# Syntax and Parsing II

## Dependency Parsing

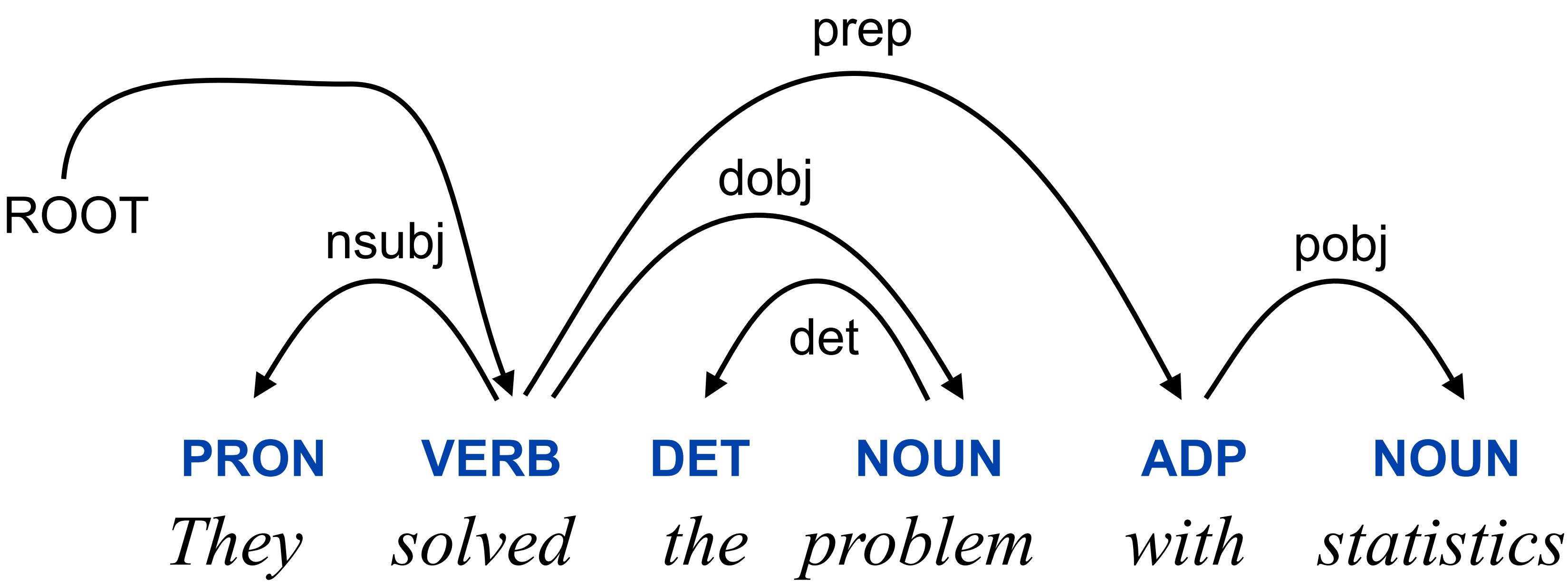
Slav Petrov – Google

Thanks to:

Dan Klein, Ryan McDonald, Alexander Rush, Joakim Nivre,
Greg Durrett, David Weiss, Luheng He, Timothy Dozat

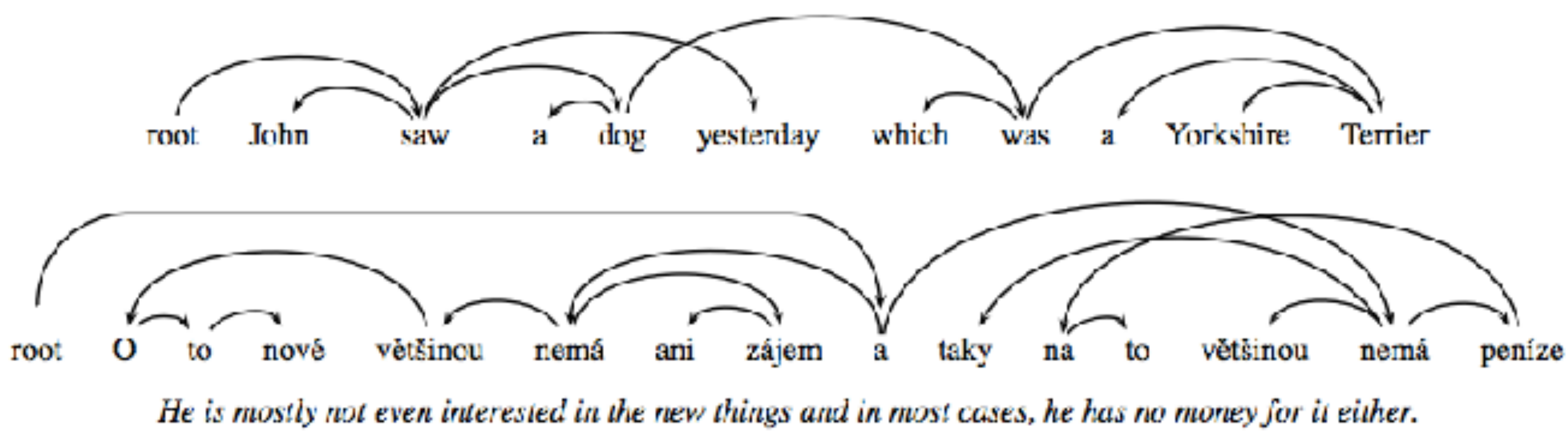
Lisbon Machine Learning School 2018

# Dependency Parsing



# (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):

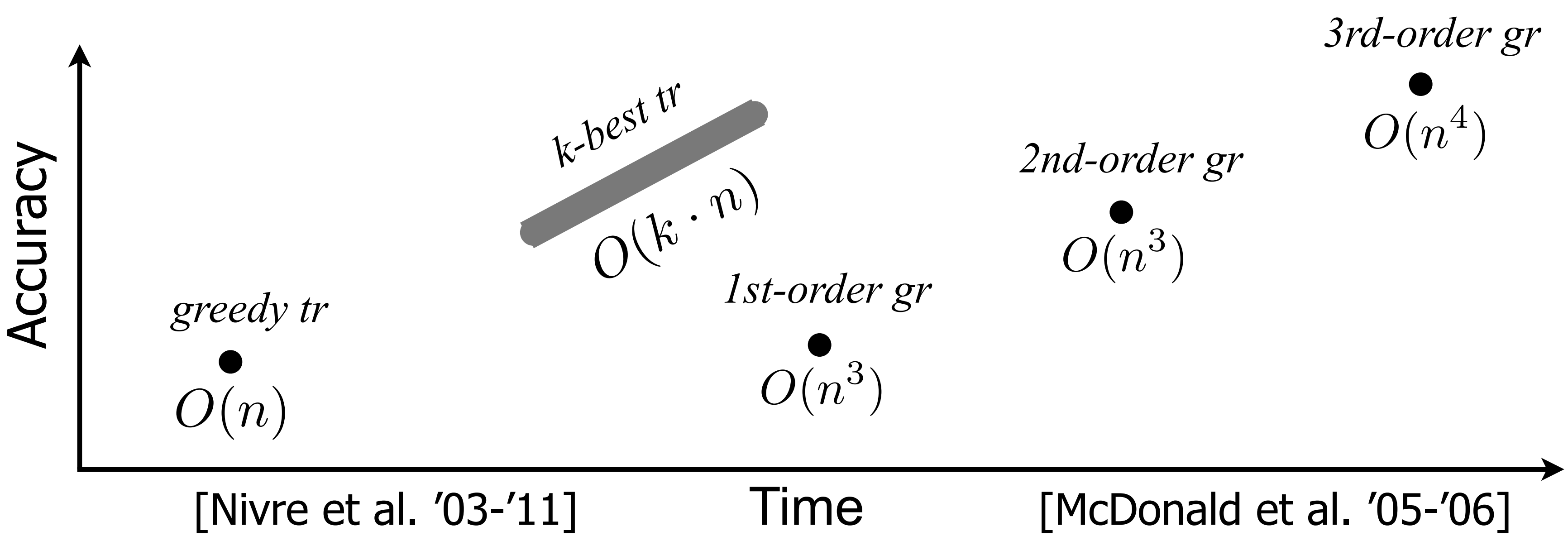


# Formal Conditions

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$
- $G$ 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

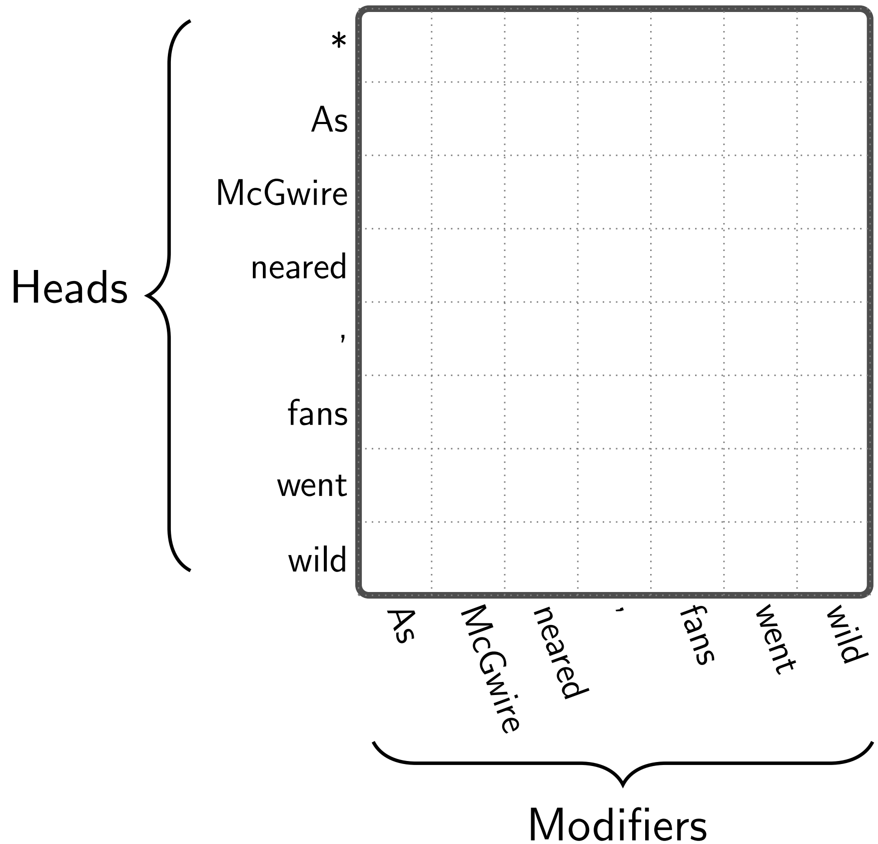
$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k$$

look familiar?

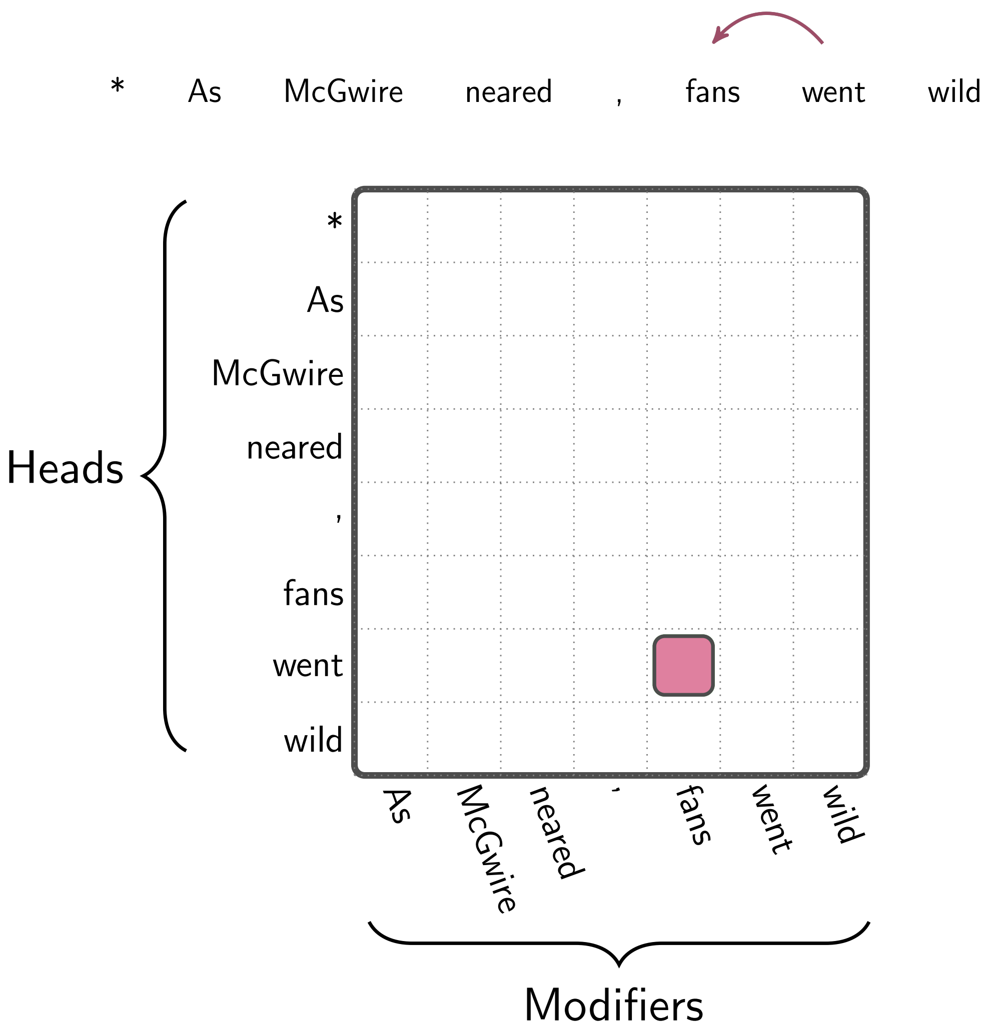
- $w_{ij}^k$ is the weight of creating a dependency from word $w_i$ to $w_j$ with label $l_k$
- Thus there is an assumption that each dependency decision is independent
  - Strong assumption! Will address this later.

