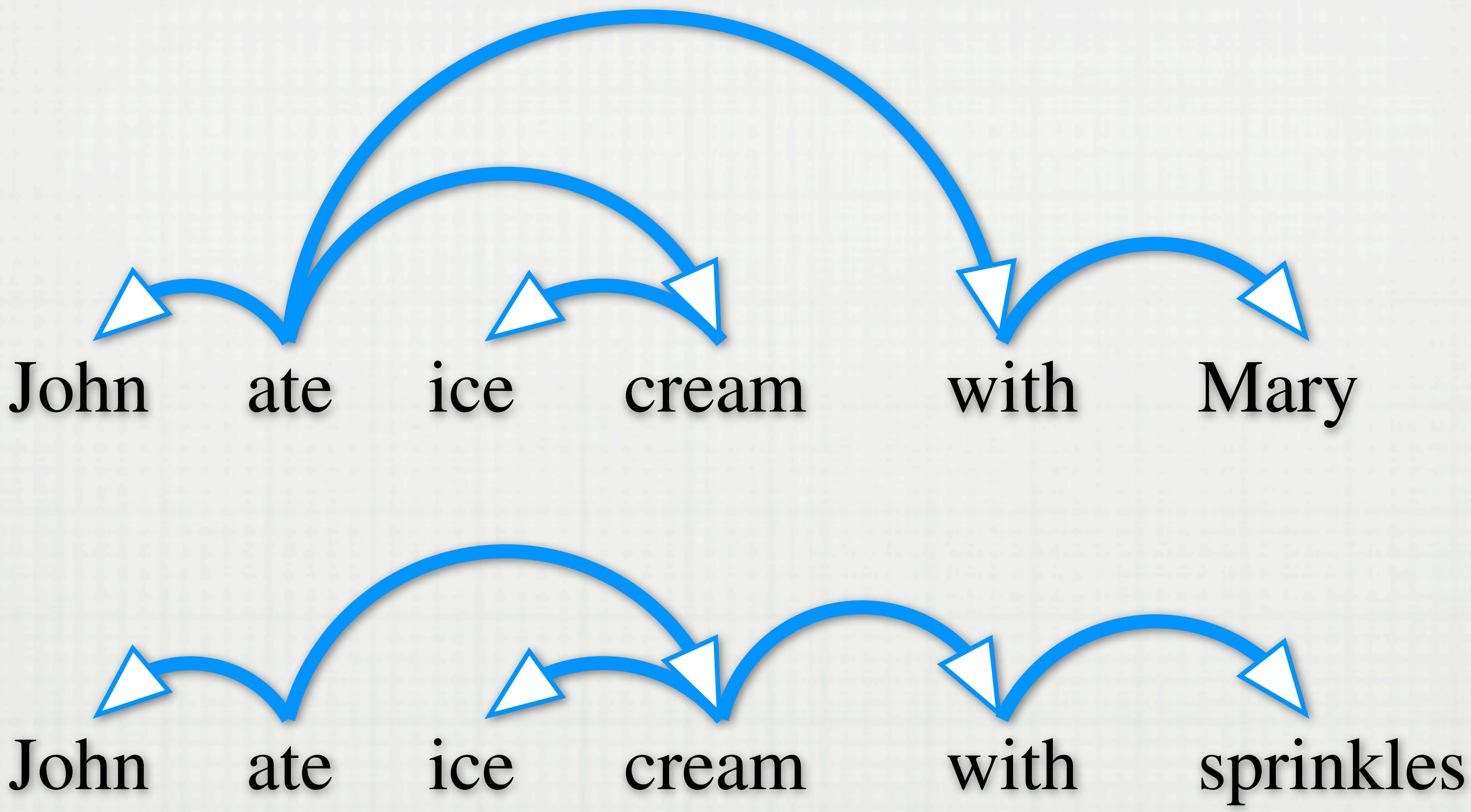
Advances in Discriminative Dependency Parsing

Terry Koo

Dependency Grammar



- Syntax represented by head-modifier dependencies
- Applications include machine translation, semantic role labeling, etc.

Discriminative Parsing

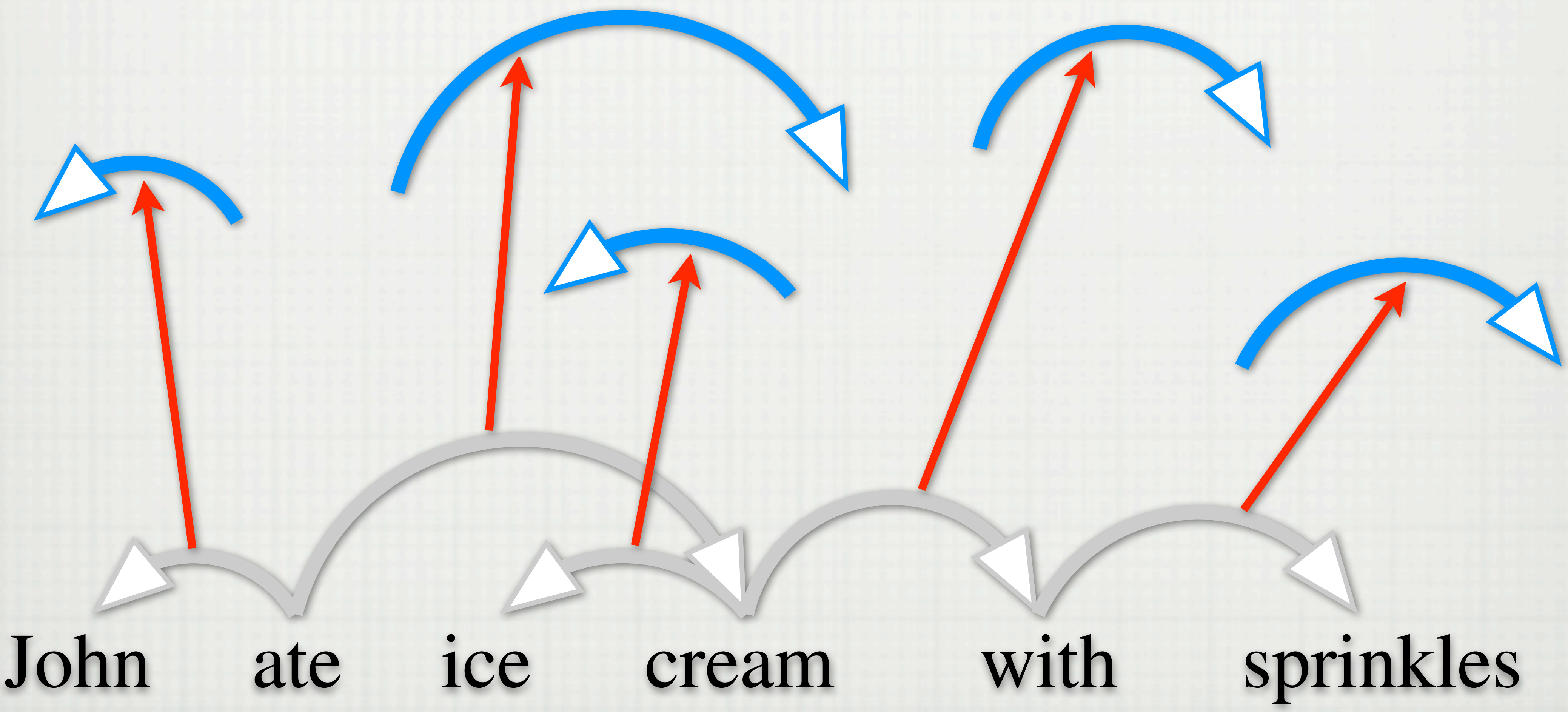
- *Structured linear model* for parsing (McDonald, 2005):

$$y^\star(\mathbf{x}; \mathbf{w}) = \operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y)$$

- $\mathbf{x}$ is a sentence, $\mathcal{Y}(\mathbf{x})$ is the set of possible trees
- Structures represented via feature mapping $\mathbf{\Phi}(\mathbf{x}, y)$
- Parameters $\mathbf{w}$ provide a weight for each feature
- Direct maximization is generally *intractable*

Factorization of Structures

- Assume trees can be *factored* into *parts*



- Feature decomposition: $\mathbf{\Phi}(\mathbf{x}, y) = \sum_{p \in y} \phi(\mathbf{x}, p)$

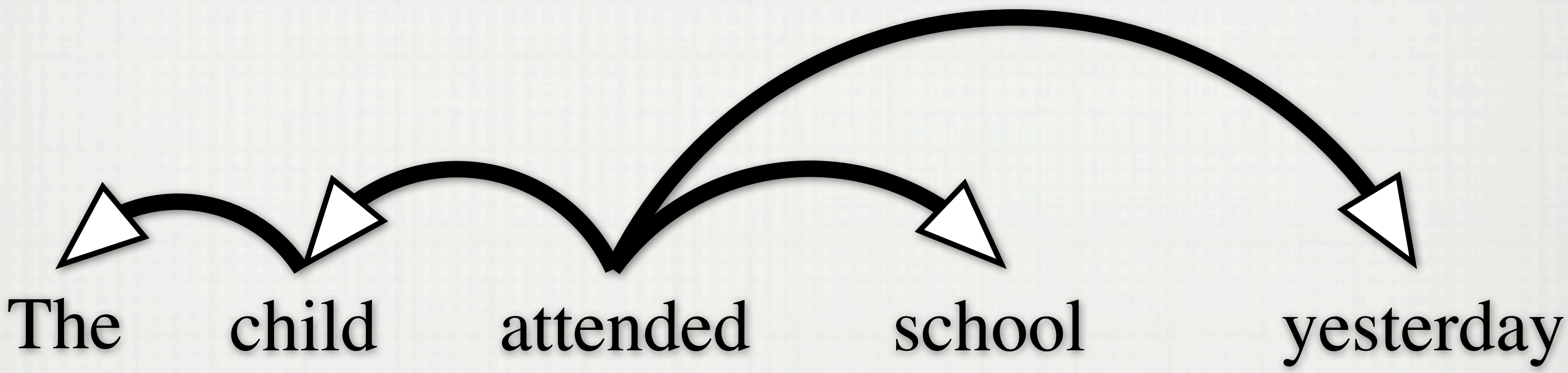
Factored Discriminative Parsing

- Factored structured linear model:

$$y^\star(\mathbf{x};\mathbf{w}) \;=\; \operatorname*{argmax}_{y\in\mathcal{Y}(\mathbf{x})} \sum_{p\in y} \mathbf{w}\cdot\boldsymbol{\phi}(\mathbf{x},p)$$

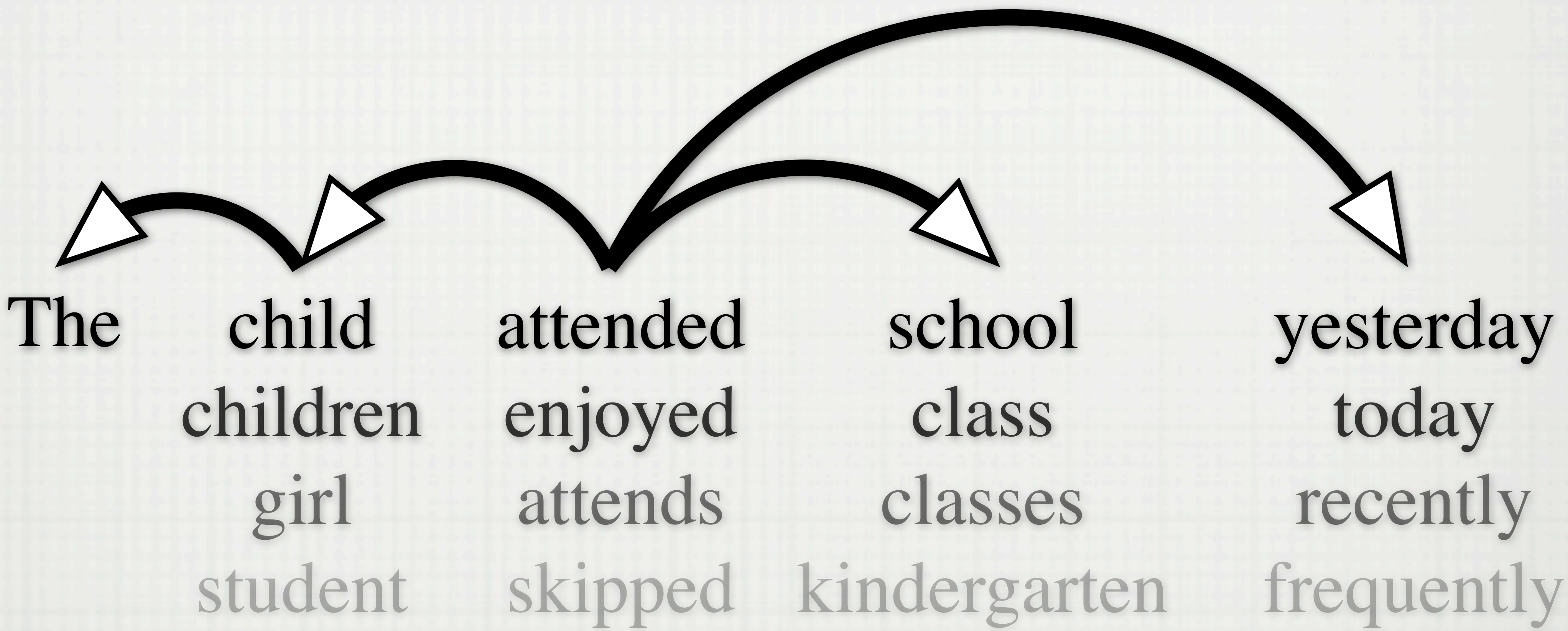
- Tractable for many factorizations
- Three main components:
  - *Feature mapping*: what features appear in ϕ?
  - *Parameters*: how do we estimate $\mathbf{w}$?
  - *Factorization*: which parts p make up y?

Lexicalized Representations



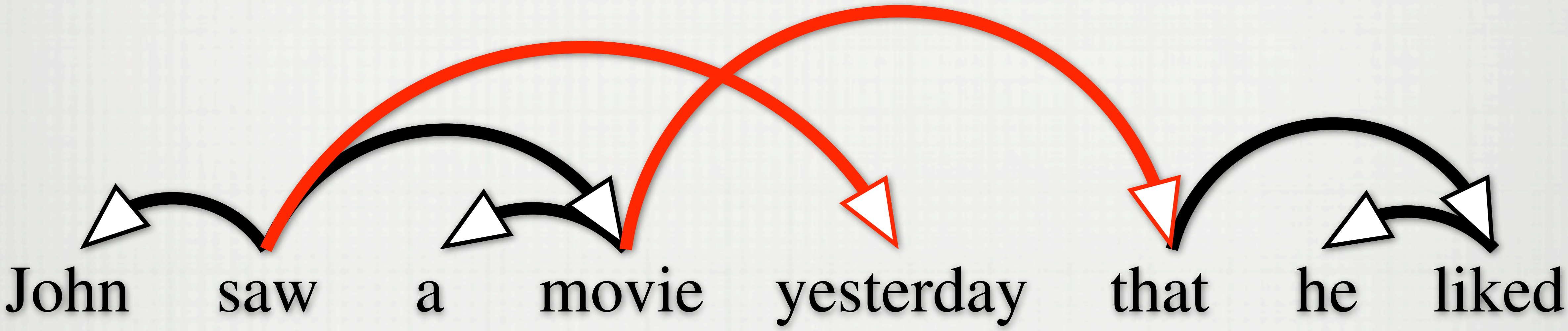
- Statistical parsers make heavy use of lexicalized features
- Alternate lexical representations?

Lexicalized Representations



- Statistical parsers make heavy use of lexicalized features
- Alternate lexical representations?

Non-Projective Parsing

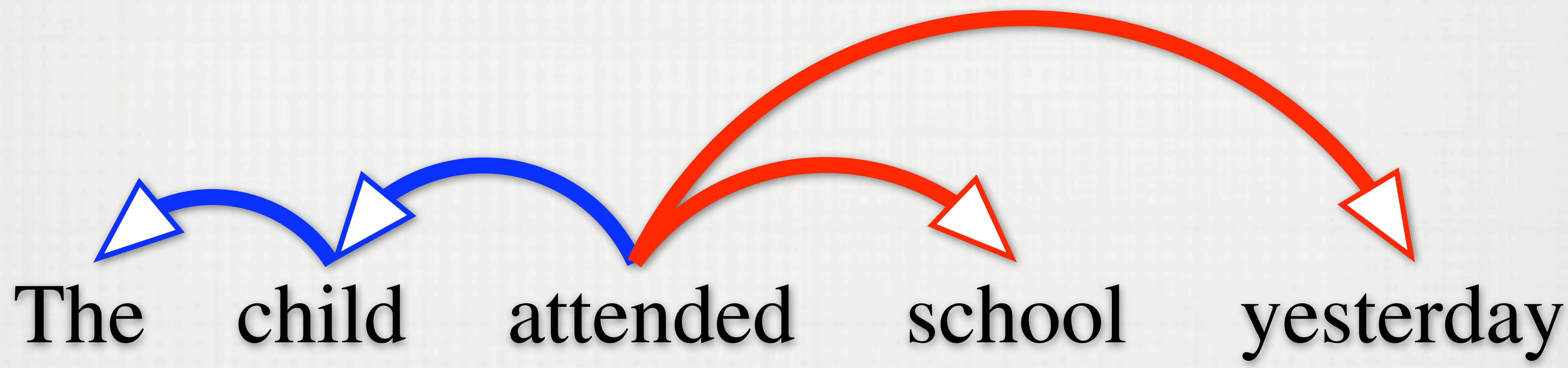


- Non-projective parsing allows *crossing dependencies*
- Frequent in languages like Czech, Dutch, etc.
- Non-projective parsing = maximum spanning-tree (McDonald et al., 2005)

Non-Projective Parsing

- Many parameter estimation methods depend on summations over $\mathcal{Y}(\mathbf{x})$
  - Baum-Welch algorithm for Hidden Markov Models
  - Conditional Random Fields
- Efficient algorithms exist for many types of structure
- Algorithms for non-projective trees?

Higher-Order Factorizations



- First-order factorization: individual dependencies
- Second-order factorization: pairs of dependencies
  - *Sibling* and *Grandchild* interactions
- Factorizations with larger sub-structures?

