

---

# Syntax and Parsing II

## Dependency Parsing

Slav Petrov – Google

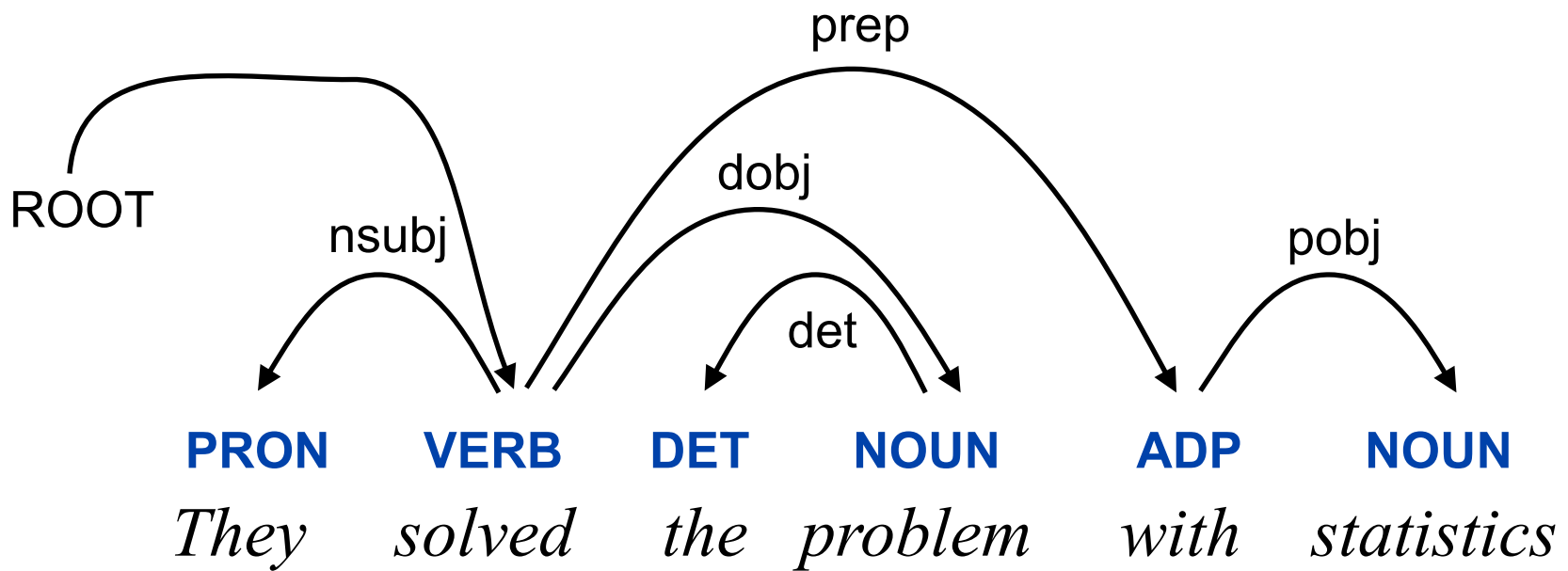
Thanks to:

Dan Klein, Ryan McDonald, Alexander Rush, Joakim Nivre,  
Greg Durrett, David Weiss

Lisbon Machine Learning School 2016

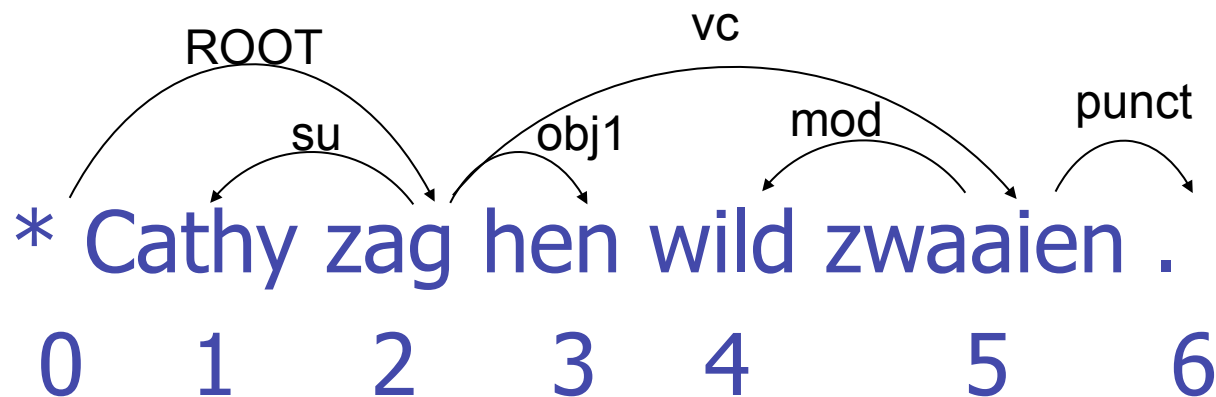
# Dependency Parsing

---



# CoNLL Format

1	Cathy	Cathy	N	N	eigen ev neut	2	su
2	zag	zie	V	V	trans ovt 1of2of3 ev	0	ROOT
3	hen	hen	Pron	Pron	per 3 mv datofacc	2	obj1
4	wild	wild	Adj	Adj	attr stell onverv	5	mod
5	zwaaien	zwaai	N	N	soort mv neut	2	vc
6	.	.	Punc	Punc	punct	5	punct

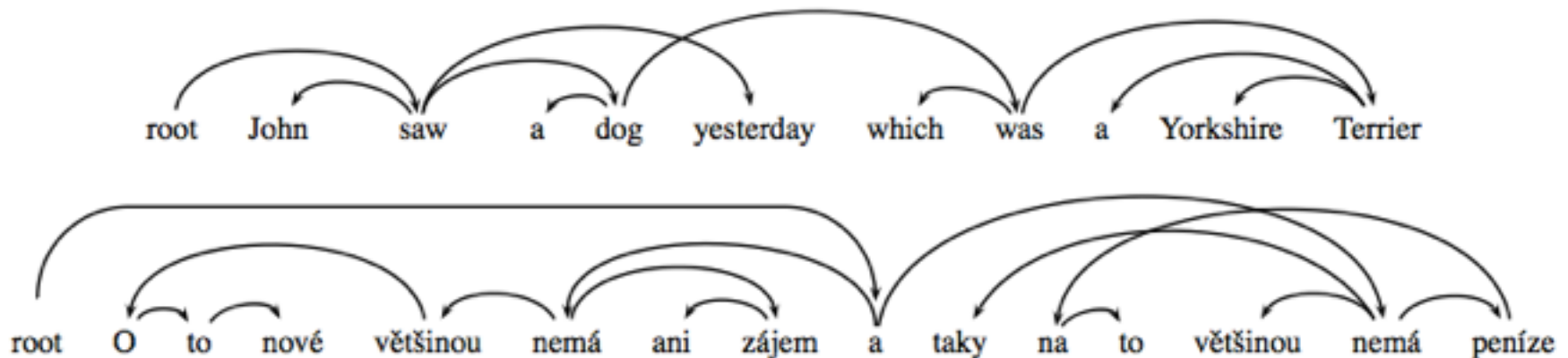


<http://ilk.uvt.nl/conll/>

# (Non-)Projectivity

---

- Crossing Arcs needed to account for non-projective constructions
- Fairly rare in English but can be common in other languages (e.g. Czech):



*He is mostly not even interested in the new things and in most cases, he has no money for it either.*

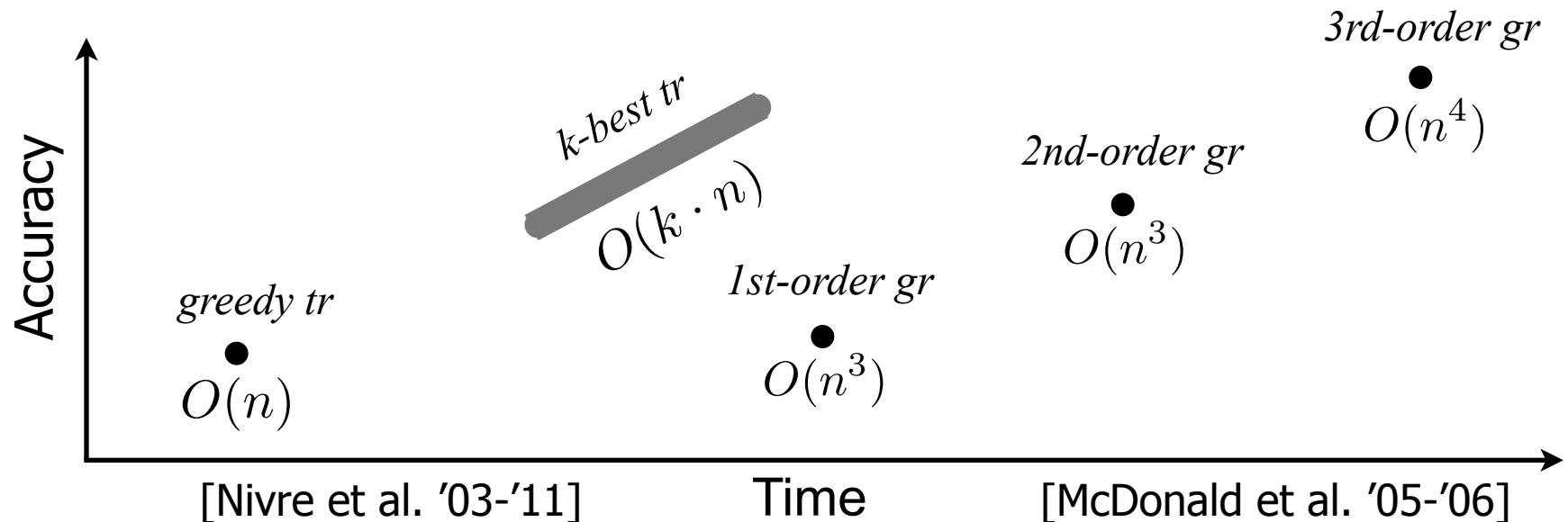
# Formal Conditions

---

- ▶ For a dependency graph  $G = (V, A)$
- ▶ With label set  $L = \{l_1, \dots, l_{|L|}\}$
- ▶  $G$  is (weakly) **connected**:
  - ▶ If  $i, j \in V$ ,  $i \leftrightarrow^* j$ .
- ▶  $G$  is **acyclic**:
  - ▶ If  $i \rightarrow j$ , then not  $j \rightarrow^* i$ .
- ▶  $G$  obeys the **single-head** constraint:
  - ▶ If  $i \rightarrow j$ , then not  $i' \rightarrow j$ , for any  $i' \neq i$ .
- ▶  $G$  is **projective**:
  - ▶ If  $i \rightarrow j$ , then  $i \rightarrow^* i'$ , for any  $i'$  such that  $i < i' < j$  or  $j < i' < i$ .

# Styles of Dependency Parsing

- Transition-Based (tr)
  - Fast, greedy, linear time inference algorithms
  - Trained for greedy search
  - Beam search
- Graph-Based (gr)
  - Slower, exhaustive, dynamic programming inference algorithms
  - Higher-order factorizations



# Arc-Factored Models

---

- ▶ Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

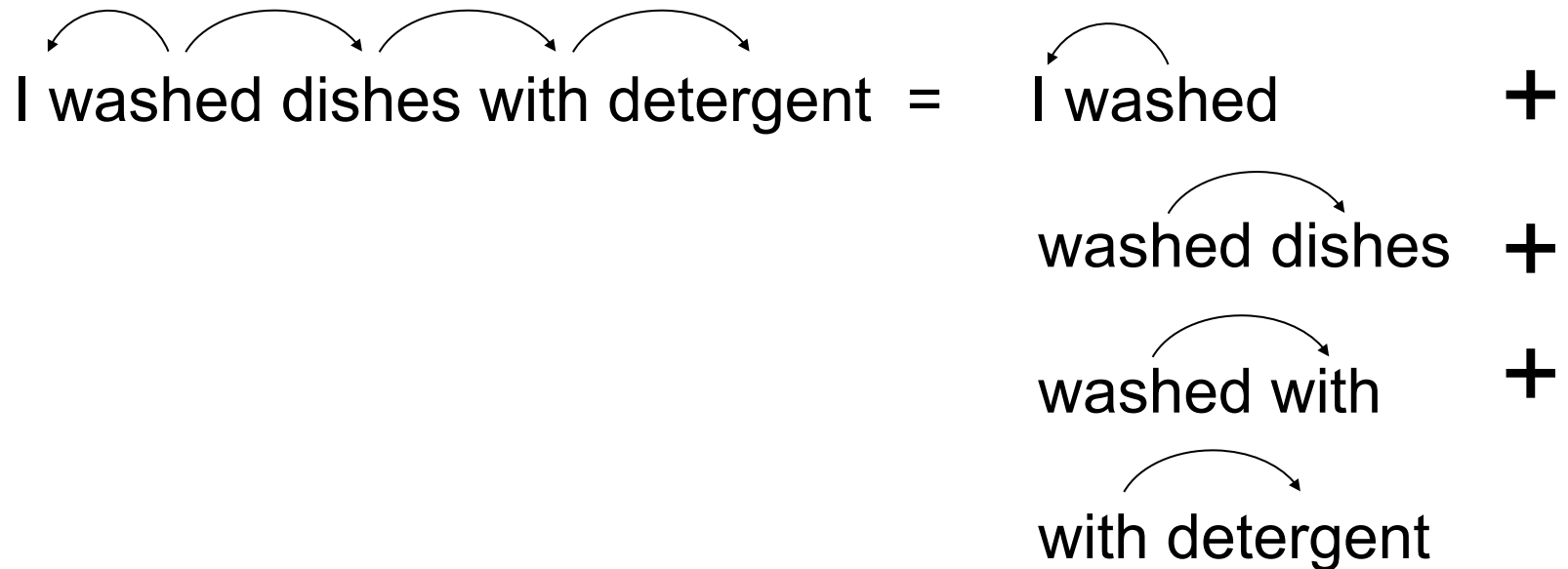
$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k \quad \text{look familiar?}$$

- ▶  $w_{ij}^k$  is the weight of creating a dependency from word  $w_i$  to  $w_j$  with label  $l_k$
- ▶ Thus there is an assumption that each dependency decision is independent
  - ▶ Strong assumption! Will address this later.

# Graph-based Parsing

---

- Assumes that scores factor over the tree
- Arc-factored models
  - $\text{Score}(\text{tree}) = \sum \text{edges}$



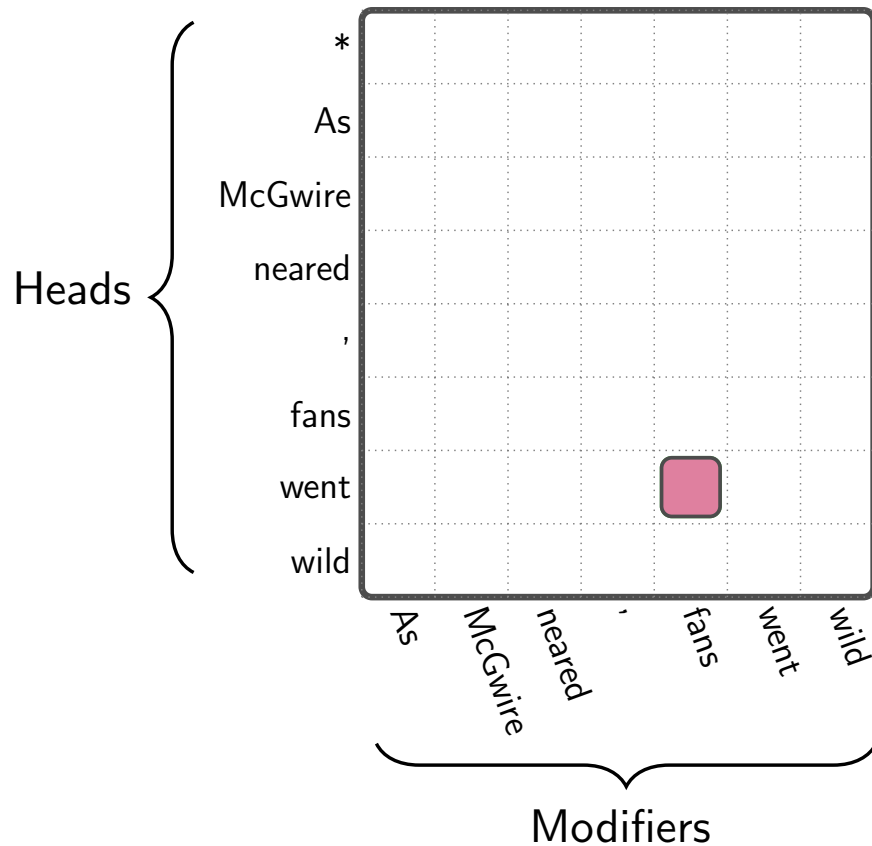





# Dependency Representation

---

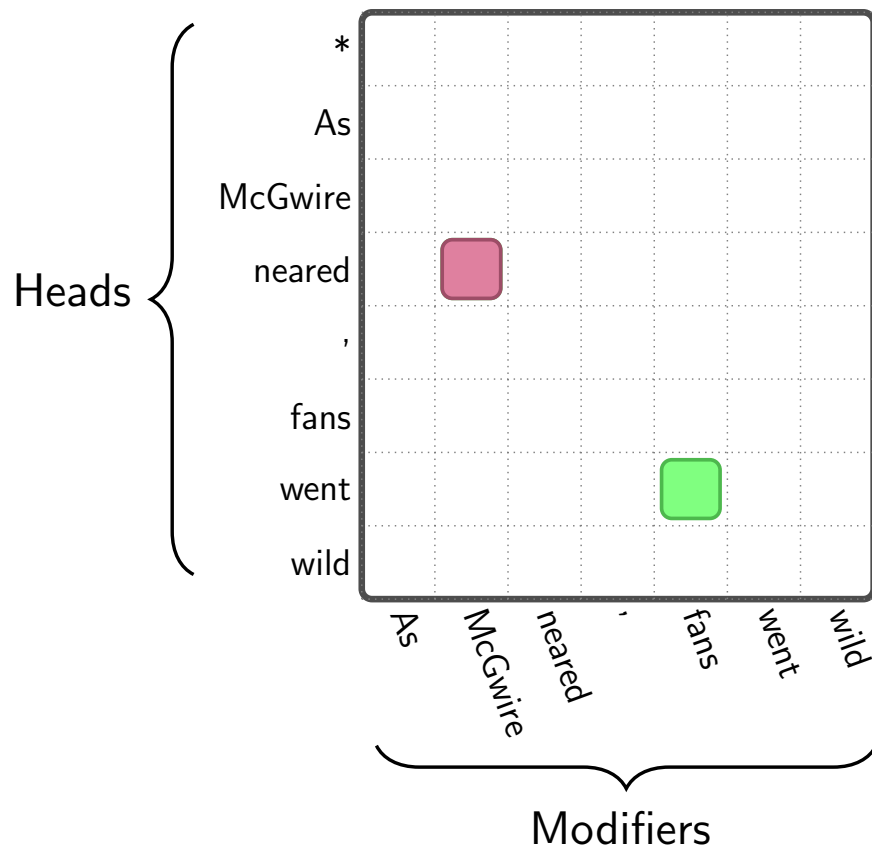
\* As McGwire neared , fans went wild



# Dependency Representation

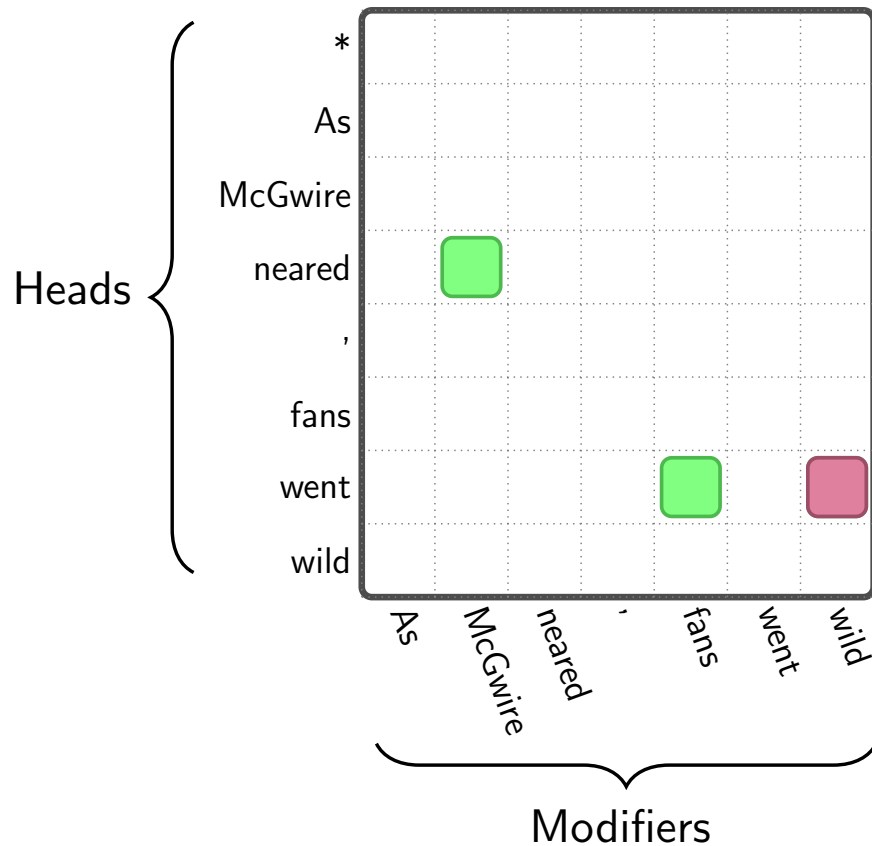
---

\* As McGwire neared , fans went wild



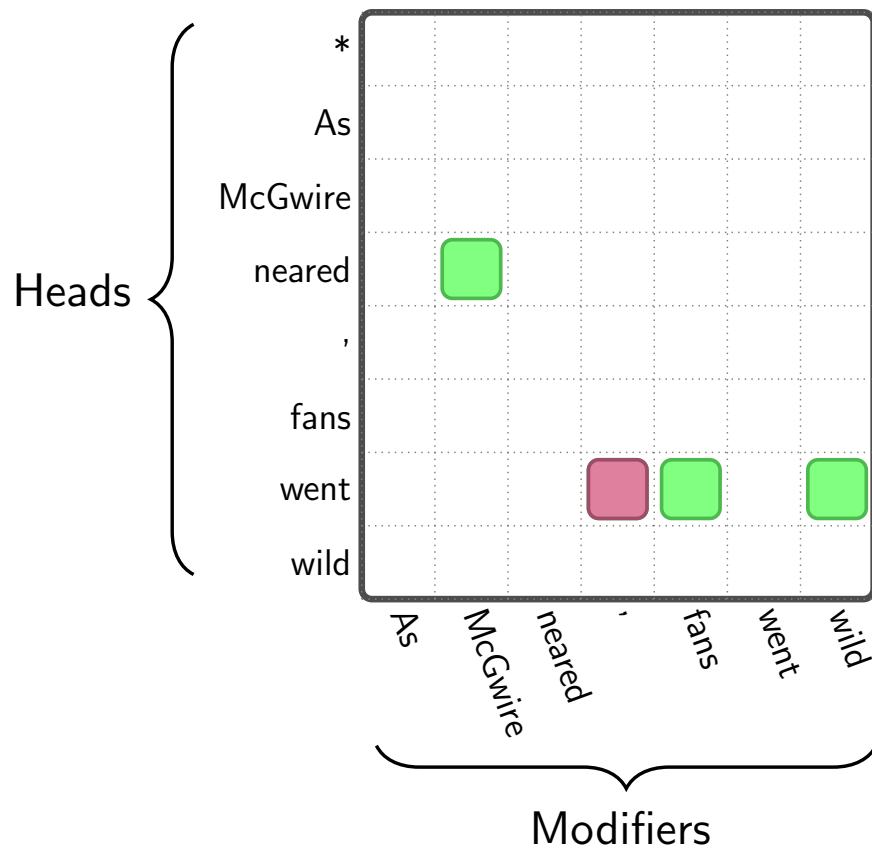
# Dependency Representation

\* As McGwire neared , fans went wild

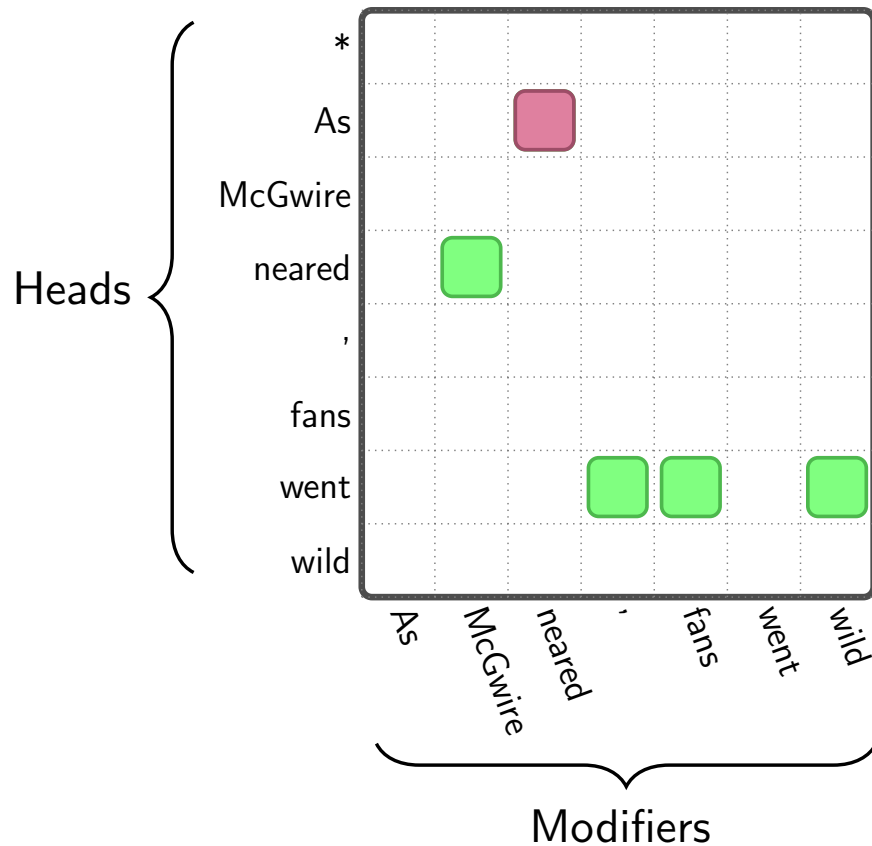
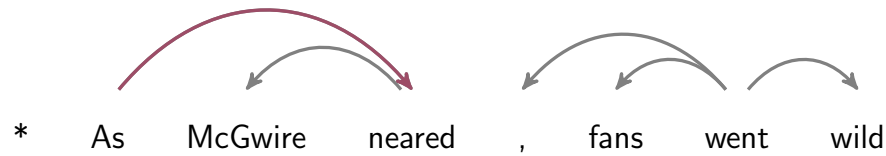