# Dependency Representation

* As McGwire neared , fans went wild

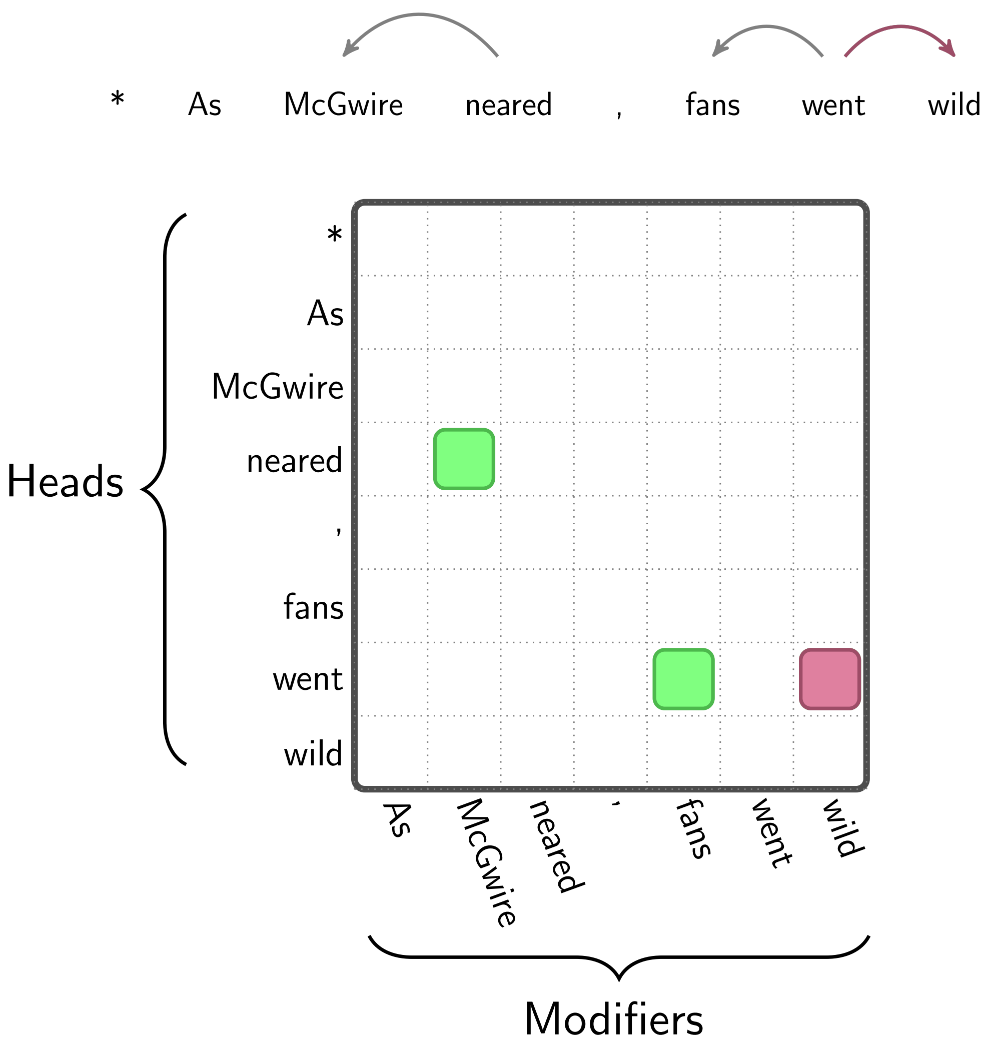


# Dependency Representation

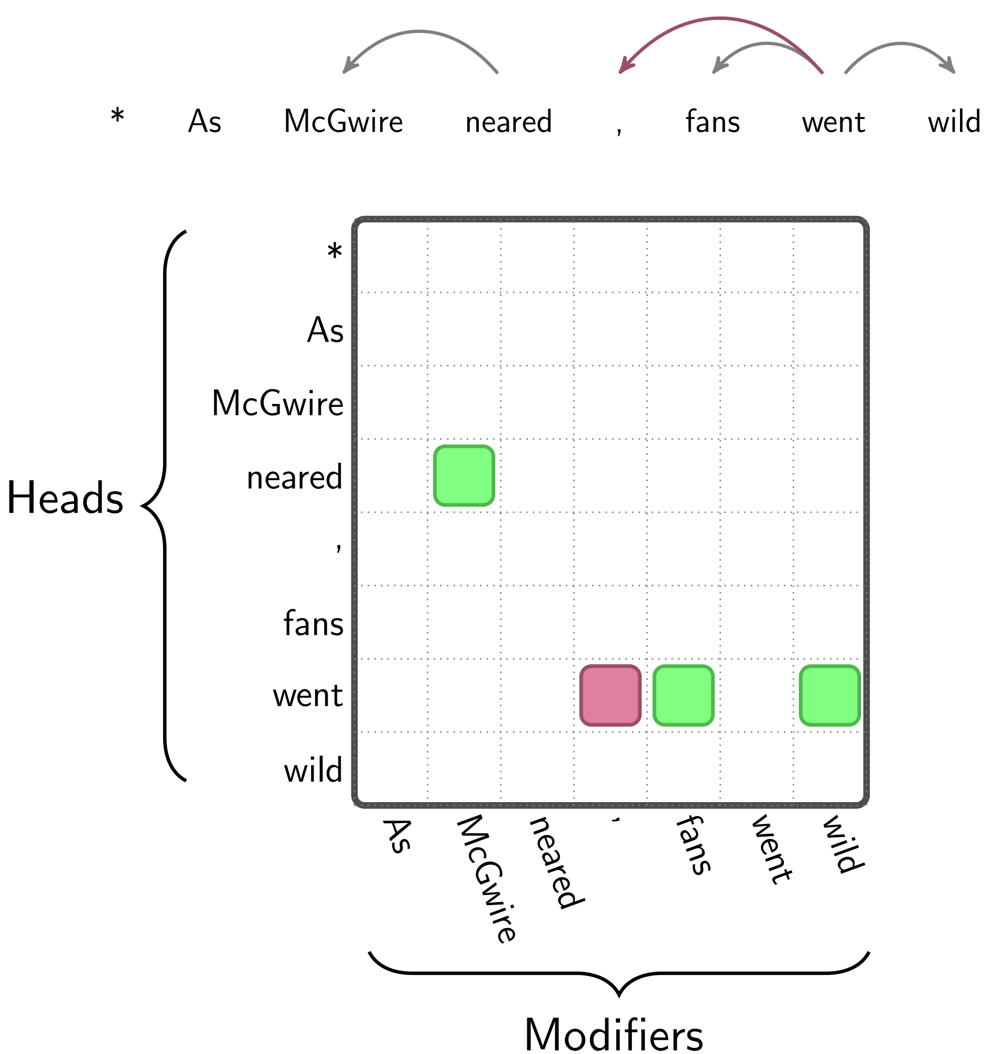


# Dependency Representation

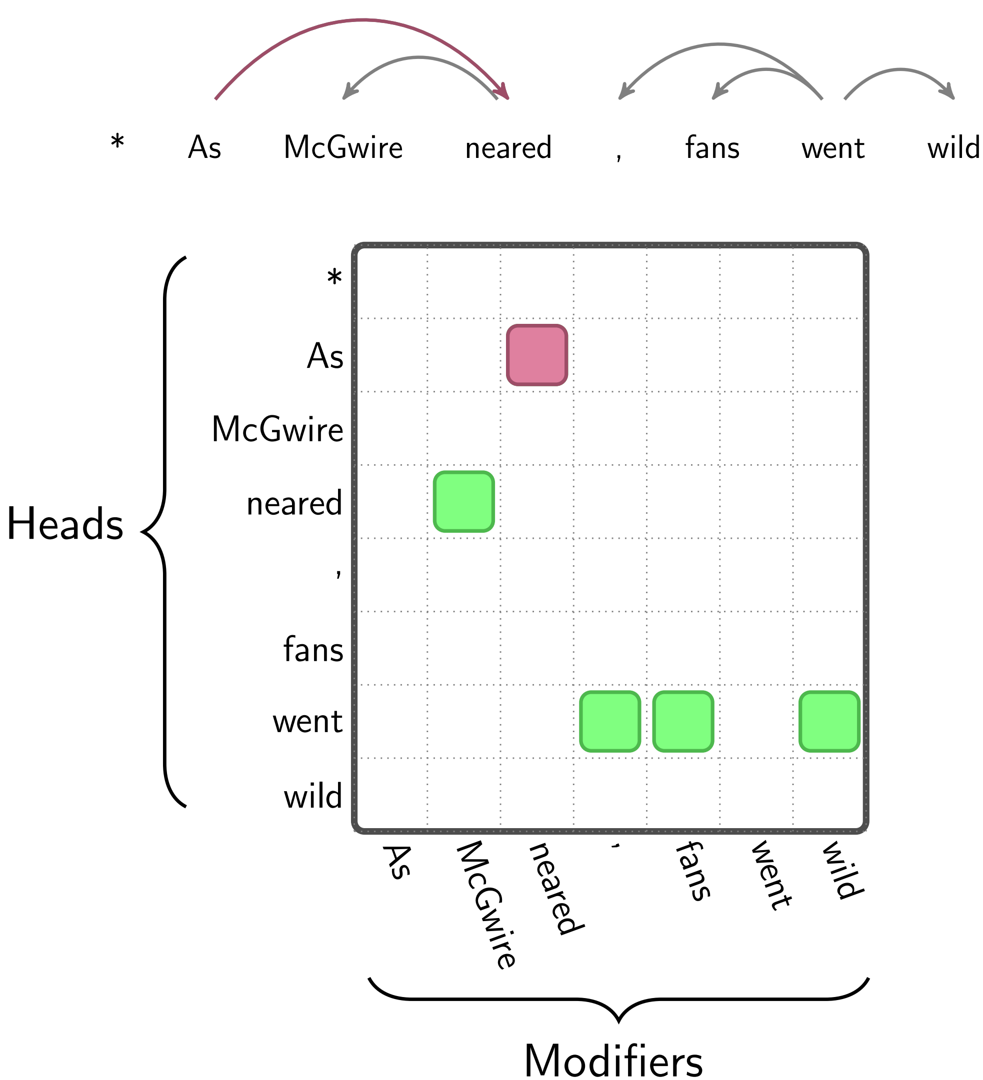
# Dependency Representation



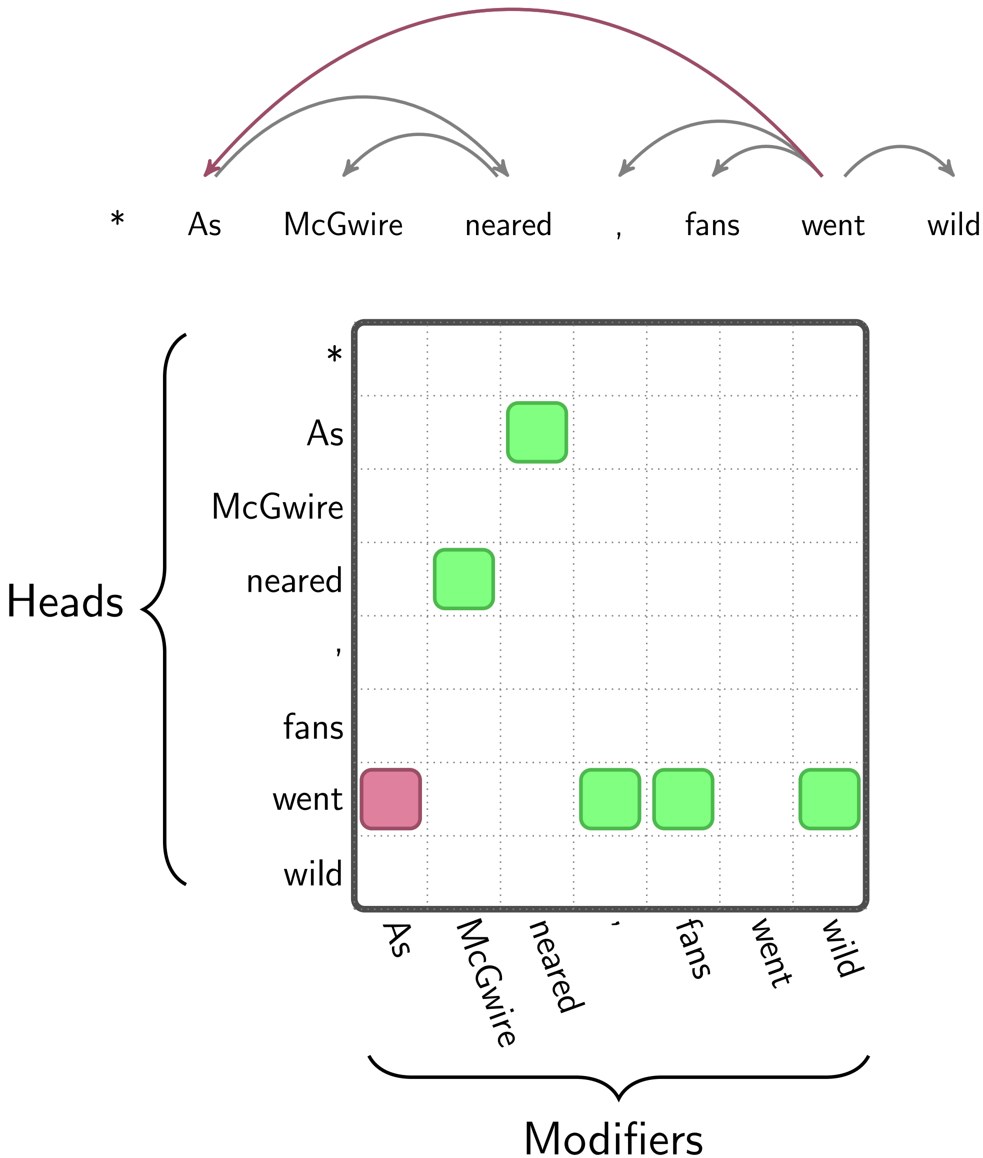
# Dependency Representation



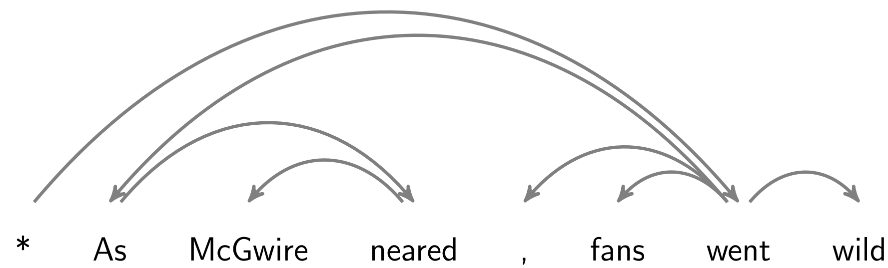
# Dependency Representation

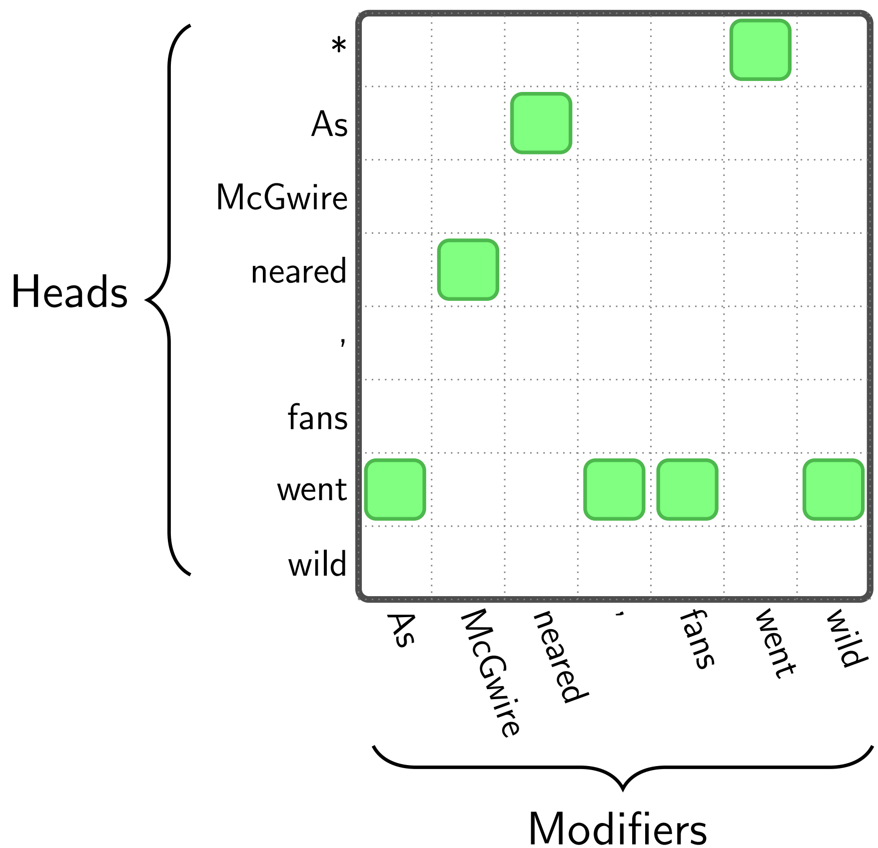


# Dependency Representation



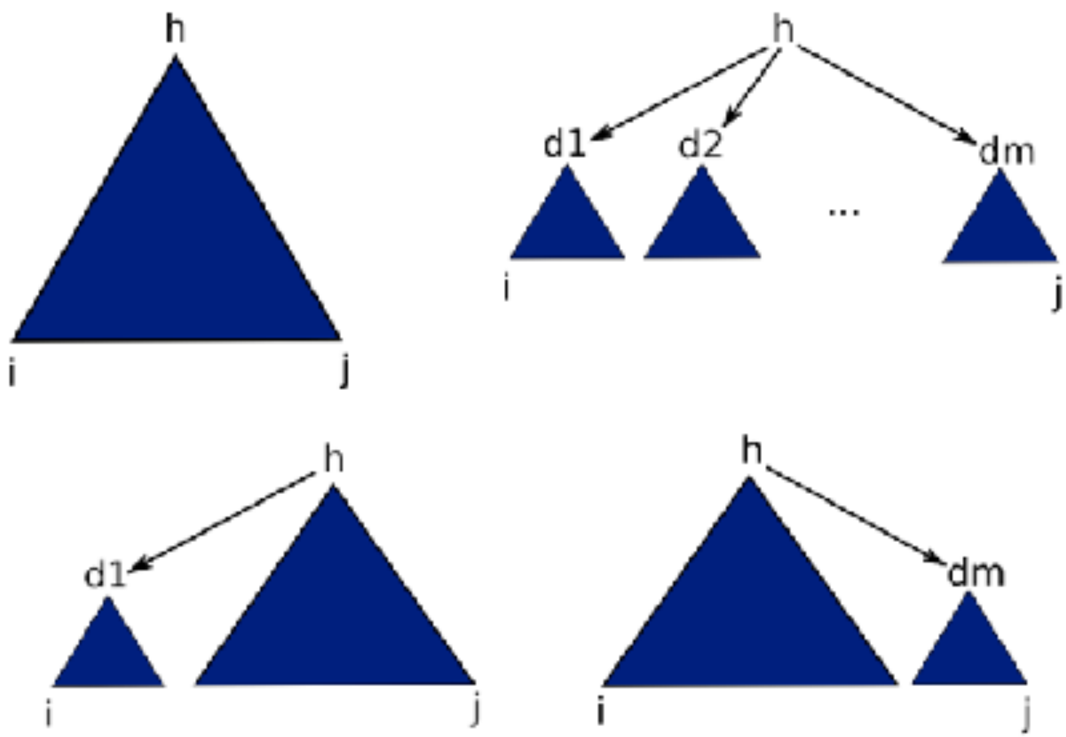
# Dependency Representation



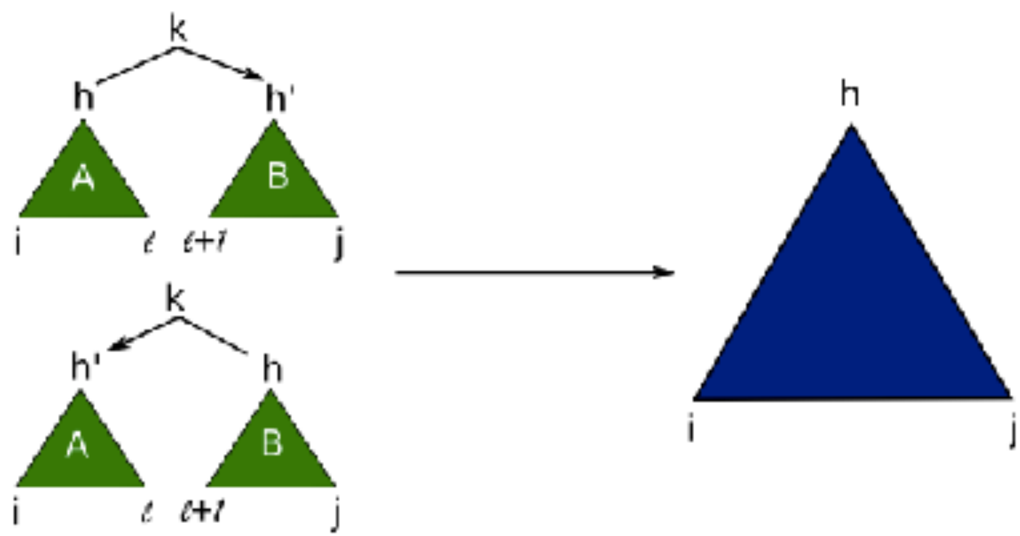
# Arc-factored Projective Parsing

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



# Arc-factored Projective Parsing

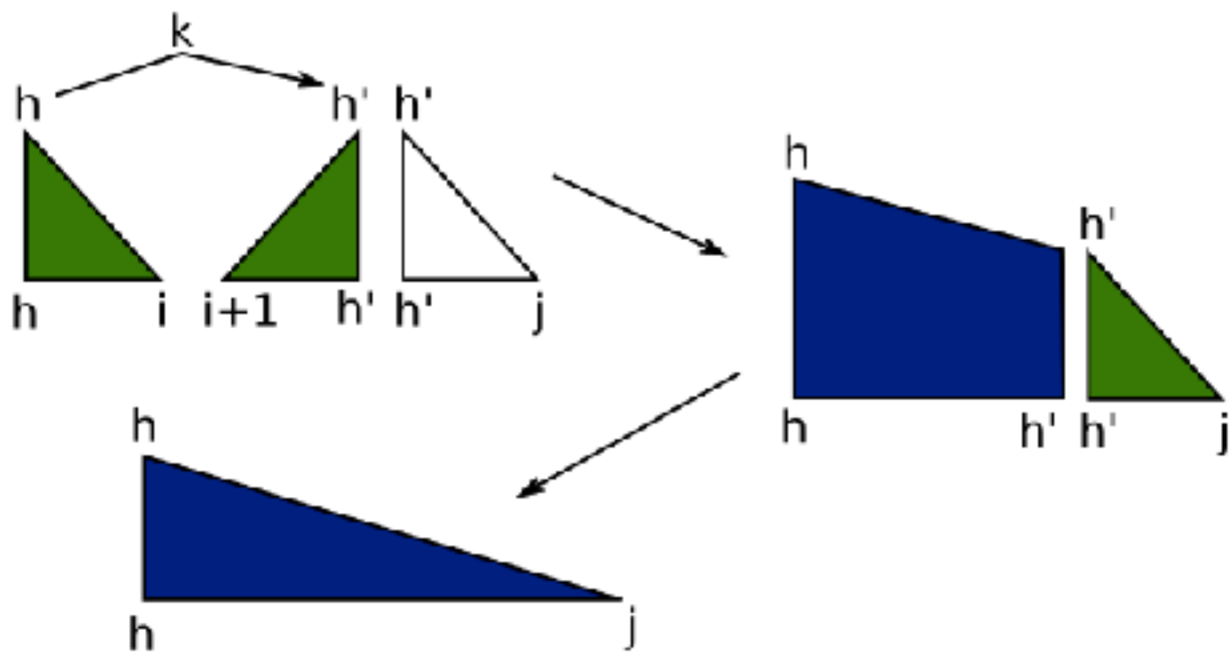
- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY



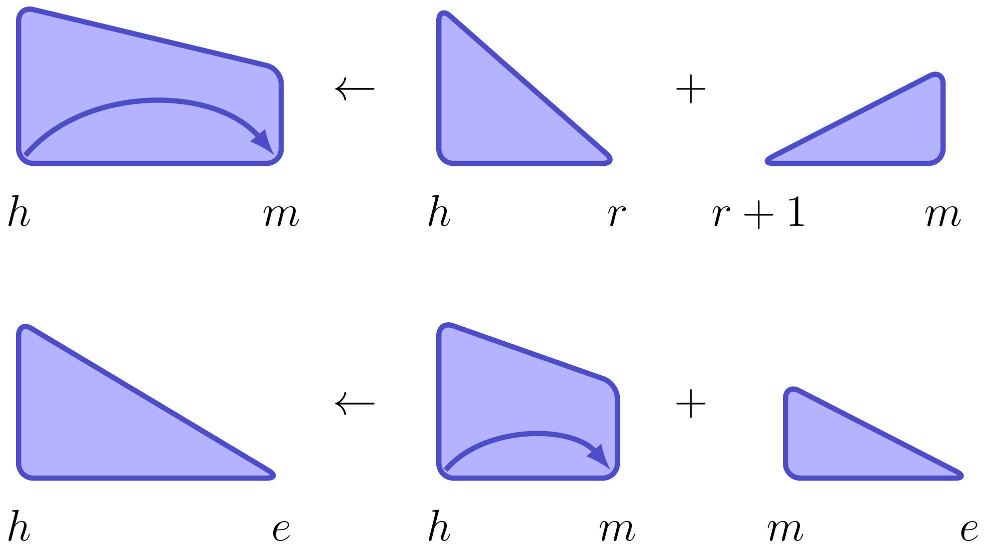

- Weight of new item: $\max_{l,j,k} \; w(A) \times w(B) \times w^{k}_{hh'}$
- Algorithm runs in $O(|L|n^5)$
- Use back-pointers to extract best parse (like CKY)

