Dependency Grammars and Parser

LING 571 — Deep Processing for NLP

Shane Steinert-Threlkeld

Announcements

- HW2 ref code available
- HW3 due tonight
- HW4 now available

Ambiguity of the Week

Adam Macqueen
@adam_macqueen

Personally feel not enough hospitals are named after sandwiches.

Ambiguity of the Week 2

“What if my pet is not made of chicken and turkey?” —my brother

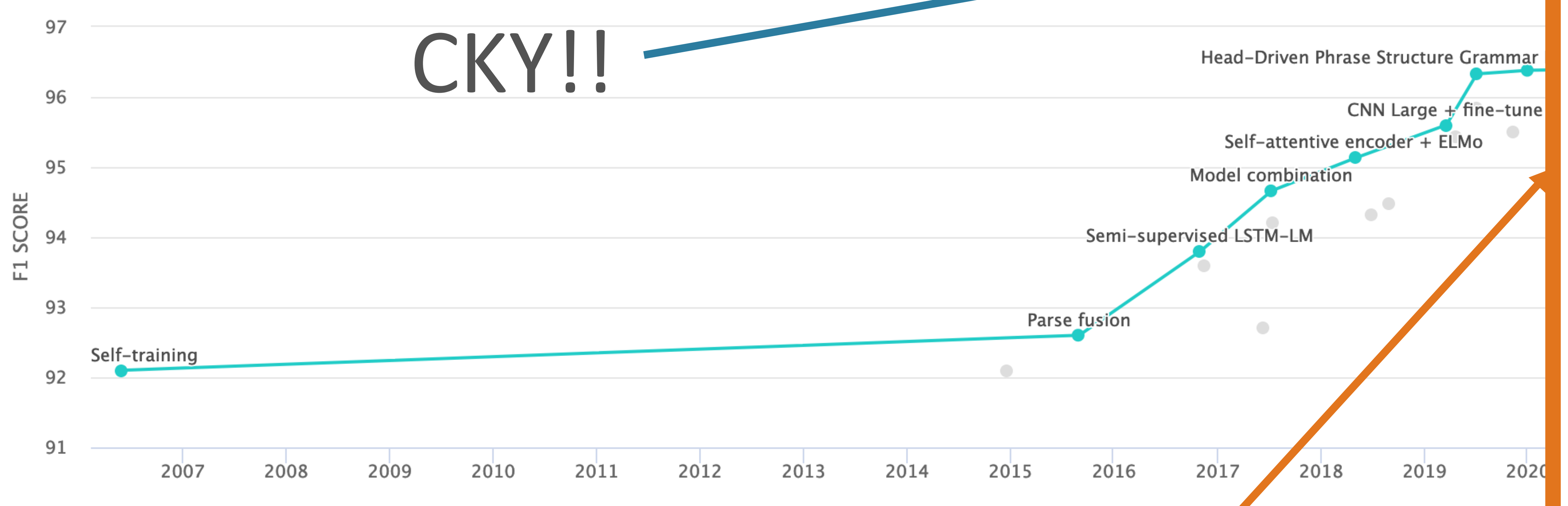
Parsing in the LLM era

Constituency Parsing on Penn Treebank

Leaderboard Dataset

View F1 score by Date for All models

CKY!!



Filter: LSTM XLNet BERT RoBERTa ELMo Transformer untagged

Rank	Model	F1 score	Extra Training Data	Paper	Code	Result	Year	Tags
1	SAPar + XLNet	96.40	×	Improving Constituency Parsing with Span Attention			2020	XLNet
2	Label Attention Layer + HPSG + XLNet	96.38	×	Rethinking Self-Attention: Towards Interpretability in Neural Parsing			2019	XLNet
3	Attach-Juxtapose Parser + XLNet	96.34	×	Strongly Incremental Constituency Parsing with Graph Neural Networks			2020	XLNet

https://paperswithcode.com/sota/constituency-parsing-on-penn-treebank

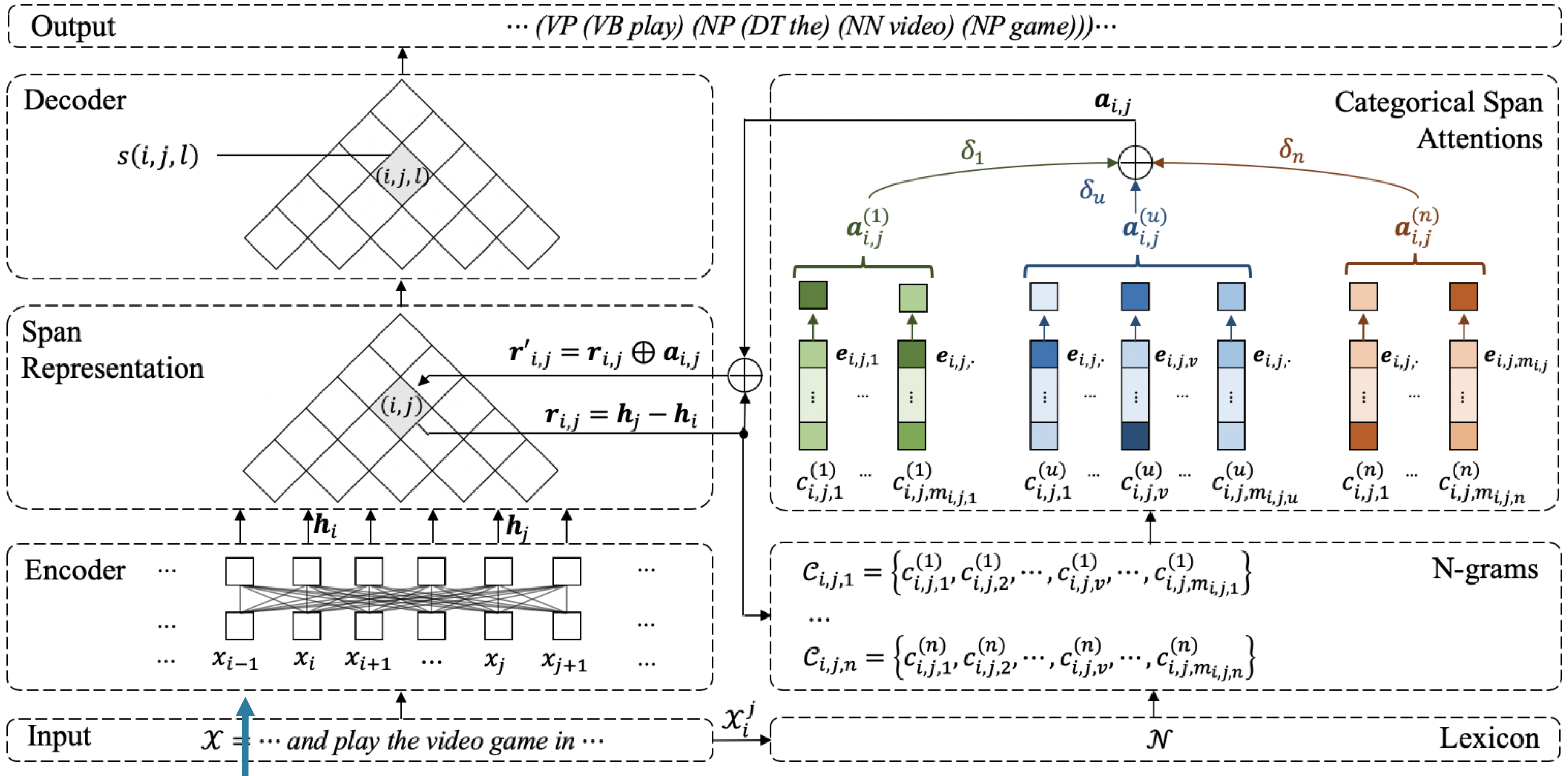


Figure 2: The architecture of the chart-based constituency parser with span attention, with an example partial input sentence and its output. The right part of the figure shows the categorical span attention, where extracted

https://aclanthology.org/2020.findings-emnlp.153/

Pre-trained LM

Roadmap

- Dependency Grammars
  - Definition
  - Motivation:
    - Limitations of Context-Free Grammars
- Dependency Parsing
  - By conversion to CFG
  - By Graph-based models
  - By transition-based parsing
- HW4 + mid-term feedback

Dependency Grammar

- [P]CFGs:
  - Phrase-Structure Grammars
  - Focus on modeling constituent structure
- Dependency grammars:
  - Syntactic structure described in terms of
    - Words
    - Syntactic/semantic relations between words

Dependency Parse

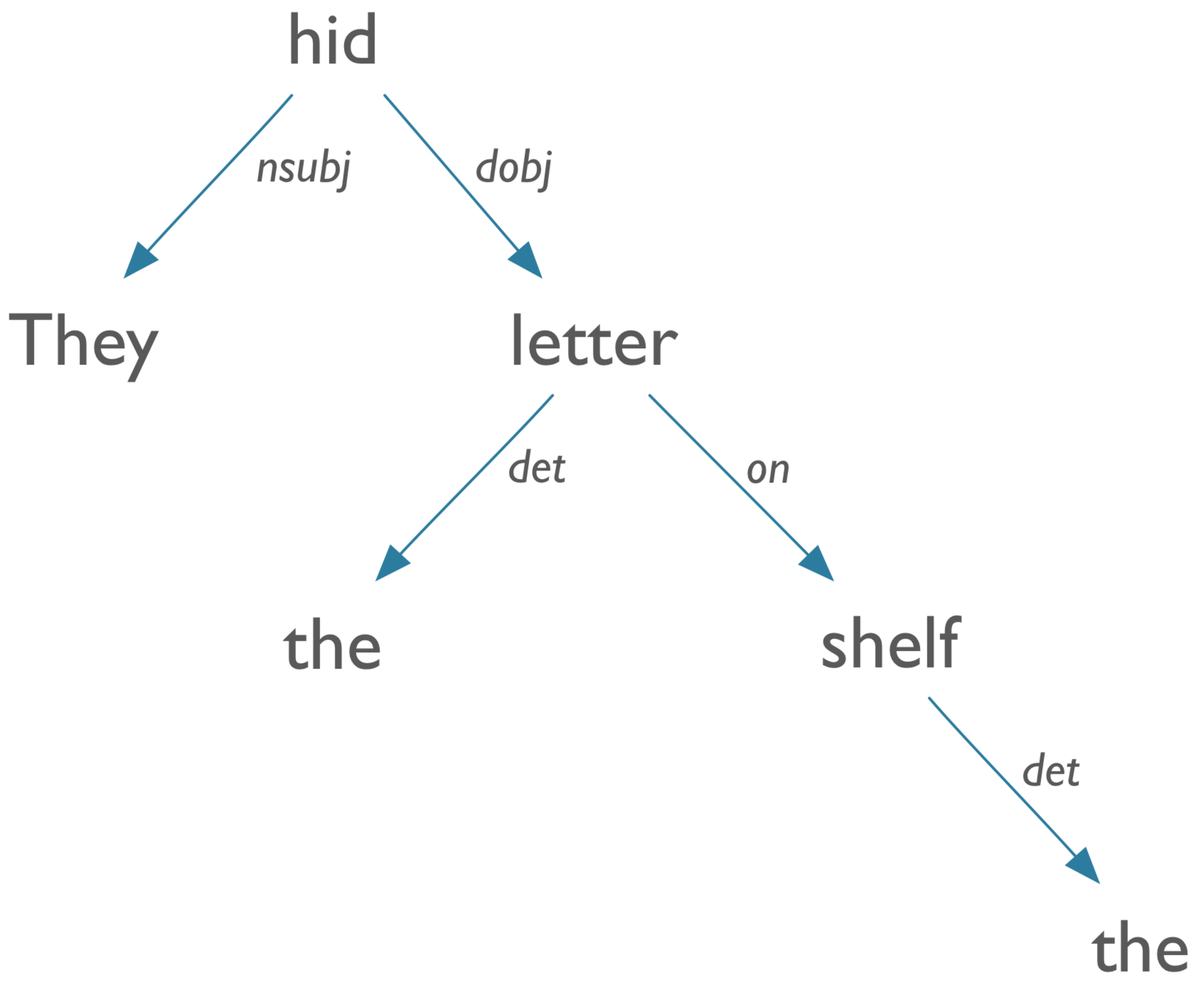
- A Dependency parse is a tree,* where:
  - Nodes correspond to words in string
  - Edges between nodes represent dependency relations
    - Relations may or may not be labeled (aka typed)
  - *: in very special cases, can argue for cycles

Dependency Parse Example:

They hid the letter on the shelf

Argument Dependencies	
Abbreviation	**Description**
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition

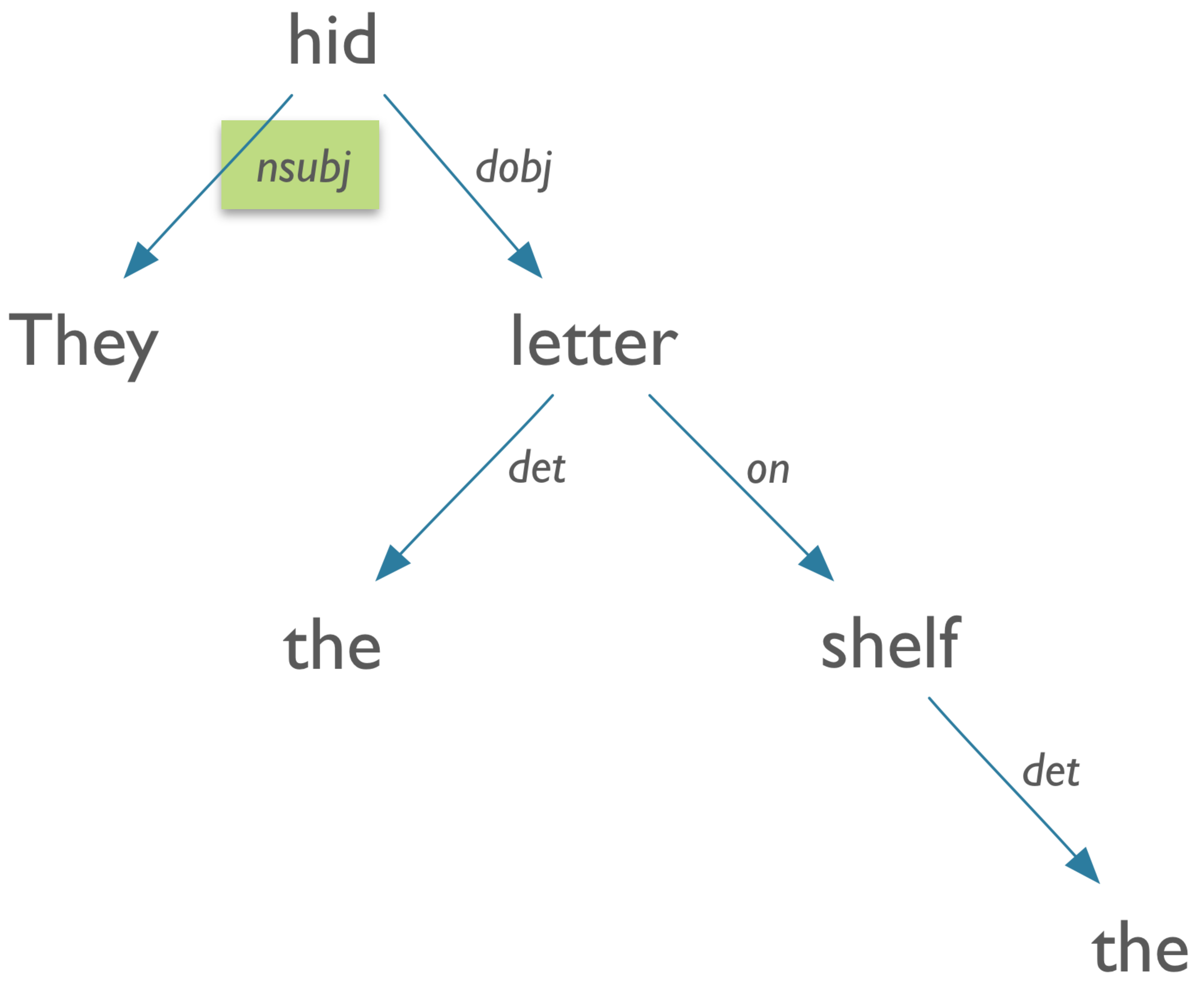
Modifier Dependencies	
Abbreviation	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier



Dependency Parse Example:

They hid the letter on the shelf

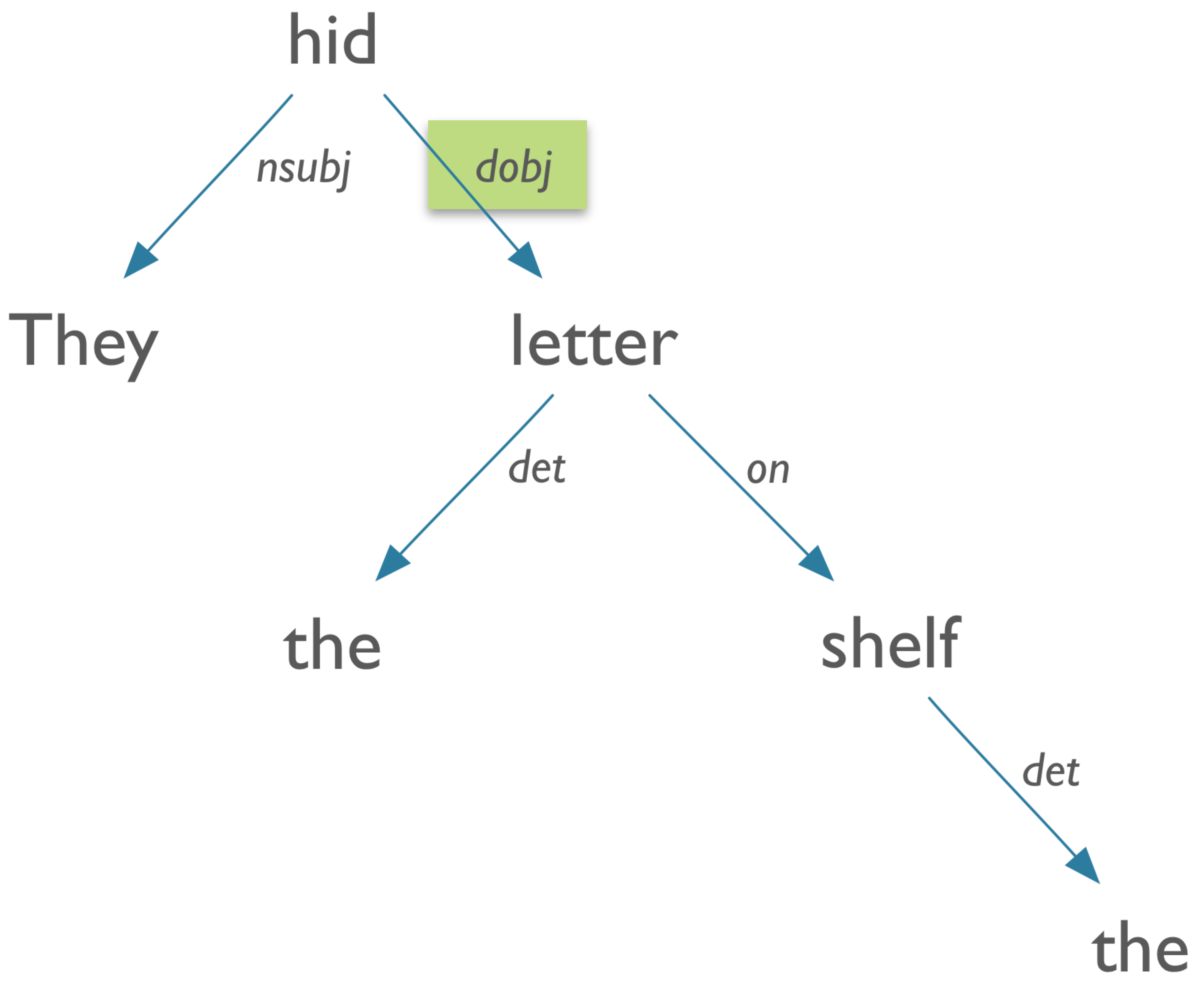
Argument Dependencies	
Abbreviation	**Description**
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	
Abbreviation	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier



Dependency Parse Example:

They hid the letter on the shelf

Argument Dependencies	
Abbreviation	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	
Abbreviation	Description
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

