10-418 / 10-618 Machine Learning for Structured Data

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Belief Propagation

Matt Gormley
Lecture 9
Sep. 25, 2019

Q&A

Q: What if I already answered a homework question using different assumptions than what was clarified in a Piazza note?

A: Just write down the assumptions you made.

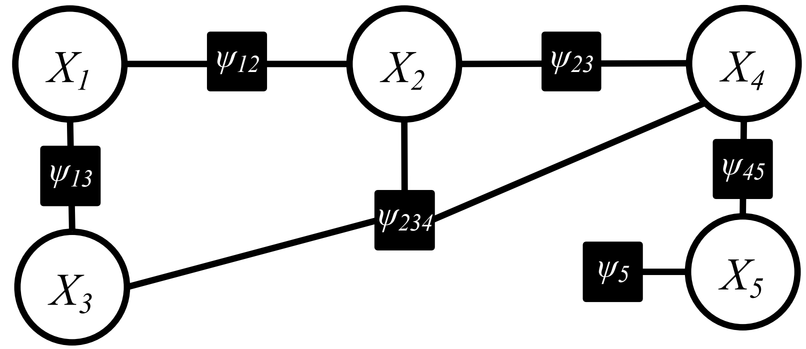
We will usually give credit so long as your assumptions are clear in the writeup and your answer correct under those assumptions.

(Obviously, this only applies to underspecified / ambiguous questions. You can't just add arbitrary assumptions!)

Reminders

- **Homework 1: DAgger for seq2seq**
  - Out: Thu, Sep. 12
  - Due: Thu, Sep. 26 at 11:59pm
- **Homework 2: Labeling Syntax Trees**
  - Out: Thu, Sep. 26
  - Due: Thu, Oct. 10 at 11:59pm

Variable Elimination Complexity



Instead, capitalize on the factorization of $p(\mathbf{x})$.

In-Class Exercise: *Fill in the blank*

Brute force, naïve, inference is O(____)

Variable elimination is O(____)

where
n = # of variables
k = max # values a variable can take
r = # variables participating in largest "intermediate" table

Exact Inference

Variable Elimination

- *Uses*
  - Computes the **partition function** of any factor graph
  - Computes the **marginal probability** of a query variable in any factor graph
- *Limitations*
  - Only computes the marginal for **one variable at a time** (i.e. need to re-run variable elimination for each variable if you need them all)
  - **Elimination order** affects runtime

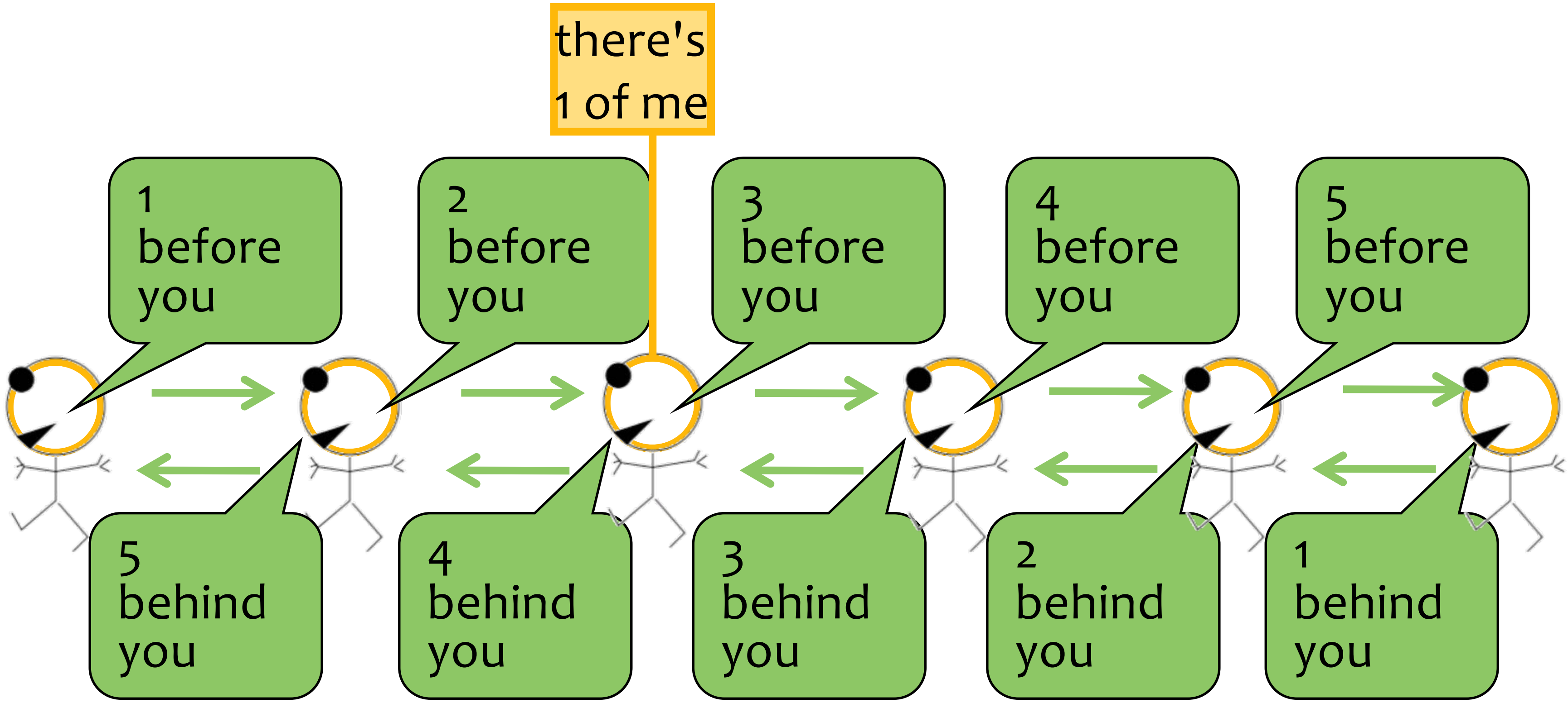
Belief Propagation

- *Uses*
  - Computes the **partition function** of any acyclic factor graph
  - Computes all **marginal probabilities** of factors and variables at once, for any acyclic factor graph
- *Limitations*
  - Only **exact** on acyclic factor graphs (though we'll consider its "loopy" variant later)
  - **Message passing order** affects runtime (but the obvious topological ordering always works best)

MESSAGE PASSING

Great Ideas in ML: Message Passing

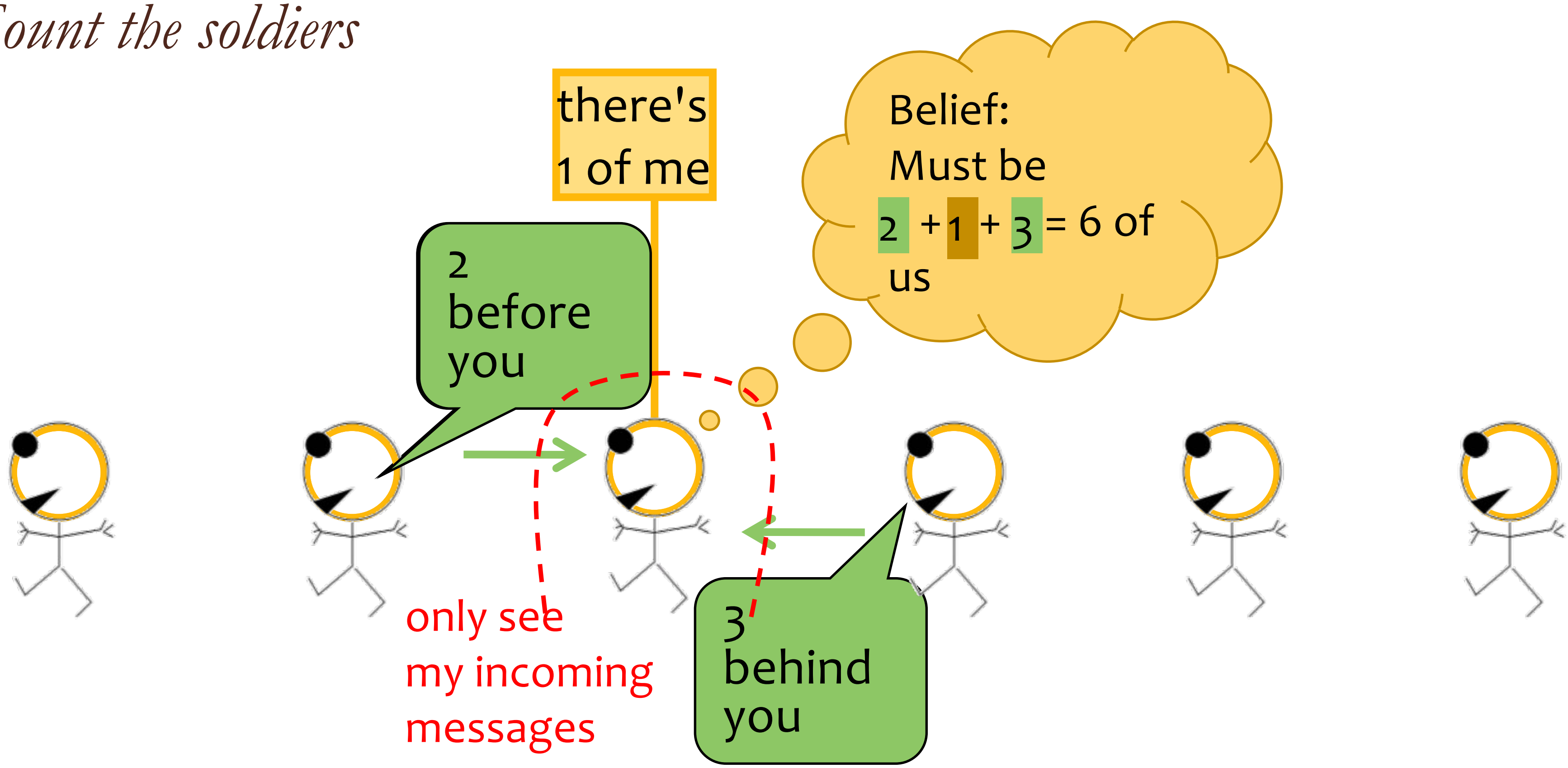
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

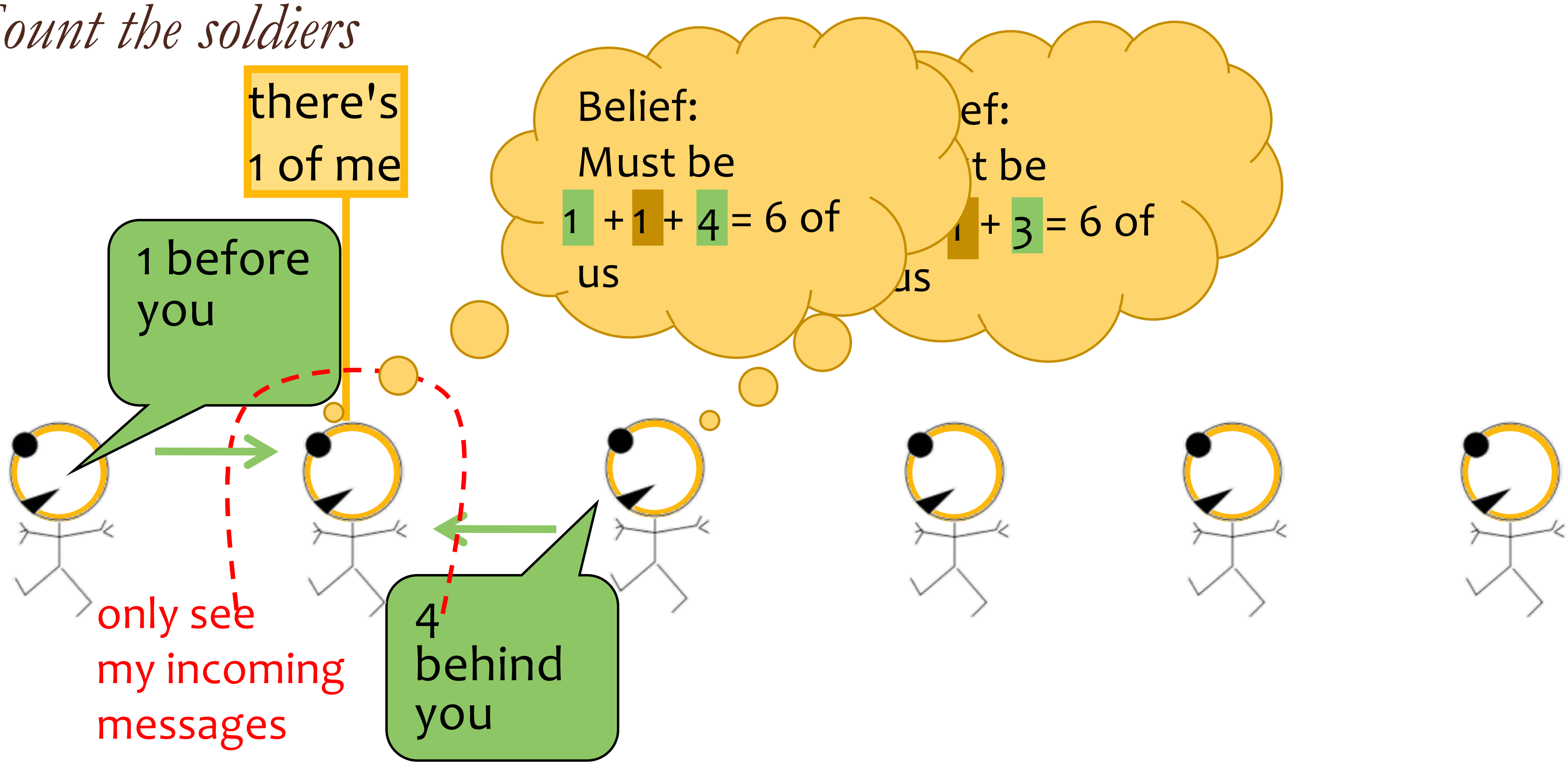
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

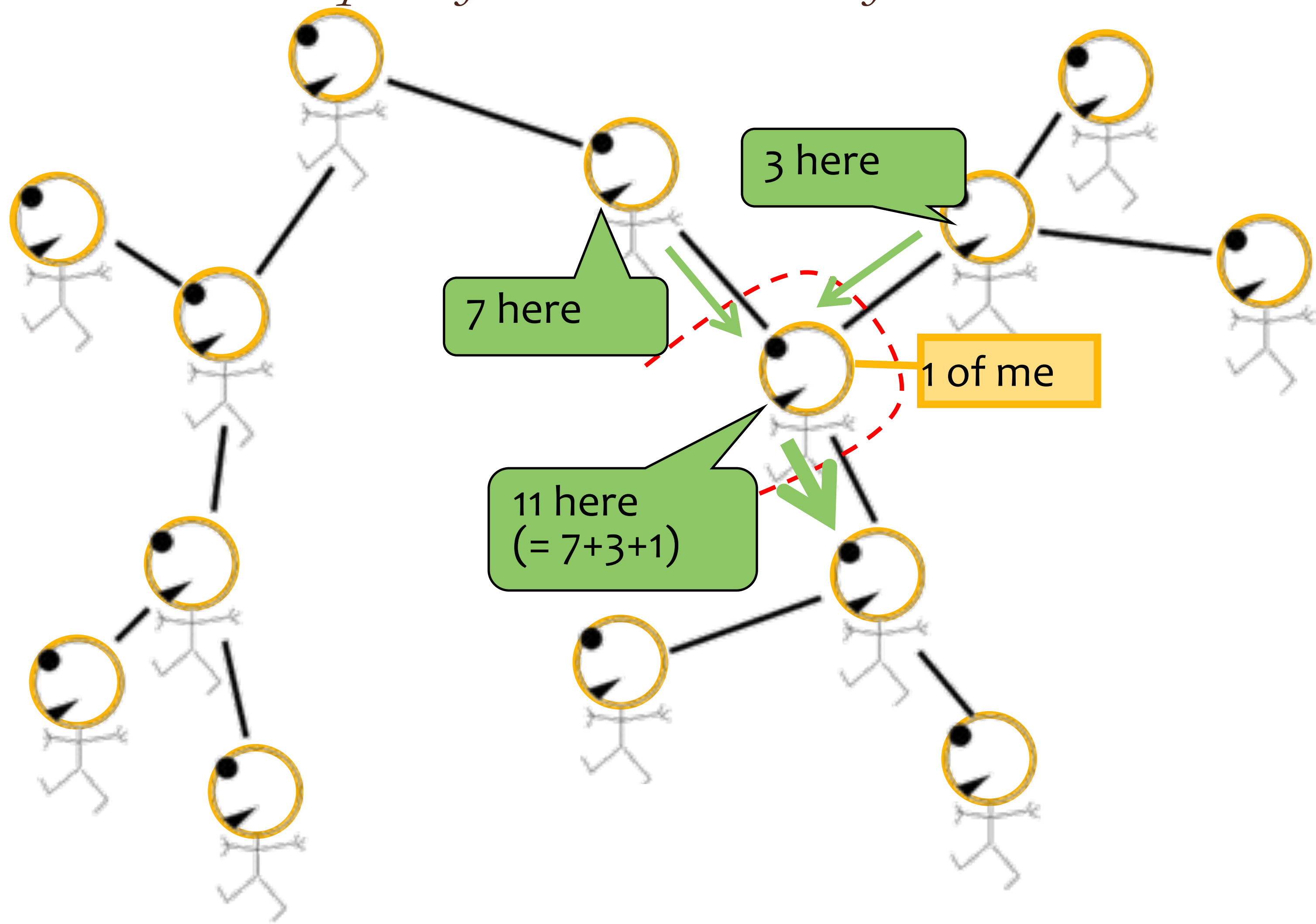
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

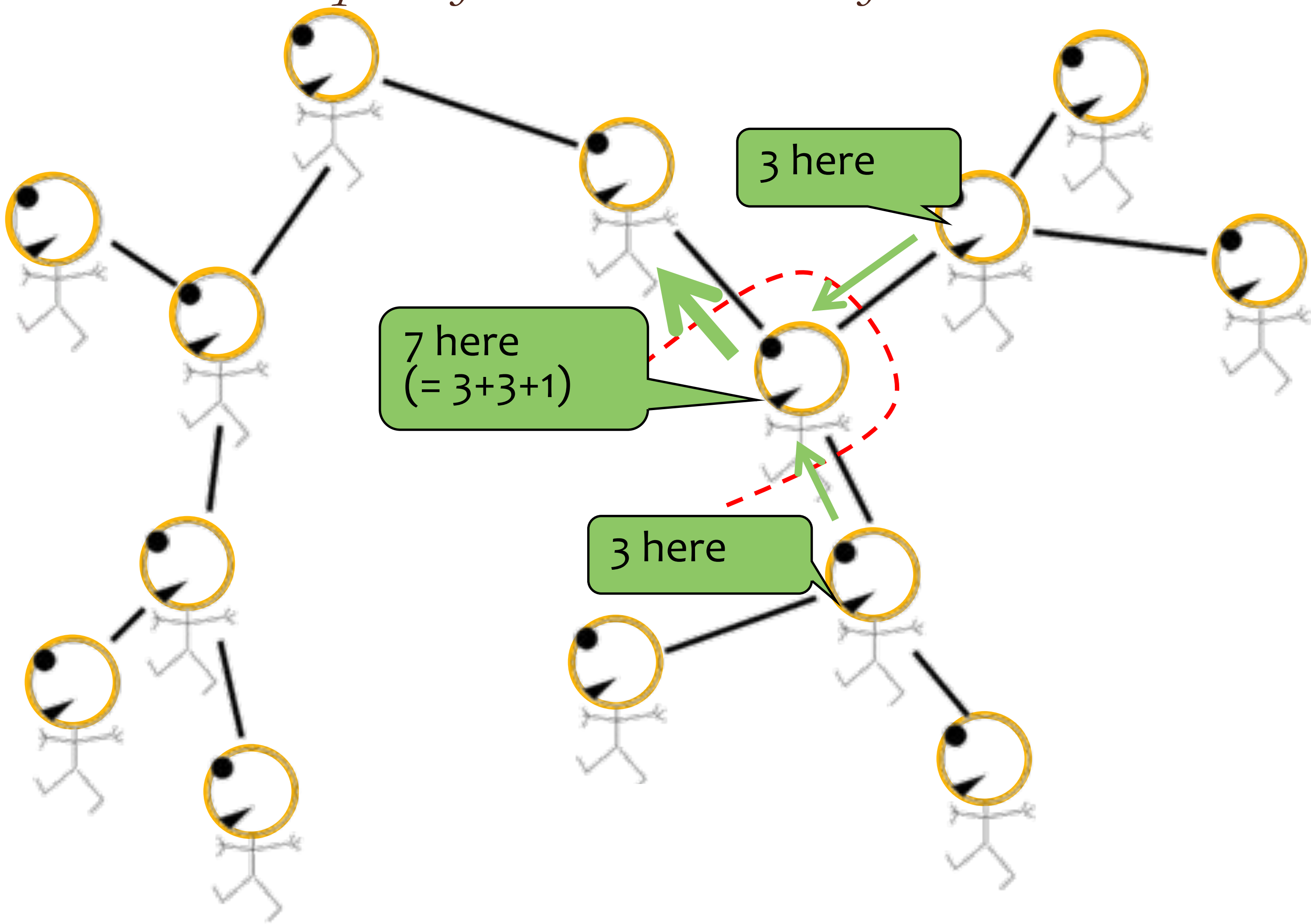
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

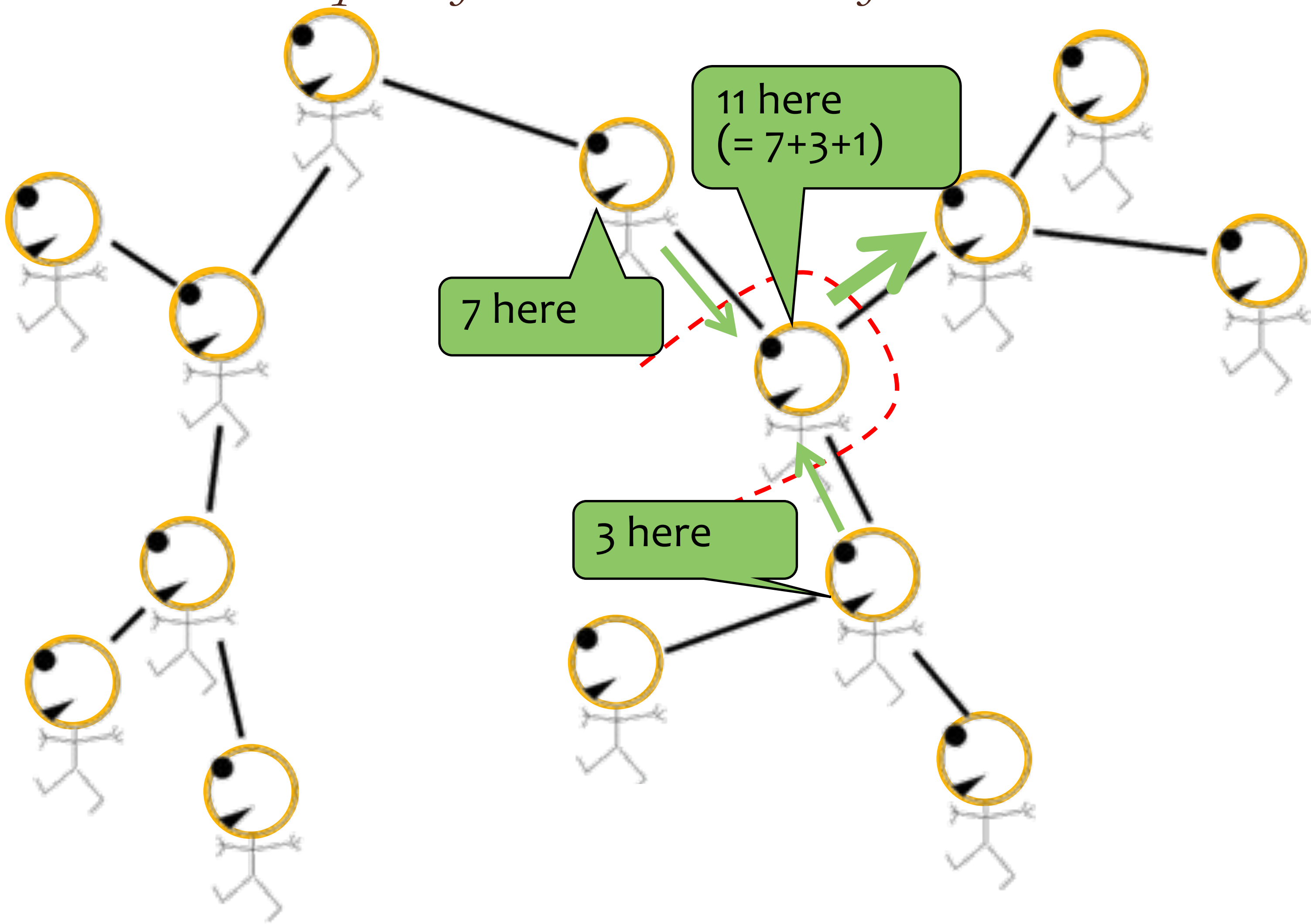
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