Outline

- Introduction
- Three advances in discriminative dependency parsing:

$$\underset{y \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \sum_{p \in y} \mathbf{w} \cdot \phi(\mathbf{x}, p)$$

  - *Simple and effective lexical representations*
  - Parameter estimation for non-projective parsing
  - Efficient third-order dependency parsers
- Conclusion

Dependency Parsing Features

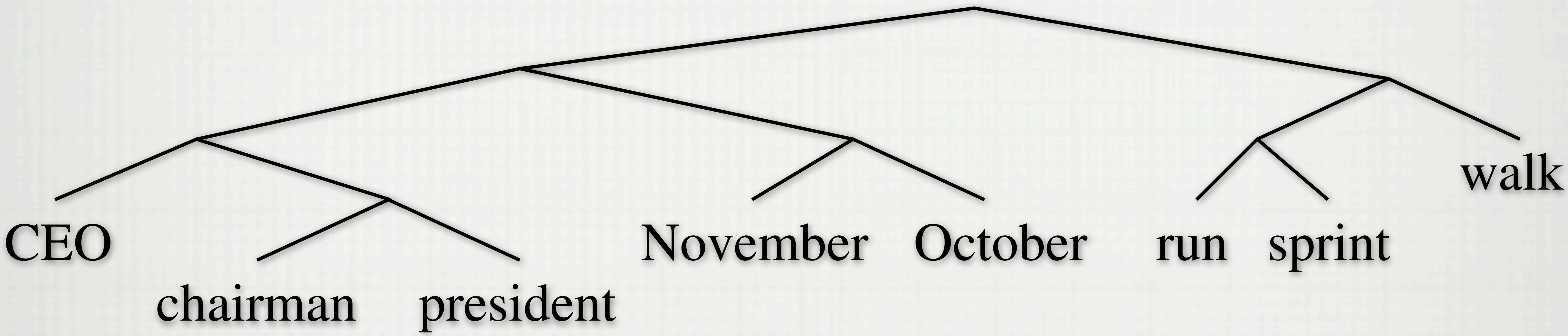
1	2	3	4	5	6
NNP	VBD	NN	NN	IN	NNS
John	ate	ice	cream	with	sprinkles

- A dependency is a pair (h, m), e.g., $(2, 5)$
- Features are $0/1$ indicators for words, parts of speech
  - $\phi_1(\mathbf{x}, h, m) = [\![$ "ate" $\longrightarrow$ "with" $]\!]$
  - $\phi_2(\mathbf{x}, h, m) = [\![$ "VBD" $\longrightarrow$ "IN" $]\!]$

Alternate Lexical Representations

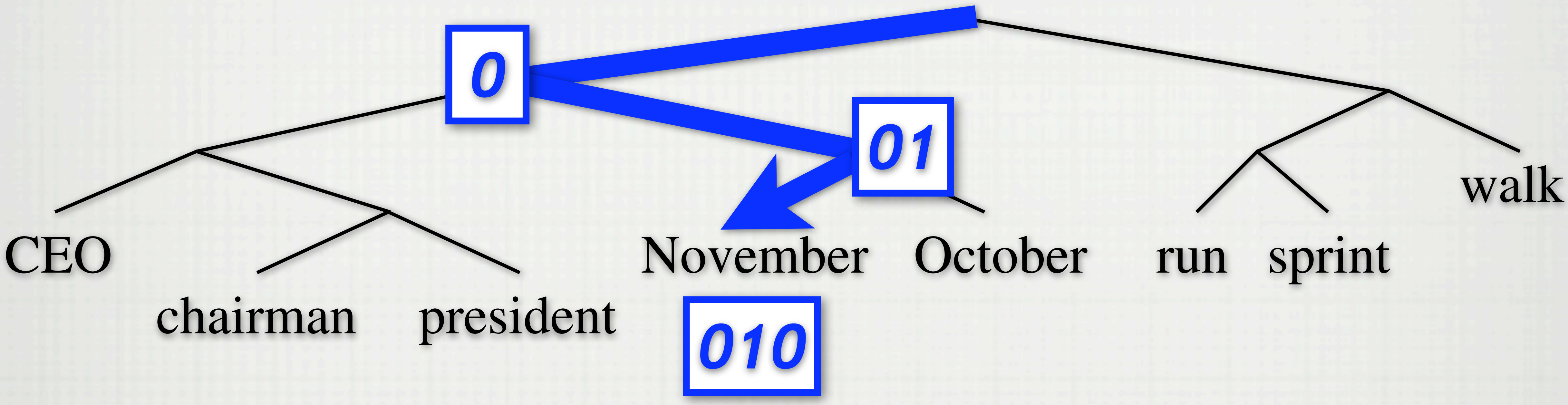
- Intermediate lexical representations derived from unlabeled data: *word clusters*
- Clusters are easily incorporated as features
- Improvements in English and Czech parsing
- Previous work:
  - Brown et al. (1992): clustering algorithm, applied to language modeling
  - Miller et al. (2004): named-entity tagging with word clusters from the Brown algorithm

Brown Algorithm



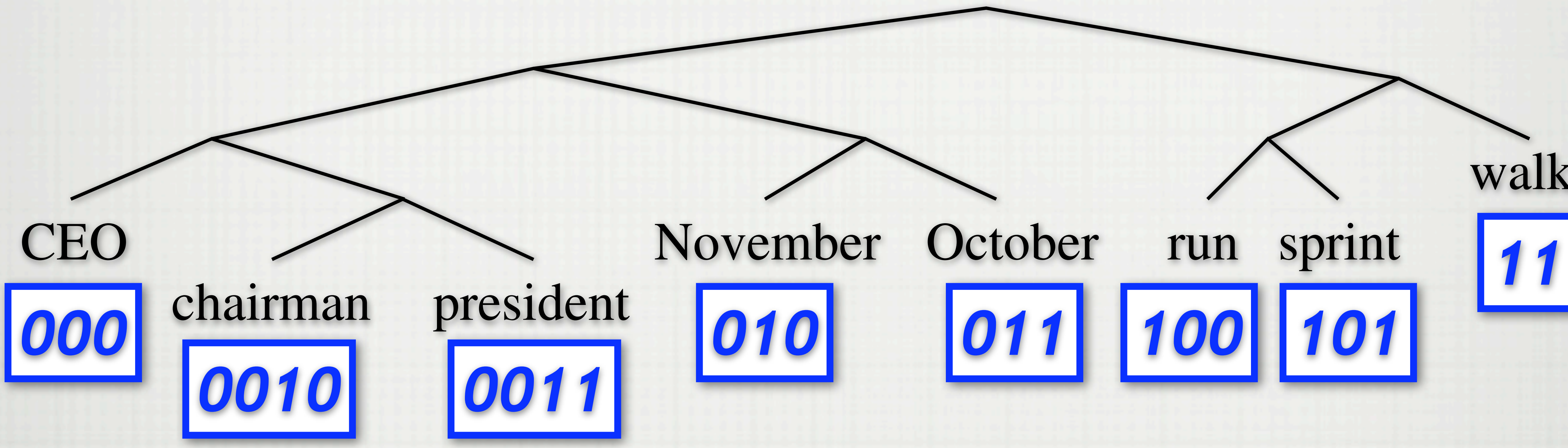
- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
  - Prefixes of bit strings yield clusterings
  - Prefix length determines granularity

Brown Algorithm



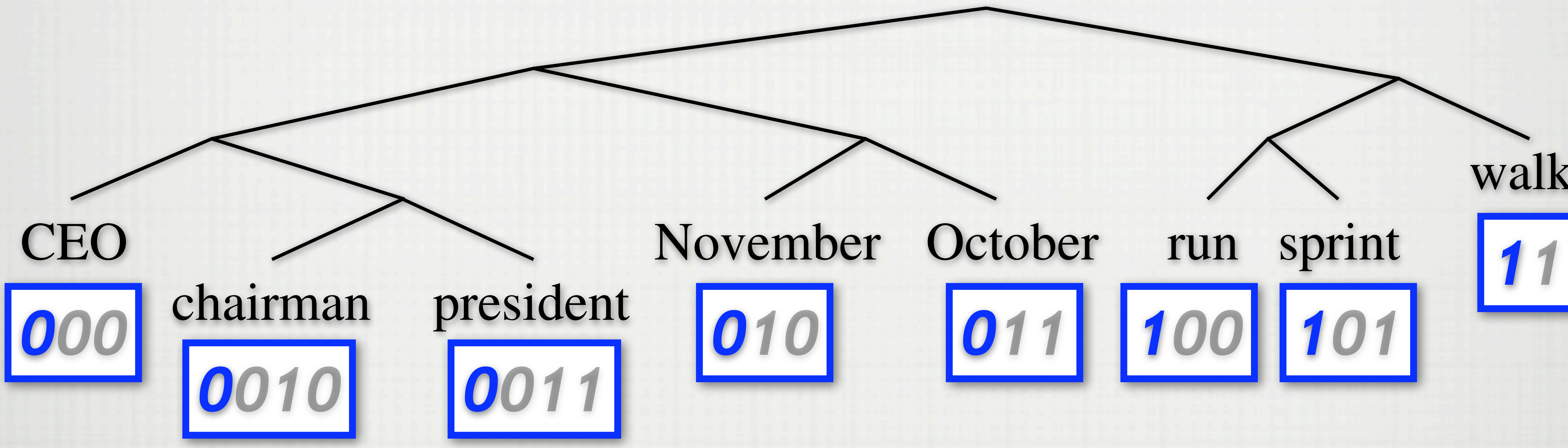
- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
  - Prefixes of bit strings yield clusterings
  - Prefix length determines granularity

Brown Algorithm



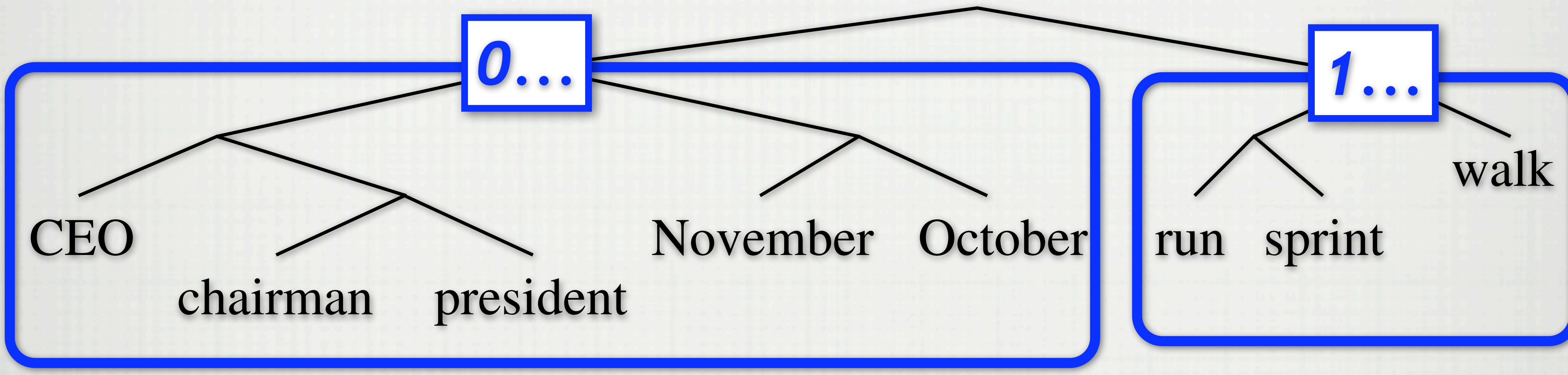
- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
    - Prefixes of bit strings yield clusterings
    - Prefix length determines granularity

Brown Algorithm



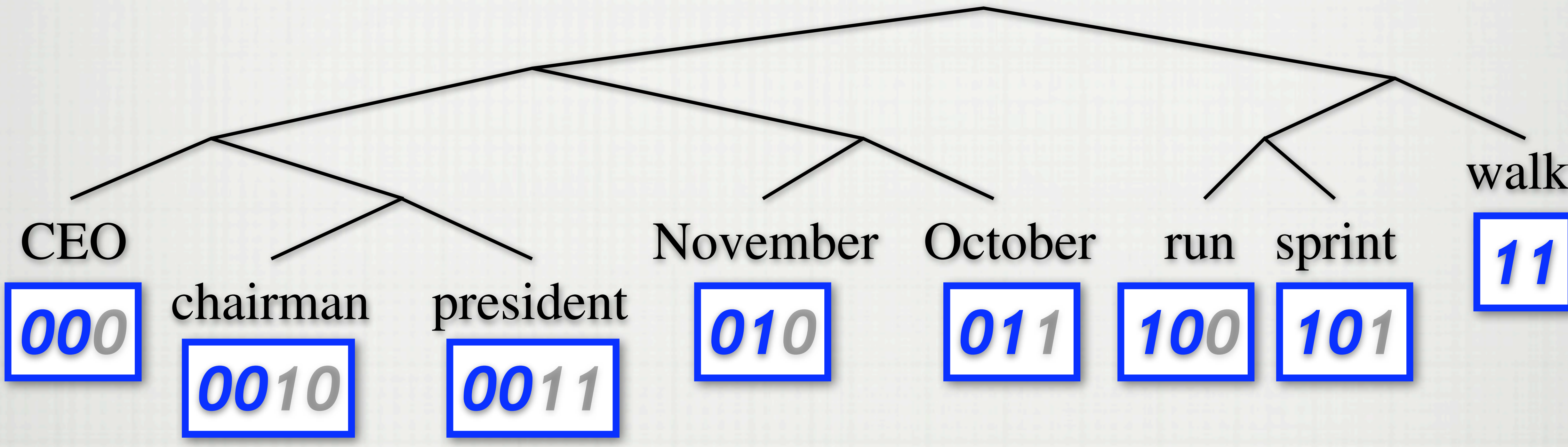
- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
  - Prefixes of bit strings yield clusterings
  - Prefix length determines granularity

Brown Algorithm



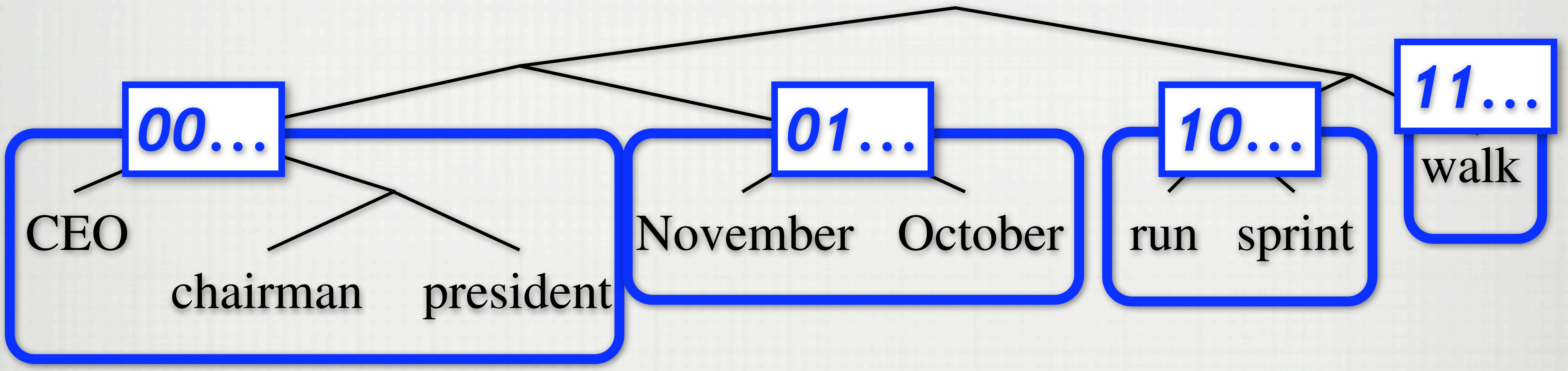
- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
    - Prefixes of bit strings yield clusterings
    - Prefix length determines granularity

Brown Algorithm



- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
  - Prefixes of bit strings yield clusterings
  - Prefix length determines granularity

Brown Algorithm



- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
  - Prefixes of bit strings yield clusterings
  - Prefix length determines granularity

Brown Algorithm

- Examples of clusters from our English experiments

0101010101110011 constructed
0101010101110011 elucidated
0101010101110011 inhaled
0101010101110011 rewritten

Past Participle Verbs

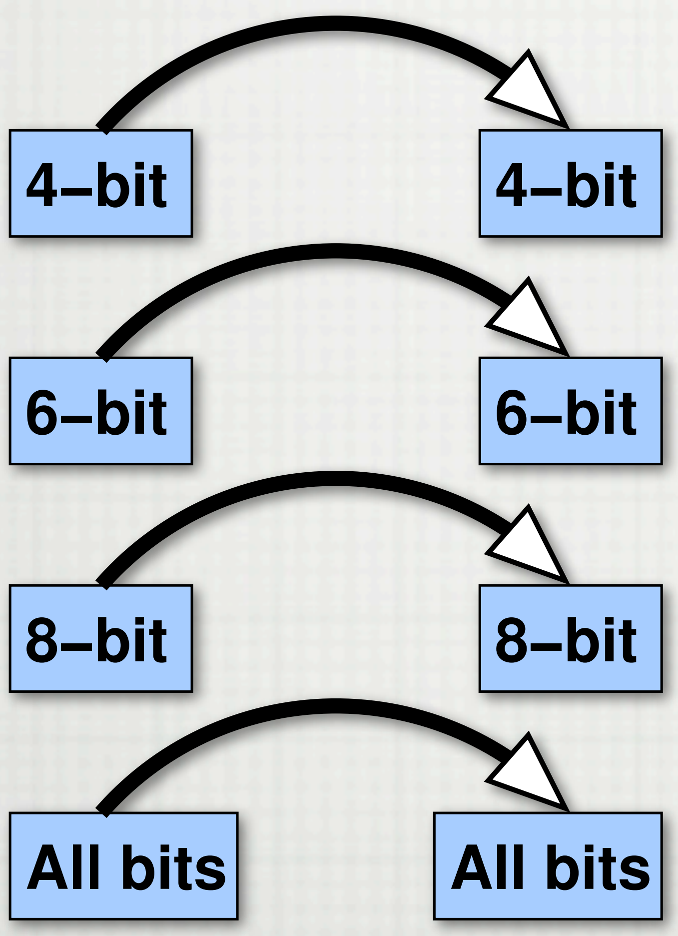
100111111111100 precious-metal
100111111111100 grain-futures
100111111111100 crude-oil-futures

Financial Categories

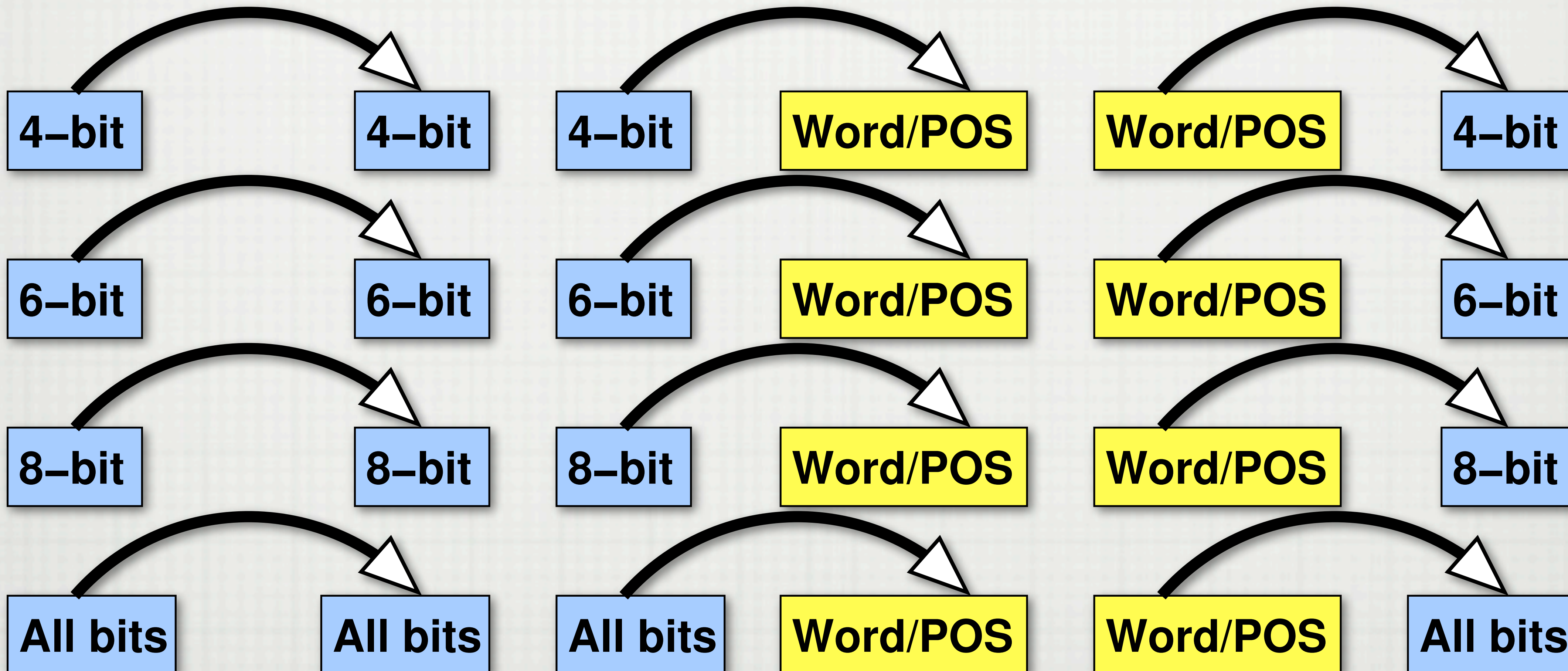
Cluster-based Features

- Feature mappings can include arbitrary information
- Two types of features:
  - *Baseline* features include words and POS
  - *Cluster-based* feature sets add information from clusters

