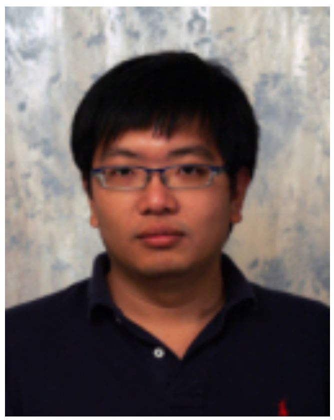
Parallel Gibbs Sampling

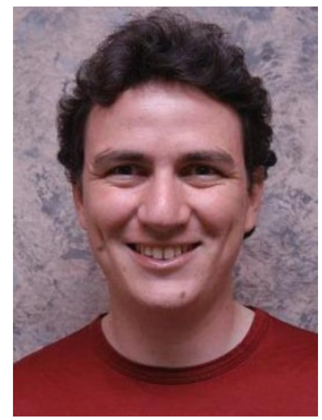
From Colored Fields to Thin Junction Trees

Joseph Gonzalez



Yucheng Low

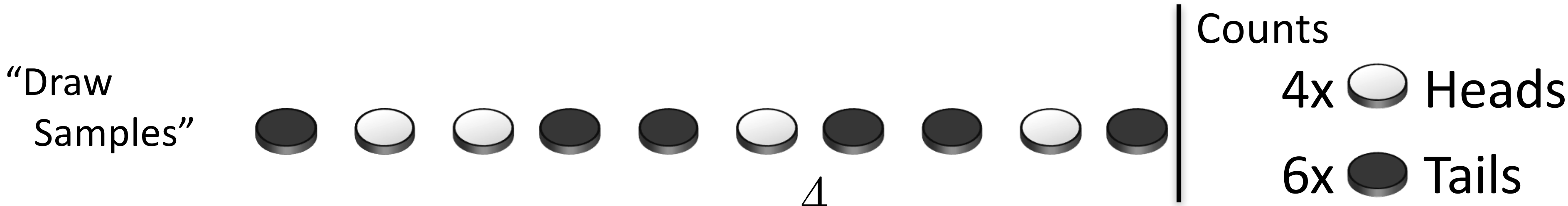
Arthur Gretton



Carlos Guestrin

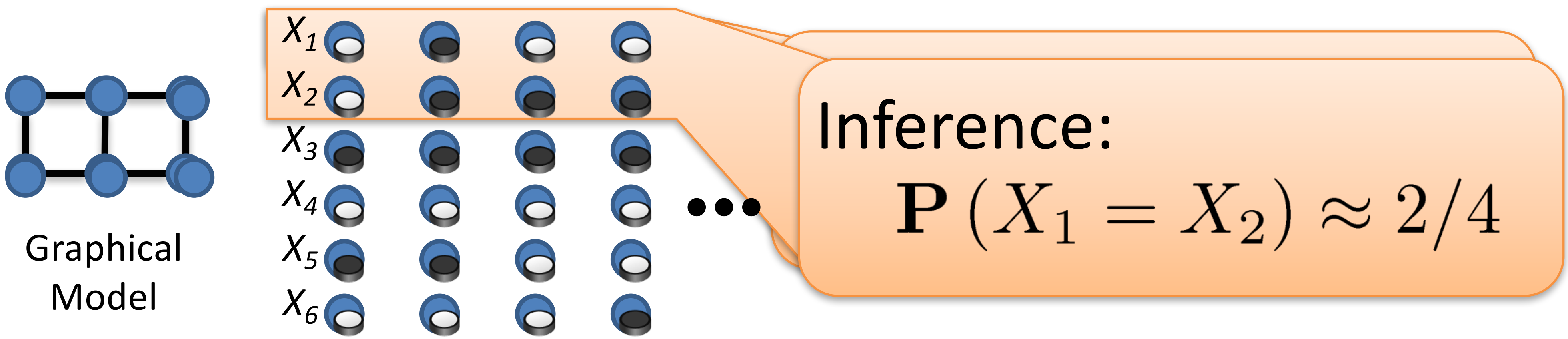
Sampling as an Inference Procedure

- Suppose we wanted to know the probability that coin lands "heads"



$$\mathbf{P}\left(\text{Heads}\right) \approx \frac{4}{10}$$

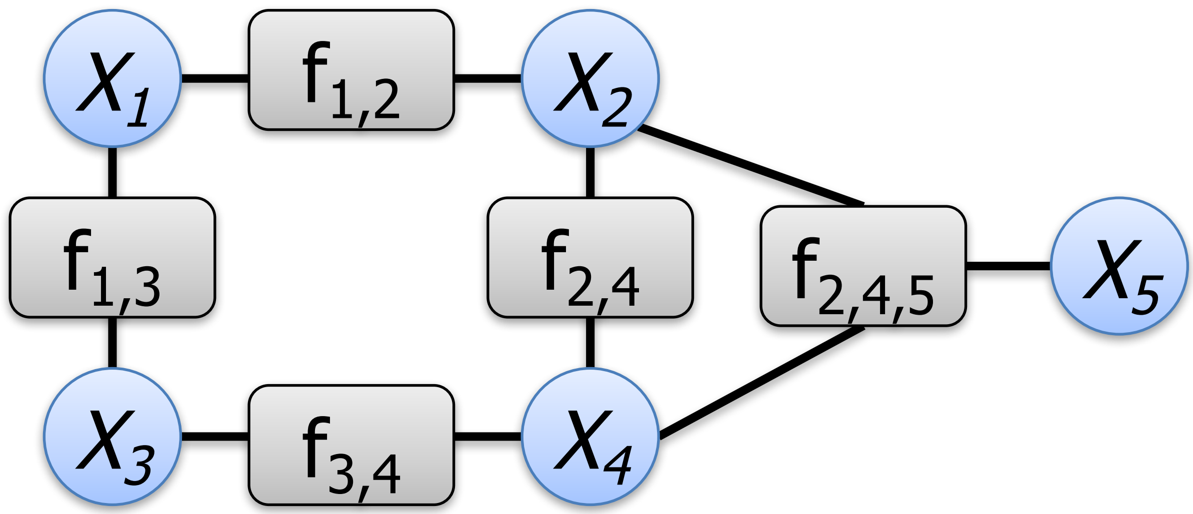
- We use the same idea for graphical model inference



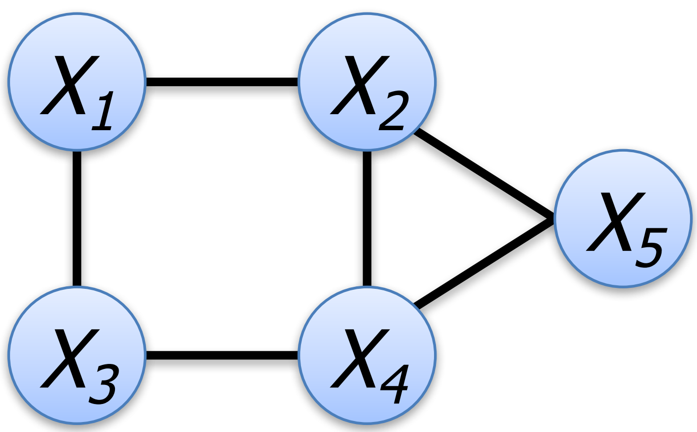
Terminology: Graphical Models

- Focus on **discrete** factorized models with **sparse structure**:

Factor Graph



Markov Random Field



Terminology: Ergodicity

- The goal is to estimate:

$$\mathbf{E}\left[h(X_1, \ldots, X_n)\right]$$

  - Example: marginal estimation

$$h_i(x) = \mathbf{I}[x == i] \Rightarrow \mathbf{E}[h_i(X_k)] = \mathbf{P}(X_k = i)$$

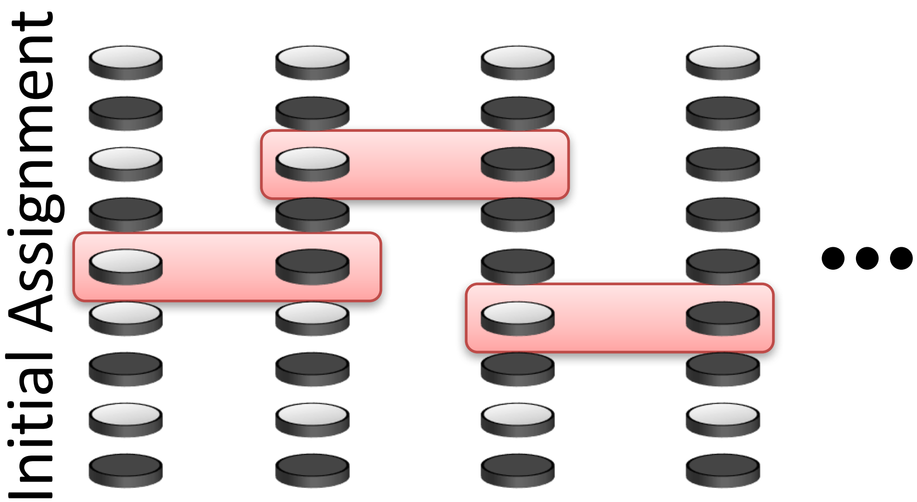
- If the sampler is **ergodic** the following is true*:

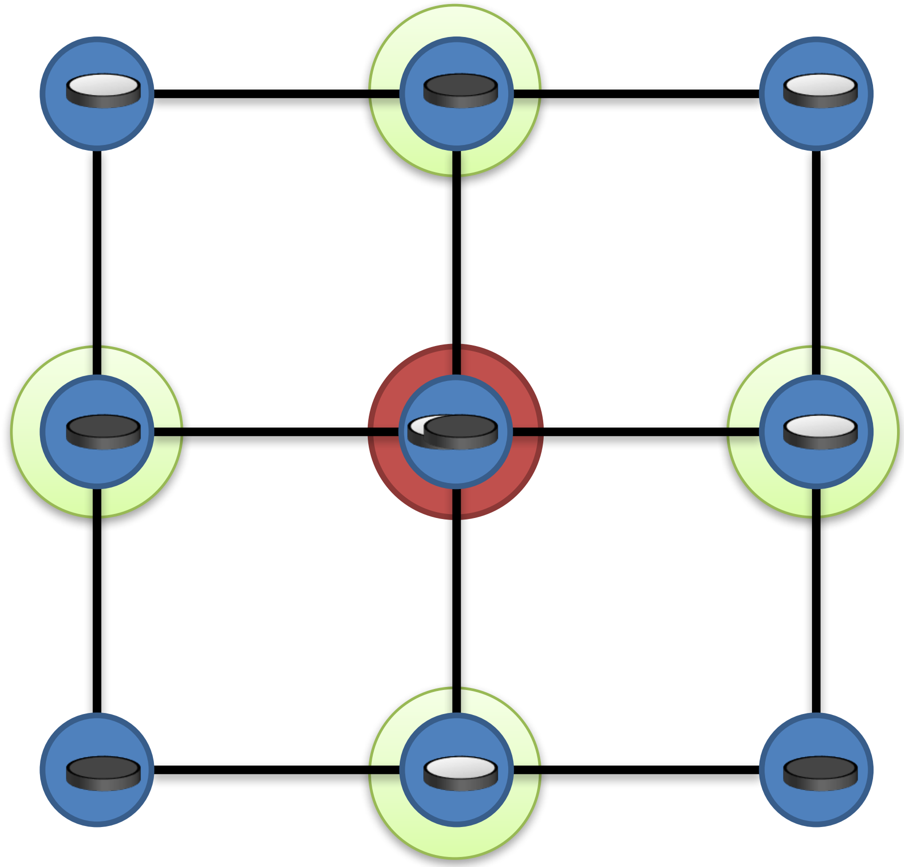
$$\lim_{m \to \infty} \frac{1}{m} \sum_{t=1}^{m} h(x_1^{(t)}, \ldots, x_n^{(t)}) \xrightarrow{a.s.} \mathbf{E}\left[h(X_1, \ldots, X_n)\right]$$

*Consult your statistician about potential risks before using.

Gibbs Sampling [Geman & Geman, 1984]

- **Sequentially** for each variable in the model
  - Select variable
  - Construct conditional given adjacent assignments
  - Flip coin and update assignment to variable





Why Study Parallel Gibbs Sampling?

"The Gibbs sampler ... might be considered the workhorse of the MCMC world."

–Robert and Casella

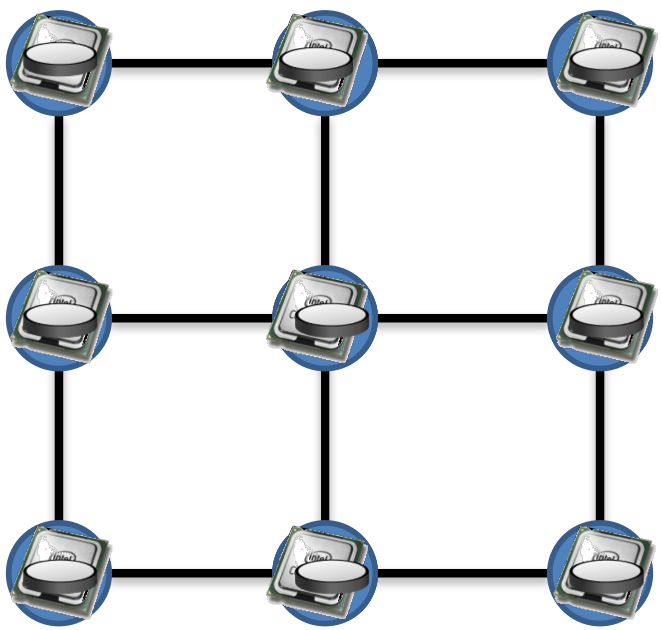
- Ergodic with geometric convergence
- Great for high-dimensional models
  - No need to tune a joint proposal
- Easy to construct algorithmically
  - WinBUGS
- Important Properties that help Parallelization:
  - **Sparse structure → factorized** computation

Is the Gibbs Sampler trivially parallel?

From the original paper on Gibbs Sampling:

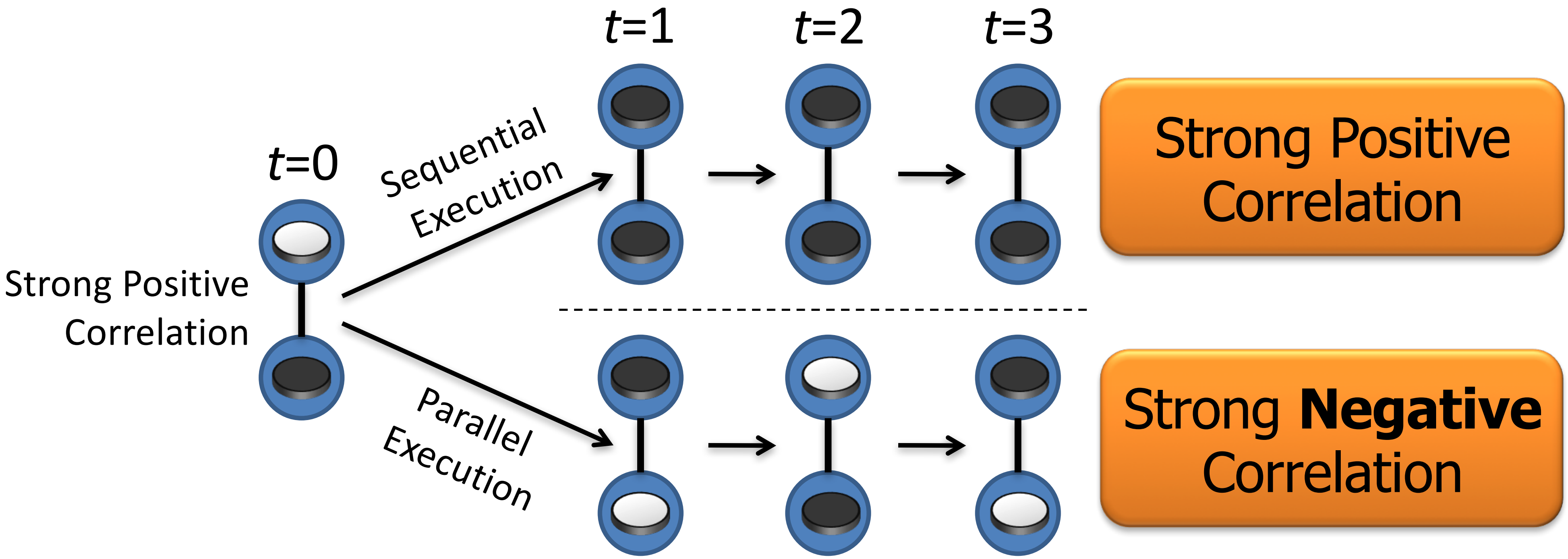
"…the MRF can be divided into collections of [variables] with each collection assigned to an ***independently*** *running* ***asynchronous processor****."*

-- Stuart and Donald Geman, 1984.



