10-715 Advanced Intro. to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Hidden Markov Models
+
Conditional Random Fields

Matt Gormley
Guest Lecture 2
Oct. 31, 2018

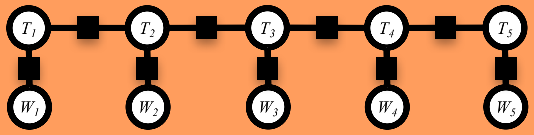
1. Data

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

Sample 1:	n time	v flies	p like	d an	n arrow
Sample 2:	n time	n flies	v like	d an	n arrow
Sample 3:	n flies	v fly	p with	n their	n wings
Sample 4:	p with	n time	n you	v will	v see

2. Model

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

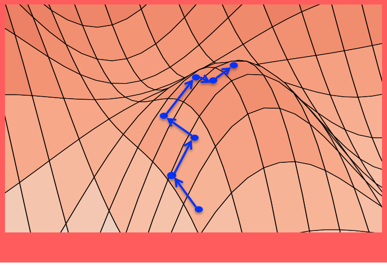


3. Objective

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

4. Learning

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



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

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

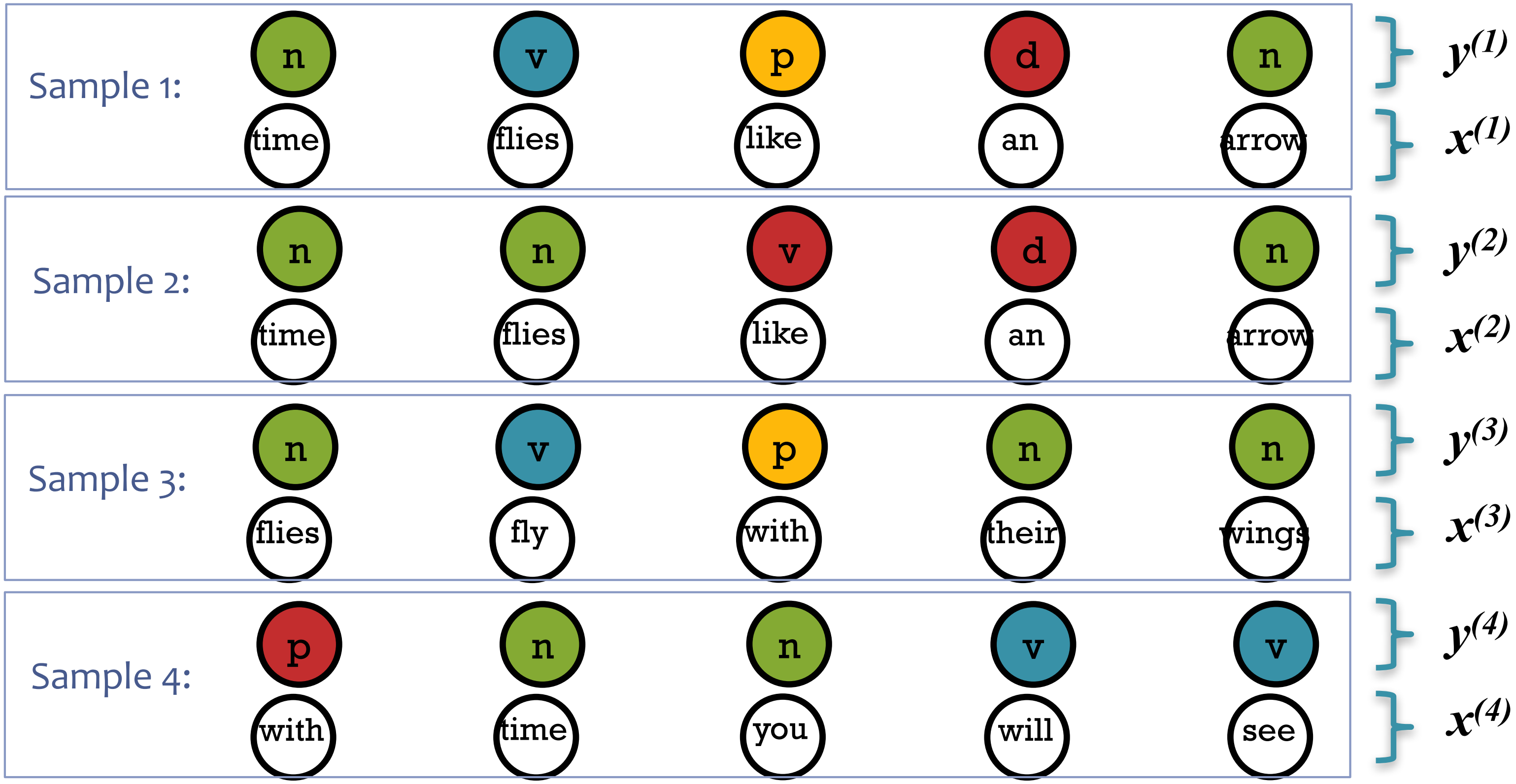
3. MAP Inference

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

HIDDEN MARKOV MODEL (HMM)

Dataset for Supervised Part-of-Speech (POS) Tagging

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

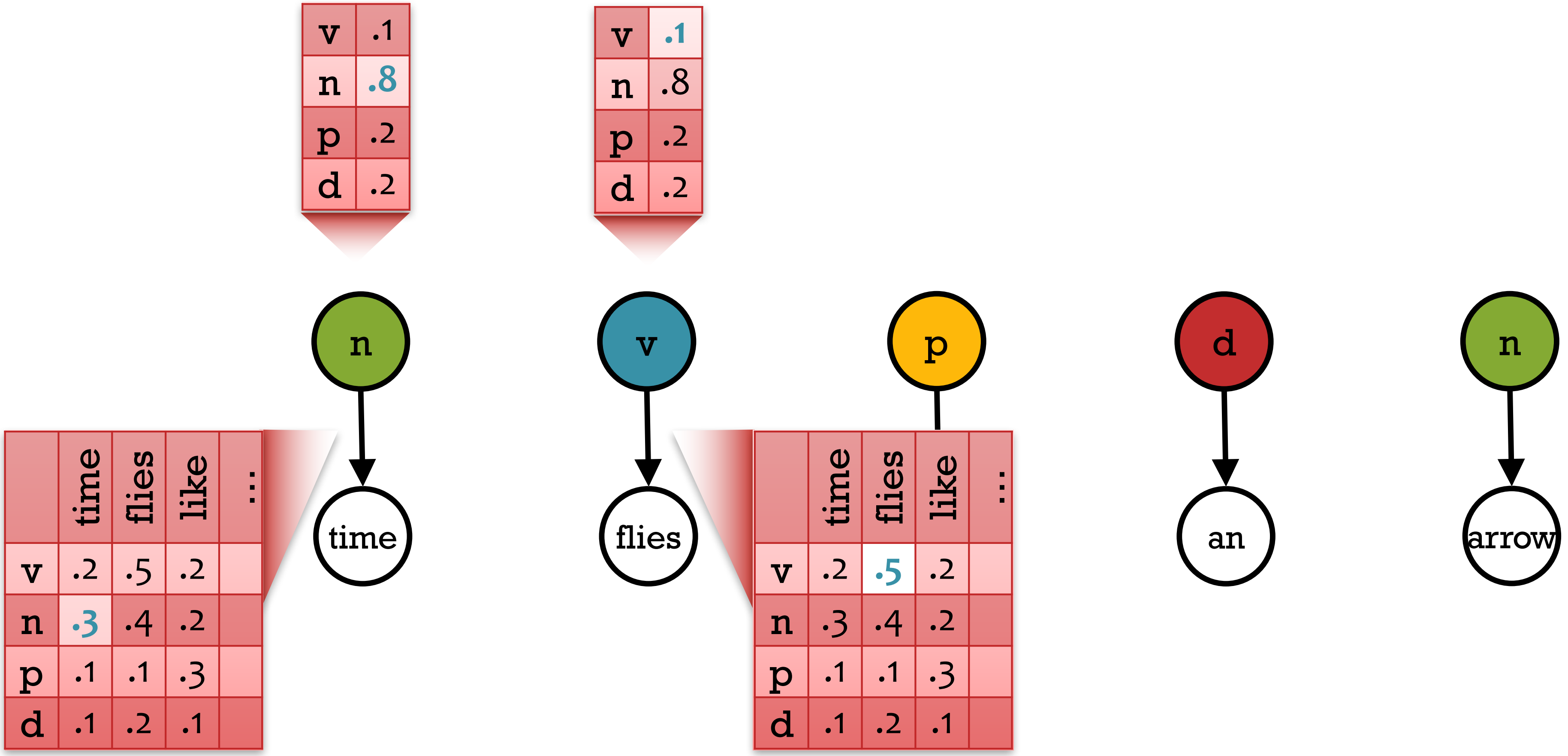


Sample 1:	n	v	p	d	n	$y^{(1)}$
	time	flies	like	an	arrow	$x^{(1)}$
Sample 2:	n	n	v	d	n	$y^{(2)}$
	time	flies	like	an	arrow	$x^{(2)}$
Sample 3:	n	v	p	n	n	$y^{(3)}$
	flies	fly	with	their	wings	$x^{(3)}$
Sample 4:	p	n	n	v	v	$y^{(4)}$
	with	time	you	will	see	$x^{(4)}$

Naïve Bayes for Time Series Data

We could treat each word-tag pair (i.e. token) as independent. This corresponds to a Naïve Bayes model with a single feature (the word).

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



v	.1
n	.8
p	.2
d	.2

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

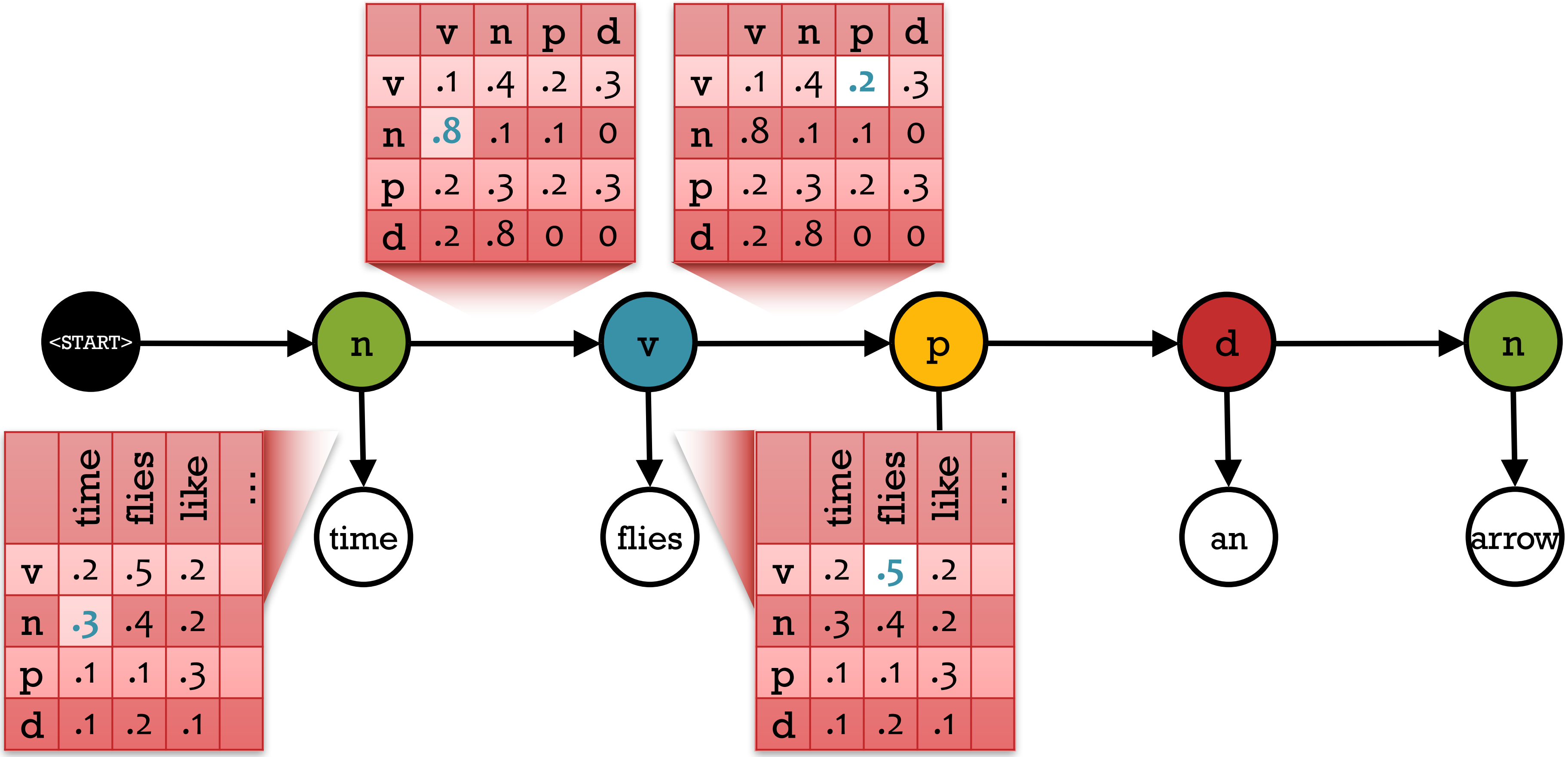
v	.1
n	.8
p	.2
d	.2

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

Hidden Markov Model

A Hidden Markov Model (HMM) provides a joint distribution over the the sentence/tags with an assumption of dependence between adjacent tags.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = (.3 * .8 * .2 * .5 * \ldots)$$



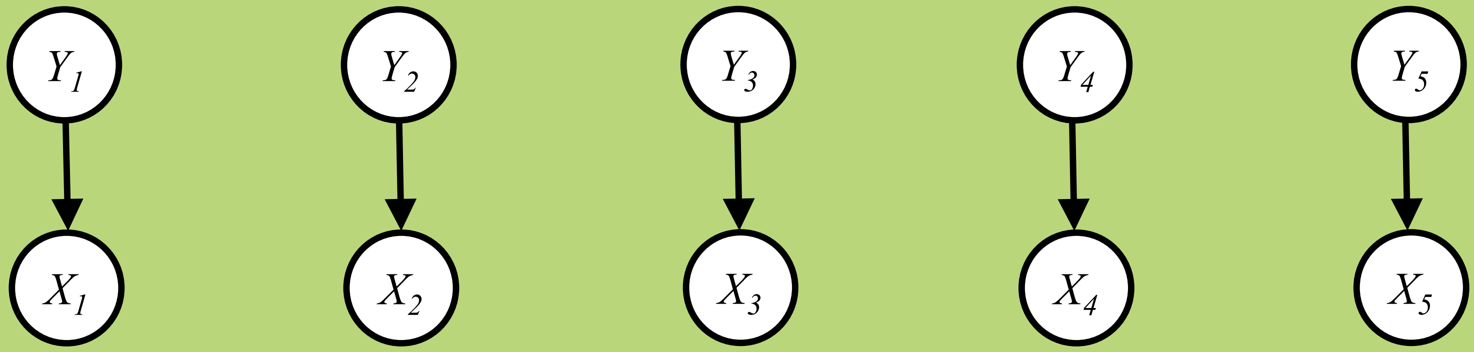
	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

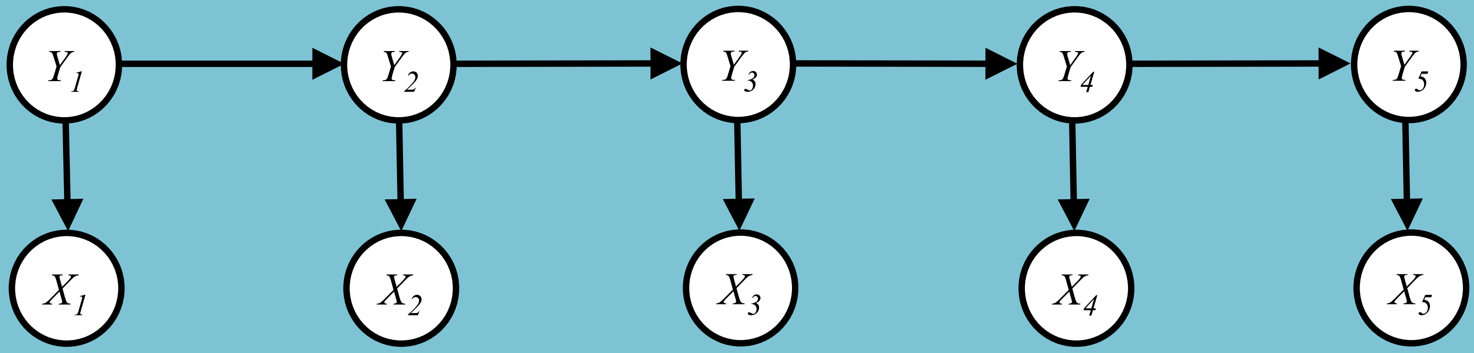
	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

From Mixture Model to HMM

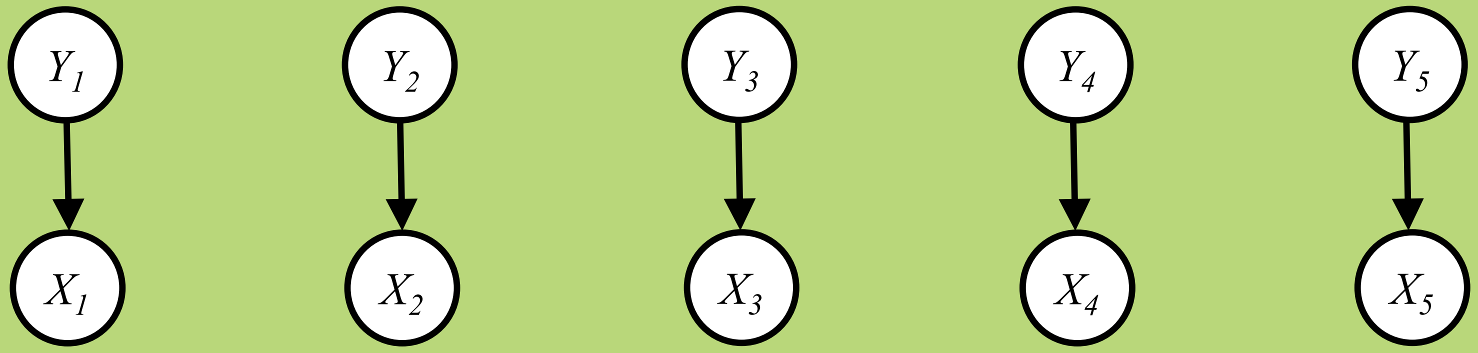


"Naïve Bayes": $$P(\mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} P(X_t|Y_t)p(Y_t)$$

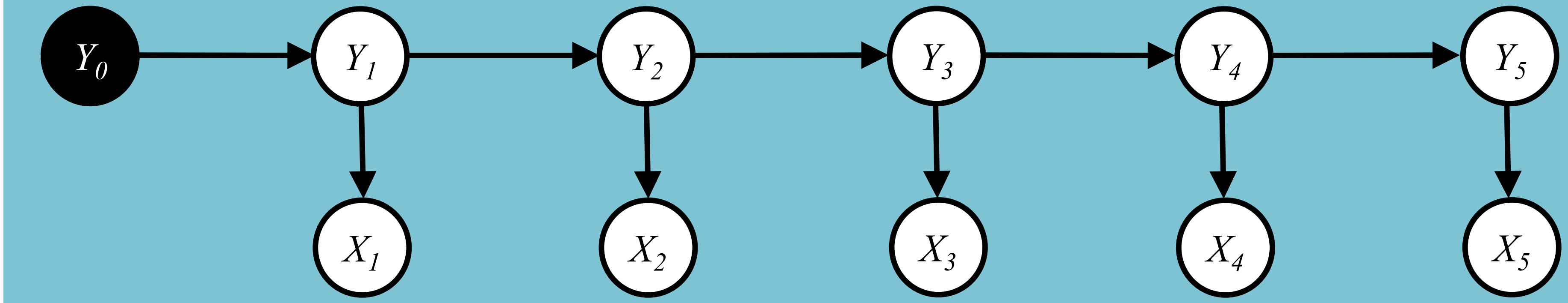


HMM: $$P(\mathbf{X}, \mathbf{Y}) = P(Y_1)\left(\prod_{t=1}^{T} P(X_t|Y_t)\right)\left(\prod_{t=2}^{T} p(Y_t|Y_{t-1})\right)$$

From Mixture Model to HMM



“Naïve Bayes”: $$P(\mathbf{X},\mathbf{Y}) = \prod_{t=1}^{T} P(X_t|Y_t)p(Y_t)$$



$$P(\mathbf{X},\mathbf{Y}|Y_0) = \prod_{t=1}^{T} P(X_t|Y_t)p(Y_t|Y_{t-1})$$

HMM:

SUPERVISED LEARNING FOR HMMS

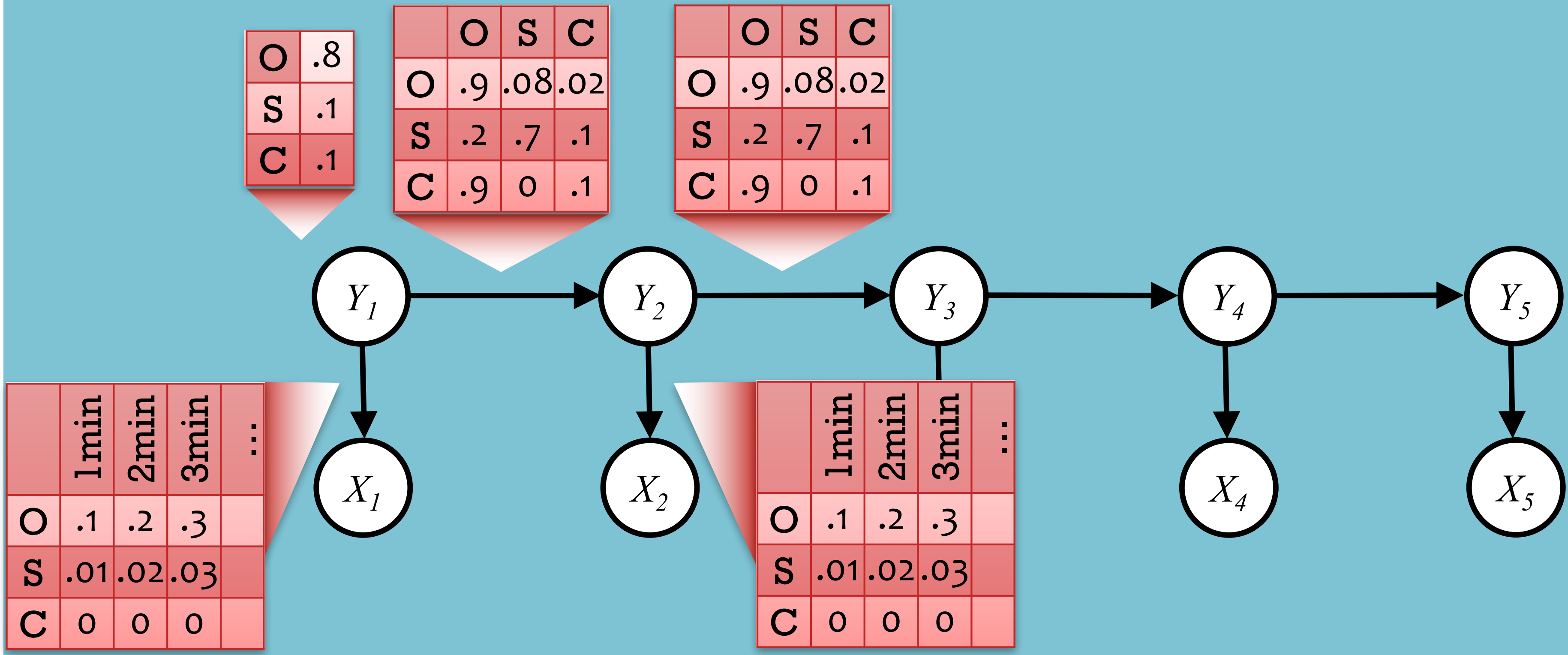
Hidden Markov Model

HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_t = k | Y_t = j) = A_{j,k}, \forall t, k$

Transition matrix, $\mathbf{B}$, where $P(Y_t = k | Y_{t-1} = j) = B_{j,k}, \forall t, k$

Initial probs, $\mathbf{C}$, where $P(Y_1 = k) = C_k, \forall k$



O	.8
S	.1
C	.1

	O	S	C
O	.9	.08	.02
S	.2	.7	.1
C	.9	0	.1

	O	S	C
O	.9	.08	.02
S	.2	.7	.1
C	.9	0	.1

	1min	2min	3min	...
O	.1	.2	.3	
S	.01	.02	.03	
C	0	0	0	

	1min	2min	3min	...
O	.1	.2	.3	
S	.01	.02	.03	
C	0	0	0	

Hidden Markov Model

HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_t = k | Y_t = j) = A_{j,k}, \forall t, k$

Transition matrix, $\mathbf{B}$, where $P(Y_t = k | Y_{t-1} = j) = B_{j,k}, \forall t, k$

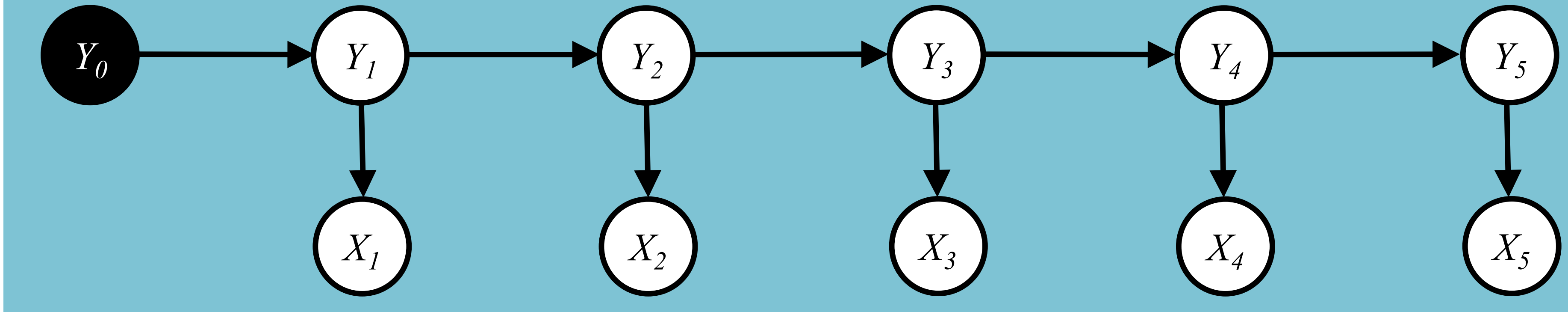
Assumption: $y_0 = \text{START}$

Generative Story:

$$Y_t \sim \text{Multinomial}(\mathbf{B}_{Y_{t-1}}) \ \forall t$$

$$X_t \sim \text{Multinomial}(\mathbf{A}_{Y_t}) \ \forall t$$

For notational convenience, we fold the *initial probabilities* **C** into the *transition matrix* **B** by our assumption.



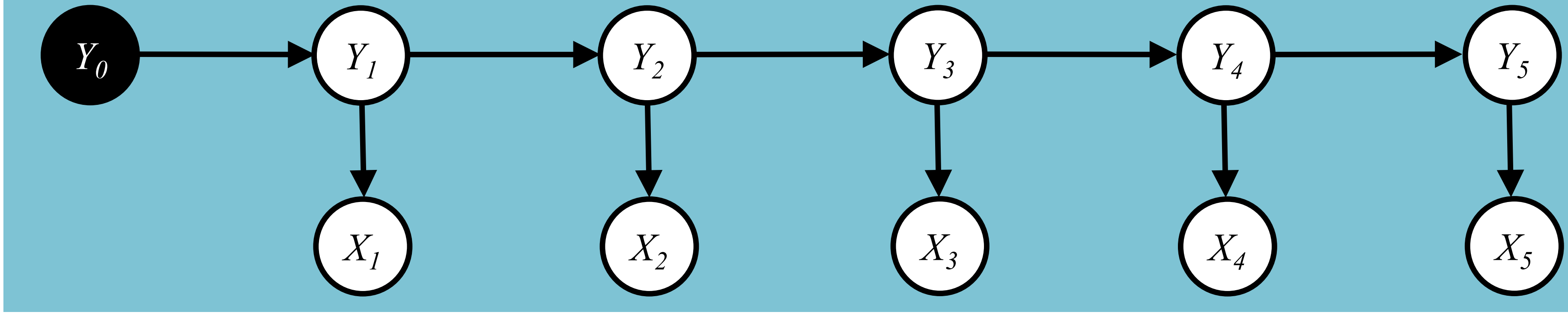
Hidden Markov Model

Joint Distribution:

$$y_0 = \text{START}$$

$$p(\mathbf{x}, \mathbf{y} | y_0) = \prod_{t=1}^{T} p(x_t | y_t) p(y_t | y_{t-1})$$

$$= \prod_{t=1}^{T} A_{y_t, x_t} B_{y_{t-1}, y_t}$$



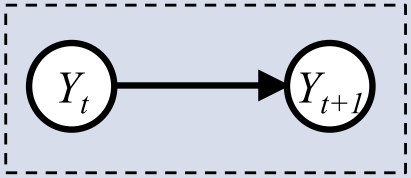
Training HMMs

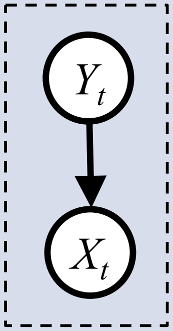
Whiteboard

- (Supervised) Likelihood for an HMM
- Maximum Likelihood Estimation (MLE) for HMM

Supervised Learning for HMMs

Learning an HMM decomposes into solving two (independent) Mixture Models





Data: $\mathcal{D} = \{(\vec{x}^{(i)}, \vec{y}^{(i)})\}_{i=1}^{N}$ $\quad \vec{x} = [x_1, \dots, x_T]^T$

$\vec{y} = [y_1, \dots, y_T]^T$

Likelihood:

$$\ell(A,B,C) = \sum_{i=1}^{N} \log p(\vec{x}^{(i)}, y^{(i)} | A,B,C)$$

$$= \sum_{i=1}^{N} \left[\underbrace{\log p(y_1^{(i)} | C)}_{\text{initial}} + \underbrace{\left(\sum_{t=2}^{T} \log p(y_t^{(i)} | y_{t-1}^{(i)}, B) \right)}_{\text{transition}} + \underbrace{\left(\sum_{t=1}^{T} \log p(x_t^{(i)} | y_t^{(i)}, A) \right)}_{\text{emission}} \right]$$

MLE:

$$\hat{A}, \hat{B}, \hat{C} = \underset{A,B,C}{\text{argmax}} \; \ell(A,B,C)$$

$$\Rightarrow \left. \begin{aligned} \hat{C} &= \underset{C}{\text{argmax}} \sum_{i=1}^{N} \log p(y_1^{(i)} | C) \\ \hat{B} &= \underset{B}{\text{argmax}} \sum_{i=1}^{N} \sum_{t=2}^{T} \log p(y_t^{(i)} | y_{t-1}^{(i)}, B) \\ \hat{A} &= \underset{A}{\text{argmax}} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p(x_t^{(i)} | y_t^{(i)}, A) \end{aligned} \right\} \text{Can solve in closed form, which yields...}$$

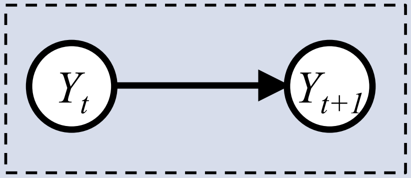
$$\hat{C}_k = \frac{\#(y_1^{(i)} = k)}{N} \qquad \forall i,k$$

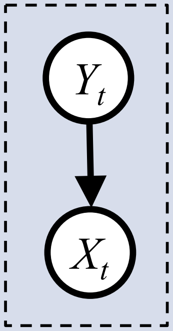
$$\hat{B}_{jk} = \frac{\#(y_t^{(i)} = k \text{ and } y_{t-1}^{(i)} = j)}{\#(y_{t-1}^{(i)} = j)} \qquad \forall i, t>1, j, k$$

$$\hat{A}_{jk} = \frac{\#(x_t^{(i)} = k \text{ and } y_t^{(i)} = j)}{\#(y_t^{(i)} = j)} \qquad \forall i, t, j, k$$