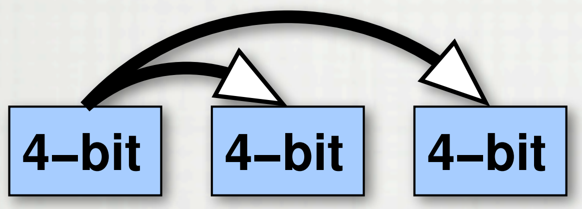
Cluster-based Features

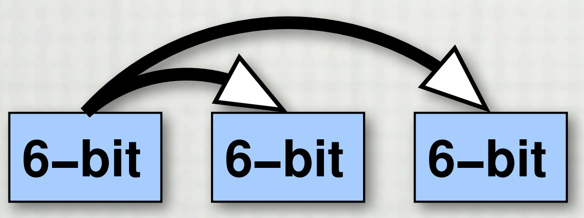


Cluster-based Features

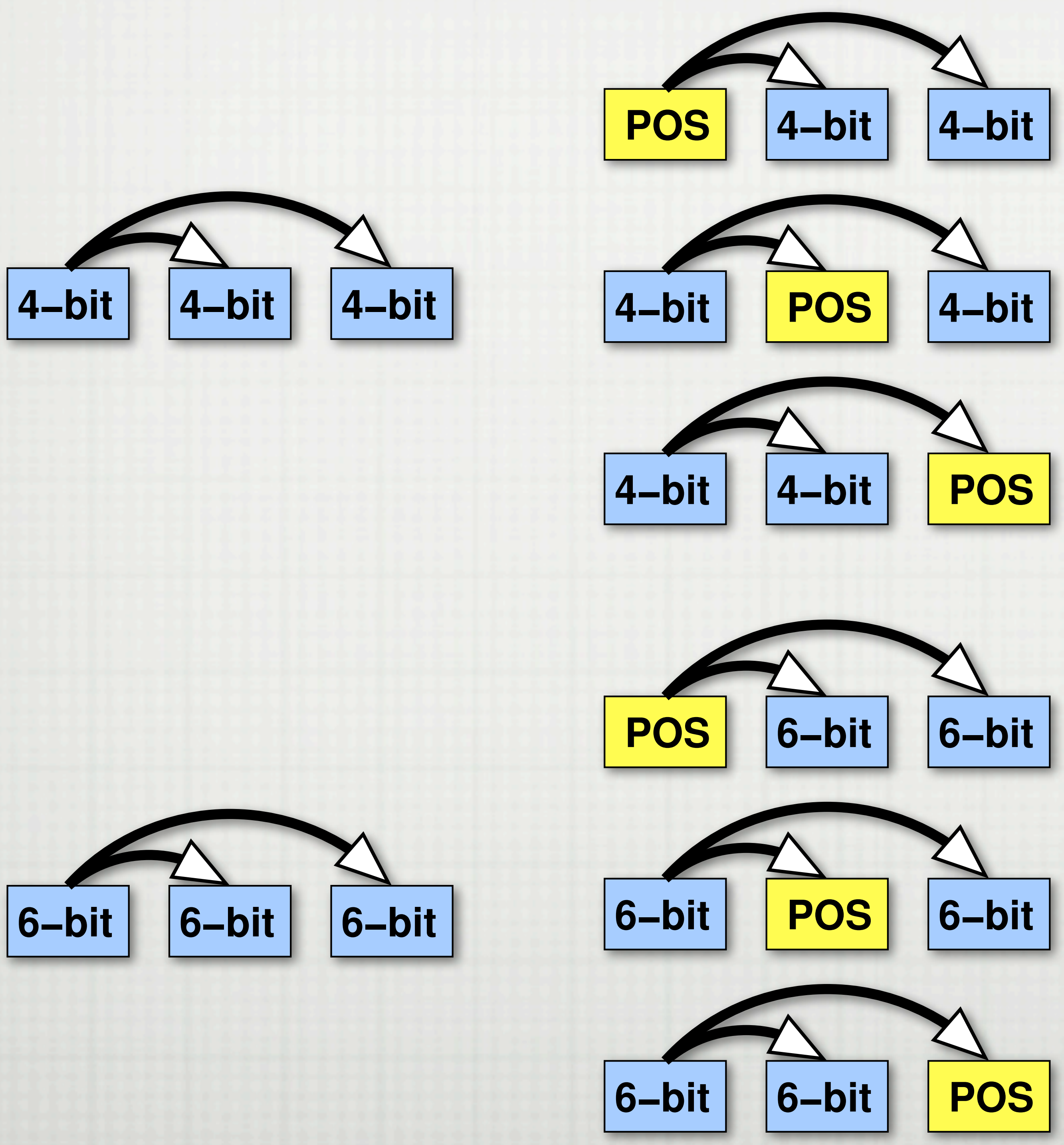


Cluster-based Features

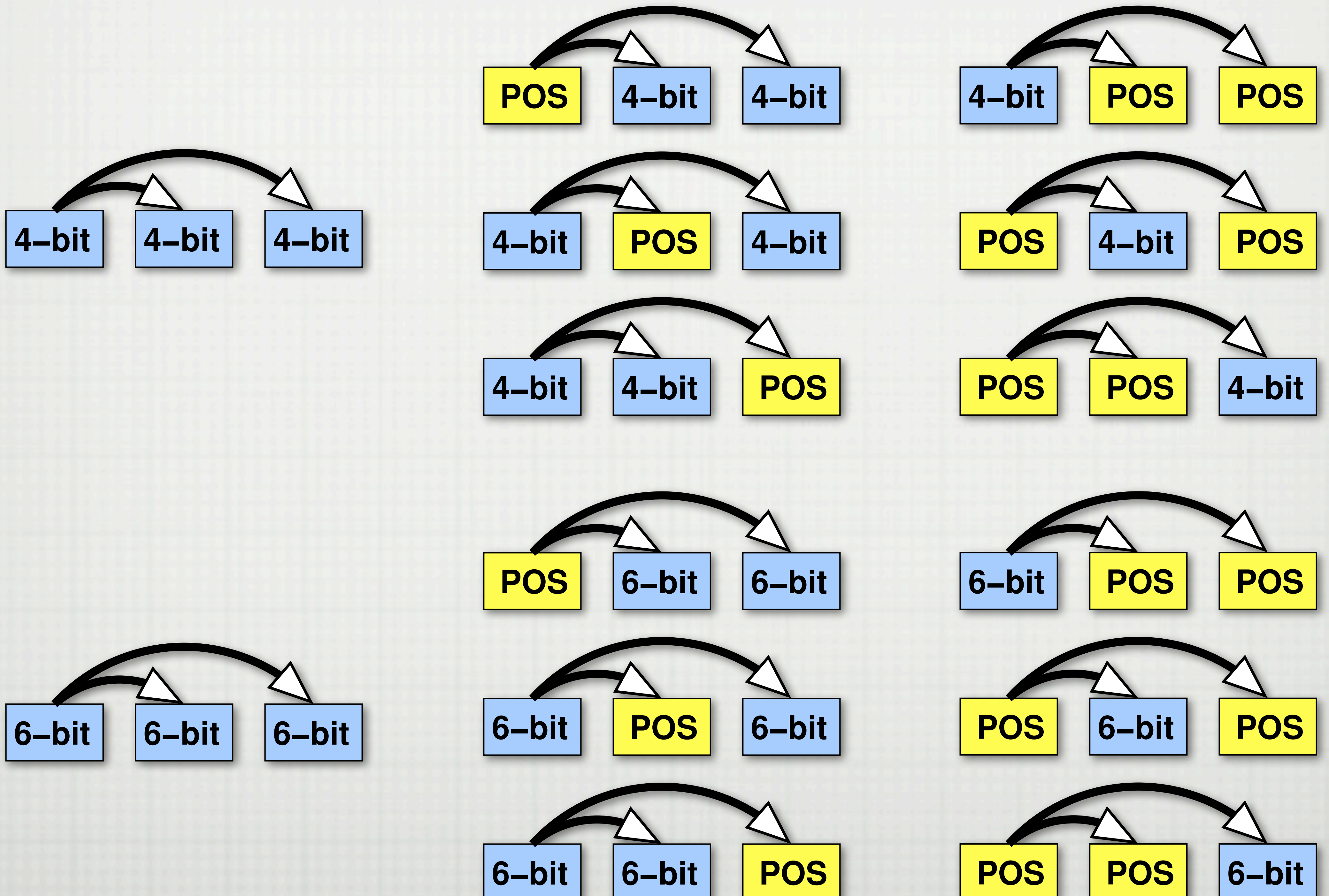




Cluster-based Features



Cluster-based Features



Feature Pruning

- Cluster-based feature sets were very large
- Eliminate features using word frequency
  - Only use the top-800 most frequent words
  - Cluster-based features were not affected

Experiments

- English parsing (Penn Treebank)
- Czech parsing (Prague Dependency Treebank)
- Brown clustering algorithm (Liang, 2005)
- Averaged perceptron training
    - Second-order projective parsers (Carreras, 2007)
    - First-order max-spanning-tree (McDonald, 2005)
- Compare between baseline and cluster-based features

Baseline Comparison

Parsing Model	*Accuracy*
McDonald (2006)	91.5
Baseline Features	92.0

- *Attachment score* on English test set
  - Percent of words attached to correct head
- Baseline parser is state of the art

English Parsing Results

Test Set	Baseline	Cluster-Based
Sec 00	91.8	92.8 (+1.0)
Sec 01	92.5	93.3 (+0.8)
Sec 23	92.0	93.2 (+1.2)
Sec 24	90.9	91.9 (+1.0)

- Attachment score on all English test sets (Sections 00, 01, 23, and 24 of the Penn Treebank)
- Cluster-based features outperform baseline

Effect of Training Corpus Size

#Sentences	Baseline	Cluster-Based
1000	82.0	85.3 (+3.3)
2000	85.0	87.5 (+2.5)
4000	87.9	89.7 (+1.8)
8000	89.7	91.4 (+1.7)
16000	91.1	92.2 (+1.1)
32000	92.1	93.2 (+1.1)
39832	92.4	93.3 (+0.9)

- Attachment score on English development set
- Part-of-speech tagger trained on reduced dataset

Effect of Training Corpus Size

#Sentences	Baseline	Cluster-Based
1000	82.0	85.3 (+3.3)
2000	85.0	87.5 (+2.5)
4000	87.9	89.7 (+1.8)
8000	89.7	91.4 (+1.7)
16000	91.1	92.2 (+1.1)
32000	92.1	93.2 (+1.1)
39832	92.4	93.3 (+0.9)

- Attachment score on English development set
- Part-of-speech tagger trained on reduced dataset

Czech Parsing Results

Parsing Model	*Baseline*	*Cluster-Based*
First-order MST	84.5	86.1 (+1.6)
McDonald (2006) second-order	85.2	—
Second-order	86.1	87.1 (+1.0)

- Attachment score on Czech test set
- Results are similar to English

Removal of Direct Lexicalization

Threshold	*Baseline*	*Cluster-based*
100	90.6 (-1.8)	93.1 (-0.2)
800	91.9 (-0.5)	93.3
All words	92.4	—

- Attachment score on English development set
- Cluster-based features are far less sensitive

Clusters vs Part-of-Speech Tags

	Ignore POS Tags	Use POS Tags
Ignore Clusters	86.7	92.4
Use Clusters	91.8	93.3

- Attachment score on English development set
- Clusters *alone* are almost as good as baseline

Summary

- Lexical statistics are important but sparse
- Word clusters as an alternate lexical representation
- Clusters incorporated as features for a discriminative parser
- Performance gains over a state-of-the-art baseline

Outline

- Introduction
- Three advances in discriminative dependency parsing:

$$\operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \sum_{p \in y} \mathbf{w} \cdot \phi(\mathbf{x}, p)$$

  - Simple and effective lexical representations
  - *Parameter estimation for non-projective parsing*
  - Efficient third-order dependency parsers
- Conclusion

Non-Projective Inference

- Fundamental inference algorithms that sum over possible structures:

Structured Model	*Inference Algorithm*
Hidden Markov Model	Forward-Backward
Graphical Model	Belief Propagation
Context-Free Grammar	Inside-Outside
Projective Dependencies	Inside-Outside
Non-Projective Dependencies	???

- New inference algorithms for non-projective parsing

Log-Linear Dependency Parsers

- Distribution over trees in a first-order factorization

$$P(y \mid \mathbf{x}; \mathbf{w}) \propto \prod_{(h,m) \in y} e^{\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, h, m)}$$

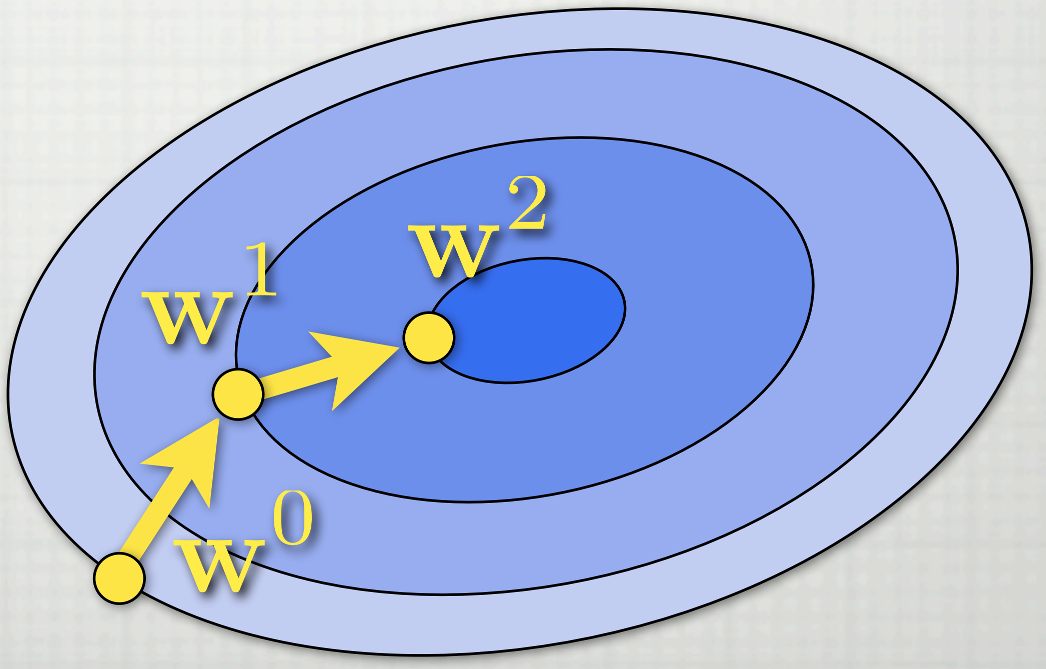
- Parsing is a search for the most probable tree
- A popular method for modeling structured data
- Also known as a Conditional Random Field (CRF)

Log-Linear Parameter Estimation

- Learn $\mathbf{w}$ from labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- Maximize (regularized) conditional log-likelihood:

$$f_{\mathrm{LL}}(\mathbf{w}) \quad = \quad -\frac{C}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \log P(y_i \mid \mathbf{x}_i; \mathbf{w})$$

- Gradient-based optimization
  - $f_{\mathrm{LL}}(\mathbf{w})$ and $\nabla f_{\mathrm{LL}}(\mathbf{w})$
  - e.g., L-BFGS



Log-Linear Inference Problems

- $f_{\mathrm{LL}}(\mathbf{w})$ requires the *partition function*:

$$Z(\mathbf{x};\mathbf{w}) = \sum_{y} \prod_{(h,m)\in y} e^{\mathbf{w}\cdot\boldsymbol{\phi}(\mathbf{x},h,m)}$$

- $\nabla f_{\mathrm{LL}}(\mathbf{w})$ requires the *marginal probabilities*:

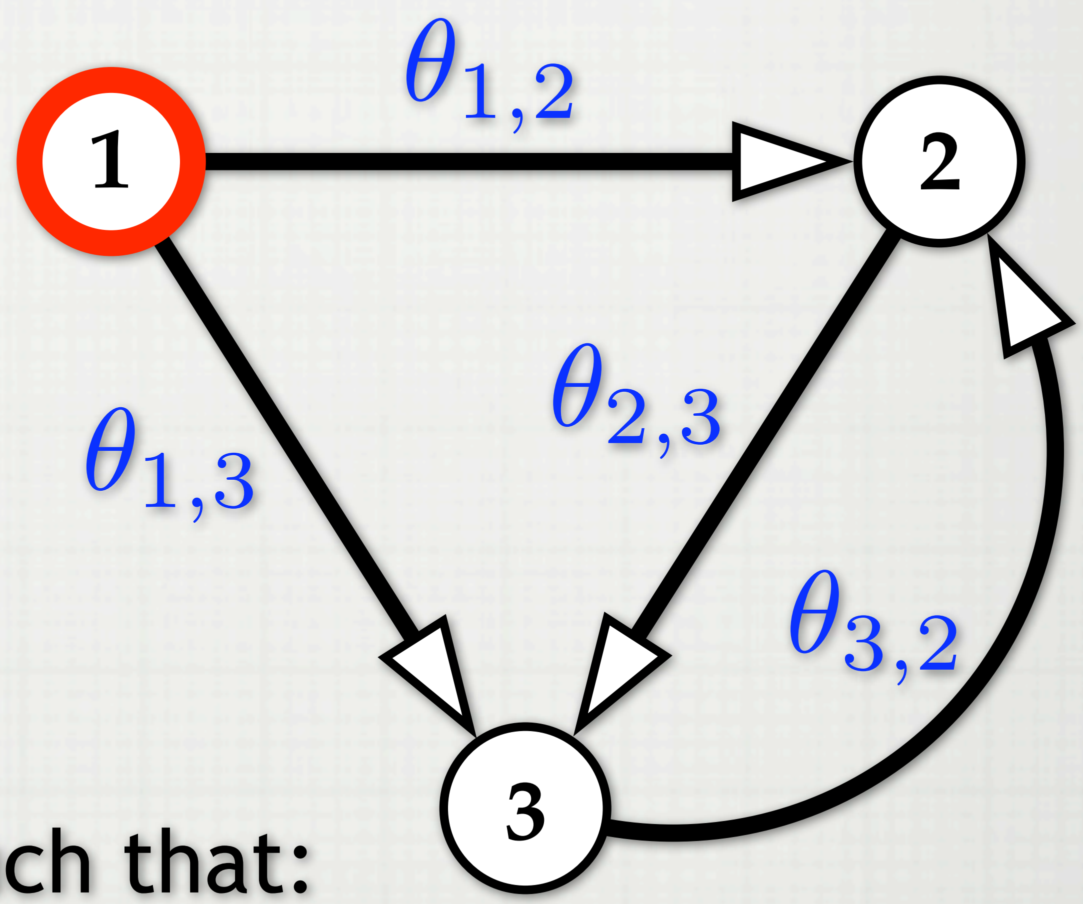
$$P(h,m\,|\,\mathbf{x};\mathbf{w}) = \sum_{y\,:\,(h,m)\in y} P(y\,|\,\mathbf{x};\mathbf{w})$$

The Matrix-Tree Theorem

- Originally developed by Kirchhoff (1847)
  - Count the number of undirected spanning trees
  - Determinant of a specially-constructed matrix
- Extended by Tutte (1984)
  - Summations over weighted, rooted, directed spanning trees

The Matrix-Tree Theorem

- Given:
  - Directed graph G
  - Edge weights $\theta_{i,j}$
  - Root node r

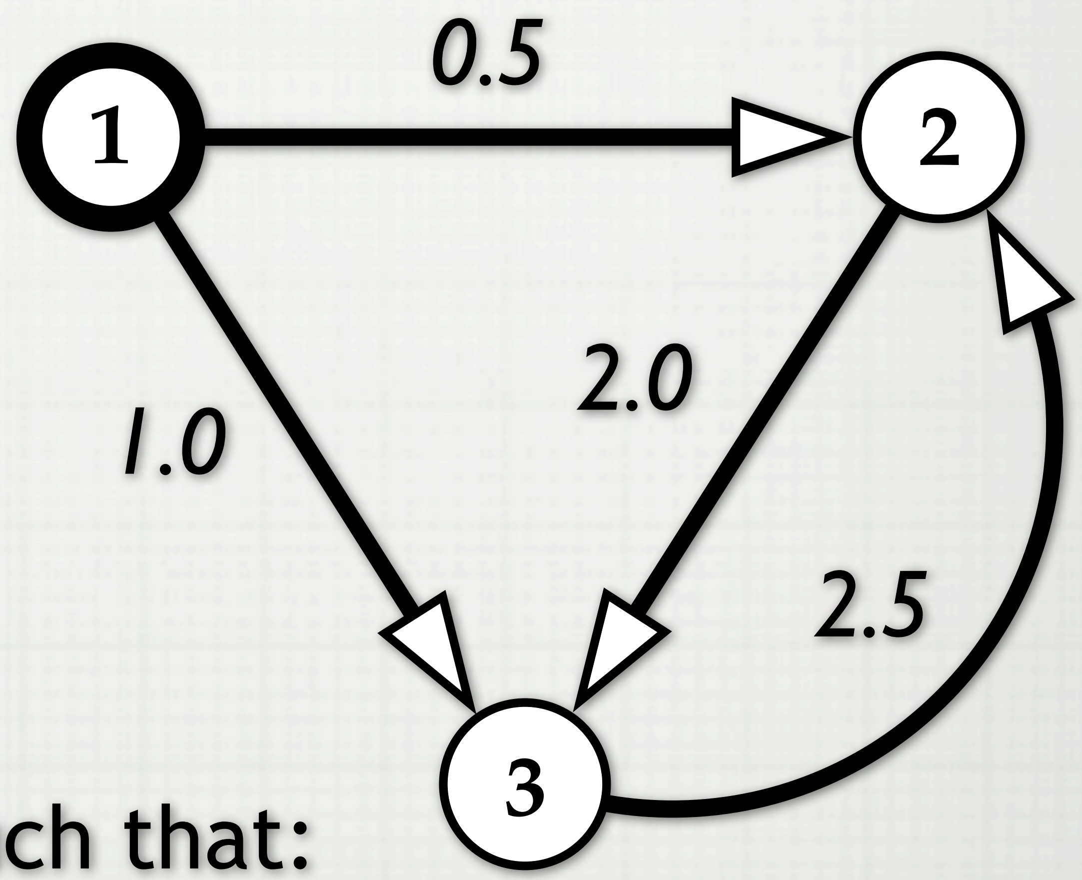


- Construct a matrix $L^{(r)}$ such that:

$$\left|L^{(r)}\right| = \sum_{\text{Trees}} \prod_{\text{Edges in Tree}} e^{\theta_{h,m}}$$

