Log-Linear Models with Structured Outputs

Natural Language Processing
CS 6120—Spring 2014
Northeastern University

David Smith
(some slides from Andrew McCallum)

Overview

- Sequence labeling task (cf. POS tagging)
- Independent classifiers
- HMMs
- (Conditional) Maximum Entropy Markov Models
- Conditional Random Fields
- Beyond Sequence Labeling

Sequence Labeling

- Inputs: $x = (x_1, \ldots, x_n)$
- Labels: $y = (y_1, \ldots, y_n)$
- Typical goal: Given x, predict y

- Example sequence labeling tasks
  - Part-of-speech tagging
  - Named-entity-recognition (NER)
    - Label people, places, organizations

NER Example:

Red Sox and Their Fans Let Loose

Elise Amendola/Associated Press

Fans of the slugger David Ortiz in Boston's Copley Square.

By PETE THAMEL
Published: October 31, 2007

E-MAIL
PRINT
REPRINTS
SAVE

BOSTON, Oct. 30 — Jonathan Papelbon turned Boston's World Series victory parade into a full-scale dance party Tuesday as the Red Sox put an exclamation point on the 2007 season.

First Solution: Maximum Entropy Classifier

- Conditional model $p(y|x)$.
  - Do not waste effort modeling $p(x)$, since x is given at test time anyway.
  - Allows more complicated input features, since we do not need to model dependencies between them.
- Feature functions $f(x,y)$:
  - $f_1(x,y)$ = { word is Boston & y=Location }
  - $f_2(x,y)$ = { first letter capitalized & y=Name }
  - $f_3(x,y)$ = { x is an HTML link & y=Location}

First Solution: MaxEnt Classifier

- How should we choose a classifier?

- Principle of maximum entropy
  - We want a classifier that:
    - Matches feature constraints from training data.
    - Predictions maximize entropy.

- There is a unique, exponential family distribution that meets these criteria.

First Solution: MaxEnt Classifier

- Problem with using a maximum entropy classifier for sequence labeling:
- It makes decisions at each position independently!

Second Solution: HMM

$$P(\mathbf{y}, \mathbf{x}) = \prod_t P(y_t \mid y_{t-1}) P(x \mid y_t)$$

- Defines a generative process.
- Can be viewed as a weighted finite state machine.

Second Solution: HMM

- How can represent we multiple features in an HMM?
  - Treat them as conditionally independent given the class label?
    - The example features we talked about are not independent.
  - Try to model a more complex generative process of the input features?
    - We may lose tractability (i.e. lose a dynamic programming for exact inference).

Second Solution: HMM

- Let’s use a conditional model instead.

Third Solution: MEMM

- Use a series of maximum entropy classifiers that know the previous label.
- Define a Viterbi algorithm for inference.

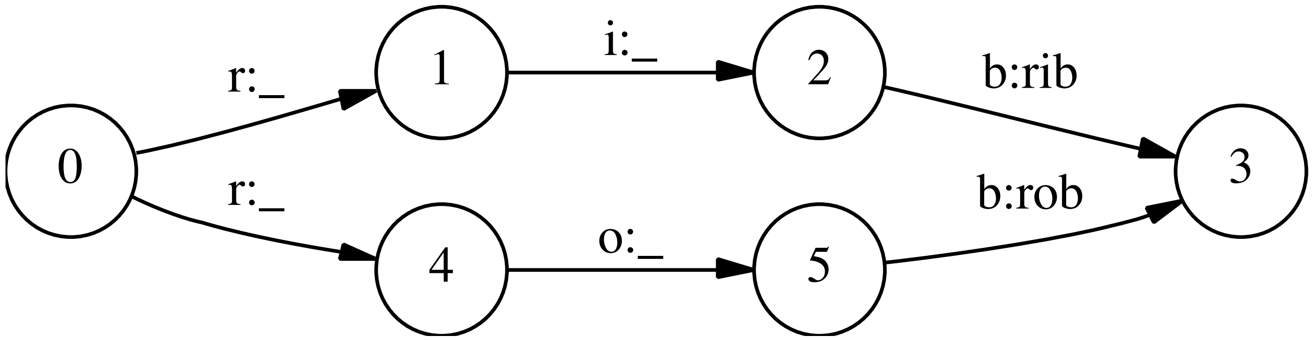
$$P(\mathbf{y} \mid \mathbf{x}) = \prod_t P_{y_{t-1}}(y_t \mid \mathbf{x})$$

Third Solution: MEMM

- Combines the advantages of maximum entropy and HMM!
- But there is a problem…

Problem with MEMMs: Label Bias

- In some state space configurations, MEMMs essentially completely ignore the inputs.



- This is not a problem for HMMs, because the input sequence is generated by the model.

Fourth Solution: Conditional Random Field

- Conditionally-trained, undirected graphical model.
- For a standard linear-chain structure:

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_t \Psi_k(y_t, y_{t-1}, \mathbf{x})$$

$$\Psi_k(y_t, y_{t-1}, \mathbf{x}) = \exp\left(\sum_k \lambda_k f(y_t, y_{t-1}, \mathbf{x}) \right)$$

Fourth Solution: CRF

- Have the advantages of MEMMs, but avoid the label bias problem.
- CRFs are globally normalized, whereas MEMMs are locally normalized.
- Widely used and applied. CRFs give state-the-art results in many domains.

Fourth Solution: CRF

- Have the advantages of MEMMs, but avoid the label bias problem.

- CRFs are globally normalized, whereas MEMMs are locally normalized.

- Widely used and applie
state-the-art results in n

Remember, *Z* is the normalization constant. How do we compute it?

CRF Applications

- Part-of-speech tagging
- Named entity recognition
- Document layout (e.g. table) classification
- Gene prediction
- Chinese word segmentation
- Morphological disambiguation
- Citation parsing
- Etc., etc.

NER as Sequence Tagging

The Phoenicians came from the Red Sea

NER as Sequence Tagging

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Capitalized word

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Capitalized word

Ends in “s”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Capitalized word

Ends in “s”

Ends in “ans”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Capitalized word

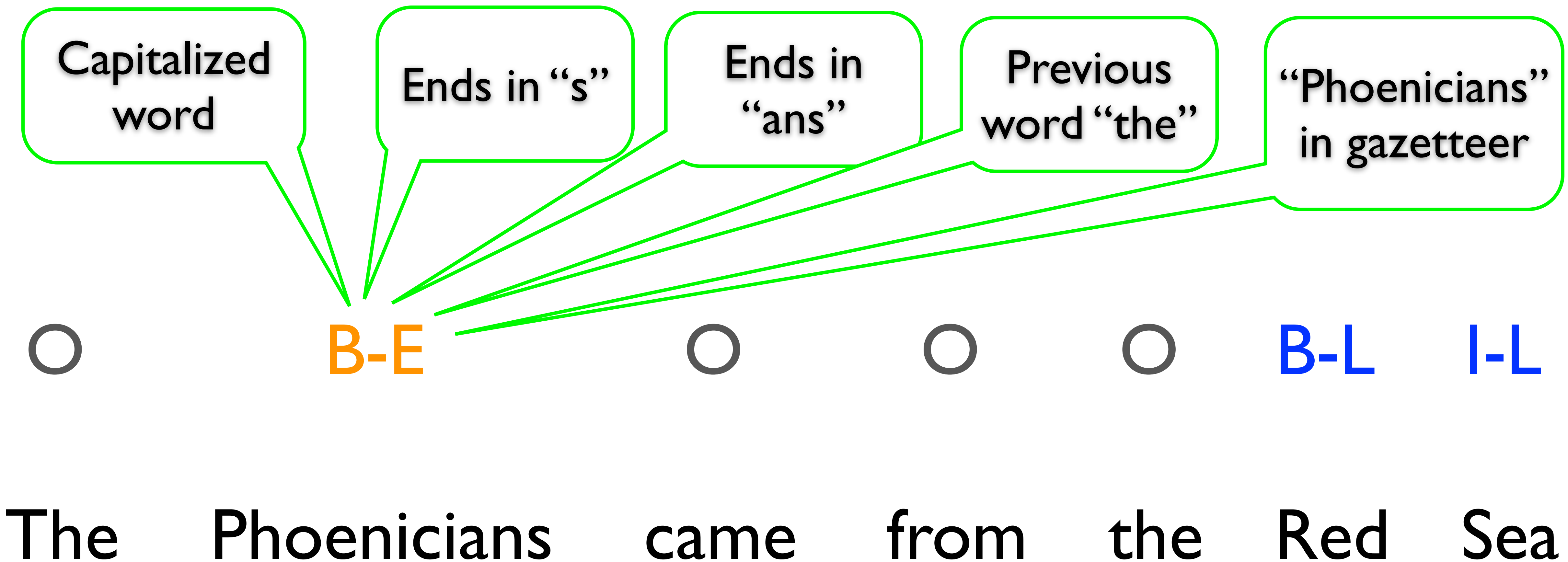
Ends in “s”

Ends in “ans”

Previous word “the”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging



NER as Sequence Tagging

The	Phoenicians	came	from	the	Red	Sea
O	B-E	O	O	O	B-L	I-L

NER as Sequence Tagging

Not capitalized

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

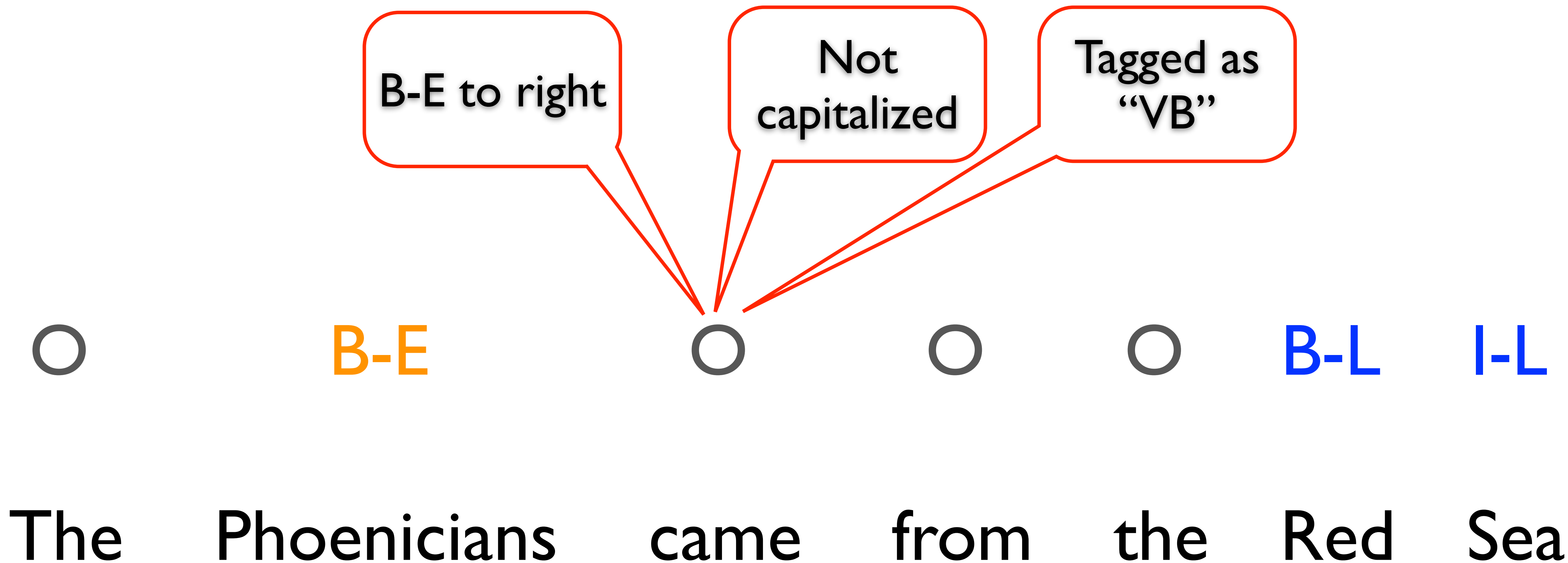
NER as Sequence Tagging

Not capitalized

Tagged as “VB”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging



NER as Sequence Tagging

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Word “sea”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Word “sea” preceded by “the ADJ”

Word “sea”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

NER as Sequence Tagging

Hard constraint: I-L must follow B-L or I-L

Word “sea” preceded by “the ADJ”

Word “sea”

O	B-E	O	O	O	B-L	I-L
The	Phoenicians	came	from	the	Red	Sea

Overview

- What computations do we need?
- Smoothing log-linear models
- MEMMs vs. CRFs again
  - Action-based parsing and dependency parsing

Recipe for Conditional Training of p(y | x)

1. Gather constraints/features from training data

$$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero
3. Classify training data with current parameters; calculate expectations $$E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y'|x_j) f_{iy}(x_j, y')$$
4. Gradient is $\tilde{E}[f_{iy}] - E_\Theta[f_{iy}]$
5. Take a step in the direction of the gradient
6. Repeat from 3 until convergence

Recipe for Conditional Training of p(y | x)

1. Gather constraints/features from training data

$$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero
3. Classify training data with current parameters; calculate expectations $$E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y'|x_j) f_{iy}(x_j, y')$$
4. Gradient is $\tilde{E}[f_{iy}] - E_\Theta[f_{iy}]$
5. Take a step in the direction of the gradient
6. Repeat from 3 until convergence

Where have we seen expected counts before?

Recipe for Conditional Training of p(y | x)

1. Gather constraints/features from training data

$$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero
3. Classify training data with current parameters; calculate expectations $$E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y'|x_j) f_{iy}(x_j, y')$$
4. Gradient is $\tilde{E}[f_{iy}] - E_\Theta[f_{iy}]$

EM!

5. Take a step in the direction of the gradient
6. Repeat from 3 until convergence

Where have we seen expected counts before?

Gradient-Based Training

- λ := λ + rate * Gradient(F)
- After all training examples? (batch)
- After every example? (on-line)
- Use second derivative for faster learning?
- A big field: numerical optimization

Overfitting

- If we have too many features, we can choose weights to model the training data perfectly
- If we have a feature that only appears in spam training, not ham training, it will get weight ∞ to maximize p(spam | feature) at 1.
- These behaviors
  - Overfit the training data
  - Will probably do poorly on test data

Solutions to Overfitting

- Throw out rare features.
  - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.
- Only keep, e.g., 1000 features.
  - Add one at a time, always greedily picking the one that most improves performance on held-out data.
- Smooth the observed feature counts.
- Smooth the weights by using a prior.
  - $\max p(\lambda|\text{data}) = \max p(\lambda, \text{data}) = p(\lambda)p(\text{data}|\lambda)$
  - decree $p(\lambda)$ to be high when most weights close to 0

Smoothing with Priors

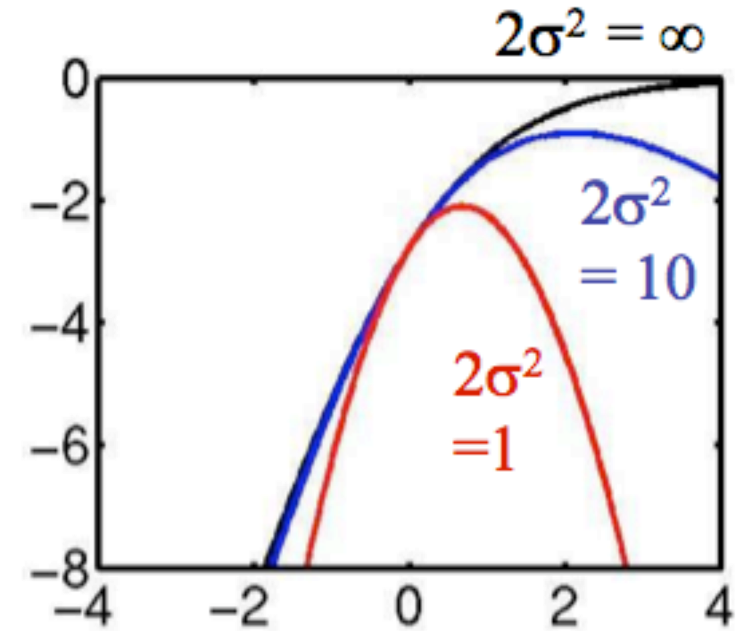
- What if we had a prior expectation that parameter values wouldn't be very large?
- We could then balance evidence suggesting large (or infinite) parameters against our prior expectation.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite)
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

  $\log P(y, \lambda \mid x) = \log P(\lambda) + \log P(y \mid x, \lambda)$

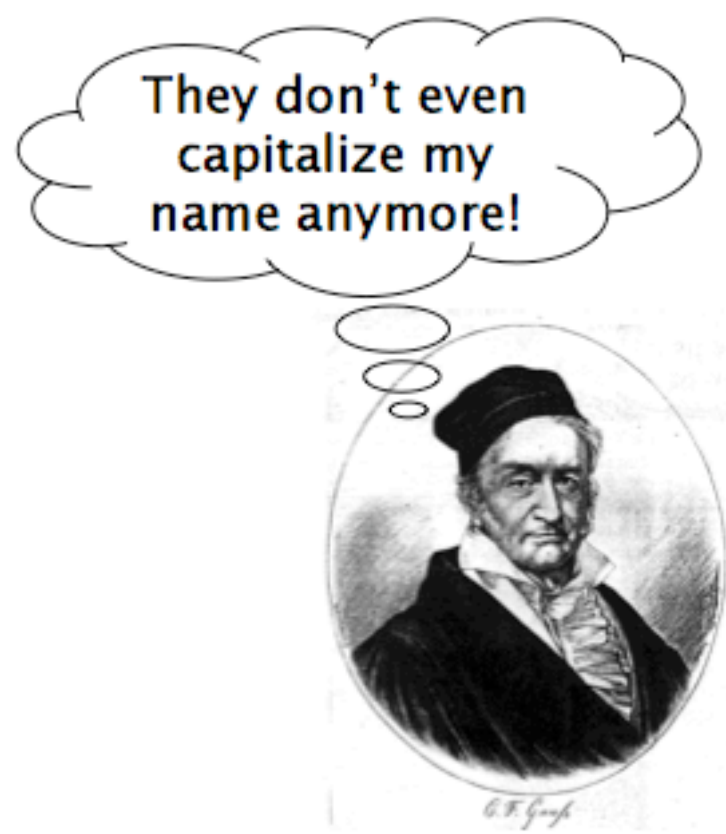
  Posterior Prior Likelihood

Smoothing: Priors

- Gaussian, or quadratic, priors:
  - Intuition: parameters shouldn't be large.
  - Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean μ and variance σ^2.



$$P(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} \right)$$



  - Penalizes parameters for drifting to far from their mean prior value (usually μ=0).
  - $2\sigma^2$=1 works surprisingly well.

Parsing as Structured Prediction

Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb $\rightarrow$ book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det $\rightarrow$ that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun $\rightarrow$ flight
(Verb Det Noun)		reduce, NOM $\rightarrow$ Noun
(Verb Det NOM)		reduce, NP $\rightarrow$ Det NOM
(Verb NP)		reduce, VP $\rightarrow$ Verb NP
(Verb)		reduce, S $\rightarrow$ V
(S)		SUCCESS!

Ambiguity may lead to the need for backtracking.

Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb $\rightarrow$ book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det $\rightarrow$ that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun $\rightarrow$ flight
(Verb Det Noun)		reduce, NOM $\rightarrow$ Noun
(Verb Det NOM)		reduce, NP $\rightarrow$ Det NOM
(Verb NP)		reduce, VP $\rightarrow$ Verb NP
(Verb)		reduce, S $\rightarrow$ V
(S)		SUCCESS!

~~Ambiguity may lead to the need for backtracking.~~

Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb → book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det → that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun → flight
(Verb Det Noun)		reduce, NOM → Noun
(Verb Det NOM)		reduce, NP → Det NOM
(Verb NP)		reduce, VP → Verb NP
(Verb)		reduce, S → V
(S)		SUCCESS!

~~Ambiguity may lead to the need for backtracking.~~

Train log-linear model of p(action | context)

Compare to an MEMM

Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb $\rightarrow$ book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det $\rightarrow$ that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun $\rightarrow$ flight
(Verb Det Noun)		reduce, NOM $\rightarrow$ Noun
(Verb Det NOM)		reduce, NP $\rightarrow$ Det NOM
(Verb NP)		reduce, VP $\rightarrow$ Verb NP
(Verb)		reduce, S $\rightarrow$ V
(S)		SUCCESS!

~~Ambiguity may lead to the need for backtracking.~~

Train log-linear model of p(action | context)

Structured Log-Linear Models

Structured Log-Linear Models

- Linear model for scoring structures

$$\mathrm{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing

$$\mathrm{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

$$p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}$$

Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models

$$\mathrm{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

$$p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}$$

Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models

Usually the bottleneck in NLP

$$\text{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

$$p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}$$

Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models
- Inference: sampling, variational methods, dynamic programming, local search, ...

Usually the bottleneck in NLP

$$\text{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

$$p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}$$

Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models
- Inference: sampling, variational methods, dynamic programming, local search, ...
- Training: maximum likelihood, minimum risk, etc.

Usually the bottleneck in NLP

$$\mathrm{score}(out, in) = \theta \cdot \mathbf{features}(out, in)$$

$$p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}$$

Structured Log-Linear Models

With latent variables

- Several layers of linguistic structure
- Unknown correspondences
- Naturally handled by probabilistic framework
- Several inference setups, for example:

$$p(out_1 \mid in) = \sum_{out_2, alignment} p(out_1, out_2, alignment \mid in)$$

Structured Log-Linear Models

With latent variables

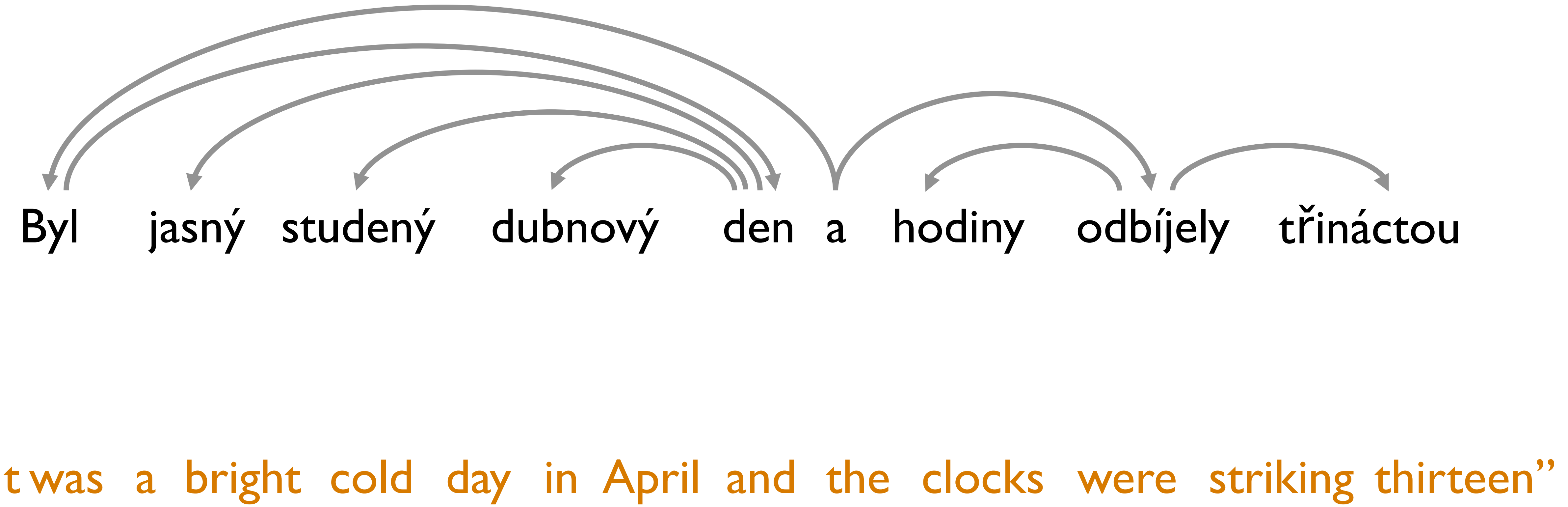
- Several layers of linguistic structure
- Unknown correspondences
- Naturally handled by probabilistic framework
- Several inference setups, for example:

$$p(out_1 \mid in) = \sum_{out_2, alignment} p(out_1, out_2, alignment \mid in)$$

Another computational problem

Edge-Factored Parsers

- No <u>global</u> features of a parse *(McDonald et al. 2005)*
- Each feature is attached to some edge
- *MST or CKY-like DP for fast $O(n^2)$ or $O(n^3)$ parsing*



Edge-Factored Parsers

- Is this a good edge?

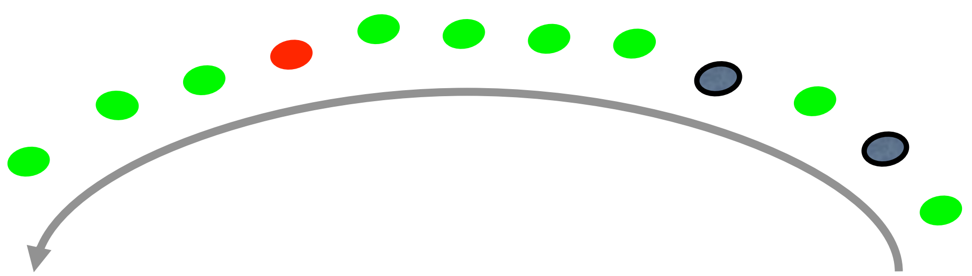
Byl jasný studený dubnový den a hodiny odbíjely třináctou

“It was a bright cold day in April and the clocks were striking thirteen”

Edge-Factored Parsers

- Is this a good edge?

yes, lots of positive features ...

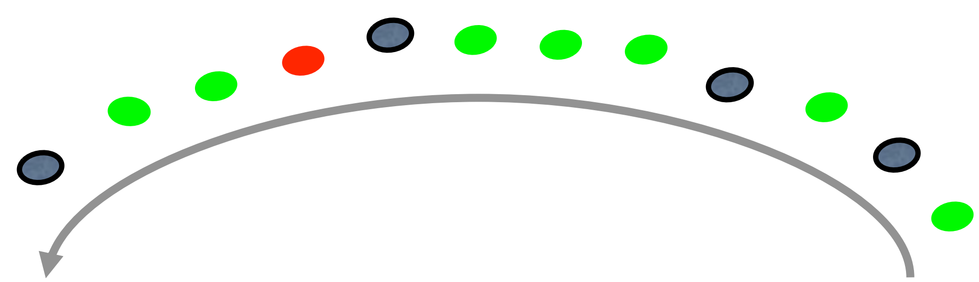


Byl jasný studený dubnový den a hodiny odbíjely třináctou

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- Is this a good edge?



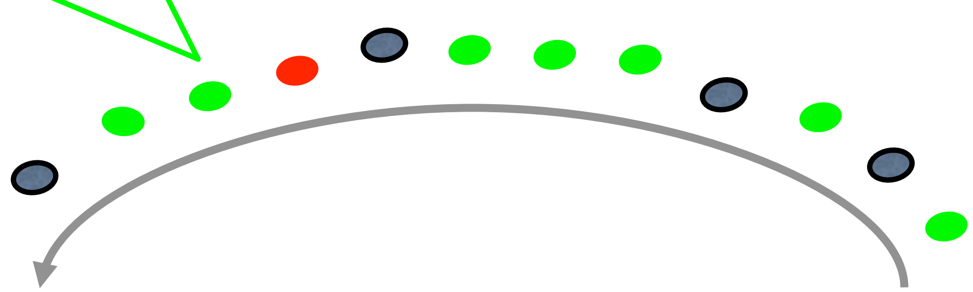
Byl jasný studený dubnový den a hodiny odbíjely třináctou

“It was a bright cold day in April and the clocks were striking thirteen”

Edge-Factored Parsers

- Is this a good edge?

jasný ← den
("bright day")



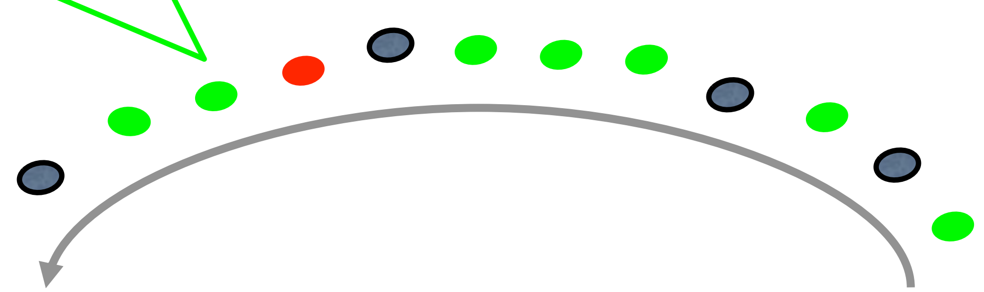
Byl **jasný** studený dubnový **den** a hodiny odbíjely třináctou

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- Is this a good edge?

jasný ← den
("bright day")



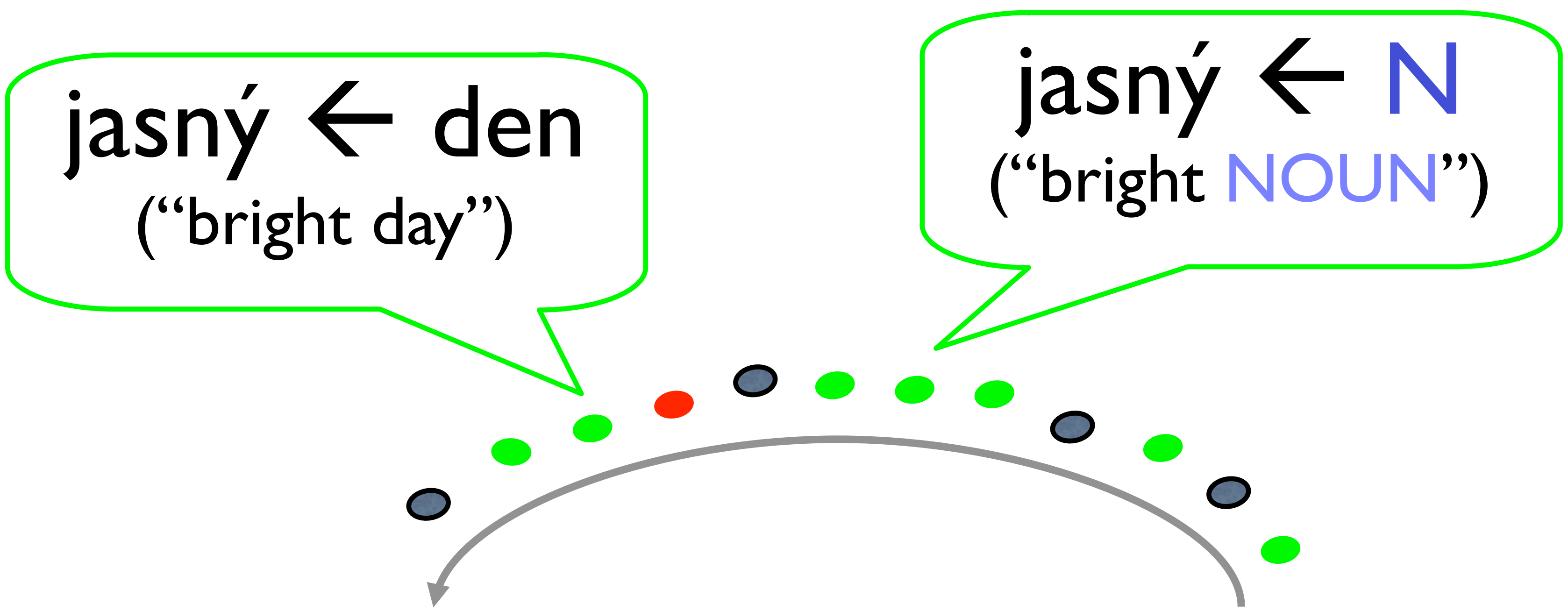
Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A N J N V C

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- Is this a good edge?

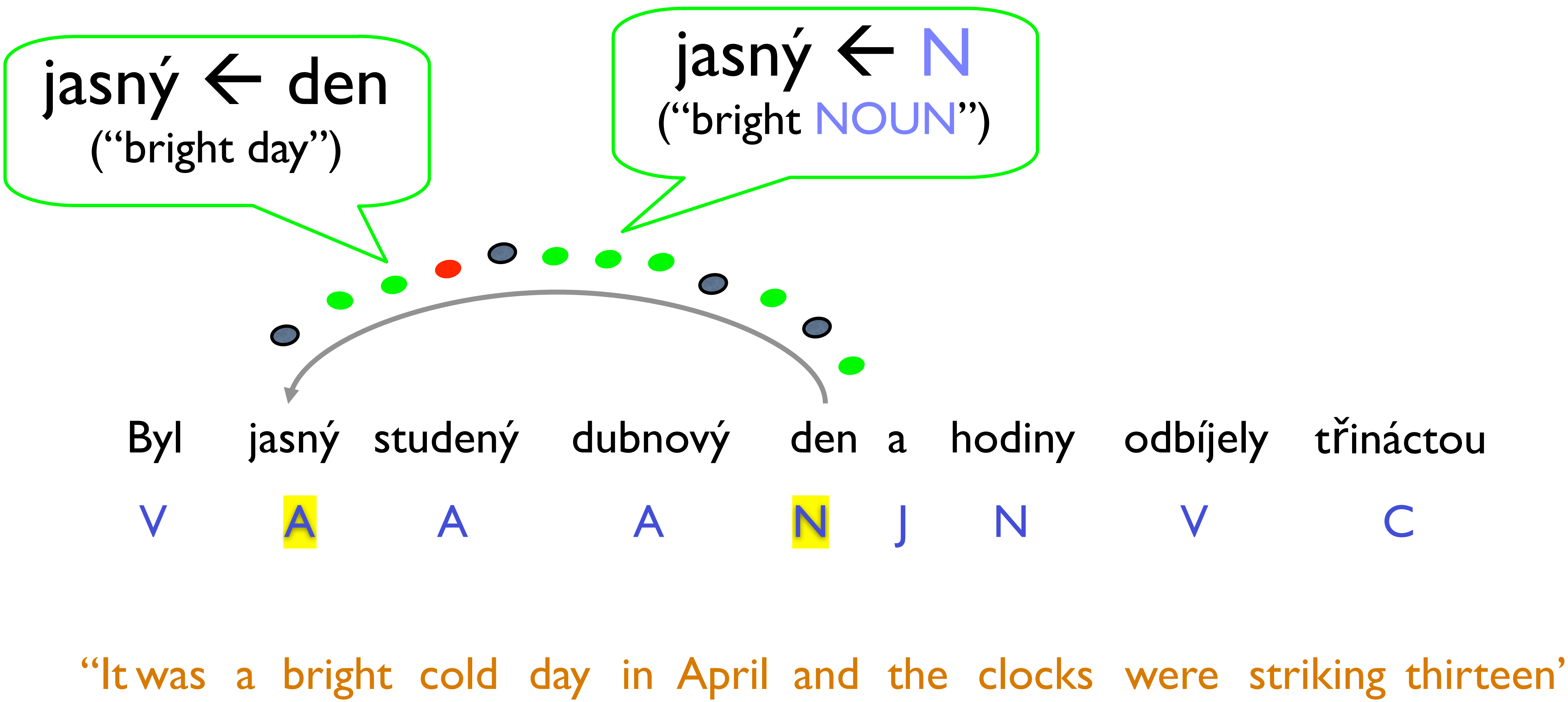


Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
V	A	A	A	N	J	N	V	C

"It was a bright cold day in April and the clocks were striking thirteen"

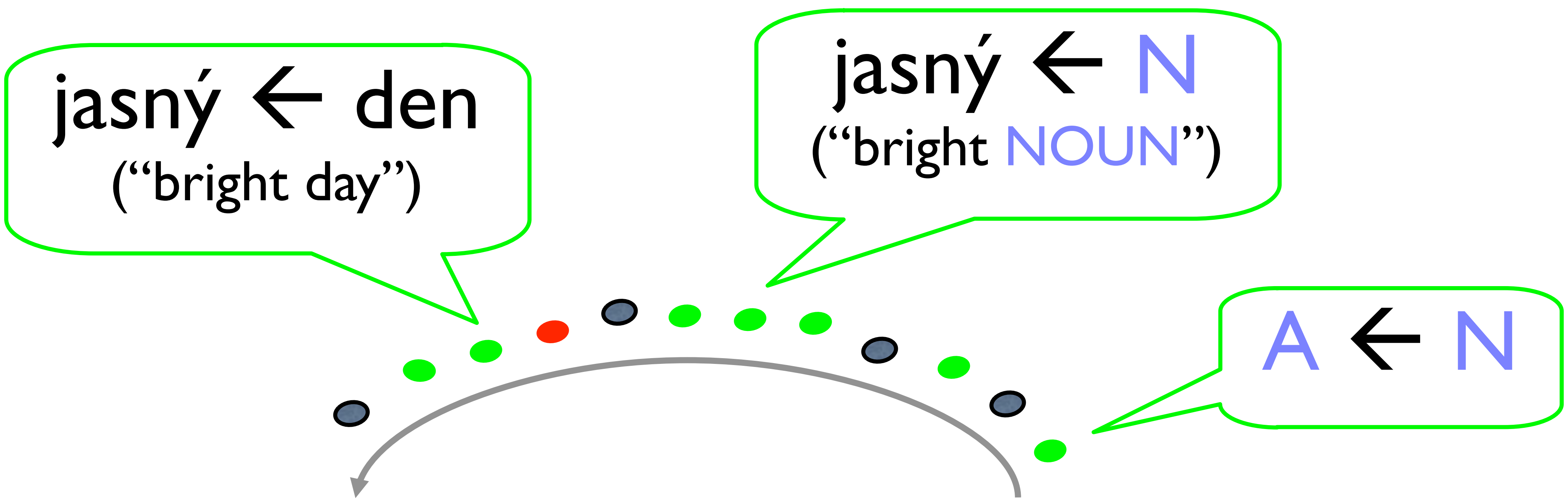
Edge-Factored Parsers

- Is this a good edge?



Edge-Factored Parsers

- Is this a good edge?



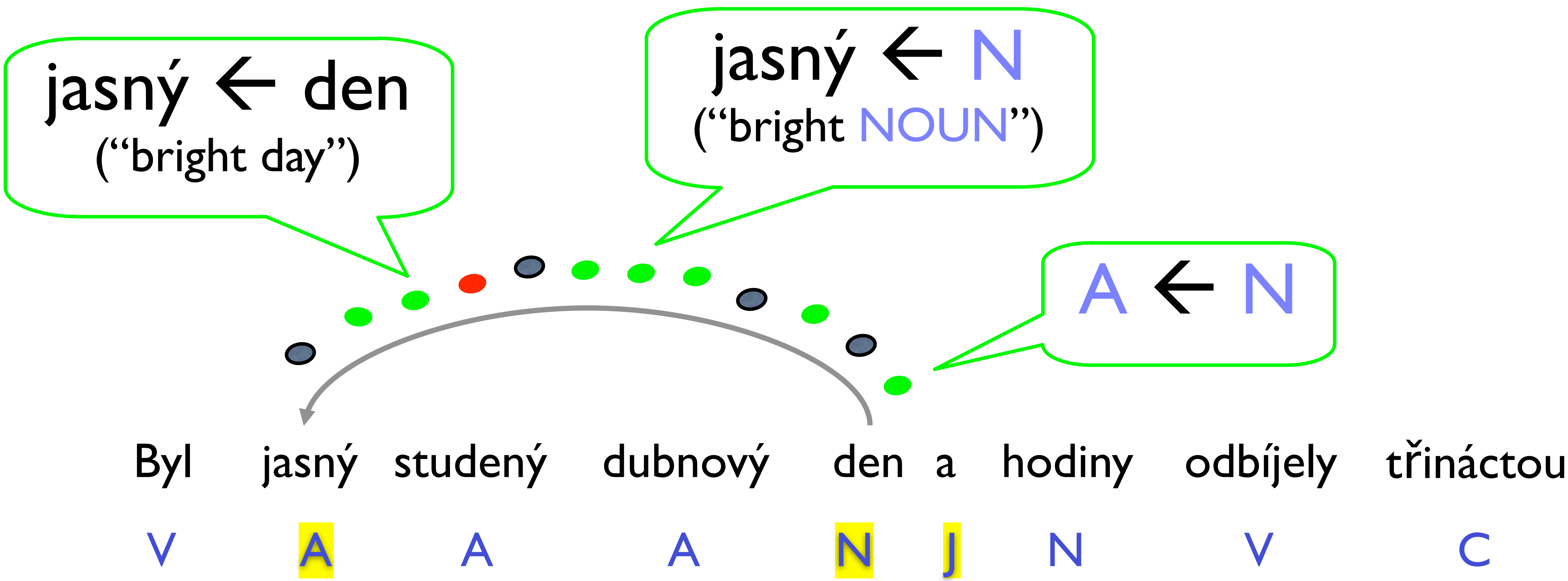
Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A N J N V C

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

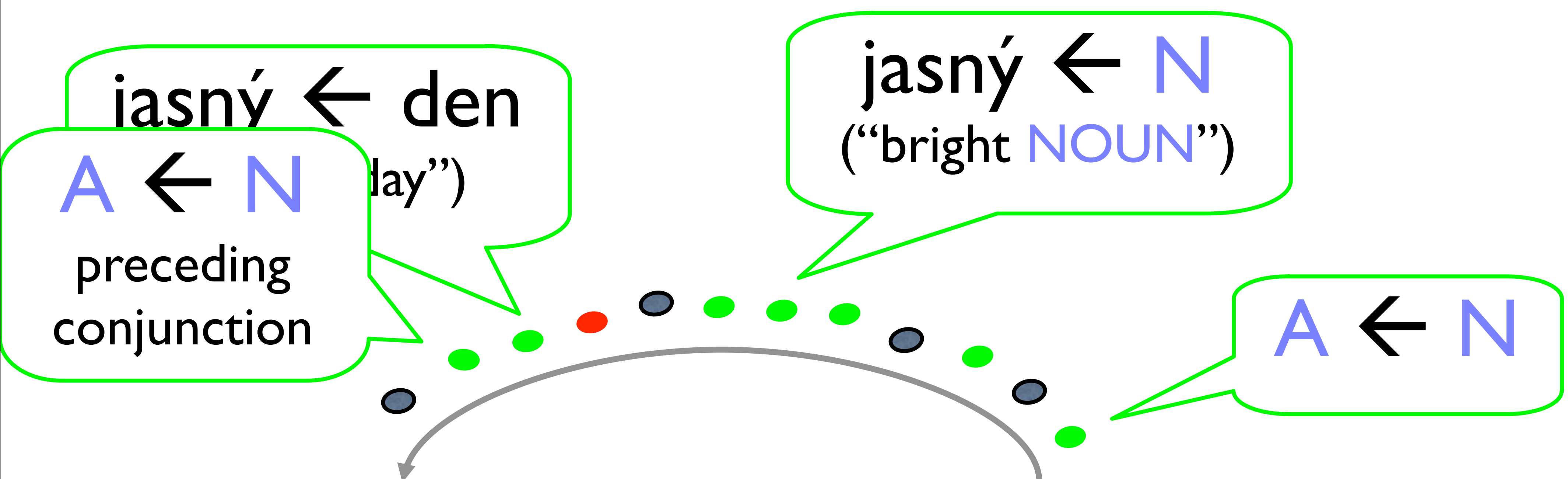
- Is this a good edge?



"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- Is this a good edge?



Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A N J N V C

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?

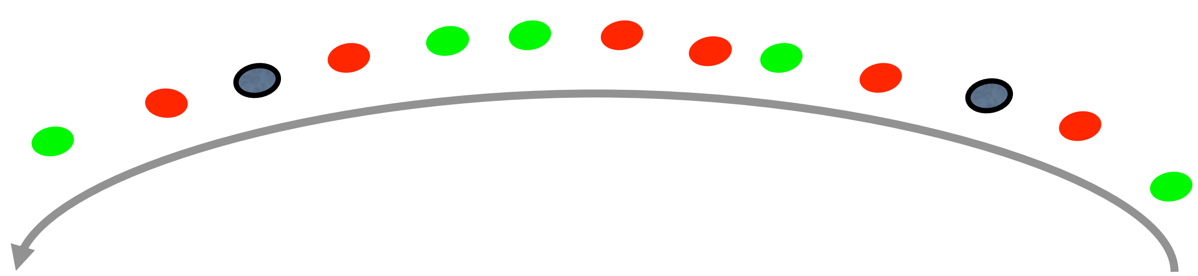
Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
V	A	A	A	N	J	N	V	C

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?

not as good, lots of red ...



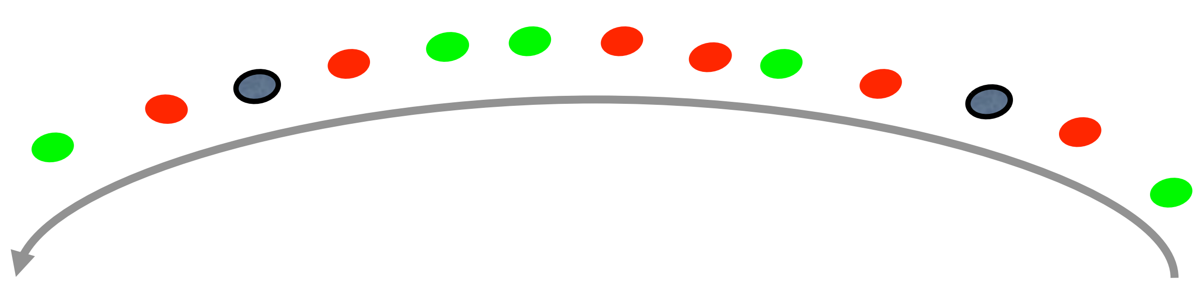
Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A N J N V C

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?



