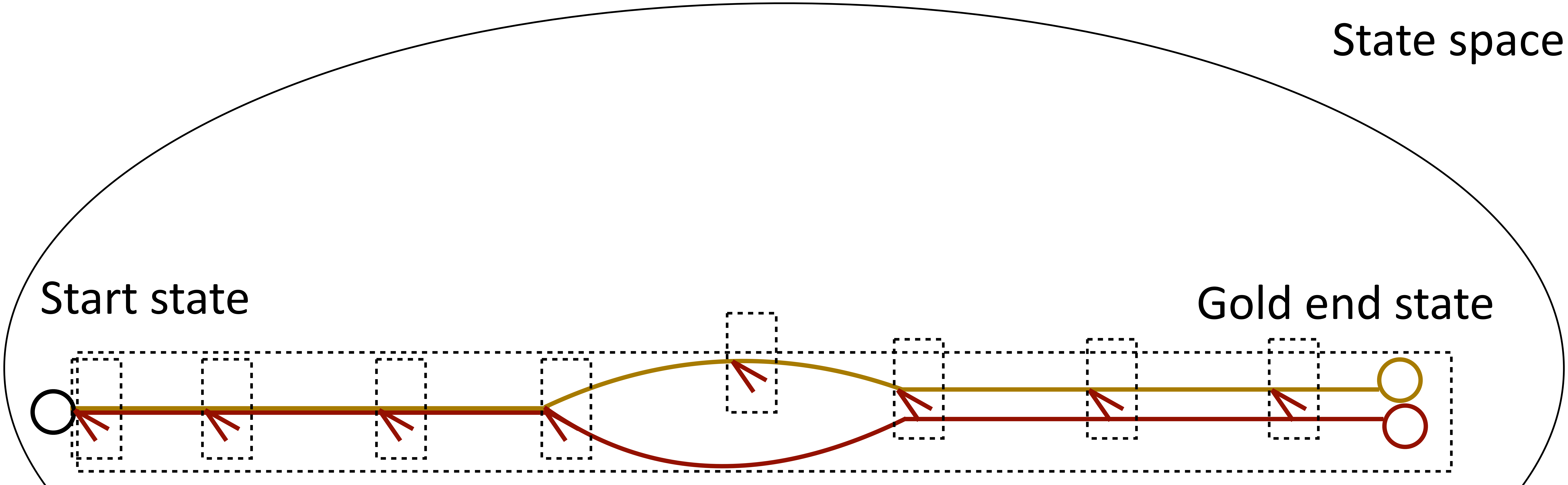
CS395T: Structured Models for NLP
Lecture 10: Loopy Graphical Models

Greg Durrett

Recall: Global vs. Greedy



- Greedy: 2n local training examples, only see gold states
- Global: one global example, might see new states

Recall: Global Training with Early Updating

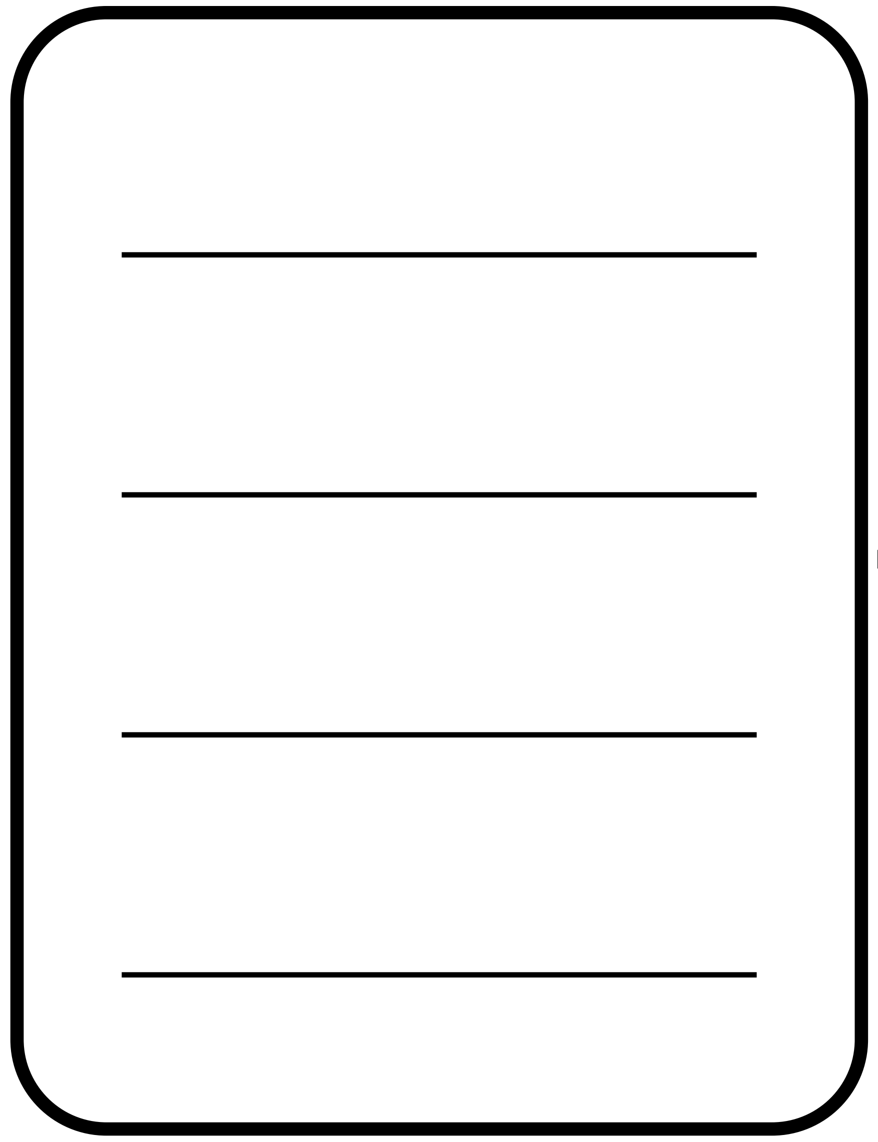
```
For each epoch
  For each sentence
    For i=1…2*len(sentence)      # 2n transitions in arc-standard
      beam[i] = compute_successors(beam[i-1])
      If beam[i] does not contain gold:
        # Feats are cumulative up until this point
        apply_gradient_update(feats(gold[0:i]) - feats(beam[i,0]))
        break
    # If we got to the end, gold may still not be one-best
    If i == 2*len(sentence):
      apply_gradient_update(feats(gold) - feats(beam[2*len(sentence),0]))
```

Administrivia

- Survey results: pace a bit too fast (assumes too much prior knowledge)
  - Fast pace for a couple of lectures on graph-structured models, classical machine translation
  - More moderate pace on fundamentals of NNs / RNNs / neural MT
- Details for projects: I'll try to do this more
- Frontiers / current research: after RNNs
- More materials: precision/recall of readings?
- "Don't have expectations for the final project"
- It starts at 9:30am: sorry :(

Road Map

Text



Text Analysis

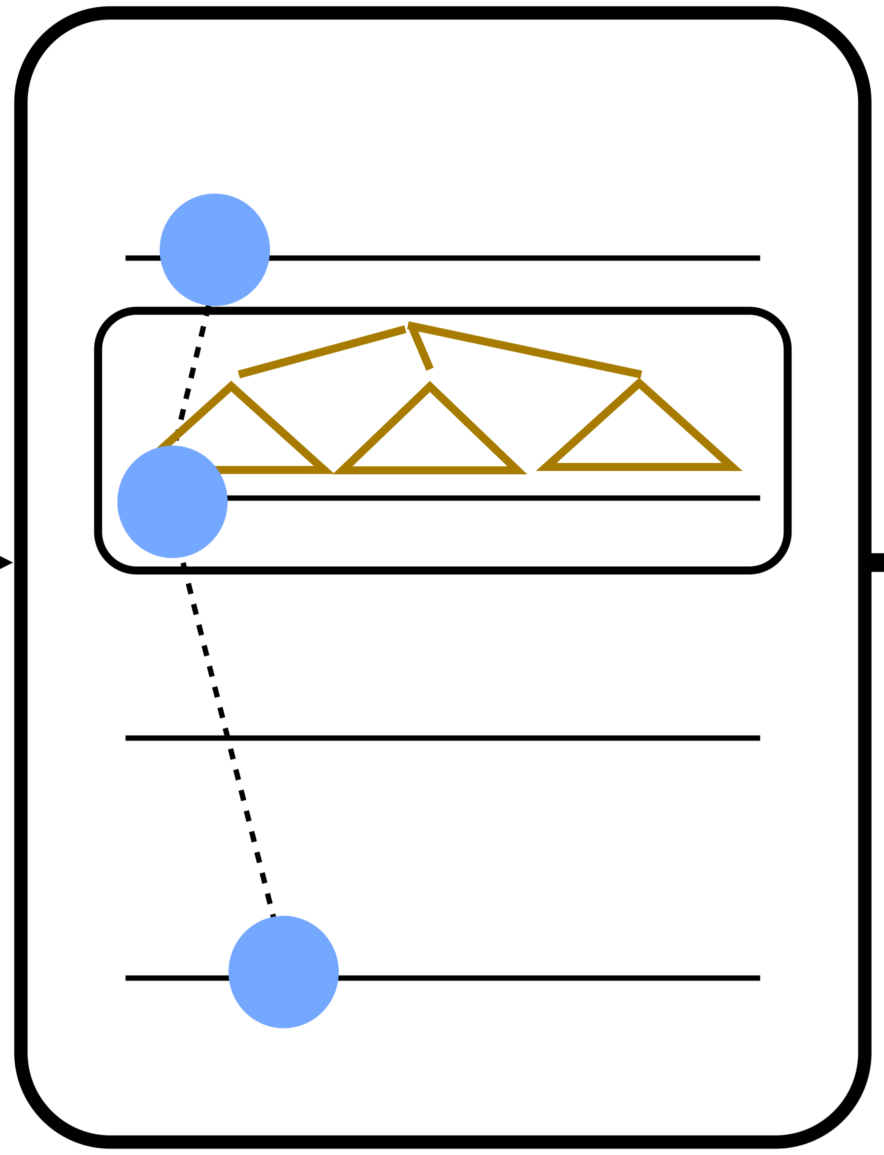
POS tagging

Syntactic parsing

NER

Coreference resolution

Annotations



Applications

Summarize

Extract information

Answer questions

Identify sentiment

Translate

- First half of the class: more text analysis
- Second half of the class: more applications

