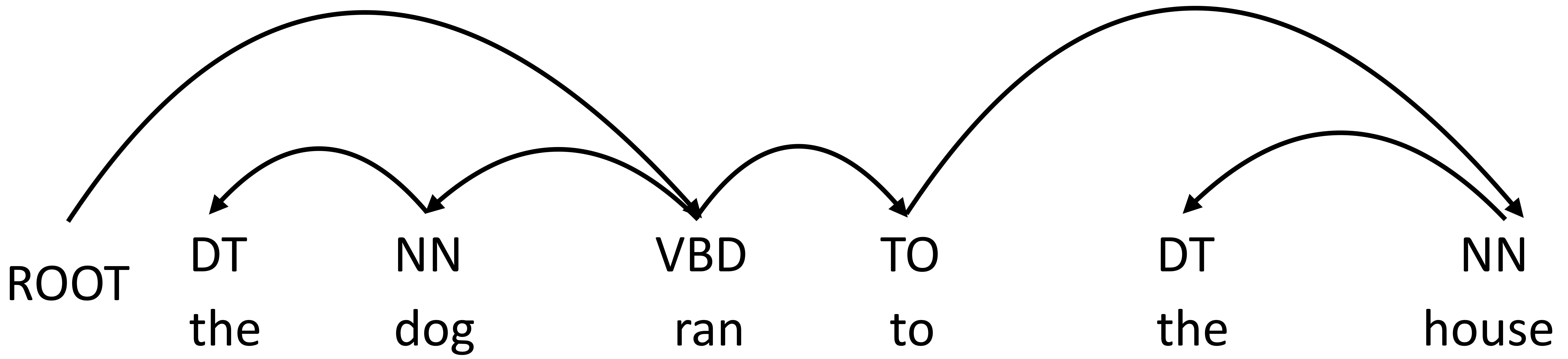
CS388: Natural Language Processing
Lecture 12: Dependency II

Greg Durrett

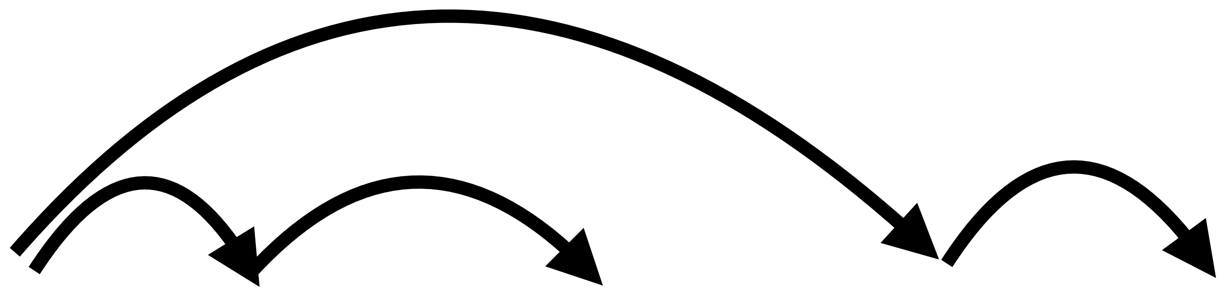
Recall: Dependencies

- Dependency syntax: syntactic structure is defined by dependencies
  - 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

