SFU NatLangLab

CMPT 413/713: Natural Language Processing

Dependency Parsing

Spring 2024

2024-03-06

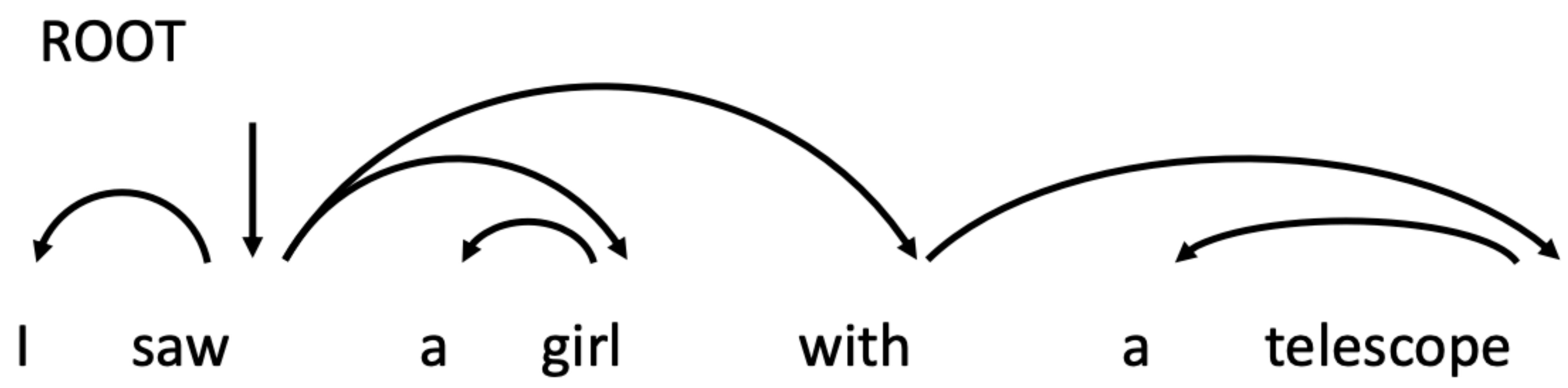
Adapted from slides from Danqi Chen and Karthik Narasimhan
(with some content from slides from Chris Manning and Graham Neubig)

Overview

- What is dependency parsing?
- Two families of algorithms
  - Transition-based dependency parsing
  - Graph-based dependency parsing

Dependency and constituency

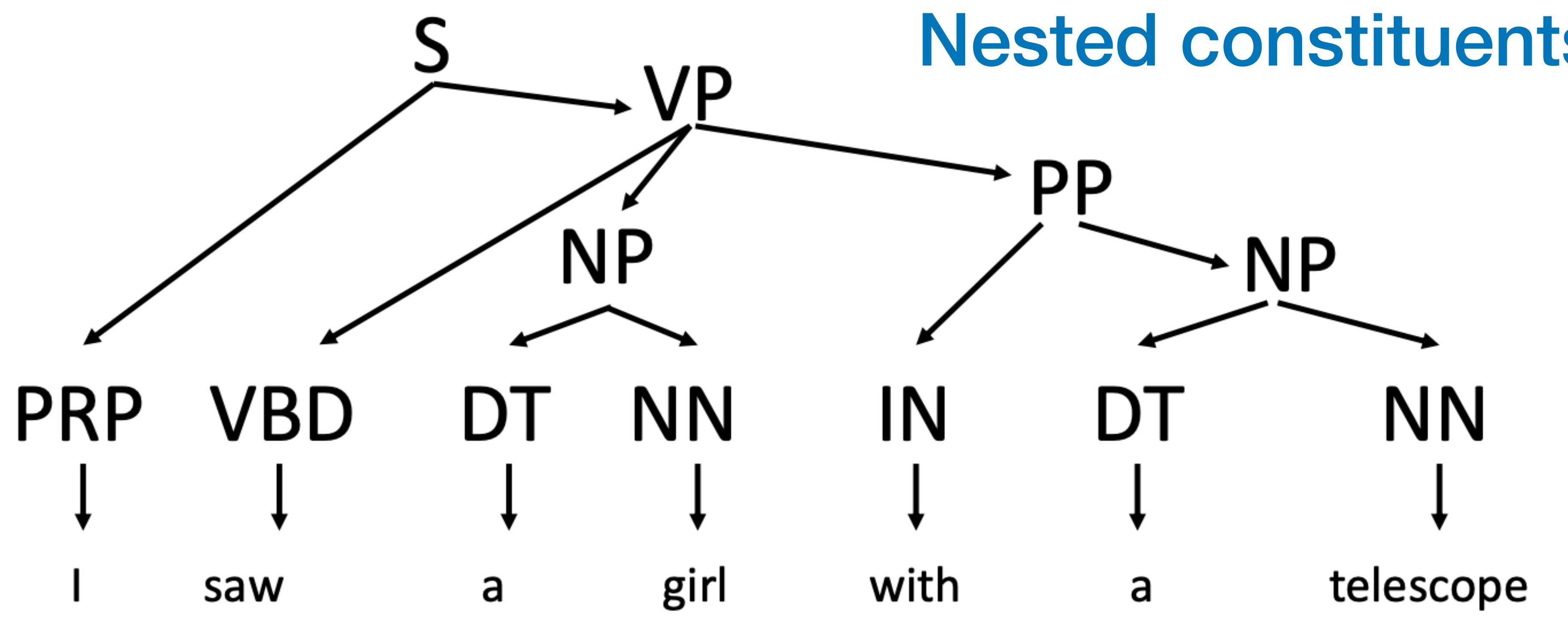
- **Dependency Trees** focus on relations between words



Words directly linked to each other

- **Phrase Structure** models the structure of a sentence

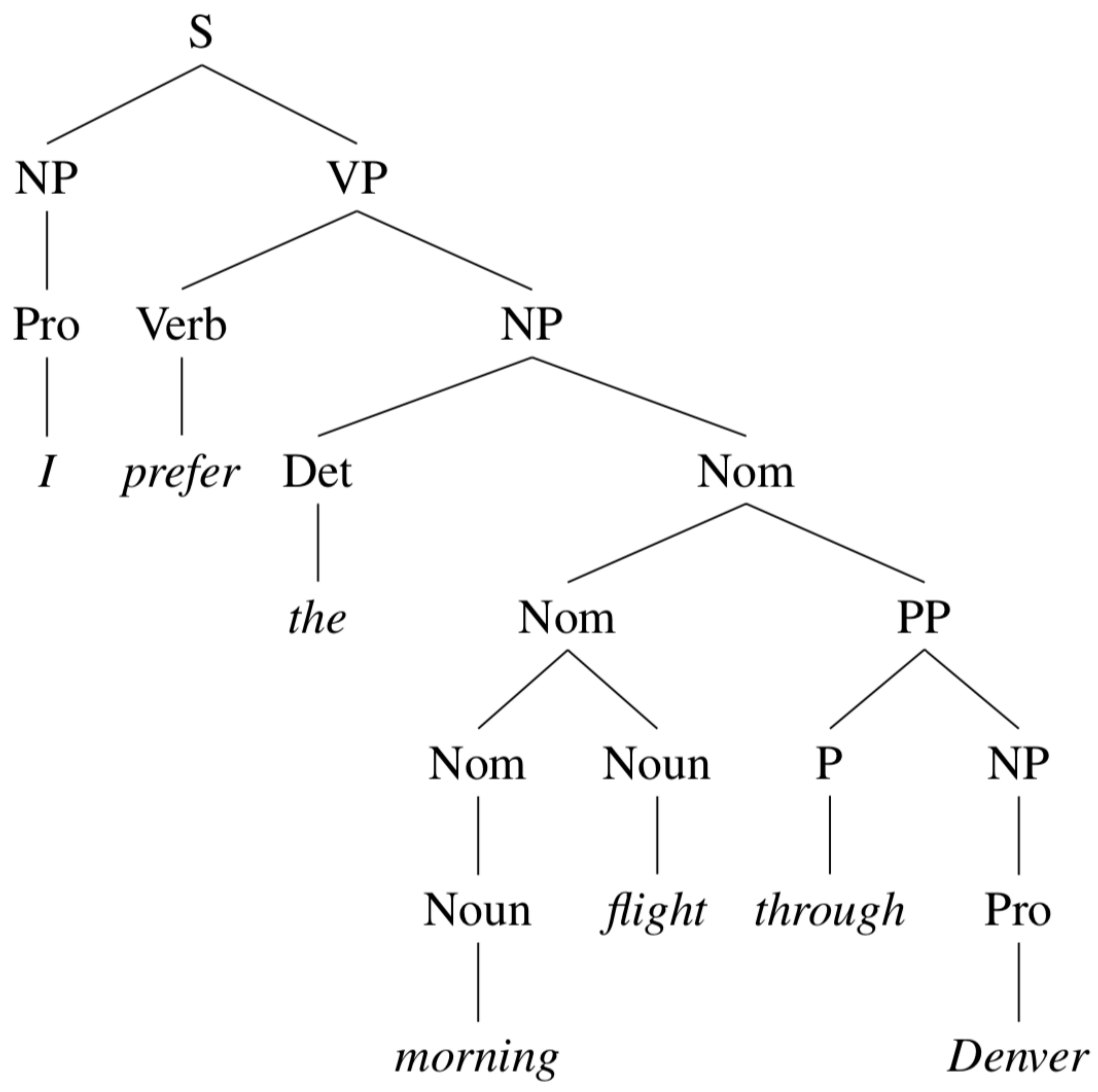
Nested constituents

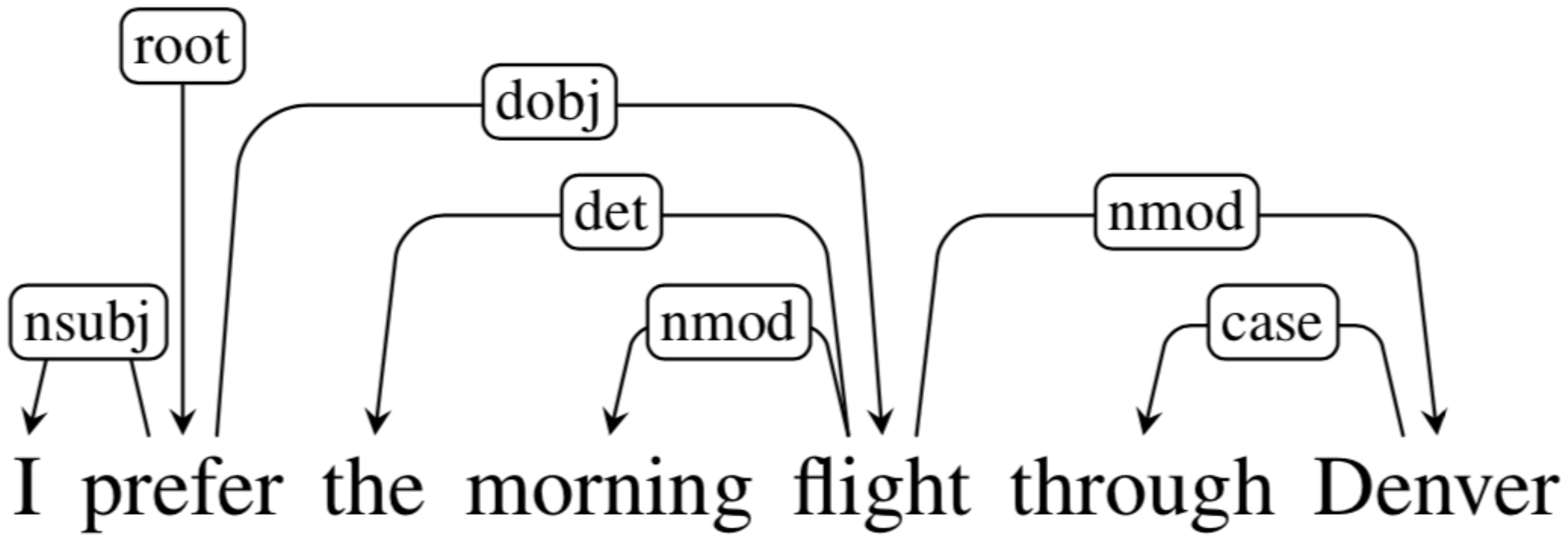


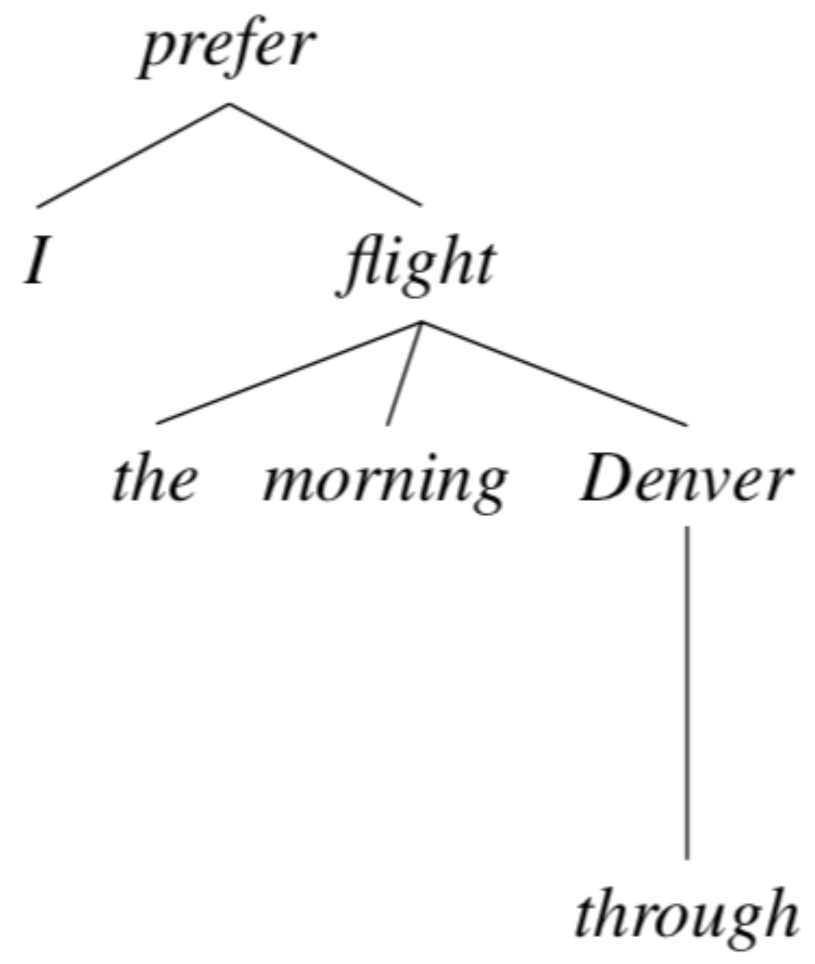
Constituency Parse generated from Context Free Grammars (CFGs)

(figure credit: CMU CS 11-747, Graham Neubig)

Constituency vs dependency structure







Pāṇini's grammar of Sanskrit (c. 5th century BCE)

