Dependency Grammars and Parser

LING 571 — Deep Processing for NLP

October 16, 2019

Shane Steinert-Threlkeld

Ambiguity of the Week

Adam Macqueen
@adam_macqueen

Personally feel not enough hospitals are named after sandwiches.

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 evaluation

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

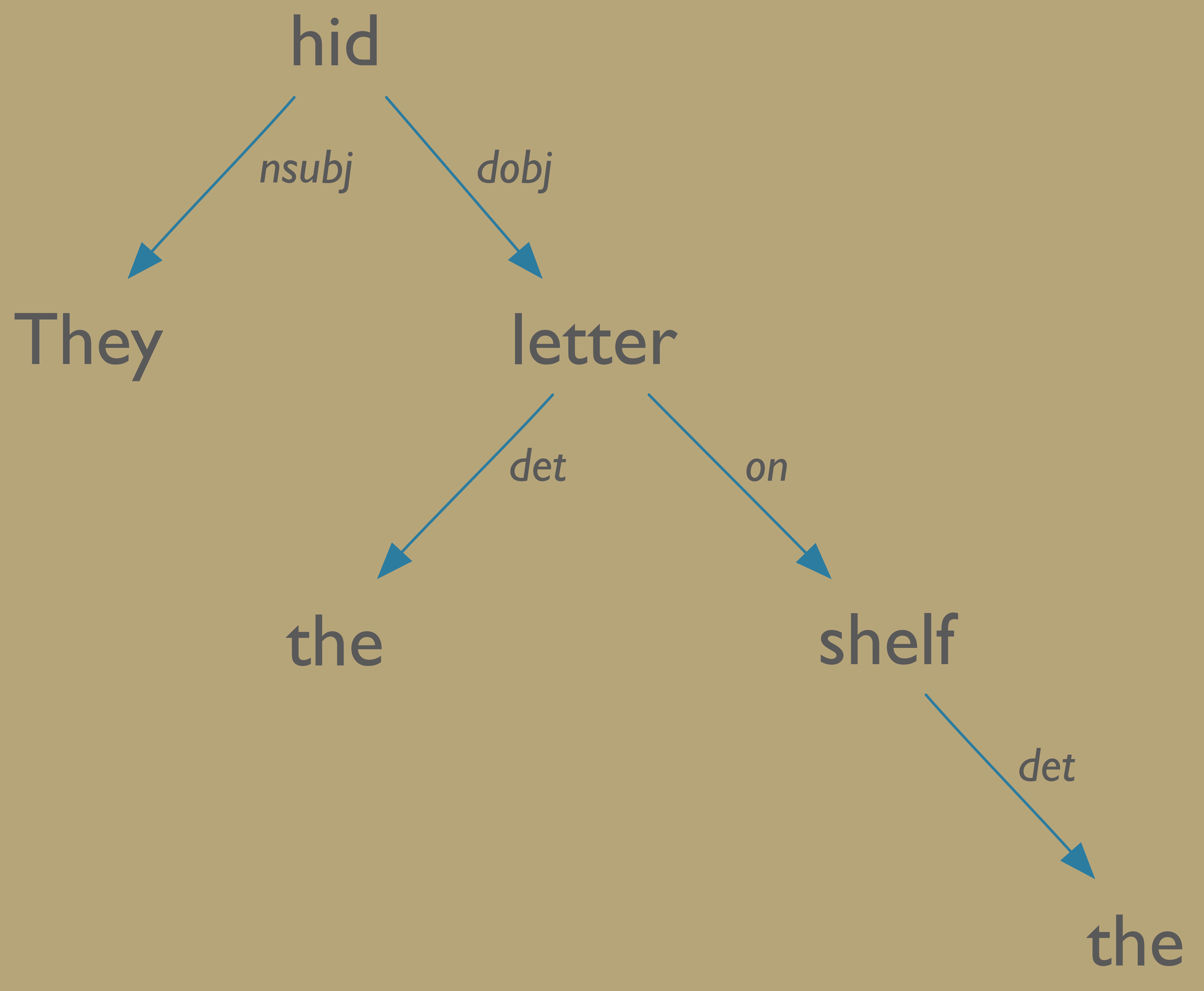
- A Dependency parse is a tree,* where:
  - Nodes correspond to words in string
  - Edges between nodes represent dependency relations
    - Relations may or may not be labeled (aka typed)
  - *: in very special cases, can argue for cycles

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

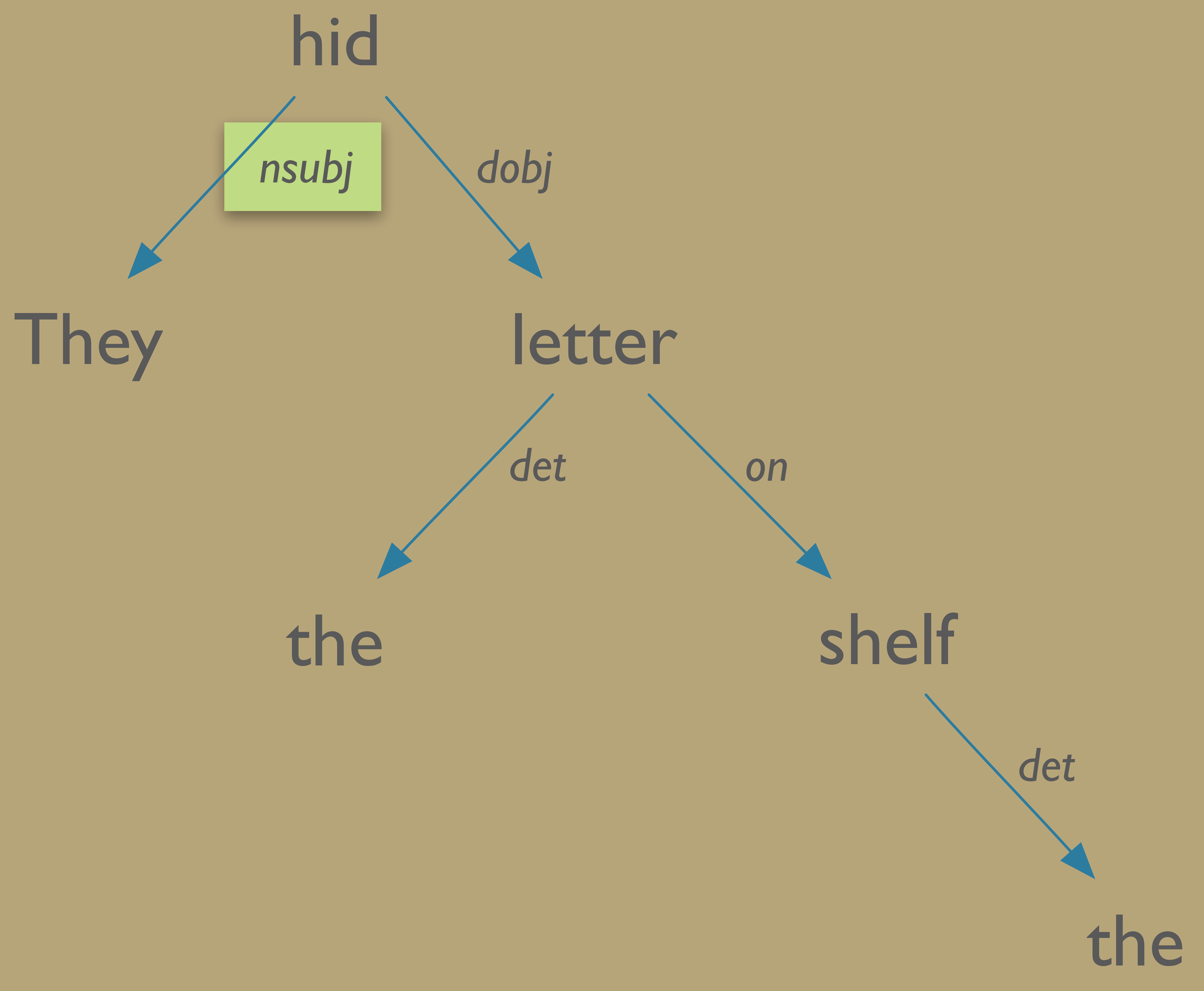


Dependency Parse Example:

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

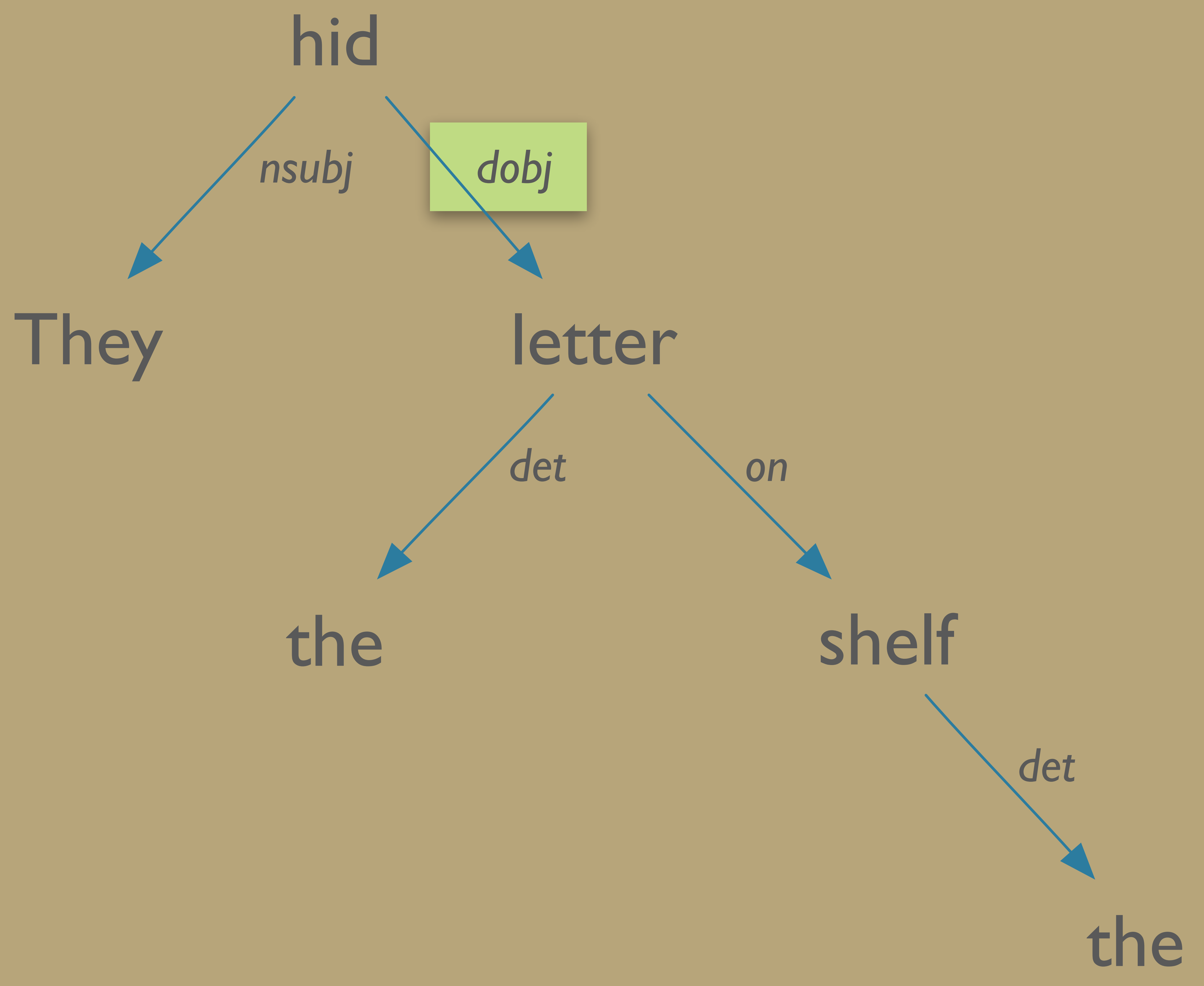


Dependency Parse Example:

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

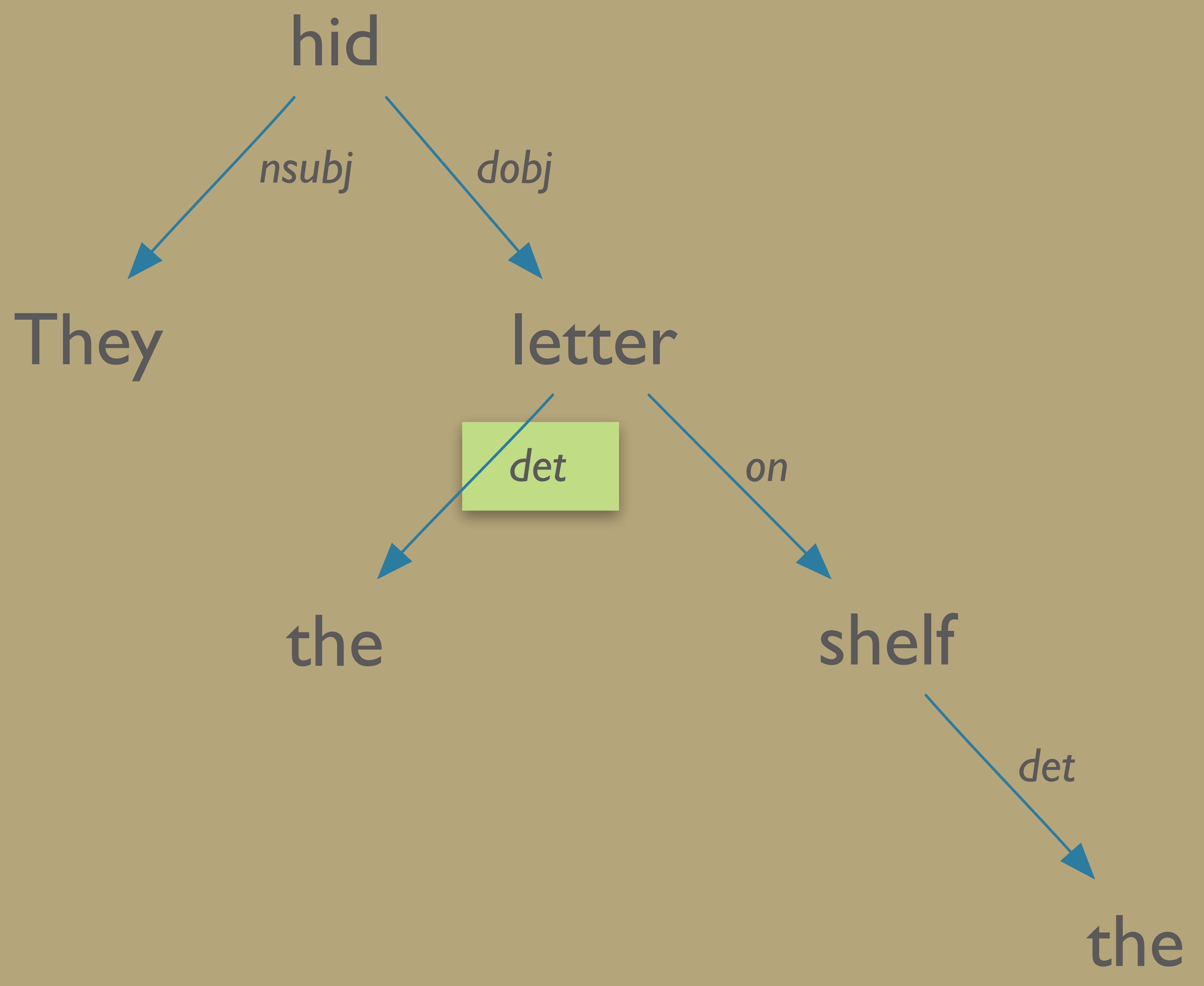


Dependency Parse Example:

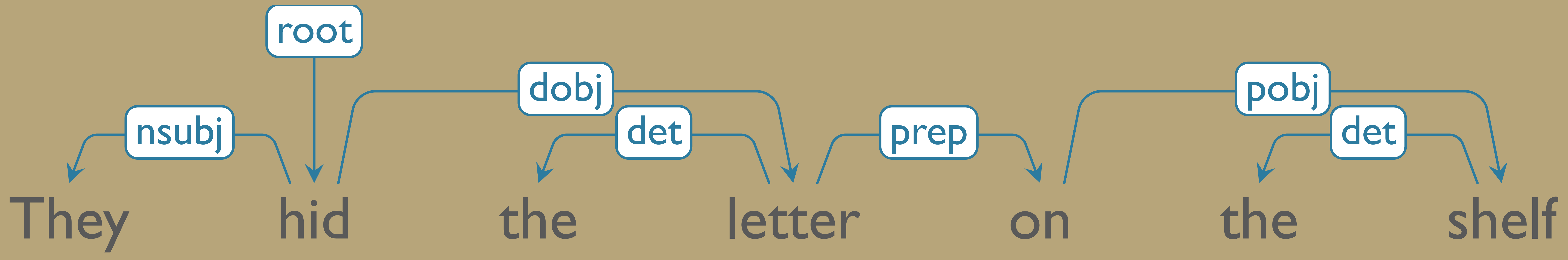
They hid the letter on the shelf

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

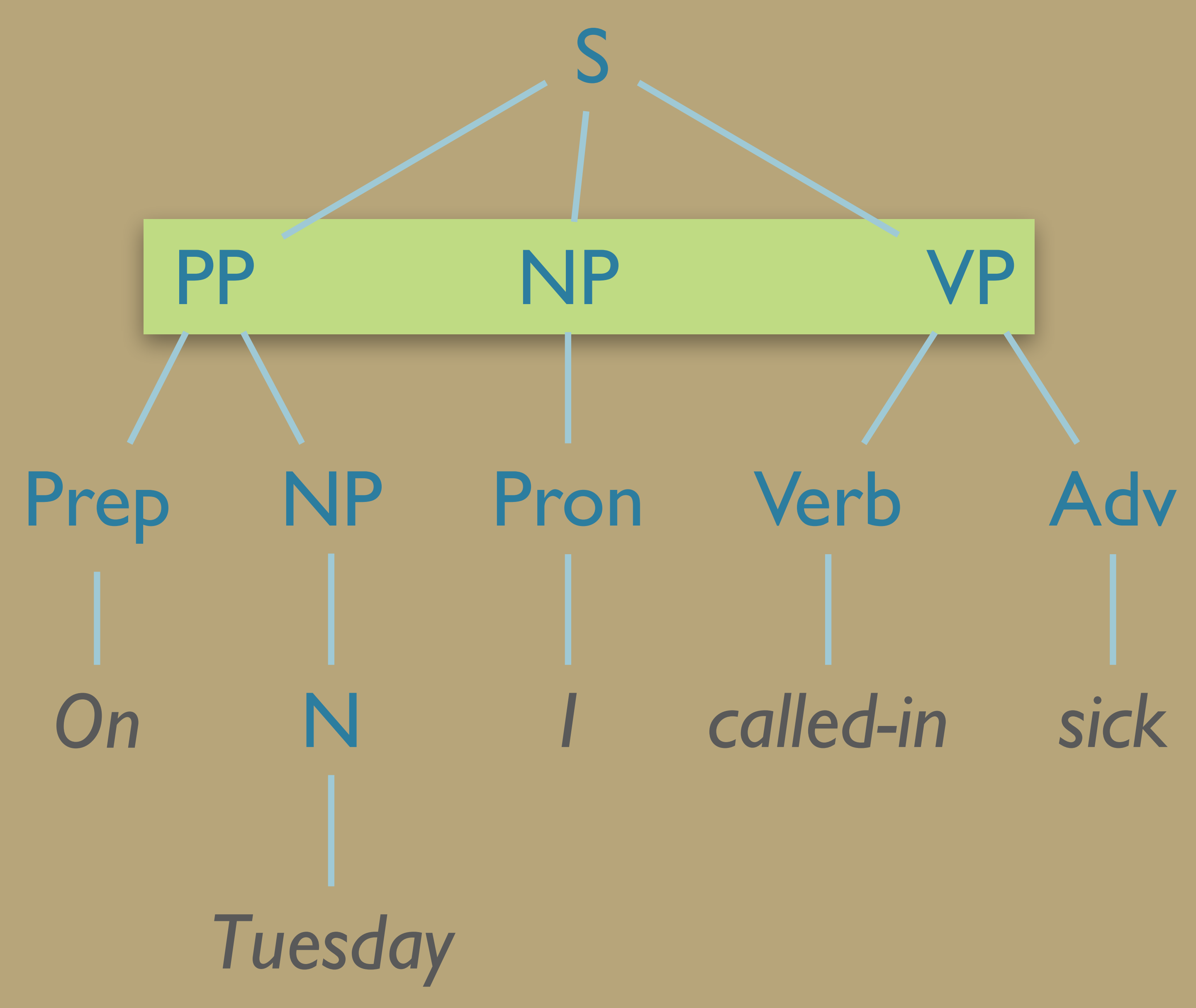


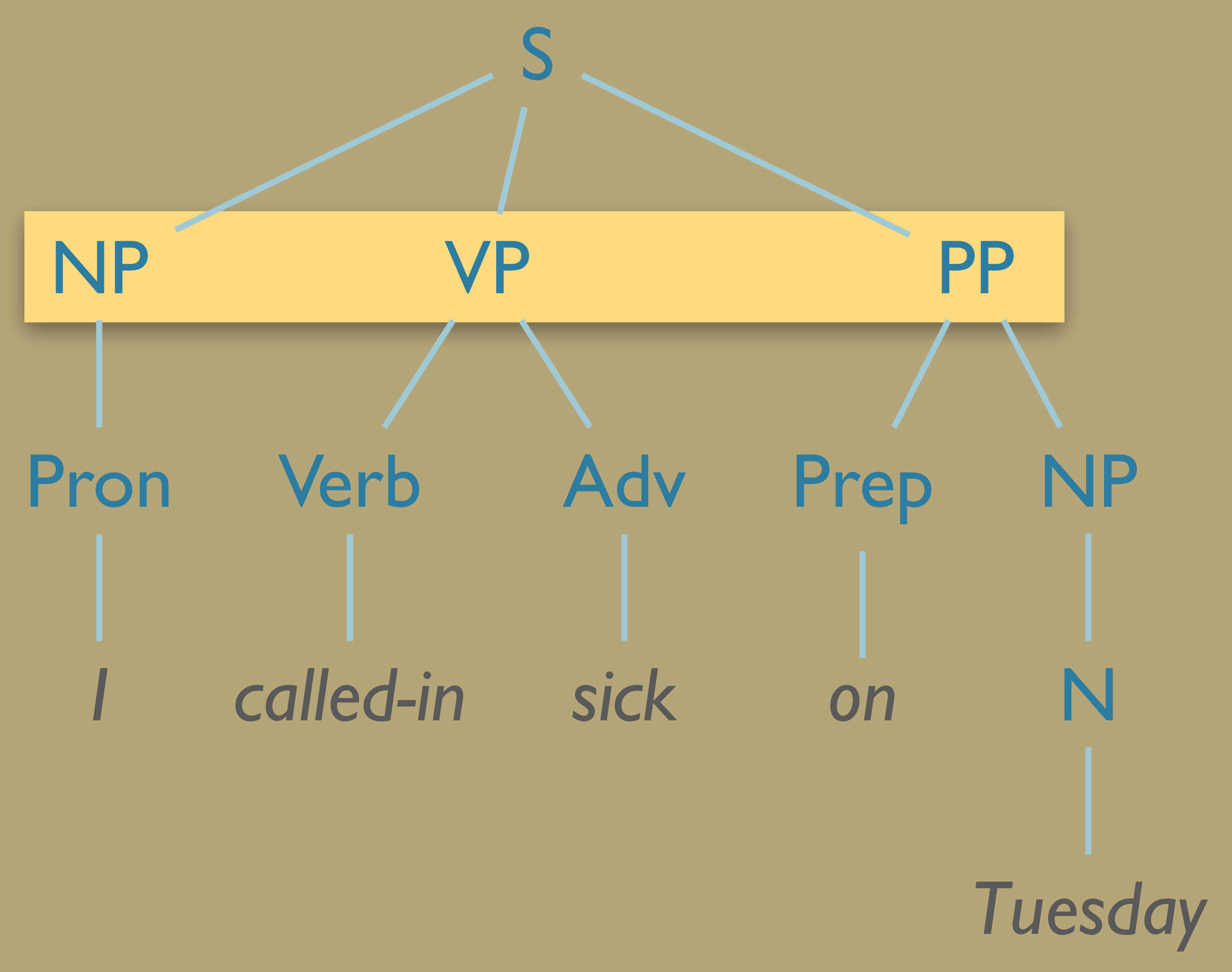
Why Dependency Grammar?

- 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?



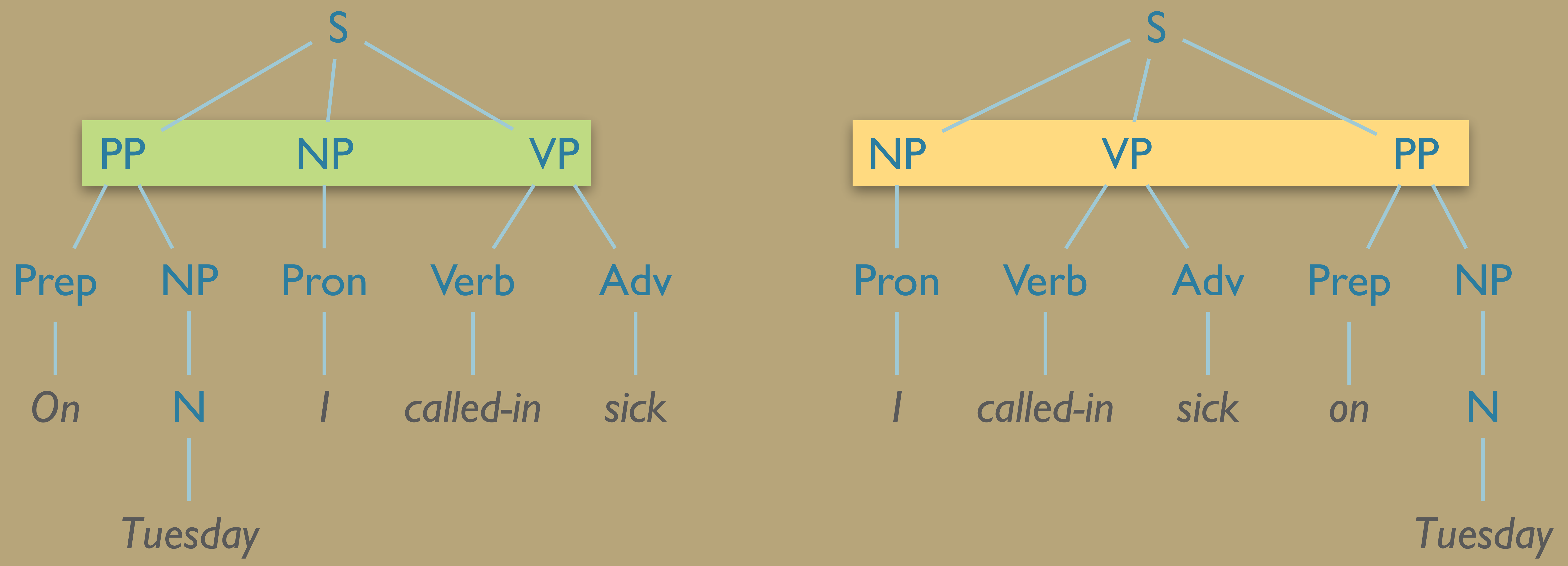


$S \rightarrow PP\ NP\ VP$

$S \rightarrow NP\ VP\ PP$

Why Dependency Grammar?

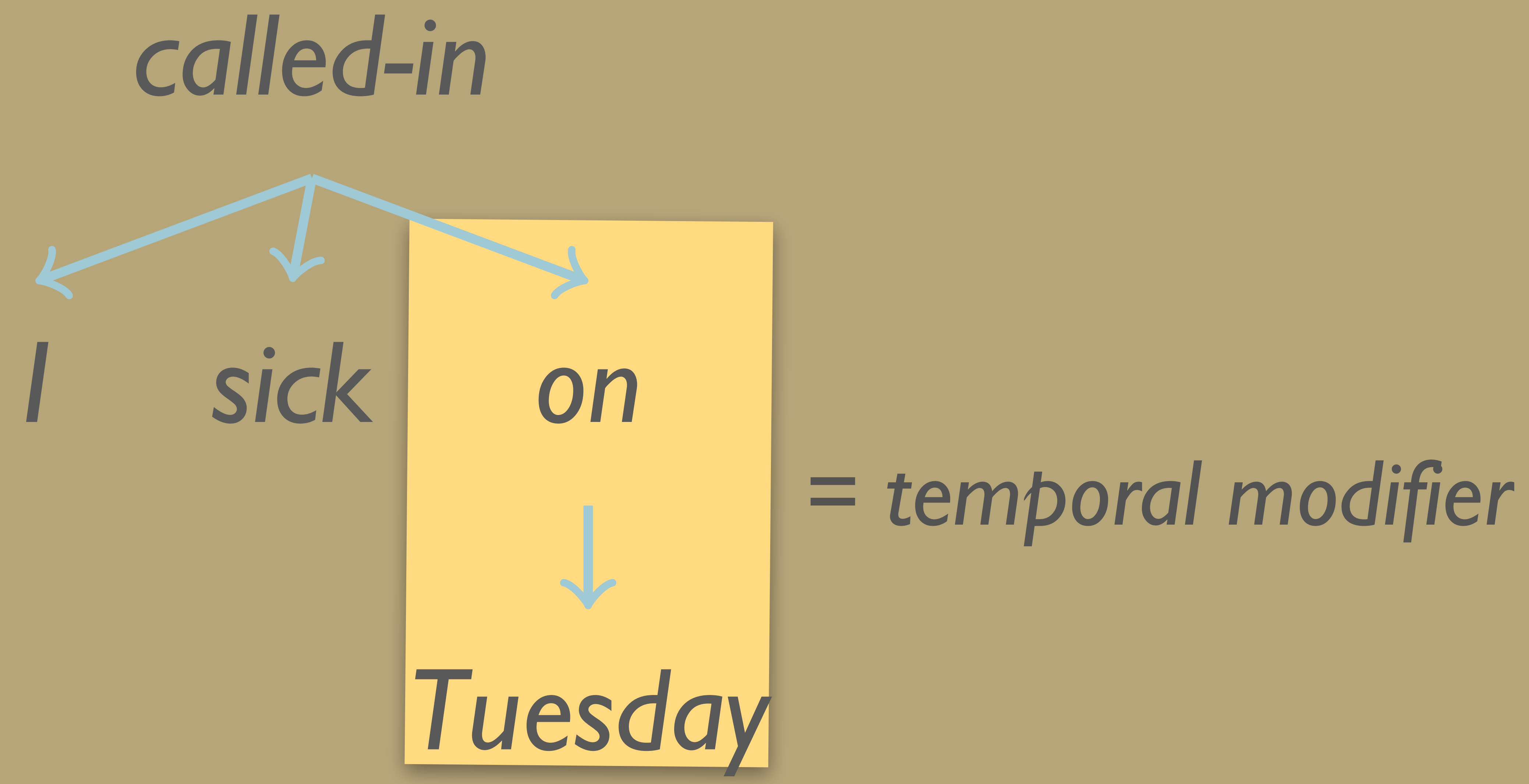
- English has relatively fixed word order
- Big problem for languages with freer word order



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

Why Dependency Grammar?

