Distributed Parallel Inference on Large Factor Graphs

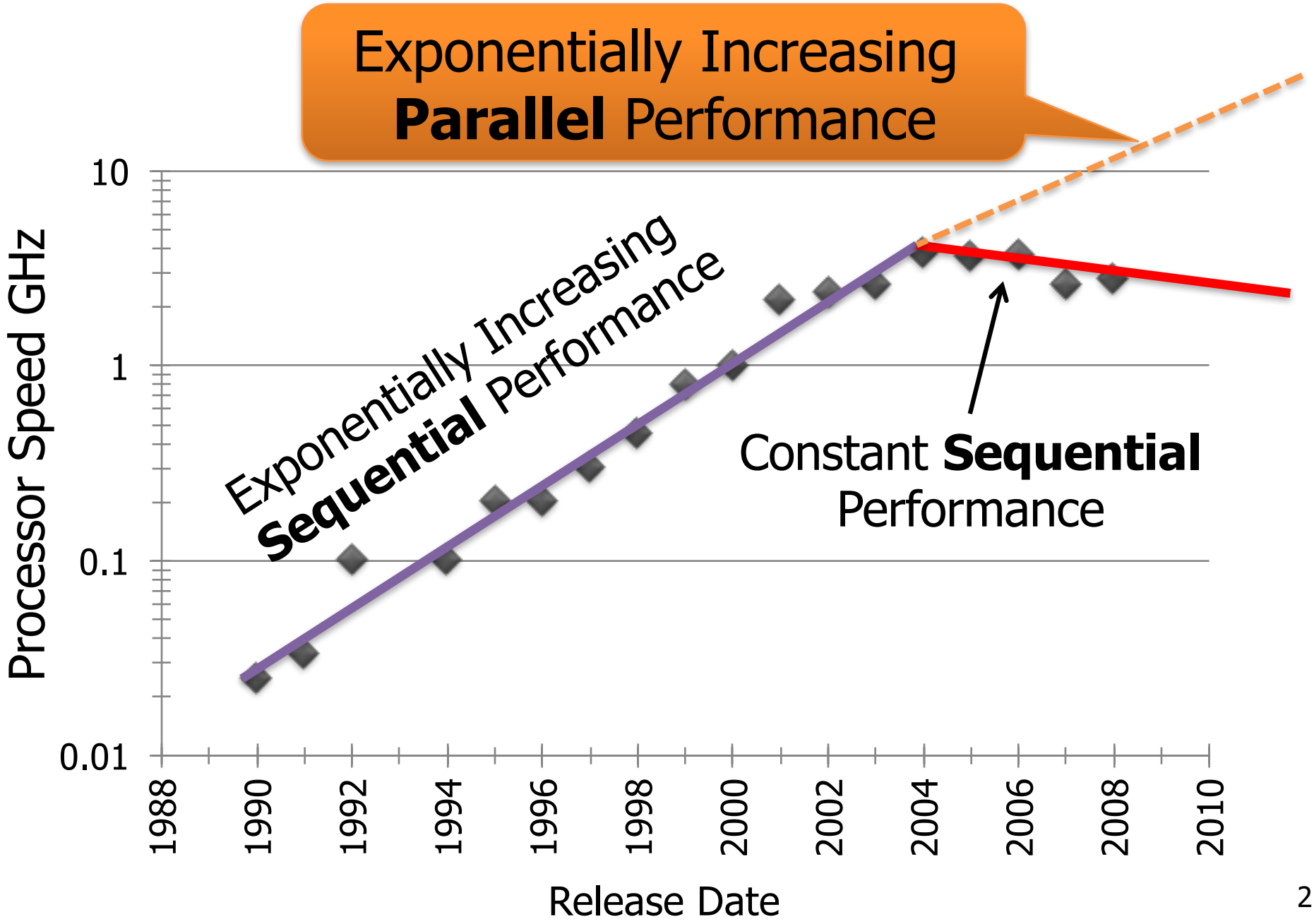
Joseph E. Gonzalez
Yucheng Low
Carlos Guestrin
David O'Hallaron

Select Lab

Carnegie Mellon

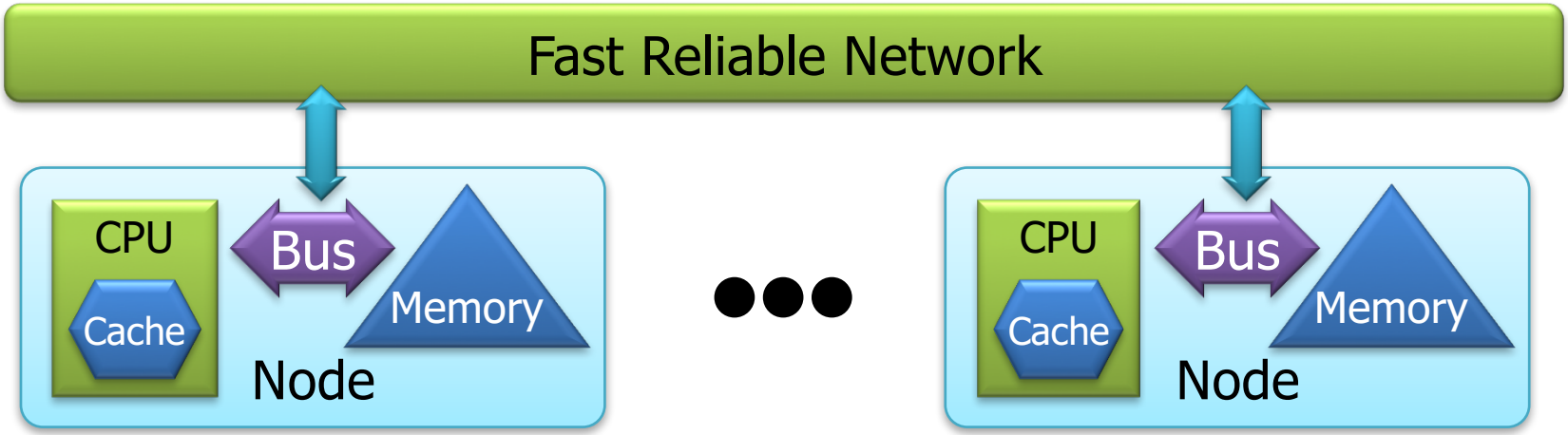
sense learn act

Exponential Parallelism



sense learn act

Distributed Parallel Setting

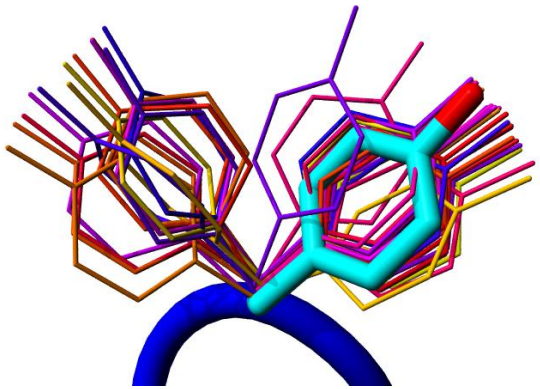


- Opportunities:
  - Access to larger systems: 8 CPUs → 1000 CPUs
  - Linear Increase:
    - RAM, **Cache Capacity,** and **Memory Bandwidth**
- Challenges:
  - Distributed state, Communication and Load Balancing

Graphical Models and Parallelism

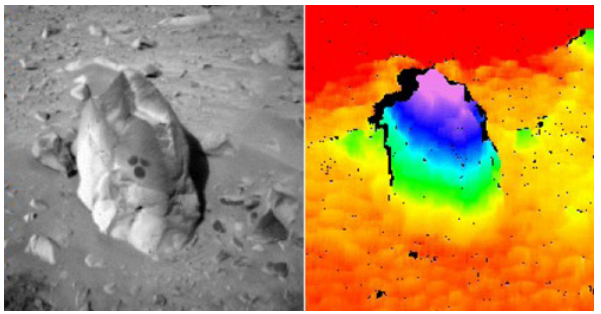
*Graphical models provide a common language for **general purpose** parallel algorithms in machine learning*

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



Protein Structure Prediction

Movie Recommendation

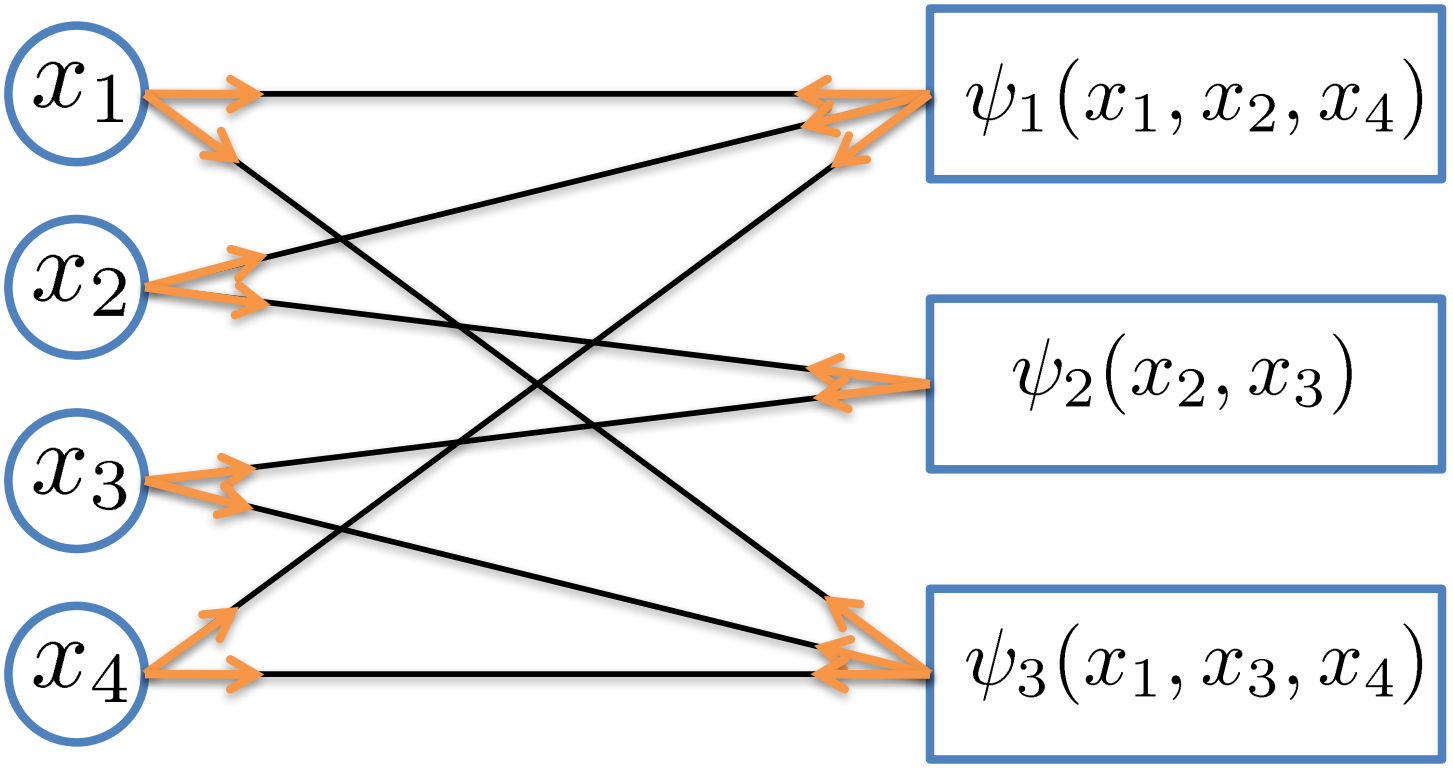


Computer Vision

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

Belief Propagation (BP)