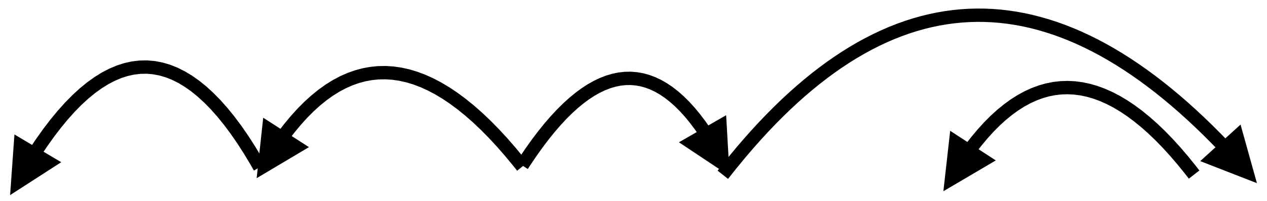


Recall: Projectivity

- Projective <-> no “crossing” arcs

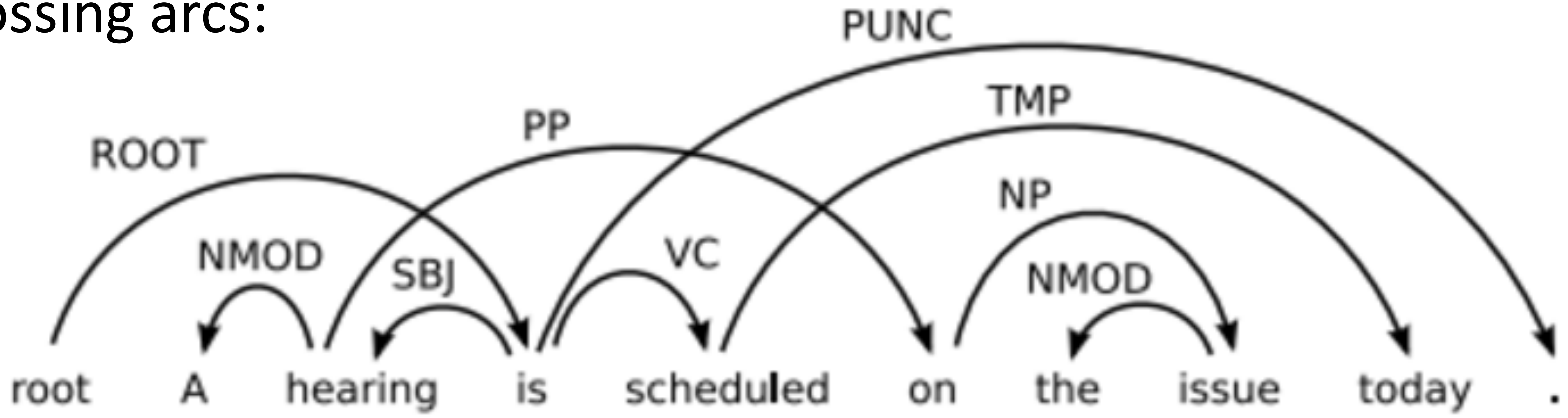


dogs in houses and cats



the dog ran to the house

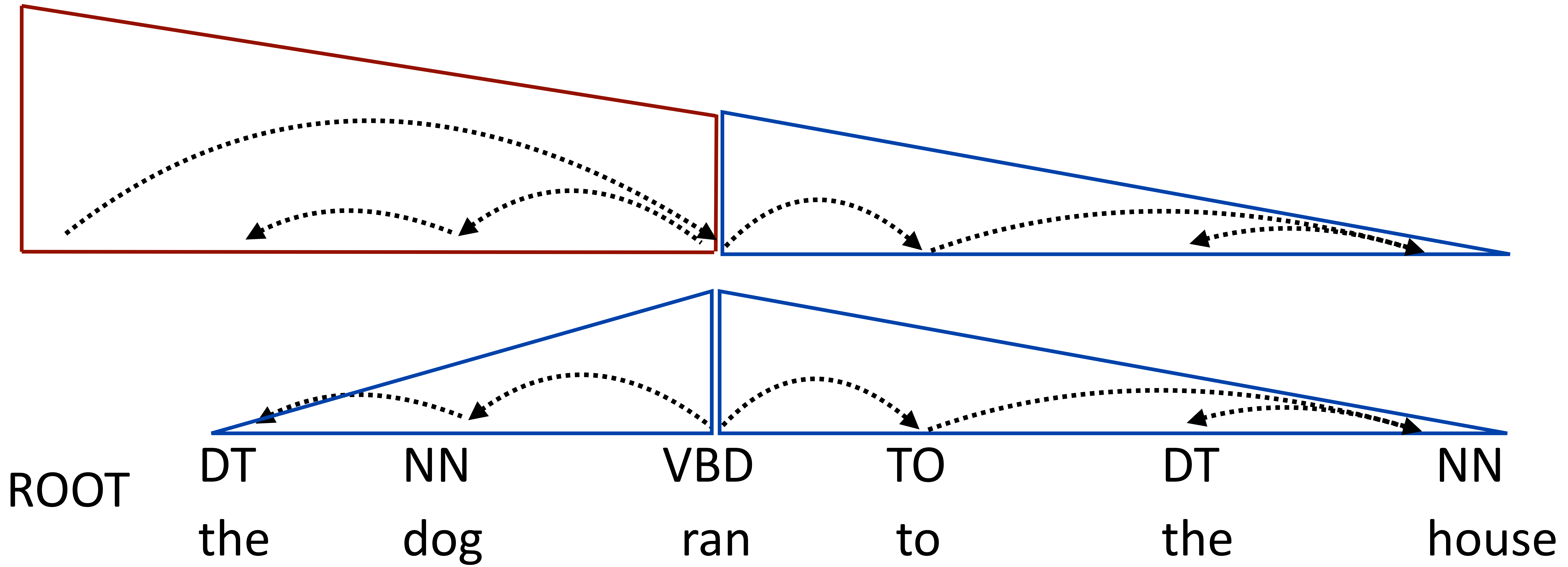
- Crossing arcs:



credit: Language Log

Recall: Eisner's Algorithm

- Left and right children are built independently, heads are edges of spans
- 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



This Lecture

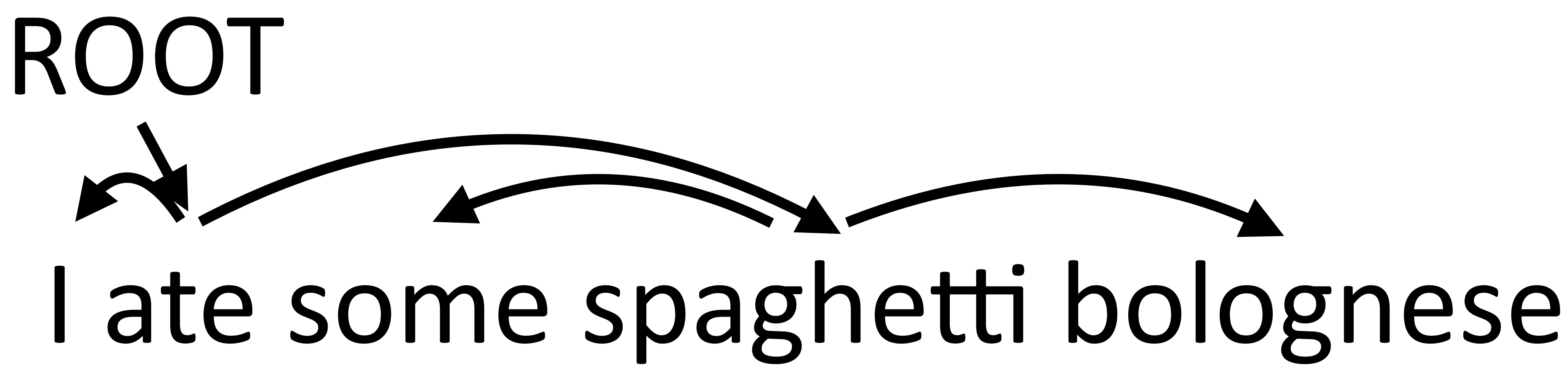
- Transition-based (shift-reduce) dependency parsing
  - Approximate, greedy inference — fast, but a little bit weird!

Shift-Reduce Parsing

Shift-Reduce Parsing

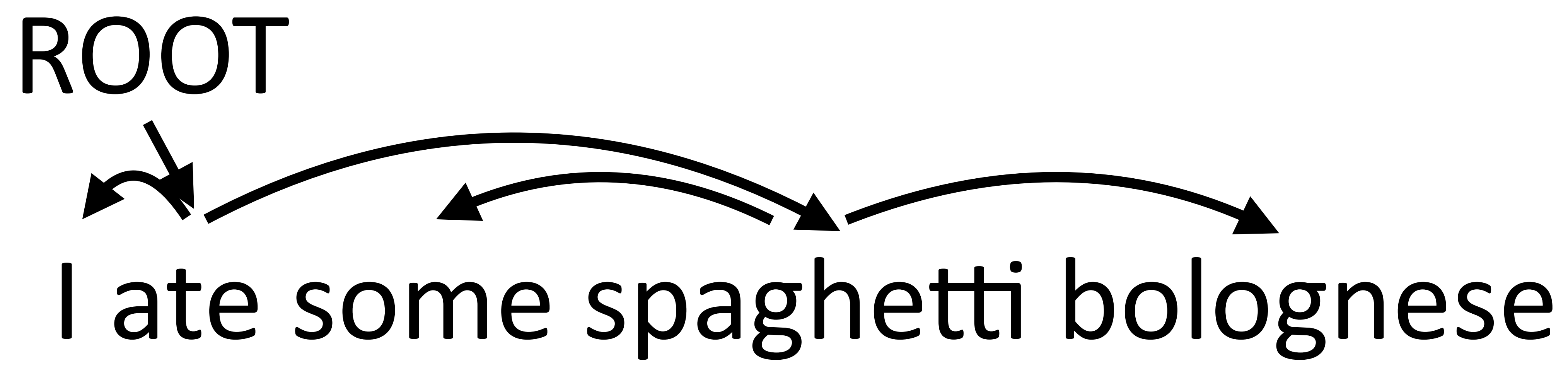
- Similar to deterministic parsers for compilers
  - Also called transition-based parsing
- A tree is built from a sequence of incremental decisions moving left to right through the sentence
- Stack containing partially-built tree, buffer containing rest of sentence
- Shifts consume the buffer, reduces build a tree on the stack

