CS388: Natural Language Processing

Lecture 12: Dependency I

Greg Durrett

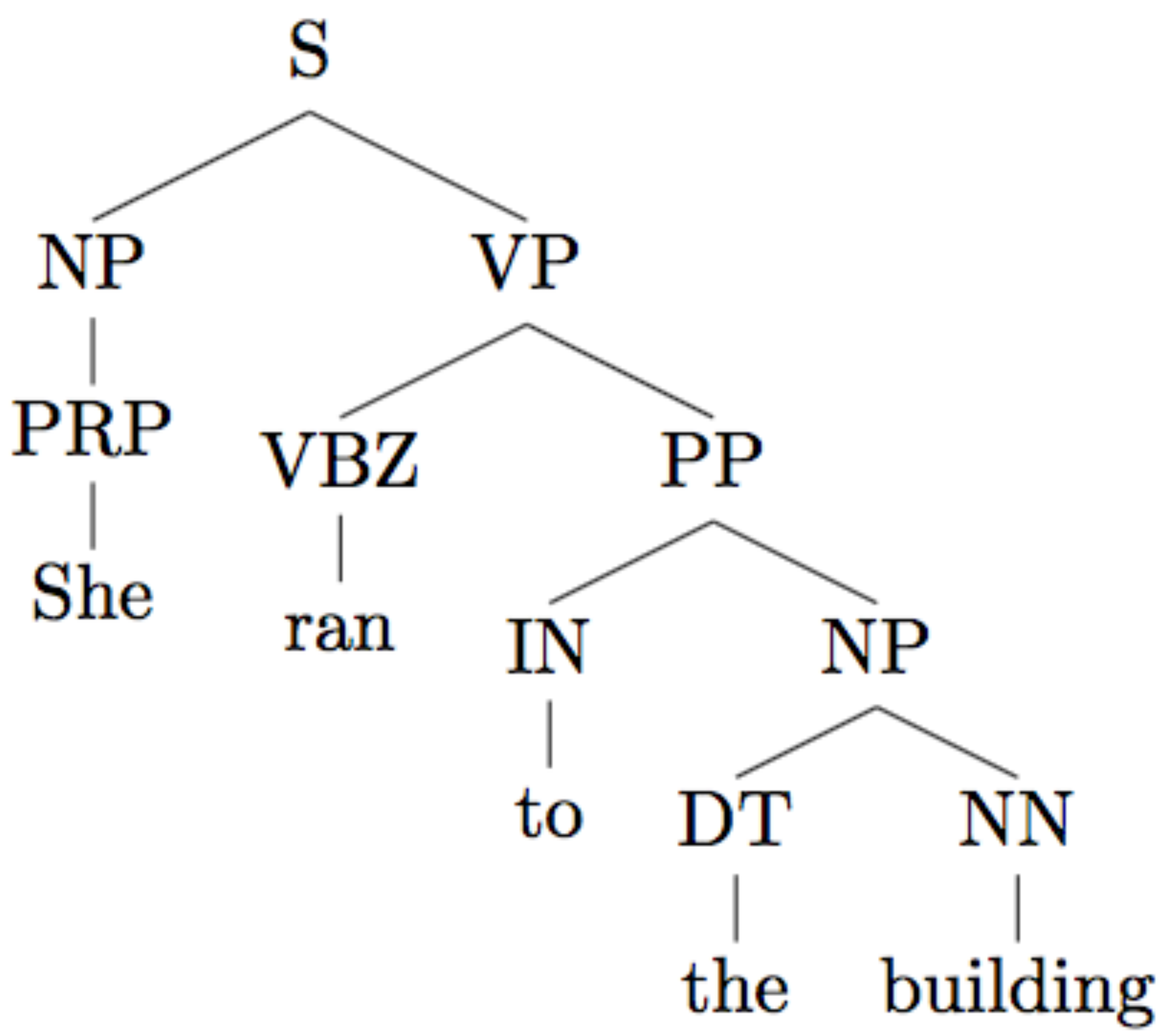
TEXAS
The University of Texas at Austin

Administrivia

- Project 1 graded, discussion at end of lecture
- Mini 2 due tonight
- Final project proposals due next Tuesday

Recall: Constituency

- Tree-structured syntactic analyses of sentences
- Nonterminals (NP, VP, etc.) as well as POS tags (bottom layer)
- Structured is defined by a CFG

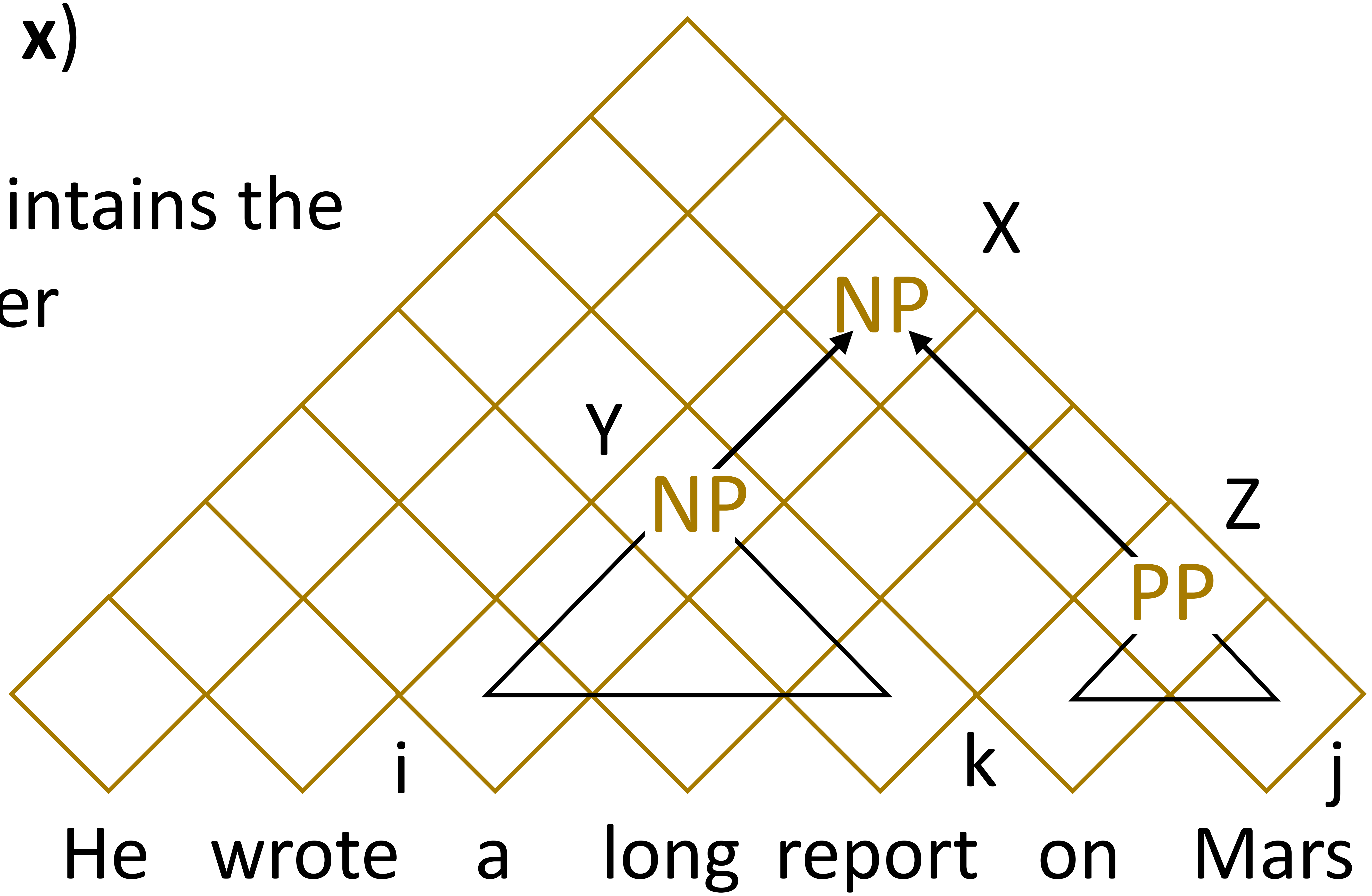


Recall: CKY

- Find argmax P(T|**x**) = argmax P(T, **x**)

- Dynamic programming: chart maintains the best way of building symbol X over span (i, j)

- Loop over all split points k, apply rules X -> Y Z to build X in every possible way



Cocke-Kasami-Younger

Recall: Top-down Parsing

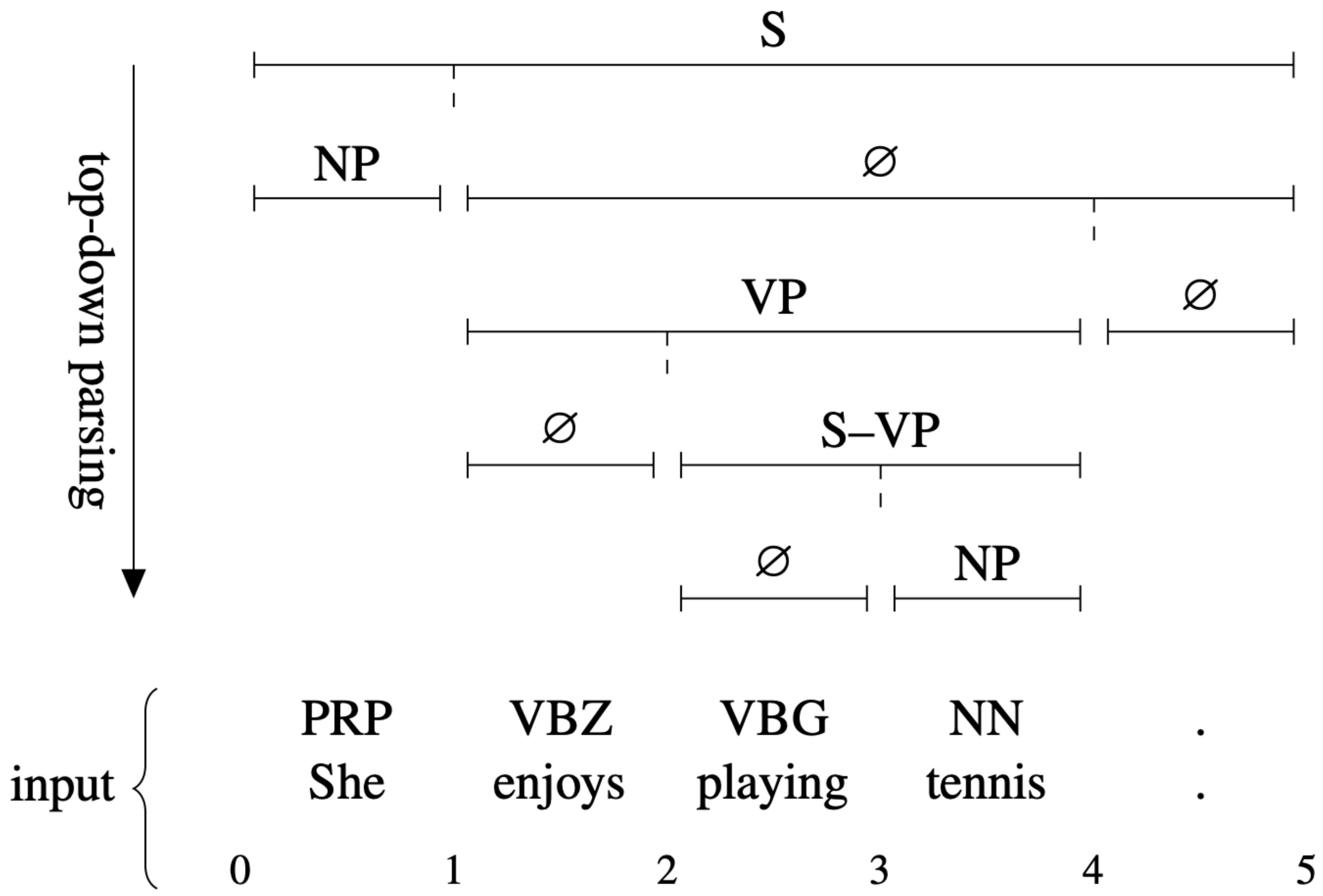
- Can score split points and also labels
- Dynamic programming version:

$$s_{\text{best}}(i,j) = \max_{\ell,k} \left[s_{\text{label}}(i,j,\ell) + \tilde{s}_{\text{split}}(i,k,j) \right]$$

(best way of building *i* and *j* involves maxing over split point and a *single* label)

- Greedy top-down version: at each stage, predict split point *k* and label *l*

$$(\widehat{\ell}, \widehat{k}) = \operatorname*{argmax}_{\ell,k} \left[s_{\text{label}}(i,j,\ell) + s_{\text{split}}(i,k,j) \right]$$



(a) Execution of the top-down parsing algorithm.