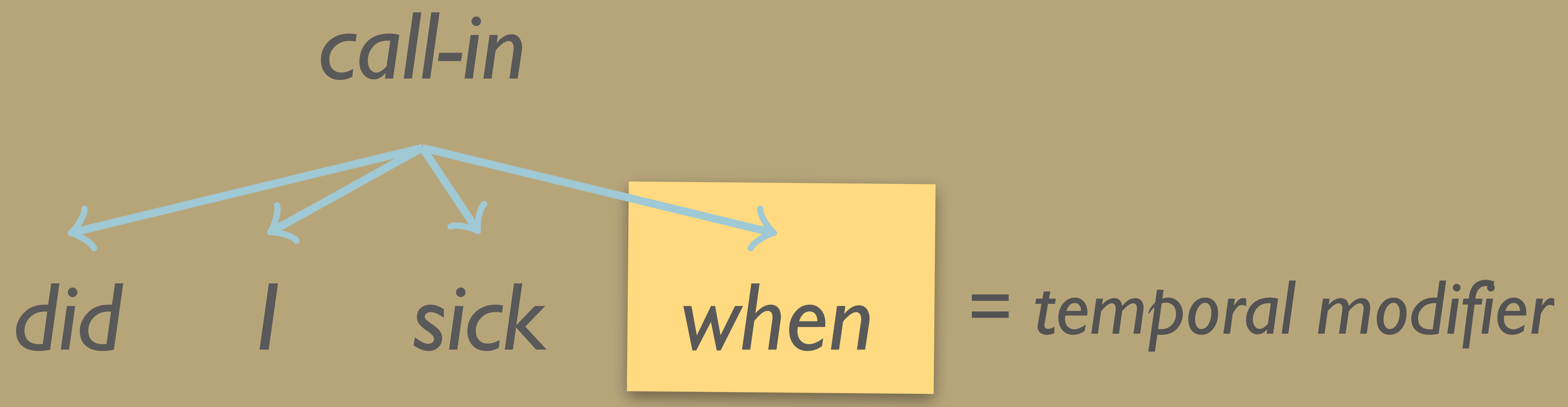
- How do dependency structures represent the difference?
  - Same structure
  - Relationships are between words, order insensitive



I called in sick on Tuesday

Why Dependency Grammar?

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



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 [stanfordnlp]() Python packages have good built-in parsers
- LaTeX: tiki-tikz-dependency (https://ctan.org/pkg/tikz-dependency)

```
>>> import stanfordnlp
>>> stanfordnlp.download('en')   # This downloads the English models for the neural pipeline
>>> nlp = stanfordnlp.Pipeline() # This sets up a default neural pipeline in English
>>> doc = nlp("Barack Obama was born in Hawaii.  He was elected president in 2008.")
>>> doc.sentences[0].print_dependencies()
```

```
('Barack', '4', 'nsubj:pass')
('Obama', '1', 'flat')
('was', '4', 'aux:pass')
('born', '0', 'root')
('in', '6', 'case')
('Hawaii', '4', 'obl')
('.', '4', 'punct')
```

Resources

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

Upcoming UD Languages

Assamese	1	–	IE, Indic
Bengali	2	–	IE, Indic
Bhojpuri	1	–	IE, Indic
Cusco Quechua	1	–	Quechuan
Dargwa	1	–	Nakho-Dagestanian
Georgian	1	–	Kartvelian
Kannada	1	–	Dravidian, Southern
Komi Permyak	1	–	Uralic, Permic
Kyrgyz	1	–	Turkic, Northwestern
Livvi	1	–	Uralic, Finnic
Macedonian	1	–	IE, Slavic
Pnar	1	–	Austro-Asiatic, Khasian
Romansh	2	–	IE, Romance
Scottish Gaelic	1	–	IE, Celtic
Shipibo Konibo	1	–	Panoan
Sindhi	1	–	IE, Indic
Somali	1	–	Afro-Asiatic, Cushitic
Sorani	1	–	IE, Iranian
Swiss German	1	–	IE, Germanic

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.

