

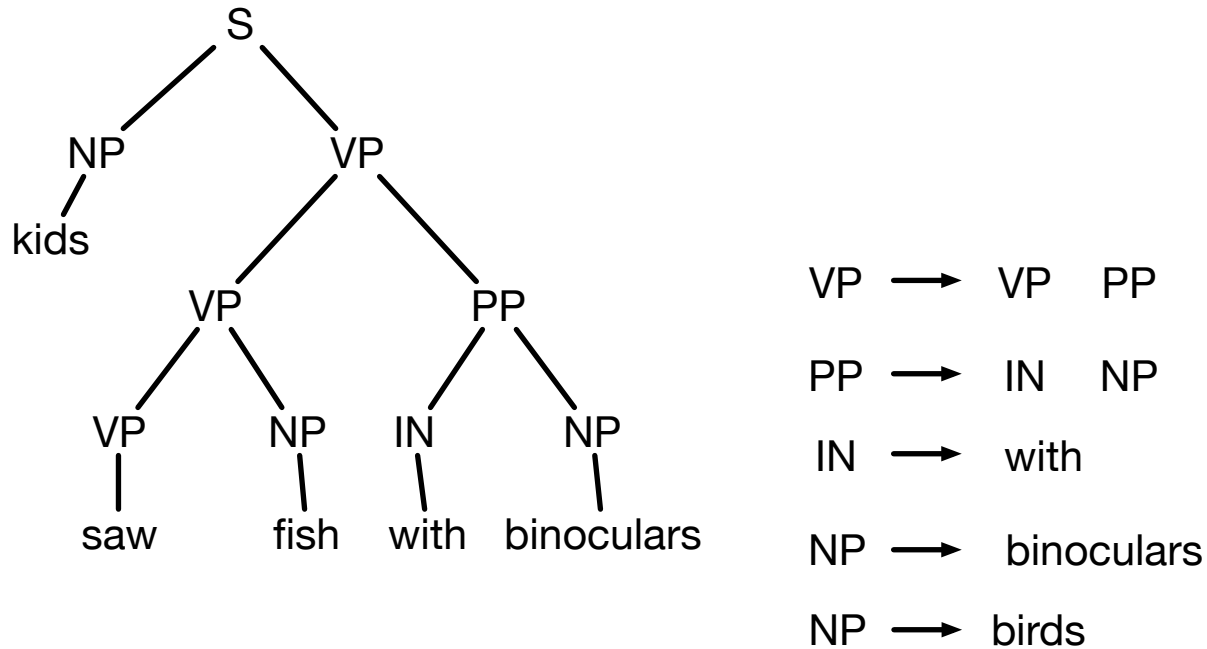
# Lecture 9: Dependency Trees and Shift-Reduce Dependency Parsing

USC VSoE CSCI 544: Applied Natural Language Processing

Jonathan May -- 梅約納

September 20, 2017

# Constituent View of Syntax



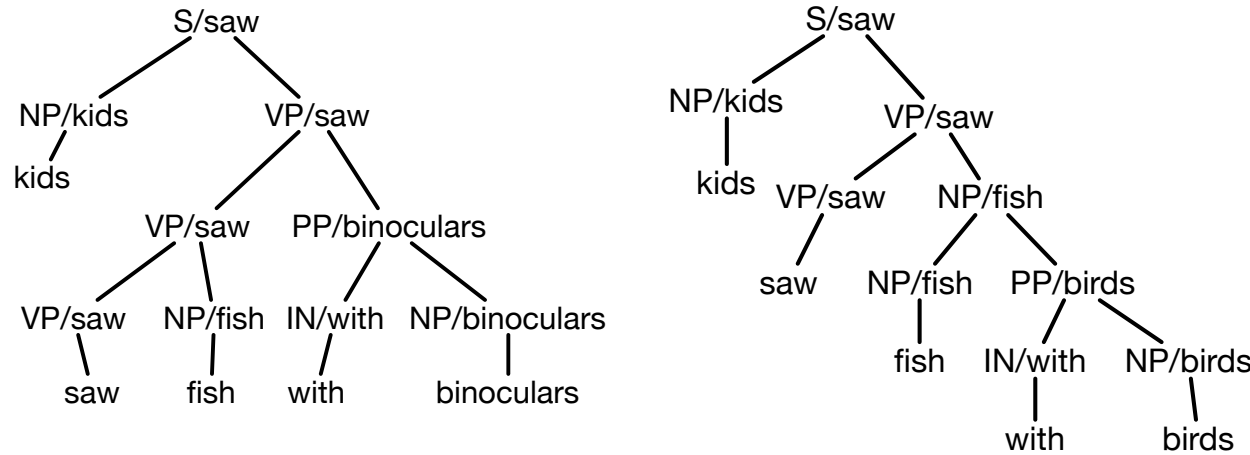
Similarly:

She stood by the door covered in tears vs.  
She stood by the door covered in ivy

stray cats and dogs vs.  
Siamese cats and dogs

- Words group into higher-order linguistic units
- Disadvantages:
  - Doesn't model languages with discontinuous information, free word order as well
  - May be too much information for downstream applications
  - In pure CFG form, lexically grounded selectional preference is lost

# Fix With Lexicalization



VP/saw → VP/saw NP/fish

VP/saw → VP/saw PP/binoculars

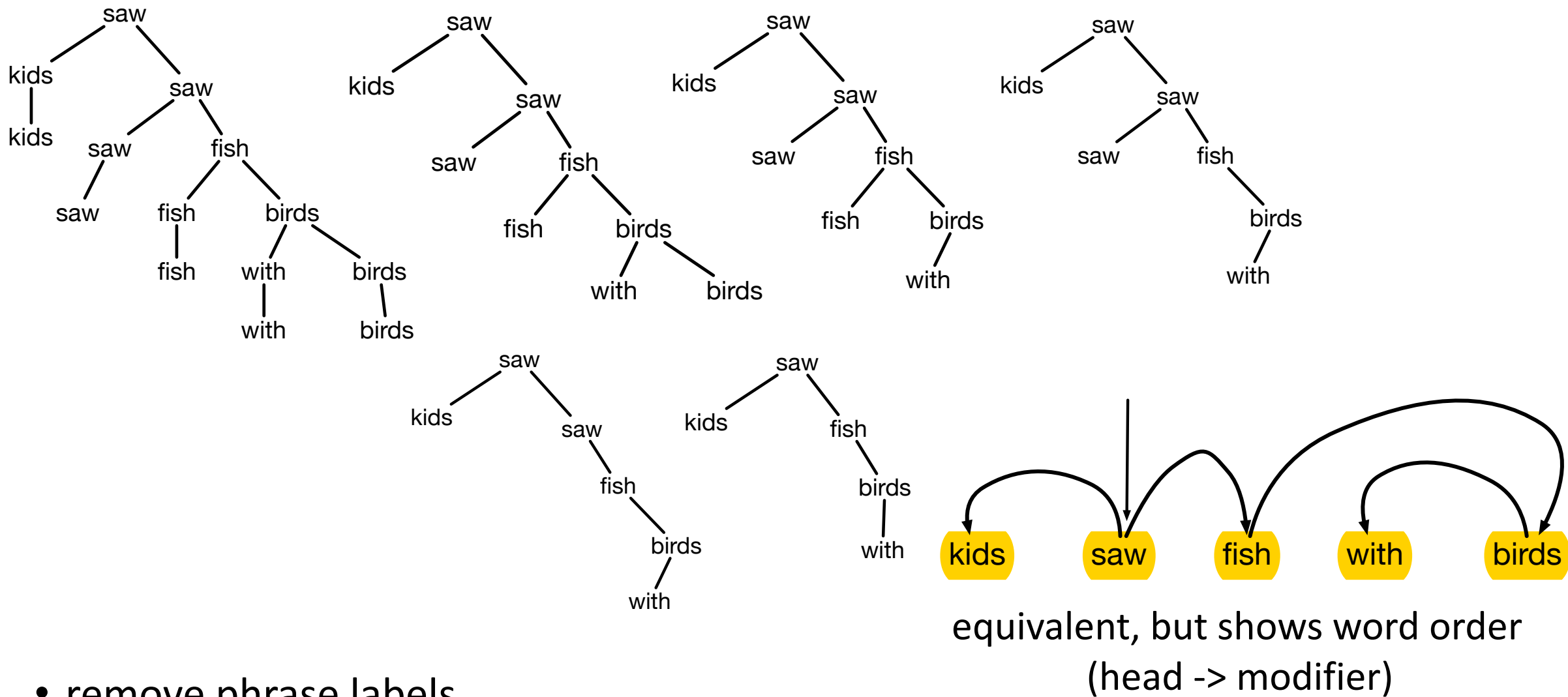
PP/binoculars → IN/with NP/binoculars

IN/with → with

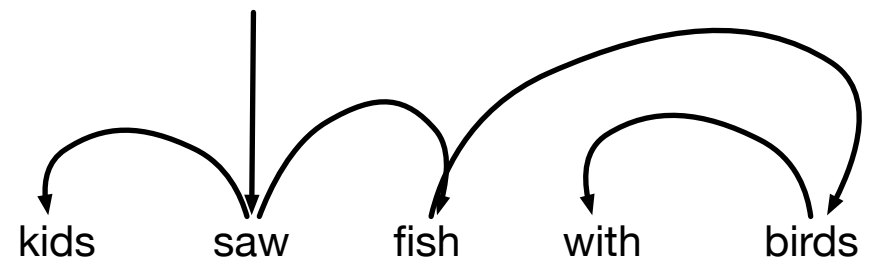
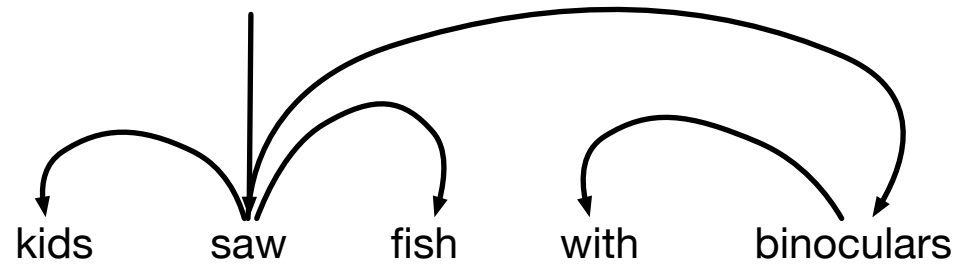
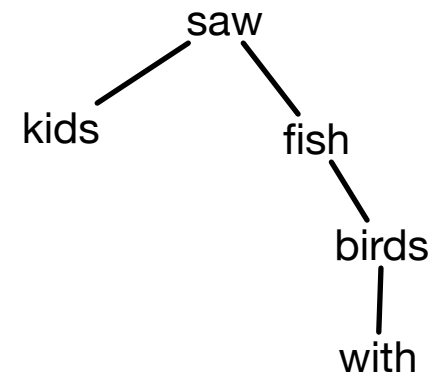
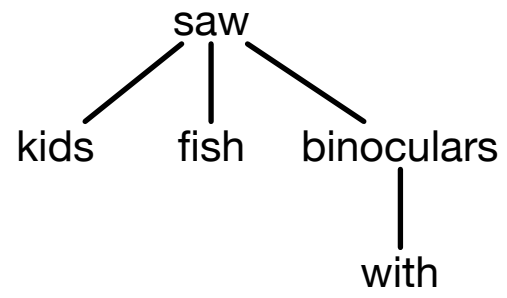
NP/binoculars → binoculars

