CSCI 544: Applied Natural Language Processing

Syntactic Parsing

Xuezhe Ma (Max)

USC University of Southern California

Course Organization

NLP Tasks

Models

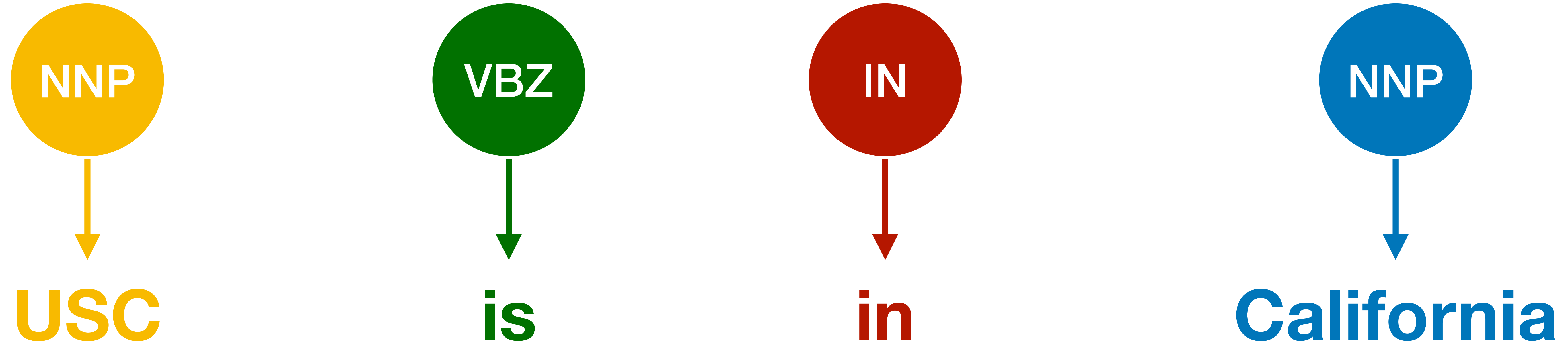
	Text Classification	Sequence Labeling	Syntactic Parsing	Seq2seq & MT	Language Modeling & Generation
	Naive Bayes Logistic Regression SVM …	HMMs MEMMs CRFs	Tree-structured Models/Algorithms	……	……
	Word Embedding & Neural Networks	RNN LSTM	Neural Structured Networks	Attention & Transformers	Generative Models & LLMs

Recap: Sequence Labeling?

A type of structured prediction tasks

$$Y = < y_i, y_2, \ldots, y_n >$$

$$X = < x_i, x_2, \ldots, x_n >$$



Assigning each token of X, e.g. x_i a corresponding label y_i

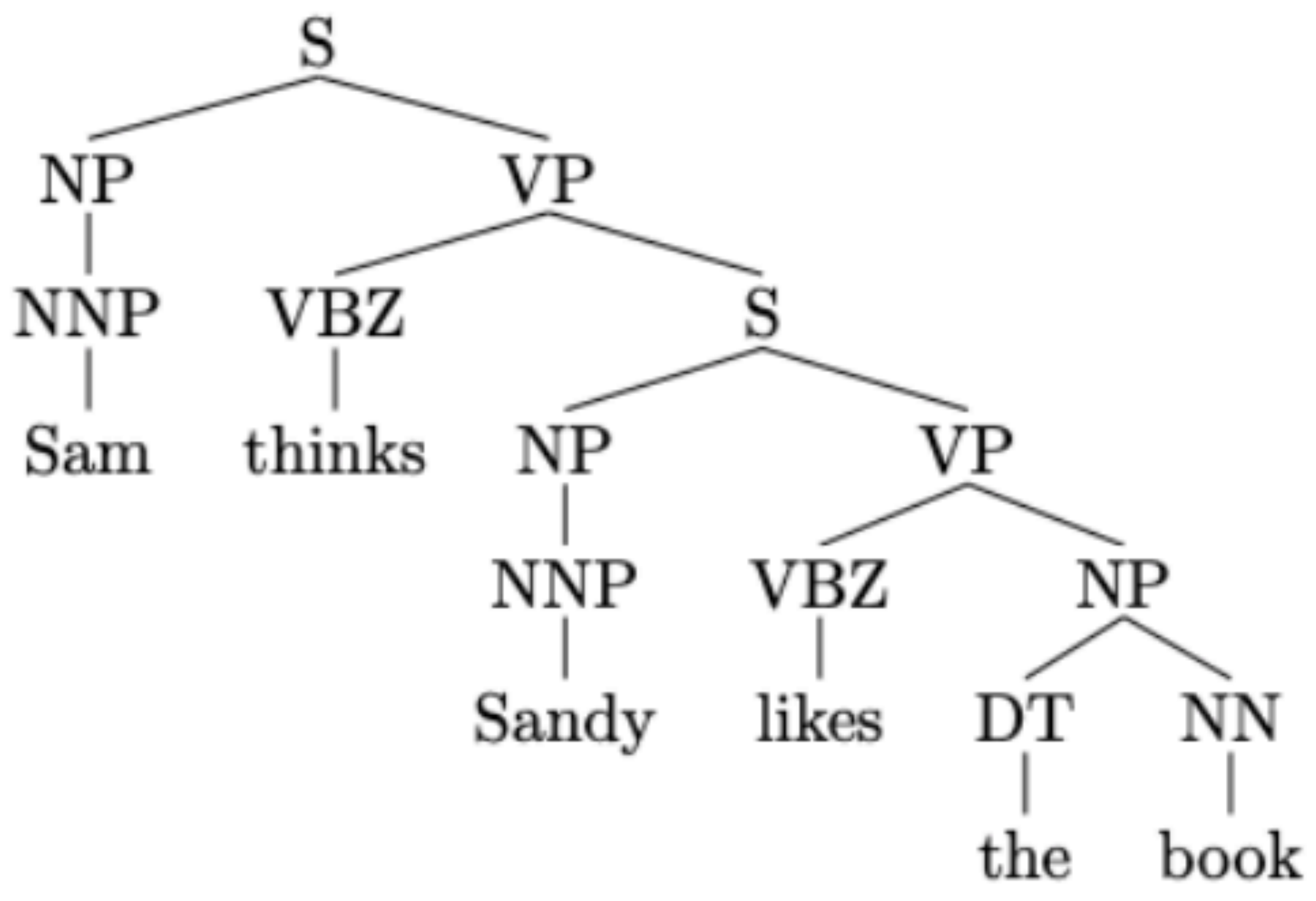
Syntactic Structure: Constituency vs. Dependency

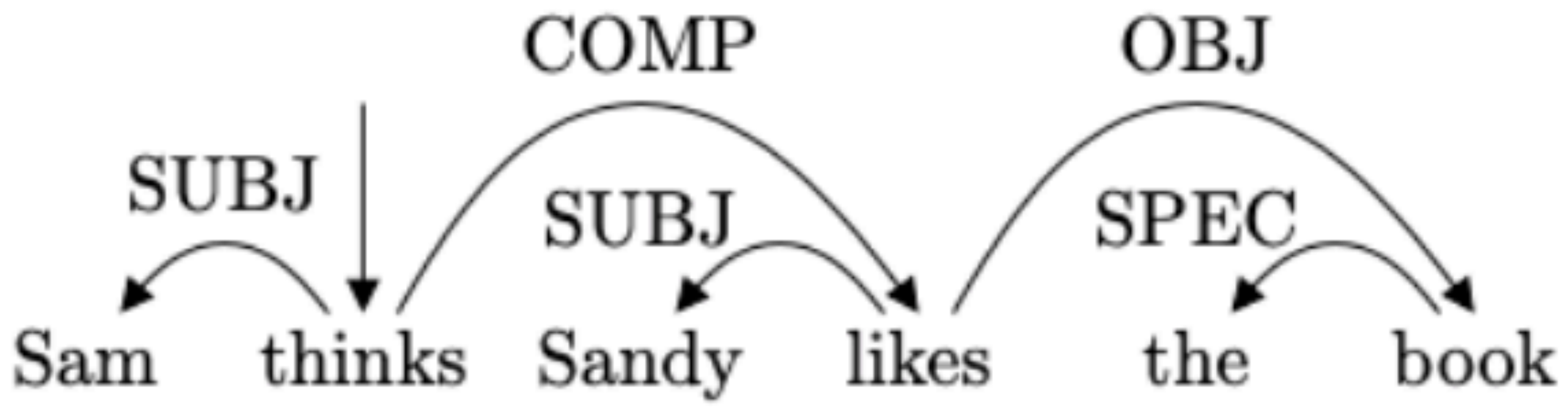
Theme: How to represent the structure of sentences using (syntax) trees?

Two views of linguistic structures

- **Constituency**
  - = phrase structure grammar
  - Based on context-free grammars (CFGs)

- **Dependency**
  - = phrase structure grammar

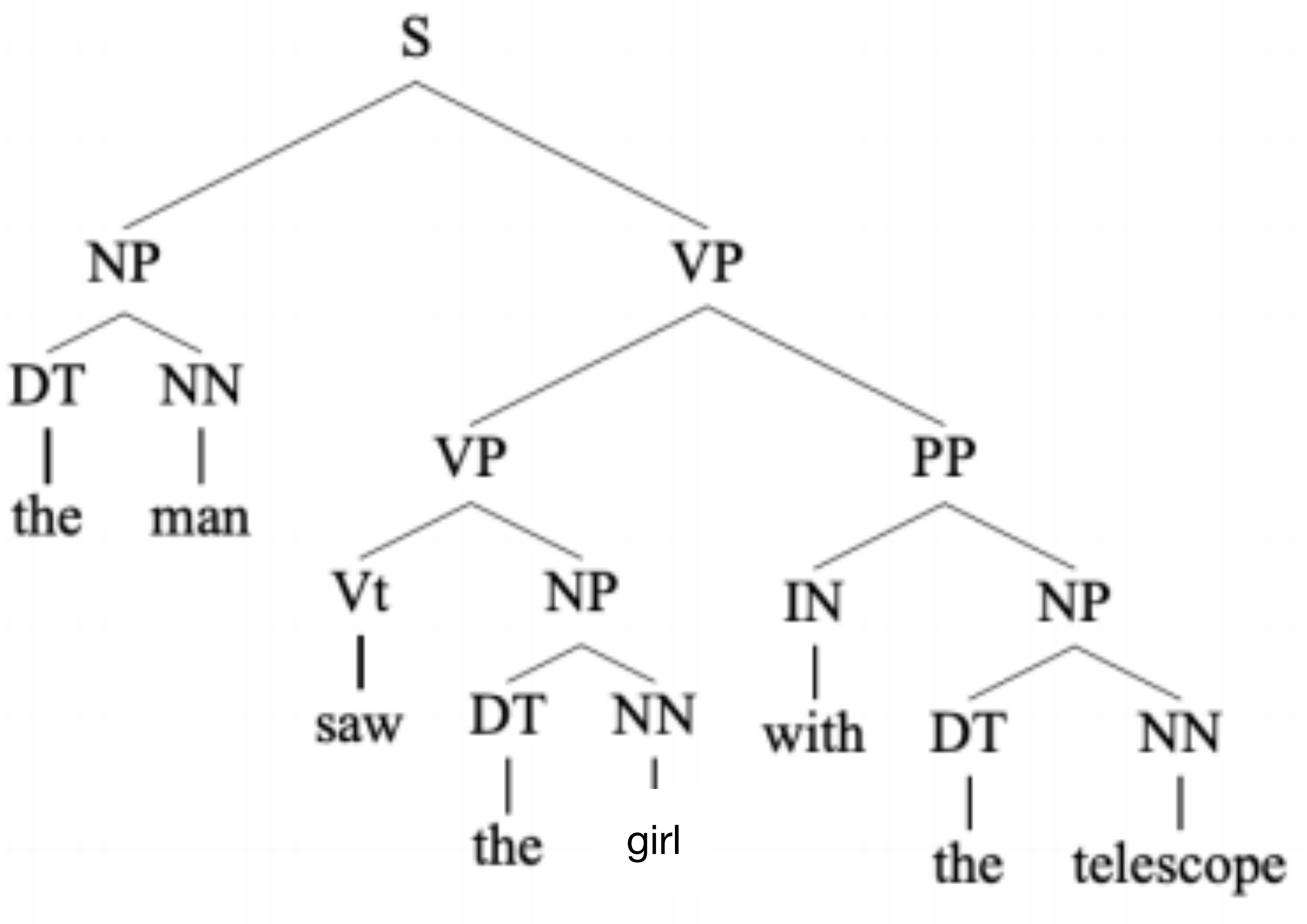


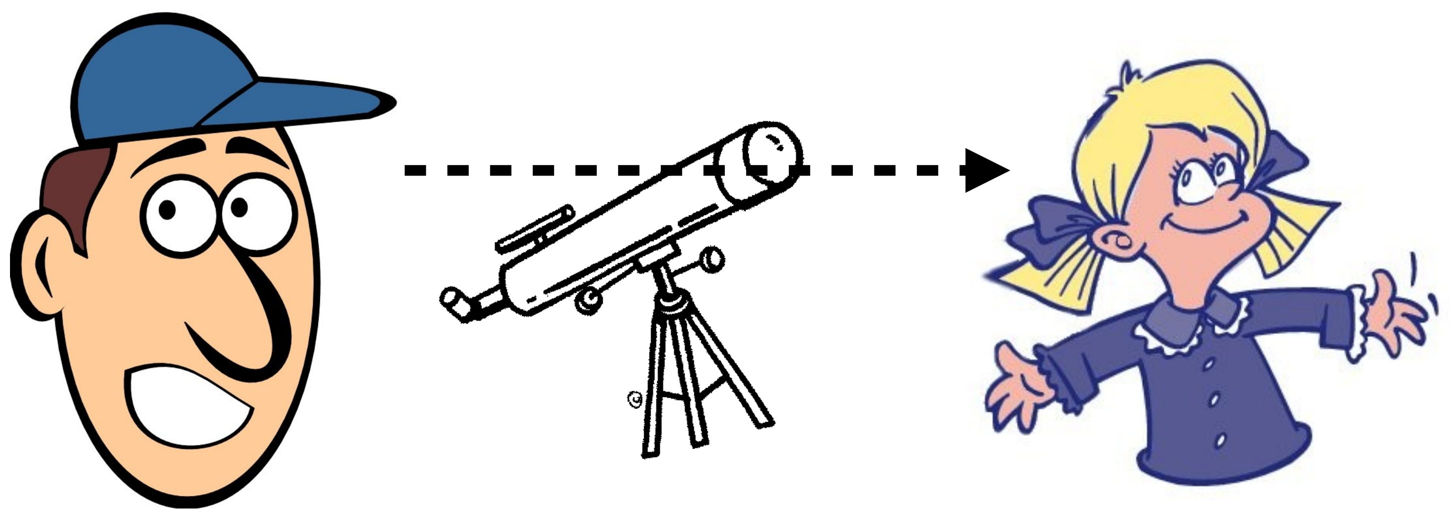


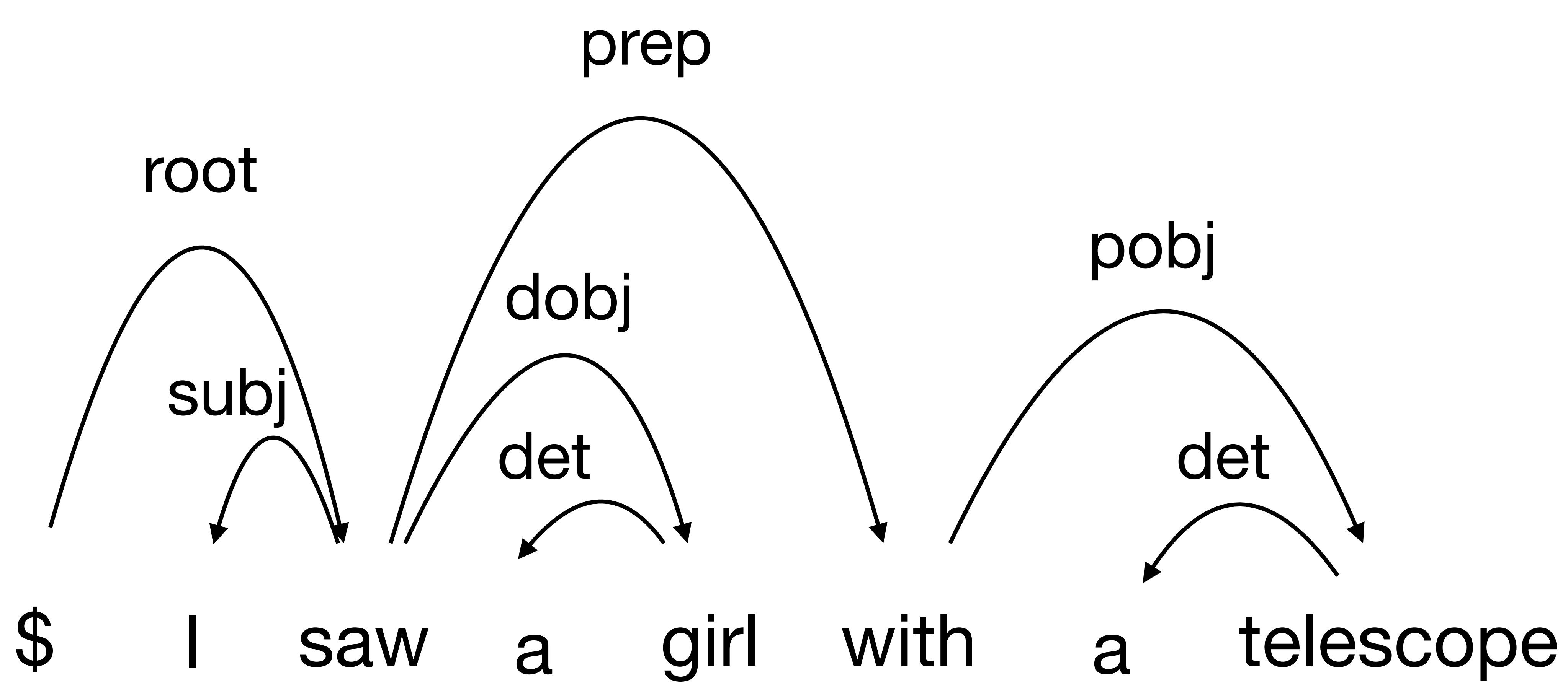
Slide credit: Danqi Chen COS 484/584

Constituency vs. Dependency

The man saw the girl with the telescope



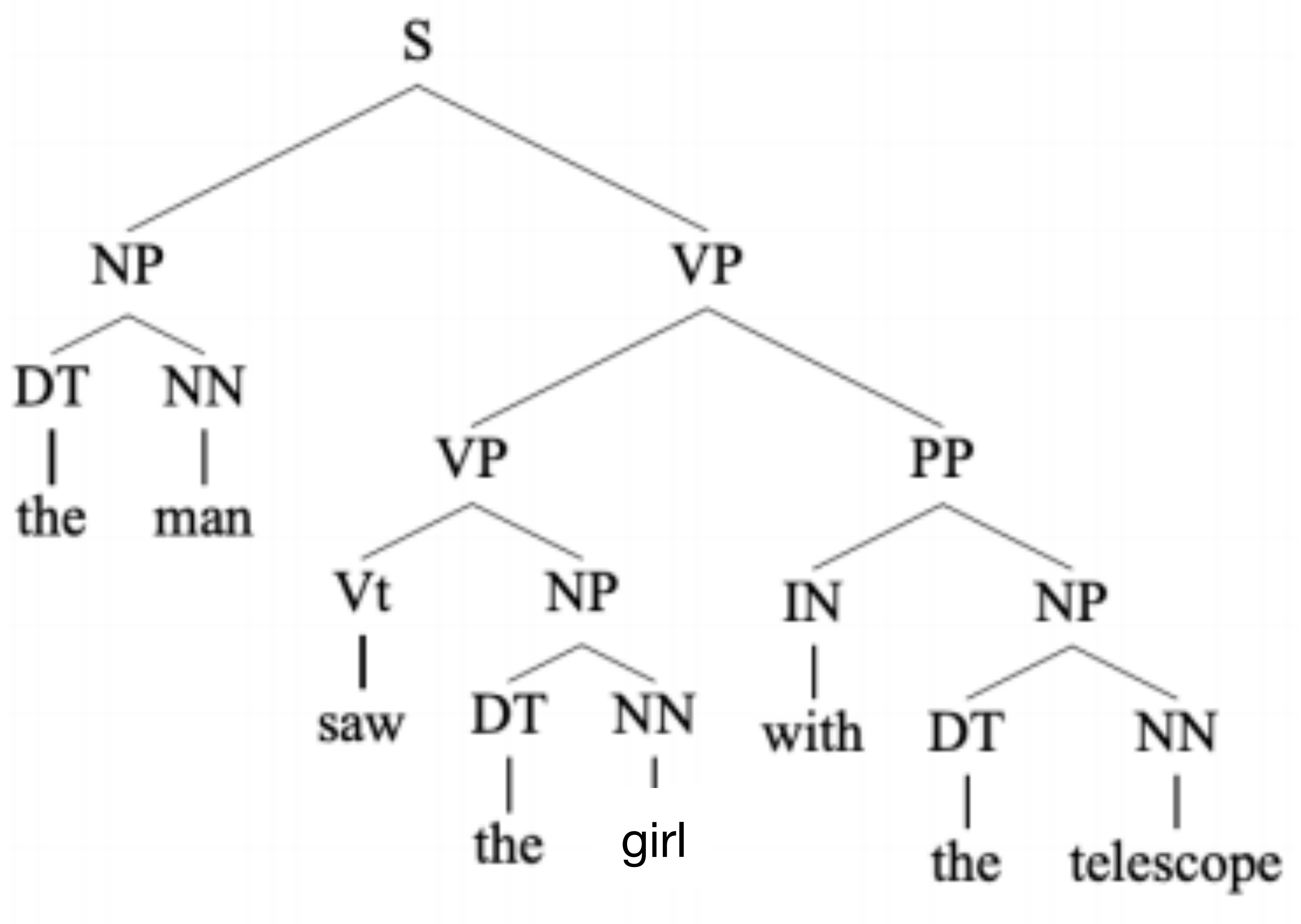




Constituency Structure

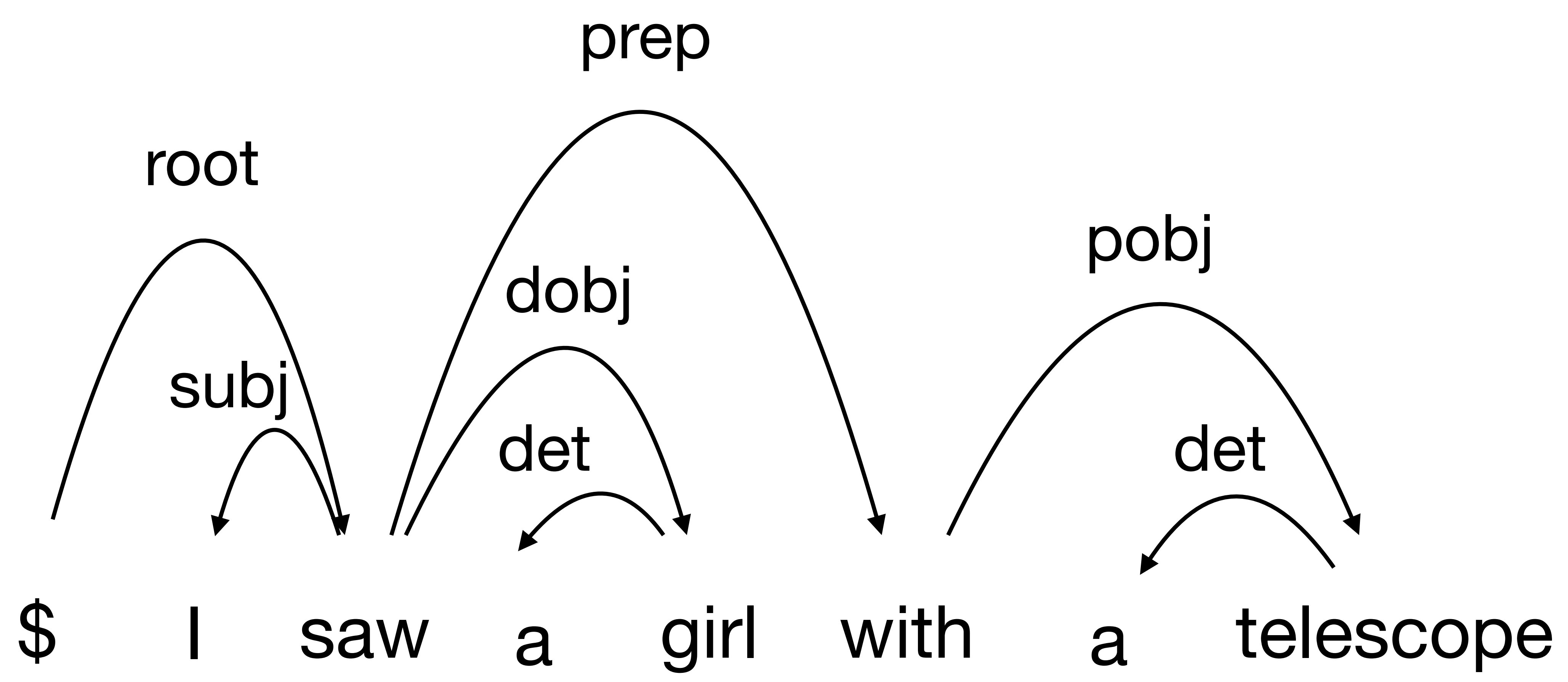
- **Starting units:** words are given a category: part-of-speech tags
  - N = noun, V = verb, DT = determiner
- **Phrases:** words combine into phrases with categories
  - NP = noun phrase, VP = verb phrase, S = sentence
  - Phrases can combine into bigger phrases recursively