# 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

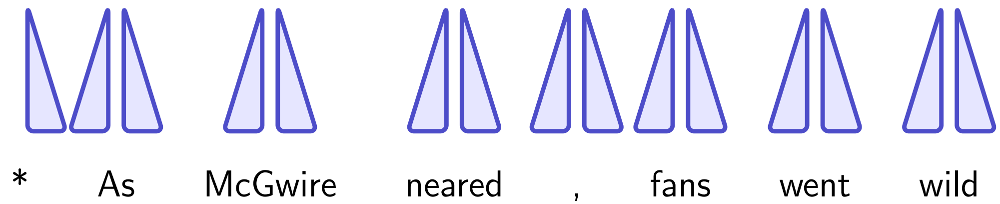


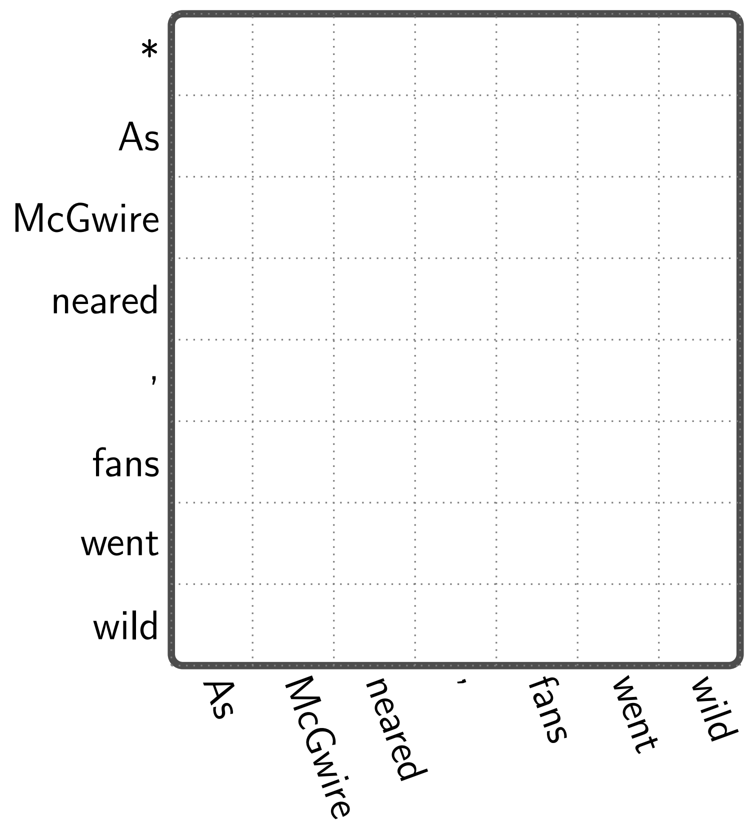
# Eisner First-Order Parsing



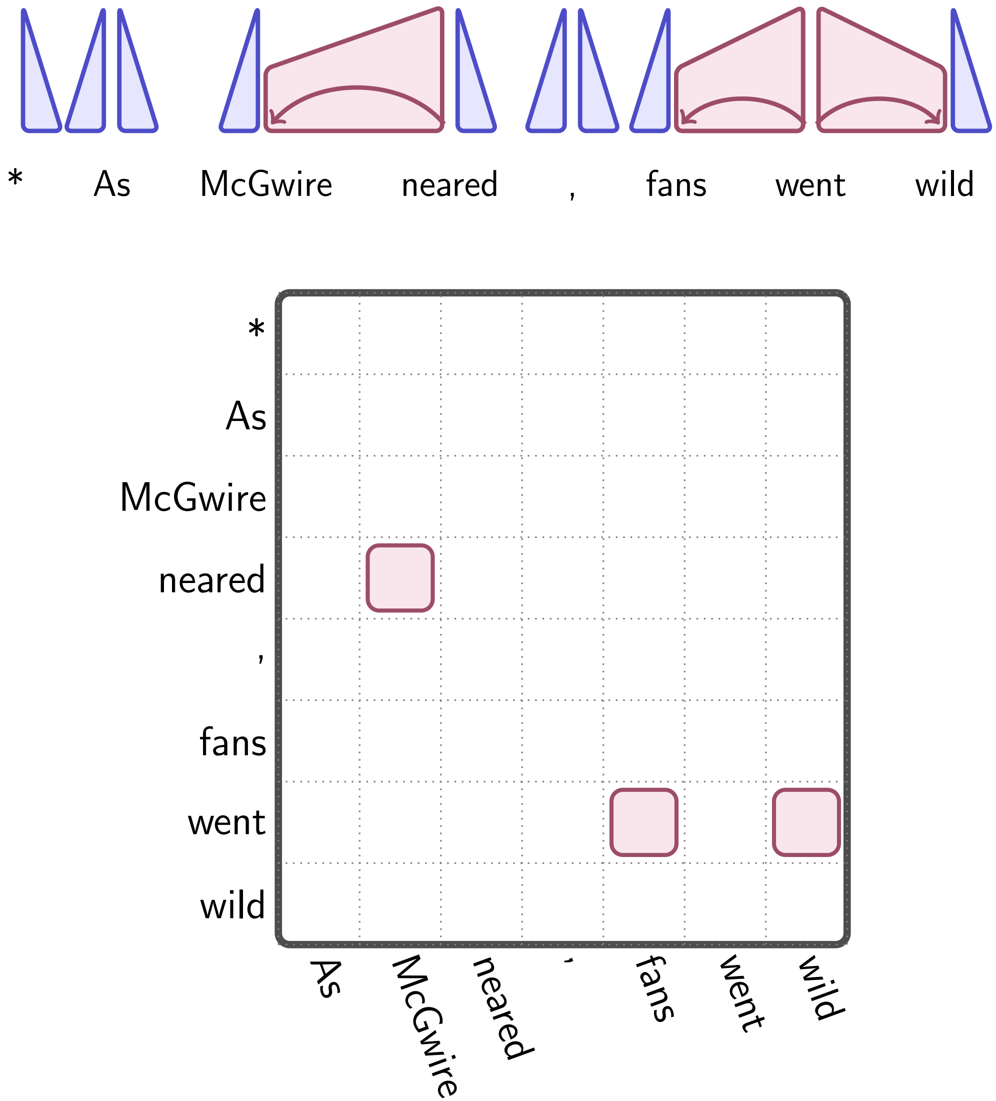

In practice also left arc version

# Eisner First-Order Parsing

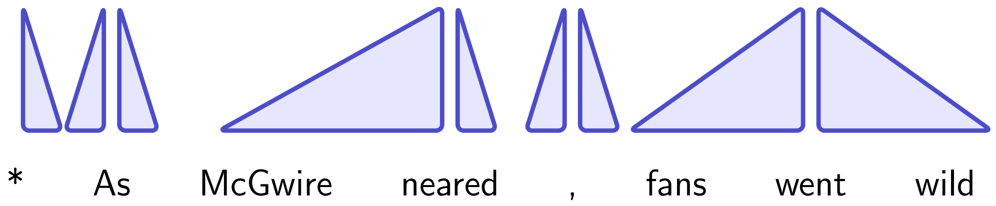


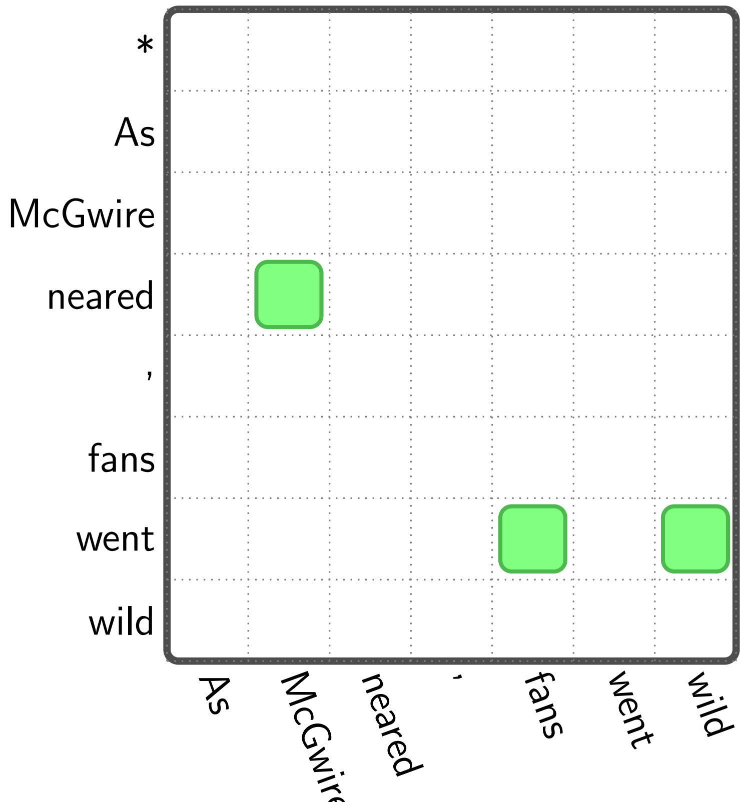



# Eisner First-Order Parsing

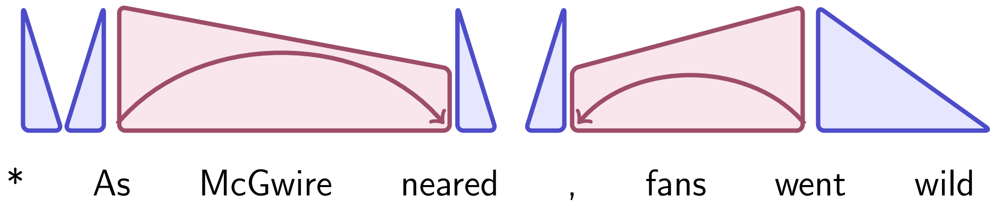


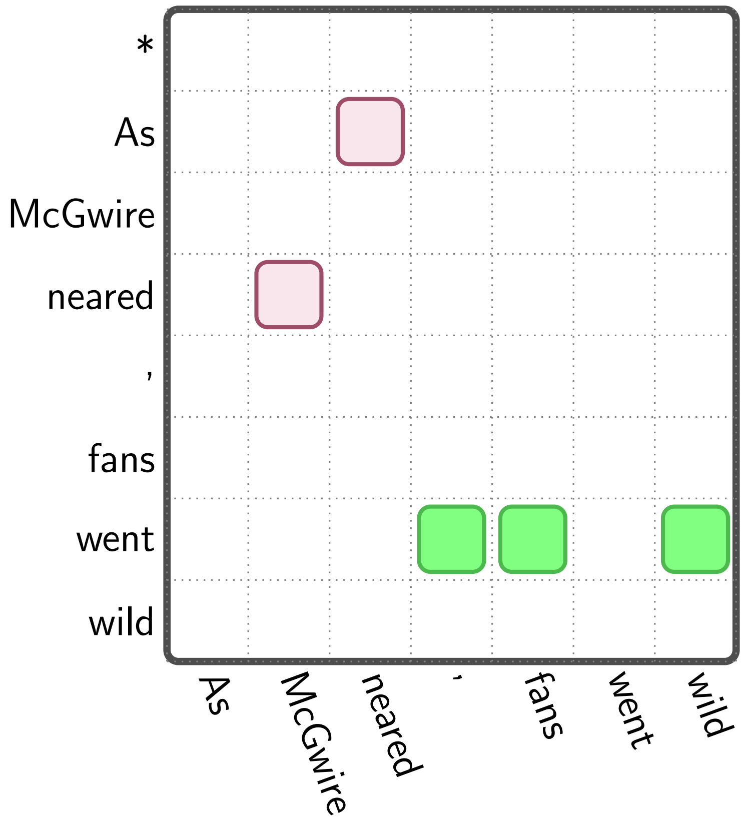
# Eisner First-Order Parsing



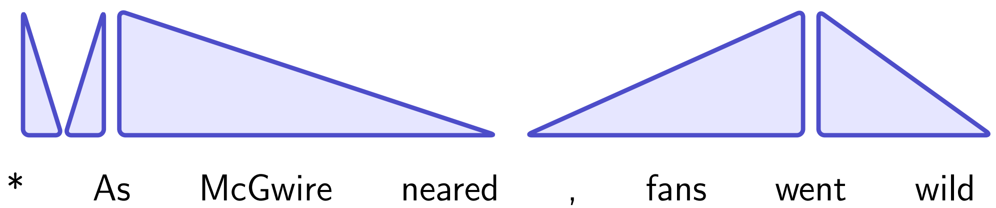



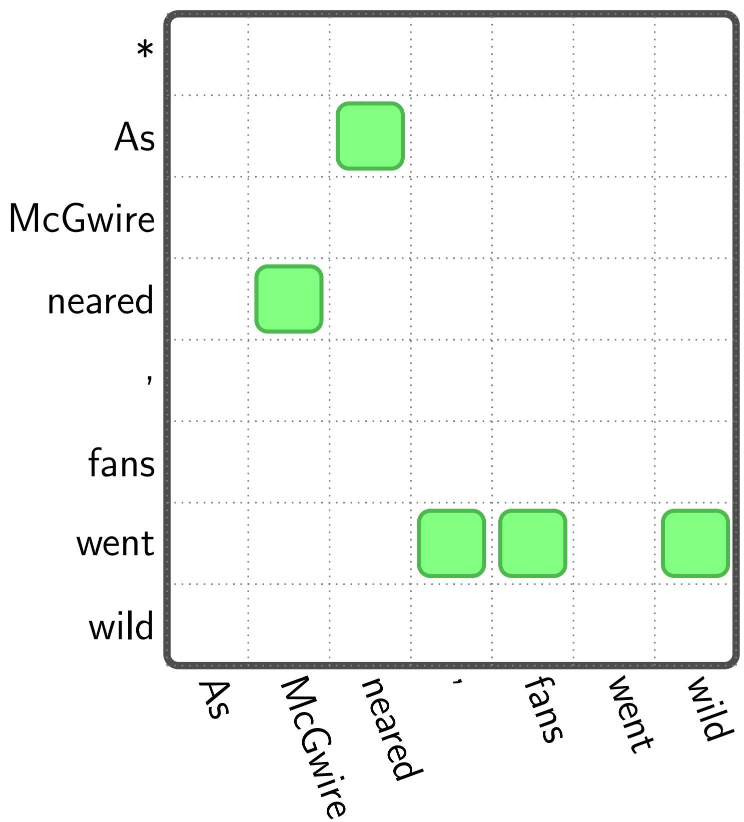
# Eisner First-Order Parsing



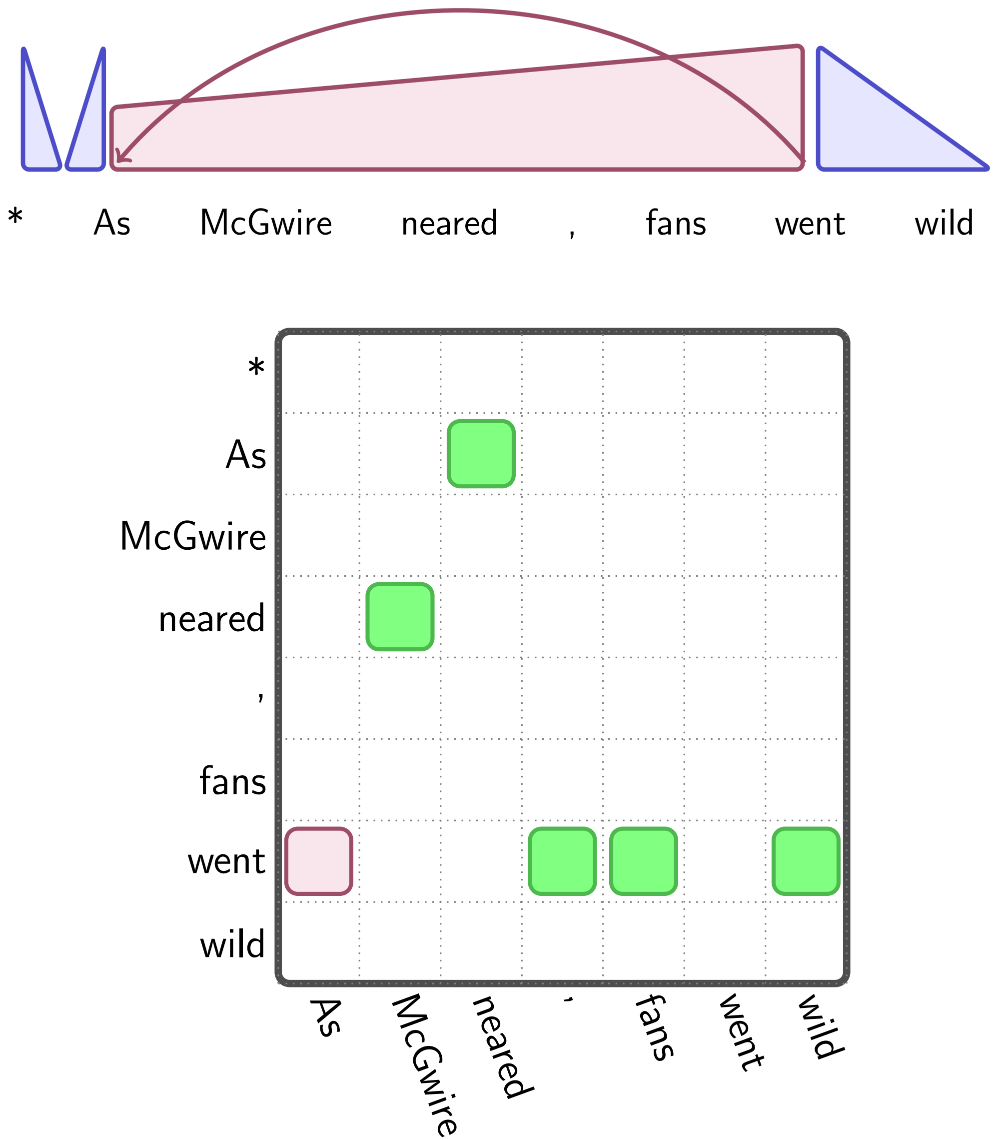



# Eisner First-Order Parsing

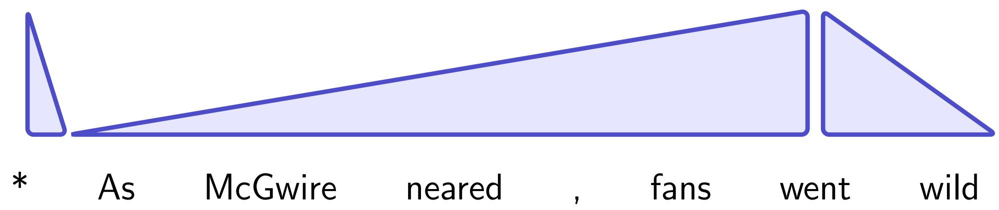


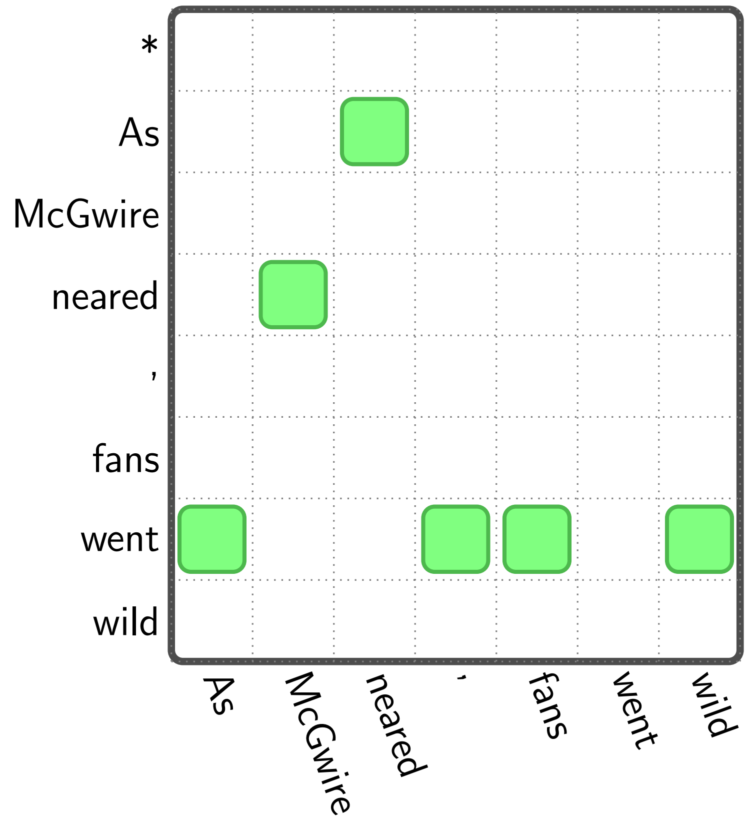



# Eisner First-Order Parsing

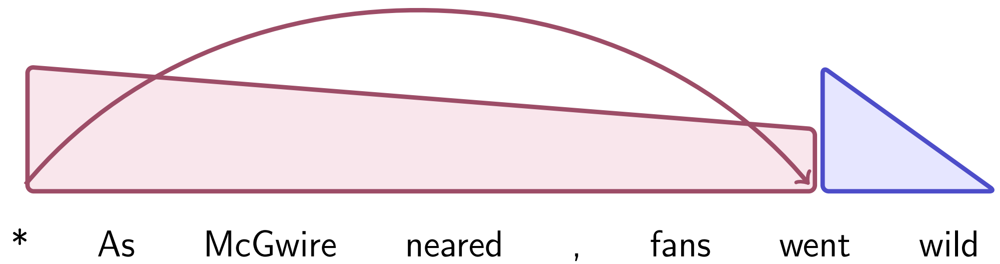


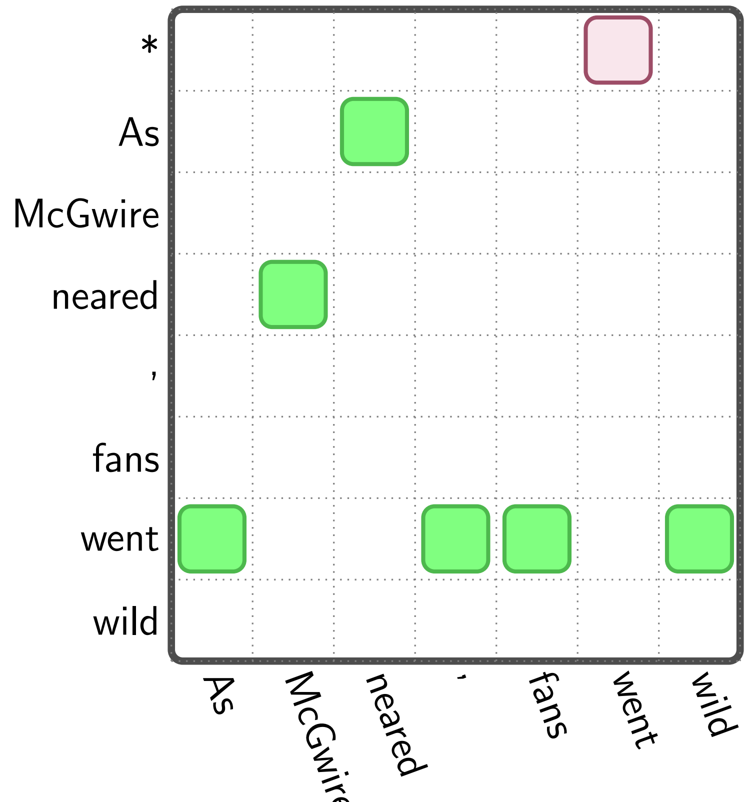
# Eisner First-Order Parsing



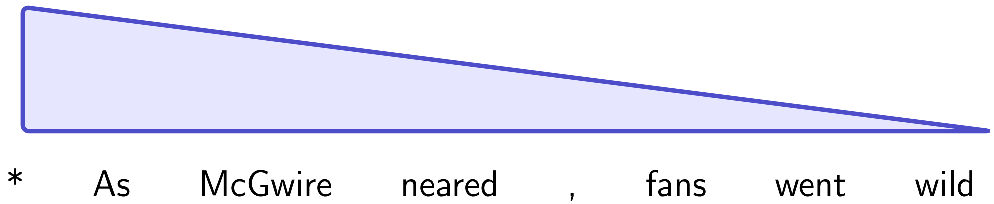



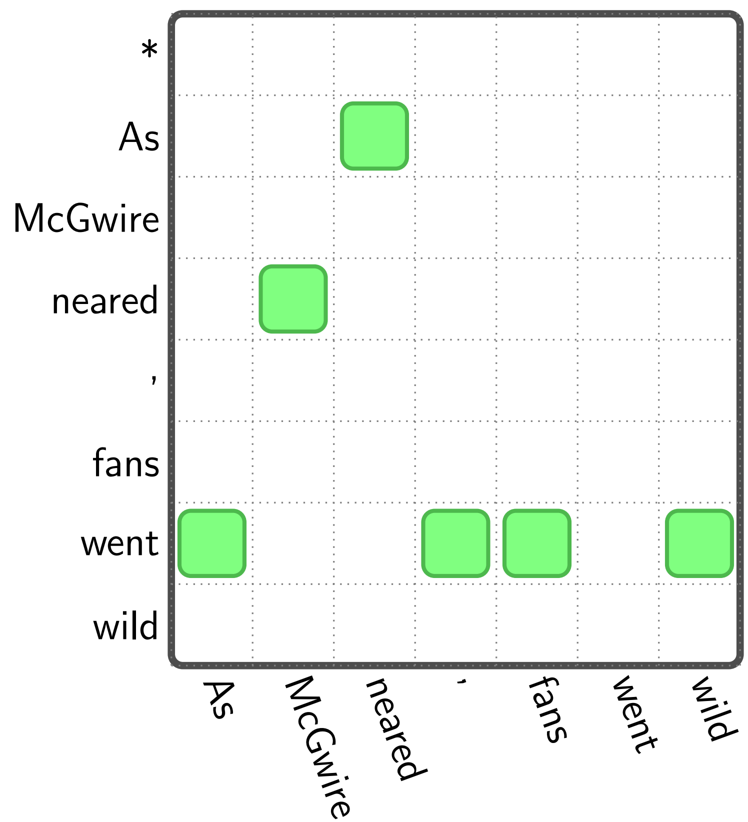
# Eisner First-Order Parsing



# 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

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

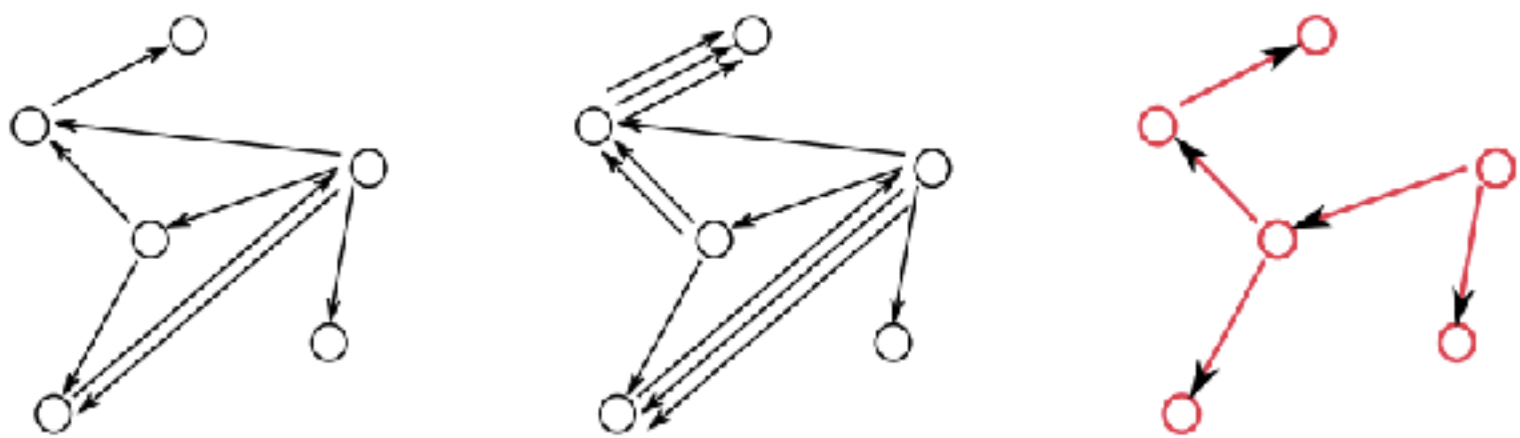
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} \; (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

  end for

end for

# Maximum Spanning Trees (MSTs)

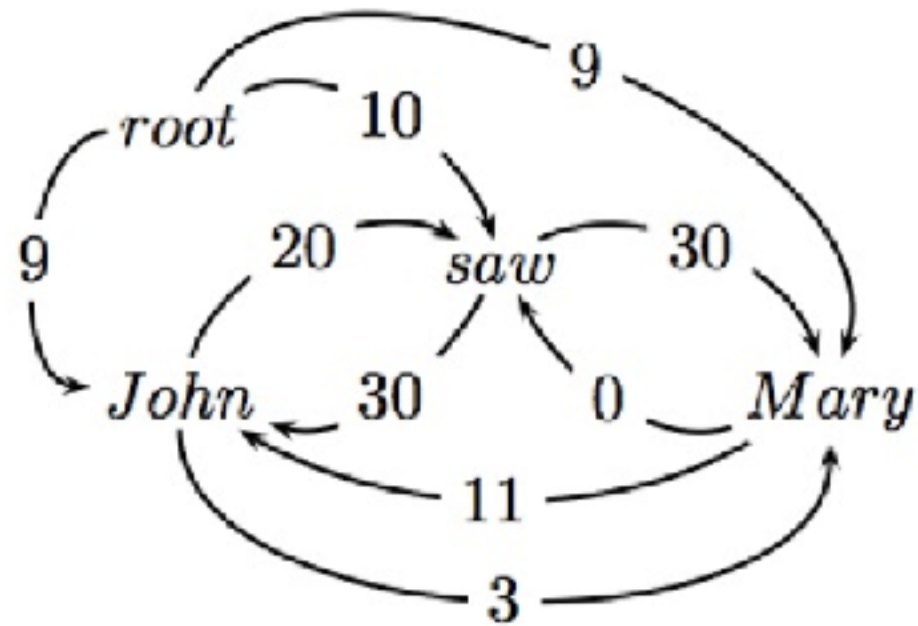
- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



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

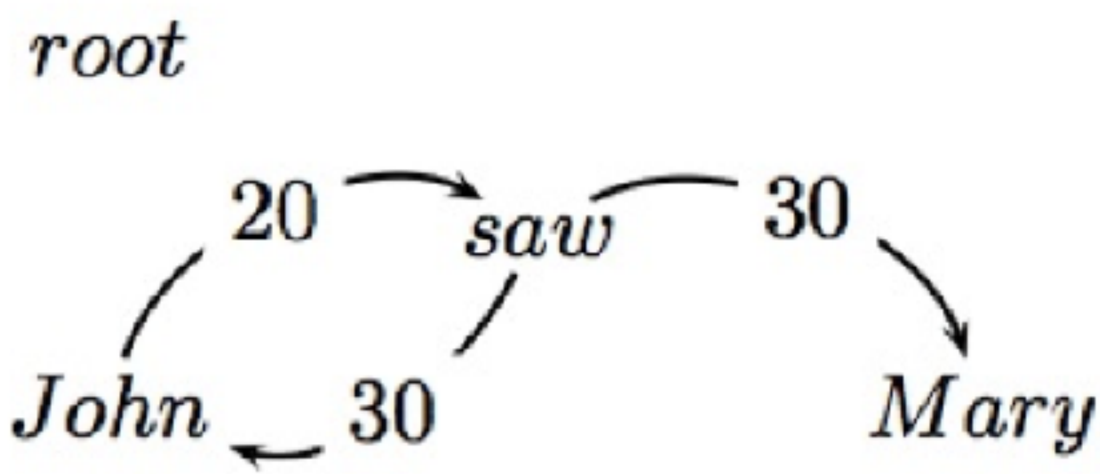
# Chu-Liu-Edmonds

- $x$ = root John saw Mary



# Chu-Liu-Edmonds

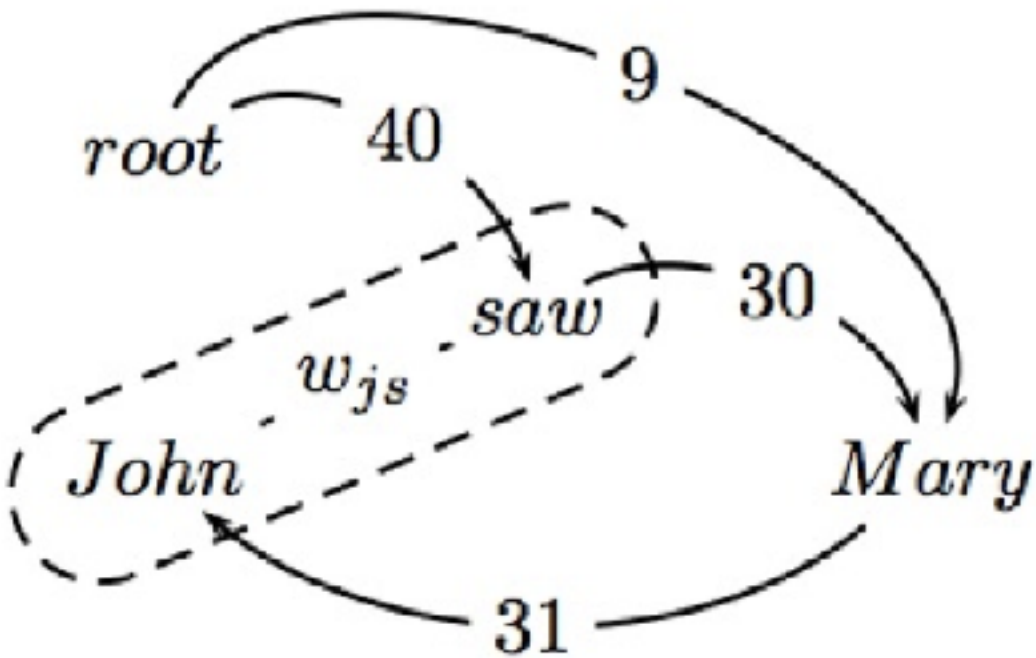
- Find highest scoring incoming arc for each vertex



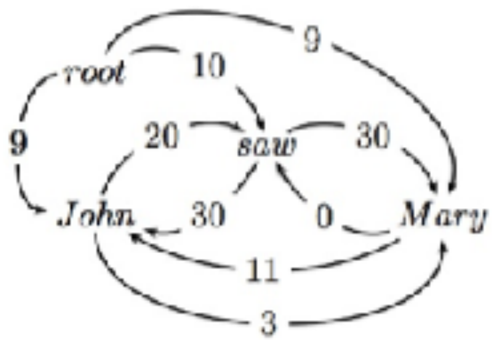

- If this is a tree, then we have found MST!!

