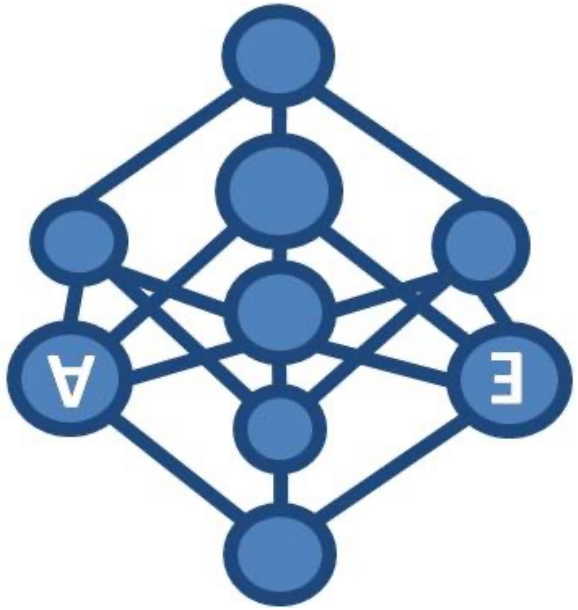
# Probabilistic Graphical Models

## Approximate Inference

TECHNISCHE UNIVERSITÄT DARMSTADT




*Thanks to Adnan Darwiche, Matt Gormley, Carlos Guestrin, Pedro Domingos and many others for making their slides publically available

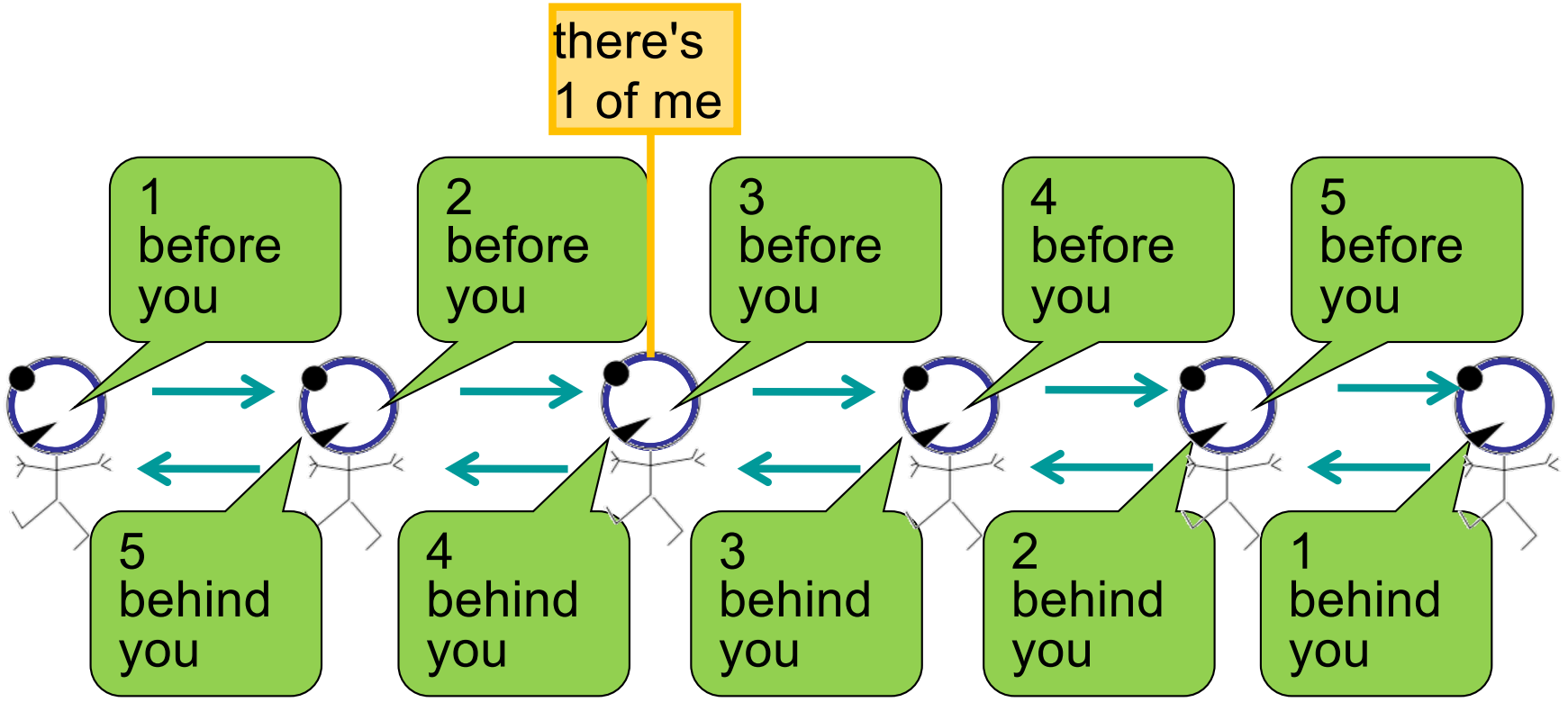
# Today

TECHNISCHE UNIVERSITÄT DARMSTADT

- From class, we know that computing the aposteriori belief of a variable in a general BN is NP-hard
- In particular, exact inference for DBNs is intractable even for simple cases
- Solution: approximate inference
  - Loopy Belief Propagation
  - Sampling

# Message Passing

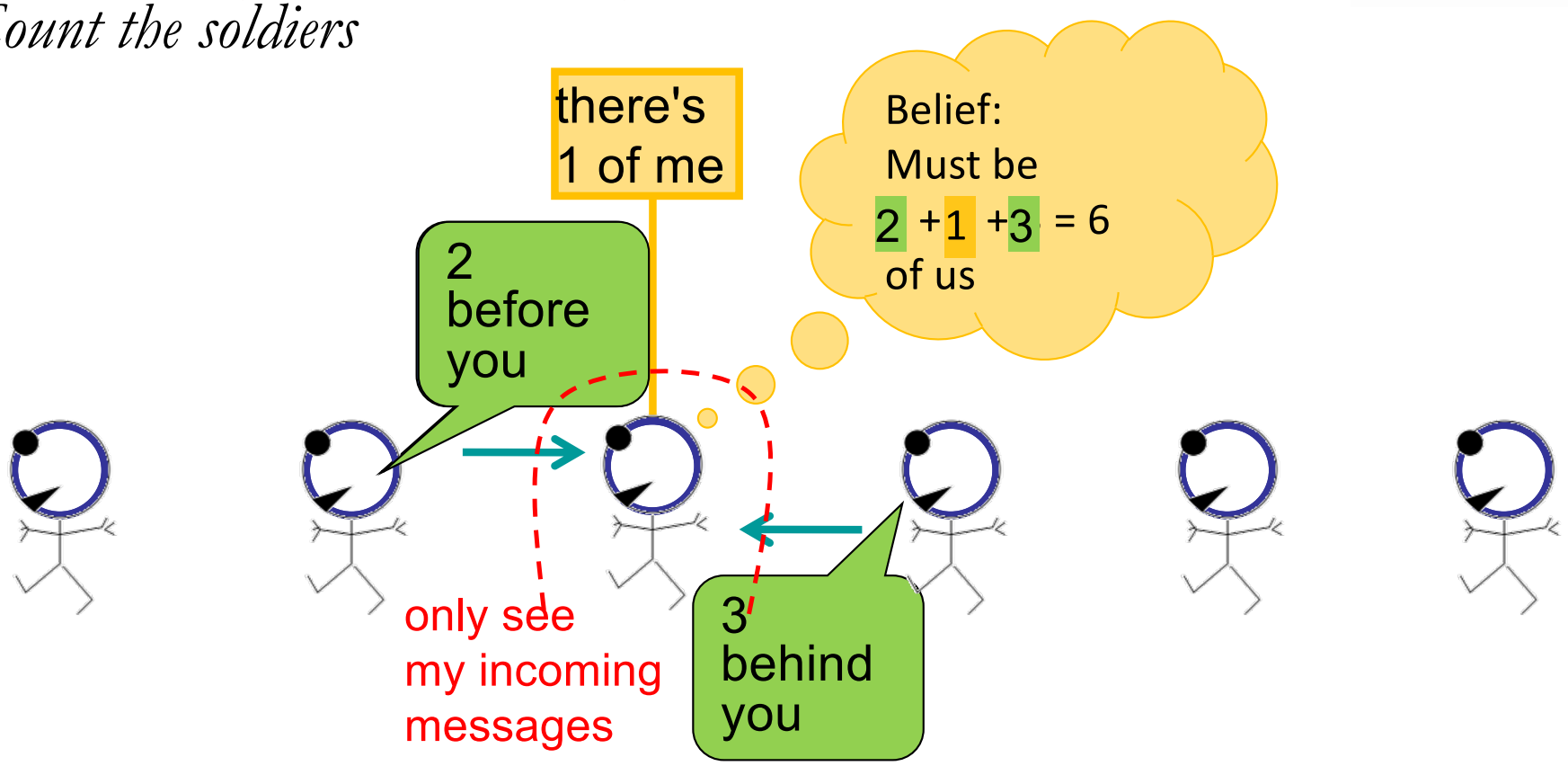
*Count the soldiers*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

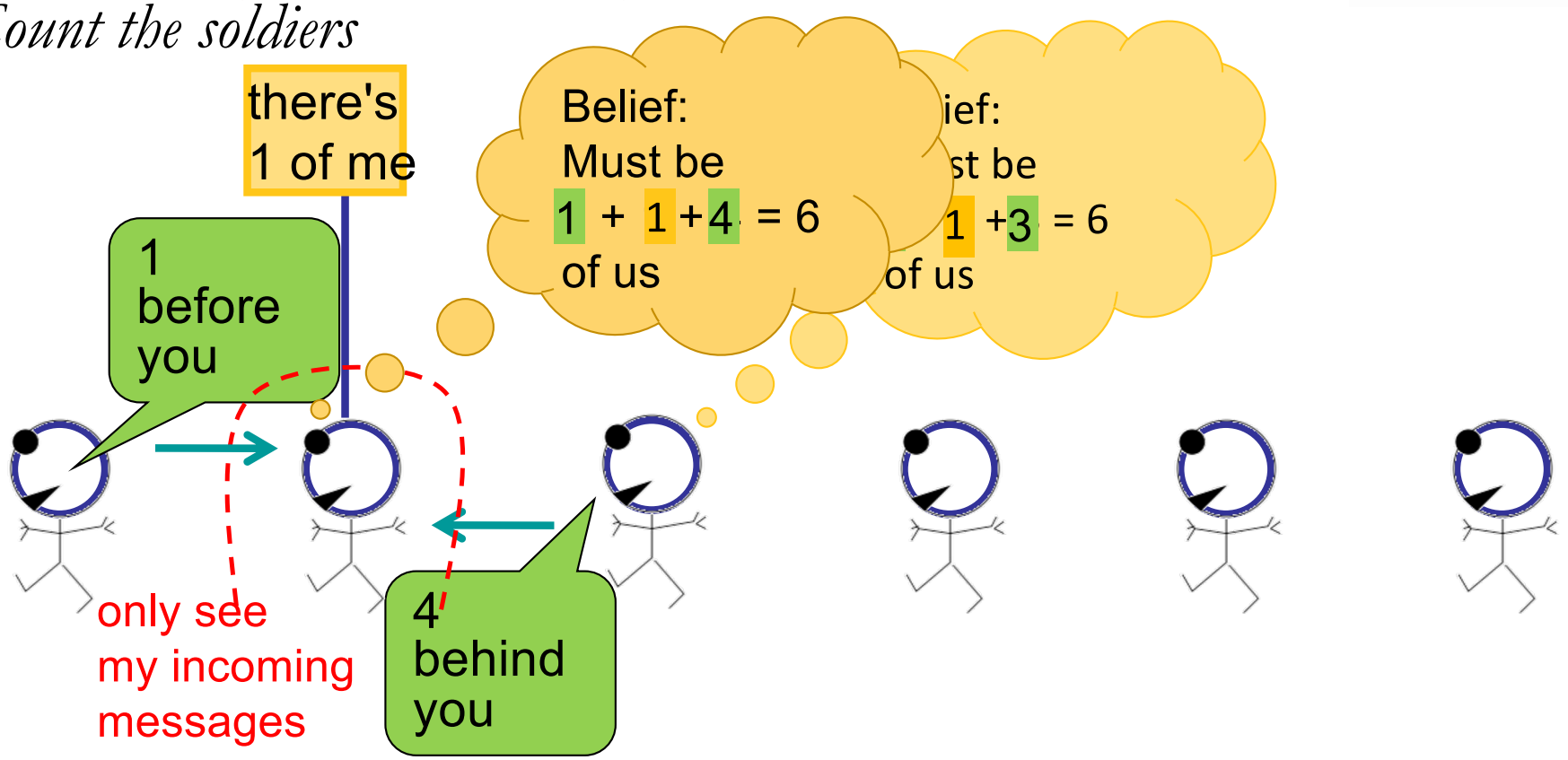
*Count the soldiers*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

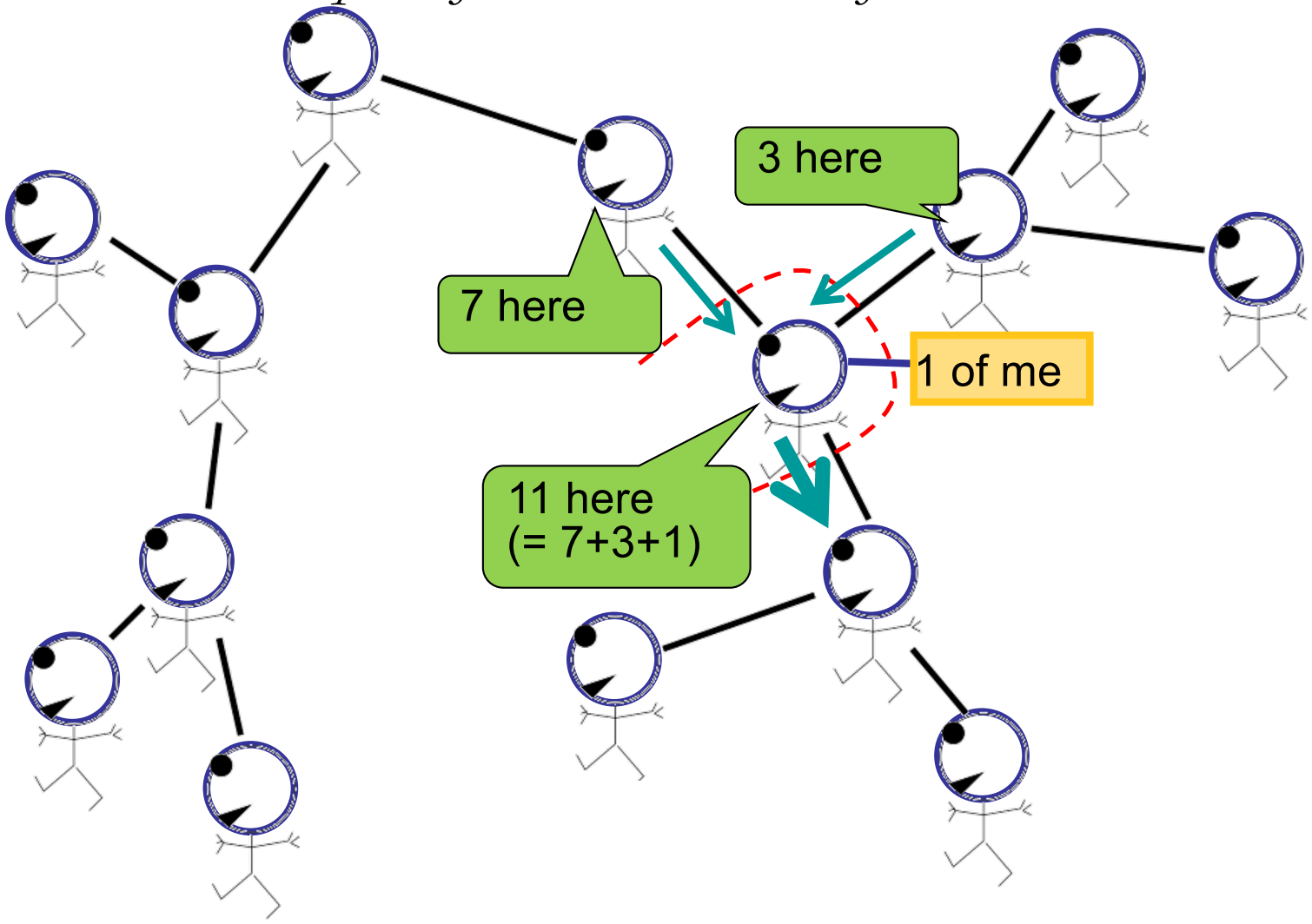
*Count the soldiers*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

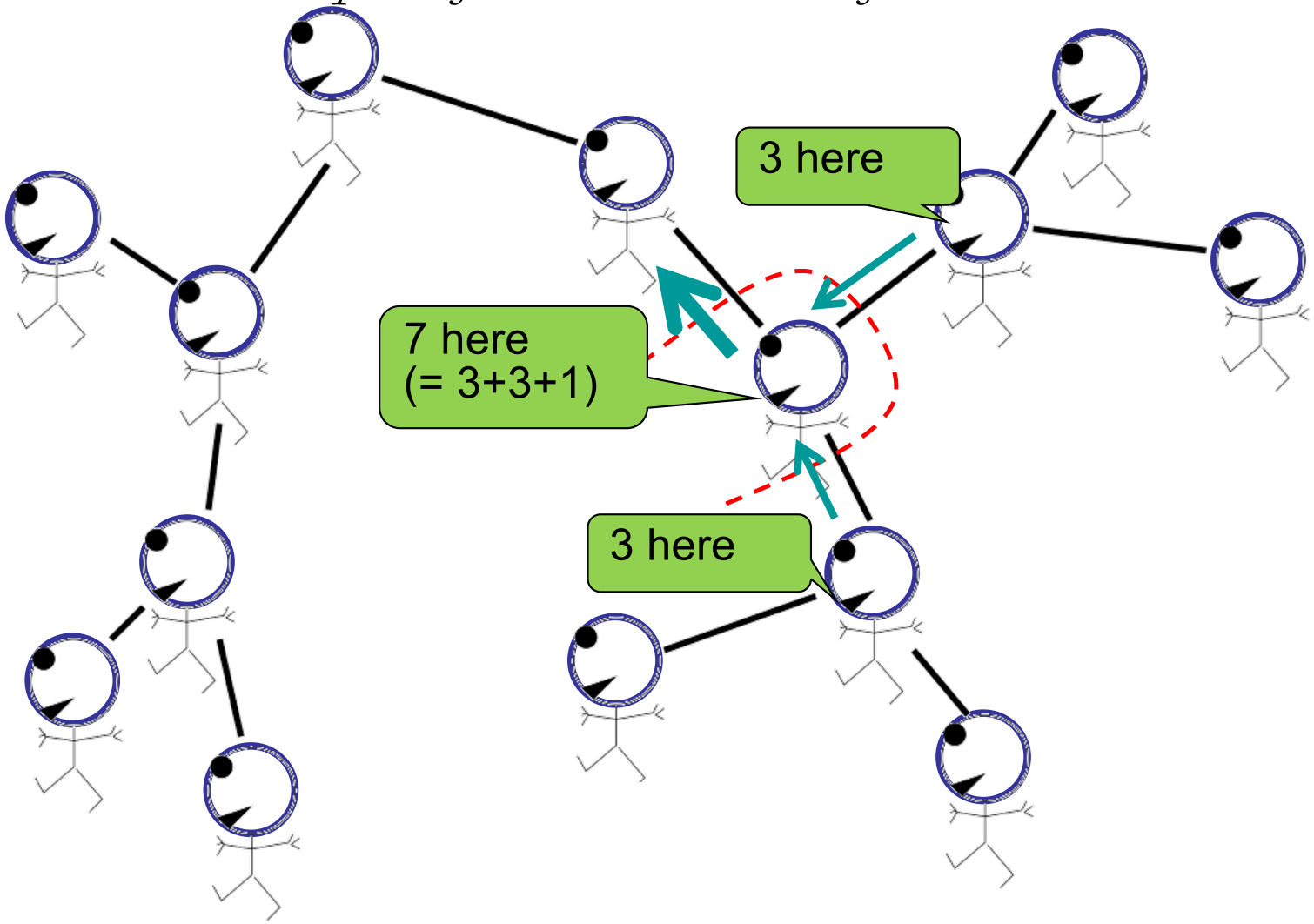
*Each soldier receives reports from all branches of tree*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