Gallery: http://wellcomeimages.org/indexplus/image/L0032691.html

 File:Birch bark MS from Kashmir of the Rupavatra Wellcome L0032691.jpg

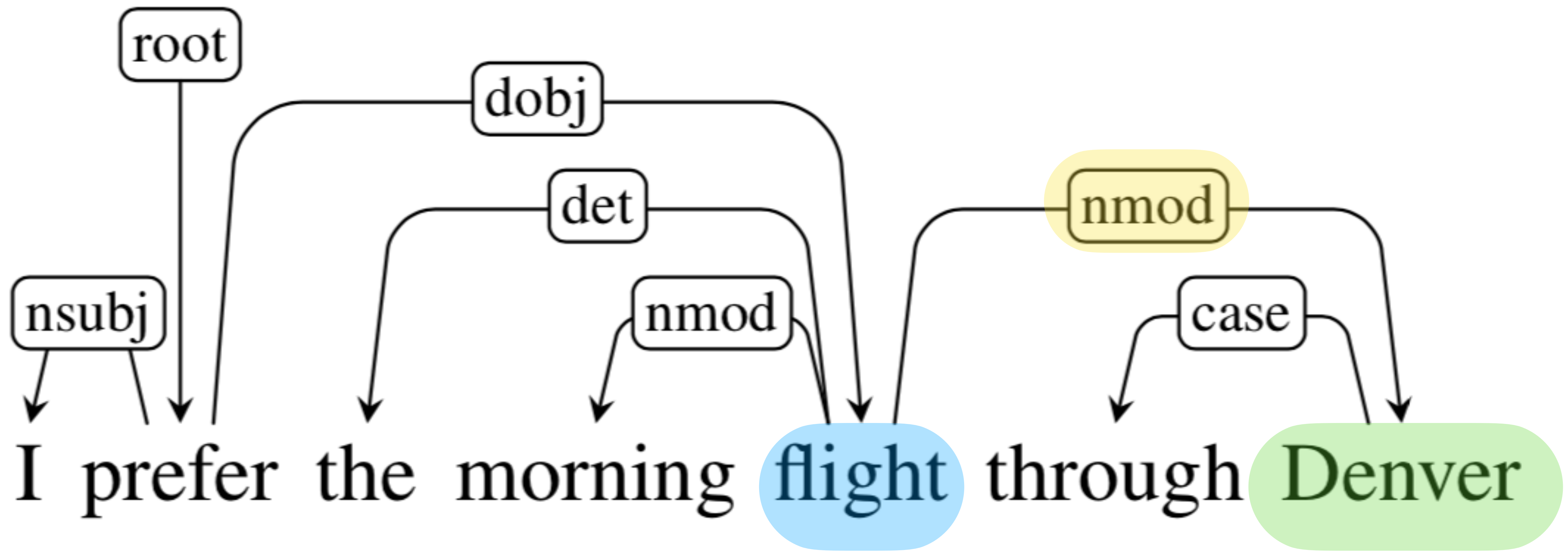
(slide credit: Stanford CS224N, Chris Manning)

Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammars is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947; then Chomsky)
- Modern dependency work often sourced to L. Tesnière (1959)
  - Was dominant approach in "East" in 20th Century (Russia, China, …)
  - Good for free-er word order languages
- Among the earliest kinds of parsers in NLP, even in the US:
  - David Hays, one of the founders of U.S. computational linguistics, built early (first?) dependency parser (Hays 1962)

(slide credit: Stanford CS224N, Chris Manning)

Dependency structure



- Consists of relations between lexical items, normally *binary*, *asymmetric* relations ("arrows") called **dependencies**
- The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc)

- The arrow connects a **head** (governor) and a **dependent** (modifier)
- Usually, dependencies form a tree (single-head, connected, acyclic)

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

(de Marneffe and Manning, 2008): Stanford typed dependencies manual

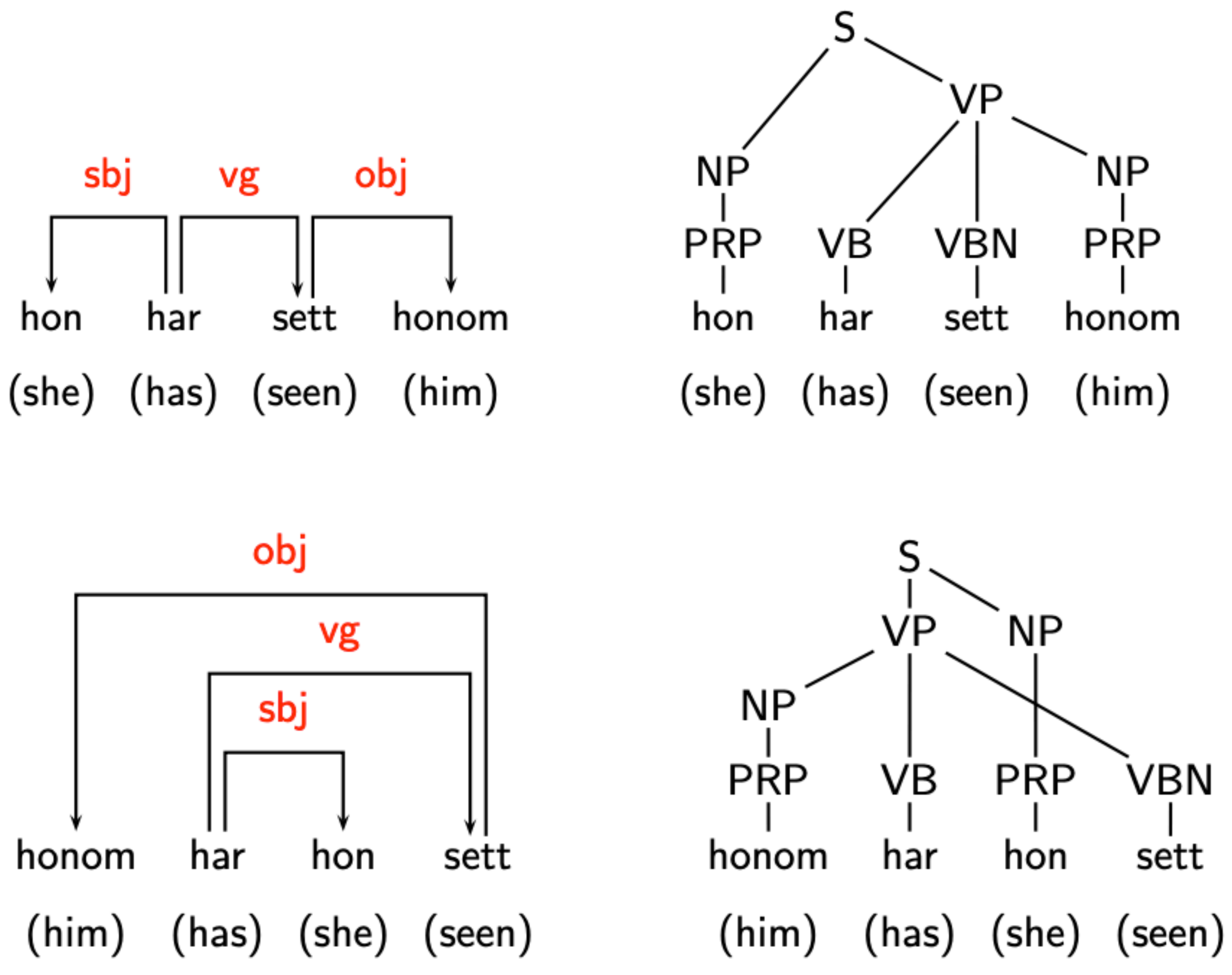
Dependency relations

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*.

(de Marneffe and Manning, 2008): Stanford typed dependencies manual

Advantages of dependency structure

- More suitable for free word order languages



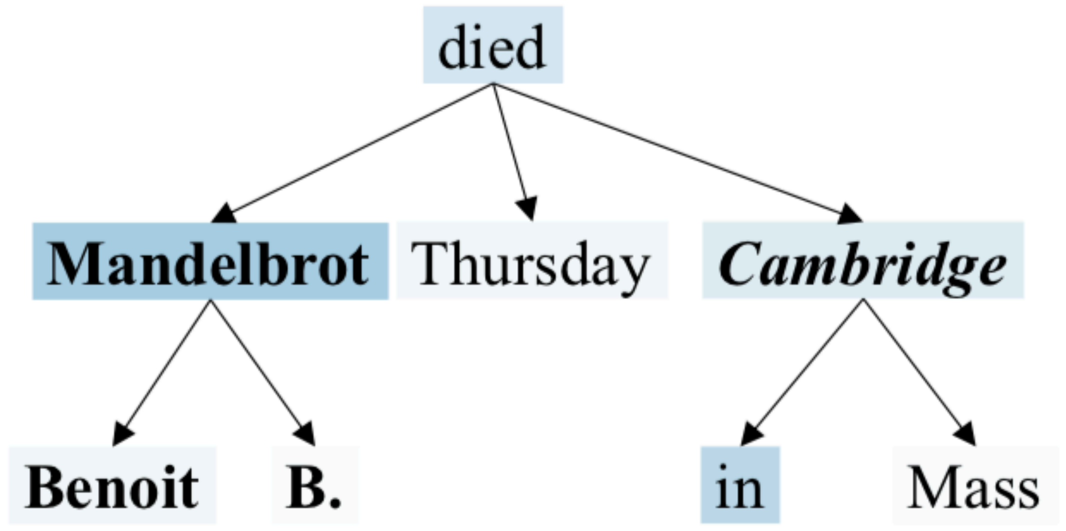
Advantages of dependency structure

- More suitable for free word order languages

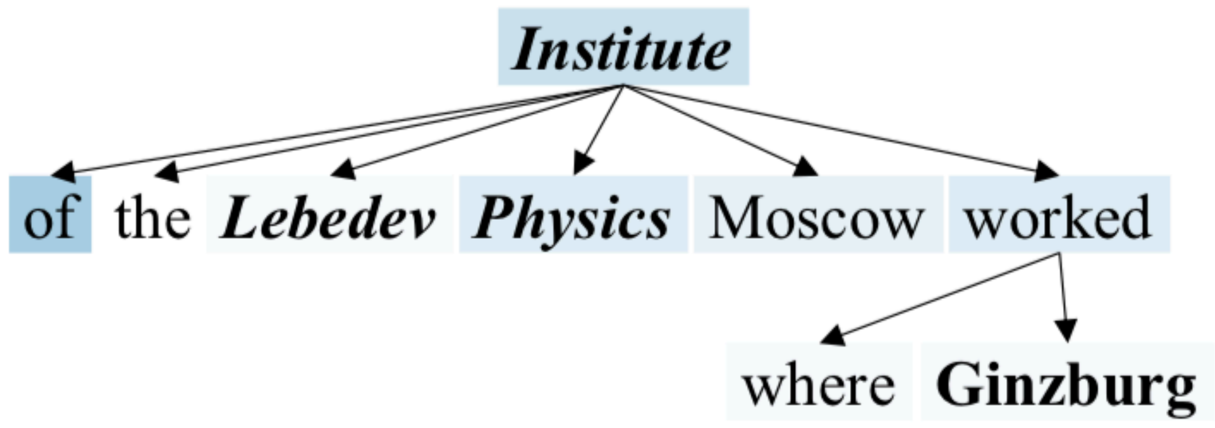
- The predicate-argument structure is more useful for many applications

Relation Extraction

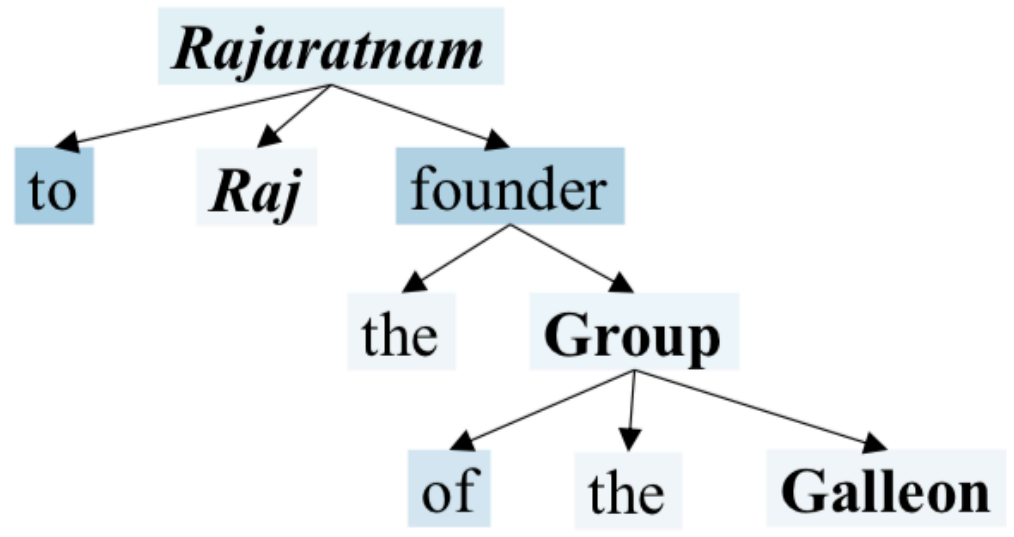
Relation: *per:city_of_death*
Benoit B. Mandelbrot, a maverick mathematician who developed an innovative theory of roughness and applied it to physics, biology, finance and many other fields, died Thursday in ***Cambridge***, Mass.



Relation: *per:employee_of*
In a career that spanned seven decades, Ginzburg authored several groundbreaking studies in various fields -- such as quantum theory, astrophysics, radio-astronomy and diffusion of cosmic radiation in the Earth's atmosphere -- that were of "Nobel Prize caliber," said Gennady Mesyats, the director of the ***Lebedev Physics Institute*** in Moscow, where **Ginzburg** worked .



Relation: *org:founded_by*
Anil Kumar, a former director at the consulting firm McKinsey & Co, pleaded guilty on Thursday to providing inside information to ***Raj Rajaratnam***, the founder of the **Galleon Group**, in exchange for payments of at least $ 175 million from 2004 through 2009.