Road Map

- Sequences: POS, NER

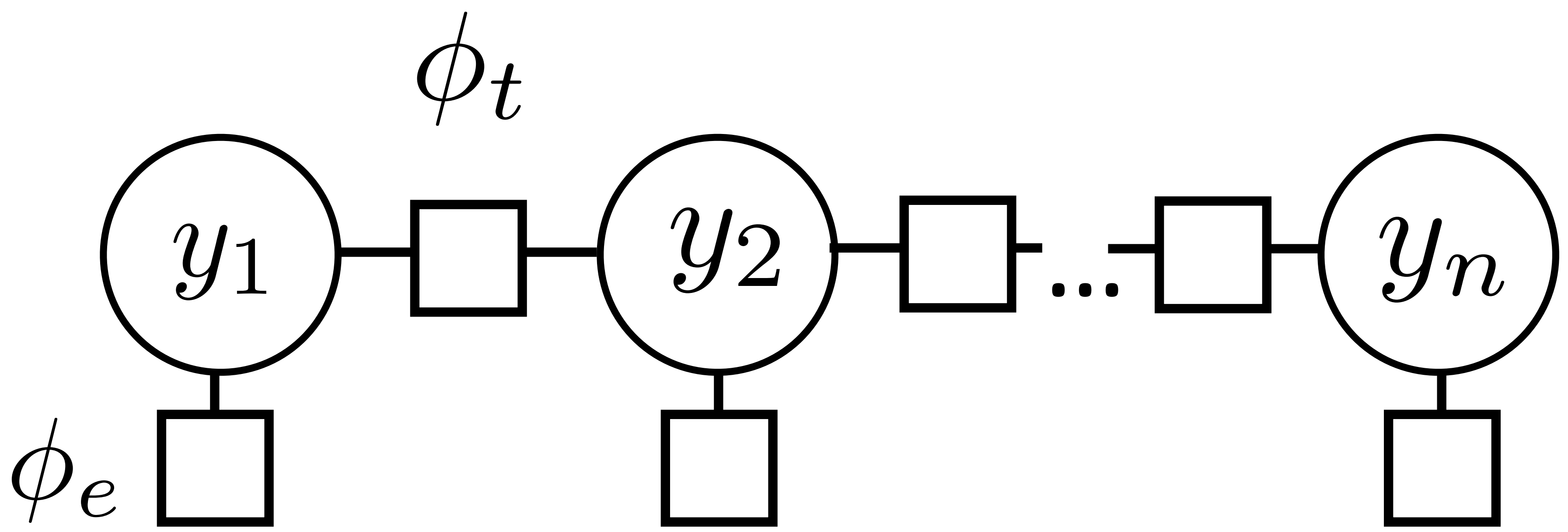
- Trees: constituency parsing, dependency parsing, semantic role labeling

- Today: graph-structured models with two inference techniques: belief propagation, Gibbs sampling

- Next time: classical (non-neural) machine translation

- Then, part 2 of the class: neural networks, RNNs, CNNs, etc.

Recall: CRFs



$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{i=2}^{n}\exp(\phi_t(y_{i-1}, y_i))\prod_{i=1}^{n}\exp(\phi_e(y_i, i, \mathbf{x}))$$

- Z is normalizing constant: how did we compute it? And marginals?
- Forward-backward: efficient dynamic program for summing things out

Skip-chain CRFs

ORG

The news agency Tanjug reported on the outcome of the meeting.

ORG?

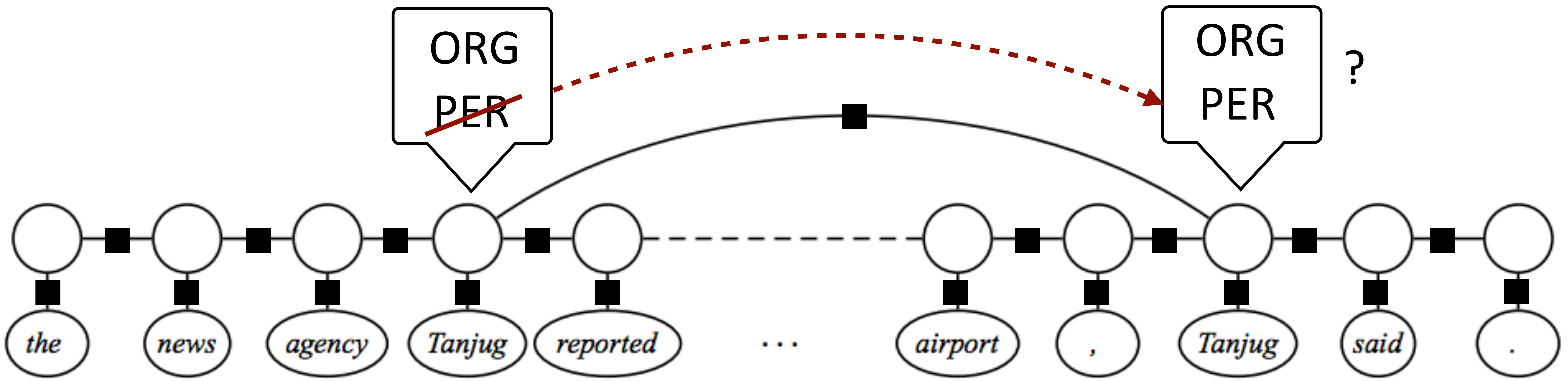
PER?

The delegation met the president at the airport, Tanjug said.

- Coreferent entities — should be the same type
- "One sense per discourse" assumption: "bank" (river) and "bank" (financial) rarely occur in the same context

Finkel and Manning (2008)

Skip-chain CRFs



$$P(\mathbf{y}|\mathbf{x}) \propto \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, \mathbf{x})) \boxed{\prod_{(j,k)} \exp(\phi_\ell(y_j, y_k))}$$

- *(j, k)* are pairs of variables that we manually linked up

Finkel and Manning (2008)

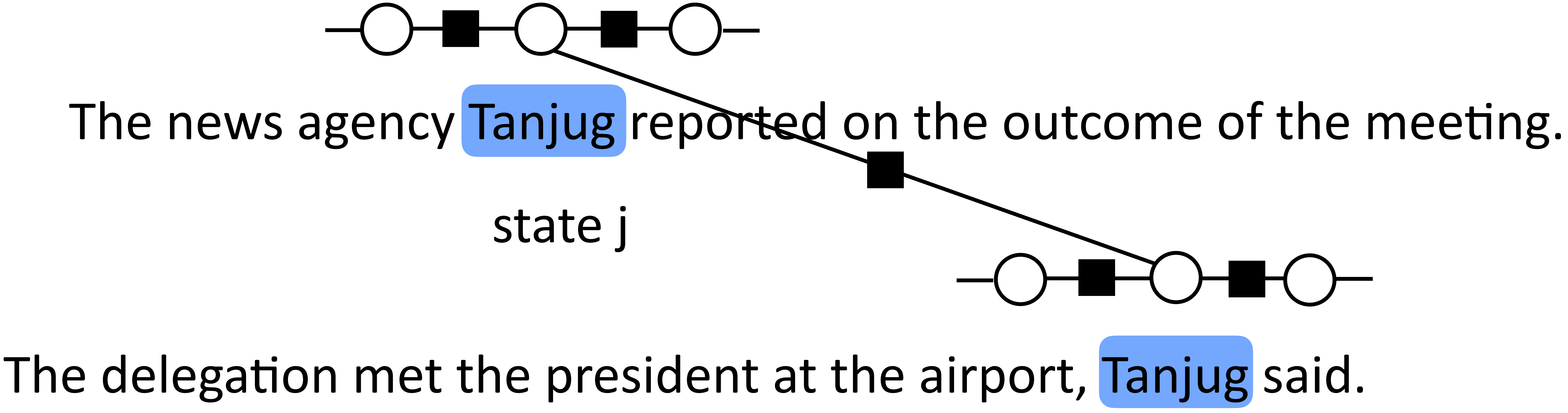
Inference

$$P(\mathbf{y}|\mathbf{x}) \propto \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, \mathbf{x})) \boxed{\prod_{(j,k)} \exp(\phi_\ell(y_j, y_k))}$$

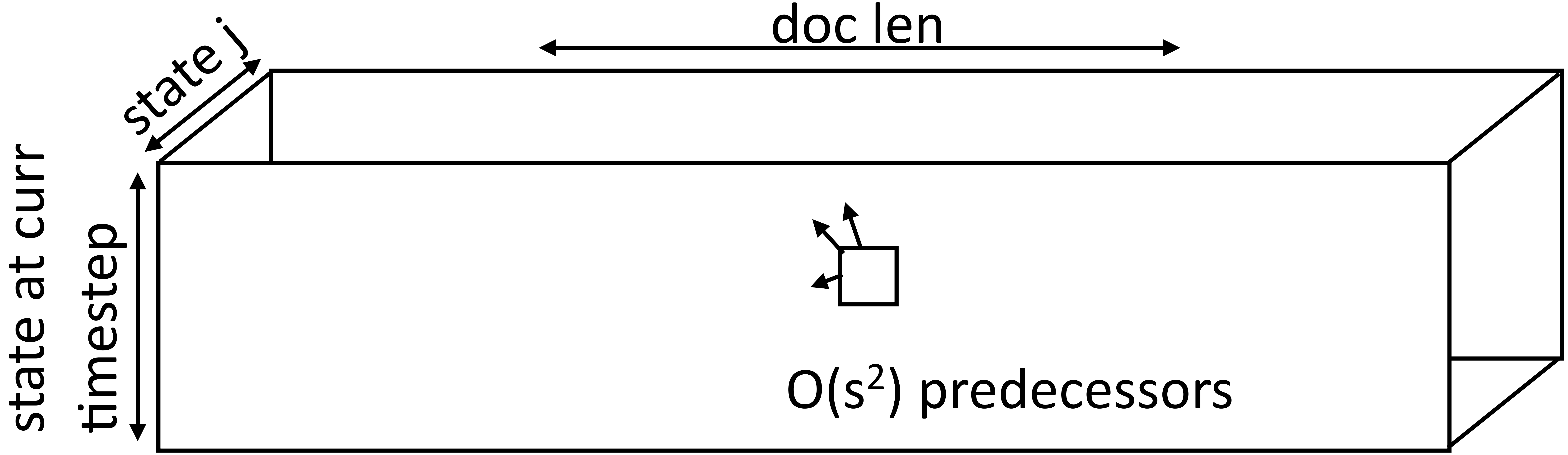
- How do we do forward-backward in this case? Assume just one sentence
- What if there are no links (j, k)?
- What if there's one link (j, k)?
  - Iterate upward through i: keep track of state i-1, keep track of state j
  - Dynamic program now tracks two states, so an extra factor of *s*
- For k links, state blows up by factor of s^k