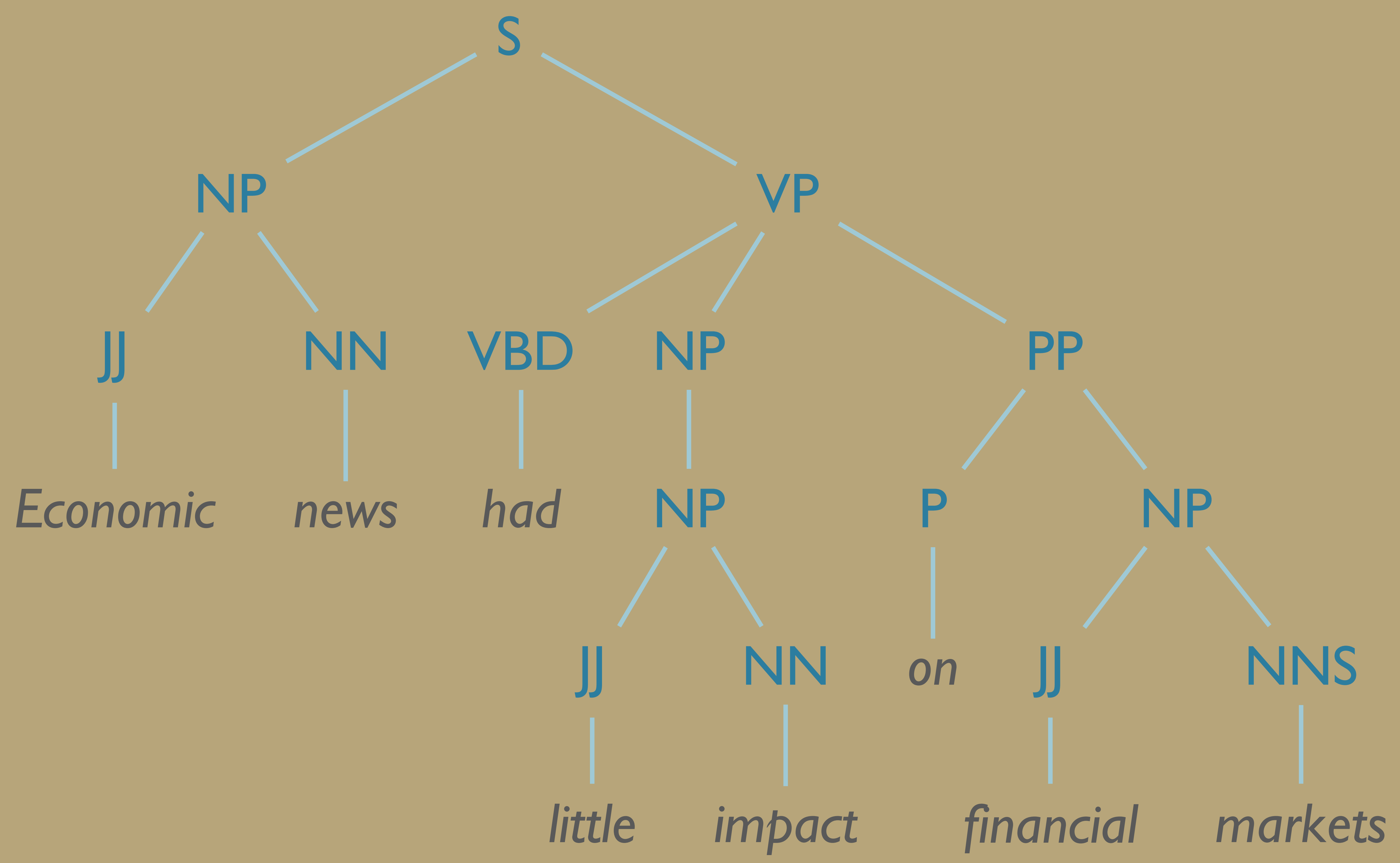
Roadmap

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

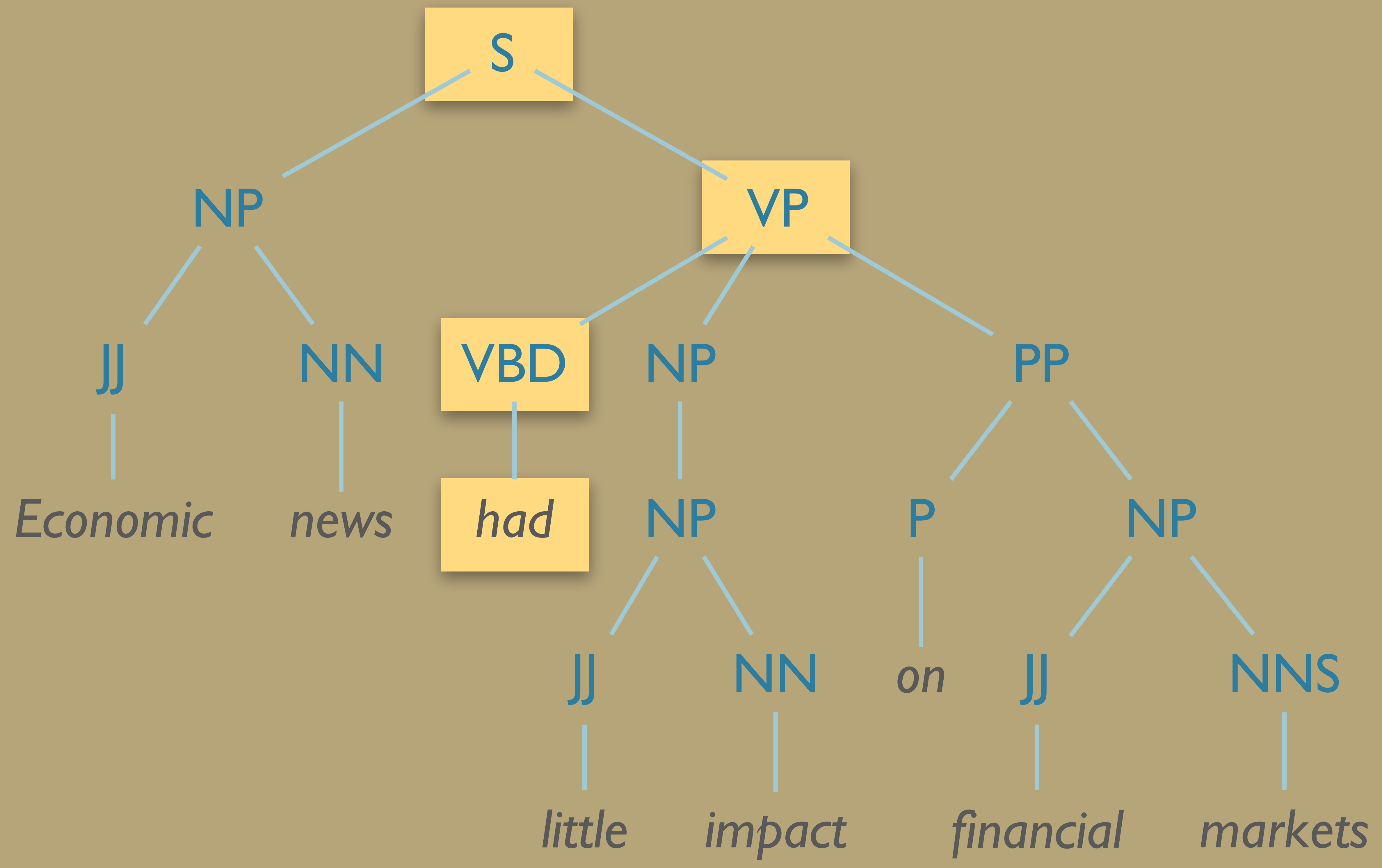
Conversion: PS → DS

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

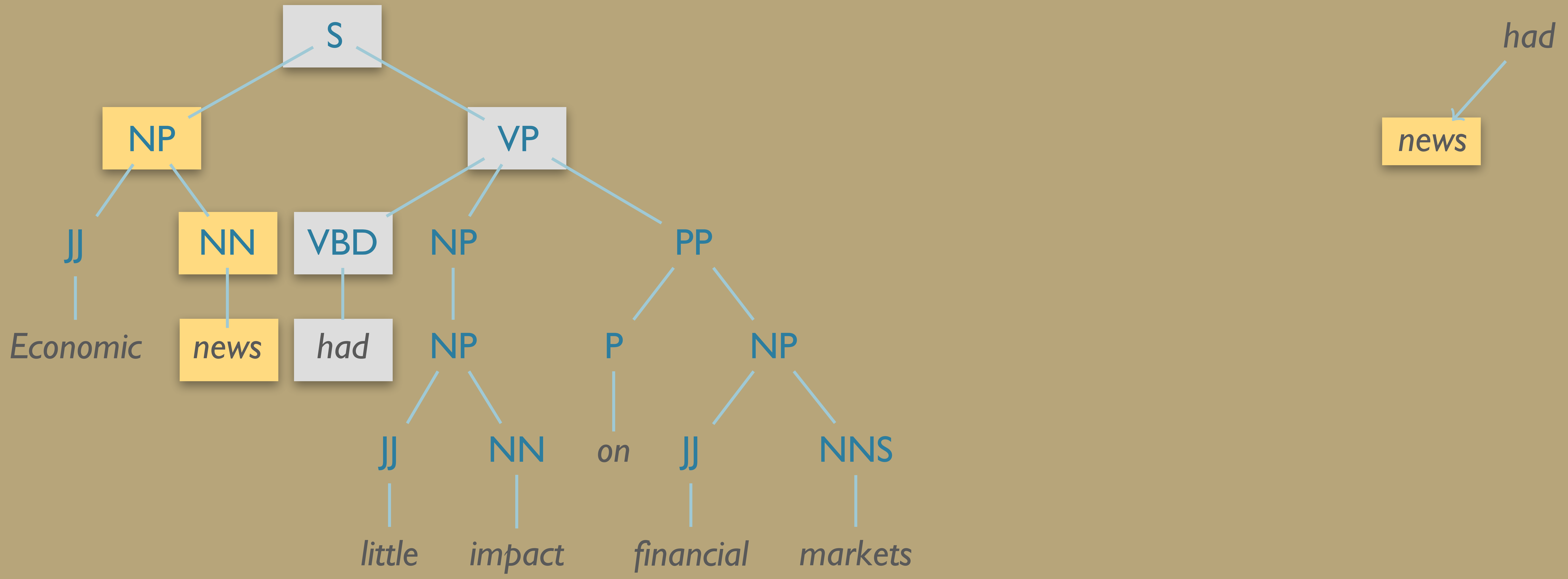
Conversion: PS → DS



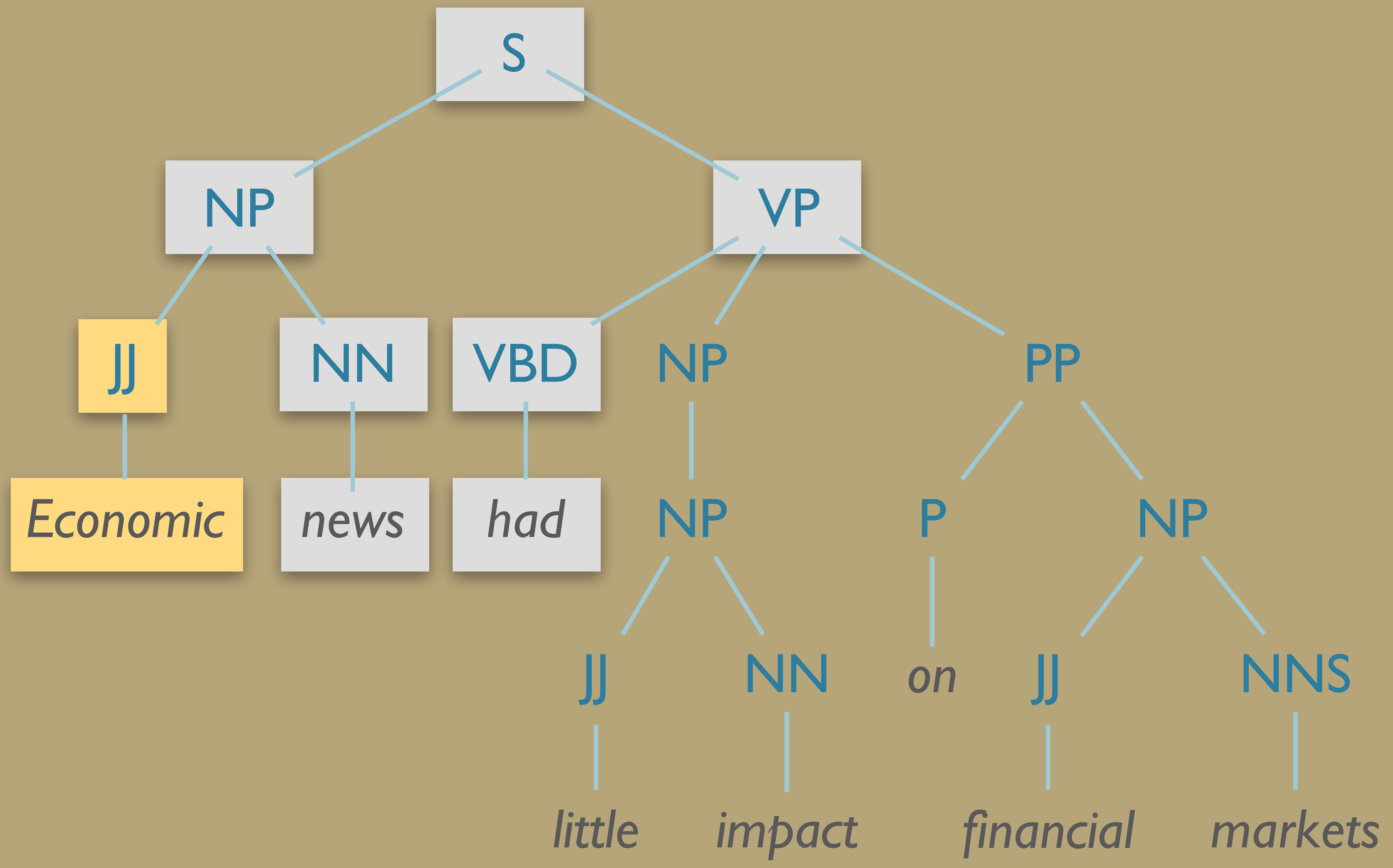
Conversion: PS → DS

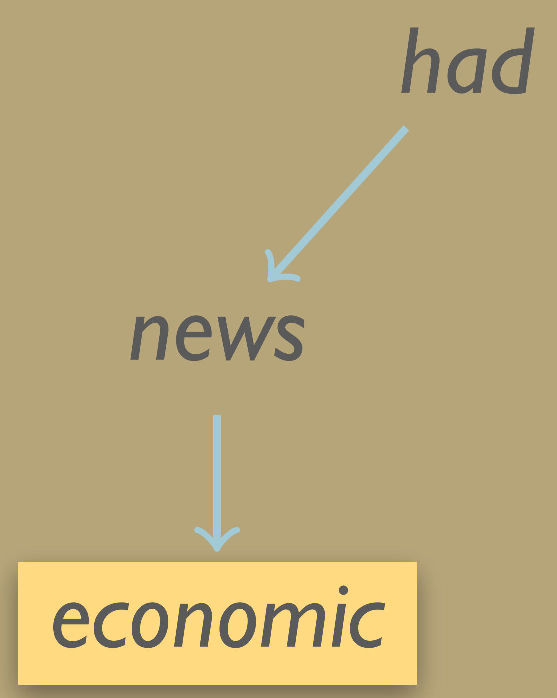


Conversion: PS → DS

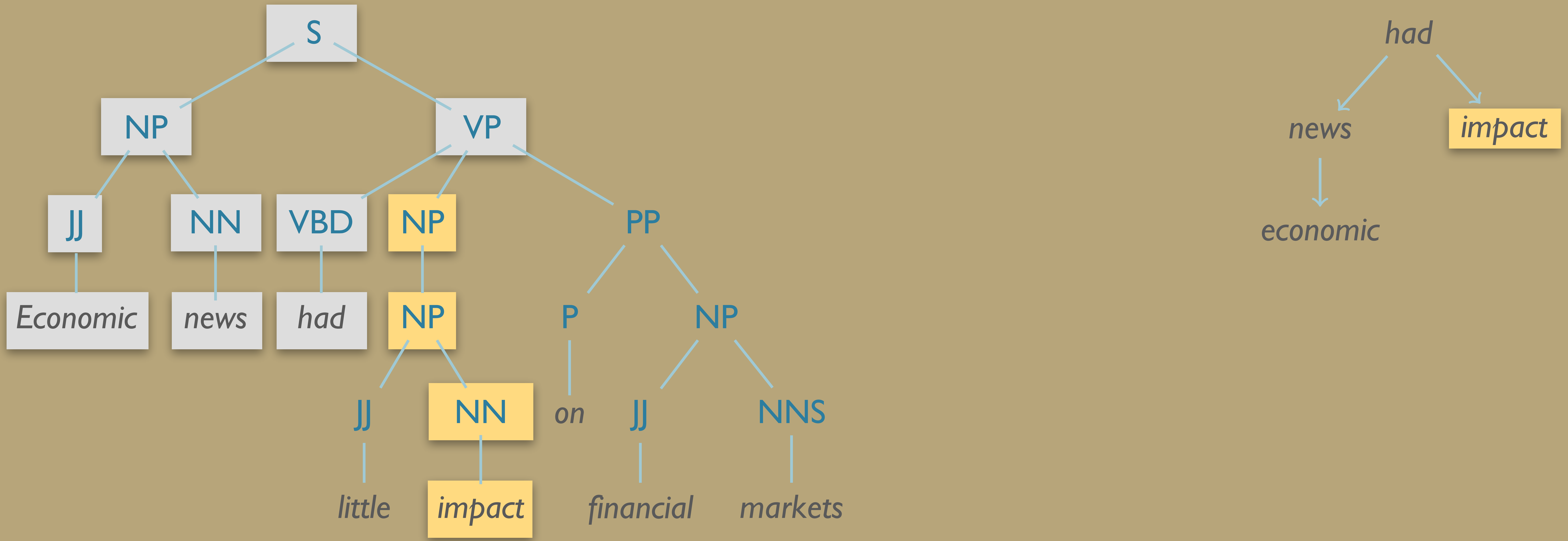


Conversion: PS → DS

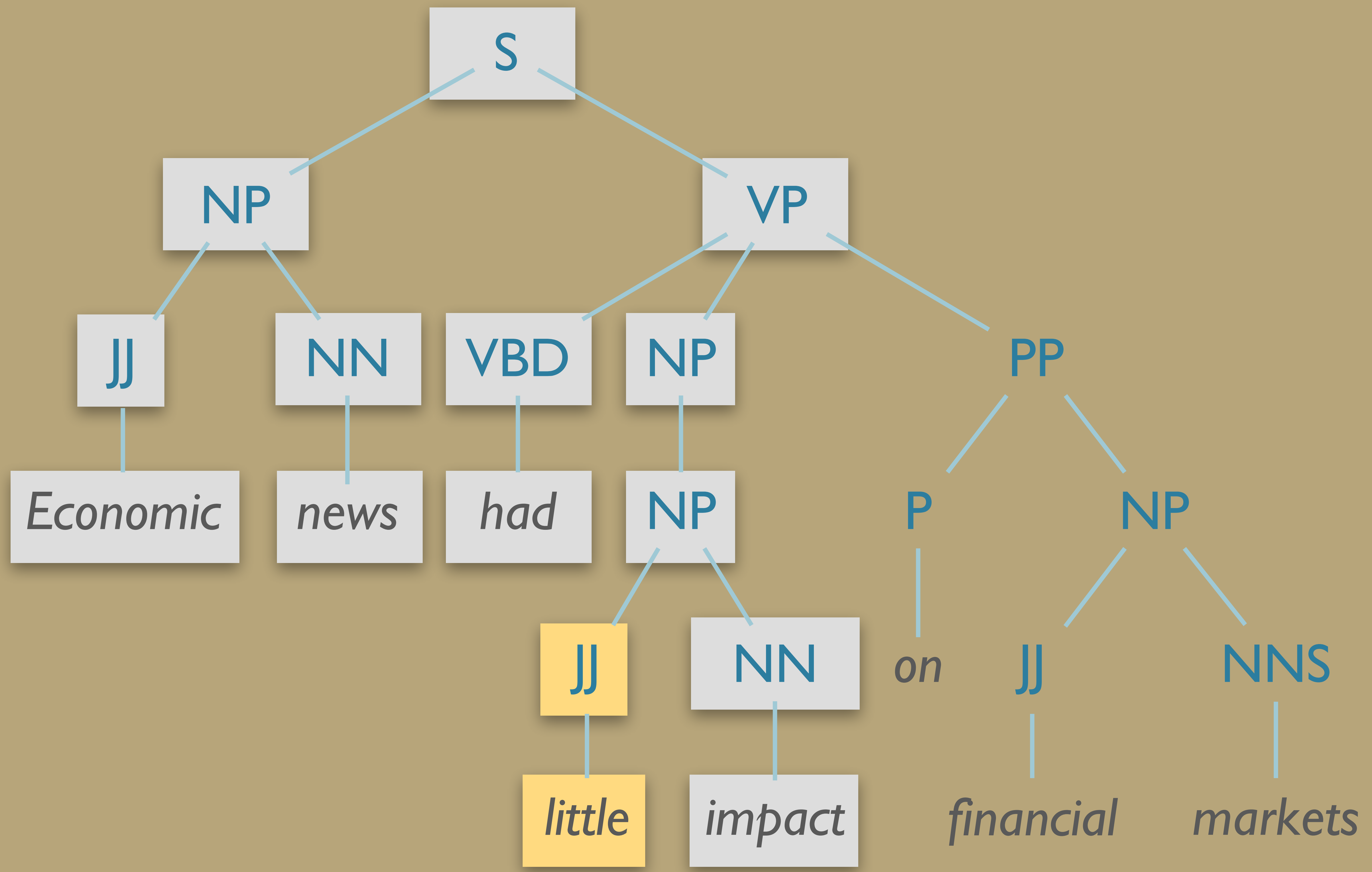