Converges to the **wrong** distribution!

The problem with **Synchronous** Gibbs



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

How has the machine learning community solved this problem?

Two Decades later

1. Newman et al., *Scalable Parallel Topic Models.* Jnl. Intelligen. Comm. R&D, 2006.
2. Newman et al., *Distributed Inference for Latent Dirichlet Allocation.* NIPS, 2007.
3. Asuncion et al., *Asynchronous Distributed Learning of Topic Models.* NIPS, 2008.
4. Doshi-Velez et al., *Large Scale Nonparametric Bayesian Inference: Data Parallelization in the Indian Buffet Process.* NIPS 2009
5. Yan et al., *Parallel Inference for Latent Dirichlet Allocation on GPUs.* NIPS, 2009.

- Same problem as the original Geman paper
  - Parallel version of the sampler is **not** *ergodic*.
- Unlike Geman, the recent work:
  - Recognizes the issue
  - Ignores the issue
  - Propose an “approximate” solution

Two Decades Ago

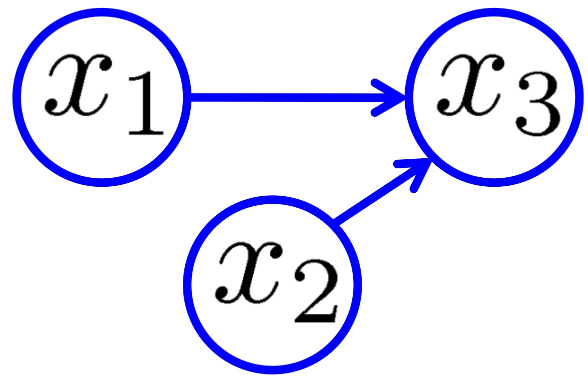
- Parallel computing community studied:

Sequential Algorithm

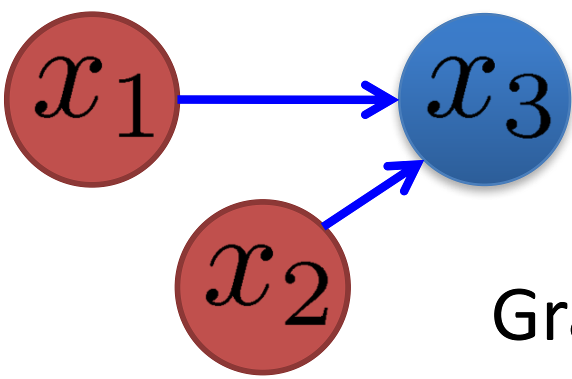
Time

$$\begin{aligned} x_1^{(t+1)} &= f_1(x_3^{(t)}) \\ x_2^{(t+1)} &= f_2(x_2^{(t)}, x_3^{(t)}) \\ x_3^{(t+1)} &= f_3(x_1^{(t+1)}, x_1^{(t+1)}) \end{aligned}$$

Directed Acyclic Dependency Graph



Construct an **Equivalent** Parallel Algorithm



Using Graph Coloring

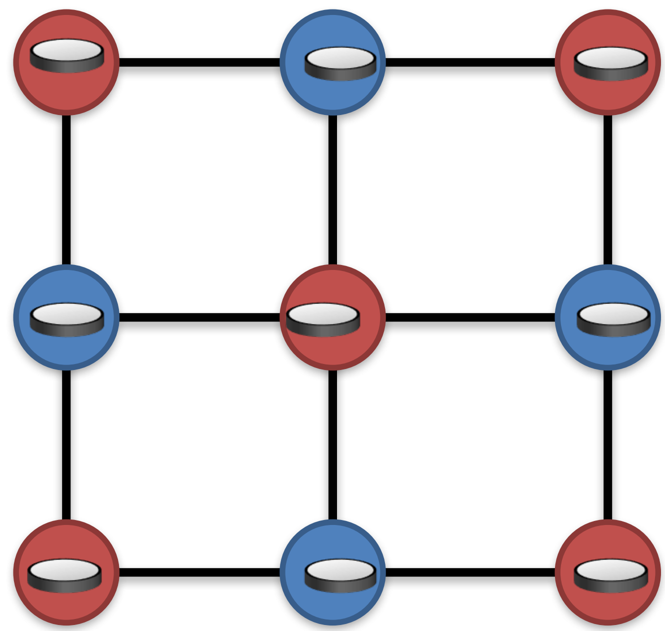
$$x_1^{(t+1)} = f_1(x_3^{(t)}) \quad x_2^{(t+1)} = f_2(x_2^{(t)}, x_3^{(t)})$$

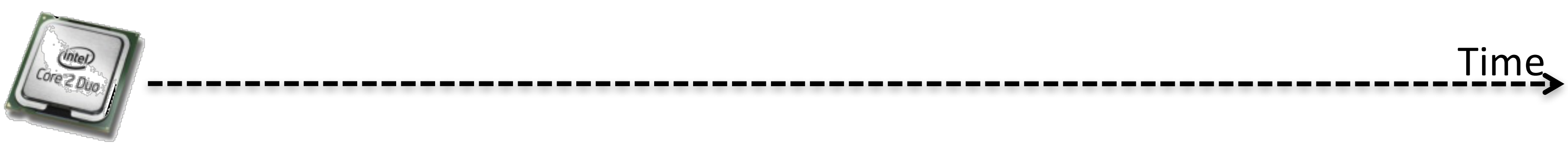
$$x_3^{(t+1)} = f_3(x_1^{(t+1)}, x_1^{(t+1)})$$

Chromatic Sampler

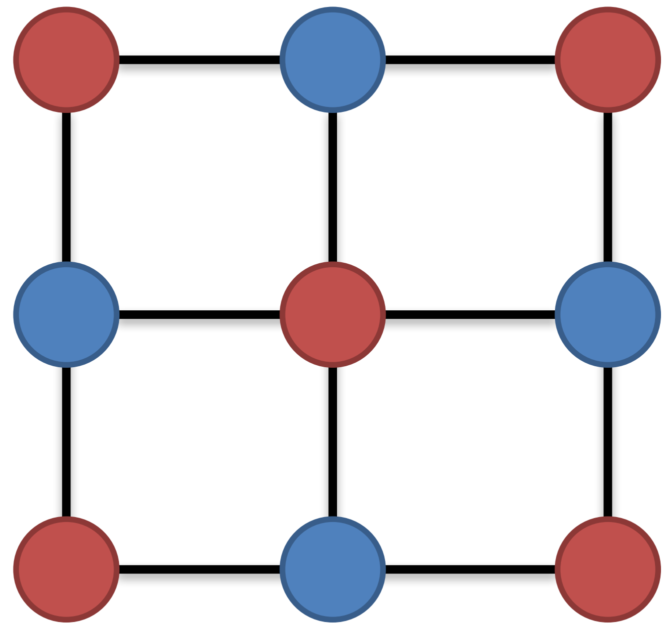
- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel

- Sequential Consistency:





Chromatic Sampler Algorithm



For t from 1 to T do

$$x^{(t)} \leftarrow x^{(t-1)}$$

For k from 1 to K do

Parfor i in color k:

$$x_i^{(t)} \sim \mathbf{P}(X_i \mid x_{-i}^{(t)})$$

Asymptotic Properties

- **Quantifiable** acceleration in **mixing**

Time to update all variables once

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

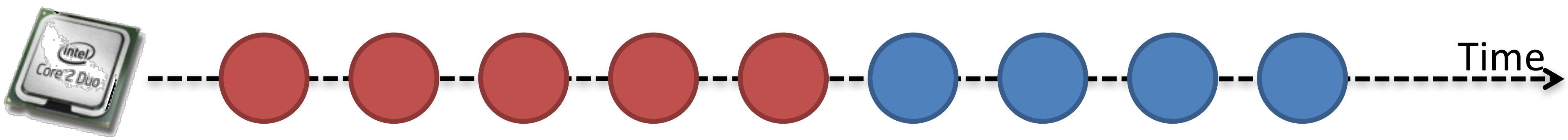
Variables
Colors
Processors

- **Speedup:**

$$O\left(p\left(\underbrace{\frac{n}{n+pk}}_{\text{Penalty Term}}\right)\right)$$

Proof of Ergodicity

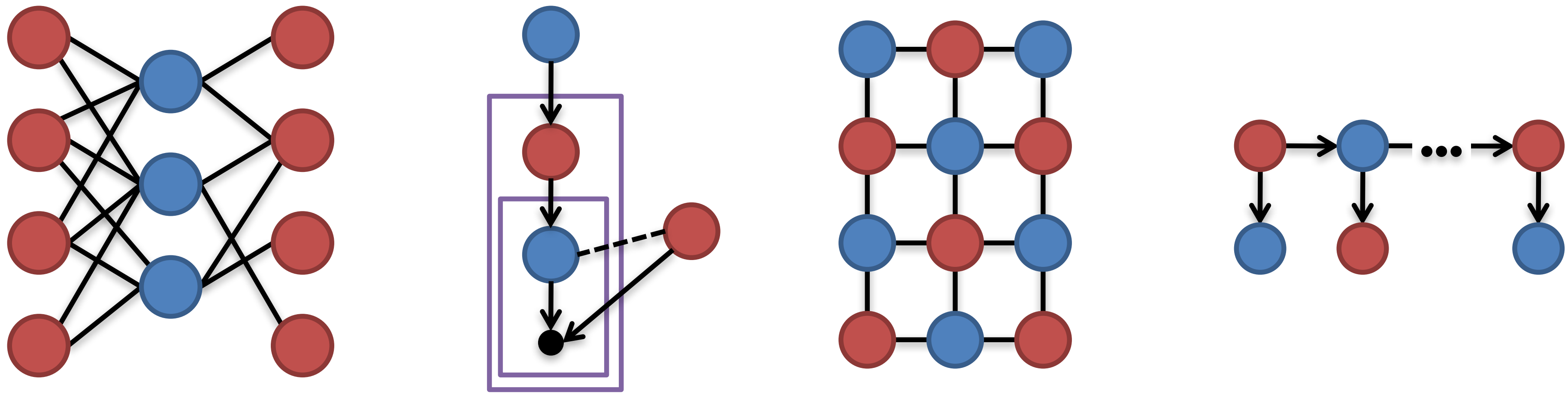
- Version 1 (Sequential Consistency):
  - **Chromatic Gibbs Sampler** is *equivalent* to a ***Sequential Scan*** Gibbs Sampler



- Version 2 (Probabilistic Interpretation):
  - Variables in same color are **Conditionally Independent** → *Joint Sample* is *equivalent* to **Parallel Independent Samples**

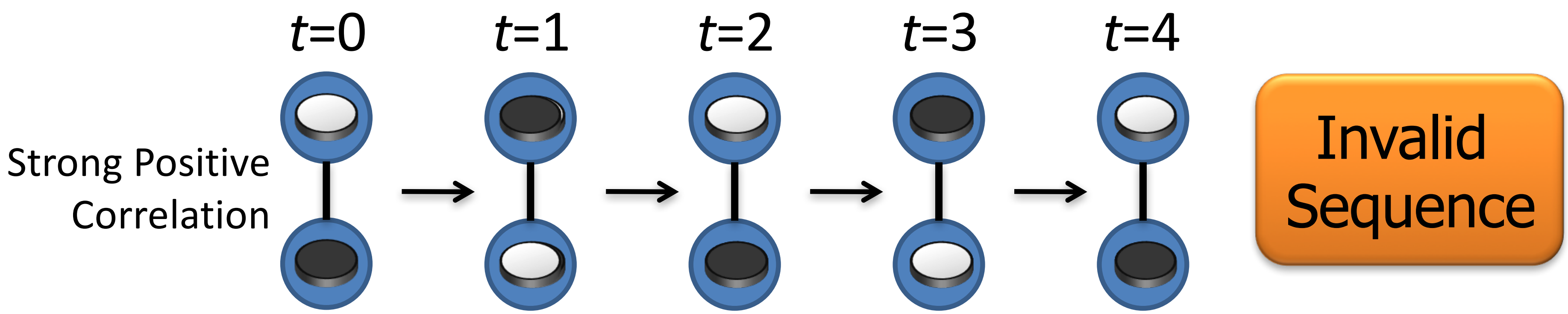
Special Properties of 2-Colorable Models

- Many common models have two colorings

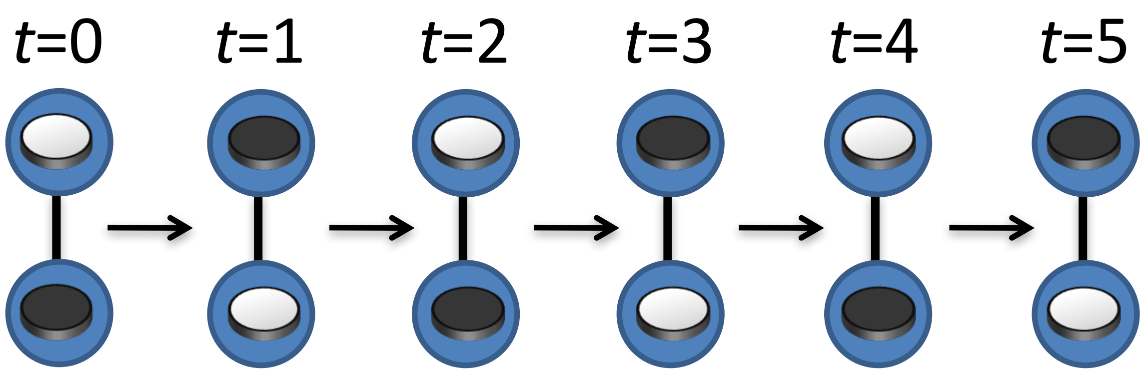


- For the [Incorrect] Synchronous Gibbs Samplers
  - Provide a method to correct the chains
  - Derive the stationary distribution

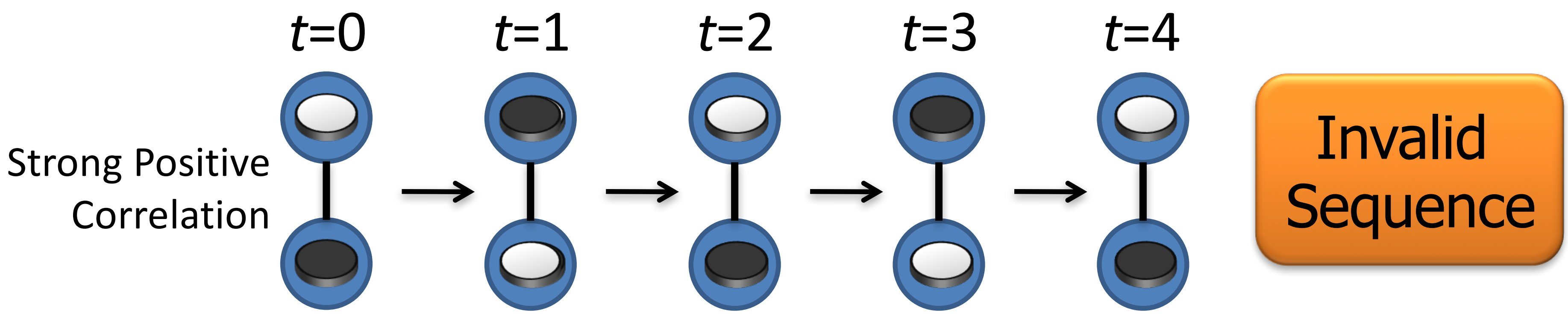
Correcting the *Synchronous* Gibbs Sampler