Supervised Learning for HMMs

Learning an HMM decomposes into solving two (independent) Mixture Models





$$D = \{(\vec{x}^{(i)}, \vec{y}^{(i)})\}_{i=1}^{N}$$

Likelihood:

$$\ell(A,B) = \sum_{i=1}^{N} \log p(\vec{x}^{(i)}, \vec{y}^{(i)})$$

$$= \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \log p(y_t^{(i)} | y_{t-1}^{(i)}, B) + \log p(x_t^{(i)} | y_t^{(i)}, A) \right]$$

MLE:

$$\hat{A}, \hat{B} = \operatorname{argmax} \ \ell(A,B)$$

$$\hat{A} = \operatorname{argmax} \ \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \log p(x_t^{(i)} | y_t^{(i)}, A) \right]$$

$$\hat{B} = \operatorname{argmax} \ \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \log p(y_t^{(i)} | y_{t-1}^{(i)}, B) \right]$$

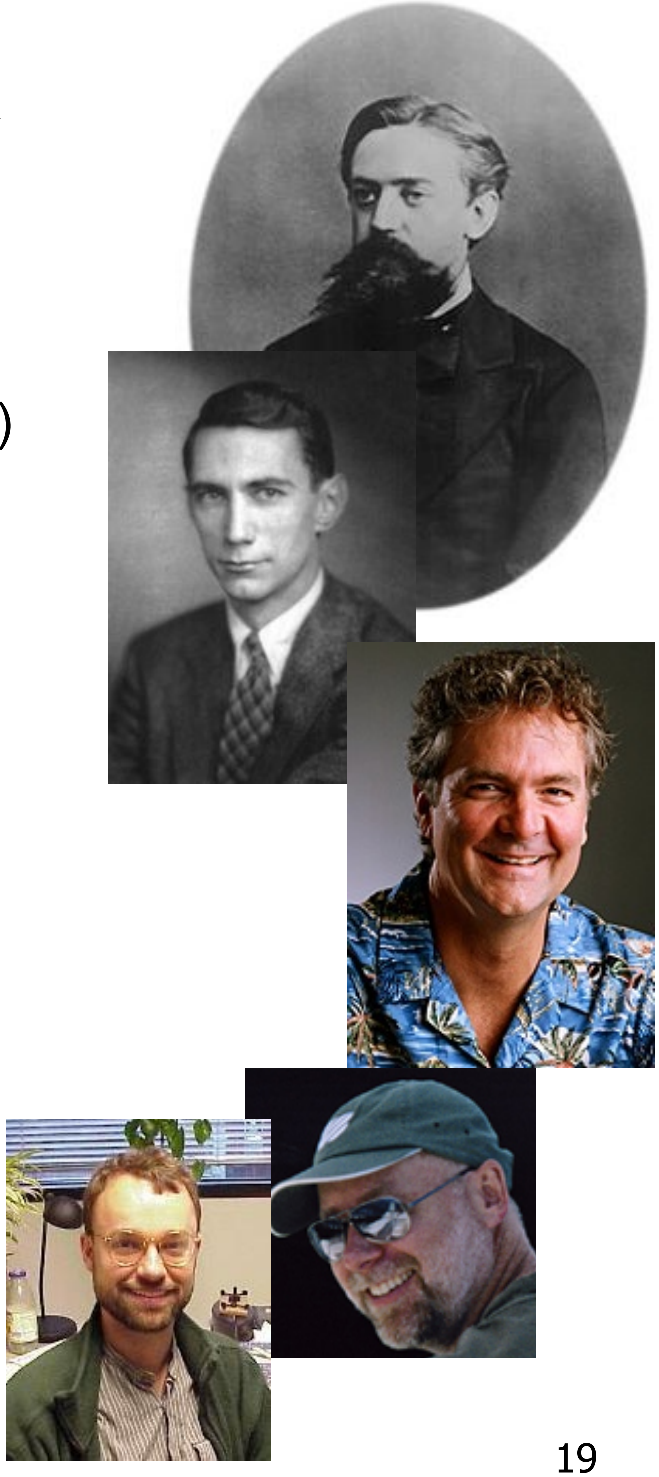
↑ can solve in closed form to get...

$$\hat{B}_{jk} = \frac{\#(y_t^{(i)} = k \text{ and } y_{t-1}^{(i)} = j)}{\#(y_{t-1}^{(i)} = j)}$$

$$\hat{A}_{jk} = \frac{\#(x_t^{(i)} = k \text{ and } y_t^{(i)} = j)}{\#(y_t^{(i)} = j)}$$

HMMs: History

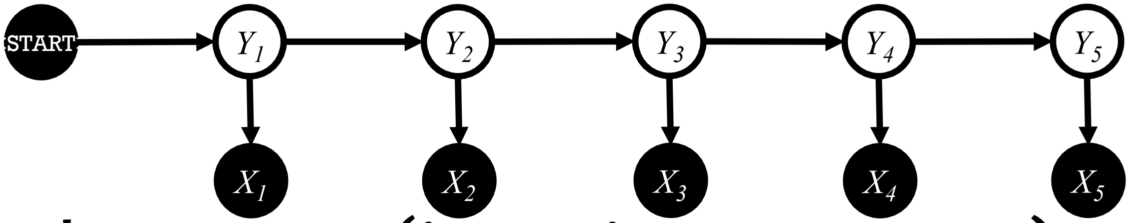
- Markov chains: Andrey Markov (1906)
  - Random walks and Brownian motion
- Used in Shannon’s work on information theory (1948)
- Baum-Welsh learning algorithm: late 60’s, early 70’s.
  - Used mainly for speech in 60s-70s.
- Late 80’s and 90’s: David Haussler (major player in learning theory in 80’s) began to use HMMs for modeling biological sequences
- Mid-late 1990’s: Dayne Freitag/Andrew McCallum
  - Freitag thesis with Tom Mitchell on IE from Web using logic programs, grammar induction, etc.
  - McCallum: multinomial Naïve Bayes for text
  - With McCallum, IE using HMMs on CORA
- …



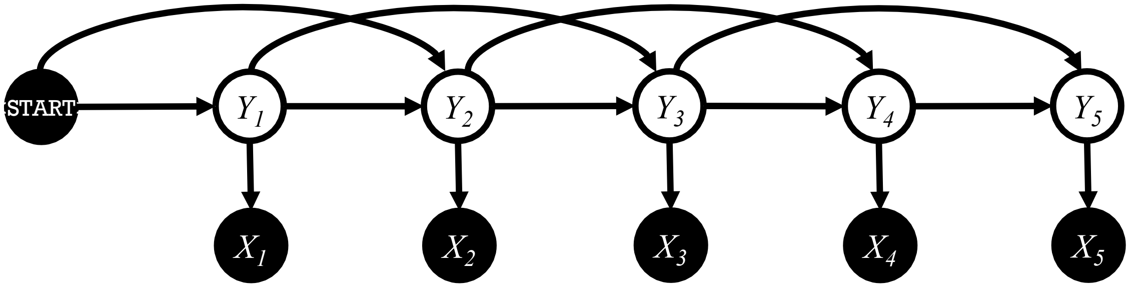
Slide from William Cohen

Higher-order HMMs

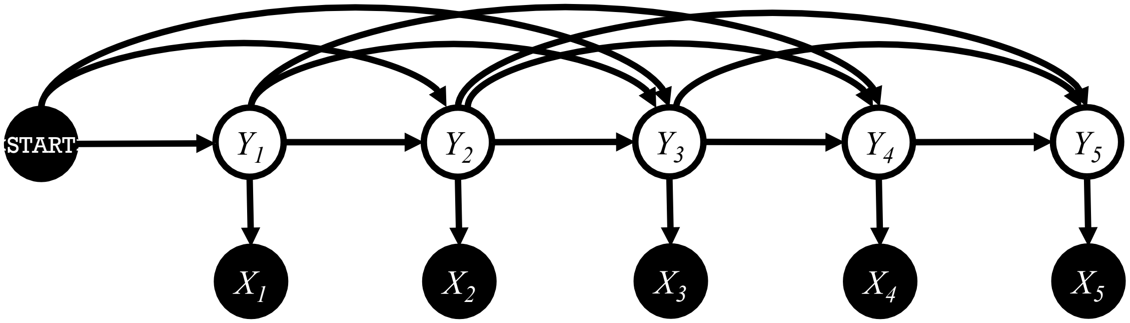
- 1st-order HMM (i.e. bigram HMM)



- 2nd-order HMM (i.e. trigram HMM)



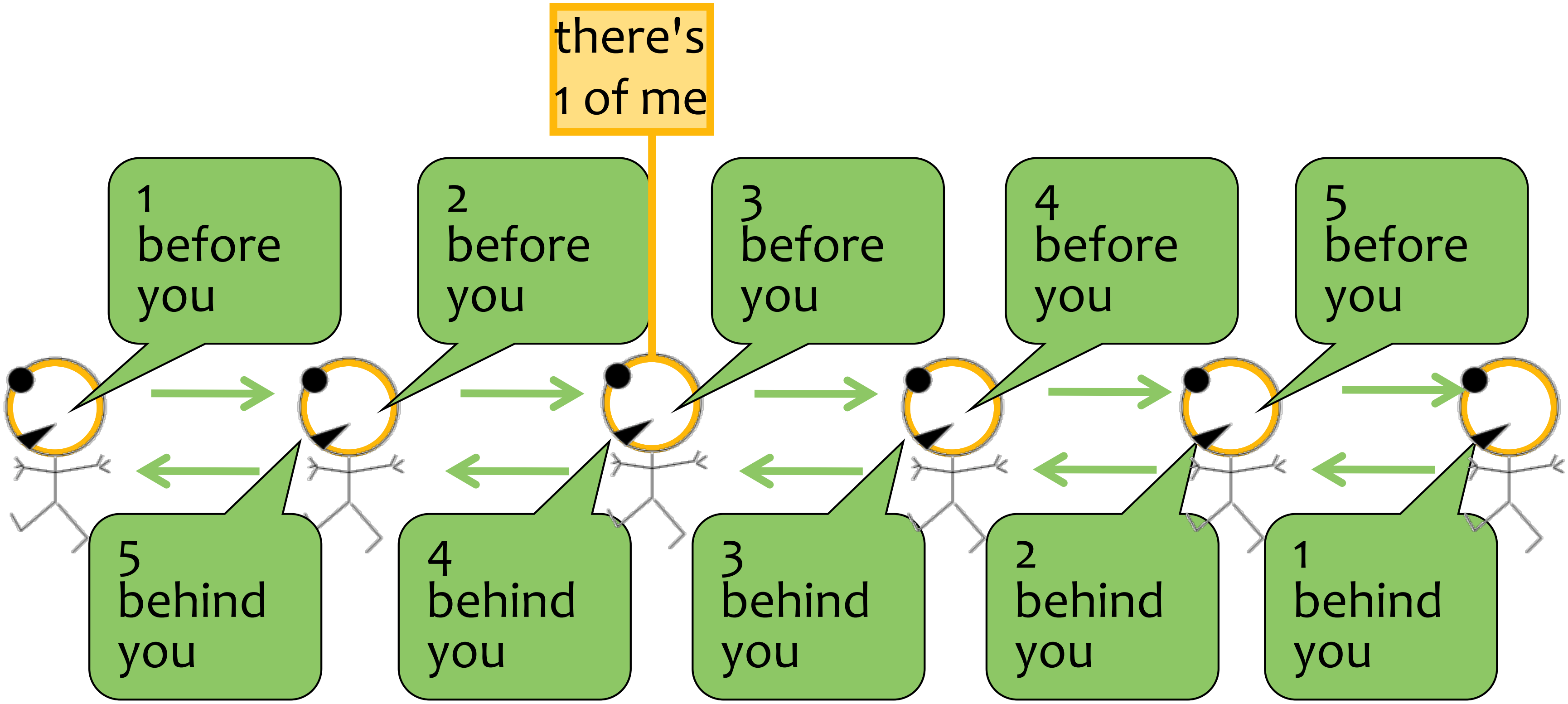
- 3rd-order HMM



BACKGROUND: MESSAGE PASSING

Great Ideas in ML: Message Passing

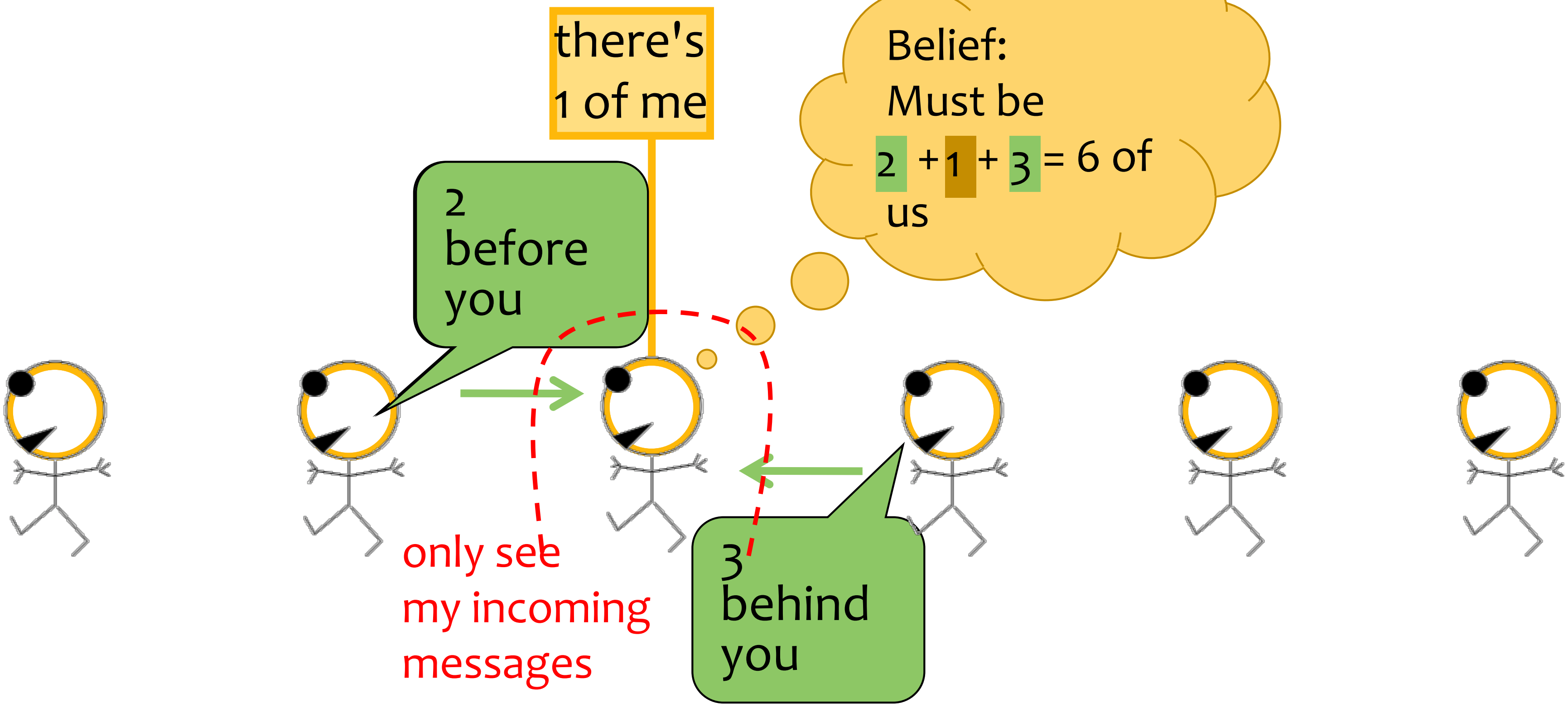
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

Count the soldiers



there's 1 of me

2 before you

3 behind you

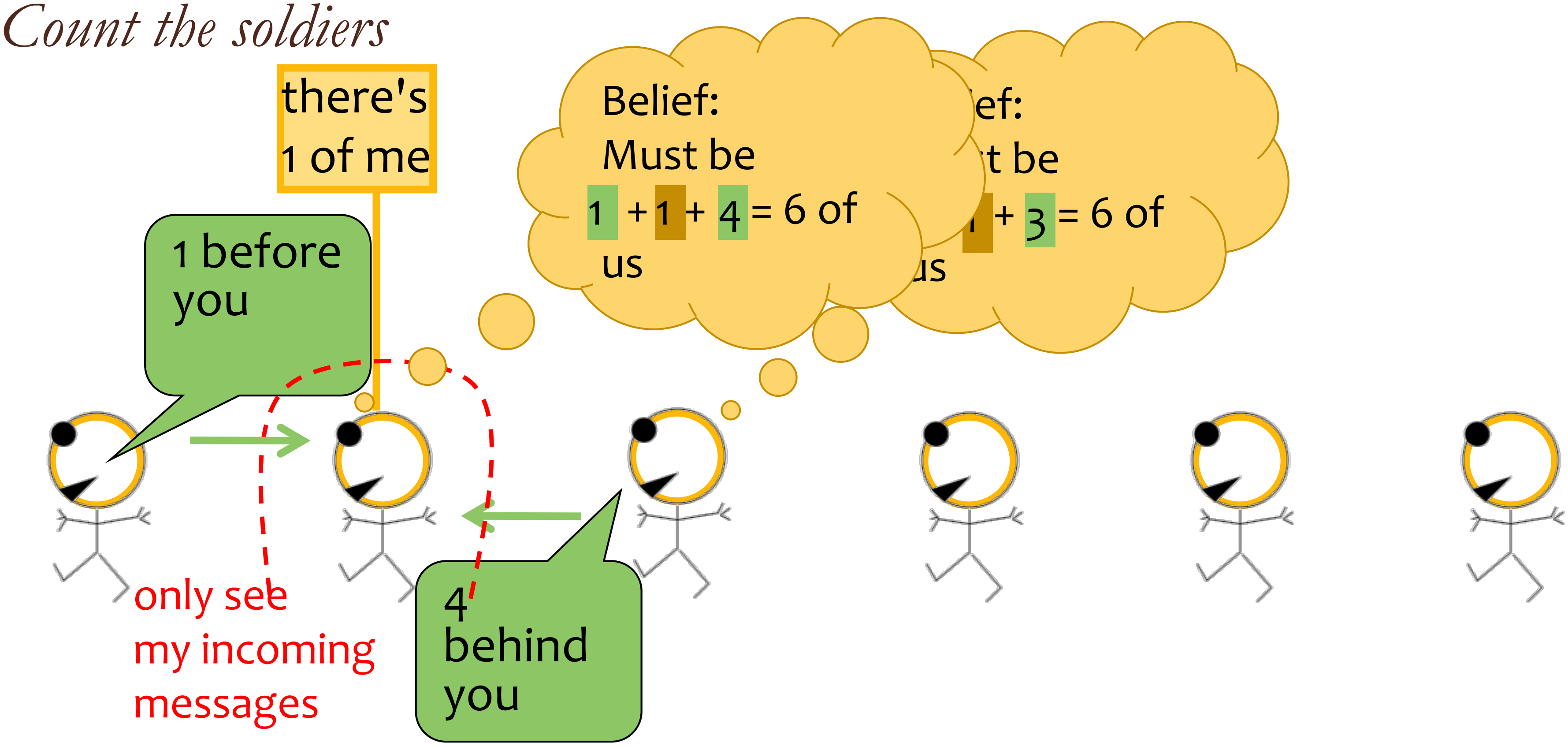
Belief: Must be 2 + 1 + 3 = 6 of us

only see my incoming messages

adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

Count the soldiers



there's 1 of me

1 before you

Belief: Must be 1 + 1 + 4 = 6 of us

ef: t be 1 + 3 = 6 of us

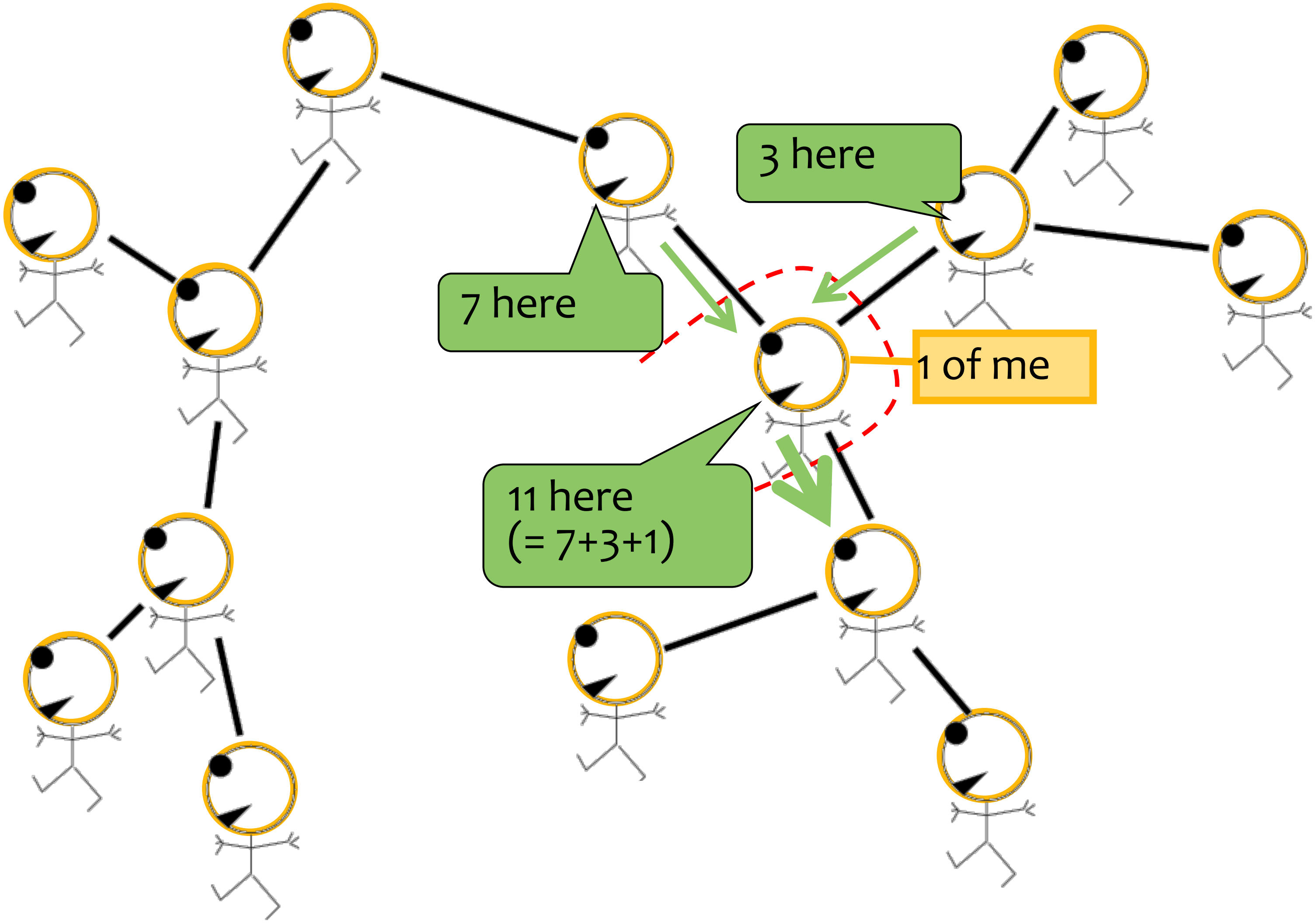
4 behind you

only see my incoming messages

adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

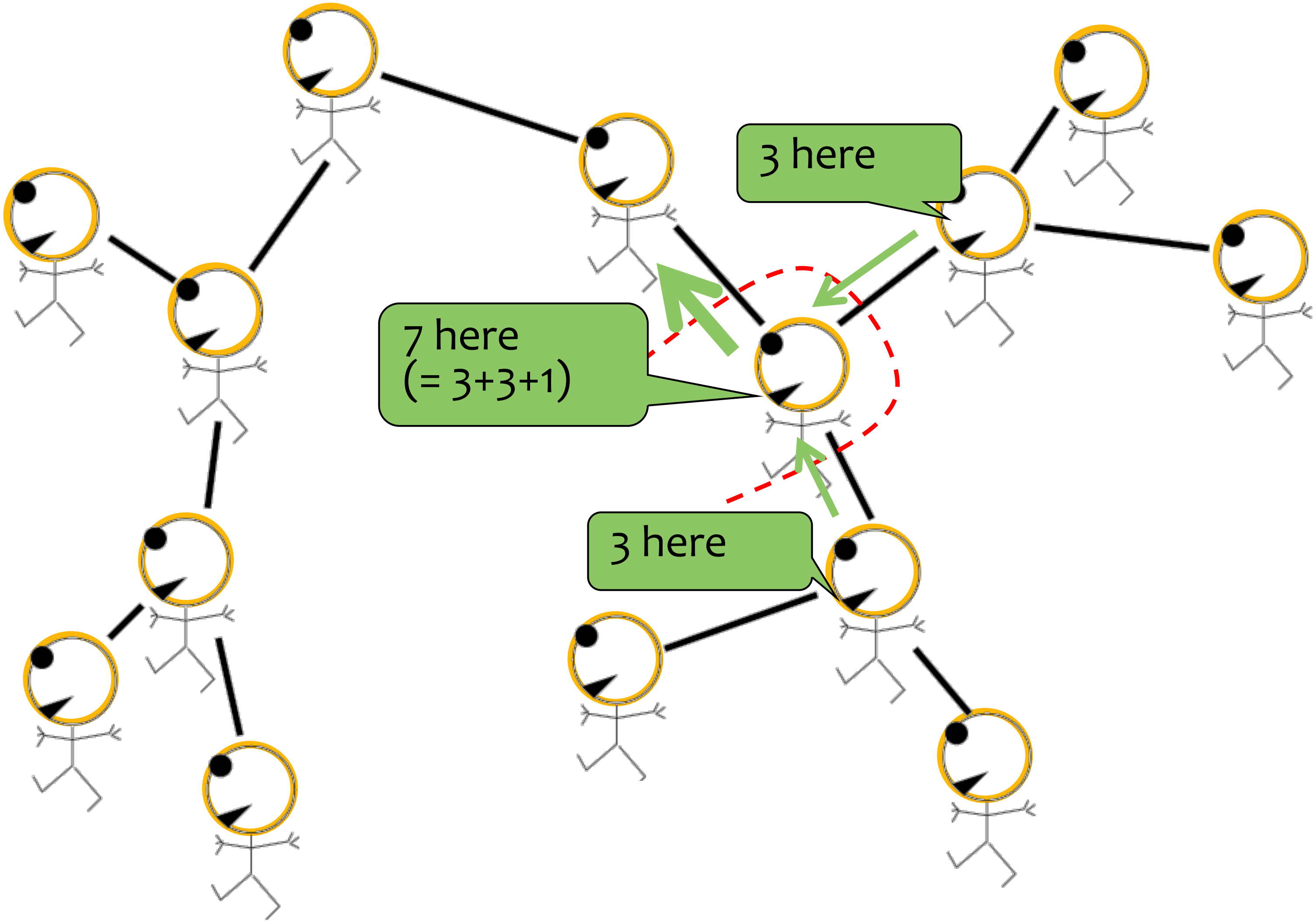
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

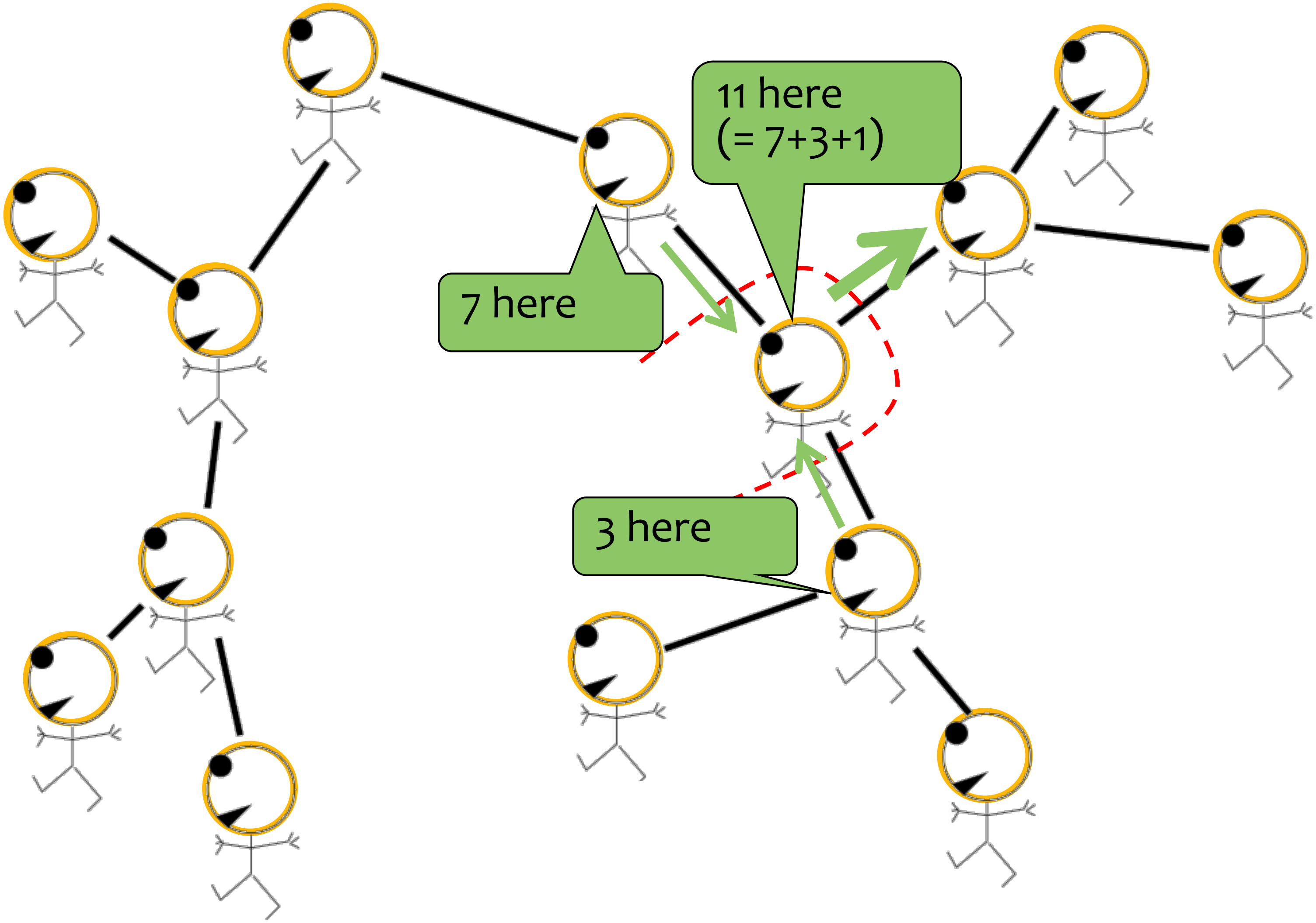
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