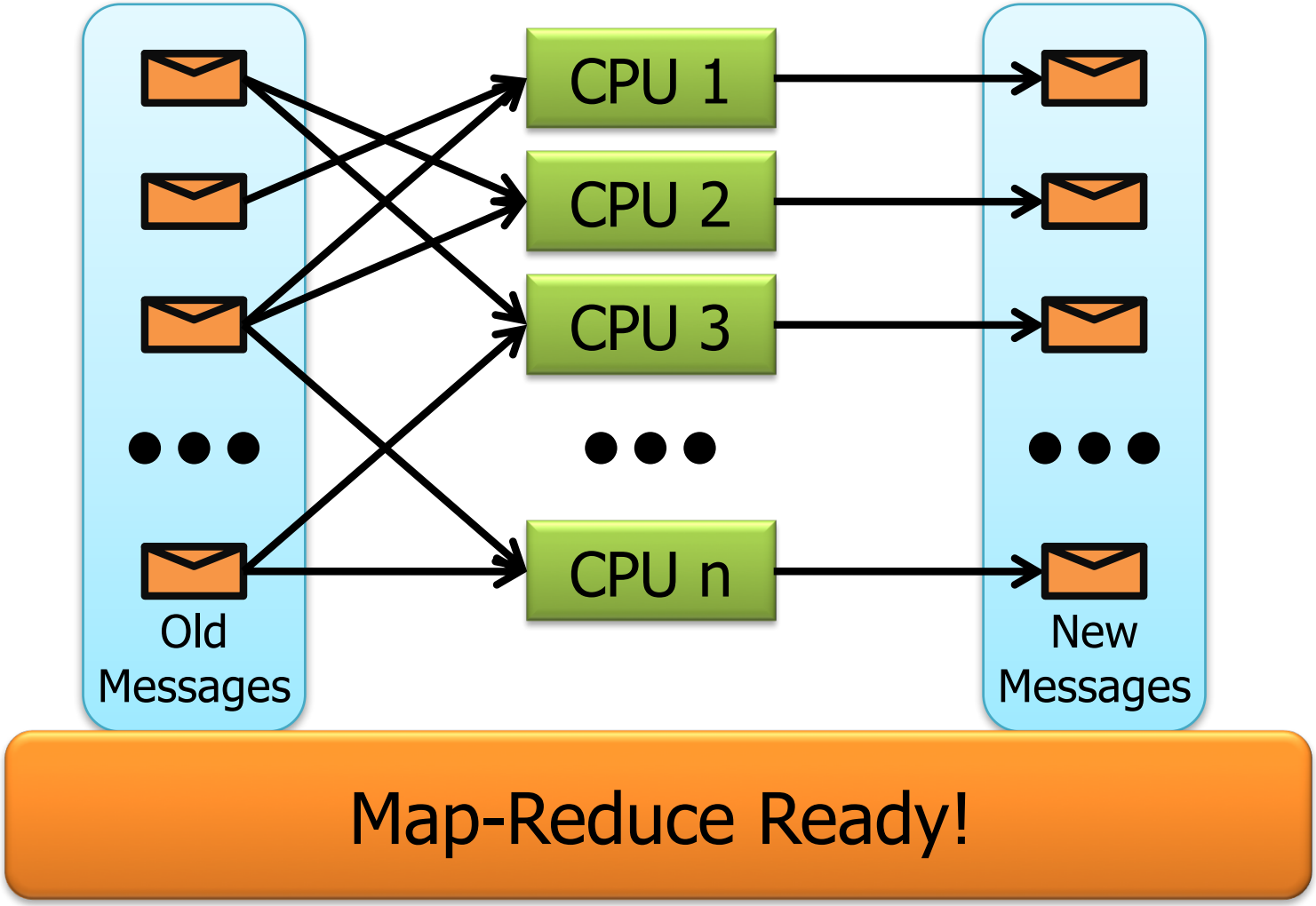
- Message passing algorithm



Naturally Parallel Algorithm

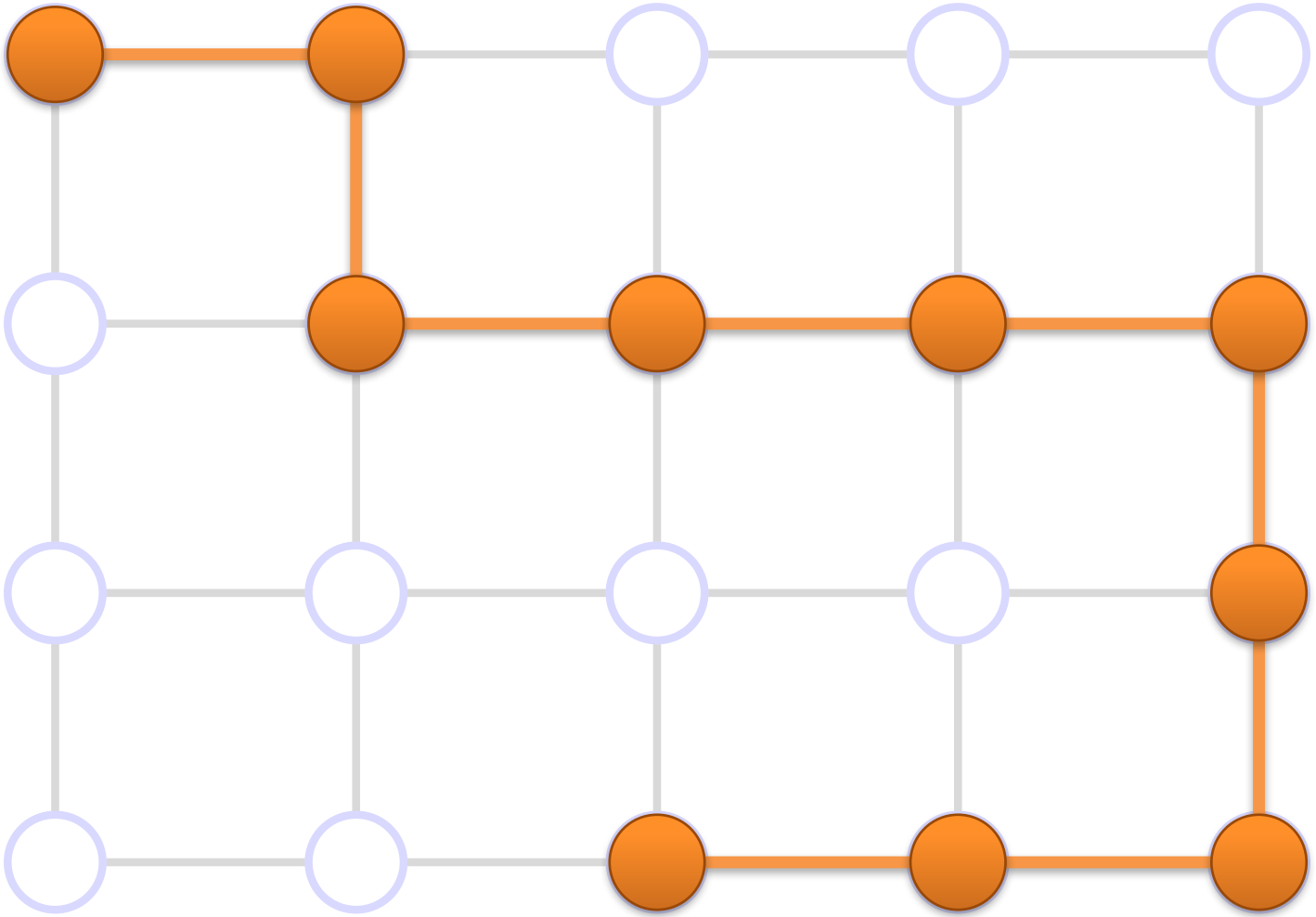
Parallel Synchronous BP

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



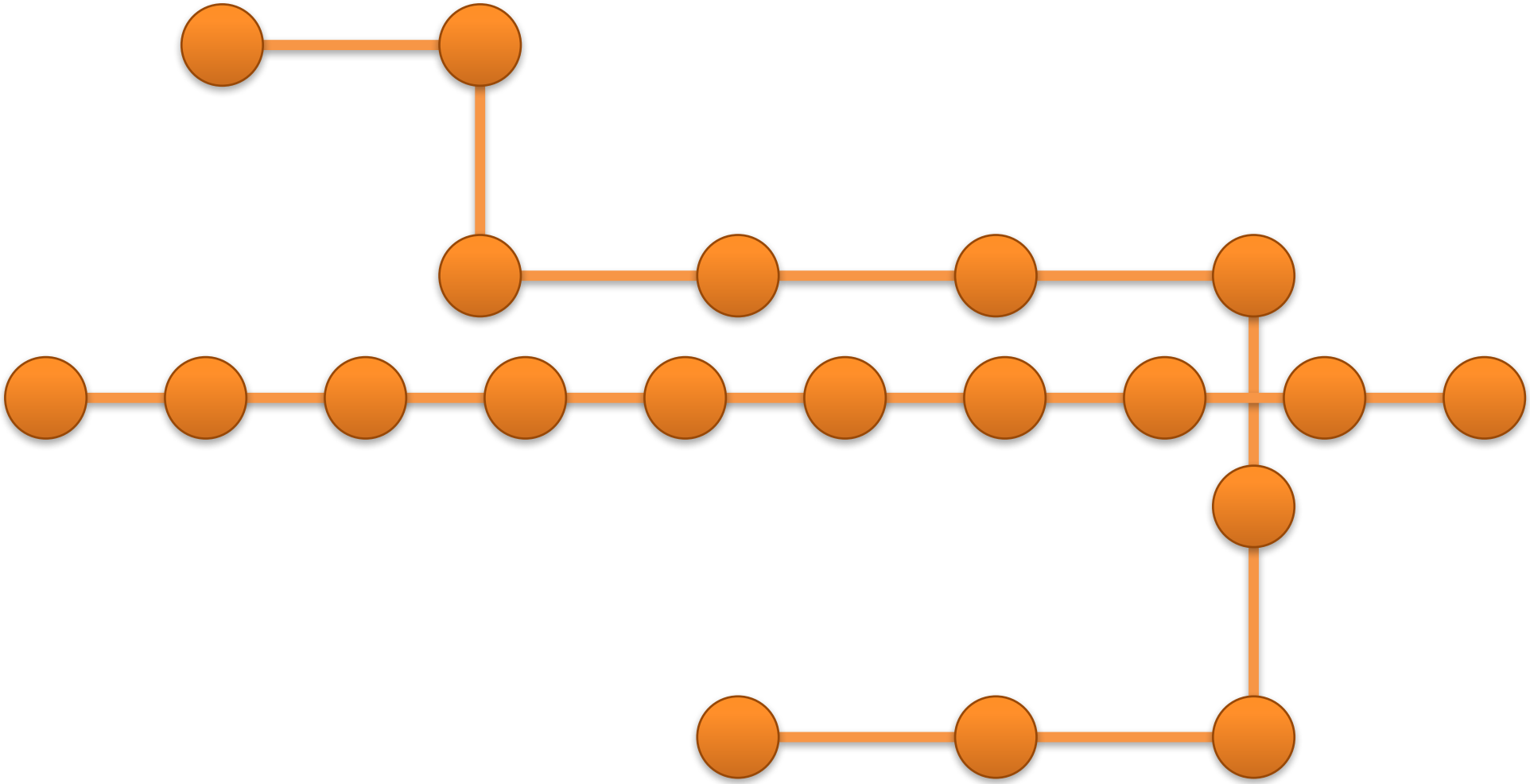
sense
learn
act

Hidden Sequential Structure

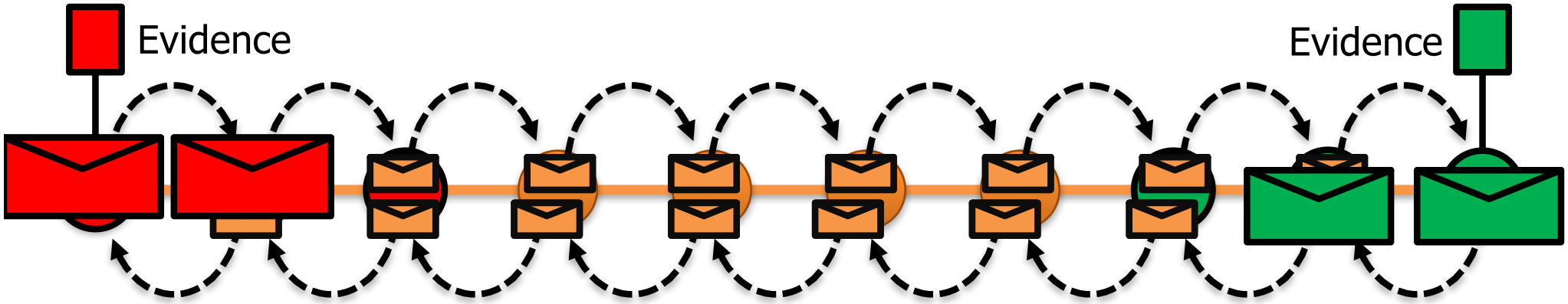


sense
learn
act

Hidden Sequential Structure



Hidden Sequential Structure



- Running Time:

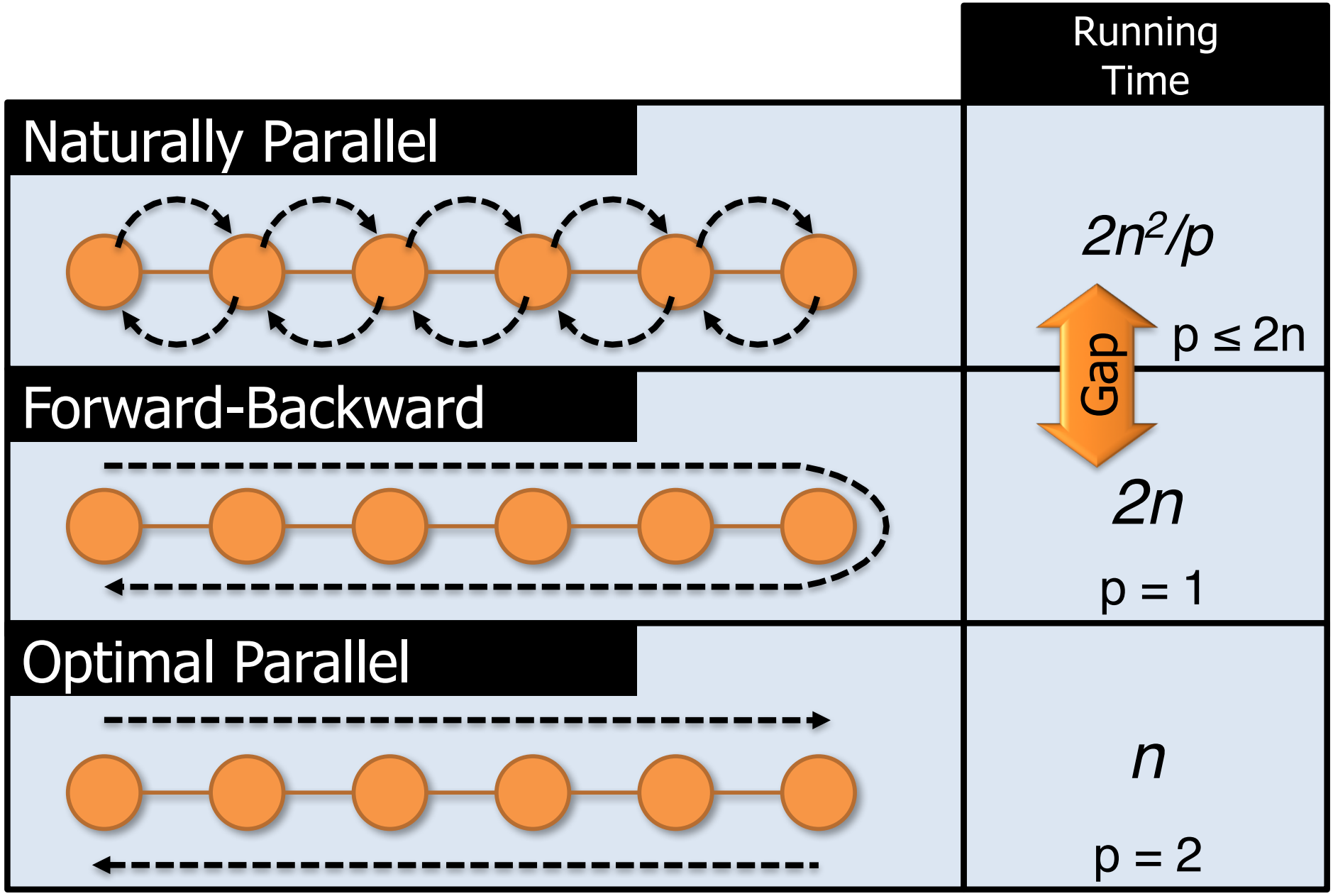
$$\frac{2n \text{ Messages Calculations}}{p \text{ Processors}} \times (n \text{ Iterations to Converge}) = \frac{2n^2}{p}$$