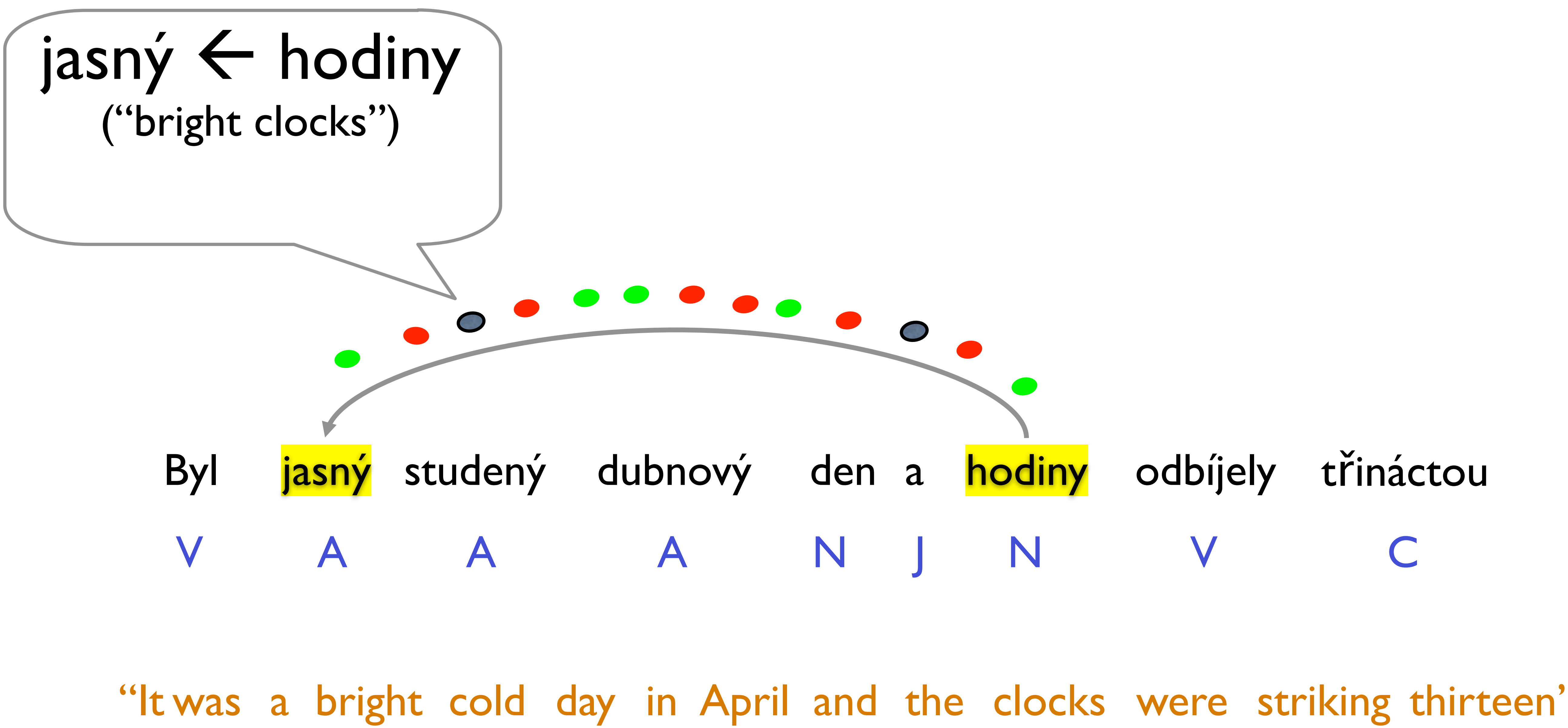
Byl **jasný** studený dubnový den a **hodiny** odbíjely třináctou

V A A A N J N V C

"It was a bright cold day in April and the clocks were striking thirteen"

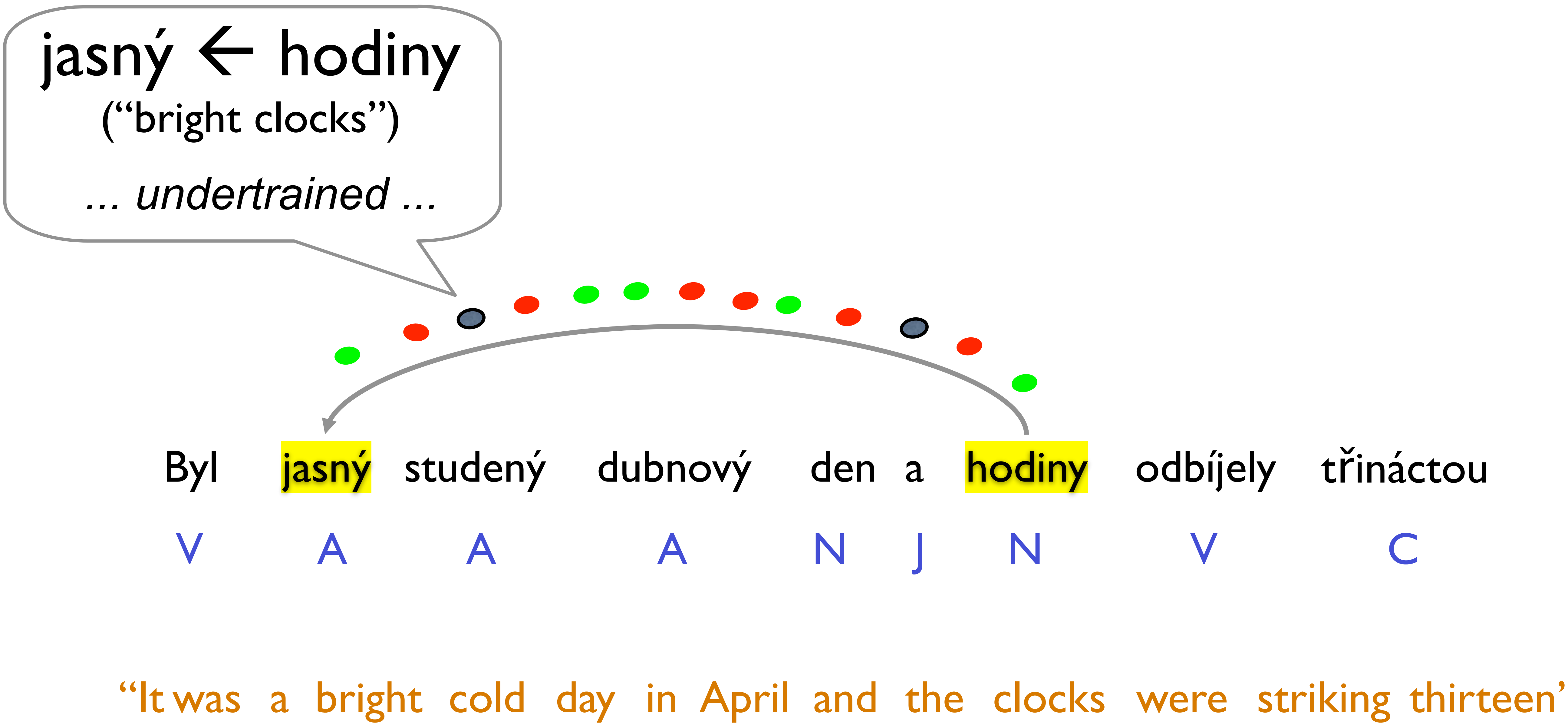
Edge-Factored Parsers

- How about this competing edge?



Edge-Factored Parsers

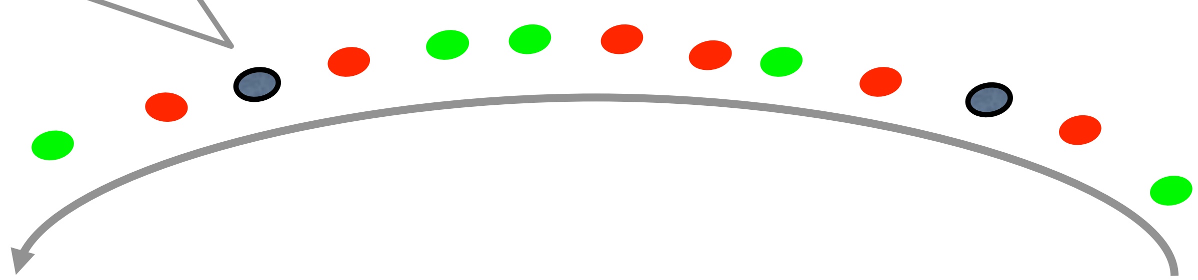
- How about this competing edge?



Edge-Factored Parsers

- How about this competing edge?

jasný ← hodiny
("bright clocks")
... undertrained ...



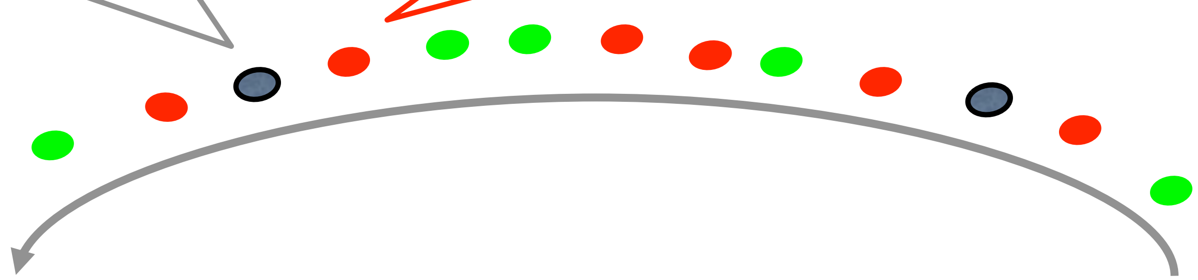
Byl	jasný	studený		dubnový	den	a	hodiny	odbíjely	třináctou	
V	A	A		A	N	J	N	V	C	
byl	jasn	stud		dubn	den	a	hodi	odbí	třin	
"It was	a	bright	cold	day	in April	and	the clocks	were	striking	thirteen"

Edge-Factored Parsers

- How about this competing edge?

jasný ← hodiny
("bright clocks")
... undertrained ...

jasn ← hodi
("bright clock," stems only)



Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
V	A	A	A	N	J	N	V	C
byl	jasn	stud	dubn	den	a	hodi	odbí	třin

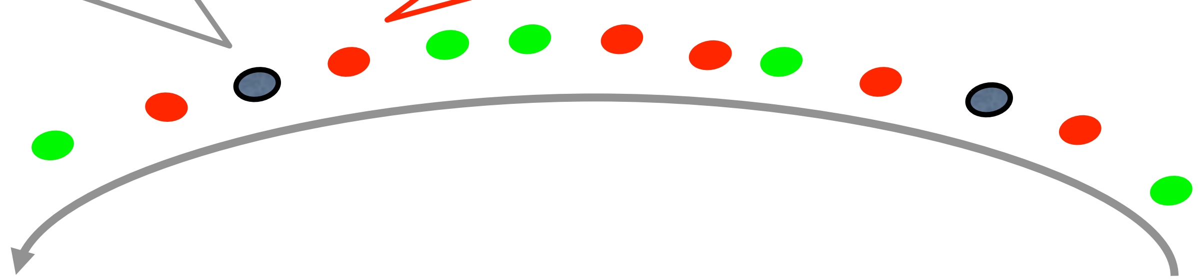
"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?

jasný ← hodiny
("bright clocks")
... undertrained ...

jasn ← hodi
("bright clock," stems only)



Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A N J N V C

byl jasn stud dubn den a hodi odbí třin

"It was a bright cold day in April and the clocks were striking thirteen"

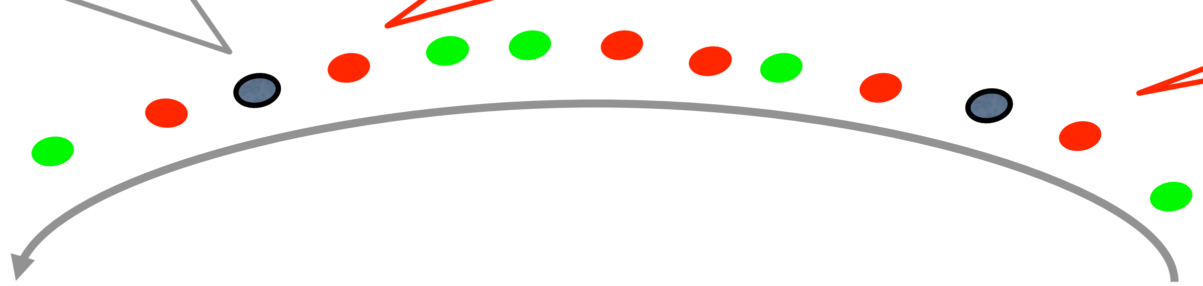
Edge-Factored Parsers

- How about this competing edge?

jasný ← hodiny
("bright clocks")
... undertrained ...

jasn ← hodi
("bright clock," stems only)

$A_{singular} \leftarrow N_{plural}$

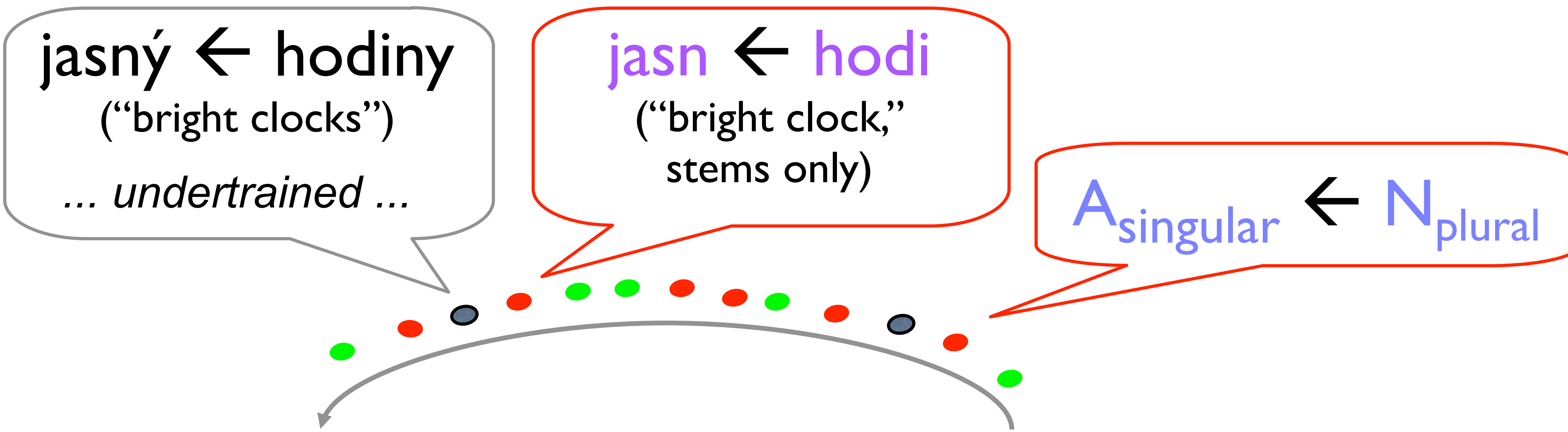


Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou	
V	A	A	A	N	J	N	V	C	
byl	jasn	stud	dubn	den	a	hodi	odbí	třin	

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?



Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
V	A	A	A	N	J	N	V	C
byl	jasn	stud	dubn	den	a	hodi	odbí	třin

"It was a bright cold day in April and the clocks were striking thirteen"

Edge-Factored Parsers

- How about this competing edge?

jasný ← hodiny

A ← N
where N follows
a conjunction

jasn ← hodi
("bright clock,"
stems only)

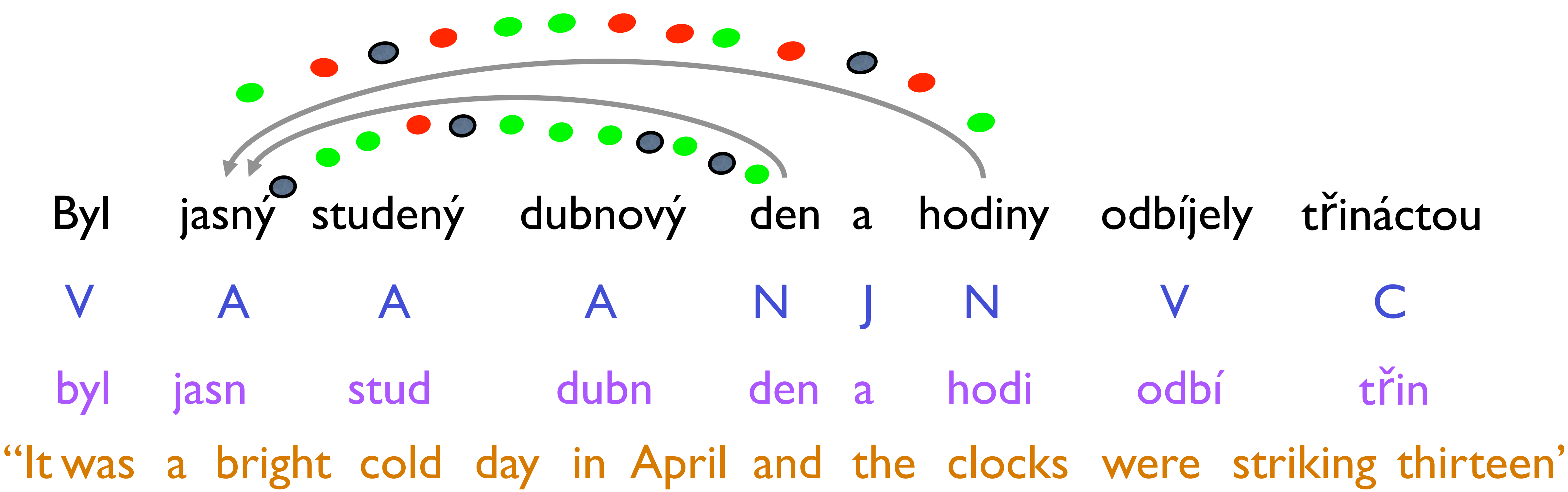
$A_{singular} \leftarrow N_{plural}$

Byl	jasný	studený	dubnový	den	a	hodiny	odbíjely	třináctou
V	A	A	A	N	J	N	V	C
byl	jasn	stud	dubn	den	a	hodi	odbí	třin

"It was a bright cold day in April and the clocks were striking thirteen"

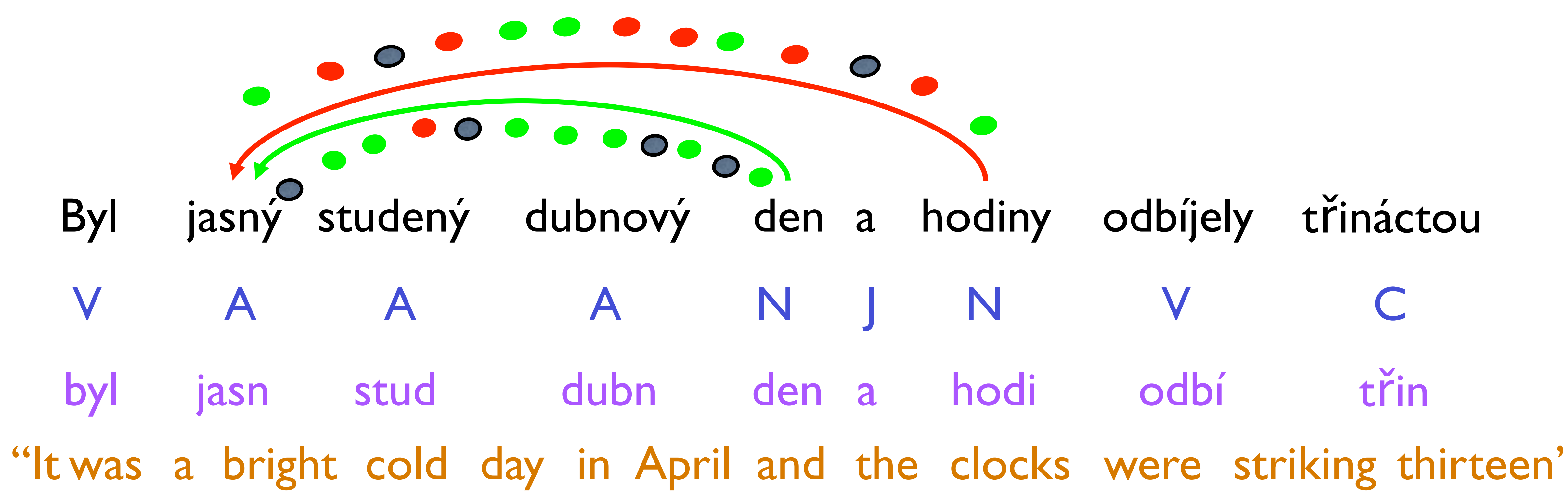
Edge-Factored Parsers

- Which edge is better?
  - “bright day” or “bright clocks”?



Edge-Factored Parsers

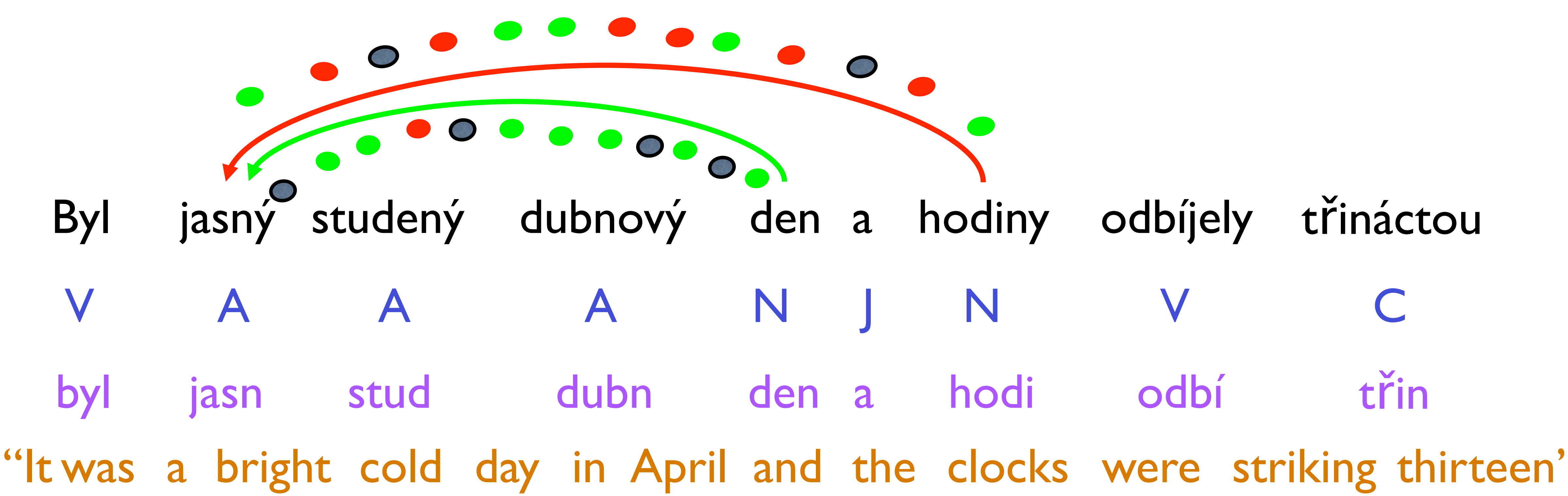
- Which edge is better?
- Score of an edge e = $\theta \cdot$ **features**(e)
- Standard algos ➔ valid parse with max total score



Edge-Factored Parsers

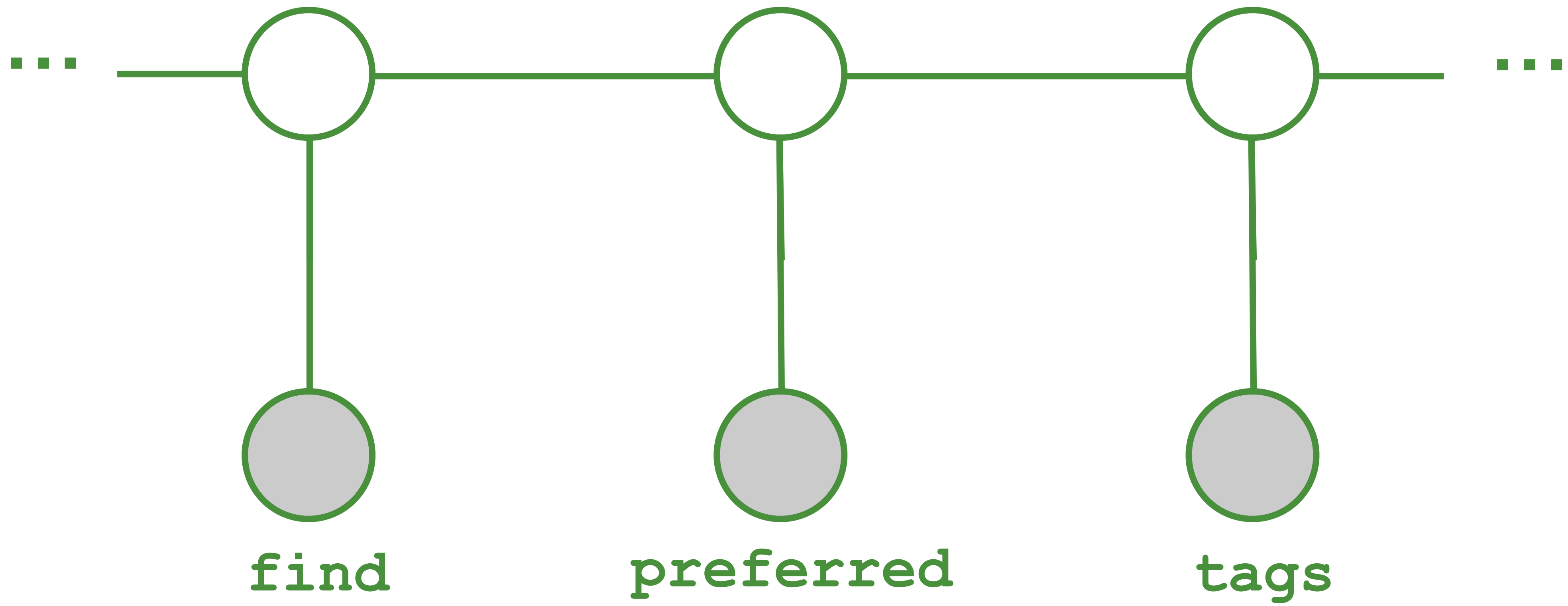
our current weight vector

- Which edge is better?
- Score of an edge e = θ · **features**(e)
- Standard algos ➔ valid parse with max <u>total</u> score



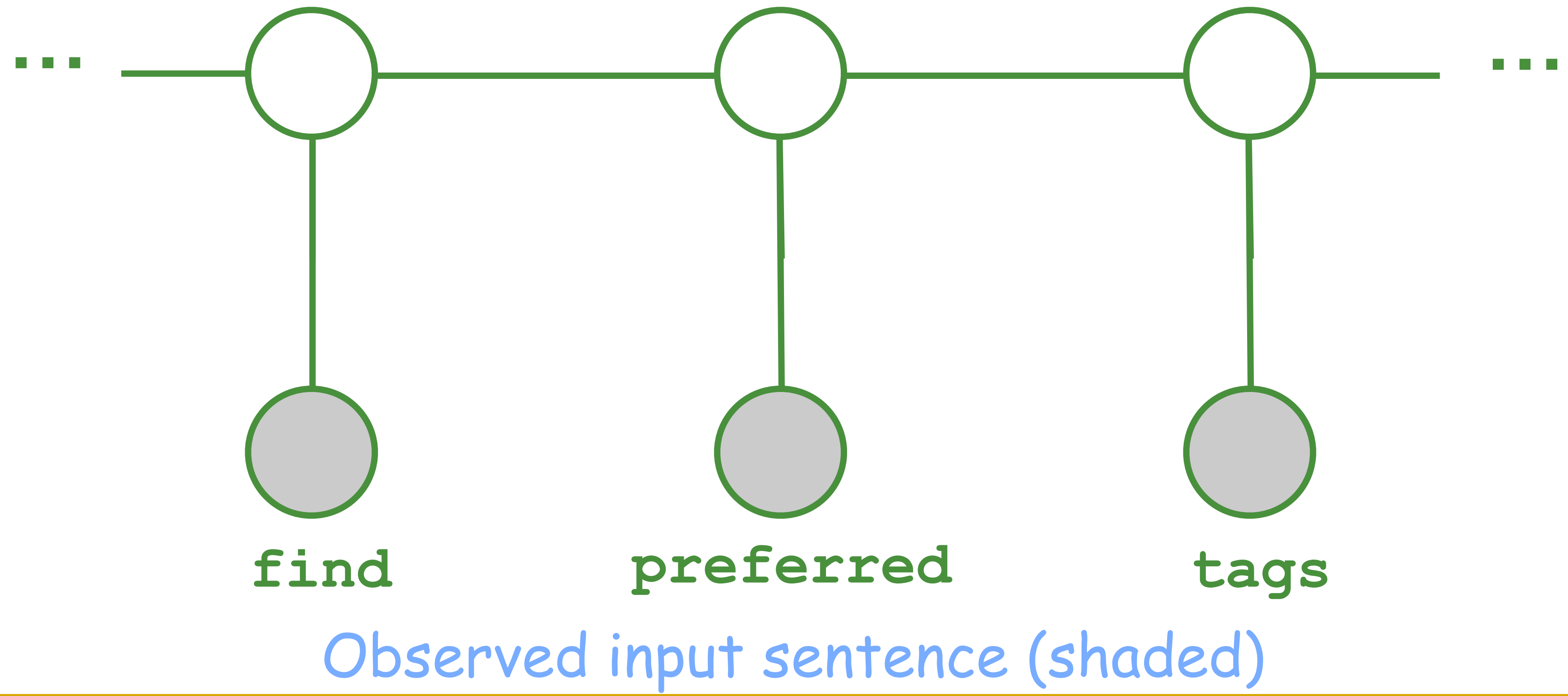
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



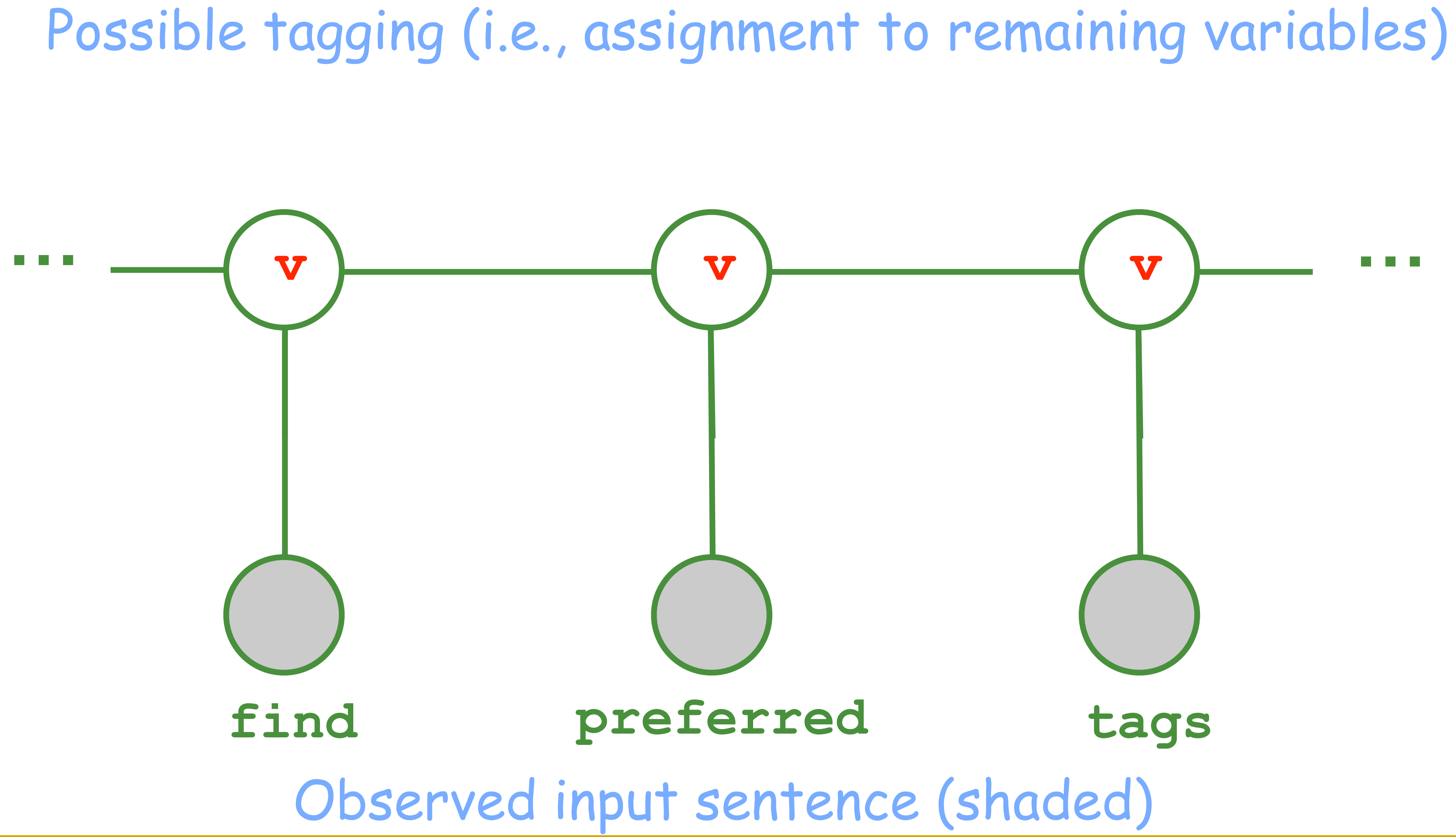
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

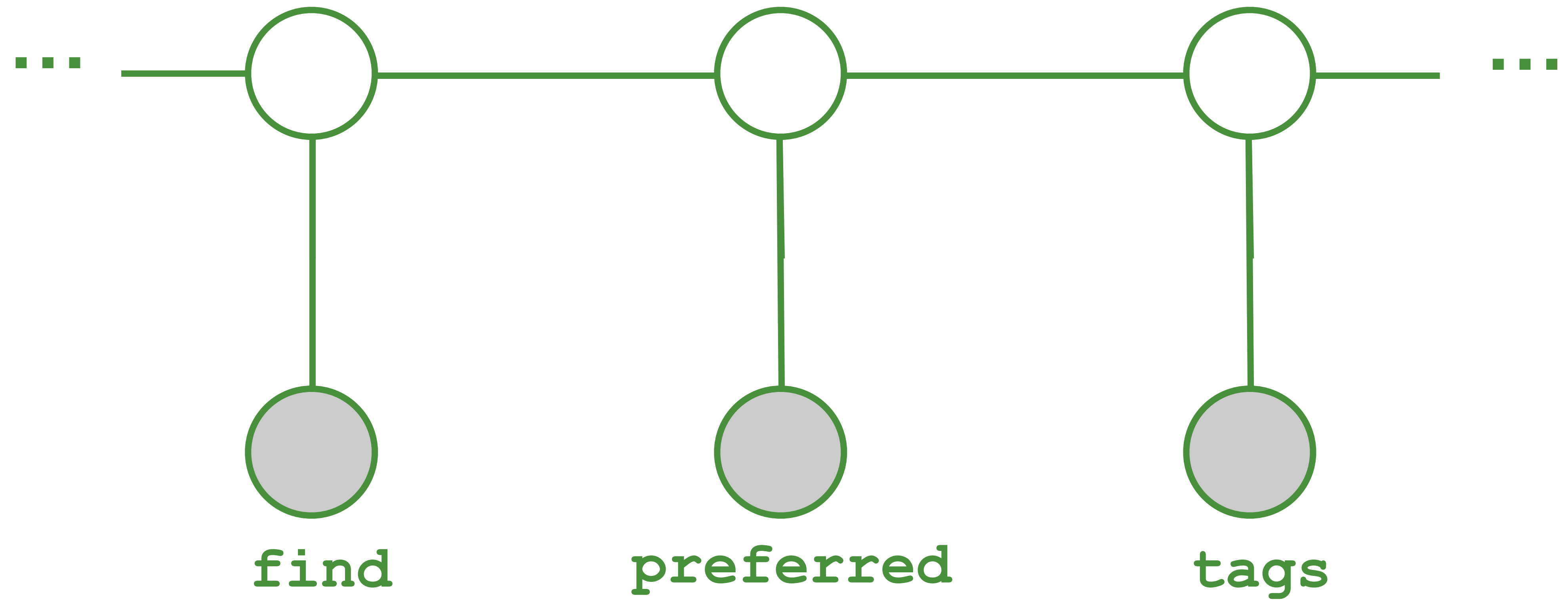
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

Possible tagging (i.e., assignment to remaining variables)
Another possible tagging

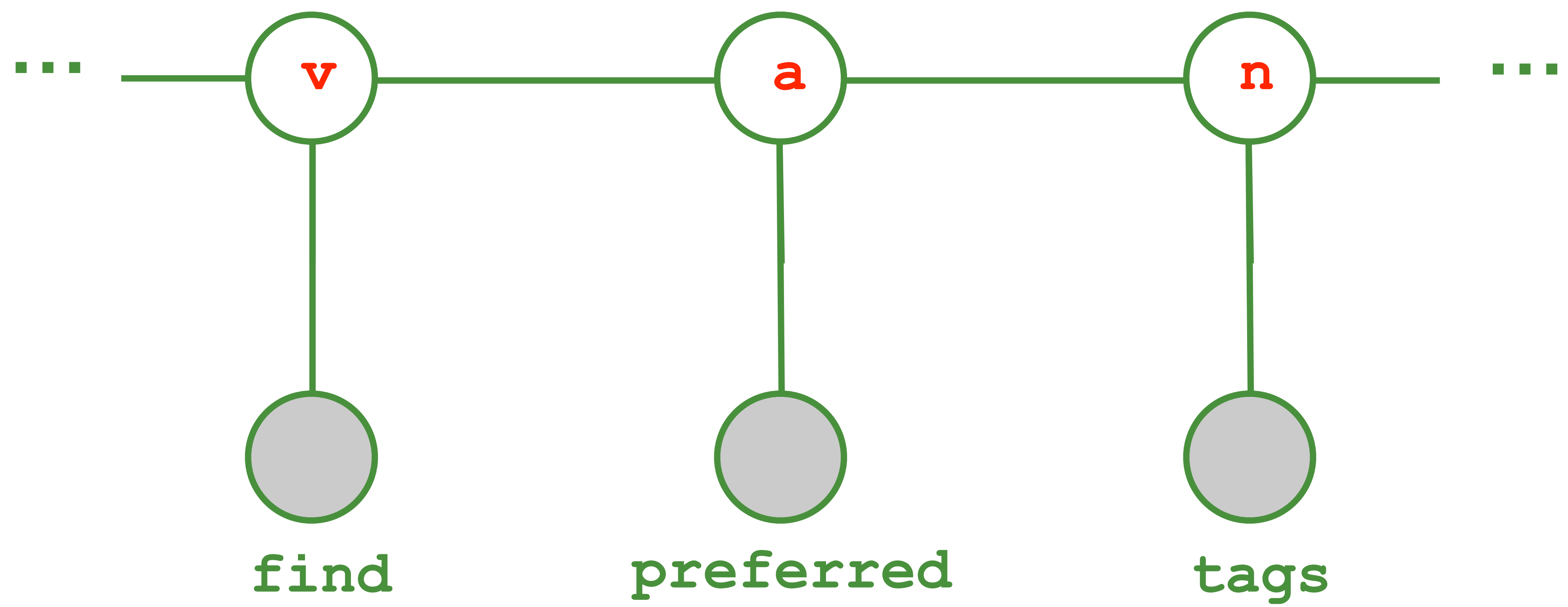


Observed input sentence (shaded)

Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

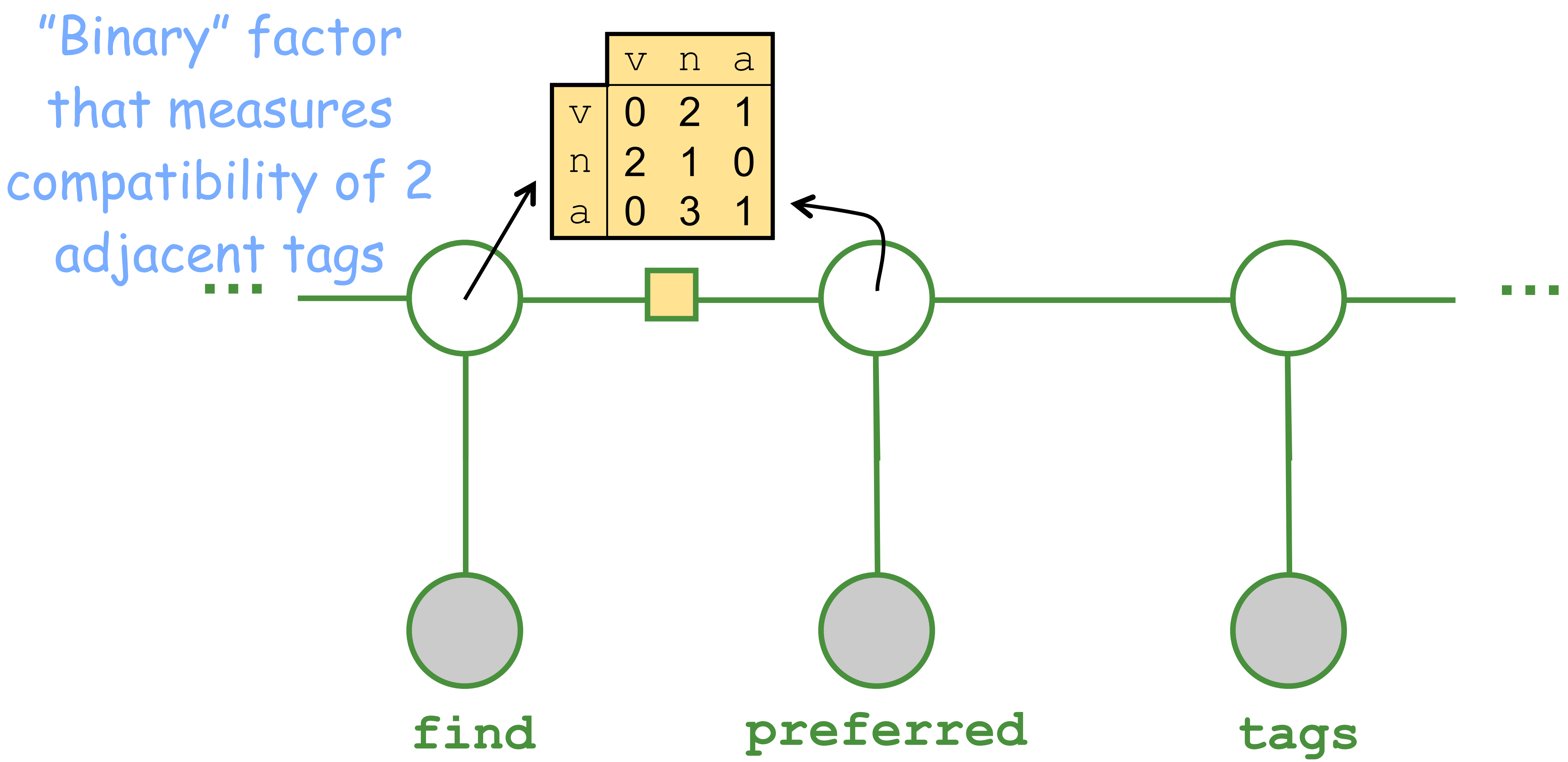
Possible tagging (i.e., assignment to remaining variables)
Another possible tagging



Observed input sentence (shaded)

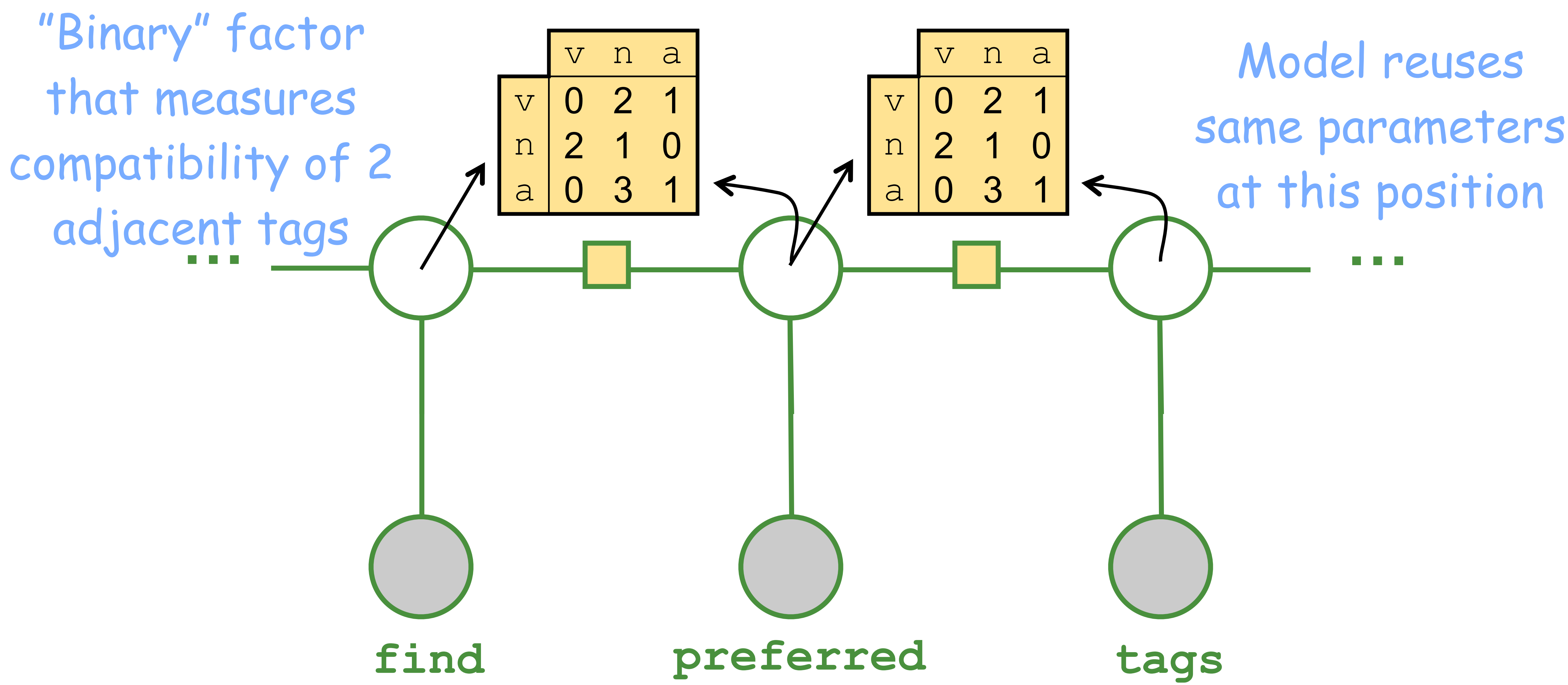
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