The man saw the girl with the telescope



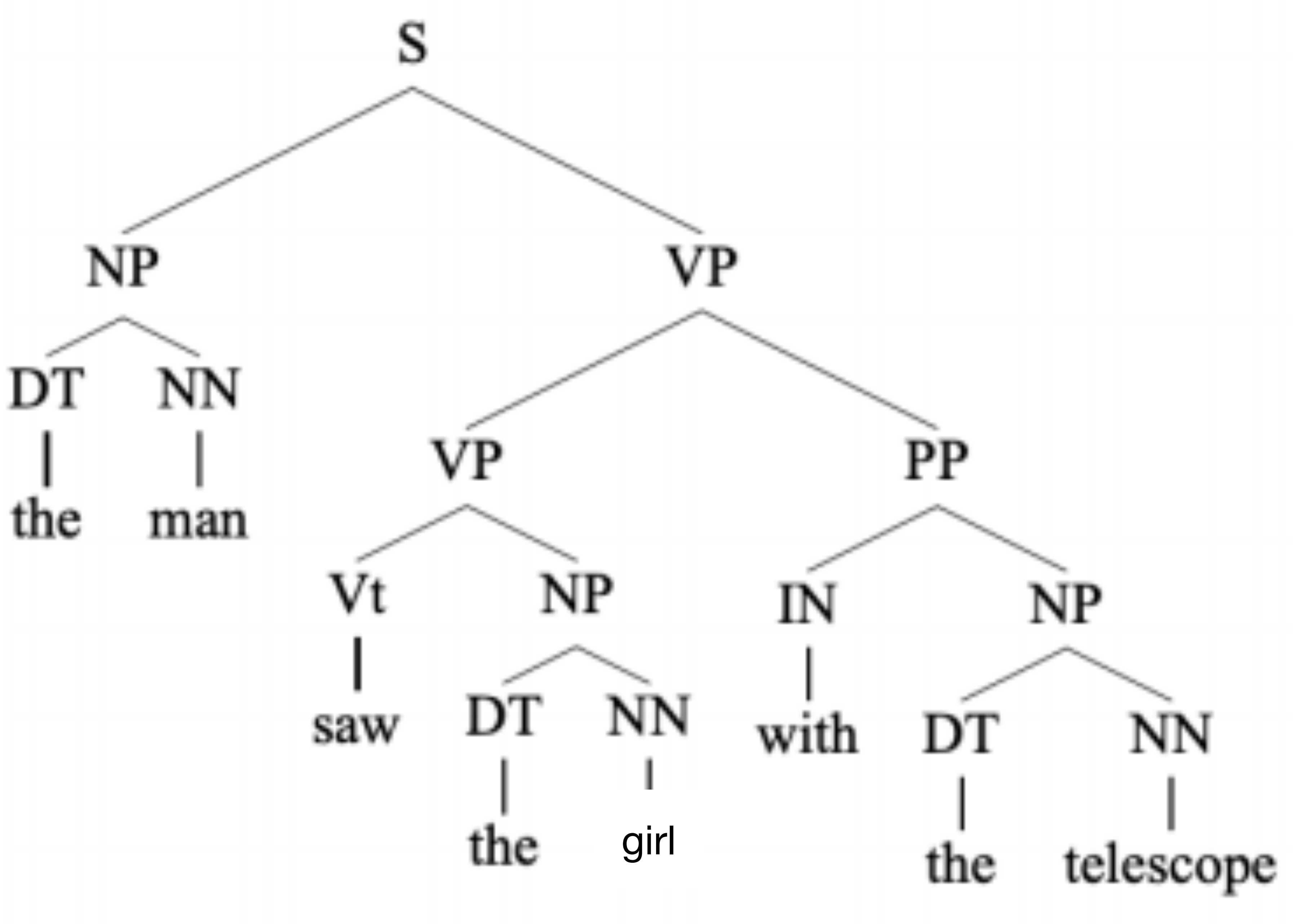
Dependency Structure

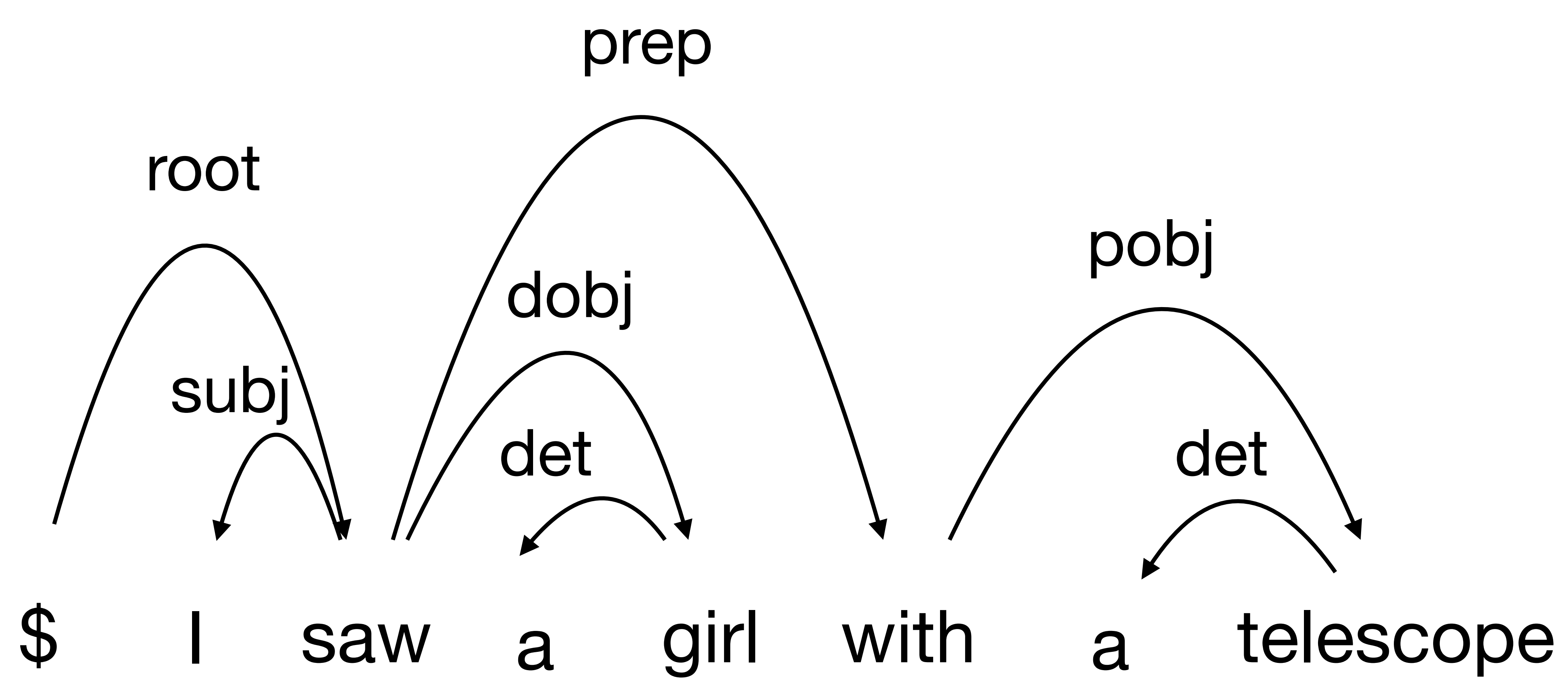
- **The basic idea:**
  - Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.
- **In the words of Lucien Tesniere** [Tesniere1959]:
  - The sentence is an organized whole, the constituent elements of which are *words* [1.2]. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives *connection*, the totality of which forms the structure of the sentence [1.3]. The structural connections establish *dependency* relations between the words. Each connection in principle unites a *superior* term and an *inferior* term [2.1]. The superior term receives the name *governor*, and the inferior term receives the name *subordinate*.



Constituency vs. Dependency

- **Dependency structures explicitly represent**
  - Head-dependent relations (directed arcs)
  - Functional categories (arc labels)
- **Constituent structures explicitly represent**
  - Phrases (non-terminal nodes)
  - Structural categories (non-terminal symbols)





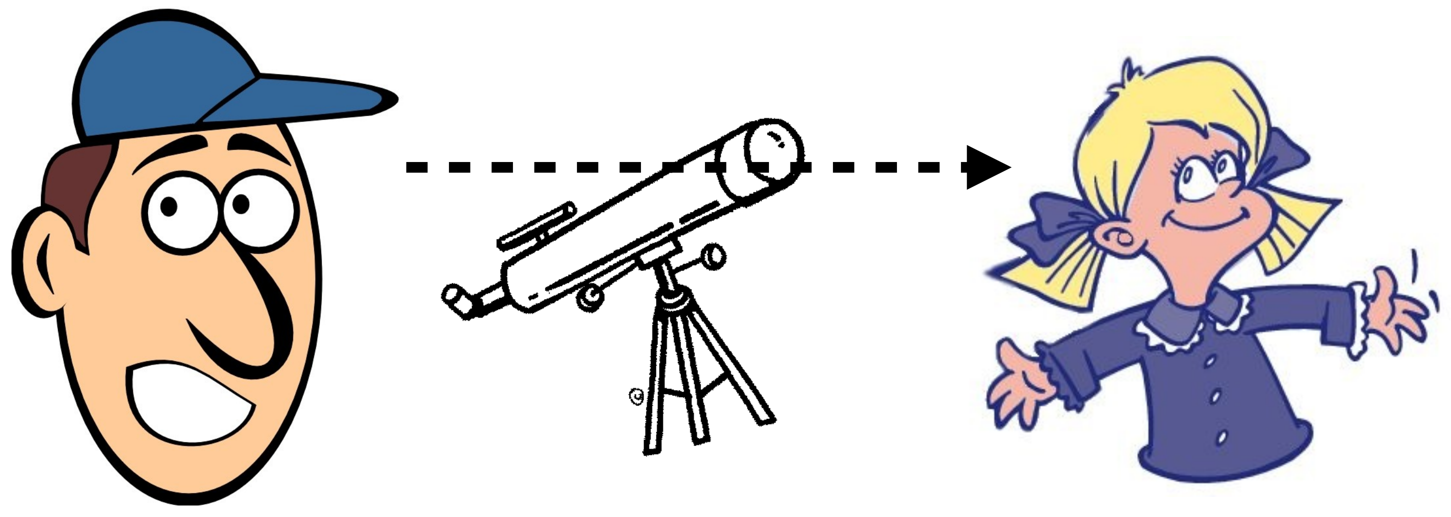
Some Theoretical Frameworks

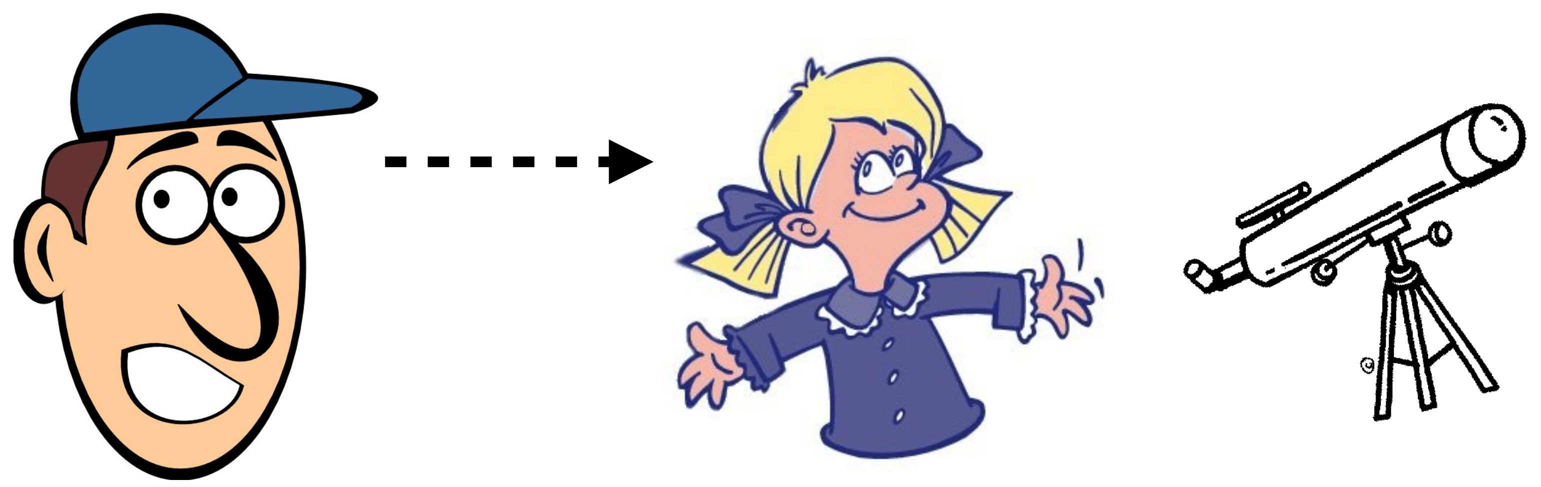
- Word Grammar (WG) [Hudson 1984, Hudson 1990, Hudson 2007]
- Functional Generative Description (FGD) [Sgall et al. 1986]
- Dependency Unification Grammar (DUG)
  [Hellwig 1986, Hellwig 2003]
- Meaning-Text Theory (MTT) [Mel'čuk 1988, Milićević 2006]
- (Weighted) Constraint Dependency Grammar ([W]CDG)
  [Maruyama 1990, Menzel and Schröder 1998, Schröder 2002]
- Functional Dependency Grammar (FDG)
  [Tapanainen and Järvinen 1997, Järvinen and Tapanainen 1998]
- Topological/Extensible Dependency Grammar ([T/X]DG)
  [Duchier and Debusmann 2001, Debusmann et al. 2004]

Dependency Parsing

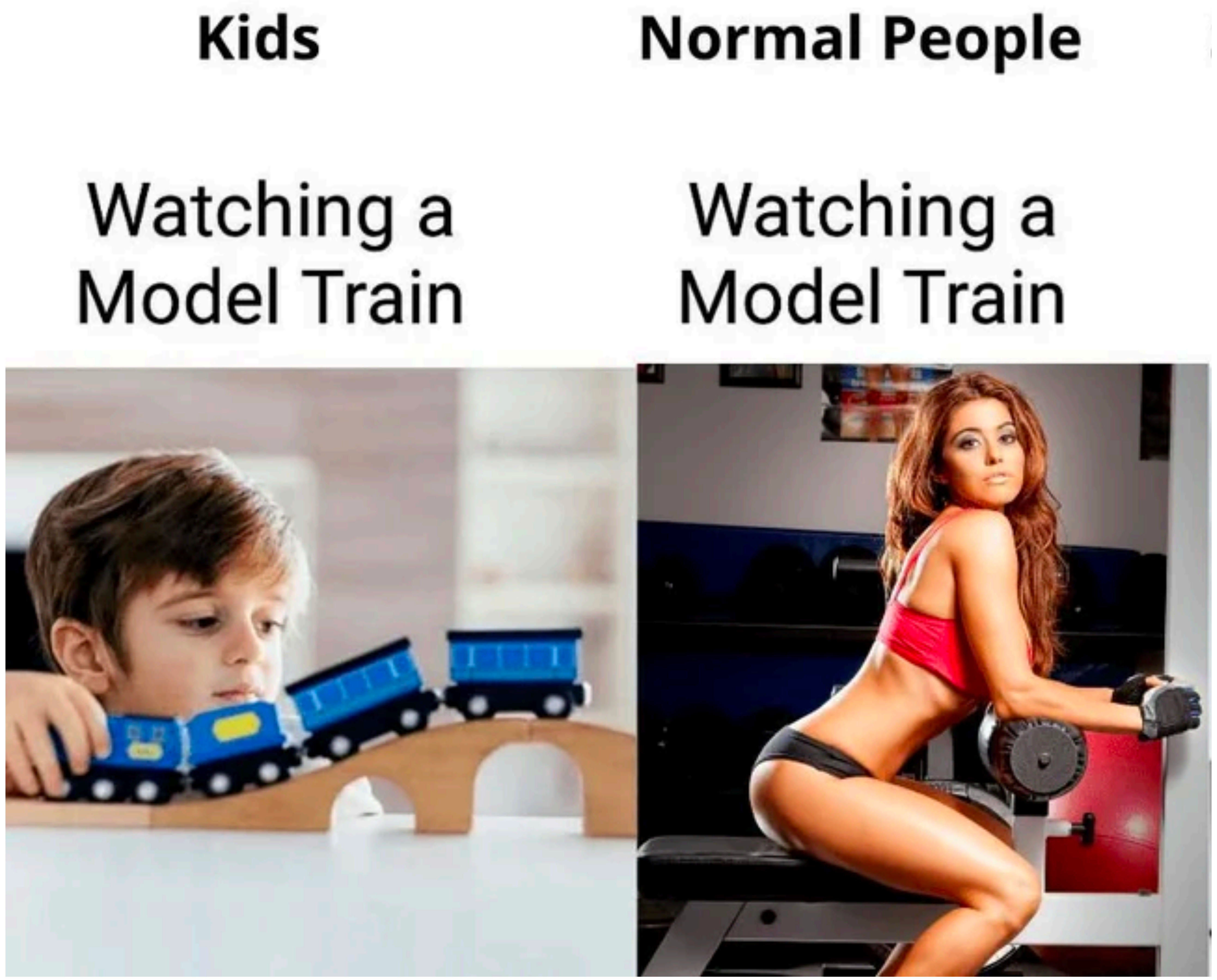
Why Syntactic Structures?

I saw a girl with a telescope

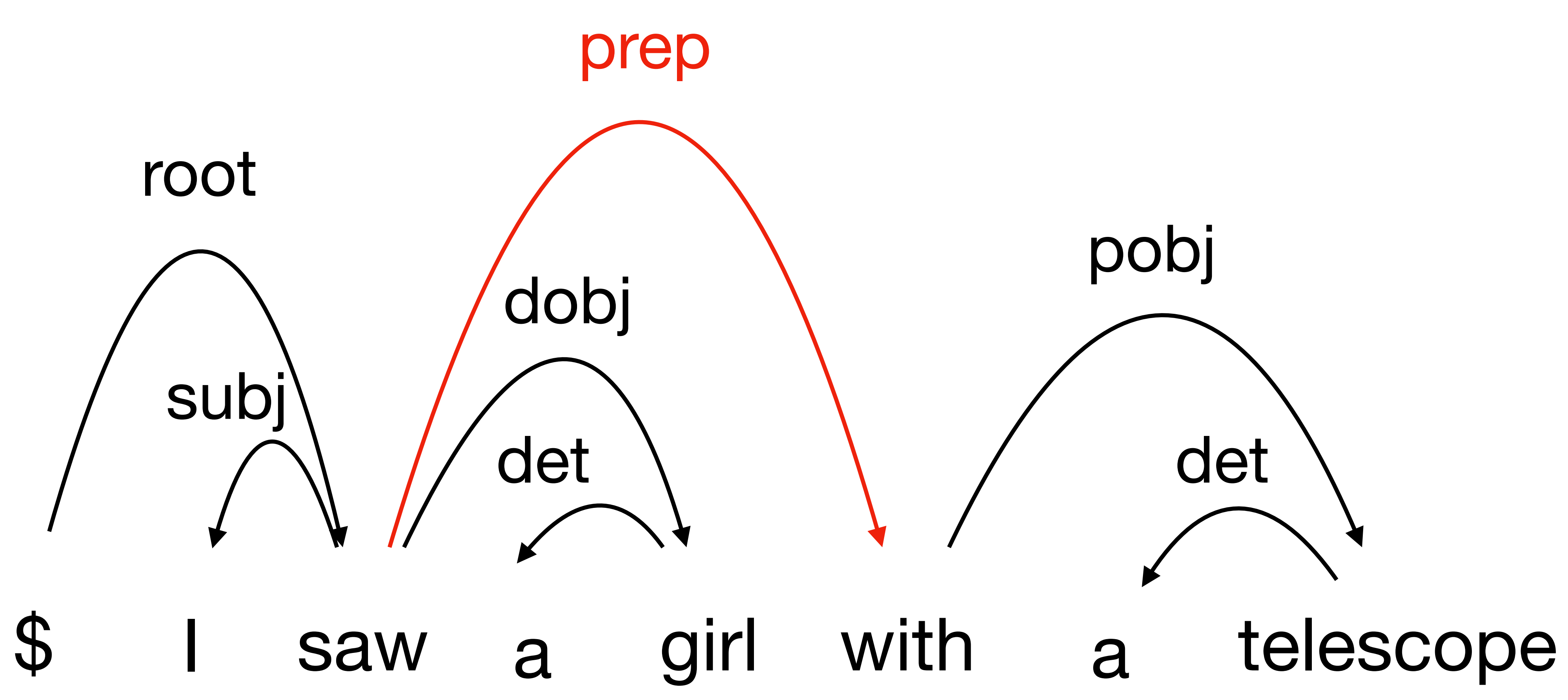


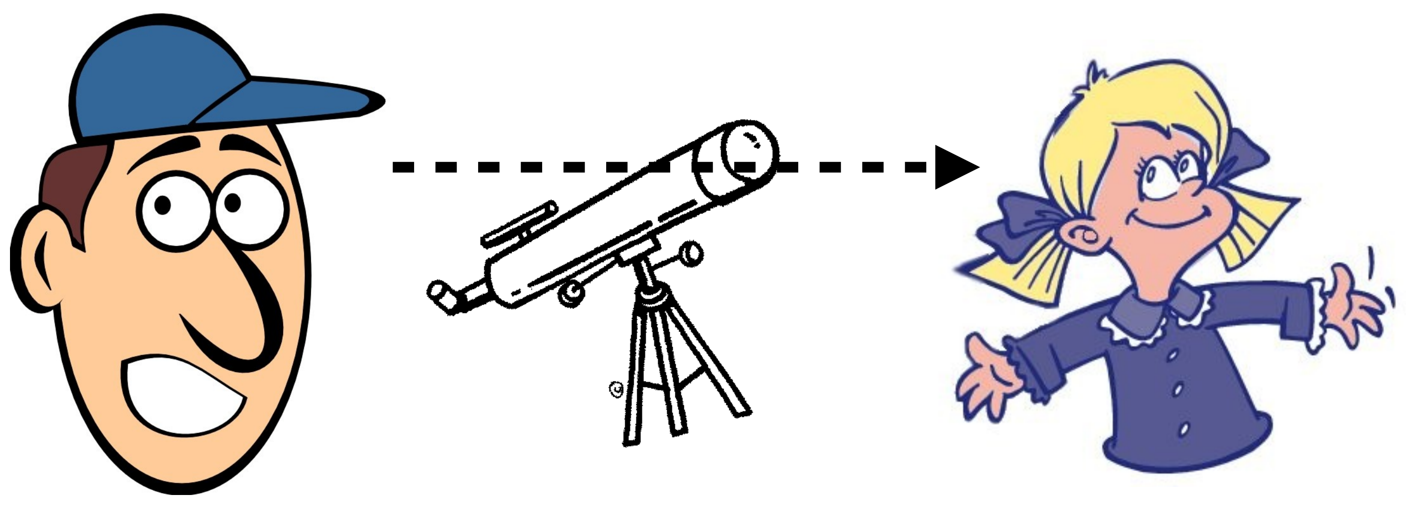


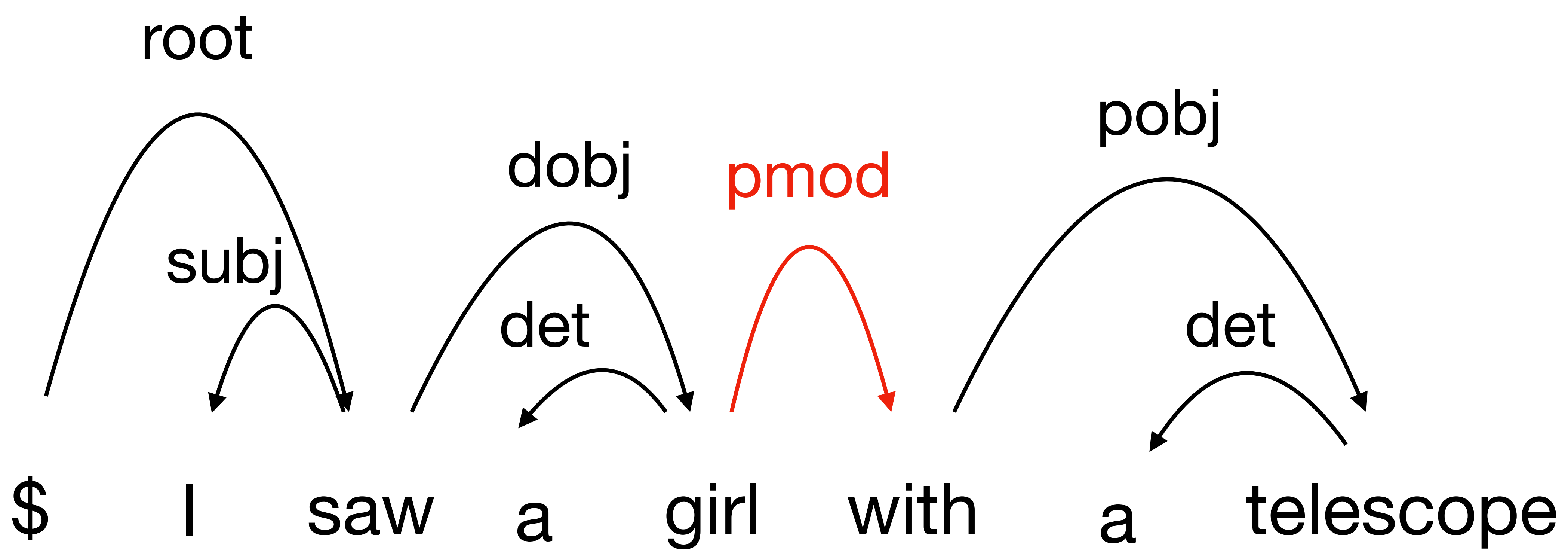
Why Syntactic Structures?