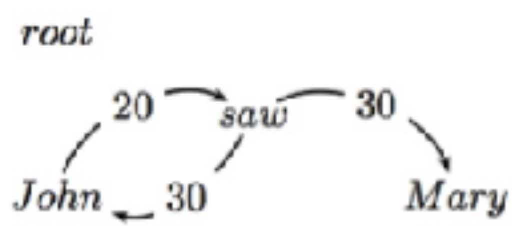
# Find Cycle and Contract

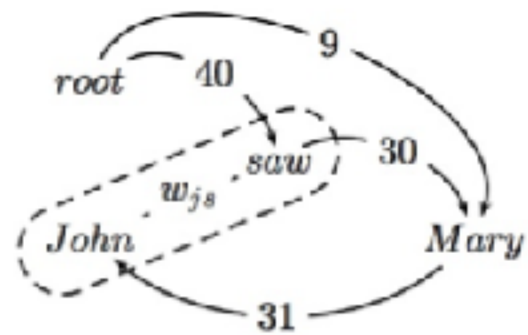
- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle



# Recalculate Edge Weights



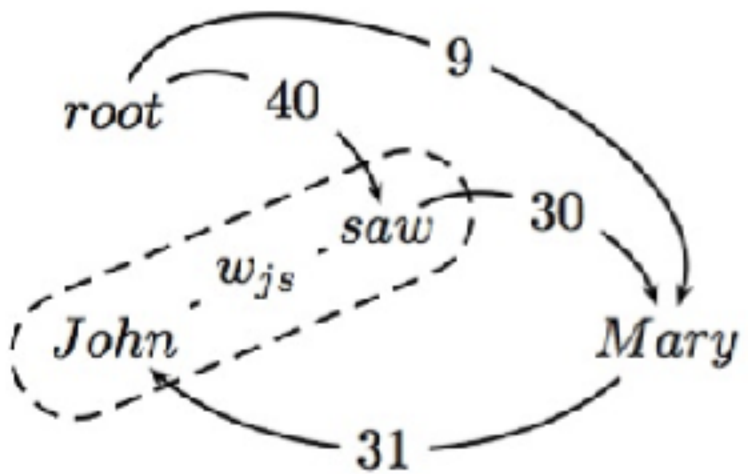




- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29

# Theorem

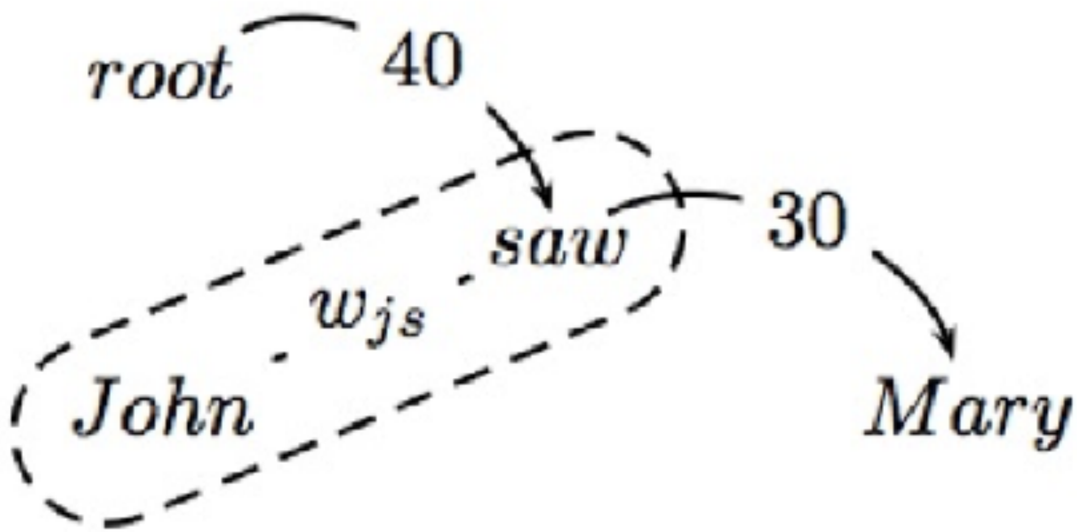
The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph




- 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 $< G_C, c, ma >=$ contract($G, C, w$)
6. Let $G =$ 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 $< G_C, c, ma >$

# Arc Weights

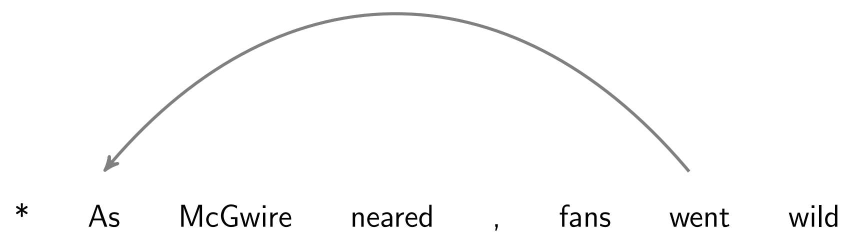
$$w_{ij}^{k} = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

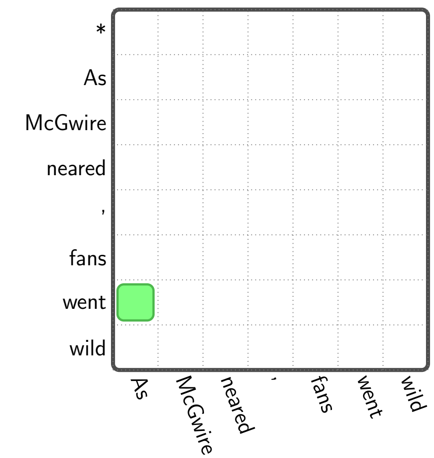
- Arc weights are a linear combination of features of the arc, **f**, and a corresponding weight vector **w**
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features

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


- Identities of the words wi and wj and the label lk
- Part-of-speech tags of the words wi and wj and the label lk
- Part-of-speech of words surrounding and between wi and wj
- Number of words between wi and wj , 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. $\mathbf{w}^{(0)} = 0; \; i = 0$
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]

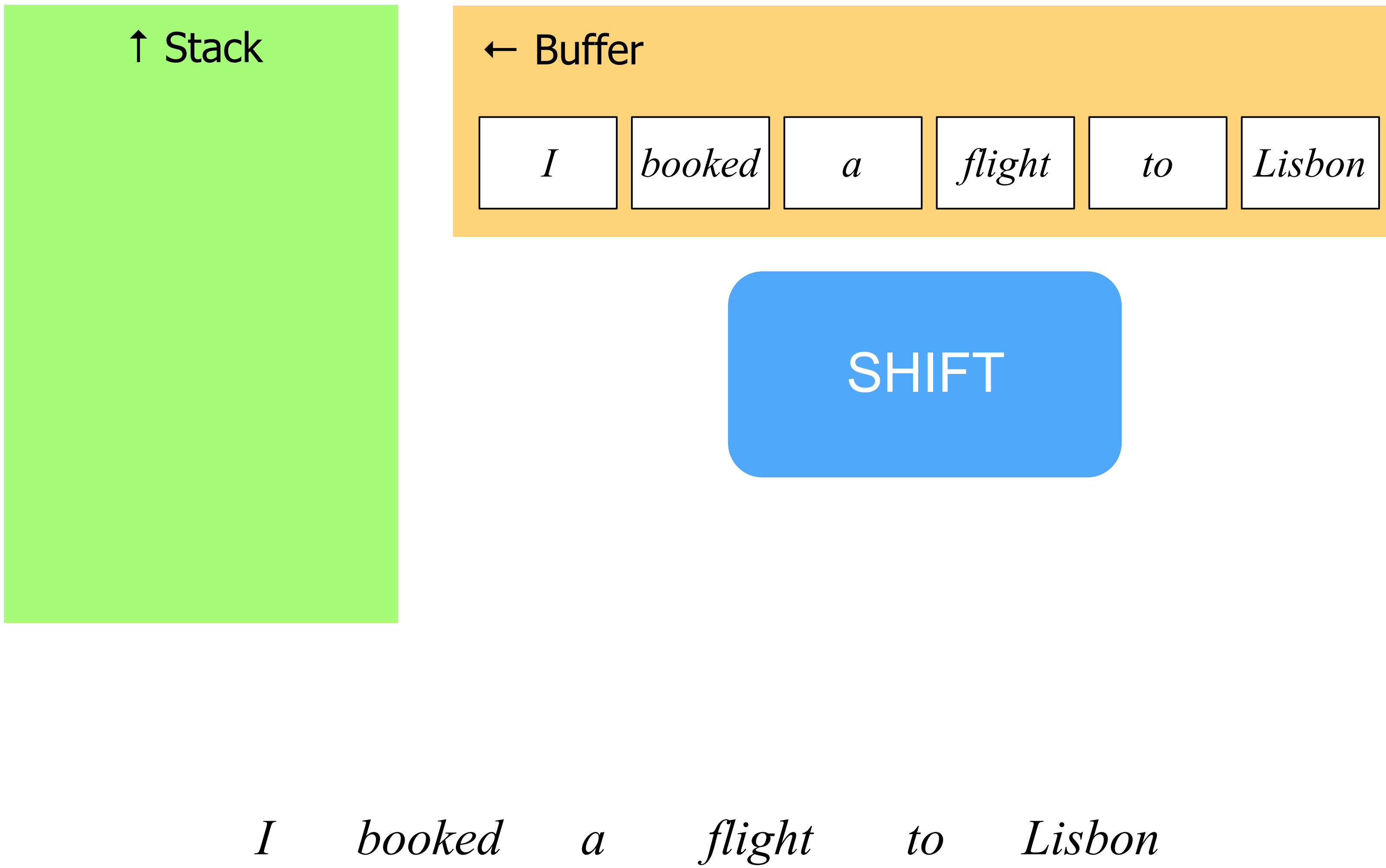
Initial configuration: ([],[0,…,n],[])
Terminal configuration: ([0],[],A)

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

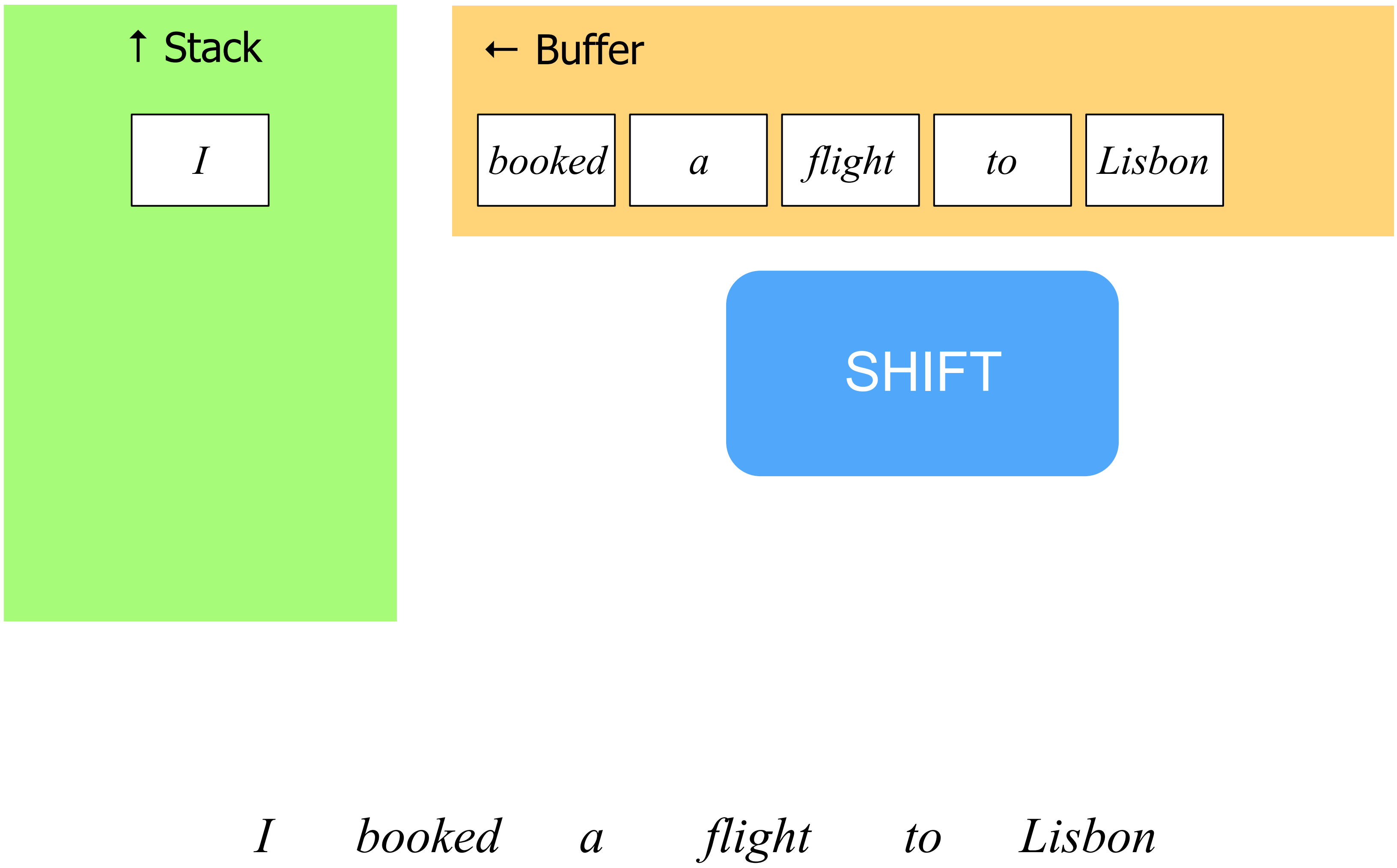
left-arc (label): $([\sigma|i|j],B,A) \Rightarrow ([\sigma|j],B,A\cup\{j,l,i\})$