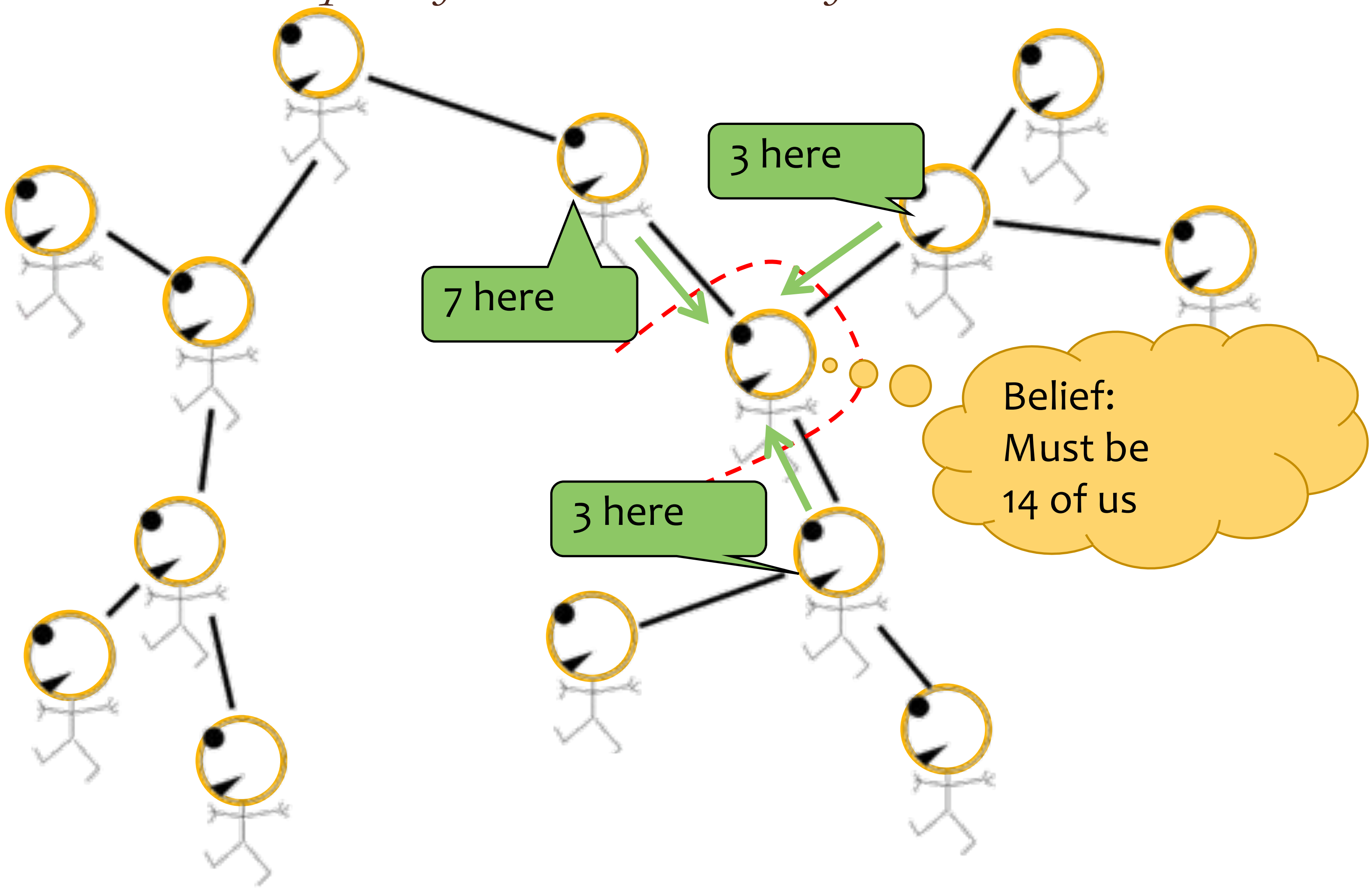
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

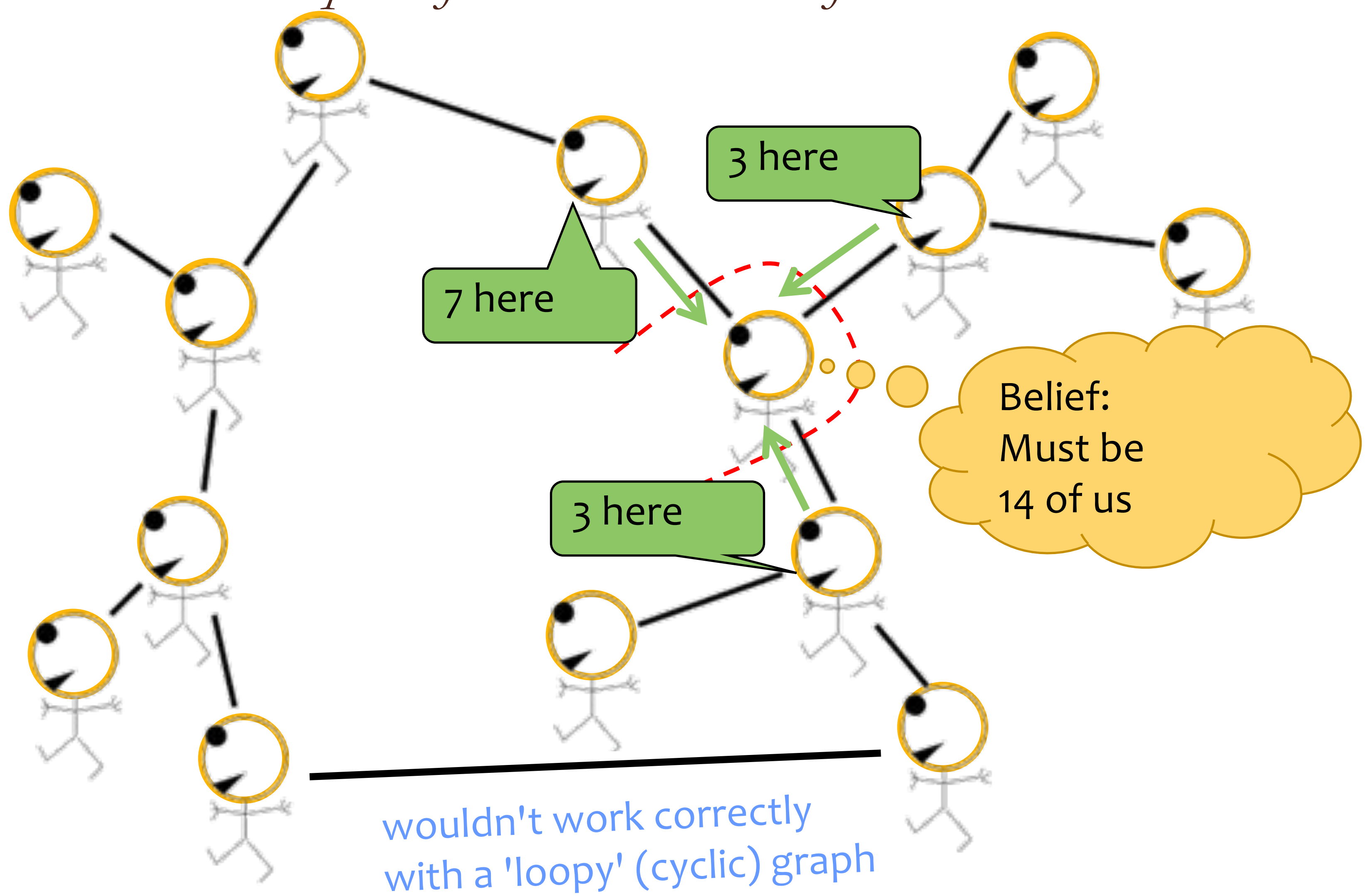
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

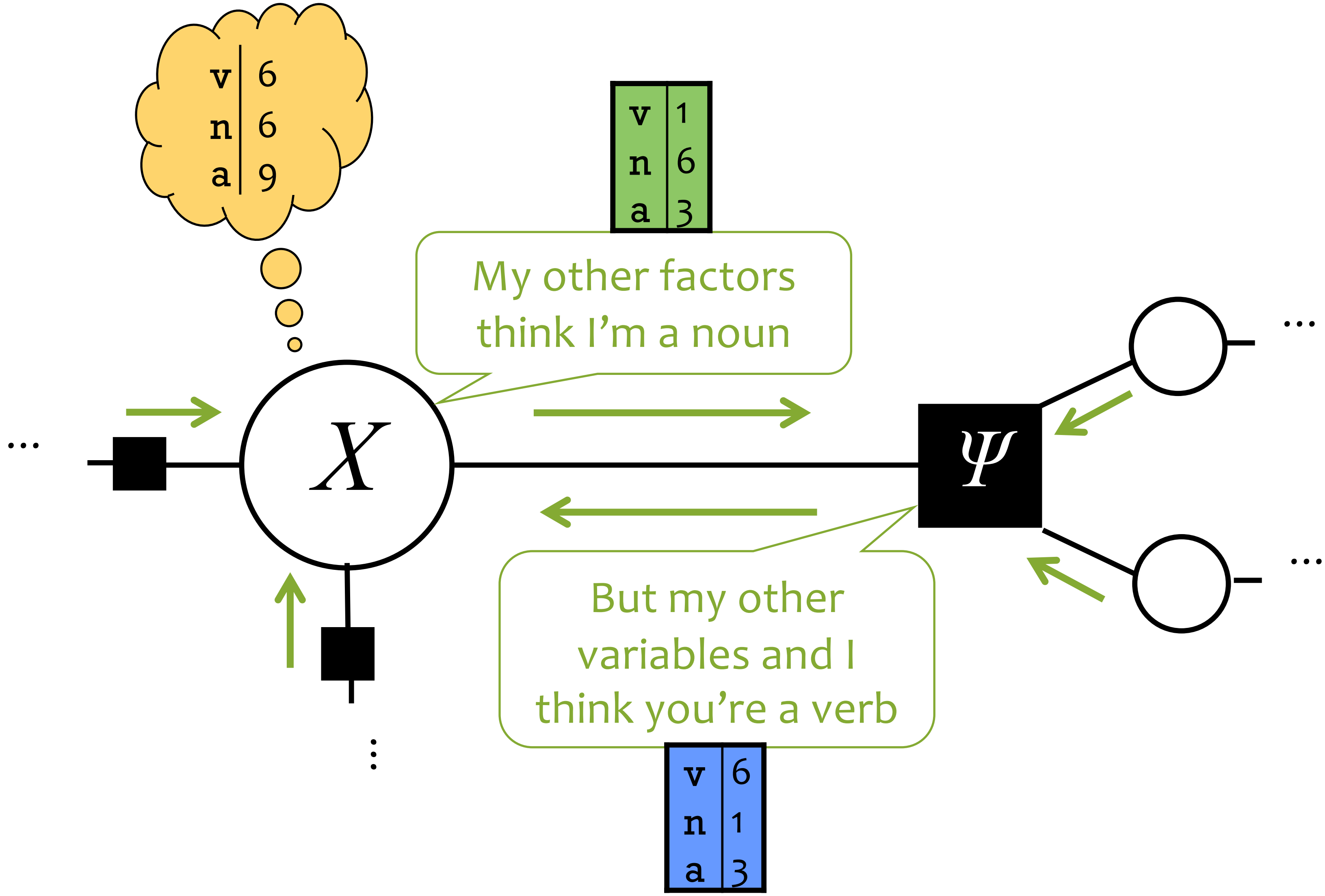


adapted from MacKay (2003) textbook

Exact marginal inference for factor trees

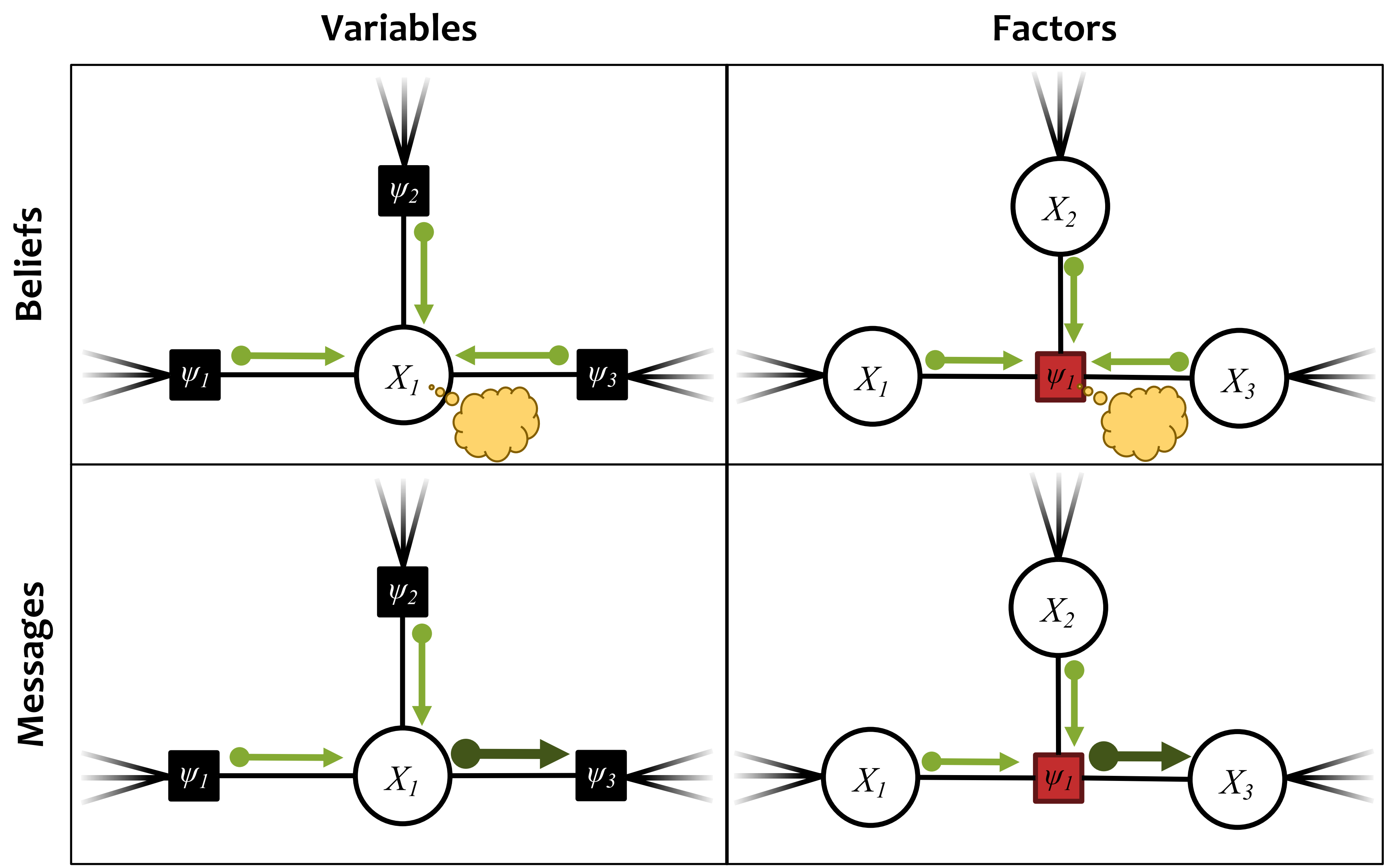
SUM-PRODUCT BELIEF PROPAGATION

Message Passing in Belief Propagation



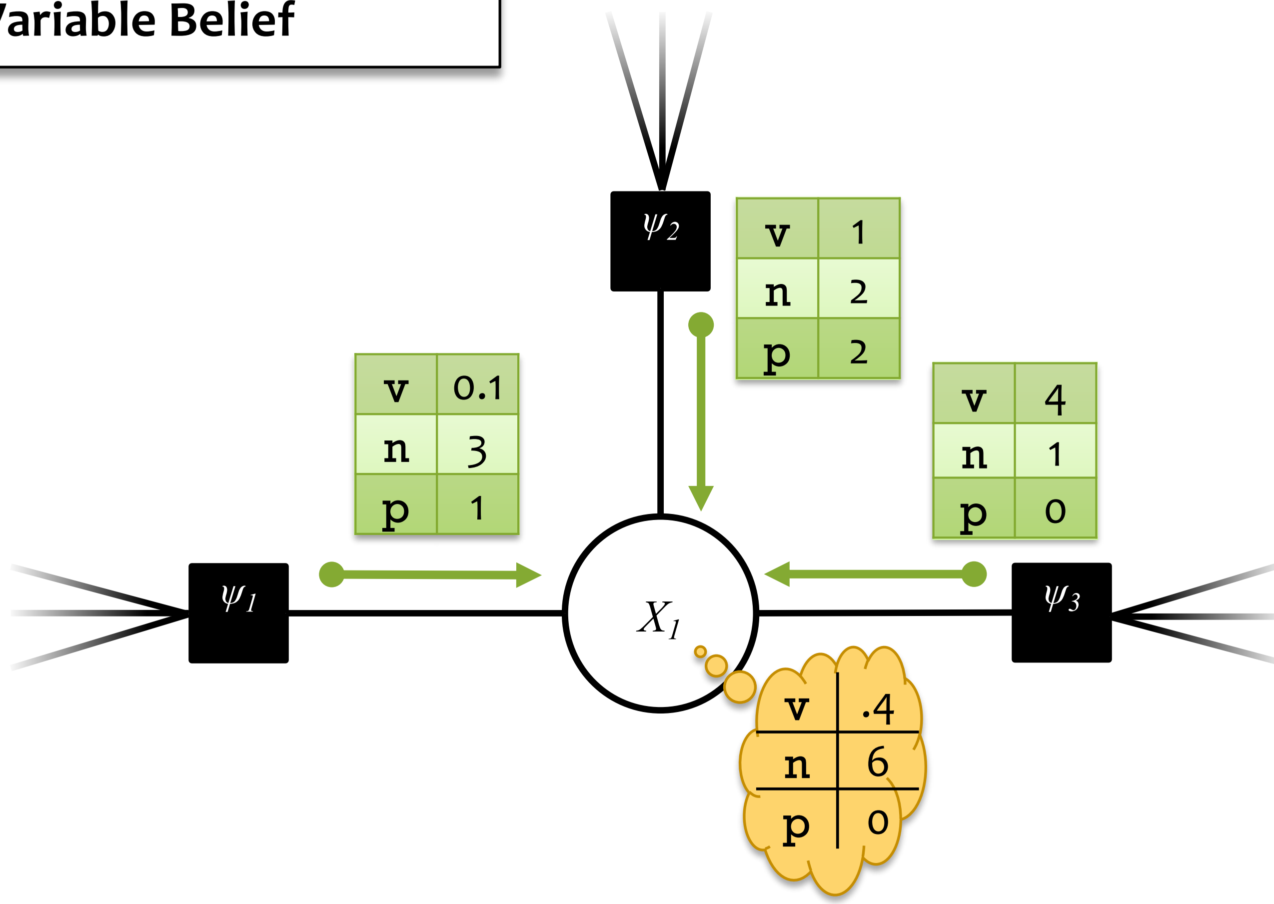
Both of these messages judge the possible values of variable X.
Their product = belief at X = product of all 3 messages to X.