Finkel and Manning (2008)

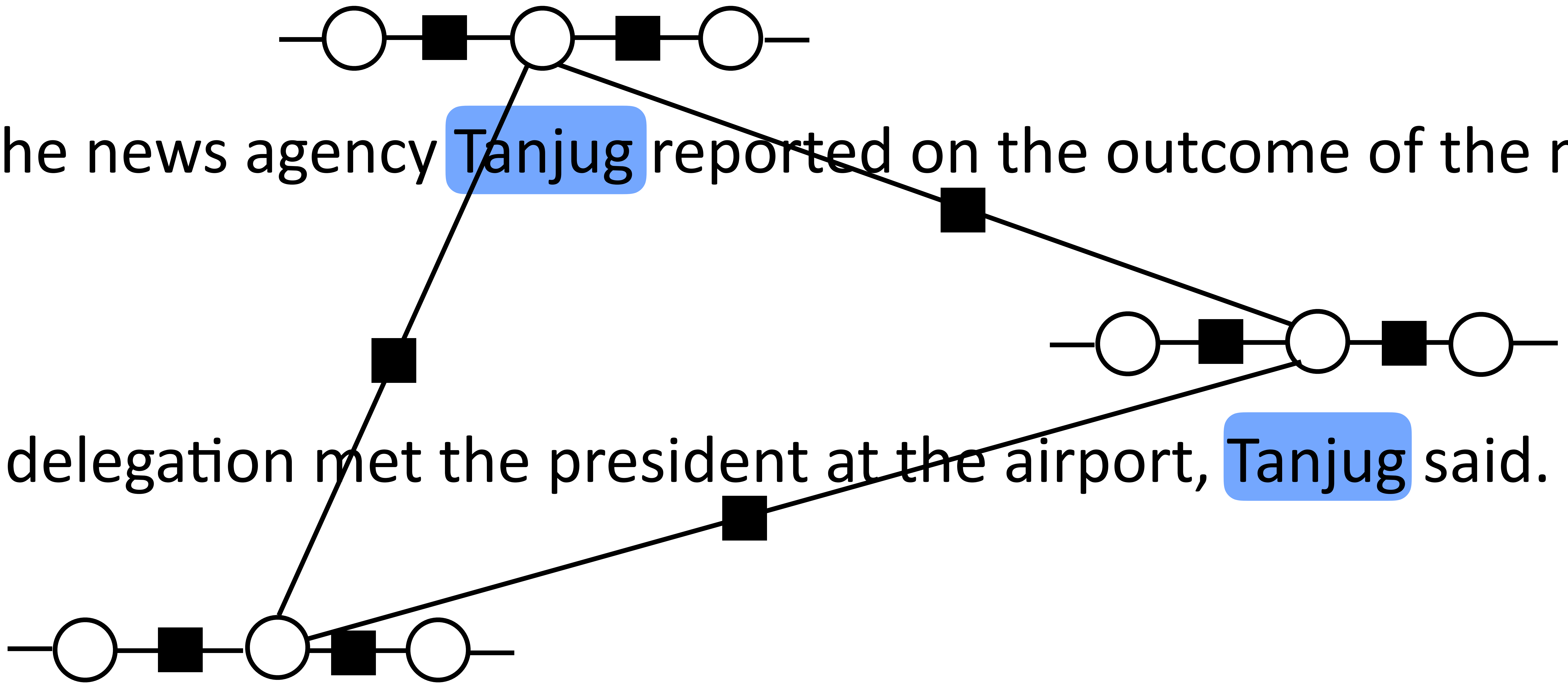
Inference



- Forward matrix (tensor):



Inference



The news agency Tanjug reported on the outcome of the meeting.

The delegation met the president at the airport, Tanjug said.

Yesterday, Tanjug also reported on…

- Now would need to track two prior states…generally becomes intractable

Inference

- Solution 1: Belief propagation

- Solution 2: Gibbs sampling

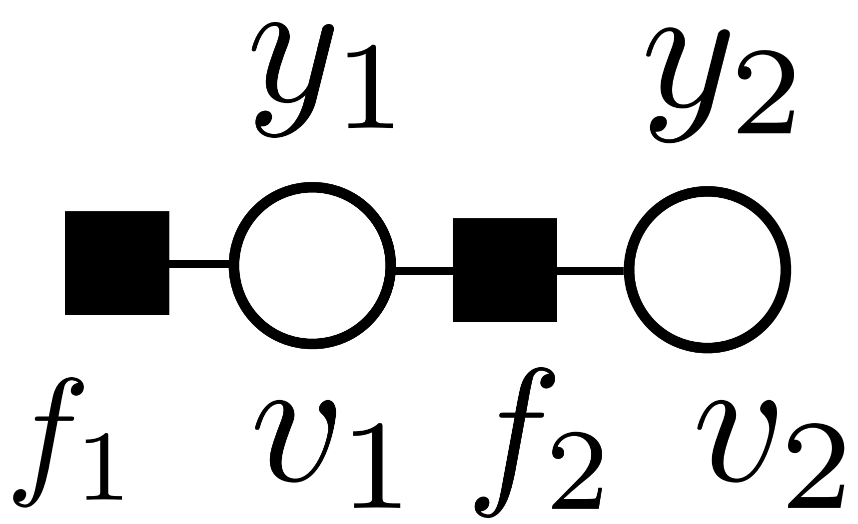
Belief Propagation

Belief Propagation

- Forward-backward: instance of sum-product algorithm for inference in general tree-structured CRFs

- Sum-product doesn’t work when there are loops, but it’s usually a good approximation, so we can just use it anyway

Sum-Product Algorithm



$$P(y_1, y_2|\mathbf{x}) = \frac{\exp(\phi(y_1))\exp(\phi(y_1, y_2))}{\sum_{y_1', y_2'} \exp(\phi(y_1'))\exp(\phi(y_1', y_2'))}$$

- Notation:

  $N(v)$ factors that are neighbors of variable *v* (use *y* for values)

  $N(f)$ variables that are neighbors of factor *f*

  $\mathbf{y}_f$ y values associated with a factor *f*

- "Messages" μ: vectors of values on edges between variable and factor (one message in each direction along edge). "Distributions" over *y*

  - Posteriors are products of messages from factors:

$$P(y_i = y|\mathbf{x}) \propto \prod_{f \in N(v_i)} \mu_{f \to v_i}(y)$$

Sum-Product Algorithm

- V->f messages: $$\mu_{v\to f}(y) = \prod_{f'\in N(v), f'\neq f} \mu_{f'\to v}(y)$$

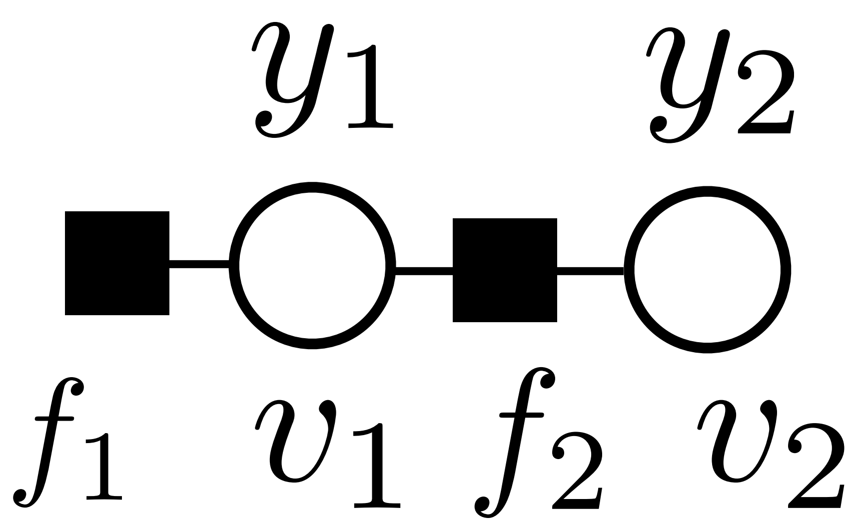
- Value of *y* is a product of what all other incoming messages say about *y*. I.e., propagate information from the rest of the graph, but don't feed the factor its own outputs

- F->v messages:

$$\mu_{f\to v_i}(y_i) = \sum_{\mathbf{y}_{f,-i}} \exp(\phi(\mathbf{y}_{f,-i}, y_i)) \prod_{k:v_k\in N(f), k\neq i} \mu_{v_k\to f}(y_{f,k})$$

  - Sum over all values of ***y*** for this factor with the *i*th coordinate set to y_i
  - Product over all other factors' messages

Sum-Product Algorithm



$$P(y_1, y_2|\mathbf{x}) = \frac{\exp(\phi(y_1))\exp(\phi(y_1, y_2))}{\sum_{y_1', y_2'} \exp(\phi(y_1'))\exp(\phi(y_1', y_2'))}$$

$$\exp(\phi(y_1)) = [0.9, 0.1]$$

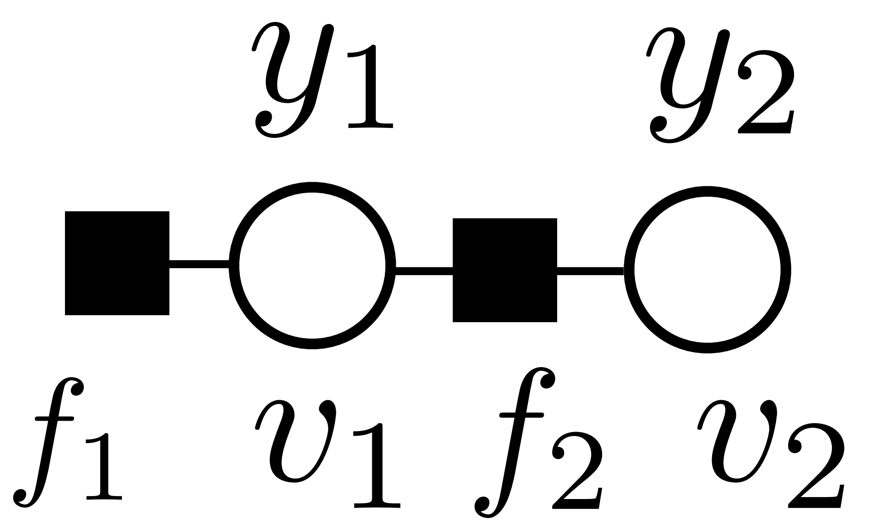
$$\exp(\phi(y_1, y_2)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