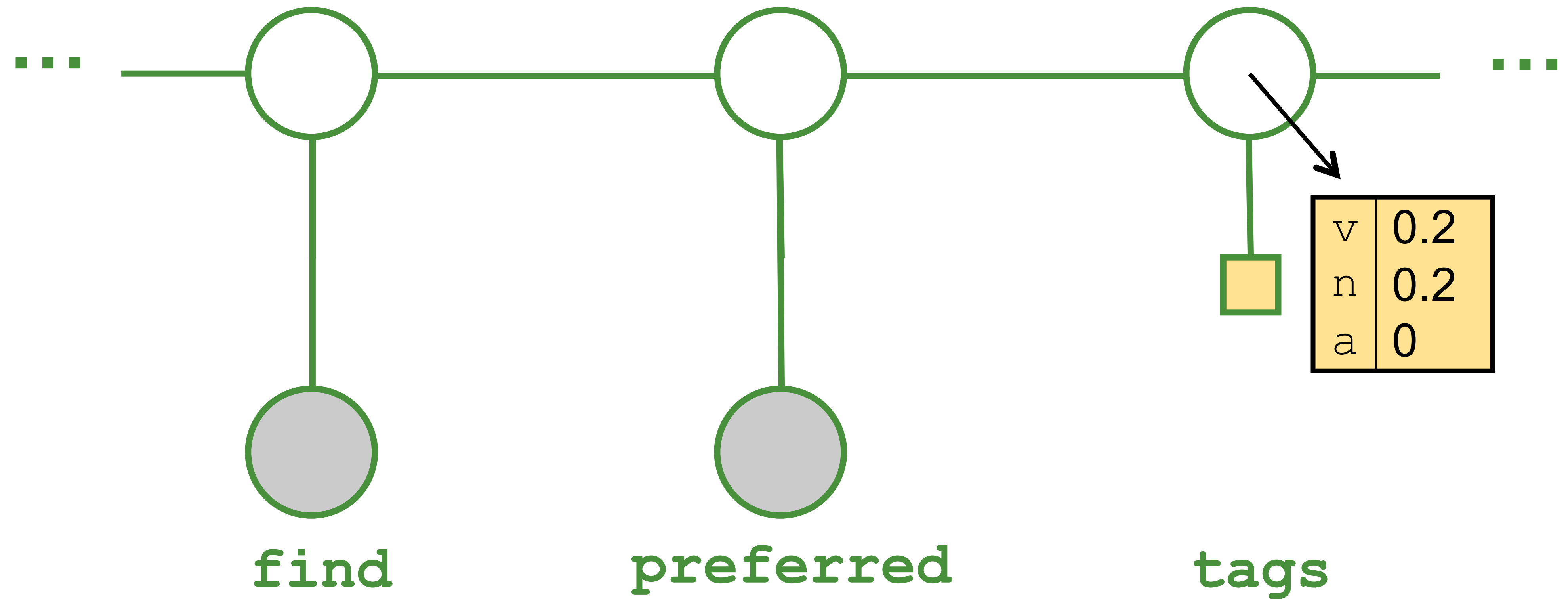


	v	n	a
v	0	2	1
n	2	1	0
a	0	3	1

	v	n	a
v	0	2	1
n	2	1	0
a	0	3	1

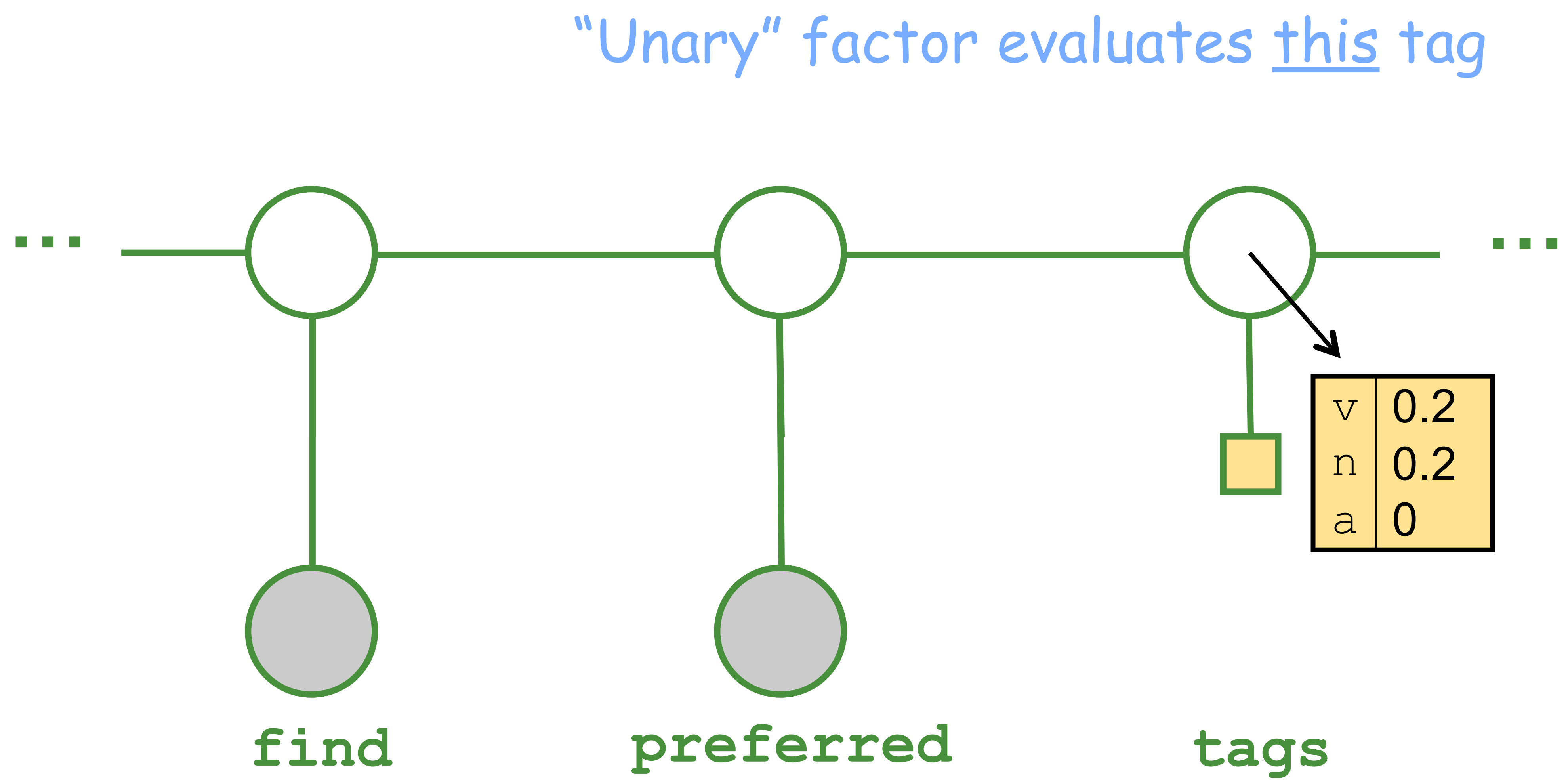
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



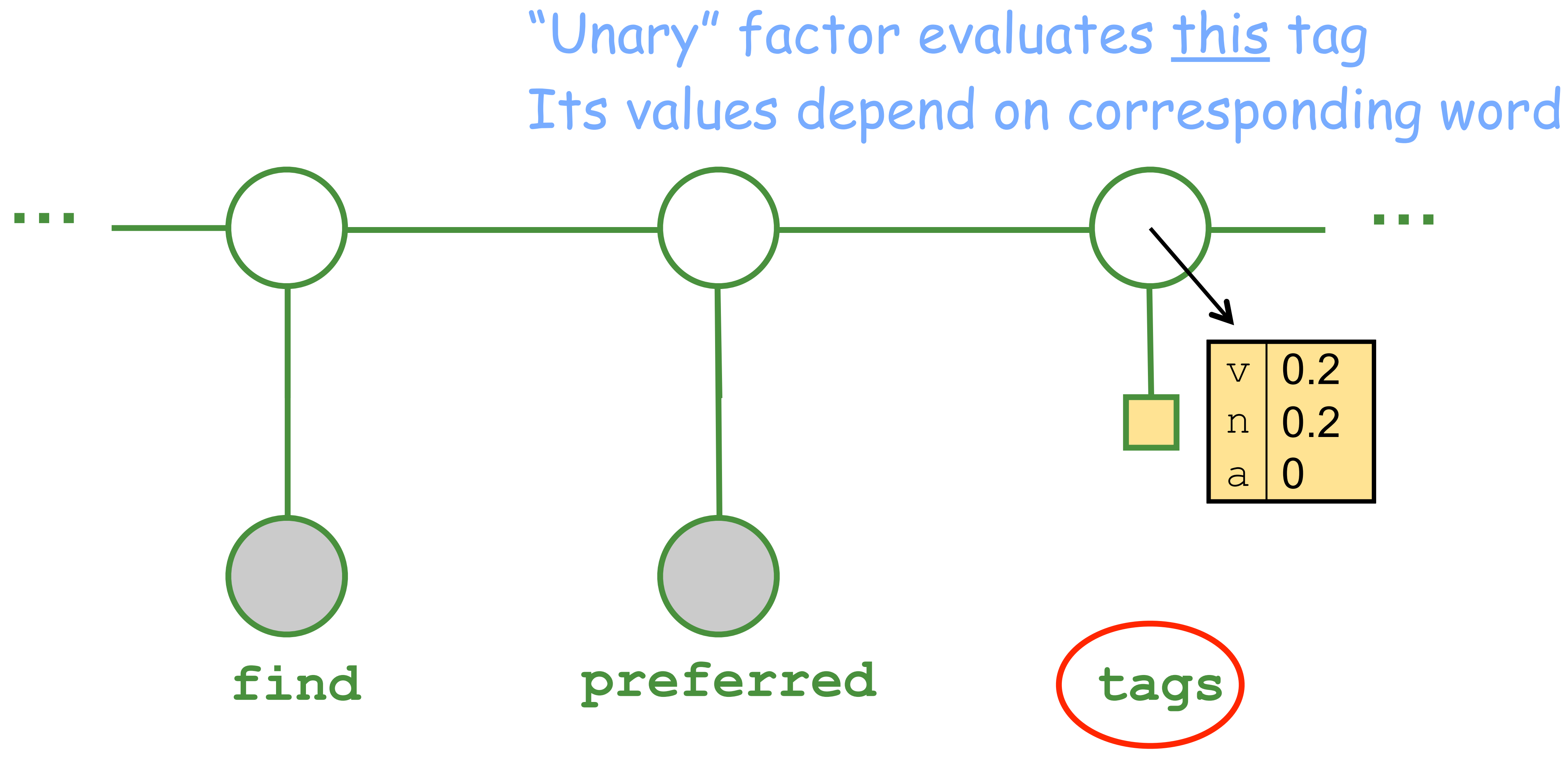
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



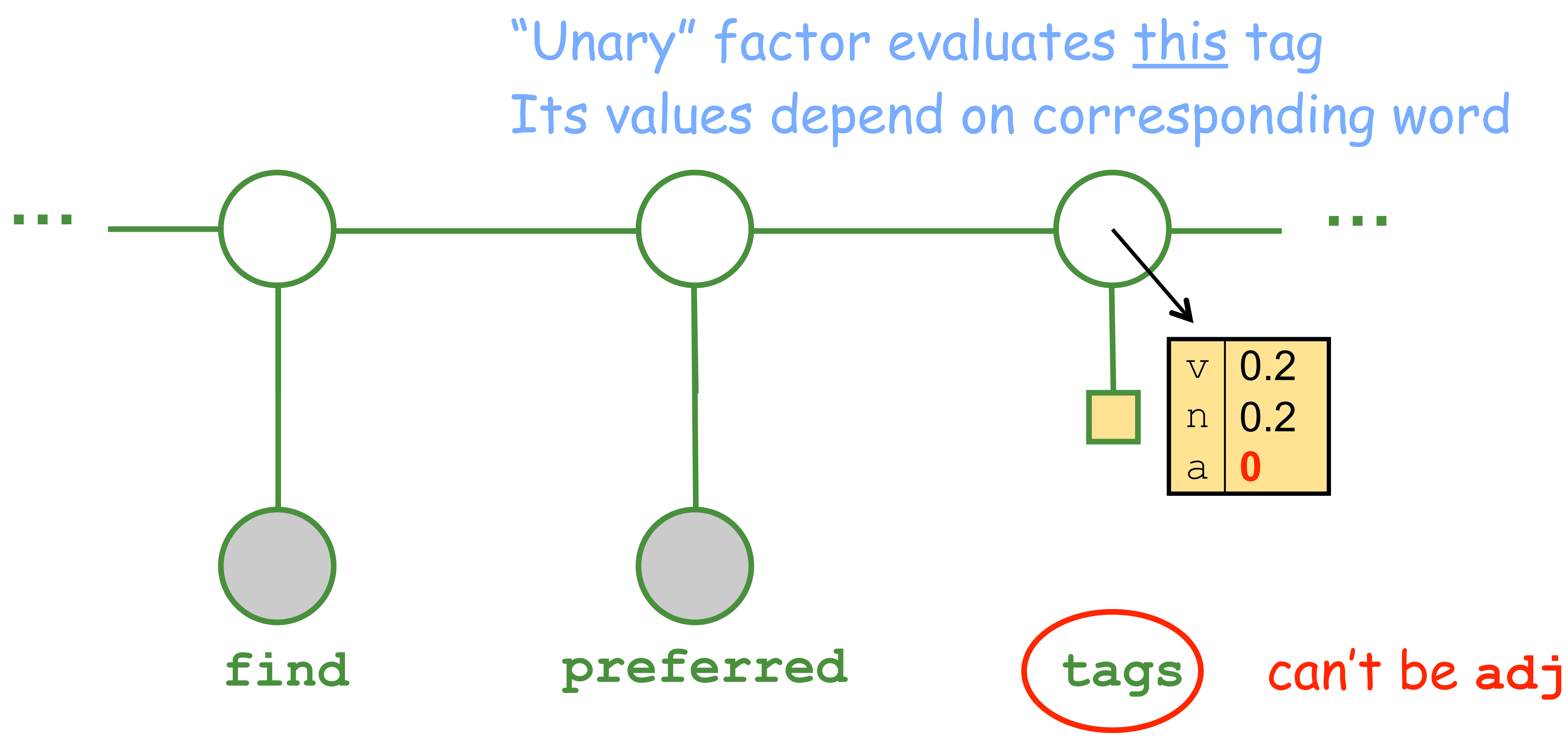
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



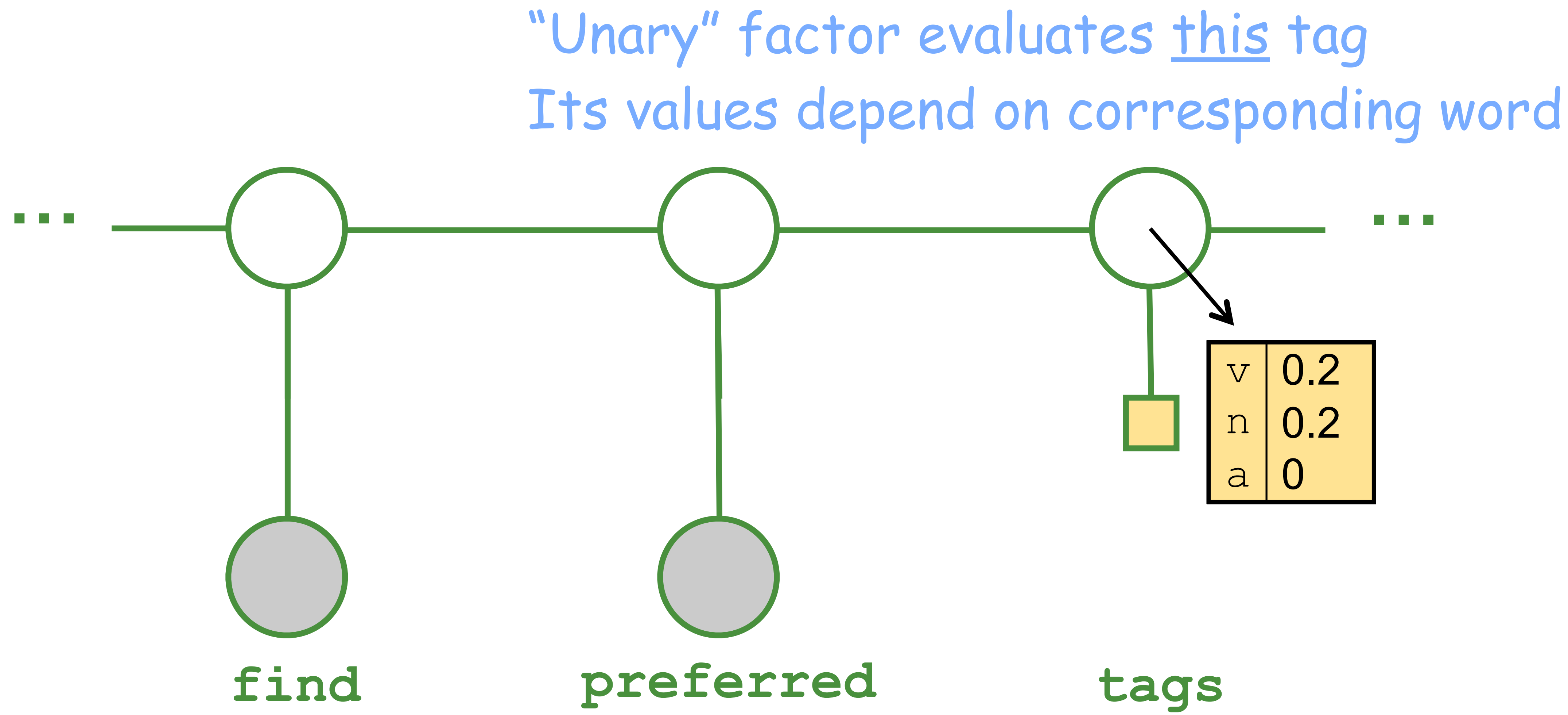
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

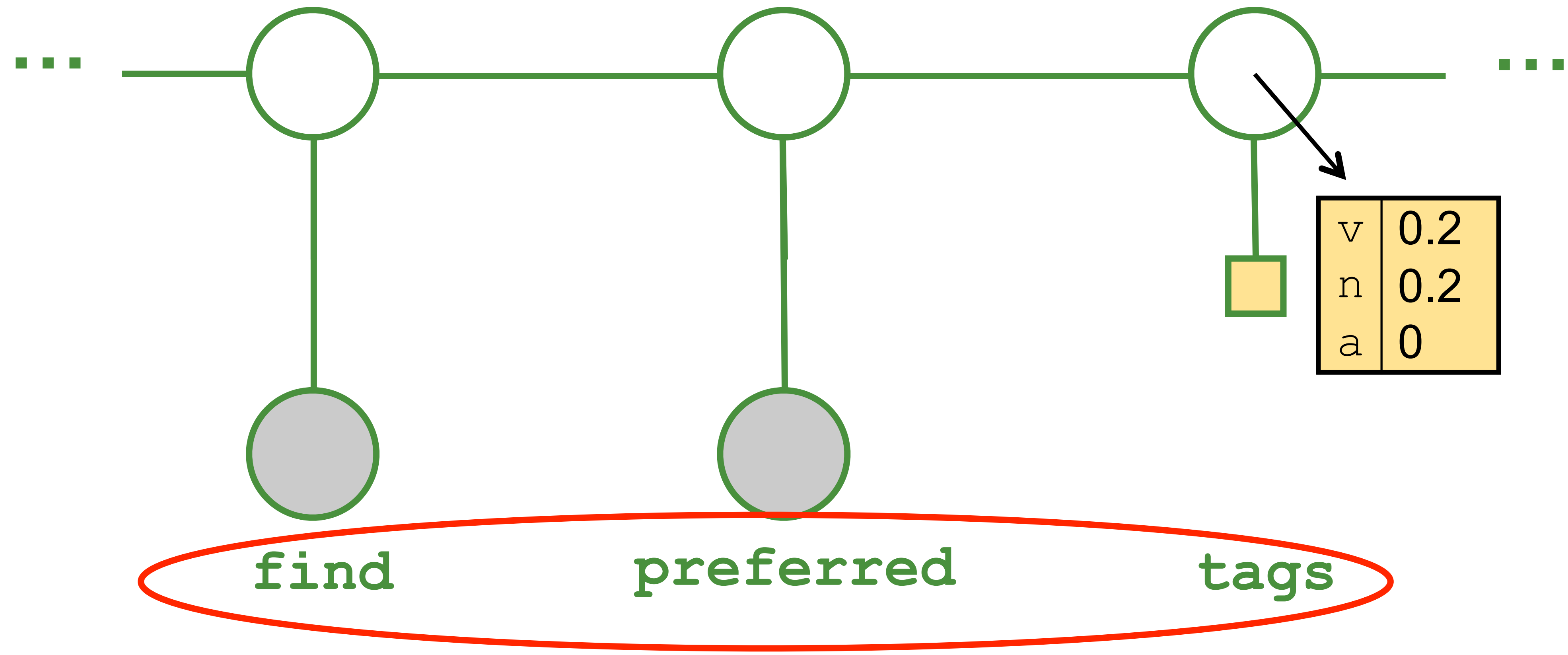
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

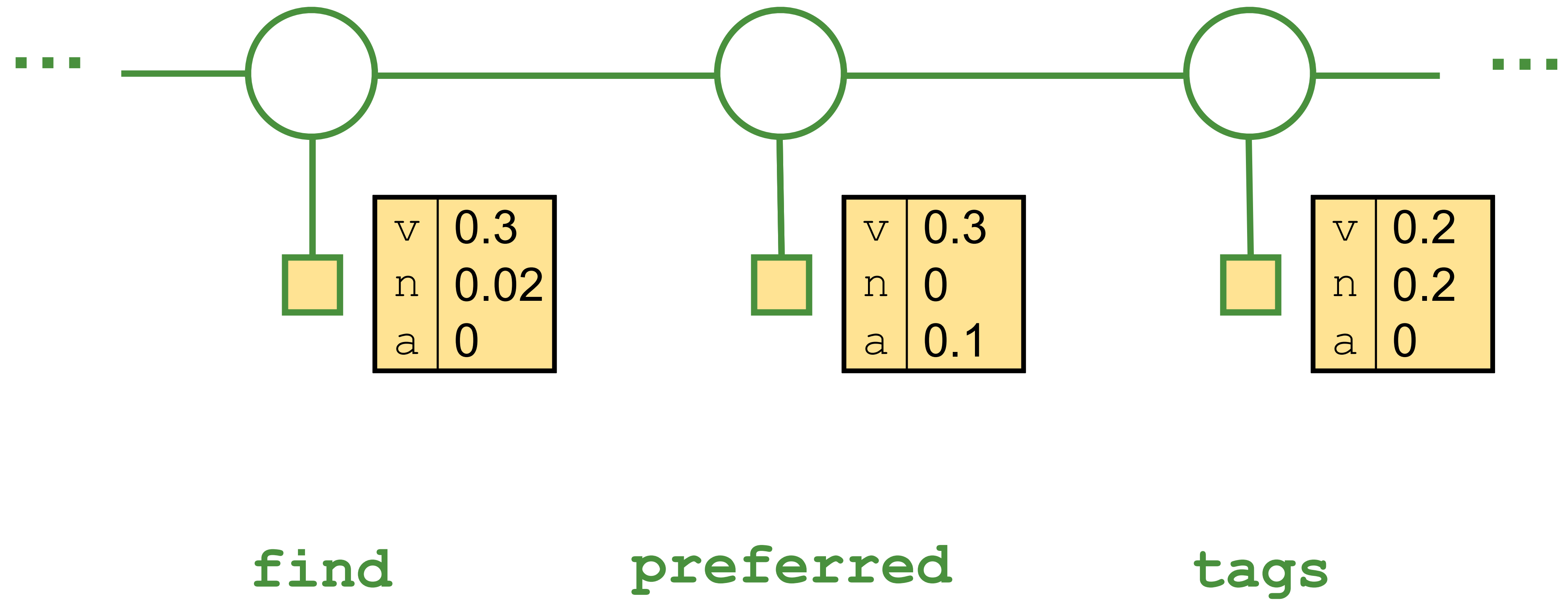
“Unary” factor evaluates this tag
Its values depend on corresponding word



(could be made to depend on entire observed sentence)

Local factors in a graphical model

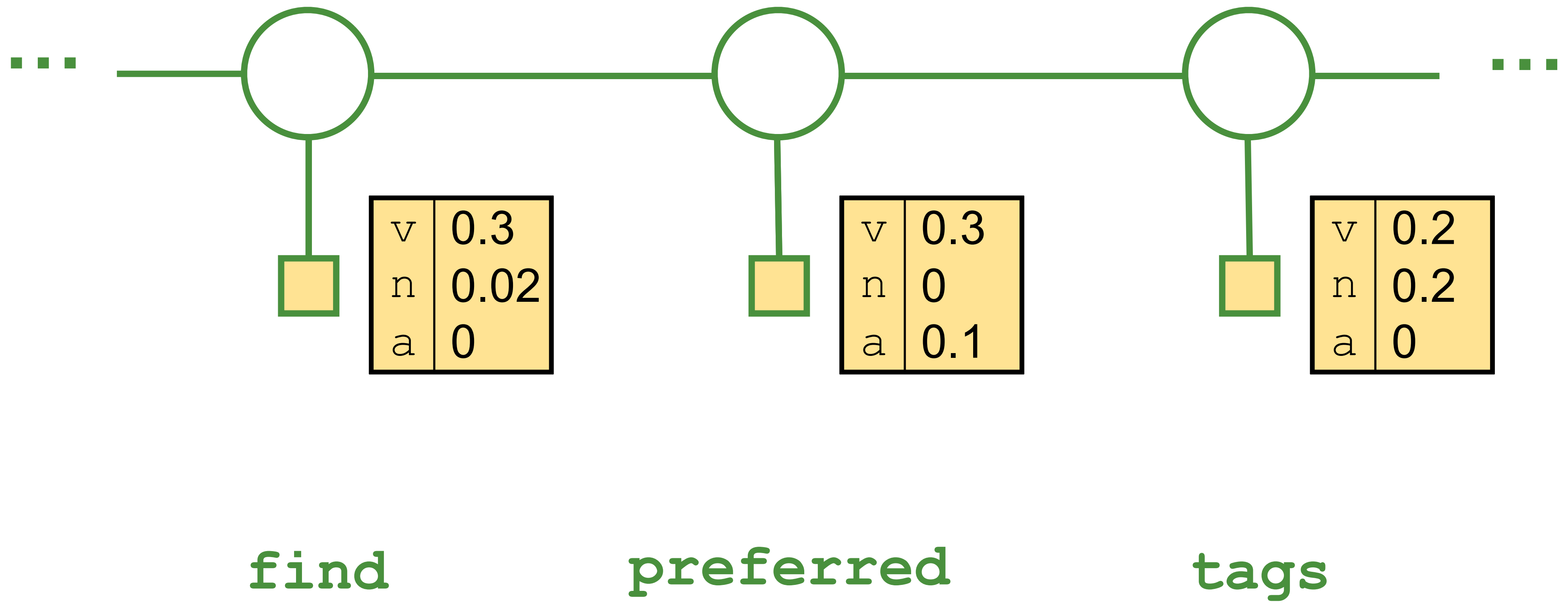
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

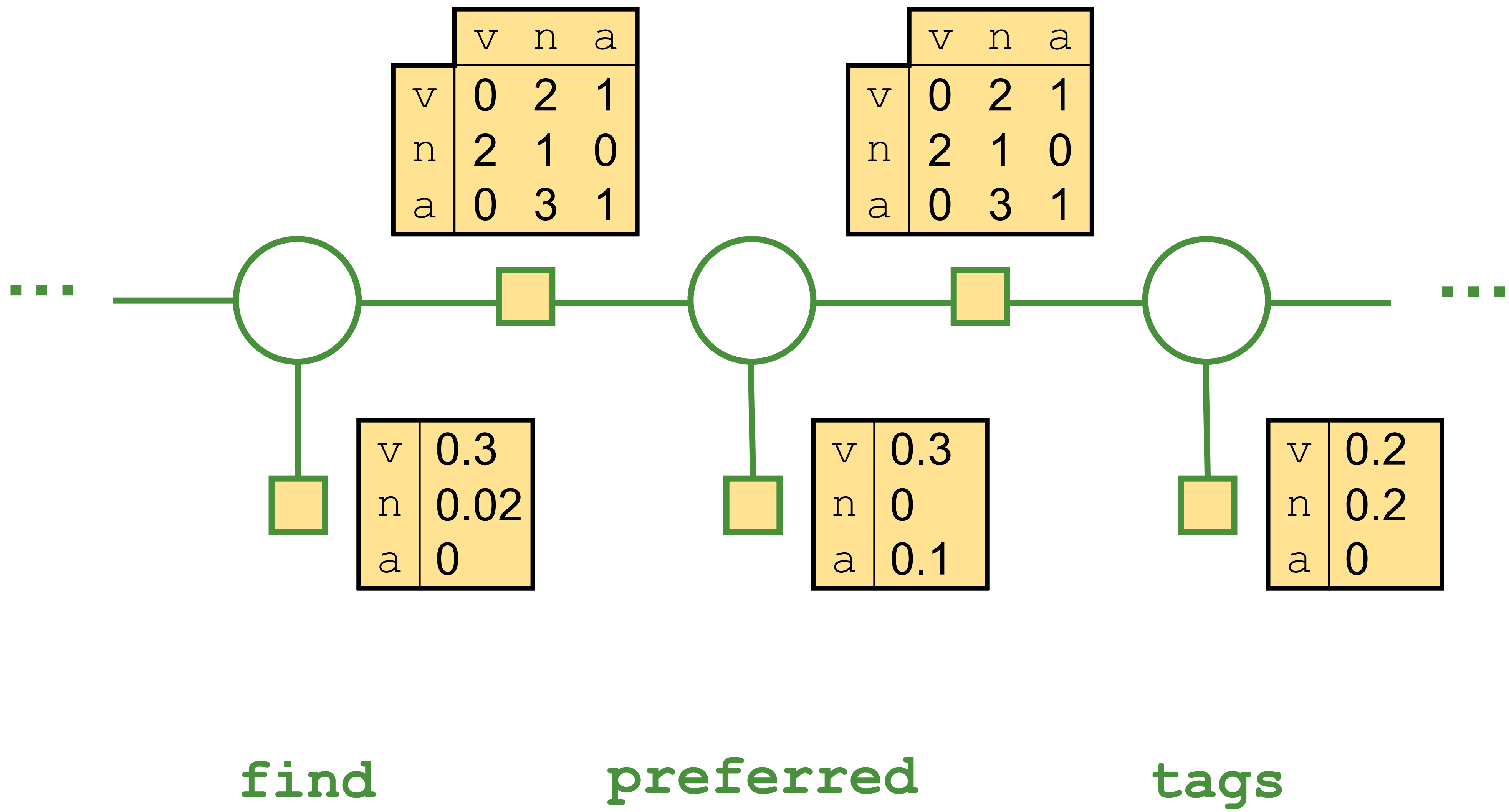
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

“Unary” factor evaluates this tag
Different unary factor at each position



Local factors in a graphical model

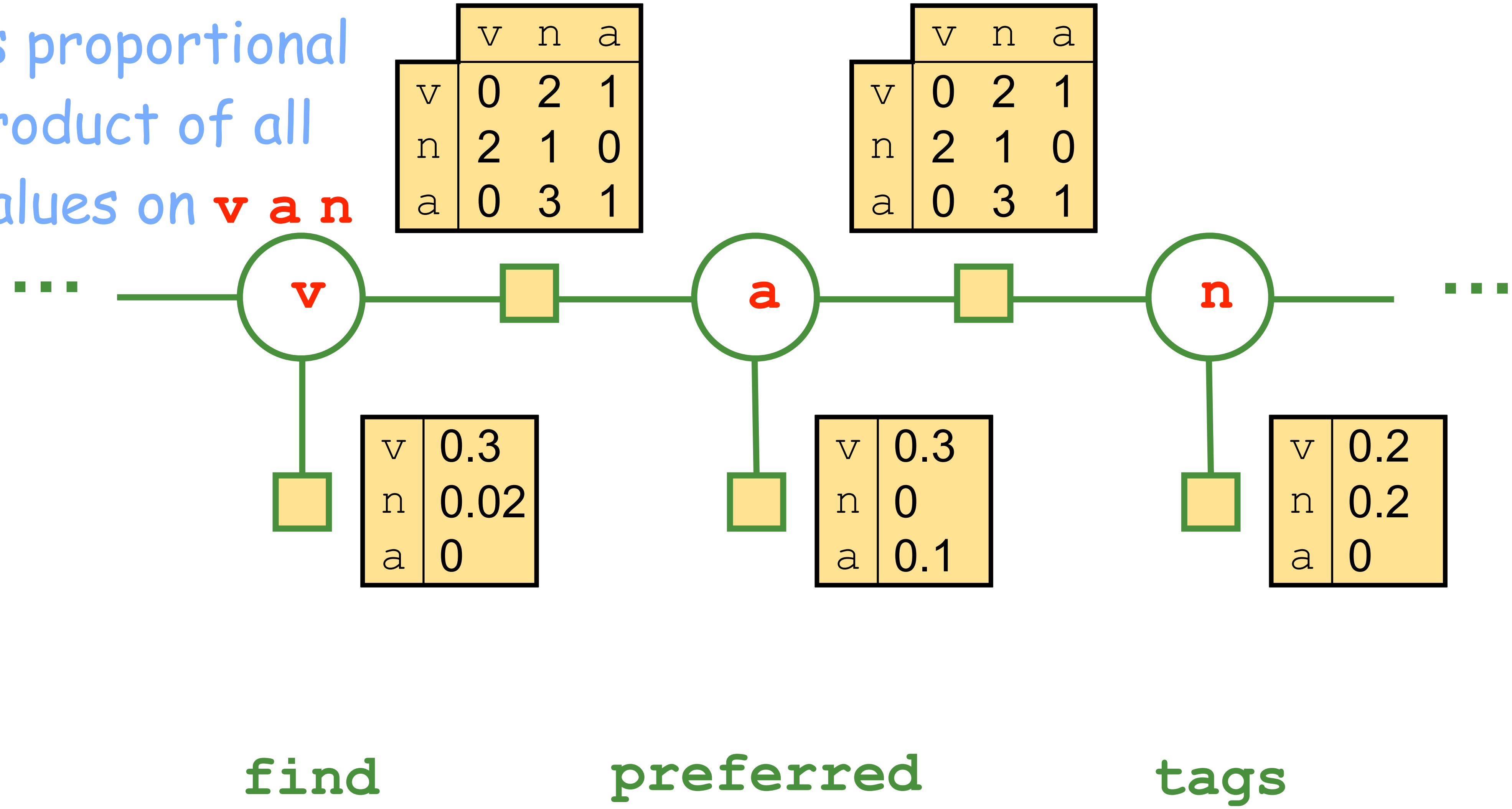
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

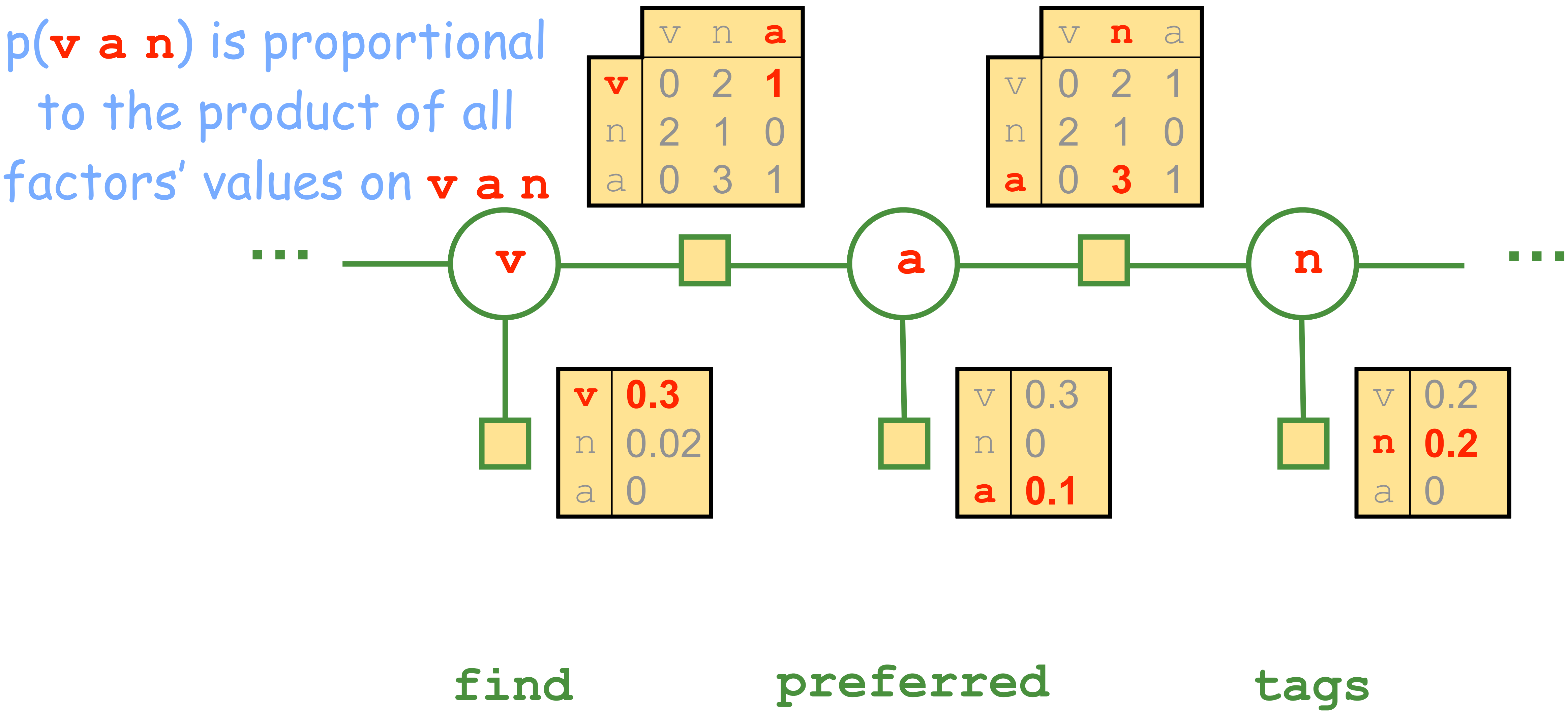
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

p(**v a n**) is proportional to the product of all factors' values on **v a n**



Local factors in a graphical model

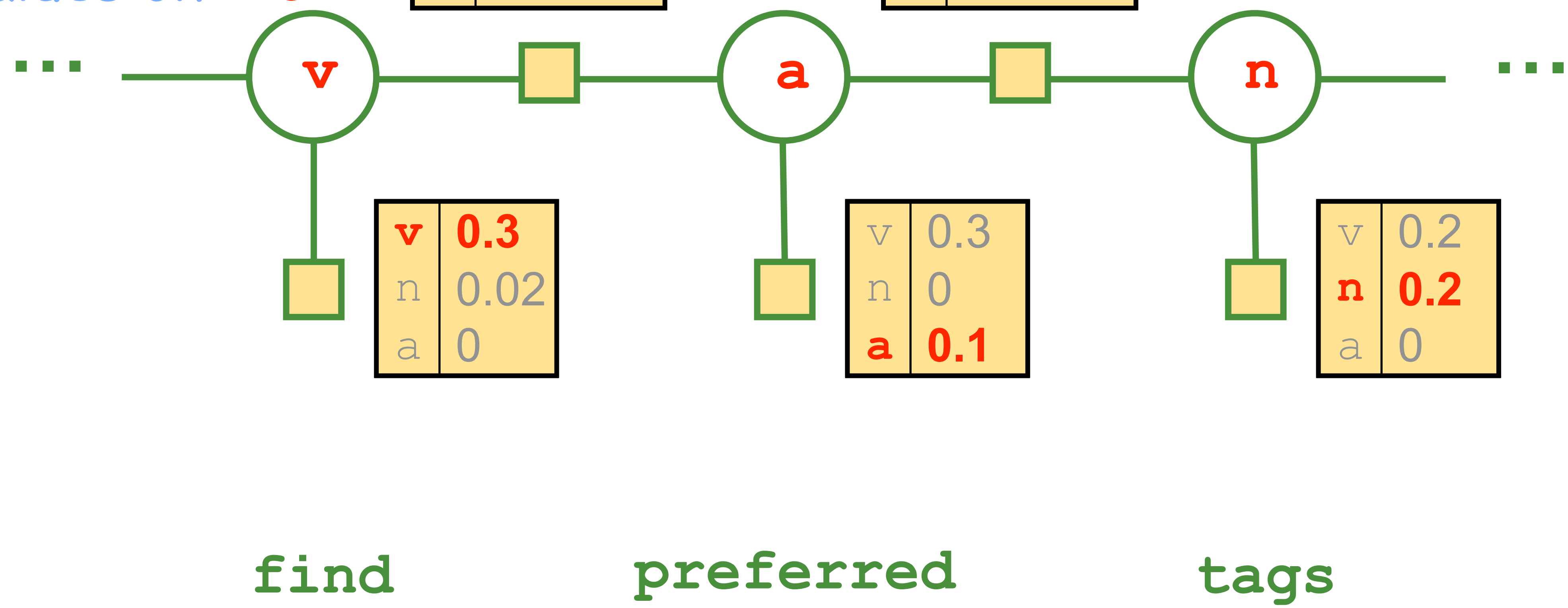
- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging



Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

p(**v a n**) is proportional to the product of all factors' values on **v a n**



	v	n	a
v	0	2	1
n	2	1	0
a	0	3	1

	v	n	a
v	0	2	1
n	2	1	0
a	0	3	1

v	0.3
n	0.02
a	0

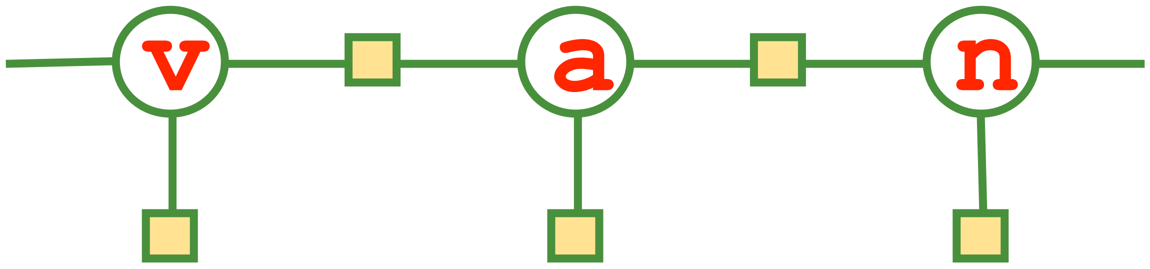
v	0.3
n	0
a	0.1

v	0.2
n	0.2
a	0

= … 1*3*0.3*0.1*0.2 …

Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

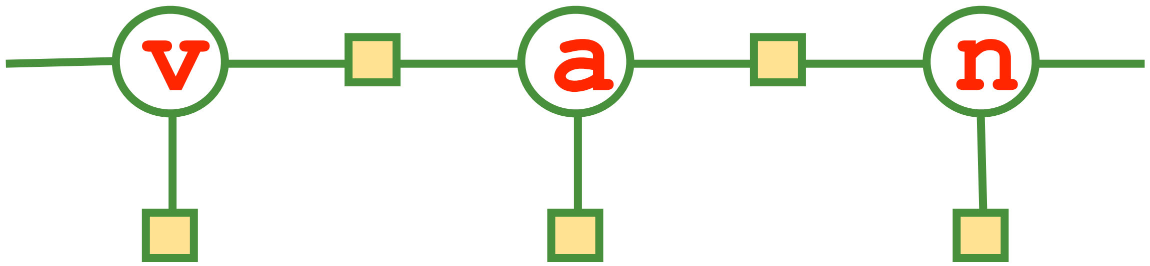


- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

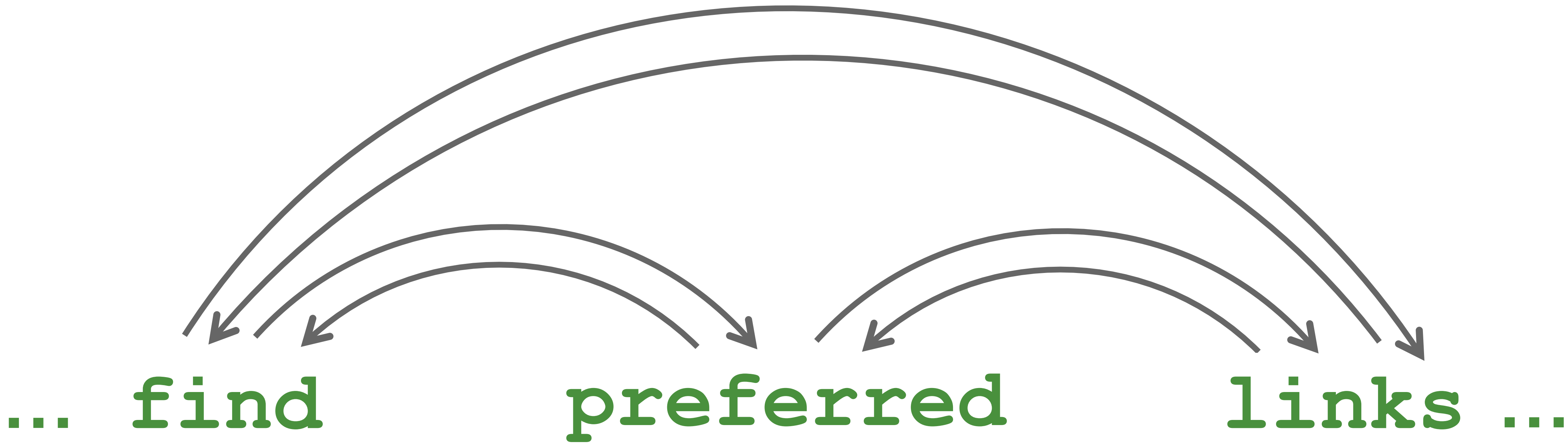
... find preferred links ...

Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

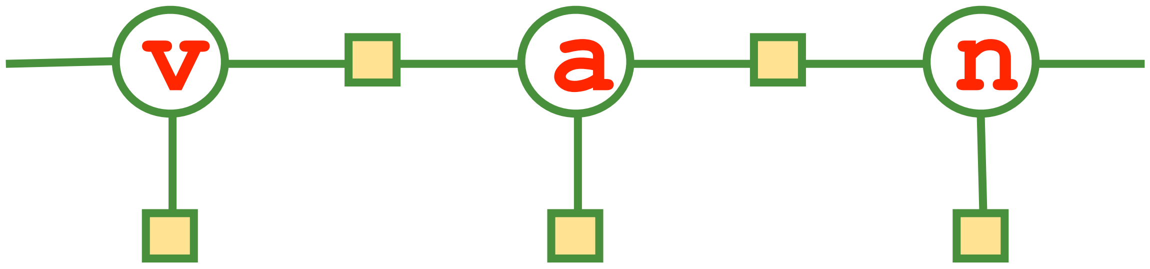


- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

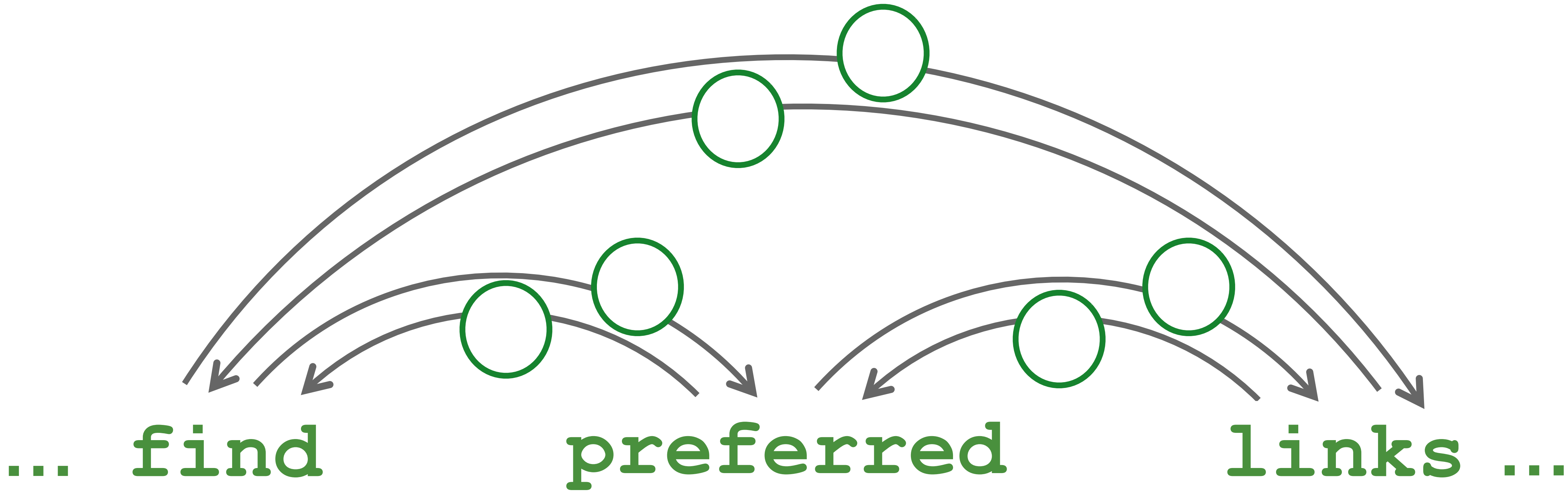


Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

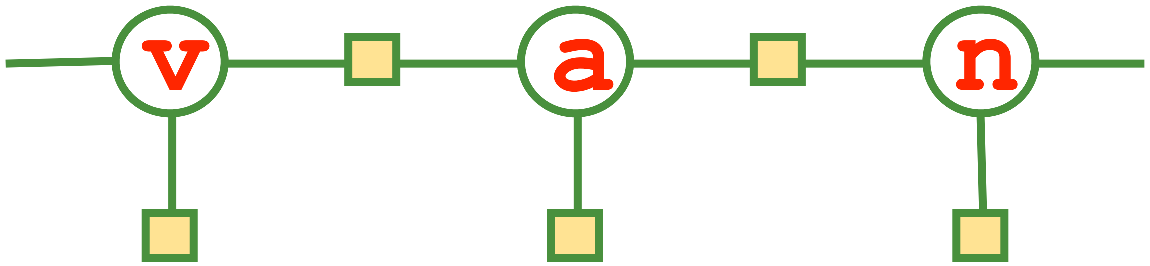


- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links



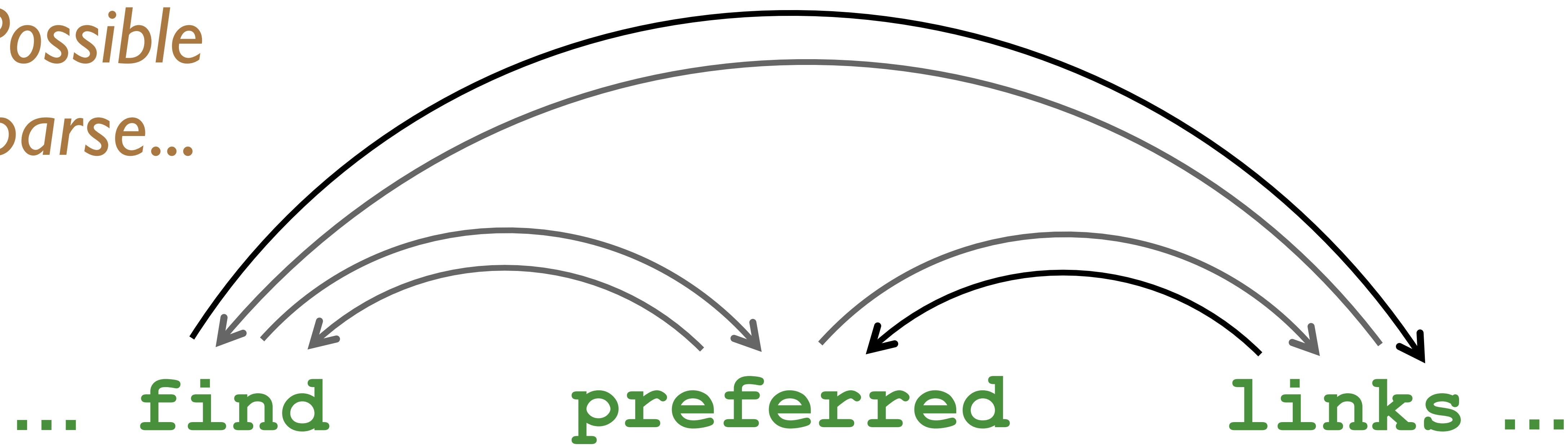
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging



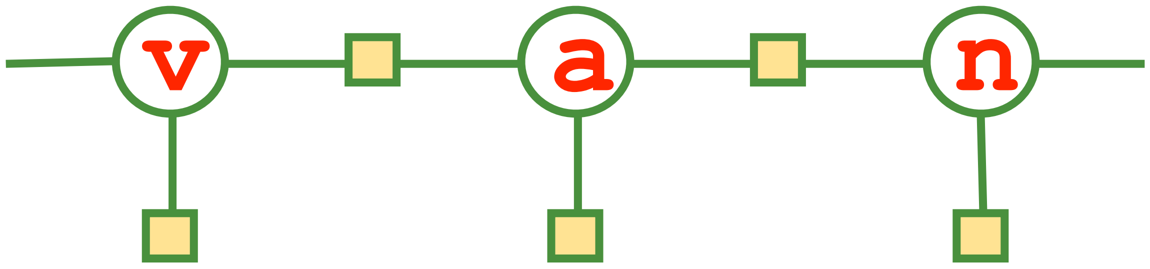
- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

Possible parse…



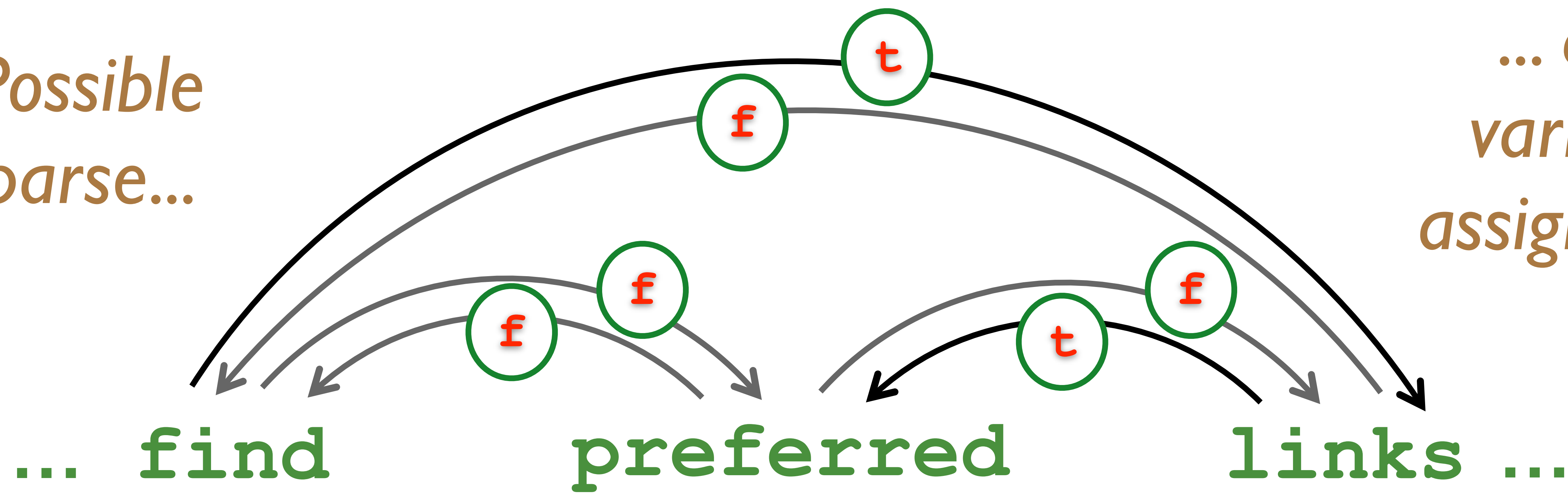
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging



- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

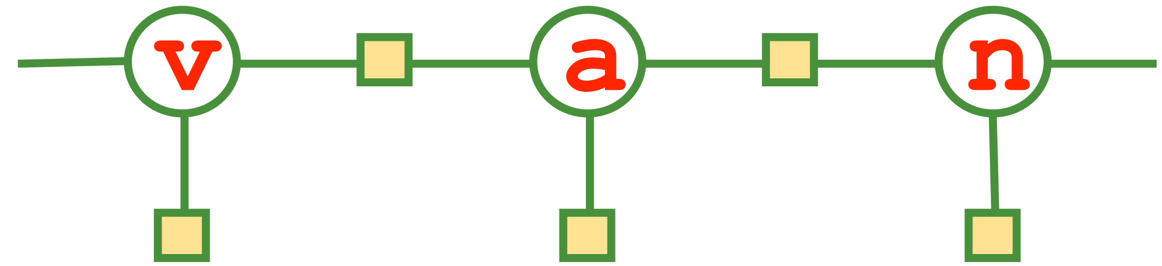
Possible parse...



... and variable assignment

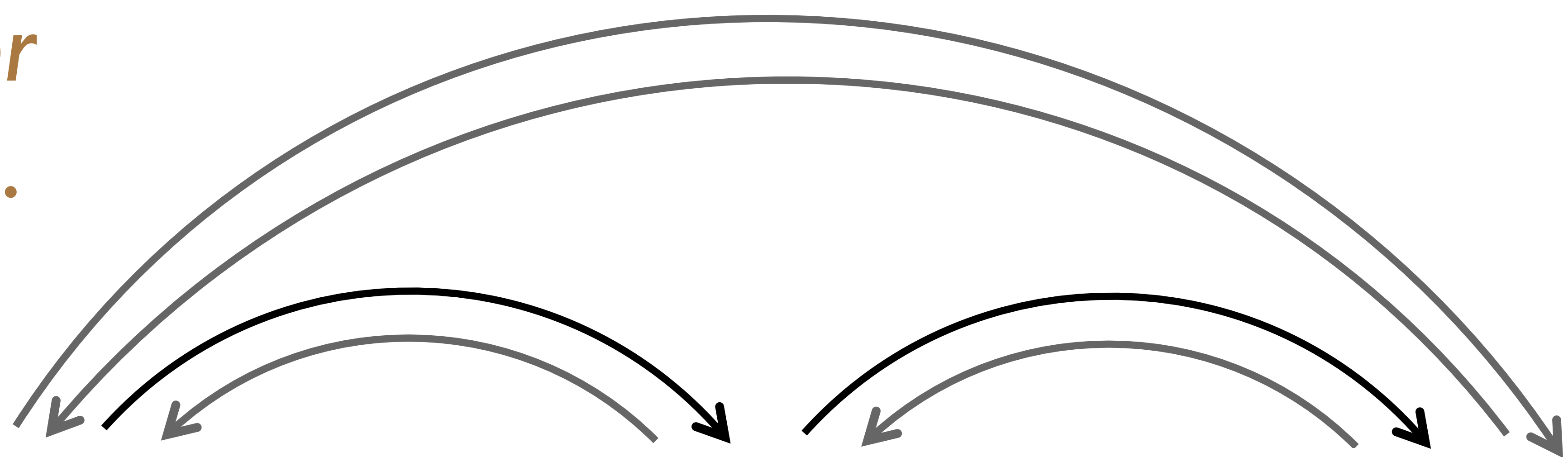
Graphical Models for Parsing

- First, a labeling example
  - ✤ CRF for POS tagging



- Now let's do dependency parsing!
  - ✤ $O(n^2)$ boolean variables for the possible links

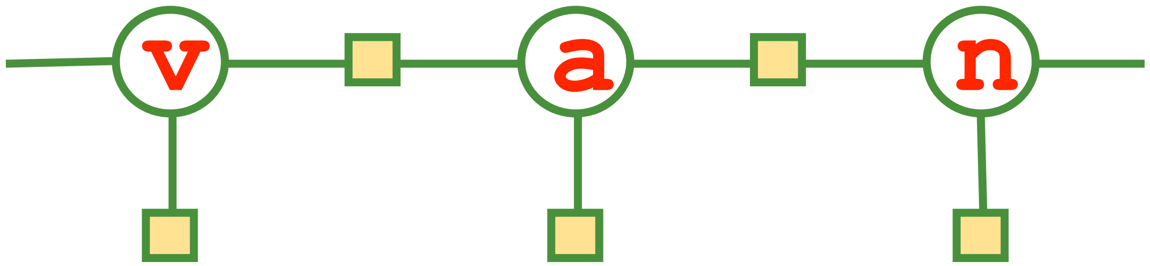
Another parse...



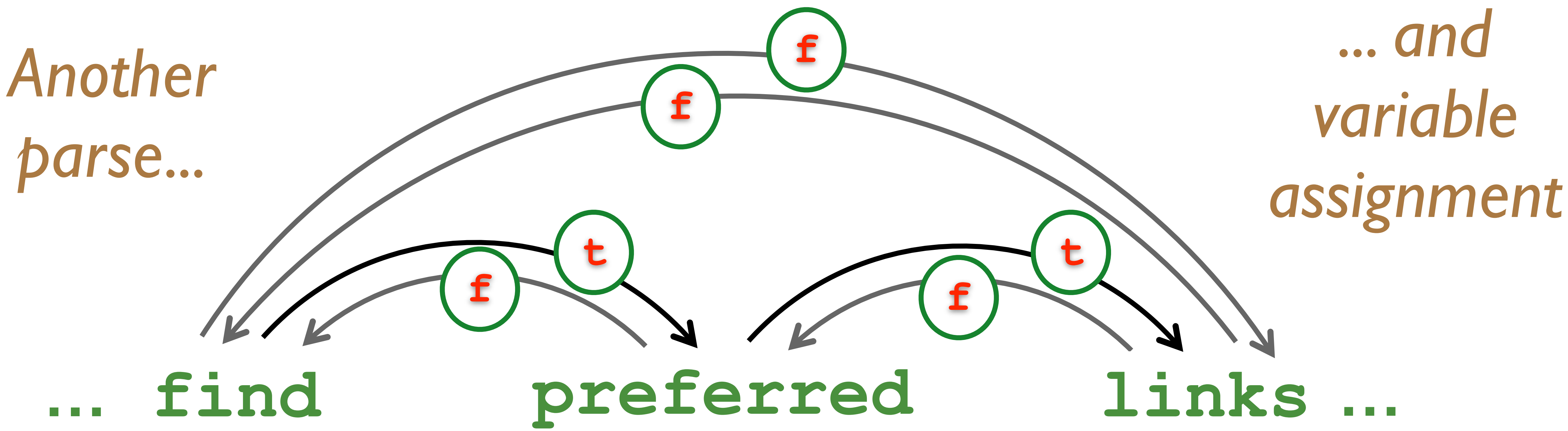
… find preferred links …

Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

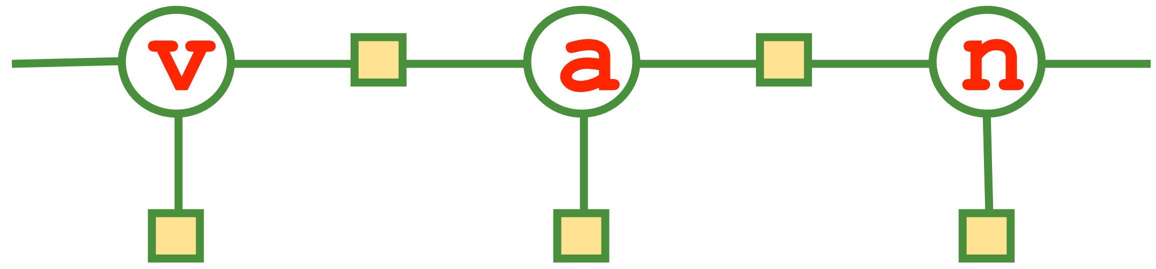


- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

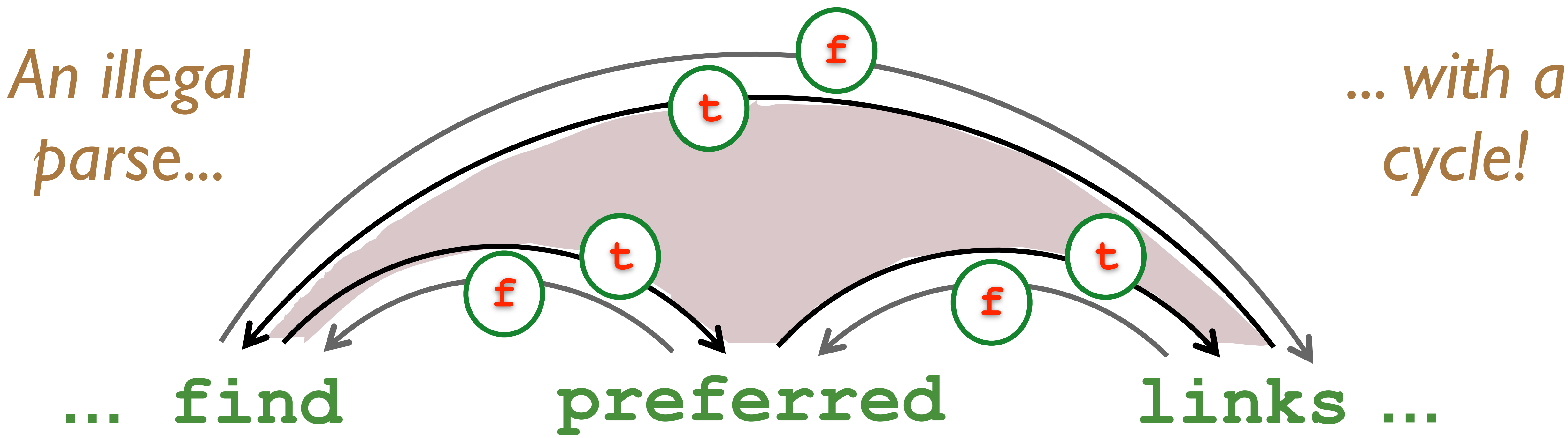


Graphical Models for Parsing

- First, a labeling example
  - ✤ CRF for POS tagging

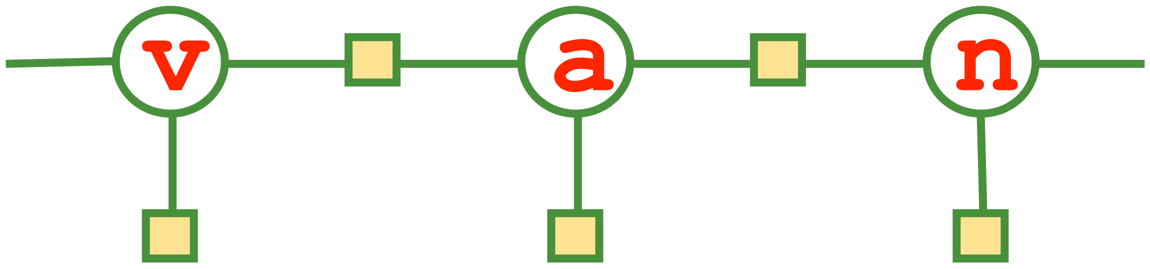


- Now let's do dependency parsing!
  - ✤ $O(n^2)$ boolean variables for the possible links

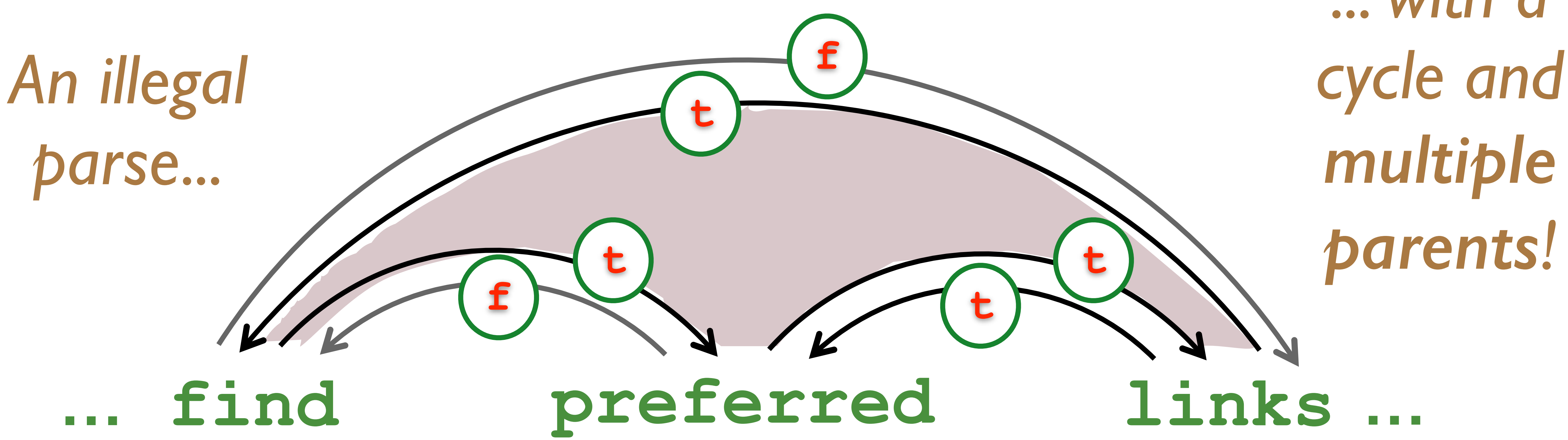


Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

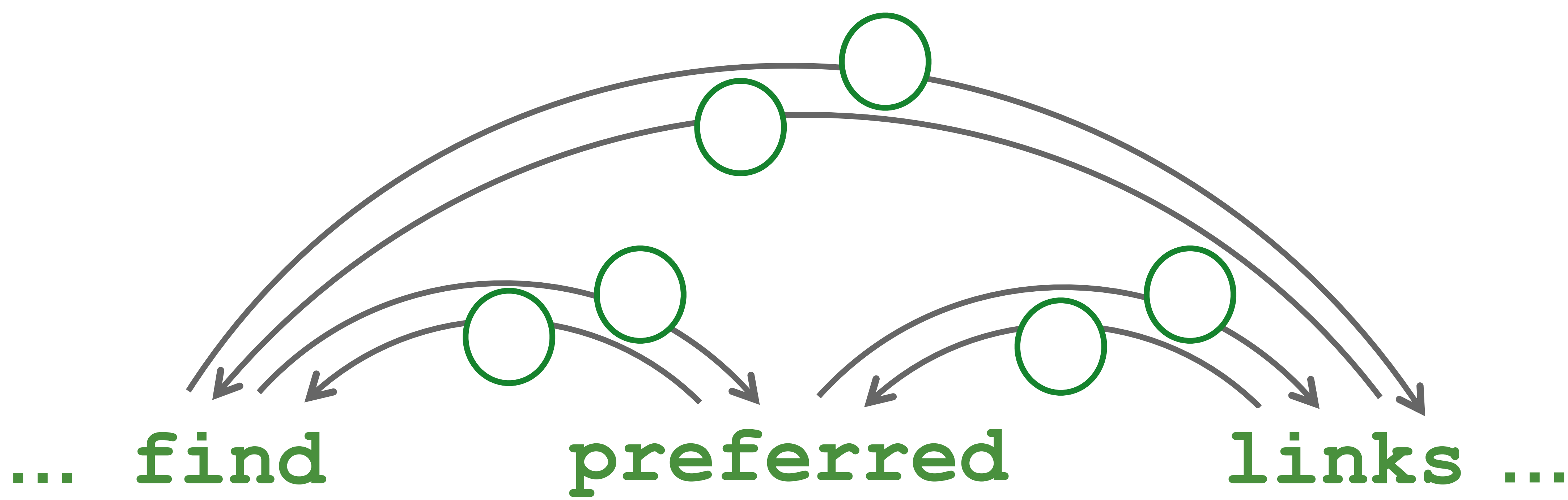


- Now let's do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links



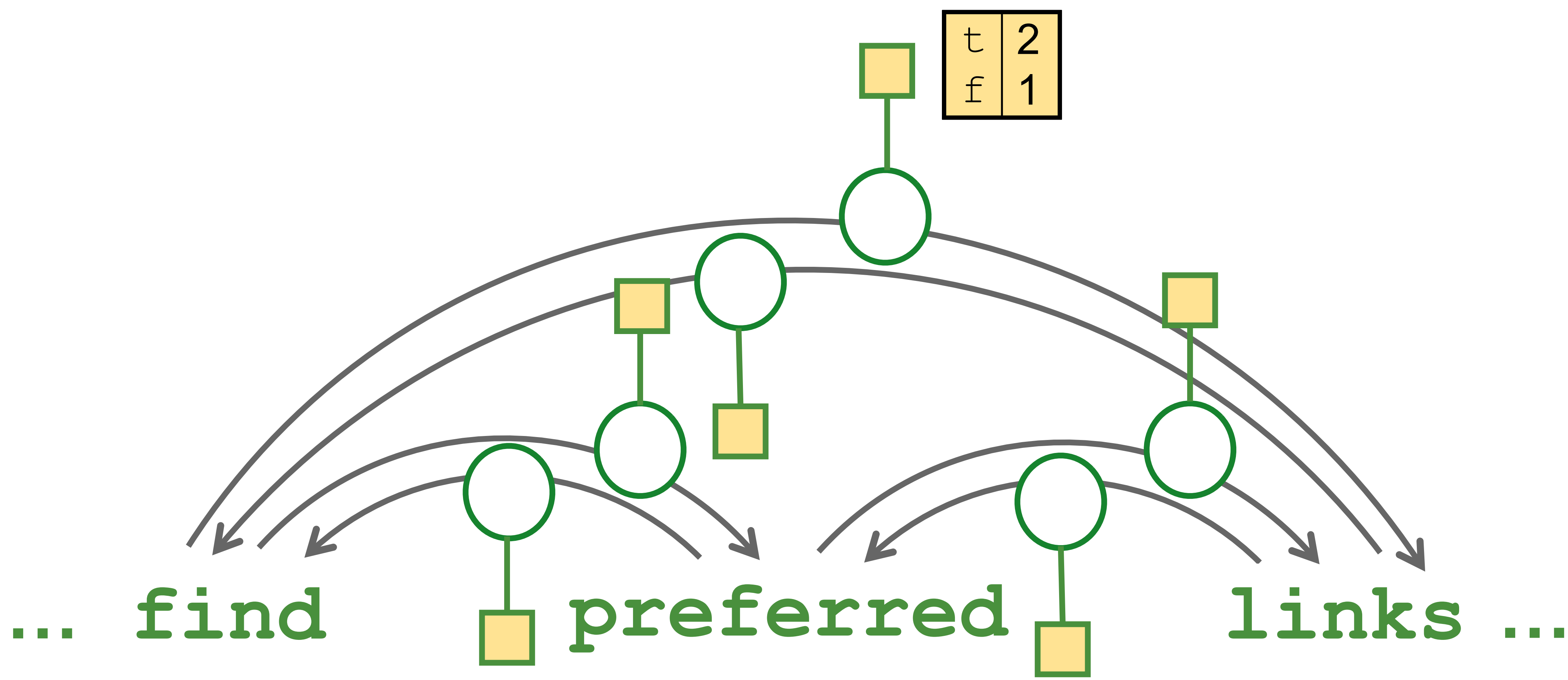
Local Factors for Parsing

- What factors determine parse probability?



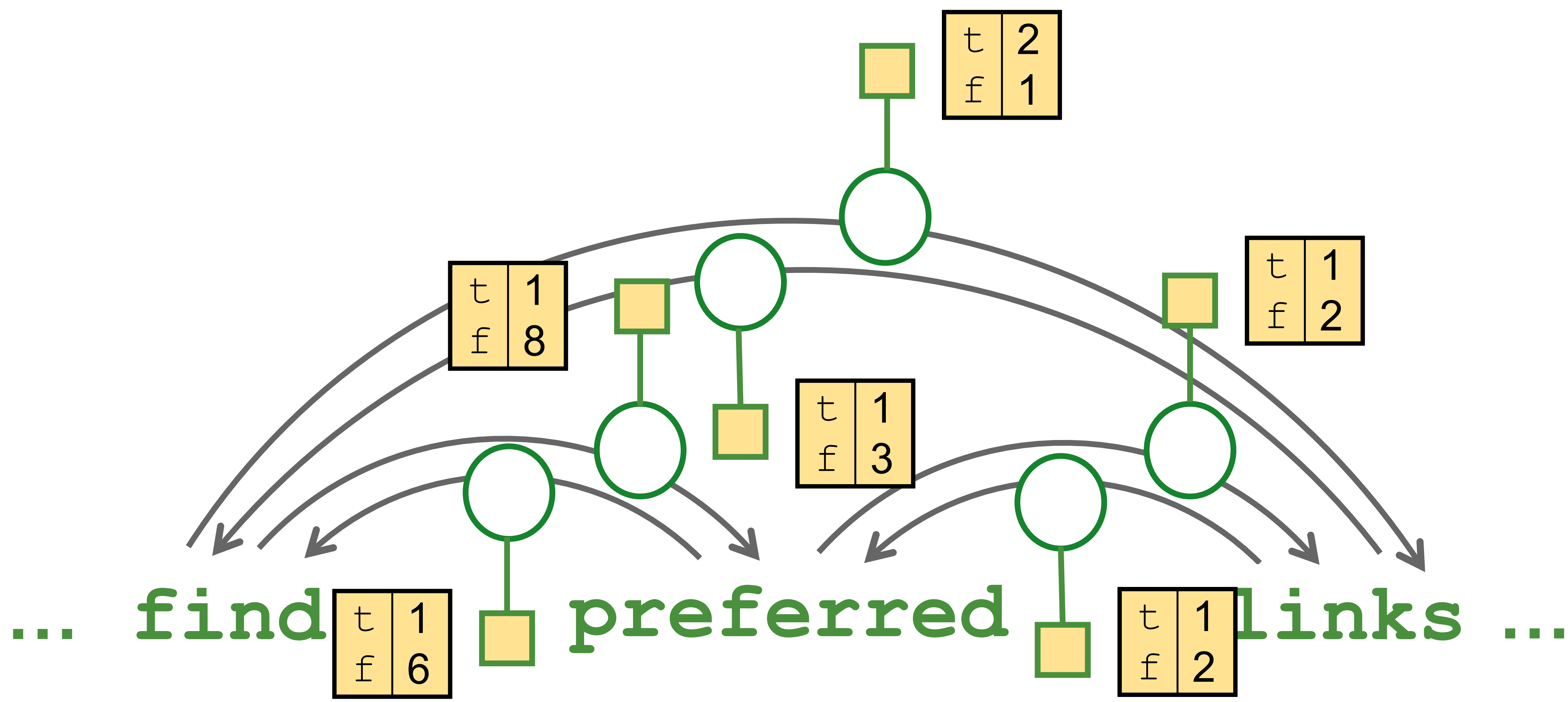
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation



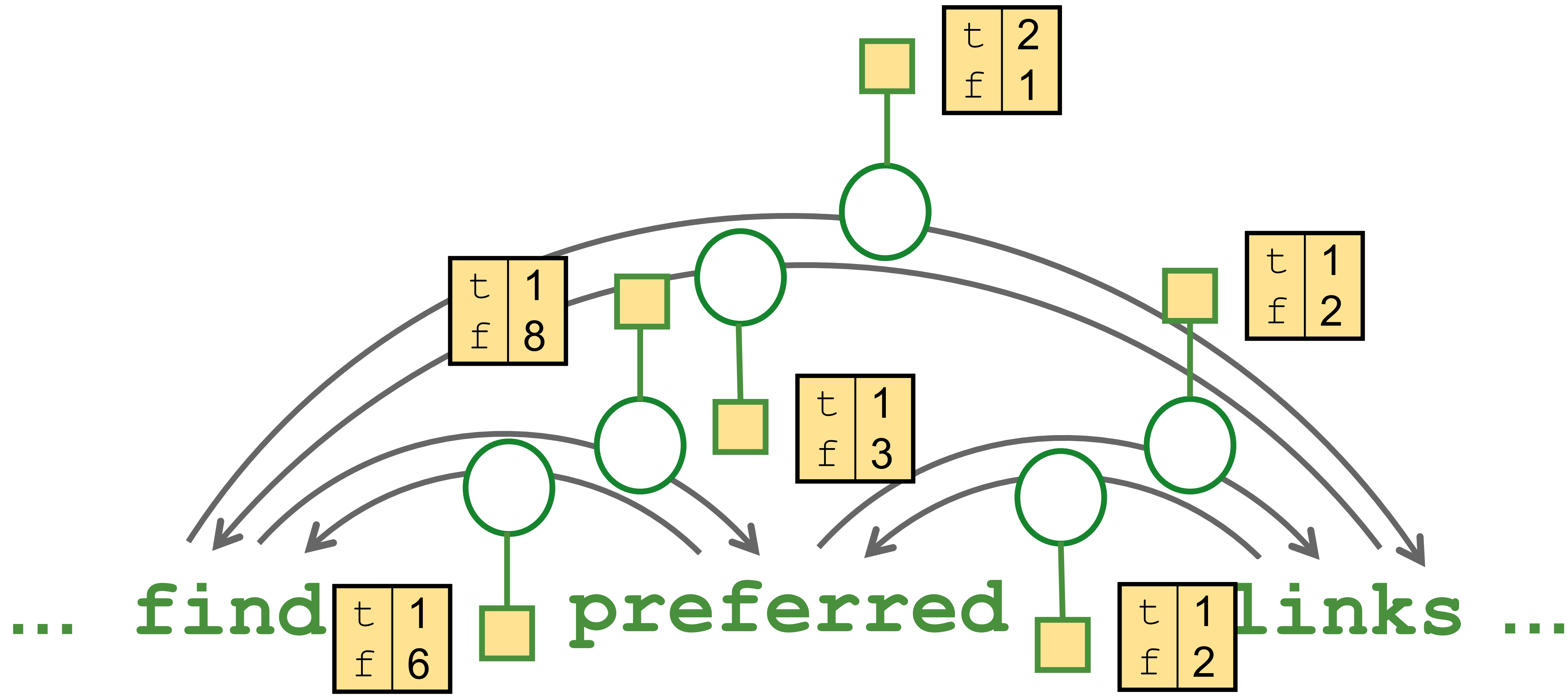
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation



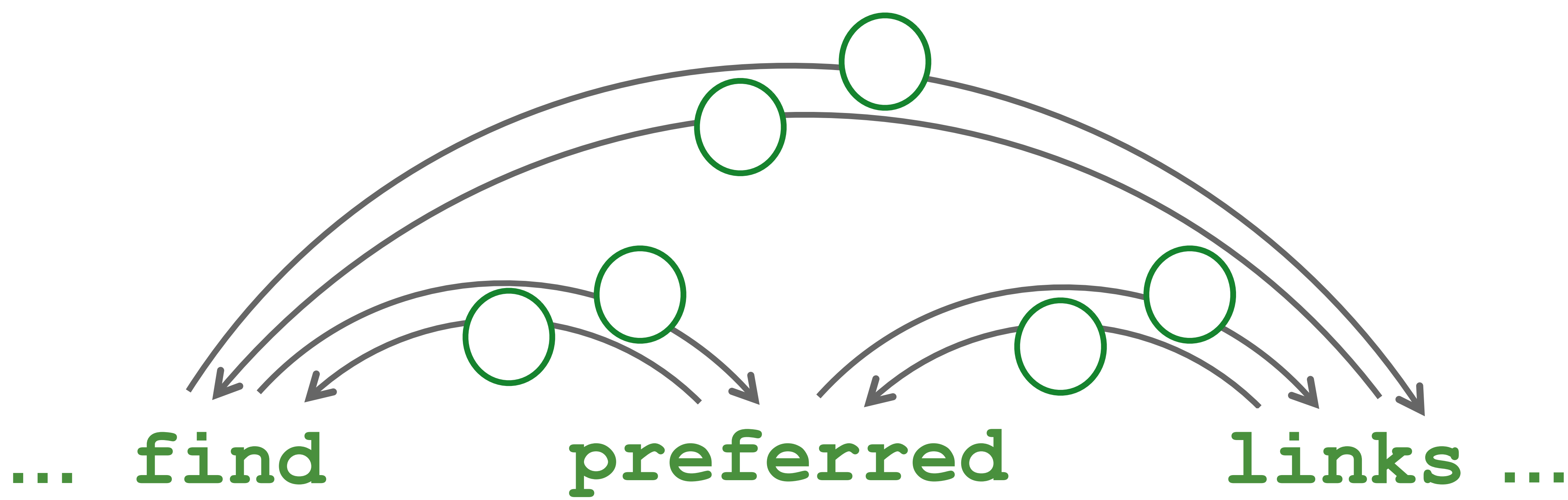
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
- But what if the best assignment isn't a tree?



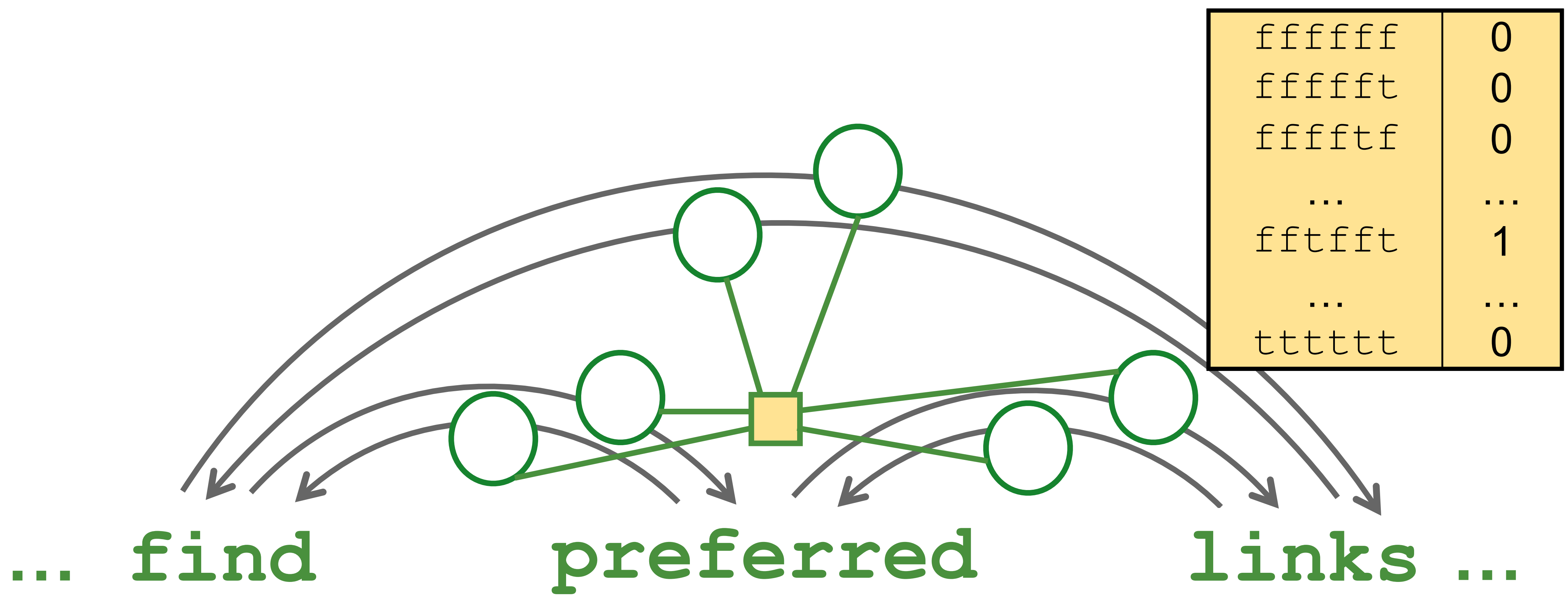
Global Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation



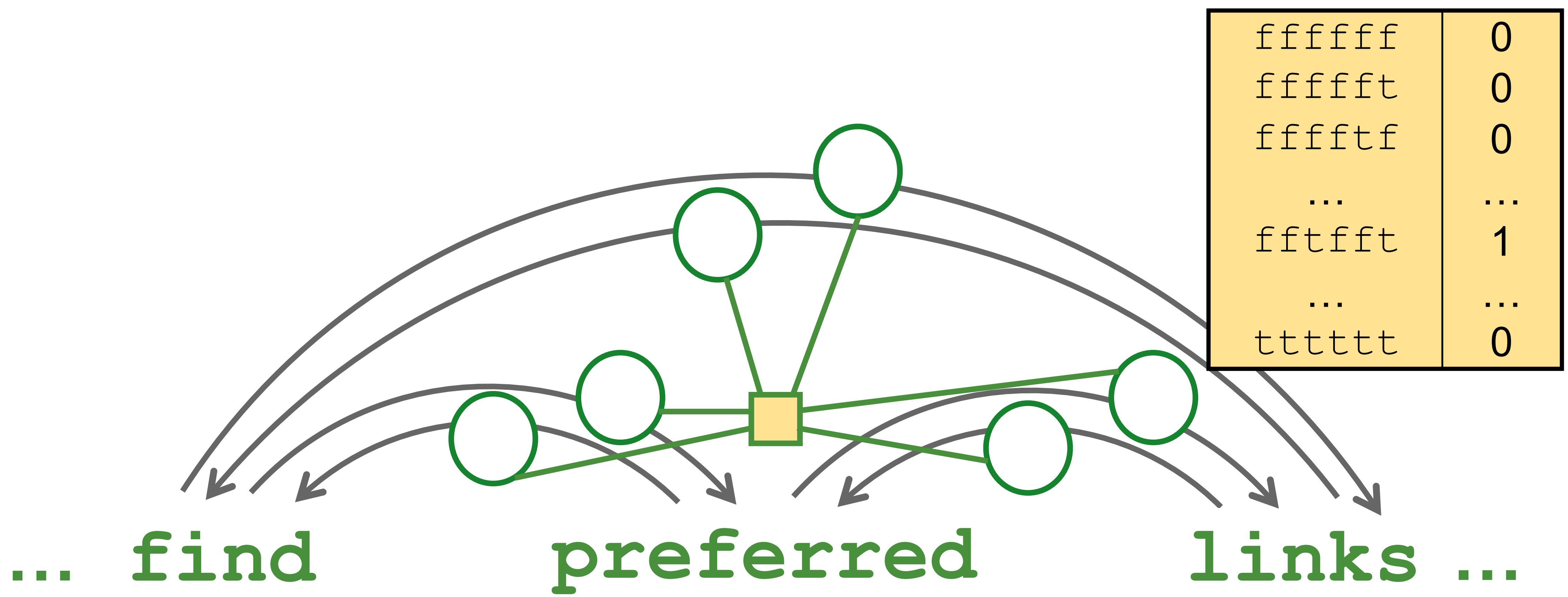
Global Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to *require* links to form a legal tree



Global Factors for Parsing

- What factors determine parse probability?
  - ✤ Unary factors to score each link in isolation
  - ✤ Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1

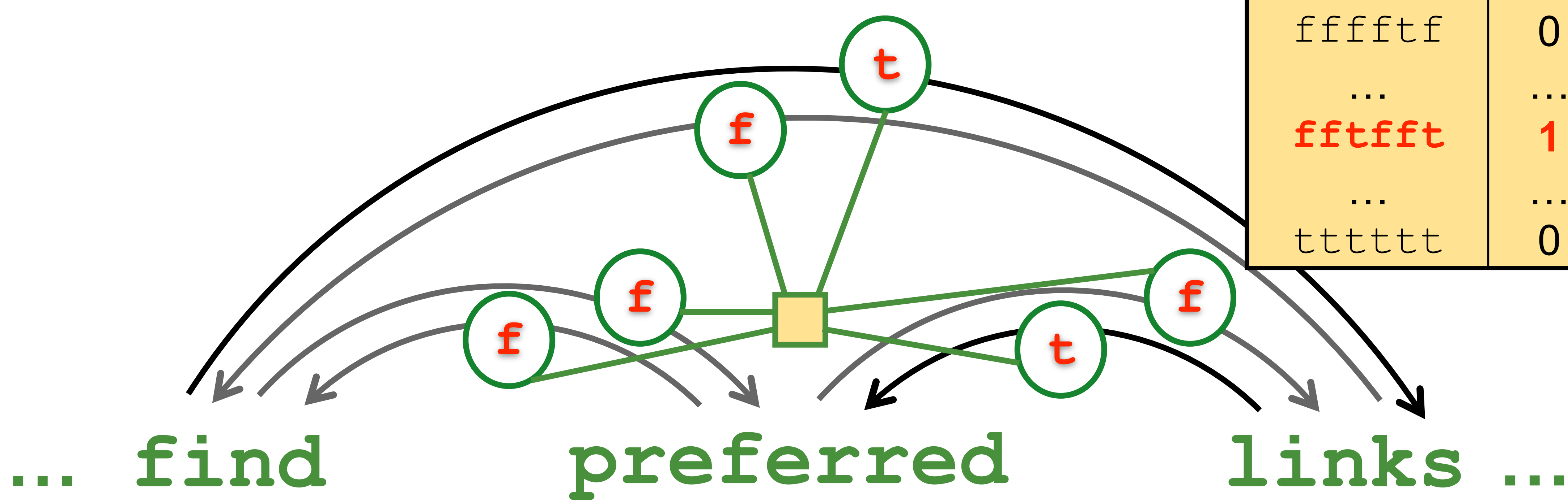


Global Factors for Parsing

- What factors determine parse probability?
  - ✤ Unary factors to score each link in isolation
  - ✤ Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1

ffffff	0
ffffft	0
fffftf	0
…	…
fftfft	**1**
…	…
tttttt	0

we're legal!



Global Factors for Parsing

optionally require the tree to be projective (no crossing links)

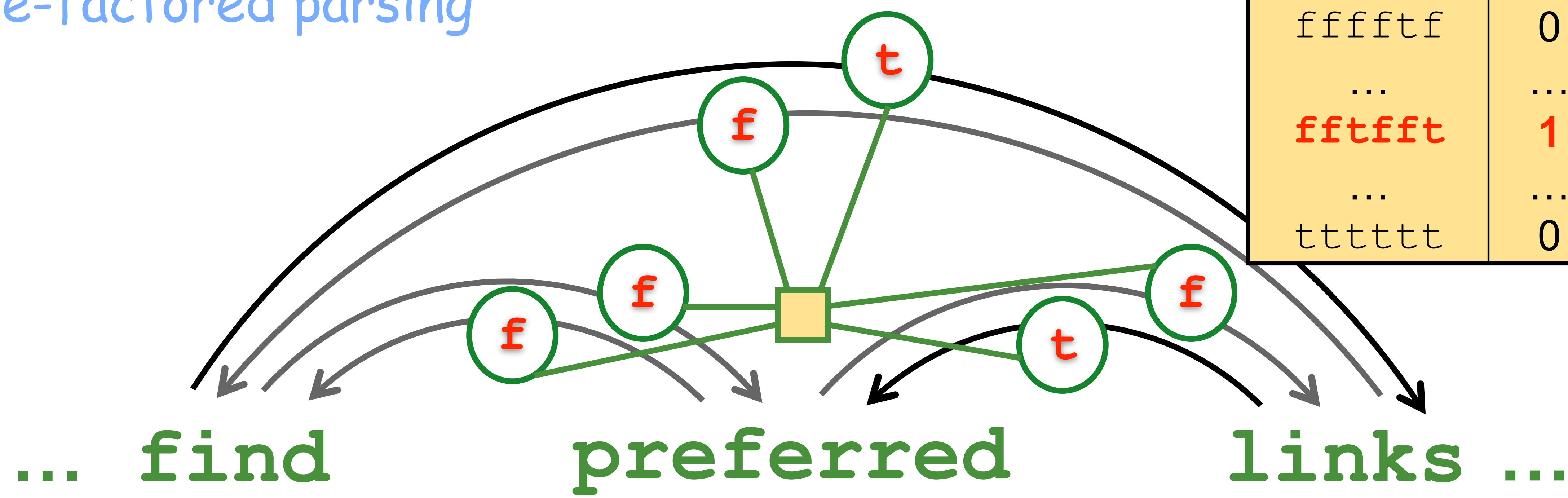
- What factors determine parse
  - Unary factors to score each link in isolation
  - Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1

So far, this is equivalent to edge-factored parsing

64 entries (0/1)

ffffff	0
ffffft	0
fffftf	0
…	…
fftfft	1
…	…
tttttt	0

we're legal!