Shift-Reduce Parsing



- Initial state: Stack: [ROOT] Buffer: [I ate some spaghetti bolognese]
- Shift: top of buffer -> top of stack
  - Shift 1: Stack: [ROOT I] Buffer: [ate some spaghetti bolognese]
  - Shift 2: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]

Shift-Reduce Parsing



- State: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]

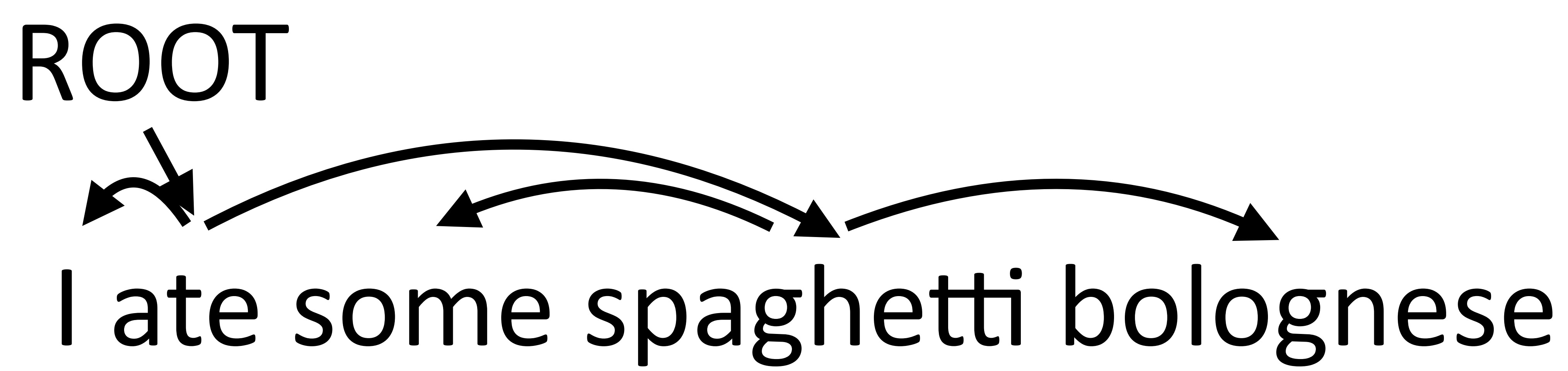
- Left-arc (reduce): Let σ denote the stack, $\sigma|w_{-1}$ = stack ending in w_{-1}
  - "Pop two elements, add an arc, put them back on the stack"

$$\boxed{\sigma|w_{-2}, w_{-1}} \rightarrow \boxed{\sigma|w_{-1}}, \quad w_{-2} \text{ is now a child of } w_{-1}$$

- State: Stack: [ROOT ate] Buffer: [some spaghetti bolognese]

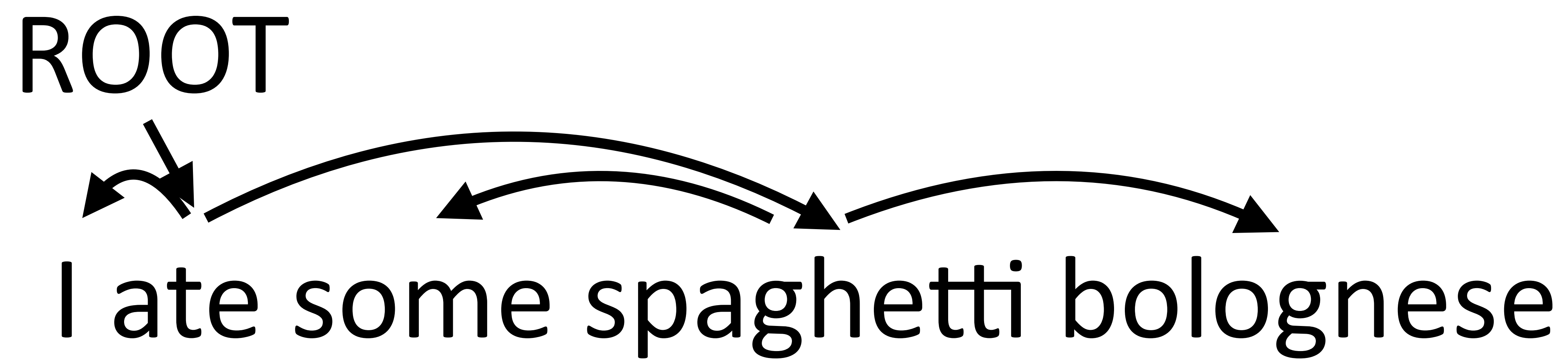
I

Arc-Standard Parsing



- Start: stack contains [ROOT], buffer contains [I ate some spaghetti bolognese]
- 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}
- End: stack contains [ROOT], buffer is empty []
- How many transitions do we need if we have n words in a sentence?

Arc-Standard Parsing



S	top of buffer -> top of stack
LA	pop two, left arc between them
RA	pop two, right arc between them

[ROOT] [I ate some spaghetti bolognese]

S

[ROOT I] [ate some spaghetti bolognese]

S

[ROOT I ate] [some spaghetti bolognese]

L

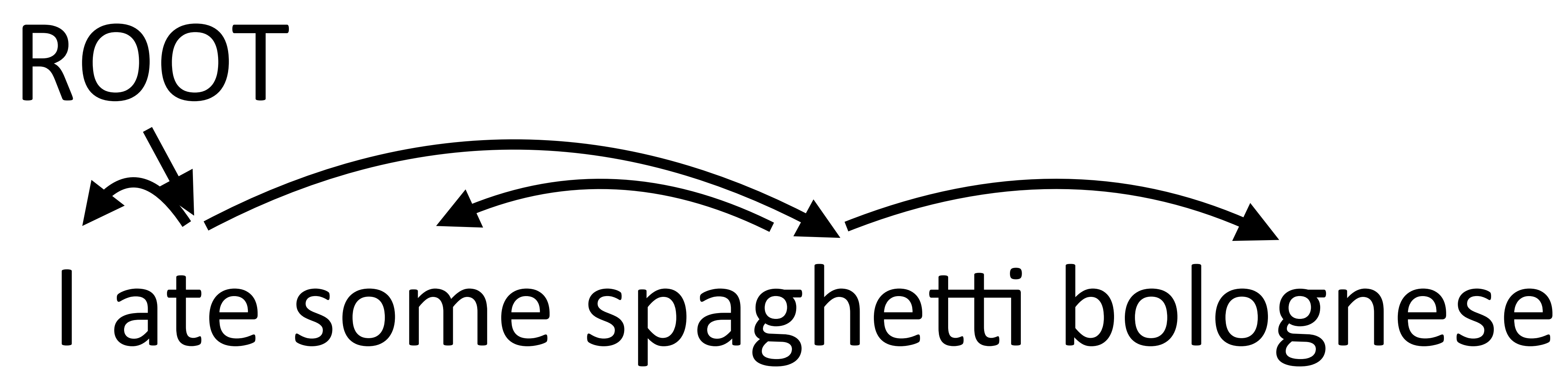
[ROOT ate] [some spaghetti bolognese]