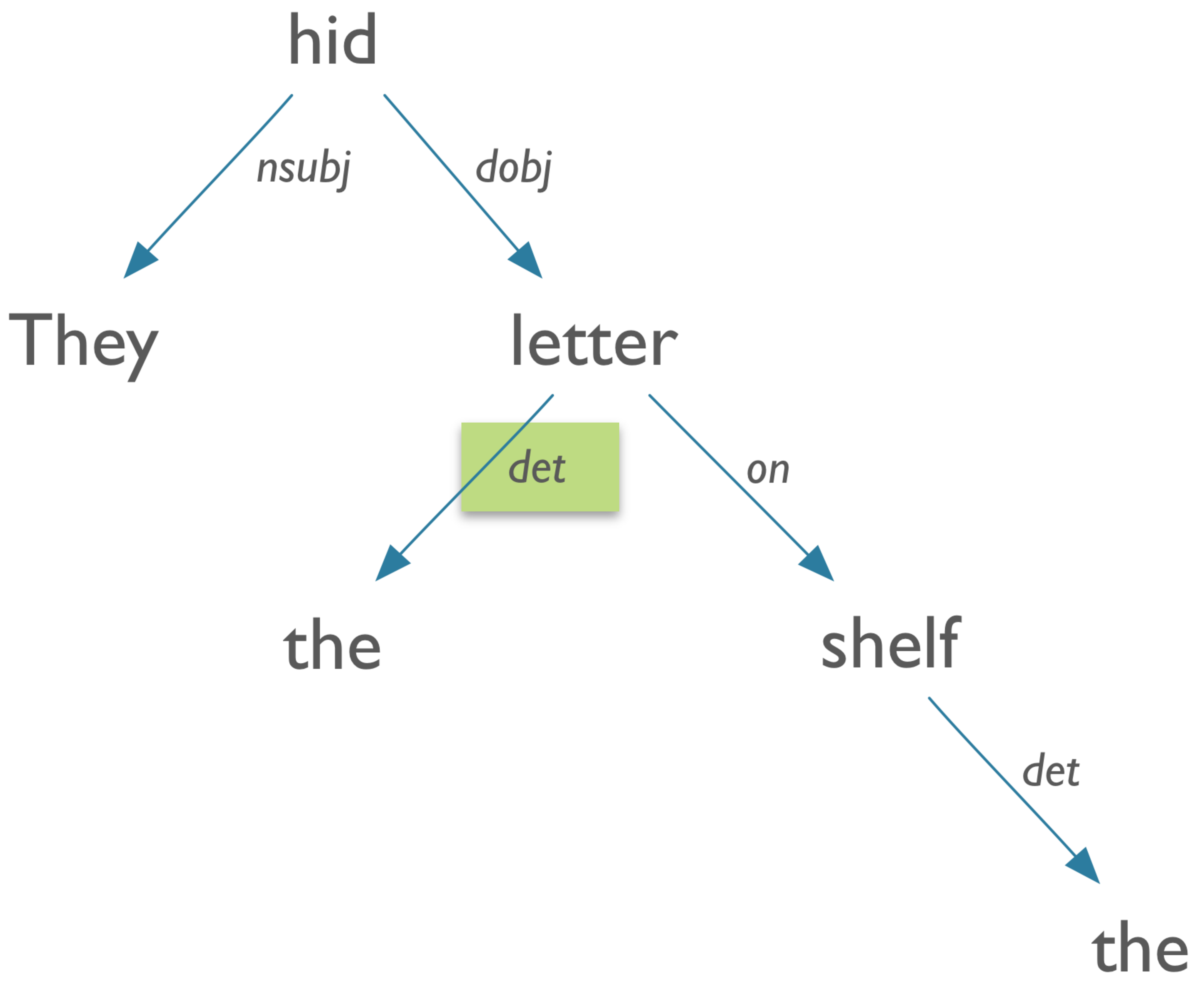


Dependency Parse Example:

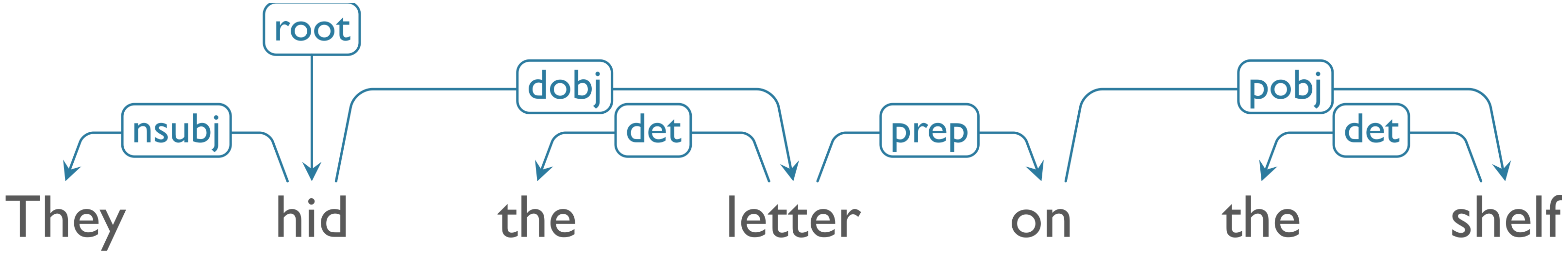
They hid the letter on the shelf

Argument Dependencies	
Abbreviation	**Description**
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition

Modifier Dependencies	
Abbreviation	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier



Alternative Representation

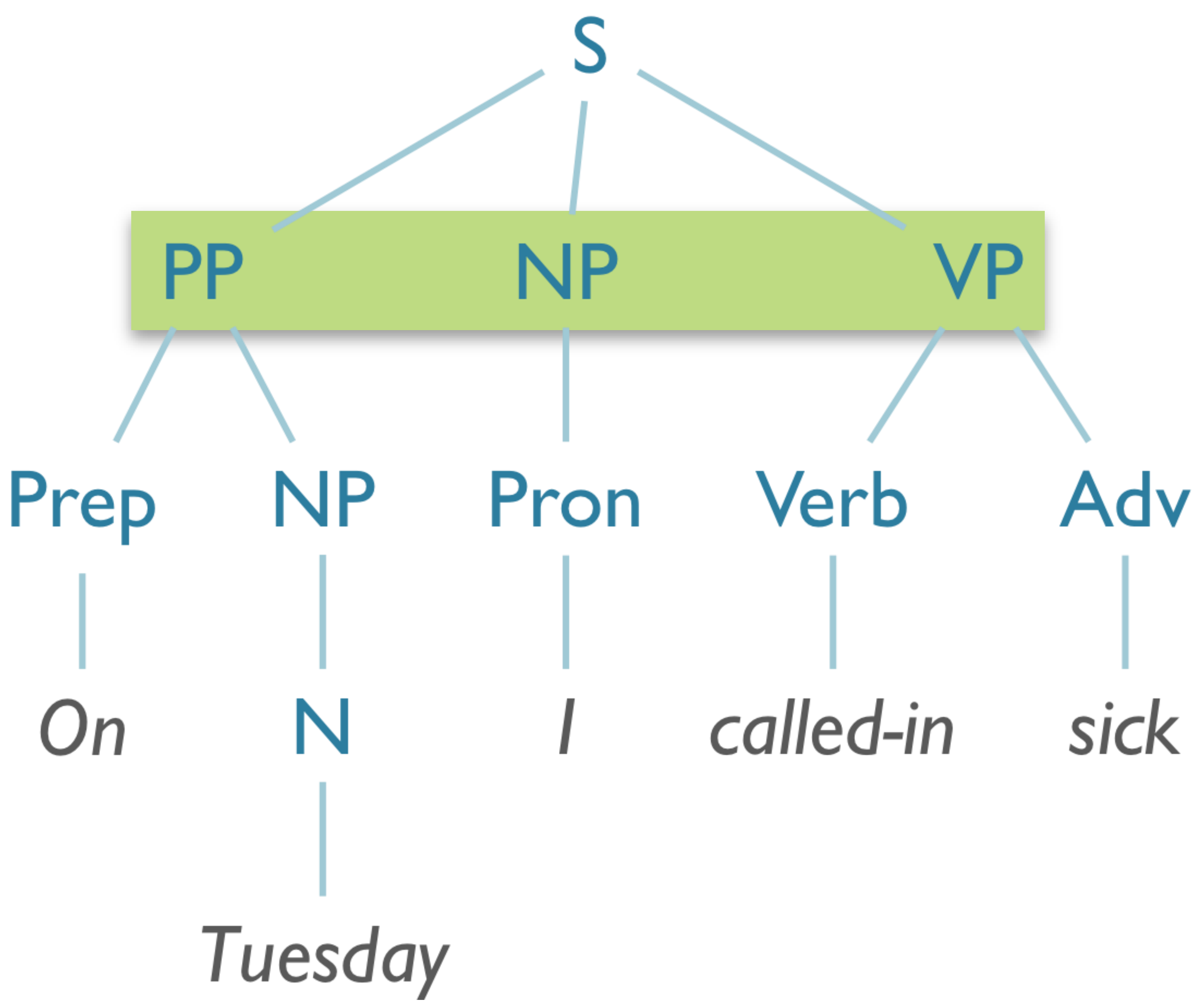


Why Dependency Grammar?

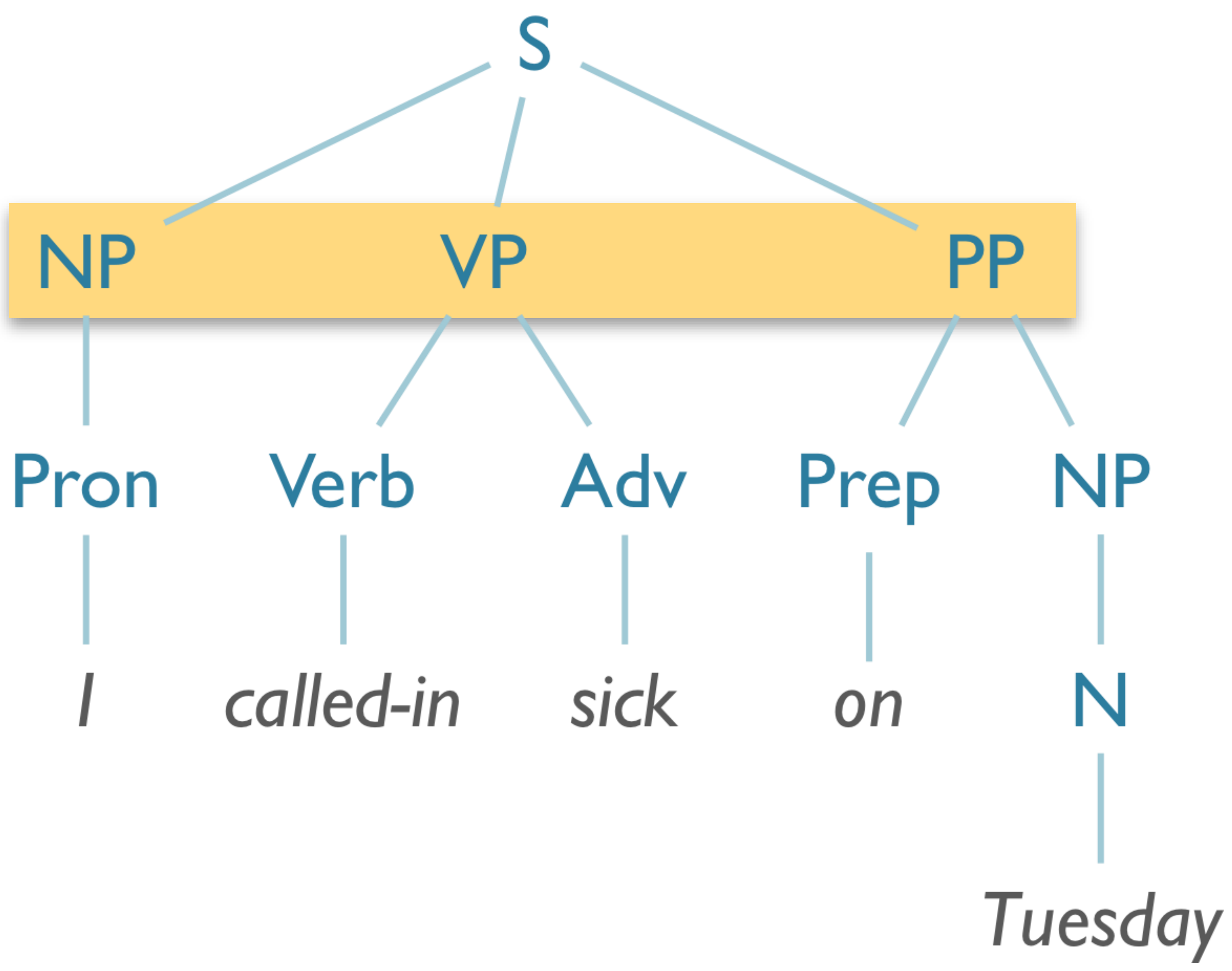
- More natural representation for many tasks
  - Clear encapsulation of predicate-argument structure
    - Phrase structure may obscure, e.g. *wh-movement*
- Good match for question-answering, relation extraction
  - *Who* did *what* to *whom*?
  - = (*Subject*) did (*theme*) to (*patient*)
  - Helps with parallel relations between roles in questions, and roles in answers

Why Dependency Grammar?

- Easier handling of flexible or free word order
- How does CFG handle variation in word order?



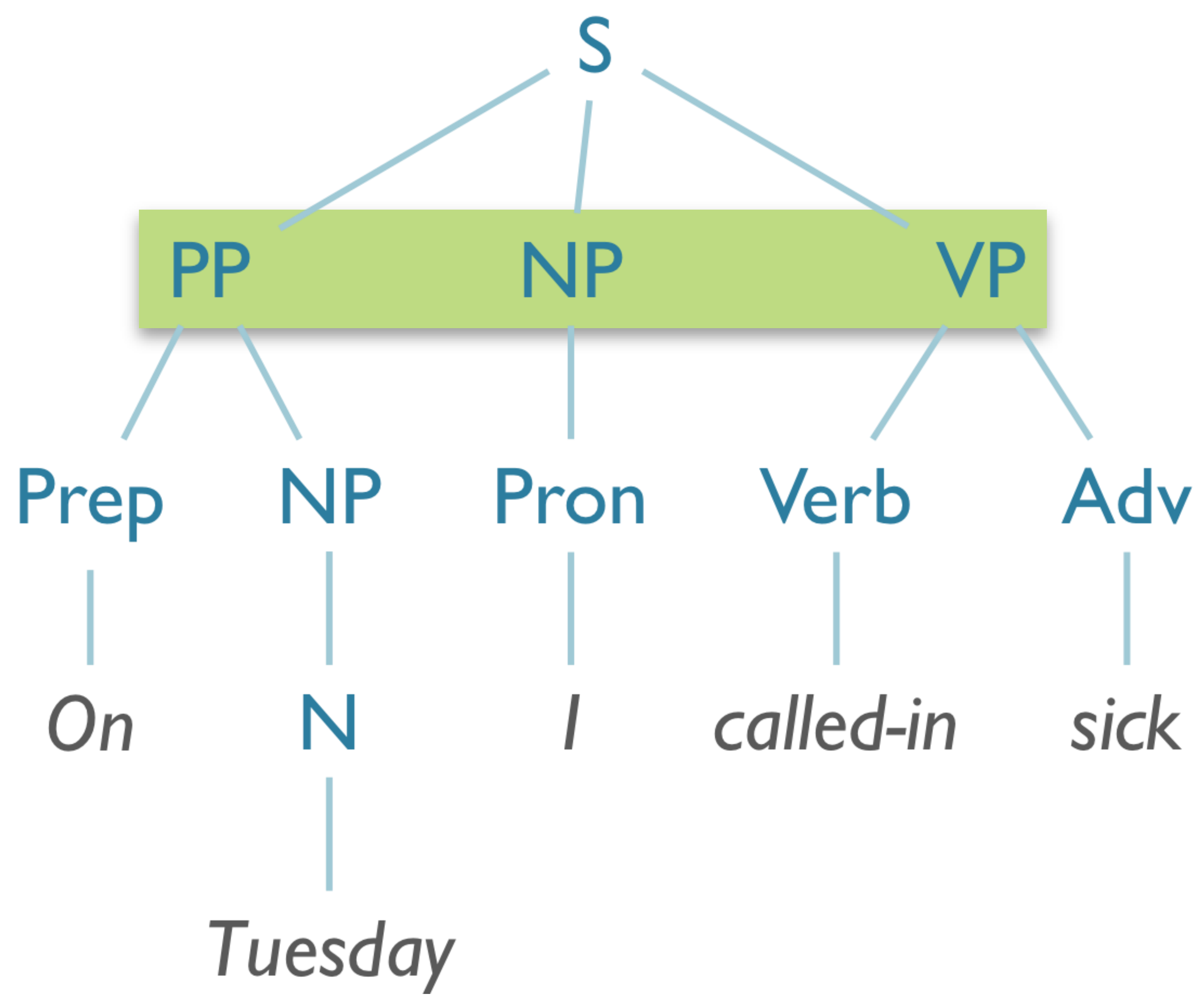
$S \rightarrow PP\ NP\ VP$



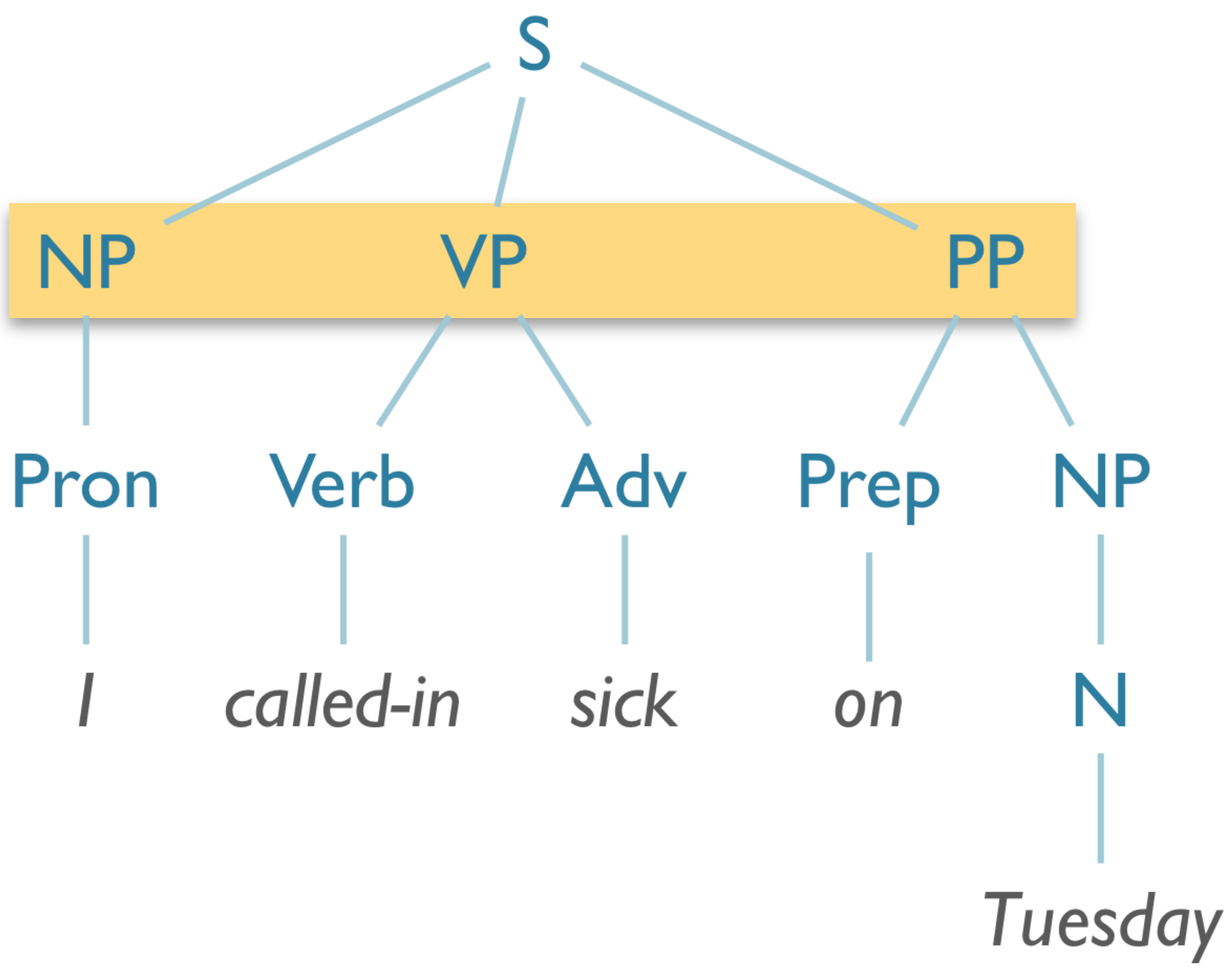
$S \rightarrow NP\ VP\ PP$

Why Dependency Grammar?