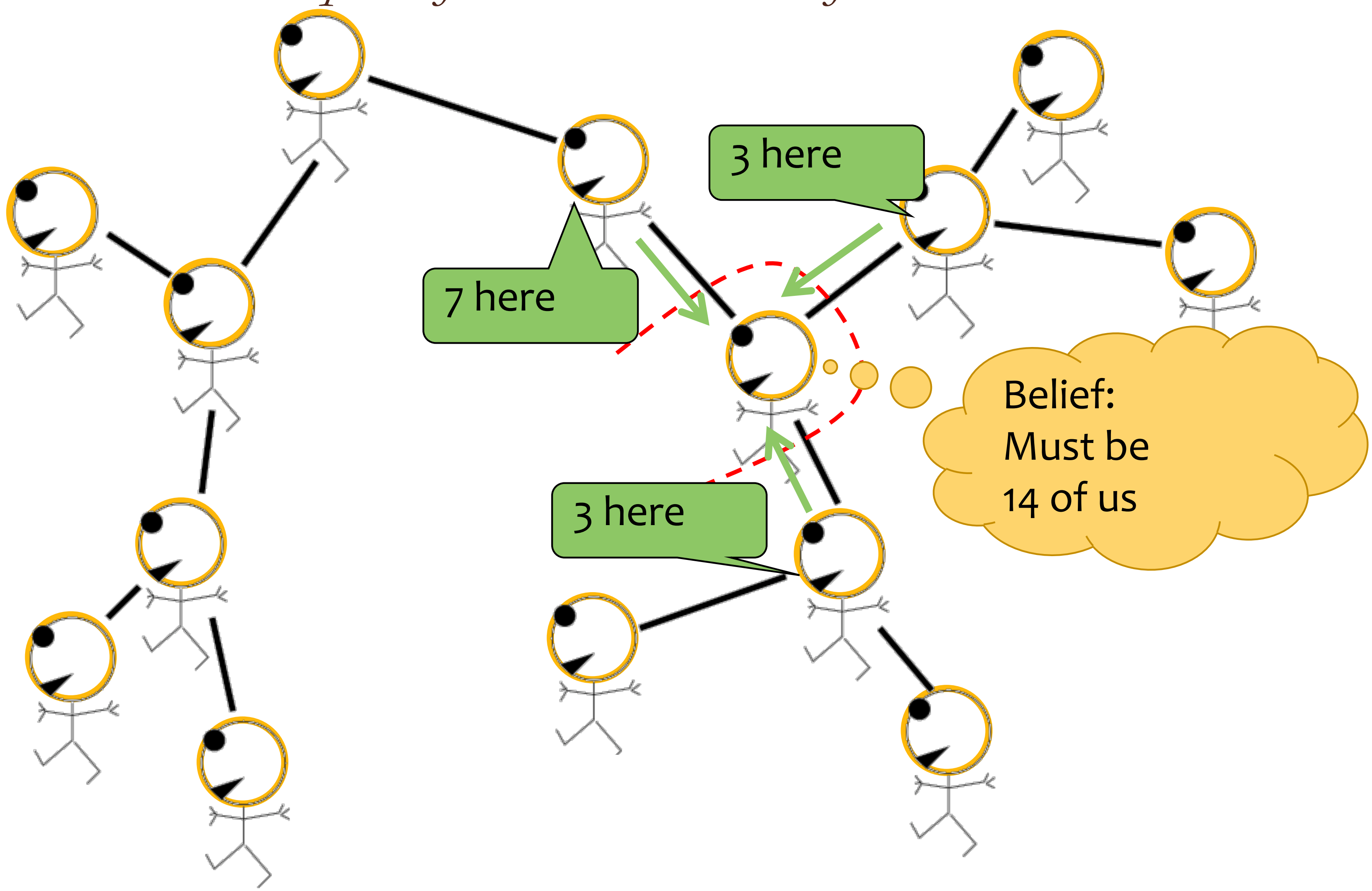
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

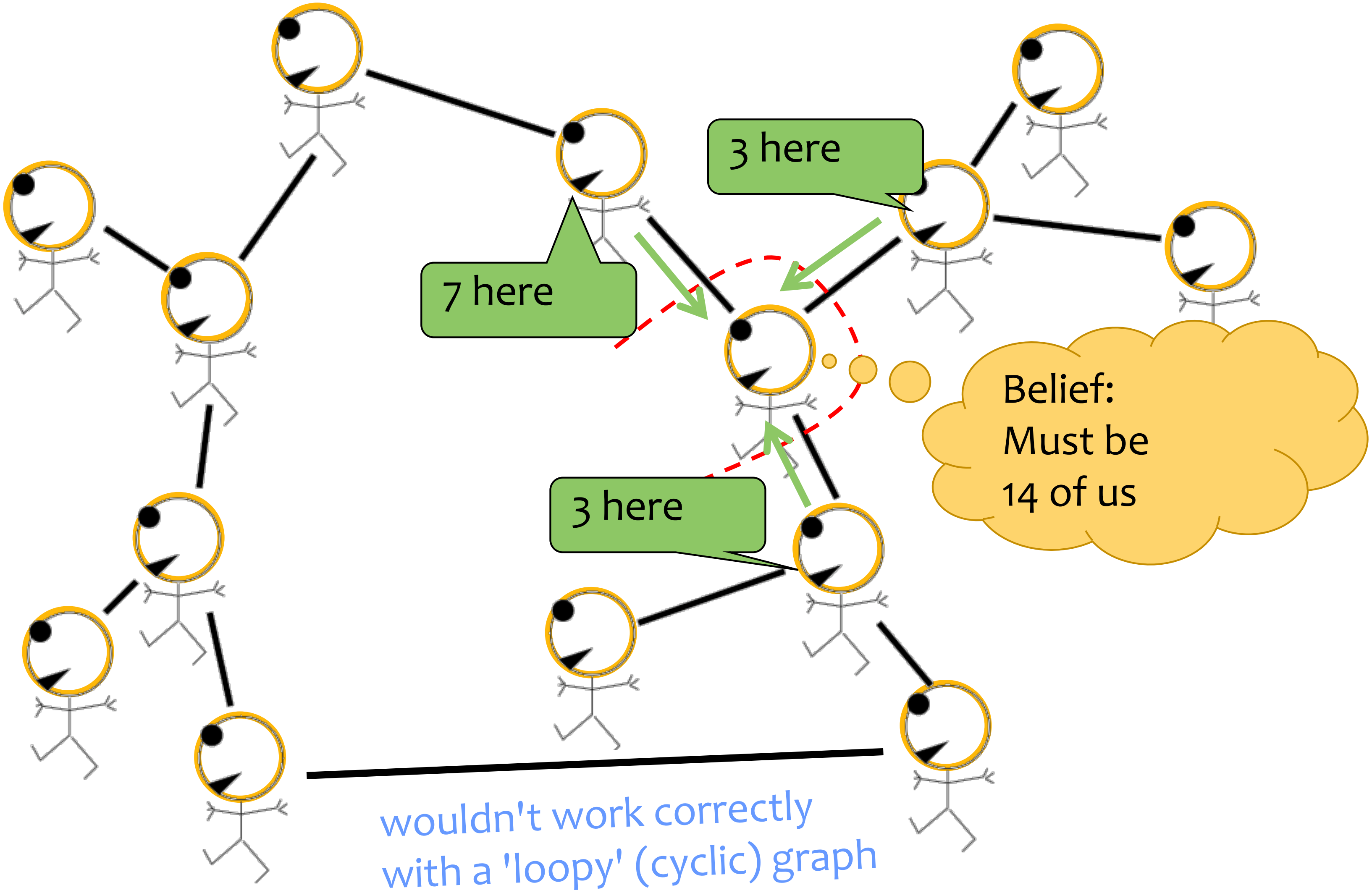
Each soldier receives reports from all branches of tree



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

THE FORWARD-BACKWARD ALGORITHM

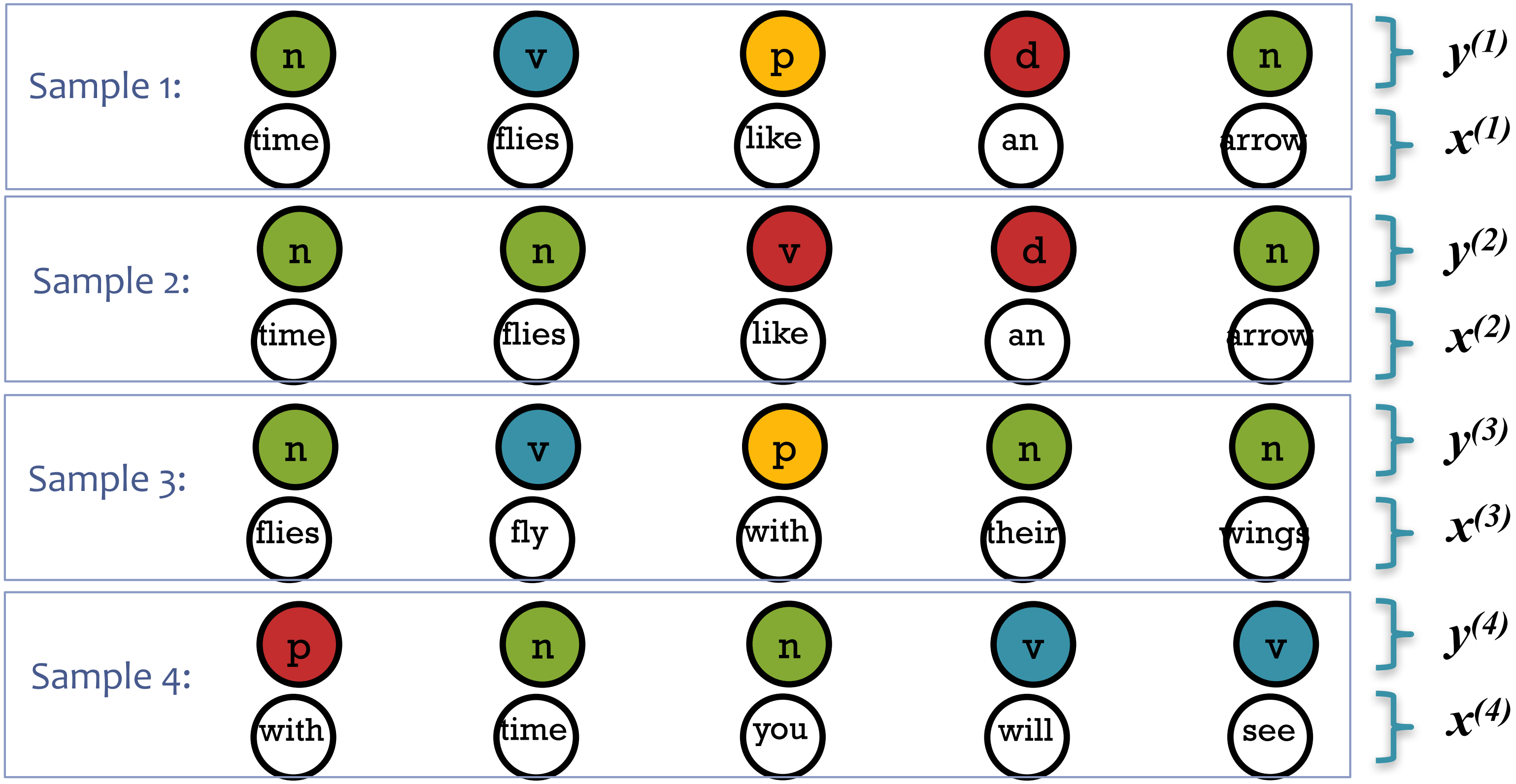
Inference for HMMs

Whiteboard

- Three Inference Problems for an HMM
  1. Evaluation: Compute the probability of a given sequence of observations
  2. Decoding: Find the most-likely sequence of hidden states, given a sequence of observations
  3. Marginals: Compute the marginal distribution for a hidden state, given a sequence of observations

Dataset for Supervised Part-of-Speech (POS) Tagging

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

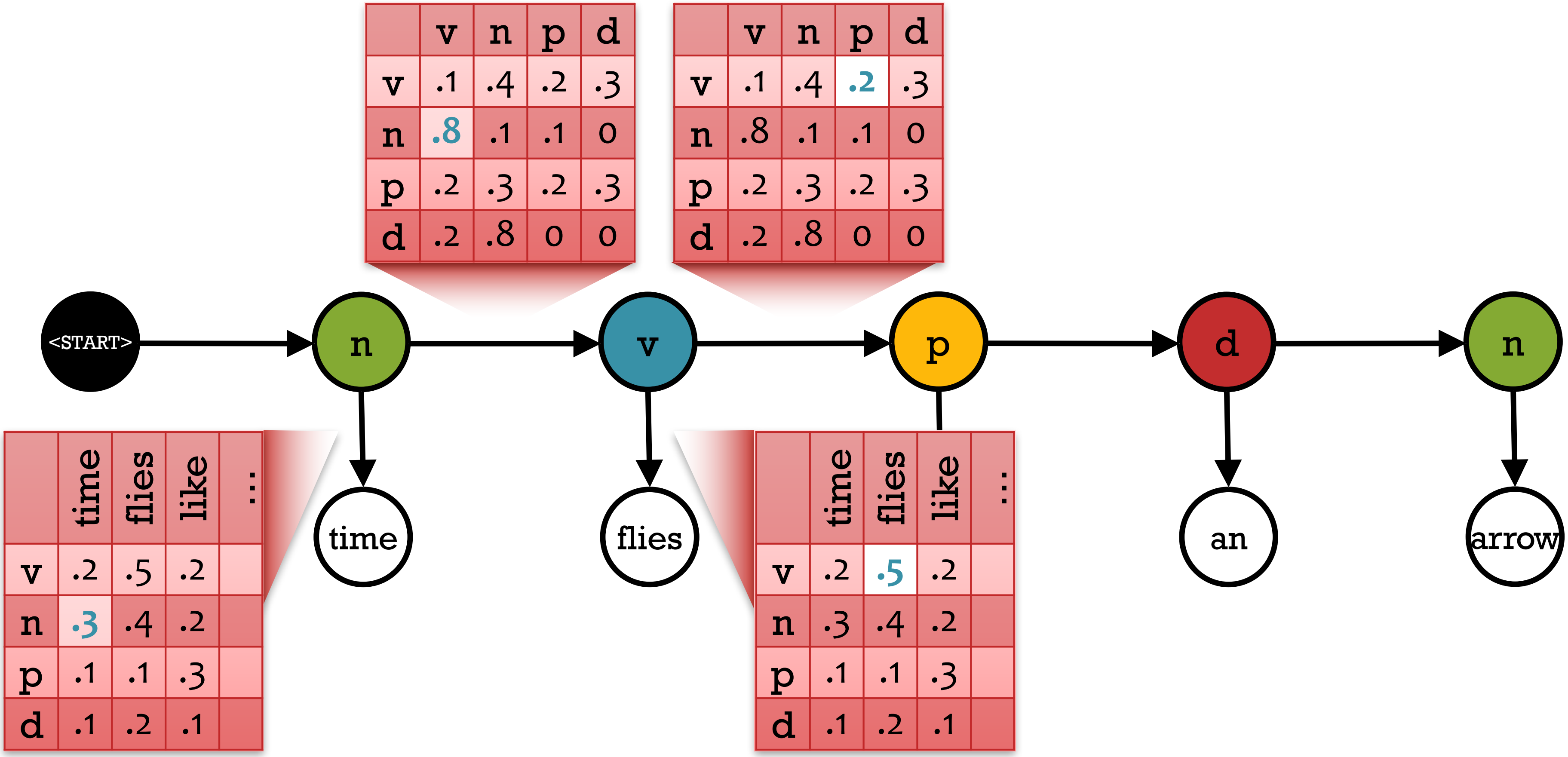


Sample 1:	n	v	p	d	n	$y^{(1)}$
	time	flies	like	an	arrow	$x^{(1)}$
Sample 2:	n	n	v	d	n	$y^{(2)}$
	time	flies	like	an	arrow	$x^{(2)}$
Sample 3:	n	v	p	n	n	$y^{(3)}$
	flies	fly	with	their	wings	$x^{(3)}$
Sample 4:	p	n	n	v	v	$y^{(4)}$
	with	time	you	will	see	$x^{(4)}$

Hidden Markov Model

A Hidden Markov Model (HMM) provides a joint distribution over the the sentence/tags with an assumption of dependence between adjacent tags.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = (.3 * .8 * .2 * .5 * \ldots)$$



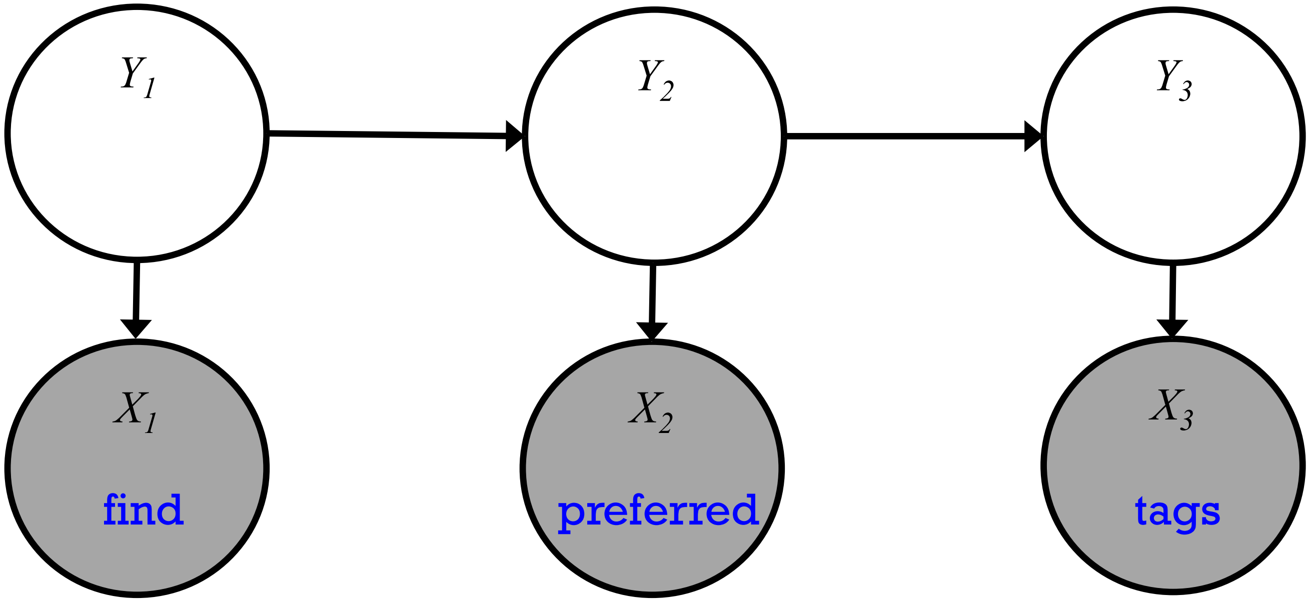
	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

Forward-Backward Algorithm

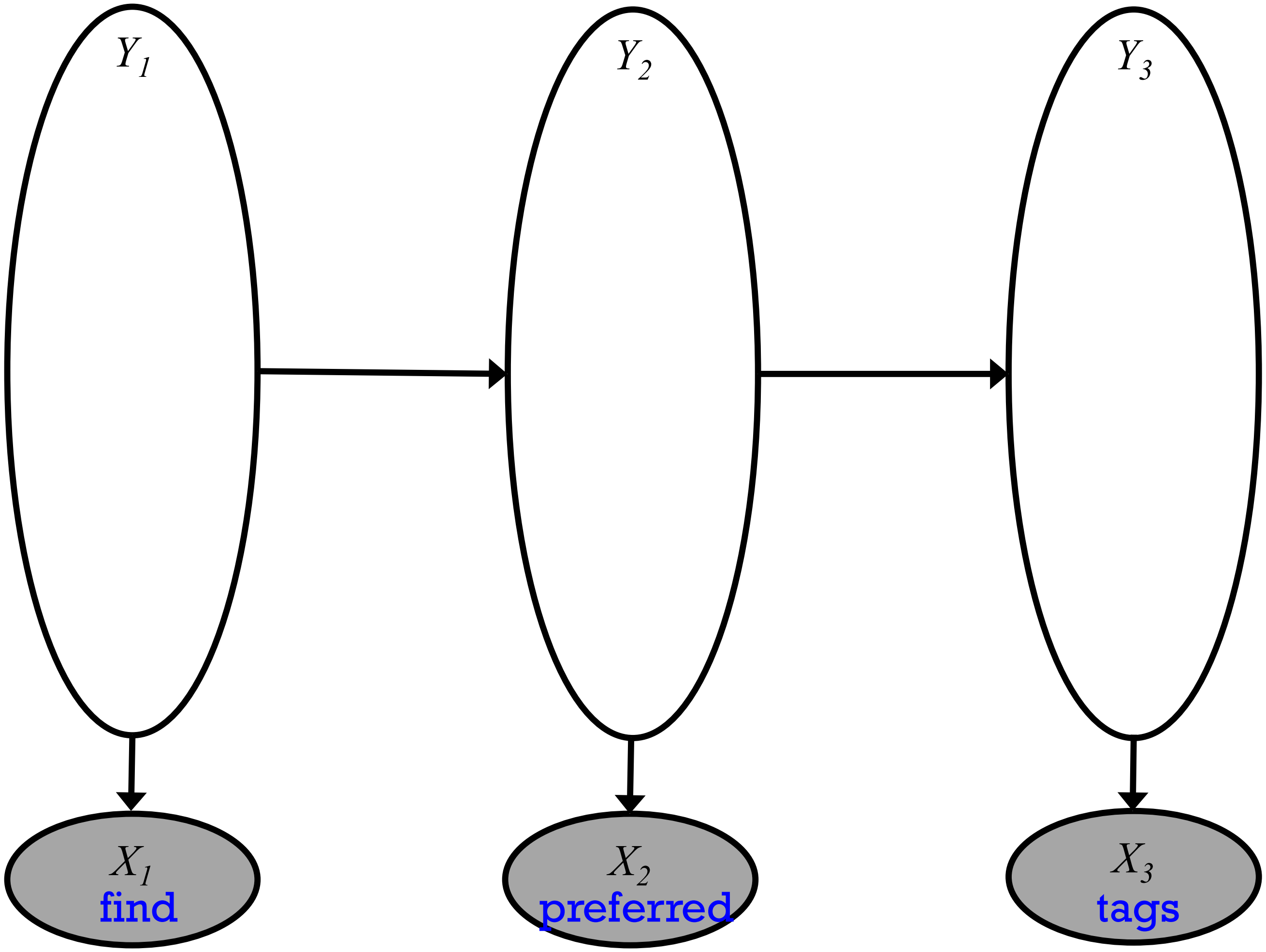


Could be verb or noun

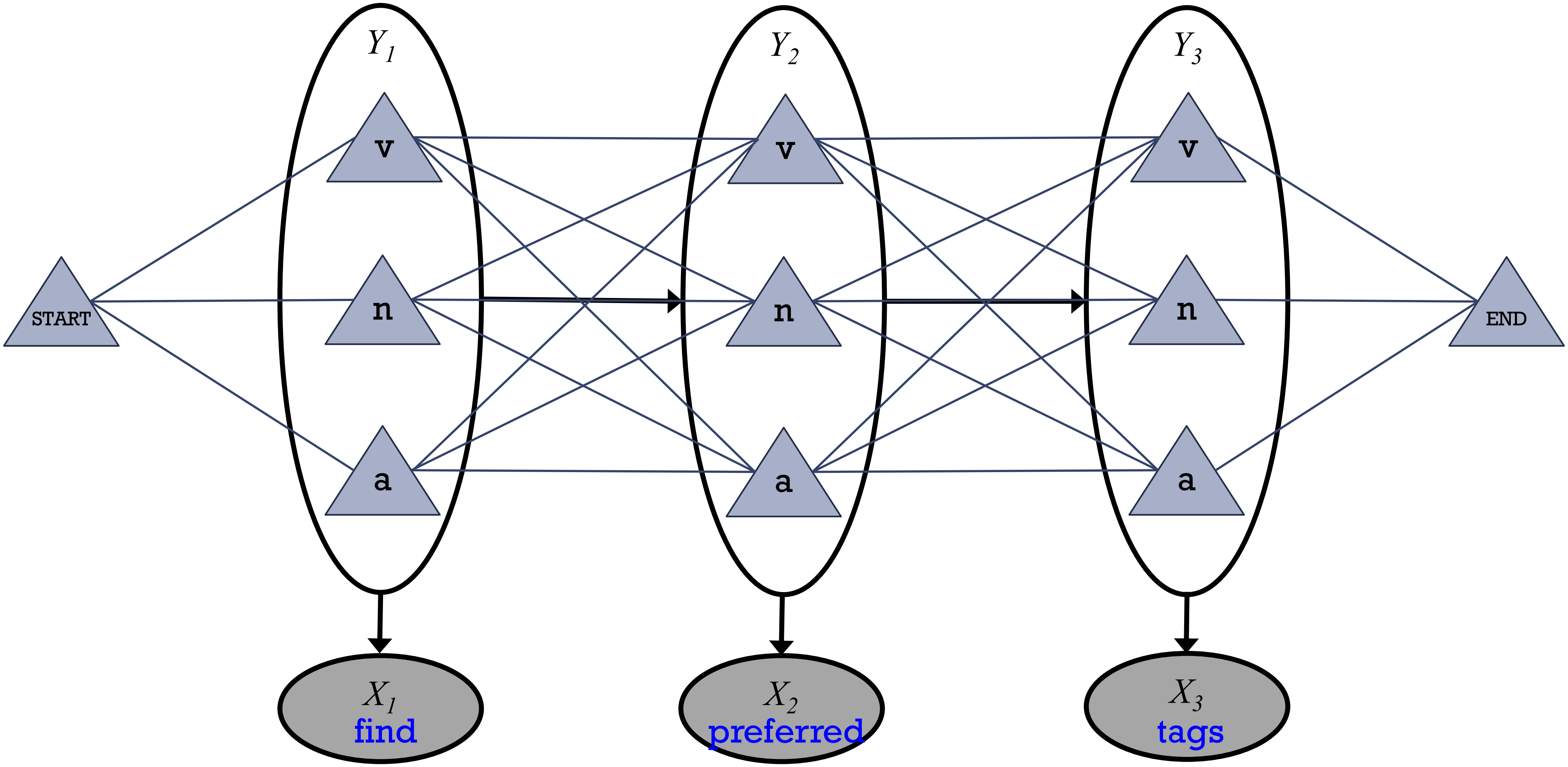
Could be adjective or verb

Could be noun or verb

Forward-Backward Algorithm

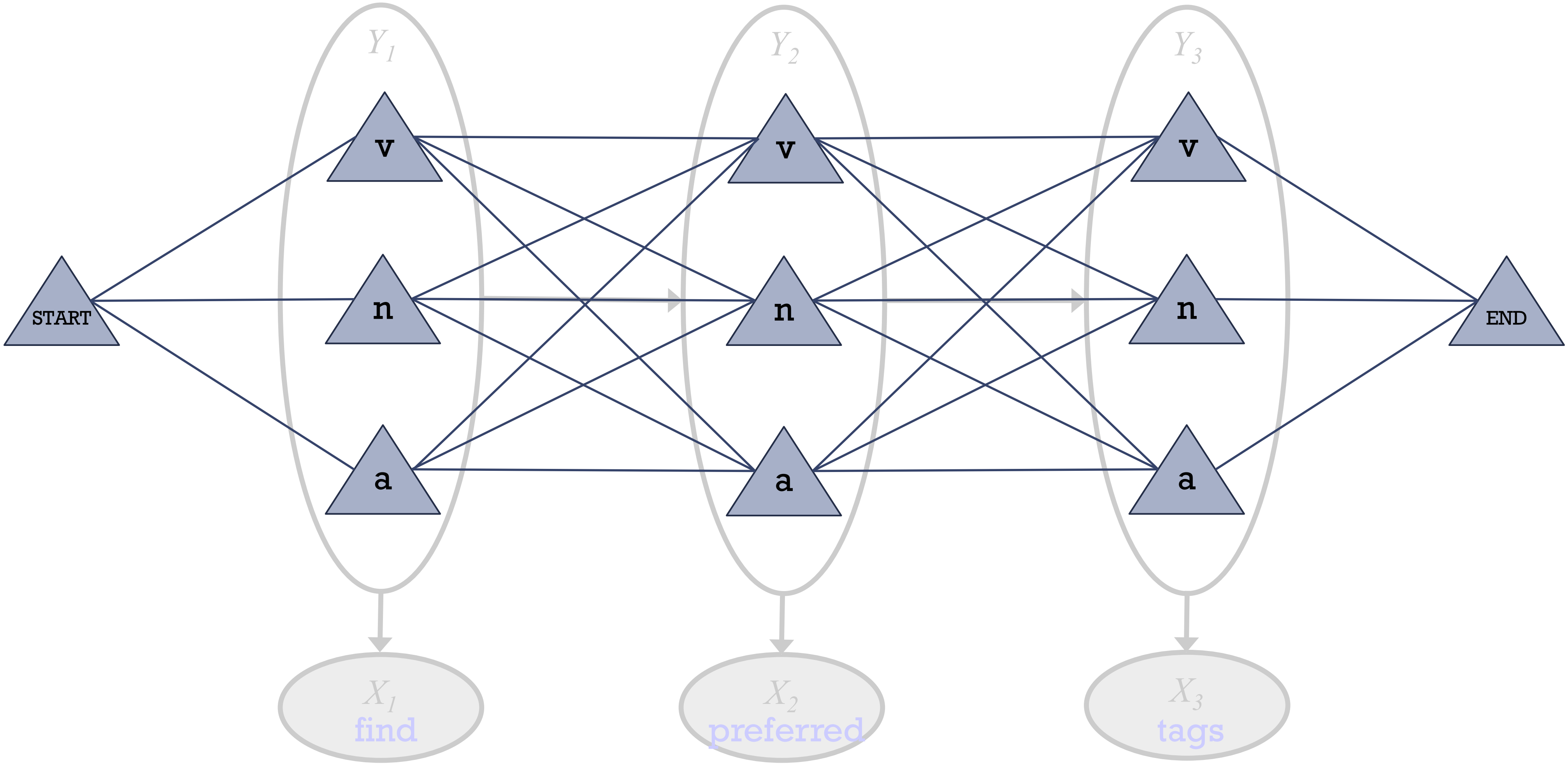


Forward-Backward Algorithm



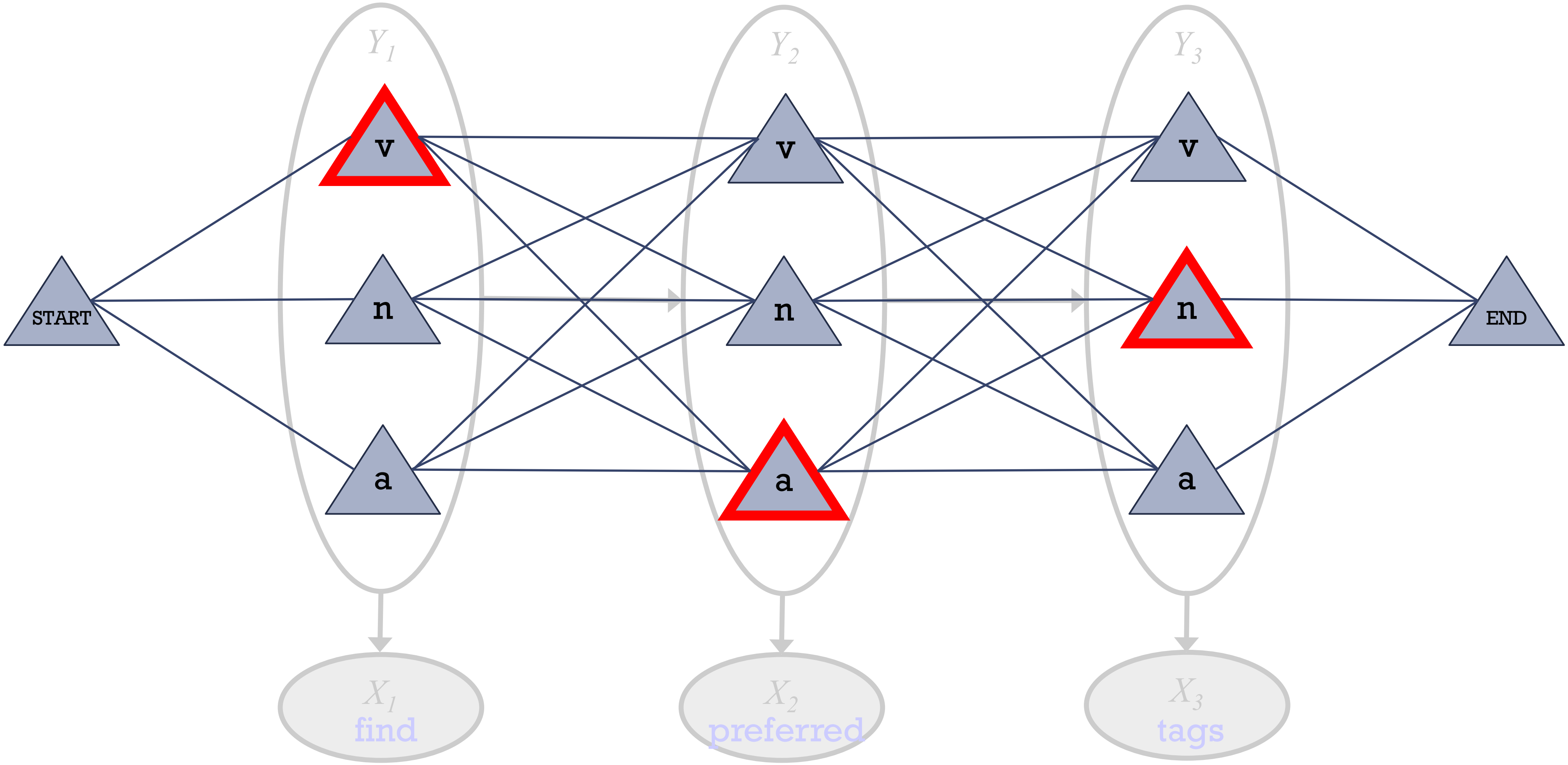
- Let’s show the possible *values* for each variable