Outline

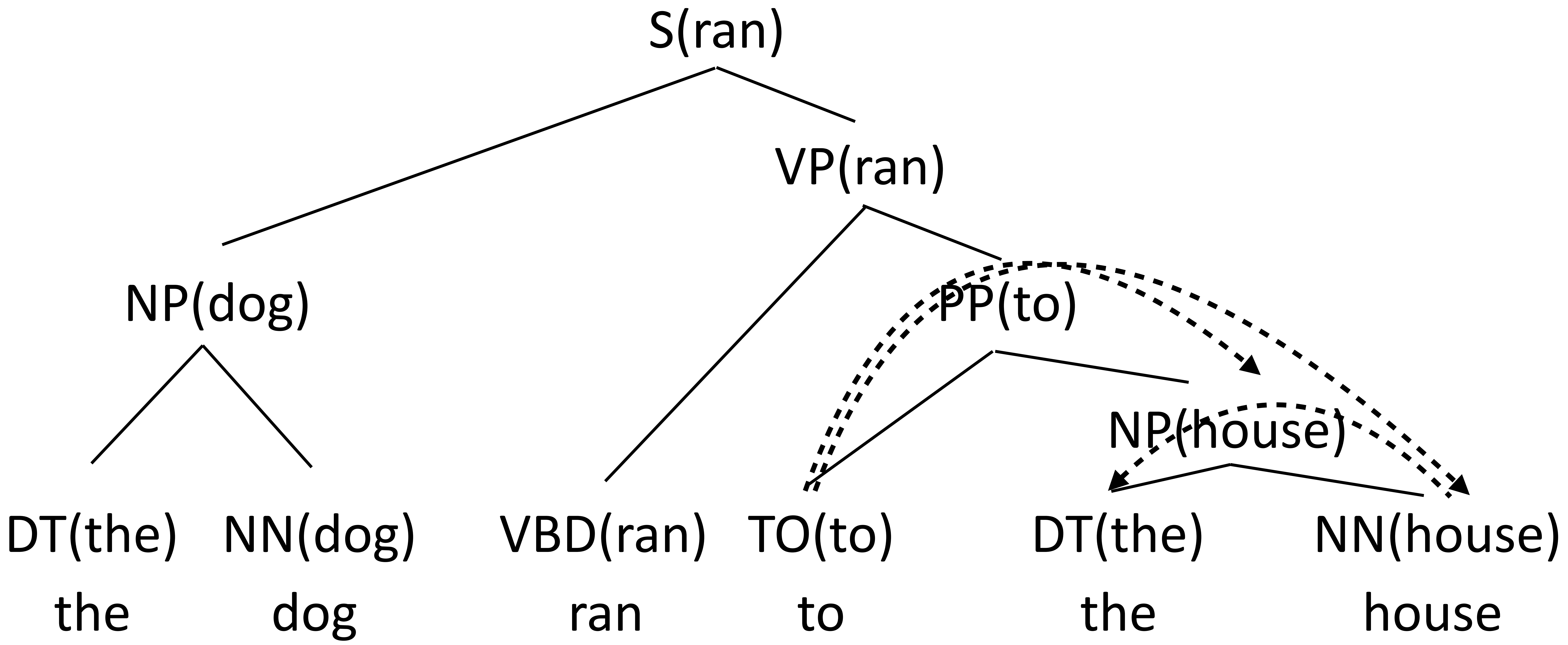
- Dependency representation, contrast with constituency

- Projectivity

- Graph-based dependency parsers

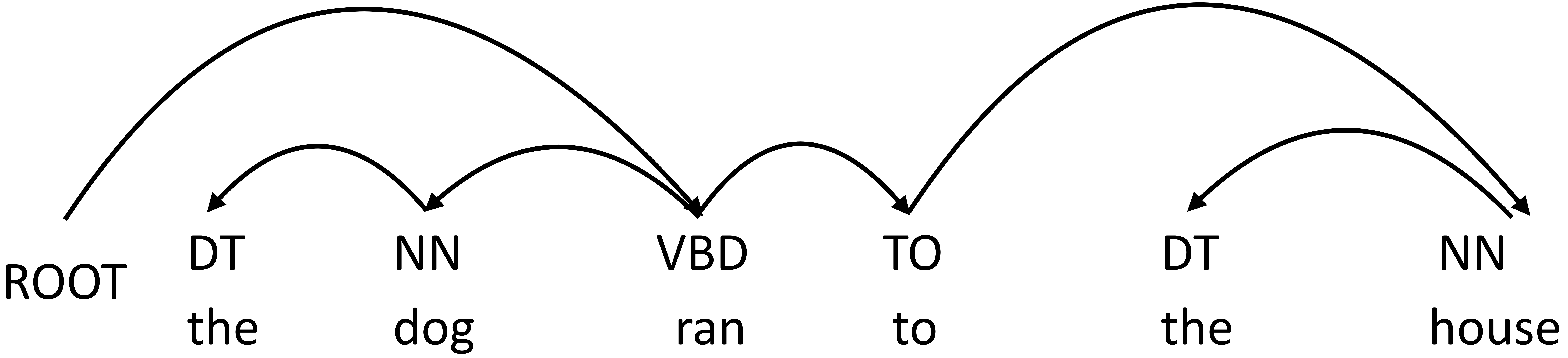
Dependency Representation

Lexicalized Parsing



Dependency Parsing

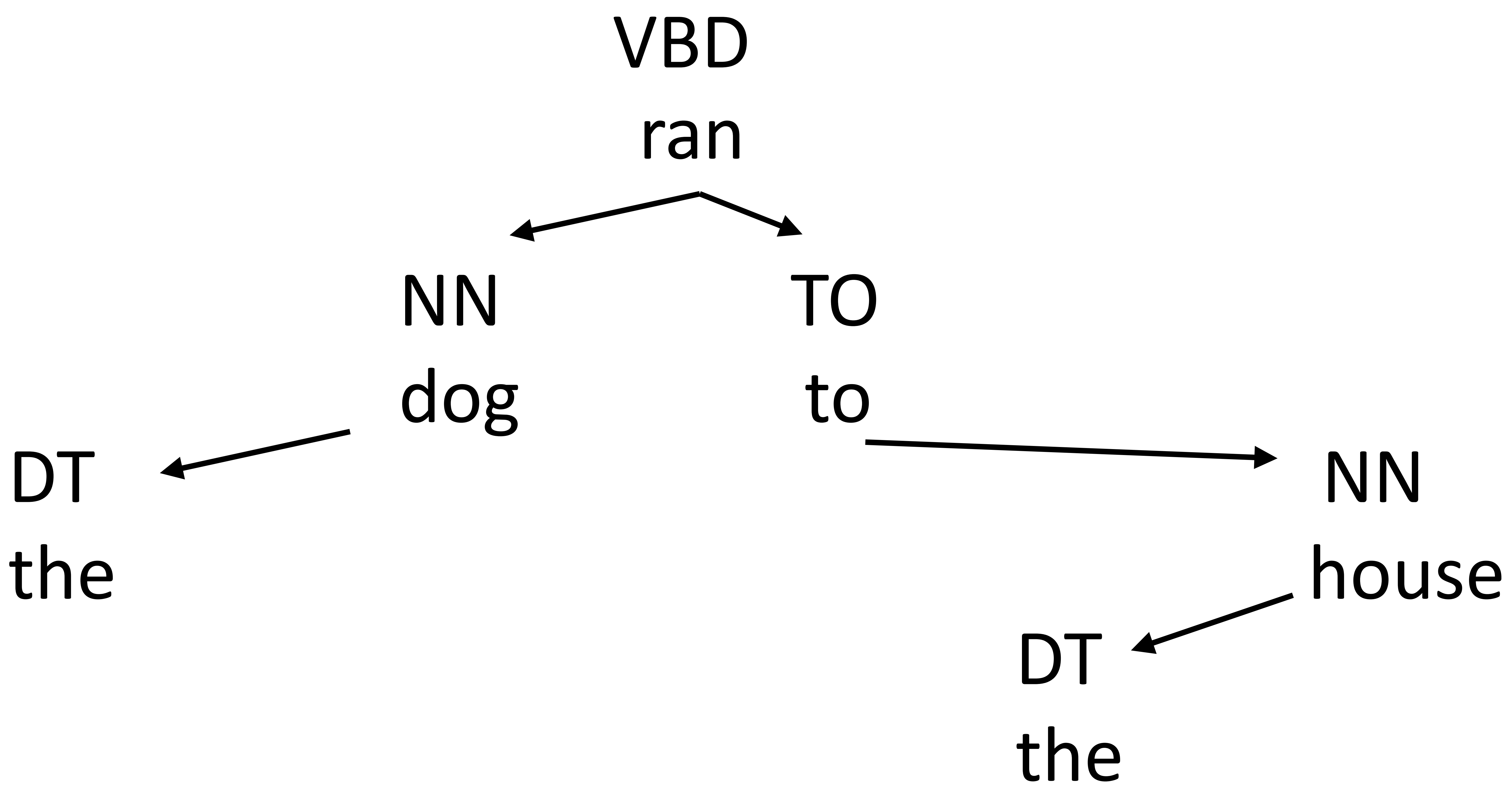
- Dependency syntax: syntactic structure is defined by these arcs
  - Head (parent, governor) connected to dependent (child, modifier)
  - Each word has exactly one parent except for the ROOT symbol, dependencies must form a directed acyclic graph



- POS tags same as before, usually run a tagger first as preprocessing

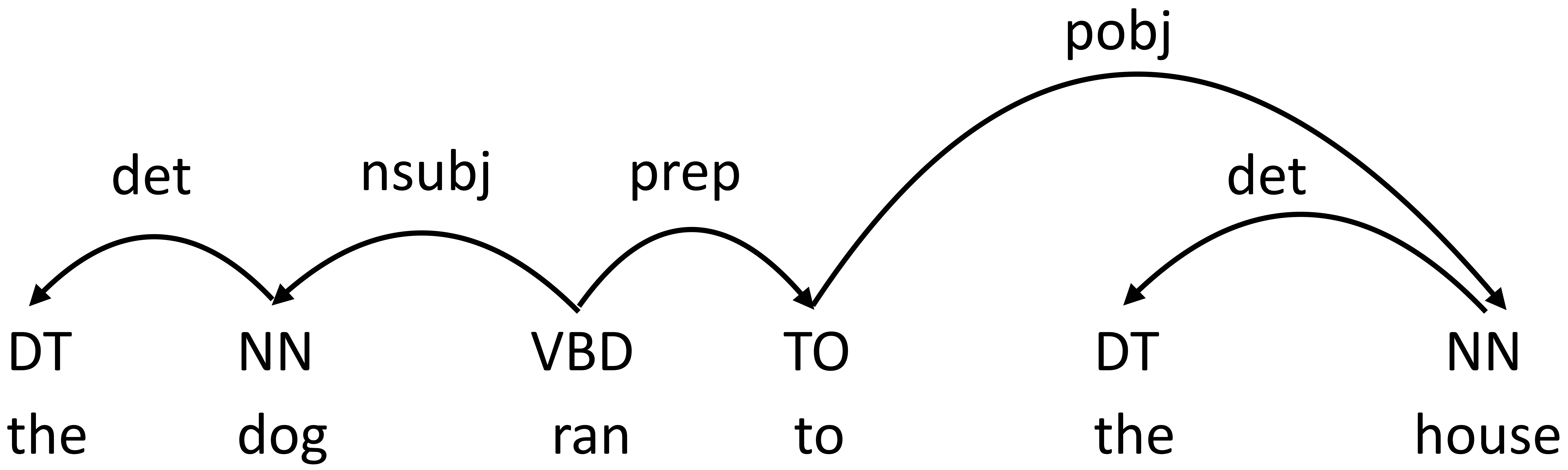
Dependency Parsing

- Still a notion of hierarchy! Subtrees often align with constituents



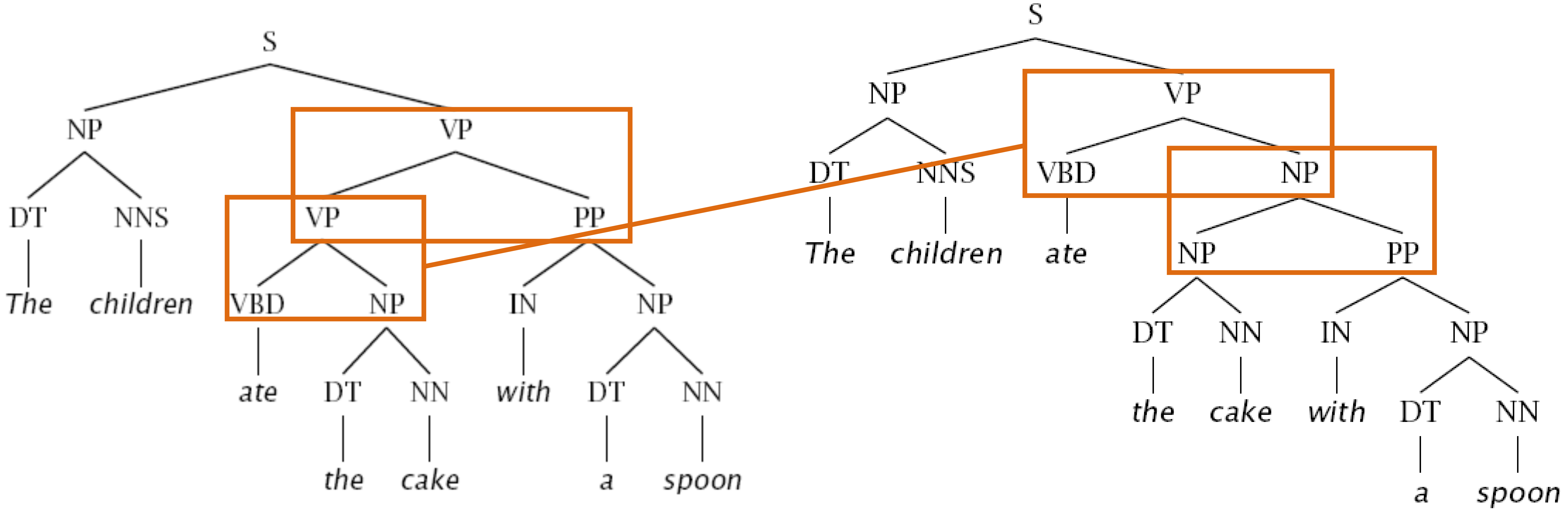
Dependency Parsing

- Can label dependencies according to syntactic function
- Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)