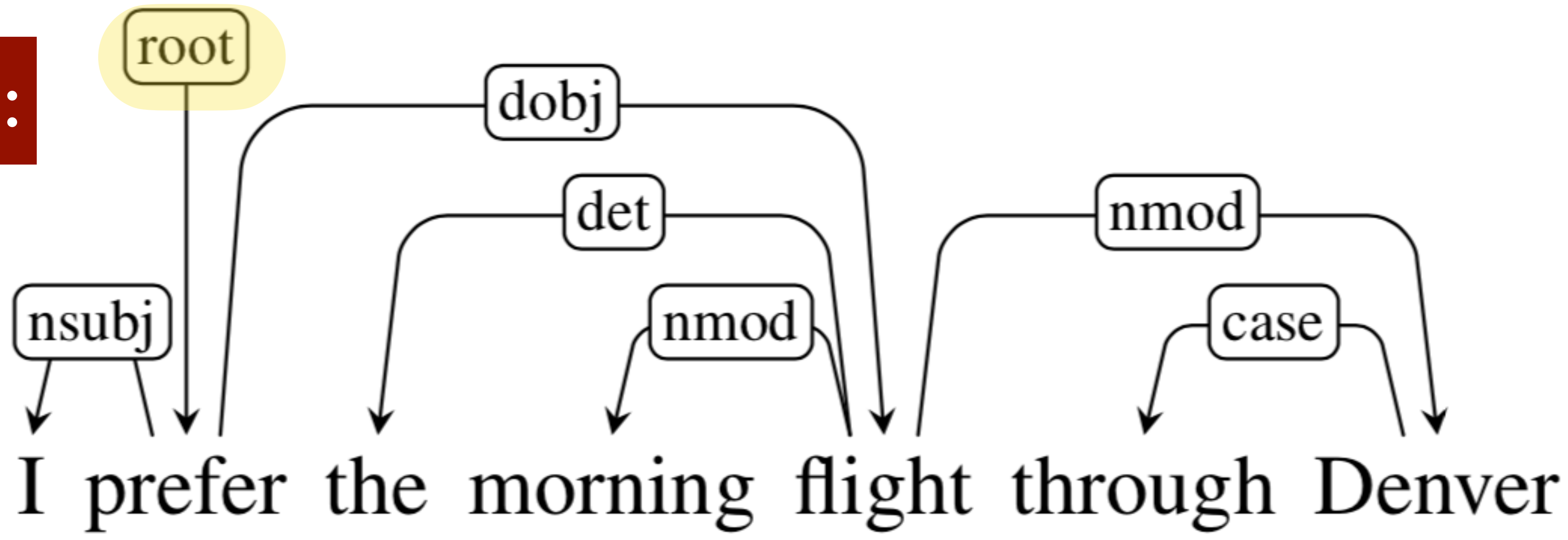


Dependency parsing

Input:

I prefer the morning flight through Denver

Output:



- A sentence is parsed by choosing for each word what other word is it a dependent of (and also the relation type)
- We usually add a fake ROOT at the beginning so every word has one head
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - No cycles: A —> B, B —> C, C —> A

Learning from data: treebanks!

Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [discussion → issues] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material

Dependencies rarely span intervening verbs or punctuation

4. Valency of heads

How many dependents on which side are usual for a head?

(slide credit: Stanford CS224N, Chris Manning)

Dependency treebanks

Stanford Dependencies (English)

- The major English dependency treebank: converting 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 (Multilingual)

- Universal Dependencies: more than 100 treebanks in 70 languages were 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 200 contributors producing more than 100 treebanks in over 70 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/

  - (De Marneffe et al, CL, 2021): Universal Dependencies

Universal Dependencies

https://universaldependencies.org/

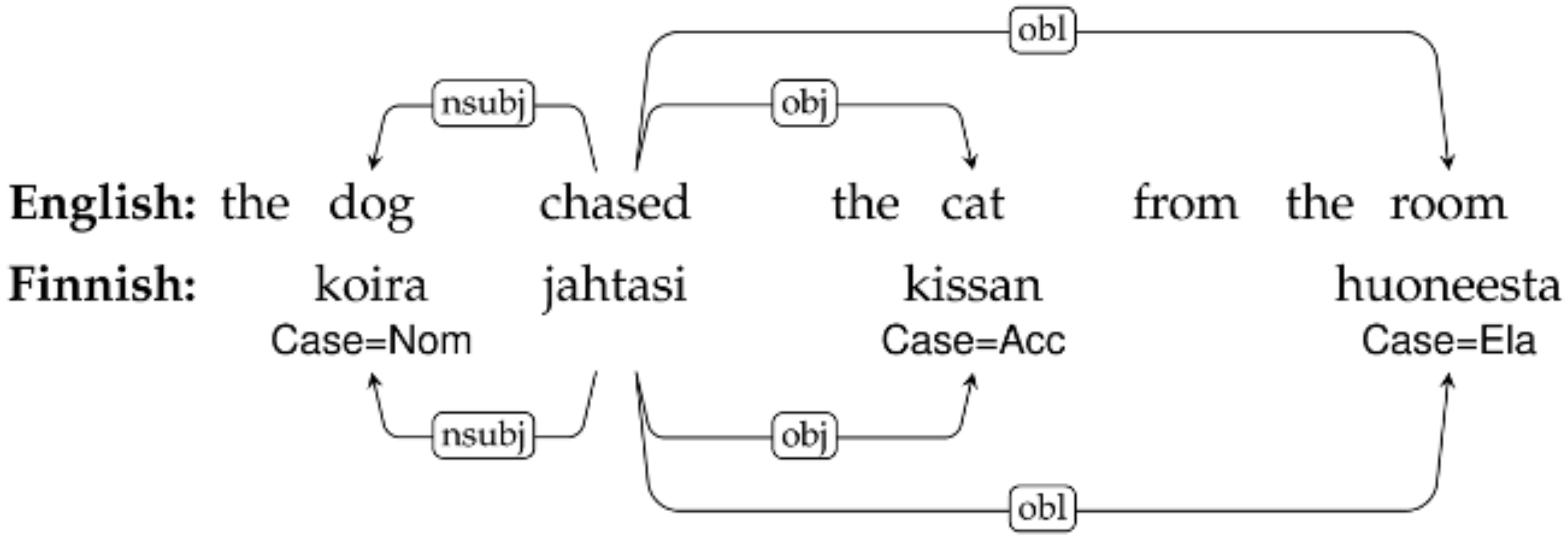
228 treebanks over 130 languages as of 2022

Afrikaans	1	49K	IE, Germanic
Akkadian	1	1K	Afro-Asiatic, Semitic
Amharic	1	10K	Afro-Asiatic, Semitic
Ancient Greek	2	416K	IE, Greek
Arabic	3	1,042K	Afro-Asiatic, Semitic
Armenian	1	36K	IE, Armenian
Assyrian	1	<1K	Afro-Asiatic, Semitic
Bambara	1	13K	Mande
Basque	1	121K	Basque
Belarusian	1	13K	IE, Slavic
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	161K	Sino-Tibetan
Classical Chinese	1	55K	Sino-Tibetan
Coptic	1	25K	Afro-Asiatic, Egyptian
Croatian	1	199K	IE, Slavic
Czech	5	2,222K	IE, Slavic
Danish	2	100K	IE, Germanic
Dutch	2	307K	IE, Germanic
English	6	603K	IE, Germanic
Erzya	1	15K	Uralic, Mordvin
Estonian	2	461K	Uralic, Finnic
Faroese	1	10K	IE, Germanic
Finnish	3	377K	Uralic, Finnic
French	8	1,156K	IE, Romance
Galician	2	164K	IE, Romance
German	4	3,409K	IE, Germanic
Gothic	1	55K	IE, Germanic
Greek	1	63K	IE, Greek
Hebrew	1	161K	Afro-Asiatic, Semitic
Hindi	2	375K	IE, Indic
Hindi English	1	26K	Code switching
Hungarian	1	42K	Uralic, Ugric
Indonesian	2	141K	Austronesian, Malayo-Sumbawan
Irish	1	23K	IE, Celtic
Italian	6	781K	IE, Romance
Japanese	5	1,688K	Japanese
Karelian	1	3K	Uralic, Finnic
Kazakh	1	10K	Turkic, Northwestern
Komi Zyrian	2	3K	Uralic, Permic
Korean	5	446K	Korean

Universal Dependencies

- Developing cross-linguistically consistent treebank annotation for many languages

- Goals:
  - Facilitating multilingual parser development
  - Cross-lingual learning
  - Parsing research from a language typology perspective.



Universal Dependencies

Manning's Law:

- UD needs to be satisfactory for analysis of individual languages.
- UD needs to be good for linguistic typology.
- UD must be suitable for rapid, consistent annotation.
- UD must be suitable for computer parsing with high accuracy.
- UD must be easily comprehended and used by a non-linguist.
- UD must provide good support for downstream NLP tasks.

Universal POS tags, features, and relations

https://universaldependencies.org/guidelines.html

- Small set of universal POS tags with
- Separate set of universal features to specify lexical and grammatical properties

POS tags

Open class words	Closed class words	Other
ADJ	ADP	PUNCT
ADV	AUX	SYM
INTJ	CCONJ	X
NOUN	DET	
PROPN	NUM	
VERB	PART	
	PRON	
	SCONJ	

Features

Lexical features*	Inflectional features*	
	*Nominal**	*Verbal**
PronType	Gender	VerbForm
NumType	Animacy	Mood
Poss	NounClass	Tense
Reflex	Number	Aspect
Foreign	Case	Voice
Abbr	Definite	Evident
Typo	Degree	Polarity
		Person
		Polite
		Clusivity

Universal POS tags, features, and relations

https://universaldependencies.org/guidelines.html

- 37 universal syntactic relations
  - Individual languages may have more specific relations

Structural category of dependent

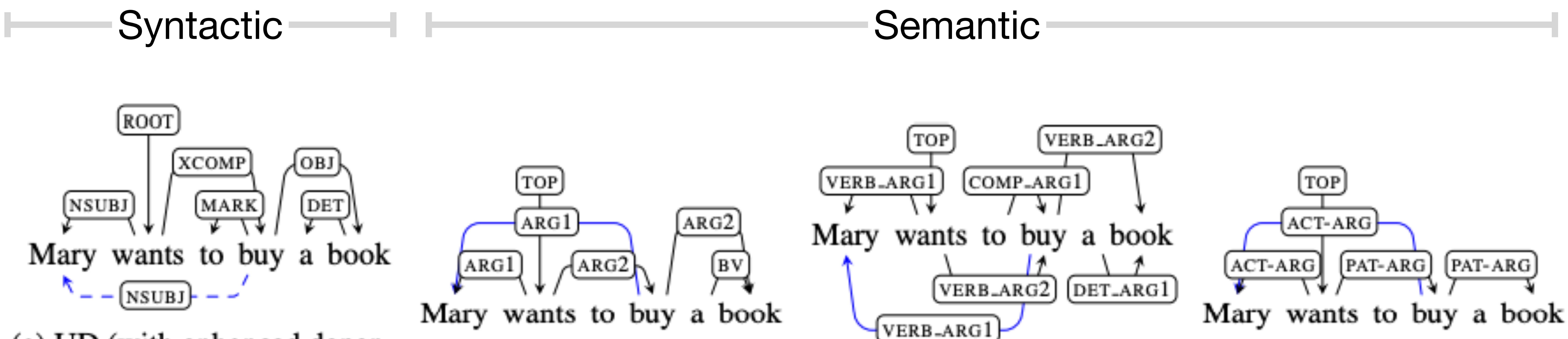
Functional relation to head

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

Other relations

Other types of dependency parses

- There can be other types of dependencies
- UD is a Syntactic dependency (designed to be easy to use)



(a) UD (with enhanced dependencies dashed) (b) DM (c) PAS (d) PSD

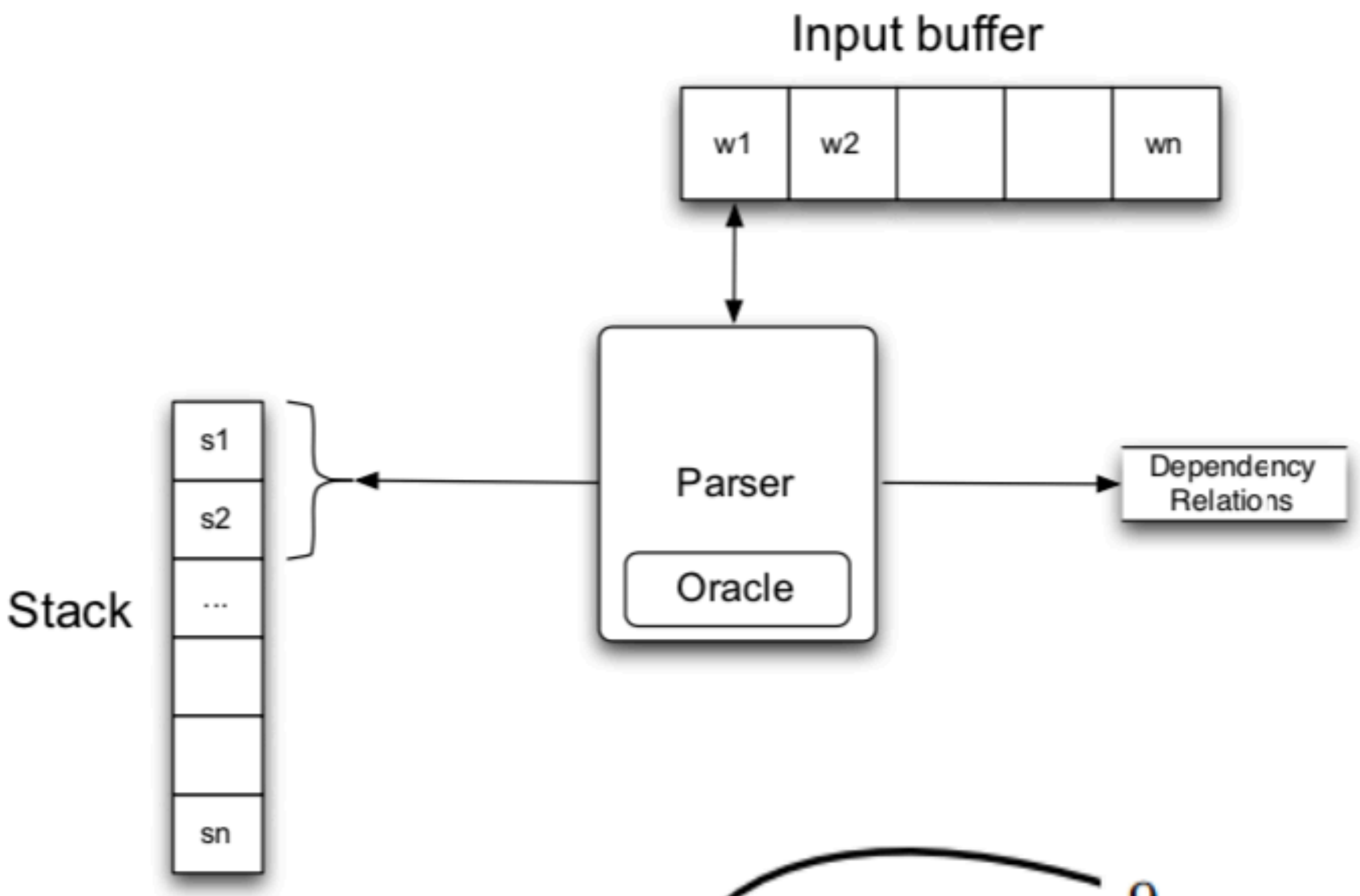
Simpler but More Accurate Semantic Dependency Parsing
https://arxiv.org/pdf/1807.01396.pdf (Dozat and Manning, ACL 2018)