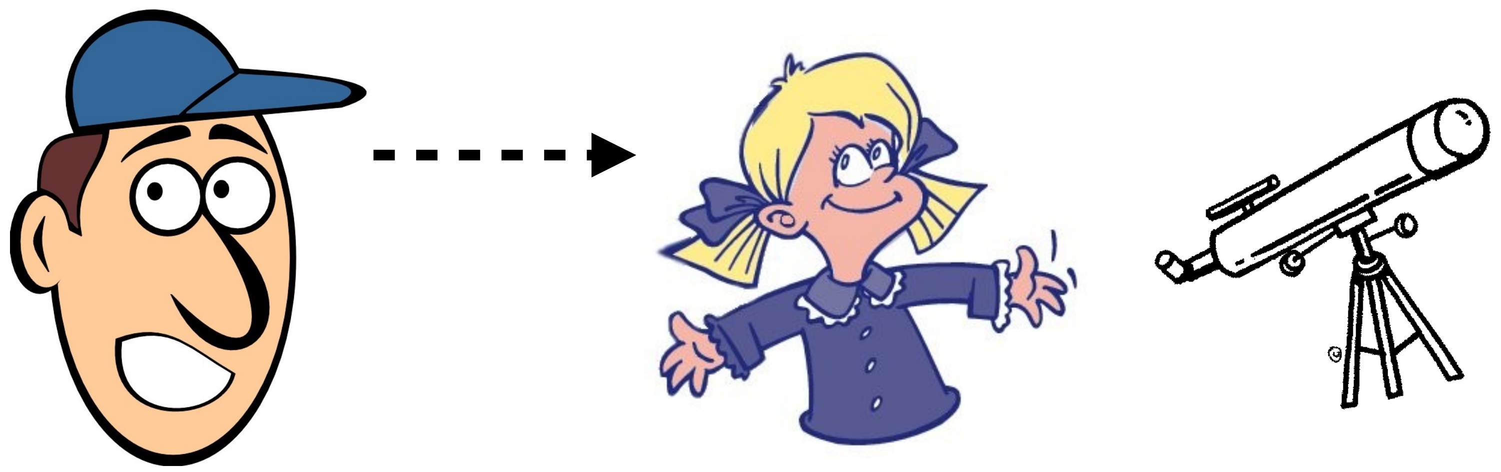


Syntactic Structures Resolve Ambiguity



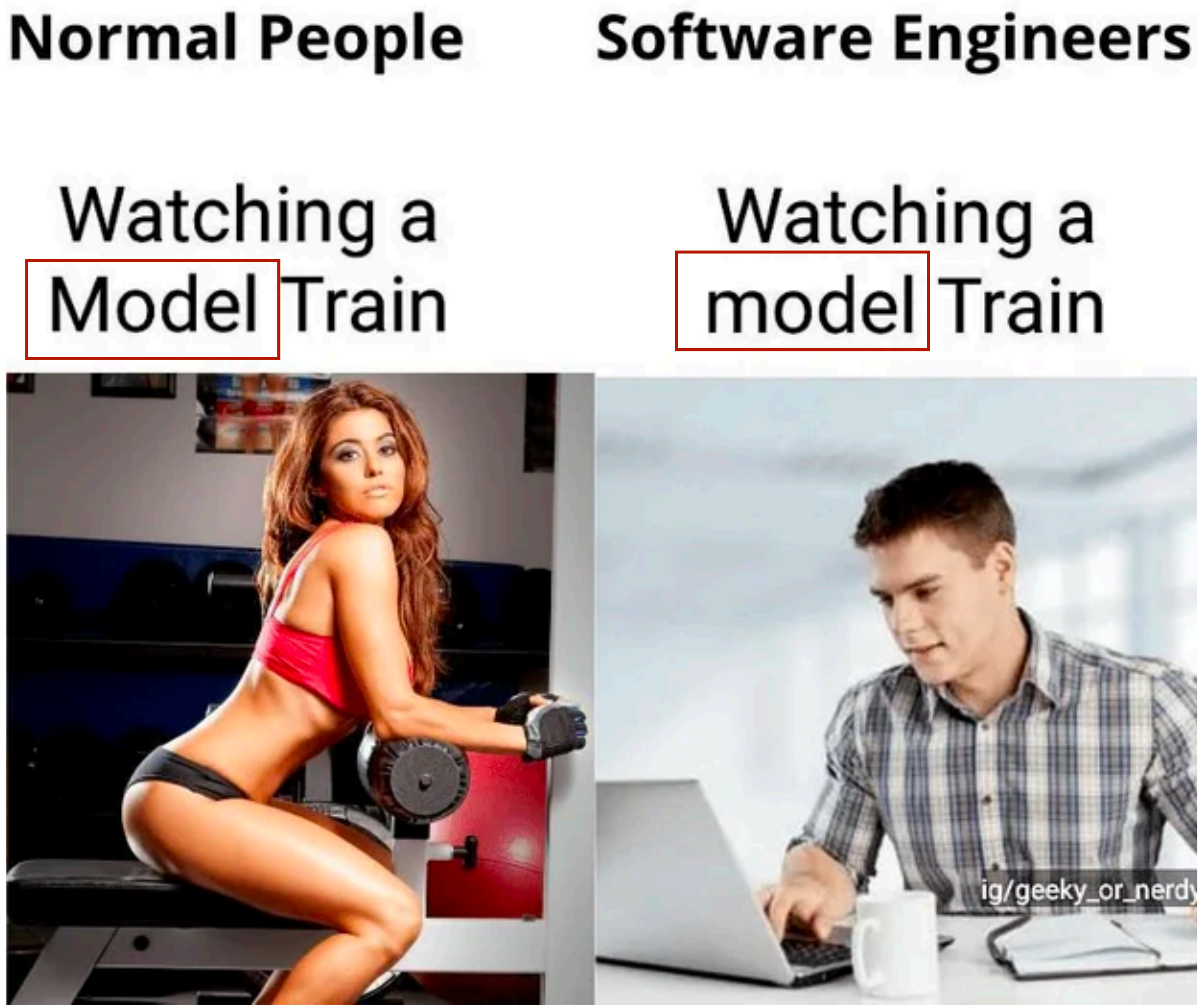






Limitation of Syntactic Structures

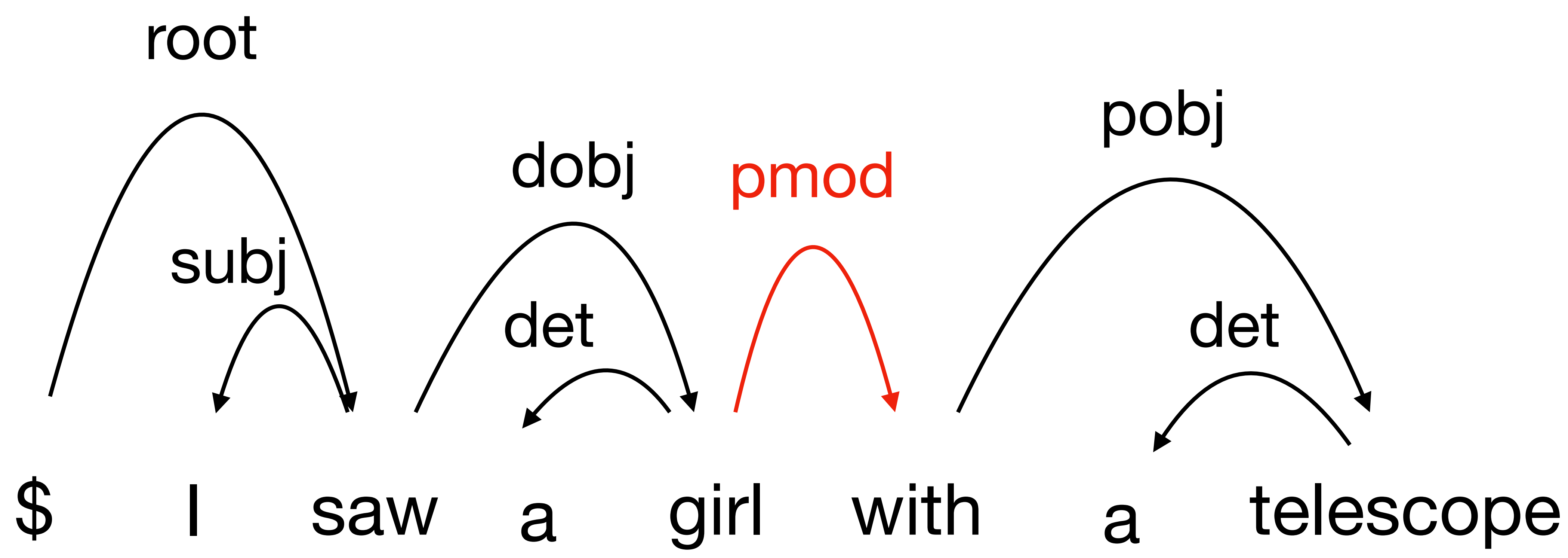
Syntax structures cannot resolve semantic ambiguities



The same syntactic structures
Different semantic meaning of “*model*”

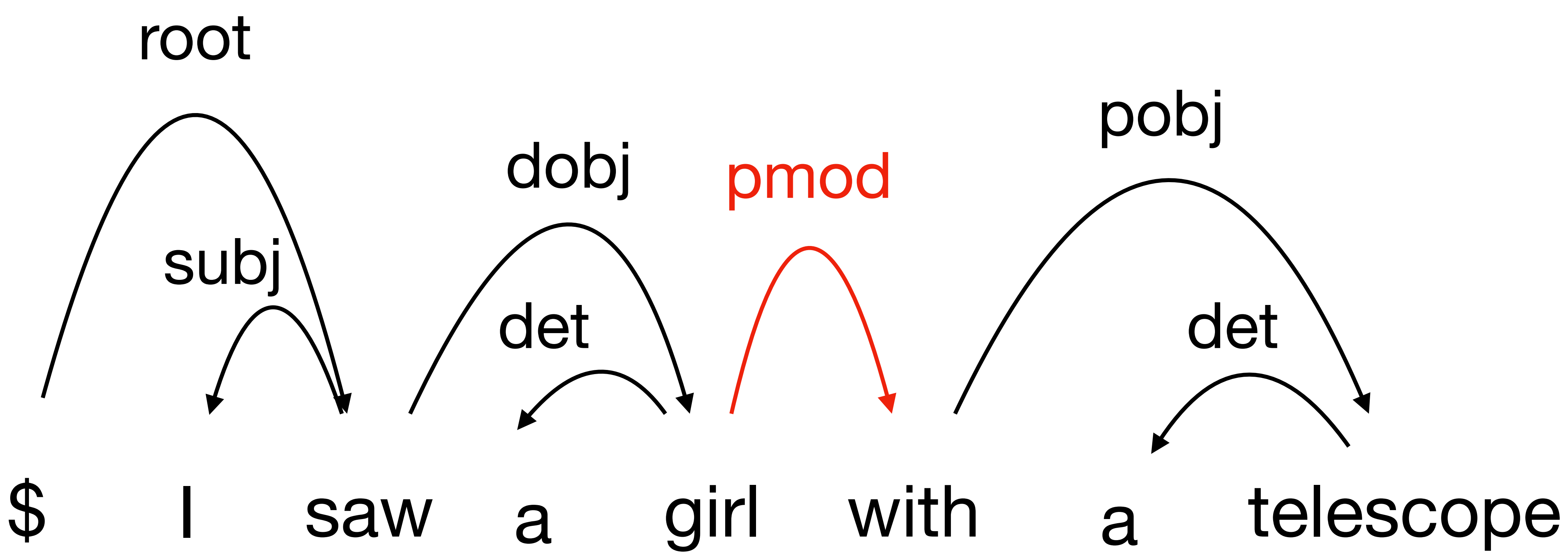
Terminology

Superior	**Inferior**
Head	Dependent
Governor	Modifier
Regent	Subordinate
⋮	⋮



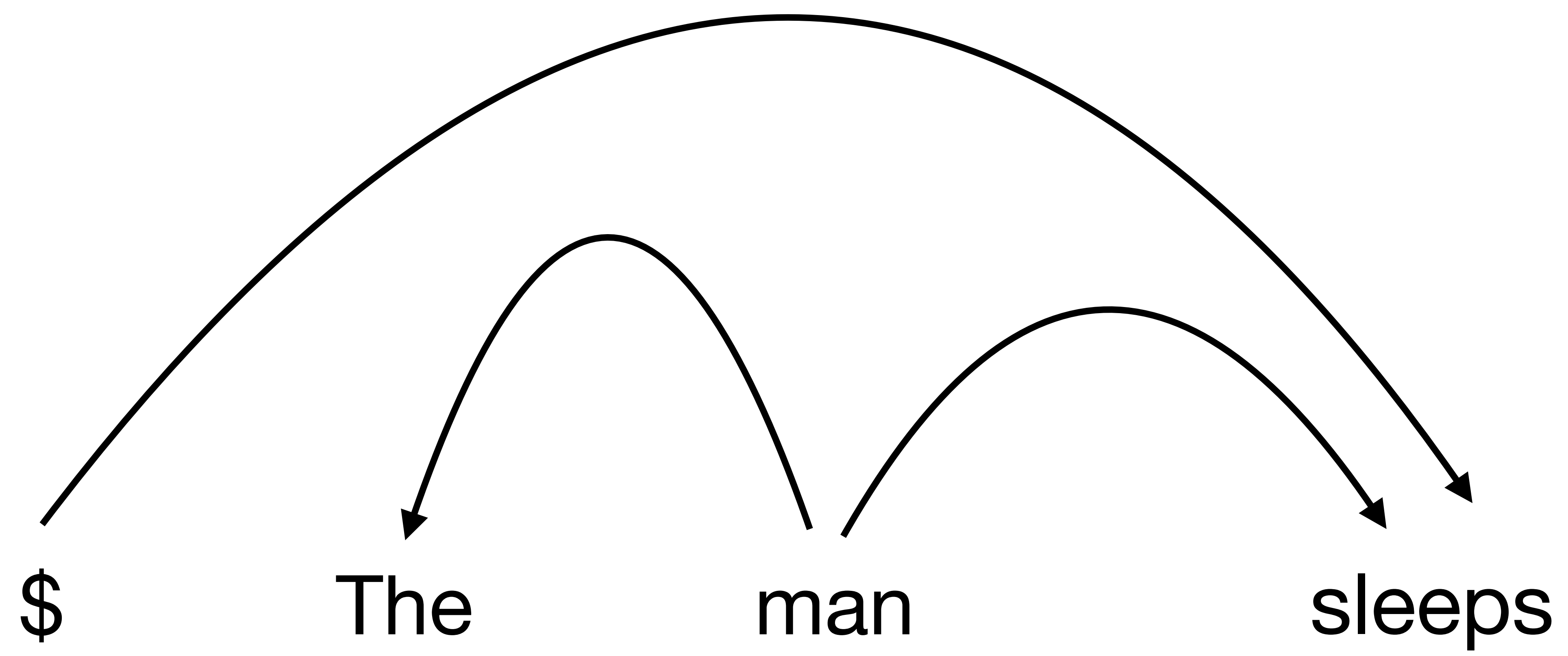
Terminology

Superior	Inferior
Head	Dependent
Governor	Modifier
Regent	Subordinate
⋮	⋮



A Formal Definition of Dependency Structures

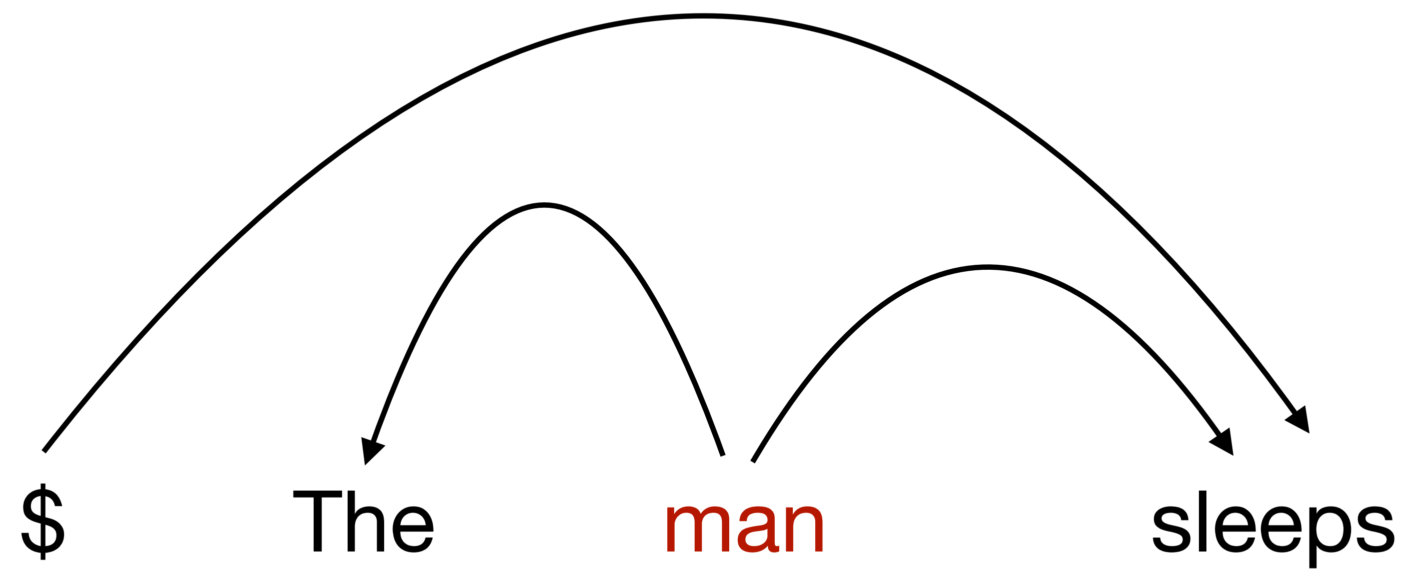
- **A dependency structure can be defined as a directed graph G, consisting of**
  - A set of nodes V
  - A set of directed arcs E (directed edges)
  - A linear precedence order $<$ on V (word order)



Is this directed graph a valid dependency structure?

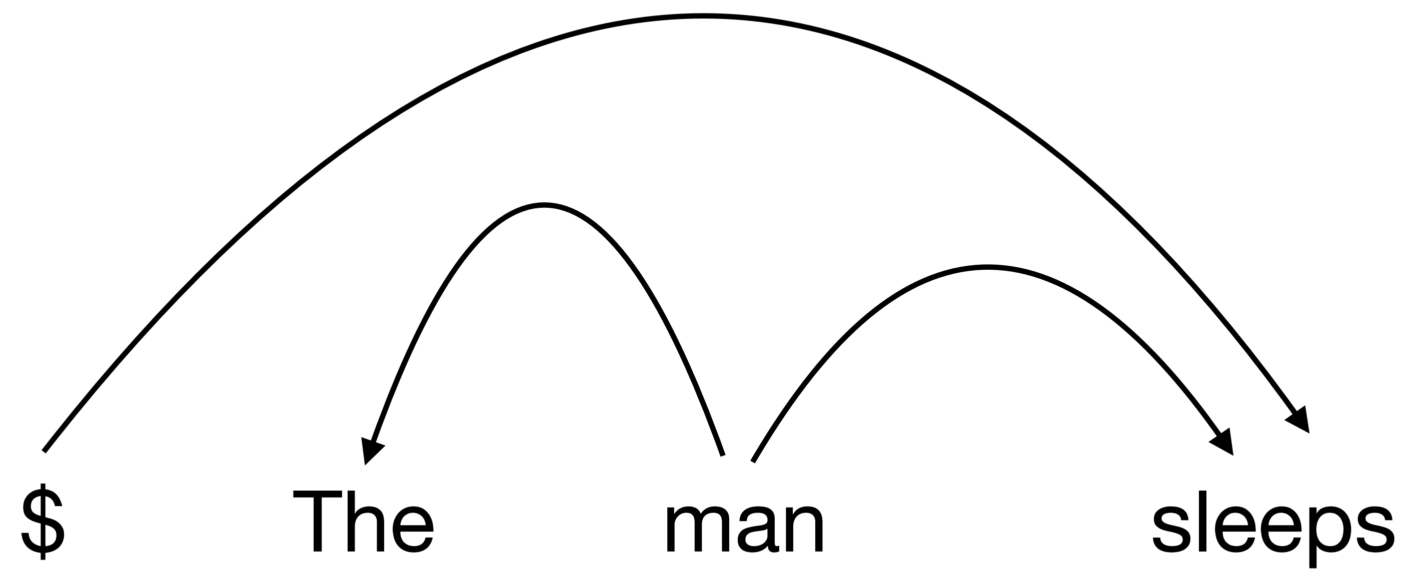
A Formal Definition of Dependency Structures

- **A dependency structure can be defined as a directed graph G, consisting of**
  - A set of V of nodes
  - A set of E of directed arcs (directed edges)
  - A linear precedence order $<$ on V (word order)
- **Formal Conditions of Dependency Structures**
  - G is connected: there exists a directed path from the root to every other node



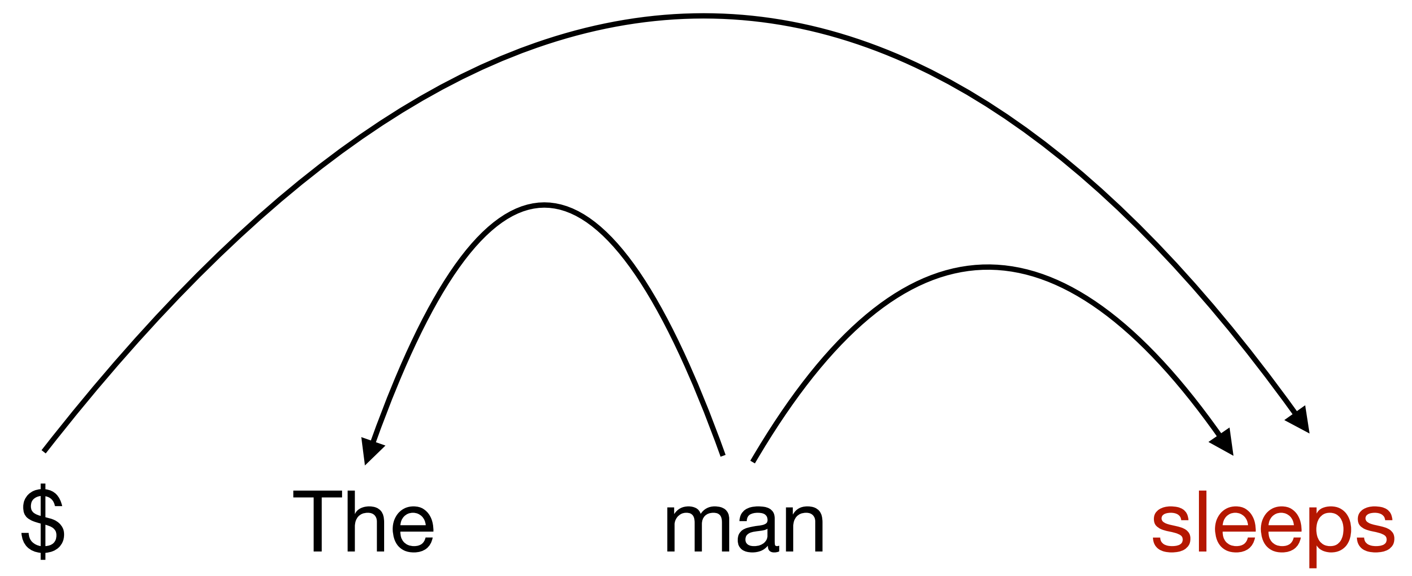
A Formal Definition of Dependency Structures

- **A dependency structure can be defined as a directed graph G, consisting of**
  - A set of V of nodes
  - A set of E of directed arcs (directed edges)
  - A linear precedence order $<$ on V (word order)
- **Formal Conditions of Dependency Structures**
  - G is connected: there exists a directed path from the root to every other node
  - G is acyclic: no cycles like $A \rightarrow B, B \rightarrow C, C \rightarrow A$



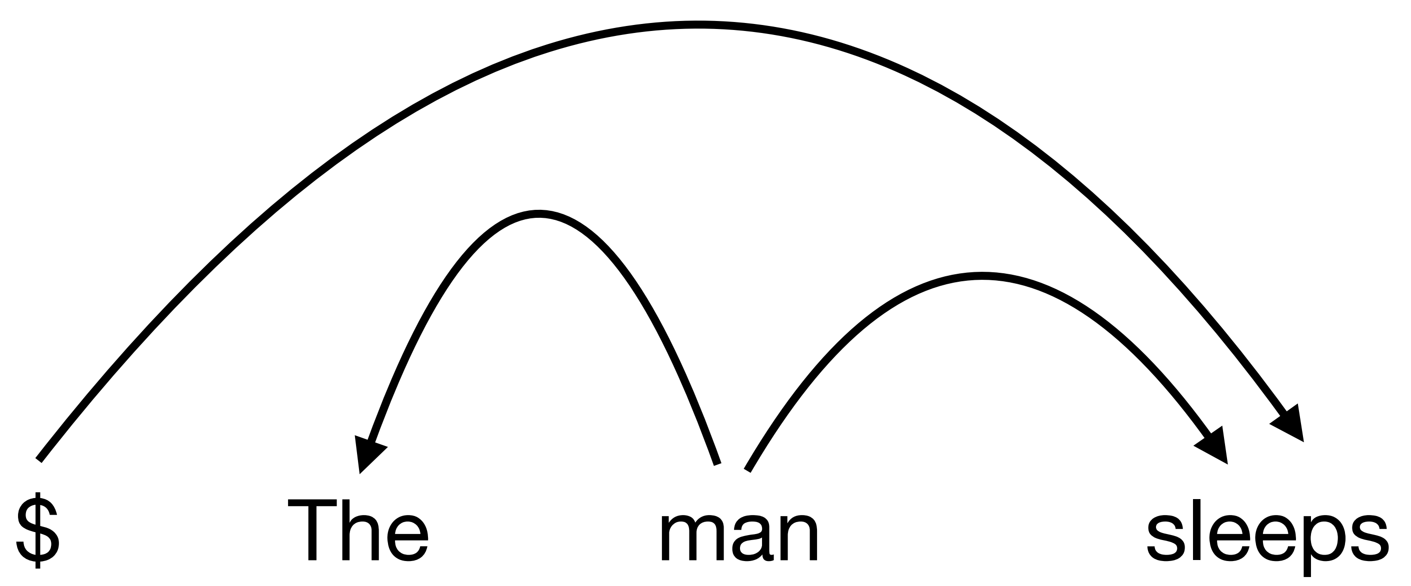
A Formal Definition of Dependency Structures

- **A dependency structure can be defined as a directed graph G, consisting of**
  - A set of V of nodes
  - A set of E of directed arcs (directed edges)
  - A linear precedence order $<$ on V (word order)
- **Formal Conditions of Dependency Structures**
  - G is connected: there exists a directed path from the root to every other node
  - G is acyclic: no cycles like $A \rightarrow B, B \rightarrow C, C \rightarrow A$
  - G obeys the single-head constraint: each non-root node has only one head