Dependency vs. Constituency: PP Attachment

- Constituency: several rule productions need to change



Dependency vs. Constituency: PP Attachment

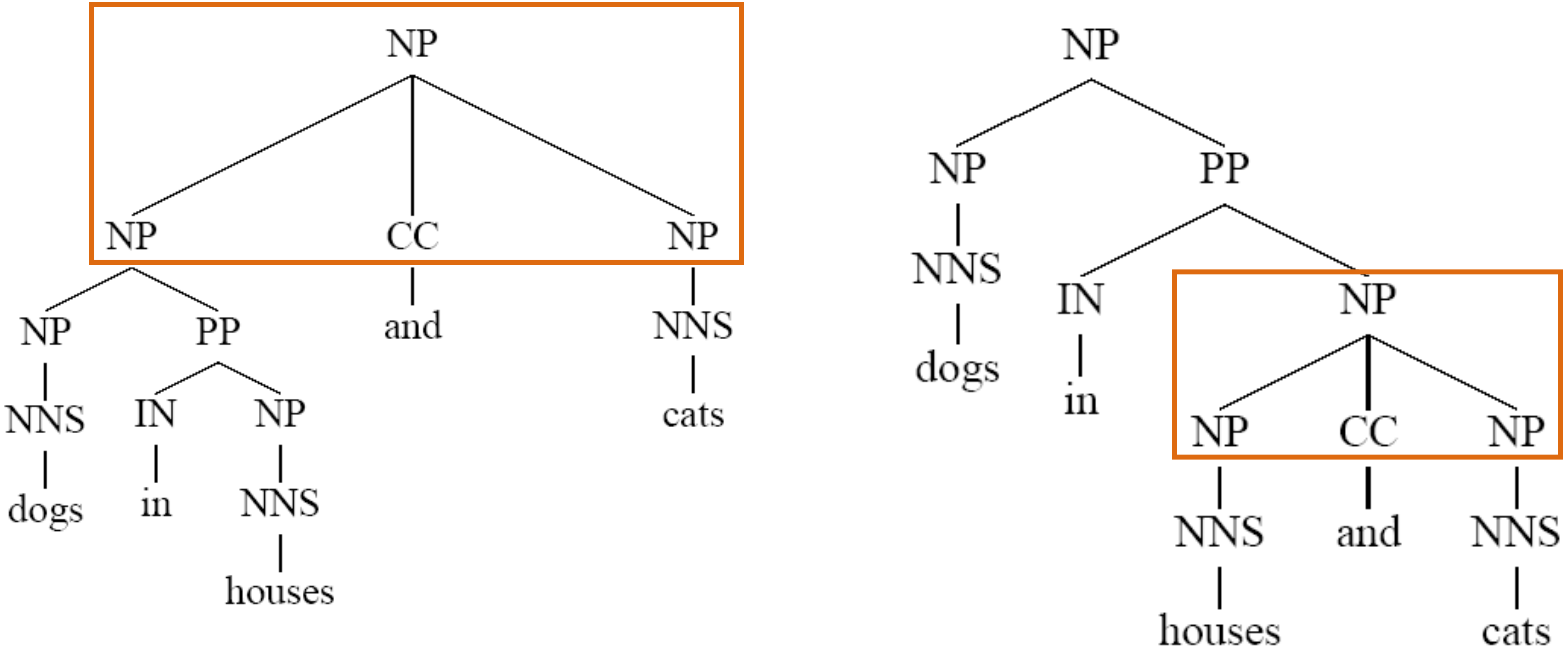
- Dependency: one word (with) assigned a different parent

the children ate the cake with a spoon

- More predicate-argument focused view of syntax
- “What’s the main verb of the sentence? What is its subject and object?” — easier to answer under dependency parsing

Dependency vs. Constituency: Coordination

- Constituency: ternary rule NP -> NP CC NP



Dependency vs. Constituency: Coordination

- Dependency: first item is the head

dogs in houses **and cats**

[dogs in houses] and cats

dogs in **houses and cats**

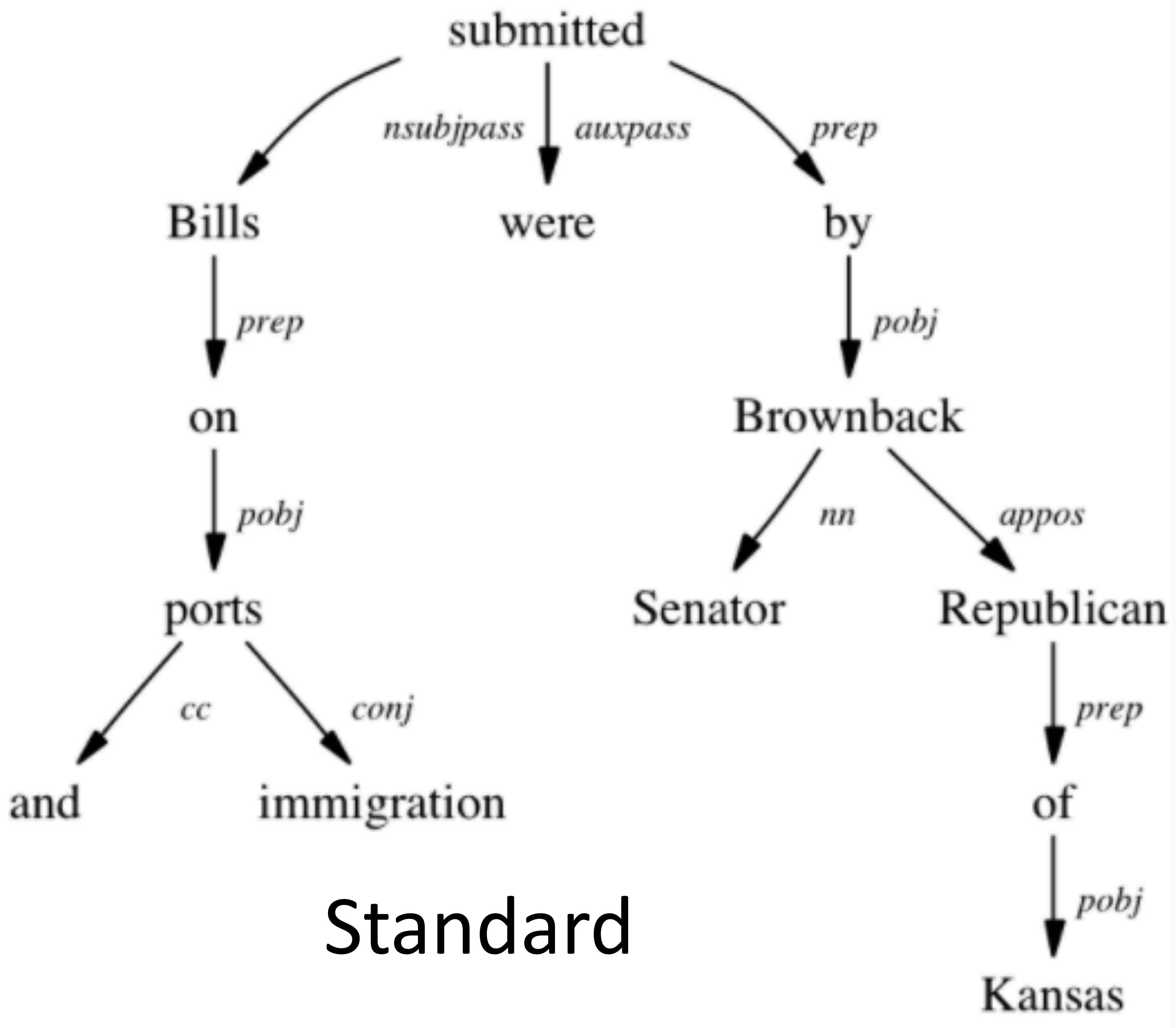
dogs in [houses and cats]

- Coordination is decomposed across a few arcs as opposed to being a single rule production as in constituency
- Can also choose *and* to be the head
- In both cases, headword doesn't really represent the phrase — constituency representation makes more sense