↓

I

- Could do the left arc later! But no reason to wait
- Can’t attach ROOT <- ate yet even though this is a correct dependency!

Arc-Standard Parsing



S	top of buffer -> top of stack
LA	pop two, left arc between them
RA	pop two, right arc between them

[ROOT ate] [some spaghetti bolognese]

↓
I

S

S

[ROOT ate some spaghetti] [bolognese]

↓
I

L

[ROOT ate spaghetti] [bolognese]

↓ ↓
I some

S

Arc-Standard Parsing

ROOT

I ate some spaghetti bolognese

[ROOT ate spaghetti bolognese] []

I some

R

[ROOT ate spaghetti] []

I some bolognese

R

[ROOT ate] []

I spaghetti

some bolognese

S	top of buffer -> top of stack
LA	pop two, left arc between them
RA	pop two, right arc between them

- Stack consists of all words that are still waiting for right children, end with a bunch of right-arc ops

Final state:

[ROOT] []

ate

I spaghetti

some bolognese

Other Systems

- Arc-eager (Nivre, 2004): lets you add right arcs sooner and keeps items on stack, separate reduce action that clears out the stack
- Arc-swift (Qi and Manning, 2017): explicitly choose a parent from what's on the stack
- Many ways to decompose these, which one works best depends on the language and features (nonprojective variants too!)

Building Shift-Reduce Parsers

[ROOT] [I ate some spaghetti bolognese]

- How do we make the right decision in this case?
- Only one legal move (shift)

[ROOT ate some spaghetti] [bolognese]

$$\text{ate} \downarrow \text{I}$$

- How do we make the right decision in this case? (all three actions legal)
- Multi-way classification problem: shift, left-arc, or right-arc?

$$\text{argmax}_{a \in \{\text{S,LA,RA}\}} w^\top f(\text{stack}, \text{buffer}, a)$$

Features for Shift-Reduce Parsing

[ROOT ate some spaghetti]   [bolognese]

↓

I

- Features to know this should left-arc?
- One of the harder feature design tasks!
- In this case: the stack tag sequence VBD - DT - NN is pretty informative — looks like a verb taking a direct object which has a determiner in it
- Things to look at: top words/POS of buffer, top words/POS of stack, leftmost and rightmost children of top items on the stack

Training a Greedy Model

[ROOT ate some spaghetti] [bologenese]

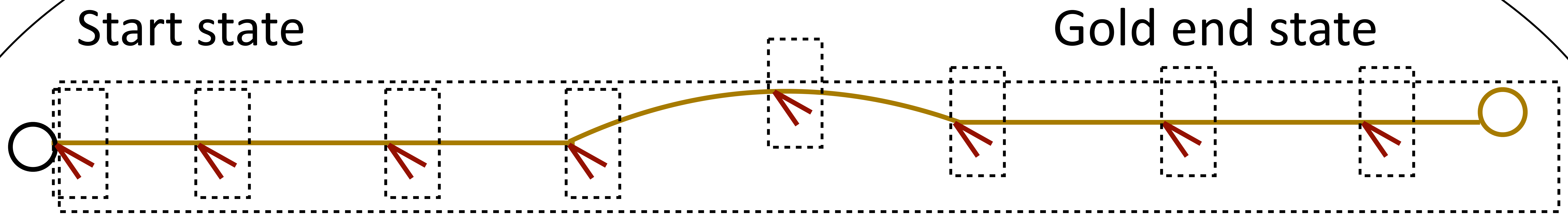
$\downarrow$

I

$$\mathrm{argmax}_{y \in \{\mathrm{S},\mathrm{LA},\mathrm{RA}\}} w^\top f(y, \mathrm{stack}, \mathrm{buffer})$$

- Can turn a tree into a decision sequence **a** by building an *oracle*
- Train a classifier to predict the right decision using these as training data
- Training data assumes you made correct decisions up to this point and teaches you to make the correct decision, but what if you screwed up…

Greedy training



- Greedy: 2n local training examples
- Non-gold states unobserved during training: consider making bad decisions but don't *condition* on bad decisions

Speed Tradeoffs

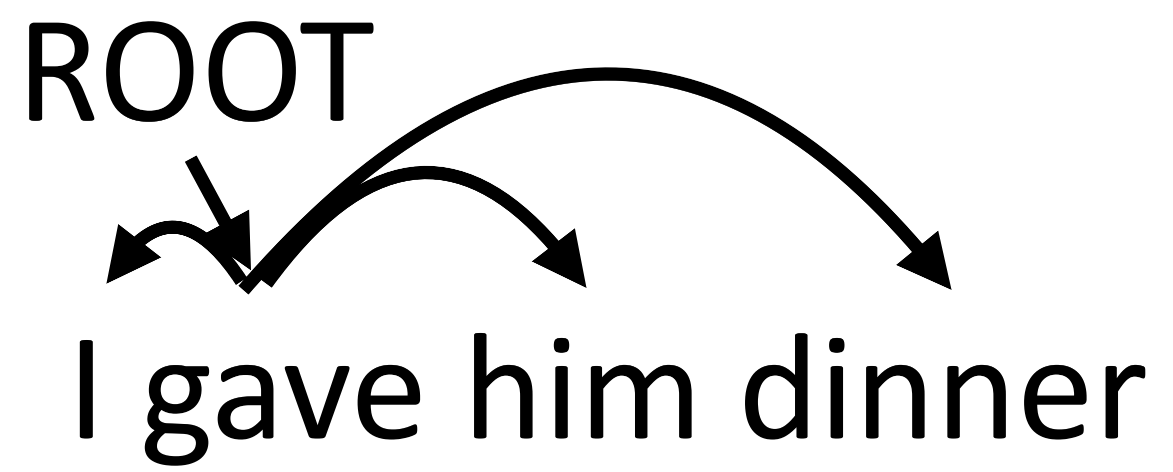
	Parser	Dev UAS	Dev LAS	Test UAS	Test LAS	Speed (sent/s)
Unoptimized S-R	standard	89.9	88.7	89.7	88.3	51
	eager	90.3	89.2	89.9	88.6	63
Optimized S-R	Malt:sp	90.0	88.8	89.9	88.5	560
	Malt:eager	90.1	88.9	90.1	88.7	535
Graph-based	MSTParser	92.1	90.8	**92.0**	90.5	12
Neural S-R	Our parser	**92.2**	**91.0**	**92.0**	**90.7**	**1013**