The Matrix-Tree Theorem

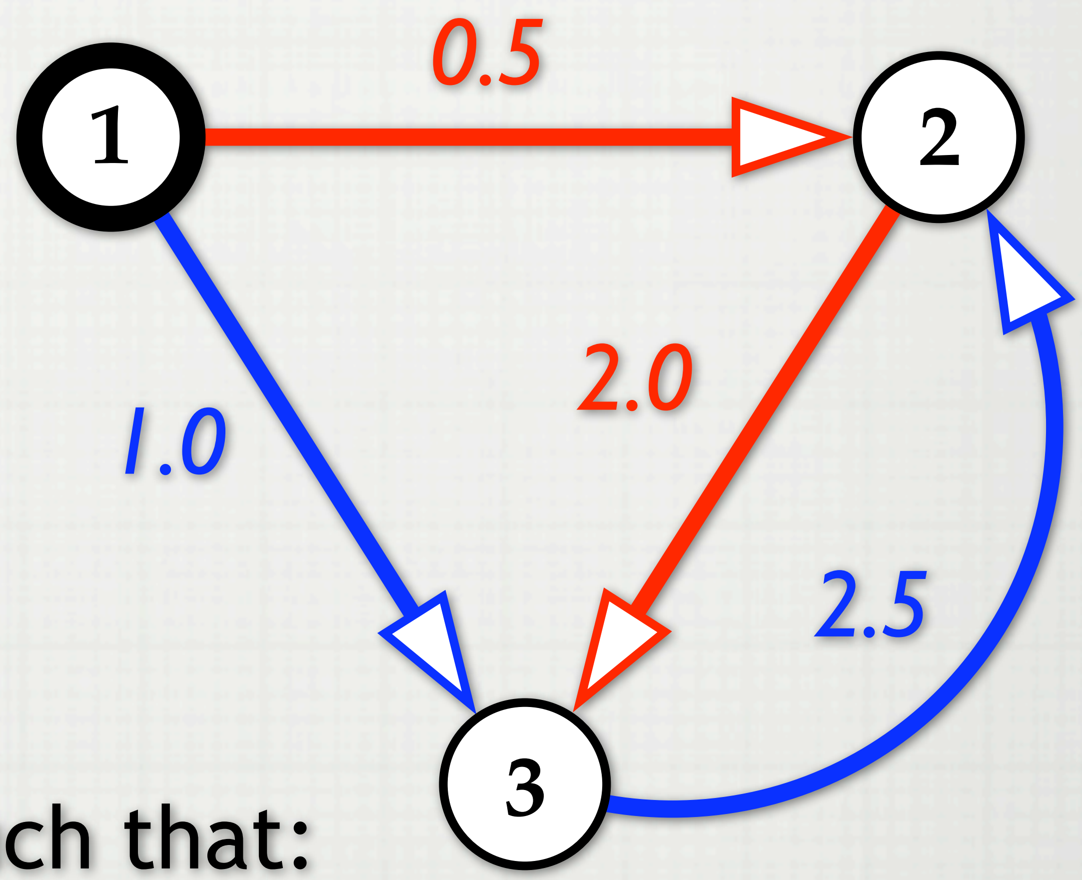
- Given:
  - Directed graph G
  - Edge weights $\theta_{i,j}$
  - Root node r
- Construct a matrix $L^{(r)}$ such that:



$$\left| L^{(r)} \right| \quad =$$

The Matrix-Tree Theorem

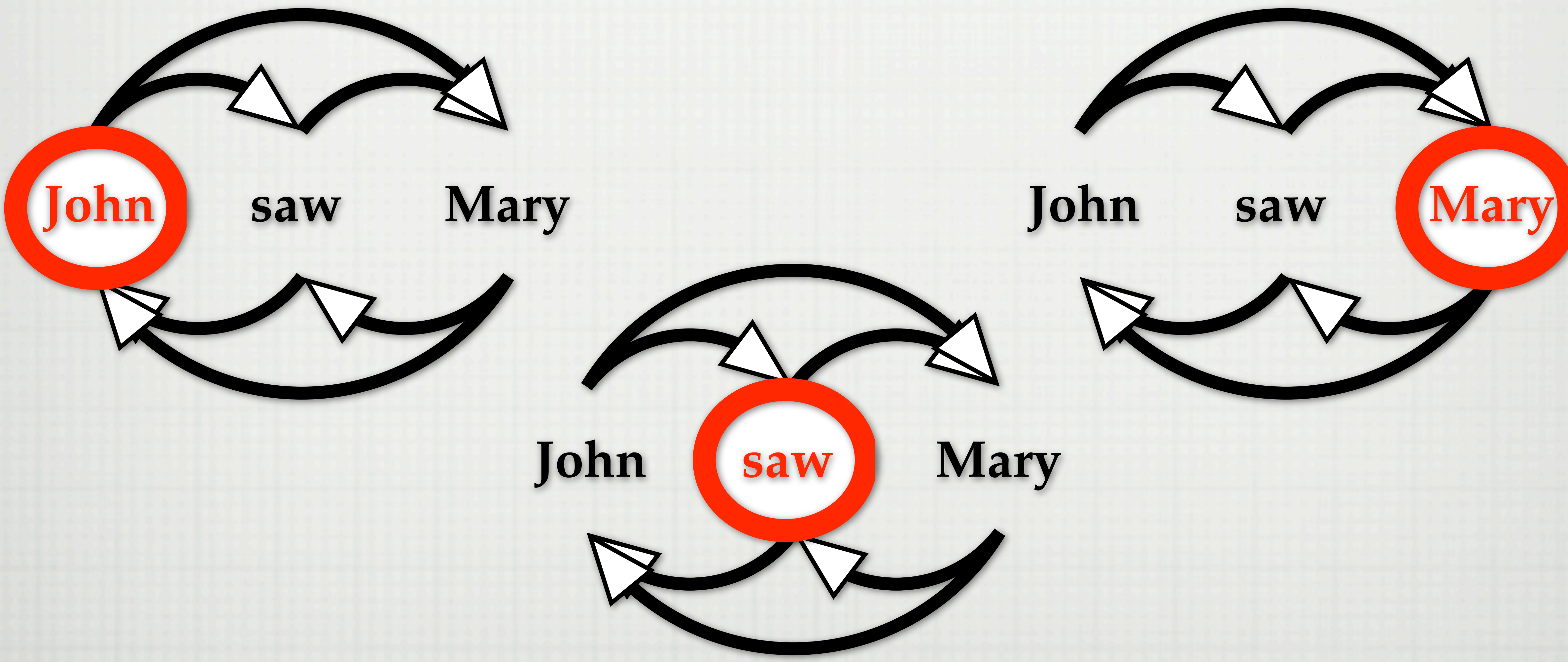
- Given:
  - Directed graph G
  - Edge weights $\theta_{i,j}$
  - Root node r
- Construct a matrix $L^{(r)}$ such that:



$$\left|L^{(r)}\right| = e^{1.0+2.5} + e^{0.5+2.0}$$

The Partition Function

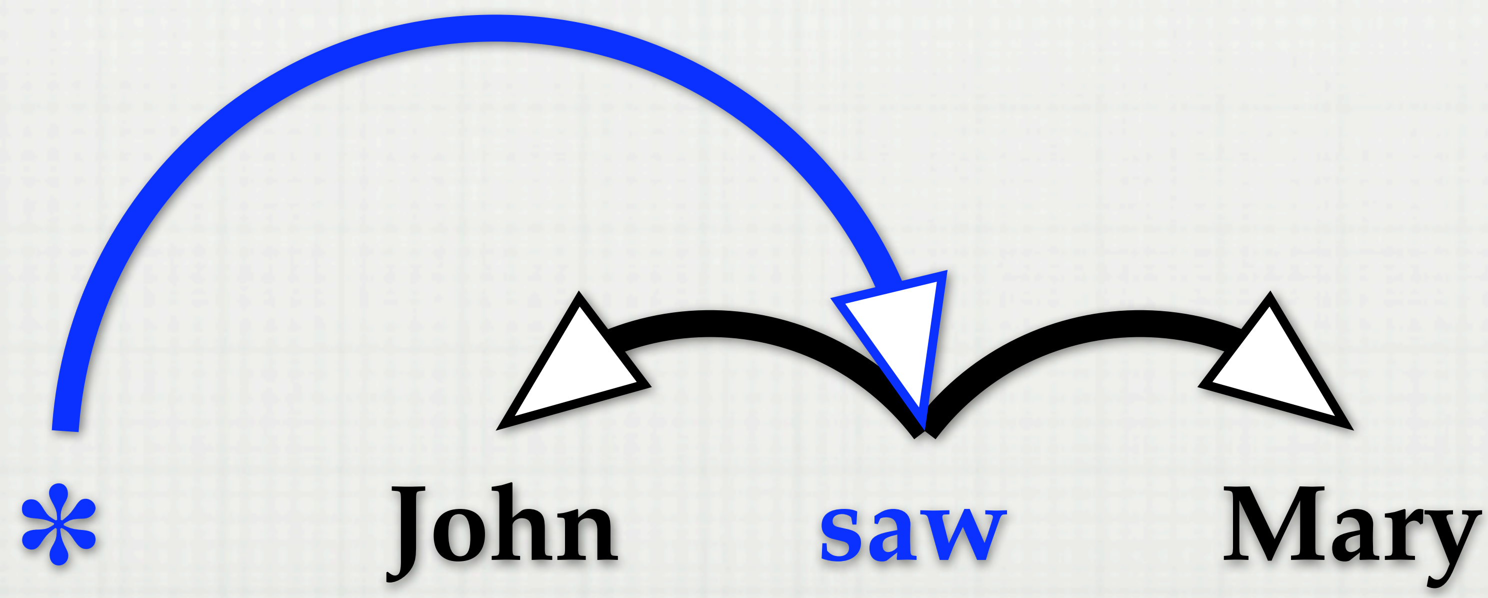
- A naive method: one invocation per root



- Inefficient: requires n determinants

The Partition Function

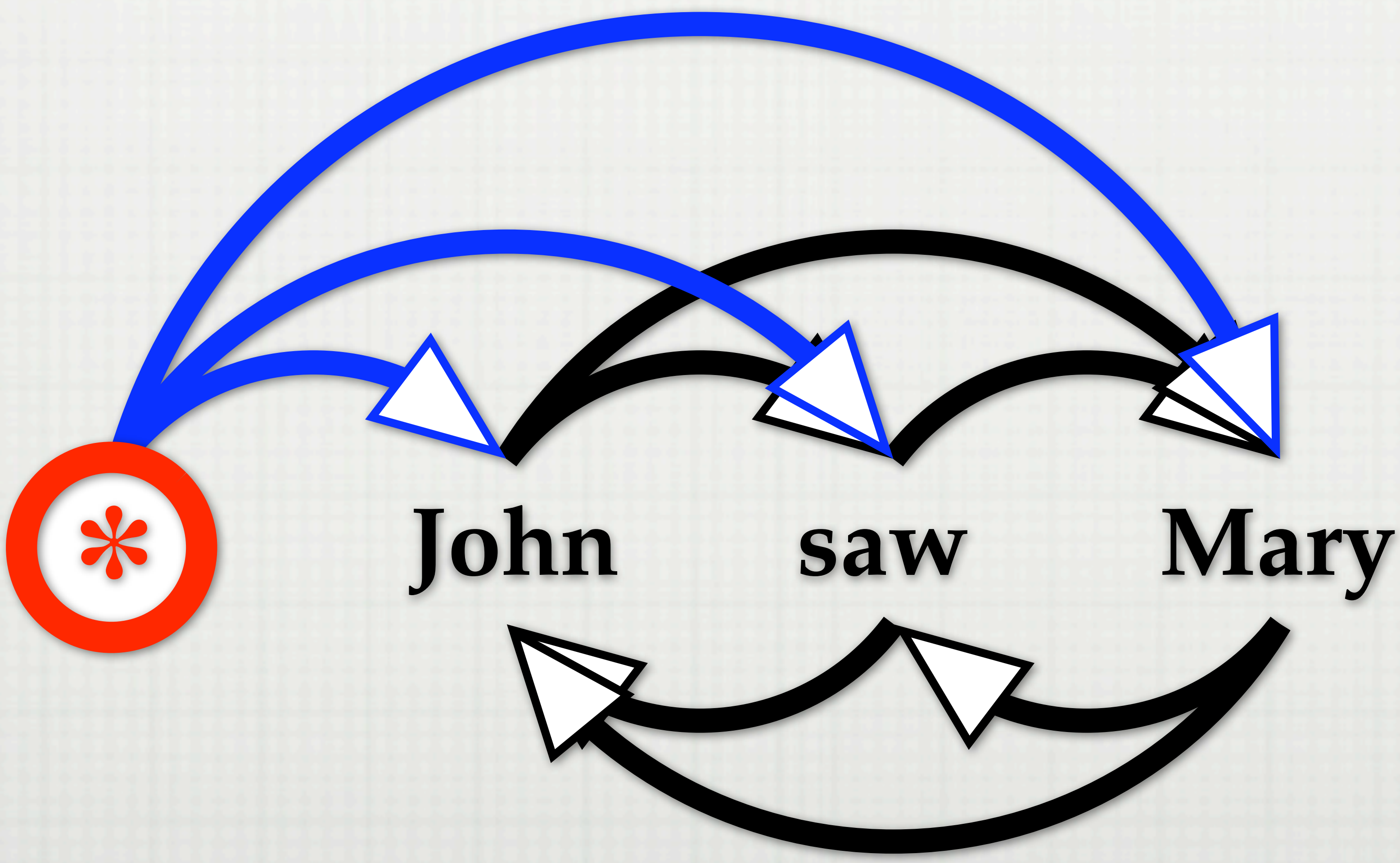
- A simple method for summing over all roots:



- The modifier of * is the root

The Partition Function

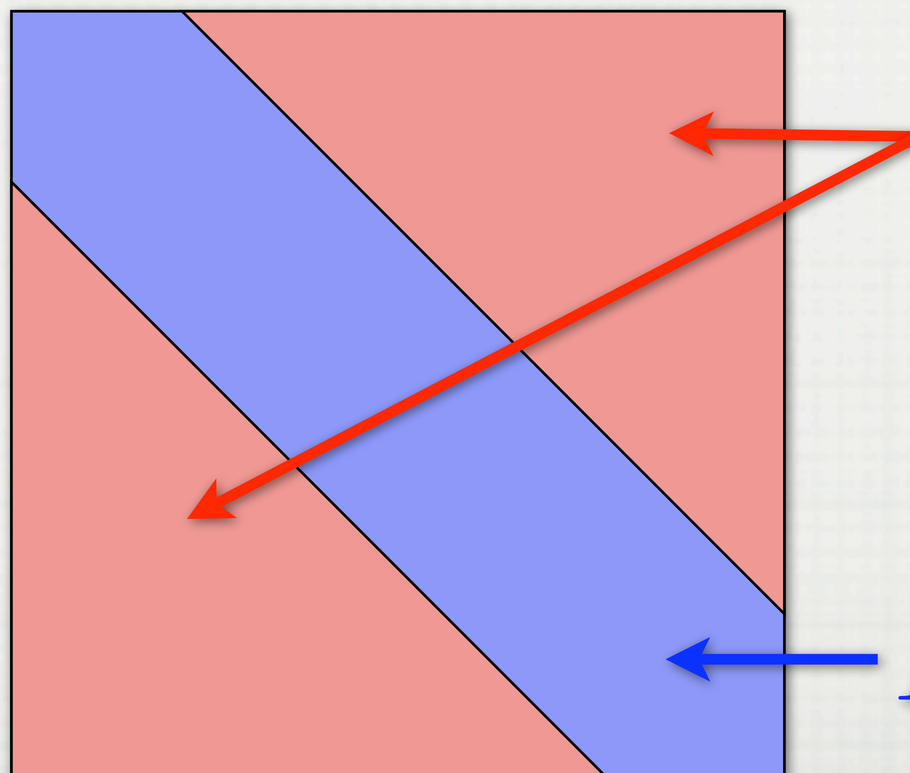
- A simple method for summing over all roots: $\left|L^{(*)}\right|$



The Partition Function

- A simple method for summing over all roots: $\left|L^{(*)}\right|$

$$L^{(*)} =$$



$$L^{(*)}_{i,j} = -e^{\theta_{i,j}}$$

$$L^{(*)}_{j,j} = \sum_i e^{\theta_{i,j}}$$

- Determinant of $n \times n$ matrix: $O(n^3)$

Multi-Root Dependency Trees

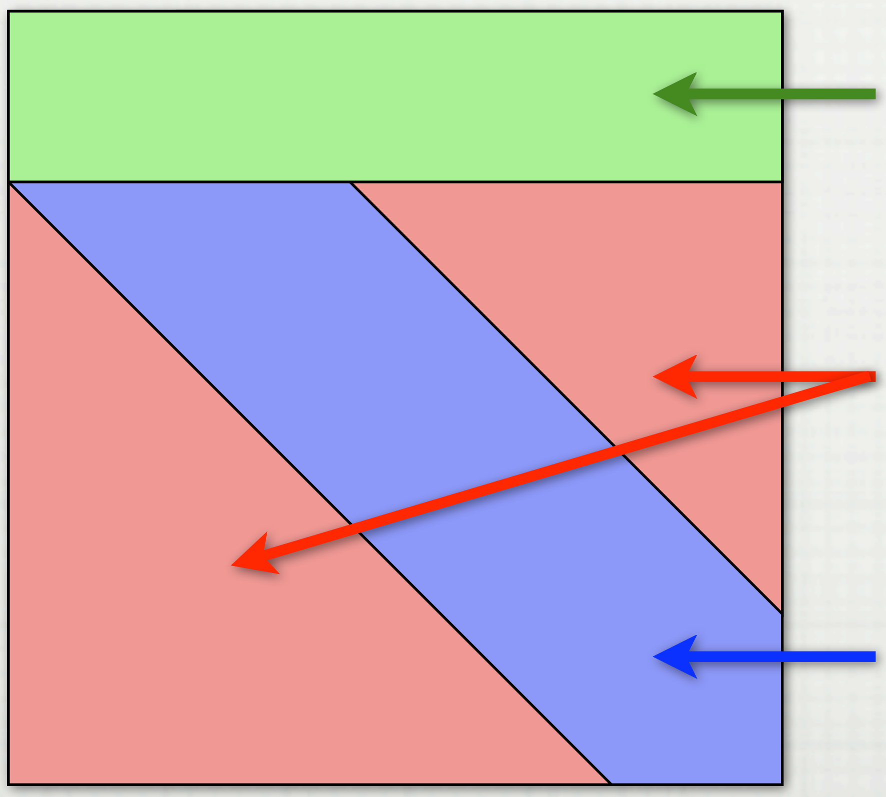
- A simple method for summing over all roots: $\left| L^{(*)} \right|$

- Structures with *multiple* roots are counted

Single-Root Partition Function

- A new matrix for summing over single-root trees:

$$\hat{L} =$$



$$\hat{L}_{1,j} = e^{\theta_{*,j}}$$

$$\hat{L}_{i,j} = -e^{\theta_{i,j}}$$

$$\hat{L}_{j,j} = \sum_{i \neq *} e^{\theta_{i,j}}$$

- Determinant of $n \times n$ matrix: $O(n^3)$

Marginal Probabilities

- Single-root and multi-root partition functions:

$$Z(\boldsymbol{\theta}) = \left|\hat{L}\right| \qquad\qquad Z(\boldsymbol{\theta}) = \left|L^{(*)}\right|$$

- Marginals are derivatives of log partition function:

$$P(h \rightarrow m; \boldsymbol{\theta}) \;=\; \frac{\partial \log Z(\boldsymbol{\theta})}{\partial \theta_{h,m}}$$

- Derivative of log-determinant: $\frac{\partial \log |X|}{\partial X} = \left(X^{-1}\right)^T$
- Inverse of $n \times n$ matrix: $O(n^3)$

Application to Parsing

- Training a log-linear parser:
  - Define edge scores $\theta_{h,m} = \mathbf{w} \cdot \phi(\mathbf{x}, h, m)$
  - Construct appropriate matrix $\hat{L}$ or $L^{(*)}$
  - $Z(\mathbf{x}; \mathbf{w})$ via matrix determinant
  - $P(h, m \,|\, \mathbf{x}; \mathbf{w})$ via matrix inverse
- Max-margin training for dependency parsers
  - Exponentiated Gradient (Collins et al., 2008)

Multilingual Parsing Experiments

- Six languages from CoNLL-X shared task
- Three training algorithms:
  - Averaged perceptron
  - Log-linear models
  - Max-margin models
- Projective and non-projective parsing

Dutch Parsing Experiments

Training Algorithm	*Projective Training*	*Non-Projective Training*
Perceptron	77.2	78.8 (+1.6)
Log-Linear	76.2	79.6 (+3.4)
Max-Margin	76.5	79.7 (+3.2)

- Attachment score on Dutch test set
- 4.93% of dependencies are crossing
- Non-projective training is beneficial for languages with non-projectivity

Aggregate Multilingual Results

Training Algorithm	*Overall Results*
Perceptron	79.1
Log-Linear	79.7 (+0.6)
Max-Margin	79.8 (+0.7)

- Cumulative attachment score over 6 languages:
  - Arabic, Dutch, Japanese, Slovene, Spanish, Turkish
- Improvements are statistically significant

Summary

- New algorithms for weighted summations over non-projective dependency trees
  - Covering both single-root and multi-root trees
  - Efficient $O(n^3)$ algorithms
- An application: log-linear and max-margin parsers

Outline

- Introduction
- Three advances in discriminative dependency parsing:

$$\operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \sum_{p \in y} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, p)$$

  - Simple and effective lexical representations
  - Parameter estimation for non-projective parsing
  - *Efficient third-order dependency parsers*
- Conclusion