Time for a single parallel iteration

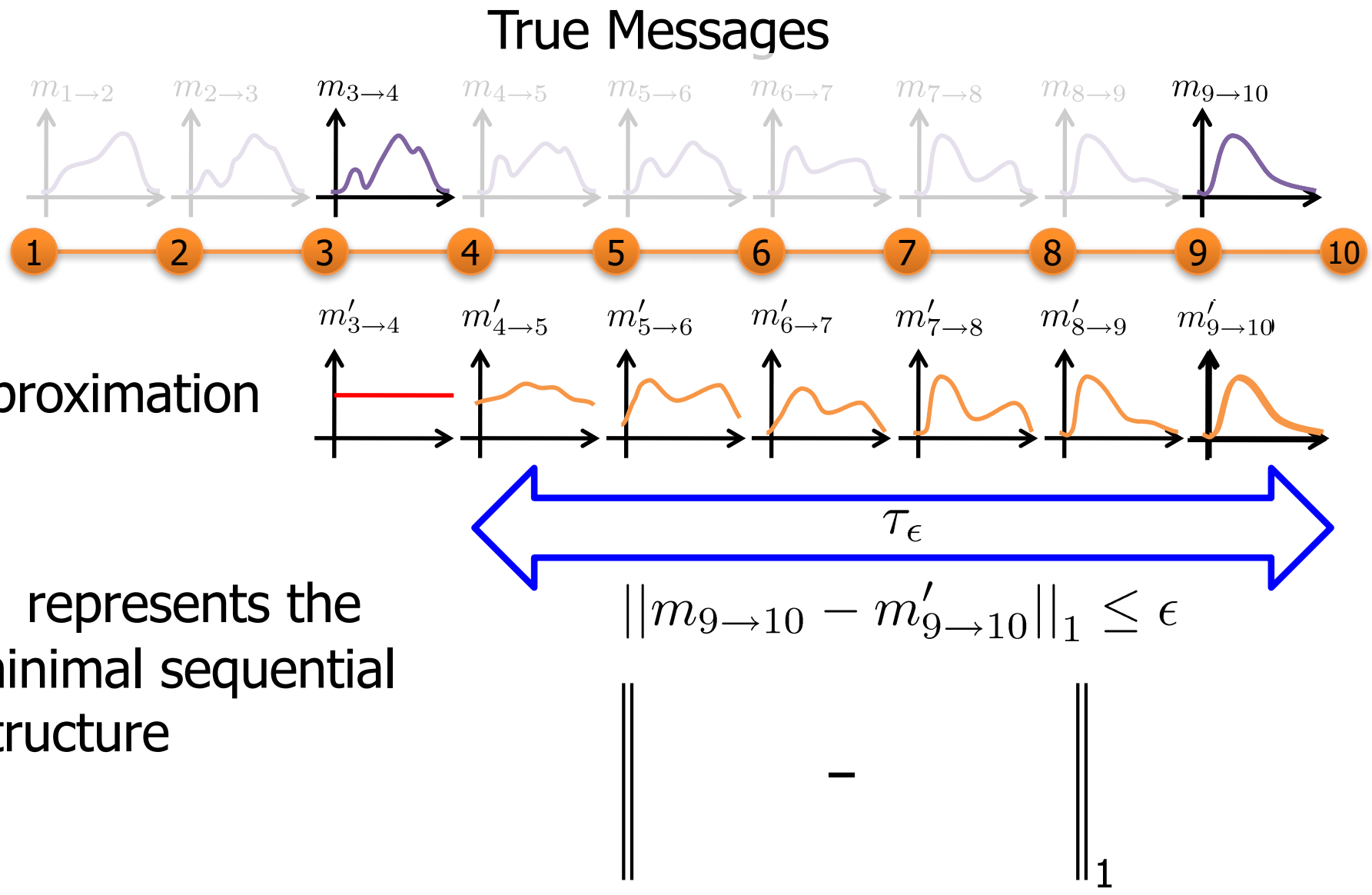
Number of Iterations

sense learn act

Optimal Sequential Algorithm



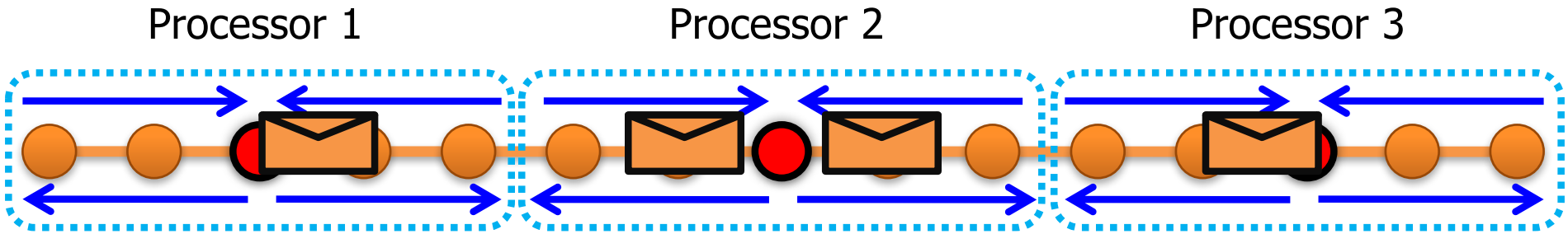
Parallelism by Approximation



τ_ε -Approximation

- τ_ε represents the minimal sequential structure

Optimal Parallel Scheduling



- In [AIStats 09] we demonstrated that this algorithm is **optimal**:

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

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

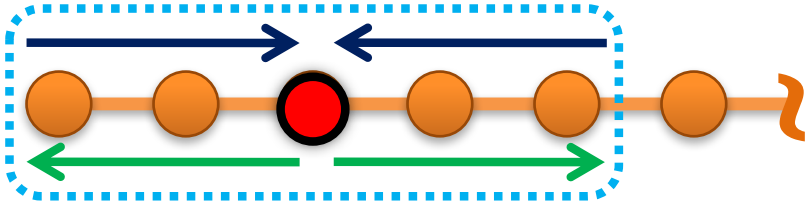
Gap

Parallel Component

Sequential Component

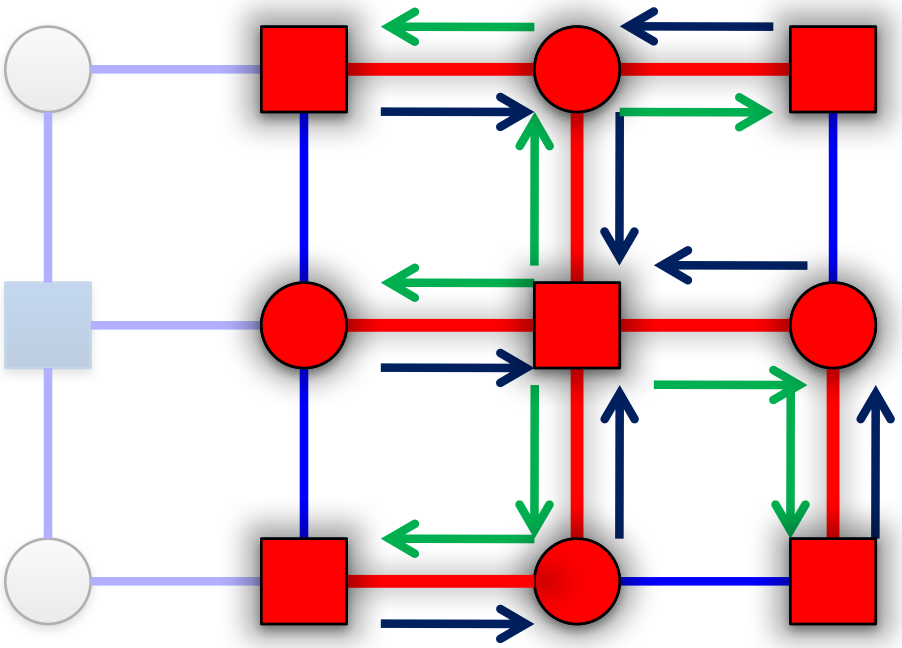
The Splash Operation

- 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



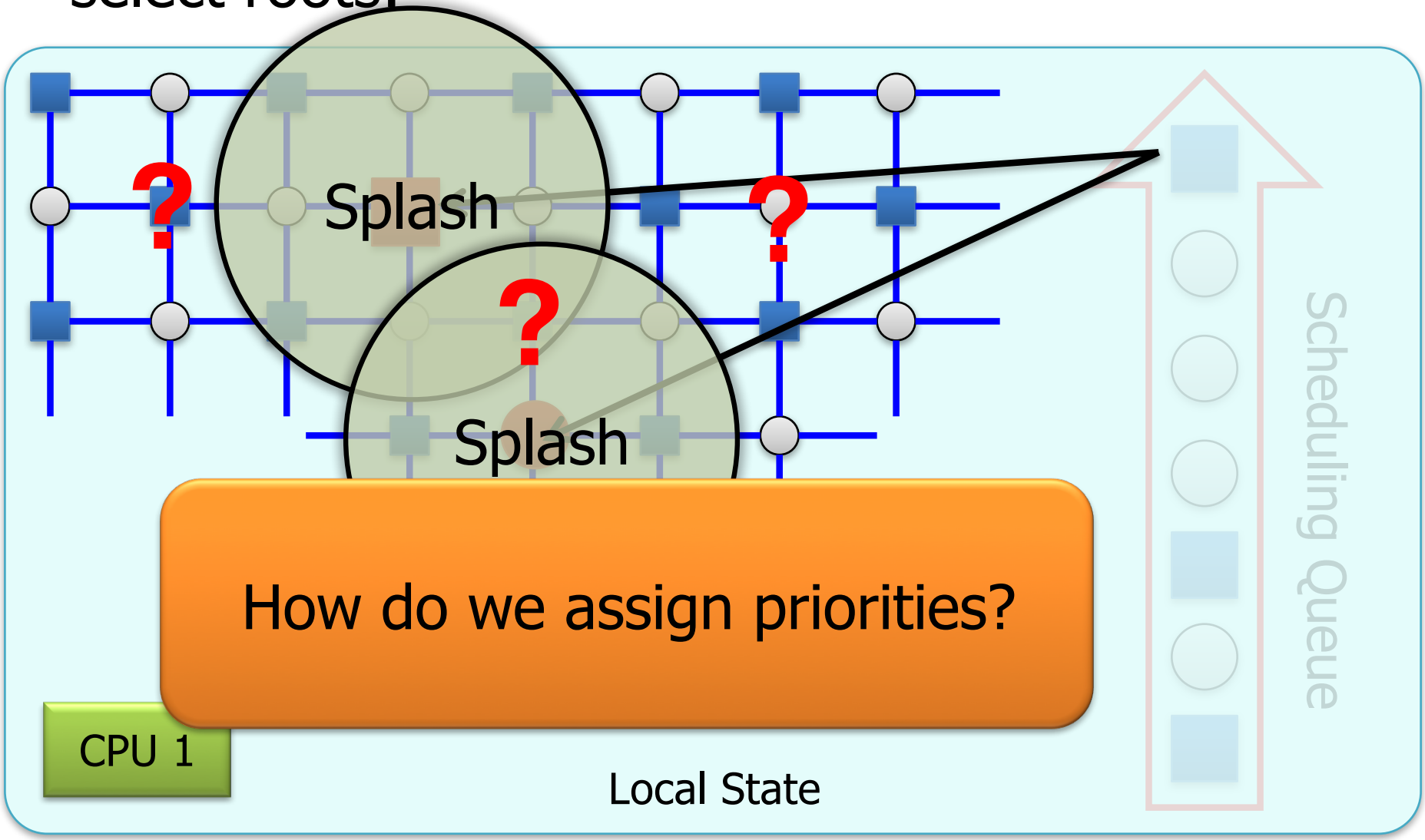
Running Parallel Splashes

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

sense learn act

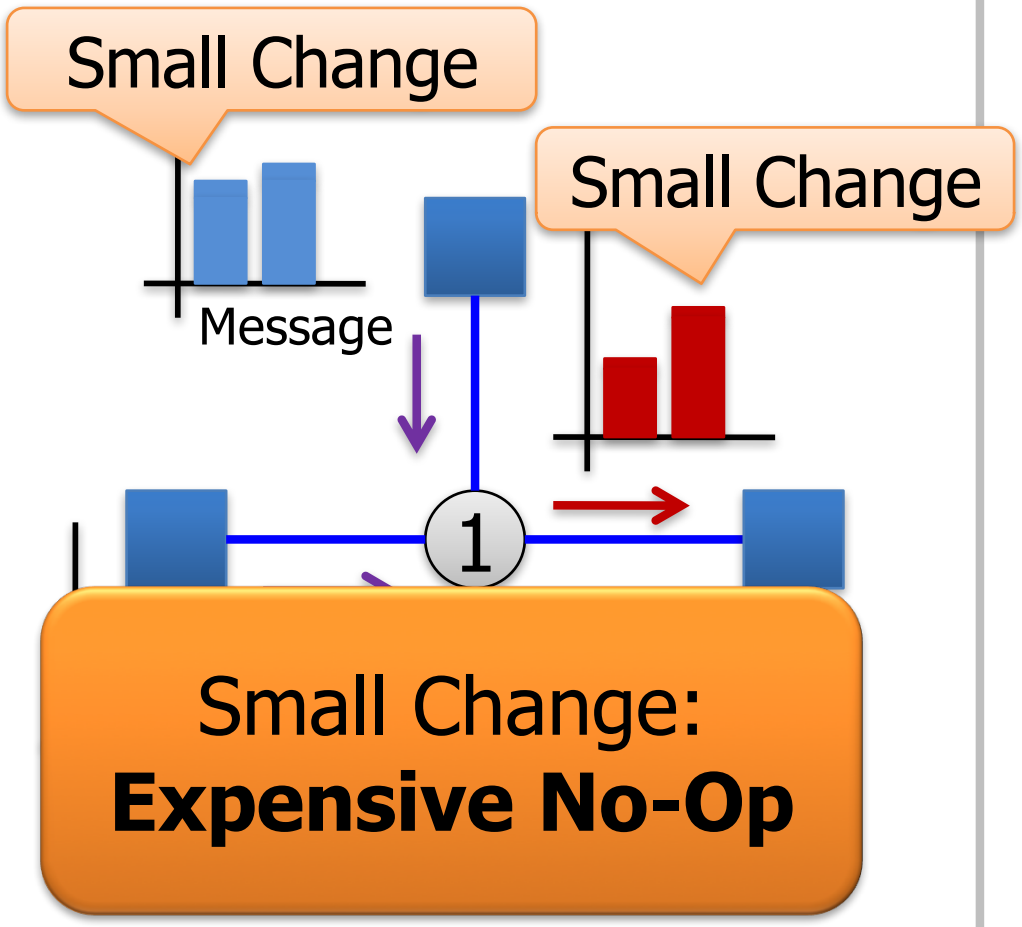
Where do we Splash?

