Dependency Grammars and Parser

LING 571 — Deep Processing for NLP
October 19, 2022
Shane Steinert-Threlkeld

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

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 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:

Dependency Parse

- A Dependency parse is a tree,* where:
  - Nodes correspond to words in string

Dependency Parse

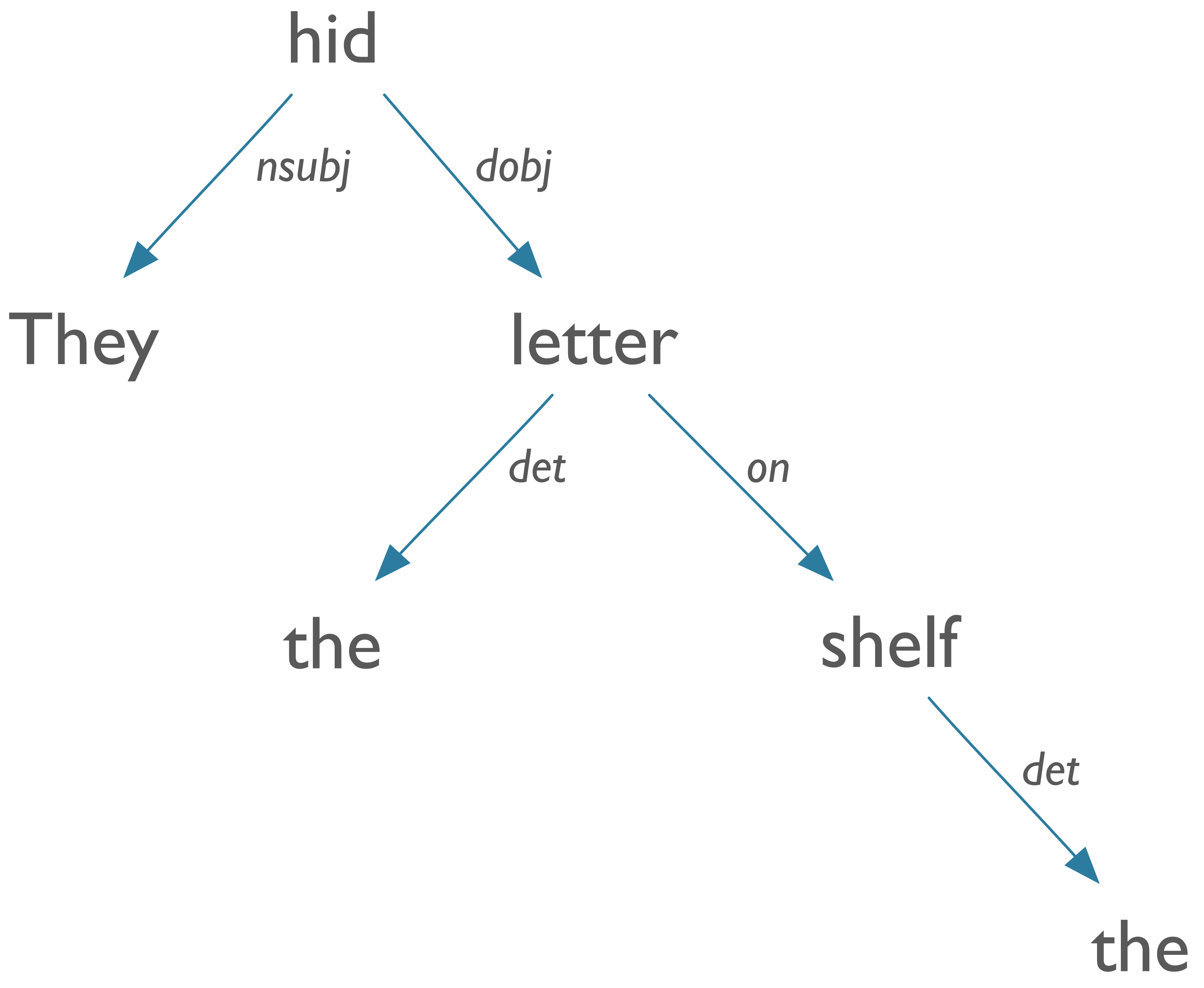
- A Dependency parse is a tree,* where:
  - Nodes correspond to words in string
  - Edges between nodes represent dependency relations
    - Relations may or may not be labeled (aka typed)
  - *: in very special cases, can argue for cycles

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

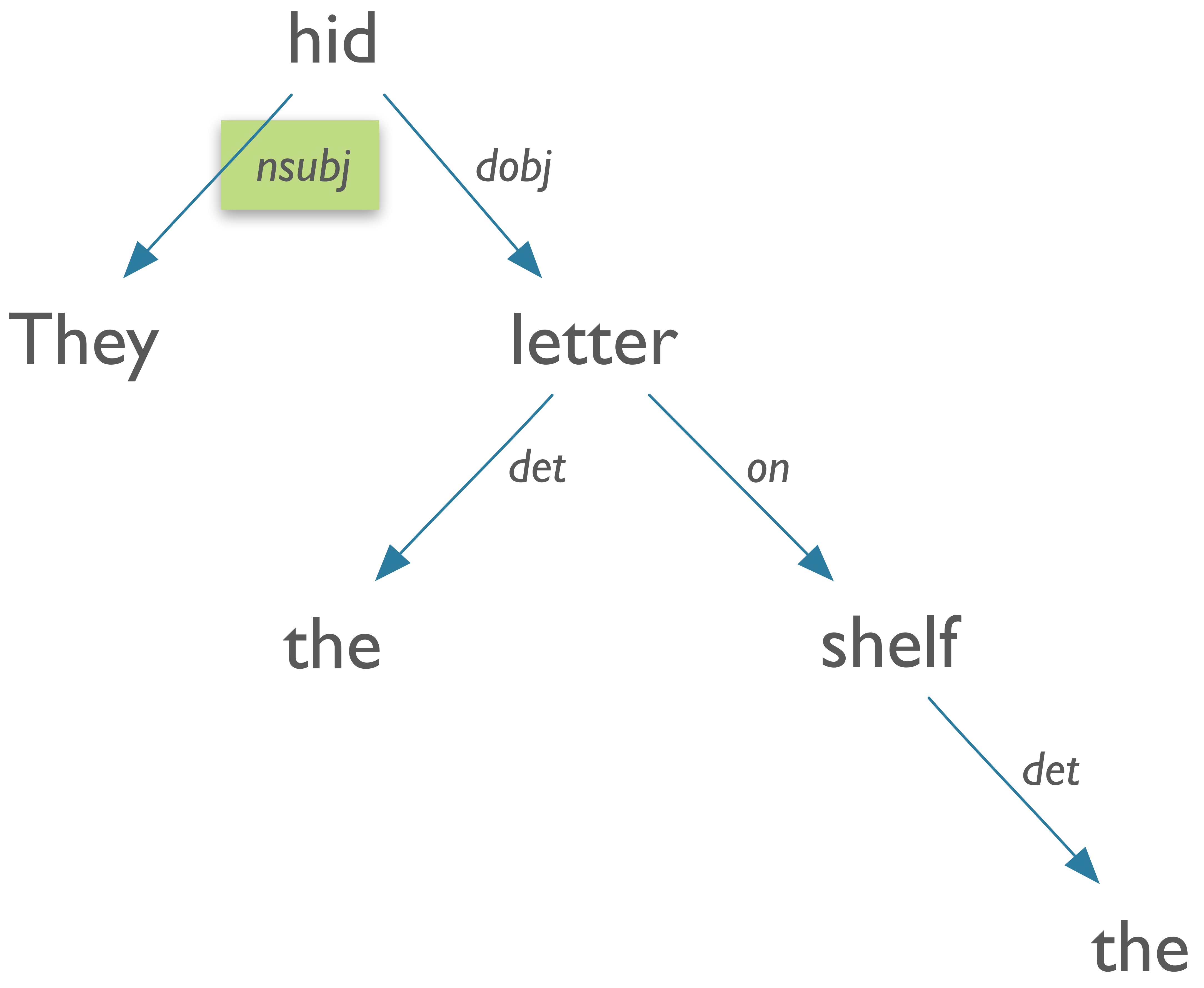


Dependency Parse Example:

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

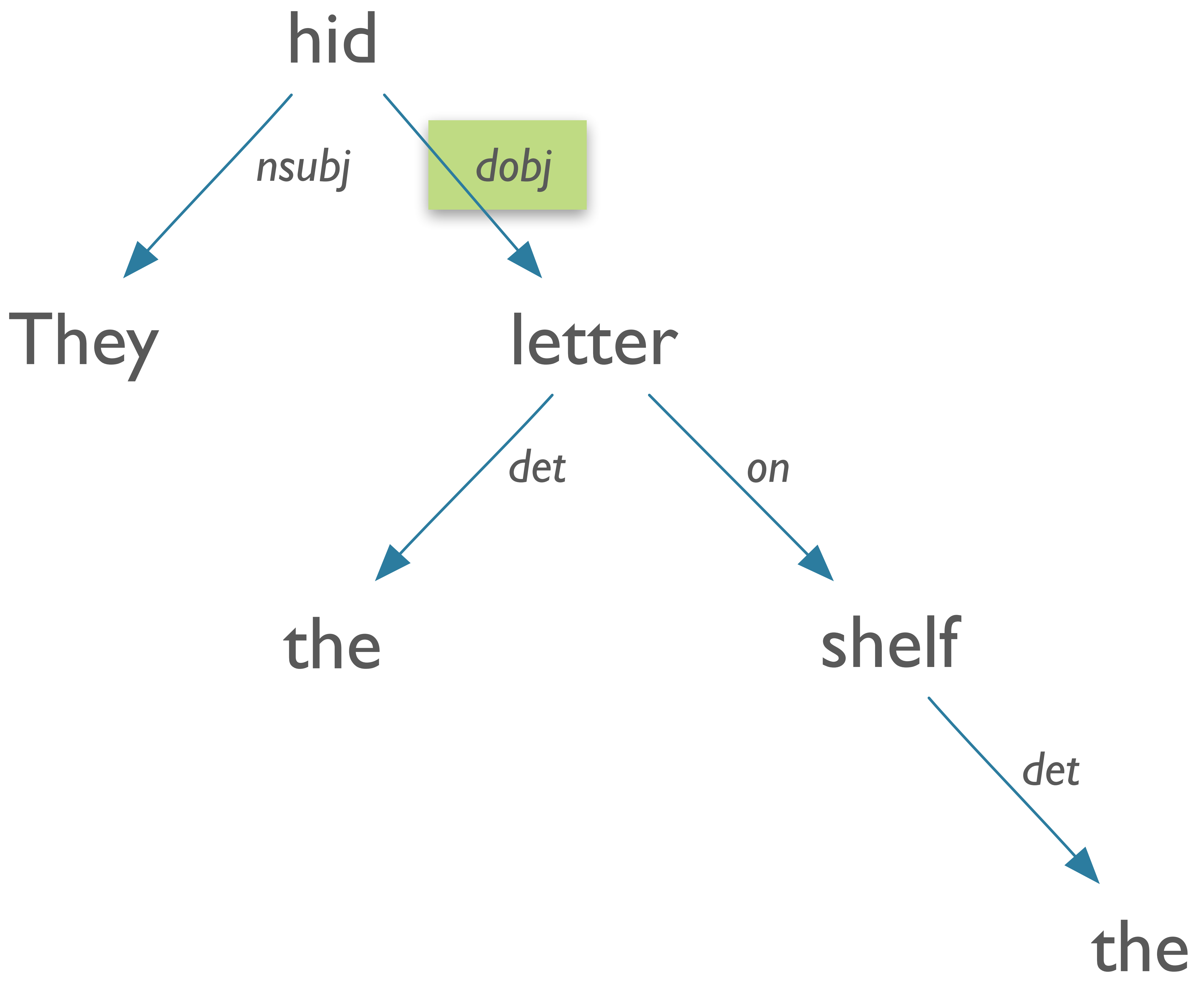


Dependency Parse Example:

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

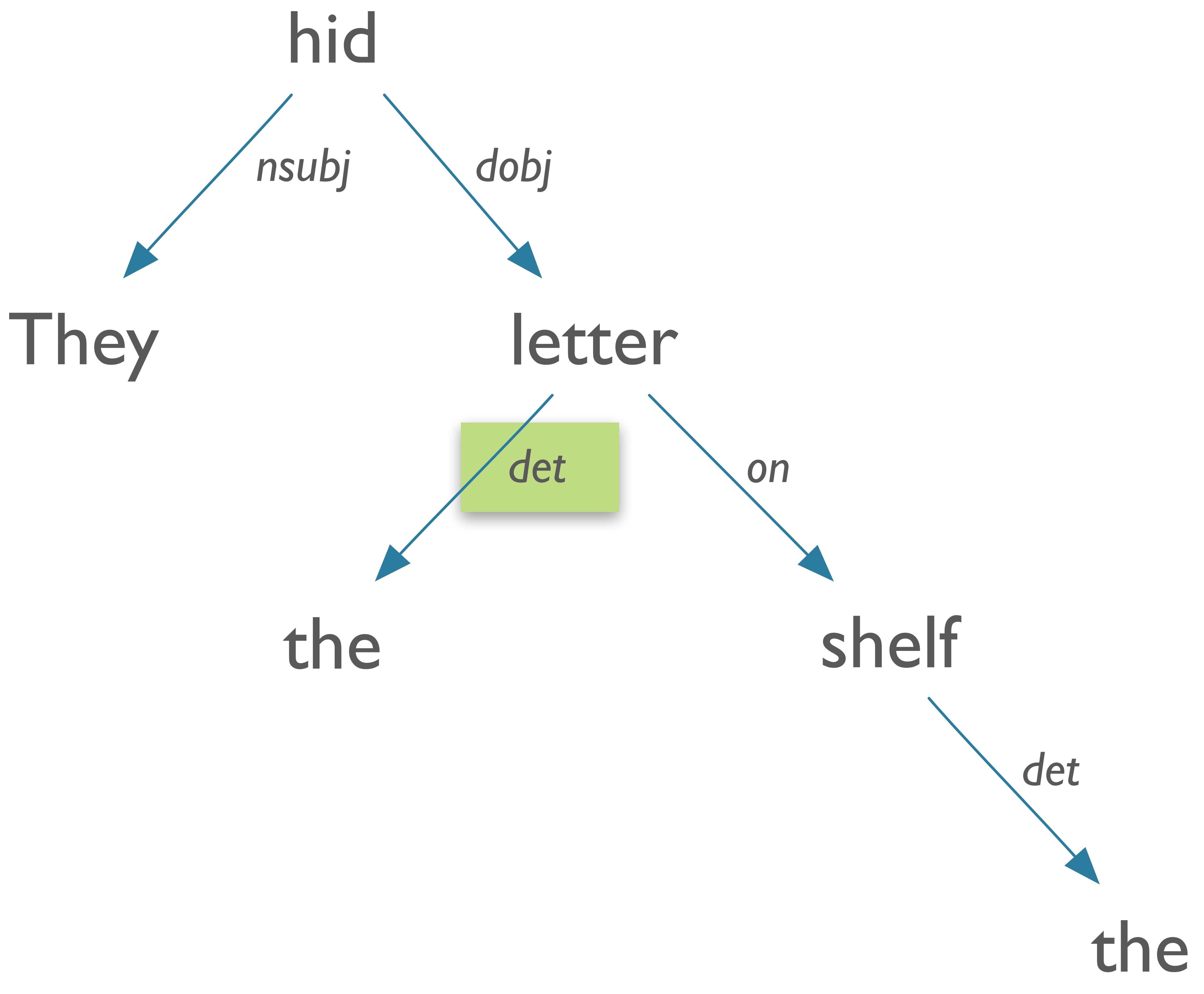


Dependency Parse Example:

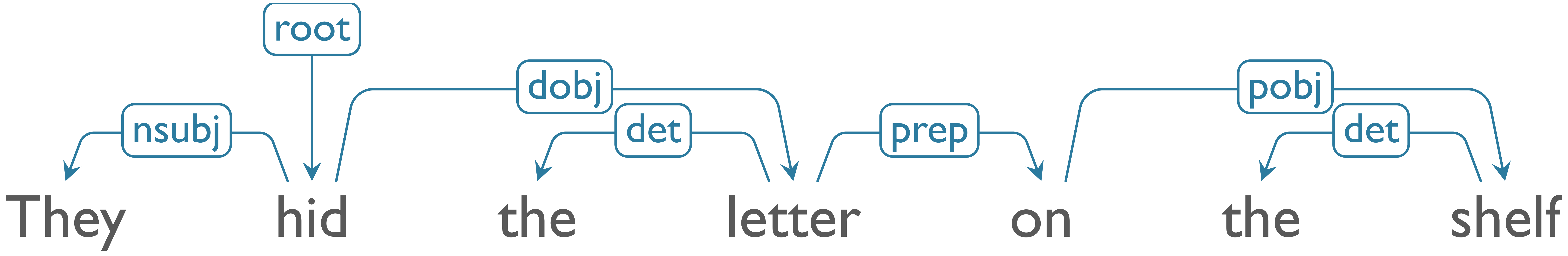
They hid the letter on the shelf

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

Why Dependency Grammar?

- More natural representation for many tasks
  - Clear encapsulation of predicate-argument structure

Why Dependency Grammar?

- More natural representation for many tasks
  - Clear encapsulation of predicate-argument structure
    - Phrase structure may obscure, e.g. *wh-movement*

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

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*?

Why Dependency Grammar?

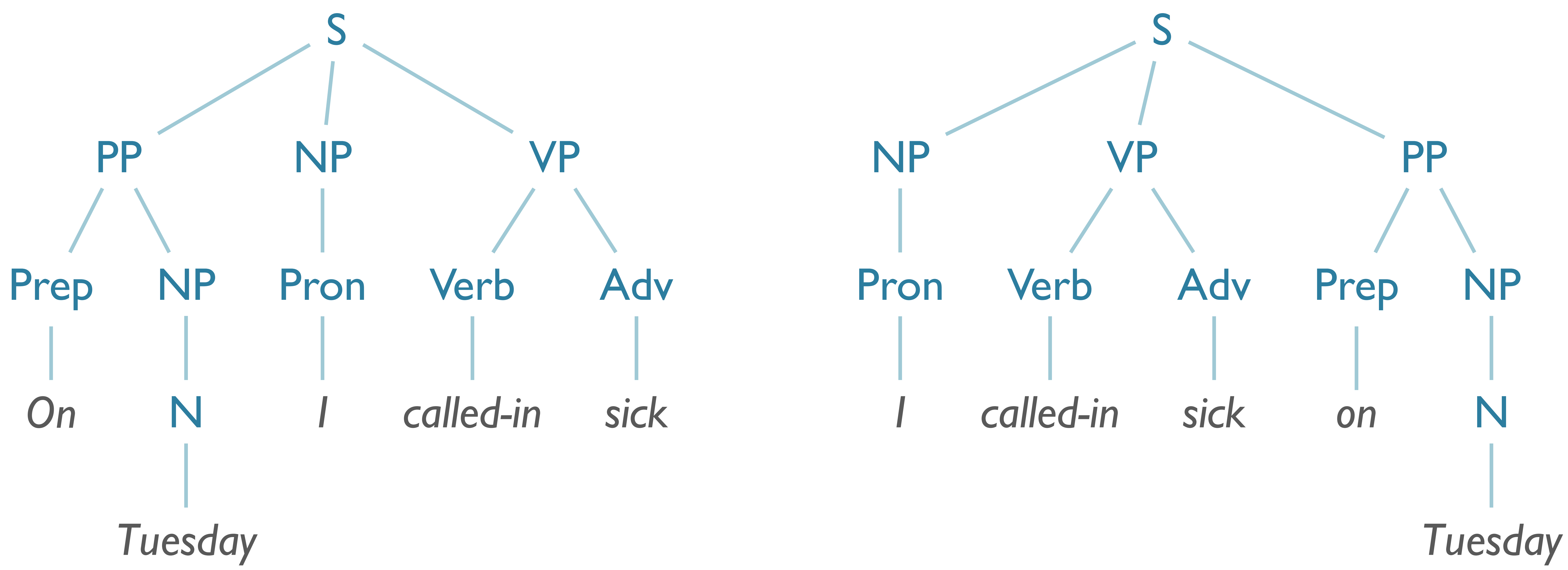
- More natural representation for many tasks
  - Clear encapsulation of predicate-argument structure
    - Phrase structure may obscure, e.g. *wh-movement*
- Good match for question-answering, relation extraction
  - *Who* did *what* to *whom*?
  - = (*Subject*) did (*theme*) to (*patient*)

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**

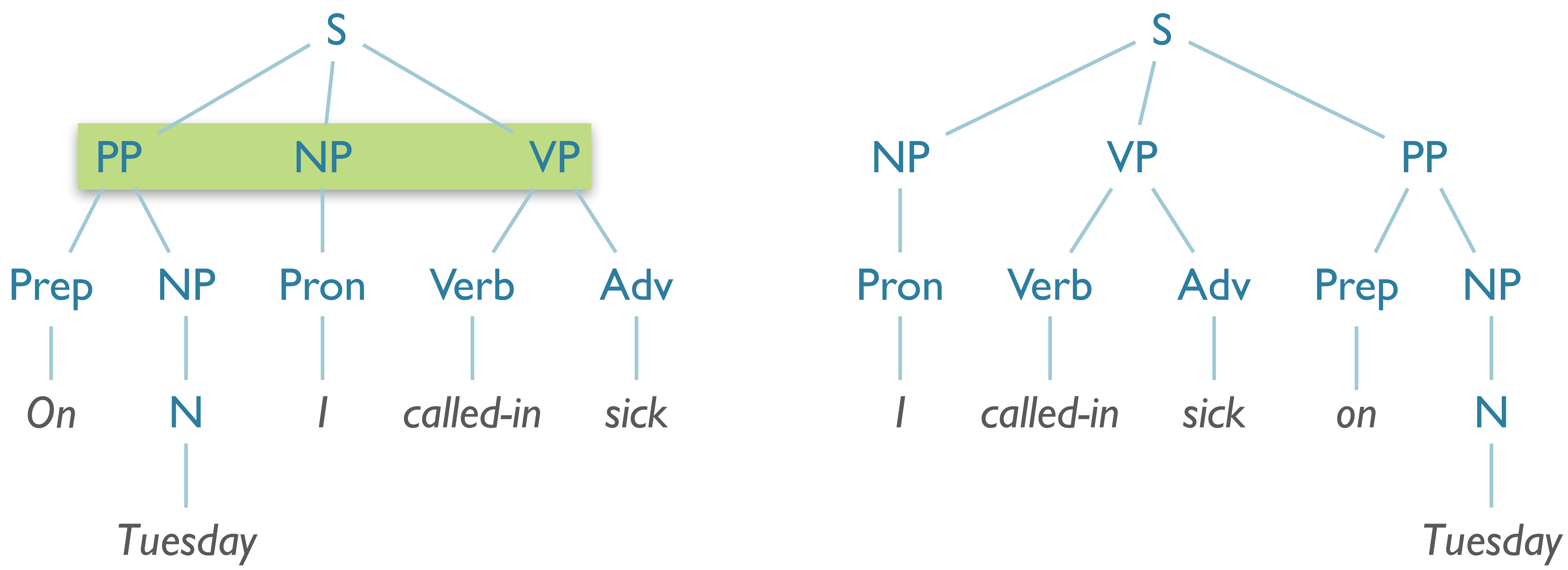
Why Dependency Grammar?

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



Why Dependency Grammar?

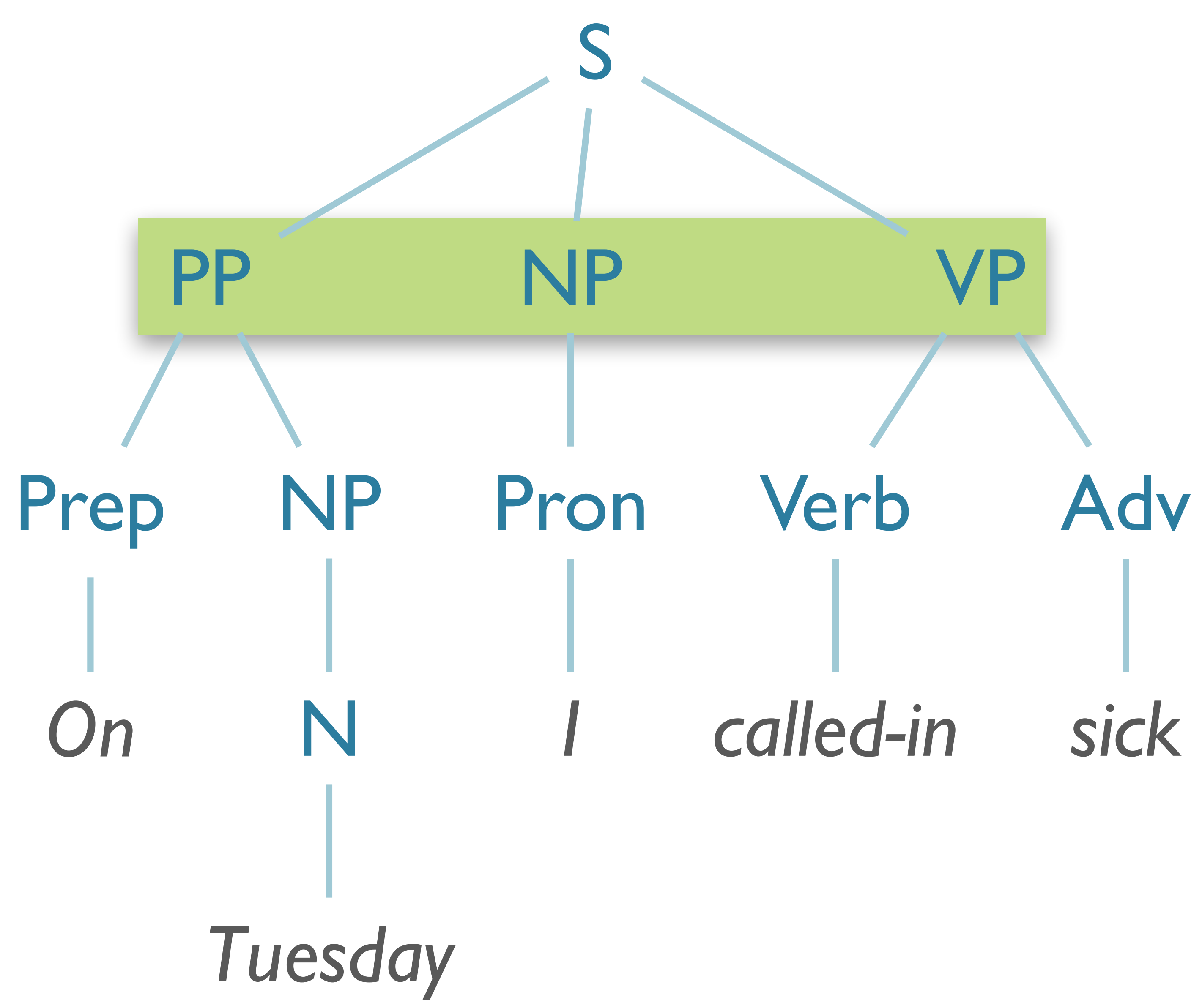
- Easier handling of flexible or free word order
- How does CFG handle variation in word order?