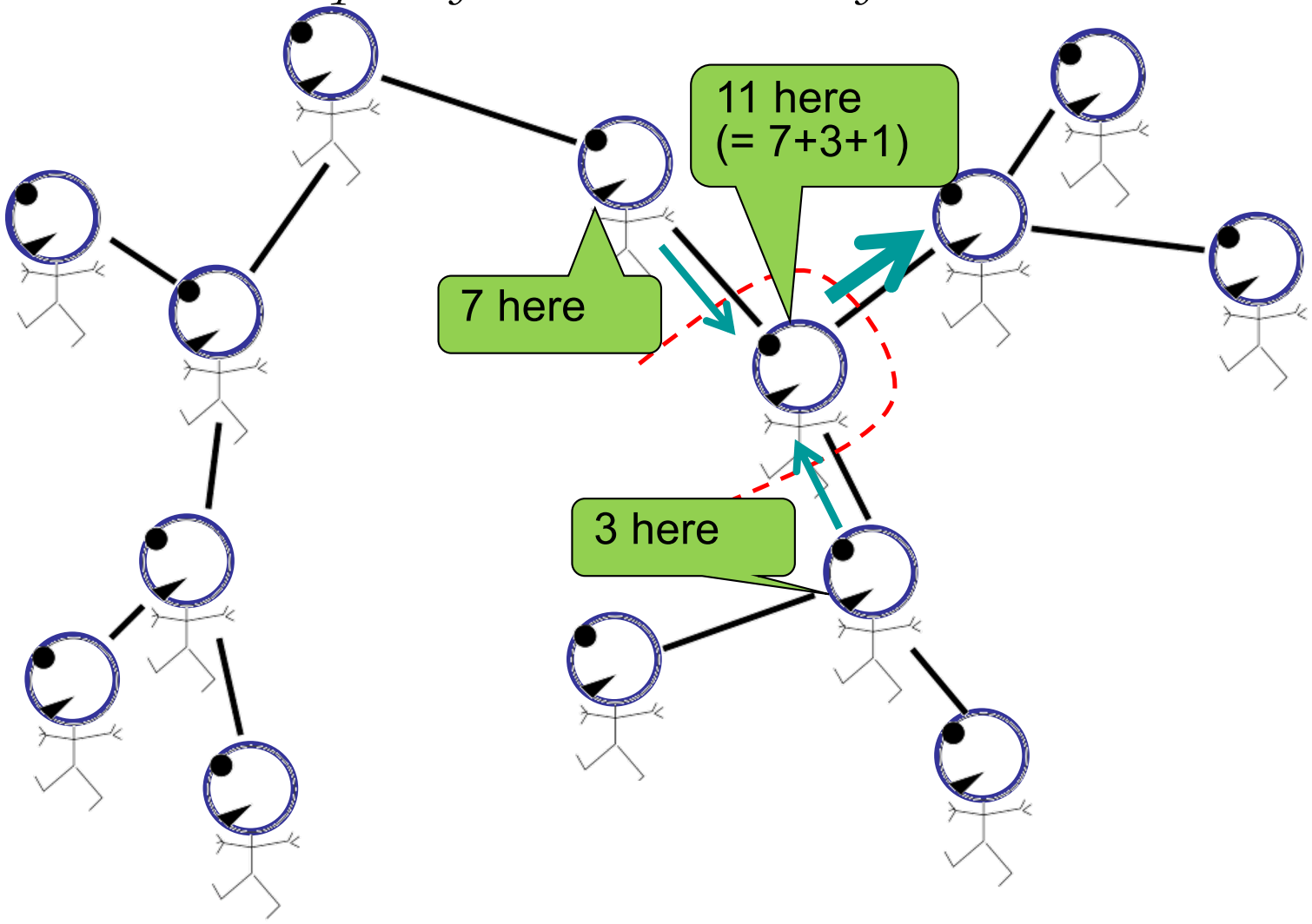
*Each soldier receives reports from all branches of tree*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

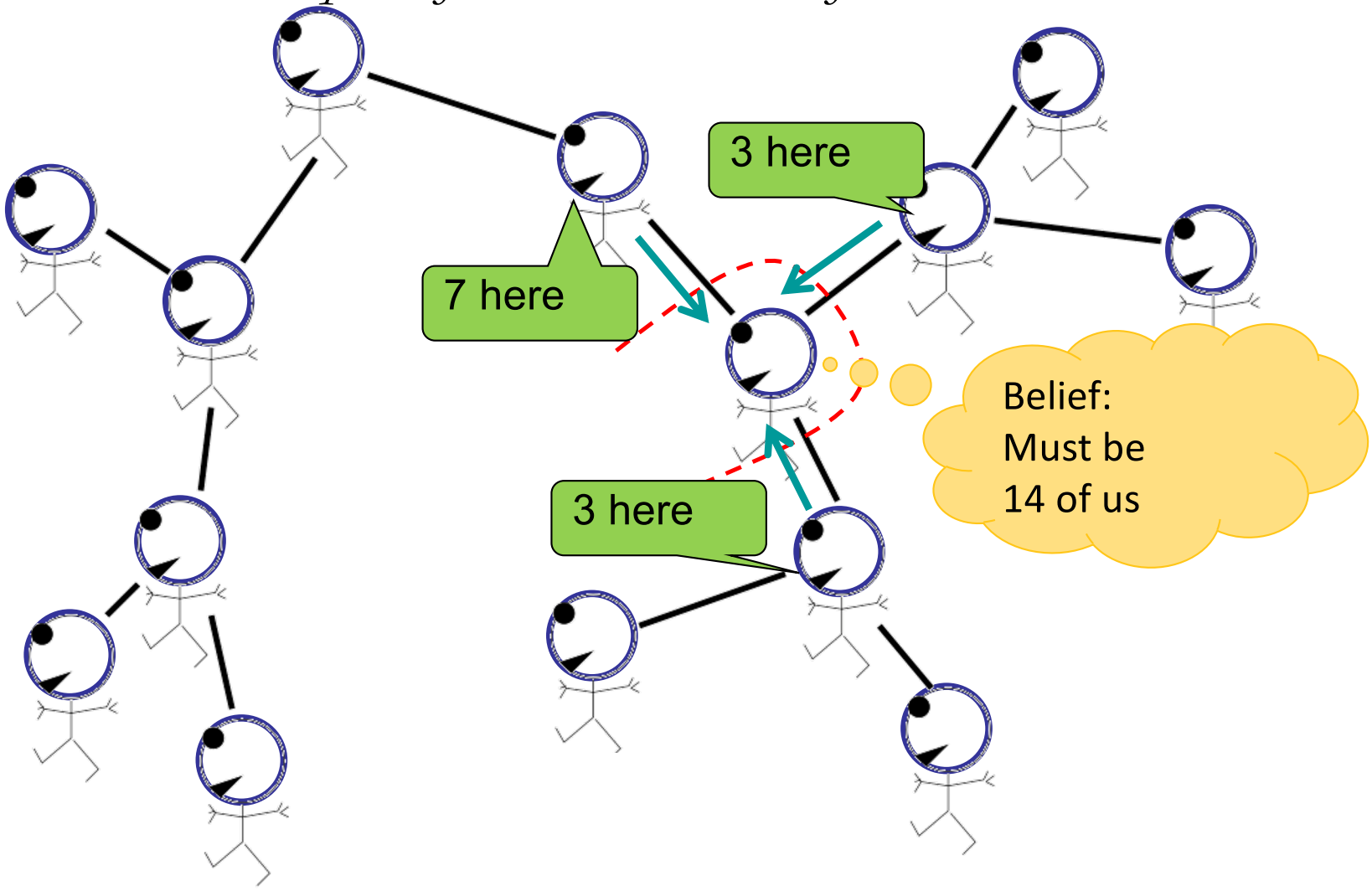
*Each soldier receives reports from all branches of tree*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing

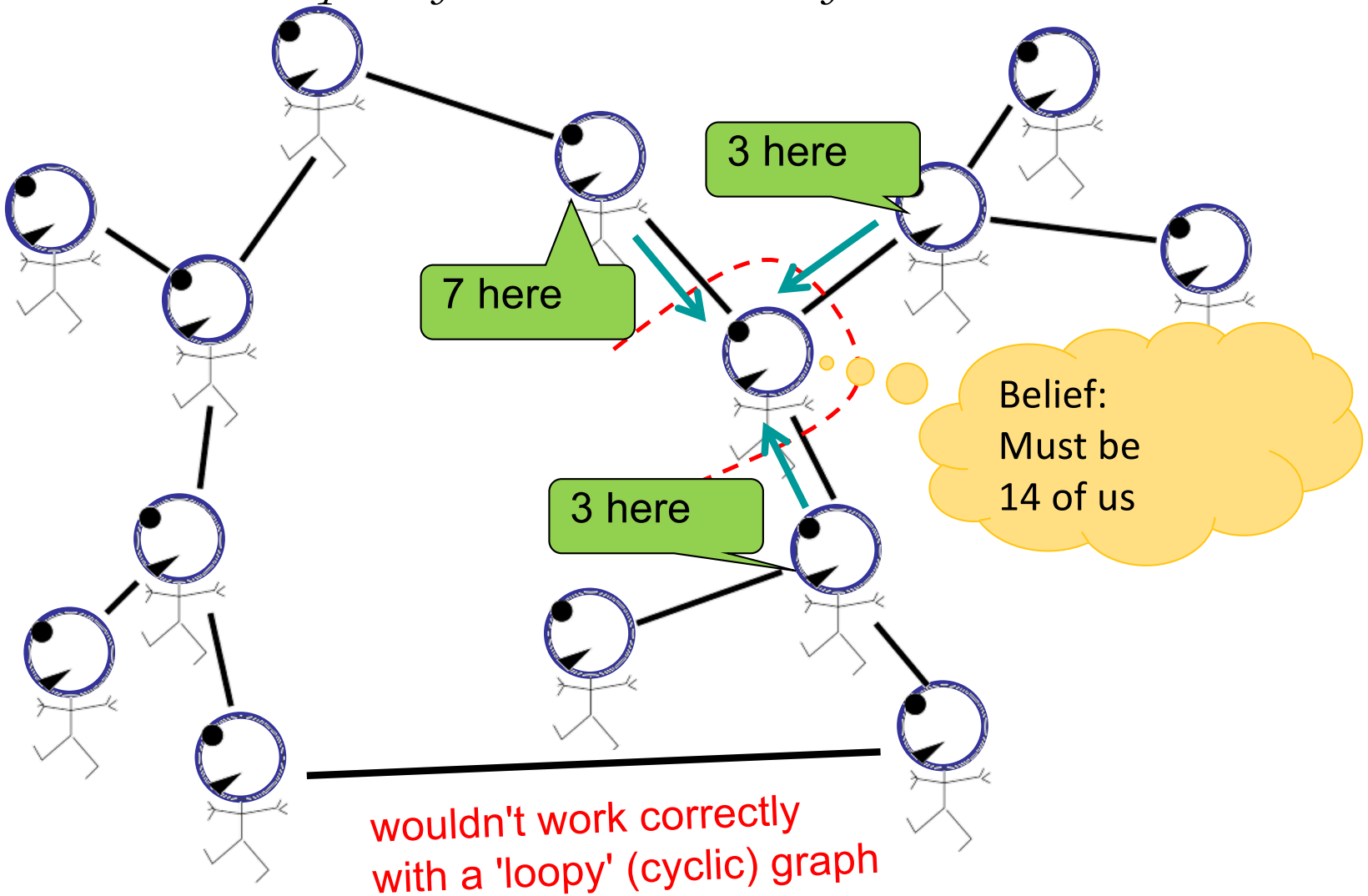
*Each soldier receives reports from all branches of tree*




adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

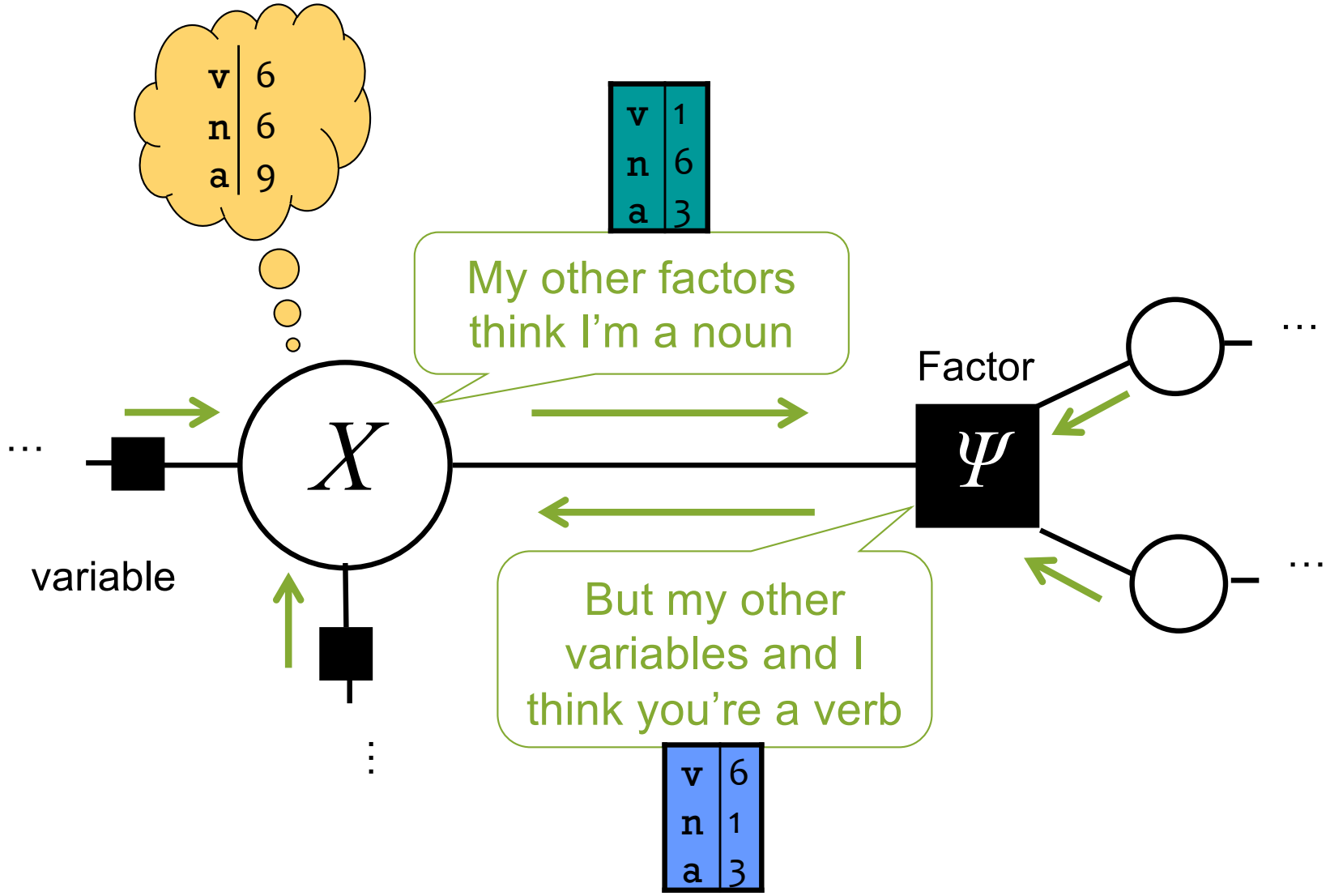
# Message Passing

*Each soldier receives reports from all branches of tree*



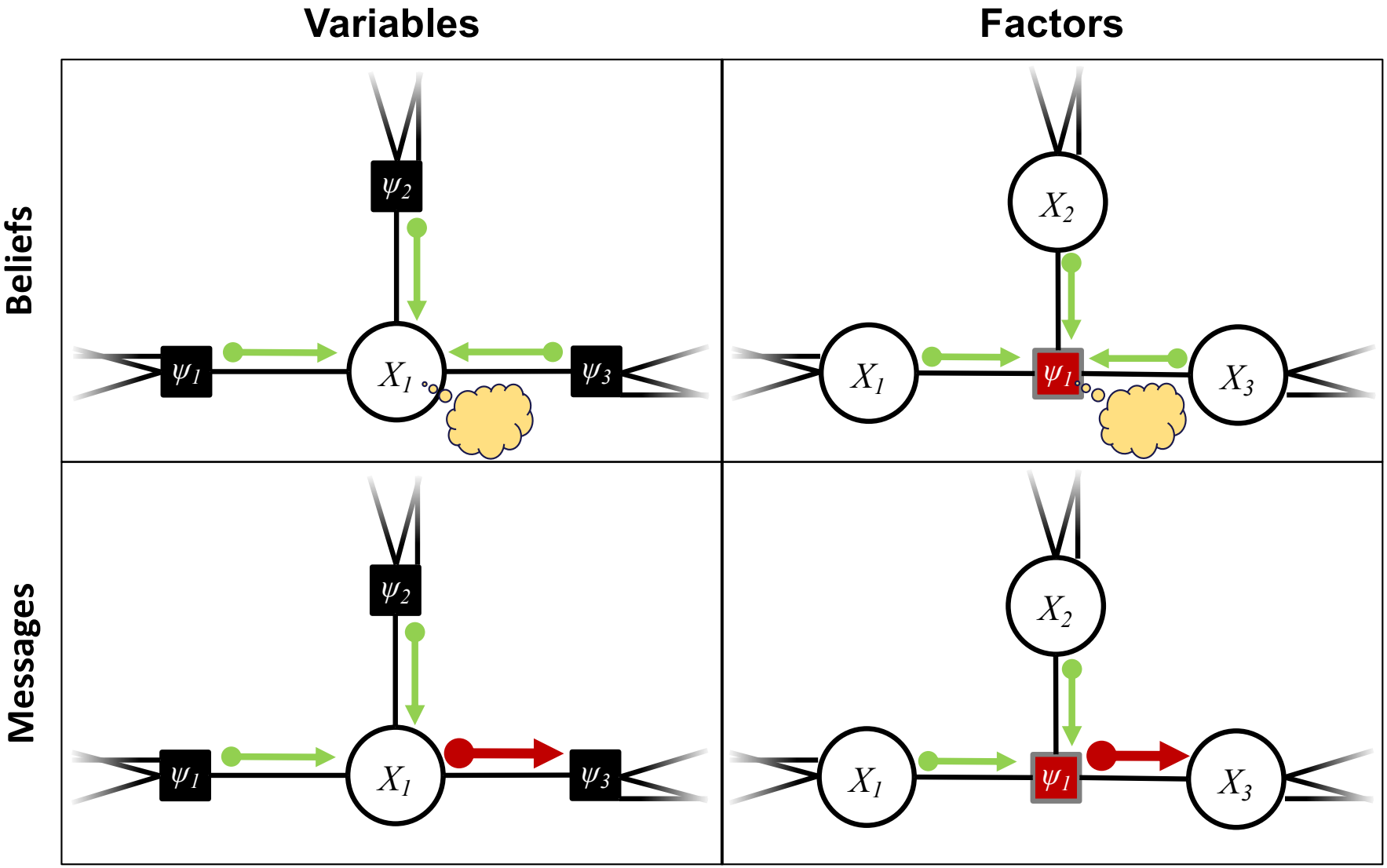

adapted from Matt Gormnley, whi in turn addepted from MacKay (2003) textbook

# Message Passing in Belief Propagation




*Both* of these messages judge the possible values of variable $X$.
Their product = belief at $X$ = product of all 3 messages to $X$.