Forward-Backward Algorithm



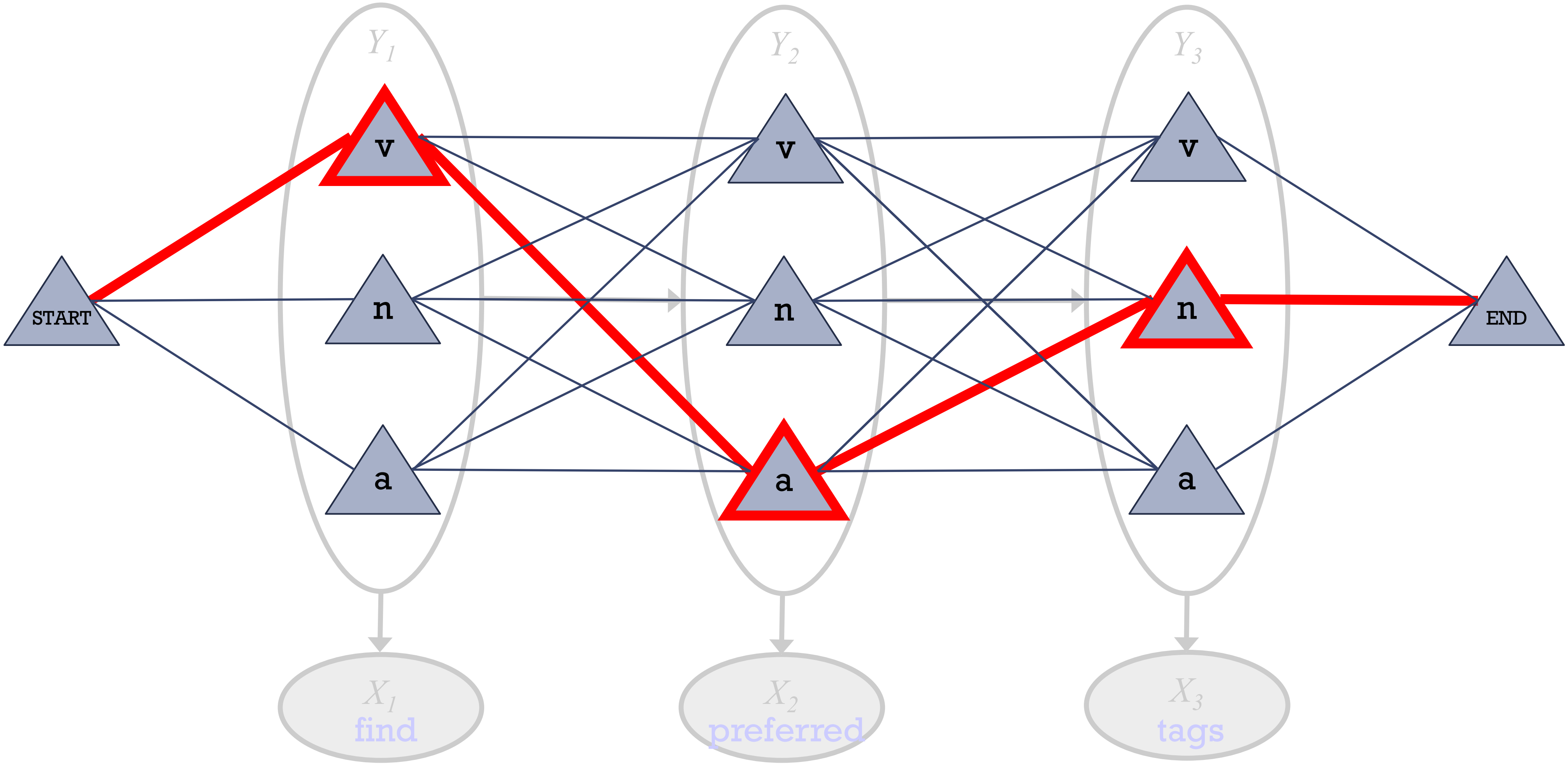
- Let’s show the possible *values* for each variable

Forward-Backward Algorithm



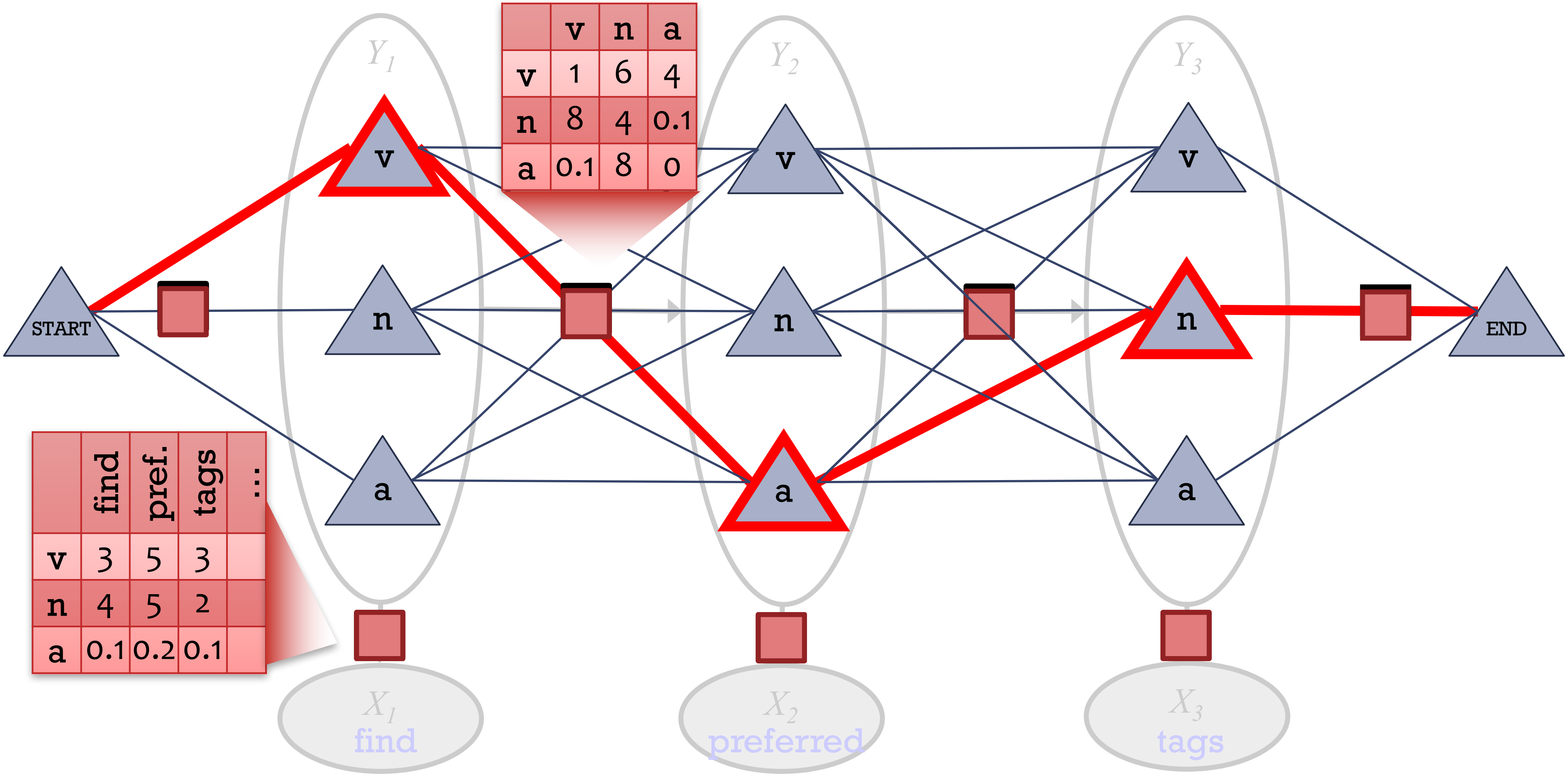
- Let’s show the possible *values* for each variable
- One possible assignment

Forward-Backward Algorithm



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

Forward-Backward Algorithm

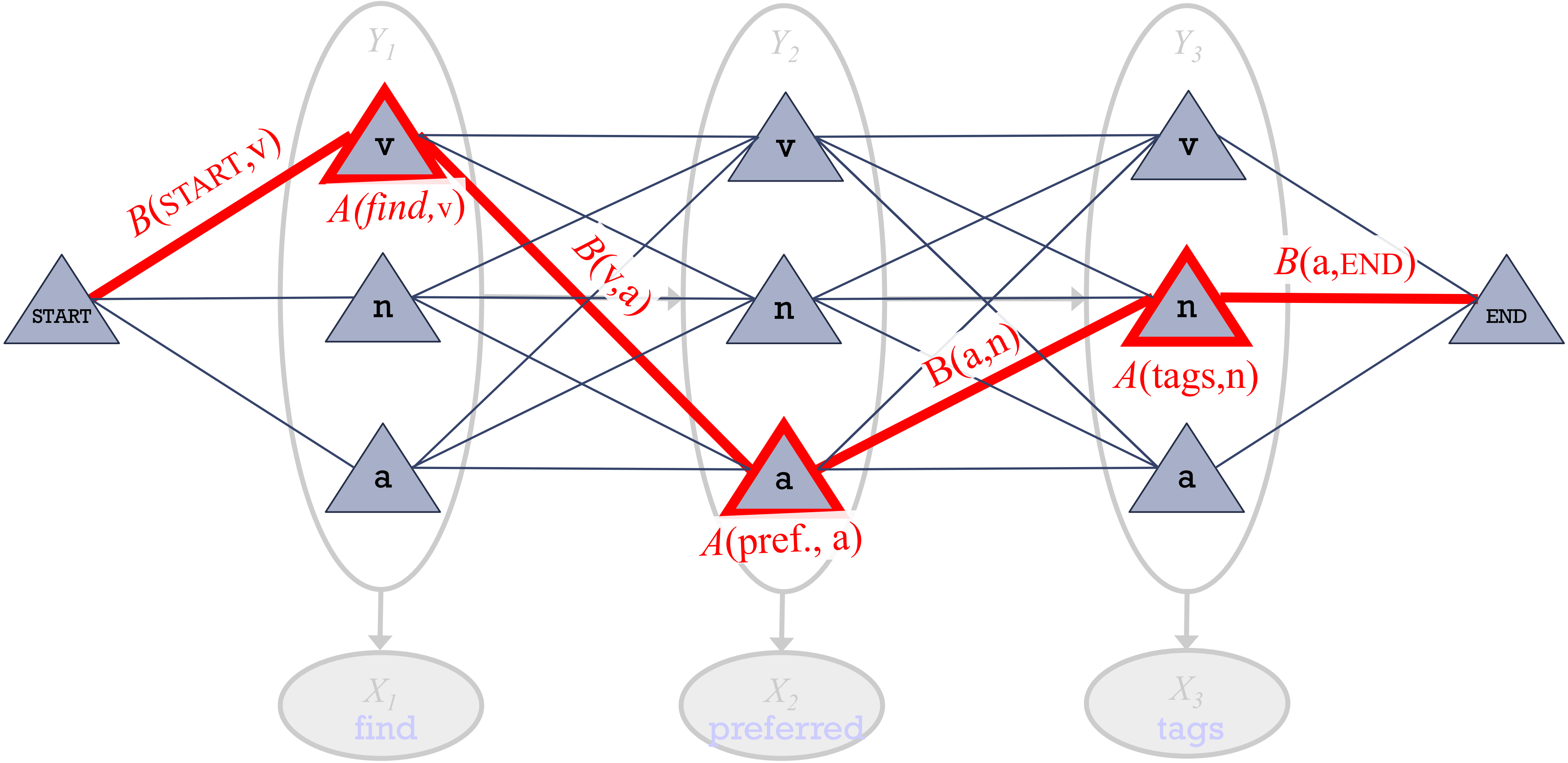


	v	n	a
v	1	6	4
n	8	4	0.1
a	0.1	8	0

	find	pref.	tags	…
v	3	5	3	
n	4	5	2	
a	0.1	0.2	0.1	

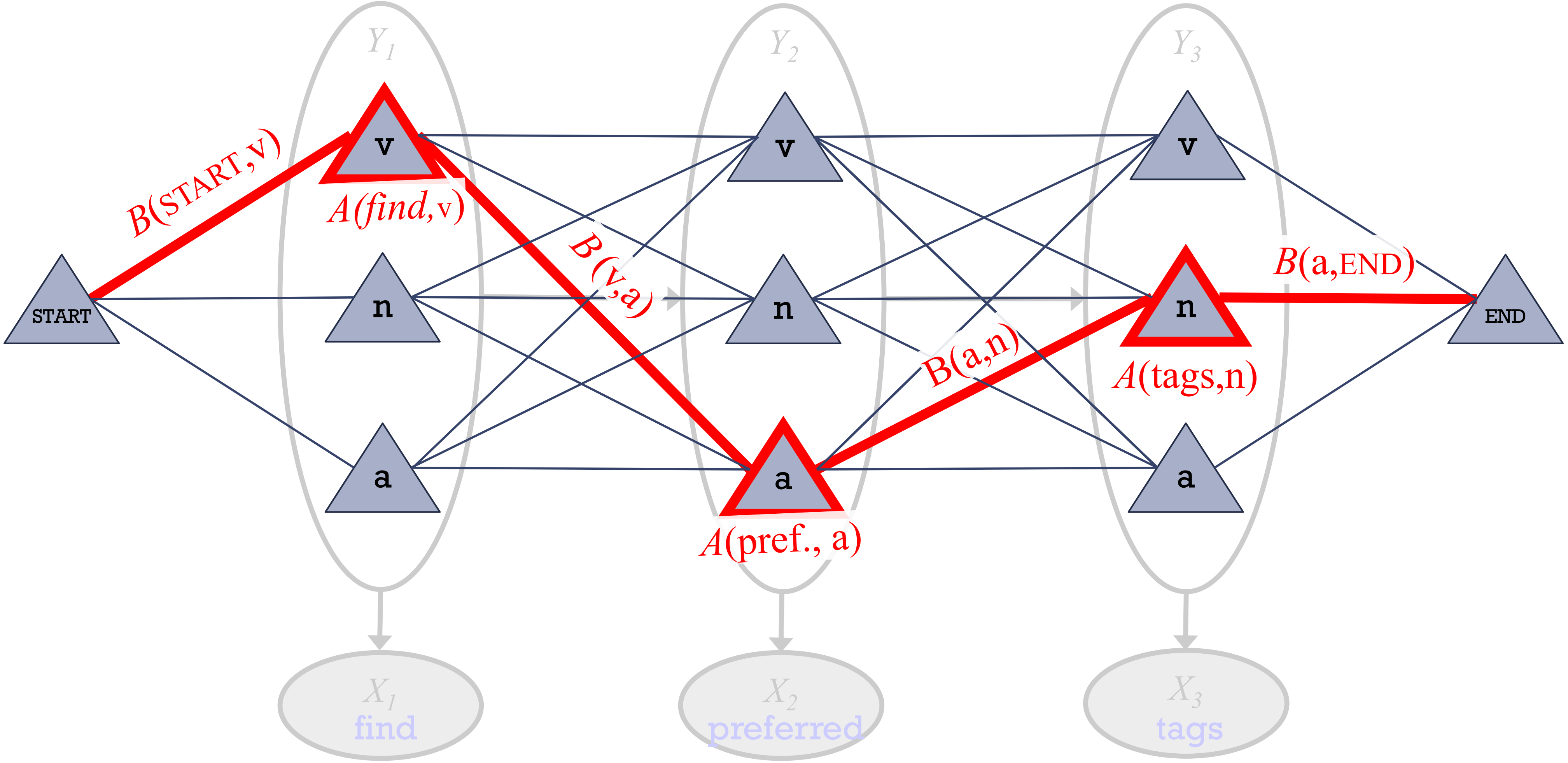
- Let’s show the possible *values* for each variable
- One possible assignment
- And what the 7 transition / emission factors think of it …

Viterbi Algorithm: Most Probable Assignment



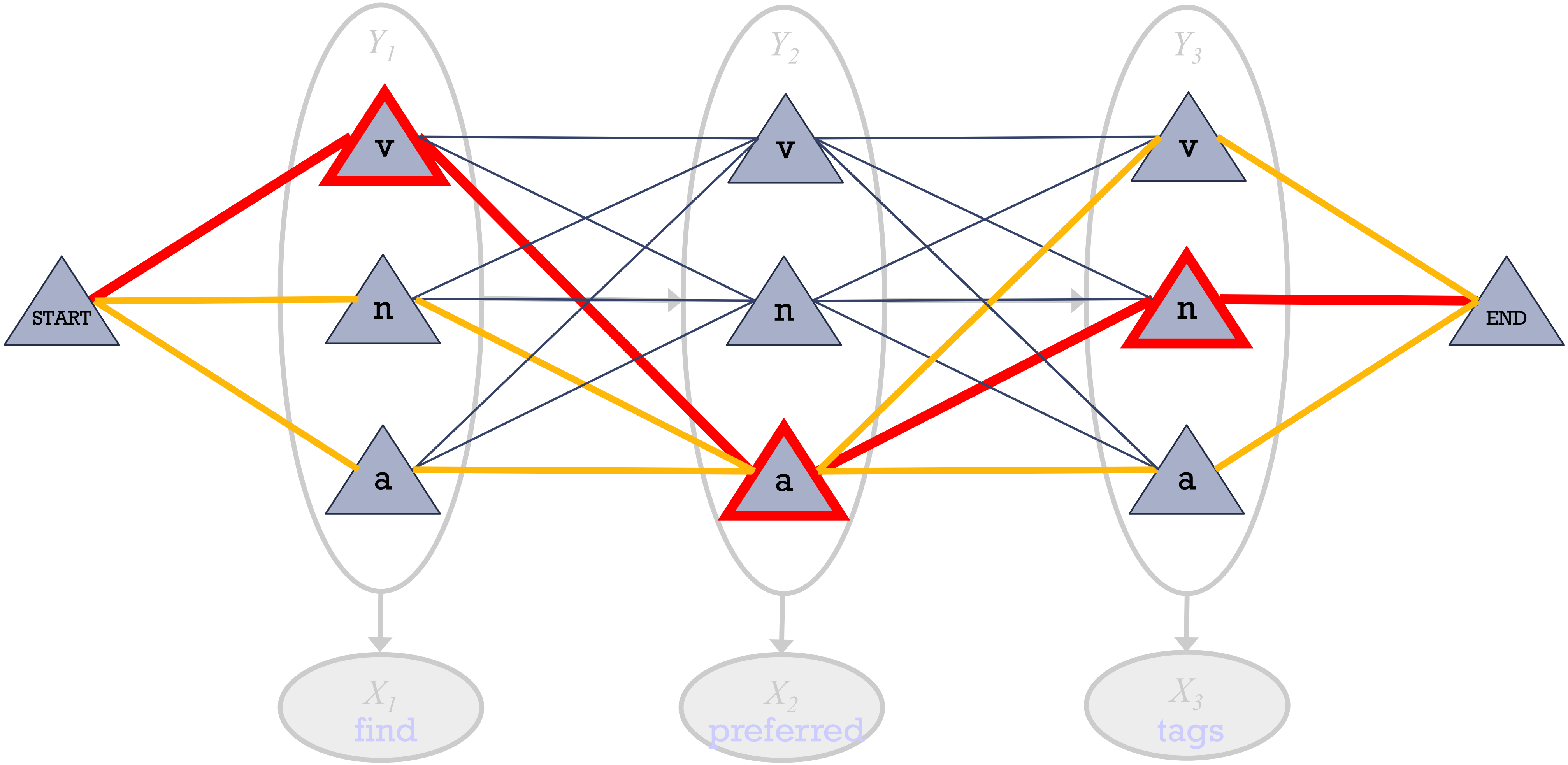
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

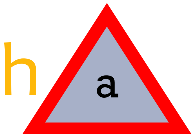
Viterbi Algorithm: Most Probable Assignment



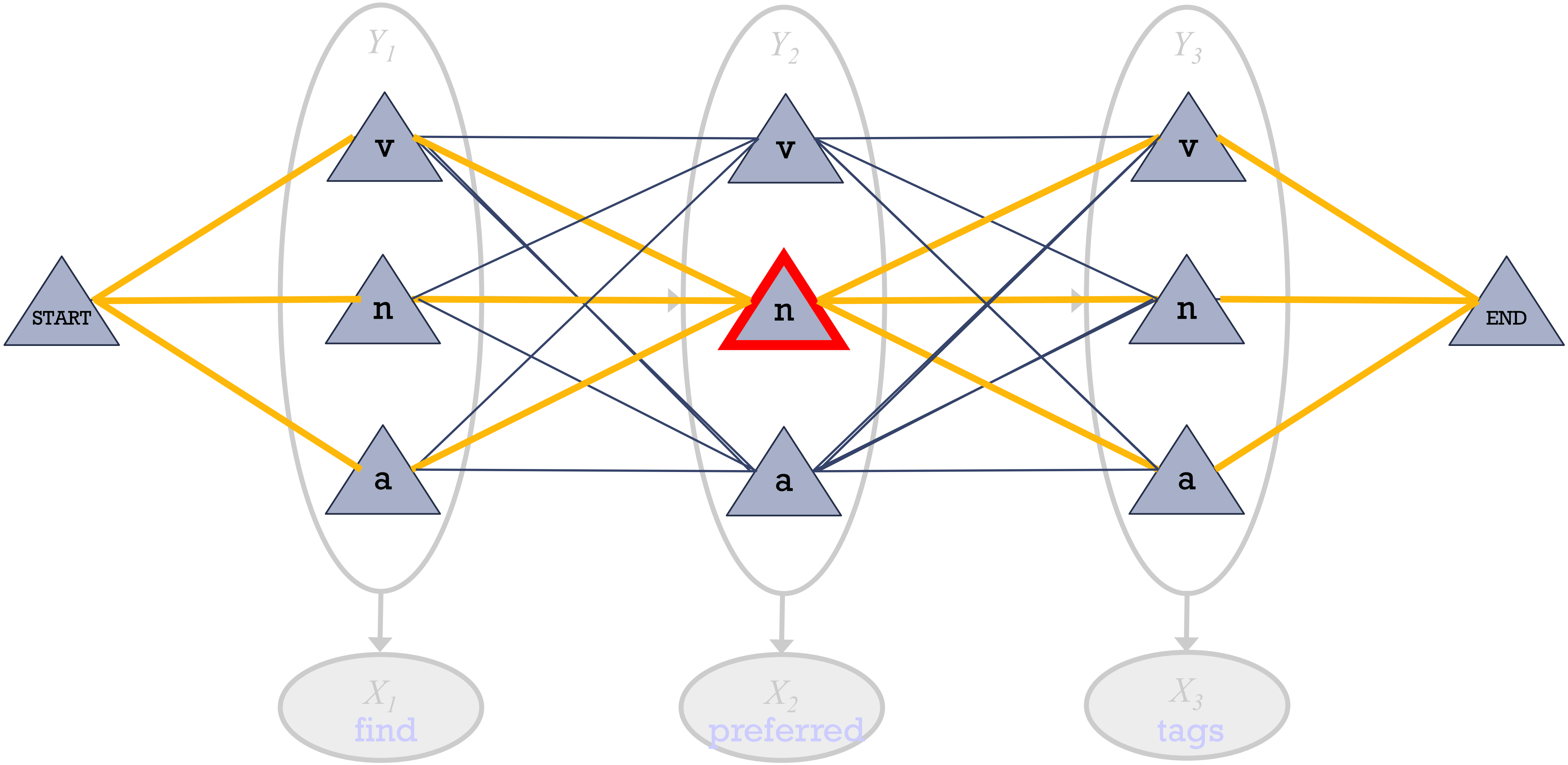
- So p(**v a n**) = (1/Z) * product weight of one path

Forward-Backward Algorithm: Finds Marginals



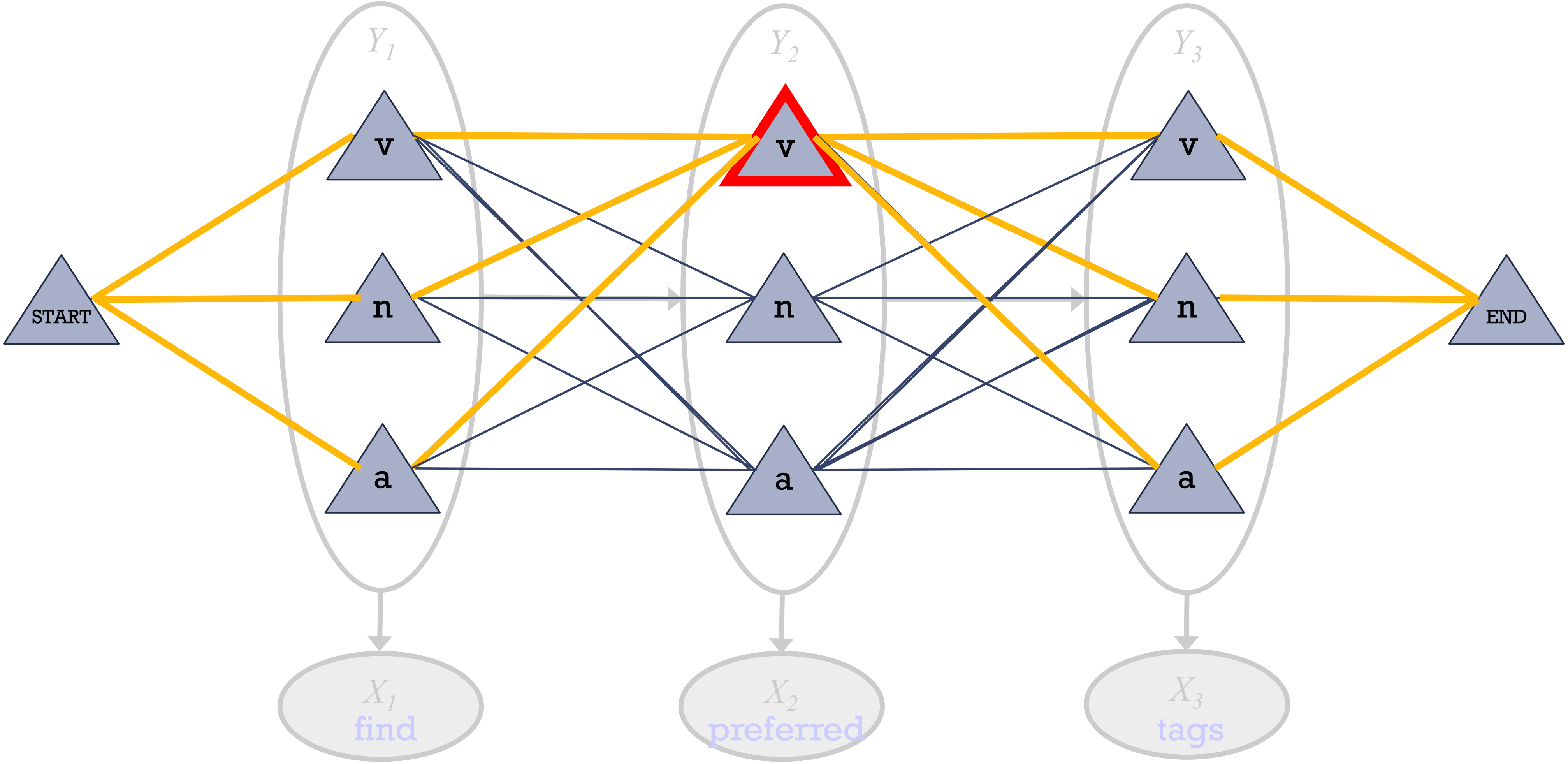
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(Y_2 = a)
  = (1/Z) * total weight of *all* paths through 

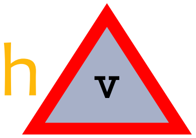
Forward-Backward Algorithm: Finds Marginals



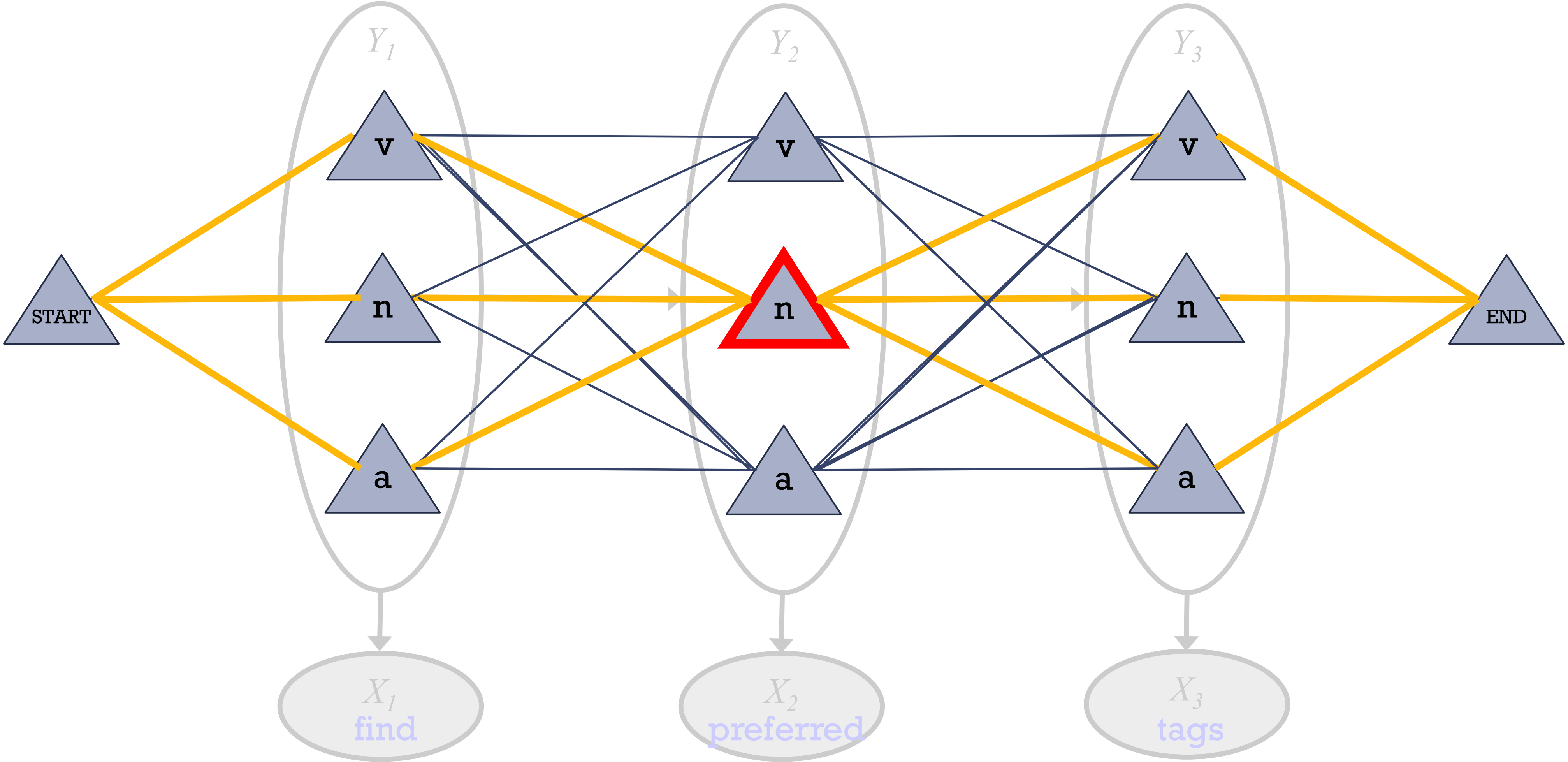
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = n)$
  = (1/Z) * total weight of *all* paths through n