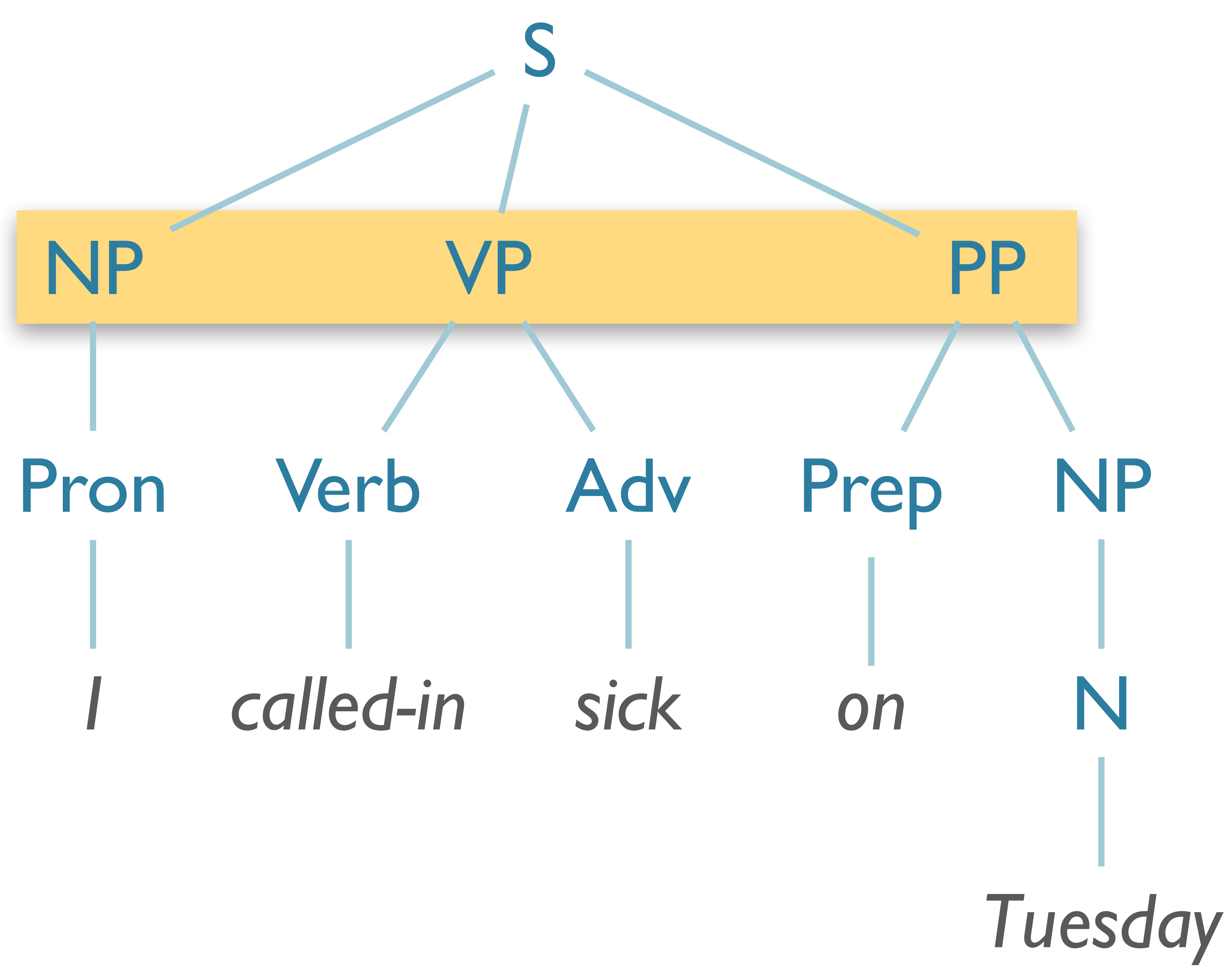


$$S \rightarrow \boxed{PP\ NP\ VP}$$

Why Dependency Grammar?

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

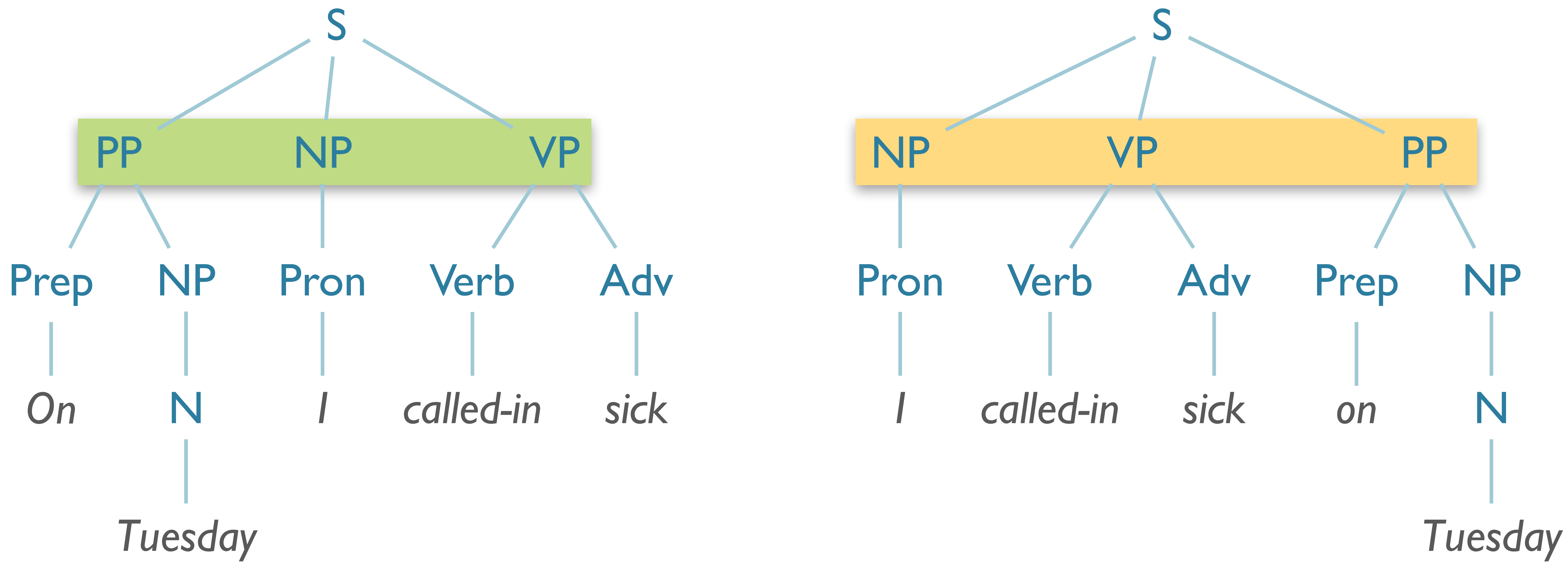




$S \rightarrow PP\ NP\ VP$ $\qquad$ $S \rightarrow NP\ VP\ PP$

Why Dependency Grammar?

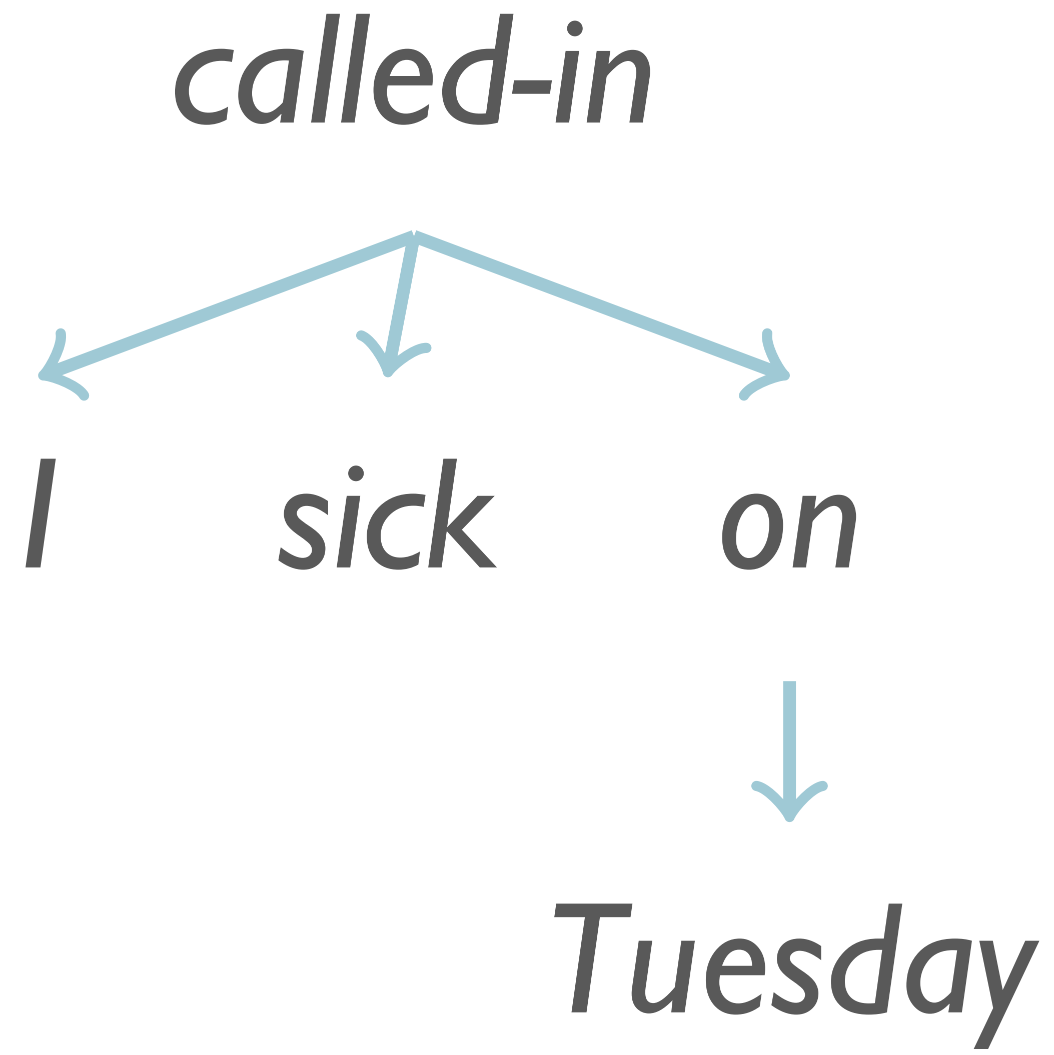
- English has relatively fixed word order
- Big problem for languages with freer word order



$S \rightarrow PP\ NP\ VP$ $\quad$ $S \rightarrow NP\ VP\ PP$

Why Dependency Grammar?

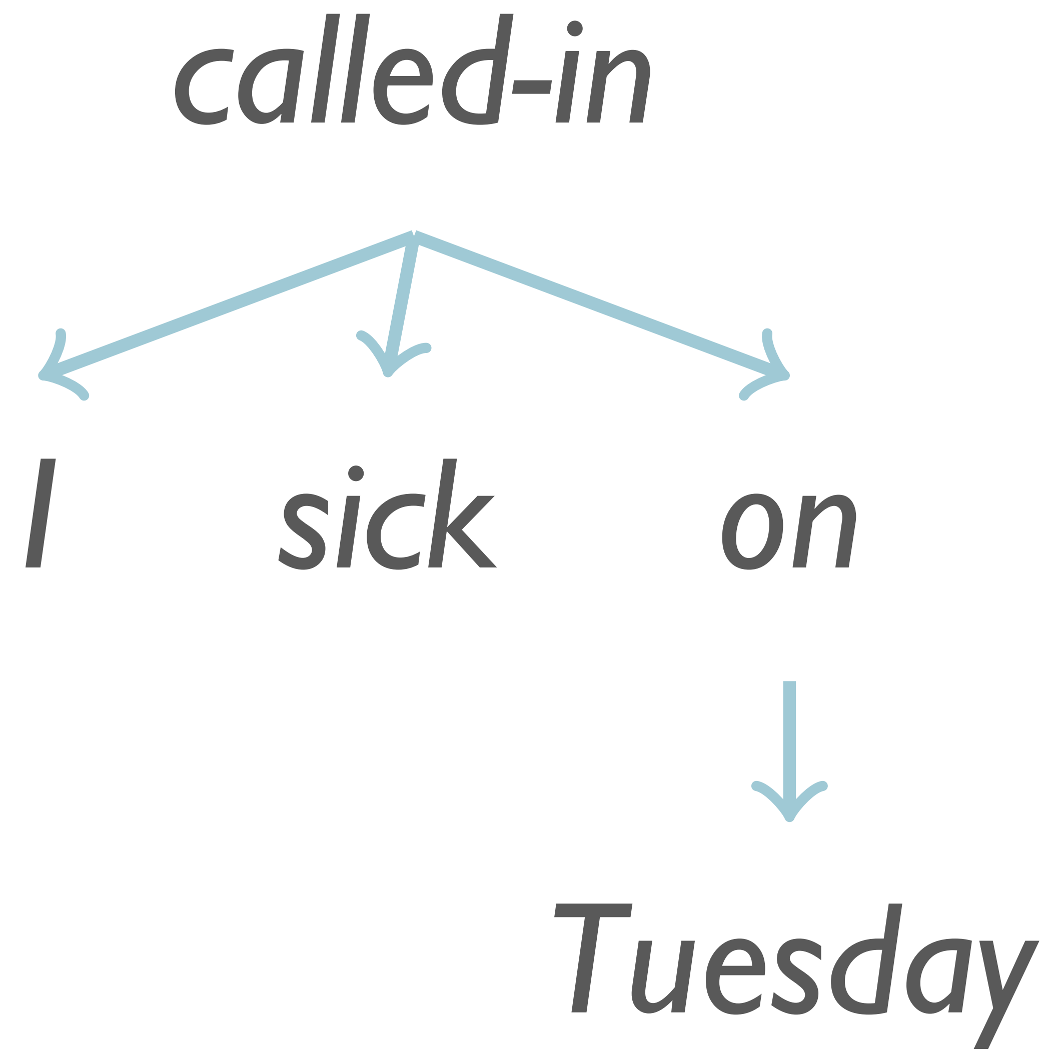
- How do dependency structures represent the difference?



I called in sick on Tuesday

Why Dependency Grammar?

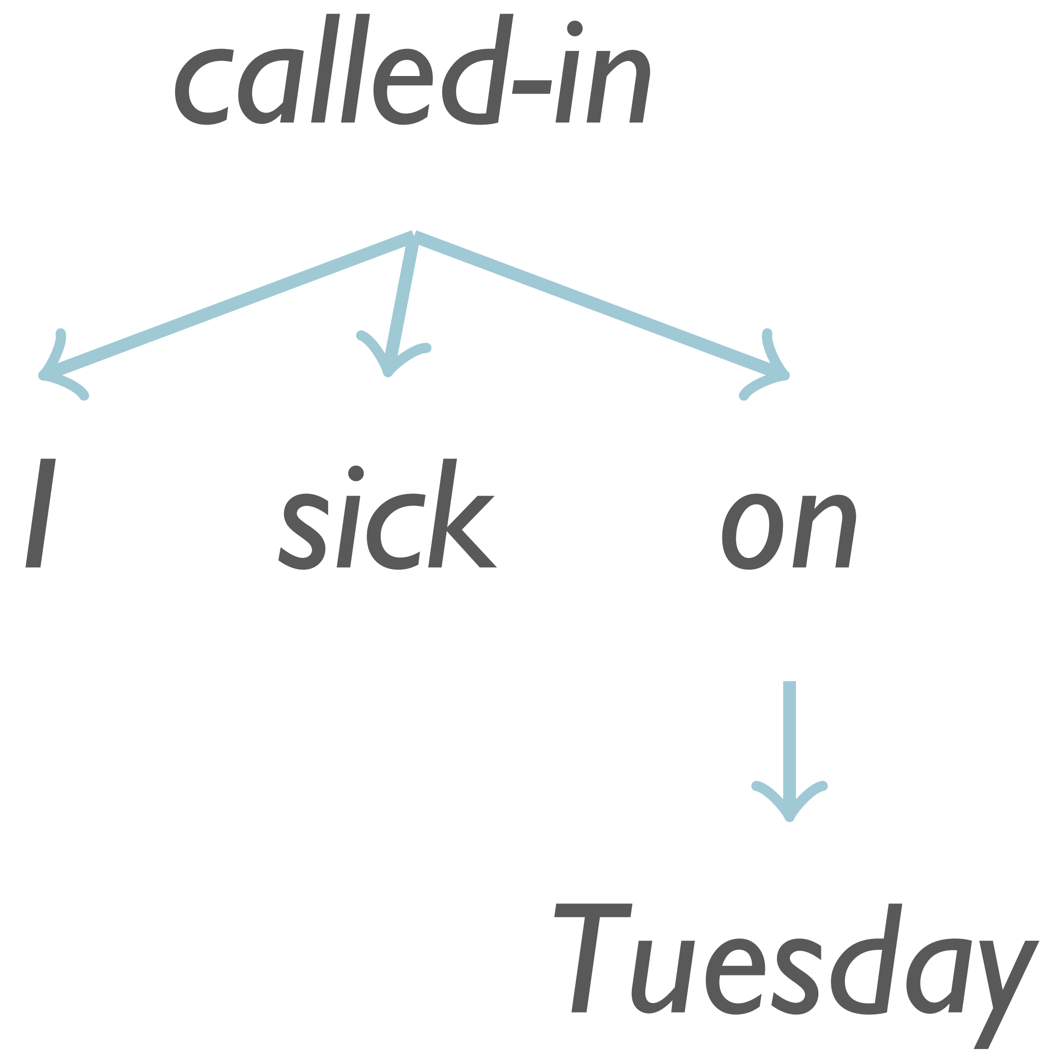
- How do dependency structures represent the difference?
  - Same structure



I called in sick on Tuesday

Why Dependency Grammar?

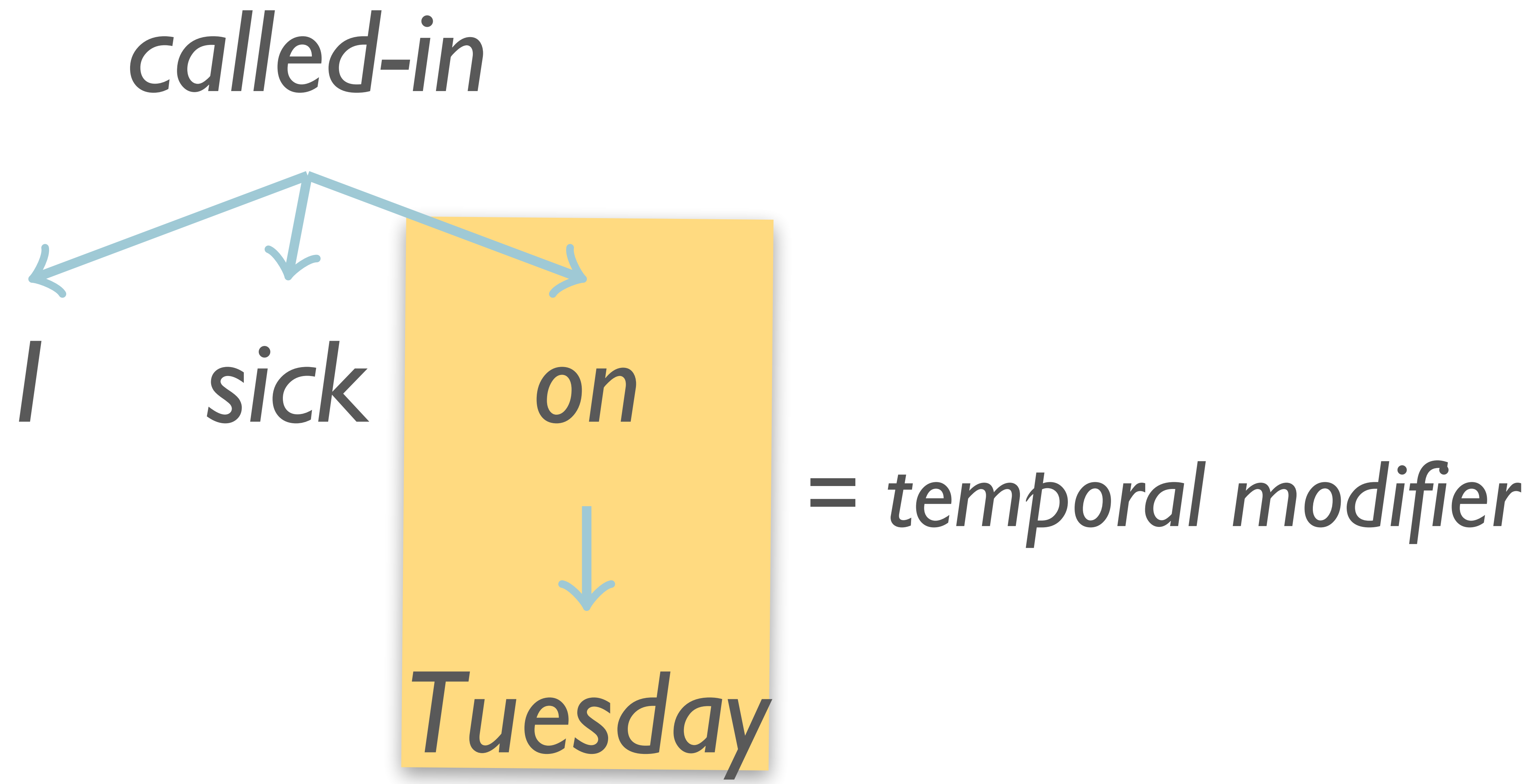
- How do dependency structures represent the difference?
  - Same structure
  - Relationships are between words, order insensitive



I called in sick on Tuesday

Why Dependency Grammar?

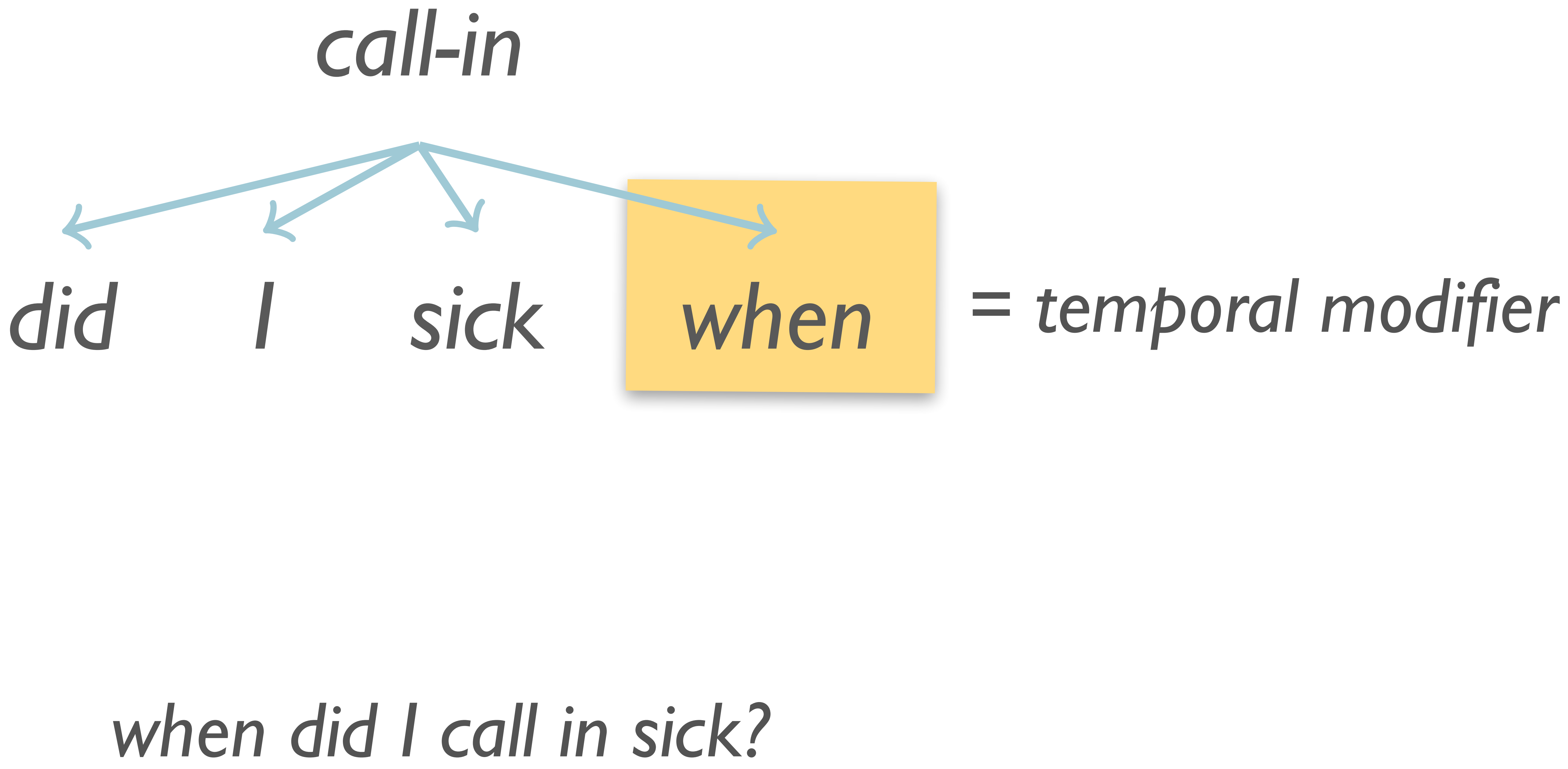
- How do dependency structures represent the difference?
  - Same structure
  - Relationships are between words, order insensitive



I called in sick on Tuesday

Why Dependency Grammar?