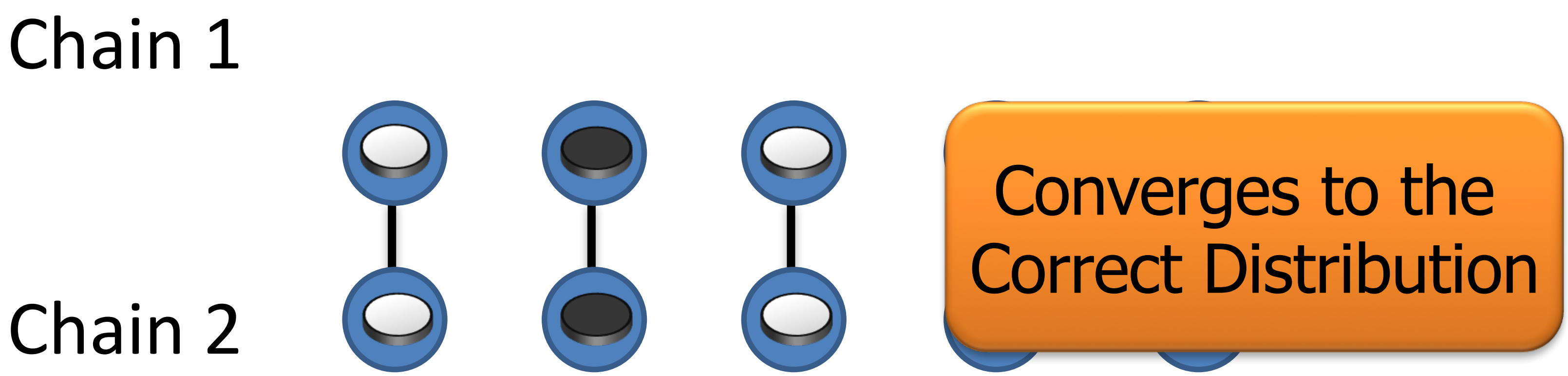
- We can derive two **valid** chains:



Correcting the *Synchronous* Gibbs Sampler



- We can derive two **valid** chains:



Theoretical Contributions on 2-colorable models

- Stationary distribution of **Synchronous Gibbs**:

$$\begin{aligned}
\mathbf{P}(x') &= \sum_{x} K(x' \mid x)\pi(x_{\kappa_1})\pi(x_{\kappa_2}) \\
&= \sum_{x_{\kappa_1}} \sum_{x_{\kappa_2}} \pi(x'_{\kappa_1} \mid x_{\kappa_2})\pi(x'_{\kappa_2} \mid x_{\kappa_1})\pi(x_{\kappa_1})\pi(x_{\kappa_2}) \\
&= \sum_{x_{\kappa_1}} \sum_{x_{\kappa_2}} \pi(x_{\kappa_1}, x'_{\kappa_2})\pi(x'_{\kappa_1}, x_{\kappa_2}) \\
&= \pi(x'_{\kappa_1})\pi(x'_{\kappa_2})
\end{aligned}$$

Variables in Color 1

Variables in Color 2

Theoretical Contributions on 2-colorable models

- Stationary distribution of **Synchronous Gibbs**

$$\mathbf{P}_{\text{sync}}\left(X_1, \ldots, X_n\right) = \pi\left(X_{\kappa_1}\right)\pi\left(X_{\kappa_2}\right)$$

Variables in Color 1

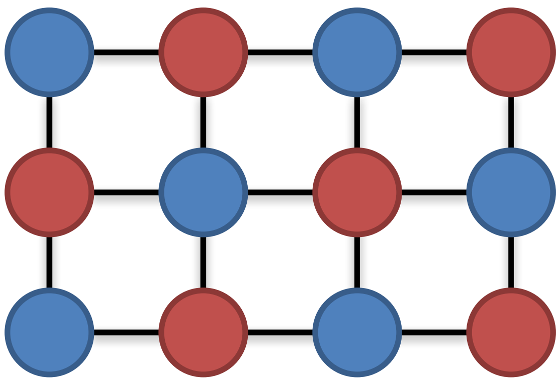
Variables in Color 2

- **Corollary**: Synchronous Gibbs sampler is **correct** for single variable marginals.

$$\lim_{m \to \infty} \frac{1}{m} \sum_{t=1}^{m} h(x_i^{(t)}) \xrightarrow{a.s.} \mathbf{E}\left[h(X_i)\right]$$

From Colored Fields to Thin Junction Trees

Chromatic Gibbs Sampler



- Ideal for:
  - **Rapid mixing** models
  - Conditional structure does not admit **Splash**

Splash Gibbs Sampler

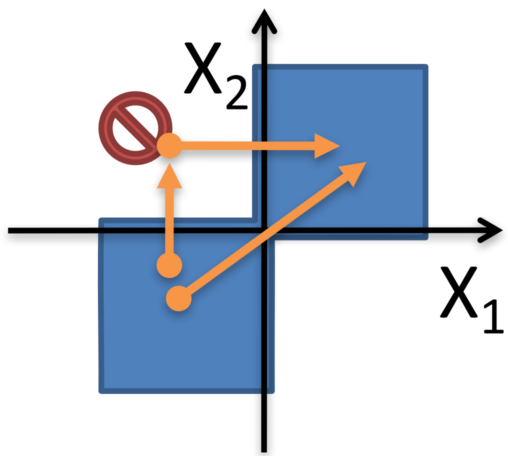
Slowly Mixing Models

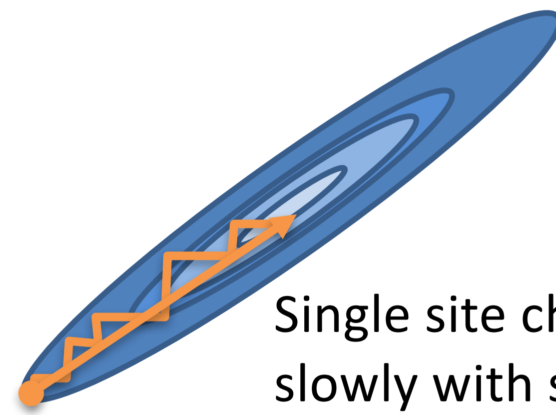
?

- Ideal for:
  - **Slowly mixing models**
  - Conditional structure admits Splash
    - Discrete models

Models With Strong Dependencies

- **Single variable** Gibbs updates tend to mix **slowly:**

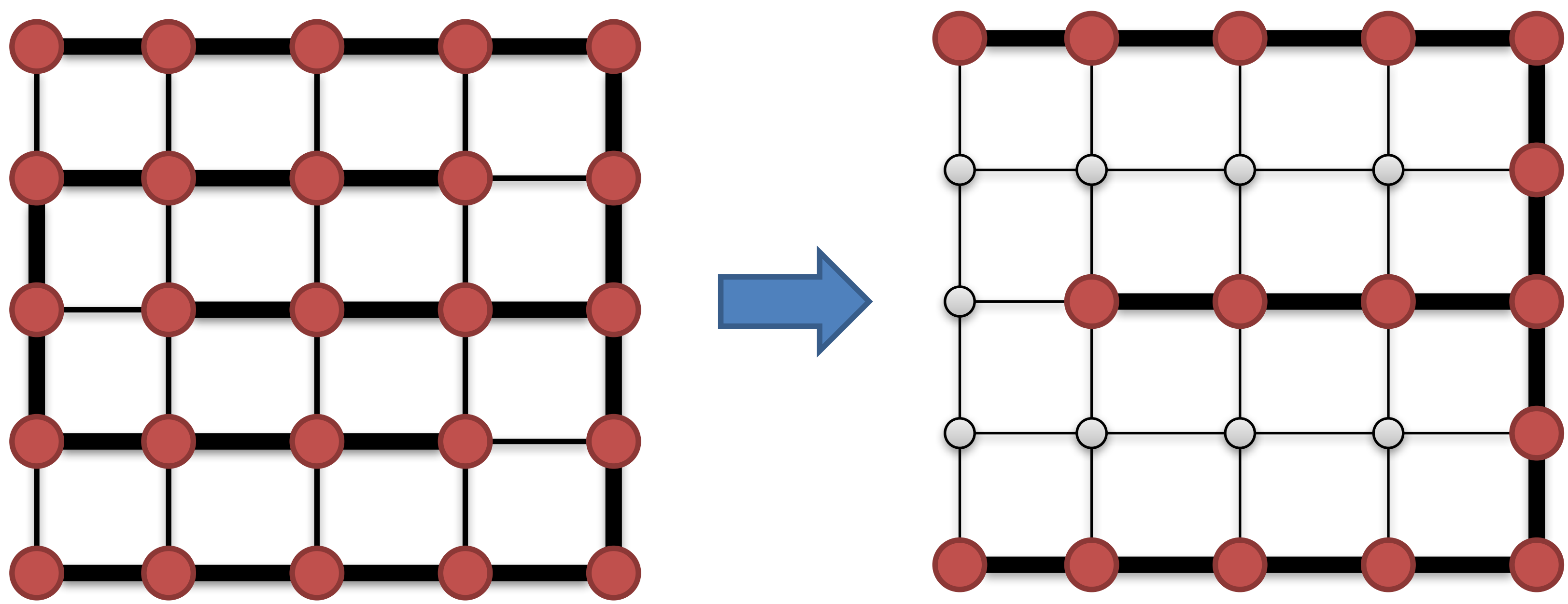




- Ideally we would like to draw joint samples.
  - Blocking

Blocking Gibbs Sampler

- Based on the papers:
  1. Jensen et al., *Blocking Gibbs Sampling for Linkage Analysis in Large Pedigrees with Many Loops.* **TR 1996**
  2. Hamze et al., *From Fields to Trees*. **UAI 2004**.

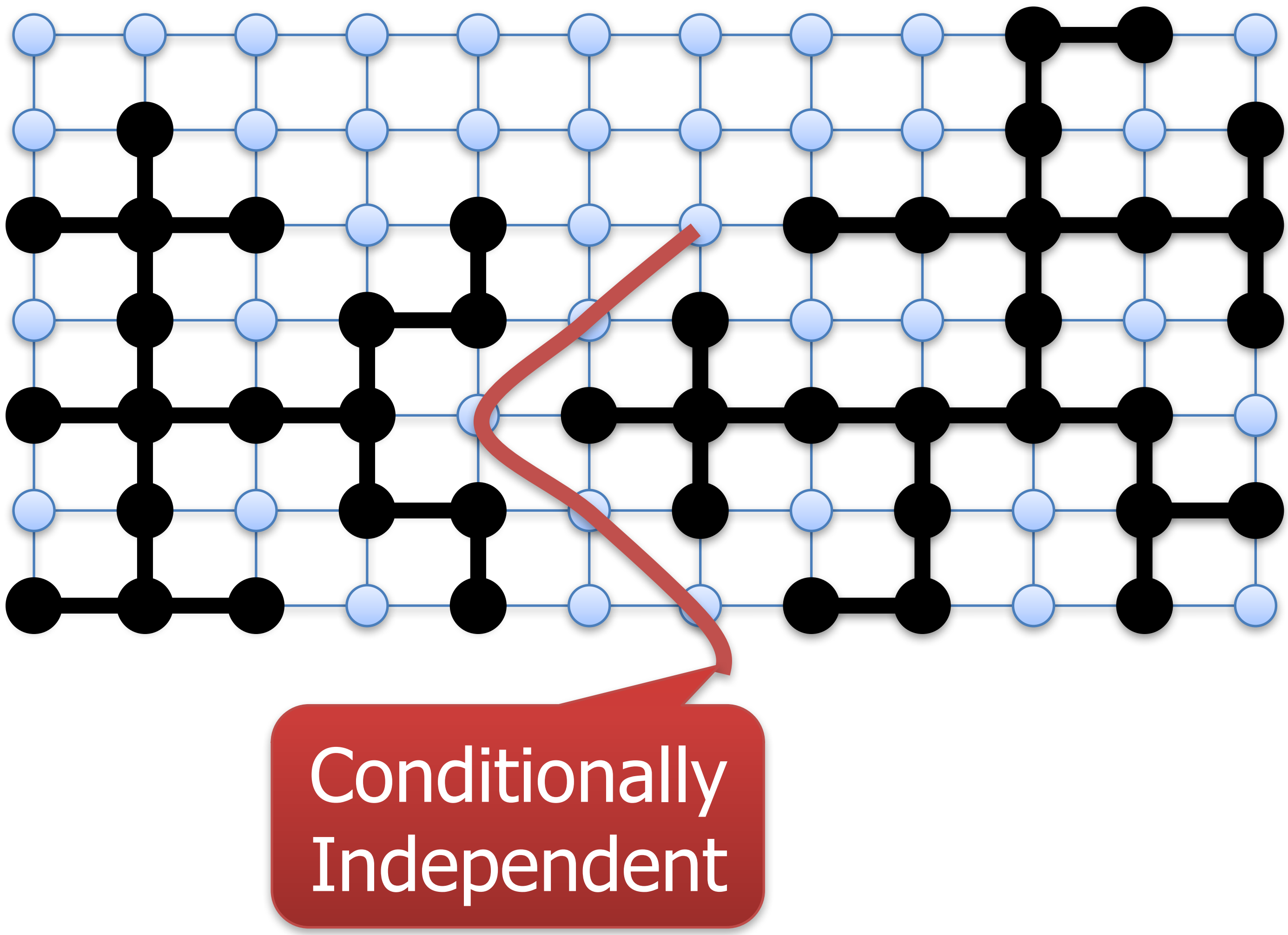


Splash Gibbs Sampler

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

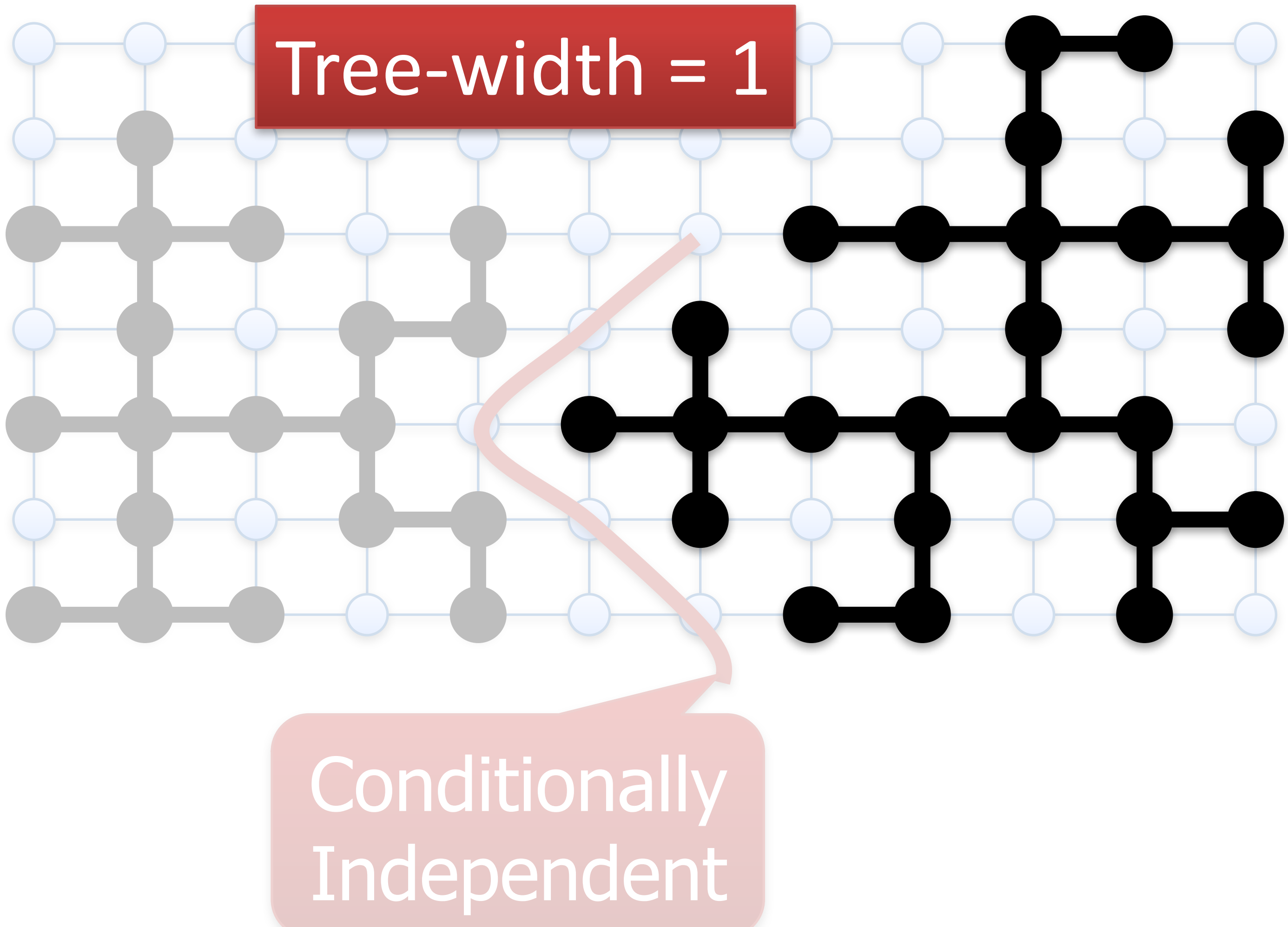
Splash Gibbs Sampler

- **Step 1:** Grow multiple Splashes in parallel:



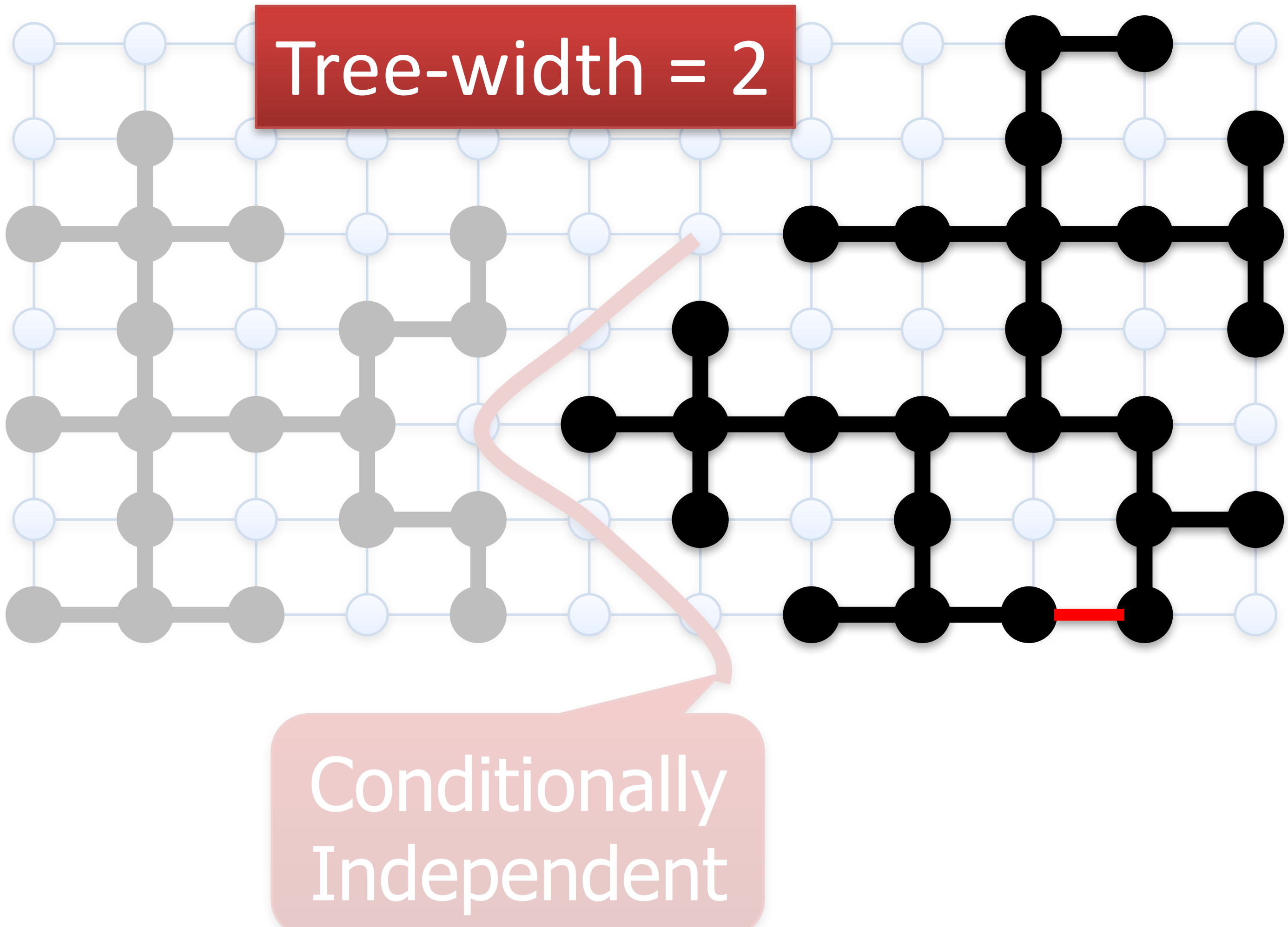
Splash Gibbs Sampler

- **Step 1:** Grow multiple Splashes in parallel:



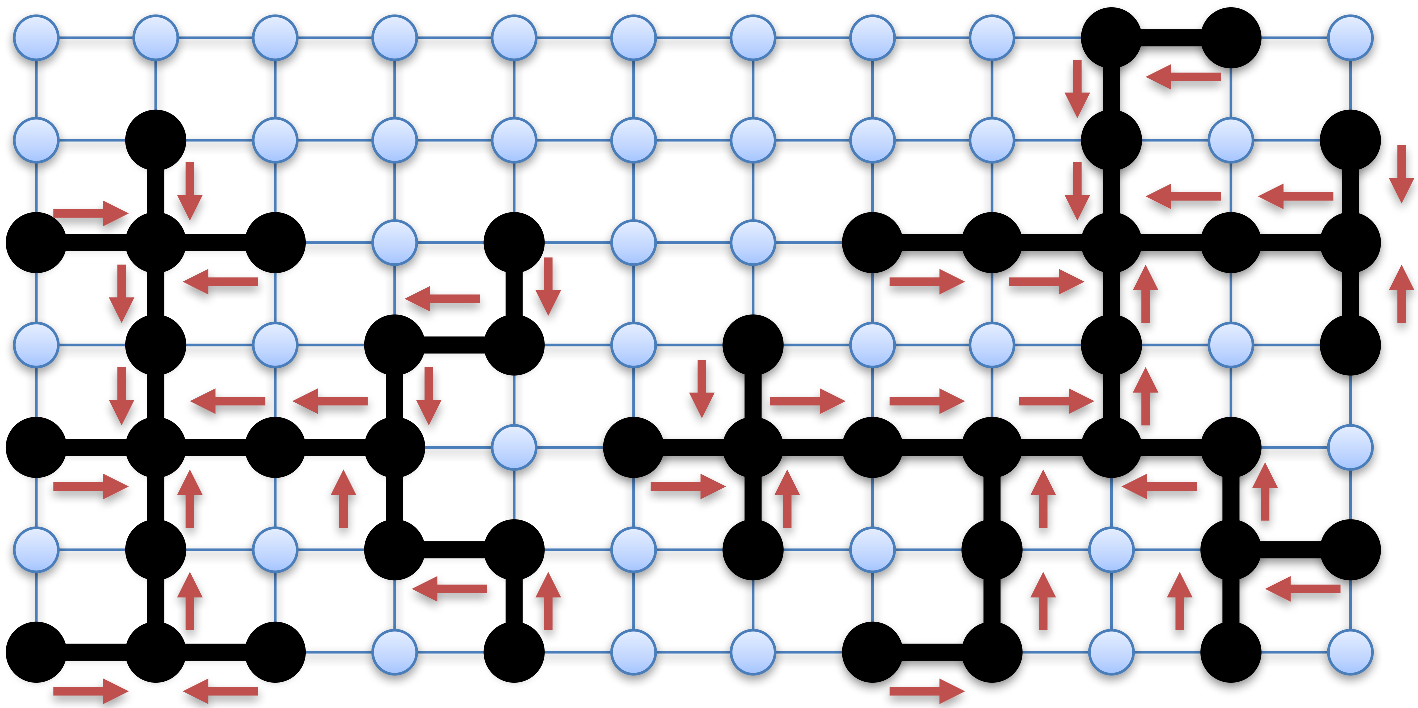
Splash Gibbs Sampler

- **Step 1:** Grow multiple Splashes in parallel:



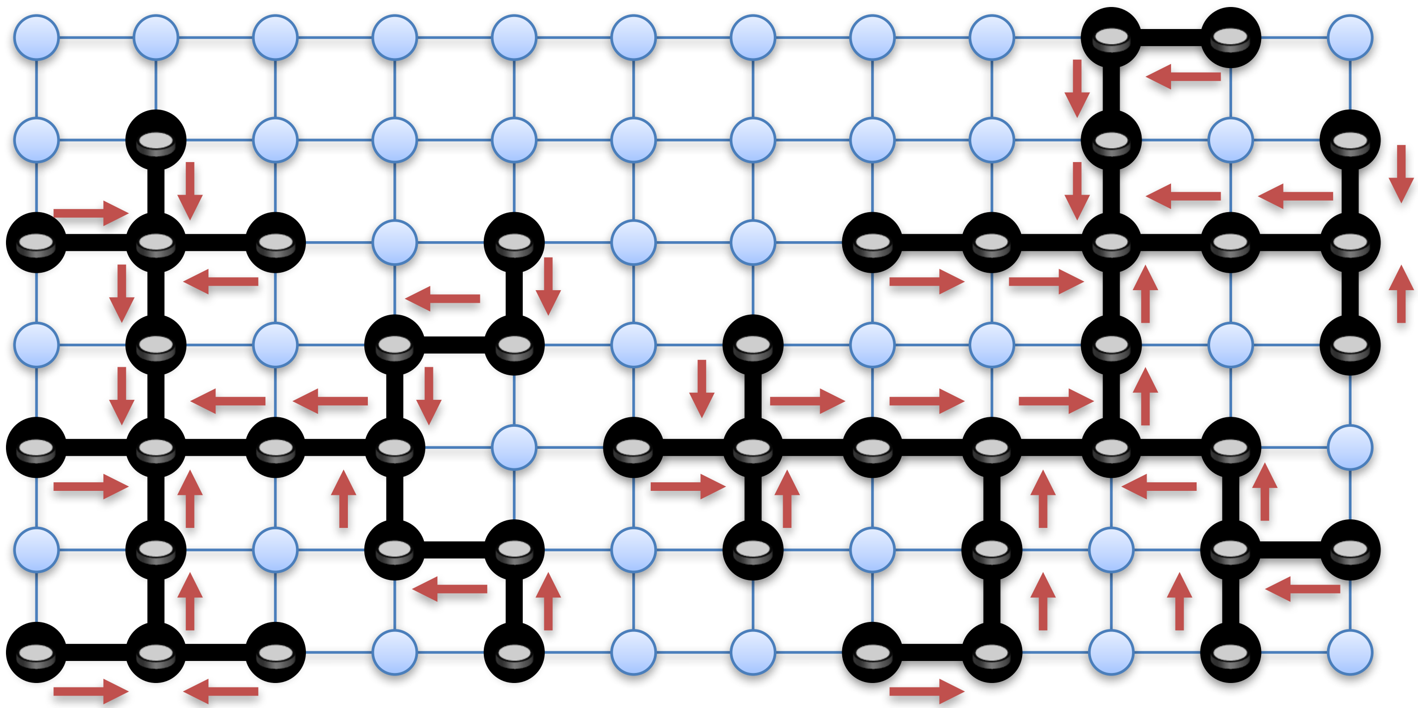
Splash Gibbs Sampler

- **Step 2:** Calibrate the trees in parallel



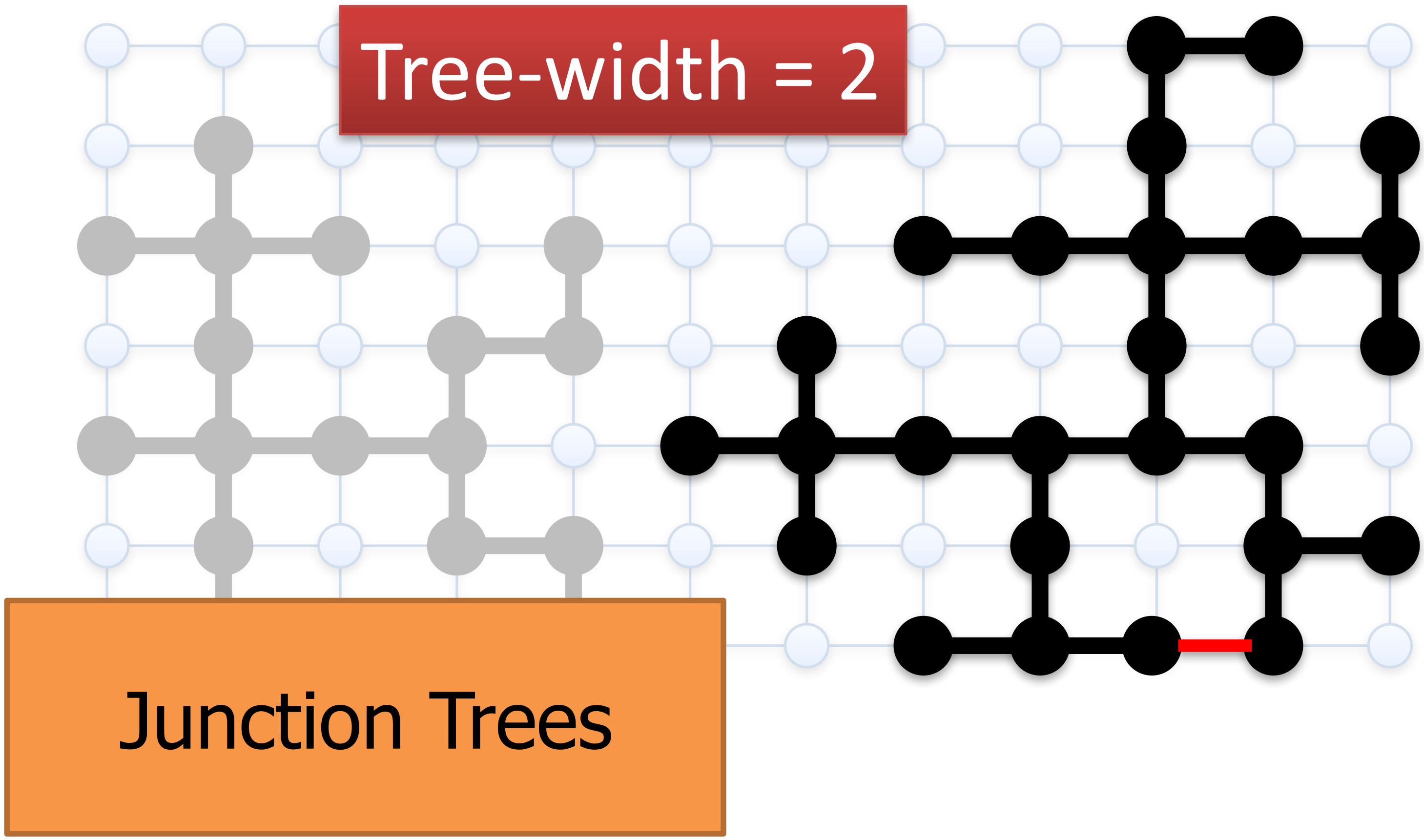
Splash Gibbs Sampler

- **Step 3:** Sample trees in parallel



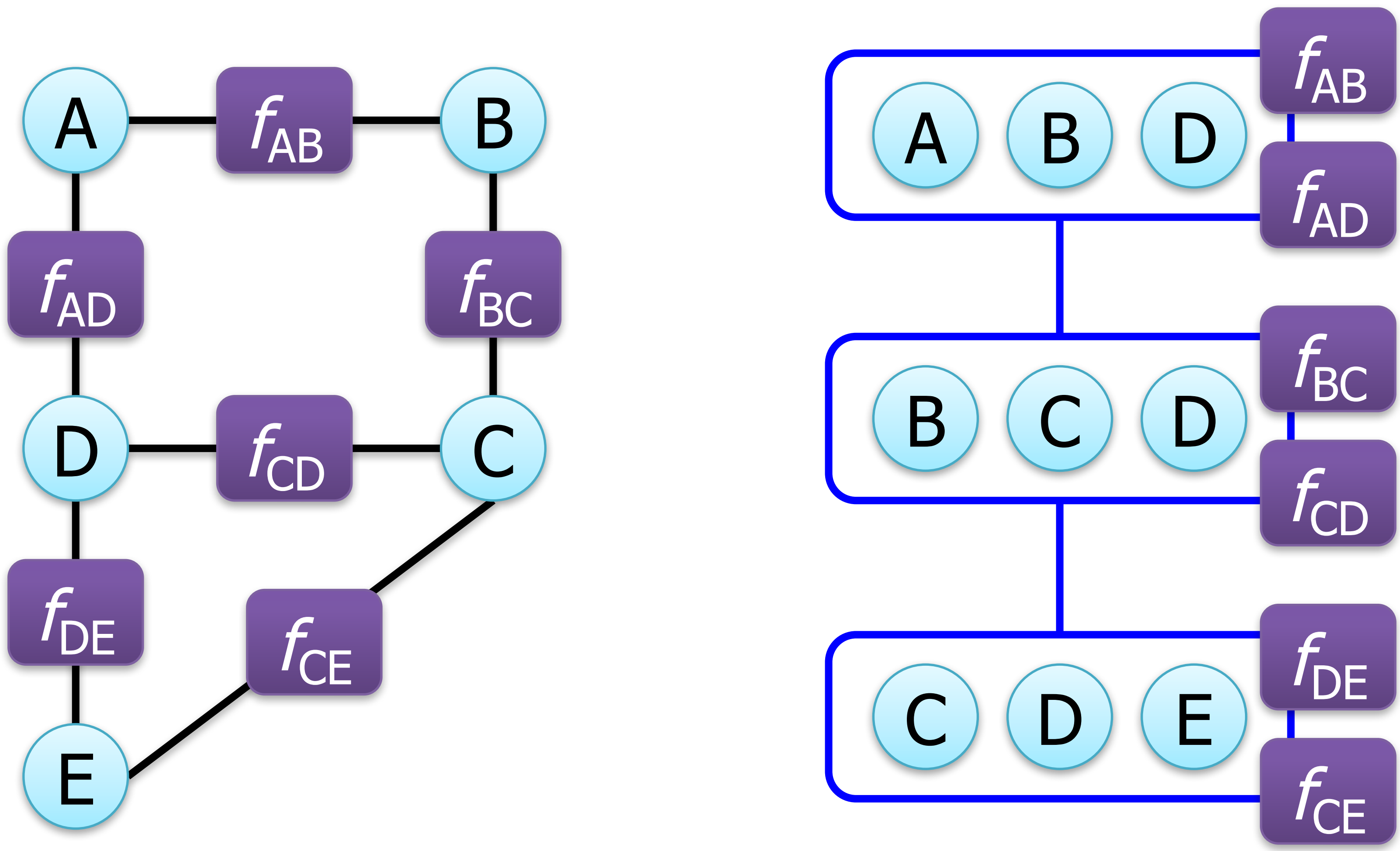
Higher Treewidth Splashes

- **Recall:**



Junction Trees

- Data structure used for exact inference in loopy graphical models



Tree-width = 2

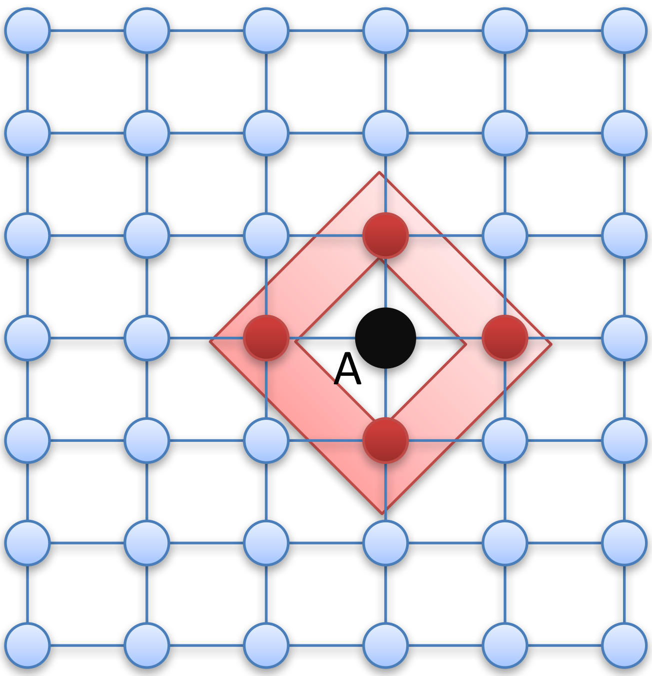
Splash Thin Junction Tree

- Parallel Splash Junction Tree Algorithm
  - Construct multiple conditionally independent thin (bounded treewidth) junction trees **Splashes**
    - Sequential junction tree extension
  - Calibrate the each thin junction tree in parallel
    - Parallel belief propagation
  - Exact backward sampling
    - Parallel exact sampling

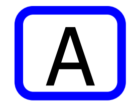
Splash generation

- Frontier extension algorithm:

Markov Random Field



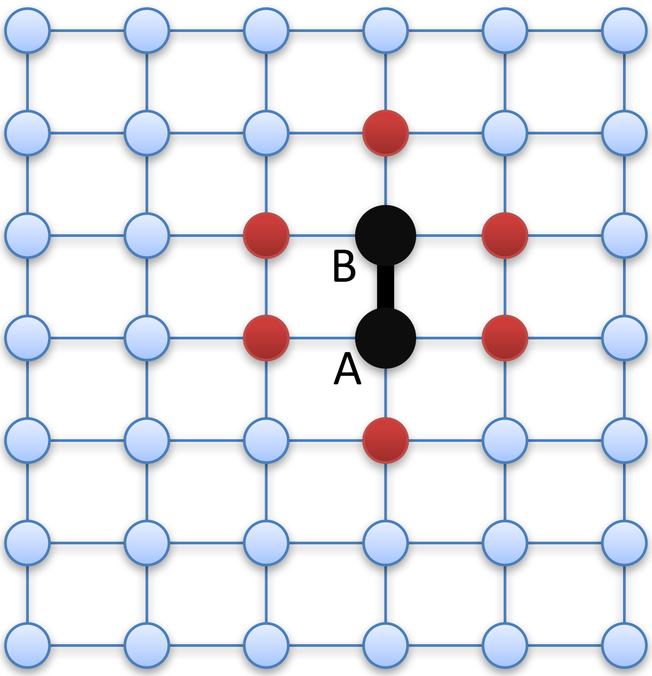
Corresponding Junction tree



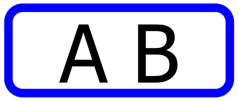
Splash generation

- Frontier extension algorithm:

Markov Random Field



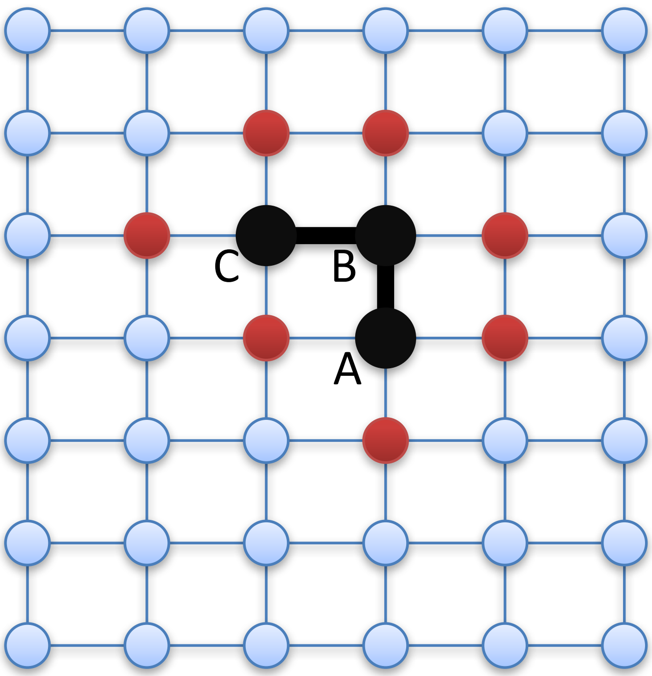
Corresponding Junction tree



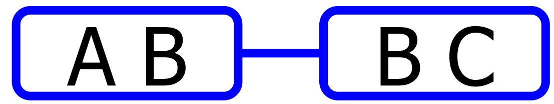
Splash generation

- Frontier extension algorithm:

Markov Random Field



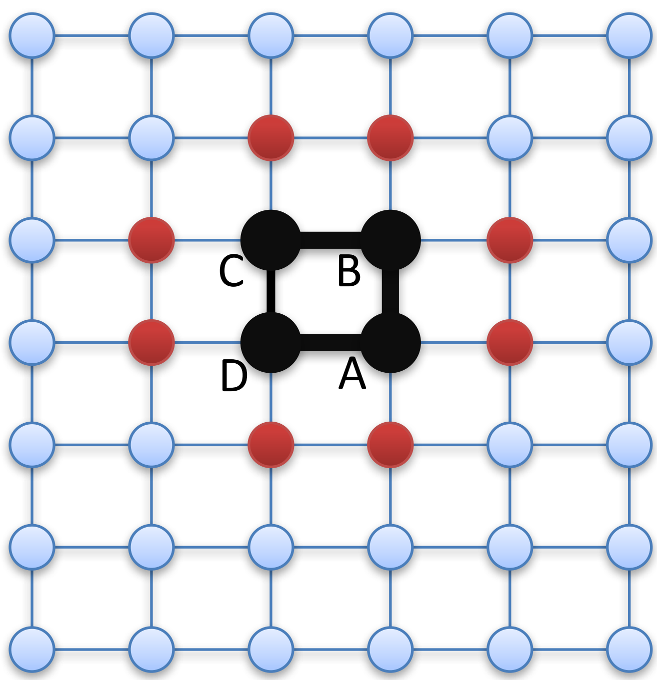
Corresponding Junction tree



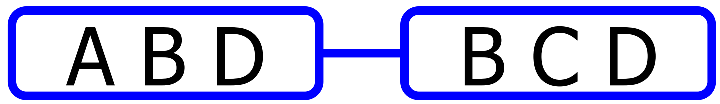
Splash generation

- Frontier extension algorithm:

Markov Random Field



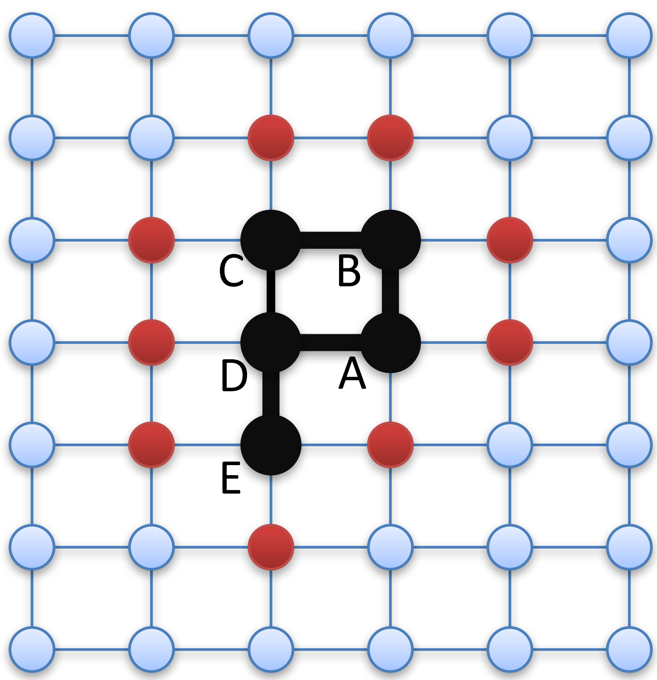
Corresponding Junction tree



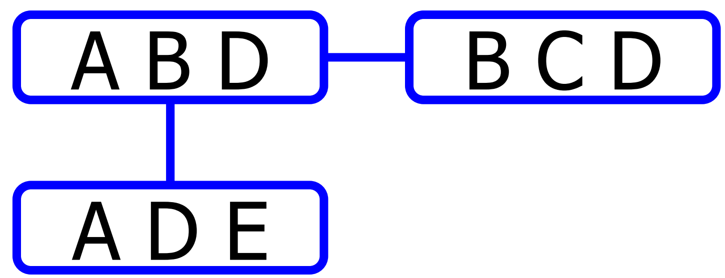
Splash generation

- Frontier extension algorithm:

Markov Random Field



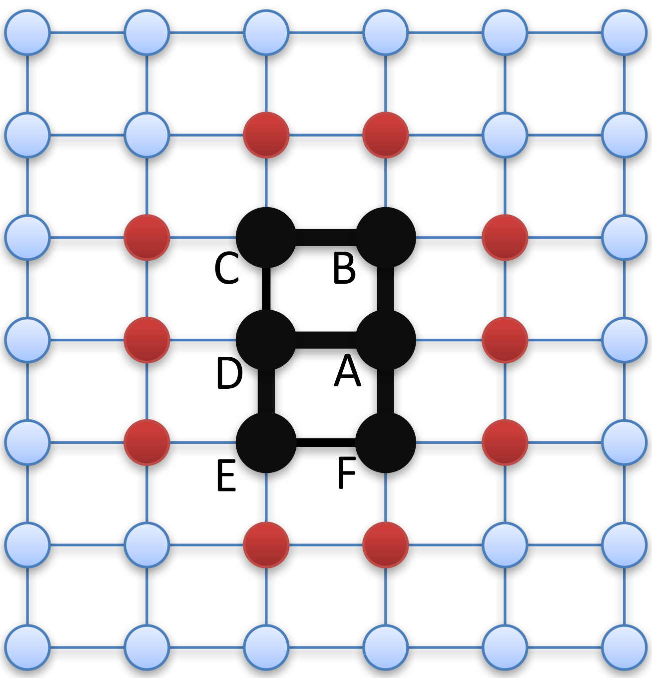
Corresponding Junction tree



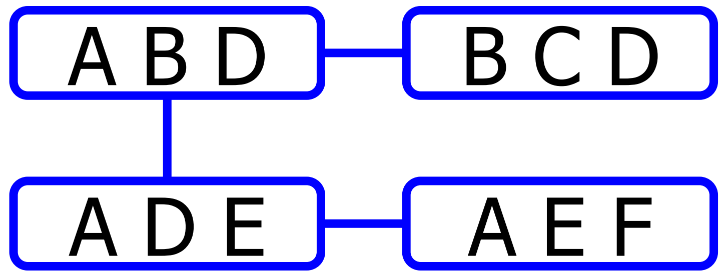
Splash generation

- Frontier extension algorithm:

Markov Random Field



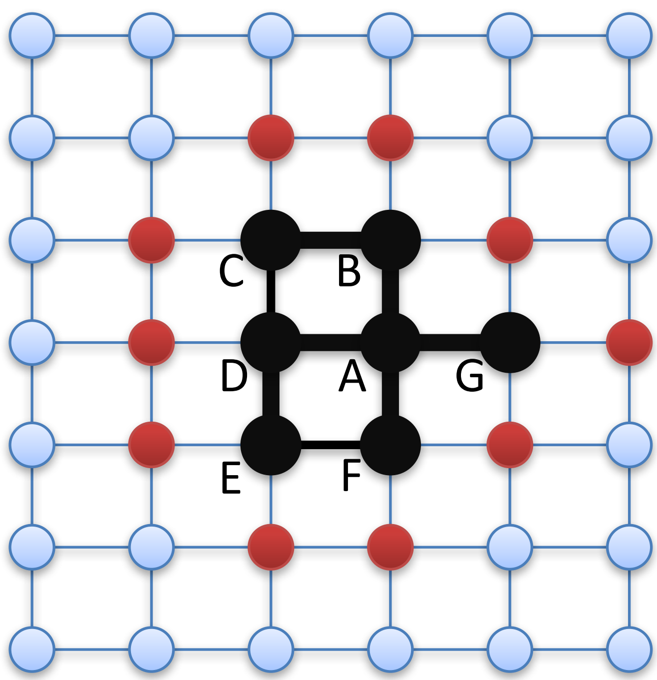
Corresponding Junction tree



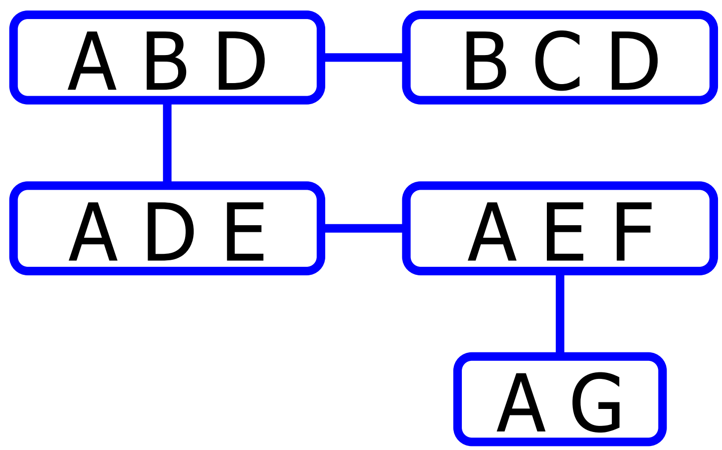
Splash generation

- Frontier extension algorithm:

Markov Random Field



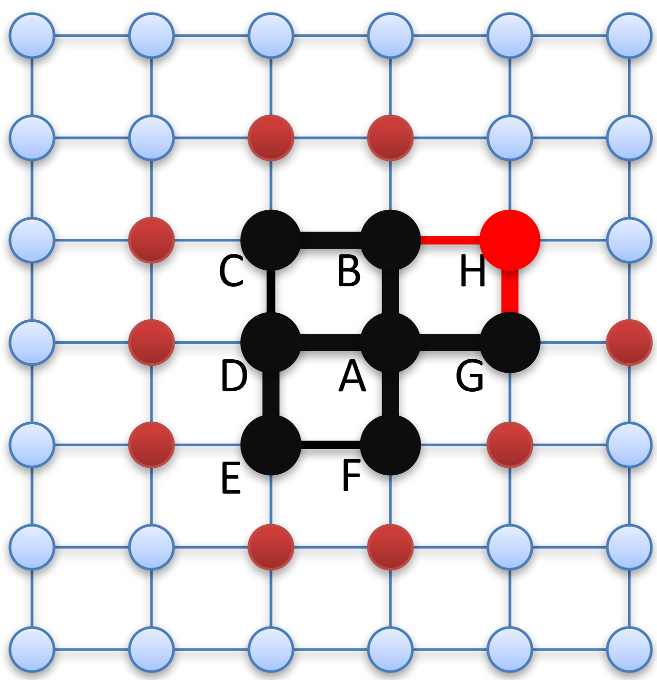
Corresponding Junction tree



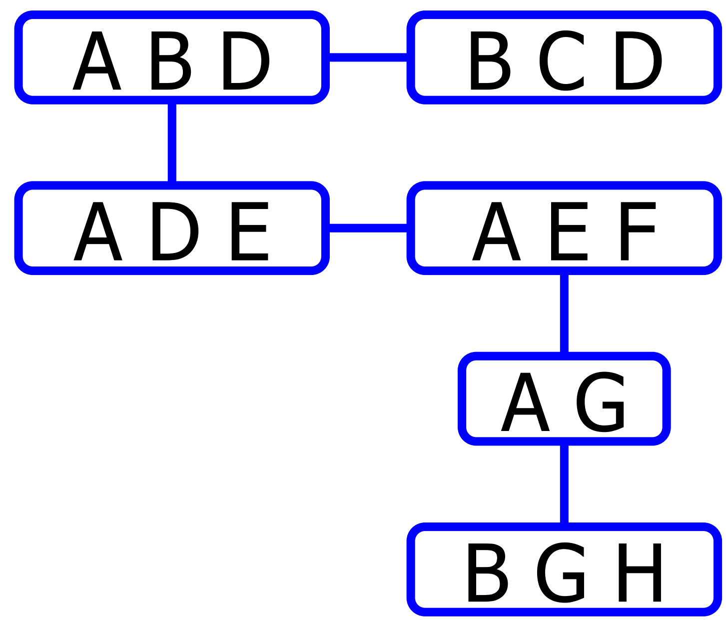
Splash generation

- Frontier extension algorithm:

Markov Random Field



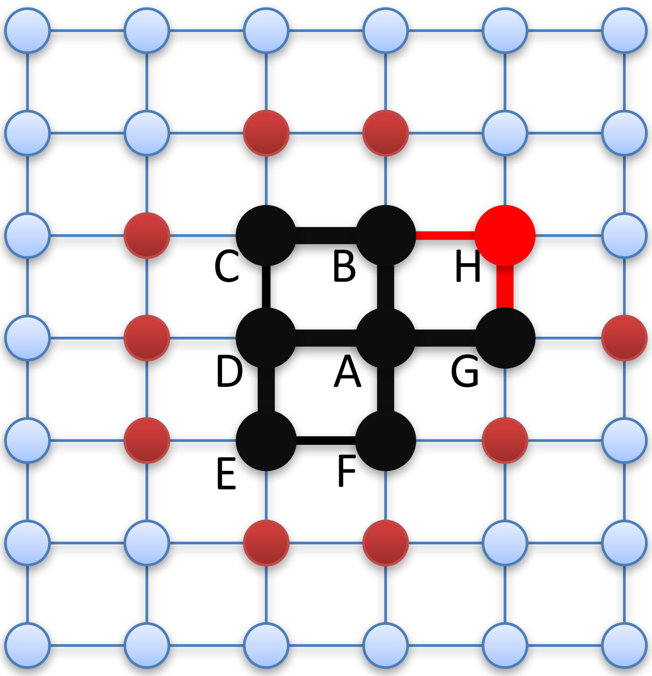
Corresponding Junction tree



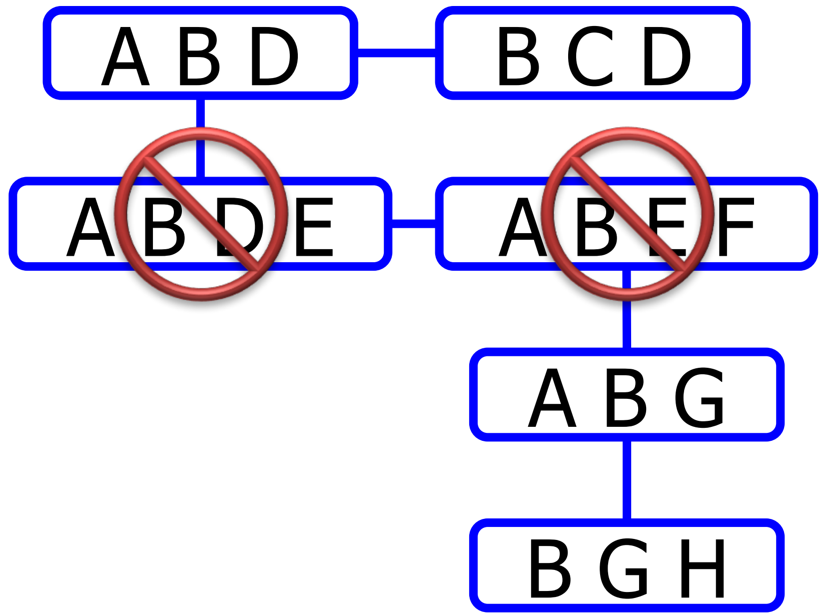
Splash generation

- Frontier extension algorithm:

Markov Random Field



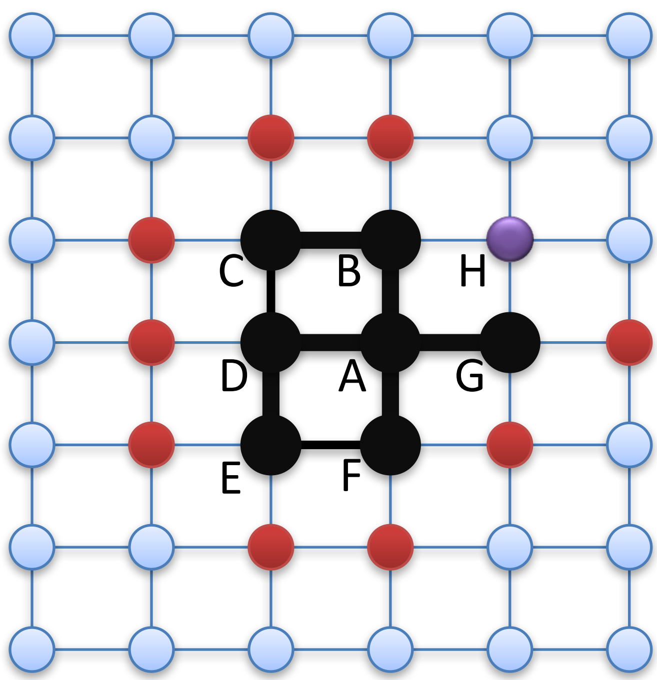
Corresponding Junction tree



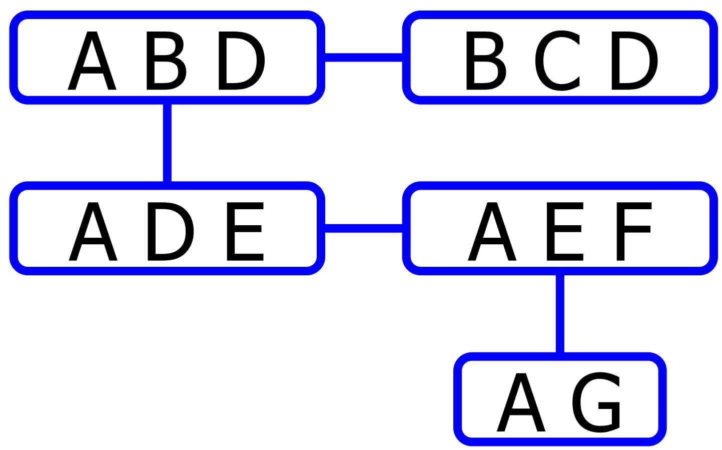
Splash generation

- Frontier extension algorithm:

Markov Random Field



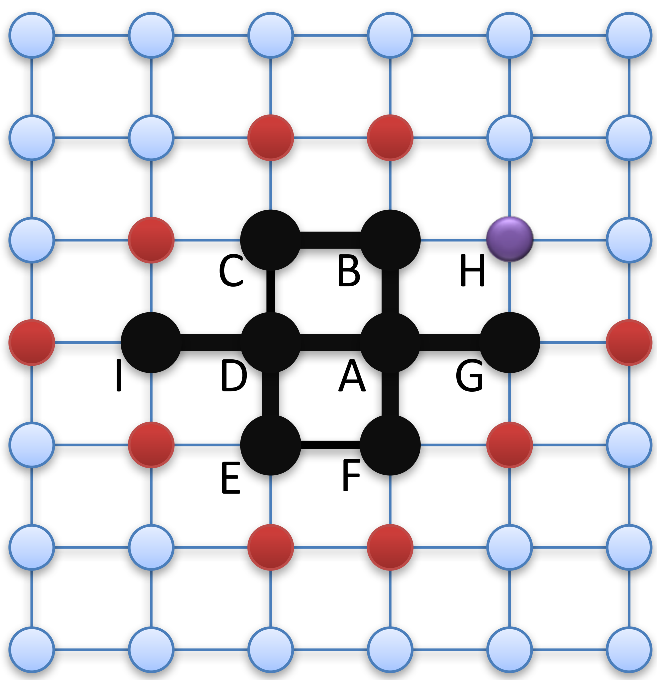
Corresponding Junction tree



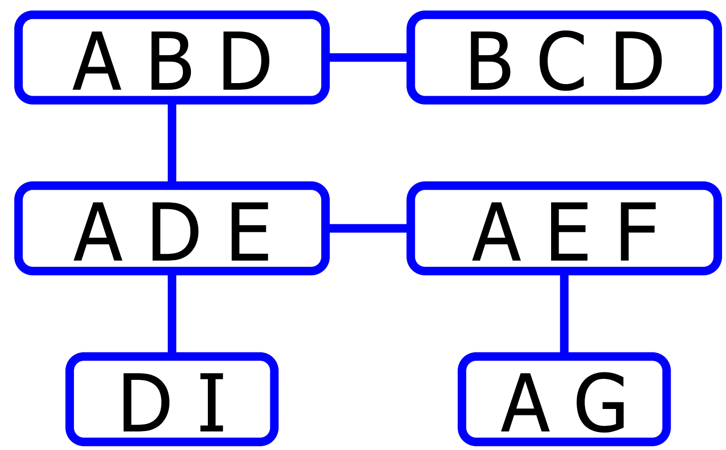
Splash generation

- Frontier extension algorithm:

Markov Random Field

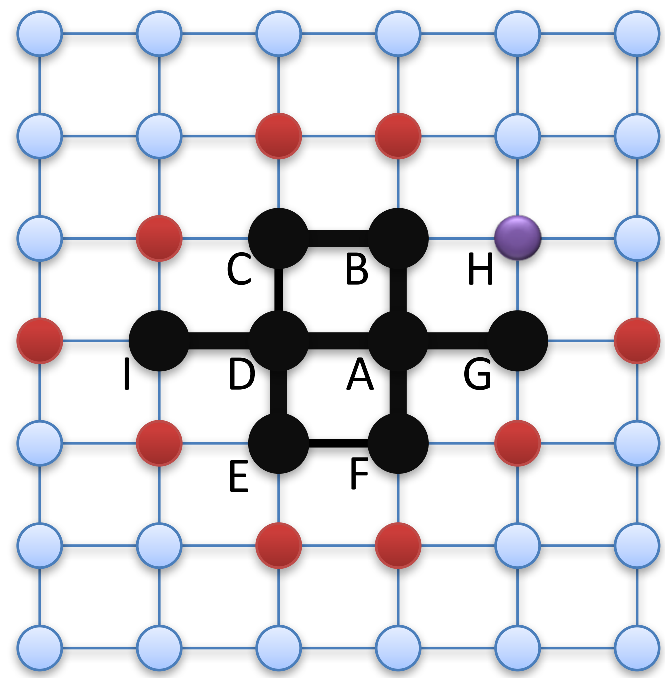


Corresponding Junction tree



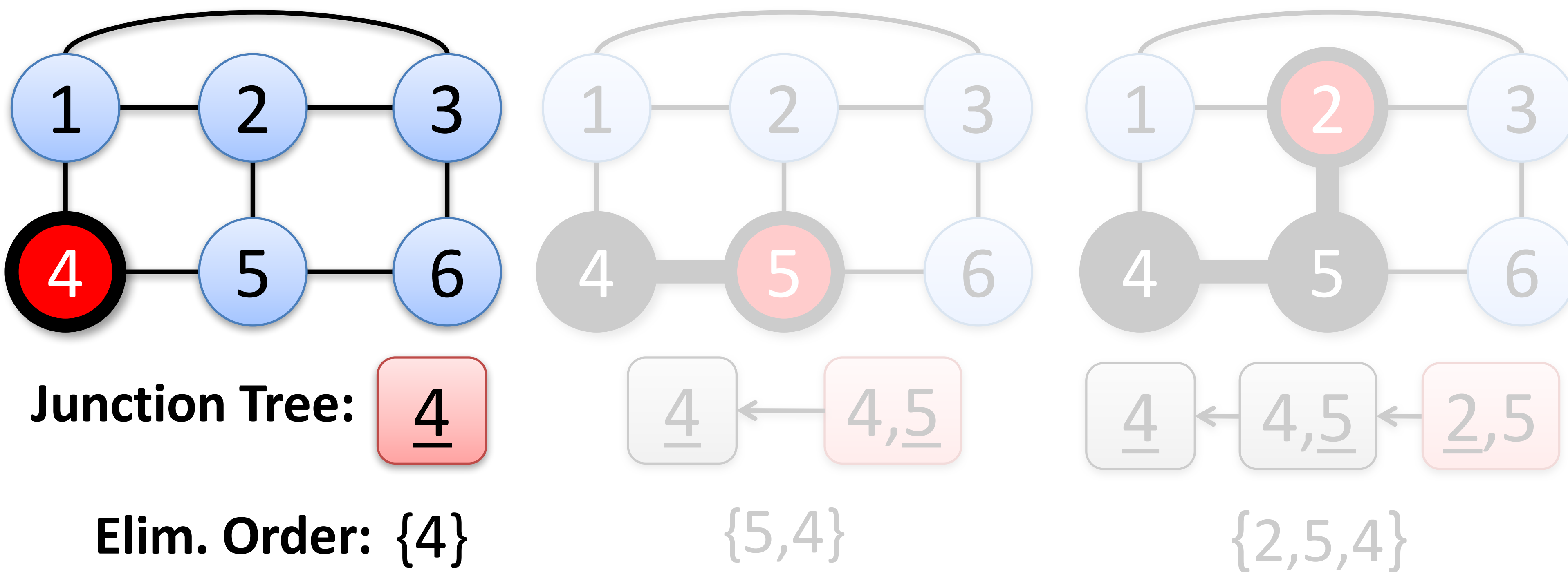
Splash generation

- Challenge:
  - Efficiently reject vertices that violate treewidth constraint
  - Efficiently extend the junction tree
  - Choosing the next vertex
- Solution Splash Junction Trees:
  - Variable elimination with reverse visit ordering
    - I,G,F,E,D,C,B,A
  - Add new clique and update RIP
    - If a clique is created which exceeds treewidth terminate extension
  - Adaptive prioritize boundary



