Empirical Methods in Natural Language Processing
Lecture 18
Dependency Parsing

(some slides from Sharon Goldwater)

9 November 2016

Vanilla PCFGs: no lexical dependencies

Replacing one word with another with the same POS will never result in a different parsing decision, even though it should!

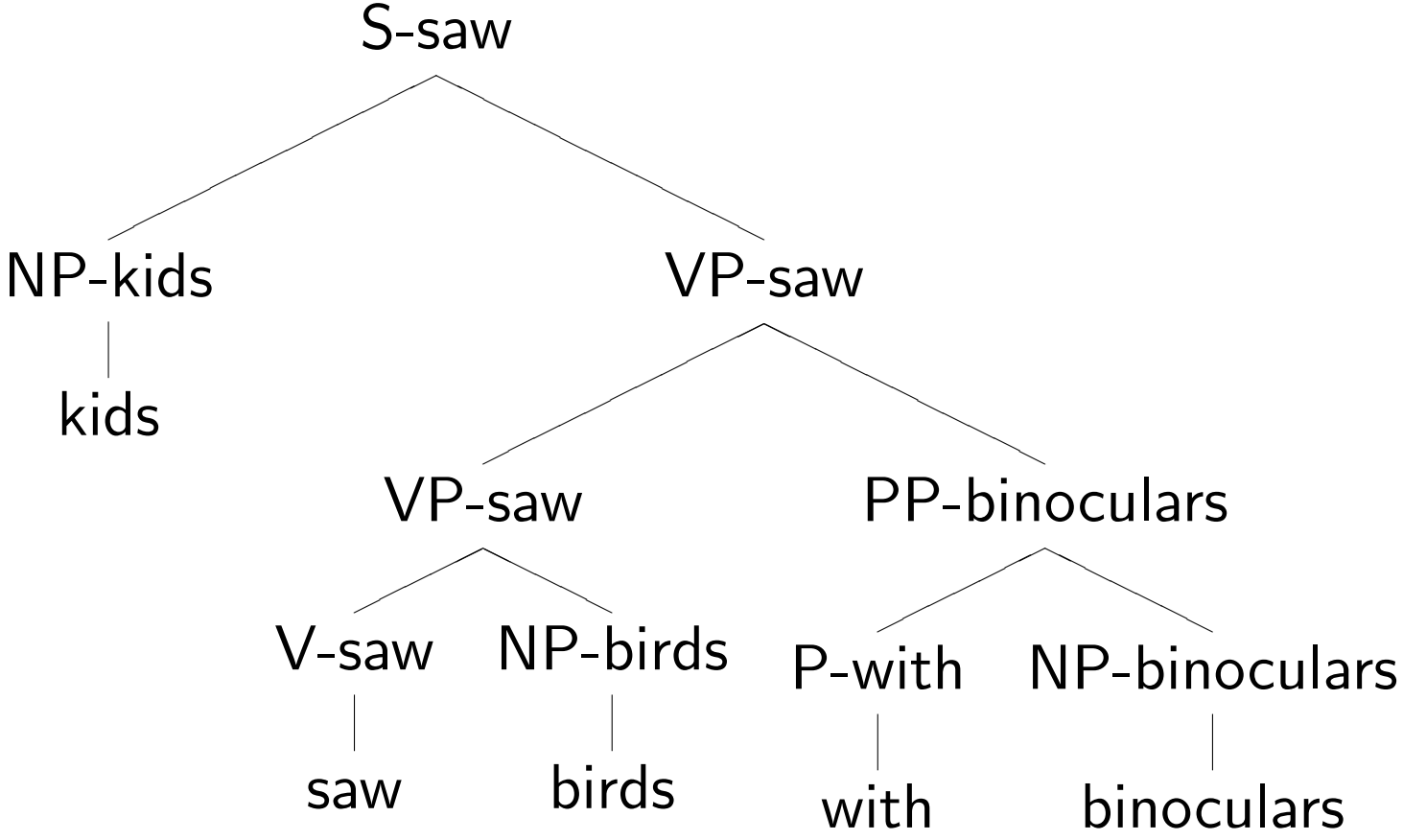
- kids saw birds with fish vs.
  kids saw birds with binoculars

- 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

A way to fix PCFGs: lexicalization

Create new categories, this time by adding the **lexical head** of the phrase (note: N level under NPs not shown for brevity):



- Now consider:

`VP-saw` → `VP-saw PP-fish` vs. `VP-saw` → `VP-saw PP-binoculars`

Practical issues, again

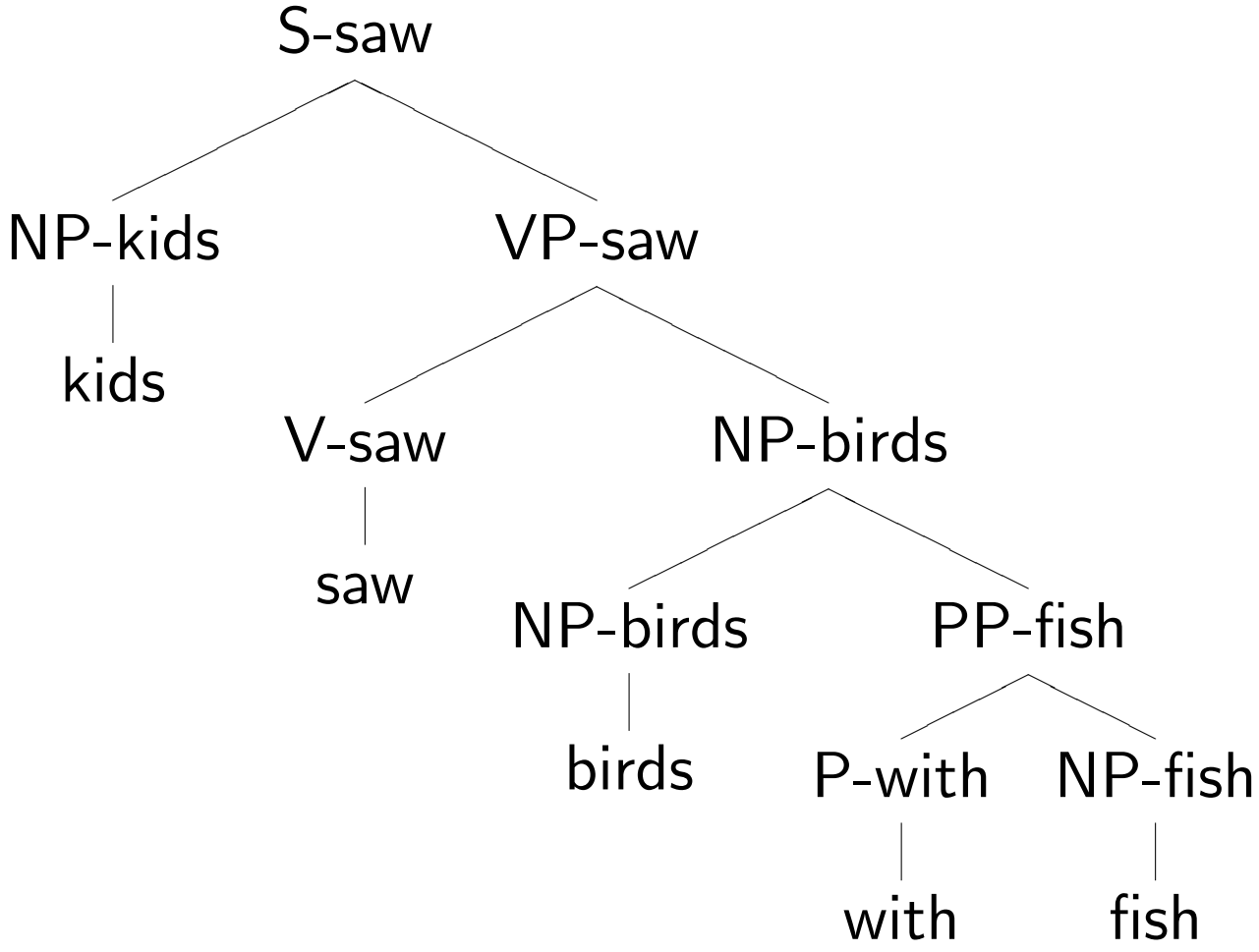
- Leads to huge grammar blowup and very sparse data (bad!)
  - There are fancy techniques to address these issues. . .
  - But: Do we really need phrase structures in the first place? Not always!

- Today: Syntax without constituent structure.