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



Natural Efficiencies

- Phrase Structures:
  - Must derive full trees of many non-terminals

Natural Efficiencies

- Phrase Structures:
  - Must derive full trees of many non-terminals
- Dependency Structures:
  - For each word, identify
    - Syntactic head, h
    - Dependency label, d

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
  - [quick live demo]
- LaTeX: tikz-dependency (https://ctan.org/pkg/tikz-dependency)

Resources

- Universal Dependencies:
  - Consistent annotation scheme (i.e. same POS, dependency labels)
  - Treebanks for >70 languages
    - Sizes: German, Czech, Japanese, Russian, French, Arabic, …

Possible Future Extensions

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

Abaza	1	<1K	Northwest Caucasian
Amharic	1	–	Afro-Asiatic, Semitic
Ancient Greek	1	19K	IE, Greek
Archaic Irish	1	–	IE, Celtic
Assamese	1	–	IE, Indic
Bengali	2	–	IE, Indic
Bhojpuri	1	–	IE, Indic
Cappadocian	1	–	IE, Greek
Cusco Quechua	1	–	Quechuan
Czech	1	1,198K	IE, Slavic
Danish	1	–	IE, Germanic
Dargwa	1	–	Nakh-Daghestanian, Lak-Dargwa
English	3	1,209K	IE, Germanic
French	1	–	IE, Romance
Frisian	1	–	IE, Germanic
Georgian	1	–	Kartvelian
Gheg	1	–	IE, Albanian
Greek	1	–	IE, Greek
Gujarati	1	–	IE, Indic
Hiligaynon	1	–	Austronesian, Central Philippine
Icelandic	1	–	IE, Germanic
Irish	1	–	IE, Celtic
Italian	1	–	IE, Romance
Kabyle	1	47K	Afro-Asiatic, Berber
Kannada	1	–	Dravidian, Southern
Khoekhoe	1	–	Khoe-Kwadi
Kiga	1	–	Niger-Congo, Bantoid
Korean	2	–	Korean
Kyrgyz	1	–	Turkic, Northwestern
Ladino	1	–	IE, Romance
Laz	1	2K	Kartvelian
Macedonian	1	–	IE, Slavic
Magahi	2	7K	IE, Indic
Maghrebi Arabic French	1	–	Code switching
Mandyali	1	–	IE, Indic
Marathi	1	205K	IE, Indic

Summary

- Dependency grammars balance complexity and expressiveness
  - Sufficiently expressive to capture predicate-argument structure
  - Sufficiently constrained to allow efficient parsing

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

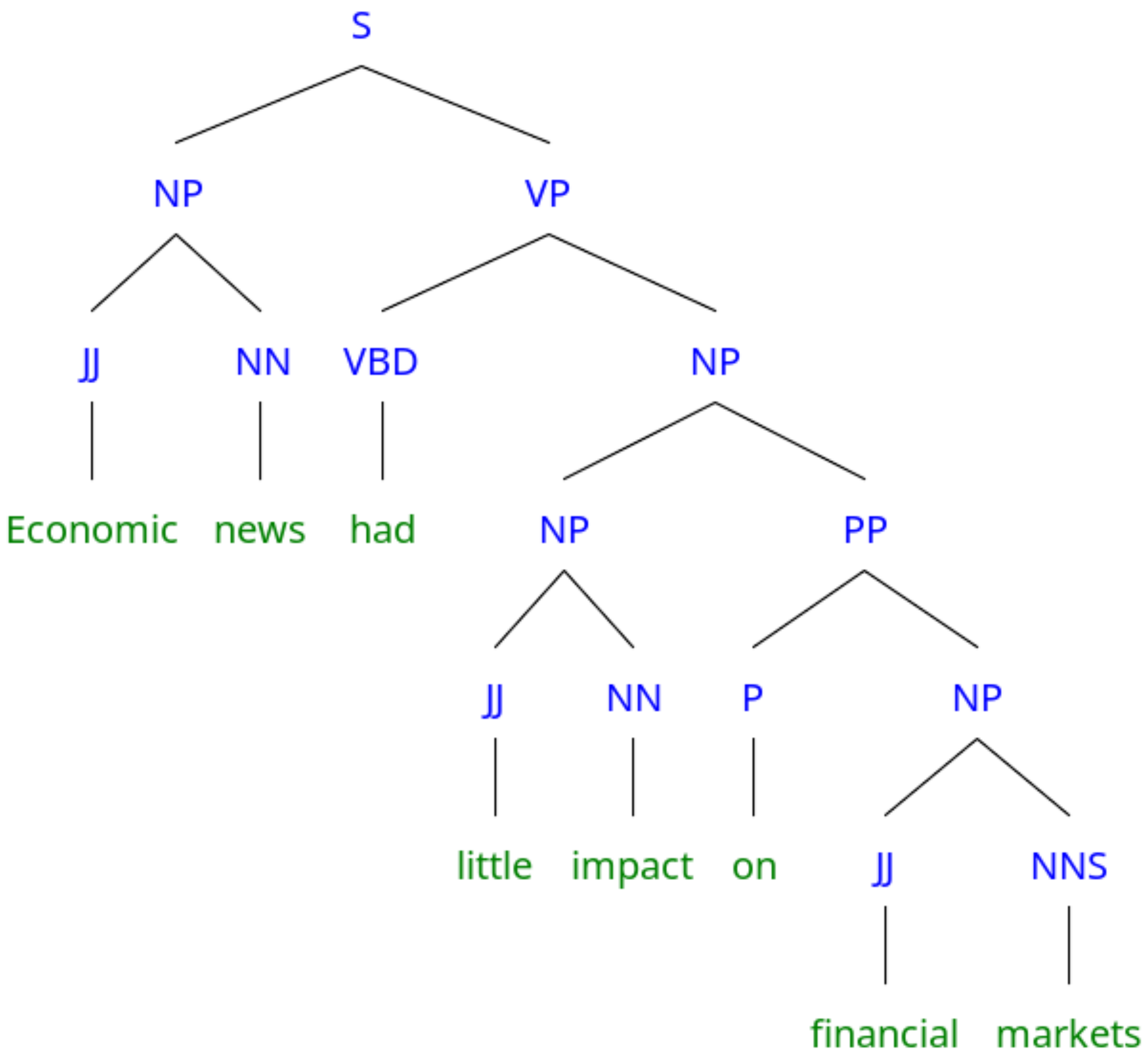
Conversion: PS → DS

- Can convert Phrase Structure (PS) to Dependency Structure (DS)
  - …without the dependency labels

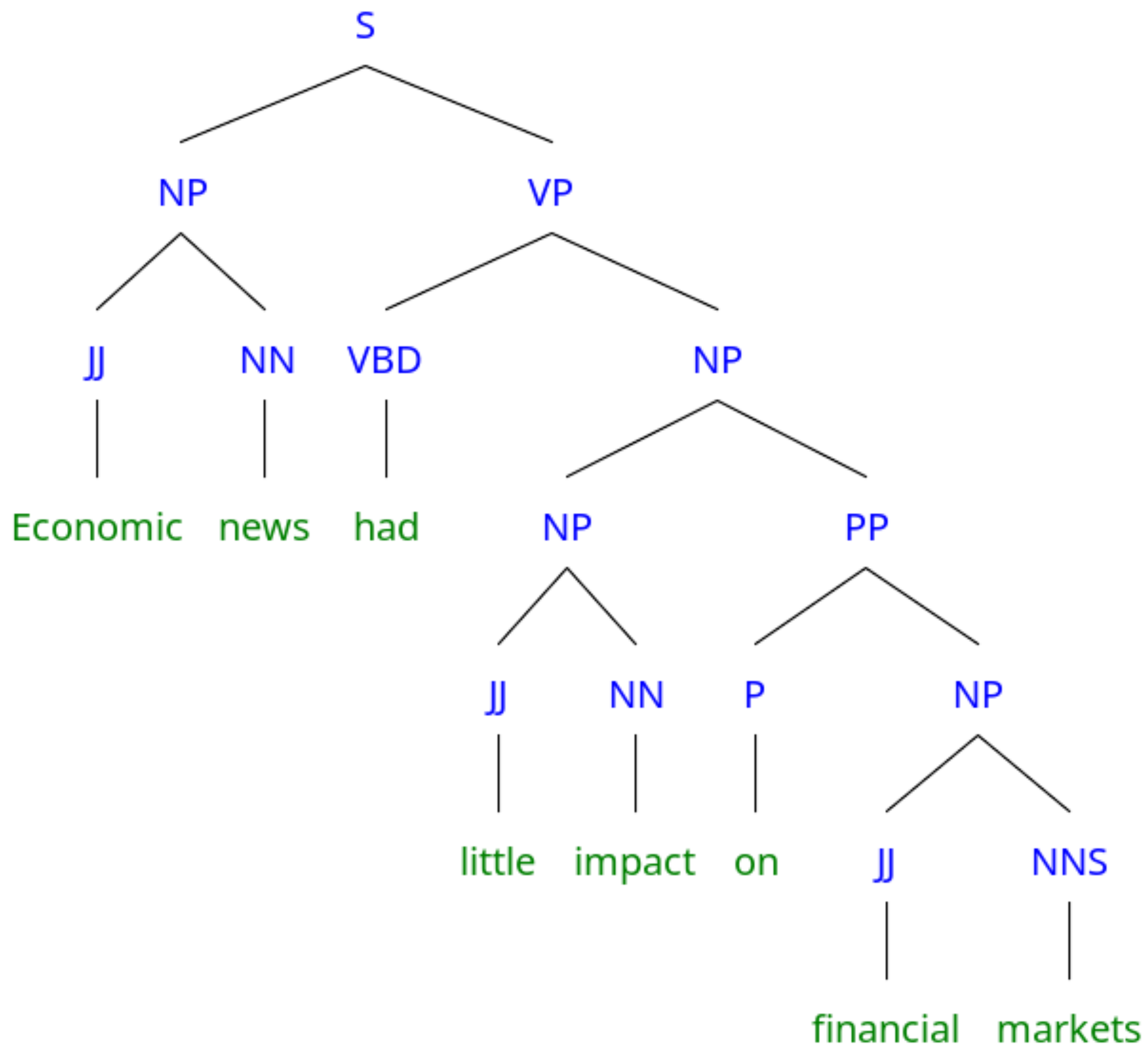
Conversion: PS → DS

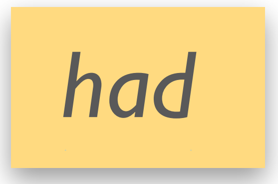
- Can convert Phrase Structure (PS) to Dependency Structure (DS)
  - …without the dependency labels
- Algorithm:
  - Identify all head children in PS
  - Make head of each non-head-child depend on head of head-child
  - Use a *head percolation* table to determine headedness