Forward-Backward Algorithm: Finds Marginals



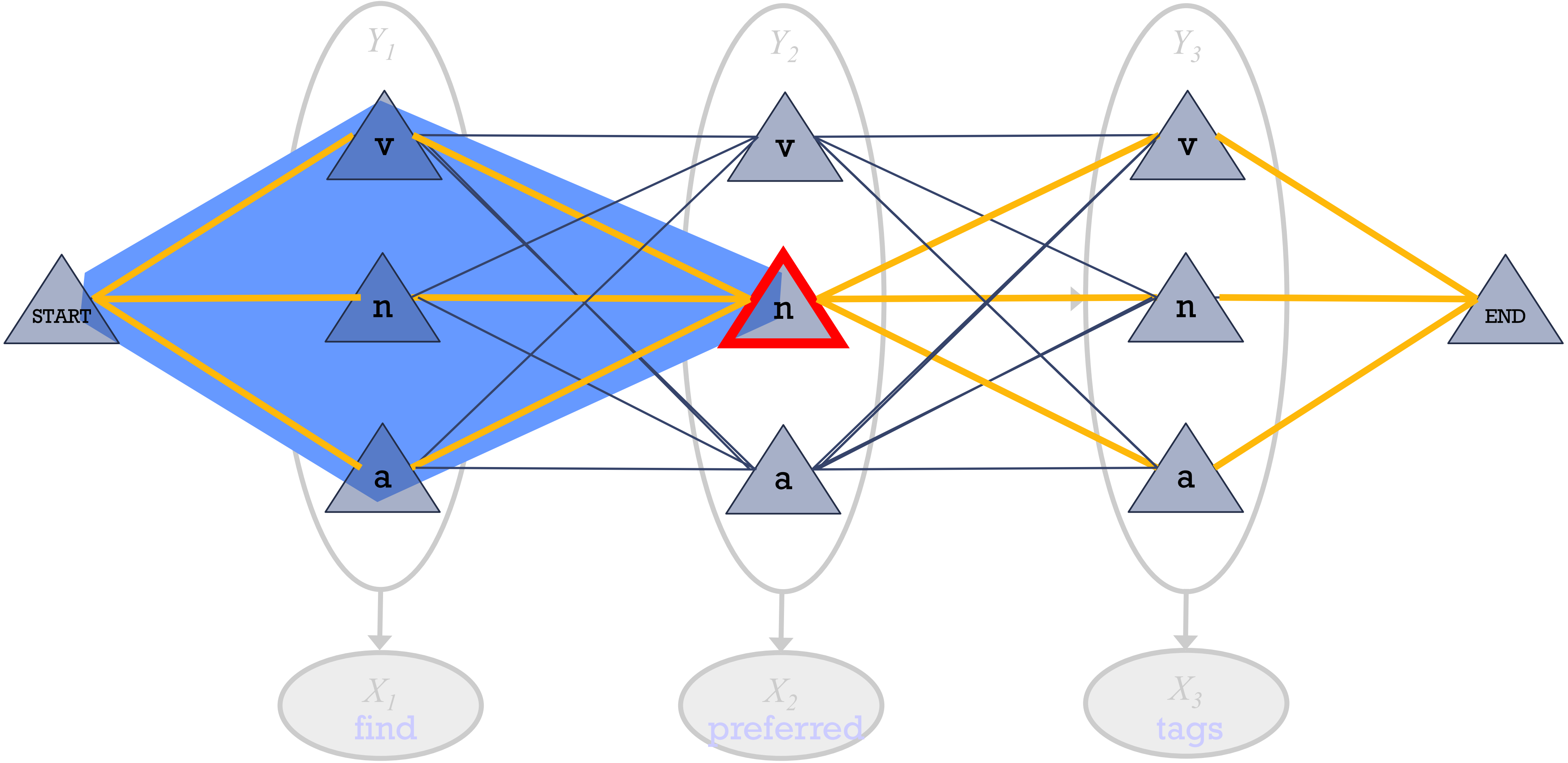
- So $p(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path
- Marginal probability $p(Y_2 = \text{v})$
  $= (1/Z)$ * total weight of *all* paths through 

Forward-Backward Algorithm: Finds Marginals



- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = n)$
  = (1/Z) * total weight of *all* paths through n

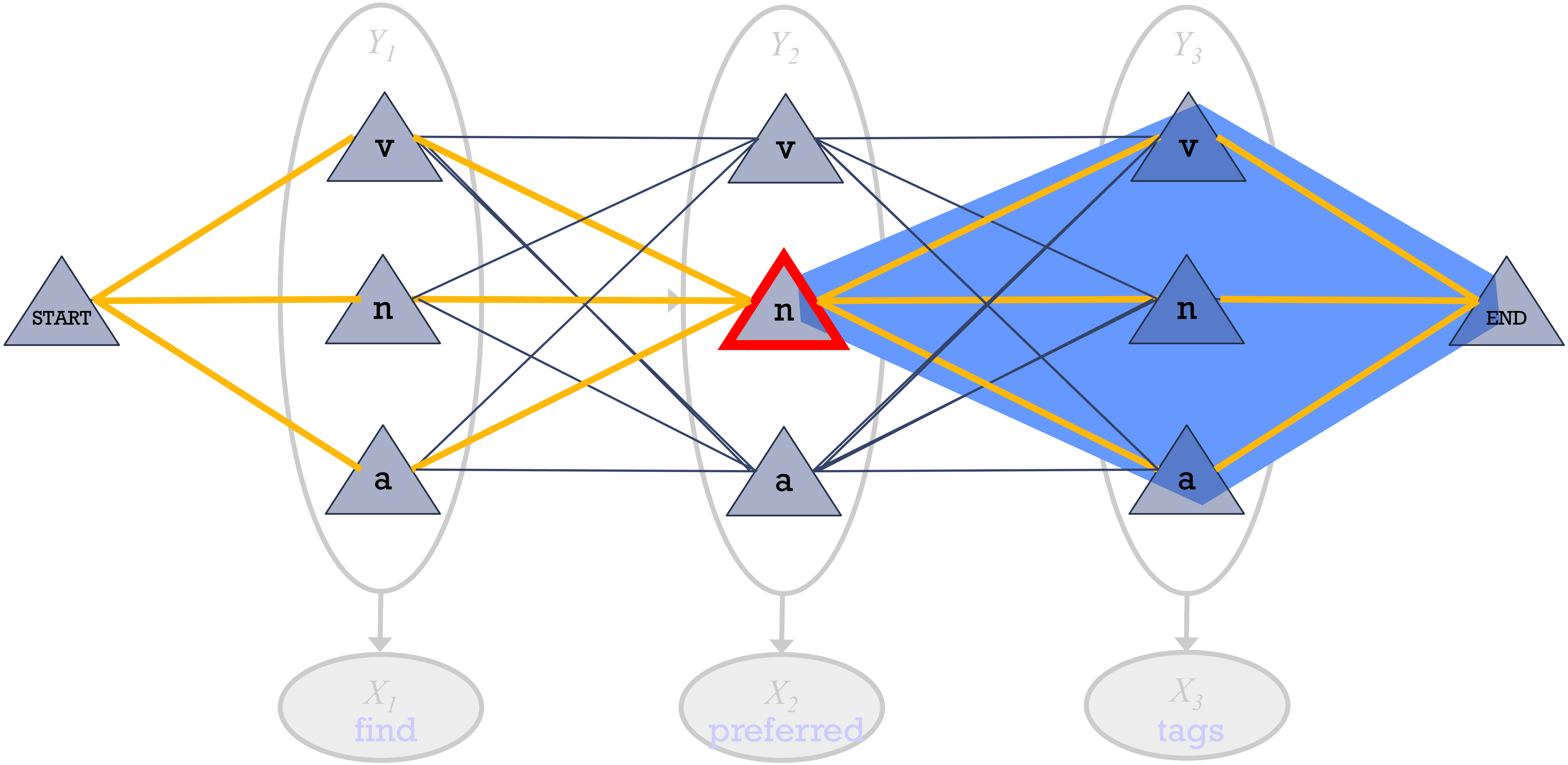
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

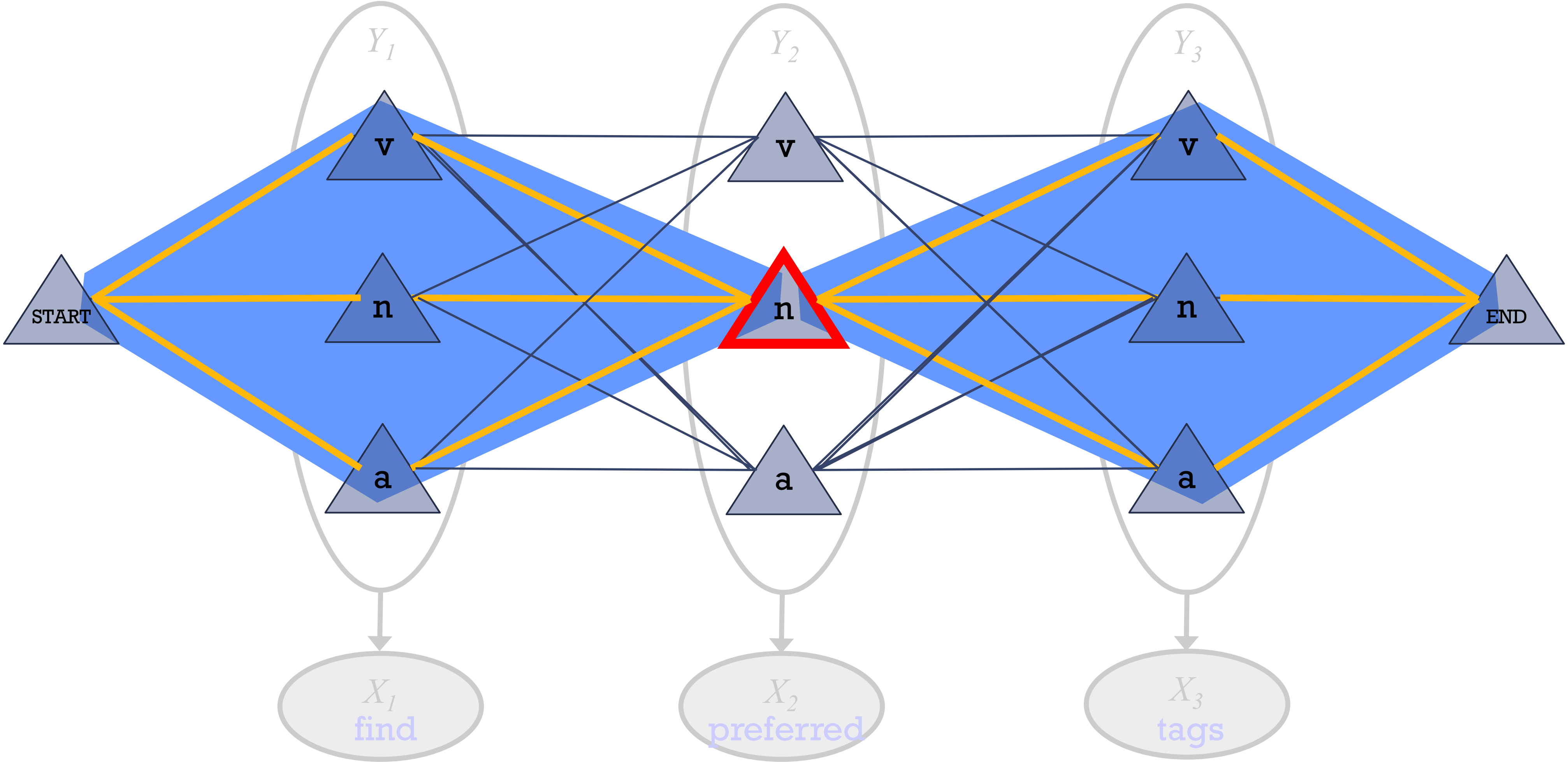
Forward-Backward Algorithm: Finds Marginals



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

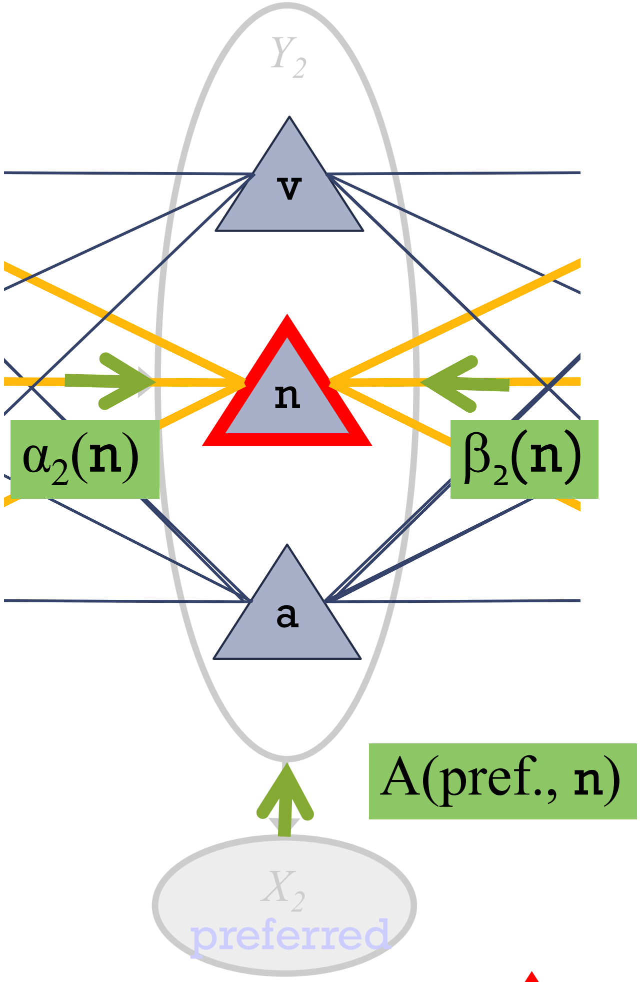
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

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 emission probability at this variable.

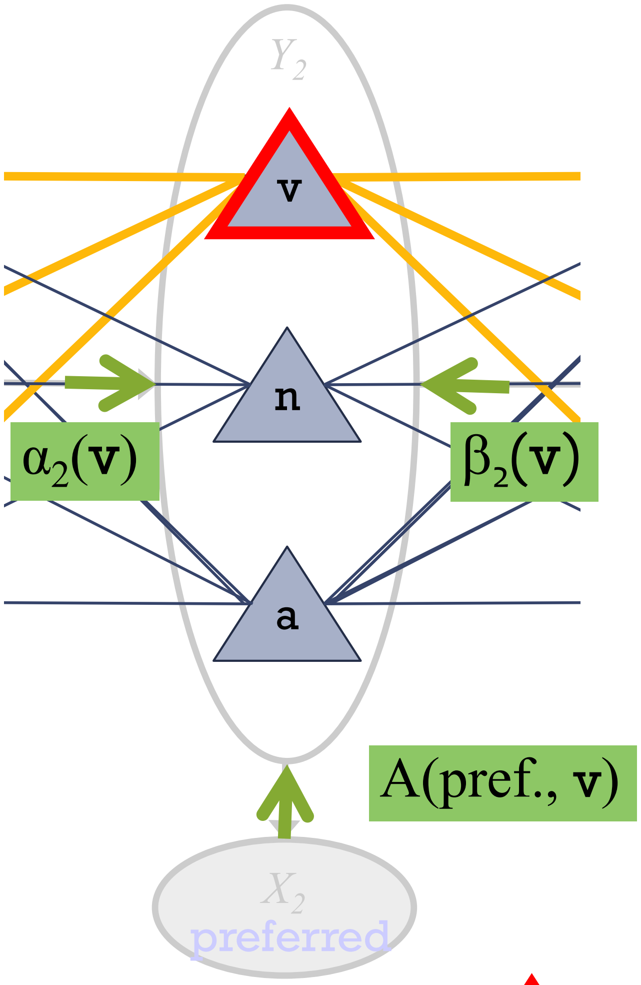


“belief that $Y_2 = \mathbf{n}$”

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

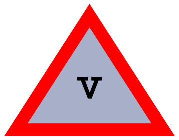
$$= \alpha_2(\mathbf{n}) \quad A(\text{pref.}, \mathbf{n}) \quad \beta_2(\mathbf{n})$$

Forward-Backward Algorithm: Finds Marginals



"belief that Y_2 = **v**"

"belief that Y_2 = **n**"

total weight of *all* paths through 

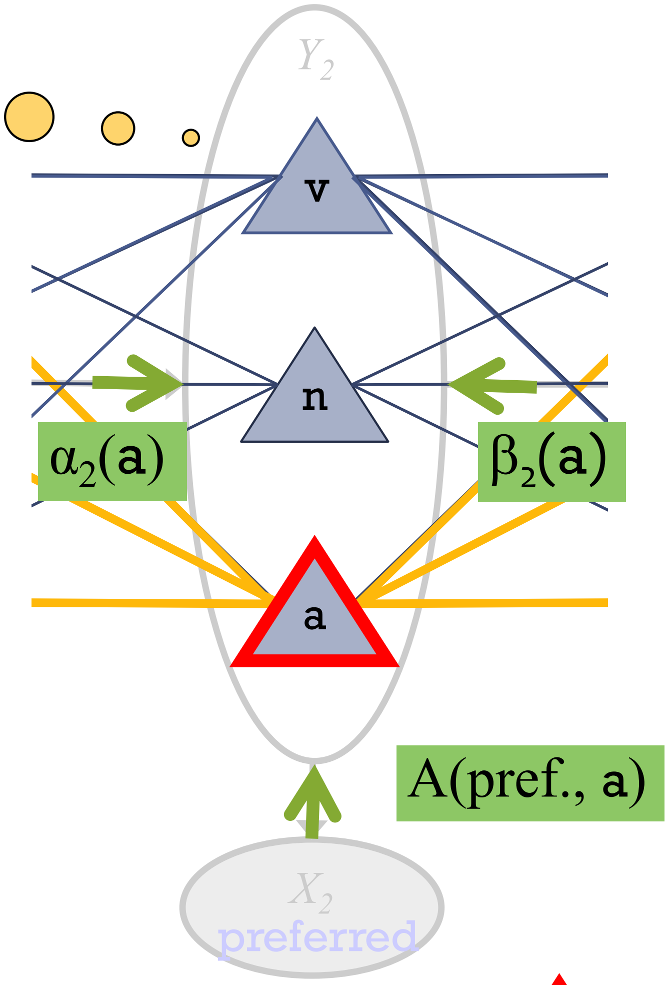
= $\alpha_2(\mathbf{v})$ A(pref., **v**) $\beta_2(\mathbf{v})$

Forward-Backward Algorithm: Finds Marginals

v	0.1
n	0
a	0.4

v	0.2
n	0
a	0.8

divide by Z=0.5 to get marginal probs



“belief that Y_2 = **v**”

“belief that Y_2 = **n**”

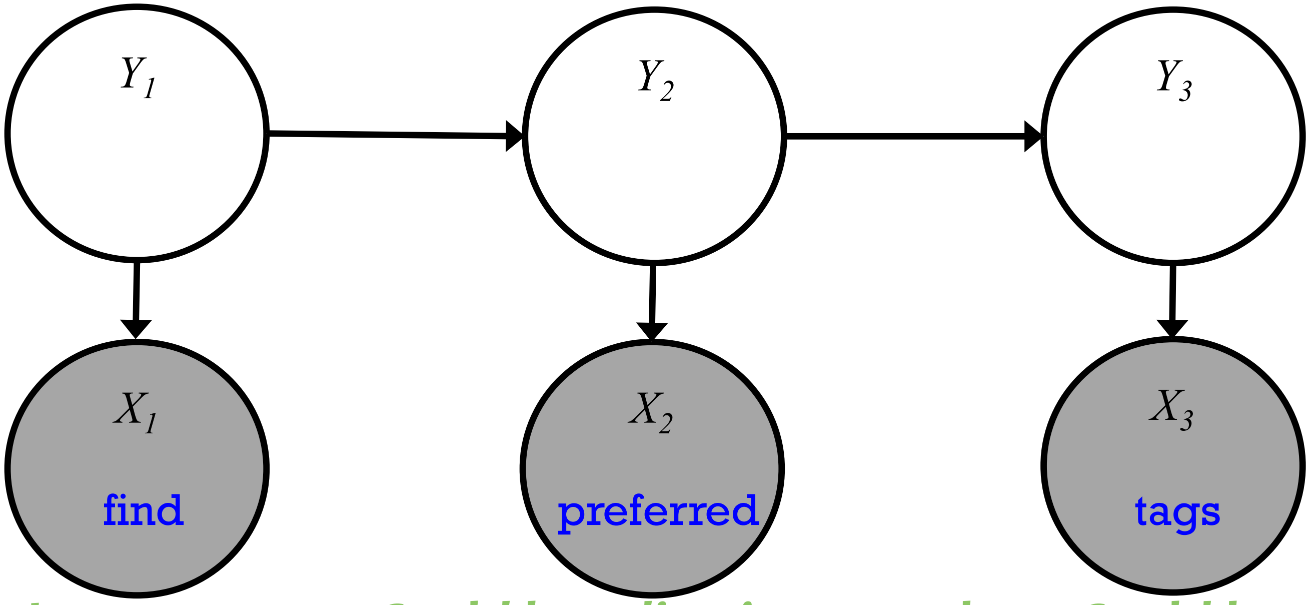
“belief that Y_2 = **a**”

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

total weight of *all* paths through a

= α_2(a) A(pref., a) β_2(a)

Forward-Backward Algorithm



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

Inference for HMMs

Whiteboard

- Derivation of Forward algorithm
- Forward-backward algorithm
- Viterbi algorithm

Inference in HMMs

What is the **computational complexity** of inference for HMMs?

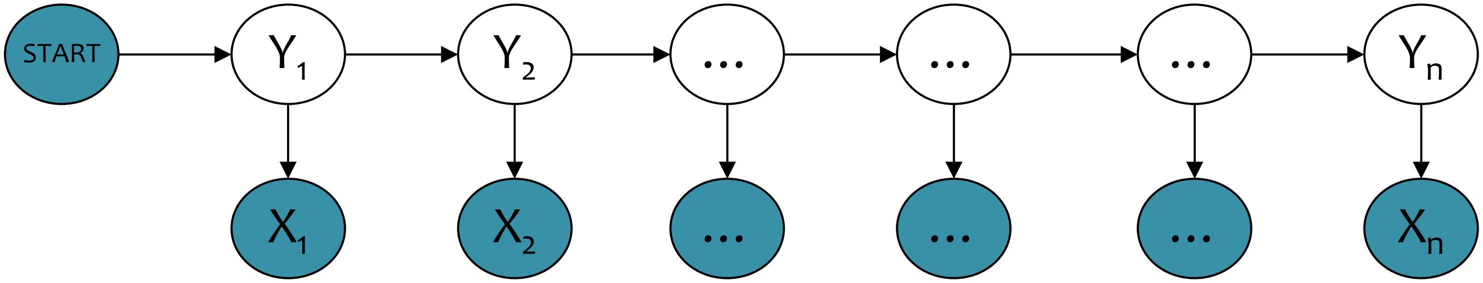
- The **naïve** (brute force) computations for *Evaluation*, *Decoding*, and *Marginals* take **exponential time**, O(K^T)

- The **forward-backward** algorithm and **Viterbi** algorithm run in **polynomial time**, O($T*K^2$)
  - Thanks to dynamic programming!

Conditional Random Fields (CRFs) for time series data

LINEAR-CHAIN CRFS

Shortcomings of Hidden Markov Models

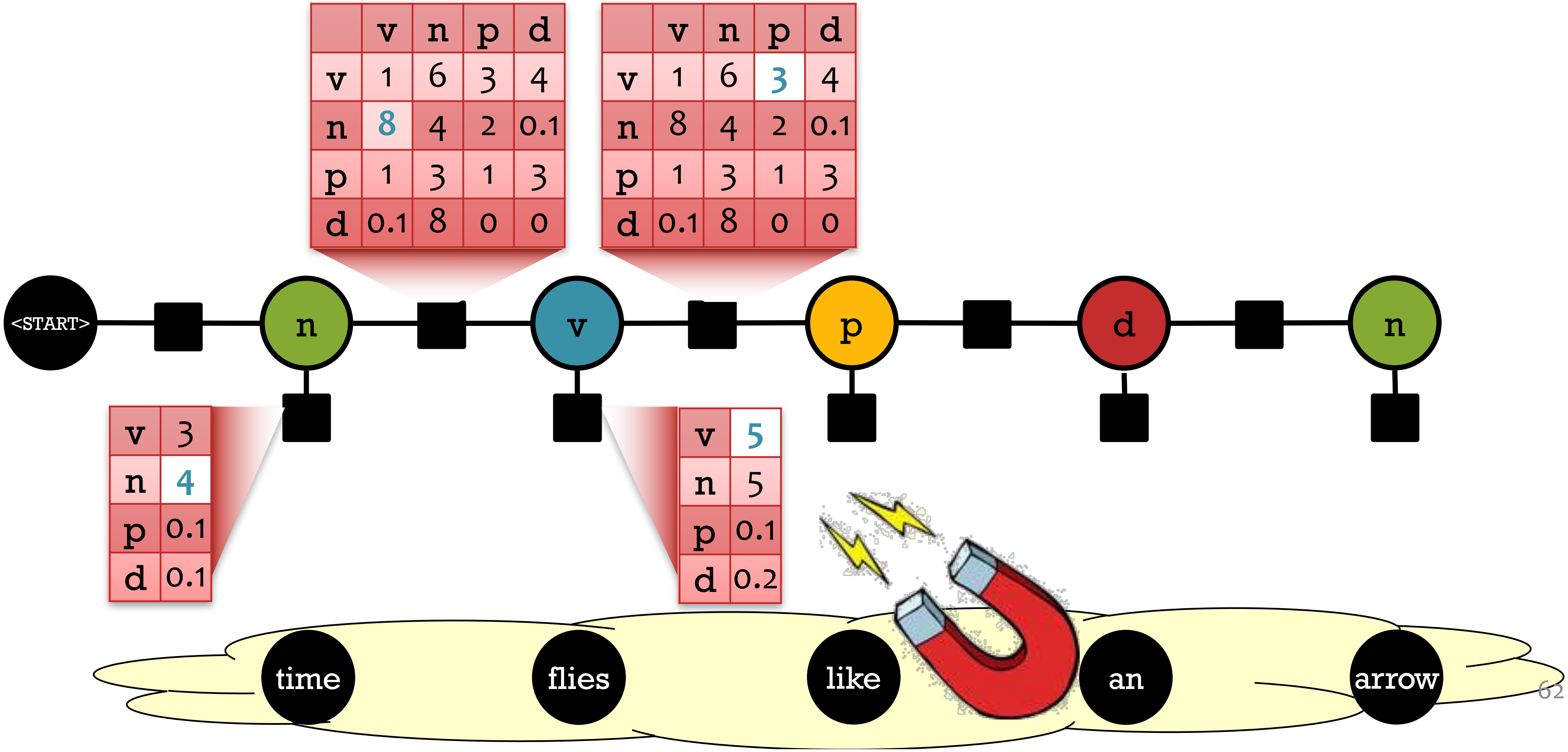


- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.
- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations P(**Y**, **X**), but in a prediction task, we need the conditional probability P(**Y**|**X**)

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.

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



	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

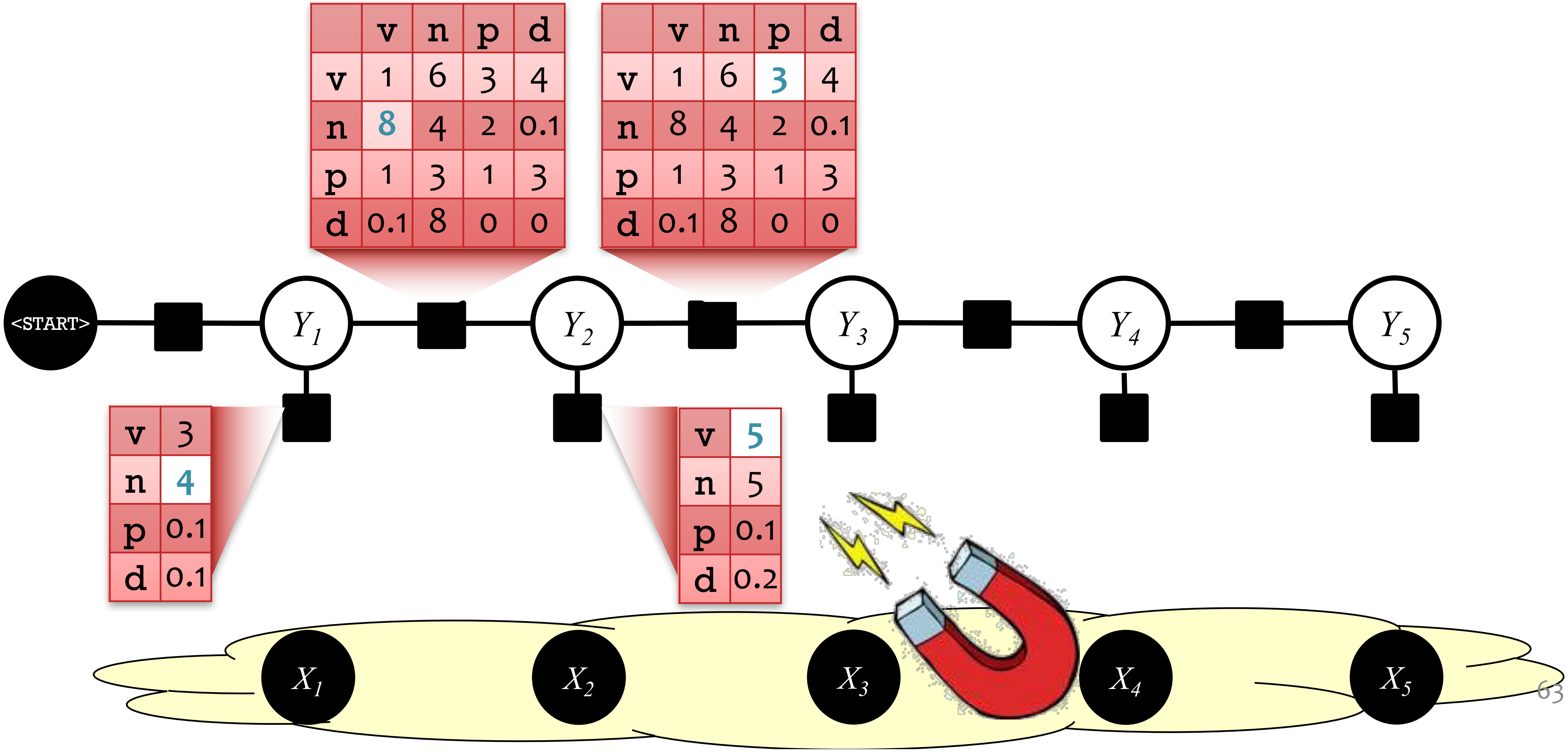
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

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**



	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

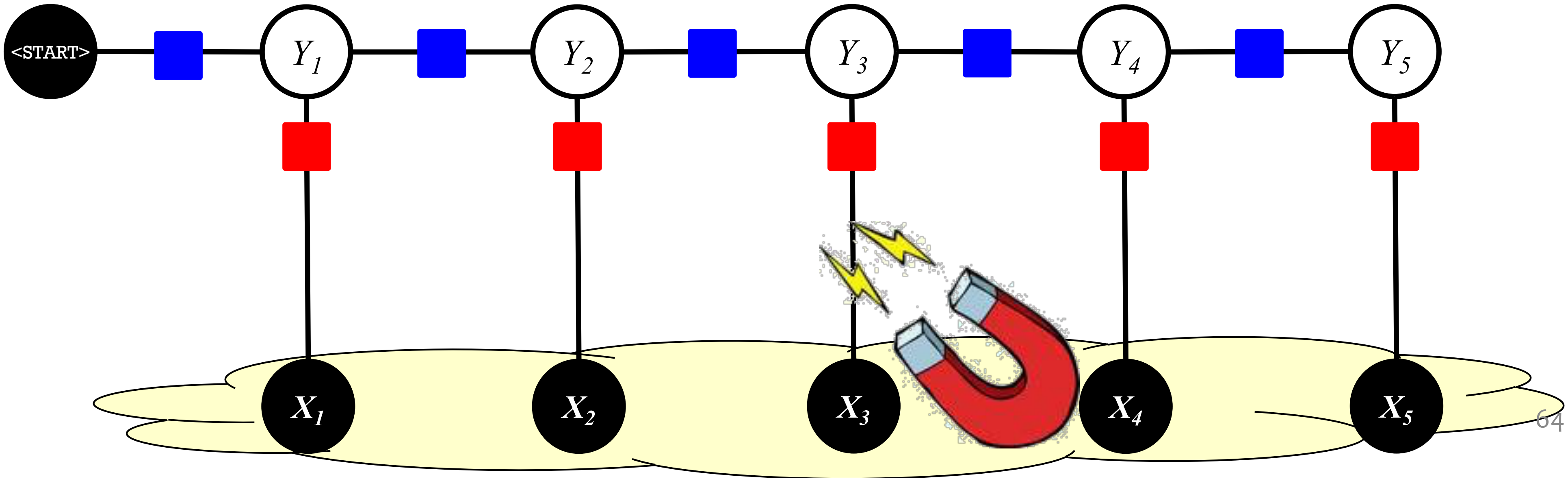
v	5
n	5
p	0.1
d	0.2

Conditional Random Field (CRF)

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

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

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



Exercise

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

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

Multiple Choice: Which model does the above distribution share the most in common with?