NP/birds → birds

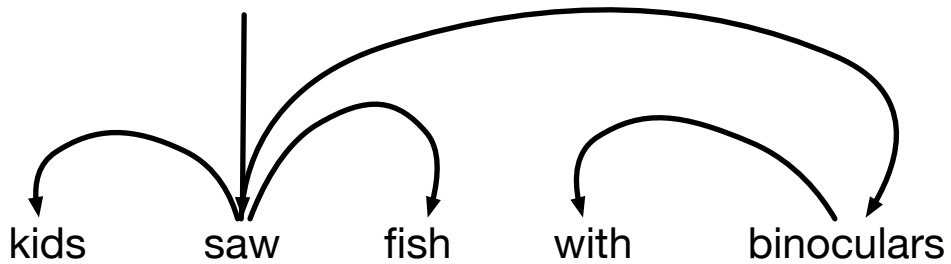
- propagate heads (Cf. last week's lecture)
- But, this leads to sparsity
- Requires lots of smoothing
- Maybe it's too much information?
- Can we lose the categories and just focus on the relationship of the words to each other?



- remove phrase labels
- throw away node and merge children if parent has the propagated label
- annotation can be in-line



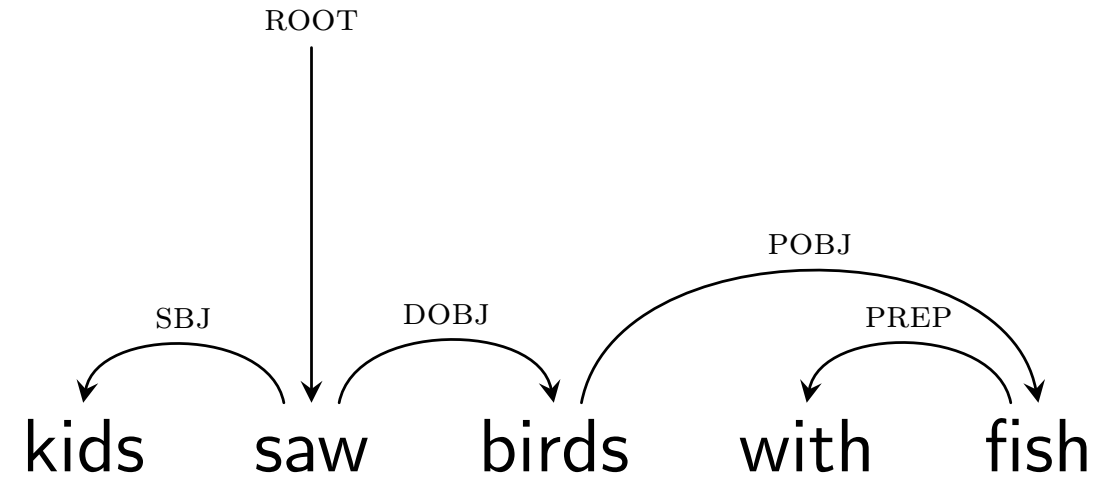
# Dependency View of Syntax



- No explicit phrase structure
- Key feature: how do words relate to each other?
- **Advantages:**
  - More conducive to free word order
  - Closer alignment between analyses of translations
  - Natural model of relationships between discontinuous words
  - Lexically grounded relationships
- **Disadvantages:**
  - Some loss of expressivity
  - For labeled examples, need to build more treebanks

# Edge Labels

- It can be useful to distinguish different kinds of head-modifier **relations**, by labeling edges
- Important relations for English:
  - subject
  - direct object
  - determiner
  - adjective modifier
- Different treebanks use different sets



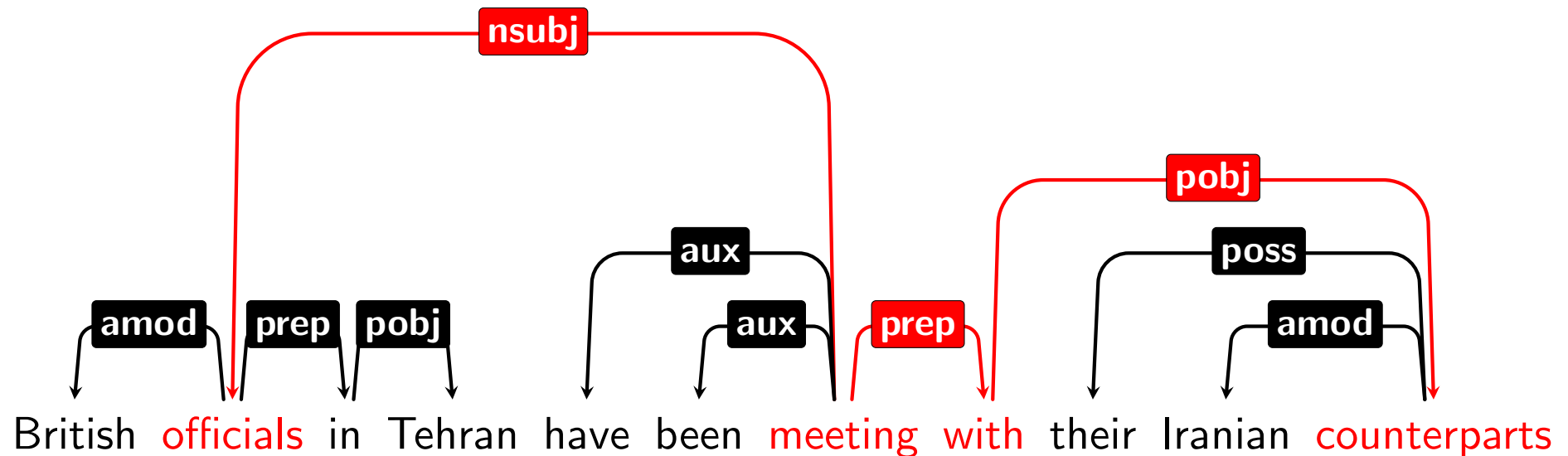
# Where do Edge Labels Come From?

- Dependency Treebanks
  - Prague Dependency Treebank: 1.5mw words of direct annotation (in Czech)
  - Universal Dependencies ([universaldependencies.org](http://universaldependencies.org)): many languages, various sizes: 1k words of Sanskrit, 10k words Kurmanji, 400k words English, 123k words Chinese, etc.
- Conversion approaches
  - rules for converting combination of constituency labels, heads, and other factors into dependency labels (deMarneffe, 2006)
- In algorithms we mostly won't discuss them but in real-world applications they're quite important



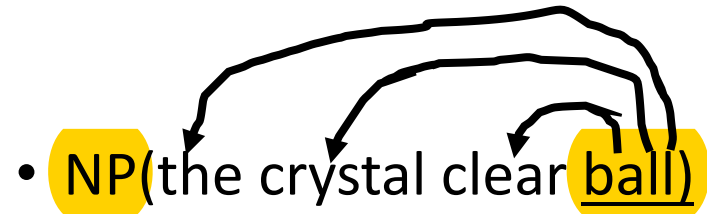
# Utility

- For **information extraction** tasks involving real-world relationships between entities, chains of dependencies provide good features



# Content vs Functional Heads


- We previously said the head was the "important" word in a phrase




# Content vs Functional Heads

- But...

- PP(into the woods) or PP(into the woods)?



- VP(was always watching tv) or VP(was always watching tv)?

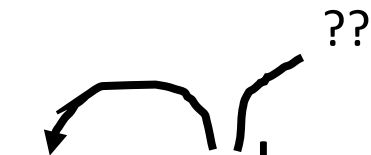


- This is characterized as having a functional head or content head
- Choice depends on goals; for universal dependencies, trend is toward content

# Head Choice

- What to do about this?

we wash cats and dogs

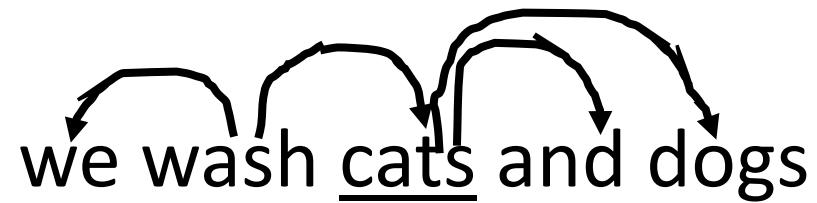


The diagram consists of a curved arrow starting from the word 'wash' and pointing to the word 'cats'. A second line starts from the word 'wash' and points upwards and to the right towards the text '??'.

# Head Choice

- What to do about this?

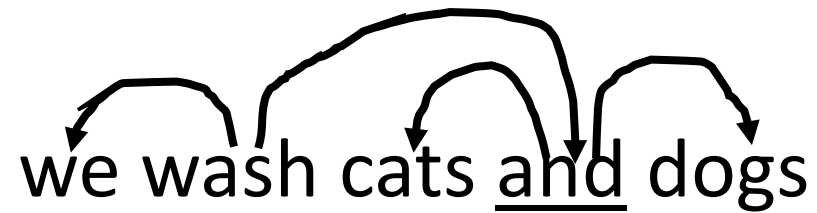
we wash cats and dogs



first conjunct is the head of the NP

# Head Choice

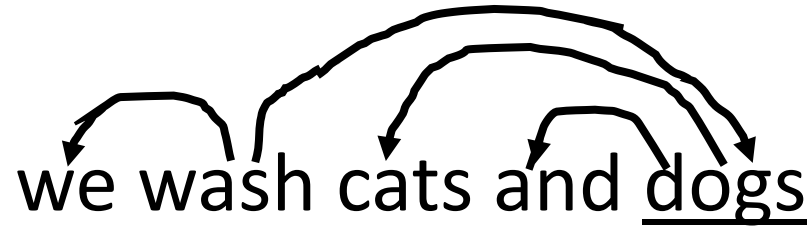
- What to do about this?



coordinatng conjunction is the head of the NP

# Head Choice

- What to do about this?