- Many early-2000s constituency parsers were ~5 sentences/sec
- Using S-R used to mean taking a performance hit compared to graph-based, that's no longer true

Chen and Manning (2014)

Global Decoding

Global Decoding



[ROOT gave him] [dinner]

I

- Is it a problem that we make decisions greedily?
- Correct: Right-arc, Shift, Right-arc, Right-arc

[ROOT gave] [dinner]

I him

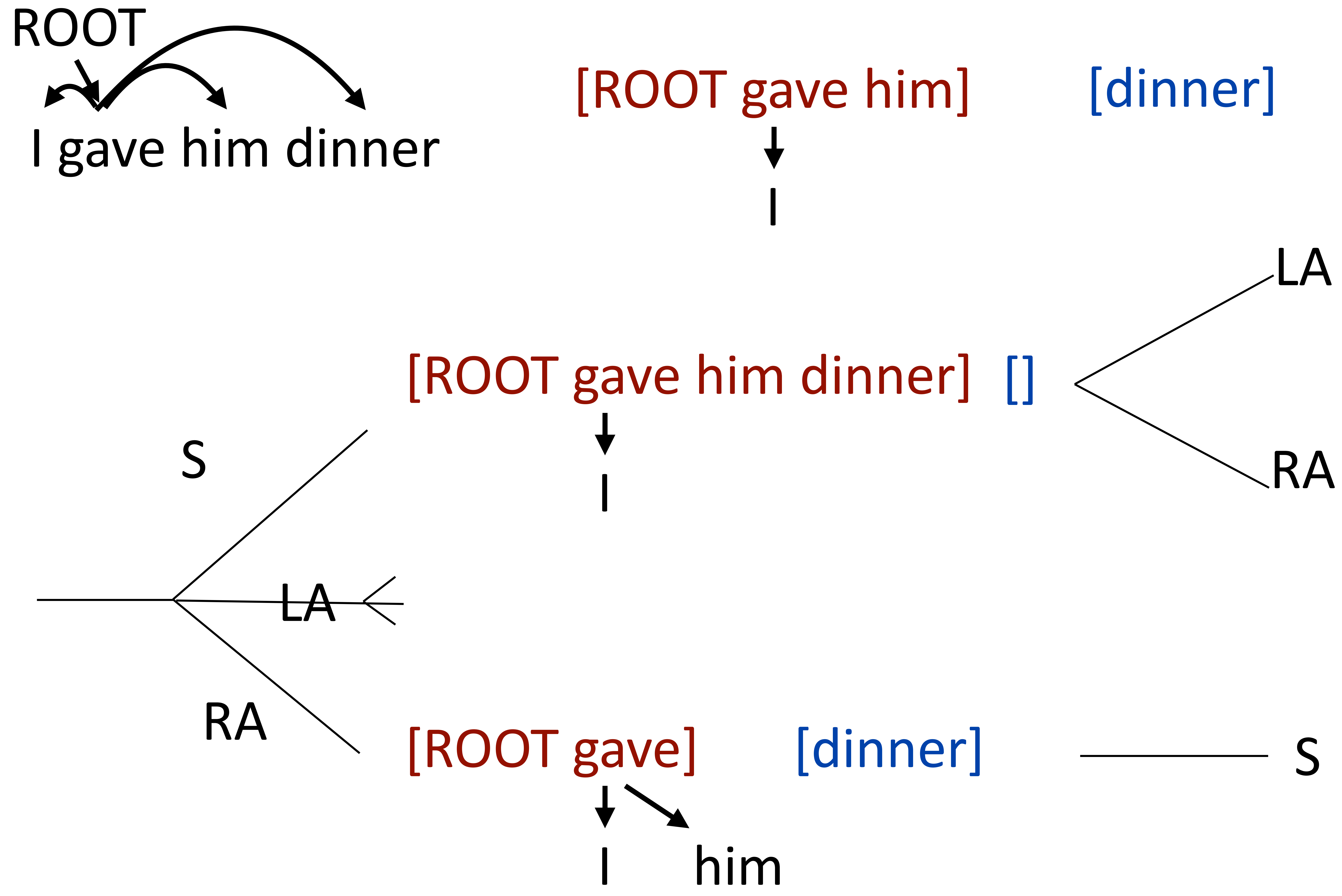
[ROOT gave dinner] []

I him

[ROOT gave] []