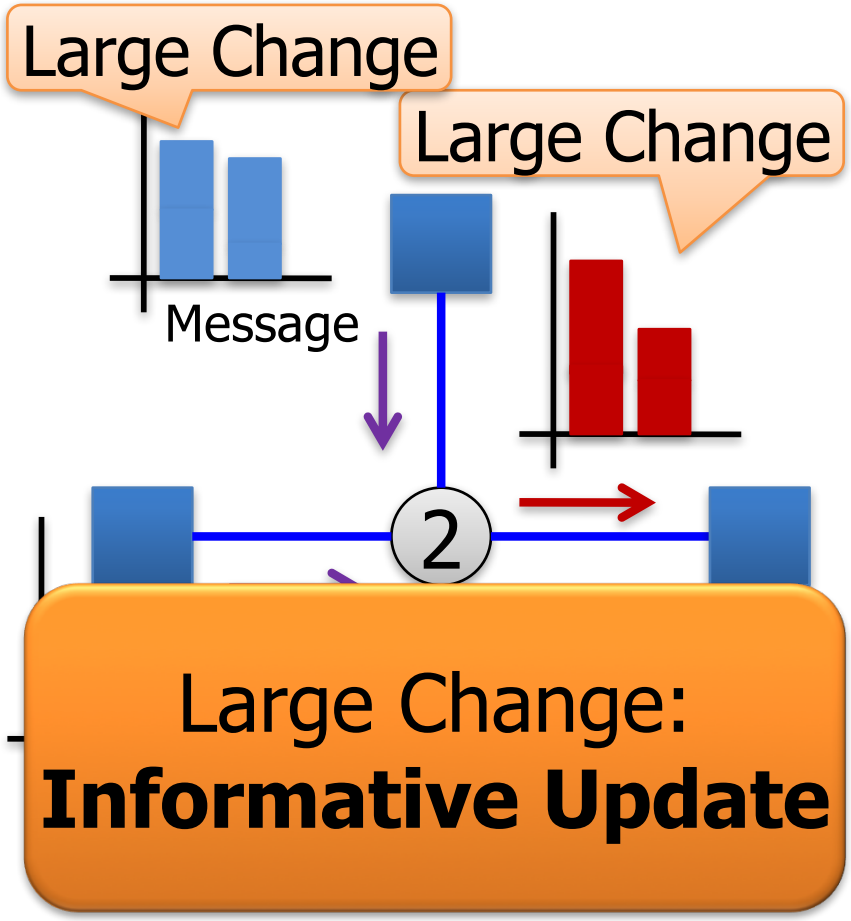
- Assign priorities and use a scheduling queue to select roots:



Message Scheduling

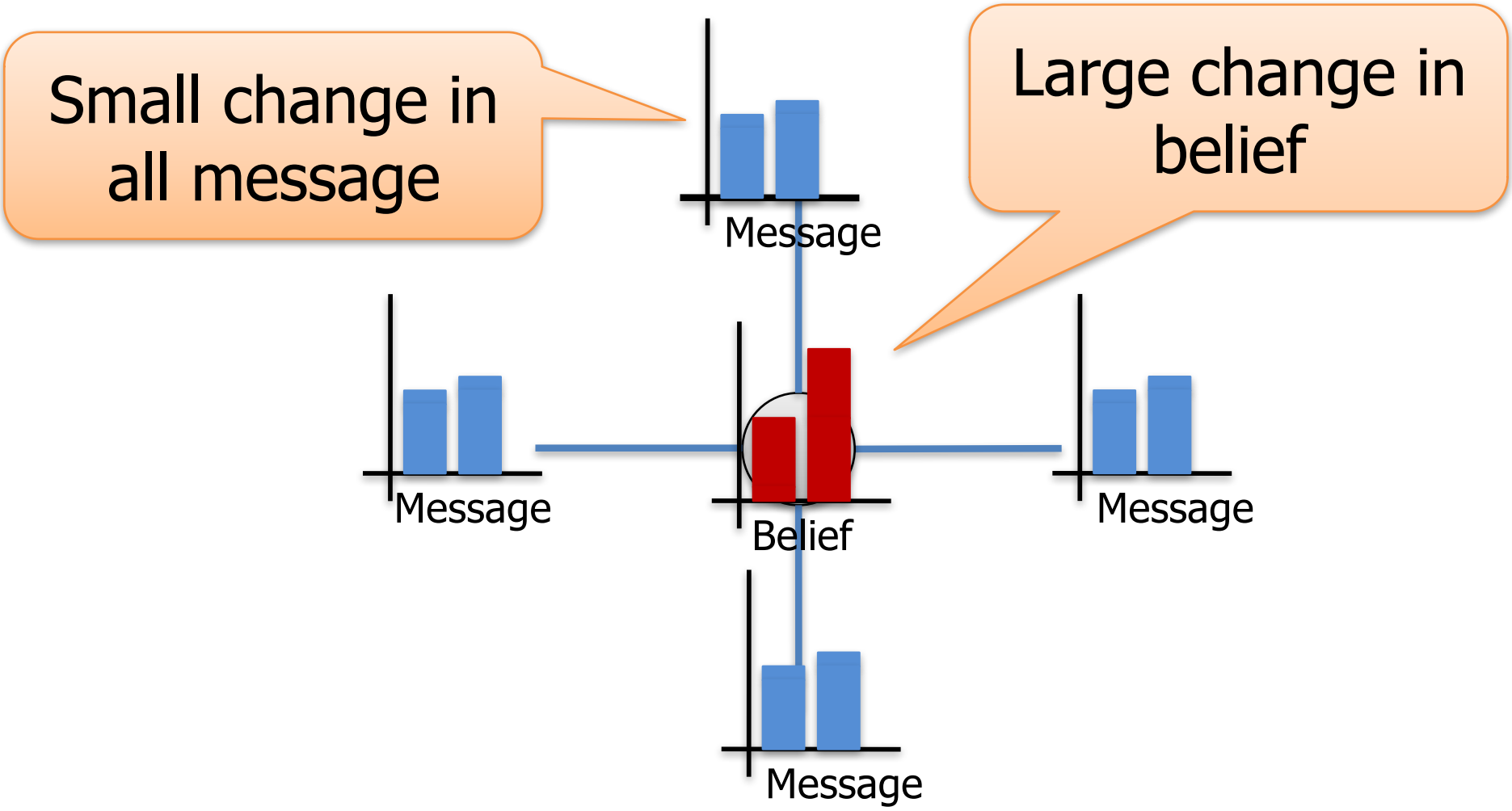
- Residual Belief Propagation [Elidan et al., UAI 06]:
  - Assign priorities based on change in inbound messages





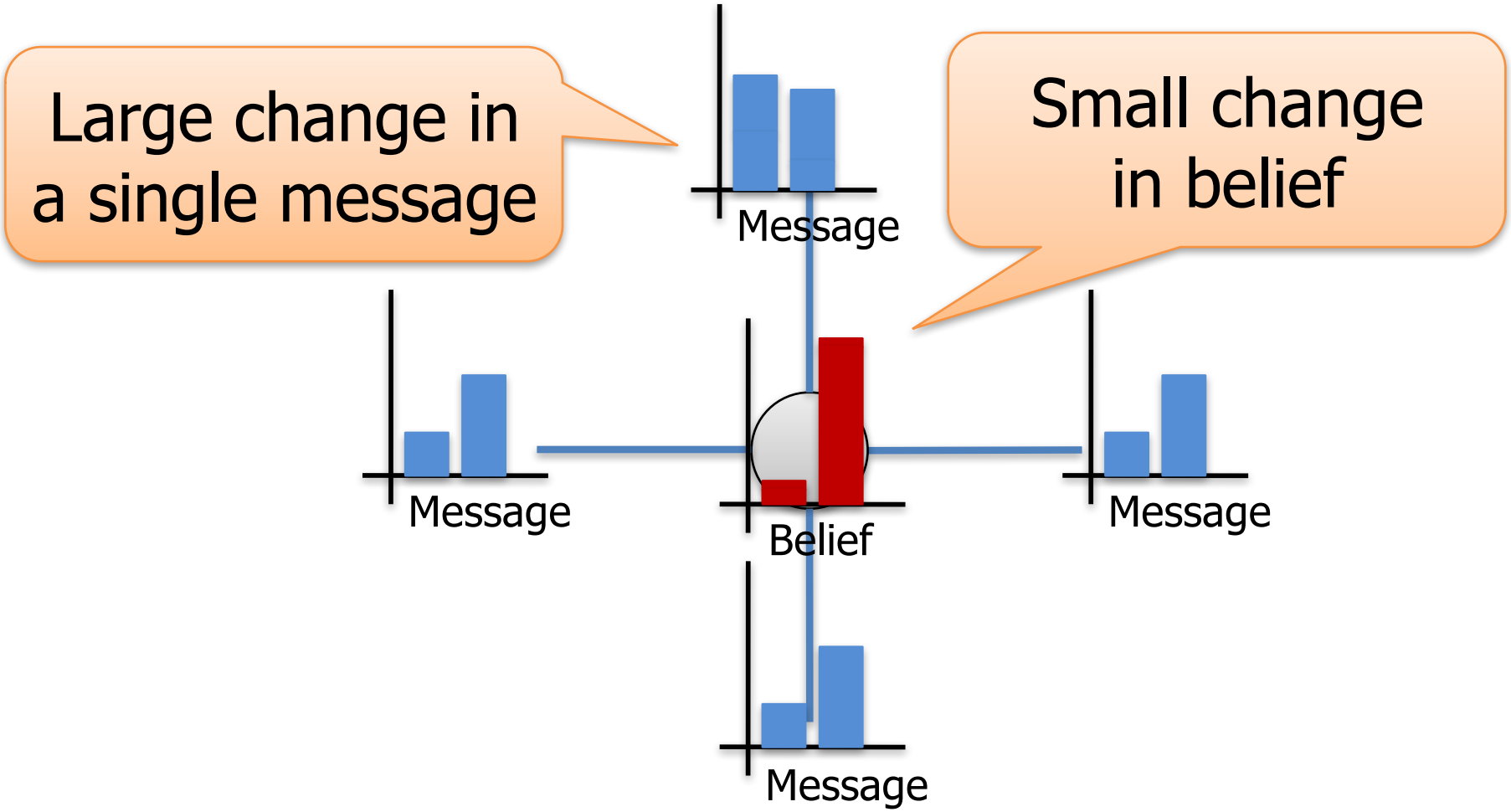
Problem with Message Scheduling

- 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:

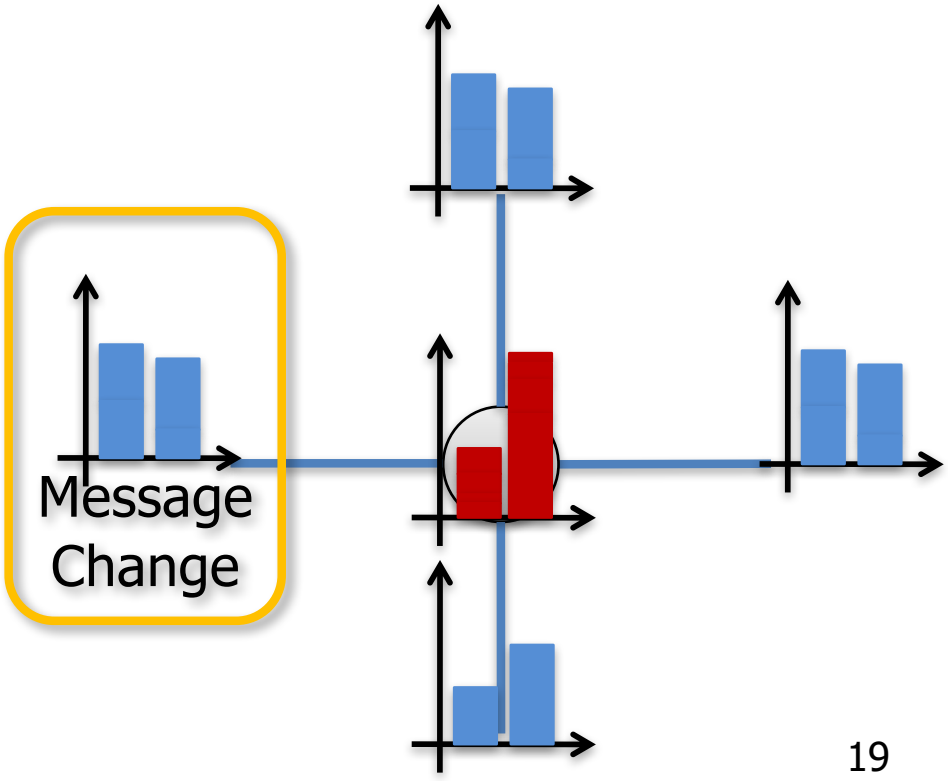


Belief Residual Scheduling

- Assign priorities based on the cumulative change in belief:

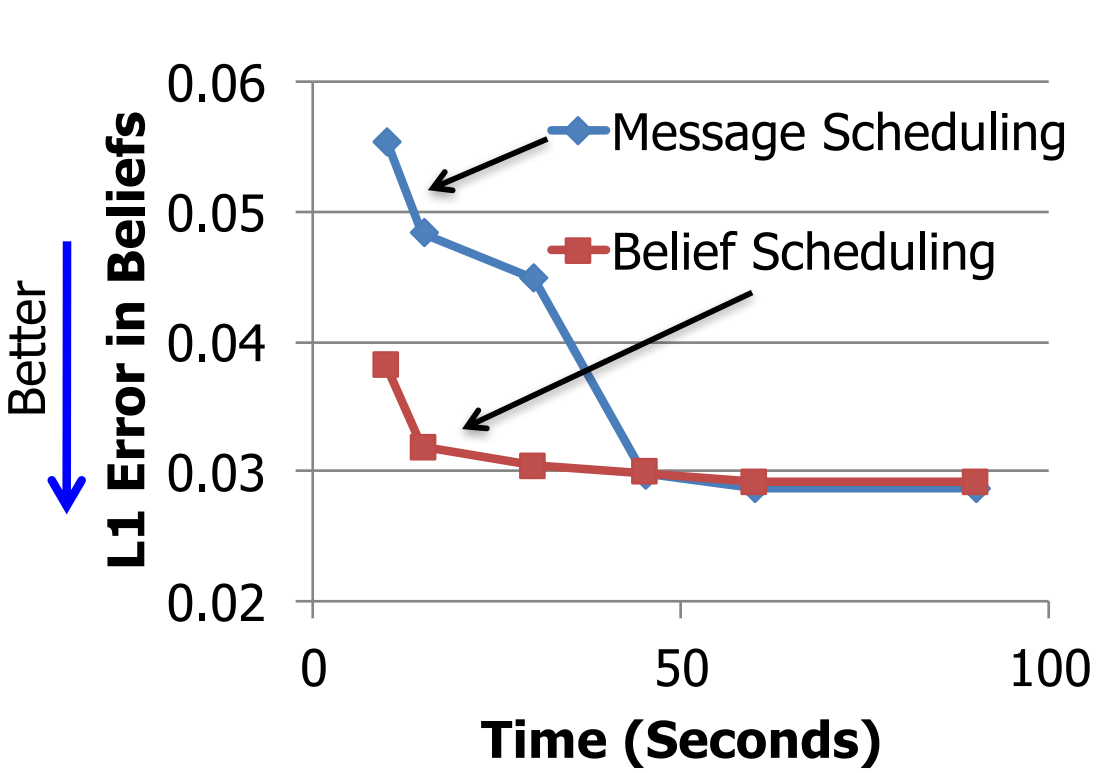
$$r_v = \left\| \; - \; \right\|_1 + \left\| \; - \; \right\|_1 + \left\| \; - \; \right\|_1$$