# Dependency Representation

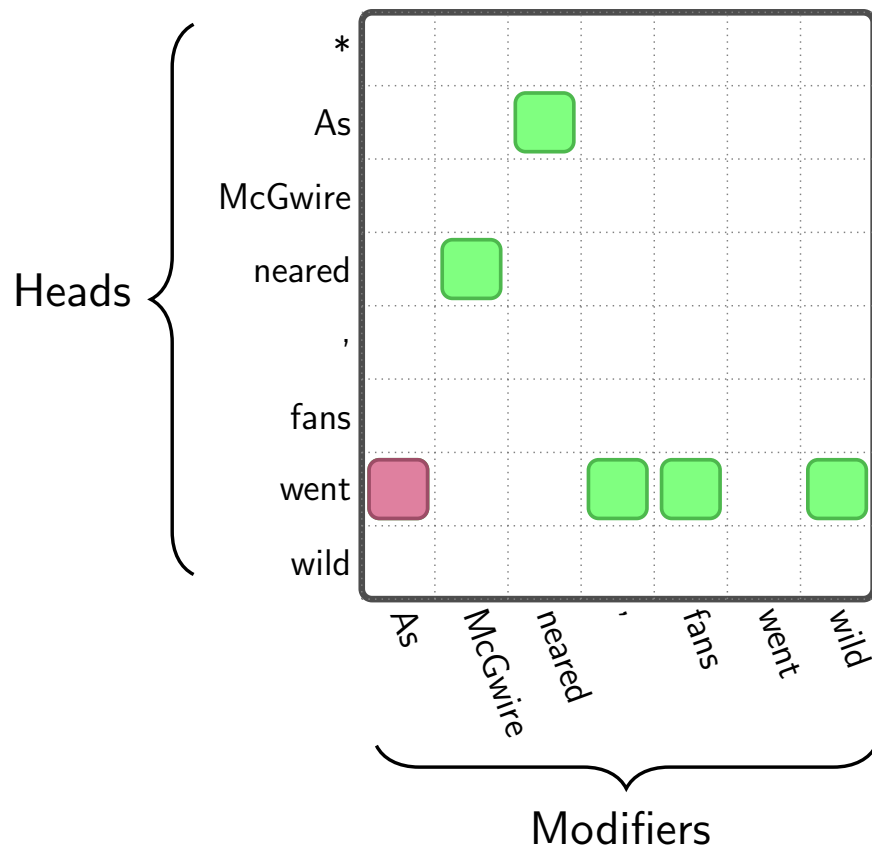
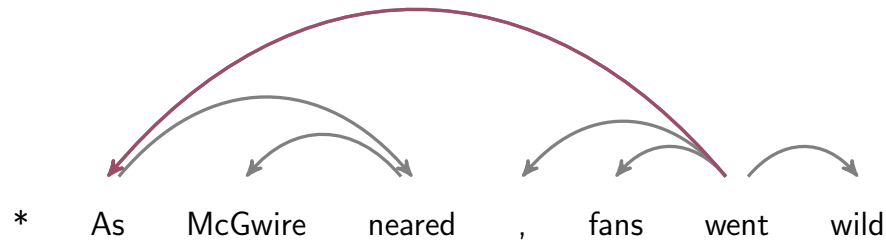
\* As McGwire neared , fans went wild



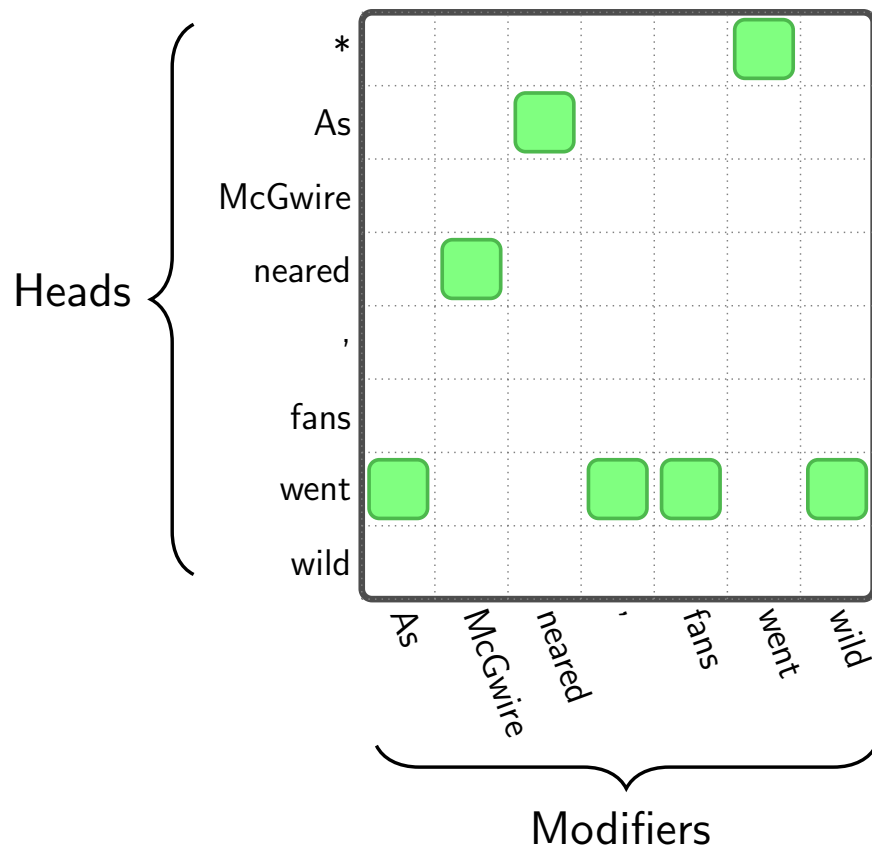
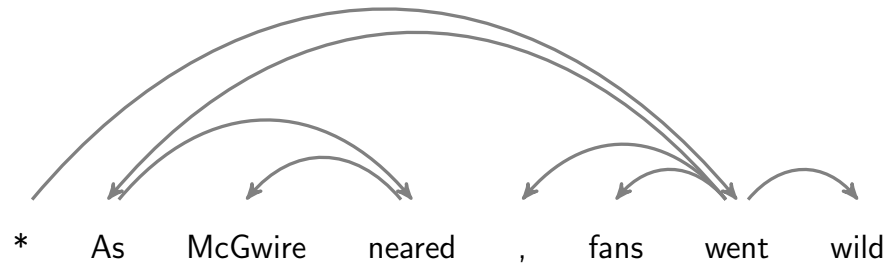
# Dependency Representation



# Dependency Representation



# Dependency Representation





# Graph-Based Parsing

Searching

Scoring

$$y^* = \operatorname{argmax}_{y \in Y}$$

$$\sum_{p \in P(y)} w \cdot \phi(p, x)$$

Parsing algorithm

Set of possible trees

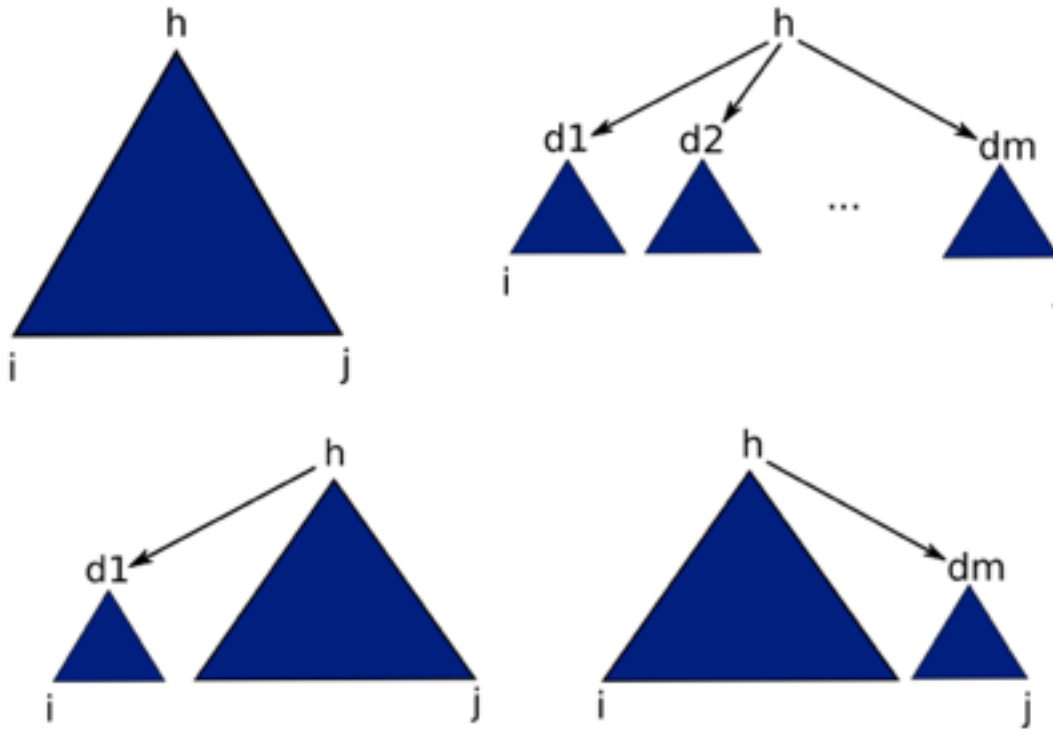
Factorization: local parts of the tree

Features

# Arc-factored Projective Parsing

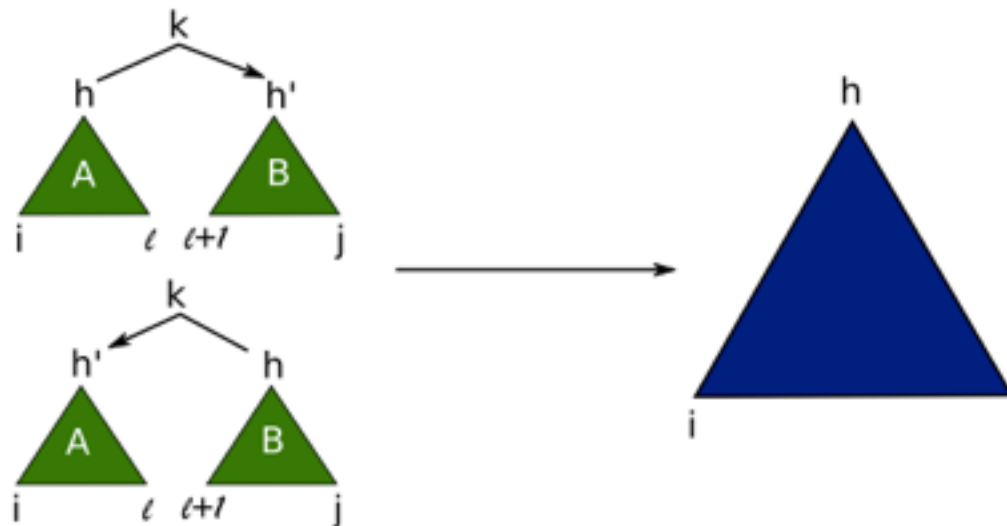
---

- ▶ All projective graphs can be written as the combination of two smaller **adjacent** graphs



# Arc-factored Projective Parsing

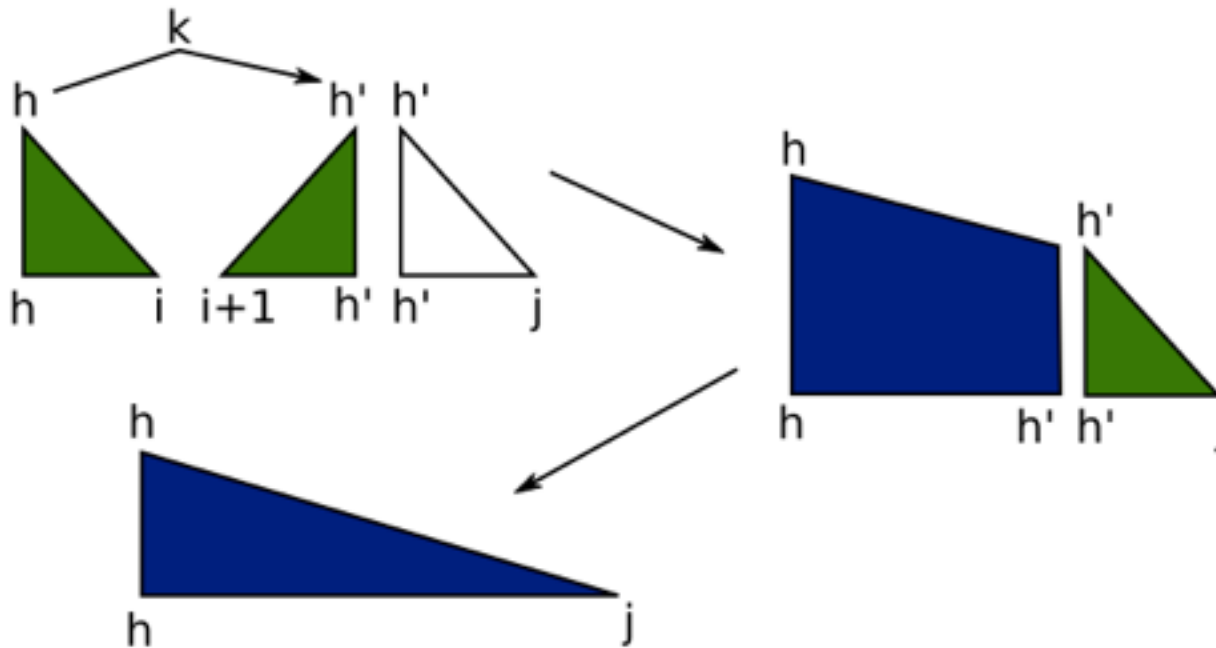
- ▶ Chart item filled in a bottom-up manner
  - ▶ First do all strings of length 1, then 2, etc. just like CKY



- ▶ Weight of new item:  $\max_{l,j,k} w(A) \times w(B) \times w_{hh'}^k$
- ▶ Algorithm runs in  $O(|L|n^5)$
- ▶ Use back-pointers to extract best parse (like CKY)

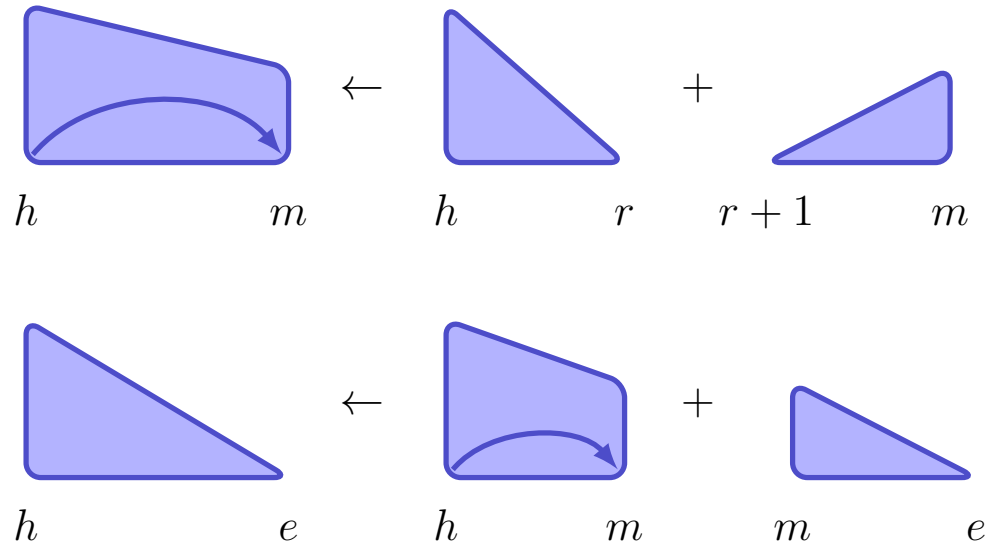
# Eisner Algorithm

- ▶  $O(|L|n^5)$  is not that good
- ▶ [Eisner 1996] showed how this can be reduced to  $O(|L|n^3)$ 
  - ▶ Key: split items so that sub-roots are always on periphery



# Eisner First-Order Parsing

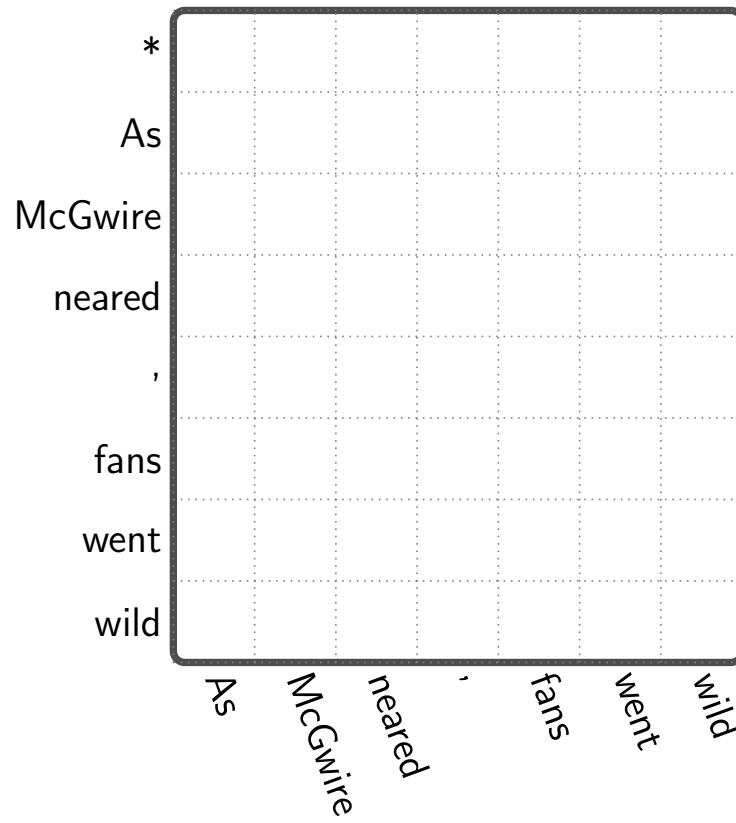
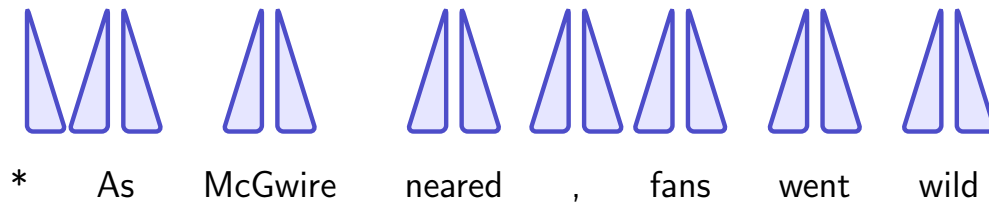
---



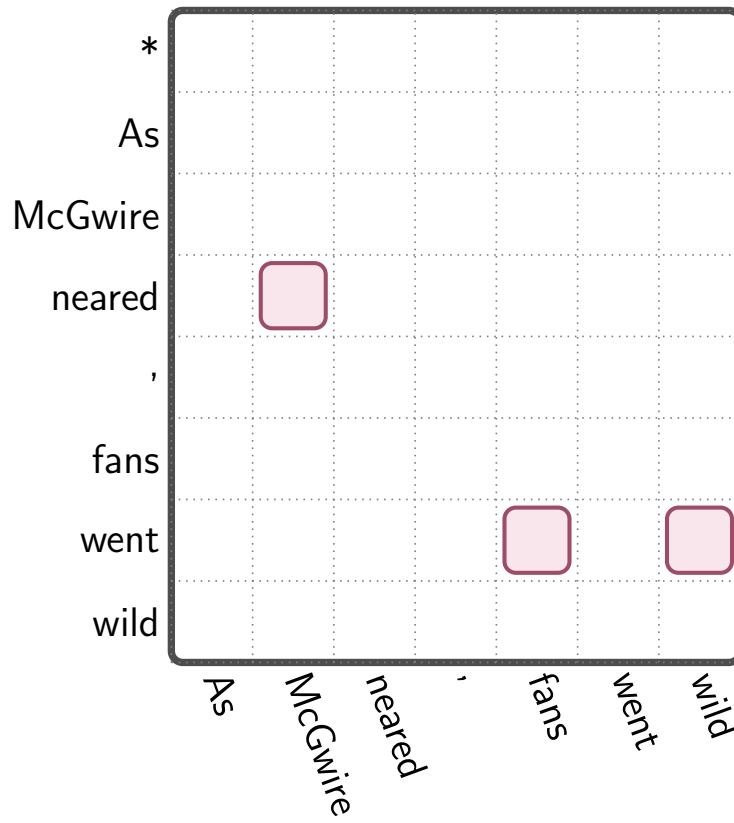
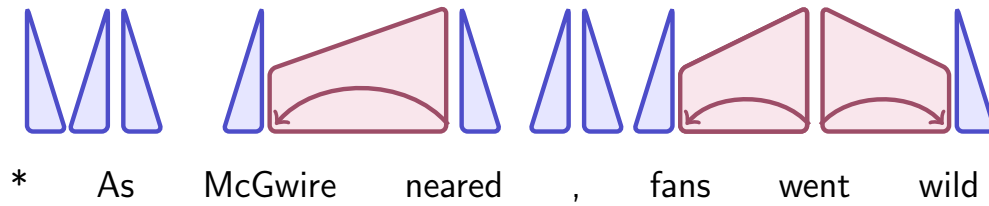
In practice also left arc version