A. Hidden Markov Model

B. Bernoulli Naïve Bayes

C. Gaussian Naïve Bayes

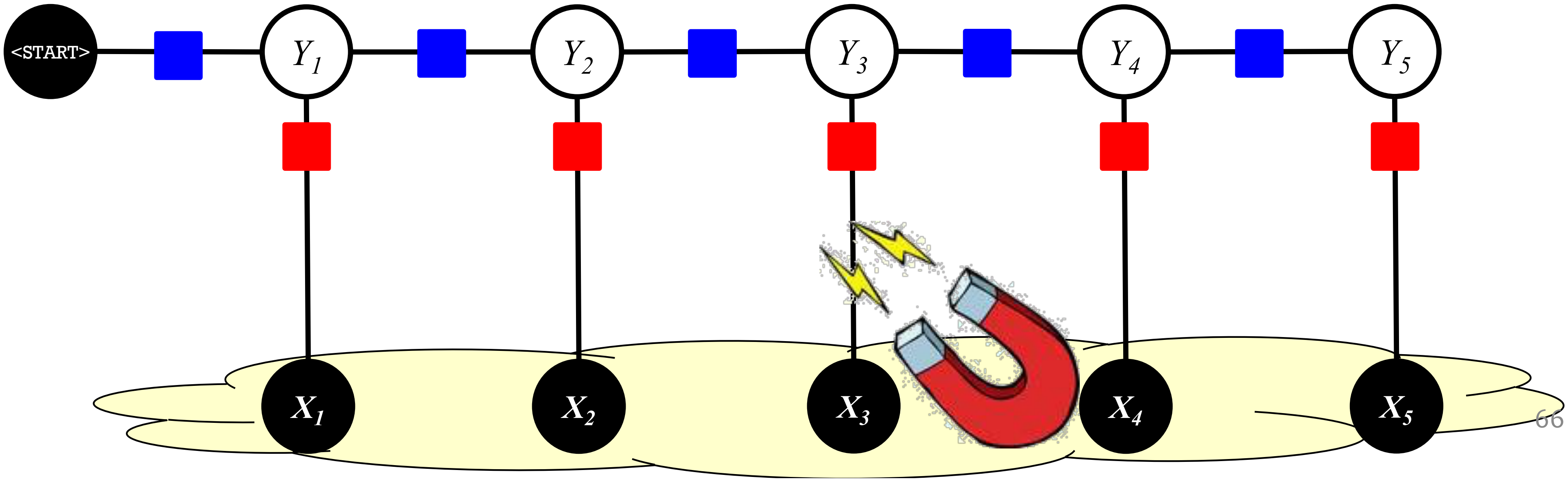
D. Logistic Regression

Conditional Random Field (CRF)

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

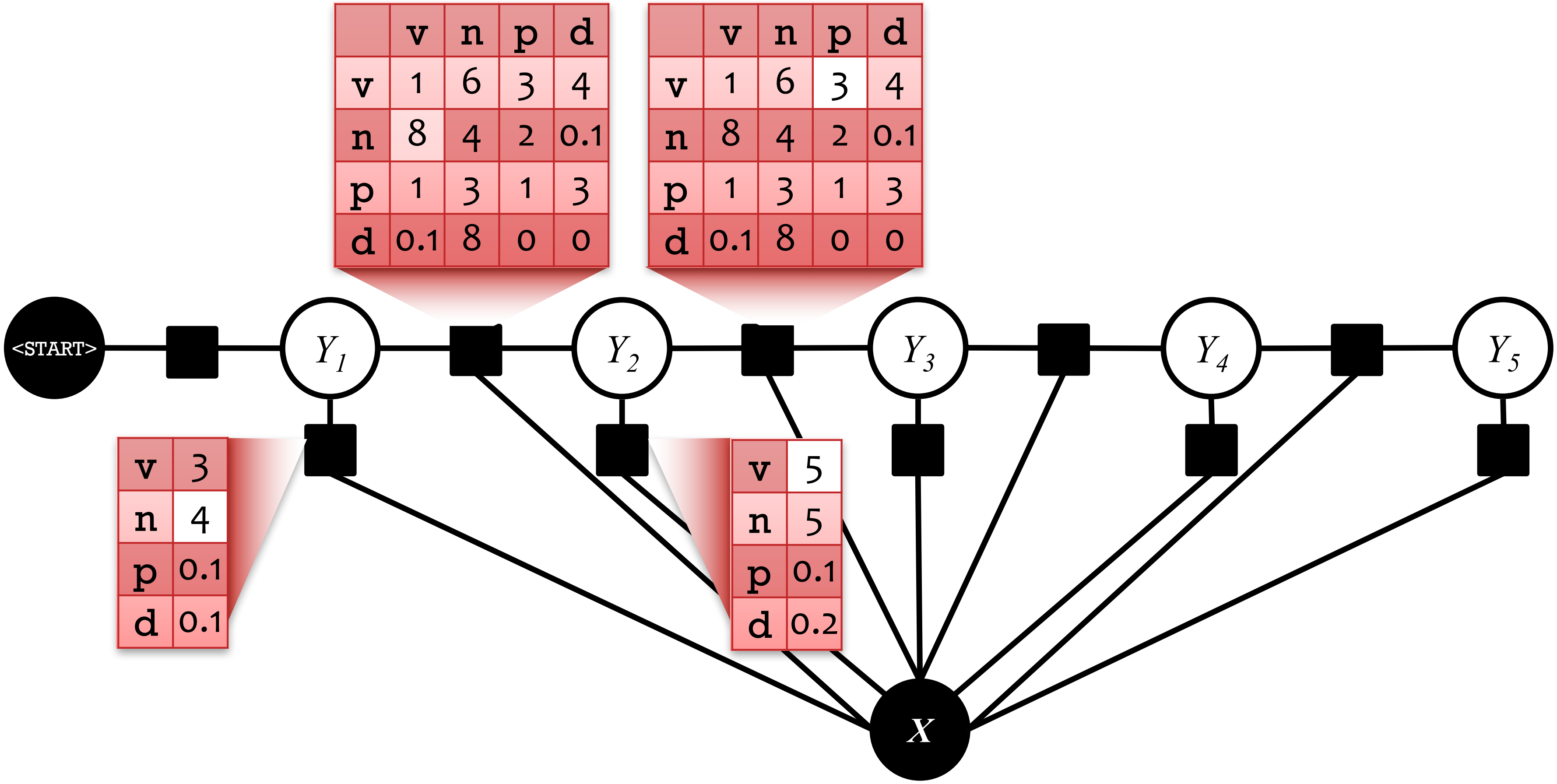
$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k)\psi_{\text{tr}}(y_k, y_{k-1})$$

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



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)



	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

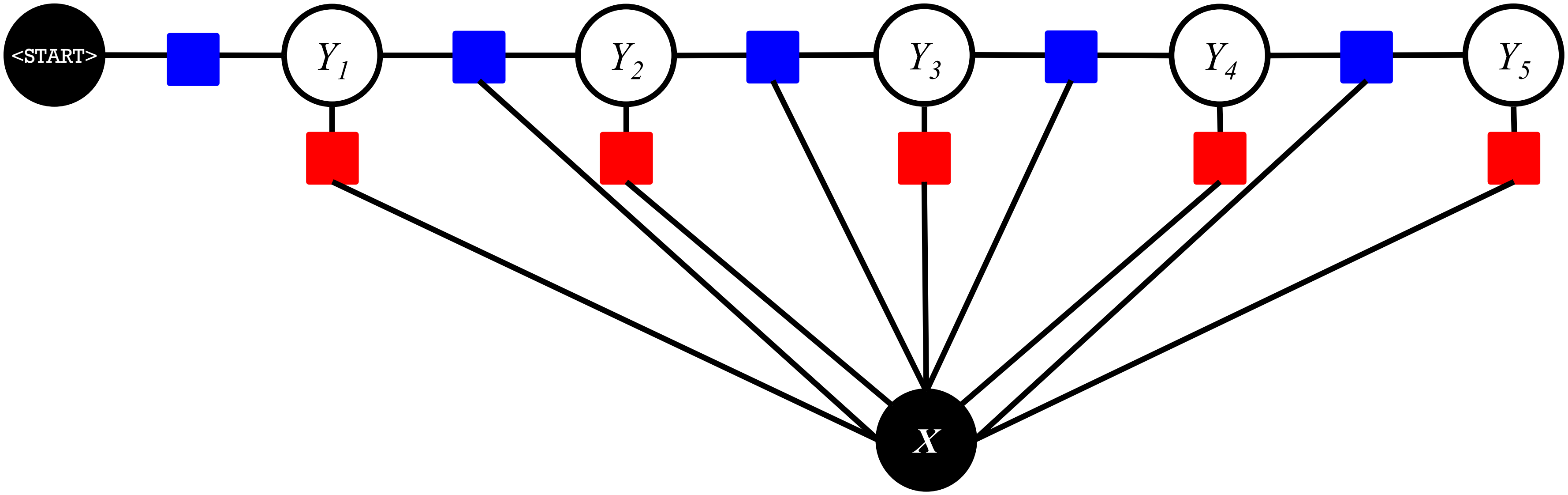
v	5
n	5
p	0.1
d	0.2

Conditional Random Field (CRF)

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

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

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

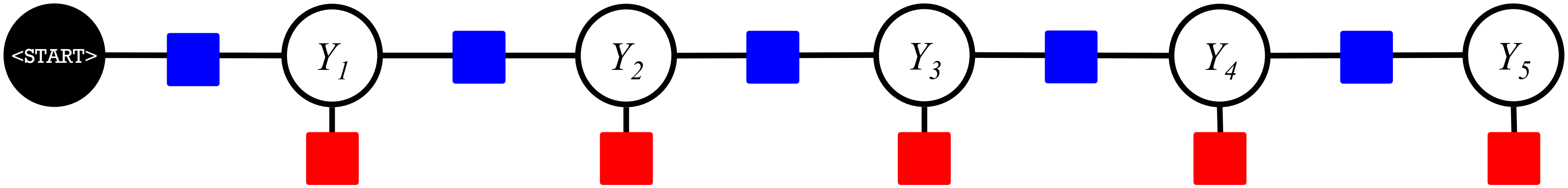


Conditional Random Field (CRF)

- 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 $\boldsymbol{X}$

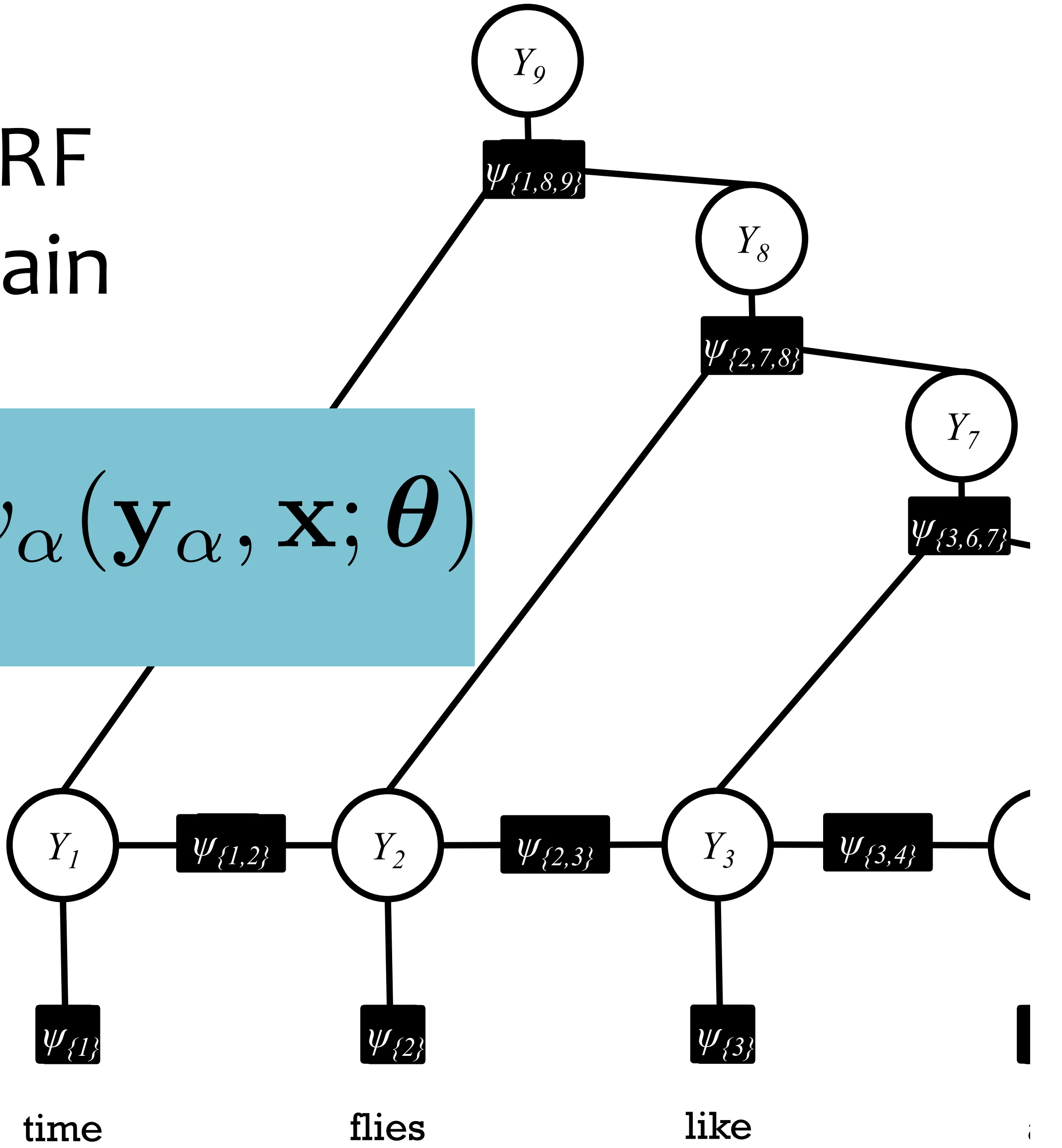
Whiteboard

- Forward-backward algorithm for linear-chain CRF

General CRF

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

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



Standard CRF Parameterization

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

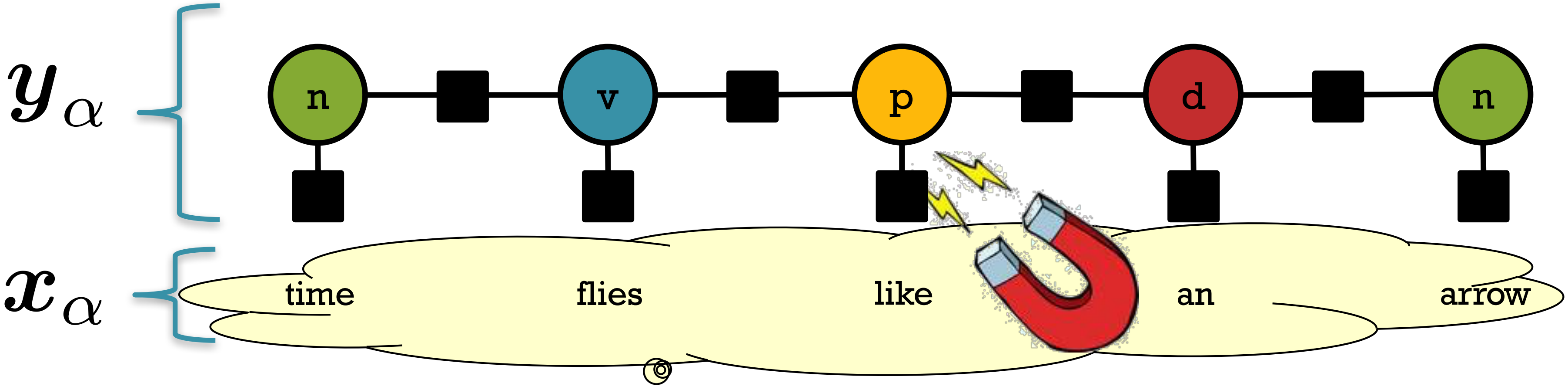
Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\underset{\substack{\uparrow \\ \text{Predicted} \\ \text{variables}}}{\mathbf{y}_\alpha}, \underset{\substack{\uparrow \\ \text{Observed} \\ \text{variables}}}{\mathbf{x}}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$

Standard CRF Parameterization

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

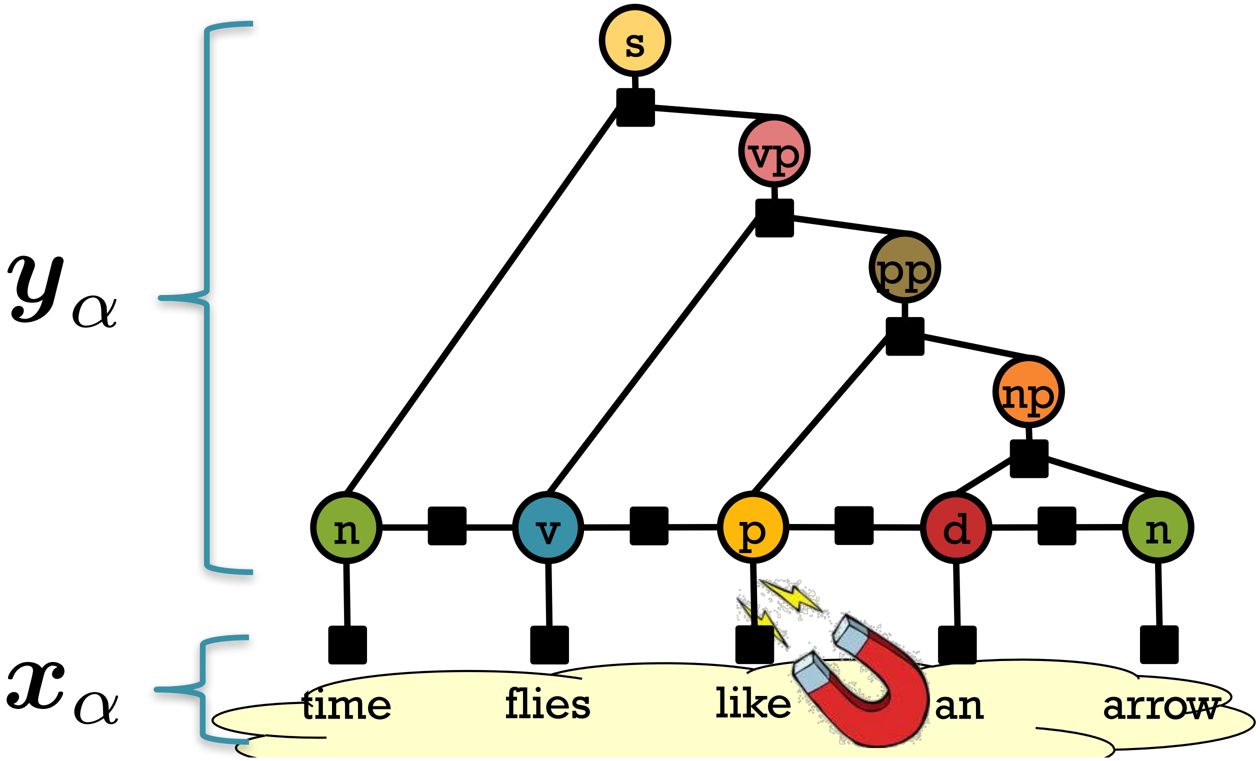
$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$



Standard CRF Parameterization

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

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



SUPERVISED LEARNING FOR CRFS

What is Training?

That’s easy:

Training = picking **good** model parameters!

But how do we know if the
model parameters are any “good”?

Machine Learning

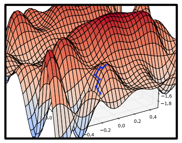
Log-likelihood Training

1. Choose **model**

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$$

2. Choose **objective:** Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} p_\theta(\boldsymbol{y}'_\alpha) f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

Machine Learning

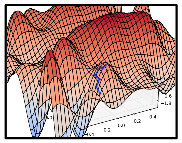
Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_{\alpha} \exp(\theta \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha))$$

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_{\alpha} \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} \boxed{p_\theta(\boldsymbol{y}'_\alpha)} f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

4. Compute **the marginals** by exact inference

Note that these are **factor marginals** which are just the (normalized) **factor beliefs** from BP!

Recipe for Gradient-based Learning

1. Write down the objective function
2. Compute the partial derivatives of the objective (i.e. gradient, and maybe Hessian)
3. Feed objective function and derivatives into black box

Optimization

4. Retrieve optimal parameters from black box

Optimization Algorithms

What is the black box?

- Newton's method
- Hessian-free / Quasi-Newton methods
  - Conjugate gradient
  - L-BFGS
- Stochastic gradient methods
  - Stochastic gradient descent (SGD)
  - Stochastic meta-descent
  - AdaGrad

Stochastic Gradient Descent

- Suppose we have N training examples s.t. $f(x) = \sum_{i=1}^{N} f_i(x)$.
- This implies that $\nabla f(x) = \sum_{i=1}^{N} \nabla f_i(x)$.

```
SGD Algorithm:
  1. Choose a starting point $x$.
  2. While not converged:
       ○ Choose a step size $t$.
       ○ Choose $i$ so that it sweeps through the training set.
       ○ Update
             $\vec{x}^{(k+1)} = \vec{x}^{(k)} + t \nabla f_i(\vec{x})$
```

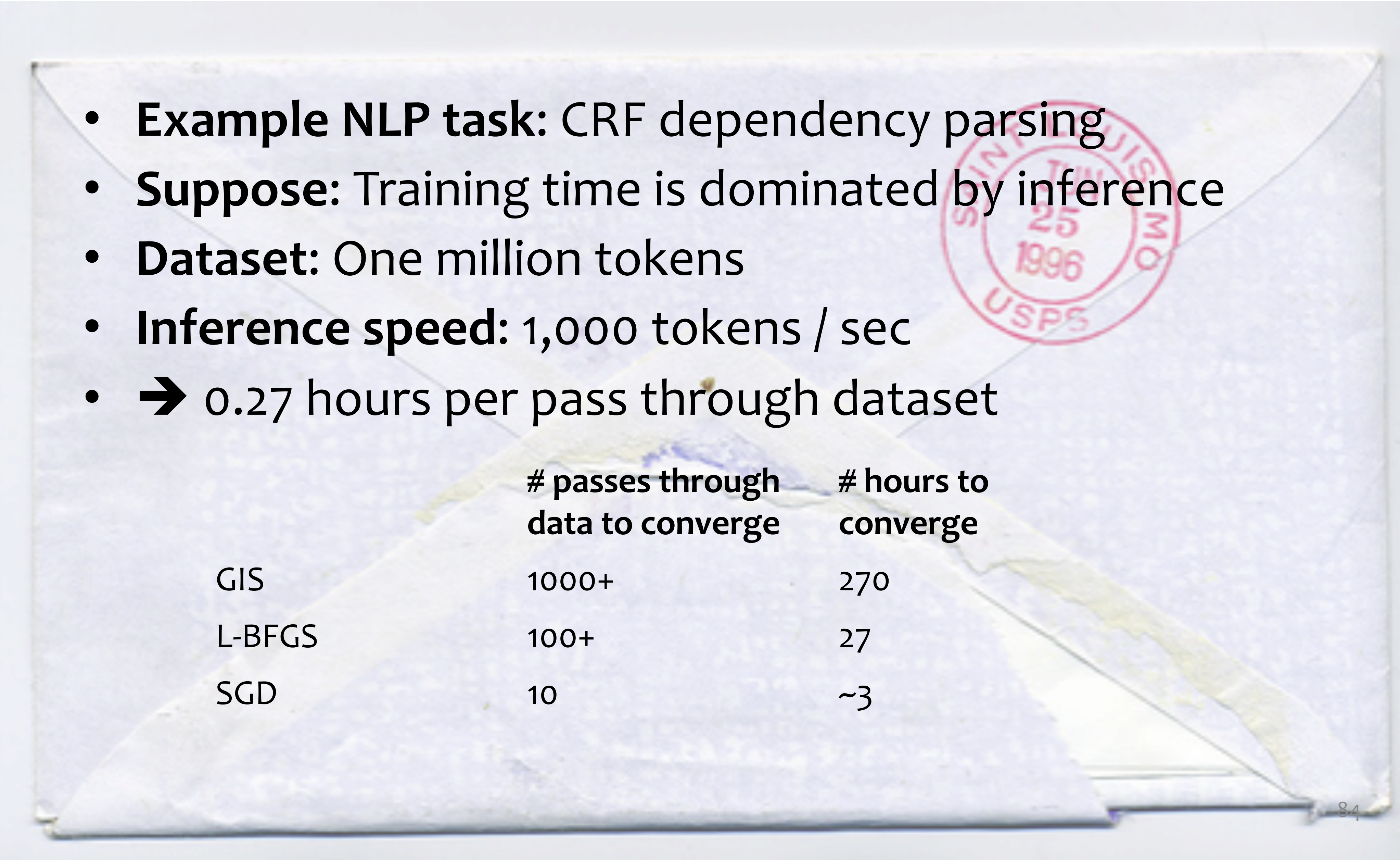
Whiteboard

- CRF model
- CRF data log-likelihood
- CRF derivatives

Practical Considerations for Gradient-based Methods

- Overfitting
  - L2 regularization
  - L1 regularization
  - Regularization by early stopping
- For SGD: Sparse updates

"Empirical" Comparison of Parameter Estimation Methods



- **Example NLP task:** CRF dependency parsing
- **Suppose:** Training time is dominated by inference
- **Dataset:** One million tokens
- **Inference speed:** 1,000 tokens / sec
- ➔ 0.27 hours per pass through dataset

	# passes through data to converge	**# hours to converge**
GIS	1000+	270
L-BFGS	100+	27
SGD	10	~3

Exact inference for tree-structured factor graphs

BELIEF PROPAGATION

Inference for **HMMs**

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

Inference for CRFs

- Sum-product BP on a CRF is called the **forward-backward algorithm**
- Max-product BP on a CRF is called the **Viterbi algorithm**

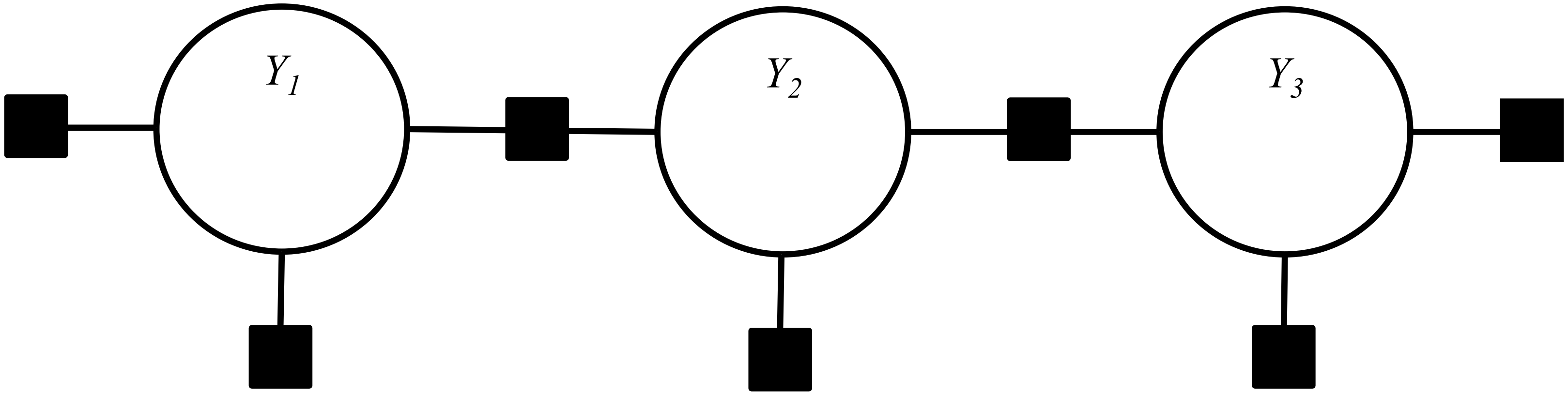
THE FORWARD-BACKWARD ALGORITHM

Learning and Inference Summary

For discrete variables:

	Learning	Marginal Inference	MAP Inference
HMM		Forward-backward	Viterbi
Linear-chain CRF		Forward-backward	Viterbi