- English has relatively fixed word order
- Big problem for languages with freer word order



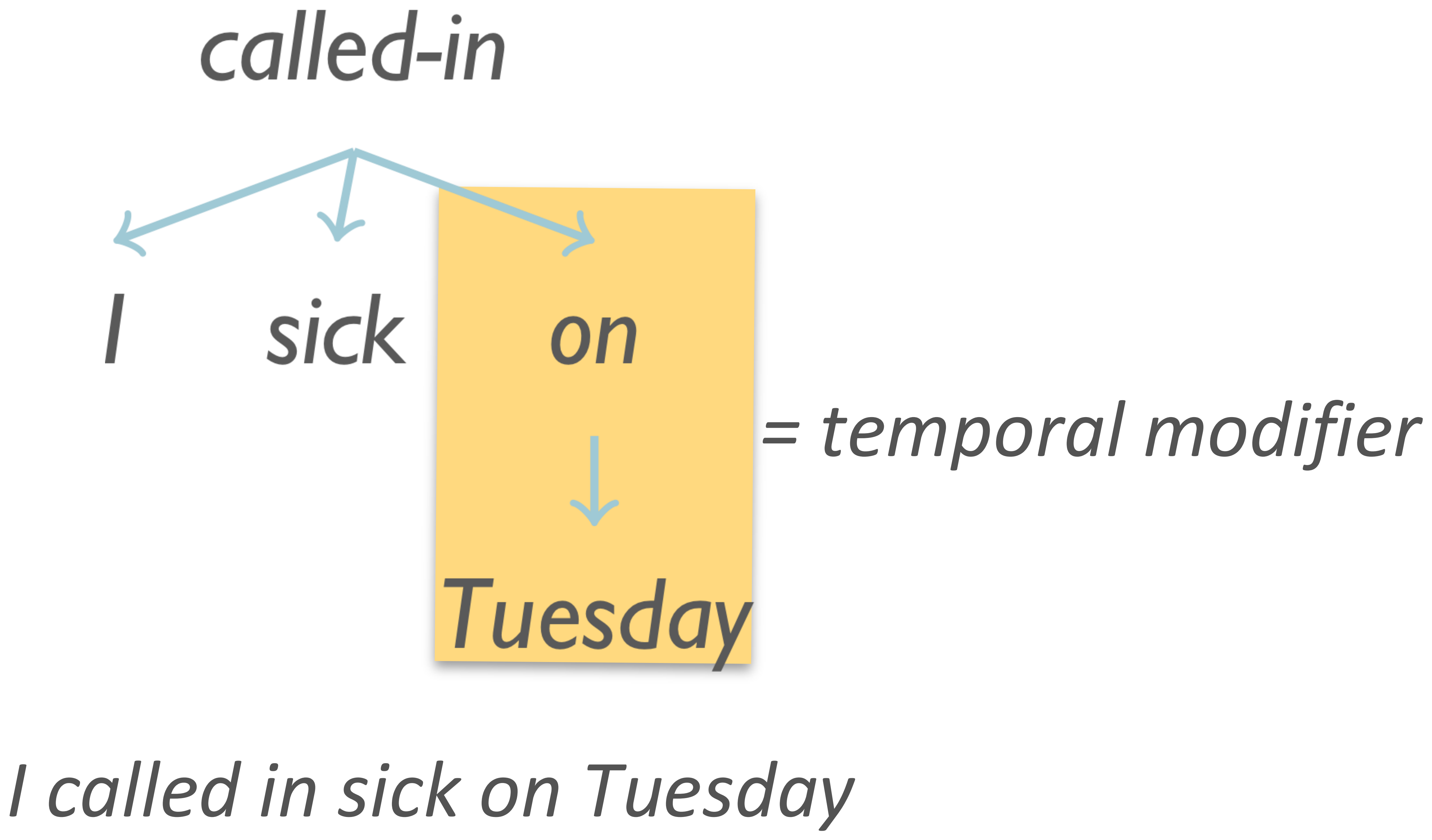
$S \rightarrow PP\ NP\ VP$



$S \rightarrow NP\ VP\ PP$

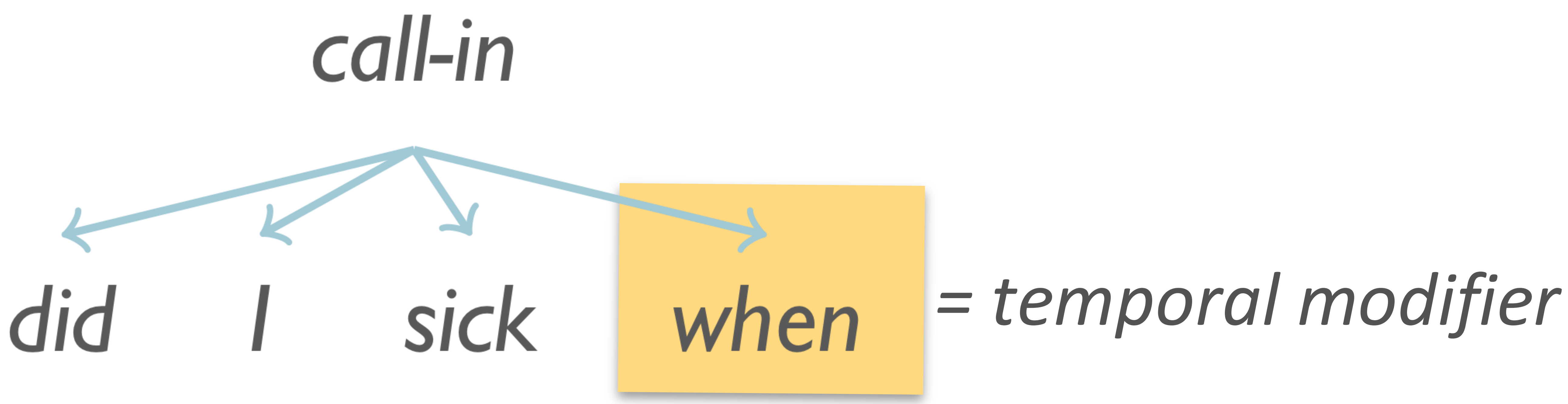
Why Dependency Grammar?

- How do dependency structures represent the difference?
  - Same structure
  - Relationships are between words, order insensitive



Why Dependency Grammar?

- How do dependency structures represent the difference?
  - Same structure
  - Relationships are between words, order insensitive



when did I call in sick?

Natural Efficiencies

- Phrase Structures:
  - Must derive full trees of many non-terminals
- Dependency Structures:
  - For each word, identify
    - Syntactic head, ***h***
    - Dependency label, ***d***
  - Inherently lexicalized
    - Strong constraints hold between pairs of words

Visualization

- Web demos:
  - displaCy: https://explosion.ai/demos/displacy
  - Stanford CoreNLP: http://corenlp.run/
- spaCy and stanza Python packages have good built-in parsers
- LaTeX: tikz-dependency (https://ctan.org/pkg/tikz-dependency)

Resources

- Universal Dependencies:
  - Consistent annotation scheme (i.e. same POS, dependency labels)
  - Treebanks for >150 languages
    - Sizes: German, Czech, Japanese, Russian, French, Arabic, …

Resources

- Universal Dependencies:
  - Consistent annotation sch
  - Treebanks for >150 langu
    - Sizes: German, Czech, J

Possible Future Extensions

People have expressed interest in providing annotated data for the following languages but no valid data has been provided so far.

Akkadian	1	117K	Afro-Asiatic, Semitic
Amharic	2	-	Afro-Asiatic, Semitic
Archaic Irish	1	-	IE, Celtic
Assamese	1	-	IE, Indic
Bengali	3	-	IE, Indic
Bhojpuri	1	-	IE, Indic
Cappadocian	1	-	IE, Greek
Classical Nahuatl	1	-	Uto-Aztecan
Cuicatec	1	-	Oto-Manguean
Cusco Quechua	1	-	Quechuan
Czech	1	1,191K	IE, Slavic
Danish	1	-	IE, Germanic
Dargwa	1	-	Nakh-Daghestanian, Lak-Dargwa
English	1	-	IE, Germanic
French	1	-	IE, Romance
Frisian	1	-	IE, Germanic
Gedeo	1	-	Afro-Asiatic, Cushitic
Georgian	1	-	Kartvelian
Greek	3	-	IE, Greek
Gwichin	1	-	Na-Dene
Hebrew	1	-	Afro-Asiatic, Semitic
Hiligaynon	1	<1K	Austronesian, Central Philippine
Hindi	1	4K	IE, Indic
Huave	1	-	Huavean
Italian	1	-	IE, Romance
Japanese	2	-	Japanese
Kabyle	1	23K	Afro-Asiatic, Berber

Summary

- Dependency grammars balance complexity and expressiveness
  - Sufficiently expressive to capture predicate-argument structure
  - Sufficiently constrained to allow efficient parsing
- Still not perfect
  - “On Tuesday I called in sick” vs. “I called in sick on Tuesday”
  - These feel pragmatically different (e.g. topically), might want to represent difference syntactically.

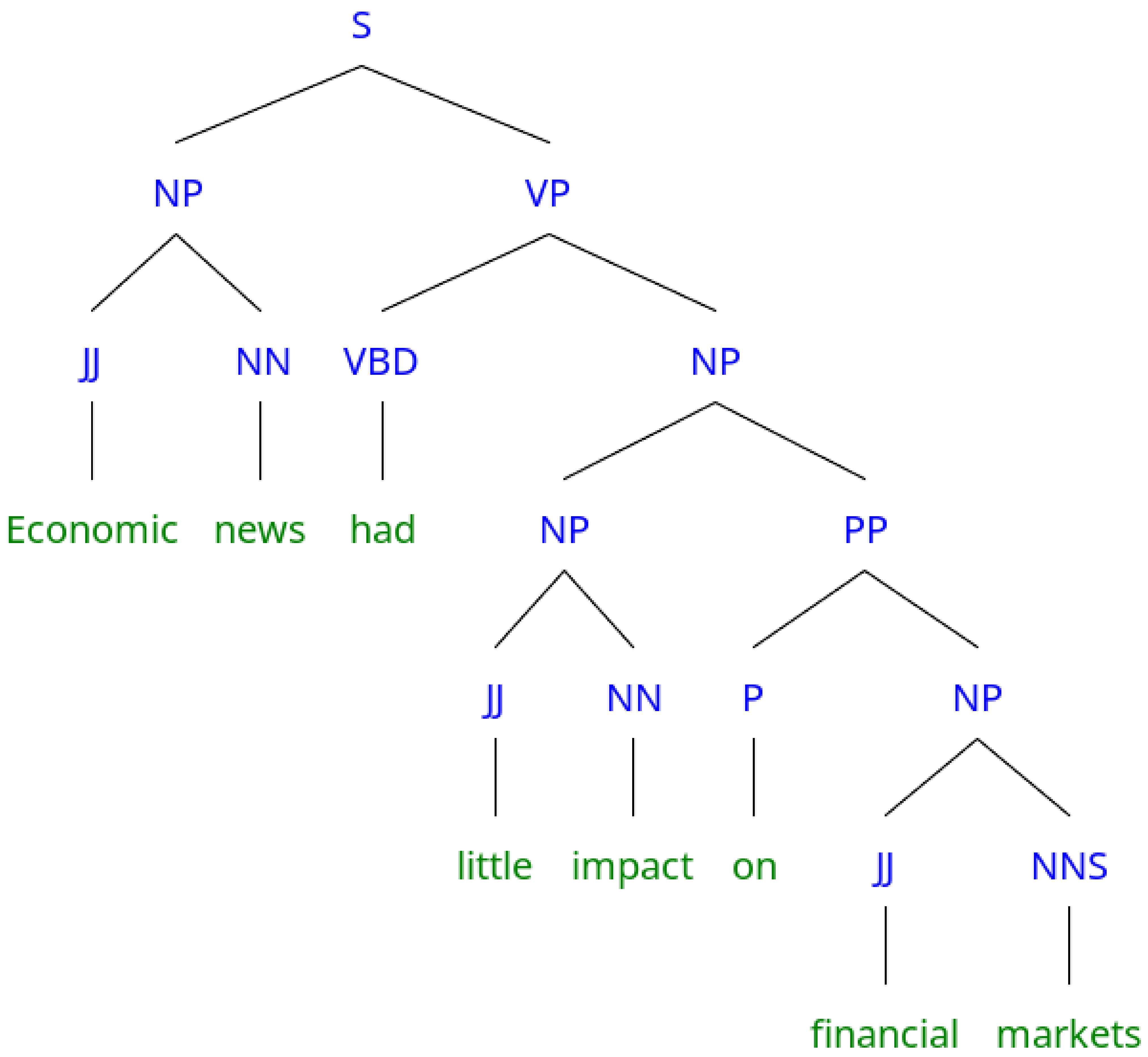
Roadmap

- Dependency Grammars
  - Definition
  - Motivation:
    - Limitations of Context-Free Grammars
- **Dependency Parsing**
  - By conversion from CFG
  - By Graph-based models
  - By transition-based parsing

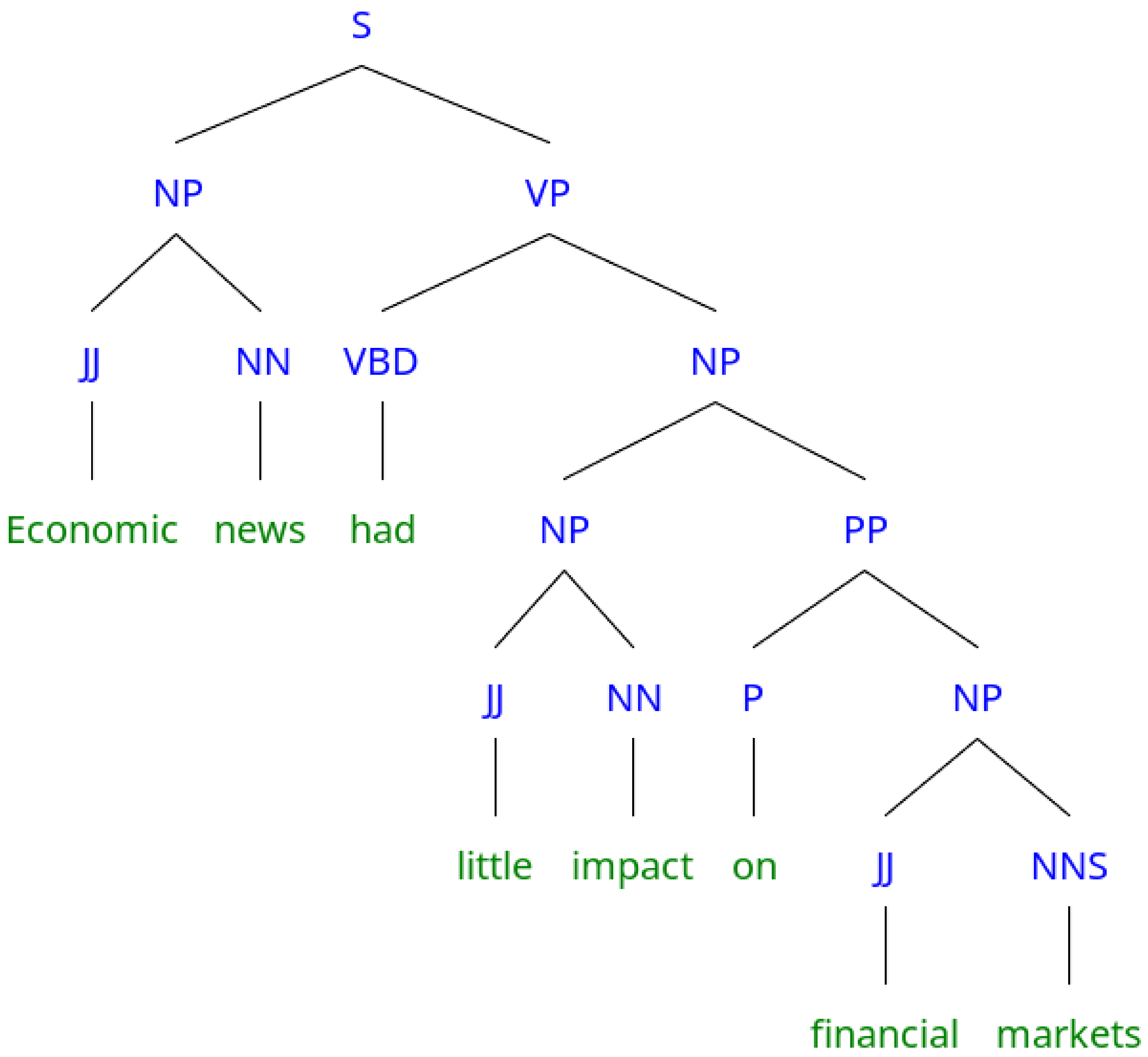
Conversion: PS → DS

- Can convert Phrase Structure (PS) to Dependency Structure (DS)
  - …without the dependency labels
- Algorithm:
  - Identify all head children in PS
  - Make head of each non-head-child depend on head of head-child
  - Use a *head percolation* table to determine headedness

