COS 484

Natural Language Processing

L9: Dependency Parsing

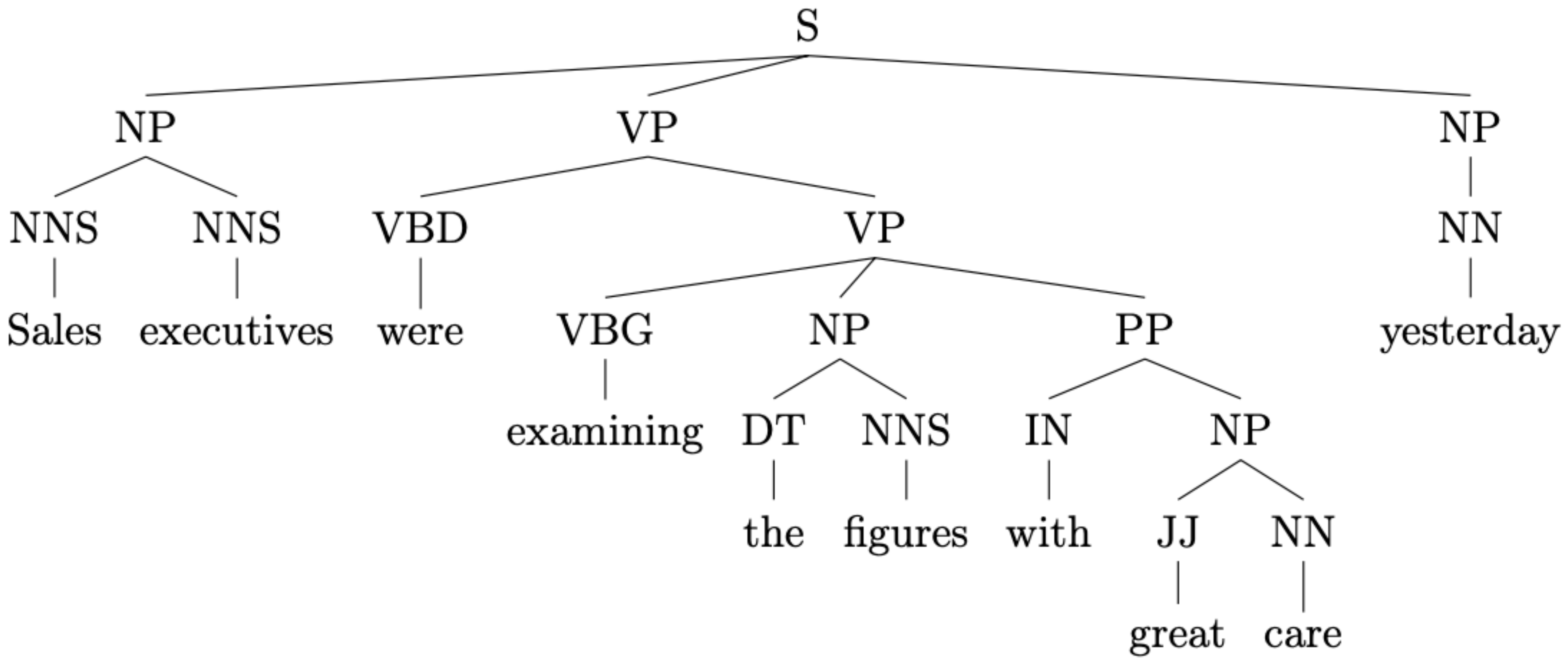
Spring 2023

Logistics

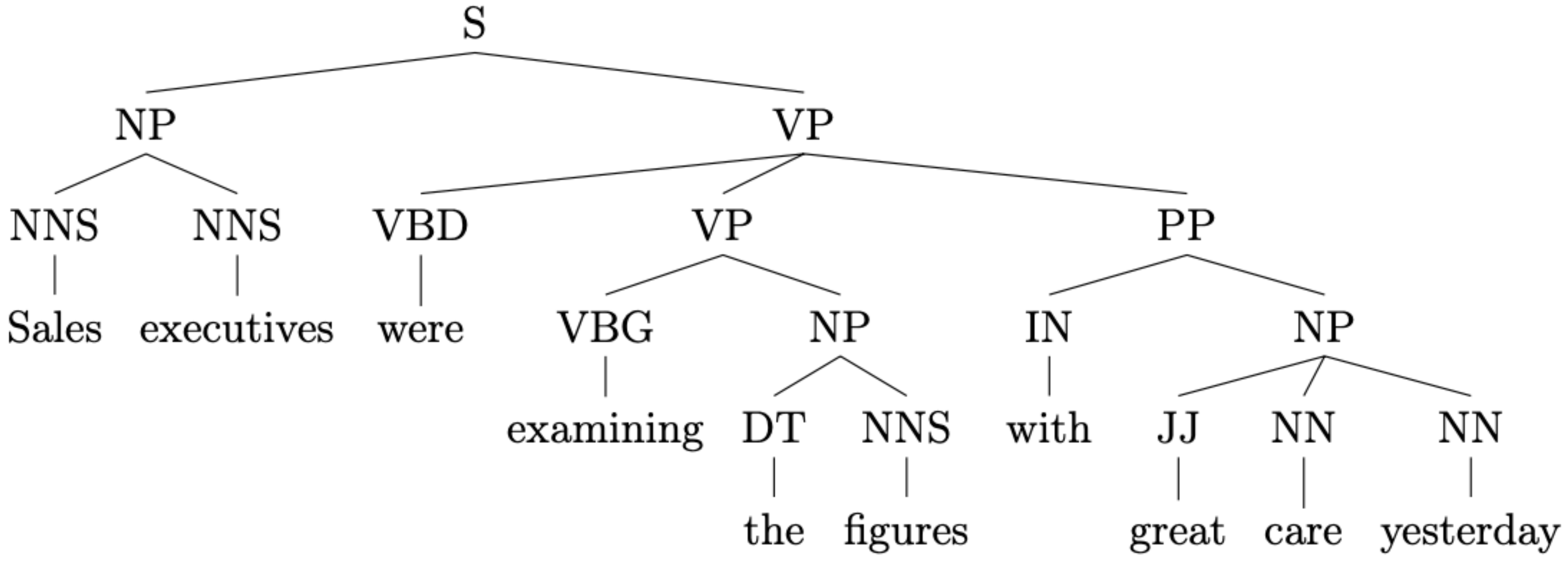
- Stay-tuned for midterm-related announcements

Recap: Constituency Parsing

Gold: **(1, 10, S), (1, 2, NP)**, (3, 9, VP), (4, 9, VP), **(5, 6, NP)**, (7, 9, PP), (8, 9, NP), (10, 10, NP)



Predicted: **(1, 10, S), (1, 2, NP)**, (3, 10, VP), (4, 6, VP), **(5, 6, NP)**, (7, 10, PP), (8, 10, NP)



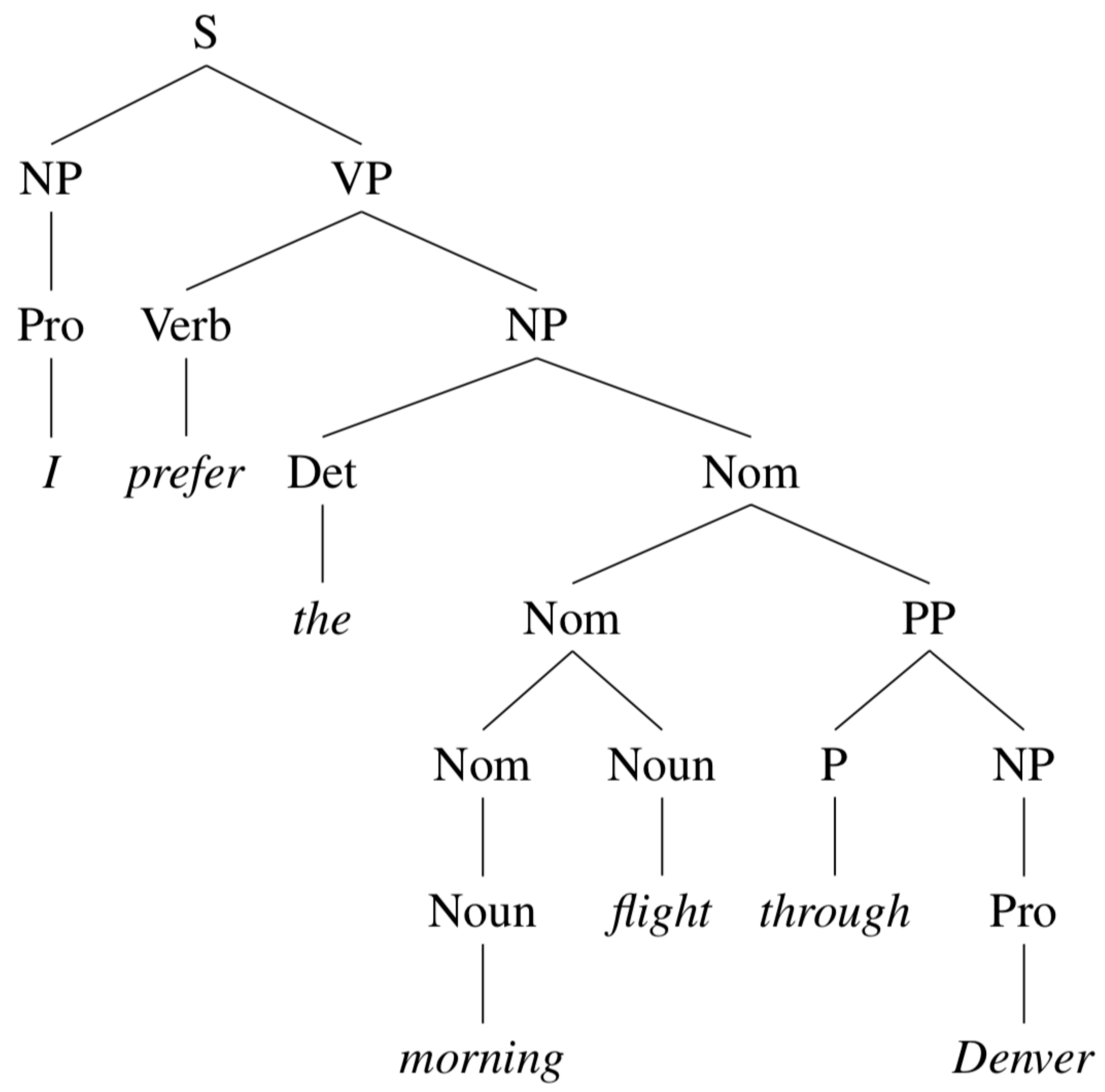
Constituency vs Dependency Parsing

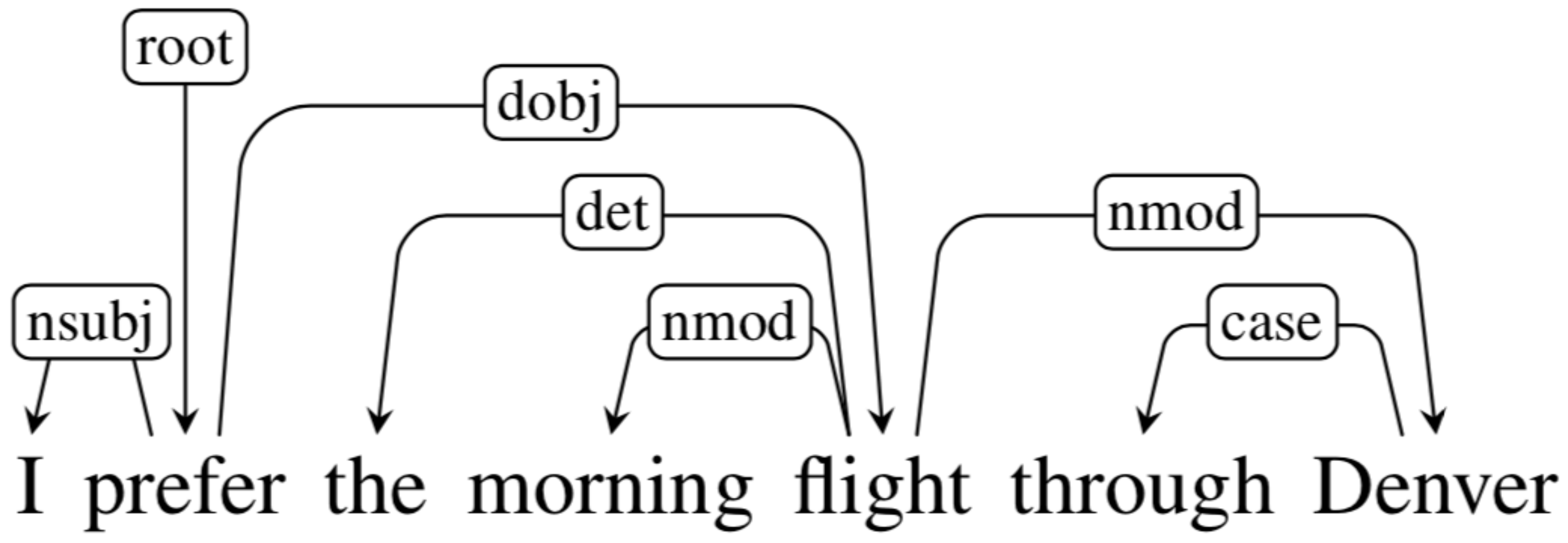
- Constituency structure
- Context-free grammar (CFG)
- Probabilistic context-free grammar (PCFG)
- Treebanks
- The CKY algorithm
- Evaluation
- Lexicalized PCFGs