Sum-Product Belief Propagation



Sum-Product Belief Propagation

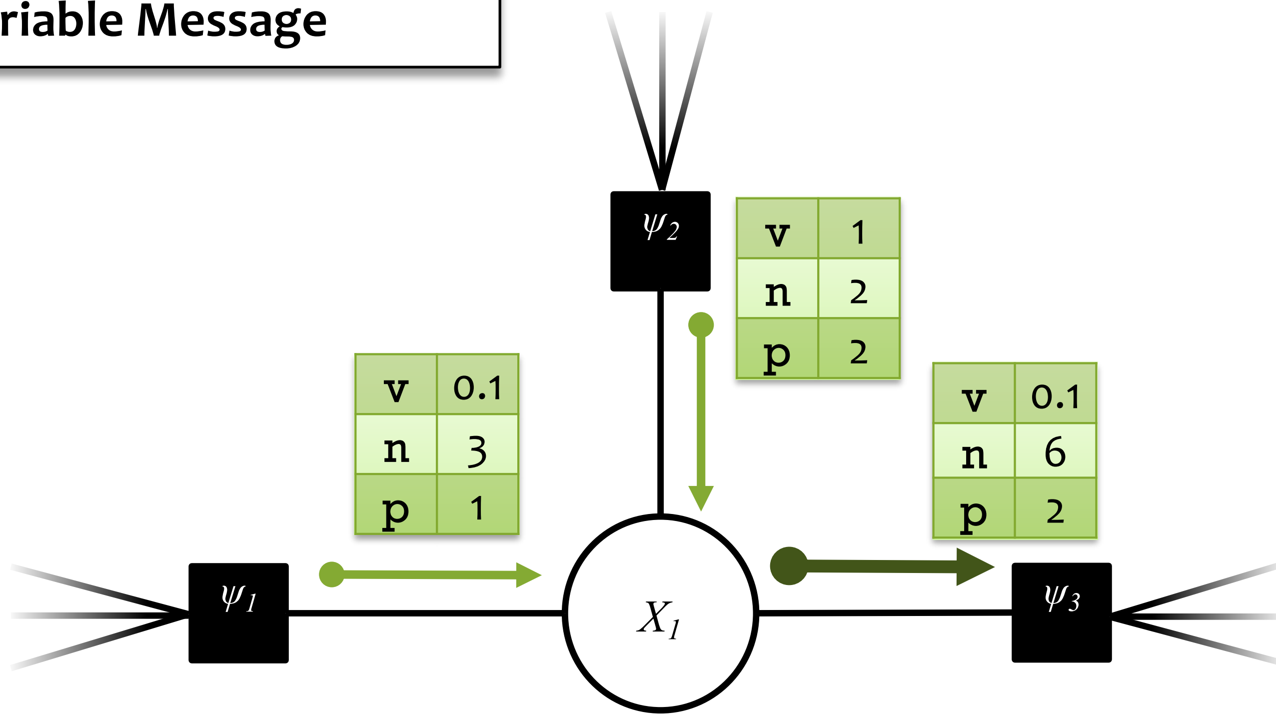
Variable Belief



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

Sum-Product Belief Propagation

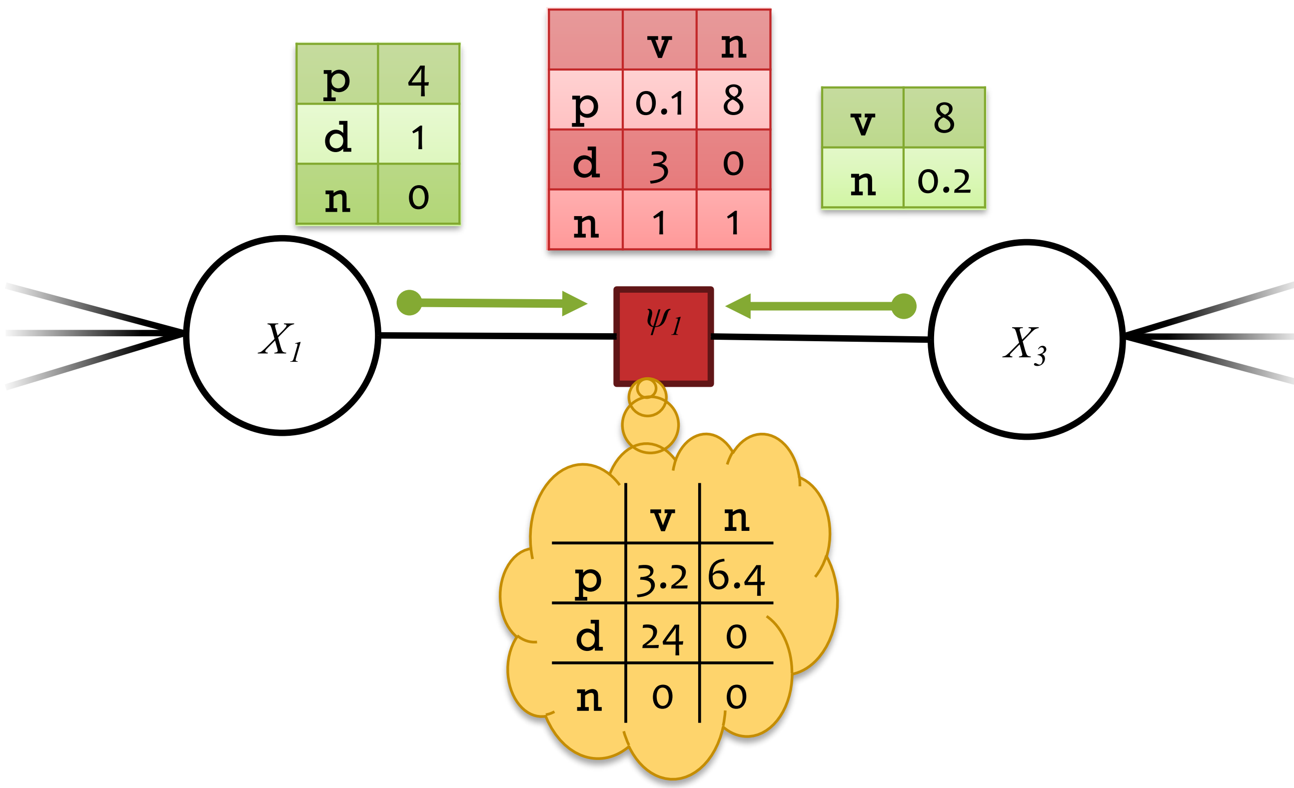
Variable Message



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

Sum-Product Belief Propagation

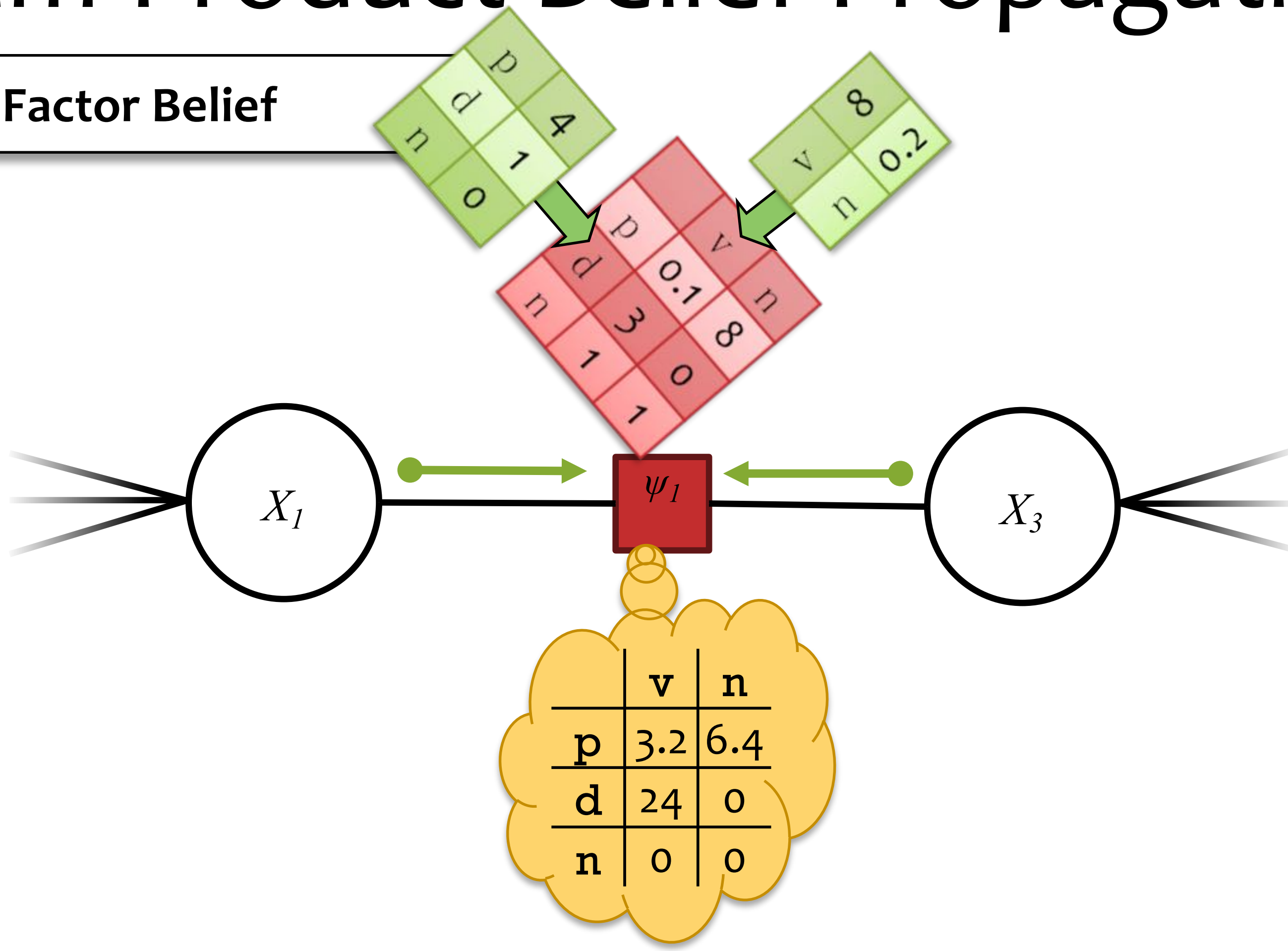
Factor Belief



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

Sum-Product Belief Propagation

Factor Belief



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

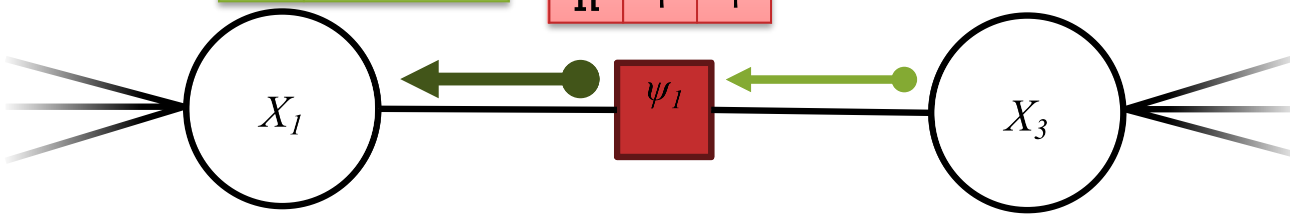
Sum-Product Belief Propagation

Factor Message

p	0.8 + 0.16
d	24 + 0
n	8 + 0.2

	v	n
p	0.1	8
d	3	0
n	1	1

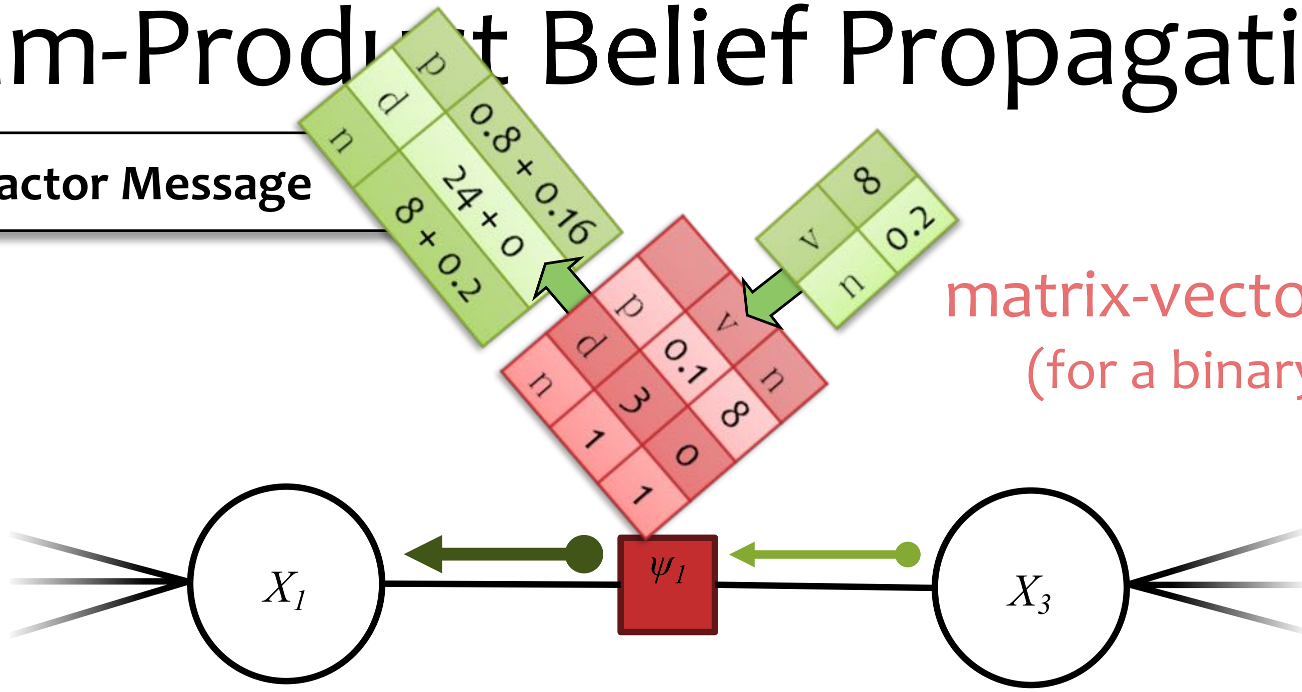
v	8
n	0.2



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

Sum-Product Belief Propagation

Factor Message



matrix-vector product
(for a binary factor)

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

Sum-Product Belief Propagation

Input: a factor graph with no cycles

Output: exact marginals for each variable and factor

Algorithm:

1. Initialize the messages to the uniform distribution.

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

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

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

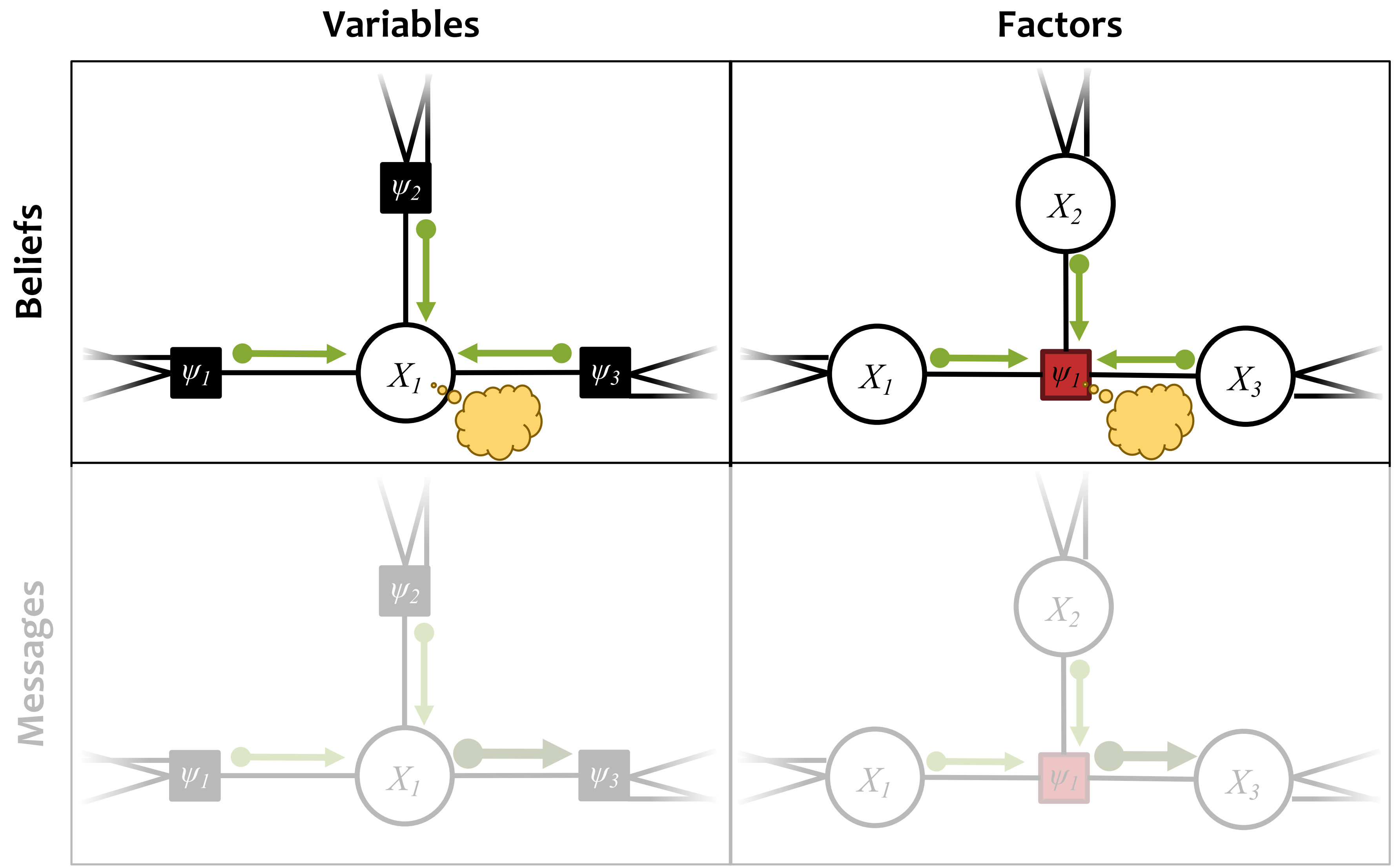
1. Compute the beliefs (unnormalized marginals).

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

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

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

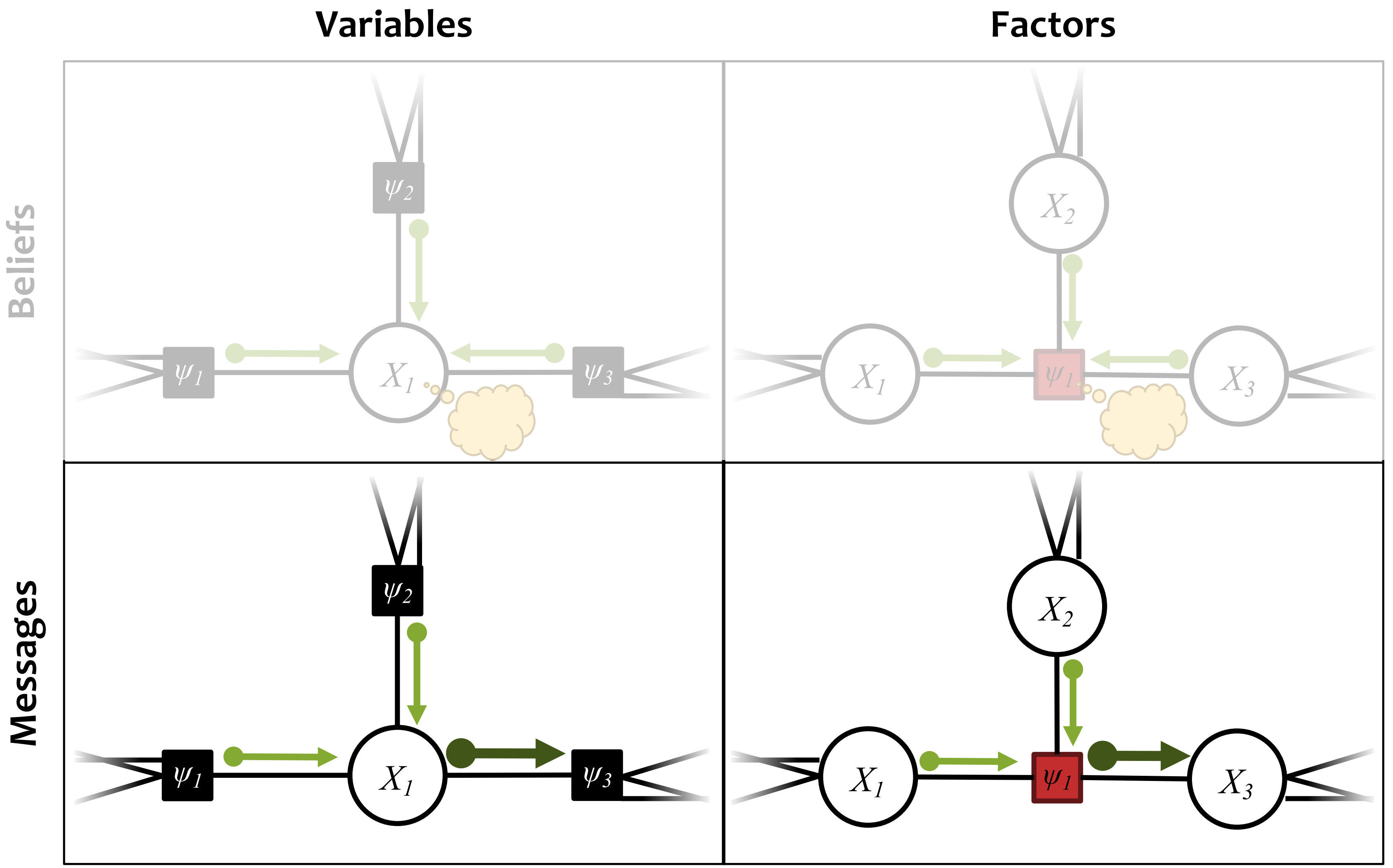
Sum-Product Belief Propagation



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

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

Sum-Product Belief Propagation

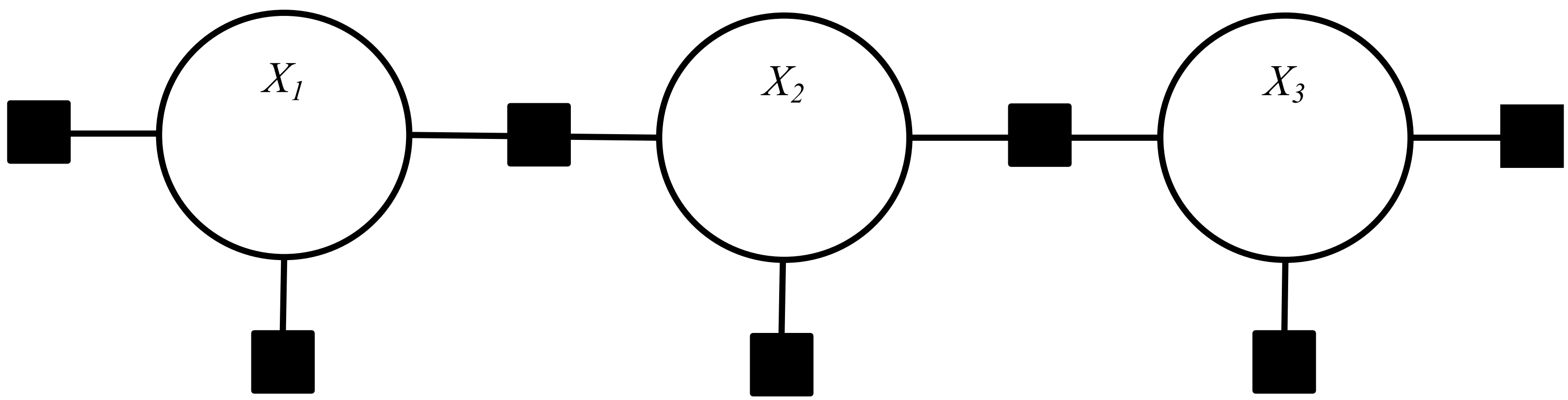


$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

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

FORWARD BACKWARD AS SUM-PRODUCT BP

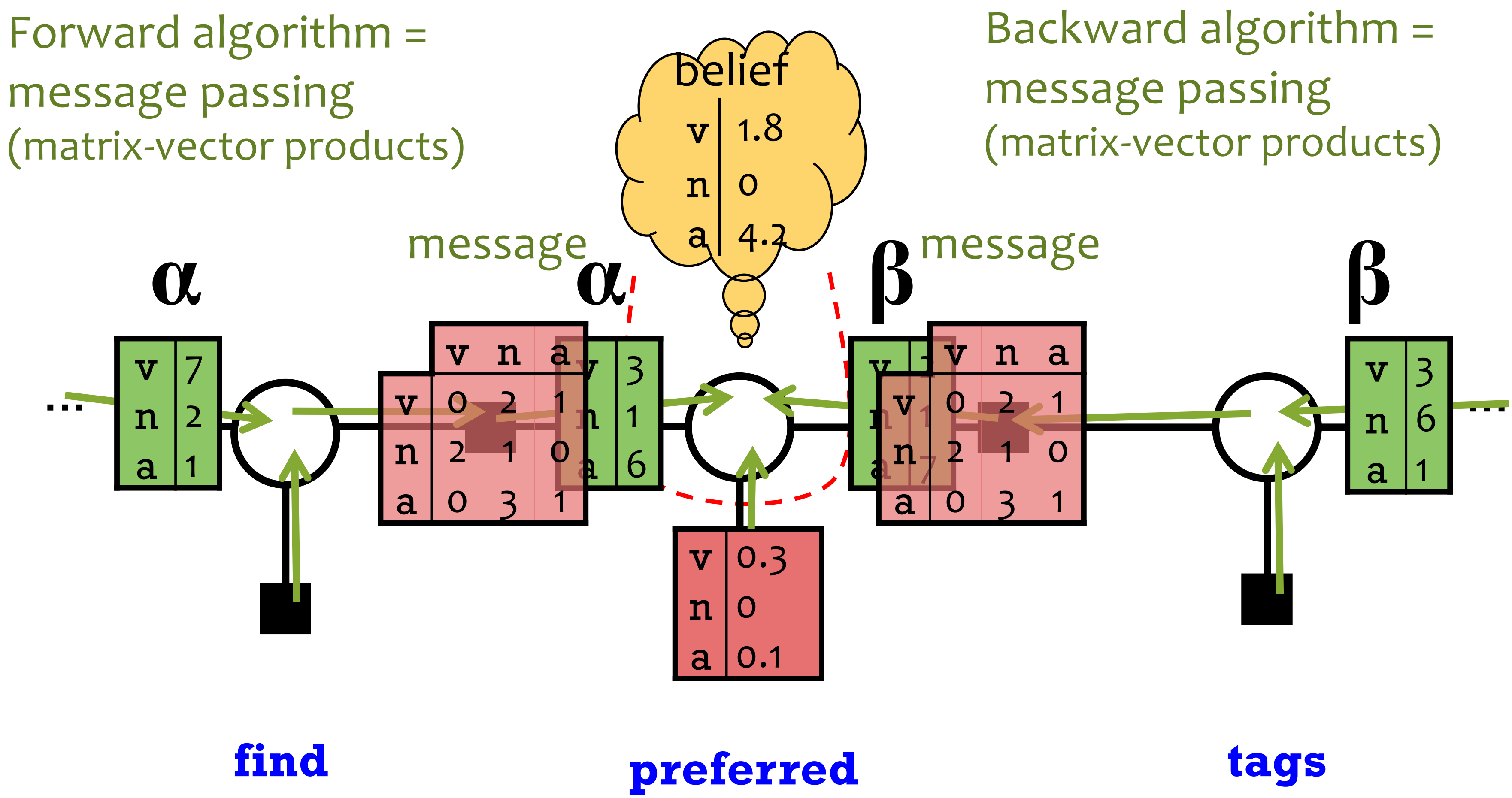
CRF Tagging Model



find | preferred | tags

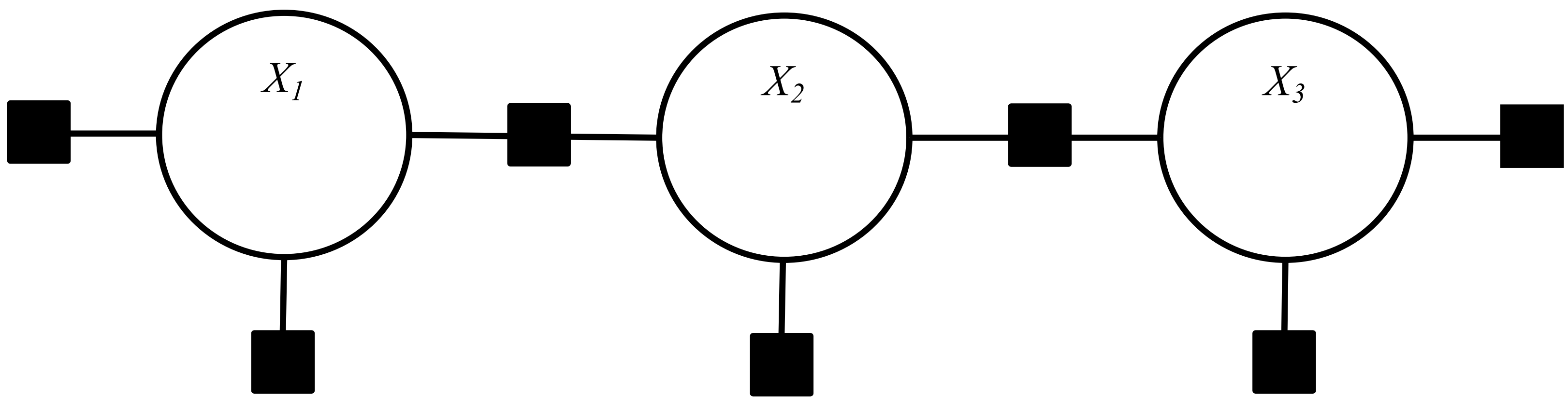
Could be verb or noun | *Could be adjective or verb* | *Could be noun or verb*

CRF Tagging by Belief Propagation



- Forward-backward is a message passing algorithm.
- It's the simplest case of belief propagation.