# Eisner First-Order Parsing

---

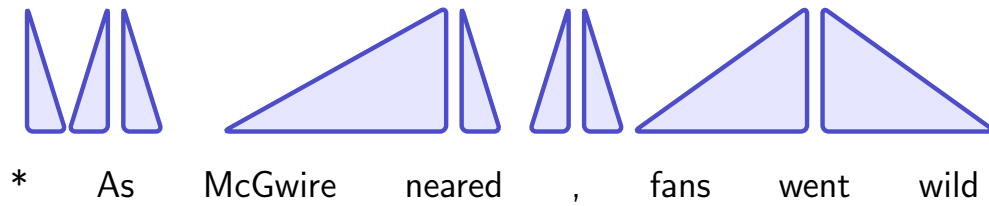


# Eisner First-Order Parsing



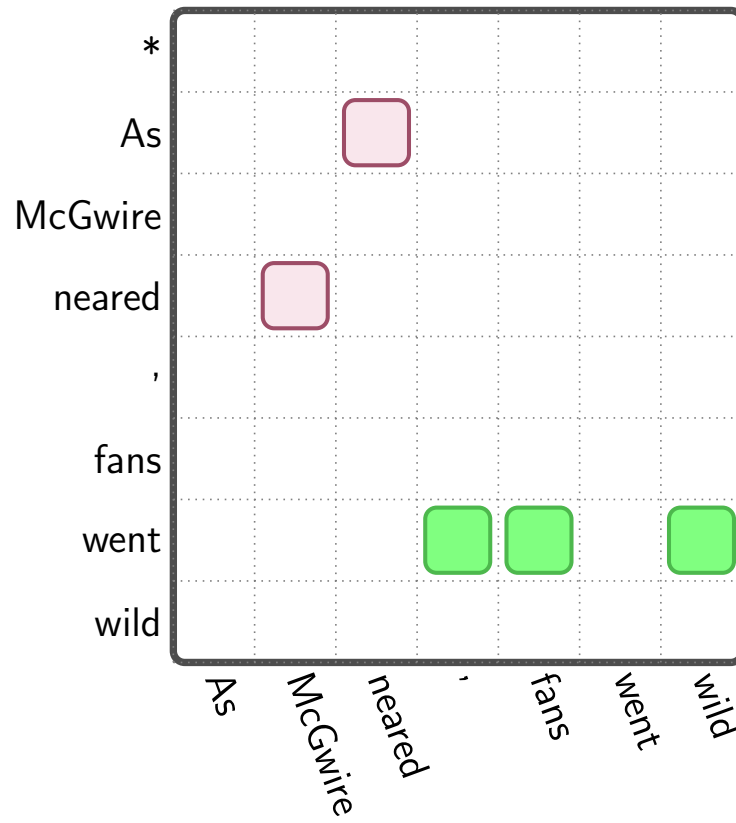
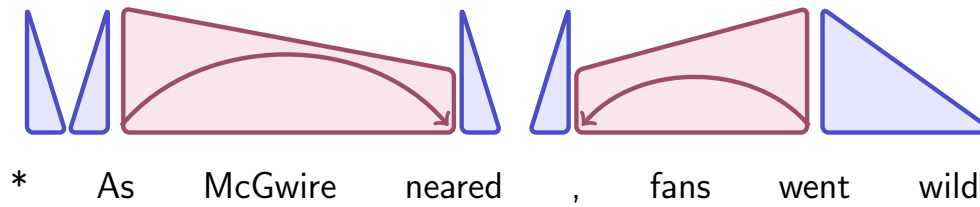
# Eisner First-Order Parsing

---

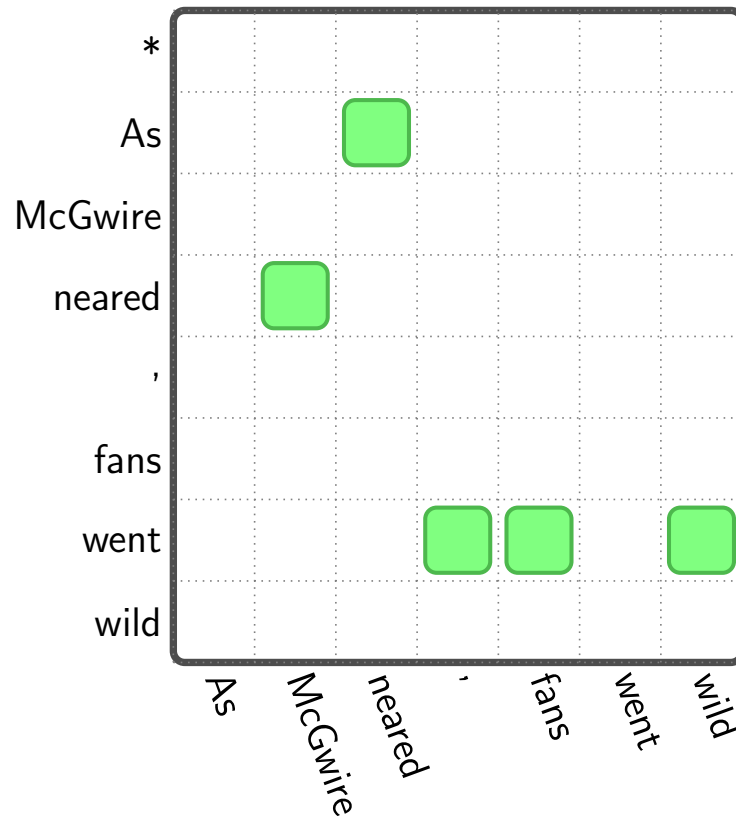
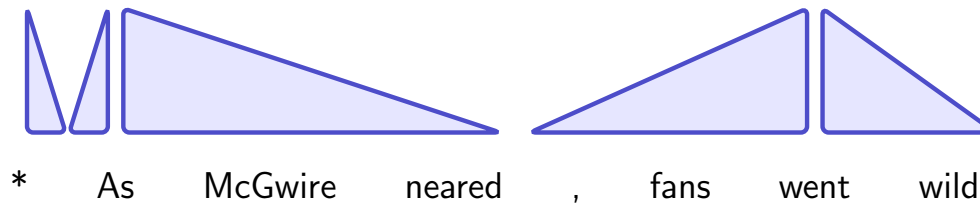




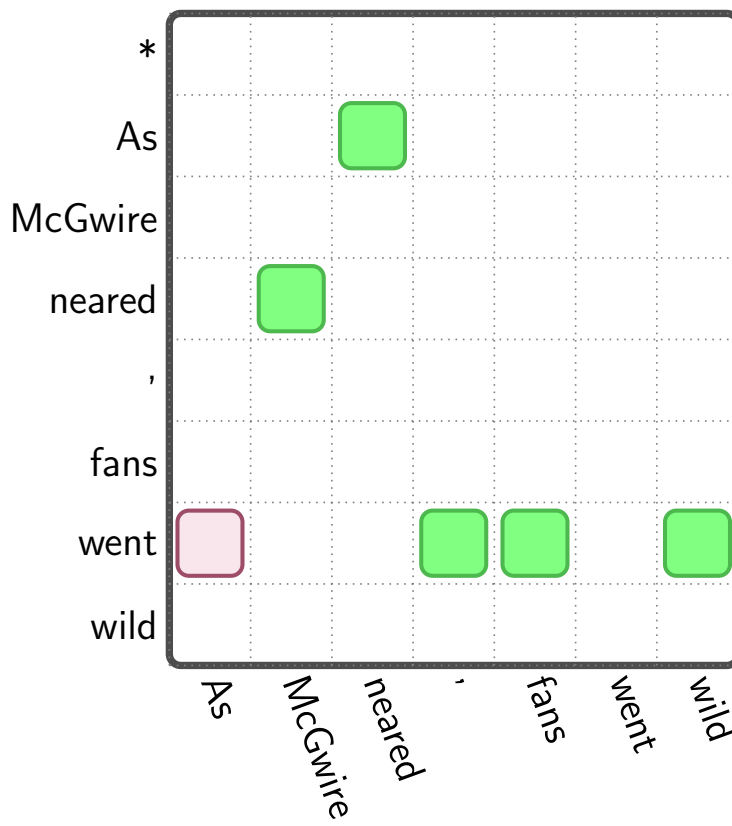
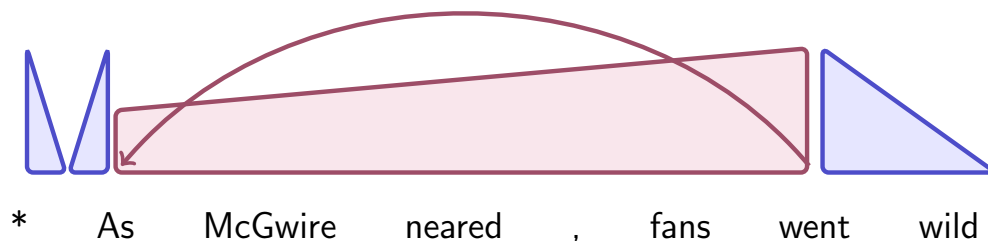
# Eisner First-Order Parsing



# Eisner First-Order Parsing

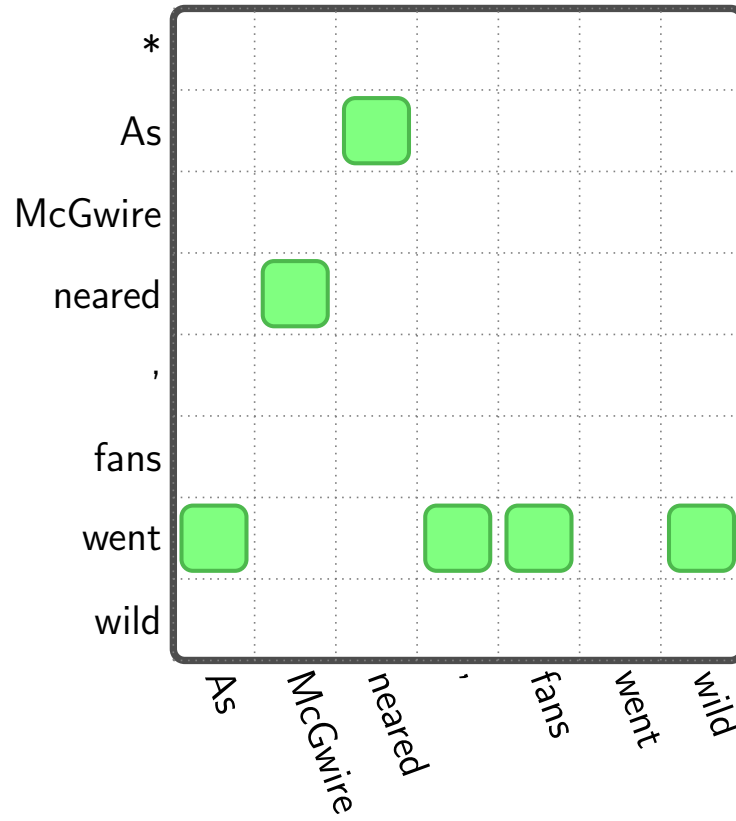
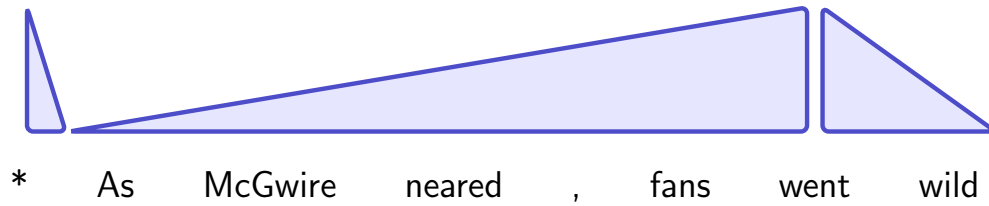


# Eisner First-Order Parsing

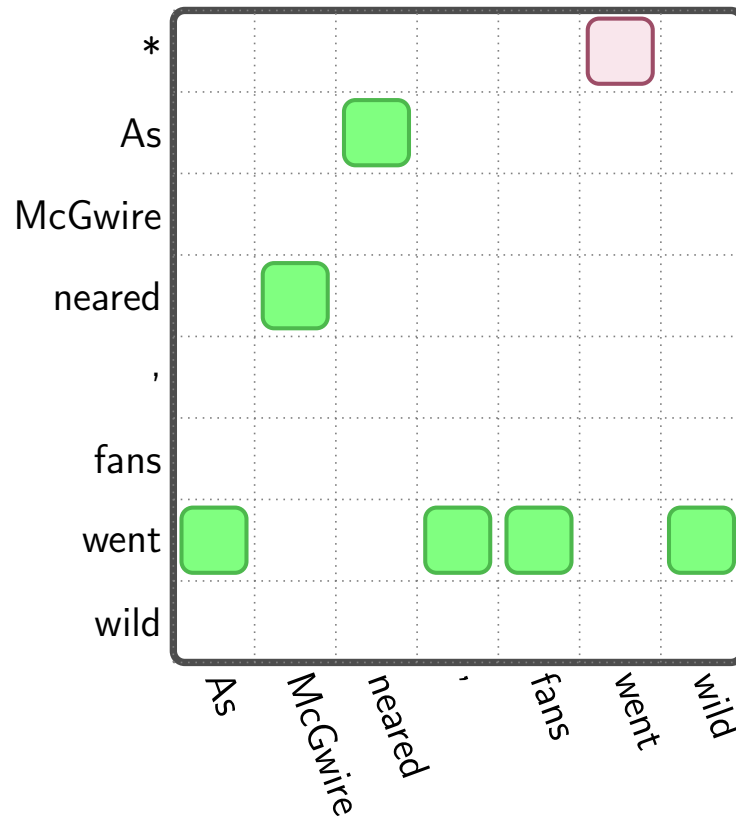
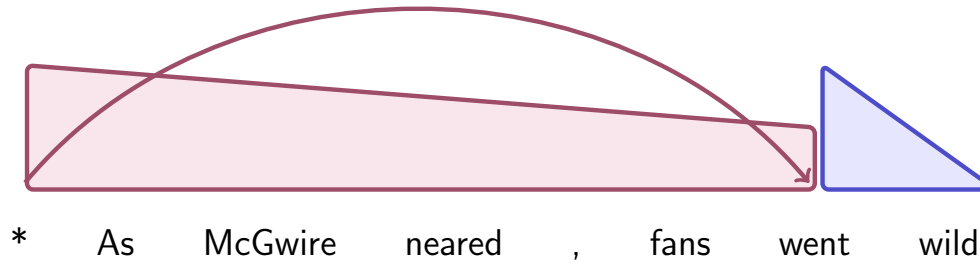


# Eisner First-Order Parsing

---

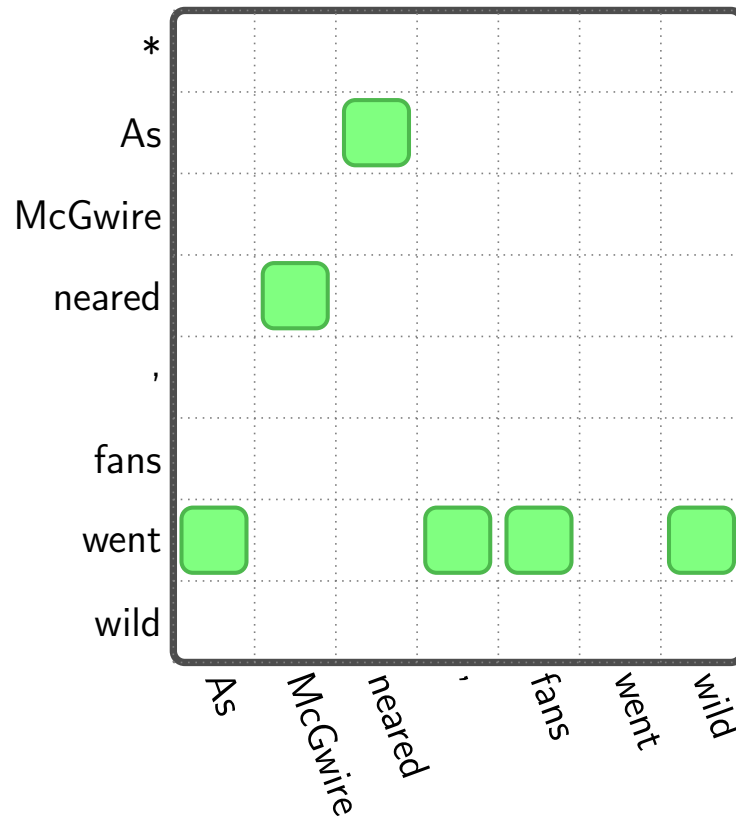
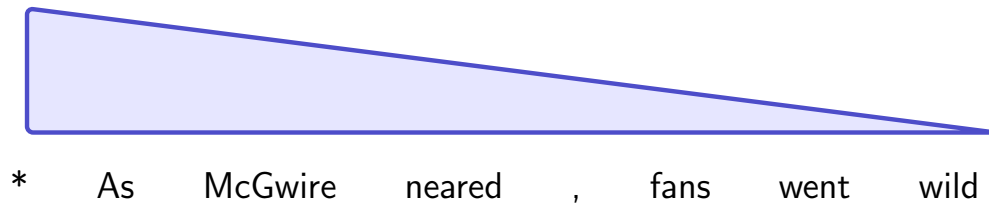


# Eisner First-Order Parsing



# Eisner First-Order Parsing

---



# Eisner Algorithm Pseudo Code

---

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

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

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

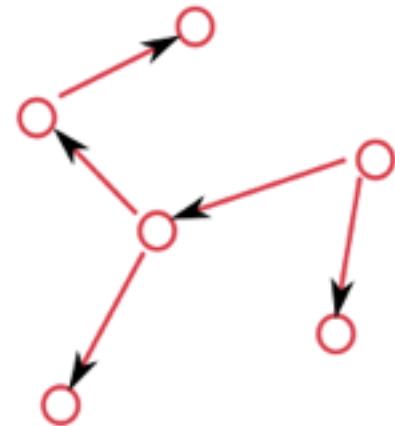
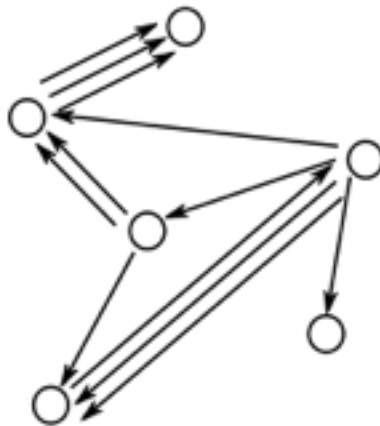
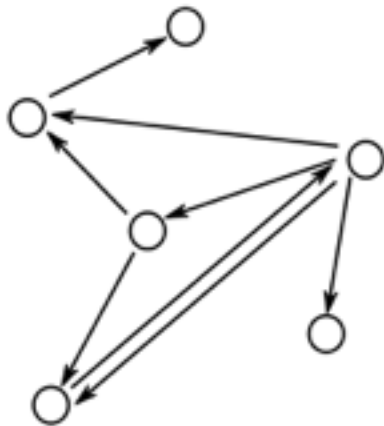
  end for

end for

# Maximum Spanning Trees (MSTs)

---

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs



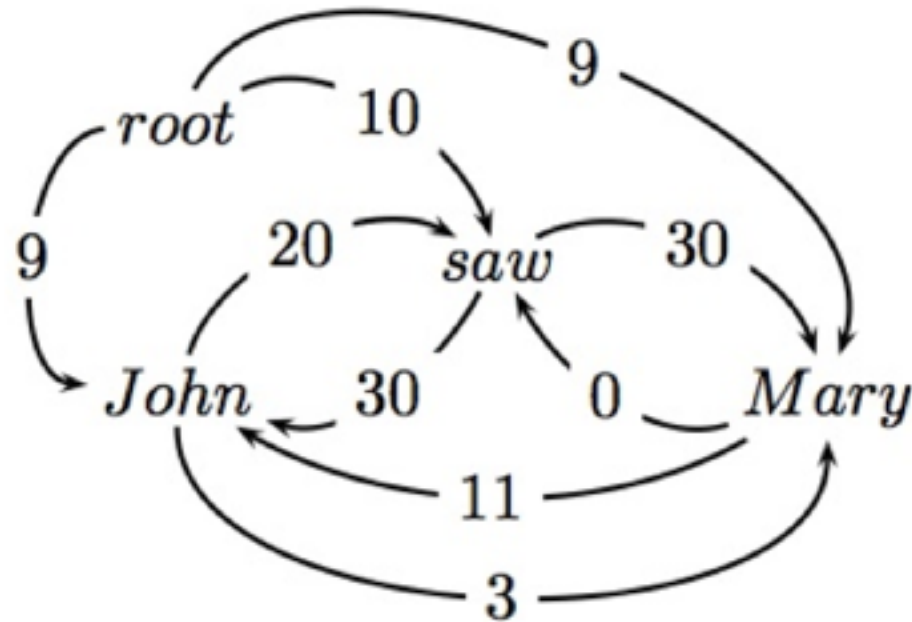
**Can use MST algorithms for nonprojective parsing!**



# Chu-Liu-Edmonds

---

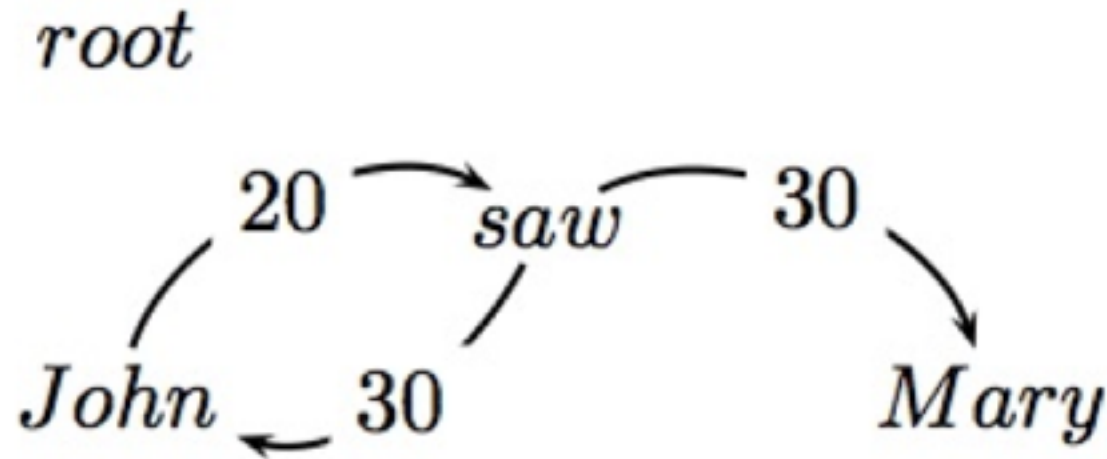
- ▶  $x = \text{root}$  John saw Mary



# Chu-Liu-Edmonds

---

- ▶ Find highest scoring incoming arc for each vertex

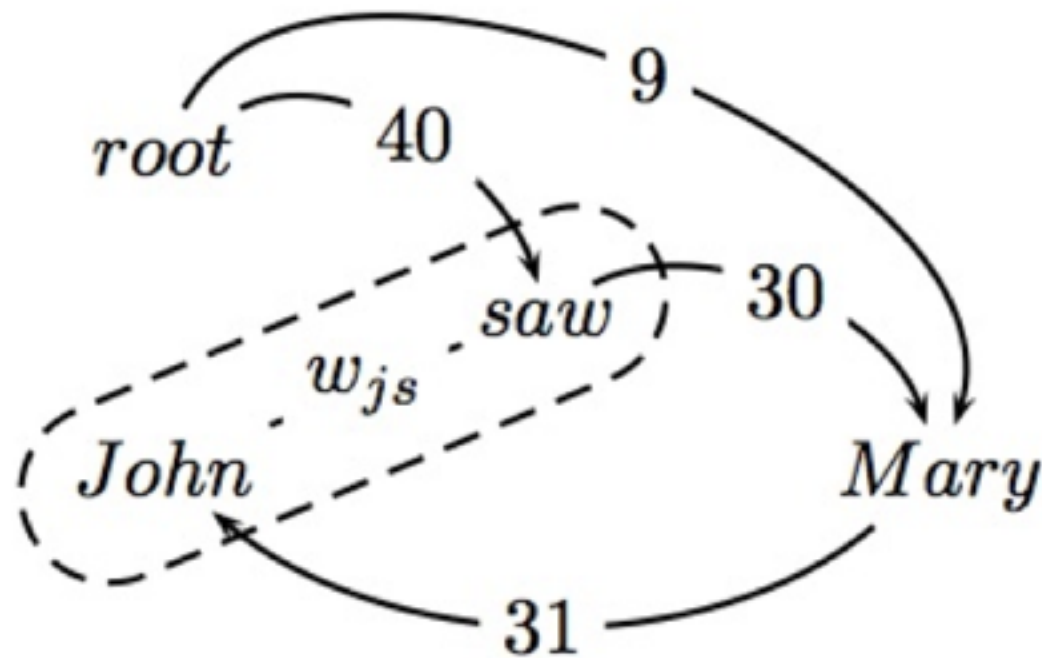


- ▶ If this is a tree, then we have found MST!!