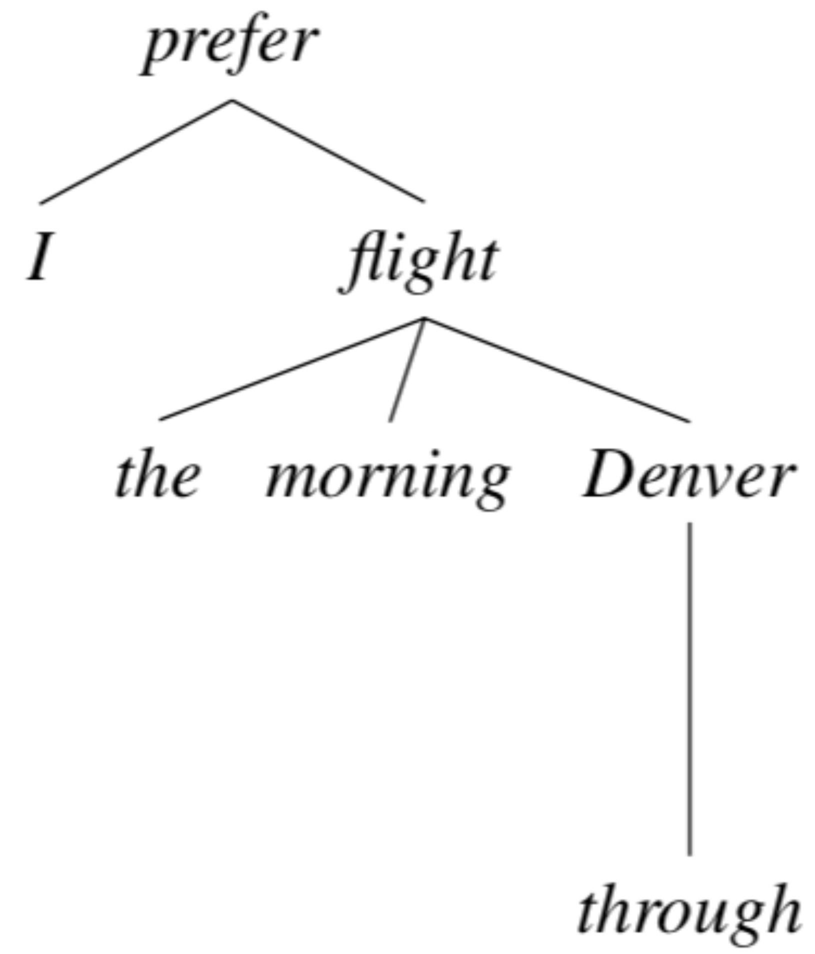
→

- Dependency structure
- The Arc-standard algorithm
- Dependency treebanks
- Evaluation

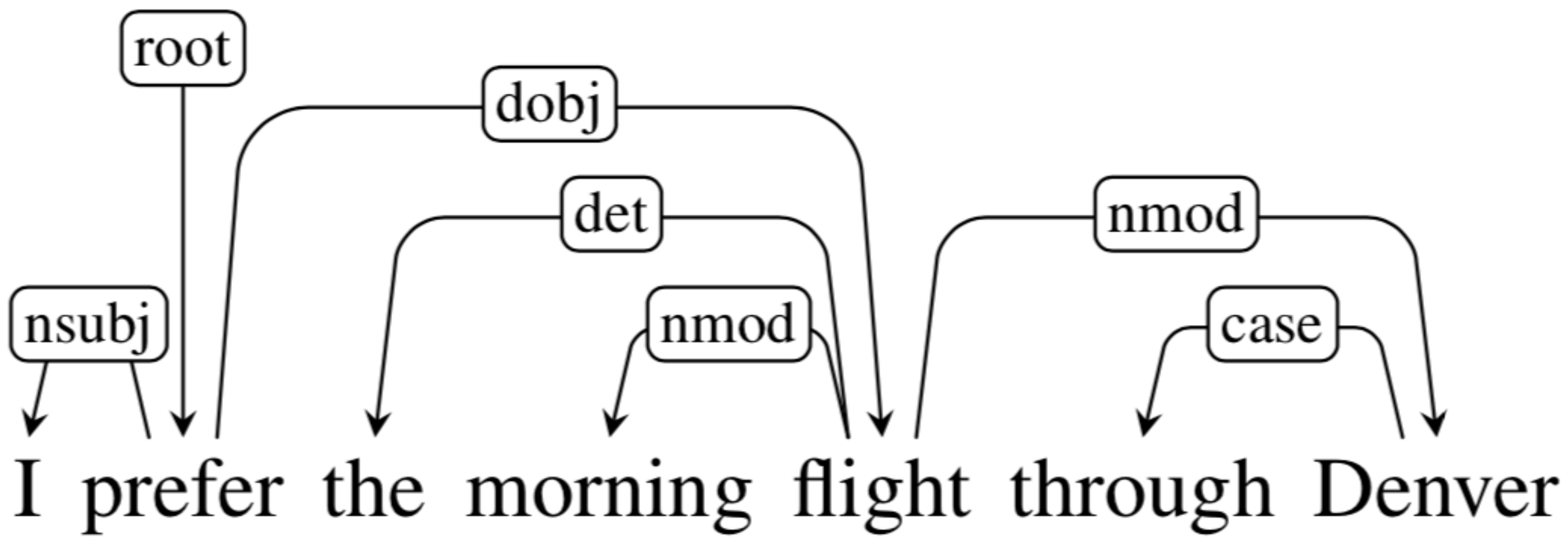
Constituency vs Dependency Structure







Dependency Structure



- Consists of relations between lexical items, normally *binary*, *asymmetric* relations ("arrows") called **dependencies**
- The arrow connects a **head** (governor) and a **dependent** (modifier)
- The arrows are commonly **typed** with the name of grammatical relations (e.g., **nominal subject**, **direct object**)
- Dependencies form a tree

Dependency Relations

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

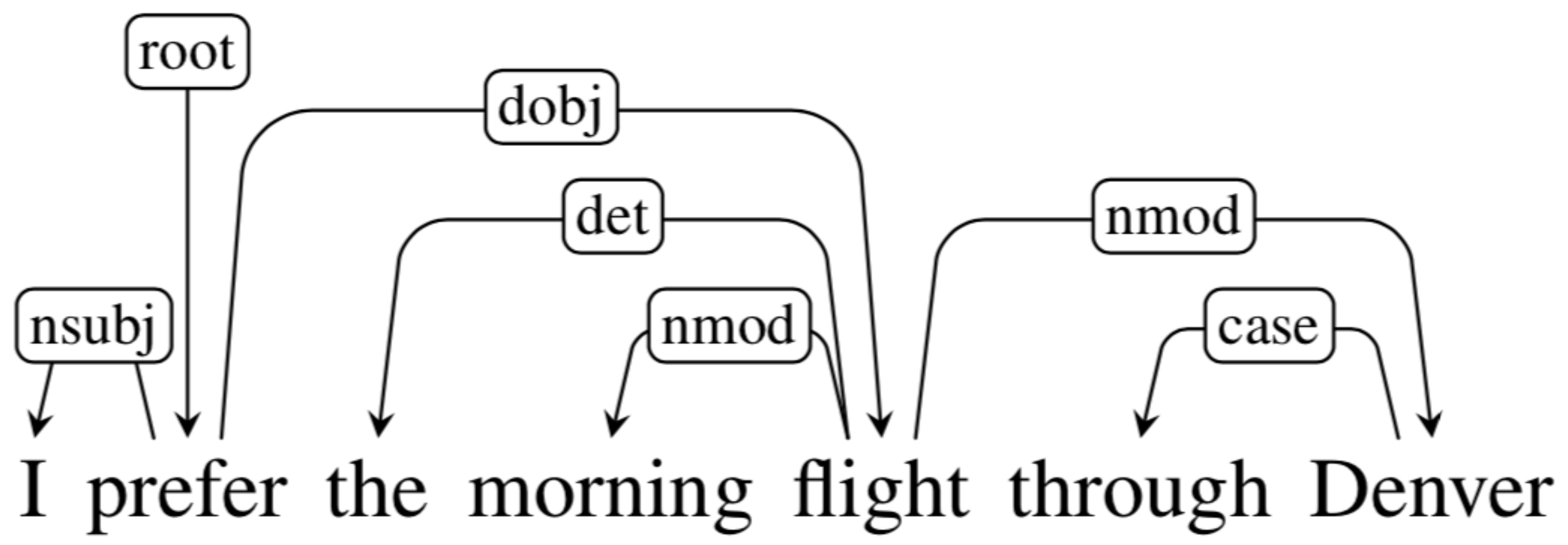
Dependency Relations

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

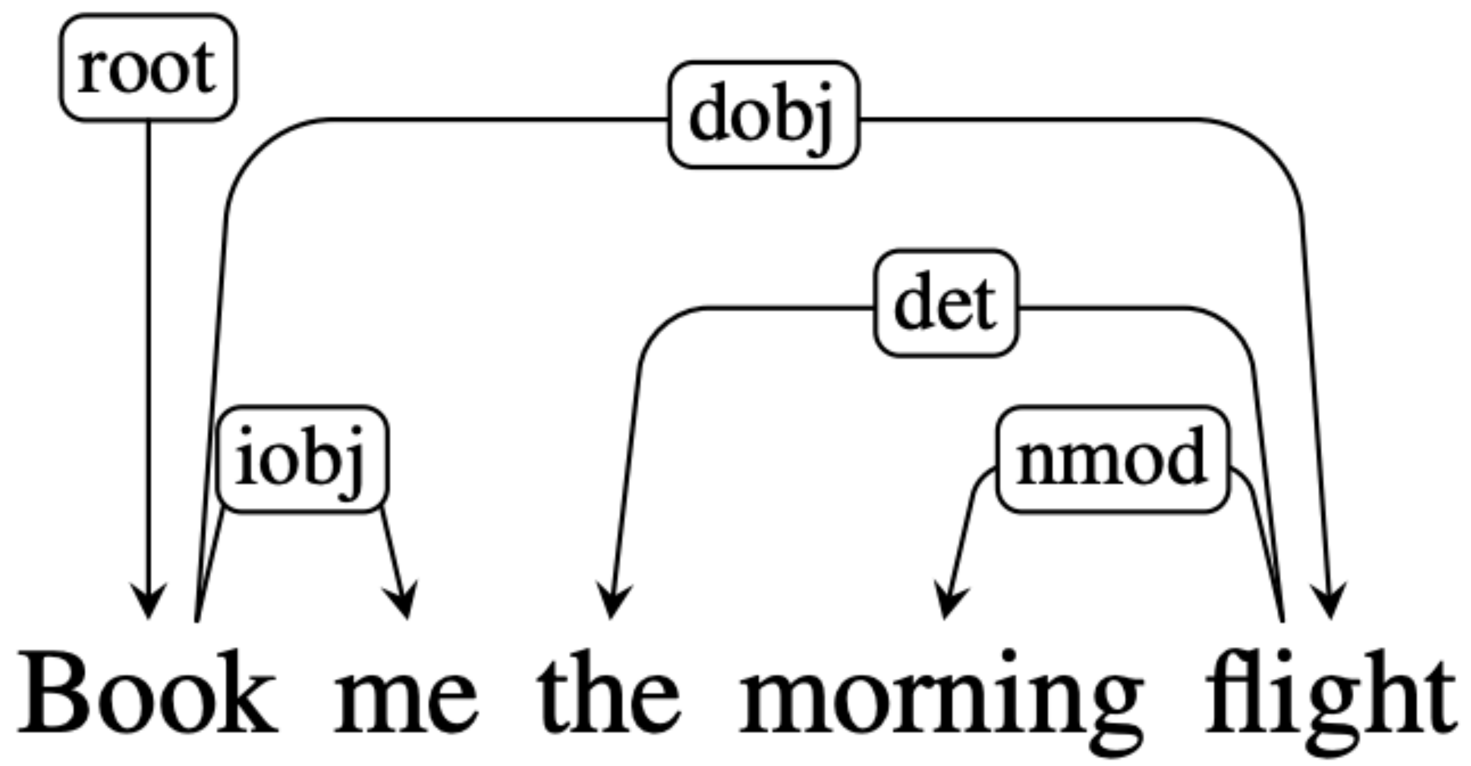
Figure 14.3 Examples of core Universal Dependency relations.