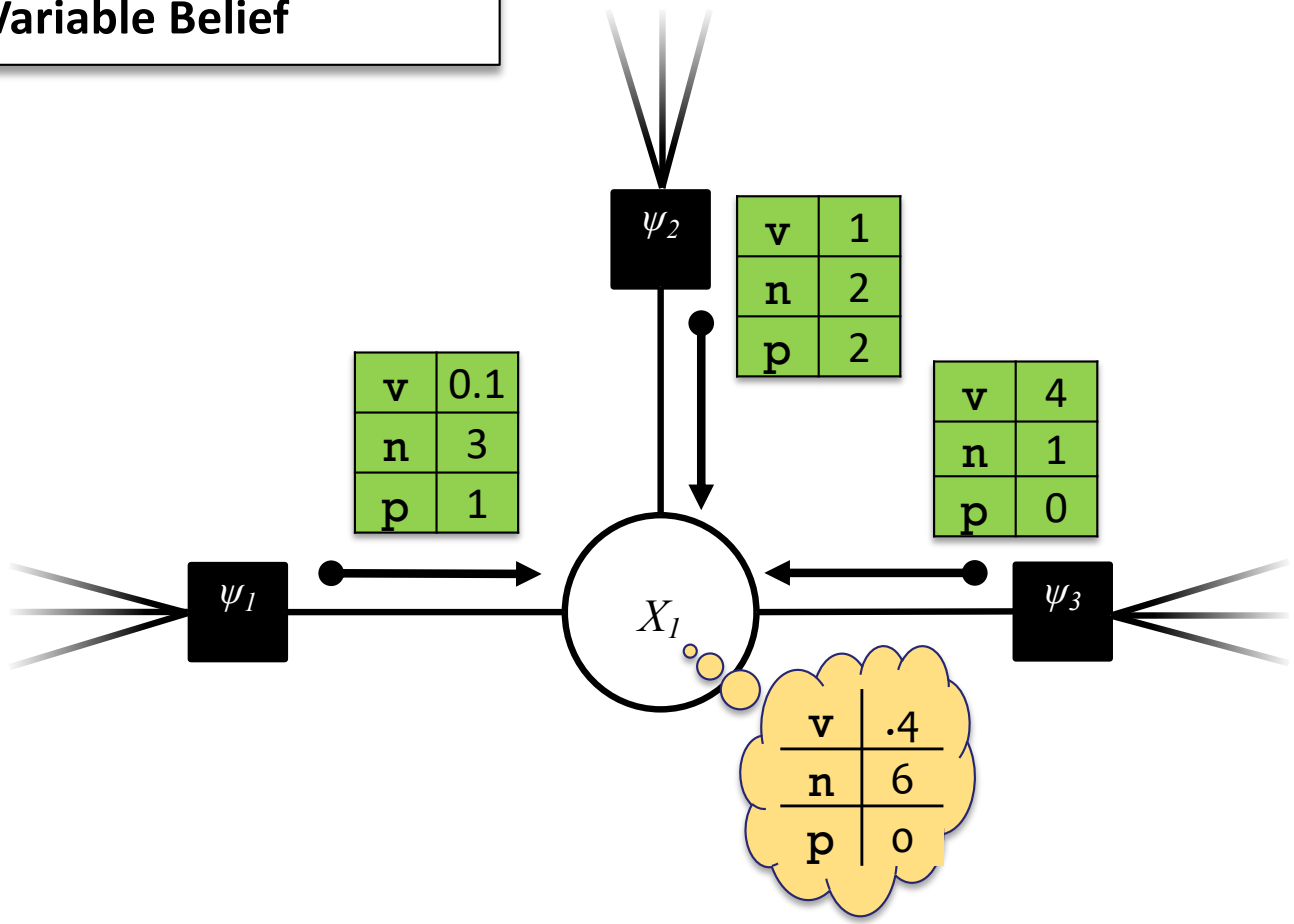
# Sum-Product Belief Propagation



# Sum-Product Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

**Variable Belief**

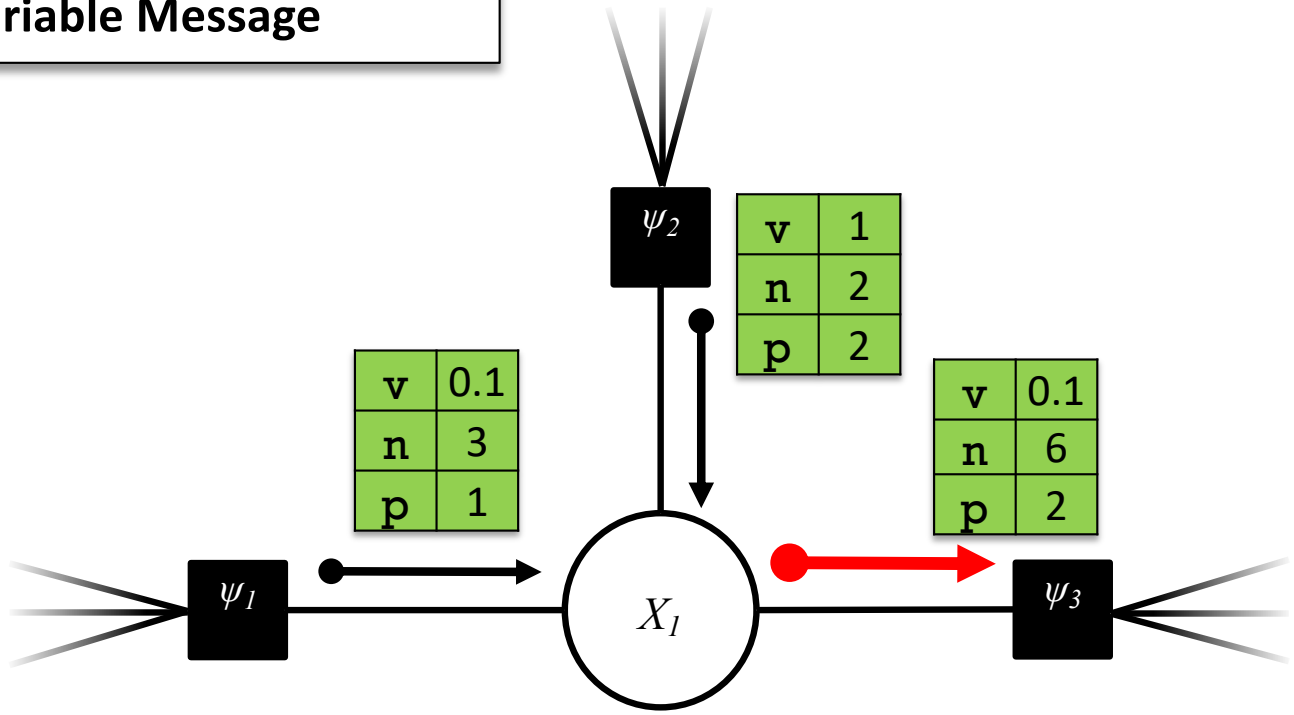



$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

# Sum-Product Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

**Variable Message**

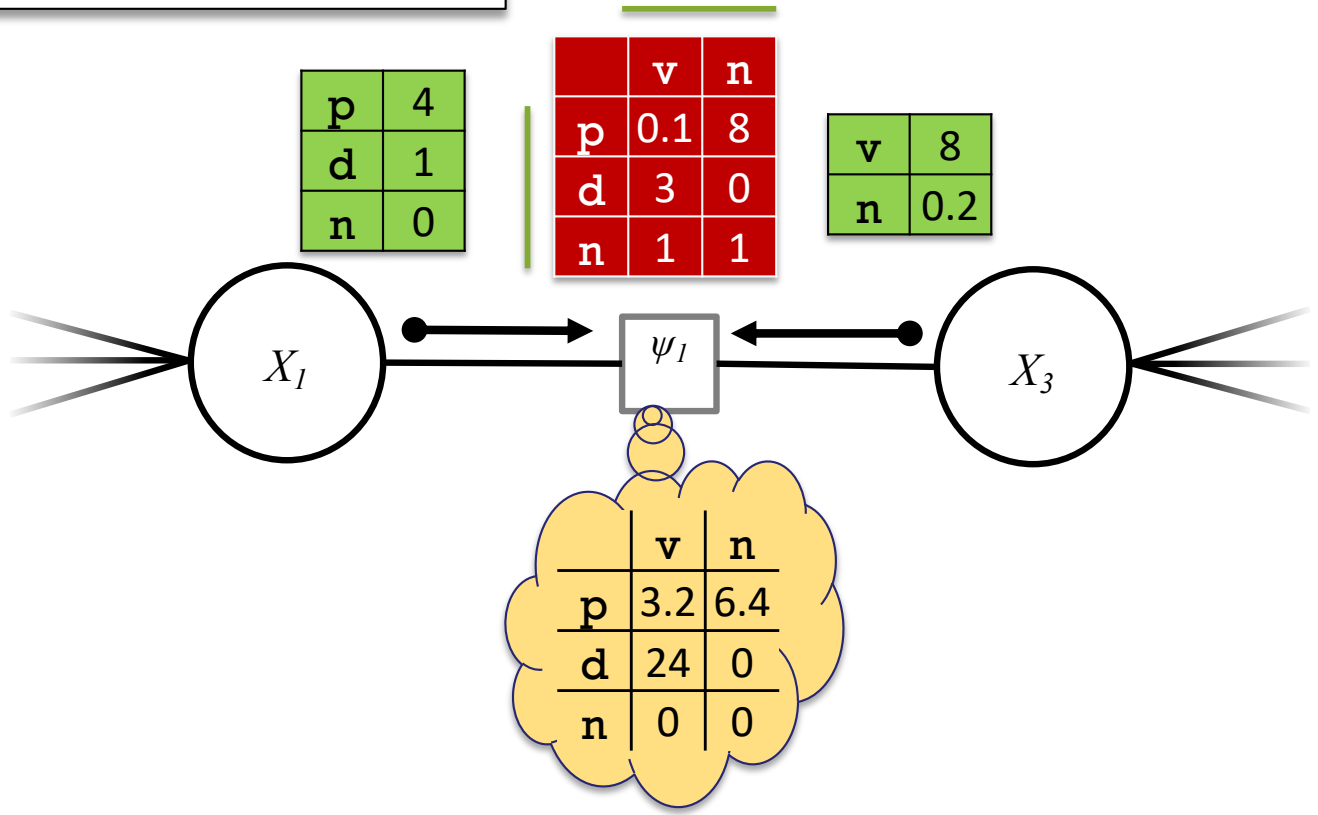



$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

# Sum-Product Belief Propagation

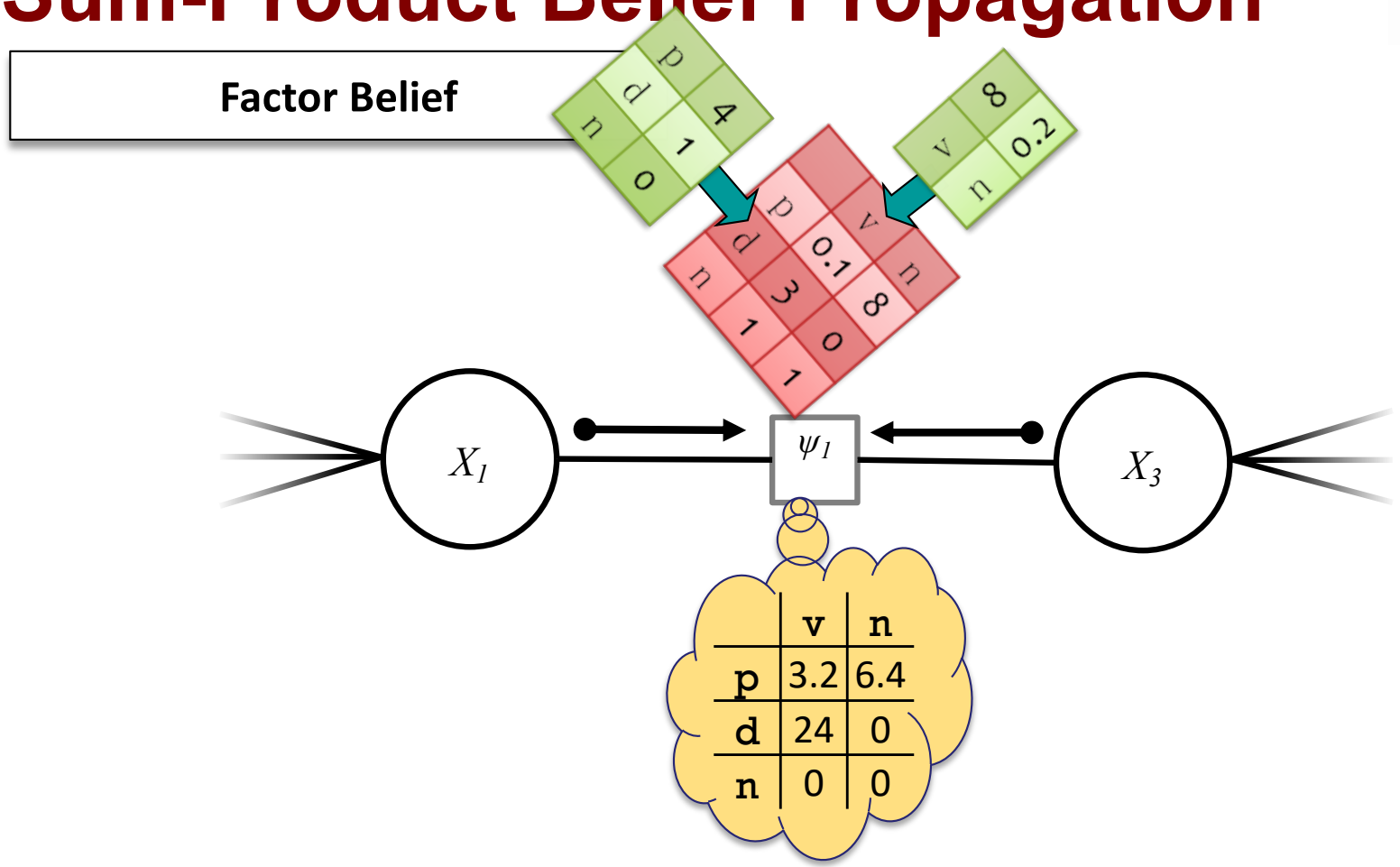
TECHNISCHE UNIVERSITÄT DARMSTADT

**Factor Belief**




$$b_\alpha(\boldsymbol{x_\alpha}) = \psi_\alpha(\boldsymbol{x_\alpha}) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x_\alpha}[i])$$

# Sum-Product Belief Propagation

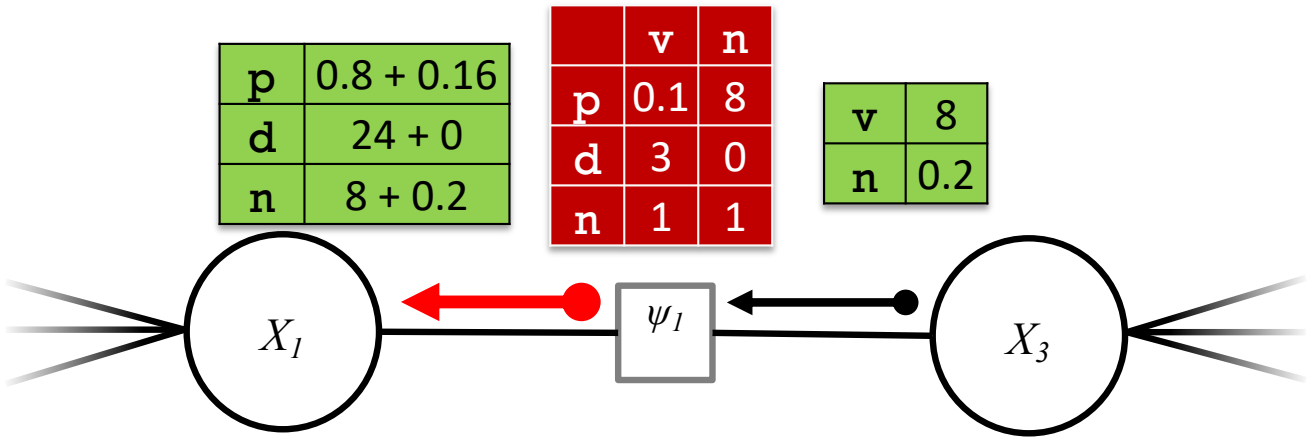



$$b_\alpha(\boldsymbol{x}_\alpha) = \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x}_\alpha[i])$$

# Sum-Product Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

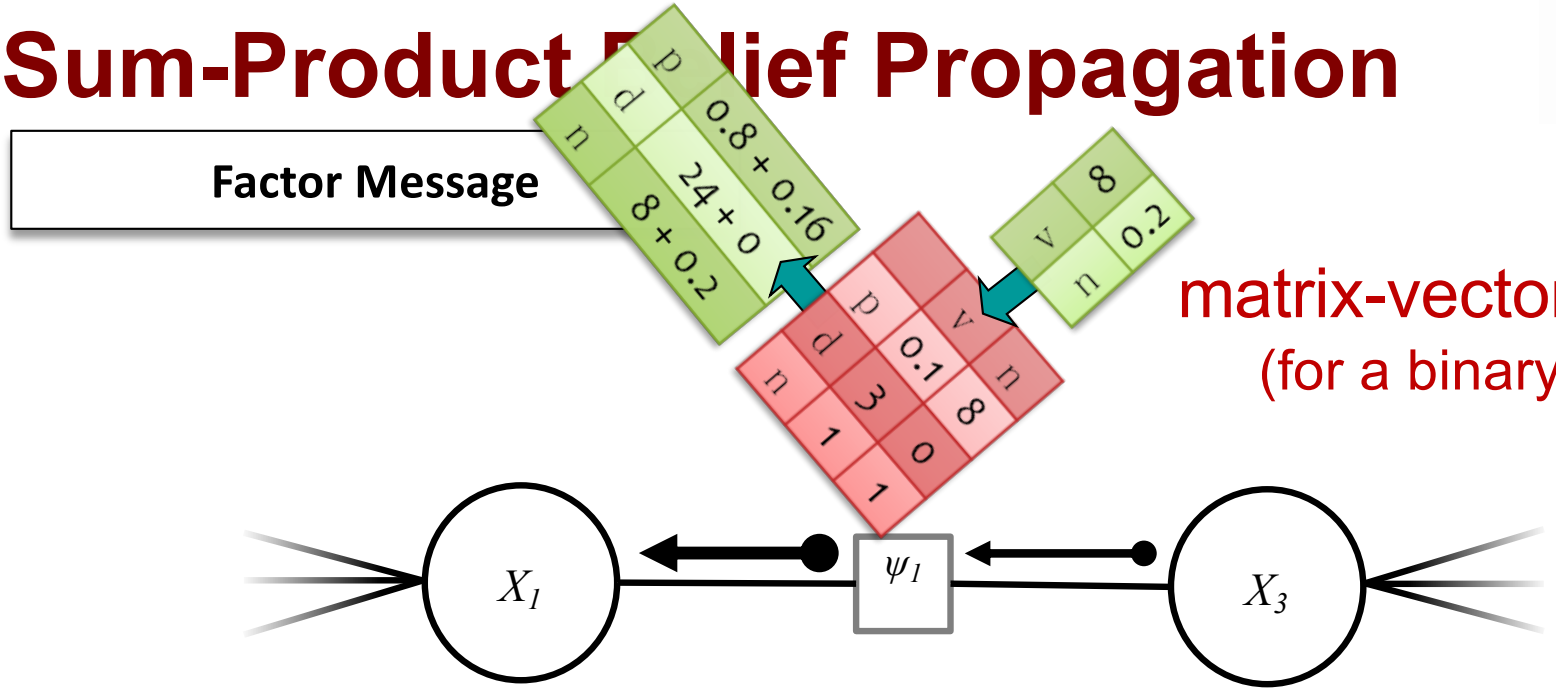
**Factor Message**




$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_\alpha : \boldsymbol{x}_\alpha[i] = x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_\alpha[i])$$

We marginalize of the variable we do not need (like the separators in JT)

# Sum-Product Belief Propagation




matrix-vector product
(for a binary factor)

$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_\alpha : \boldsymbol{x}_\alpha[i] = x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_\alpha[i])$$

# Sum-Product Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

**Input:** a factor graph with no cycles

**Output:** exact marginals for each variable and factor

**Algorithm:**

1. Initialize the messages to the uniform distribution.

$$\mu_{i\to\alpha}(x_i) = 1 \qquad \mu_{\alpha\to i}(x_i) = 1$$

1. Choose a root node.
2. Send messages from the **leaves** to the **root**.
   Send messages from the **root** to the **leaves**.

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i) \qquad \mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

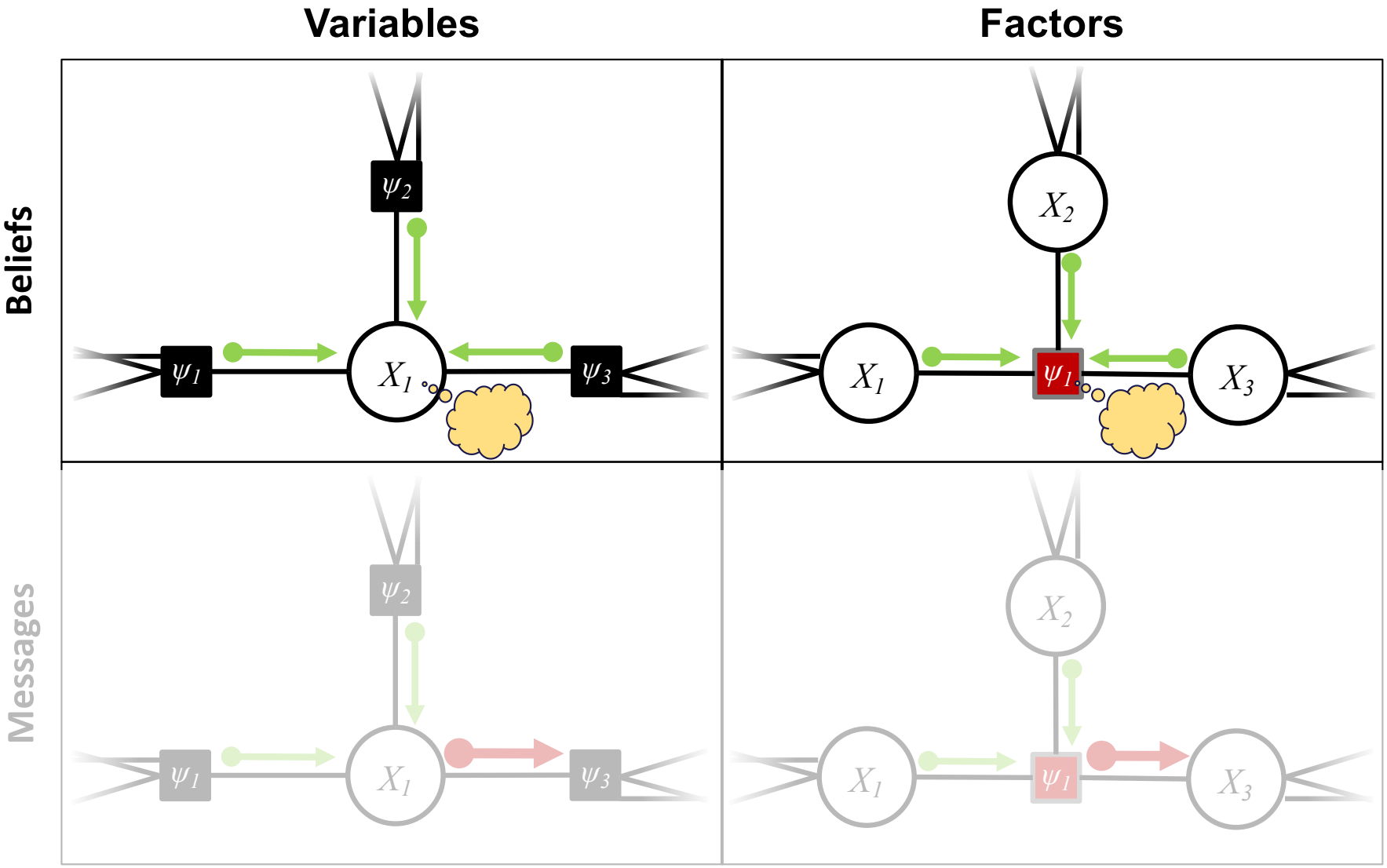
1. Compute the beliefs (unnormalized marginals).

$$b_i(x_i) = \prod_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}(x_i) \qquad b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

2. Normalize beliefs and return the **exact** marginals.

$$p_i(x_i) \propto b_i(x_i) \qquad p_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \propto b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$