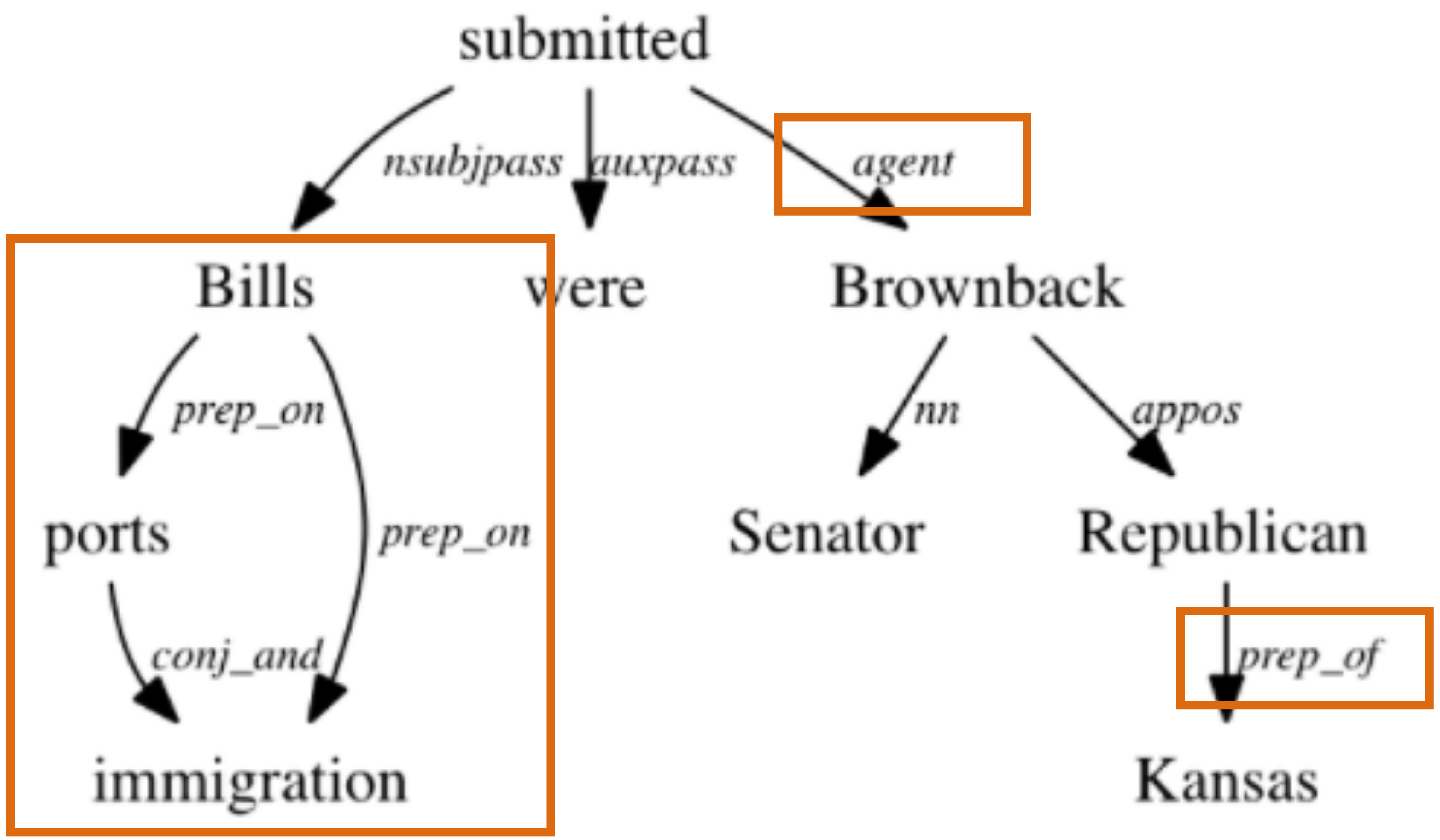
Stanford Dependencies

- Designed to be practically useful for relation extraction

Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas



Standard



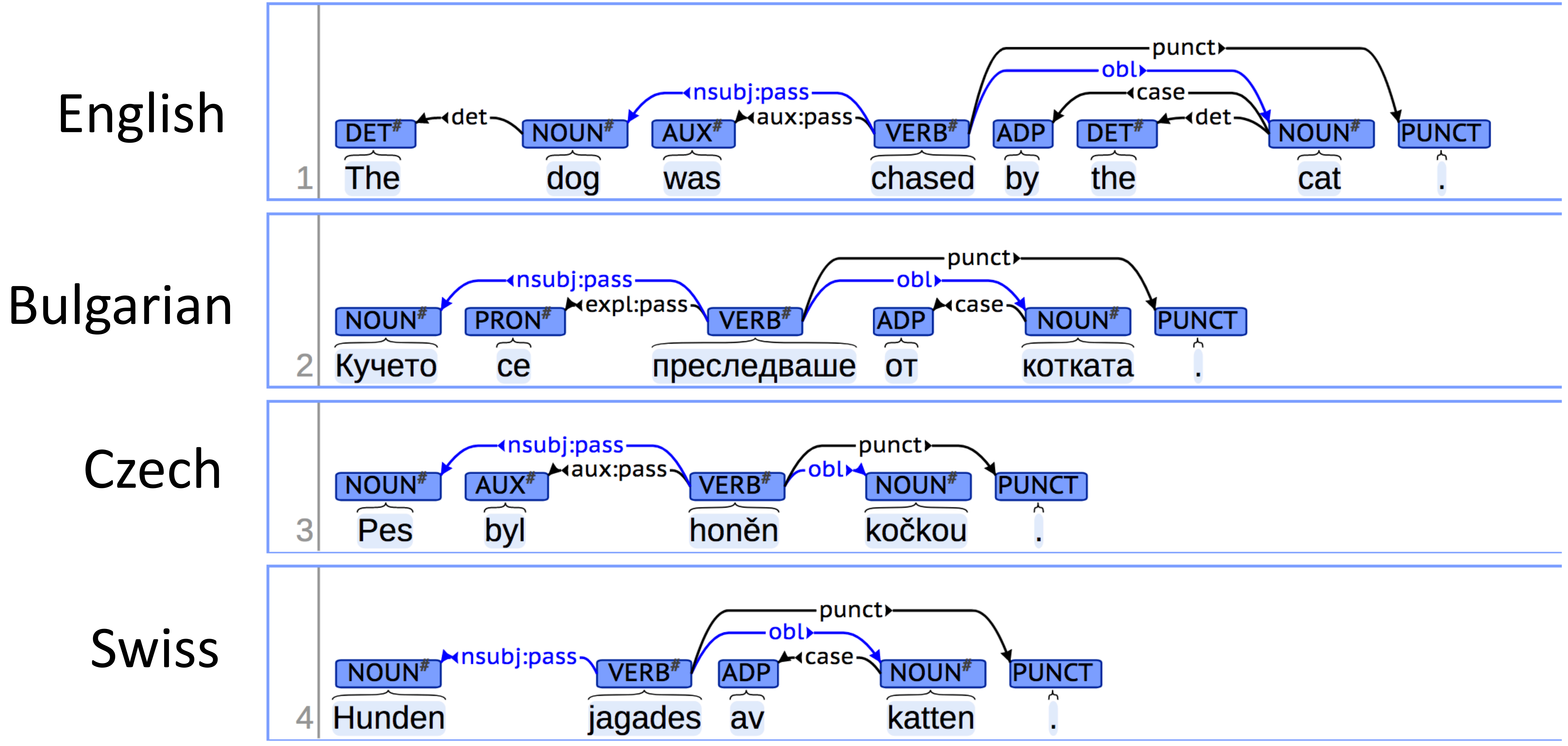
Collapsed

Dependency vs. Constituency

- Dependency is often more useful in practice (models predicate argument structure)
- Slightly different representational choices:
  - PP attachment is better modeled under dependency
  - Coordination is better modeled under constituency
- Dependency parsers are easier to build: no "grammar engineering", no unaries, easier to get structured discriminative models working well
- Dependency parsers are usually faster
- Dependencies are more universal cross-lingually

Universal Dependencies

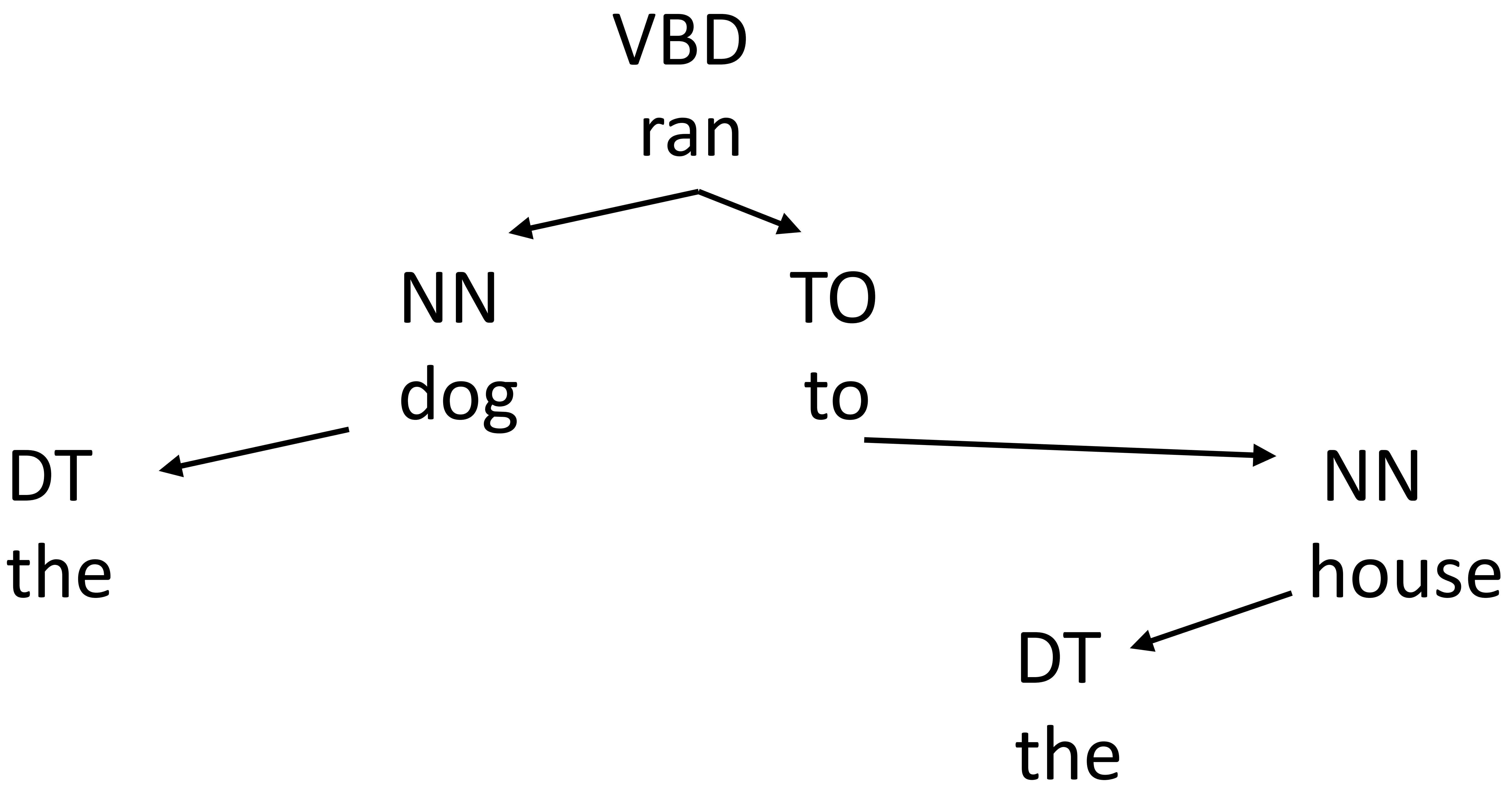
- Annotate dependencies with the same representation in many languages



http://universaldependencies.org/

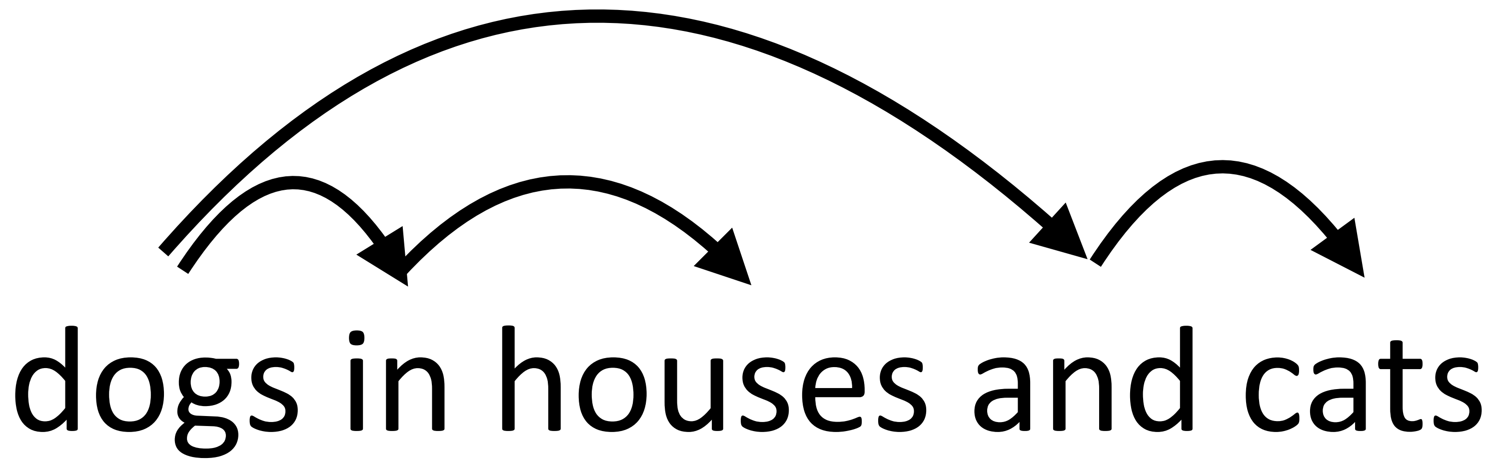
Projectivity

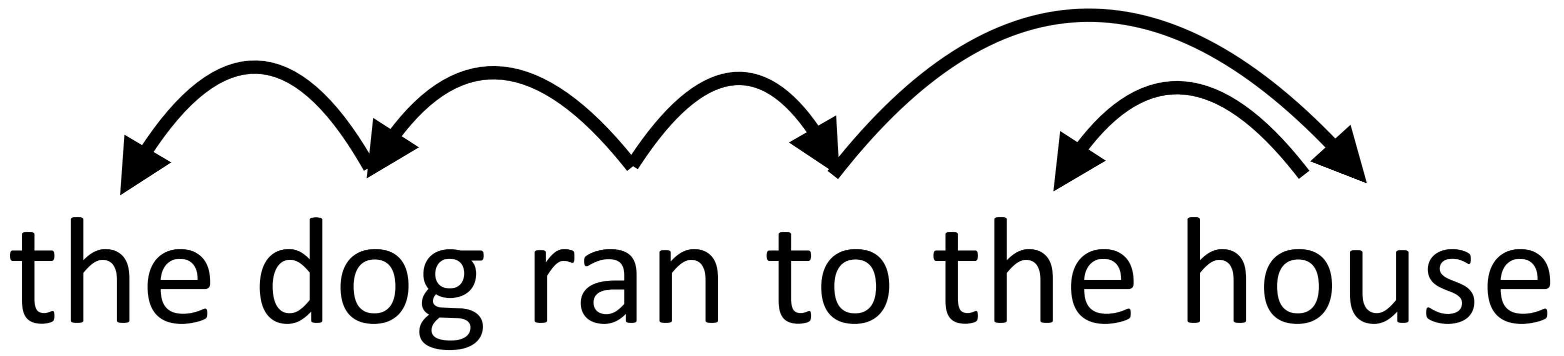
- Any subtree is a contiguous span of the sentence <-> tree is *projective*