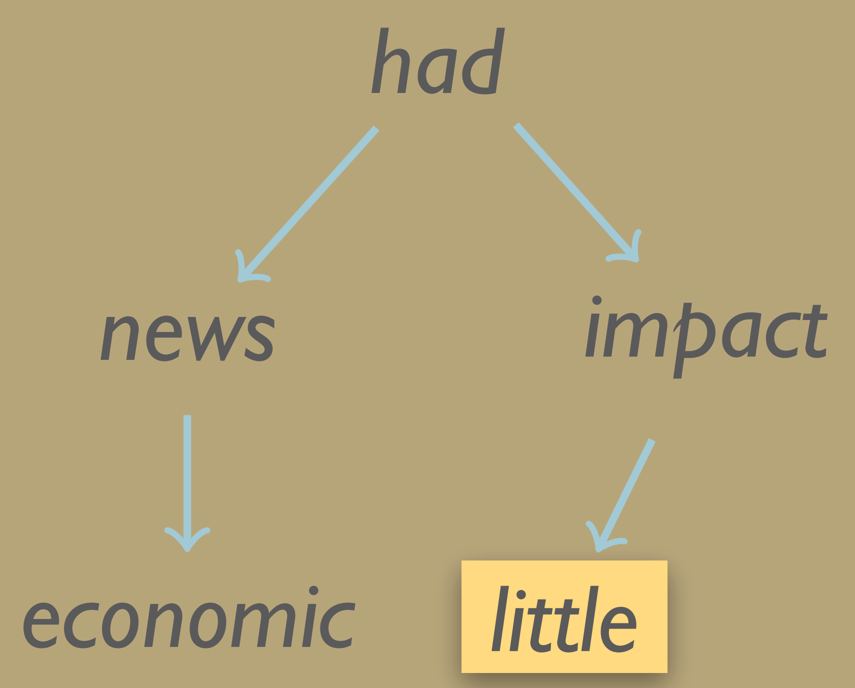


Conversion: PS → DS

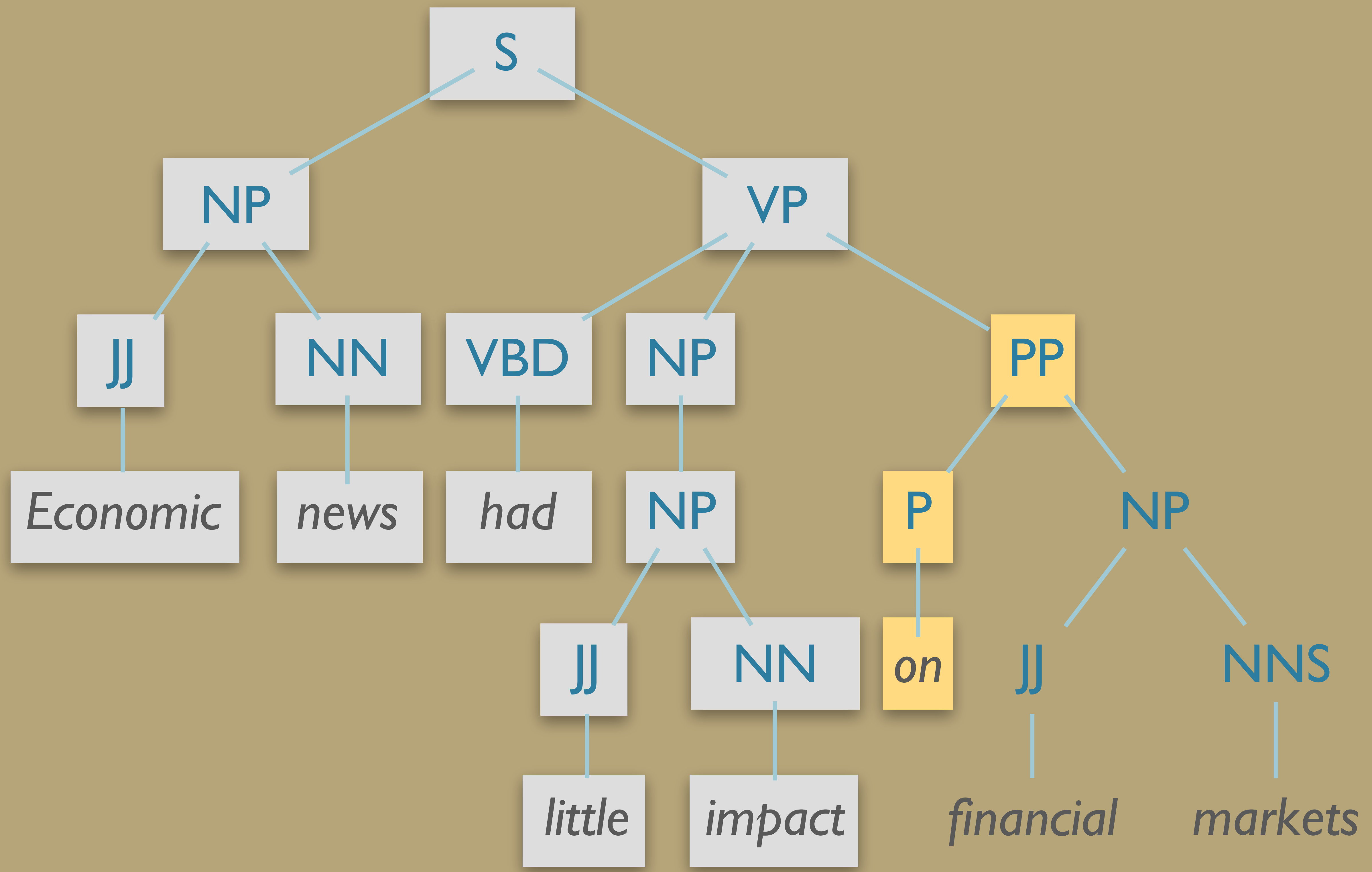


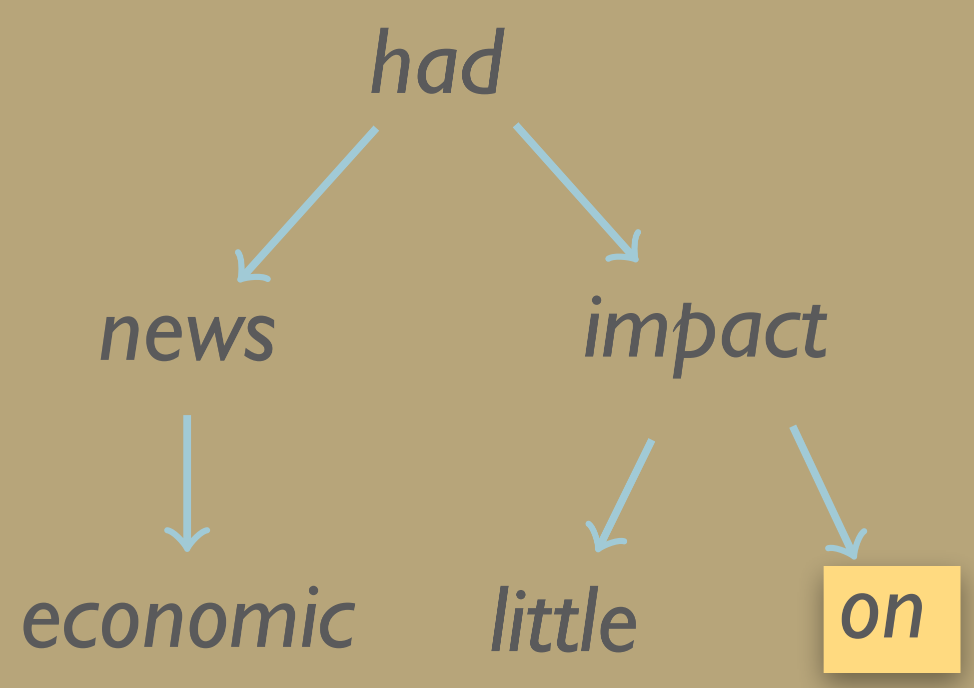
Conversion: PS → DS



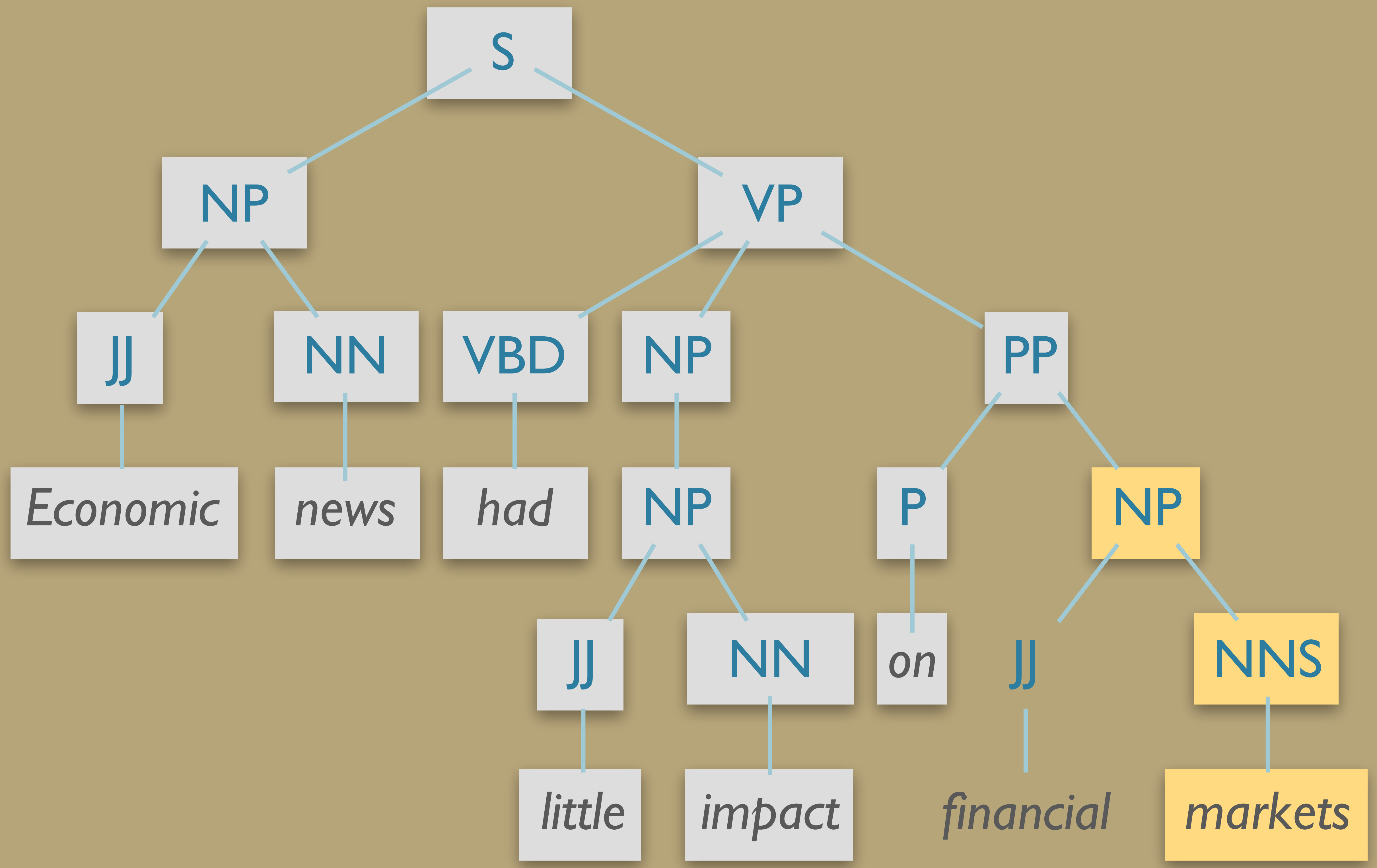


Conversion: PS → DS

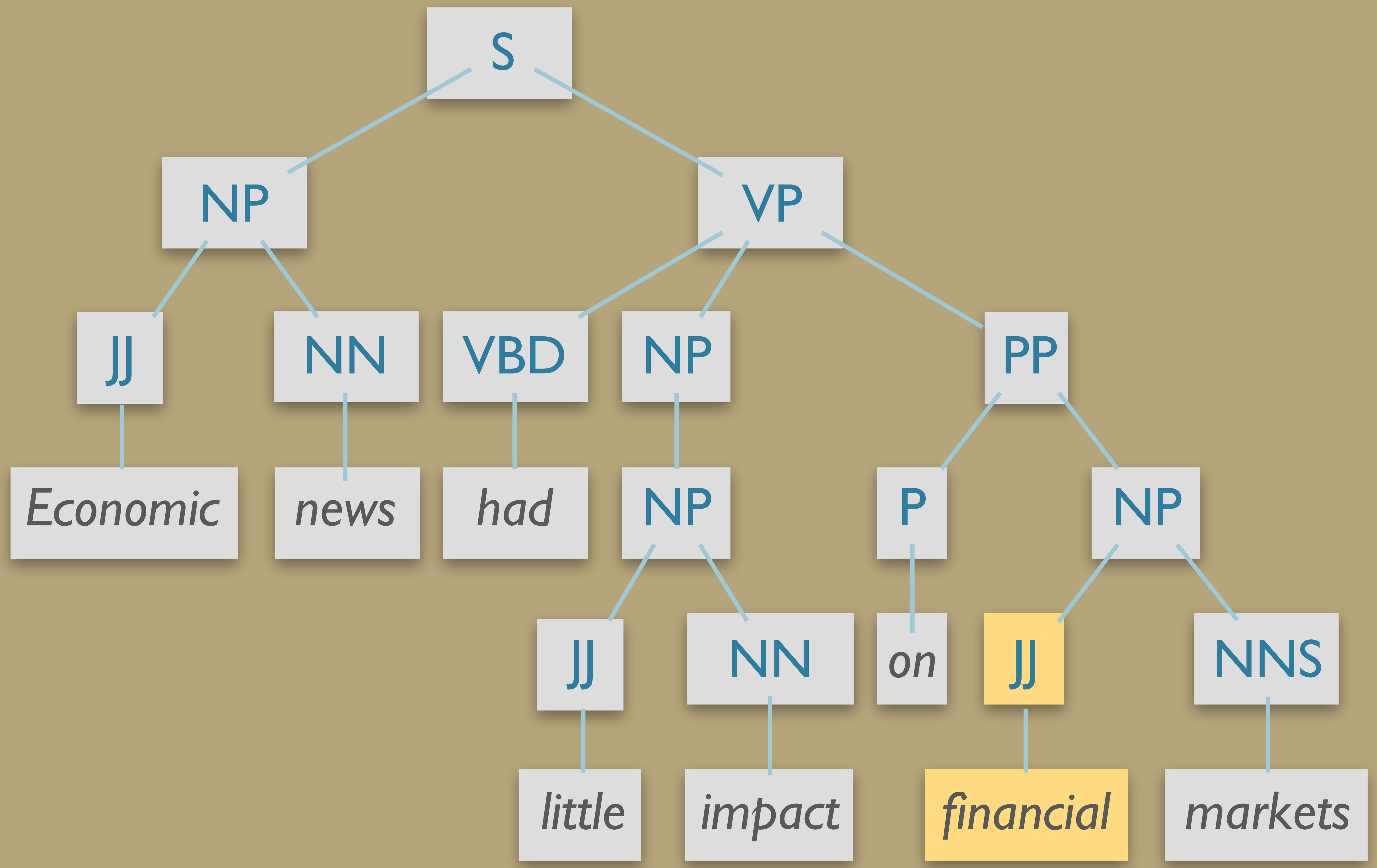




Conversion: PS → DS



Conversion: PS → DS



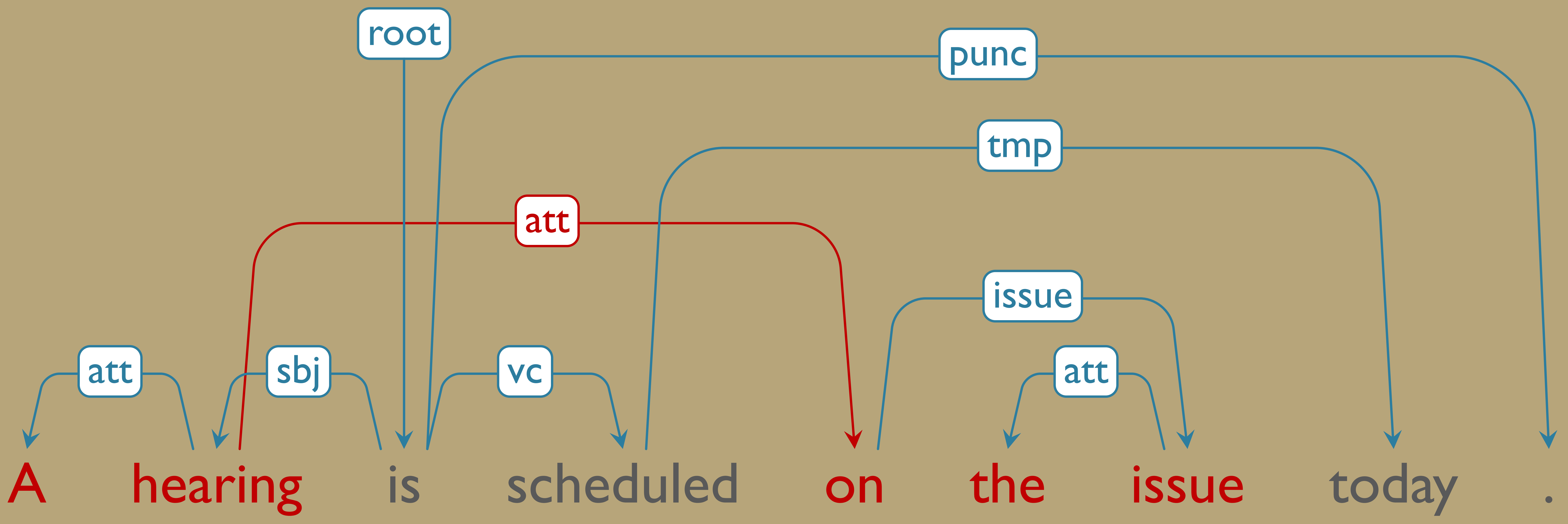
Head Percolation Table

- 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.

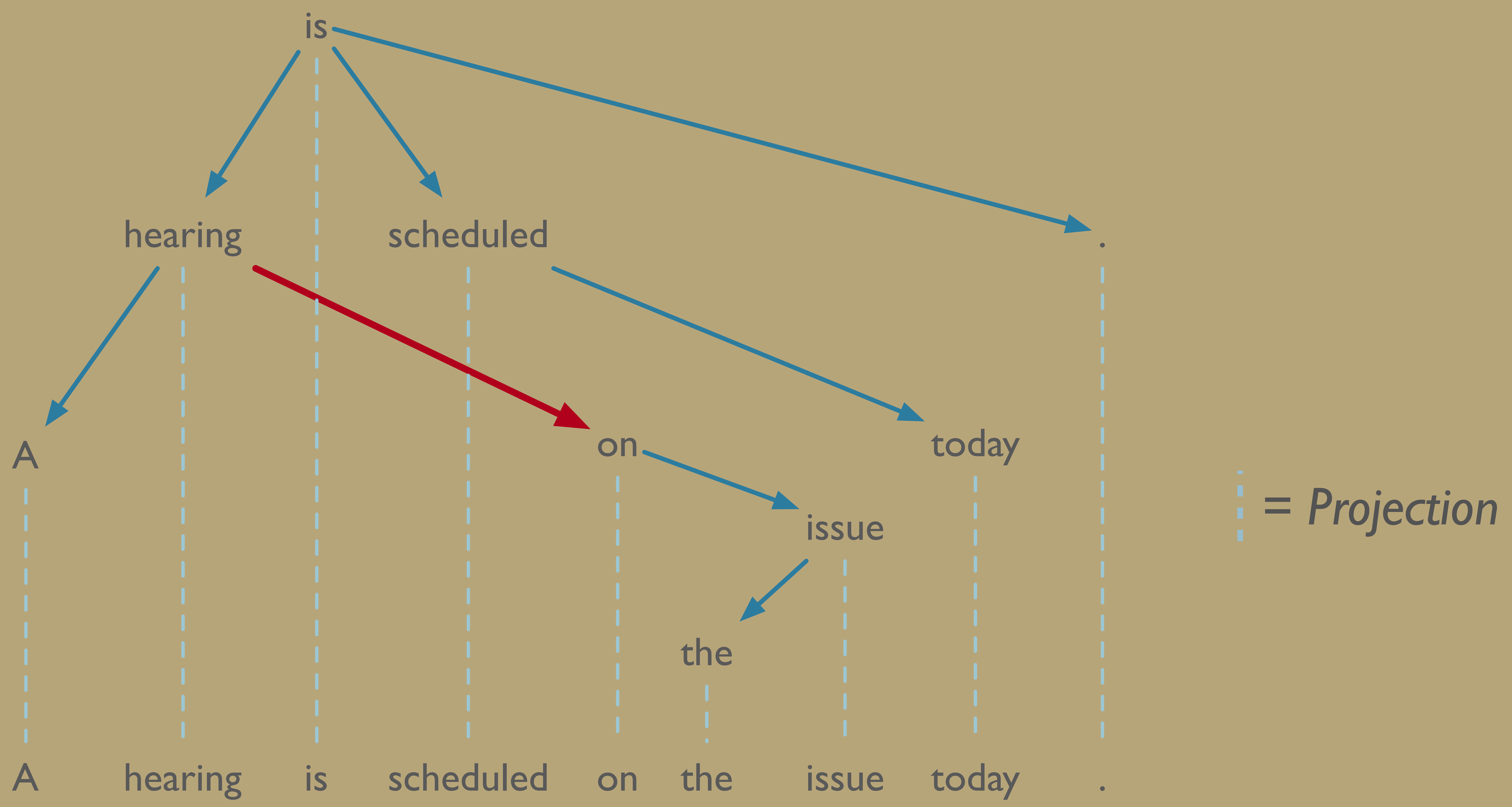
From J&M Page 411, via [Collins (1999)]