# Sum-Product Belief Propagation

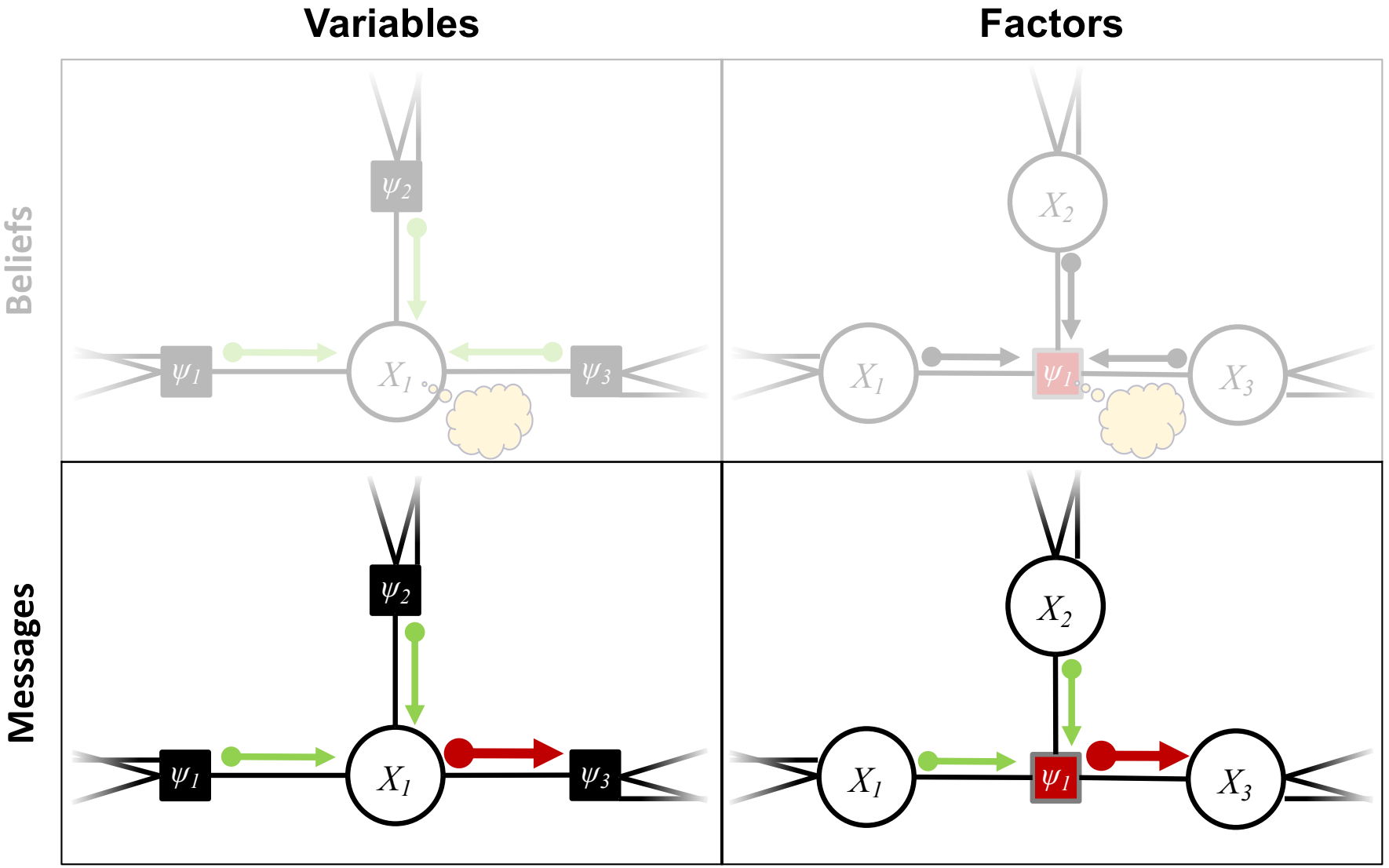



$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

$$b_\alpha(\boldsymbol{x}_\alpha) = \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x}_\alpha[i])$$

# Sum-Product Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT




$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha:\boldsymbol{x}_\alpha[i]=x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_\alpha[i])$$

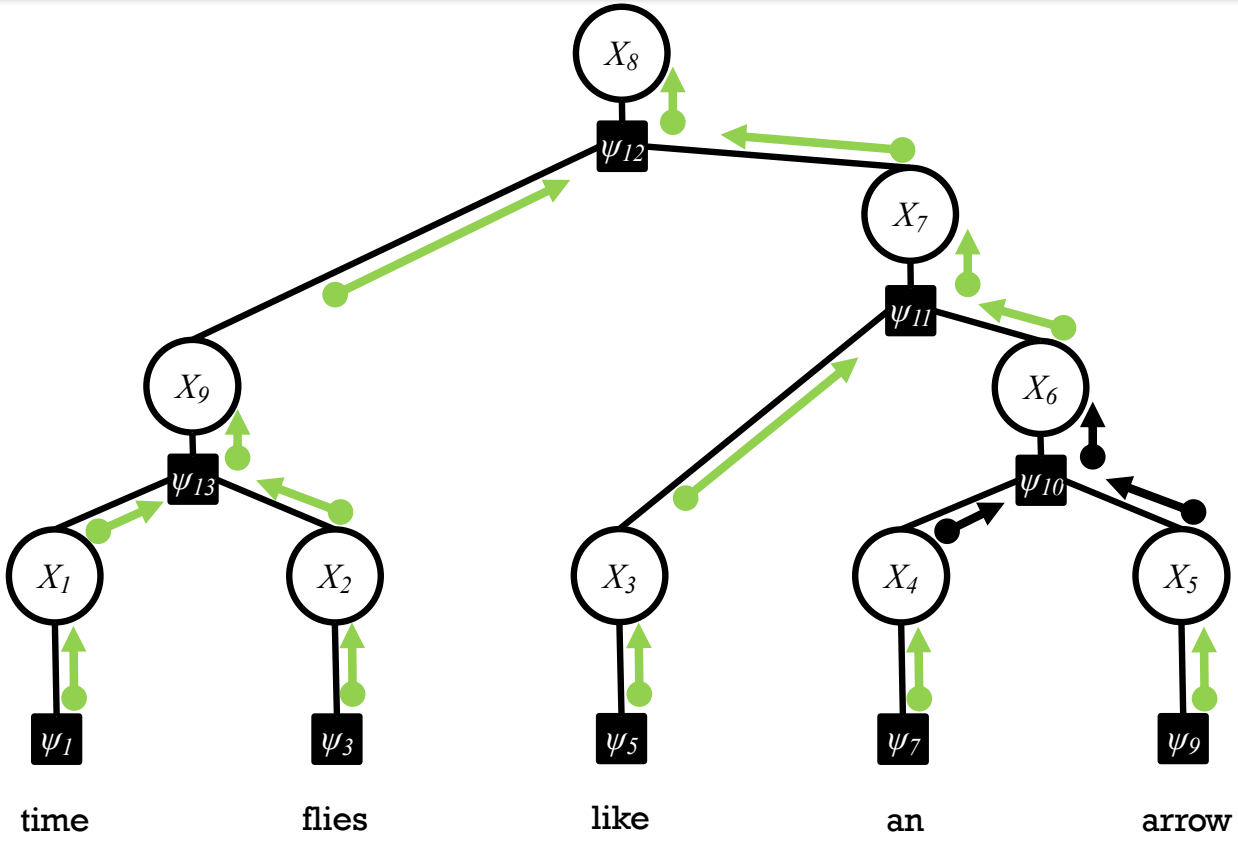
# (Acyclic) Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

In a factor graph with no cycles:

1. Pick any node to serve as the root.
2. Send messages from the **leaves** to the **root**.
3. Send messages from the **root** to the **leaves**.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.



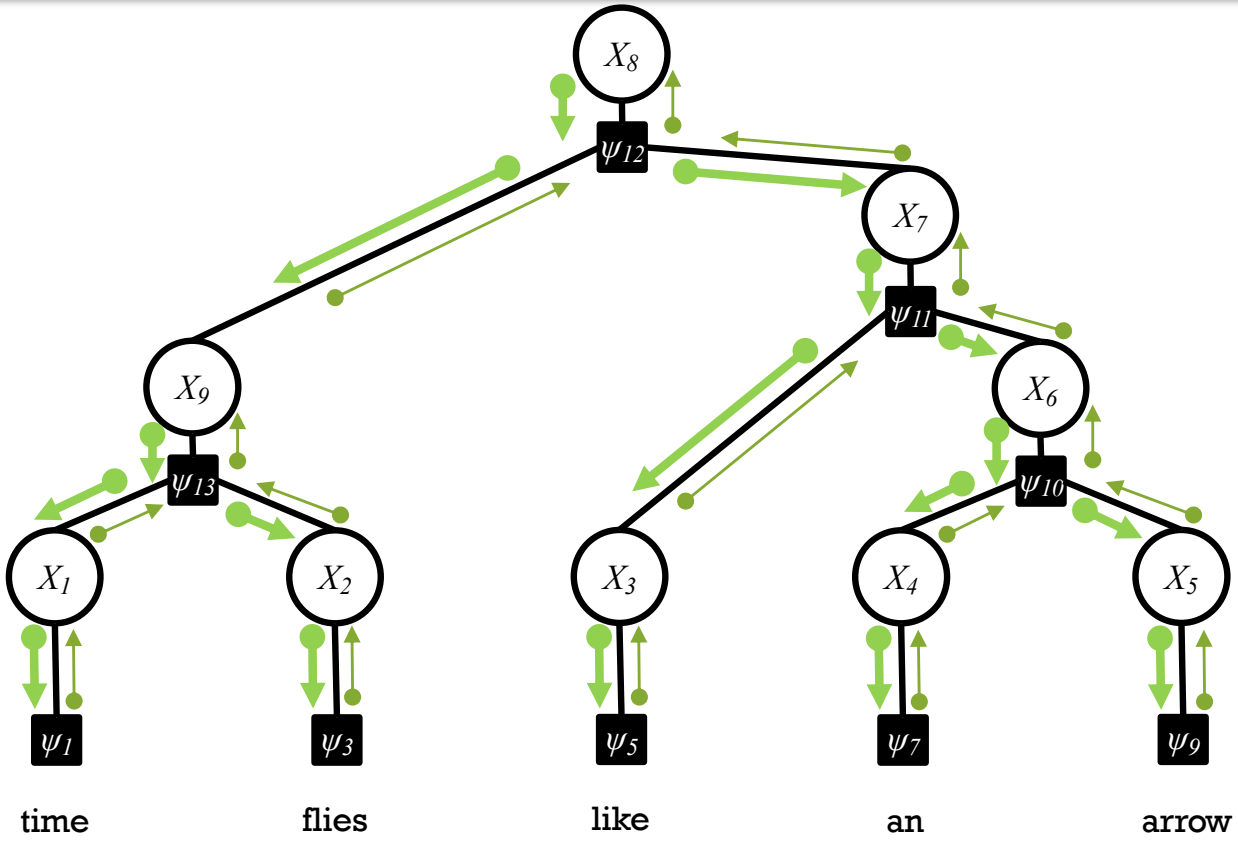
# (Acyclic) Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

In a factor graph with no cycles:

1. Pick any node to serve as the root.
2. Send messages from the **leaves** to the **root**.
3. Send messages from the **root** to the **leaves**.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.

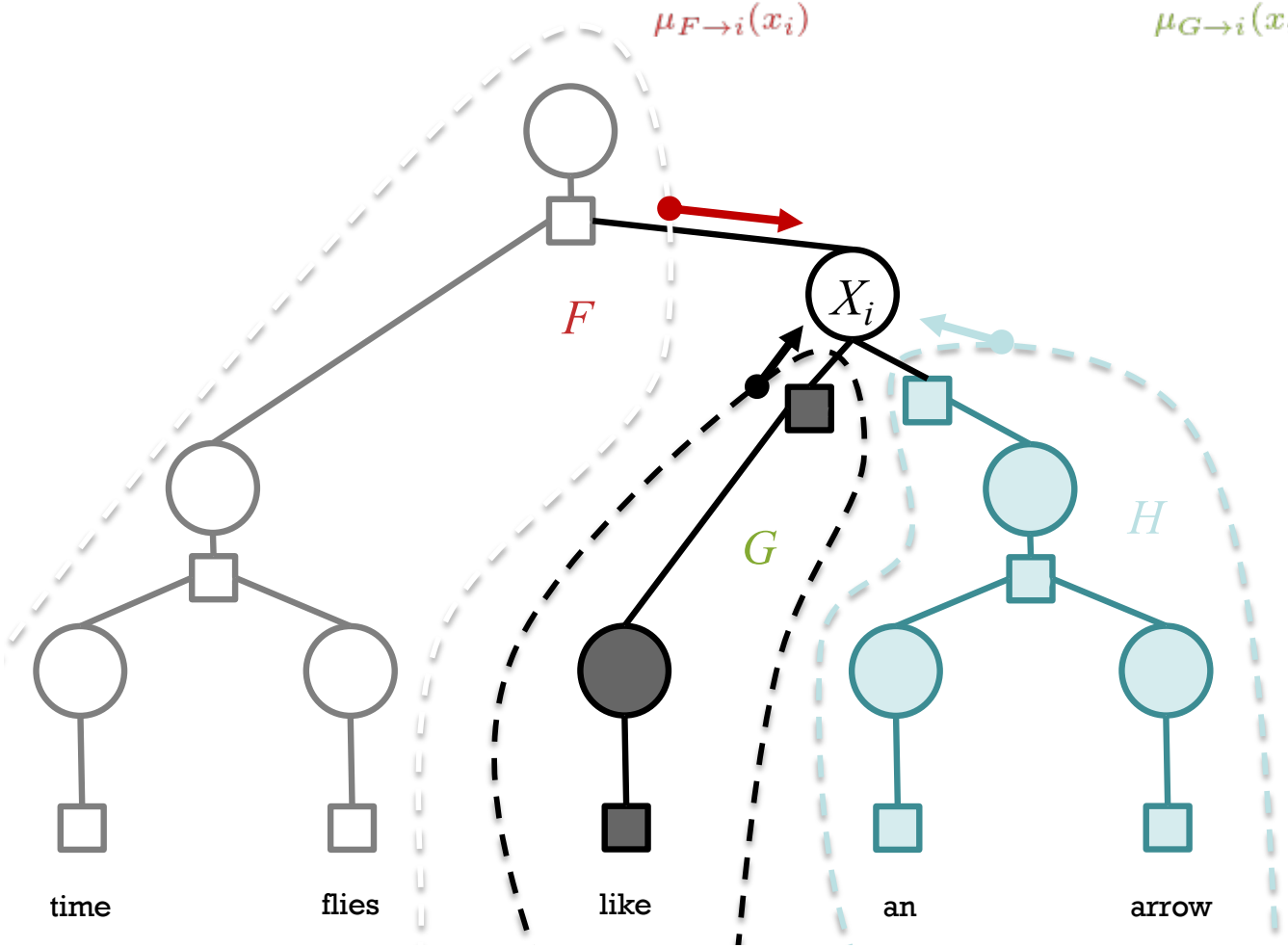


# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$




**Subproblem:**
Inference using just the factors in subgraph $H$

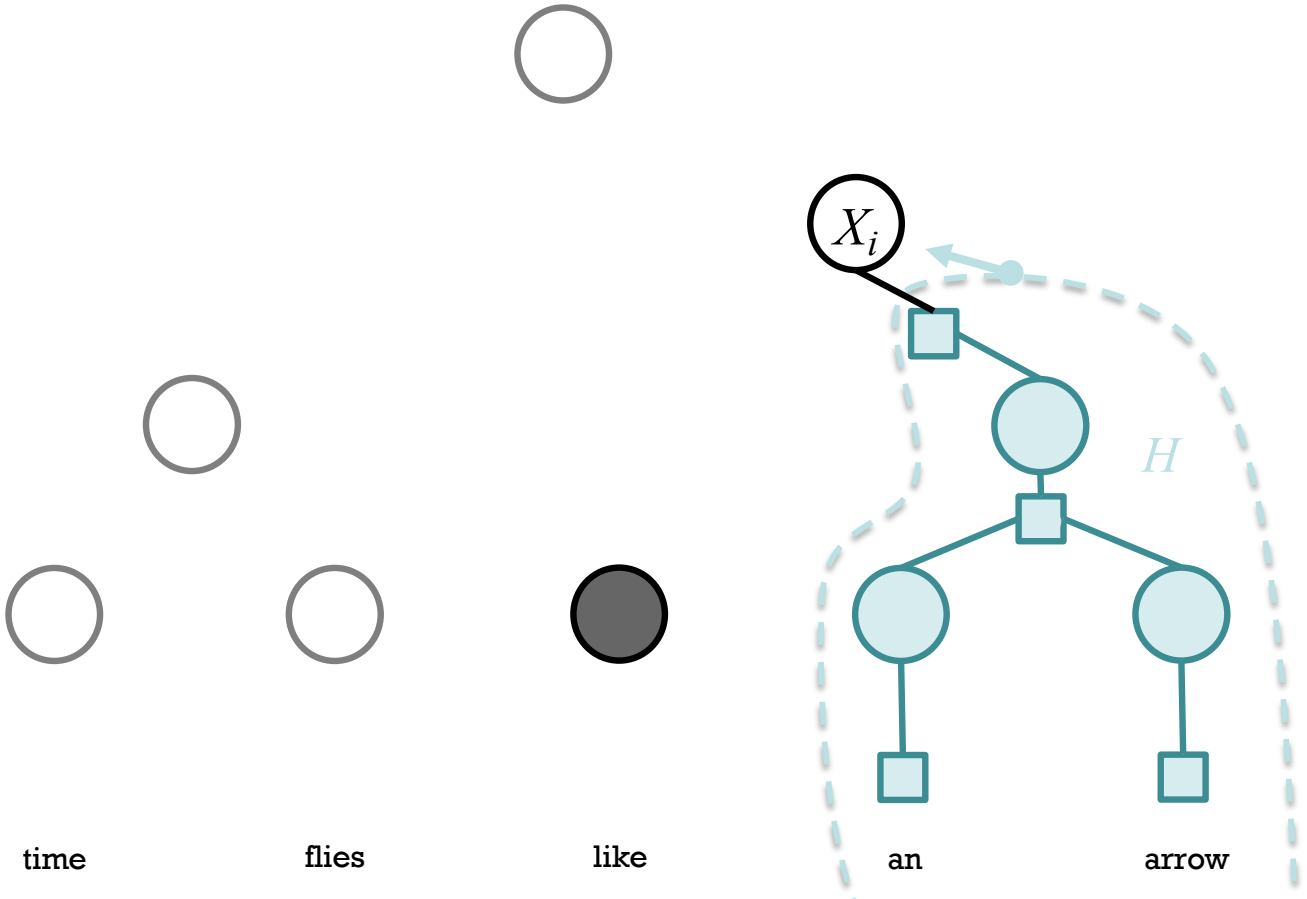
Figure adapted from Burkett & Klein (2012)

# Acyclic BP as Dynamic Programming



$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$




**Subproblem:**
Inference using just the factors in subgraph $H$

The marginal of $X_i$ in that smaller model is the message sent to $X_i$ from subgraph $H$

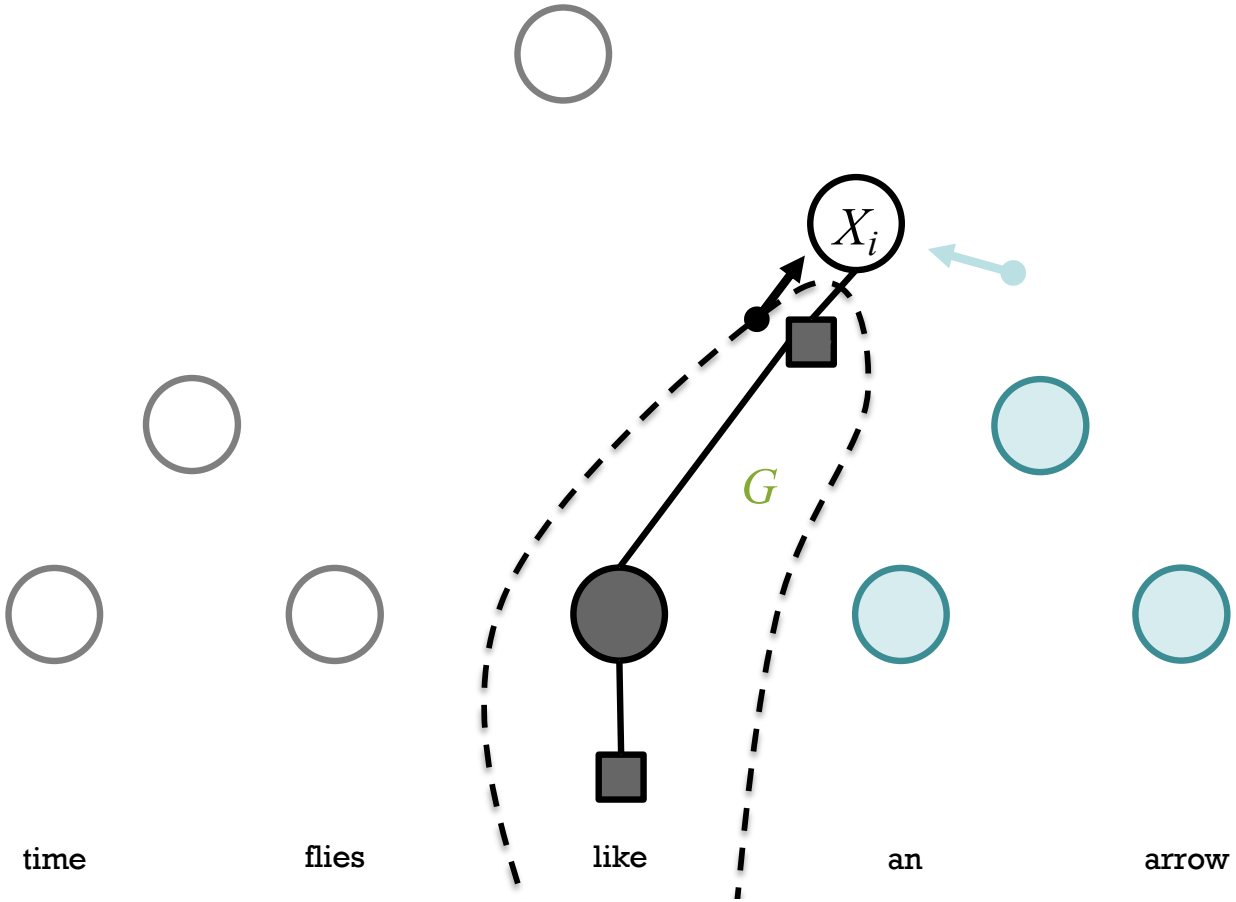
*Message **to** a variable*

# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$




**Subproblem:**
Inference using just the factors in subgraph $H$

The marginal of $X_i$ in that smaller model is the message sent to $X_i$ from subgraph $H$