Dependency Structure: More Examples

I prefer the morning flight through Denver



Book me the morning flight



Dependency Parsing

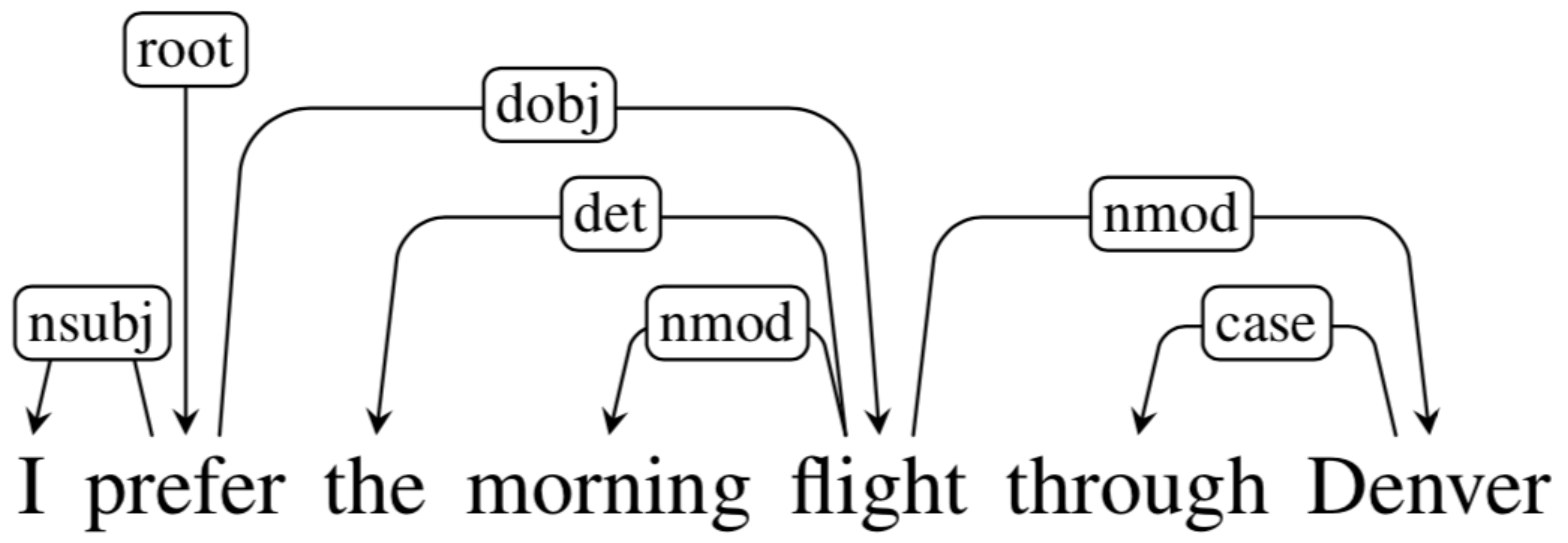
Syntactic parsing is the task of recognizing a sentence and assigning a structure to it.

Dependency parsing is the task of recognizing a sentence and assigning a **dependency** structure to it.

Input

I prefer the morning flight through Denver

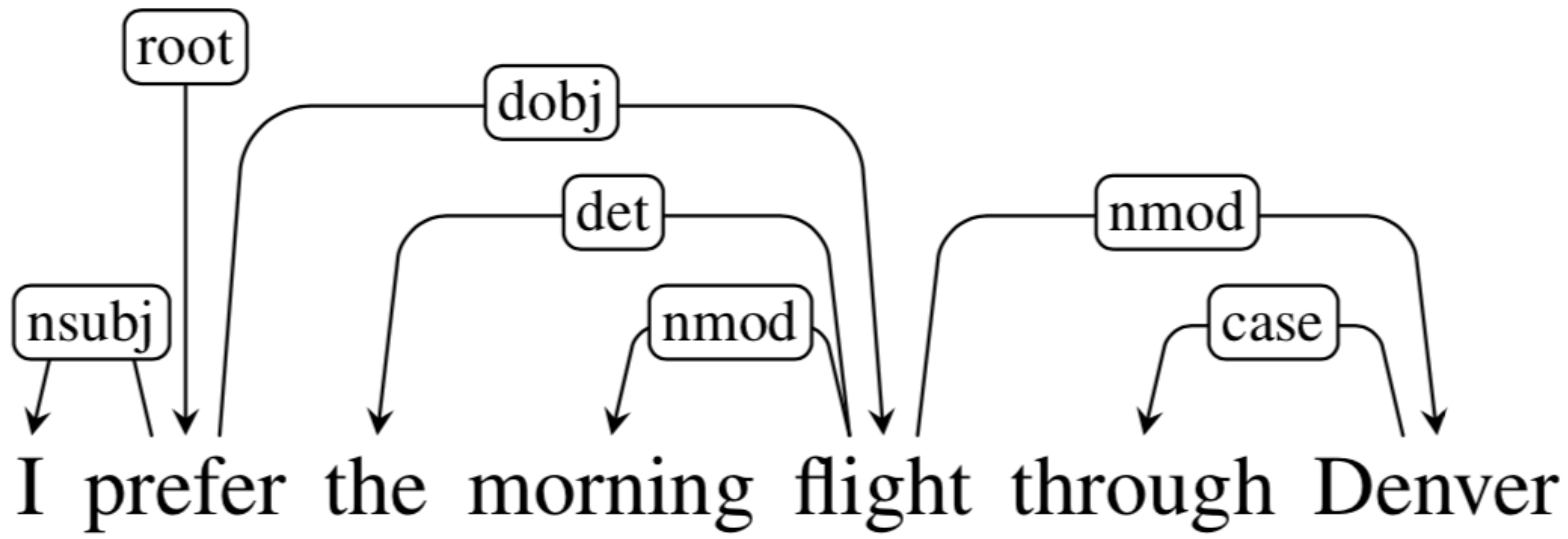
Output



Dependency Formalisms

Structure is a **dependency tree (a directed graph)**

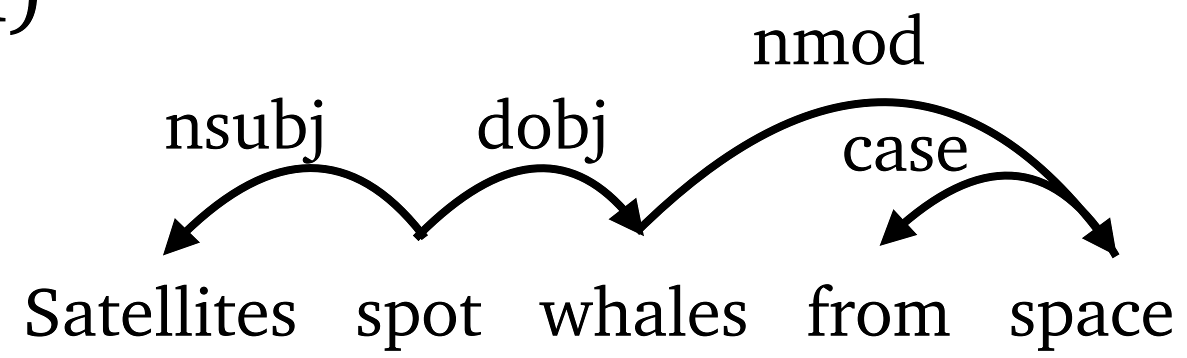
- There is only one root
- Every vertex, except the root, has one head (parent)
    - Alternatively, we can just add a fake node ROOT, so each word has exactly one head
- There is a unique path from the root node to each vertex
    - No cycles: A —> B, B —> C, C —> A



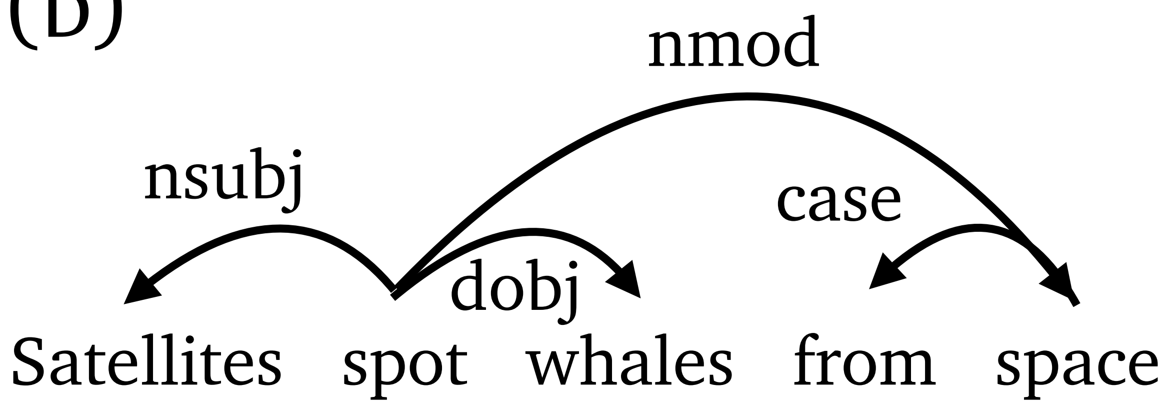
Poll

Which of the following is the correct dependency structure for "Satellites spot whales from space"?

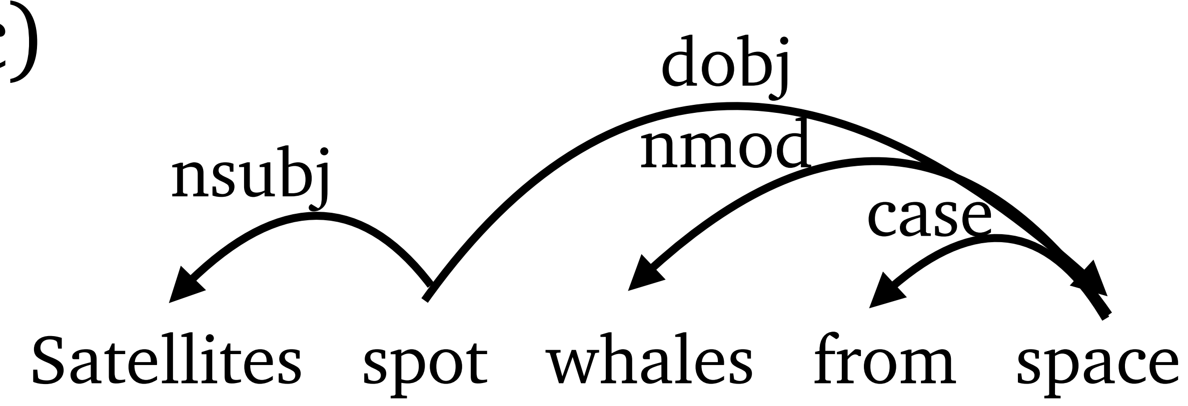
(a)



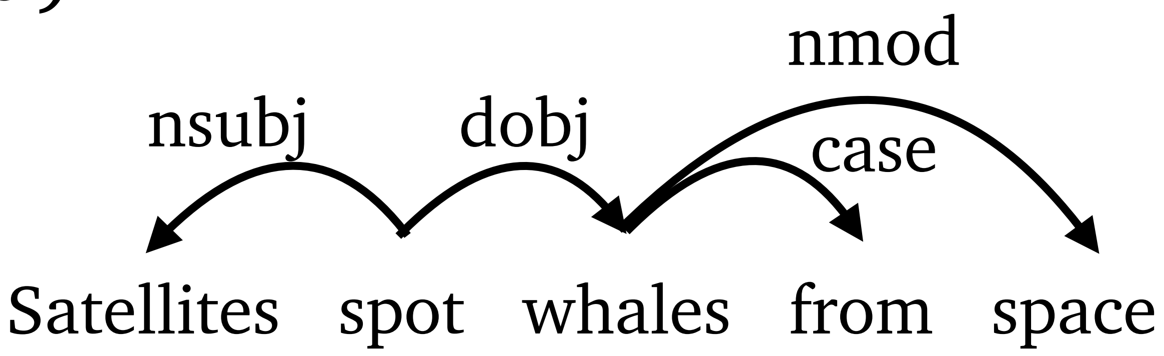
(b)



(c)