Conversion: PS → DS

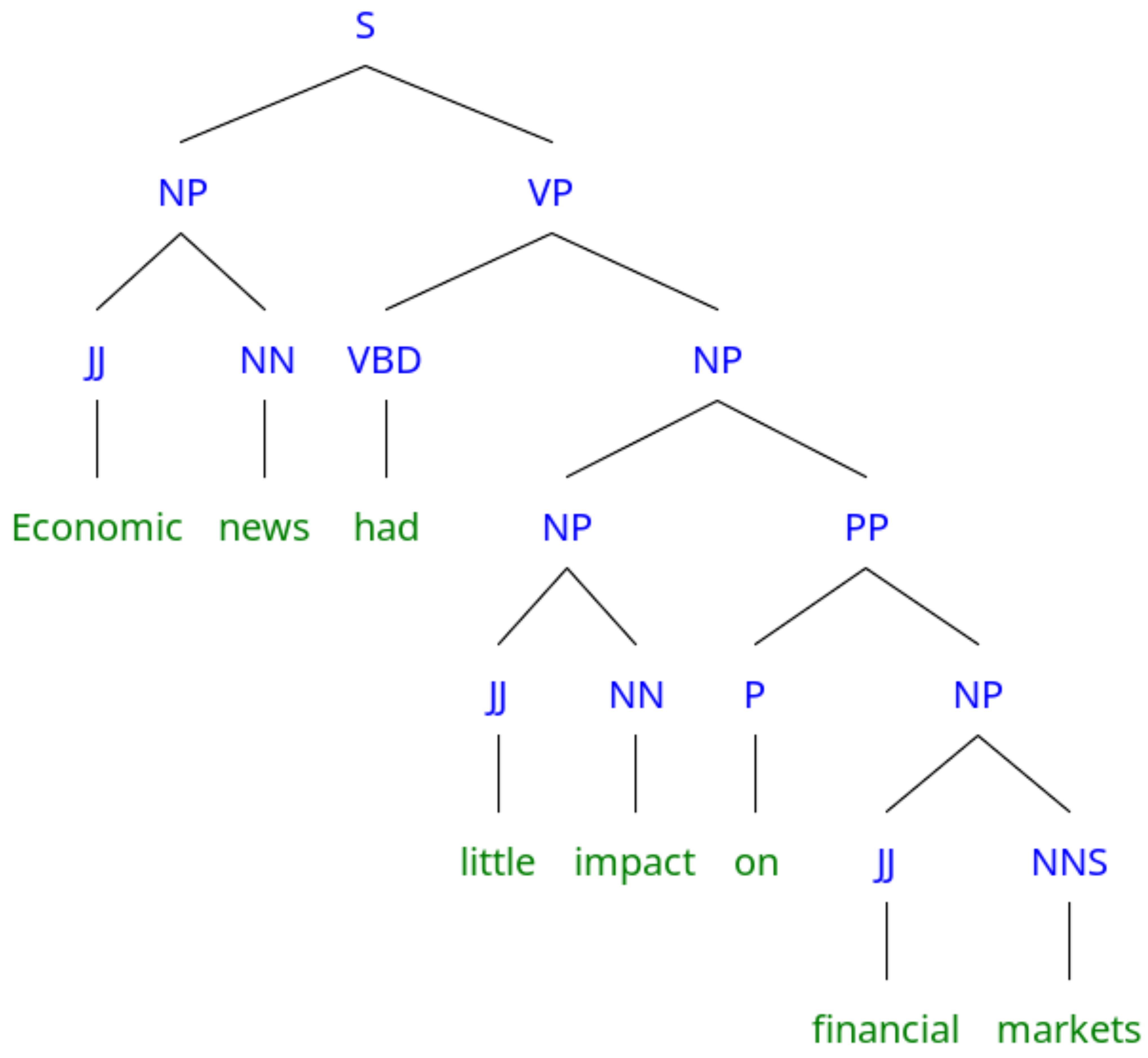


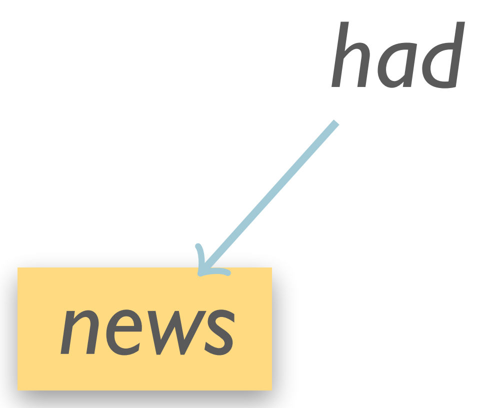
Conversion: PS → DS



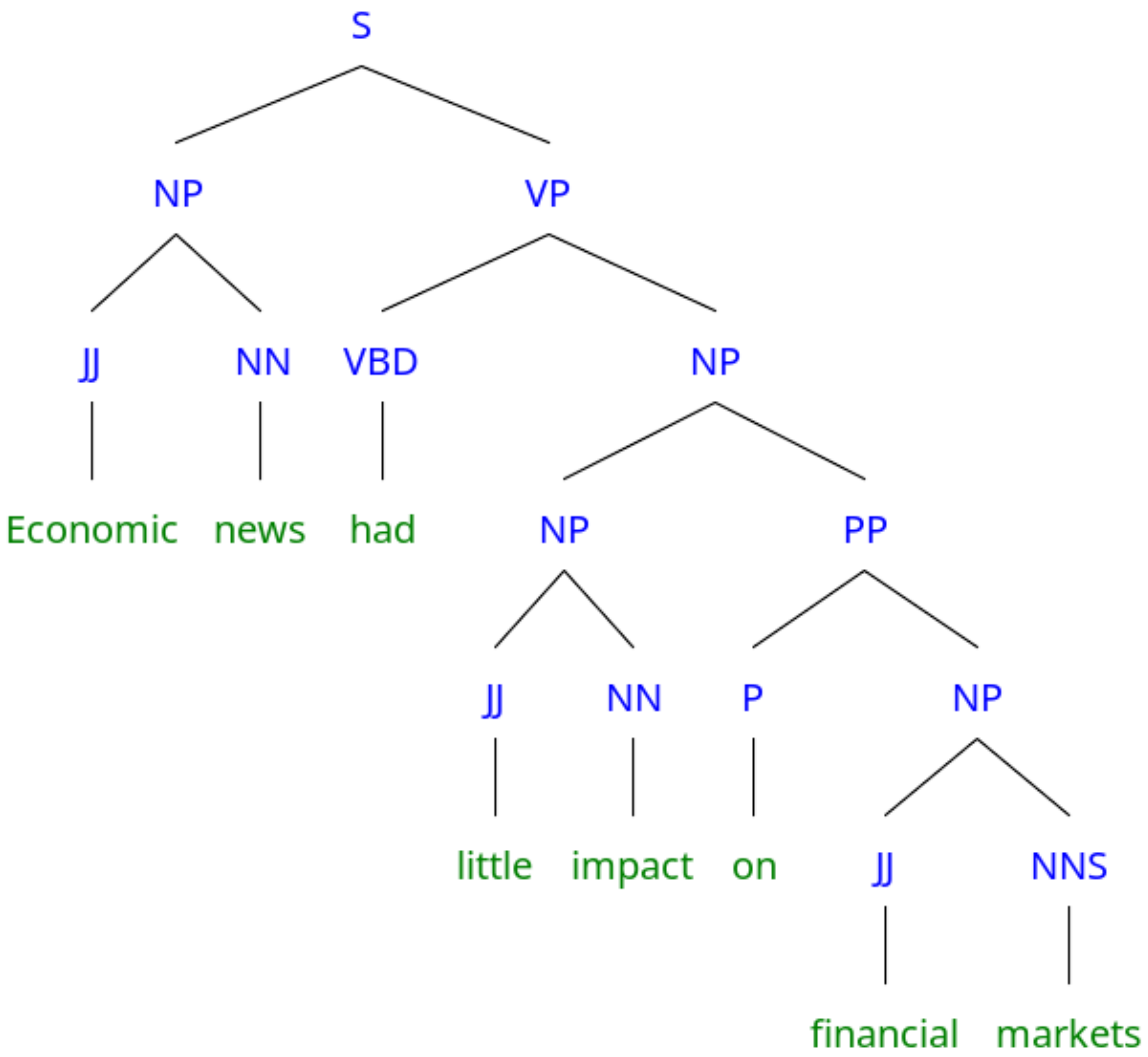


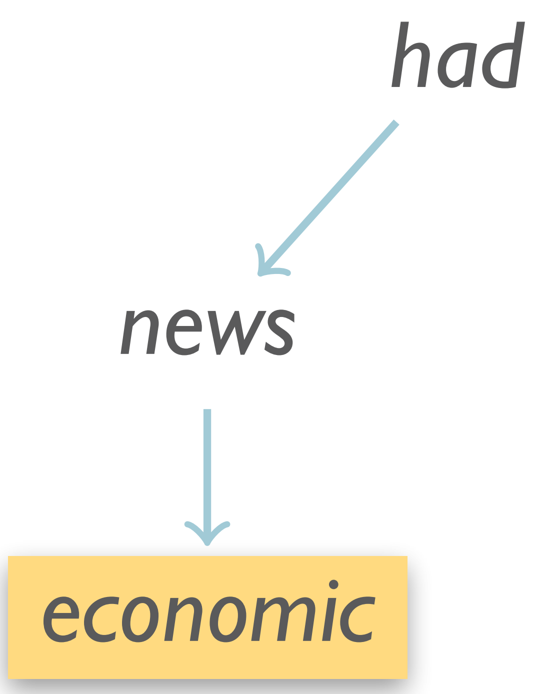
Conversion: PS → DS



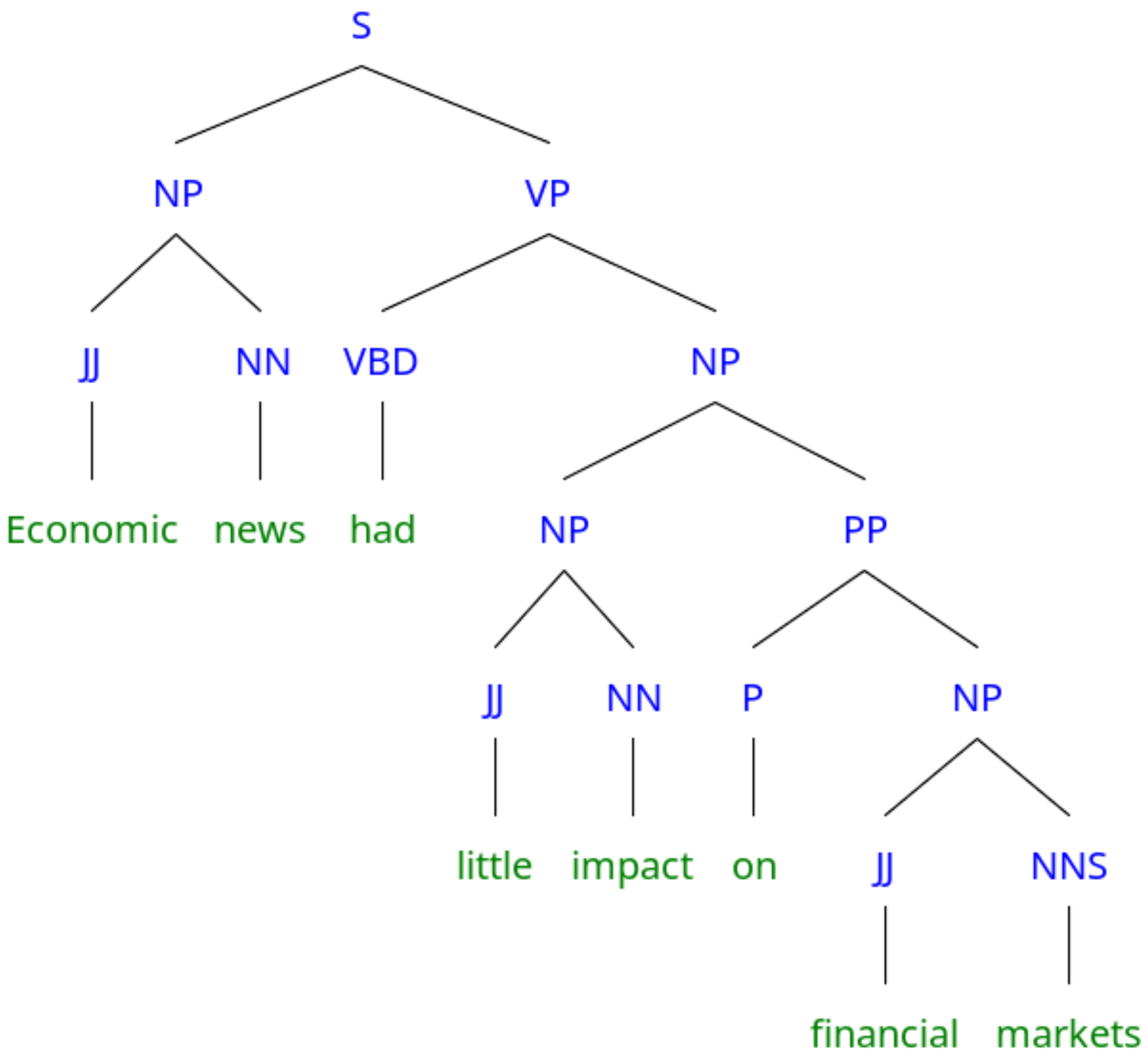


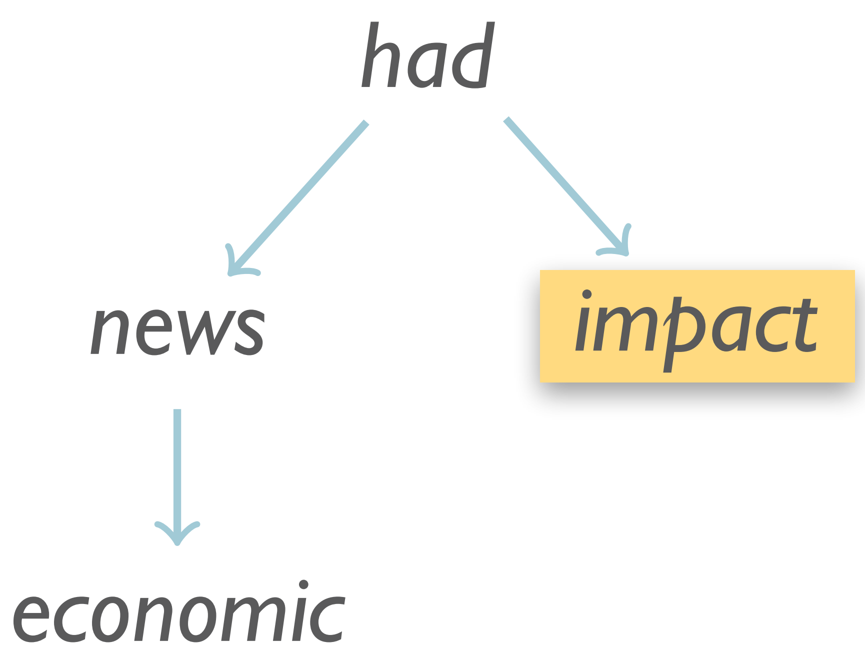
Conversion: PS → DS



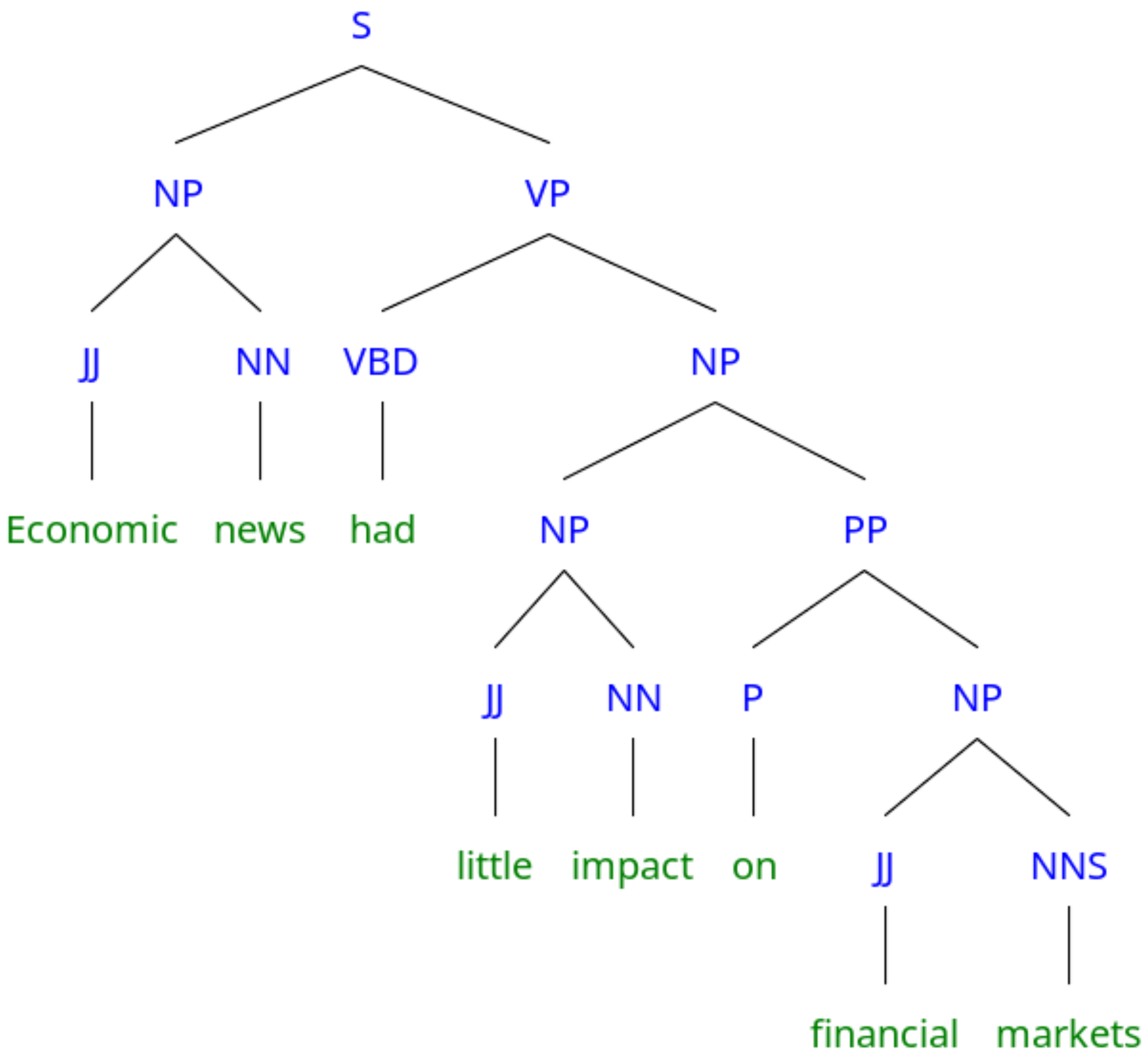


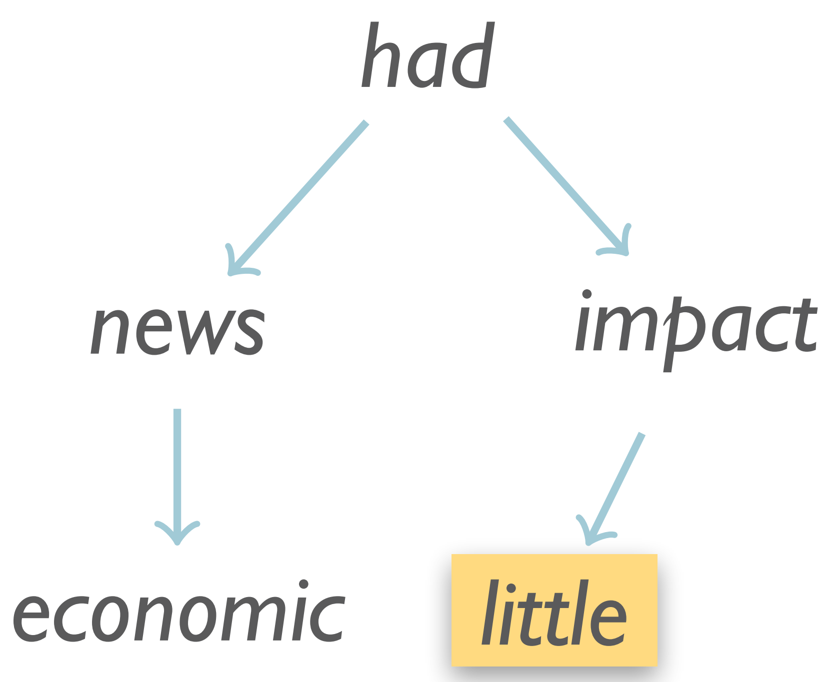
Conversion: PS → DS



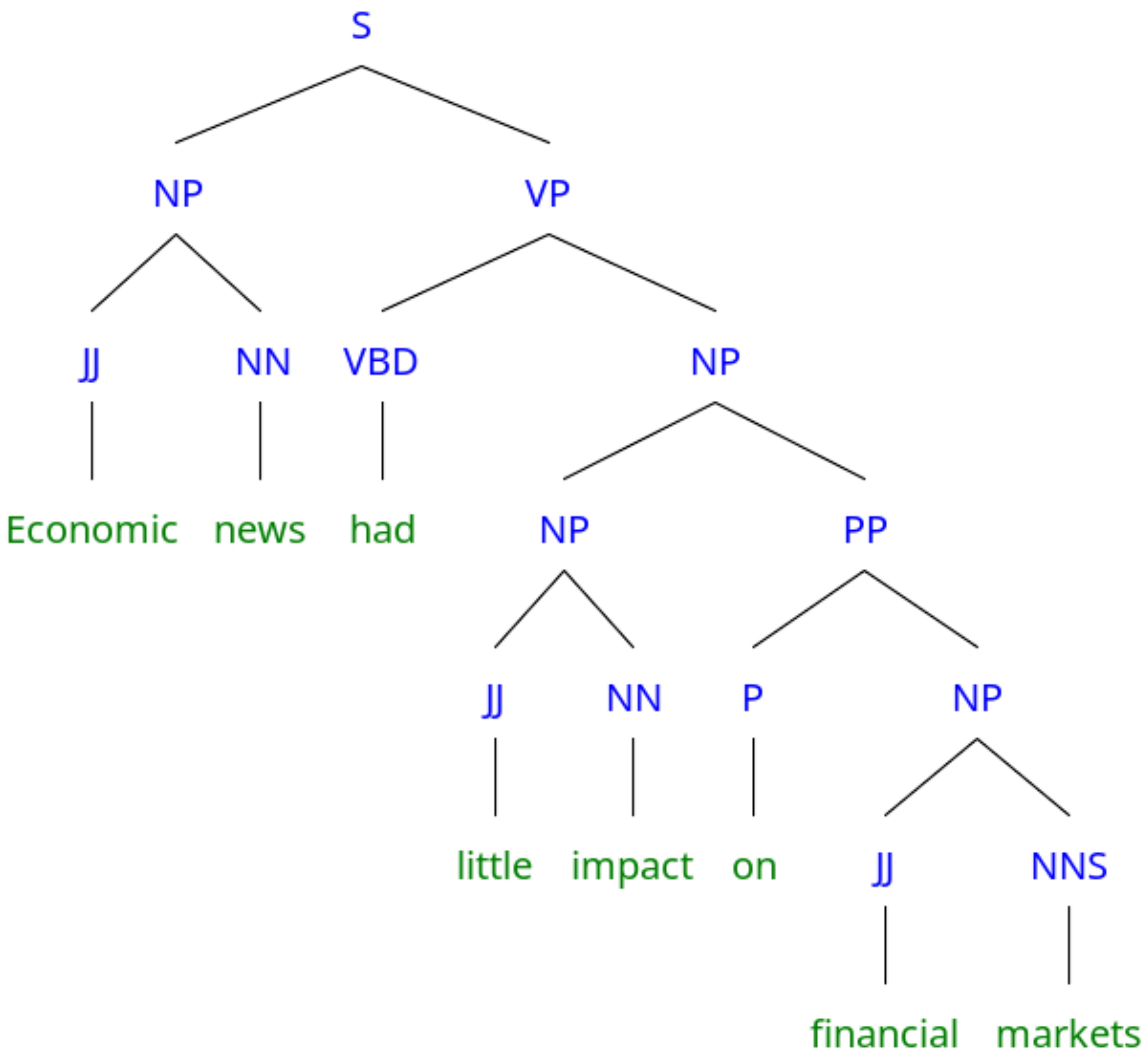


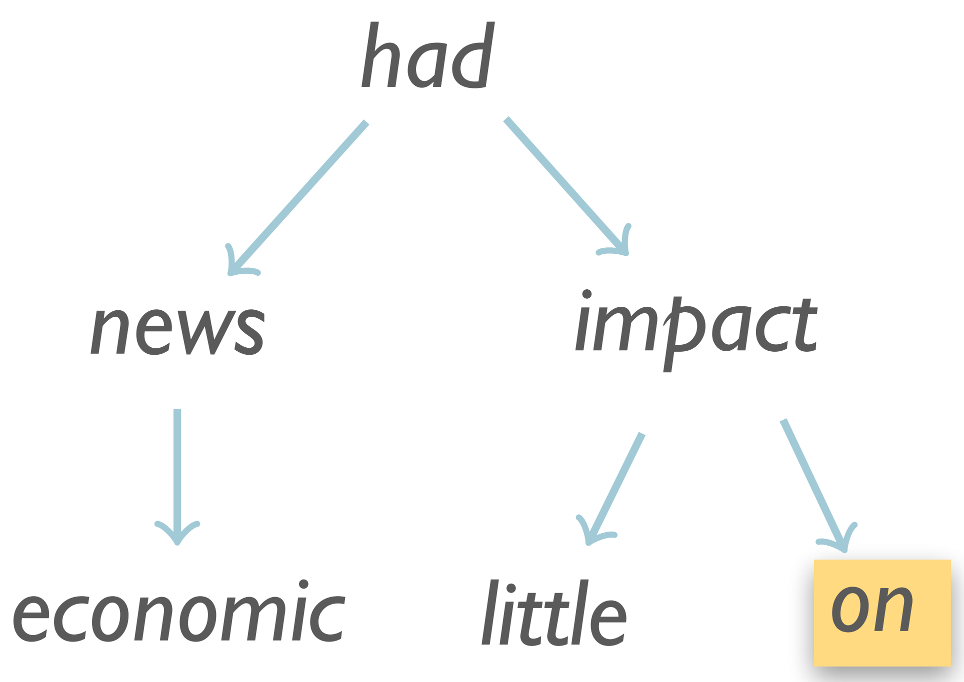
Conversion: PS → DS



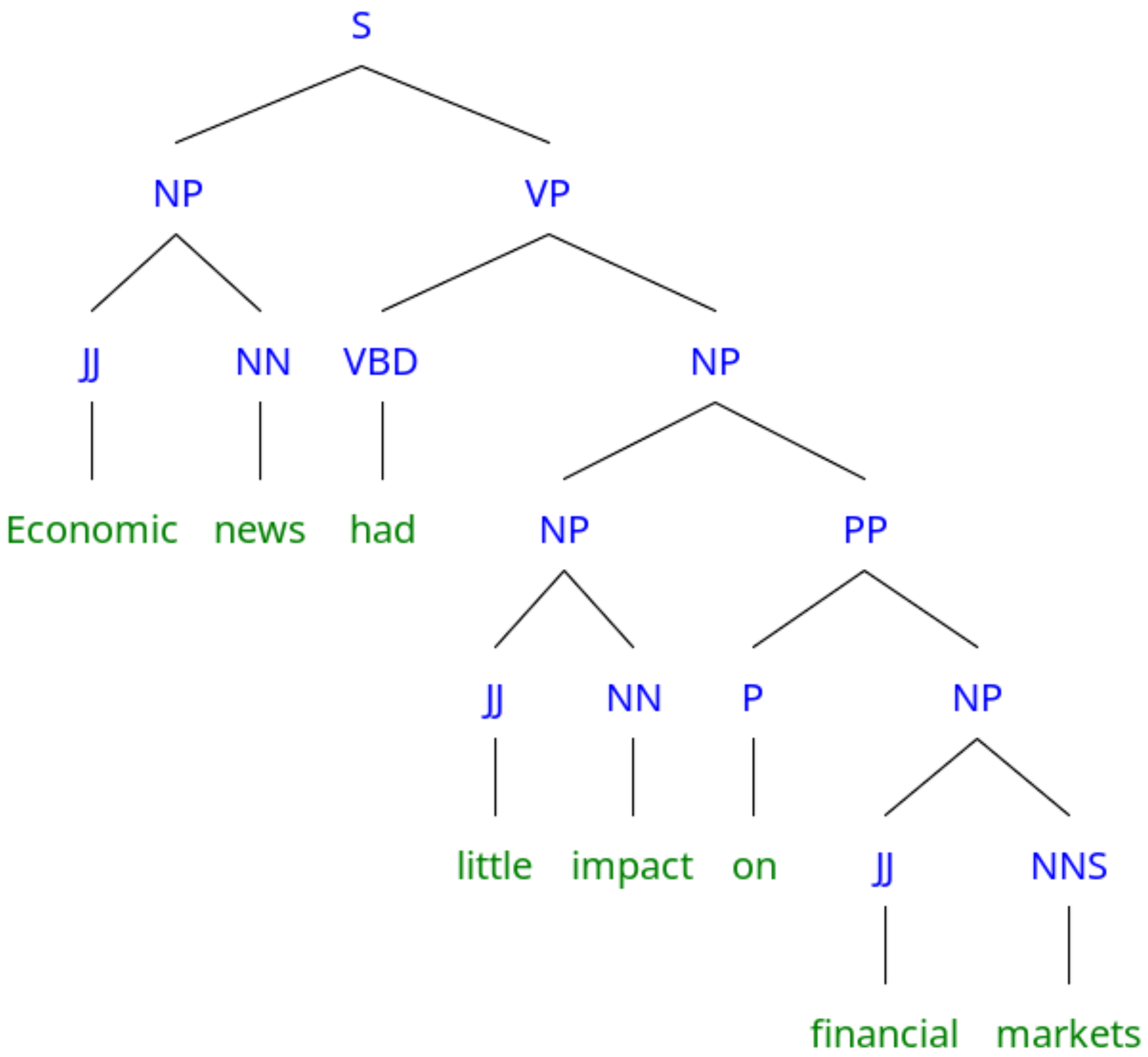


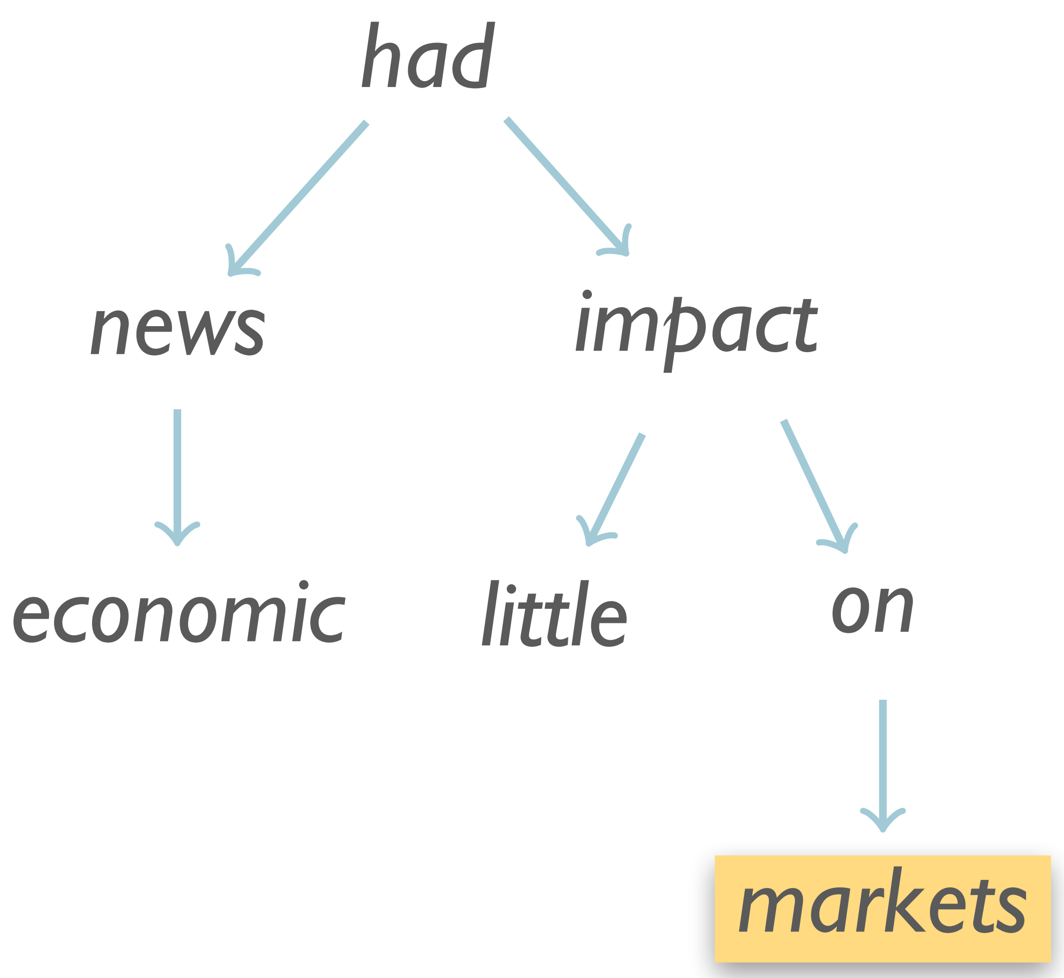
Conversion: PS → DS



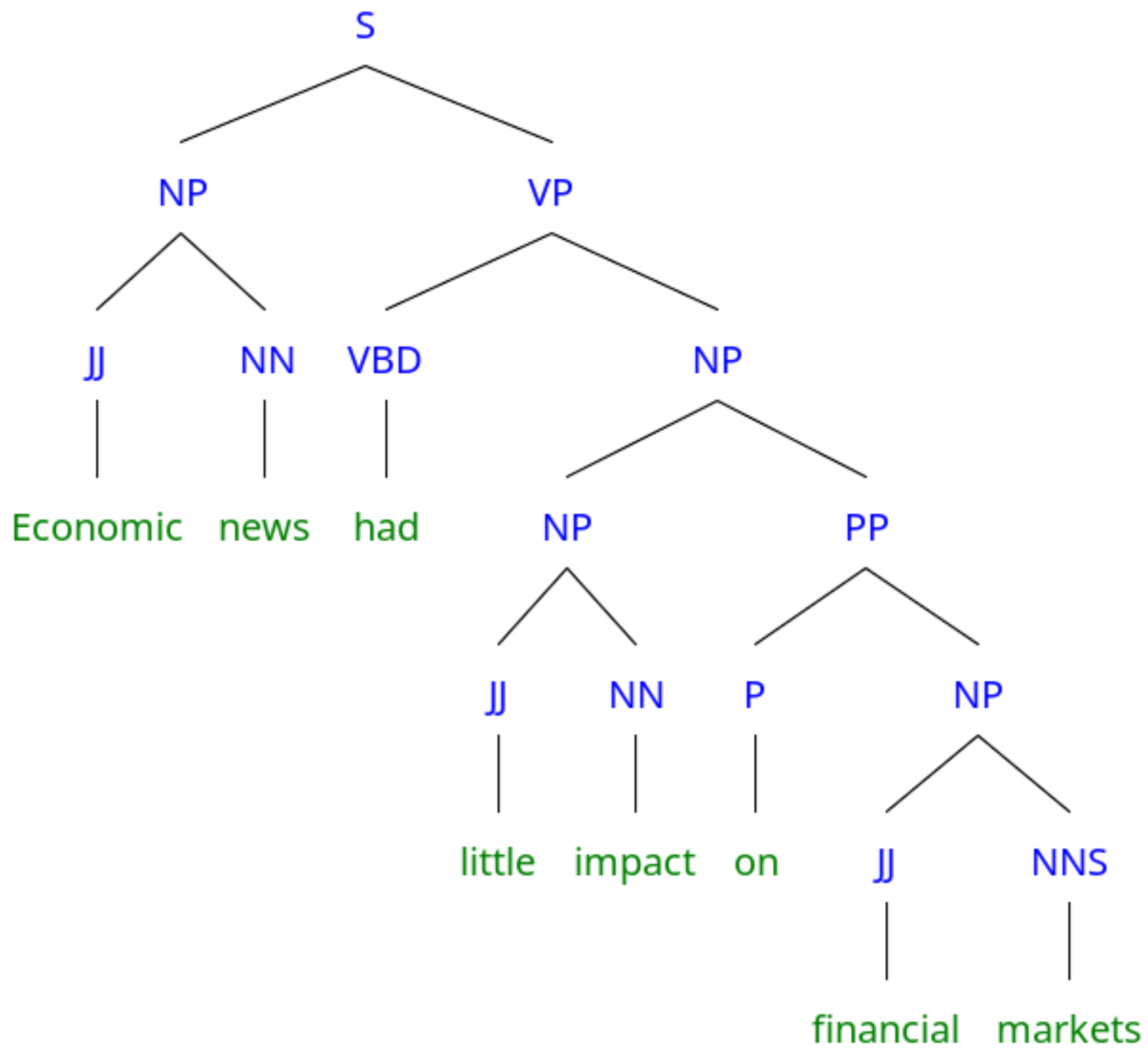


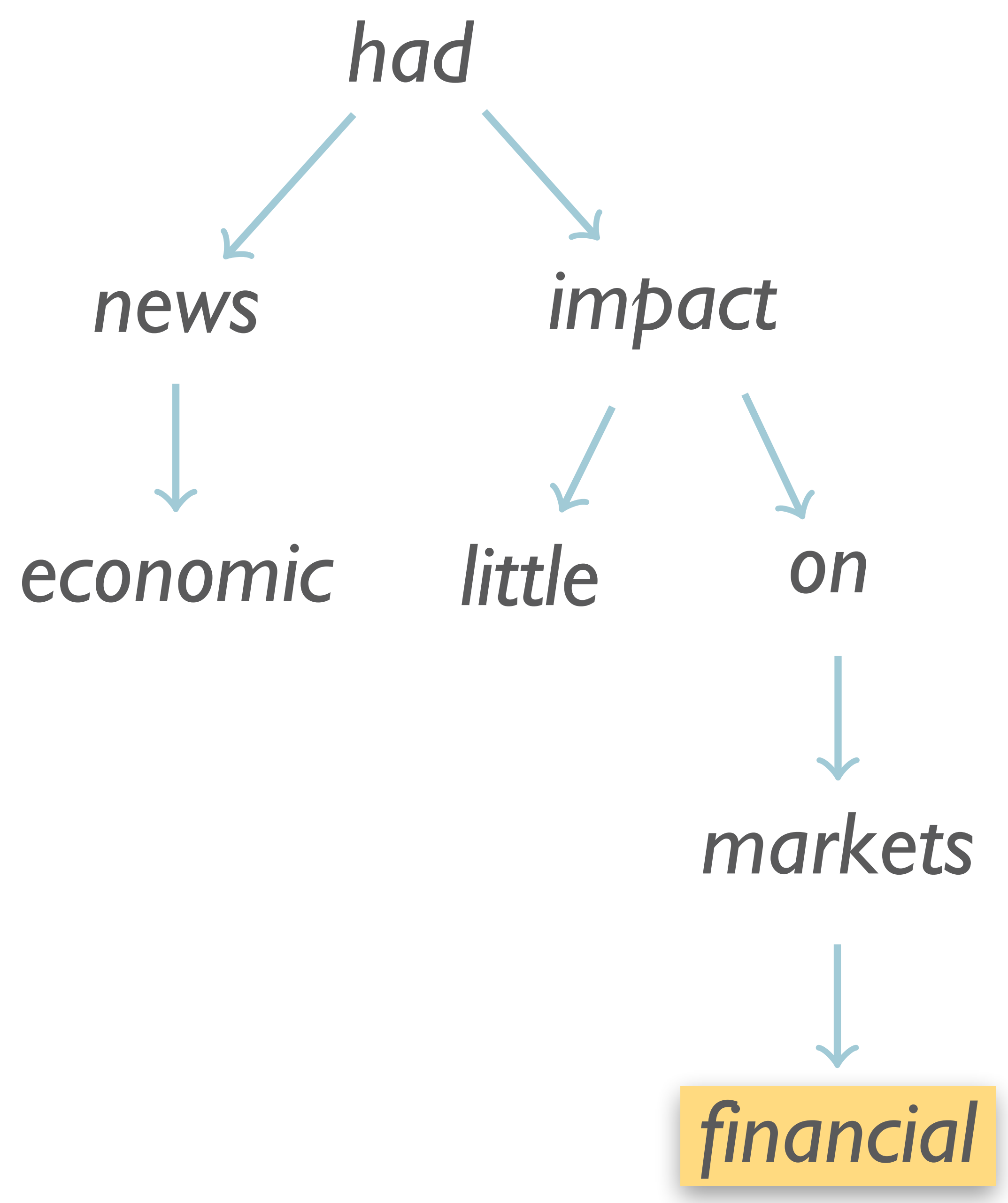
Conversion: PS → DS





Conversion: PS → DS





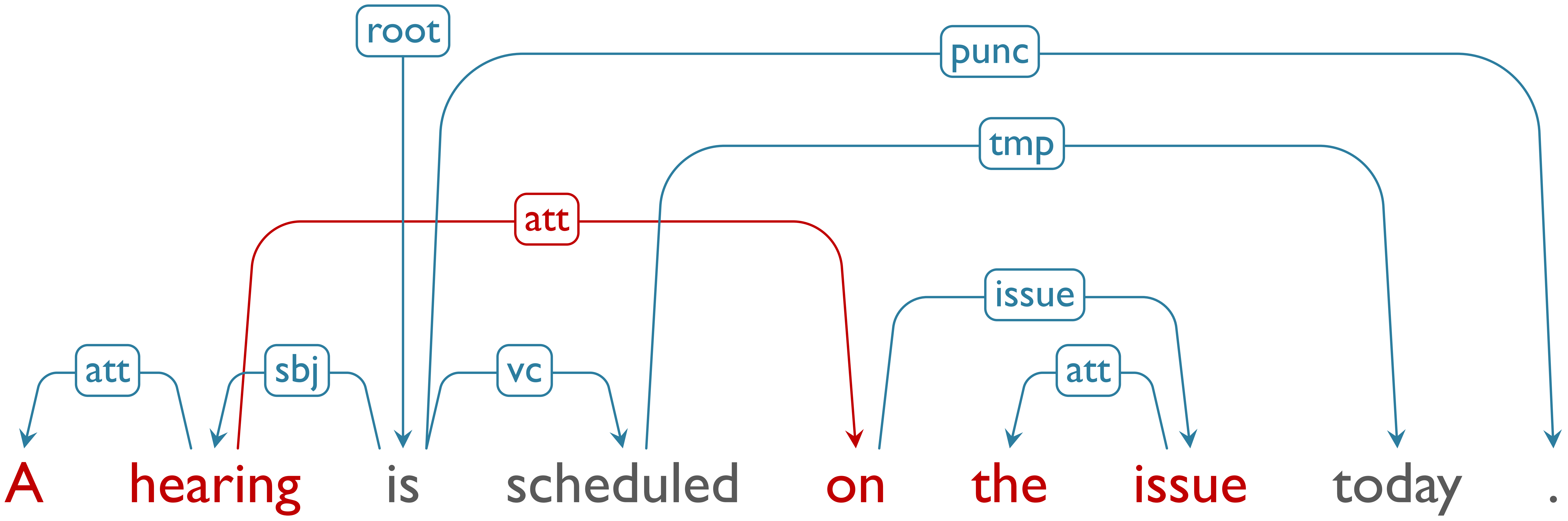
Head Percolation Table

- 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.