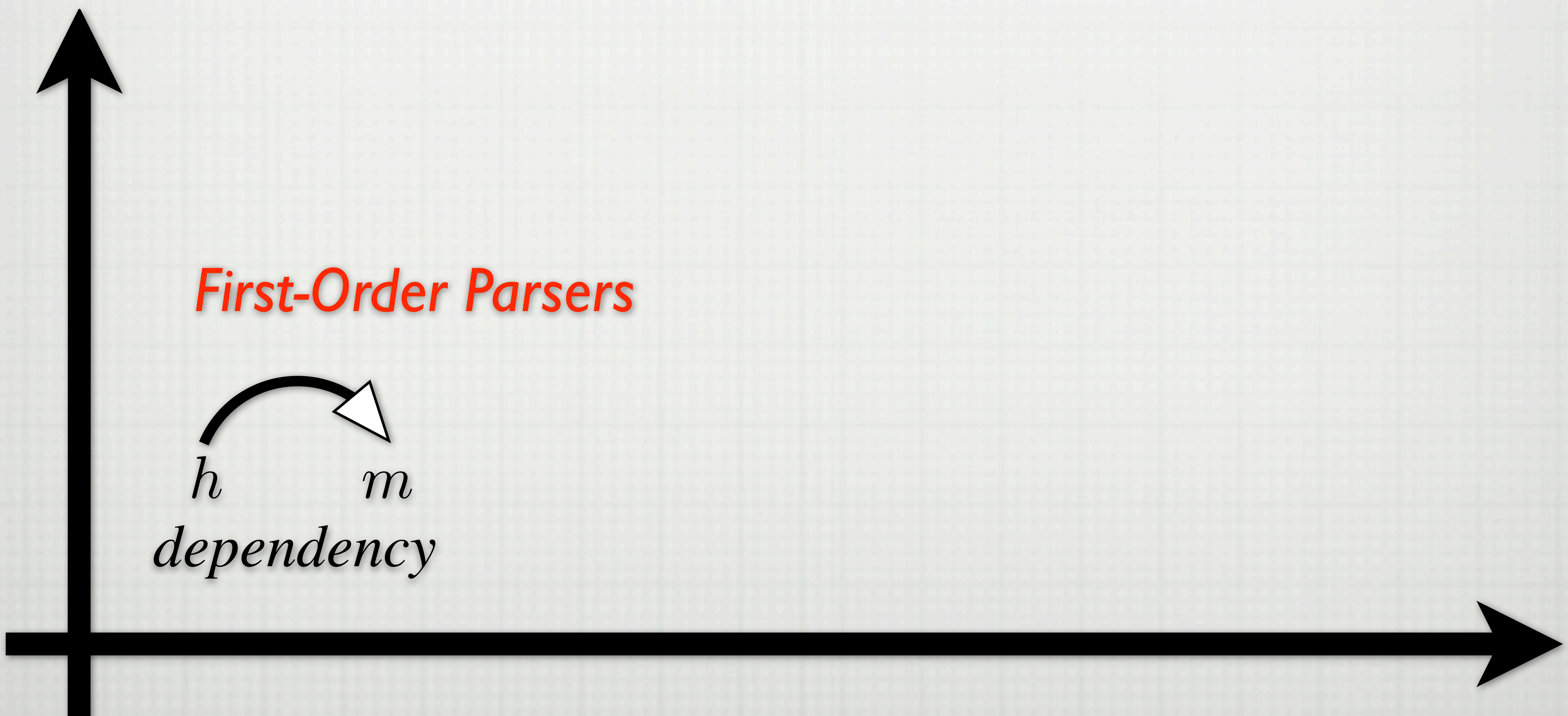
Higher-Order Parsers

Parsing Approach	*First-Order*	*Second-Order*
McDonald's Models	90.9	91.5 (+0.6)
Baseline Features	90.8	92.0 (+1.2)
Cluster-Based Features	92.2	93.2 (+1.0)

- Attachment scores on English test set
- Can we get more by going *beyond* second-order?
- How much will it cost to get there?
  - Carreras (2007) second-order is already $O(n^4)$

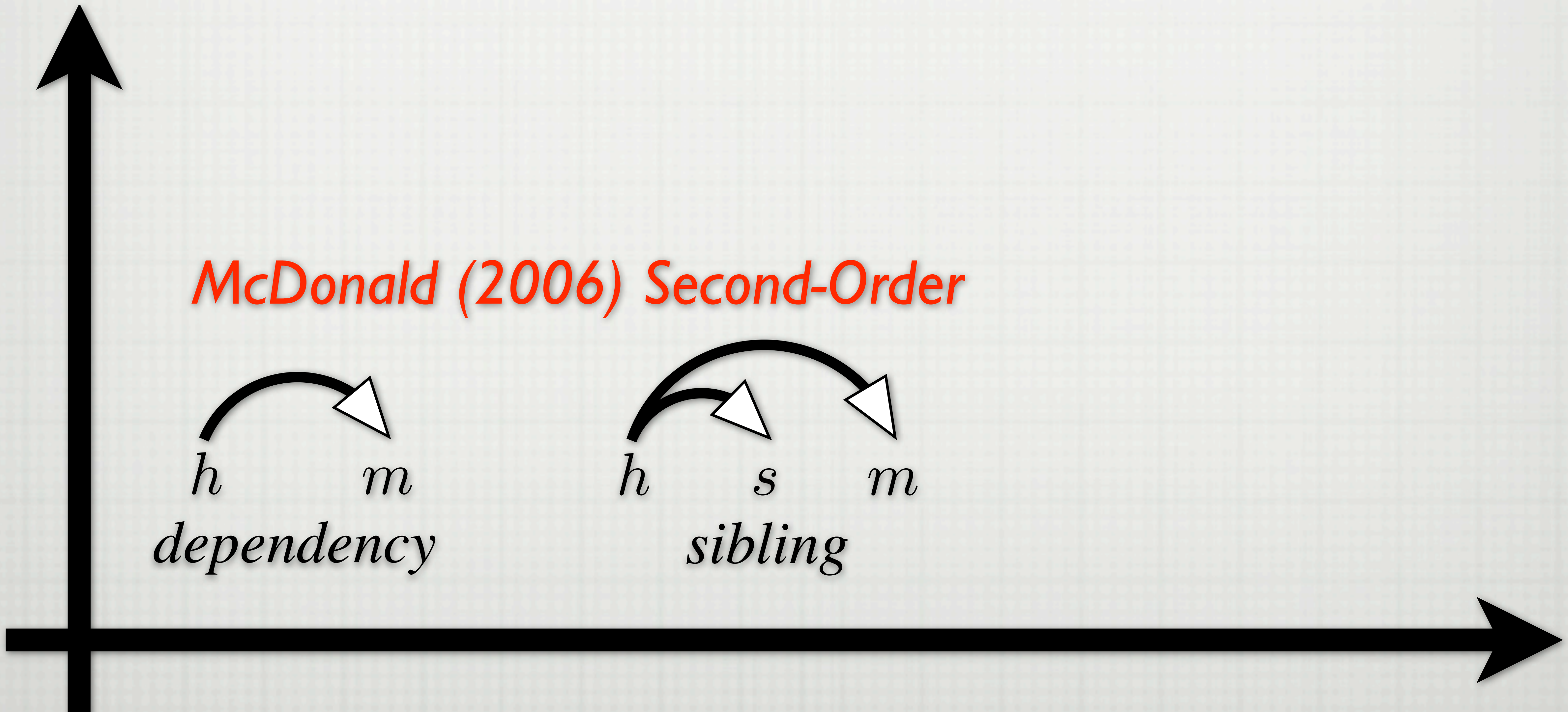
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context



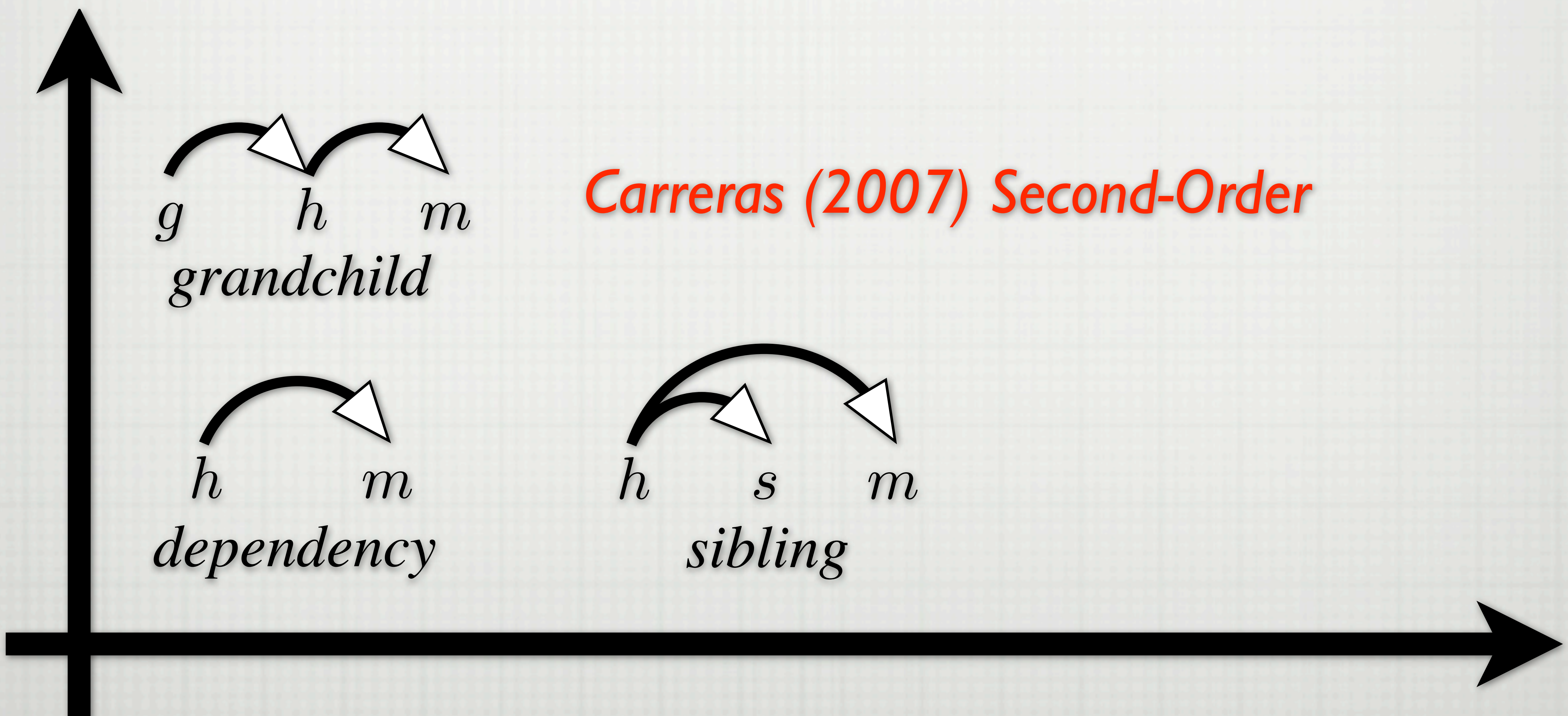
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context



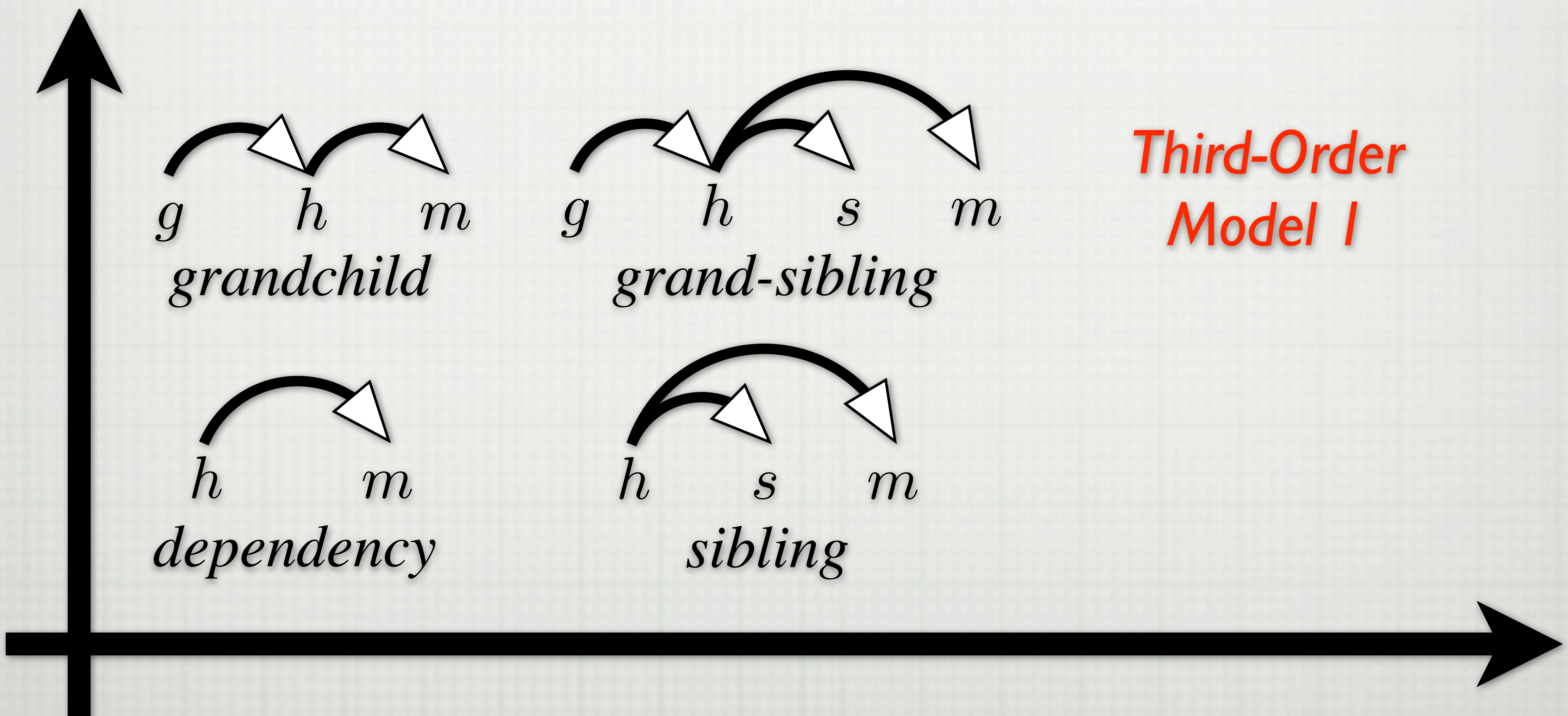
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context



Third-Order Factorizations

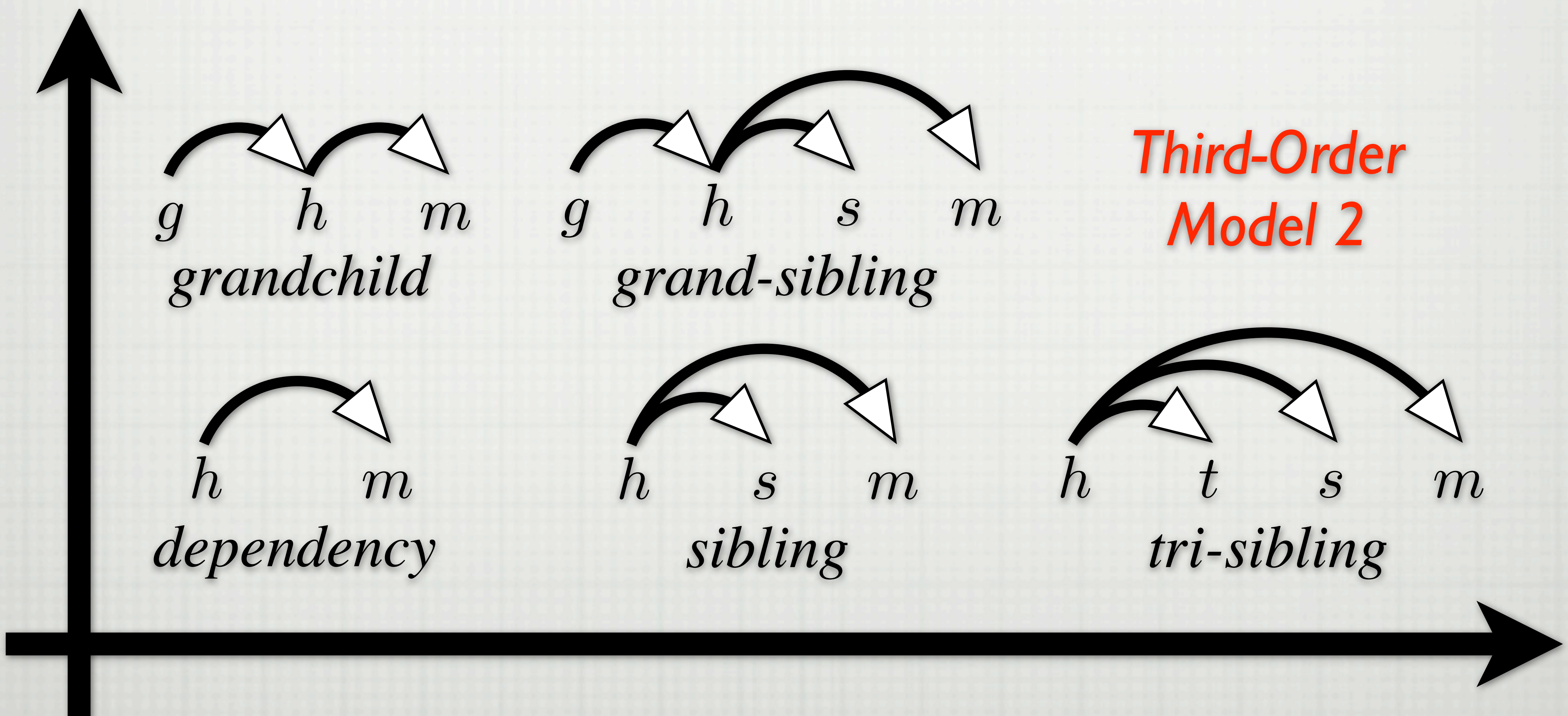
- Two axes: Vertical context and Horizontal context



Third-Order
Model 1

Third-Order Factorizations

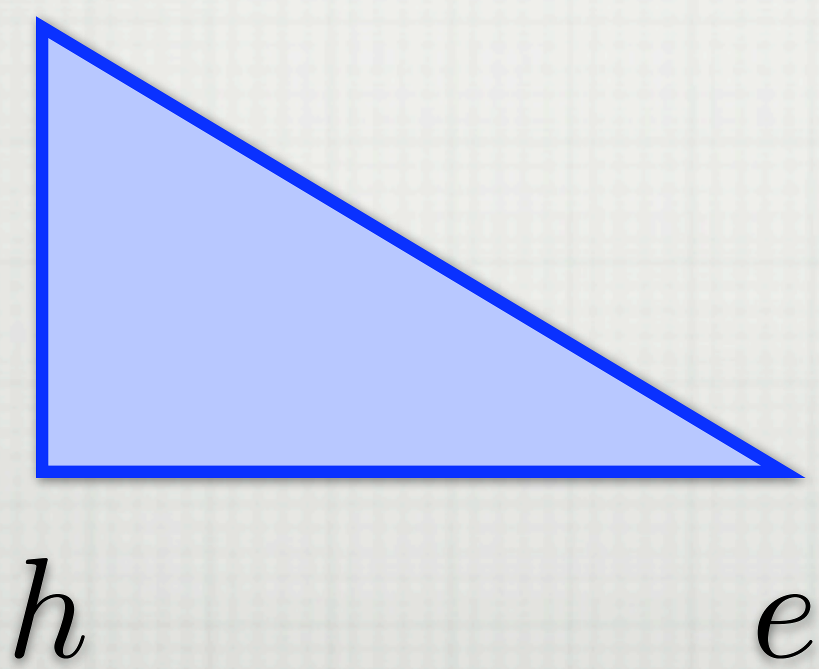
- Two axes: Vertical context and Horizontal context



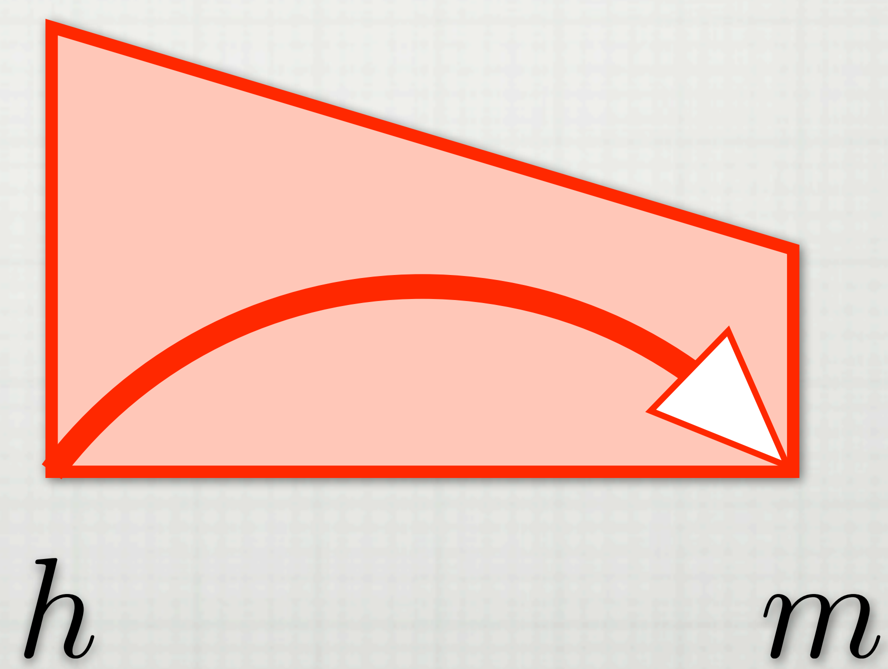
First-Order Parsing Algorithm

- Eisner (2000) algorithm: $O(n^3)$

Complete Span
A "half-constituent"