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

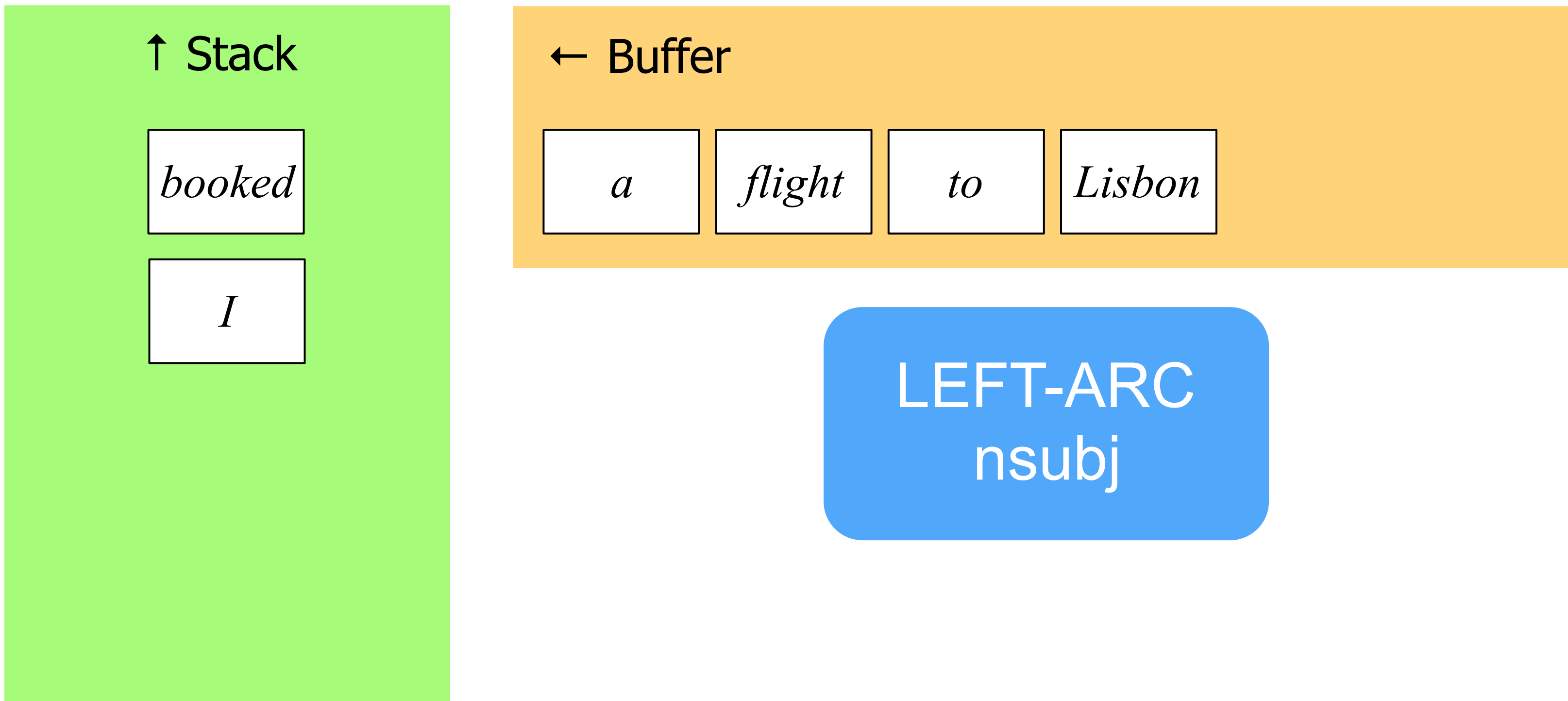
# Arc-Standard Example



# Arc-Standard Example

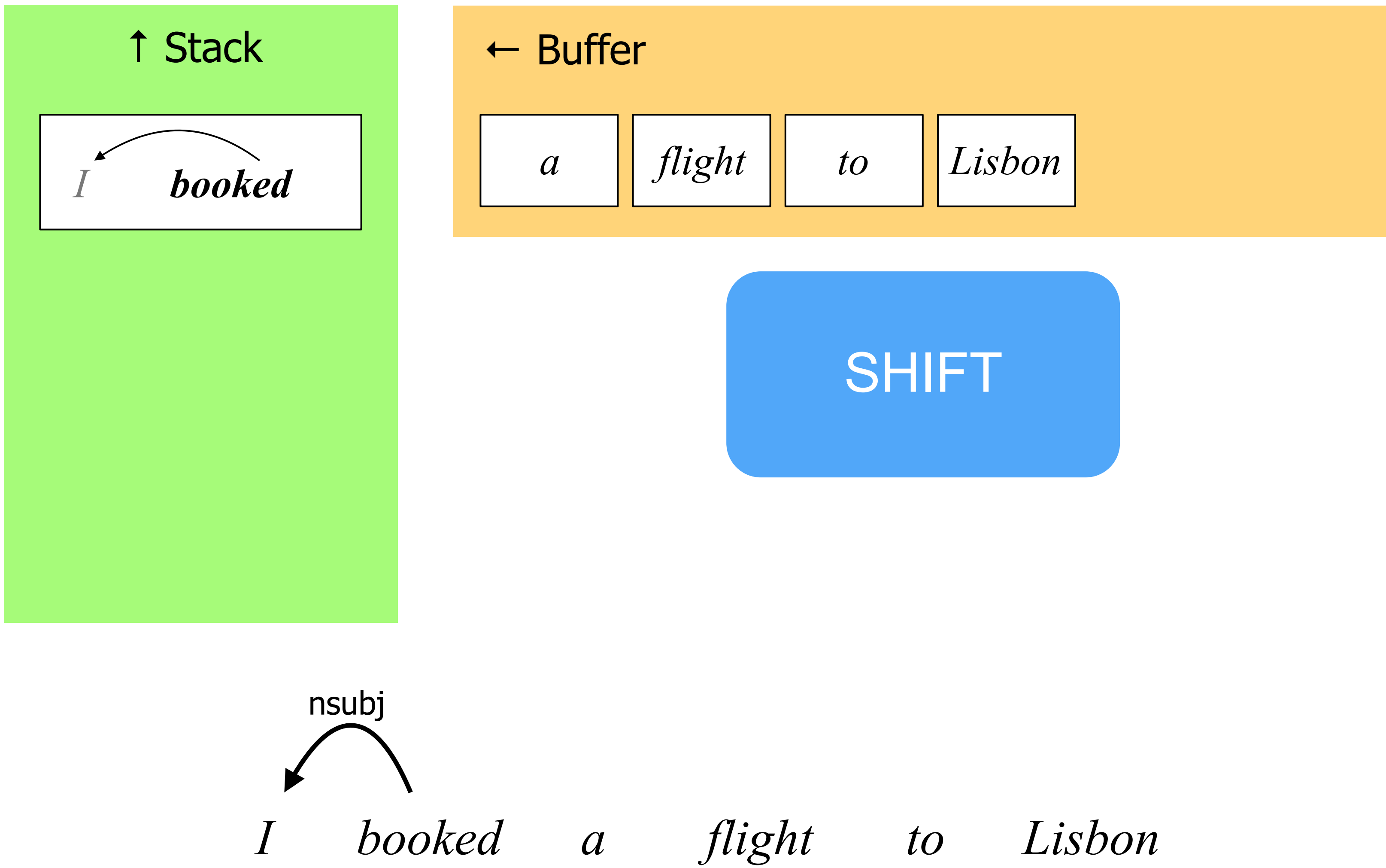


# Arc-Standard Example

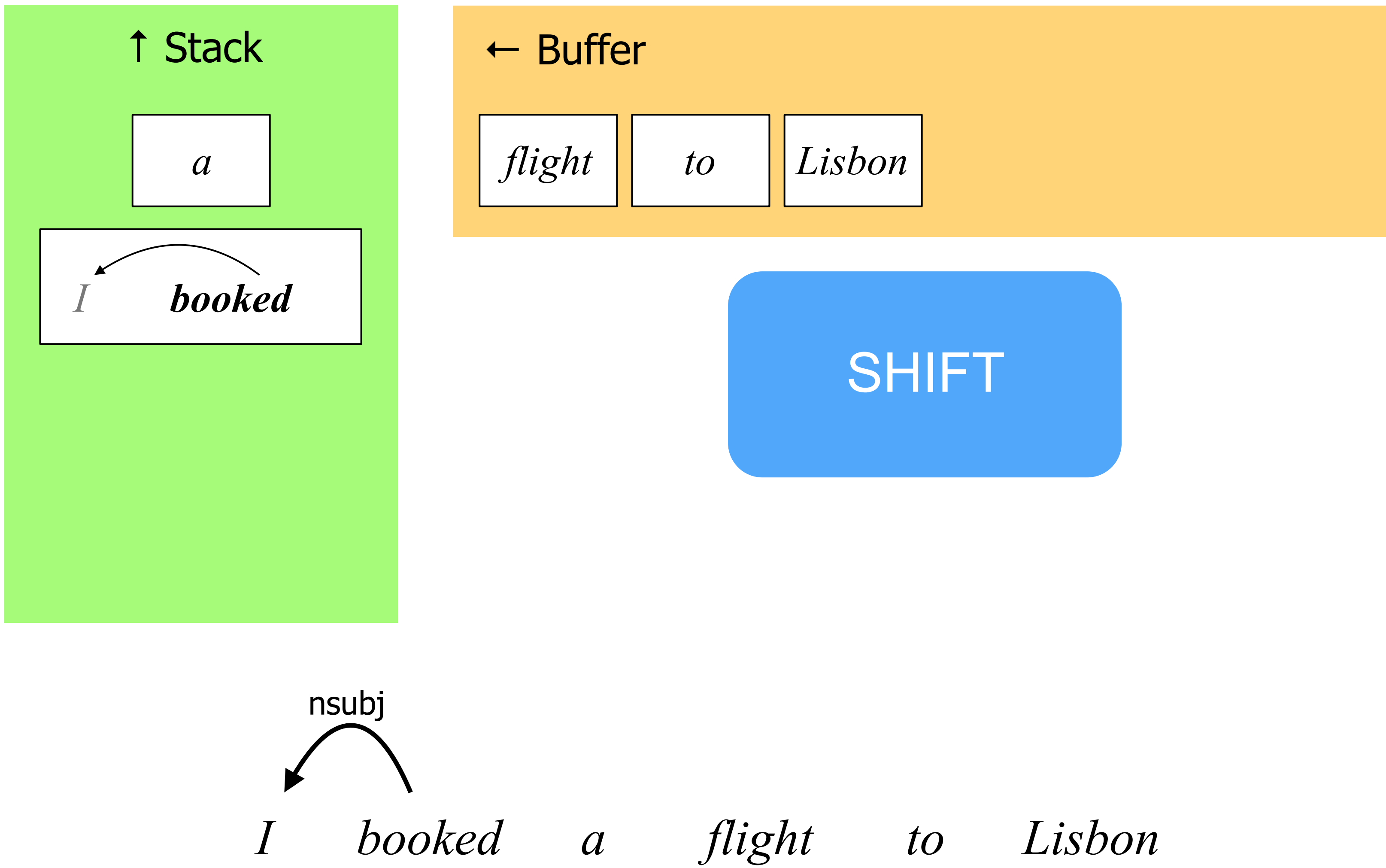



*I booked a flight to Lisbon*

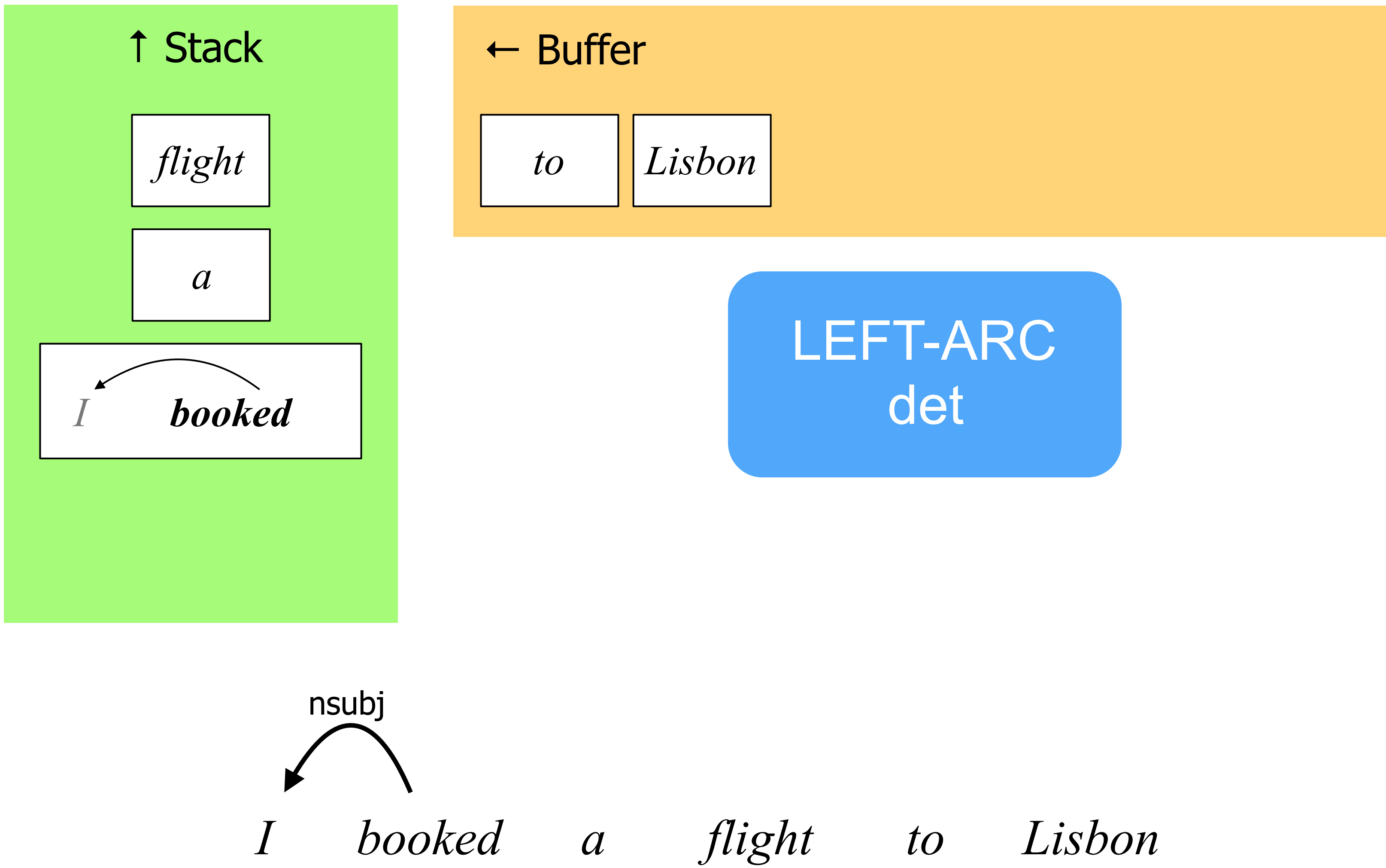
# Arc-Standard Example



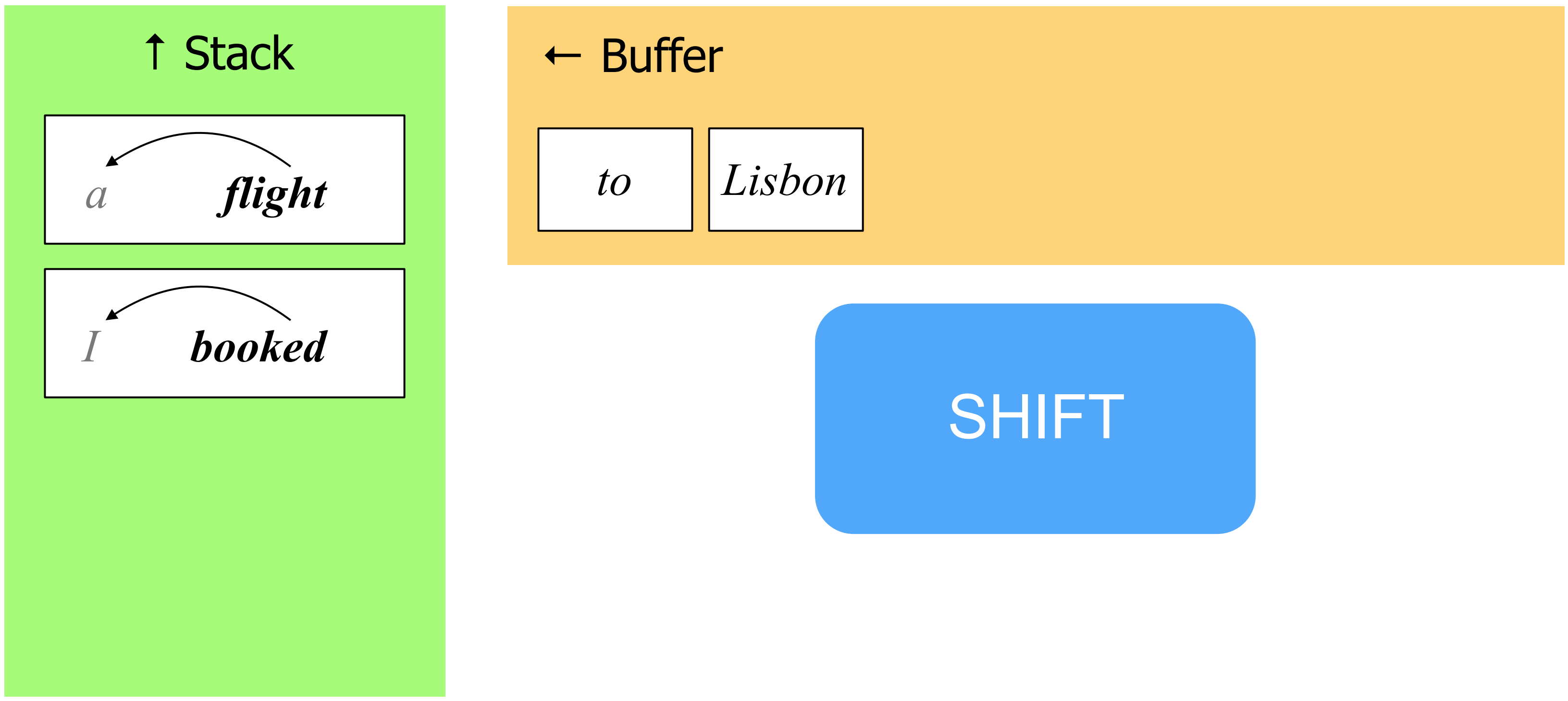
# Arc-Standard Example

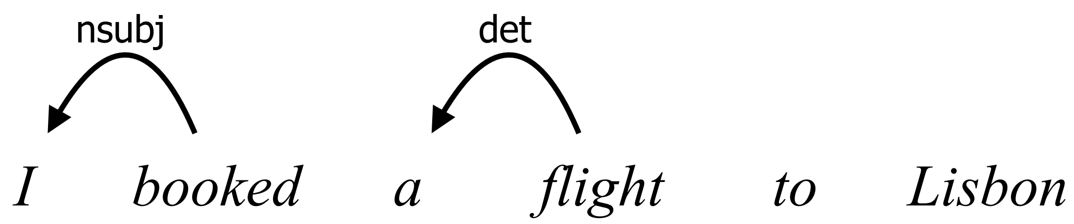


# Arc-Standard Example

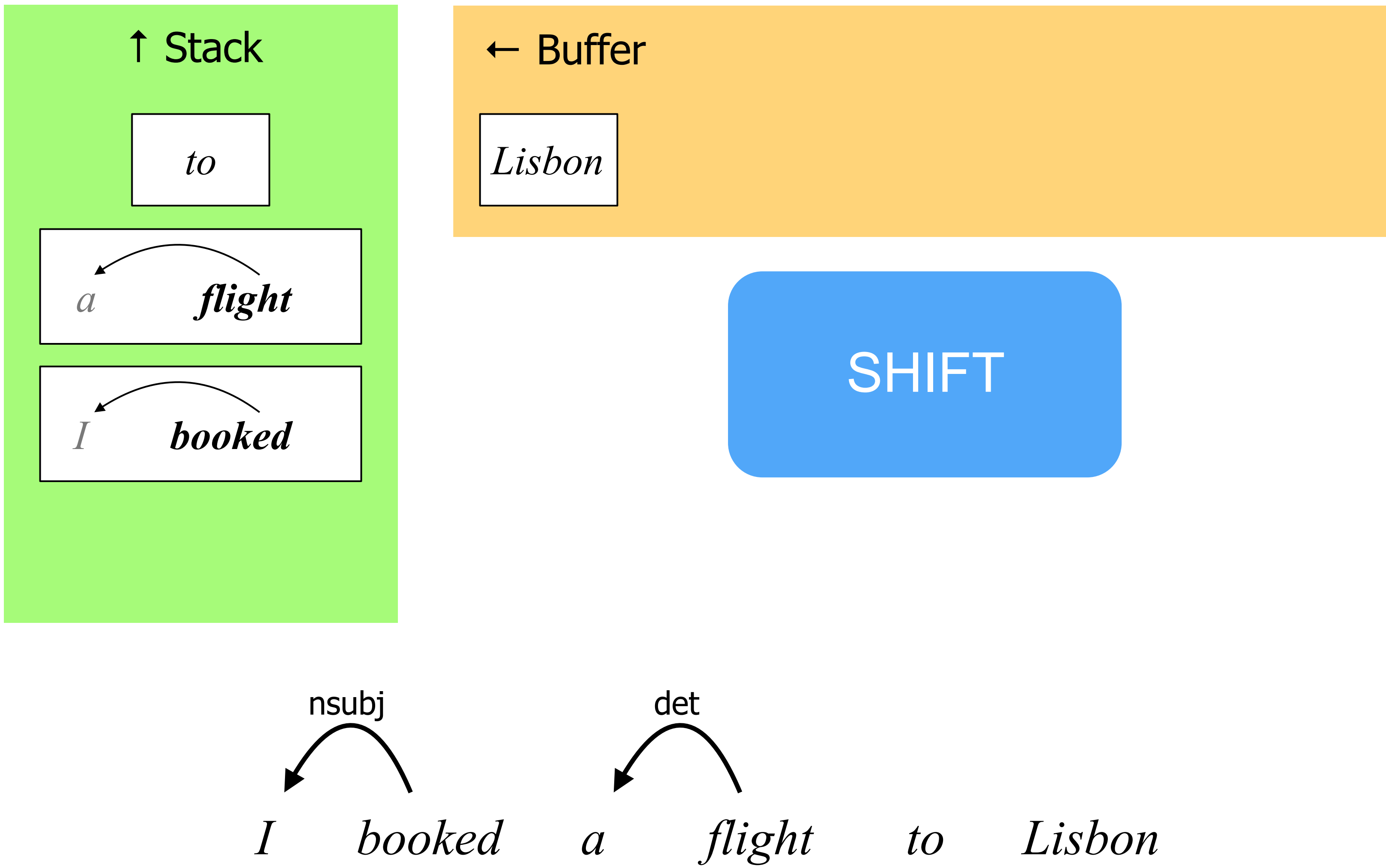


# Arc-Standard Example

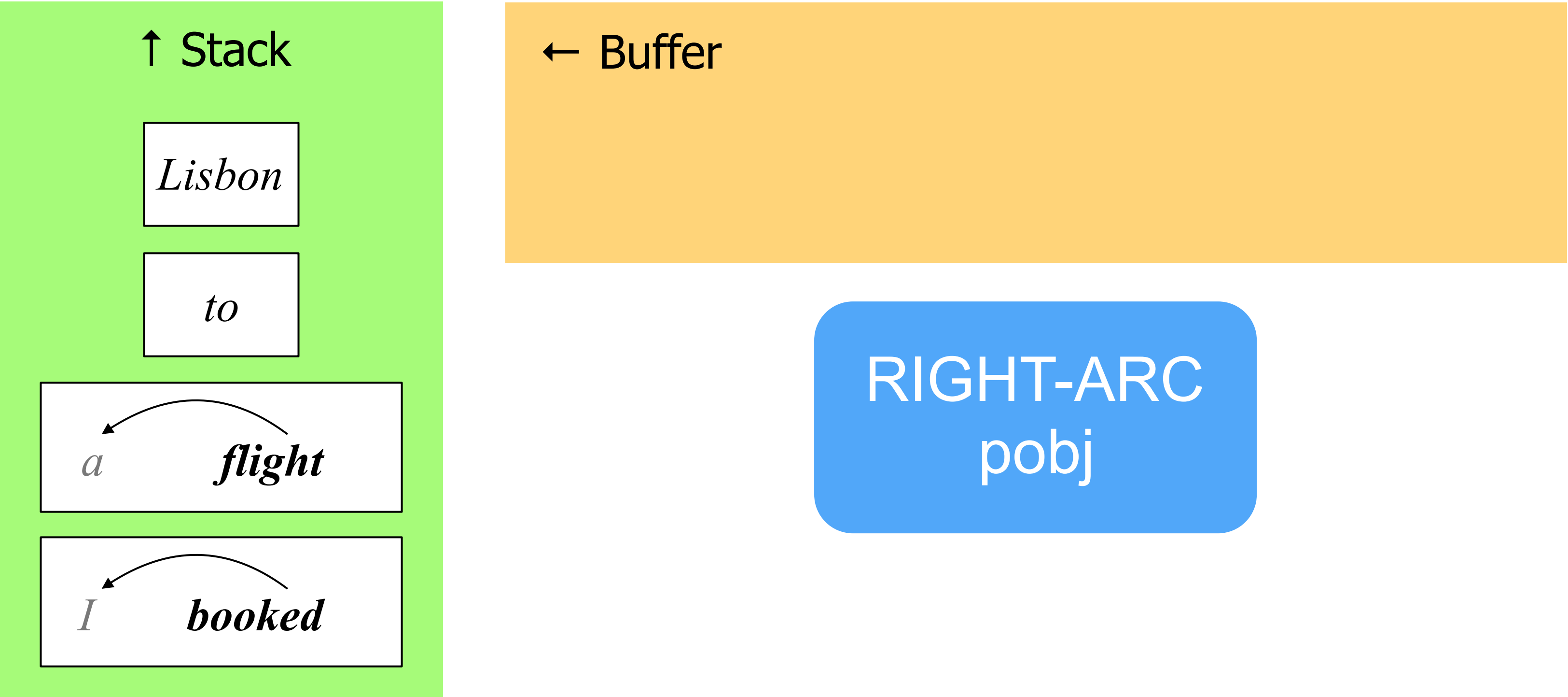


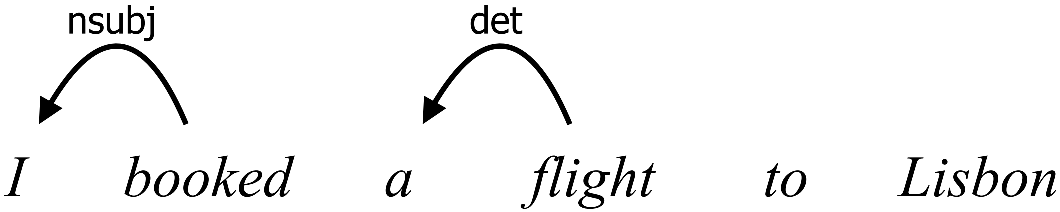



# Arc-Standard Example

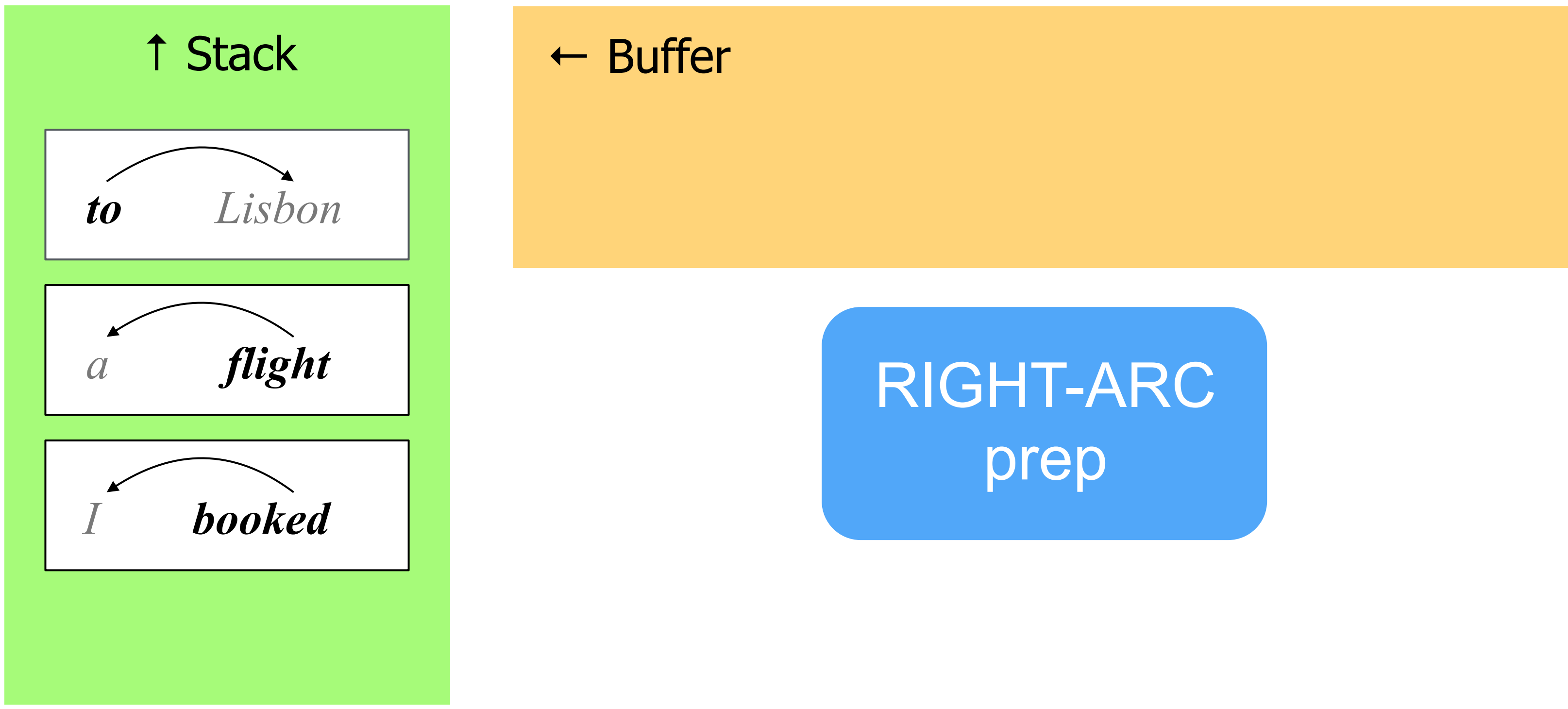


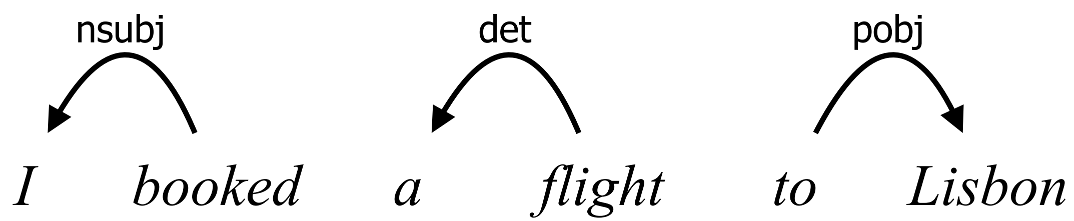
# Arc-Standard Example



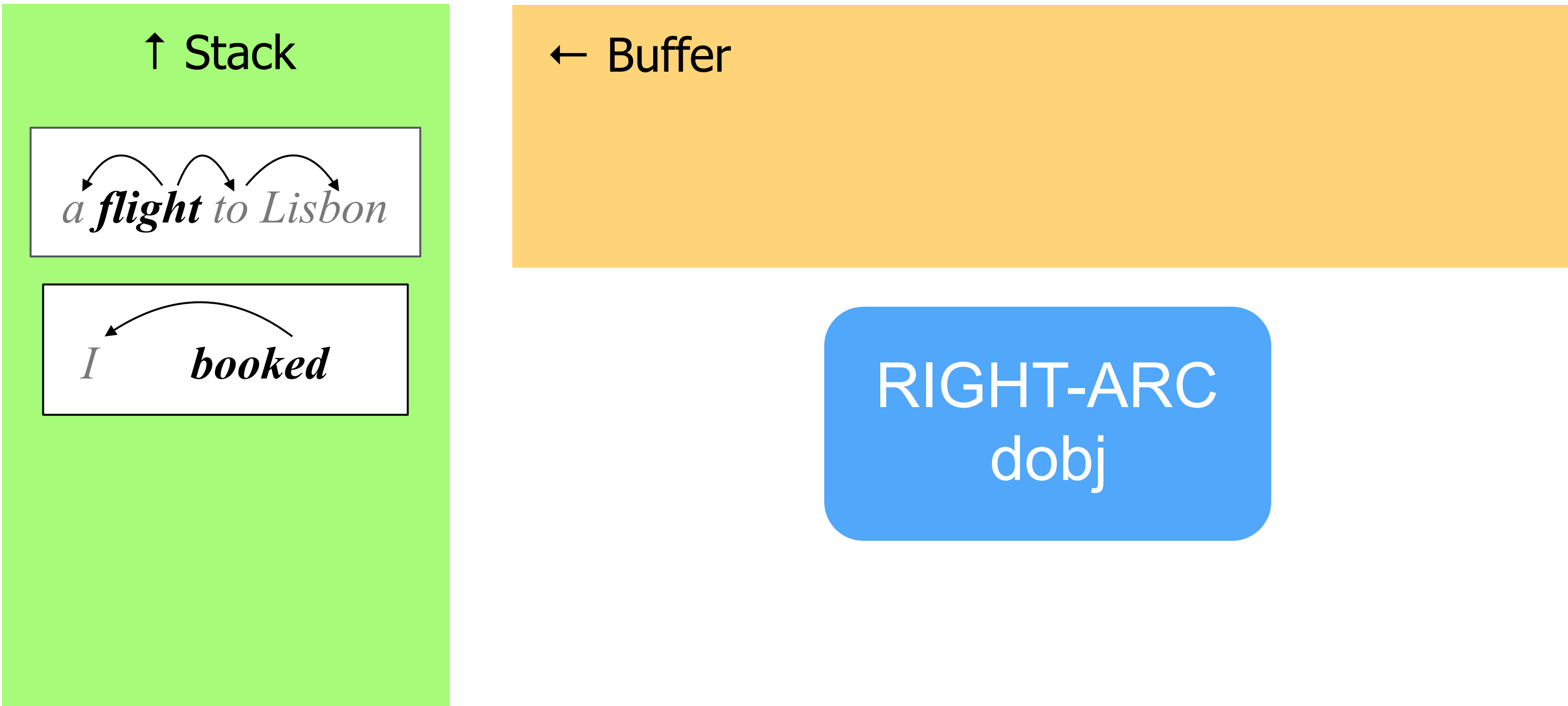



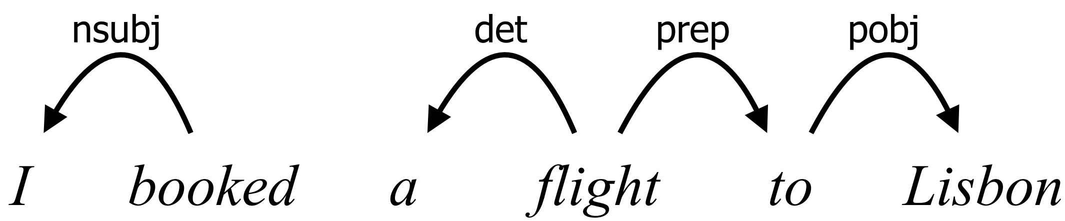
# Arc-Standard Example



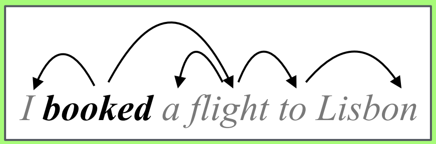
# Arc-Standard Example



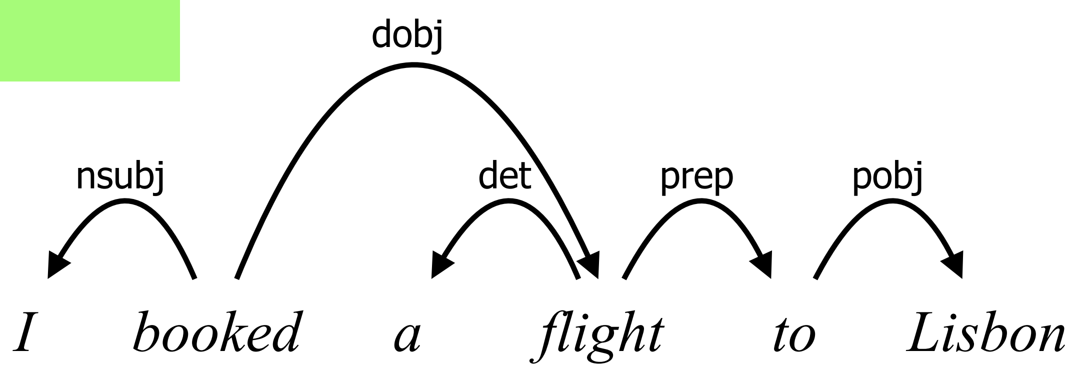