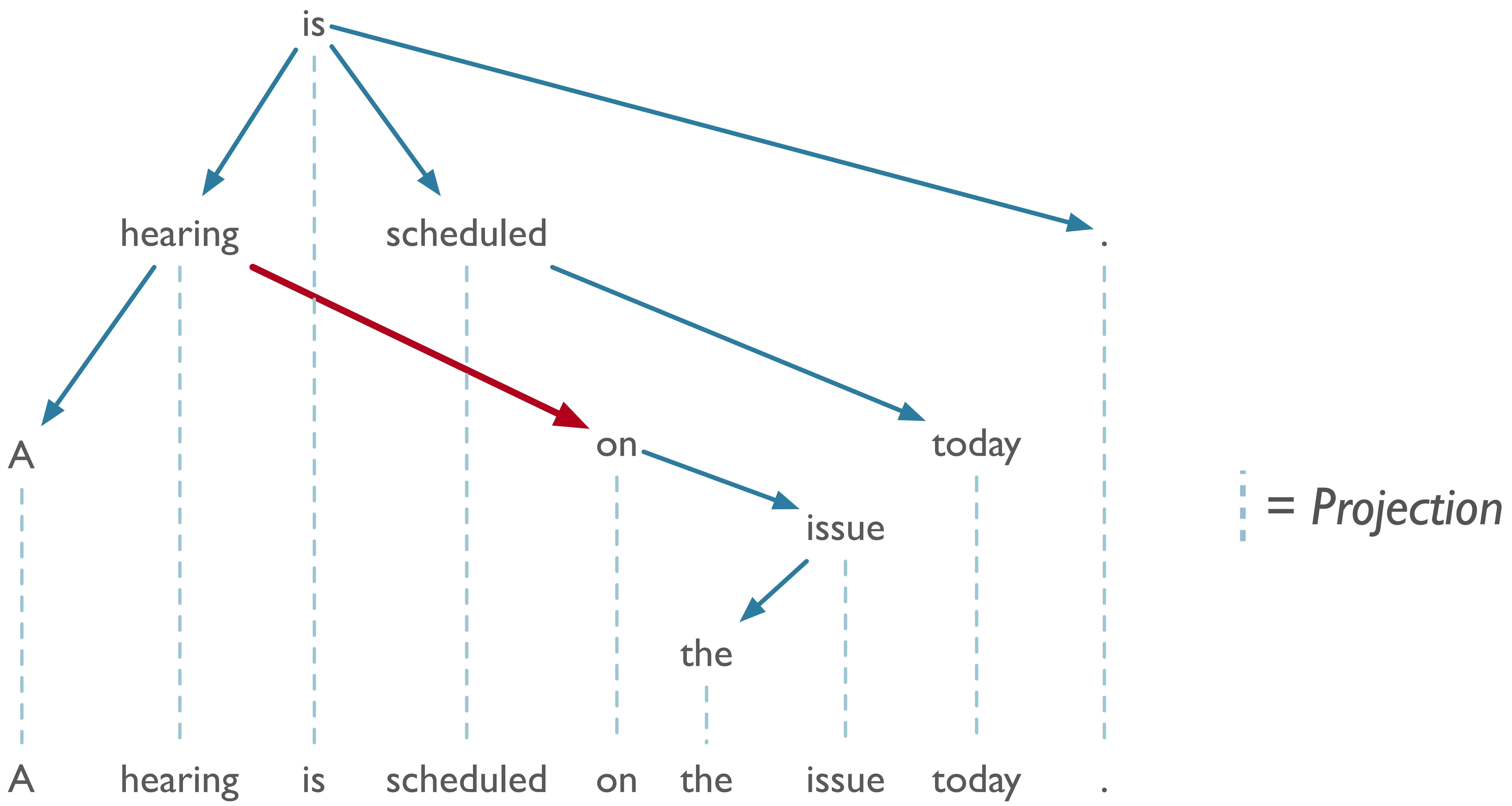
From J&M Page 411, via [Collins (1999)]

Conversion: DS → PS

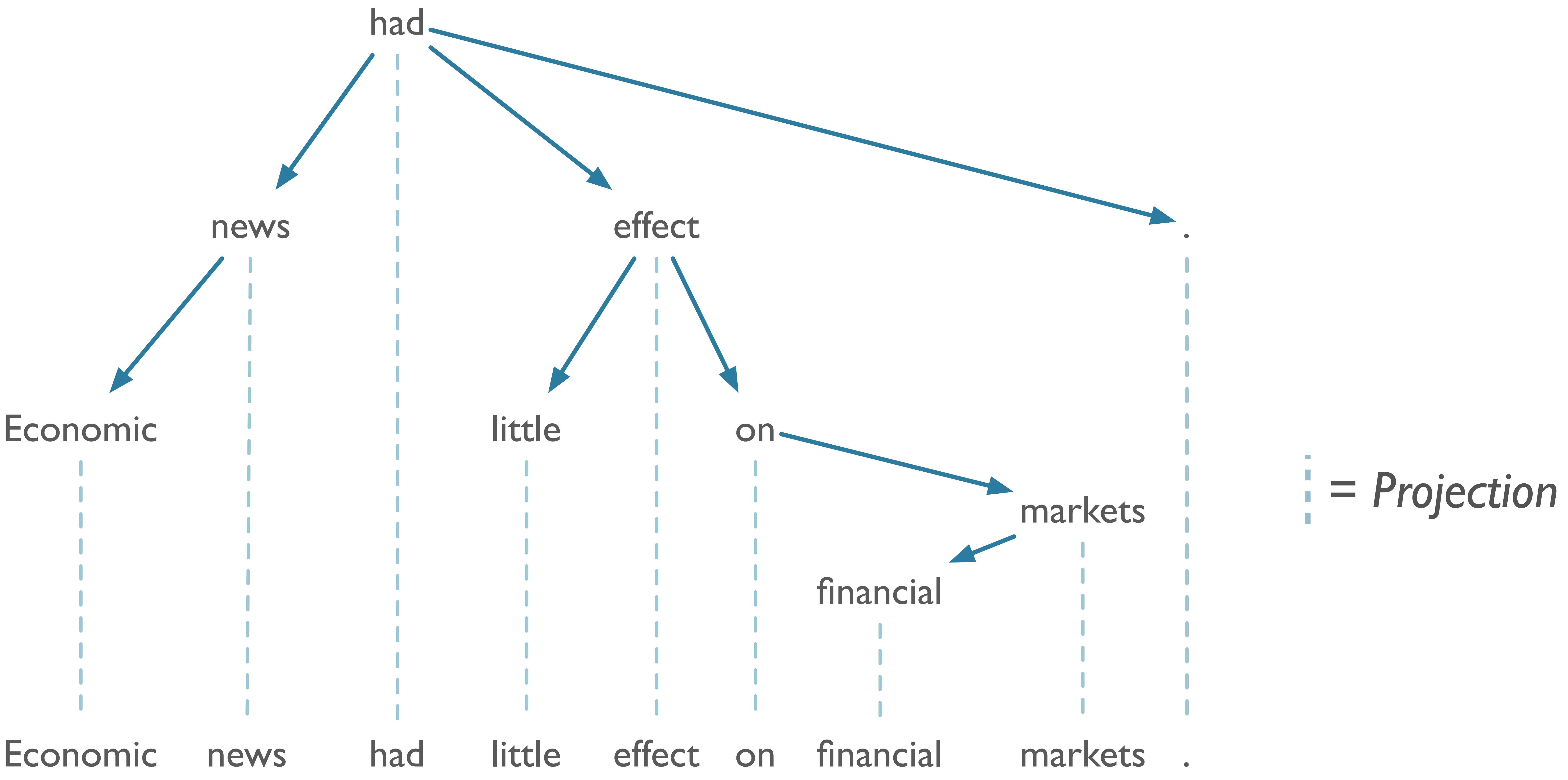
- Can map any *projective* dependency tree to PS tree
- Projective:
  - Does not contain “crossing” dependencies w.r.t. word order



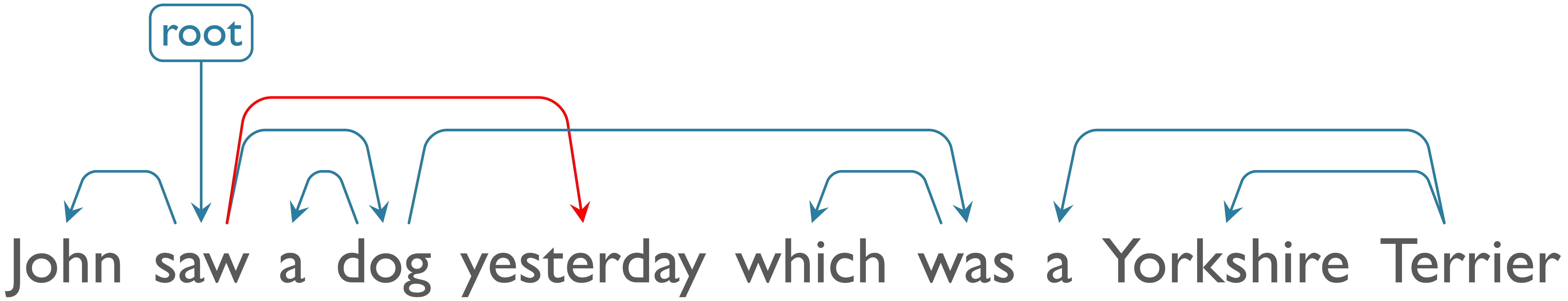
Non-Projective DS

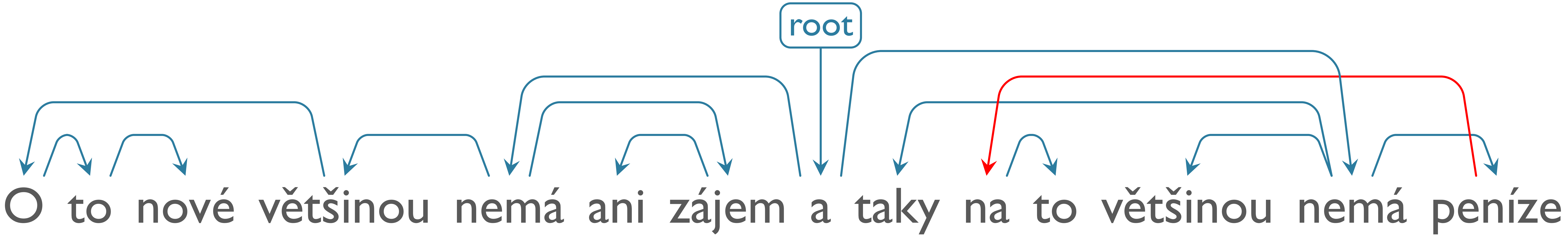


Projective DS



More Non-Projective Parses





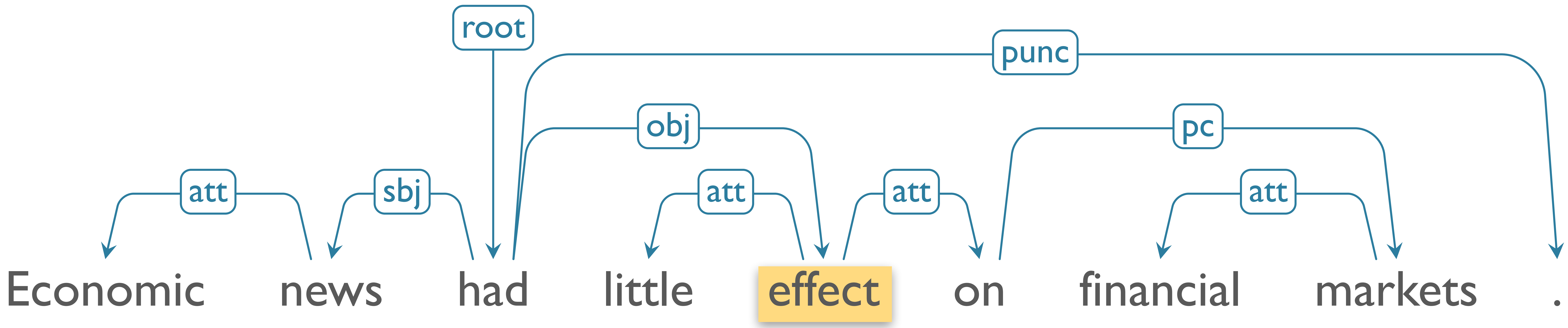
He is mostly not even interested in the new things and in most cases, he has no money for it either.

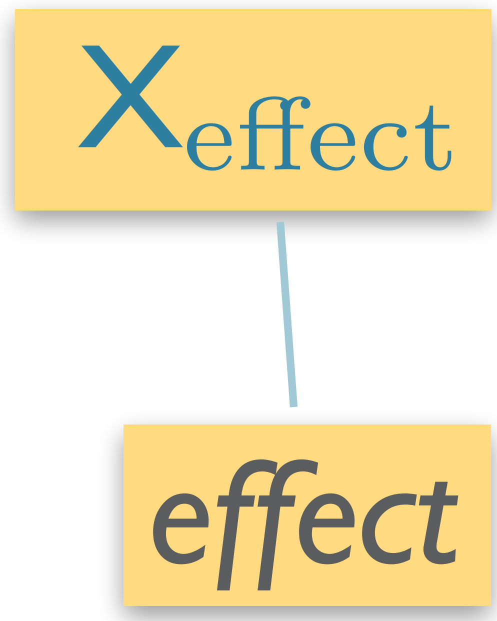
From [McDonald et. al, 2005]

Conversion: DS → PS

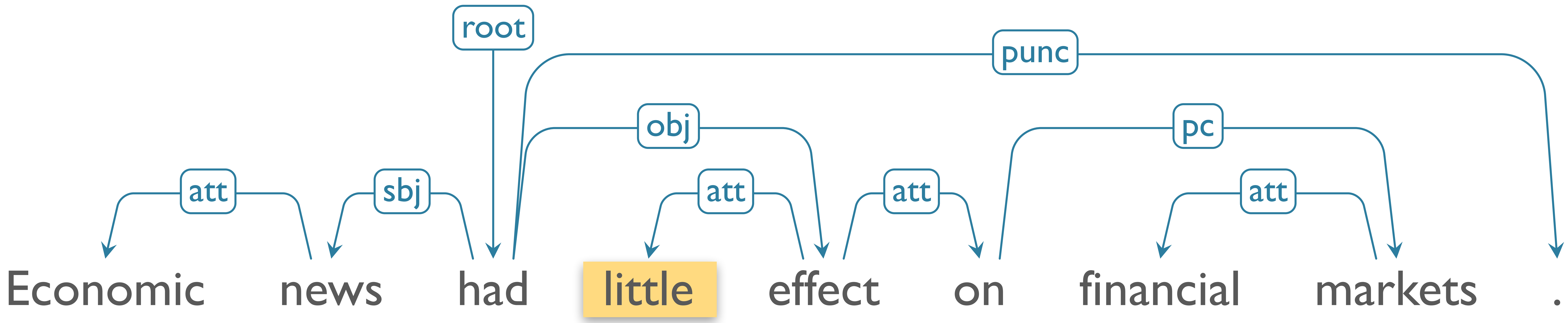
- For each node w with outgoing arcs…
  - …convert the subtree w and its dependents $t_1,\ldots,t_n$ to a new subtree:
    - Nonterminal: X_w
    - Child: w
    - Subtrees $t_1,\ldots,t_n$ in original sentence order

