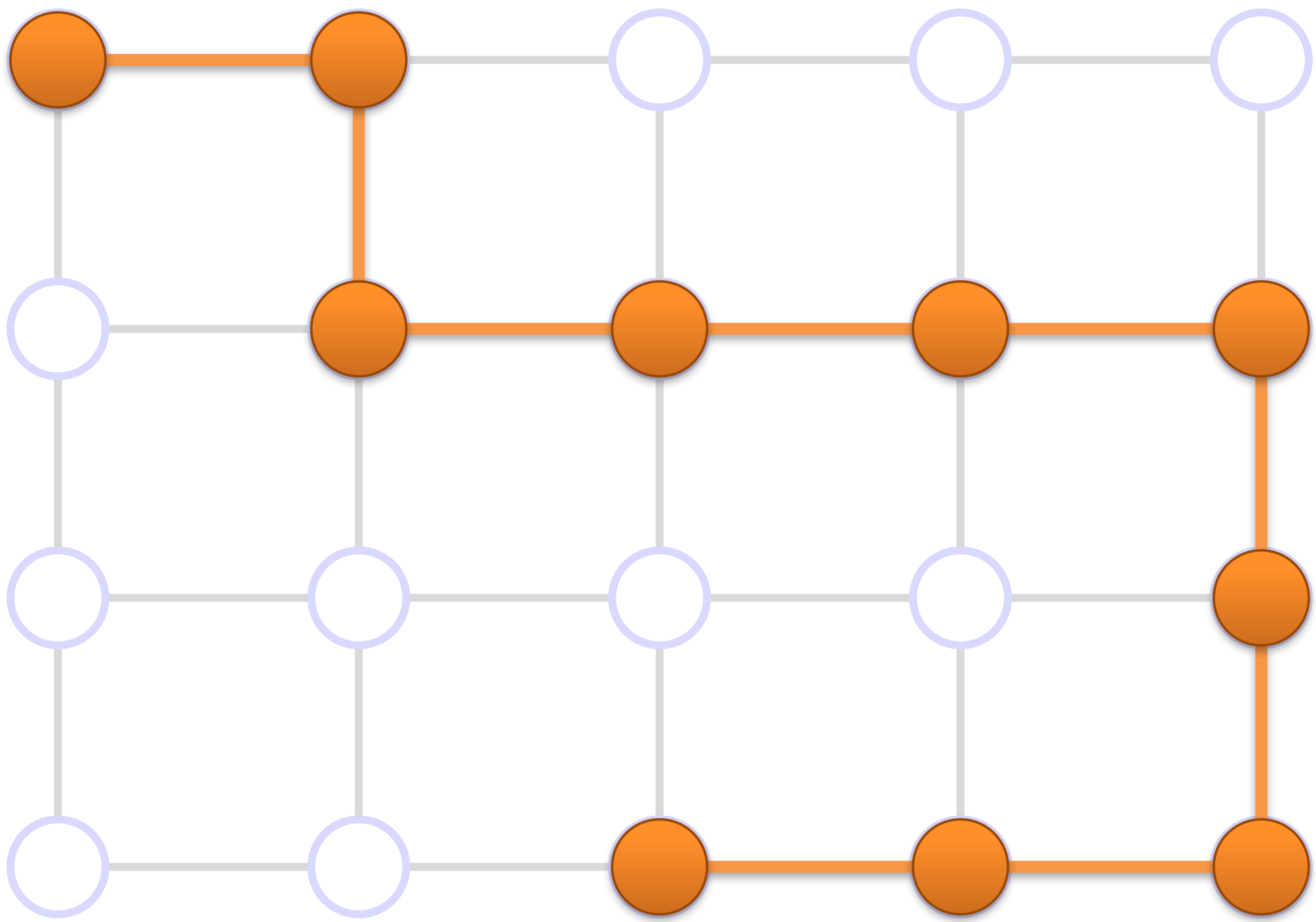
Naturally Parallel Algorithm

Parallel Synchronous BP

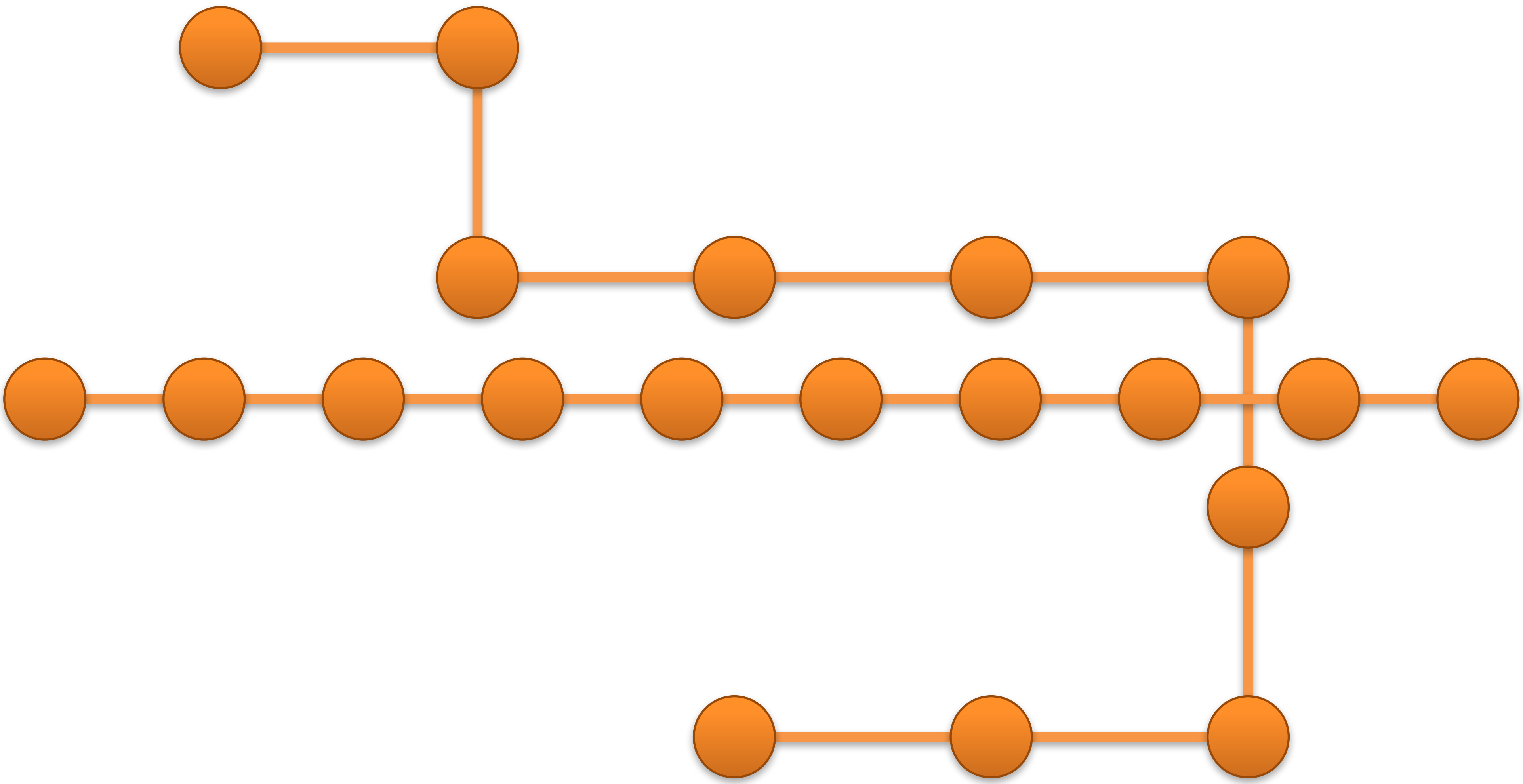
- Given the old messages all new messages can be computed in parallel:



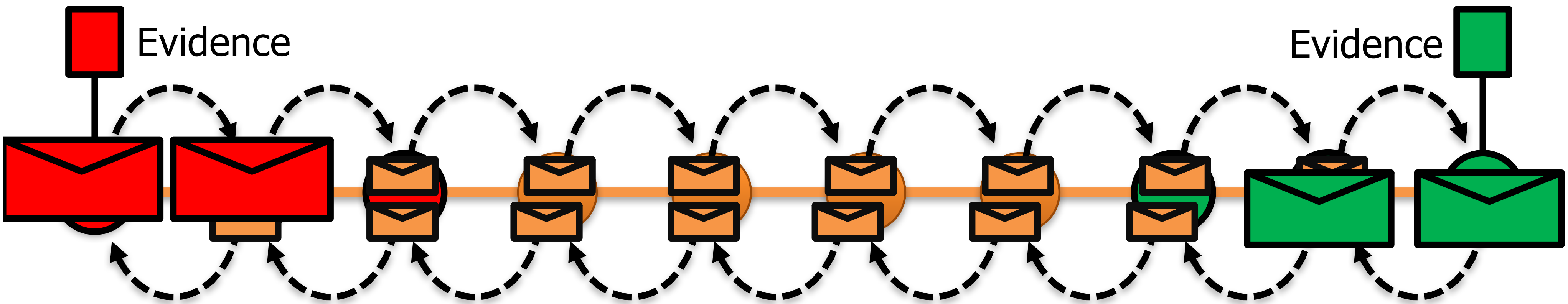
Sequential Computational Structure



Hidden Sequential Structure



Hidden Sequential Structure



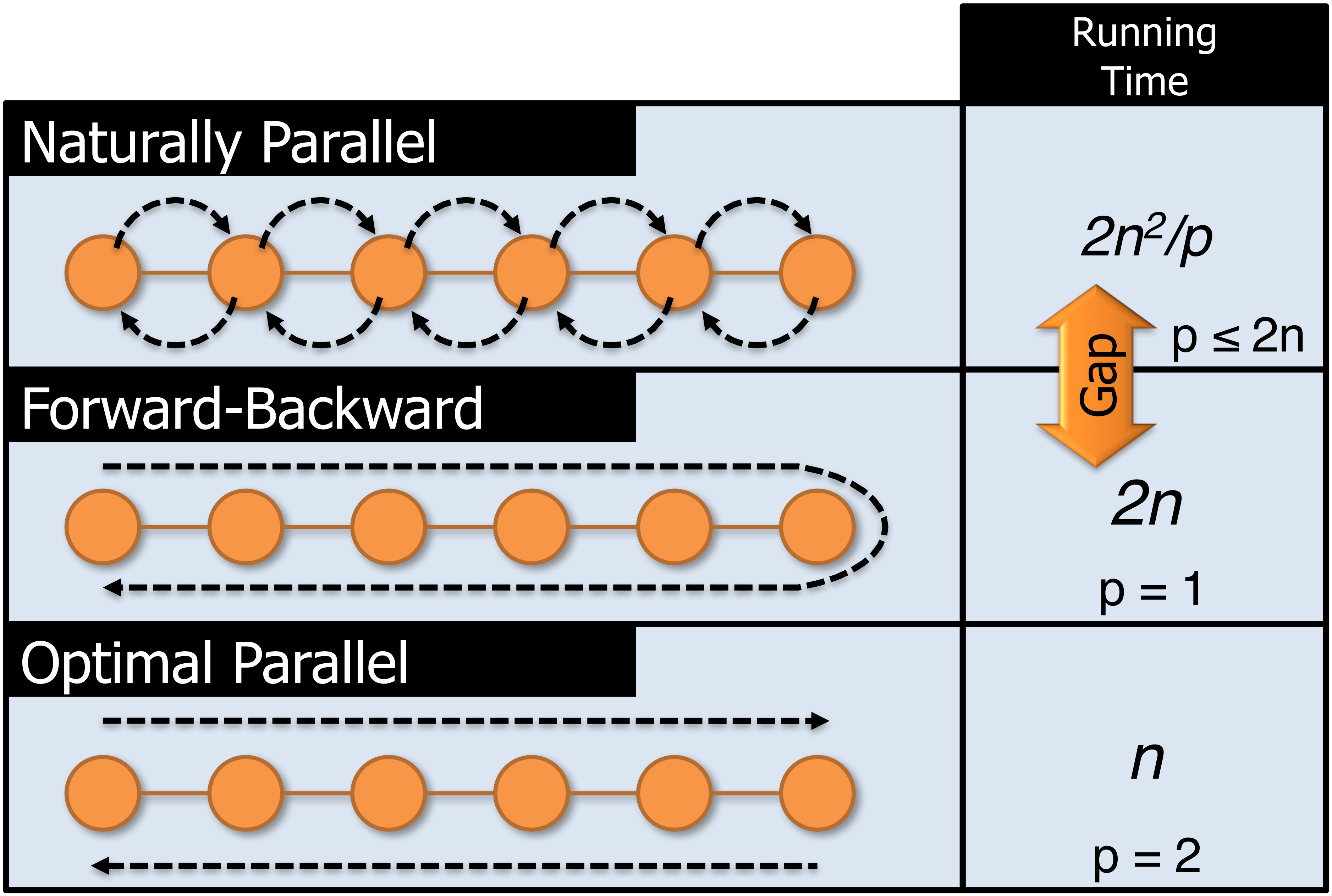
- Running Time:

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

Time for a single parallel iteration

Number of Iterations

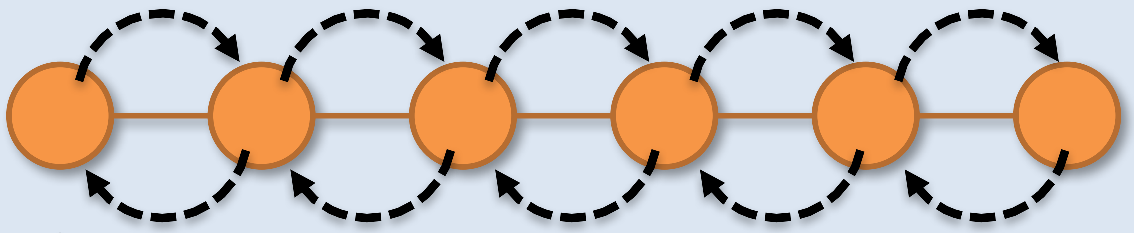
Optimal Sequential Algorithm

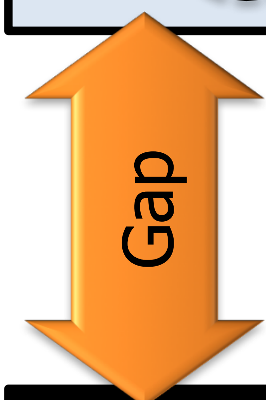


sense learn act

Key Computational Structure

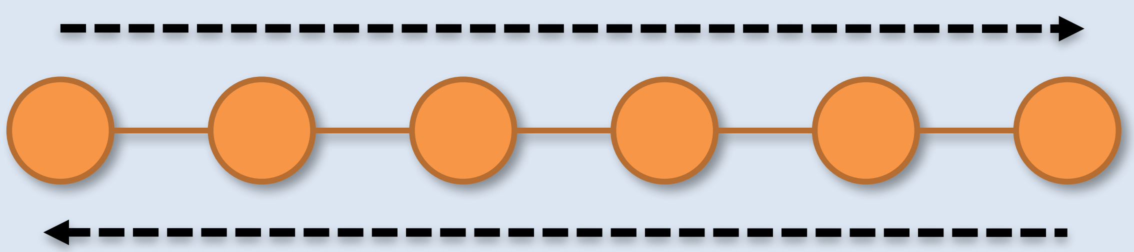
	Running Time
Naturally Parallel	$2n^2/p$ $p \le 2n$
Optimal Parallel	n $p = 2$