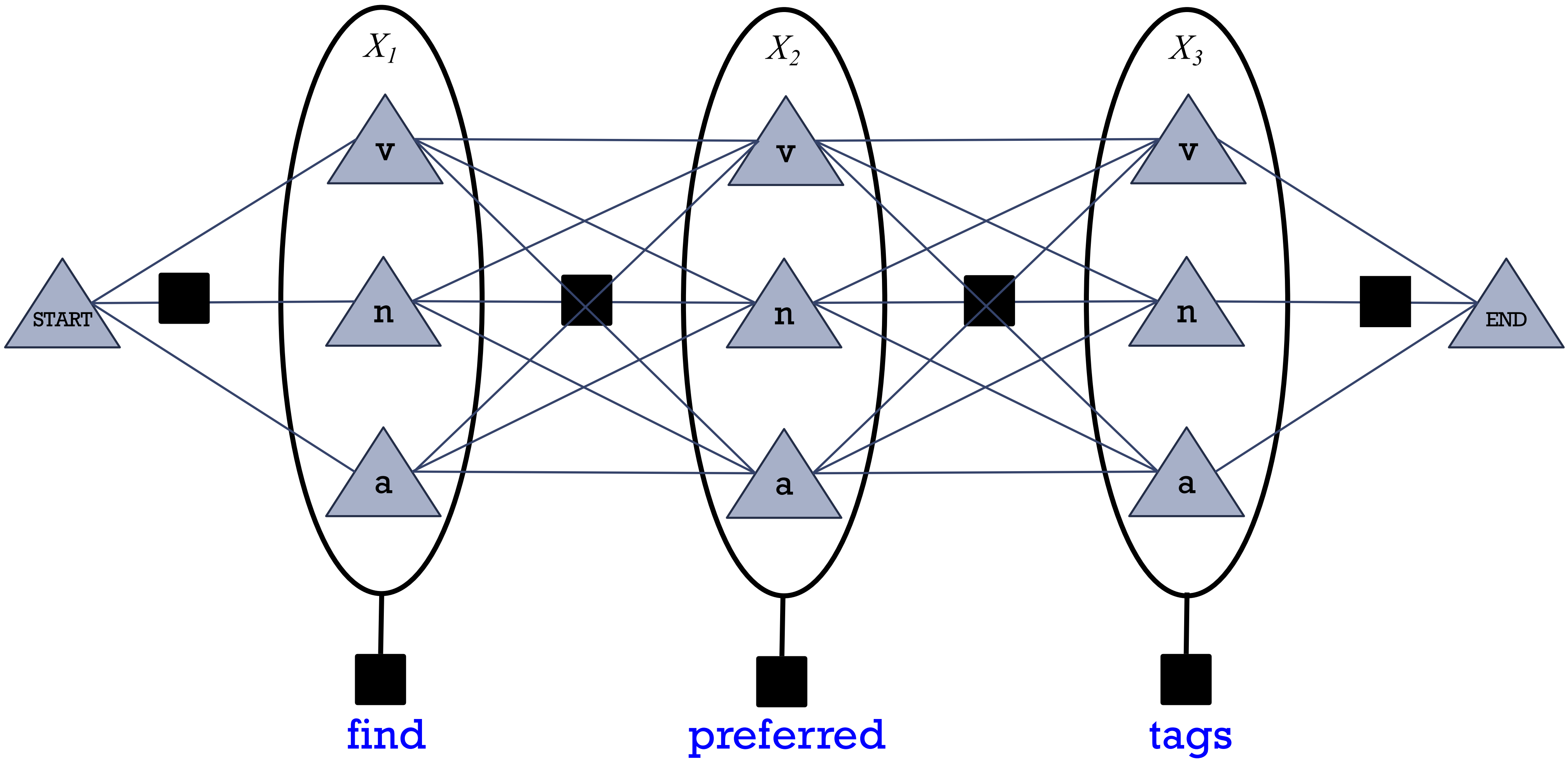
So Let's Review Forward-Backward …



find preferred tags

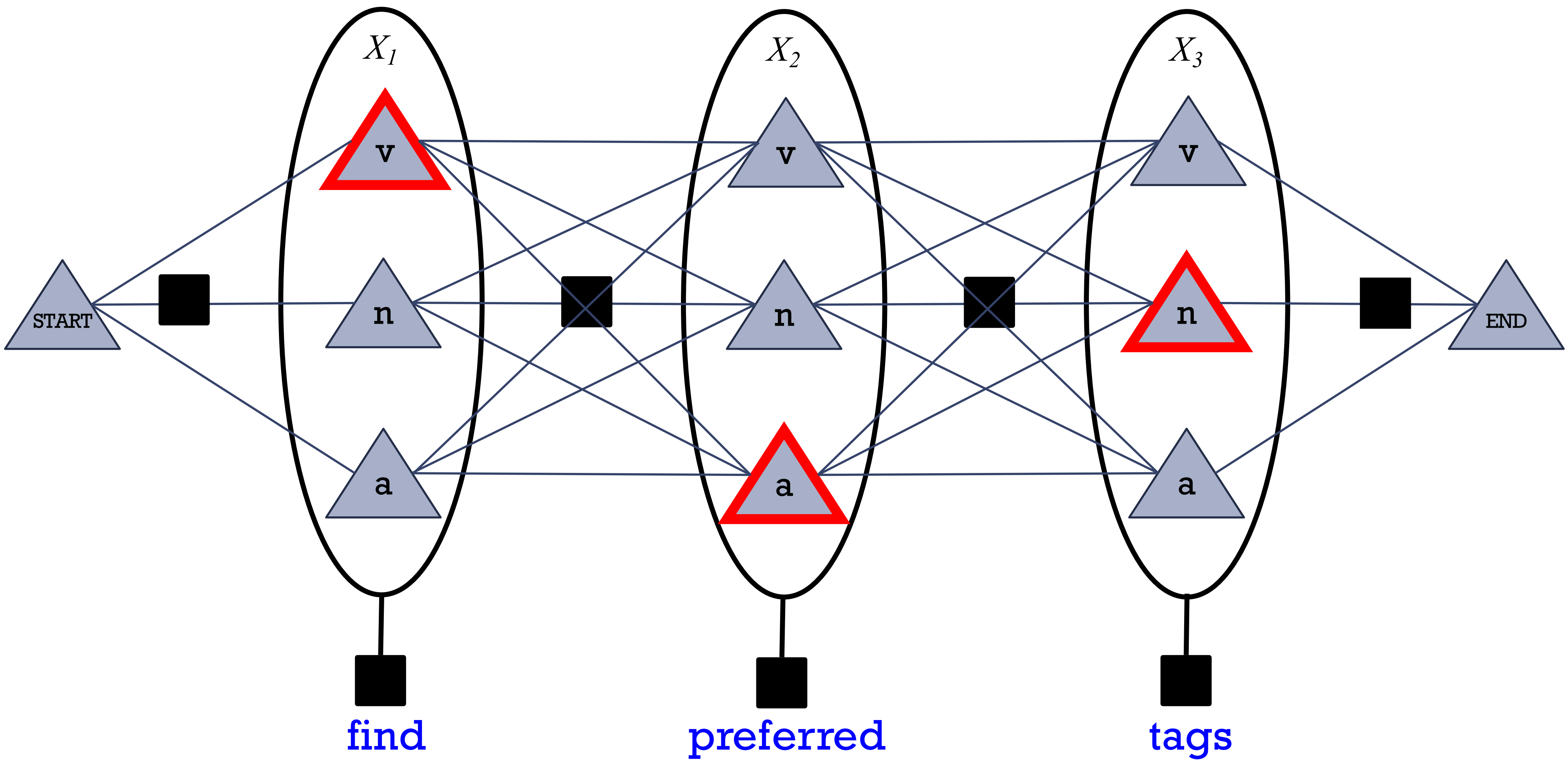
Could be verb or noun *Could be adjective or verb* *Could be noun or verb*

So Let's Review Forward-Backward …



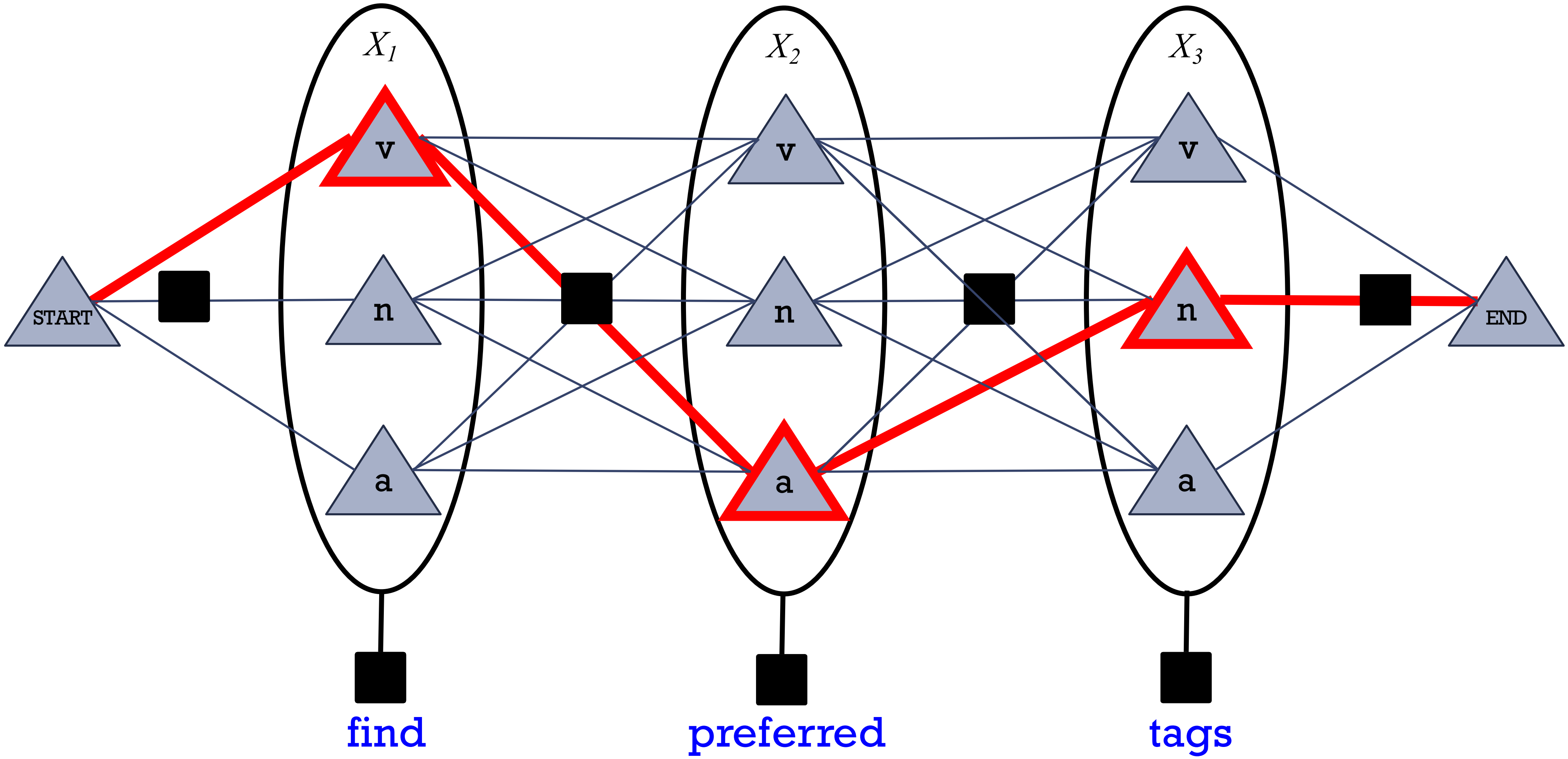
- Show the possible *values* for each variable

So Let's Review Forward-Backward …



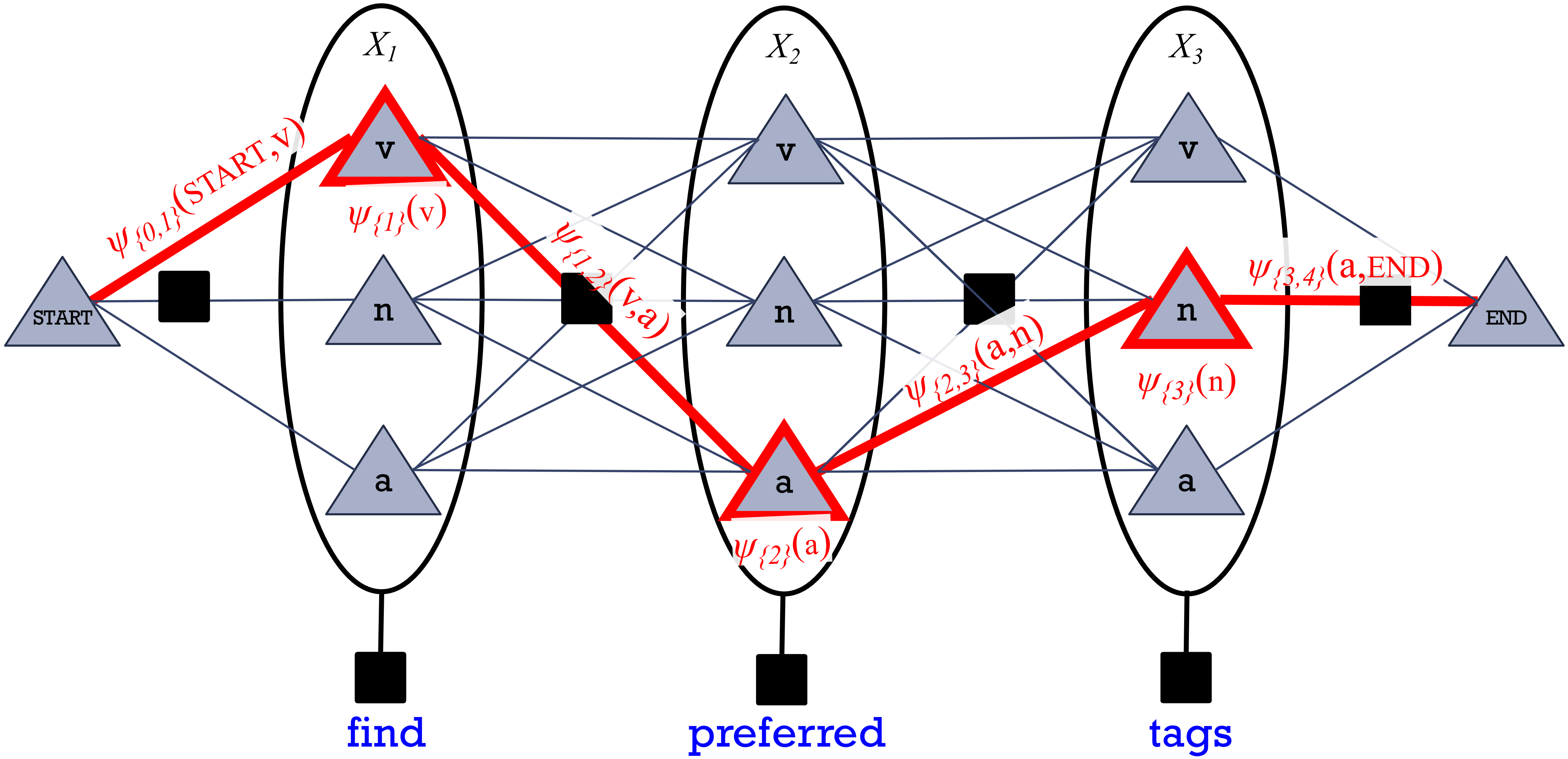
- Let's show the possible *values* for each variable
- One possible assignment

So Let's Review Forward-Backward …



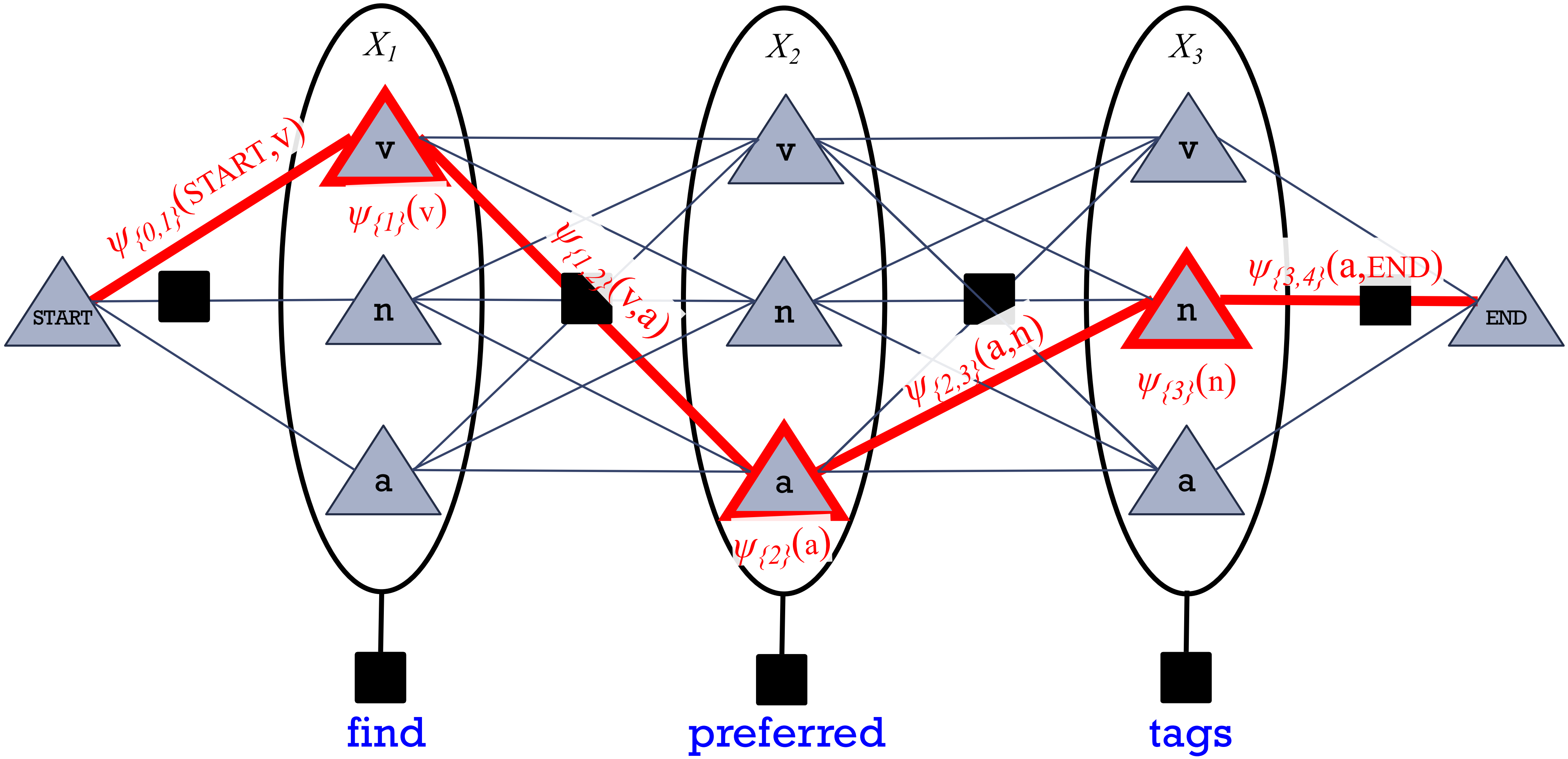
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