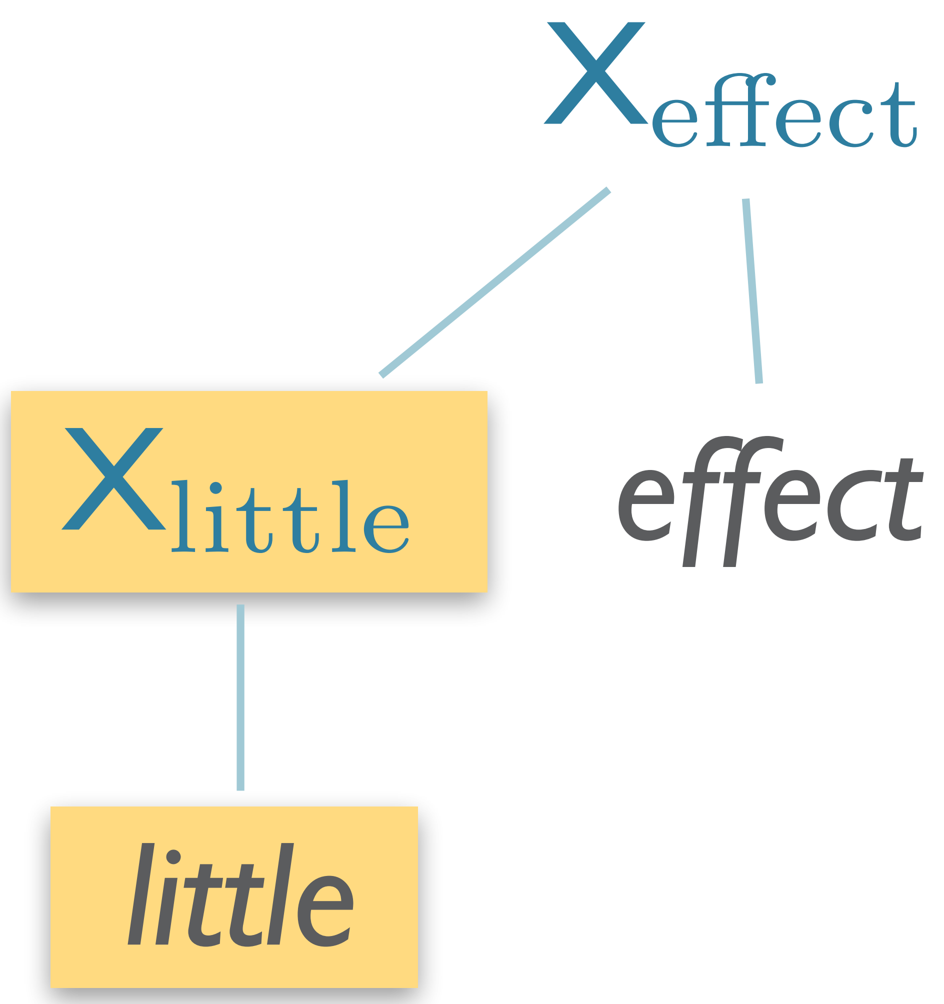
Conversion: DS → PS



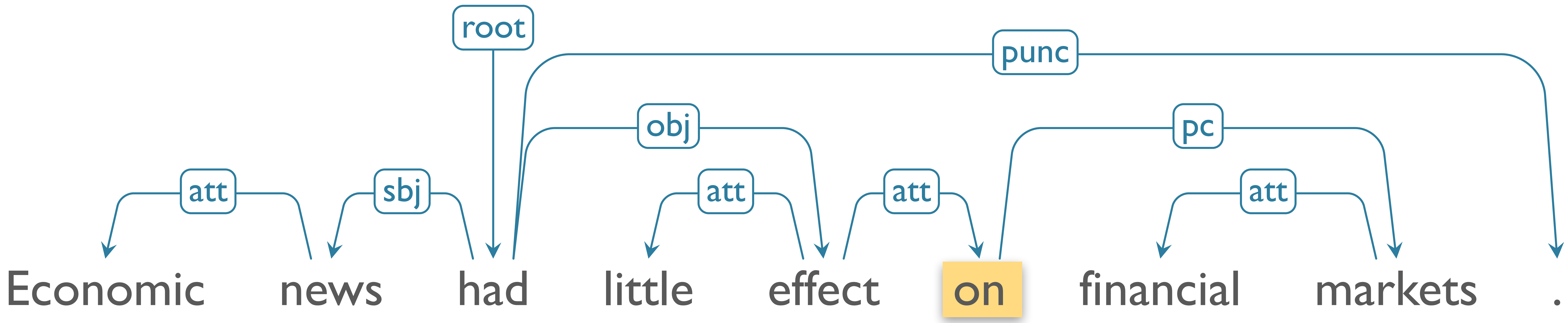


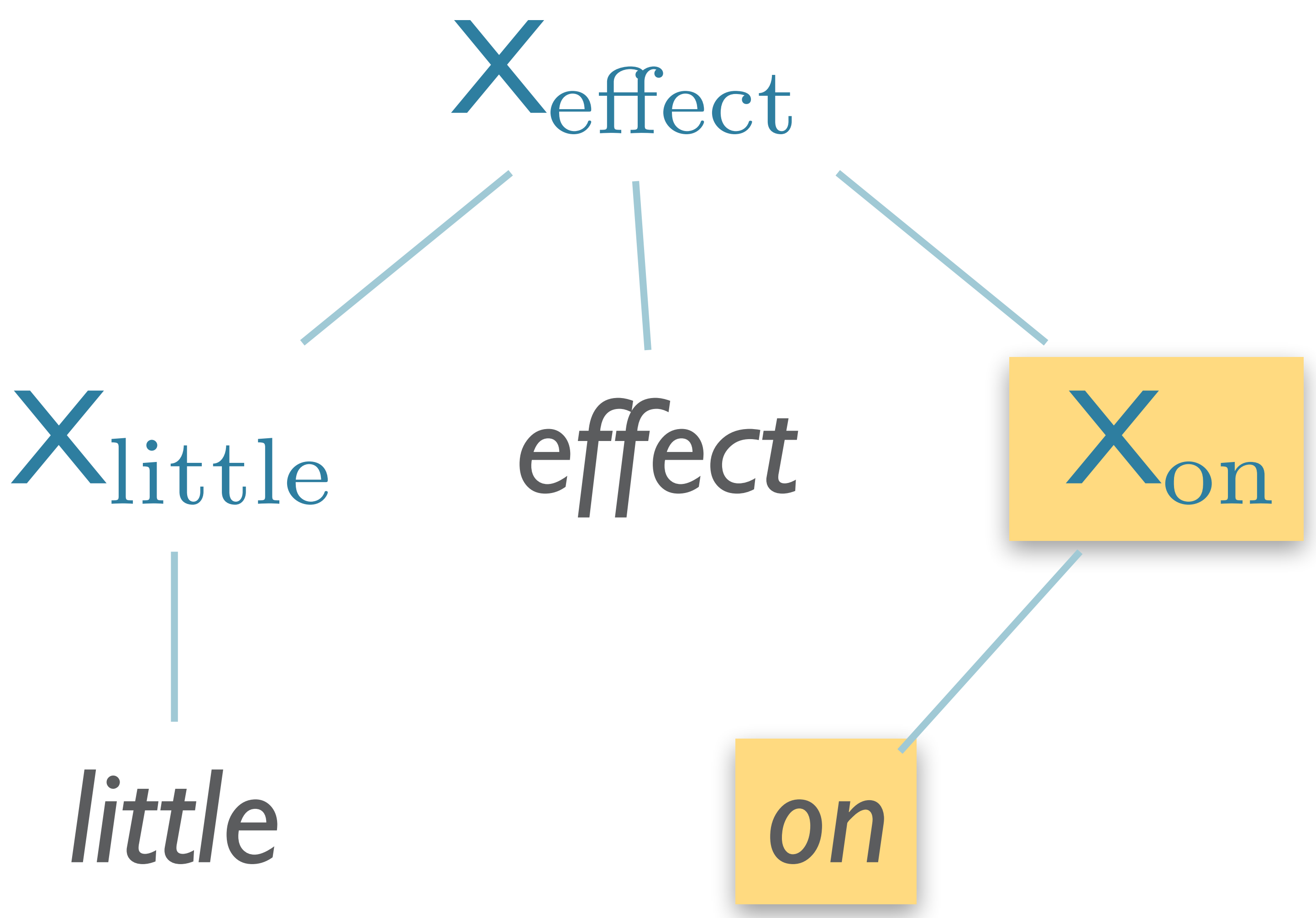
Conversion: DS → PS



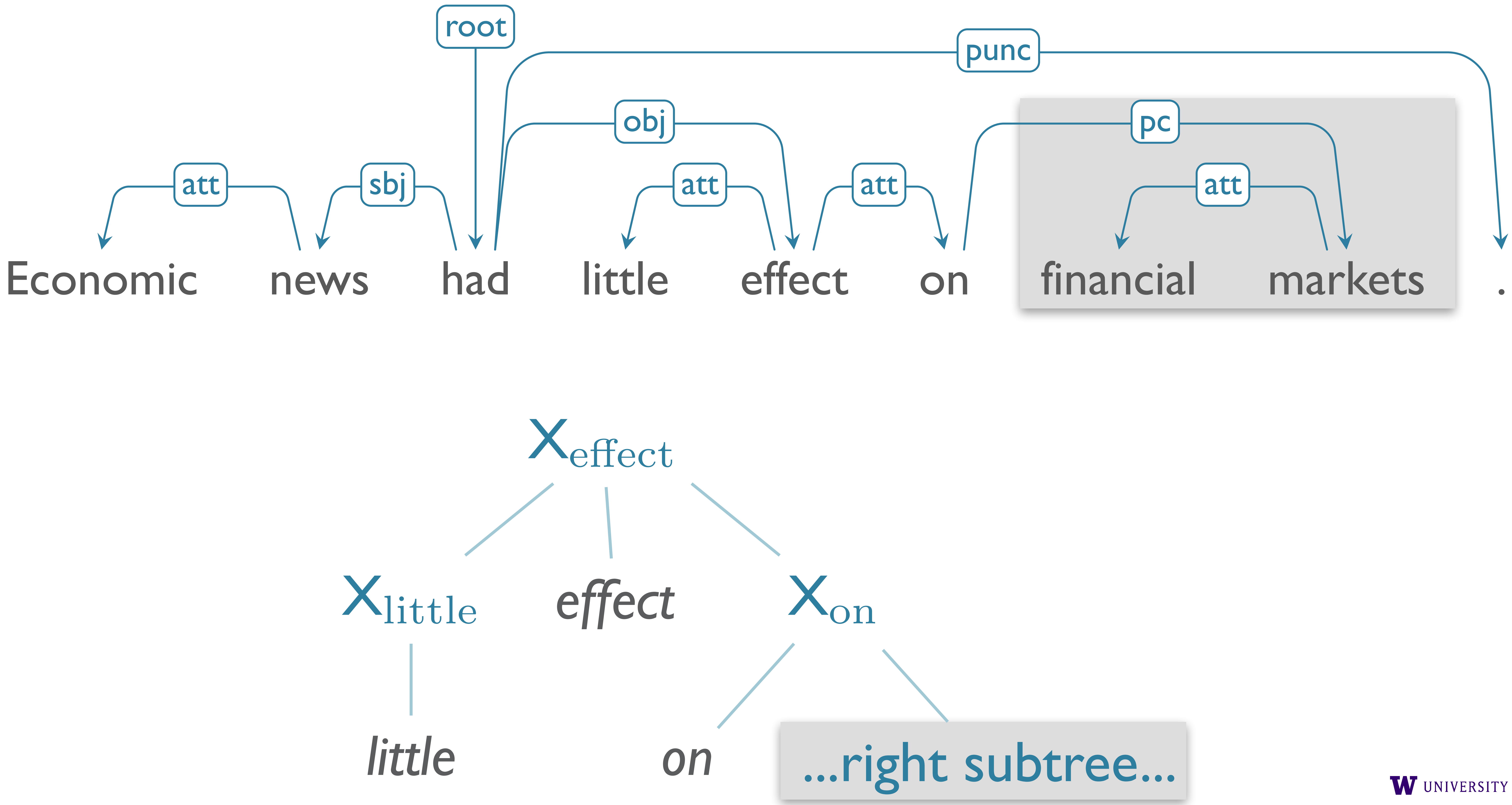


Conversion: DS → PS



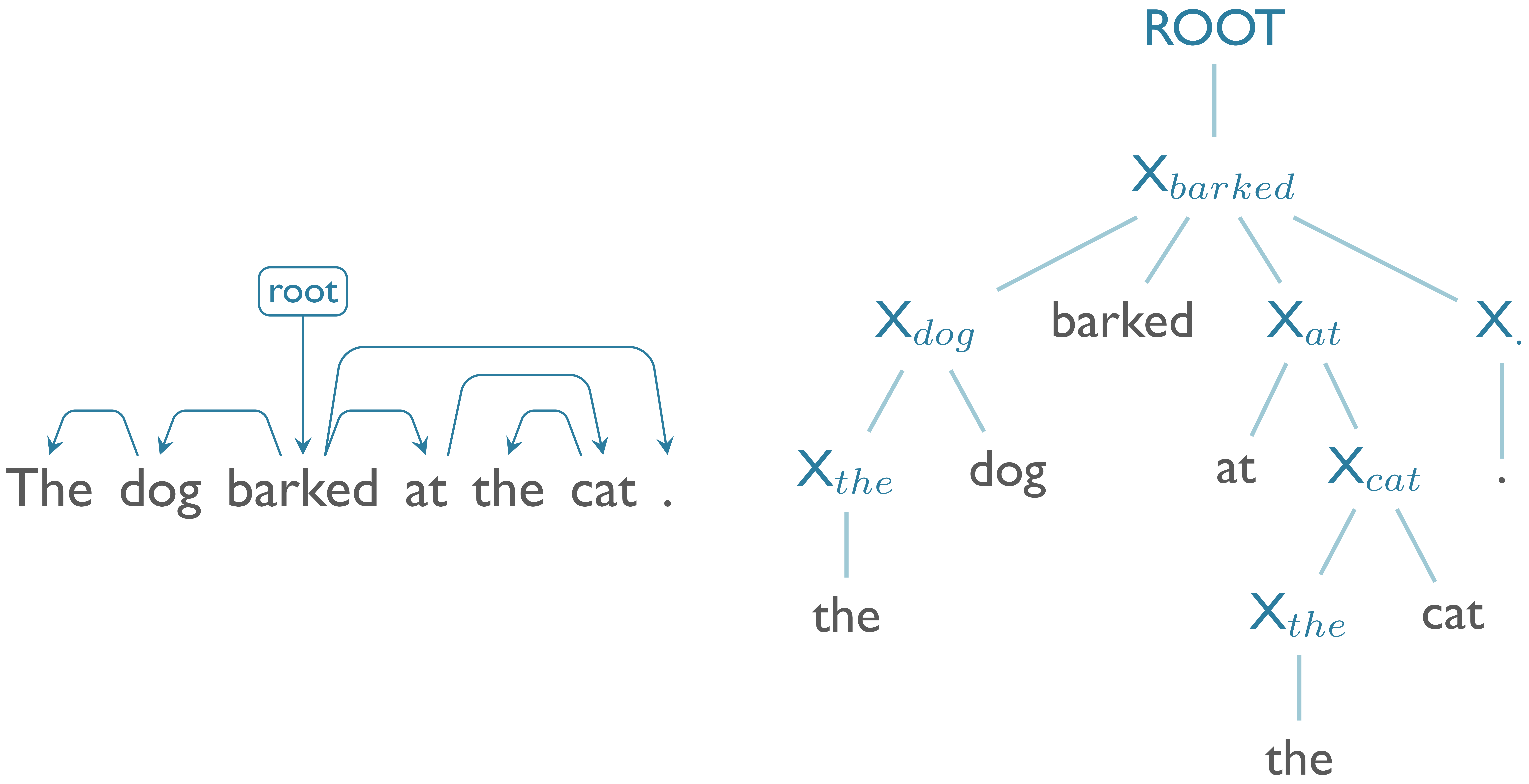


Conversion: DS → PS



Conversion: DS → PS

- What about labeled dependencies?
  - Can attach labels to nonterminals associated with non-heads
  - e.g. $X_{little} \rightarrow X_{little:nmod}$
- 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

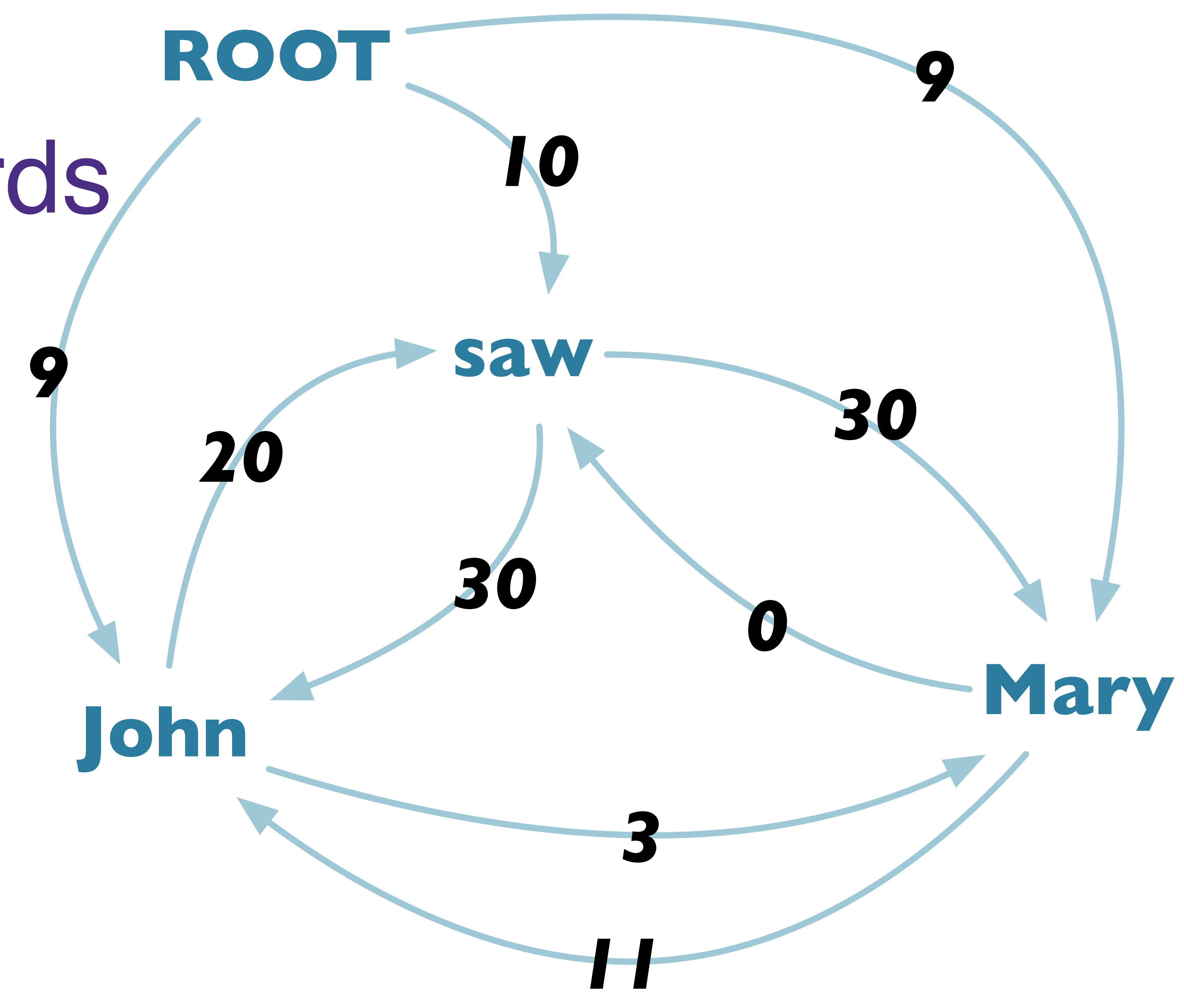
- Map dependency parsing to Maximum Spanning Tree (MST)
- Build fully connected initial graph:
  - Nodes: words in sentence to parse
  - Edges: directed edges between all words
    - + Edges from ROOT to all words
- Identify maximum spanning tree
  - Tree s.t. all nodes are connected
  - Select such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

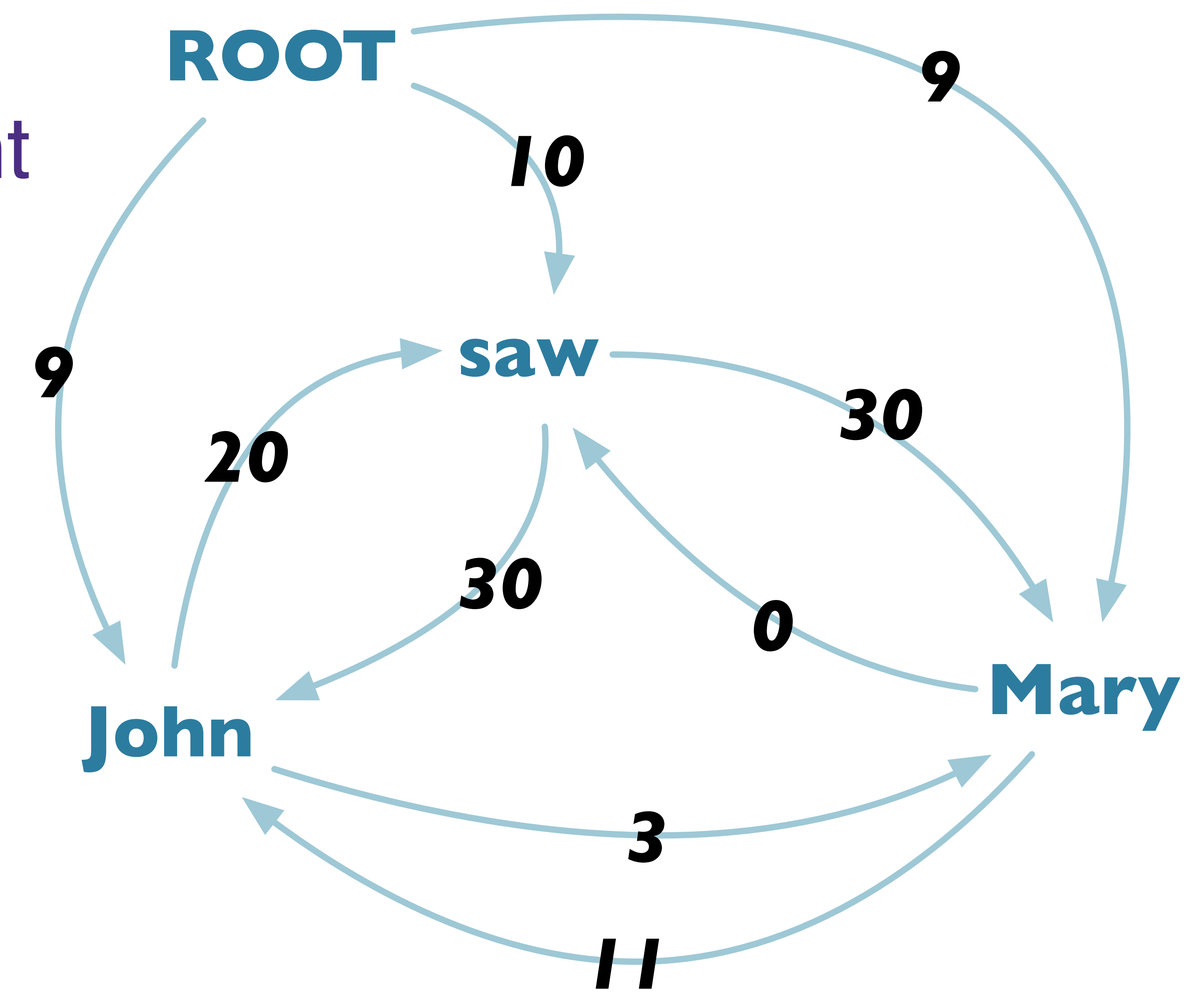
Initial Graph: (McDonald et al, 2005b)

- *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



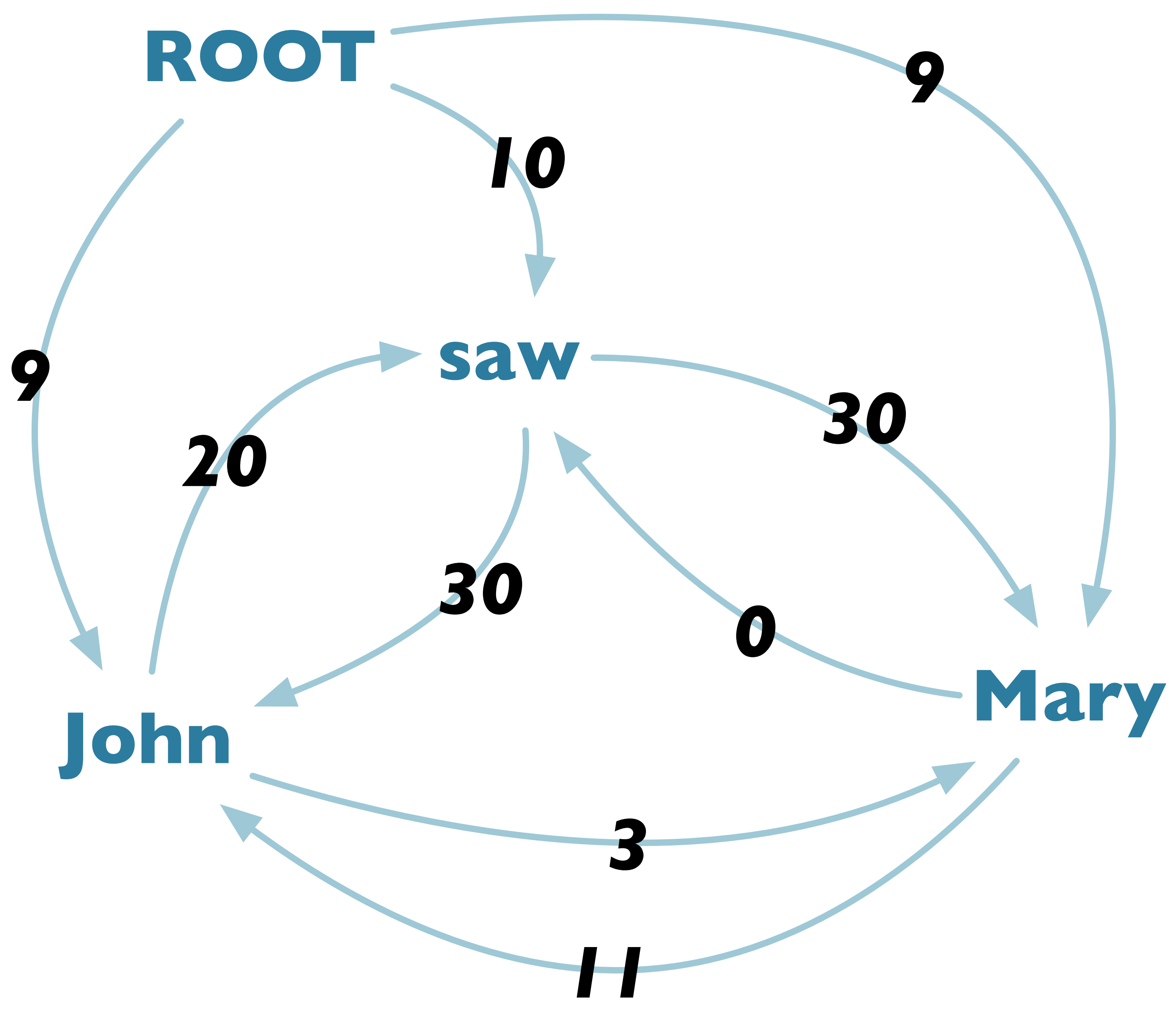
Maximum Spanning Tree

- 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



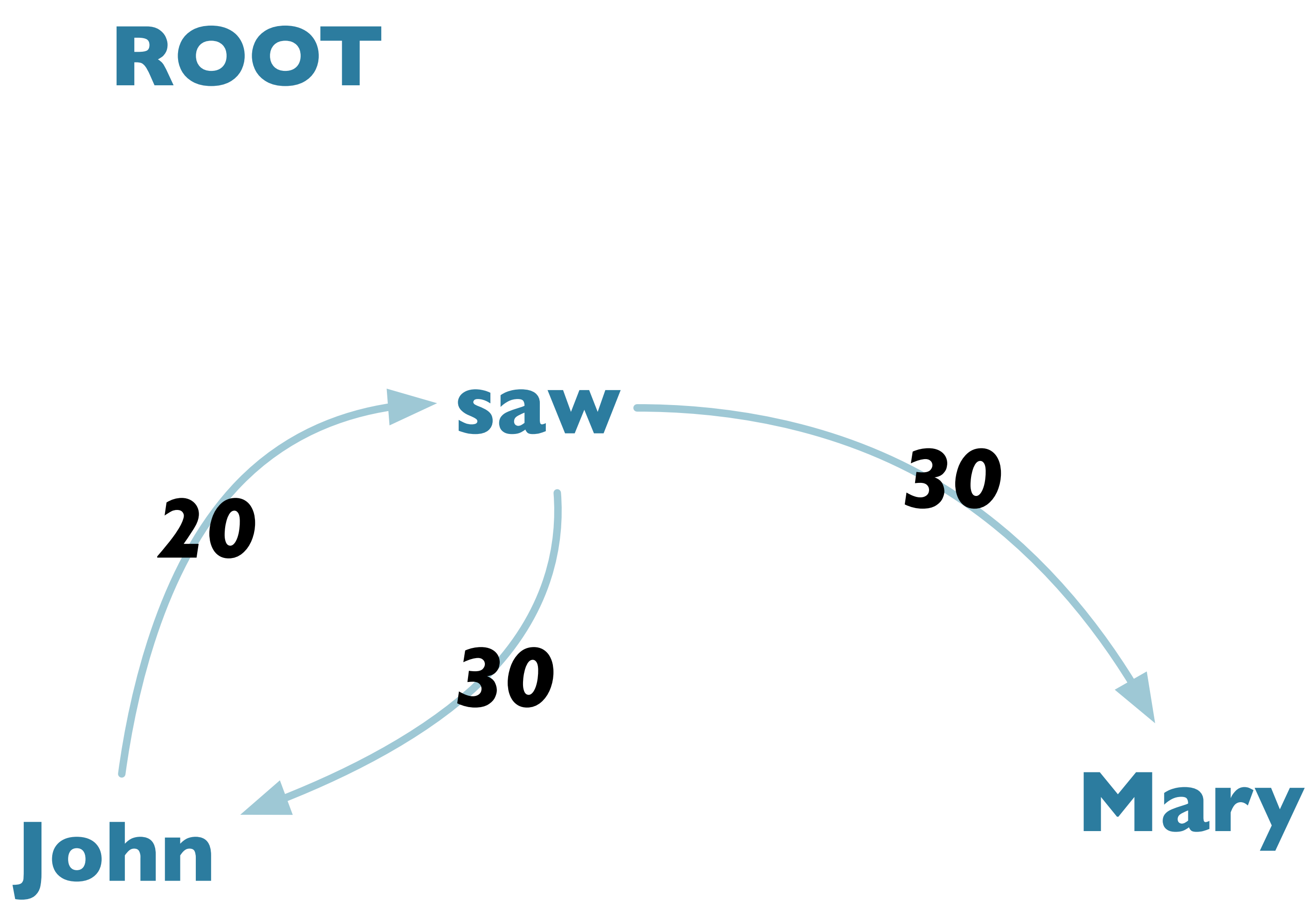
Step 1 & 2

- Find, for each word, the highest scoring incoming edge.



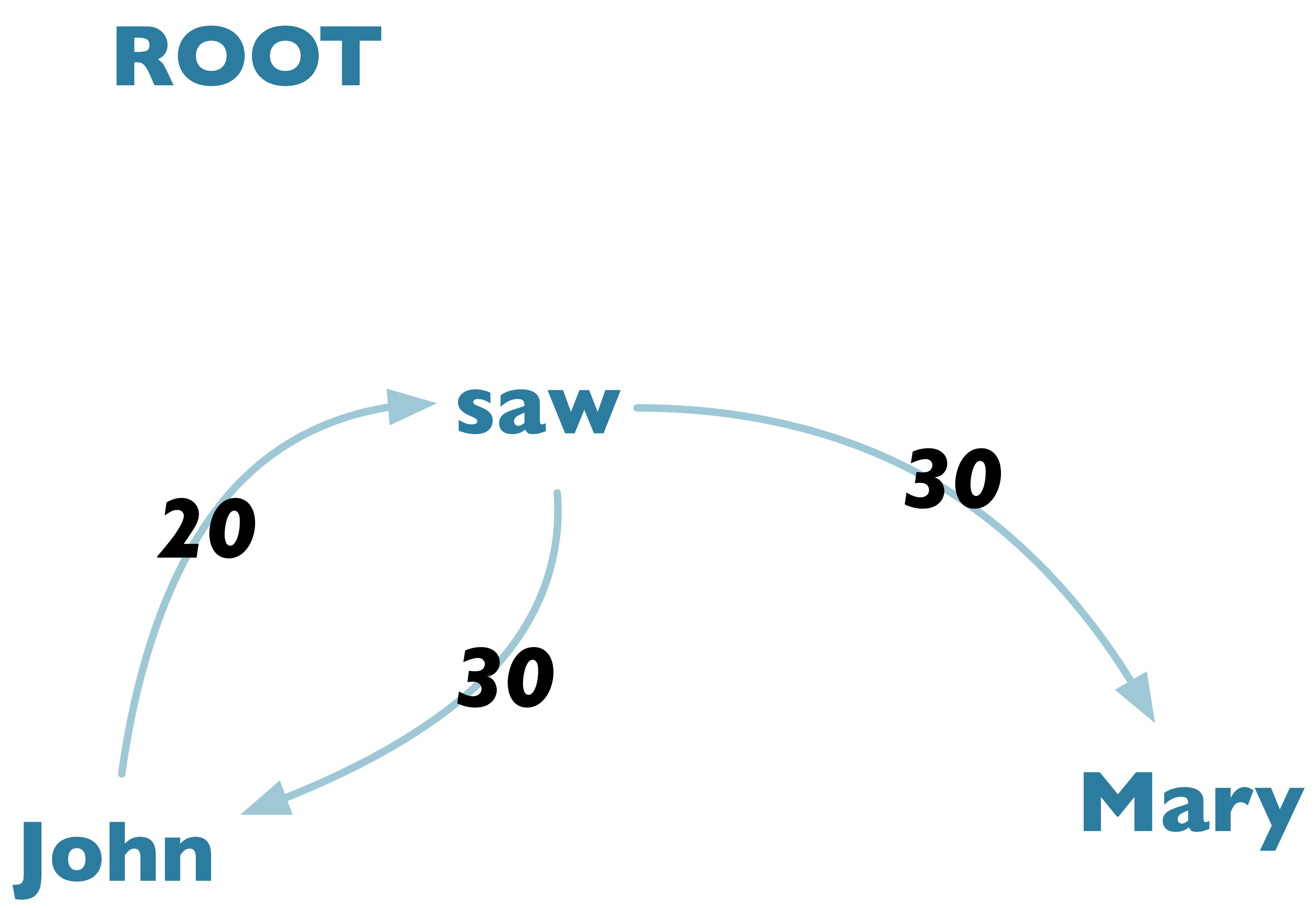
Step 1 & 2

- Find, for each word, the highest scoring incoming edge.