Inherent **Sequential** Structure

Requires Efficient Scheduling

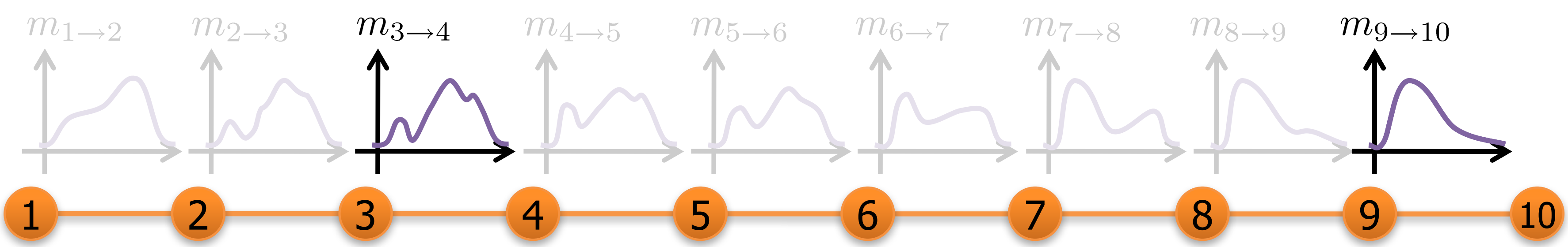


Outline

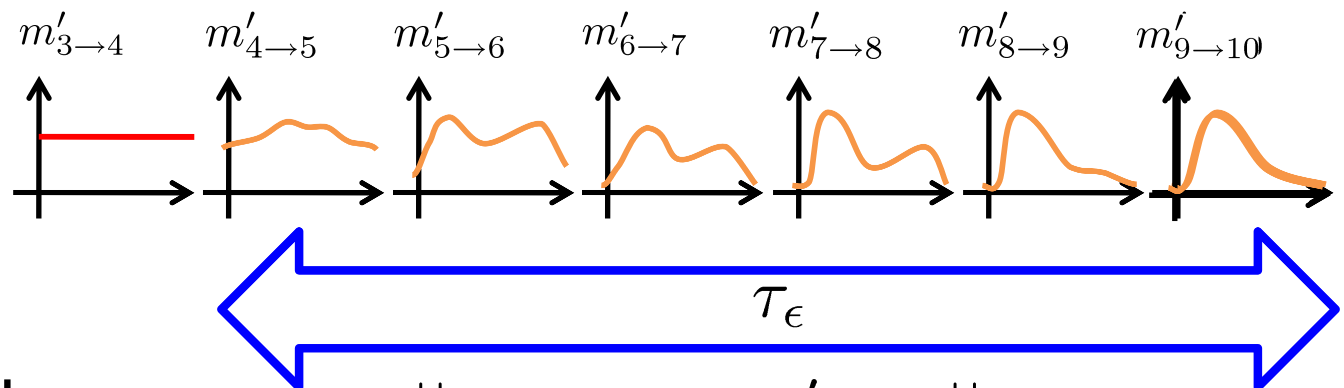
- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_ε - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

Parallelism by Approximation

True Messages



τ_ε -Approximation



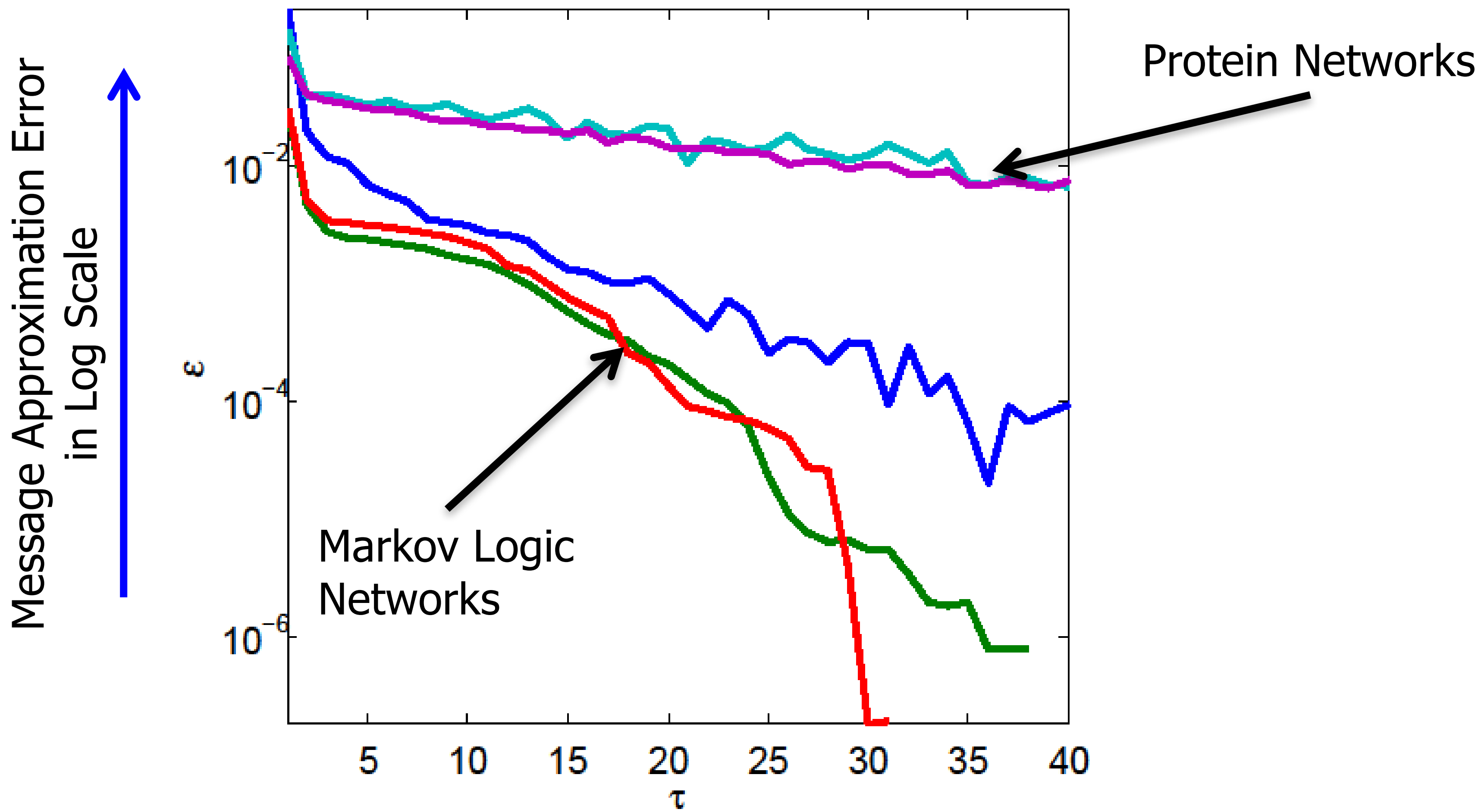
$$||m_{9\to10} - m'_{9\to10}||_1 \leq \epsilon$$

- τ_ε represents the minimal sequential structure

$$\Big\| \qquad - \qquad \Big\|_1$$

Tau-Epsilon Structure

- Often τ_ε decreases quickly:



Running Time Lower Bound

Theorem:
Using p processors it is not possible to obtain a τ_ε approximation in time less than:

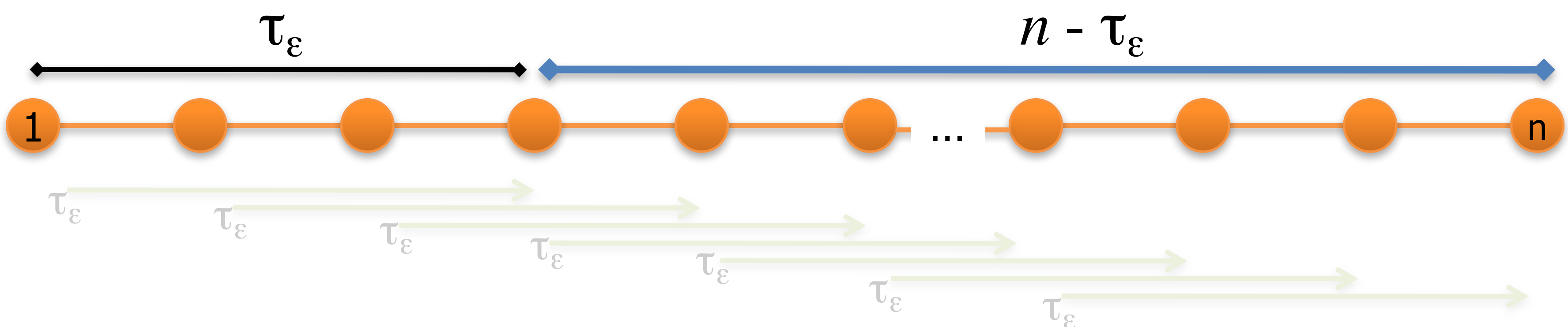
$$\Omega\left(\frac{n}{p} + \tau_\epsilon\right)$$

Parallel Component

Sequential Component

Proof: Running Time Lower Bound

- Consider one direction using *p/2* processors (*p≥2*):



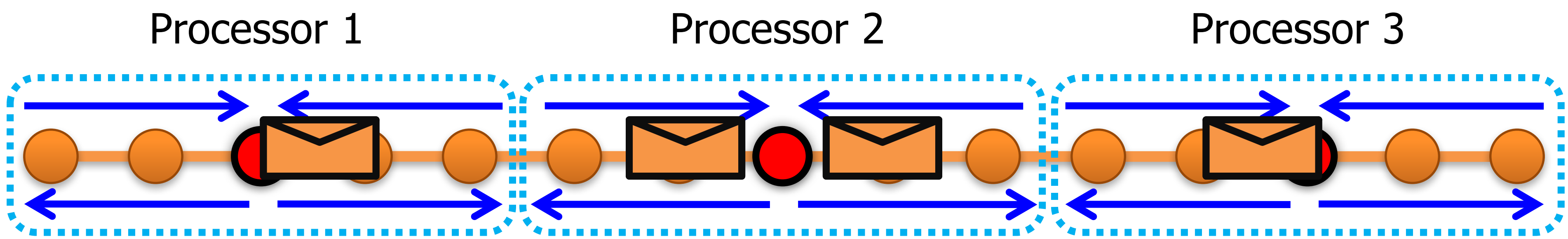
We must make n - τ_ε vertices τ_ε **left-aware**

A single processor can only make k-τ_ε+1 vertices left aware in k-iterations

$$n - \tau_\epsilon \leq \frac{p}{2}(k - \tau_\epsilon + 1)$$

$$k \geq \frac{2n}{p} + \tau_\epsilon\left(1 - \frac{2}{p}\right) - 1$$

Optimal Parallel Scheduling

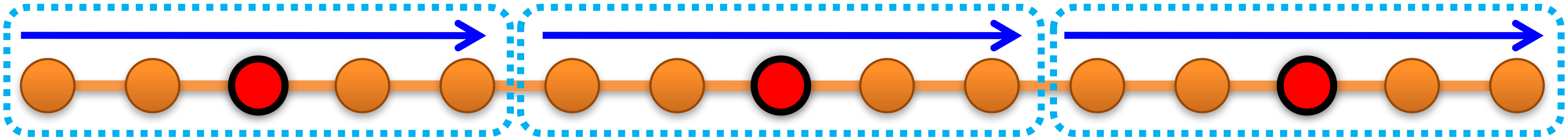


Theorem:
Using p processors this algorithm achieves a τ_ε approximation in time:

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

Proof: Optimal Parallel Scheduling

- All vertices are **left-aware** of the left most vertex on their processor



- After exchanging messages

- After next iteration:

- After k parallel iterations each vertex is $(k\text{-}1)(n/p)$ **left-aware**

Proof: Optimal Parallel Scheduling

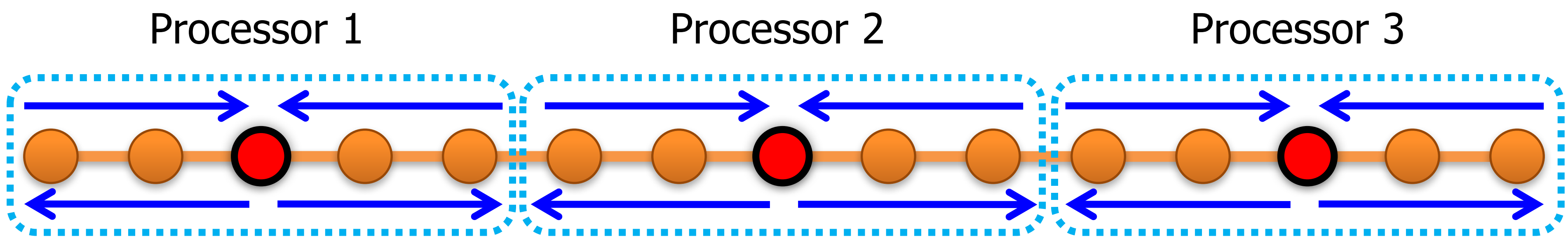
- After k parallel iterations each vertex is $(k\text{-}1)(n/p)$ **left-aware**
- Since all vertices must be made τ_ε left aware:

$$(k-1)\frac{n}{p} = \tau_\epsilon \Rightarrow k = \frac{p}{n}\tau_\epsilon + 1$$

- Each iteration takes $O(n/p)$ time:

$$\frac{2n}{p}\left(\frac{p}{n}\tau_\epsilon + 1\right) \in O\left(\frac{n}{p} + \tau_\epsilon\right)$$

Comparing with SynchronousBP



Synchronous Schedule

Optimal Schedule

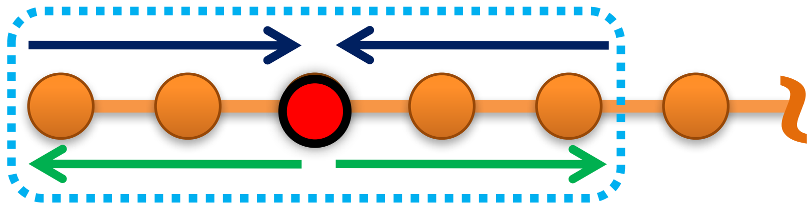
$$\mathrm{O}\left(\frac{n\tau_\epsilon}{p}\right) \quad \text{Gap} \quad \mathrm{O}\left(\frac{n}{p}+\tau_\epsilon\right)$$

Outline

- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_{ε} - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

The Splash Operation

- Generalize the optimal chain algorithm:

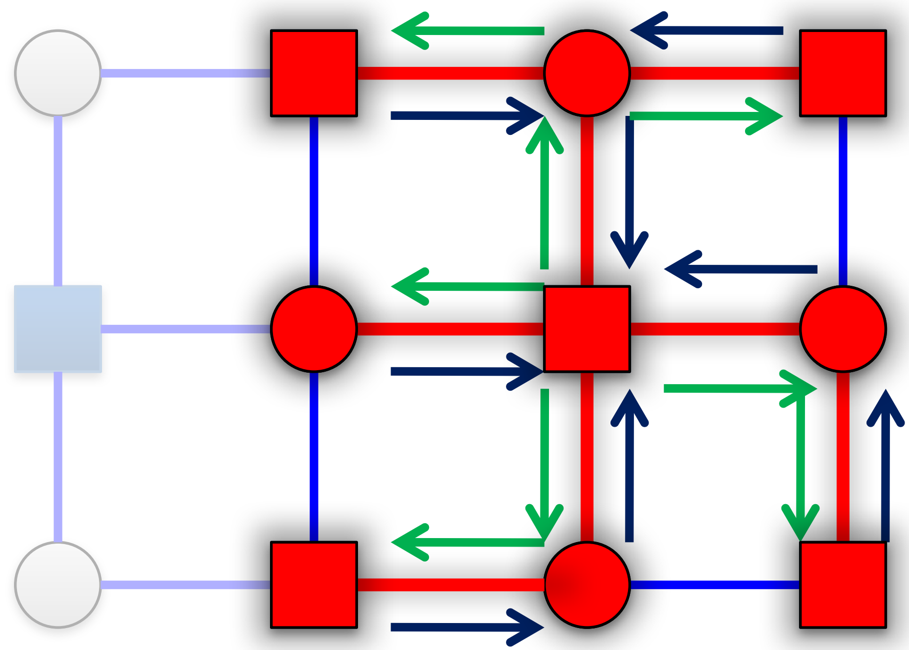


to arbitrary cyclic graphs:

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



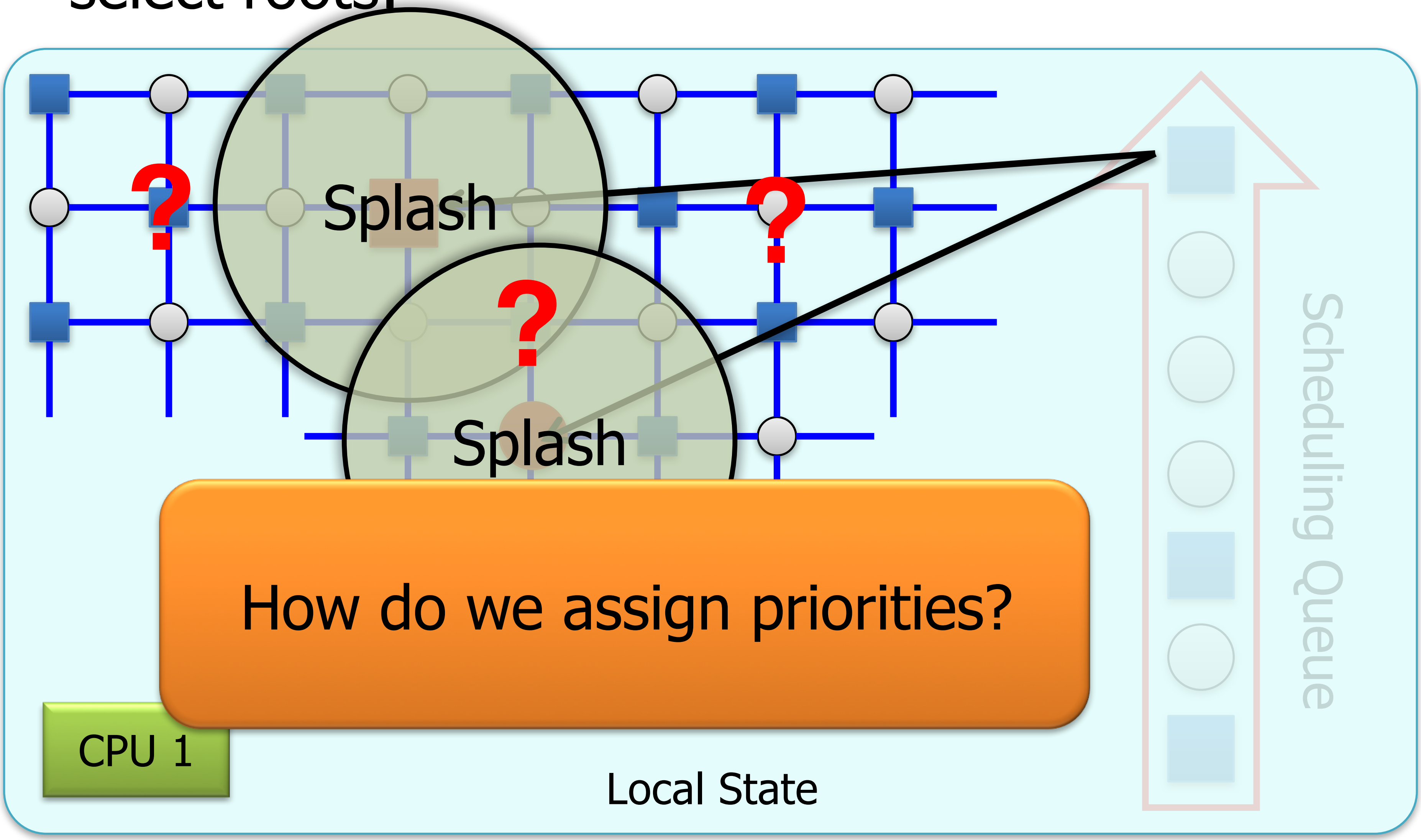
Running Parallel Splashes

Key Challenges:
1) How do we schedules Splashes?
2) How do we partition the Graph?

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

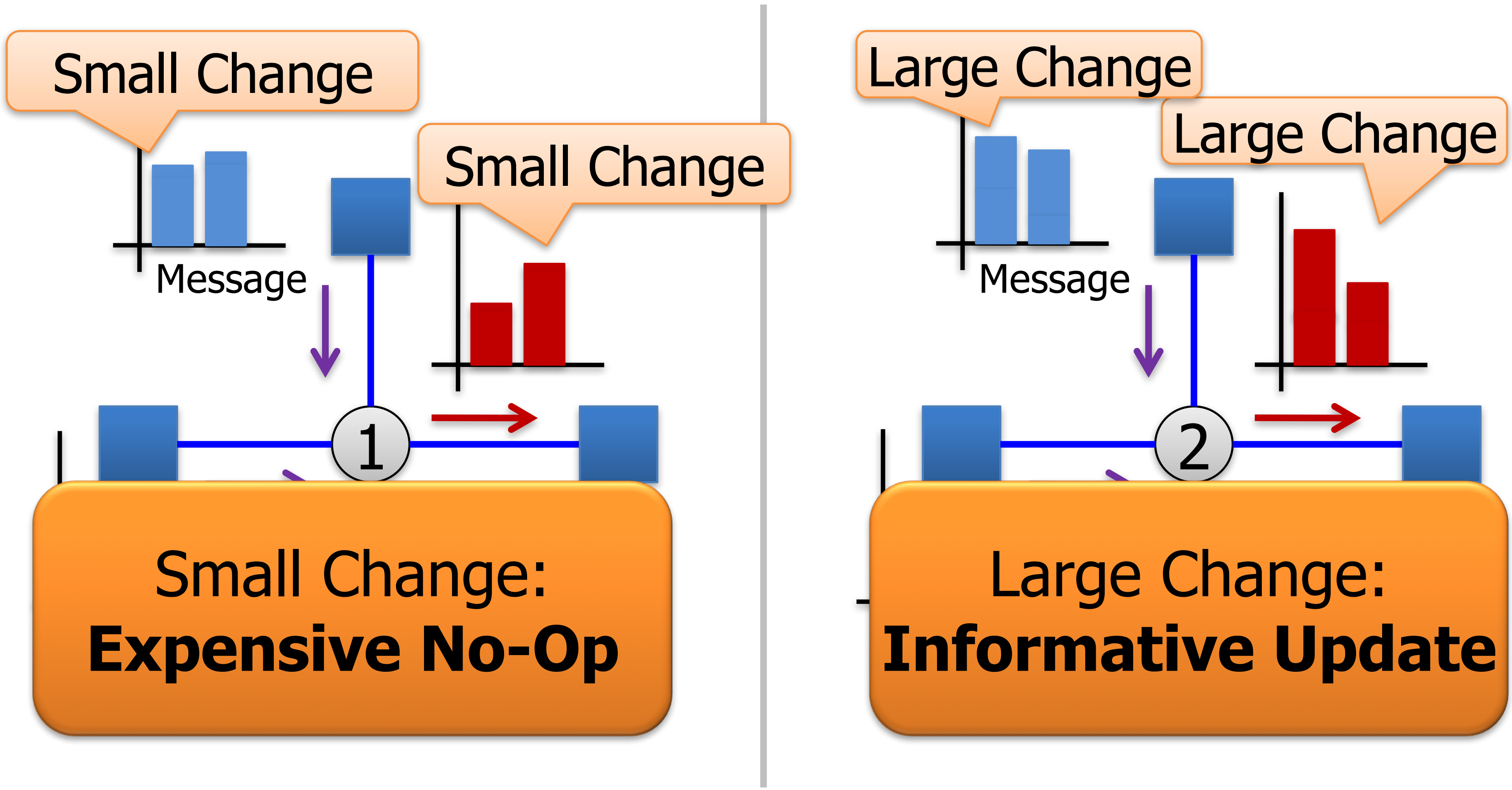
Where do we Splash?

- Assign priorities and use a scheduling queue to select roots:



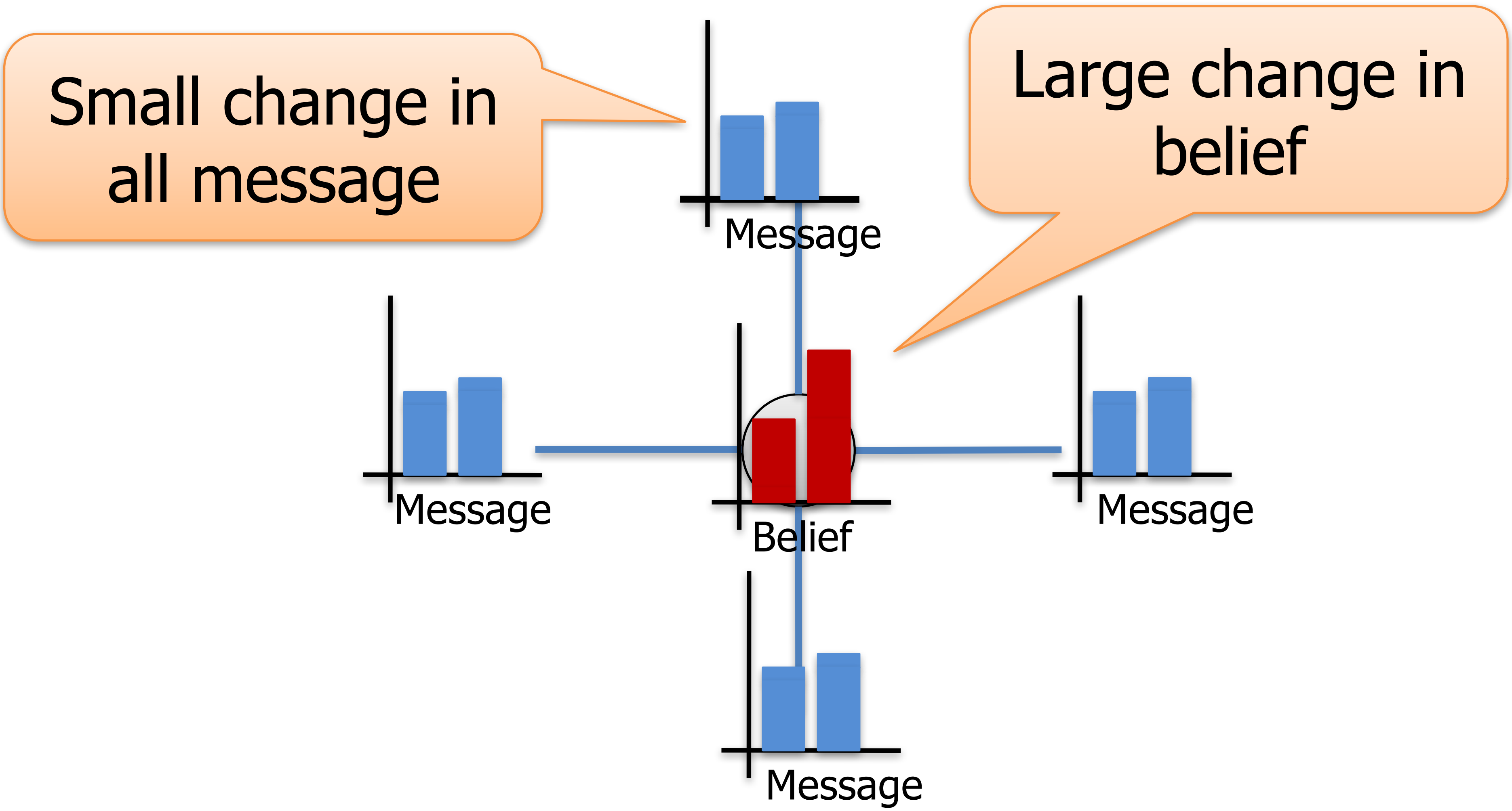
Message Scheduling

- Residual Belief Propagation [Elidan et al., UAI 06]:
  - Assign priorities based on change in inbound messages



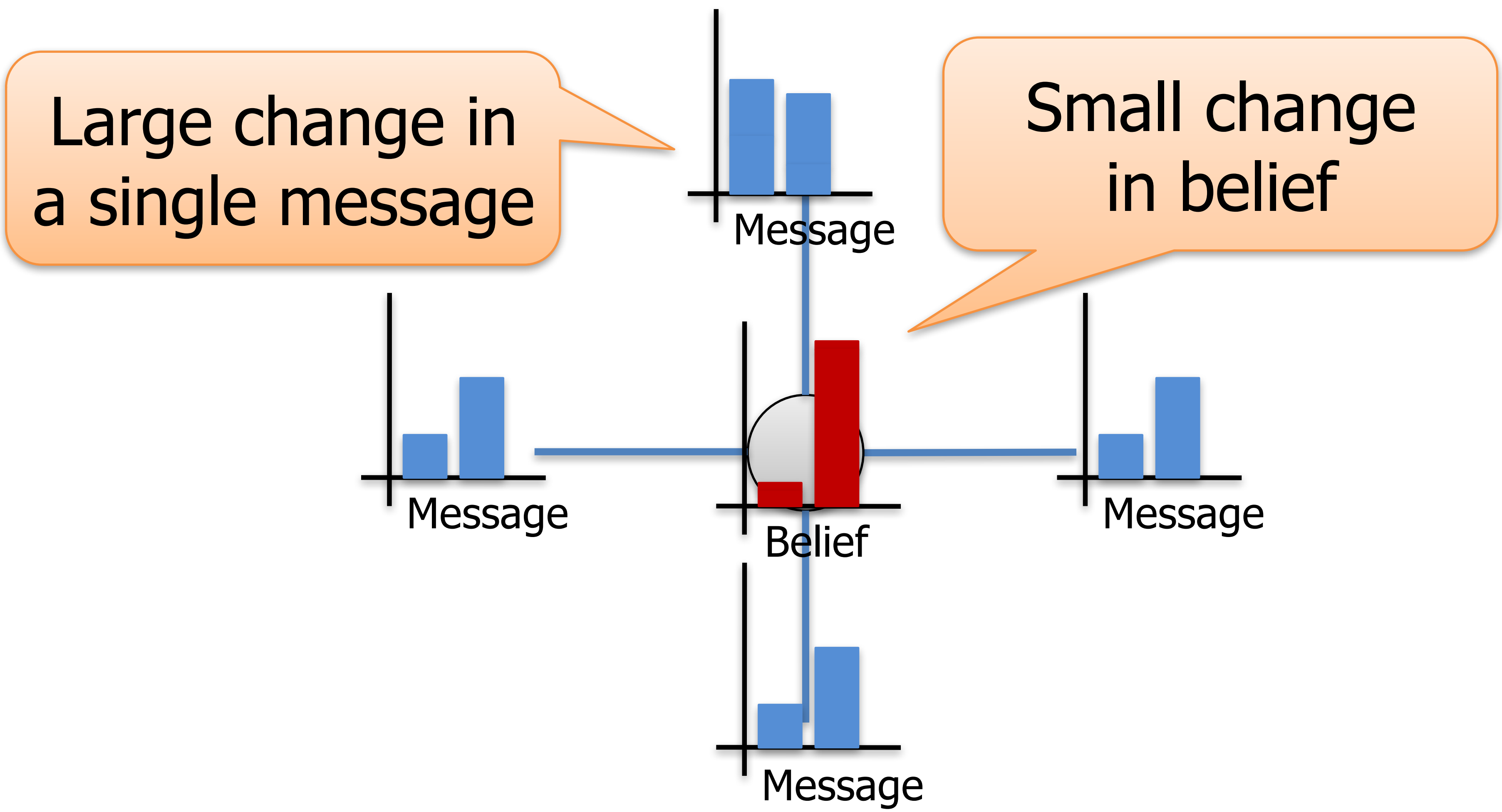
Problem with Message Scheduling

- Small changes in messages do not imply small changes in belief:



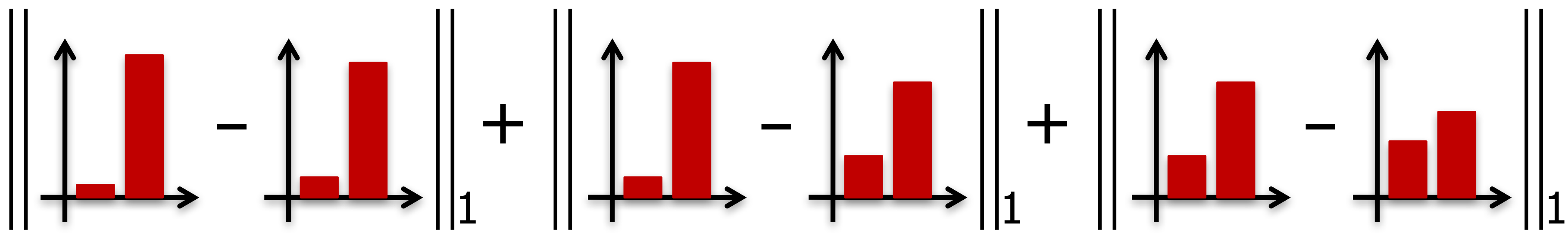
Problem with Message Scheduling

- Large changes in a single message do not imply large changes in belief:

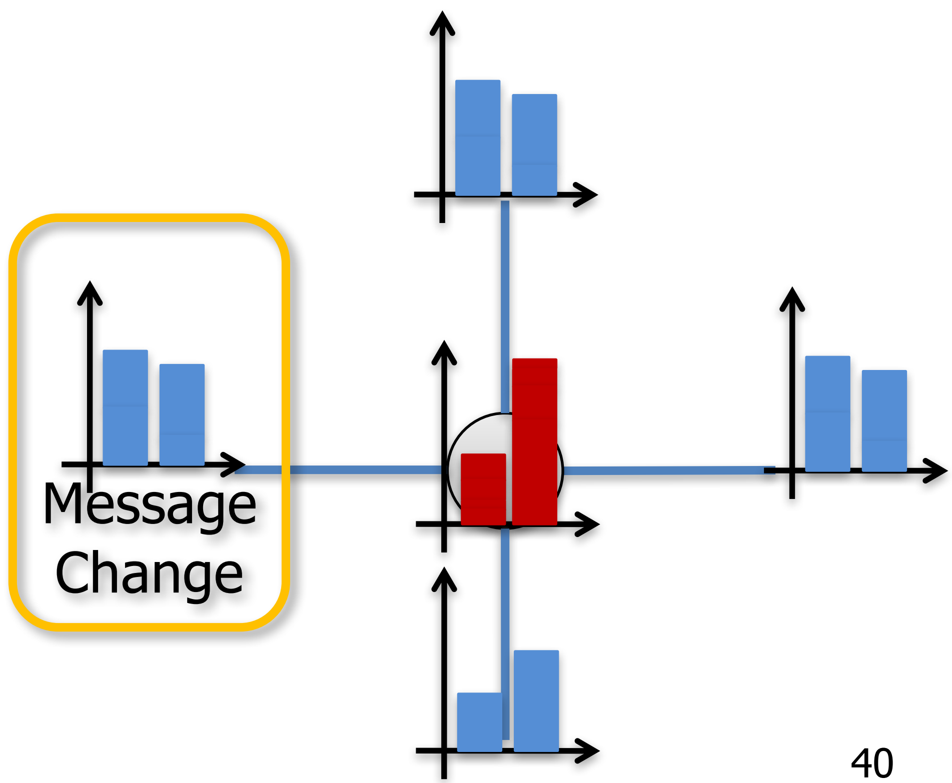


Belief Residual Scheduling

- Assign priorities based on the cumulative change in belief:

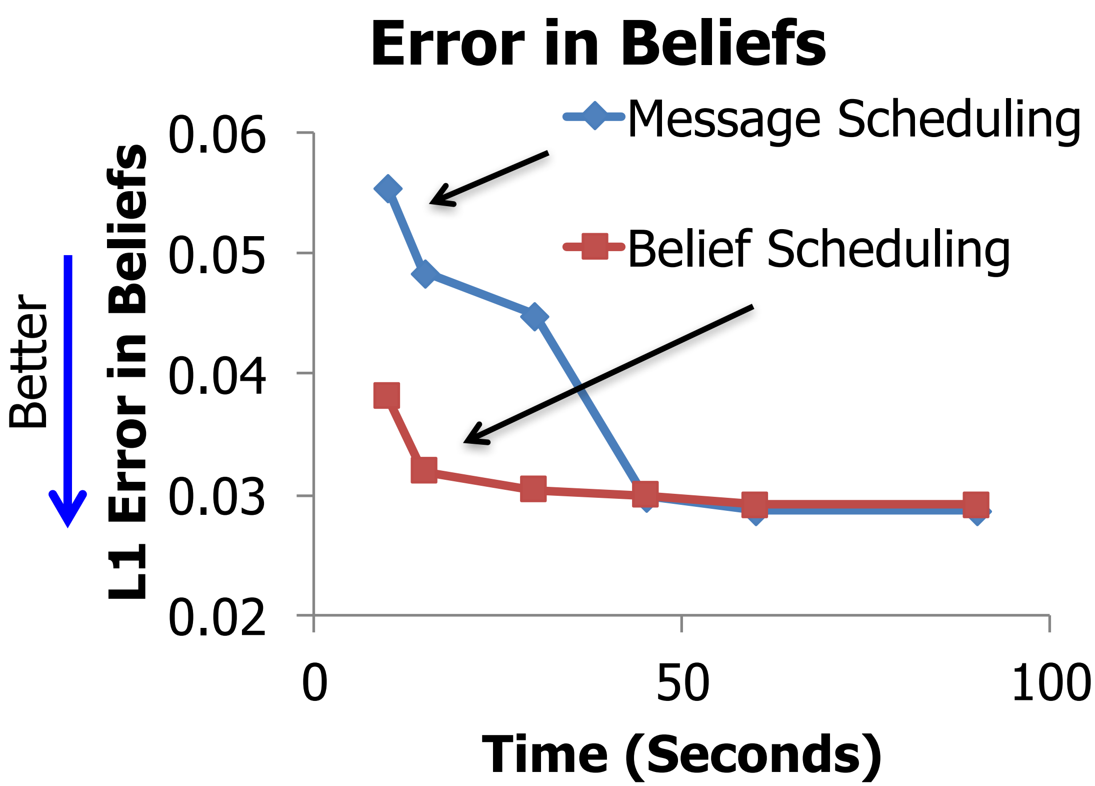
$r_v =$ 

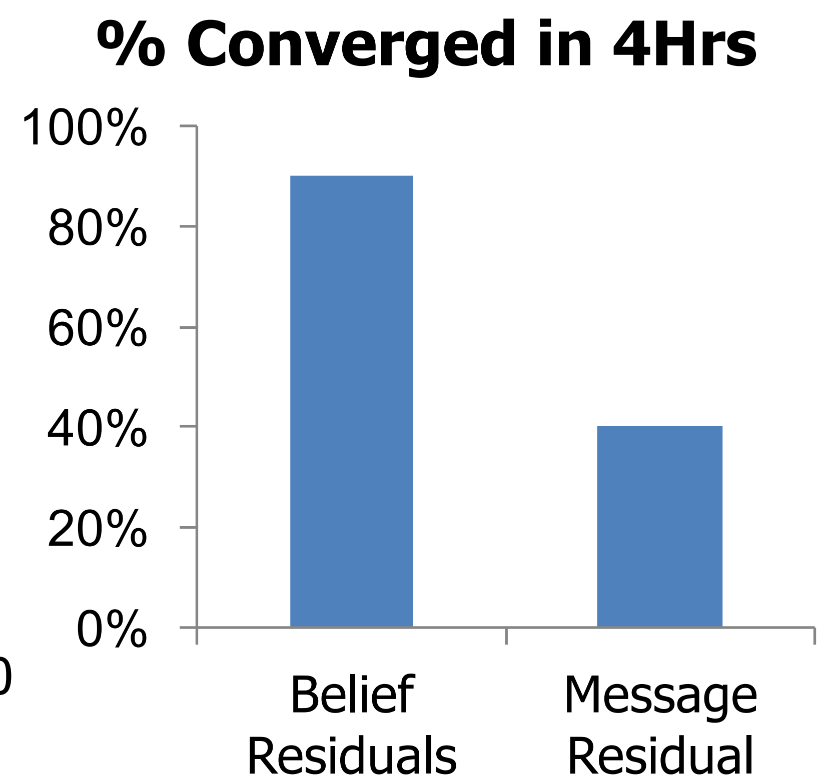
A vertex whose belief has changed substantially since last being updated will likely produce informative new messages.



