Machine Learning Department
School of Computer Science
Carnegie Mellon University

Belief Propagation
+
Learning fully observable MRFs and CRFs

Matt Gormley
Lecture 9
Sep. 28, 2022

Reminders

- **Homework 2: Learning to Search for RNNs**
  - Out: Sun, Sep 18
  - Written (except for Empirical Questions)
    - Due: Thu, Sep 29 at 11:59pm
  - Programming + Empirical Questions
    - Due: NEVER?
- **Homework 3: General Graph CRF Module**
  - Out: Thu, Sep 29
  - Due: Mon, Oct 10 at 11:59pm

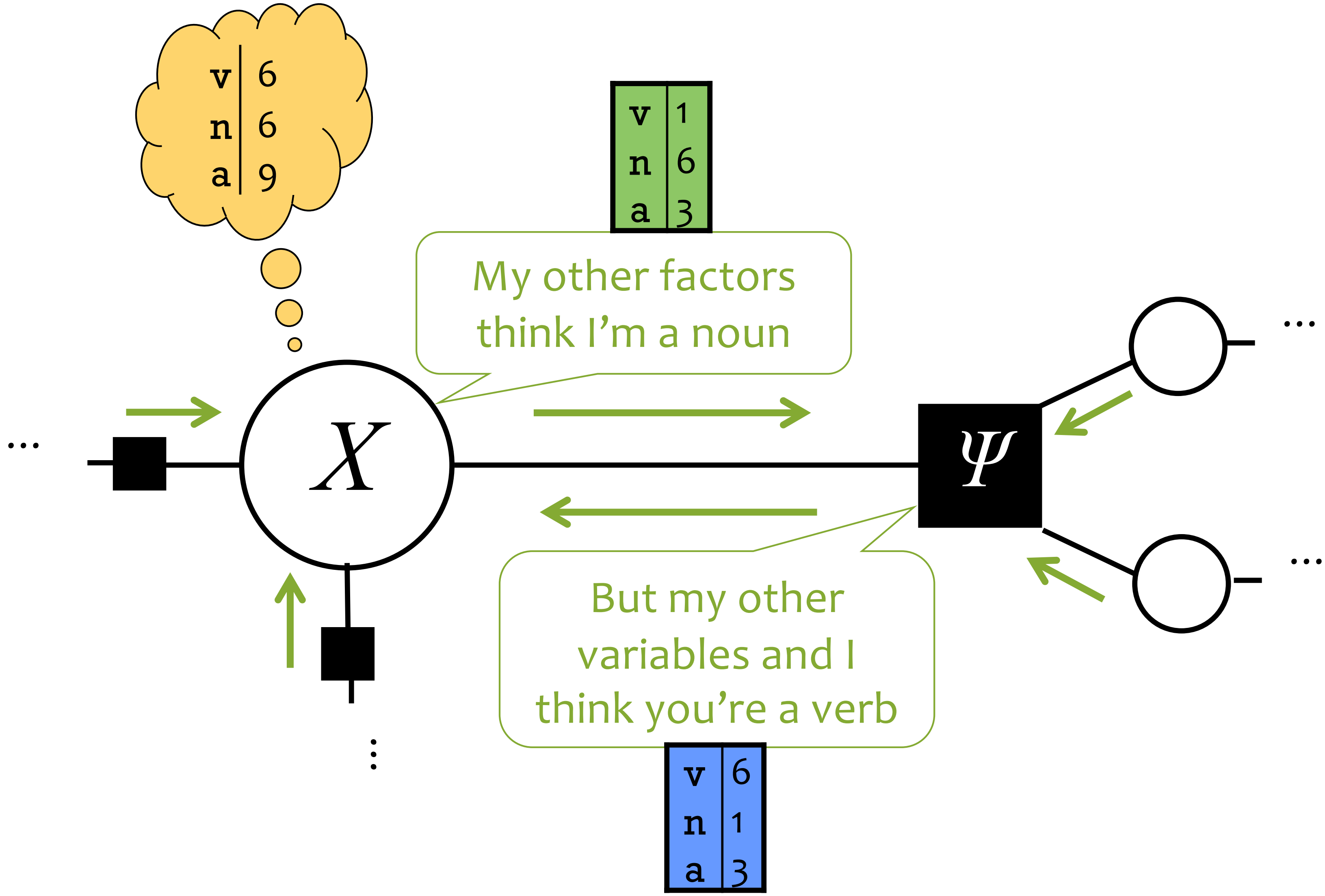
Reminders

- Homework 2: Learning to Search for RNNs
  - Out: Sun, Sep 18
  - Written (except for Empirical Questions)
    - Due: Thu, Sep 29 at 11:59pm
  - Programming + Empirical Questions
    - Due: Mon, Oct 24 at 9:00am
- Homework 3: General Graph CRF Module
  - Out: Thu, Sep 29
  - Due: Mon, Oct 10 at 11:59pm

Exact marginal inference for factor trees

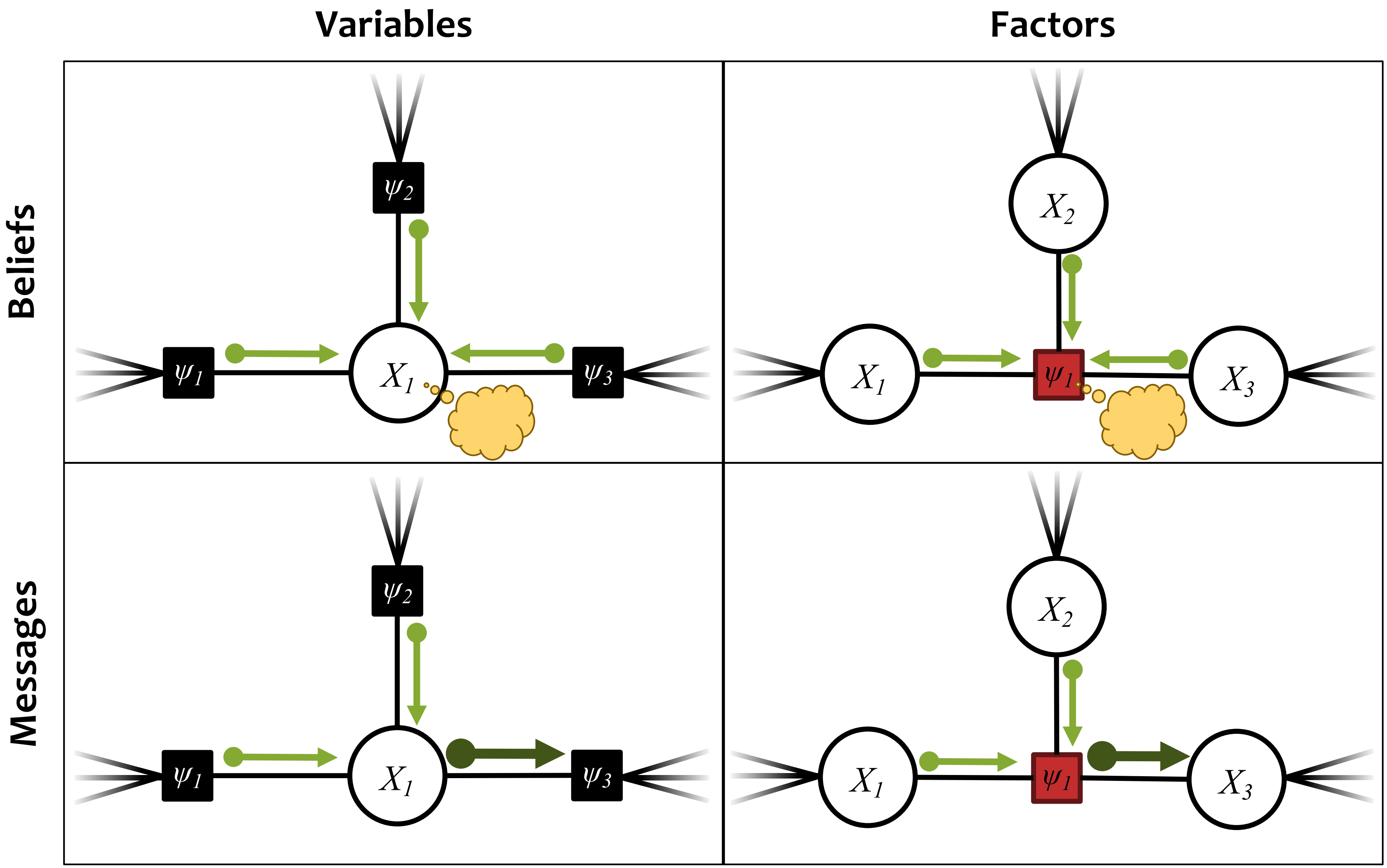
SUM-PRODUCT BELIEF PROPAGATION

Message Passing in Belief Propagation



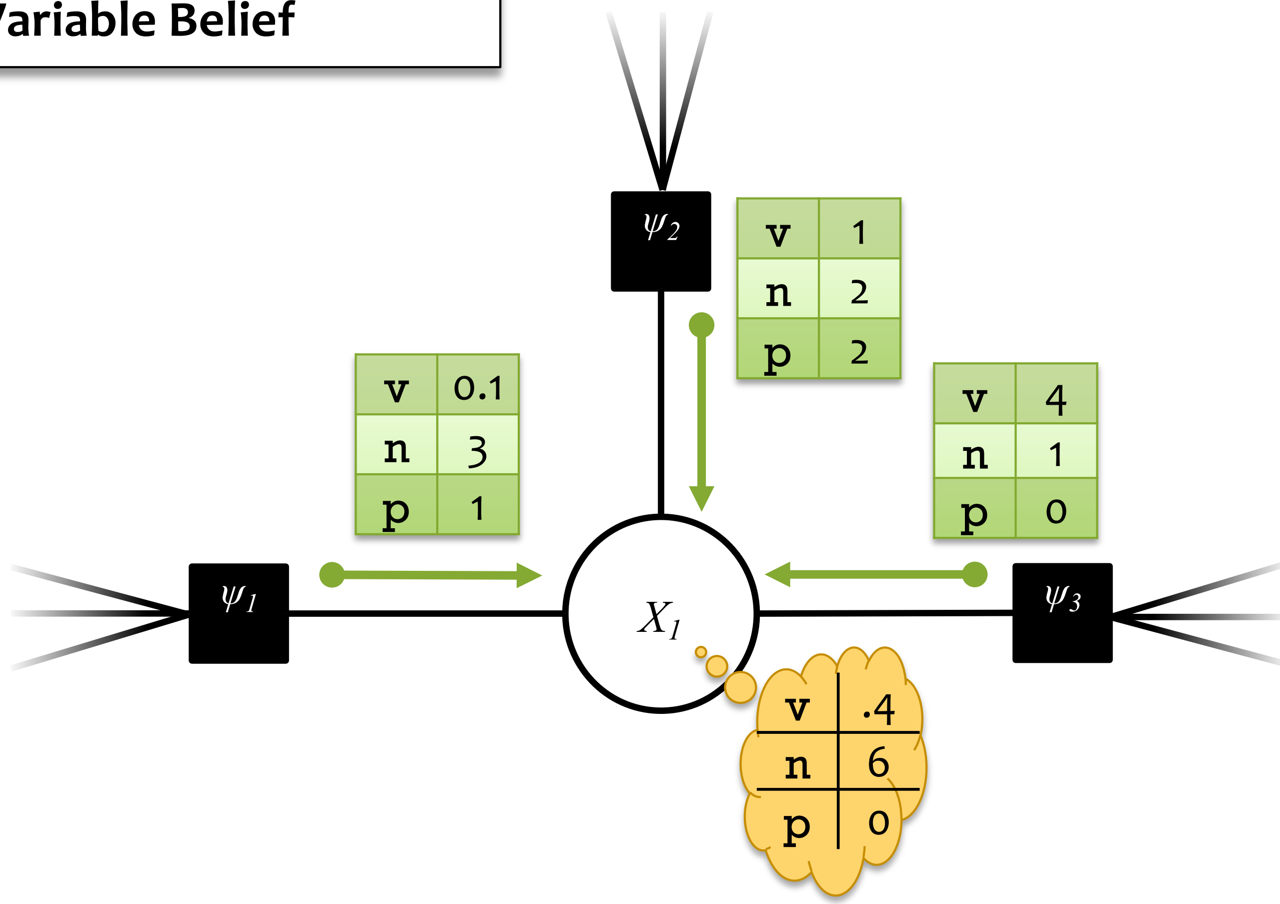
Both of these messages judge the possible values of variable X.
Their product = belief at X = product of all 3 messages to X.

Sum-Product Belief Propagation



Sum-Product Belief Propagation

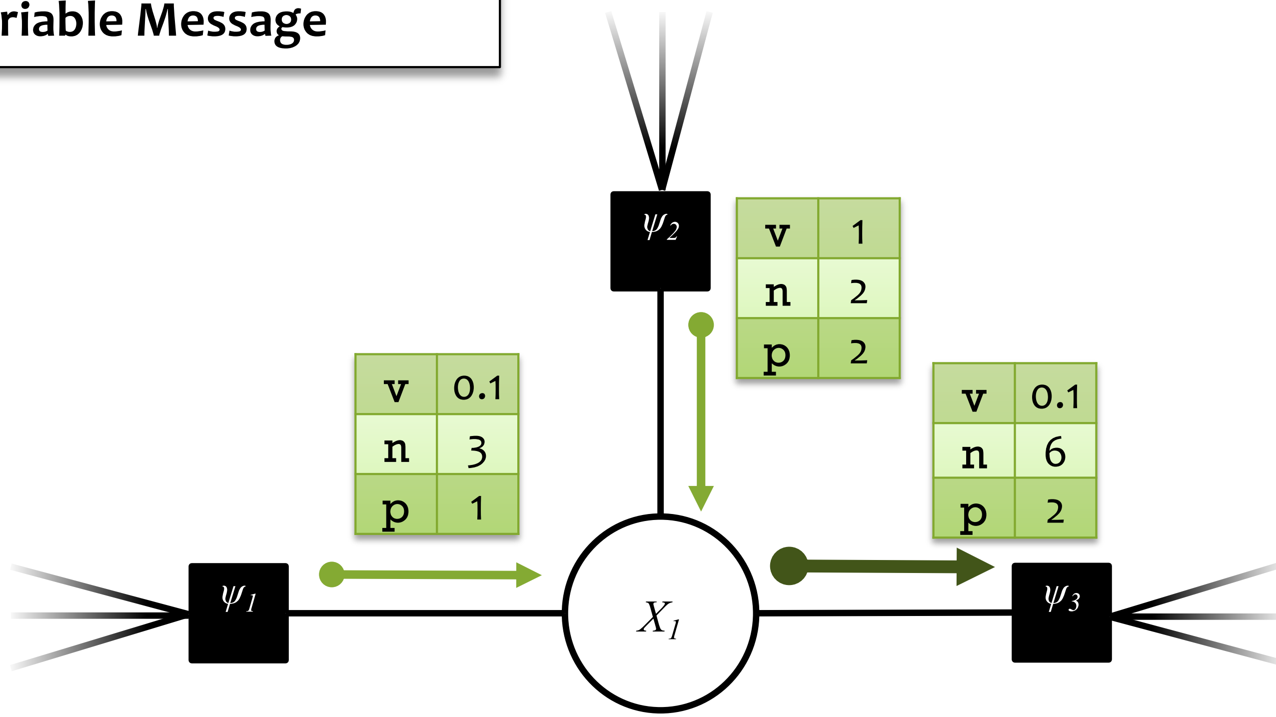
Variable Belief



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

Sum-Product Belief Propagation

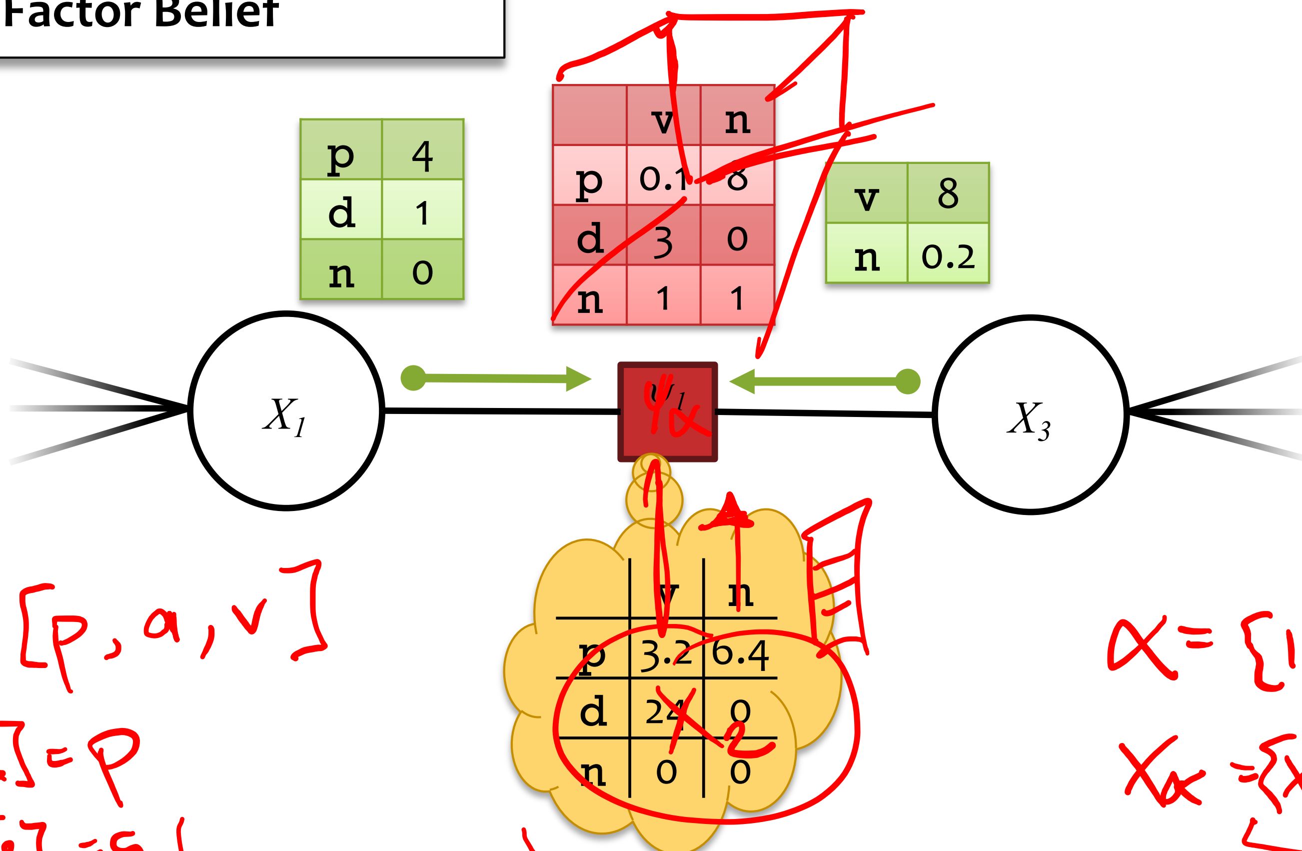
Variable Message



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

Sum-Product Belief Propagation

Factor Belief



$X_\alpha = [p, a, v]$

$X_\alpha[1] = p$

$X_\alpha[2] = a$

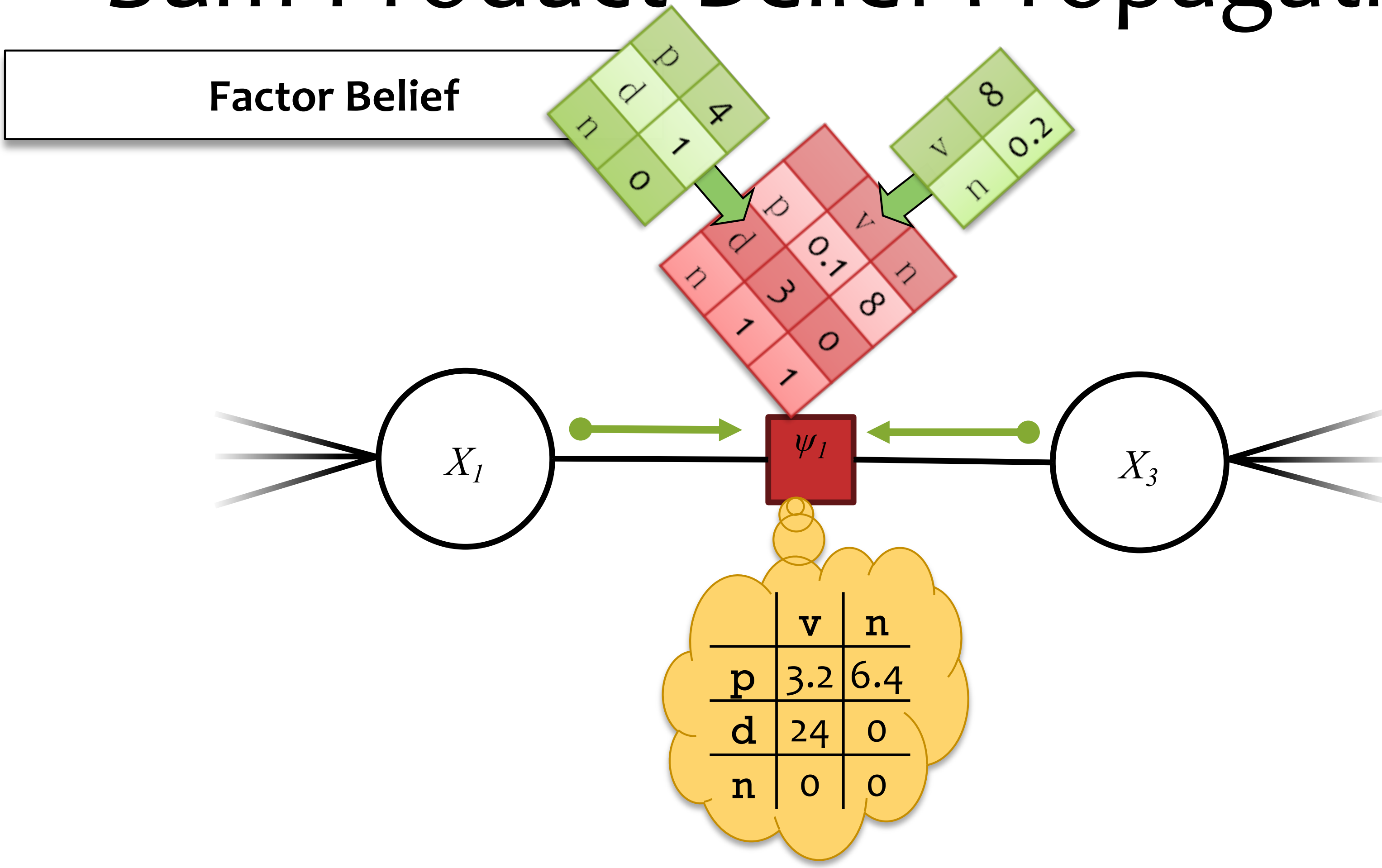
$X_\alpha[3] = v$

$\alpha = \{1, 2, 3\}$

$X_\alpha = \{X_1, X_2, X_3\}$

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

Sum-Product Belief Propagation



	v	n
p	3.2	6.4
d	24	0
n	0	0

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

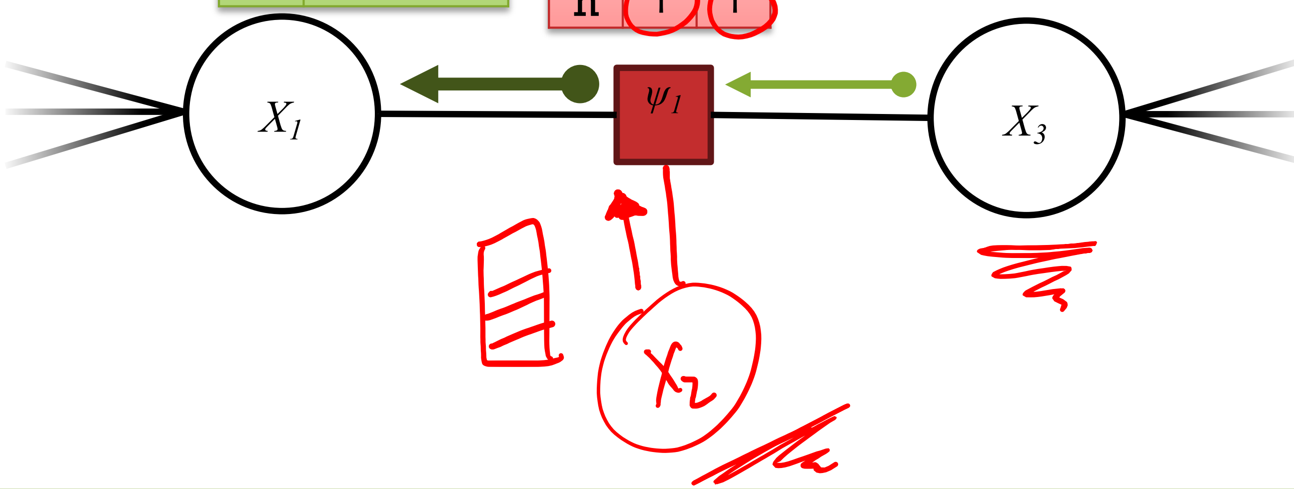
Sum-Product Belief Propagation

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

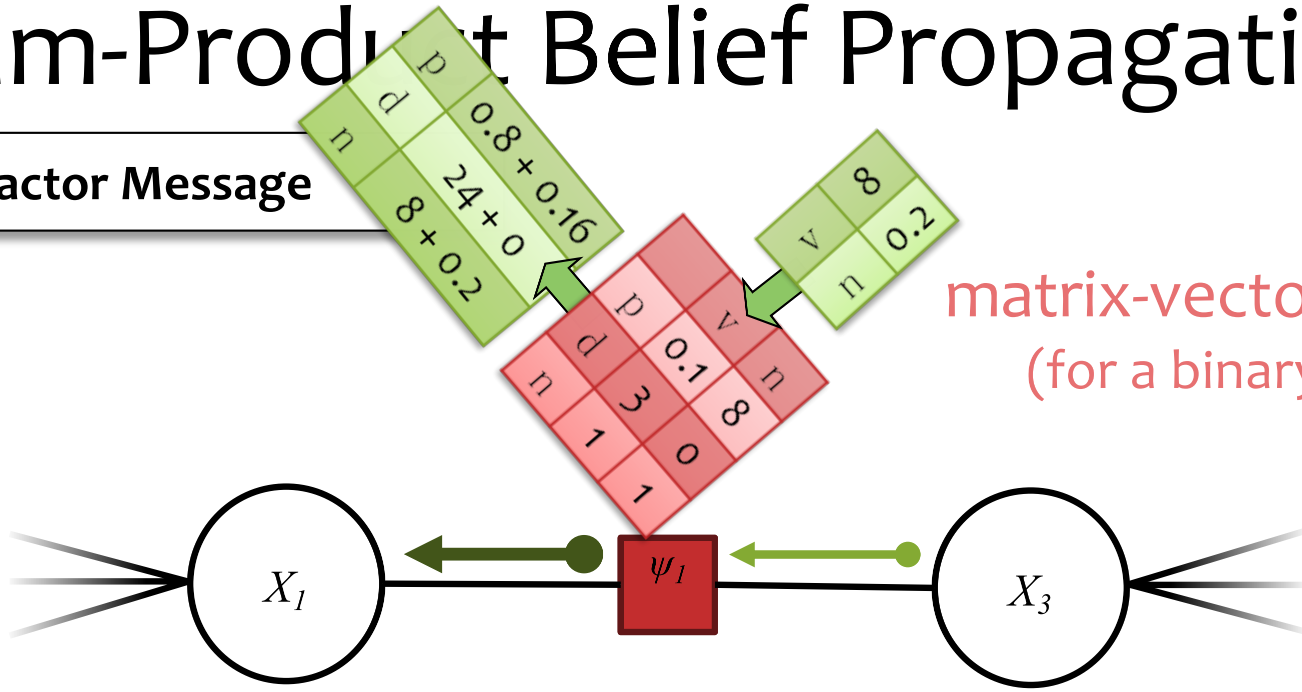
v	8
n	0.2



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

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**.