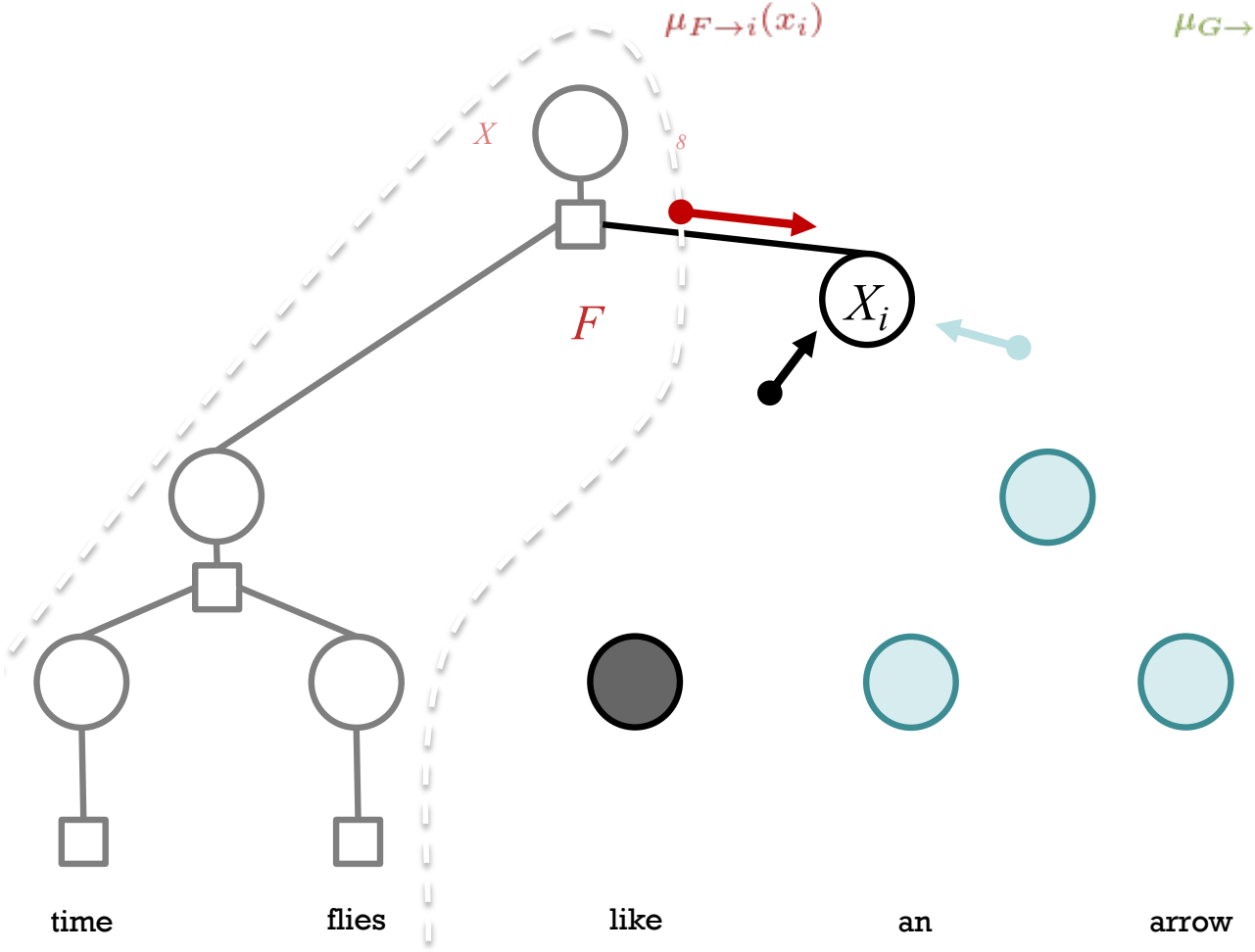
*Message **to** a variable*

# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$




**Subproblem:**
Inference using just the factors in subgraph $H$

The marginal of $X_i$ in that smaller model is the message sent to $X_i$ from subgraph $H$

*Message **to** a variable*

# Acyclic BP as Dynamic Programming

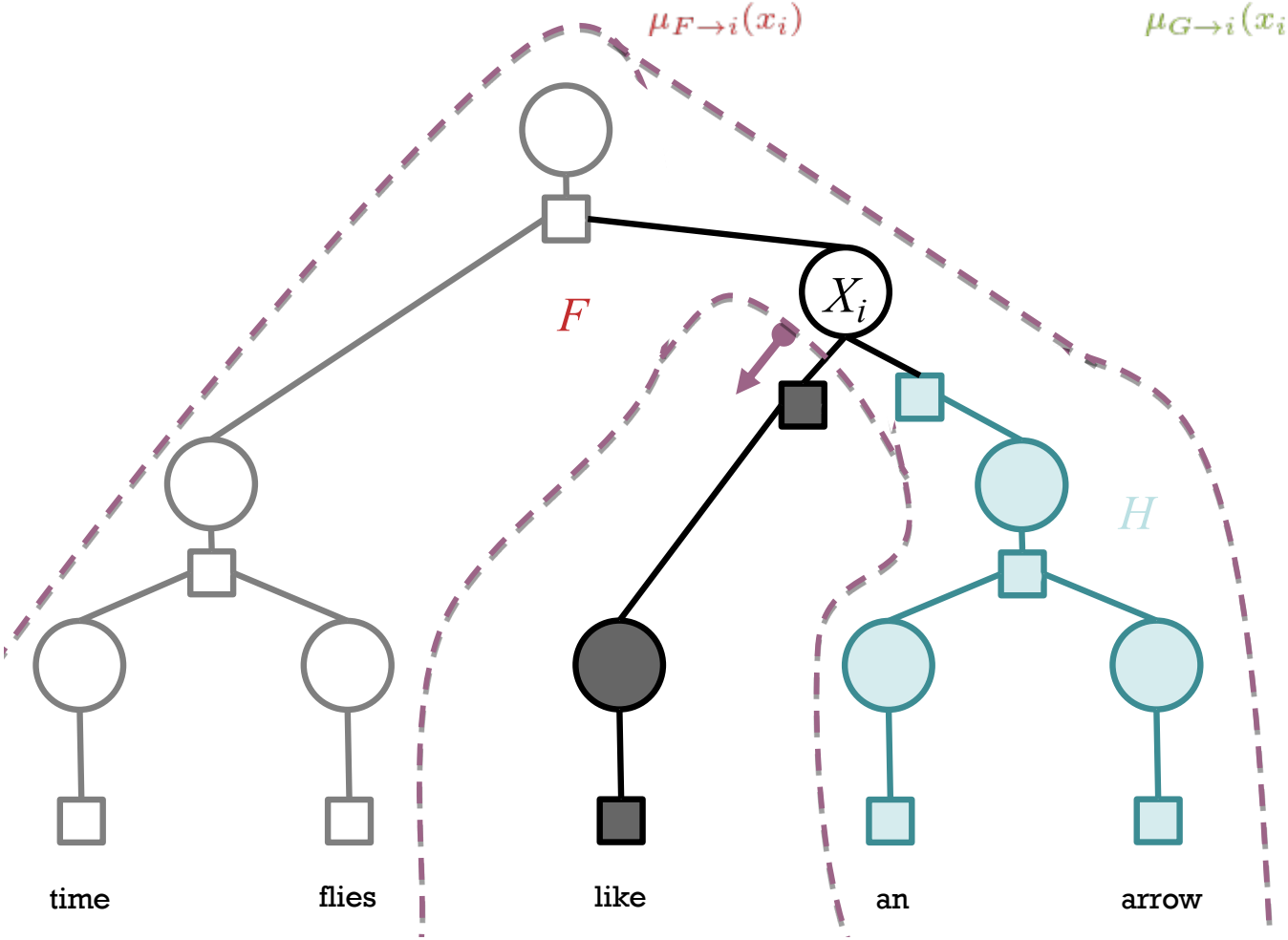
TECHNISCHE UNIVERSITÄT DARMSTADT

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

Multiplying together the red term and the blue term

$$= \left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right) \left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right) \left( \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)$$

$$= \mu_{F \to i}(x_i) \quad \mu_{G \to i}(x_i) \quad \mu_{H \to i}(x_i)$$



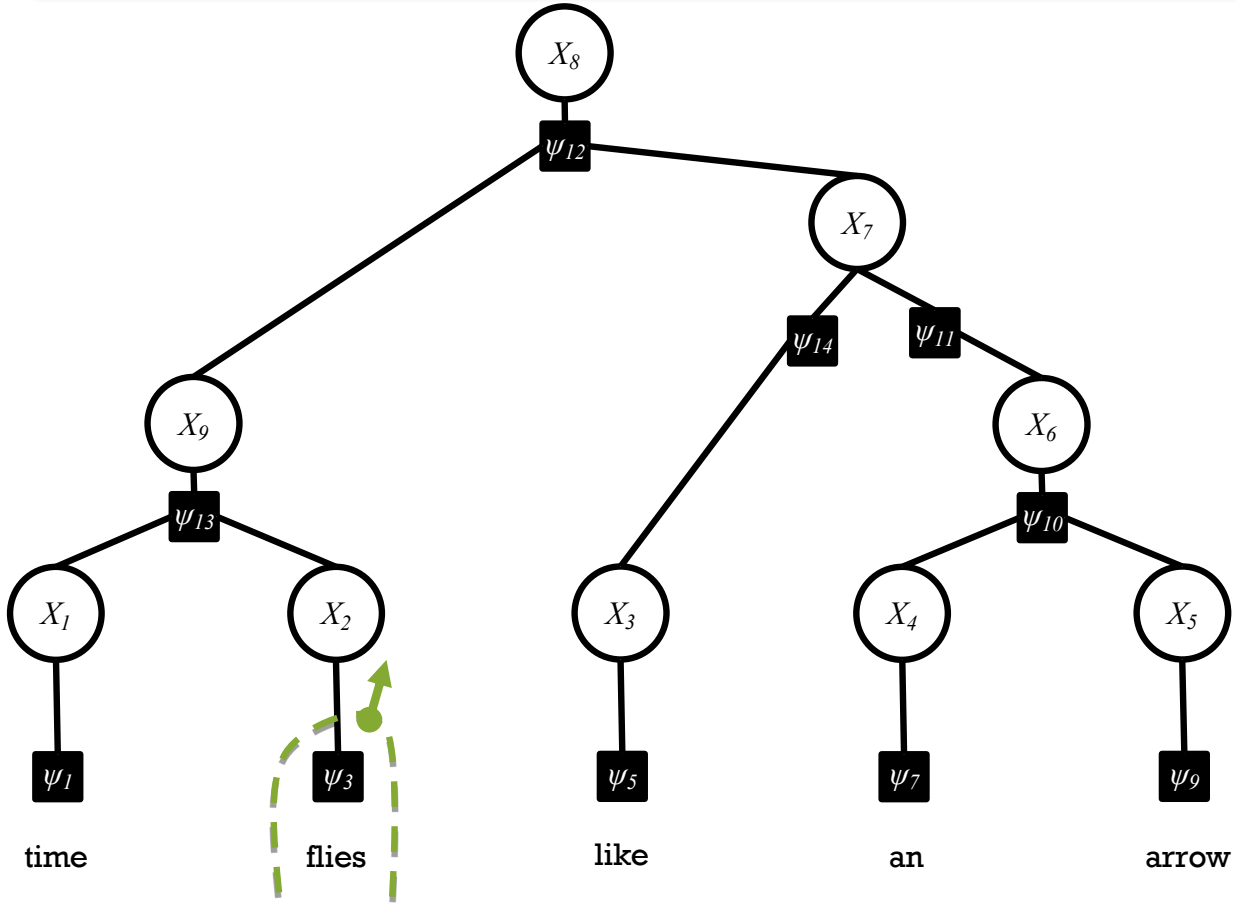

**Subproblem:**
Inference using just the factors in subgraph $F \cup H$
The marginal of $X_i$ in that smaller model is the message sent by $X_i$ out of subgraph $F \cup H$

*Message **from** a variable*

# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

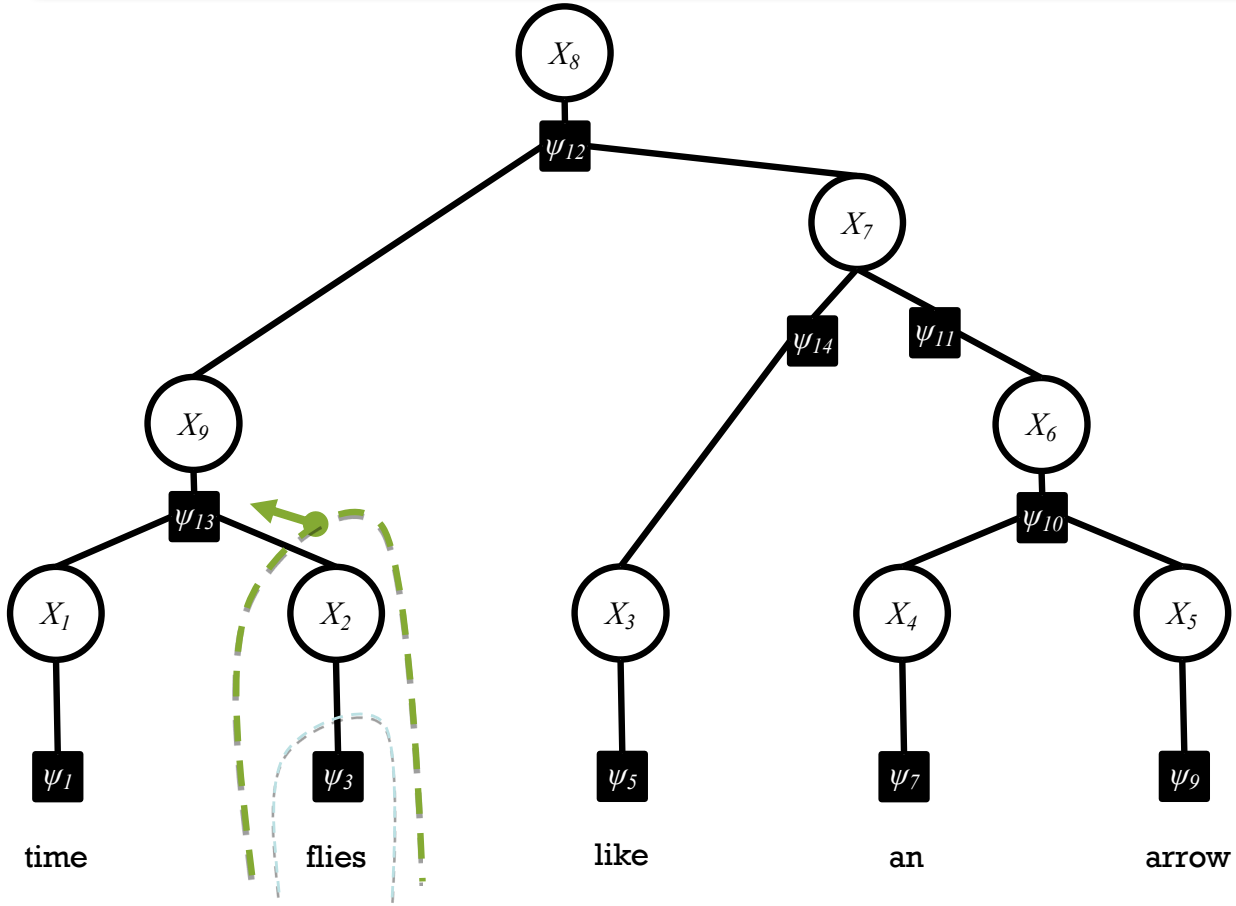
- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



# Acyclic BP as Dynamic Programming

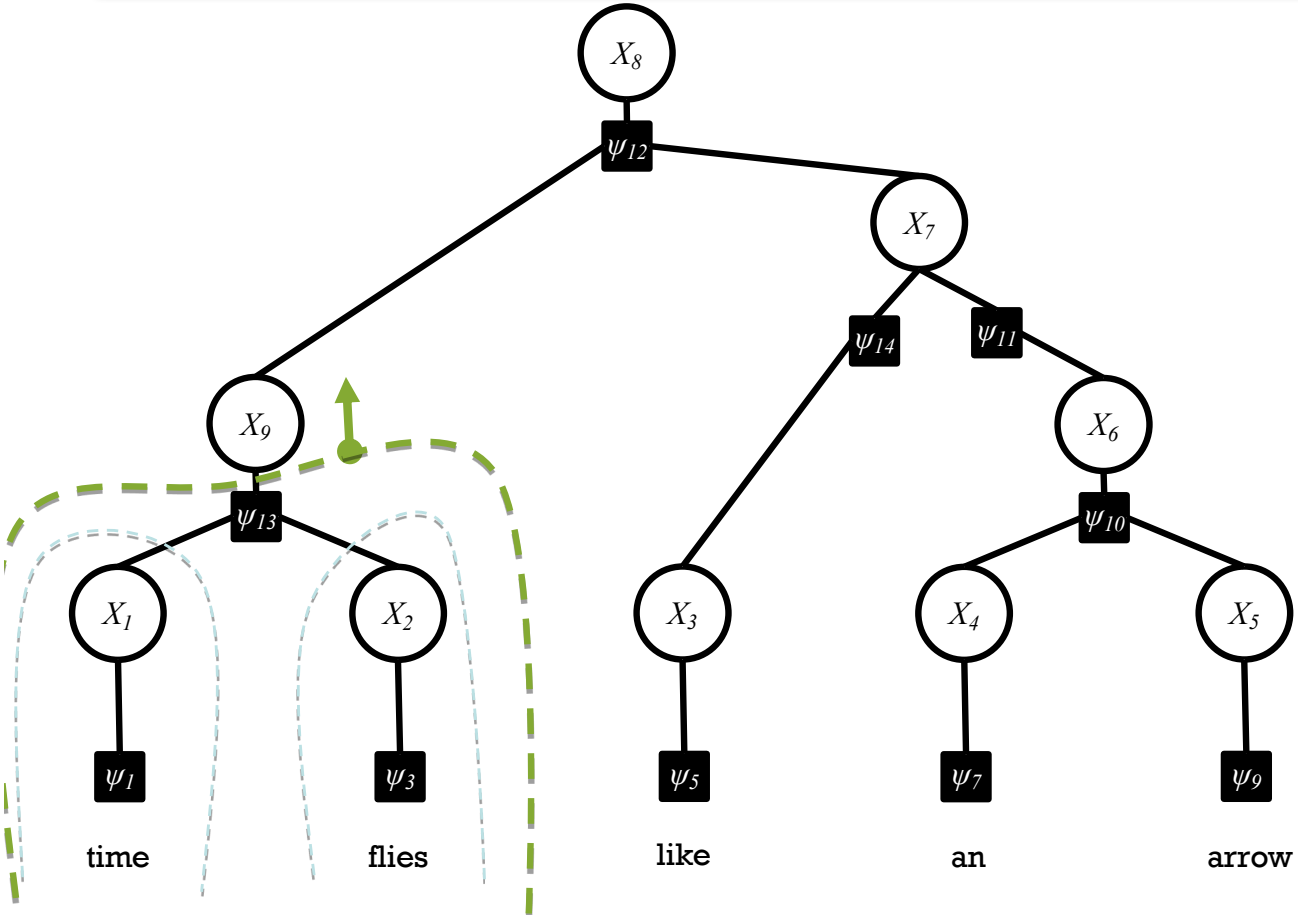
TECHNISCHE UNIVERSITÄT DARMSTADT

- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



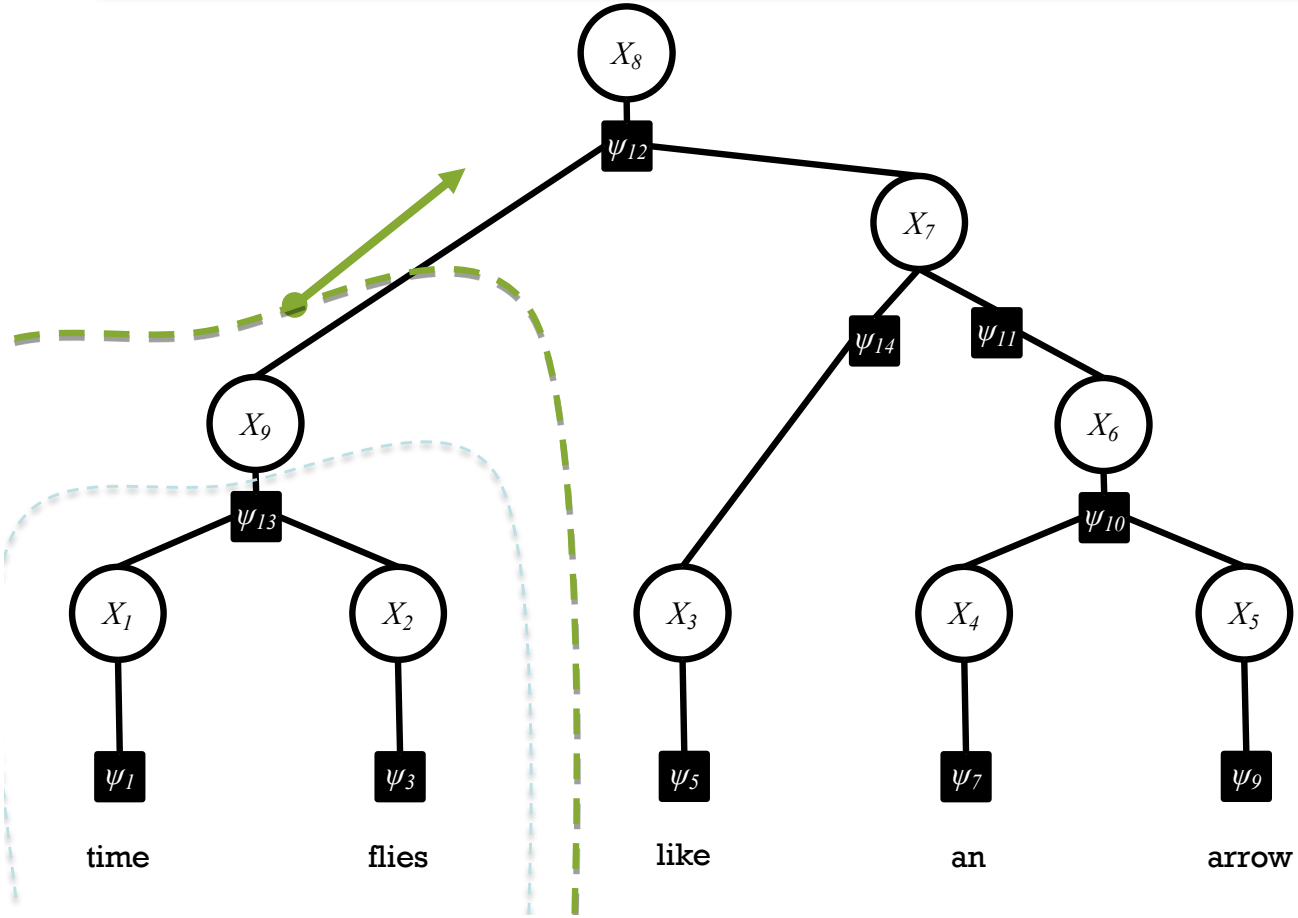
# Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