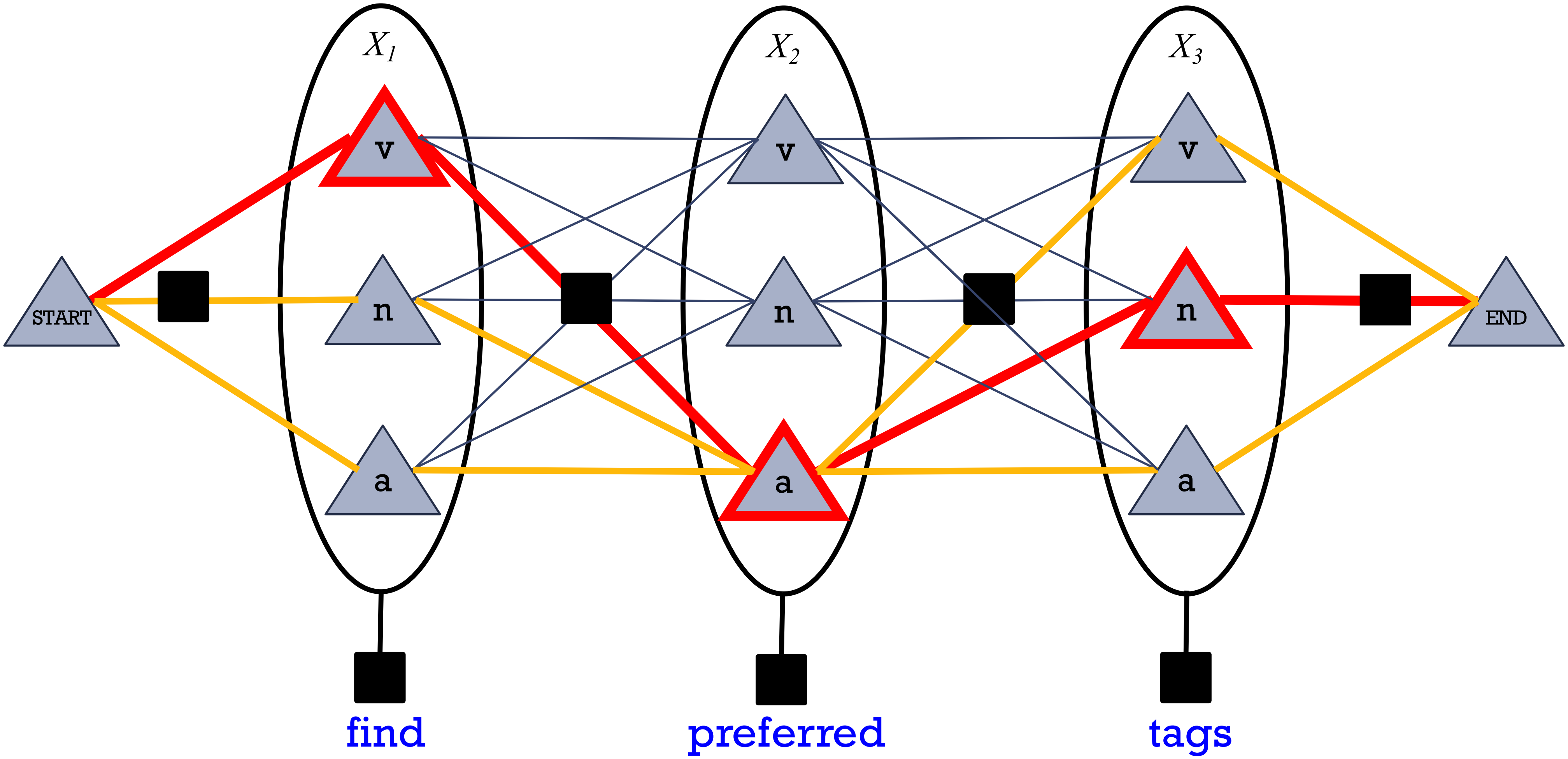
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

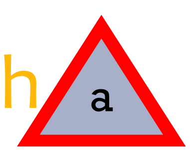
Viterbi Algorithm: Most Probable Assignment



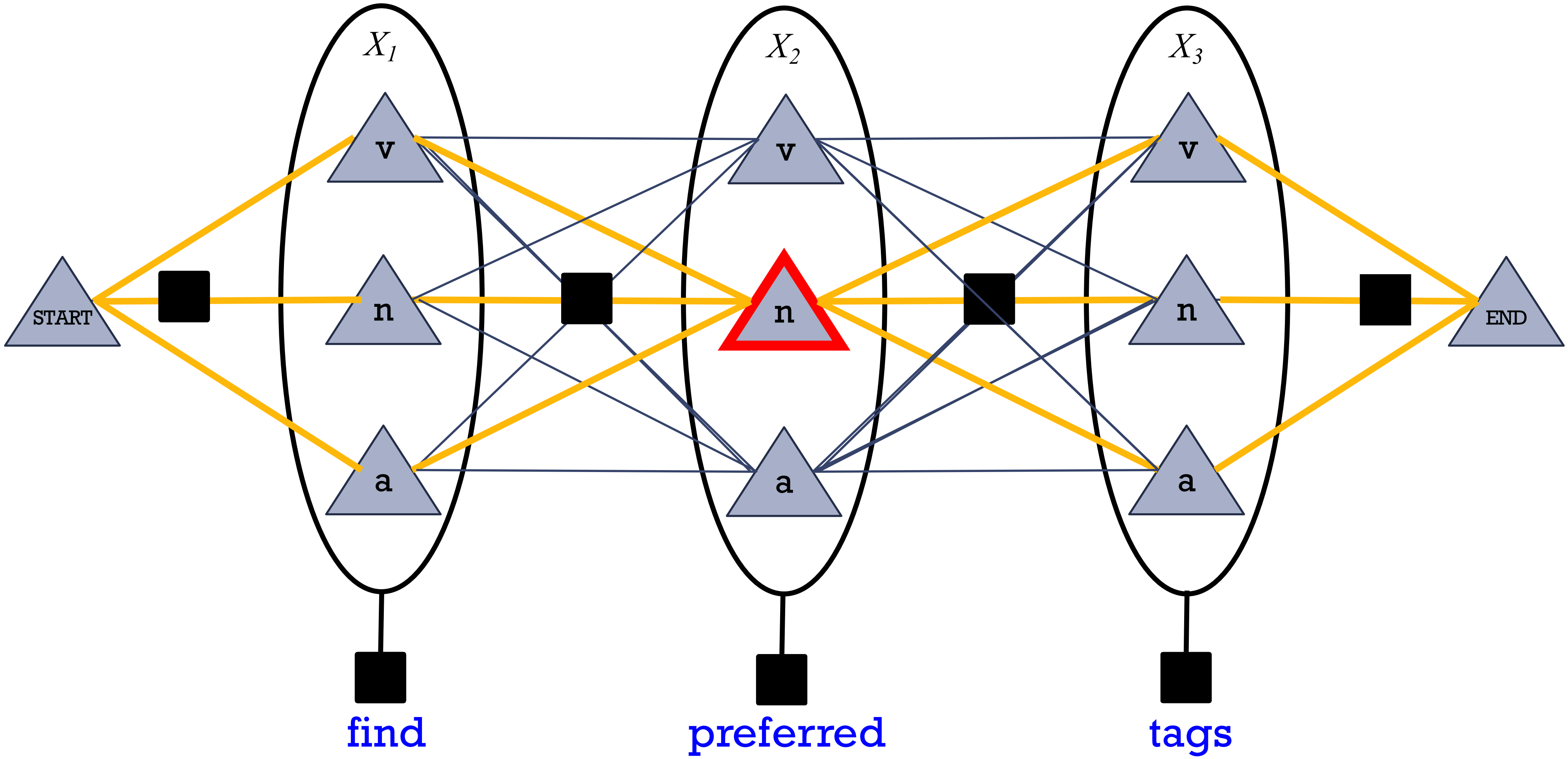
- So p(**v a n**) = (1/Z) * product weight of one path

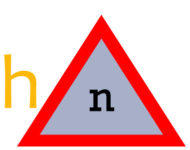
Forward-Backward Algorithm: Finds Marginals



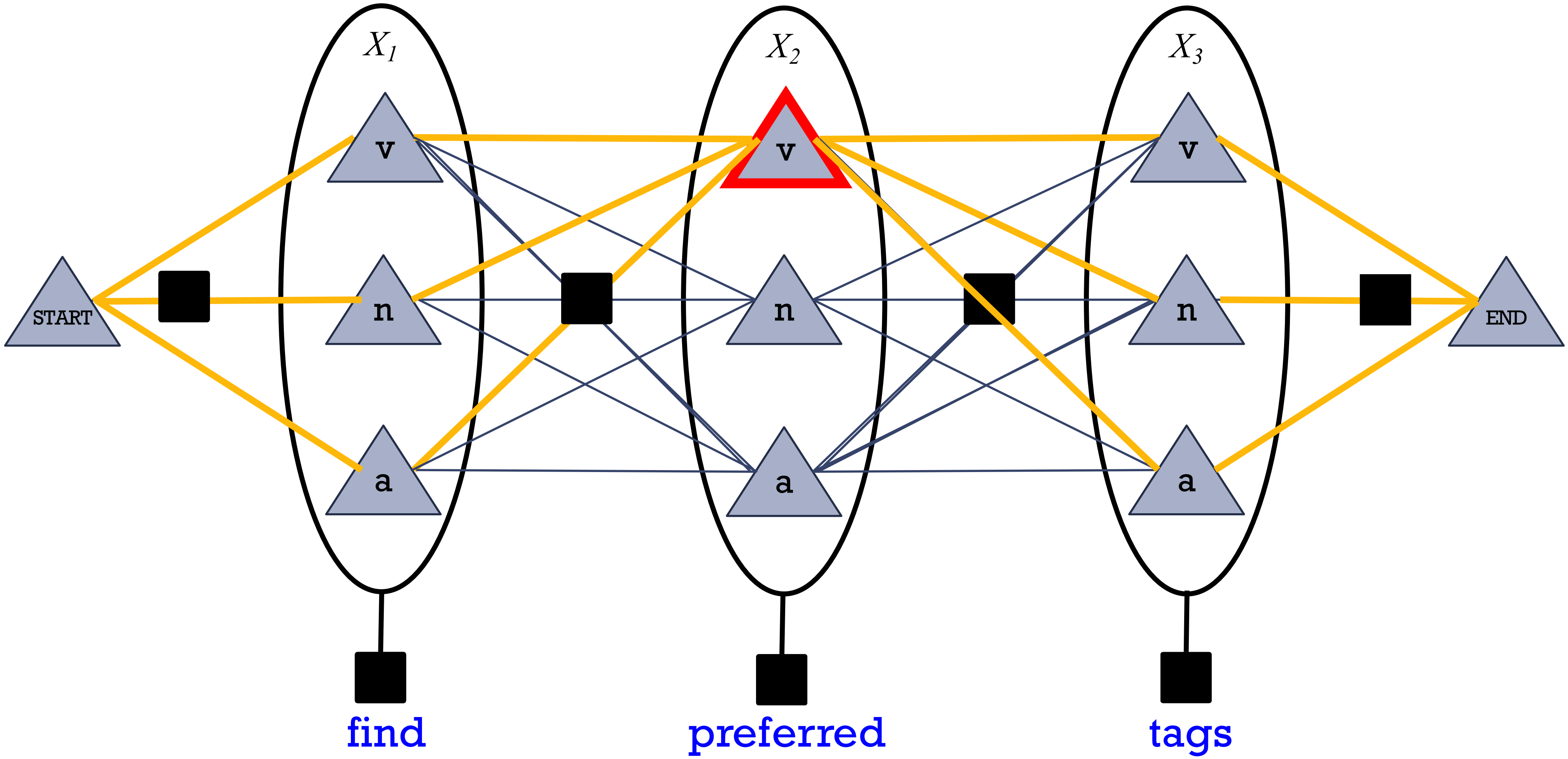
- So $p(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path
- Marginal probability $p(X_2 = \mathrm{a})$
  $= (1/Z)$ * total weight of *all* paths through 

Forward-Backward Algorithm: Finds Marginals



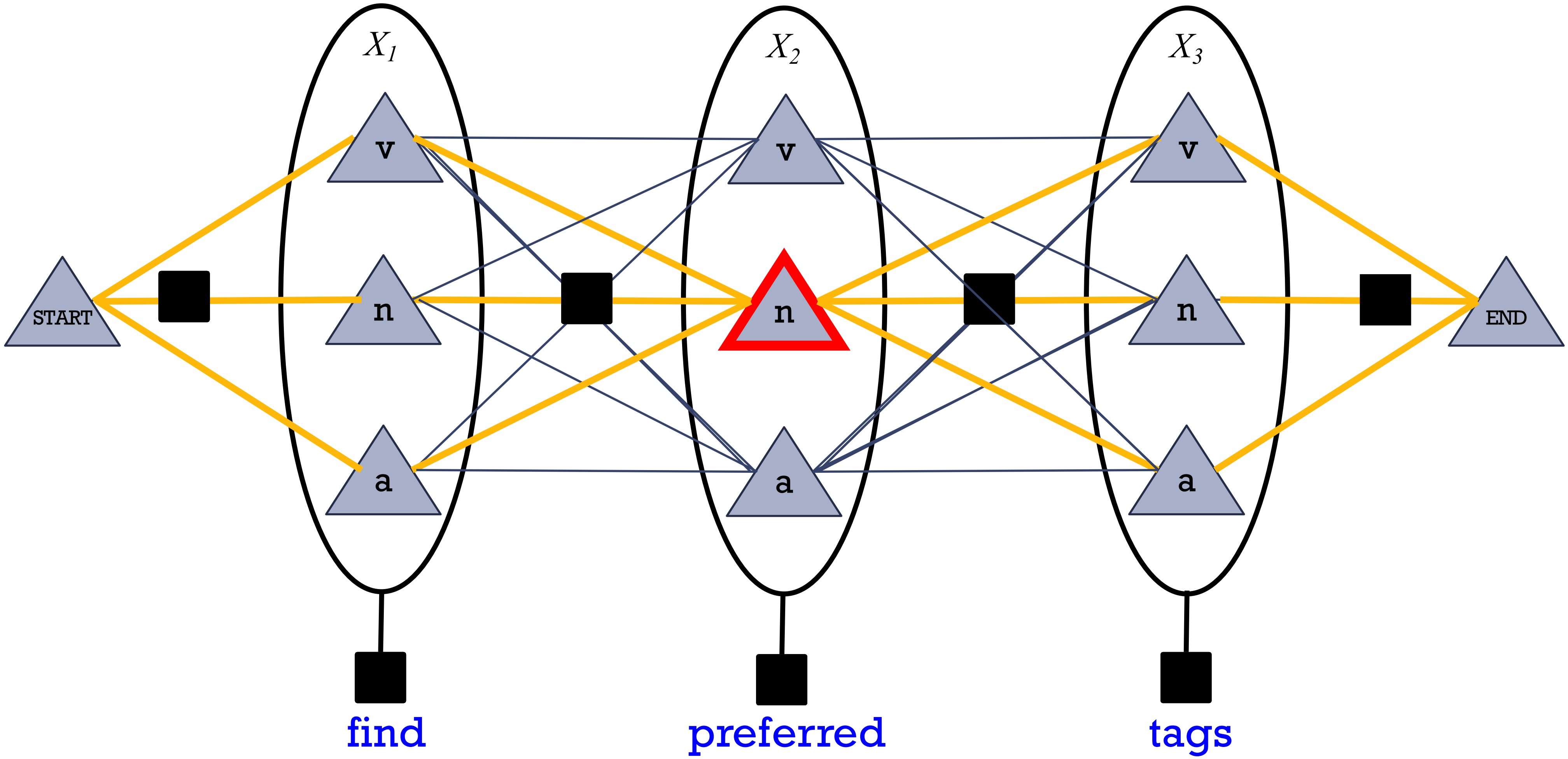
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = a)$
  = (1/Z) * total weight of *all* paths through 

Forward-Backward Algorithm: Finds Marginals



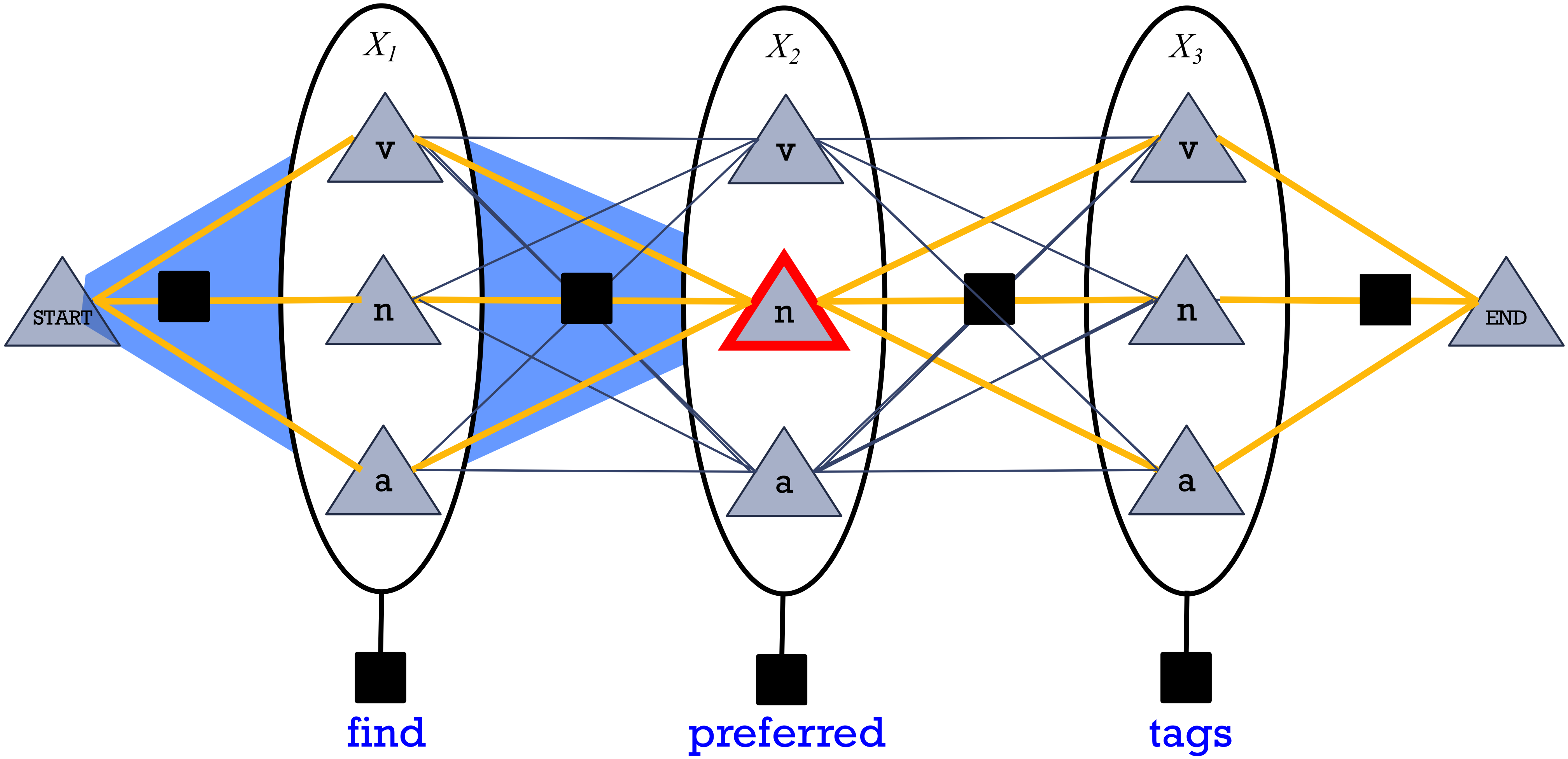
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through v

Forward-Backward Algorithm: Finds Marginals



- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = \text{a})$
  = (1/Z) * total weight of *all* paths through **n**

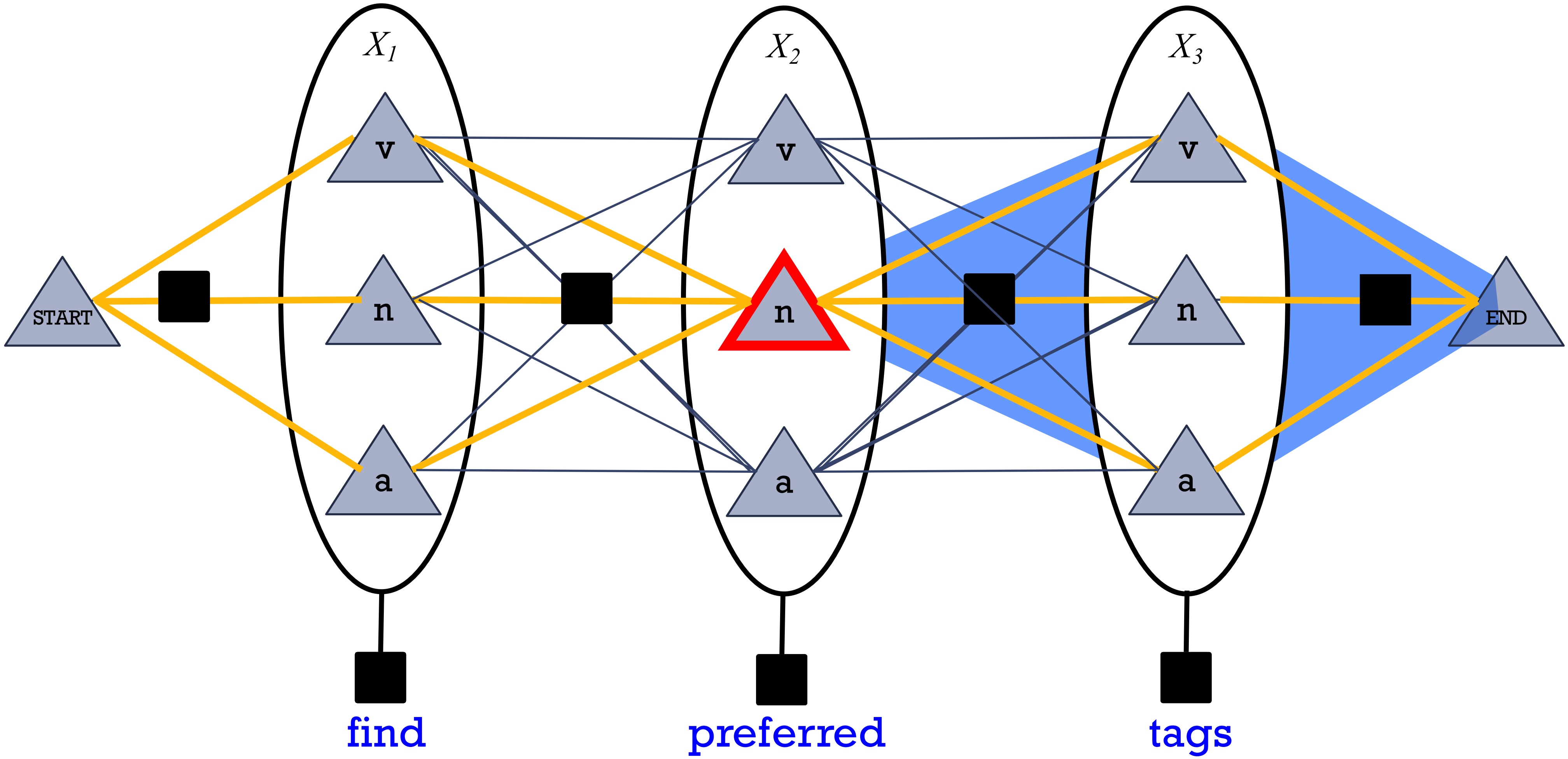
Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes*

(found by dynamic programming: matrix-vector products)