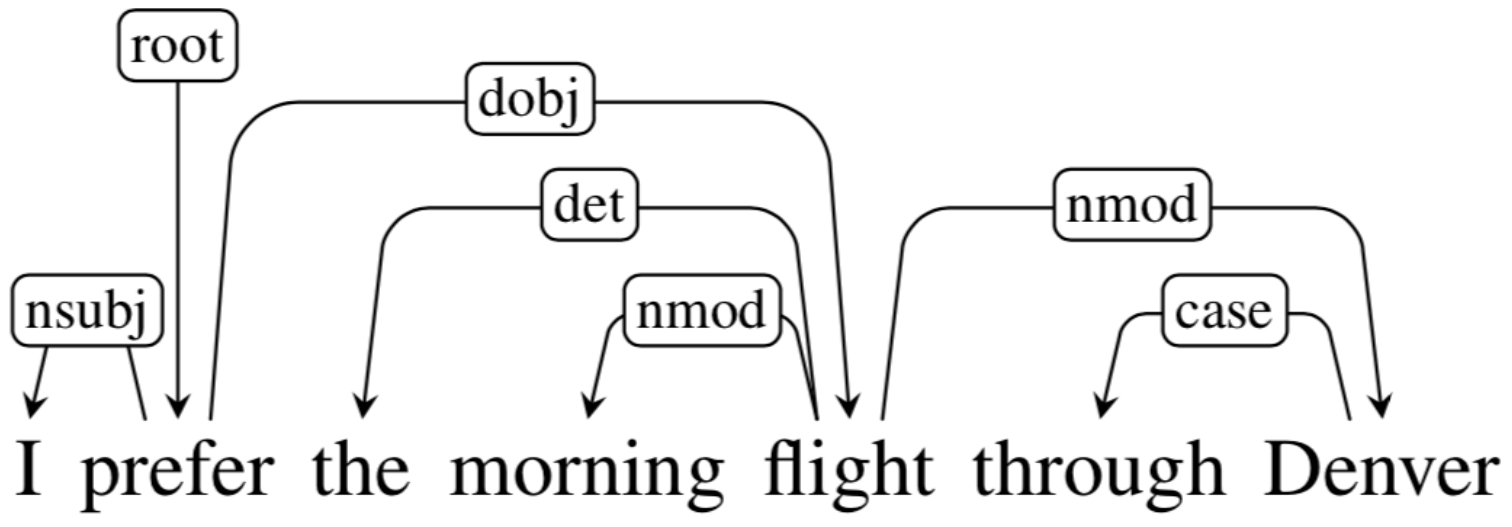
(d)



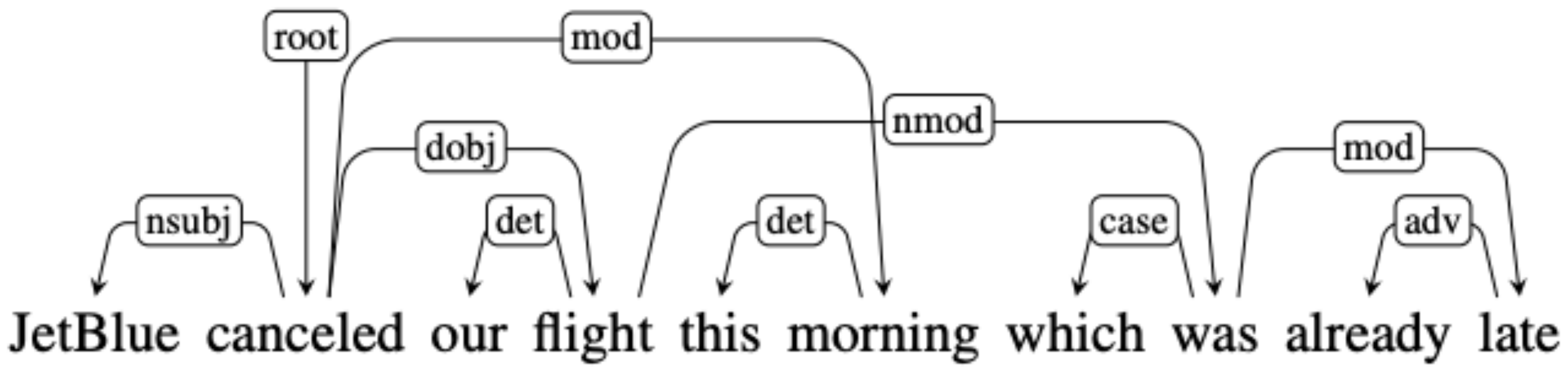
Dependency formalisms

Additional constraint: **projectivity**

- **Definition**: there are no crossing dependency arcs 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.

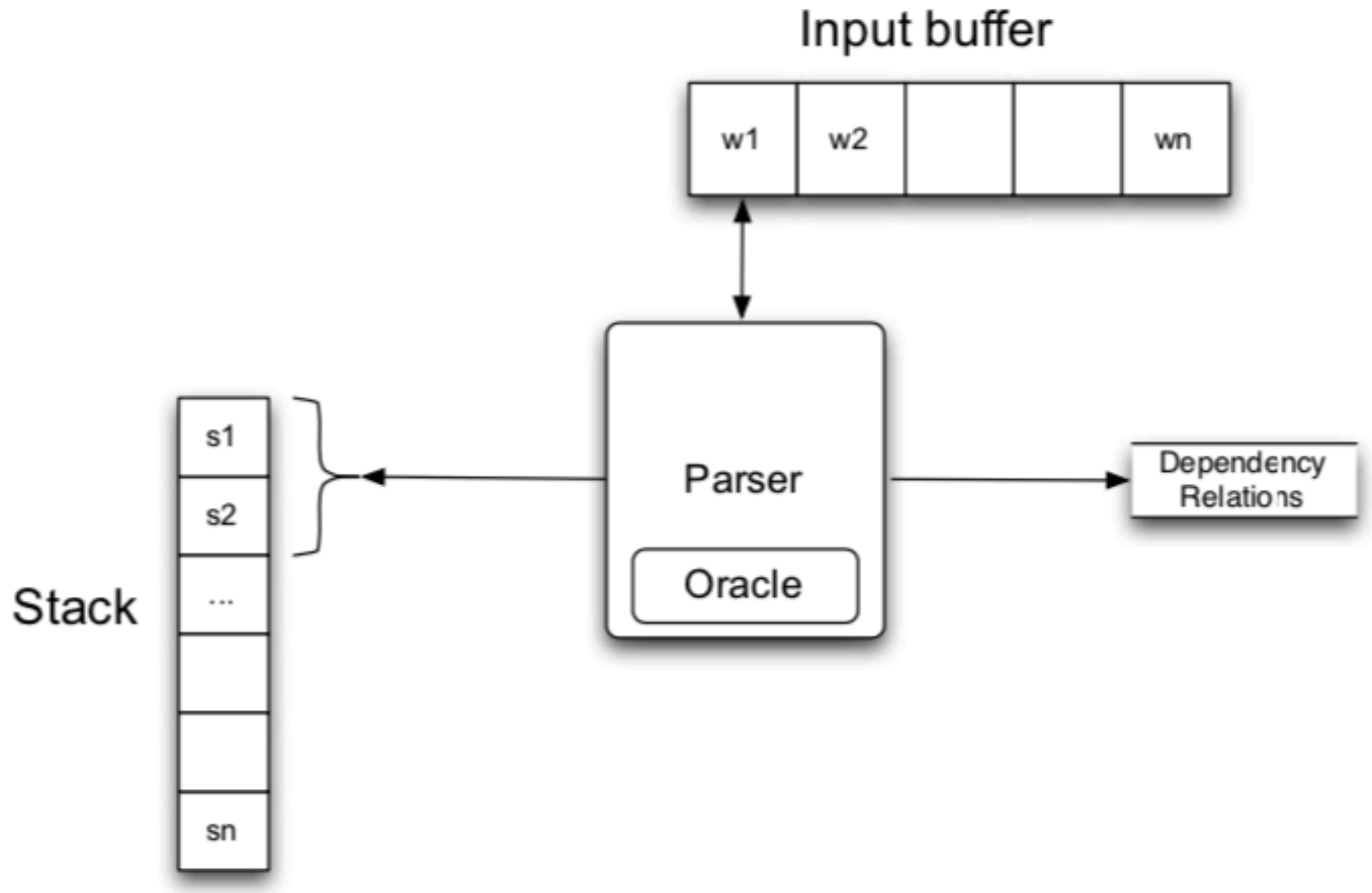
We will focus on projective parsing

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

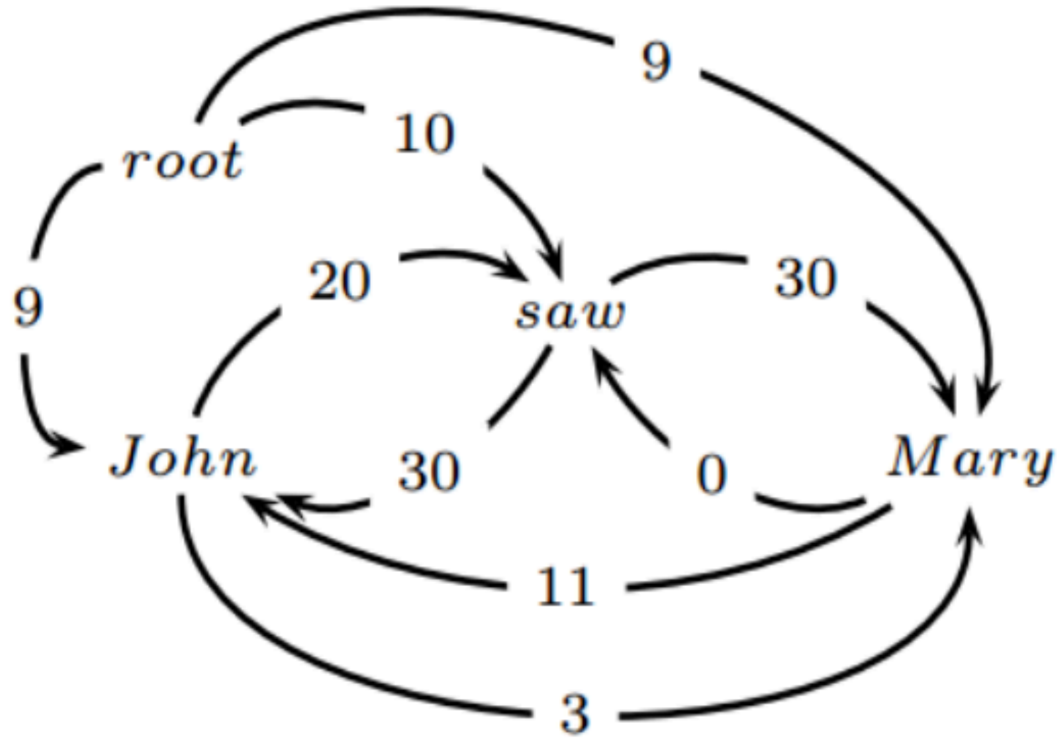
Two families of algorithms

Transition-based dependency parsing

- Also called “shift-reduce parsing”



Graph-based dependency parsing



The Arc-standard algorithm

- Given: a sentence of $w_1, w_2, \ldots, w_n$
- The parsing process is modeled as a sequence of transitions
- A configuration (current state of parse) consists of a stack s, a buffer b and a set of dependency arcs A: $c = (s, b, A)$
- Initially, $s = [\text{ROOT}]$, $b = [w_1, w_2, \ldots, w_n]$, $A = \varnothing$
- A configuration is terminal if $s = [\text{ROOT}]$ and $b = \varnothing$
- Three types of transitions: SHIFT, LEFT-ARC (l), RIGHT-ARC (r)

The Arc-standard algorithm

- Three types of transitions: SHIFT, LEFT-ARC (r), RIGHT-ARC (r)

Arc-standard system: three operations

- Shift: top of buffer -> top of stack
- Left-Arc: $\boxed{\sigma | w_{-2}, w_{-1}} \rightarrow \boxed{\sigma | w_{-1}}$, w_{-2} is now a child of w_{-1}
- Right-Arc $\boxed{\sigma | w_{-2}, w_{-1}} \rightarrow \boxed{\sigma | w_{-2}}$, w_{-1} is now a child of w_{-2}