Message vs. Belief Scheduling

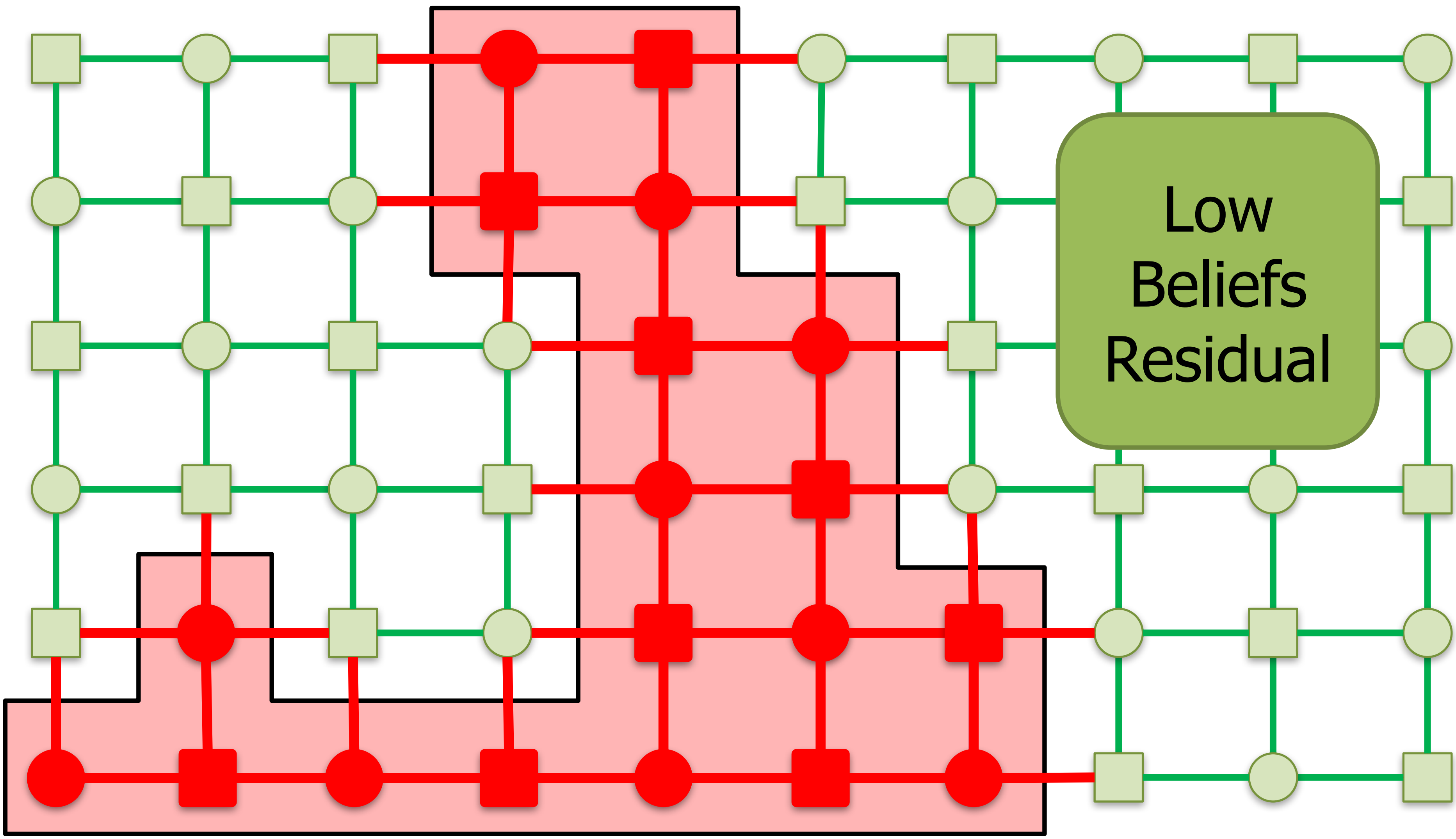
Belief Scheduling improves **accuracy**





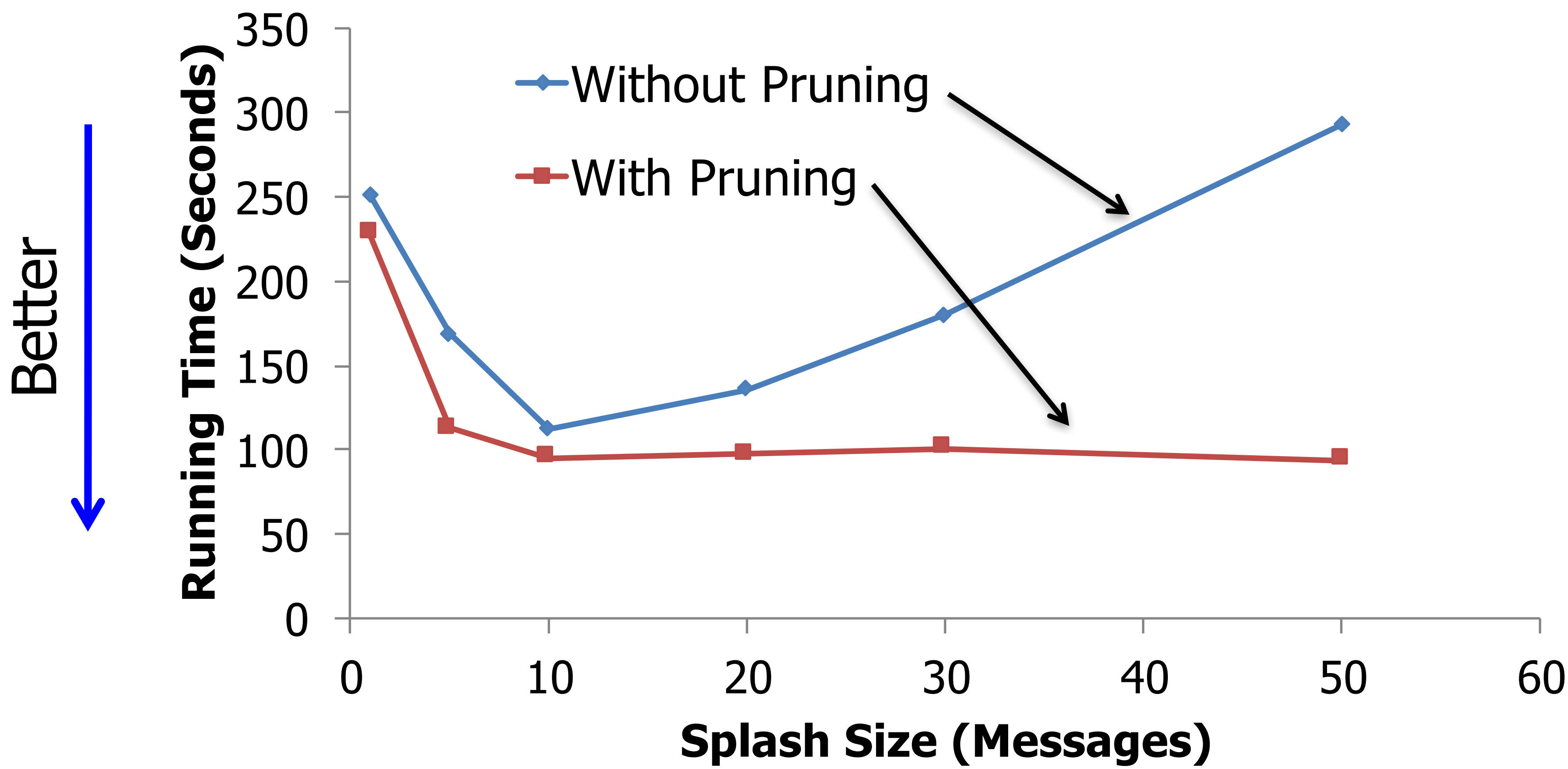
Splash Pruning

- Belief residuals can be used to **dynamically** reshape and resize Splashes:

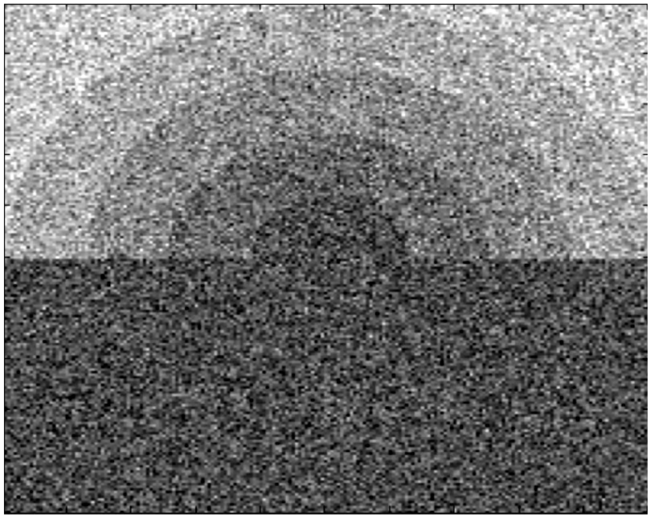


Splash Size

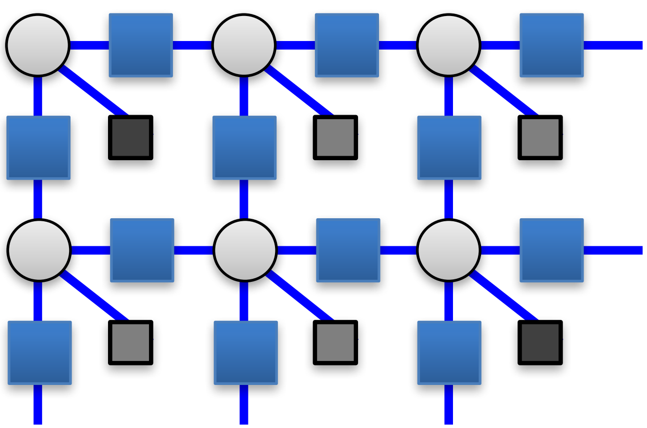
- Using **Splash Pruning** our algorithm is able to dynamically select the **optimal** splash size



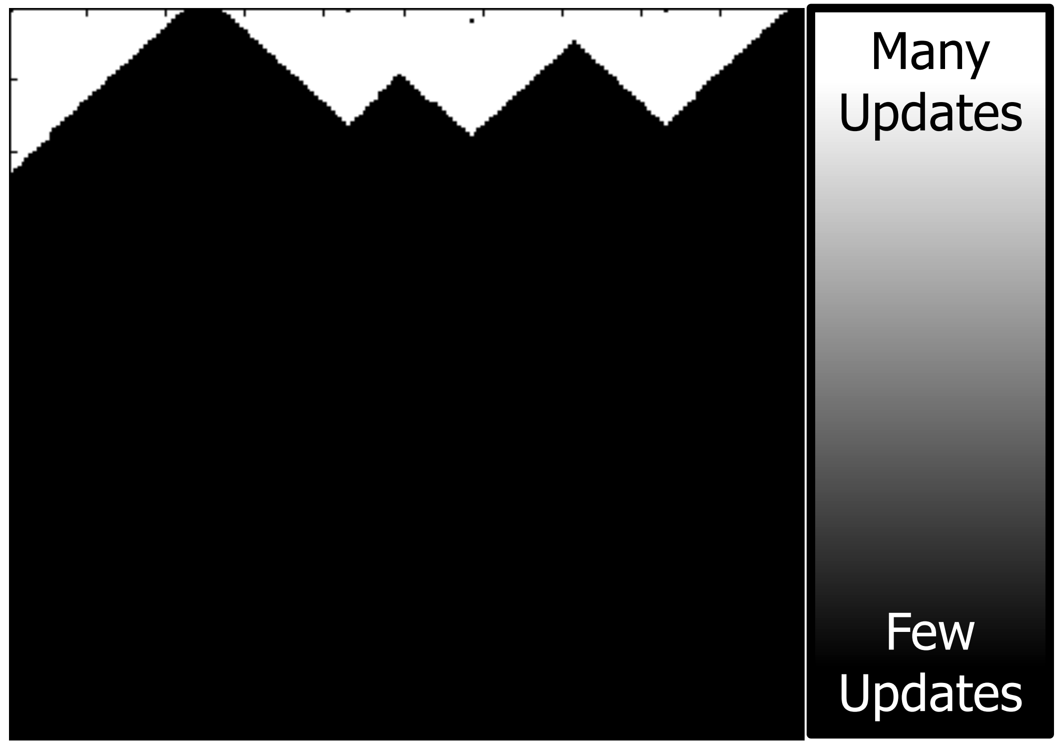
Example



Synthetic Noisy Image



Factor Graph



Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

Parallel Splash Algorithm

Fast Reliable Network

Theorem:

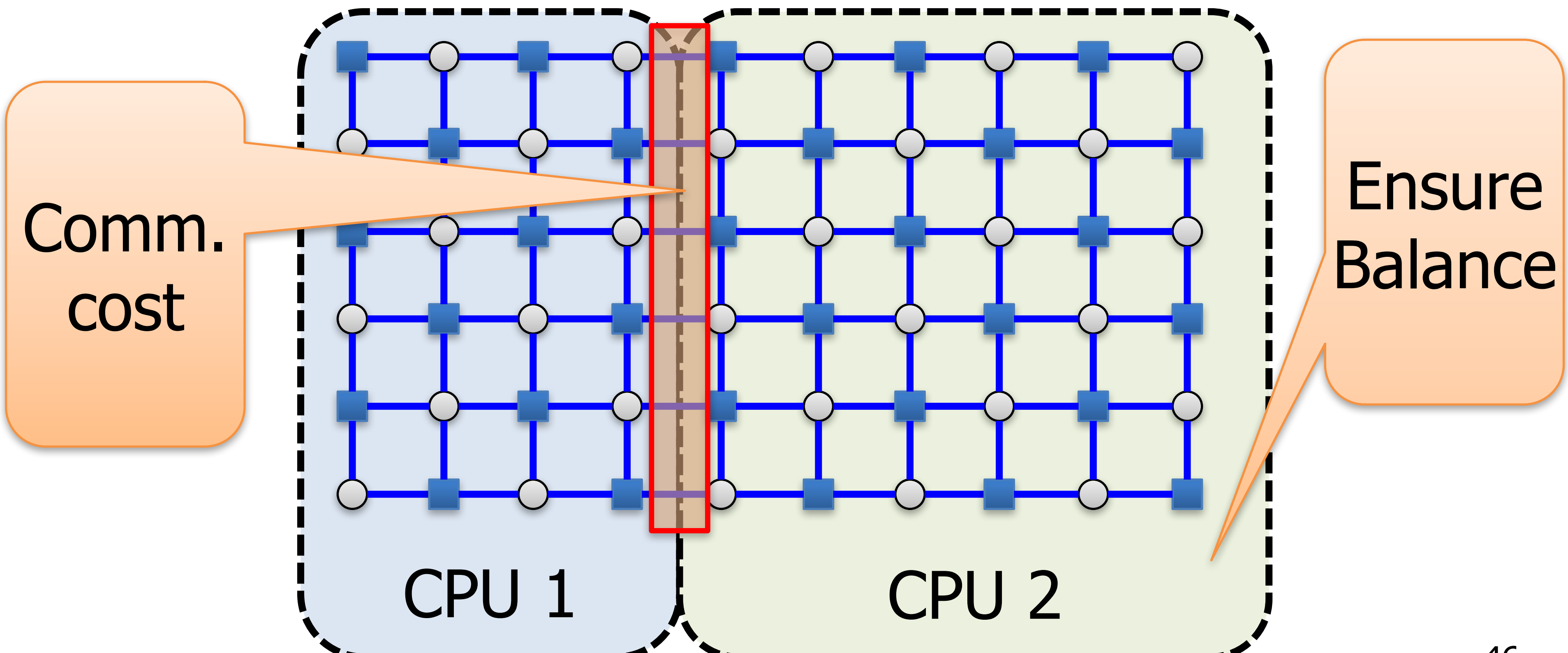
Given a uniform partitioning of the chain graphical model, Parallel Splash will run in time:

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

retaining optimality.

Partitioning Objective

- The partitioning of the factor graph determines:
  - Storage, Computation, and Communication
- Goal:
  - Balance **Computation** and Minimize **Communication**



The Partitioning Problem

- Objective:

$$\text{minimize:} \quad \sum_{(i,j)\in \text{Cut Edges}} c_{ij}$$

Minimize Communication

$$\text{subj. to:} \quad \sum_{i \in \text{Largest Block}} w_i \leq \frac{\gamma}{p} \sum_{v \in V} w_v$$

Ensure Balance

- Depends on:

Update counts are not known!

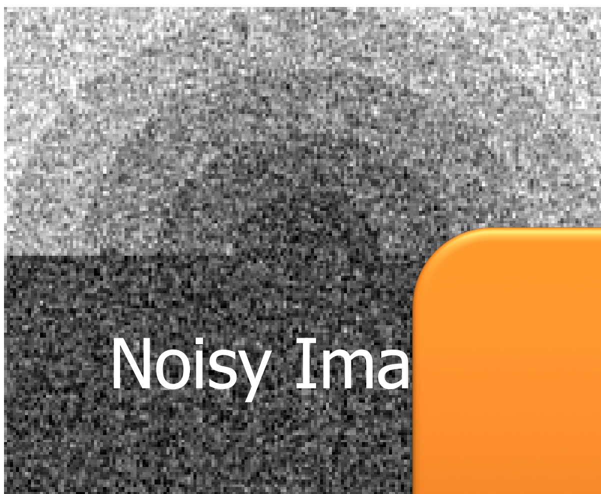
Work: $w_i = \boxed{\text{Updates}_i} \times \text{Size}(i) \times \text{Degree}(i)$

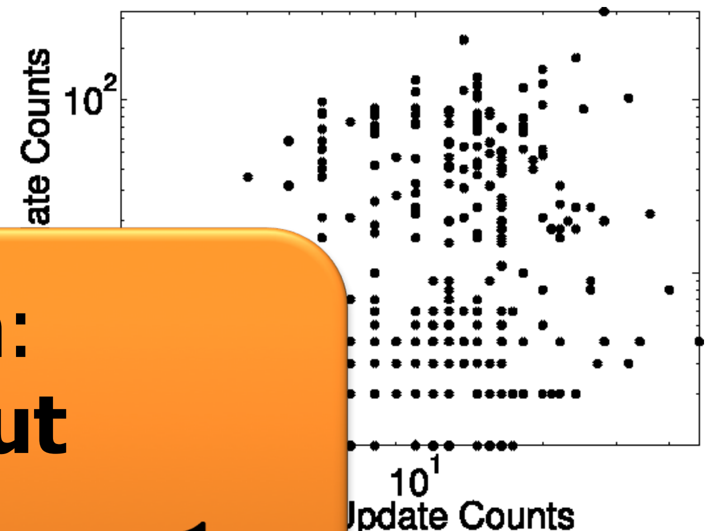
Comm: $c_{ij} = \boxed{(\text{Updates}_i + \text{Updates}_j)} \times \text{MessageSize}(i,j)$

- NP-Hard → METIS fast partitioning heuristic

Unknown Update Counts

- Determined by belief scheduling
- Depends on: graph structure, factors, …
- Little correlation between past & future update counts