Incremental Junction Trees

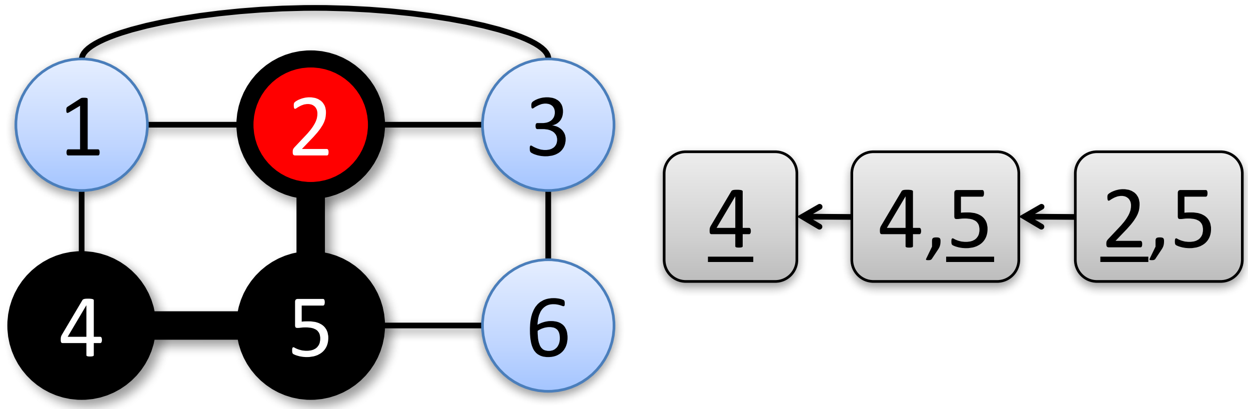
- First 3 Rounds:



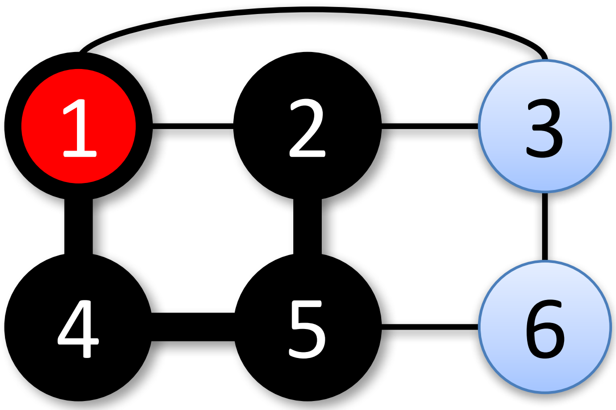
Incremental Junction Trees

- Result of third round:

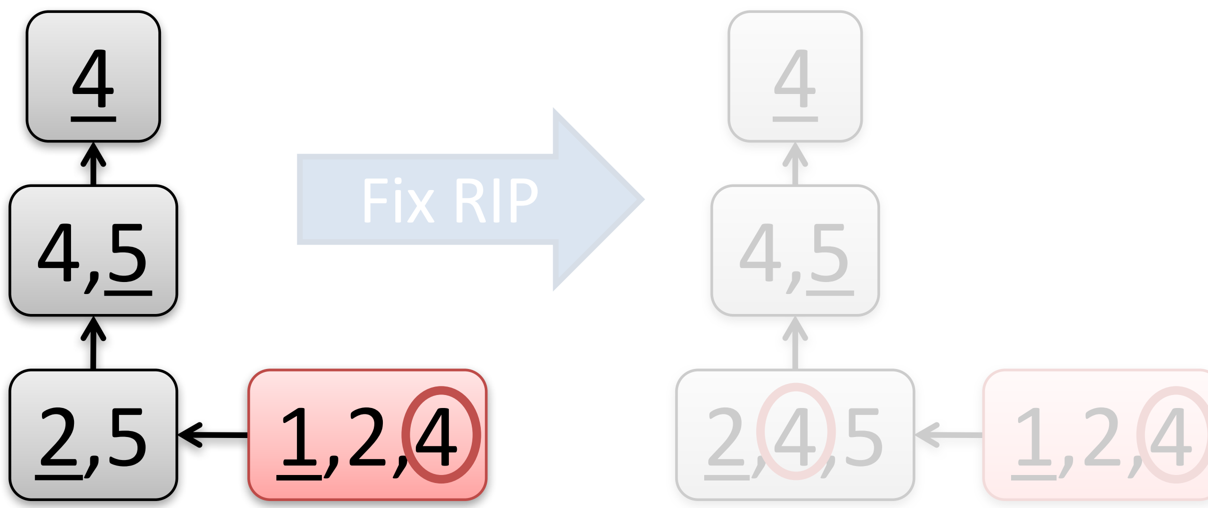
{2,5,4}



- Fourth round:



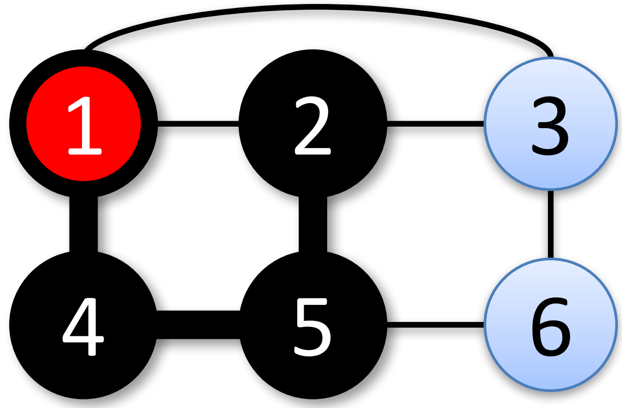
{1,2,5,4}

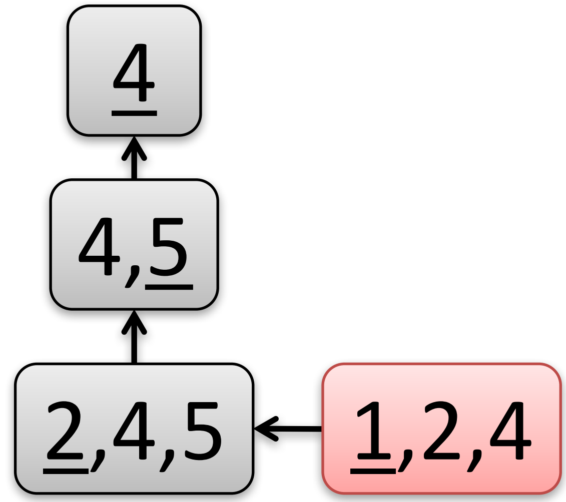


Incremental Junction Trees

- Results from 4th round:

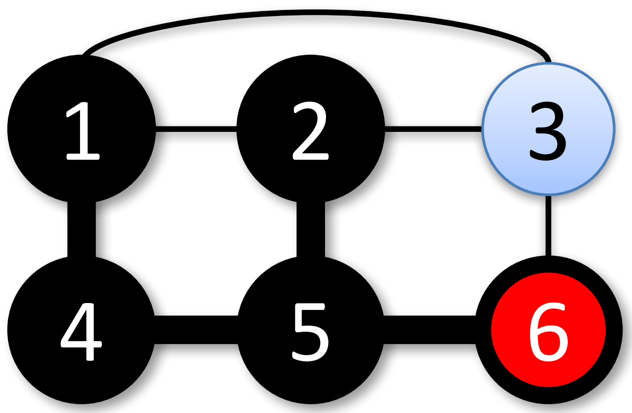
{1,2,5,4}

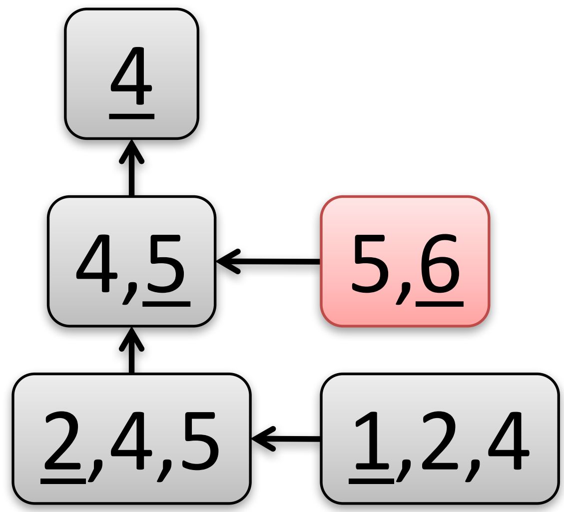




- 5th Round:

{6,1,2,5,4}

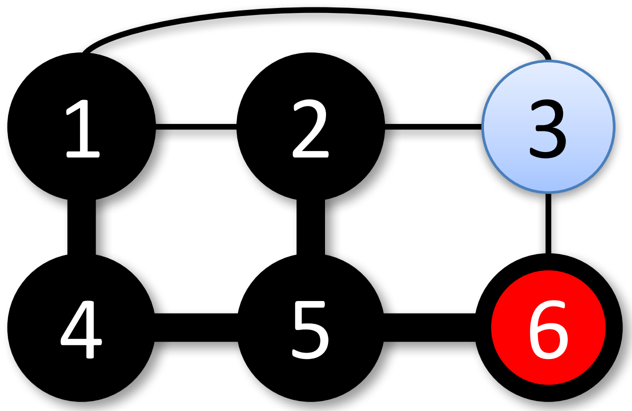


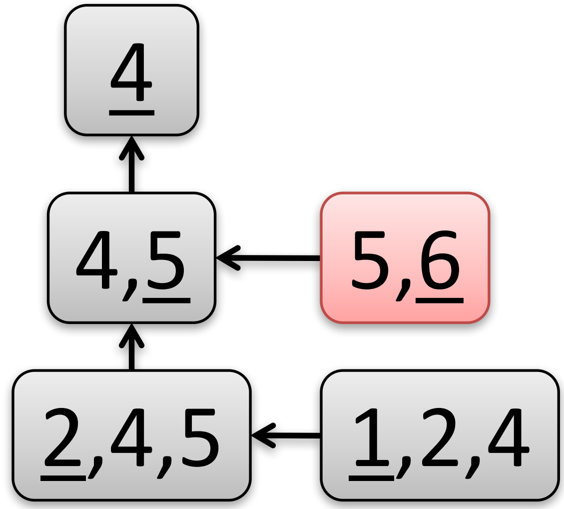


Incremental Junction Trees

- Results from 5th round:

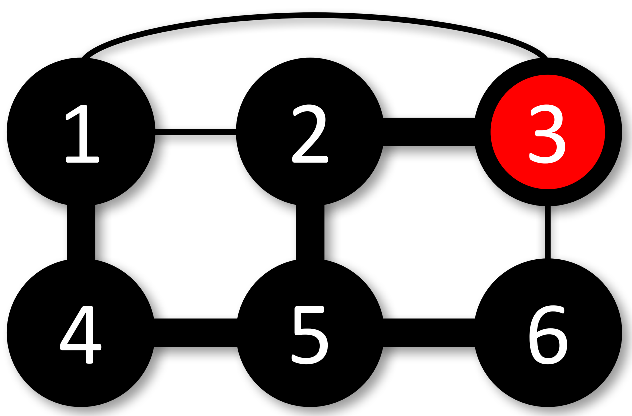
{6,1,2,5,4}

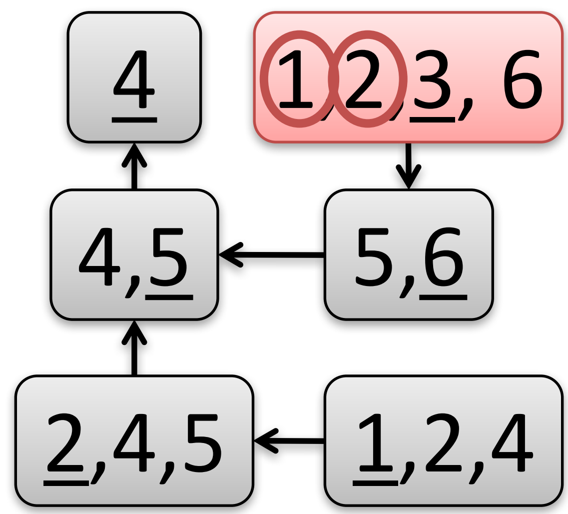




- 6th Round:

{3,6,1,2,5,4}

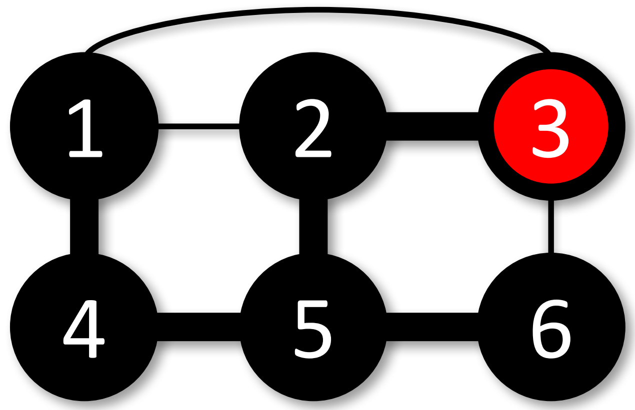


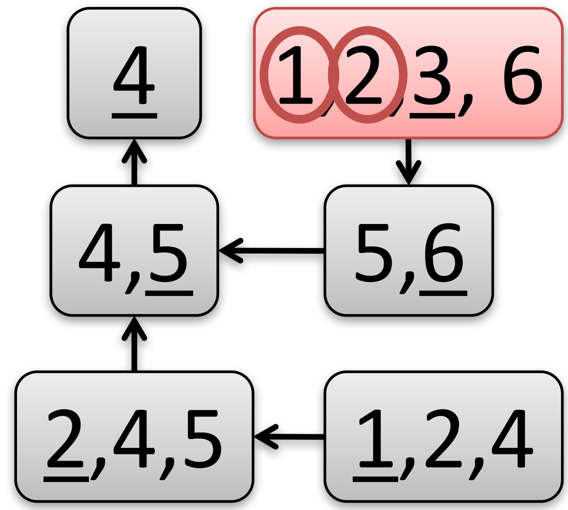


Incremental Junction Trees

- Finishing 6th round:

{3,6,1,2,5,4}





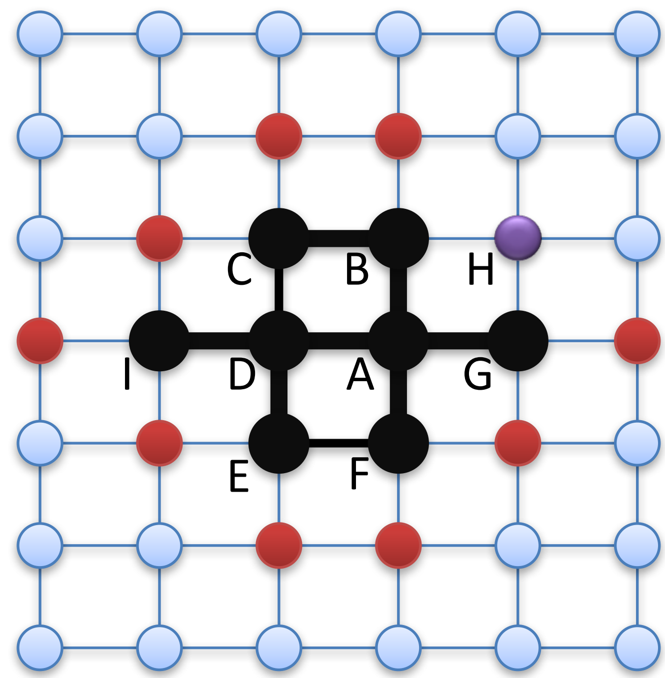
Algorithm Block [Skip]

Input: The original junction tree $(\mathcal{C}, E) = \mathbf{J}_{\mathcal{S}}$.
Input: The variable X_i to add to $\mathbf{J}_{\mathcal{S}}$
Output: $\mathbf{J}_{\mathcal{S}+i}$
Define : $\mathbf{C}_u$ as the clique created by eliminating $u \in \mathcal{S}$
Define : $\mathtt{V}[\mathbf{C}] \in \mathcal{S}$ as the variable eliminated when creating $\mathbf{C}$
Define : $\mathtt{t}[v]$ as the time $v \in \mathcal{S}$ was added to $\mathcal{S}$
Define : $\mathtt{P}[v] \in \mathcal{N}_v \cap \mathcal{S}$ as the next neighbor of v to be eliminated.