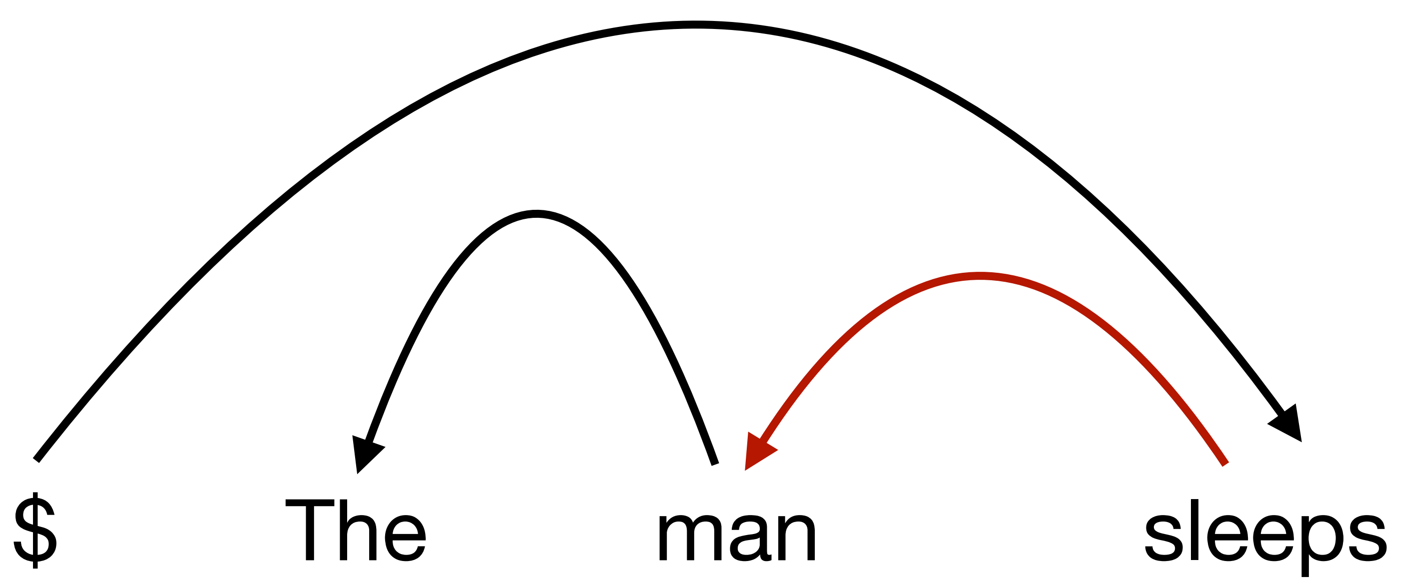


Dependency Structures: An Example

Invalid

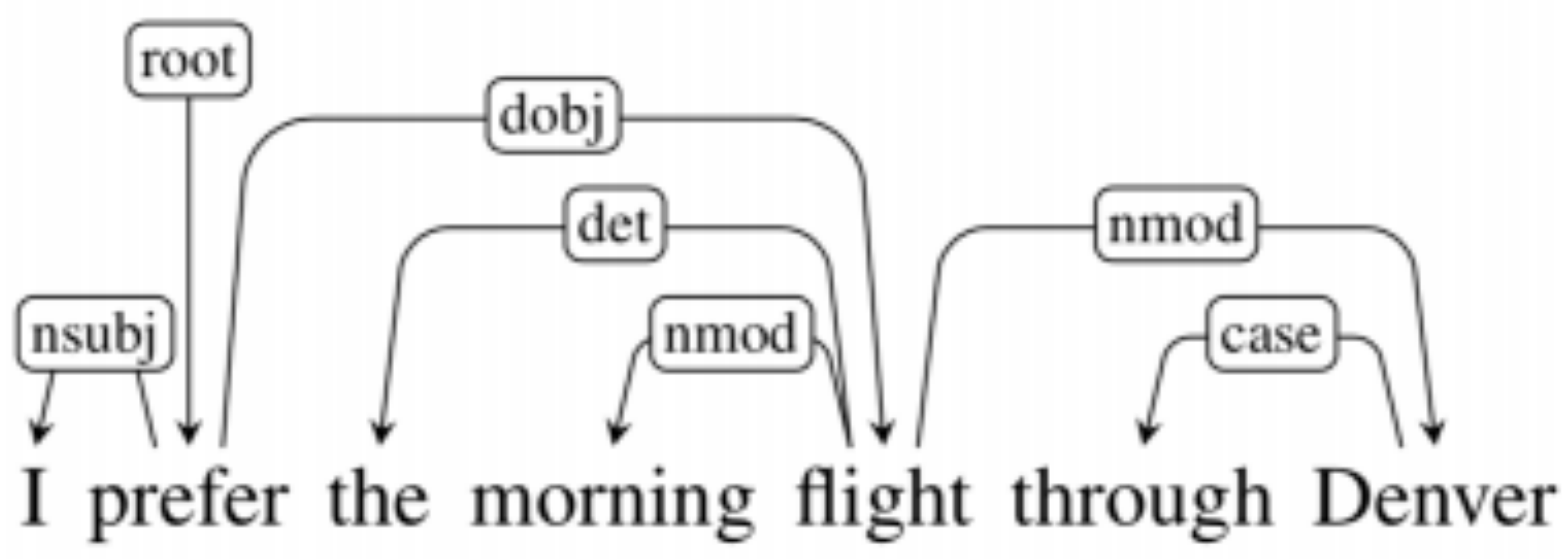


Valid

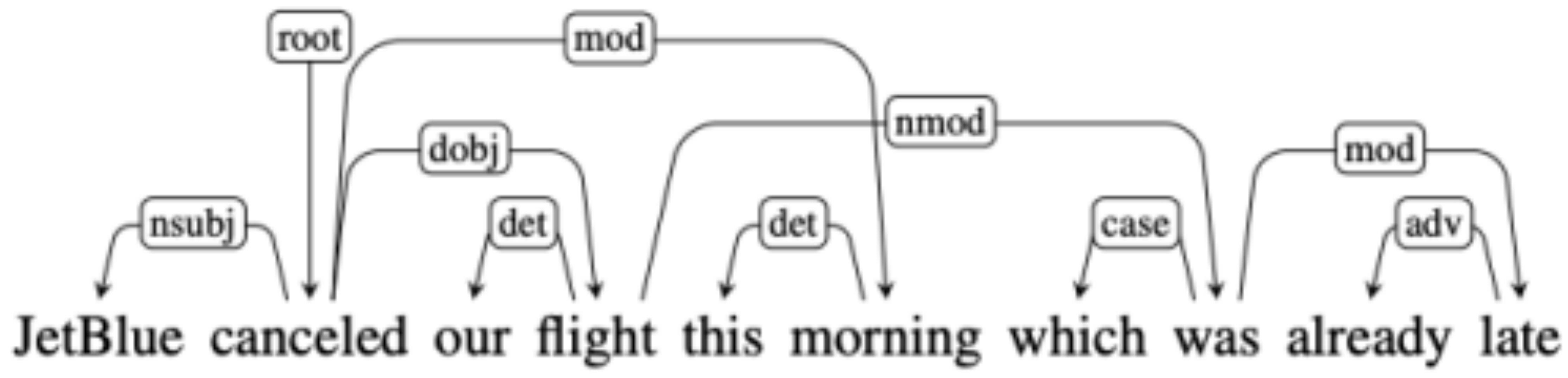


Additional Constraint: Projectivity

- Definition of projectivity: there are no crossing dependency when the words are laid out in their linear order, with all arcs above the words



projective



non-projective

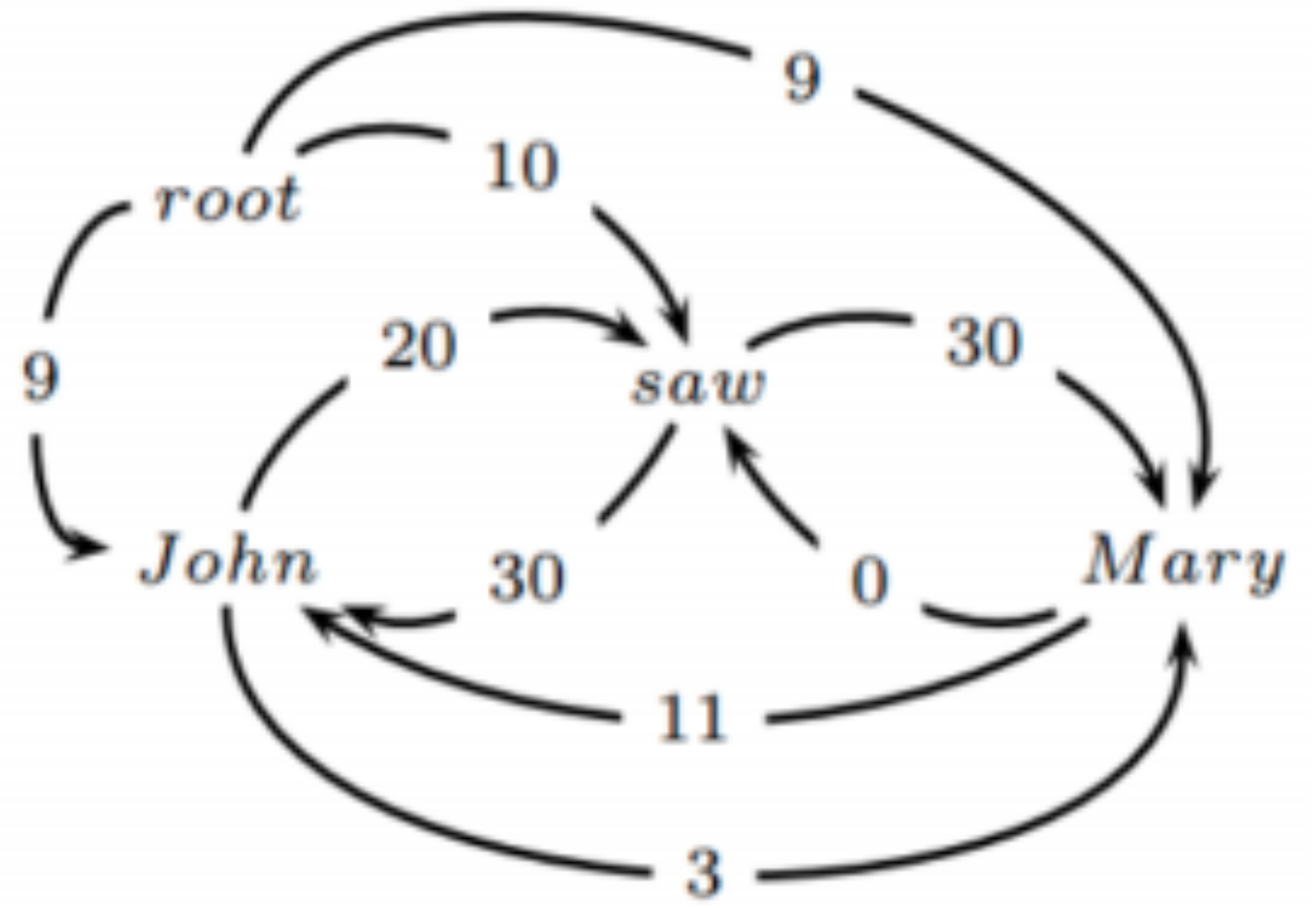
Non-projectivity arises due to long distance dependencies or in languages with flexible word order.

We will first consider projective parsing

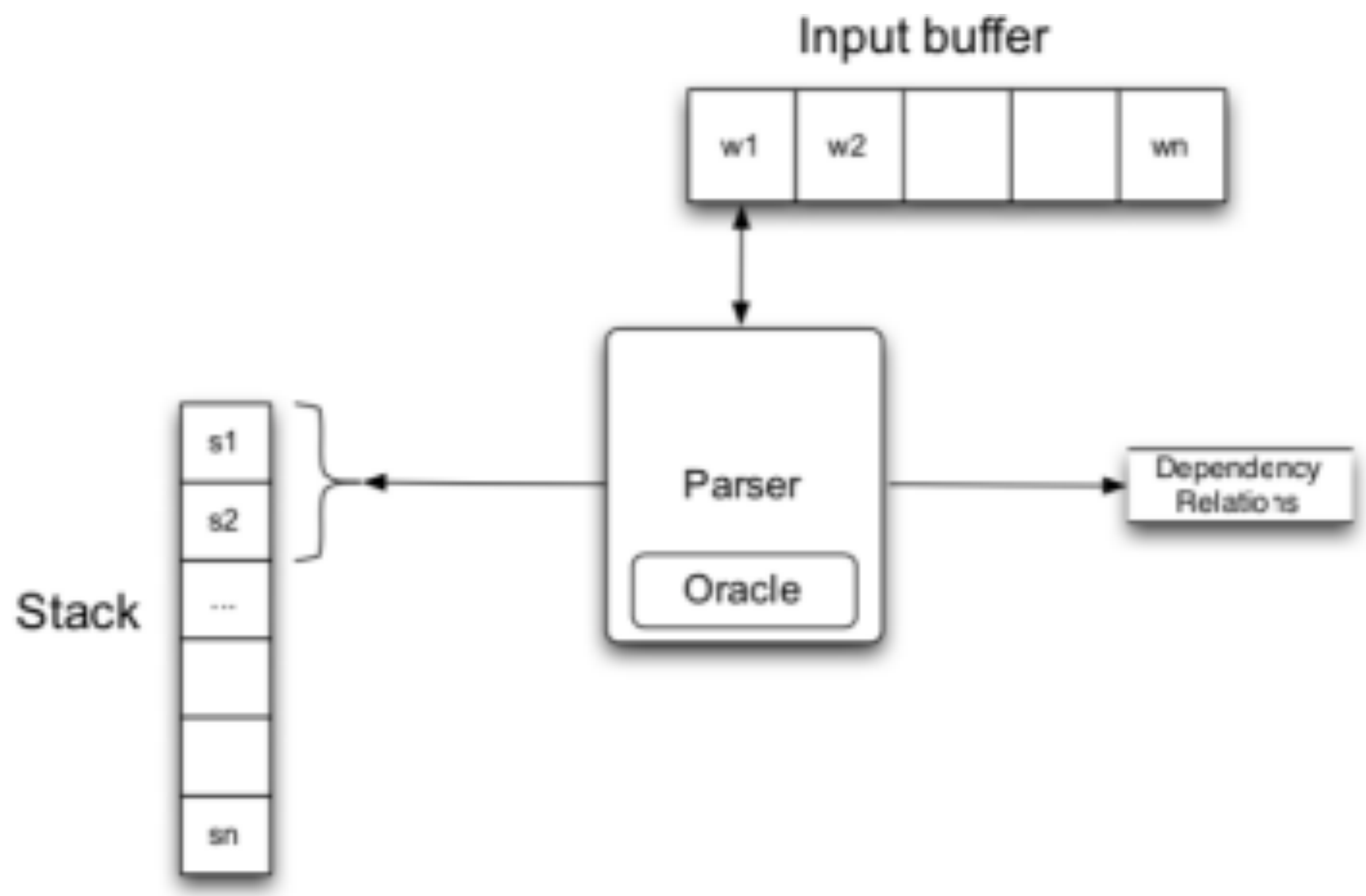
Dataset	# Sentences	(%) Projective
English	39,832	99.9
Chinese	16,091	100.0
Czech	72,319	76.9
German	38,845	72.2

Two Families of Dependency Parsing Algorithms

- **Graph-based Dependency Parsing**
  - Learning: Induce a model for scoring an entire dependency graph for a sentence
  - Parsing: Find the highest-scoring dependency graph



- **Transition-based Dependency Parsing**
  - Learning: Induce a model for predicting the next state transition, given the transition history
  - Parsing: Construct the optimal transition sequence



Graph-based Dependency Parsing

Graph-based Dependency Parsing

- **The General Problem**
  - **We have an input sentence x**
  - **We have a set valid dependency structures $\mathcal{T}(x)$**
  - **Aim is to provide a conditional probability $p(y \mid x),\ y \in \mathcal{T}(x)$**

Log-linear Model: $$p(y \mid x) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))}$$

How to simplify the feature function $f(x, y)$?

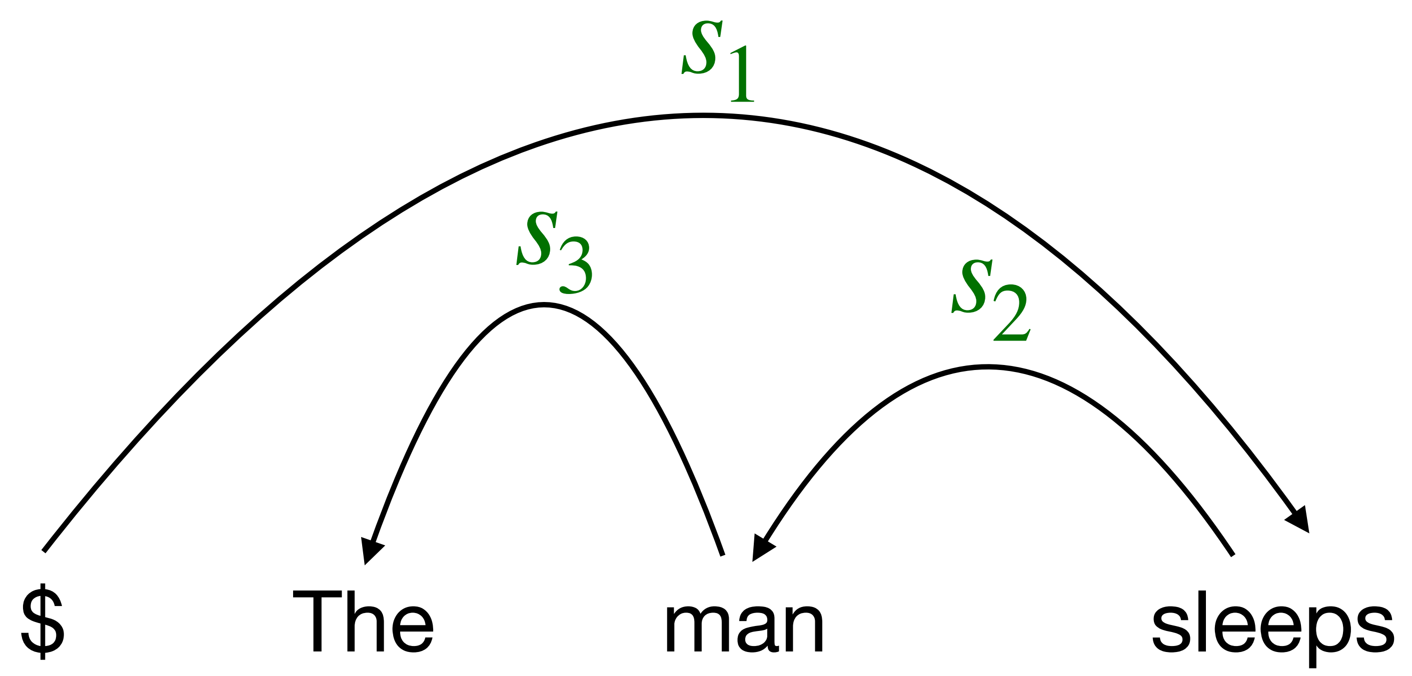
First-order Model

- **Factorize** $f(x, y)$ **into each edge of** y

$$p(y \mid x) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))} \qquad f(x, y) = \sum_{e \in y} f(x, e)$$

the score of an edge

$$\exp(v \cdot f(x, y)) = \exp(v \cdot \sum_{e \in y} f(x, e)) = \prod_{e \in y} \exp(\boxed{v \cdot f(x, e)})$$



First-order Model

- **Factorize $f(x, y)$ into each edge of y**

$$p(y \mid x) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))} \qquad f(x, y) = \sum_{e \in y} f(x, e)$$

- **Two standard problems:**
  - Learning: $\sum_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))$
  - Parsing: $\arg\max_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))$

First-order Projective Parsing Algorithm

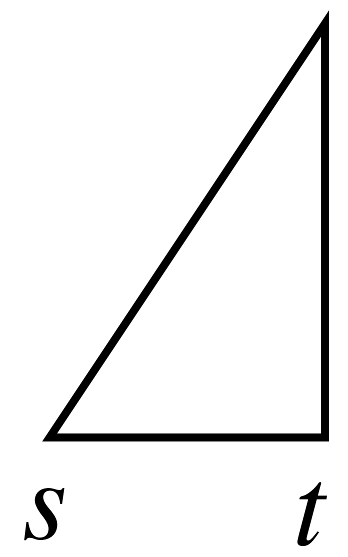
- **Cubic Parsing Algorithm** [Eisner, 1996]
- **Projective Parse Trees only**
  - $\mathcal{T}(x)$ only contains projective trees
- **Define a dynamic programming table**
  - $\pi[s, t, d, c]$ = maximum probability of a dependency graph spanning words $s, \ldots, t$ inclusive, with direction $d \in \{\rightarrow, \leftarrow\}$, and completeness $c \in \{0,1\}$

- **Our goal is to calculate** $\max_{y \in \mathcal{T}(x)} p(y \mid x) = \pi[0, n, \rightarrow, 1]$

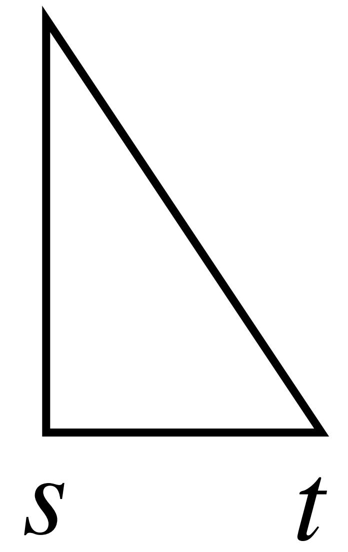
First-order Projective Parsing Algorithm

complete items

$\pi[s, t, \rightarrow, 1]$ dependency graphs from word s to t, with s as the root



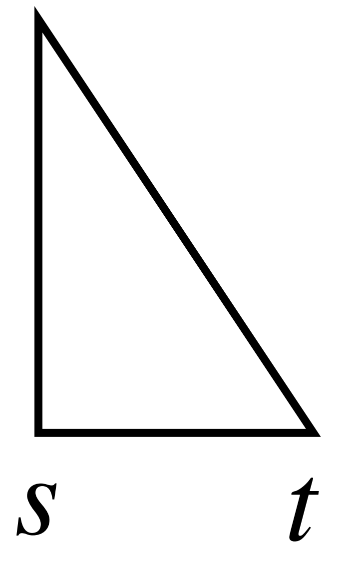
$\pi[s, t, \leftarrow, 1]$ dependency graphs from word s to t, with t as the root



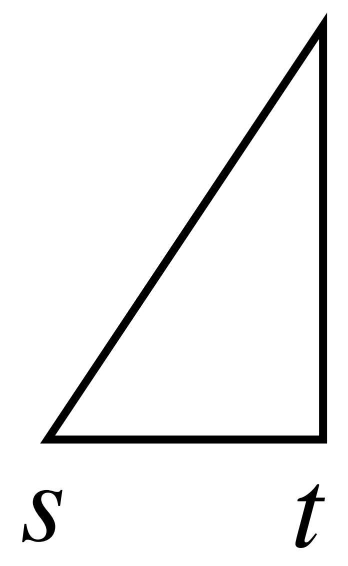
First-order Projective Parsing Algorithm

complete items

$\pi[s, t, \rightarrow, 1]$ dependency graphs from word s to t, with s as the root



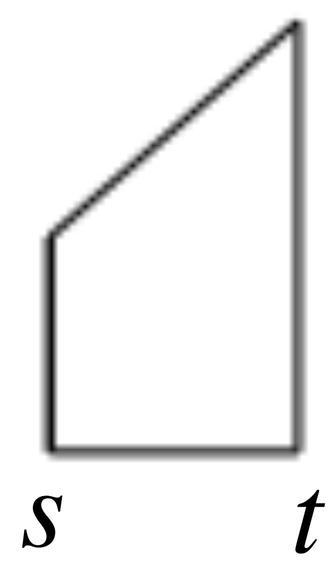
$\pi[s, t, \leftarrow, 1]$ dependency graphs from word s to t, with t as the root



incomplete items

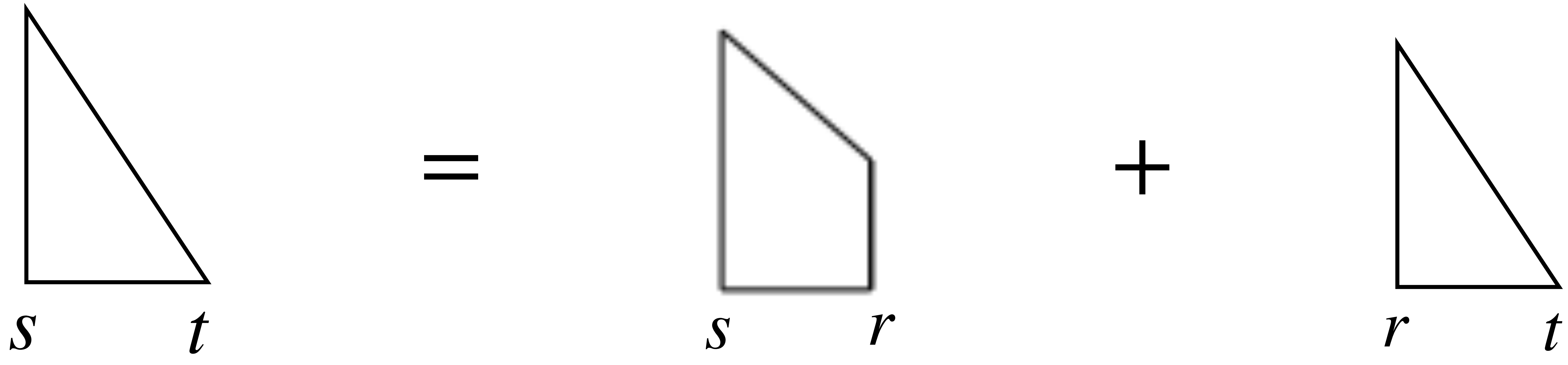
$\pi[s, t, \rightarrow, 0]$ dependency graphs from word s to t, with s as the root and an edge $s \rightarrow t$

$\pi[s, t, \leftarrow, 0]$ dependency graphs from word s to t, with t as the root and an edge $s \leftarrow t$