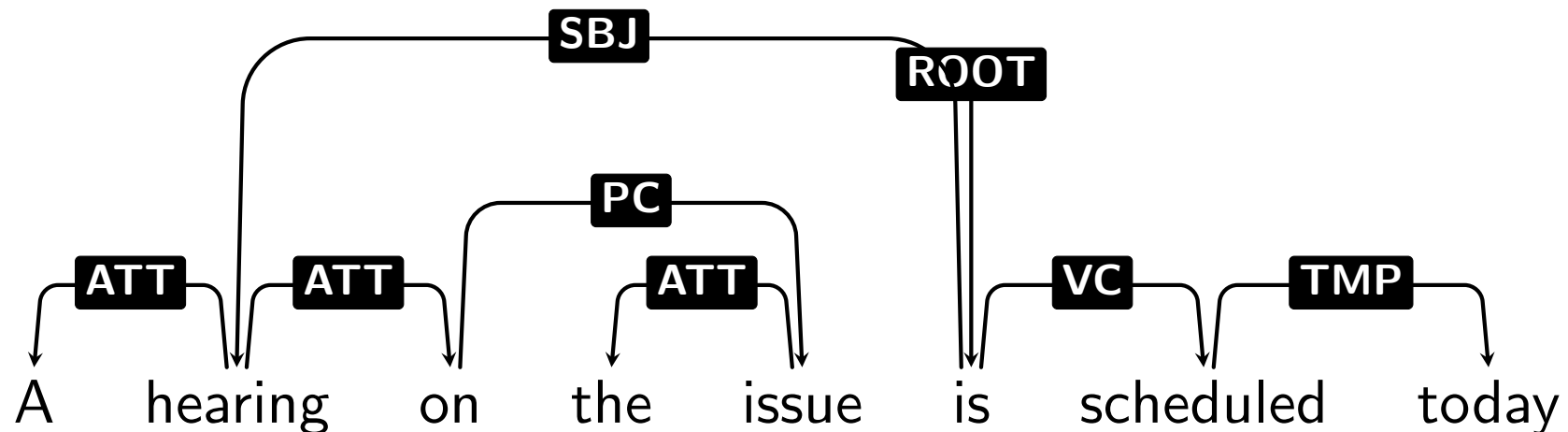


second conjunct is the head of the NP

- As usual with such things, pick a standard that is most helpful and stick with it

# Projectivity

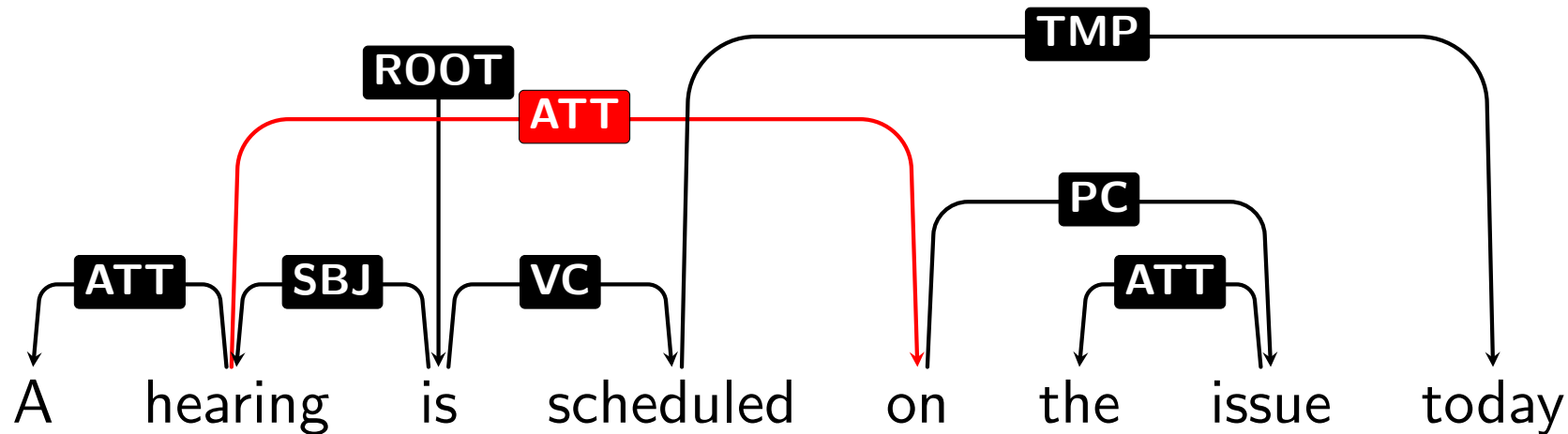
- A sentence's dependency parse is said to be projective if every subtree (node and all its descendants) occupies a *contiguous span* of the sentence
- This also means the dependency parse can be drawn on top of the sentence without any crossing edges





# Nonprojectivity

- Other sentences are nonprojective



- Nonprojectivity is rare in English, but quite common in many languages
- Note: in the above example, reordering the clauses would result in projectivity without changing the meaning

# Parsing Approaches

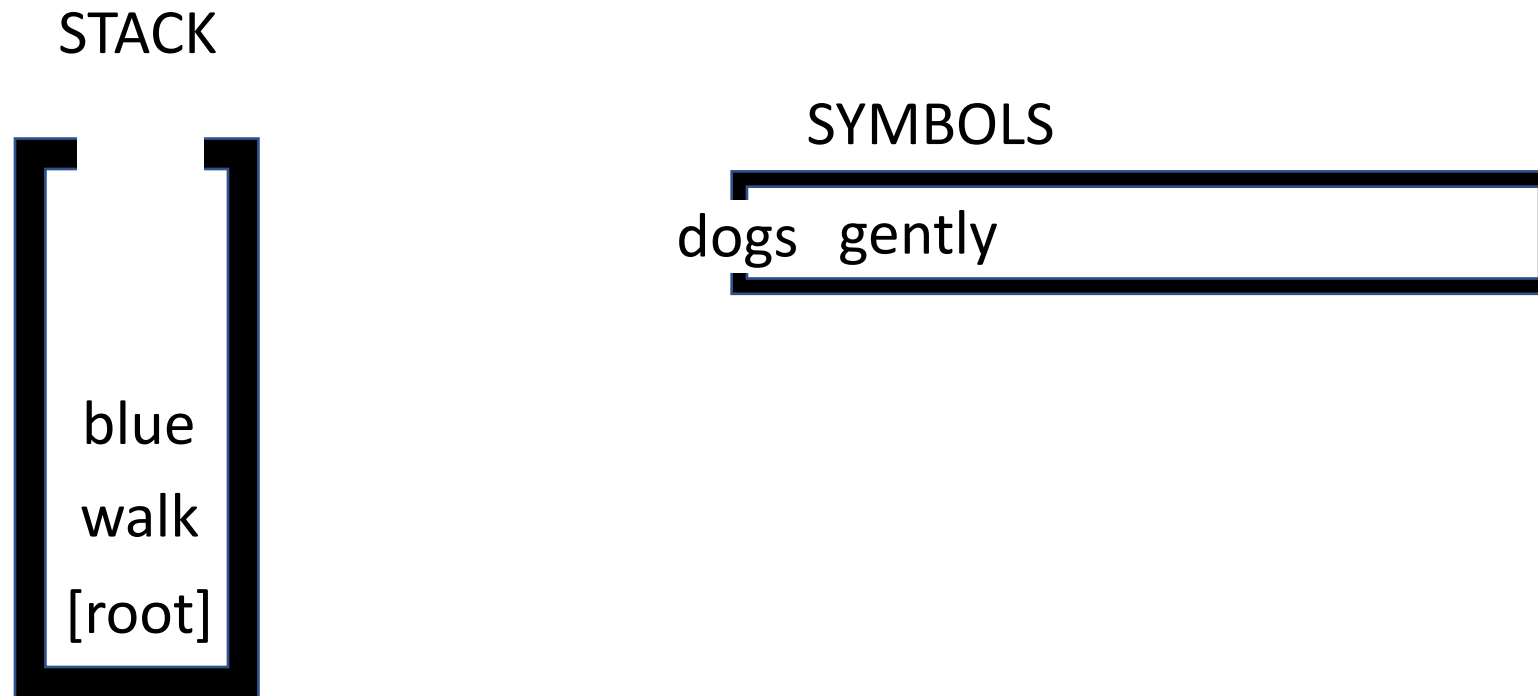
- **constituency and project**
  - pretty good for English
  - not as much for others
  - pipeline effect: quality dependent on underlying model
  - doesn't handle non-projective
- **graph based** (Chiu-Liu Edmonds; see reading)
  - can handle non-projective
  - quadratic time
  - optimal solution
  - score features must decompose across edges
- **transition based**
  - **linear** time
  - approximate solution
  - much more flexible potential feature set
  - doesn't handle non-projective

# Shift-Reduce Parsing

- Big Idea: keep a stack of words and do work at the top of the stack
- Fundamental operations:
  - SHIFT: add a word to the stack
  - REDUCE-LEFT: link top 2 stack members, pop one member, and create a dependency
  - REDUCE-RIGHT: link top 2 stack members, pop one member, and create a dependency
- Difference between REDUCE operations is which direction the dependency goes and which member gets popped
- (Note: In Jurafsky/Manning chapter, these are called "LEFT-ARC" and "RIGHT-ARC")

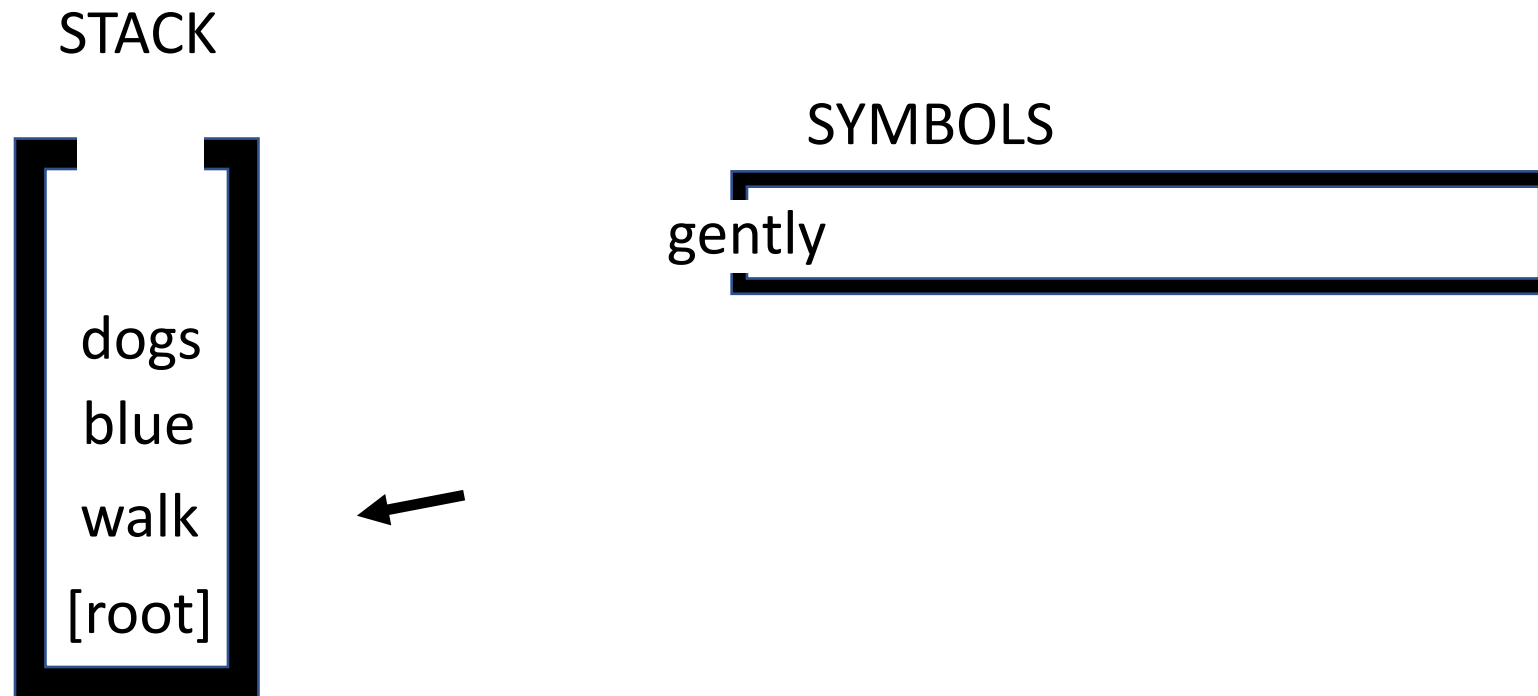
# Shift

- Very simple: just move one word over!