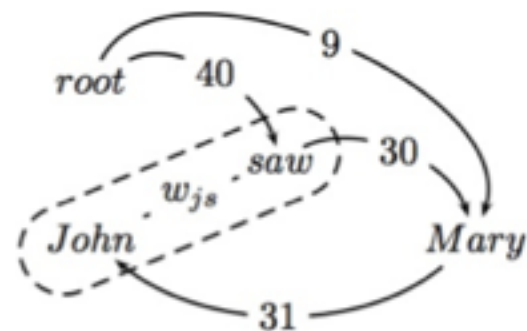
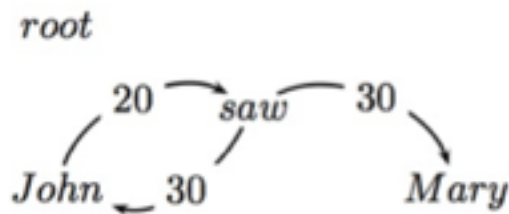
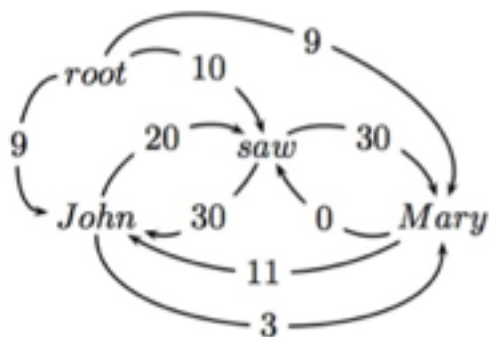
# Find Cycle and Contract

---

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle



# Recalculate Edge Weights



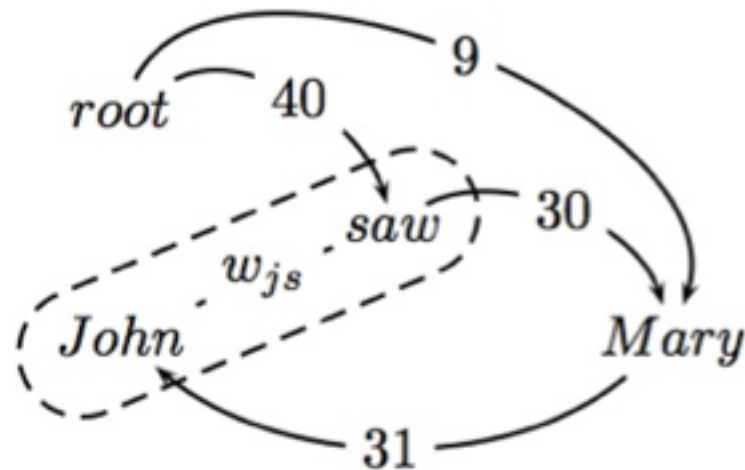
## ► Incoming arc weights

- Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- *root* → *saw* → *John* is 40 (\*\*)
- *root* → *John* → *saw* is 29

# Theorem

---

The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph

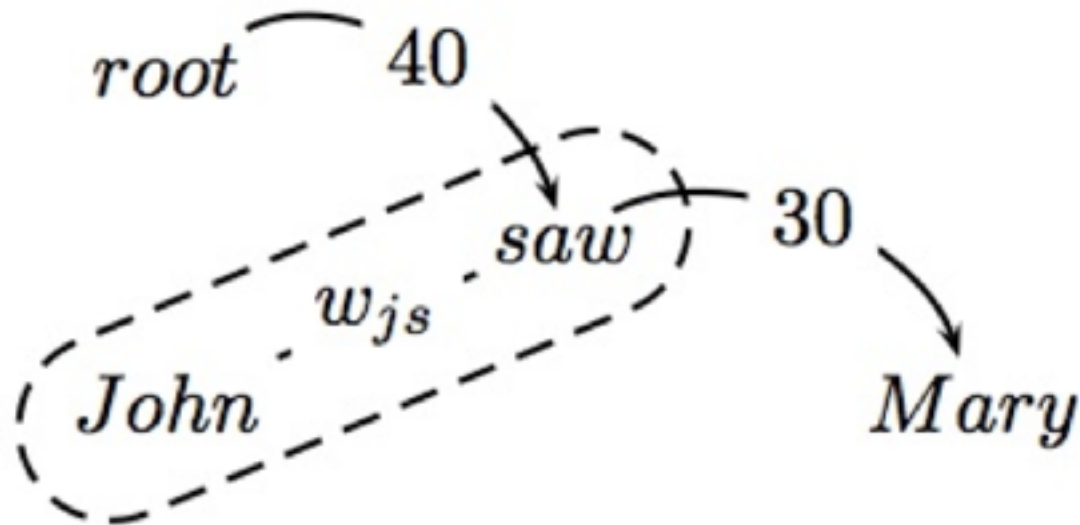


- ▶ Therefore, recursively call algorithm on new graph

# Final MST

---

- ▶ This is a tree and the MST for the contracted graph!!



- ▶ Go back up recursive call and reconstruct final graph

# Chu-Liu-Edmonds PseudoCode

---

## Chu-Liu-Edmonds( $G_x, w$ )

1. Let  $M = \{(i^*, j) : j \in V_x, i^* = \arg \max_{i'} w_{ij}\}$
2. Let  $G_M = (V_x, M)$
3. If  $G_M$  has no cycles, then it is an MST: return  $G_M$
4. Otherwise, find a cycle  $C$  in  $G_M$
5. Let  $\langle G_C, c, ma \rangle = \text{contract}(G, C, w)$
6. Let  $G = \text{Chu-Liu-Edmonds}(G_C, w)$
7. Find vertex  $i \in C$  such that  $(i', c) \in G$  and  $ma(i', c) = i$
8. Find arc  $(i'', i) \in C$
9. Find all arc  $(c, i''') \in G$
10.  $G = G \cup \{(ma(c, i'''), i''')\}_{\forall (c, i''') \in G} \cup C \cup \{(i', i)\} - \{(i'', i)\}$
11. Remove all vertices and arcs in  $G$  containing  $c$
12. return  $G$

► Reminder:  $w_{ij} = \arg \max_k w_{ij}^k$

# Chu-Liu-Edmonds PseudoCode

---

**contract**( $G = (V, A), C, w$ )

1. Let  $G_C$  be the subgraph of  $G$  excluding nodes in  $C$
2. Add a node  $c$  to  $G_C$  representing cycle  $C$
3. For  $i \in V - C : \exists i' \in C (i', i) \in A$   
Add arc  $(c, i)$  to  $G_C$  with  
 $ma(c, i) = \arg \max_{i' \in C} score(i', i)$   
 $i' = ma(c, i)$   
 $score(c, i) = score(i', i)$
4. For  $i \in V - C : \exists i' \in C (i, i') \in A$   
Add edge  $(i, c)$  to  $G_C$  with  
 $ma(i, c) = \arg \max_{i' \in C} [score(i, i') - score(a(i'), i')]$   
 $i' = ma(i, c)$   
 $score(i, c) = [score(i, i') - score(a(i'), i') + score(C)]$   
where  $a(v)$  is the predecessor of  $v$  in  $C$   
and  $score(C) = \sum_{v \in C} score(a(v), v)$
5. return  $\langle G_C, c, ma \rangle$



# Arc Weights

---

$$w_{ij}^k = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- ▶ Arc weights are a linear combination of features of the arc,  $\mathbf{f}$ , and a corresponding weight vector  $\mathbf{w}$
- ▶ Raised to an exponent (simplifies some math ...)
- ▶ What arc features?
- ▶ [McDonald et al. 2005] discuss a number of binary features

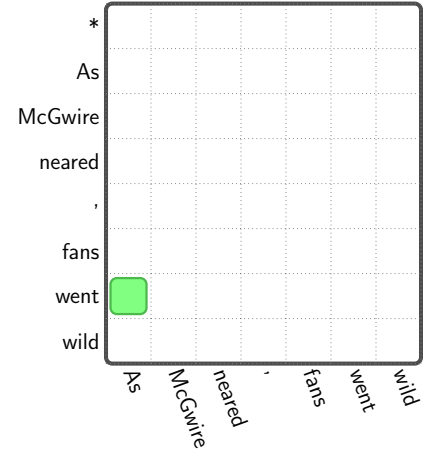
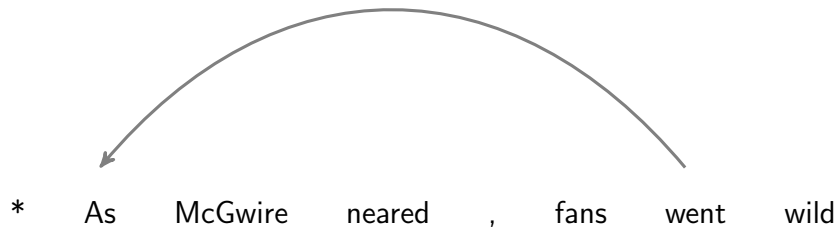
# Arc Feature Ideas for $f(i,j,k)$

---



- Identities of the words  $w_i$  and  $w_j$  and the label  $l_k$
- Part-of-speech tags of the words  $w_i$  and  $w_j$  and the label  $l_k$
- Part-of-speech of words surrounding and between  $w_i$  and  $w_j$
- Number of words between  $w_i$  and  $w_j$ , and their orientation
- Combinations of the above

# First-Order Feature Computation



[went]	[VBD]	[As]	[ADP]	[went]
[VERB]	[As]	[IN]	[went, VBD]	[As, ADP]
[went, As]	[VBD, ADP]	[went, VERB]	[As, IN]	[went, As]
[VERB, IN]	[VBD, As, ADP]	[went, As, ADP]	[went, VBD, ADP]	[went, VBD, As]
[ADJ, *, ADP]	[VBD, *, ADP]	[VBD, ADJ, ADP]	[VBD, ADJ, *]	[NNS, *, ADP]
[NNS, VBD, ADP]	[NNS, VBD, *]	[ADJ, ADP, NNP]	[VBD, ADP, NNP]	[VBD, ADJ, NNP]
[NNS, ADP, NNP]	[NNS, VBD, NNP]	[went, left, 5]	[VBD, left, 5]	[As, left, 5]
[ADP, left, 5]	[VERB, As, IN]	[went, As, IN]	[went, VERB, IN]	[went, VERB, As]
[JJ, *, IN]	[VERB, *, IN]	[VERB, JJ, IN]	[VERB, JJ, *]	[NOUN, *, IN]
[NOUN, VERB, IN]	[NOUN, VERB, *]	[JJ, IN, NOUN]	[VERB, IN, NOUN]	[VERB, JJ, NOUN]
[NOUN, IN, NOUN]	[NOUN, VERB, NOUN]	[went, left, 5]	[VERB, left, 5]	[As, left, 5]
[IN, left, 5]	[went, VBD, As, ADP]	[VBD, ADJ, *, ADP]	[NNS, VBD, *, ADP]	[VBD, ADJ, ADP, NNP]
[NNS, VBD, ADP, NNP]	[went, VBD, left, 5]	[As, ADP, left, 5]	[went, As, left, 5]	[VBD, ADP, left, 5]
[went, VERB, As, IN]	[VERB, JJ, *, IN]	[NOUN, VERB, *, IN]	[VERB, JJ, IN, NOUN]	[NOUN, VERB, IN, NOUN]
[went, VERB, left, 5]	[As, IN, left, 5]	[went, As, left, 5]	[VERB, IN, left, 5]	[VBD, As, ADP, left, 5]
[went, As, ADP, left, 5]	[went, VBD, ADP, left, 5]	[went, VBD, As, left, 5]	[ADJ, *, ADP, left, 5]	[VBD, *, ADP, left, 5]
[VBD, ADJ, ADP, left, 5]	[VBD, ADJ, *, left, 5]	[NNS, *, ADP, left, 5]	[NNS, VBD, ADP, left, 5]	[NNS, VBD, *, left, 5]
[ADJ, ADP, NNP, left, 5]	[VBD, ADP, NNP, left, 5]	[VBD, ADJ, NNP, left, 5]	[NNS, ADP, NNP, left, 5]	[NNS, VBD, NNP, left, 5]
[VERB, As, IN, left, 5]	[went, As, IN, left, 5]	[went, VERB, IN, left, 5]	[went, VERB, As, left, 5]	[JJ, *, IN, left, 5]
[VERB, *, IN, left, 5]	[VERB, JJ, IN, left, 5]	[VERB, JJ, *, left, 5]	[NOUN, *, IN, left, 5]	[NOUN, VERB, IN, left, 5]

# (Structured) Perceptron

---

Training data:  $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{|\mathcal{T}|}$

1.  $\mathbf{w}^{(0)} = \mathbf{0}$ ;  $i = 0$
2. for  $n : 1..N$
3.     for  $t : 1..T$
4.         Let  $G' = \arg \max_{G'} \mathbf{w}^{(i)} \cdot \mathbf{f}(G')$
5.         if  $G' \neq G_t$
6.              $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(G_t) - \mathbf{f}(G')$
7.              $i = i + 1$
8. return  $\mathbf{w}^i$

# Transition Based Dependency Parsing

---

- Process sentence left to right
  - Different transition strategies available
  - Delay decisions by pushing on stack
- Arc-Standard Transition Strategy [Nivre '03]

Initial configuration:  $([], [0, \dots, n], [])$

Terminal configuration:  $([0], [], A)$

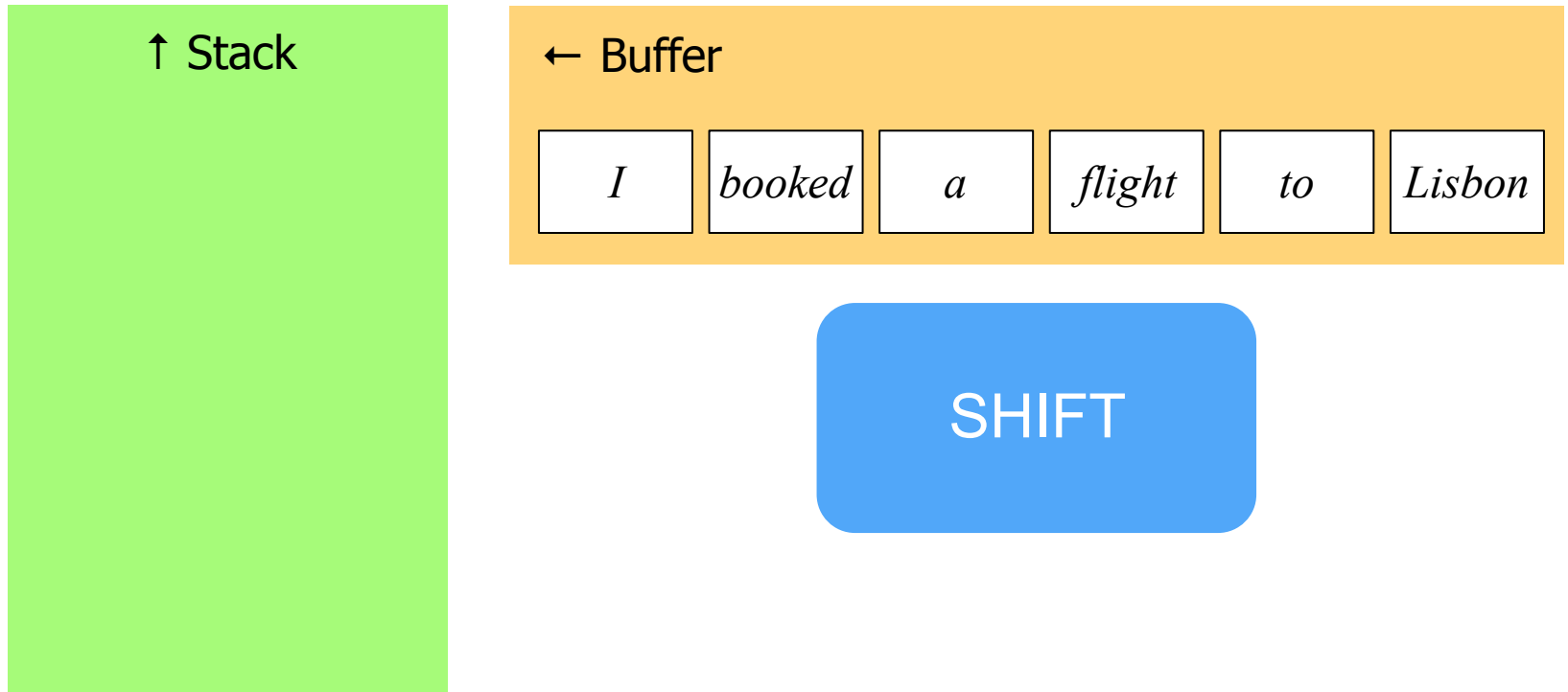
shift:  $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$

left-arc (label):  $([\sigma|i|j], B, A) \Rightarrow ([\sigma|j], B, A \cup \{j, l, i\})$

right-arc (label):  $([\sigma|i|j], B, A) \Rightarrow ([\sigma|i], B, A \cup \{i, l, j\})$