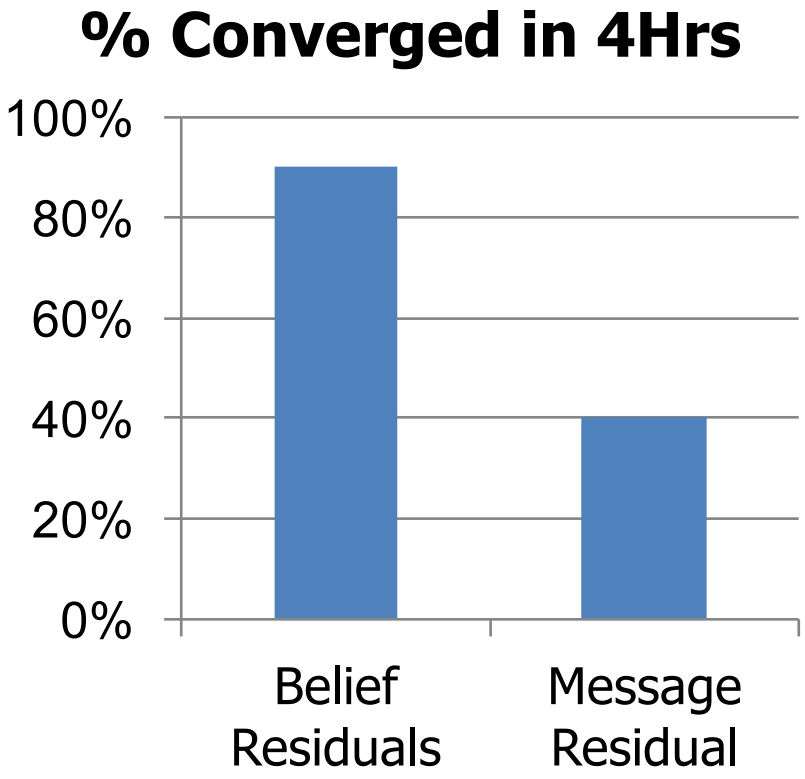
A vertex whose belief has changed substantially since last being updated will likely produce informative new messages.



Message vs. Belief Scheduling

- Belief scheduling improves accuracy more quickly
- Belief scheduling improves convergence

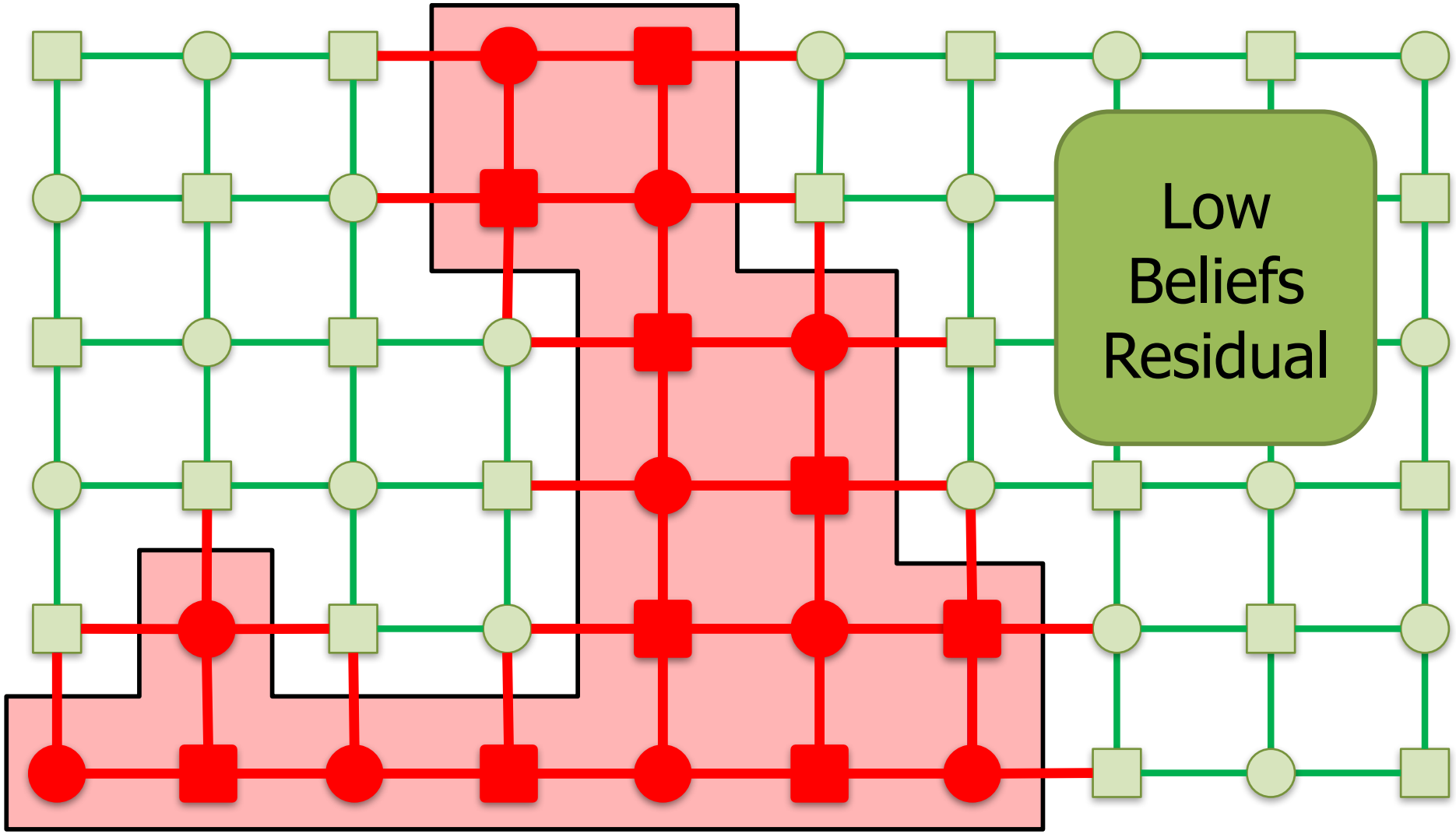




sense learn act

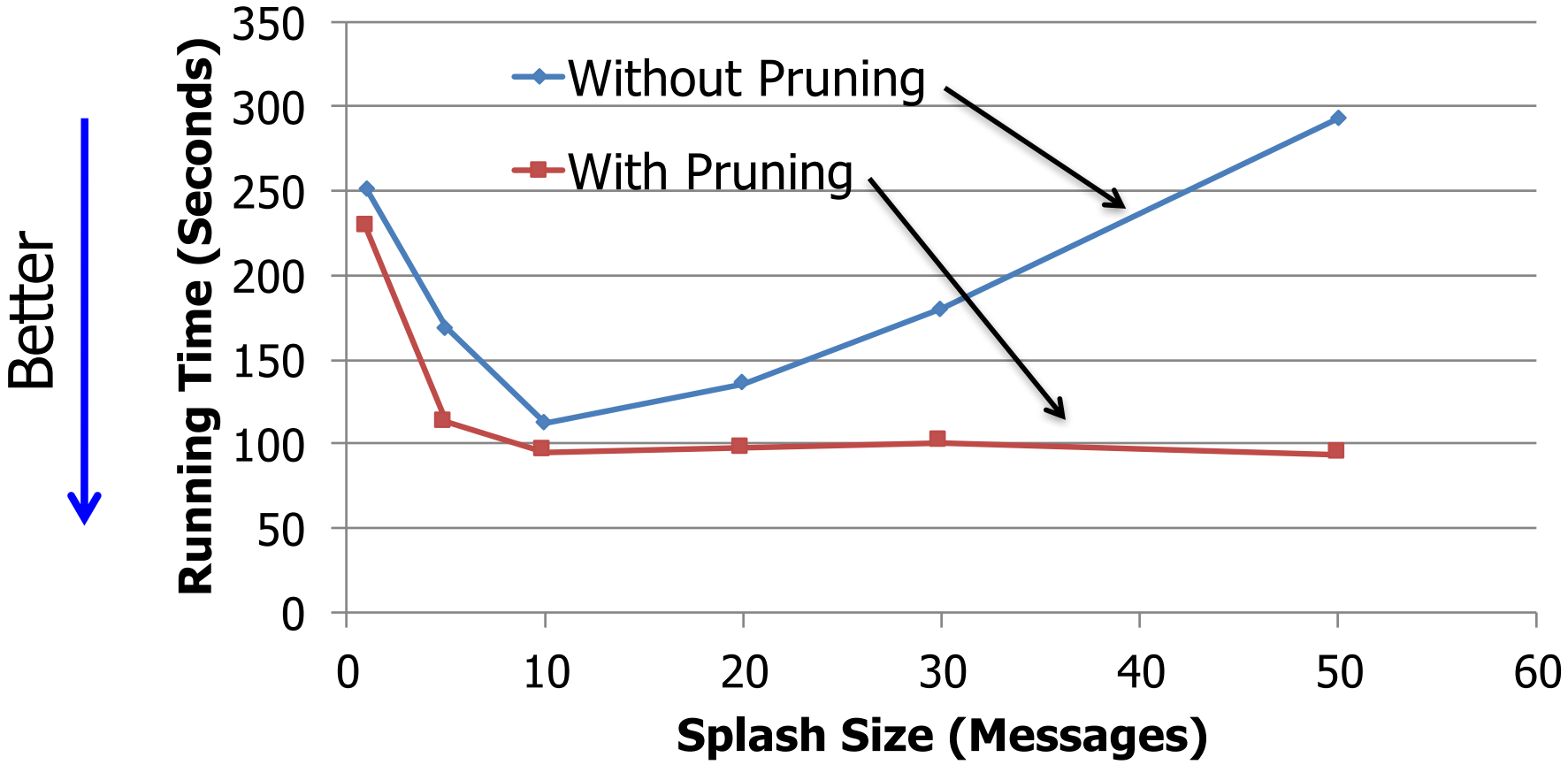
Splash Pruning

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

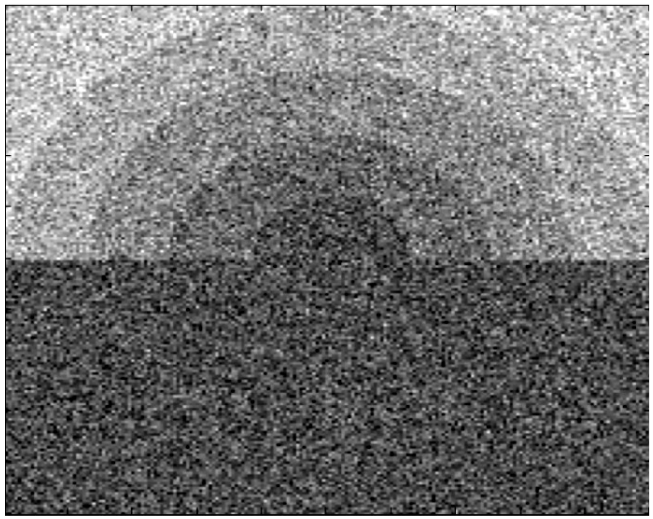


Splash Size

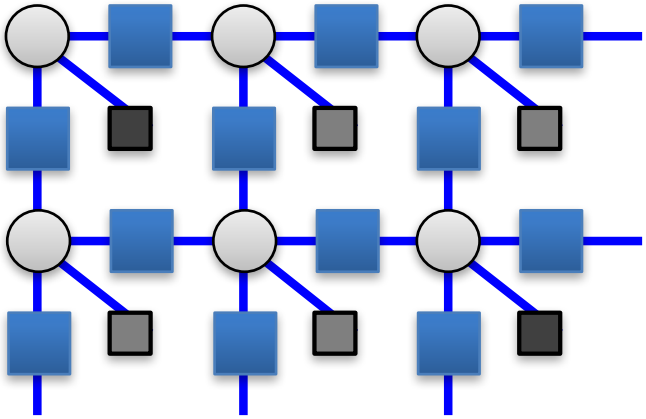
- Using **Splash Pruning** our algorithm is able to dynamically select the **optimal** splash size