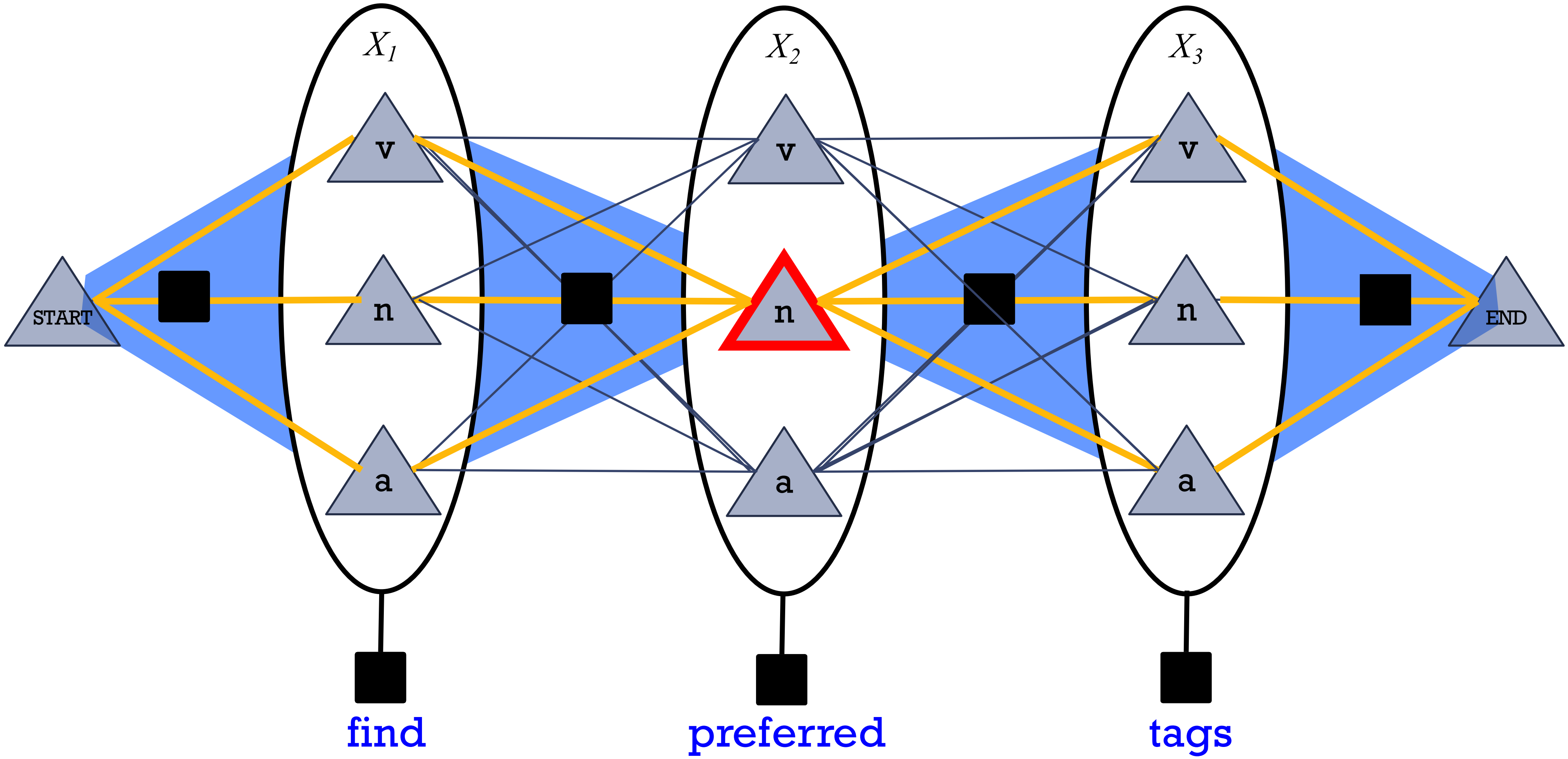
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = total weight of these path *suffixes*

(found by dynamic programming: matrix-vector products)

Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes* (a + b + c)

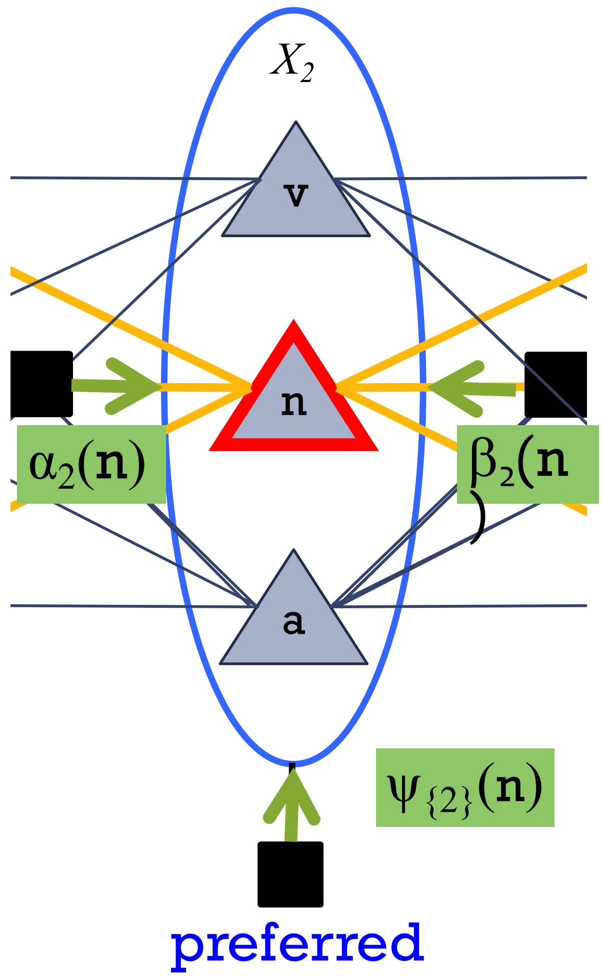
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

Product gives ax+ay+az+bx+by+bz+cx+cy+cz = total weight of paths

Forward-Backward Algorithm: Finds Marginals

Oops! The weight of a path through a state also includes a weight at that state.
So $\alpha(\mathbf{n})\cdot\beta(\mathbf{n})$ isn't enough.

The extra weight is the opinion of the unigram factor at this variable.

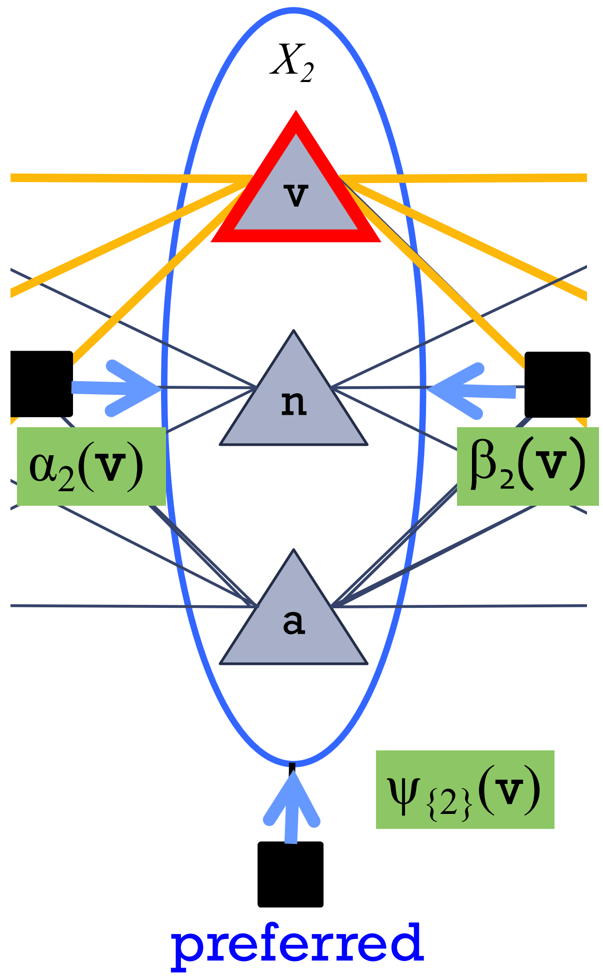


"belief that X_2 = **n**"

total weight of *all* paths through **n**

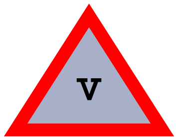
$= \alpha_2(\mathbf{n})\ \psi_{\{2\}}(\mathbf{n})\ \beta_2(\mathbf{n})$

Forward-Backward Algorithm: Finds Marginals



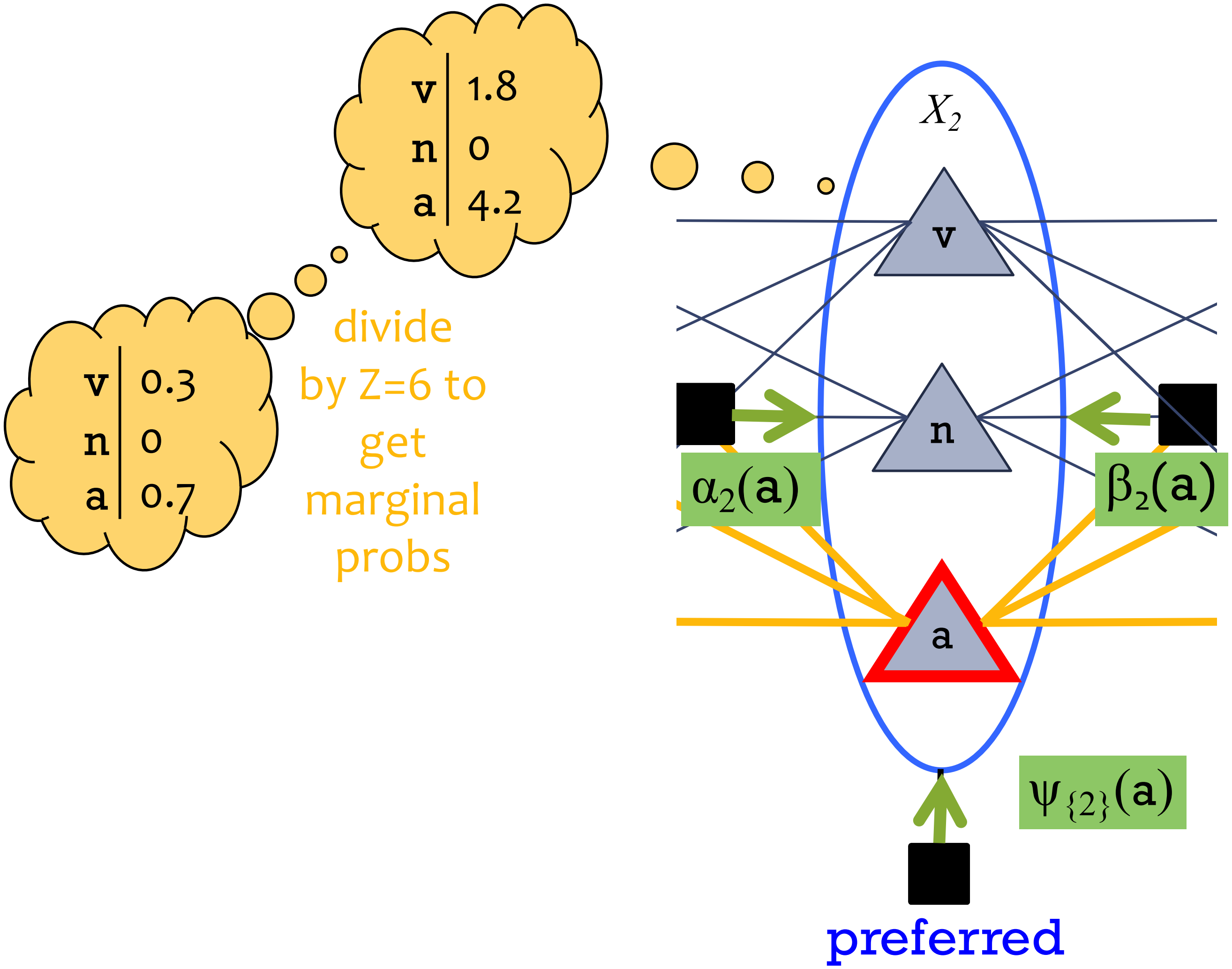
"belief that $X_2 = \mathbf{v}$"

"belief that $X_2 = \mathbf{n}$"

total weight of *all* paths through 

= $\alpha_2(\mathbf{v})$ $\psi_{\{2\}}(\mathbf{v})$ $\beta_2(\mathbf{v})$

Forward-Backward Algorithm: Finds Marginals



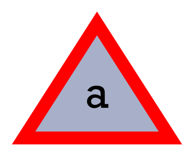
“belief that X_2 = **v**”

“belief that X_2 = **n**”

“belief that X_2 = **a**”

---

sum = Z
(total probability of *all* paths)

total weight of *all* paths through 

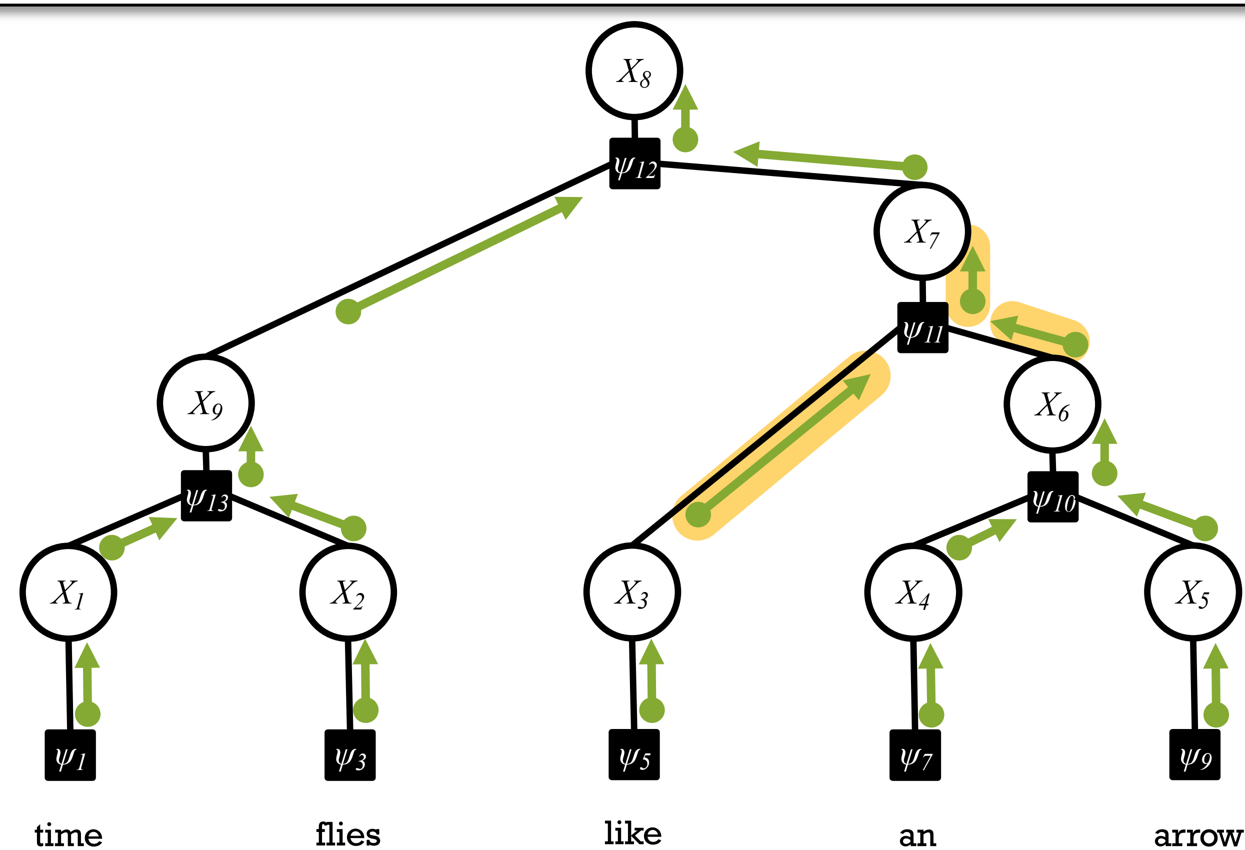
= $\alpha_2(\mathtt{a})$ $\psi_{\{2\}}(\mathtt{a})$ $\beta_2(\mathtt{a})$

BP AS DYNAMIC PROGRAMMING

(Acyclic) Belief Propagation

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.

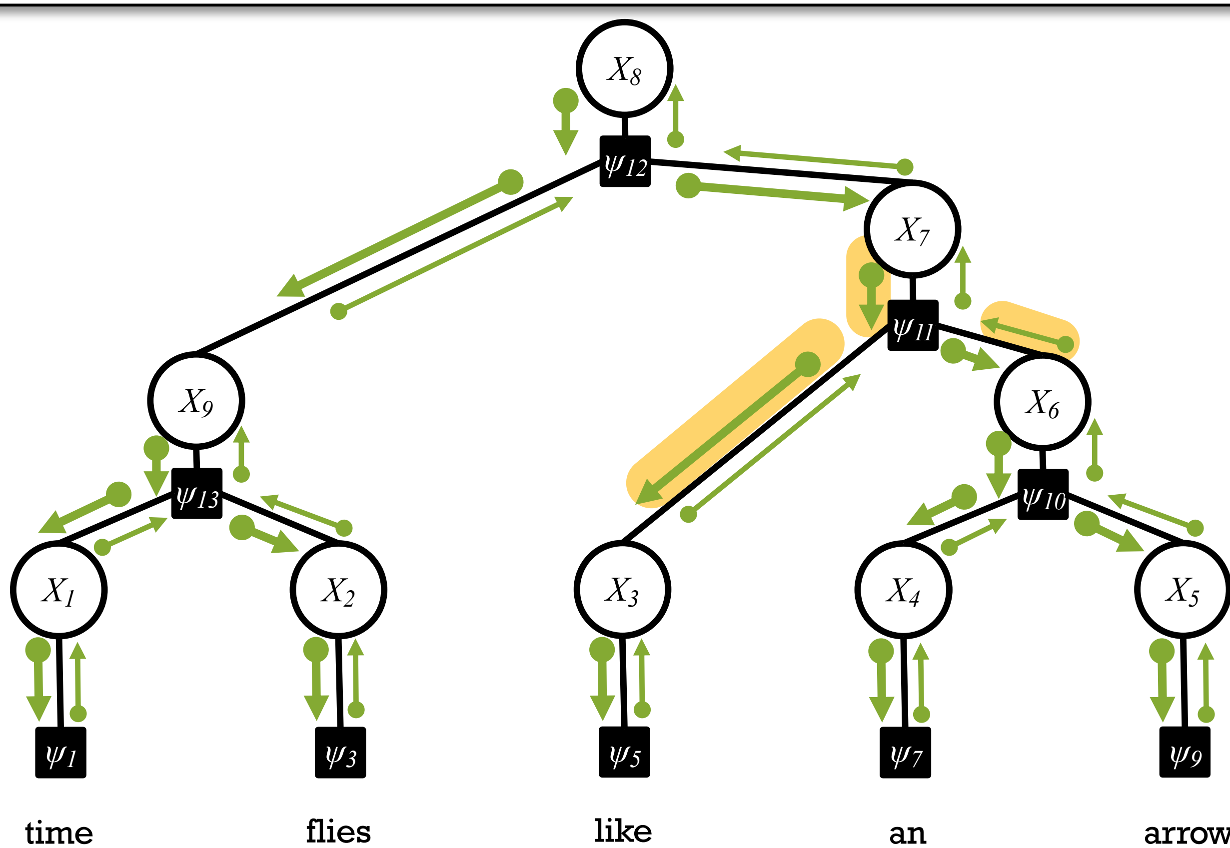


(Acyclic) Belief Propagation

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**.

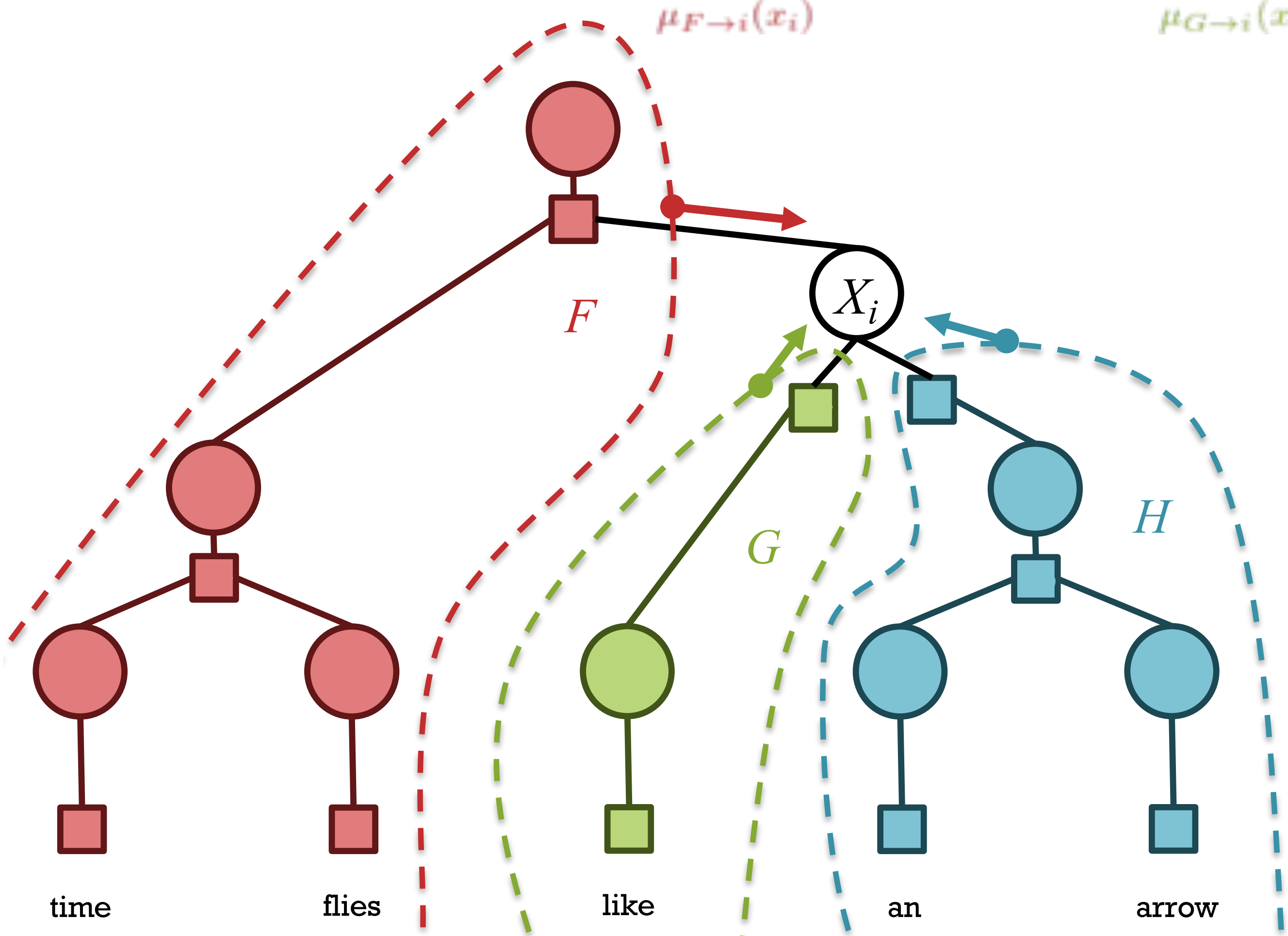
A node computes an outgoing message along an edge
only after it has received incoming messages along all its other edges.