Global Factors for Parsing

- What factors determine parse

optionally require the tree to be projective (no crossing links)

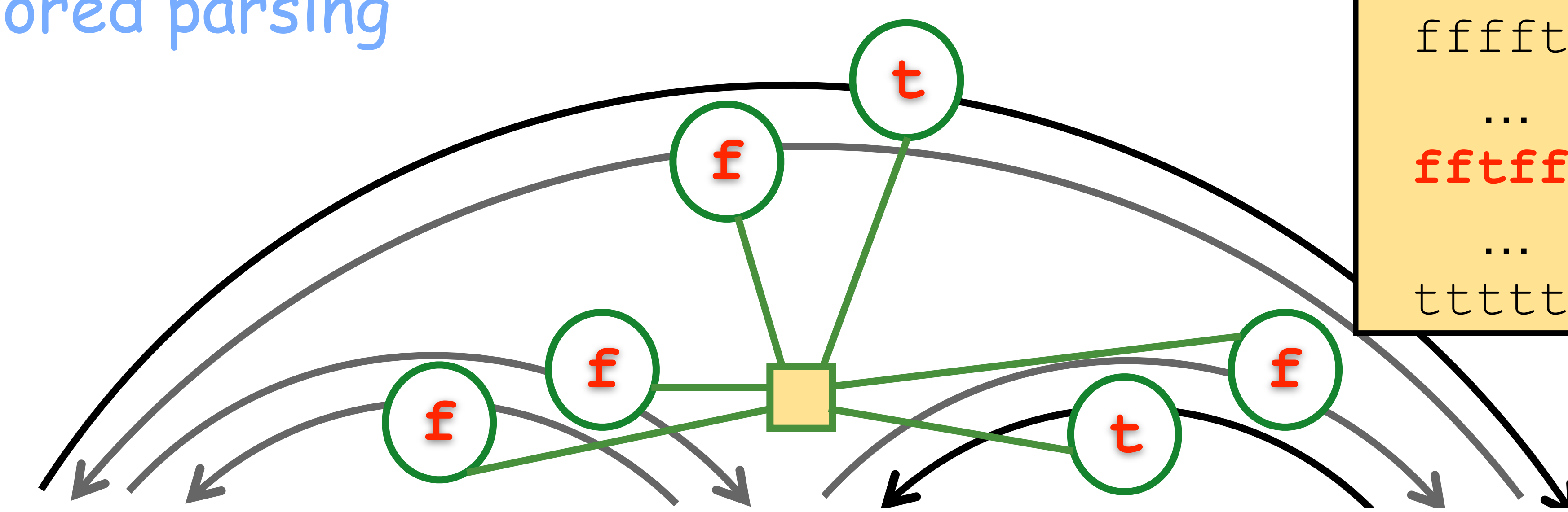
- ✤ Unary factors to score each link in isolation
- ✤ Global TREE factor to *require* links to form a legal tree
  - A *hard constraint*: potential is either 0 or 1

So far, this is equivalent to edge-factored parsing

64 entries (0/1)

ffffff	0
ffffft	0
fffftf	0
…	…
fftfft	**1**
…	…
tttttt	0

we're legal!

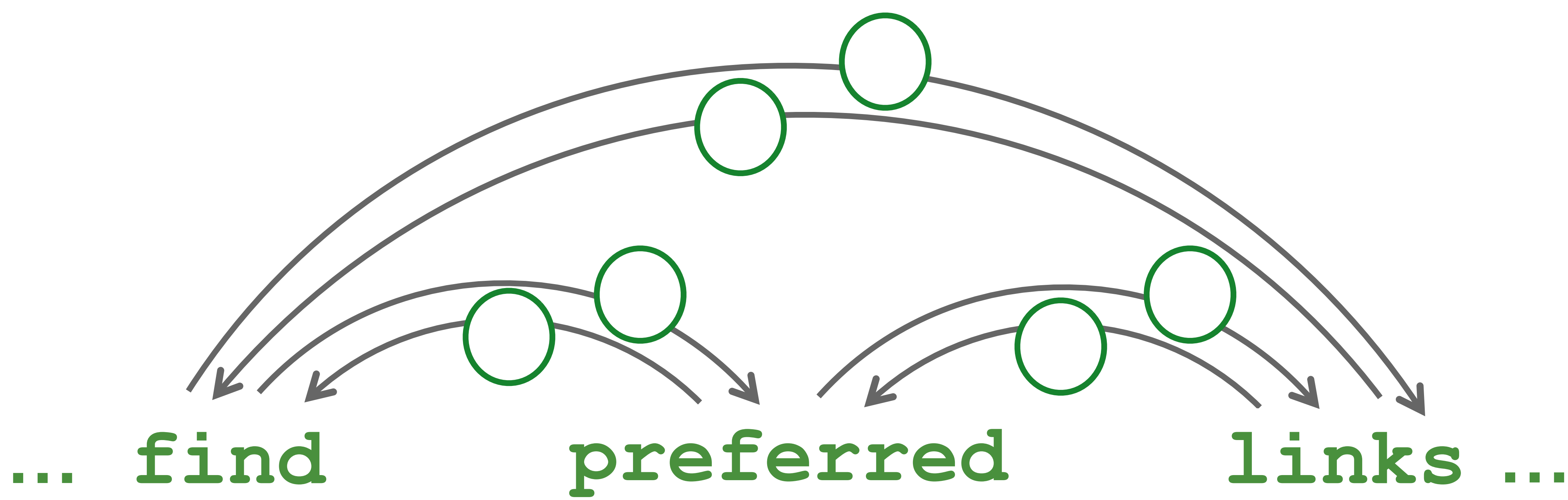


… fin

Note: traditional parsers don't loop through this table to consider exponentially many trees one at a time. They use combinatorial algorithms; so should we!

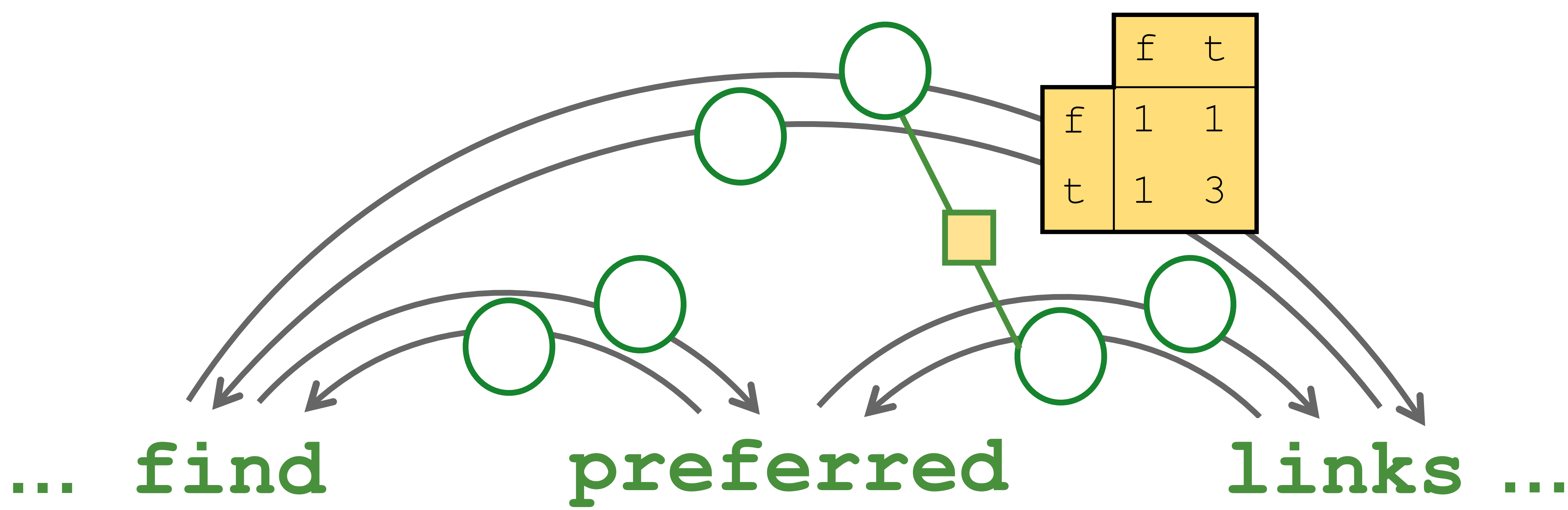
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1
  - Second order effects: factors on 2 variables
    - Grandparent–parent–child chains



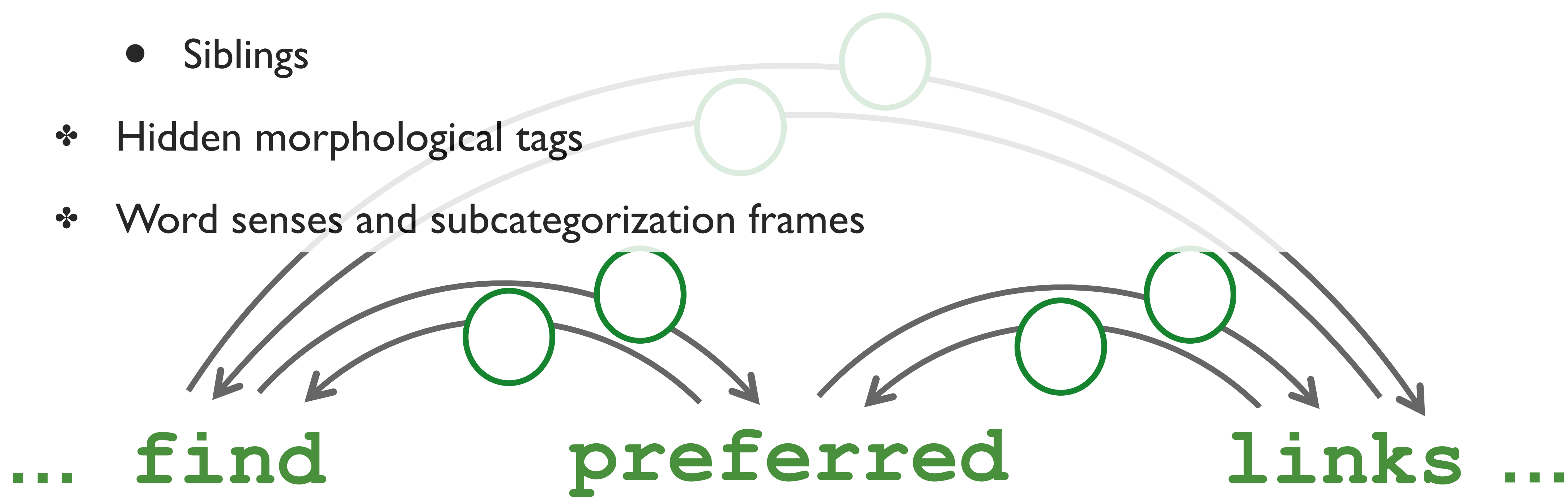
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1
  - Second order effects: factors on 2 variables
    - Grandparent–parent–child chains



Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to *require* links to form a legal tree
    - A *hard constraint*: potential is either 0 or 1
  - Second order effects: factors on 2 variables
    - Grandparent–parent–child chains
    - No crossing links
    - Siblings
  - Hidden morphological tags
  - Word senses and subcategorization frames



Great Ideas in ML: Message Passing

adapted from MacKay (2003) textbook

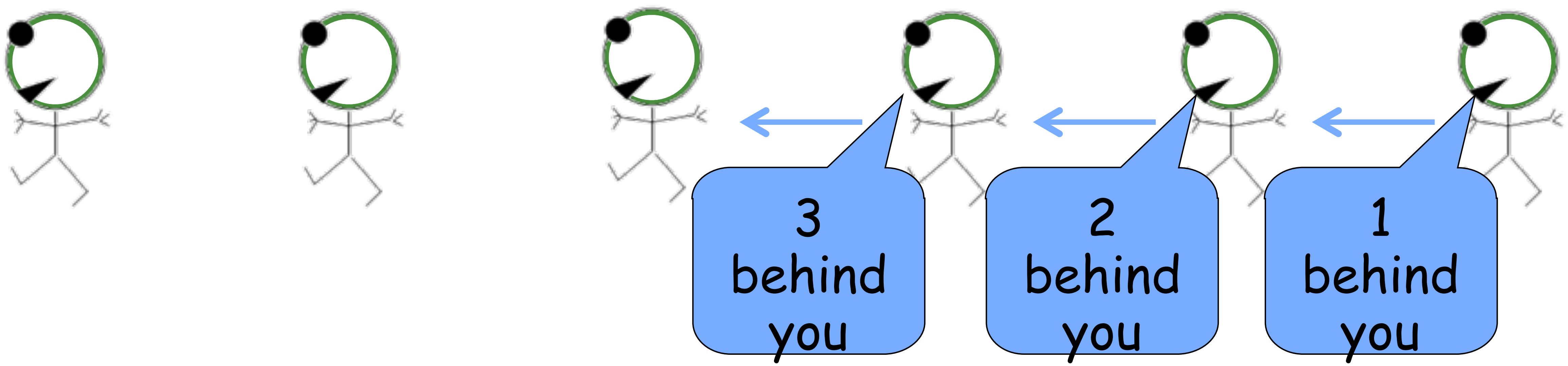
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

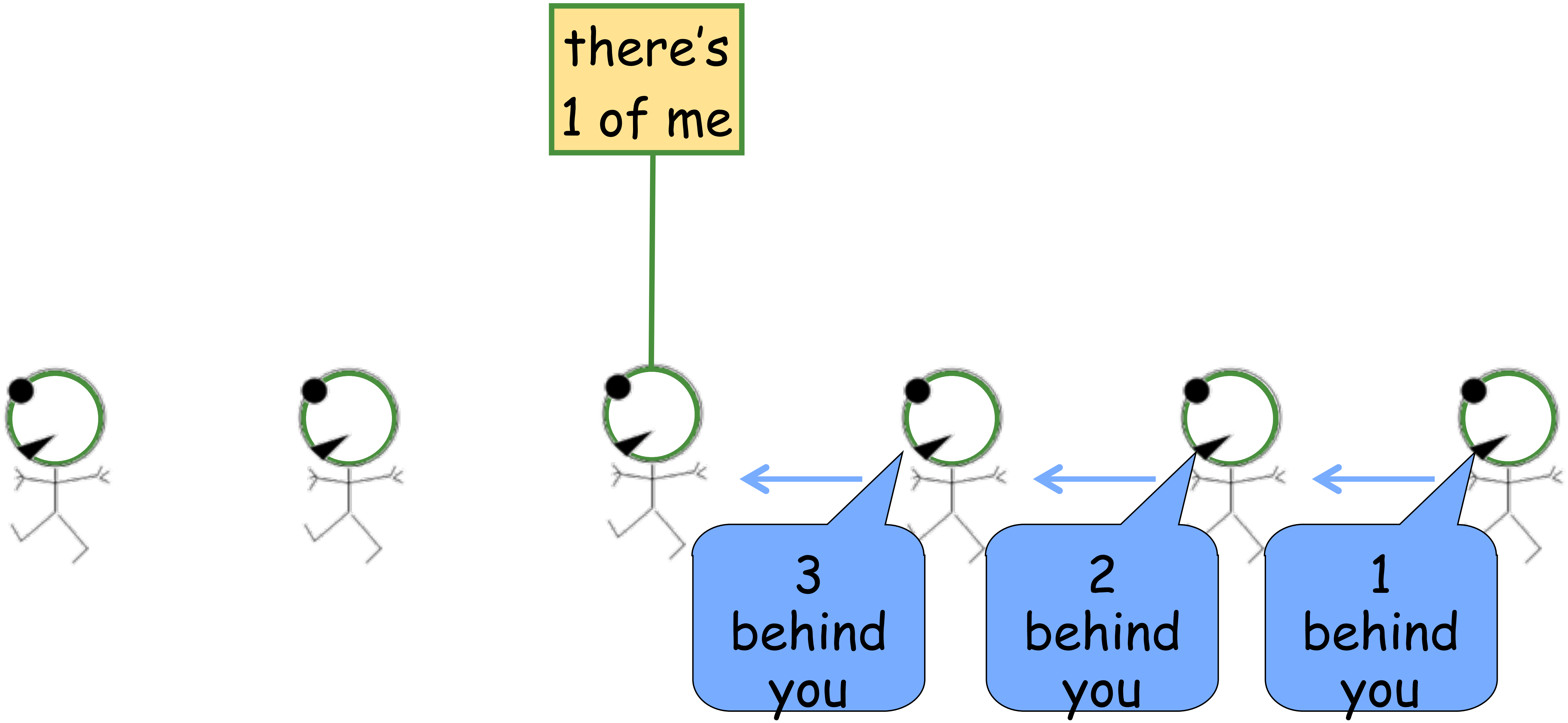
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

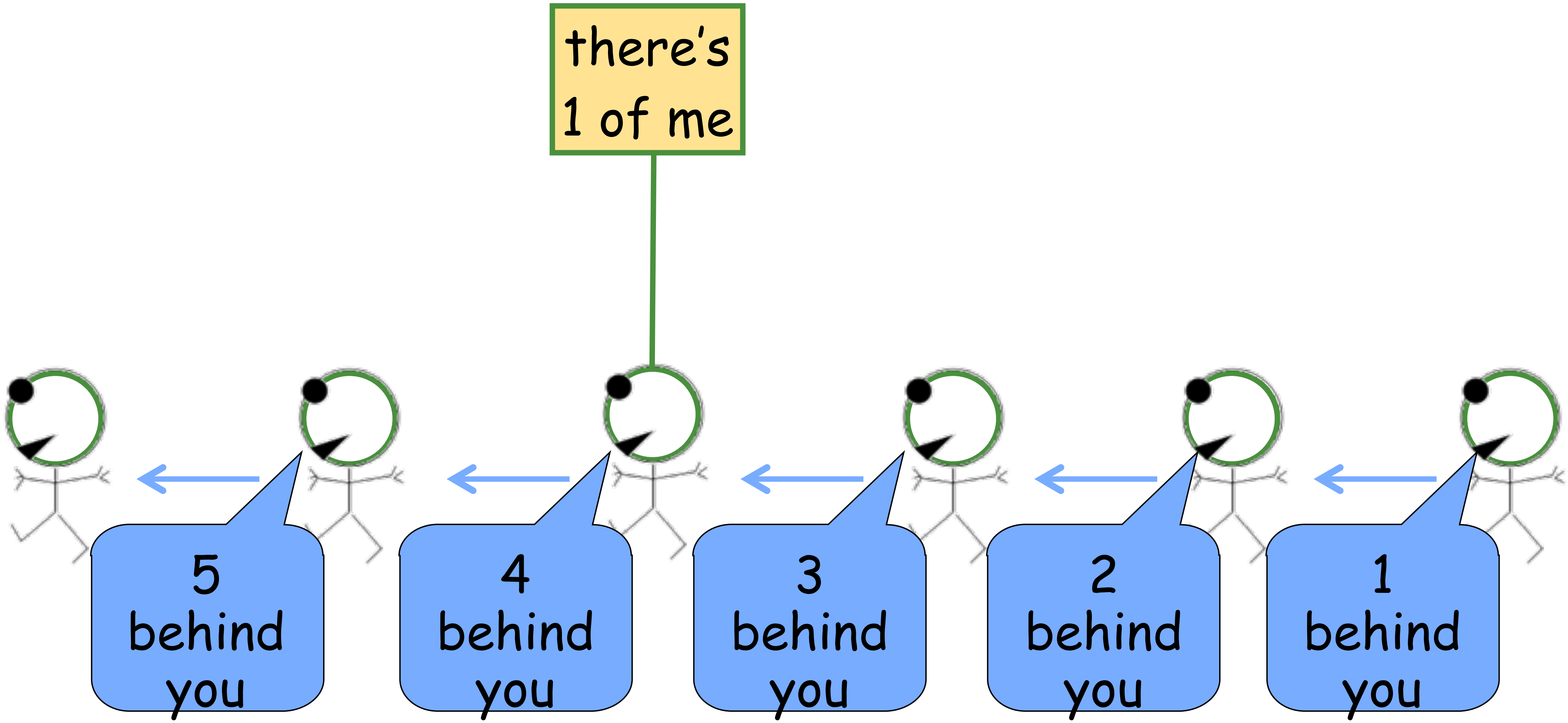
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

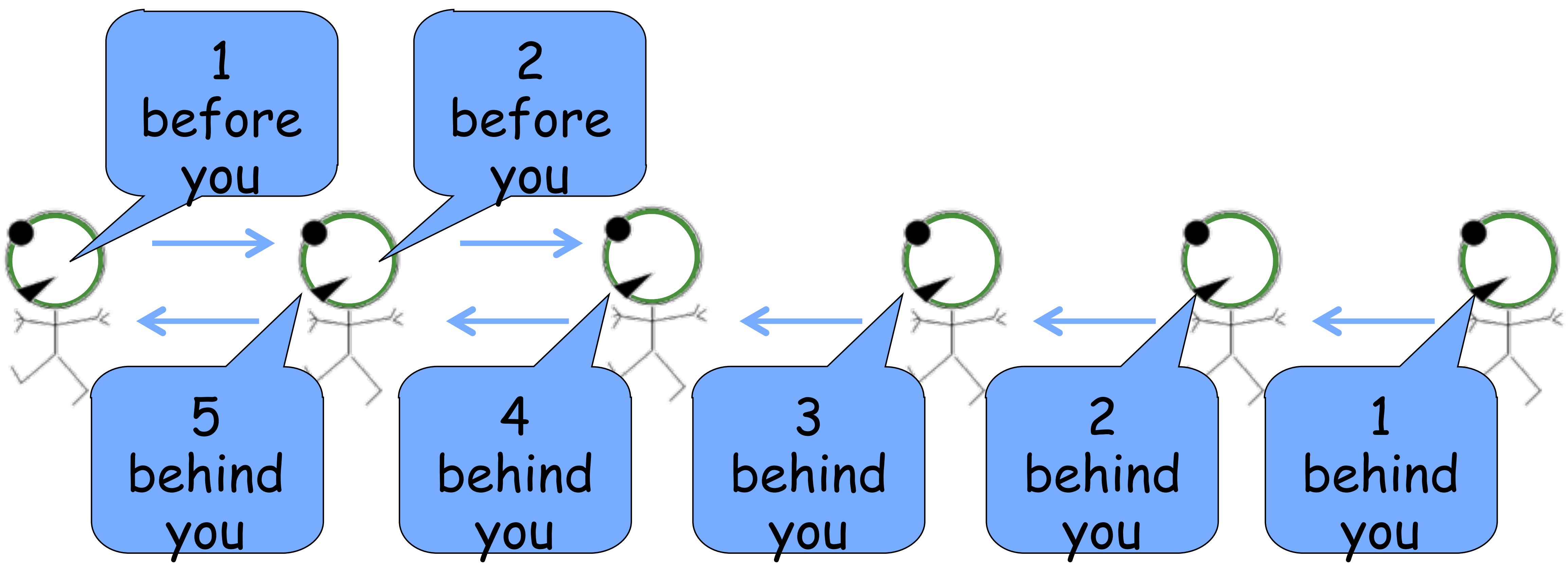
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

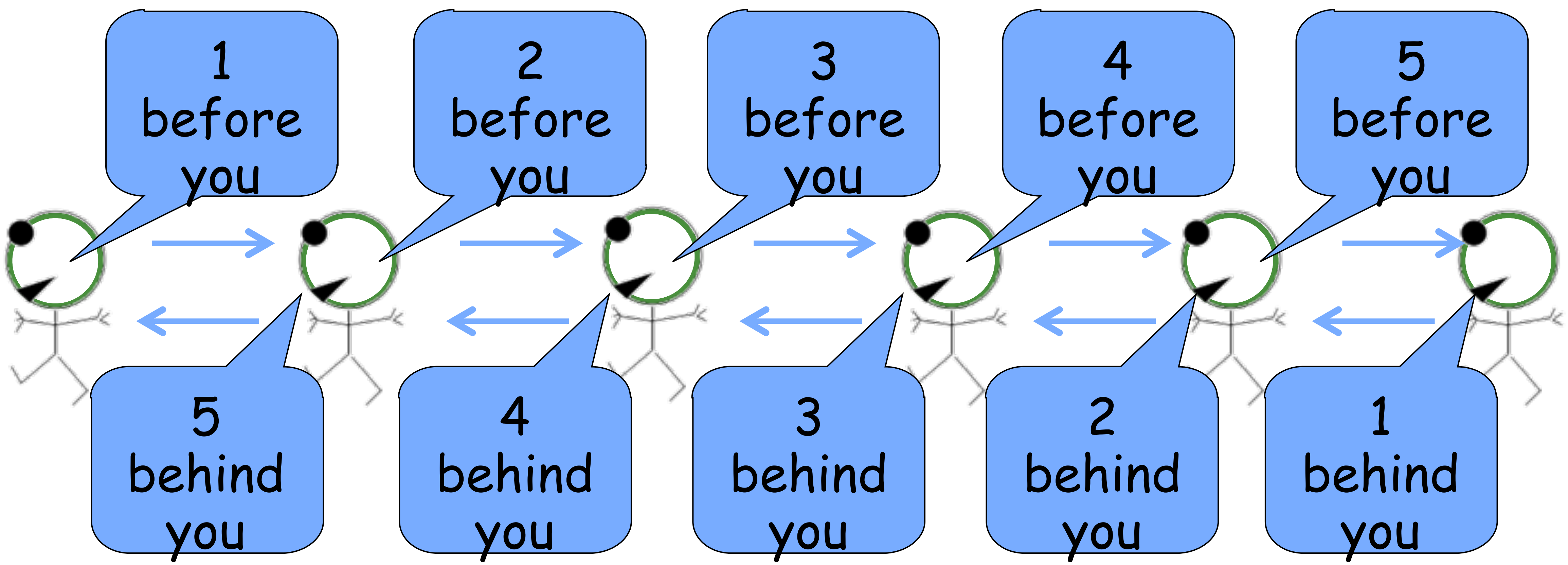
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

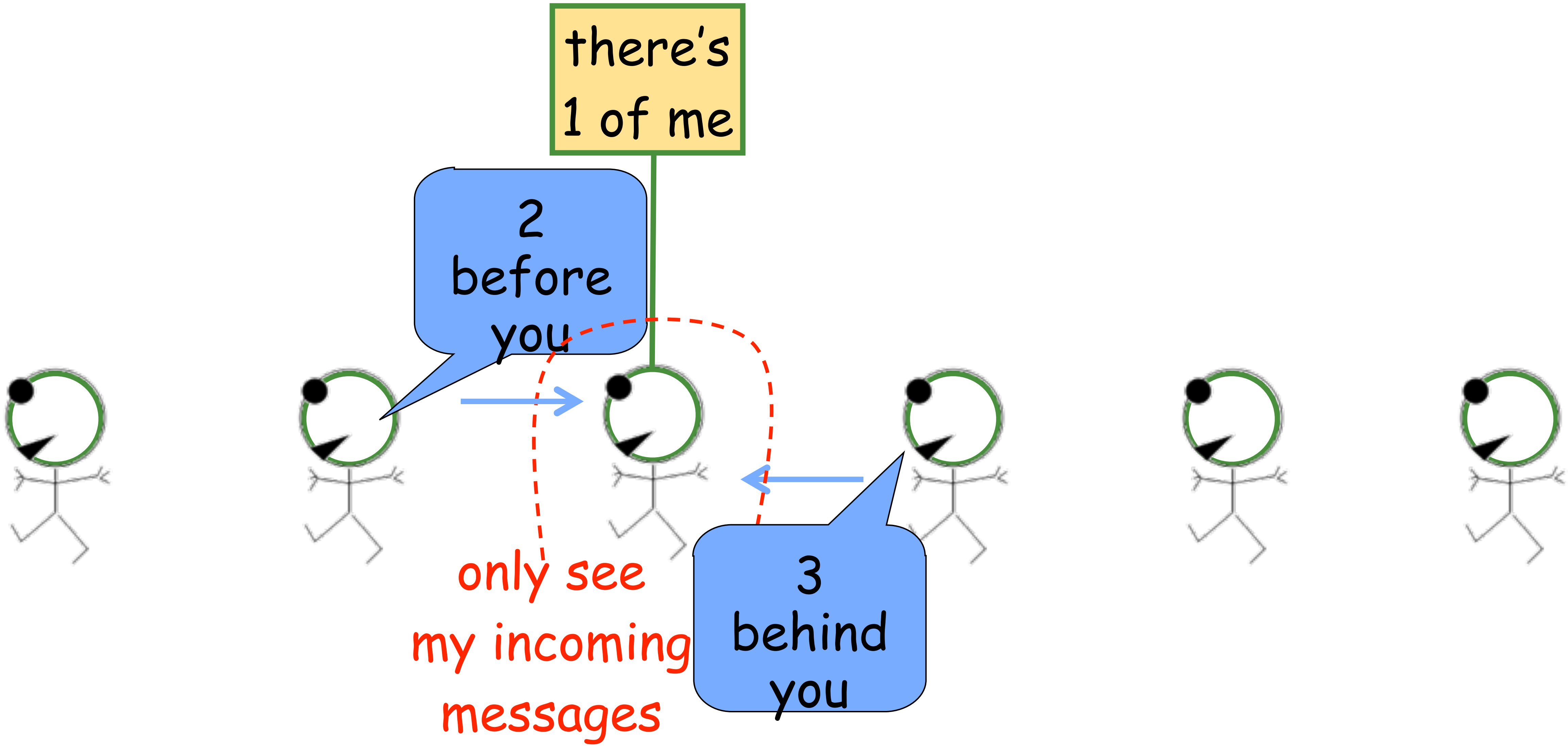
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

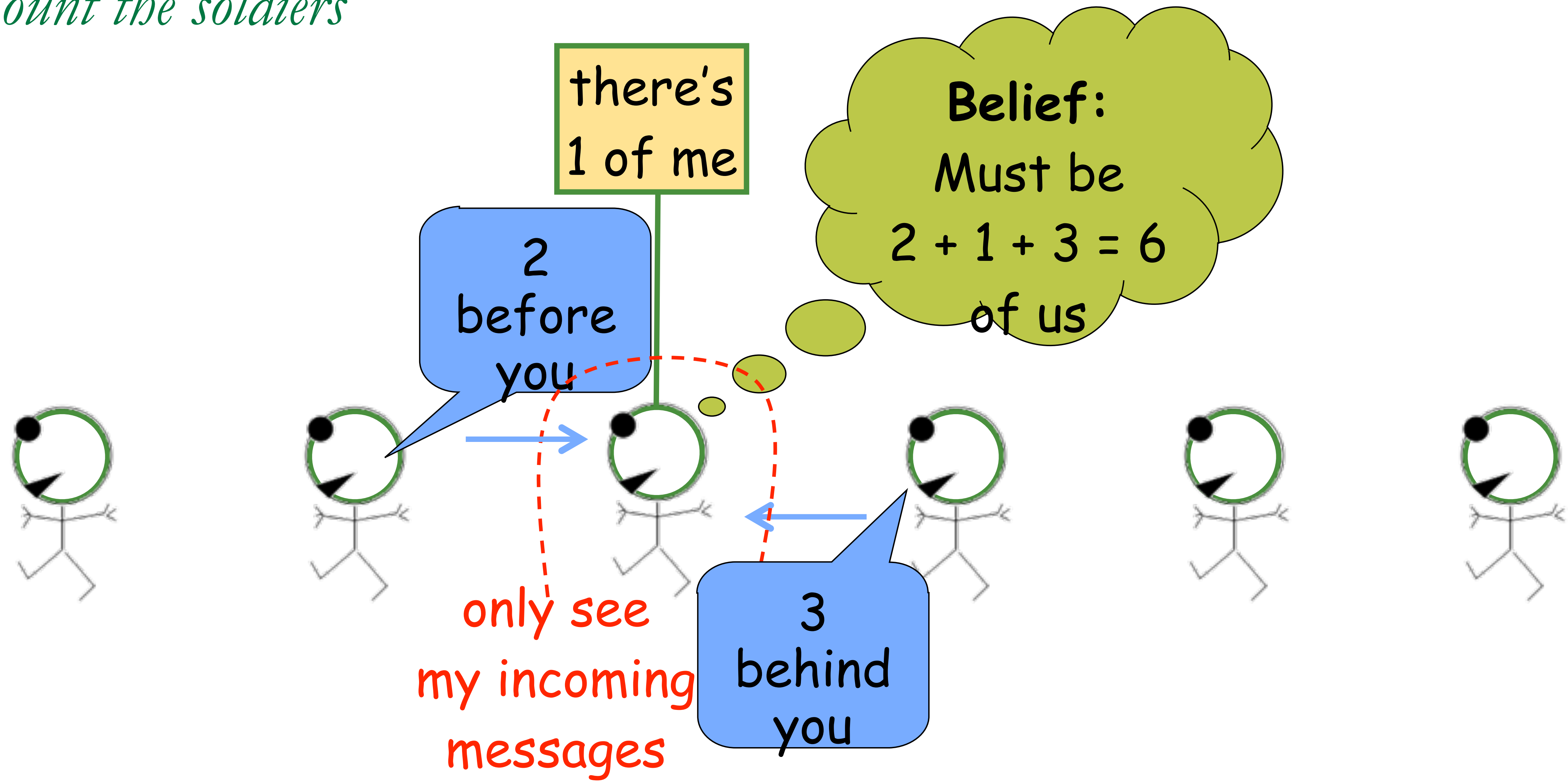
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

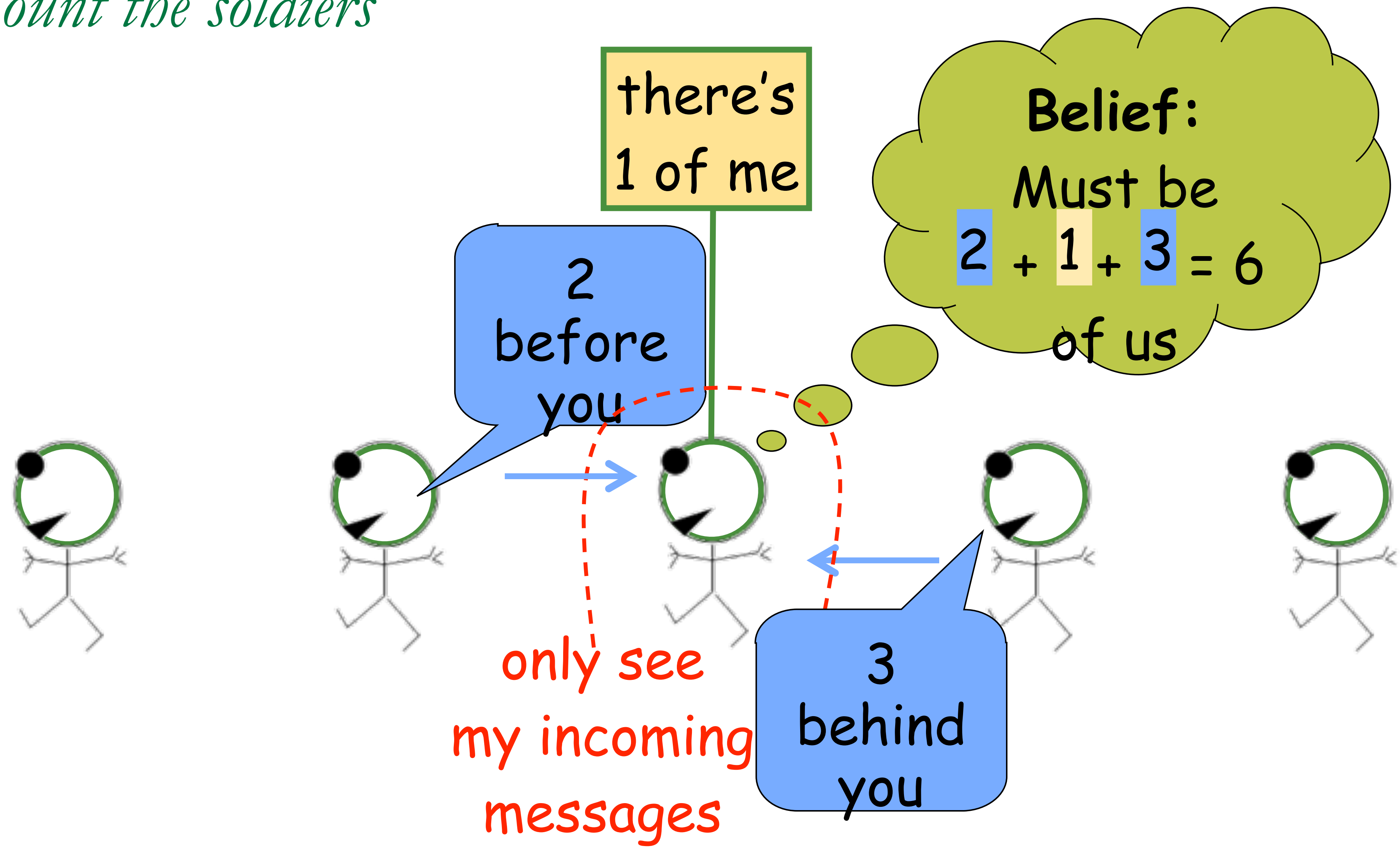
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

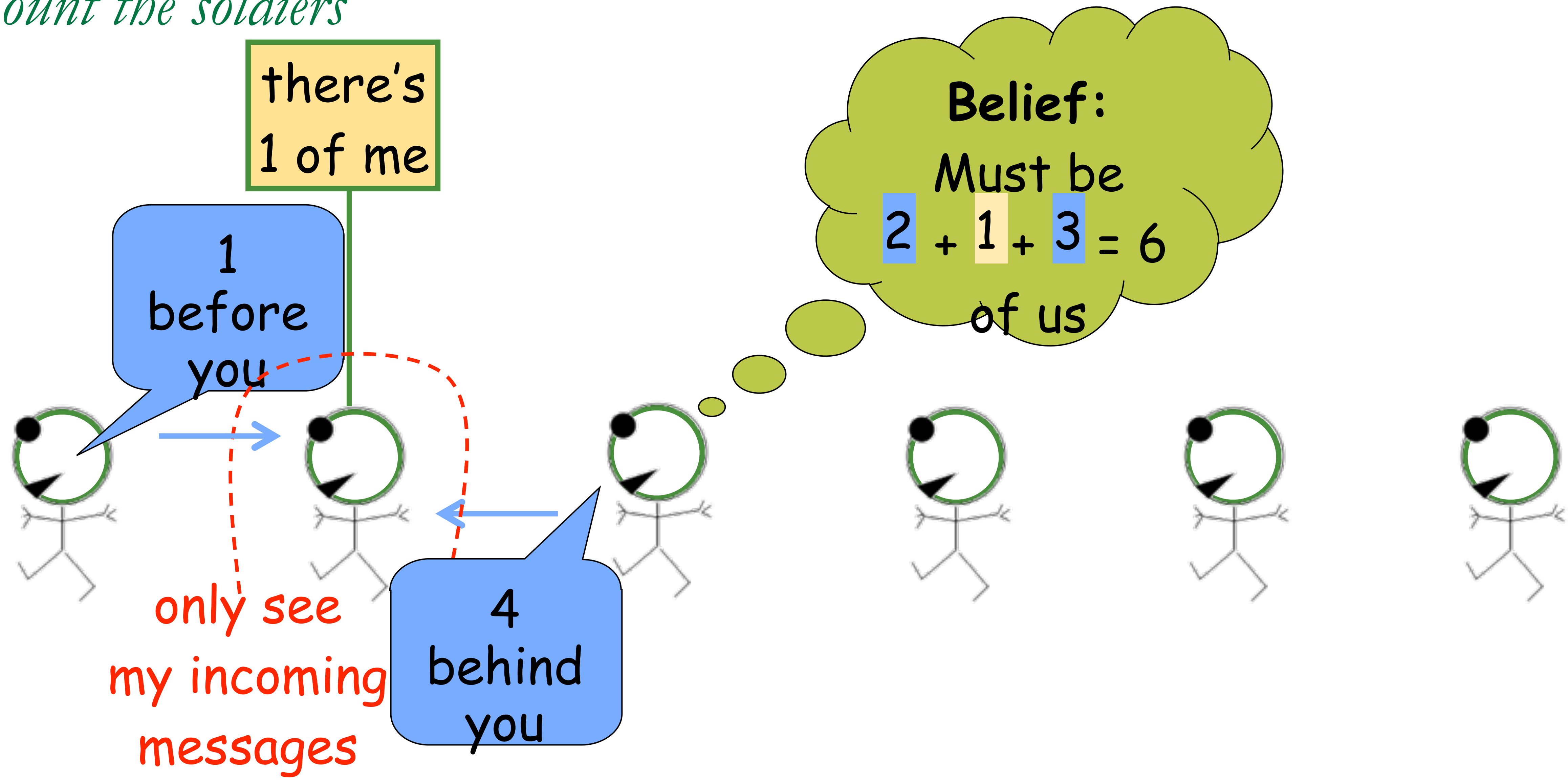
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

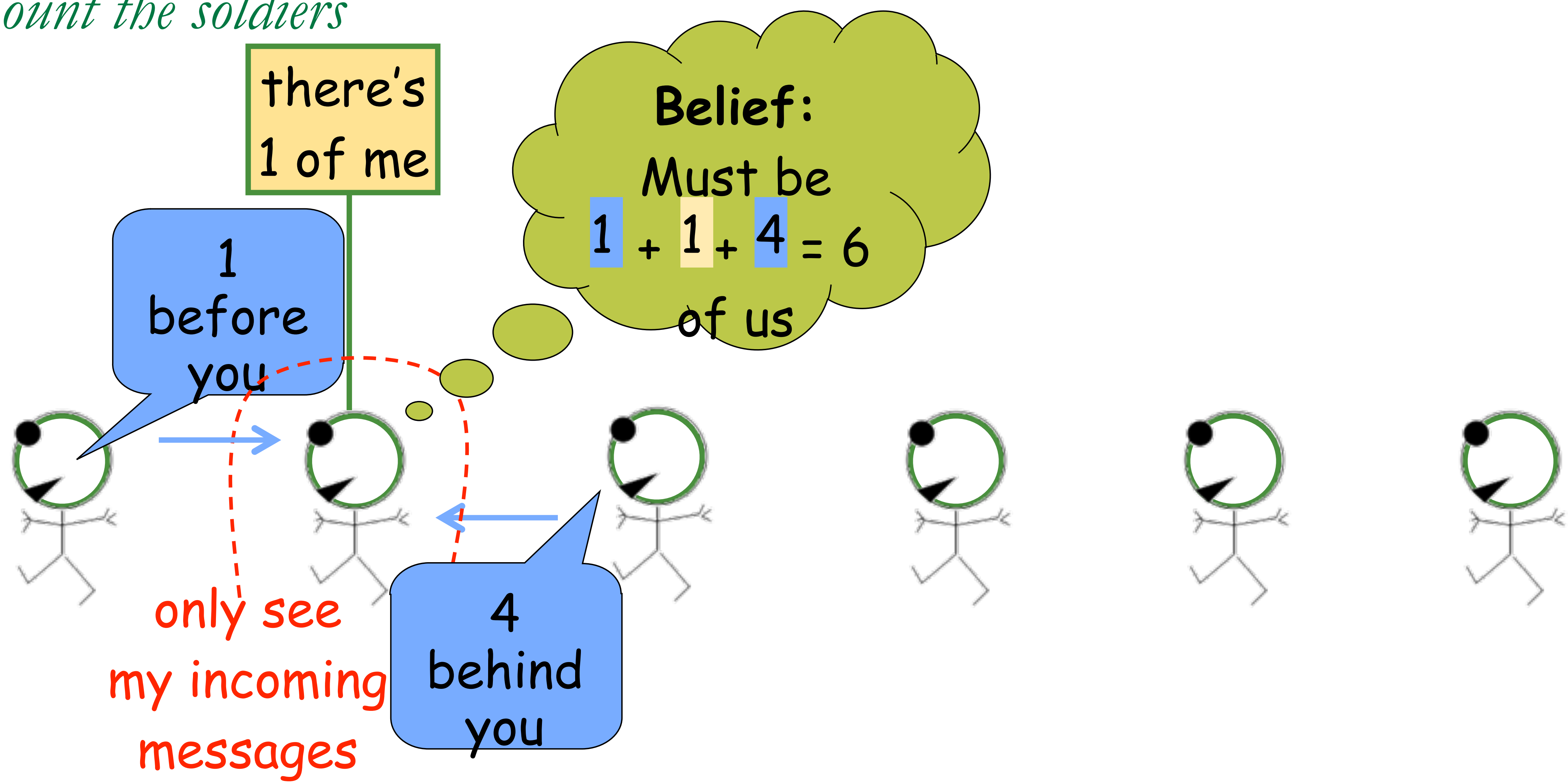
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