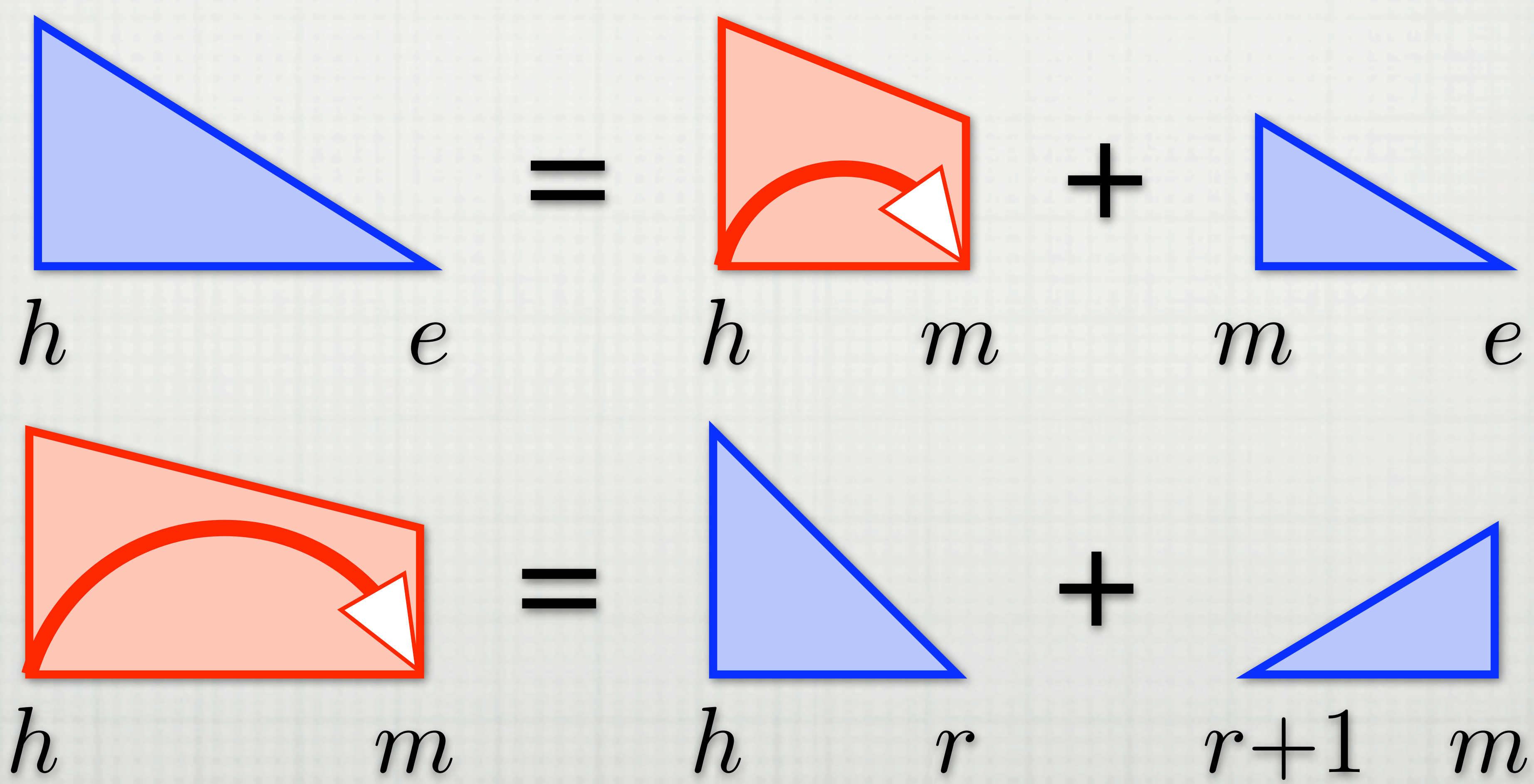


Incomplete Span
A dependency



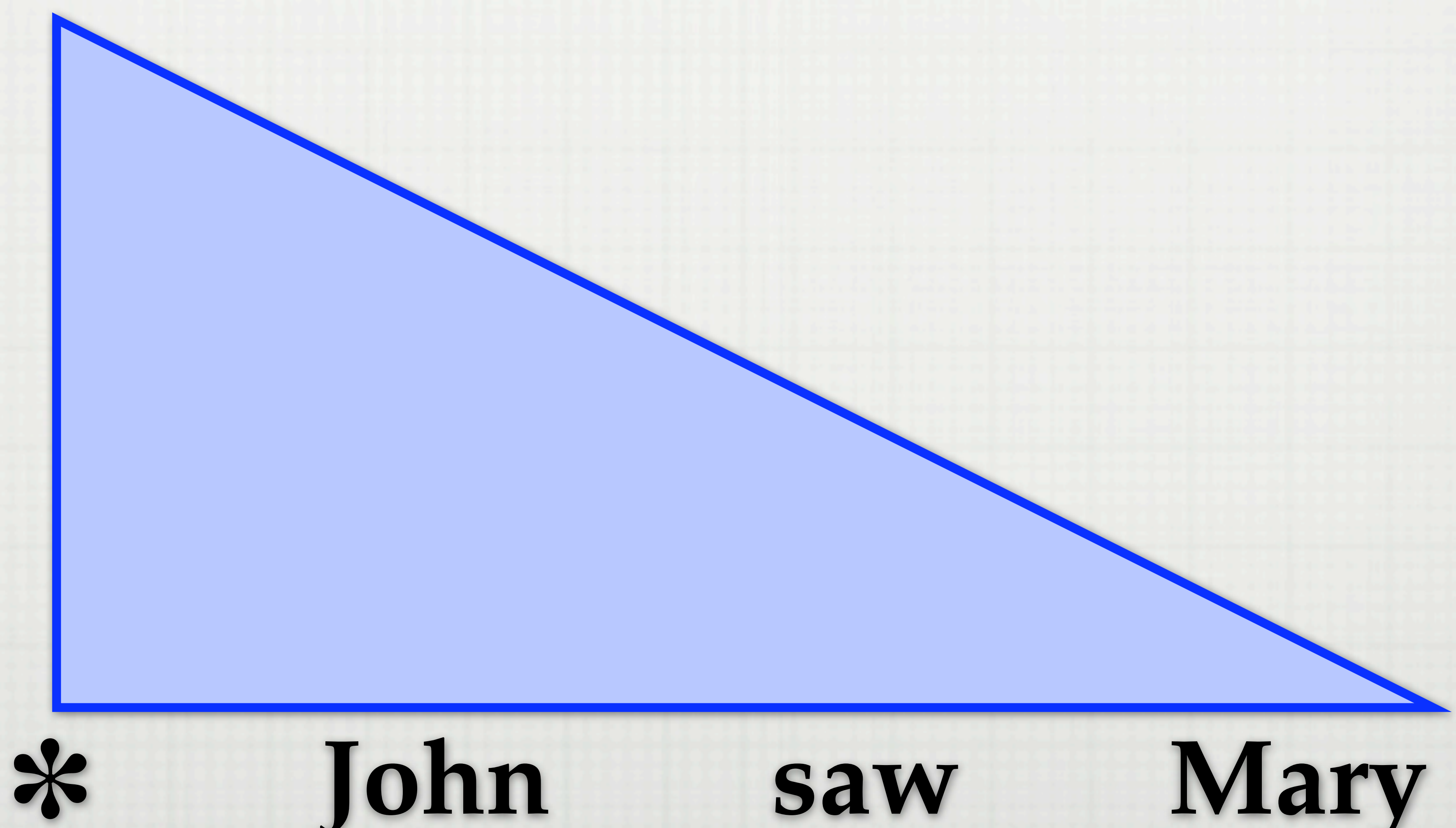
First-Order Parsing Algorithm

- Eisner (2000) algorithm: $O(n^3)$
  - Derivation of *complete* and *incomplete spans*:



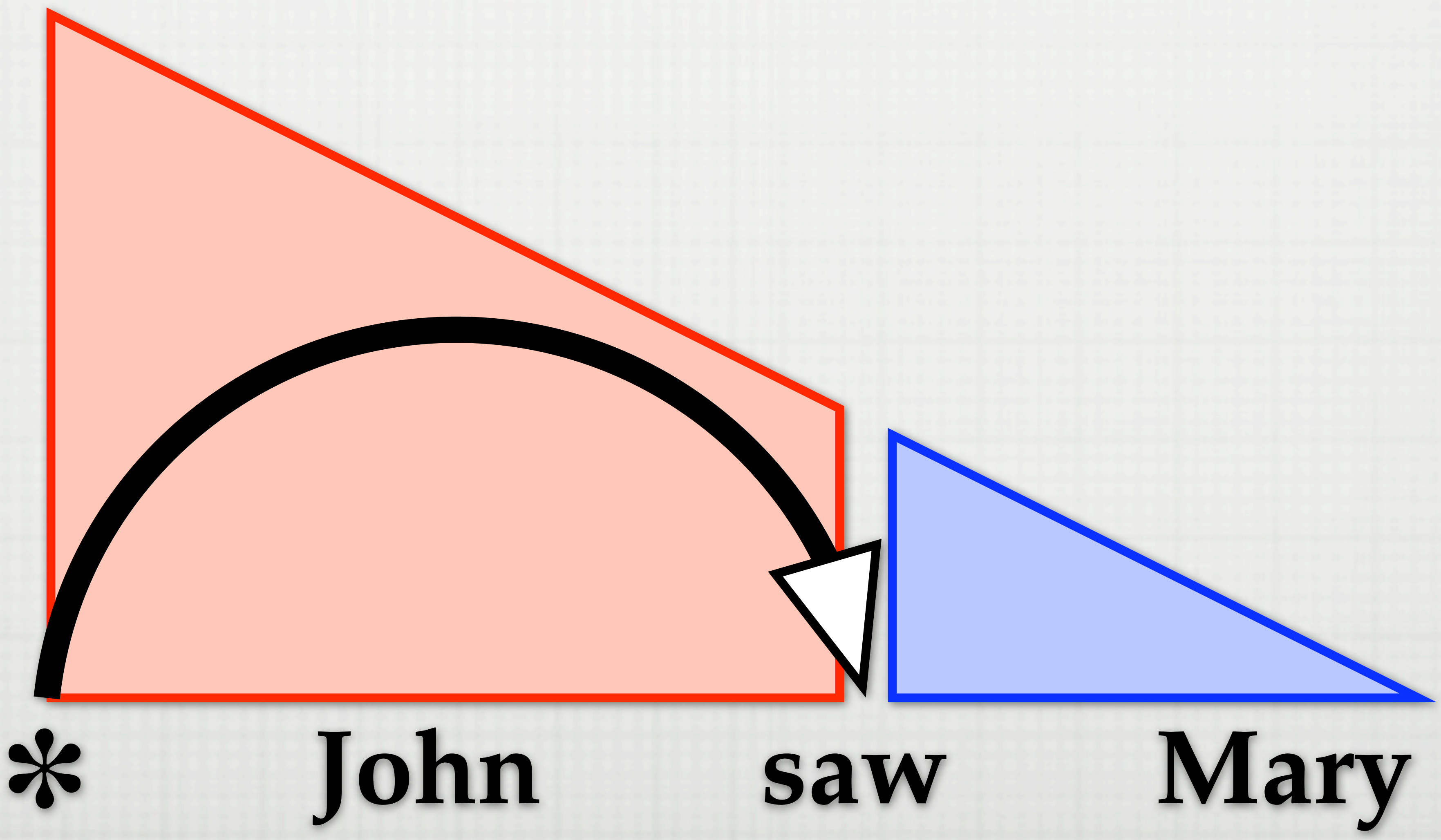
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$



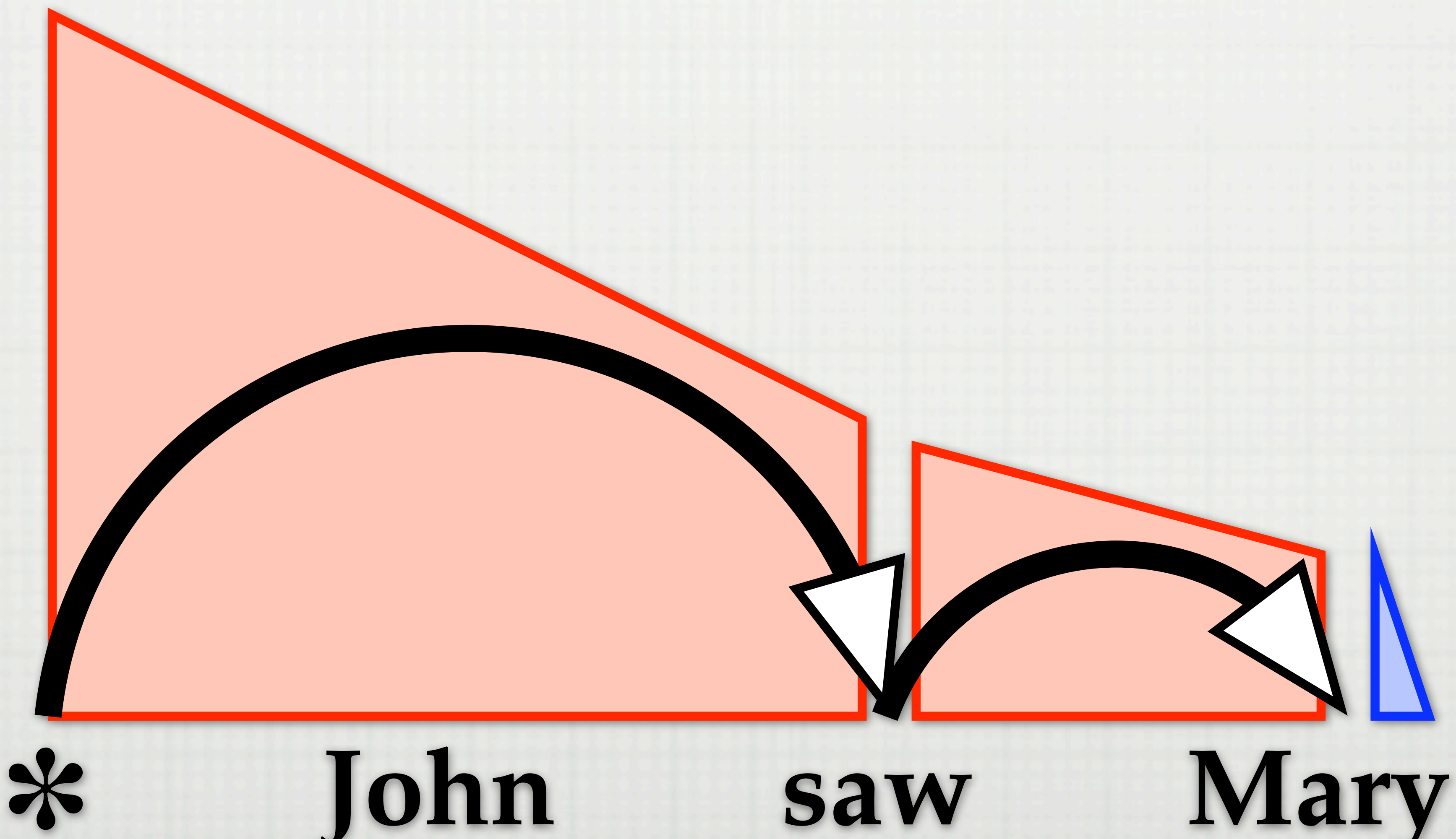
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$



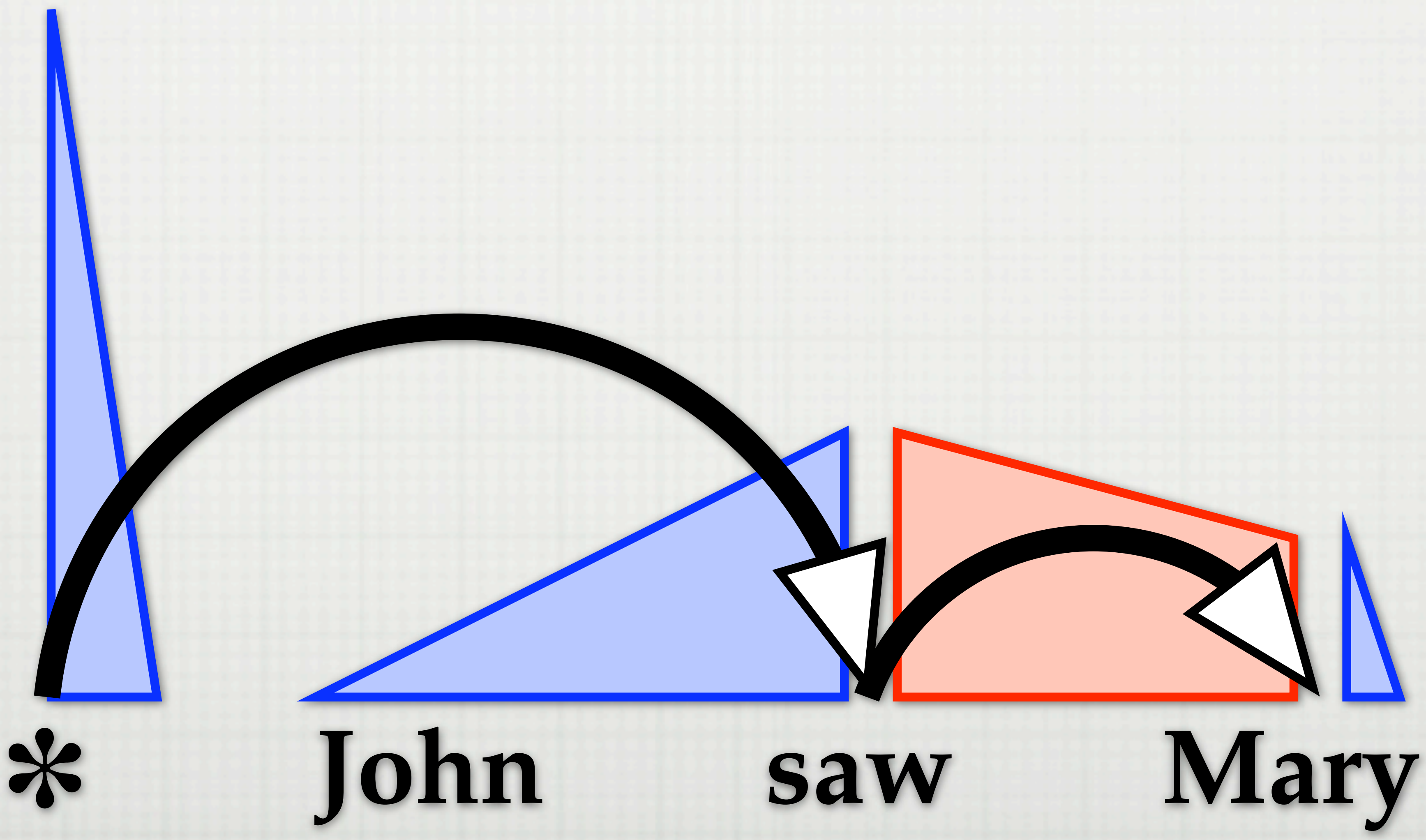
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$



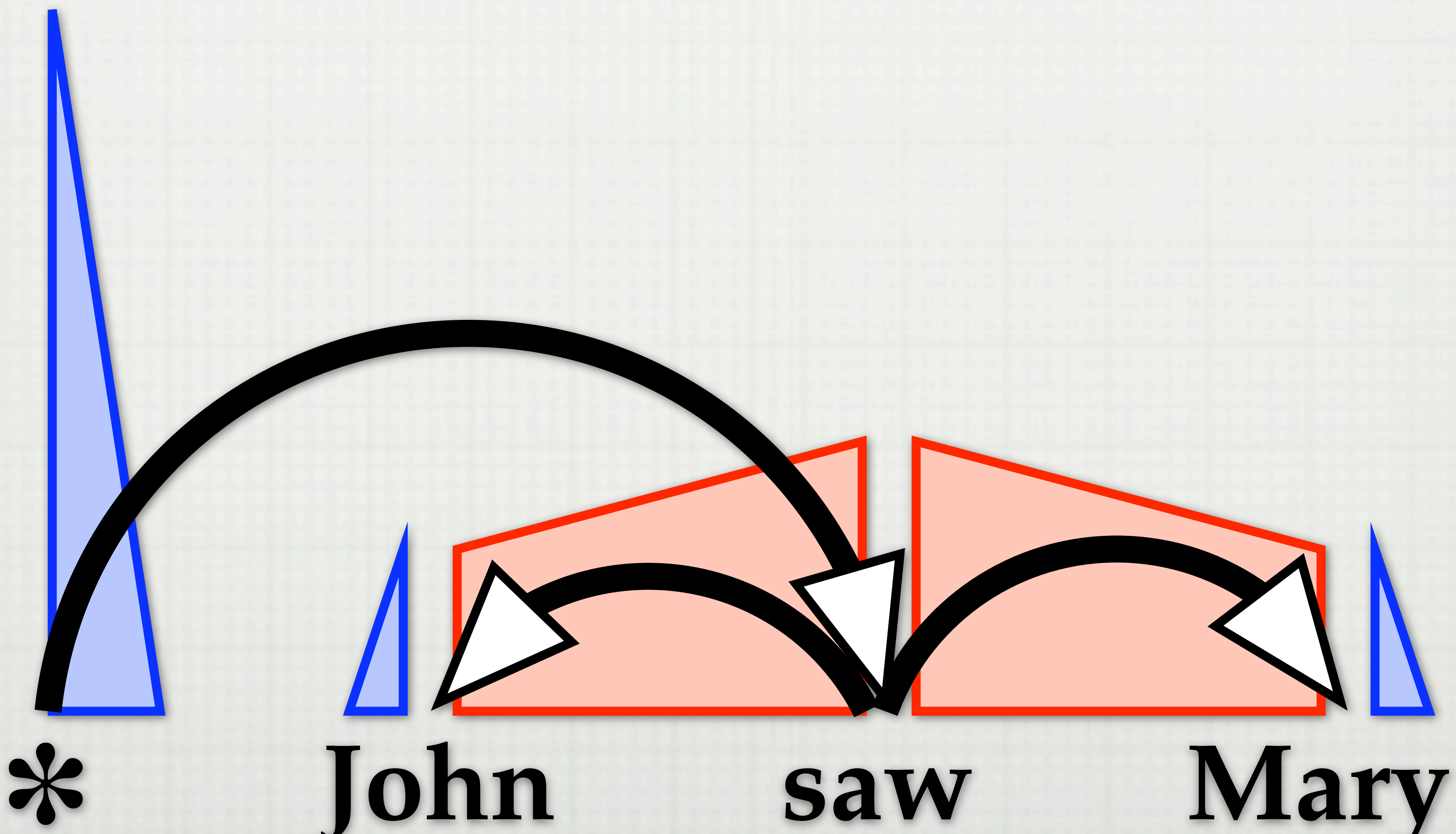
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$