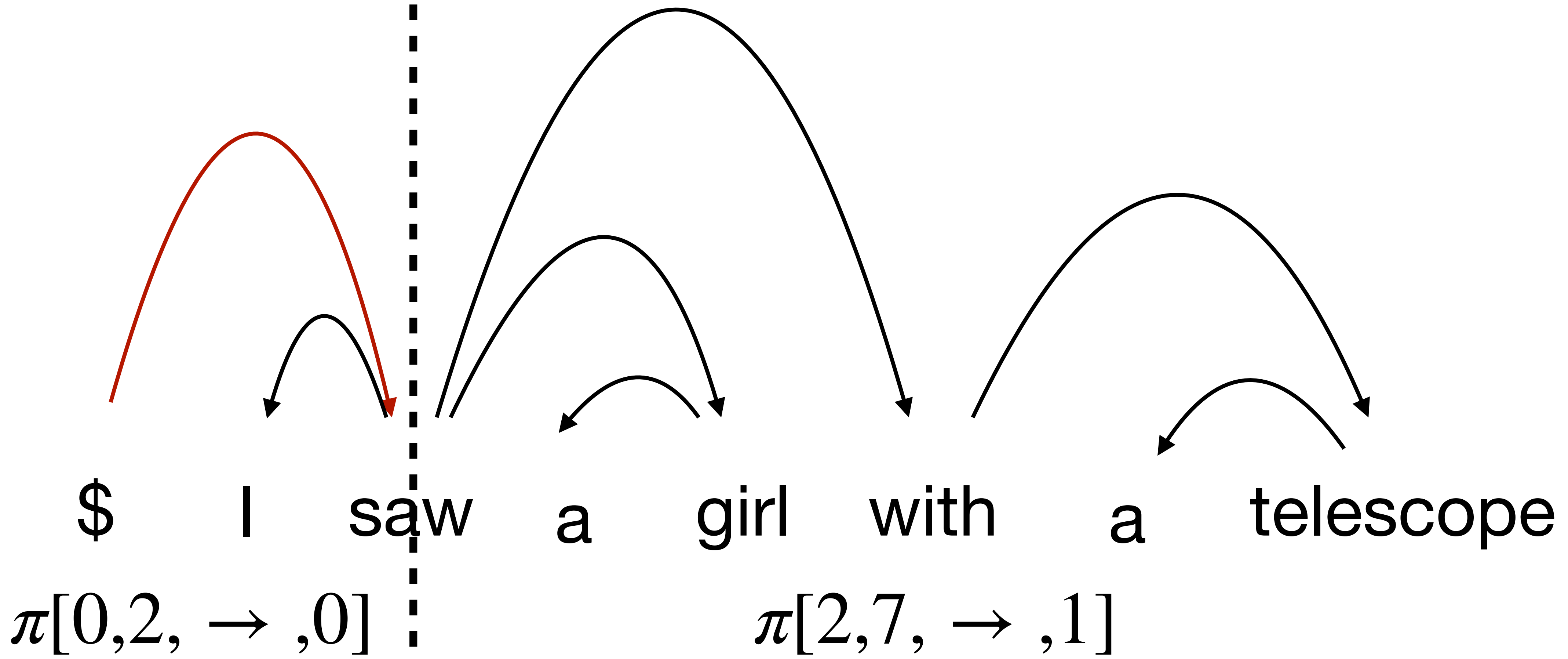


First-order Projective Parsing Algorithm

- **Dynamic programming derivations**

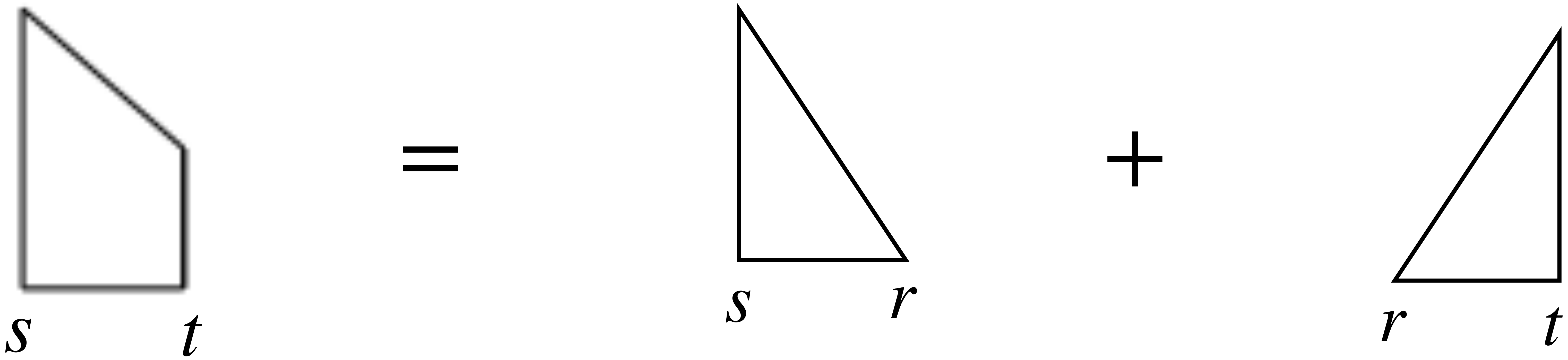


$$\pi[s, t, \rightarrow, 1] = \max_{s<r\leq t} \pi[s, r, \rightarrow, 0] + \pi[r, t, \rightarrow, 1]$$

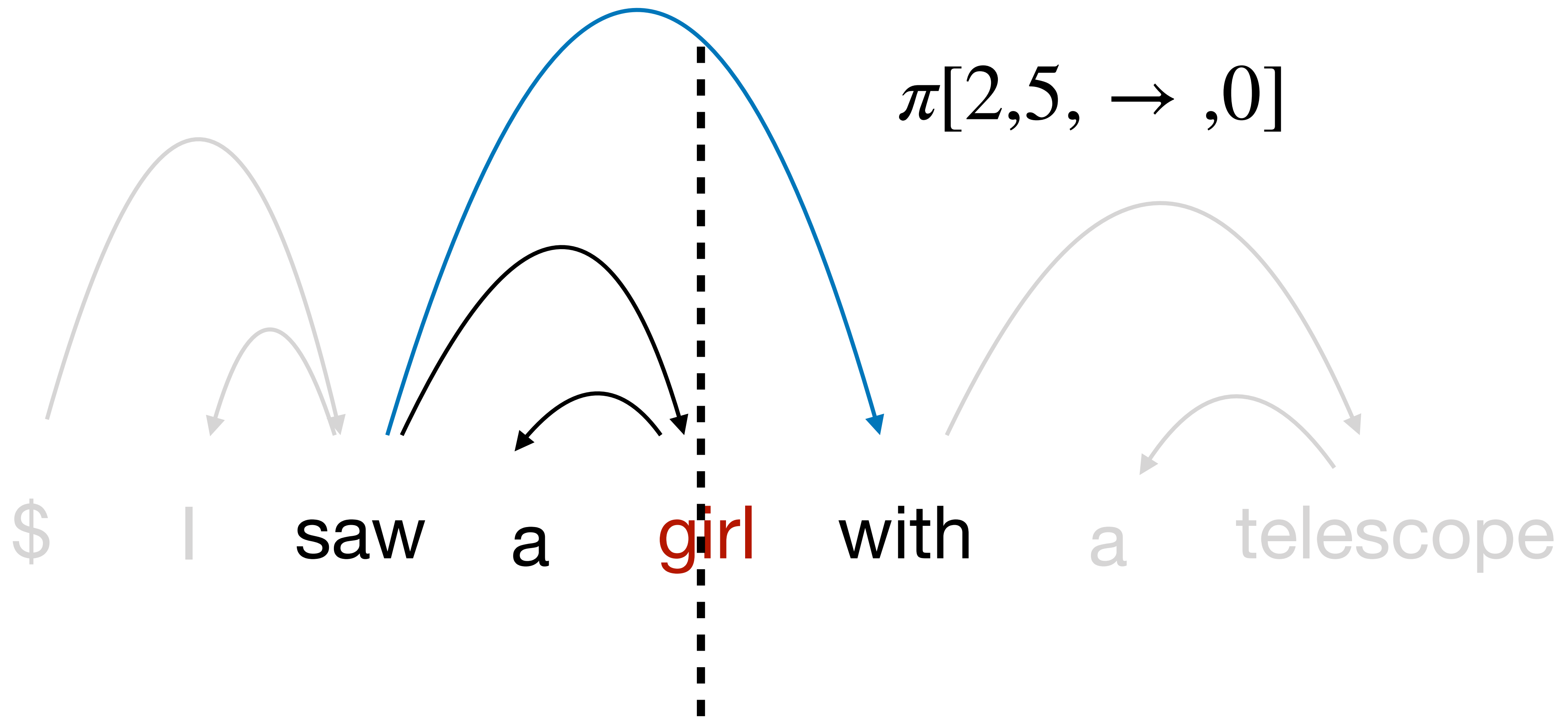


First-order Projective Parsing Algorithm

- **Dynamic programming derivations**



$$\pi[s, t, \rightarrow, 0] = \max_{s \le r < t} \Big(\pi[s, r, \rightarrow, 1] + \pi[r+1, t, \leftarrow, 1] + s(s \rightarrow t)\Big)$$



First-order Projective Parsing Algorithm

Initialization: $C[s][s][d][c] = 0.0 \quad \forall s, d, c$

for $k : 1..n$

  for $s : 1..n$

    $t = s + k$

    if $t > n$ then break

    % First: create incomplete items

    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} \; (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$

    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} \; (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

    % Second: create complete items

    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} \; (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$

    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} \; (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

  end for

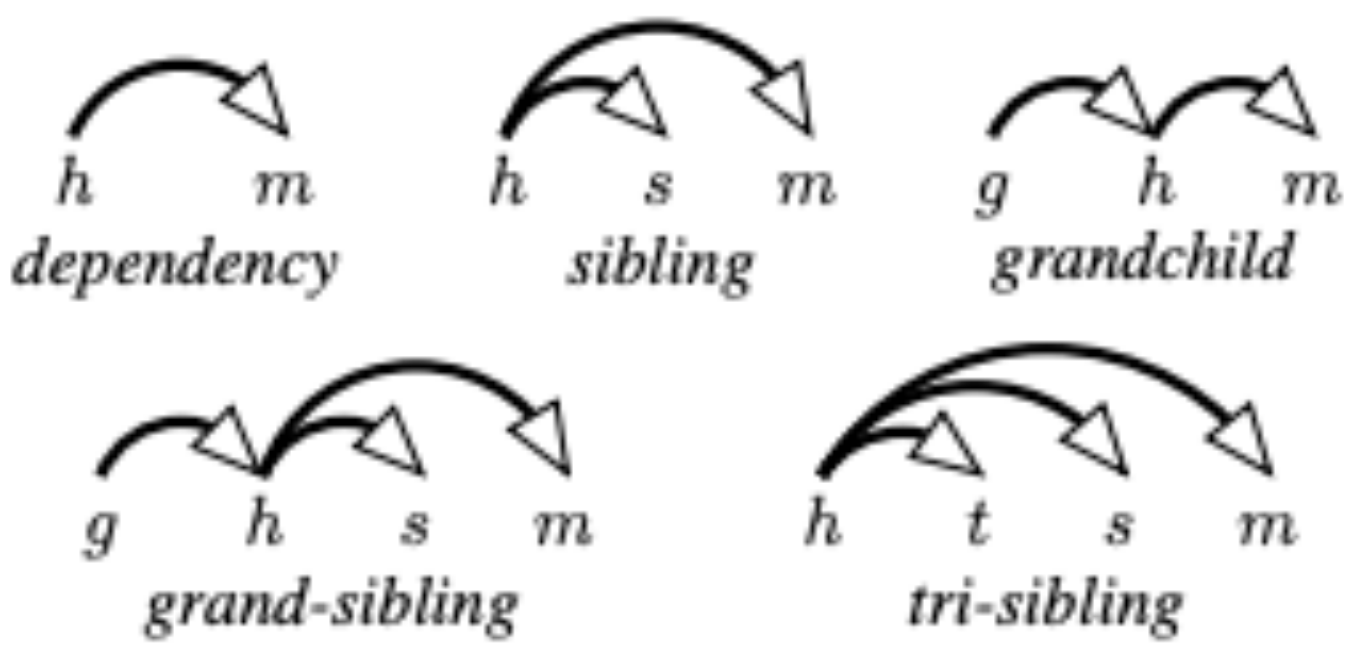
end for

Running time:
$O(n^3)$

Higher-order Parsing

- **First-order: factorizing features into each edge**
- **Higher-order: factorizing features into more complex components**

$$f(x, y) = \sum_{p \in y} f(x, p)$$

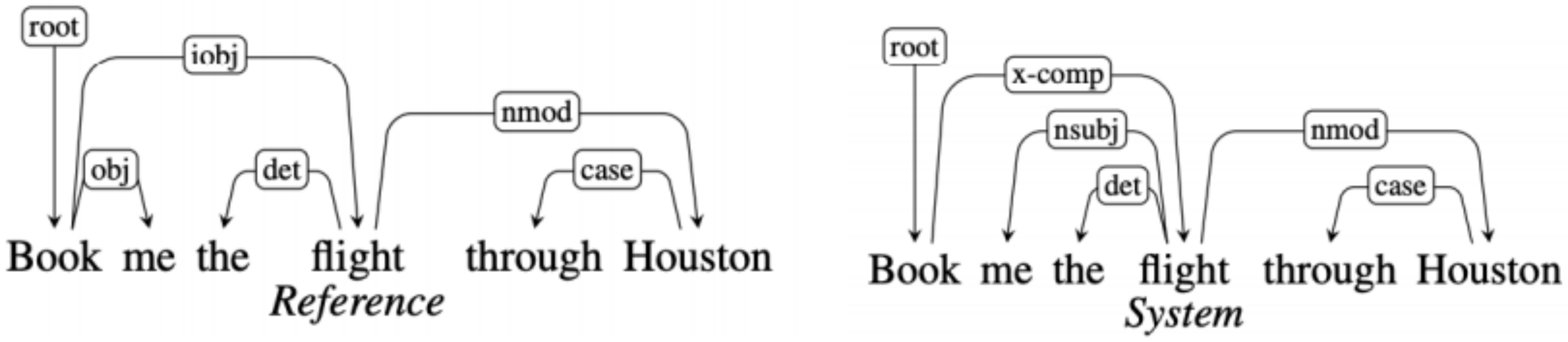


Non-projective Parsing

- **Two standard problems:**
  - Learning: $\sum_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))$
  - Parsing: $\arg \max_{y' \in \mathcal{T}(x)} \exp(v \cdot f(x, y'))$
- **First-order Model:**
  - Learning: Matrix-Tree Theorem [Koo et al., 2007]
  - Parsing: Maximum Spanning Tree algorithm [McDonald, 2005]
- **High-order Models: NP-hard**

Evaluation Dependency Parsing

- **Unlabeled Attachment Score (UAS)**
  - Percentage of words that have been assigned the corrected head
- **Labeled Attachment Score (LAS)**
  - Percentage of words that have been assigned the correct head & label
- **Root Accuracy (RA)**
  - Accuracy of the root dependencies



UAS = 5/6 LAS = 4/6 RA = 1/1

Parsing Exeriments

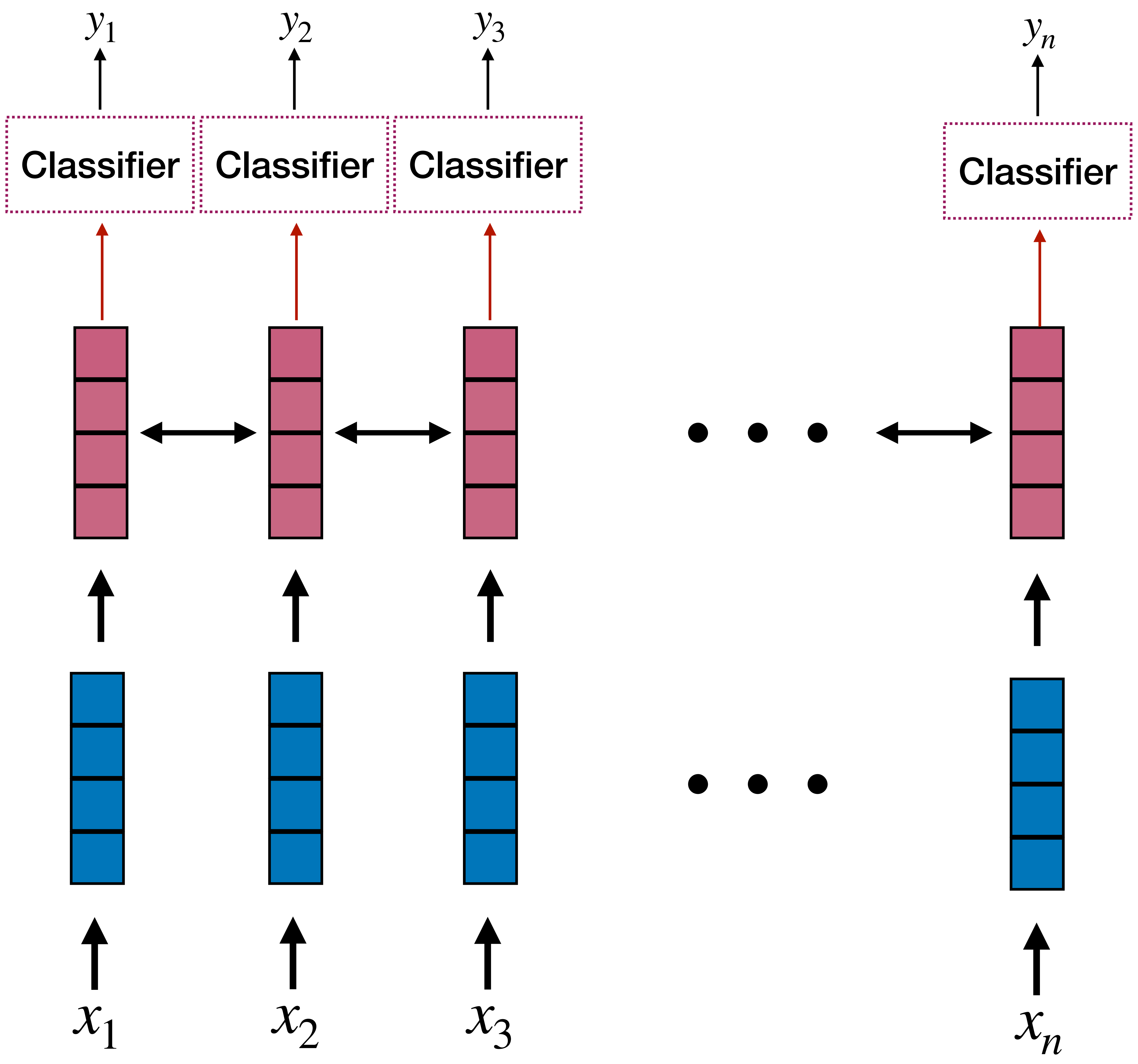
- **Penn Treebank**

	UAS	Complexity
1st-proj	91.8	$O(n^3)$
1st-non-proj	91.7	$O(n^3)$
2nd-proj	92.4	$O(n^3)$
3nd-proj	93.0	$O(n^4)$
4nd-proj	93.4	$O(n^5)$

RNNs for Dependency Parsing

Recap: RNN for Sequence Labeling

- A simple bidirectional LSTM model



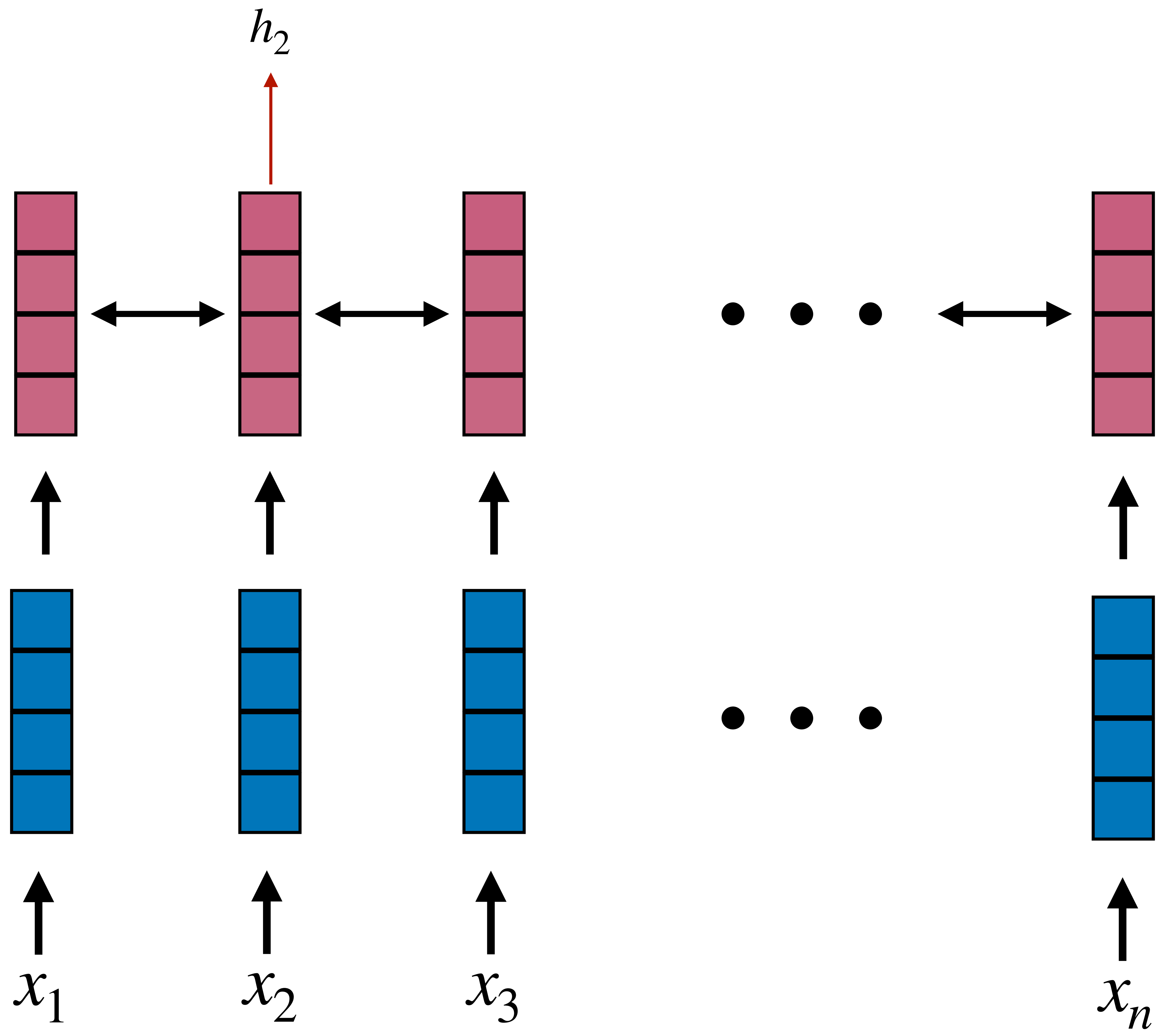
No structured models to consider dependencies between Y

Dependency Parsing

- **An Unstructured Model**
  - Deep BiAffine Parser (Dozat, 2017)

$$P(y\,|\,x) = \prod_{i=1}^{n} P(h_i \rightarrow i\,|\,x)$$

$$P(h_i \rightarrow i\,|\,X) = \frac{\exp(z_i^T W z_{h_i})}{\sum_{h=0}^{n} \exp(z_i^T W z_h)}$$



Pros:
- Simple, no structured modeling

Cons:
- Outputs are not guaranteed to be a tree

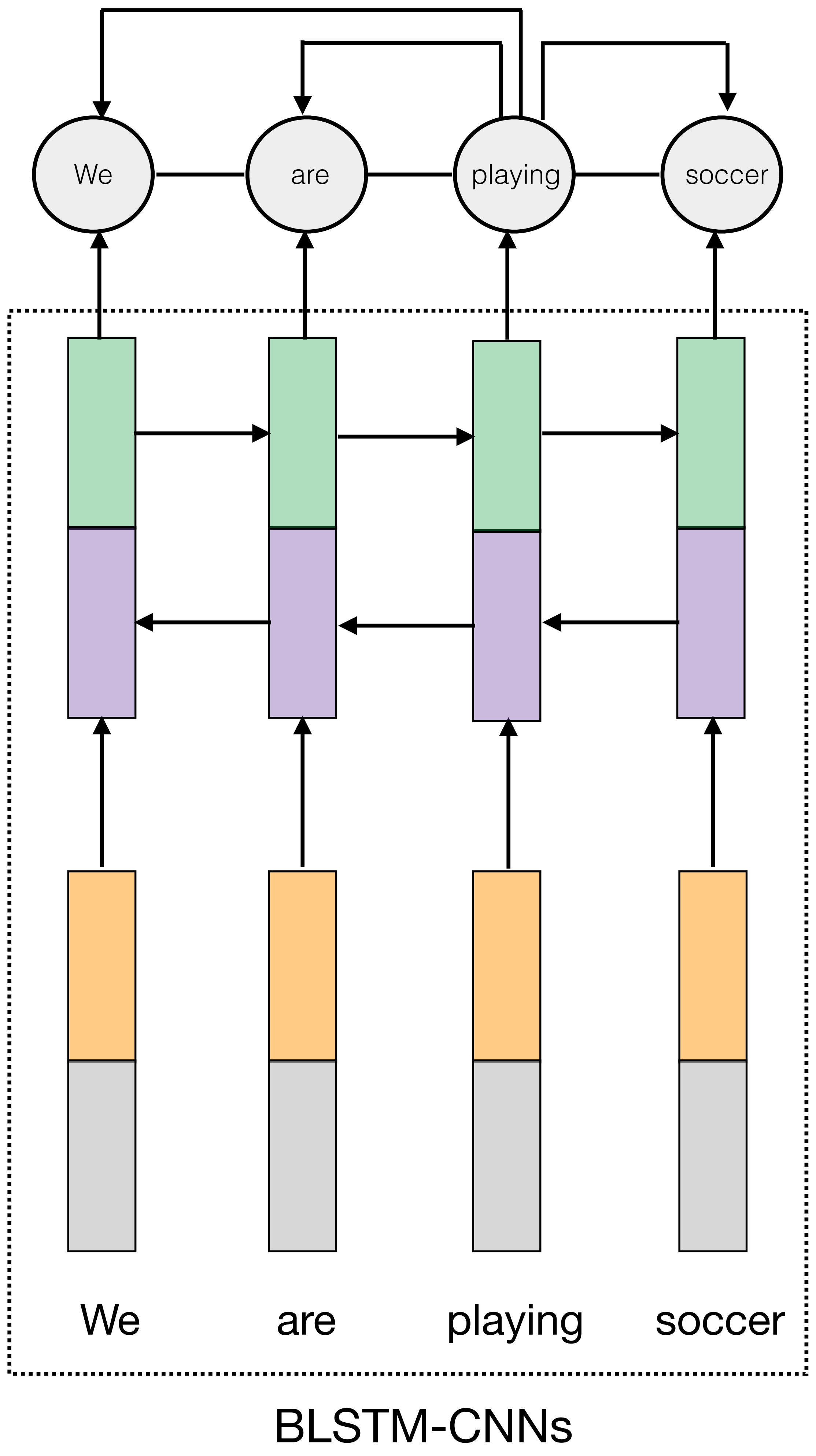
Dependency Parsing

- **DeepBiAffine Parser**

	English	German
Graph-based (2nd-order)	92.4%	89.3%
BiAffine	94.1%	91.6%
BiAffine+CNNs	94.9%	93.4%