Algorithms for dependency parsing

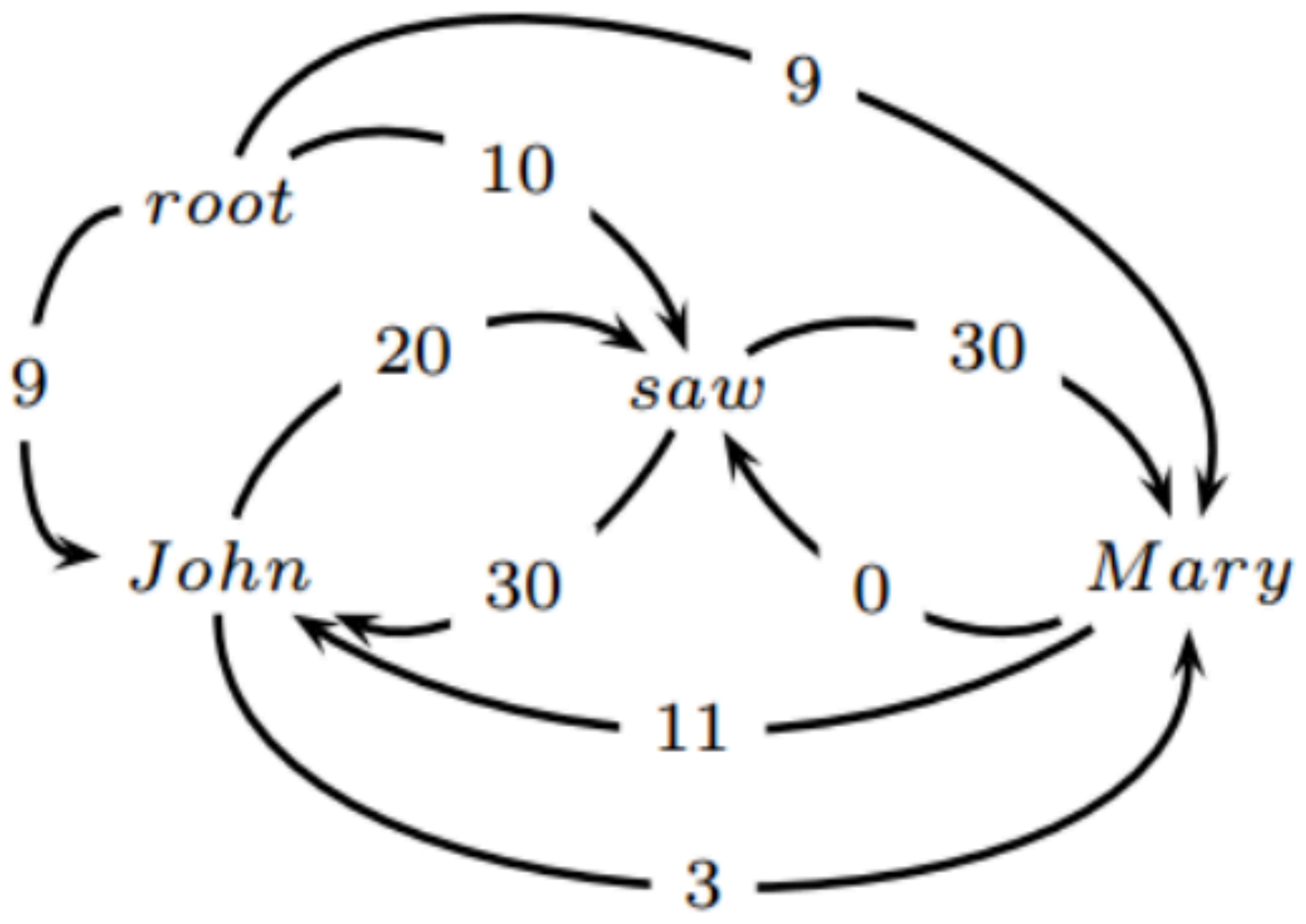
Two families of algorithms

Transition-based dependency parsing

- Also called "shift-reduce parsing"



Graph-based dependency parsing



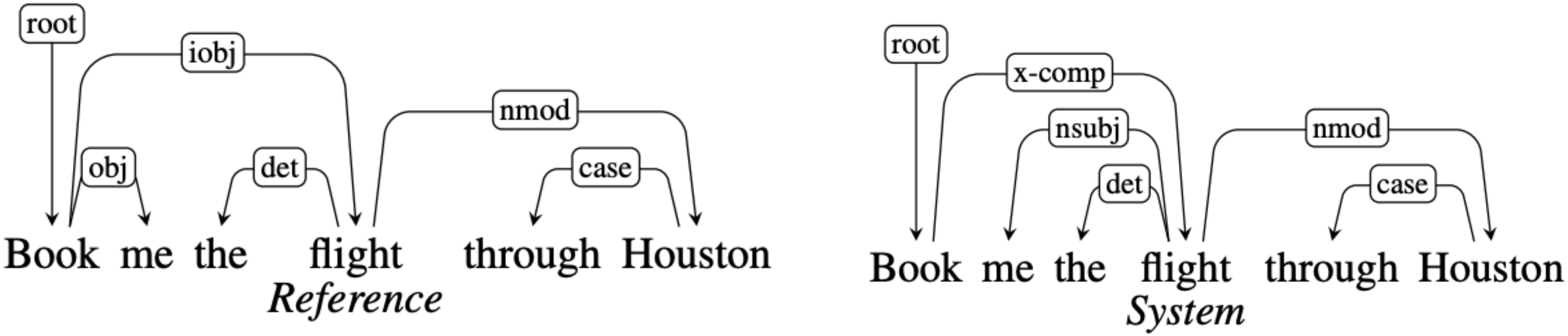
Two families of algorithms

	Parser	UAS	LAS
Transition-Based	Chen and Manning (2014)	91.8	89.6
	Dyer et al. (2015)	93.1	90.9
	Weiss et al. (2015)	93.99	92.05
	Ballesteros et al. (2016)	93.56	91.42
	Kiperwasser and Goldberg (2016)	93.9	91.9
	Alberti et al. (2015)	94.23	92.36
	Qi and Manning (2017)	94.3	92.2
	Fernández-G and Gómez-R (2018)	94.5	92.4
	Andor et al. (2016)	94.61	92.79
	Ma et al. (2018)*	95.87	94.19
	This work*	**96.04**	**94.43**
Graph-Based	Kiperwasser and Goldberg (2016)	93.1	91.0
	Wang and Chang (2016)	94.08	91.82
	Cheng et al. (2016)	94.10	91.49
	Kuncoro et al. (2016)	94.26	92.06
	Zhang et al. (2017)	94.30	91.95
	Ma and Hovy (2017)	94.88	92.96
	Dozat and Manning (2016)	95.74	94.08
	Ma et al. (2018)*	95.84	94.21

Left-to-Right Dependency Parsing with Pointer Networks
https://aclanthology.org/N19-1076.pdf (Fernandez-Gonzalez and Gomez-Rodriguez, NAACL 2019)

Evaluation

- 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

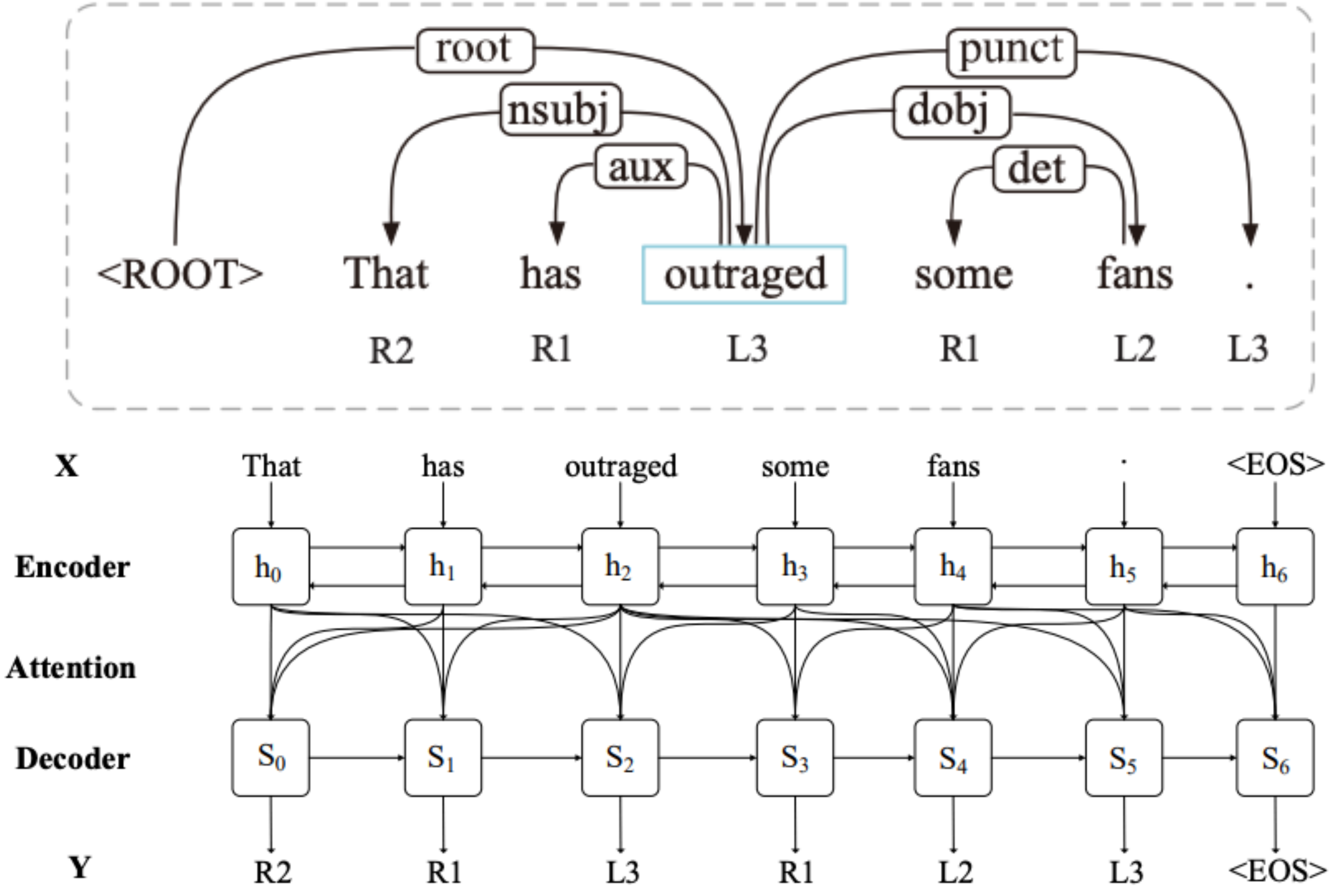


Reference *System*

UAS = ?   LAS = ?

Parsing as sequence modelling

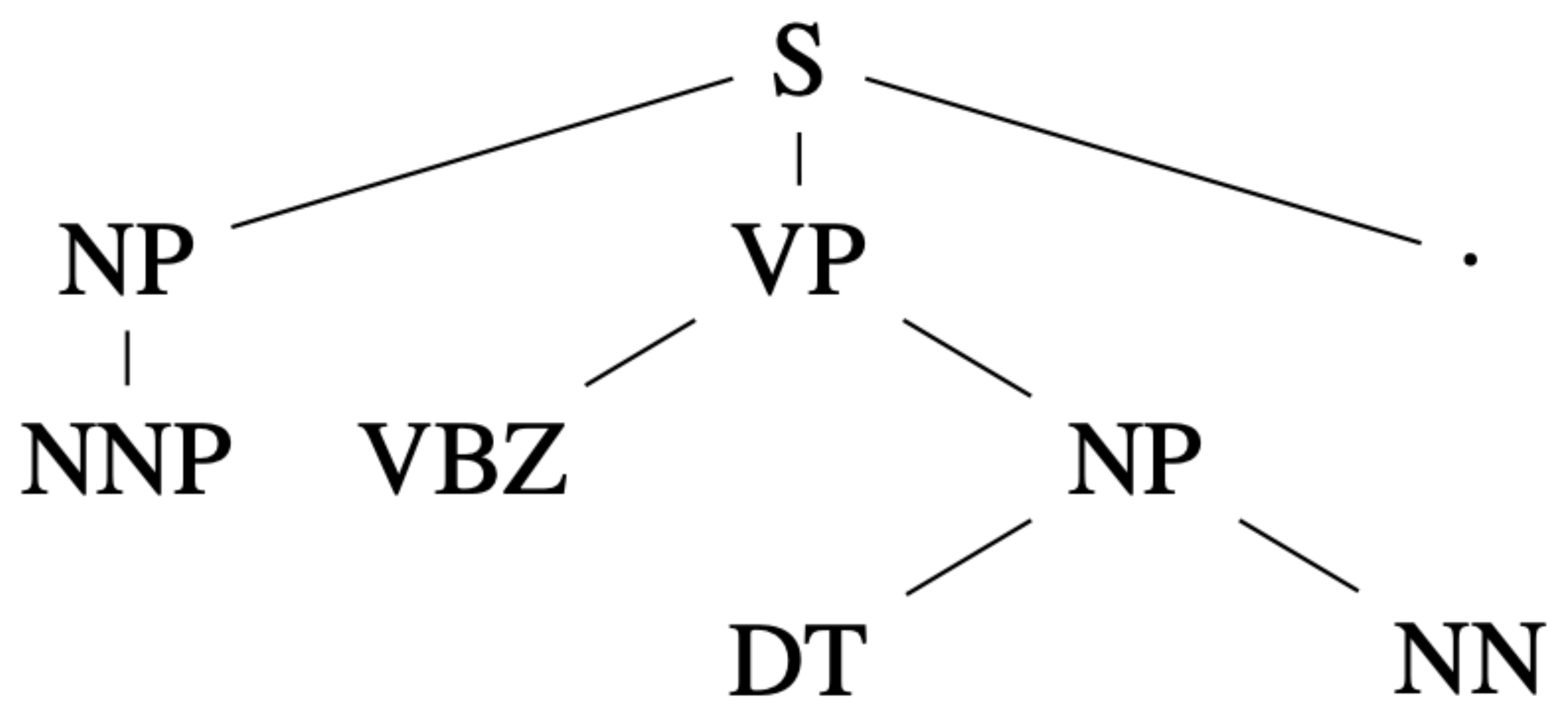
Seq2Seq



Seq2seq Dependency Parsing
https://aclanthology.org/C18-1271.pdf (Li et al, ICCL 2018)

Constituency parsing as Seq2Seq (Vinyals et al, 2015; Vaswani et al, 2017)

John has a dog . $\rightarrow$



John has a dog . $\rightarrow$ (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$

May not be structural correct
(i.e. unbalanced parenthesis)