Conversion: PS → DS

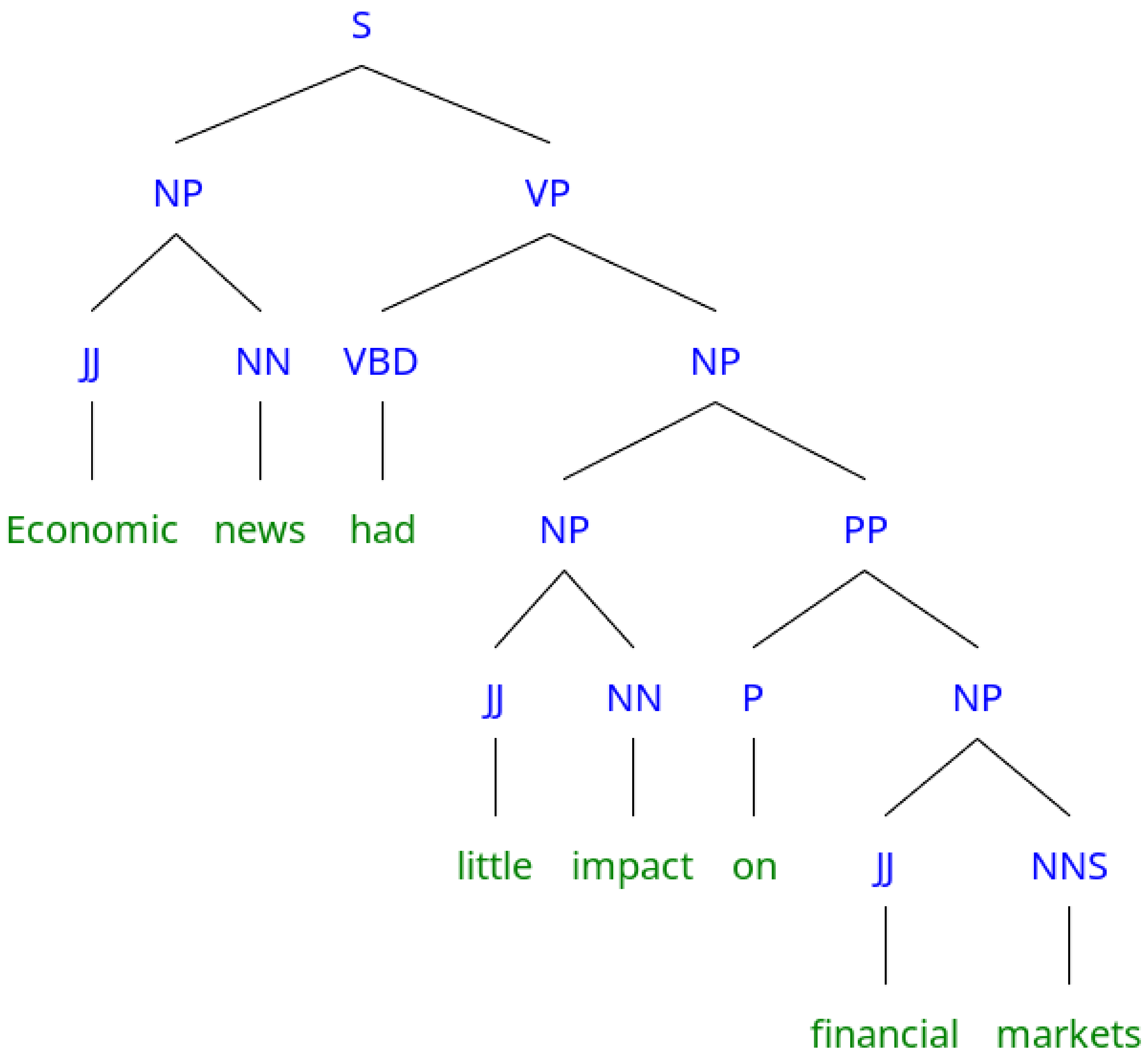


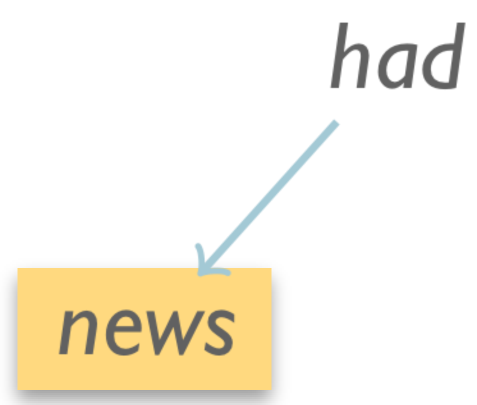
Conversion: PS → DS



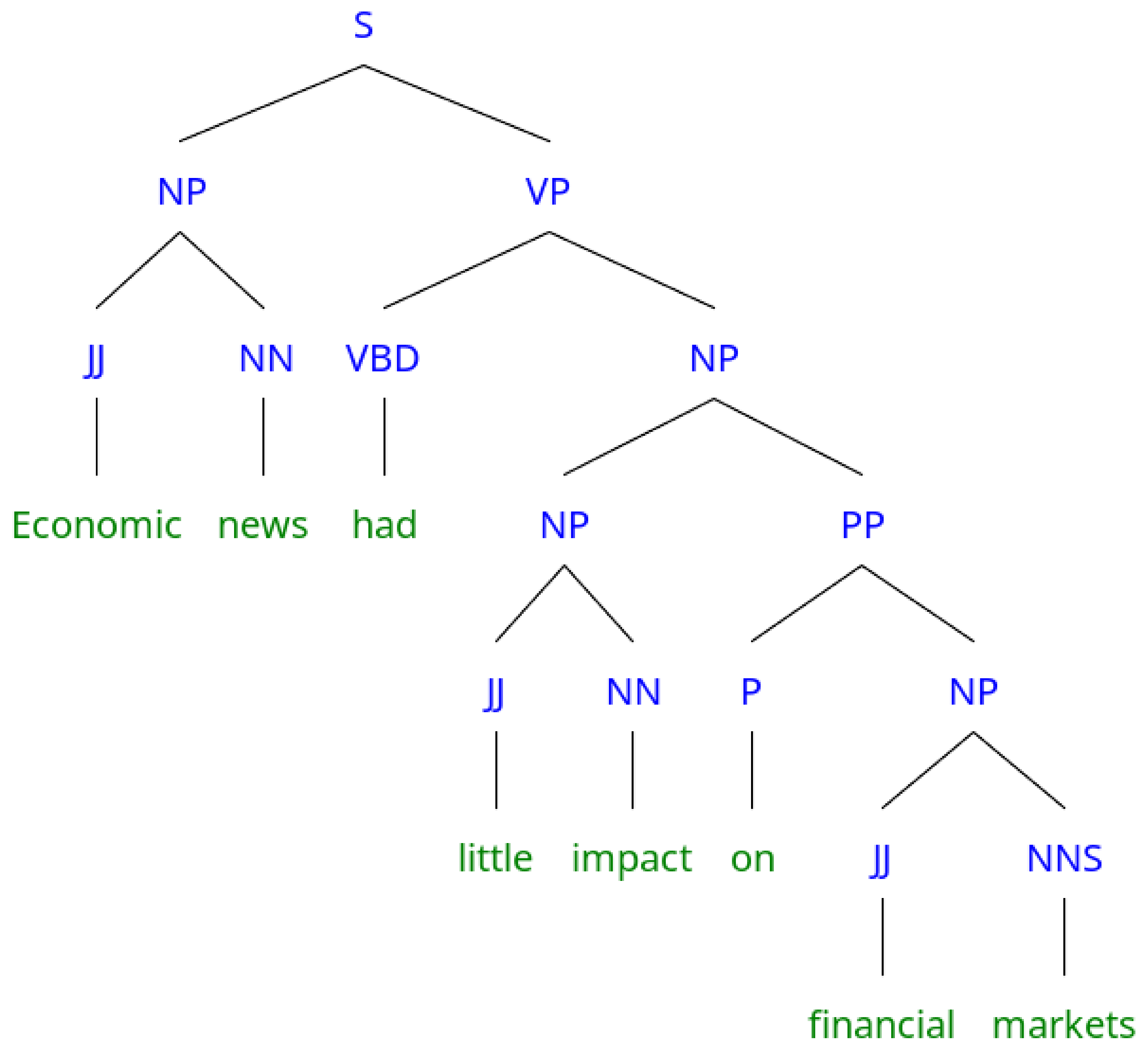
had

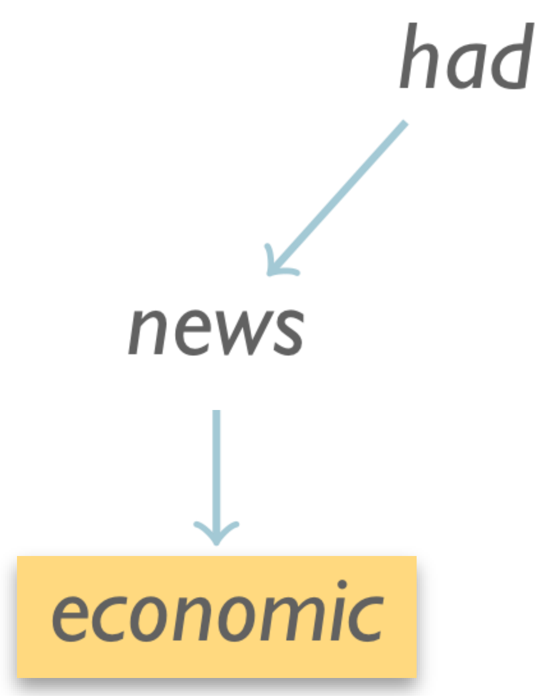
Conversion: PS → DS



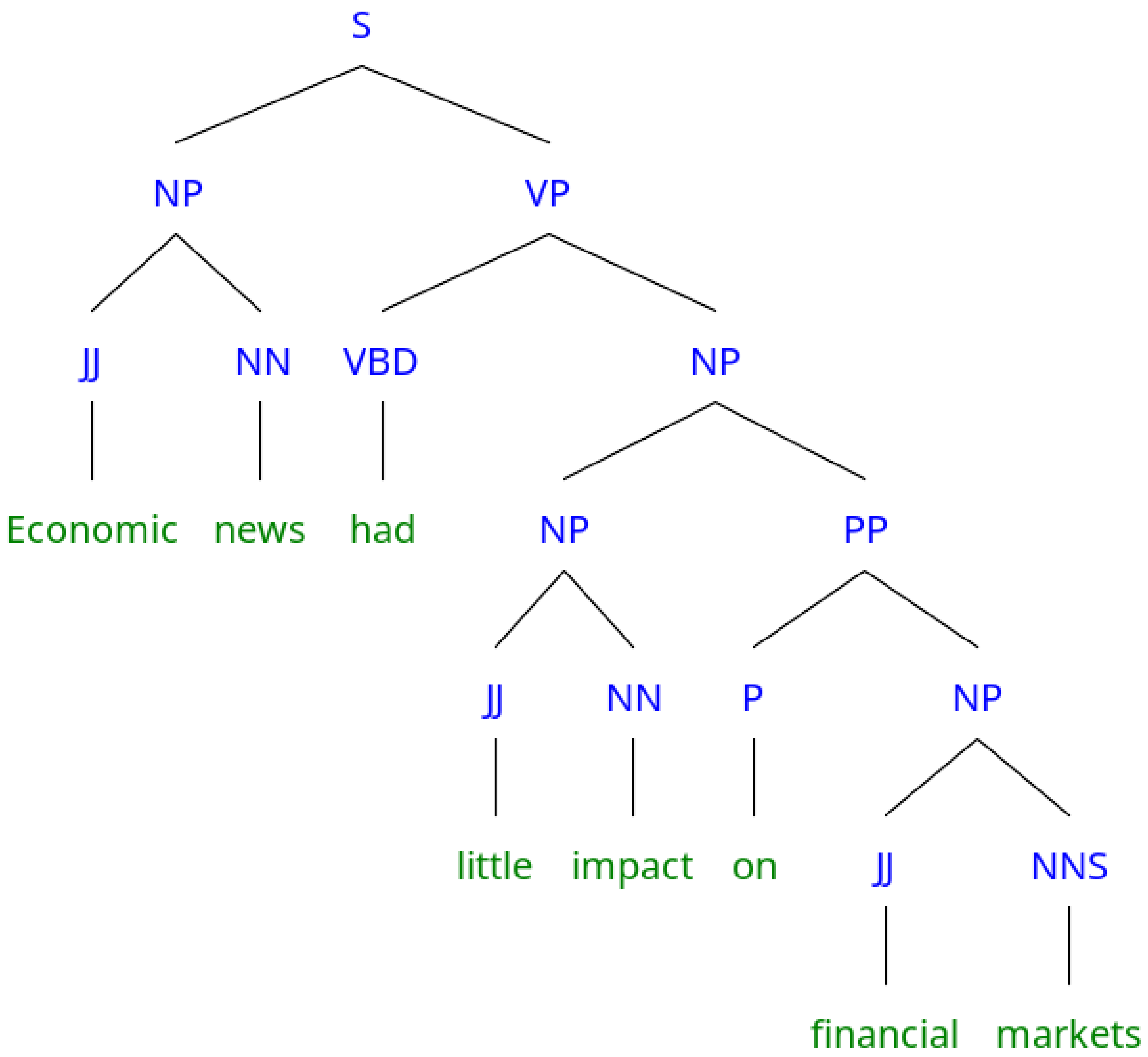


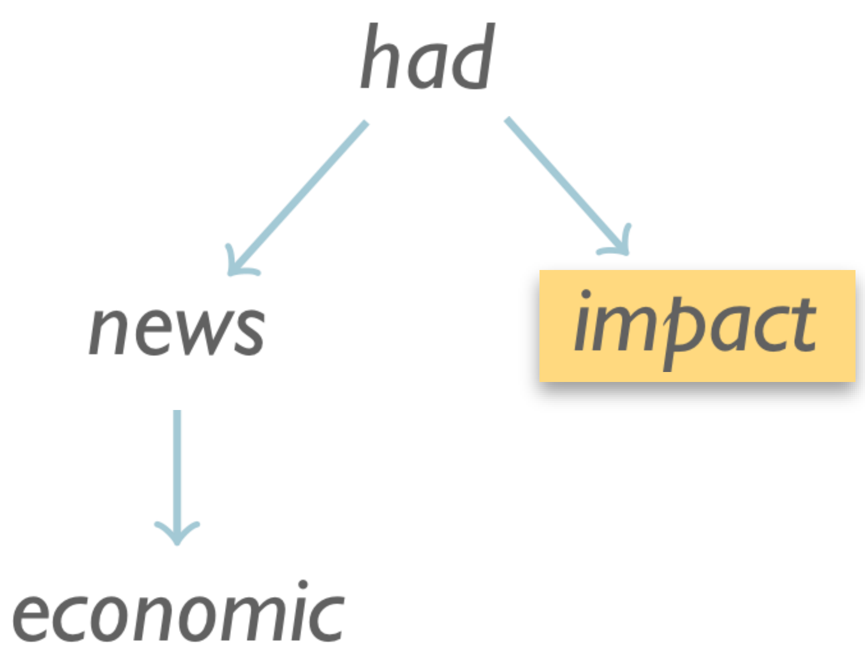
Conversion: PS → DS



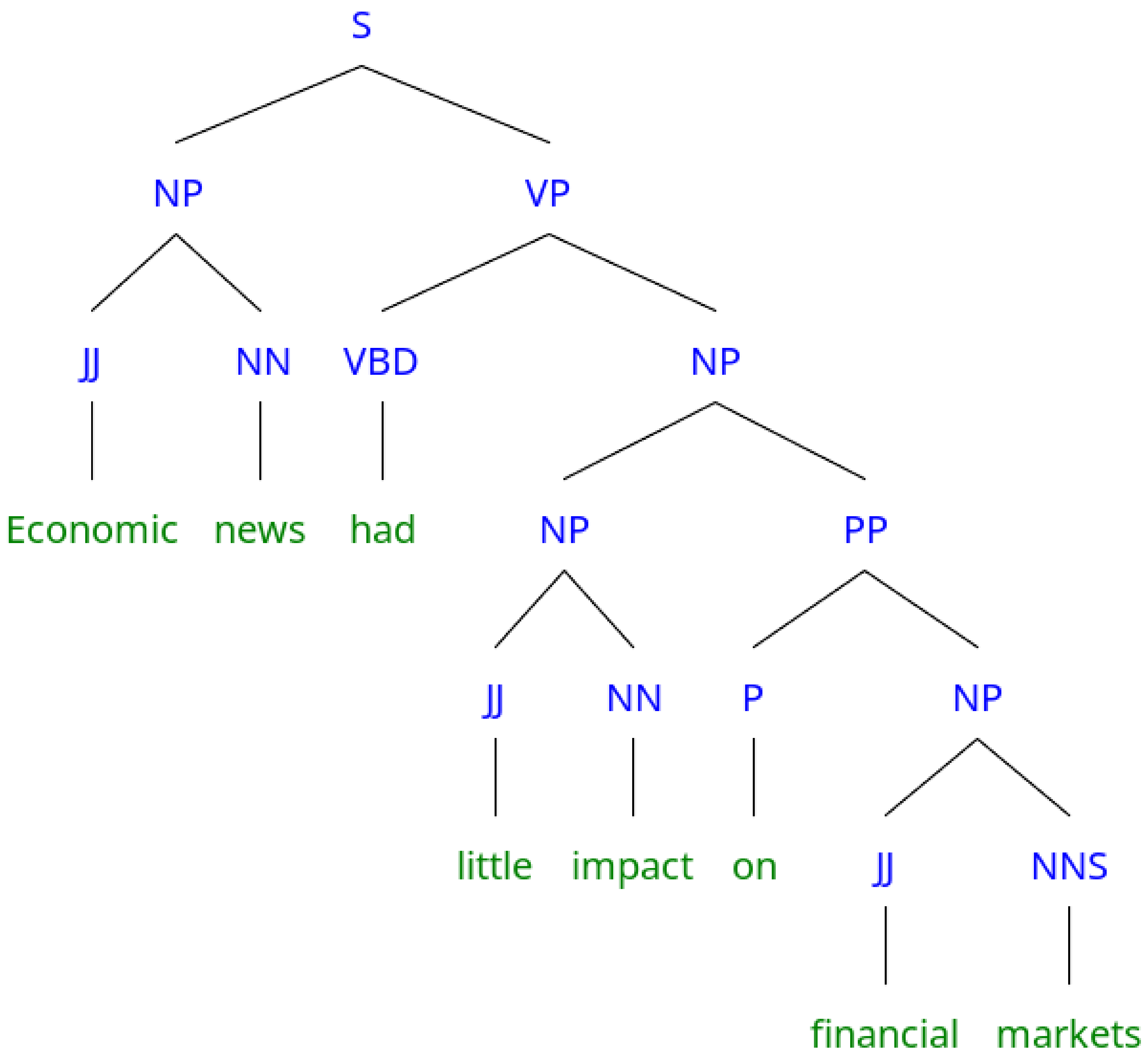


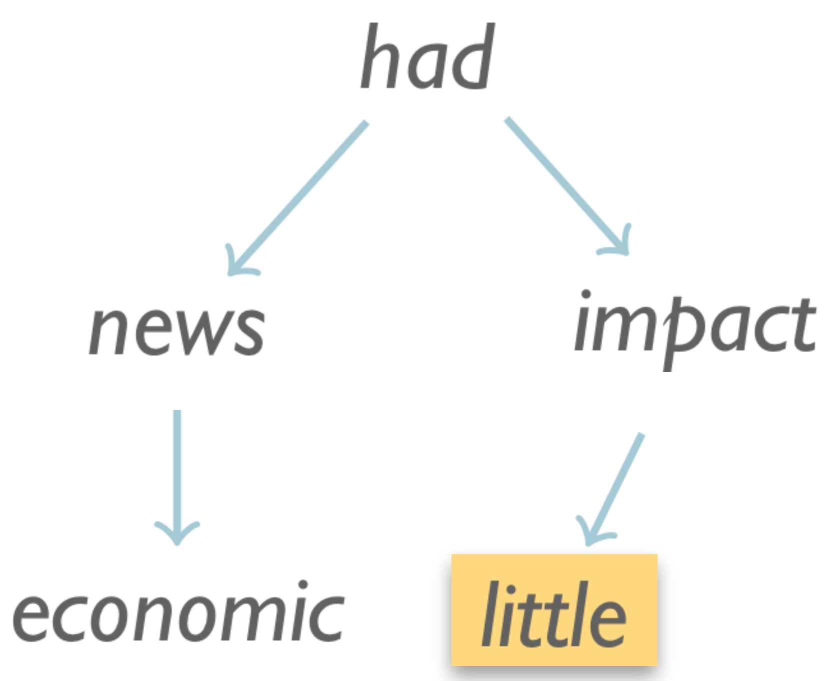
Conversion: PS → DS



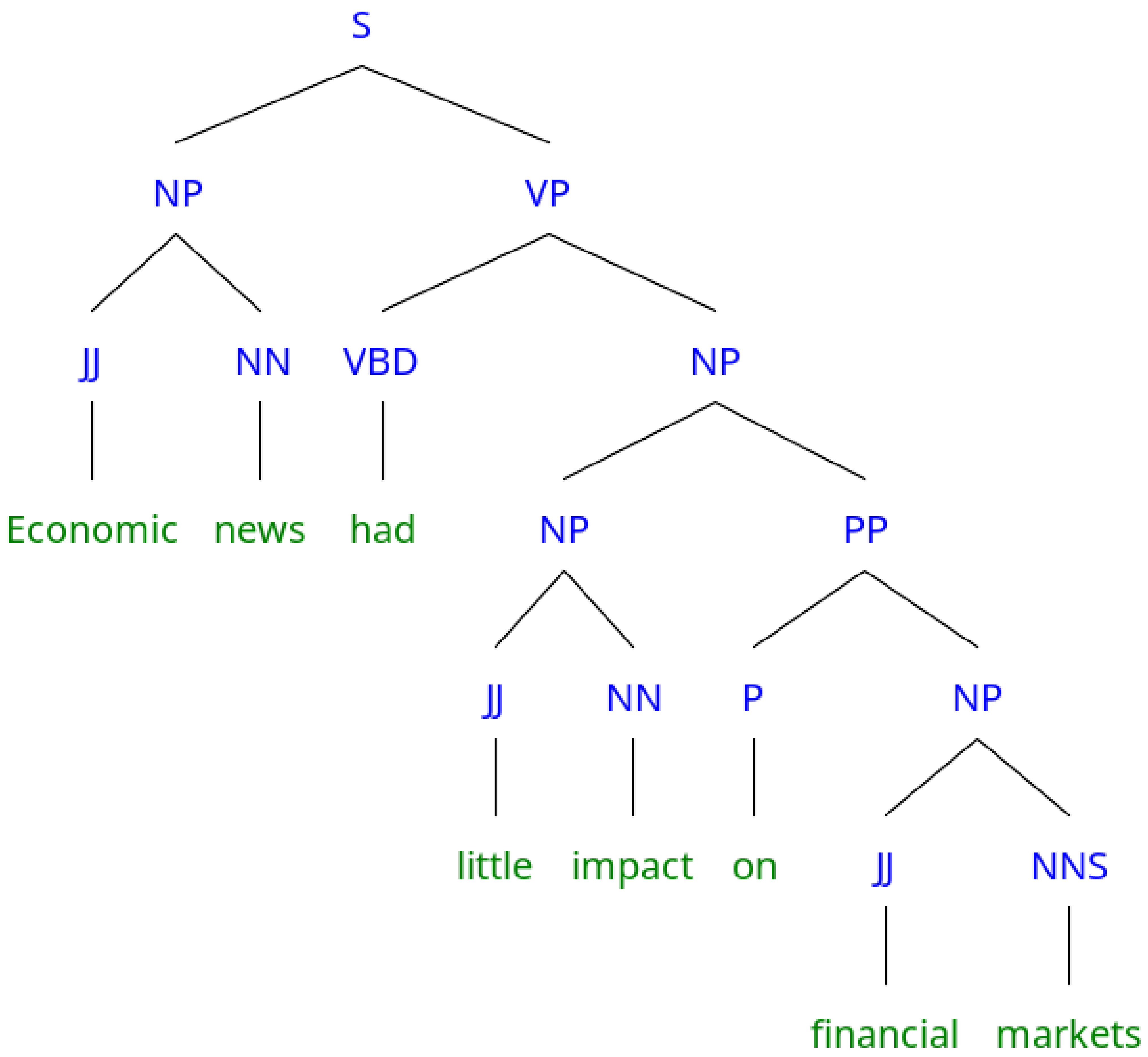


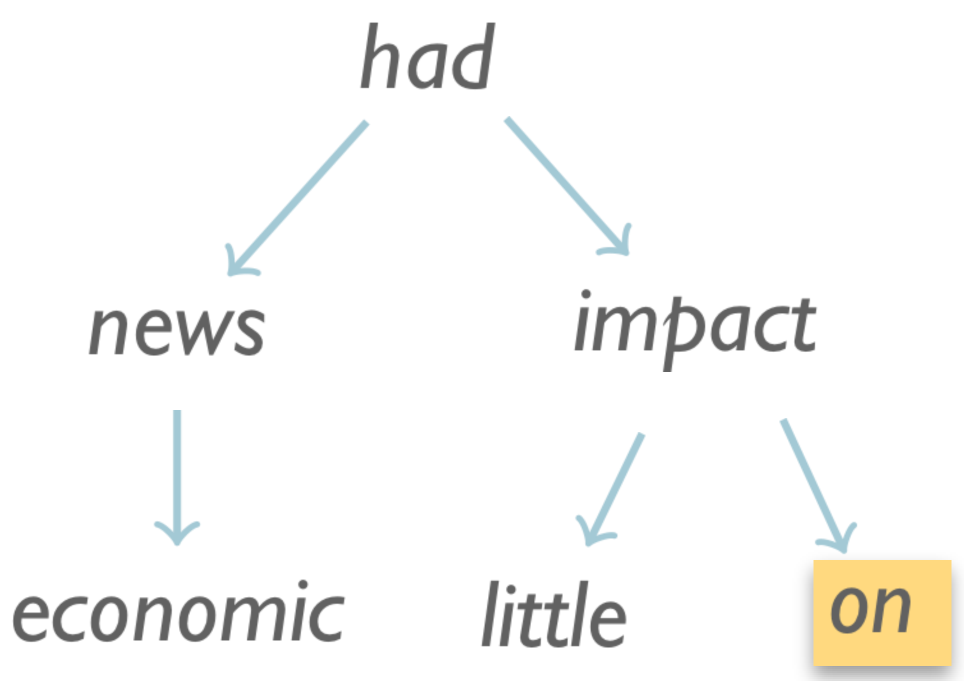
Conversion: PS → DS



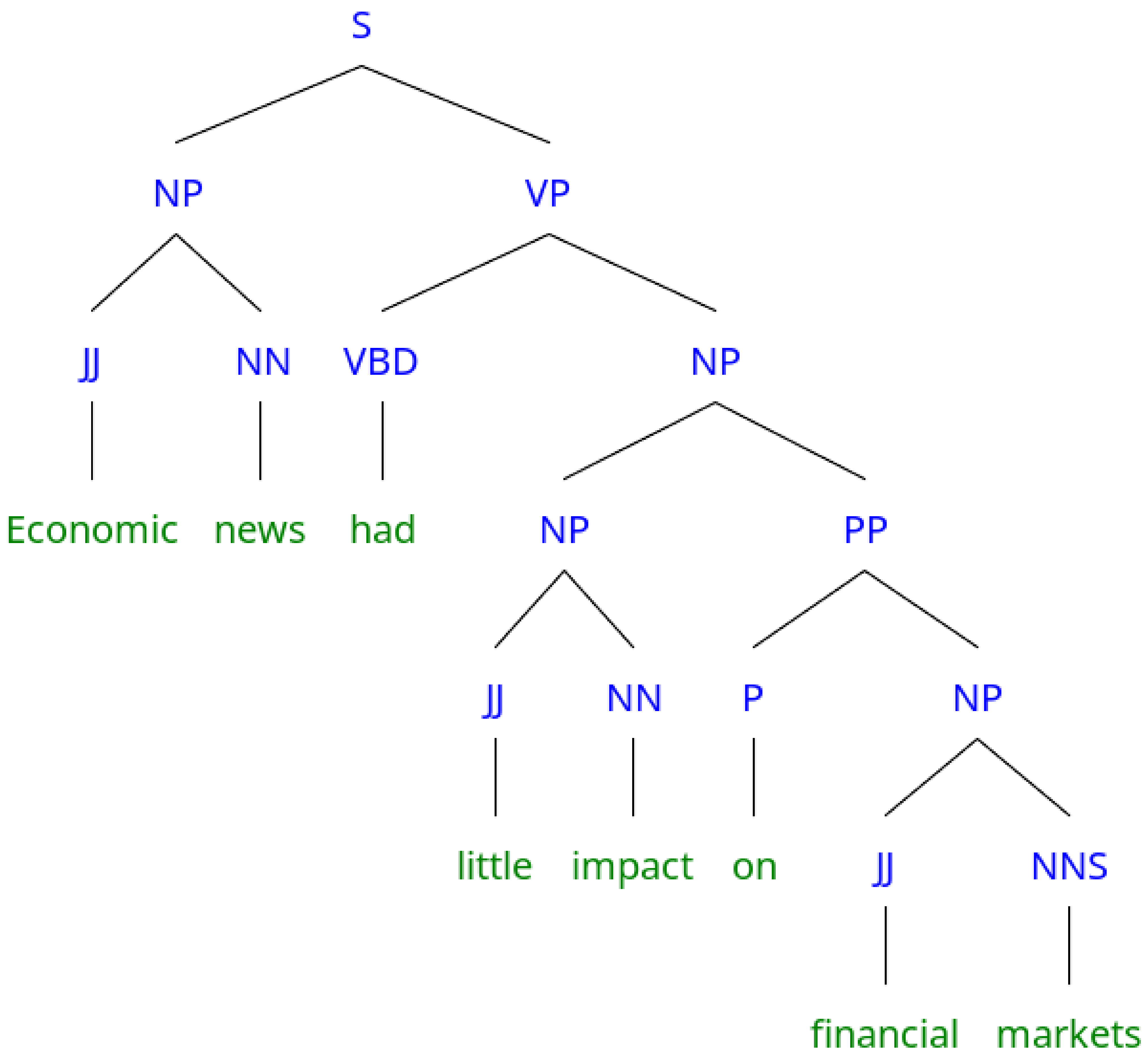


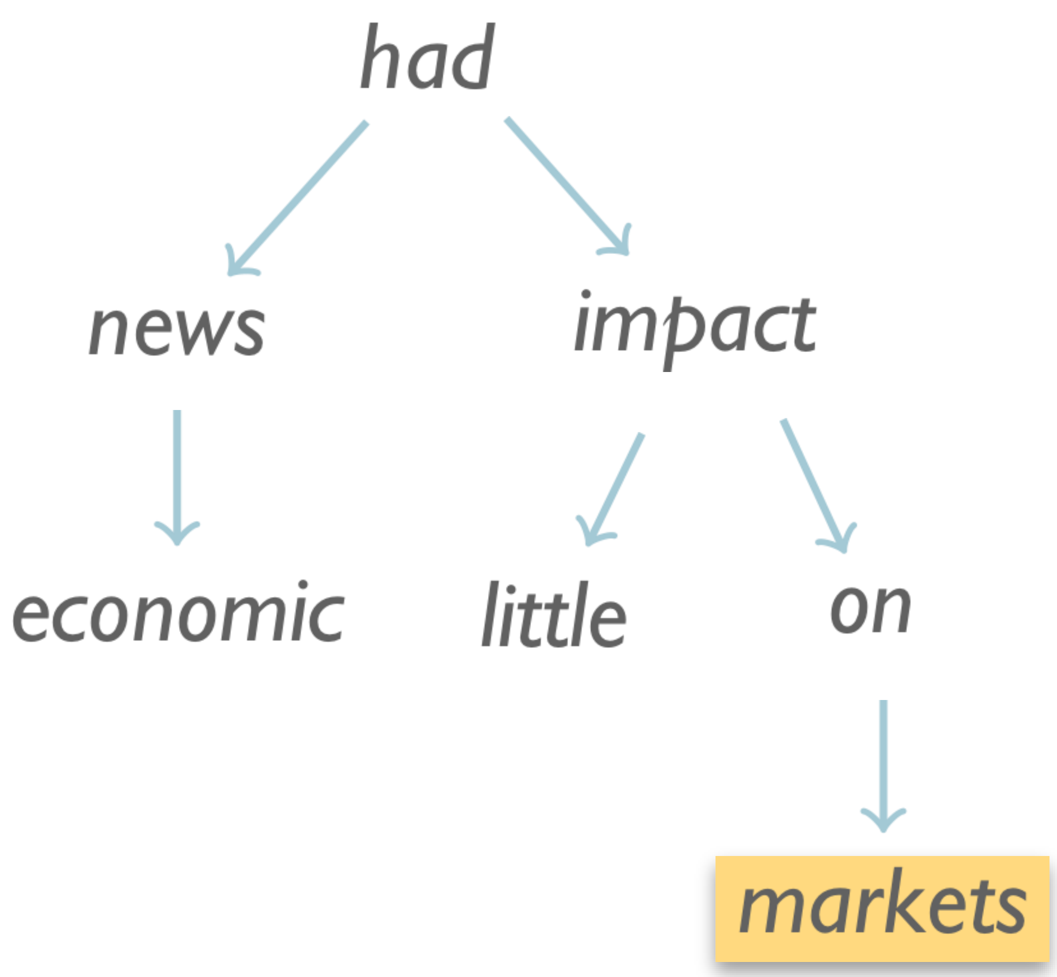
Conversion: PS → DS



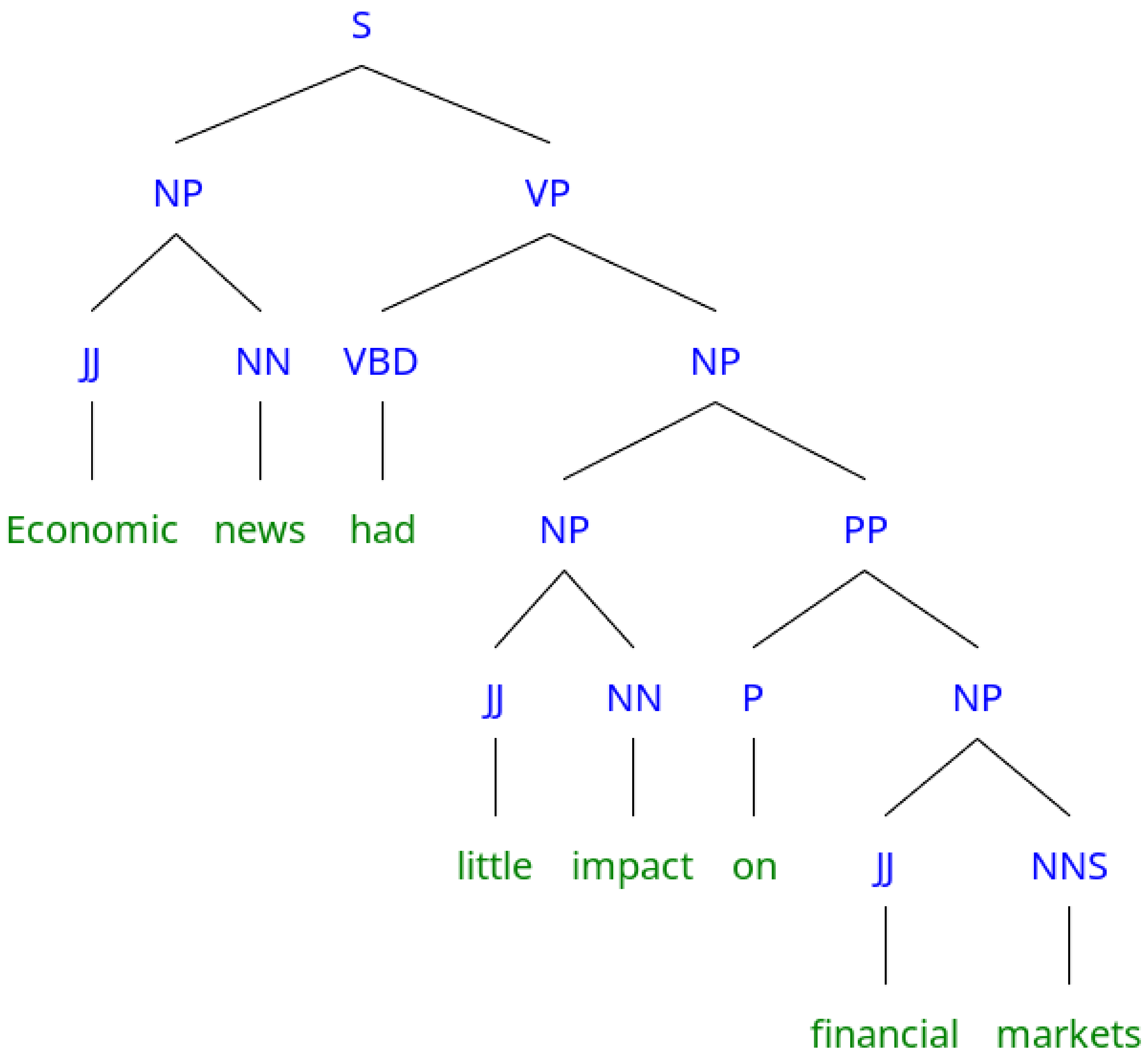