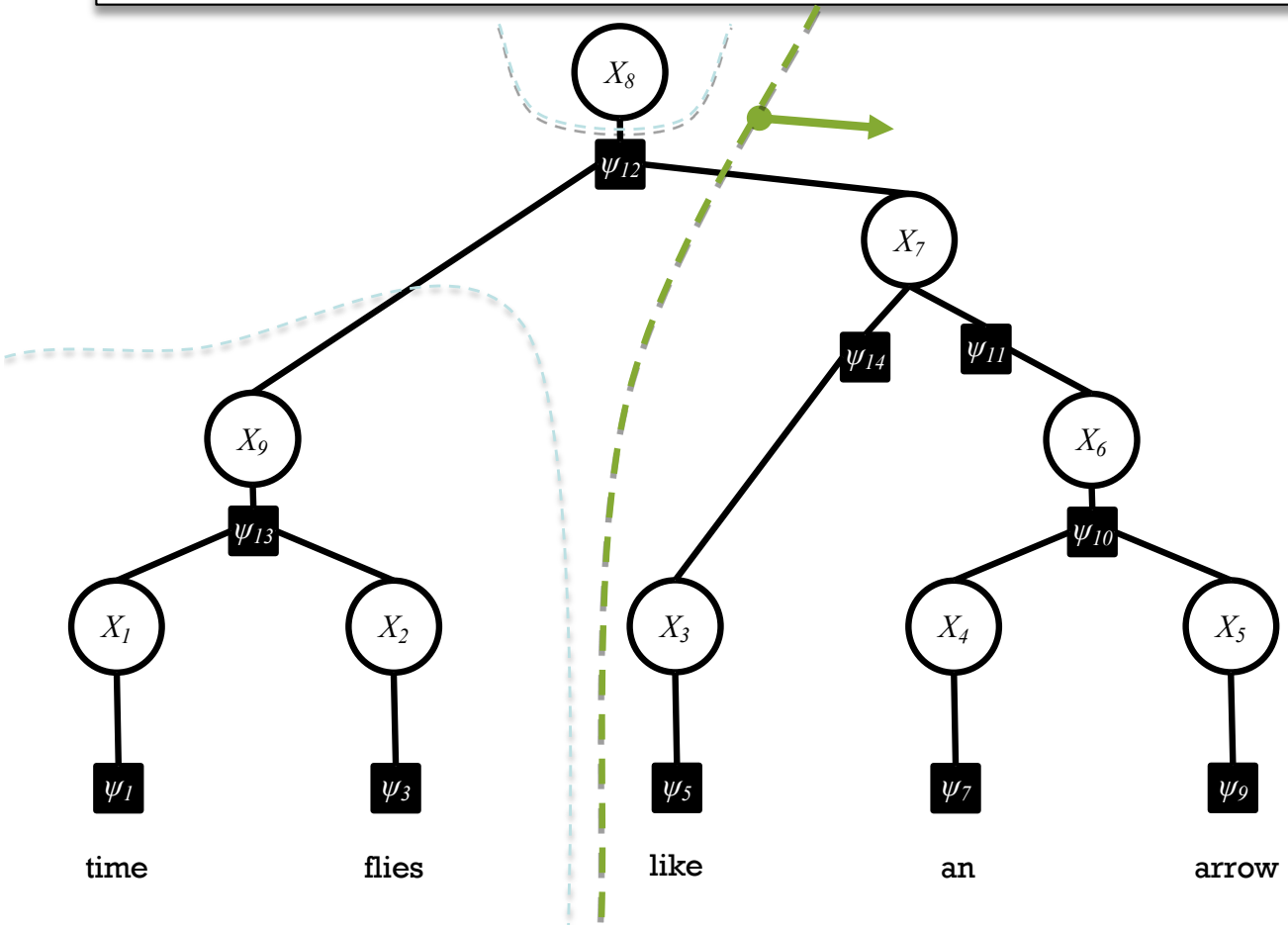
# Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



# Acyclic BP as Dynamic Programming

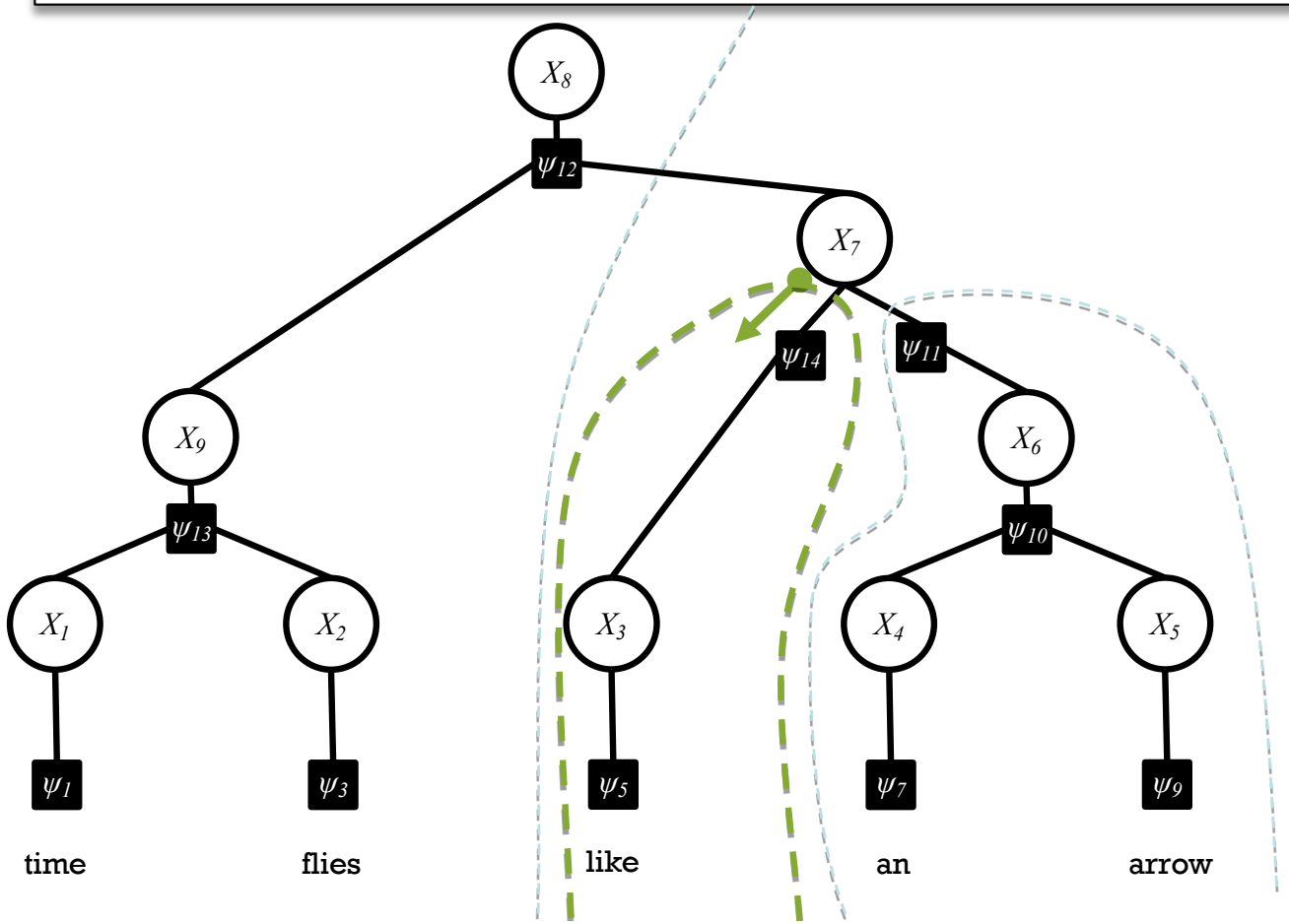
- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

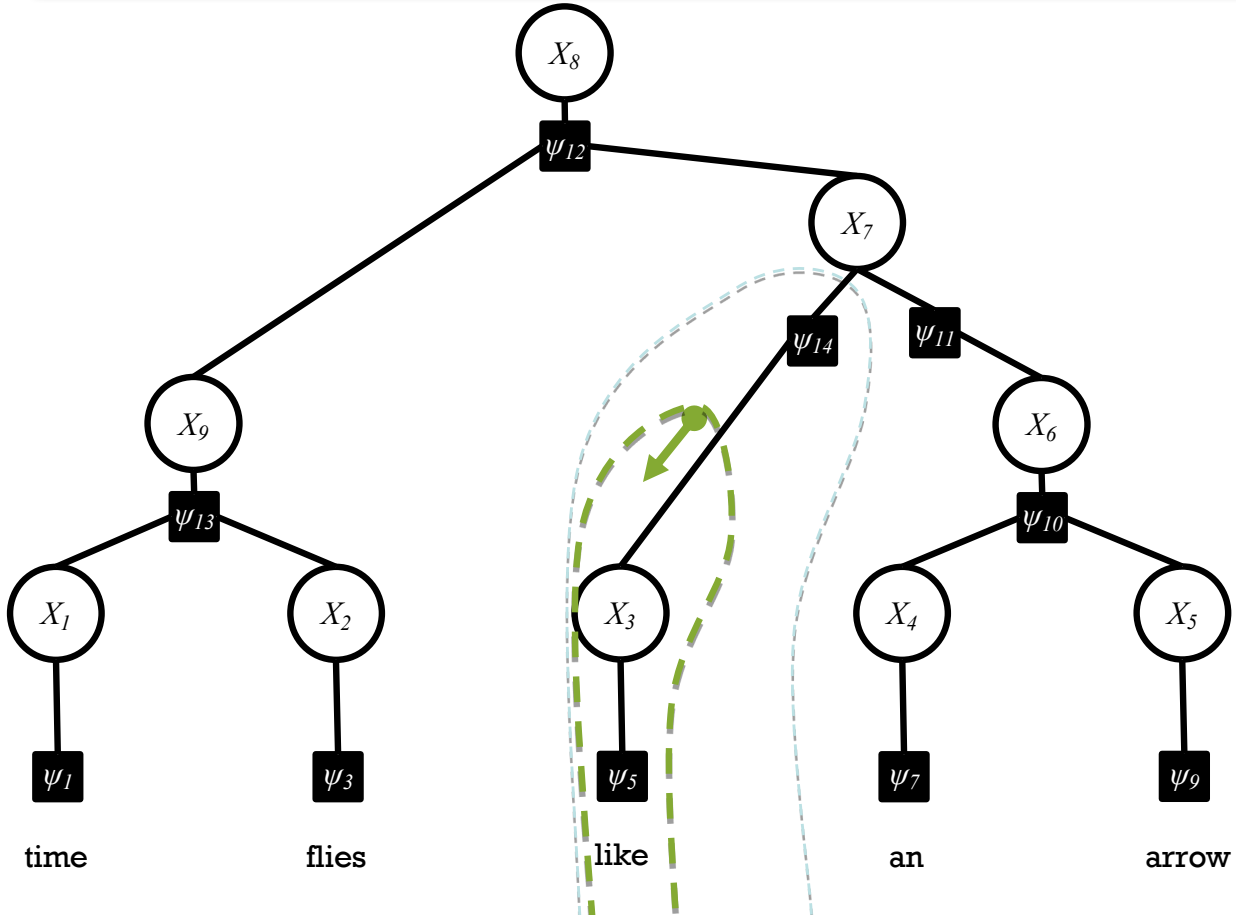
- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



# Acyclic BP as Dynamic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

- If you want the **marginal** $p_i(x_i)$ where $X_i$ has degree $k$, you can think of that summation as a **product of $k$ marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



# Acyclic BP = Variable Elimination + dynamic programming

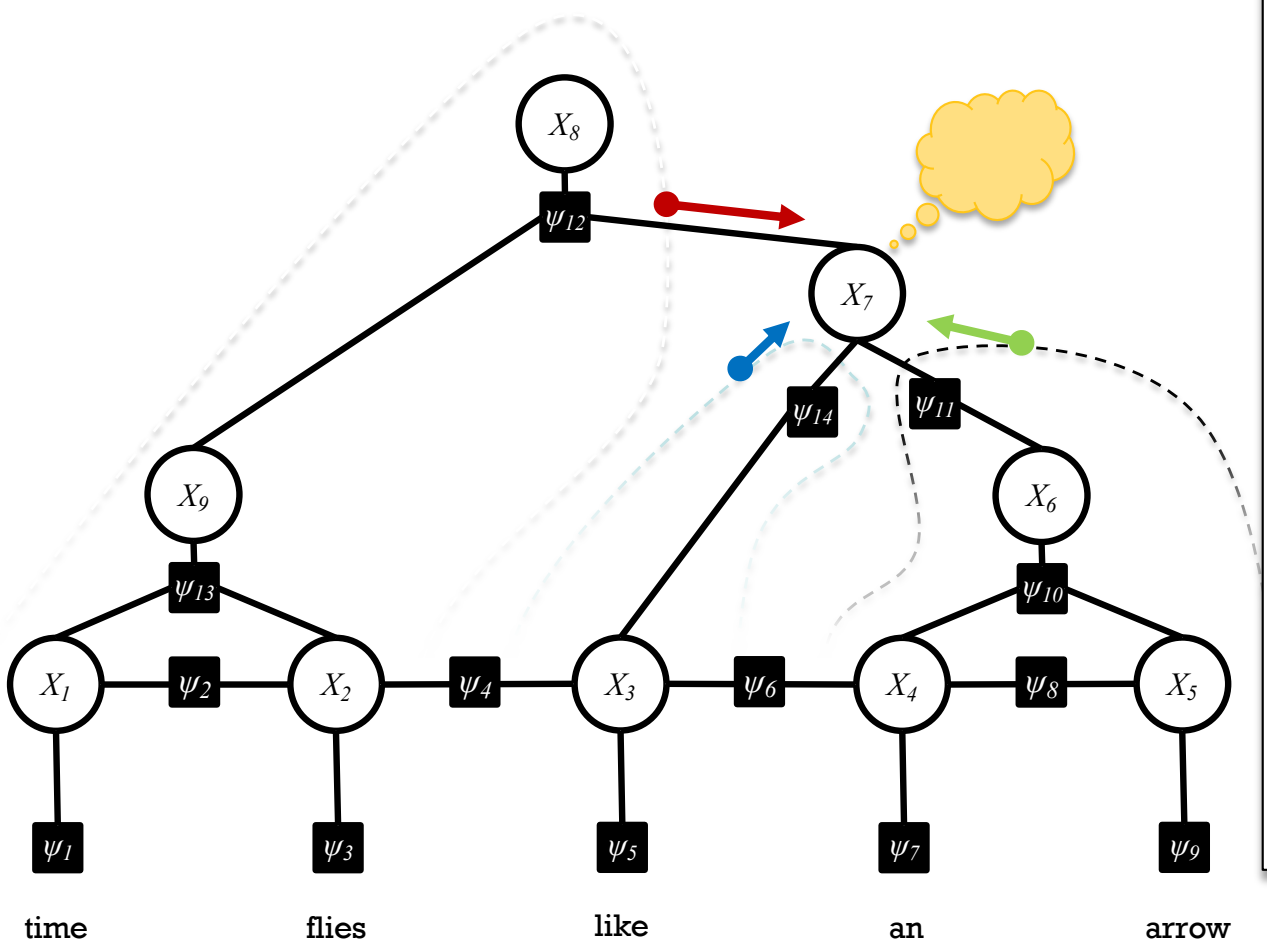
TECHNISCHE UNIVERSITÄT DARMSTADT

- So Belief Propagation on acyclic graphs is really nothing more than variable elimination together with dynamic programming
- That is, we just understood that the JT message passing can be applied directly to tree models.
- **You can also figure this out by splitting a polytree into the part above and below a variable of interest and then computing the belief of that variable recursively.**
- For decoding, replace sum by max

# Loopy Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

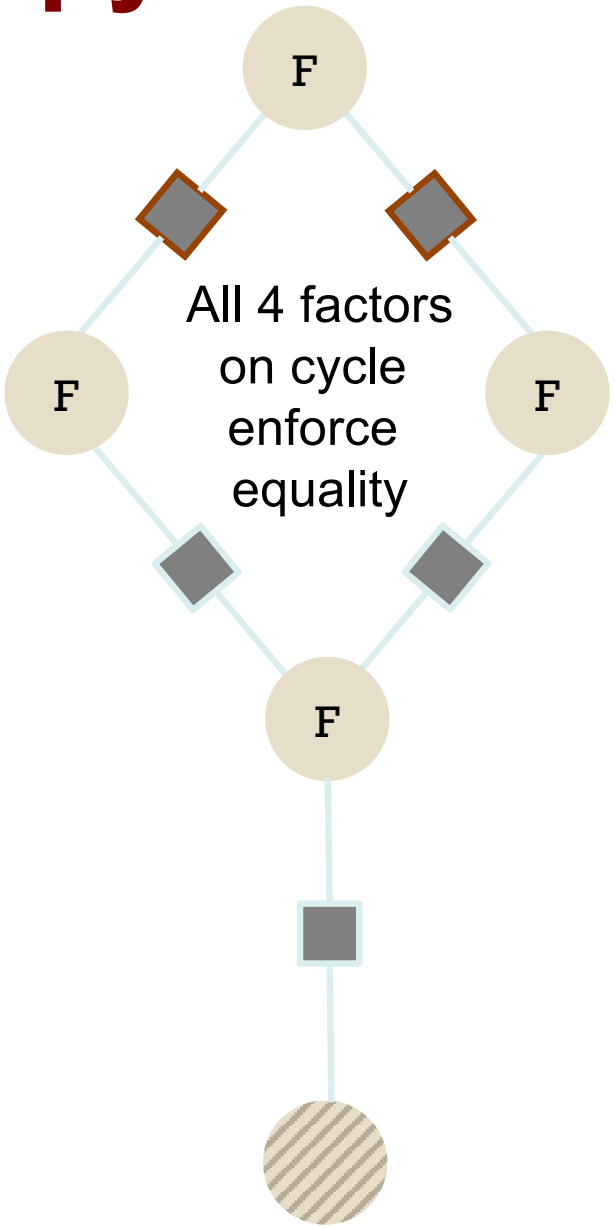
What if our graph has cycles?




- Messages from different subgraphs are **no longer independent**!
  - Dynamic programming can't help. It's now #P-hard in general to compute the exact marginals.
- **But we can still run BP** -- it's a local algorithm so it doesn't "see the cycles."

