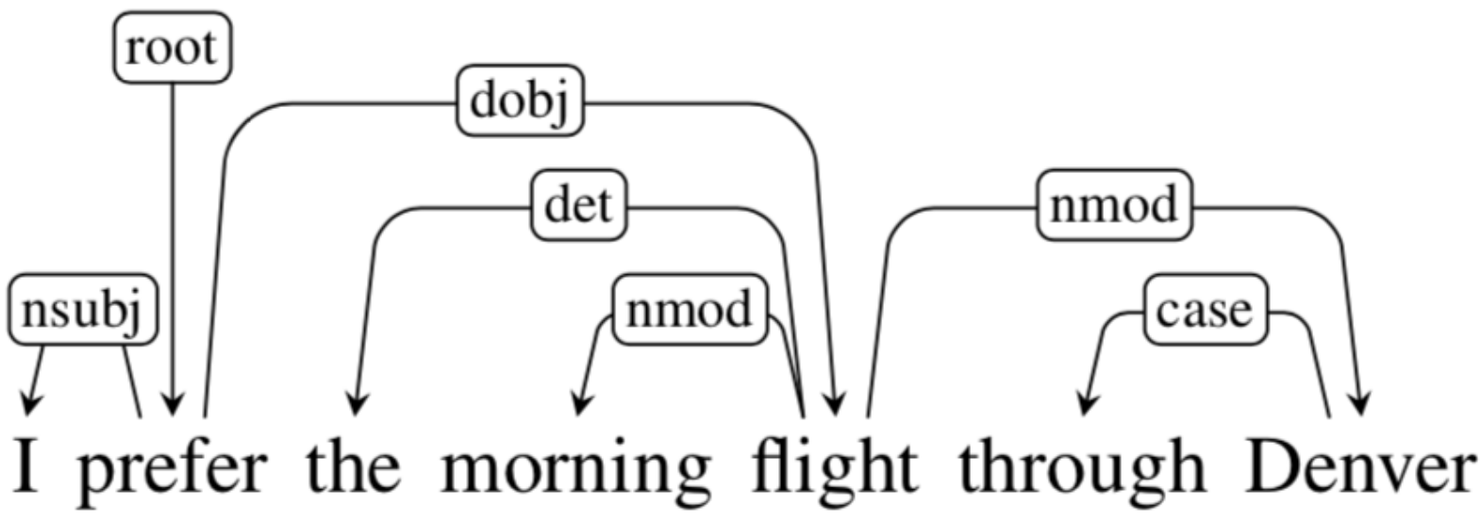
ENLP Lecture 23: Dependency Parsing

Shira Wein

With slides/text from Nathan Schneider, Harry Eldridge, Chris Manning

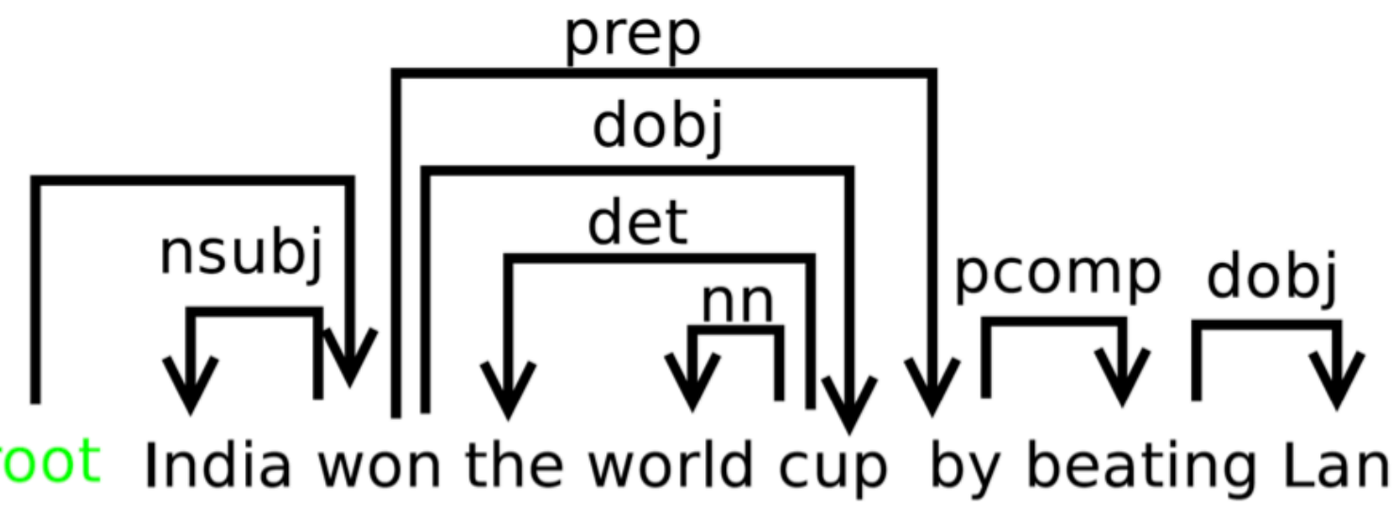
April 20, 2021



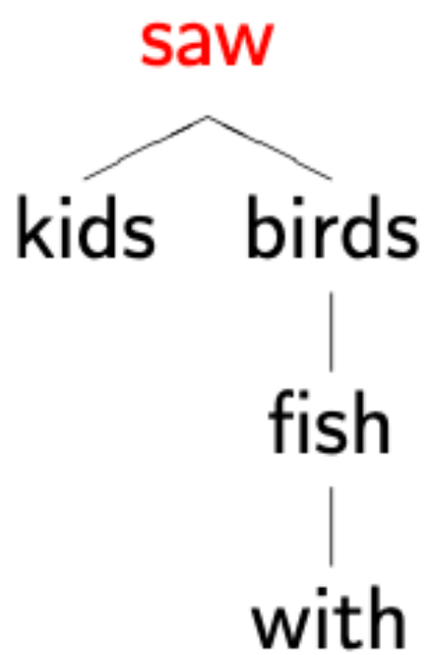
Language is complicated!

Dependency parsing can help clarify what is connected to what

Example Dependency Parse



From slides on 3/25



What is a dependency grammar?

- Dependency: a relation between two words, where one is the **head** and the other is the **dependent**

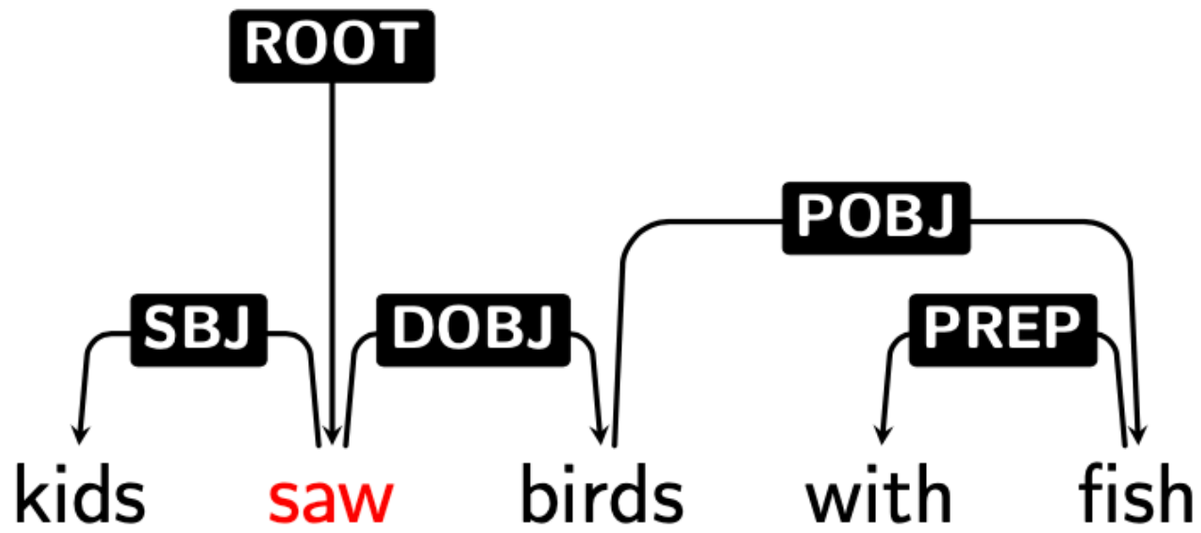
- Every word depends on exactly one other word (except for the root word)

- Build a dependency tree by determining which word every word depends on

- Normally binary asymmetric relations

How dependency parses work

- Tree
- Every word has exactly one parent (one edge pointing to it)
- Label edges to indicate the head → modifier **relations**
- Usually one word is the **root**
- Don’t want cycles



Important relations (for English)

- (Nominal) Subject
- Direct Object
- Determiner
- Adjective Modifier
- Adverbial Modifier
- etc.

Important relations (for English)

- There are many different **flavors** of dependency parses.
  - Stanford Dependencies;
    Universal Dependencies (UDv1, UDv2); …
    - Some differences in structure (head rules)
    - Different relation label sets
- Examples on different slides use different flavors.
  - For this class, the particular framework is not important.