Forward-Backward Algorithm



find

Could be verb or noun

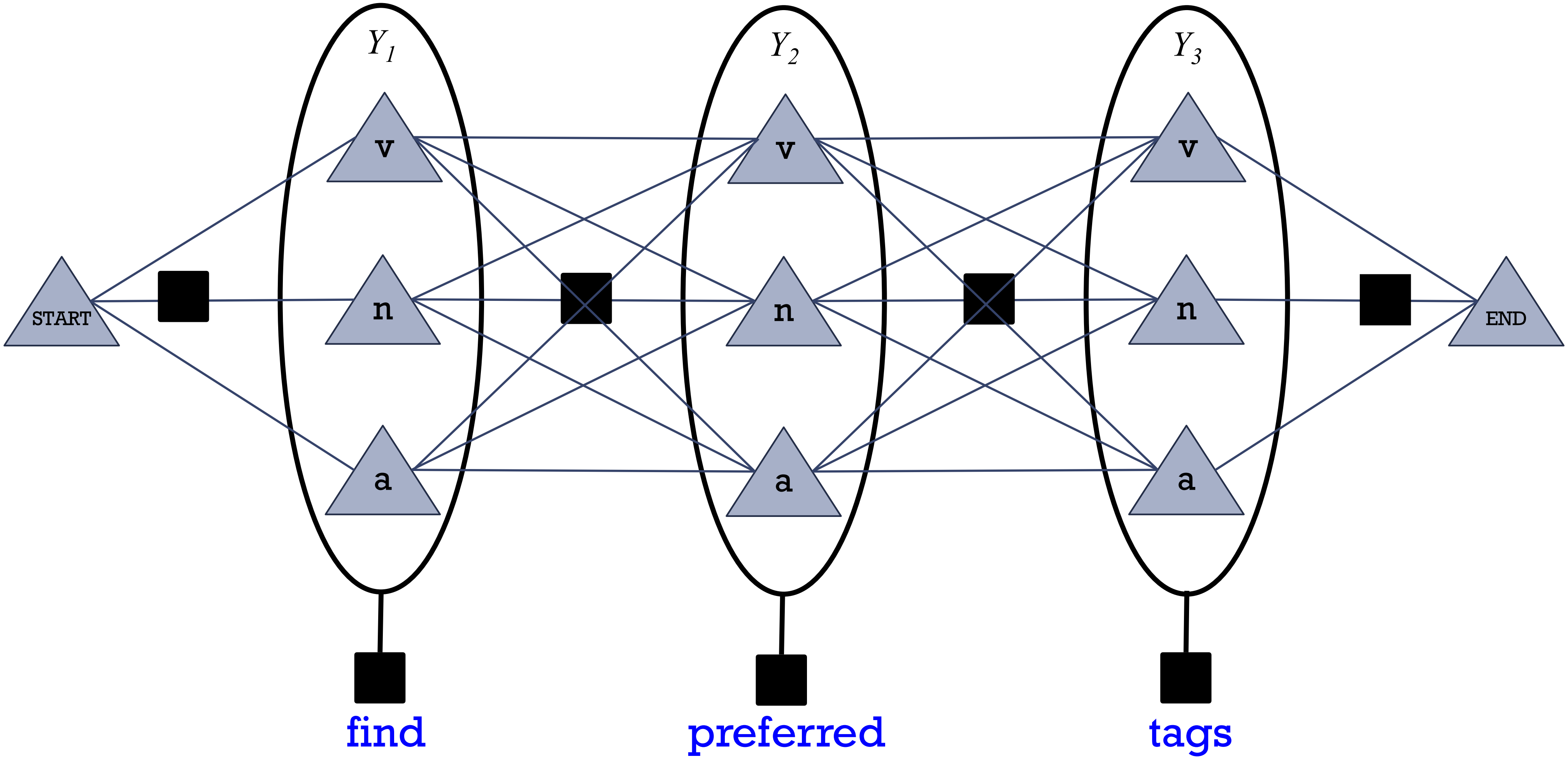
preferred

Could be adjective or verb

tags

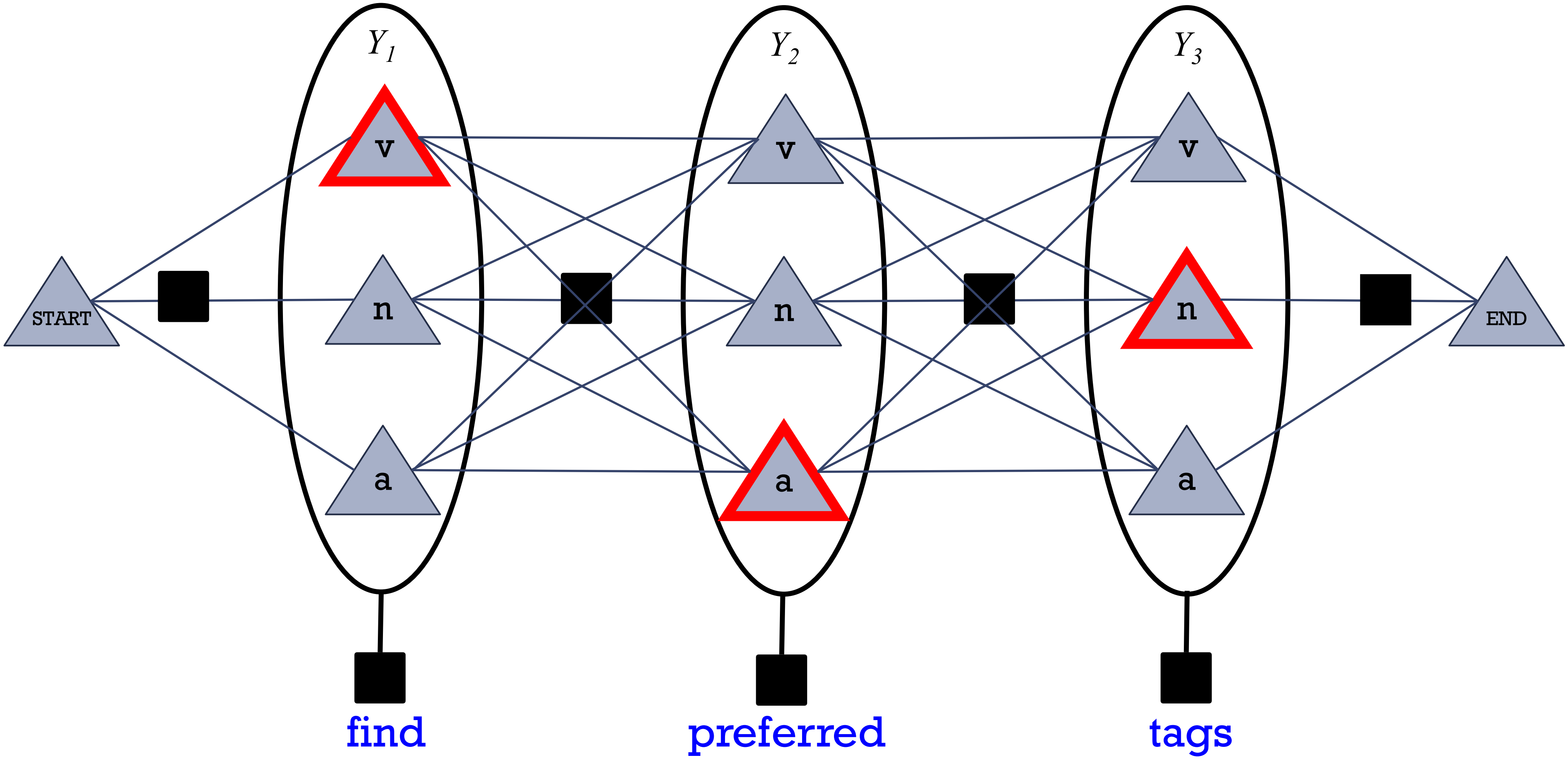
Could be noun or verb

Forward-Backward Algorithm



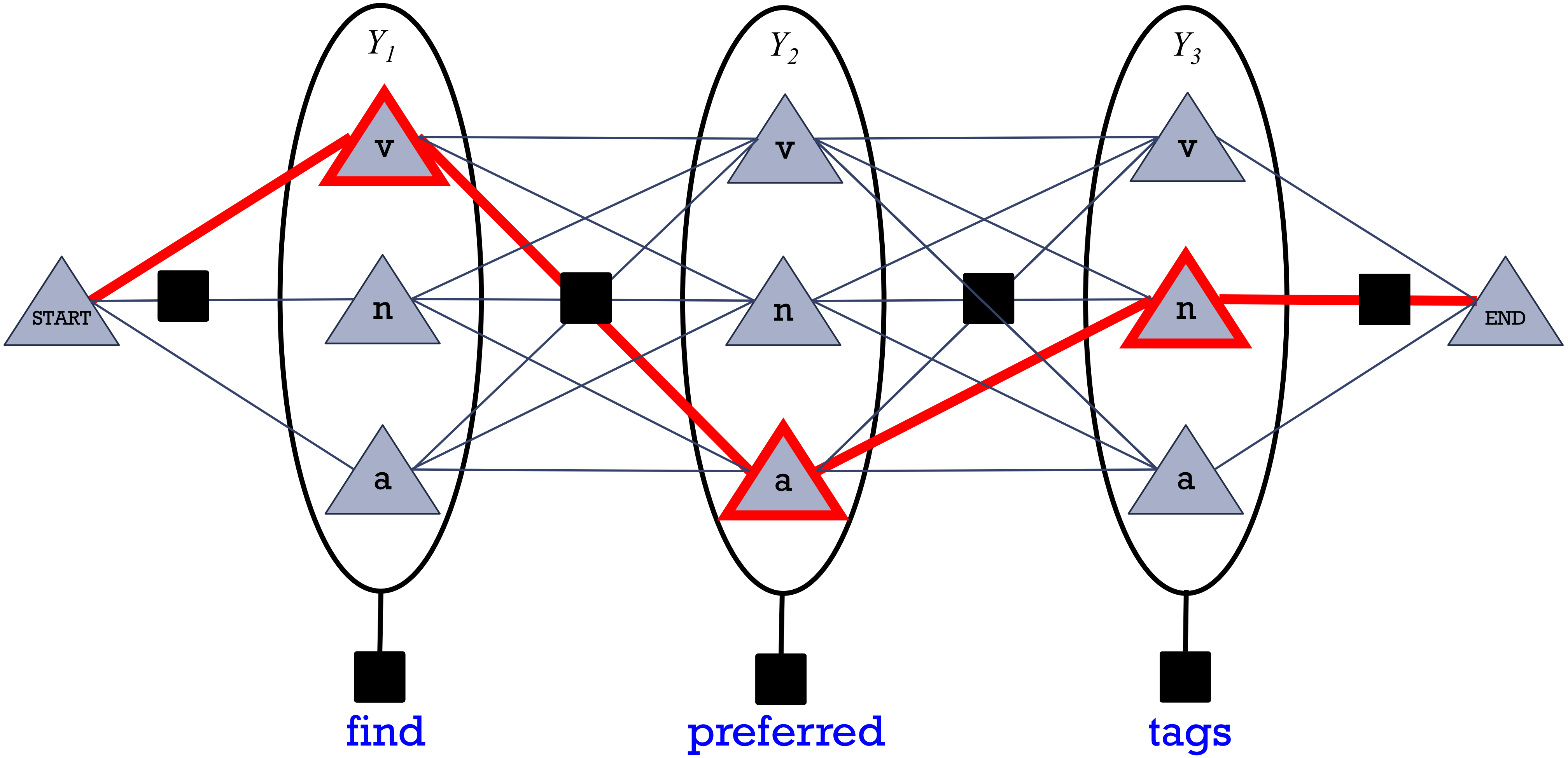
- Show the possible *values* for each variable

Forward-Backward Algorithm



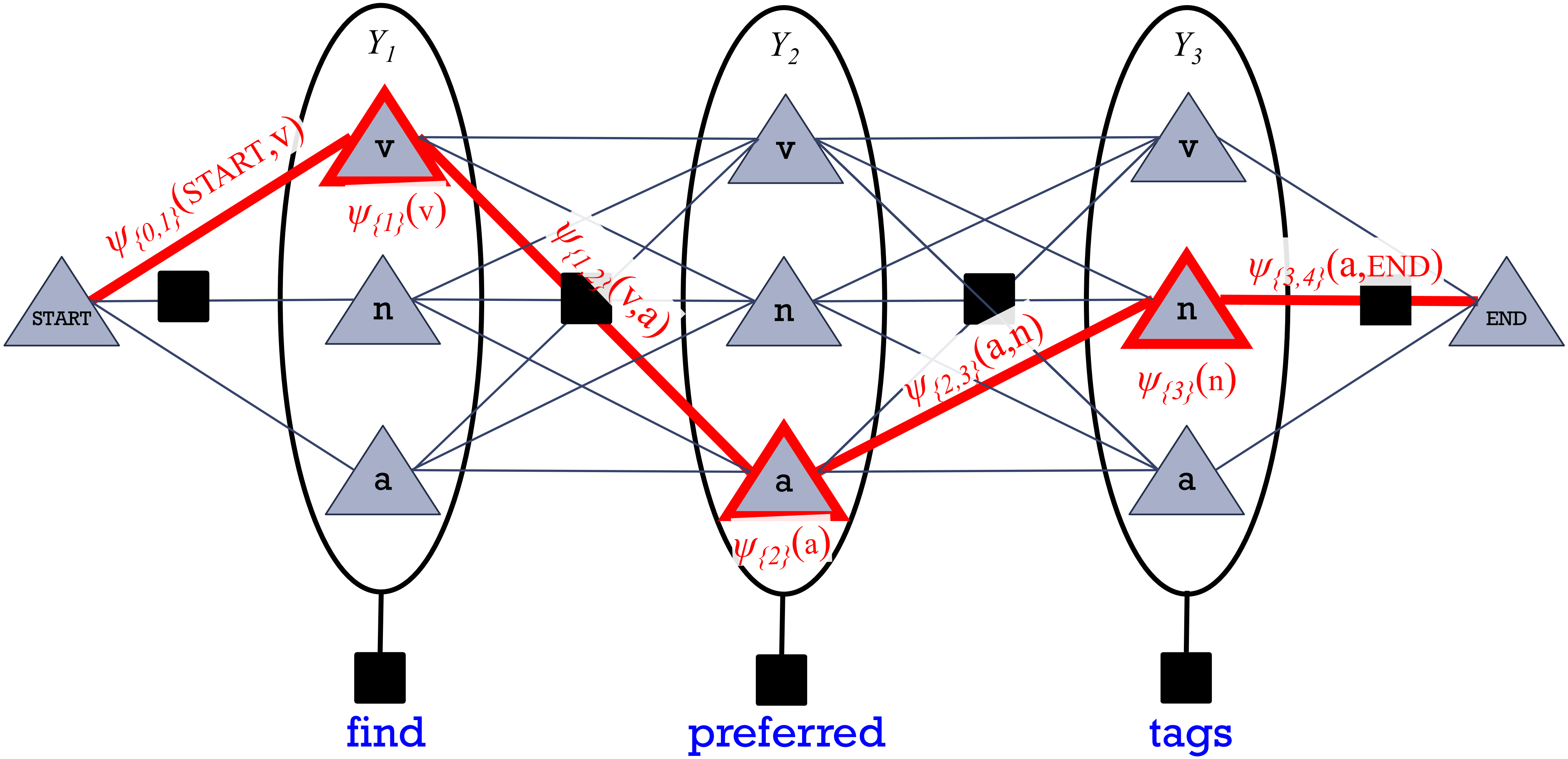
- Let’s show the possible *values* for each variable
- One possible assignment

Forward-Backward Algorithm



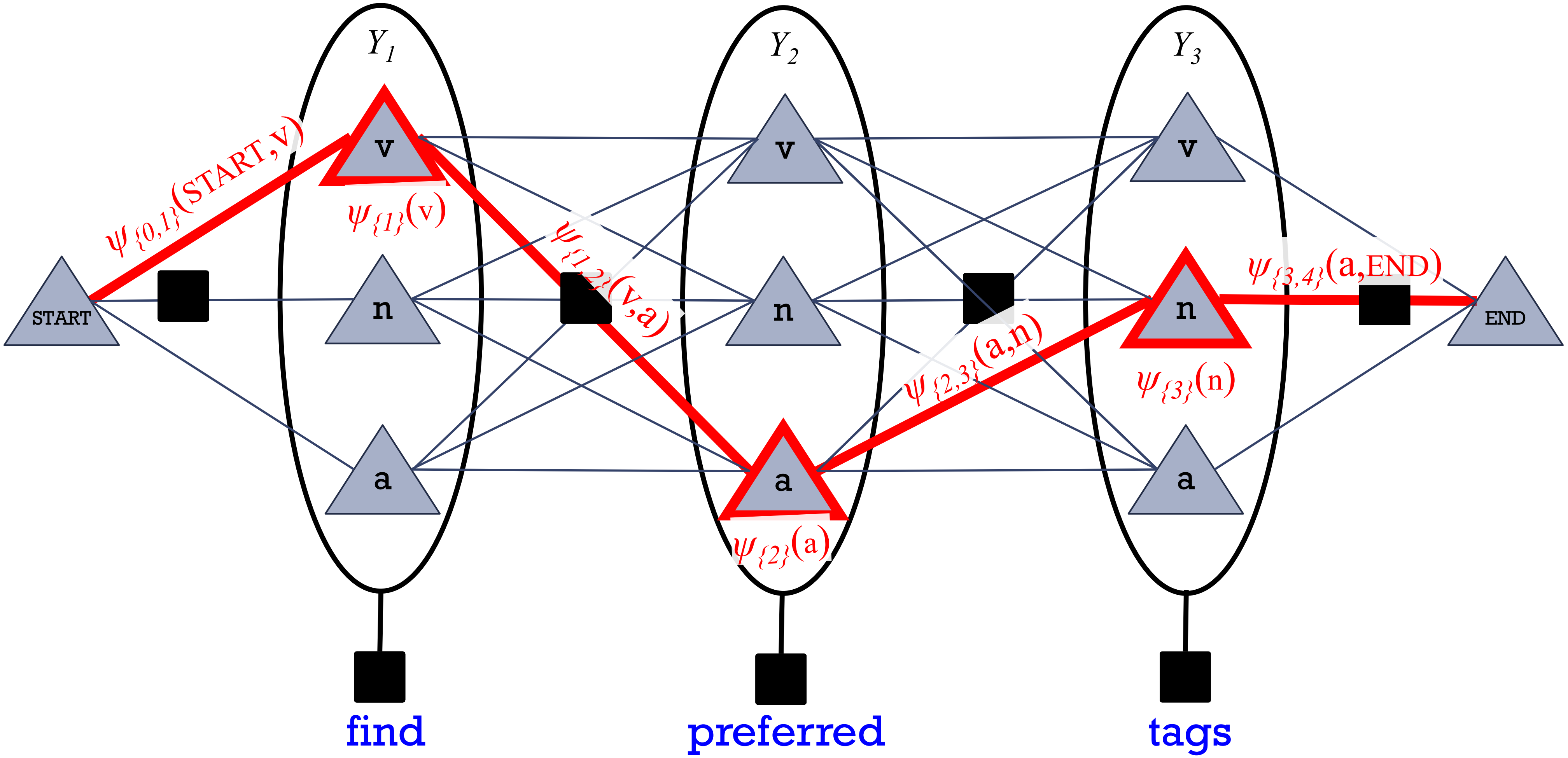
- Let’s show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



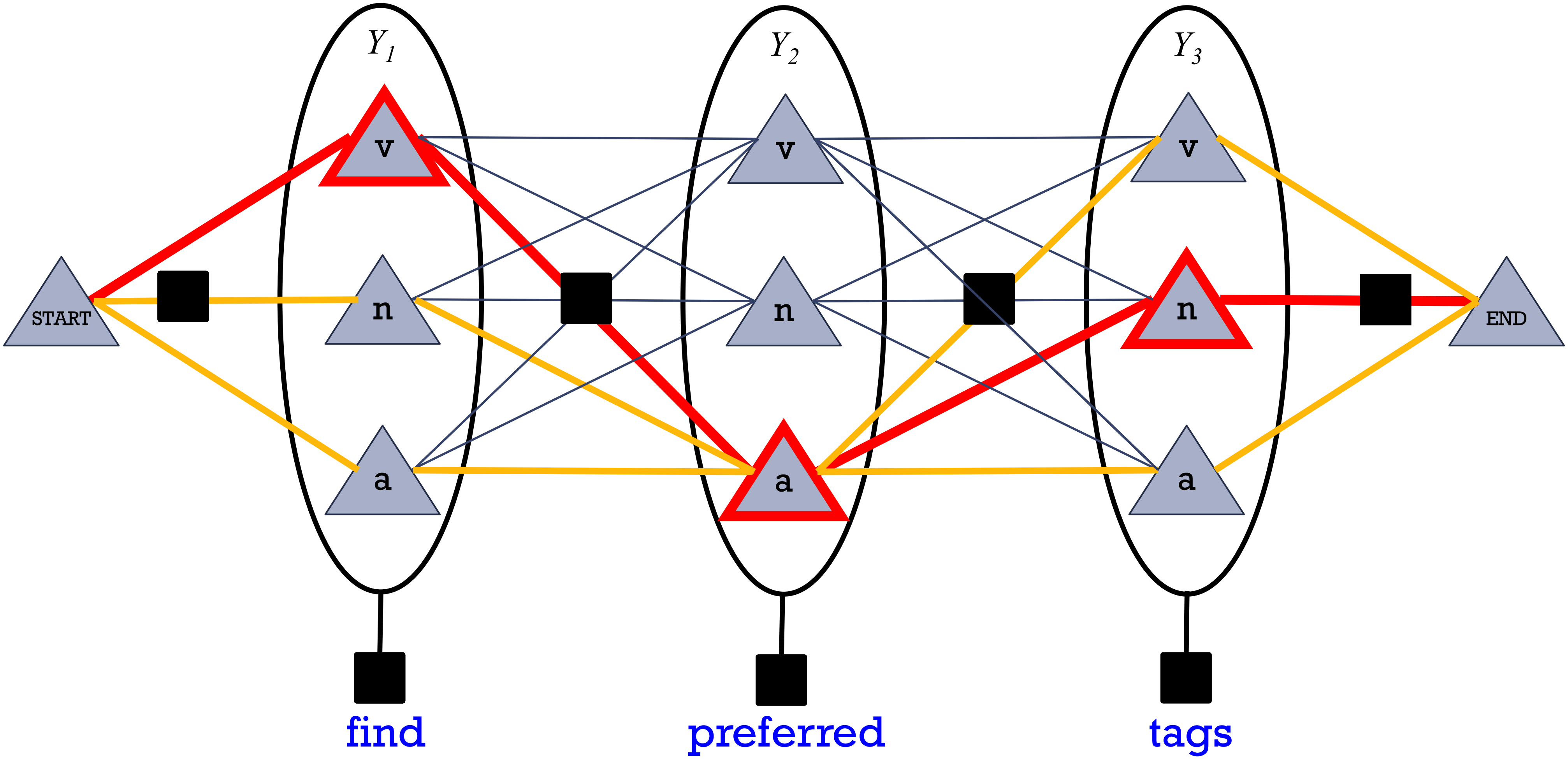
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment



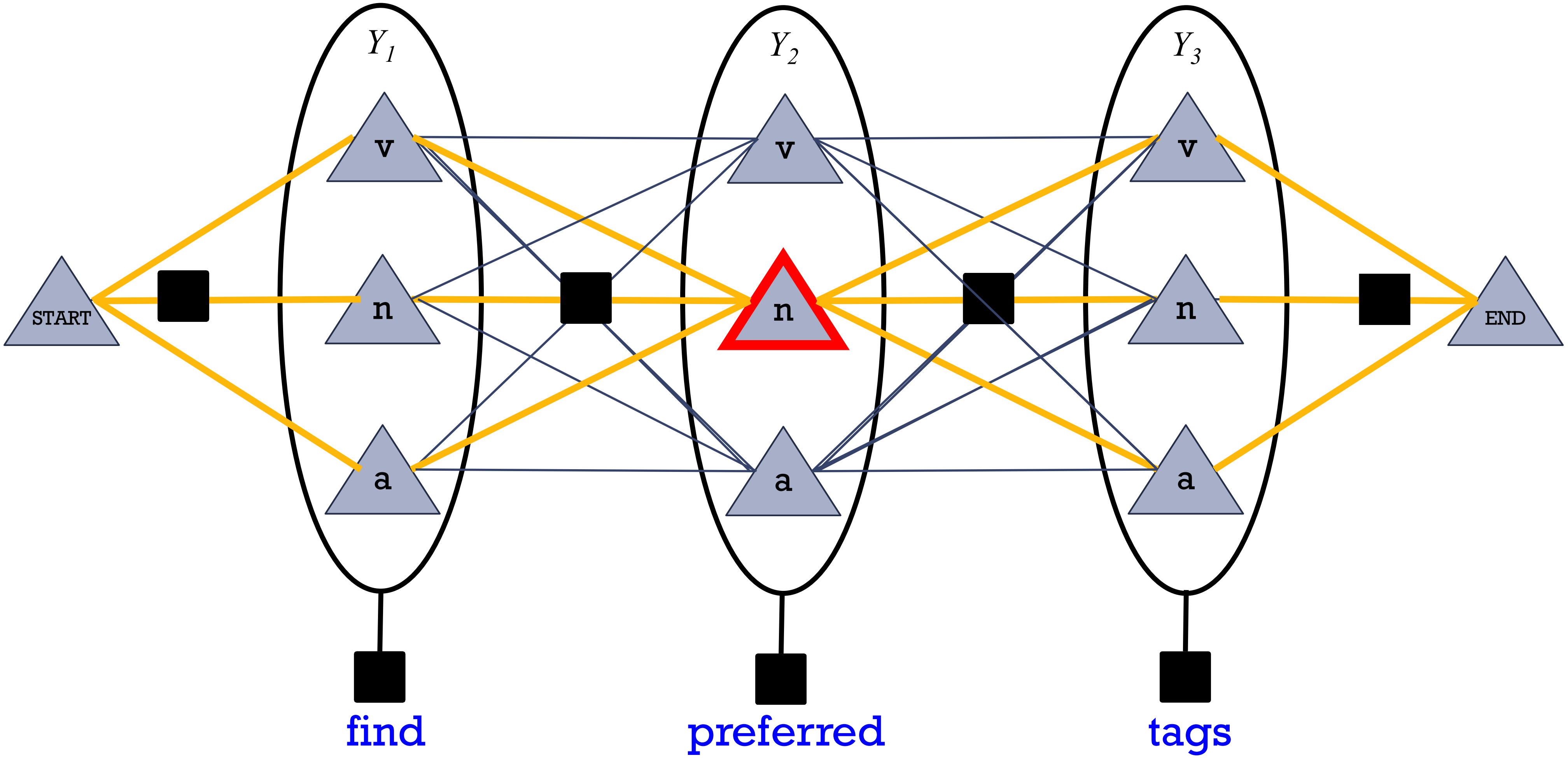
- So p(**v a n**) = (1/Z) * product weight of one path

Forward-Backward Algorithm: Finds Marginals



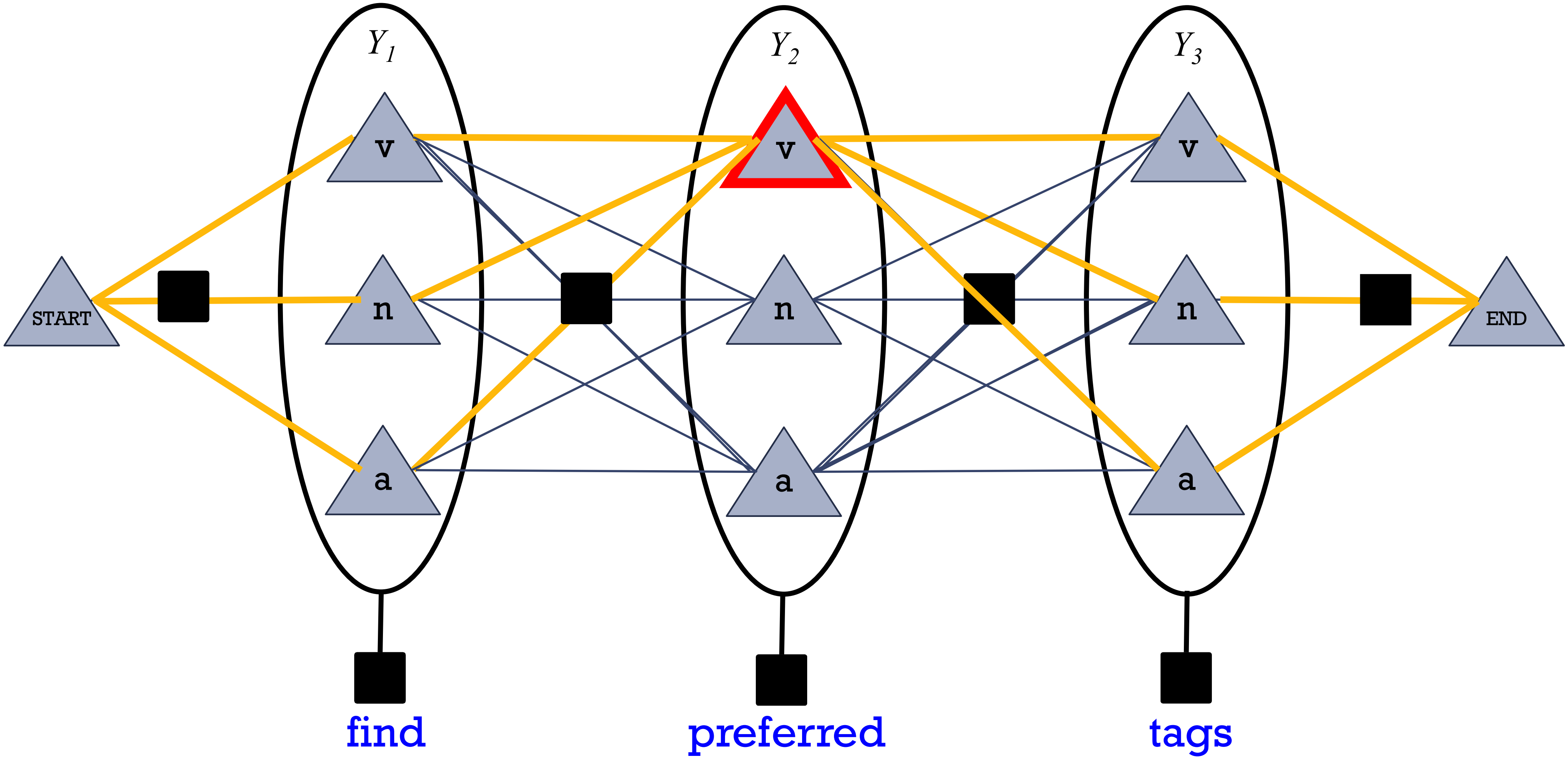
- So $p(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path
- Marginal probability $p(Y_2 = \text{a})$
  $= (1/Z)$ * total weight of *all* paths through a

Forward-Backward Algorithm: Finds Marginals



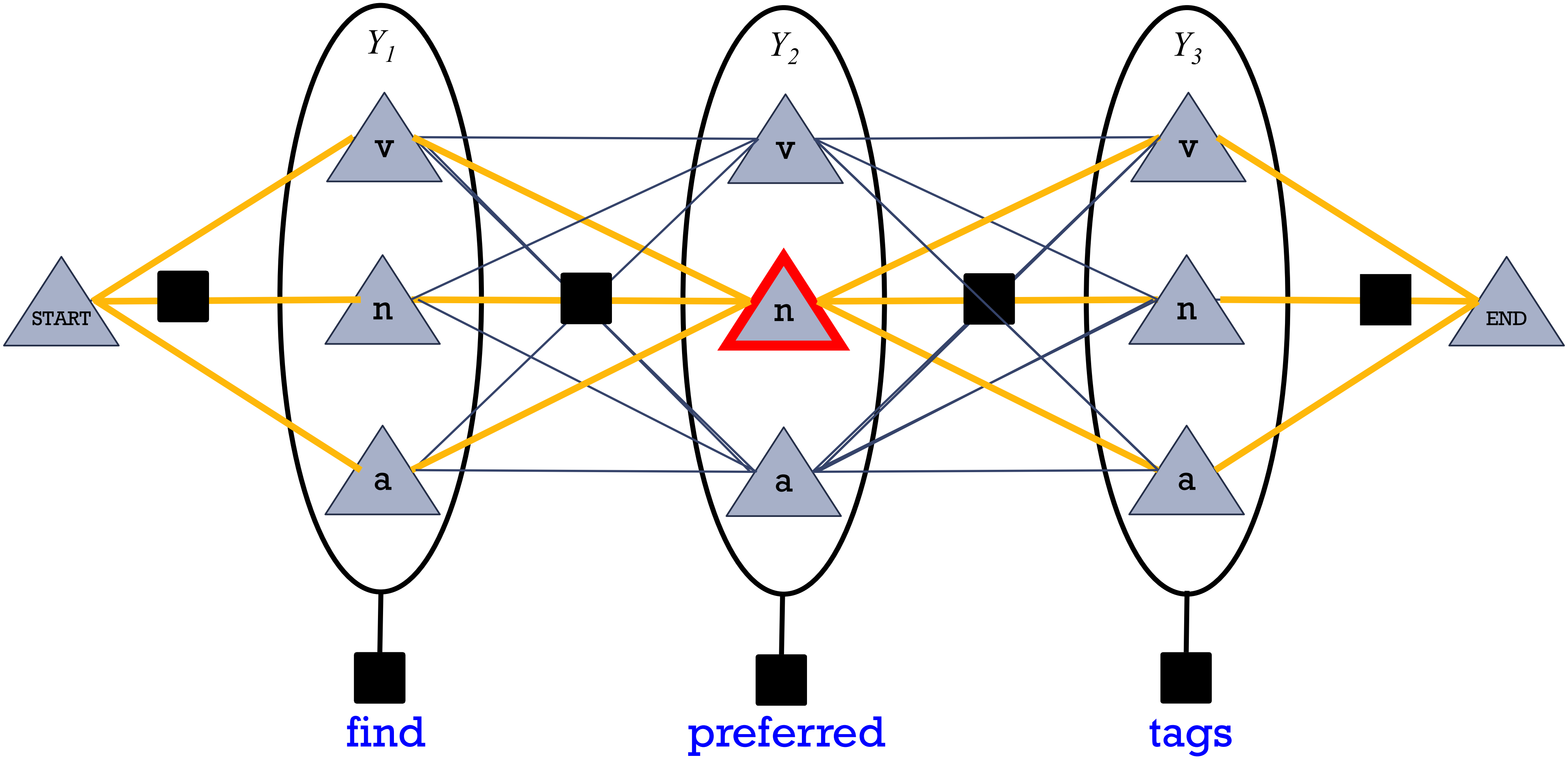
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = \mathrm{a})$
  = (1/Z) * total weight of *all* paths through n