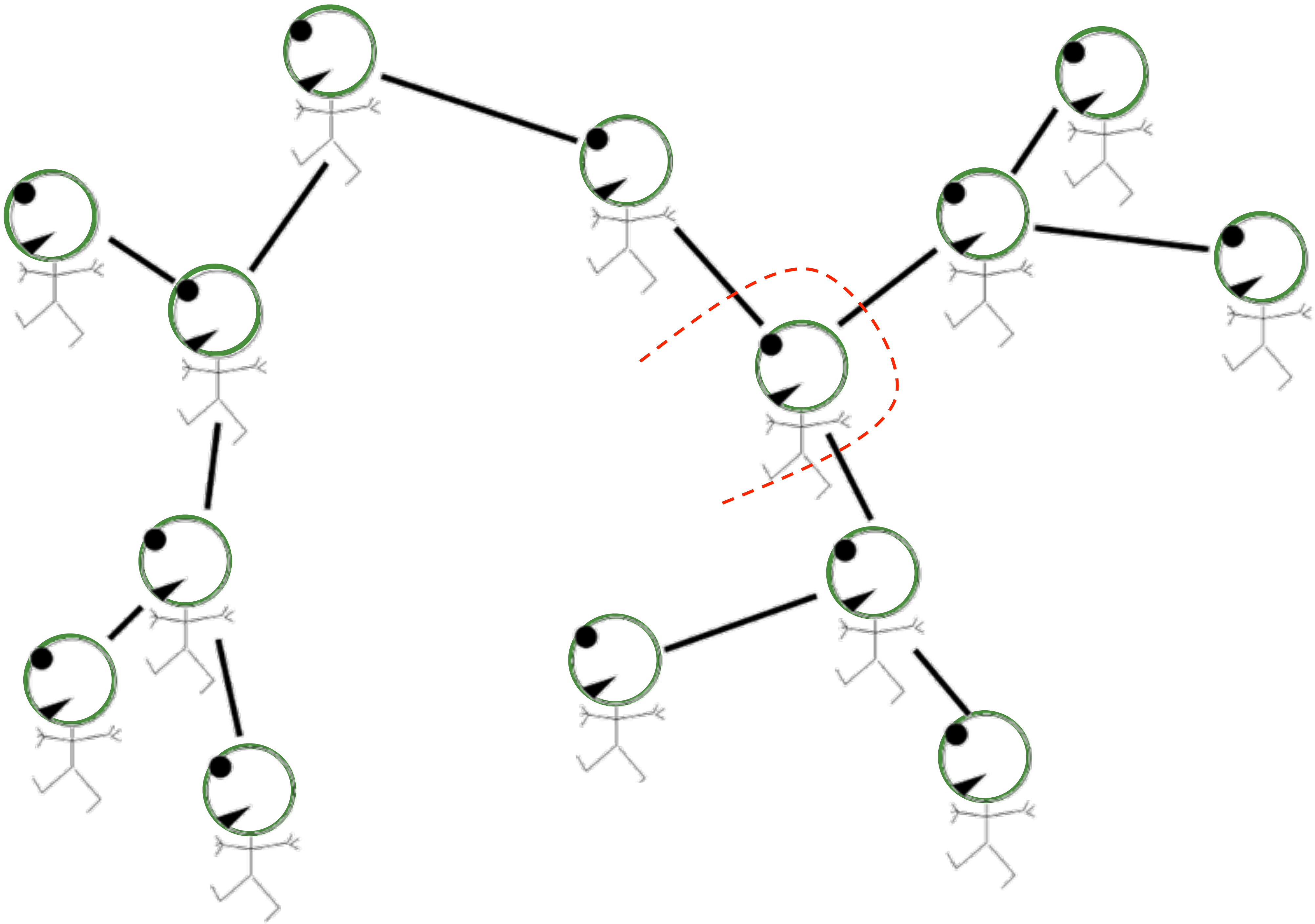
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

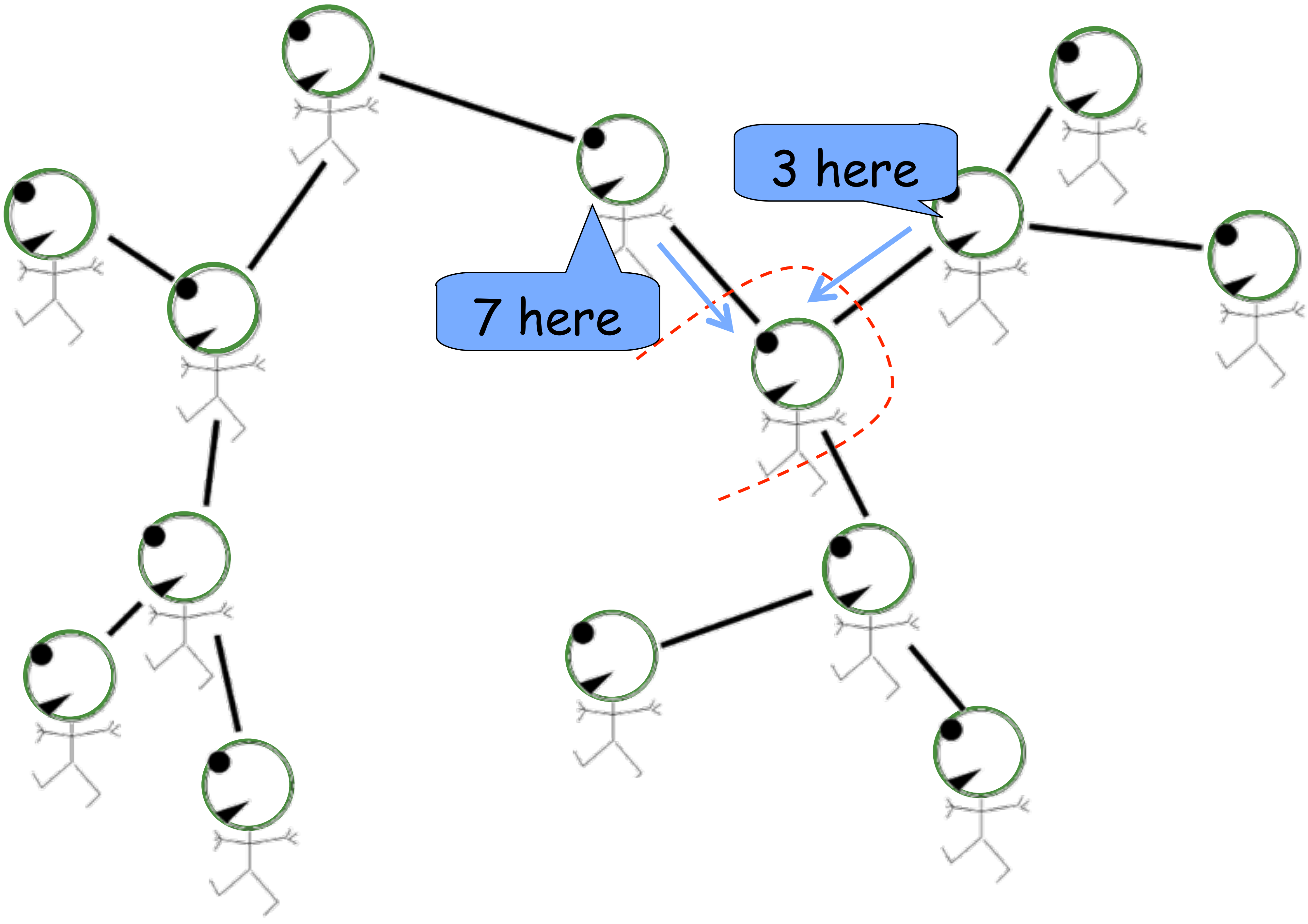
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

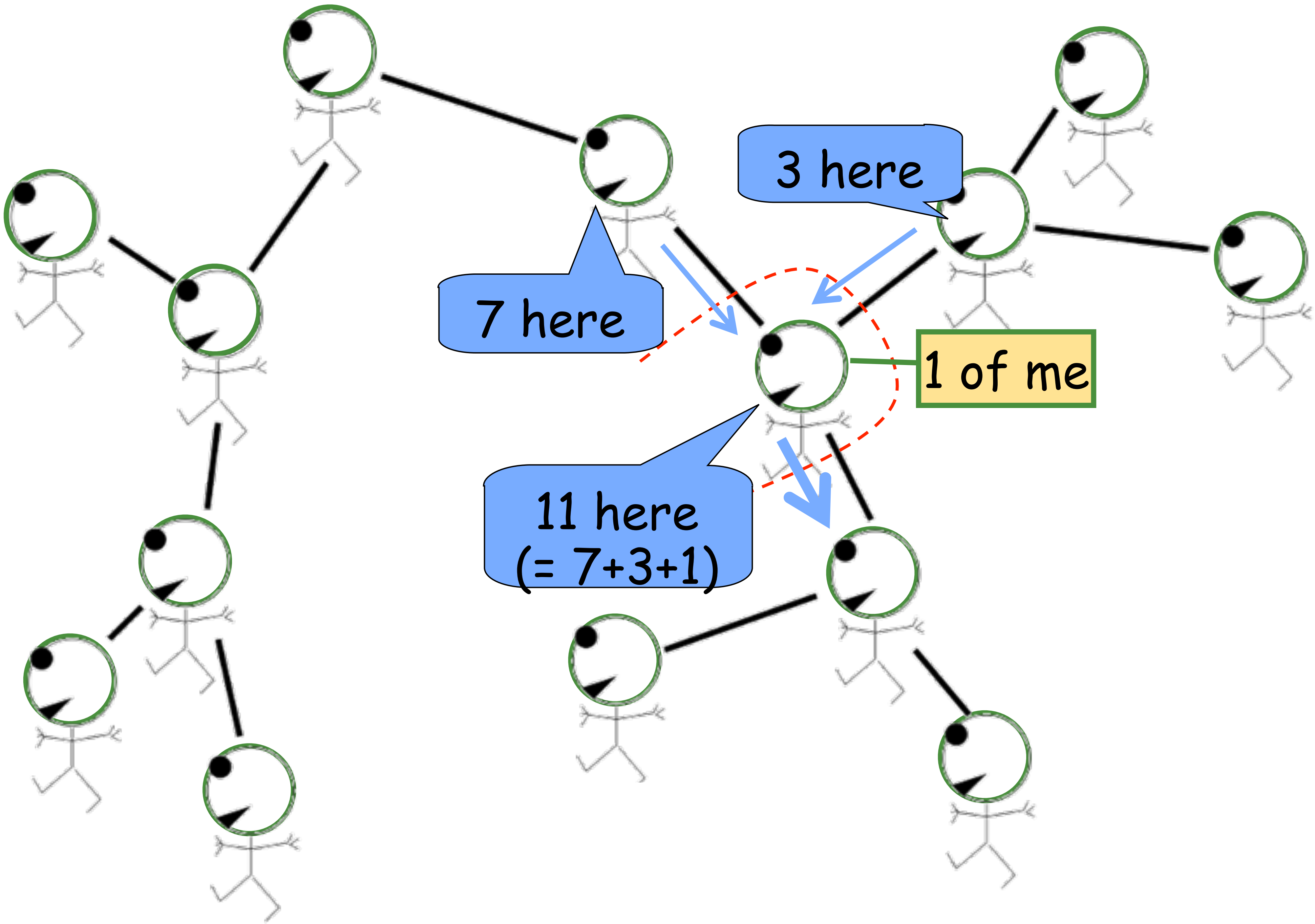
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

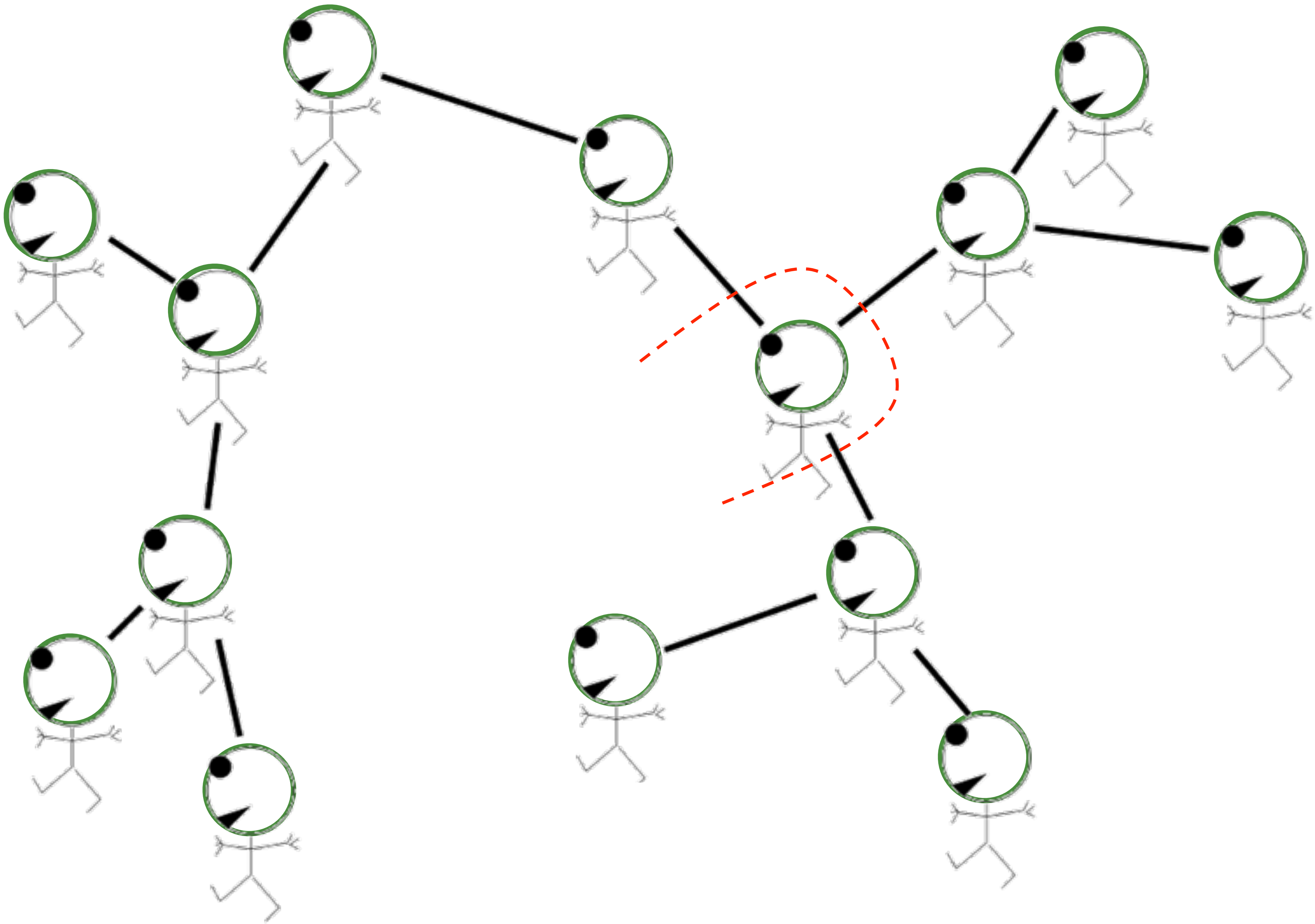
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

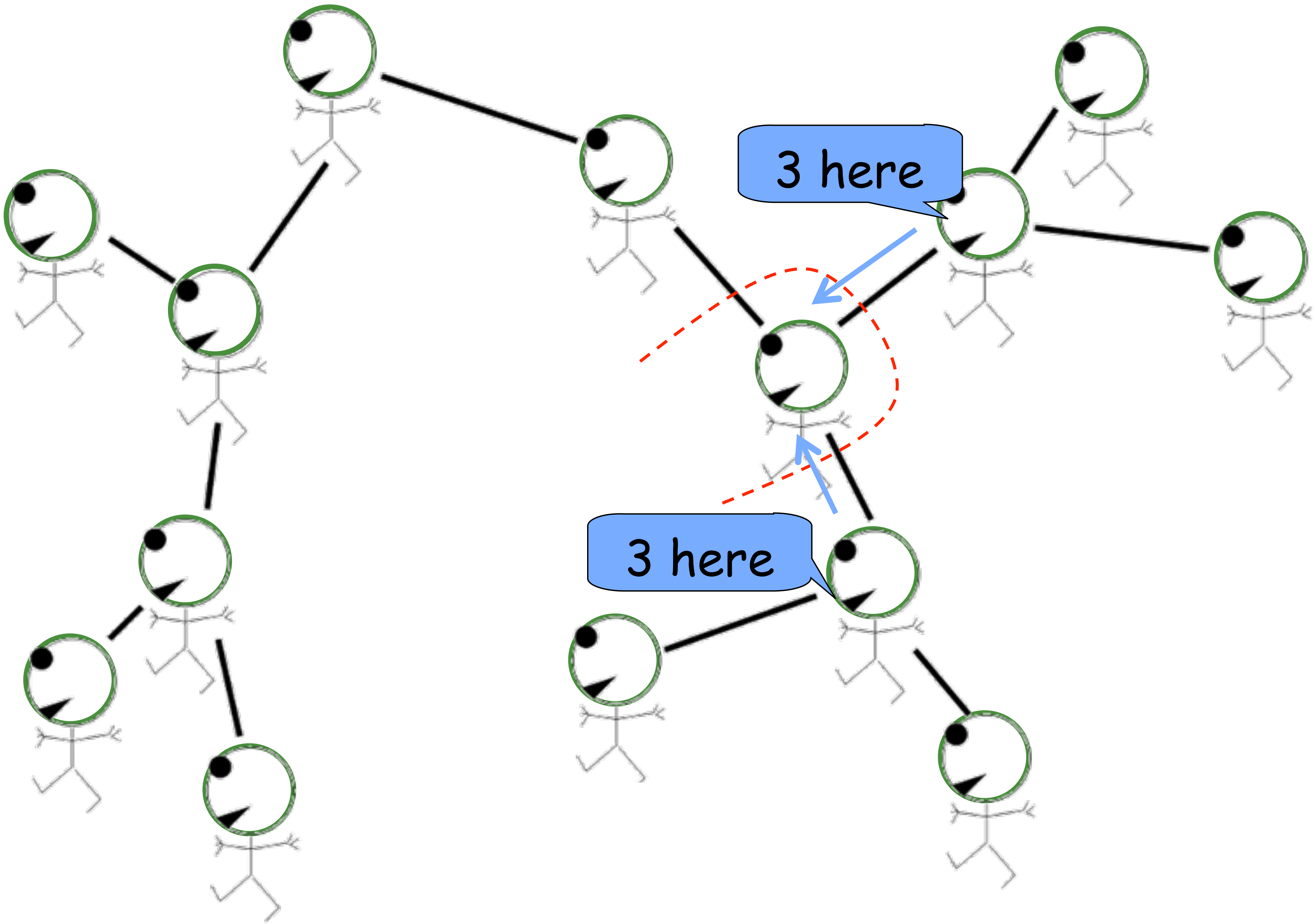
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

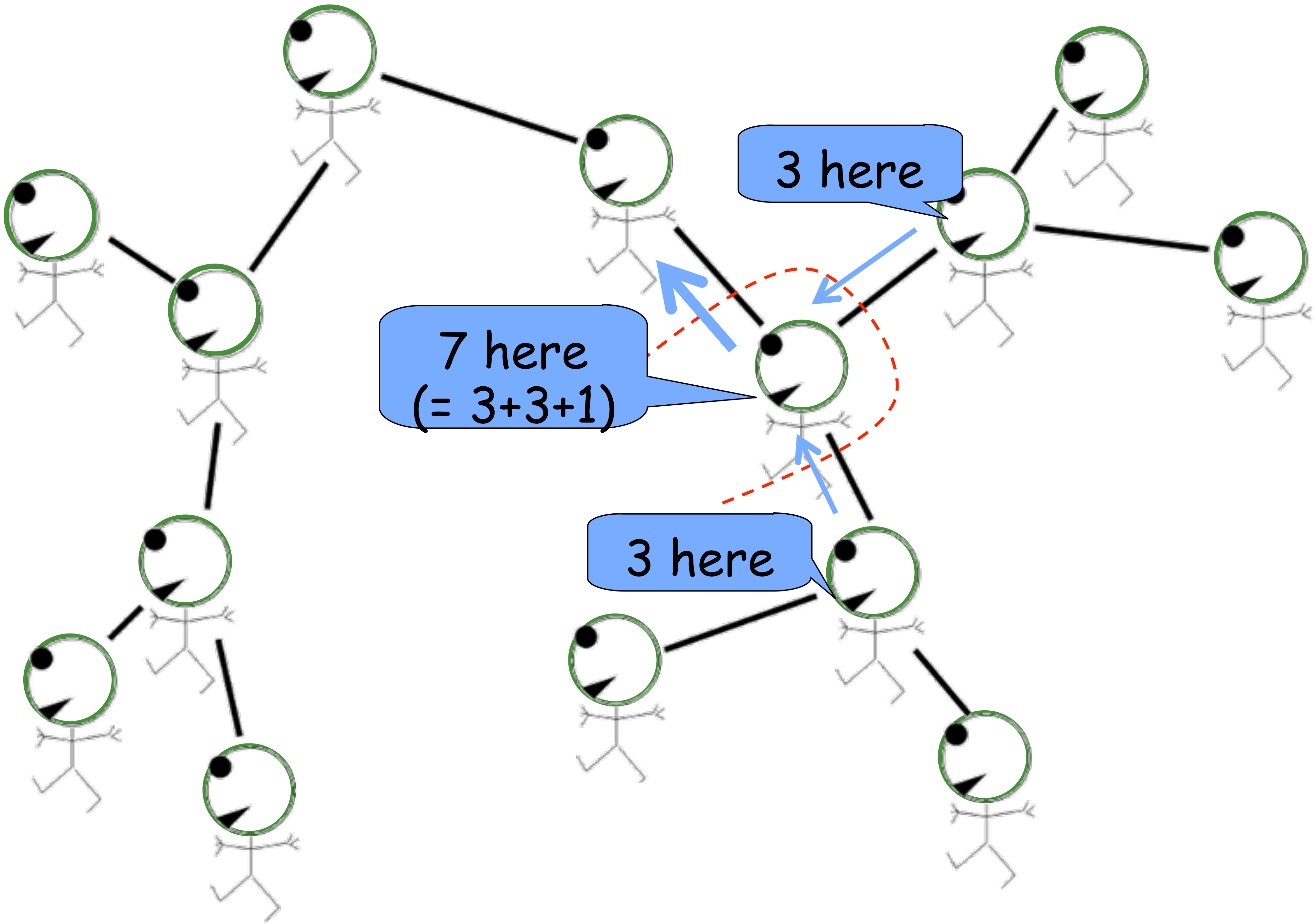
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

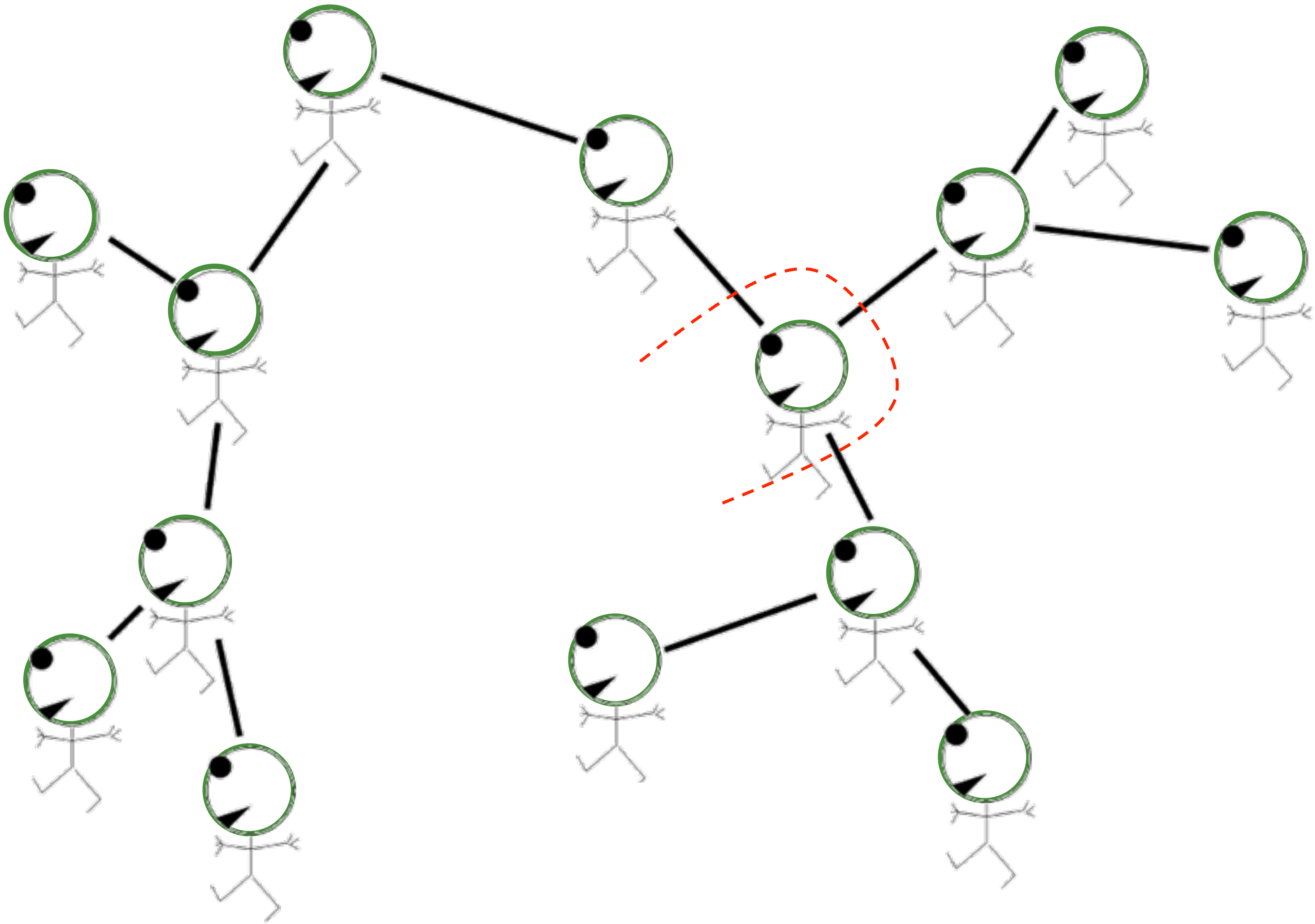
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

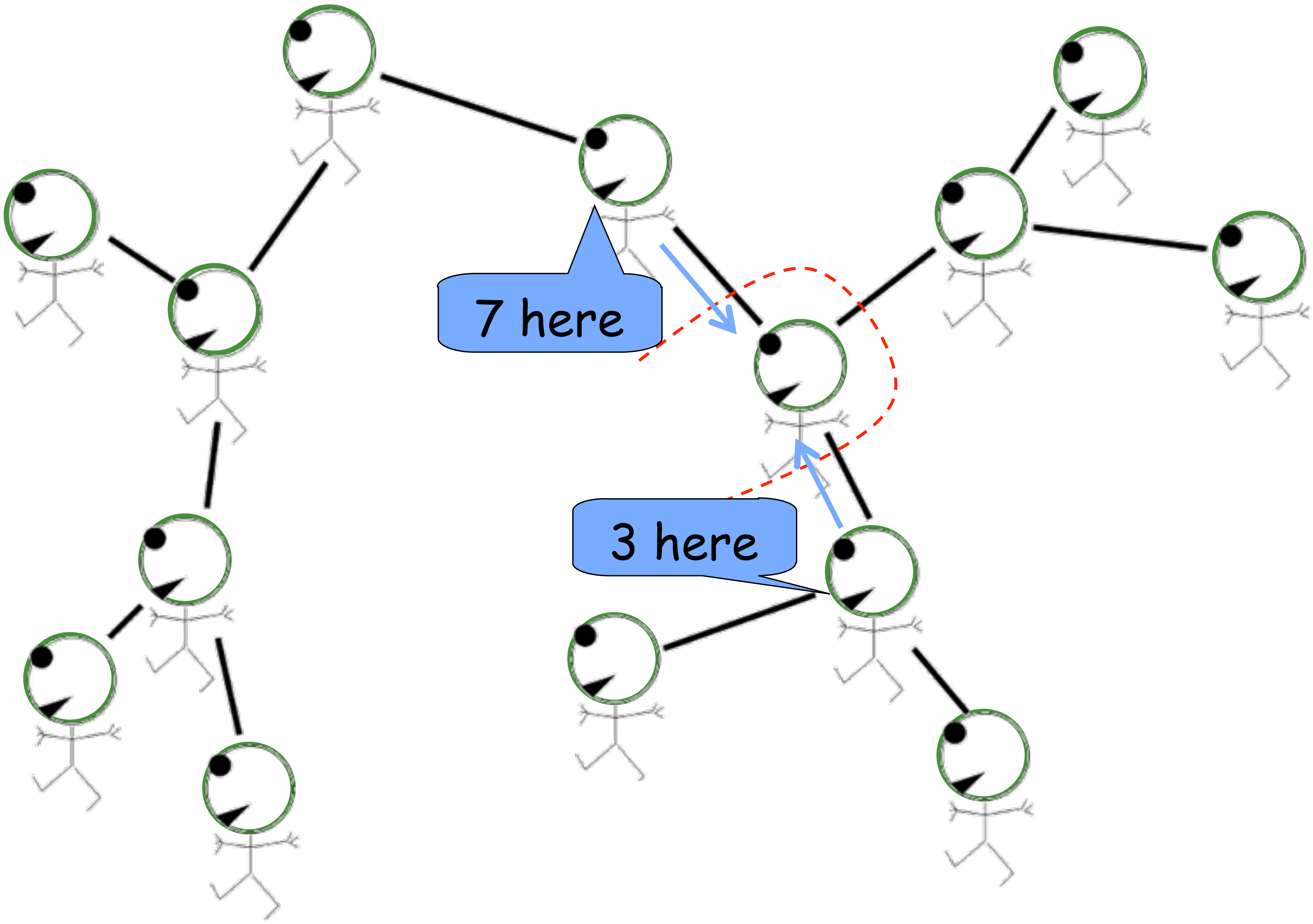
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

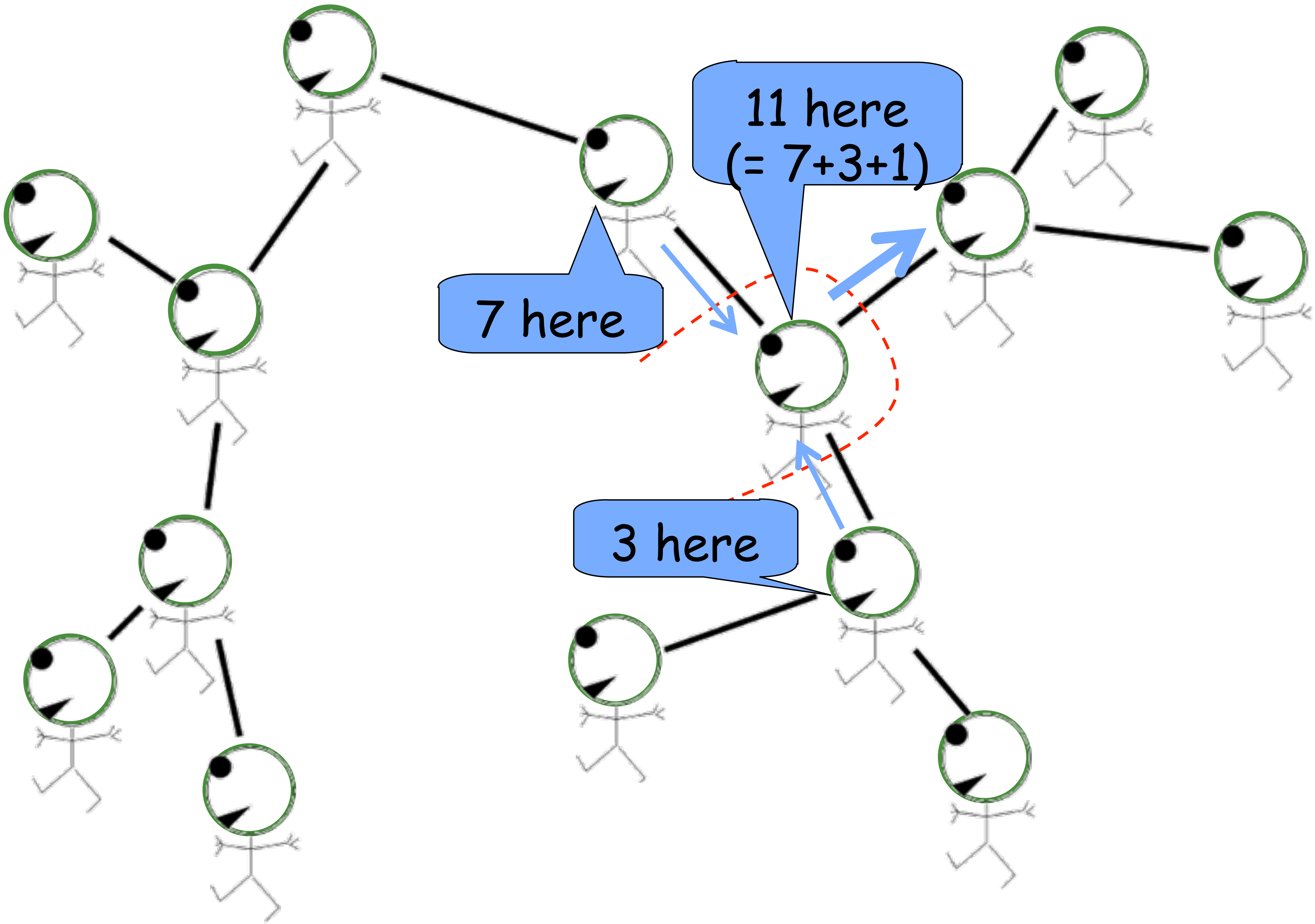
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

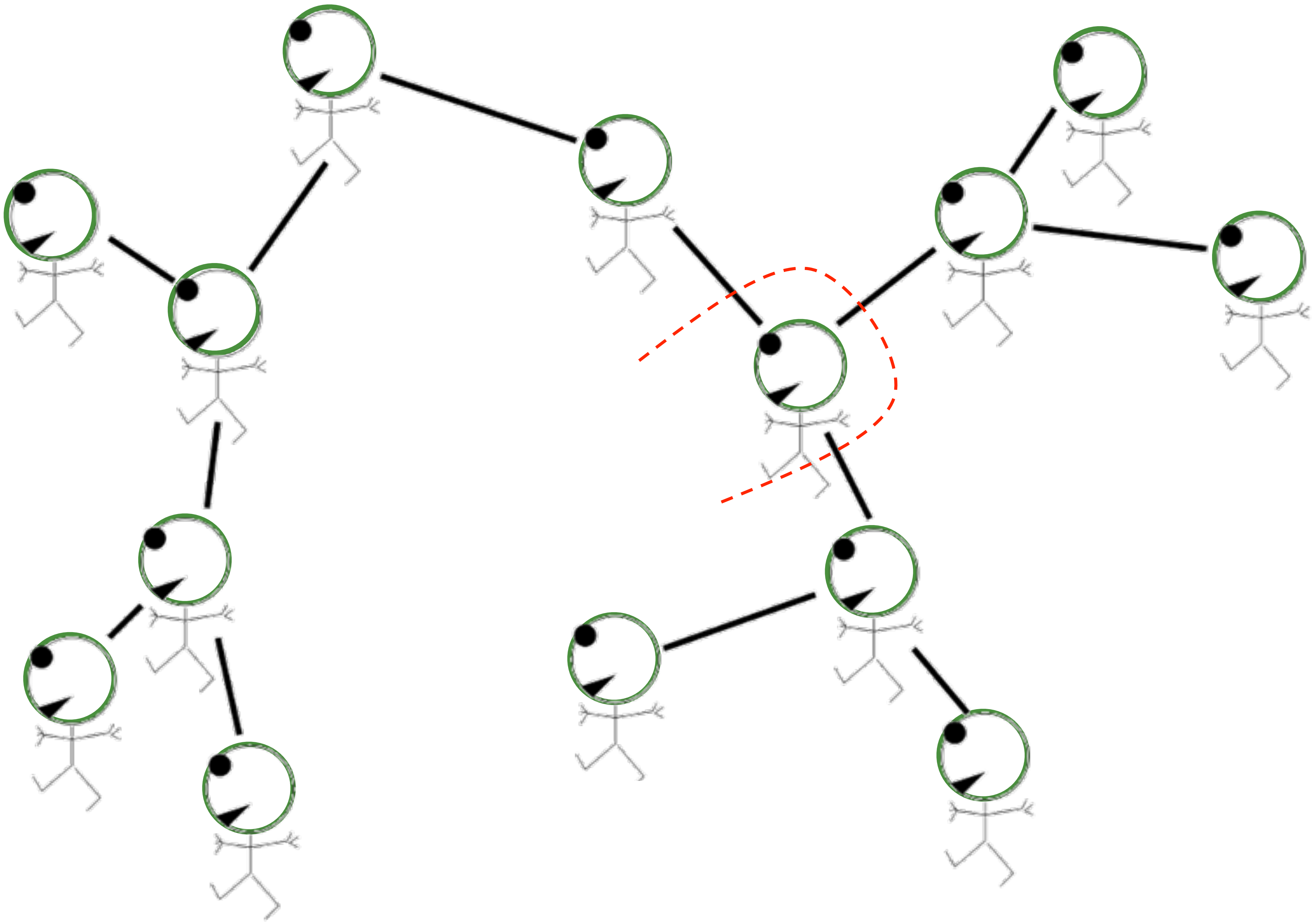
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

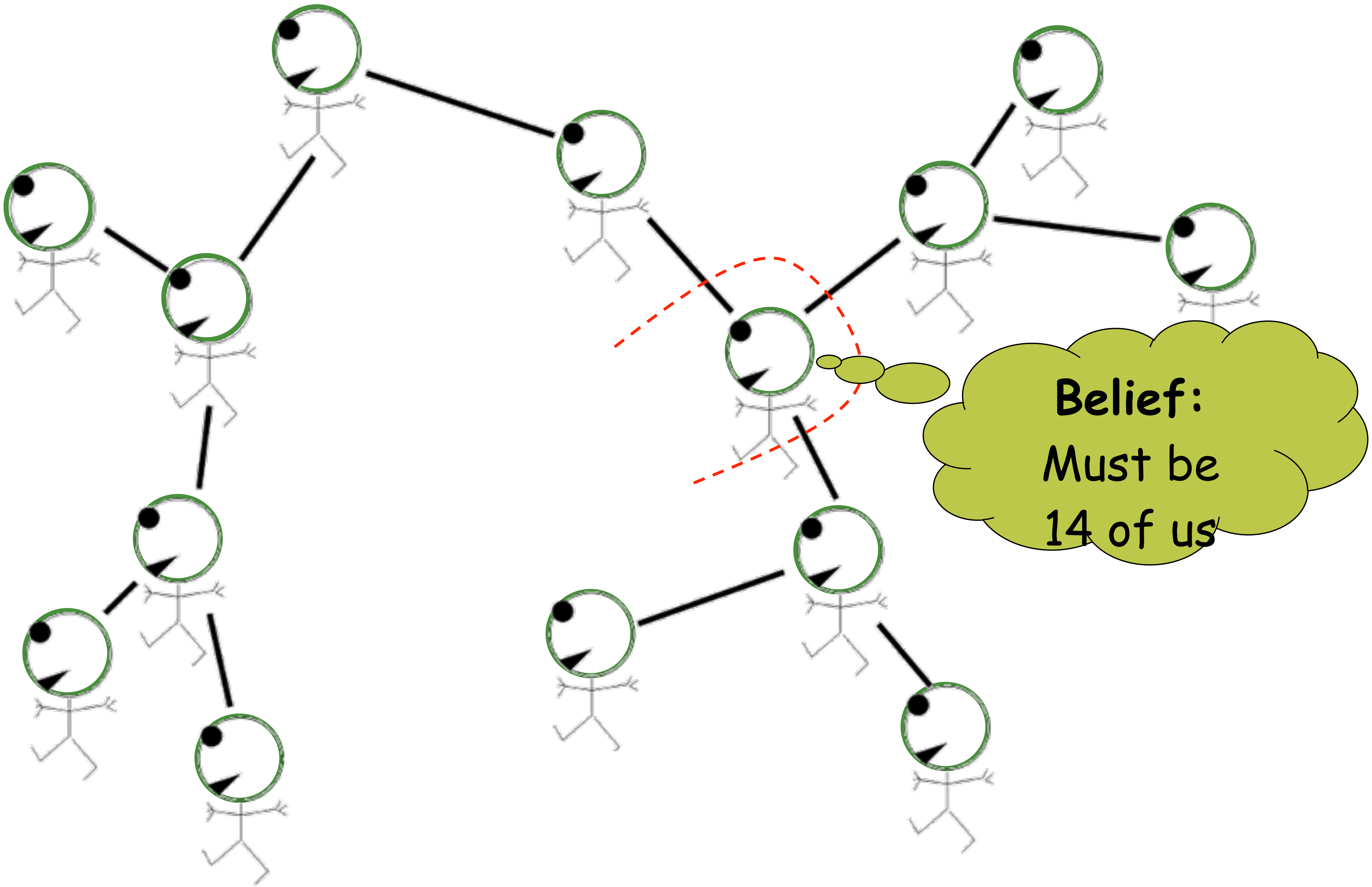
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

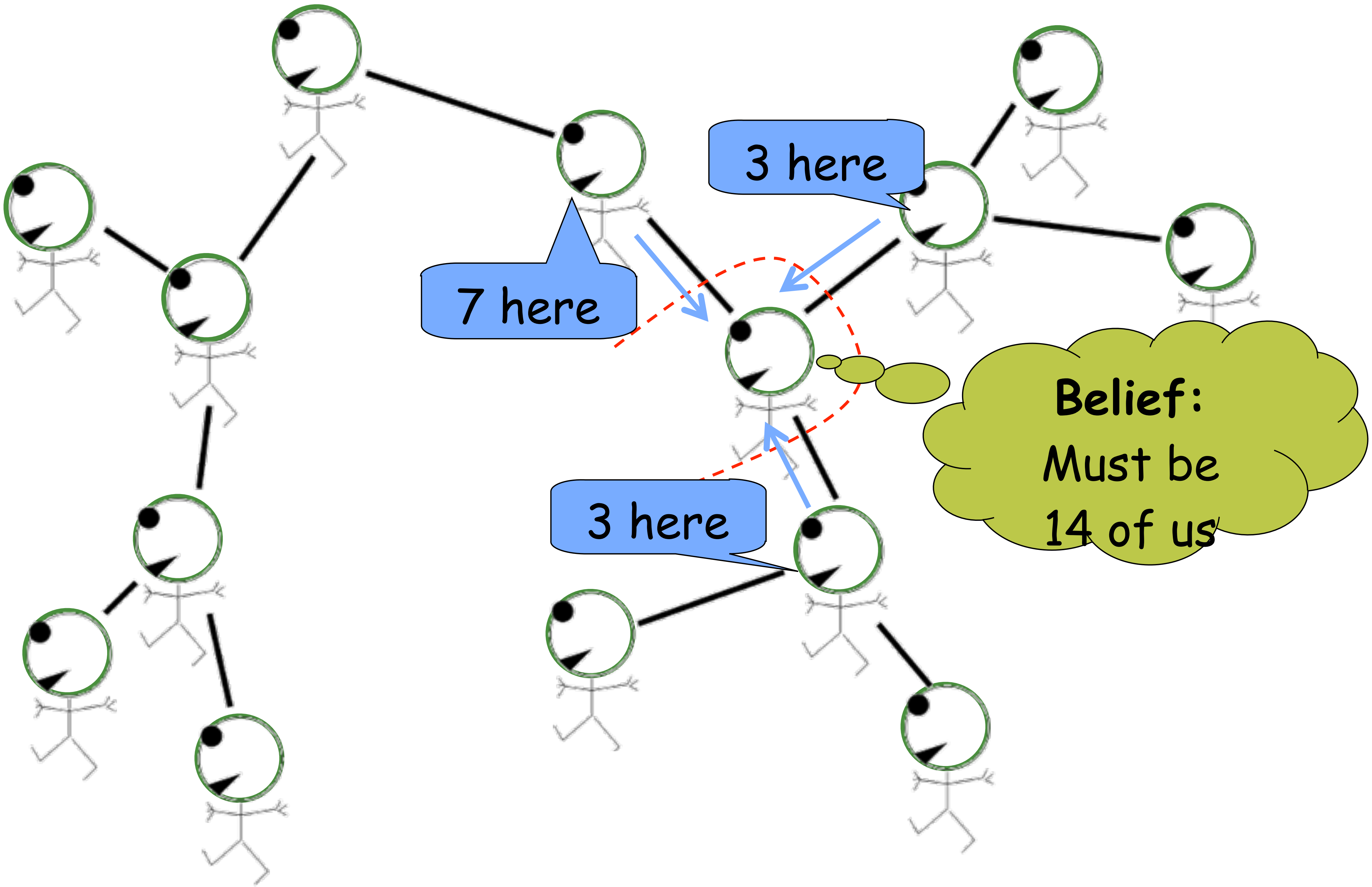
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

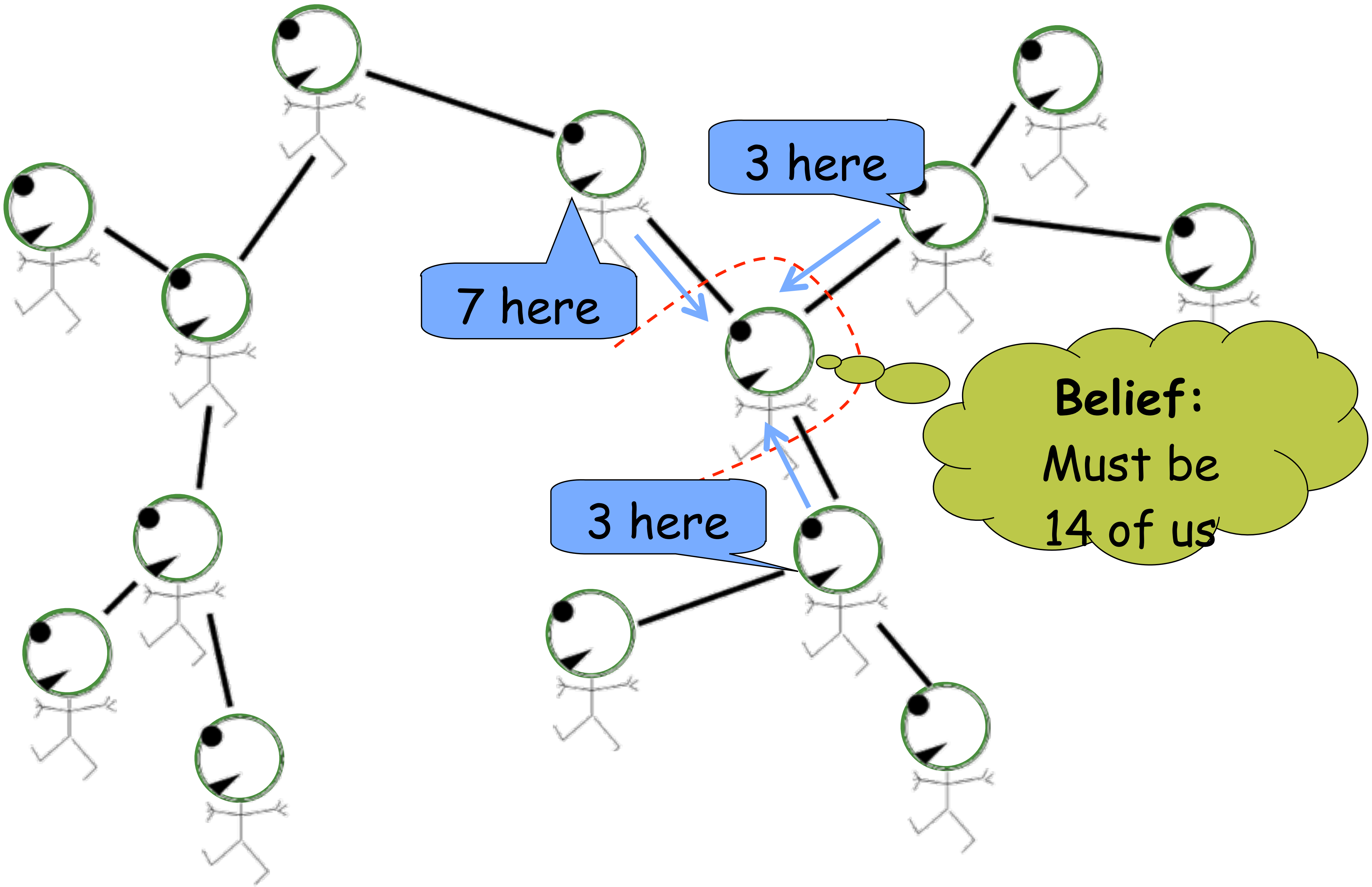
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