Example of language being complicated:

“I saw a girl with a telescope”

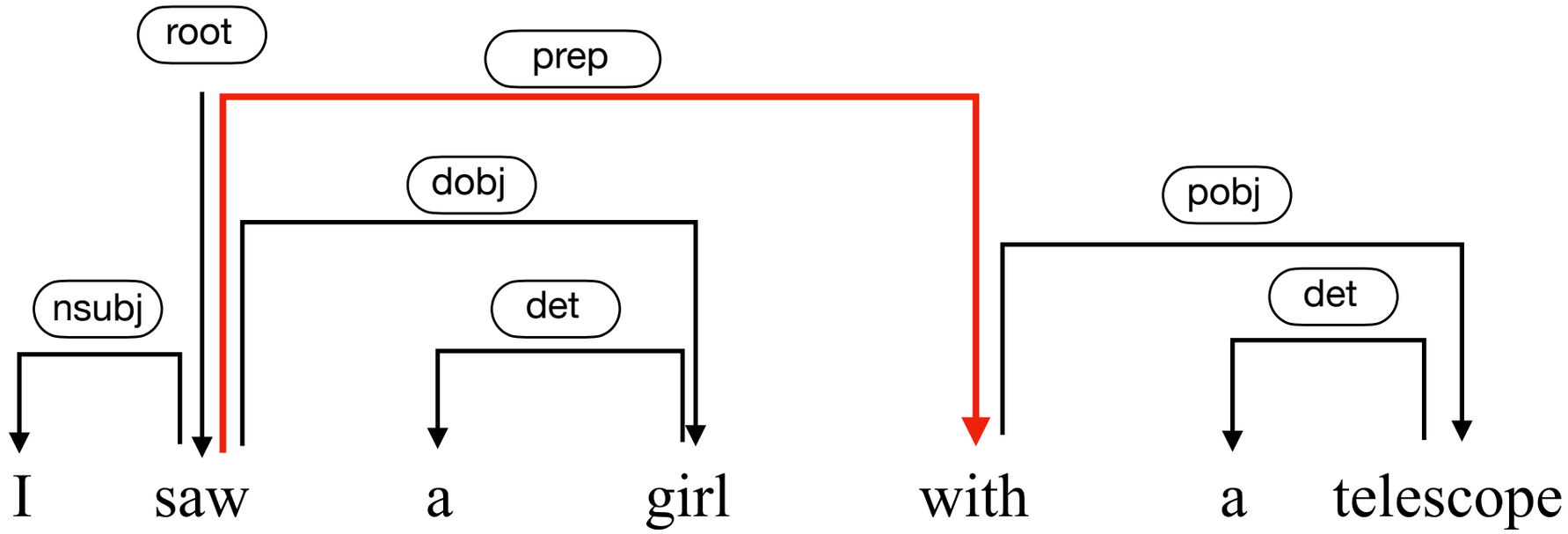
I saw a girl with a telescope

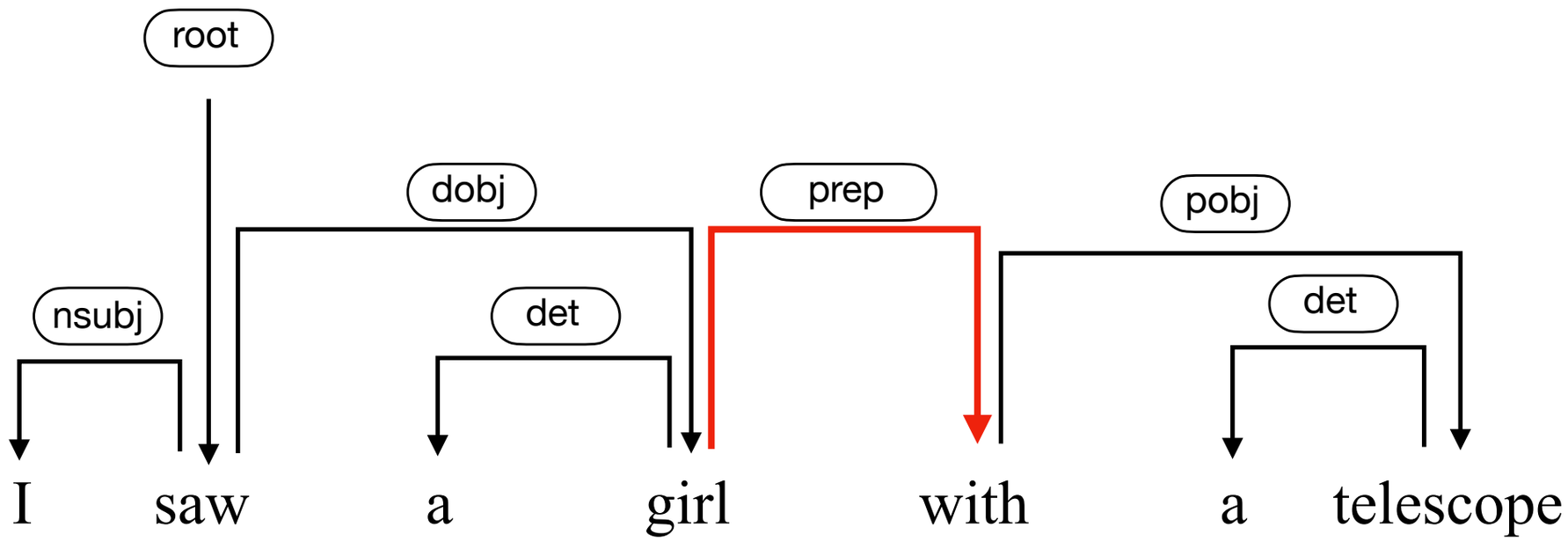
with = having

with = using

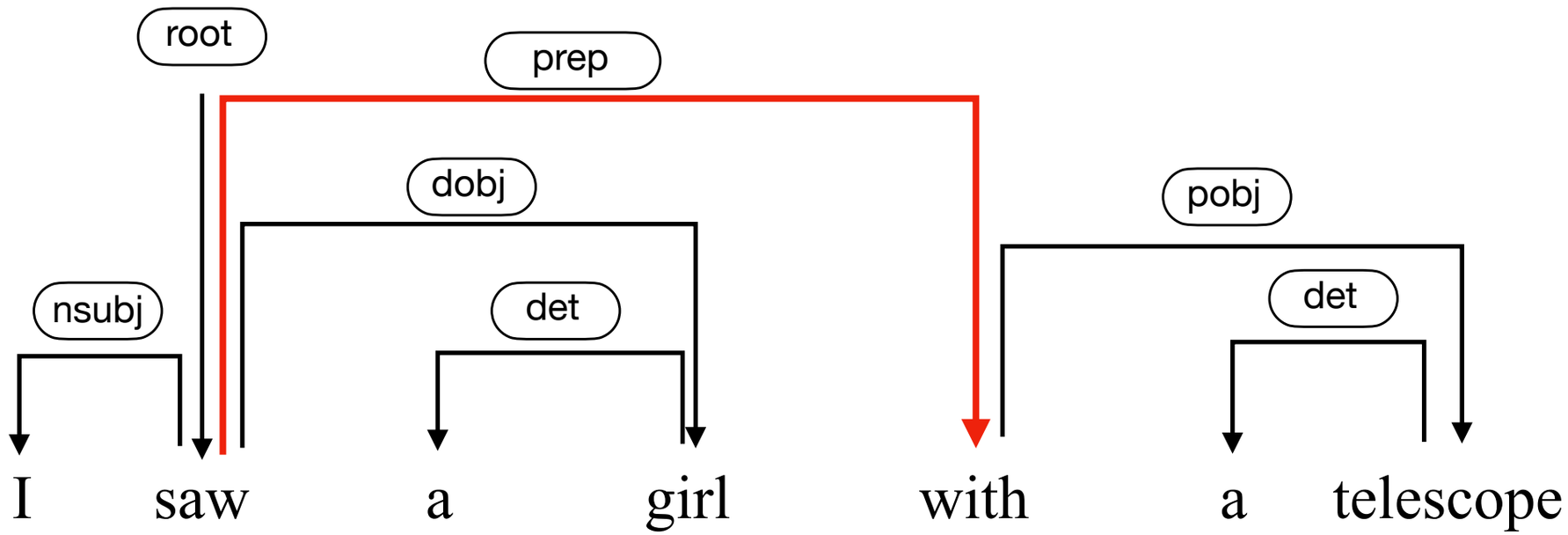
Image from https://www.cellstrat.com/2020/04/16/dependency-parsing-for-nlp/

With can connect to seeing or to the girl: two separate dependency parses

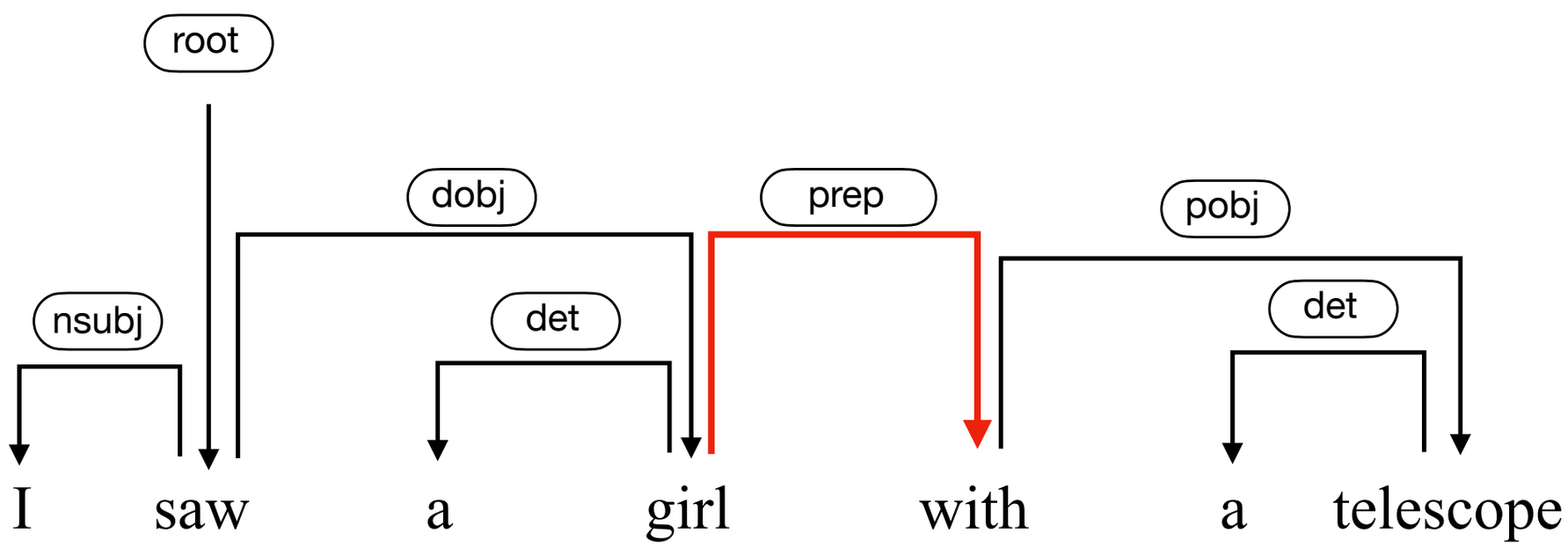




Which is with as in “using”?