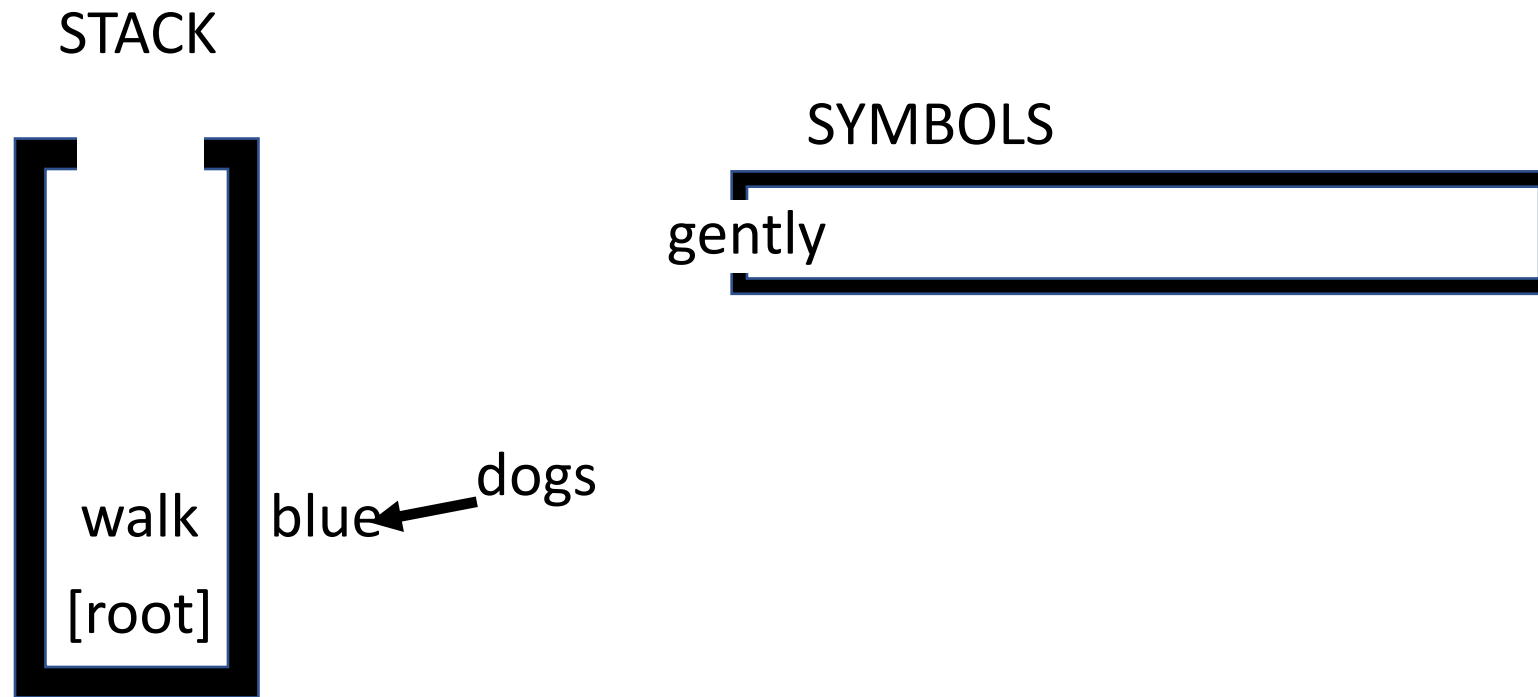
# Left Reduce

- Top of the stack is head, next on the stack is dependent.
- Pop both, form relationship, push back head



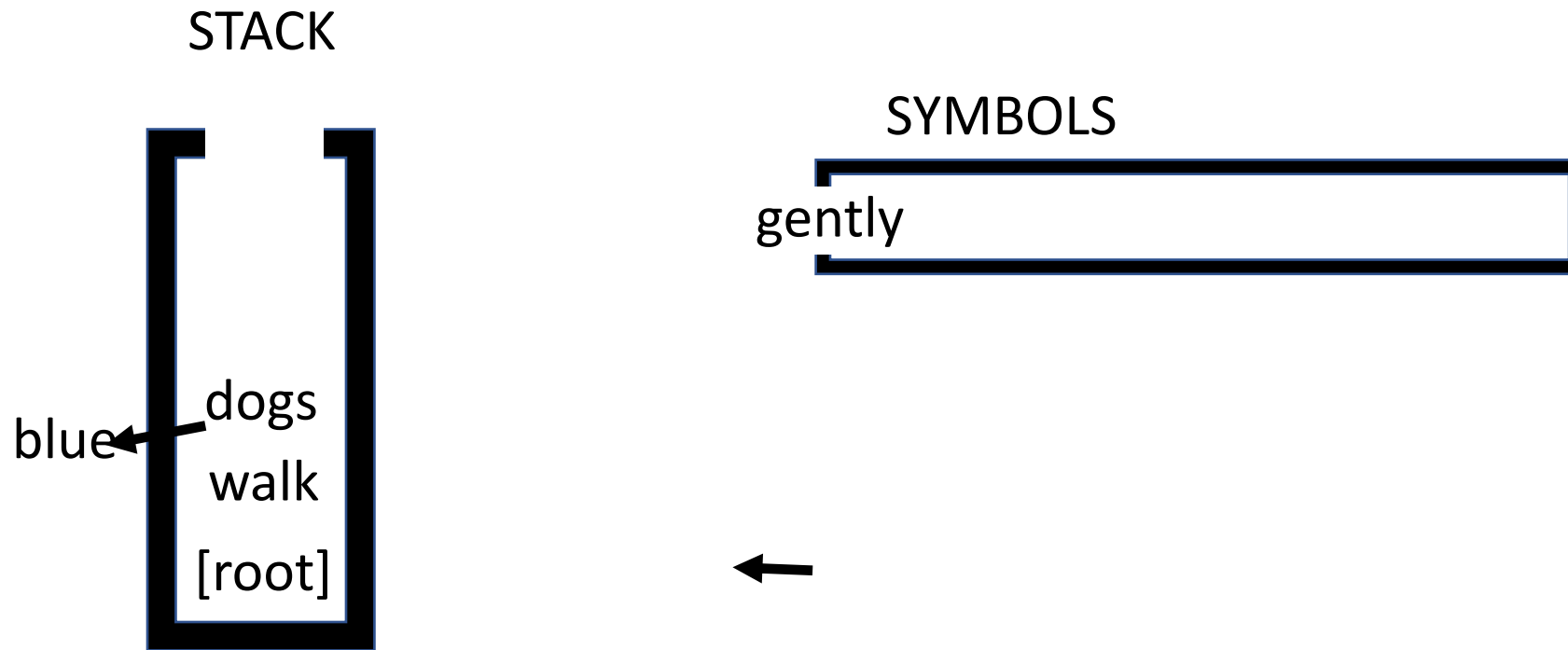
# Left Reduce

- Top of the stack is head, next on the stack is dependent.
- Pop both, form relationship, push back head



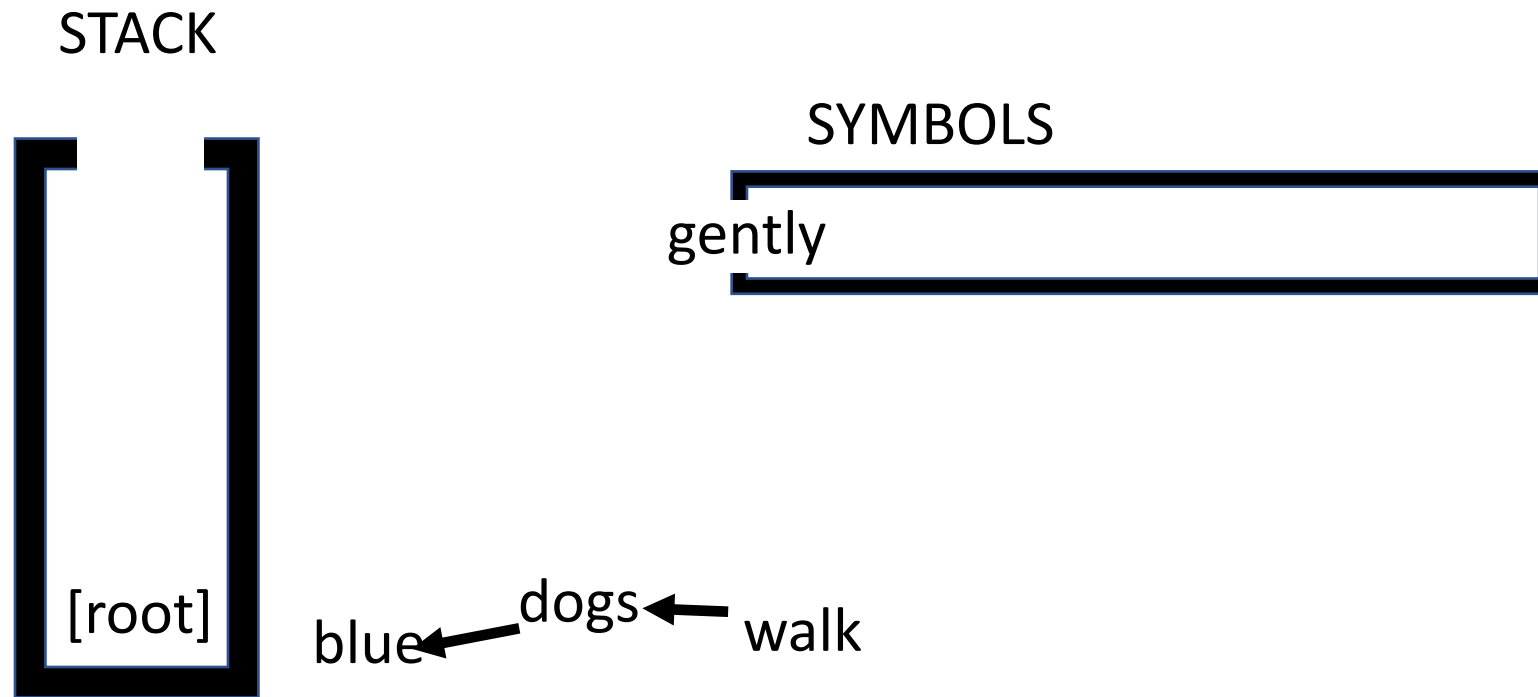
# Right Reduce

- Top of the stack is dependent, next on the stack is head.
- Pop both, form relationship, push back head



# Right Reduce

- Top of the stack is dependent, next on the stack is head.
- Pop both, form relationship, push back head





# WALKTHROUGH

book the flight through houston

stack	symbols	action	relation
[root]	book the flight through houston		

- Start off with a stack seeded with the root
- Actions are Reduce Left, Reduce Right, Shift
- (If labeled, each reduce would also have a label)
- Nothing to reduce at the beginning, so Shift

First, can we draw the tree ourselves using intuition?