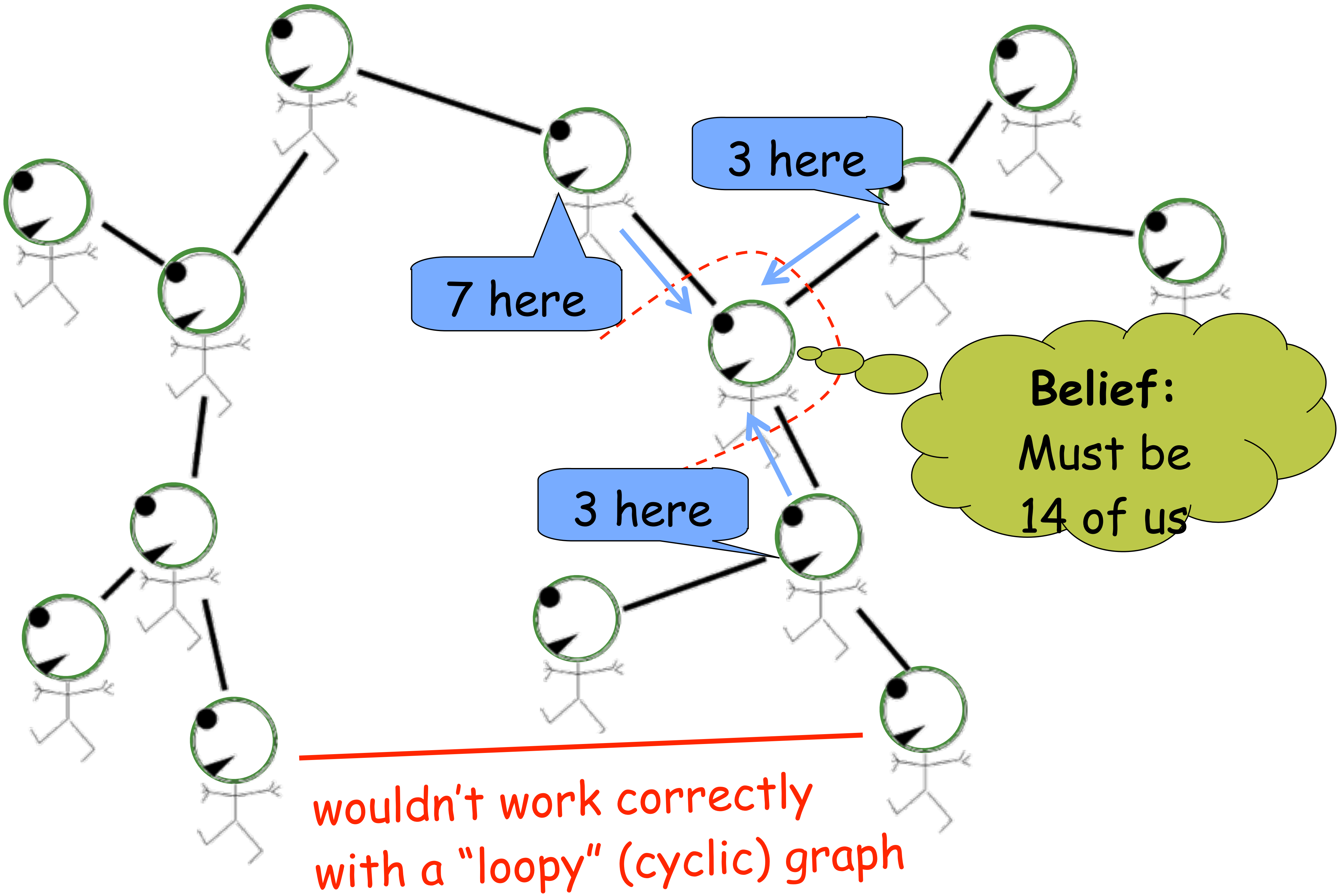
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

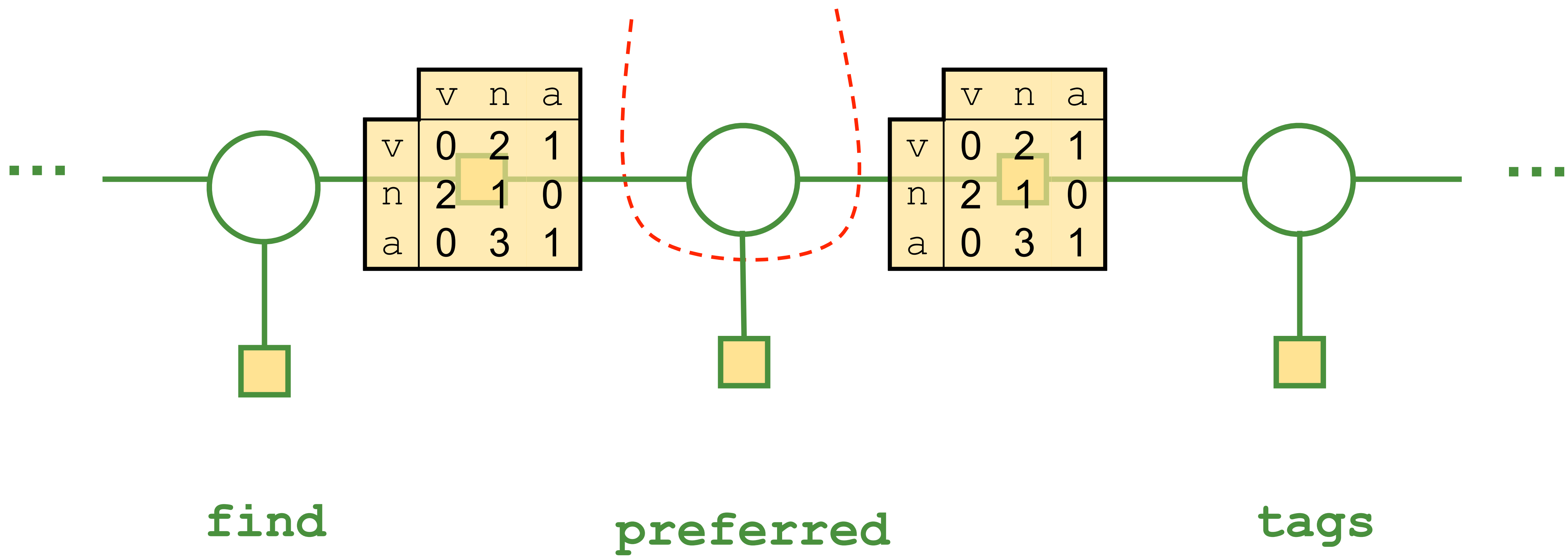
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

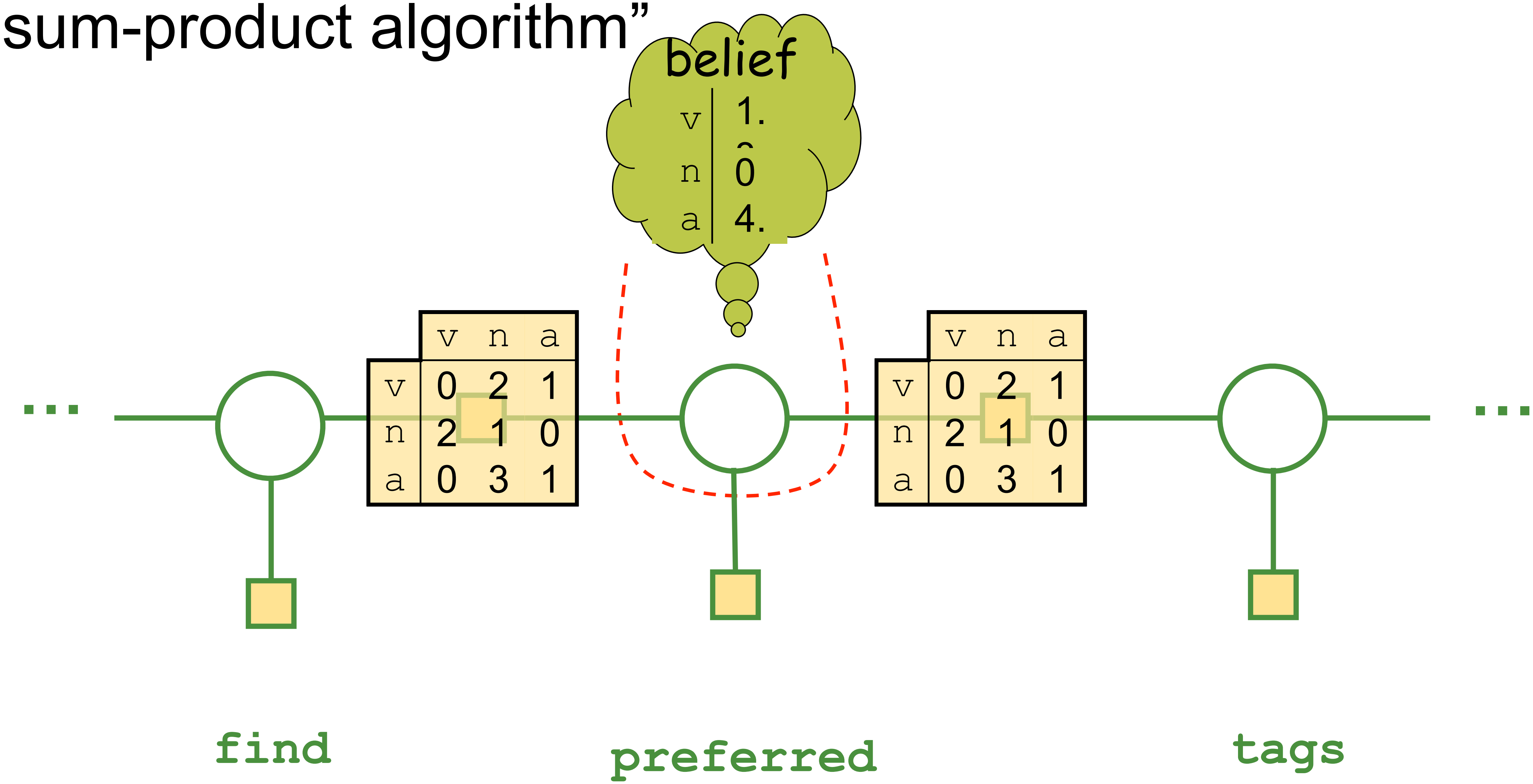
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



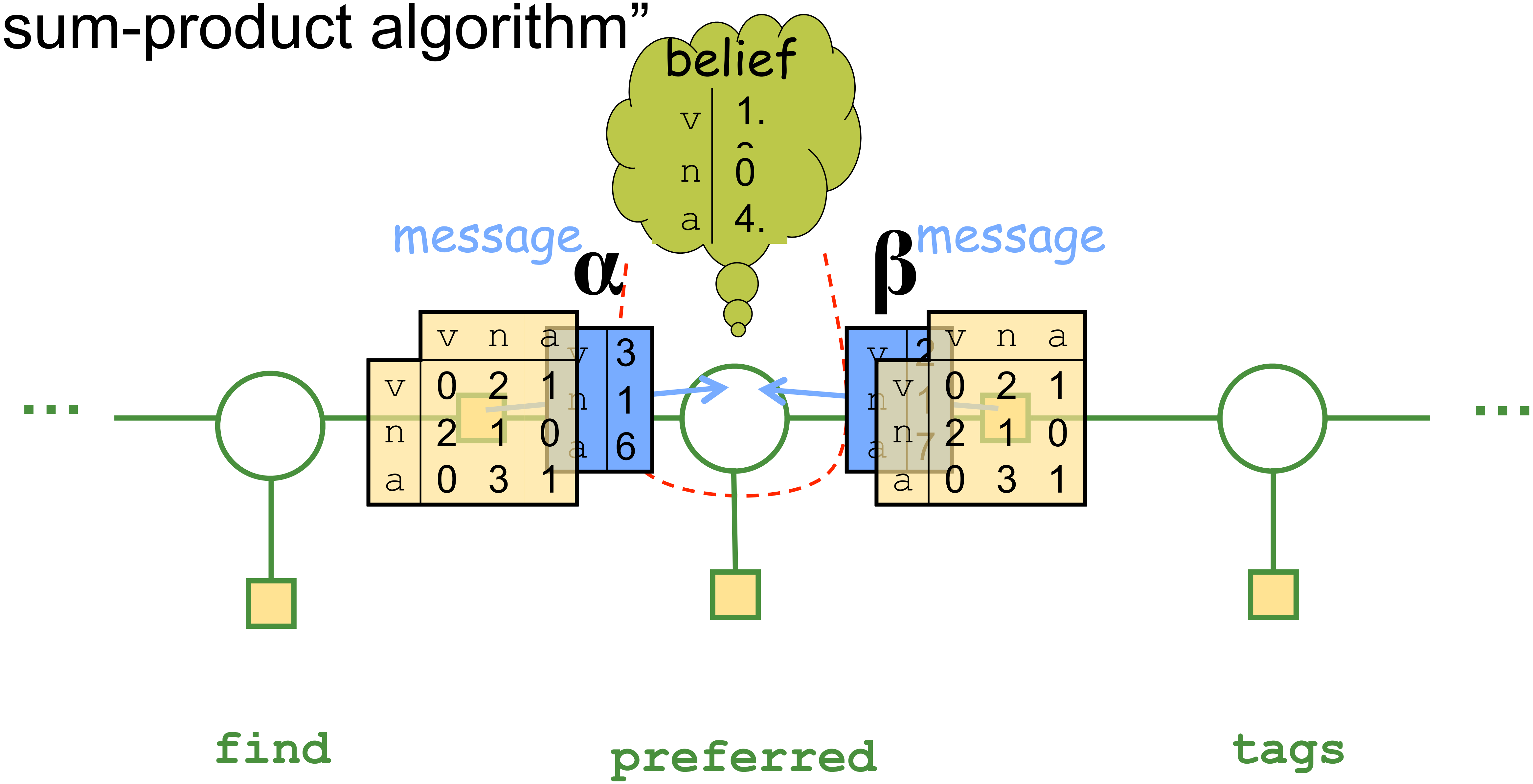
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



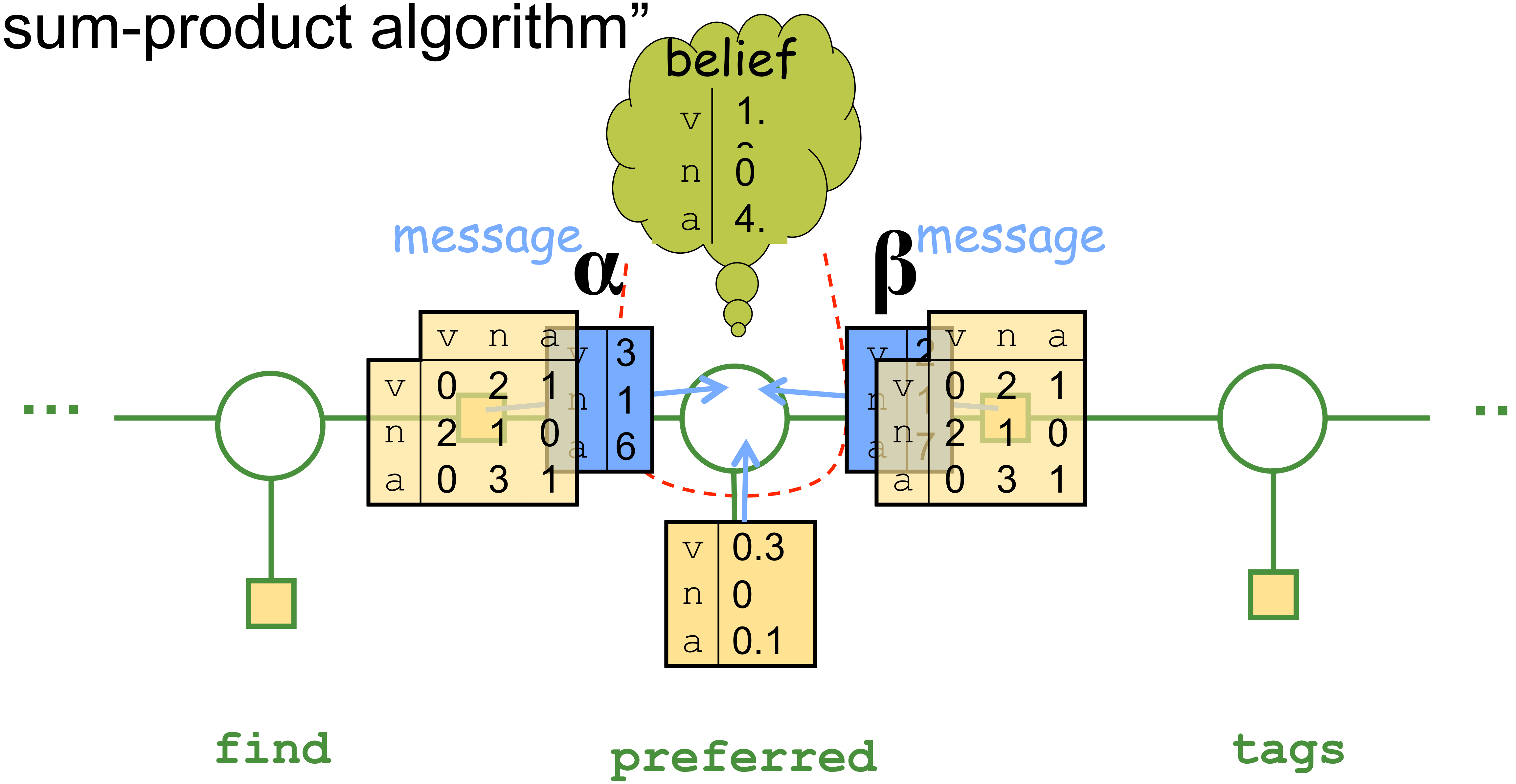
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



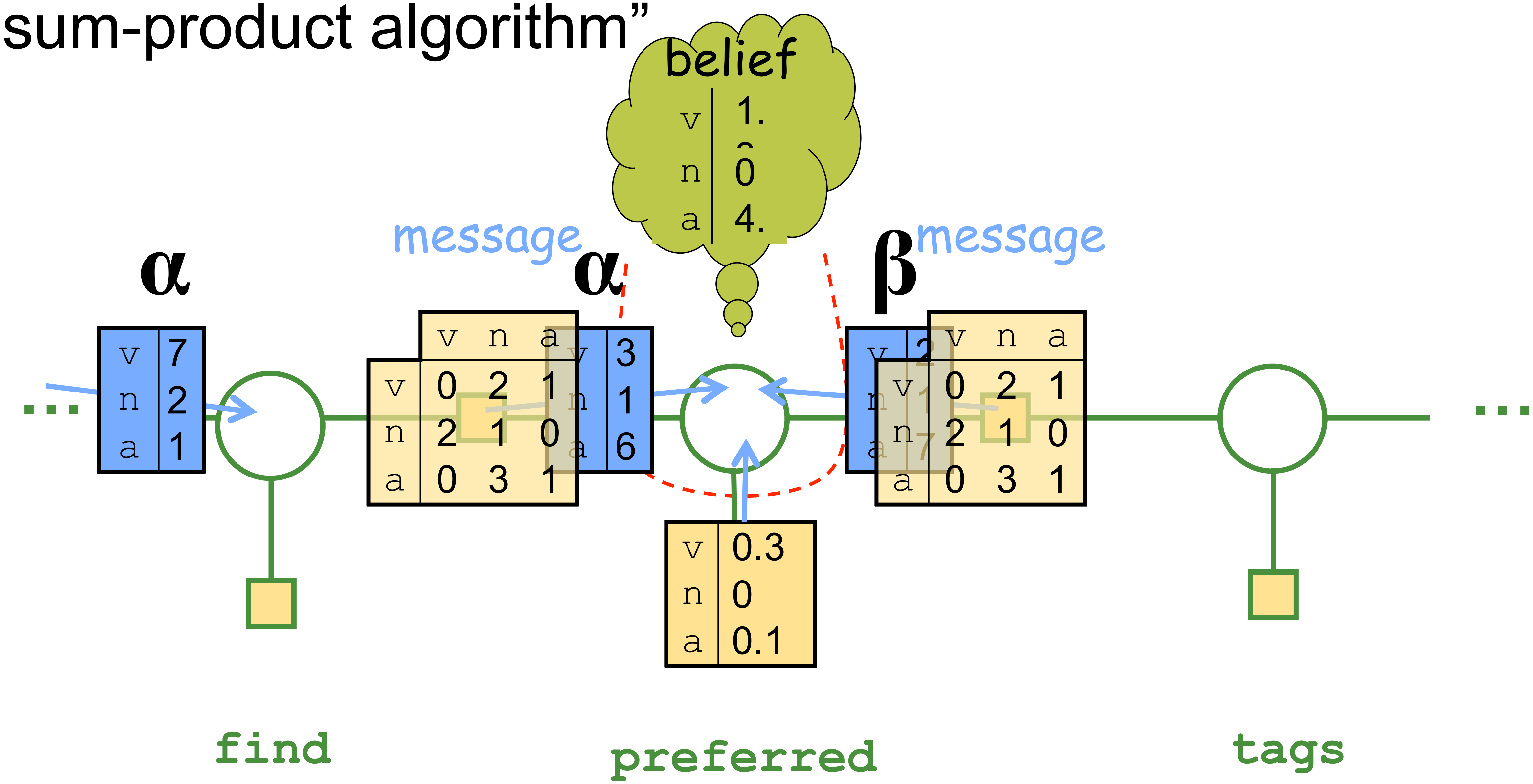
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= “sum-product algorithm”



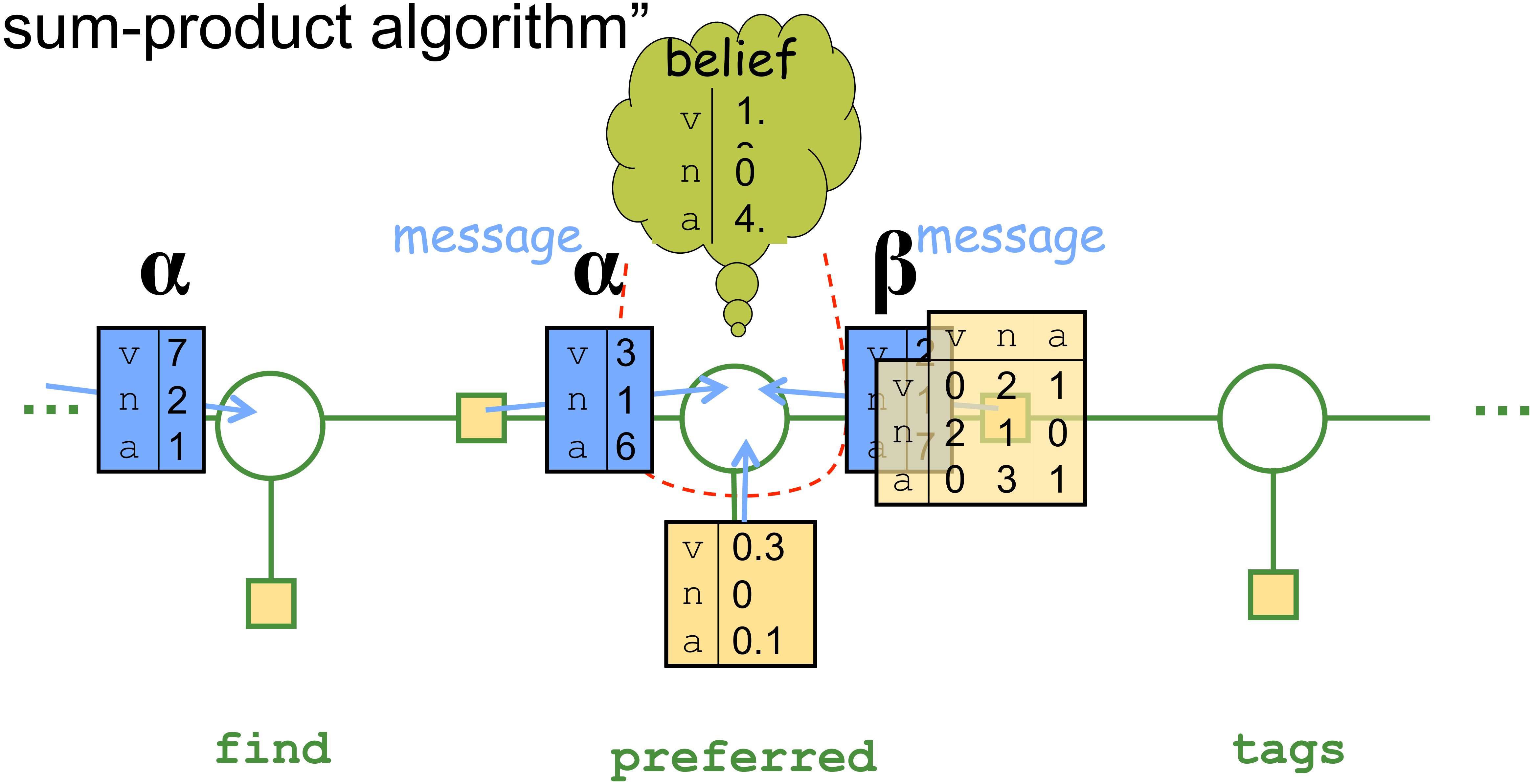
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



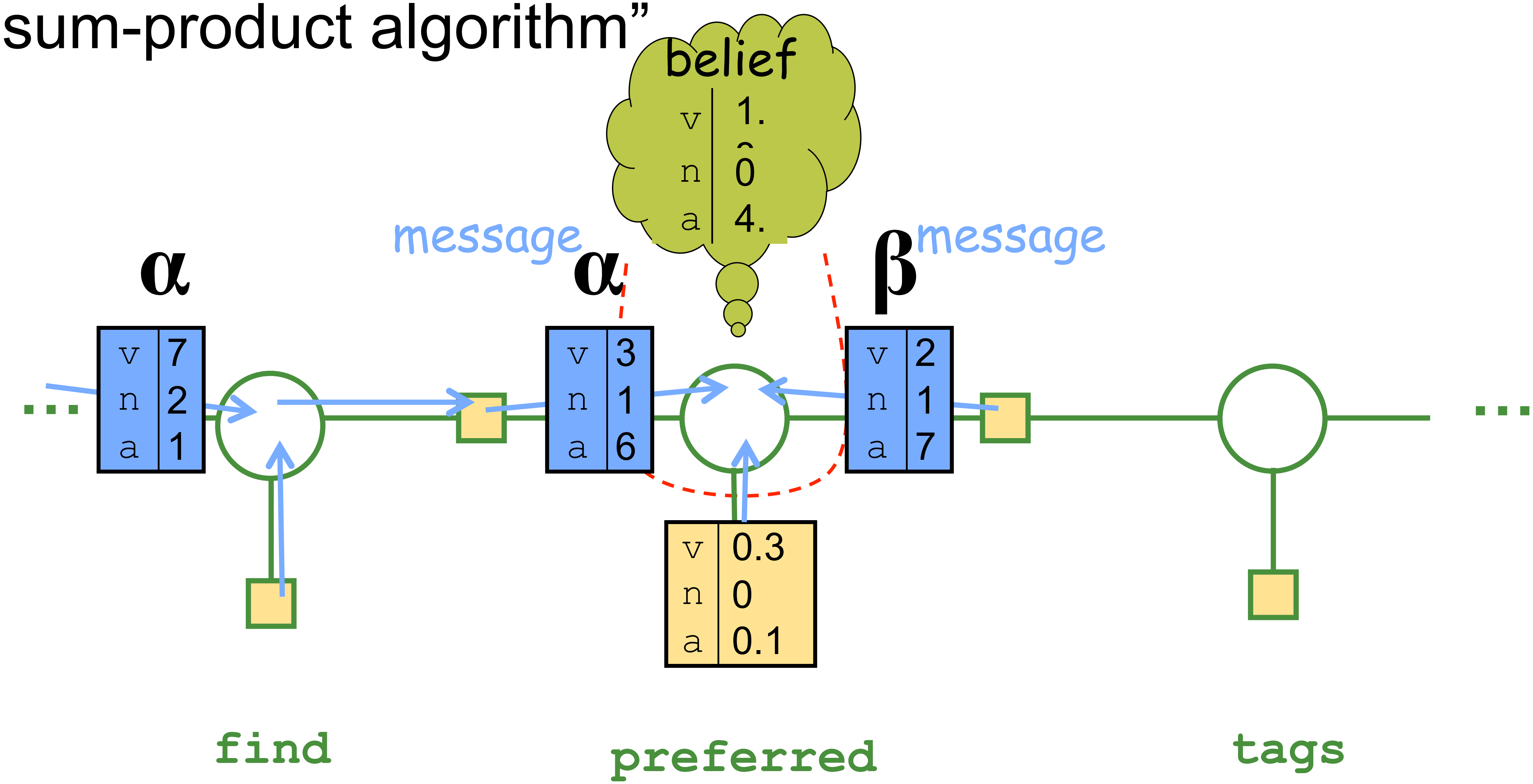
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



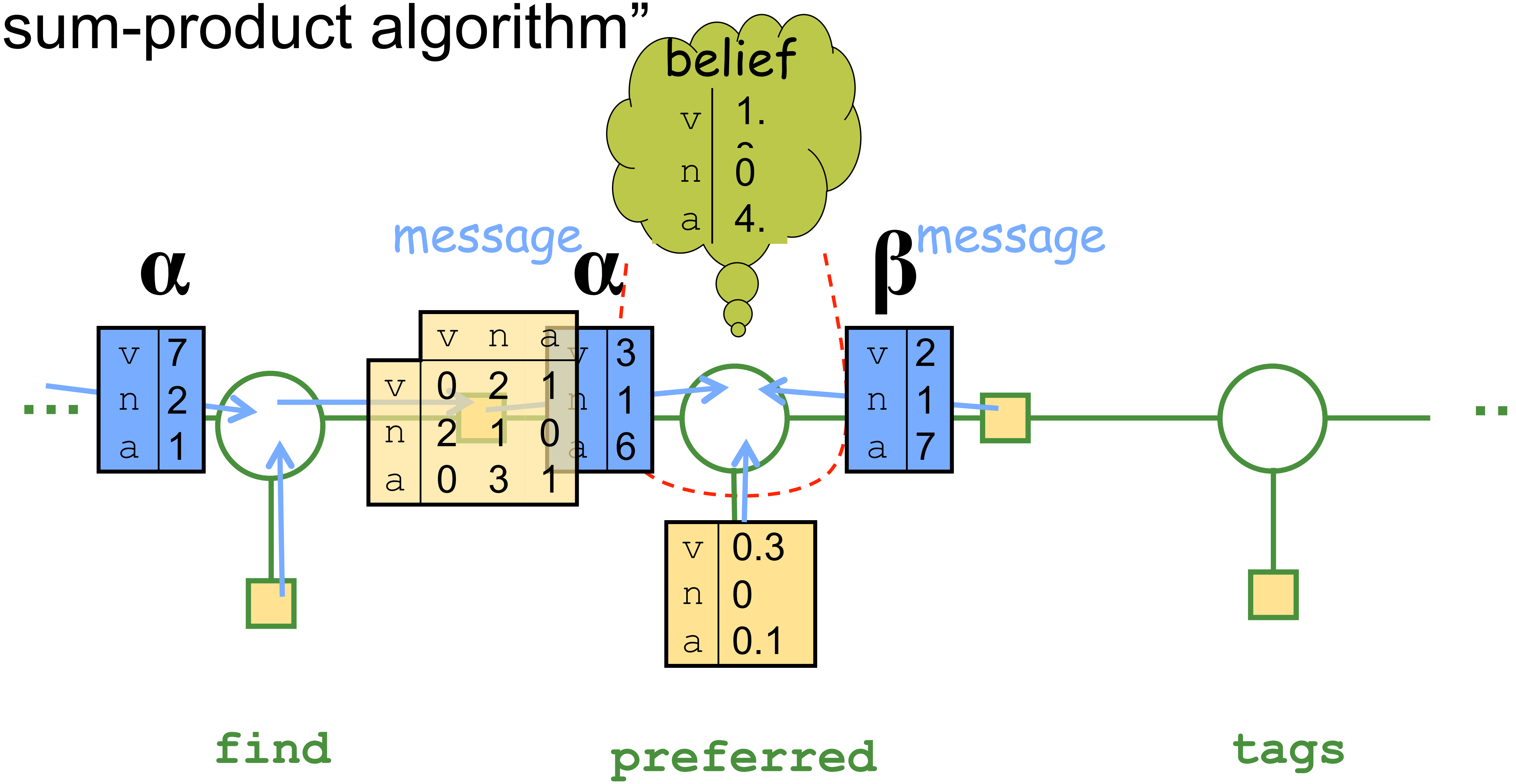
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= "sum-product algorithm"



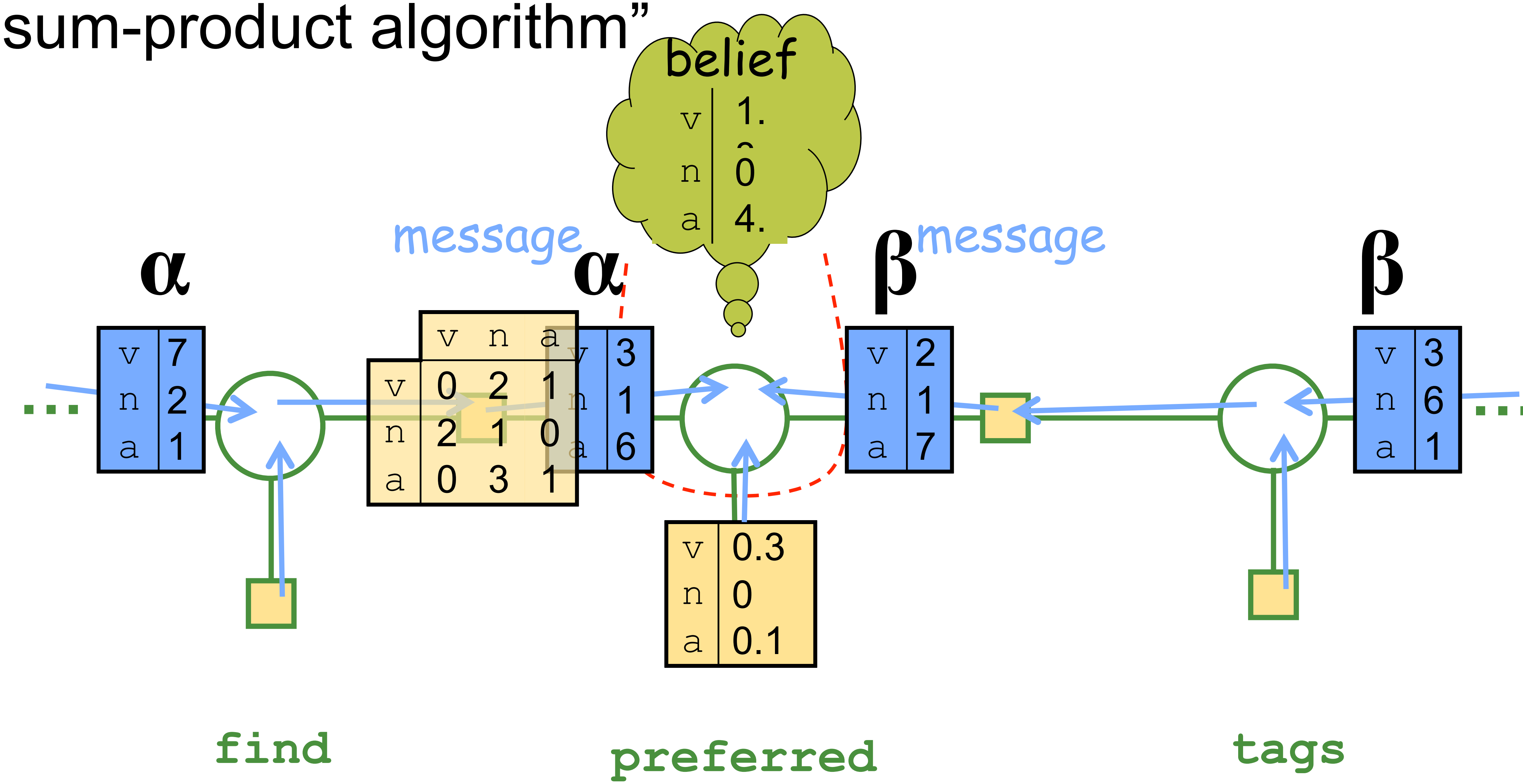
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= “sum-product algorithm”



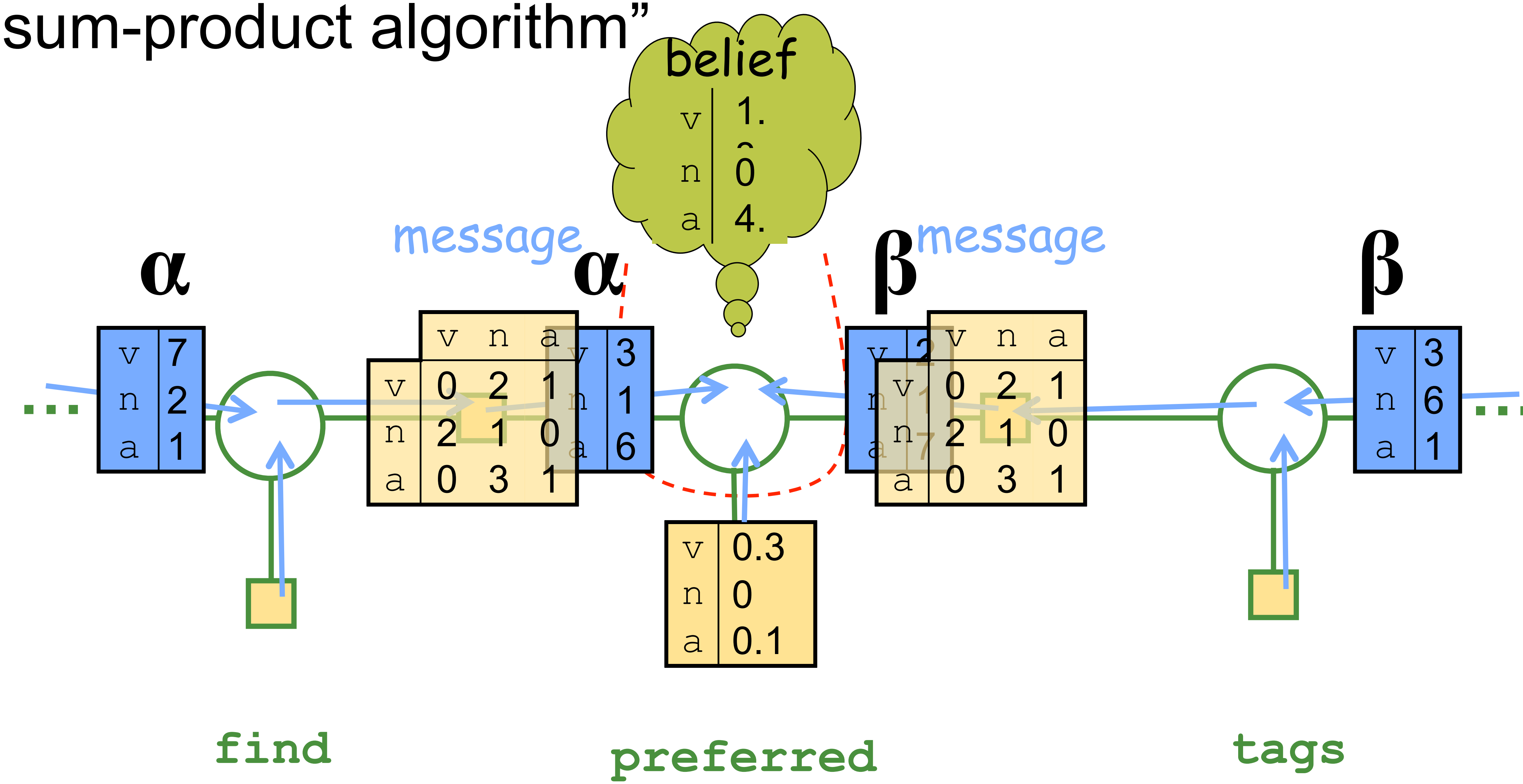
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= “sum-product algorithm”



Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward= “sum-product algorithm”



Sum-Product Equations

- Message from variable *v* to factor *f*

$$m_{v \to f}(x) = \prod_{f' \in N(v) \setminus \{f\}} m_{f' \to v}(x)$$

- Message from factor *f* to variable *v*

Recipe for Conditional Training of p(y | x)

1. Gather constraints/features from training data

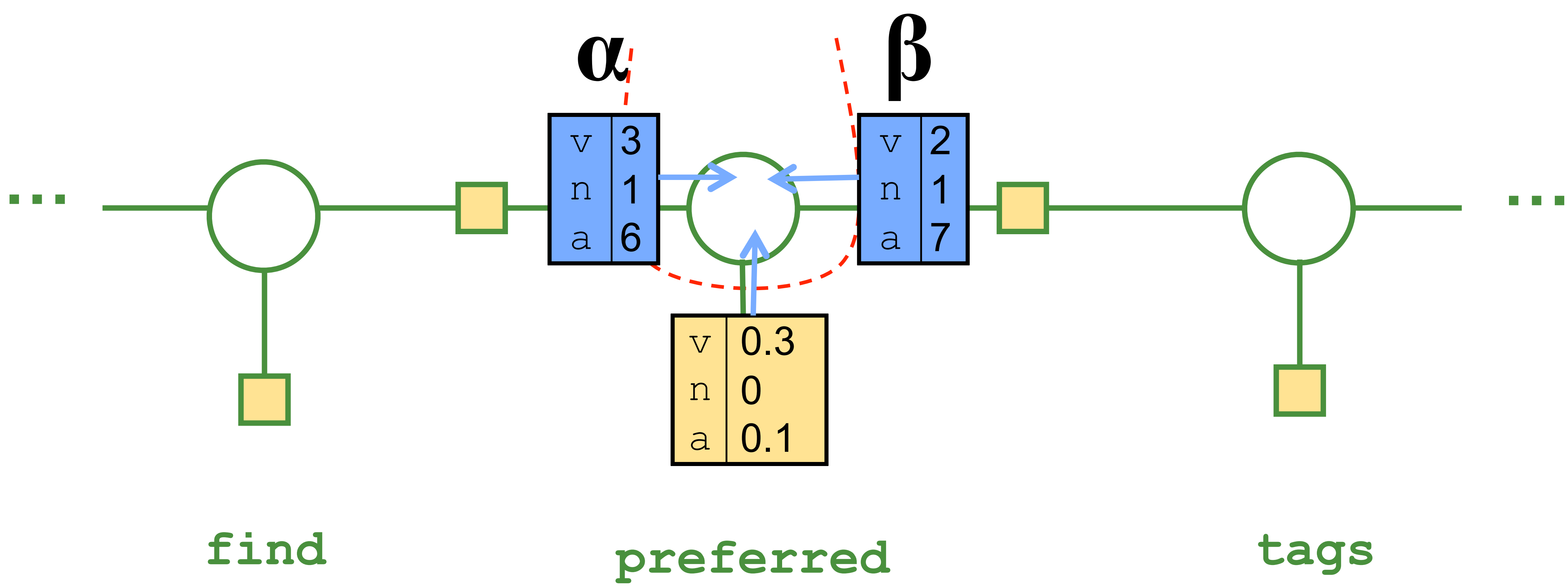
$$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero
3. Classify training data with current parameters; calculate expectations

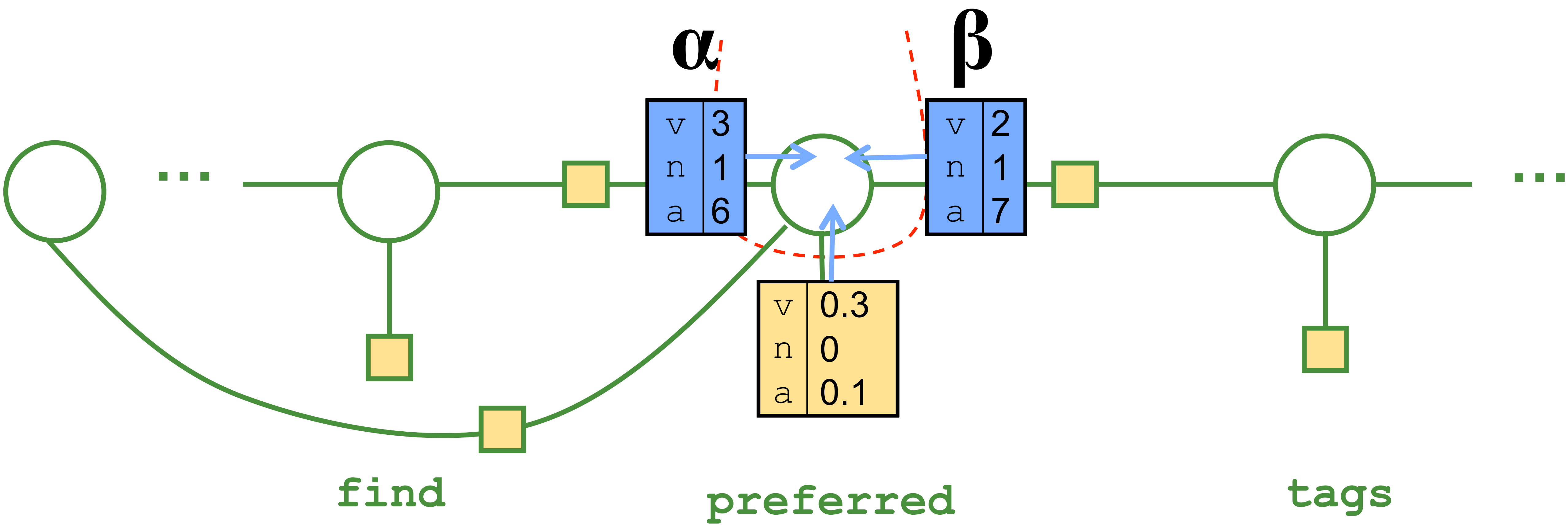
$$E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y'|x_j) f_{iy}(x_j, y')$$

4. Gradient is $\tilde{E}[f_{iy}] - E_\Theta[f_{iy}]$
5. Take a step in the direction of the gradient
6. Repeat from 3 until convergence