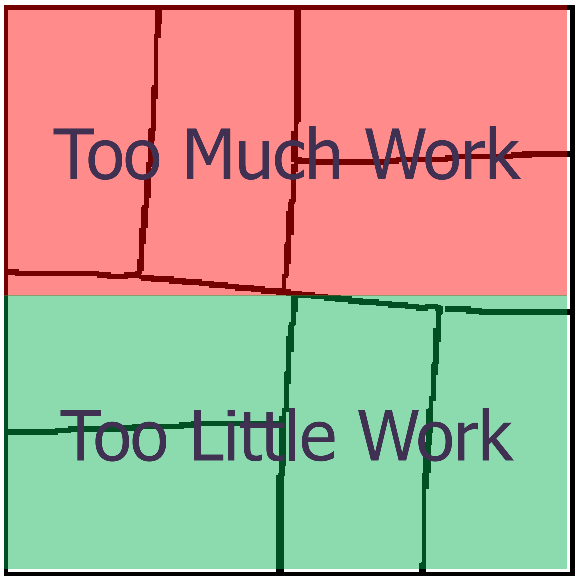


Simple Solution:
Uninformed Cut

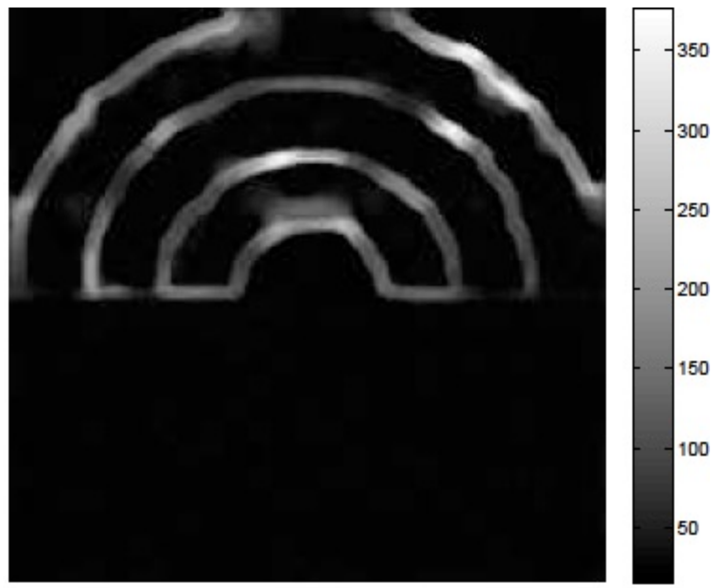
$$\text{Updates}_i = 1$$

Uniformed Cuts

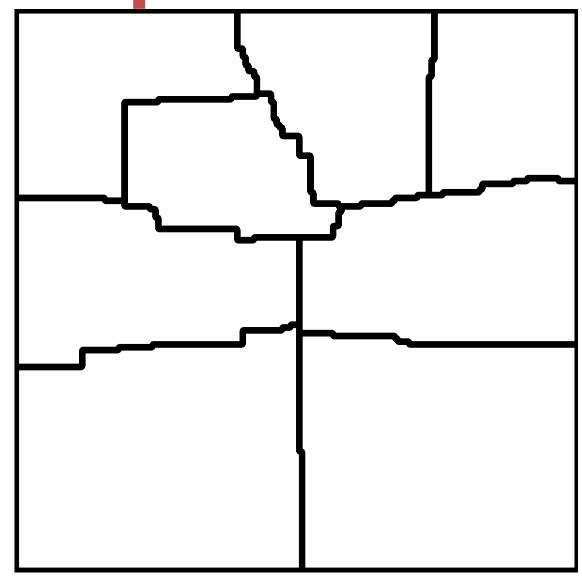
Uninformed Cut



Update Counts

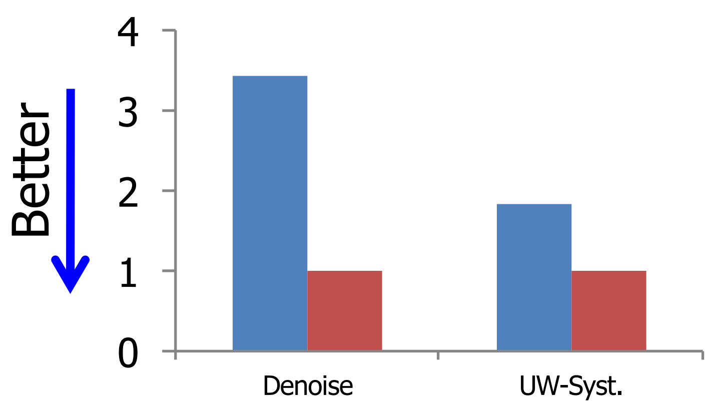


Optimal Cut

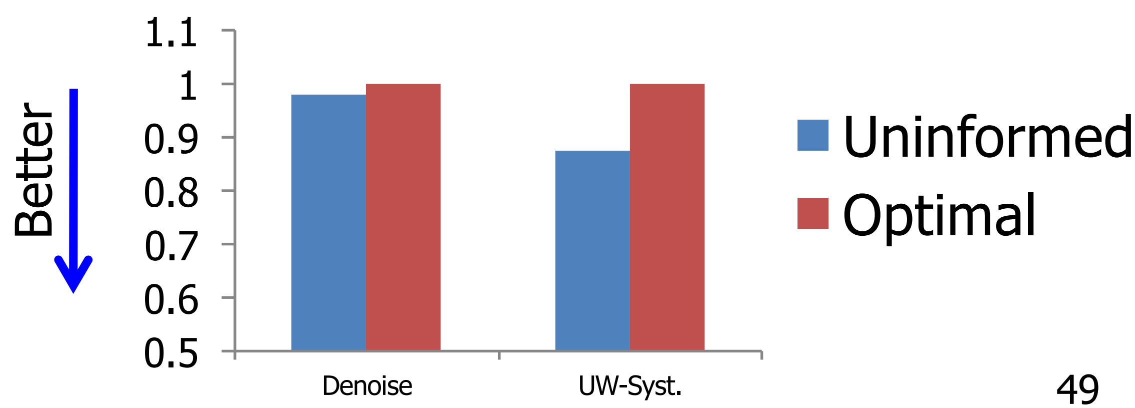


- Greater imbalance & lower communication cost

Work Imbalance

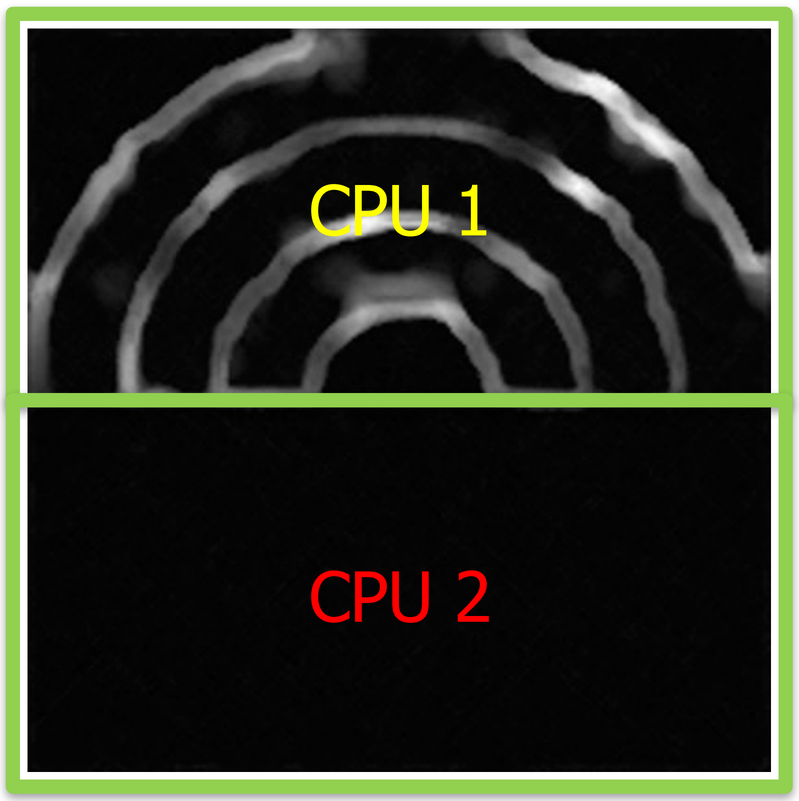


Communication Cost

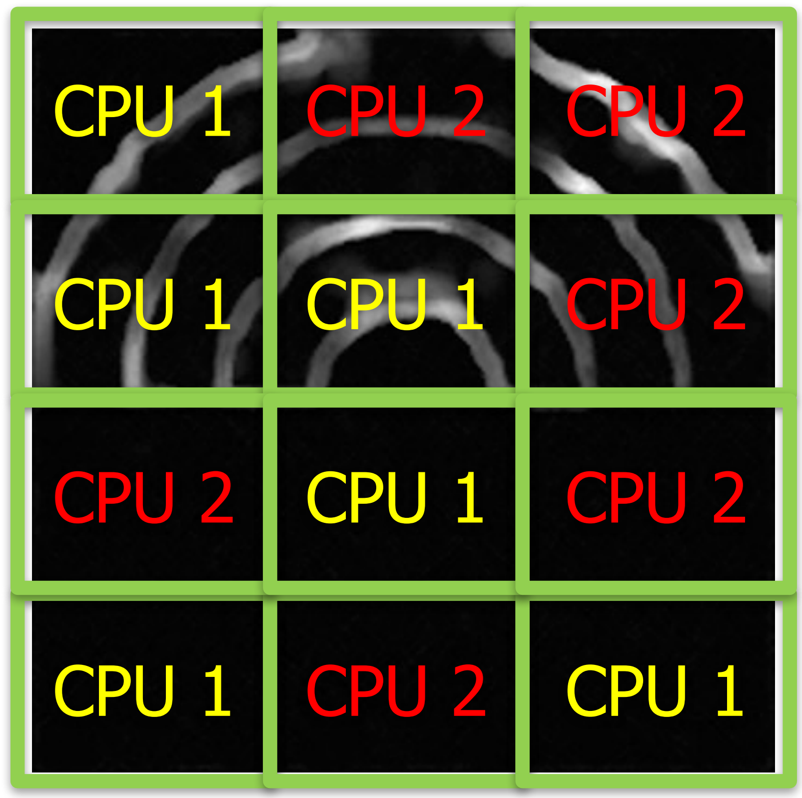


Over-Partitioning

- Over-cut graph into *k*p* partitions and randomly assign CPUs
  - Increase balance
  - Increase communication cost (More Boundary)