Dependency Parsing: NeuroMST Parser

- **We stack a first-order graph-based model on top of a BLSTM-CNN encoder**



Dependency Parsing

- **DeepBiAffine Parser**

	English	German
4th-proj	93.4%	89.3%
BiAffine	94.1%	91.6%
BiAffine+CNNs	94.9%	93.4%
NeuroMST	95.8%	93.8%

Transition-based Dependency Parsing

Transition-based Parsing

- **Basic Ideas**
  - Define a transition system for dependency parsing
  - Learn a machine learning model for scoring possible transitions
  - Parse by searching for the optimal transition sequence

Transition-based Parsing

- **The Arc-standard Transition System**
  - Three data structures, a stack σ, a buffer β and a set α
  - A configuration consists of
    1. A *stack* σ consisting of a sequence of words, e.g.,

       $$\sigma = [\text{root}_0, \text{I}_1, \text{live}_2]$$

    2. A *buffer* β consisting of a sequence of words, e.g.,

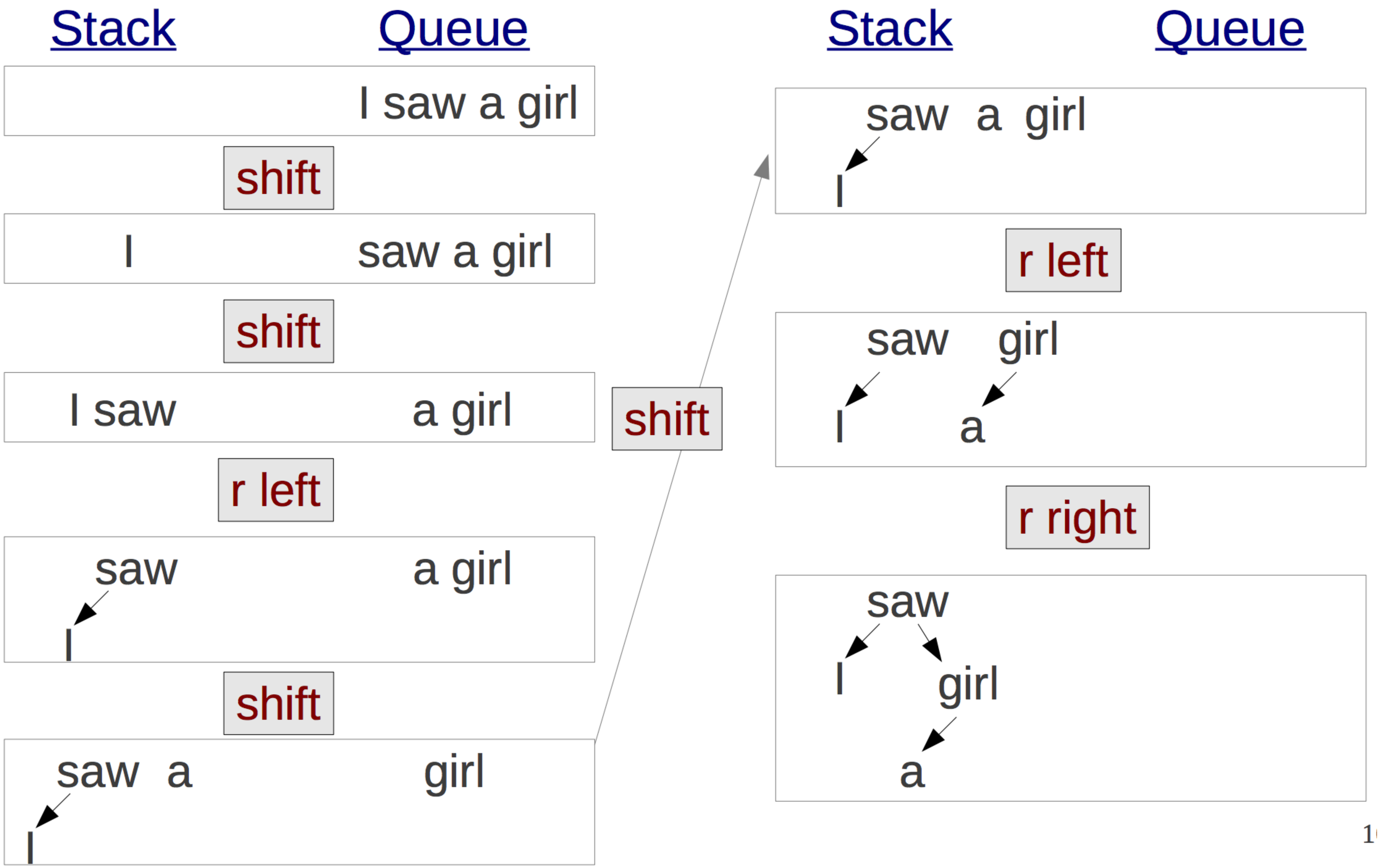
       $$\beta = [\text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7]$$

    3. A set α of *labeled dependencies*, e.g.,

       $$\alpha = \{\{1 \rightarrow^{nsubj} 2\}, \{6 \rightarrow^{nn} 5\}$$

  -
  - Initial configuration: $\sigma = [\$],\ \beta = [w_1, \ldots, w_n],\ \alpha = \{\}$
  - Three types of transition actions: LEFT-ARC, RIGHT-ARC, SHIFT
  - A terminal configuration: $\sigma = [\$],\ \beta = []$

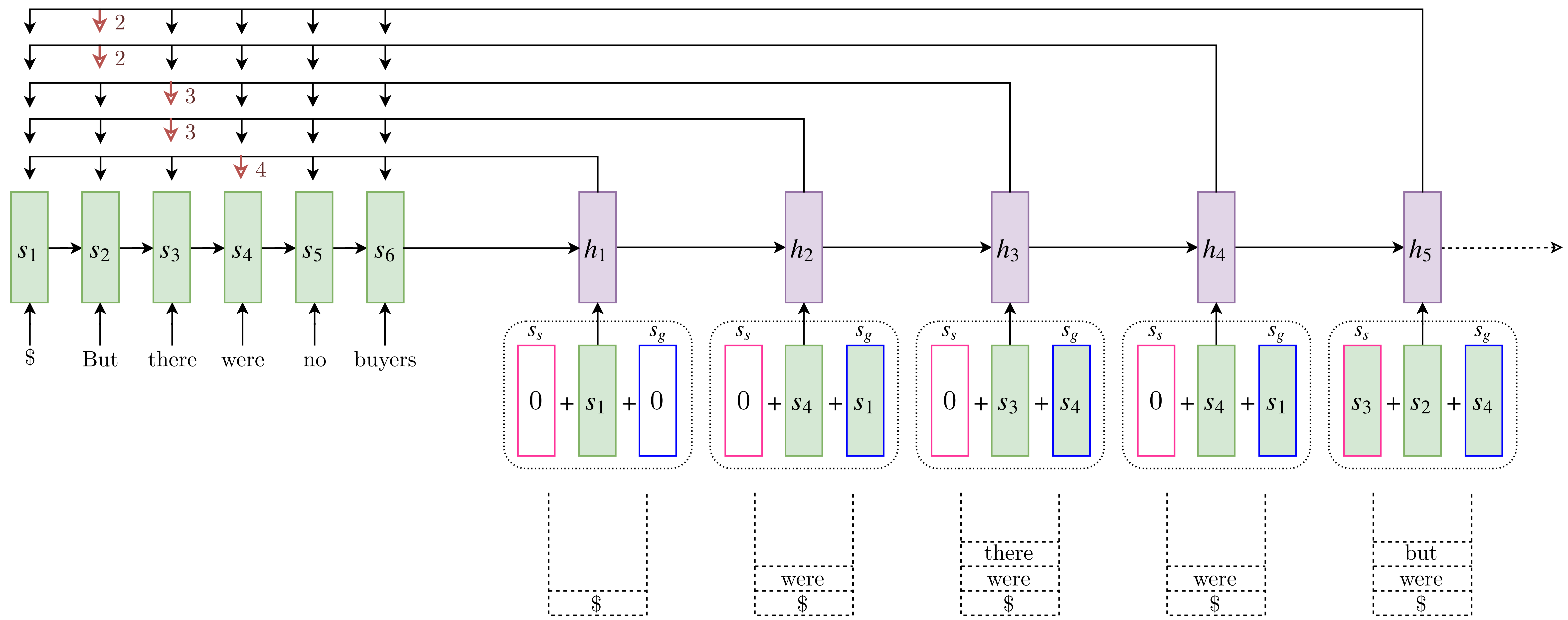
Transition-based Parsing



Transition-based Parsing: Parsing

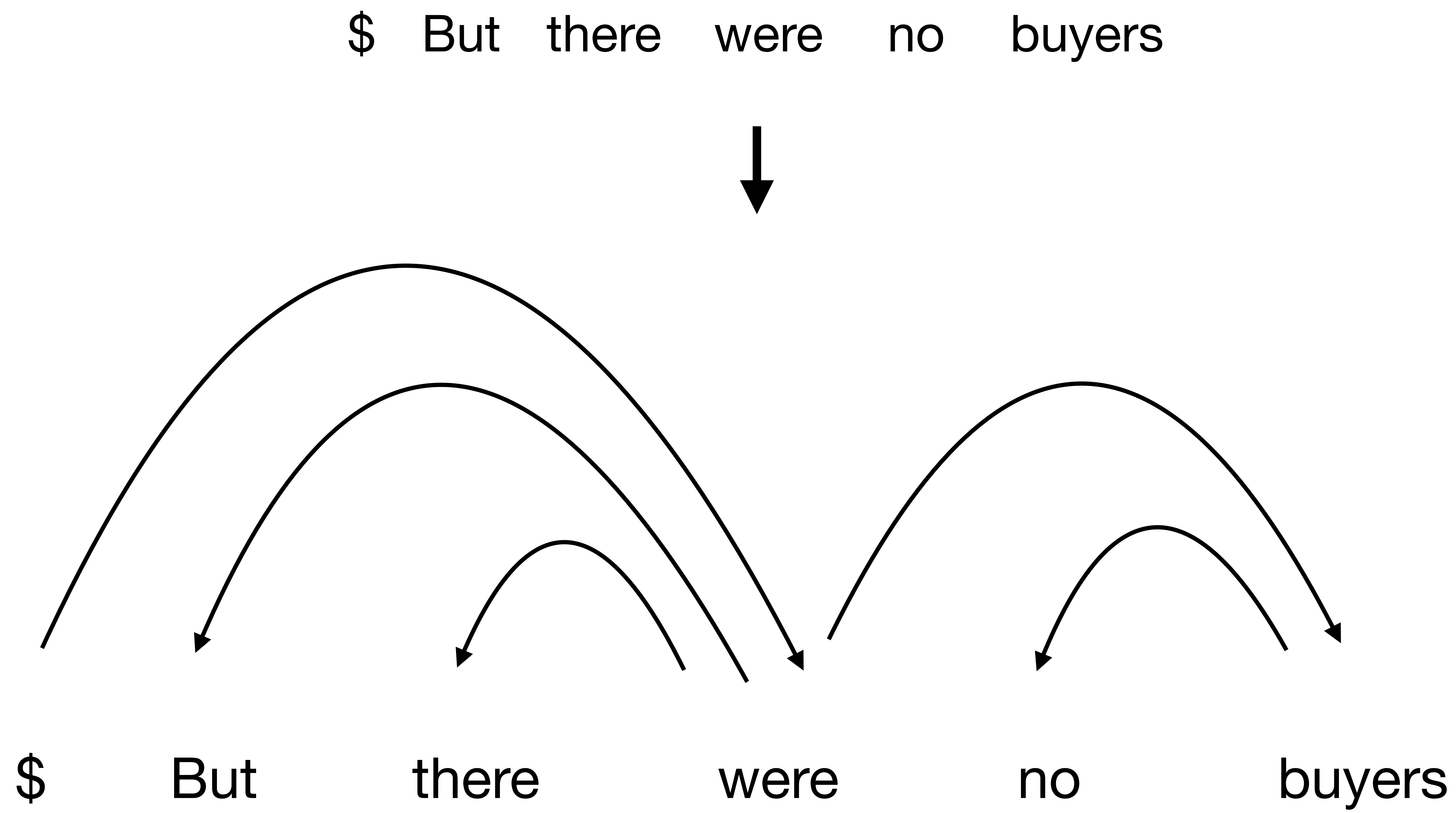
- **No Exact Parsing Algorithm**
  - Greedy search or beam search
  - Linear time complexity
  - Comparable performance with graph-based parsing algorithms

Stack-Pointer Network for Dependency Parsing

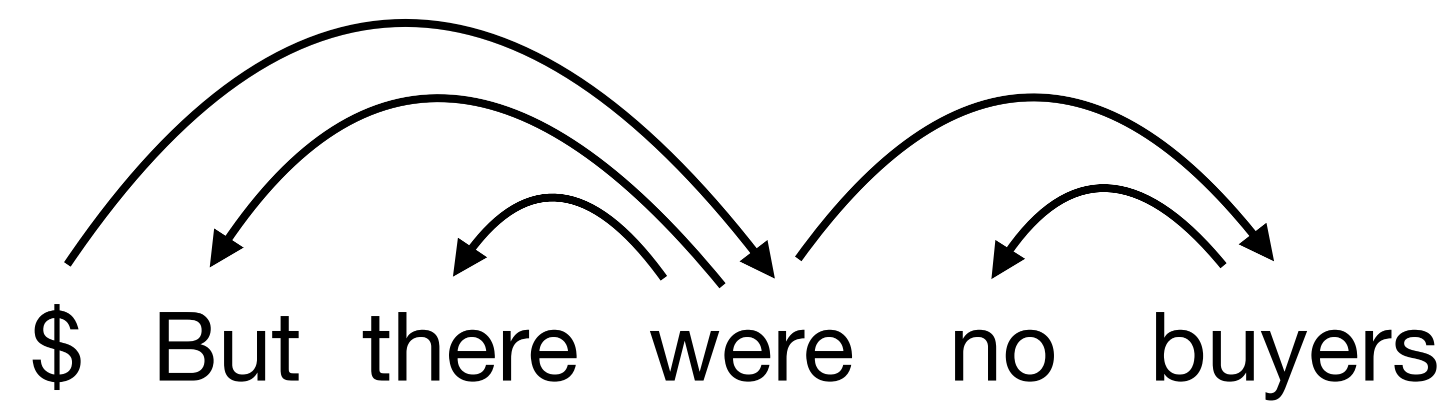


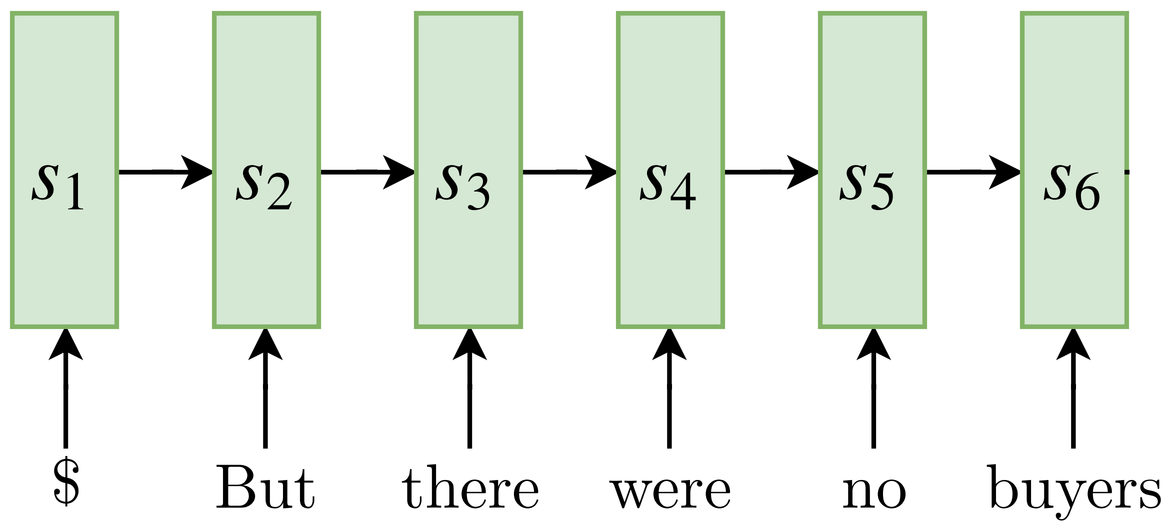
- **Order:** Top-down, depth-first
- **Actions:** "Point" to the next word to choose as a child
- **Model:** A neural network, based on "pointer networks"
- **Advantages:**
  - Top-down parsing maintains a global view of the sentence
  - High accuracy
  - Can maintain full history, low asymptotic running time (c.f. graph-based)