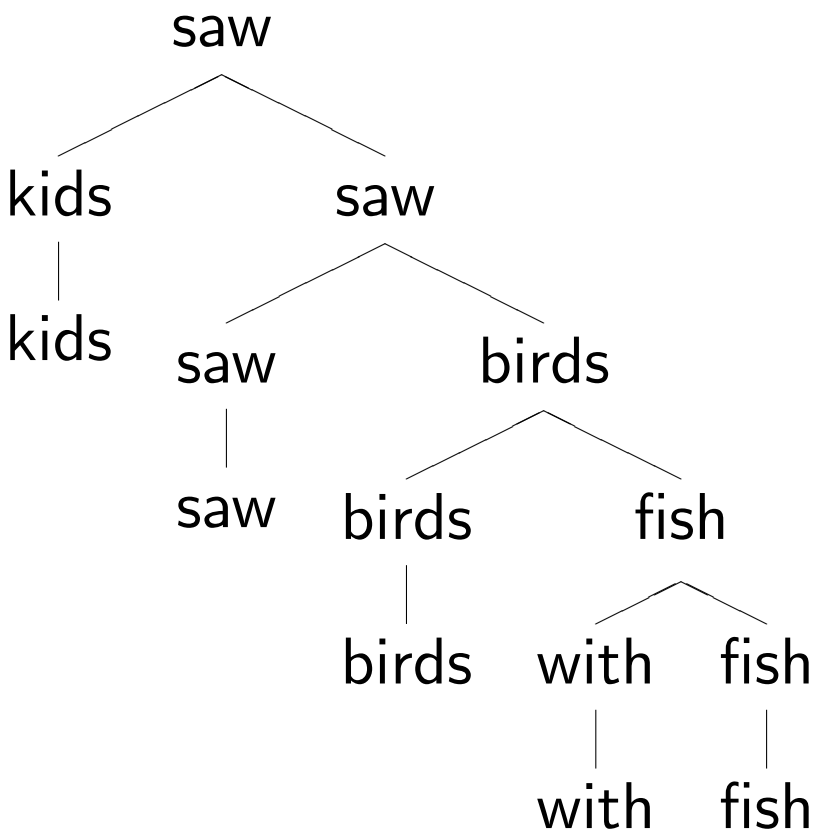
Outline

1. Dependencies: what/why
2. Transforming constituency $\rightarrow$ dependency parse
3. Direct dependency parsing
   - Transition-based (shift-reduce)
   - Graph-based

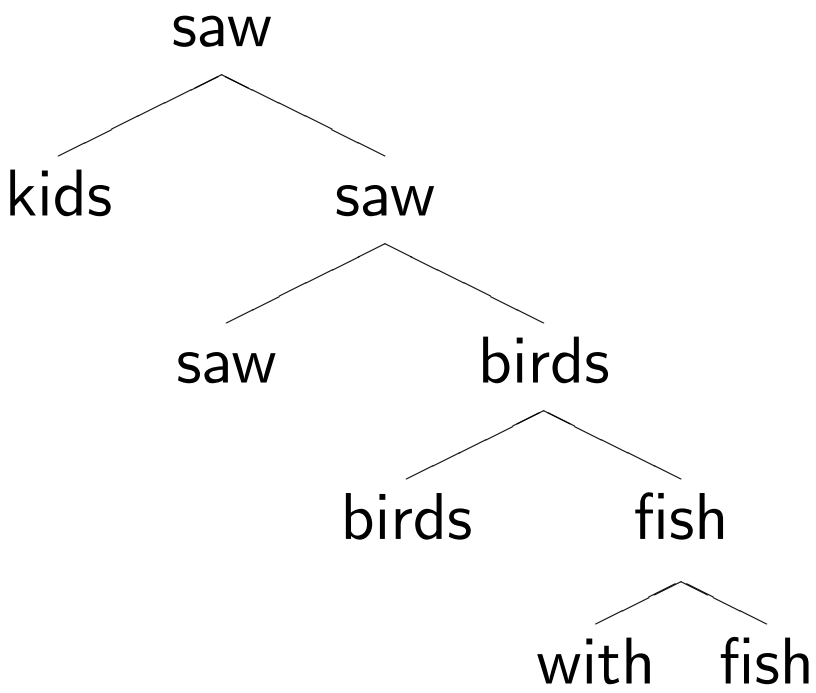
Lexicalized Constituency Parse



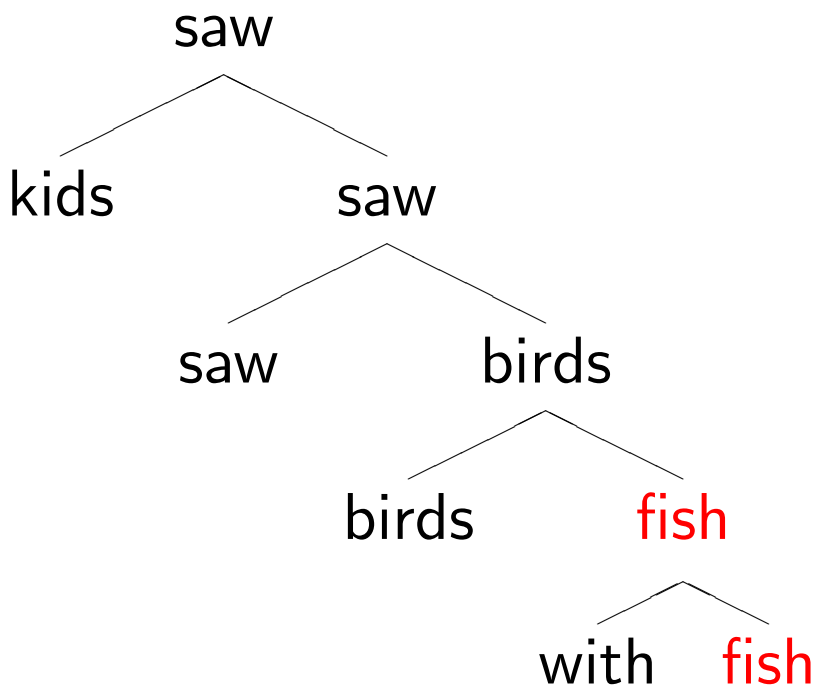
. . . remove the phrasal categories. . .



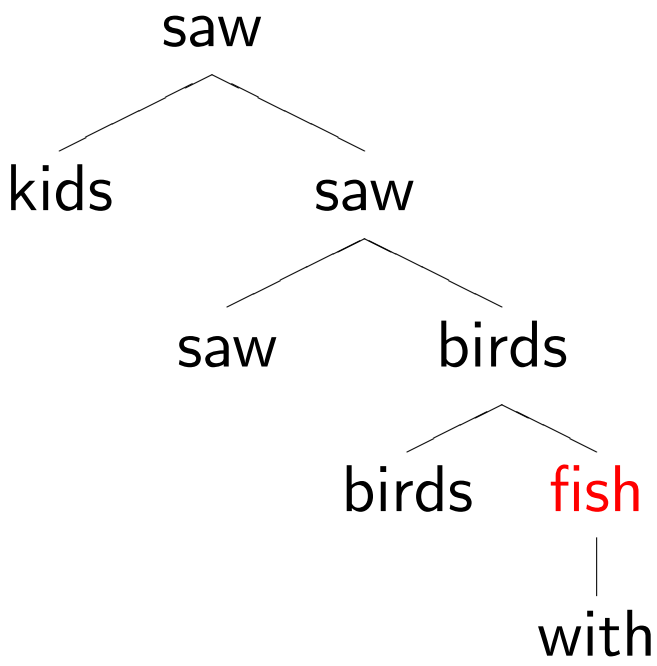
. . . remove the (duplicated) terminals. . .



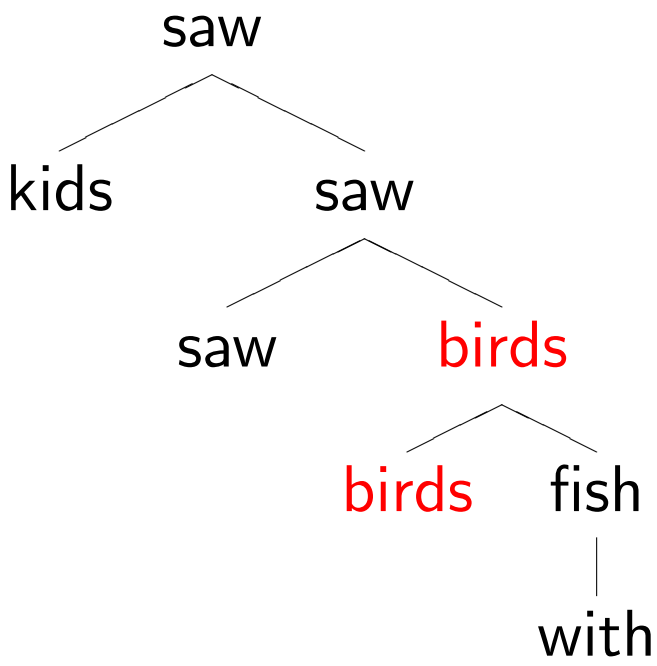
. . . and collapse chains of duplicates. . .