Conversion: DS → PS

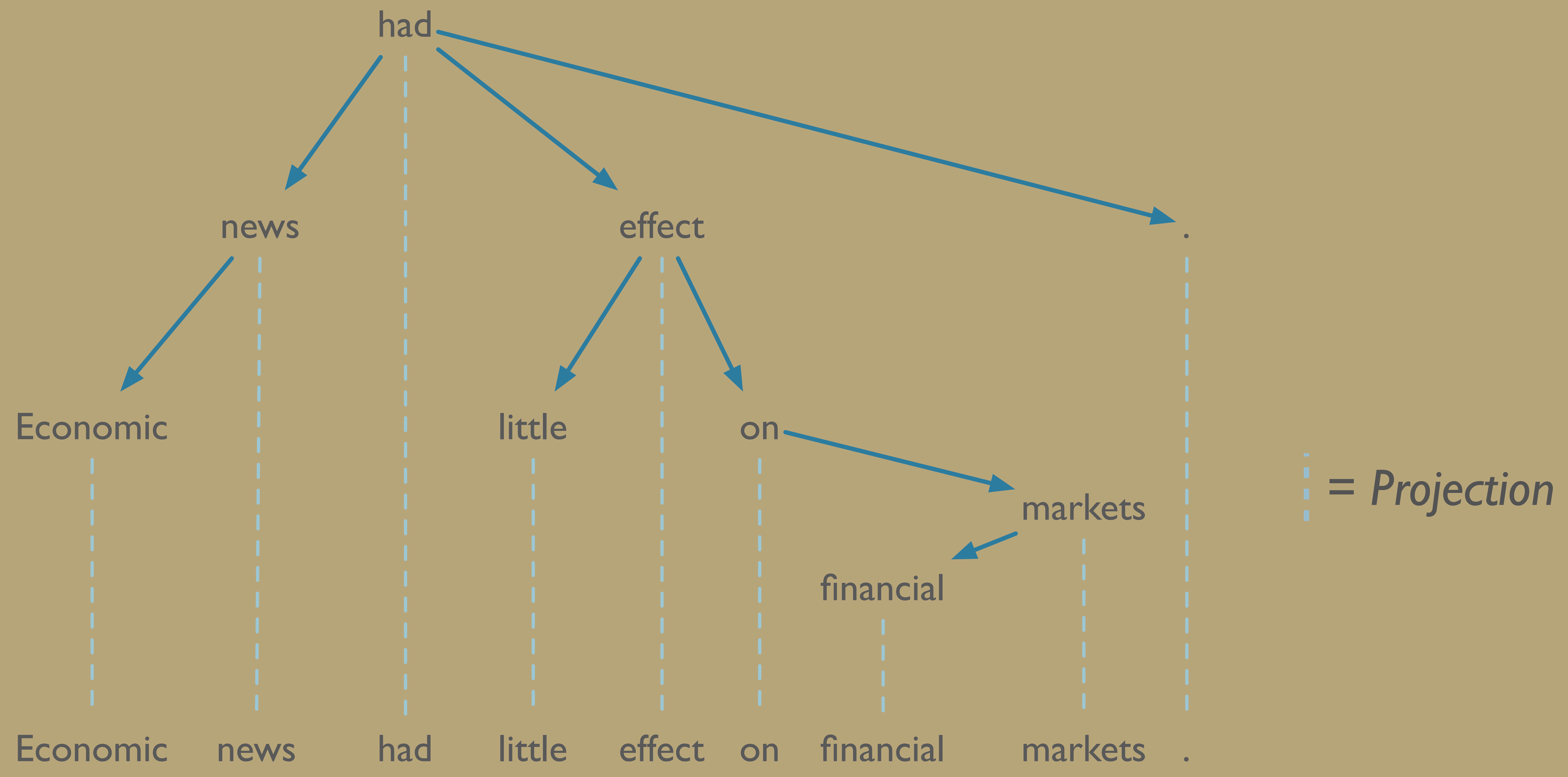
- Can map any *projective* dependency tree to PS tree
- Projective:
  - Does not contain “crossing” dependencies w.r.t. word order



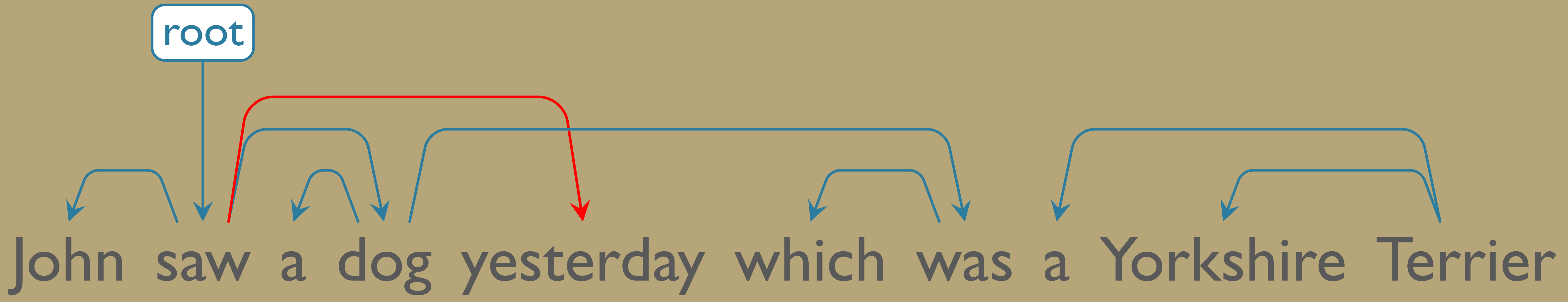
Non-Projective DS

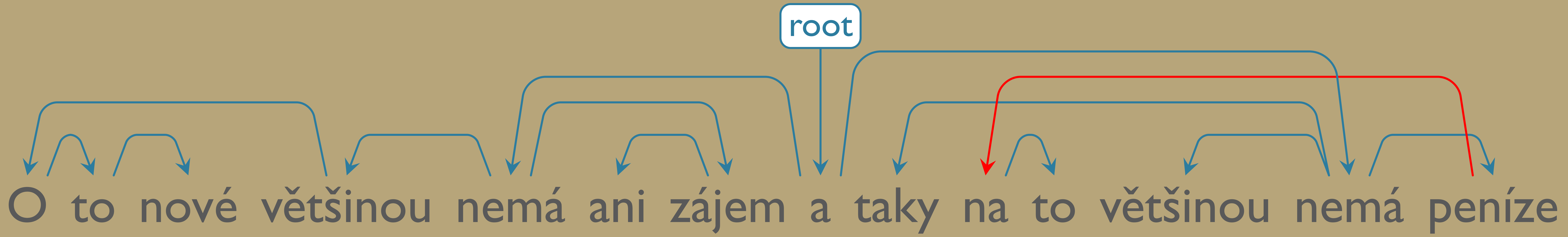


Projective DS



More Non-Projective Parses





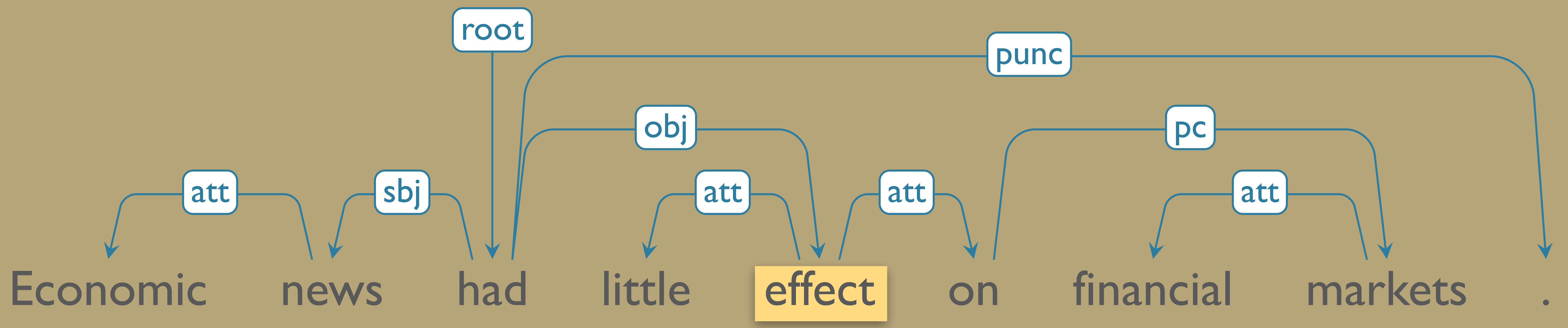
He is mostly not even interested in the new things and in most cases, he has no money for it either.

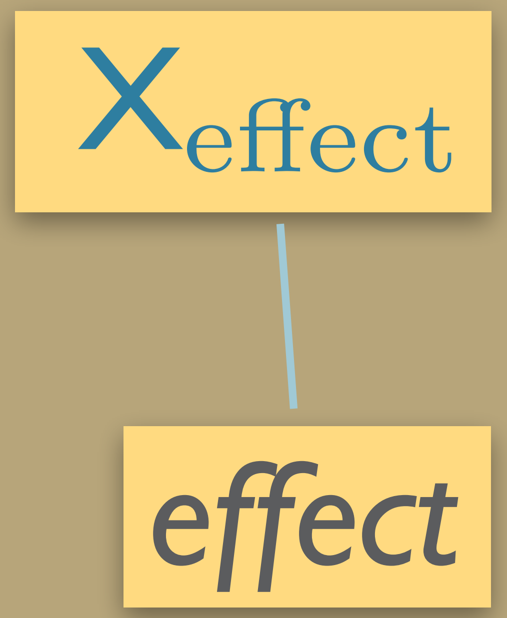
From [McDonald et. al, 2005]

Conversion: DS → PS

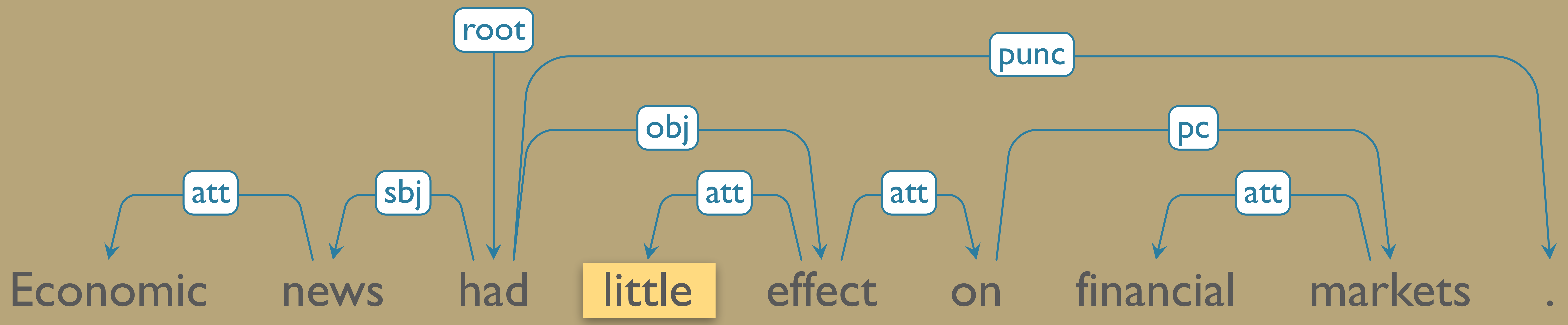
- For each node $\boldsymbol{w}$ with outgoing arcs…
  - …convert the subtree $\boldsymbol{w}$ and its dependents $\boldsymbol{t_1,...,t_n}$ to a new subtree:
    - Nonterminal: X_w
    - Child: w
    - Subtrees $\boldsymbol{t_1,...,t_n}$ in original sentence order