- Factor requires $y_1 = y_2$ (these are zeroes in *real space*!)

$$P(y_1, y_2) = \begin{matrix} 0.9 & 0 \\ 0 & 0.1 \end{matrix}$$

- Probability reflects both factors

Sum-Product Algorithm



$$P(y_1, y_2|\mathbf{x}) = \frac{\exp(\phi(y_1))\exp(\phi(y_1, y_2))}{\sum_{y_1', y_2'}\exp(\phi(y_1'))\exp(\phi(y_1', y_2'))}$$

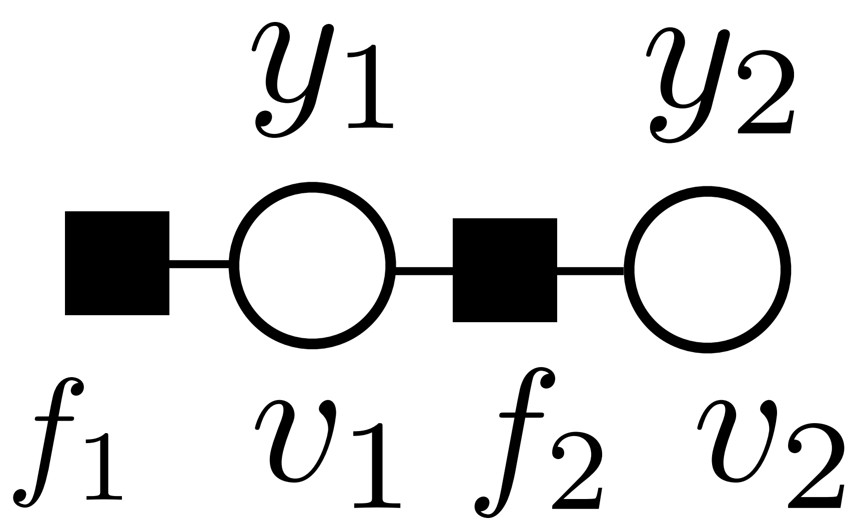
$$\exp(\phi(y_1)) = [0.9, 0.1] \qquad \exp(\phi(y_1, y_2)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

- Initialize messages arbitrarily, then iterate over nodes (in some order):

$$\mu_{v \to f}(y) = \prod_{f' \in N(v), f' \neq f} \mu_{f' \to v}(y)$$

$$\mu_{f \to v_i}(y_i) = \sum_{\mathbf{y}_{f,-i}} \exp(\phi(\mathbf{y}_{f,-i}, y_i)) \prod_{k : v_k \in N(f), k \neq i} \mu_{v_k \to f}(y_{f,k})$$

Sum-Product Algorithm



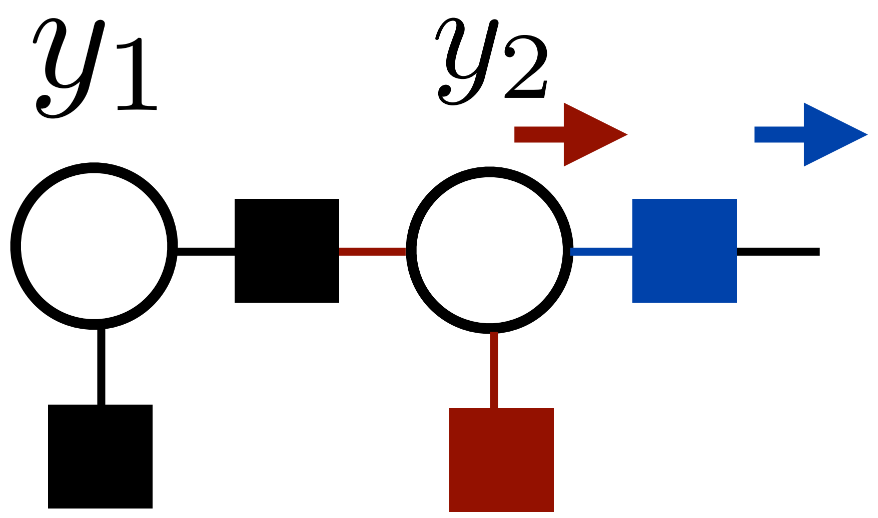
$$P(y_1, y_2|\mathbf{x}) = \frac{\exp(\phi(y_1))\exp(\phi(y_1, y_2))}{\sum_{y_1', y_2'} \exp(\phi(y_1'))\exp(\phi(y_1', y_2'))}$$

- Final marginals: $P(y_i = y|\mathbf{x}) \propto \prod_{f \in N(v_i)} \mu_{f \to v_i}(y)$

- ***If graph is tree structured***: once every node/factor has "talked to" every other node/factor, we have convergence

- For linear chains: need to run a "forward" pass and a "backward" pass

Connections to Forward-Backward



$$\mu_{v \to f}(y) = \prod_{f' \in N(v), f' \neq f} \mu_{f' \to v}(y)$$

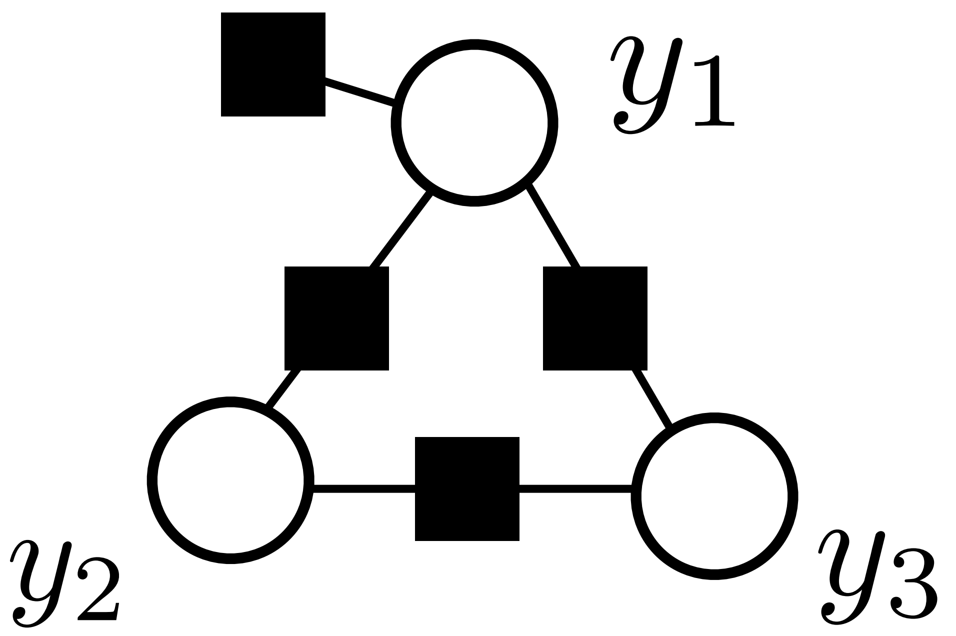
$$\mu_{f \to v_i}(y_i) = \sum_{\mathbf{y}_{f,-i}} \exp(\phi(\mathbf{y}_{f,-i}, y_i)) \prod_{k: v_k \in N(f), k \neq i} \mu_{v_k \to f}(y_{f,k})$$

- Message from variable to “next” factor: product over current emission and previous factor message

- Message from factor to variable: incorporates transition scores

- We’ve just broken the forward update into two pieces!

Loopy Sum-Product



$$\exp(\phi(y_1)) = [0.9, 0.1]$$

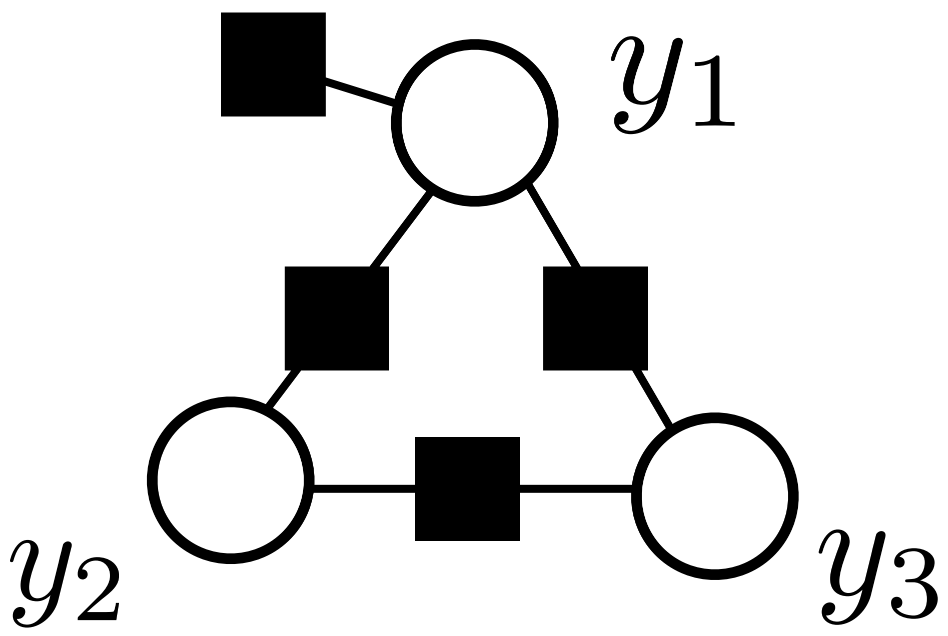
$$\exp(\phi(y_1, y_2)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$$\exp(\phi(y_2, y_3)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$$\exp(\phi(y_3, y_1)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