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

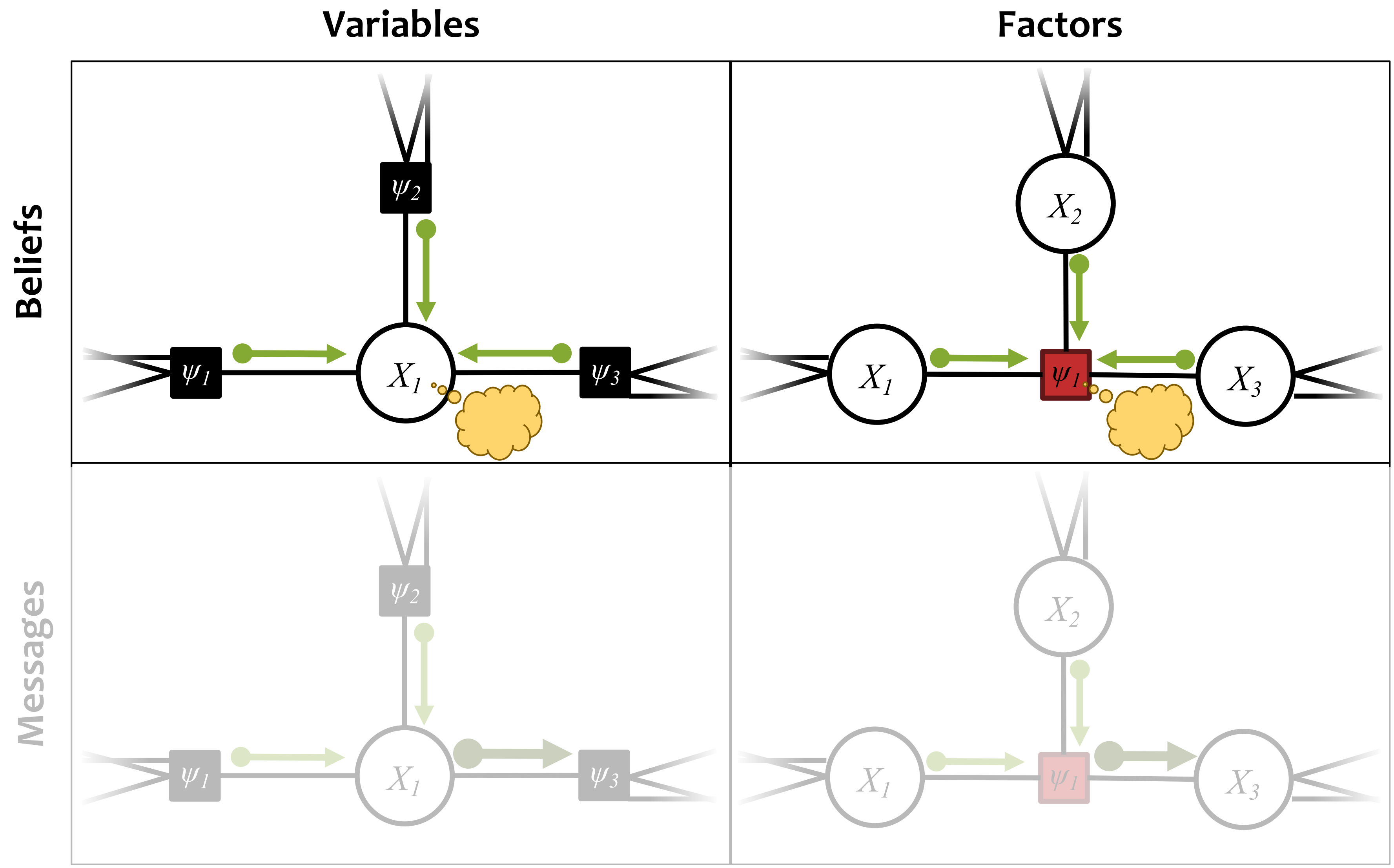
1. Compute the beliefs (unnormalized marginals).

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

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

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

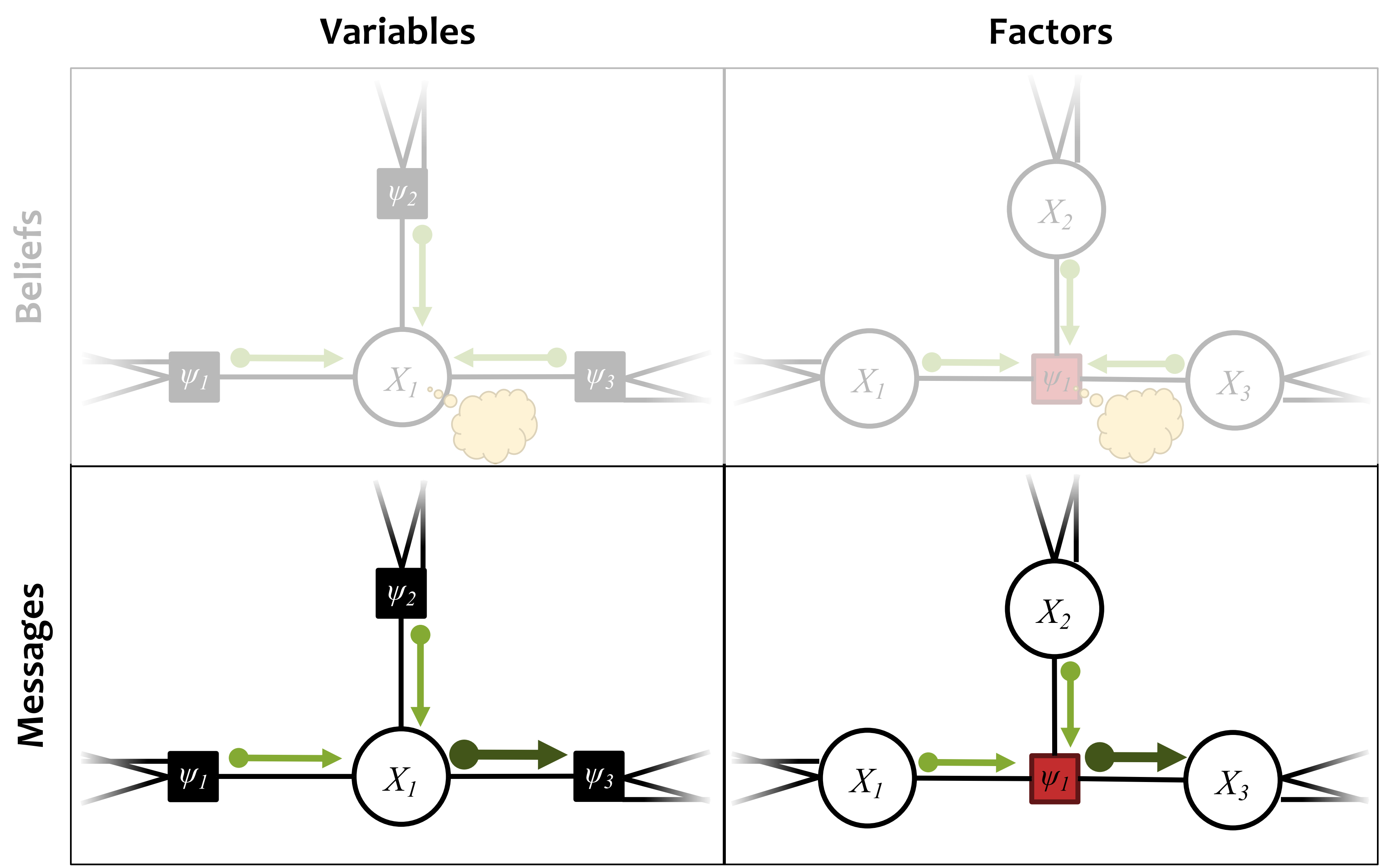
Sum-Product Belief Propagation



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

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

Sum-Product Belief Propagation

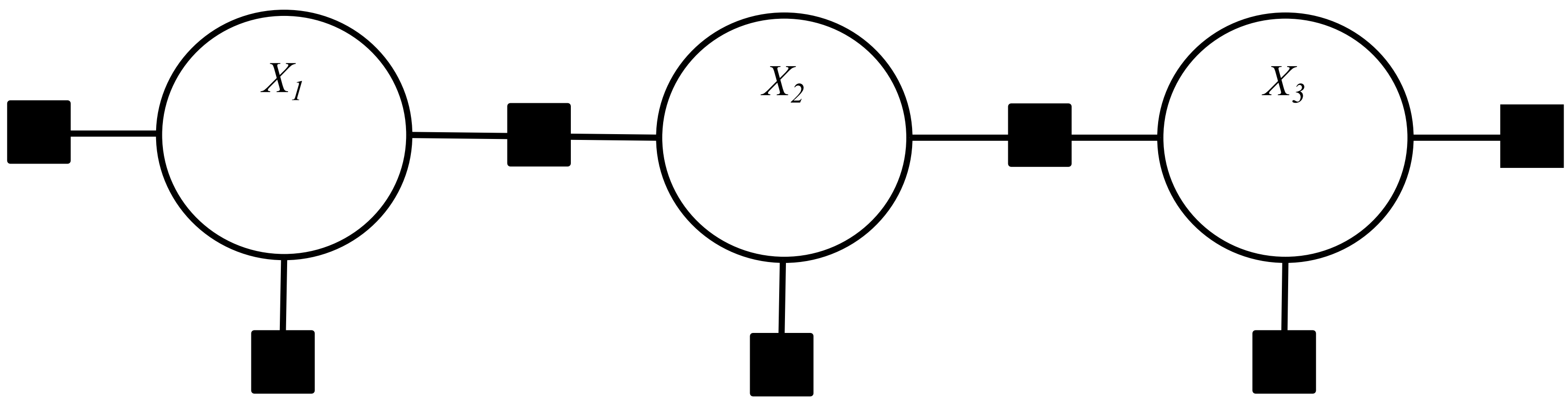


$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

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

FORWARD BACKWARD AS SUM-PRODUCT BP

CRF Tagging Model



find

preferred

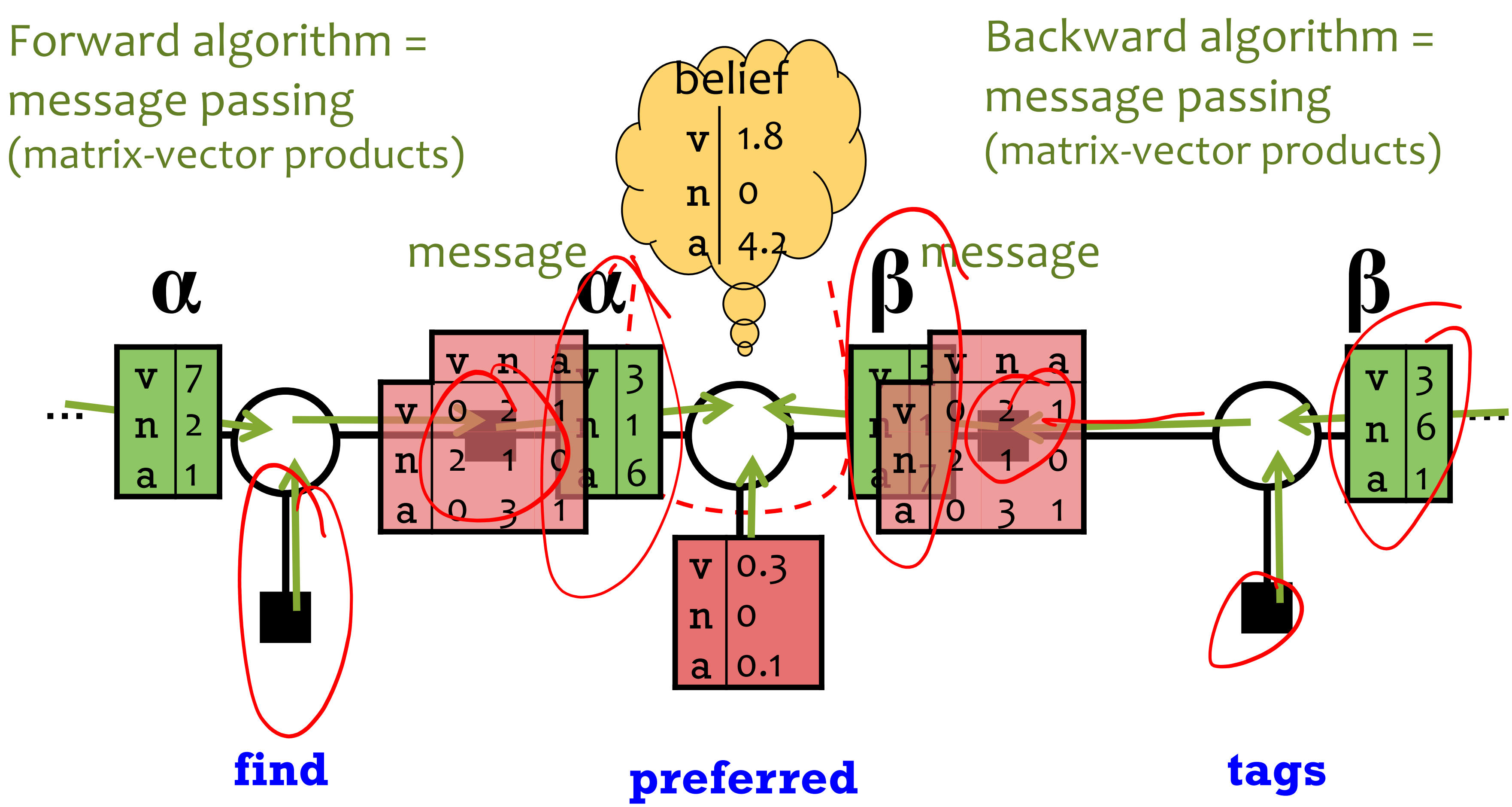
tags

Could be verb or noun

Could be adjective or verb

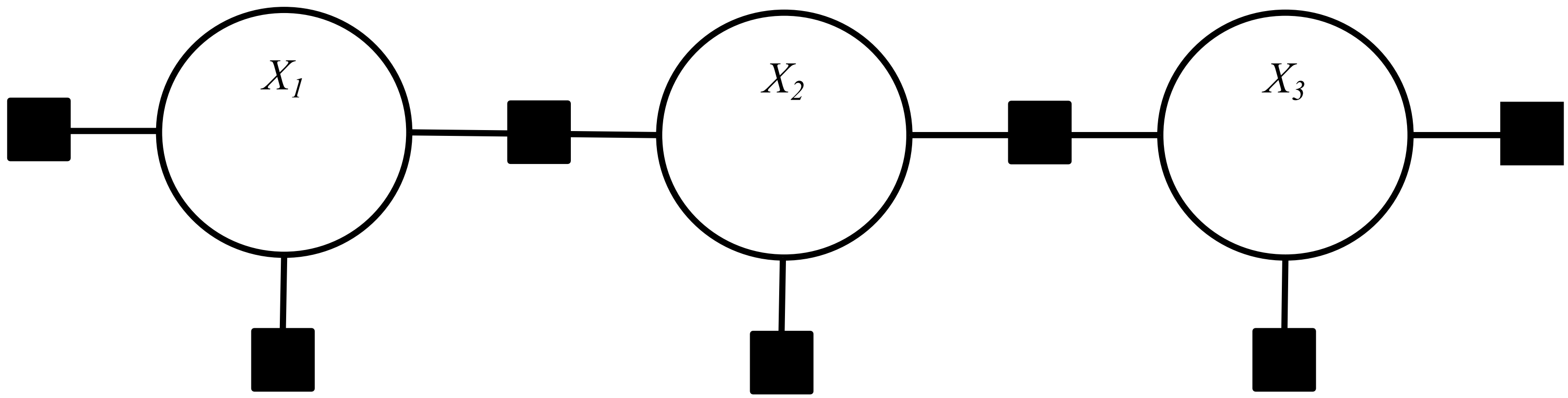
Could be noun or verb

CRF Tagging by Belief Propagation



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

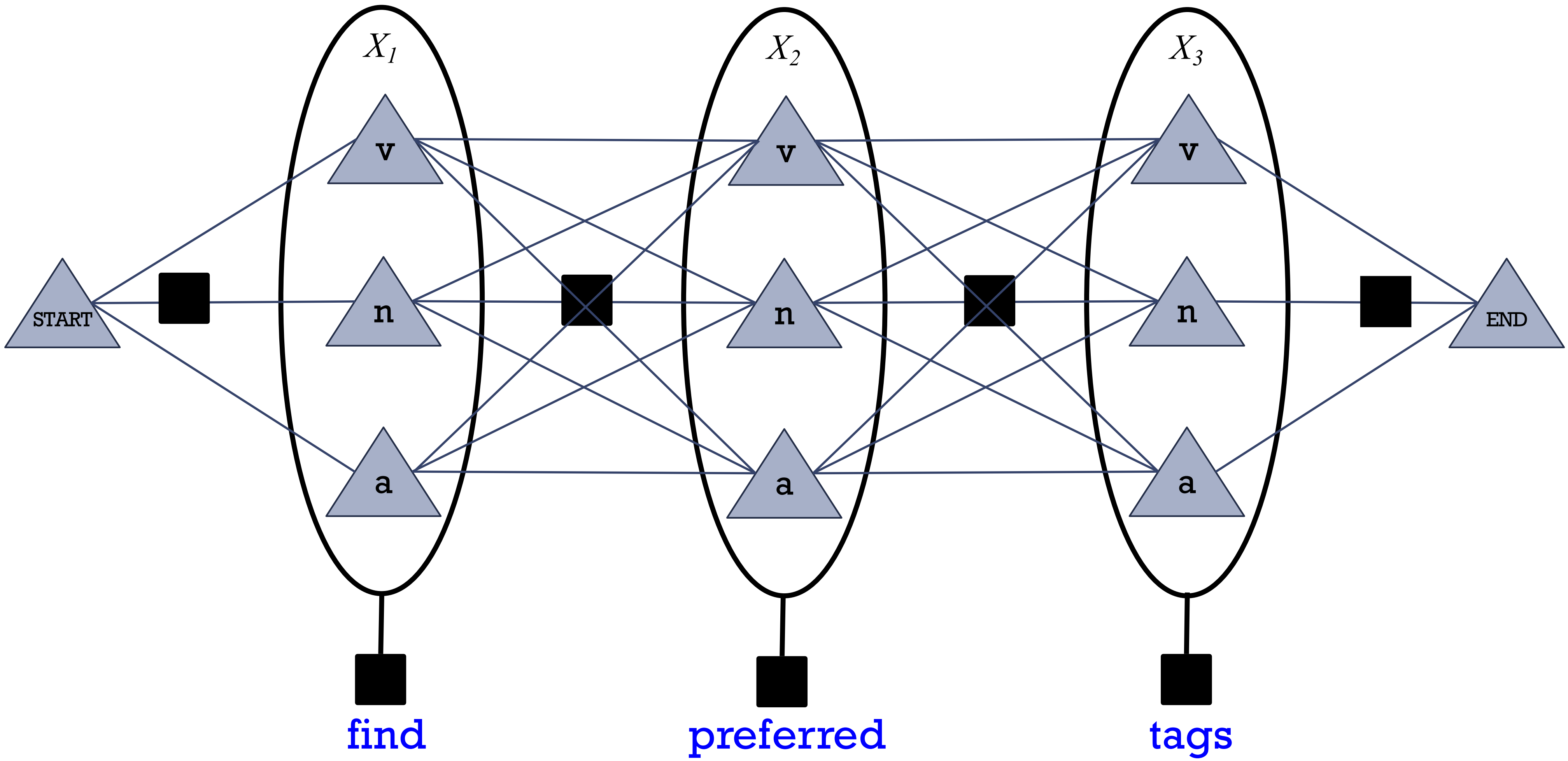
So Let's Review Forward-Backward …



find — preferred — tags

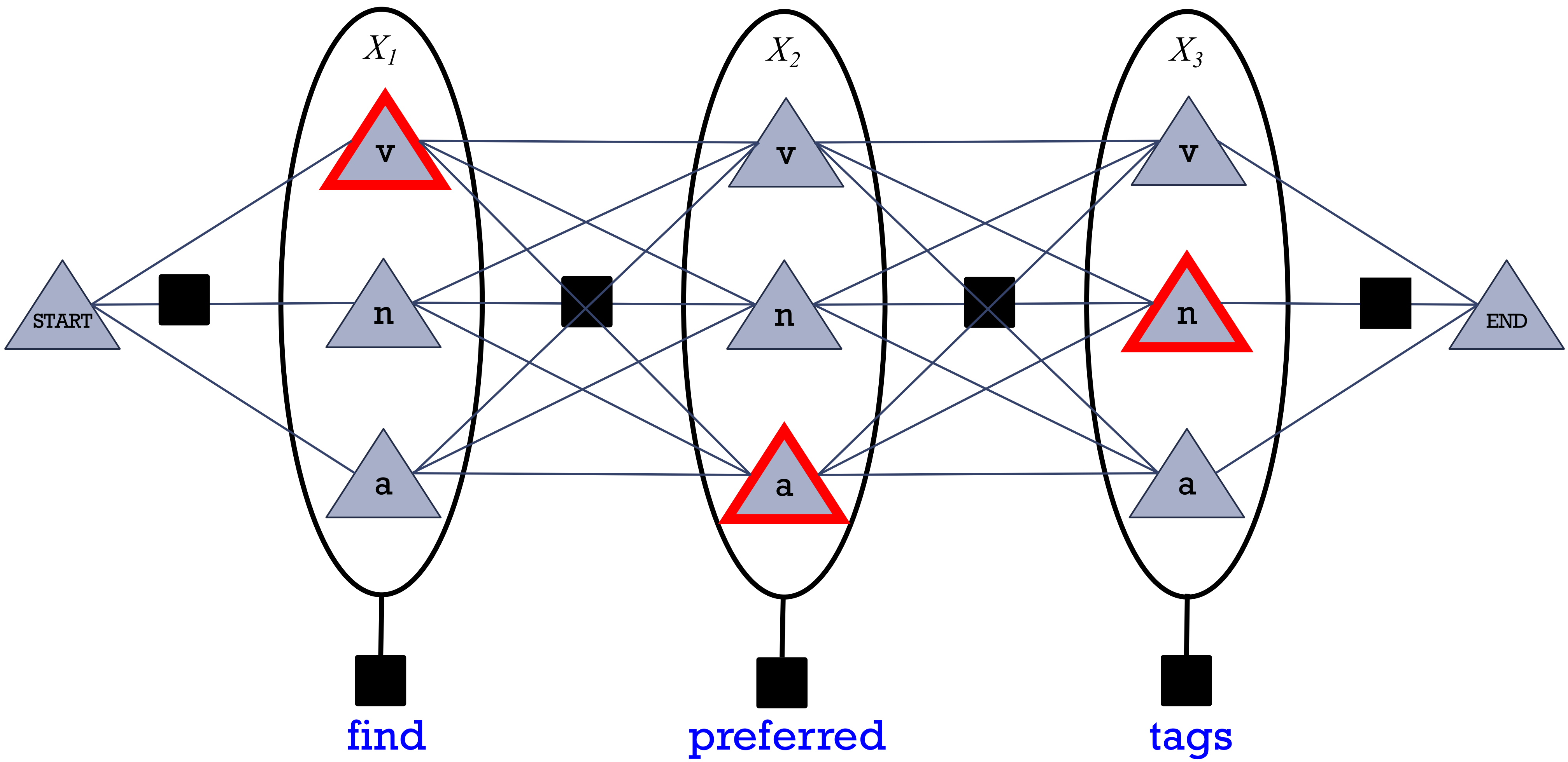
Could be verb or noun — *Could be adjective or verb* — *Could be noun or verb*