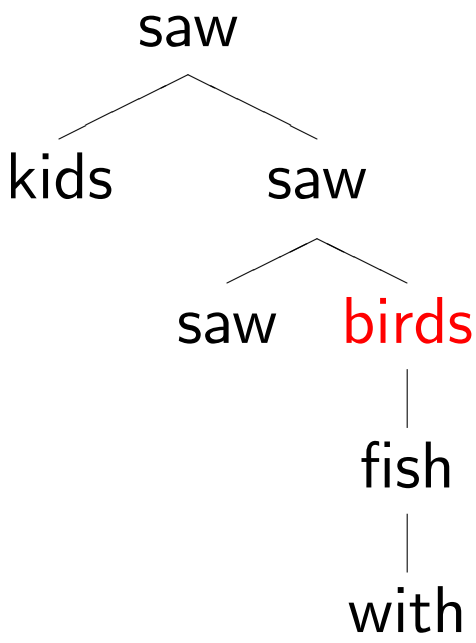
. . . and collapse chains of duplicates. . .



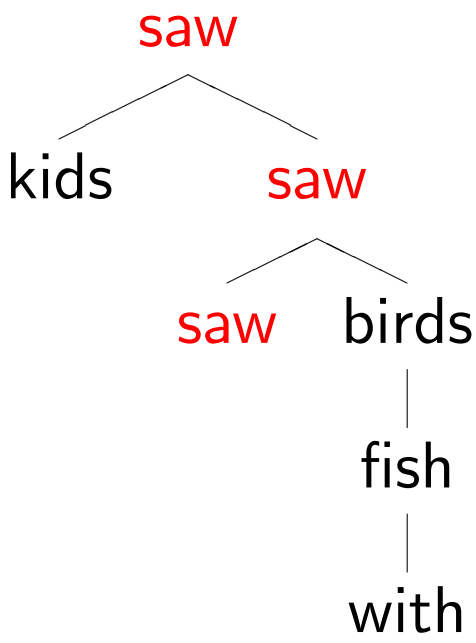
. . . and collapse chains of duplicates. . .



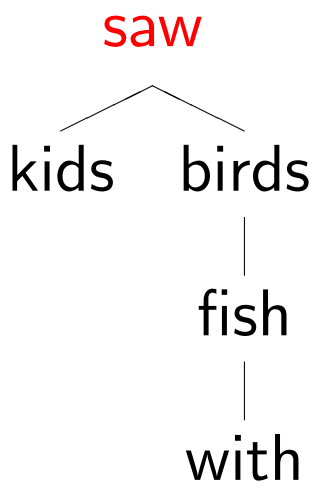
. . . and collapse chains of duplicates. . .



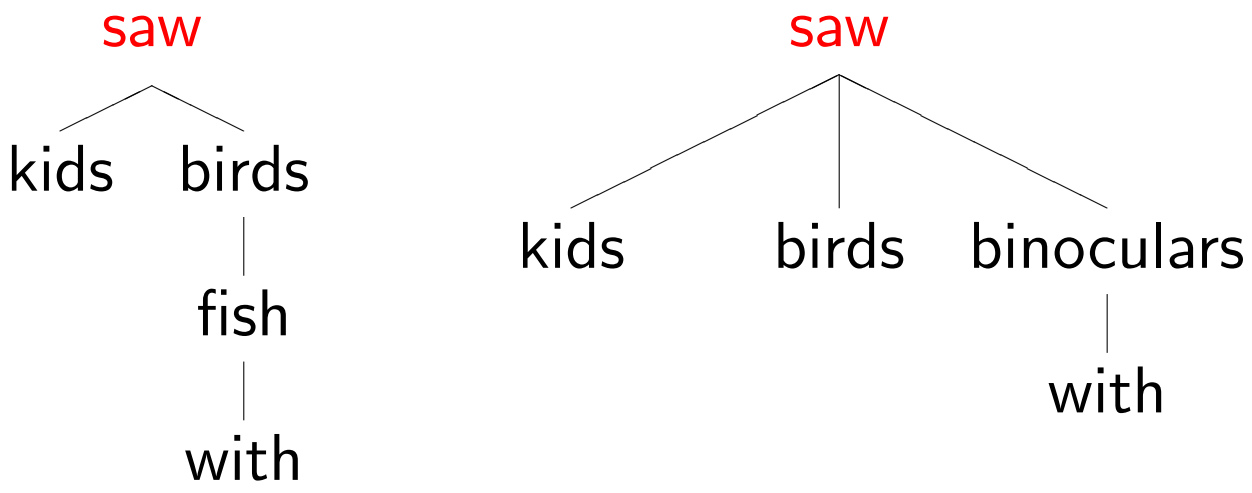
. . . and collapse chains of duplicates. . .



. . . and collapse chains of duplicates. . .



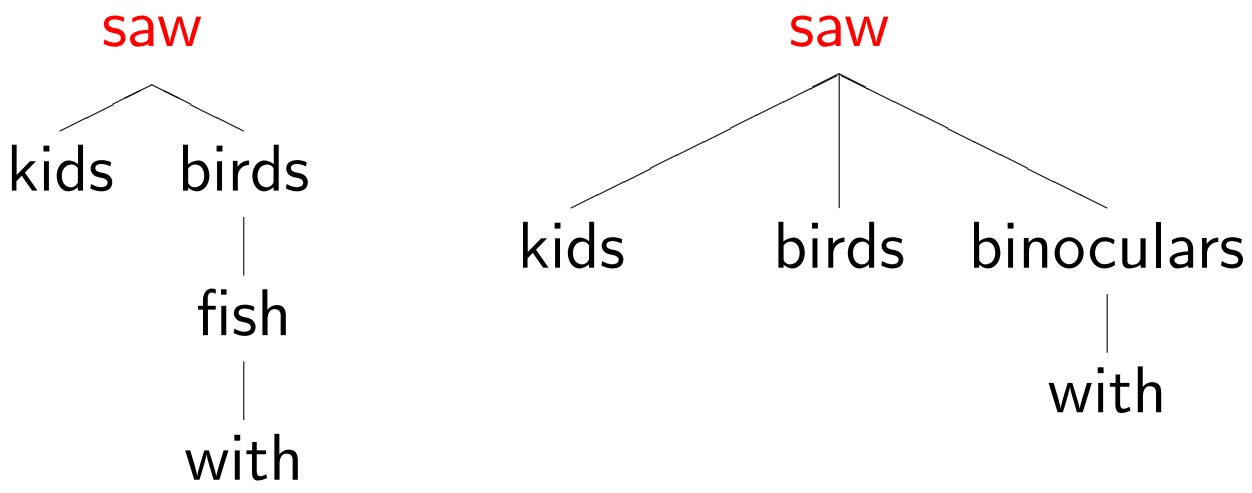
Dependency Parse



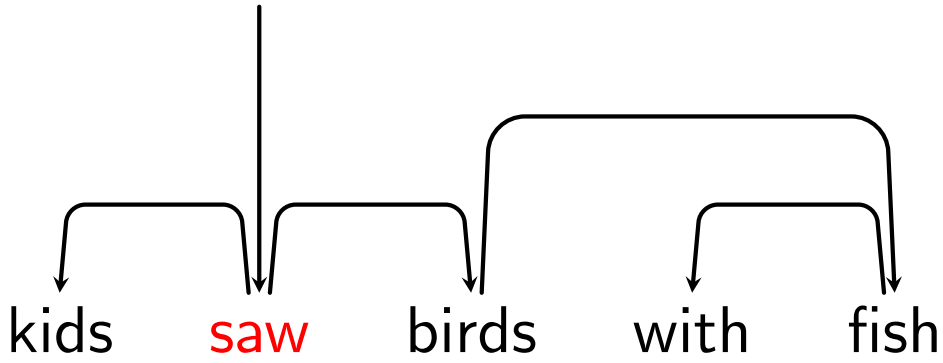
Linguists have long observed that the meanings of words within a sentence depend on one another, mostly in *asymmetric*, *binary* relations.

- Though some constructions don't cleanly fit this pattern: e.g., coordination, relative clauses.