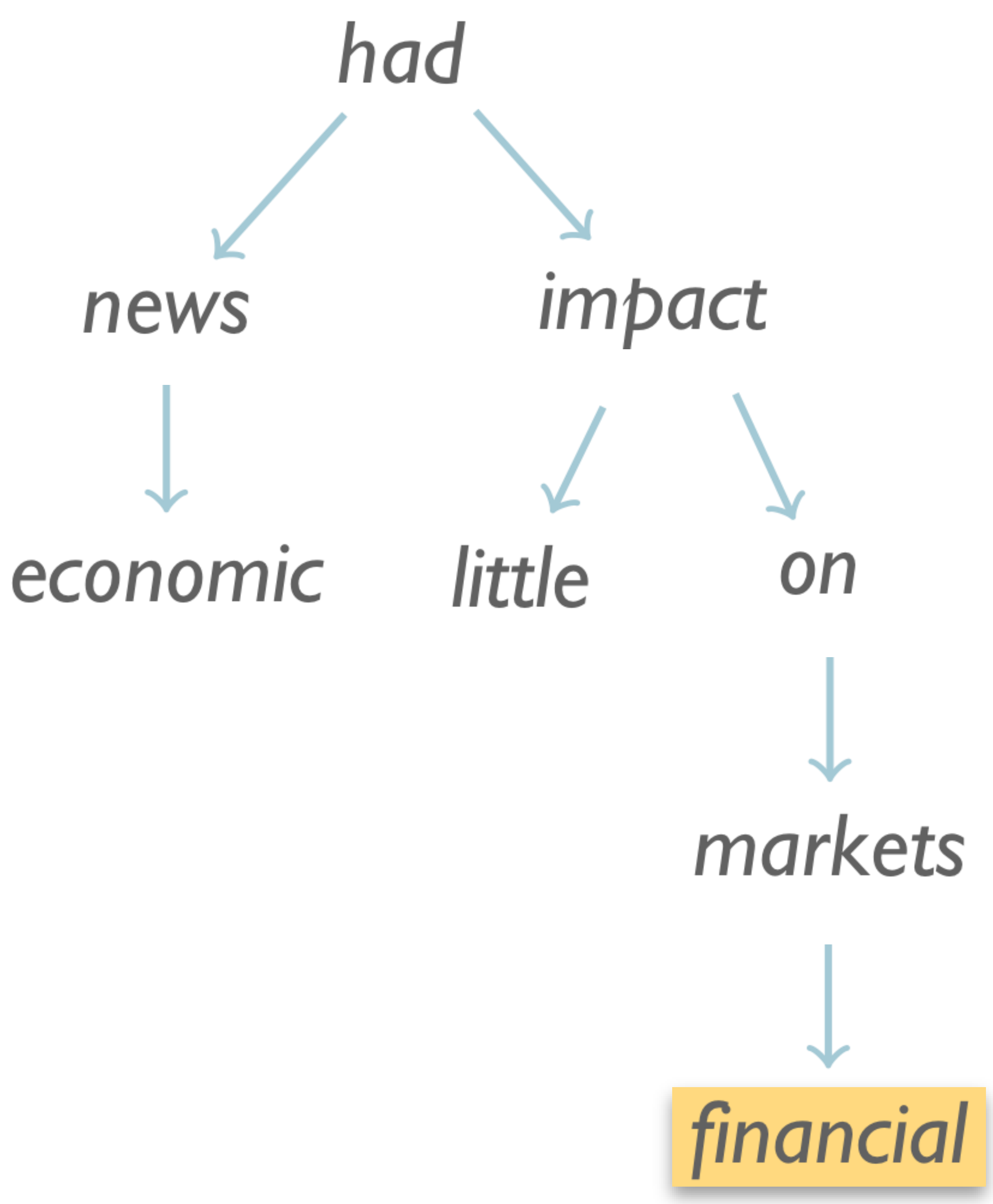
Conversion: PS → DS





Conversion: PS → DS





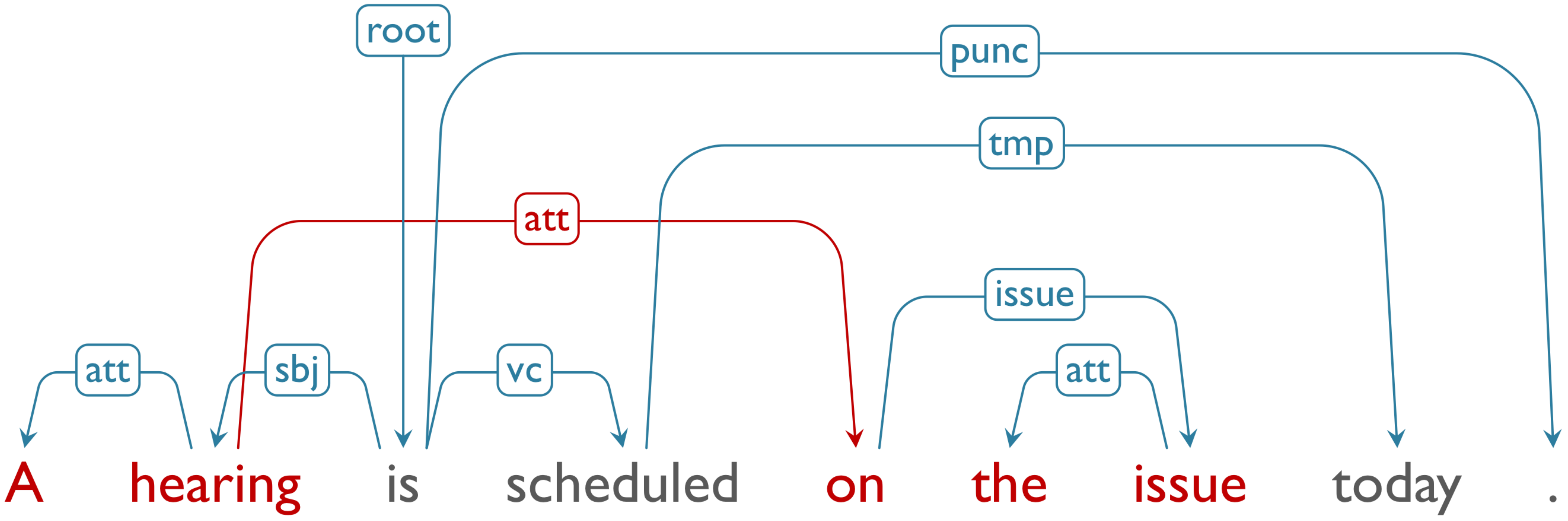
Head Percolation Table

- Finding the head of an NP:
  - If the rightmost word is preterminal, return
  - …else search Right→Left for first child which is *NN, NNP, NNPS*…
  - …else search Left→Right for first child which is *NP*
  - …else search Right→Left for first child which is *$, ADJP, PRN*
  - …else search Right→Left for first child which is *CD*
  - …else search Right→Left for first child which is *JJ, JJS, RB or QP*
  - …else return rightmost word.

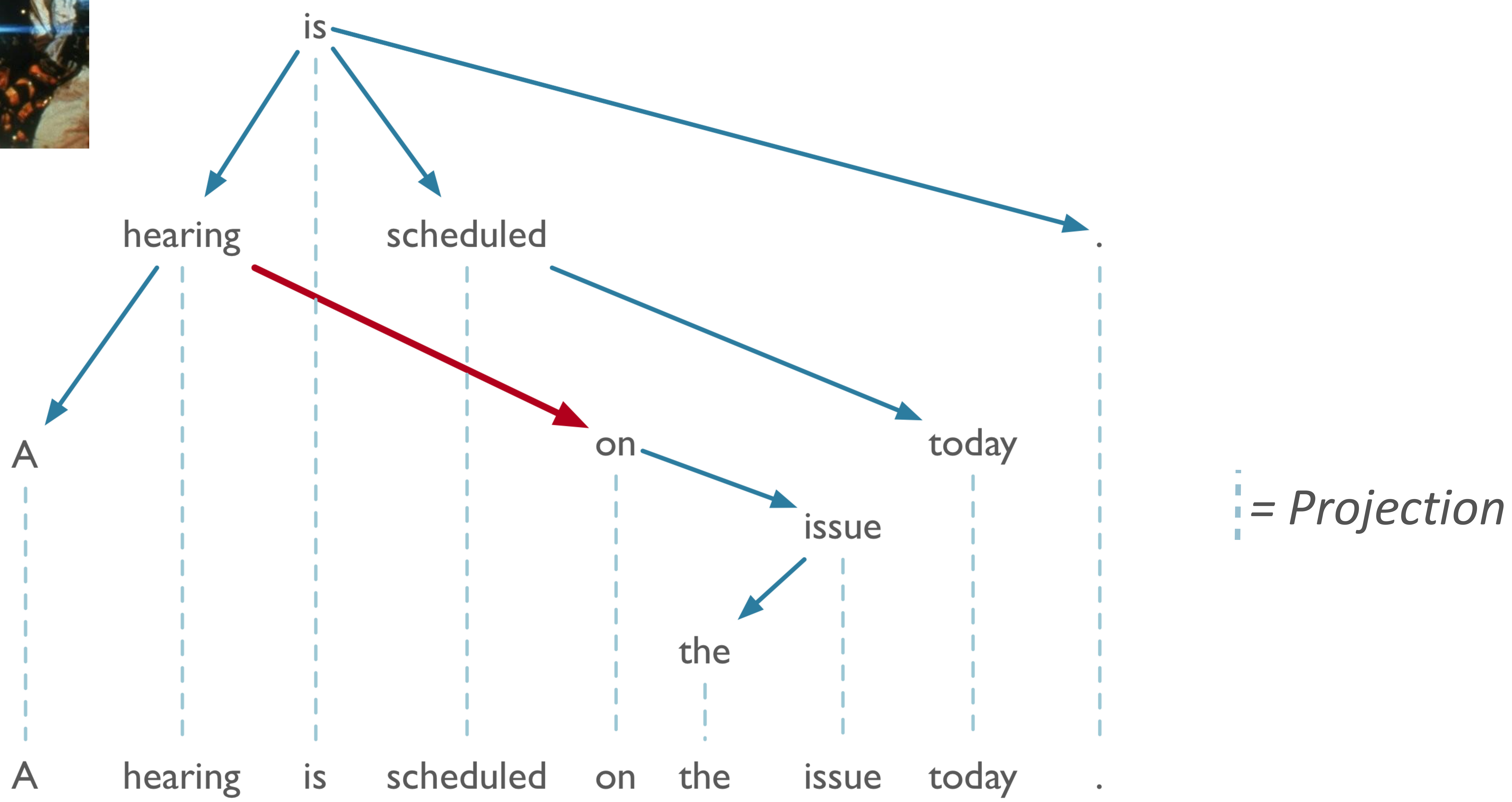
From J&M Page 411, via [Collins (1999)]