With as in using— “with a telescope” modifies “saw”



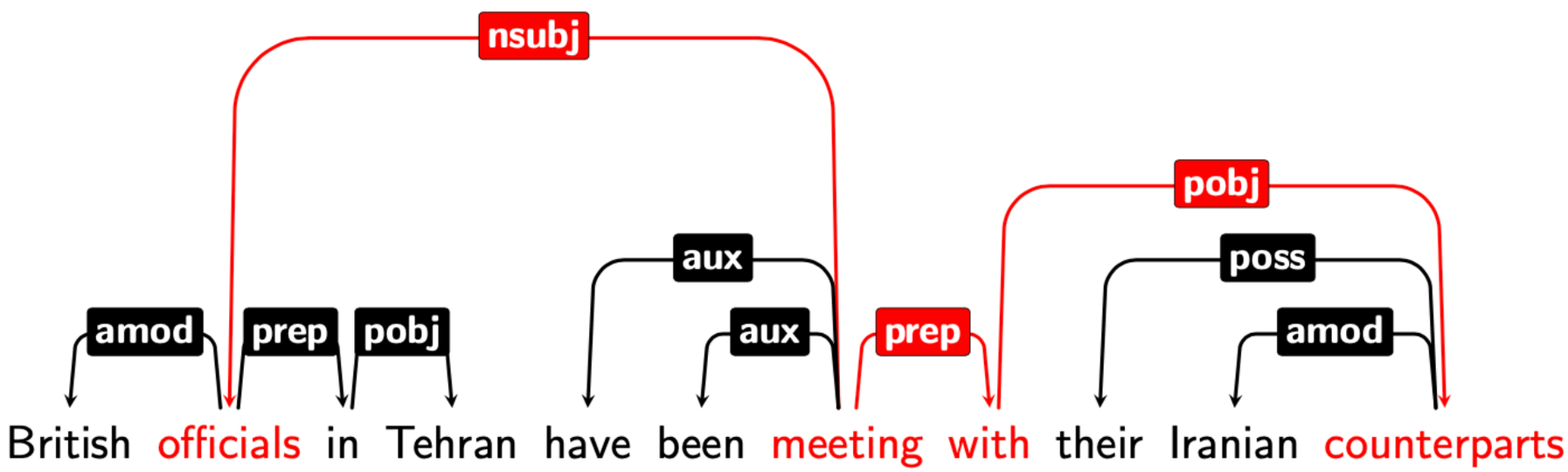
With as in having— “with a telescope” modifies “girl”

Dependency parsing is useful

- Resolves attachment ambiguities that can matter for meaning
    - Grammatical structure of a sentence based on the relationships (dependencies) between the words
- Syntactic dependencies can be close to semantic relations
- Language agnostic
- For what types of tasks might this be useful?

Information Extraction

- Can be used in **information extraction** to capture relationships
- Entity linking: maps entities to database entries
- Relation extraction: mines text to find relationships between entities



Machine Translation

- When incorporated as linguistic prior during training into neural machine translation, improves performance
    - https://www.aclweb.org/anthology/P17-2012/
- (Not standard practice to incorporate dependencies in MT)

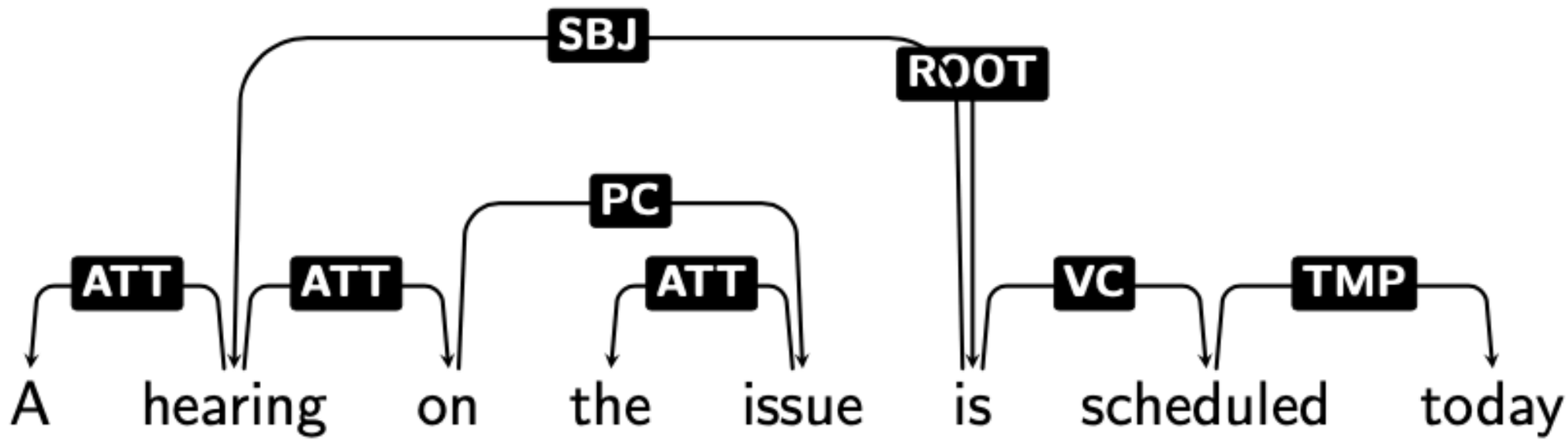
Learning to Parse and Translate Improves Neural Machine Translation

Akiko Eriguchi[†], **Yoshimasa Tsuruoka**[†], and **Kyunghyun Cho**[‡]
[†]The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
{eriguchi, tsuruoka}@logos.t.u-tokyo.ac.jp
[‡]New York University, New York, NY 10012, USA
kyunghyun.cho@nyu.edu

Before we try it ourselves, some details on edges & heads

Can arrows cross? Projectivity

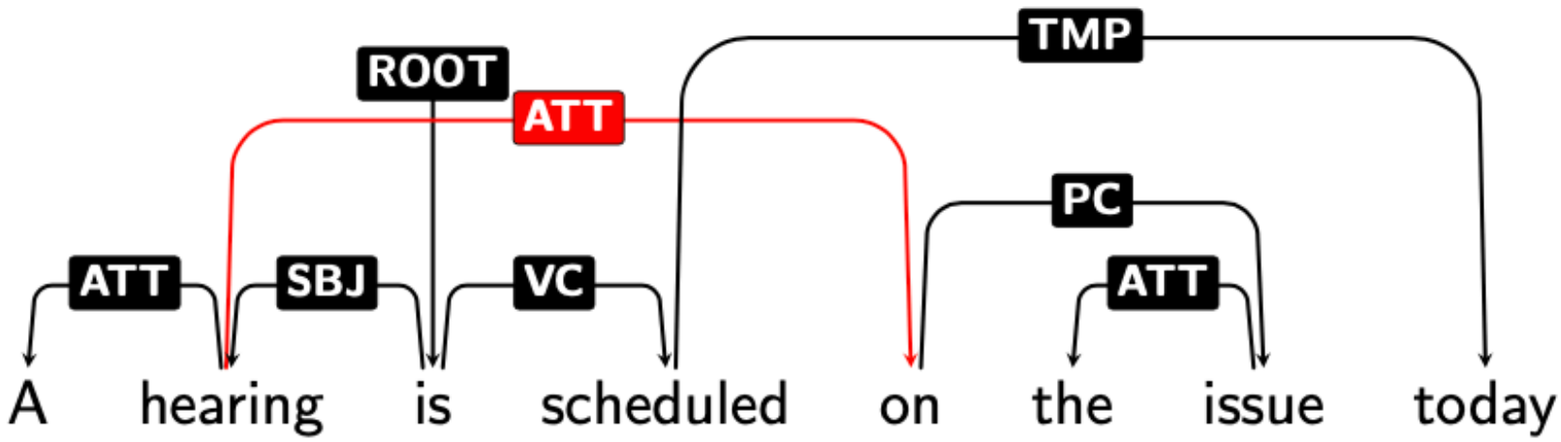
- A dependency parse is **projective** if every subtree is a contiguous span of the sentence
- i.e. **Projective** = there are no crossing edges



Is this tree projective?

Can arrows cross?
Projectivity

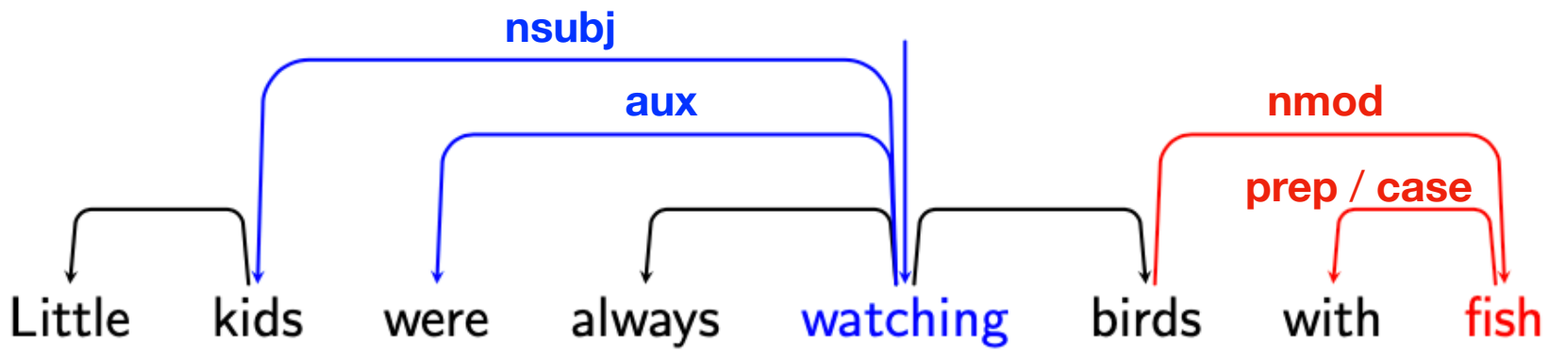
- A dependency parse is **projective** if every subtree is a contiguous span of the sentence
- i.e. **Projective** = there are no crossing edges



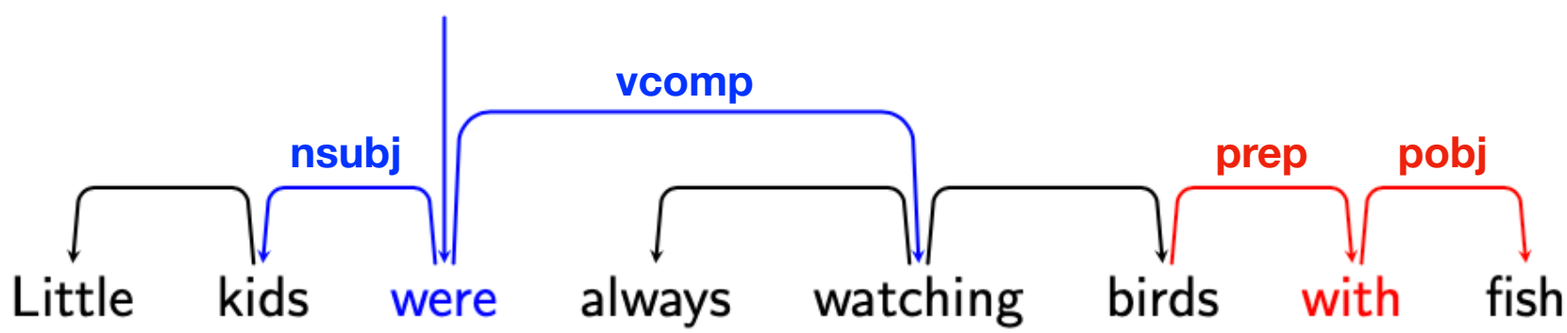
How about this one?