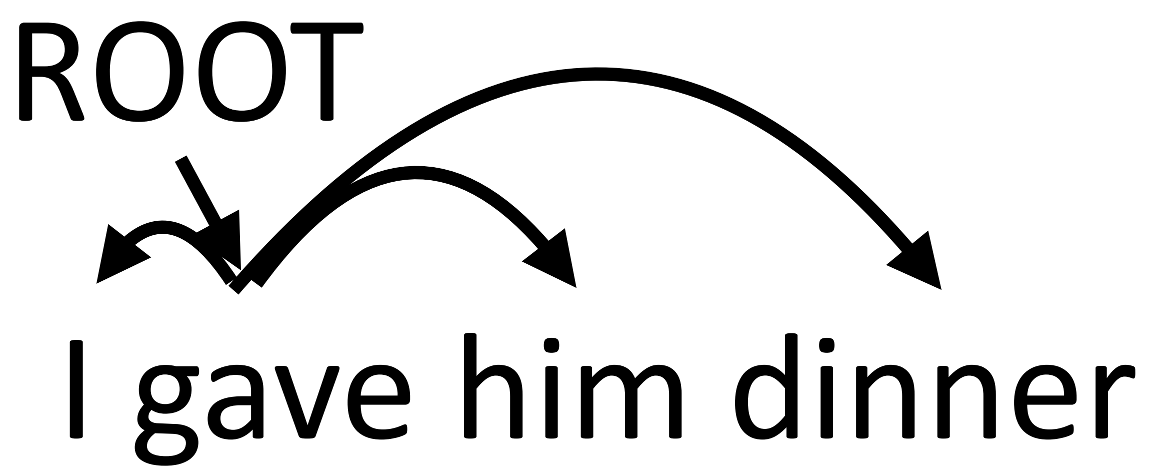
I him dinner

Global Decoding: A Cartoon



- Both wrong! Also both probably low scoring!
- Correct, high scoring option

Global Decoding: A Cartoon



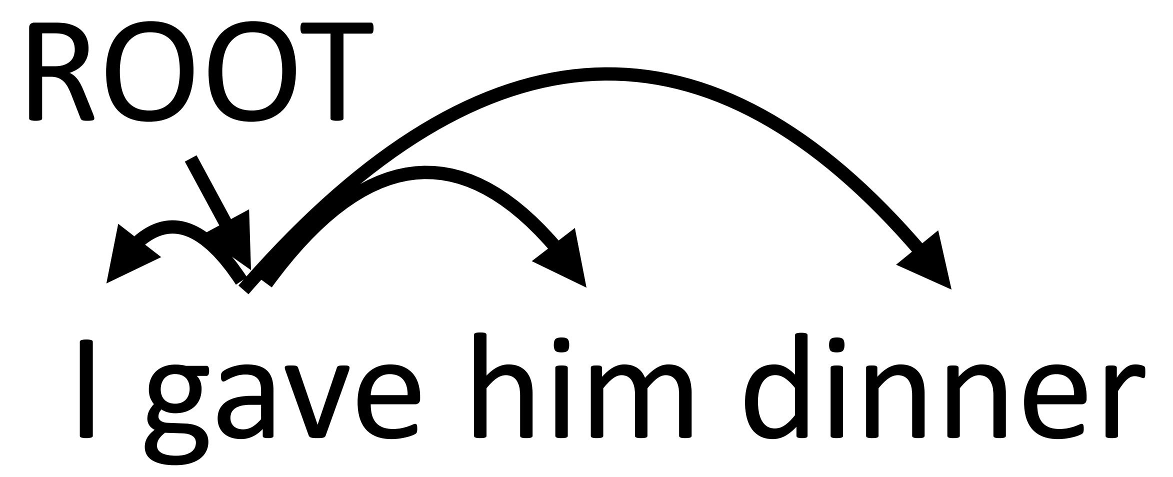
[ROOT gave him]   [dinner]

↓

I

- Lookahead can help us avoid getting stuck in bad spots
- Global model: maximize sum of scores over all decisions
- Similar to how Viterbi works: we maintain uncertainty over the current state so that if another one looks more optimal going forward, we can use that one

Global Shift-Reduce Parsing



[ROOT gave him] [dinner]

I

- Greedy: repeatedly execute

$$a_{\text{best}} \leftarrow \text{argmax}_a w^\top f(s,a)$$
$$s \leftarrow a_{\text{best}}(s)$$

- Can we do search exactly?
  - How many states *s* are there?
- No! Use beam search

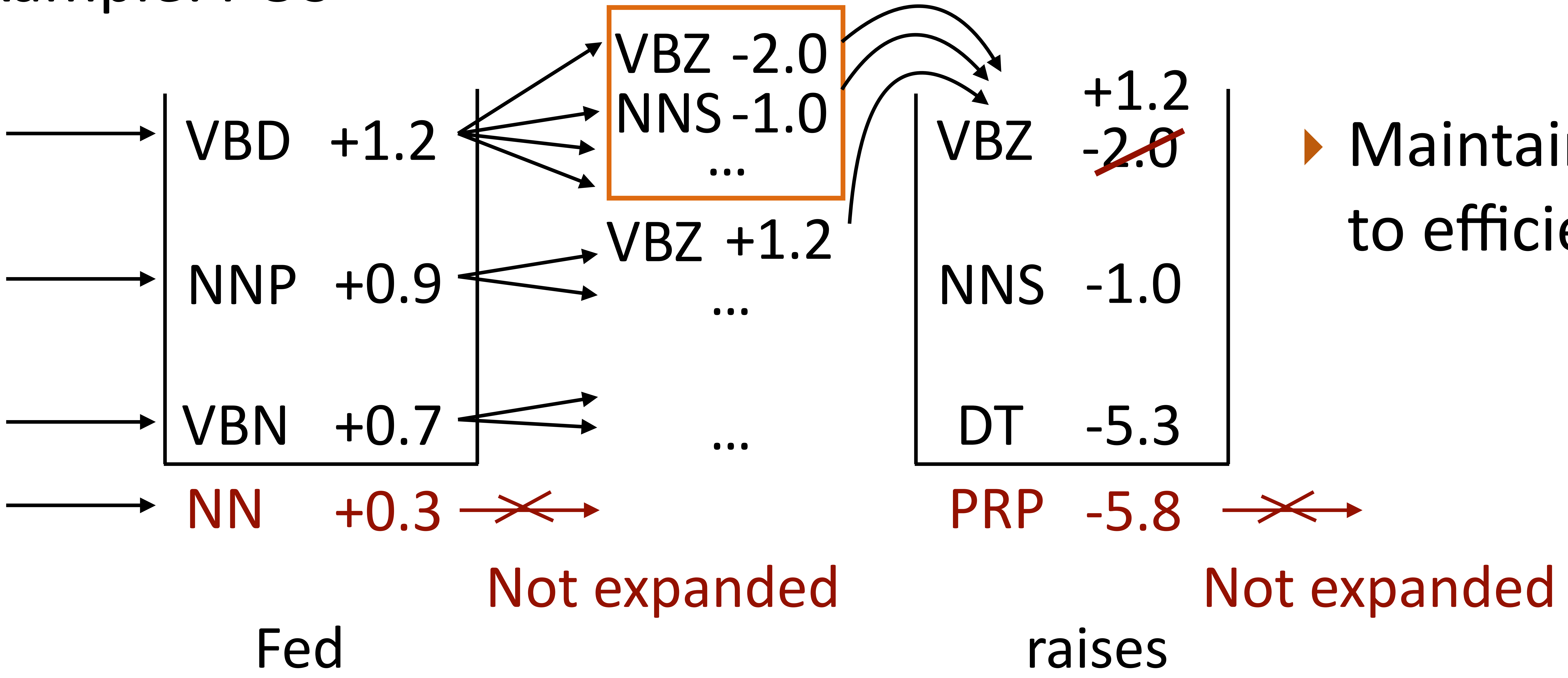
- Global:

$$\text{argmax}_{\mathbf{s},\mathbf{a}} w^\top f(\mathbf{s},\mathbf{a}) = \sum_{i=1}^{2n} w^\top f(s_i, a_i)$$
$$s_{i+1} = a_i(s_i)$$

Beam Search

- Maintain a beam of *k* plausible states at the current timestep, expand each and only keep top *k* best new ones