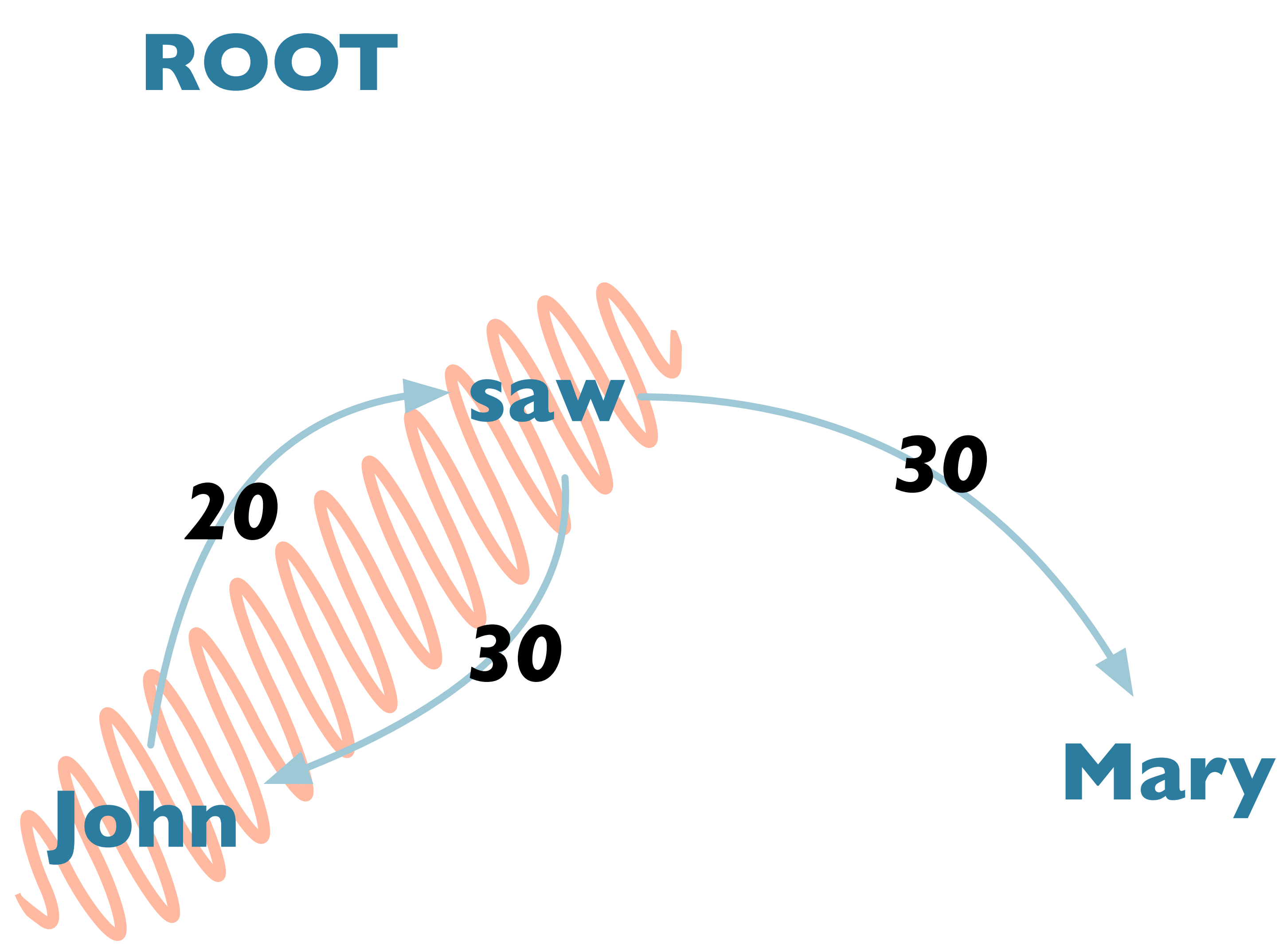
Step 1 & 2

- Find, for each word, the highest scoring incoming edge.
- Is it a tree?



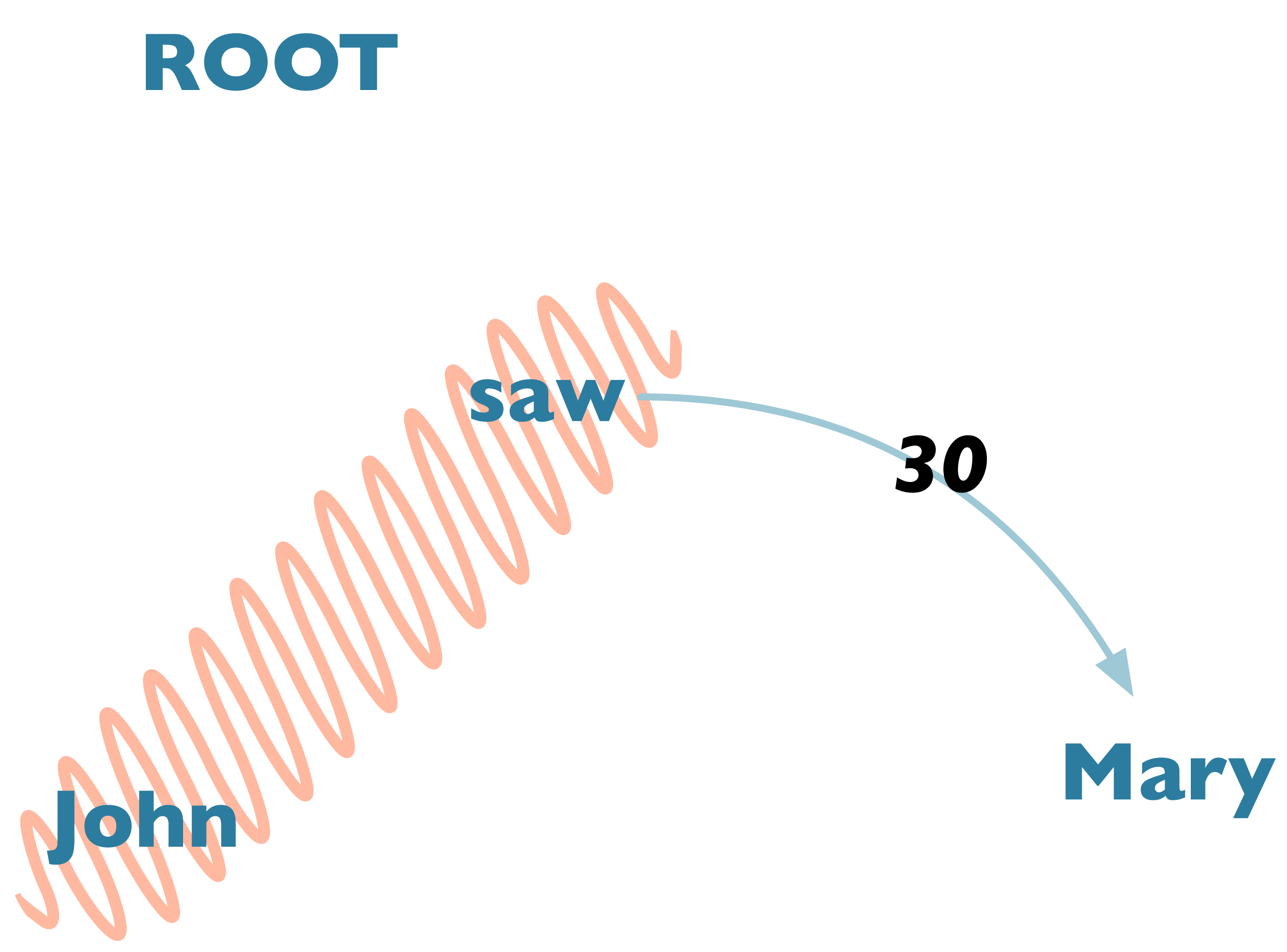
Step 1 & 2

- Find, for each word, the highest scoring incoming edge.
- Is it a tree?
  - No, there’s a cycle.



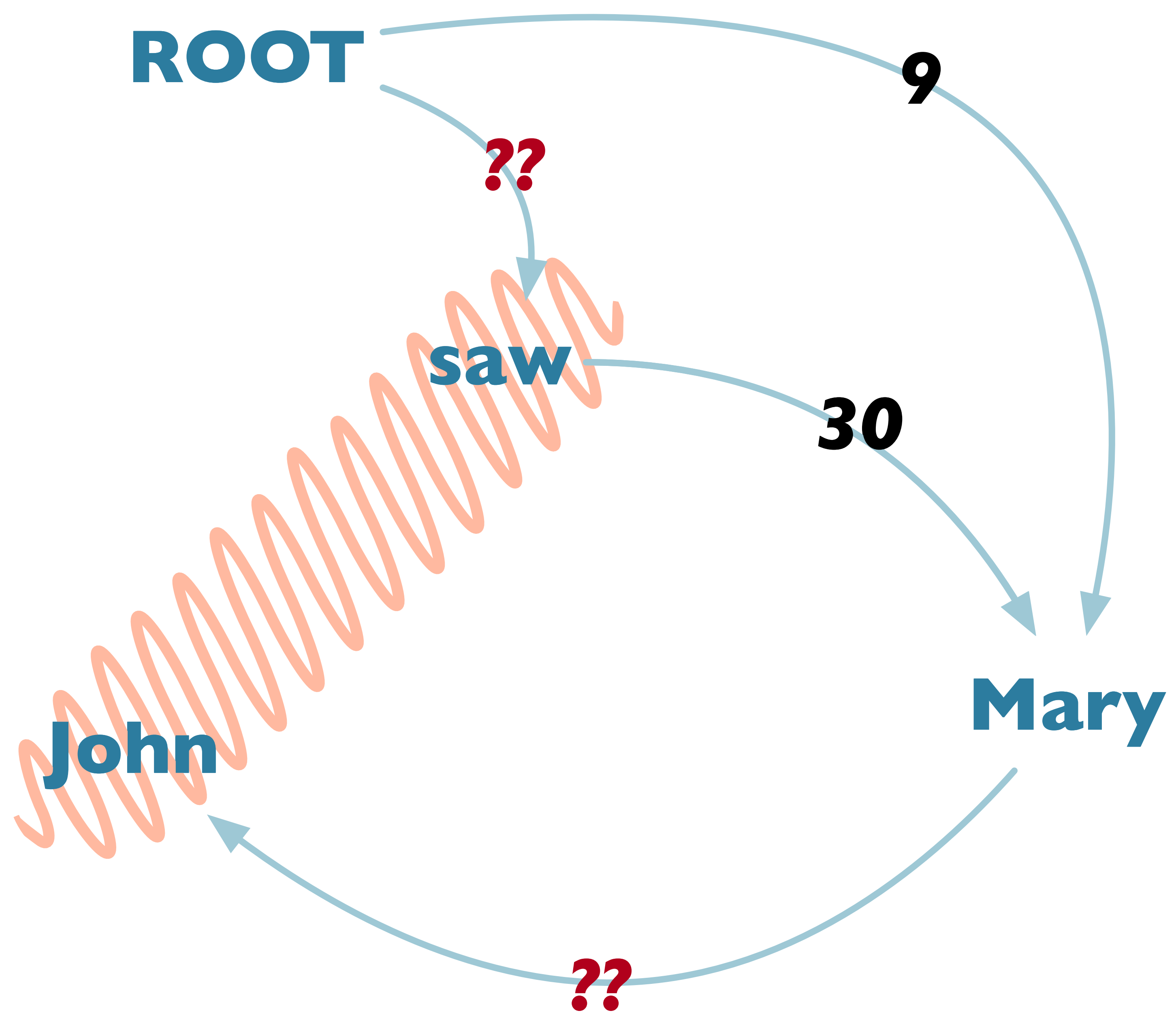
Step 1 & 2

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



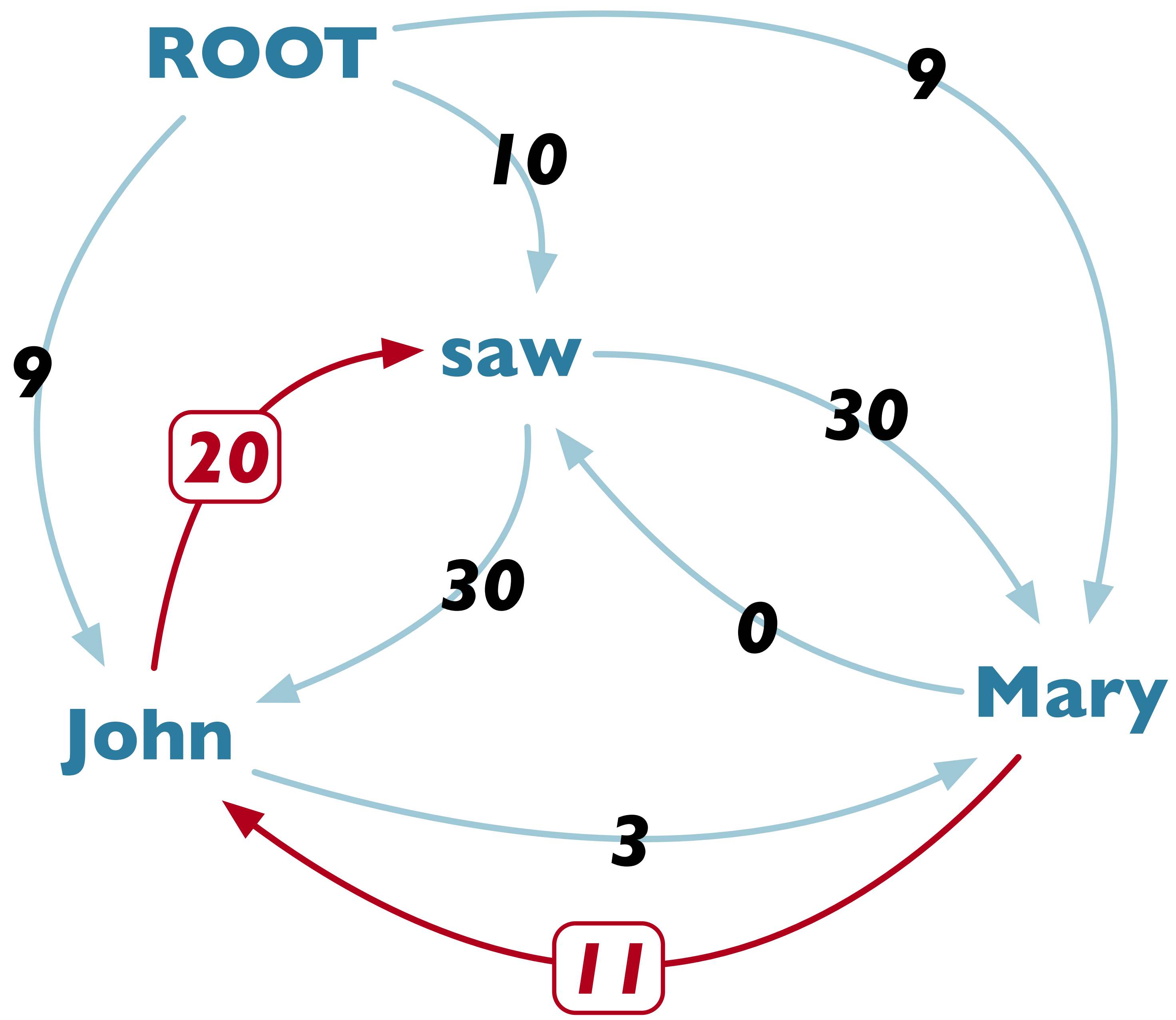
Step 1 & 2

- 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



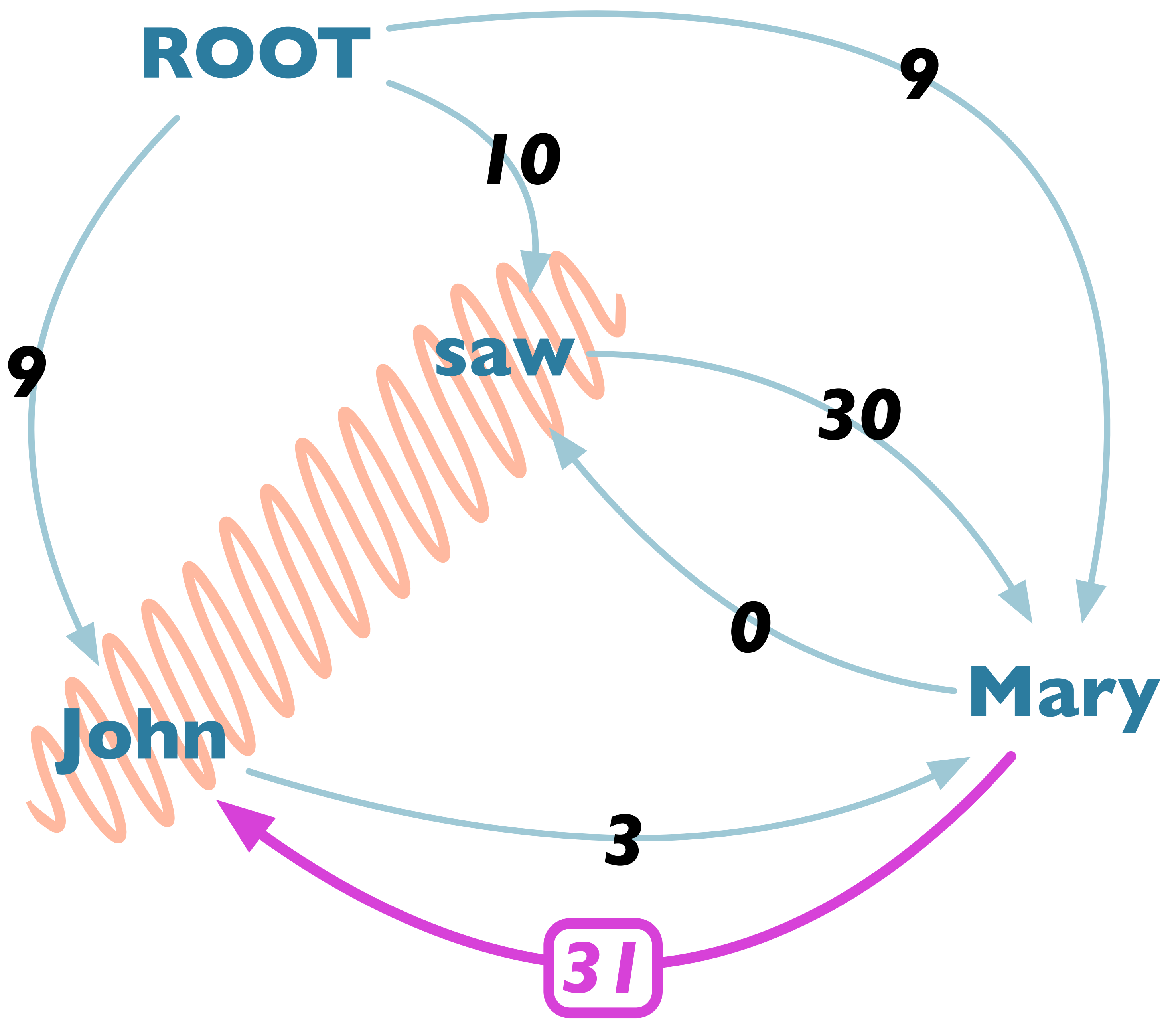
Calculating Weights for Collapsed Vertex

$$\mathrm{s}(\ \mathrm{Mary},\ C\ )\ 11 + 20 = 31$$



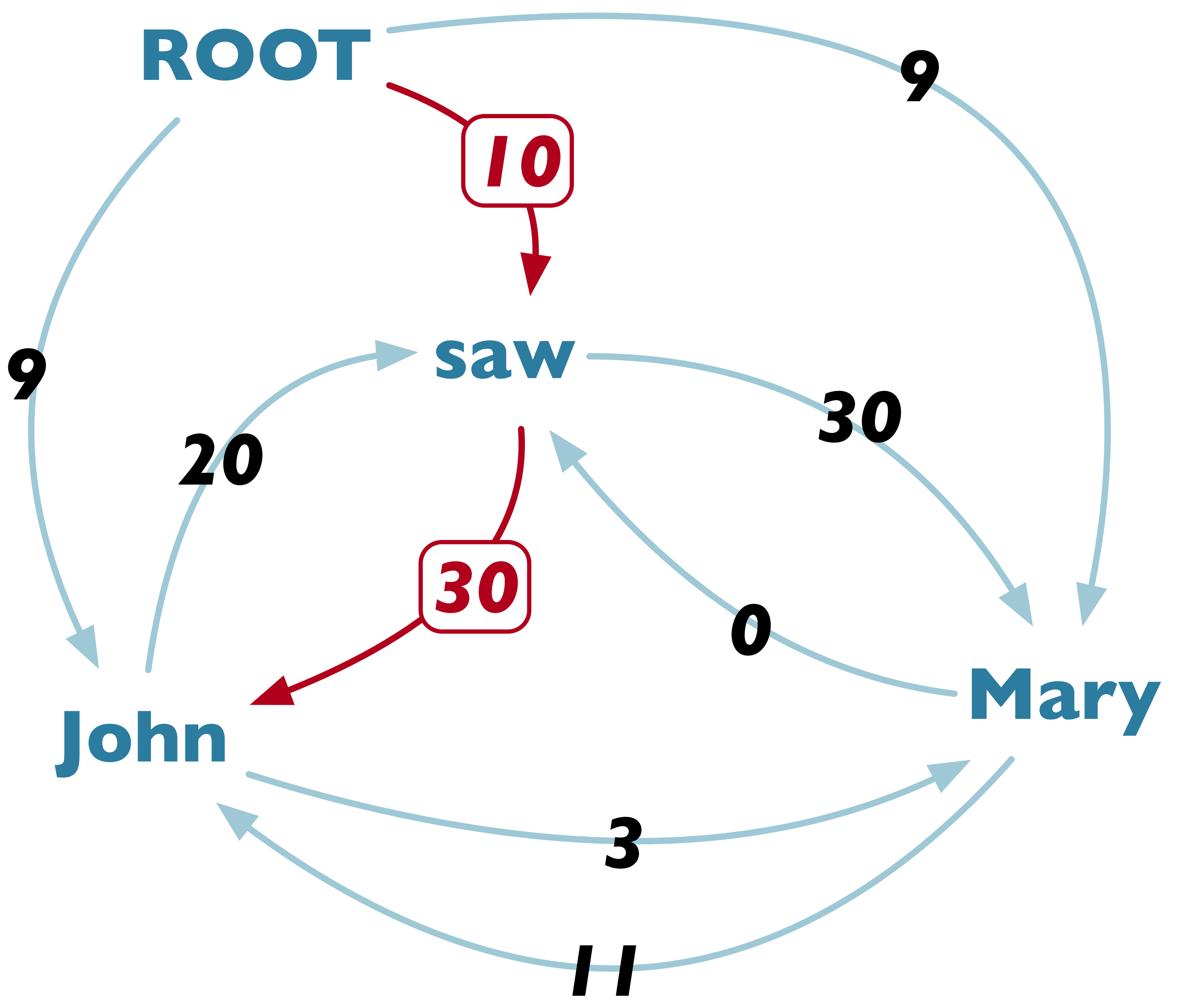
Calculating Weights for Collapsed Vertex