$\mathbf{C}_i \leftarrow (\mathcal{N}_i \cap \mathcal{S}) \cup \{i\}$
$\mathtt{P}[i] \leftarrow \arg\max_{v \in \mathbf{C}_i \setminus \{i\}} \mathtt{t}[v]$
`// ----------- Repair RIP -------------`
$\mathcal{R} \leftarrow \mathbf{C}_i \setminus \{i\}$ `// RIP Set`
$v \leftarrow \mathtt{P}[i]$
while $|\mathcal{R}| > 0$ **do**
    $\mathbf{C}_v \leftarrow \mathbf{C}_v \cup \mathcal{R}$ `// Add variables to parent`
    $w \leftarrow \arg\max_{w \in \mathbf{C}_v \setminus \{v\}} \mathtt{t}[w]$ `// Find new parent`
    if $w = \mathtt{P}[v]$ **then**
        $\mathcal{R} \leftarrow (\mathcal{R} \setminus \mathbf{C}_i) \setminus \{i\}$
    else
        $\mathcal{R} \leftarrow (\mathcal{R} \cup \mathbf{C}_i) \setminus \{i\}$
        $\mathtt{P}[v] \leftarrow w$ `// New parent`
    $v \leftarrow \mathtt{P}[v]$ `// Move upwards`

Splash generation

- Challenge:
  - Efficiently reject vertices that violate treewidth constraint
  - Efficiently extend the junction tree
  - Choosing the next vertex
- Solution Splash Junction Trees:
  - Variable elimination with reverse visit ordering
    - I,G,F,E,D,C,B,A
  - Add new clique and update RIP
    - If a clique is created which exceeds treewidth terminate extension
  - Adaptive prioritize boundary



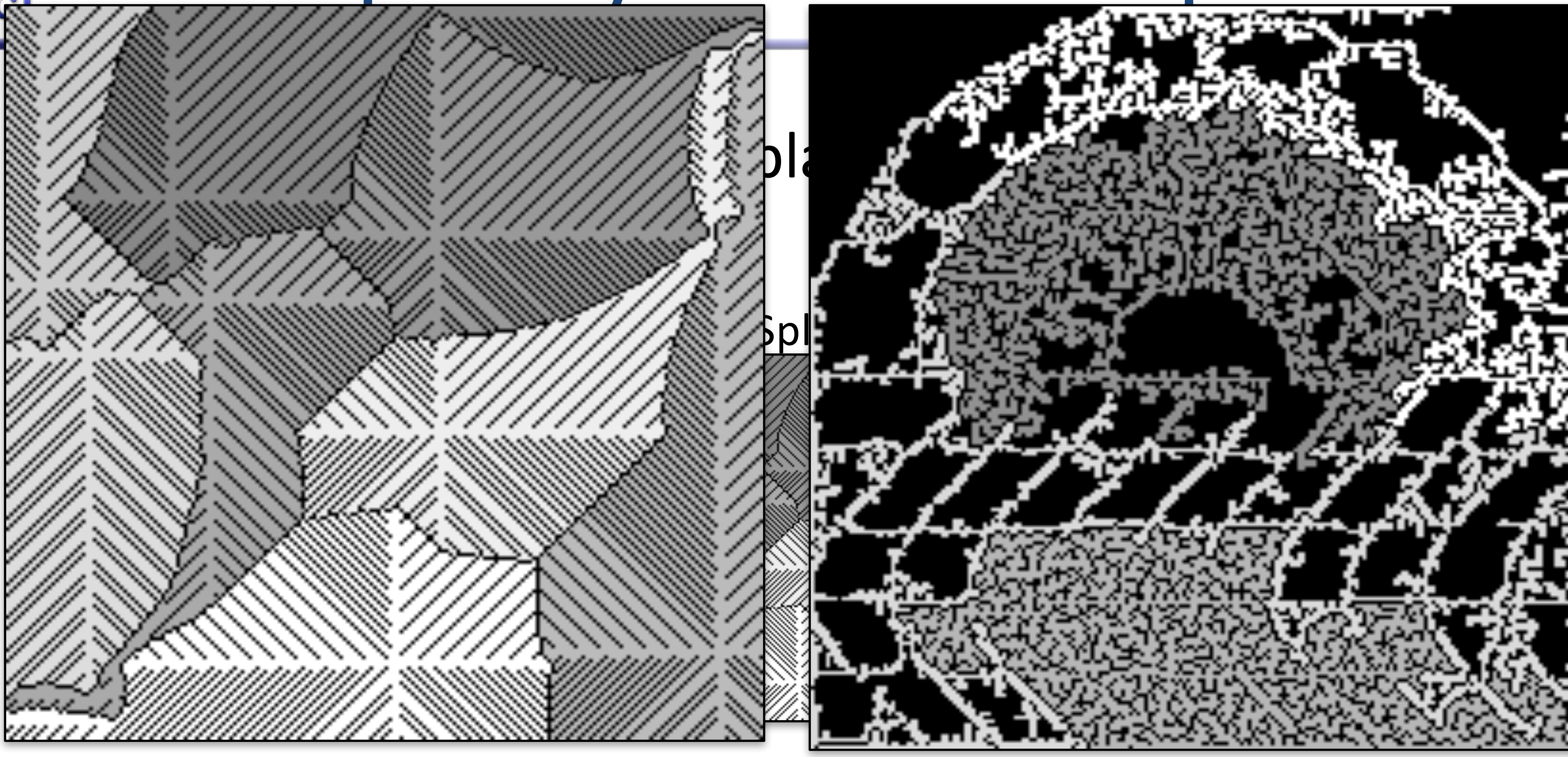
Adaptive Vertex Priorities

- Assign priorities to boundary vertices:

$$\mathbf{s}[X_v] = \left\| \log \frac{\sum_x \pi\left(X_{\mathcal{S}}, X_v = x \mid X_{-\mathcal{S}} = x_{-\mathcal{S}}^{(t)}\right)}{\pi\left(X_{\mathcal{S}} \mid X_v = x_v^{(t)}, X_{-\mathcal{S}} = x_{-\mathcal{S}}^{(t)}\right)} \right\|_1$$

  - Can be computed using only factors that depend on X_v
  - Based on current sample
  - Captures difference between marginalizing out the variable (in Splash) fixing its assignment (out of Splash)
  - Exponential in treewidth
- Could consider other metrics …

Adaptively Prioritized Splashes



- Provably converges to the correct distribution
  - Requires vanishing adaptation
  - Identify a bug in the Levine & Casella seminal work in adaptive random scan

Experiments

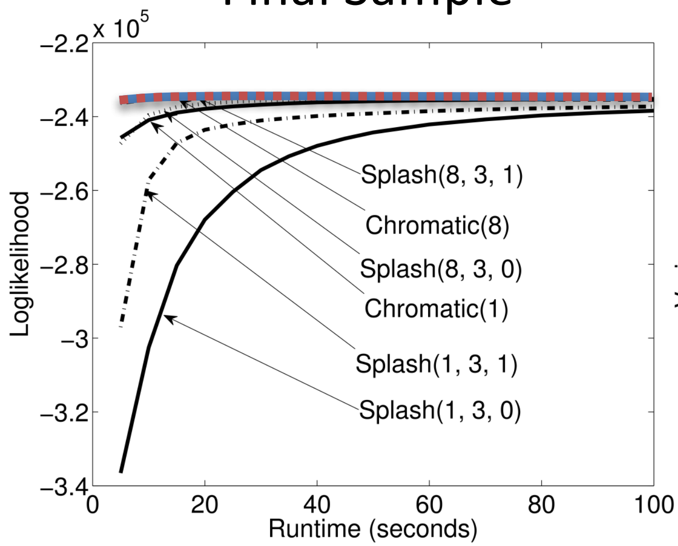
- Implemented using GraphLab
  - Treewidth = 1 :
    - Parallel tree construction, calibration, and sampling
    - No incremental junction trees needed
  - Treewidth > 1 :
    - Sequential tree construction (use multiple Splashes)
    - Parallel calibration and sampling
    - Requires incremental junction trees
  - Relies heavily on:
    - Edge consistency model to prove ergodicity
    - FIFO/ Prioritized scheduling to construct Splashes
- Evaluated on 32 core Nehalem Server

Rapidly Mixing Model

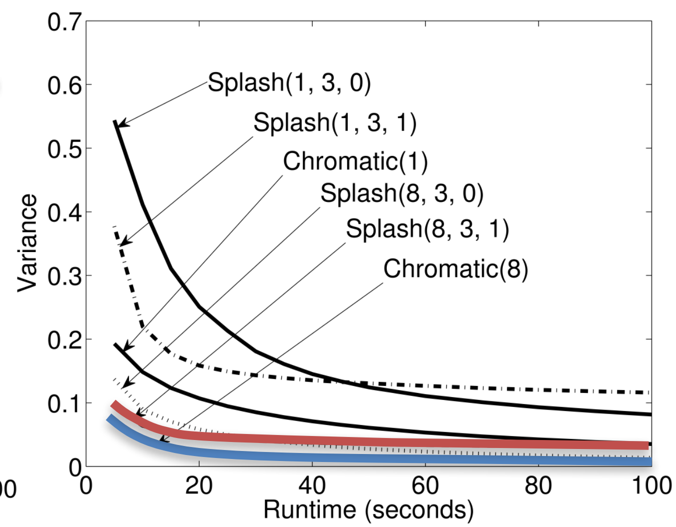
- Grid MRF with weak attractive potentials

  40K Variables 80K Factors

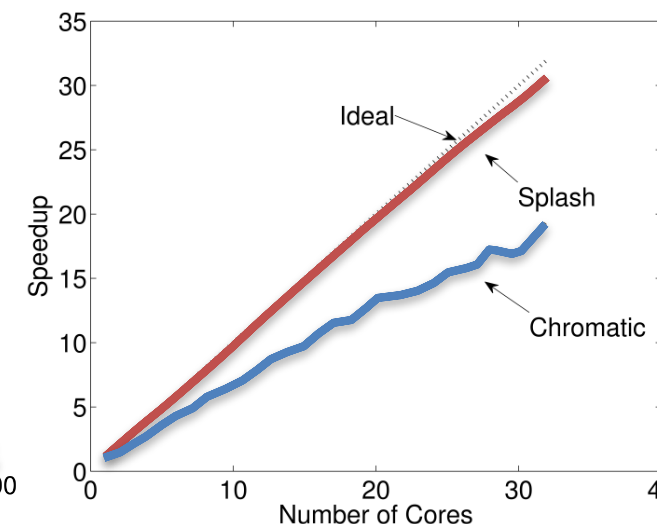
Likelihood Final Sample



"Mixing"



Speedup



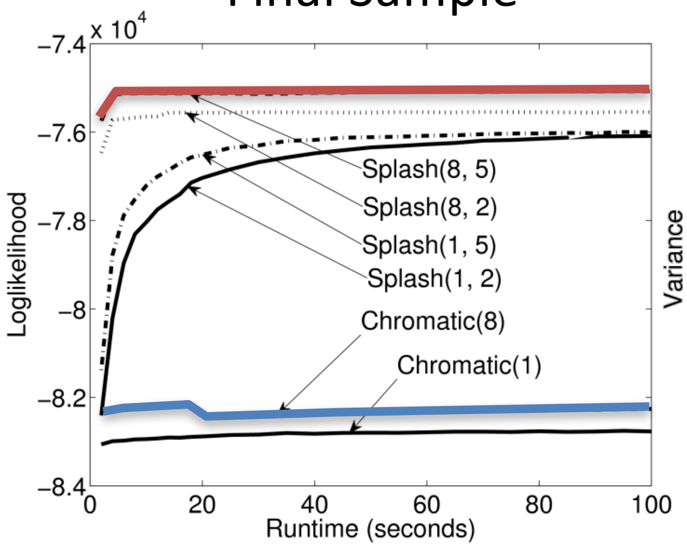
- The Chromatic sampler slightly outperforms the Splash Sampler

Slowly Mixing Model

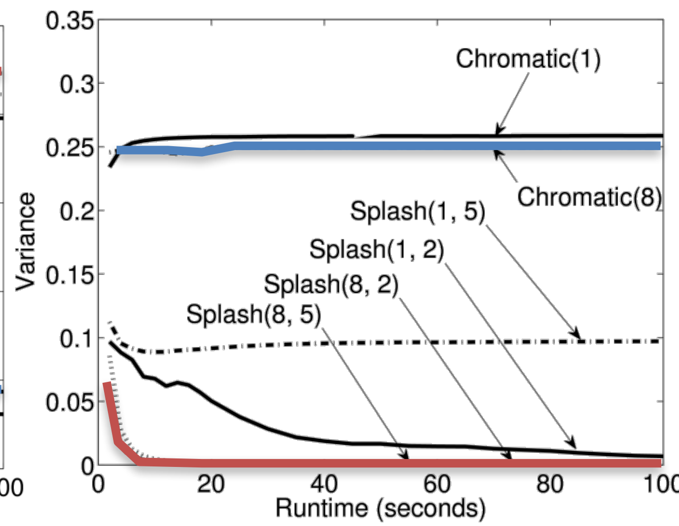
- Markov logic network with strong dependencies

10K Variables 28K Factors

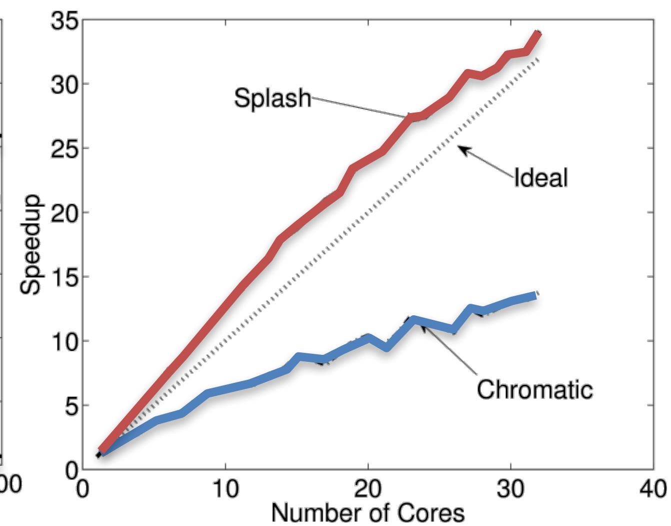
Likelihood Final Sample



"Mixing"



Speedup in Sample Generation



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

Conclusion

- **Chromatic Gibbs** sampler for models with *weak dependencies*
  - *Converges to the correct distribution*
  - *Quantifiable improvement in mixing*
- **Theoretical analysis** of the Synchronous Gibbs sampler on *2-colorable models*
  - *Proved marginal convergence on 2-colorable models*
- **Splash Gibbs** sampler for models with *strong dependencies*
  - *Adaptive asynchronous tree construction*
  - *Experimental evaluation demonstrates an improvement in mixing*

Future Work

- Extend Splash algorithm to models with continuous variables
  - Requires continuous junction trees (Kernel BP)
- Consider “freezing” the junction tree set
  - Reduce the cost of tree generation?
- Develop better adaptation heuristics
  - Eliminate the need for vanishing adaptation?
- Challenges of Gibbs sampling in high-coloring models
  - Collapsed LDA
- High dimensional pseudorandom numbers
  - Not currently addressed in the MCMC literature