Acyclic BP as Dynamic Programming

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

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



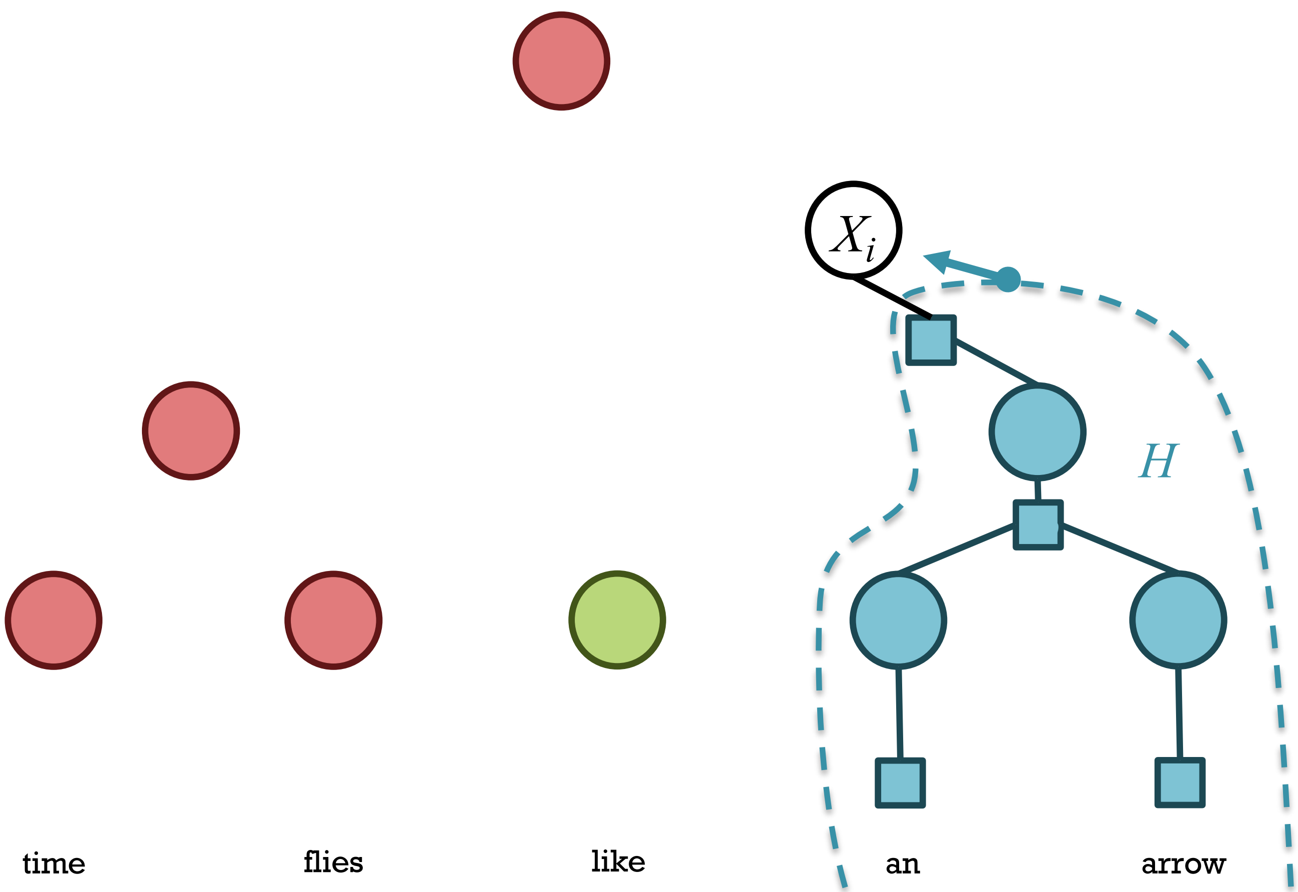
Subproblem:
Inference using just the factors in subgraph H

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

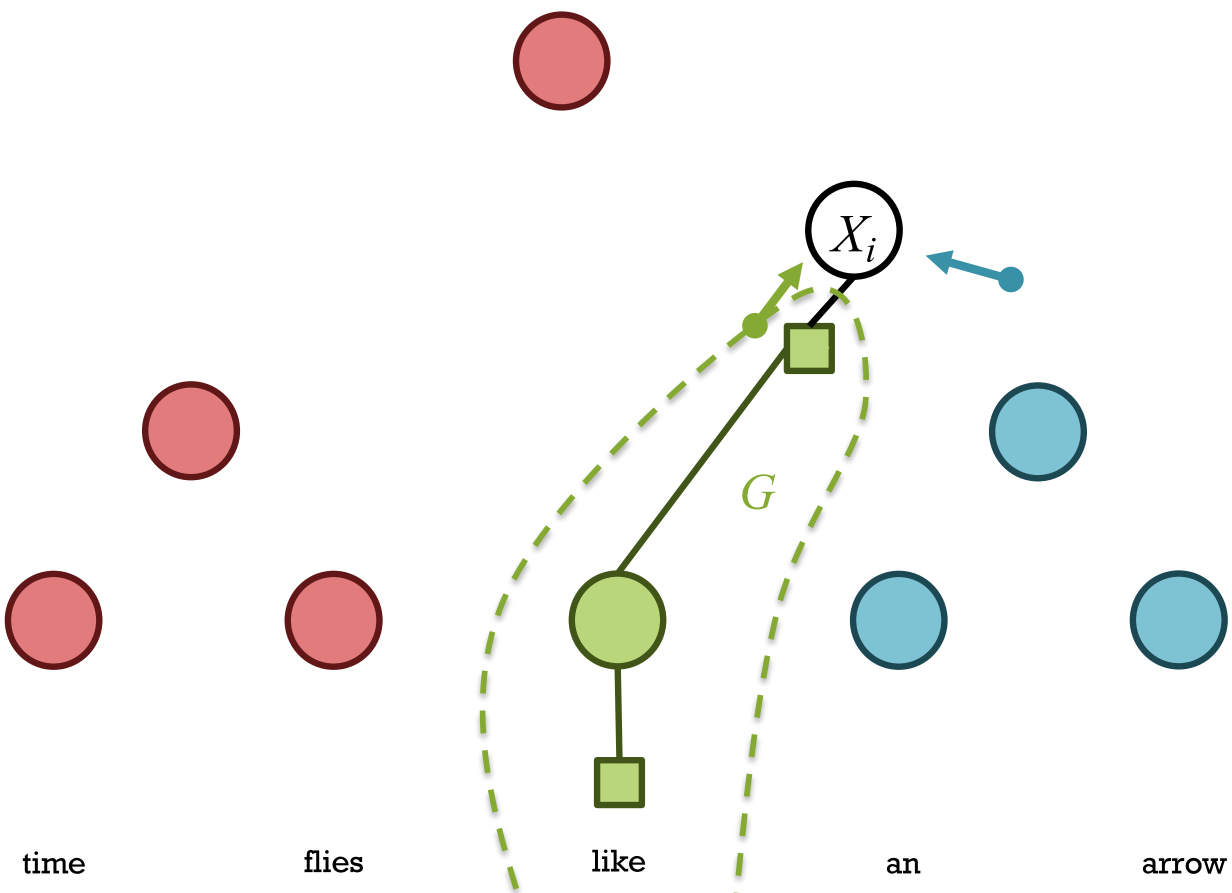
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

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

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



Subproblem:
Inference using just the factors in subgraph H

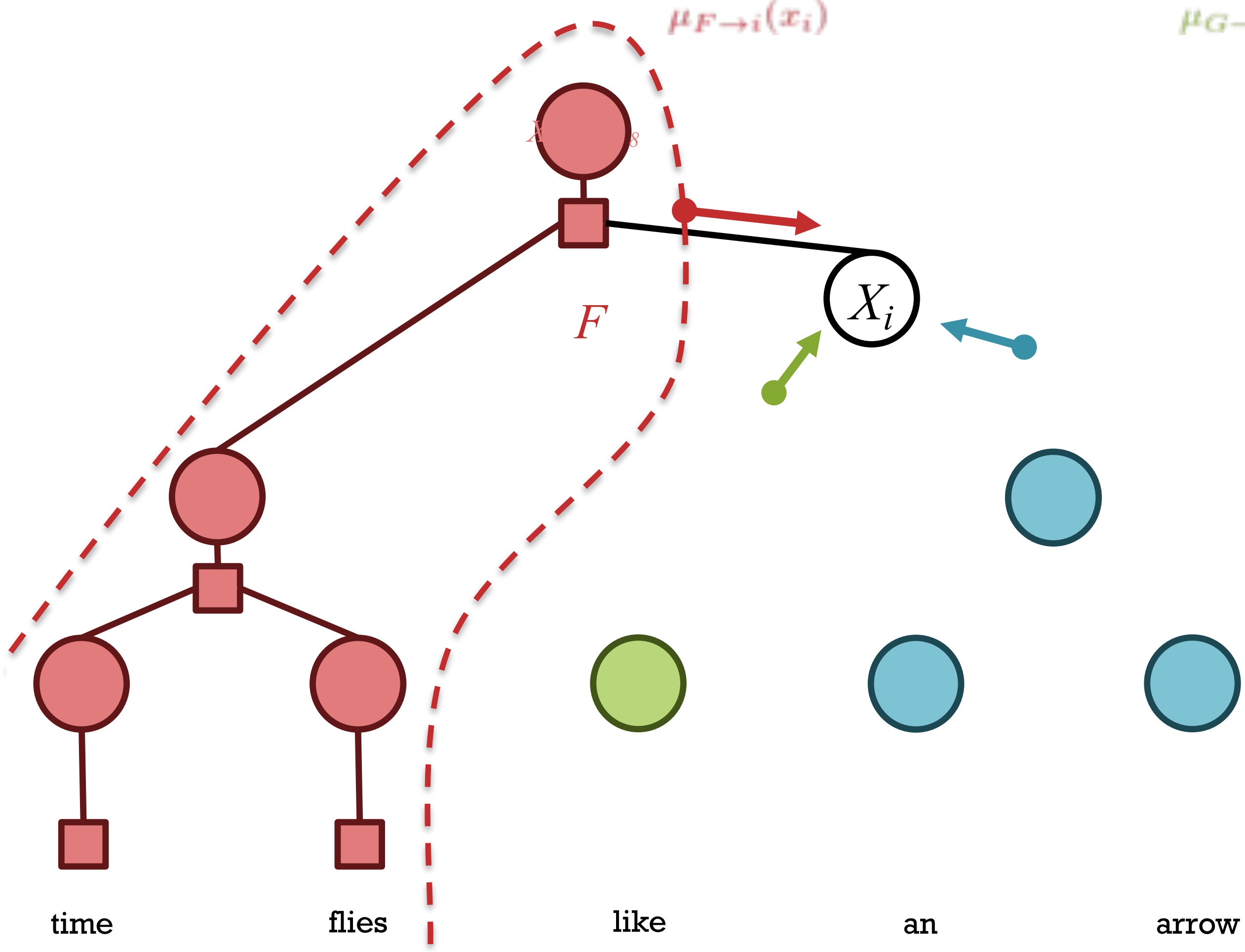
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

Message ***to*** *a variable*

Acyclic BP as Dynamic Programming

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

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



Subproblem:
Inference using just the factors in subgraph H

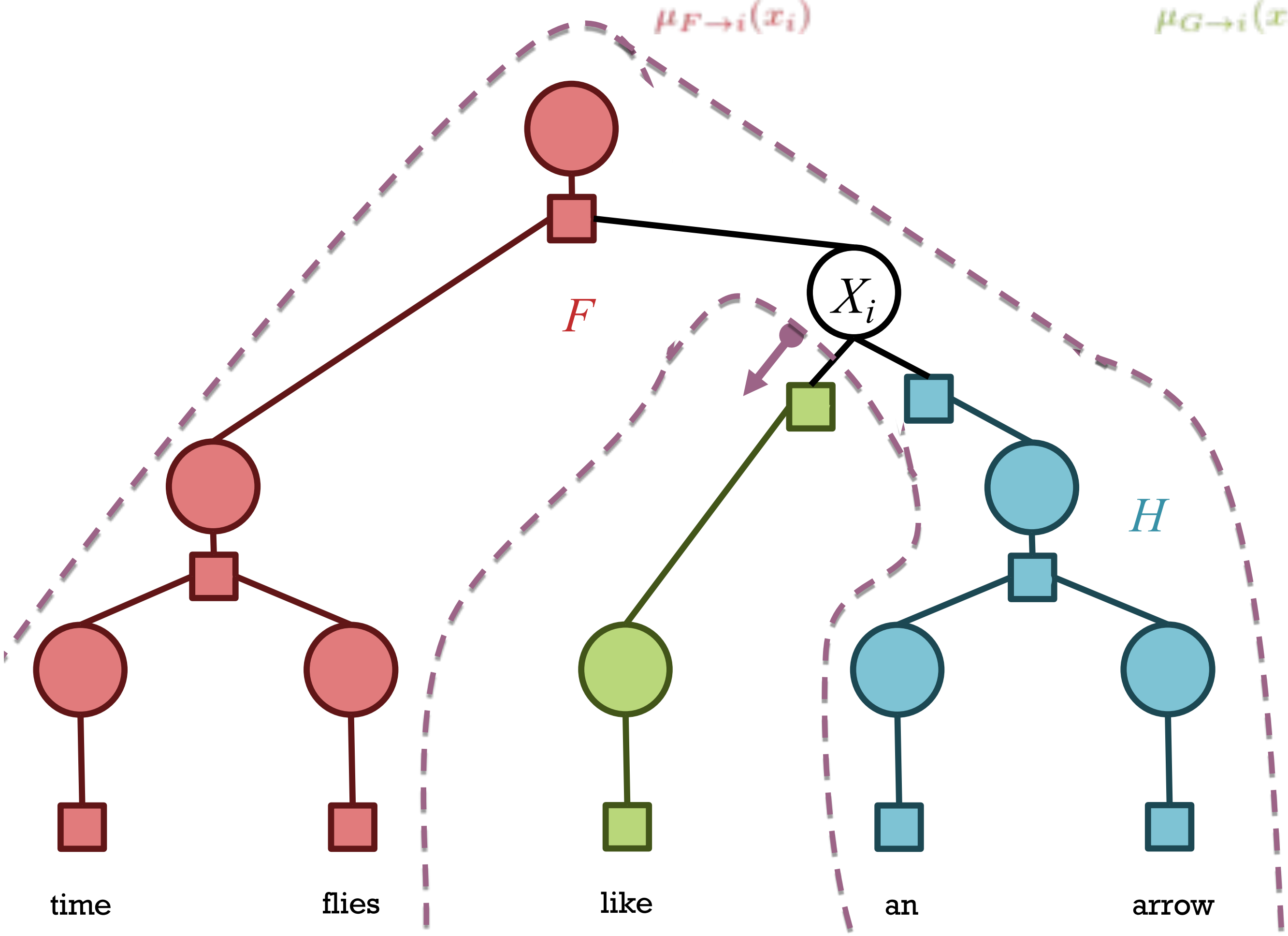
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

Message to a variable

Acyclic BP as Dynamic Programming

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

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



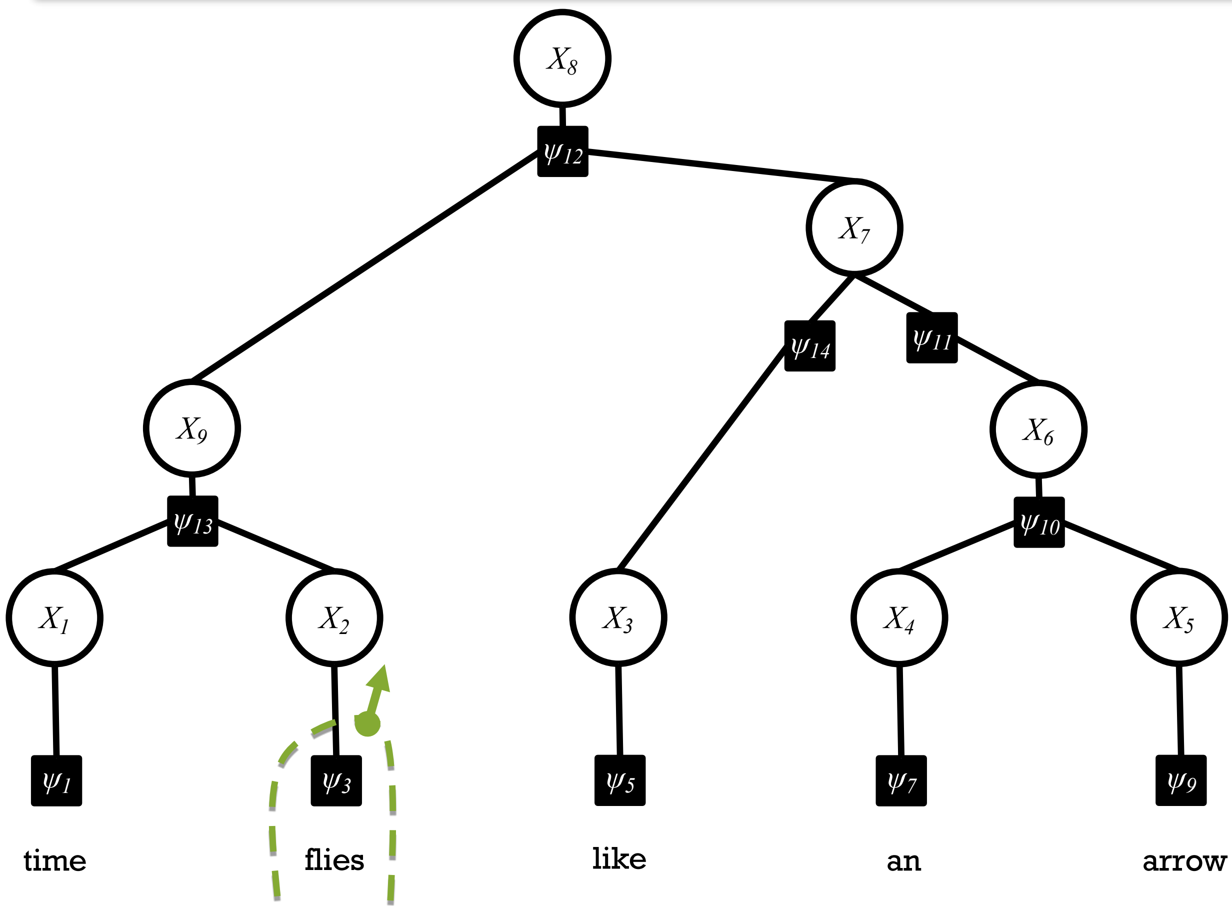
Subproblem:
Inference using just the factors in subgraph $F \cup H$

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*

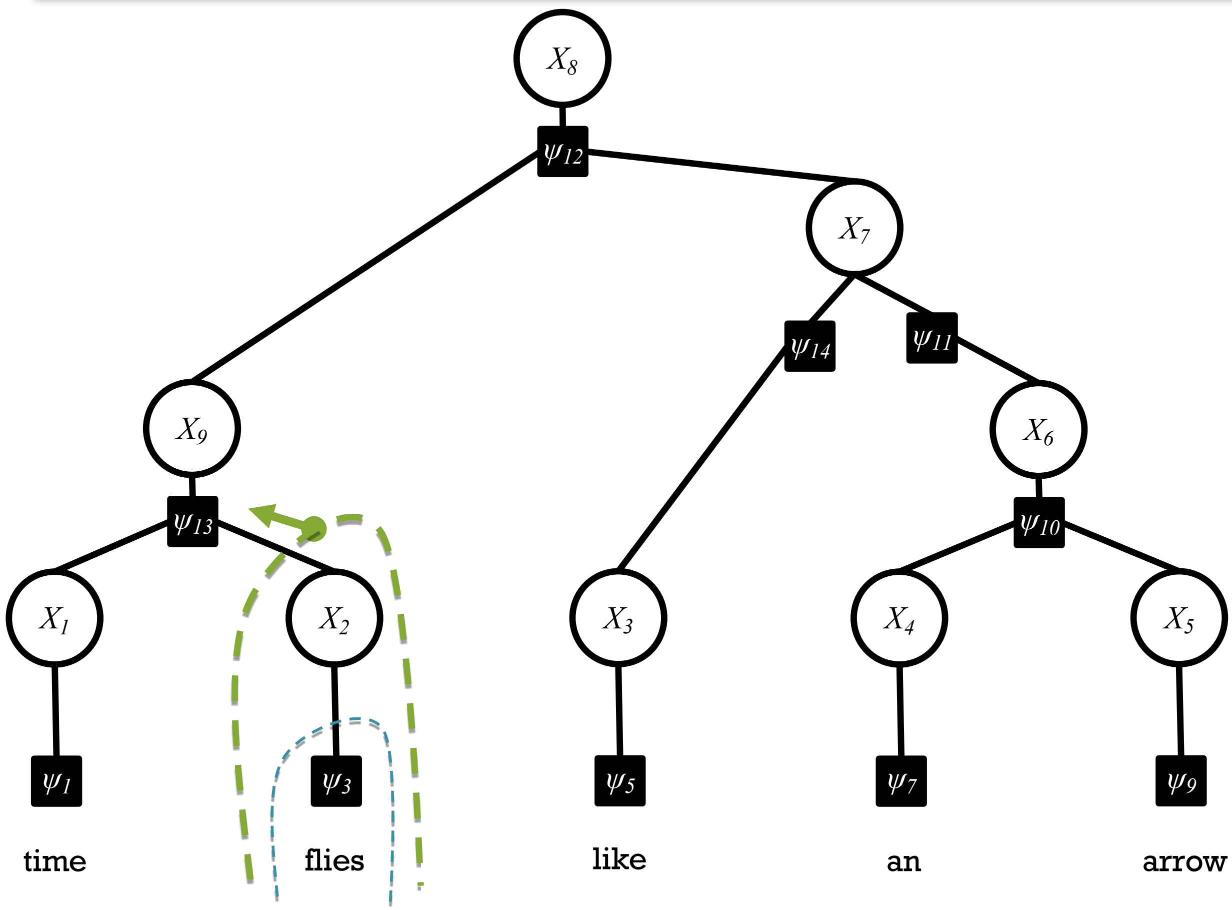
Acyclic BP as Dynamic Programming

- 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.



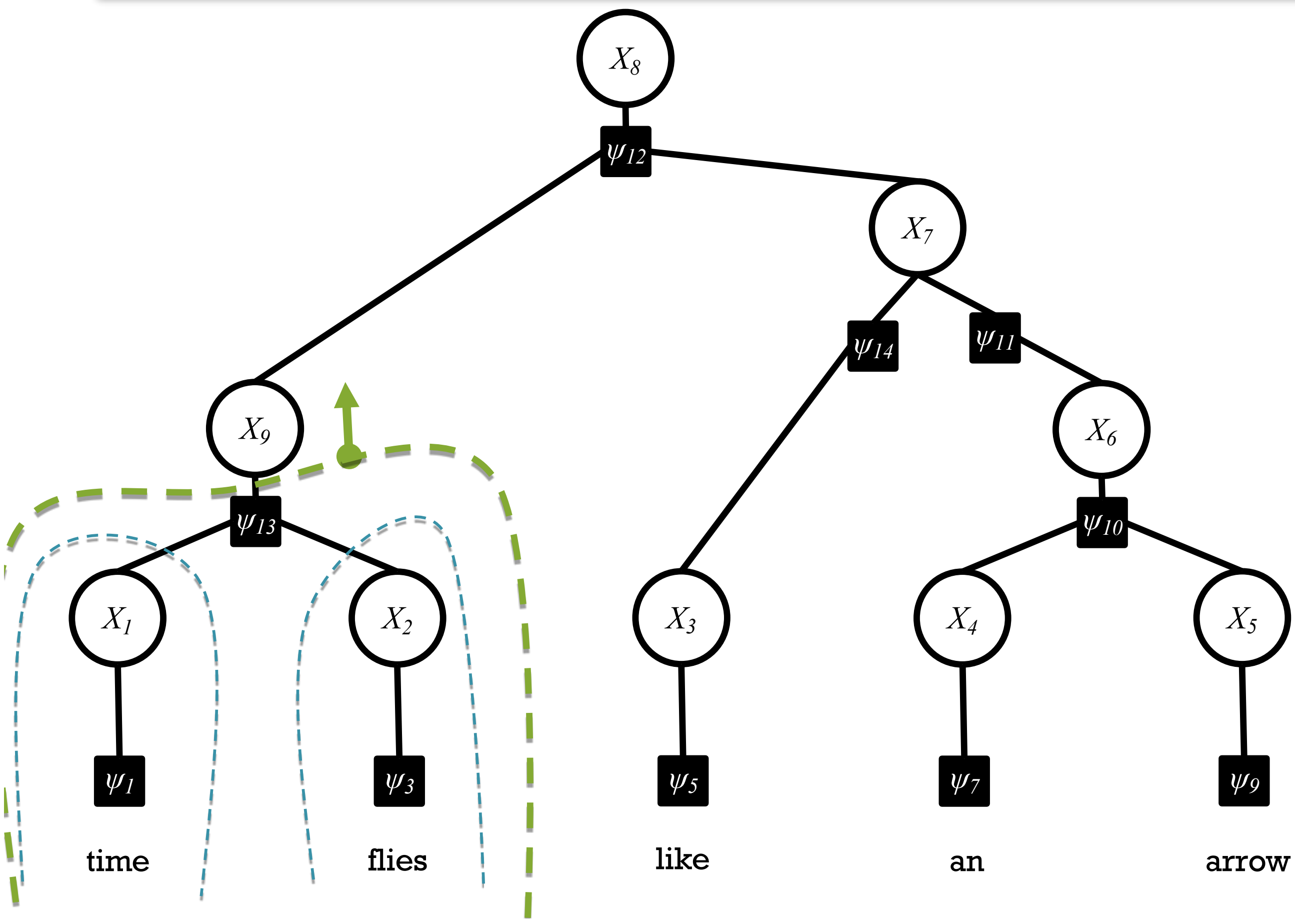
Acyclic BP as Dynamic Programming

- 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.