$$\mathrm{s}(\ \mathrm{Mary},\ C\ )\ 11 + 20 = 31$$



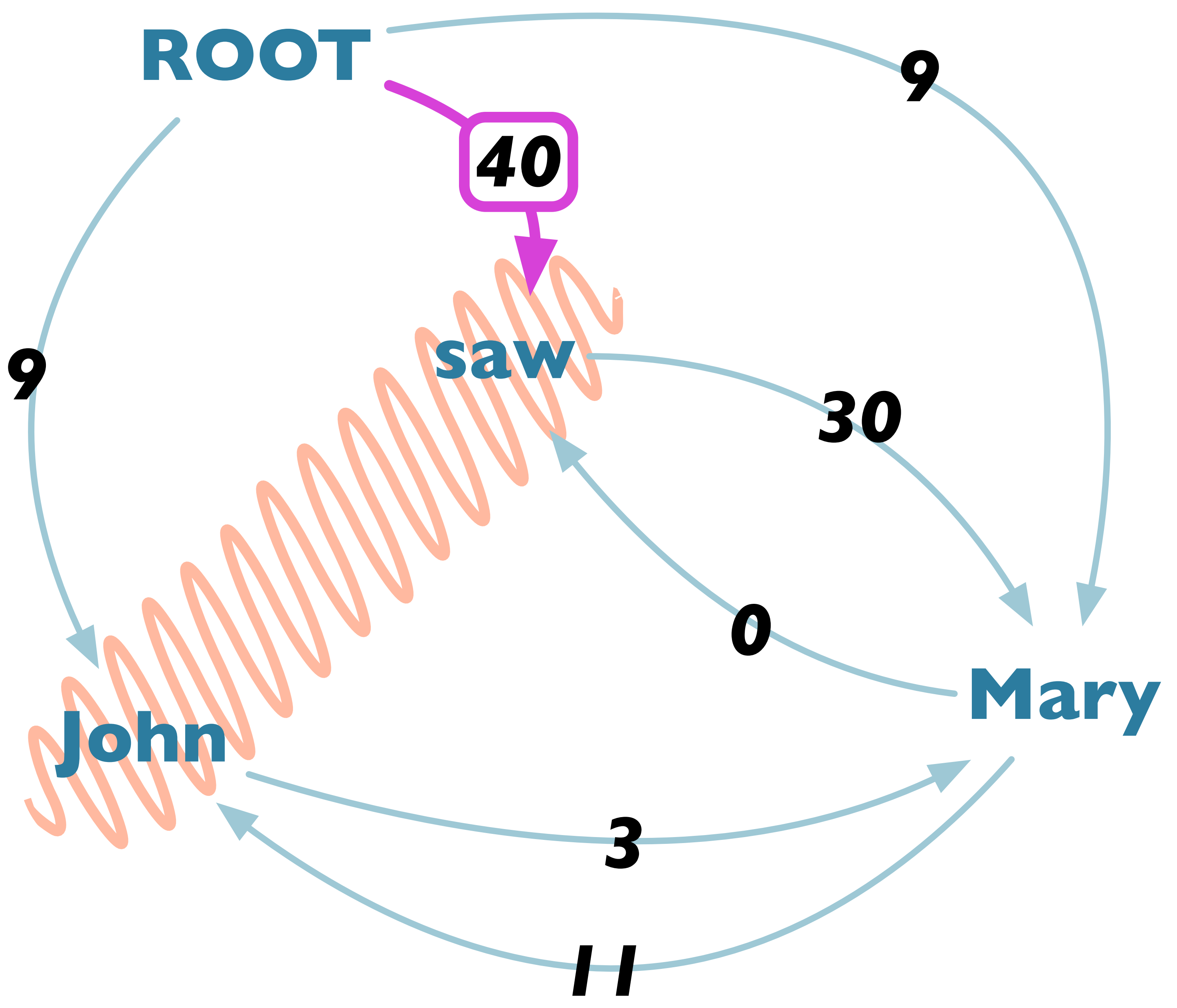
Calculating Weights for Collapsed Vertex

$$s(\text{ ROOT}, C\ )\ 10 + 30 = 40$$



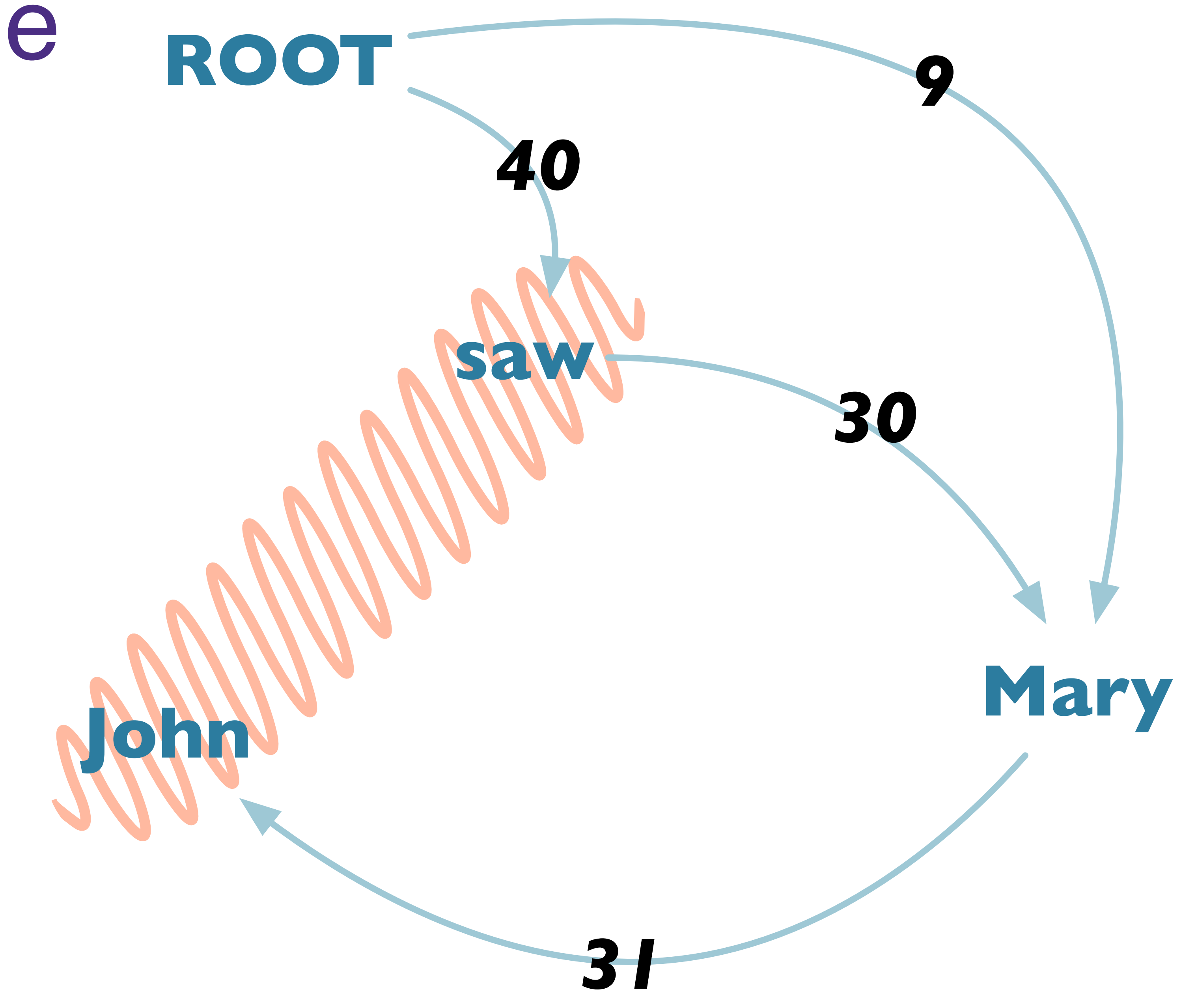
Calculating Weights for Collapsed Vertex

$$\mathrm{s}(\ \mathrm{ROOT},\ C\ )\ 10 + 30 = 40$$



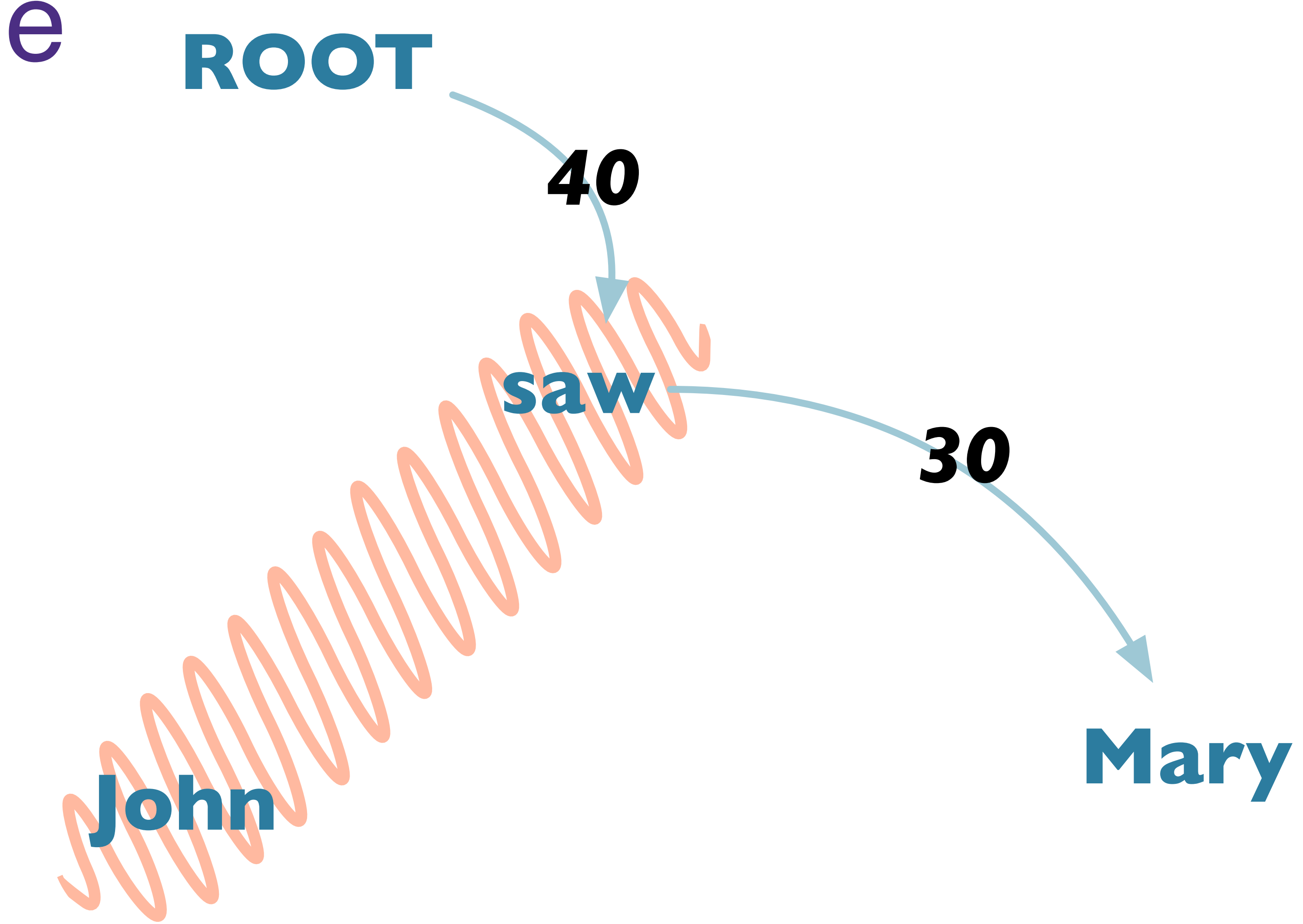
Step 3

- With cycle collapsed, recurse on step 1:
- Keep highest weighted incoming edge for each edge



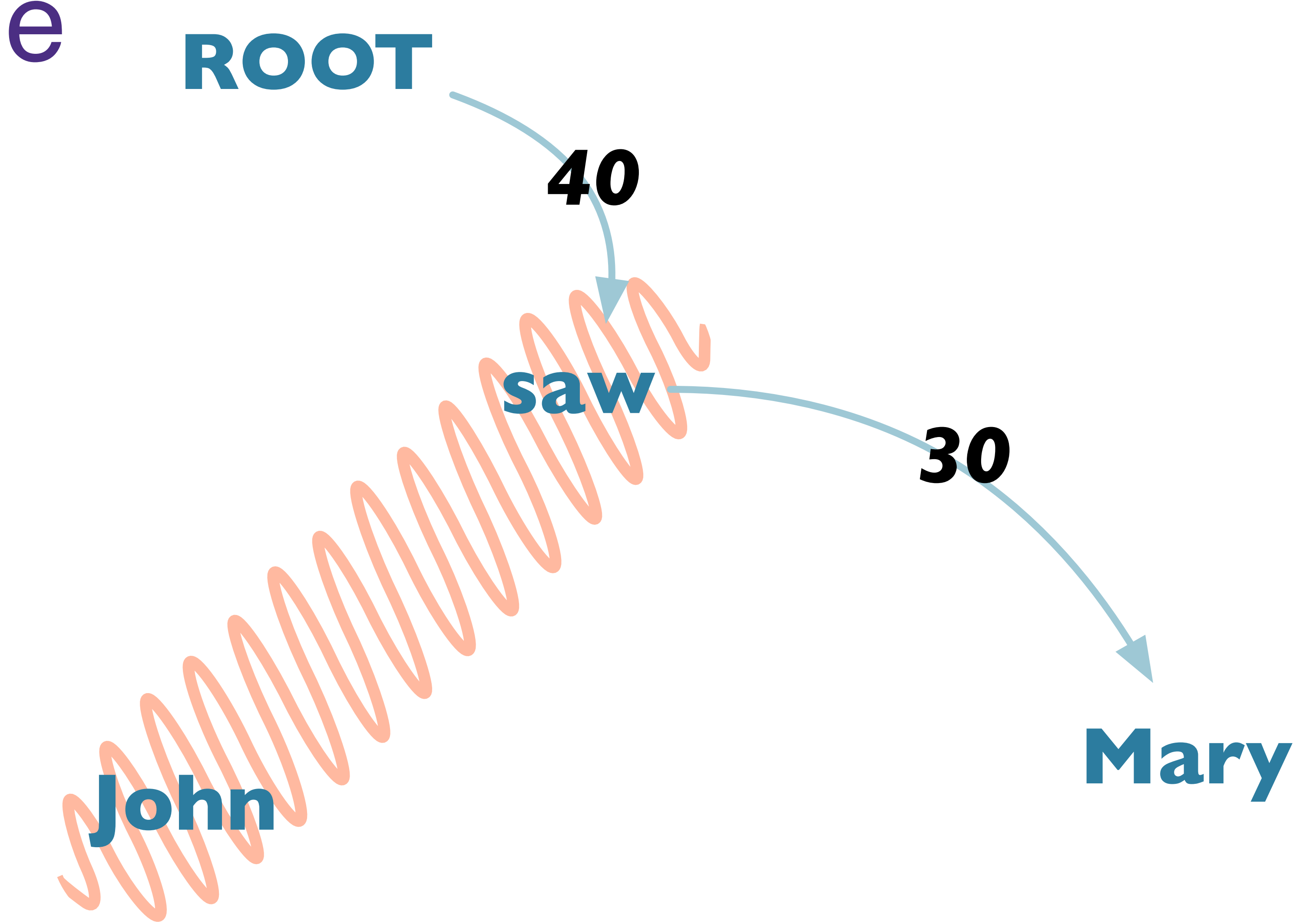
Step 3

- With cycle collapsed, recurse on step 1:
- Keep highest weighted incoming edge for each edge



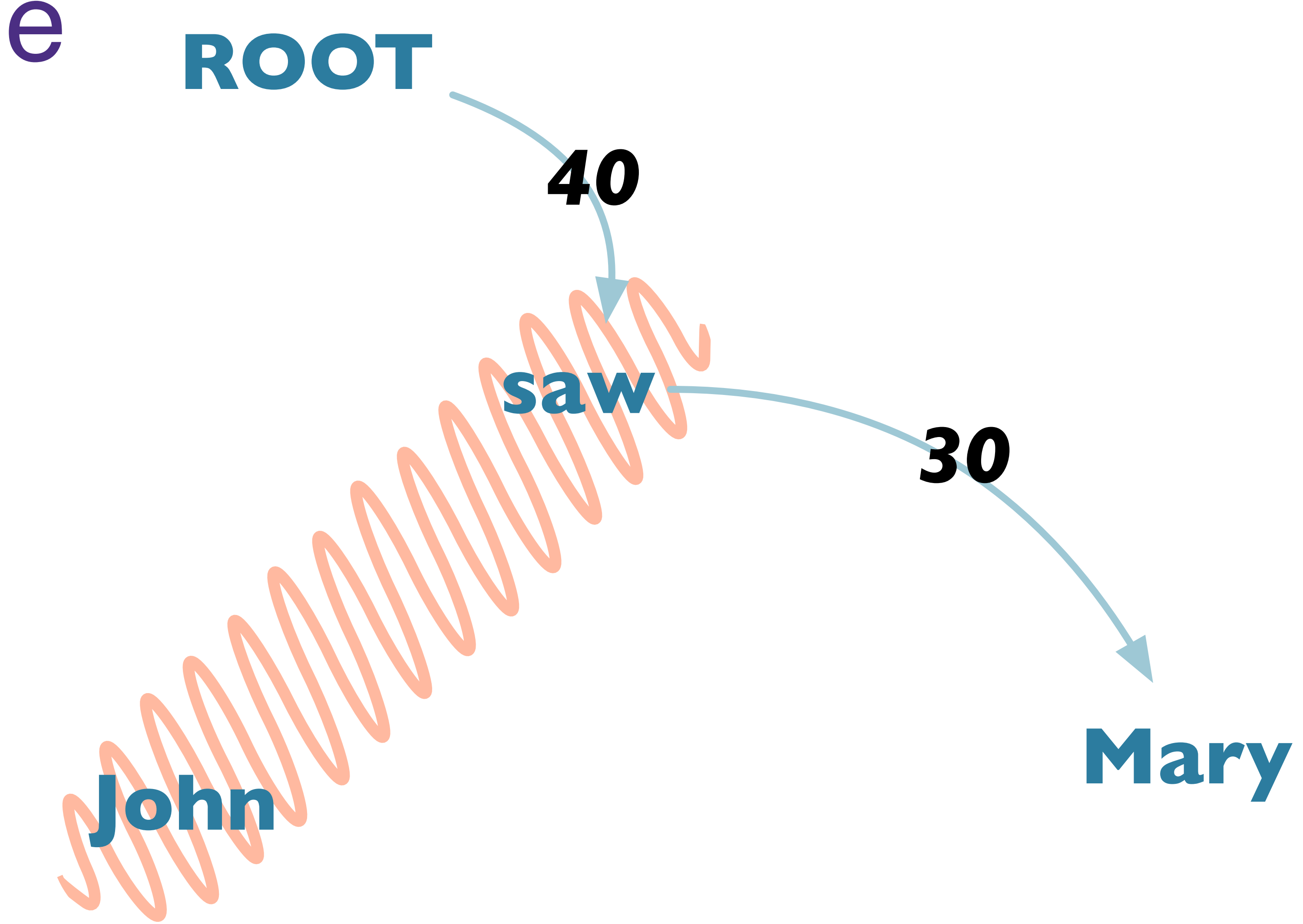
Step 3

- With cycle collapsed, recurse on step 1:
- Keep highest weighted incoming edge for each edge
- Is it a tree?



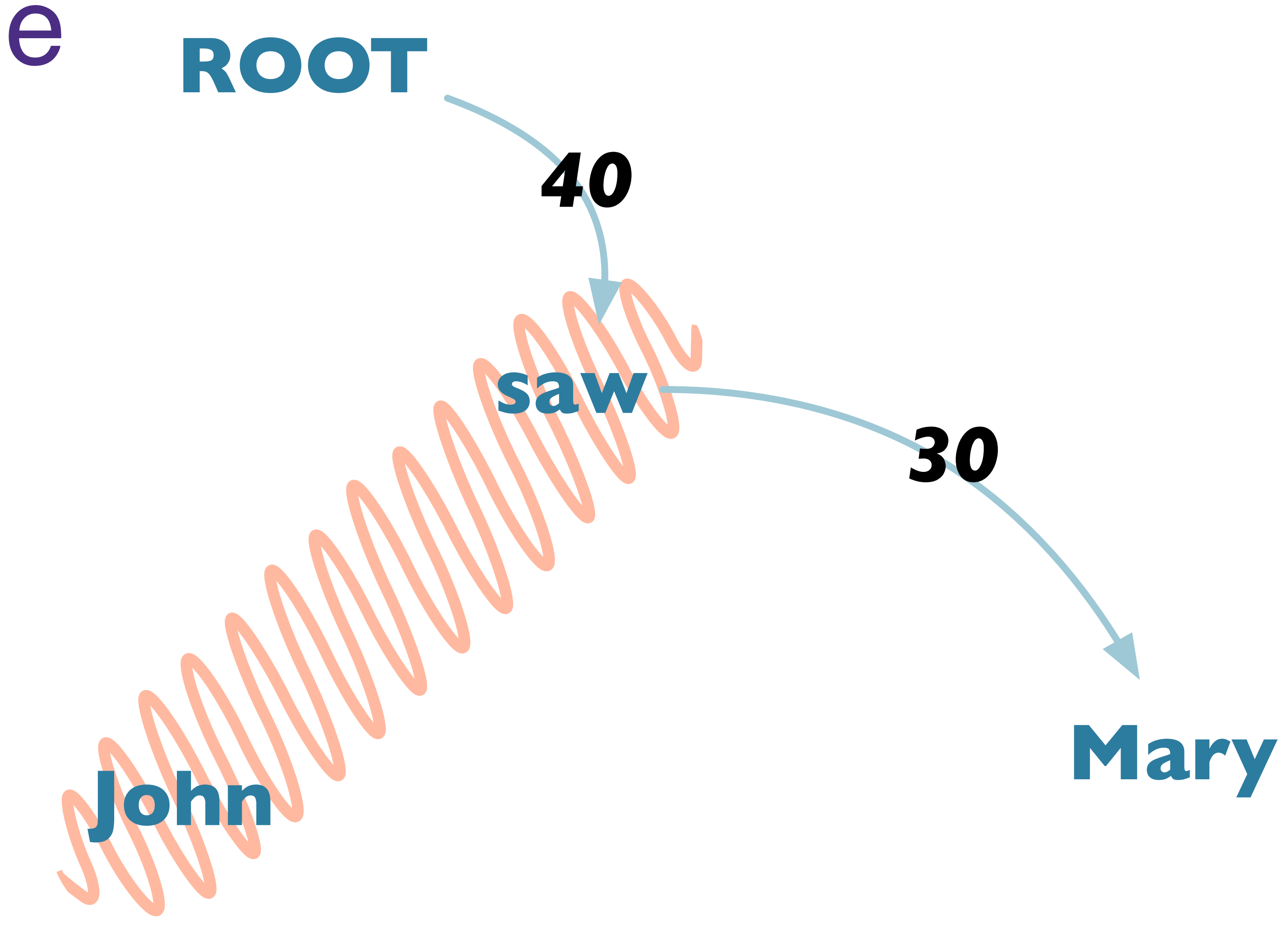
Step 3

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



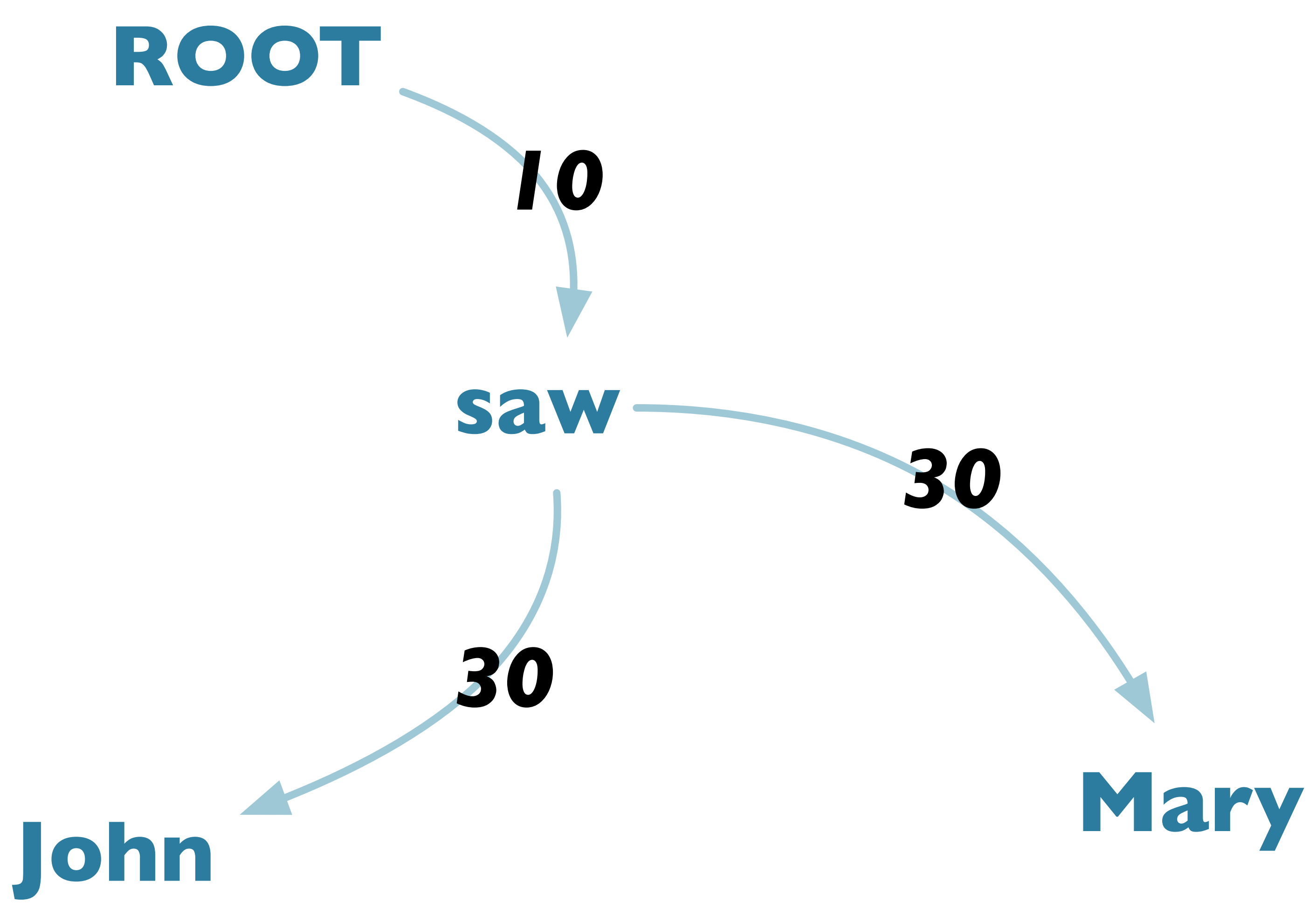
Step 3

- 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.



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*

$F \leftarrow []$
$T' \leftarrow []$
$score' \leftarrow []$
for each $v \in V$ **do**
  $bestInEdge \leftarrow \text{argmax}_{e=(u,v)\in E}\ score[e]$
  $F \leftarrow F \cup bestInEdge$
  for each $e{=}(u{,}v) \in E$ **do**
    $score'[e] \leftarrow score[e] - score[bestInEdge]$

  if $T{=}(V{,}F)$ is a spanning tree **then return** it
  else
    $C \leftarrow$ a cycle in F
    $G' \leftarrow$ CONTRACT(G, C)
    $T' \leftarrow$ MAXSPANNINGTREE(G', *root*, *score'*)
    $T \leftarrow$ 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 $w_{i\text{-}1}$, POS tag of w_{i+1}
  - POS tag of $w_{j\text{-}1}$, POS tag of w_{j+1}

- Features conjoined with direction of attachment and distance between words

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]**