- Example: POS



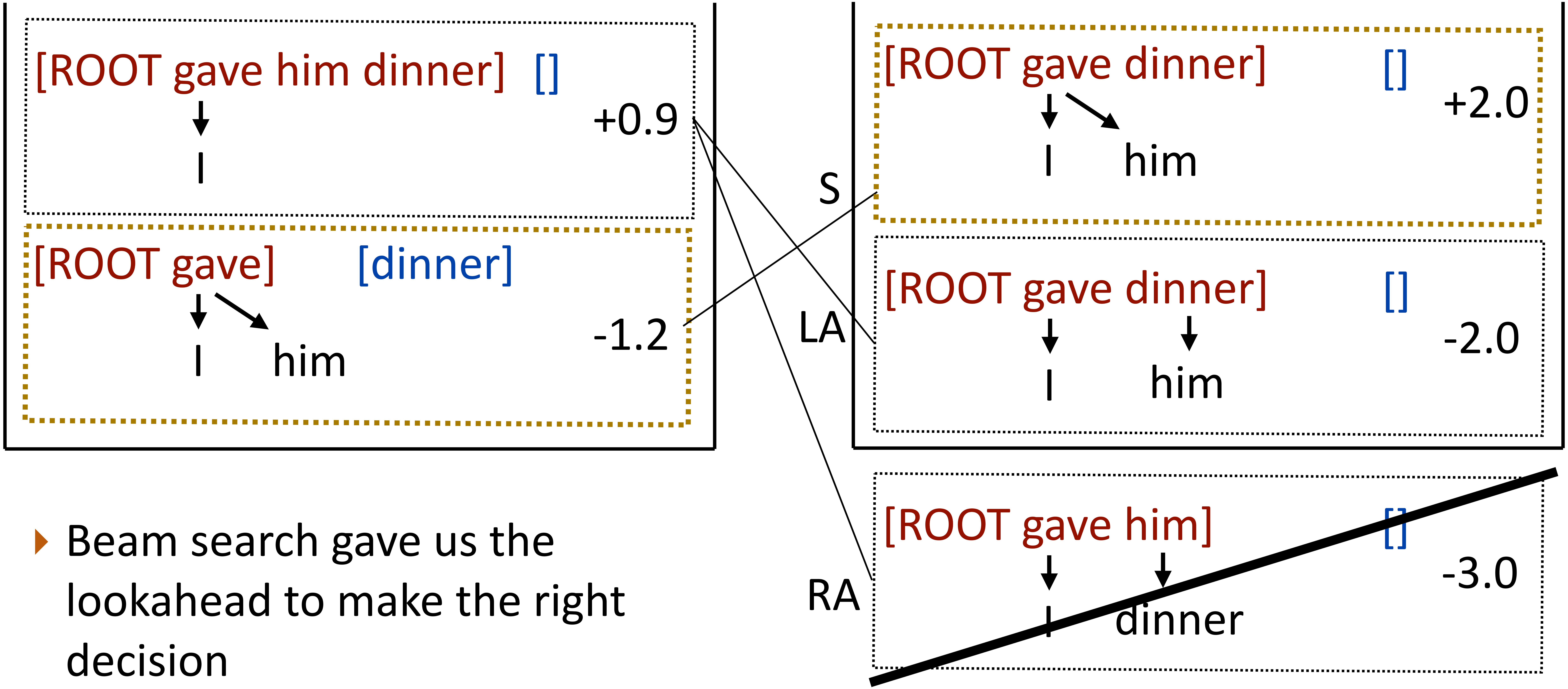
- Maintain priority queue to efficiently add things

- Beam size of k, n words, s states, time complexity O(nks log(ks))

How good is beam search?

- *k*=1: greedy search
- Choosing beam size:
  - 2 is usually better than 1
  - Usually don’t use larger than 50
  - Depends on problem structure

Global Shift-Reduce Parsing



- Beam search gave us the lookahead to make the right decision

Global Training

- If using global inference, should train the parser in a global fashion as well: use structured perceptron / structured SVM
- Model treats an entire derivation as something to featurize
- No algorithm like Viterbi for doing efficient parsing, so use beam search

State-of-the-art Parsers

State-of-the-art Parsers

- 2005: Eisner algorithm graph-based parser was SOTA (~91 UAS)
- 2010: Koo's 3rd-order parser was SOTA for graph-based (~93 UAS)
- 2012: Maltparser was SOTA was for transition-based (~90 UAS)
- 2014: Chen and Manning got 92 UAS with transition-based neural model
- 2016: Improvements to Chen and Manning

State-of-the-art Parsers

Softmax layer:

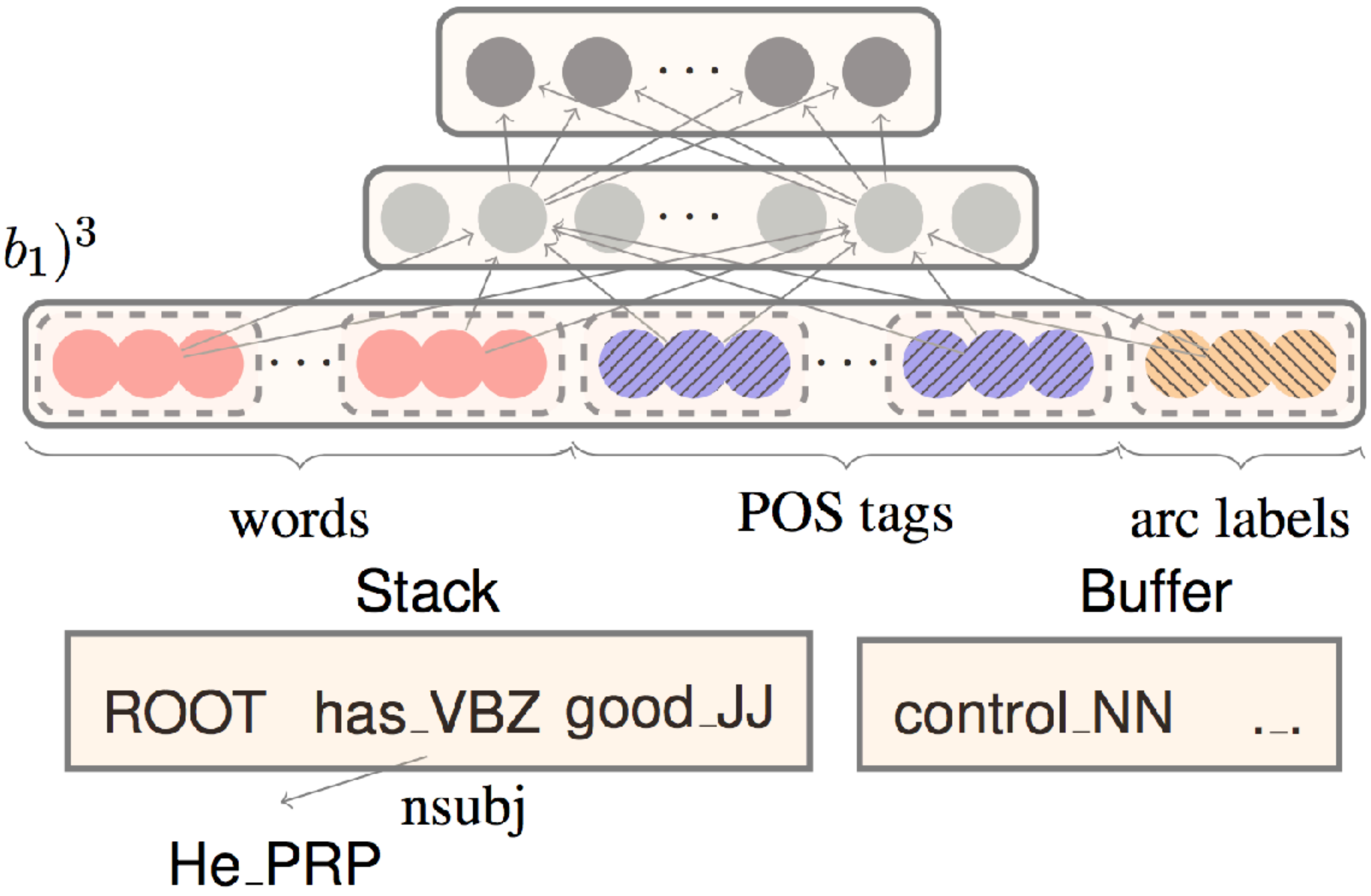
$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$