Conversion: DS → PS

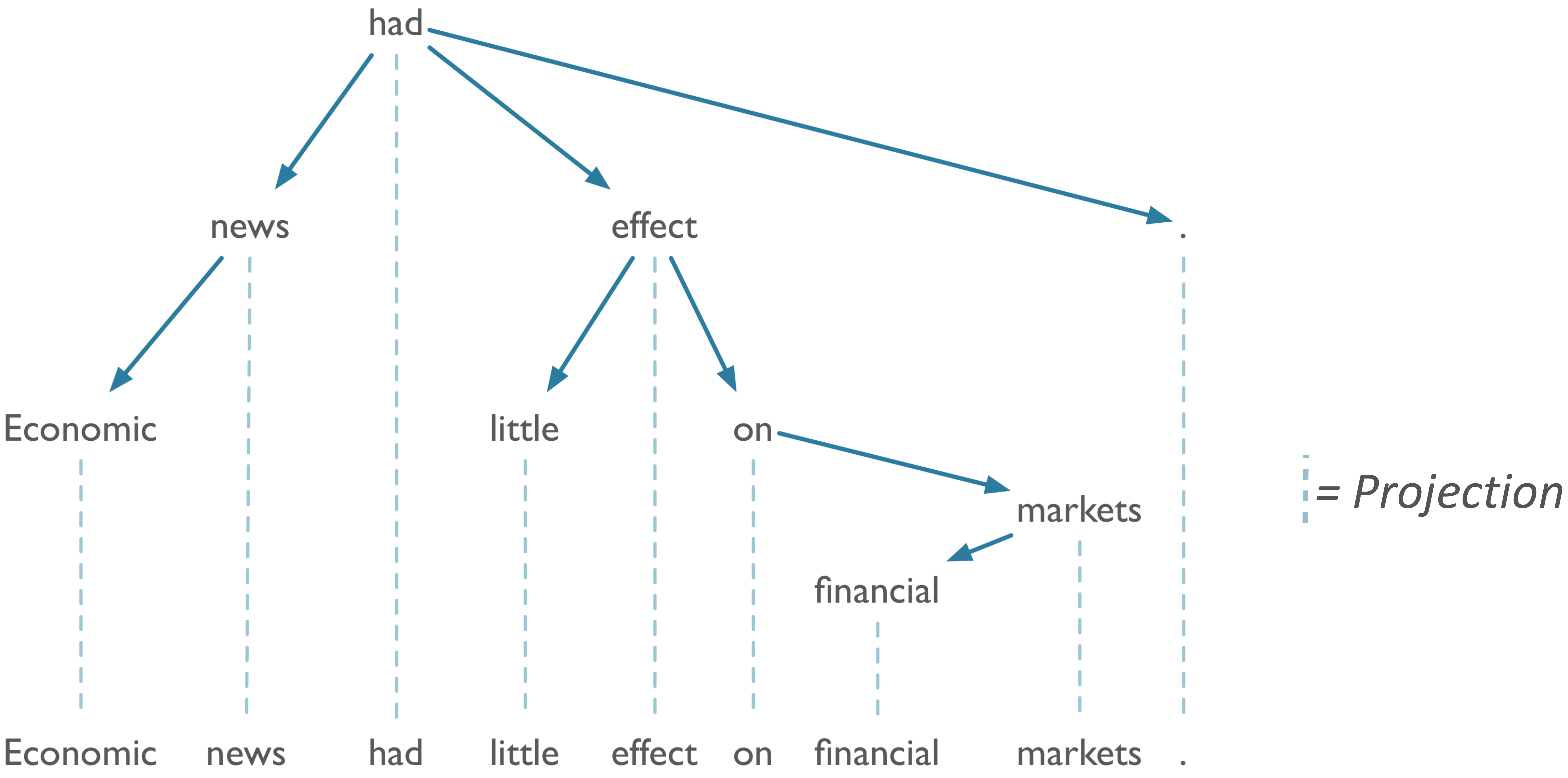
- Can map any *projective* dependency tree to PS tree
- Projective:
  - Does not contain “crossing” dependencies w.r.t. word order



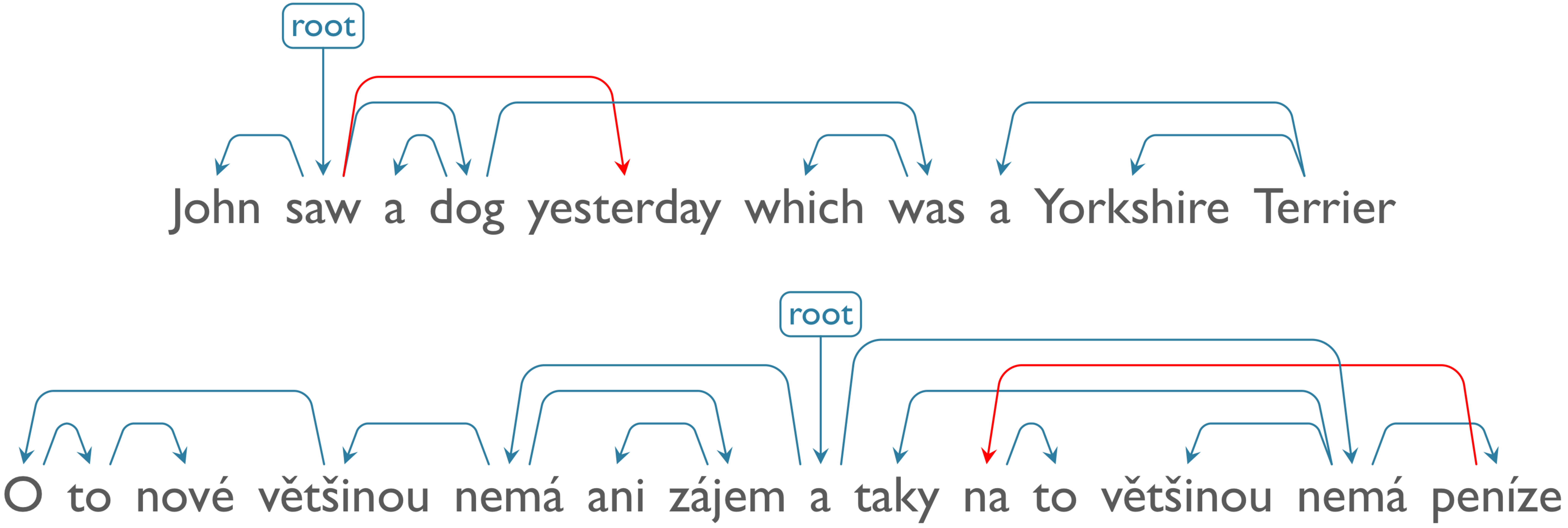
Non-Projective DS



Projective DS



More Non-Projective Parses



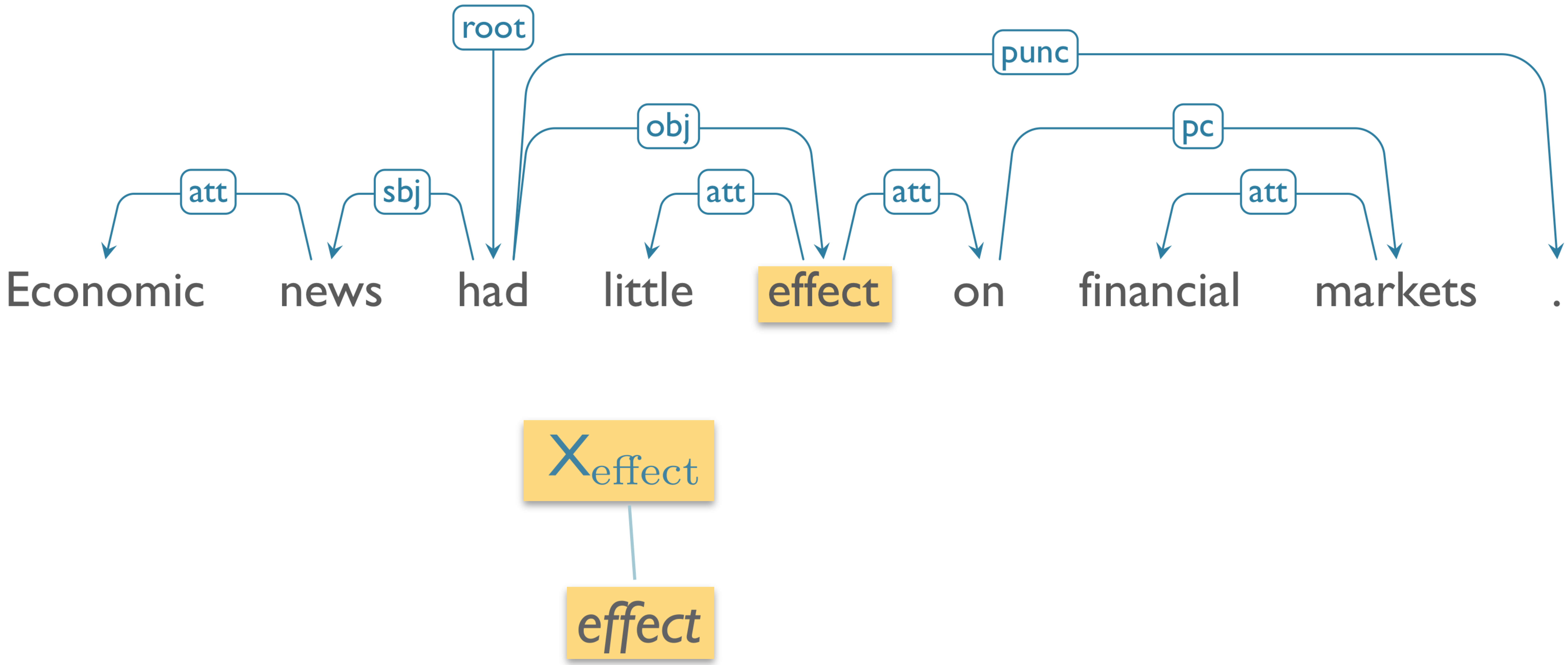
He is mostly not even interested in the new things and in most cases, he has no money for it either.

From [McDonald et. al, 2005]

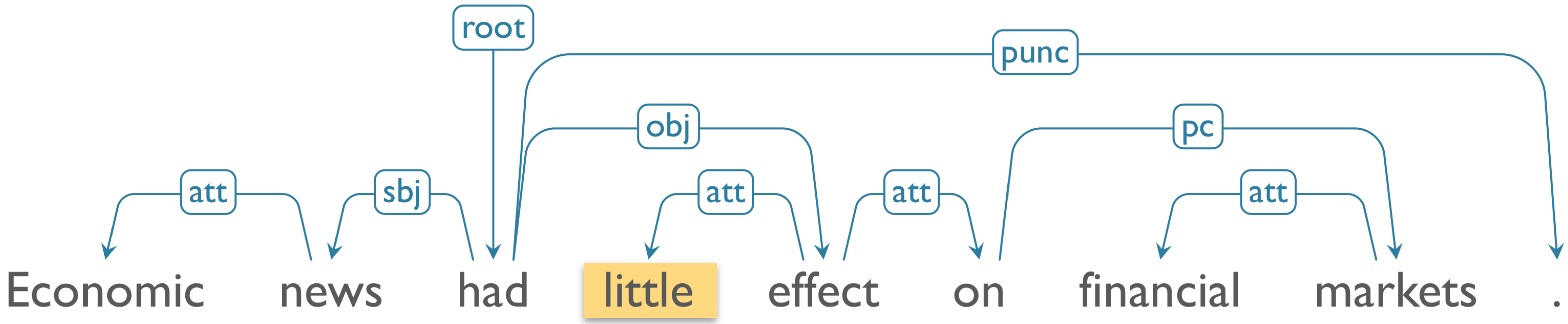
Conversion: DS → PS

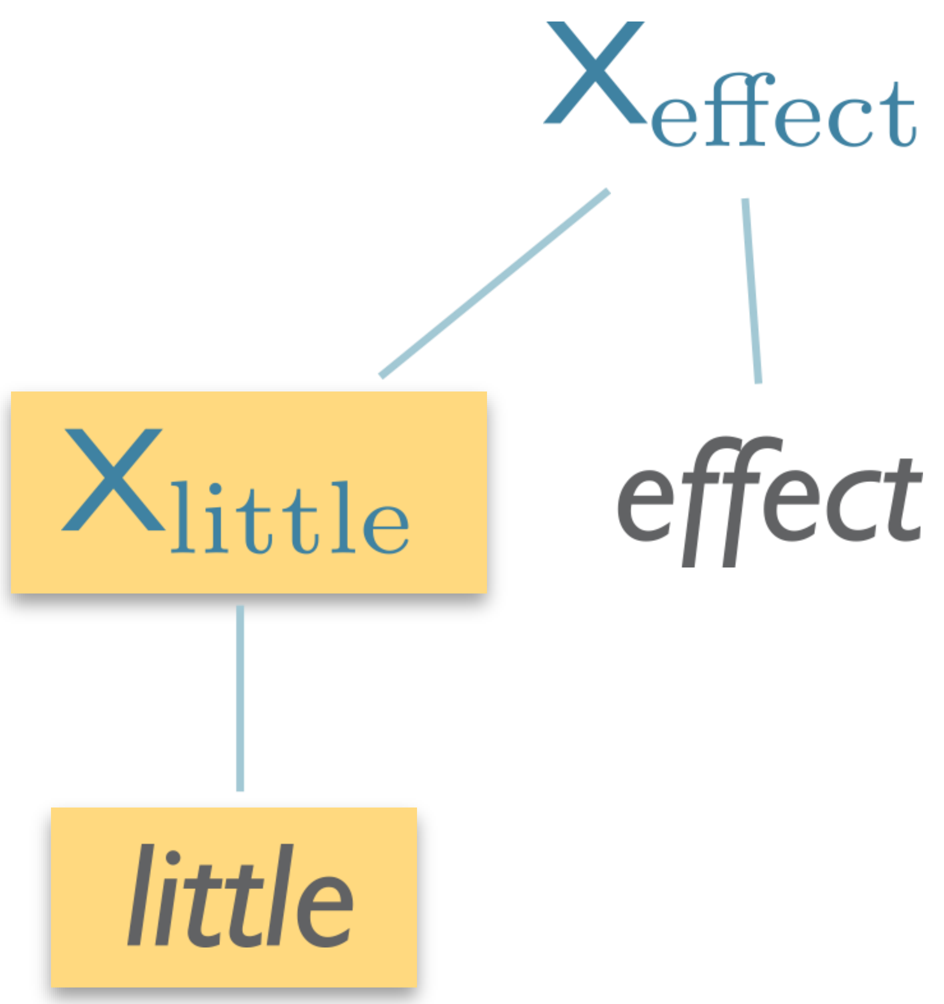
- For each node ***w*** with outgoing arcs…
  - …convert the subtree ***w*** and its dependents $\boldsymbol{t_1},\ldots,\boldsymbol{t_n}$ to a new subtree:
    - Nonterminal: X_w
    - Child: *w*
    - Subtrees $\boldsymbol{t_1},\ldots,\boldsymbol{t_n}$ in original sentence order