So Let's Review Forward-Backward …



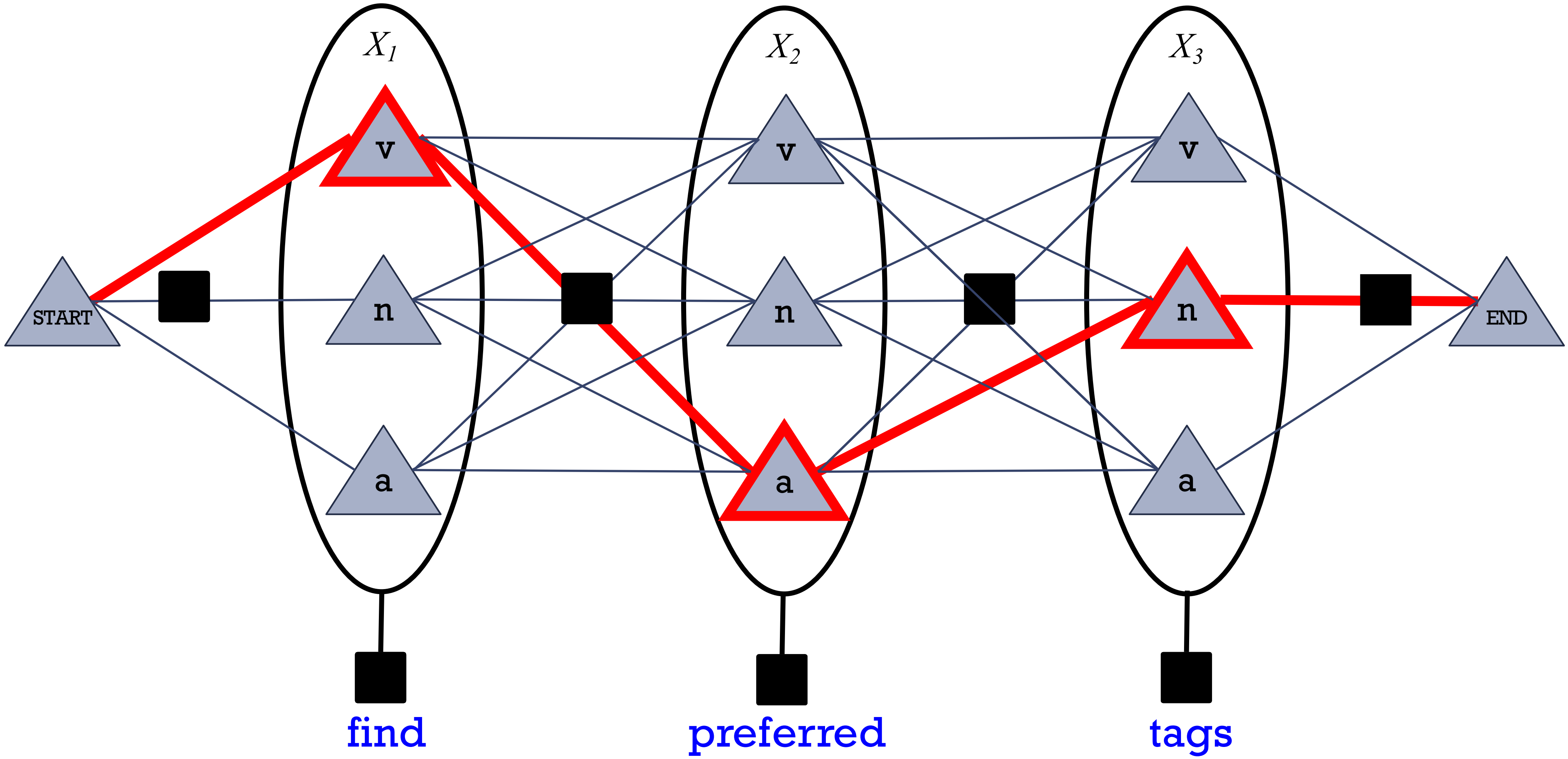
- Show the possible *values* for each variable

So Let's Review Forward-Backward …



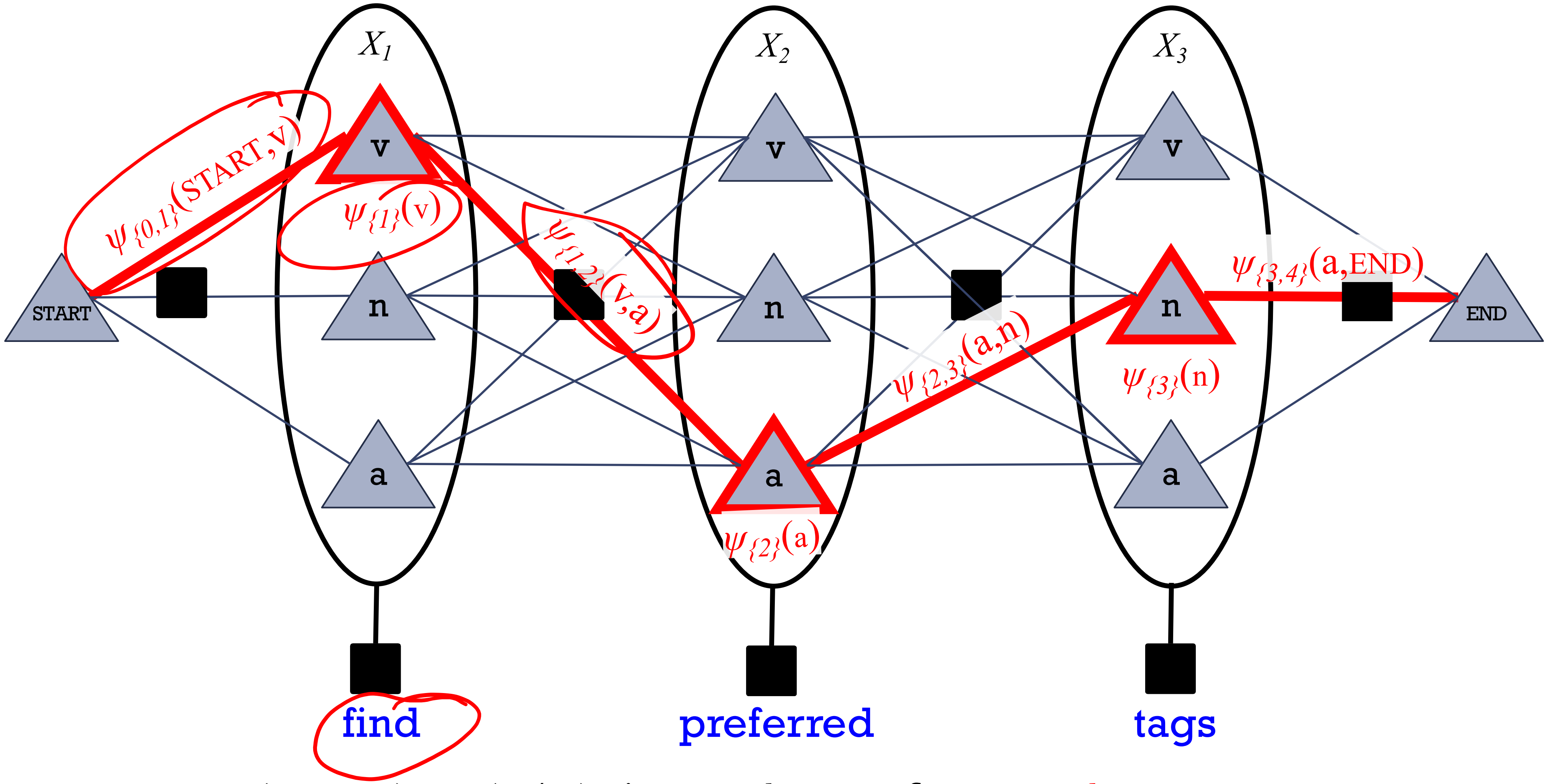
- Let's show the possible *values* for each variable
- One possible assignment

So Let's Review Forward-Backward …



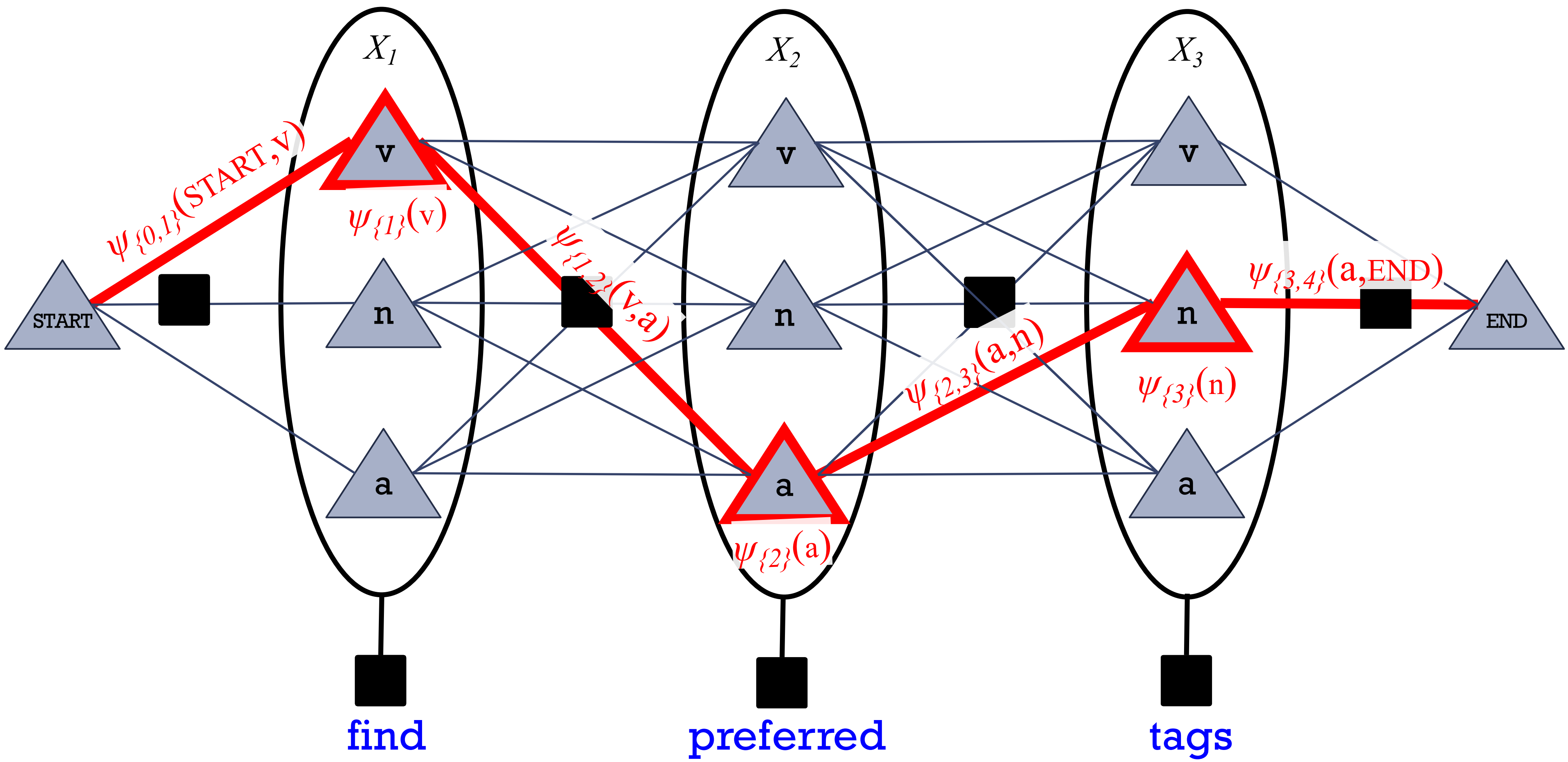
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



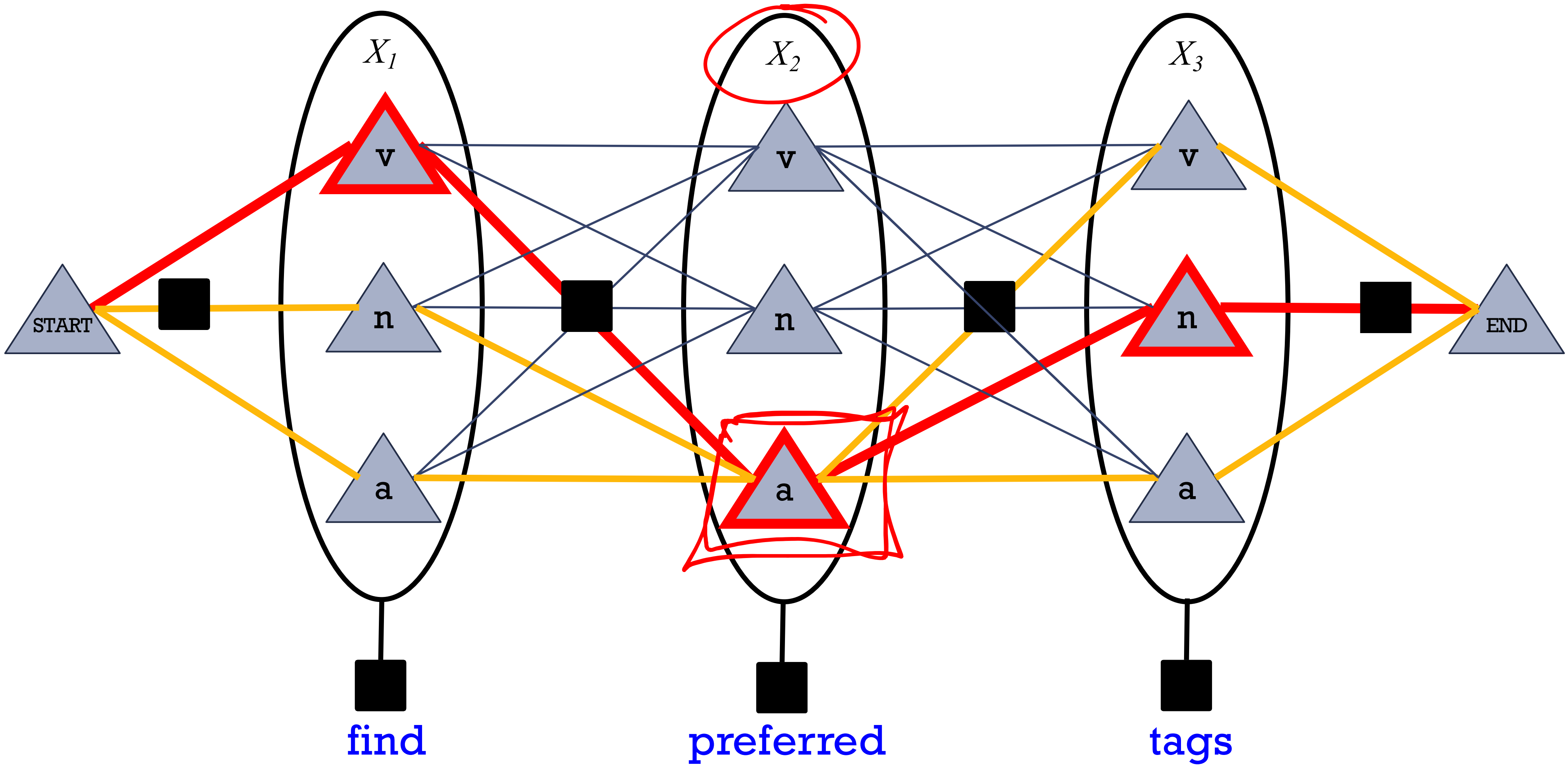
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment



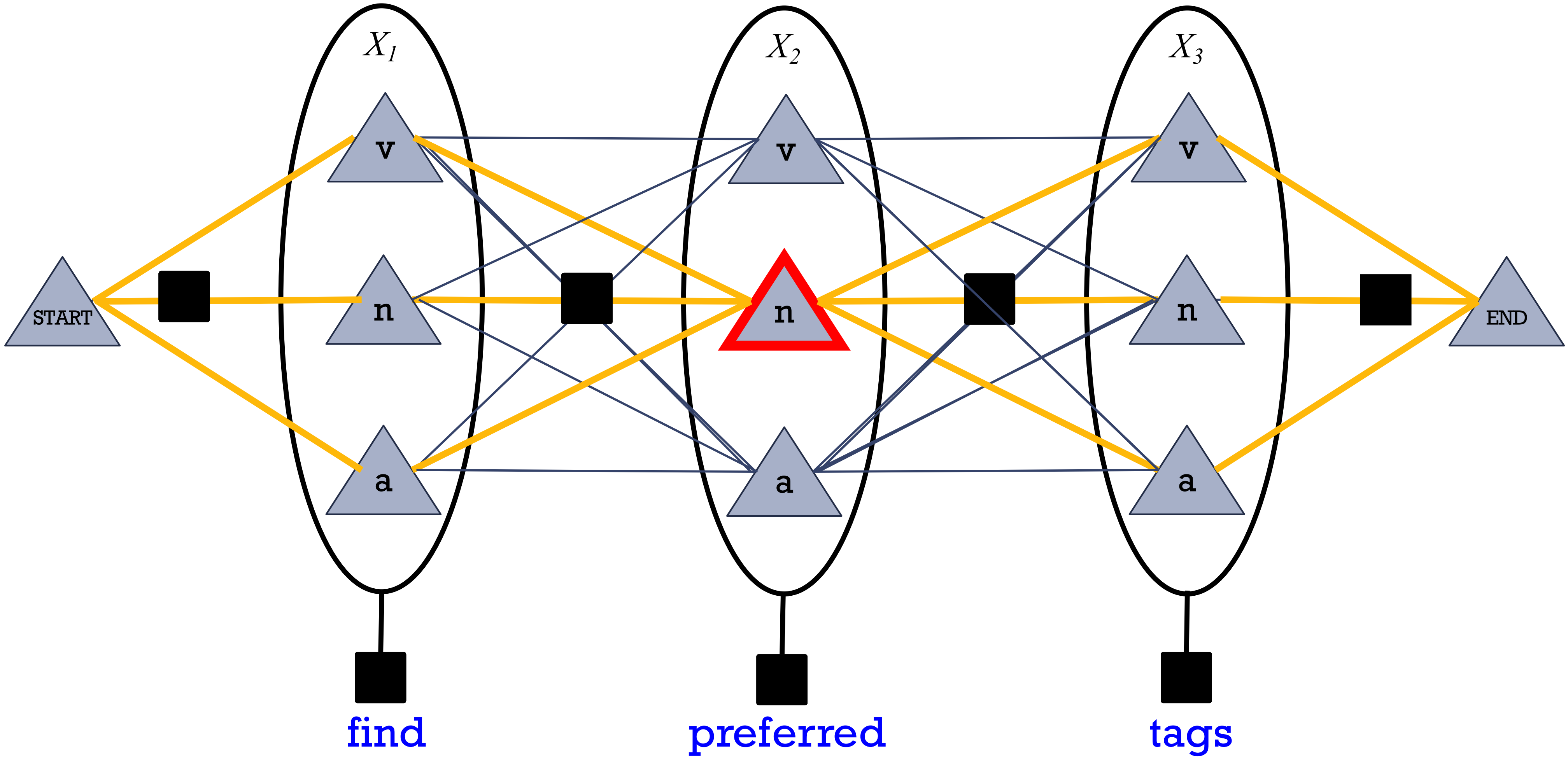
- So p(**v a n**) = (1/Z) * product weight of one path

Forward-Backward Algorithm: Finds Marginals



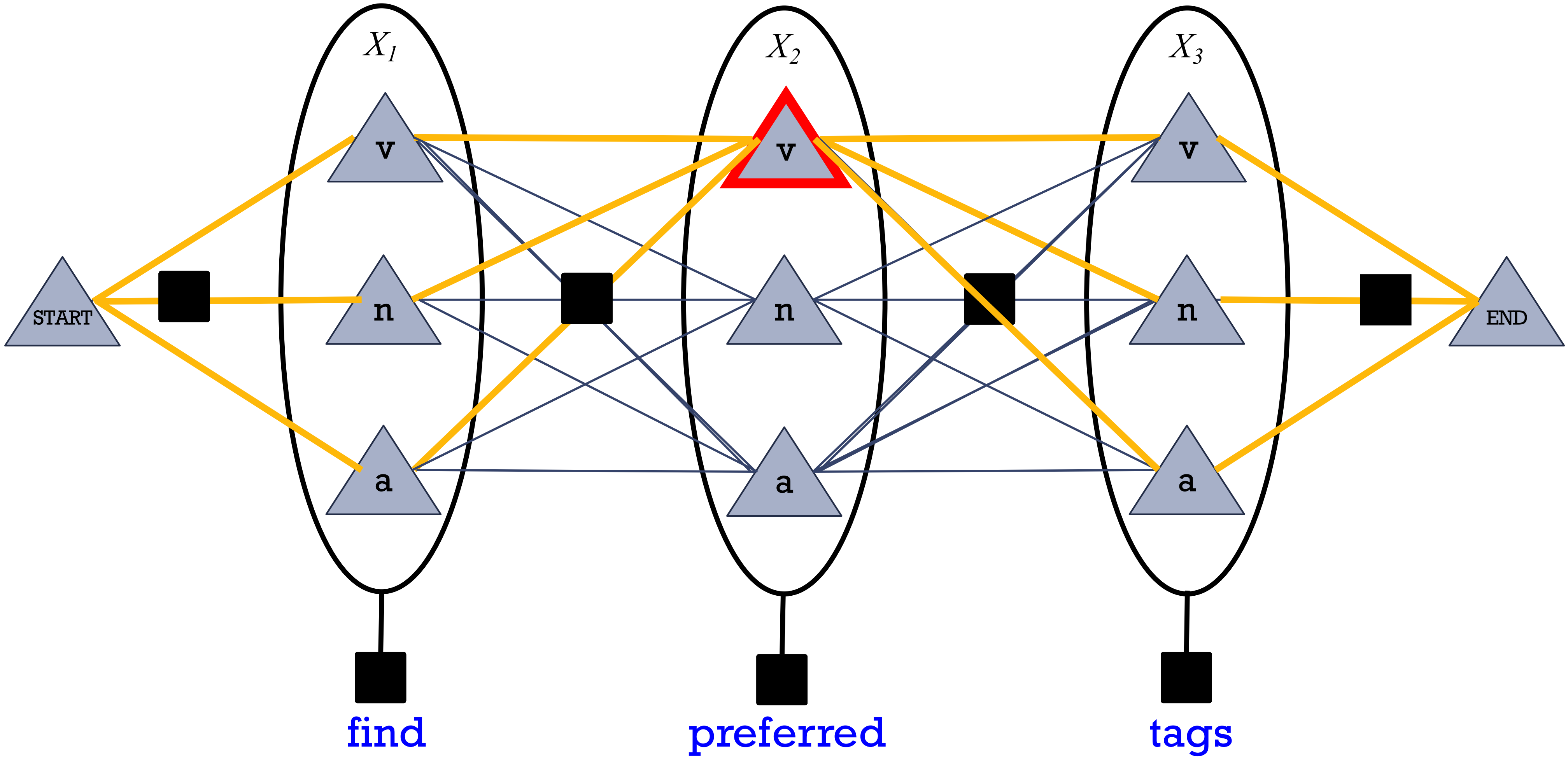
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through **a**