Configuration



Chen and Manning (2014)

Parsey McParseFace (a.k.a. SyntaxNet)

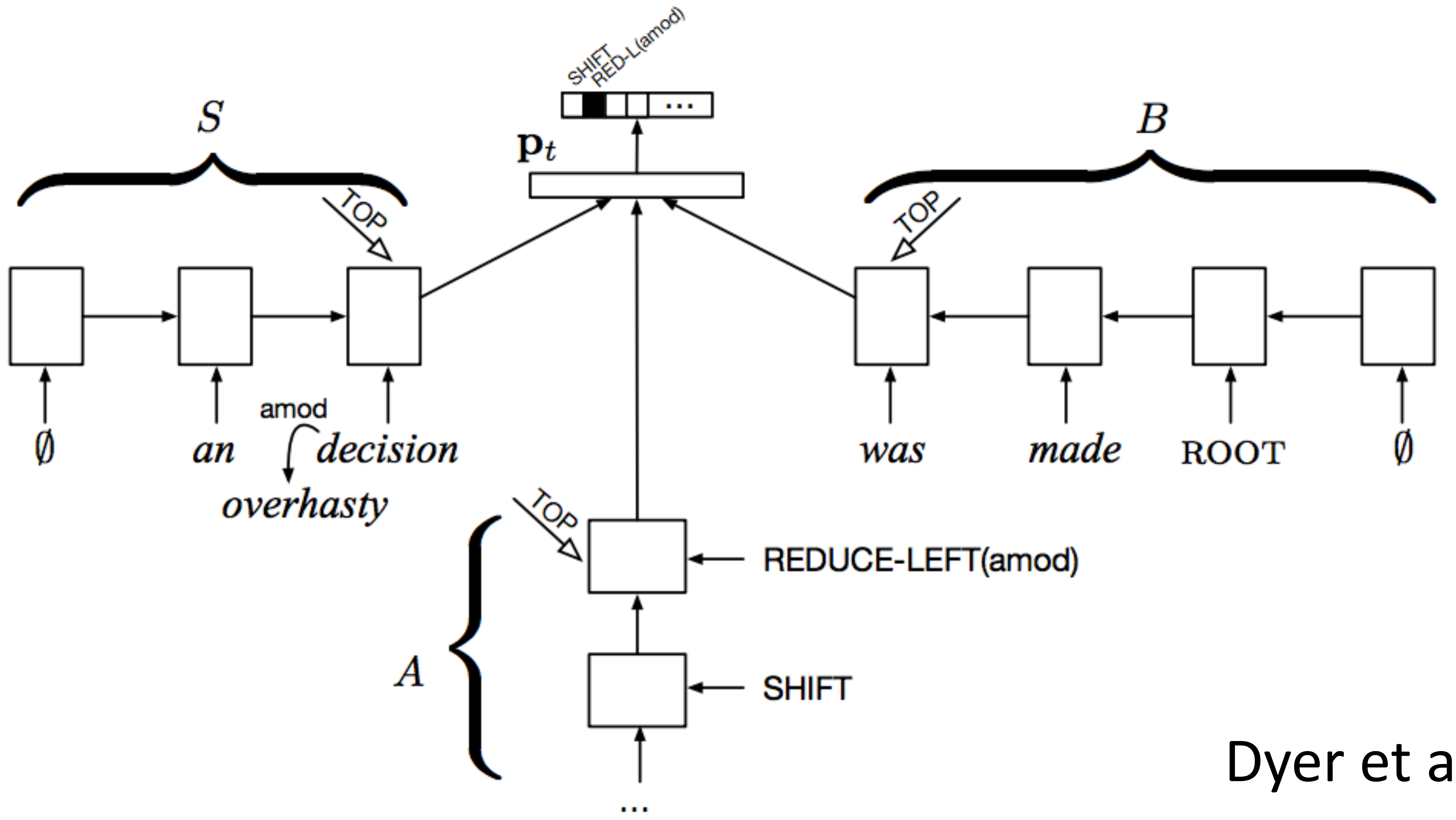
- Close to state-of-the-art, released by Google publicly

- 94.61 UAS on the Penn Treebank using a global transition-based system with early updating (compared to 95.8 for Dozat, 93.7 for Koo in 2009)
  - Additional data harvested via “tri-training”, form of self-training
- Feedforward neural nets looking at words and POS associated with
  - Words at the top of the stack
  - Those words’ children
  - Words in the buffer
- Feature set pioneered by Chen and Manning (2014), Google fine-tuned it

Andor et al. (2016)

Stack LSTMs

- Use LSTMs over stack, buffer, past action sequence. Trained greedily
- Slightly less good than Parsey



Dyer et al. (2015)

Recap

- Shift-reduce parsing can work nearly as well as graph-based
- Arc-standard system for transition-based parsing
- Purely greedy or more “global” approaches
- Next time: semantic parsing