Conversion: DS → PS

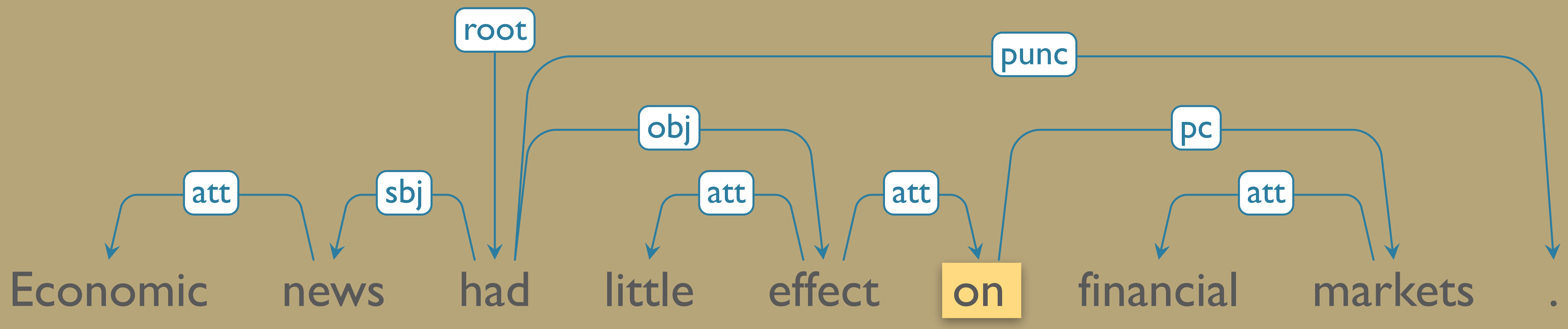


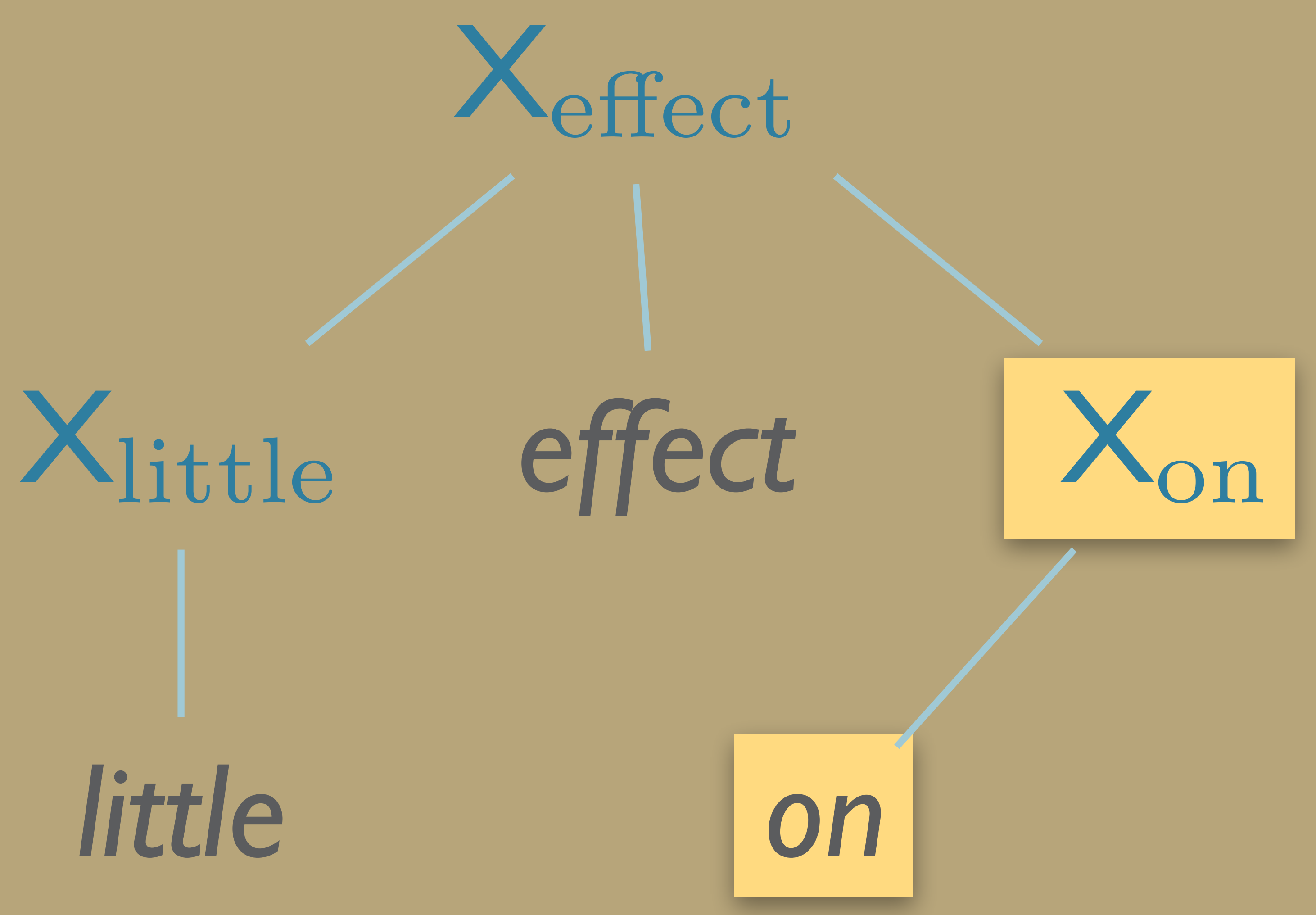


Conversion: DS → PS

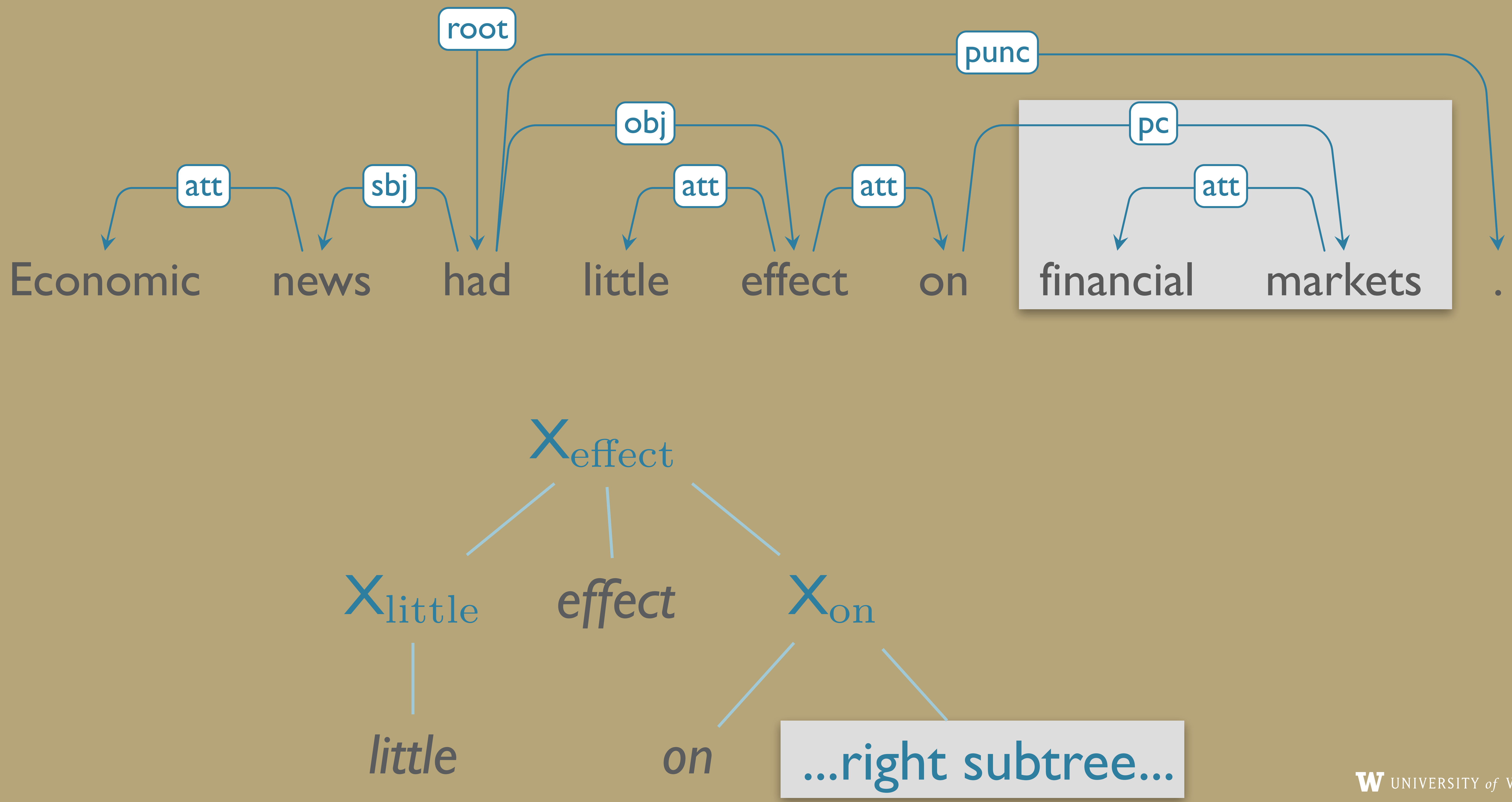


Conversion: DS → PS



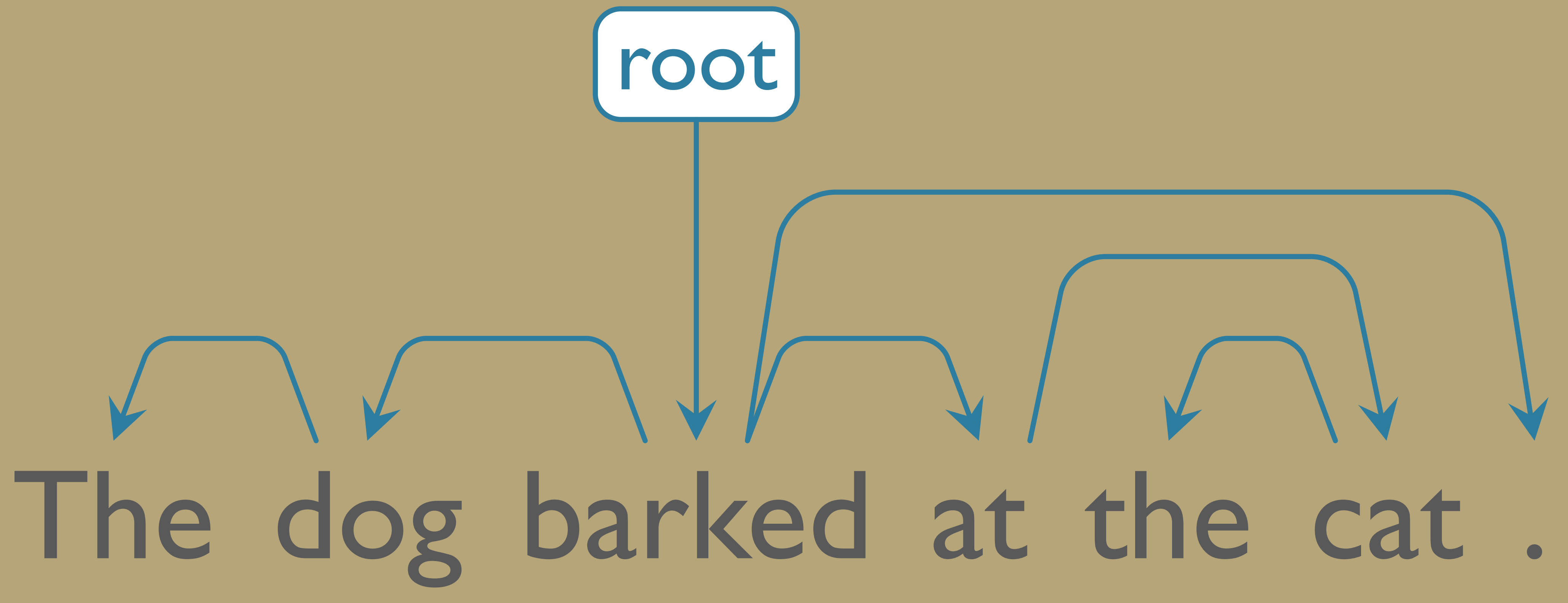


Conversion: DS → PS



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

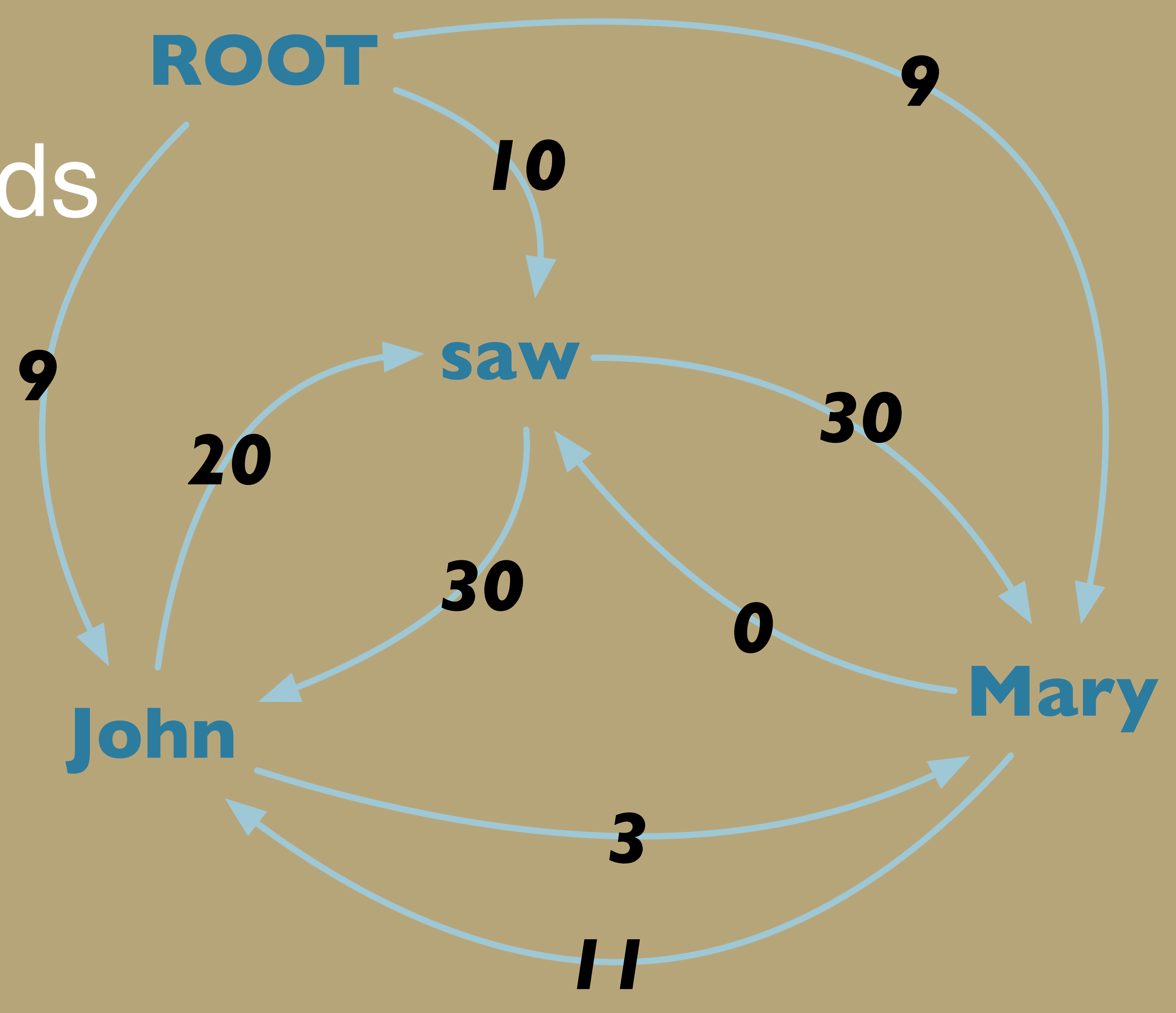
- Map dependency parsing to Maximum Spanning Tree (MST)
- Build fully connected initial graph:
  - Nodes: words in sentence to parse
  - Edges: directed edges between all words
    - + Edges from ROOT to all words
- Identify maximum spanning tree
  - Tree s.t. all nodes are connected
  - Select such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

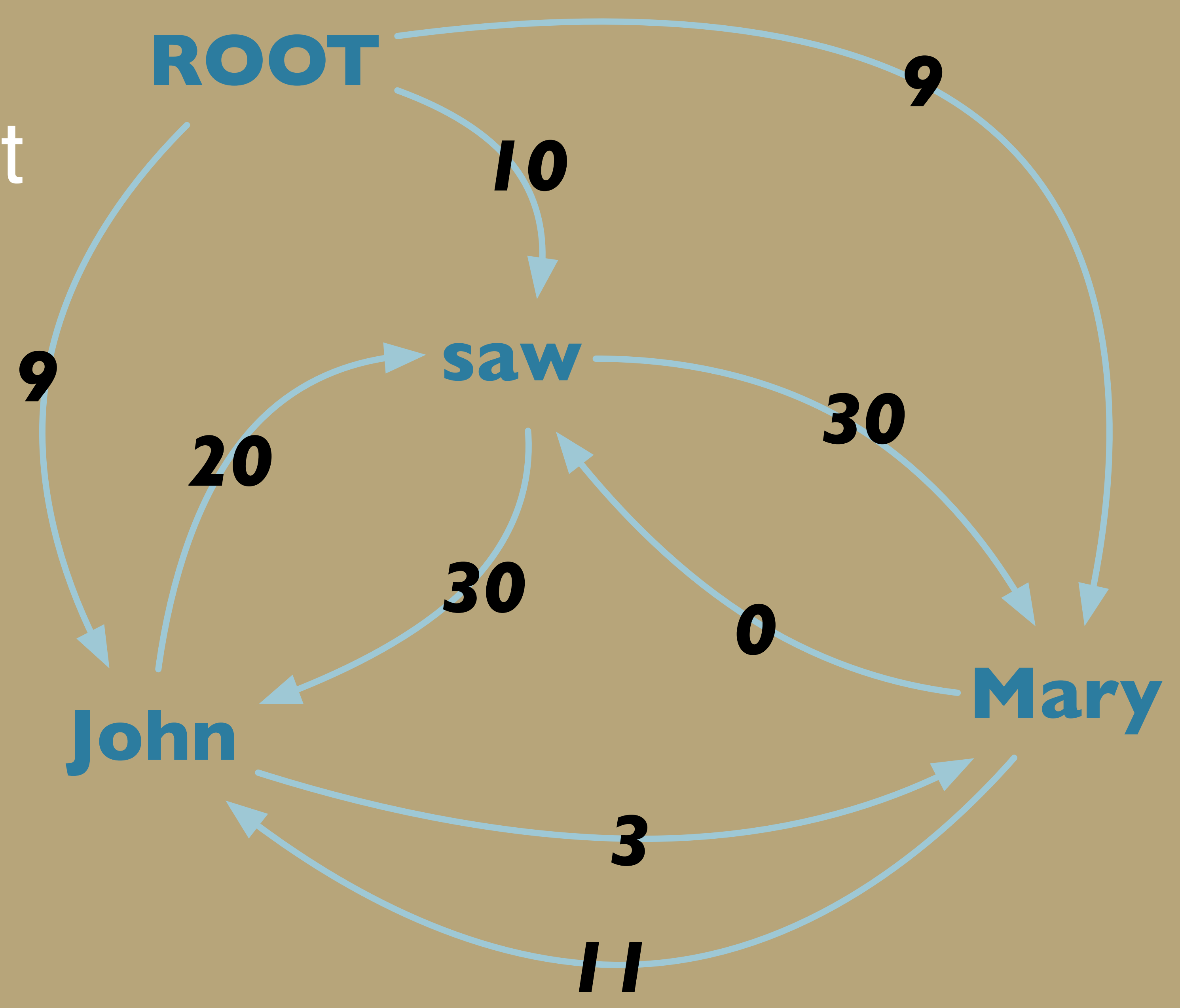
Initial Graph: [(McDonald et al, 2005b)]