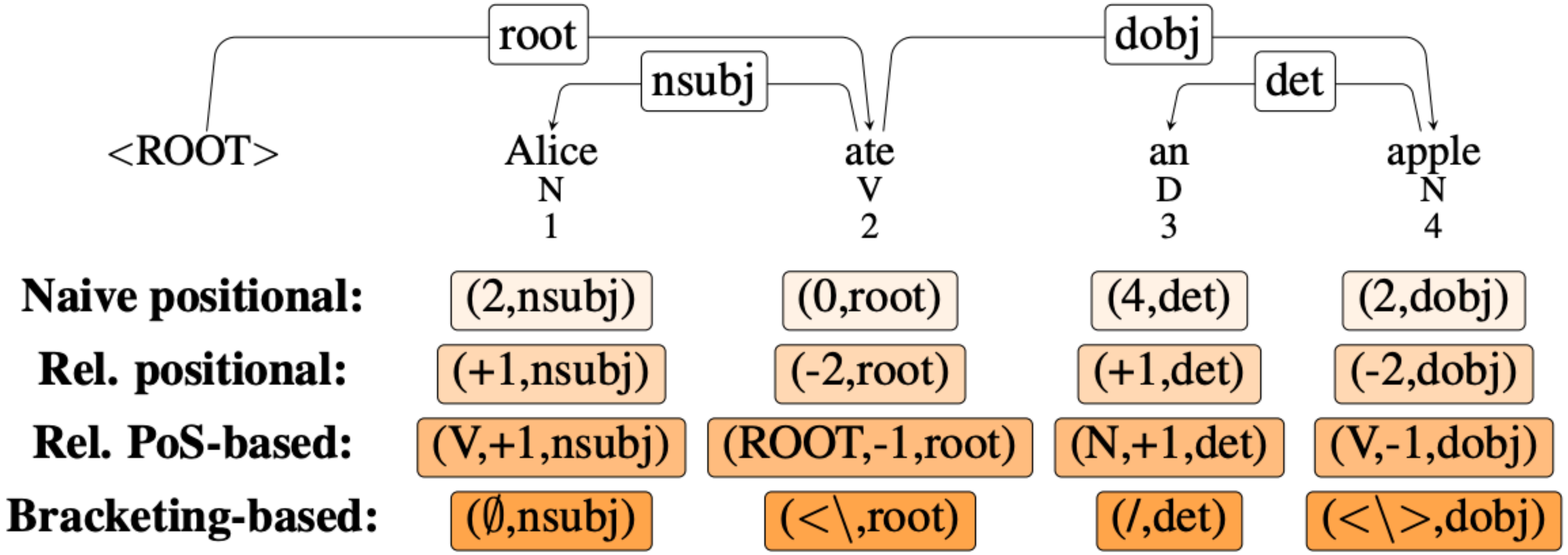
88.3 F1

91.3 F1

- Linearize parse tree and train LSTM seq2seq model with attention
- With transformers

Parsing as sequence modelling

Sequence labeling



Viable Dependency Parsing as Sequence Labeling
https://aclanthology.org/N19-1077.pdf (Strzyz, NAACL 2019)

Parsing as sequence modelling

Faster, with lower performance than some graph based methods

Encoding	UAS	LAS
Li et al. (2018) (sequence labeling)	87.58	83.81
Li et al. (2018) (seq2seq)	89.16	84.99
Li et al. (2018) (seq2seq+beam+subroot)	93.84	**91.86**
Naive positional	45.41	42.65
Rel. positional	91.05	88.67
Rel. PoS-based	**93.99**	91.76
Bracketing-based	93.45	91.17

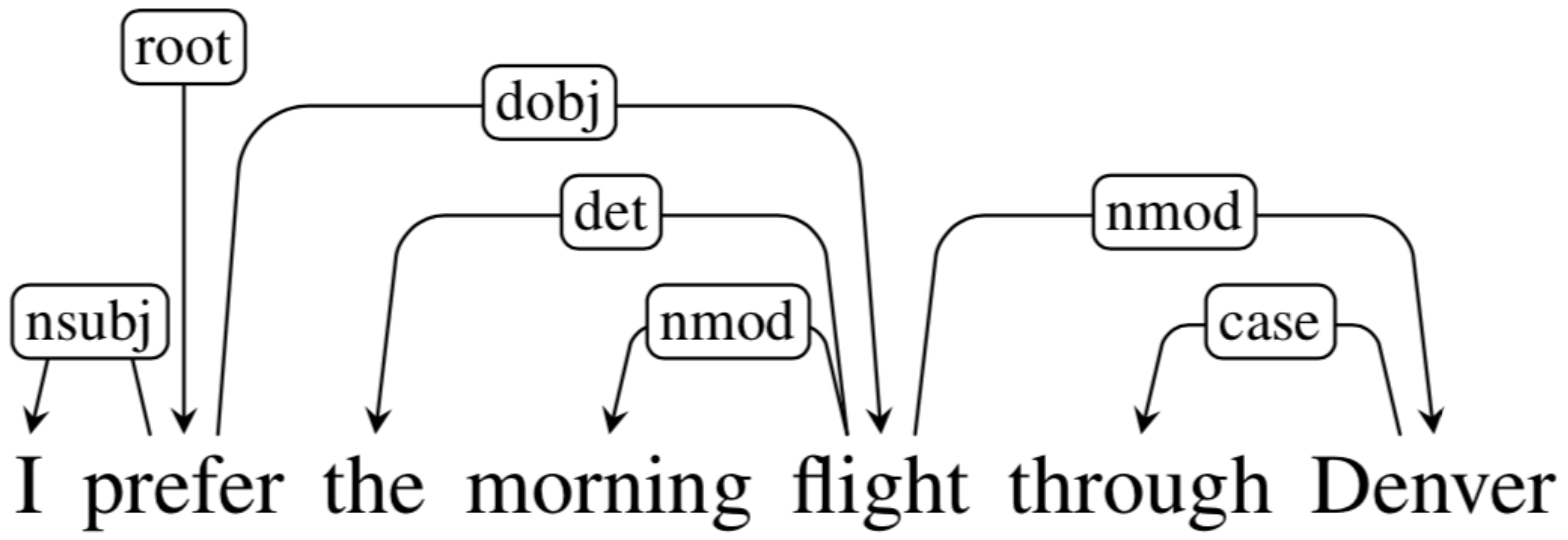
Sequence labeling variants

Model	sent/s		UAS	LAS
	CPU	GPU		
$P_{2,250}$	$267_{\pm 1}$	$777_{\pm 24}$	92.95	90.96
$P^{C}_{2,400}$	$165_{\pm 1}$	$700_{\pm 5}$	93.34	91.34
$P^{C}_{2,800}$	$101_{\pm 2}$	$648_{\pm 20}$	**93.67**	**91.72**
$B_{2,250}$	$310_{\pm 30}$	$730_{\pm 53}$	92.64	90.59
KG (transition-based)	$76_{\pm 1}$		93.90	91.90
KG (graph-based)	$80_{\pm 0}$		93.10	91.00
CM	$654^{\diamond}$		91.80	89.60
DM		$411^{\diamond}$	95.74	94.08
Ma et al. (2018)		$10_{\pm 0}$	95.87	94.19

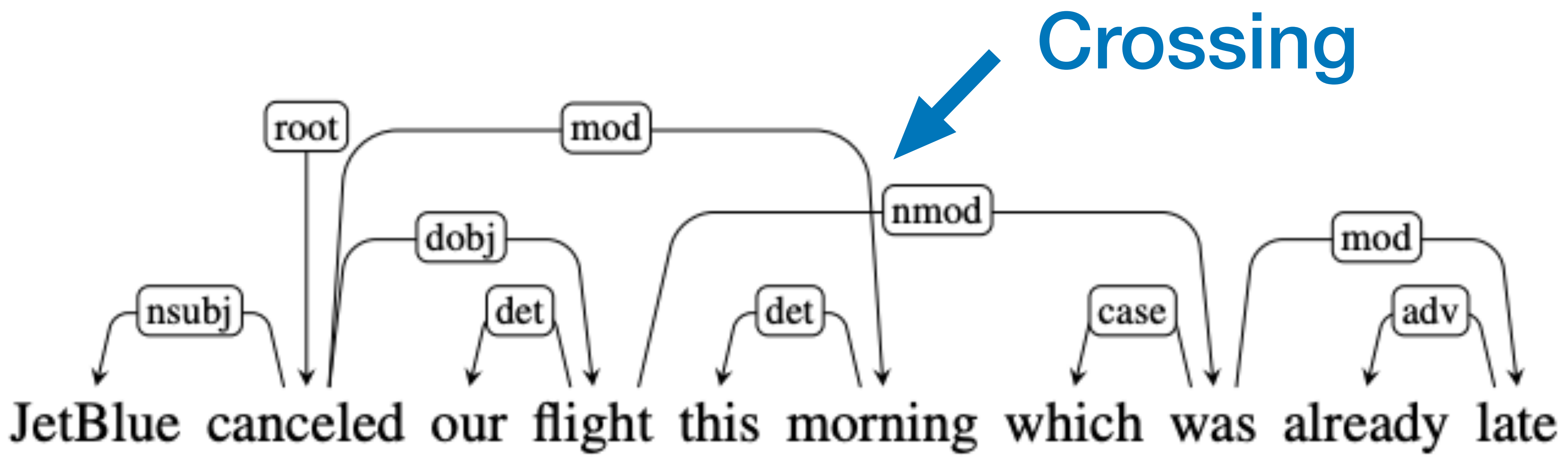
Viable Dependency Parsing as Sequence Labeling
https://aclanthology.org/N19-1077.pdf (Strzyz, NAACL 2019)

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.

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

This class: focuses on projective parsing

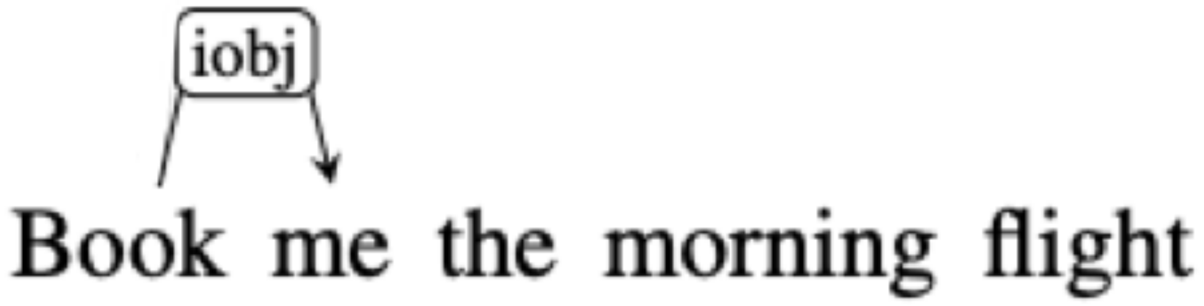
Transition-based dependency parsing

- The parsing process is modeled as a sequence of transitions
- A configuration consists of a stack s, a buffer b and a set of dependency arcs A: $c = (s, b, A)$

Stack: Can add arcs to 1st two words on stack

Buffer: Unprocessed words

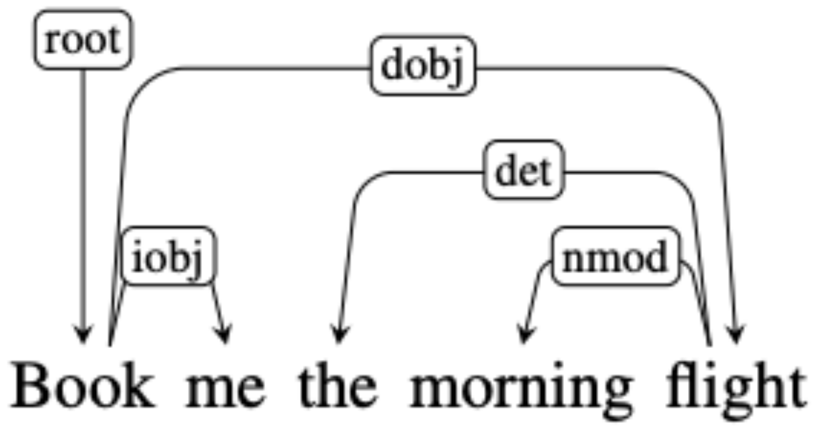
Current graph:



Transition-based dependency parsing

- The parsing process is modeled as a sequence of transitions
- A configuration consists of a stack s, a buffer b and a set of dependency arcs A: $c = (s, b, A)$
- Initially, $s = [\text{ROOT}], b = [w_1, w_2, \ldots, w_n], A = \emptyset$
- Three types of transitions (s_1, s_2: the top 2 words on the stack; b_1: the first word in the buffer)
    - LEFT-ARC (r): add an arc ($s_1 \xrightarrow{r} s_2$) to A, remove s_2 from the stack
    - RIGHT-ARC (r): add an arc ($s_2 \xrightarrow{r} s_1$) to A, remove s_1 from the stack
    - SHIFT: move b_1 from the buffer to the stack
- A configuration is terminal if $s = [\text{ROOT}]$ and $b = \emptyset$

This is called "Arc-standard"; There are other transition schemes…

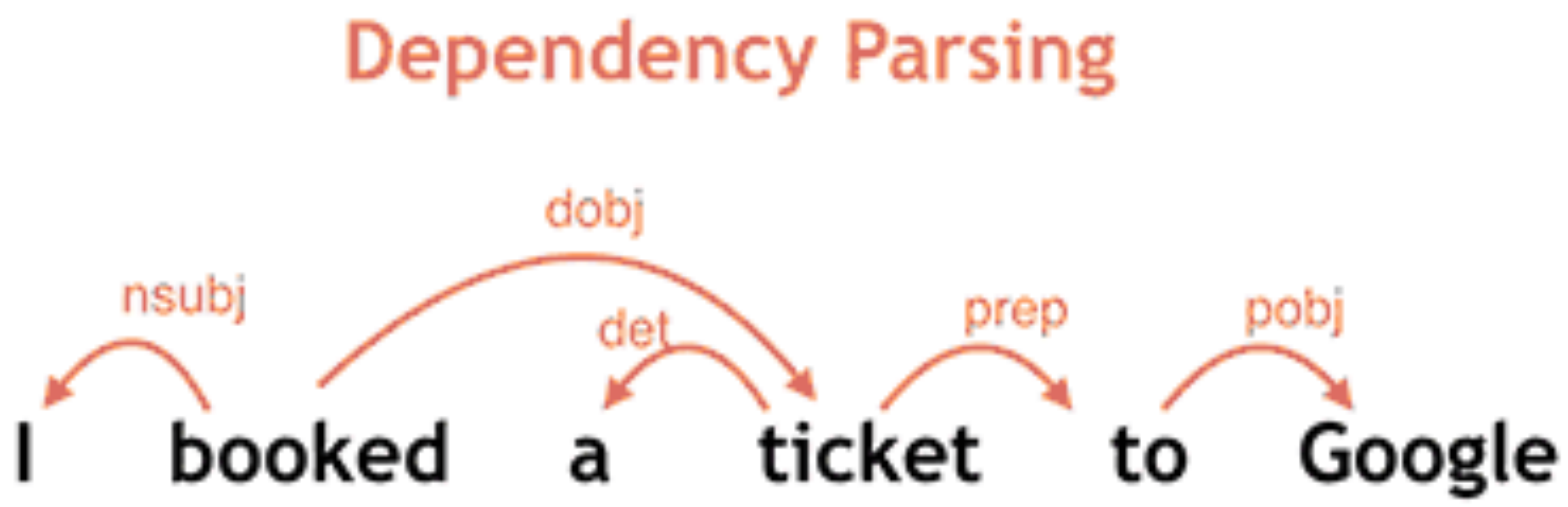


“Book me the morning flight”