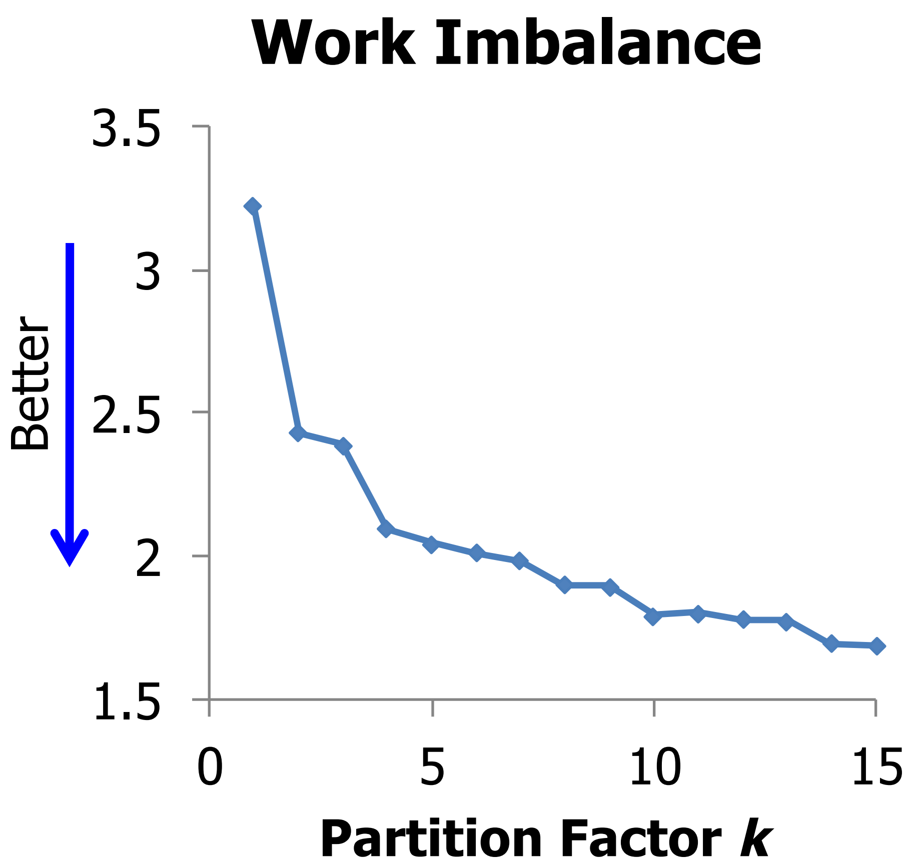
Without Over-Partitioning

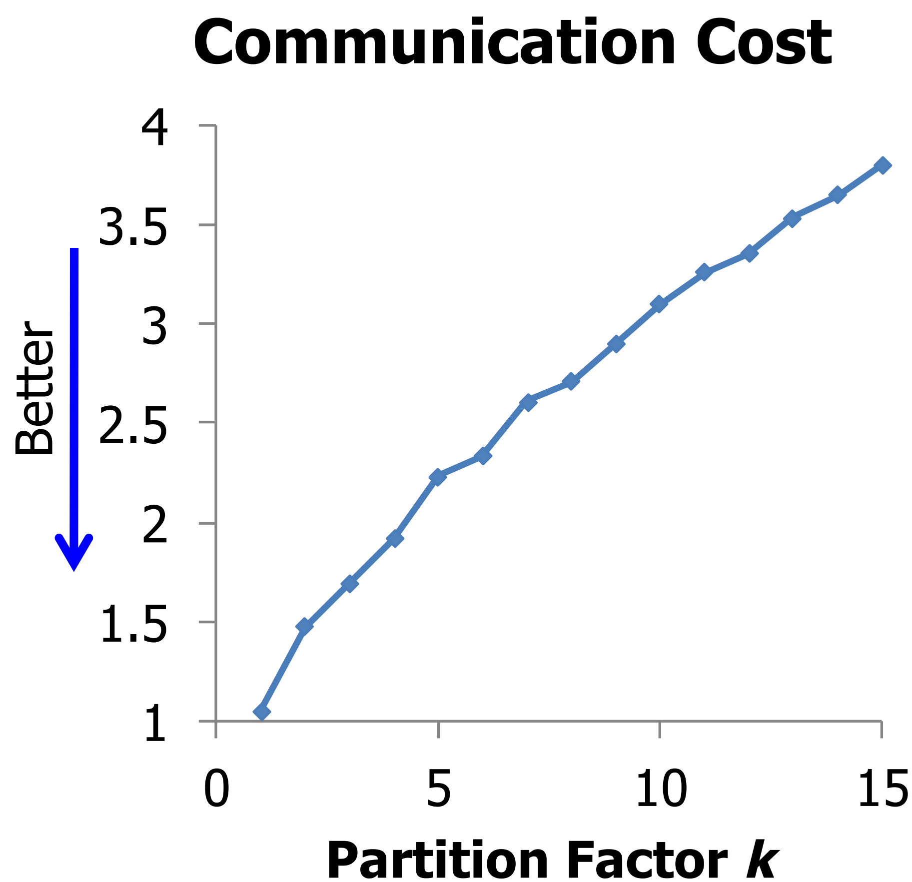


k=6

Over-Partitioning Results

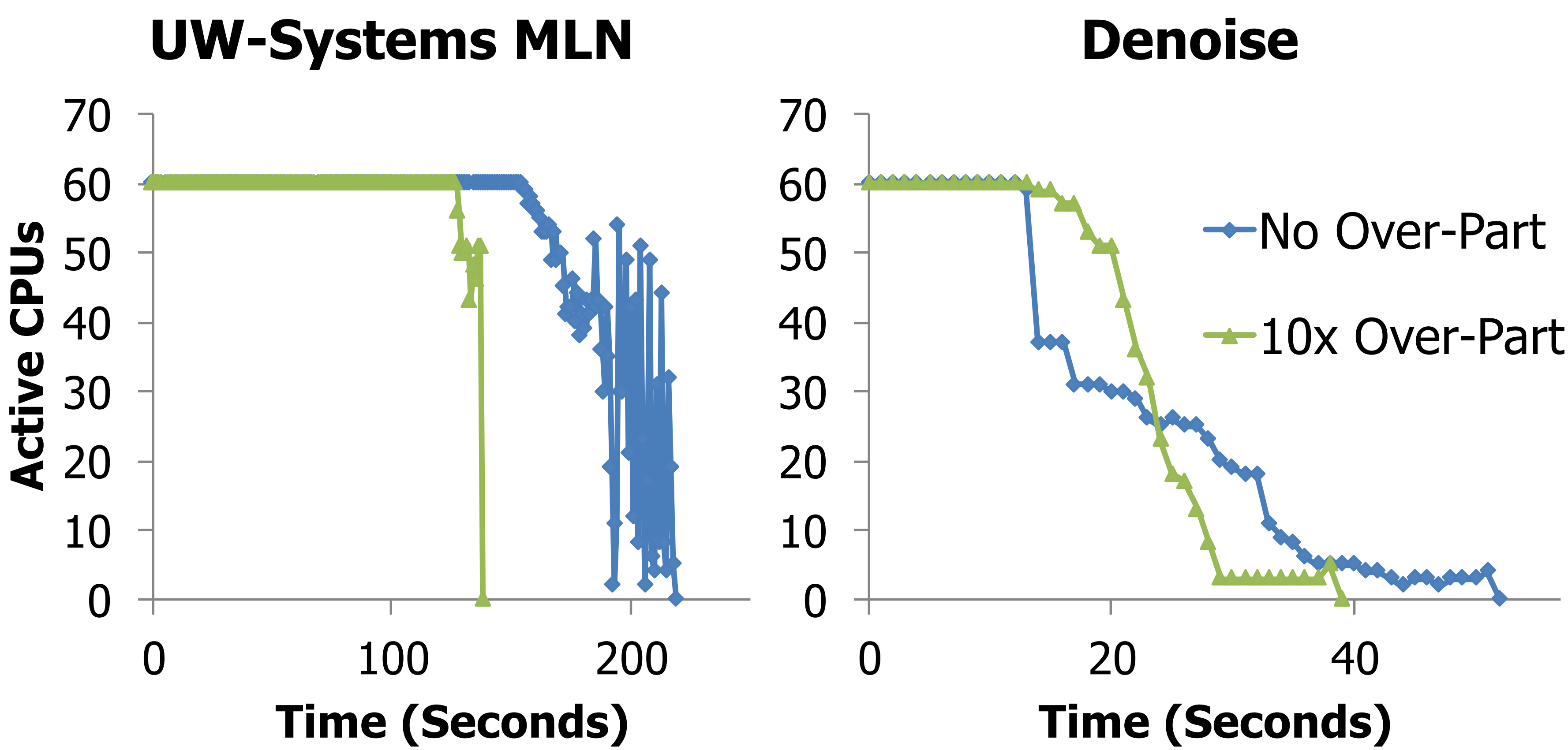
- Provides a simple method to trade between work balance and communication cost



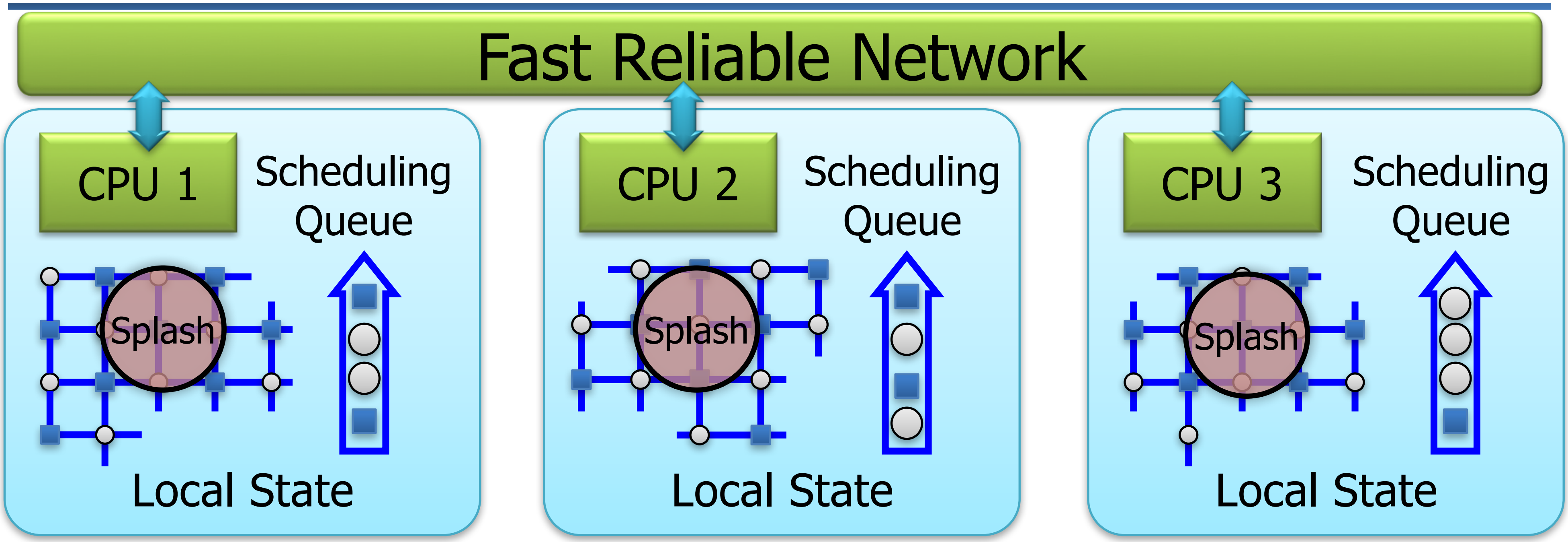


CPU Utilization

- Over-partitioning improves CPU utilization:



Parallel Splash Algorithm



- Over-Partition factor graph
  - Randomly assign pieces to processors
- Schedule Splashes locally using belief residuals
- Transmit messages on boundary

Outline

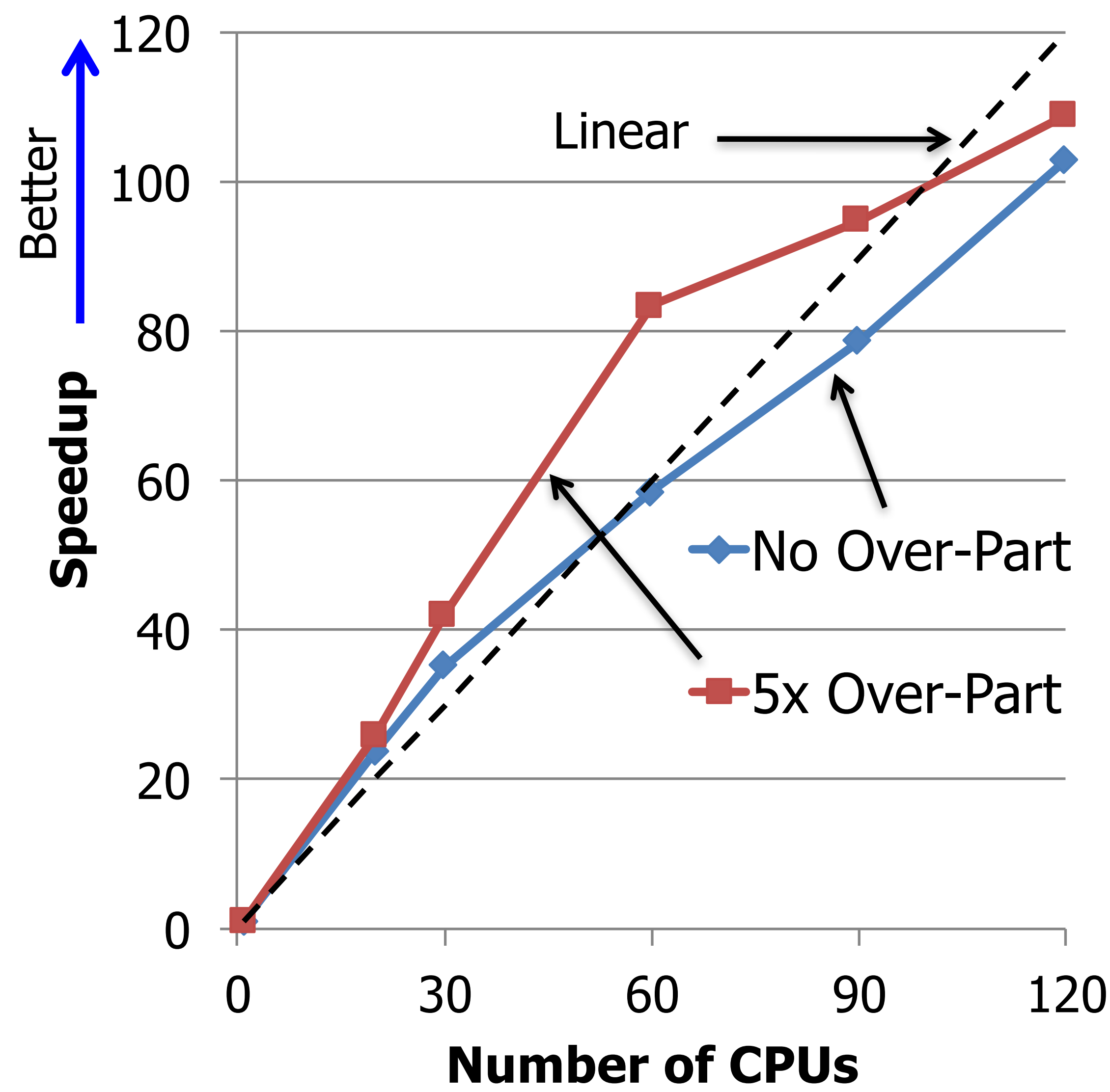
- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_{ε} - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

Experiments

- Implemented in C++ using MPICH2 as a message passing API
- Ran on Intel OpenCirrus cluster: 120 processors
  - 15 Nodes with 2 x Quad Core Intel Xeon Processors
  - Gigabit Ethernet Switch
- Tested on Markov Logic Networks obtained from Alchemy [Domingos et al. SSPR 08]
  - Present results on largest UW-Systems and smallest UW-Languages MLNs

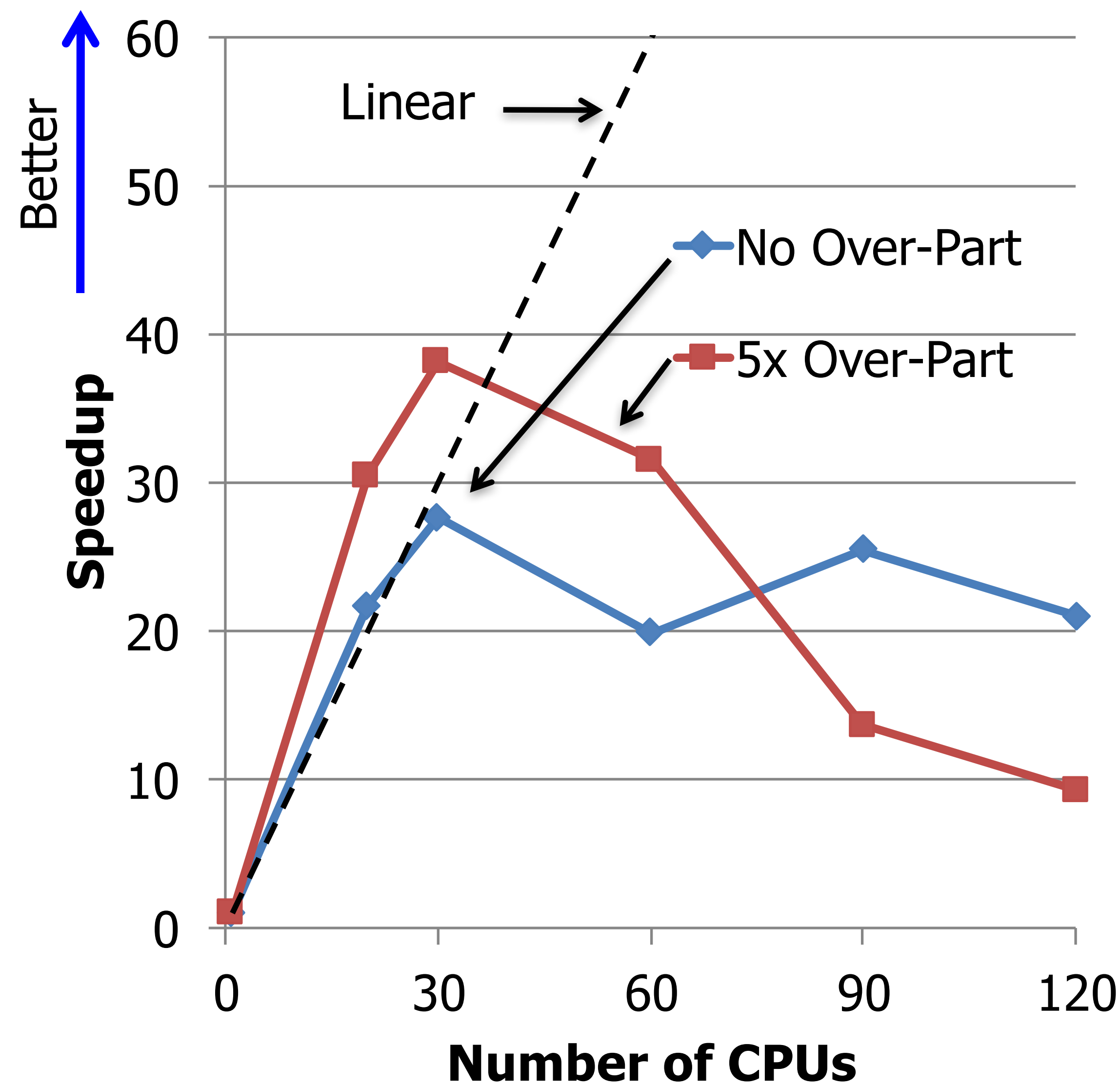
Parallel Performance (Large Graph)

- UW-Systems
  - 8K Variables
  - 406K Factors
- Single Processor Running Time:
  - 1 Hour
- Linear to Super-Linear up to 120 CPUs
  - Cache efficiency



Parallel Performance (Small Graph)

- UW-Languages
  - 1K Variables
  - 27K Factors
- Single Processor Running Time:
  - 1.5 Minutes
- Linear to Super-Linear up to 30 CPUs
  - Network costs quickly dominate short running-time