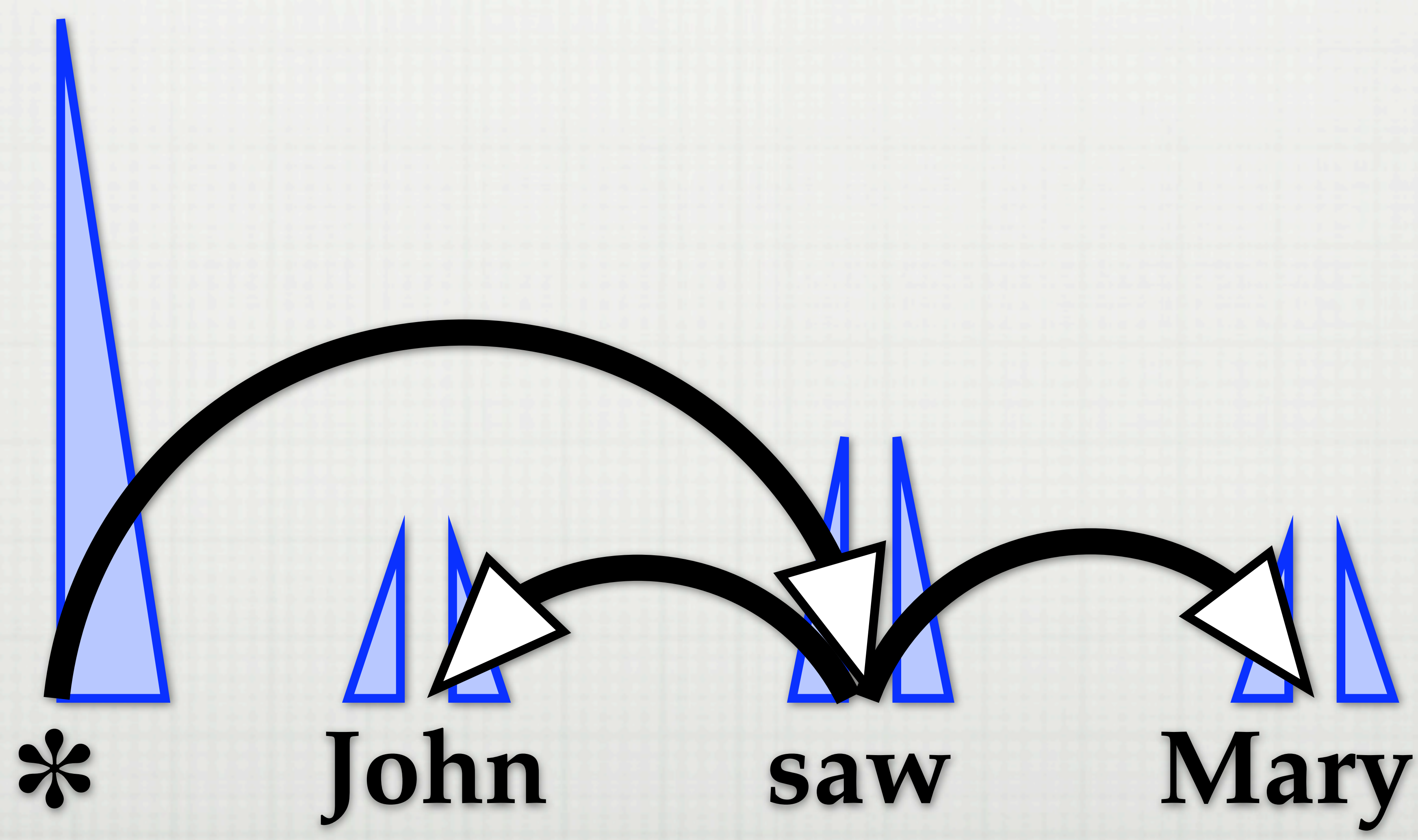
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$



First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$

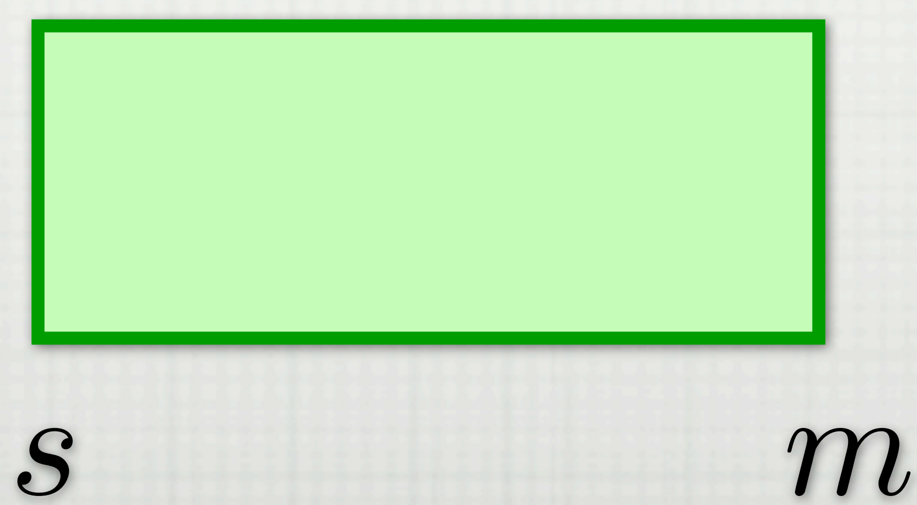


Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$
  - Introduce a third type of span:

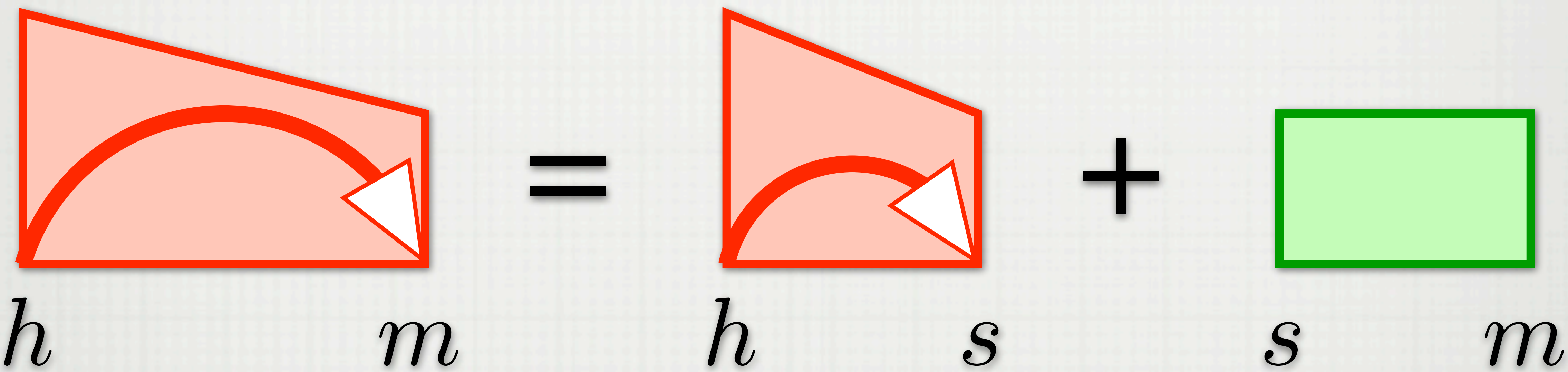
Sibling Span

A pair of adjacent modifiers

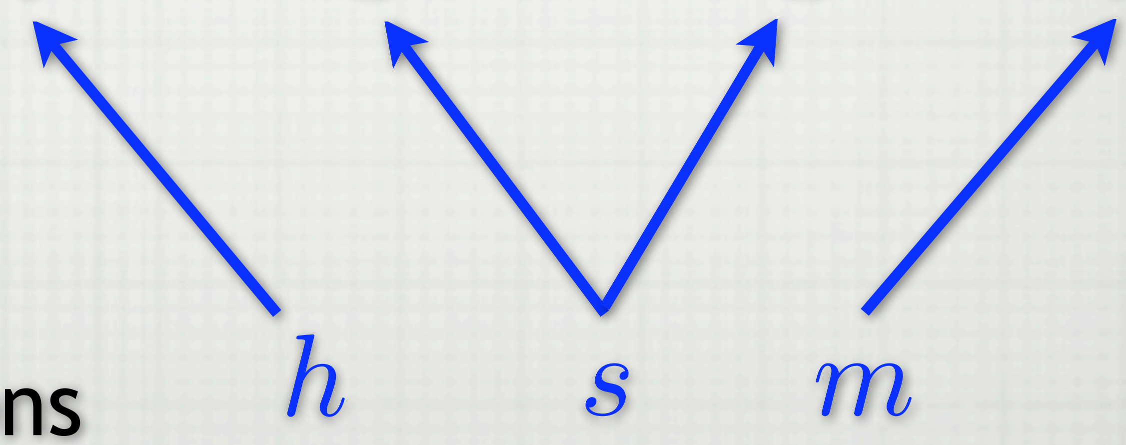


Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$

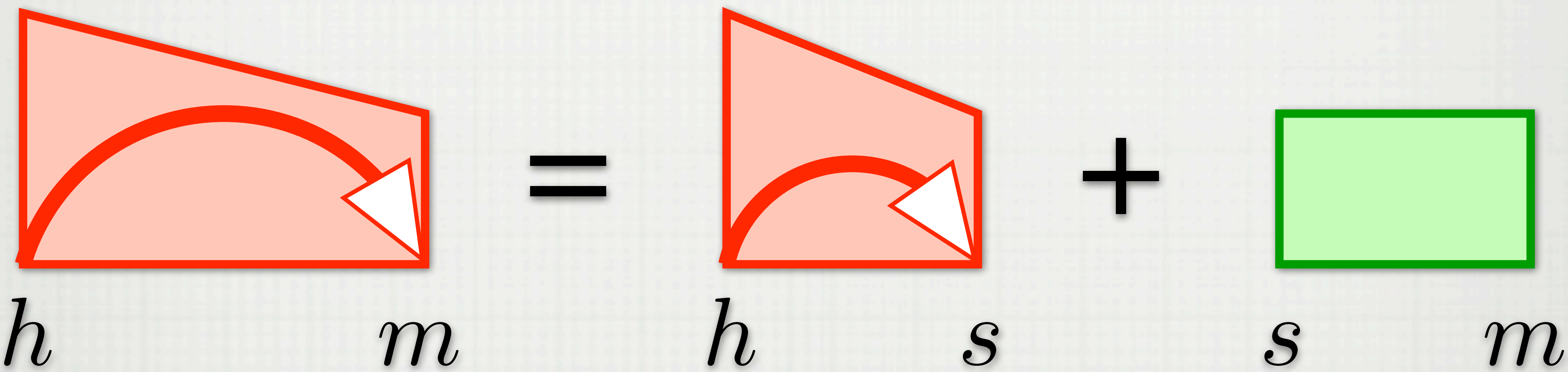


- Scores sibling interactions

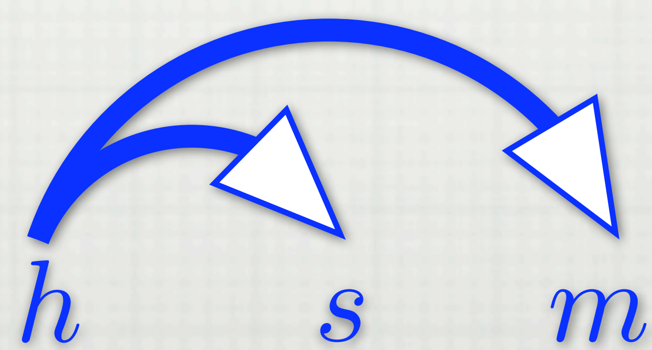


Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$

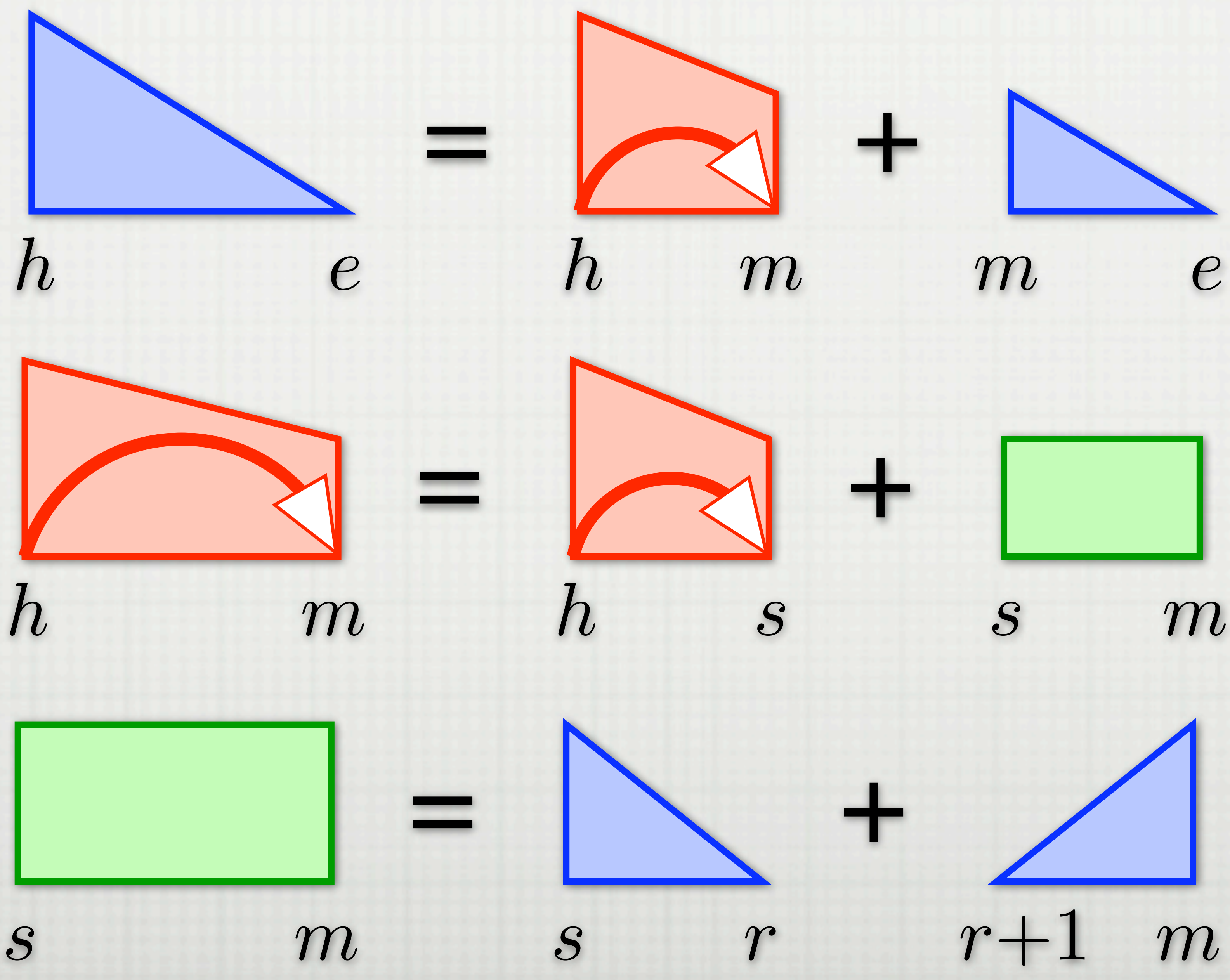


- Scores sibling interactions



Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$



Model 0

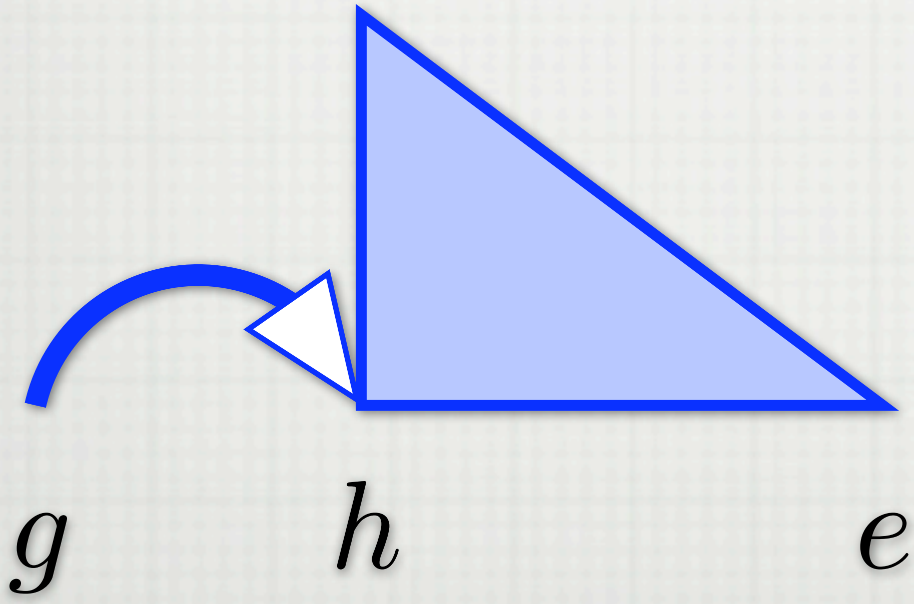
- Model 0, all grandparents: $O(n^4)$

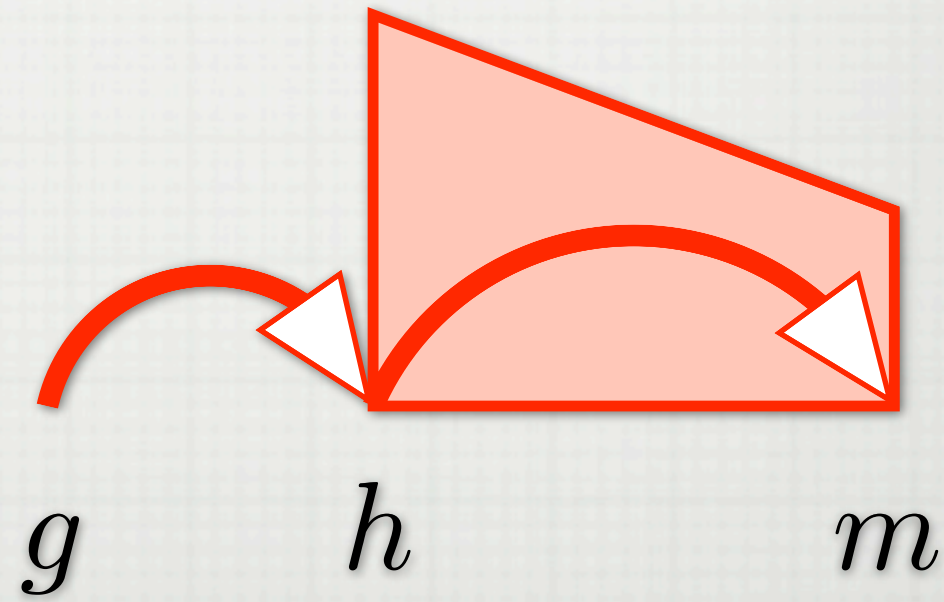
Complete G-Span

A "half-constituent"
with its grandparent

Incomplete G-Span

A dependency
with its grandparent

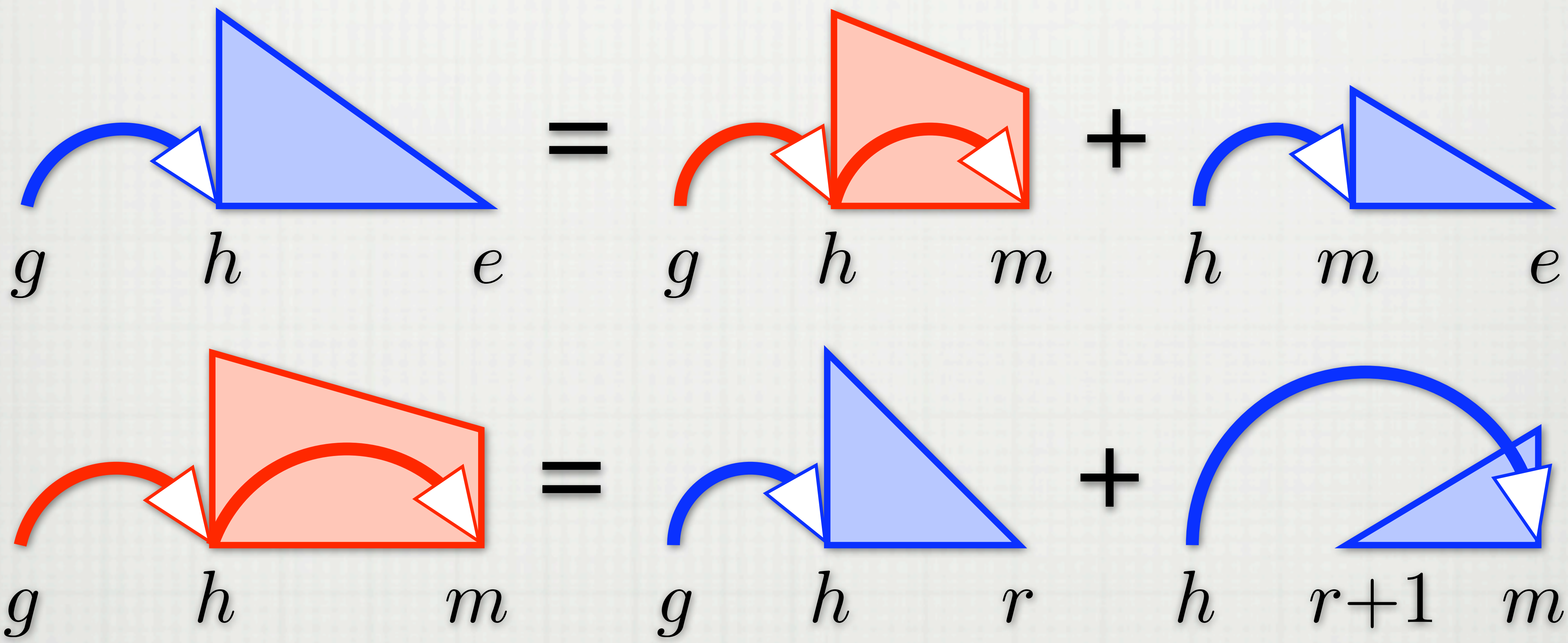




- Superficially similar to parent annotation in CFGs

Model 0: Derivations

- Model 0, all grandparents: $O(n^4)$



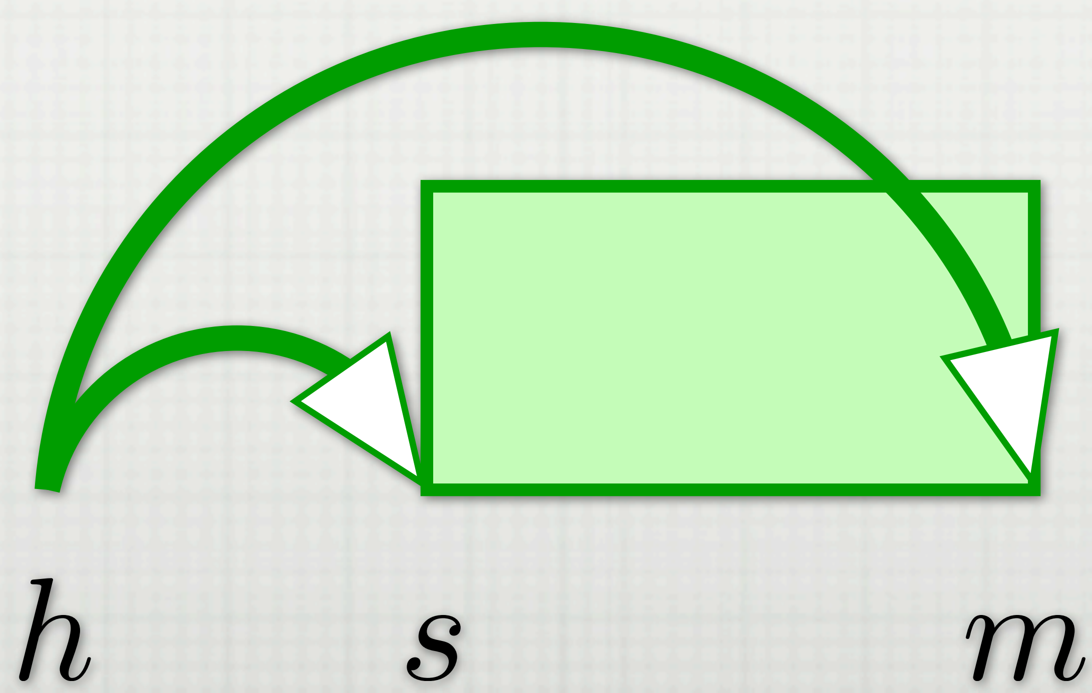
- Grandparent indices propagated to smaller g-spans
- 4 active indices, runtime $O(n^4)$

Model 1

- Model 1, grand-siblings: $O(n^4)$
  - Introduce a third type of span:

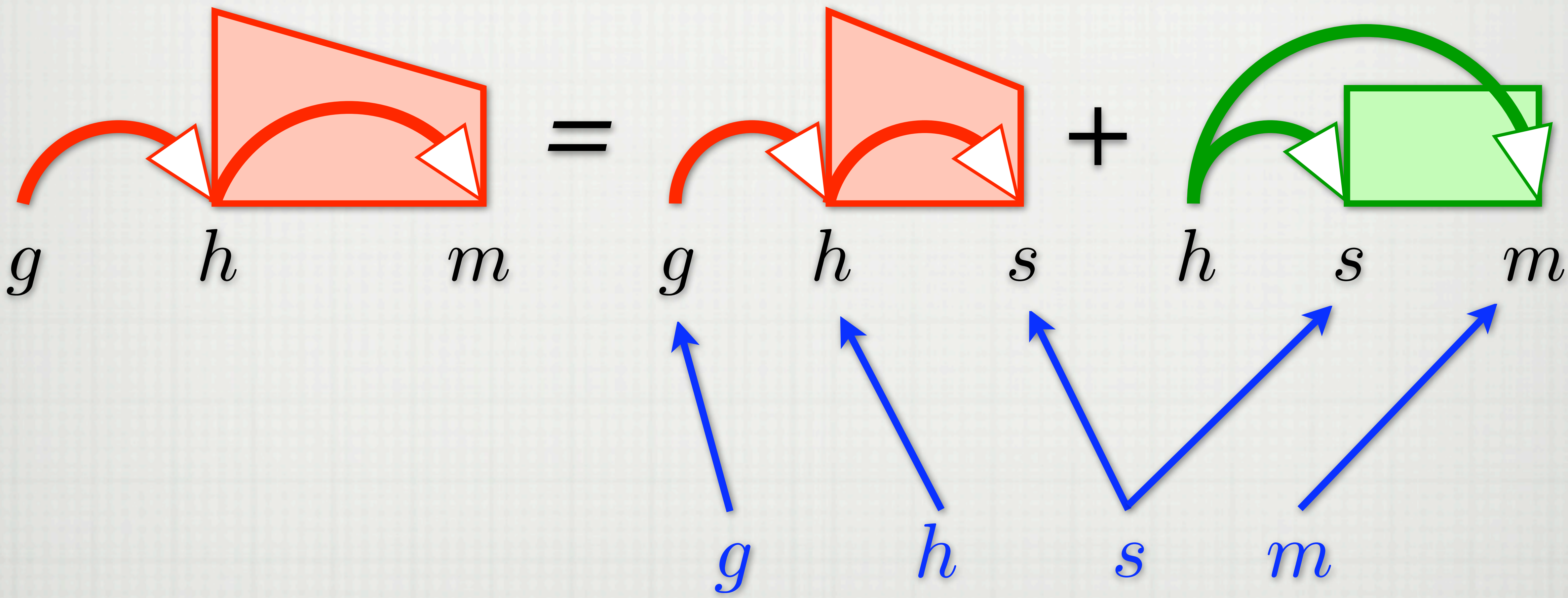
Sibling G-Span

A pair of adjacent modifiers with their shared head



Model 1: Grand-Sibling Scores

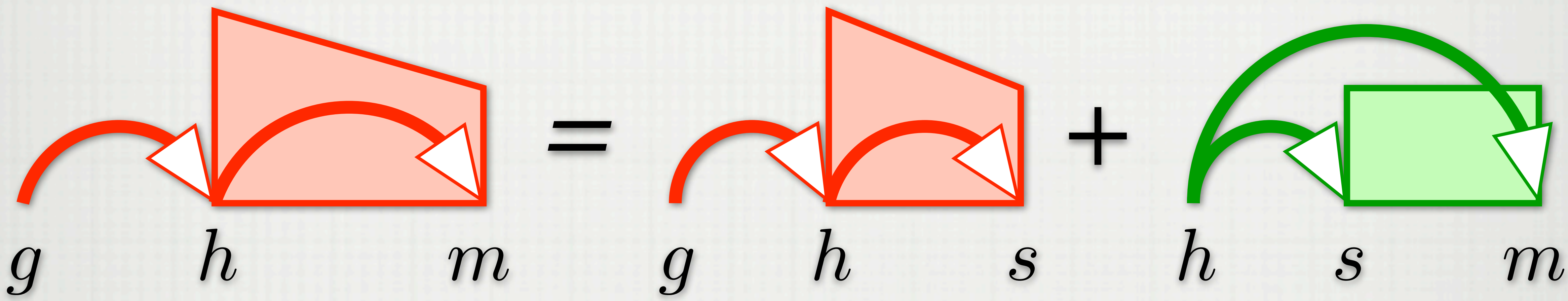
- Model 1, grand-siblings: $O(n^4)$

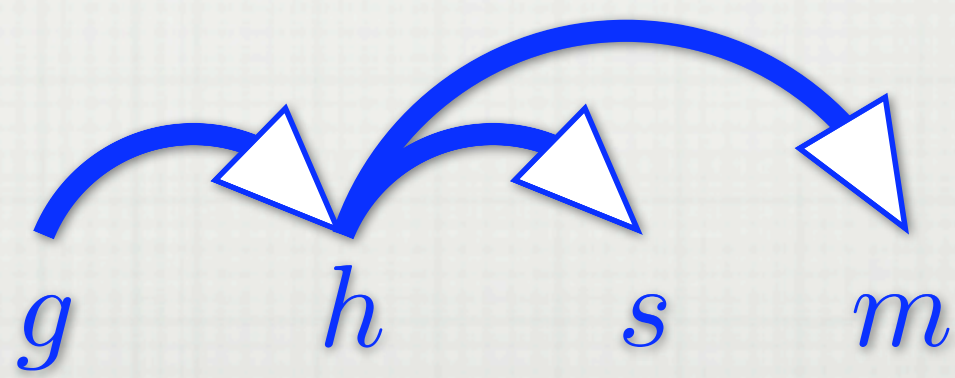


- Scores grand-sibling interactions

Model 1: Grand-Sibling Scores

- Model 1, grand-siblings: $O(n^4)$

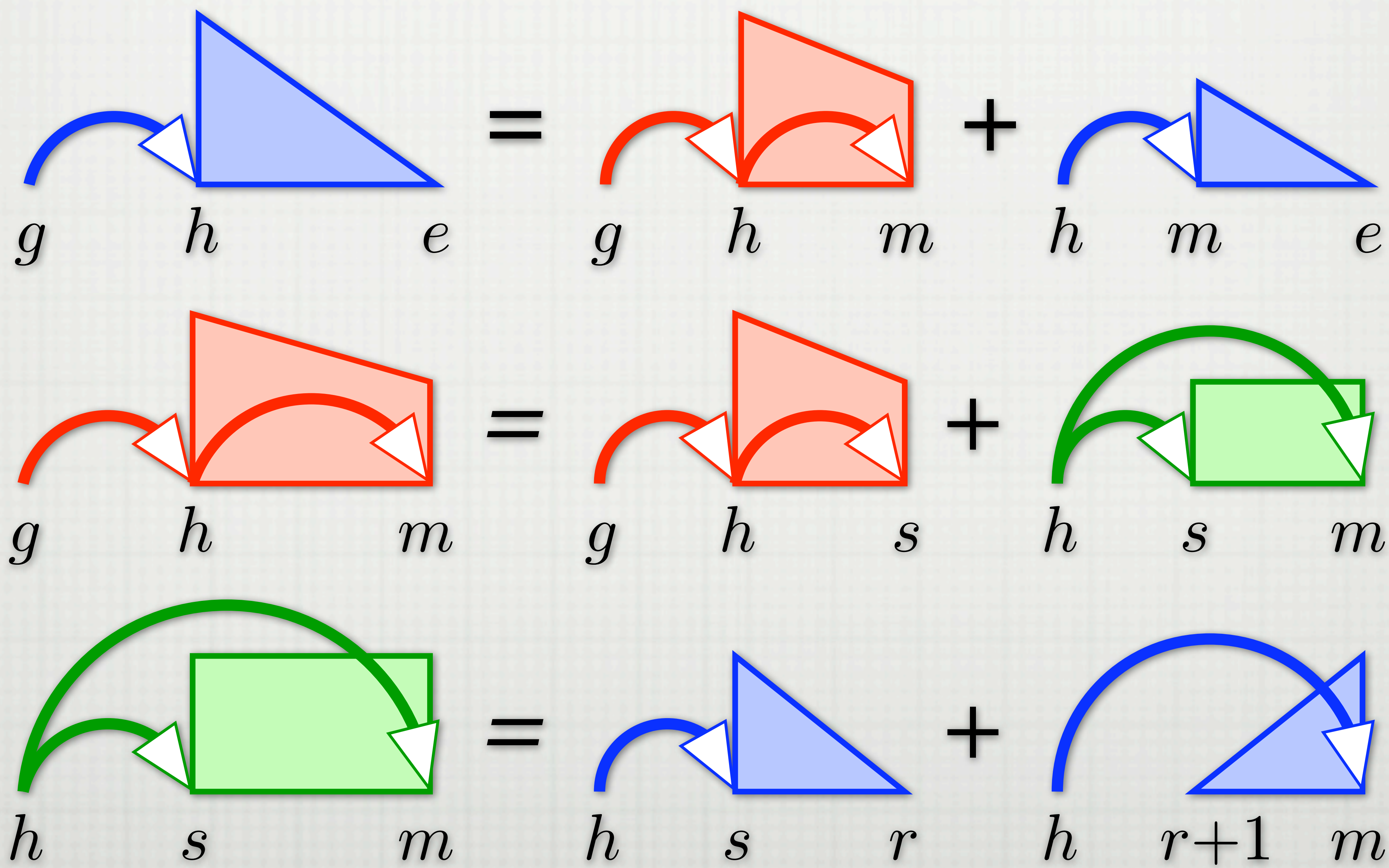




- Scores grand-sibling interactions

Model 1: Derivations

- Model 1, grand-siblings: $O(n^4)$

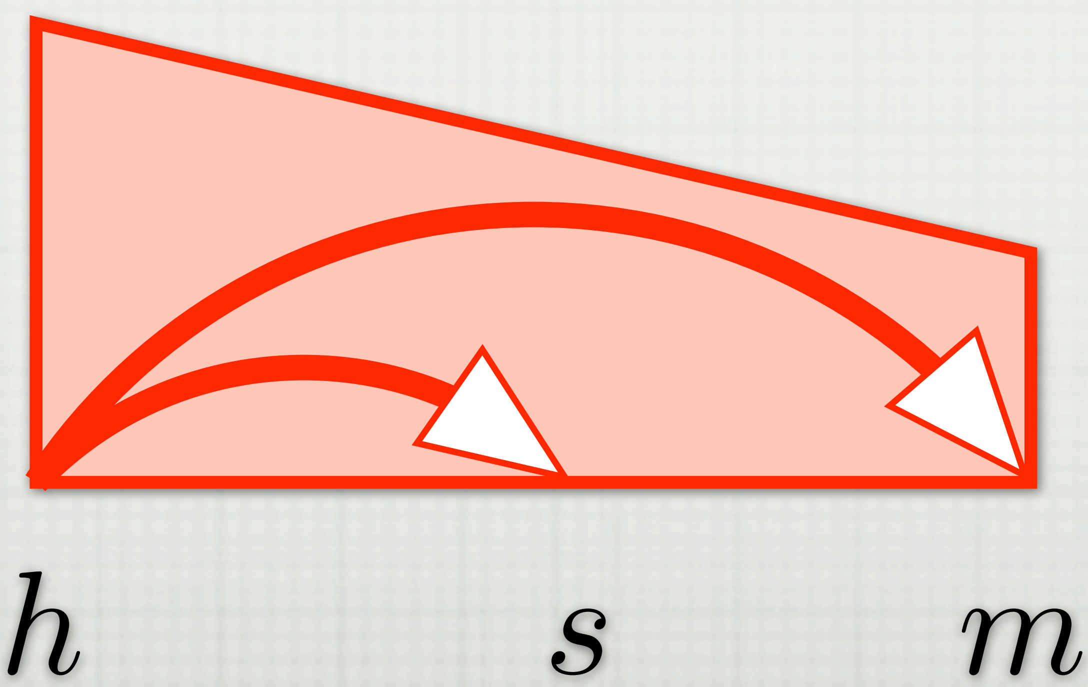


Model 2

- Model 2, grand-siblings and tri-siblings: $O(n^4)$
  - Introduce a fourth type of span:

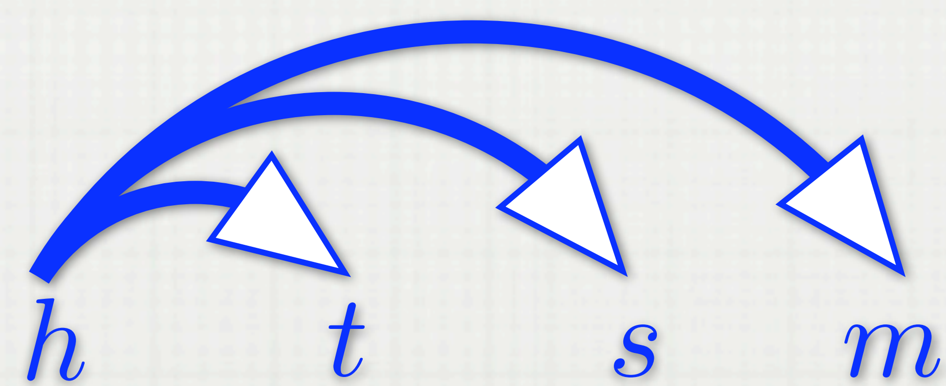
Incomplete S-Span

A dependency with its next-inner modifier

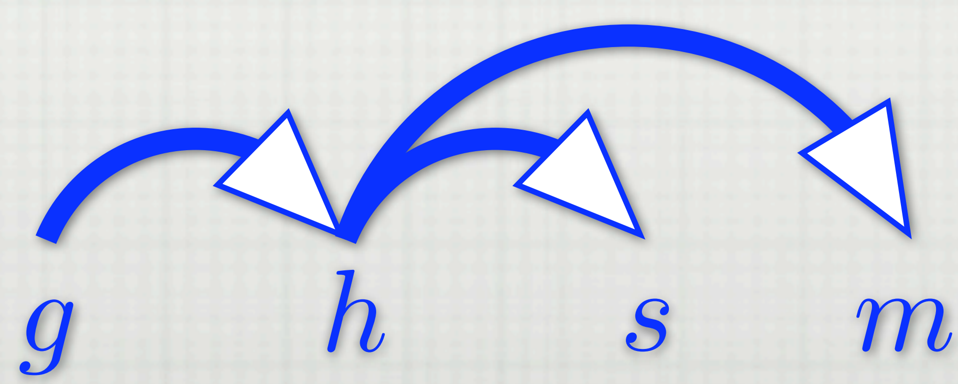


Model 2

- Model 2, grand-siblings and tri-siblings: $O(n^4)$
  - Capable of recovering all tri-siblings:



  - And some grand-siblings:



Summary of Parsing Algorithms

- Model 0 parses an all-grandchildren factorization
- Model 1 parses an all-grand-siblings factorization
- Model 2 parses all-tri-siblings and some grand-siblings
- All parsers require $O(n^4)$ time and $O(n^3)$ space
  - Identical to Carreras (2007) second-order
- Models 1 and 2 have *optimal* runtime
  - Total number of third-order parts: $O(n^4)$

English and Czech Parsing

Parser	*English*	*Czech*
McDonald (2006)	91.5	85.2
Second-order, Baseline	92.0	86.1
Model 1	93.0	87.4
Model 2	92.9	87.4
Second-order, Clusters	93.2	87.1

- Attachment score on the English and Czech test sets
- Third-order comparable to *semi-supervised*

Summary

- Third-order factorizations can be parsed in $O(n^4)$
- Third-parsers work well in practice
- Possible extensions:
  - Recovering word senses or dependency labels
  - Increasing context to fourth-order or more
  - Using cluster-based features

Outline

- Introduction
- Three advances in discriminative dependency parsing:

$$\underset{y \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \sum_{p \in y} \mathbf{w} \cdot \phi(\mathbf{x}, p)$$

  - Simple and effective lexical representations
  - Parameter estimation for non-projective parsing
  - Efficient third-order dependency parsers
- *Conclusion*

Conclusions

- Dependency parsing is a simple and effective framework for syntactic analysis
- Structured linear models provide three opportunities for improvements
  - Feature representations
  - Parameter estimation
  - Factorization