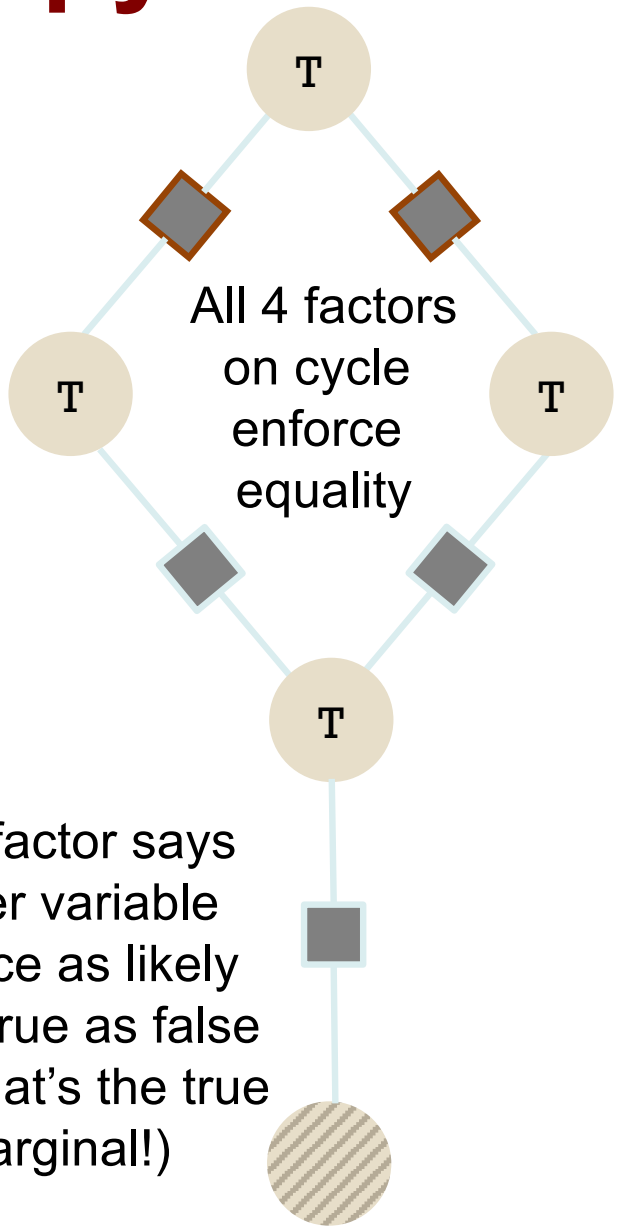
# What can go wrong with loopy BP?

TECHNISCHE UNIVERSITÄT DARMSTADT



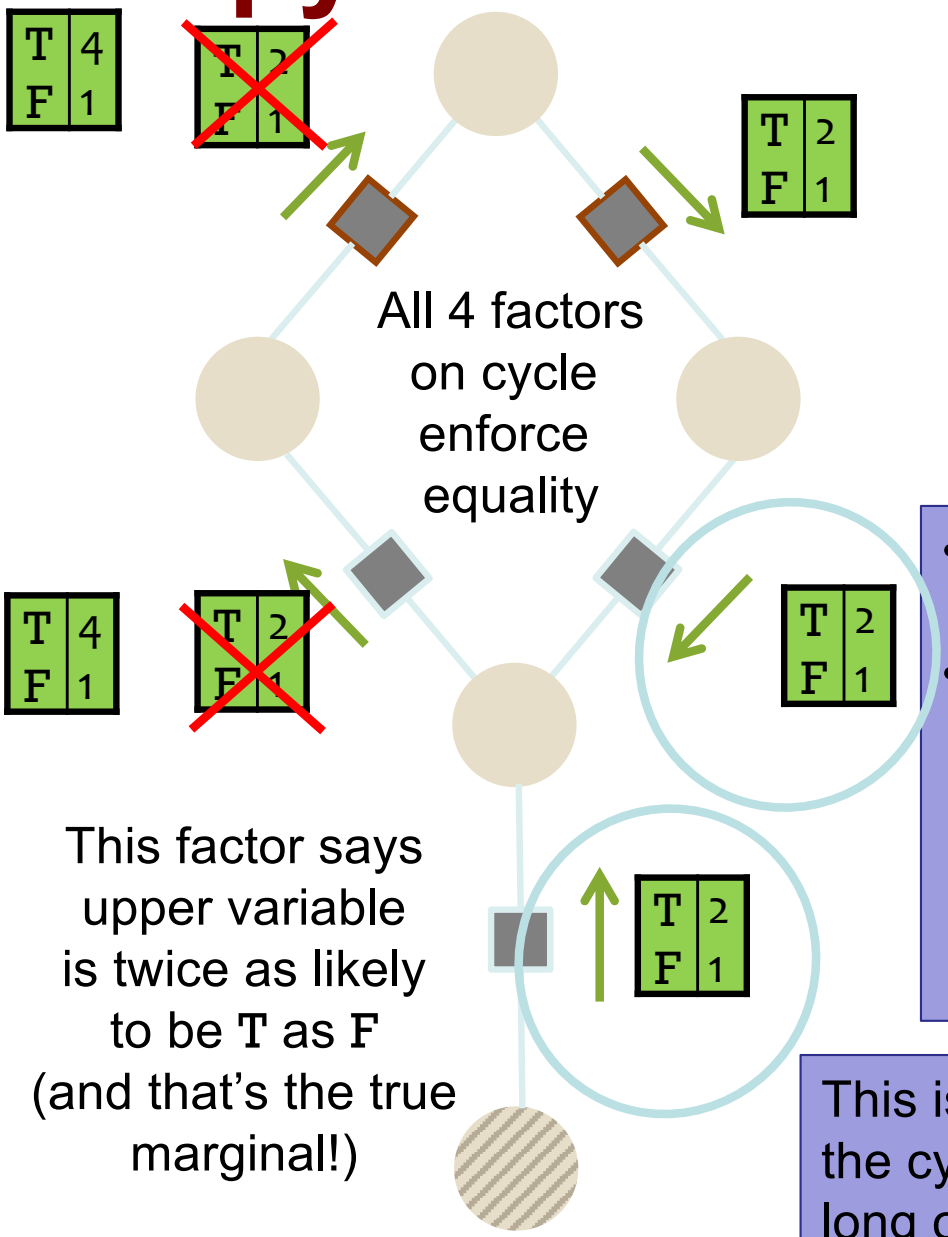
# What can go wrong with loopy BP?

TECHNISCHE UNIVERSITÄT DARMSTADT




This factor says upper variable is twice as likely to be true as false (and that's the true marginal!)

# What can go wrong with loopy BP?




- Messages loop around and around …
- 2, 4, 8, 16, 32, ... More and more convinced that these variables are T!
- So beliefs converge to marginal distribution (1, 0) rather than (2/3, 1/3).

- BP incorrectly treats this message as separate evidence that the variable is T.
- Multiplies these two messages as if they were independent.
  - But they don't actually come from *independent* parts of the graph.
  - One influenced the other (via a cycle).

This is an extreme example. Often in practice, the cyclic influences are weak. (As cycles are long or include at least one weak correlation.)

# Loopy Belief Propagation

TECHNISCHE UNIVERSITÄT DARMSTADT

- If BP is used on graphs with loops, messages may circulate indefinitely
- **Empirically, a good approximation is still achievable**
  - Stop after fixed # of iterations
  - Stop when no significant change in beliefs
  - If solution is not oscillatory but converges, it usually is a good approximation
- Example: Turbo Codes, **Lifted Inference**

# Sampling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Input: Bayesian network with set of nodes X
- Sample = a tuple with assigned values

$$s=(X_1=x_1,X_2=x_2,\ldots ,X_k=x_k)$$

- Tuple may include all variables (except evidence) or a subset
- Sampling schemas dictate how to generate samples
- Ideally, samples are distributed according to P(X|E)

# Sampling Fundamentals

TECHNISCHE UNIVERSITÄT DARMSTADT

Given a set of variables $X = \{X_1, X_2, \ldots X_n\}$ that represent a joint probability distribution $\boldsymbol{\pi(X)}$ and some function g(X), we can compute the expected value of **g(X)** as follows:

$$E_\pi g = \int g(x)\pi(X)dx$$

# Sampling From $\pi(X)$

TECHNISCHE UNIVERSITÄT DARMSTADT

A sample $\mathbf{S^t}$ is an instantiation:

$$S^t = \{x_1^t, x_2^t, \ldots, x_n^t\}$$

Given independent, identically distributed samples (iid) $S^1, S^2, \ldots S^T$ from $\pi(X)$, it follows from **Strong Law of Large Numbers**:

$$\overline{g} = \frac{1}{T}\sum_{t=1}^{T} g(S^t)$$

# Sampling Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Given random variable X over D(X)={0, 1}
- Given P(X) = {0.3, 0.7}
- Generate k samples: 0,1,1,1,0,1,1,0,1
- Approximate P'(X):

$$P'(X=0) = \frac{\#samples(X=0)}{\#samples} = \frac{4}{10} = 0.4$$

$$P'(X=1) = \frac{\#samples(X=1)}{\#samples} = \frac{6}{10} = 0.6$$

$$P'(X) == \{0.4, 0.6\}$$

# How to draw a sample ?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Given random variable X, D(X)={0, 1}
- Given P(X) = {0.3, 0.7}
- Sample X $\leftarrow$ P (X)
  - draw random number r $\in$ [0, 1]
  - If (r < 0.3) then set X=0
  - Else set X=1
- Can generalize for any domain size

# Sampling in BN

TECHNISCHE UNIVERSITÄT DARMSTADT

- Same Idea: generate a set of samples T
- Estimate $P(X_i|E)$ from samples
- Challenge: X is a vector and P(X) is a huge distribution represented by BN
- Need to know:
  - How to generate a new sample ?
  - How many samples T do we need ?
  - How to estimate $P(E=e)$ and $P(X_i|e)$ ?

# Sampling Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Forward Sampling
- Gibbs Sampling (MCMC)
  - Blocking
  - Rao-Blackwellised
- Likelihood Weighting
- Importance Sampling
- Sequential Monte-Carlo (Particle Filtering) in Dynamic Bayesian Networks

# Forward Sampling - No Evidence (Henrion 1988)

TECHNISCHE UNIVERSITÄT DARMSTADT

Input: Bayesian network

X= $\{X_1,\ldots,X_N\}$, N- #nodes, T - # samples

Output: T samples

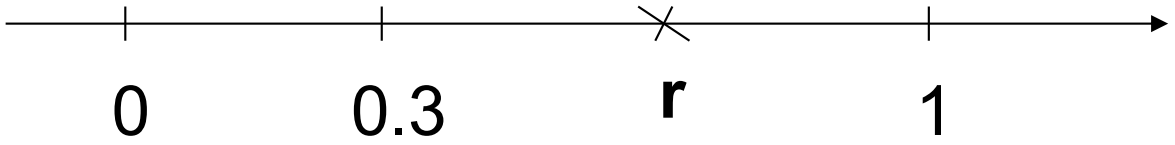
*Process nodes in topological order – first process the ancestors of a node, then the node itself:*

1. For t = 0 to T
2. For i = 0 to N
3. $X_i \leftarrow$ sample $x_i^t$ from $P(x_i \mid pa_i)$

# Sampling a Value

What does it mean to sample $x_i^t$ from $P(X_i \mid pa_i)$ ?

- Assume $D(X_i)=\{0,1\}$
- Assume $P(X_i \mid pa_i) = (0.3, 0.7)$

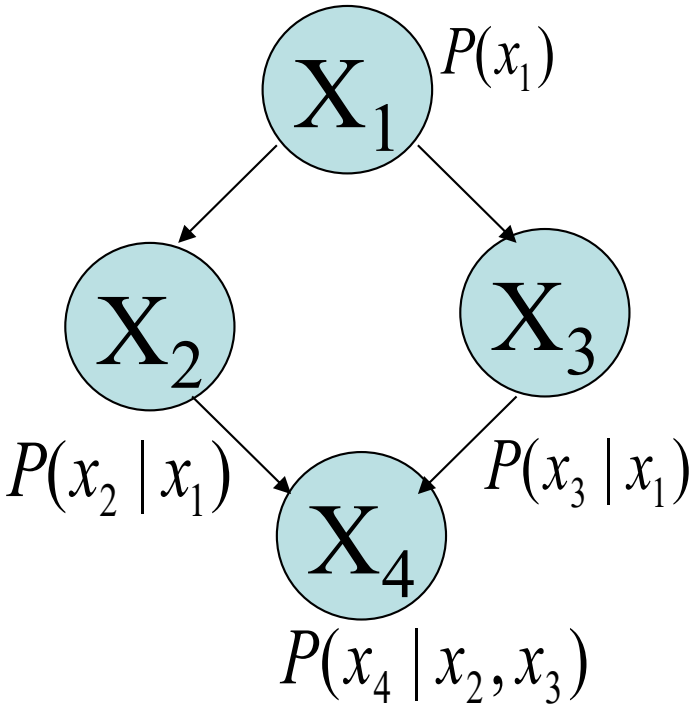



- Draw a random number **r** from [0,1]

  If **r** falls in [0,0.3], set $X_i = 0$

  If **r** falls in [0.3,1], set $X_i=1$

# Forward sampling (example)




No Evidence

// generate sample $k$

1. Sample $x_1$ from $P(x_1)$
2. Sample $x_2$ from $P(x_2 \mid x_1)$
3. Sample $x_3$ from $P(x_3 \mid x_1)$
4. Sample $x_4$ from $P(x_4 \mid x_2, x_3)$

# Forward Sampling-Answering Queries

TECHNISCHE UNIVERSITÄT DARMSTADT

**Task:**

Given **T** samples $\{S^1, S^2, \ldots, S^n\}$ , estimate $P(X_i = x_i)$ :

$$\overline{P}(X_i = x_i) = \frac{\#samples(X_i = x_i)}{T}$$

**Basically, count the proportion of samples where $X_i = x_i$**

# Forward Sampling w/ Evidence

TECHNISCHE UNIVERSITÄT DARMSTADT

Input: Bayesian network

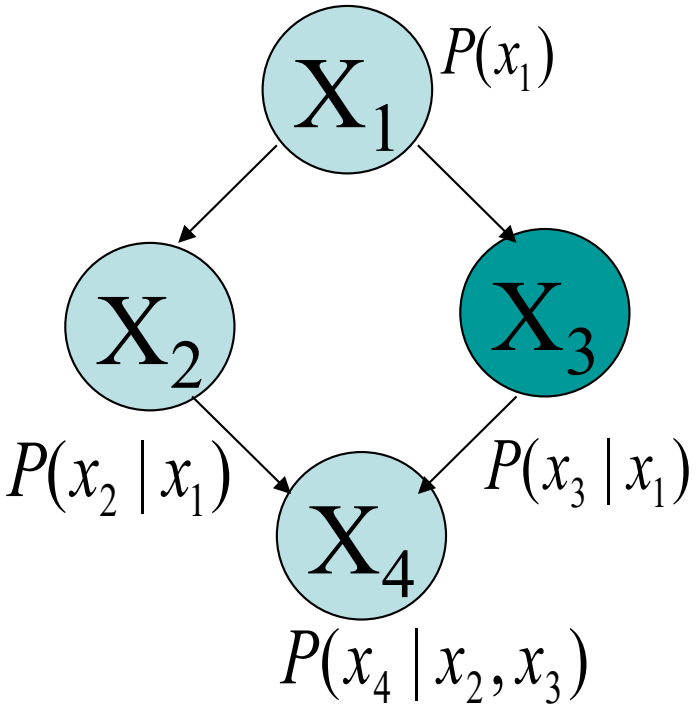
$X = \{X_1,\ldots,X_N\}$, N- #nodes

E – evidence, T - # samples

Output: T samples consistent with E

1. For t=1 to T
2. For i=1 to N
3. $X_i \leftarrow$ sample $x_i^t$ from $P(x_i \mid pa_i)$
4. **If $X_i$ in E and $X_i \neq x_i$, reject sample:**
5. **i = 1 and go to step 2**

# Forward sampling (example)



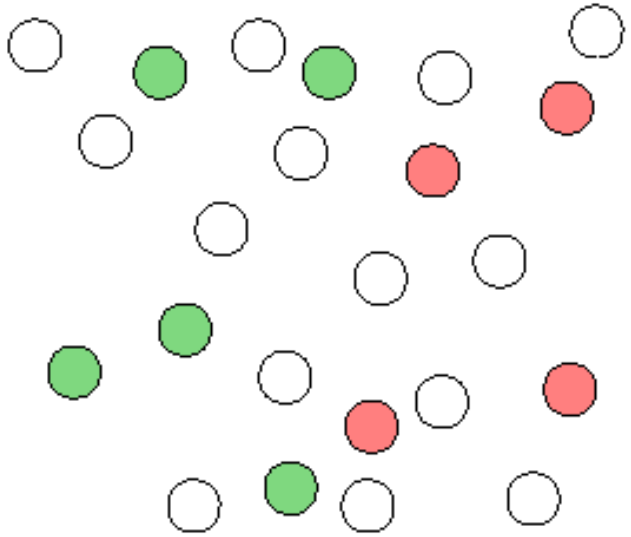

Evidence: $X_3 = 0$

// generate sample $k$

1. Sample $x_1$ from $P(x_1)$
2. Sample $x_2$ from $P(x_2 \mid x_1)$
3. Sample $x_3$ from $P(x_3 \mid x_1)$
4. If $x_3 \neq 0$, reject sample and start from 1, otherwise
5. Sample $x_4$ from $P(x_4 \mid x_2, x_3)$

# Forward Sampling: Illustration

Let Y be a subset of evidence nodes s.t. Y=u



Sample with

not $\boldsymbol{Y} = \boldsymbol{u}$

$\boldsymbol{Y} = \boldsymbol{u}, X \neq w$

$\boldsymbol{Y} = \boldsymbol{u}, X = w$

Approximation for $P^{\boldsymbol{X}}(X = w \mid \boldsymbol{Y} = \boldsymbol{u})$: $\frac{\#\,\bigcirc}{\#\,\bigcirc \cup \bigcirc}$

# Forward Sampling:Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

Advantages:

- $P(x_i \mid pa(x_i))$ is readily available
- Samples are independent !

Drawbacks:

- If evidence **E** is rare (P(e) is low), then we will reject most of the samples!
- Since **P(y)** in estimate of **T** is unknown, must estimate **P(y)** from samples themselves!
- If **P(e)** is small, **T** will become very big!

# Problem: Evidence

TECHNISCHE UNIVERSITÄT DARMSTADT

- Forward Sampling
  - High Rejection Rate
- Fix evidence values
  - Gibbs sampling (MCMC)
  - Likelihood Weighting
  - Importance Sampling

# Gibbs Sampling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Markov Chain Monte Carlo method

**(Gelfand and Smith, 1990, Smith and Roberts, 1993, Tierney, 1994)**

- Samples are **dependent**, form Markov Chain
- Sample from P'(X|e) which **converges** to P(X|e)
- Guaranteed to converge when all P > 0
- Methods to improve convergence:
  - Blocking
  - Rao-Blackwellised

# Gibbs Sampling (Pearl, 1988)

TECHNISCHE UNIVERSITÄT DARMSTADT

- A sample $\mathbf{t} \in [1,2,\ldots]$, is an instantiation of all variables in the network:

$$x^t = \{X_1 = x_1^t, X_2 = x_2^t, \ldots, X_N = x_N^t\}$$

- Sampling process
  - Fix values of observed variables e
  - Instantiate node values in sample $x^0$ at random
  - Generate samples $x^1, x^2, \ldots x^T$ from P(x|e)
  - As before, compute posteriors from samples

# Ordered Gibbs Sampler

Generate sample $x^{t+1}$ from $x^t$ :

Process All Variables In Some Order

$$X_1 = x_1^{t+1} \leftarrow P(x_1 \mid x_2^t, x_3^t, \ldots, x_N^t, e)$$

$$X_2 = x_2^{t+1} \leftarrow P(x_2 \mid x_1^{t+1}, x_3^t, \ldots, x_N^t, e)$$

...

$$X_N = x_N^{t+1} \leftarrow P(x_N \mid x_1^{t+1}, x_2^{t+1}, \ldots, x_{N-1}^{t+1}, e)$$

In short, for i=1 to N:

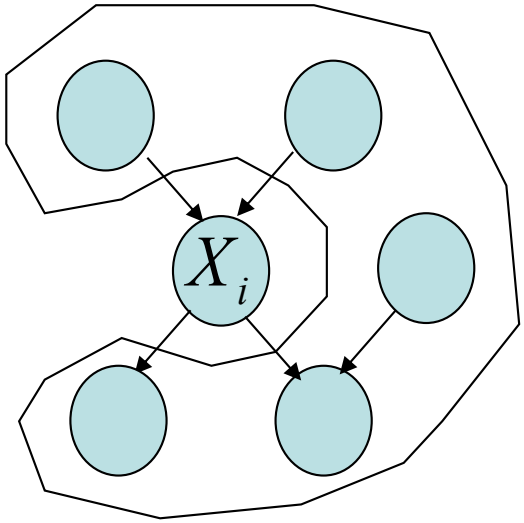
$$X_i = x_i^{t+1} \leftarrow \textbf{sampled from } P(x_i \mid x^t \setminus x_i, e)$$

# Gibbs Sampling (cont'd)

**(Pearl, 1988)**

$$\text{Important: } P(x_i \mid x^t \setminus x_i) = P(x_i \mid markov^t \setminus x_i):$$

$$P(x_i \mid x^t \setminus x_i) \propto P(x_i \mid pa_i) \prod_{X_j \in ch_i} P(x_j \mid pa_j)$$




Markov blanket:

$$M(X_i) = pa_i \cup ch_i \cup (\bigcup_{X_j \in ch_j} pa_j)$$

Given *Markov blanket*

(parents, children,and their parents),

$X_i$ is independent of all other nodes

# Ordered Gibbs Sampling Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

Input: X, E

Output: T samples $\{x^t\}$

- Fix evidence E
- Generate samples from P(X | E)

1. For t = 1 to T (compute samples)
2. For i = 1 to N (loop through variables)
3. $X_i \leftarrow$ sample $x_i^t$ from $P(X_i \mid markov^t \setminus X_i)$