StackPtr: An Example

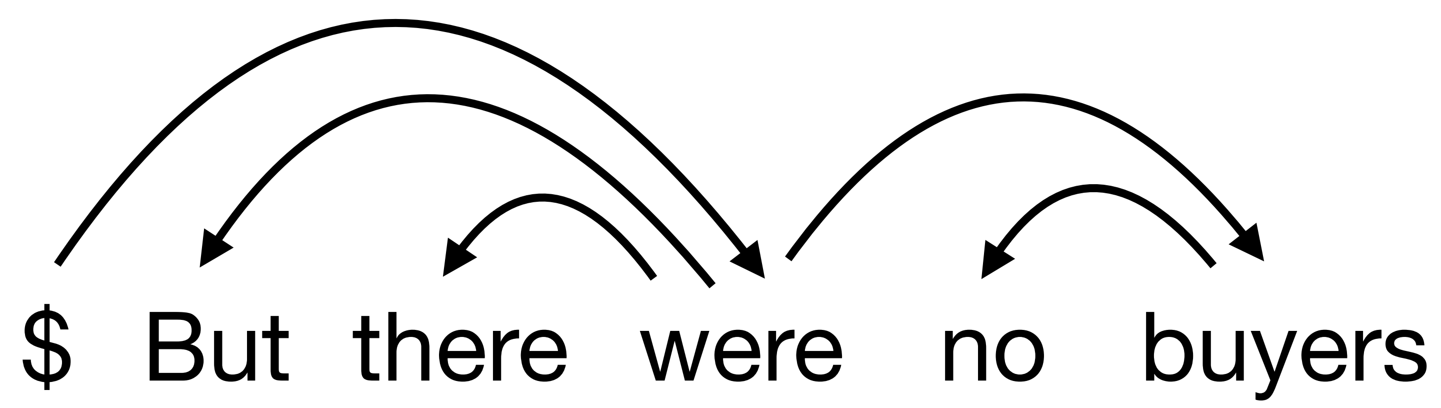


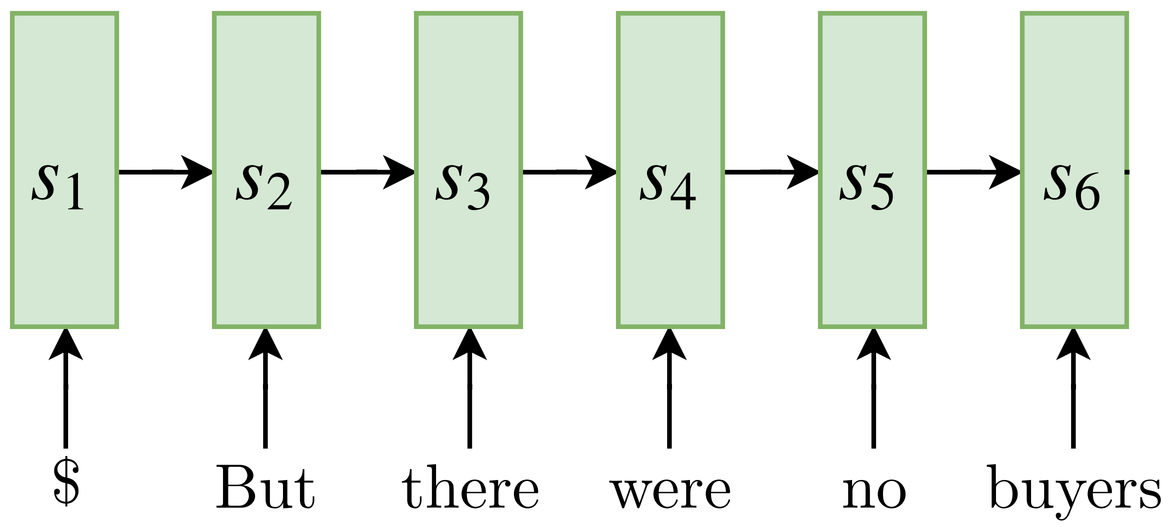
StackPtr: An Example

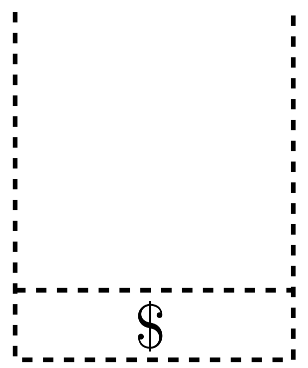




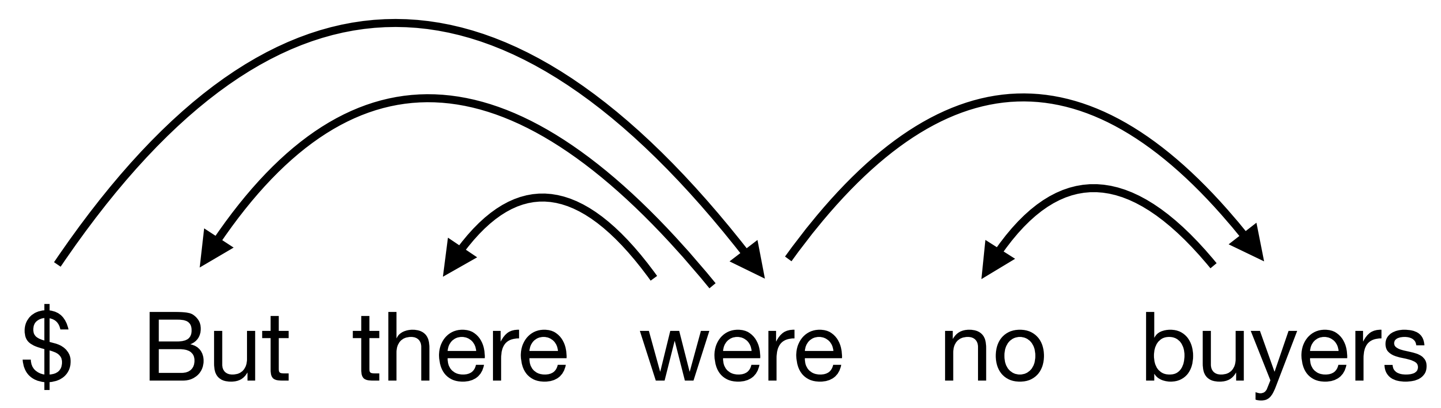
StackPtr: An Example

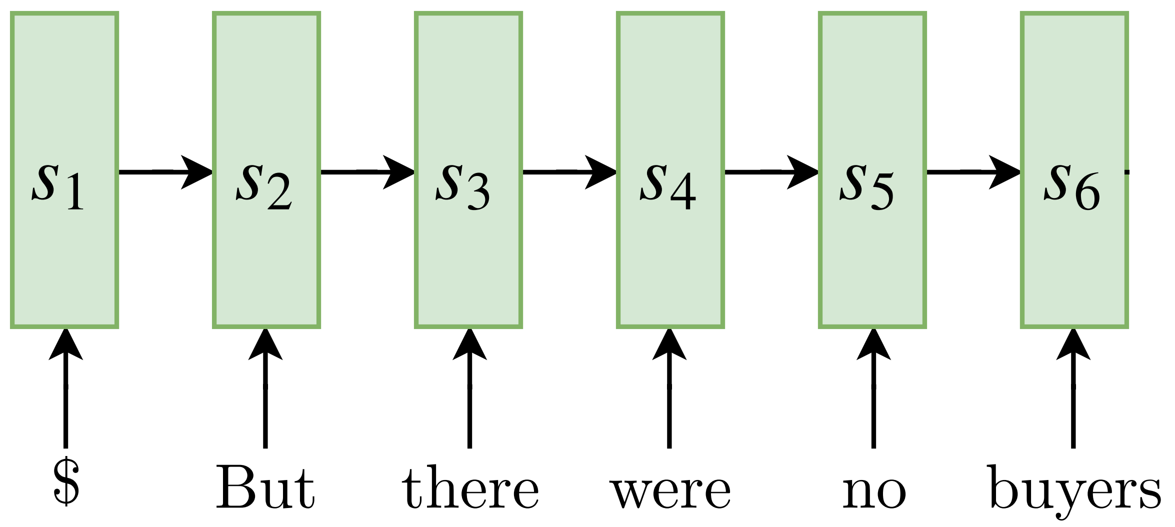


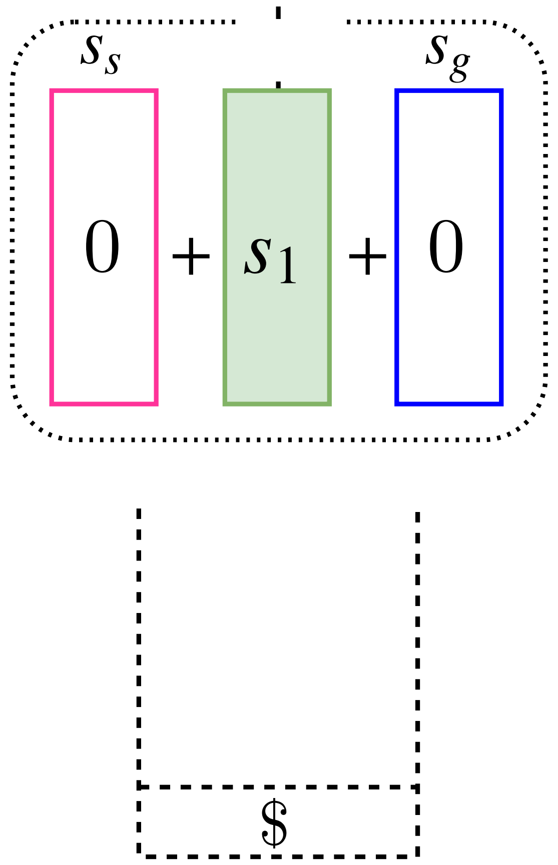




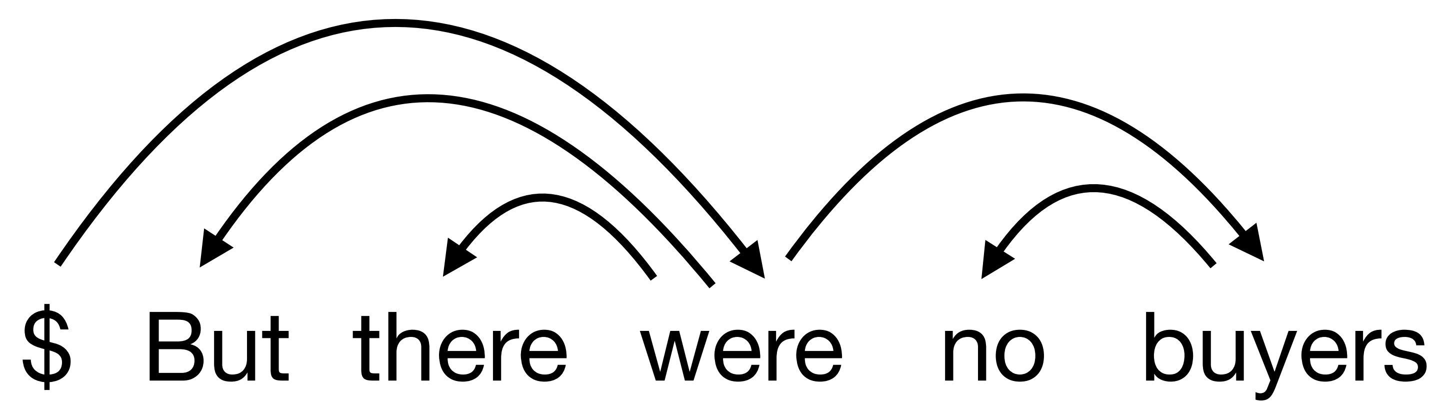
StackPtr: An Example

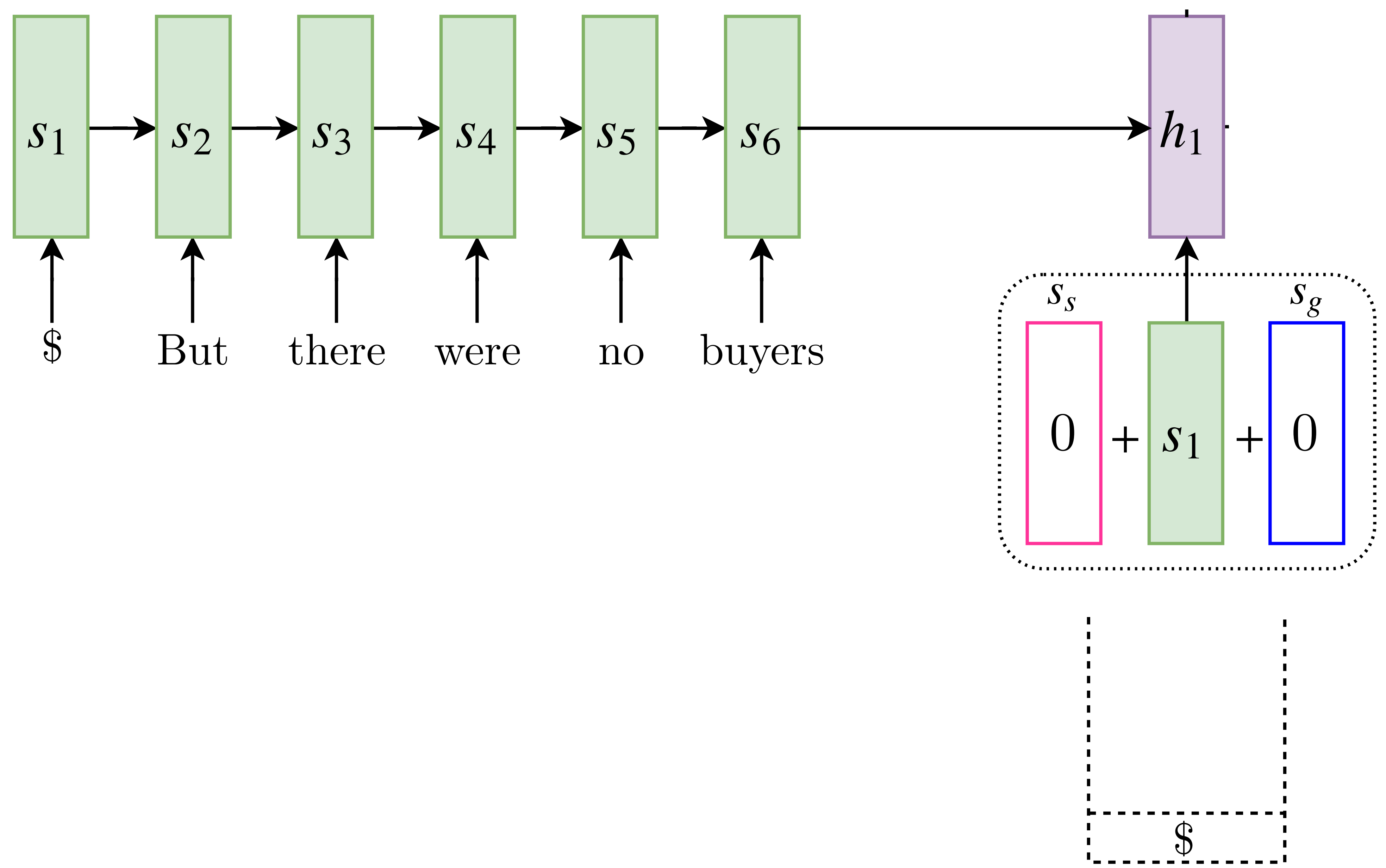




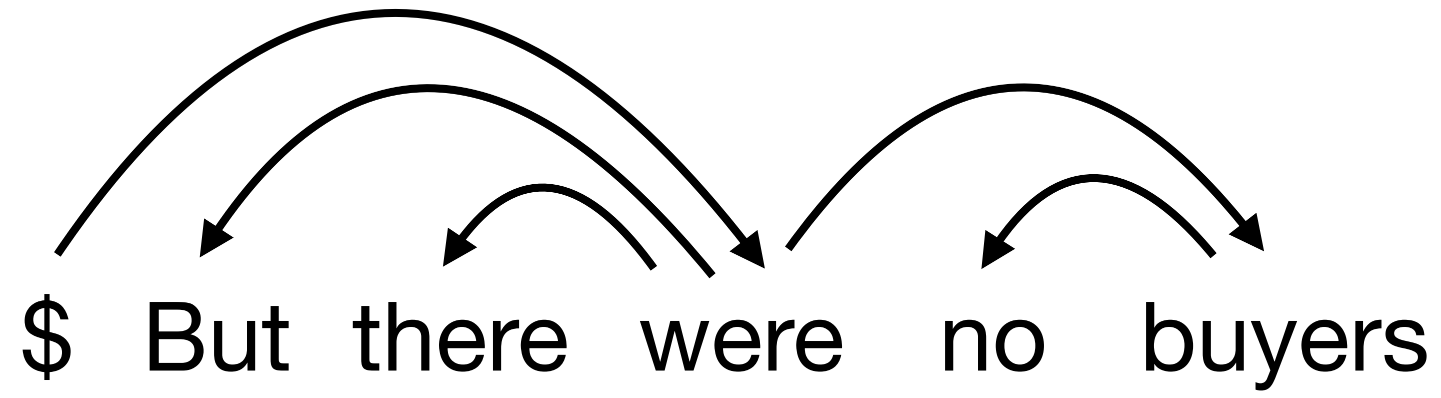


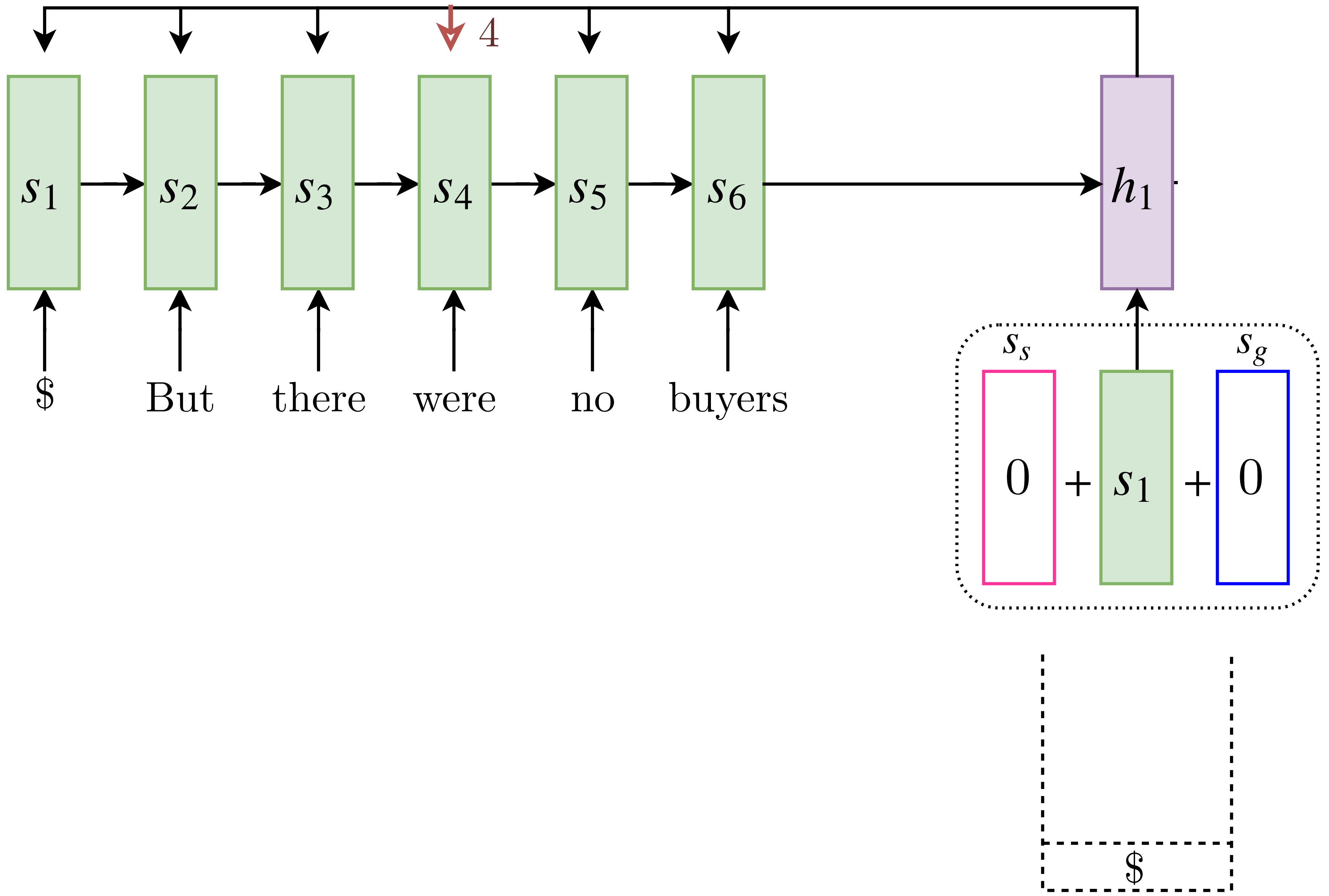
StackPtr: An Example



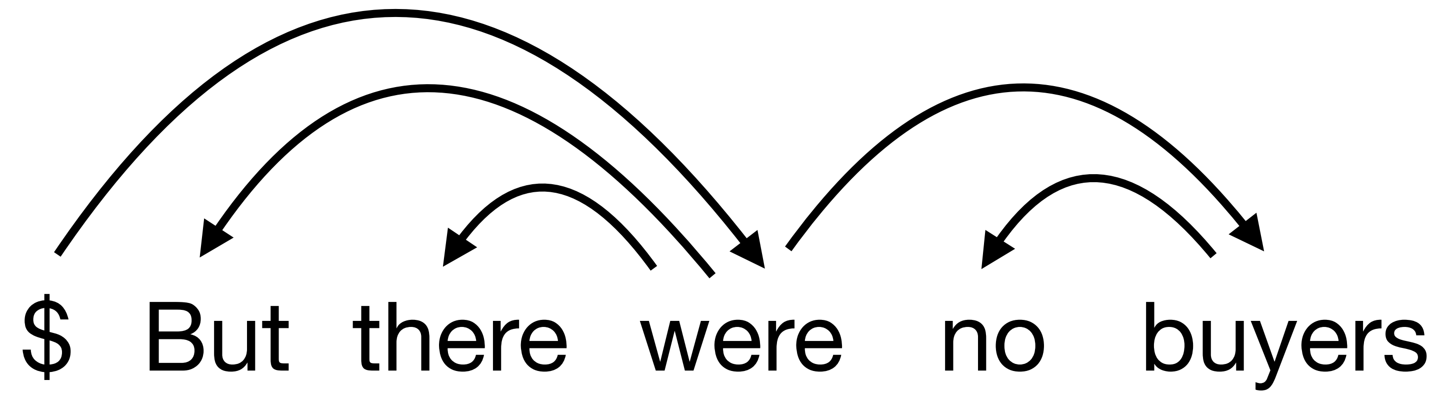


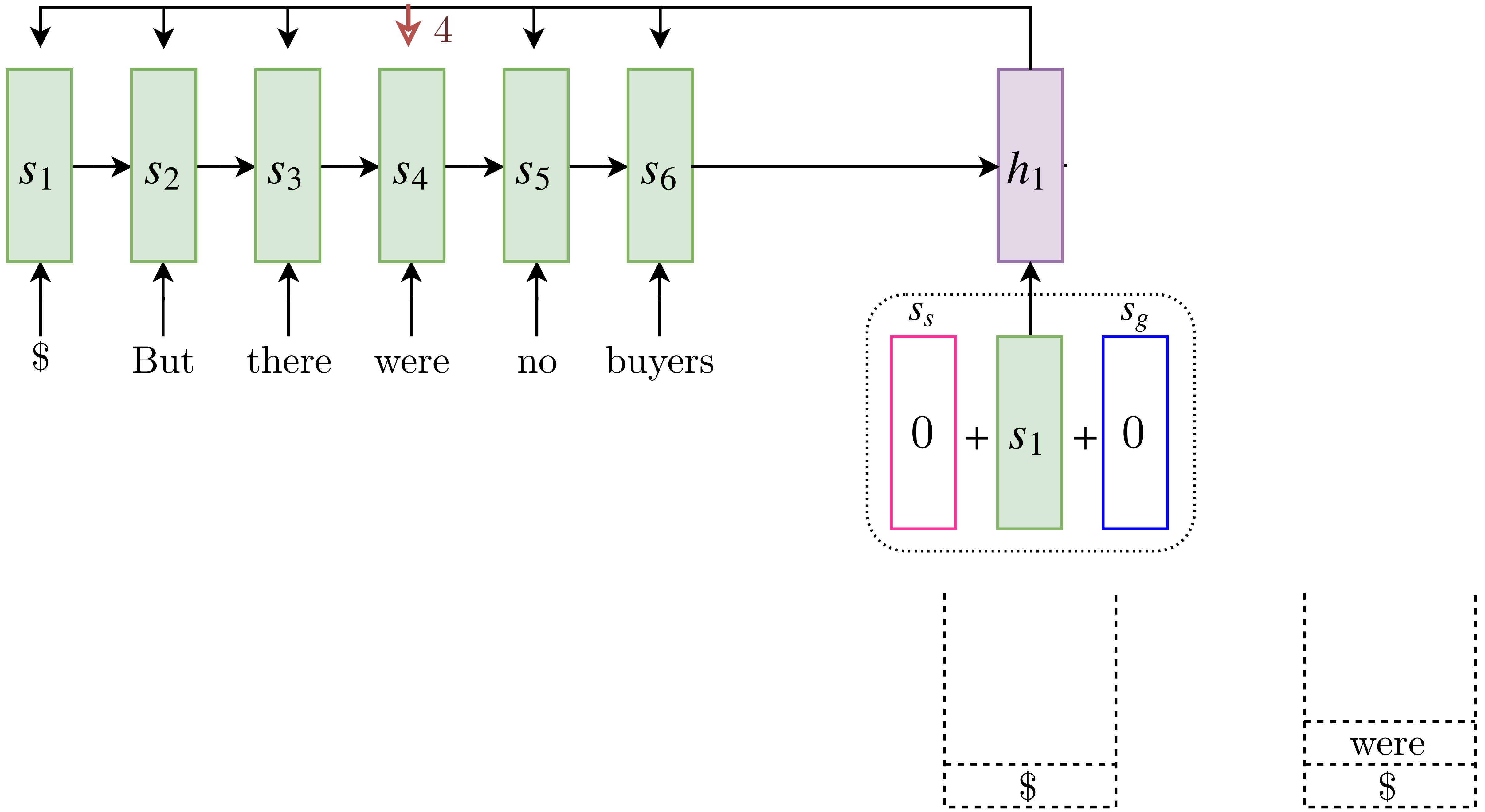
StackPtr: An Example



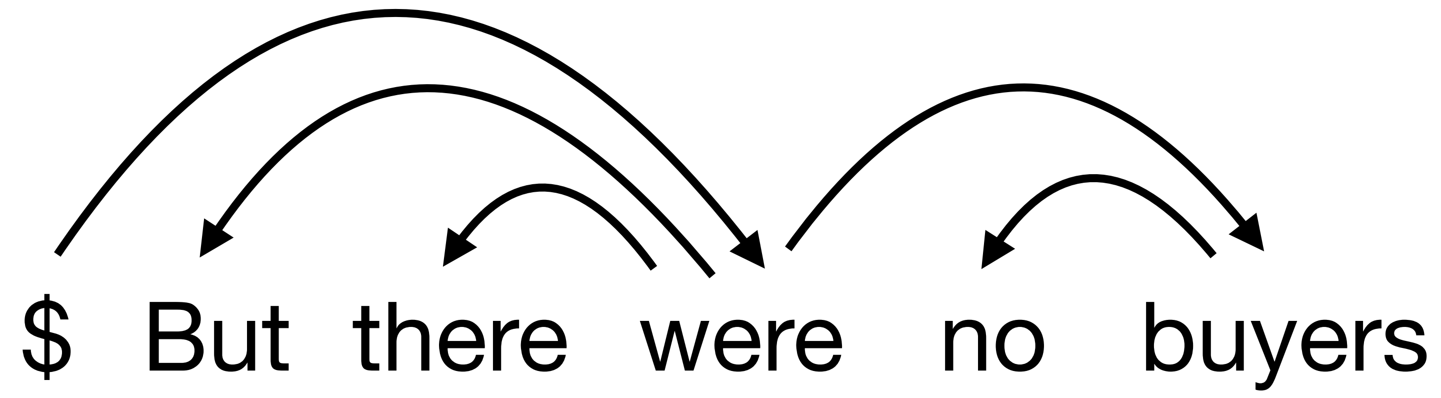


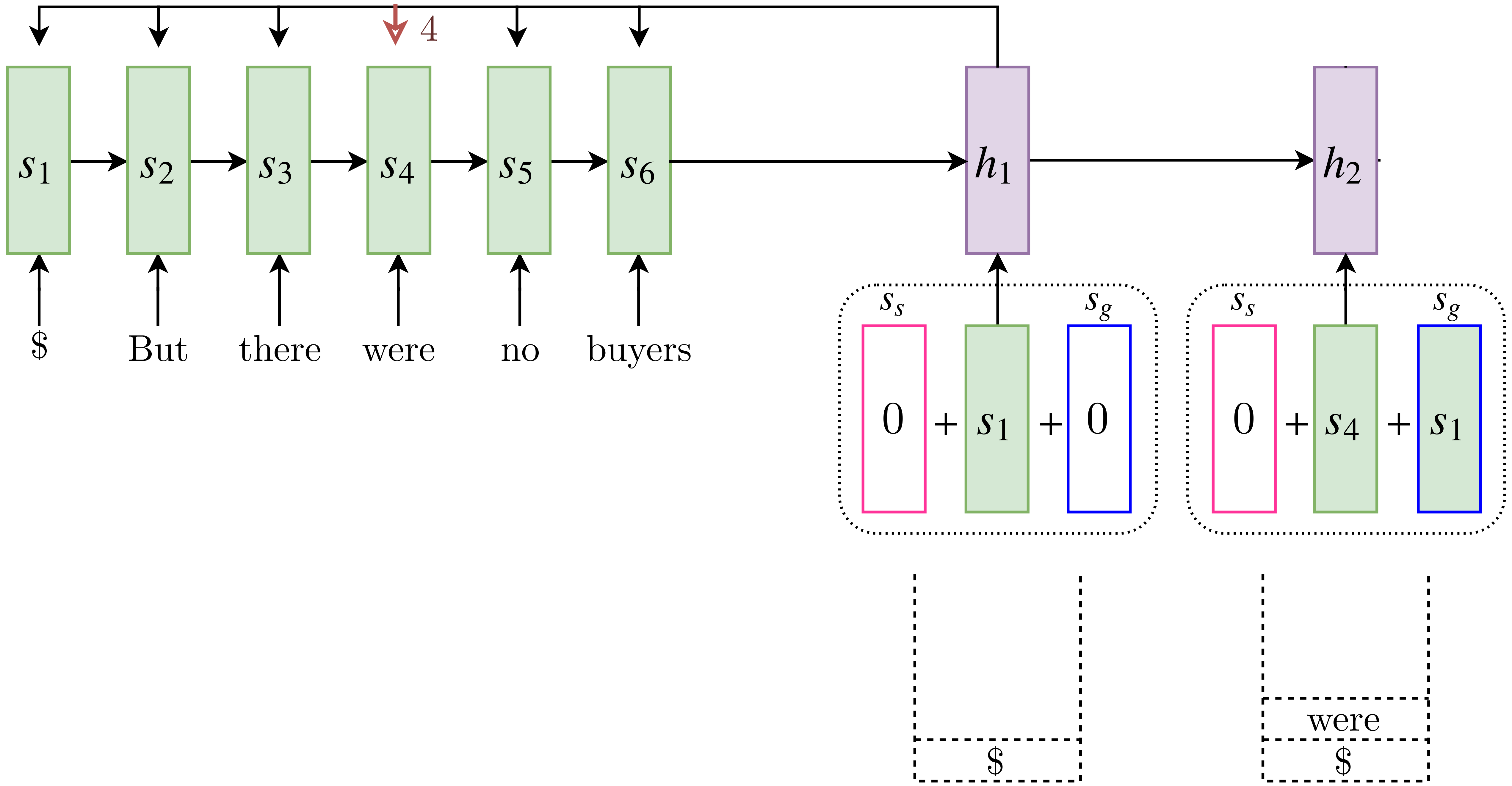
StackPtr: An Example



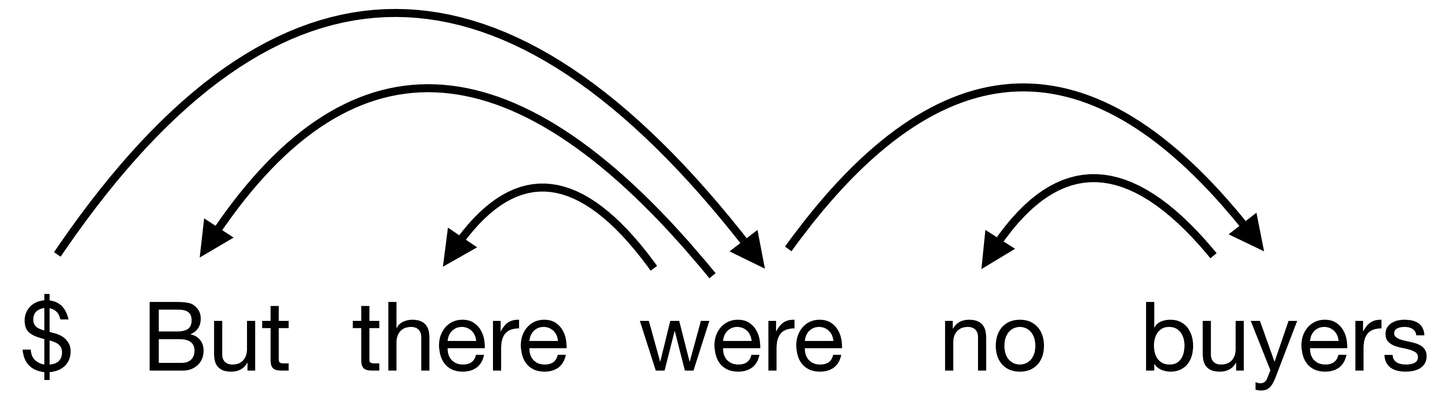


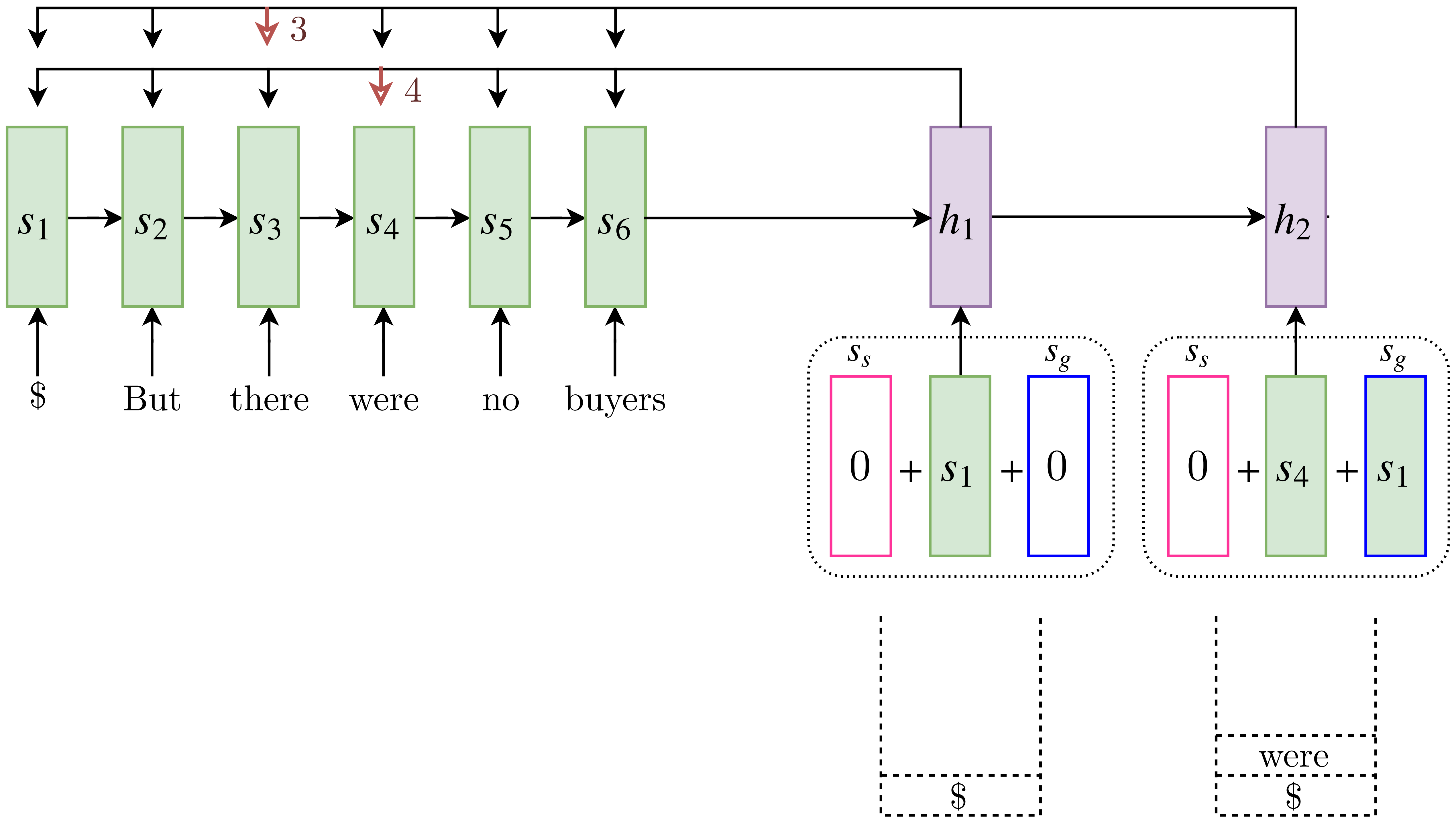
StackPtr: An Example



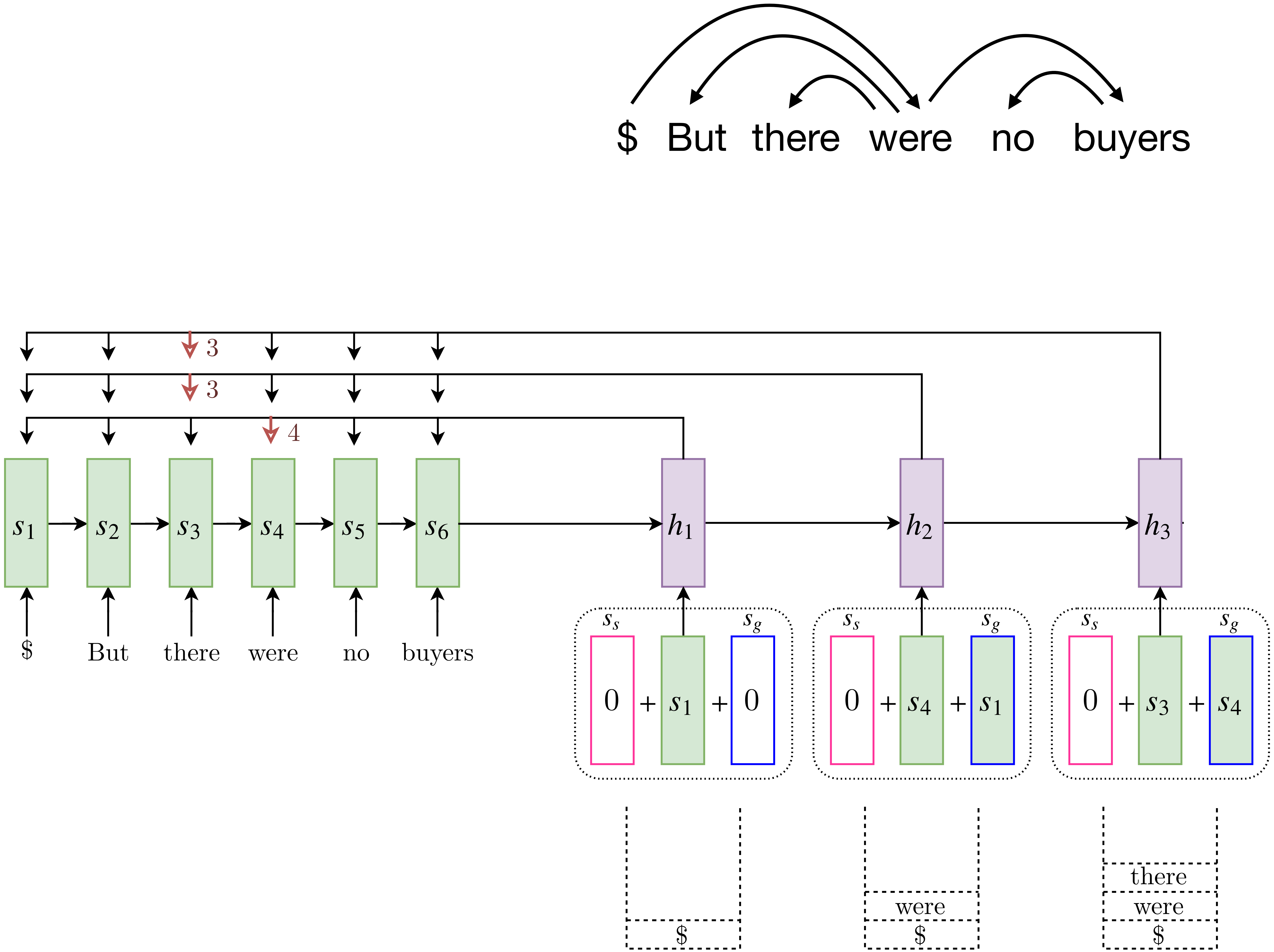


StackPtr: An Example

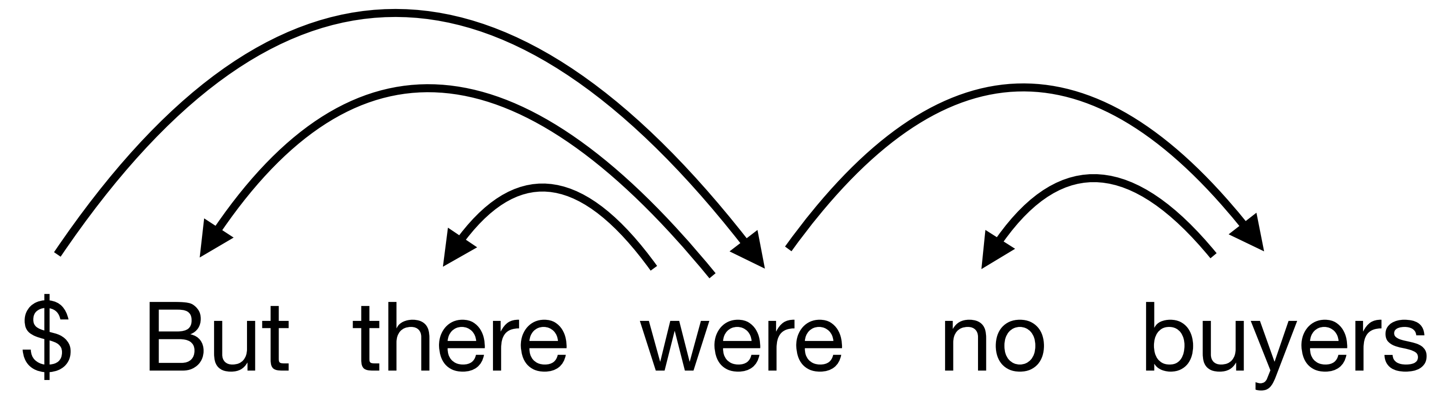


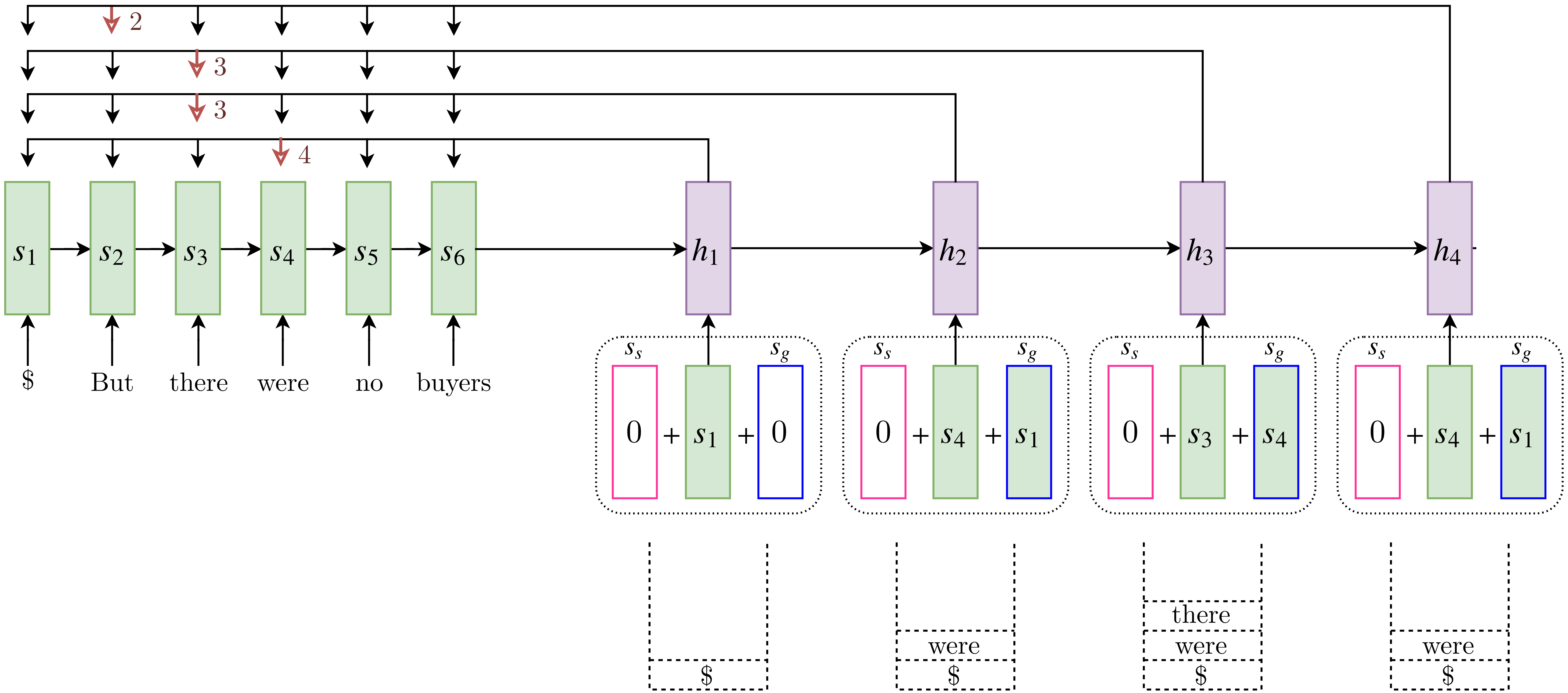


StackPtr: An Example

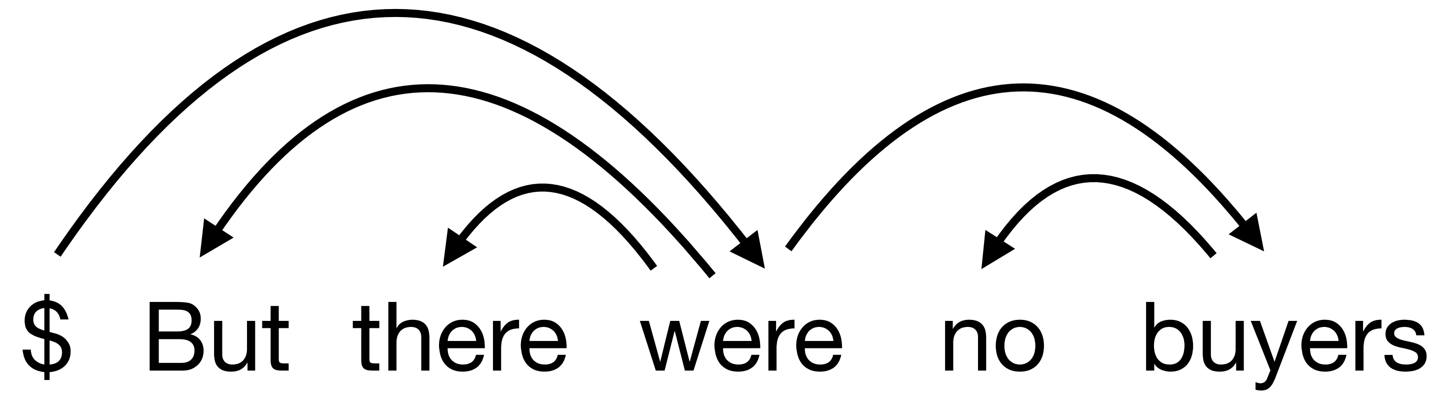


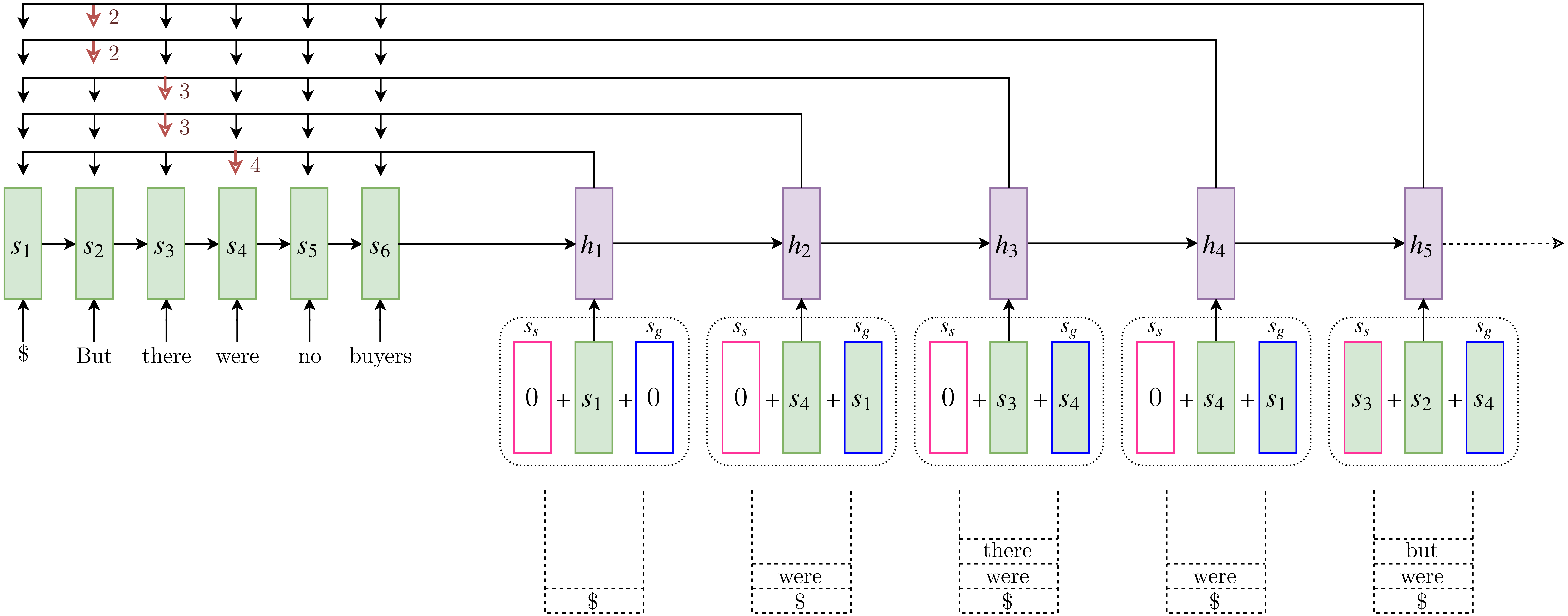
StackPtr: An Example





StackPtr: An Example





Transition System in StackPtr

- **Two data structures**
  - **List (α):** of words whose head has not been selected
  - **Stack (σ):** of partially processed head words whose children have not been fully selected