Conversion: DS → PS

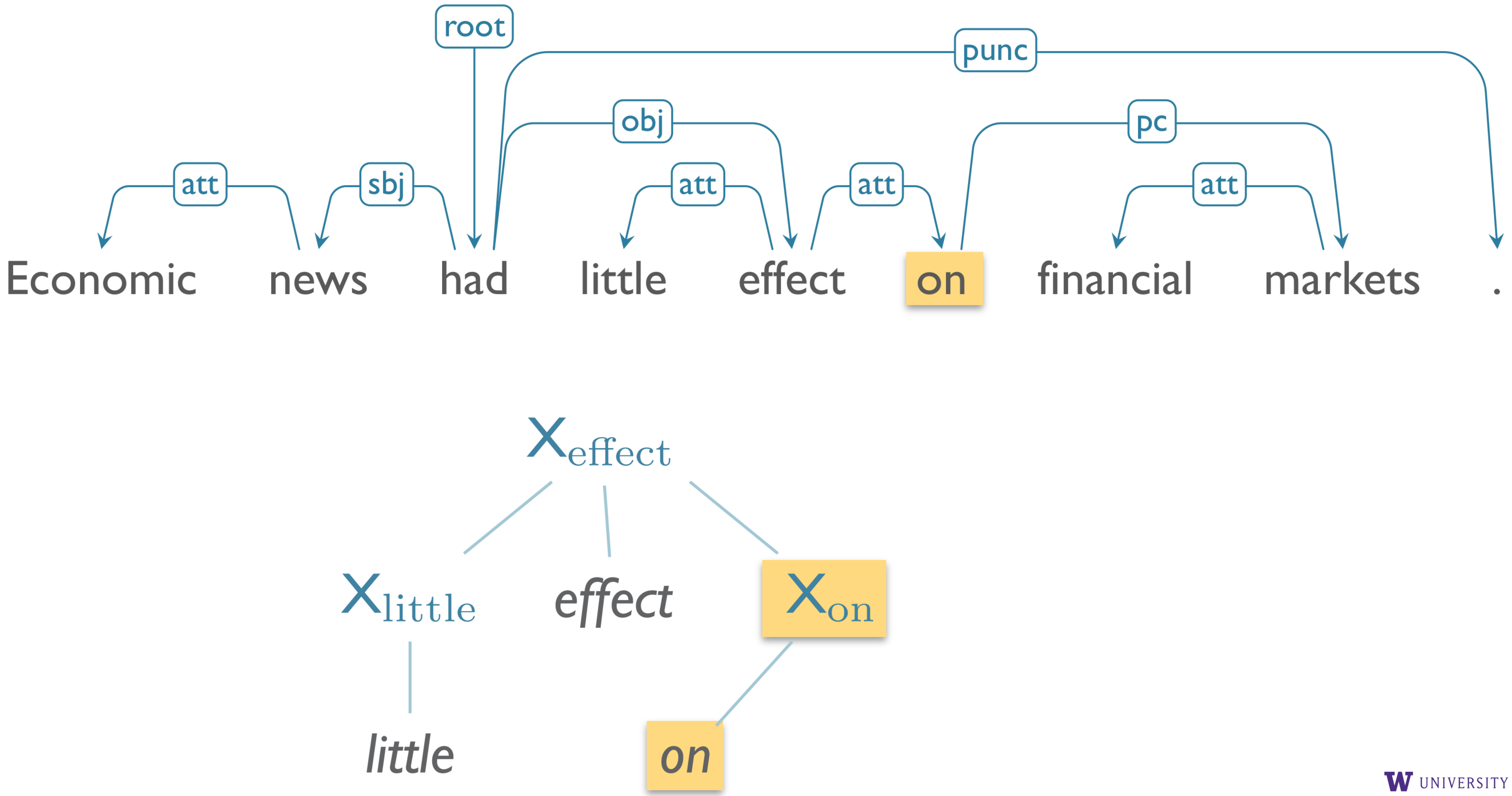


Conversion: DS → PS

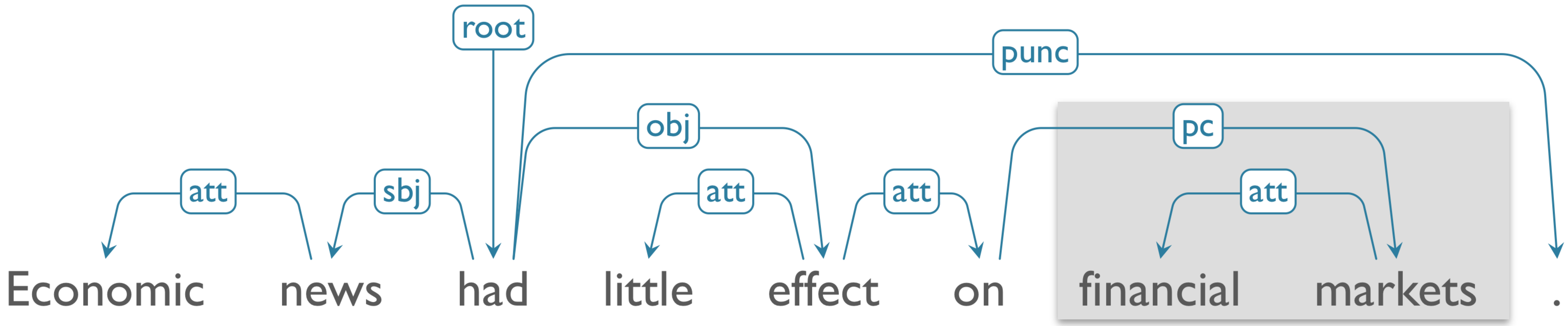


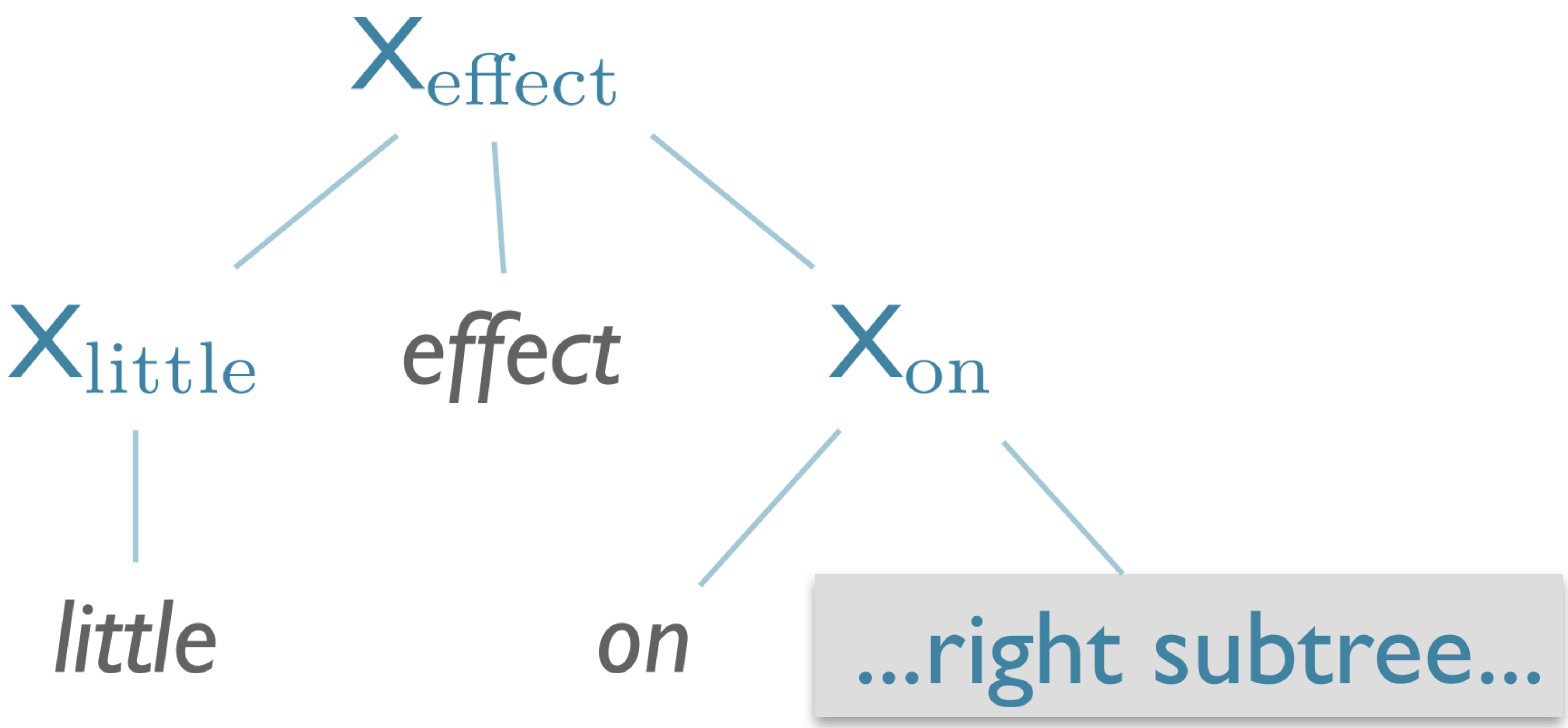


Conversion: DS → PS



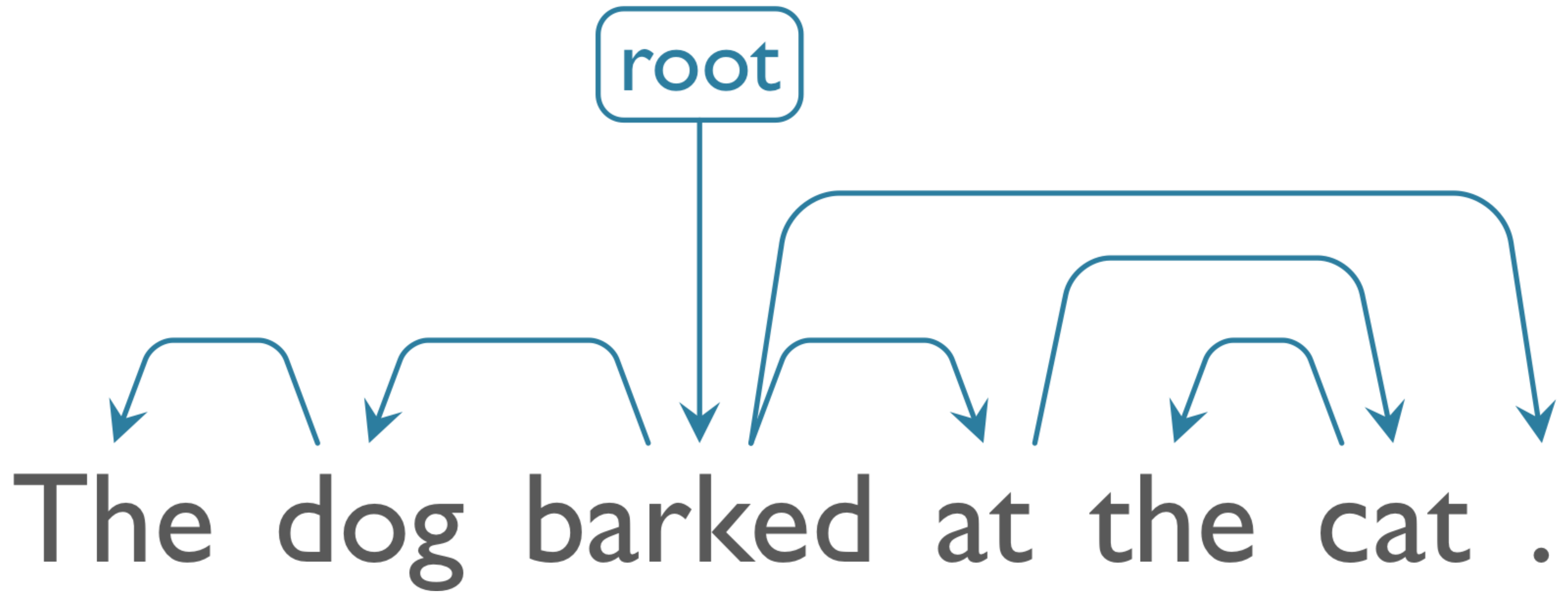
Conversion: DS → PS

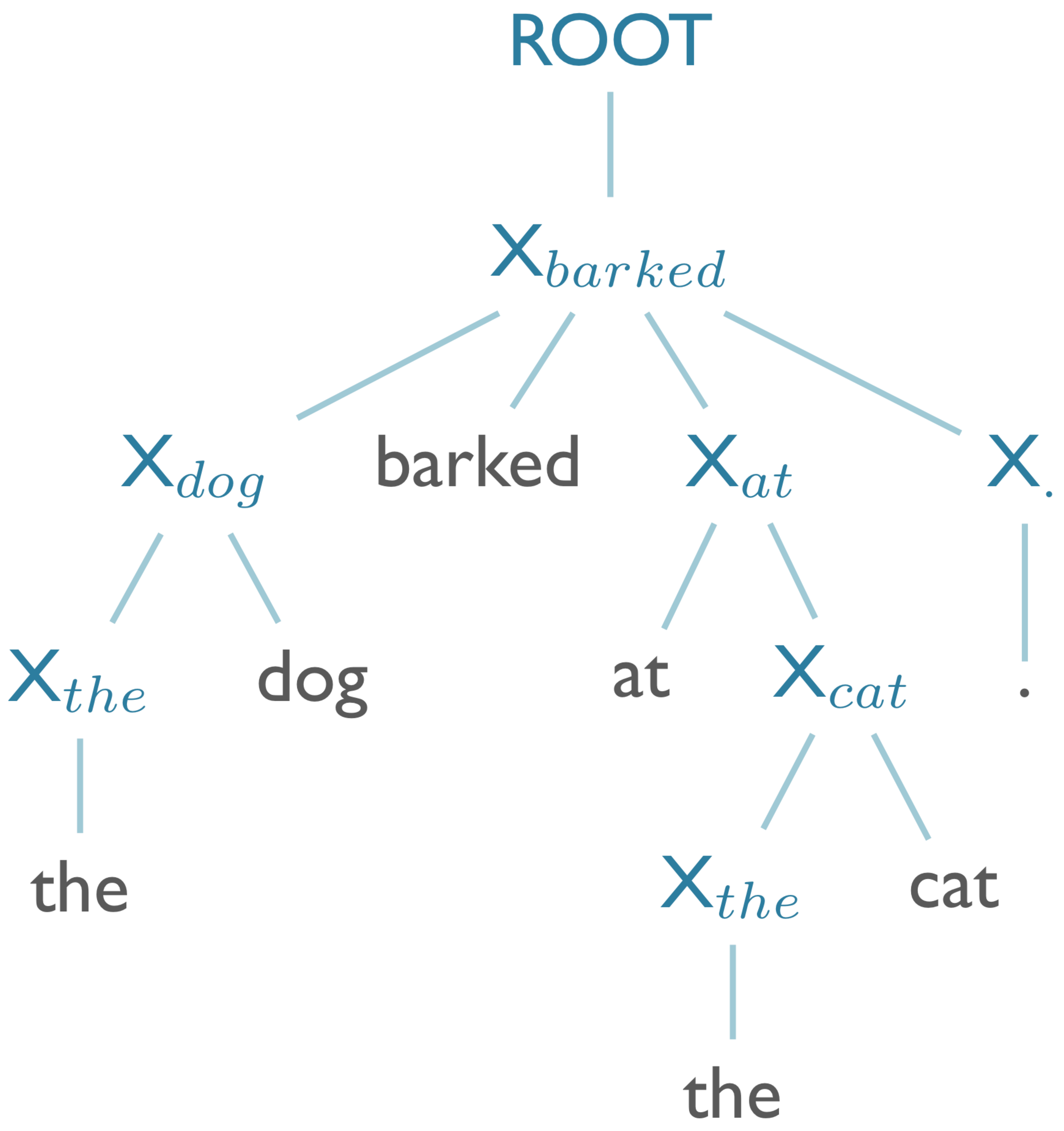




Conversion: DS → PS

- What about labeled dependencies?
  - Can attach labels to nonterminals associated with non-heads
  - e.g. $X_{\textbf{\textit{little}}} \rightarrow X_{\textbf{\textit{little:nmod}}}$
- Doesn't create typical PS trees
  - *Does* create fully lexicalized, labeled, context-free trees
- Can be parsed with any standard CFG parser





Example from J. Moore, 2013

Roadmap

- Dependency Grammars
  - Definition
  - Motivation:
    - Limitations of Context-Free Grammars
- **Dependency Parsing**
  - By conversion to CFG
  - By Graph-based models
  - By transition-based parsing

Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree $\hat{T}$ for sentence ***S***
  - If ***S*** is unambiguous, ***T*** is the correct parse
  - If ***S*** is ambiguous, ***T*** is the highest scoring parse
- Where do scores come from?
  - Weights on dependency edges by learning algorithm
  - Learned from dependency treebank
- Where are the grammar rules?
  - …there aren't any! All data-driven.

Graph-based Dependency Parsing

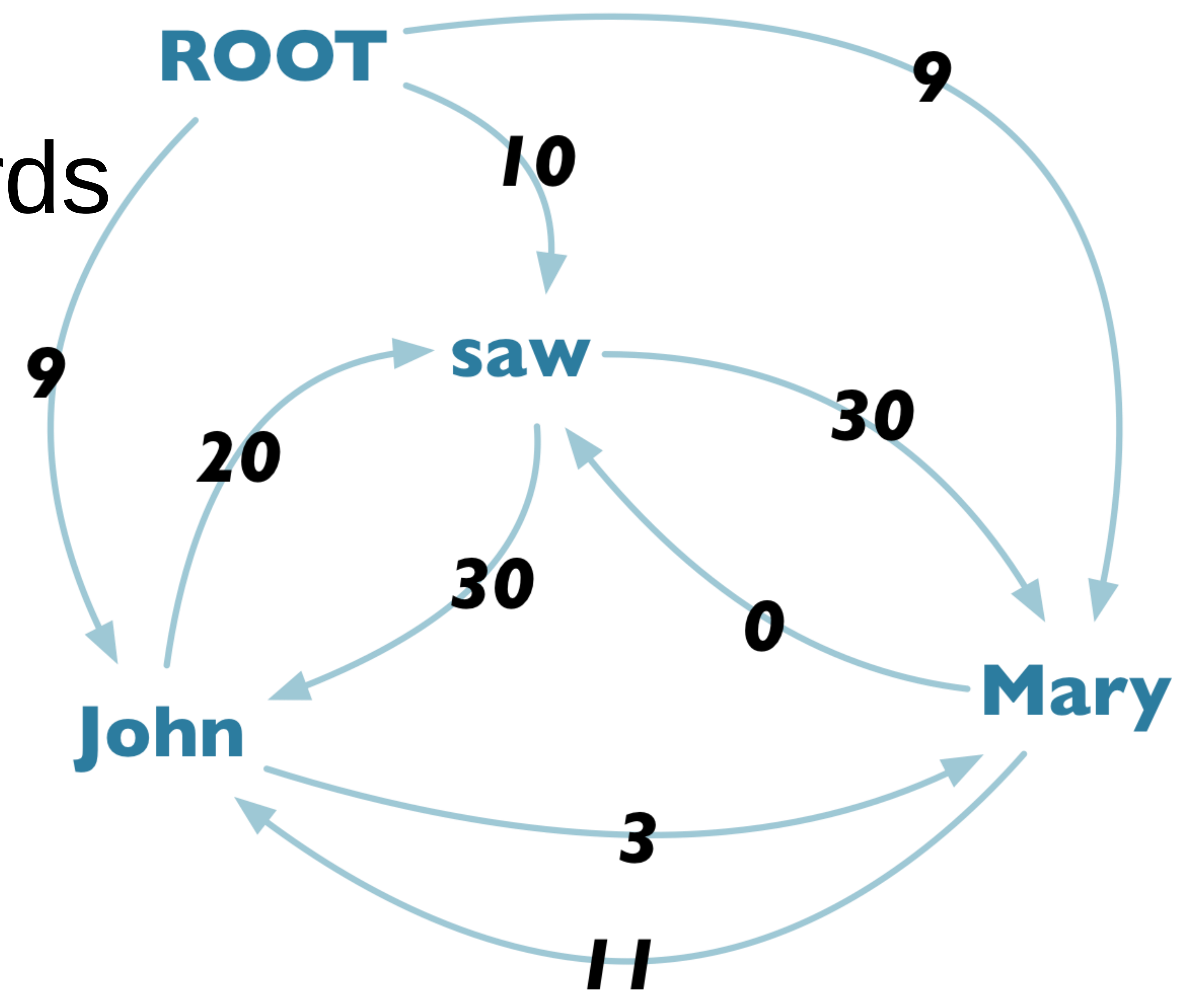
- Map dependency parsing to Maximum Spanning Tree (MST)
- Build fully connected initial graph:
  - Nodes: words in sentence to parse
  - Edges: directed edges between all words
    - + Edges from ROOT to all words
- Identify maximum spanning tree
  - Tree s.t. all nodes are connected
  - Select such tree with highest weight

Graph-based Dependency Parsing

- Arc-factored model:
  - Weights depend on end nodes & link
  - Weight of tree is sum of participating arcs

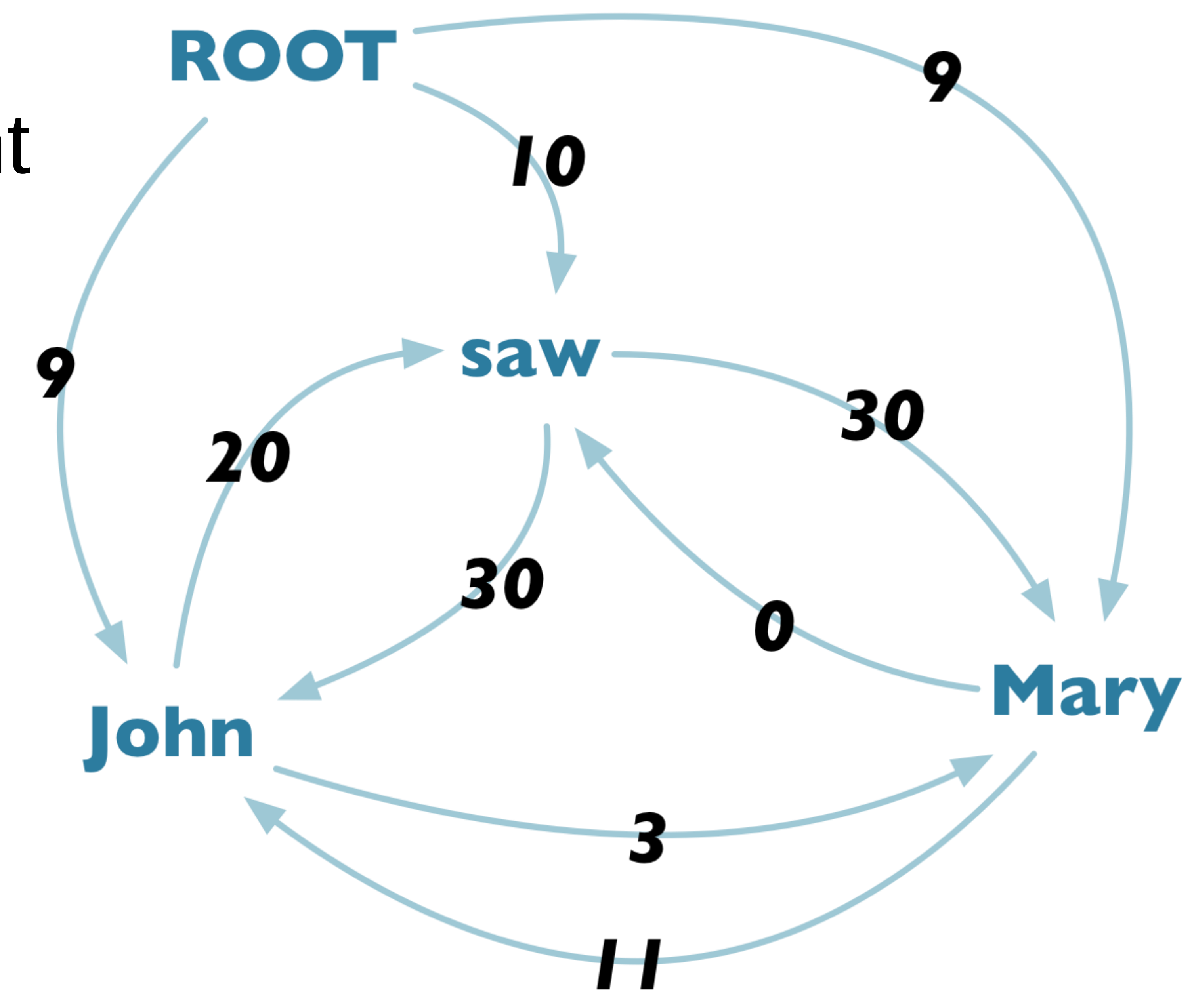
Initial Graph: *(McDonald et al, 2005b)*

- *John saw Mary*
  - All words connected: ROOT only has outgoing arcs
- Goal: Remove arcs to create a tree covering all words
  - Resulting tree is parse