# Arc-Standard Example

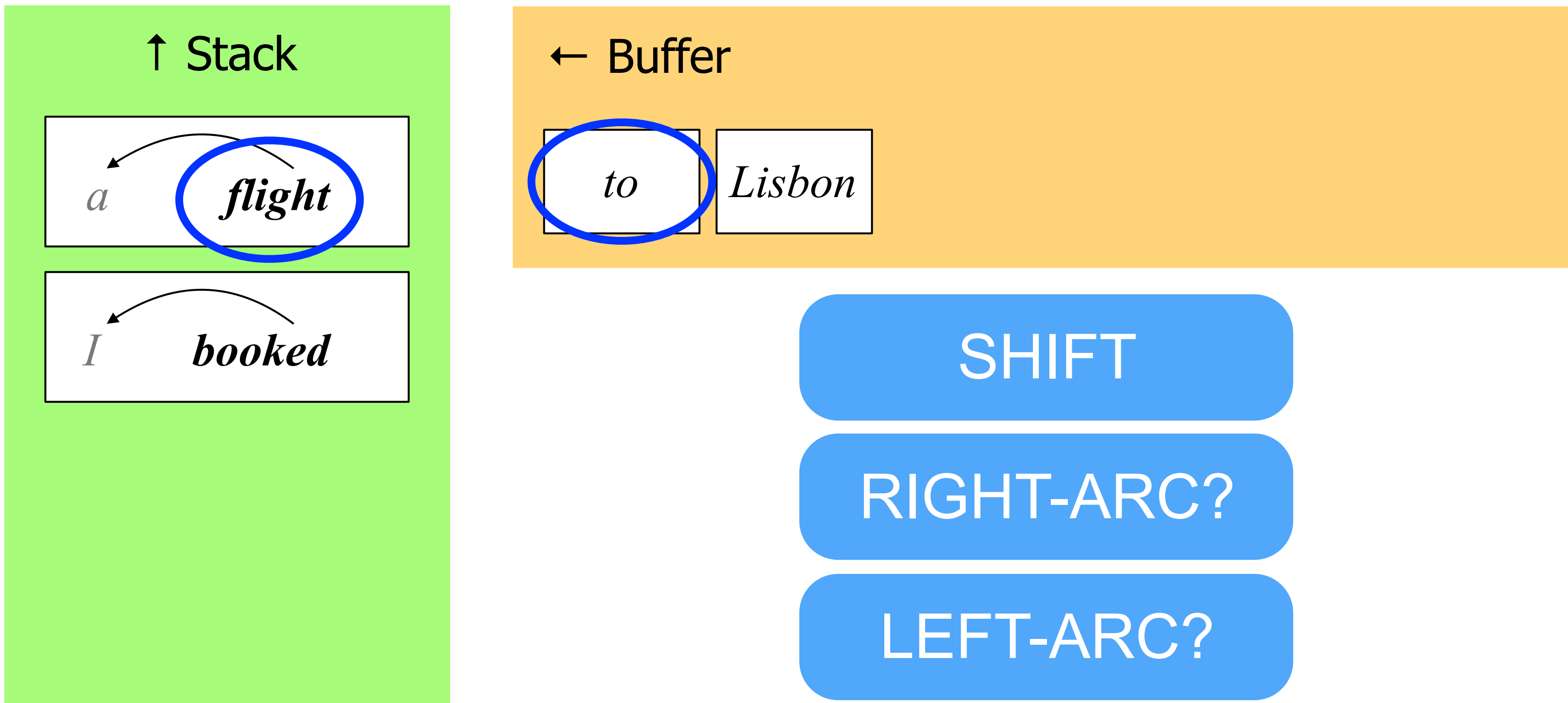
↑ Stack




← Buffer



# Features




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

# SVM / Structured Perceptron Hyperparameters

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

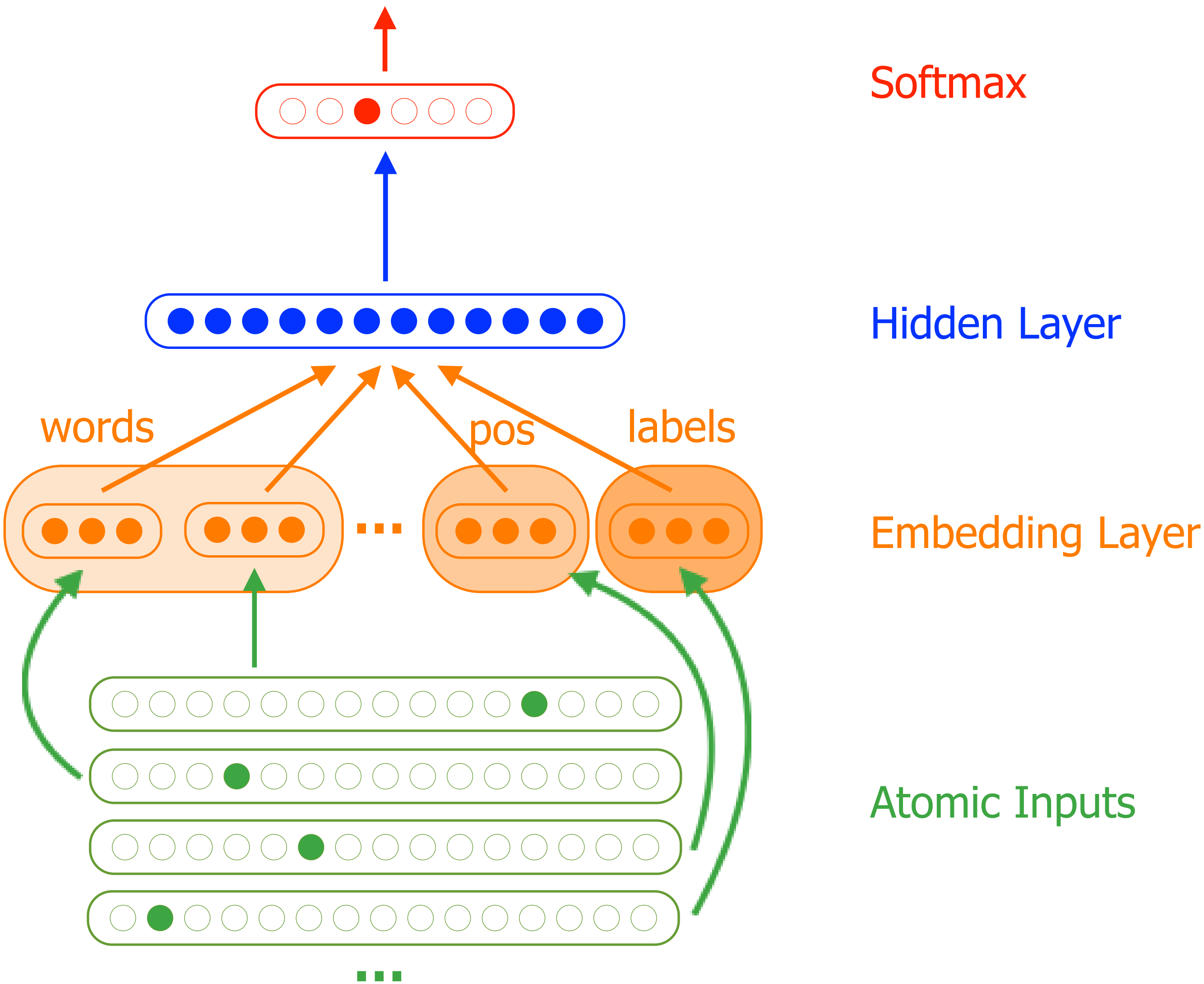
# Features ZPar Parser

```
# 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).lab
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).labe
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).lab
quad { input.tag input.child(-1).label input.child(-1).sibling(1).label
```

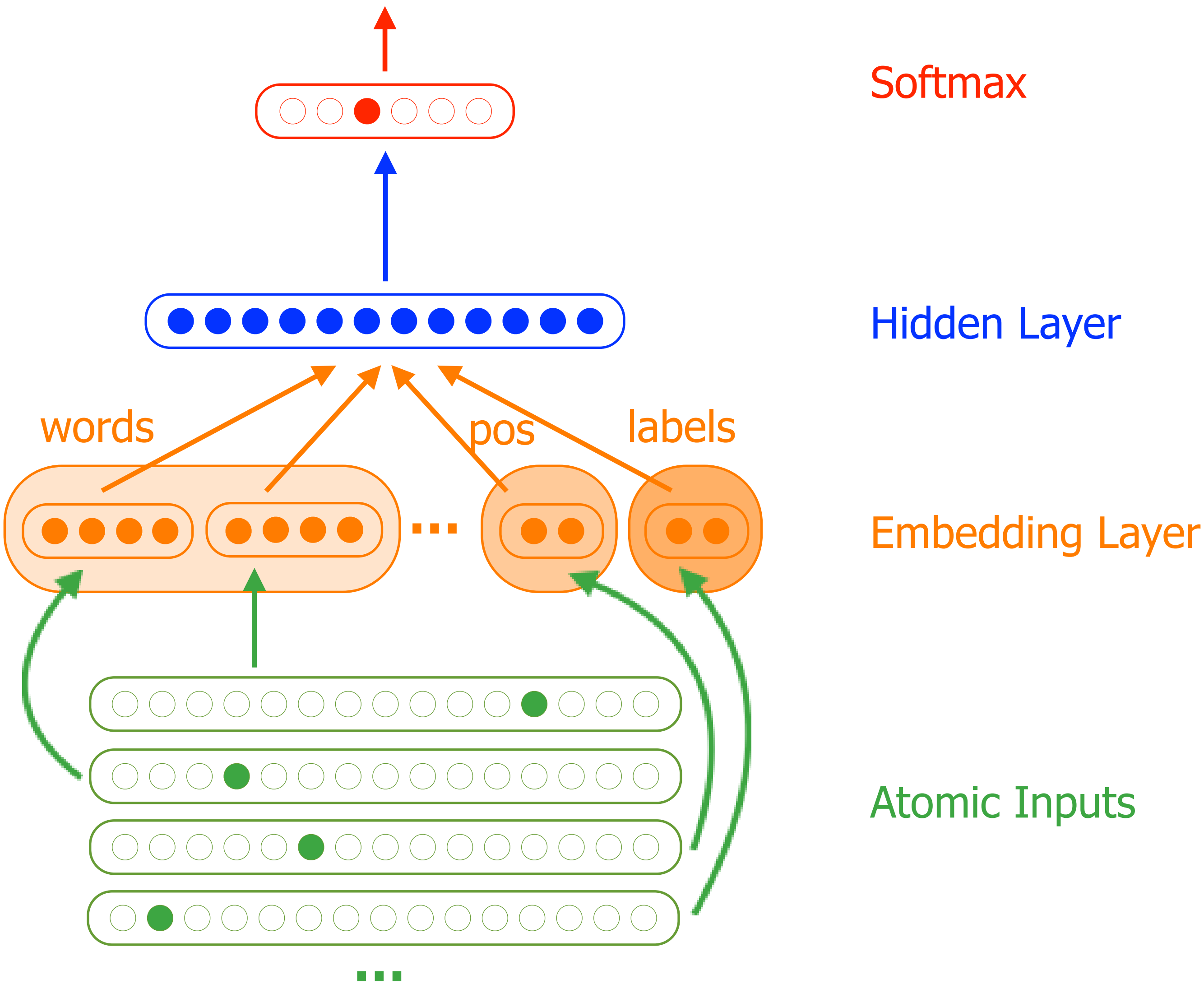
# Neural Network Transition Based Parser

[Chen & Manning '14] and [Weiss et al. '15, Andor et al. '16]



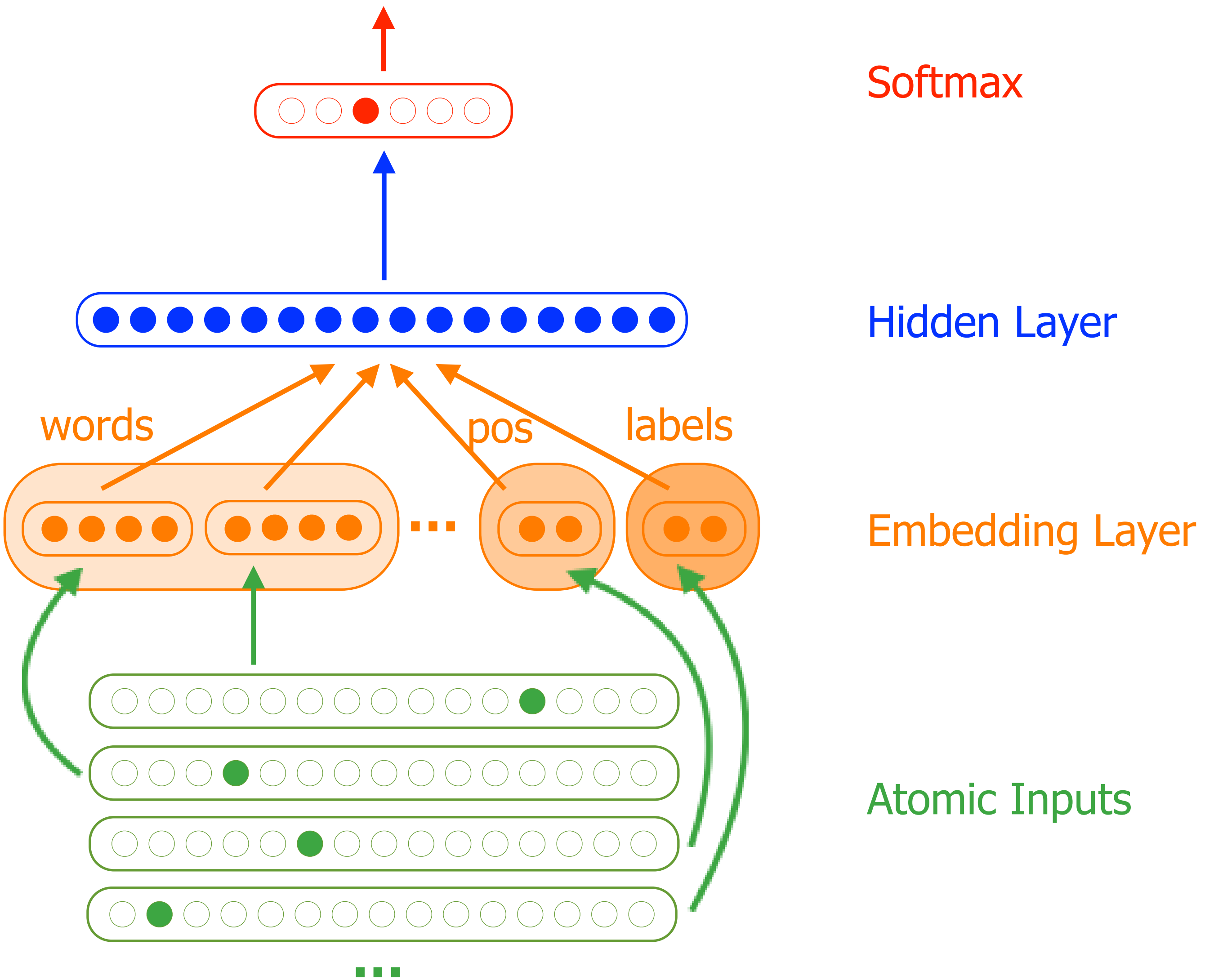
# Neural Network Transition Based Parser

[Weiss et al. ’15]



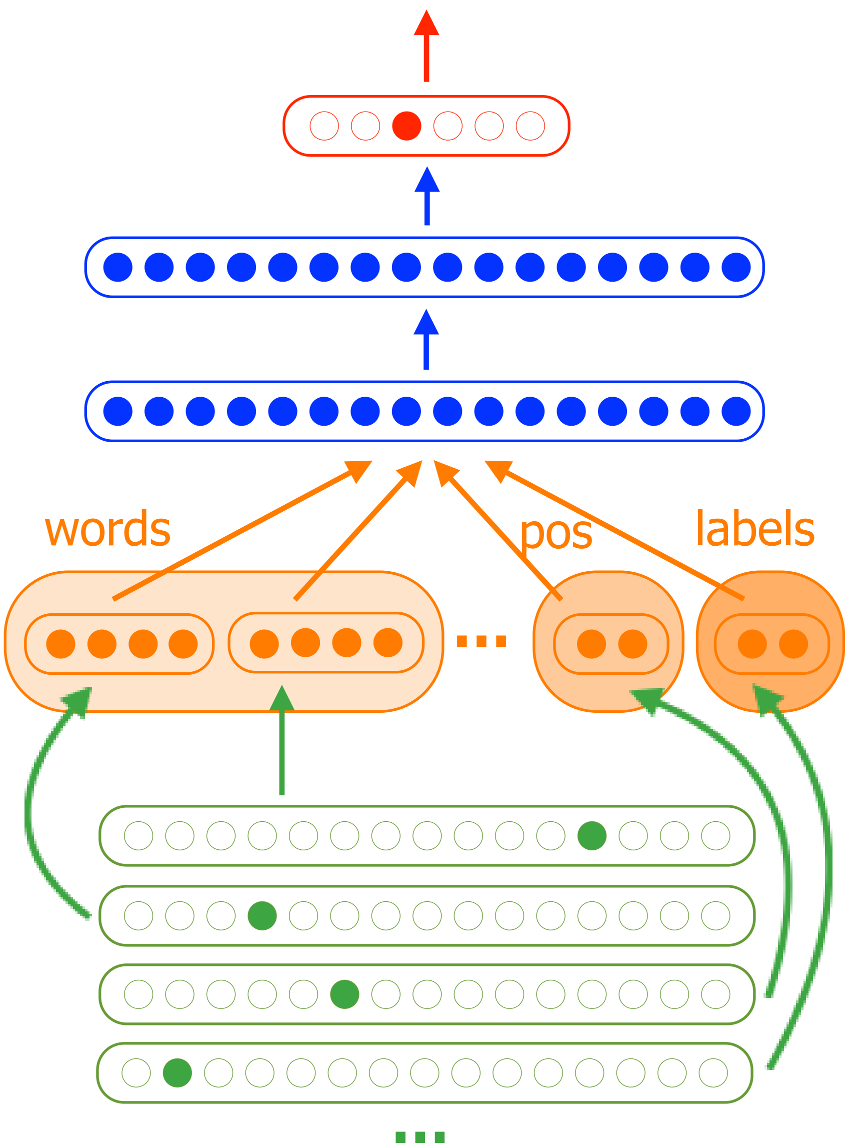
# Neural Network Transition Based Parser

[Weiss et al. ’15]



# Neural Network Transition Based Parser

[Weiss et al. ’15]