- What happens in this case? Posteriors blow up!
- Sum-product algorithm is not correct with loops

Belief Propagation Algorithm



$$\exp(\phi(y_1)) = [0.9, 0.1]$$

$$\exp(\phi(y_1, y_2)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

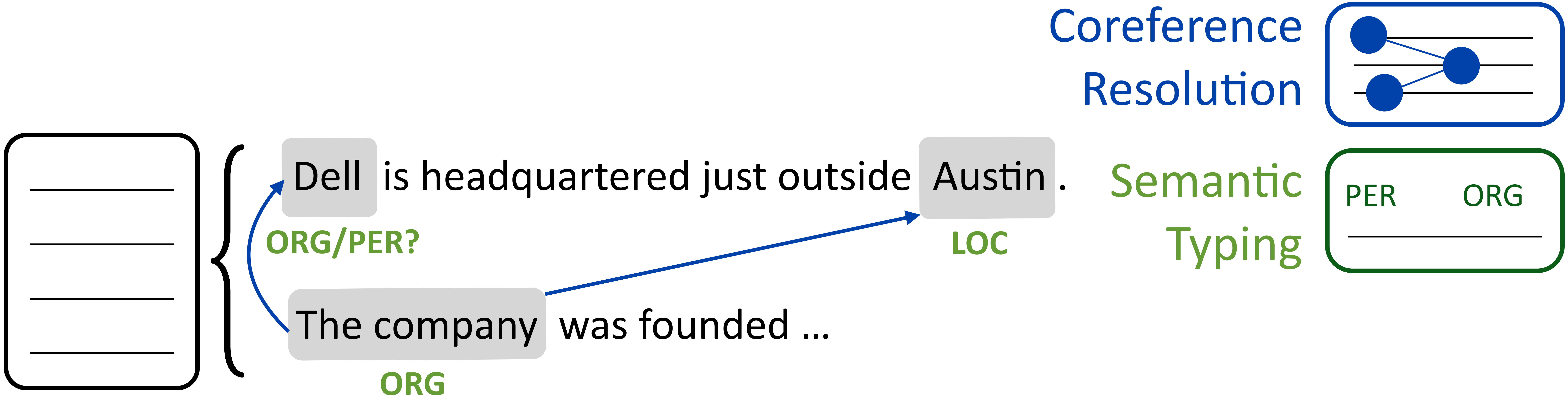
$$\exp(\phi(y_2, y_3)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$$\exp(\phi(y_3, y_1)) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

- **Belief propagation:** ignore this problem. Run sum-product for a while and use what it computes as an *approximation* to the true posterior
  - Most of the time cyclic dependencies are not strong and it works out
  - Some motivation from statistical physics, no guarantees on results

Entity Analysis

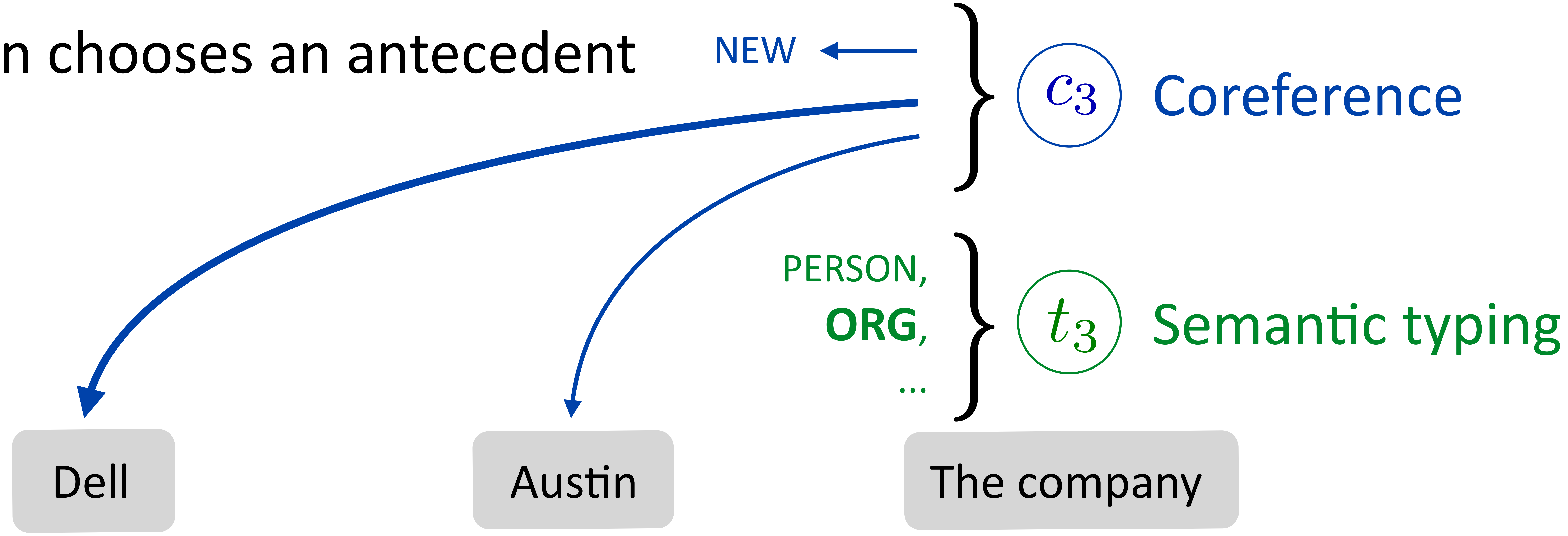
- Model for joint coreference, NER, and entity linking to Wikipedia; here we'll just look at coref+NER



Durrett and Klein (2014)