# Arc-Standard Example

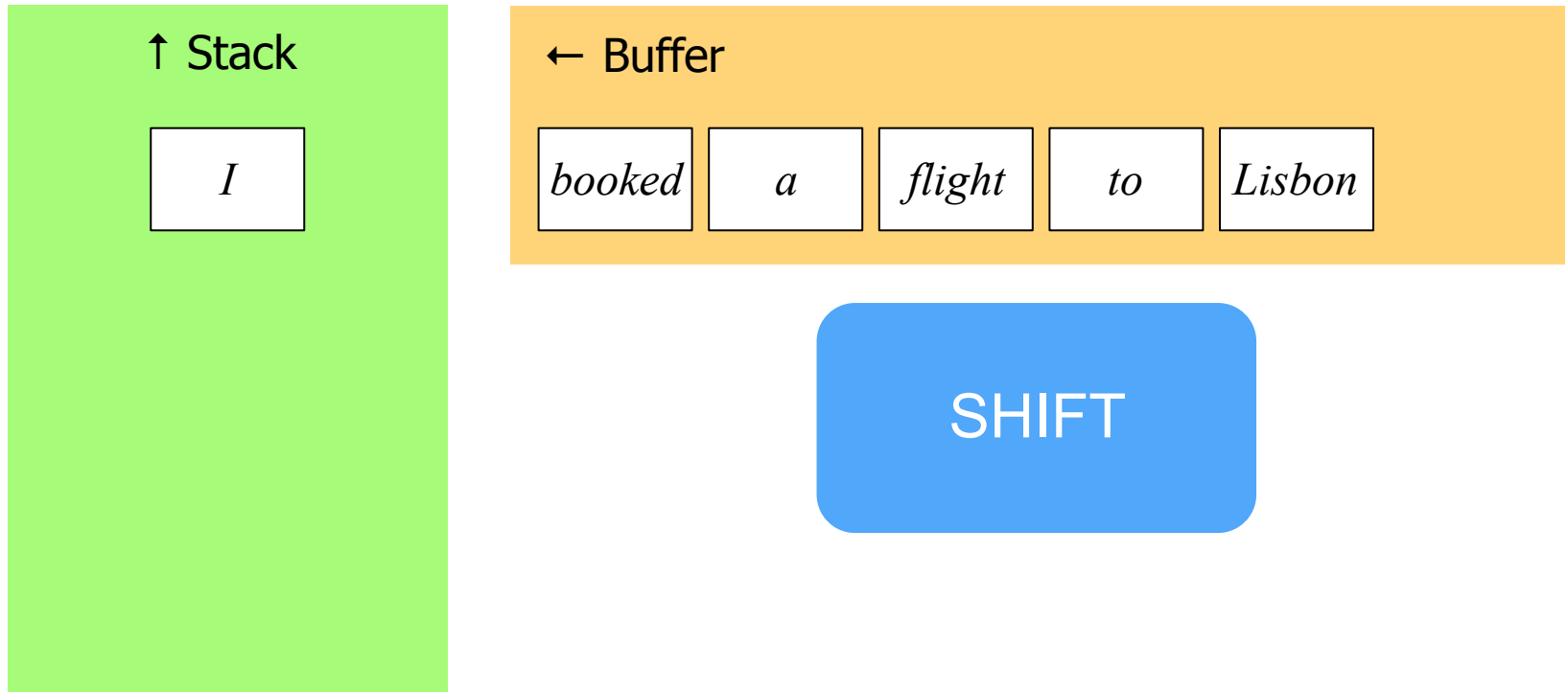
---



*I booked a flight to Lisbon*

# Arc-Standard Example

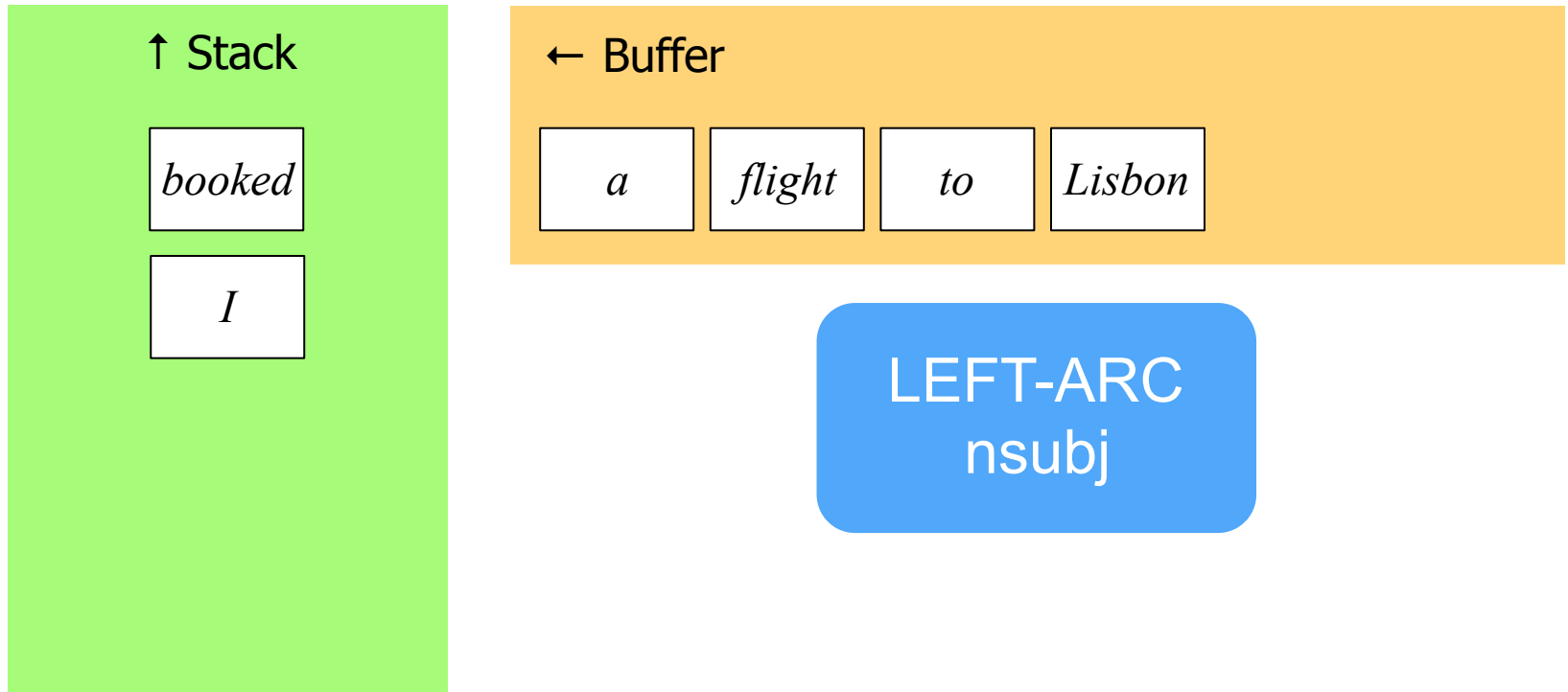
---



*I booked a flight to Lisbon*

# Arc-Standard Example

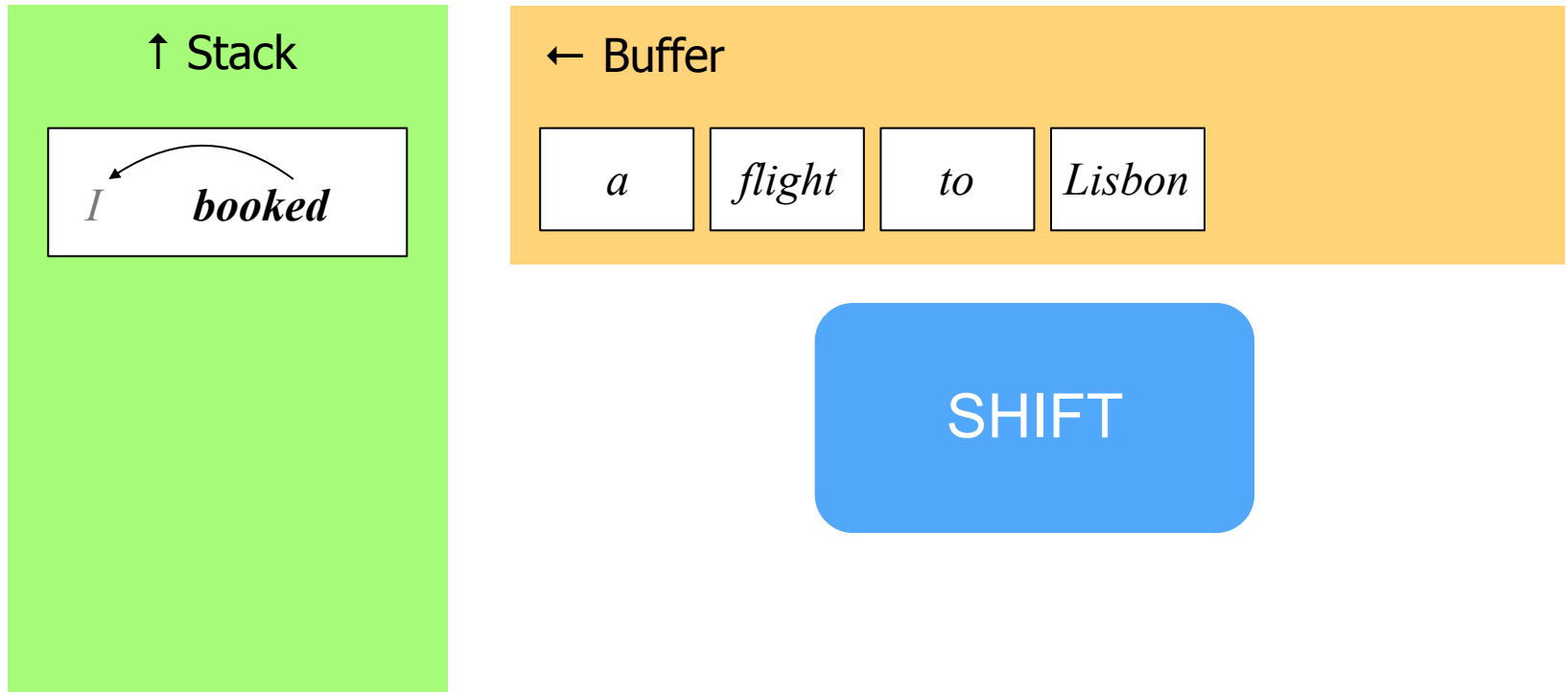
---



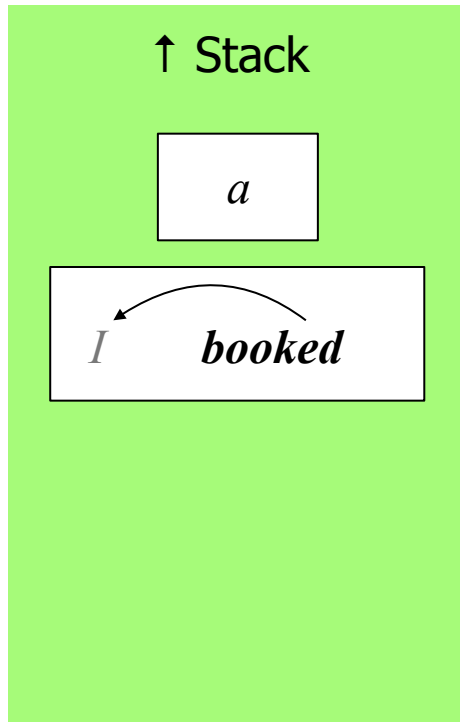
*I booked a flight to Lisbon*



# Arc-Standard Example



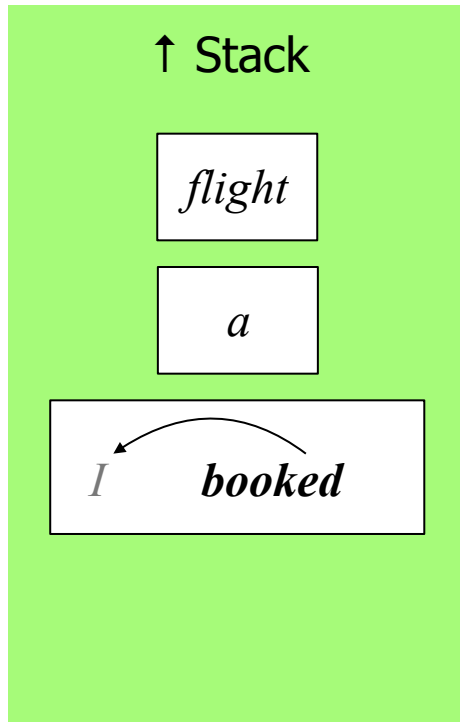
# Arc-Standard Example



SHIFT



# Arc-Standard Example

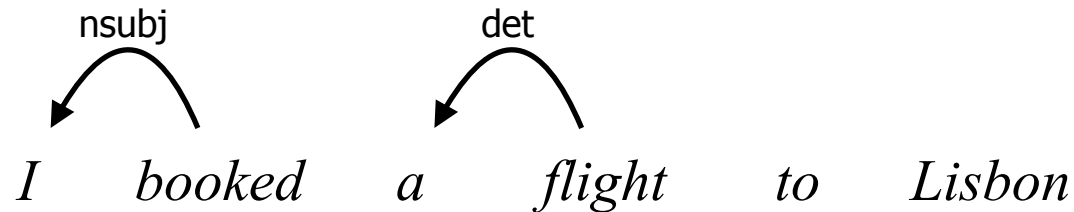
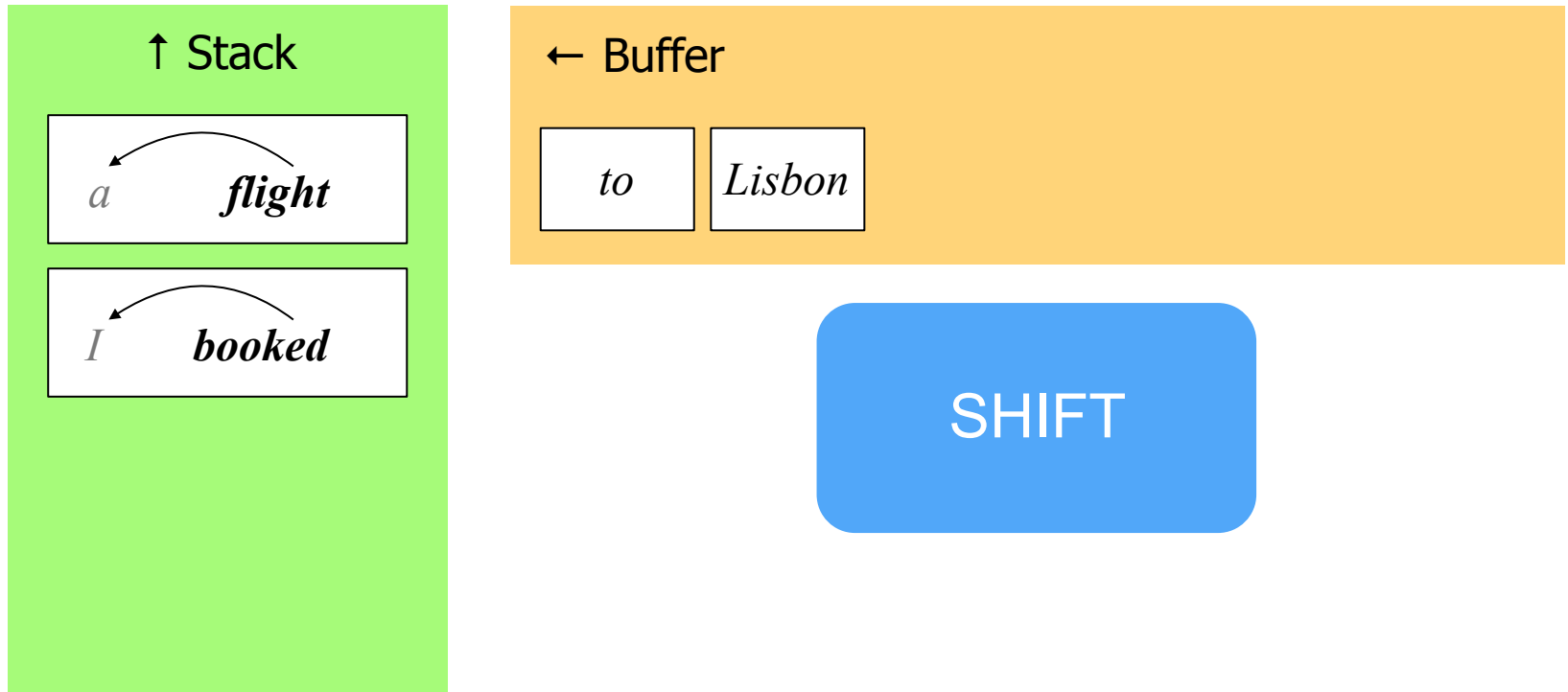


LEFT-ARC  
det

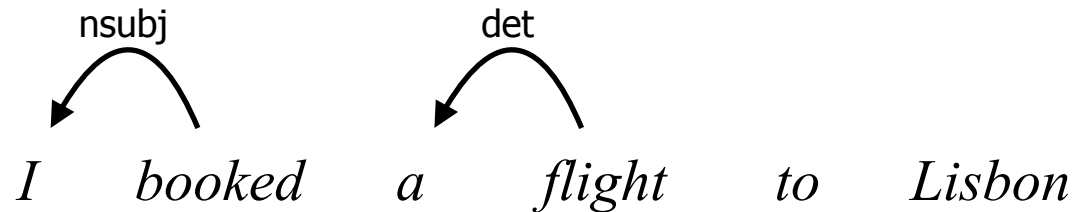
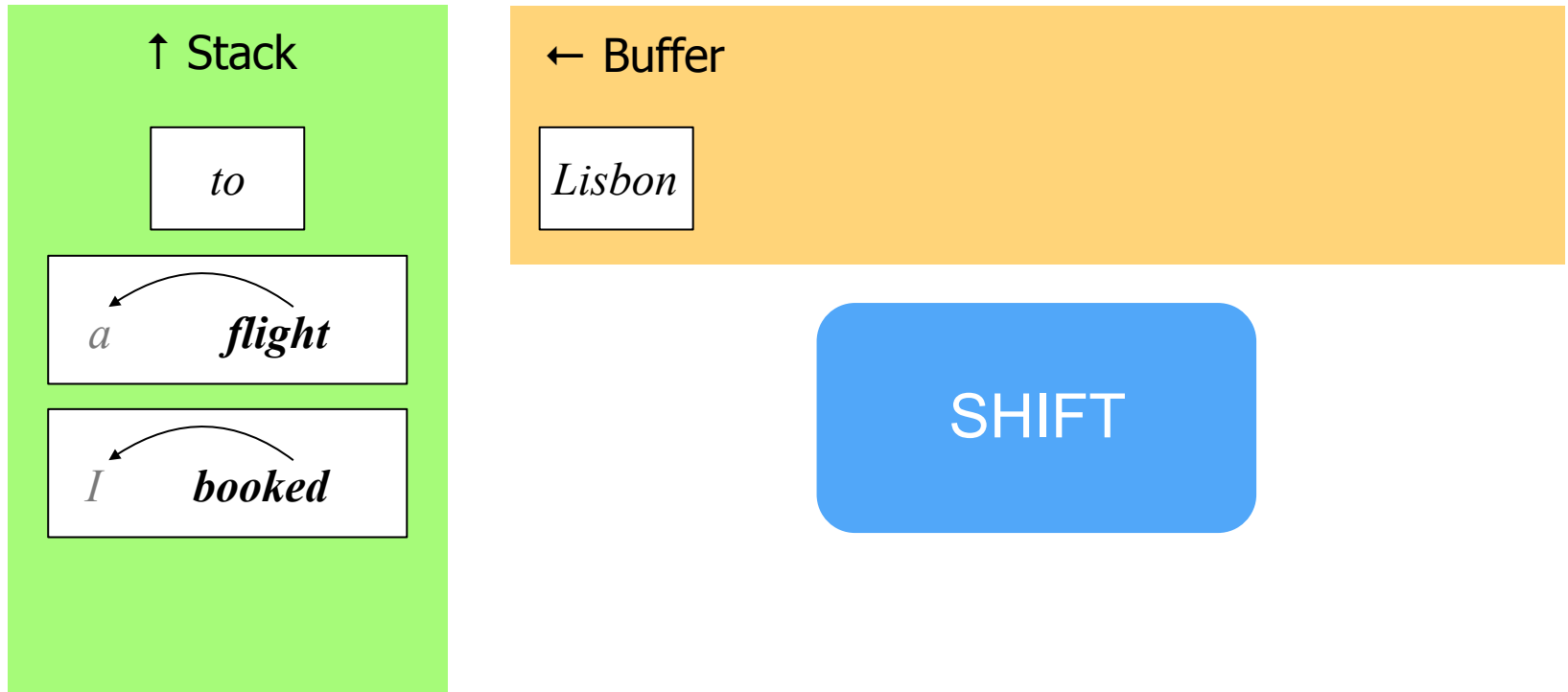
*I* *booked* *a* *flight* *to* *Lisbon*

nsubj

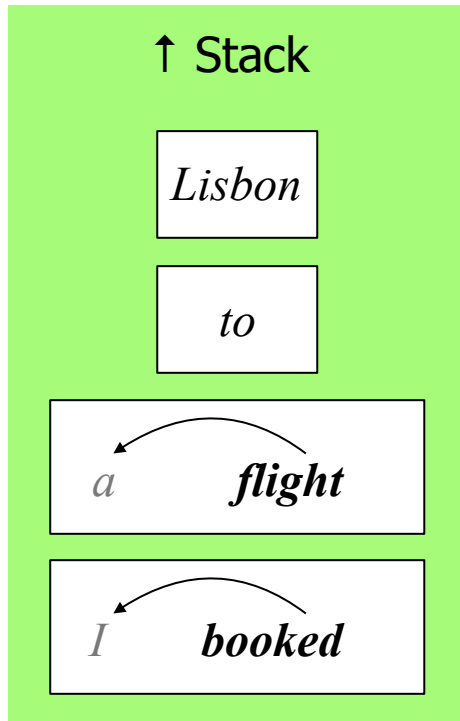
# Arc-Standard Example



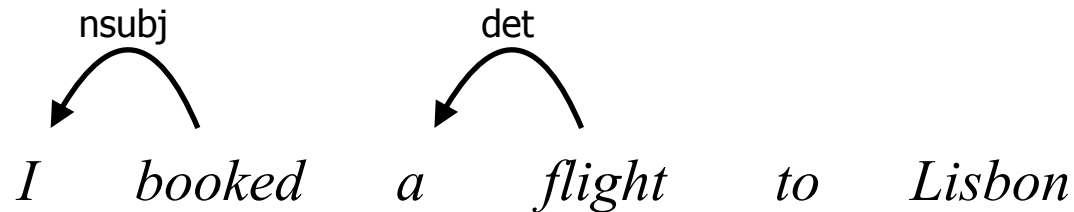
# Arc-Standard Example



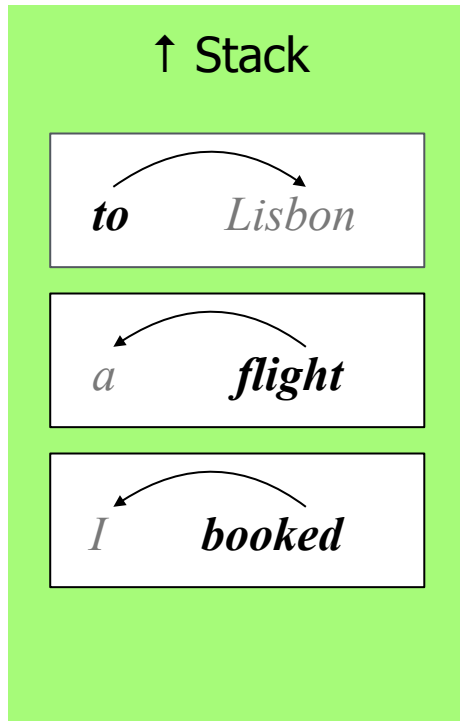
# Arc-Standard Example



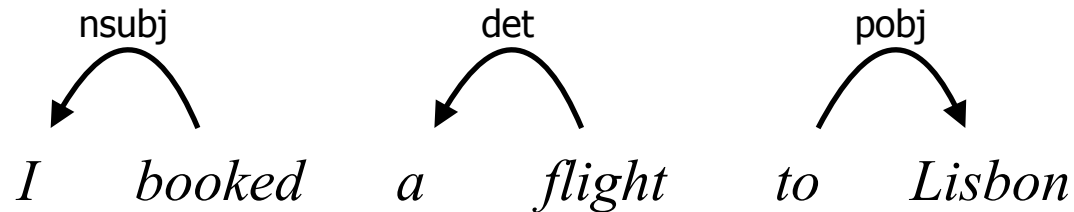
RIGHT-ARC  
pobj



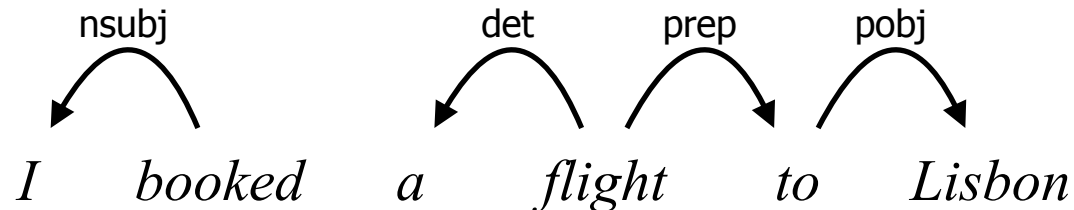
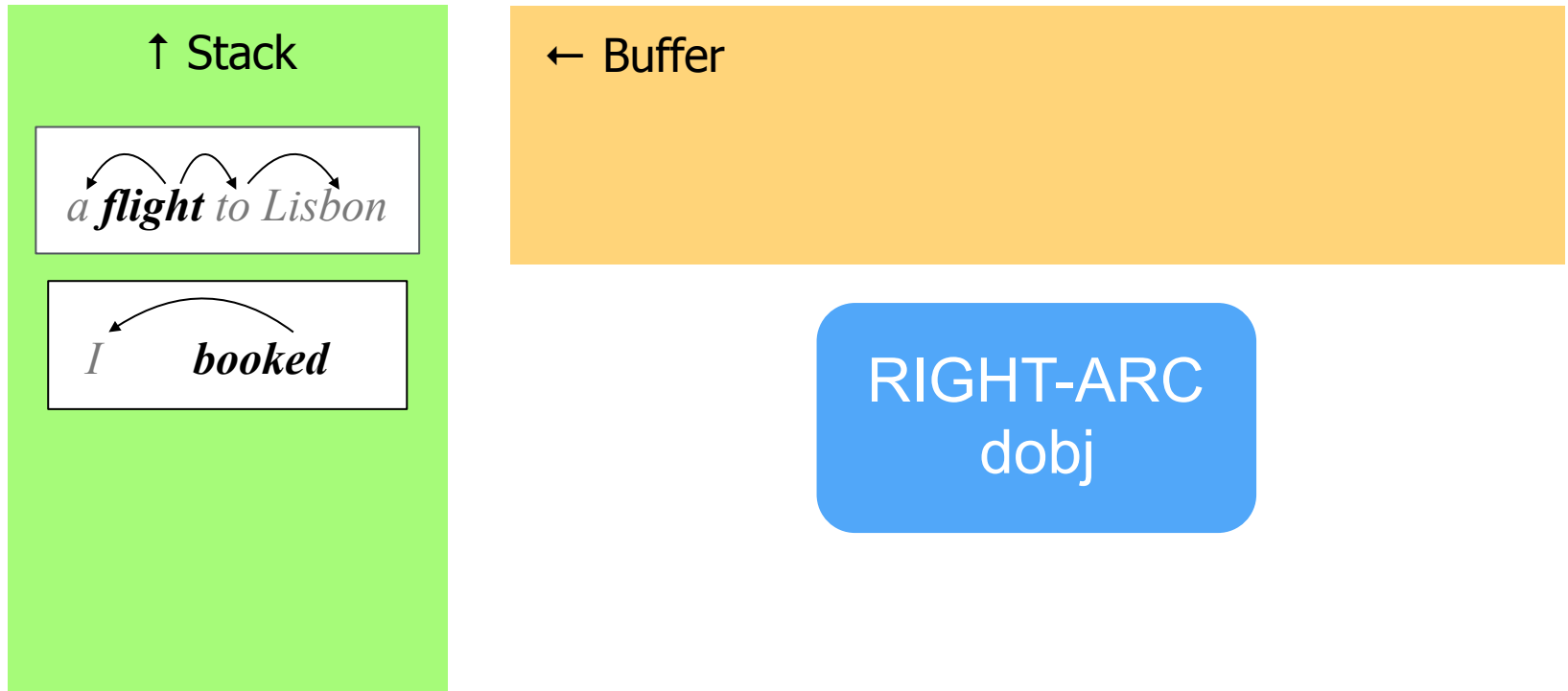
# Arc-Standard Example



RIGHT-ARC  
prep



# Arc-Standard Example





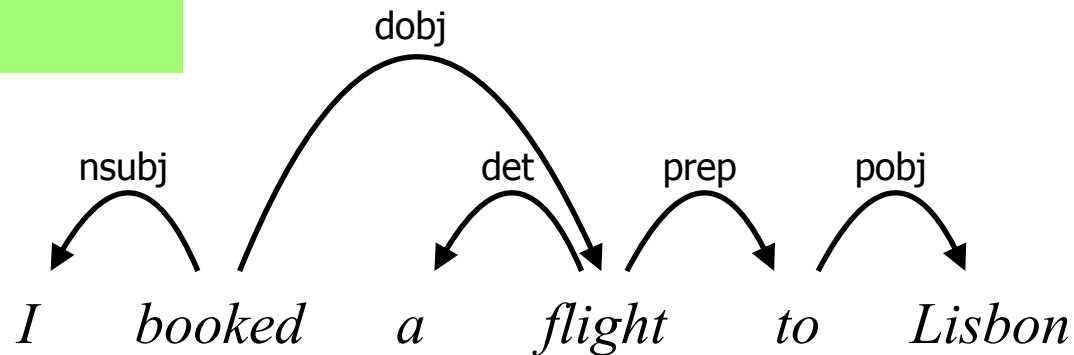
# Arc-Standard Example

---

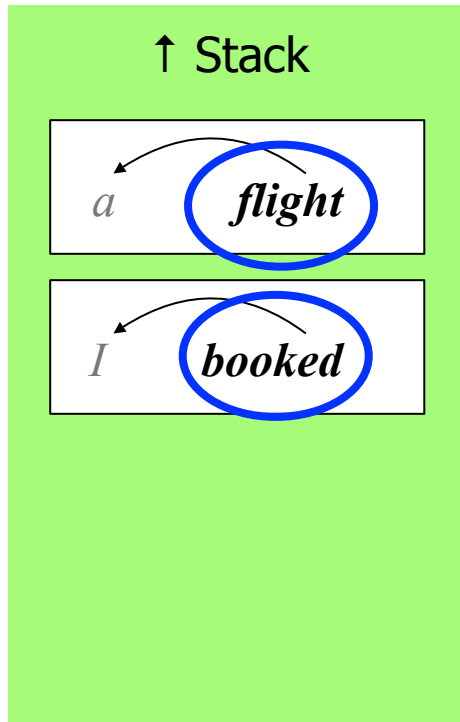
↑ Stack



← Buffer



# Features



SHIFT

RIGHT-ARC?