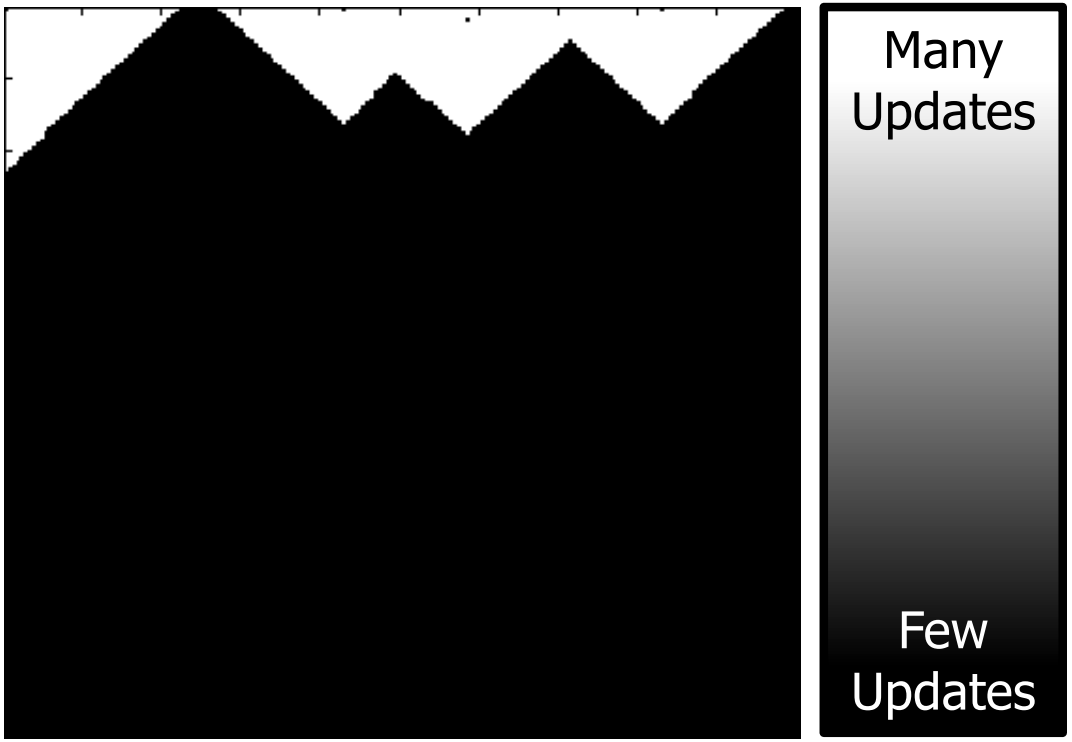
Example



Synthetic Noisy Image



Factor Graph



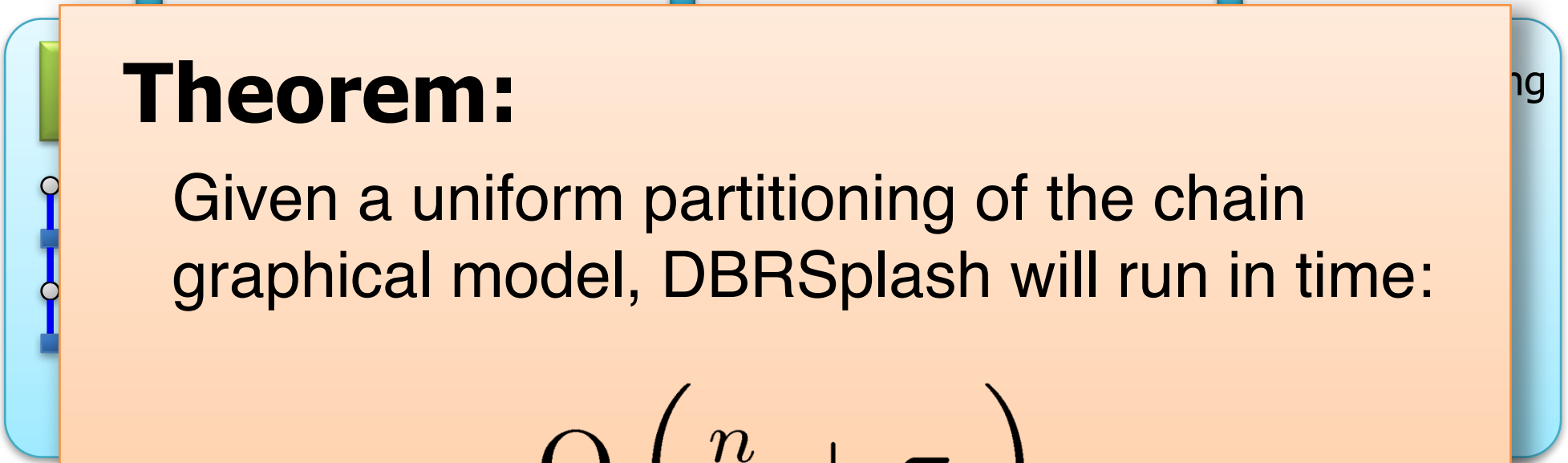
Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

sense learn act

Distributed Belief Residual Splash Algorithm

Fast Reliable Network



Theorem:

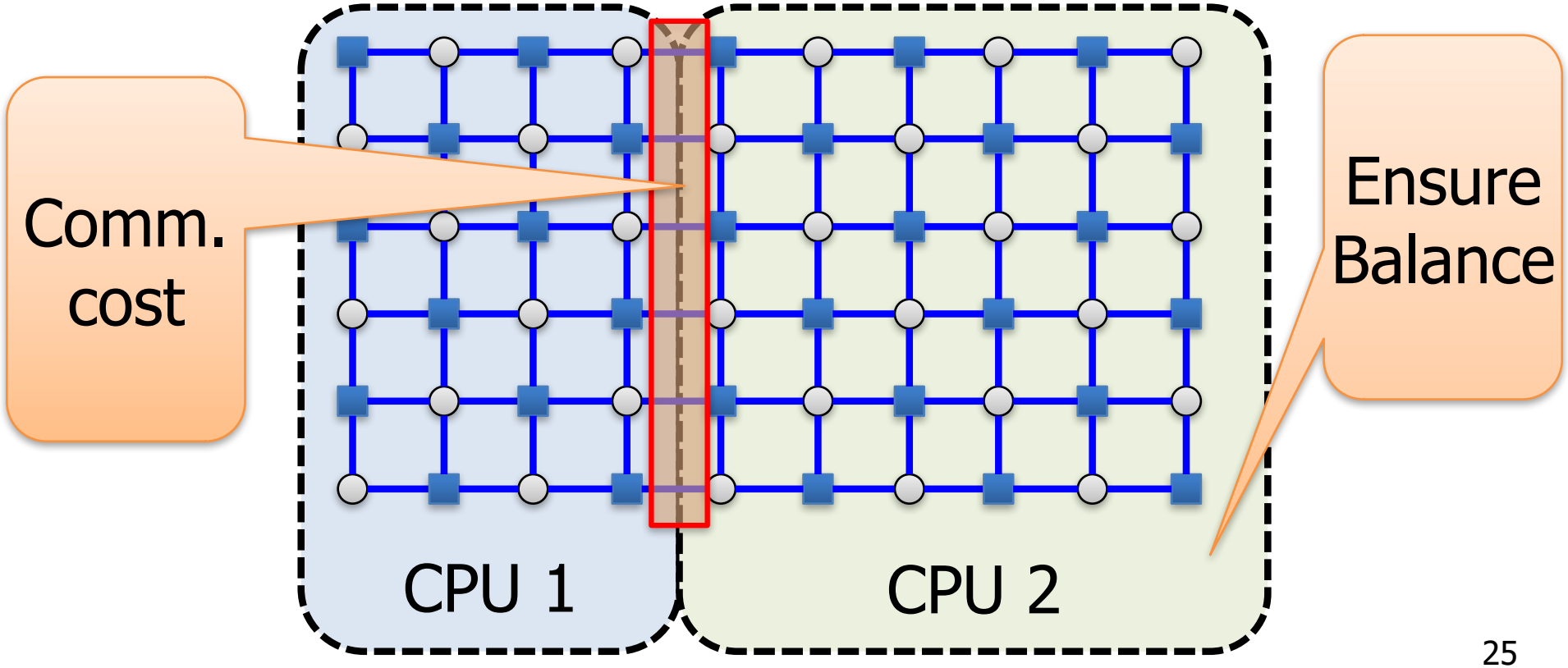
Given a uniform partitioning of the chain graphical model, DBRSplash will run in time:

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

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:

Minimize Communication

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

Ensure Balance

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

- Depends on:

Update counts are not known!