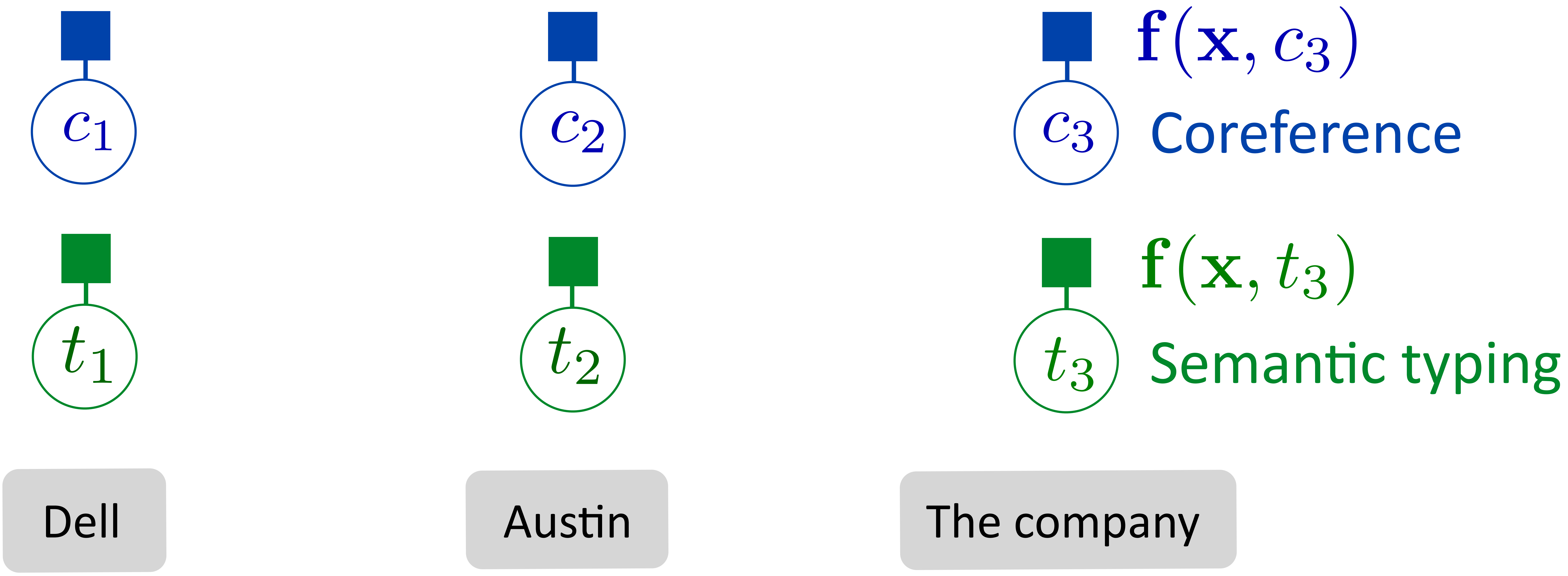
Entity Analysis

Each mention chooses an antecedent



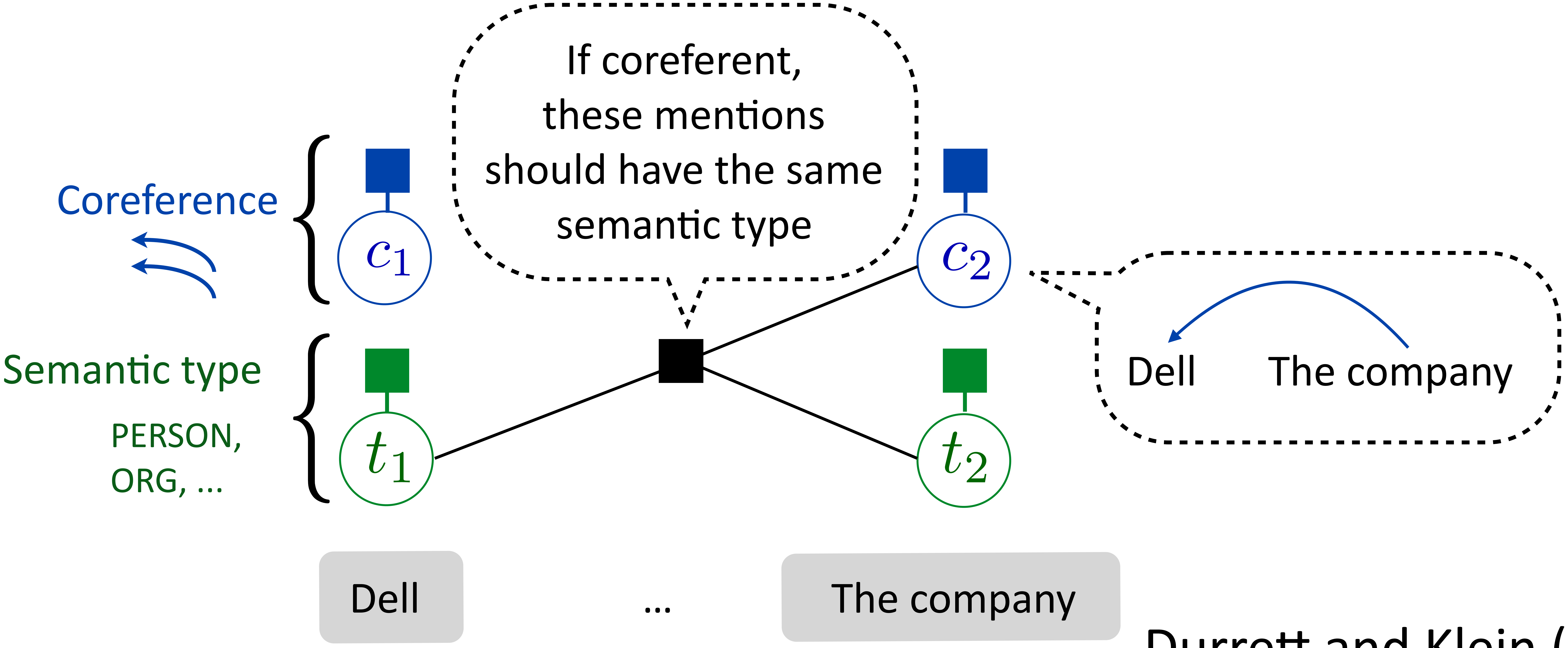
Durrett and Klein (2014)

Entity Analysis



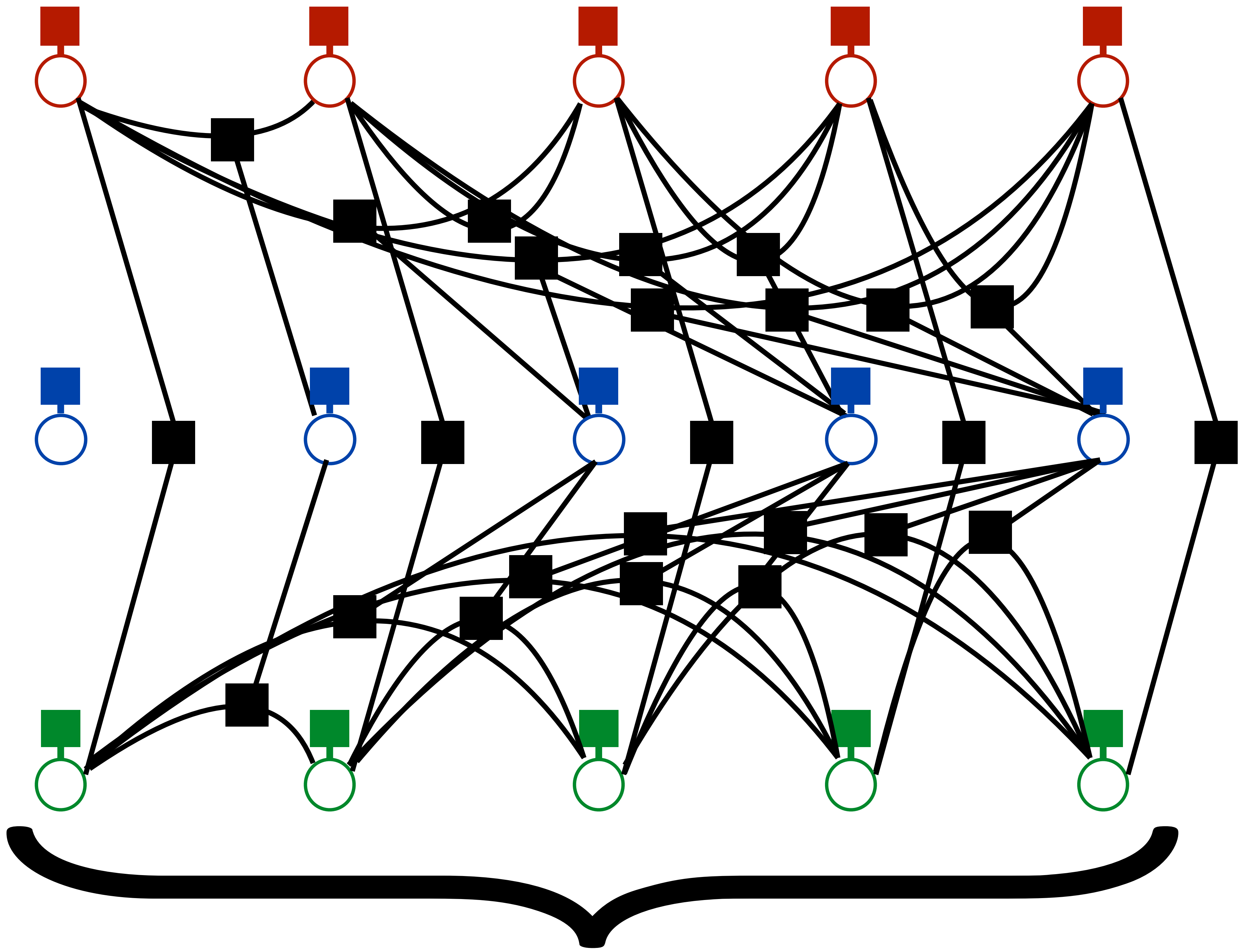
Durrett and Klein (2014)

Entity Analysis



Durrett and Klein (2014)