- 1) Agree on a reading
- 2) Agree on a head standard

# WALKTHROUGH

book the flight through houston

stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston		

- Now, we can choose RR, RL, Shift, based on  $P(RR \mid \dots)$  vs  $P(RL \mid \dots)$  vs  $P(S \mid \dots)$
- $\dots = ??$ 
  - What are the words to be related?
  - What has been built?
  - What is in the stack (nothing other than those tokens)?
  - What is left in the symbol queue (most of the sentence)?
- Note that 'book' probably should be the main verb in the sentence
- But we want to build its children before we build it
- So, a well-trained parser should "Shift" here

# WALKTHROUGH

book the flight through houston

stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston		

- Again we can choose...
  - "the" is the head of "book"? (L)
    - No, that would be really weird...looking ahead, "the" should be a child of "flight"
  - "book" is the head of "the"? (R)
    - That would make sense in the noun phrase "the book" but not here
    - It's possible a parser could get confused...
    - Word order does matter!
- Again, a well-trained parser should "Shift" here

# WALKTHROUGH

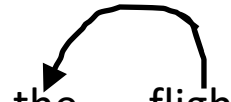
book the flight through houston

stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston		

- Now consider
  - "flight" is the head of "the"? (L)
    - Seems like a likely pairing. If we ran a POS tagger we would ask "noun head of a det?" and be confident. Order is good. And no other surrounding words seem like better candidates to wait.
  - "the" is the head of "flight"? (R)
    - No, that seems backwards for the reasons above
  - should we shift? (S)
    - No, since "L" seems reasonable

# WALKTHROUGH

book   the   flight   through   houston

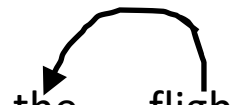


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston		

- "flight" is the head of "book"? (L)
  - If it's two nouns, that might make sense (noun compound). We might note that there's no other likely verb in the sentence though, so hopefully this wouldn't happen. But it's a place we could go wrong!
- "book" is the head of "flight"? (R)
  - If "book" is a verb this seems likely. However, in this case we want to specify a particular flight (the "flight through houston"). So hopefully we can encourage the parser to defer this decision
- should we shift? (S)
  - We're asking a lot of the model here but that is the "correct" action to take

# WALKTHROUGH

book   the   flight   through   houston




stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston		

- "through" is the head of "flight"? (L)
  - We're assuming content heads, so this would be backwards
- "flight" is the head of "through"? (R)
  - In a prepositional phrase, but then "through" would come first
- should we shift? (S)
  - Yes

# WALKTHROUGH

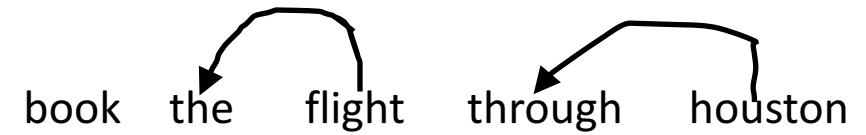
book   the   flight   through   houston



stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston			

- "houston" is the head of "through"? (L)
  - Yes, in a prepositional phrase, with content heads
- "through" is the head of "houston"? (R)
  - No, because we're content-head not function-head
- should we shift? (S)
  - No more symbols so this should have a probability of 0

# WALKTHROUGH

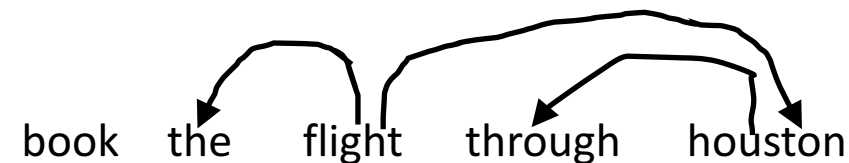


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston			

- "houston" is the head of "flight"? (L)
  - No, "flight" is more important to the sentence. When joining a noun phrase and prep phrase the noun phrase should dominate
- "flight" is the head of "houston"? (R)
  - Yes, for the reasons above
- should we shift? (S)
  - No more symbols so this should have a probability of 0



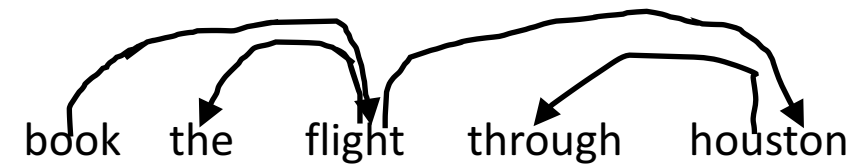
# WALKTHROUGH



stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight			

- "flight" is the head of "book"? (L)
  - If it was a noun compound, maybe, but then "flight" wouldn't already have a child (and we'd have no verbs)
- "book" is the head of "flight"? (R)
  - Yes, if "flight" is a verb. Note that we were presented with this option before and shifted

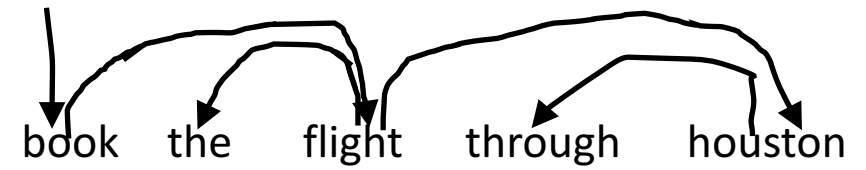
# WALKTHROUGH



stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight		R	book -> flight
[root] book			

- "book" is the head of [root]? (L)
  - No, [root] should never be a dependent
- [root] is the head of "book"? (R)
  - Yes, but we should not look for a head for "book" if it is not the head of the sentence.

# WALKTHROUGH



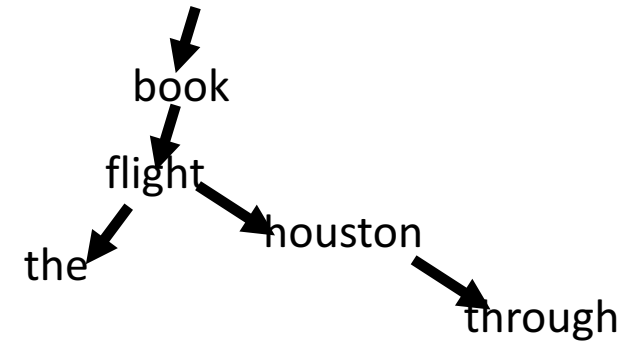
stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight		R	book -> flight
[root] book		R	[root] -> book
[root]		Done	

# How to Determine a Step Sequence Given a Tree?

- Much less uncertainty than the parsing walkthrough. There is a deterministic algorithm:
  - If you can Reduce Left (i.e. if the result is legitimate), do it, or
  - If you can Reduce Right and all dependents of the top have been added, do it, or
  - Shift

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

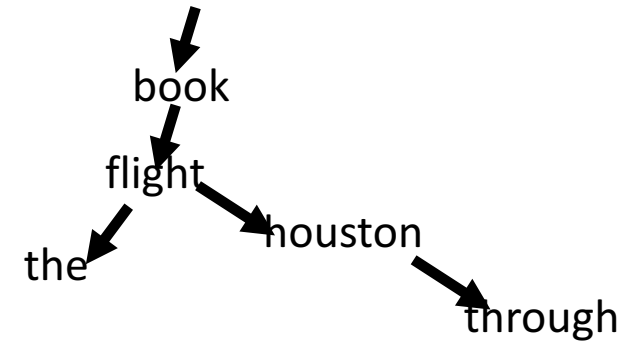


stack	symbols	action	relation
[root]	book the flight through houston		

1. Nothing to reduce
2. Nothing to reduce
3. Shift!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

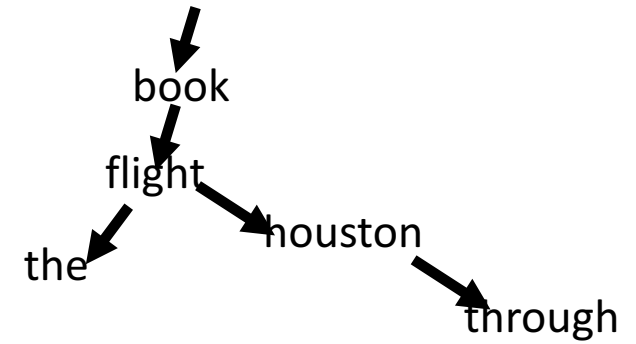


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston		

1. "book" is not the head of [root]
2. [root] is the head of "book" but "book" has unsatisfied dependencies
3. Shift!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

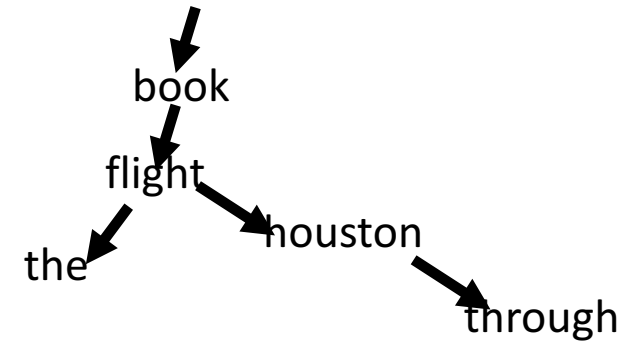


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston		

1. "the" is not the head of "book"
2. "book" is not head of "the"
3. Shift!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift



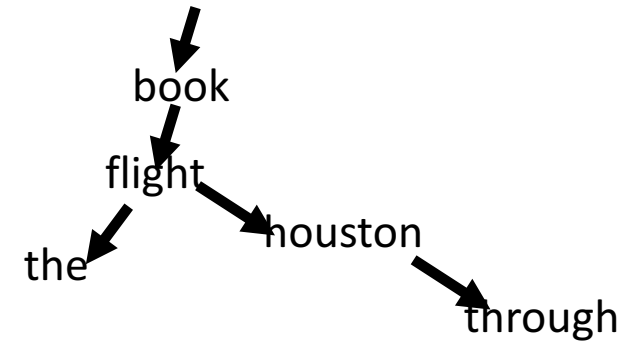
stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston		

1. "flight" is head of "the". LR!



# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

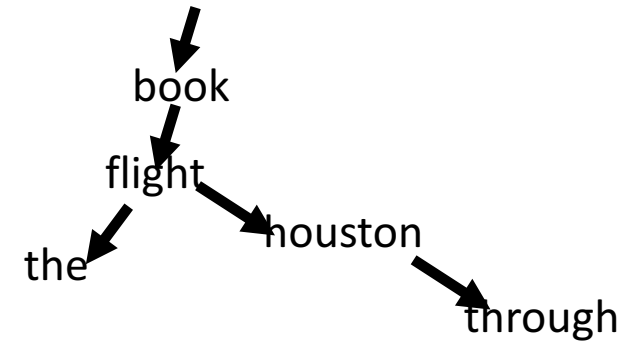


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston		

1. "flight" is not head of "book"
2. "book" is head of "flight" but "flight" has unsatisfied dependents
3. Shift!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

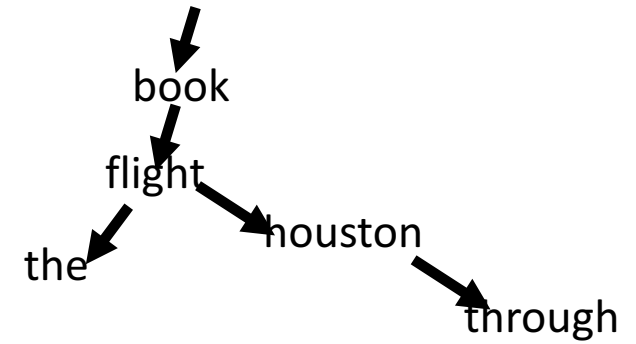


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston		

1. "through" is not head of "flight"
2. "flight" is not head of "through"
3. Shift!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

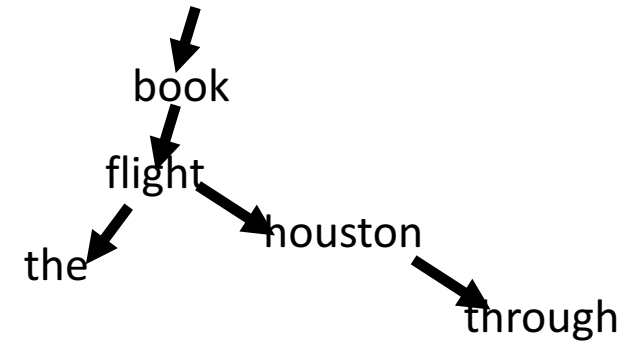


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston			

1. "houston" is head of "through". LR!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

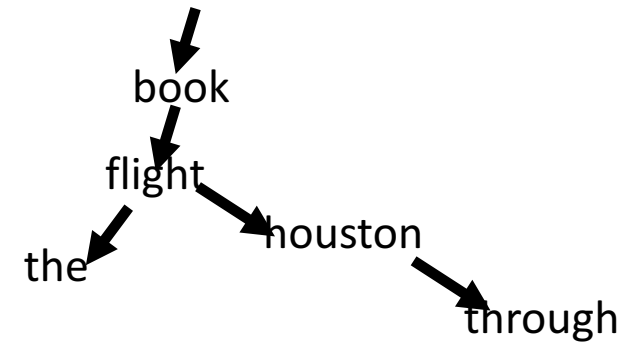


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston			

1. "houston" is not head of "flight"
2. "flight" is head of "houston" and all other dependencies of "houston" are satisfied. RR!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

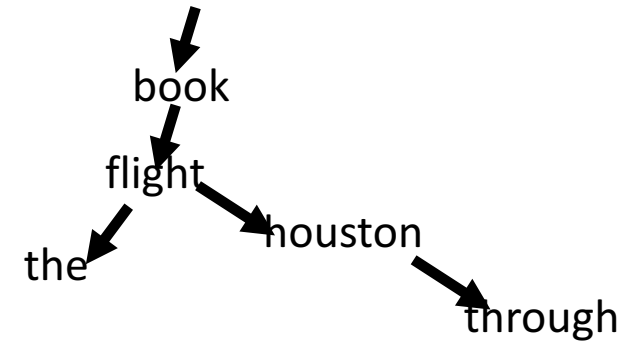


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight			

1. "flight" is not head of "book"
2. "book" is head of "flight" and all other dependencies of "flight" are satisfied. RR!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift

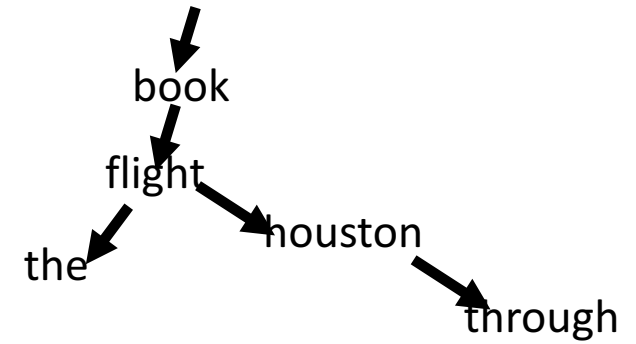


stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight		R	book -> flight
[root] book			

1. "book" is not head of [root]
2. [root] is head of "book" and all other dependencies of "book" are satisfied. RR!

# Generating a step sequence

1. LR
2. RR and all deps of dep satisfied
3. Shift



stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	through houston	L	the <- flight
[root] book flight	through houston	S	
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight		R	book -> flight
[root] book		R	[root] -> book
[root]		Done	

# Training

- Given a sentence and its correct tree, there is a deterministic sequence of shift-reduce operations that will produce the tree from the sentence
- Thus, given a state in the process of decoding, there is one correct operation
- So train it like a tagger!
  - Naive Bayes
  - Perceptron
  - Logistic Regression



# Features

- These can generally come from anything we can see
  - What have we seen (the stack)?
  - What will we see (the buffer)?
  - What have we done (the already-extracted links)?
- Using all of this can be sparse. Use the most 'actionable' elements
  - What did we just see (the top of the stack)?
  - What will we soon see (the next word or two)?
  - What have we done that is relevant (links to words in the top of the stack)?
- Granular bits:
  - word
  - POS tag

# Feature Templates

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	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

**Figure 14.9** Standard feature templates for training transition-based dependency parsers. In the template specifications  $s_n$  refers to a location on the stack,  $b_n$  refers to a location in the word buffer,  $w$  refers to the wordform of the input, and  $t$  refers to the part of speech of the input.

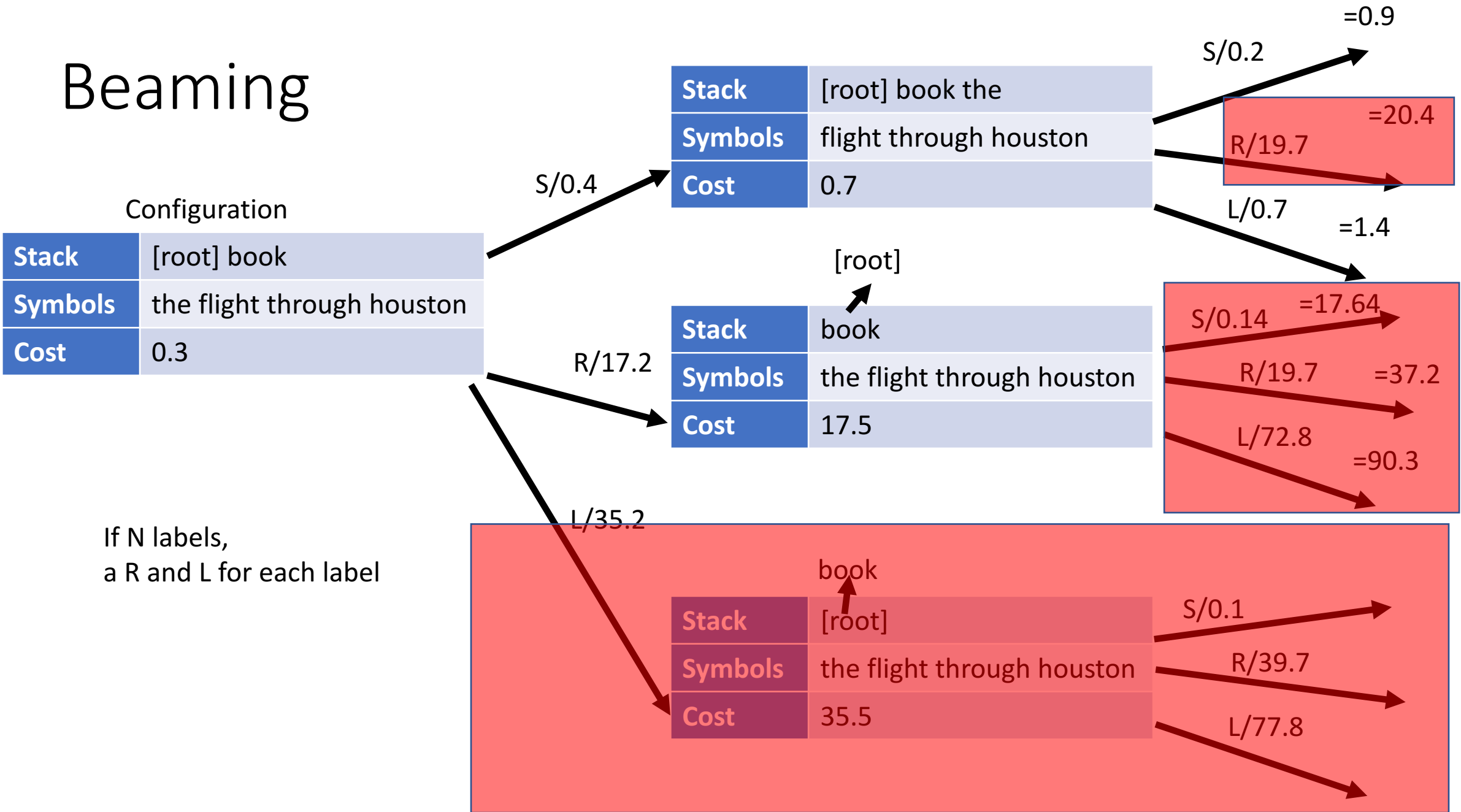
Pretty good starting set

Refine for your language, use case, etc.