The Arc-standard algorithm

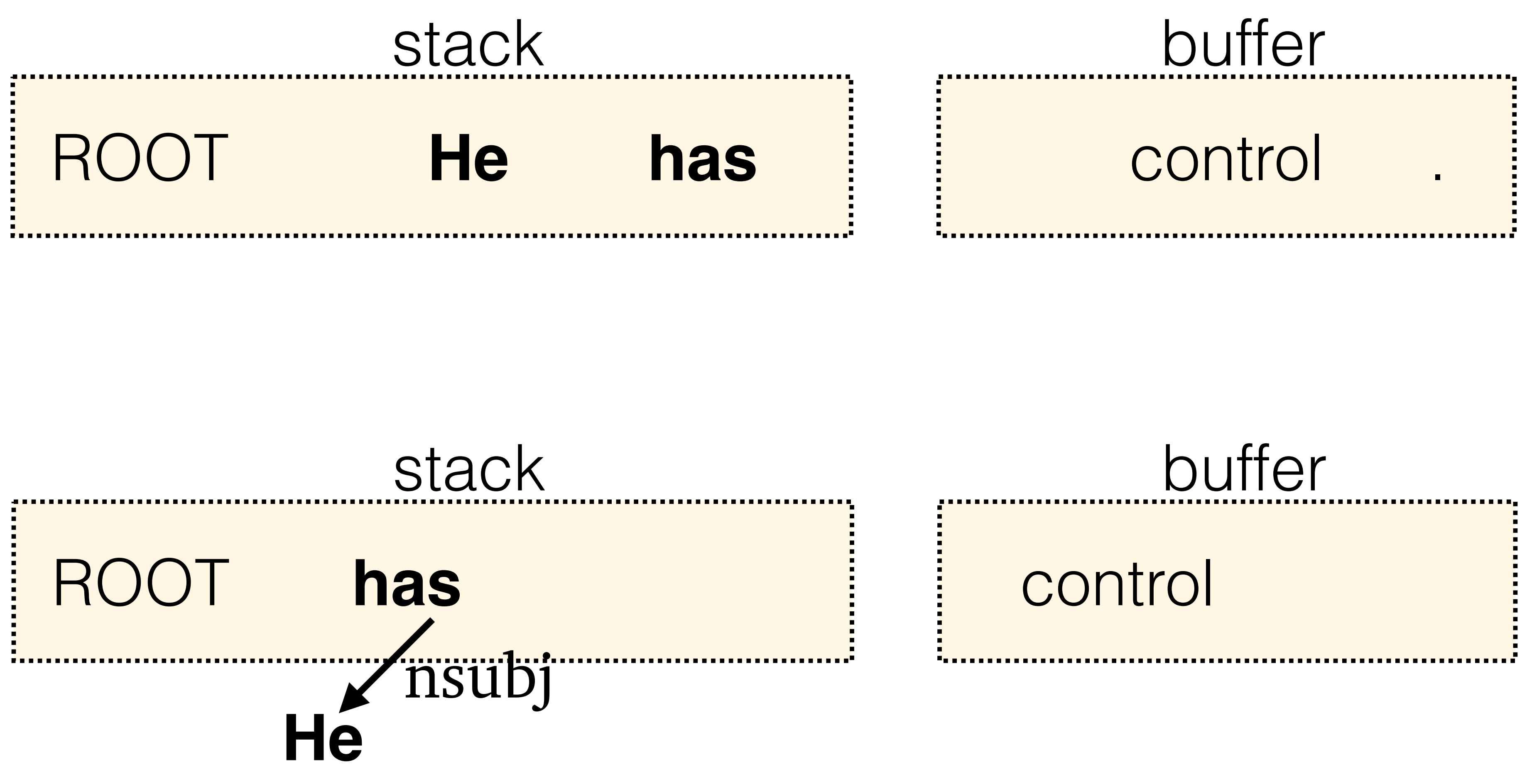
s_1, s_2: the top 2 words on the stack (s_1 = He, s_2 = has);

b_1: the first word in the buffer (b_1 = control)

LEFT-ARC (r): add an arc ($s_2 \xrightarrow{r} s_1$) to A, remove s_1 (modifier) from stack

Current configuration

After transition



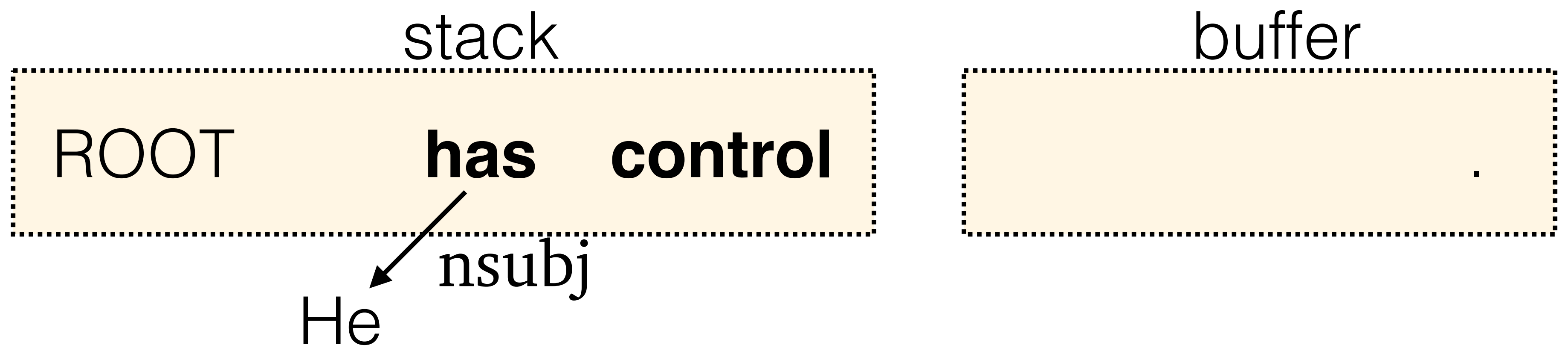
The Arc-standard algorithm

s_1, s_2: the top 2 words on the stack (s_1 = has, s_2 = control);

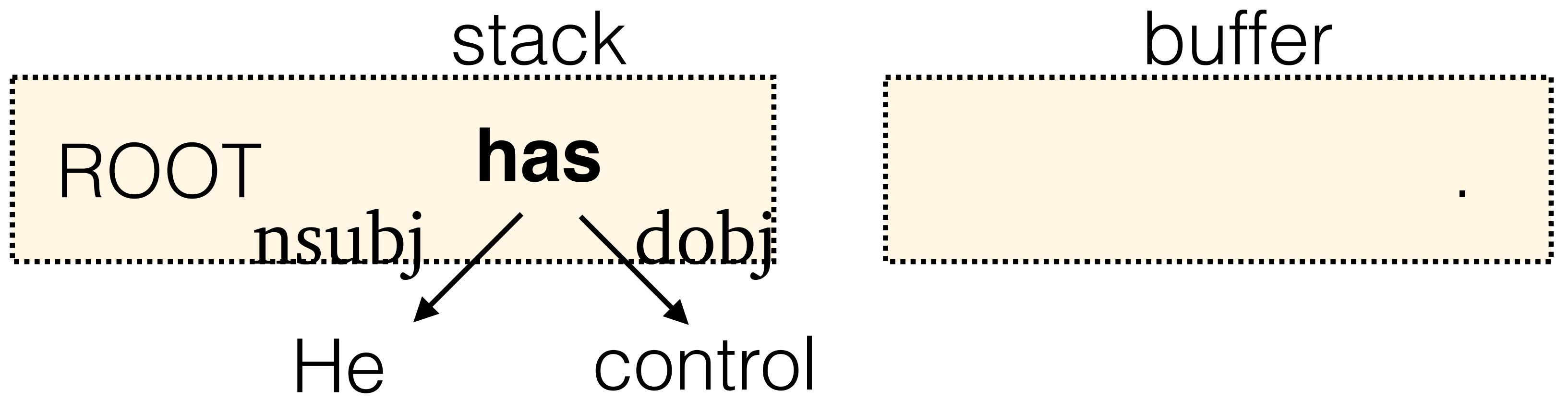
b_1: the first word in the buffer (b_1 = .)

RIGHT-ARC (r): add an arc ($s_1 \xrightarrow{r} s_2$) to A, remove s_1 from the stack

Current configuration



After transition



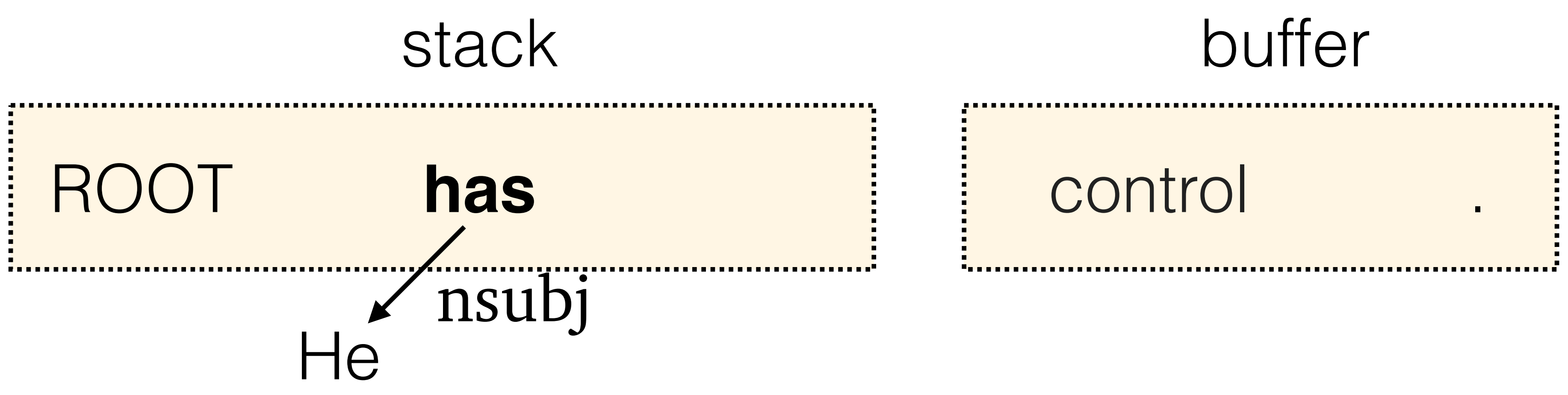
The Arc-standard algorithm

s_1, s_2: the top 2 words on the stack (s_1 = ROOT, s_2 = has);

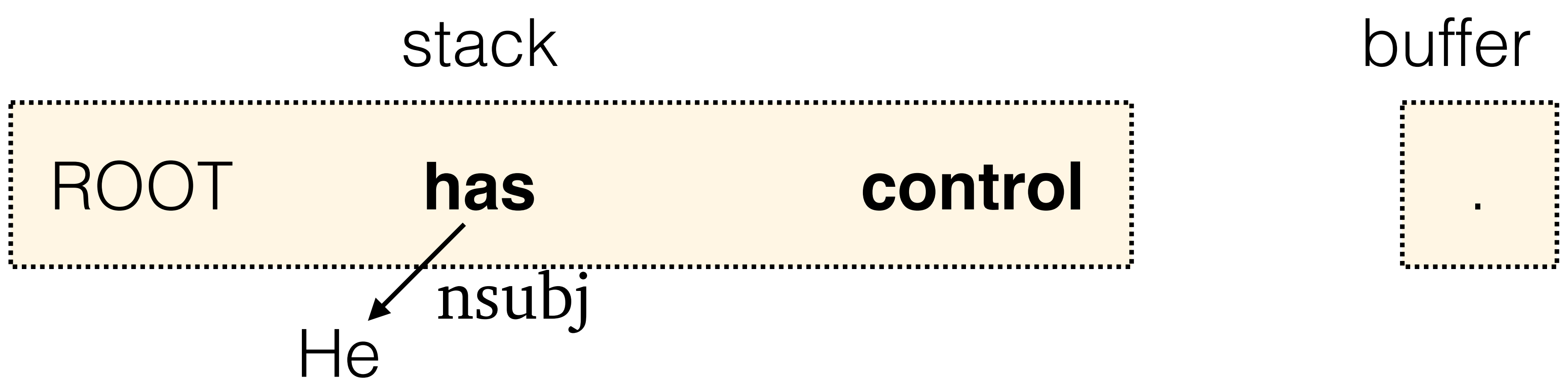
b_1: the first word in the buffer (b_1 = control)