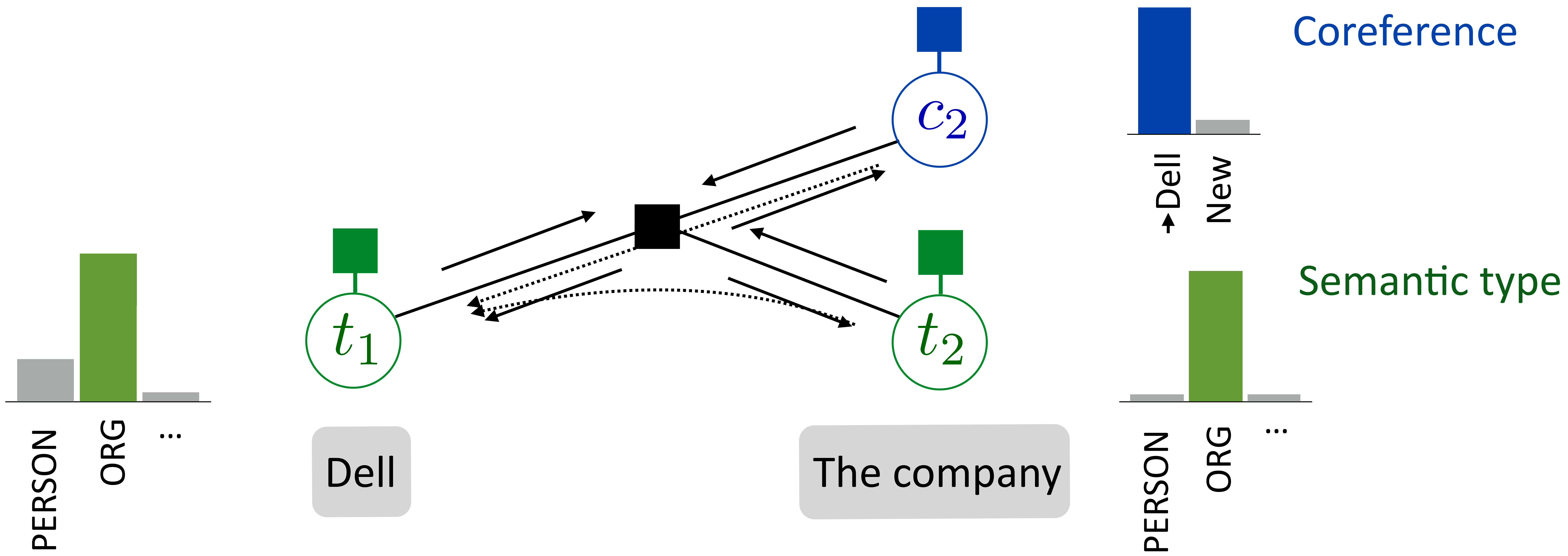
Entity Analysis



5 mentions…and a typical document has 200

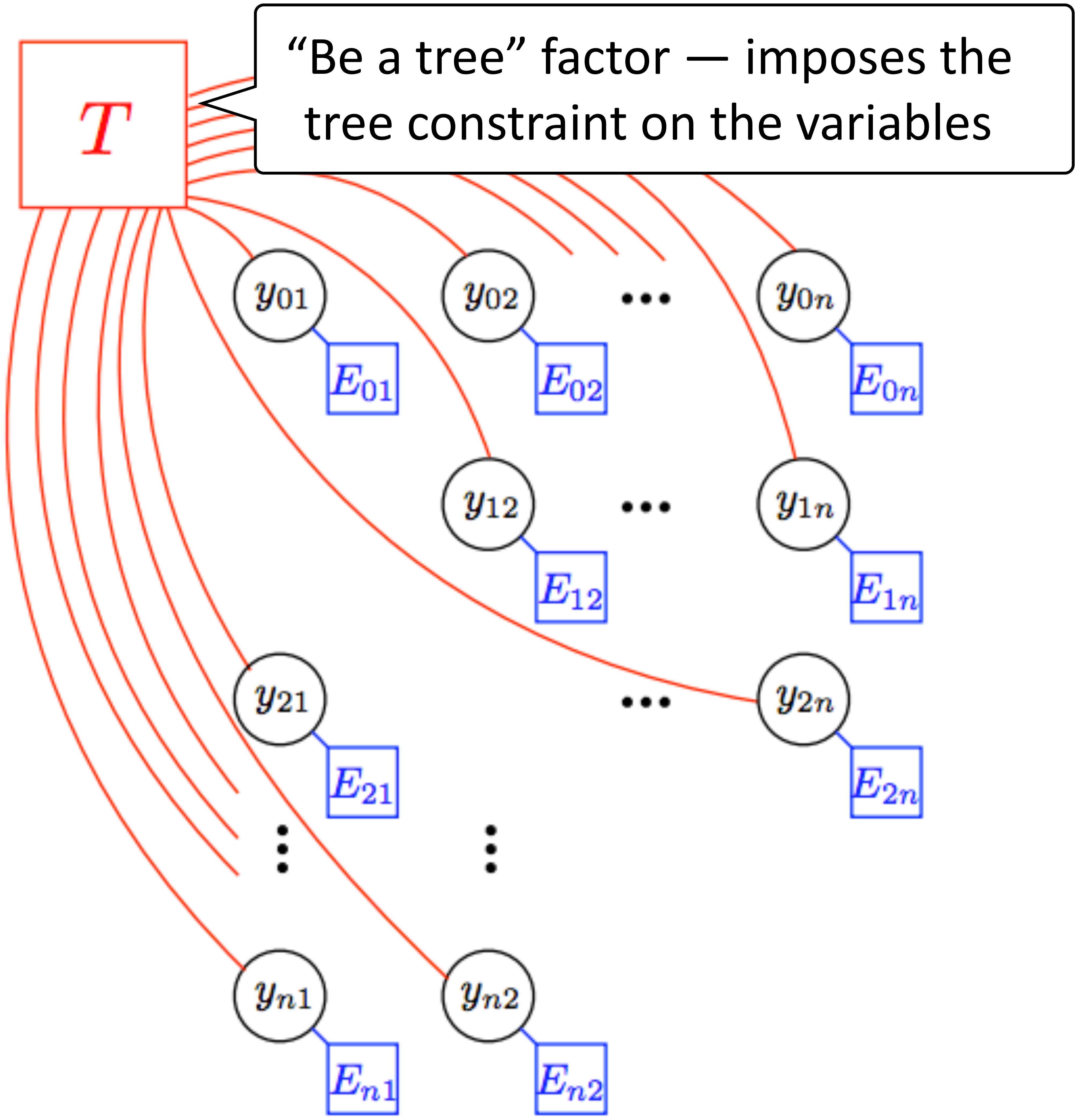
Durrett and Klein (2014)

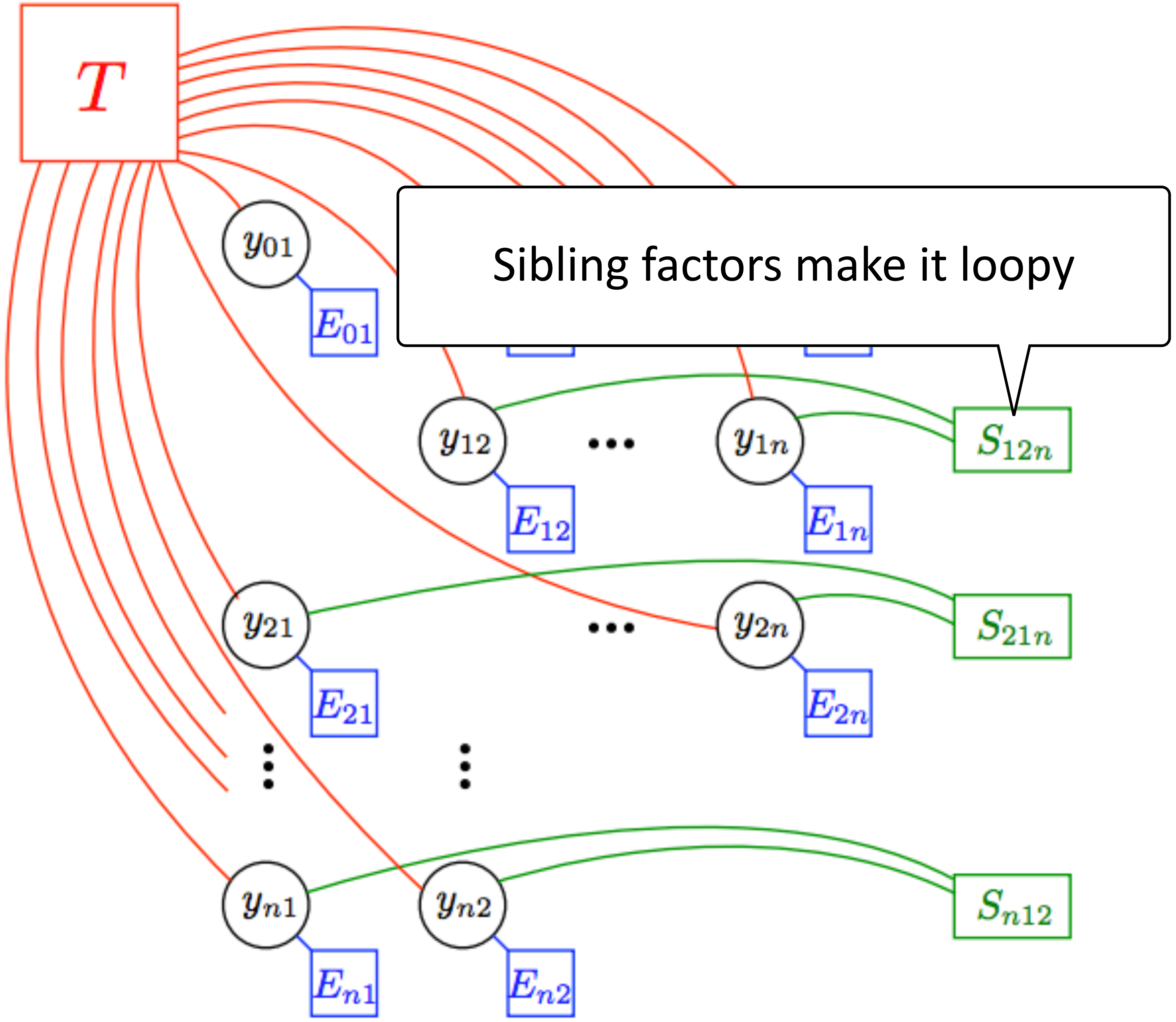
What does BP inference look like?



Durrett and Klein (2014)

Belief Propagation



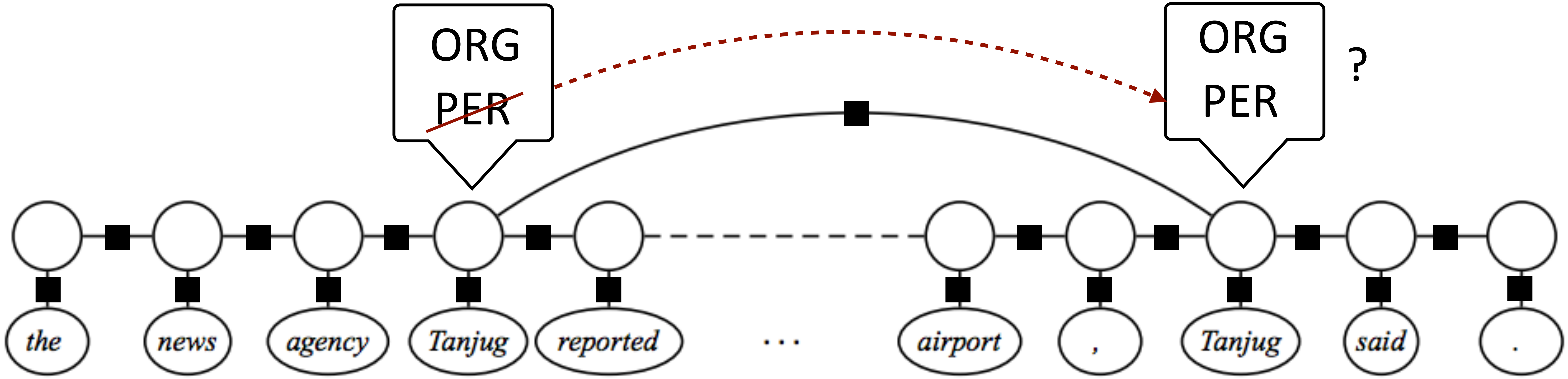


- Achieve the same thing as Koo’s higher-order features

Bansal et al. (2014)

Gibbs Sampling

Skip-chain CRFs



$$P(\mathbf{y}|\mathbf{x}) \propto \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, \mathbf{x})) \boxed{\prod_{(j,k)} \exp(\phi_\ell(y_j, y_k))}$$

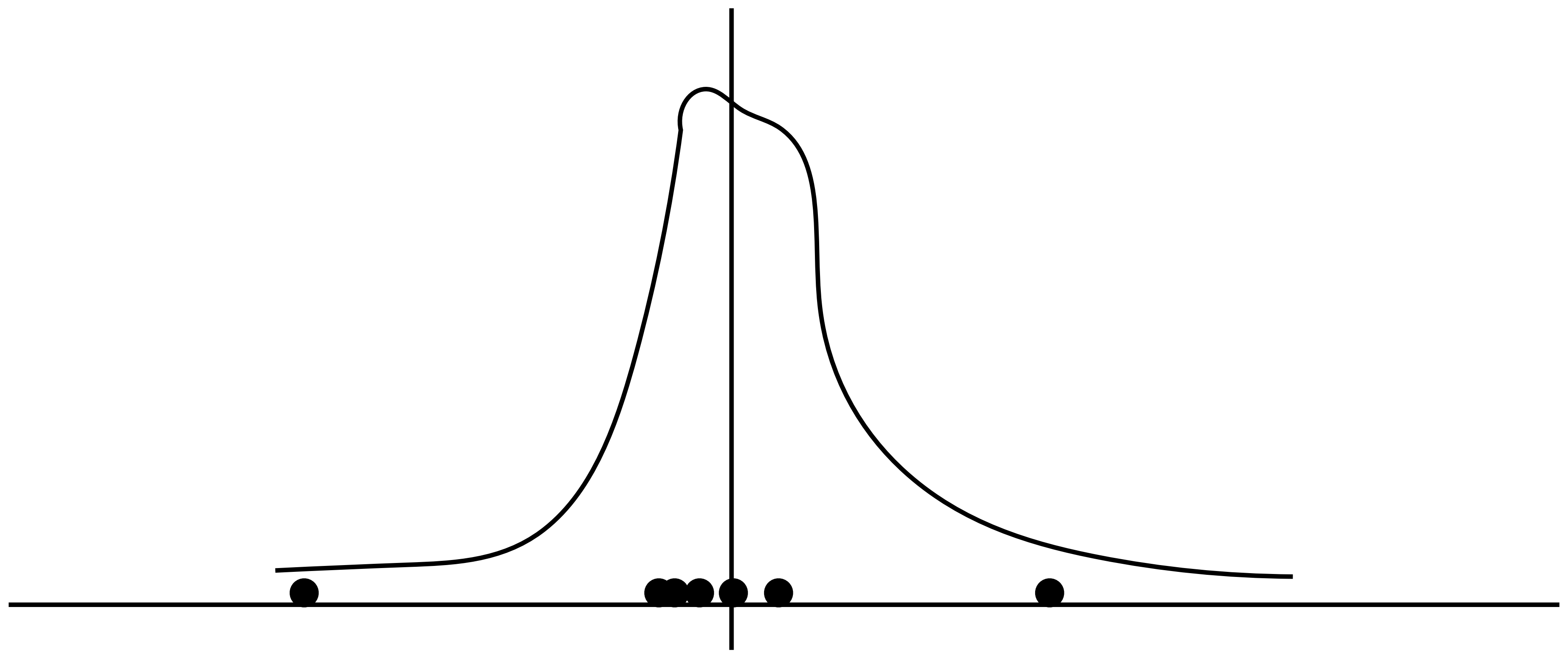
- Can we approximate P(**y**|**x**) in other ways?