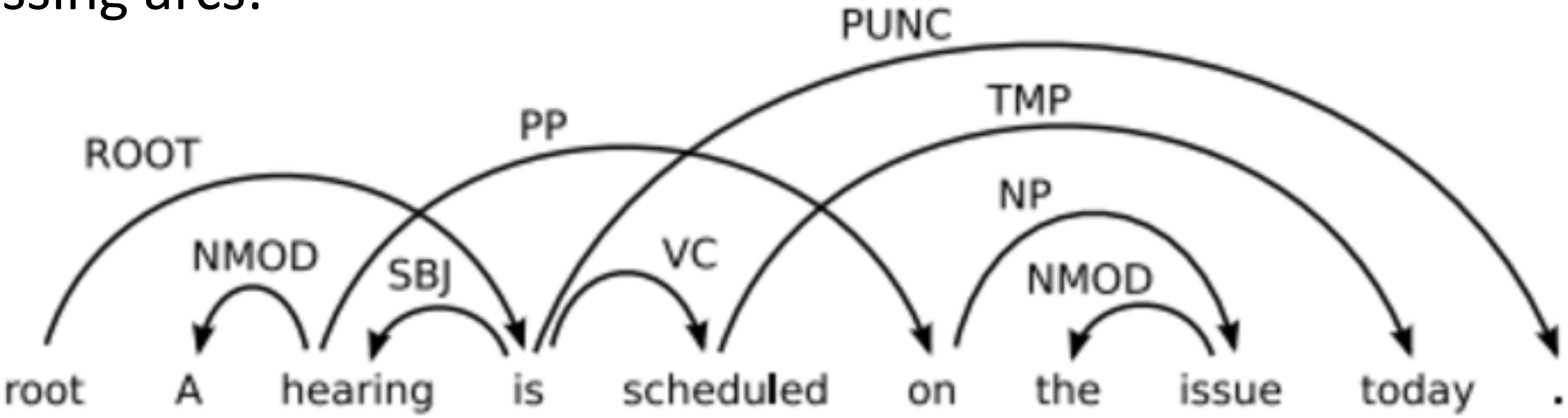
Projectivity

- Projective <-> no “crossing” arcs



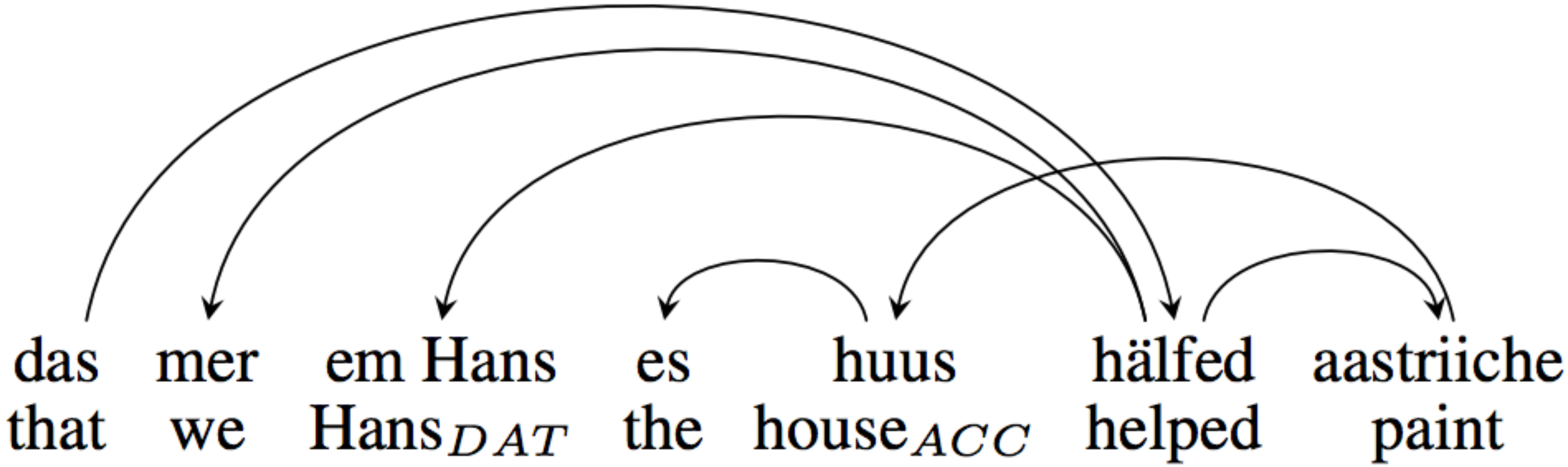


- Crossing arcs:



credit: Language Log

Projectivity in other languages



- Swiss German example

- (Swiss German also has famous non-context-free constructions)

credit: Pitler et al. (2013)

Projectivity

- Number of trees produceable under different formalisms

	Arabic	Czech	Danish
Projective	1297 (88.8)	55872 (76.8)	4379 (84.4)
Sentences	1460	72703	5190

- Many trees in other languages are nonprojective

Pitler et al. (2013)

Projectivity

- Number of trees produceable under different formalisms

	Arabic	Czech	Danish
1-Endpoint-Crossing	1457 (99.8)	71810 (98.8)	5144 (99.1)
Well-nested, block degree 2	1458 (99.9)	72321 (99.5)	5175 (99.7)
Gap-Minding	1394 (95.5)	70695 (97.2)	4985 (96.1)
Projective	1297 (88.8)	55872 (76.8)	4379 (84.4)
Sentences	1460	72703	5190

- Many trees in other languages are nonprojective
- Some other formalisms (that are harder to parse in), most useful one is 1-Endpoint-Crossing

Pitler et al. (2013)

Graph-Based Parsing

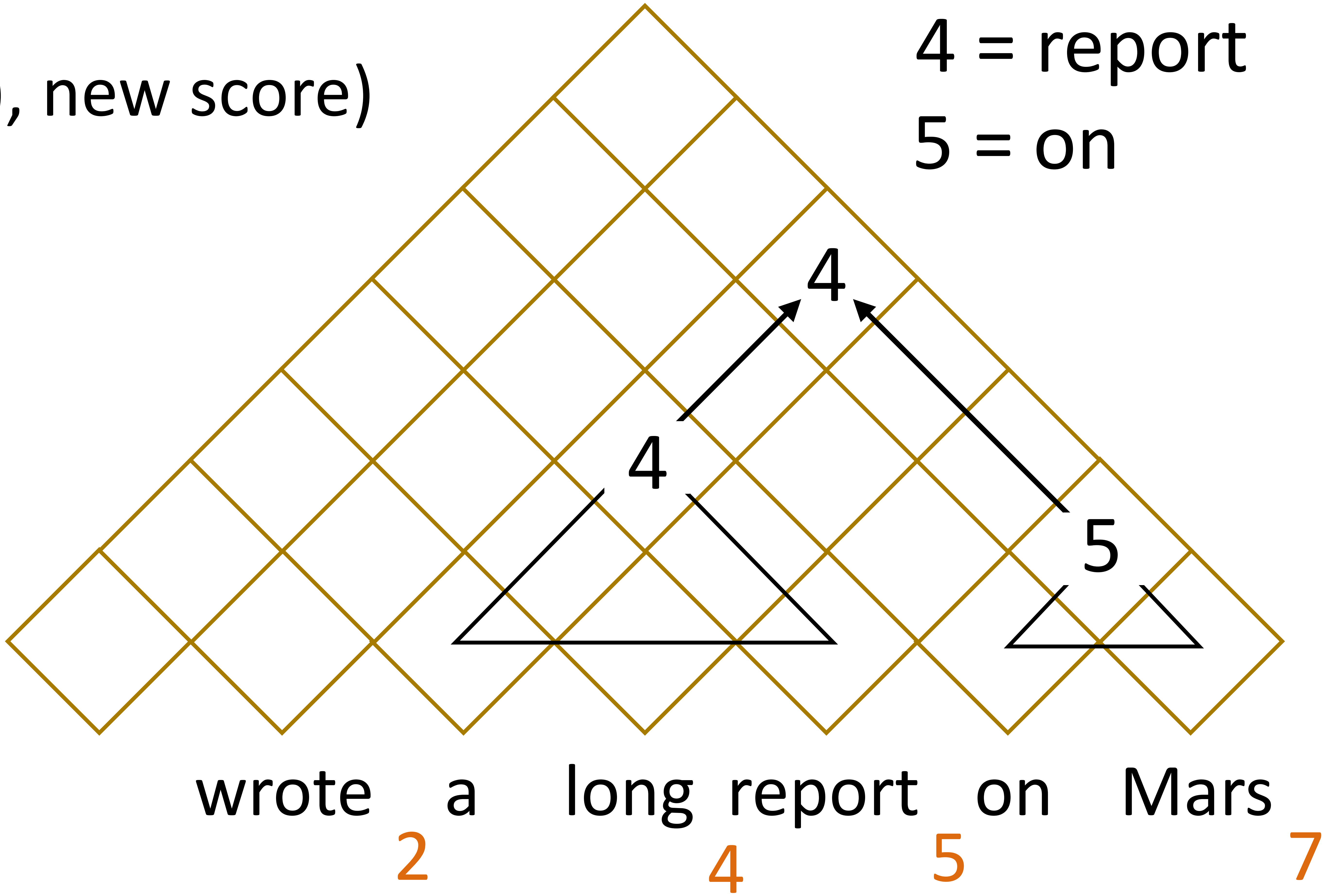
Defining Dependency Graphs

- Words in sentence **x**, tree T is a collection of directed edges (parent(i), i) for each word i
  - Parsing = identify parent(i) for each word
  - Each word has exactly one parent. Edges must form a projective tree
- Log-linear CRF (discriminative): $P(T|\mathbf{x}) = \exp\left(\sum_i w^\top f(i, \mathrm{parent}(i), \mathbf{x})\right)$
- Example of a feature = I[head=*to* & modifier=*house*] (more in a few slides)

ROOT the dog ran to the house

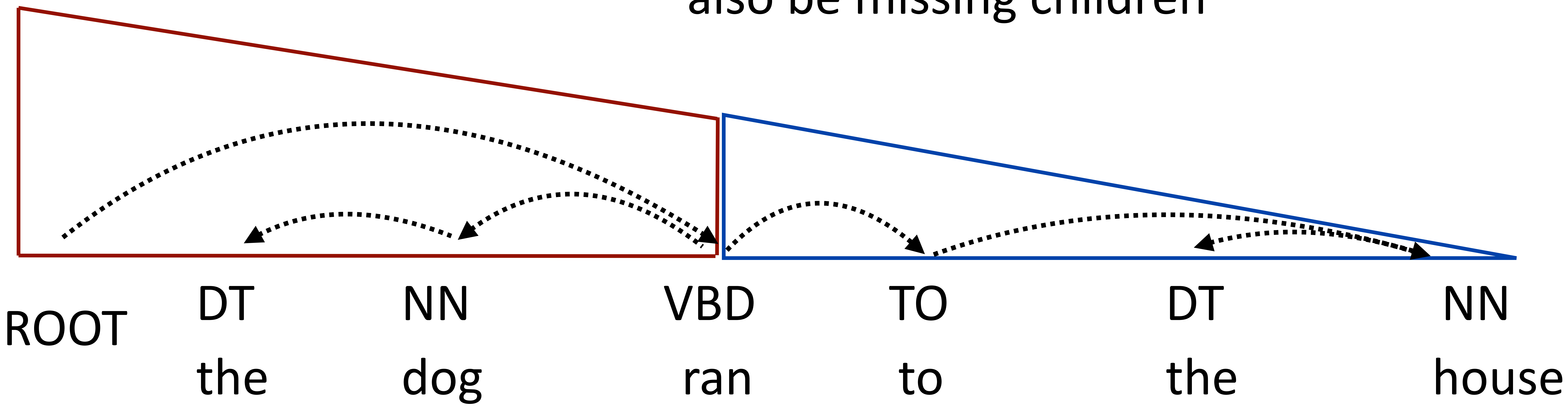
Generalizing CKY

- DP chart with three dimensions: start, end, and head, start <= head < end
- new score = chart(2, 5, 4) + chart(5, 7, 5) + edge score(4 -> 5)
- score(2, 7, 4) = max(score(2, 7, 4), new score)
- Time complexity of this?
- Many *spurious derivations*: can build the same tree in many ways...need a better algorithm



Eisner's Algorithm: $O(n^3)$

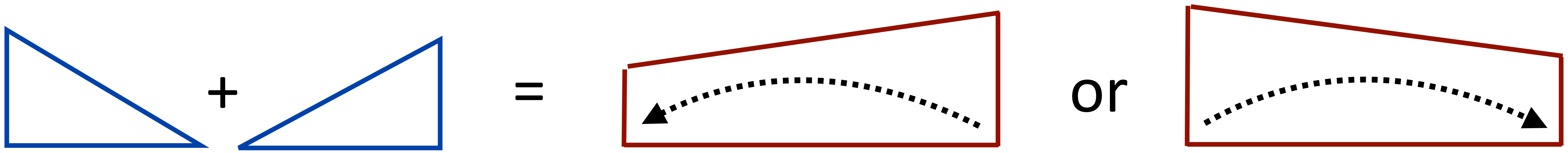
- Cubic-time algorithm
- Maintain two dynamic programming charts with dimension [n, n, 2]:
  - Complete items: head is at "tall end", may be missing children on tall side
  - Incomplete items: arc from "tall" to "short" end, word on short end may also be missing children