Heads

- Some dependency parse flavors prioritize **content words** as heads (auxiliaries, prepositions, etc. are modifiers)



- Other flavors use **functional** heads (prepositions head their objects, auxiliaries head main verbs, …)

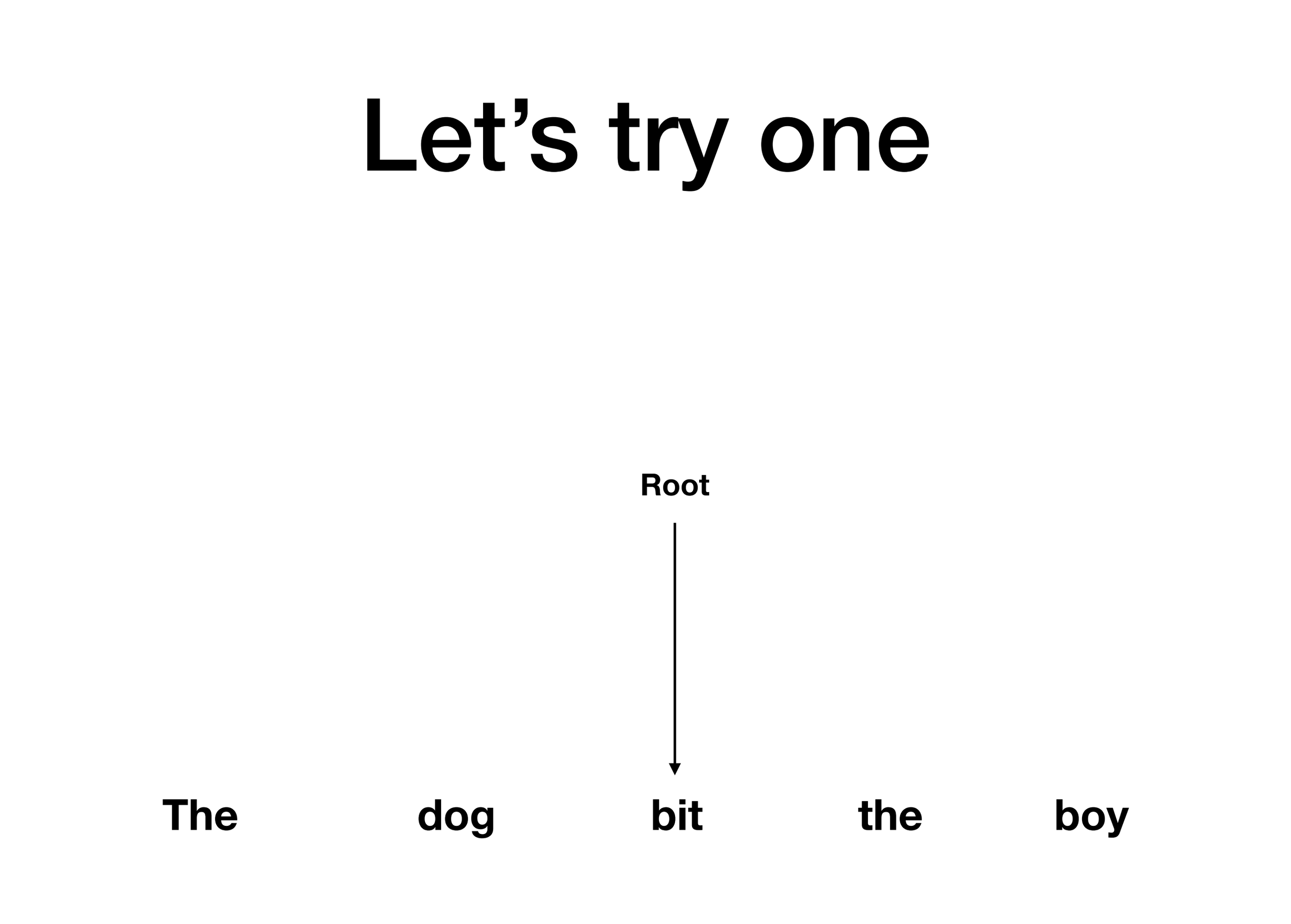


Let’s try one

1. Root?

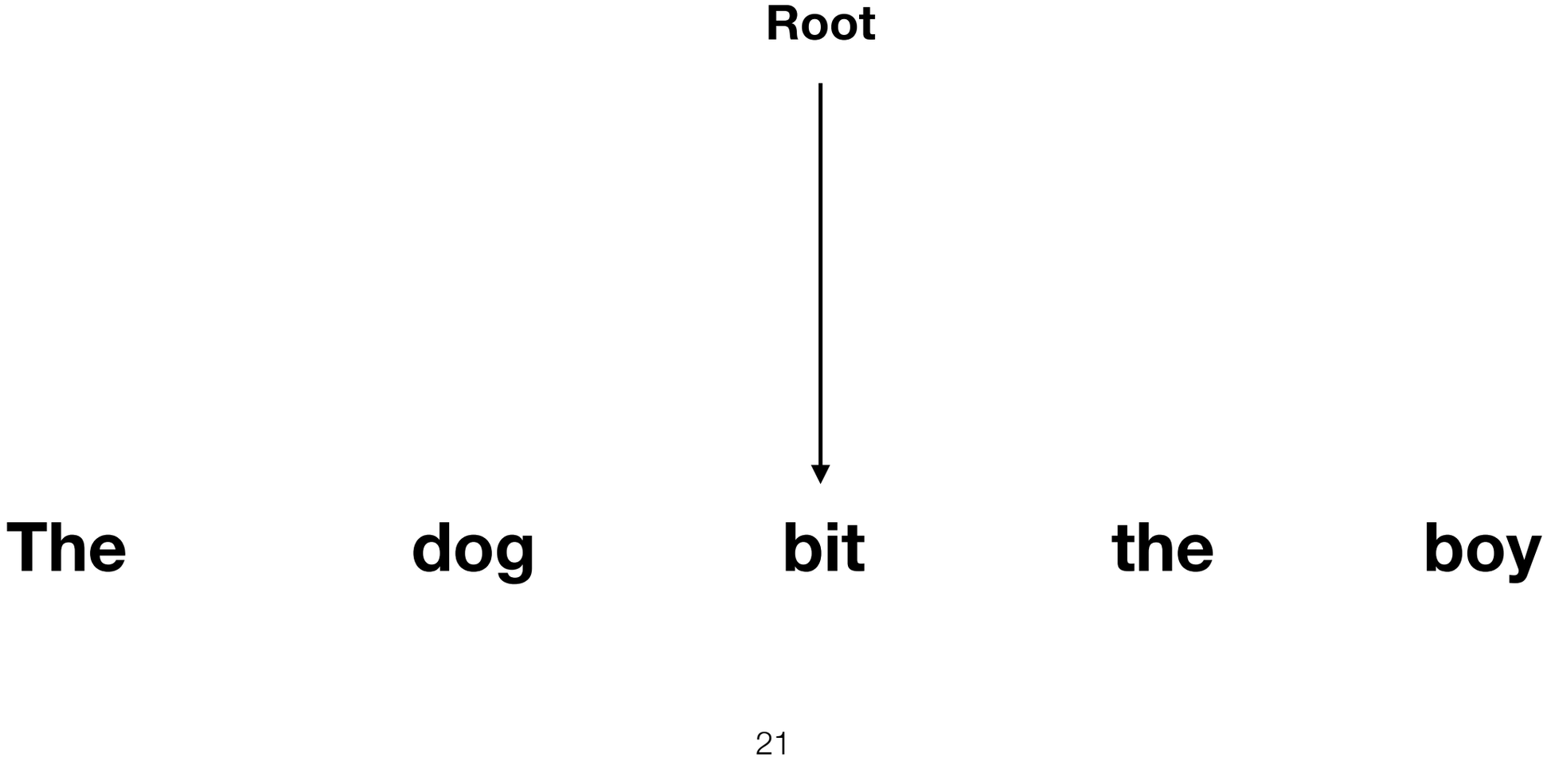