Finkel and Manning (2008)

Inference

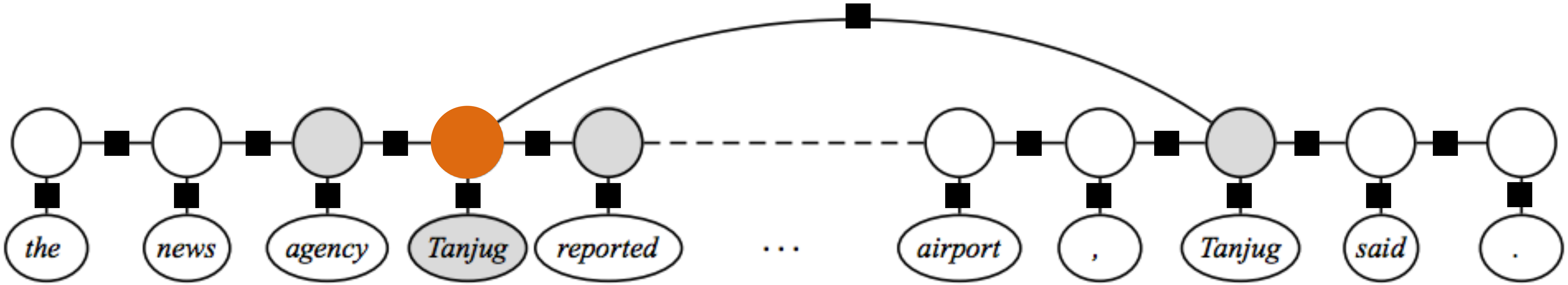
$$P(\mathbf{y}|\mathbf{x}) \propto \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, \mathbf{x})) \boxed{\prod_{(j,k)} \exp(\phi_\ell(y_j, y_k))}$$

- Can we sample from P(**y**|**x**) and use those samples to approximate it? (Monte Carlo methods)
- For distributions that are very peaked, samples look like the max anyway…



Finkel and Manning (2008)

Gibbs Sampling



- Key idea: resample a single variable at a time conditioned on all others

$$P(y_i = y|\mathbf{y}_{-i}, \mathbf{x}) \propto \exp\Big[\phi_t(y_{i-1}, y) + \phi_t(y, y_{i+1}) + \phi_e(y, i, \mathbf{x}) + \sum_{k:(i,k)\ \mathrm{linked}} \phi_\ell(y, y_k) + \sum_{k:(k,i)\ \mathrm{linked}} \phi_\ell(y_k, y)\Big]$$

Finkel and Manning (2008)

Gibbs Sampling

$$P(y_i = y|\mathbf{y}_{-i}, \mathbf{x}) \propto \exp\Big[\phi_t(\boxed{y_{i-1}}, y) + \phi_t(y, \boxed{y_{i+1}}) + \phi_e(y, i, \mathbf{x}) + \sum_{k:(i,k)\ \text{linked}} \phi_\ell(y, \boxed{y_k}) + \sum_{k:(k,i)\ \text{linked}} \phi_\ell(\boxed{y_k}, y)\Big]$$

- Orange things are all constants now!
- Fix all predictions except one, easy to compute conditional probabilities (normalize scores for this particular variable y)
- Iterate over all variables repeatedly, like belief propagation

Finkel and Manning (2008)

Gibbs Sampling

Initialize y values to something reasonable

for k=1…t iterations:

  for i=1…m words in the document:

    y_i = Sample from $P(y_i | \mathbf{y}_{1,\ldots,i-1,i+1,\ldots,m}, \mathbf{x})$

- Note: we need to iterate over the document several times!
- The Gibbs sampling procedure forms a Markov chain whose equilibrium distribution is the posterior
- However, you might need to run it for a very long time to get samples which don't depend on the initialization….

Problems with Gibbs Sampling

$P(x_1, x_2)$

		x_2: 0	x_2: 1
x_1	0	0.49	0.01
	1	0.01	0.49

- Start with x = (0, 0)

$P(x_2|x_1 = 0) = [0.98, 0.02]$
- stay at (0, 0) 98% of the time

$P(x_1|x_2 = 0) = [0.98, 0.02]$
- stay at (0, 0) 98% of the time

- Takes ~50 steps before we switch to (1, 1) — need to run Gibbs sampling for a long time to get a good approximation of the posterior

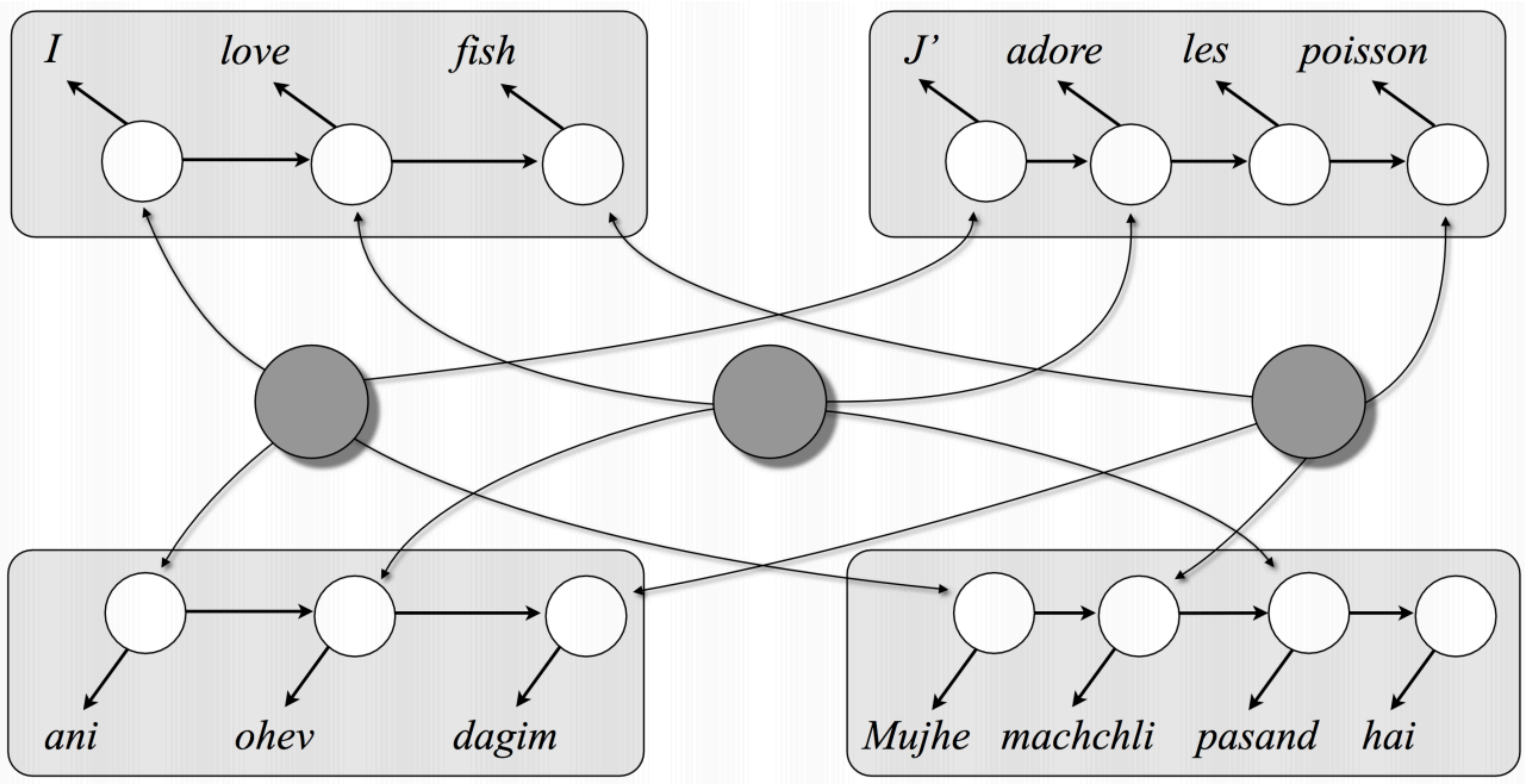
Gibbs Sampling

CoNLL					
Approach	LOC	ORG	MISC	PER	ALL
B&M LT-RMN	–	–	–	–	80.09
B&M GLT-RMN	–	–	–	–	82.30
Local+Viterbi	88.16	80.83	78.51	90.36	85.51
NonLoc+Gibbs	88.51	81.72	80.43	92.29	86.86

Finkel and Manning (2008)

Gibbs Sampling

- Unsupervised POS induction with alignments across languages



Naseem et al. (2009)

Takeaways

- Can define "loopy" factor graphs and still do inference
- Belief propagation and Gibbs sampling both work best if there are only weak cyclic dependencies. This is usually the case if the loopy factors incorporate features and the loops are large
- Can incorporate nice features this way, not as commonplace and a bit harder to get working, but everyone thinks this stuff is cool
  - Other ways of doing this: output reranking, beam search (give up on doing principled inference), …