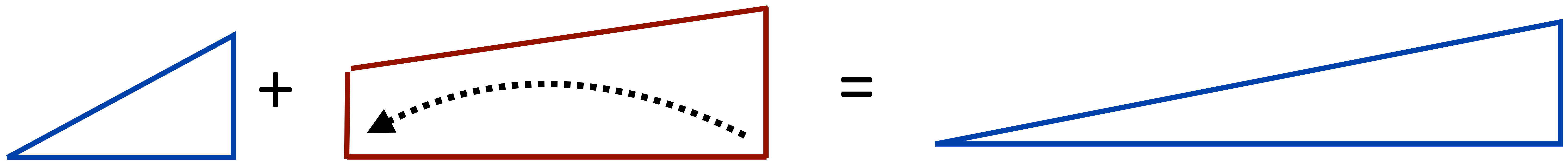
Eisner's Algorithm: O(n³)

- Complete item: all children are attached, head is at the "tall end"
- Incomplete item: arc from "tall end" to "short end", may still expect children

- Take two adjacent complete items, add arc and build incomplete item



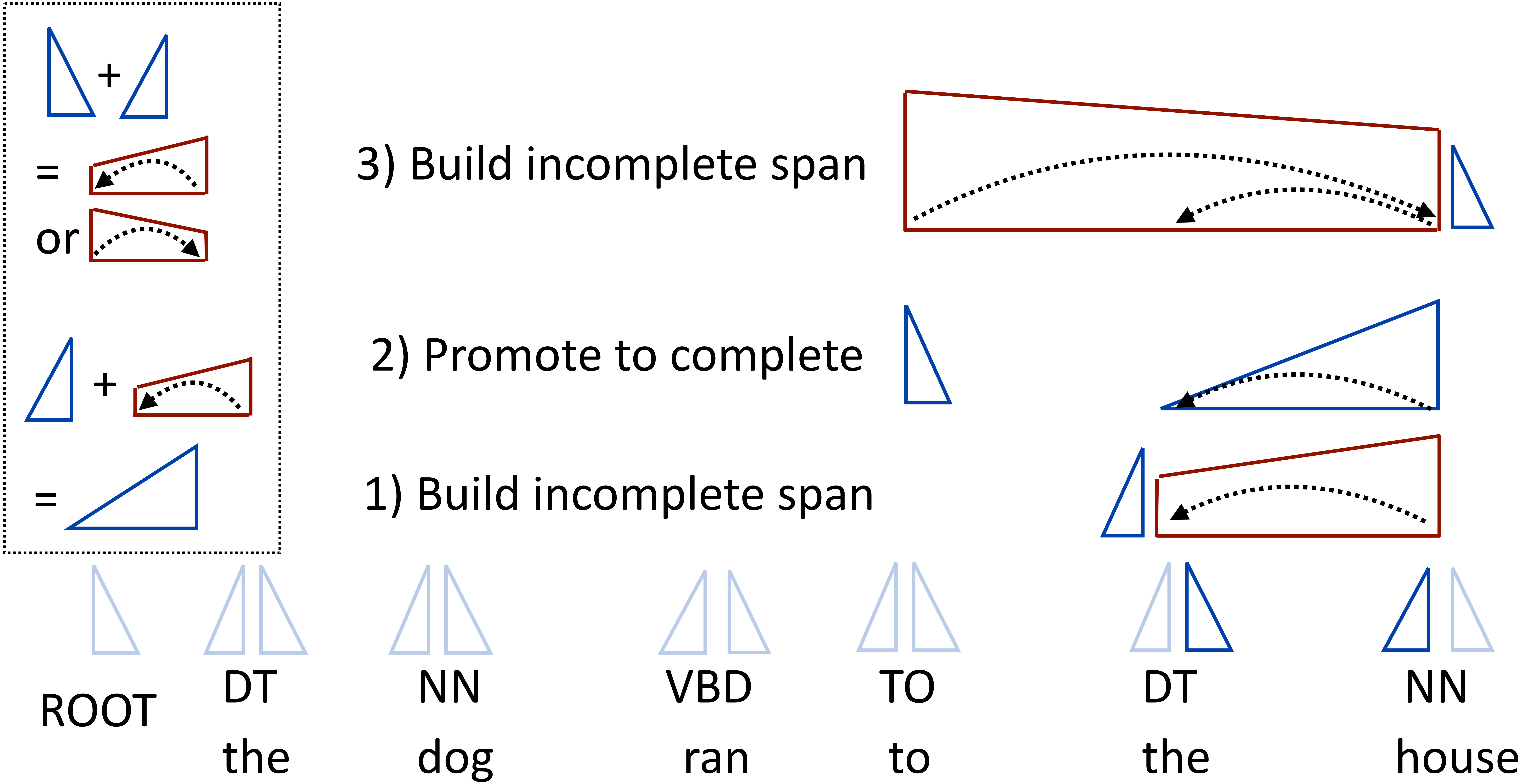
- Take an incomplete item, complete it



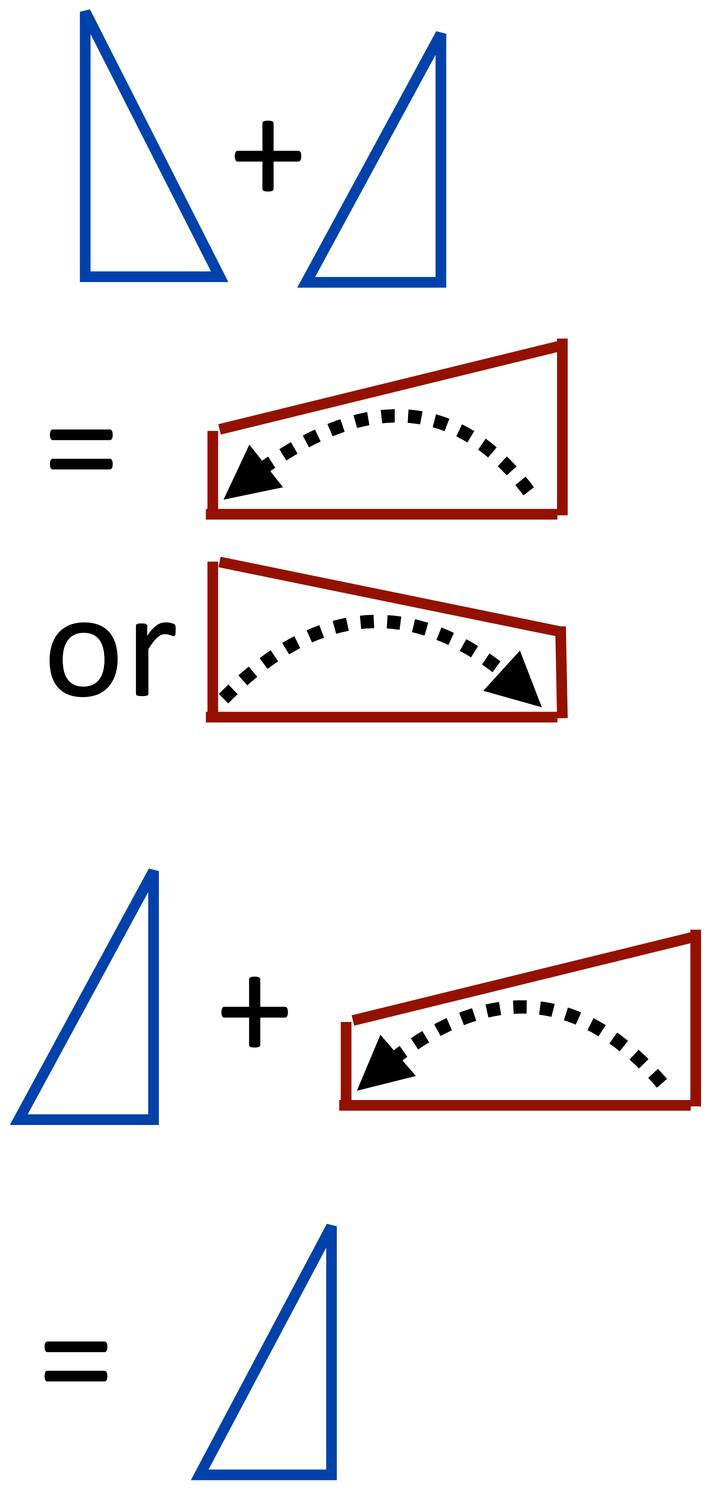
(other case is symmetric)

ROOT	DT	NN	VBD	TO	DT	NN
	the	dog	ran	to	the	house

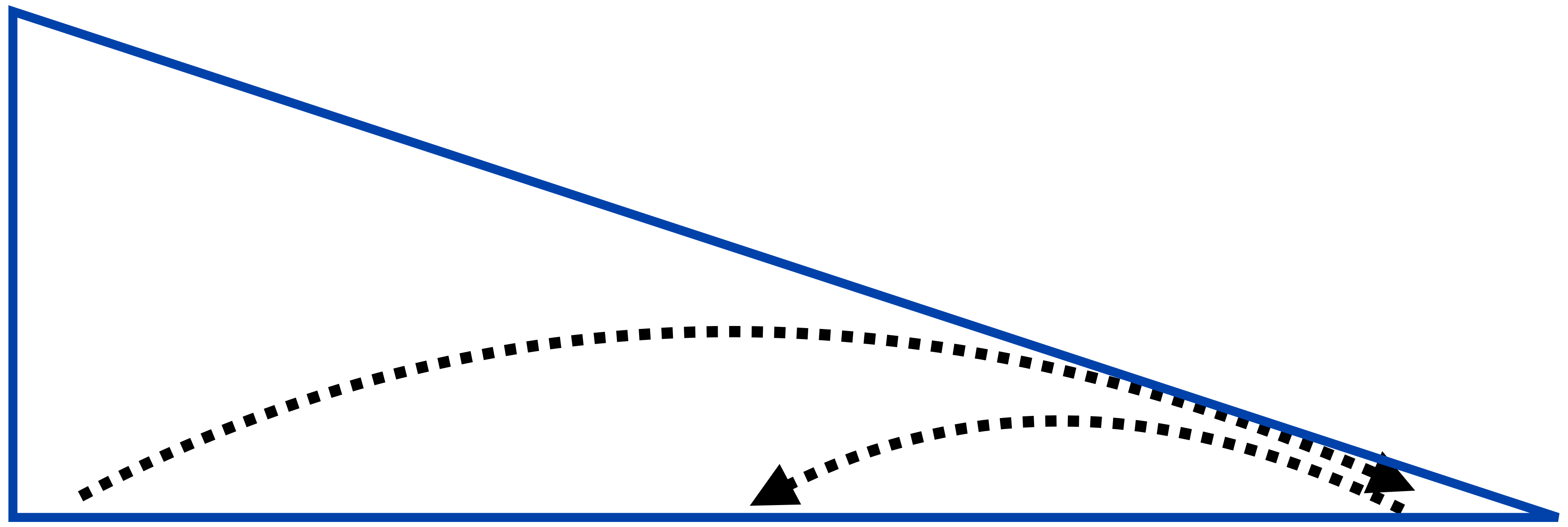
Eisner's Algorithm: $O(n^3)$



Eisner's Algorithm: O(n³)