The dog bit the boy

Let’s try one

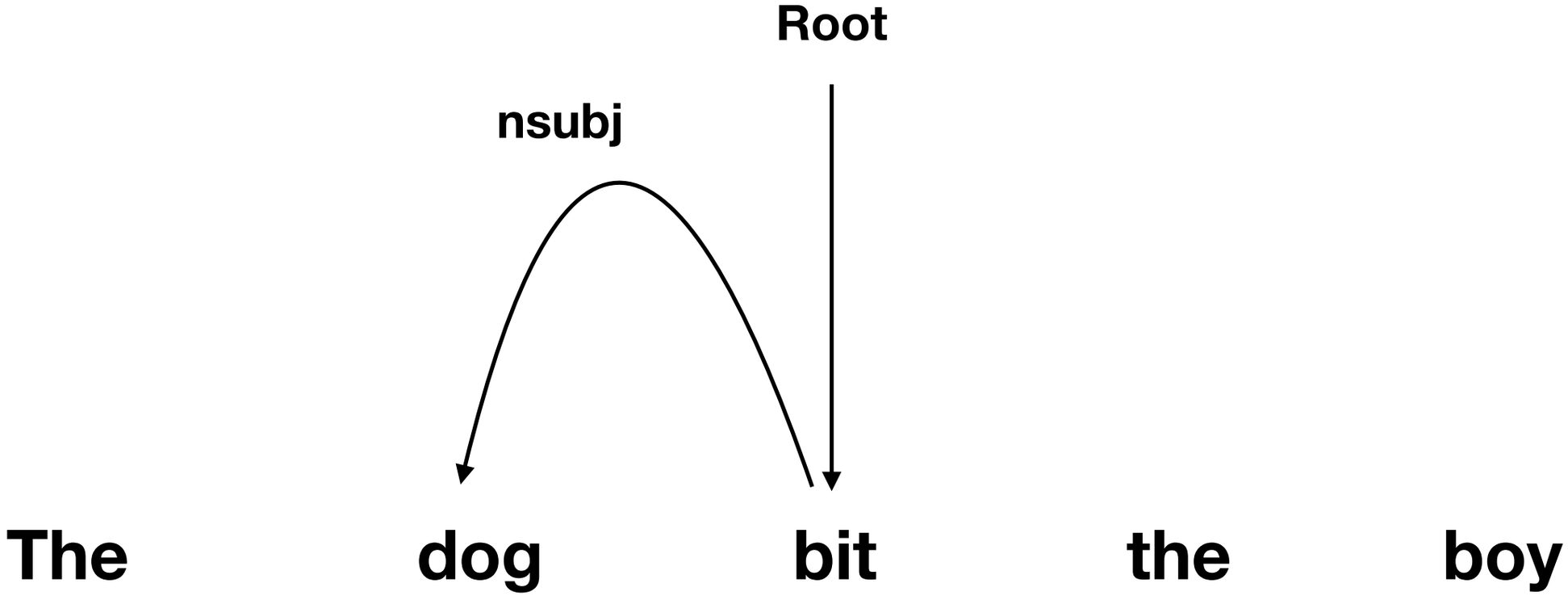


Let’s try one

2. Relation to dog?

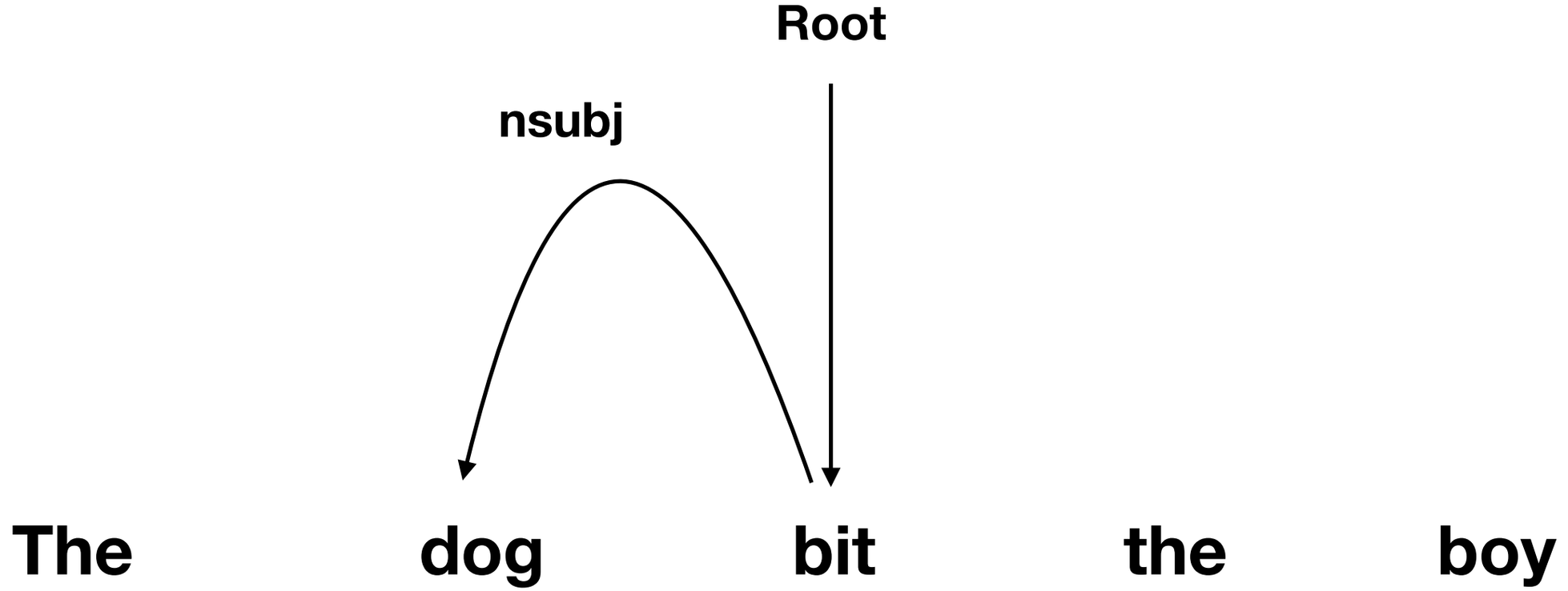


Let’s try one

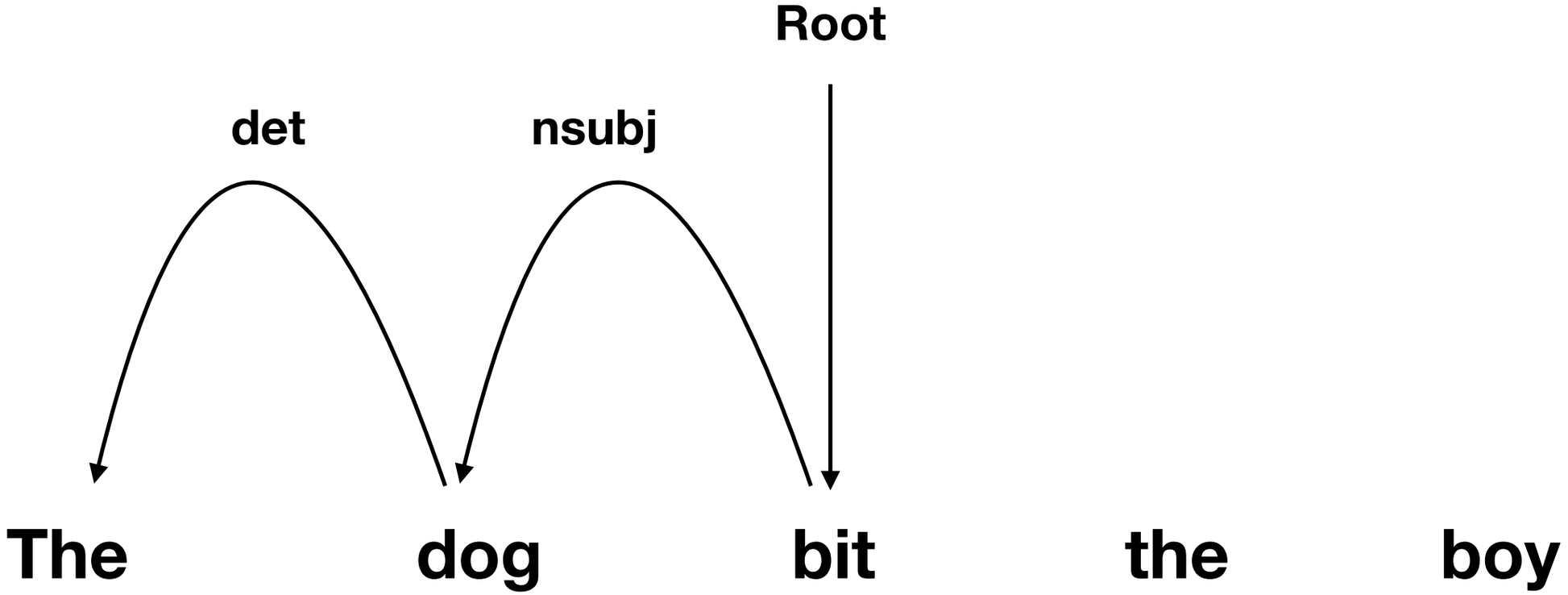


Let’s try one

3. Dog’s relation to the?

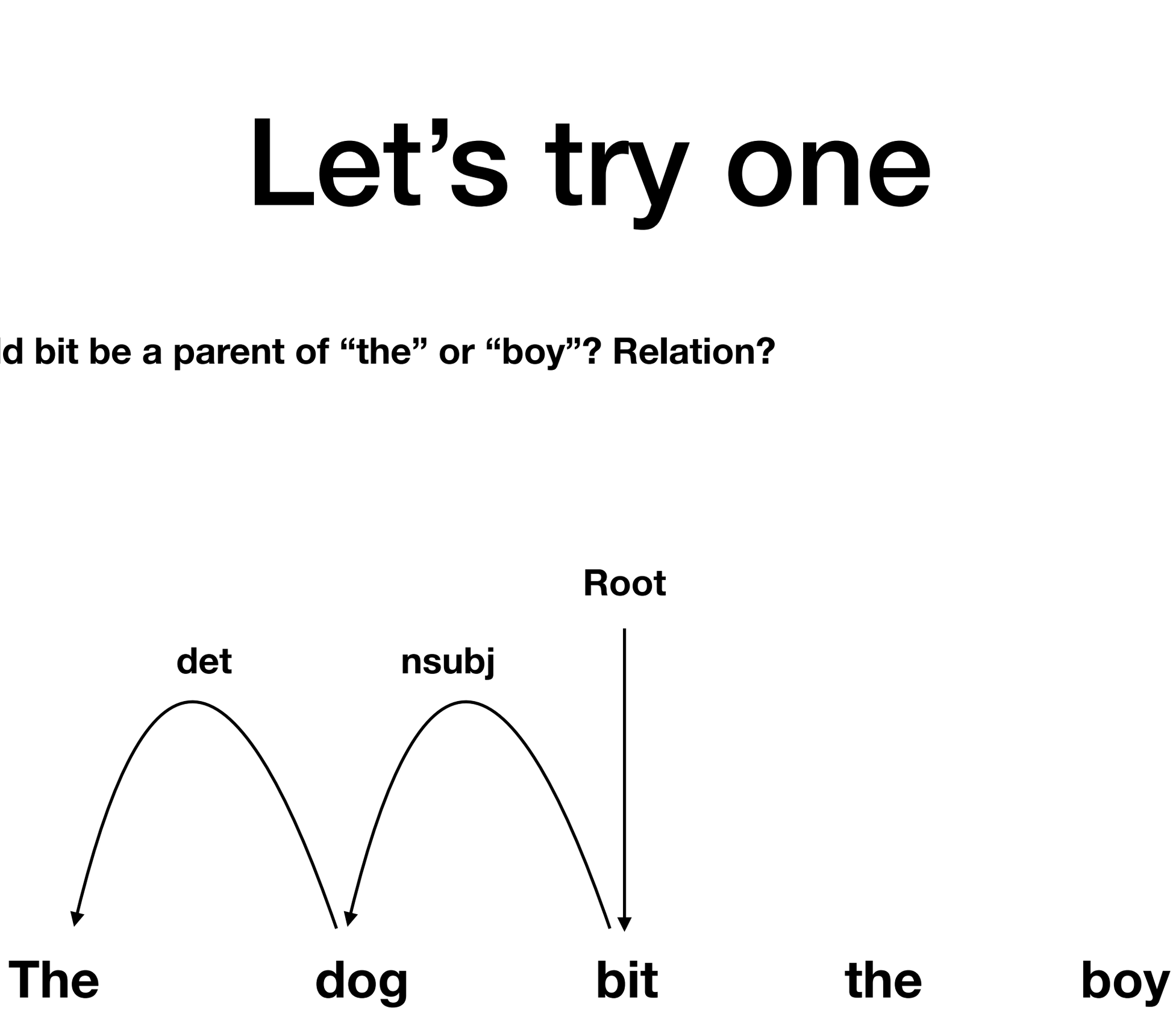


Let’s try one