# Beam Decoding

- Shift-Reduce algorithm is greedy; early bad decision can doom later possibilities
- Beam search is a general way to be "not as greedy"
  - Instead of the best step, take  $k$  best steps
  - From each of those  $k$  steps, take  $k$  best next steps
  - Keep only the top  $k$  of those  $k^2$  steps
  - What is the runtime for a sentence of length  $n$  and beam of width  $k$ ?
- Beam search is widely used in many other NLP tasks:
  - MT
  - Constituency Parsing
  - ASR
  - Summarization
  - Other generation...

# Beaming



# Issues With "Arc-Standard" Shift-Reduce

stack	symbols	action	relation
[root]	book the flight through houston	S	
[root] book	the flight through houston	S	
[root] book the	flight through houston	S	
[root] book the flight	<b>"book" and "flight" first come together</b>		the <- flight
[root] book flight			
[root] book flight through	houston	S	
[root] book flight through houston		L	through <- houston
[root] book flight houston		R	flight -> houston
[root] book flight		R	book -> flight
[root] book	<b>"book" and "flight" relation is built</b>		[root] -> book
[root]			Done

# "Arc-Eager" Variant of Shift-Reduce

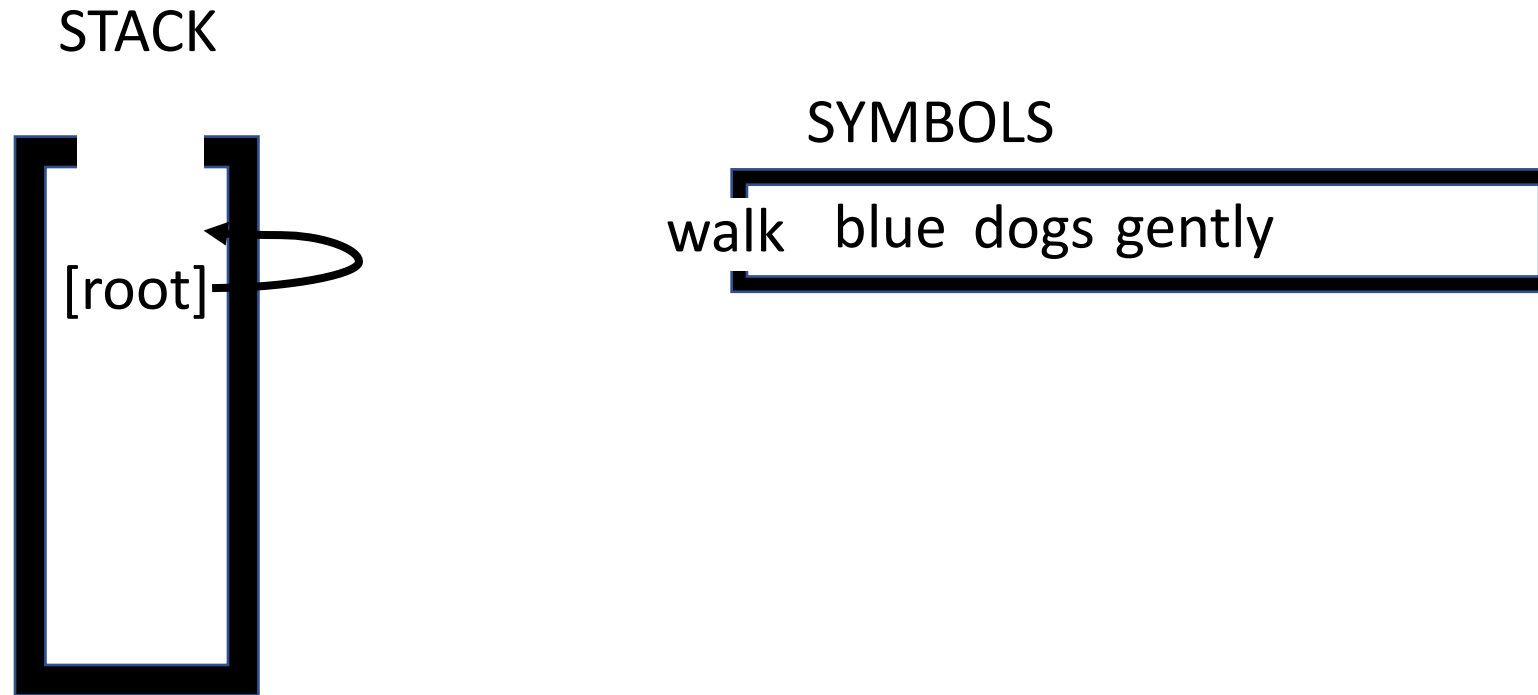
- Idea: try to form a relation as early as possible
- Arc-Standard: strictly bottom-up
- Arc-Eager: combination of bottom-up and top-down
- Potential harm: might not end up with a tree

# Arc-Eager Actions

- SHIFT: add a word to the stack
- LEFT-ARC: link top stack member with next symbol(head); pop top of the stack
- RIGHT-ARC: link top stack member (head) with next symbol; shift next symbol onto the stack
- **REDUCE**: pop top of the stack

# Right-Arc

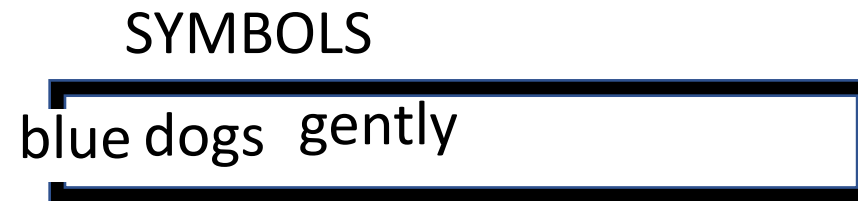
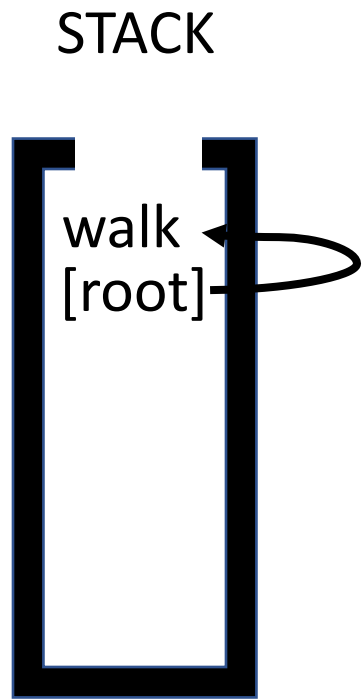
- Top of stack is head of front symbol AND push front symbol