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



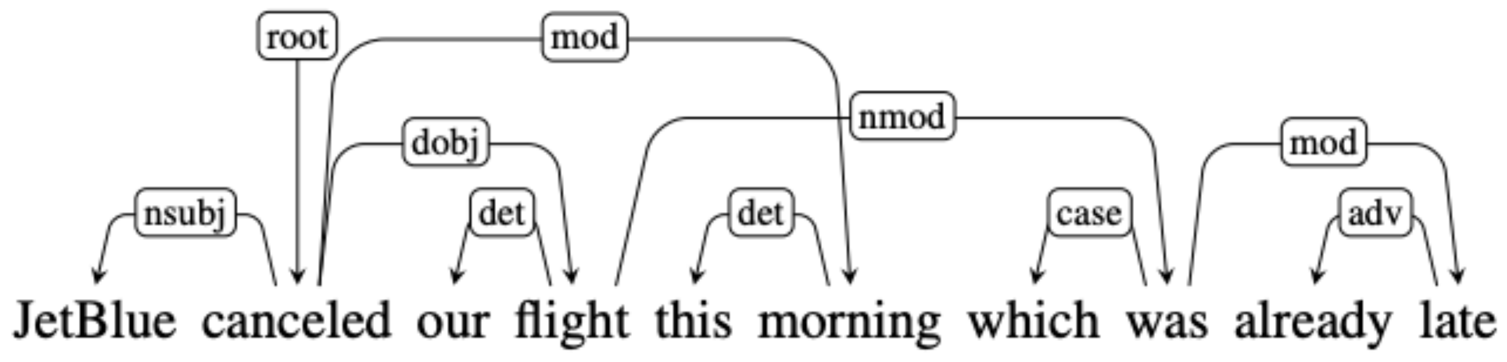
https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

Transition-based dependency parsing

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

Correctness:

- For every complete transition sequence, the resulting graph is a projective dependency forest (soundness)
- For every projective dependency forest G, there is a transition sequence that generates G (completeness)



- However, one parse tree can have multiple valid transition sequences. Why?
  - “He likes dogs”
    - Stack = [ROOT He likes]
    - Buffer = [dogs]
    - Action = ??

Train a classifier to predict actions!

- 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: $\{(c_k, a_k)\}$ c_k: configuration, a_k: action
  - "shortest stack" strategy: prefer LEFT-ARC over SHIFT.

> Given this information, the oracle chooses transitions as follows:
>
> LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$
>
> RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t.(S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$
>
> SHIFT: **otherwise**

- The goal becomes **how to learn a classifier from c_i to a_i**

How many training examples? How many classes?

Train a classifier to predict actions!

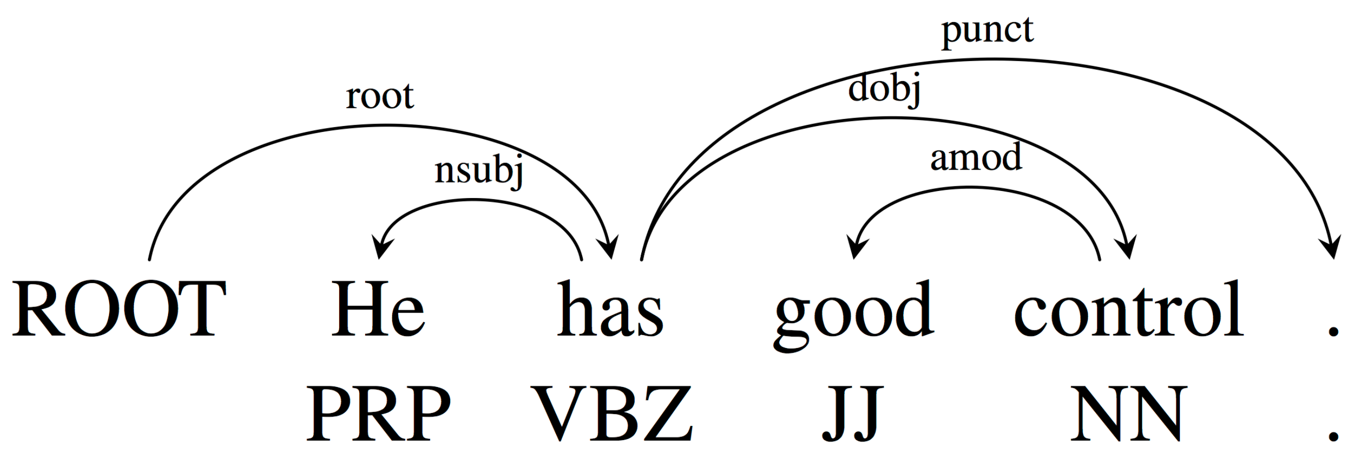
- During testing, we use the classifier to repeat predicting the action, until we reach a terminal configuration

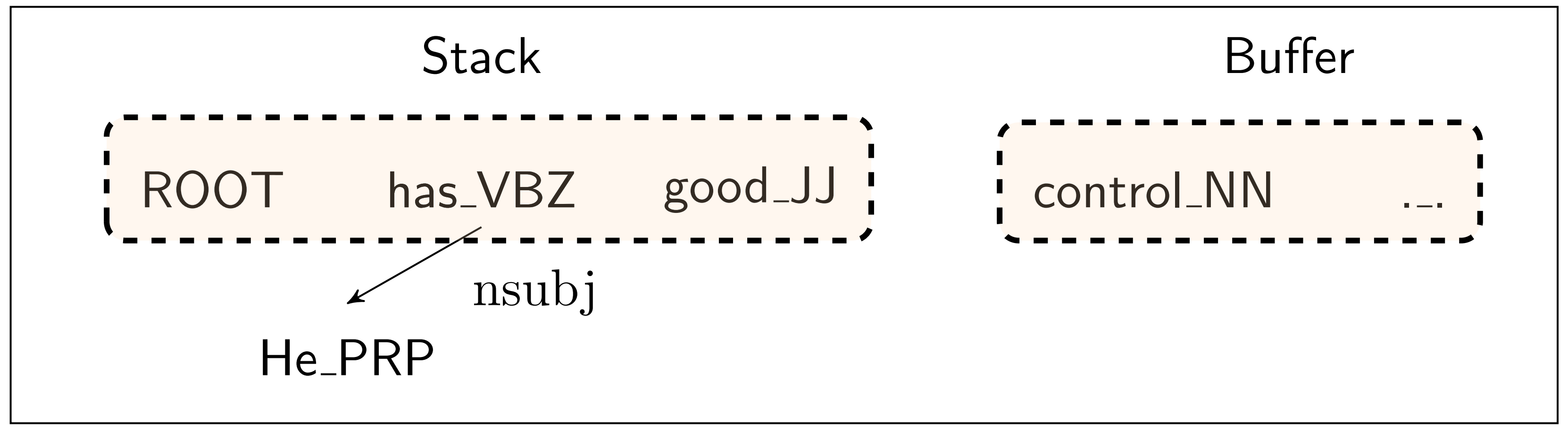
```
function DEPENDENCYPARSE(words) returns dependency tree

  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
```

- This is also called “greedy transition-based parsing” because we always make a local decision at each step
    - It is very fast (linear time!) but less accurate
    - Can easily do beam search

MaltParser





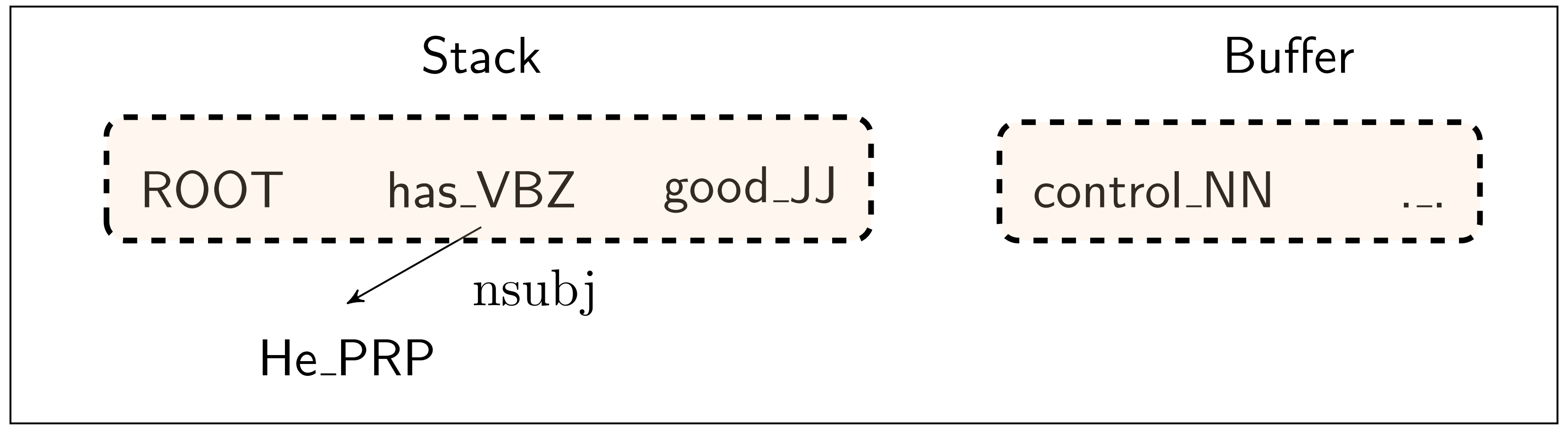
- Extract features from the configuration
- Use your favorite classifier: logistic regression, SVM...

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

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

MaltParser



Feature templates

$s_2 . w \circ s_2 . t$

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

$lc(s_2) . t \circ s_2 . t \circ s_1 . t$

$lc(s_2) . w \circ lc(s_2) . l \circ s_2 . w$

Features

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

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

$lc(s_2) . t = \text{PRP} \circ s_2 . t = \text{VBZ} \circ s_1 . t = \text{JJ}$

$lc(s_2) . w = \text{He} \circ lc(s_2) . l = \text{nsubj} \circ s_2 . w = \text{has}$