SHIFT: move b_1 from the buffer to the stack

Current configuration



After transition

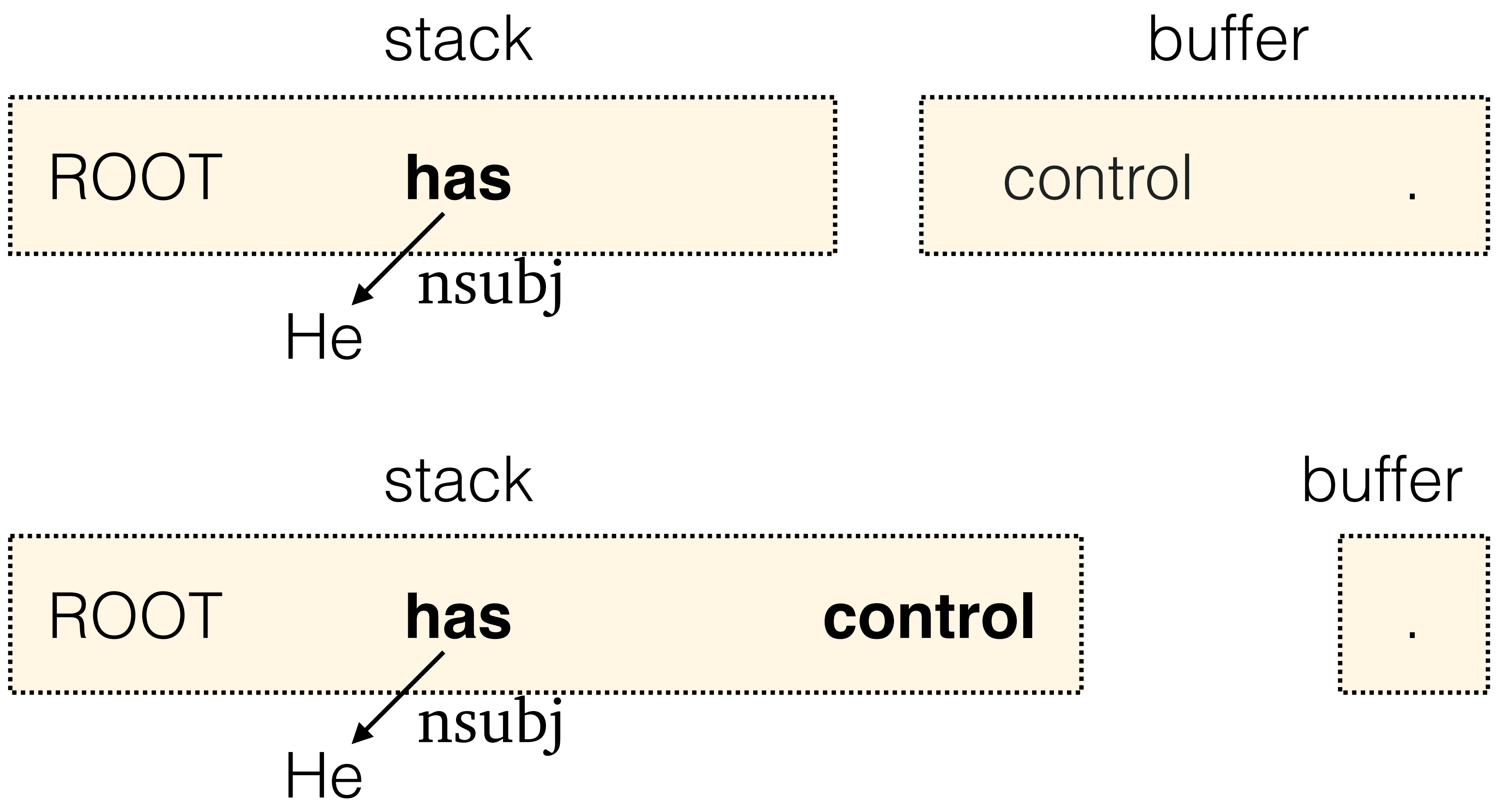


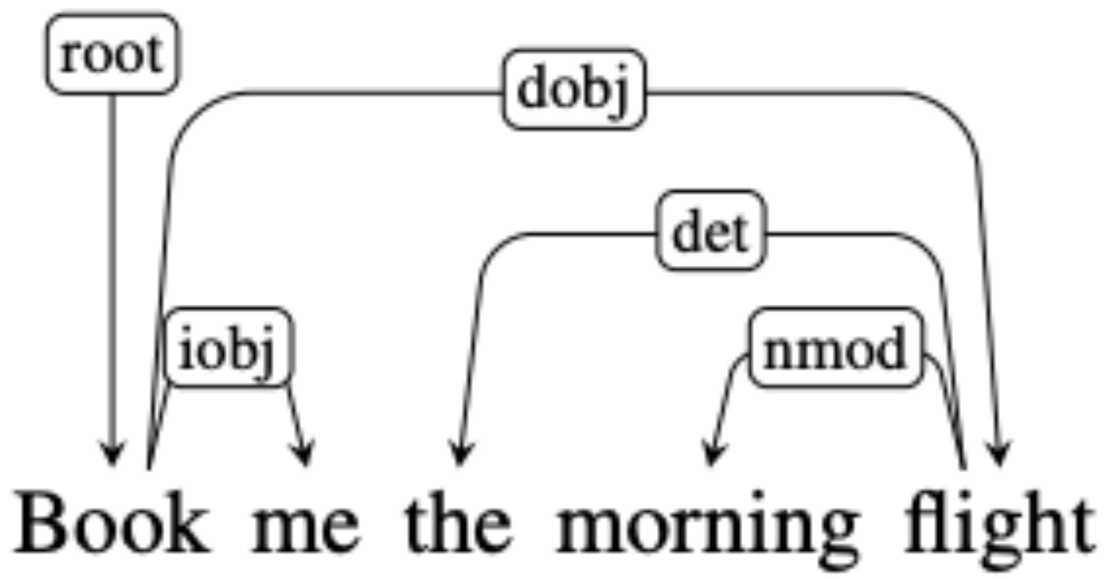
The Arc-standard algorithm

s_1, s_2: the top 2 words on the stack (s_1 = ROOT, s_2 = has);

b_1: the first word in the buffer (b_1 = control)

SHIFT: move b_1 from the buffer to the stack



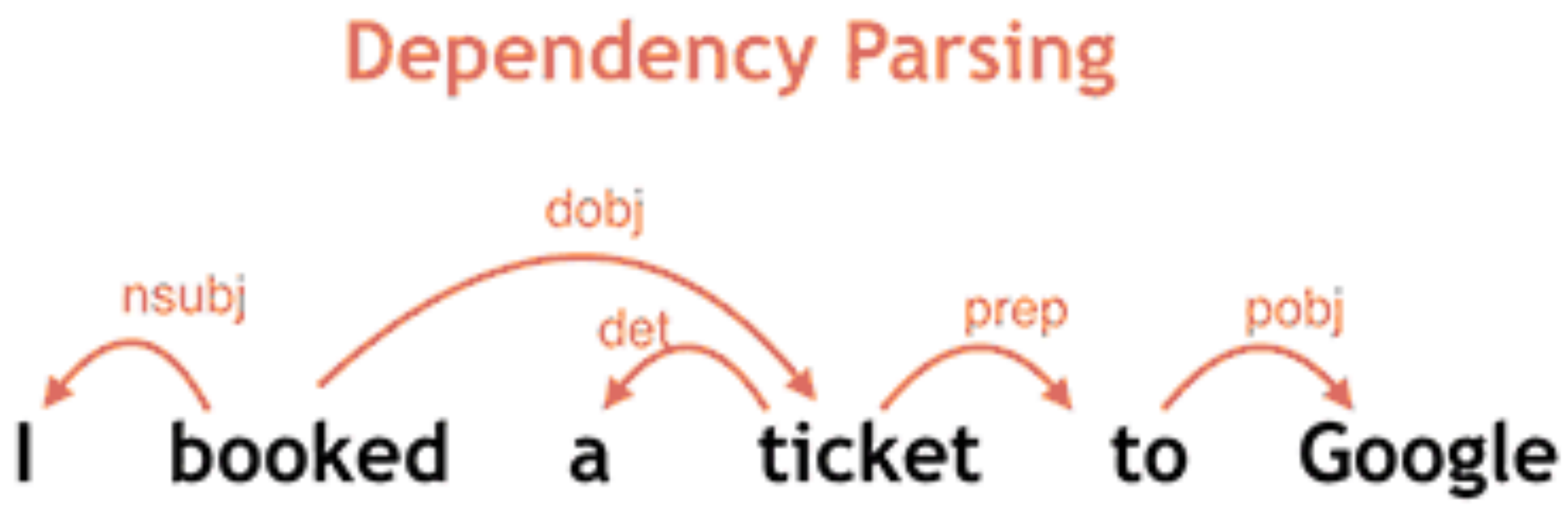


“Book me the morning flight”

A running example

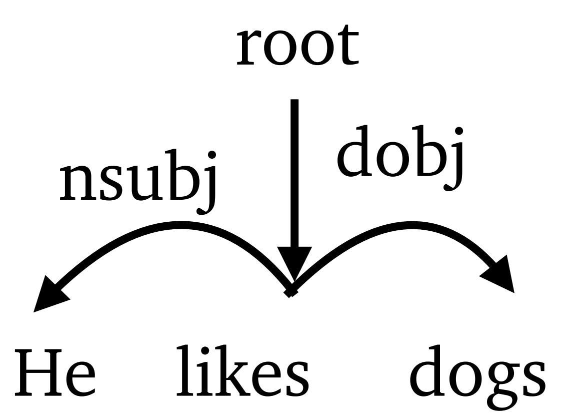
	stack	buffer	action	added arc
0	[ROOT]	[Book, me, the, morning, flight]	SHIFT	
1	[ROOT, Book]	[me, the, morning, flight]	SHIFT	
2	[ROOT, Book, me]	[the, morning, flight]	RIGHT-ARC(iobj)	(Book, iobj, me)
3	[ROOT, Book]	[the, morning, flight]	SHIFT	
4	[ROOT, Book, the]	[morning, flight]	SHIFT	
5	[ROOT, Book, the, morning]	[flight]	SHIFT	
6	[ROOT, Book, the,morning,flight]	[]	LEFT-ARC(nmod)	(flight,nmod,morning)
7	[ROOT, Book, the, flight]	[]	LEFT-ARC(det)	(flight,det,the)
8	[ROOT, Book, flight]	[]	RIGHT-ARC(dobj)	(Book,dobj,flight)
9	[ROOT, Book]	[]	RIGHT-ARC(root)	(ROOT,root,Book)
10	[ROOT]	[]		

Transition-based dependency parsing



https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

Poll



Which of the following transition sequences is correct for the sentence “He likes dogs”?

(a) SHIFT, SHIFT, RIGHT-ARC(dobj), SHIFT, LEFT-ARC(nsubj), RIGHT-ARC(root)

(b) SHIFT, SHIFT, SHIFT, RIGHT-ARC(dobj), LEFT-ARC(nsubj), RIGHT-ARC(root)

(c) SHIFT, SHIFT, LEFT-ARC(nsubj), SHIFT, RIGHT-ARC(dobj), RIGHT-ARC(root)

(d) SHIFT, SHIFT, SHIFT, LEFT-ARC(nsubj), RIGHT-ARC(dobj), RIGHT-ARC(root)

Transition-based dependency parsing

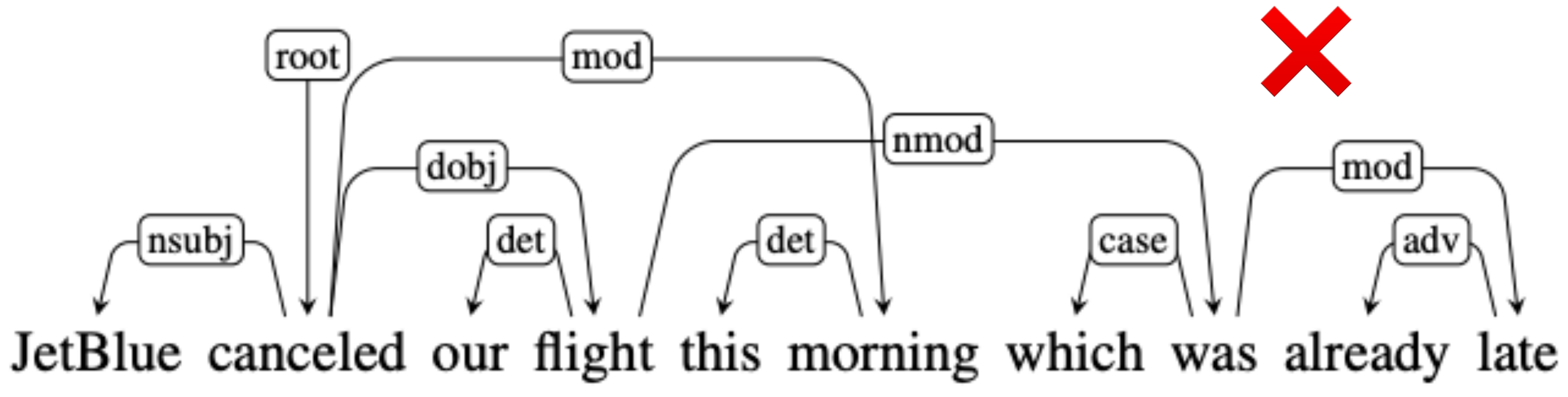
Given: a sentence of $w_1, w_2, \ldots, w_n$

Q: How many transitions are needed? How many times of SHIFT?