Dependency Parse



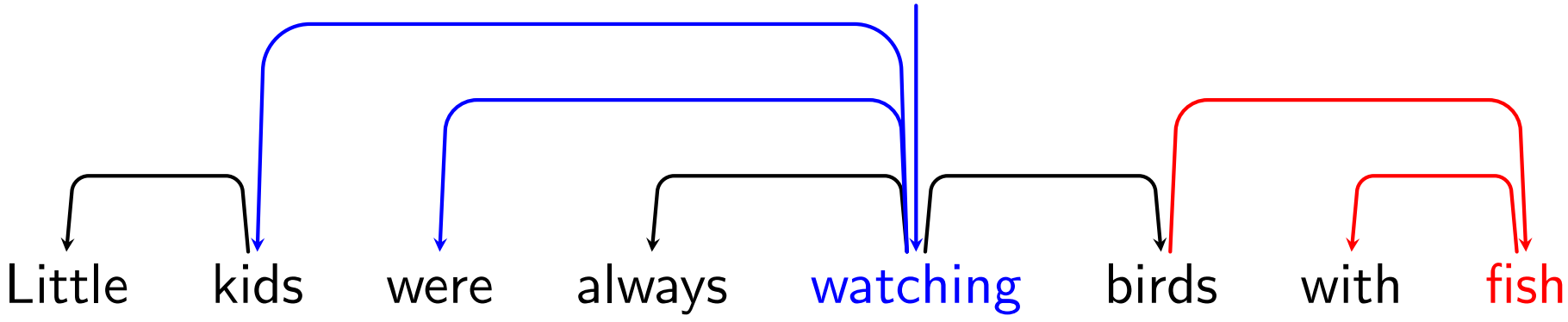
Equivalently, but showing word order (head → modifier):



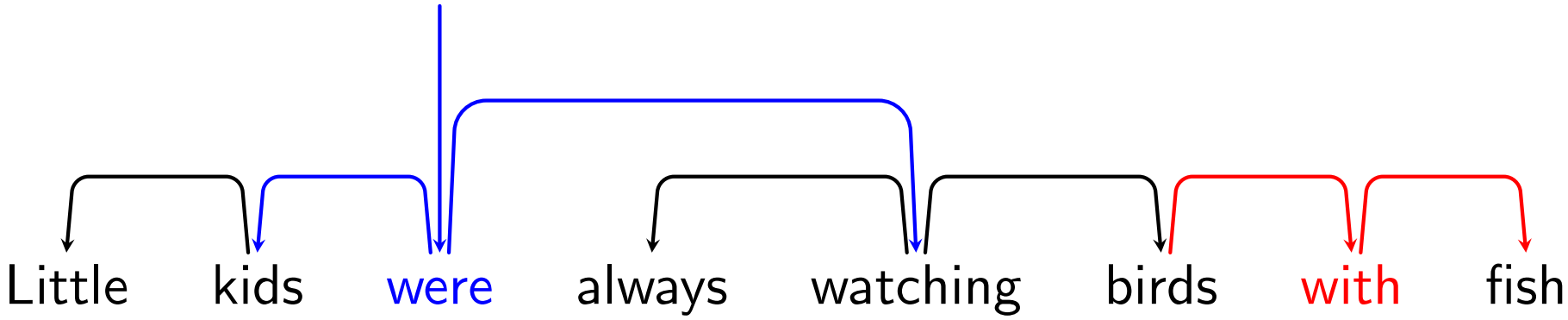
Because it is a tree, every word has exactly one parent.

Content vs. Functional Heads

Some treebanks prefer **content heads**:

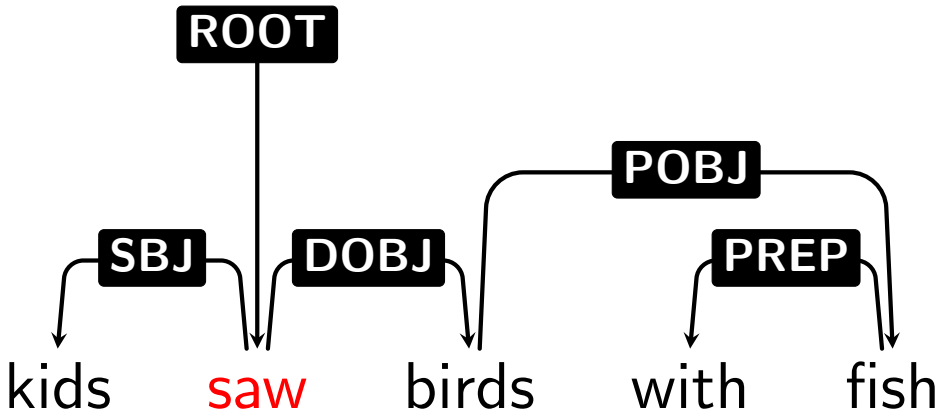


Others prefer **functional heads**:



Edge Labels

It is often useful to distinguish different kinds of head → modifier **relations**, by labeling edges:

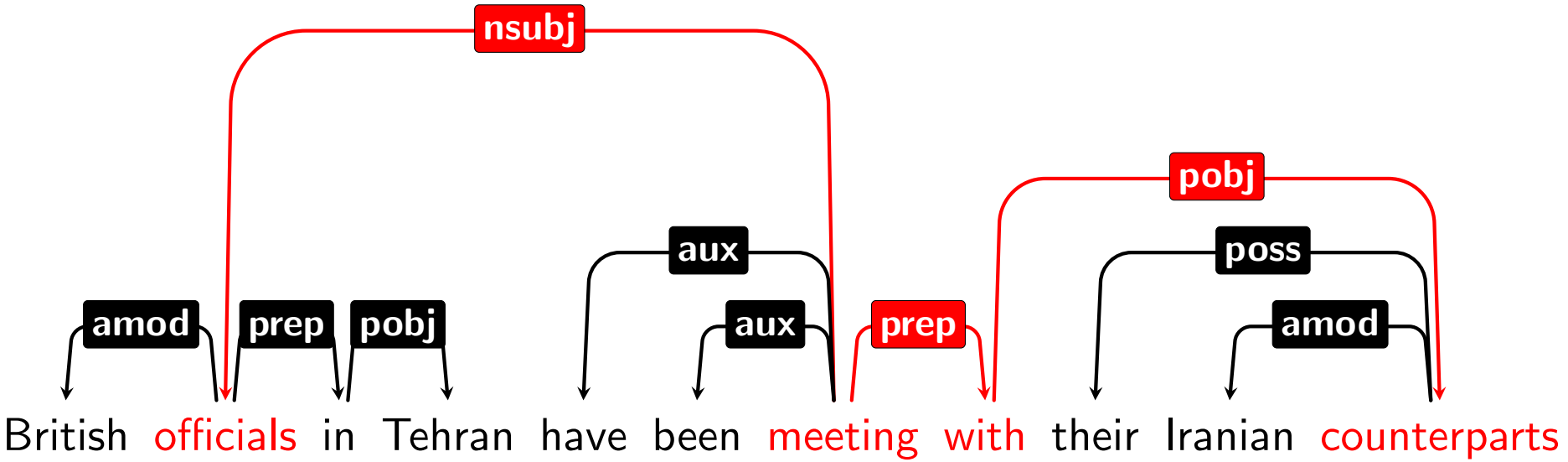


Important relations for English include subject, direct object, determiner, adjective modifier, adverbial modifier, etc. (Different treebanks use somewhat different label sets.)

- How would you identify the subject in a constituency parse?