Forward-Backward Algorithm: Finds Marginals



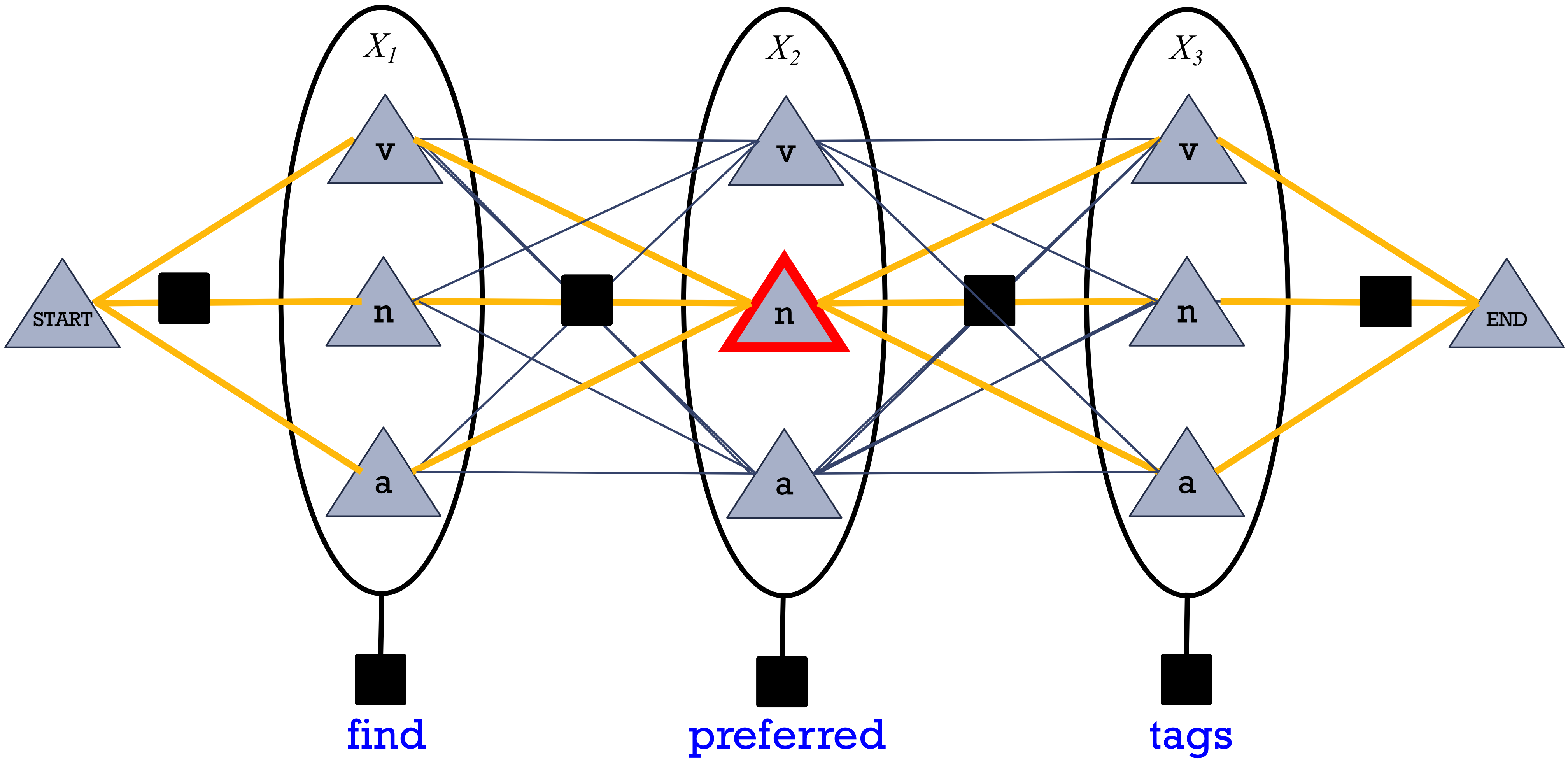
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = a)$
  = (1/Z) * total weight of *all* paths through **n**

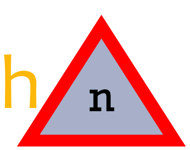
Forward-Backward Algorithm: Finds Marginals



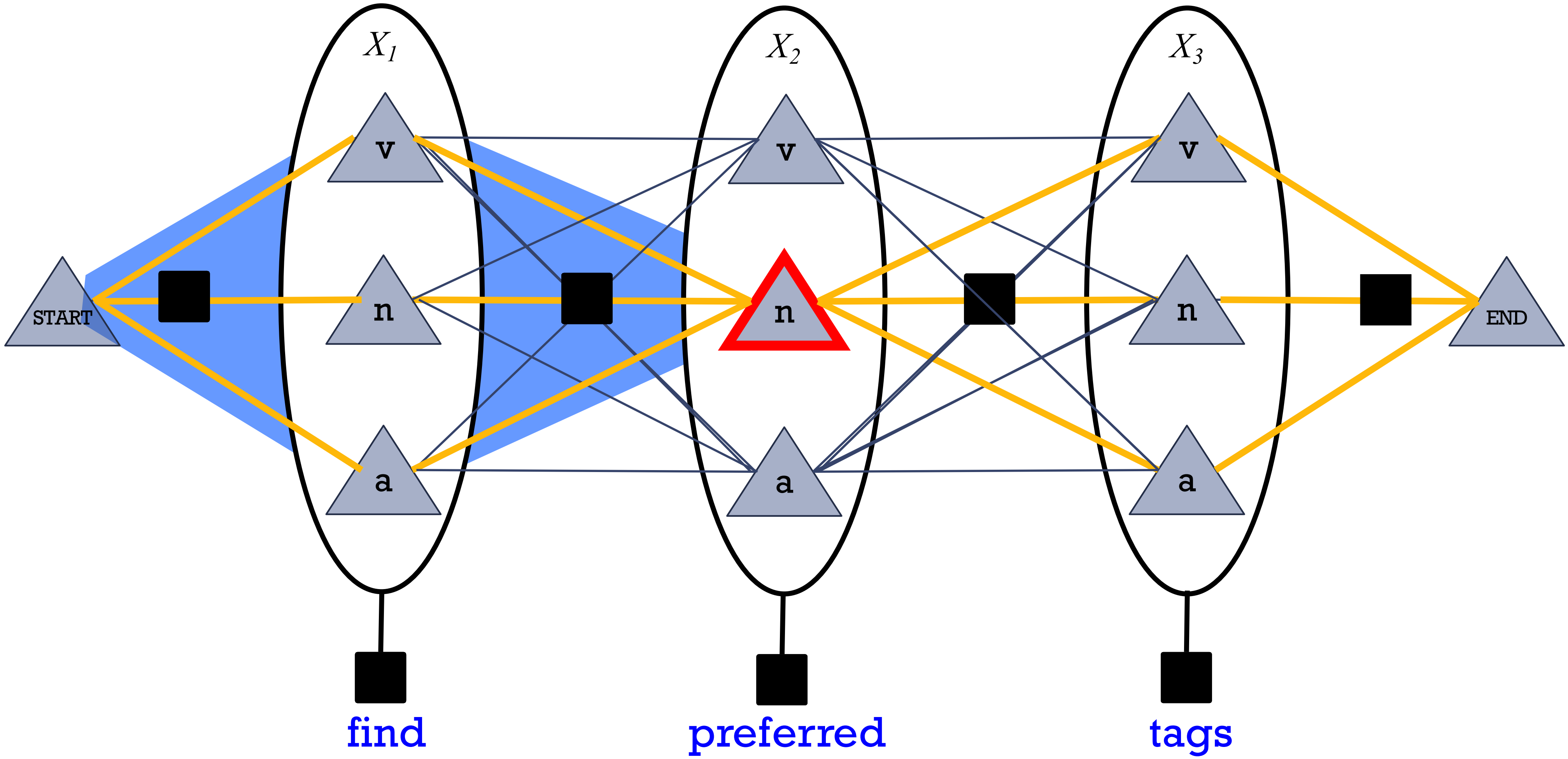
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = \text{a})$
  = (1/Z) * total weight of *all* paths through **v**

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 

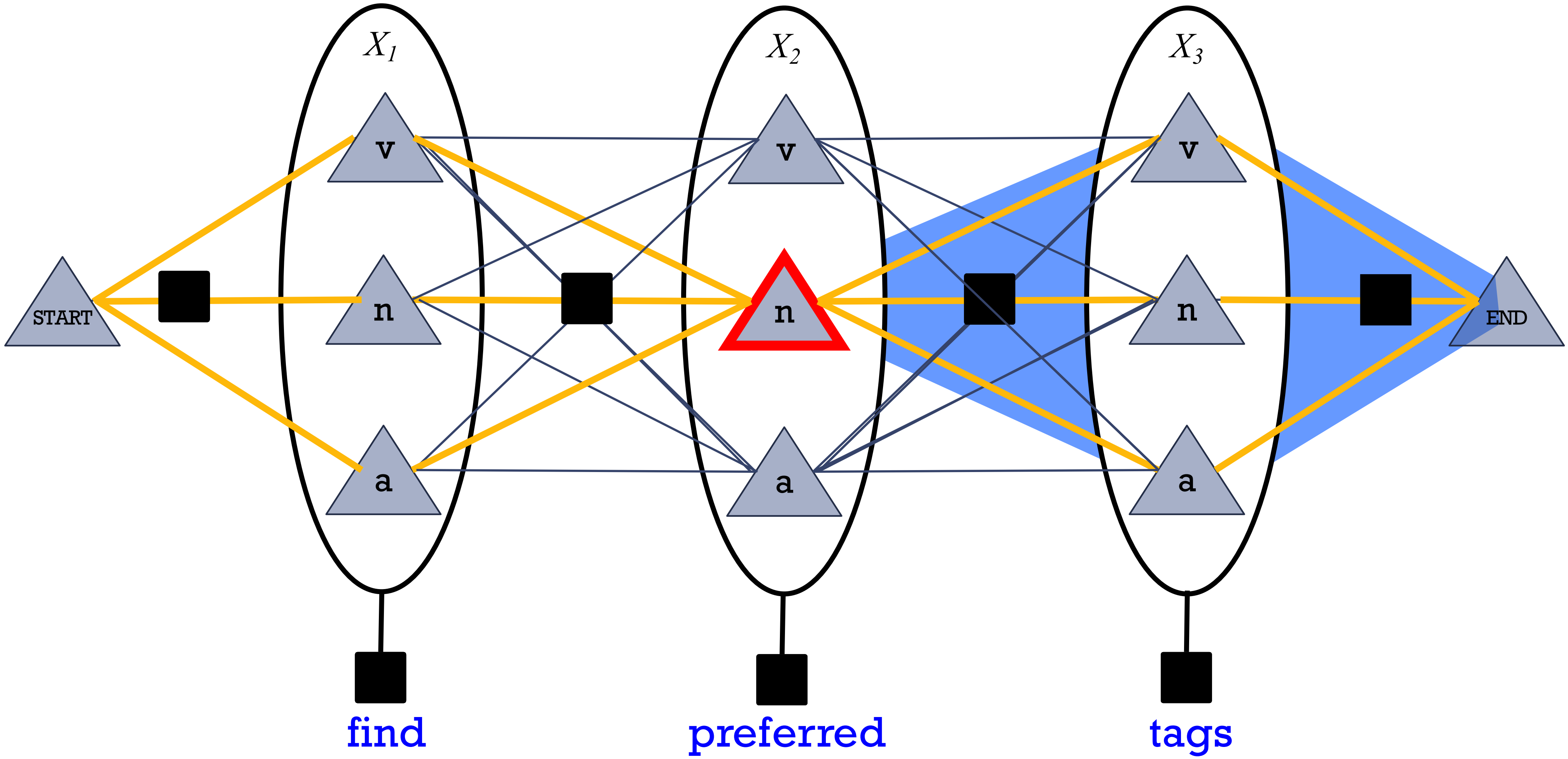
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

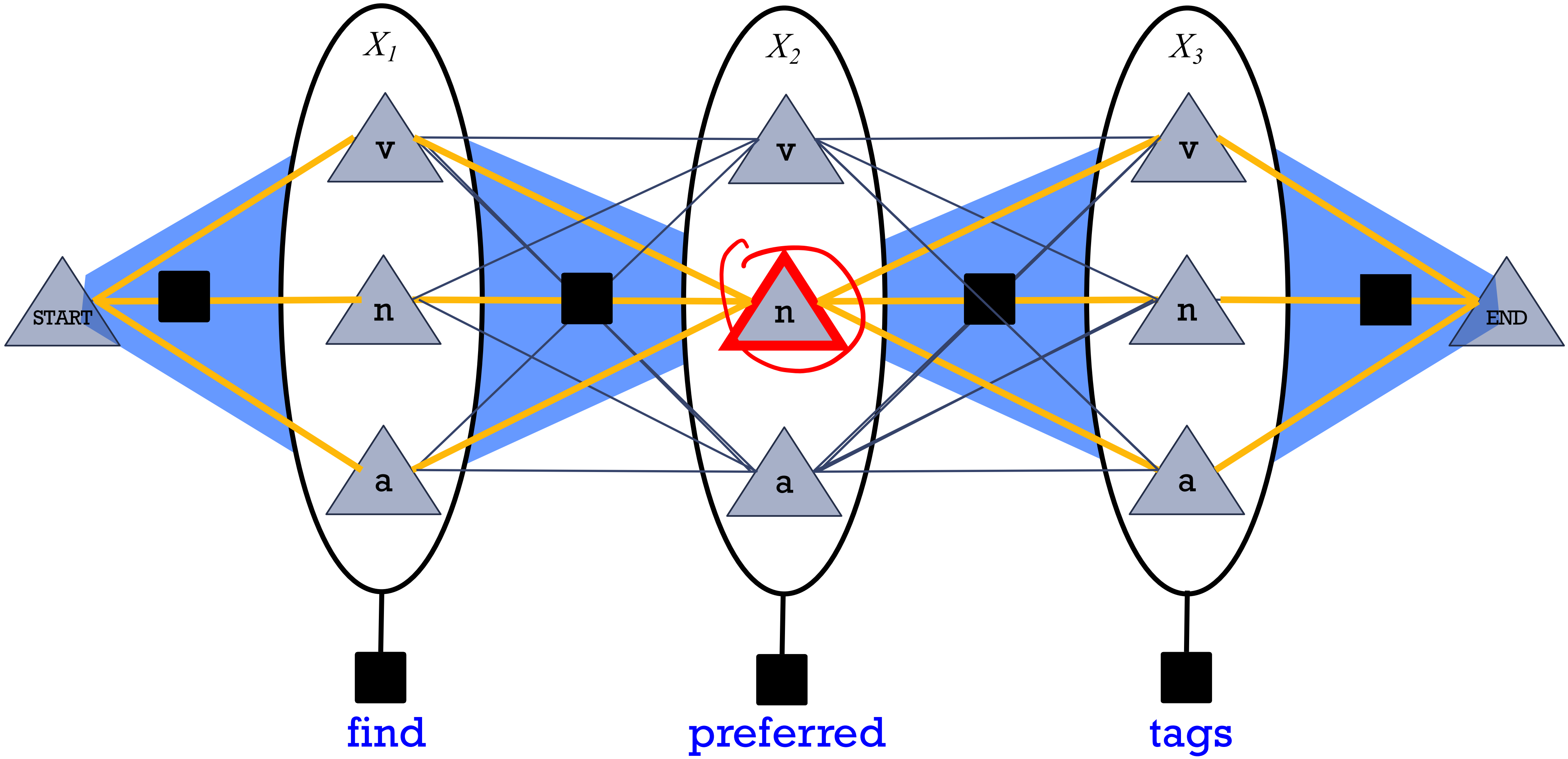
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = 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)

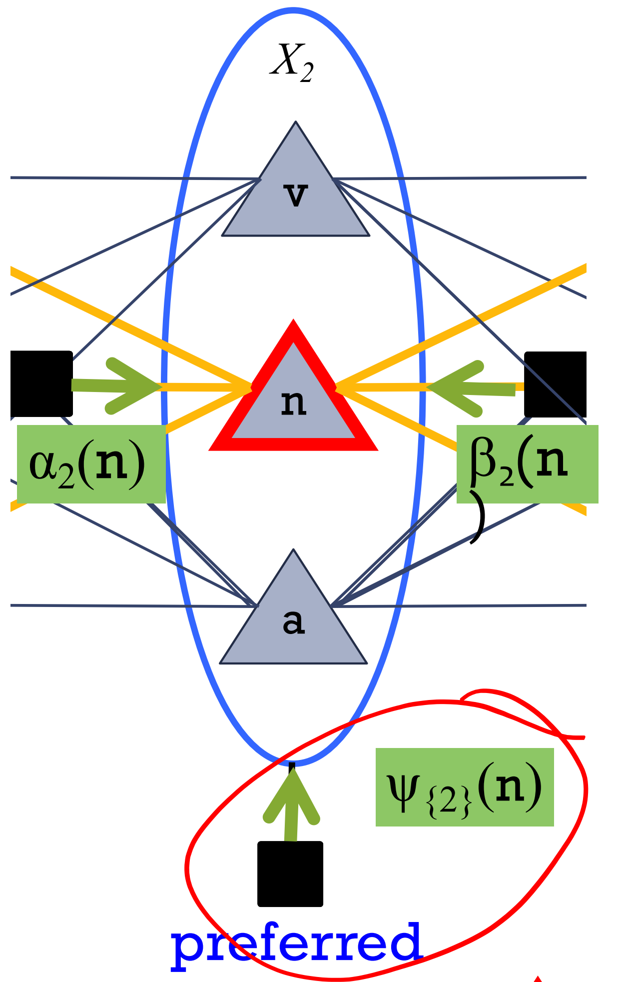
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

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

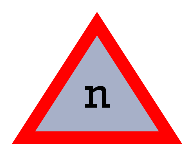
Forward-Backward Algorithm: Finds Marginals

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.

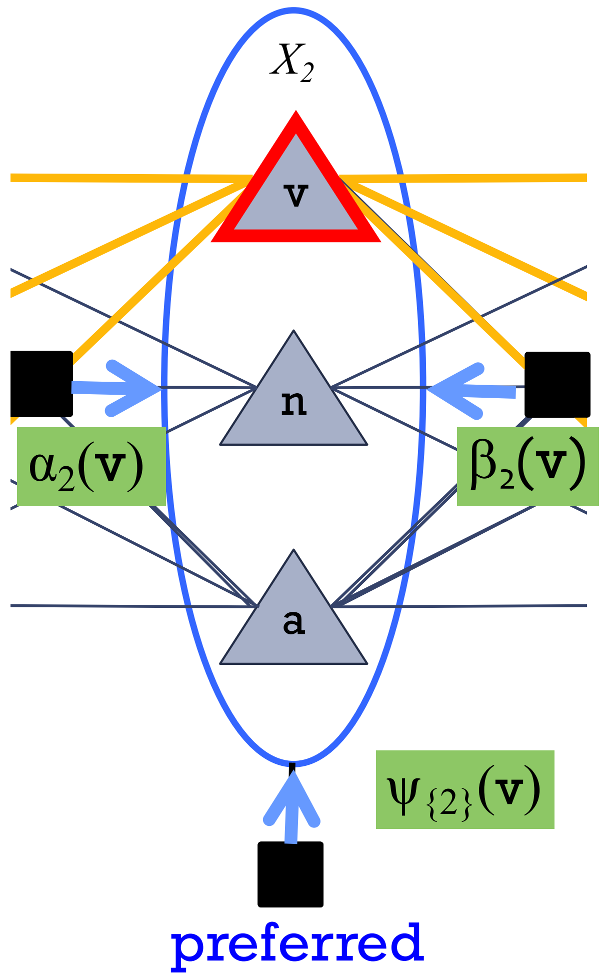


"belief that $X_2 = \mathbf{n}$"

total weight of *all* paths through 

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

Forward-Backward Algorithm: Finds Marginals



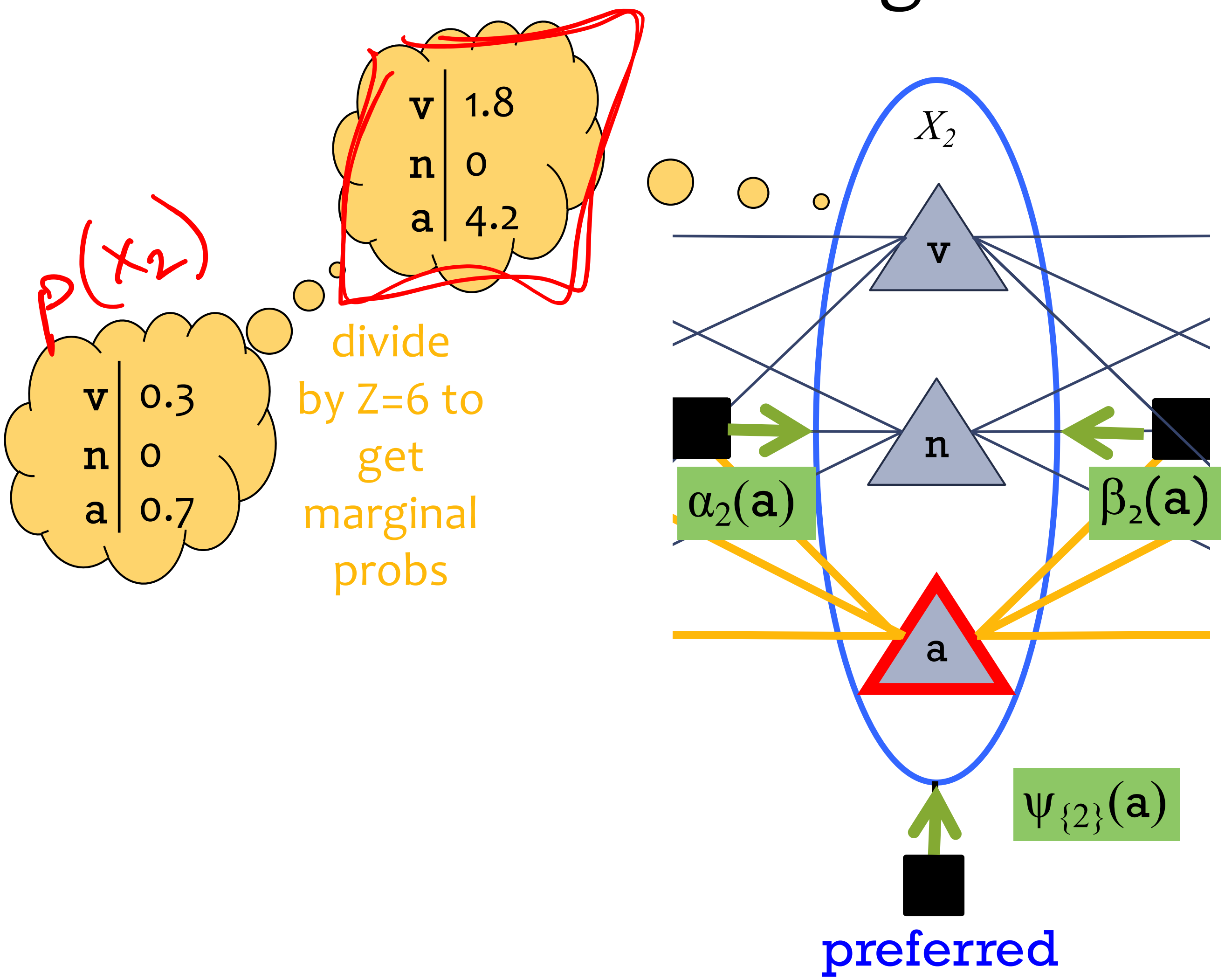
"belief that X_2 = **v**"

"belief that X_2 = **n**"

total weight of *all* paths through **v**

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

Forward-Backward Algorithm: Finds Marginals



$$\frac{\text{“belief that } X_2 = \mathbf{v}\text{”} \quad \text{“belief that } X_2 = \mathbf{n}\text{”} \quad \text{“belief that } X_2 = \mathbf{a}\text{”}}{\text{sum} = Z \text{ (total probability of } \textit{all} \text{ paths)}}$$

total weight of *all* paths through a

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

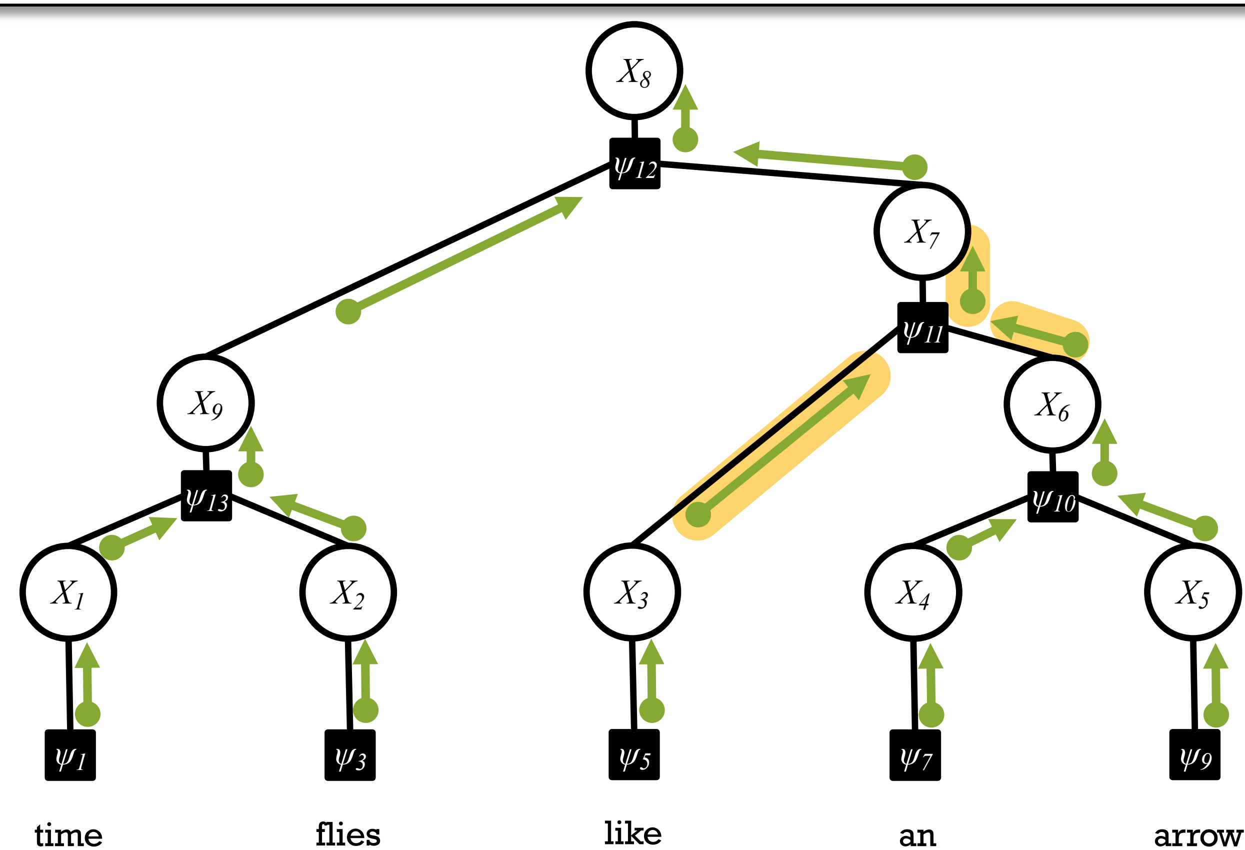
BP AS DYNAMIC PROGRAMMING

(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.