Correctness **[advanced]**

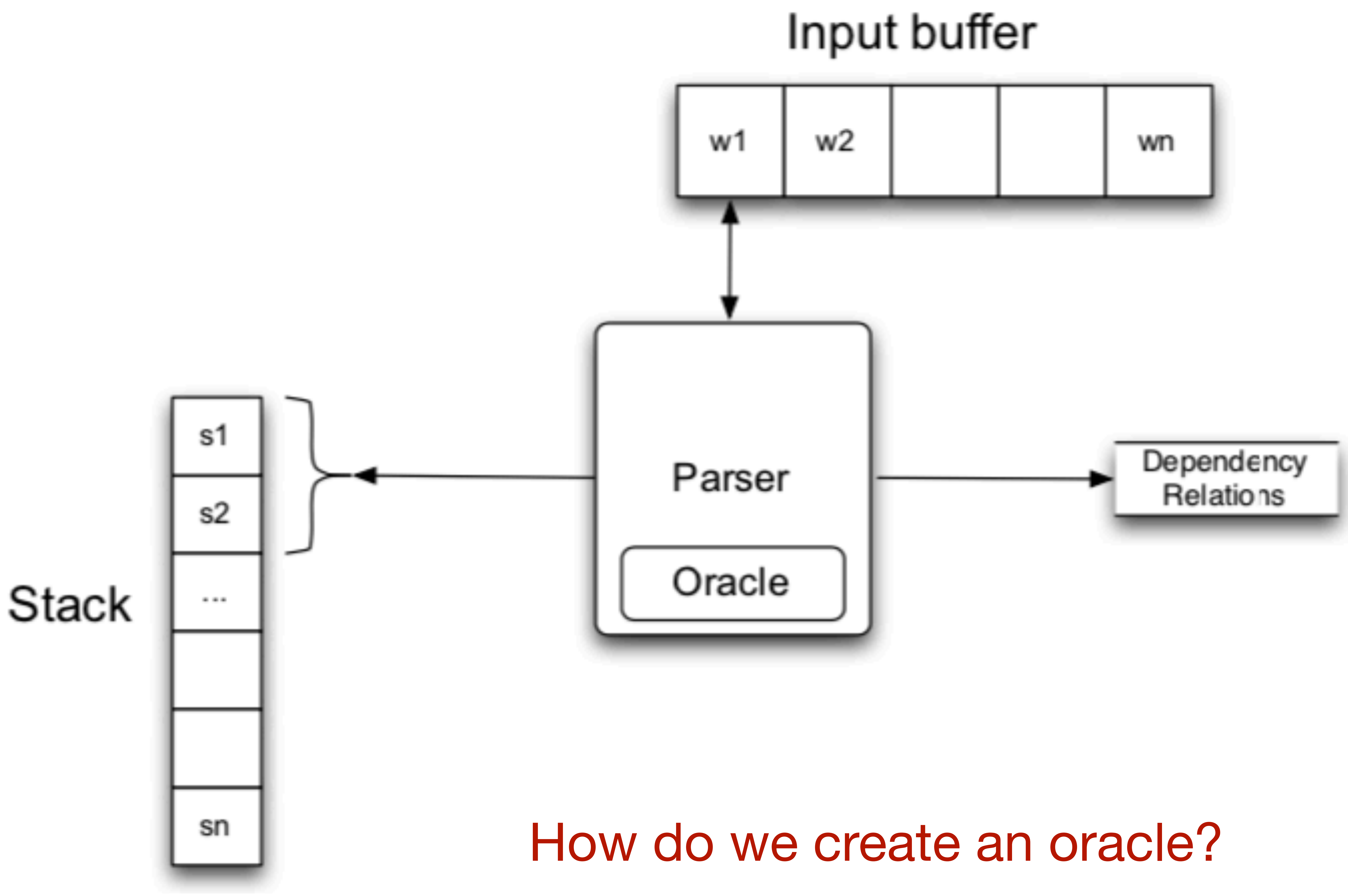
- For every complete transition sequence, the resulting graph is a projective dependency forest (soundness)

- For every projective dependency tree G, there is a transition sequence that generates G (completeness)



However, one parse tree can have multiple valid transition sequences.

Transition-based dependency parsing



How do we create an oracle?

How to decide which transitions to take?

Key idea: we can learn a statistical machine learning model from dependency treebanks!

- English dependency treebank: converted from Penn Treebank using rule-based algorithms
  - (De Marneffe et al, 2006): Generating typed dependency parses from phrase structure parses
  - (Johansson and Nugues, 2007): Extended Constituent-to-dependency Conversion for English

- Universal Dependencies: nearly 200 treebanks in 100 languages being collected since 2016

Universal Dependencies

Universal Dependencies (UD) is a framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages. UD is an open community effort with over 300 contributors producing nearly 200 treebanks in over 100 languages. If you're new to UD, you should start by reading the first part of the Short Introduction and then browsing the annotation guidelines.

https://universaldependencies.org/

Universal Dependencies

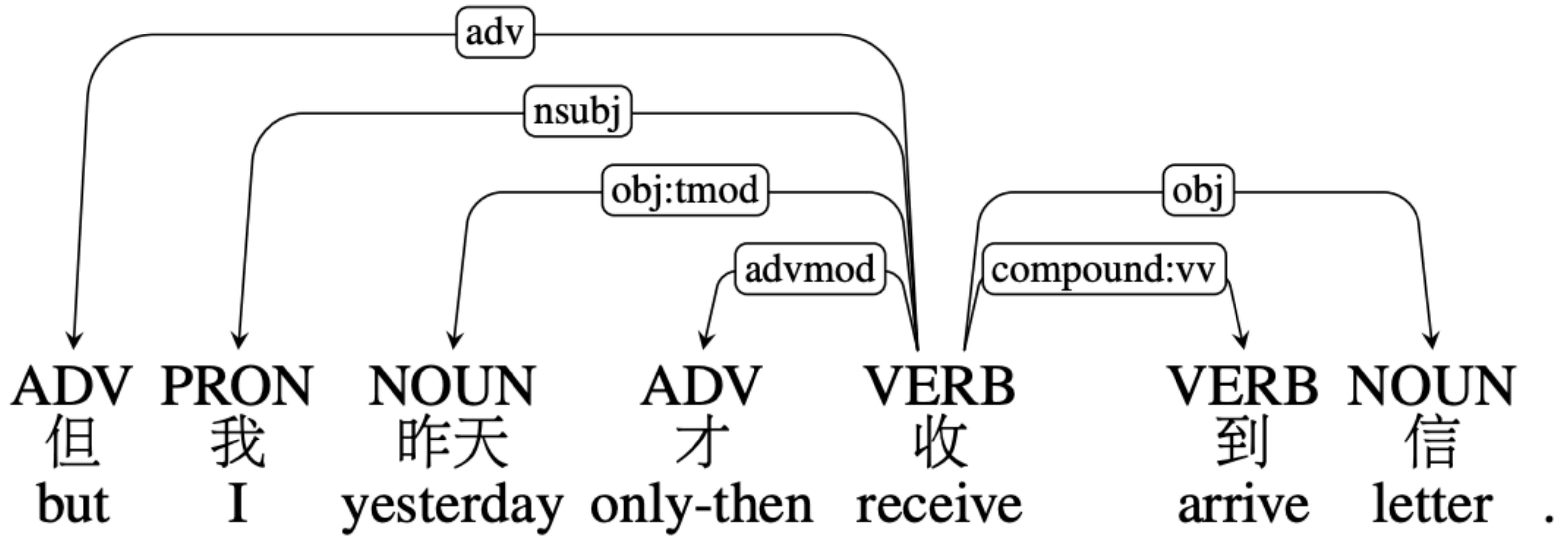
Current UD Languages

Information about language families (and genera for families with multiple branches) is mostly taken from WALS Online (IE = Indo-European).

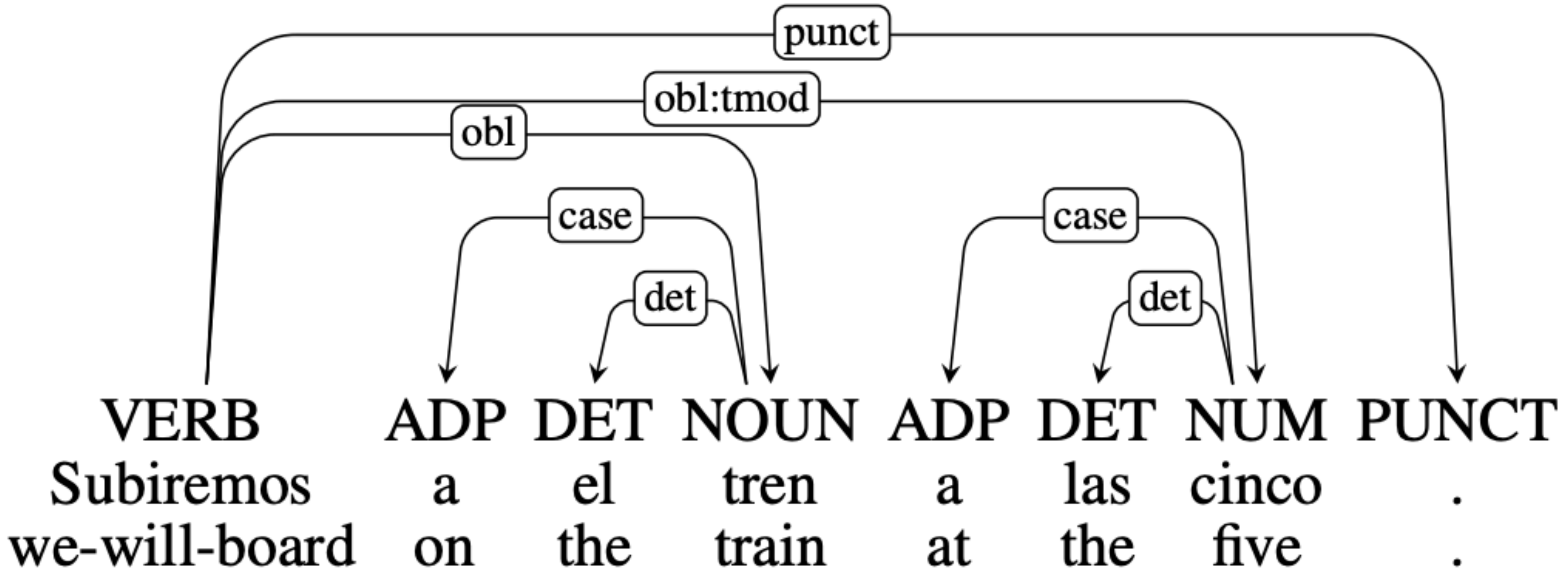
Abaza	1	<1K	Northwest Caucasian
Afrikaans	1	49K	IE, Germanic
Akkadian	2	23K	Afro-Asiatic, Semitic
Akuntsu	1	<1K	Tupian, Tupari
Albanian	1	<1K	IE, Albanian
Amharic	1	10K	Afro-Asiatic, Semitic
Ancient Greek	2	416K	IE, Greek
Apurina	1	<1K	Arawakan
Arabic	3	1,042K	Afro-Asiatic, Semitic
Armenian	1	52K	IE, Armenian
Assyrian	1	<1K	Afro-Asiatic, Semitic
Bambara	1	13K	Mande
Basque	1	121K	Basque
Belarusian	1	275K	IE, Slavic
Bhojpuri	2	6K	IE, Indic
Breton	1	10K	IE, Celtic
Bulgarian	1	156K	IE, Slavic
Buryat	1	10K	Mongolic
Cantonese	1	13K	Sino-Tibetan
Catalan	1	531K	IE, Romance
Chinese	5	285K	Sino-Tibetan
Chukchi	1	6K	Chukotko-Kamchatkan
Classical Chinese	1	233K	Sino-Tibetan
Coptic	1	48K	Afro-Asiatic, Egyptian
Croatian	1	199K	IE, Slavic
Czech	5	2,227K	IE, Slavic
Danish	2	100K	IE, Germanic
Dutch	2	306K	IE, Germanic
English	9	648K	IE, Germanic

https://universaldependencies.org/

Universal Dependencies



但我昨天才收到信 “But I didn’t receive the letter until yesterday”



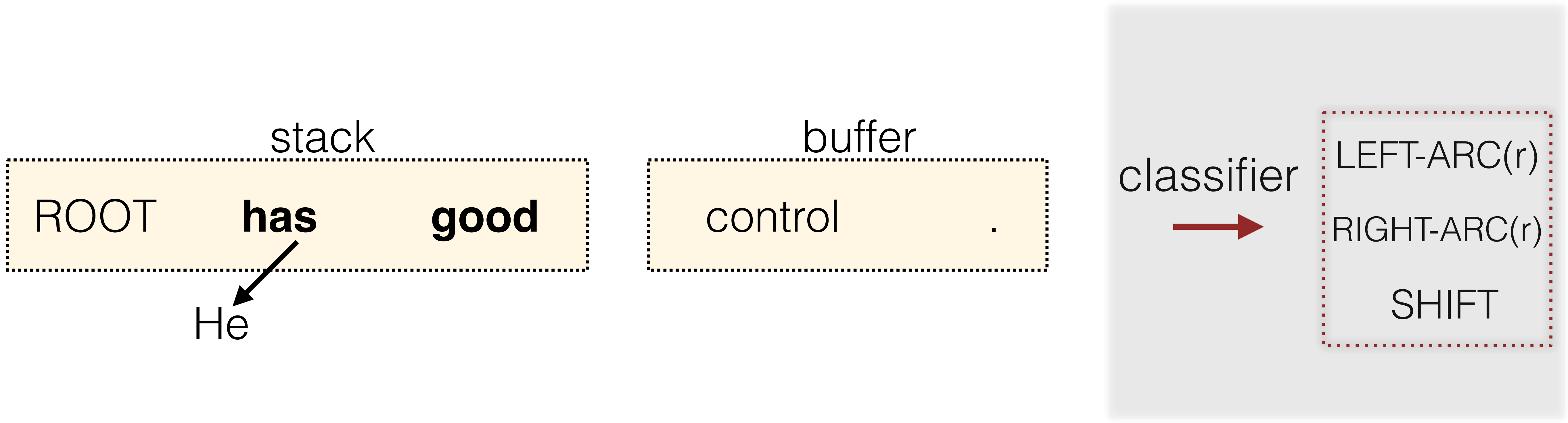
Subiremos al tren a las cinco. “We will be boarding the train at five.”