- **Stack σ is initialized with the root symbol** $\$$
- **At each decoding step** t
  - receive the top element of stack σ as head word w_h, and generate the hidden state h_t
  - compute the vector a^t using h_t and encoder hidden states s
  - **generate an arc:** choose a specific word (w_c) from α as the child of w_h , remove w_c from α and push it onto σ
  - **complete a head:** pop w_h out of σ

Dependency Parsing

- **DeepBiAffine Parser**

	English	German
4th-proj	93.4%	89.3%
BiAffine	94.1%	91.6%
BiAffine+CNNs	94.9%	93.4%
NeuroMST	95.8%	93.8%
StackPtr	95.9%	93.7%

Carnegie Mellon

Representation Transfer in Deep Learning

USC Viterbi
Information Sciences Institute

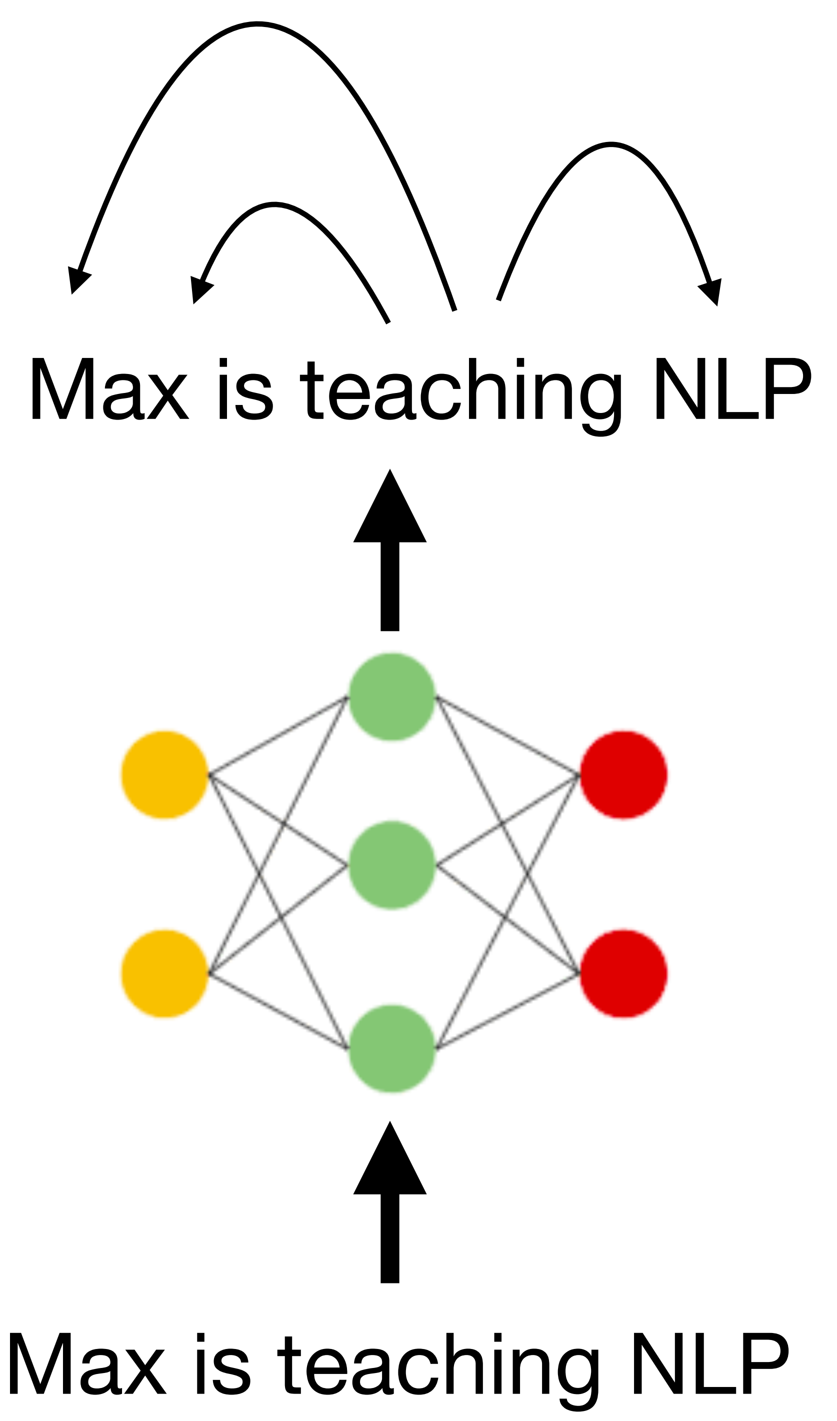
USC University of Southern California

An Interesting Observation

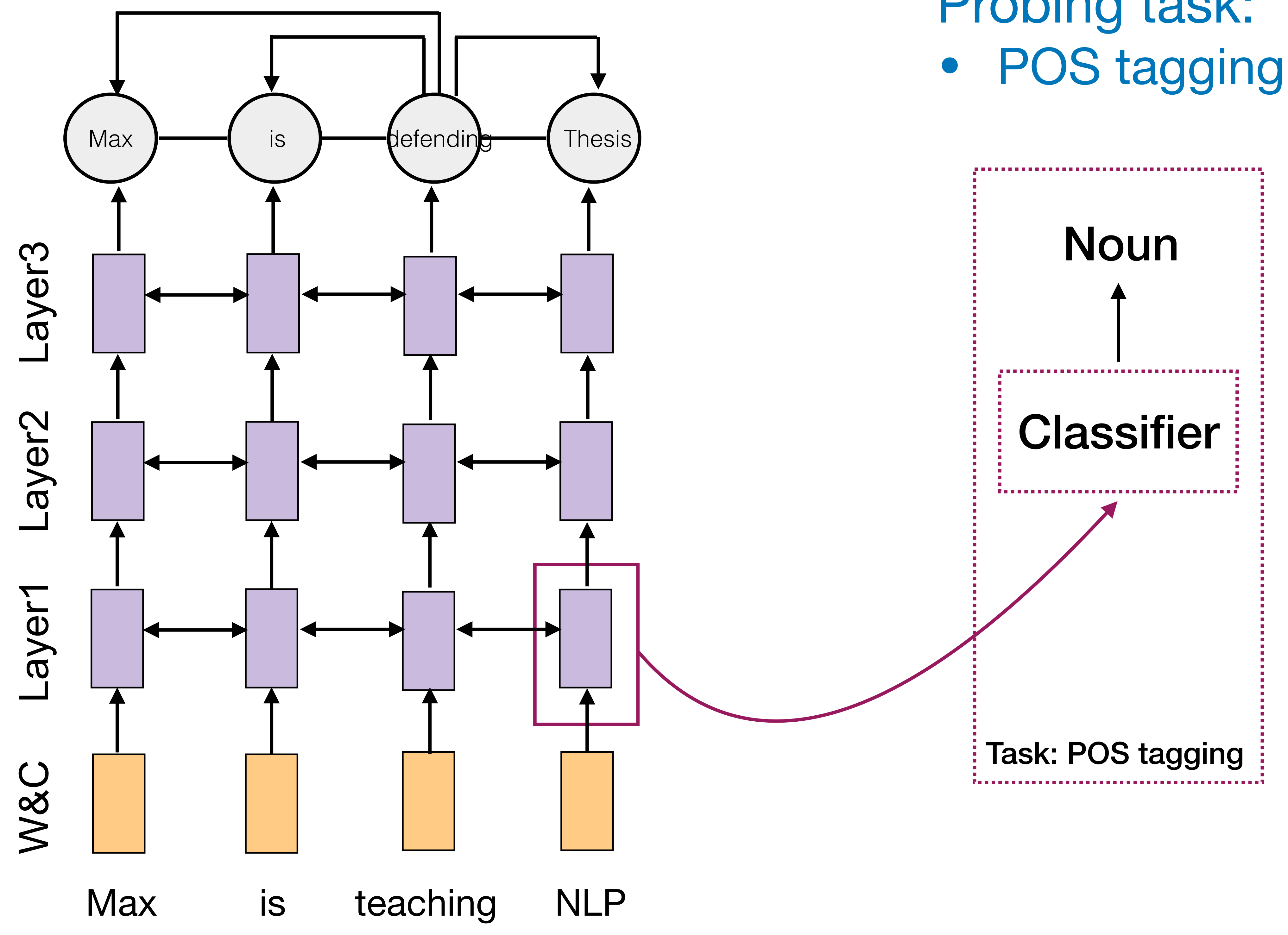
- **A Probing Experiment**

Train end-to-end Neural Network → Generate feature representations → Train and evaluate classifier

Main task: Dependency Parsing



Probing task:
- POS tagging



An Interesting Observation

	POS Tagging
BLSTM-CNN-CRF	97.6%
LSTM1 + SVM	97.7%
LSTM2 + SVM	97.8%

Neural Representations learned from a more challenging tasks can be applied to down-stream tasks!

Reading Materials

- **Comparison and Integration of graph-based and transition-based dependency parsers**
  - McDonald and Nivre, 2011