# Answering Queries

TECHNISCHE UNIVERSITÄT DARMSTADT

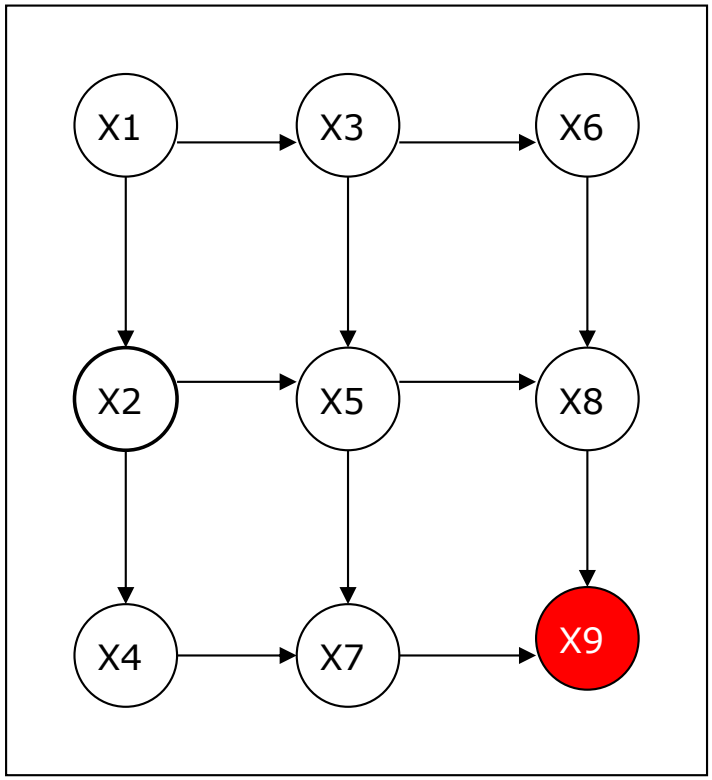
- **Query**: $P(x_i|e) = ?$
- **Method 1**: count #of samples where $X_i=x_i$:

$$\overline{P}(X_i = x_i) = \frac{\#samples(X_i = x_i)}{T}$$

**Method 2**: average probability (mixture estimator):

$$\overline{P}(X_i = x_i) = \frac{1}{T}\sum_{t=1}^{n} P(X_i = x_i \,|markov^t \setminus X_i)$$

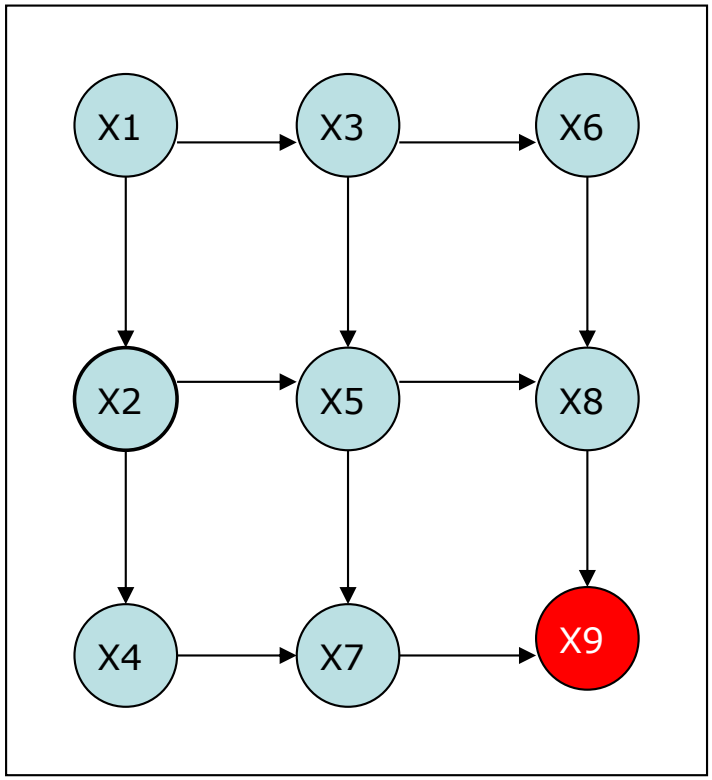
# Gibbs Sampling Example - BN




$$X = \{X_1, X_2, \ldots, X_9\}$$

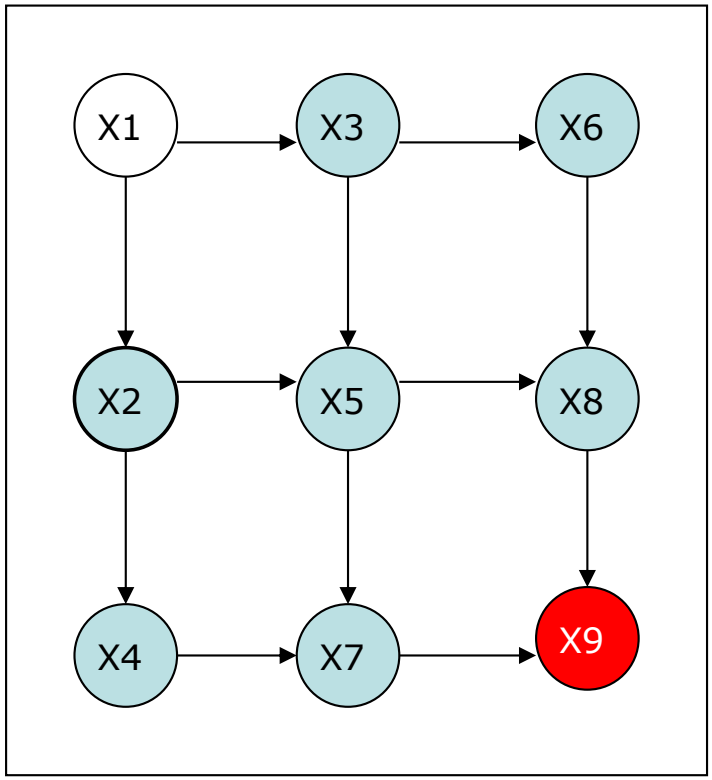
$$E = \{X_9\}$$

# Gibbs Sampling Example - BN




$$X_1 = x_1^0 \qquad X_6 = x_6^0$$
$$X_2 = x_2^0 \qquad X_7 = x_7^0$$
$$X_3 = x_3^0 \qquad X_8 = x_8^0$$
$$X_4 = x_4^0$$
$$X_5 = x_5^0$$

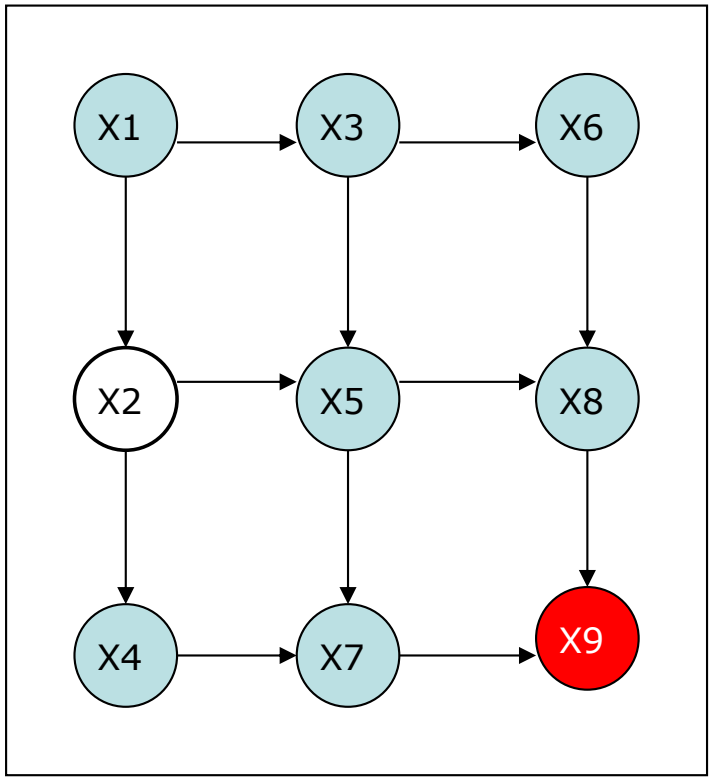
# Gibbs Sampling Example - BN




$$X_1 \leftarrow P(X_1 \mid X^0_2,\ldots,X^0_8, X_9\}$$

$$E = \{X_9\}$$

# Gibbs Sampling Example - BN



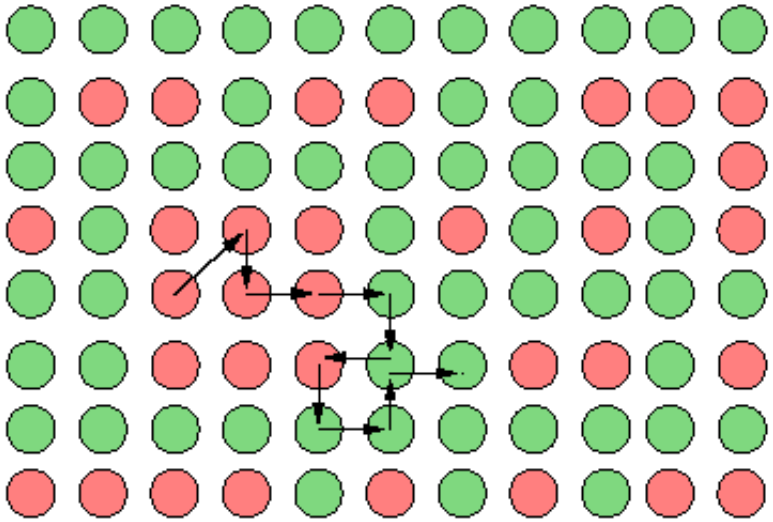

$$X_2 \leftarrow P(X_2 \mid X^1_1,\ldots,X^0_8,X_9\}$$

$$E = \{X_9\}$$

# Gibbs Sampling: Illustration

TECHNISCHE UNIVERSITÄT DARMSTADT

The process of Gibbs sampling can be understood as a *random walk* in the space of all instantiations with $\boldsymbol{Y} = \boldsymbol{u}$:



Reachable in one step: instantiations that differ from current one by value assignment to at most one variable (assume randomized choice of variable $X_k$).

# Gibbs Sampling: Burn-In

TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to sample from P(X | E)
- But…starting point is **random**
- Solution: throw away first K samples
- Known As “Burn-In”
- What is K ? Hard to tell. Use intuition.
- Alternatives: sample first samples from approximate P(x|e) (for example, run Loopy Belief Propagation first)

# Gibbs Sampling: Convergence

TECHNISCHE UNIVERSITÄT DARMSTADT

- Converge to stationary distribution $\pi^*$ :

$$\pi^* = \pi^* P$$

where P is a transition kernel

$$p_{ij} = P(X^i \rightarrow X^j)$$

- **Guaranteed to converge iff chain is** :
  - irreducible
  - aperiodic
  - ergodic ( $\forall i,j\ p_{ij} > 0$)

# Background: Irreducible

TECHNISCHE UNIVERSITÄT DARMSTADT

- A Markov chain (or its probability transition matrix) is said to be *irreducible* if it is possible to reach every state from every other state (not necessarily in one step).
- In other words, $\forall i,j \; \exists k : P^{(k)}_{ij} > 0$ where k is the number of steps taken to get to state j from state i.

# Background: Aperiodic

TECHNISCHE UNIVERSITÄT DARMSTADT

- Define **d(i) = g.c.d.{n > 0 | it is possible to go from i to i in n steps}**
- Here, g.c.d. means the greatest common divisor of the integers in the set.
- **If d(i)=1 for ∀i, then chain** is *aperiodic*, i.e., returns to state i can occur at **irregular times**

# Background: Ergodicity

TECHNISCHE UNIVERSITÄT DARMSTADT

- A *recurrent* state is a state to which the chain returns with probability 1:

$$\sum_n P^{(n)}_{ij} = \infty$$

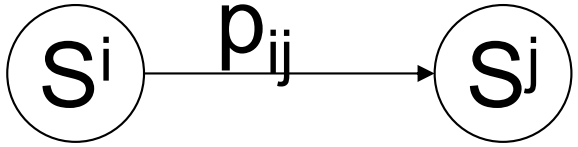
- **Recurrent, aperiodic states are *ergodic*.**

Note: an extra condition for ergodicity is that expected recurrence time is finite. This holds for recurrent states in a finite state chain.

# Background: Gibbs Sampling and Ergodicity

TECHNISCHE UNIVERSITÄT DARMSTADT

- Convergence to the correct distribution is guaranteed only if network is ***ergodic***: transition from any state $S^i$ to any state $S^j$ has non-zero probability:




$$p_{ij} > 0$$

**Intuition: if $\exists i,j$ such that $p_{ij} = 0$ , then we will not be able to explore full sampling space !**

# Gibbs Convergence

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gibbs convergence is generally guaranteed as long as all probabilities are positive!
- Intuition for **ergodicity** requirement:

  if nodes X and Y are correlated s.t. X=0 $\leftrightarrow$Y=0, then:
  - once we sample and assign X=0, then we are forced to assign Y=0;
  - once we sample and assign Y=0, then we are forced to assign X=0;

➔ we will never be able to change their values again!

- **Another problem: it can take a very long time to converge!**

# Gibbs Sampling: Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

**+Advantage: guaranteed to converge to P(X|E)**
**-Disadvantage: convergence may be slow**

**Problems:**

- Samples are **dependent** !
- Statistical variance is too big in high-dimensional problems

# Gibbs: Speeding Convergence

TECHNISCHE UNIVERSITÄT DARMSTADT

Objectives:

1. Reduce dependence between samples (autocorrelation)
   - Skip samples
   - Randomize Variable Sampling Order
2. Reduce variance
   - Blocking Gibbs Sampling
   - Rao-Blackwellisation

# Skipping Samples

TECHNISCHE UNIVERSITÄT DARMSTADT

- Pick only every k-th sample (Gayer, 1992)

Can reduce dependence between samples !

Increases variance ! Waists samples !

# Randomized Variable Order

TECHNISCHE UNIVERSITÄT DARMSTADT

**Random Scan Gibbs Sampler**

Pick each next variable $X_i$ for update at random with probability $p_i$ , $\Sigma_i\ p_i = 1$.

(In the simplest case, $p_i$ are distributed uniformly)

In some instances, reduces variance

(MacEachern, Peruggia, 1999 "Subsampling the Gibbs Sampler: Variance Reduction")

# Blocking

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sample several variables **together, as a block**
- **Example:** Given three variables X,Y,Z, with domains of size 2, group Y and Z together to form a variable W={Y,Z} with domain size 4. Then, given sample $(x^t,y^t,z^t)$, compute next sample:

$$X^{t+1} \leftarrow P(y^t,z^t)=P(w^t)$$

$$(y^{t+1},z^{t+1})=W^{t+1} \leftarrow P(x^{t+1})$$

+ Can improve convergence greatly when two variables are strongly correlated!

- Domain of the block variable grows exponentially with the #variables in a block!

# Rao-Blackwellisation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Do not sample all variables!
- Sample a subset!
- **Example:** Given three variables X,Y,Z, sample only X and Y, sum out Z. Given sample $(x^t,y^t)$, compute next sample:

$$X^{t+1} \leftarrow P(y^t)$$

$$y^{t+1} \leftarrow P(x^{t+1})$$

# Rao-Blackwell Theorem

TECHNISCHE UNIVERSITÄT DARMSTADT

**Rao Blackwell Theorem**: *Let a DBN have two groups of variables,* $\mathbf{R}$ *and* $\mathbf{L}$. *Then, for the joint distribution* $\pi(\mathbf{R}, \mathbf{L})$, *the following result applies*

$$\mathrm{Var}_\pi \left[ \mathbb{E}_\pi \{ f(\mathbf{R}) | \mathbf{L} \} \right] \leq \mathrm{Var}_\pi \left[ f(\mathbf{R}) \right]$$

*for a function of interest* $f$, *e.g. the mean or covariance (Casella & Robert, 1996, Liu et. al. 1995).*

**Bottom line: reducing number of variables in a sample reduce variance!**

# Gibbs: Multiple Chains

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generate M chains of size K
- Each chain produces independent estimate $P_m$:

$$P_m = P(x_i \mid e) = \frac{1}{K} \sum_{t=1}^{K} P(x_i \mid x^t \setminus x_i)$$

Estimate $P(x_i|e)$ as average of $P_m(x_i|e)$ :

$$\overline{P} = \frac{1}{M} \sum_{i=1}^{M} P_m$$

Treat $P_m$ as independent random variables.

# Likelihood Weighting
## (Fung and Chang, 1990; Shachter and Peot, 1990)

TECHNISCHE UNIVERSITÄT DARMSTADT

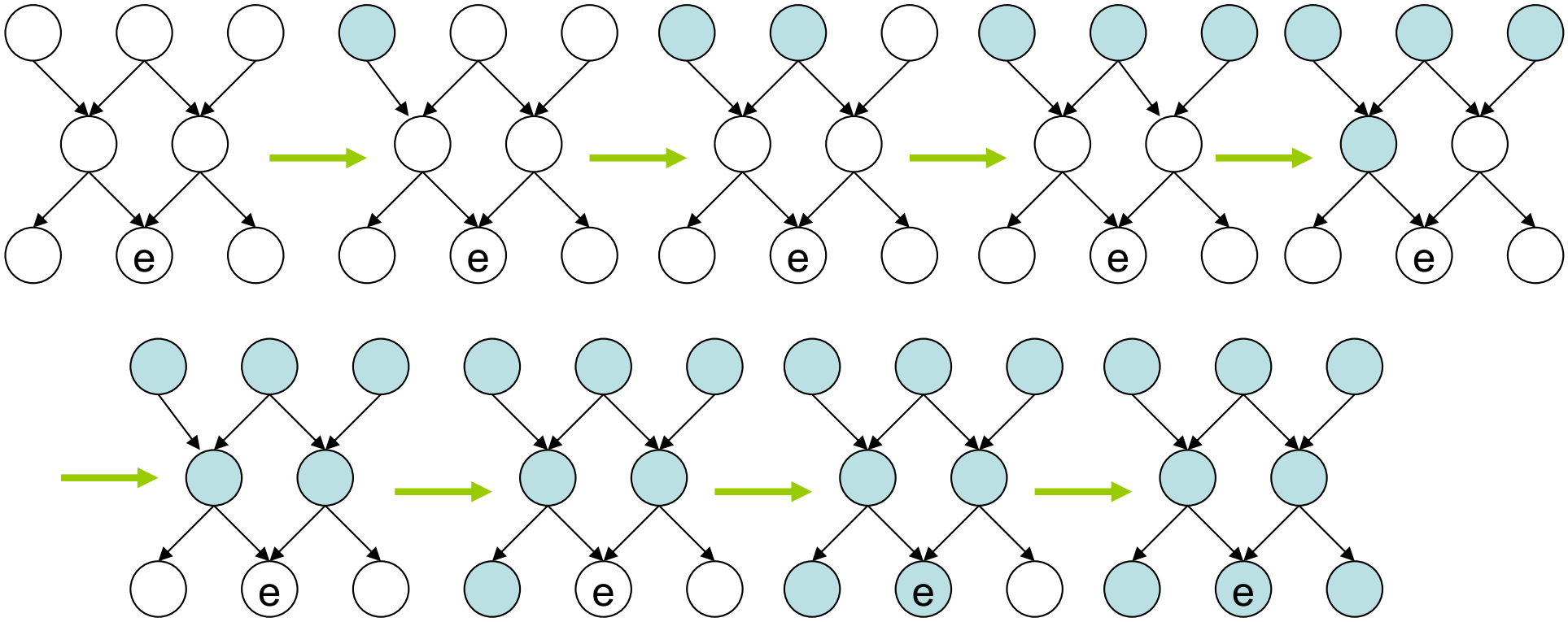
**"Clamping" evidence**
**+ forward sampling**
**+ weighing samples by evidence likelihood**

**Works well for *likely evidence!***

# Likelihood Weighting

TECHNISCHE UNIVERSITÄT DARMSTADT

Sample in topological order over **X** !




$x_i \leftarrow P(X_i|pa_i)$

$P(X_i|pa_i)$ is *a look-up* in CPT!

# Likelihood Weighting

TECHNISCHE UNIVERSITÄT DARMSTADT

**For** $sample\#k = 1$ to $T$

For each each $X_i$ in topological order $o = (X_1, \ldots, X_n)$ :

$w_k = 1$

**if** $X_i \notin E$

$X_i \leftarrow$ sample $x_i$ from $P(x_i \mid pa_i)$

**else**

assign $X_i = e_i$

$w_k = w_k \bullet P(e_i \mid pa_i)$

# Likelihood Weighting

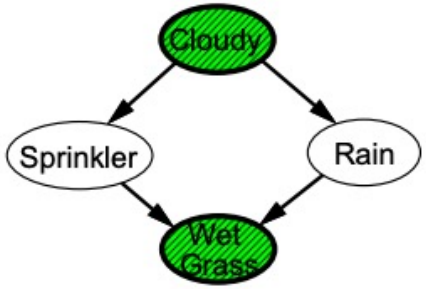
TECHNISCHE UNIVERSITÄT DARMSTADT

Sampling probability for WEIGHTEDSAMPLE is

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | parents(Z_i))$$

Note: pays attention to evidence in **ancestors** only

$\Rightarrow$ somewhere "in between" prior and posterior distribution




Weight for a given sample $\mathbf{z}, \mathbf{e}$ is

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | parents(E_i))$$

Weighted sampling probability is

$$\begin{aligned} & S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) \\ & = \prod_{i=1}^{l} P(z_i | parents(Z_i)) \prod_{i=1}^{m} P(e_i | parents(E_i)) \\ & = P(\mathbf{z}, \mathbf{e}) \text{ (by standard global semantics of network)} \end{aligned}$$

Hence likelihood weighting returns consistent estimates but performance still degrades with many evidence variables because a few samples have nearly all the total weight

# Likelihood Weighting

TECHNISCHE UNIVERSITÄT DARMSTADT

- Converges to exact posterior marginals
- Generates Samples Fast
- Sampling distribution is close to prior (especially if E $\subset$ Leaf Nodes)
- Increasing sampling variance

$\Rightarrow$Convergence may be slow

$\Rightarrow$Many samples with $P(x^{(t)})=0$ rejected