Train a classifier to predict transitions

- Given $\{x_i, y_i\}$ where x_i is a sentence and y_i is a dependency parse
- For each x_i with n words, we can construct a transition sequence of length $2n$ which generates y_i, so we can generate 2n training examples: $\{(c_k, t_k)\}$

  c_k: configuration, t_k: transition

  - "shortest stack" strategy: prefer LEFT-ARC over SHIFT.
- The goal becomes to learn a classifier that predicts t_k from c_k as input



(2|R| + 1) -way classification!

R: dependency labels

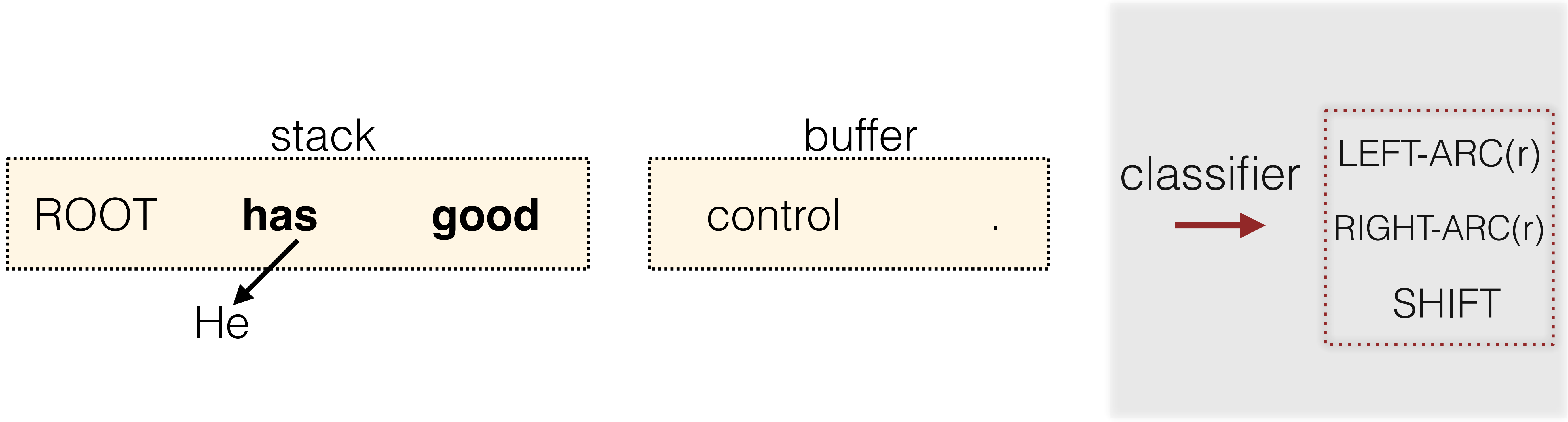
Train a classifier to predict transitions

During testing, we use the classifier to repeat predicting the transition, until we reach a terminal configuration

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← Classifier (state)        ; choose a transition operator to apply
    state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

Feature extraction



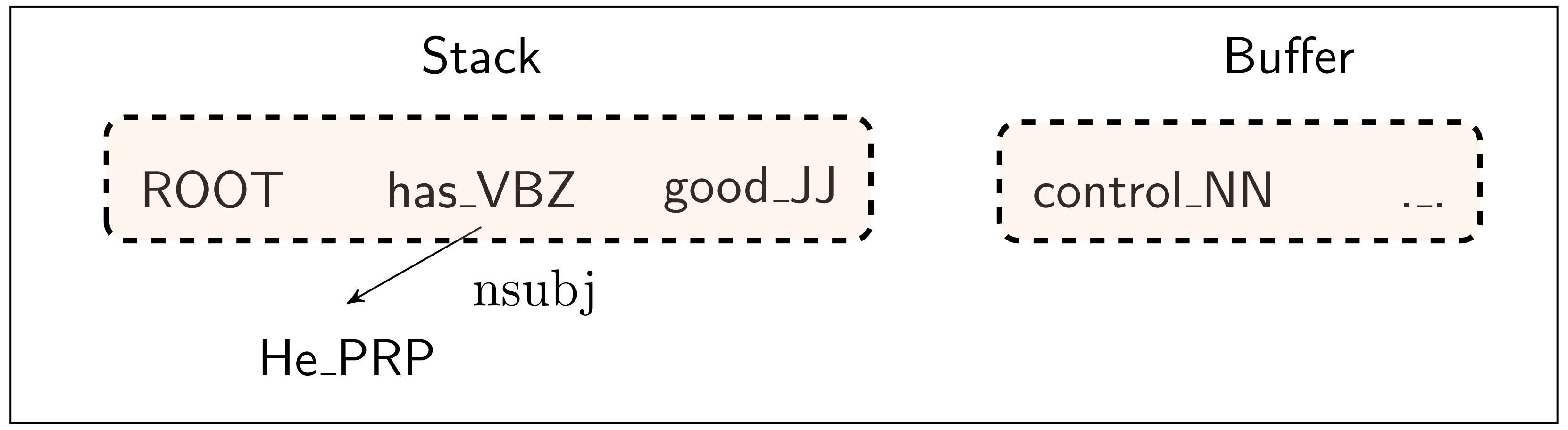
- Extract features from the configuration
- Use your favorite classifier: logistic regression, SVM, FFNNs, ...

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

w: word, t: part-of-speech tag $\circ$: concatenation

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

Feature extraction



w: words, t: part-of-speech tags

Feature templates

$s_2 . w \circ s_2 . t$

$s_1 . w \circ s_1 . t \circ b_1 . w$

Features

$s_2 . w = \text{has} \circ s_2 . t = \text{VBZ}$

$s_1 . w = \text{good} \circ s_1 . t = \text{JJ} \circ b_1 . w = \text{control}$

These days, we can use neural networks to automatically extract features!

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

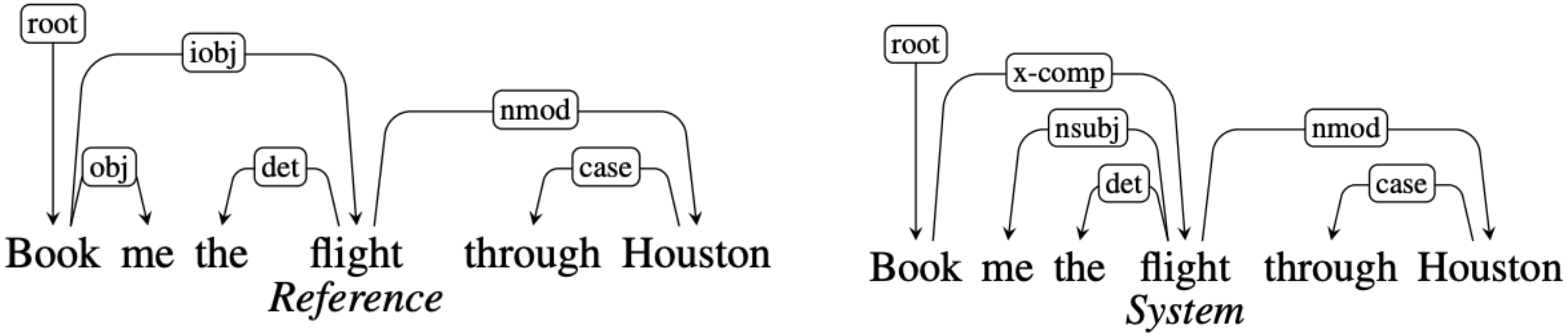
Evaluating dependency parsing

- Unlabeled attachment score (UAS)

  = percentage of words that have been assigned the correct head

- Labeled attachment score (LAS)

  = percentage of words that have been assigned the correct head & label



UAS = 5/6 LAS = 2/3

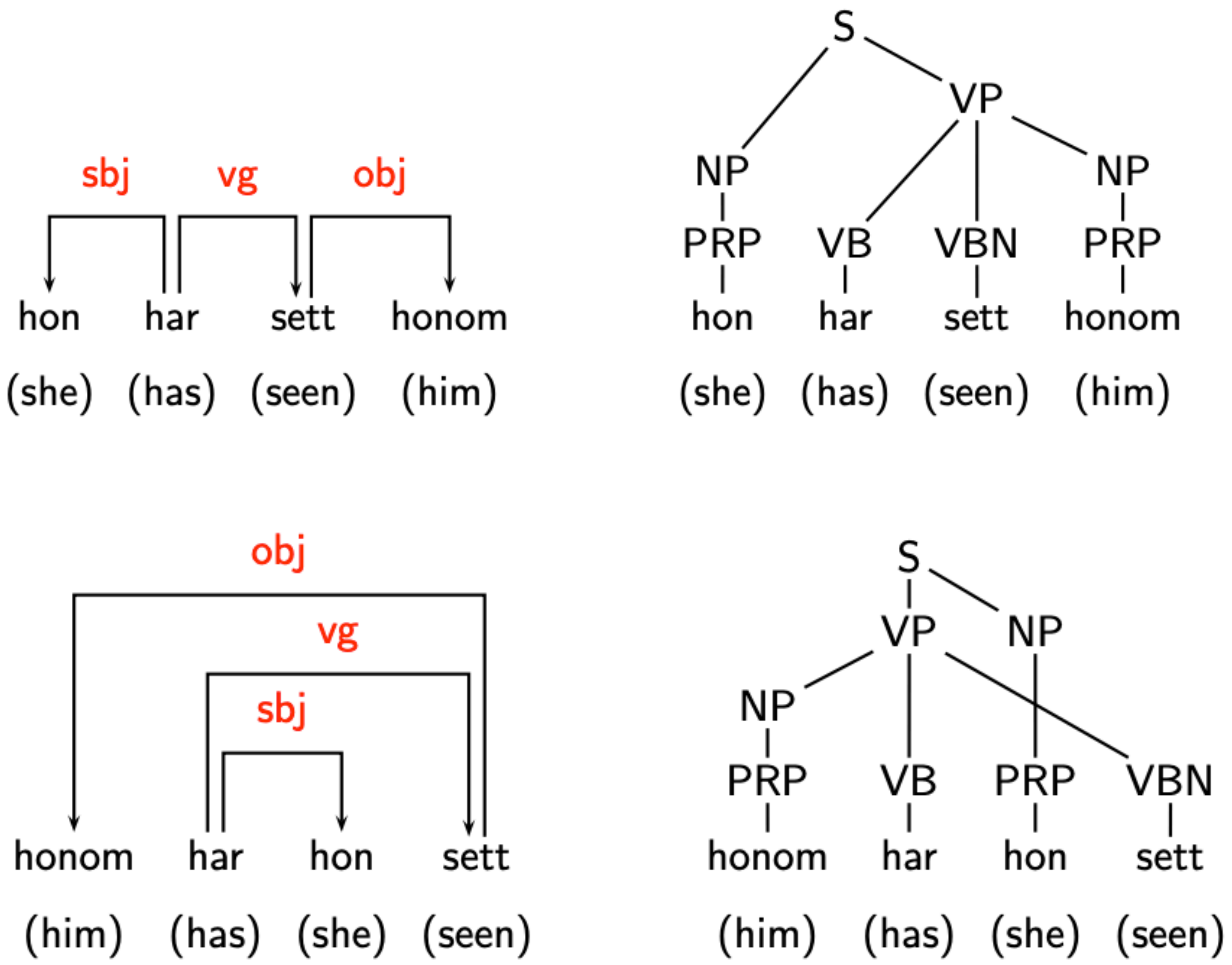
Evaluating dependency parsing

Parser		Test UAS	Test LAS
(Chen and Manning, 2014)	T	91.8	89.6
(Dyer et al., 2015)	T	93.1	90.9
(Ballesteros et al., 2016)	T	93.56	92.41
(Weiss et al., 2015)	T	94.26	91.42
(Andor et al., 2016)	T	94.61	92.79
(Ma et al., 2018) §	T	95.87	94.19
(Kiperwasser and Goldberg, 2016a) §	G	93.0	90.9
(Kiperwasser and Goldberg, 2016b)	G	93.1	91.0
(Wang and Chang, 2016)	G	94.08	91.82
(Cheng et al., 2016)	G	94.10	91.49
(Kuncoro et al., 2016)	G	94.26	92.06
(Zheng, 2017) §	G	95.53	93.94
(Dozat and Manning, 2017)	G	95.74	94.08

T: transition-based / G: graph-based

Advantages of dependency structure

- More suitable for free word order languages



Advantages of dependency structure

- More suitable for free word order languages

- The predicate-argument structure is more useful for some applications