Forward-Backward Algorithm: Finds Marginals



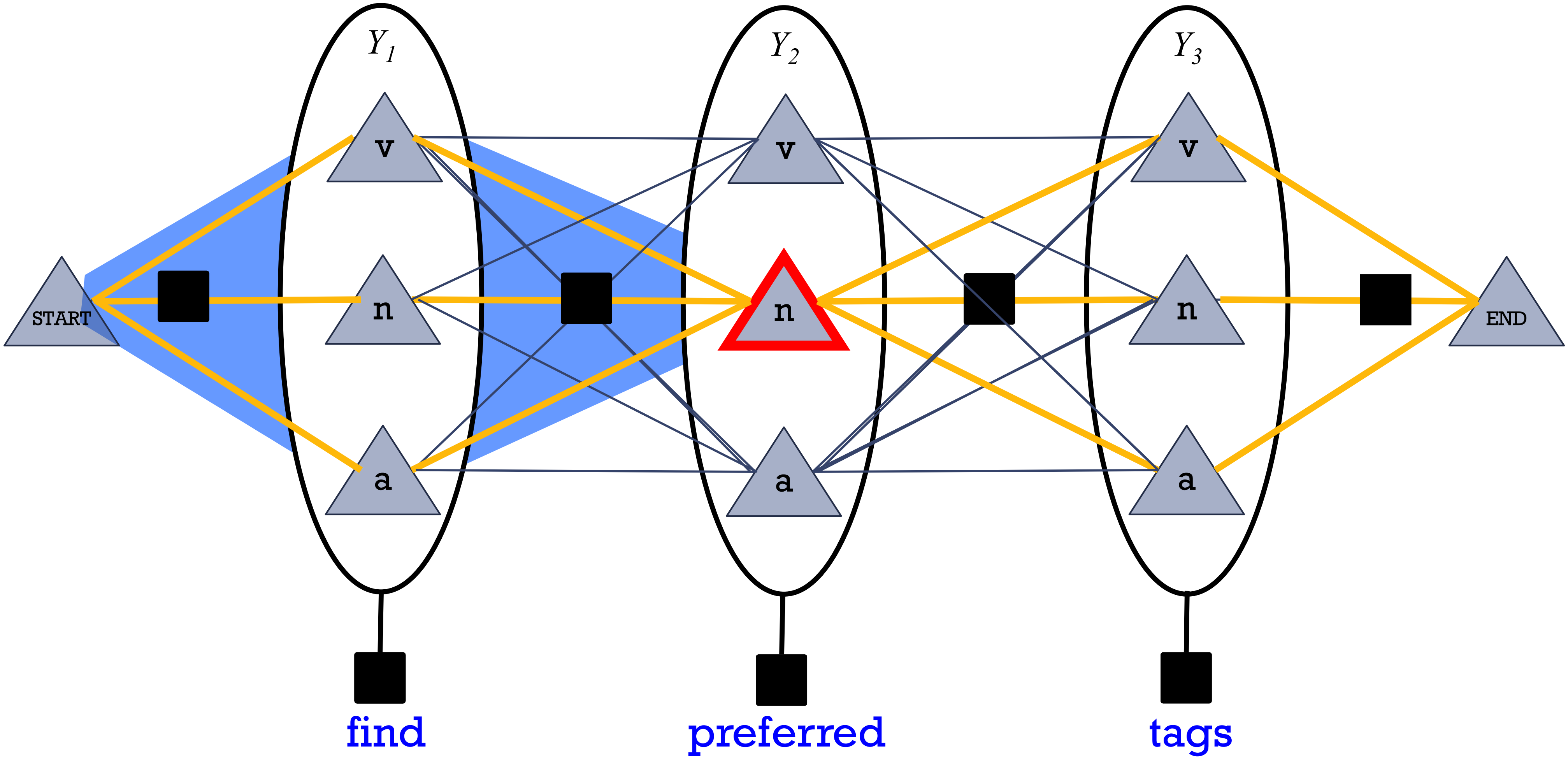
- So $p(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path
- Marginal probability $p(Y_2 = \mathrm{a})$
  $= (1/Z)$ * total weight of *all* paths through △v

Forward-Backward Algorithm: Finds Marginals



- So $\mathrm{p}(\mathbf{v\ a\ n}) = (1/Z)$ * product weight of one path
- Marginal probability $\mathrm{p}(Y_2 = \mathrm{a})$
  $= (1/Z)$ * total weight of *all* paths through n

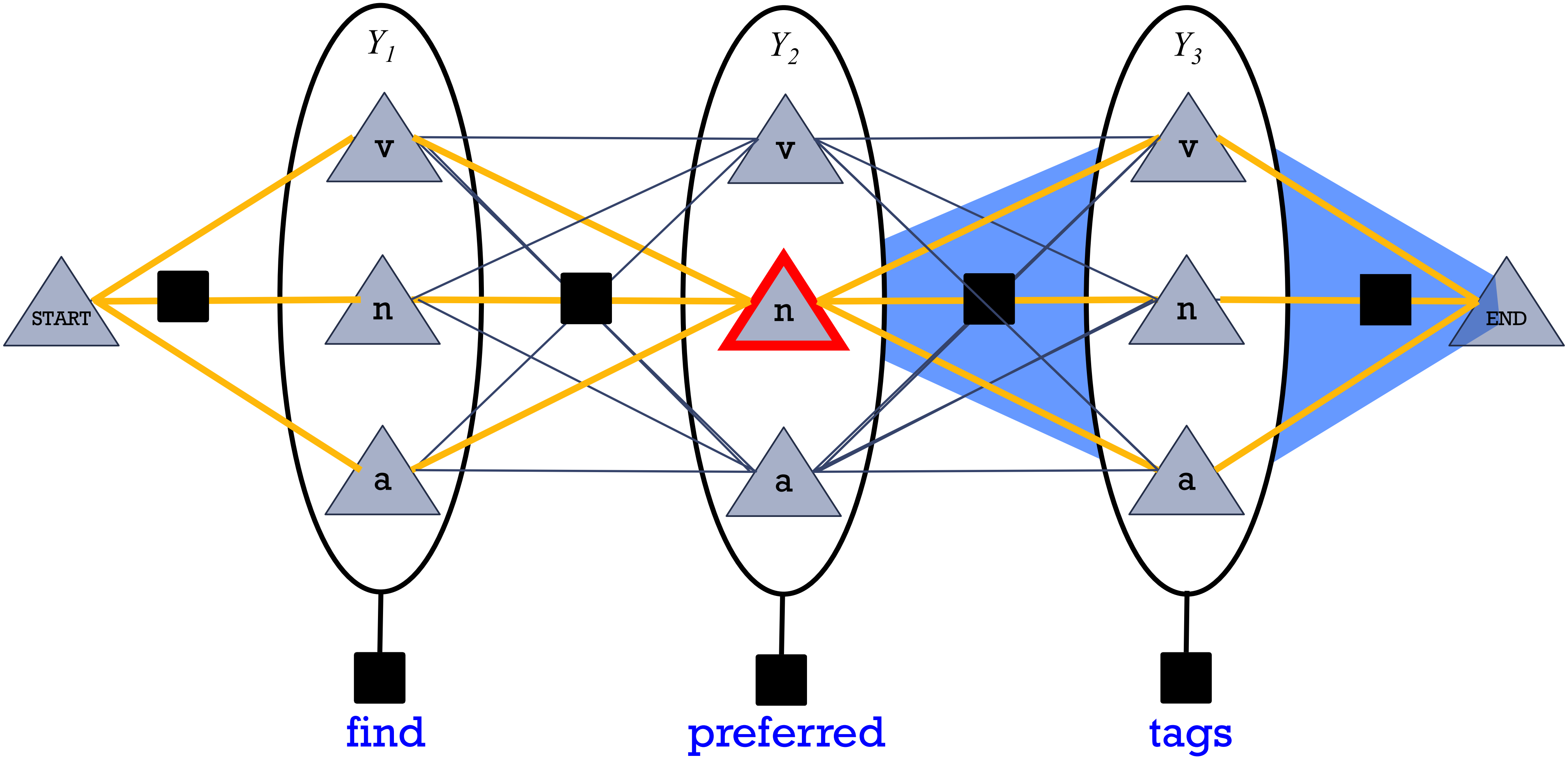
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

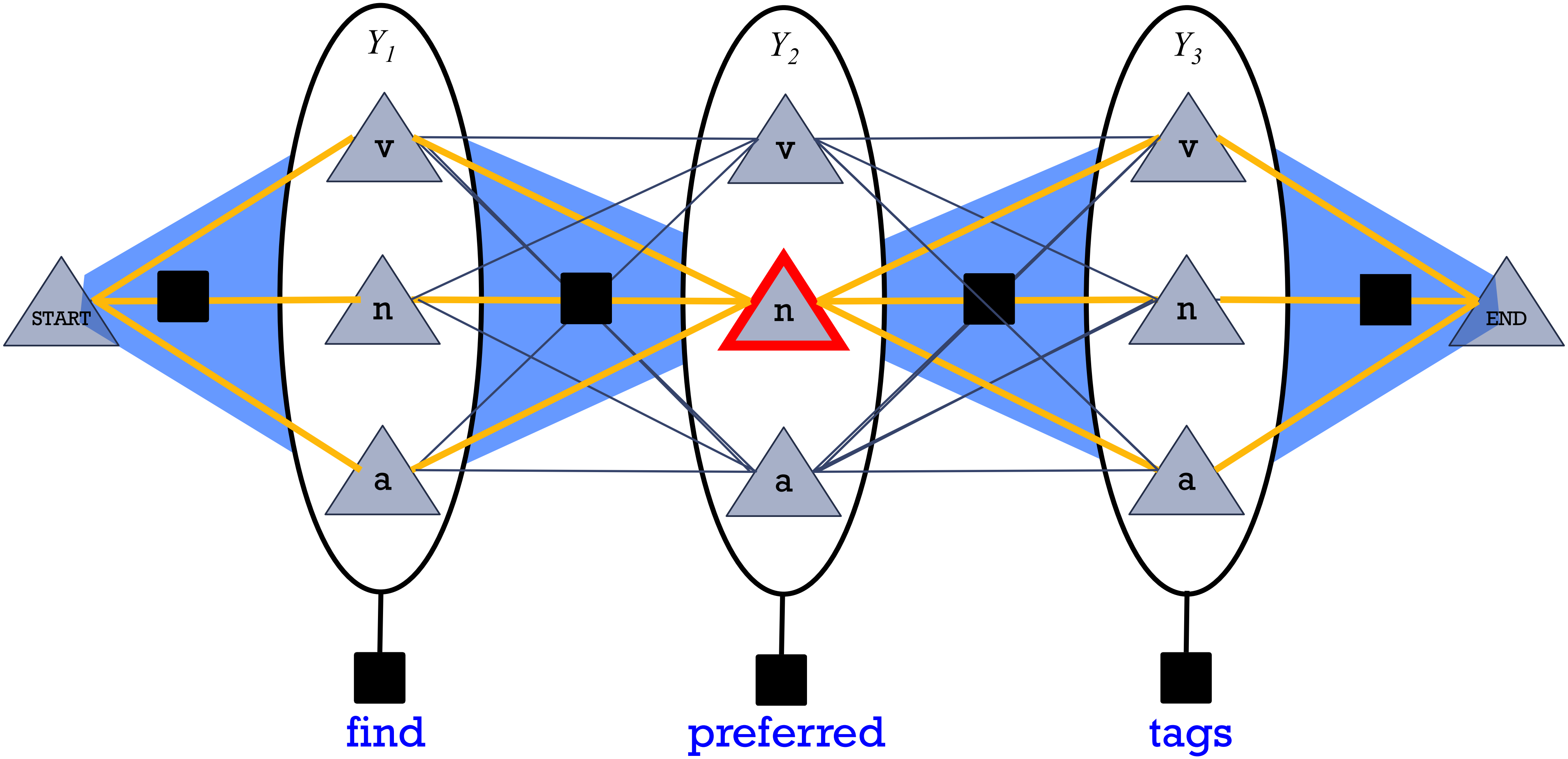
Forward-Backward Algorithm: Finds Marginals



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

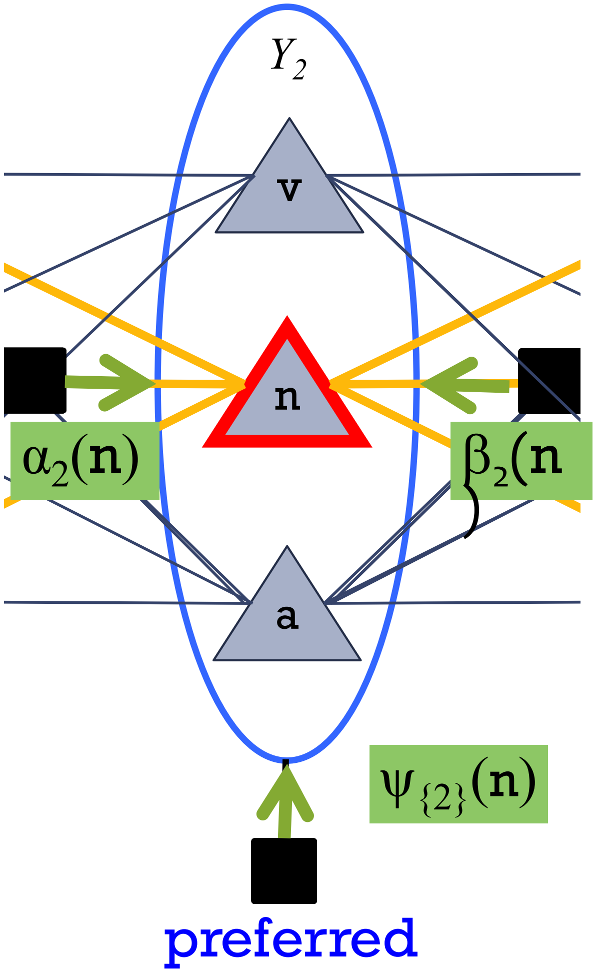
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

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

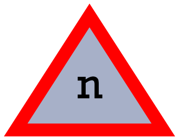
Forward-Backward Algorithm: Finds Marginals

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.

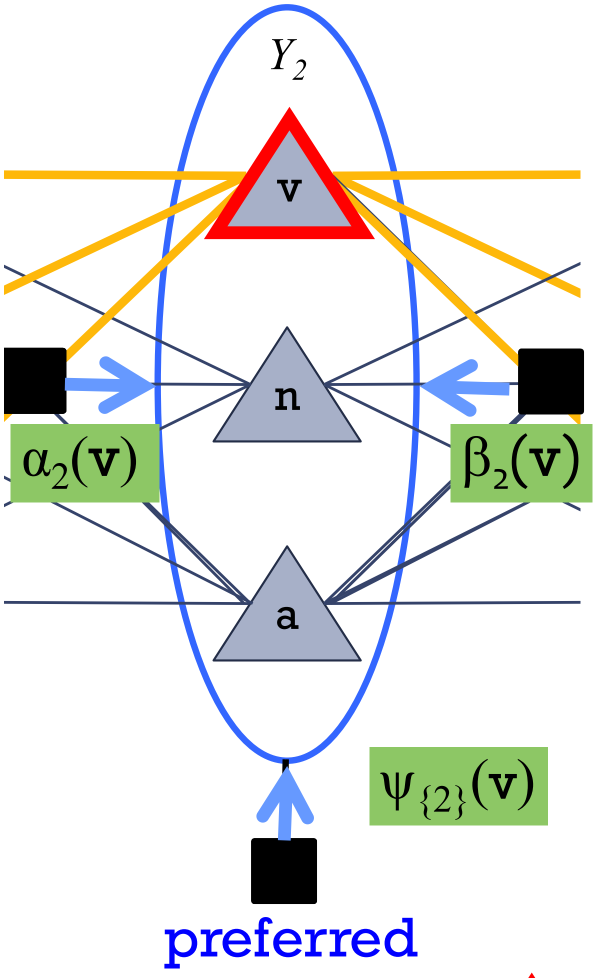


"belief that Y_2 = **n**"

total weight of *all* paths through 

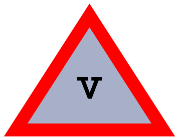
$= \alpha_2(\mathbf{n})\ \psi_{\{2\}}(\mathbf{n})\ \beta_2(\mathbf{n})$

Forward-Backward Algorithm: Finds Marginals



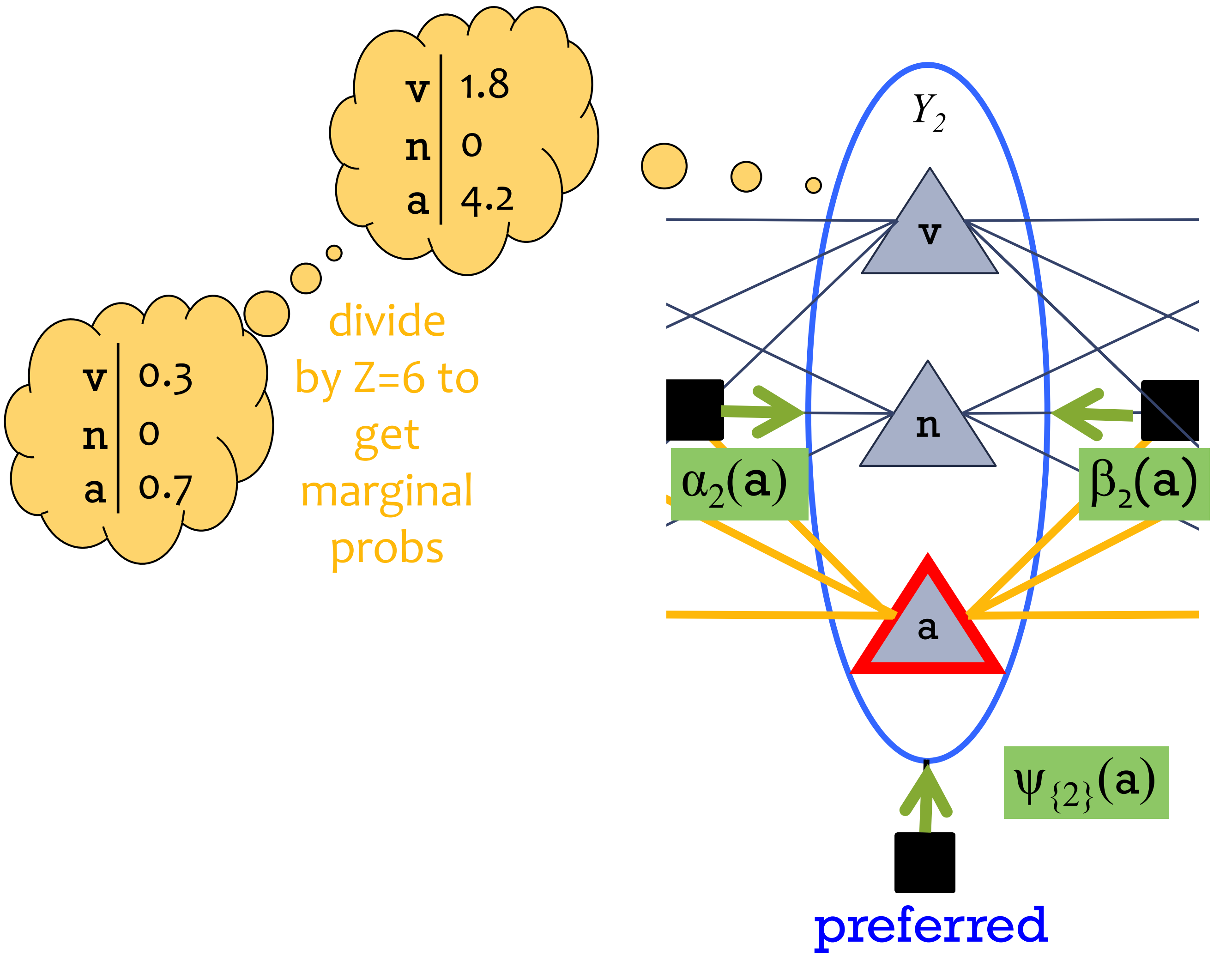
"belief that $Y_2 = \mathbf{v}$"

"belief that $Y_2 = \mathbf{n}$"

total weight of *all* paths through 

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

Forward-Backward Algorithm: Finds Marginals



v	1.8
n	0
a	4.2

v	0.3
n	0
a	0.7

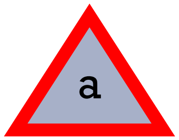
divide by Z=6 to get marginal probs

“belief that Y_2 = **v**”

“belief that Y_2 = **n**”

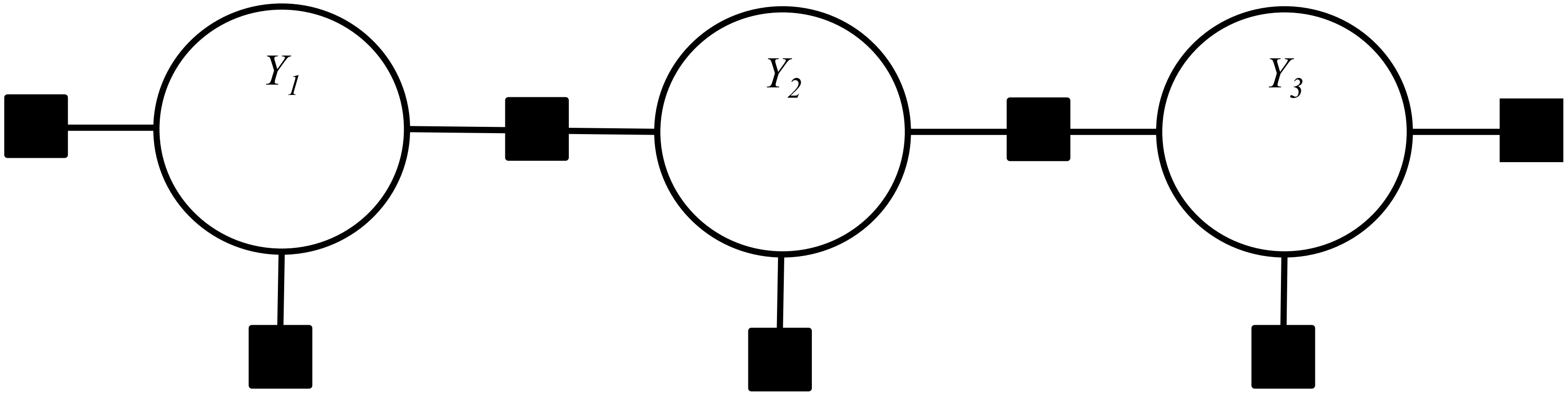
“belief that Y_2 = **a**”

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

total weight of *all* paths through 

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

CRF Tagging Model



find	preferred	tags
Could be verb or noun	*Could be adjective or verb*	*Could be noun or verb*

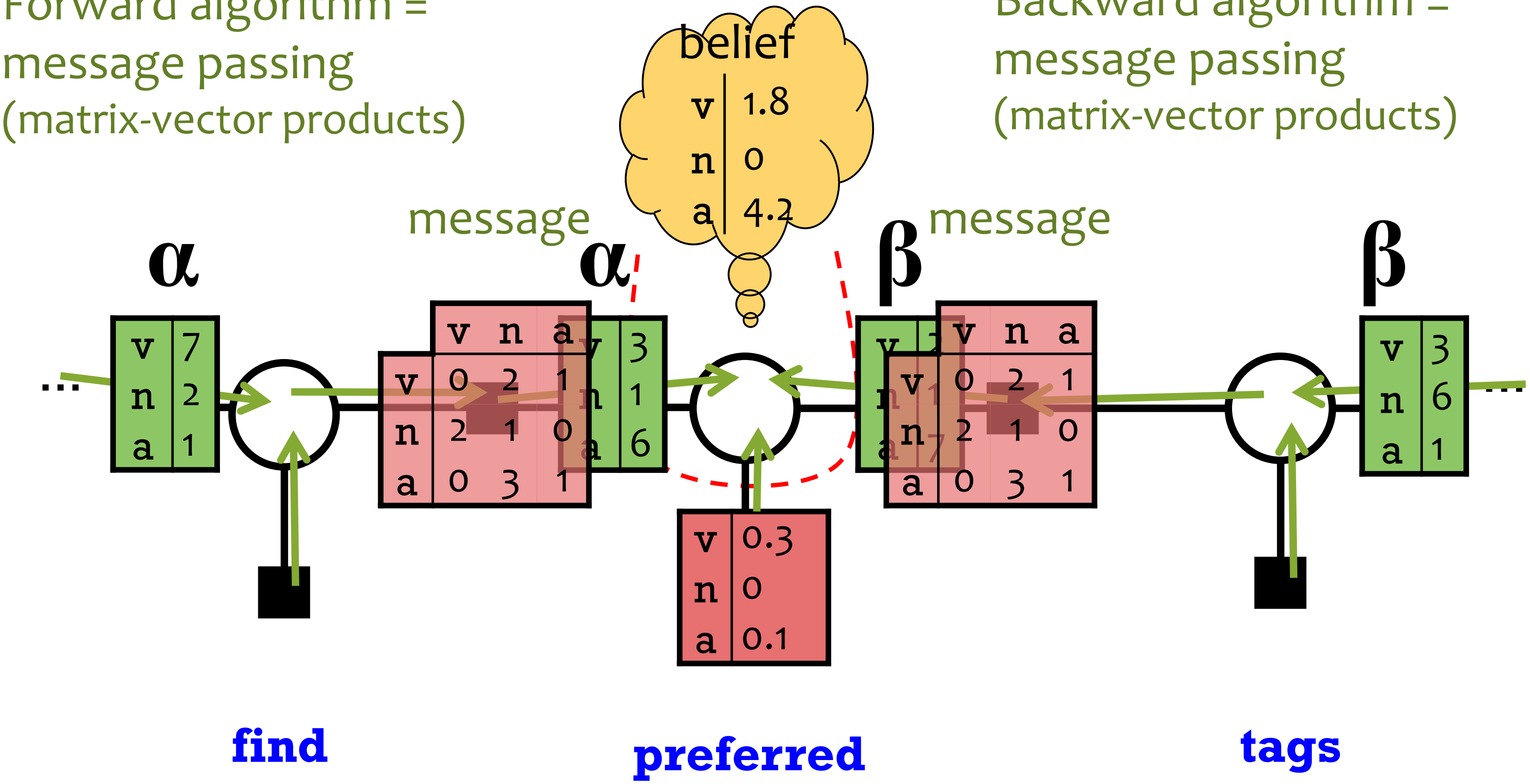
Whiteboard

- Forward-backward algorithm
- Viterbi algorithm

CRF Tagging by Belief Propagation

Forward algorithm = message passing (matrix-vector products)

Backward algorithm = message passing (matrix-vector products)



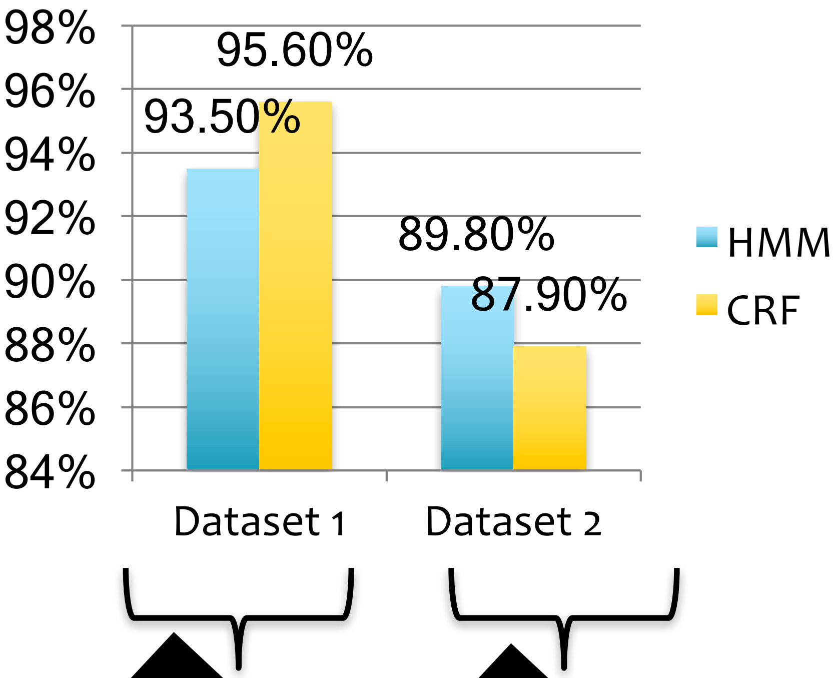
- Forward-backward is a message passing algorithm.
- It’s the simplest case of belief propagation.

HMMS VS CRFS

Generative vs. Discriminative

Liang & Jordan (ICML 2008) compares **HMM** and **CRF** with **identical features**

- Dataset 1: (Real)
  - WSJ Penn Treebank (38K train, 5.5K test)
  - 45 part-of-speech tags
- Dataset 2: (Artificial)
  - Synthetic data generated from HMM learned on Dataset 1 (1K train, 1K test)
- Evaluation Metric: Accuracy

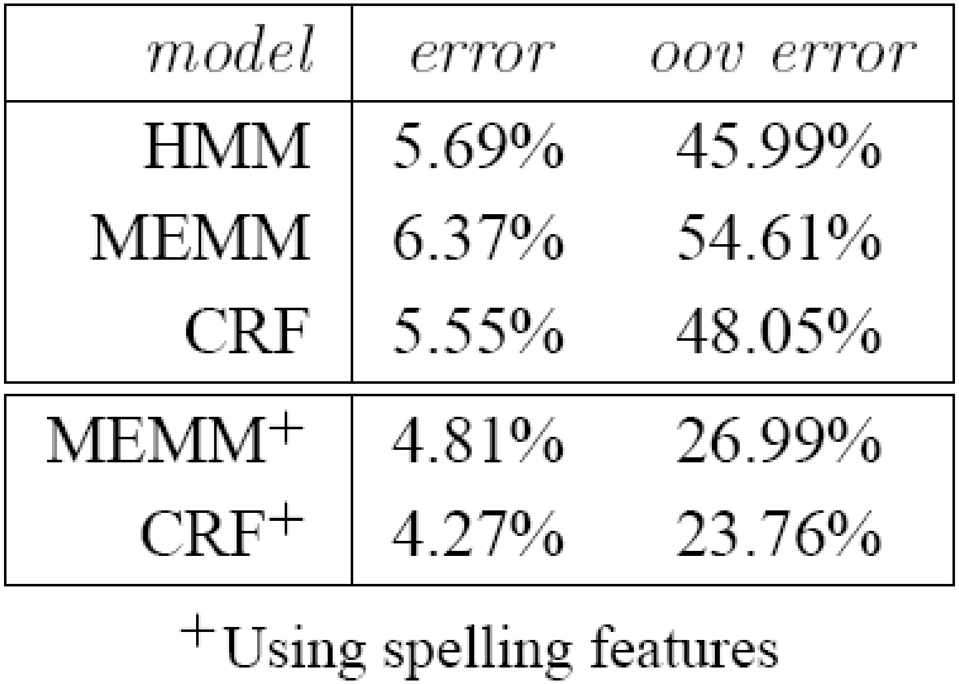


Model is misspecified

Model is well-specified

CRFs: some empirical results

- Parts of Speech tagging



model	*error*	*oov error*
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
MEMM$^+$	4.81%	26.99%
CRF$^+$	4.27%	23.76%

$^+$Using spelling features

  - Using same set of features: HMM >=< CRF > MEMM
  - Using additional overlapping features: CRF$^+$ > MEMM$^+$ >> HMM

© Eric Xing @ CMU, 2005-2015

SUMMARY

Summary: Learning and Inference

For discrete variables:

	Learning	Marginal Inference	MAP Inference
HMM	MLE by counting	Forward-backward	Viterbi
Linear-chain CRF	Gradient based – doesn't decompose because of $Z(\mathbf{x})$ and requires marginal inference	Forward-backward	Viterbi

Summary: Models

	Classification	Structured Prediction
Generative	Naïve Bayes	HMM
Discriminative	Logistic Regression	CRF