Great ideas in ML: Forward-Backward

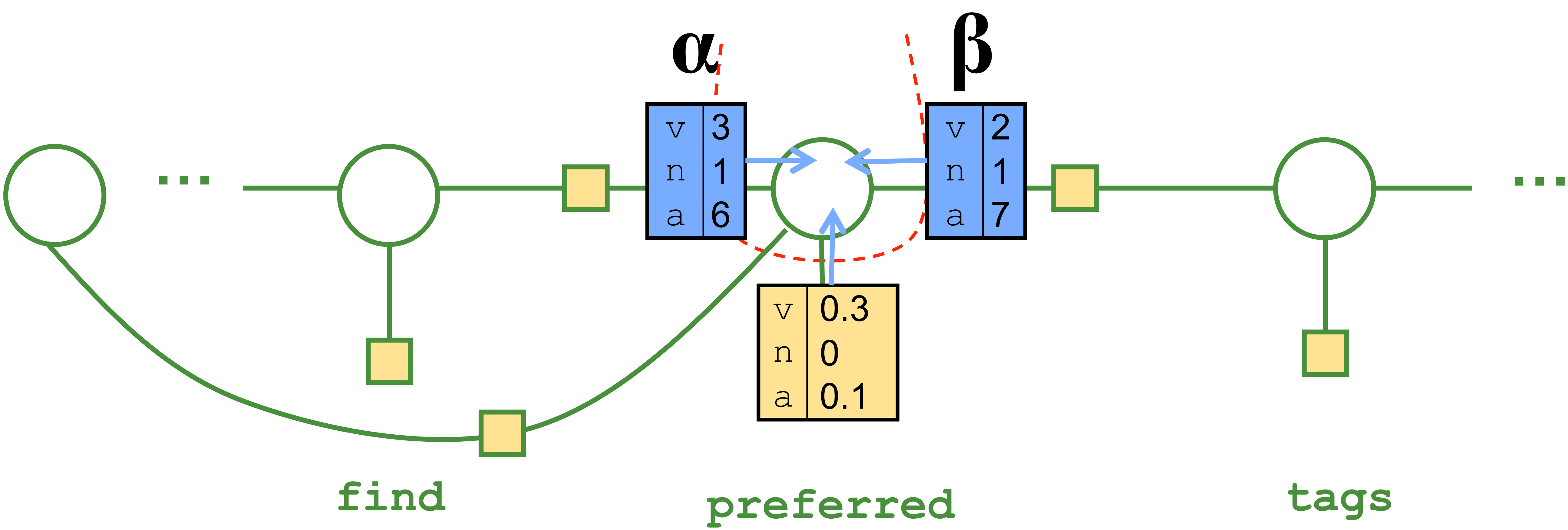


Great ideas in ML: Forward-Backward



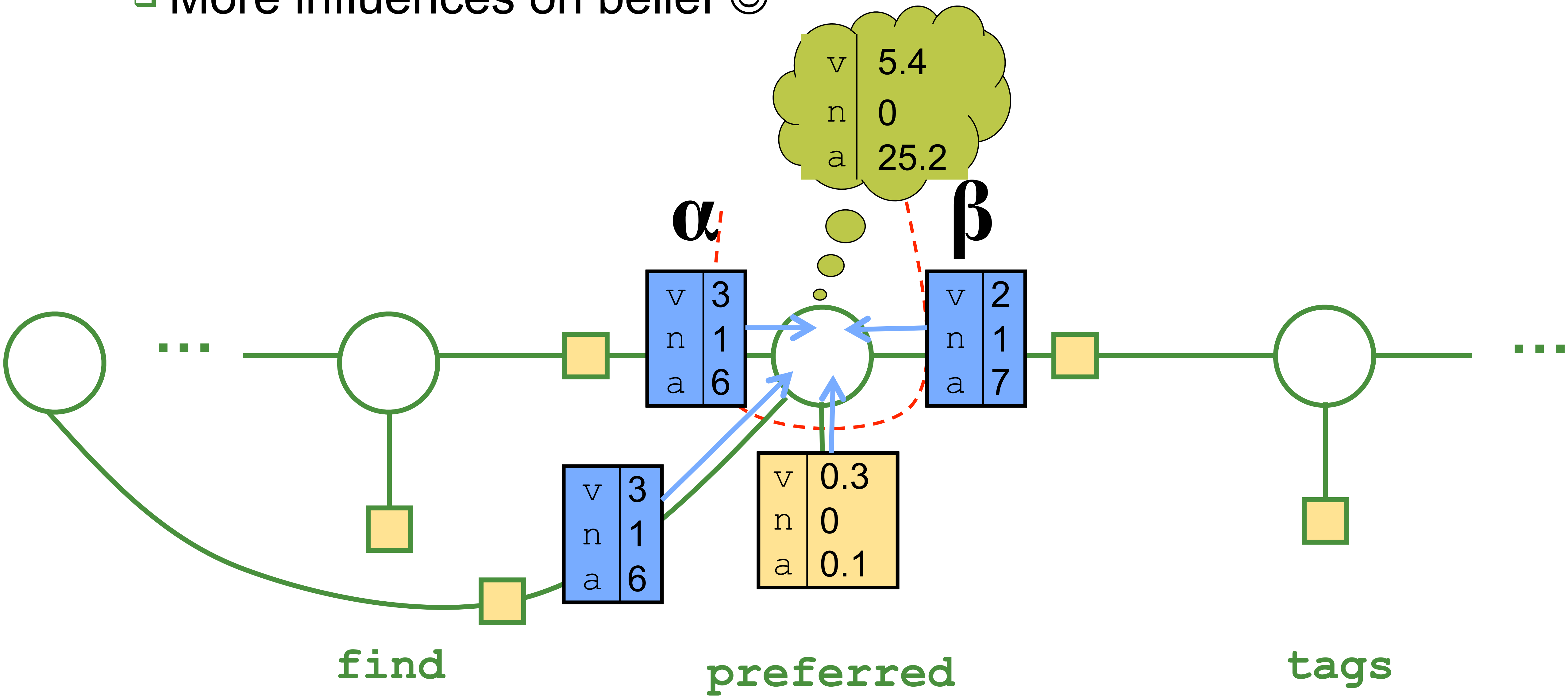
Great ideas in ML: Forward-Backward

- Extend CRF to "skip chain" to capture non-local factor



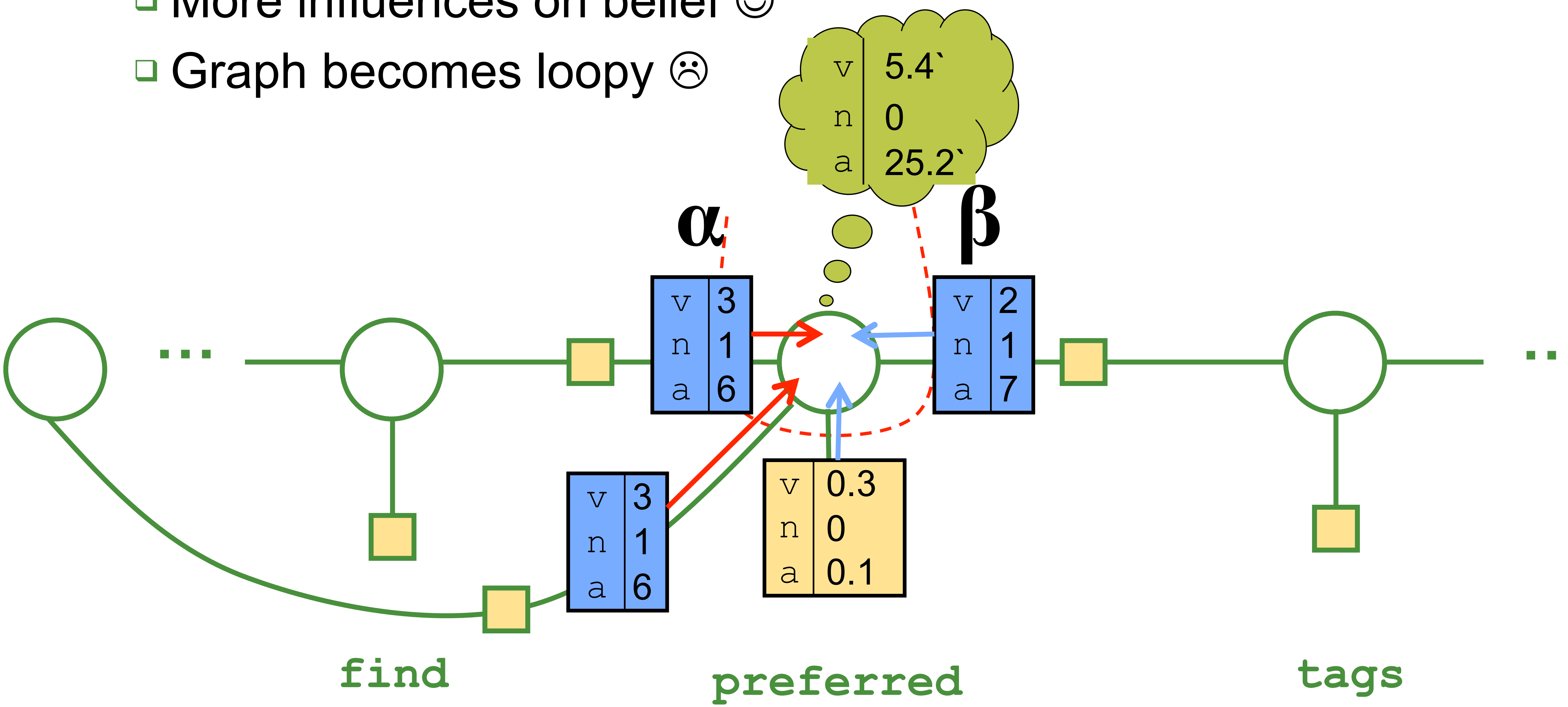
Great ideas in ML: Forward-Backward

- Extend CRF to "skip chain" to capture non-local factor
  - More influences on belief ☺



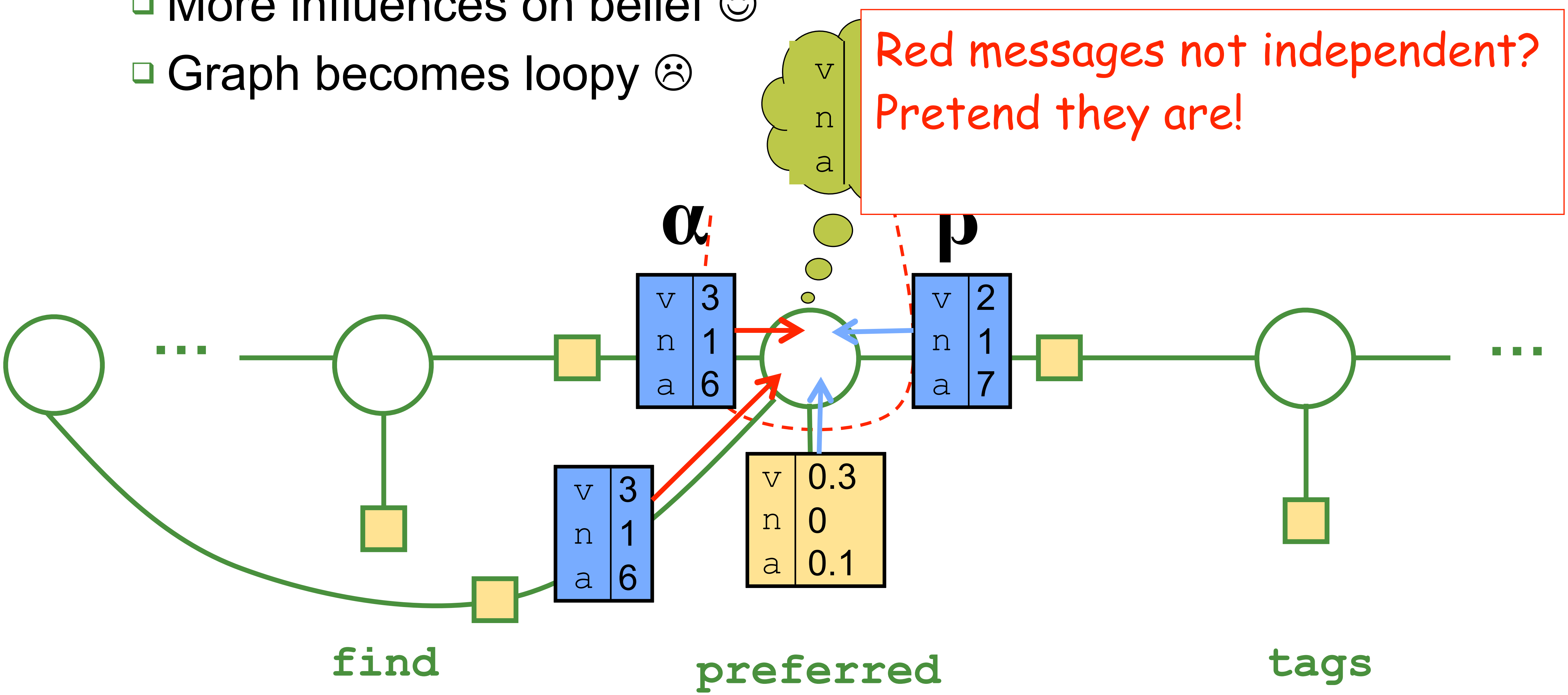
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief ☺
  - Graph becomes loopy ☹



Great ideas in ML: Forward-Backward

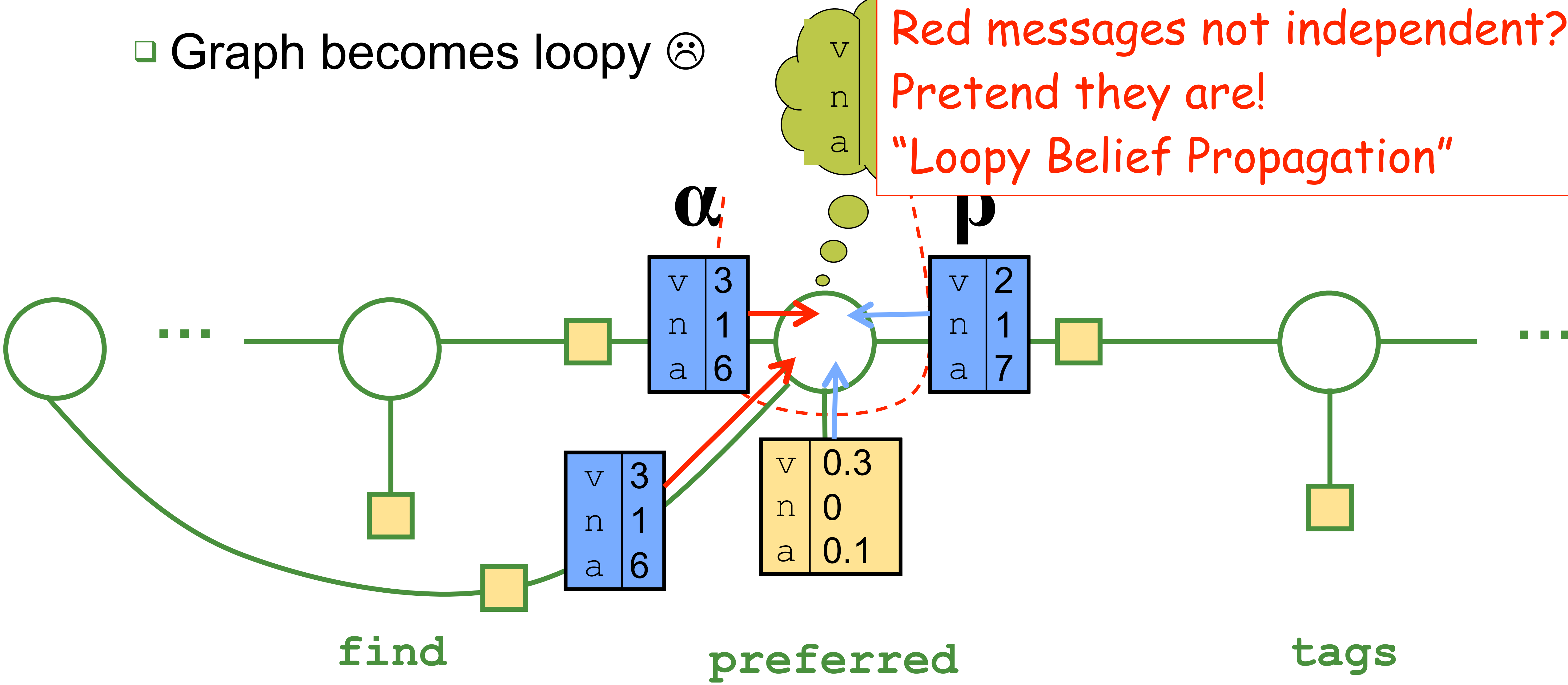
- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief ☺
  - Graph becomes loopy ☹



Great ideas in ML: Forward-Backward

- Extend CRF to "skip chain" to capture non-local factor
  - More influences on belief ☺
  - Graph becomes loopy ☹

Red messages not independent?
Pretend they are!
"Loopy Belief Propagation"



Terminological Clarification

propagation

Terminological Clarification

belief propagation

Terminological Clarification

`loopy     belief   propagation`

Terminological Clarification

[**`loopy`** **`belief`**] **`propagation`**

Terminological Clarification

[loopy belief] propagation

Terminological Clarification

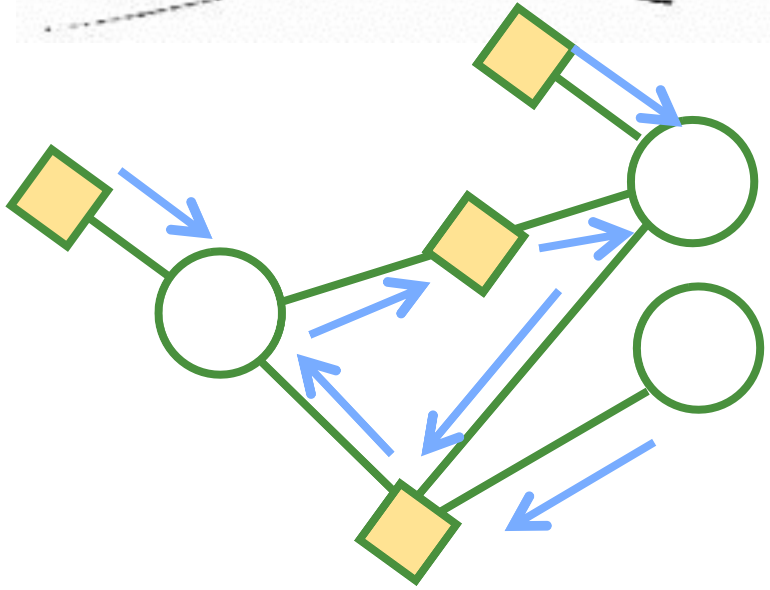
[loopy belief] propagation

loopy [belief propagation]

Terminological Clarification

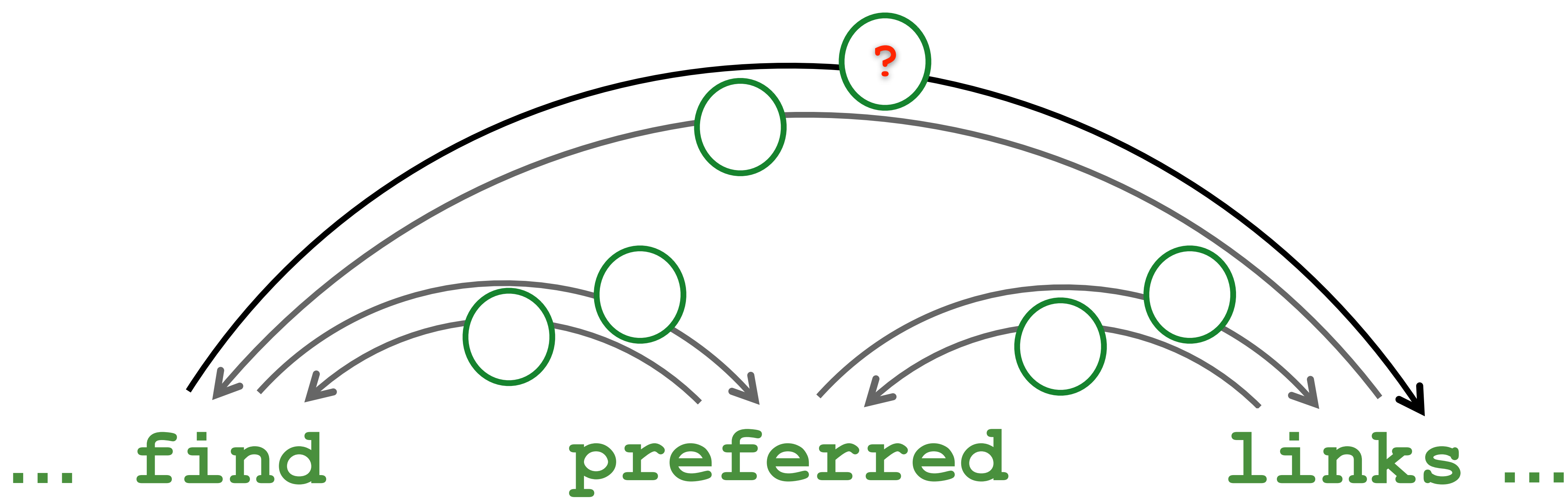
loopy belief propagation

loopy belief propagation



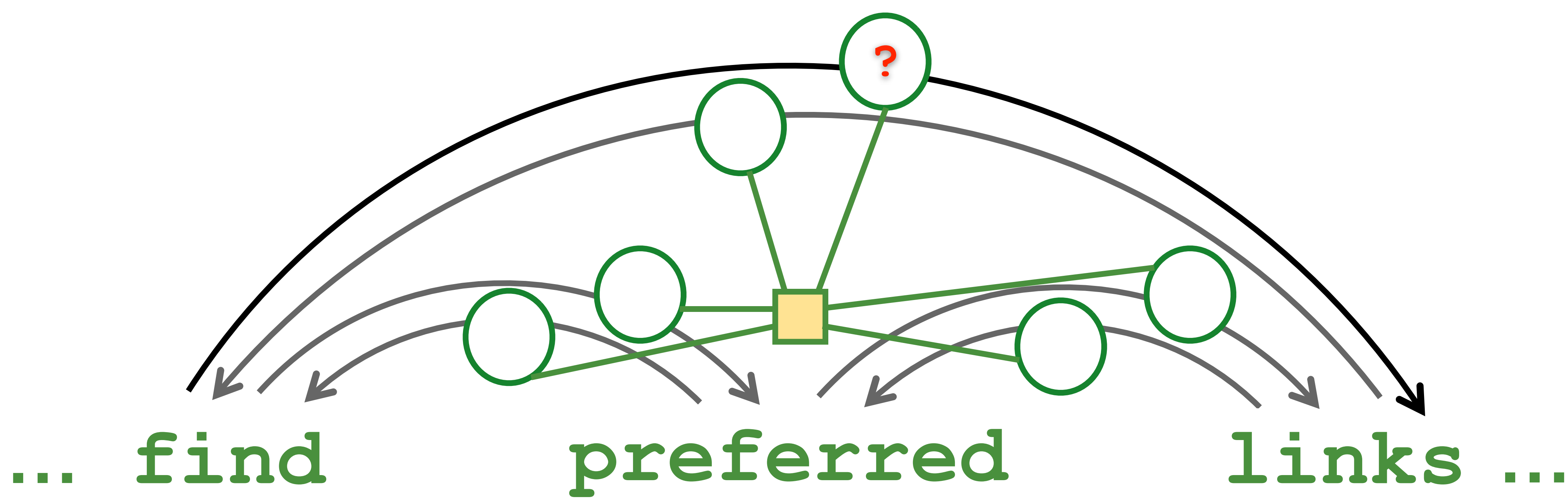
Propagating Global Factors

- Loopy belief propagation is easy for local factors
- How do combinatorial factors (like TREE) compute the message to the link in question?
  - "Does the TREE factor think the link is probably **t** given the messages it receives from *all* the other links?"



Propagating Global Factors

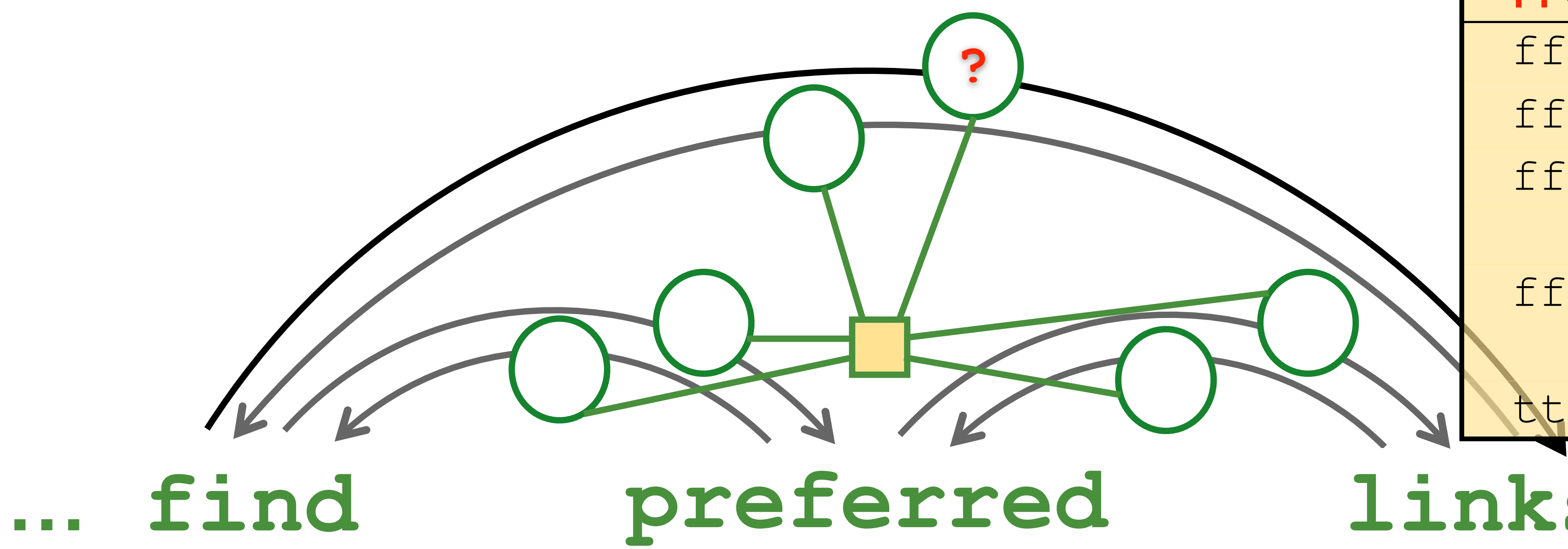
- Loopy belief propagation is easy for local factors
- How do combinatorial factors (like TREE) compute the message to the link in question?
  - "Does the TREE factor think the link is probably **t** given the messages it receives from *all* the other links?"



Propagating Global Factors

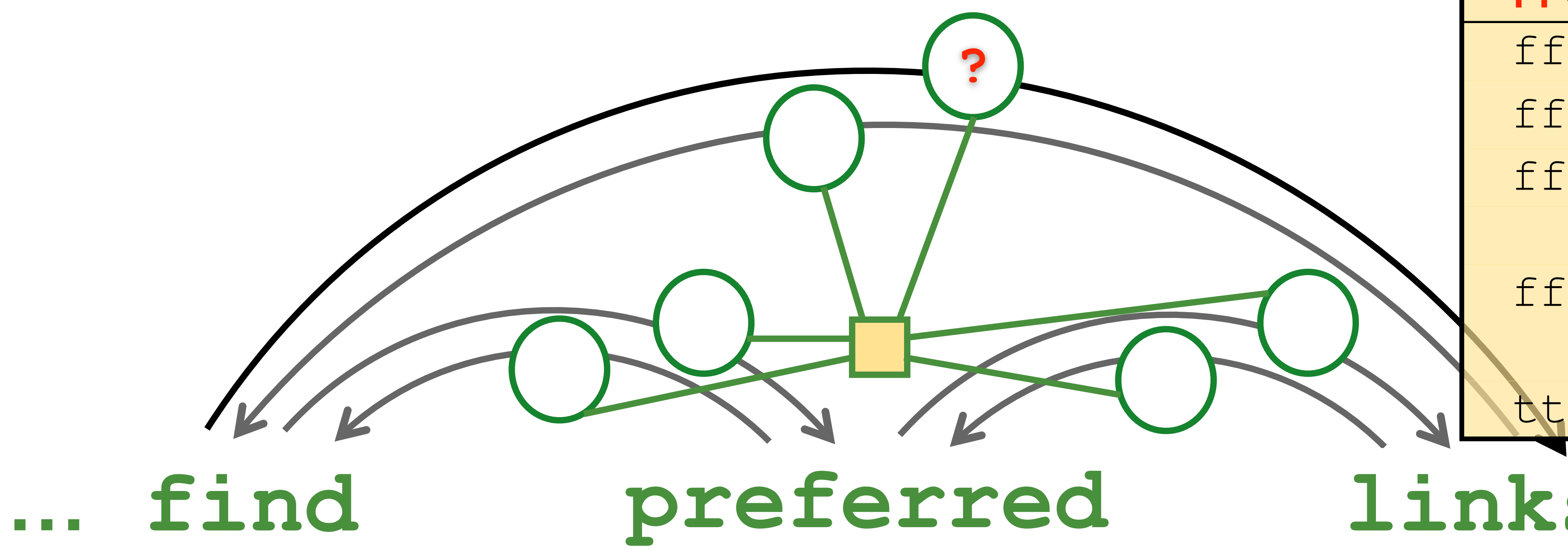
- Loopy belief propagation is easy for local factors
- How do combinatorial factors (like TREE) compute the message to the link in question?
  - "Does the TREE factor think the link is probably **t** given the messages it receives from *all* the other links?"

TREE factor	
ffffff	0
ffffft	0
fffftf	0
…	…
ff**t**ff**t**	1
…	…
tttttt	0



Propagating Global Factors

- How does the TREE factor compute the message to the link in question?
  - “Does the TREE factor think the link is probably **t** given the messages it receives from *all* the other links?”



TREE factor	
ffffff	0
ffffft	0
fffftf	0
...	...
ff**t**ff**t**	1
...	...
tttttt	0

Propagating Global Factors

- How does the TREE factor compute the message to the link in question?
  - "Does the TREE factor think the link is probably **t** given the messages it receives from *all* the other links?"

Old-school parsing to the rescue!

This is the outside probability of the link in an *edge-factored* parser!

∴ TREE factor computes all outgoing messages at once
(given all incoming messages)

Projective case: total $O(n^3)$ time by inside-outside

Non-projective: total $O(n^3)$ time by inverting Kirchhoff matrix

Graph Theory to the Rescue!

Tutte's **Matrix-Tree Theorem** (1948)

The **determinant** of the Kirchoff (aka Laplacian) adjacency matrix of directed graph *G* without row and column *r* is equal to the **sum of scores of all directed spanning trees** of *G* rooted at node *r*.

Graph Theory to the Rescue!

Tutte's **Matrix-Tree Theorem** (1948)

The **determinant** of the Kirchoff (aka Laplacian) adjacency matrix of directed graph *G* without row and column *r* is equal to the **sum of scores of all directed spanning trees** o rooted at node *r*.

Exactly the *Z* we need!

Graph Theory to the Rescue!

$O(n^3)$ time!

Tu s **Matrix-Tree Theorem** (1948)

The **determinant** of the Kirchoff (aka Laplacian) adjacency matrix of directed graph G without row and column r is equal to the **sum of scores of all directed spanning trees** o rooted at node r.

Exactly the Z we need!

Kirchoff (Laplacian) Matrix

$$\begin{bmatrix} 0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\ 0 & 0 & -s(2,1) & \cdots & -s(n,1) \\ 0 & -s(1,2) & 0 & \cdots & -s(n,2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -s(1,n) & -s(2,n) & \cdots & 0 \end{bmatrix}$$

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

Kirchoff (Laplacian) Matrix

$$\begin{bmatrix} 0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\ 0 & 0 & -s(2,1) & \cdots & -s(n,1) \\ 0 & -s(1,2) & 0 & \cdots & -s(n,2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -s(1,n) & -s(2,n) & \cdots & 0 \end{bmatrix}$$

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

Kirchoff (Laplacian) Matrix

$$\begin{bmatrix} 0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\ 0 & \sum_{j\neq 1} s(1,j) & -s(2,1) & \cdots & -s(n,1) \\ 0 & -s(1,2) & \sum_{j\neq 2} s(2,j) & \cdots & -s(n,2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -s(1,n) & -s(2,n) & \cdots & \sum_{j\neq n} s(n,j) \end{bmatrix}$$

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

Kirchoff (Laplacian) Matrix

$$\begin{vmatrix} \sum_{j\neq 1} s(1,j) & -s(2,1) & \cdots & -s(n,1) \\ -s(1,2) & \sum_{j\neq 2} s(2,j) & \cdots & -s(n,2) \\ \vdots & \vdots & \ddots & \vdots \\ -s(1,n) & -s(2,n) & \cdots & \sum_{j\neq n} s(n,j) \end{vmatrix}$$

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

Kirchoff (Laplacian) Matrix

$$
\begin{vmatrix}
\sum_{j\neq 1} s(1,j) & -s(2,1) & \cdots & -s(n,1) \\
-s(1,2) & \sum_{j\neq 2} s(2,j) & \cdots & -s(n,2) \\
\vdots & \vdots & \ddots & \vdots \\
-s(1,n) & -s(2,n) & \cdots & \sum_{j\neq n} s(n,j)
\end{vmatrix}
$$

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

N.B.: This allows multiple children of root, but see Koo et al. 2007.

Transition-Based Parsing

- Linear time
- Online
- Train a classifier to predict next action
- Deterministic or beam-search strategies
- But... generally less accurate

Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Start state: $([\,], [1, \ldots, n], \{\,\})$

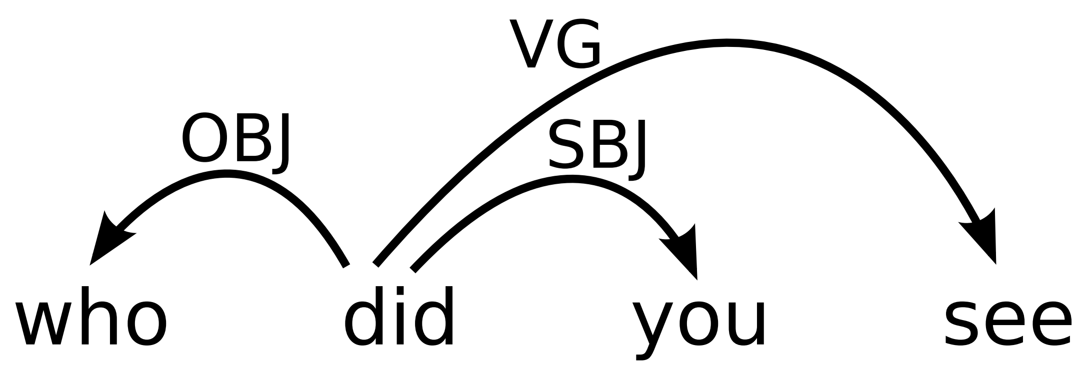
Final state: $(S, [\,], A)$

Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$
Reduce:	$(S\|i, B, A)$	$\Rightarrow$	$(S, \mathrm{B}, \mathrm{A})$
Right-Arc:	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S\|i\|j, B, A \cup \{i \rightarrow j\})$
Left-Arc:	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S, j\|B, A \cup \{i \leftarrow j\})$

Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\ ]_S$	$[\text{who, did, you, see}]_B$	{ }

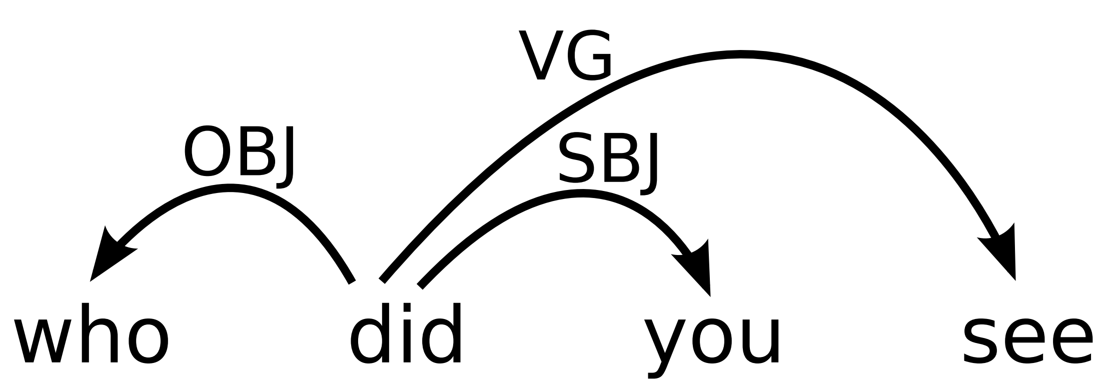


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\text{who}]_S$	$[\text{did, you, see}]_B$	{ }

Shift

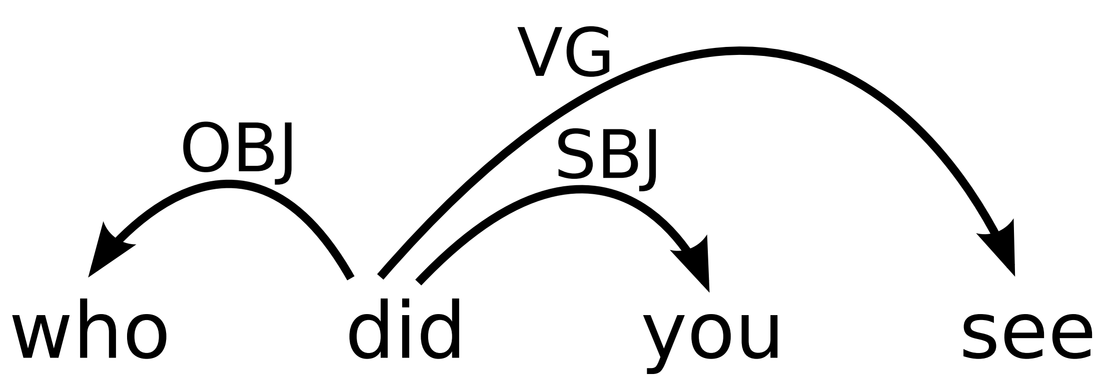


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\,]_S$	$[\text{did, you, see}]_B$	$\{\text{ who } \xleftarrow{\text{OBJ}} \text{ did }\}$

Left-arc
OBJ

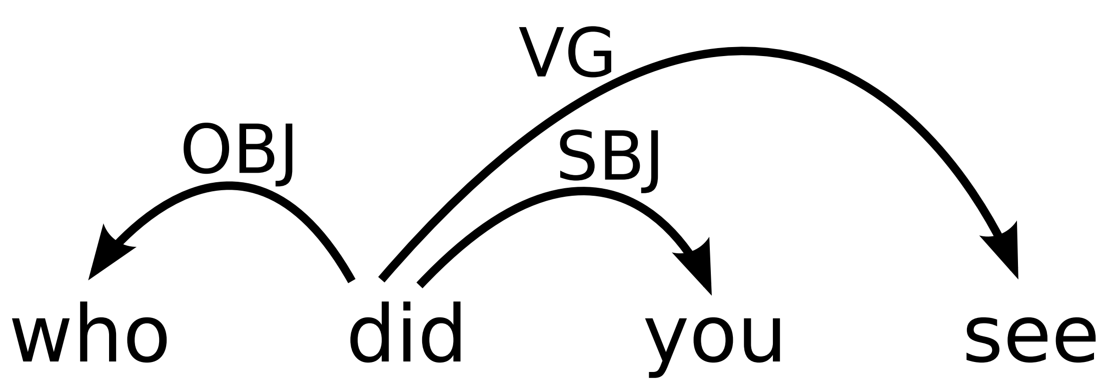


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\text{did}]_S$	$[\text{you, see}]_B$	$\{\text{ who } \xleftarrow{\text{OBJ}} \text{ did }\}$

Shift

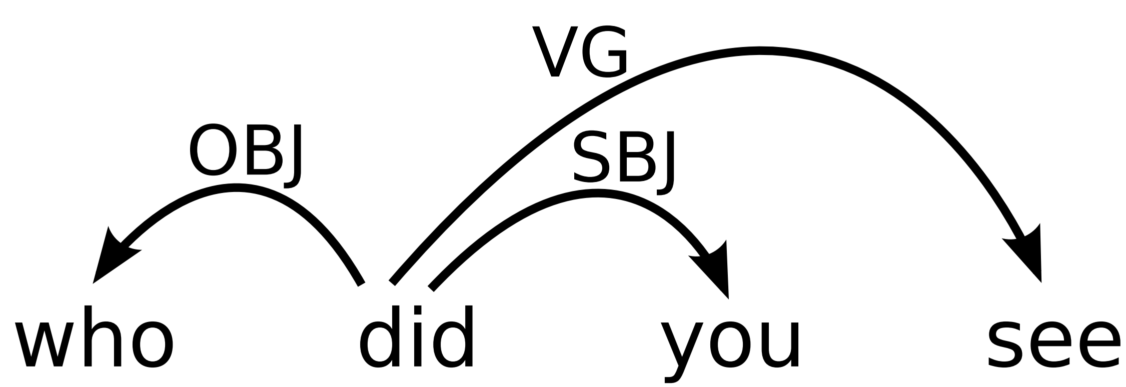


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\text{did, you}]_S$	$[\text{see}]_B$	$\{\ \text{who} \xleftarrow{\text{OBJ}} \text{did},\ \text{did} \xrightarrow{\text{SBJ}} \text{you}\ \}$

Right-arc SBJ

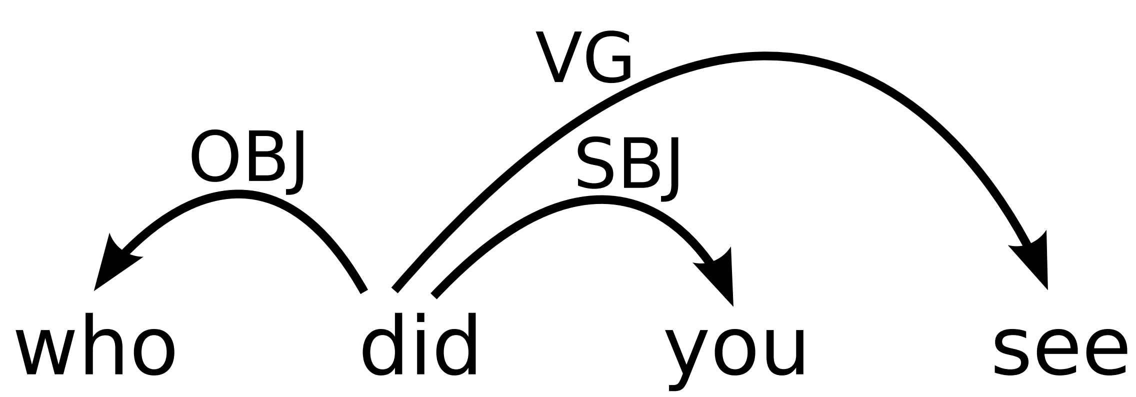


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\text{did}]_S$	$[\text{see}]_B$	$\{ \text{who} \xleftarrow{\text{OBJ}} \text{did}, \; \text{did} \xrightarrow{\text{SBJ}} \text{you} \}$

Reduce

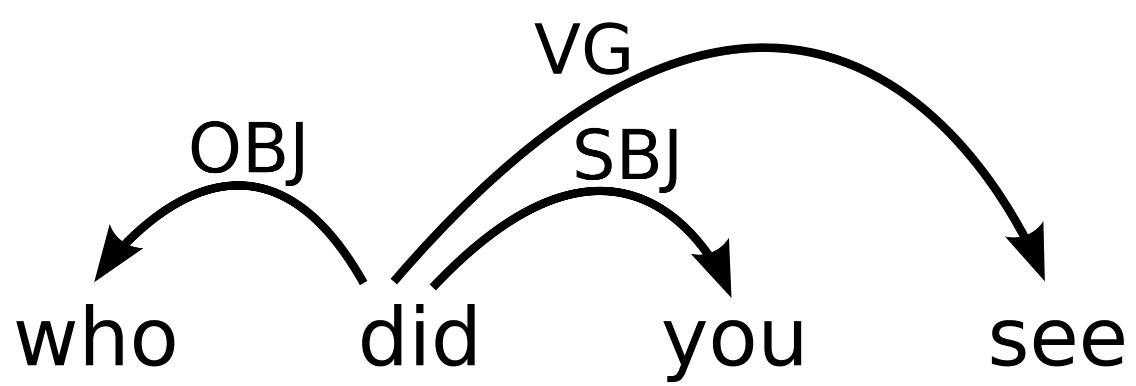


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
[did, see]$_S$	[ ]$_B$	$\{ \text{who} \xleftarrow{\text{OBJ}} \text{did}, \ \text{did} \xrightarrow{\text{SBJ}} \text{you}, \ \text{did} \xrightarrow{\text{VG}} \text{see} \}$

Right-arc
VG

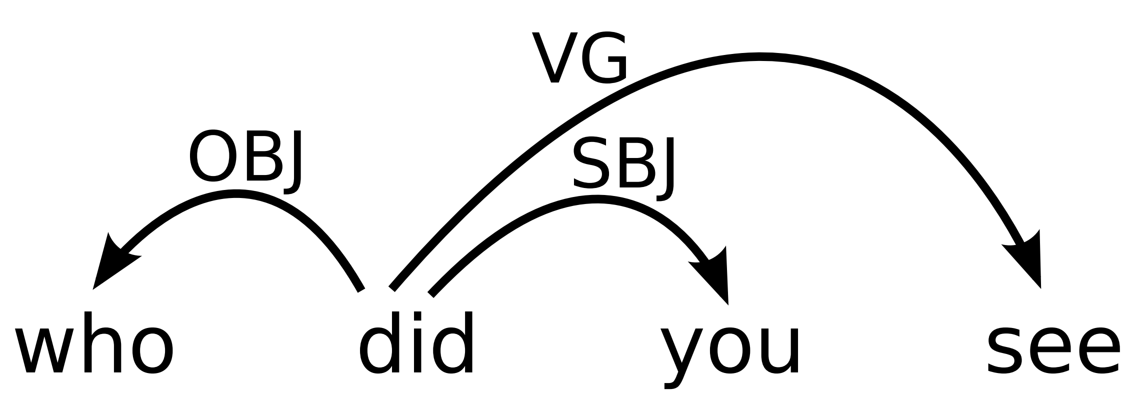


Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack	**Buffer**	**Arcs**
$[\text{did, you}]_S$	$[\text{see}]_B$	$\{\ \text{who} \xleftarrow{\text{OBJ}} \text{did},$ $\text{did} \xrightarrow{\text{SBJ}} \text{you}\ \}$

Right-arc SBJ



Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack **Buffer** **Arcs**

$[\text{did, you}]_S$ $[\text{see}]_B$ $\{\text{who} \xleftarrow{\text{OBJ}} \text{did}, \ \text{did} \xrightarrow{\text{SBJ}} \text{you}\}$

Right-arc SBJ

VG

Choose action w/best classifier score
100k - 1M features

Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack **B**

[did, you]$_S$ [s

Very fast linear-time performance
WSJ 23 (2k sentences) in 3 s

did $\longrightarrow$ you }

Right-arc
SBJ

VG

Choose action w/best classifier score
100k - 1M features