Dependency Paths

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

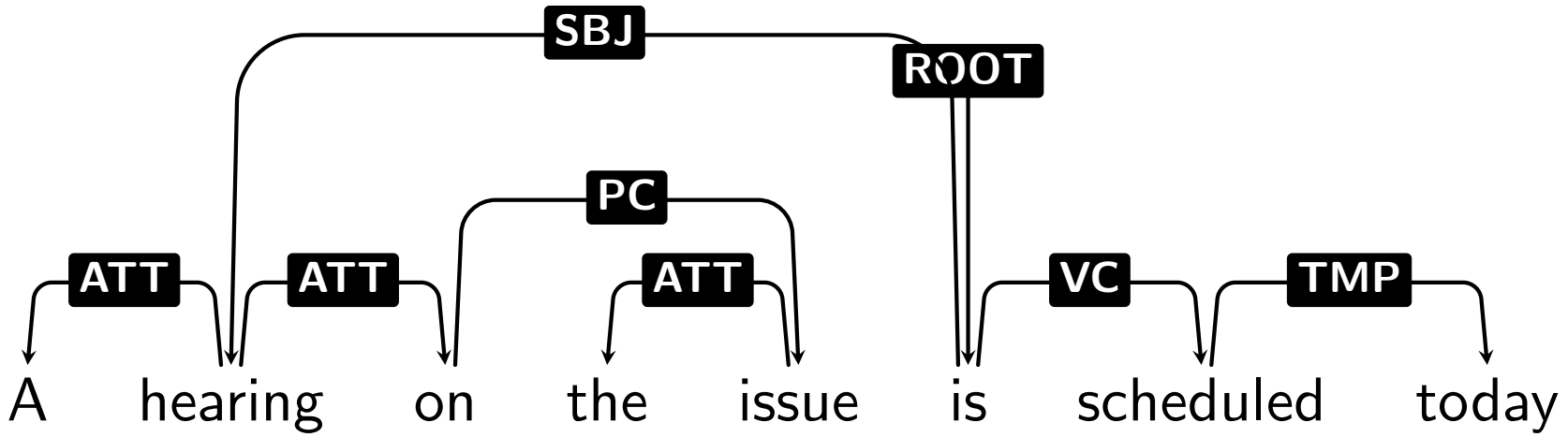


(example from Brendan O'Connor)

Projectivity

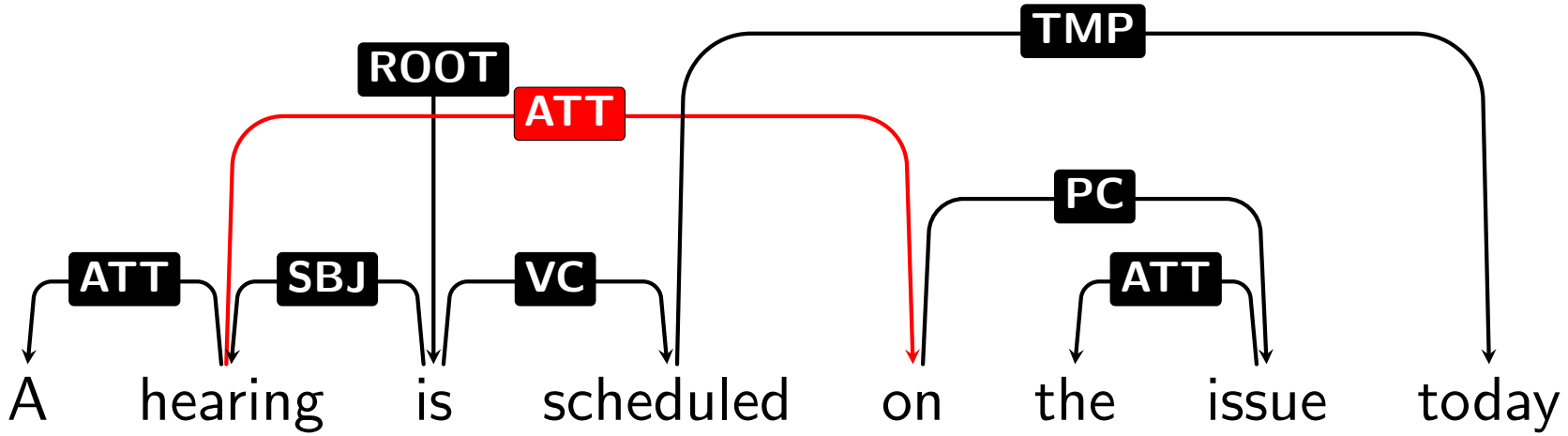
- A sentence's dependency parse is said to be **projective** if every subtree (node and all its descendants) occupies a *contiguous span* of the sentence.

- = The dependency parse can be drawn on top of the sentence without any crossing edges.



Nonprojectivity

- Other sentences are **nonprojective**:



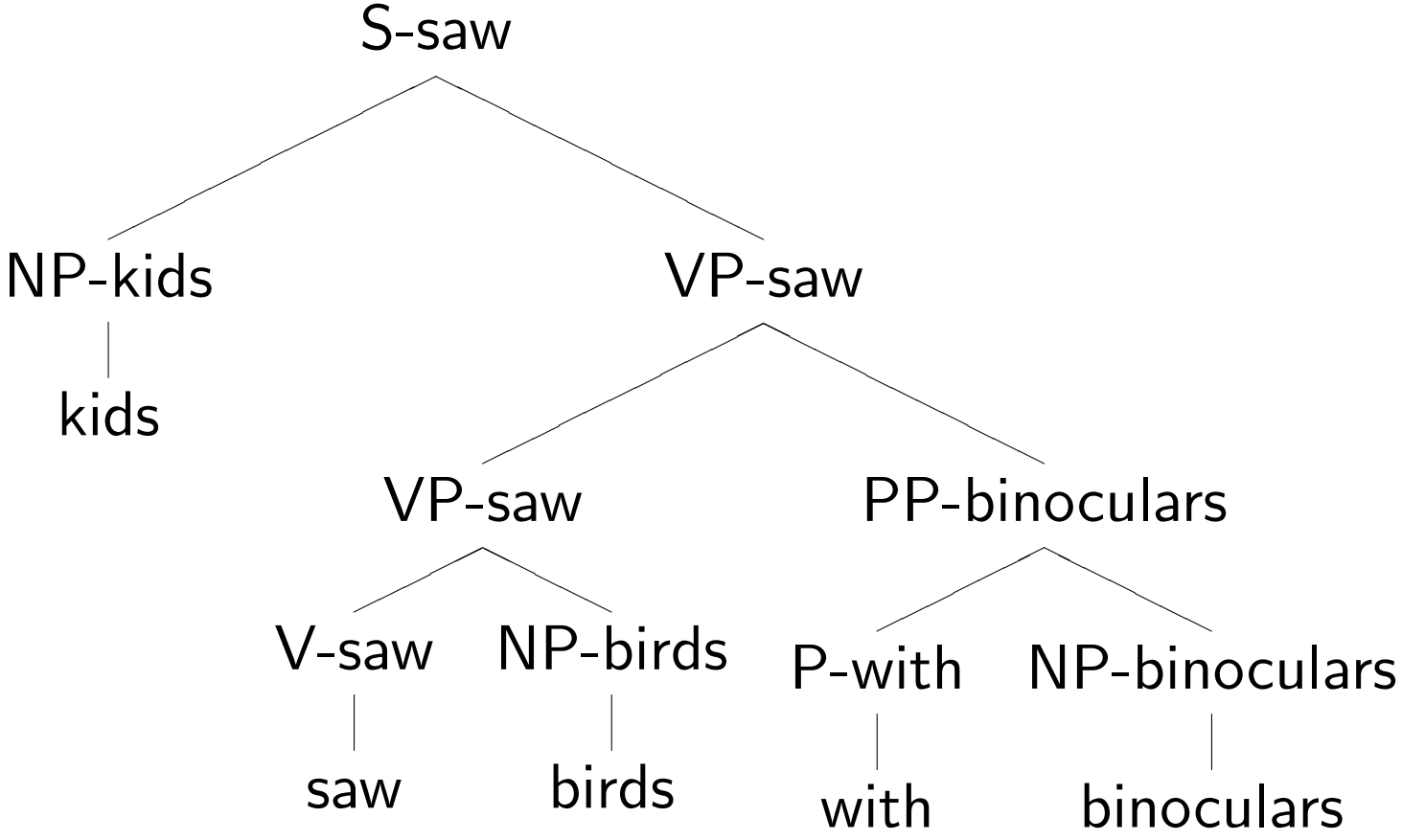
- Nonprojectivity is rare in English, but quite common in many languages.