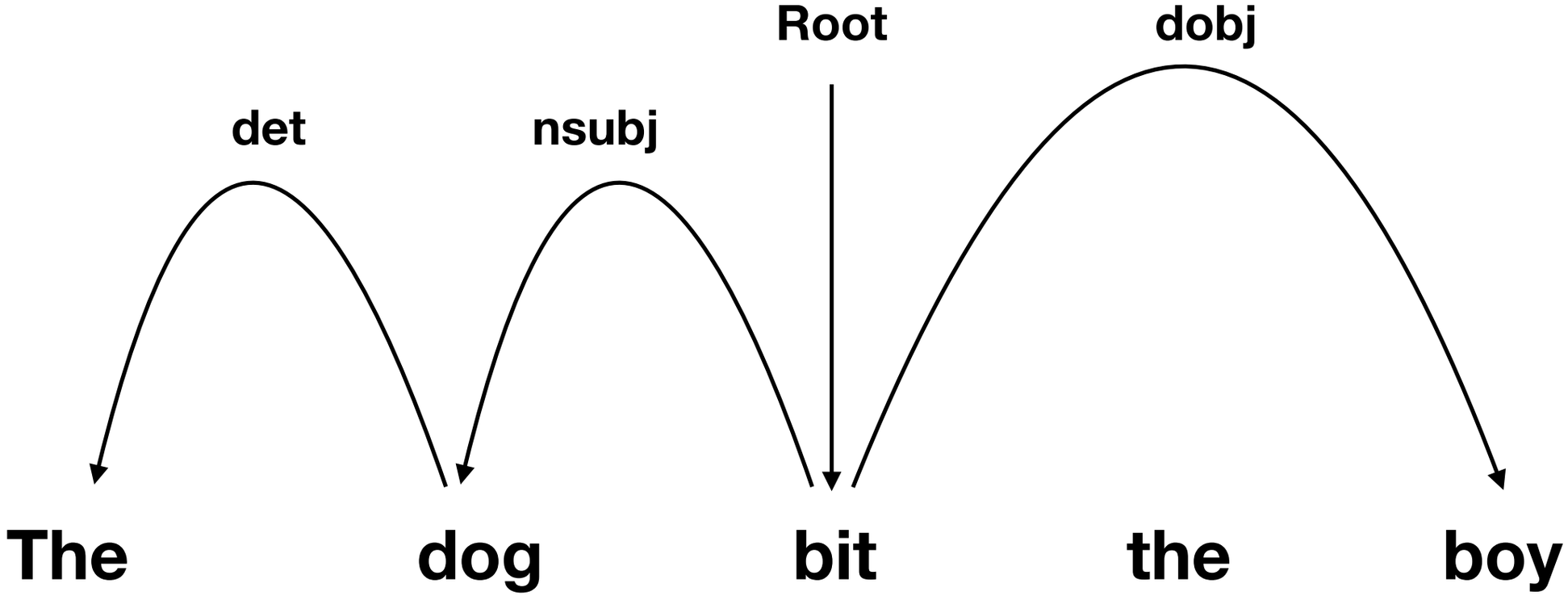


Let’s try one

4. Should bit be a parent of “the” or “boy”? Relation?

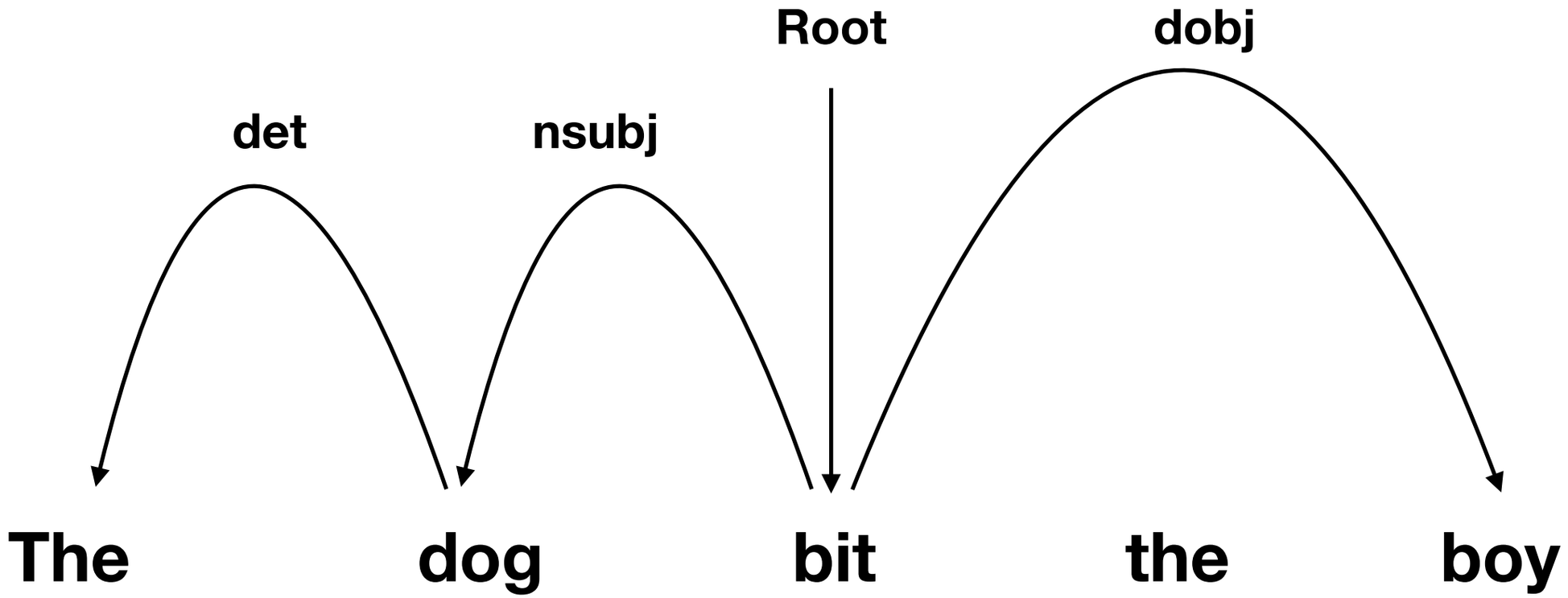


Let’s try one

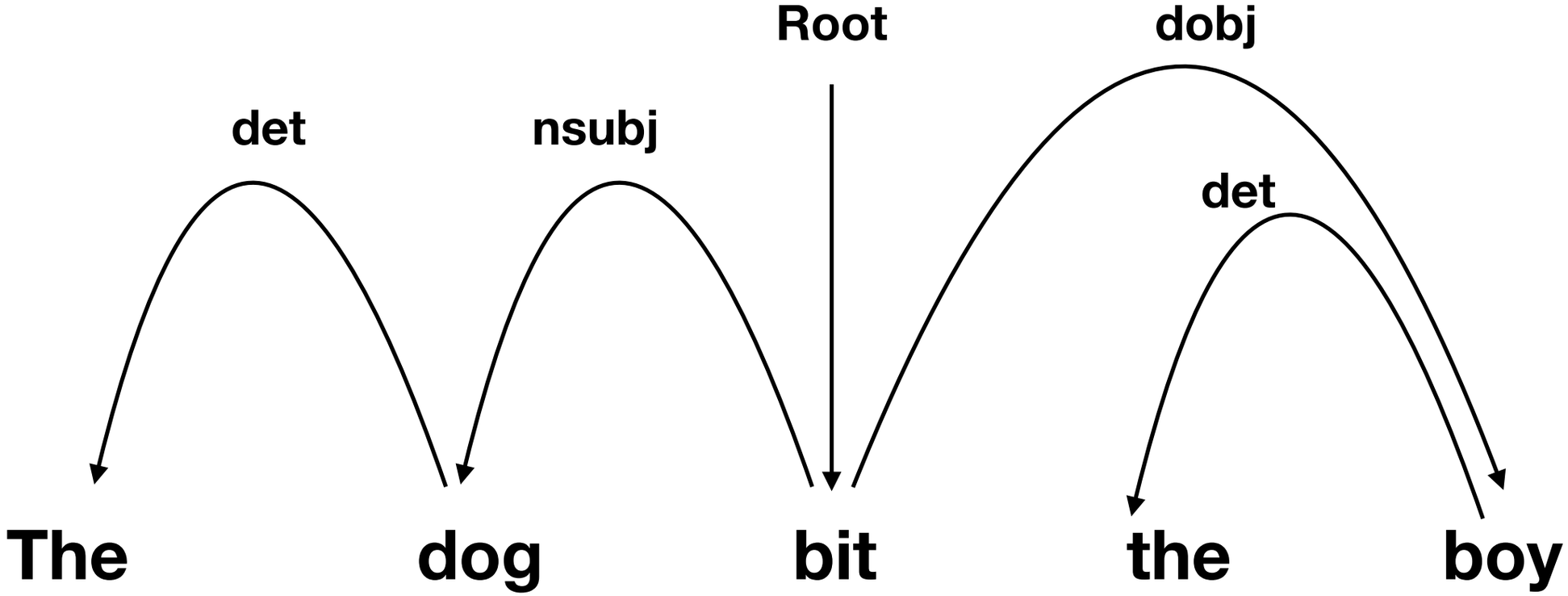


Let’s try one

5. Relation between “Boy” and “the”



Let’s try one



Transition-based Parsing (sort of tricky, part of A5)