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

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

- NP-Hard → METIS fast partitioning heuristic

sense learn act

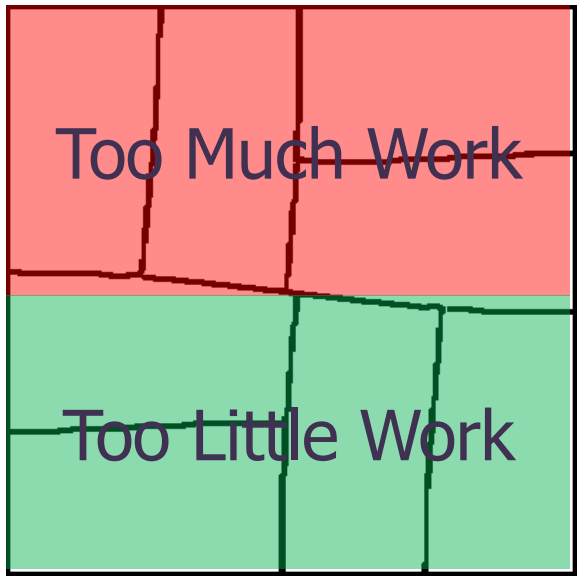
Unknown Update Counts

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

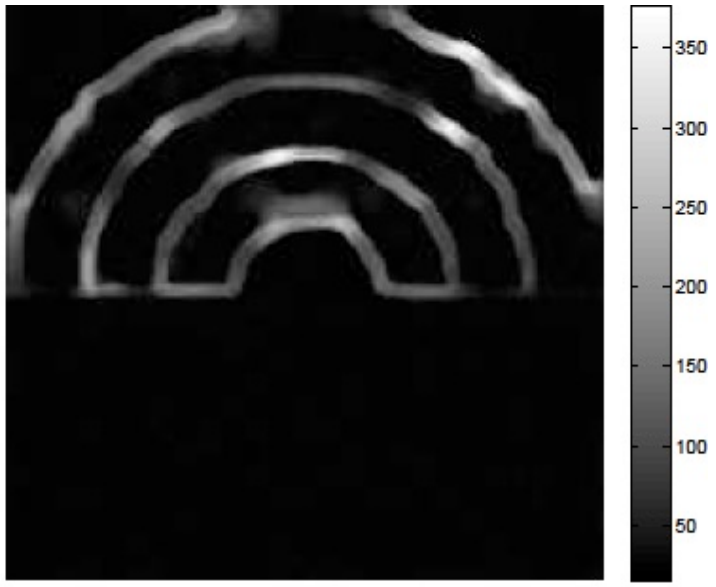
sense learn act

Uniformed Cuts

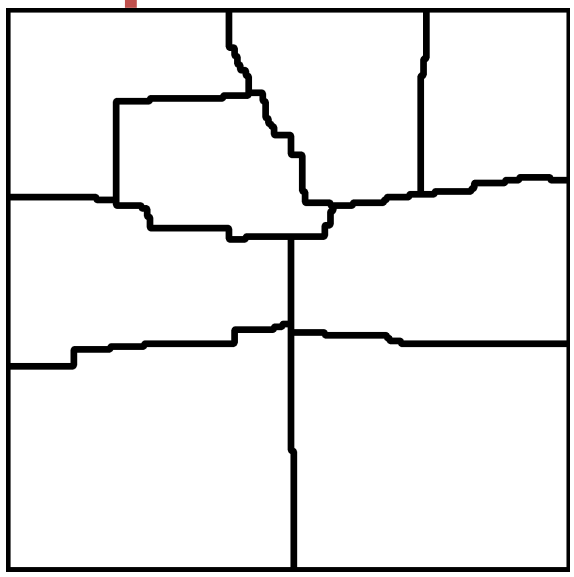
Uninformed Cut



Update Counts

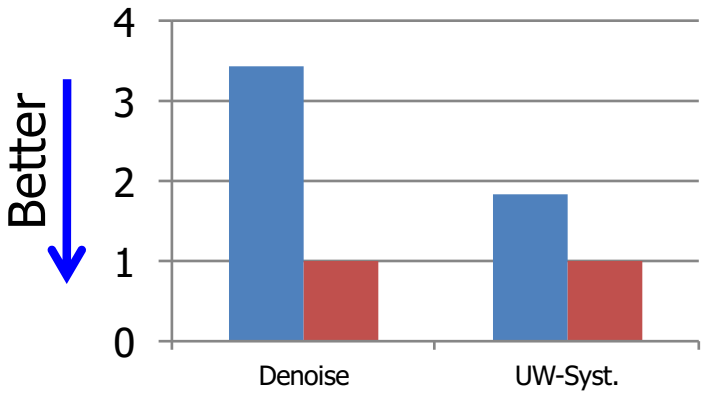


Optimal Cut

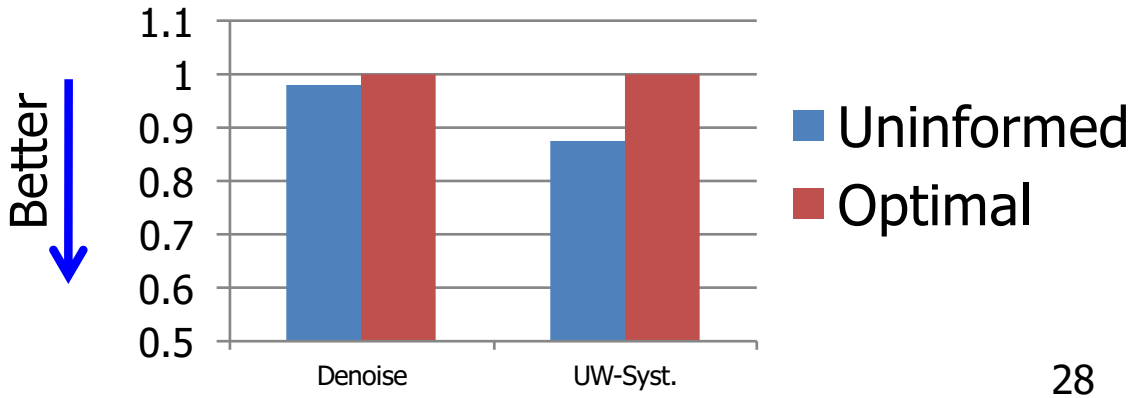


- Greater imbalance & lower communication cost

Work Imbalance

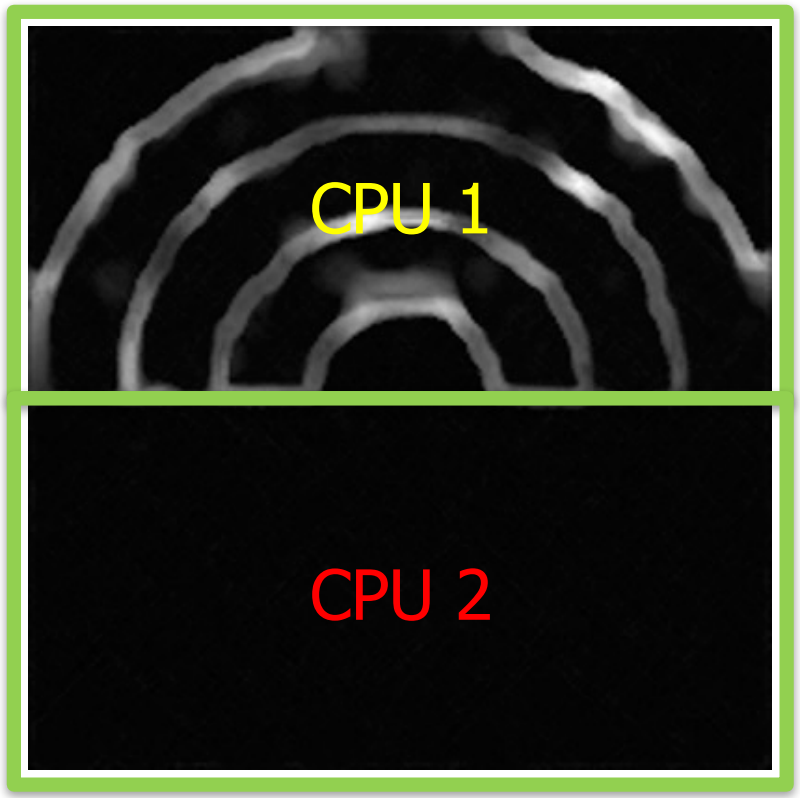


Communication Cost

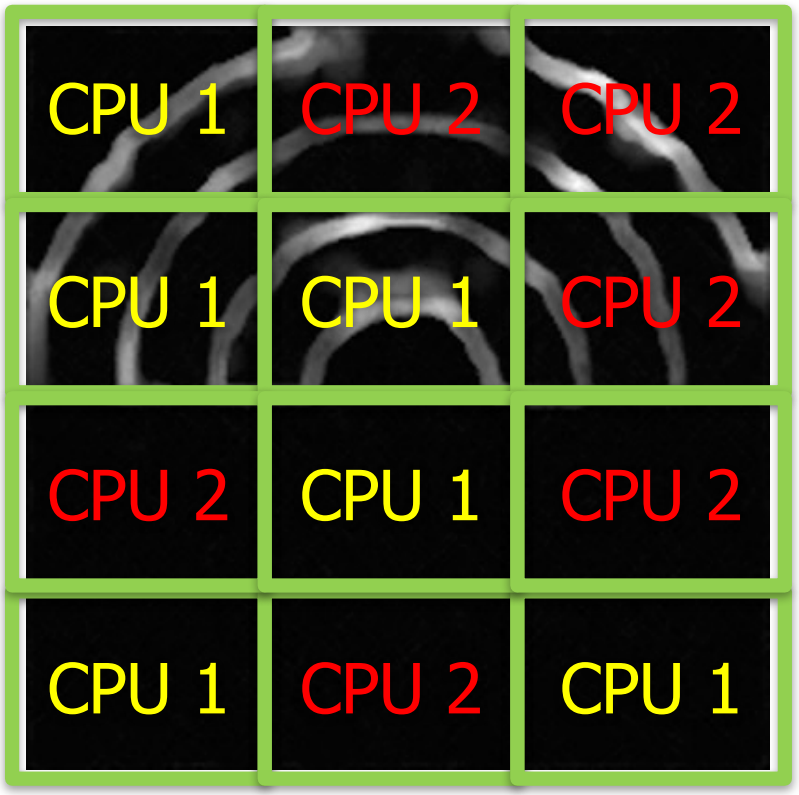


Over-Partitioning

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



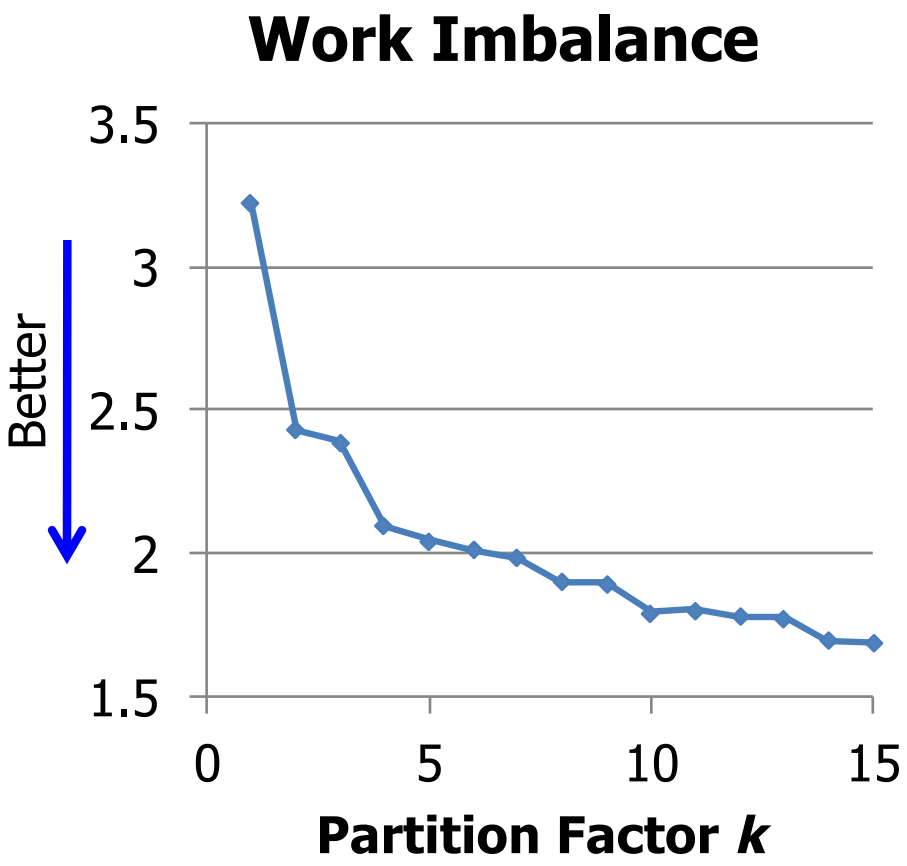
Without Over-Partitioning

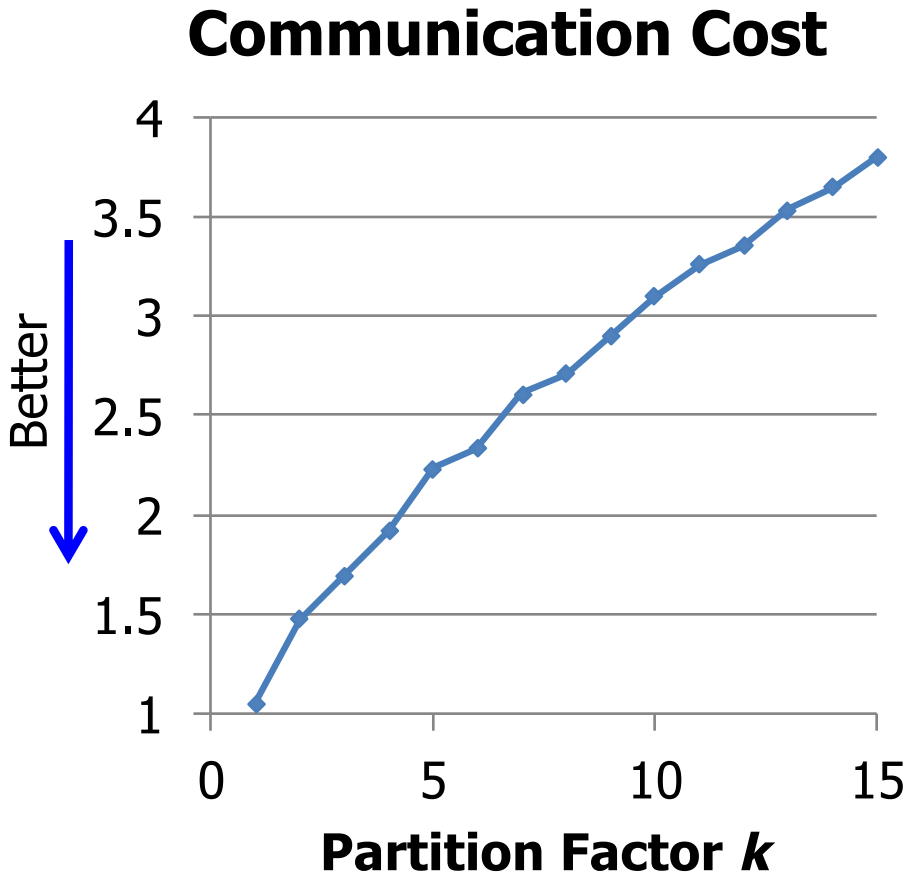


k=6

Over-Partitioning Results

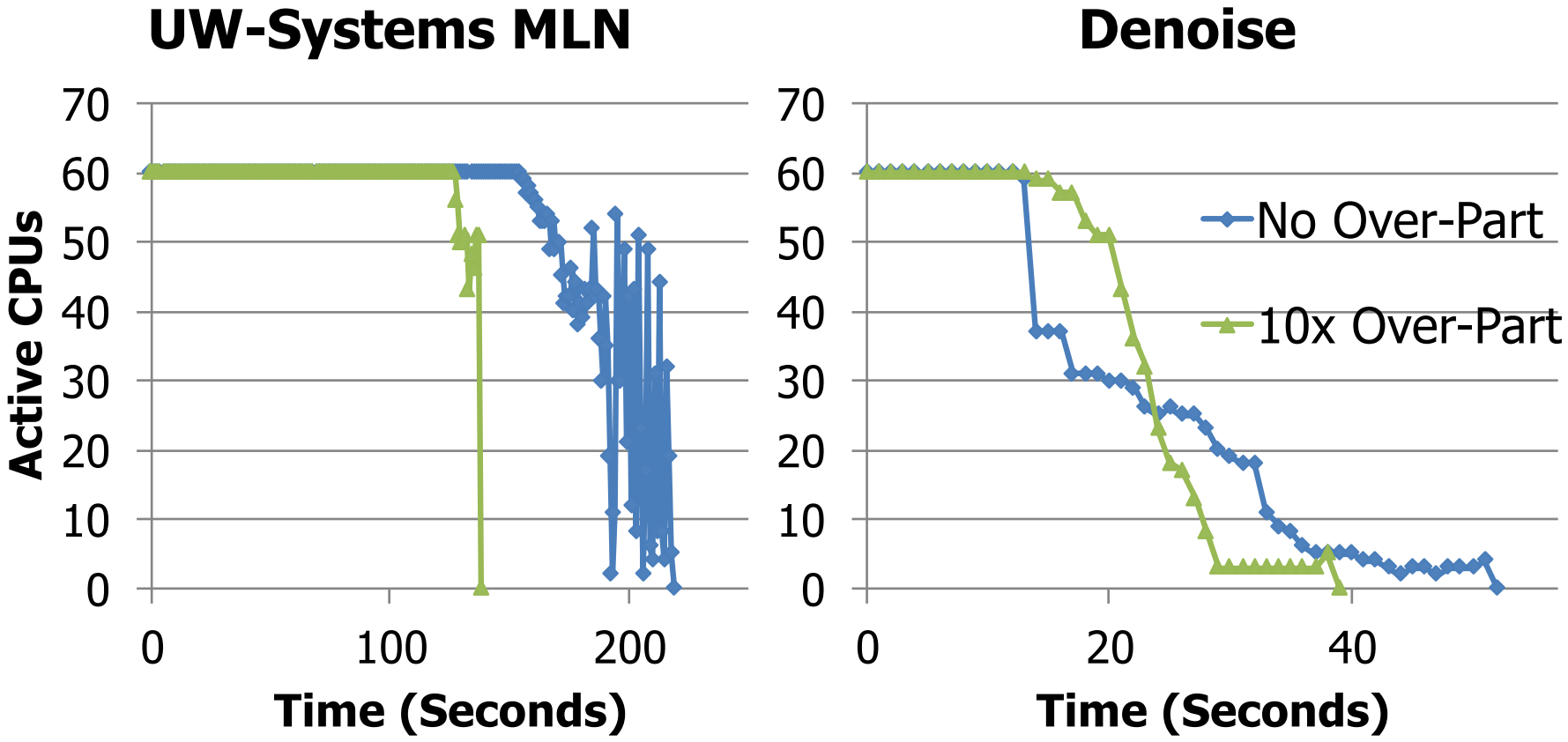
- Provides a simple method to trade between work balance and communication cost



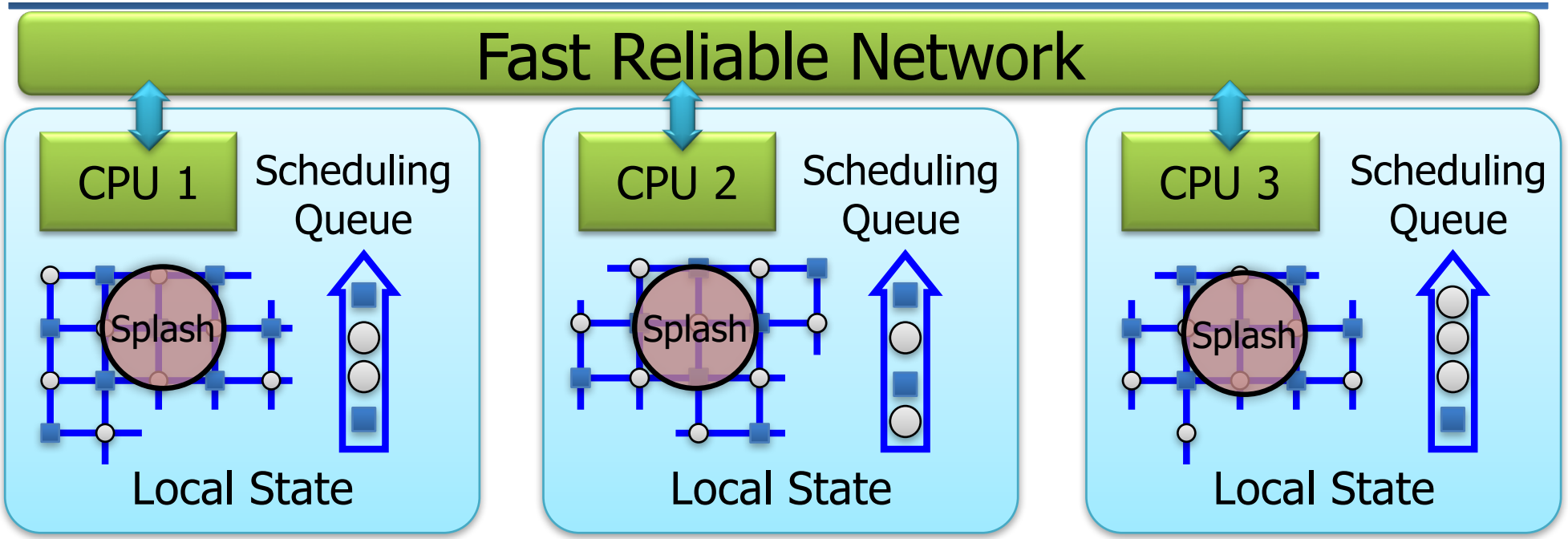


CPU Utilization

- Over-partitioning improves CPU utilization:



DBRSplash Algorithm



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

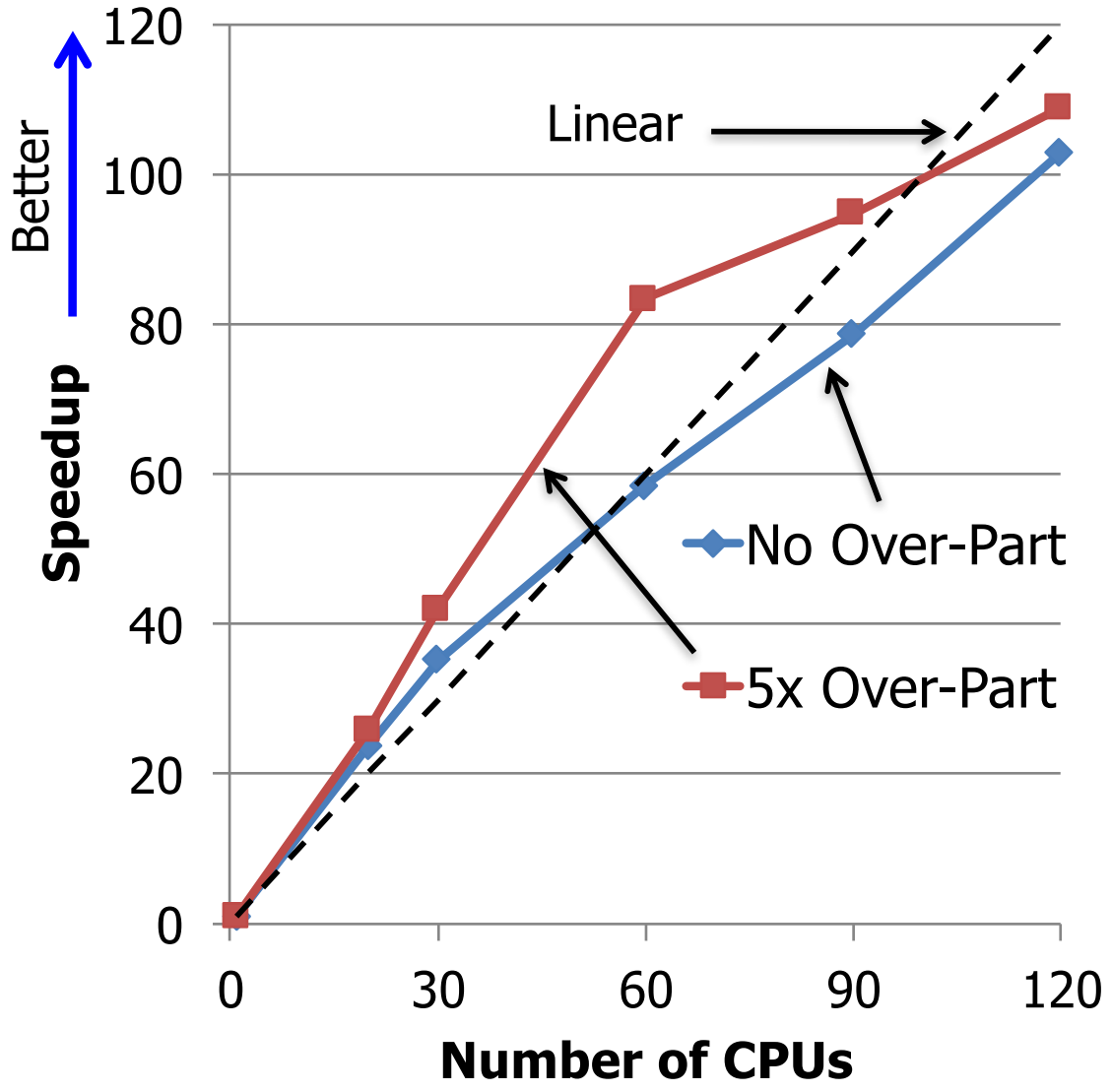
sense learn act

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

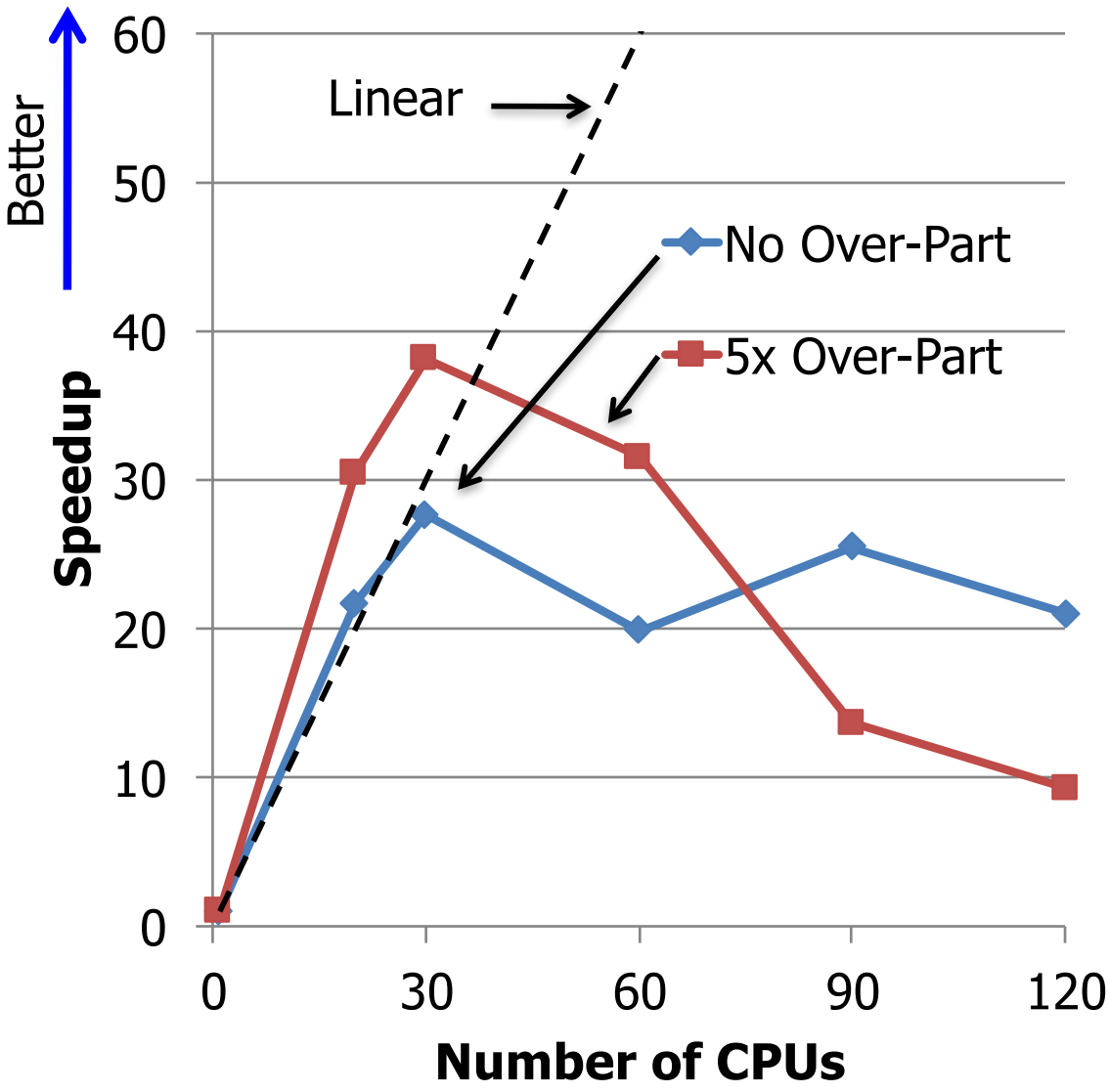
Parallel Performance (Large Graph)

- 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



sense learn act

Summary

- **Splash Operation** generalization of the optimal parallel schedule on chain graphs
- **Belief**-based scheduling
  - Addresses message scheduling issues
  - Improves accuracy and convergence
- **DBRSplash** an efficient **distributed** parallel inference algorithm
  - Over-partitioning to improve work balance
- Experimental results on large factor graphs:
  - **Linear** to **super-linear** speed-up using up to 120 processors

Thank You

Acknowledgements

Intel Research Pittsburgh: OpenCirrus Cluster

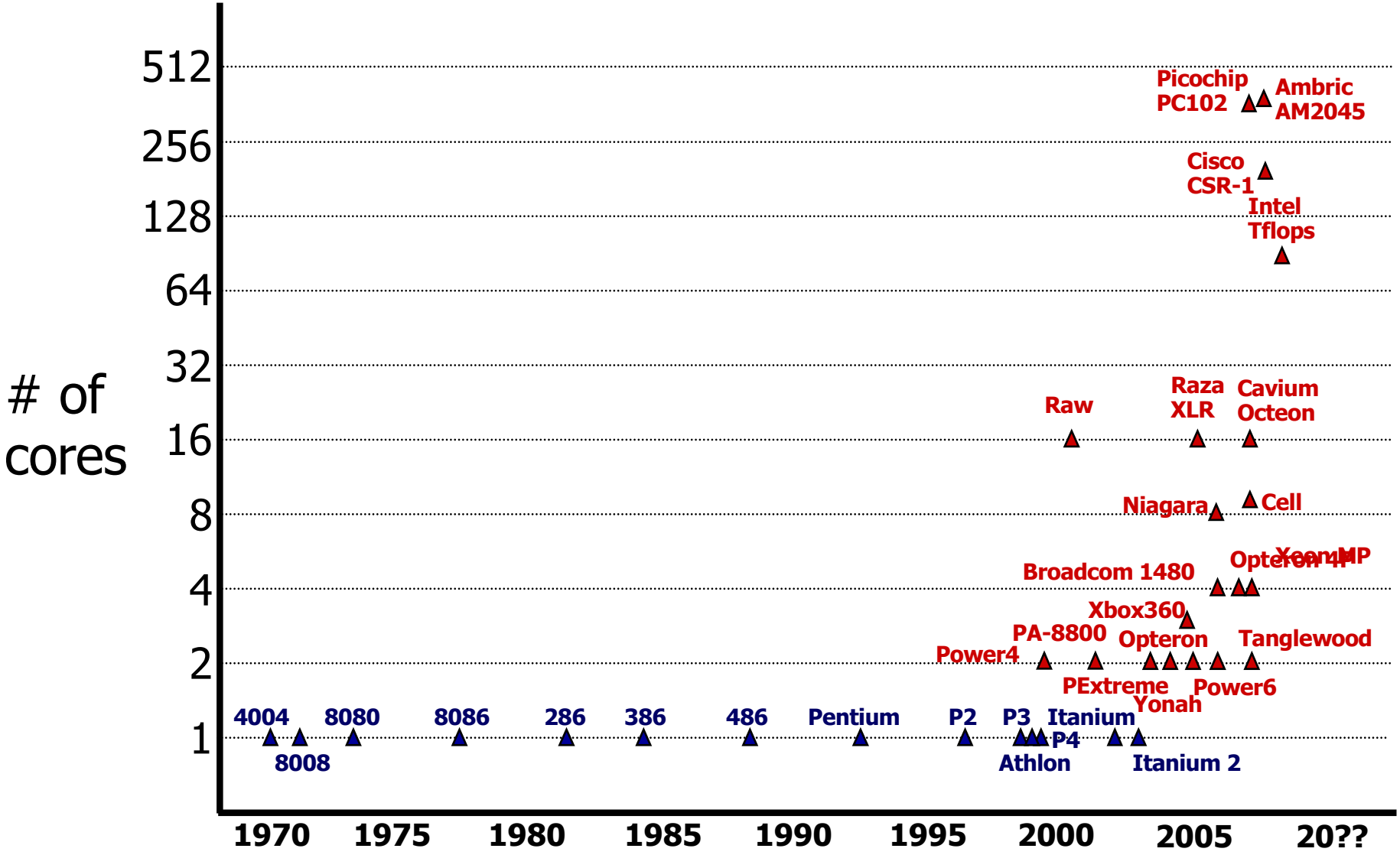
AT&T Labs

DARPA

sense learn act

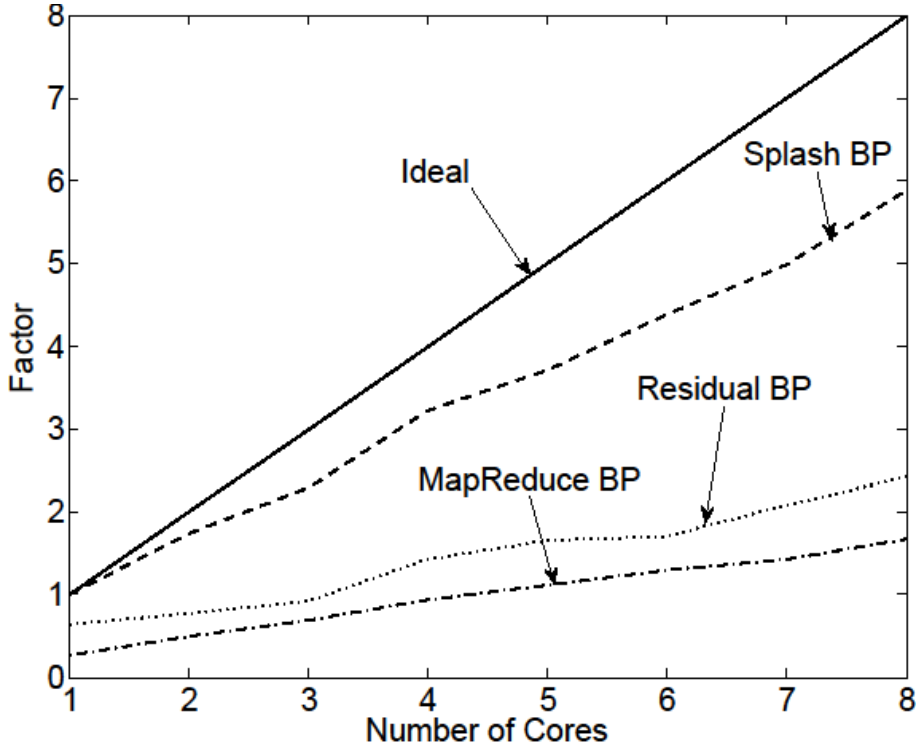
Exponential Parallelism

- From Saman Amarasinghe:



AIStats09 Speedup

3D –Video Prediction



Protein Side-Chain Prediction