Softmax

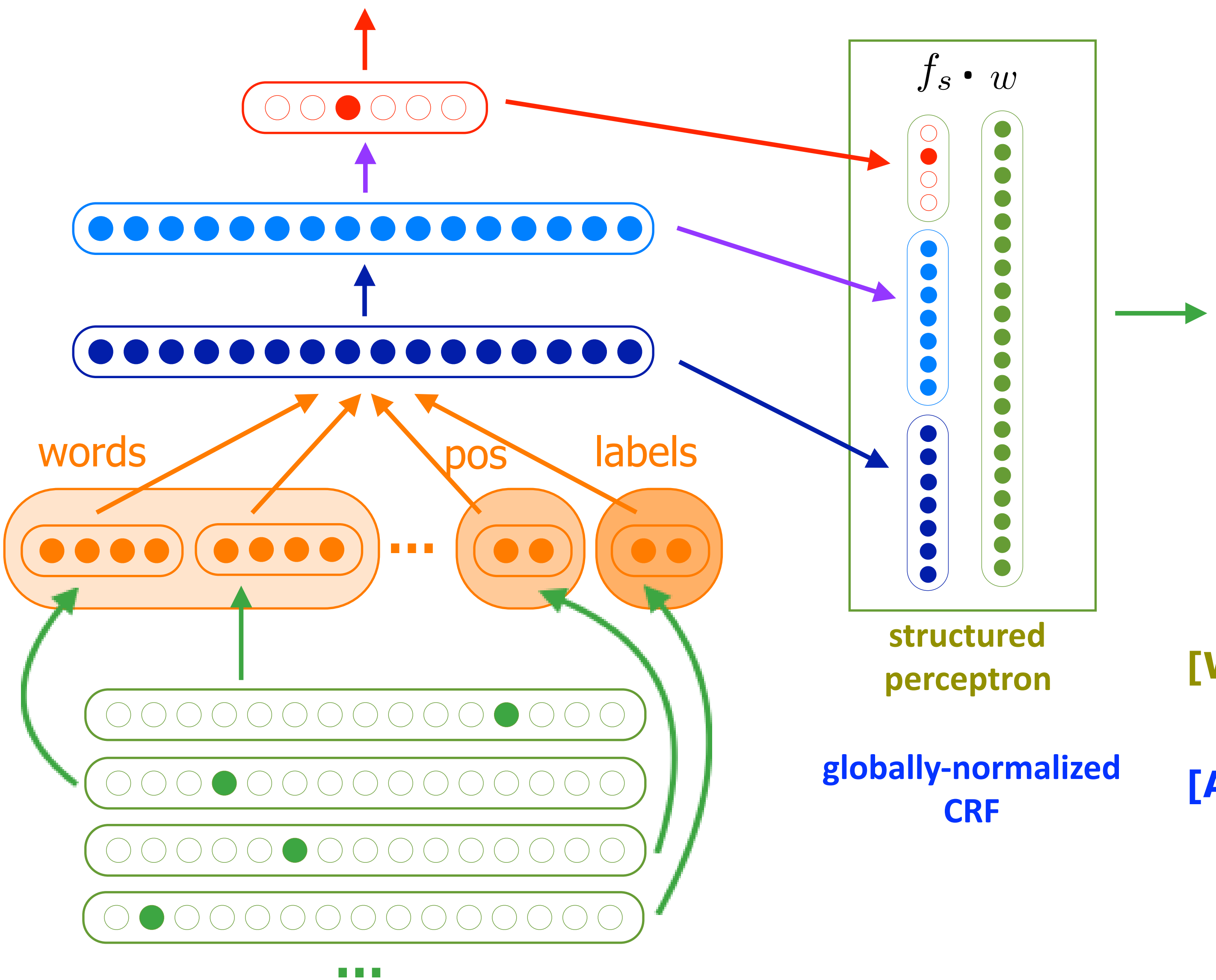
Hidden Layer 2

Hidden Layer 1

Embedding Layer

Atomic Inputs

# Neural Network Transition Based Parser




[Weiss et al. '15]

[Andor et al. '16]

# NN Hyperparameters

- Regularization
- Loss function

# NN Hyperparameters

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

# NN Hyperparameters

- Regularization
- Loss function
- Dimensions
- Activation function

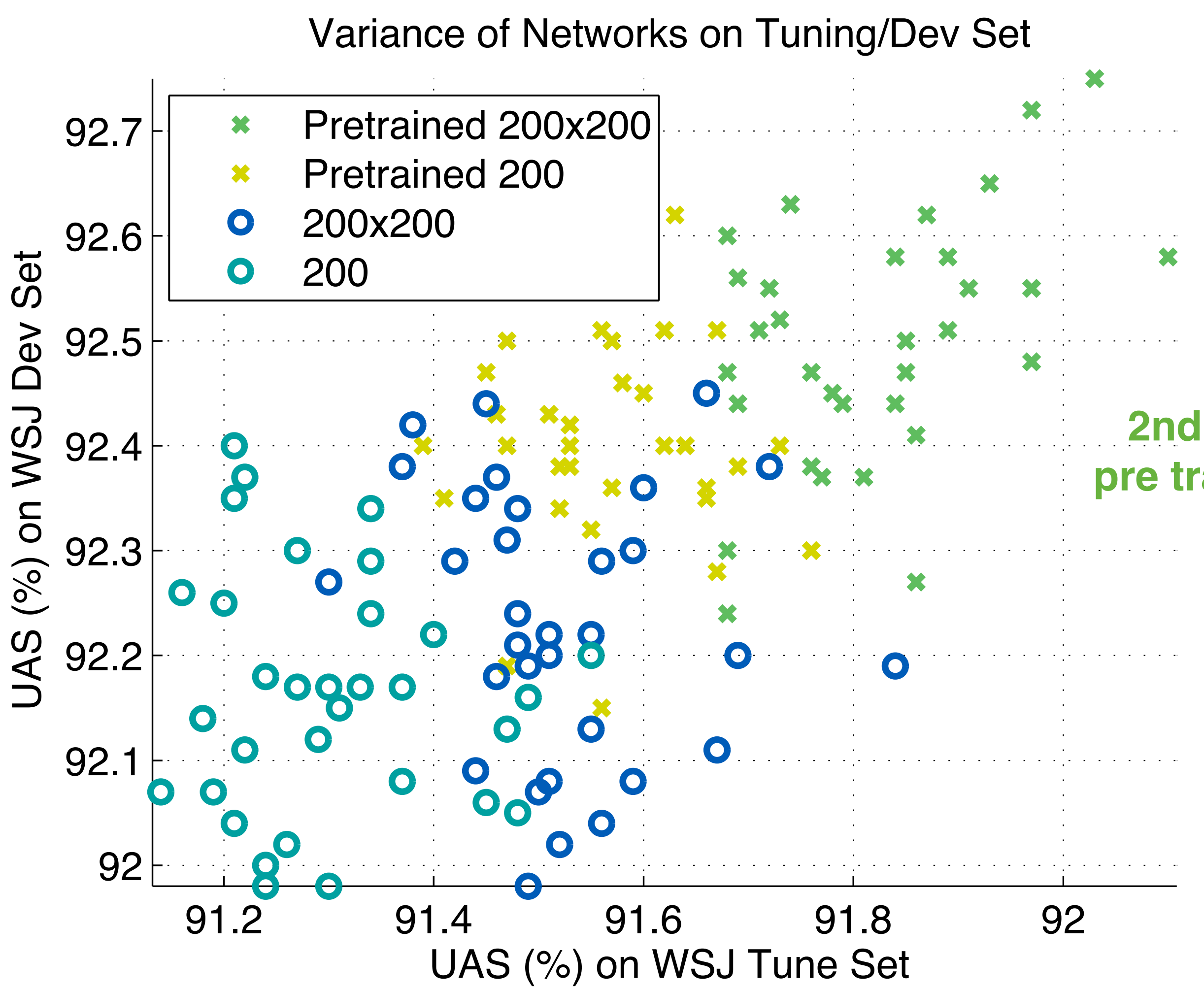
- Initialization
- Adagrad
- Dropout
- Mini-batch size
- Initial learning rate
- Learning rate schedule
- Momentum
- Stopping time
- Parameter averaging

# NN Hyperparameters

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

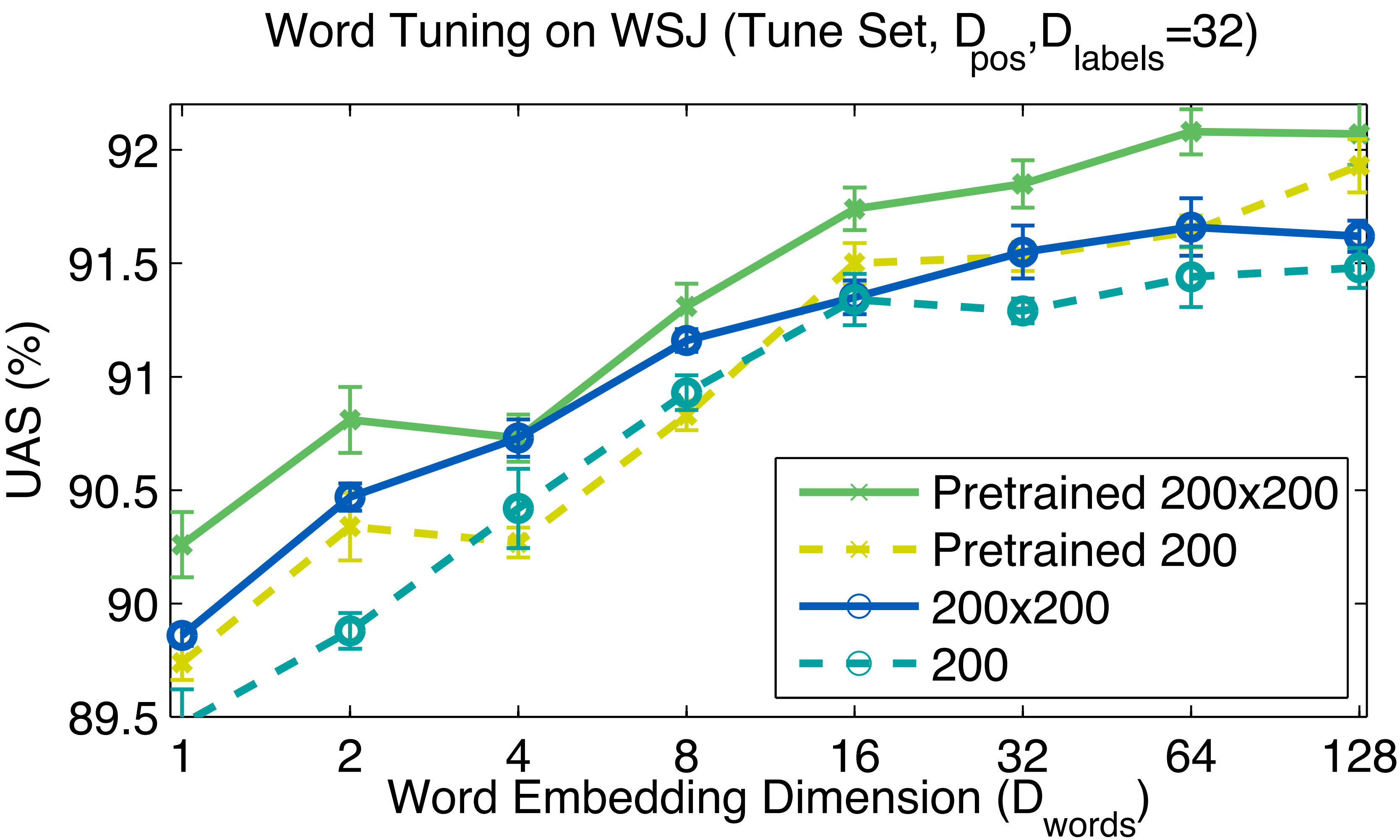
*Tune: WSJ S24*
*Dev: WSJ S22*
*Test: WSJ S23*

# Random Restarts: How much Variance?

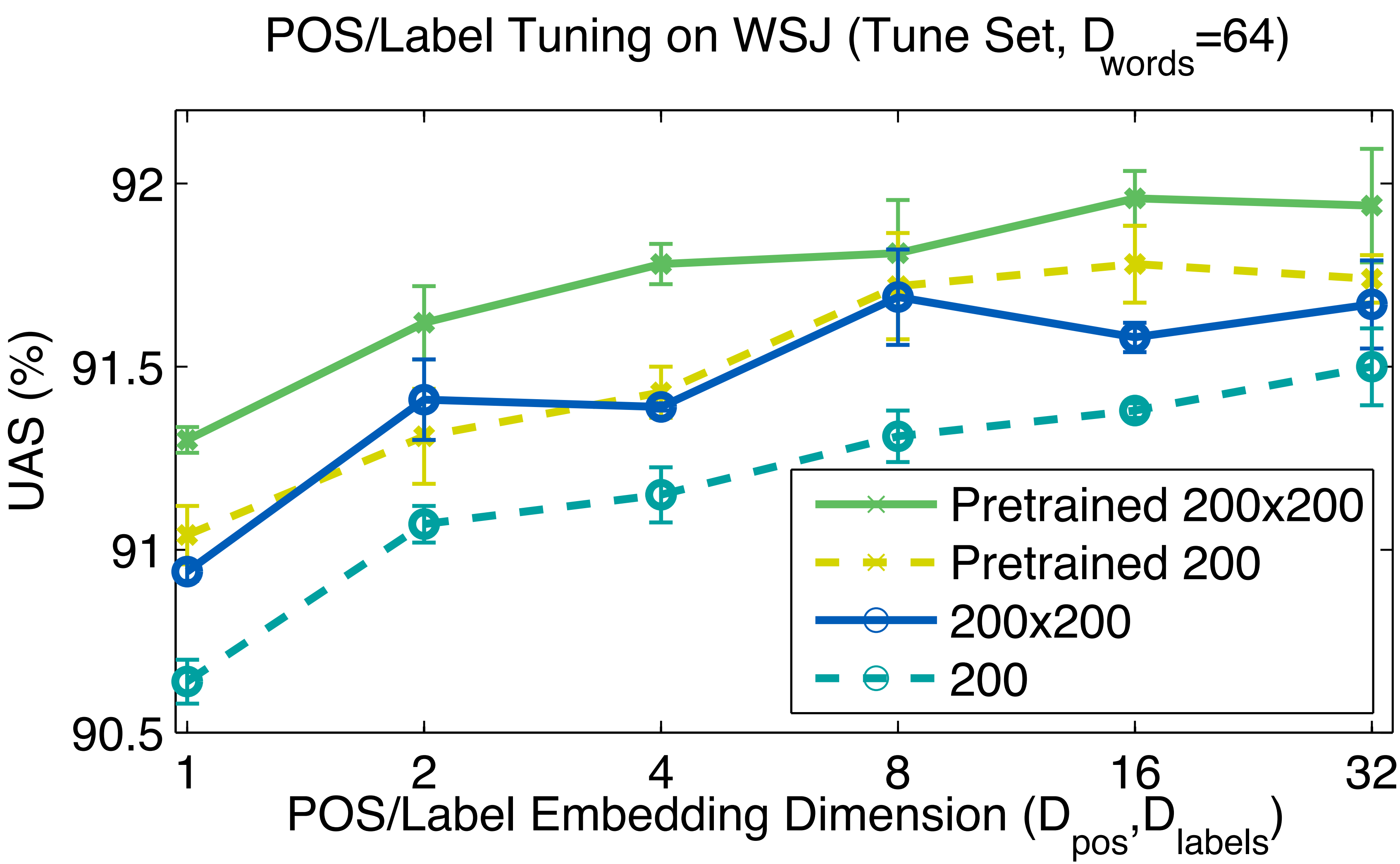



**2nd hidden layer + pre training increases correlation**

# Effect of Embedding Dimensions

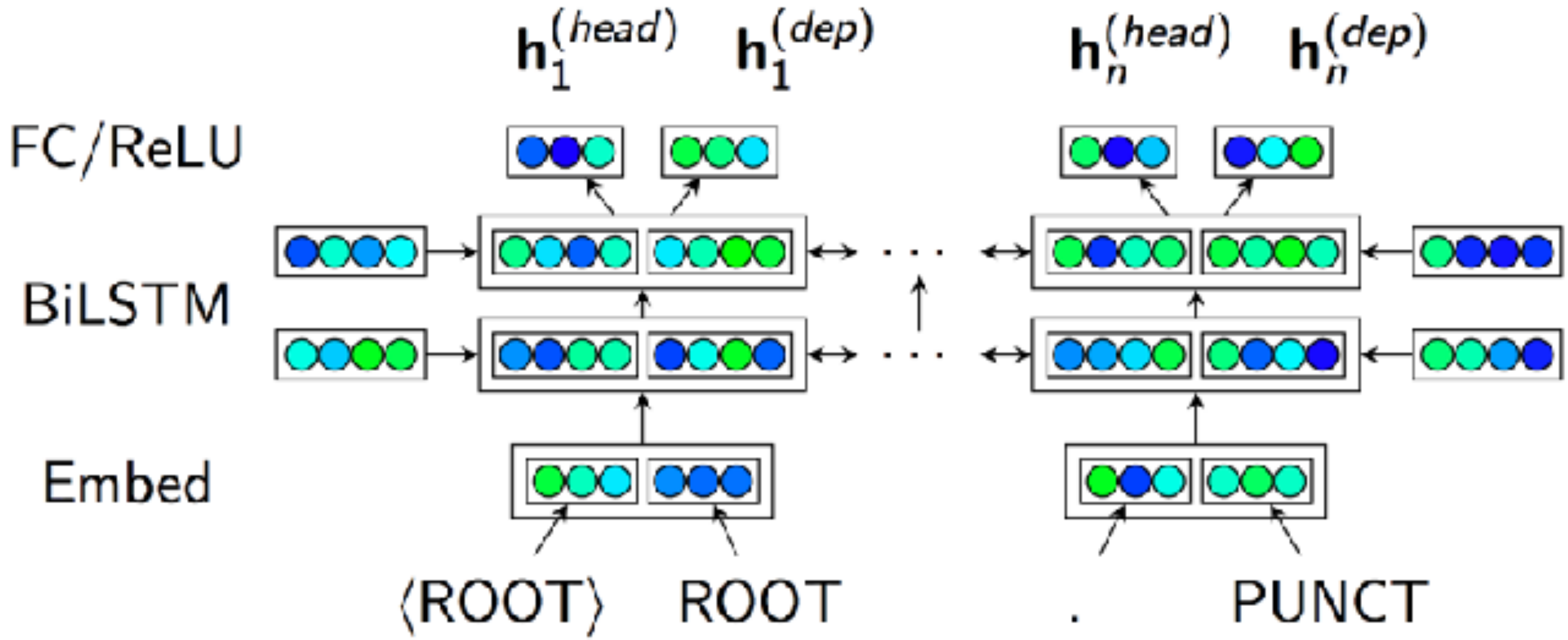


# Effect of Embedding Dimensions



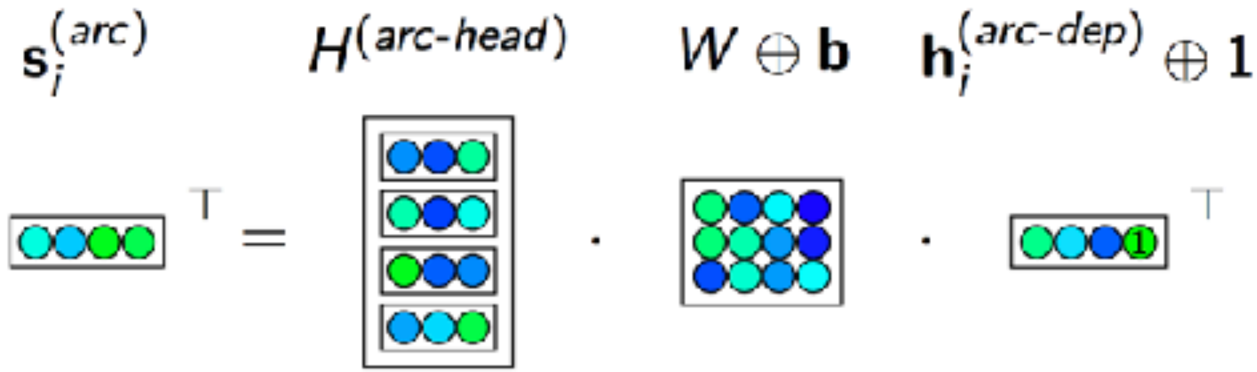
# Do we need structure?

[Dozat & Manning ’17]



# Bi-Affine Parsing

- Biaffine self-attention layer to score all possible heads for each dependent $i$

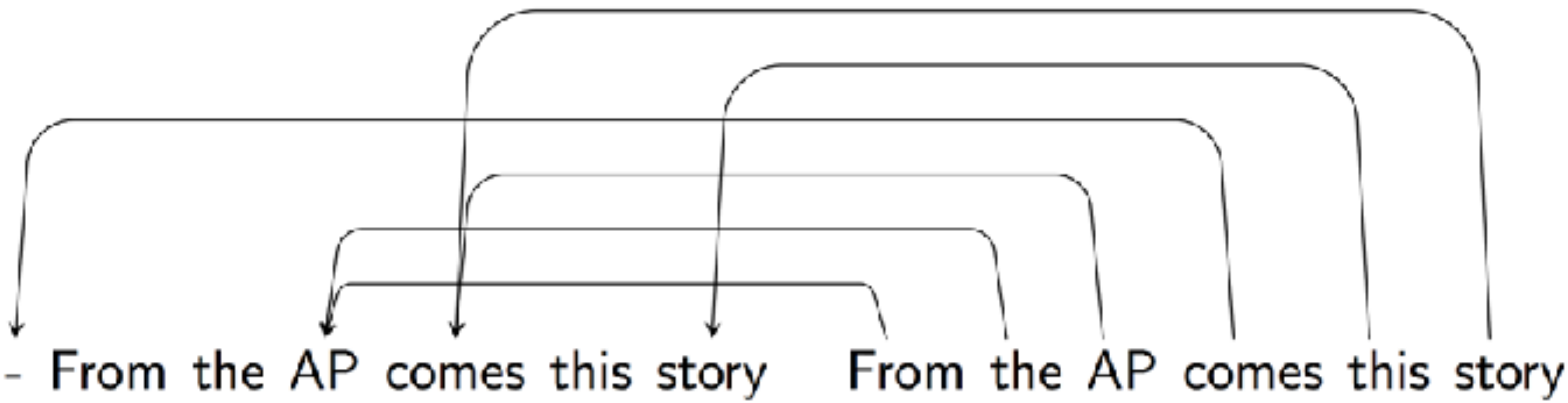



- Train with cross-entropy
- Apply a spanning tree algorithm at inference time

Note: This is just an affine layer with a linear transformation!

$$\mathbf{s}_i = H^{(arc\text{-}head)}(W\mathbf{h}_i^{(arc\text{-}dep)} + \mathbf{b})$$

# Self-Attention




**Source** **Target**

# English Results (WSJ 23)

| Method | UAS | LAS | Beam |
|---|---|---|---|
| 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 Semi-supervised (Weiss et al., 2015) | 94,26 | 92,41 | 8 |
| S-LSTM (Dyer et al., 2015) | 93,20 | 90,90 | 1 |
| Contrastive NN (Zhou et al., 2015) | 92,83 | — | 100 |

# English Results (WSJ 23)

| | | PTB | |
|---|---|---|---|
| **Type** | **Model** | **UAS** | **LAS** |
| Transition | Ballesteros et al. (2016) | 93.56 | 91.42 |
| | Andor et al. (2016) | 94.61 | 92.79 |
| | Kuncoro et al. (2016) | **95.8** | **94.6** |
| Graph | K&G (2016) | 93.9 | 91.9 |
| | Cheng et al. (2016) | 94.10 | 91.49 |
| | Hashimoto et al. (2016) | 94.67 | 92.90 |
| | D&M (2017) | 95.74 | 94.08 |

# Multilingual Results

| **Treebanks** | **UPOS** | **XPOS** | **UAS** | **LAS** | **CLAS** |
|---|---|---|---|---|---|
| All treebanks | **93.09** | **82.27** | **81.30** | **76.30** | **72.57** |
| Large treebanks | **95.58** | **94.56** | **85.16** | **81.77** | **78.40** |
| Parallell treebanks | **88.25** | 30.66 | **80.17** | **73.73** | **69.88** |
| Small treebanks | **87.02** | **82.03** | 70.19 | 61.02 | 54.76 |
| Surprise treebanks | – | – | 54.47 | 40.57 | 37.41 |

| **System** | **UPOS** | **XPOS** | **UAS** | **LAS** | **CLAS** |
|---|---|---|---|---|---|
| Dozat et al. | **93.09** | **82.27** | **81.30** | **76.30** | **72.57** |
| Björkelund et al. | *91.98* | 64.84 | 79.90 | 74.42 | 70.18 |
| Yu et al. | 91.00 | *79.93* | 74.22 | 68.41 | 63.24 |
| Shi et al. | 90.88 | 79.80 | *80.35* | *75.00* | *70.91* |

# Beyond Syntax: Semantic Structures

Time Patient

On January 13, 2018, a false ballistic missile alert was issued via the Emergency Alert System and Commercial Mobile Alert System over television, radio, and cellphones in the U.S. state of Hawaii. LOC LOC

Agent Patient

The alert stated that there was an incoming ballistic missile threat to Hawaii, advised residents to seek shelter, and concluded "This is not a drill".