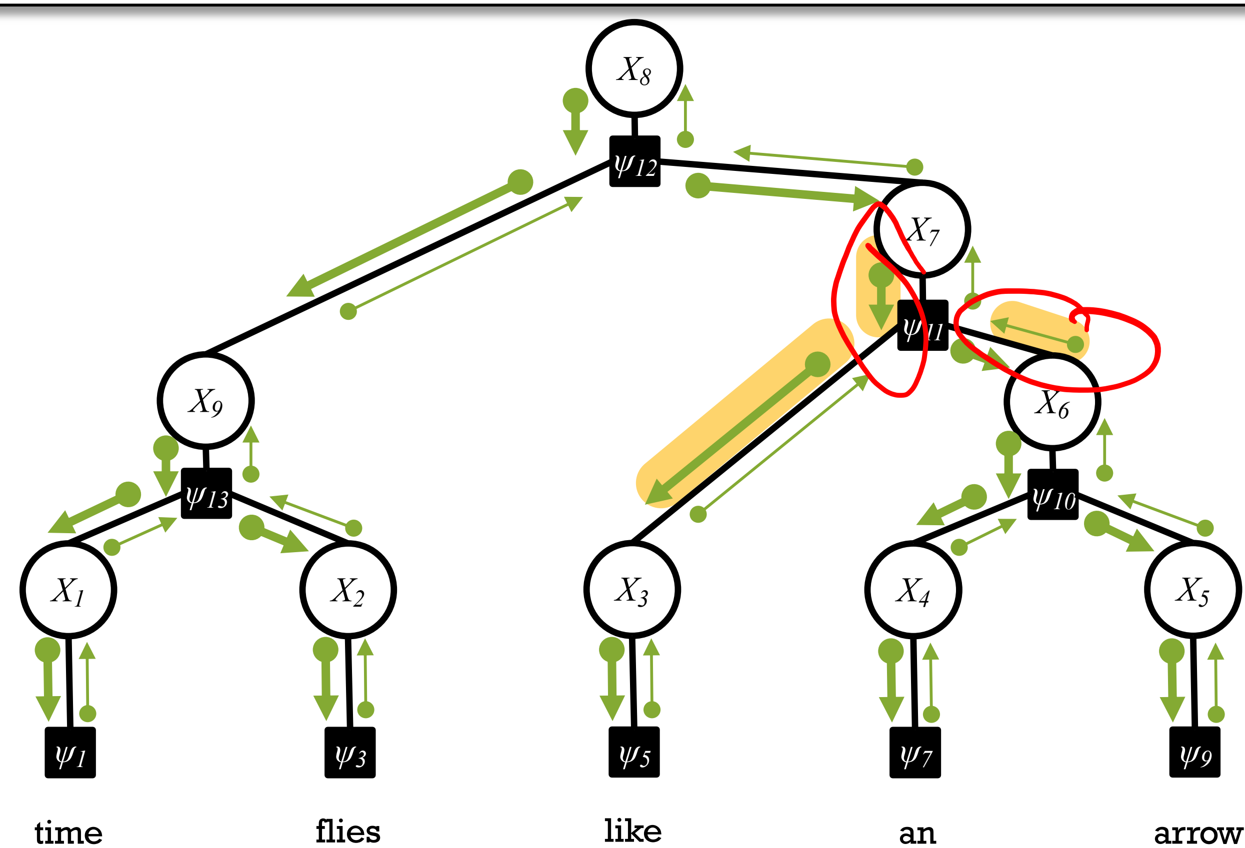


(Acyclic) Belief Propagation

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**.

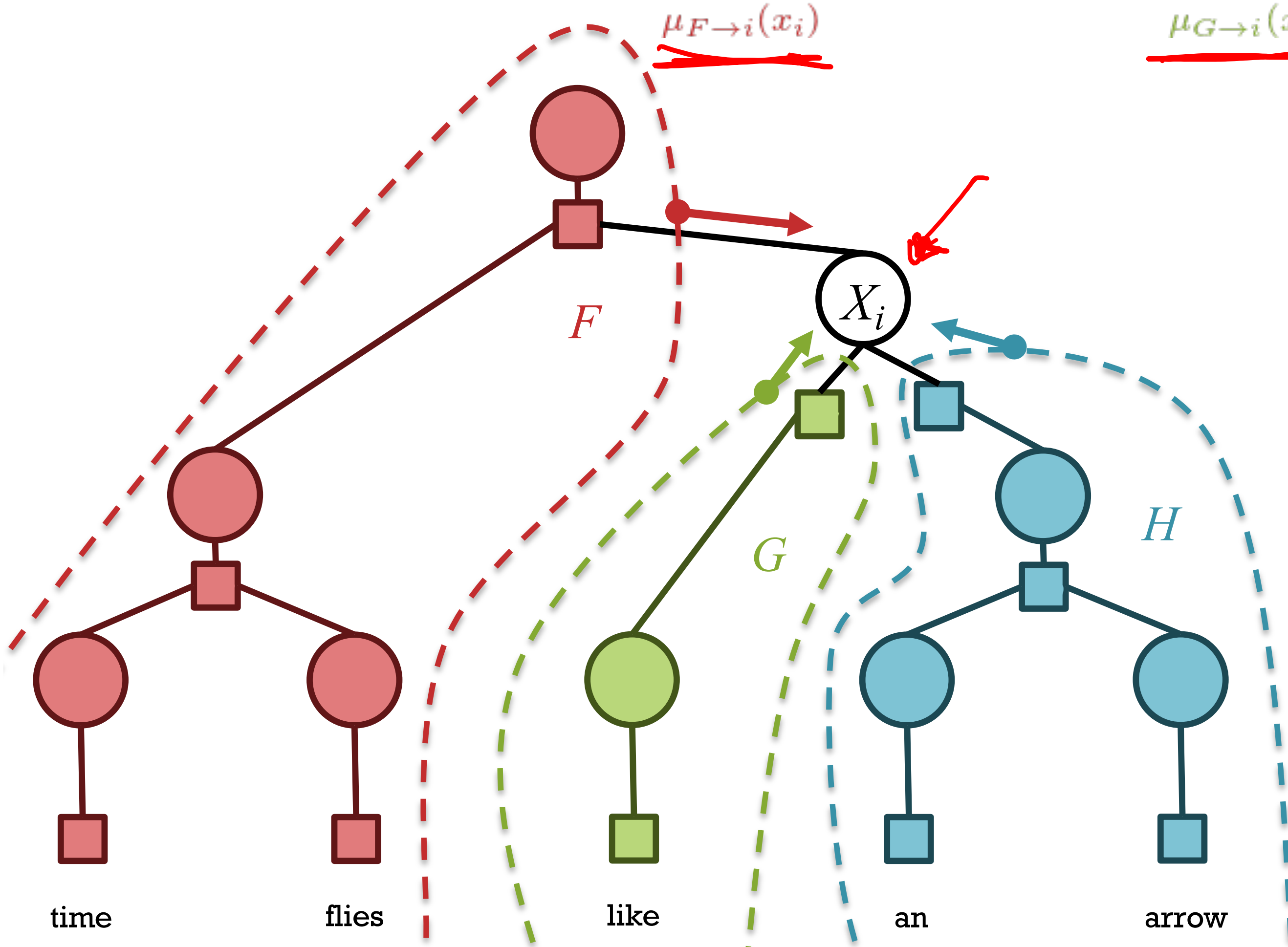
A node computes an outgoing message along an edge
only after it has received incoming messages along all its other edges.



Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \frac{1}{Z} \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \frac{1}{Z} \underbrace{\Bigg(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \Bigg)}_{\mu_{F \to i}(x_i)} \underbrace{\Bigg(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \Bigg)}_{\mu_{G \to i}(x_i)} \underbrace{\Bigg(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \Bigg)}_{\mu_{H \to i}(x_i)}$$



Subproblem:
Inference using just the factors in subgraph H

Figure adapted from Burkett & Klein (2012)

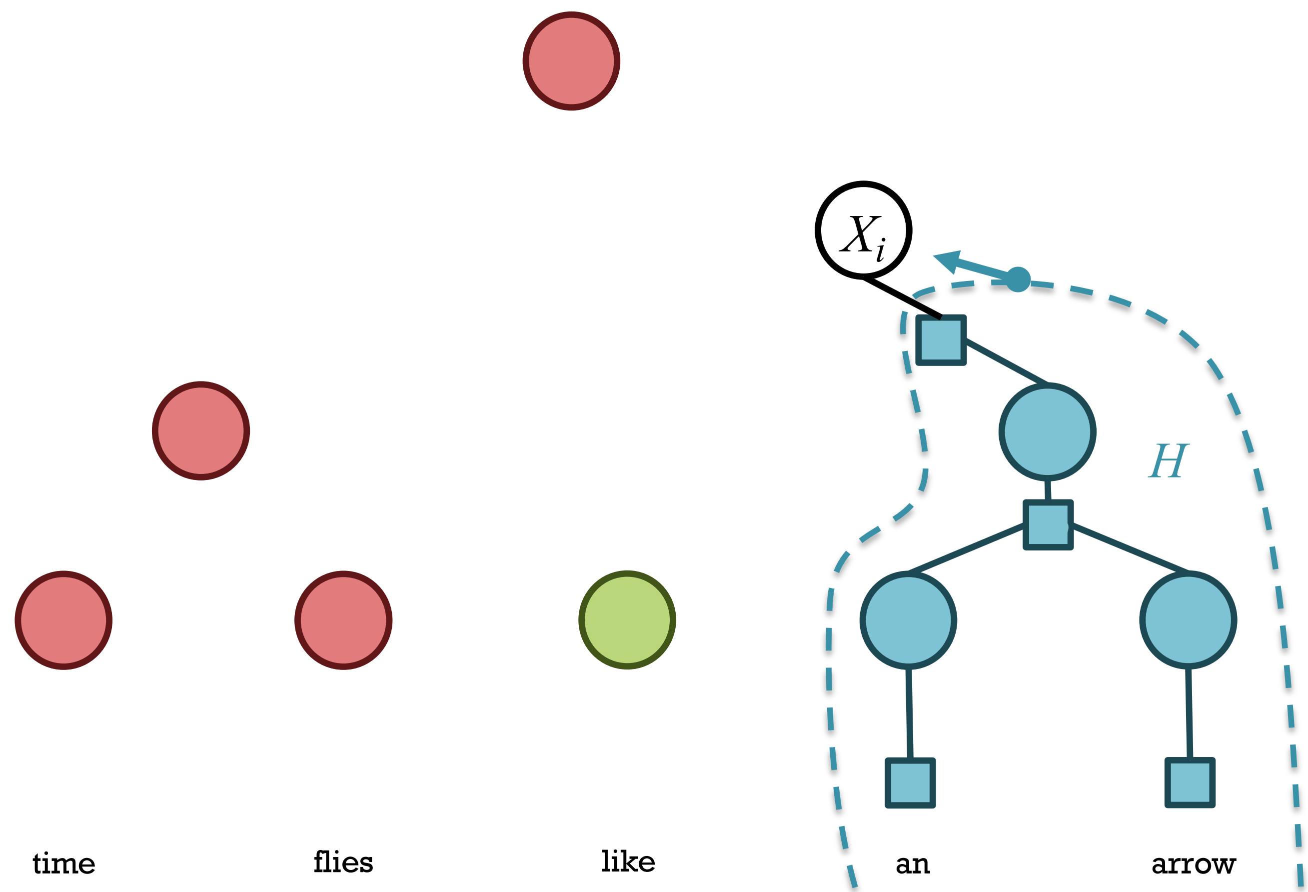
Acyclic BP as Dynamic Programming

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

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

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

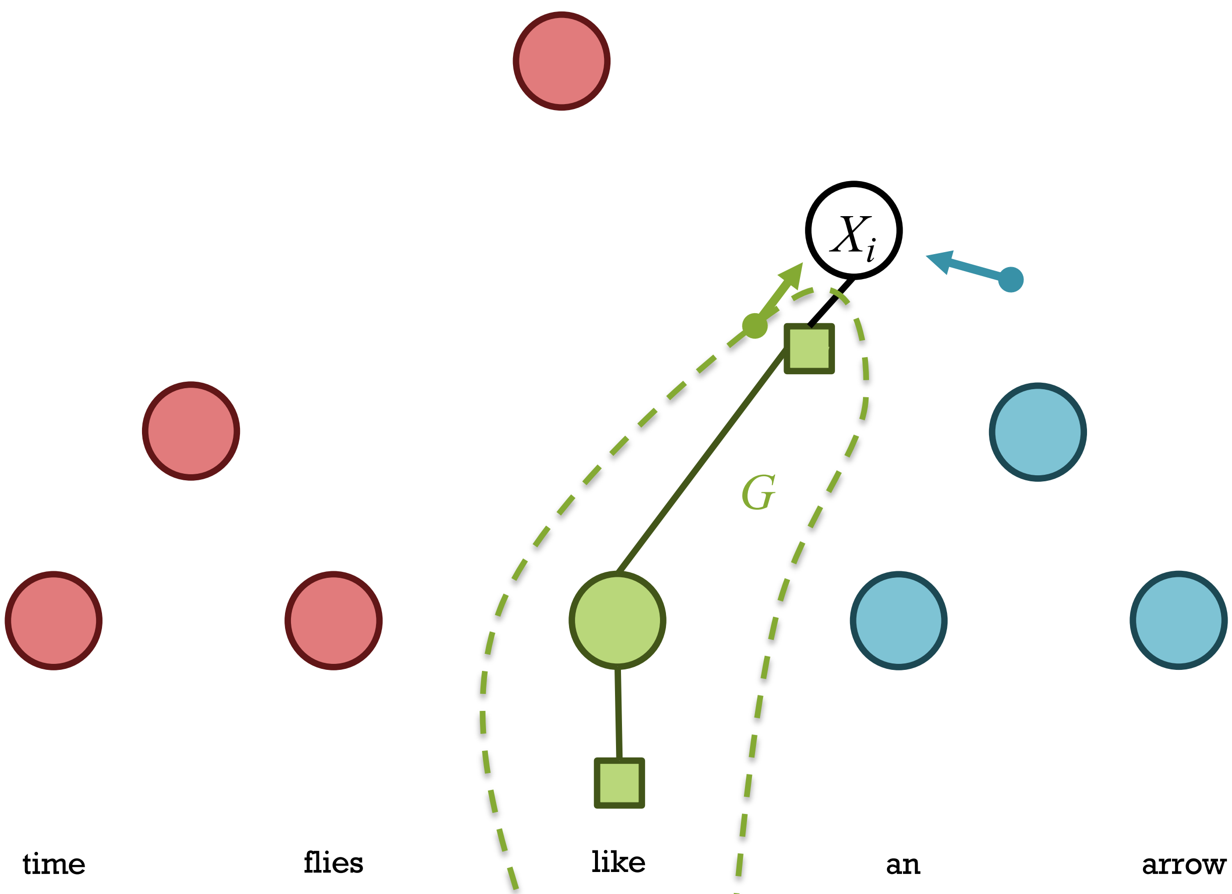


*Message **to** a variable*

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

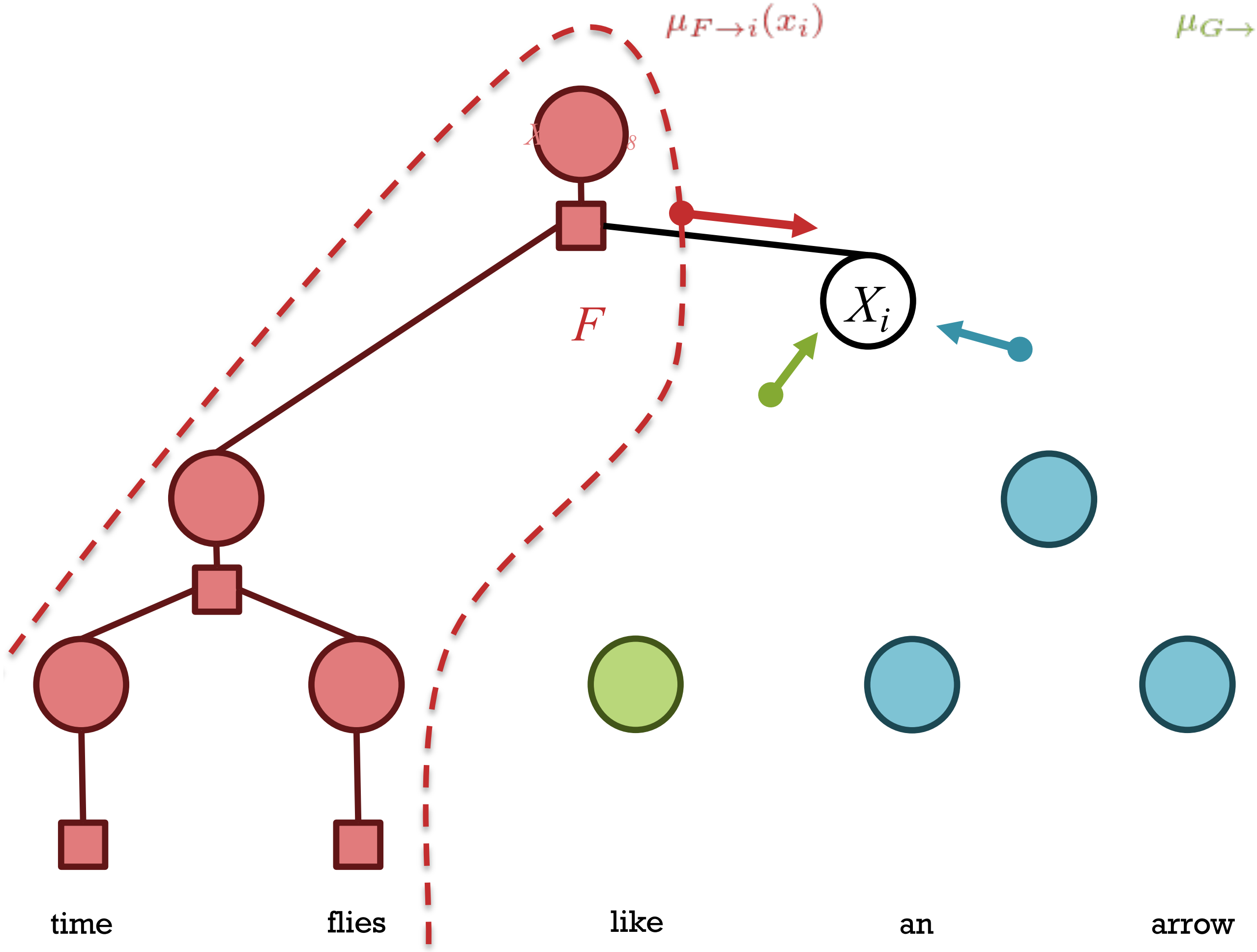
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*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

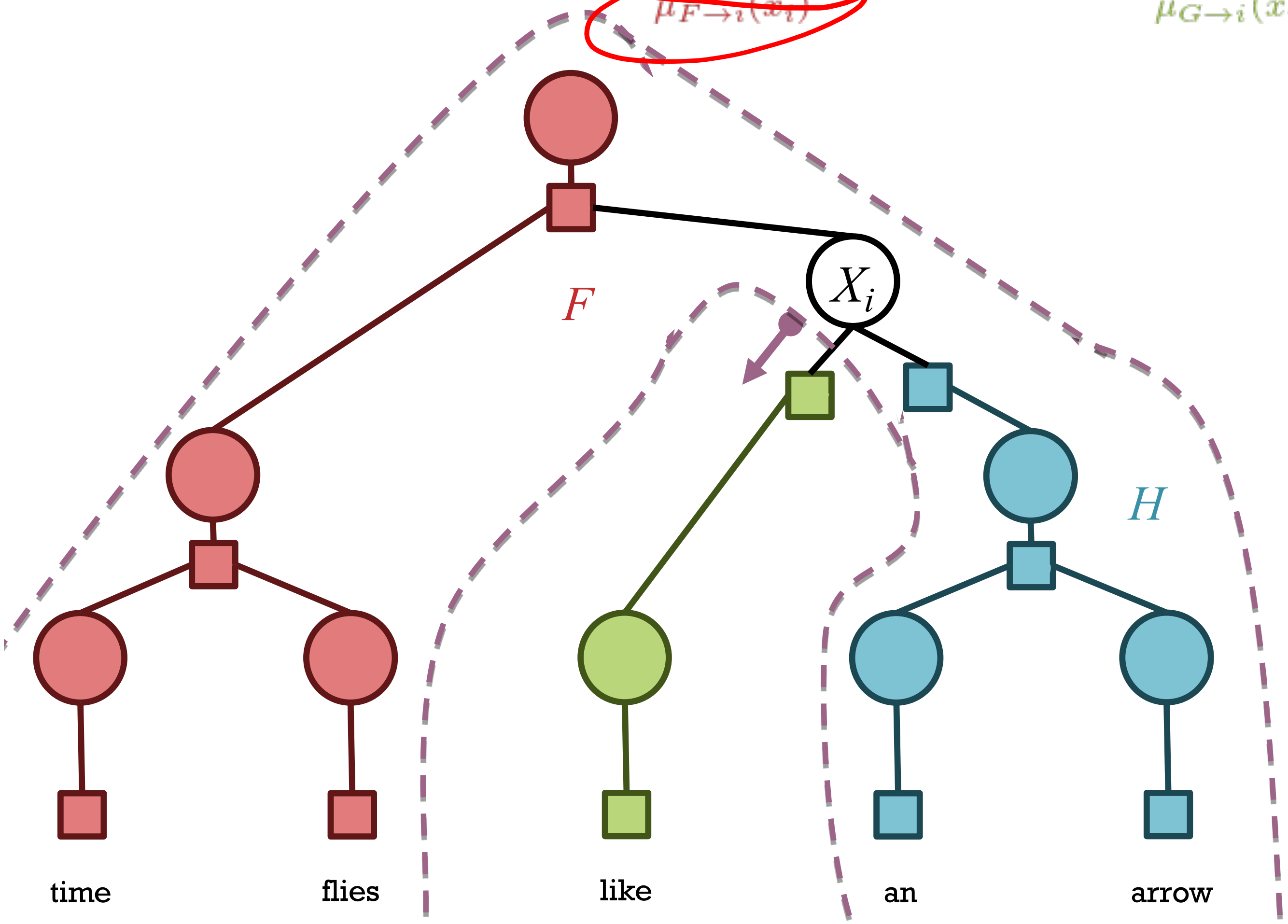
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