Transition-based Parsing

- Process words from left to right, deciding if the two words should be attached

- Builds a dependency parse using a stack and buffer

- Input buffer: words of the sentence

- Stack: to manipulate the words

- Dependency relations: list of relations that culminate in the dependency parse

[STACK]	[INPUT BUFFER]	[RELATIONS]
[root]	[these, are, words, in, a, sentence]	

Create a dependency between ‘root’ and the word after ‘root’ on the stack if the word on the stack IS the root

Arc-Standard Approach

- Build relations between words using **ARCS** and remove word from stack once you have identified the word's parent

- **LEFTARC:** the word at the top of the stack is the head of the word beneath it
    - Remove second word from stack (the word you just made a dependent of the top word)

- **RIGHTARC:** (the reverse) the second word on the stack is the head of the word on top of the stack
    - Remove top word from stack

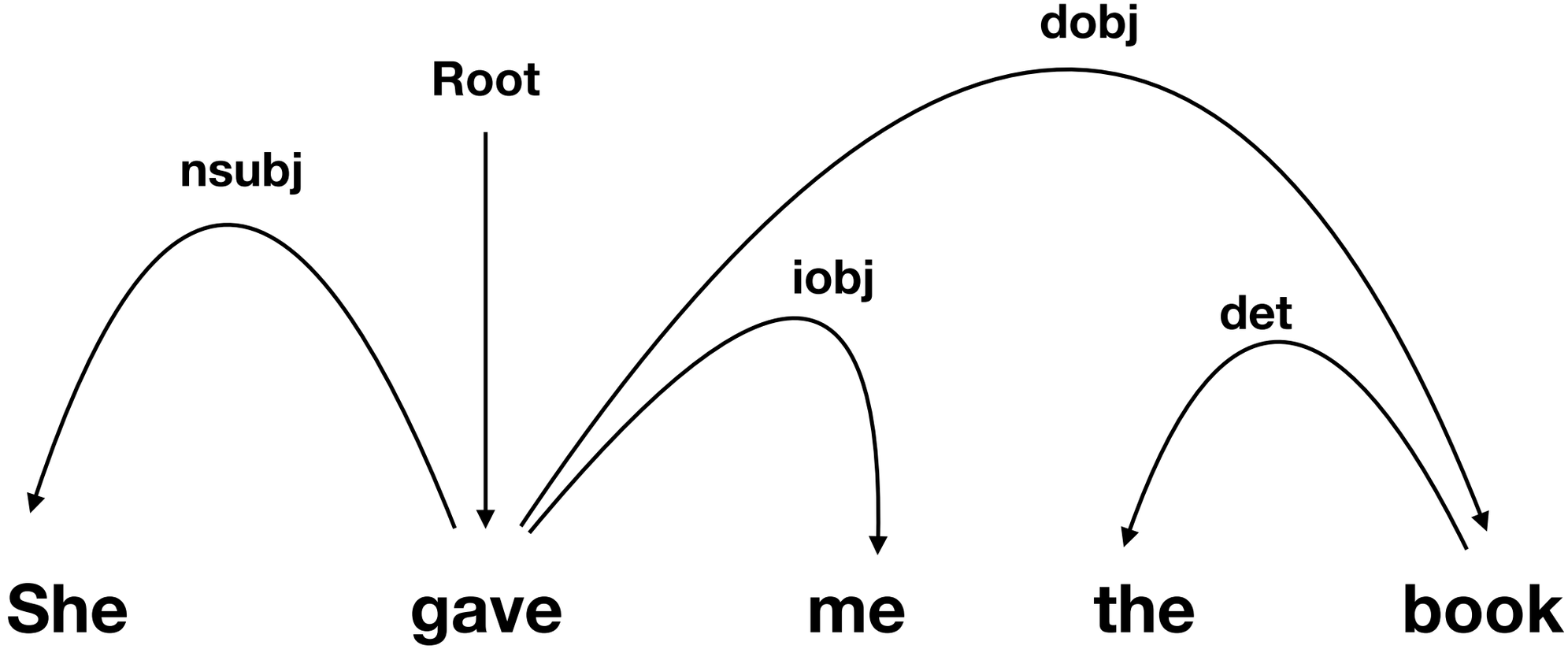
- **SHIFT:** move the word from input buffer to the stack

Arc-Standard Approach

- Note: The root cannot be a dependent, so **LEFTARC** cannot be applied when the root is the second word in the stack

- There must be at least 2 words in the stack to apply **LEFTARC** or **RIGHTARC**

Example of transition-based parsing



[STACK]	[INPUT BUFFER]	[RELATIONS]
[root]	[She, gave, me, the, book]	

[STACK]	[INPUT BUFFER]	[RELATIONS]
[root]	[She, gave, me, the, book]	

We need more words in the stack: SHIFT

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the book]	

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	

She is not the root, We need more words in the stack: SHIFT

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	

Gave is the head of She.
Word on top of stack is head of second word on stack: LEFTARC

Remove She from stack and add relation. No change to input buffer

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)

Gave is the root. We don’t want to remove it from the stack yet because we need to build more relations with the word. So, instead of building an ARC right now between root and gave, we SHIFT

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	

Gave is the head of me. RIGHTARC

Remove second word from stack, add relation, no change to input buffer.

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)

Still don’t want to get rid of ‘gave’

SHIFT

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	

Any relation between ‘gave’ and ‘the’?

No.

SHIFT

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	

‘Book’ is head of ‘the’: LEFTARC

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	
LA	[root, gave, book]		(the <- book)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	
LA	[root, gave, book]		(the <- book)

‘Gave’ is head of ‘book’: RIGHTARC

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	
LA	[root, gave, book]		(the <- book)
RA	[root, gave]		(gave -> book)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	
LA	[root, gave, book]		(the <- book)
RA	[root, gave]	**No more words left in buffer, can finally add ARC between root and gave: RIGHTARC**	(gave -> book)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S	[root, She]	[gave, me, the, book]	
S	[root, She, gave]	[me, the, book]	
LA	[root, gave]	[me, the, book]	(She <- gave)
S	[root, gave, me]	[the, book]	
RA	[root, gave]	[the, book]	(gave -> me)
S	[root, gave, the]	[book]	
S	[root, gave, the, book]	[ ]	
LA	[root, gave, book]		(the <- book)
RA	[root, gave]		(gave -> book)
RA	[root]		(root -> gave)

	[STACK]	[INPUT BUFFER]	[RELATIONS]
	[root]	[She, gave, me, the, book]	
S			
S			
LA			
S			
RA			
S			
S			
LA			
RA			
RA			
	[root]		(root -> gave)

We have (exactly) encoded the parse tree as a sequence of {S, LA, RA} actions!

(Would also need to specify relation labels in the LA, RA actions or post hoc.)

Transition-based parsing = iteratively:
(1) consult the Oracle (algorithm giving next action)
(2) modify the Configuration (state of stack, buffer, relations) according to the action

Arc-Standard Parsing

With the 3 Arc-Standard actions {S, LA, RA}:

☞ How many transitions to parse a sentence of *N* words?

- 2**N*: for each word, once to shift + once to attach to a head and remove from stack (LA or RA).

☞ Can these 3 types of actions build any tree?

- Only projective trees: only adding edges at the top of the stack & permanently removing a word from the stack once attaching it to its parent ensures that all subtrees are contiguous
- With a richer set of actions, can get non-projective trees or even graphs

☞ How would you implement an Oracle (choose the next action at test time)?

- This brings us to…

Statistical Dependency Parsing

Statistical Dependency Parsing

- Can be done by training a classifier to predict each action, using data from Treebanks
  - Neural Networks, SVM, logistic regression
- Advanced Methods: Arc Eager transition system
- MaltParser: linear time parsing, predicted by a discriminative classifier
- **Possible features?** POS tags, word at the top of the stack, etc. — we'll come back to this in a second

Training

- Can automatically convert constituency treebanks (like the Penn Treebank) to dependencies

Training

- Or, can use dependency treebanks like **Universal Dependencies** (available in many languages)
    - http://universaldependencies.org