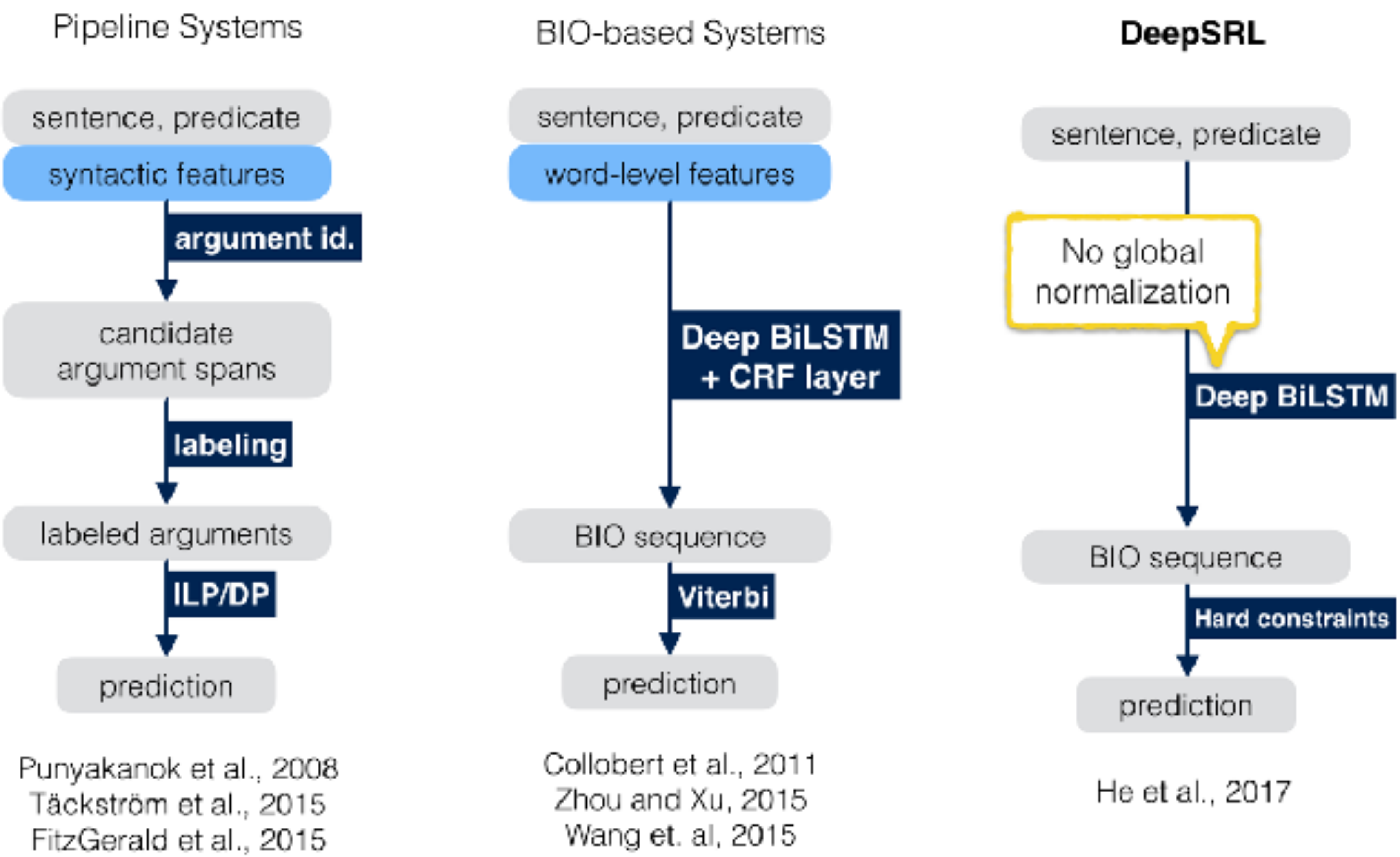
The message was sent at 8:07 a.m. local time.

Patient Time

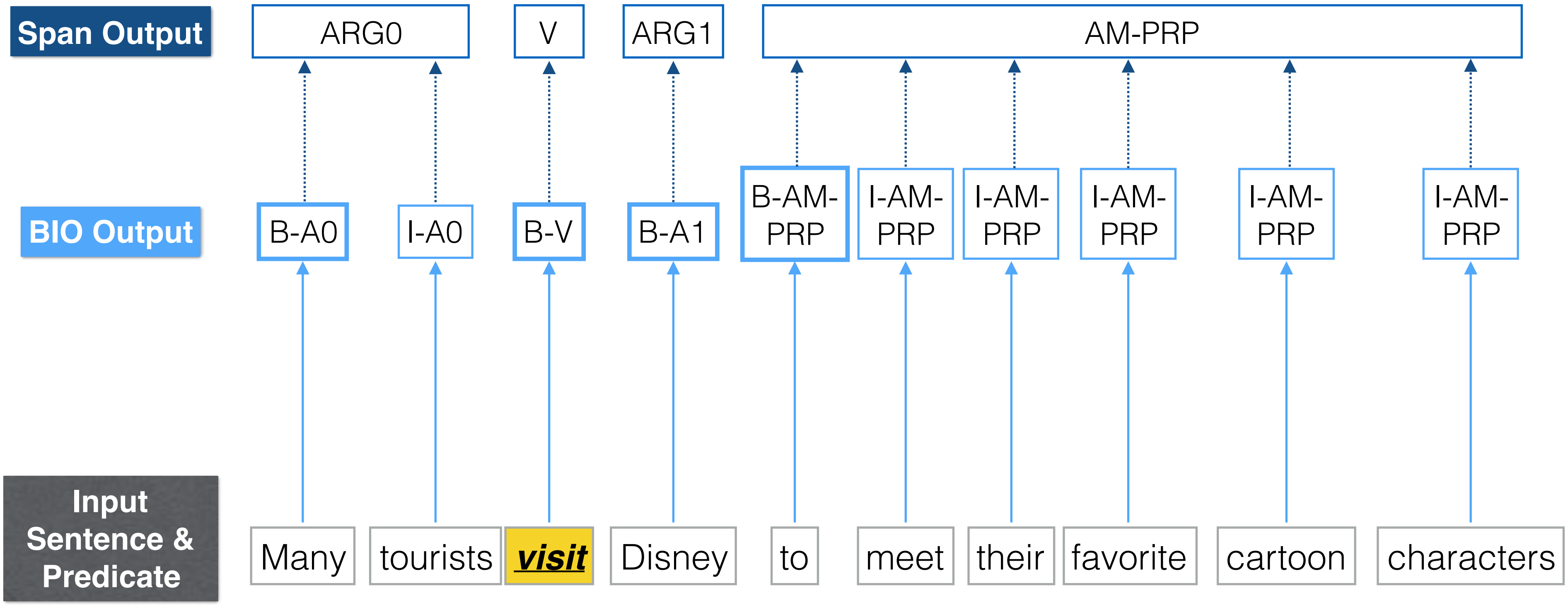
SRL
Coreference
NER

From Wikipedia: 2018 Hawaii false missile alert. Only part of the structures are visualized.

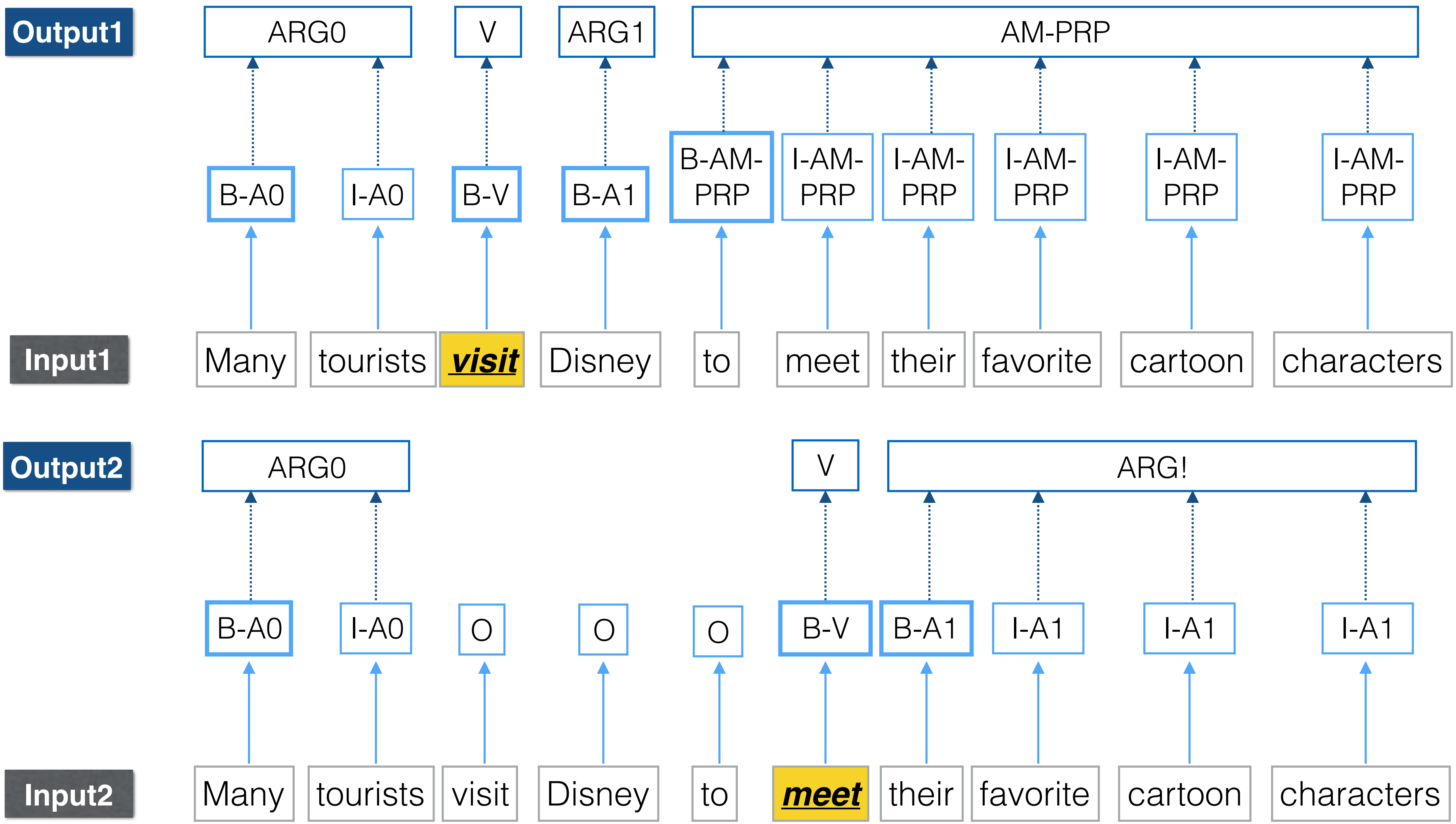
# SRL Systems: Pipelined vs. BIO-based



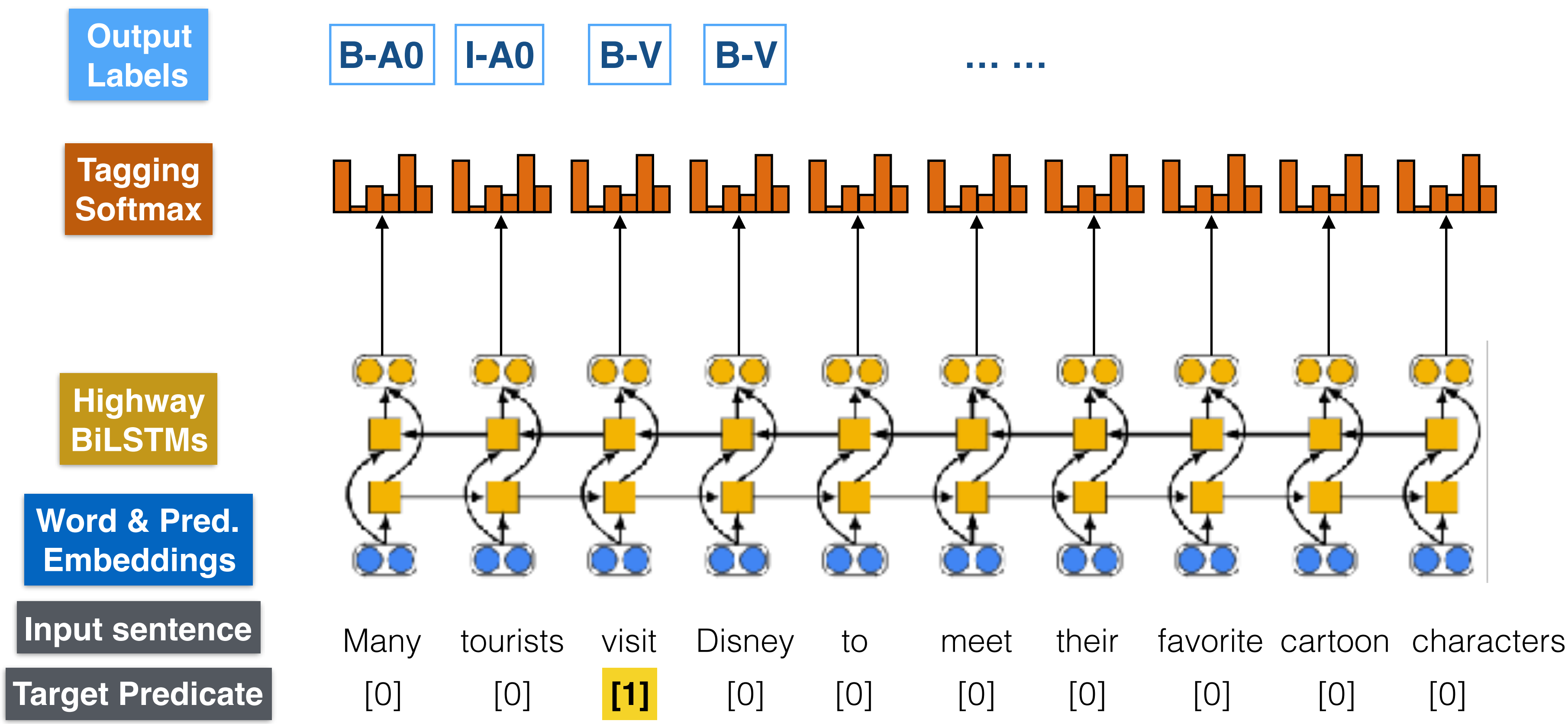
# SRL as a BIO Tagging Problem



# SRL as a BIO Tagging Problem

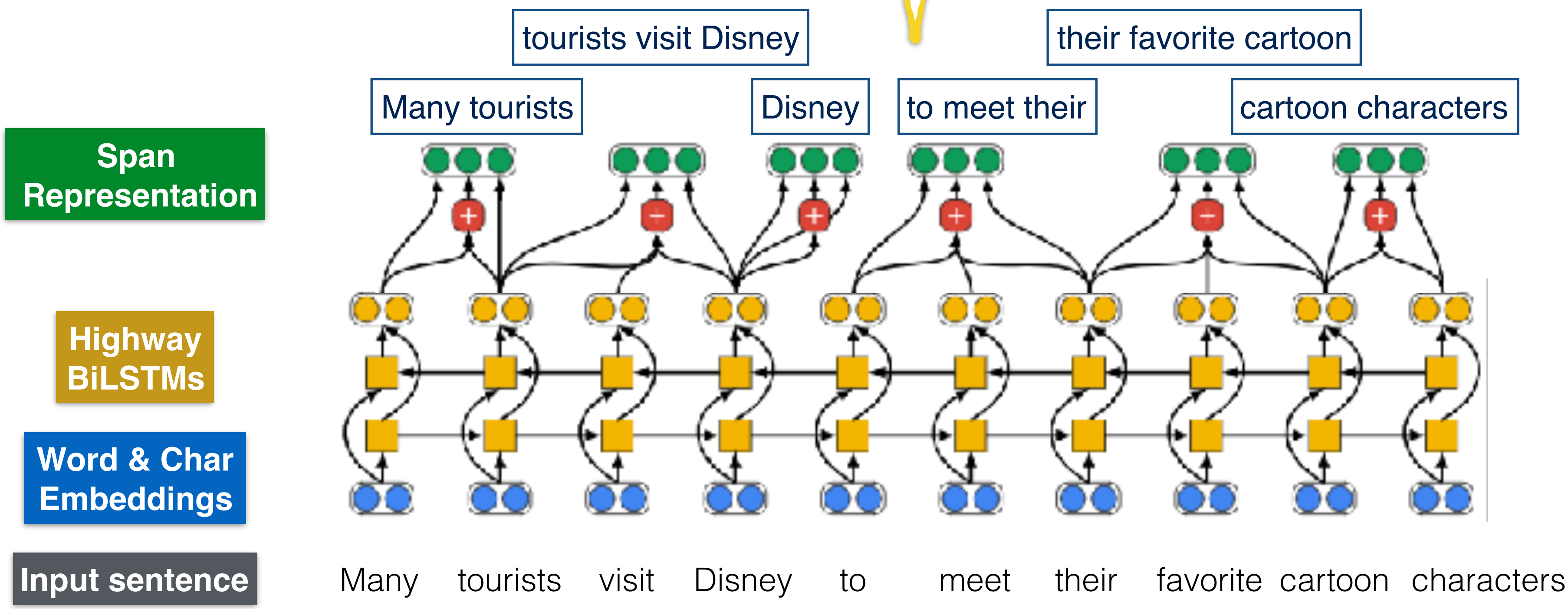


# DeepSRL Architecture (Revisit)



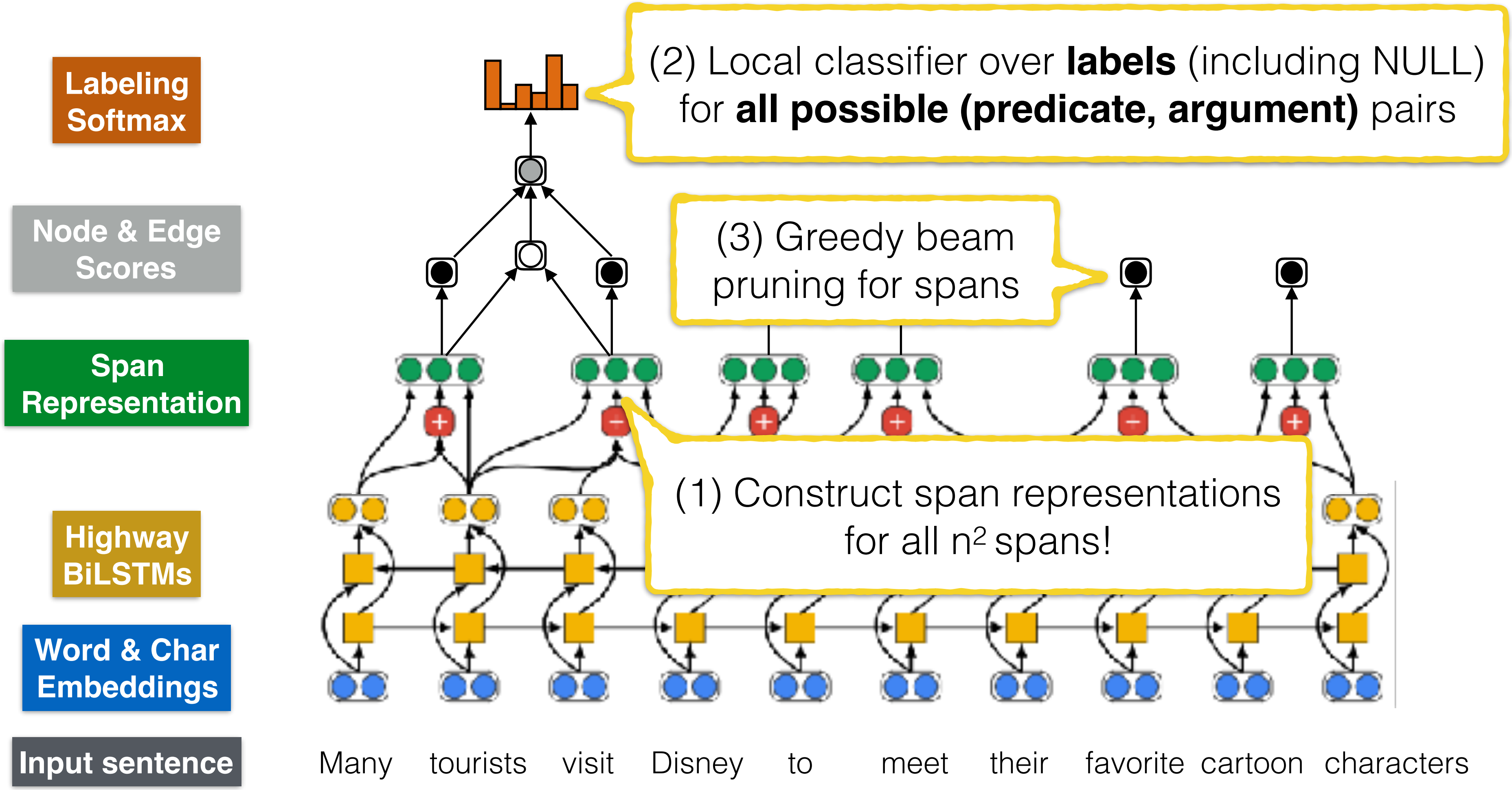
# LSGN Architecture: Overview

(1) Construct span representations for all $n^2$ spans!

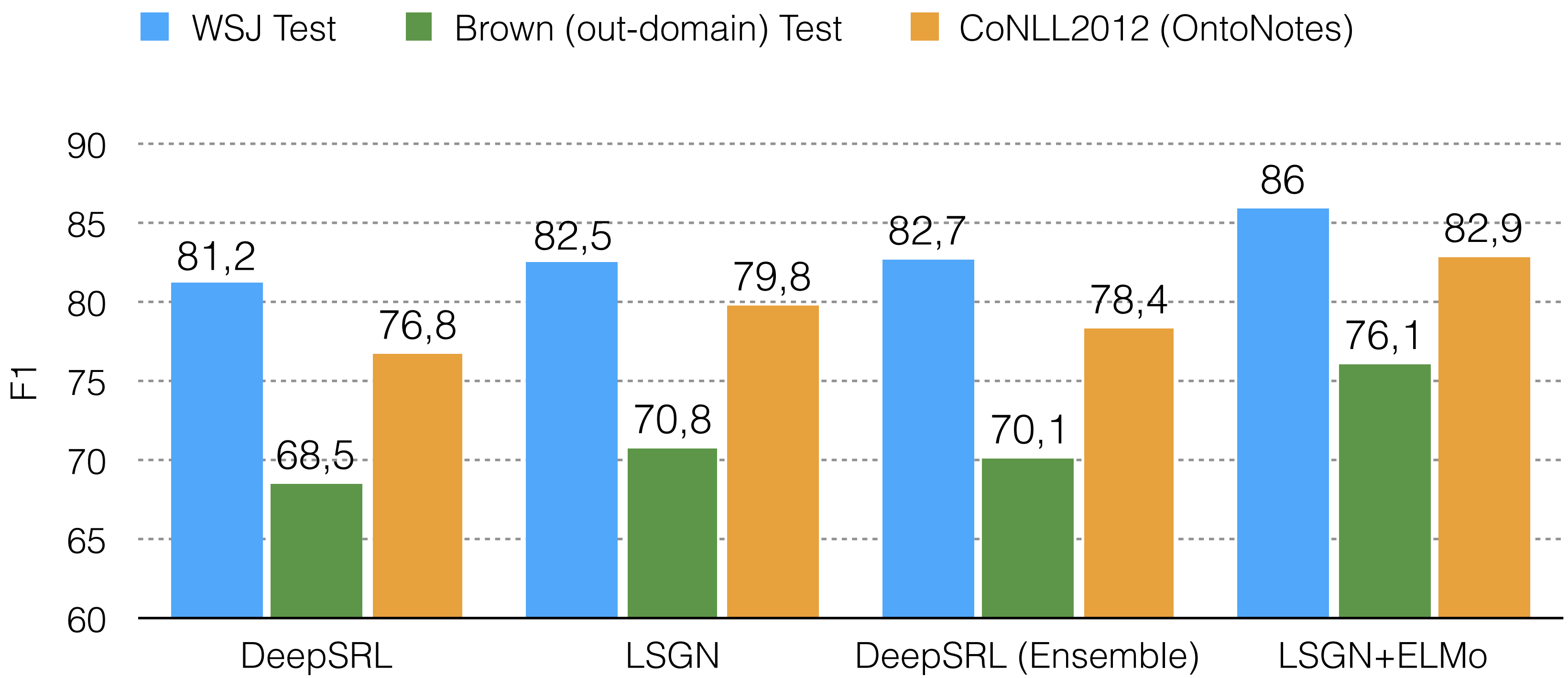



**No predicate input!**

# LSGN Architecture: Overview



# **End-to-End SRL** Results




- More improvements on Brown (out-domain) & OntoNotes (with nominal predicates)
- With ELMo, over 3 points improvement over ensemble model!

# 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
- Semantic Role Labeling