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

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



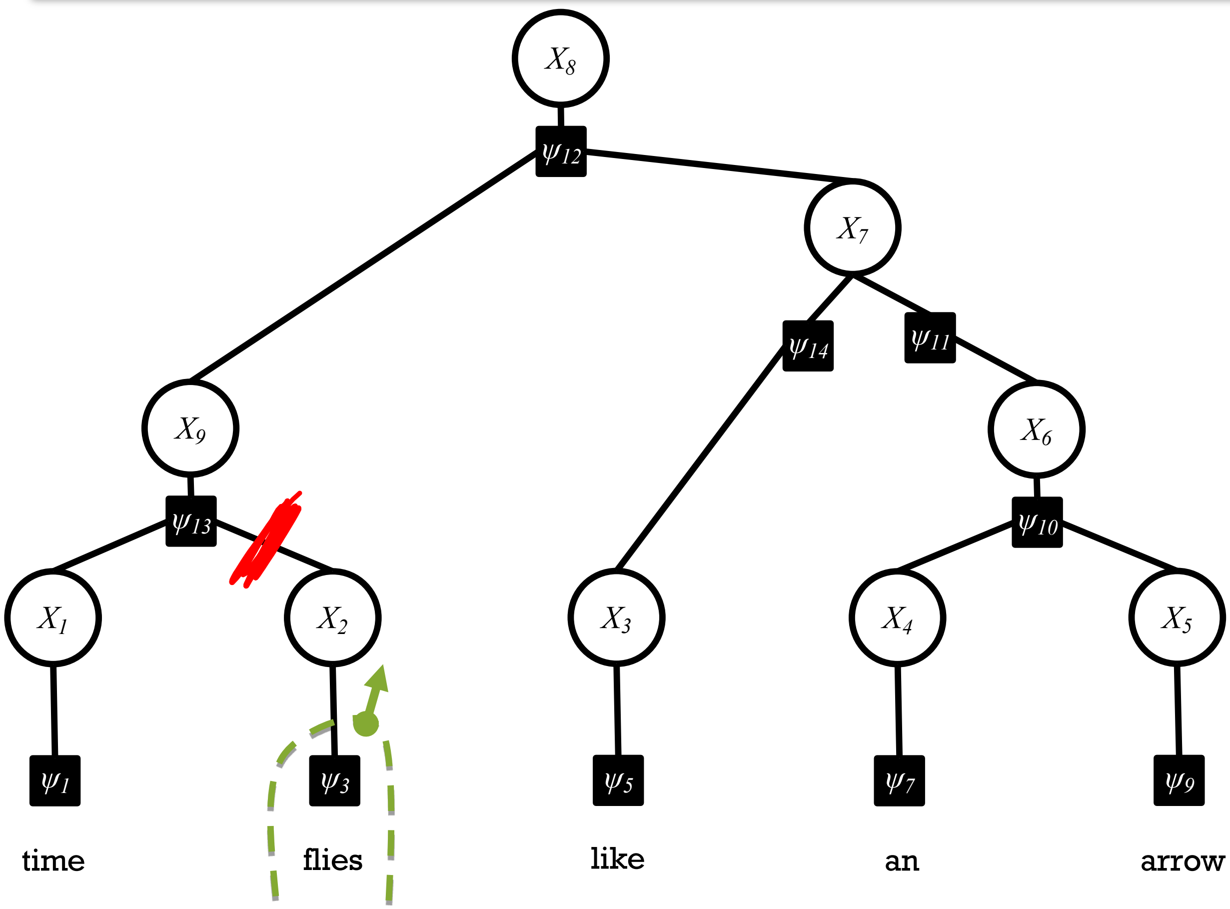
Subproblem:
Inference using just the factors in subgraph $F \cup H$

The marginal of X_i in that smaller model is the message sent by X_i <u>out of</u> subgraph $F \cup H$

*Message **from** a variable*

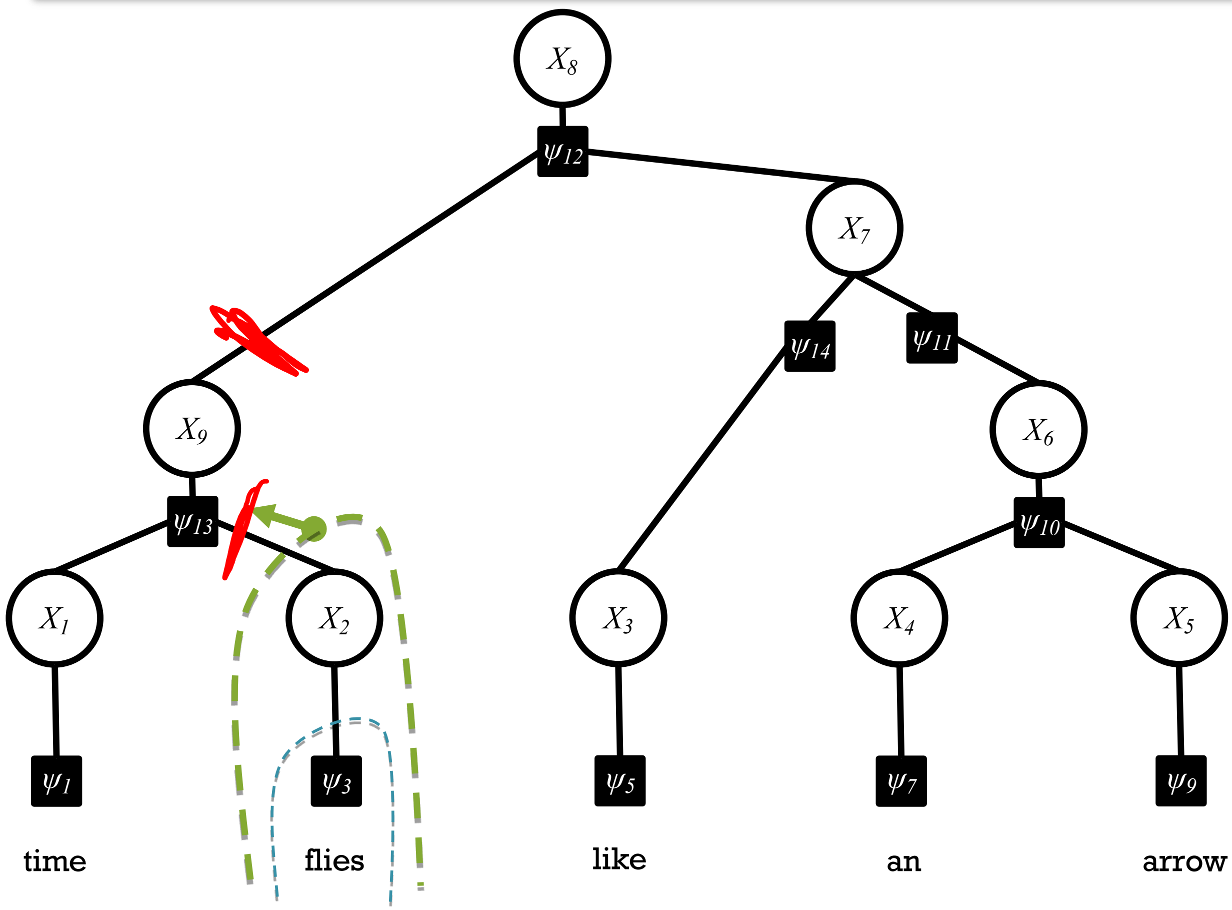
Acyclic BP as Dynamic Programming

- 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(s) 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.



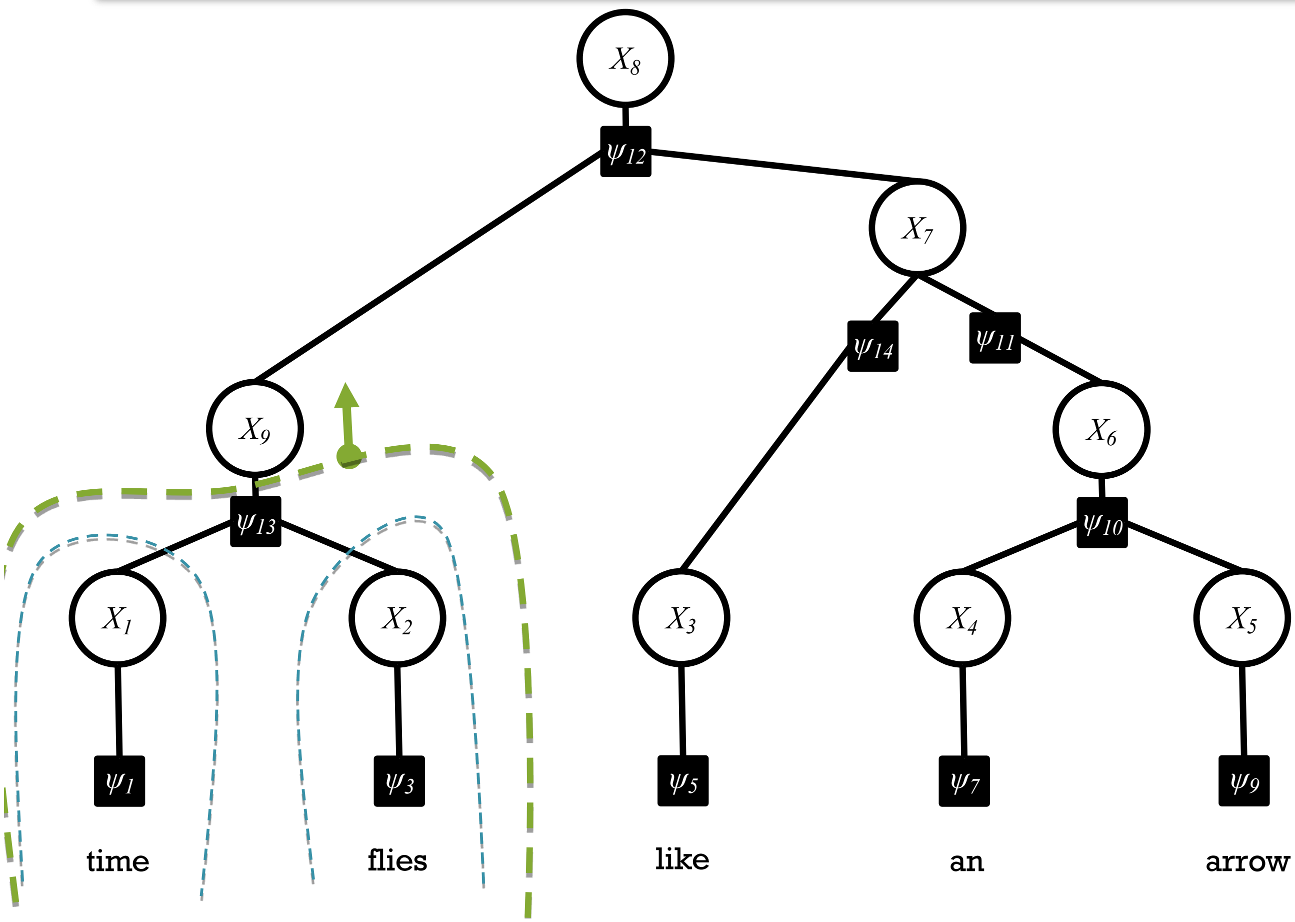
Acyclic BP as Dynamic Programming

- 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.



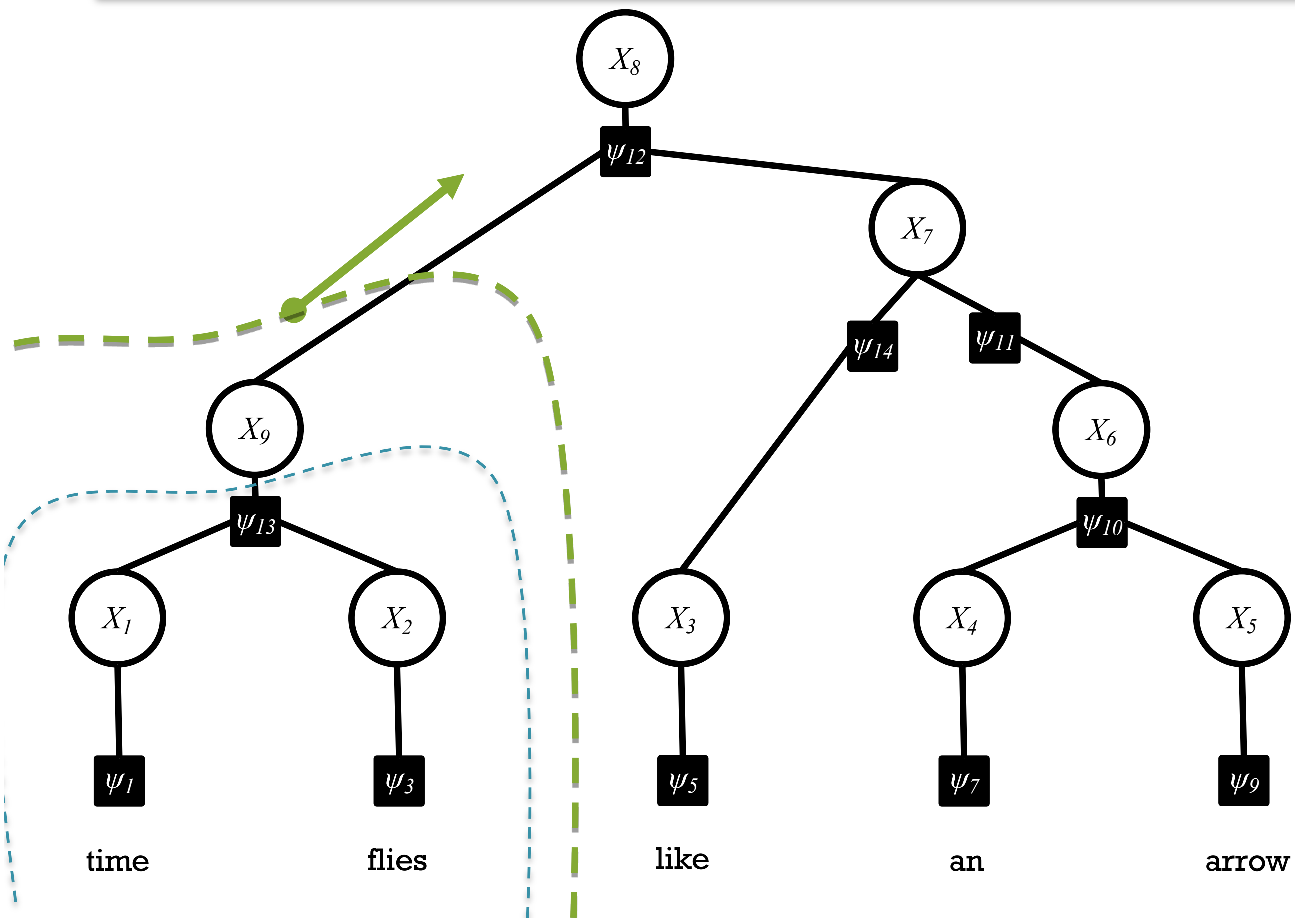
Acyclic BP as Dynamic Programming

- 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.



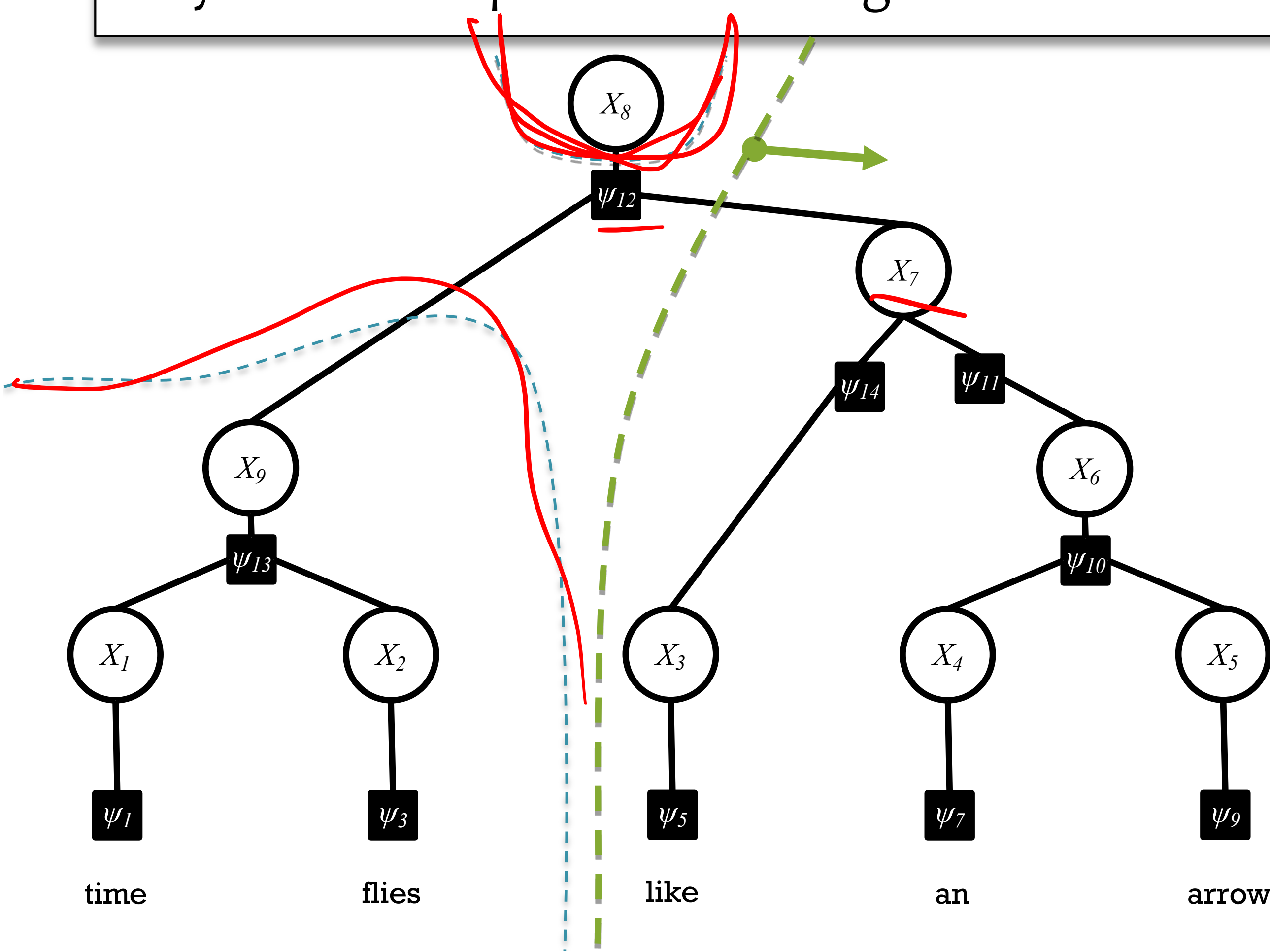
Acyclic BP as Dynamic Programming

- 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.



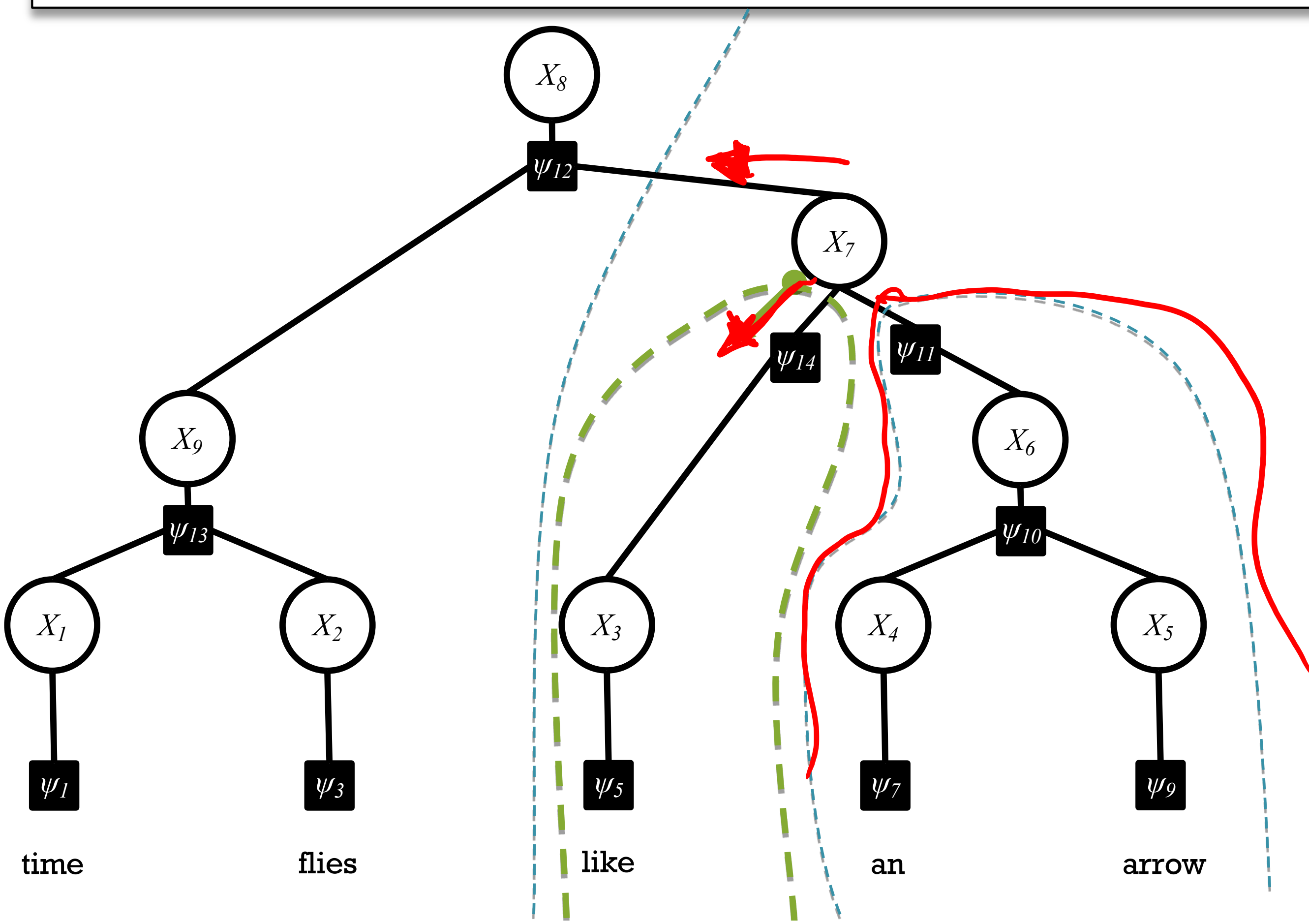
Acyclic BP as Dynamic Programming

- 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.



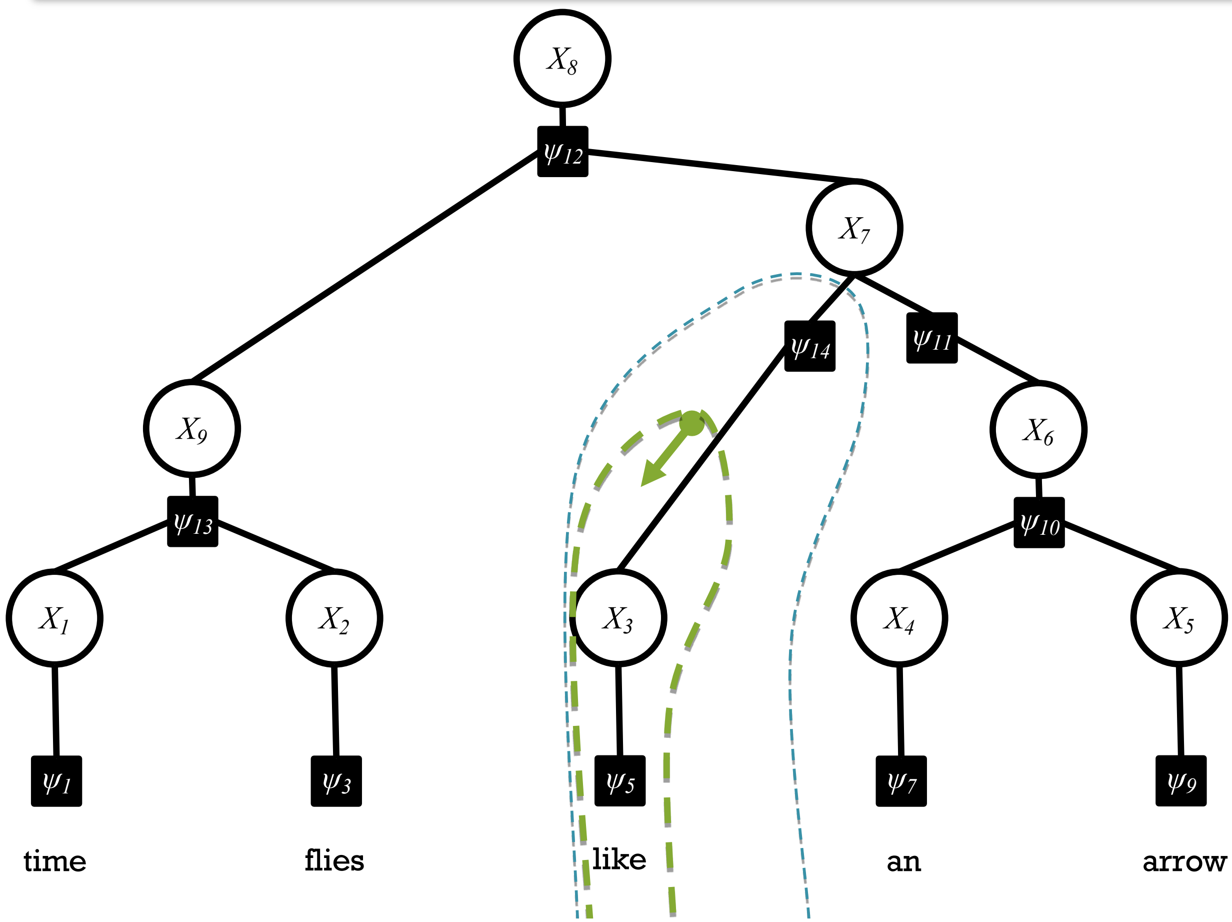
Acyclic BP as Dynamic Programming

- 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,
  $\boldsymbol{X}^* = \operatorname{argmax}_{\boldsymbol{X}} p(\boldsymbol{X})$

Max-product Belief Propagation

- Change the sum to a max:

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

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

- **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:

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

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

- **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

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

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

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

$\otimes = +$

$\oplus = \text{log-add}(a,b) = \log(\exp(a) + \exp(b))$

Exact inference for linear chain models

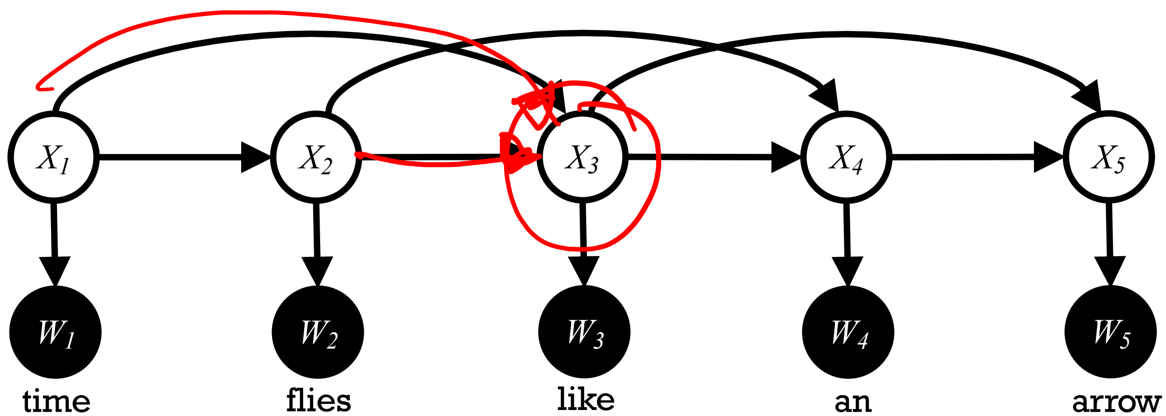
FORWARD-BACKWARD AND VITERBI ALGORITHMS

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**

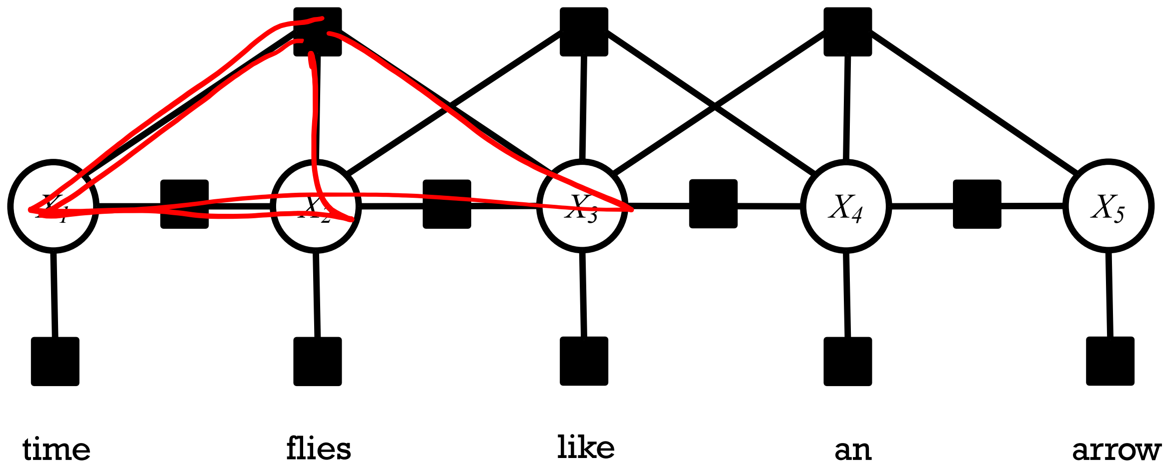
Forward-Backward Algorithm

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



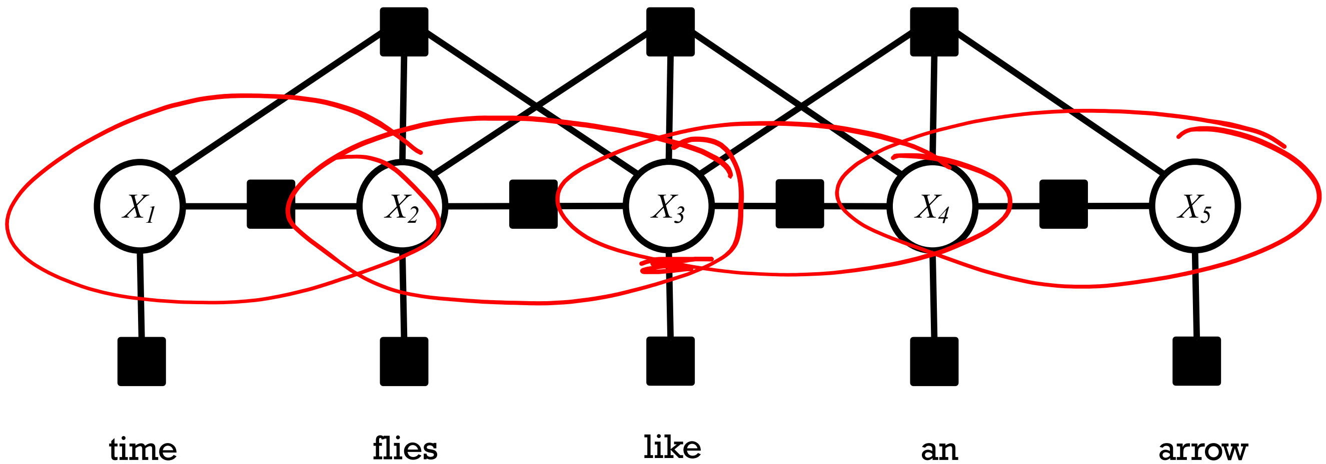
Forward-Backward Algorithm

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.

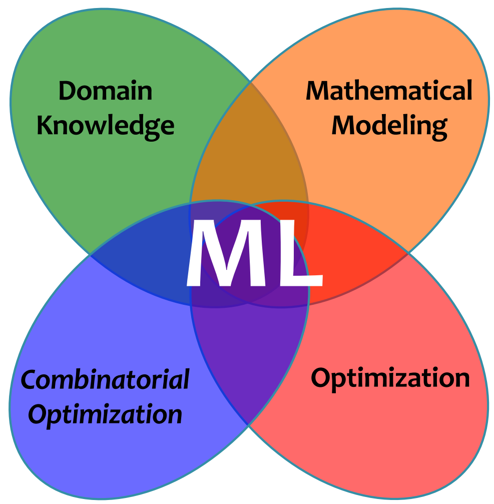
LEARNING FOR MRFS + CRFs

Machine Learning

The **data** inspires the structures we want to predict

Our **model** defines a score for each structure

It also tells us what to optimize



Learning tunes the parameters of the model

Inference finds {best structure, marginals, partition function} for a new observation

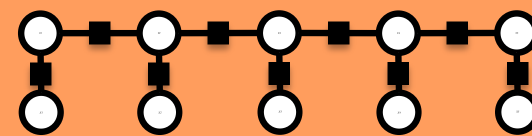
(**Inference** is usually called as a subroutine in learning)

1. Data

$$\mathcal{D} = \{\boxed{\boldsymbol{x}^{(n)}}\}_{n=1}^{N}$$

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

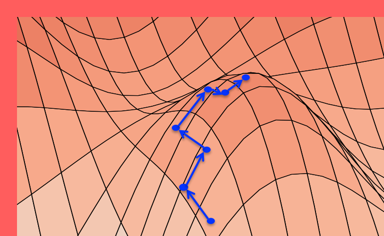
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

$$\hat{\boldsymbol{x}} = \operatorname*{argmax}_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

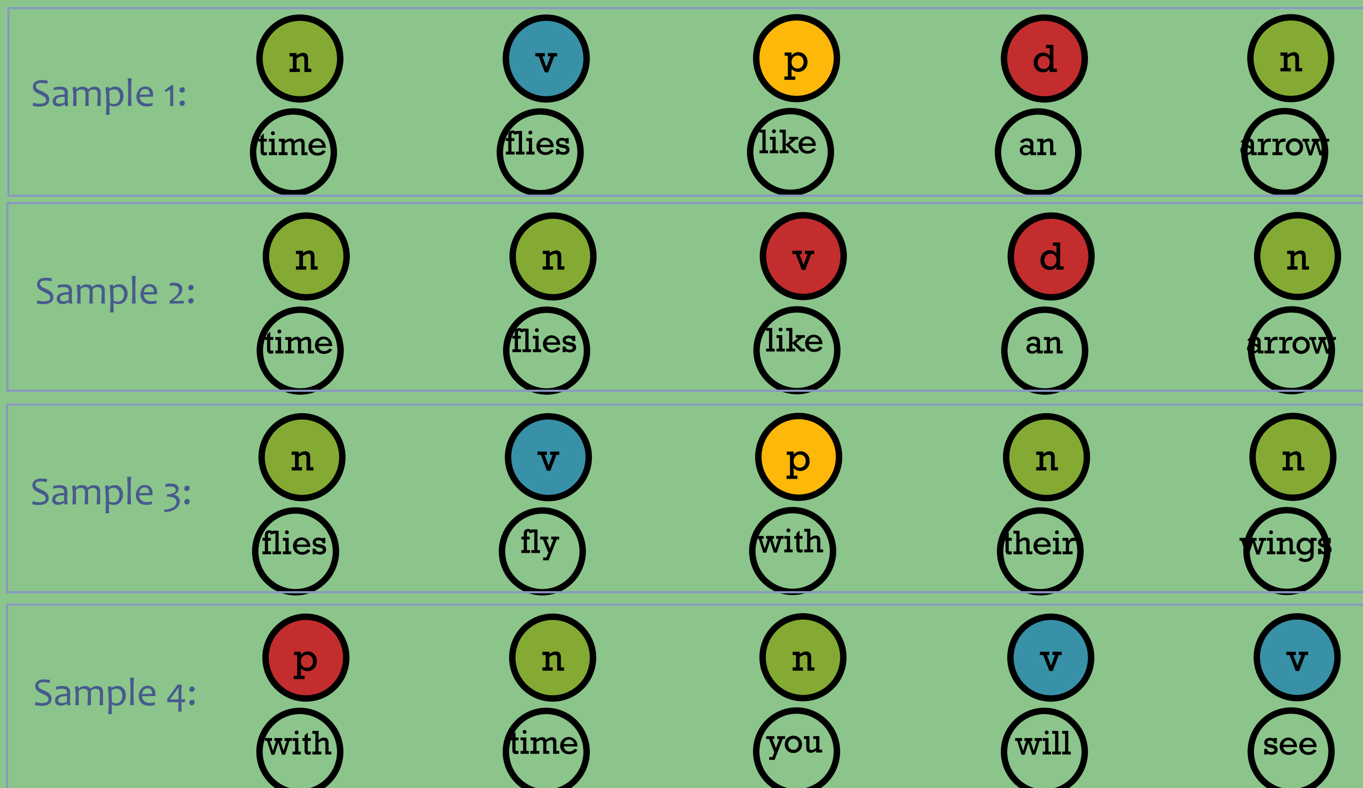
4. Learning

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \mathcal{D})$$

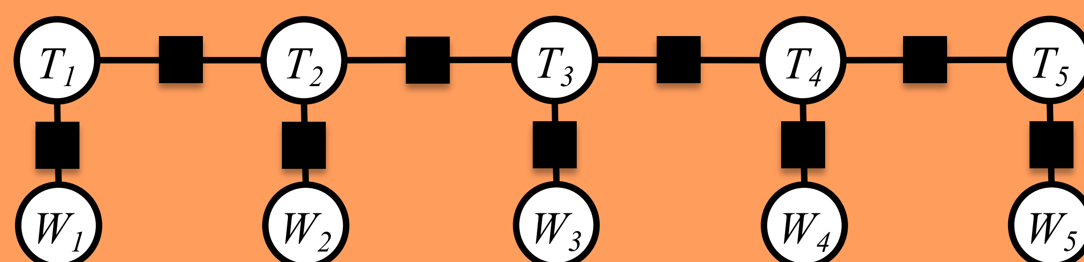


1. Data

Given training examples: $\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$



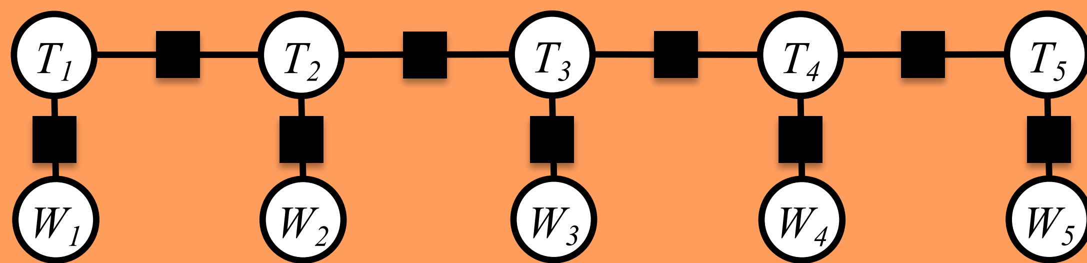
2. Model



2. Model

Define the model to be an MRF:

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

Choose the objective to be log-likelihood:

(Assign high probability to the things we observe and low probability to everything else)

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

3. Objective

Choose the objective to be log-likelihood:

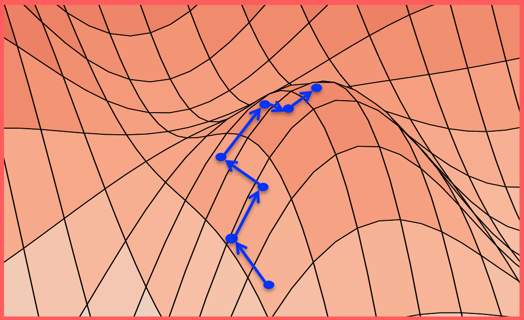
(Assign high probability to the things we observe and low probability to everything else)

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

4. Learning

Tune the parameters to maximize the objective function

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \mathcal{D})$$



3. Objective

Choose the objective to be log-likelihood:

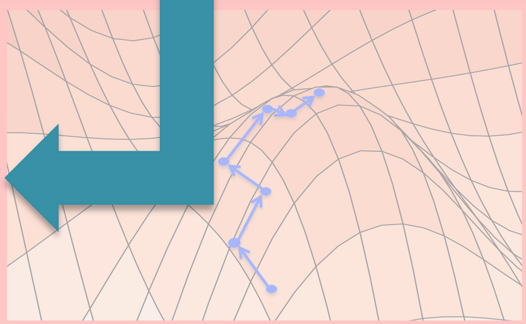
(Assign high probability to the things we observe and low probability to everything else)

Goals for Today's Lecture

1. Consider different parameterizations
2. Optimize this objective function

Tune the parameter
function

$$\theta^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \ell(\boldsymbol{\theta}; \mathcal{D})$$



5. Inference

Three Tasks:

1. Marginal Inference
Compute marginals of variables and cliques

$$p(x_i) = \sum_{\boldsymbol{x}' : x'_i = x_i} p(\boldsymbol{x}' \mid \boldsymbol{\theta}) \qquad \Big| \qquad p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function
Compute the normalization constant

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference
Compute variable assignment with highest probability

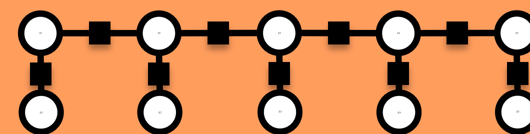
$$\hat{\boldsymbol{x}} = \operatorname*{argmax}_{\boldsymbol{x}} \; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

1. Data

$$\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$$

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

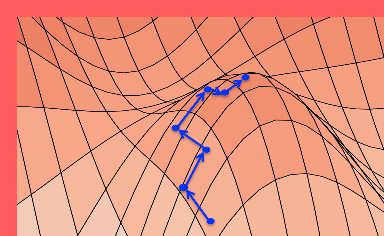
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

$$\hat{\boldsymbol{x}} = \operatorname*{argmax}_{\boldsymbol{x}} \; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

4. Learning

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \mathcal{D})$$



MLE for Undirected GMs

- Today's parameter estimation assumptions:
  1. The graphical model structure is given
  2. Every variable appears in the training examples

Questions

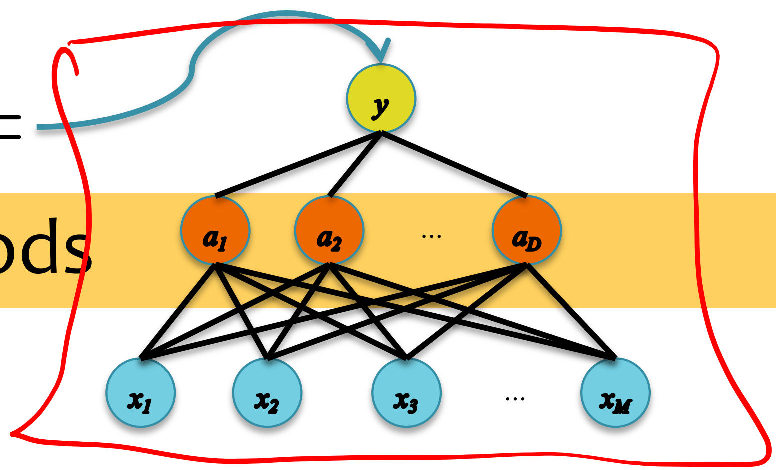
1. What does the **likelihood objective** accomplish?
2. Is likelihood the ***right*** **objective** function?
3. **How do we optimize** the objective function (i.e. learn)?
4. What **guarantees** does the optimizer provide?
5. (What is the **mapping from data → model**? In what ways can we incorporate our domain knowledge? How does this impact learning?)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



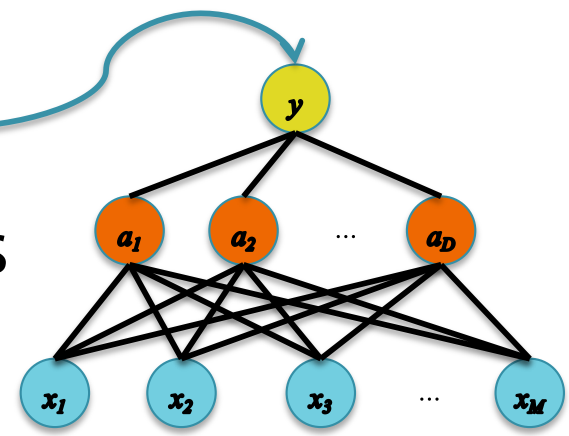
LOG-LINEAR PARAMETERIZATION OF CONDITIONAL RANDOM FIELD

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

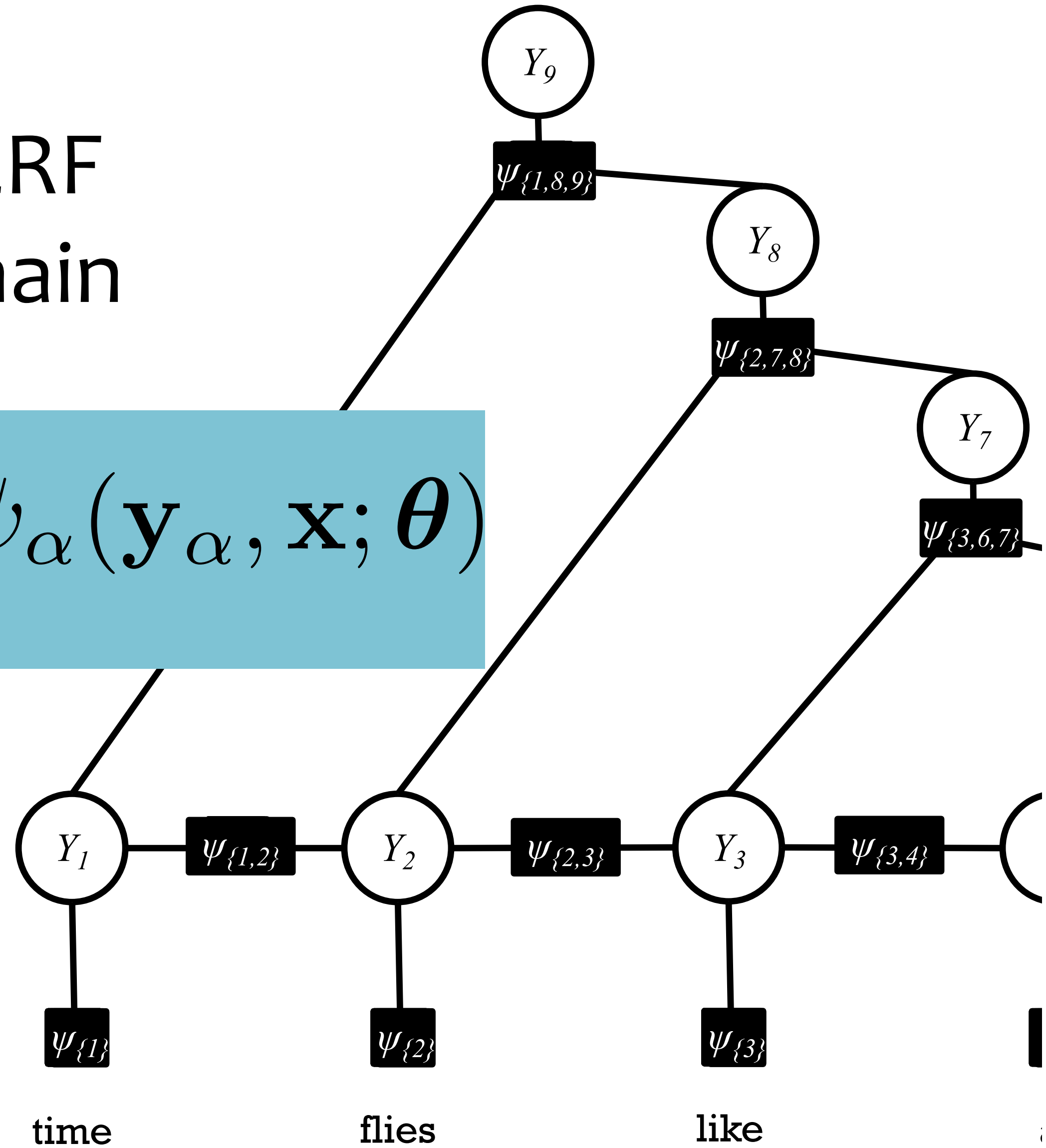
- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



General CRF

The topology of the graphical model for a CRF doesn't have to be a chain

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha}, \mathbf{x}; \boldsymbol{\theta})$$



Log-linear CRF Parameterization

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\alpha} \underline{\psi_\alpha}(\mathbf{y}_\alpha, \underline{\mathbf{x}}; \underline{\boldsymbol{\theta}})$$

Define each potential function in terms of a fixed set of feature functions:

$$\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\underline{\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta})} = \underline{\exp}(\underline{\boldsymbol{\theta}} \cdot \underline{\mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x})})$$

Predicted variables

Observed variables

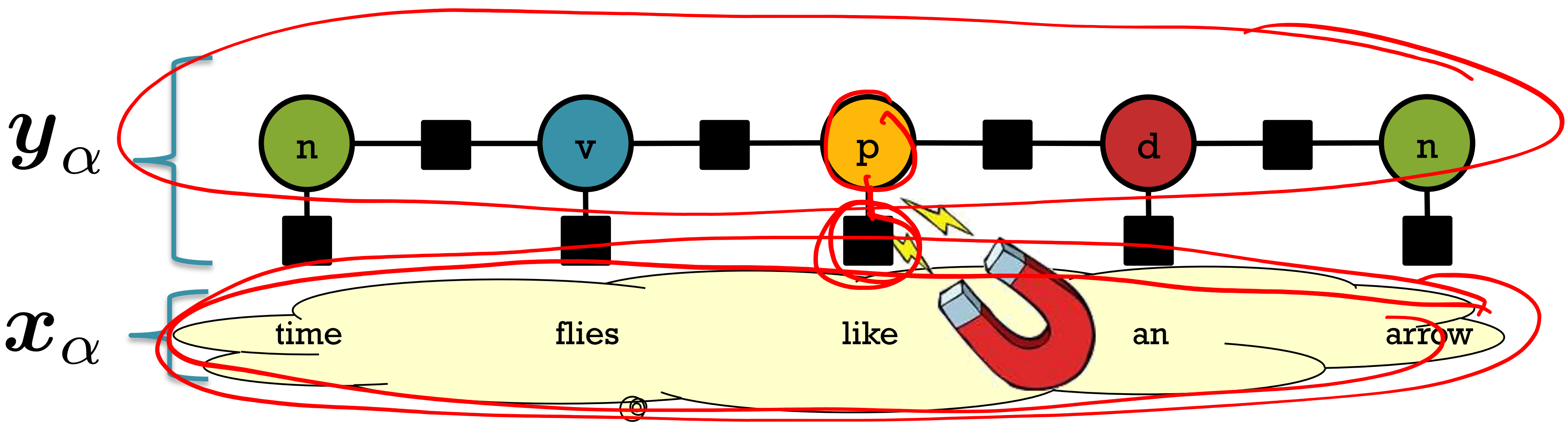
positive or non-negative

$|\theta| = |f_\alpha(\cdot)|$

Log-linear CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

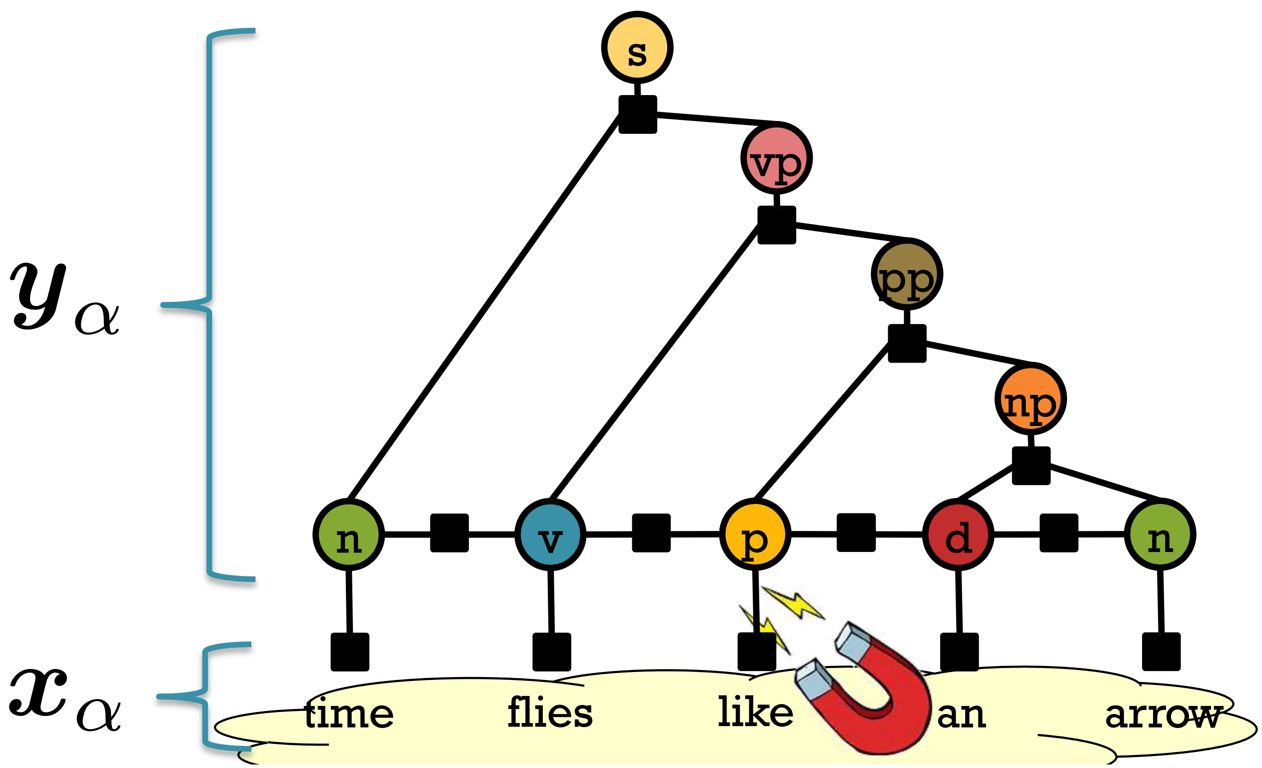
$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$