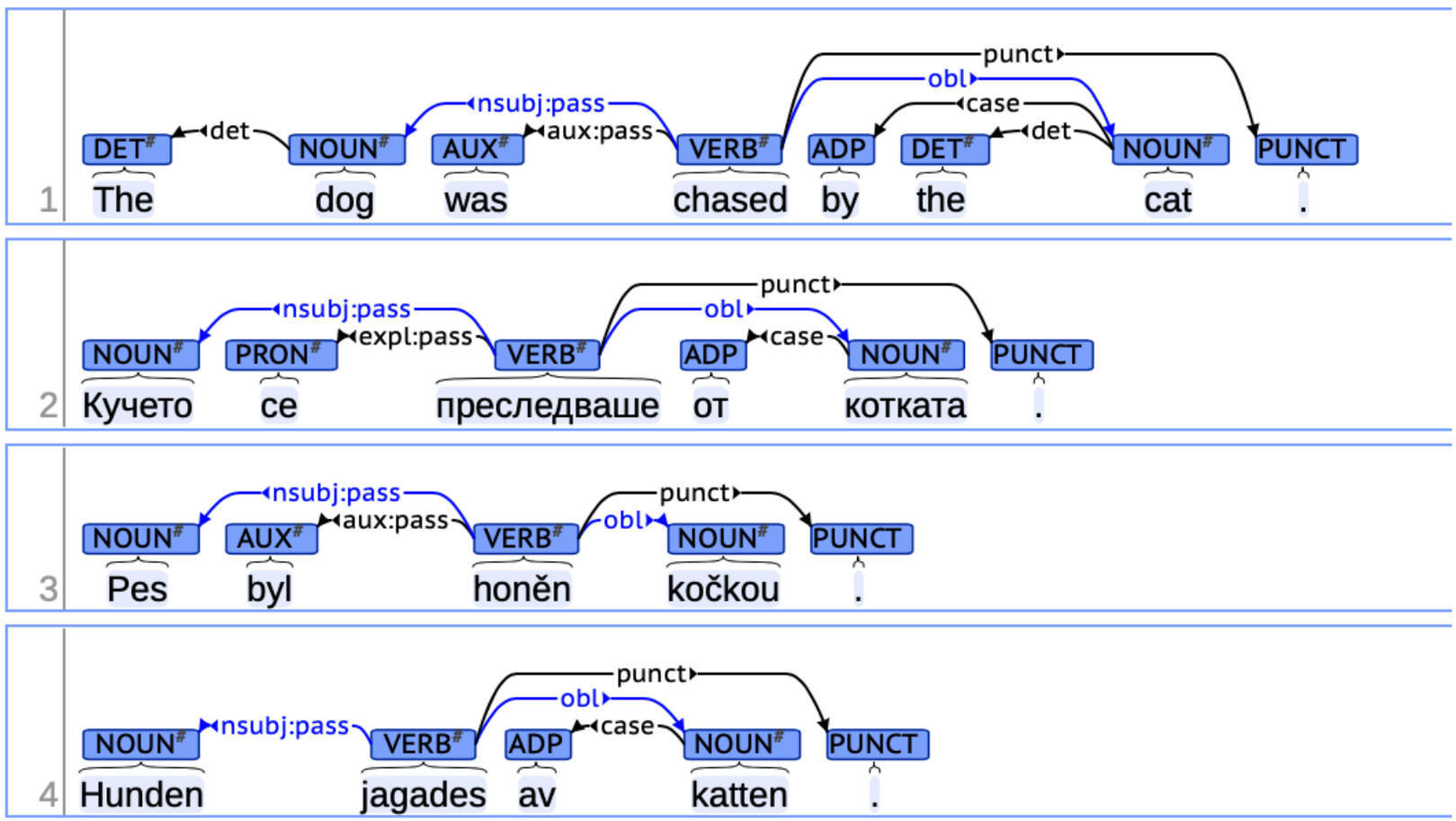
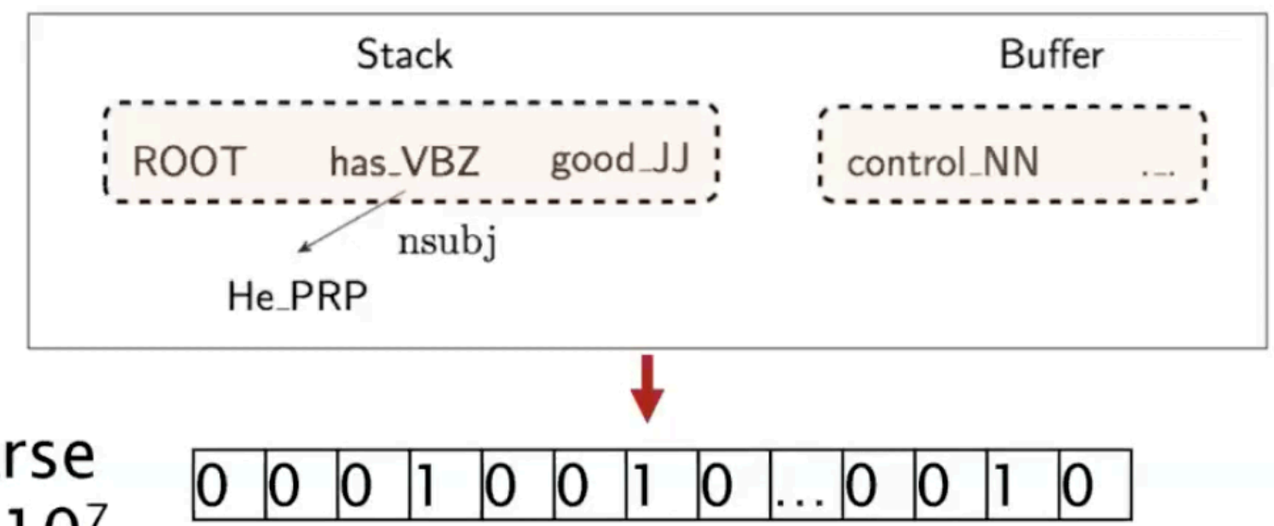


Image from https://universaldependencies.org/introduction.html

Evaluation

- Comparing against a gold standard:
    - **Accuracy** = # correct dependencies / # of dependencies
    - **Unlabeled Attachment Score** (UAS): % of words attached correctly (correct head)
    - **Labeled Attached Score** (LAS): % of words attached to the correct head with the correct relation label

Conventional Feature Representation



binary, sparse
dim =10^6 ~ 10^7

0	0	0	1	0	0	1	0	…	0	0	1	0

Feature templates: usually a combination of 1 ~ 3 elements from the configuration.

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$

$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$

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

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

- Sparse vector
- Train classifier on these vectors

Image from Chris Manning's Stanford NLP slides

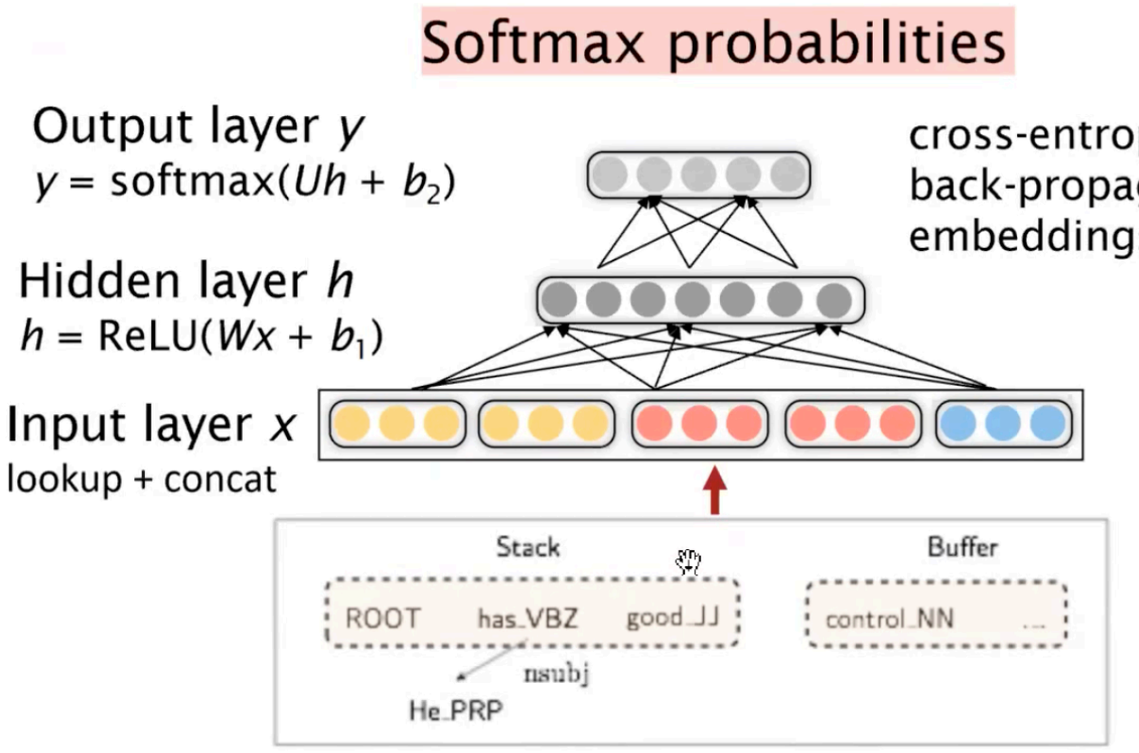
Why train a neural dependency parser?

- Problems with conventional way:
  - Sparse
  - Incomplete
  - Expensive computation
- Instead, you can learn a dense and compact feature representation (like word2vec)

Neural dependency parsing

- Chen and Manning 2014: https://www.aclweb.org/anthology/D14-1082/

- Represents words as d-dimensional dense vectors (word embeddings!)



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**

Image from Chris Manning's Stanford NLP slides

Further developments in neural dependency parsing

This work was further developed and improved by others, including in particular at Google

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global, conditional random field (CRF)-style inference over the decision sequence

Leading to SyntaxNet and the Parsey McParseFace model

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

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

Slide from Chris Manning's Stanford NLP slides

Biaffine Parser

- A neural model a lot of people start with now is Dozat & Manning (2017)
  - https://arxiv.org/abs/1611.01734

	Catalan		Chinese		Czech	
Model	**UAS**	**LAS**	**UAS**	**LAS**	**UAS**	**LAS**
Andor et al.	92.67	89.83	84.72	80.85	88.94	84.56
Deep Biaffine	**94.69**	**92.02**	**88.90**	**85.38**	**92.08**	**87.38**

	English		German		Spanish	
Model	**UAS**	**LAS**	**UAS**	**LAS**	**UAS**	**LAS**
Andor et al.	93.22	91.23	90.91	89.15	92.62	89.95
Deep Biaffine	**95.21**	**93.20**	**93.46**	**91.44**	**94.34**	**91.65**

Table 5: Results on the CoNLL '09 shared task datasets

Selected recent works

Universal Dependencies [linguistic motivation]

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, Daniel Zeman

Meta-learning for fast cross-lingual adaptation in dependency parsing

Anna Langedijk, Verna Dankers, Phillip Lippe, Sander Bos, Bryan Cardenas Guevara, Helen Yannakoudakis, Ekaterina Shutova

A Graph-based Model for Joint Chinese Word Segmentation and Dependency Parsing

Hang Yan, Xipeng Qiu, Xuanjing Huang

Improved Dependency Parsing using Implicit Word Connections Learned from Unlabeled Data

Wenhui Wang, Baobao Chang, Mairgup Mansur

Universal Dependencies v2: An evergrowing multilingual treebank collection [overview & data]

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, Daniel Zeman

Zero-Shot Cross-Lingual Dependency Parsing through Contextual Embedding Transformation

Haoran Xu, Philipp Koehn

Graph Convolution over Pruned Dependency Trees Improves Relation Extraction

Yuhao Zhang, Peng Qi, Christopher D. Manning

Dependency Parsing of Code-Switching Data with Cross-Lingual Feature Representation

Niko Partanen, Kyungtae Lim, Michael Rießler, Thierry Poibeau

Demos

- **Stanza** (from Stanford): http://stanza.run/

- **AllenNLP** (from Allen Institute for AI): https://demo.allennlp.org/dependency-parsing

Practice

1. For the sentence *a b c d e*, what parse would the action sequence **S S S S RA RA LA S LA RA** correspond to?

2. For a length-*N* sentence, what is the *O*(·) complexity of PCKY Constituency Parsing vs. Arc-Standard Transition-Based Dependency Parsing?
   - Space complexity: how large are the data structures?
   - Time complexity: how many choices do we have to consider?
   - Hint: You can introduce constants for relevant factors besides *N*.

3. Can Arc-Standard Transition-Based Parsing build any kind of tree (given the right action sequence)? Explain.

Group 1: Q1 | Group 2: Q2 for CKY | Group 3: Q2 for transition-based | Group 4: Q3