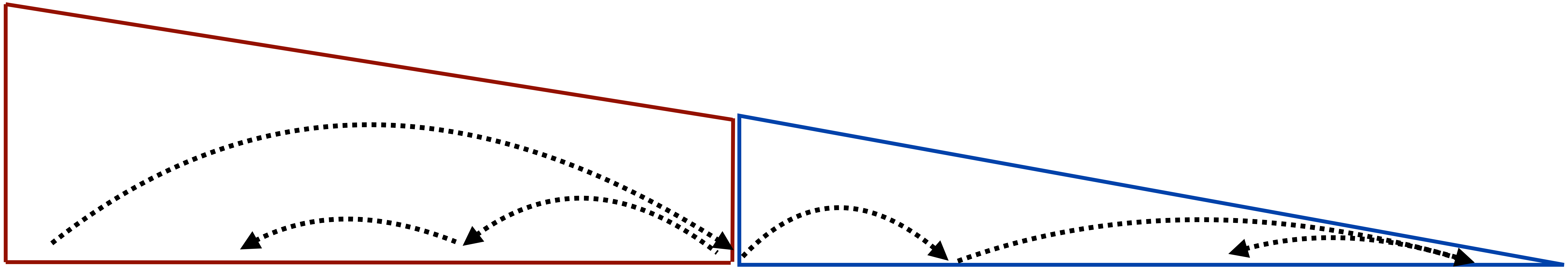
4) Promote to complete



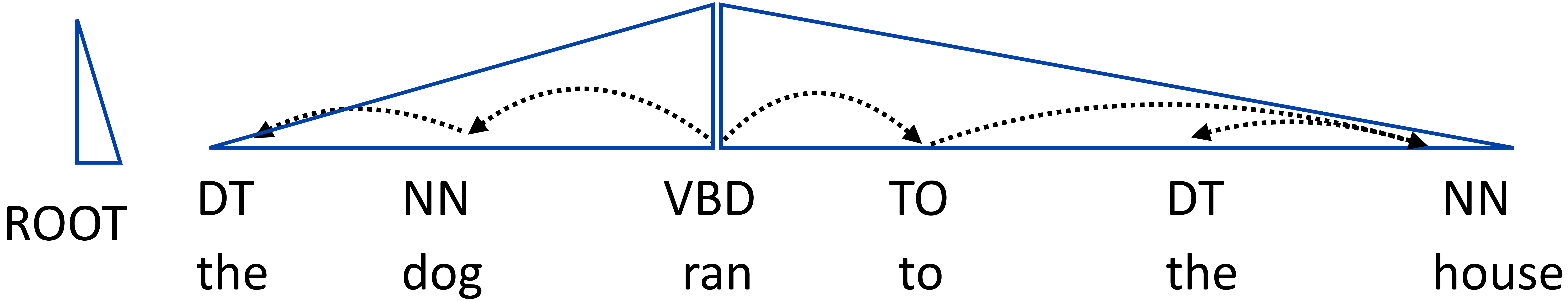
ROOT	DT	NN	VBD	TO	DT	NN
	the	dog	ran	to	the	house

Eisner's Algorithm: $O(n^3)$

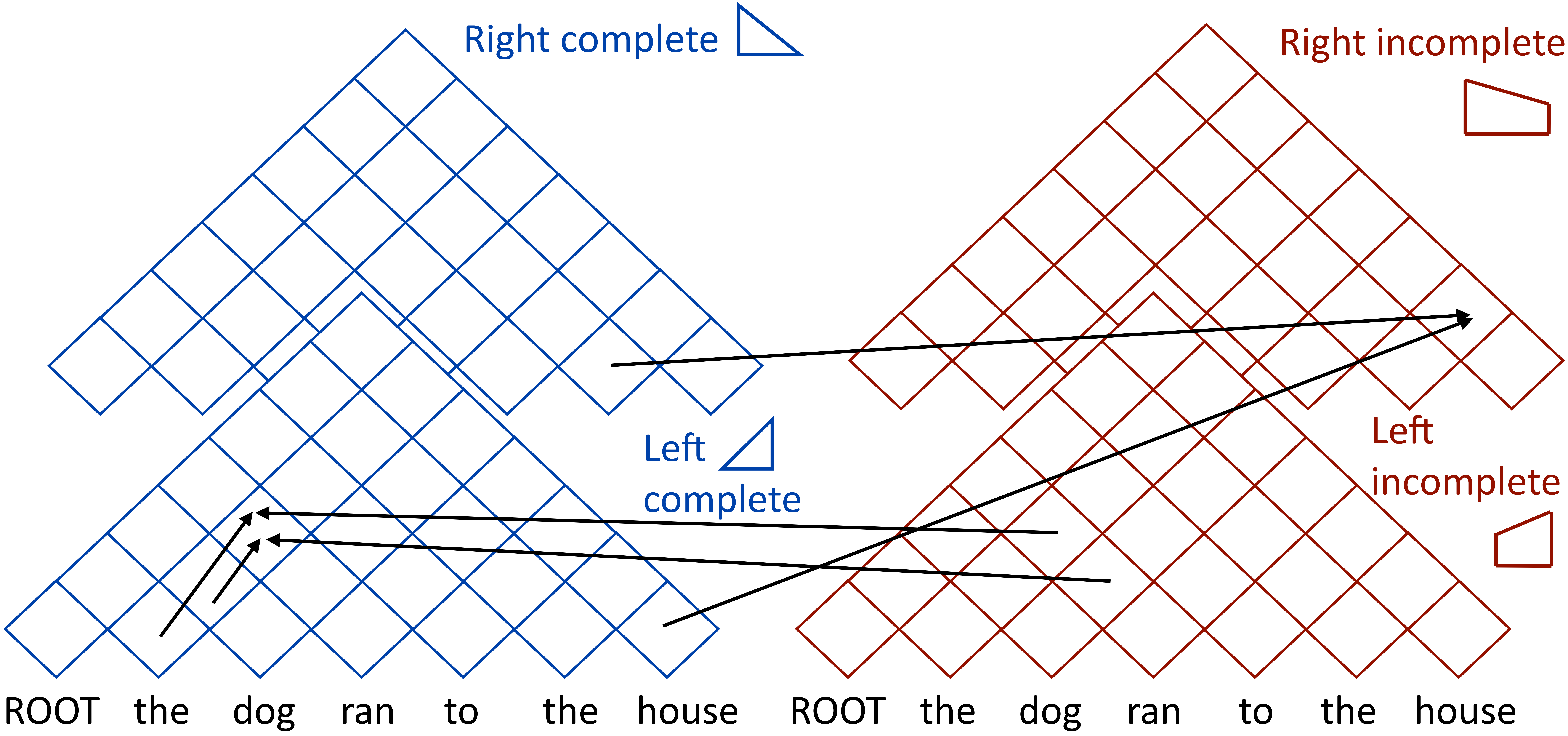
- Attaching to ROOT makes an incomplete item with left children, attaches with right children subsequently to finish the parse



- We've built left children and right children of *ran* as complete items

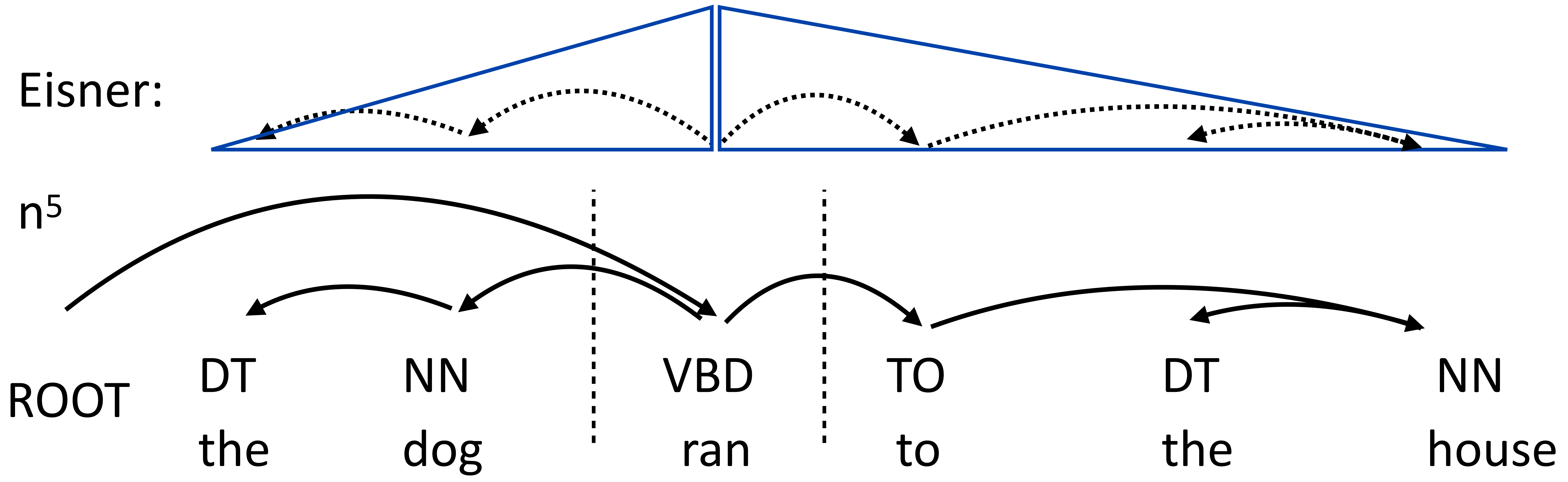


Eisner's Algorithm



Eisner's Algorithm

- Eisner's algorithm doesn't have split point ambiguities like CKY does
- Left and right children are built independently, heads are edges of spans
- Charts are n x n x 2 because we need to track arc direction / left vs right



Building Systems

- Can implement decoding and marginal computation using Eisner's algorithm to max/sum over projective trees
- Conceptually the same as inference/learning for sequential CRFs for NER, can also use margin-based methods

Features in Graph-Based Parsing

- Dynamic program exposes the parent and child indices

$$f(i, \text{parent}(i), \mathbf{x})$$

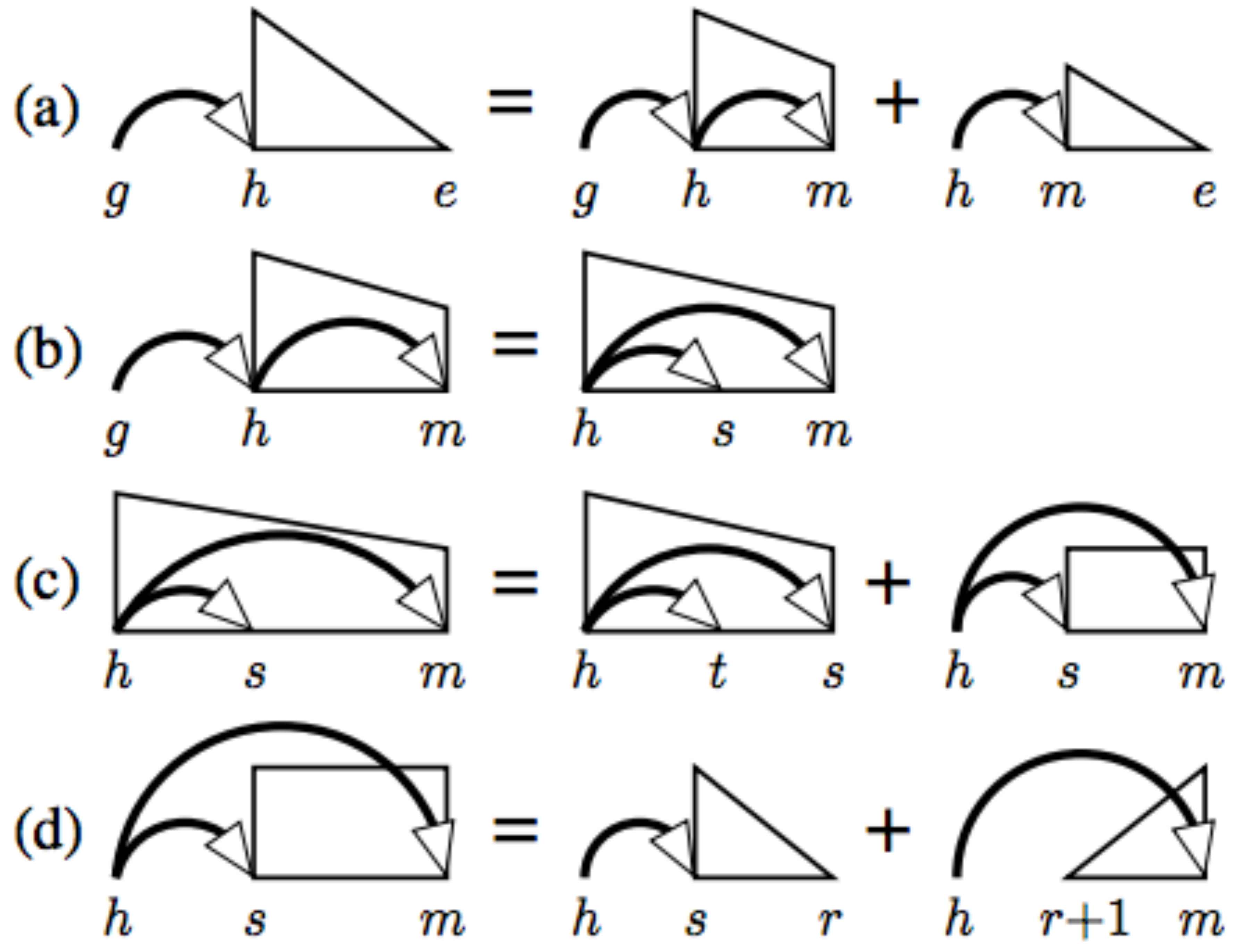
ROOT	DT	NN	VBD	TO	DT	NN
	the	dog	ran	to	the	house

- McDonald et al. (2005) — conjunctions of parent and child words + POS, POS of words in between, POS of surrounding words
  - HEAD=TO & MOD=NN
  - HEAD=TO & MOD=house
  - HEAD=TO & MOD-1=the
  - ARC_CROSSES=DT