LEFT-ARC?

Stack top word = "flight"  
Stack top POS tag = "NOUN"  
Buffer front word = "to"  
Child of stack top word = "a"

....

# Features ZPar Parser

```
# From Single Words
pair { stack.tag stack.word }
stack { word tag }
pair { input.tag input.word }
input { word tag }
pair { input(1).tag input(1).word }
input(1) { word tag }
pair { input(2).tag input(2).word }
input(2) { word tag }

# From word pairs
quad { stack.tag stack.word input.tag input.word }
triple { stack.tag stack.word input.word }
triple { stack.word input.tag input.word }
triple { stack.tag stack.word input.tag }
triple { stack.tag input.tag input.word }
pair { stack.word input.word }
pair { stack.tag input.tag }
pair { input.tag input(1).tag }

# From word triples
triple { input.tag input(1).tag input(2).tag }
triple { stack.tag input.tag input(1).tag }
triple { stack.head(1).tag stack.tag input.tag }
triple { stack.tag stack.child(-1).tag input.tag }
triple { stack.tag stack.child(1).tag input.tag }
triple { stack.tag input.tag input.child(-1).tag }

# Distance
pair { stack.distance stack.word }
pair { stack.distance stack.tag }
pair { stack.distance input.word }
pair { stack.distance input.tag }
triple { stack.distance stack.word input.word }
triple { stack.distance stack.tag input.tag }
```

```
# valency
pair { stack.word stack.valence(-1) }
pair { stack.word stack.valence(1) }
pair { stack.tag stack.valence(-1) }
pair { stack.tag stack.valence(1) }
pair { input.word input.valence(-1) }
pair { input.tag input.valence(-1) }

# unigrams
stack.head(1) {word tag}
stack.label
stack.child(-1) {word tag label}
stack.child(1) {word tag label}
input.child(-1) {word tag label}

# third order
stack.head(1).head(1) {word tag}
stack.head(1).label
stack.child(-1).sibling(1) {word tag label}
stack.child(1).sibling(-1) {word tag label}
input.child(-1).sibling(1) {word tag label}
triple { stack.tag stack.child(-1).tag stack.child(-1).sibling(1).tag }
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1).tag }
triple { stack.tag stack.head(1).tag stack.head(1).head(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(1).tag }

# label set
pair { stack.tag stack.child(-1).label }
triple { stack.tag stack.child(-1).label stack.child(-1).sibling(1).label }
quad { stack.tag stack.child(-1).label stack.child(-1).sibling(1).label }
pair { stack.tag stack.child(1).label }
triple { stack.tag stack.child(1).label stack.child(1).sibling(-1).label }
quad { stack.tag stack.child(1).label stack.child(1).sibling(-1).label }
pair { input.tag input.child(-1).label }
triple { input.tag input.child(-1).label input.child(-1).sibling(1).label }
quad { input.tag input.child(-1).label input.child(-1).sibling(1).label }
```

# SVM / Structured Perceptron Hyperparameters

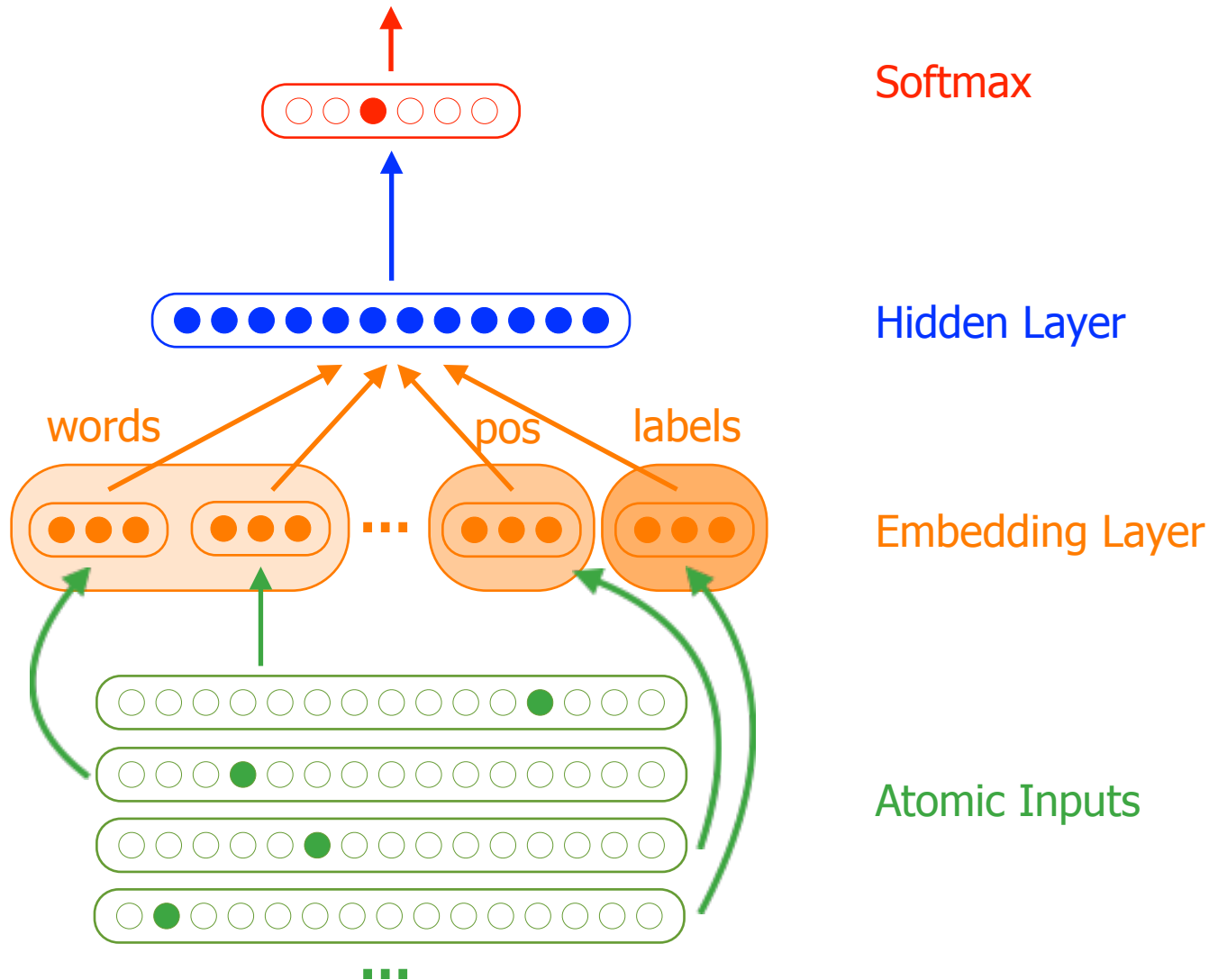
---

- Regularization
- Loss function
- **Hand-crafted features**



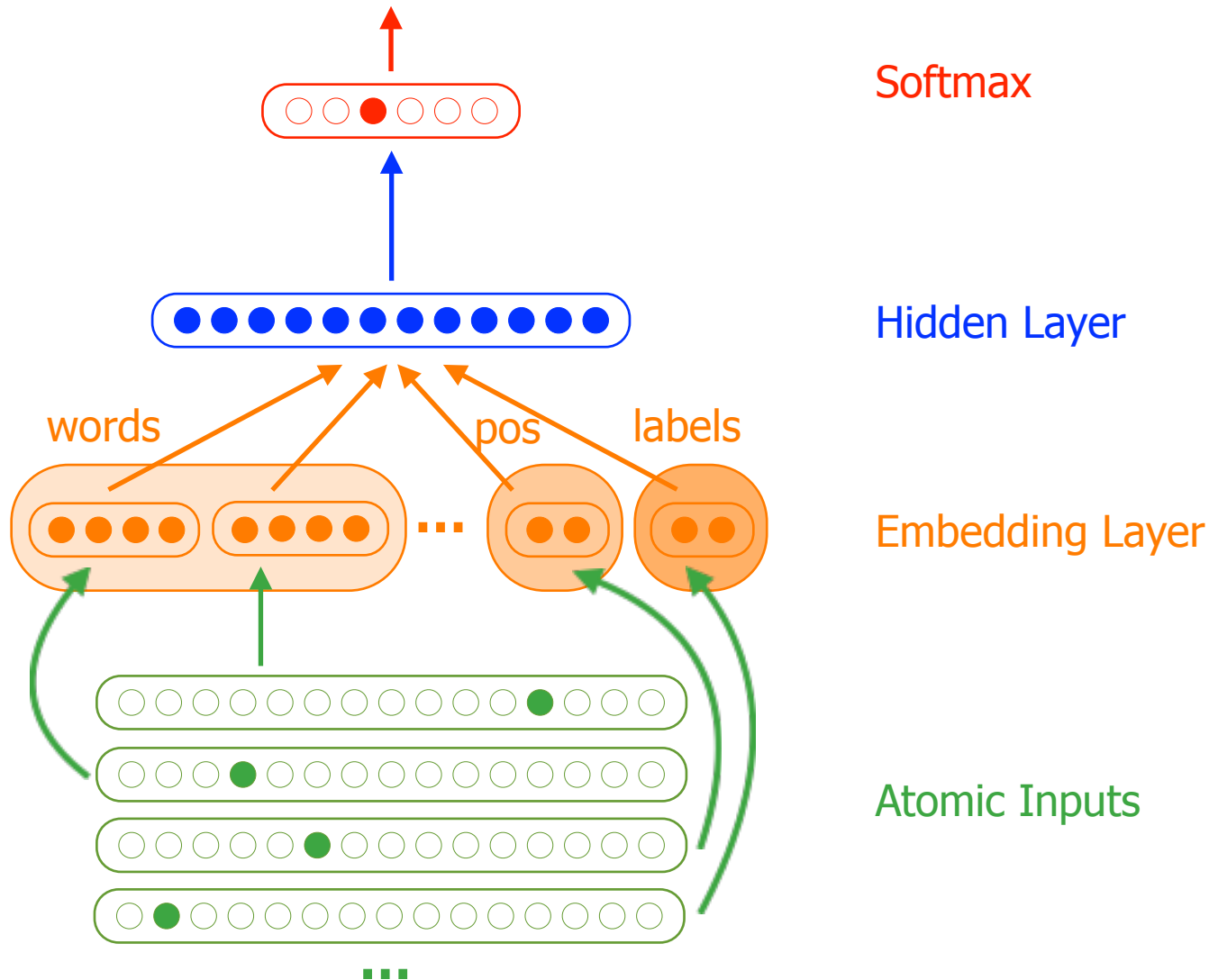
# Neural Network Transition Based Parser

[Chen & Manning '14] and [Weiss et al. '15]



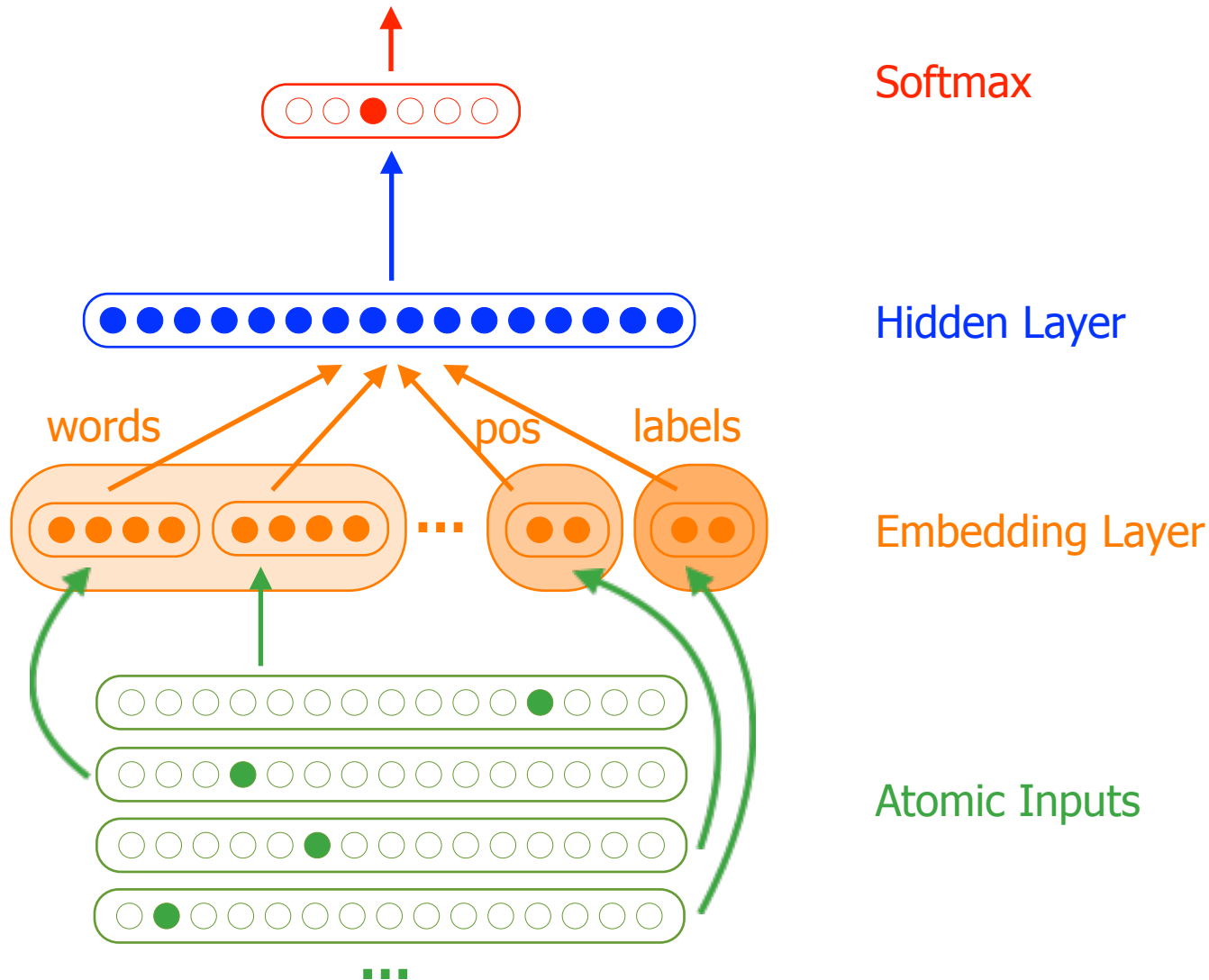
# Neural Network Transition Based Parser

[Weiss et al. '15]



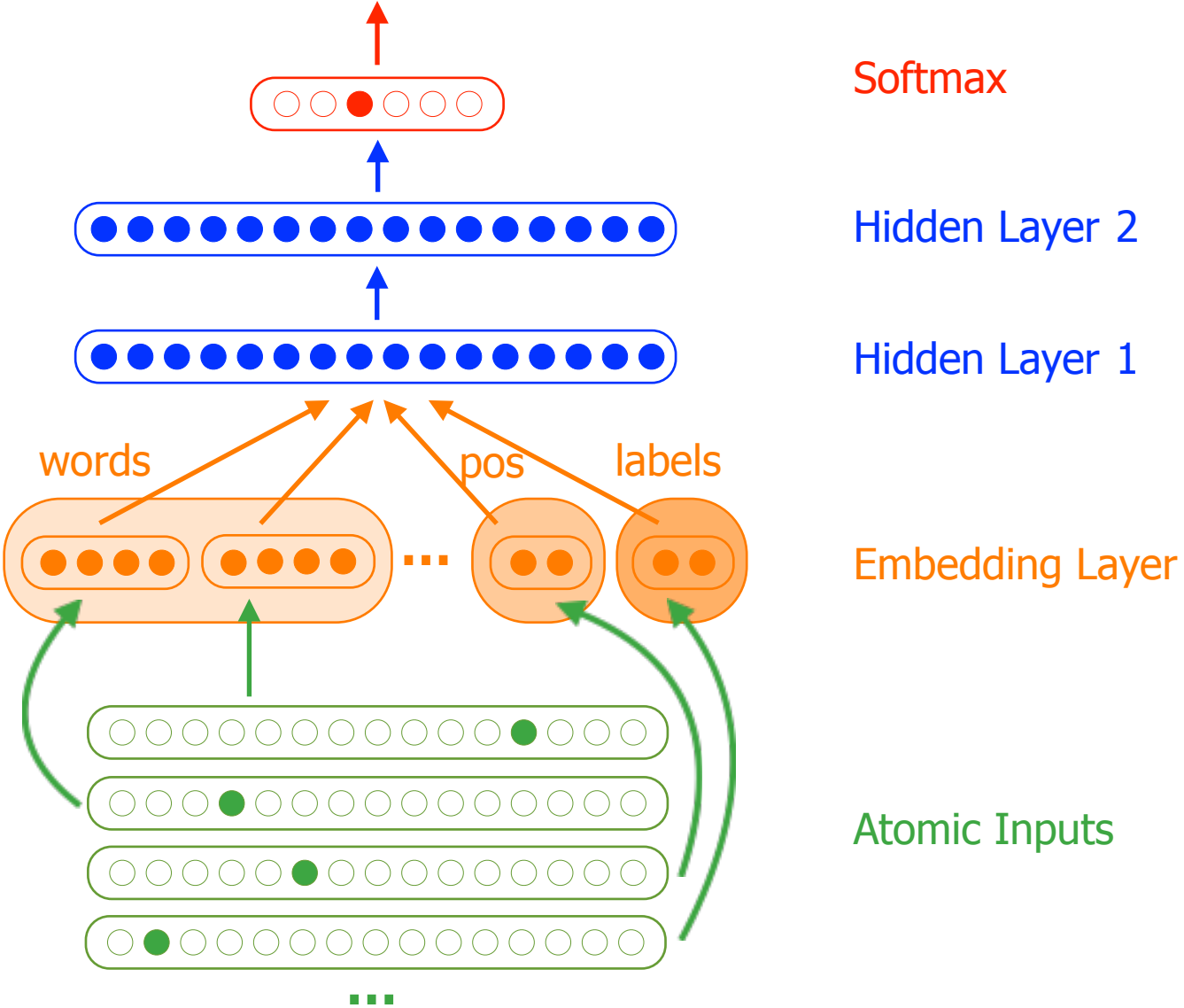
# Neural Network Transition Based Parser

[Weiss et al. '15]



# Neural Network Transition Based Parser

[Weiss et al. '15]



Softmax

Hidden Layer 2

Hidden Layer 1

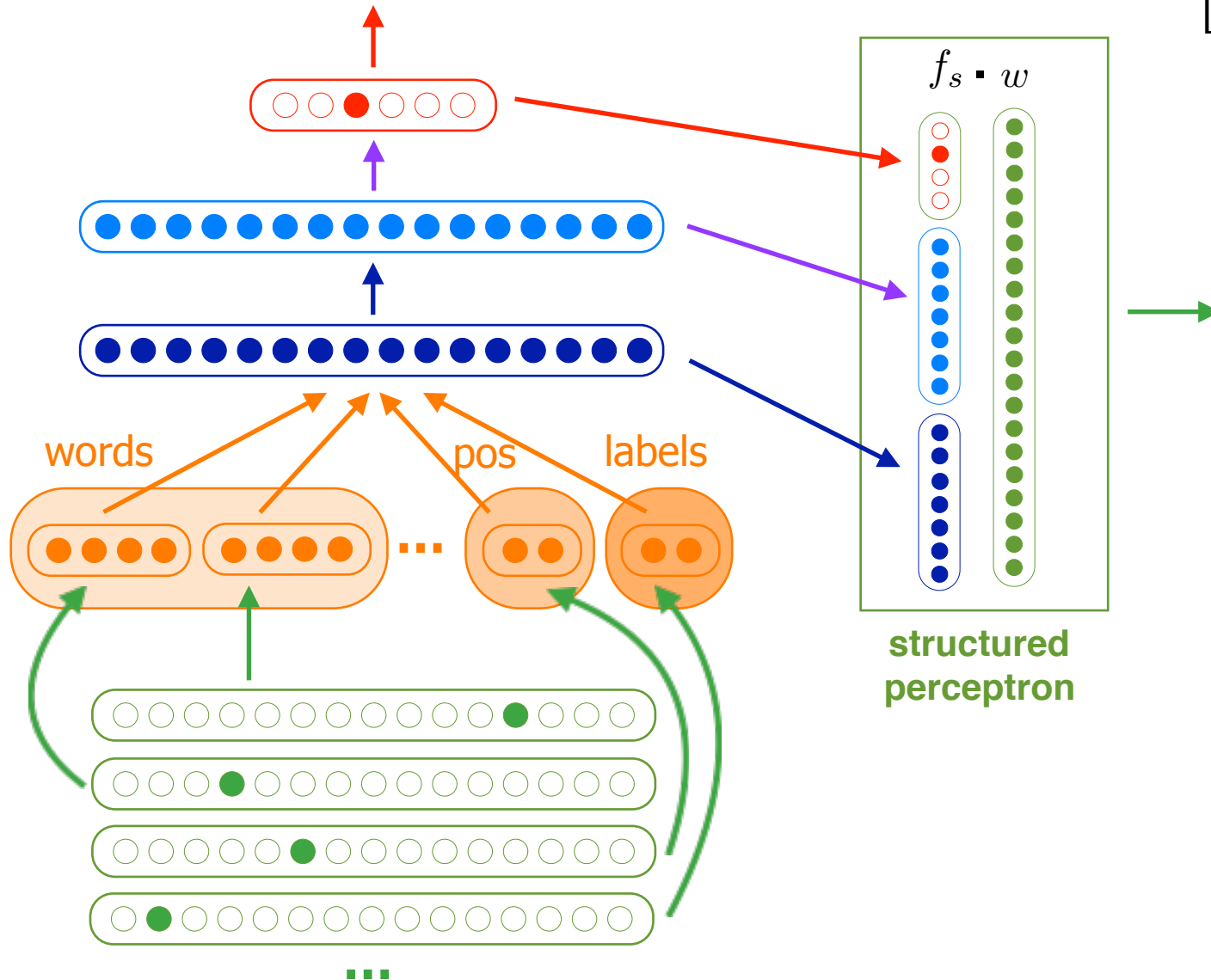
Embedding Layer

Atomic Inputs



# Neural Network Transition Based Parser

[Weiss et al. '15]