Usually a combination of 1-3 elements from the configuration

Binary, sparse, millions of features

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

More feature templates

```
# 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)
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1).
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)

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

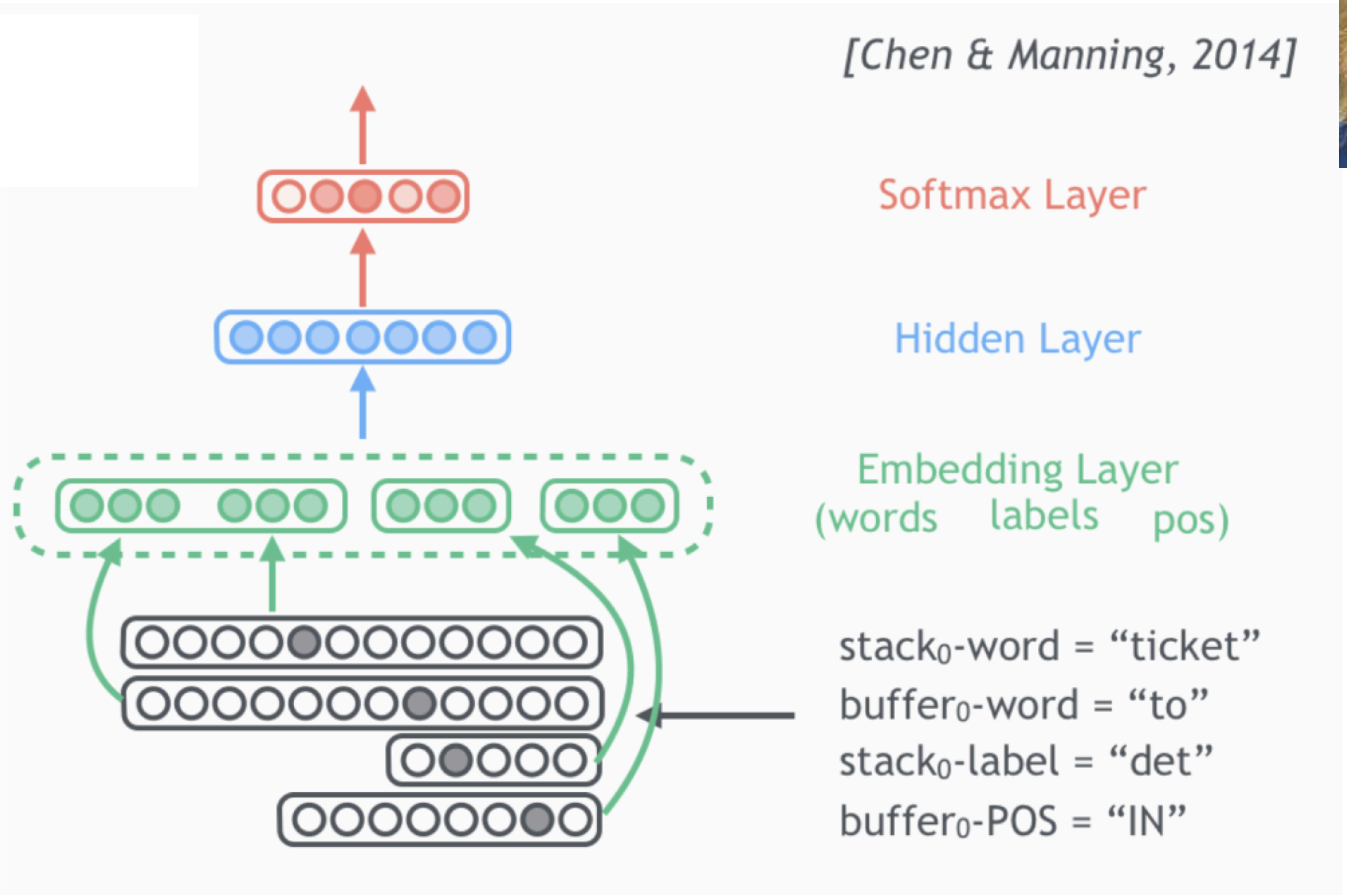
Parsing with neural networks

Representation for configuration:

- Embeddings for words/POS tags on top of stack
- Embeddings for words/POS tags at front of buffer
- Embeddings for existing arc labels at specific positions

Classifier:

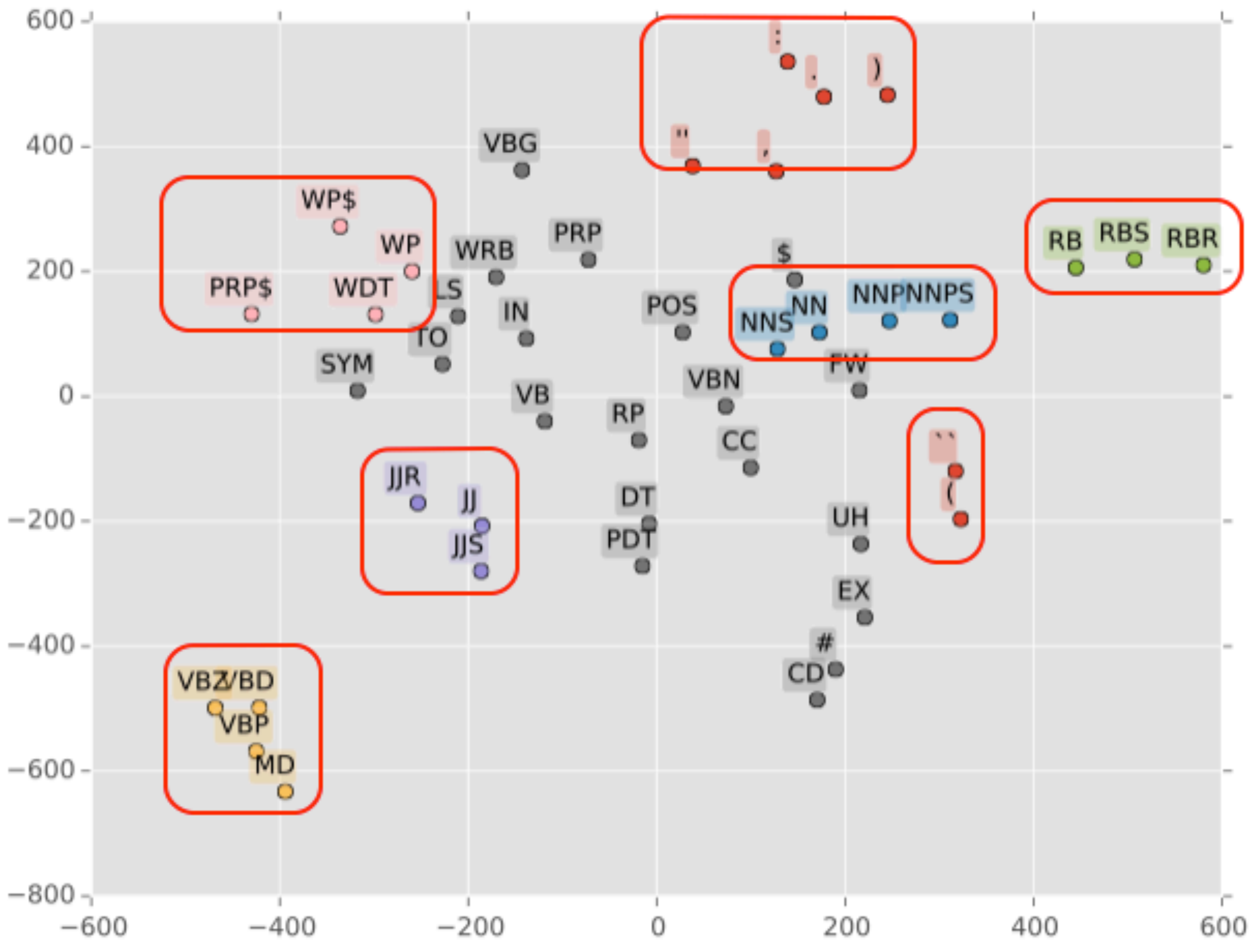
- Feed-forward neural network (input representation has a fixed dimensionality)



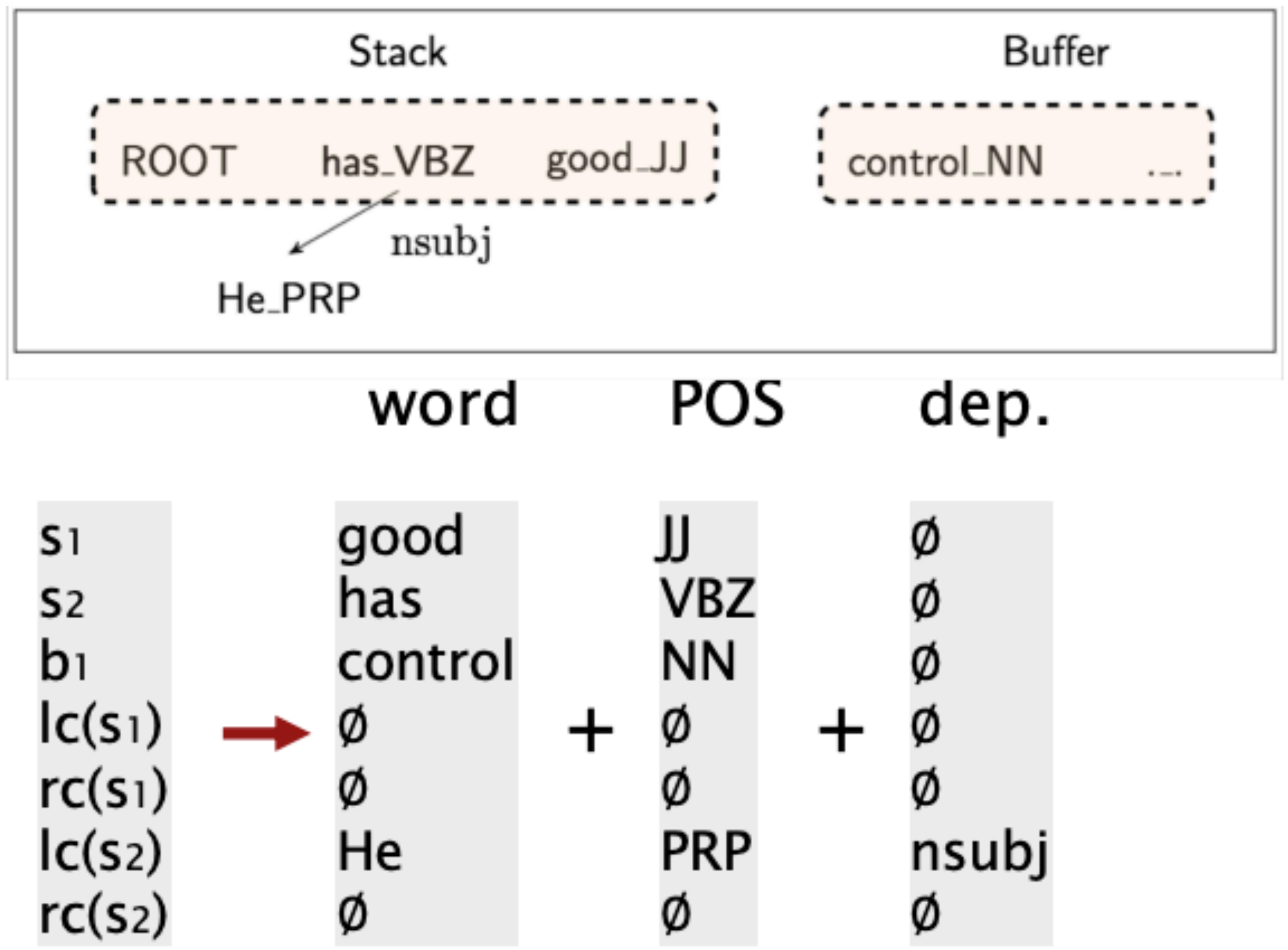
(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks

Parsing with neural networks

- Used pre-trained word embeddings
- Part-of-speech tags and dependency labels are also represented as vectors



- No feature template any more!



Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

- A simple feedforward NN: what is left is backpropagation!

(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks

Further improvements

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global normalization

Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

Google's SyntaxNet and the Parsey McParseFace (English) model

Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source

Thursday, May 12, 2016

Handling non-projectivity

- The arc-standard algorithm we presented only builds projective dependency trees
- Possible directions:
  - Give up!
  - Post-processing
  - Add new transition types (e.g., SWAP)
  - Switch to a different algorithm (e.g., graph-based parsers such as MSTParser)

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

Graph-based dependency parsing

- **Basic idea**: let's predict the dependency tree directly

$$Y^* = \arg\max_{Y \in \Phi(X)} score(X, Y)$$

X: sentence, Y: any possible dependency tree

- **Factorization**:

$$score(X, Y) = \sum_{e \in Y} score(e) = \sum_{e \in Y} w^\intercal f(e)$$

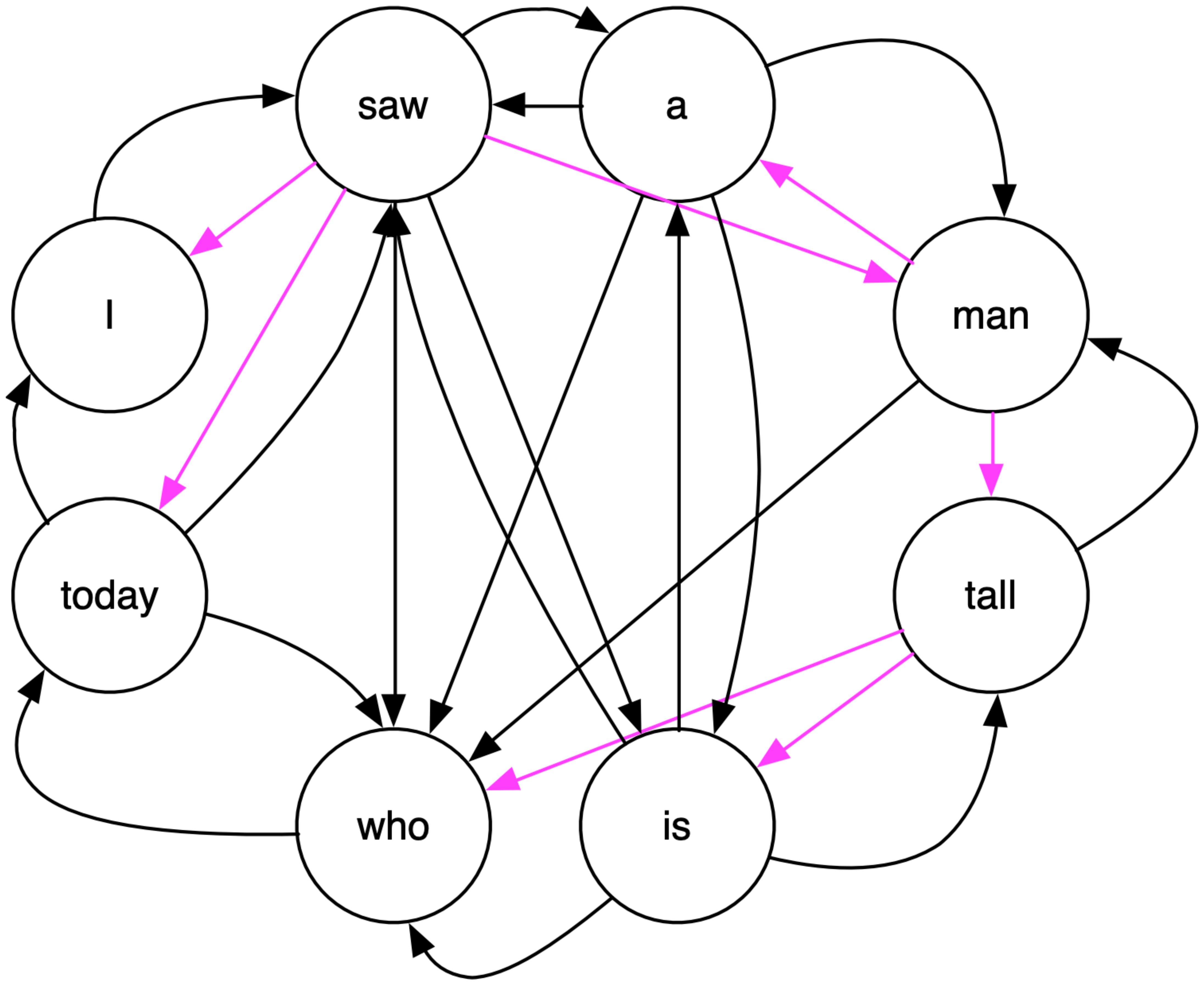
Assign scores/weights to all possible edges

Train a model to compute these scores

- **Inference**: finding maximum spanning tree (MST) for weighted, directed graph

MST Parsing Inference

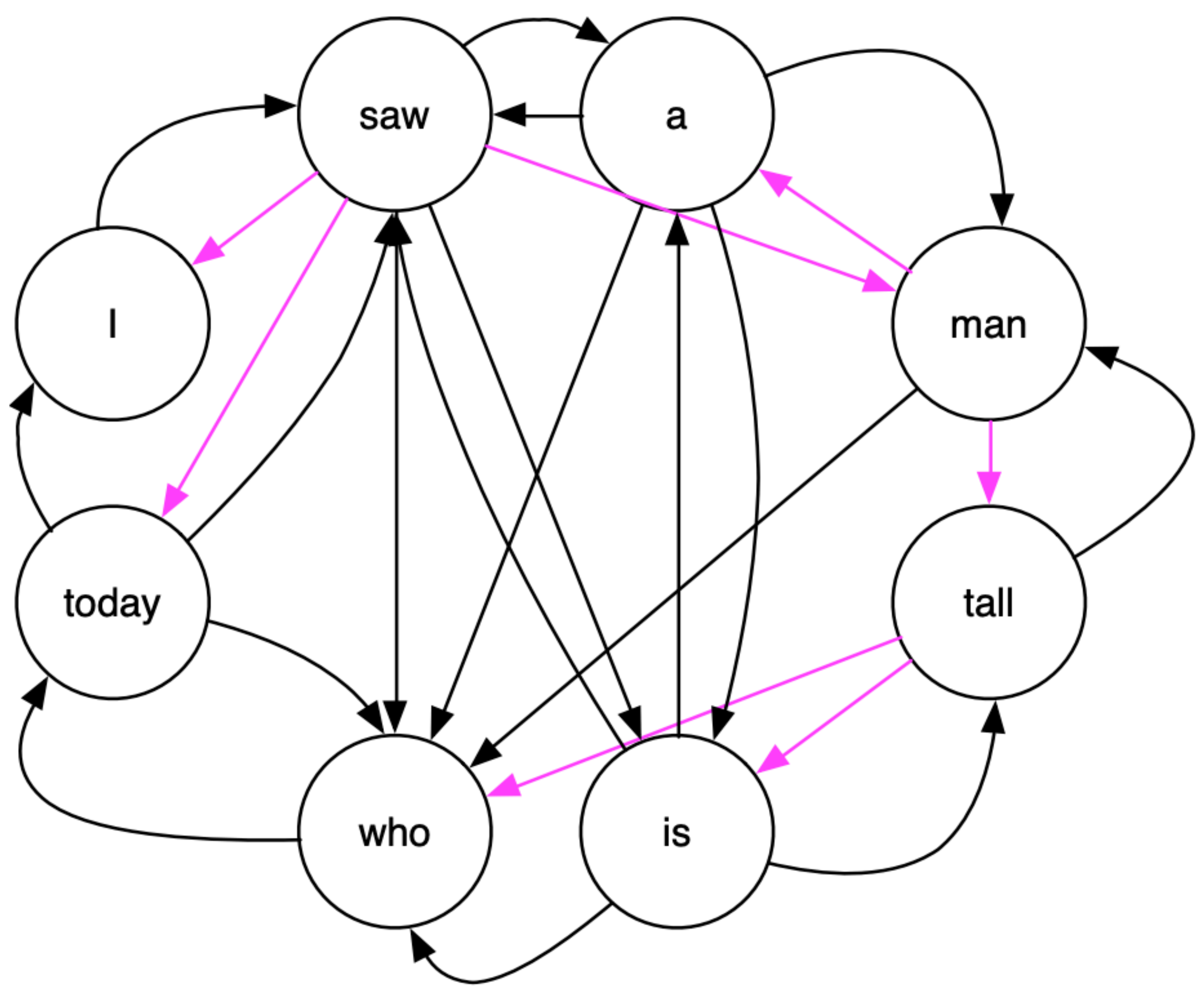
- We start out with a fully connected graph with a score for each edge
- N^2 edges total



(slide credit: Berkeley Info 159/259, David Bamman)

MST Parsing Inference

- From this graph G, we want to find a spanning tree (tree that spans G [includes all the vertices in G])
- If the edges have weights, the best parse is the maximal spanning tree (the spanning tree with the highest total weight).



(slide credit: Berkeley Info 159/259, David Bamman)

Graph-based dependency parsing

- **Training** learn parameters so the score for the gold tree is higher than for all other trees

Graph-based dependency parsing

- **Training** learn parameters so the score for the gold tree is higher than ~~for all other trees~~ a single best tree

Train using structured margin loss: structured perceptron

Structured Perceptron

- Simple way to train (non-probabilistic) global models