- *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



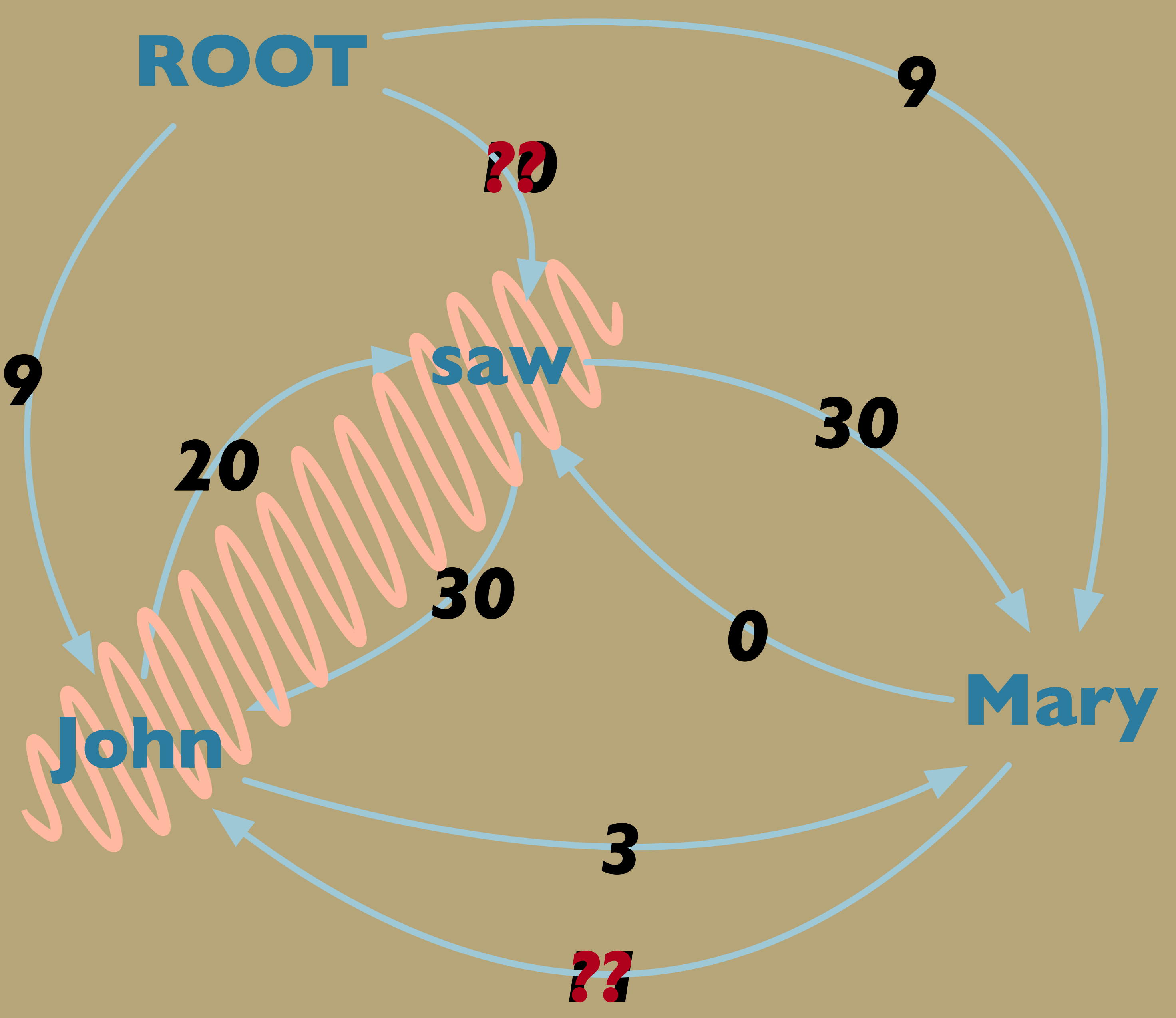
Maximum Spanning Tree

- 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: $\mathrm{O}(n^3)$; Tarjan: $\mathrm{O}(n^2)$
  - Applicable to non-projective graphs



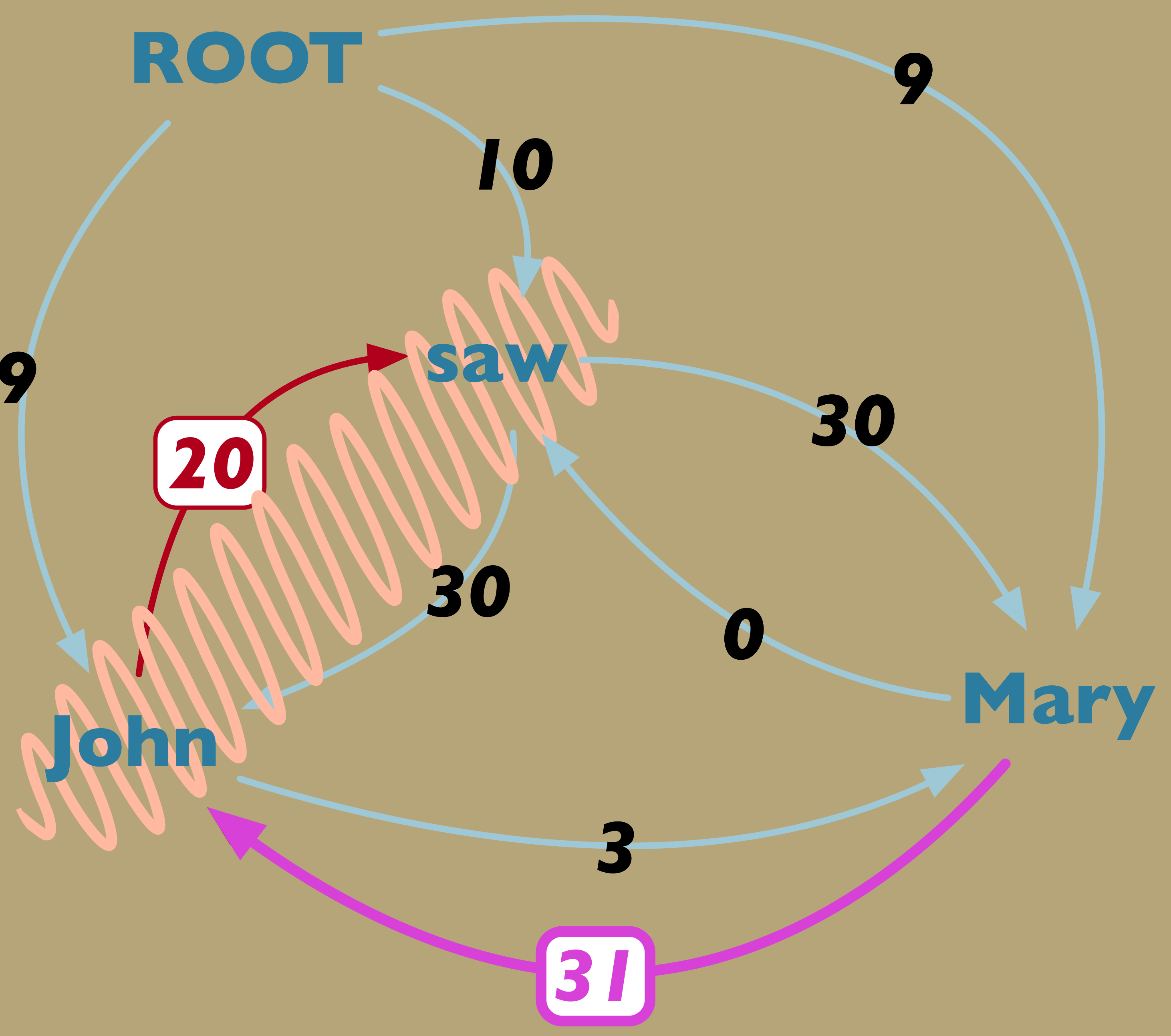
Step 1 & 2

- 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



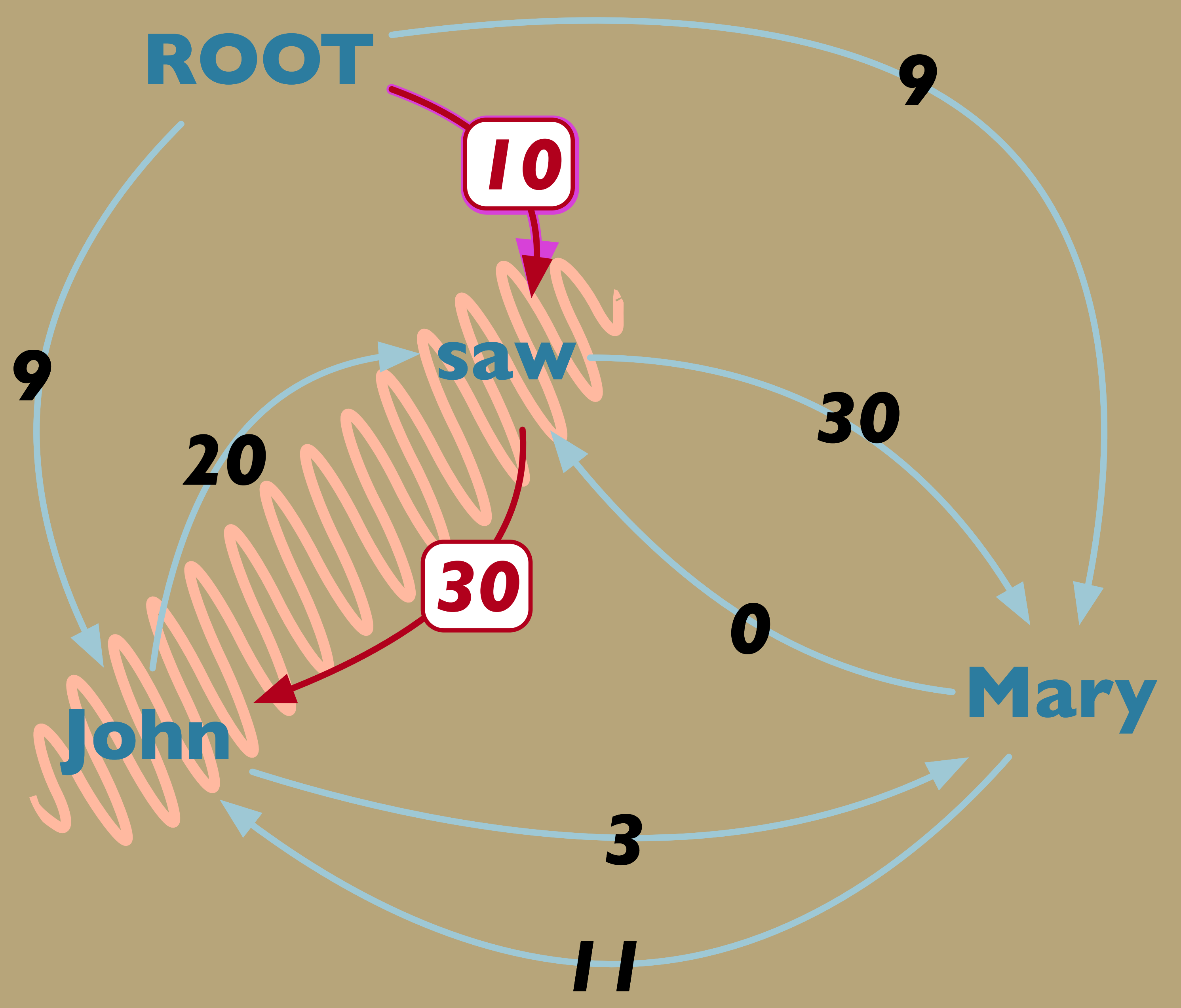
Calculating Weights for Collapsed Vertex

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



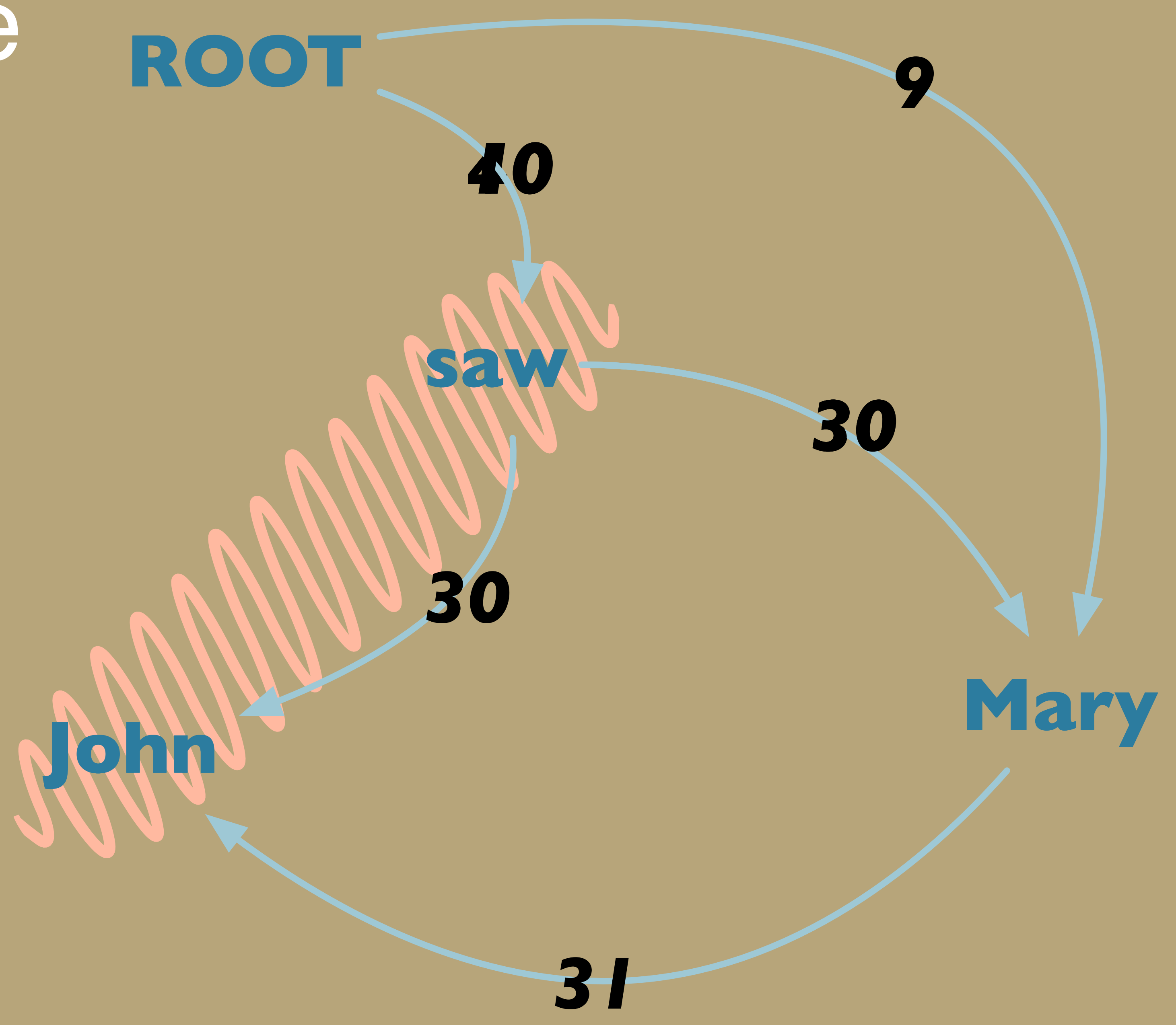
Calculating Weights for Collapsed Vertex

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



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*

    $F \leftarrow []$
    $T' \leftarrow []$
    $score' \leftarrow []$
    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]

HW #4

Probabilistic Parsing

- Goals:
  - Learn about PCFGs
  - Implement PCKY
  - Analyze Parsing Evaluation
  - Assess improvements to PCFG Parsing

Tasks

1. Train a PCFG
   1. Estimate rule probabilities from treebank
   2. Treebank is already in CNF
   3. More ATIS data from Penn Treebank
2. Build CKY Parser
   1. Modify (your) existing CKY implementation

Tasks

3. Evaluation
   1. Evaluate your parser using standard metric
   2. We will provide **evalb** program and gold standard
4. Improvement
   1. Improve your parser in some way:
      1. Coverage
      2. Accuracy
      3. Speed
   2. Evaluate new parser

Improvement Possibilities

- Coverage:
  - Some test sentences won't parse as is!
    - Lexical gaps (aka out-of-vocabulary [OOV] tokens)
      - …remember to model the probabilities, too
- Better context modeling
  - e.g. — Parent Annotation
- Better Efficiency
  - e.g. — Heuristic Filtering, Beam Search
- No "cheating" improvements:
  - improvement can't change training by looking at test data

`evalb`

- `evalb` available in `dropbox/19-20/571/hw4/tools`
- `evalb [...] <gold-file> <test-file>`
- `evalb --help` for more info

Mid-term Evaluation!

- Please take a few minutes to provide feedback on this course
  - Completely anonymous
  - All feedback valuable; will incorporate things that can be changed
  - Final week: summary, but also some current/future directions topics TBD

http://bit.ly/571-aut19-feedback