# English Results (WSJ 23)

---

<b>Method</b>	<b>UAS</b>	<b>LAS</b>	<b>Beam</b>
3rd-order Graph-based (ZM2014)	93.22	91.02	-
Transition-based Linear (ZN2011)	93.00	90.95	32
NN Baseline (Chen & Manning, 2014)	91.80	89.60	1
NN Better SGD (Weiss et al., 2015)	92.58	90.54	1
NN Deeper Network (Weiss et al., 2015)	93.19	91.18	1

# NN Hyperparameters

---

- Regularization
- Loss function



# NN Hyperparameters

---

- Regularization
- Loss function
- Dimensions
- Activation function
- Initialization
- Adagrad
- Dropout



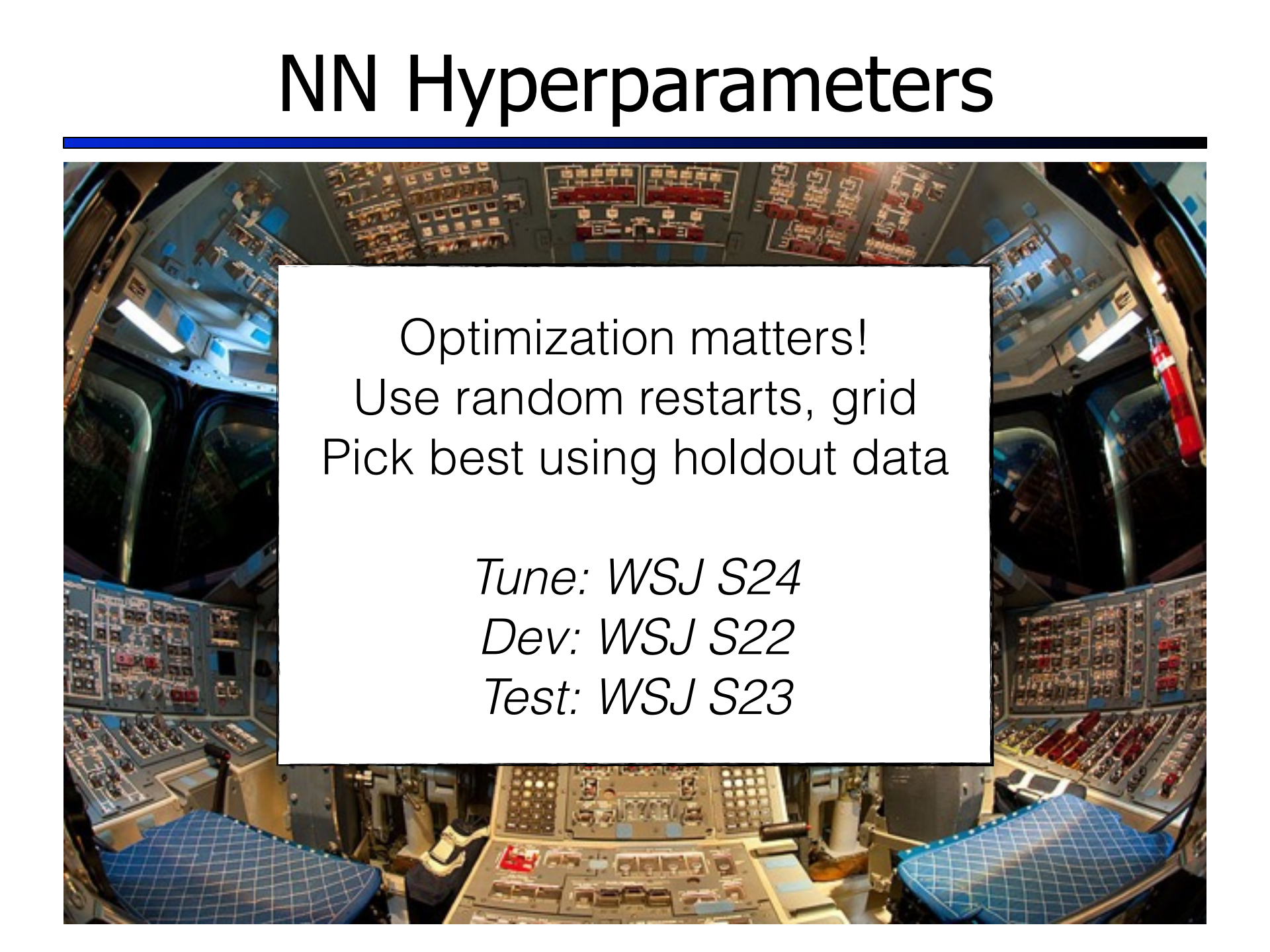
# NN Hyperparameters

- Regularization
- Loss function
- Dimensions
- Activation function
- Initialization
- Adagrad
- Dropout
- Mini-batch size
- Initial learning rate
- Learning rate schedule
- Momentum



- Stopping time
- Parameter averaging

# NN Hyperparameters

A photograph of a spacecraft cockpit, likely from the Space Shuttle era, showing various control panels, instruments, and blue seats. A white text box is overlaid in the center of the image.

Optimization matters!  
Use random restarts, grid  
Pick best using holdout data

*Tune: WSJ S24*

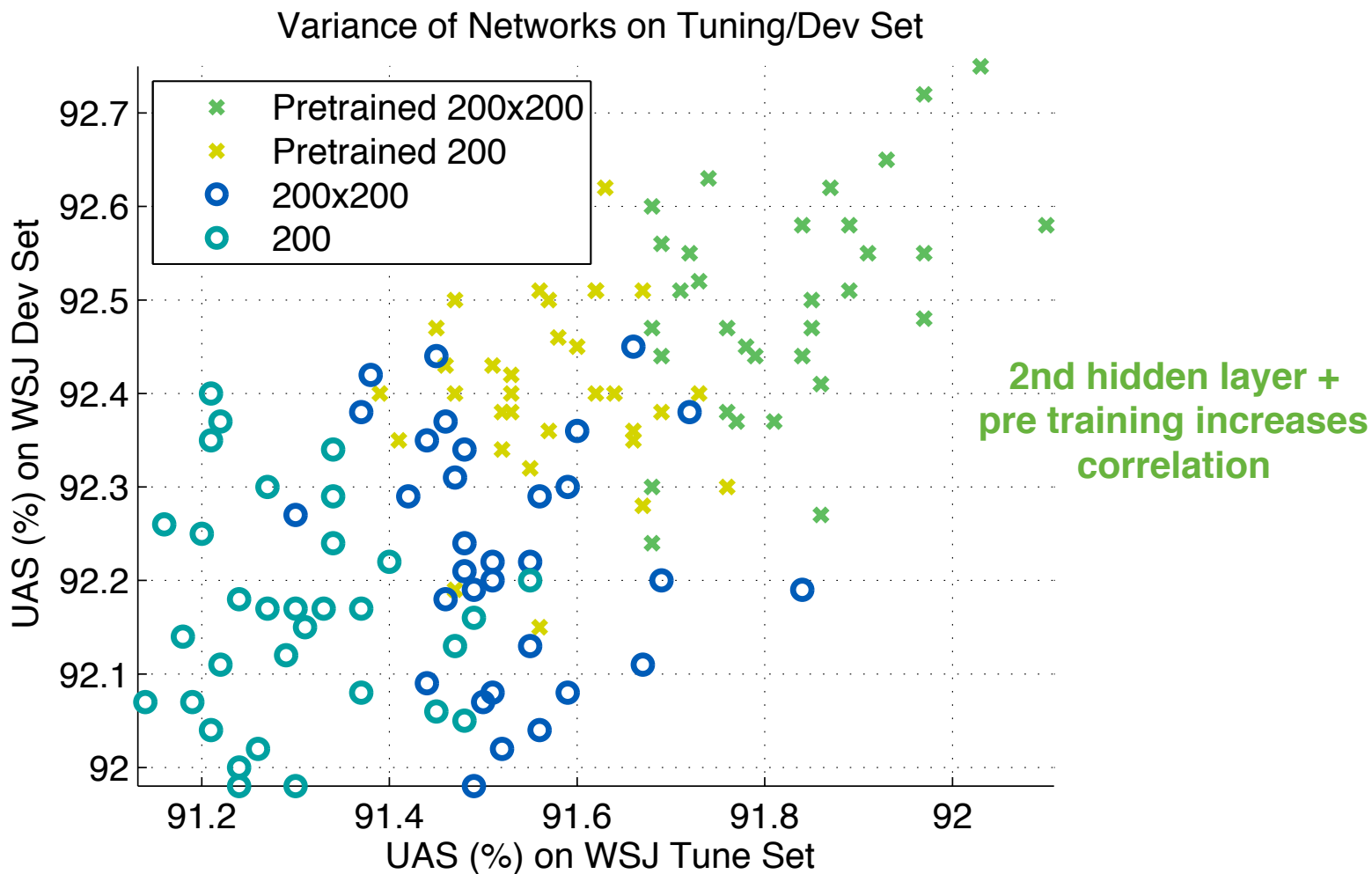
*Dev: WSJ S22*

*Test: WSJ S23*



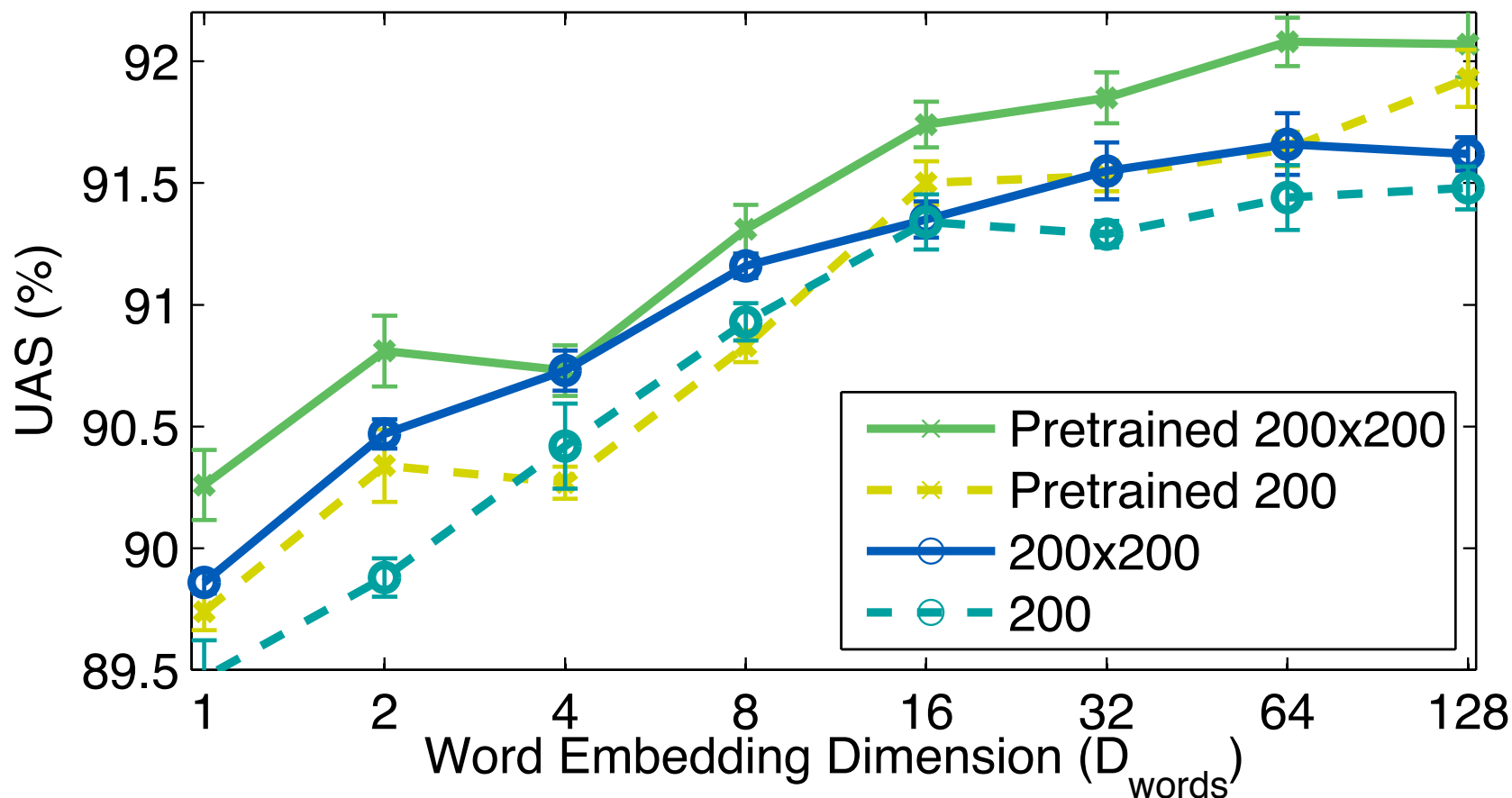
# Random Restarts: How much Variance?

---



# Effect of Embedding Dimensions

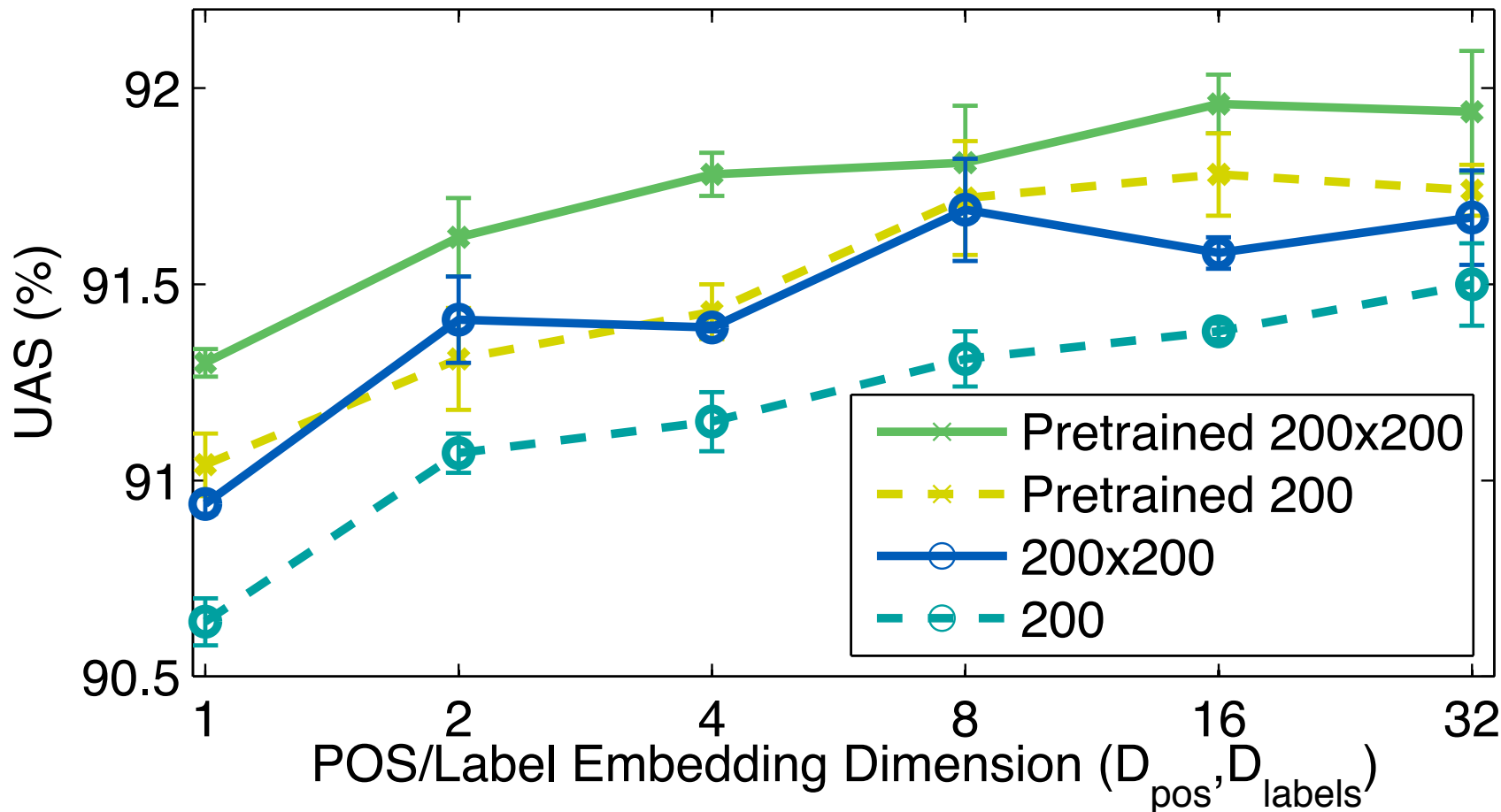
Word Tuning on WSJ (Tune Set,  $D_{\text{pos}}, D_{\text{labels}} = 32$ )





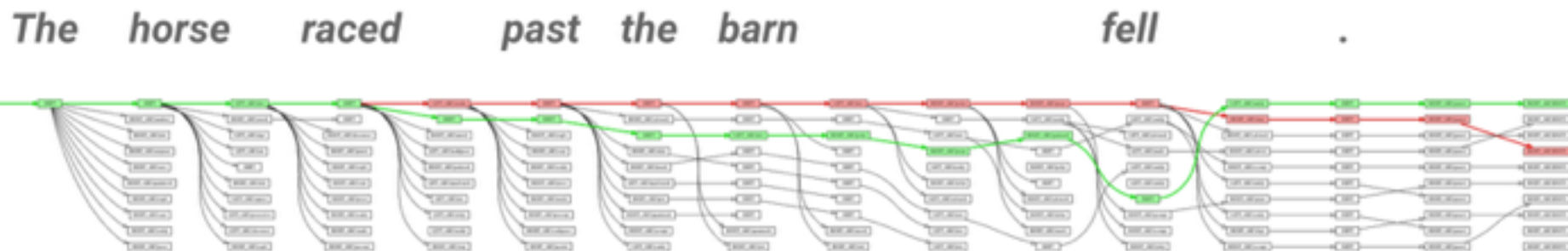
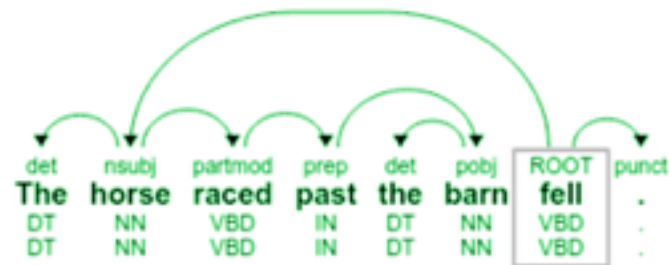
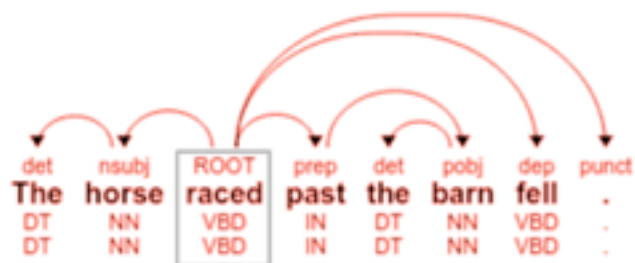
# Effect of Embedding Dimensions

POS/Label Tuning on WSJ (Tune Set,  $D_{\text{words}}=64$ )



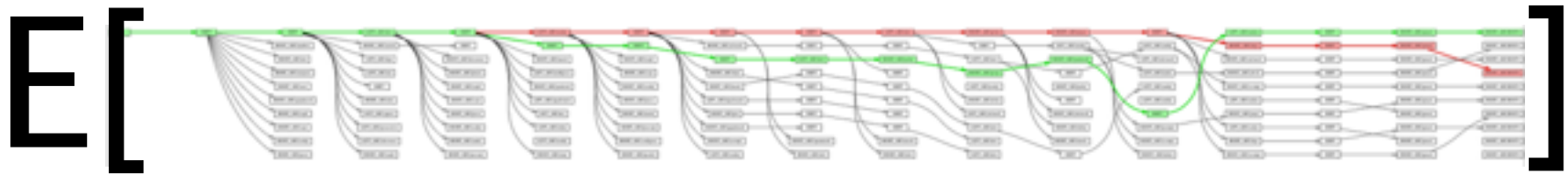
# The Importance of Search

[Weiss et al. '15, Andor et al. '16]



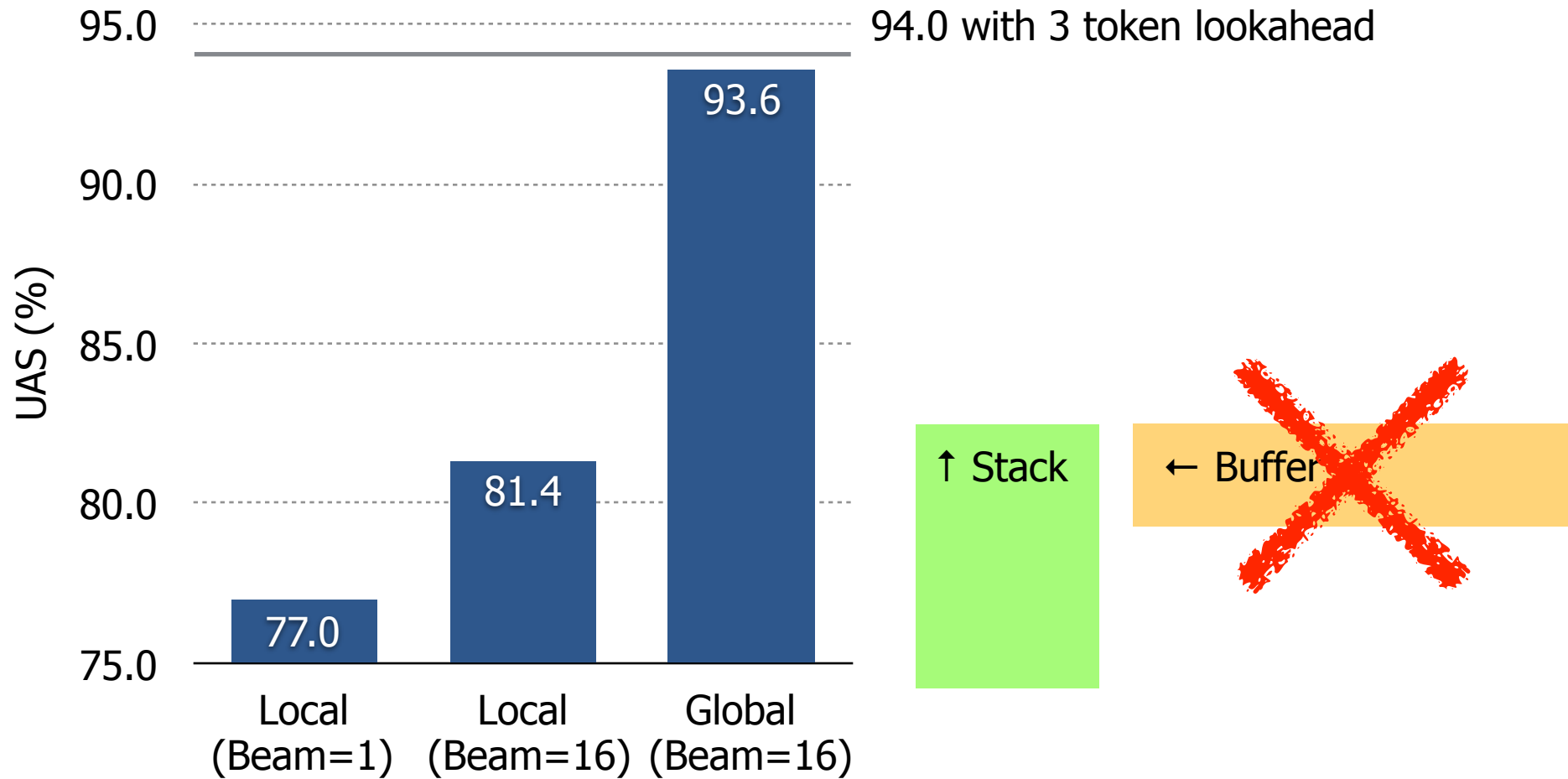
# Globally Normalized Models

[Andor et al. '16]