Outline

1. Dependencies: what/why
2. **Transforming constituency → dependency parse**
3. Direct dependency parsing
   - Transition-based (shift-reduce)
   - Graph-based

Constituency Tree → Dependency Tree

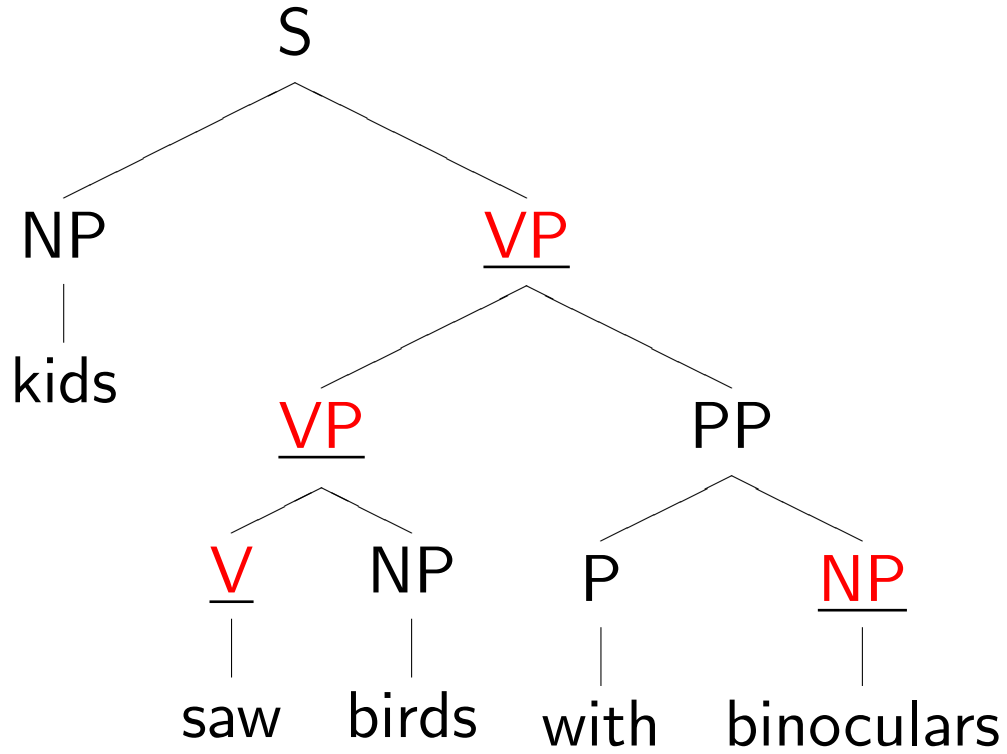
We saw how the **lexical head** of the phrase can be used to collapse down to a dependency tree:



- But how can we find each phrase's head in the first place?

Head Rules

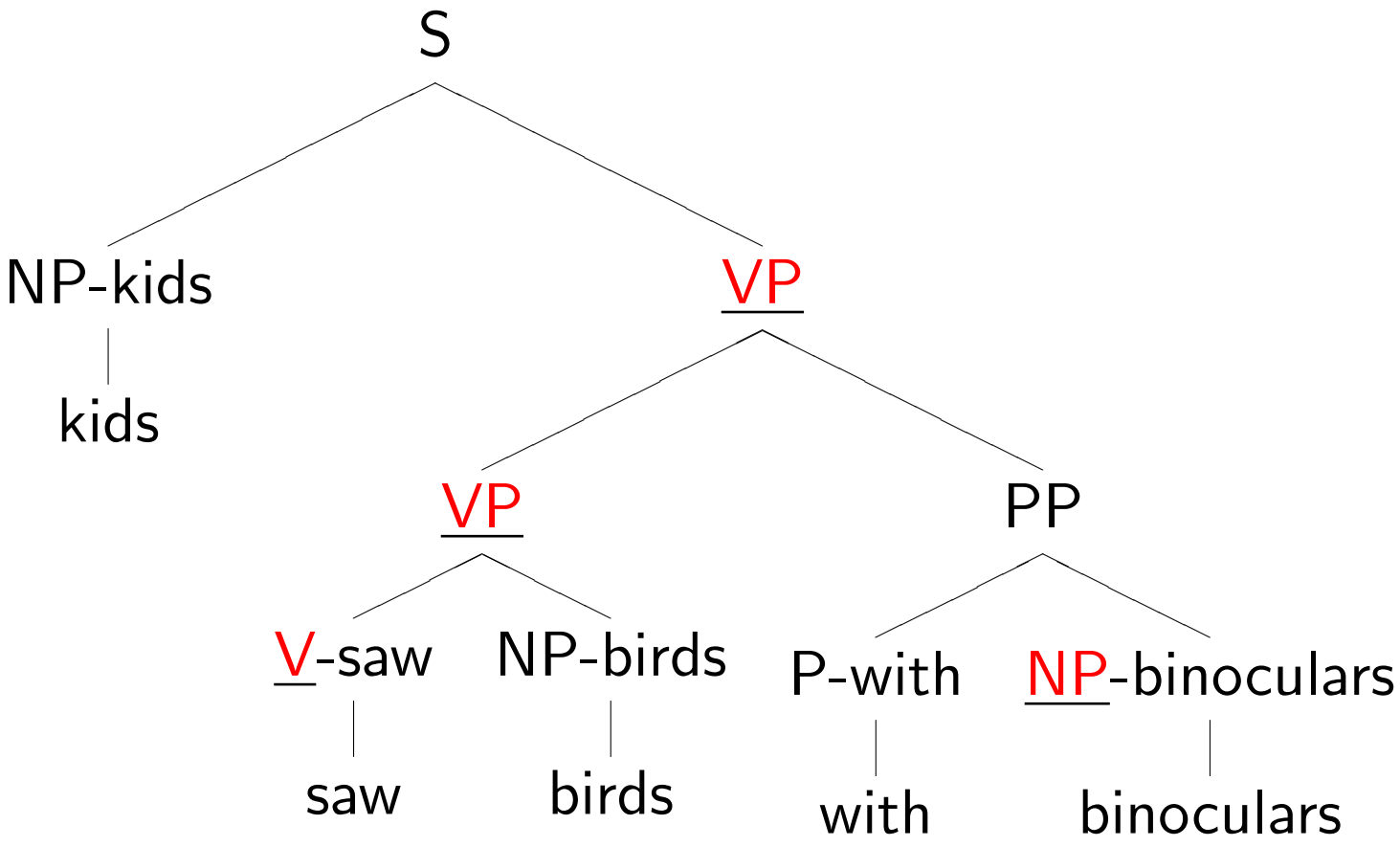
The standard solution is to use **head rules**: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. S → NP VP, VP → VP PP, PP → P NP (content head), etc.



- Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.

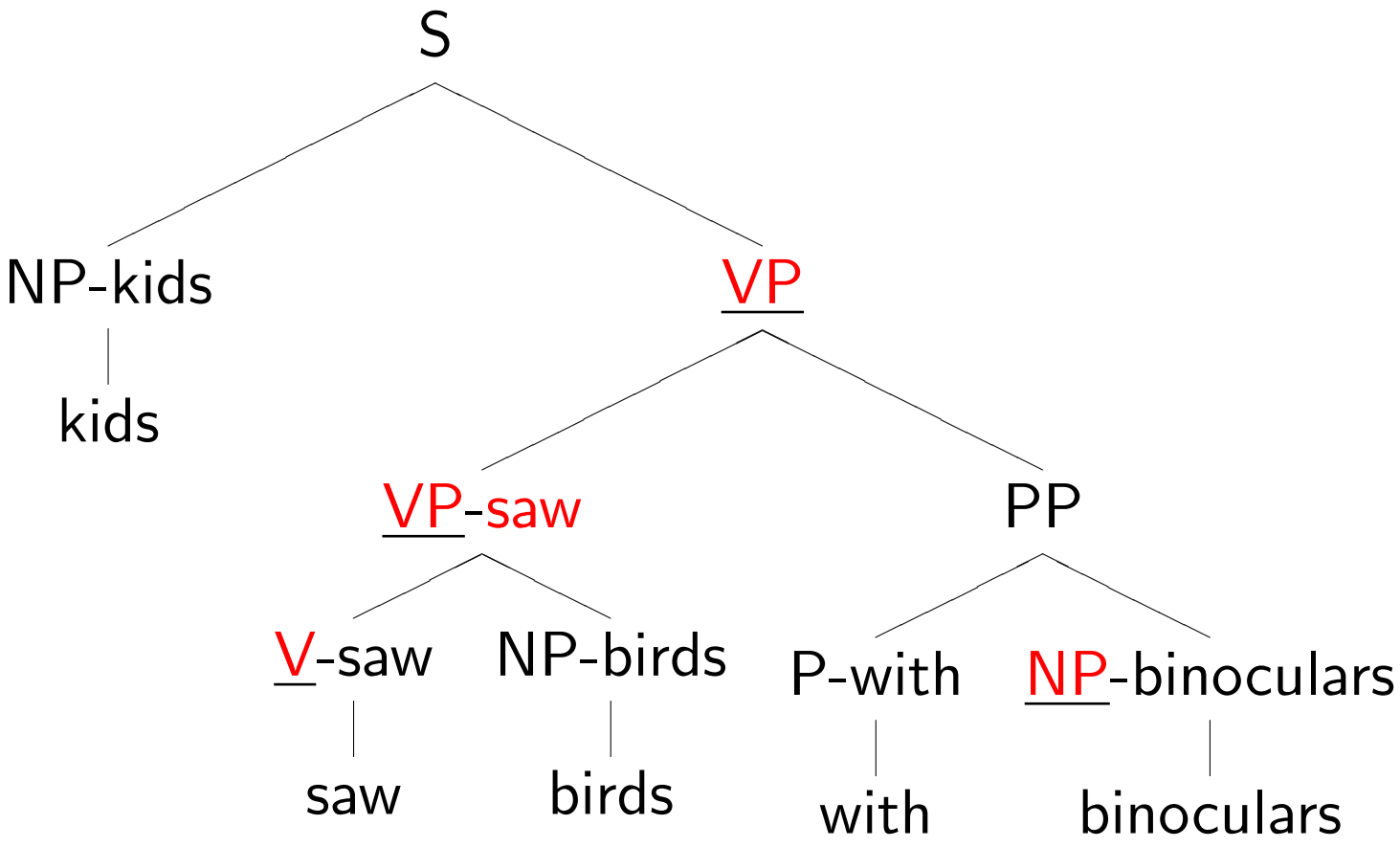
Head Rules

Then, propagate heads up the tree:



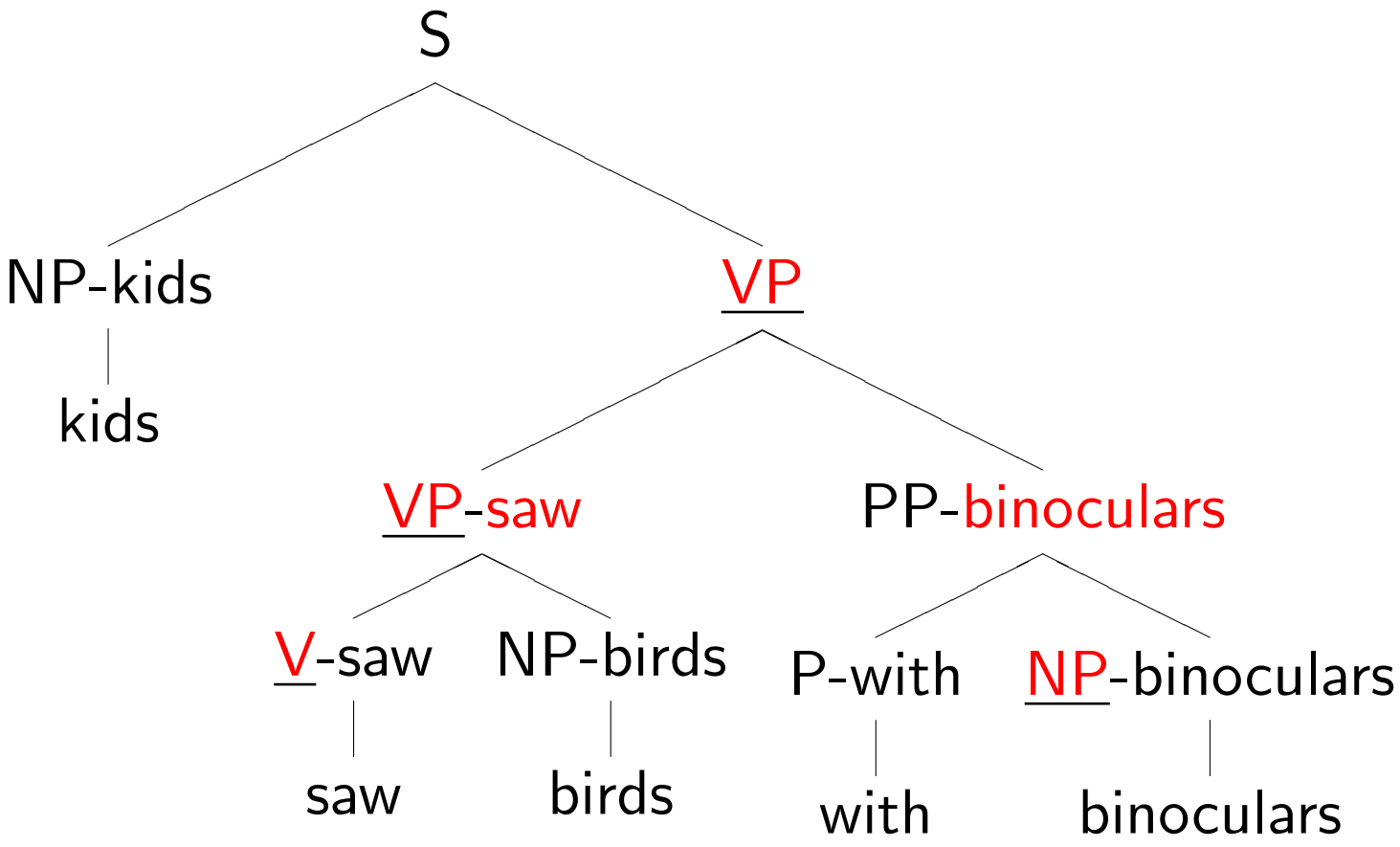
Head Rules

Then, propagate heads up the tree:



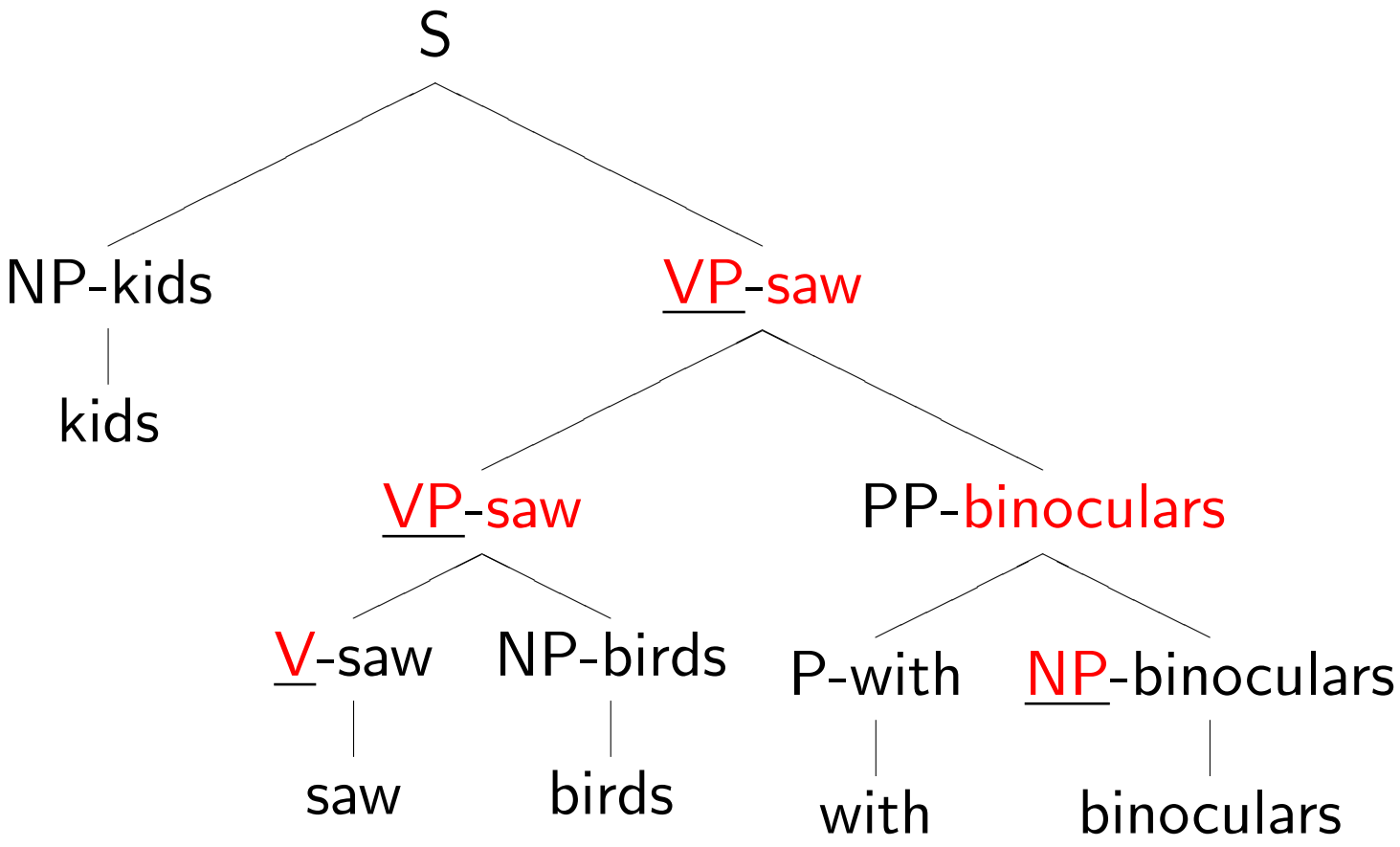
Head Rules

Then, propagate heads up the tree:



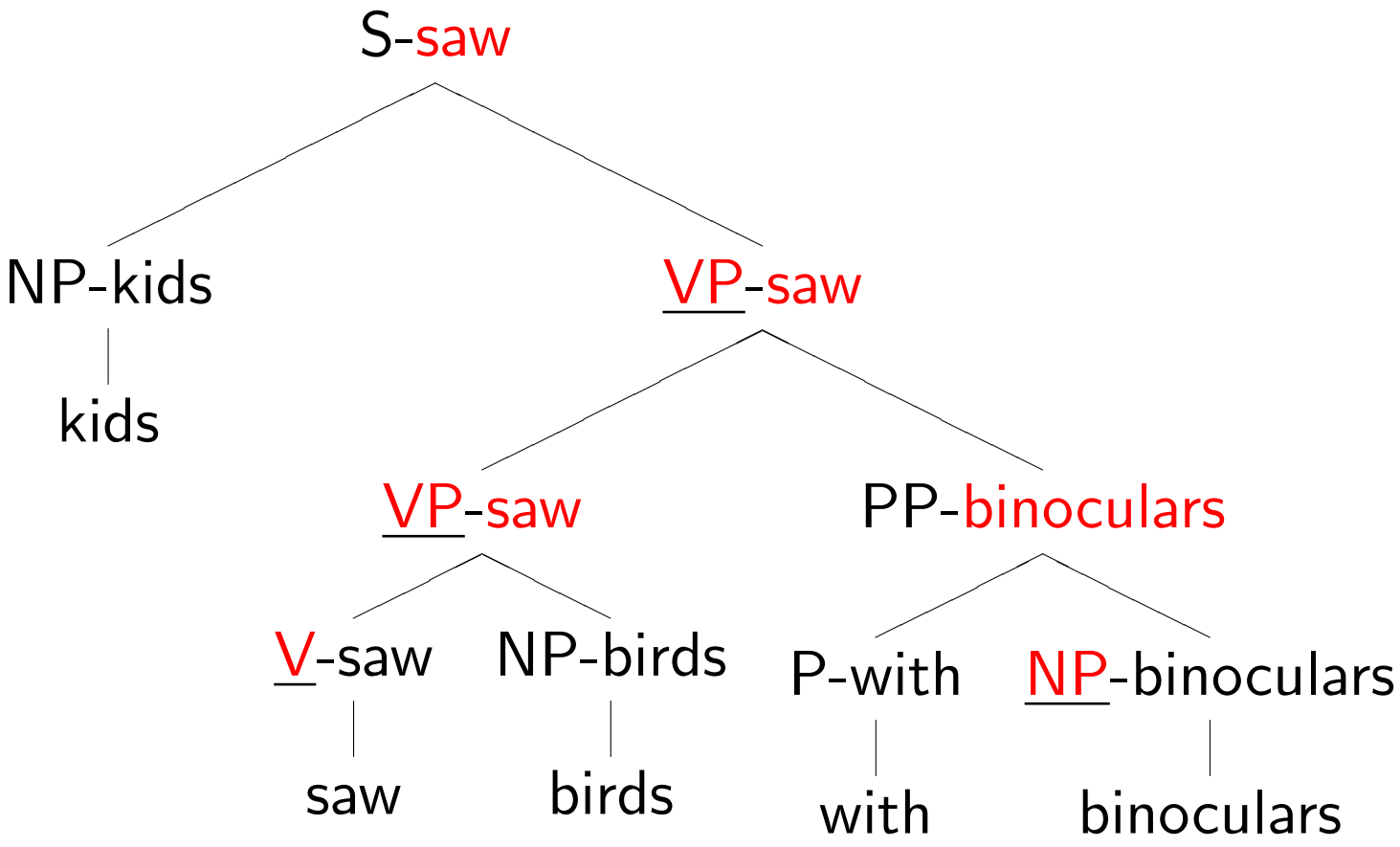
Head Rules

Then, propagate heads up the tree:



Head Rules

Then, propagate heads up the tree:



Outline

1. Dependencies: what/why
2. Transforming constituency $\rightarrow$ dependency parse
3. **Direct dependency parsing**
   - Transition-based (shift-reduce)
   - Graph-based

Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing.

- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it down to $O(Gn^3)$
  - N. Smith's slides explaining the Eisner algorithm: `http://courses.cs.washington.edu/courses/cse517/16wi/slides/an-dep-slides.pdf`
- **Shift-reduce**: more efficient, doesn't even require a grammar!

Transition-based Parsing

- Adapts shift-reduce methods: stack and buffer
- Remember: latent structure is just edges between words. Train a **classifier** to predict next action (SHIFT, REDUCE, ATTACH-LEFT, or ATTACH-RIGHT), and proceed left-to-right through the sentence. $O(n)$ time complexity!
- Only finds **projective** trees (without special extensions)
- Pioneering system: Nivre's MaltParser
- See `http://spark-public.s3.amazonaws.com/nlp/slides/Parsing-Dependency.pdf` (Jurafsky & Manning Coursera slides) for details and examples

Graph-based Parsing

- Global algorithm: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.

- **Edge-factored** models: Classifier assigns a nonnegative score to each possible edge; **maximum spanning tree** algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
  - Edge-factored assumption can be relaxed (higher-order models score larger units; more expensive).
  - Unlabeled parse $\rightarrow$ edge-labeling classifier (pipeline).

- Pioneering work: McDonald's MSTParser

- Can be formulated as constraint-satisfaction with **integer linear programming** (Martins's TurboParser)

Graph-based vs. Transition-based vs. Conversion-based

- TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only
- GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint
- CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., Stanford Parser). Slower than direct methods.

Choosing a Parser: Criteria

- Target representation: constituency or dependency?
- Efficiency? In practice, both runtime and memory use.
- Incrementality: parse the whole sentence at once, or obtain partial left-to-right analyses/expectations?
- Retrainable system?

Choosing a Parser: Performance

SOTA for English constituency parsing (WSJ §23): 91%–92% F_1

Parser	LR	LP	F1	#Toks/s.
Charniak (2000)	89.5	89.9	89.5	–
Klein and Manning (2003)	85.3	86.5	85.9	143
Petrov and Klein (2007)	90.0	90.3	90.1	169
Carreras et al. (2008)	90.7	91.4	91.1	–
Zhu et al. (2013)	90.3	90.6	90.4	1,290
Stanford Shift-Reduce (2014)	89.1	89.1	89.1	655
Hall et al. (2014)	88.4	88.8	88.6	12
This work	89.9	90.4	90.2	957
Charniak and Johnson (2005)*	91.2	91.8	91.5	84
Socher et al. (2013)*	89.1	89.7	89.4	70
Zhu et al. (2013)*	91.1	91.5	91.3	–

Table 3: Results on the English PTB §23. All systems reporting runtimes were run on the same machine. Marked as * are reranking and semi-supervised c-parsers.

(Fernández-González and Martins, 2015)

Choosing a Parser: Performance

Constituency parsing in other languages

Parser	Basque	French	German	Hebrew	Hungar.	Korean	Polish	Swedish	Avg.
Berkeley	70.50	**80.38**	78.30	86.96	81.62	71.42	79.23	79.19	78.45
Berkeley Tagged	74.74	79.76	78.28	85.42	85.22	78.56	86.75	80.64	81.17
Hall et al. (2014)	83.39	79.70	78.43	87.18	**88.25**	**80.18**	90.66	82.00	83.72
Crabbé and Seddah (2014)	85.35	79.68	77.15	86.19	87.51	79.35	**91.60**	82.72	83.69
This work	**85.90**	78.75	**78.66**	**88.97**	88.16	79.28	91.20	**82.80**	**84.22**
Björkelund et al. (2014)	88.24	82.53	81.66	89.80	91.72	83.81	90.50	85.50	86.72

(Fernández-González and Martins, 2015)

Choosing a Parser: Performance

SOTA for English dependency parsing (WSJ §23): 93%–94% UAS, 91%–92% LAS

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)†‡		93.67	92.68	0.4
Suzuki et al. (2009)†		93.79		
Koo et al. (2008)†		93.16		
Chen et al. (2014)†		93.77		
beam size				
training	decoding			
100	100	**93.28**	**92.35**	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. †: semi-supervised learning. ‡: 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. (No abstract phrase categories like NP.) Edges often labeled with relations like subject.

- Head rules govern how a lexicalized constituency grammar can be extracted from a treebank, and how a constituency parse can be coverted to a dependency parse.

- For English, it is often fastest and most convenient to parse directly to dependencies. Two main paradigms, graph-based and transition-based, with different kinds of models and search algorithms.