Higher-Order Parsing

ROOT	DT	NN	VBD	TO	DT	NN
	the	dog	ran	to	the	house

$$f(i, \mathrm{parent}(i), \mathrm{parent}(\mathrm{parent}(i)), \mathbf{x})$$

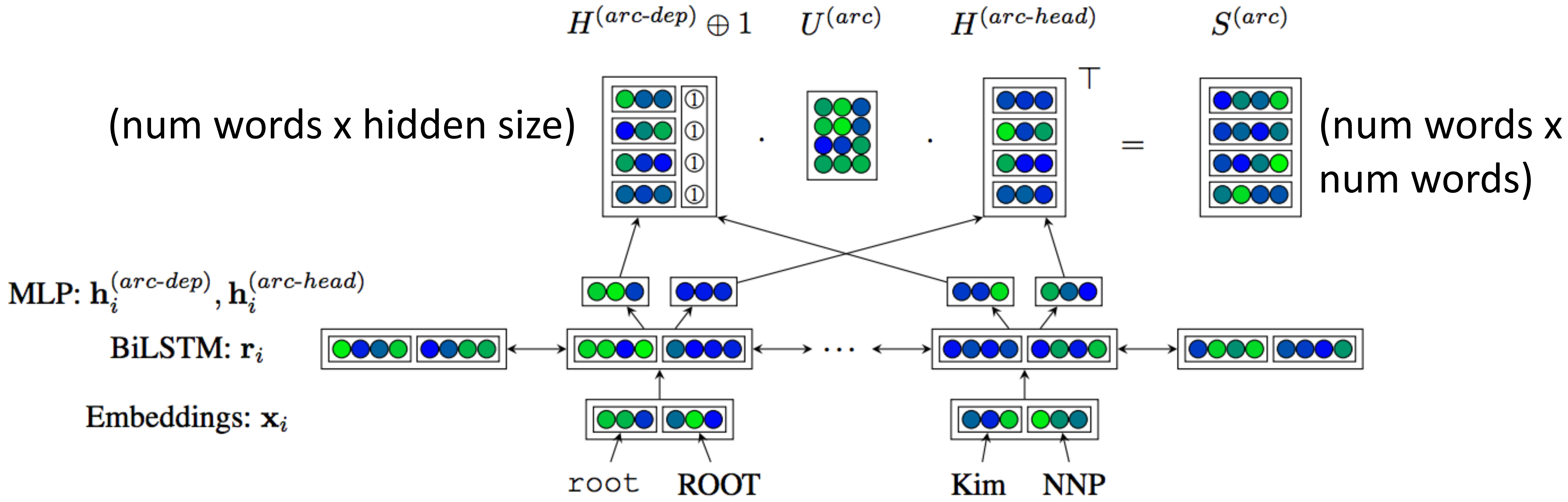


- Track additional state during parsing so we can look at "grandparents" (and siblings). O(n⁴) dynamic program or use approximate search

Koo and Collins (2009)

Biaffine Neural Parsing

- Neural CRFs for dependency parsing: let *c* = LSTM embedding of *i*, *p* = LSTM embedding of parent(*i*). *score*(*i*, parent(*i*), **x**) = $p^T U c$



LSTM looks at words and POS

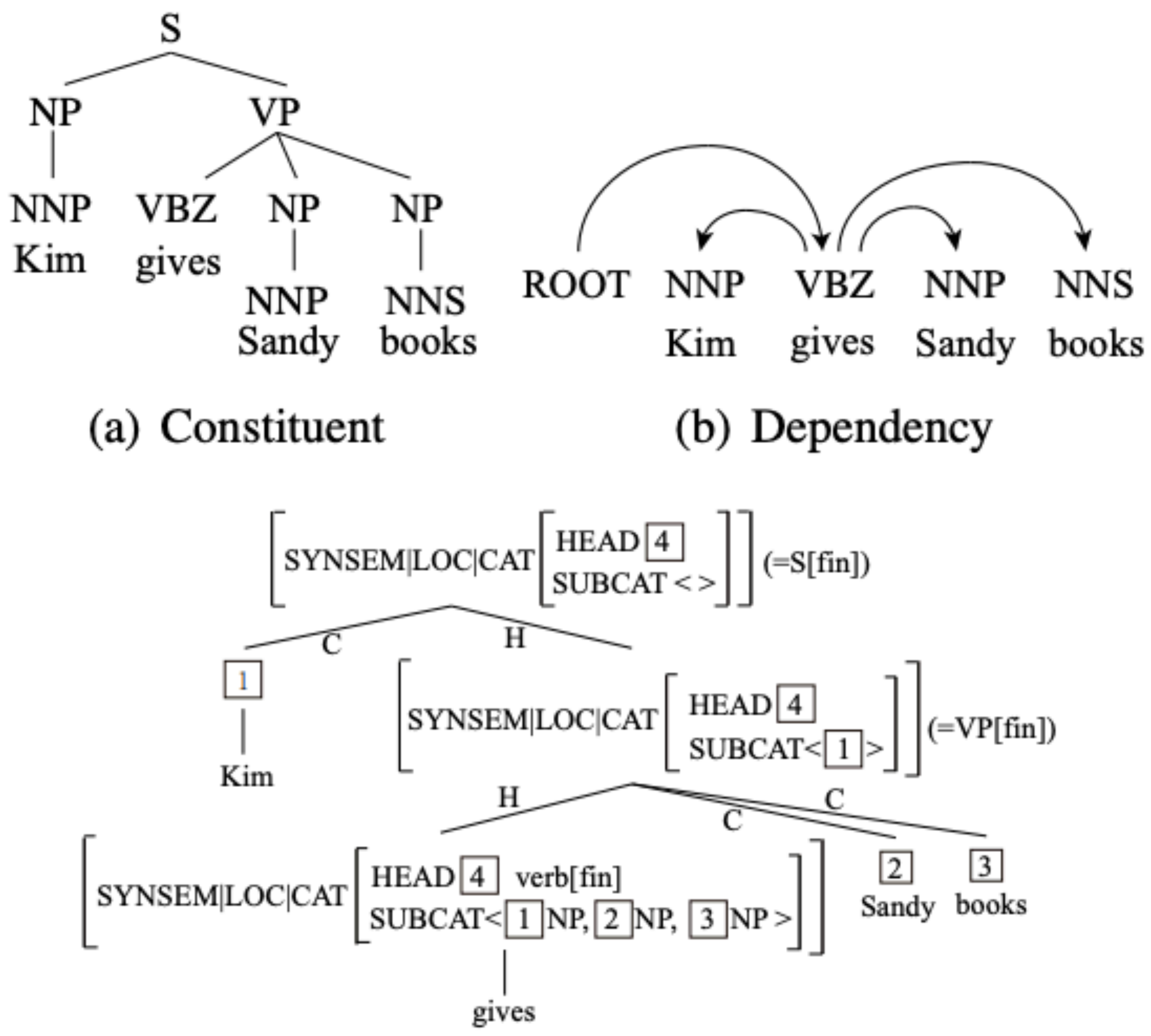
Dozat and Manning (2017)

Evaluating Dependency Parsing

- UAS: unlabeled attachment score. Accuracy of choosing each word's parent (*n* decisions per sentence)
- LAS: additionally consider label for each edge
- Log-linear CRF parser, decoding with Eisner algorithm: 91 UAS
- Higher-order features from Koo parser: 93 UAS
- Best English results with neural CRFs (Dozat and Manning): 95-96 UAS

HPSG

- Head-driven phrase structure grammar (HPSG): very complex grammar formalism which annotates large feature structures over tree
- Very little work on HPSG in NLP

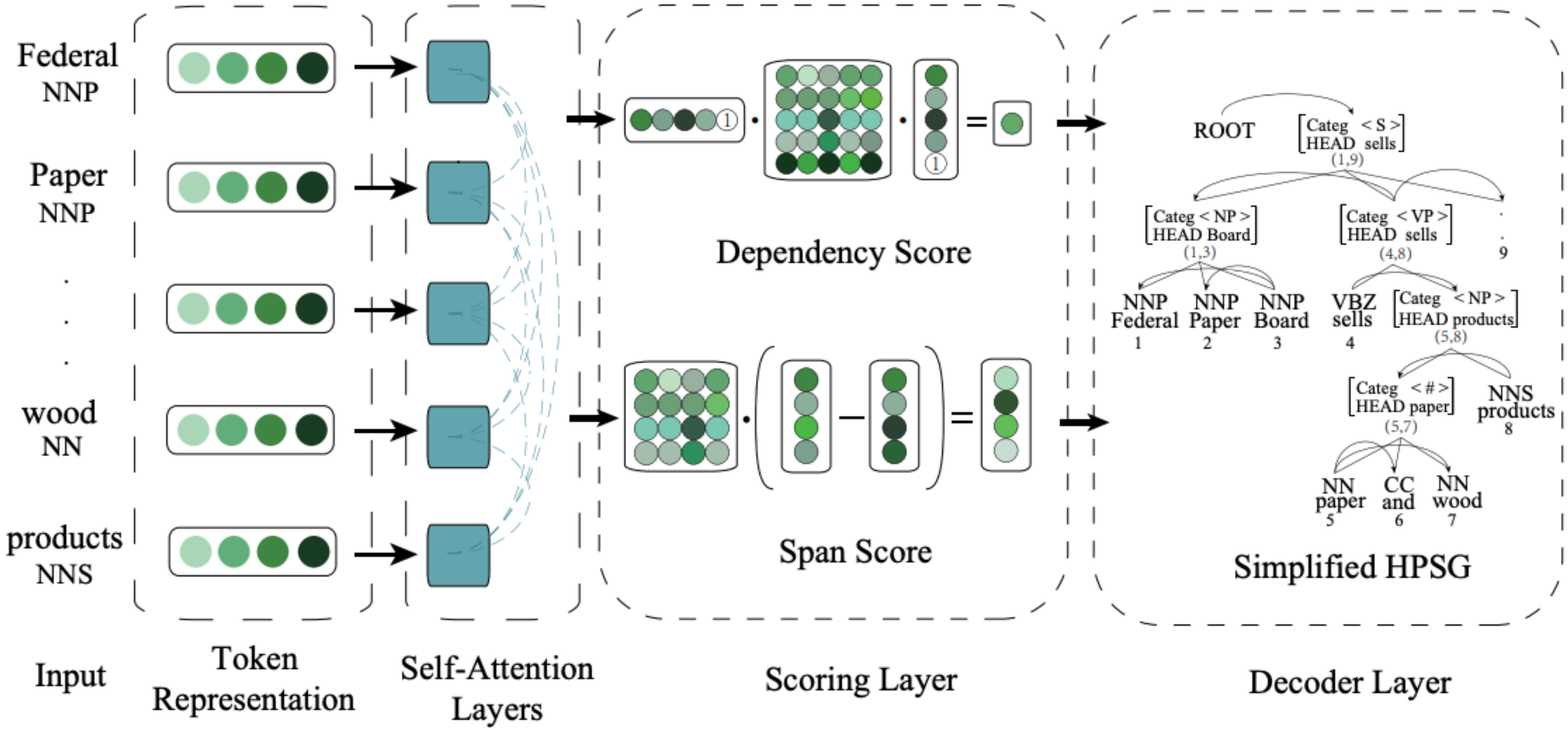


(a) Constituent (b) Dependency

(c) HPSG

Pollard and Sag (1994), Zhou and Zhao (2019)

Parsing with "HPSG"



- Joint model of constituency and dependency combining ideas from Dozat + Manning and Stern et al.

Zhou and Zhao (2019)

Parsing with "HPSG"

- Slightly stronger results than Dozat + Manning, significantly better results on Chinese

Model	English		Chinese	
	UAS	LAS	UAS	LAS
Chen and Manning (2014)	91.8	89.6	83.9	82.4
Andor et al. (2016)	94.61	92.79	–	–
Zhang et al. (2016)	93.42	91.29	87.65	86.17
Cheng et al. (2016)	94.10	91.49	88.1	85.7
Kuncoro et al. (2016)	94.26	92.06	88.87	87.30
Ma and Hovy (2017)	94.88	92.98	89.05	87.74
Dozat and Manning (2017)	95.74	94.08	89.30	88.23
Li et al. (2018a)	94.11	92.08	88.78	86.23
Ma et al. (2018)	95.87	94.19	90.59	**89.29**
Our (Division)	94.32	93.09	89.14	87.31
Our (Joint)	**96.09**	**94.68**	**91.21**	89.15
Our (Division*)	-	-	91.69	90.54
Our (Joint*)	-	-	93.24	91.95

Zhou and Zhao (2019)

Takeaways

- Dependency formalism provides an alternative to constituency, particularly useful in how portable it is across languages

- Dependency parsing also has efficient dynamic programs for inference

- CRFs + neural CRFs (again) work well

Proj 1 Results

Jiaming Chen: 82.46 F1

Po-Yi Chen: 82.02 F1

Ting-Yu Yen: 81.57 F1

Prakhar Singh: 81.54 F1

All others <81

- WordPair features, larger window for POS tag extraction ([-2, 2])
- Also larger window and data shuffling in between epochs
- Unregularized Adagrad worked best
- City gazetteer, generic date recognizer