- Find the one-best, and if it's score is better than the correct answer, adjust parameters to fix this

$$\hat{Y} = \text{argmax}_{\tilde{Y} \neq Y} S(\tilde{Y} \mid X; \theta)$$

← Find one best

$$\textbf{if } S(\hat{Y} \mid X; \theta) \geq S(Y \mid X; \theta) \textbf{ then}$$

← If score better than reference

$$\theta \leftarrow \theta + \alpha\left(\frac{\partial S(Y \mid X; \theta)}{\partial \theta} - \frac{\partial S(\hat{Y} \mid X; \theta)}{\partial \theta}\right)$$

← Increase score of ref, decrease score of one-best (here, SGD update)

$$\textbf{end if}$$

(slide credit: CMU CS 11-747, Graham Neubig)

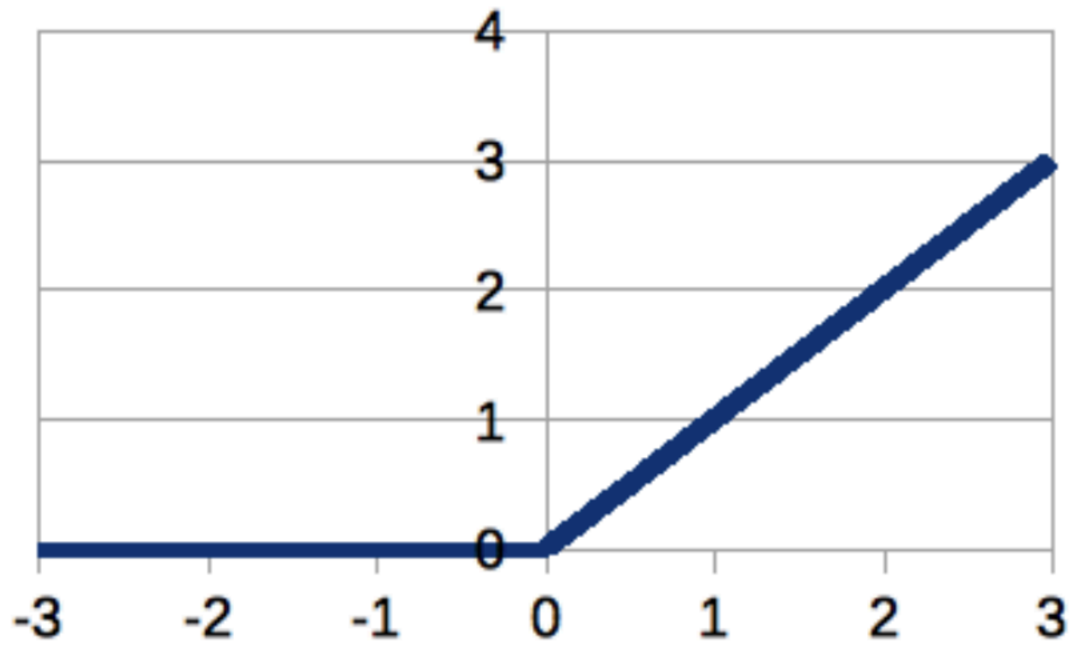
Structured Perceptron and Hinge Loss

- Loss functions for structured perceptron

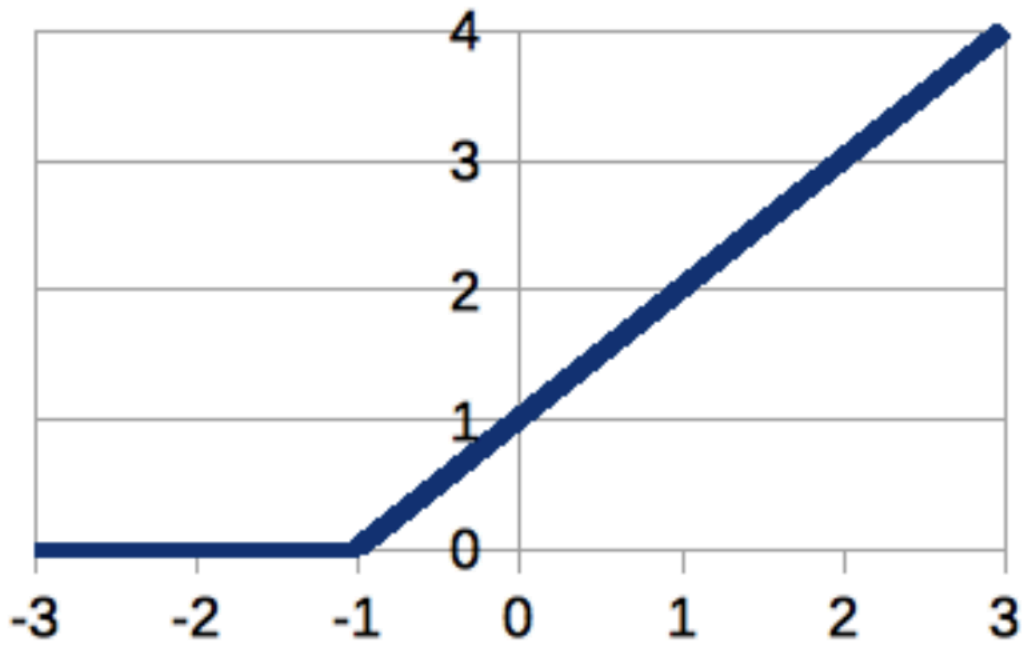
$$\ell_{\text{percept}}(X, Y) = \max(0, S(\hat{Y} \mid X; \theta) - S(Y \mid X; \theta))$$

- Penalize when incorrect answer is within margin m

$$\ell_{\text{hinge}}(x, y; \theta) = \max(0, m + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta))$$



Perceptron



Hinge

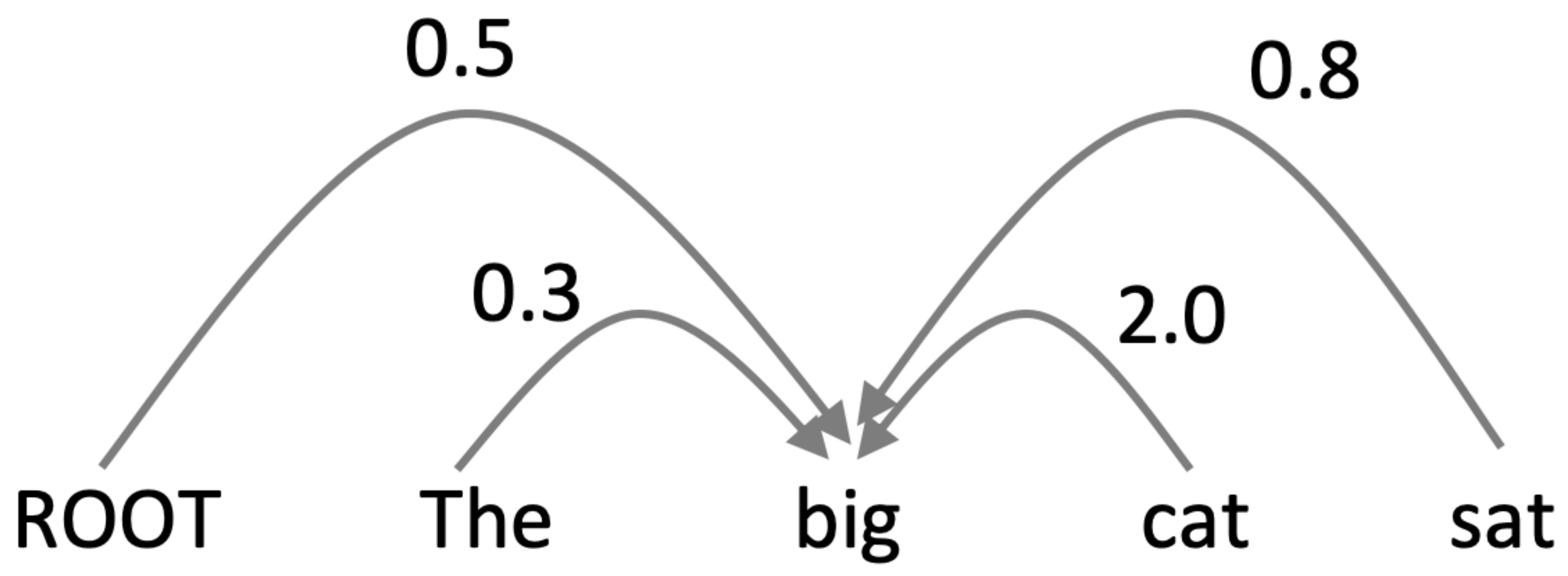
$$\ell_{\text{ca-hinge}}(x, y; \theta) = \max(0, \text{cost}(\hat{y}, y) + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta))$$

Note: hinge loss can be used instead of cross-entropy loss in other places as well

(slide credit: CMU CS 11-747, Graham Neubig)

Graph-based dependency parsing

- **Training** learn parameters so the score for the gold tree is higher than ~~for all other trees~~ a single best tree
- **To get a good tree**
  - Compute a score for every possible dependency for each word
  - With neural networks, leverage good “contexual” representations of each word token



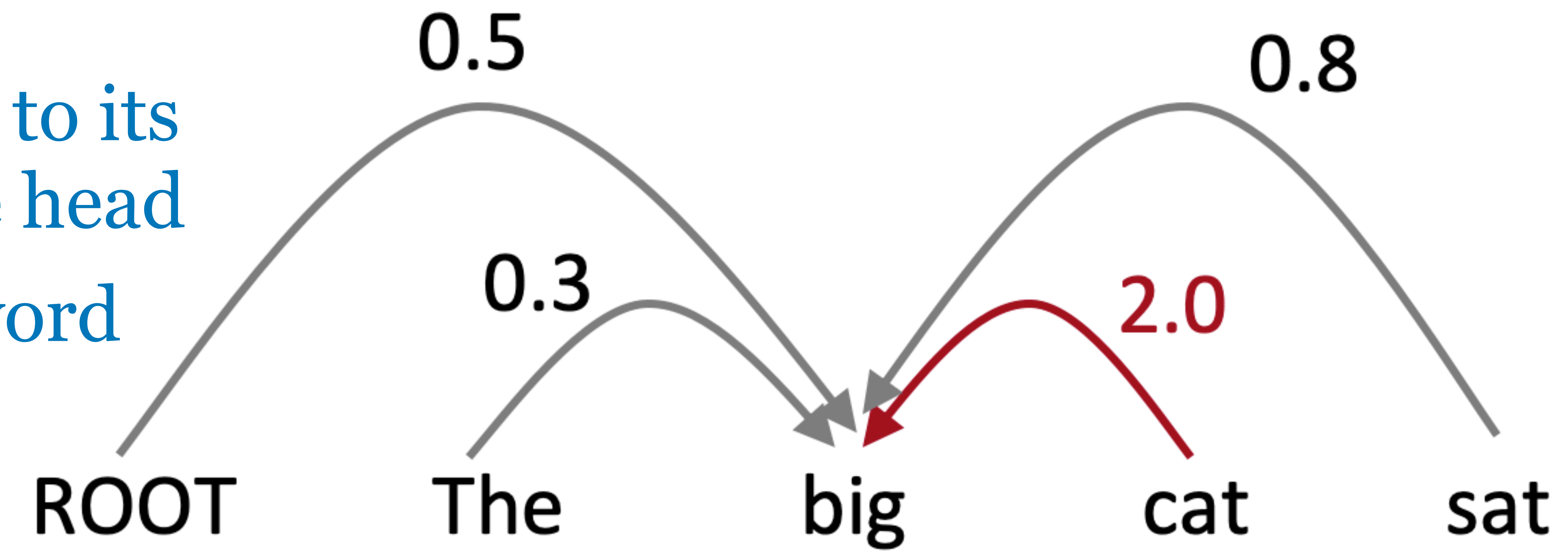
e.g., picking the head for “big”

(figure credit: Stanford CS224N, Chris Manning)

Graph-based dependency parsing

- **Training** learn parameters so the score for the gold tree is higher than ~~for all other trees~~ a single best tree
- **To get a good tree**
  - Compute a score for every possible dependency for each word
  - With neural networks, leverage good “contexual” representations of each word token

- Add edge from each word to its highest-scoring candidate head
- Repeat process for each word



e.g., picking the head for “big”

(figure credit: Stanford CS224N, Chris Manning)

Neural Networks for Graph-based Dependency Parsing

- Pre-neural networks
  - MSTParser - use hard crafted features (McDonald et al, 2005)

- Neural networks - leverage better representation ("contextual" embeddings)
  - Phrase Embeddings (Pei et al, 2015)
  - BiLSTM feature extractors (Kipperwasser and Goldberg 2016)
  - BiAffine Classifier (Dozat and Manning 2017)

Neural graph-based dependency parser (Dozat and Manning 2017)

- Great result!
- But slower than simple neural transition-based parsers
  - There are n^2 possible dependencies in a sentence of length n

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	**95.74**	**94.08**

(slide credit: Stanford CS224N, Chris Manning)

Summary

- Dependency parsing: labeled edges between words
- Two families of algorithms
  - Transition-based dependency parsing
    - Build graph incrementally:
      - train classifier to predict action based on current configuration
    - Linear time
  - Graph-based dependency parsing
    - Score graph edges
    - Get maximum spanning tree