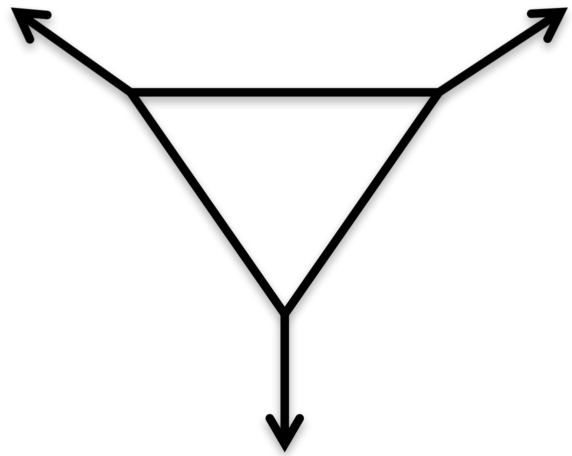
Outline

- Overview
- Graphical Models: Statistical Structure
- Inference: Computational Structure
- τ_{ε} - Approximate Messages: Statistical Structure
- Parallel Splash
  - Dynamic Scheduling
  - Partitioning
- Experimental Results
- Conclusions

Summary

Algorithmic Efficiency

Splash Structure
+
Belief Residual Scheduling



Parallel Efficiency

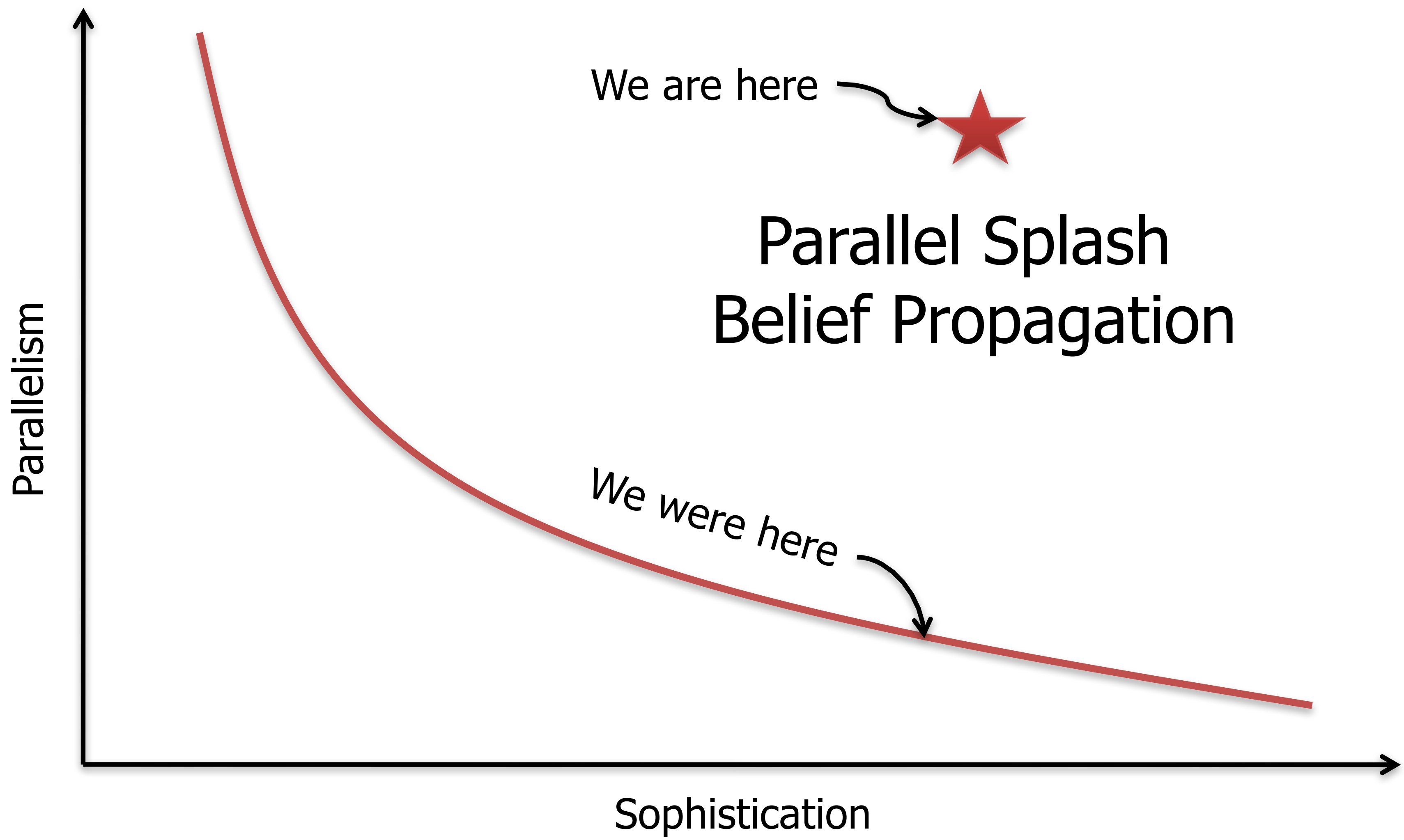
Independent Parallel Splashes

Implementation Efficiency

Distributed Queues
Asynchronous Communication
Over-Partitioning

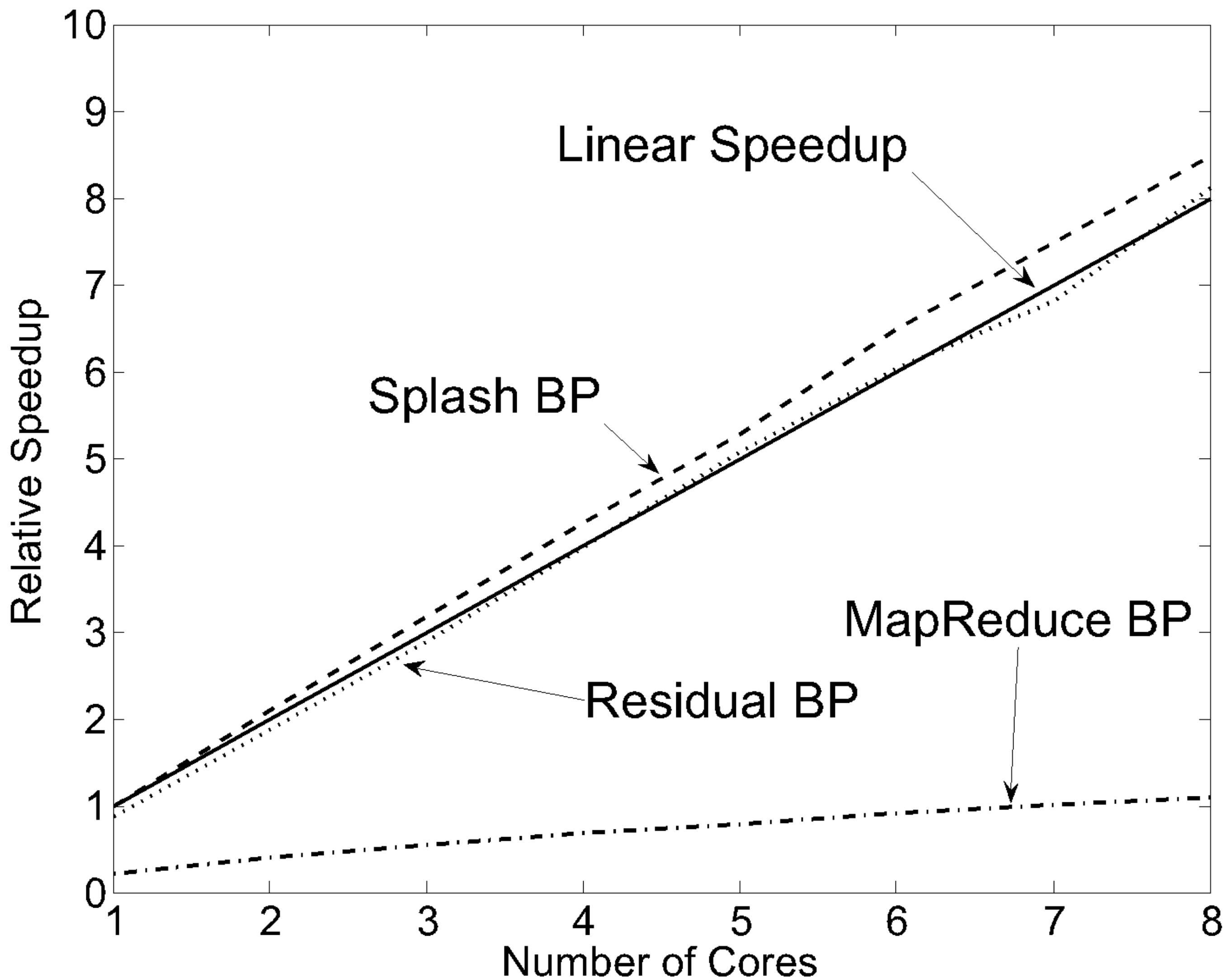
- Experimental results on large factor graphs:
  - **Linear** to **super-linear** speed-up using up to 120 processors

Conclusion

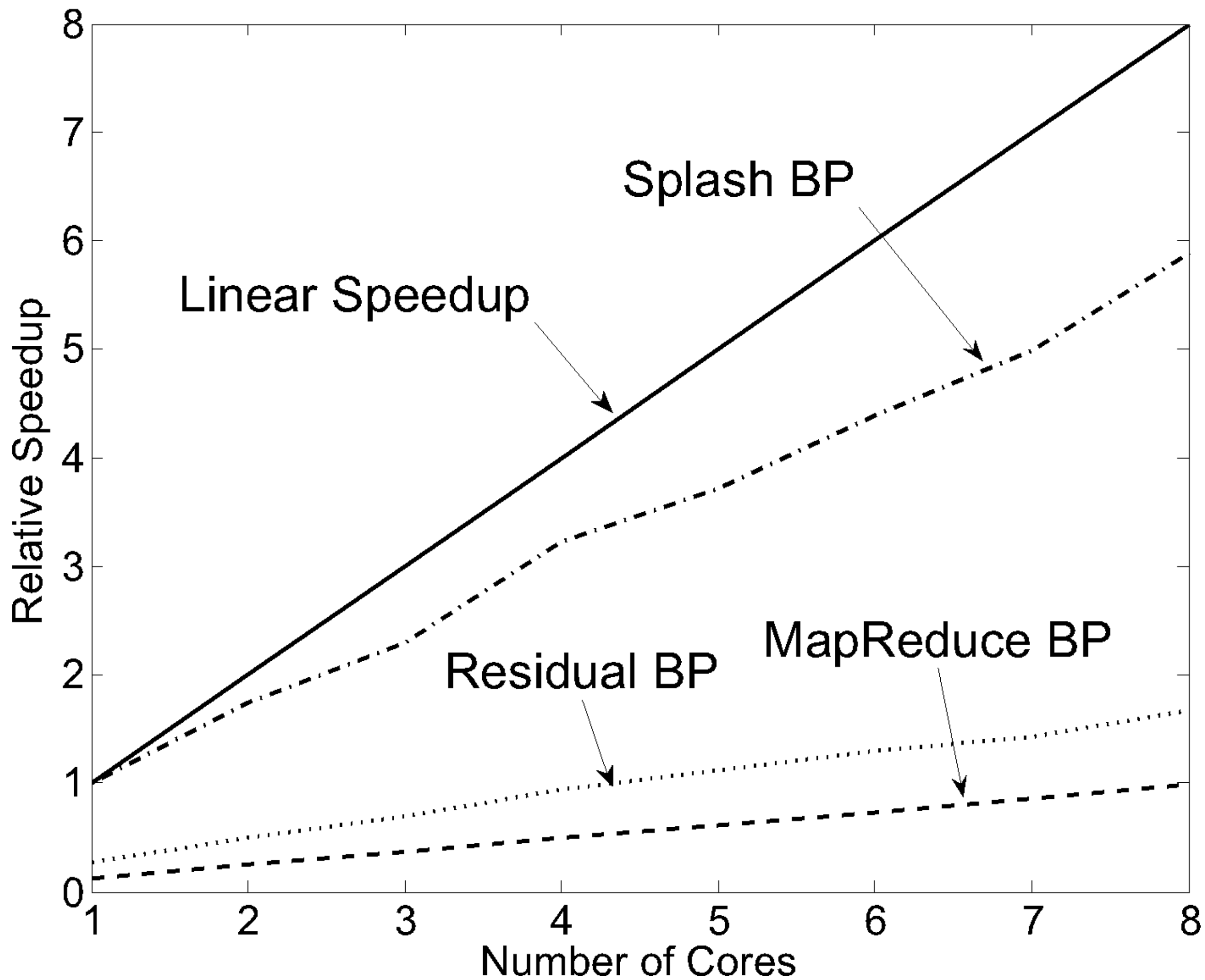

We are here
Parallel Splash
Belief Propagation
We were here
Parallelism
Sophistication

Questions

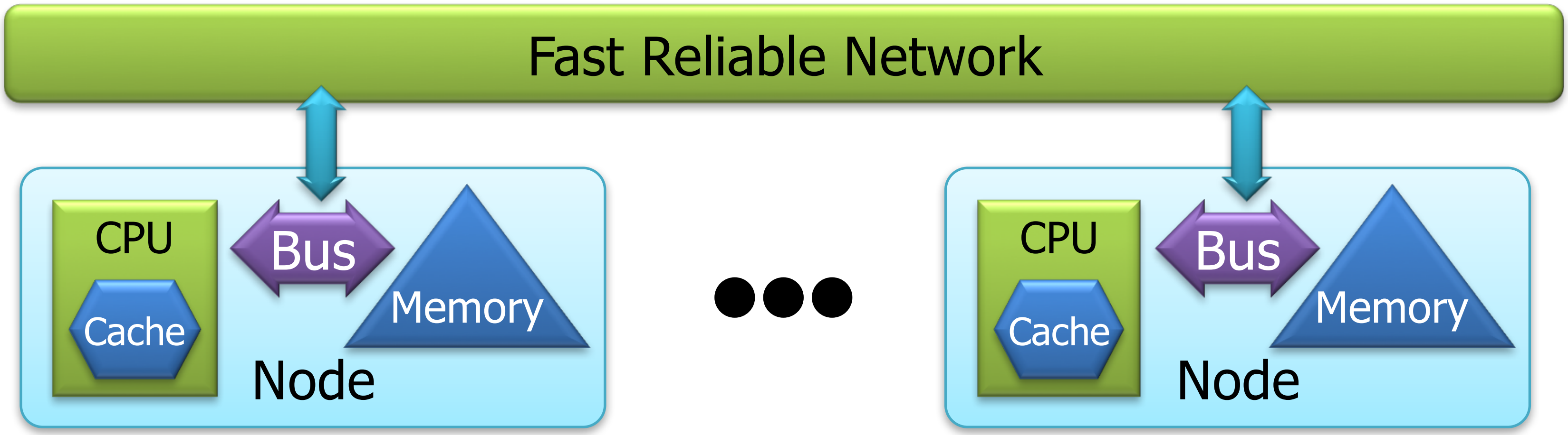
Protein Results



3D Video Task



Distributed Parallel Setting



- Opportunities:
  - Access to larger systems: 8 CPUs → 1000 CPUs
  - Linear Increase:
    - RAM, **Cache Capacity,** and **Memory Bandwidth**
- Challenges:
  - Distributed state, Communication and Load Balancing