- CRF Objective
- Full Backpropagation Training
- Novel Proof (Label Bias):
  - Globally Normalized Models are strictly more expressing than Locally Normalized Models

# No Lookahead Parsing



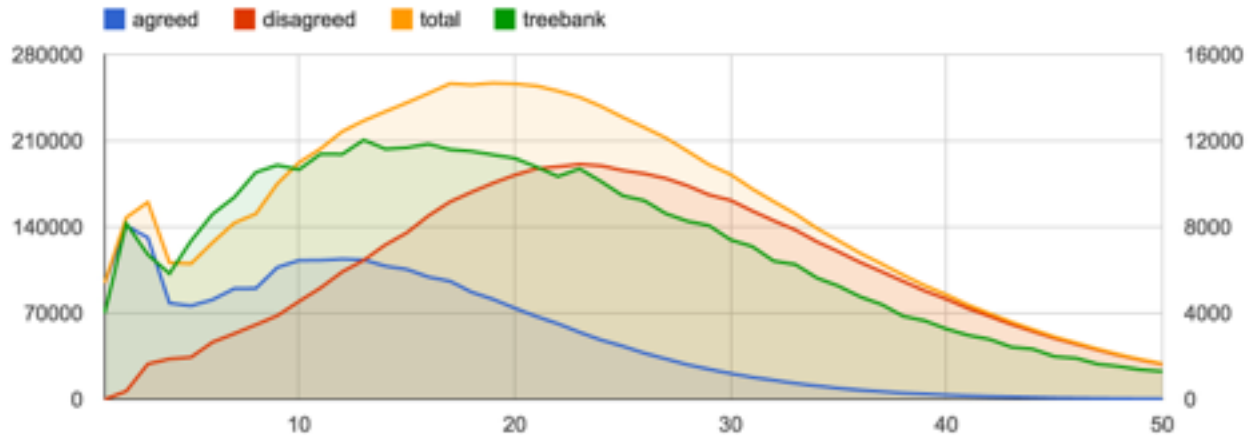
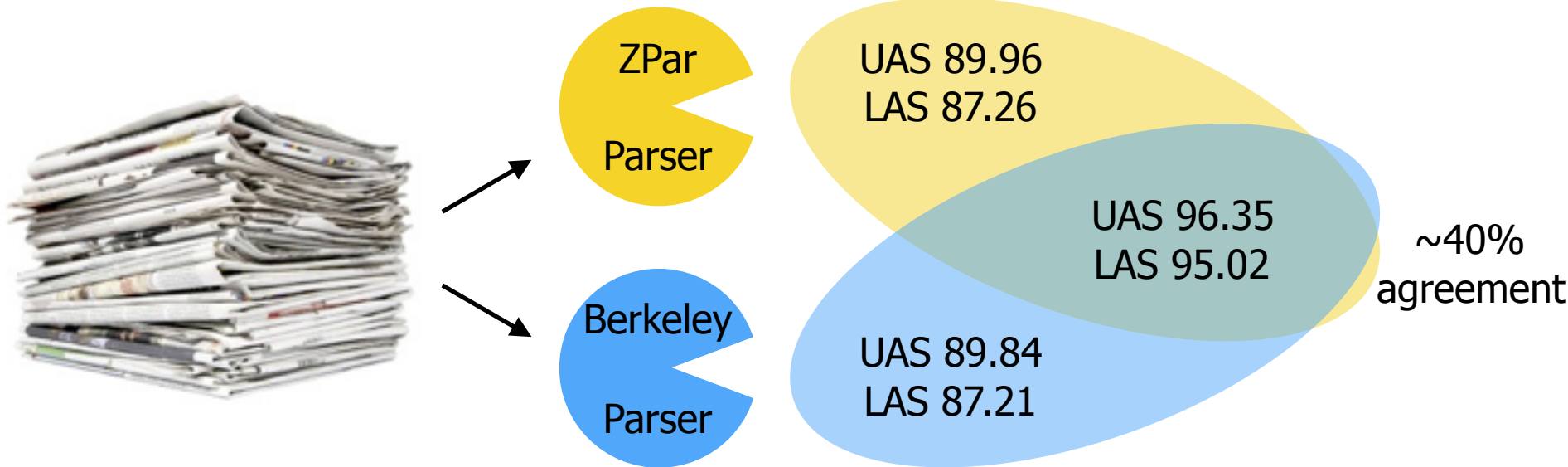
# English Results (WSJ 23)

---

<b>Method</b>	<b>UAS</b>	<b>LAS</b>	<b>Beam</b>
3rd-order Graph-based (ZM2014)	93.22	91.02	-
Transition-based Linear (ZN2011)	93.00	90.95	32
NN Baseline (Chen & Manning, 2014)	91.80	89.60	1
NN Better SGD (Weiss et al., 2015)	92.58	90.54	1
NN Deeper Network (Weiss et al., 2015)	93.19	91.18	1
NN Perceptron (Weiss et al., 2015)	93.99	92.05	8
NN CRF (Andor et al., 2016)	94.61	92.79	32
NN CRF Semi-Supervised (Andor et al.)	95.01	92.97	32
S-LSTM (Dyer et al., 2015)	93.20	90.90	1
Contrastive NN (Zhou et al., 2015)	92.83	—	100

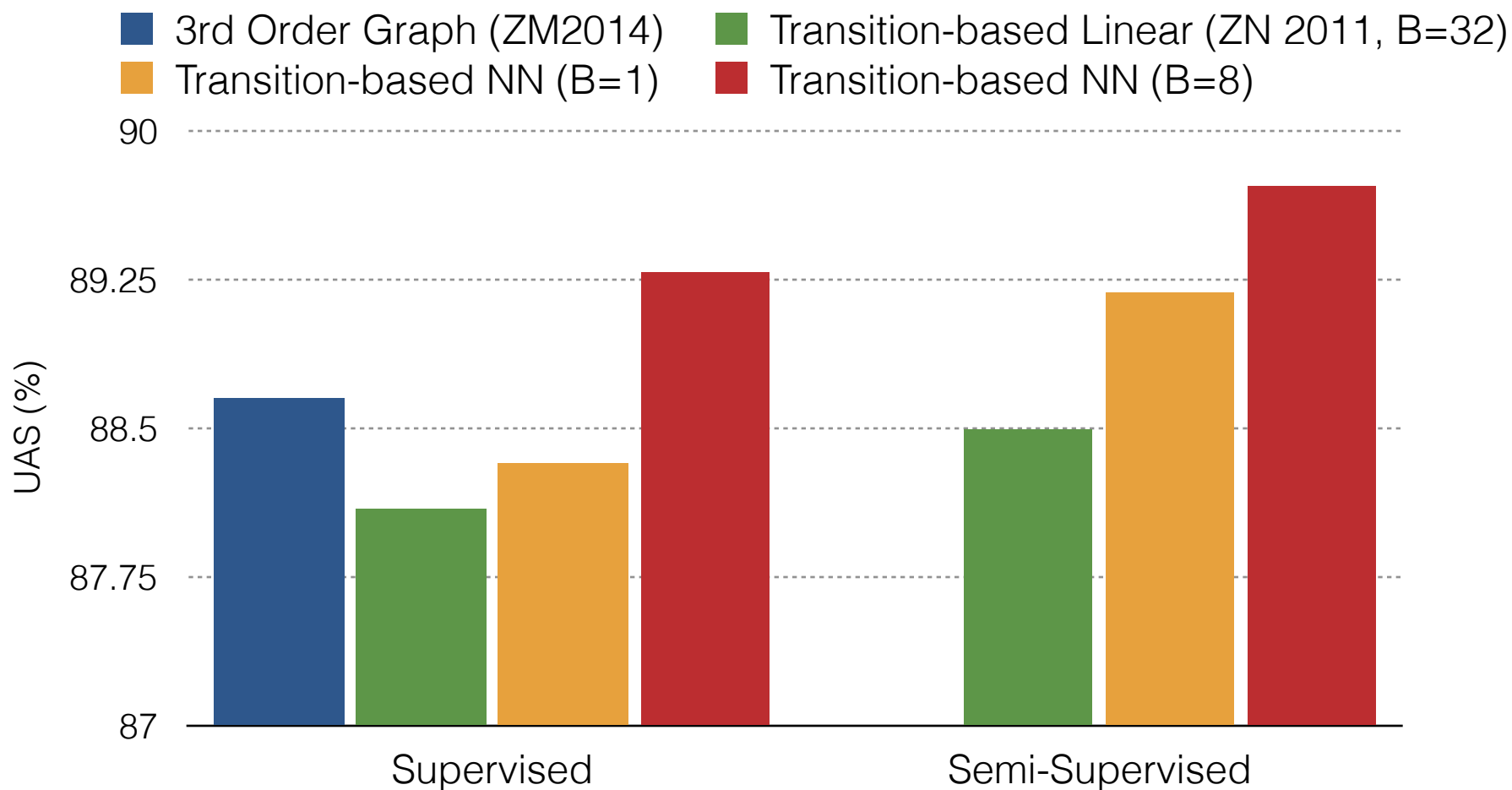
# Tri-Training

[Zhou et al. '05, Li et al. '14]



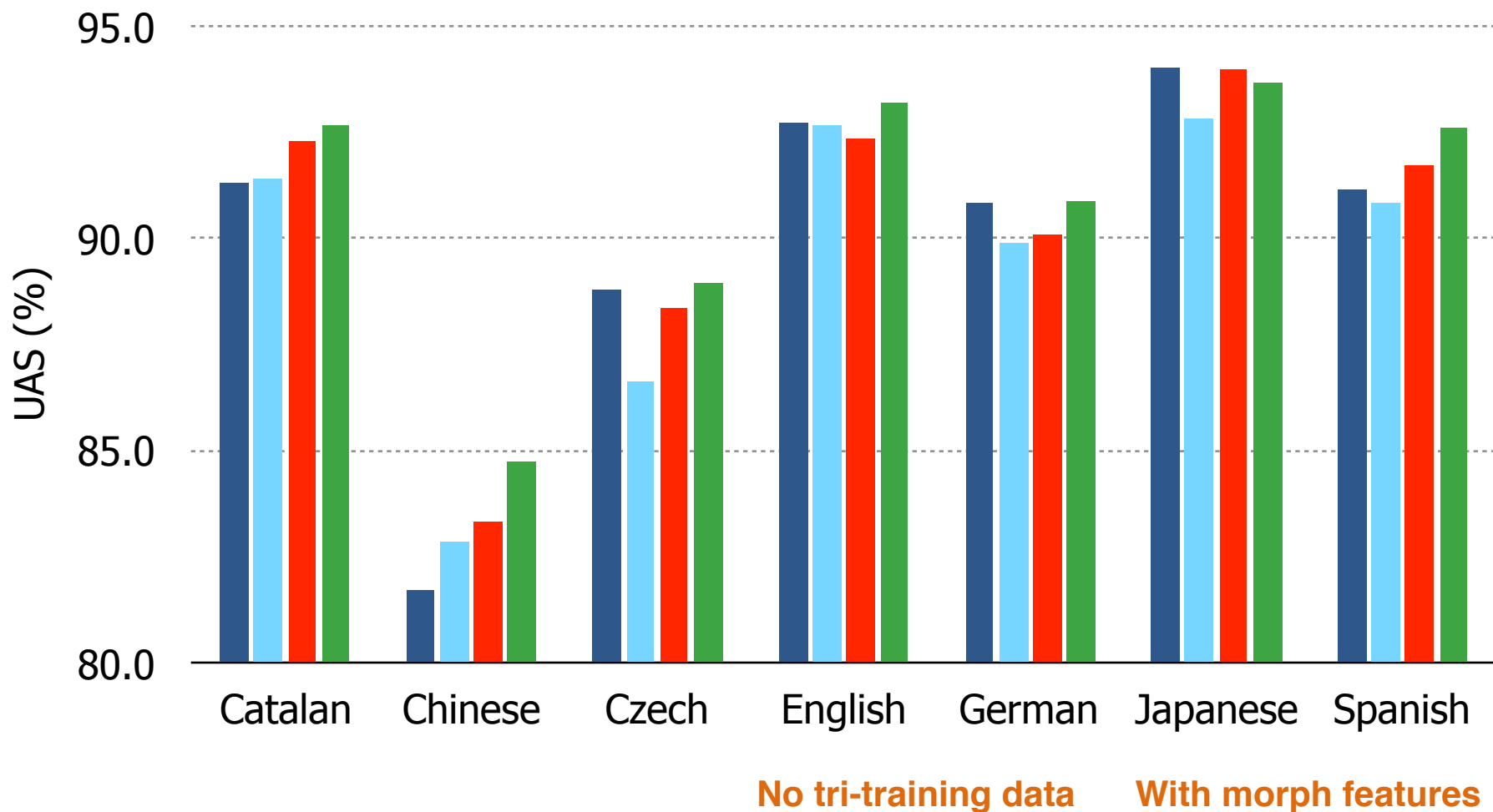
# English Out-of-Domain Results

- Train on WSJ + Web Treebank + QuestionBank
- Evaluate on Web



# Multilingual Results

■ Tensor-Based Graph (Lei et al. '14)      ■ 3rd-Order Graph (Zhang & McDonald '14)  
■ Transition-based NN (Weiss et al. '15)      ■ Transition-based CRF (Andor et al. 16)





# SyntaxNet and Parsey McParseface

WIRED



ARTIFICIAL INTELLIGENCE

Google Has Open Sourced Its AI for Understanding Language

VB

Google open-sources SyntaxNet, a natural-language understanding library for TensorFlow

MIT  
Technology  
Review

Robotics

Google's Algorithms Decode Language like a Trained Linguist

Friday - May 13, 2016

TSJ

$C_i$  I booked a ticket to

Stack

SHIFT LEFT\_ARC

TECH

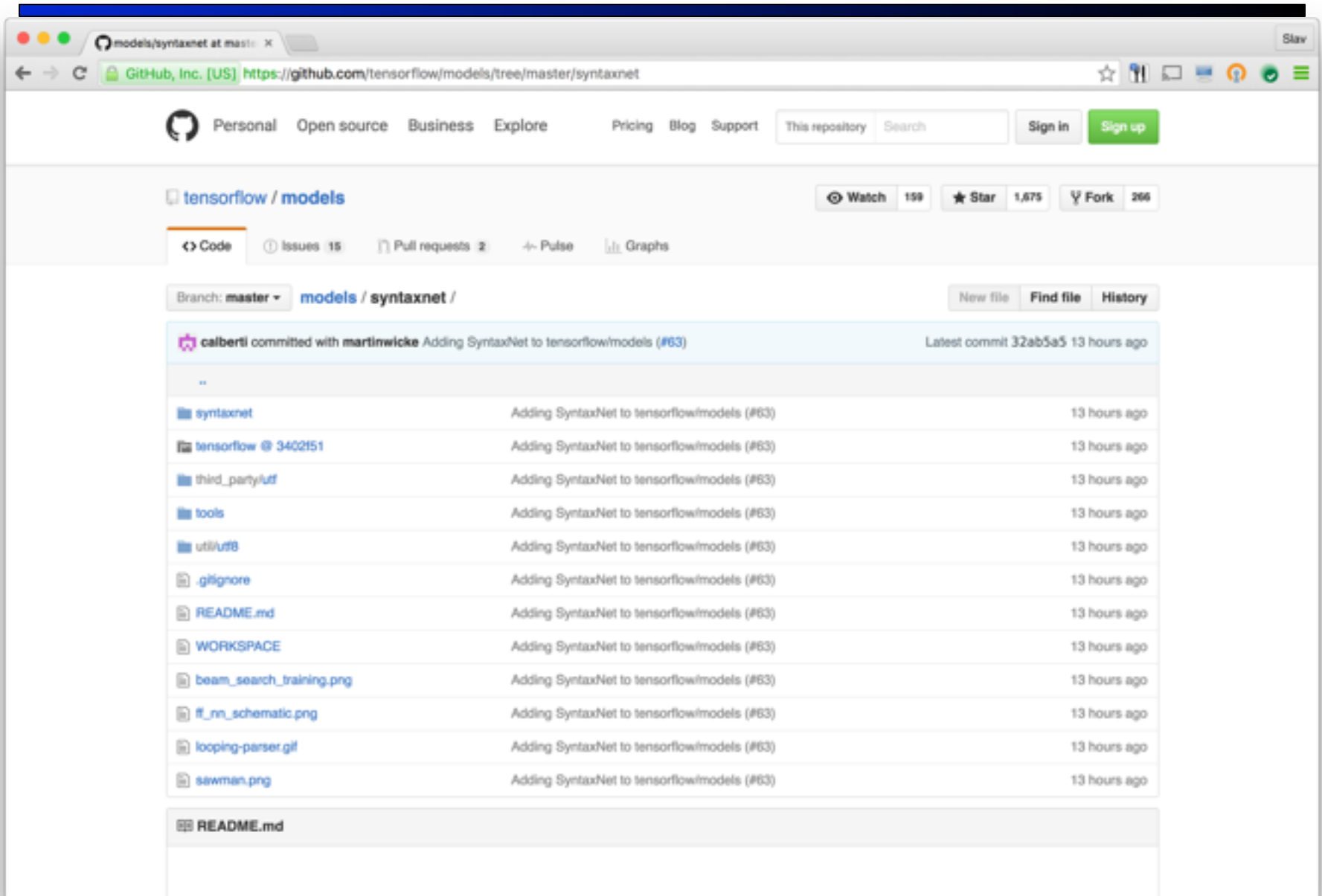
Google just open sourced something called 'Parsey McParseface,' and it could change AI forever

THE WALL STREET JOURNAL.

Google's Artificial-Intelligence Tool Is Offered for Free

Alphabet subsidiary is making code freely available for anyone to distribute or modify

# SyntaxNet and Parsey



The screenshot shows a web browser window displaying the GitHub repository page for `tensorflow/models/syntaxnet`. The browser's address bar shows the URL `https://github.com/tensorflow/models/tree/master/syntaxnet`. The repository page includes navigation links for Personal, Open source, Business, Explore, Pricing, Blog, and Support. It also features a search bar, a 'Sign in' button, and a 'Sign up' button. The repository name is `tensorflow / models`, with 199 Watchers, 1,675 Stars, and 266 Forks. The current branch is `master`, and the file path is `models / syntaxnet /`. A commit by `calberti` is highlighted, with the message `Adding SyntaxNet to tensorflowmodels (#63)` and a timestamp of 13 hours ago. Below the commit, a list of files is shown, all with the same commit message and timestamp. The files include `..`, `syntaxnet`, `tensorflow @ 3402f51`, `third_party/utf`, `tools`, `util/utf8`, `.gltignore`, `README.md`, `WORKSPACE`, `beam_search_training.png`, `fl_rnn_schematic.png`, `looping-parser.gif`, and `sawman.png`. At the bottom, the `README.md` file is partially visible.

models/syntaxnet at master · tensorflow/models

Personal Open source Business Explore Pricing Blog Support This repository Search Sign in Sign up

tensorflow / models Watch 199 Star 1,675 Fork 266

Code Issues 15 Pull requests 2 Pulse Graphs

Branch: master models / syntaxnet / New file Find file History

calberti committed with martinwicke Adding SyntaxNet to tensorflowmodels (#63) Latest commit 32ab5a5 13 hours ago

..		
syntaxnet	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
tensorflow @ 3402f51	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
third_party/utf	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
tools	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
util/utf8	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
.gltignore	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
README.md	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
WORKSPACE	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
beam_search_training.png	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
fl_rnn_schematic.png	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
looping-parser.gif	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago
sawman.png	Adding SyntaxNet to tensorflowmodels (#63)	13 hours ago

README.md

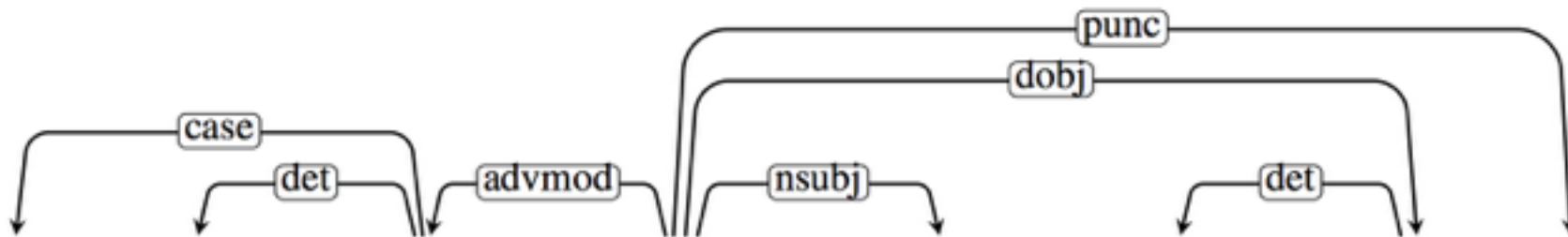
# LSTMs vs SyntaxNet

---

	LSTMs	SyntaxNet
Accuracy	+	++
Efficiency	-	+
End-to-End	++	- (yet)
Recurrence	+	- (yet)

# Universal Dependencies

## Stanford Universal++ Dependencies



## Intersect++ Morphological Features

## Google Universal++ POS Tags

1.1 Im      1.2      2 Restaurant      3 isst      4 Maria      5 den      6 Fisch      7 .

# Universal Dependencies

## UD Treebanks

Amharic	-	-	?	-		
Ancient Greek	244K	00	0	0	✓	000000
Ancient Greek-PROIEL	206K	00	-	0	✓	000000
Arabic	242K	00	-	0	✓	000000
Basque	121K	00	0	0	✓	000000
Bulgarian	156K	00	0	0	✓	000000
Buryat	3K	0	-	0	✓	000000
Catalan	530K	00	0	0	✓	000000
Chinese	123K	0	0	0	✓	000000
Coptic	4K	0	0	0	✓	000000
Croatian	87K	00	-	0	✓	000000
Czech	1,503K	00	0	0	✓	000000
Czech-CAC	493K	00	0	0	✓	000000
Czech-CLTT	35K	00	0	0	✓	000000
Danish	100K	00	0	0	✓	000000
Dutch	209K	00	-	0	✓	000000
Dutch-LassySmall	98K	00	-	0	✓	000000
English	254K	00	0	0	✓	000000
English-ESL	97K	0	0	0	✓	000000
English-LinES	82K	0	0	0	✓	000000
Estonian	234K	00	-	0	✓	000000
Finnish	181K	00	0	0	✓	000000
Finnish-FTB	159K	00	-	0	✓	000000
French	390K	00	0	0	✓	000000
Galician	138K	0	0	0	✓	000000
German	293K	0	-	0	✓	000000
Gothic	56K	00	-	0	✓	000000
Greek	59K	00	0	0	✓	000000
Hebrew	115K	0	-	0	✓	000000
Hindi	351K	00	-	0	✓	000000
Hungarian	42K	00	0	0	✓	000000
Indonesian	121K	0	-	0	✓	000000
Irish	23K	00	0	0	✓	000000
Italian	252K	00	0	0	✓	000000
Japanese-KTC	267K	0	0	0	✓	000000
Kazakh	4K	0	0	0	✓	000000
Korean	-	-	-	-	-	000000
Latin	47K	00	-	0	✓	000000
Latin-ITTB	291K	00	-	0	✓	000000
Latin-PROIEL	165K	00	-	0	✓	000000
Latvian	20K	00	-	0	✓	000000
Norwegian	311K	00	0	0	✓	000000
Old Church						

# Summary

---

- **Constituency Parsing**
  - CKY Algorithm
  - Lexicalized Grammars
  - Latent Variable Grammars
  - Conditional Random Field Parsing
  - Neural Network Representations
- **Dependency Parsing**
  - Eisner Algorithm
  - Maximum Spanning Tree Algorithm
  - Transition Based Parsing
  - Neural Network Representations