Log-linear CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$



Conditional Random Fields (CRFs) for time series data

LINEAR-CHAIN CRFS (LOG-LINEAR PARAMETERIZATION)

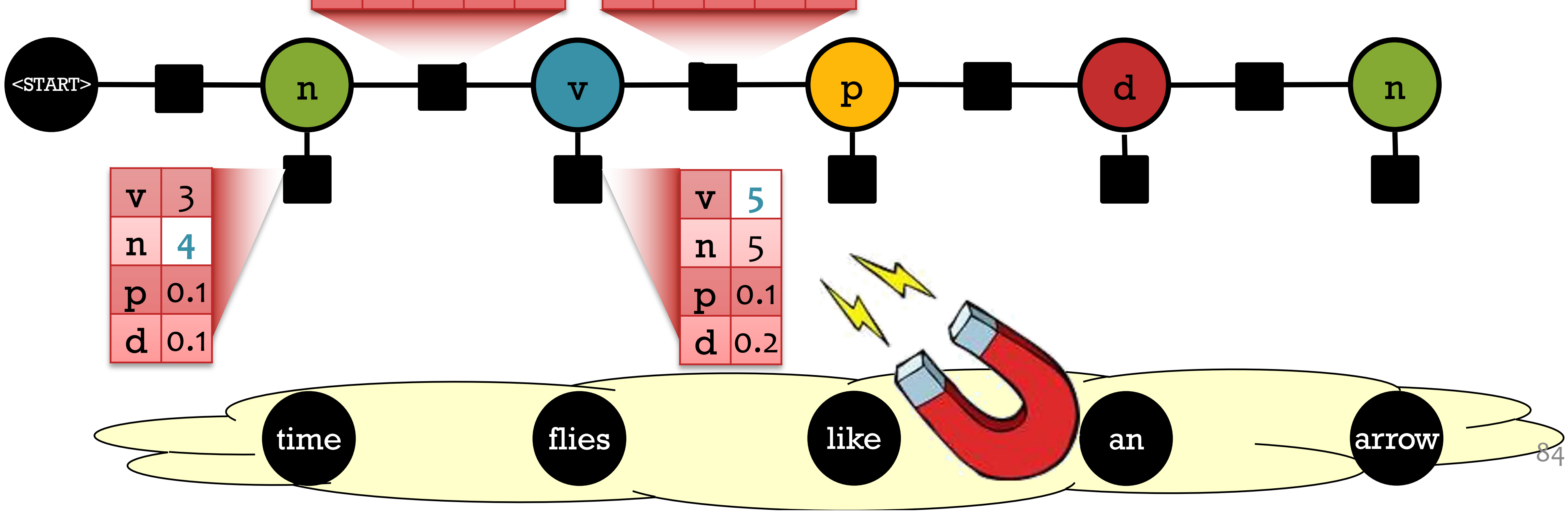
Conditional Random Field (CRF)

Conditional distribution over tags X_i given words w_i.
The factors and Z are now specific to the sentence w.

$$p(\text{n, v, p, d, n} \mid \text{time, flies, like, an, arrow}) \quad = \frac{1}{Z}\,(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

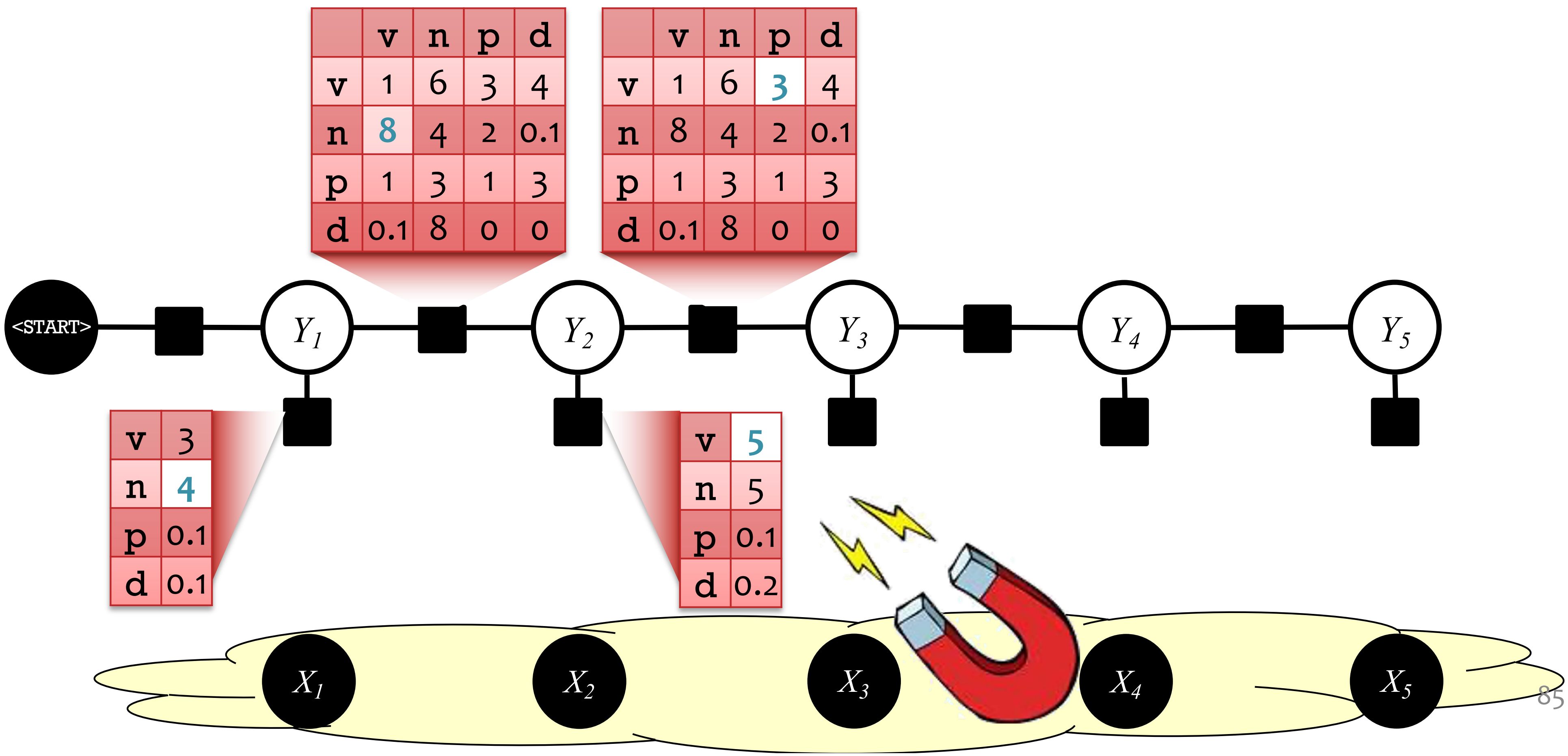


v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

Conditional Random Field (CRF)

Recall: Shaded nodes in a graphical model are **observed**

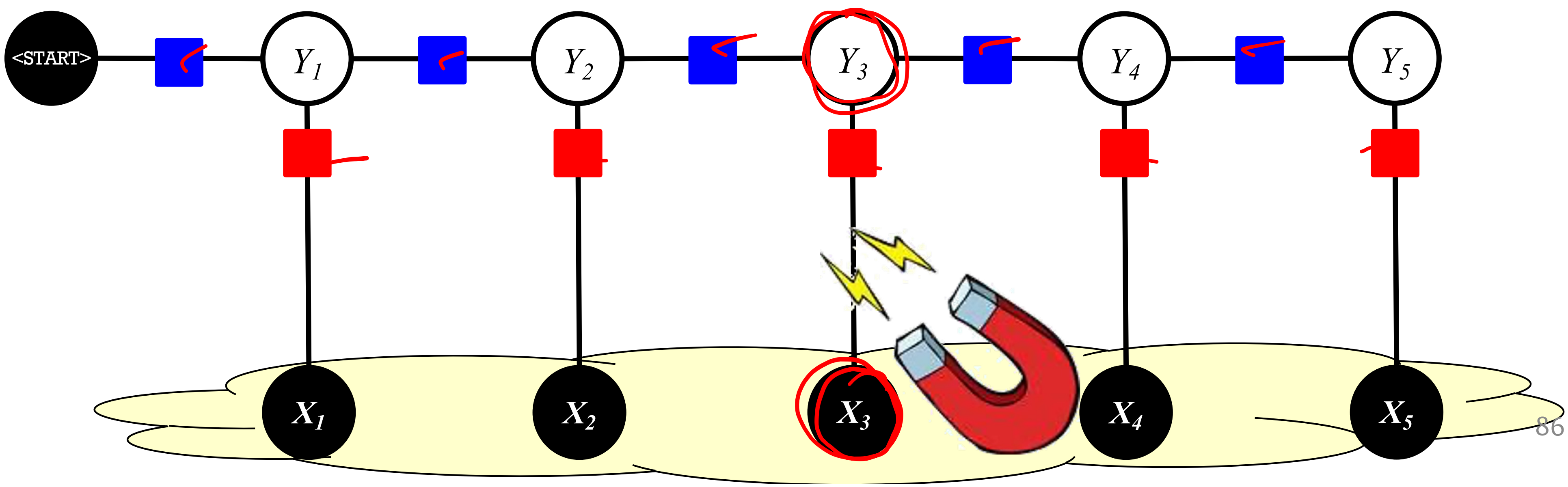


Conditional Random Field (CRF)

This **linear-chain CRF** is just **like an HMM,** except that its factors are **not** necessarily probability distributions

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k)\,\psi_{\text{tr}}(y_k, y_{k-1})$$

$$= \frac{1}{Z(\mathbf{x})}\prod_{k=1}^{K} \exp(\boldsymbol{\theta}\cdot\mathbf{f}_{\text{em}}(y_k, x_k))\exp(\boldsymbol{\theta}\cdot\mathbf{f}_{\text{tr}}(y_k, y_{k-1}))$$



$\vec{y} \in \mathcal{Y}_{\vec{x}}$ $\quad |\mathcal{Y}_x| = T^k$

$y' \in \mathcal{Y}'_{\vec{x}}$ $\quad |\mathcal{Y}'_x| = T^k$

Exercise

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k)\psi_{\text{tr}}(y_k, y_{k-1})$$

$y' = \text{string}(\vec{y})$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, x_k))\exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}))$$

$= \exp(\theta [f_{em}(\cdot) + f_{tr}(\cdot)])$ $\quad y'$

Q1

Select All That Apply: Which model does the above distribution share ~~the most~~ something in common with?

A. Hidden Markov Model 87%

B. Bernoulli Naïve Bayes 32%

C. Gaussian Naïve Bayes 53%

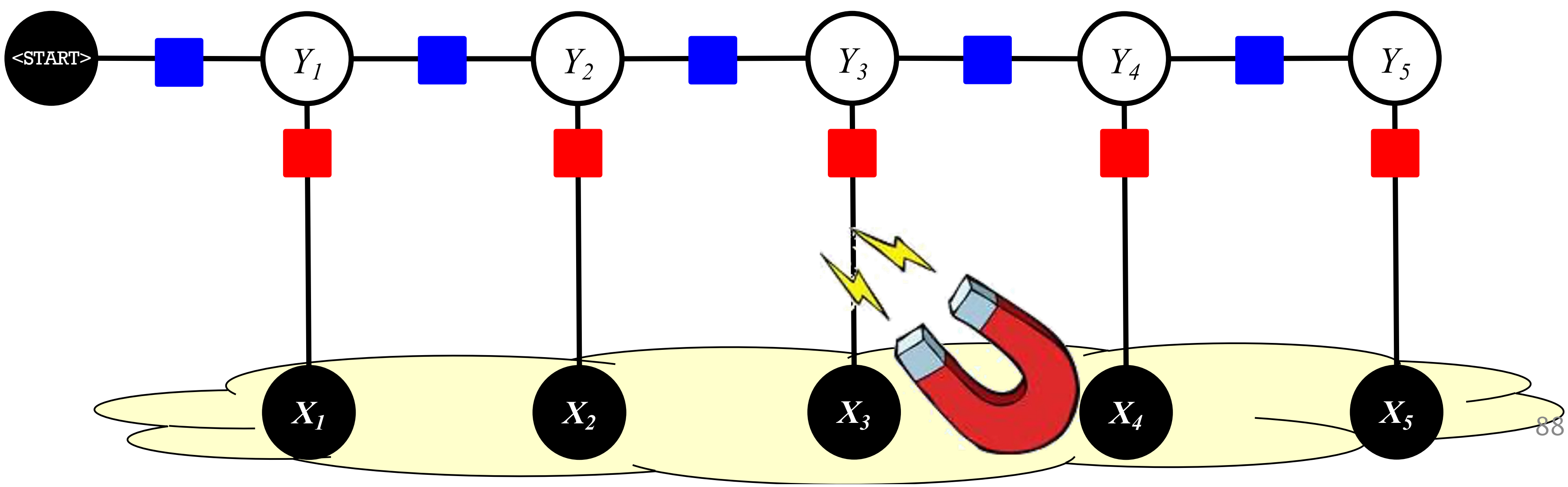
D. Logistic Regression 31%

Conditional Random Field (CRF)

This **linear-chain CRF** is just **like an HMM,** except that its factors are **not** necessarily probability distributions

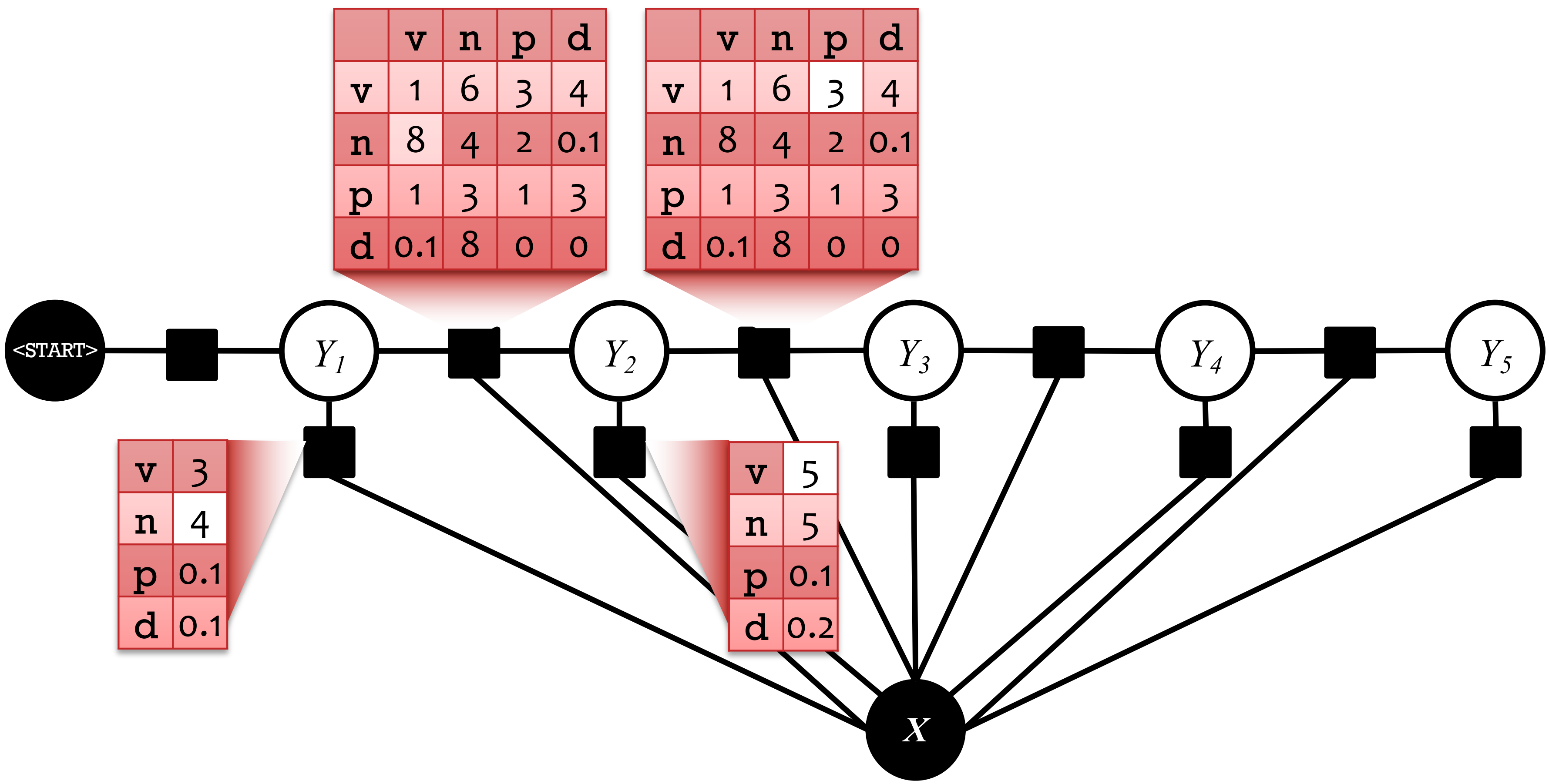
$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \textcolor{red}{\psi_{\text{em}}(y_k, x_k)} \textcolor{blue}{\psi_{\text{tr}}(y_k, y_{k-1})}$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \textcolor{red}{\exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, x_k))} \textcolor{blue}{\exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}))}$$



Conditional Random Field (CRF)

- That is the **vector** X
- Because it's observed, we can condition on it for free
- Conditioning is how we converted from the MRF to the CRF (i.e. when taking a slice of the emission factors)

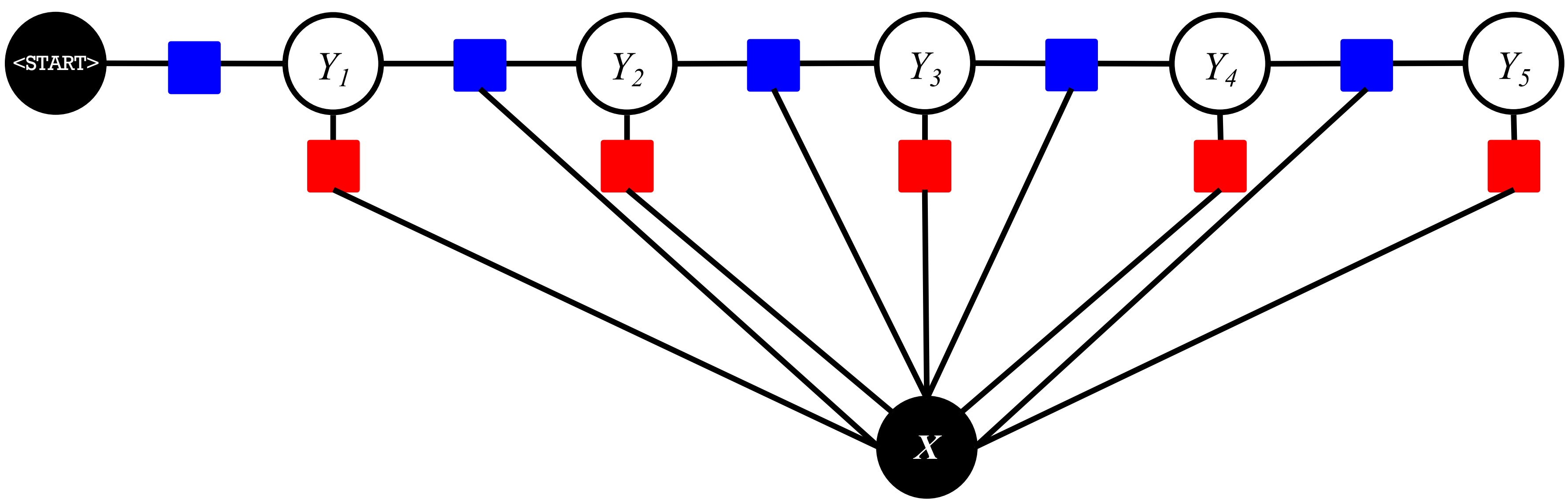


Conditional Random Field (CRF)

- This is the **standard** linear-chain CRF definition
- It permits rich, overlapping features of ~~the vector X~~

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, \mathbf{x}) \psi_{\text{tr}}(y_k, y_{k-1}, \mathbf{x})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, \mathbf{x})) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}, \mathbf{x}))$$

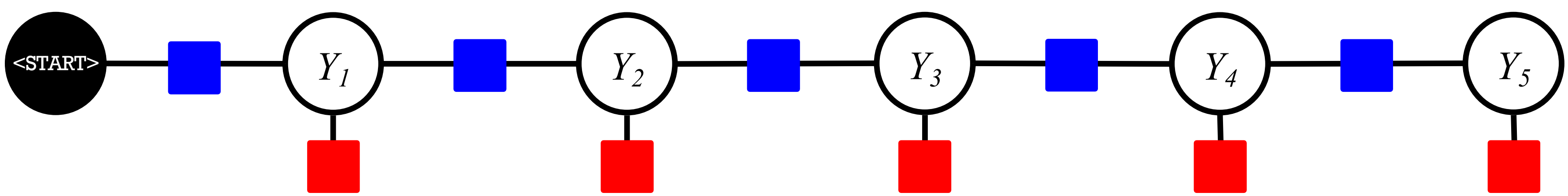


Conditional Random Field (CRF)

- This is the **standard** linear-chain CRF definition
- It permits rich, overlapping features of the vector $\boldsymbol{X}$

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, \mathbf{x}) \psi_{\text{tr}}(y_k, y_{k-1}, \mathbf{x})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, \mathbf{x})) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}, \mathbf{x}))$$



Visual Notation: Usually we draw a CRF **without** showing the variable corresponding to $\boldsymbol{X}$

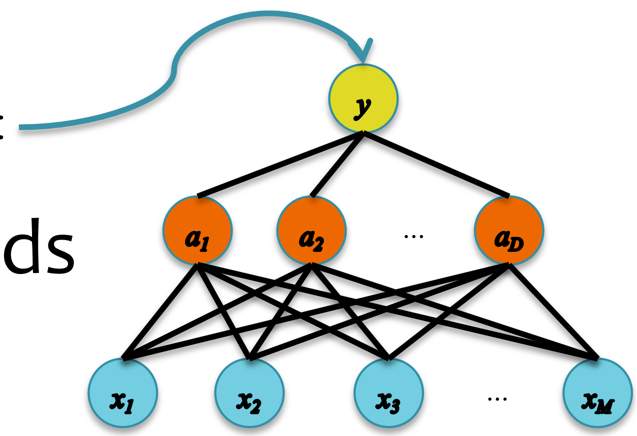
MRF AND CRF LEARNING (LOG-LINEAR PARAMETERIZATION)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



MRF and CRF Learning

Whiteboard

- log-linear MRF model (i.e. with feature based potentials)
- log-linear MRF derivatives
- log-linear MRF training with SGD
- log-linear CRF model (i.e. with feature based potentials)
- log-linear CRF derivatives
- log-linear CRF training with SGD