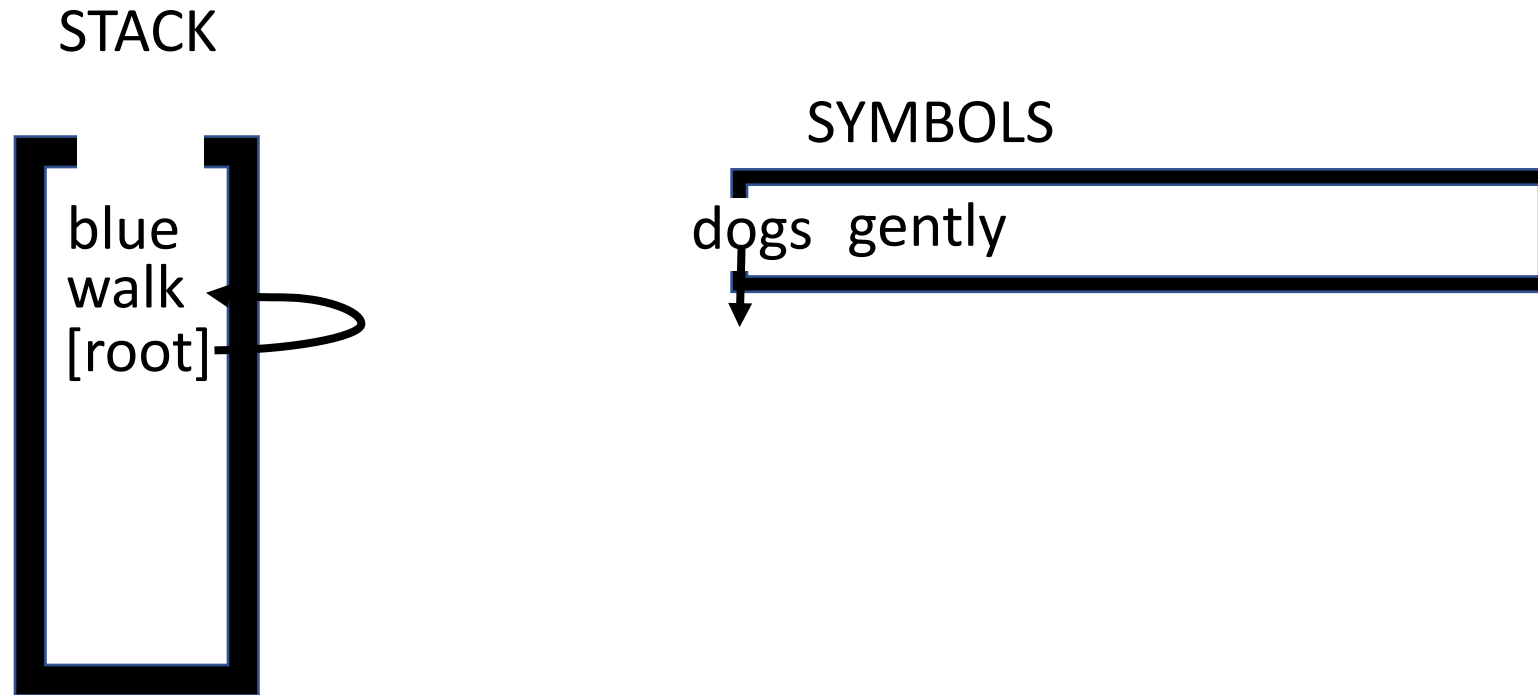
# Shift

- As before



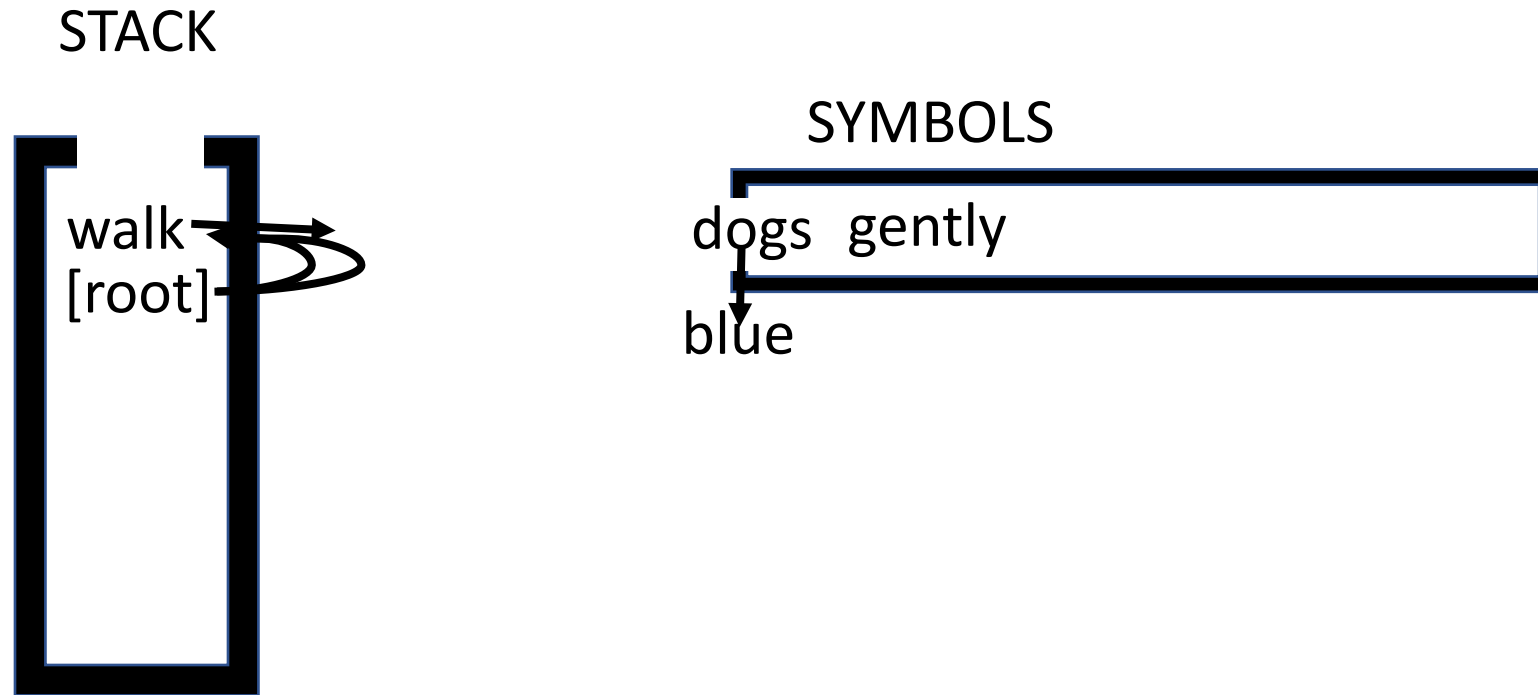
# Left-Arc

- Front symbol is head of top of the stack AND pop top of the stack



# Right-Arc

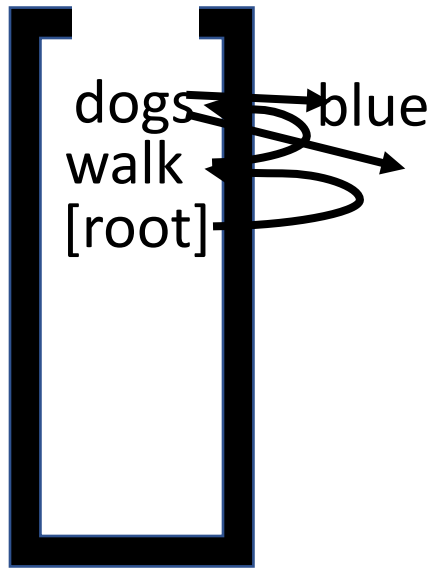
- Top of stack is head of front symbol AND push front symbol



# Reduce (NEW!)

- Pop top of stack

STACK



SYMBOLS



# Arc-Eager WALKTHROUGH


book the flight through houston

stack	symbols	action	relation
[root]	<b>book</b> the flight through houston		

We don't have to wait for descendants; "book" is the main verb so make it the direct child of [root]  
Push the child onto the stack (Right-Arc)

---


# Arc-Eager WALKTHROUGH

 book the flight through houston

stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston		

No relationship between "book" and "the" (Shift)

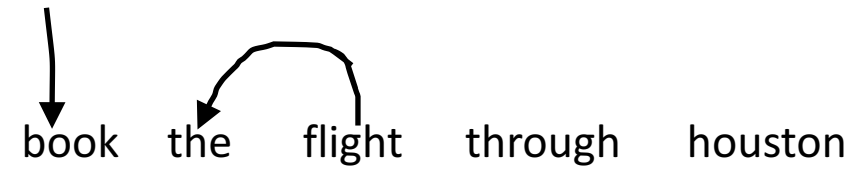
# Arc-Eager WALKTHROUGH

 book the flight through houston

stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston		

"flight" is head of "the". Pop the dependent off the stack (Left-Arc)

# Arc-Eager WALKTHROUGH

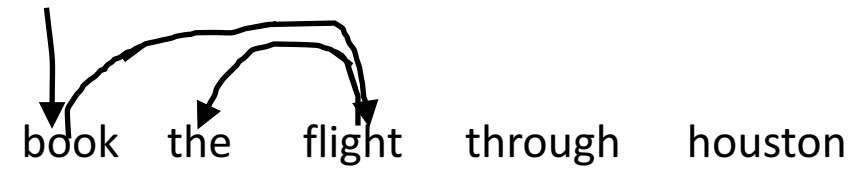


stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston		

"book" is head of "flight". Push the dependent on the stack (Right-Arc)



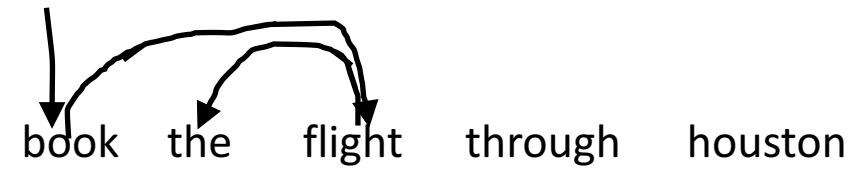
# Arc-Eager WALKTHROUGH



stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston	R	book -> flight
[root] book flight	<b>through</b> houston		

No direct relationship between "flight" and "through" but more work to do (Shift)

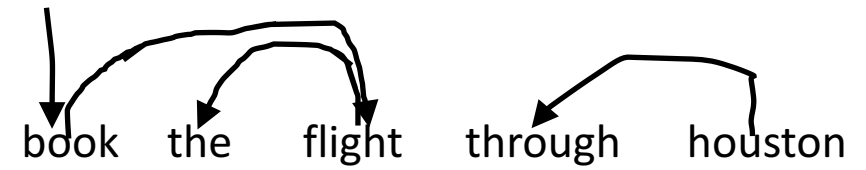
# Arc-Eager WALKTHROUGH



stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston	R	book -> flight
[root] book flight	<b>through</b> houston	S	
[root] book flight through	<b>houston</b>		

"houston" is the head of "through"; pop the dependent (Left Arc)

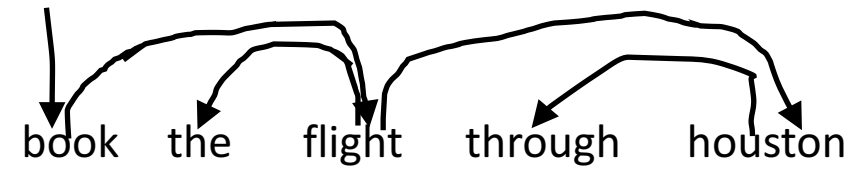
# Arc-Eager WALKTHROUGH



stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston	R	book -> flight
[root] book flight	<b>through</b> houston	S	
[root] book flight through	<b>houston</b>	L	through <- houston
[root] book flight	<b>houston</b>		

"flight" is the head of "houston"; push the dependent (Right Arc)

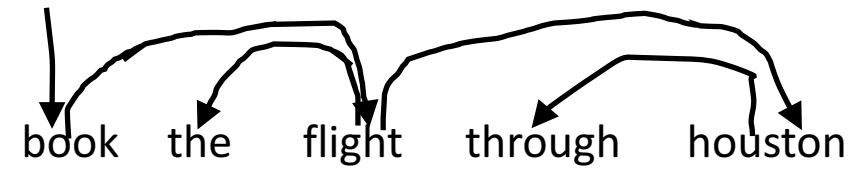
# Arc-Eager WALKTHROUGH



stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston	R	book -> flight
[root] book flight	<b>through</b> houston	S	
[root] book flight through	<b>houston</b>	L	through <- houston
[root] book flight	<b>houston</b>	R	flight -> houston
[root] book flight houston			

No more symbols to combine so reduce and we're done...hopefully with a tree and not a forest!

# Arc-Eager WALKTHROUGH



stack	symbols	action	relation
[root]	<b>book</b> the flight through houston	R	[root] -> book
[root] book	<b>the</b> flight through houston	S	
[root] book the	<b>flight</b> through houston	L	the <- flight
[root] book	<b>flight</b> through houston	R	book -> flight
[root] book flight	<b>through</b> houston	S	
[root] book flight through	<b>houston</b>	L	through <- houston
[root] book flight	<b>houston</b>	R	flight -> houston
[root] book flight houston		Reduce	
[root] book flight		Reduce	
[root] book		Reduce	
[root]		Done	

# Oracle Arc-Eager

- Much simpler than arc-standard!
- If top of stack is head of next symbol, Right-Arc
- If next symbol is head of top of stack, Left-Arc
- If no relationship and top of stack has no children in the symbol list, Shift
- Else, Reduce

# Criteria for Choosing

- Efficiency (in practice both memory and runtime)
- Incrementality (whole sentence at once or are partial parses needed?)
- Retrainingability?
- Most dominant: score (balanced f-measure of labeled/unlabeled precision/recall)

# Typical State-of-the art claims in a paper

System	UAS	LAS	Speed	
baseline greedy parser	91.47	90.43	0.001	
Huang and Sagae (2010)	92.10		0.04	
Zhang and Nivre (2011)	92.90	91.80	0.03	
Choi and McCallum (2013)	92.96	91.93	0.009	
Ma et al. (2014)	93.06			
Bohnet and Nivre (2012) <sup>†‡</sup>	93.67	92.68	0.4	
Suzuki et al. (2009) <sup>†</sup>	93.79			
Koo et al. (2008) <sup>†</sup>	93.16			
Chen et al. (2014) <sup>†</sup>	93.77			
beam size				
training	decoding			
100	100	<b>93.28</b>	<b>92.35</b>	0.07
100	64	93.20	92.27	0.04
100	16	92.40	91.95	0.01

Table 5: Results on WSJ. Speed: sentences per second. <sup>†</sup>: semi-supervised learning. <sup>‡</sup>: joint POS-tagging and dependency parsing models.

(Zhou et al., 2015)



# Summary

- While constituency parses give hierarchically nested phrases, dependency parses represent syntax with trees whose edges connect words in the sentence. Edges are often labeled with relations like subject
- Head rules govern how dependency trees are formed (whether via annotation or conversion from constituency)
- Transition-based parsing algorithms provide good accuracy and speed trade off for projective parsing