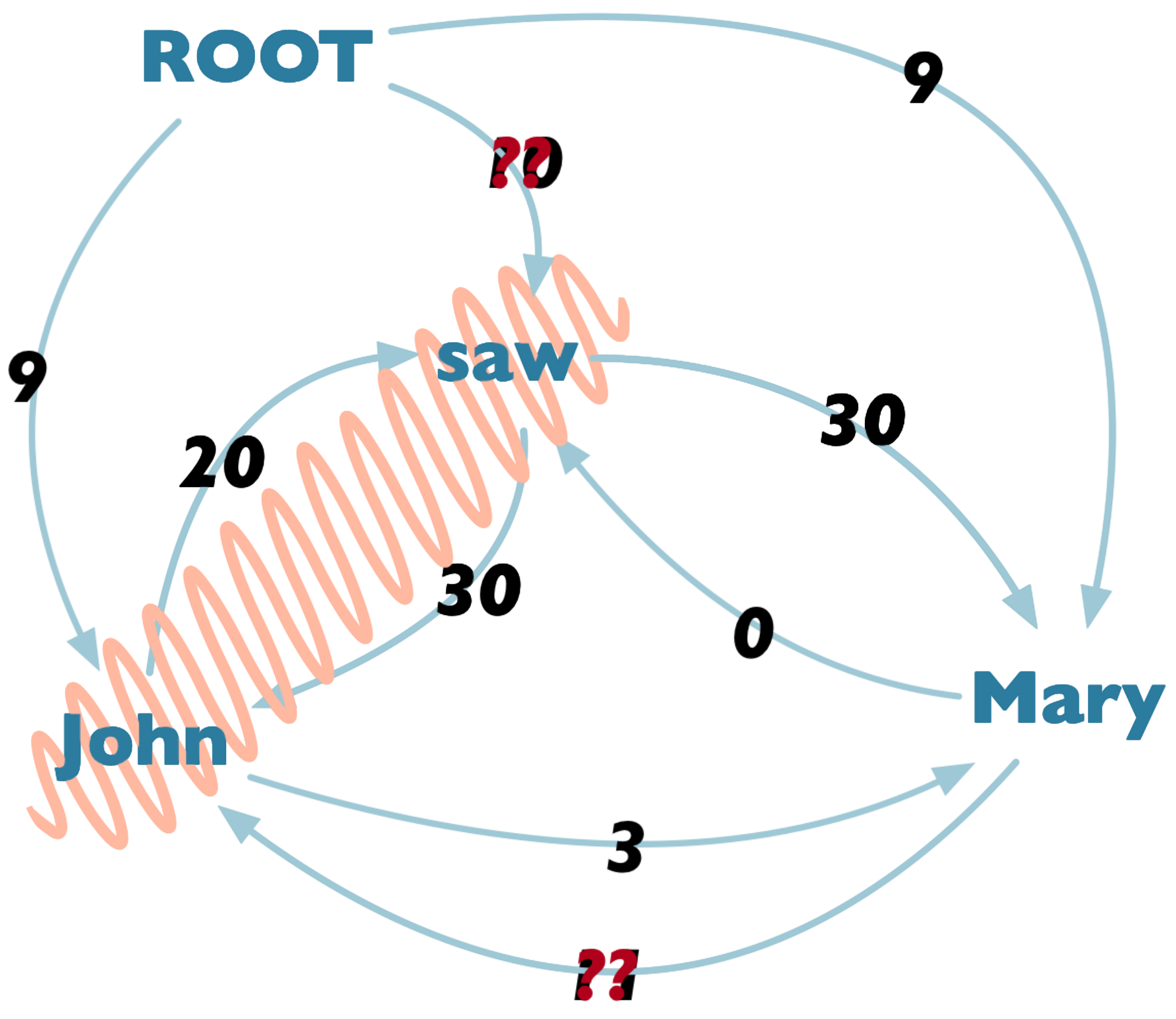
Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
- Sketch of algorithm:
  - For each node, greedily select incoming arc with max weight
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - “Contract” the cycle: Treat it as a single vertex
    - Recalculate weights into/out of the new vertex
    - Recursively do MST algorithm on resulting graph
- Running time: naïve: O(n^3); Tarjan: O(n^2)
  - Applicable to non-projective graphs



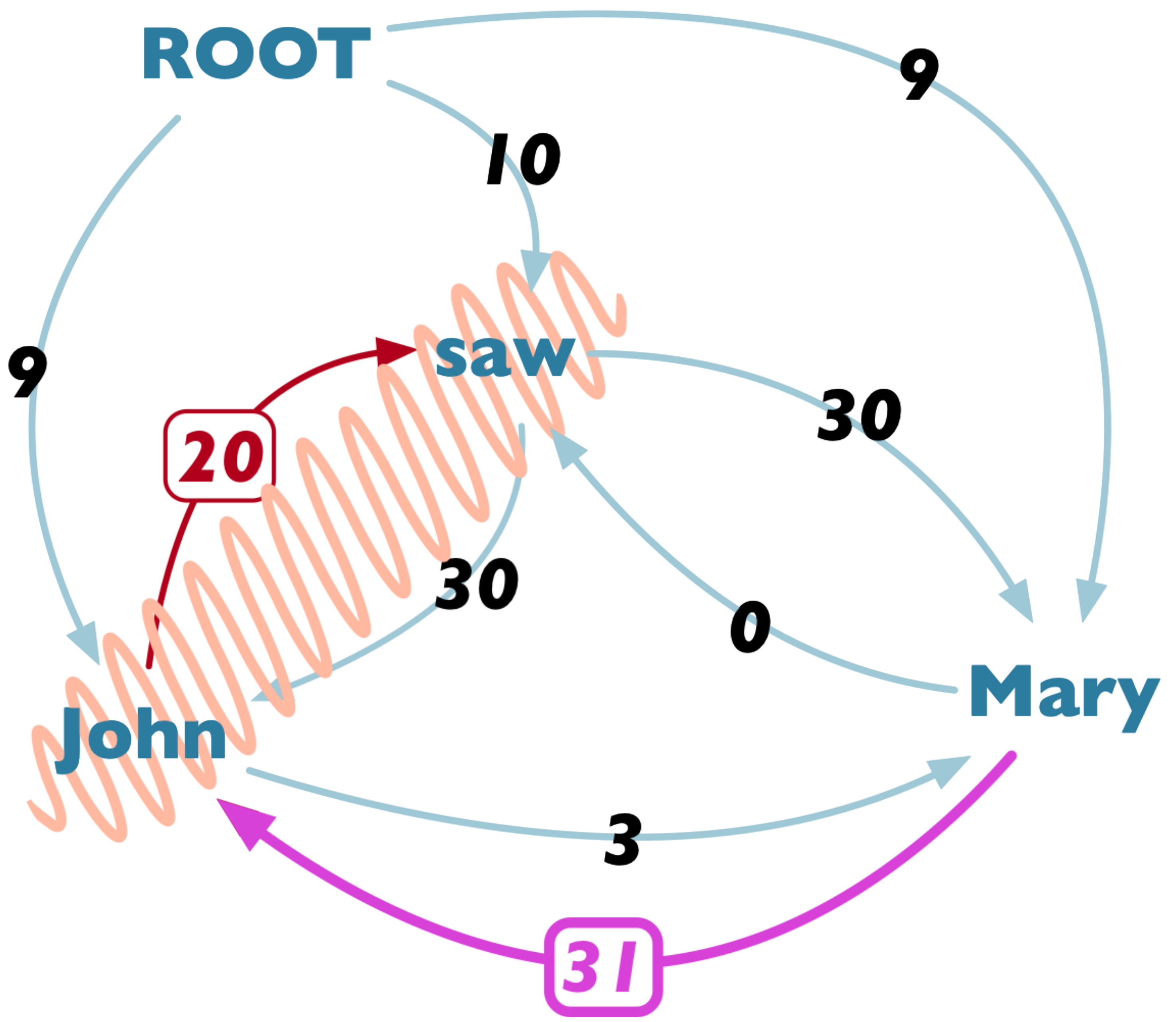
Step 1 & 2

- Find, for each word, the highest scoring incoming edge.
- Is it a tree?
  - No, there's a cycle.
- Collapse the cycle
- And re-examine the edges again



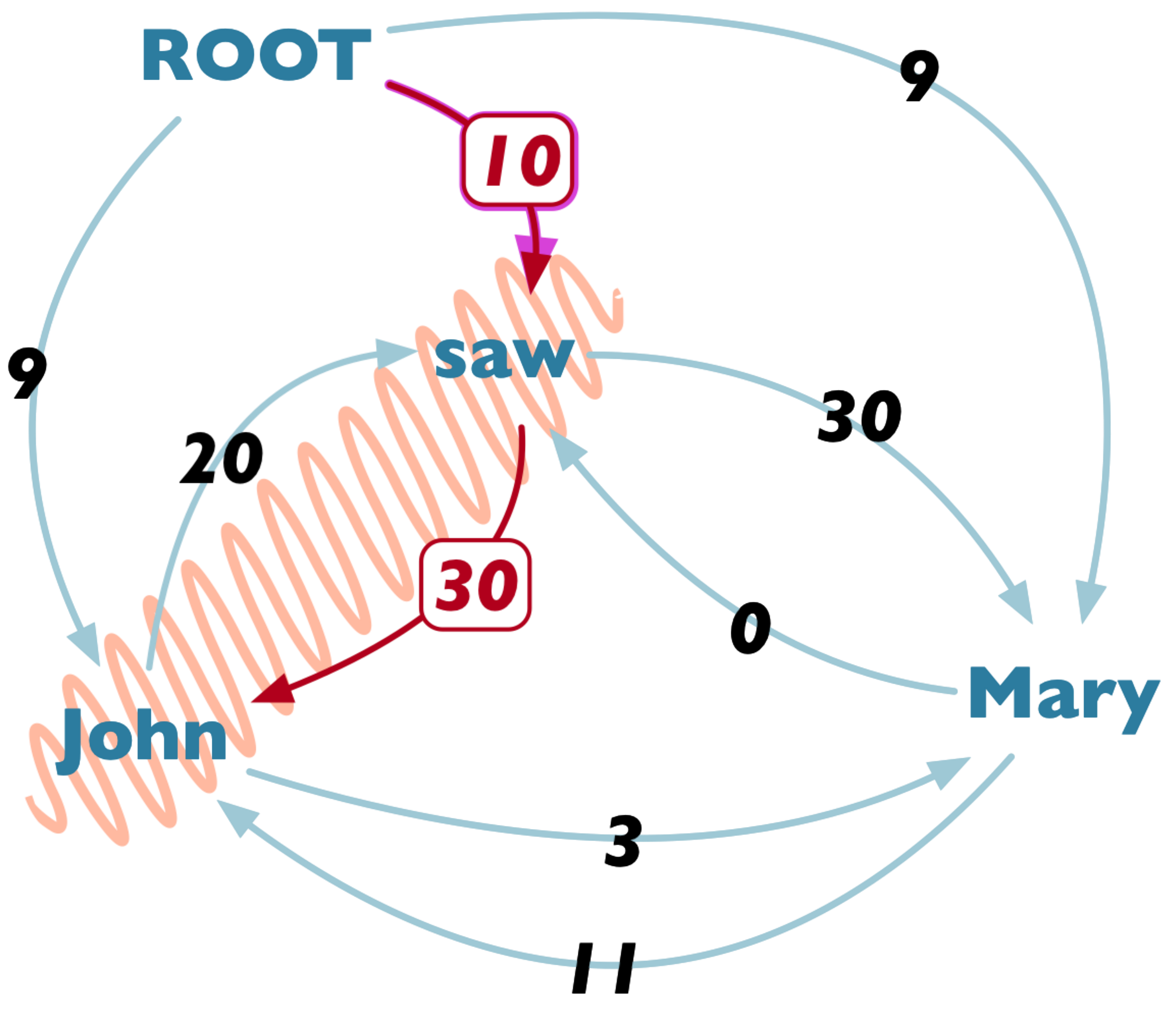
Calculating Weights for Collapsed Vertex

s(Mary, *C*) 11 + 20 = 31



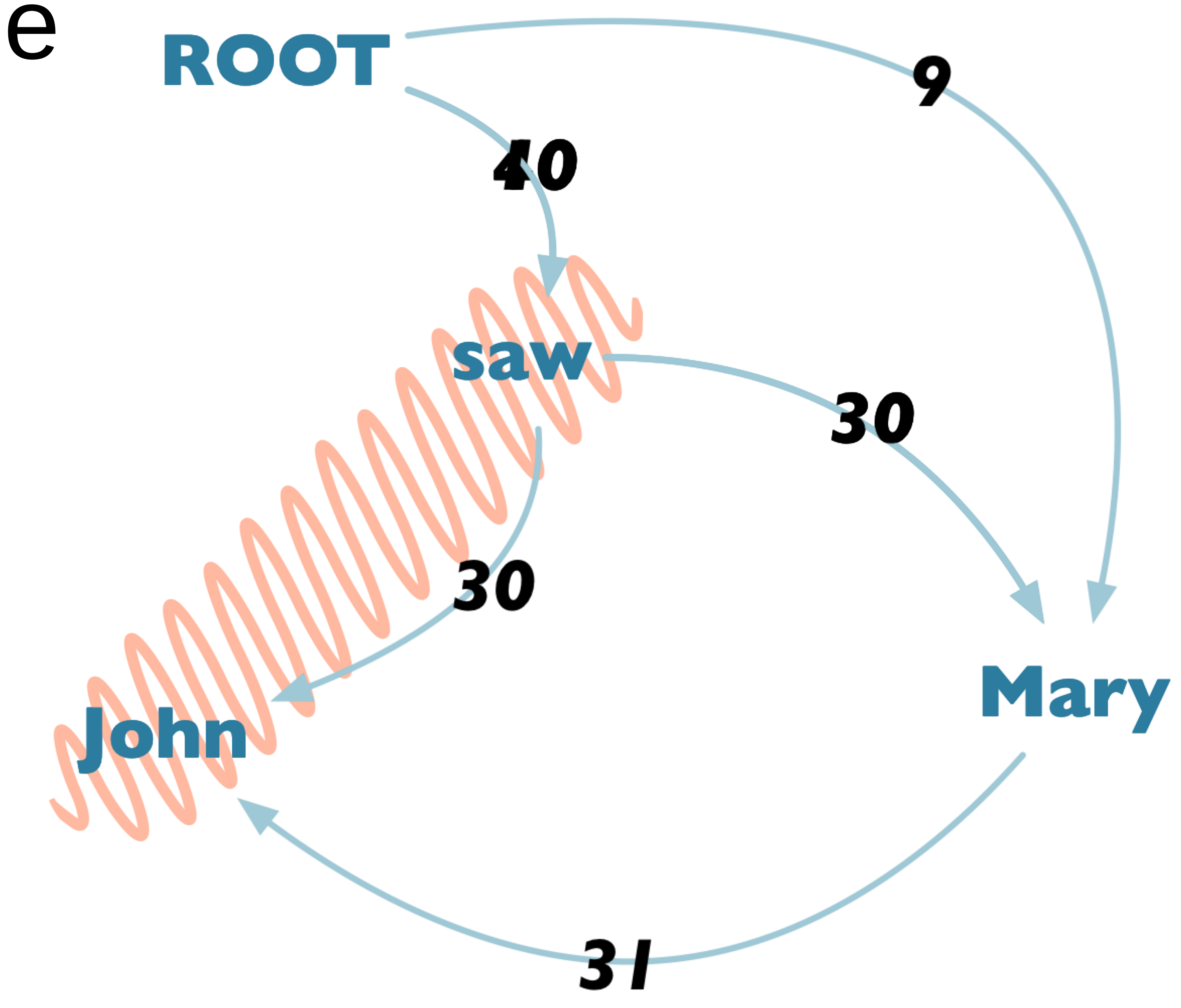
Calculating Weights for Collapsed Vertex

s(ROOT, *C*) 10 + 30 = 40



Step 3

- With cycle collapsed, recurse on step 1:
- Keep highest weighted incoming edge for each edge
- Is it a tree?
  - **Yes!**
  - …but must recover collapsed portions.



MST Algorithm

```
function MaxSpanningTree(G=(V,E), root, score) returns spanning tree

  F ← []
  T' ← []
  score' ← []
  for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
      score'[e] ← score[e] − score[bestInEdge]

    if T=(V,F) is a spanning tree then return it
    else
      C ← a cycle in F
      G' ← Contract(G, C)
      T' ← MaxSpanningTree(G', root, score')
      T ← Expand(T', C)
      return T

function Contract(G, C) returns contracted graph

function Expand(T, C) returns expanded graph
```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

Learning Weights

- Weights for arc-factored model learned from dependency treebank
  - Weights learned for tuple **(w_i, w_j, l)**
- McDonald et al, 2005a employed discriminative ML
  - MIRA (Crammer and Singer, 2003)
- Operates on vector of local features

Features for Learning Weights

- Simple categorical features for (w_i, L, w_j) including:
  - Identity of w_i (or char 5-gram prefix), POS of w_i
  - Identity of w_j (or char 5-gram prefix), POS of w_j
  - Label of L, direction of L
  - Number of words between w_i, w_j
  - POS tag of w_{i-1}, POS tag of w_{i+1}
  - POS tag of w_{j-1}, POS tag of w_{j+1}

- Features conjoined with direction of attachment and distance between words

Neural Graph-based Parsing

- Instead of hand-engineered features, let a neural network learn which features matter!
  - Same algorithm, but scores for arcs from NN

Simple and Accurate Dependency Parsing
Using Bidirectional LSTM Feature Representations

Eliyahu Kiperwasser
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
elikip@gmail.com

Yoav Goldberg
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
yoav.goldberg@gmail.com

Abstract

We present a simple and effective scheme for dependency parsing which is based on bidirectional-LSTMs (BiLSTMs). Each sentence token is associated with a BiLSTM vec-

arc-factored (first order) models (McDonald, 2006), in which the scoring function for a tree decomposes over the individual arcs of the tree. More elaborate models look at larger (overlapping) parts, requiring more sophisticated inference and training algorithms

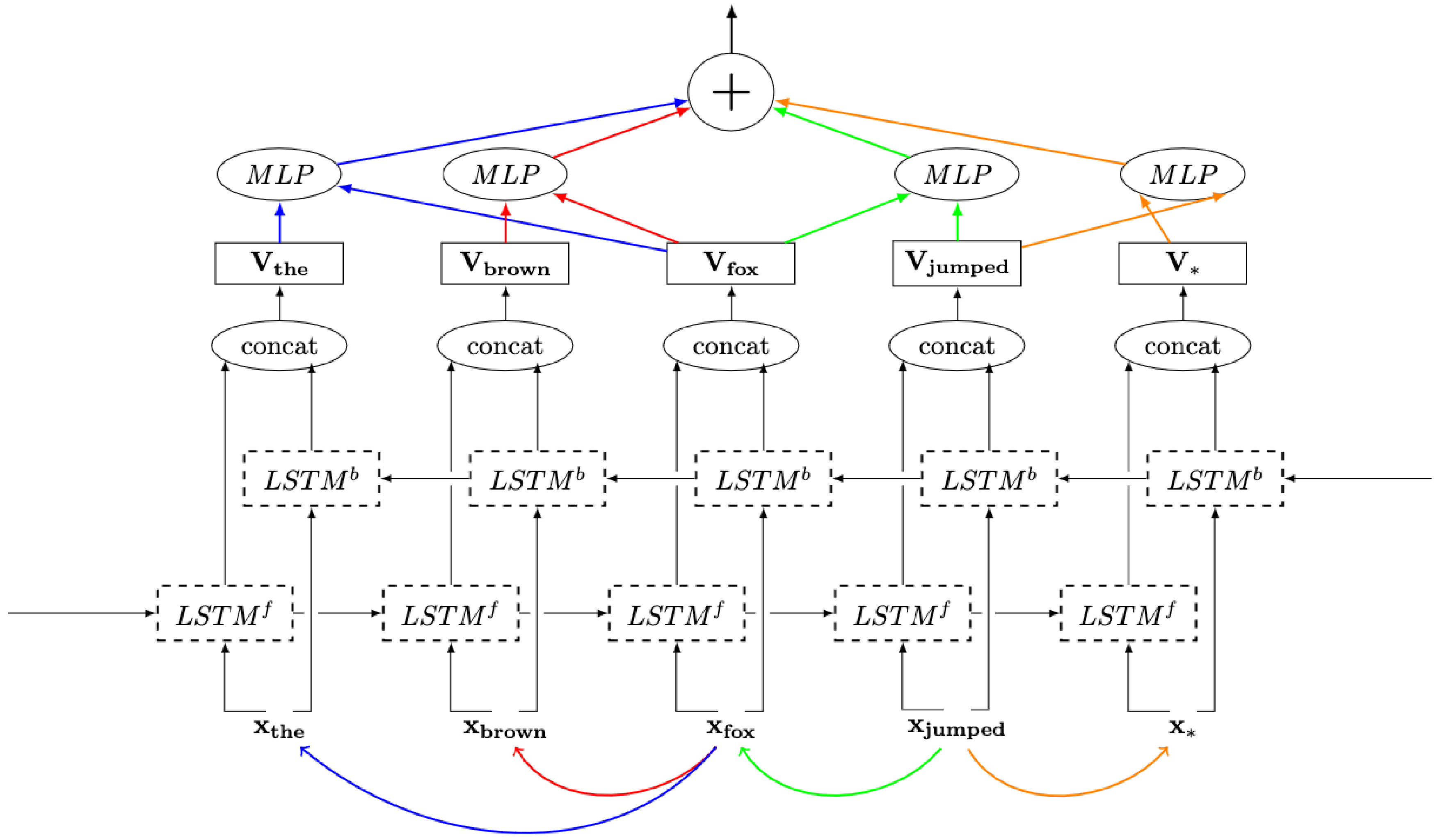


Figure 2: Illustration of the neural model scheme of the graph-based parser when calculating the score of a given parse tree. The parse tree is depicted below the sentence. Each dependency arc in the sentence is scored using an MLP that is fed the BiLSTM encoding of the words at the arc's end points (the colors of the arcs correspond to colors of the MLP inputs above), and the individual arc scores are summed to produce the final score. All the MLPs share the same parameters. The figure depicts a single-layer BiLSTM, while in practice we use two layers. When parsing a sentence, we compute scores for all possible n^2 arcs, and find the best scoring tree using a dynamic-programming algorithm.

https://aclanthology.org/Q16-1023/

Dependency Parsing

- Dependency Grammars:
  - Compactly represent predicate–argument structure
  - Lexicalized, localized
  - Natural handling of flexible word order
- Dependency parsing:
  - Conversion to phrase structure trees
  - Graph-based parsing (MST), efficient non-proj $\mathbf{O}(n^2)$
  - Next time: *Transition-based parsing*

Further Reading

- Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 91–98. May. **[link]**
- Ryan McDonald, Fernando Pereira, K. Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics. **[link]**
- Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan & Claypool. **[link]**
- Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics*, pages 340–345. Association for Computational Linguistics. **[link]**
- Michael Collins. 1999. *Head-Driven Statistical Models For Natural Language Parsing*. **[link]**
- Kiperwasser and Golberg 2016, "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations", *Transactions of the ACL.*