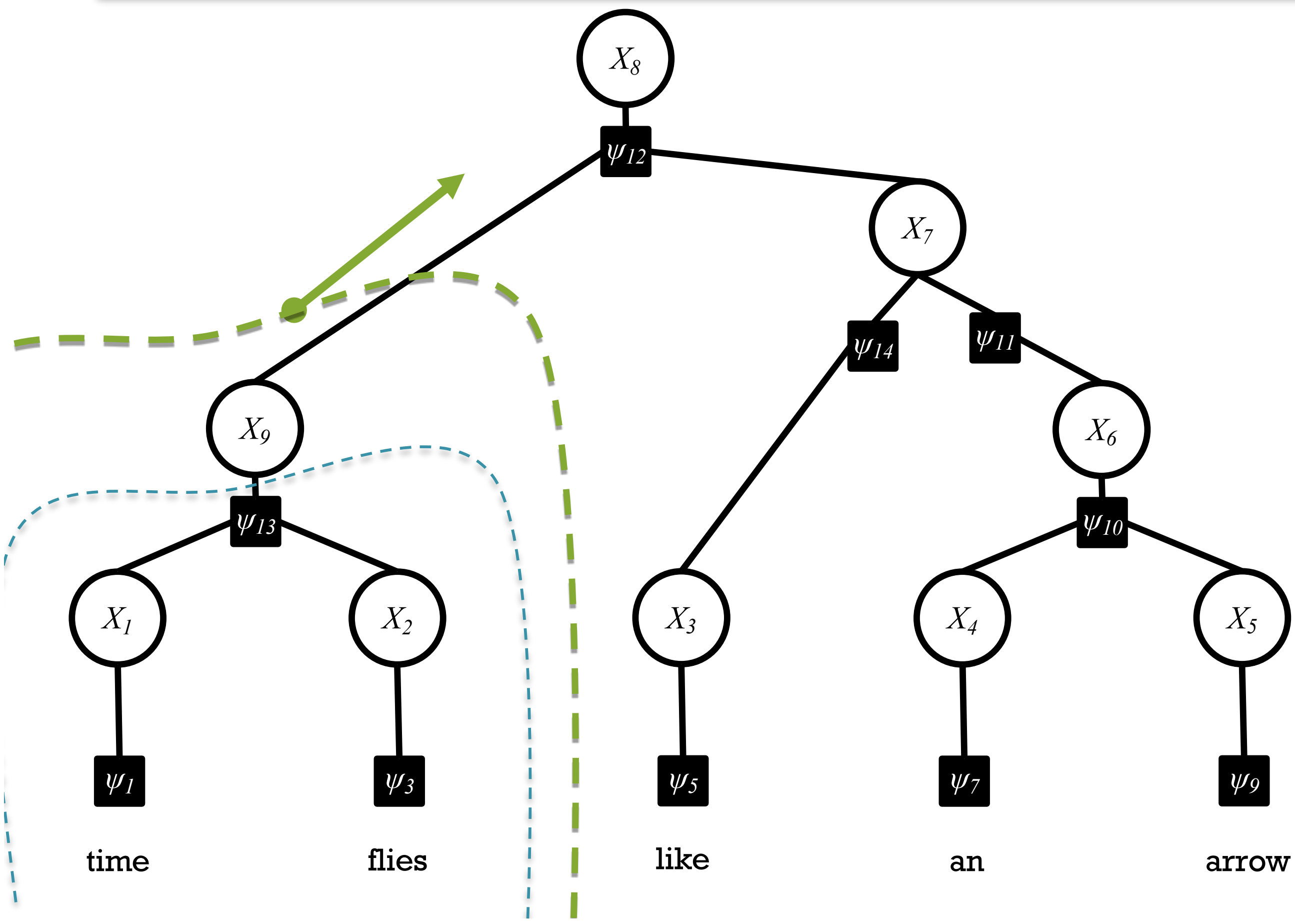
Acyclic BP as Dynamic Programming

- 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.



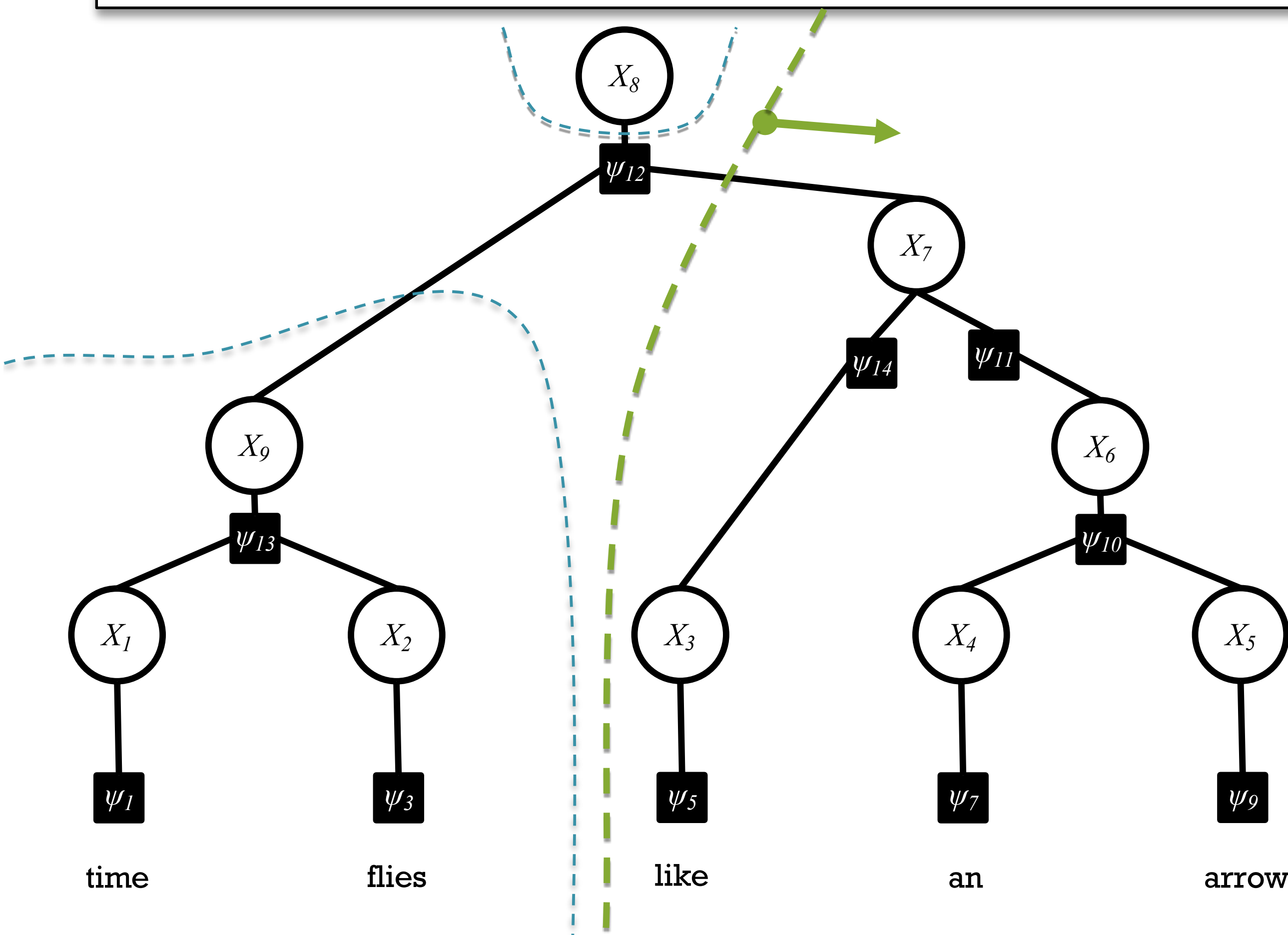
Acyclic BP as Dynamic Programming

- 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.



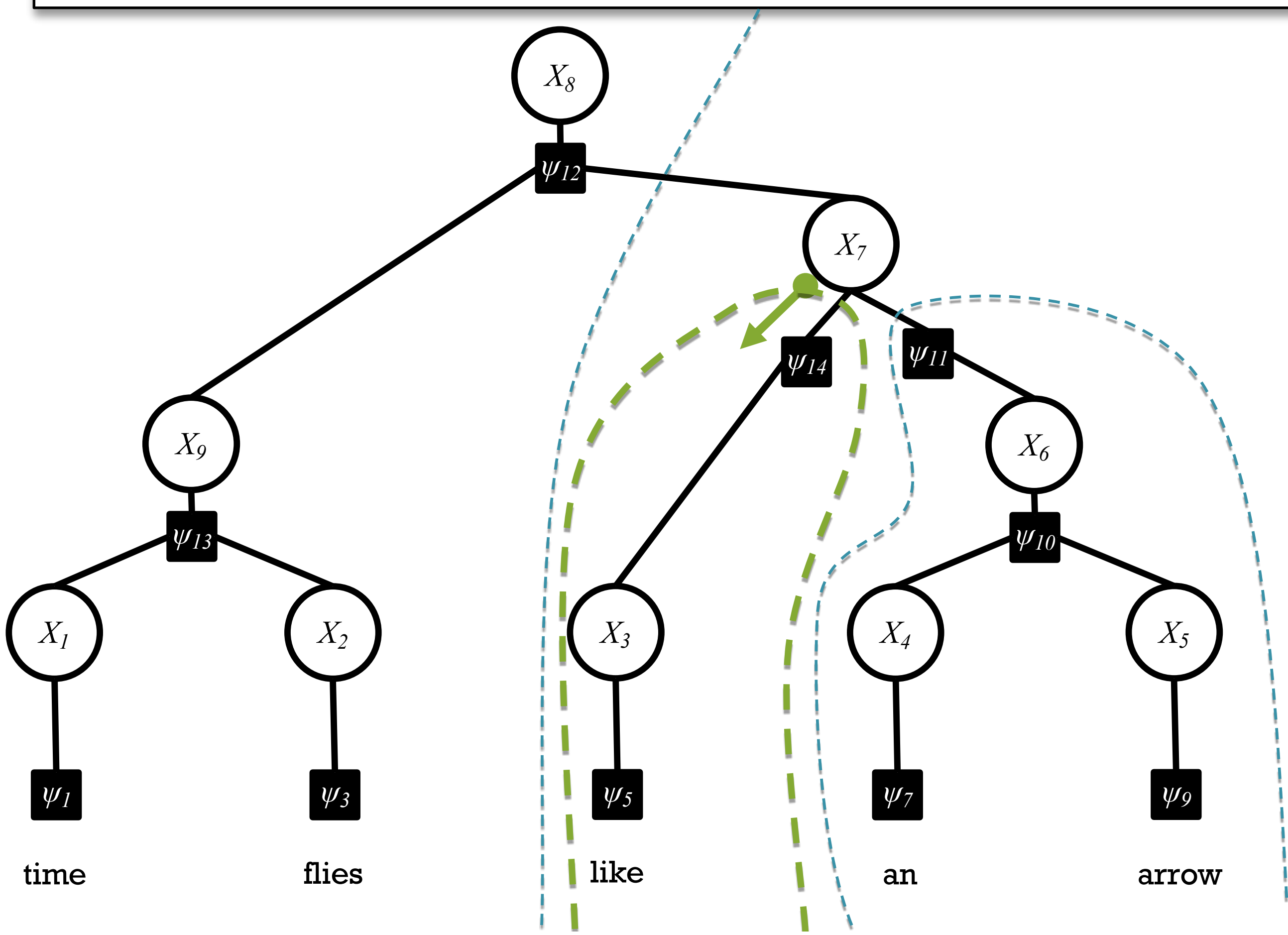
Acyclic BP as Dynamic Programming

- 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.



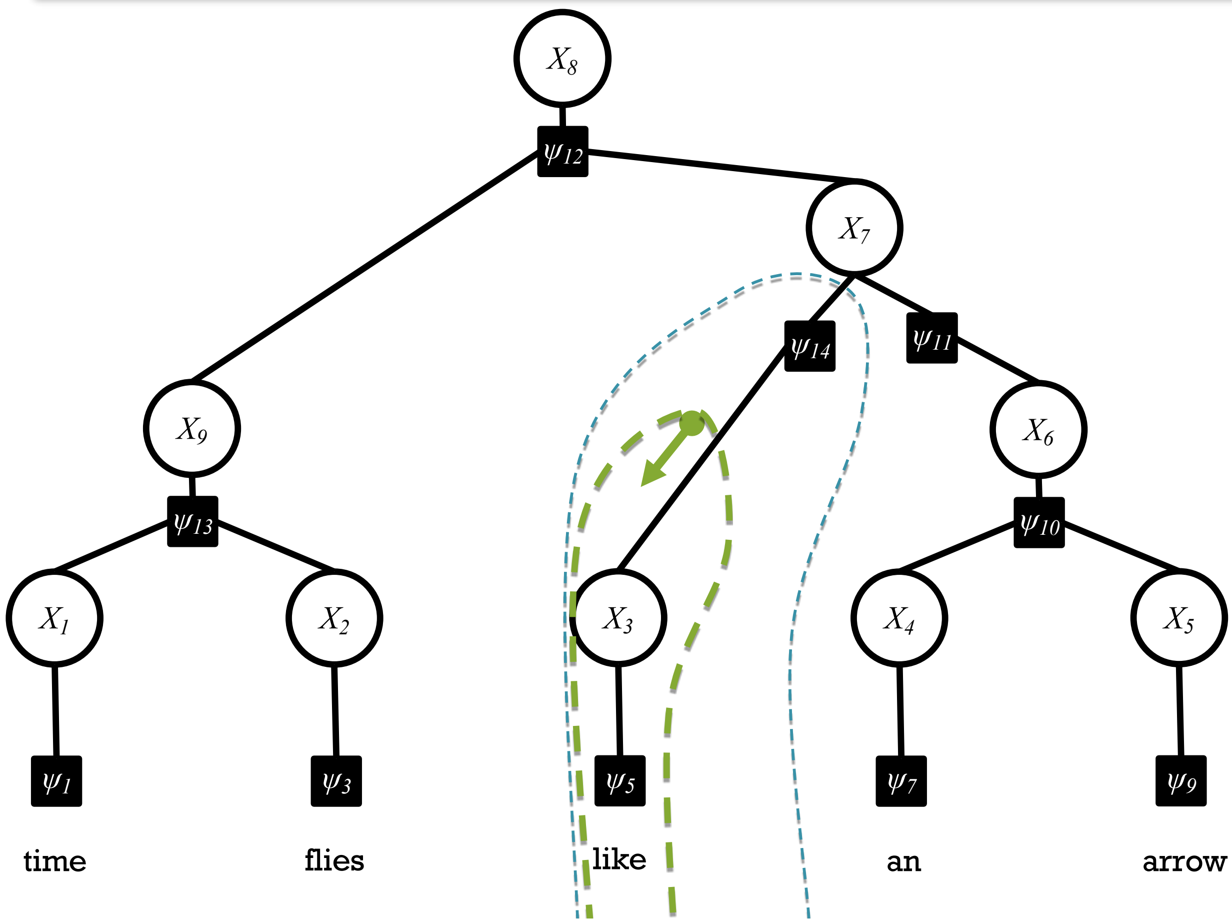
Acyclic BP as Dynamic Programming

- 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.



Exact MAP inference for factor trees

MAX-PRODUCT BELIEF PROPAGATION

Max-product Belief Propagation

- **Sum-product BP** can be used to
  compute the marginals, $p_i(X_i)$
  compute the partition function, Z

- **Max-product BP** can be used to
  compute the most likely assignment,
  $\boldsymbol{X}^* = \text{argmax}_{\boldsymbol{X}}\, p(\boldsymbol{X})$

Max-product Belief Propagation

- Change the sum to a max:

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

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

- **Max-product BP** computes **max-marginals**
  - The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.
  - For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

$$x_i^* = \arg\max_{x_i} b_i(x_i)$$

Max-product Belief Propagation

- Change the sum to a max:

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

$$\boxed{\mu_{\alpha \to i}(x_i) = \max_{\boldsymbol{x_\alpha} : \boldsymbol{x_\alpha}[i] = x_i} \psi_\alpha(\boldsymbol{x_\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x_\alpha}[i])}$$

- **Max-product BP** computes **max-marginals**
  - The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.
  - For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

$$x_i^* = \arg\max_{x_i} b_i(x_i)$$

Deterministic Annealing

Motivation: Smoothly transition from sum-product to max-product

1. Incorporate inverse temperature parameter into each factor:

Annealed Joint Distribution

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})^{\frac{1}{T}}$$

1. Send messages as usual for sum-product BP
2. Anneal T from 1 to 0:

$T = 1$	Sum-product
$T \rightarrow 0$	Max-product

3. Take resulting beliefs to power T

Semirings

- Sum-product +/* and max-product max/* are commutative semirings
- We can run BP with any such commutative semiring

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

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

- In practice, multiplying many small numbers together can yield underflow
  - instead of using +/*, we use log-add/+
  - Instead of using max/*, we use max/+

Exact inference for linear chain models

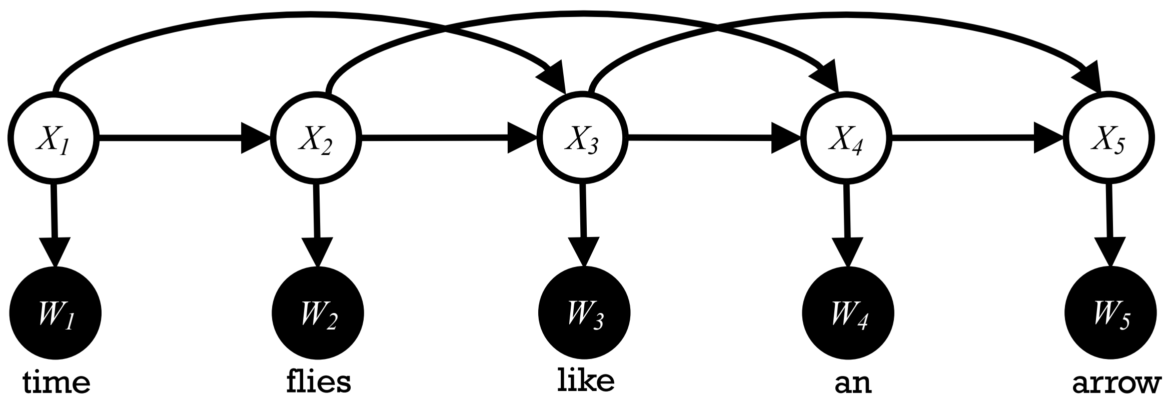
FORWARD-BACKWARD AND VITERBI ALGORITHMS

Forward-Backward Algorithm

- Sum-product BP on an HMM is called the **forward-backward algorithm**
- Max-product BP on an HMM is called the **Viterbi algorithm**

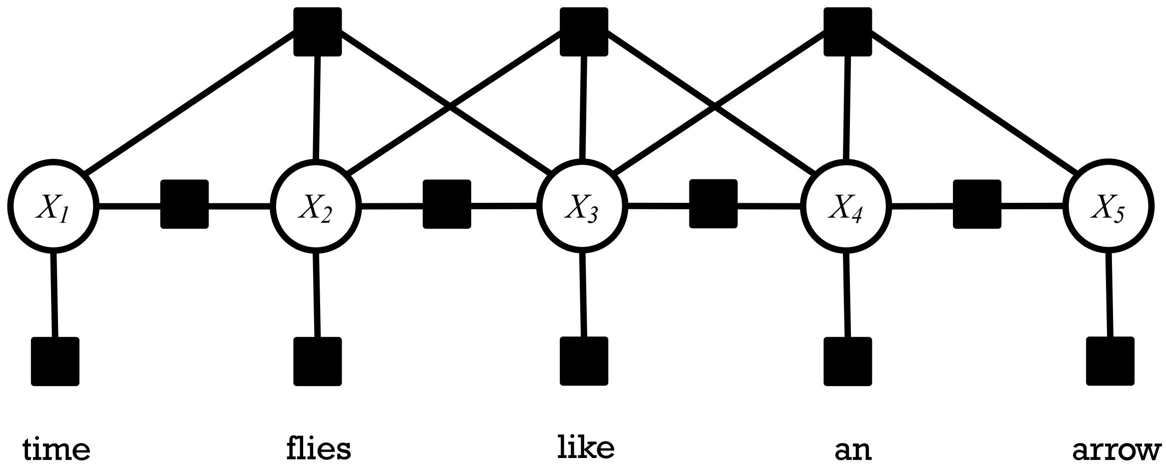
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



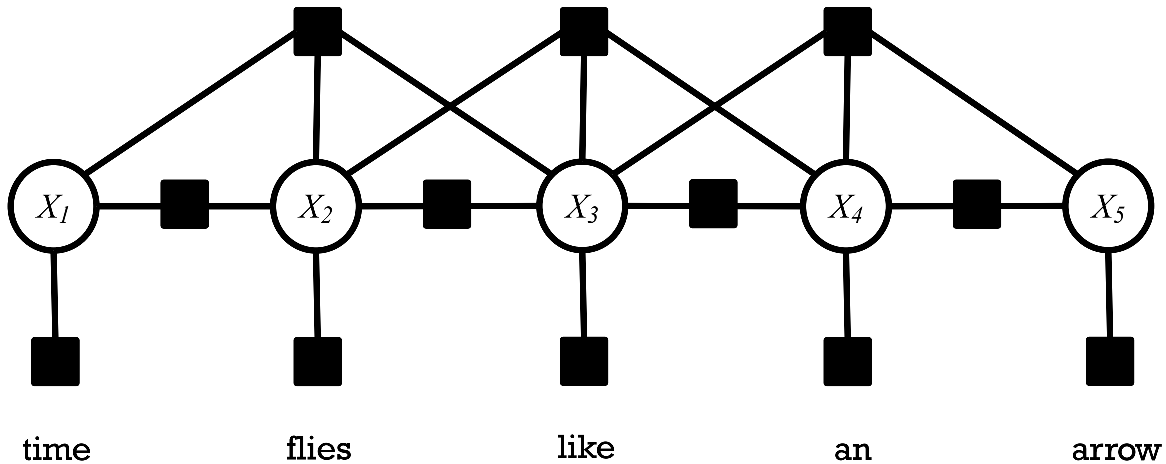
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



Trick: (See also Sha & Pereira (2003))

- Replace each variable domain with its cross product
  e.g. {B,I,O} → {BB, BI, BO, IB, II, IO, OB, OI, OO}
- Replace each pair of variables with a single one. For all i, $y_{i,i+1} = (x_i, x_{i+1})$
- Add features with weight -∞ that disallow illegal configurations between pairs of the new variables
  e.g. **legal** = BI and IO **illegal** = II and OO
- This is effectively a special case of the junction tree algorithm

Summary

1. **Factor Graphs**
   – Alternative representation of directed / undirected graphical models
   – Make the cliques of an undirected GM explicit
2. **Variable Elimination**
   – Simple and general approach to exact inference
   – Just a matter of being clever when computing sum-products
3. **Sum-product Belief Propagation**
   – Computes all the marginals and the partition function in only twice the work of Variable Elimination
4. **Max-product Belief Propagation**
   – Identical to sum-product BP, but changes the semiring
   – Computes: max-marginals, probability of MAP assignment, and (with backpointers) the MAP assignment itself.