Dependency Parsing

Computational Linguistics

Alexander Koller

21 December 2023

Final projects

- Each of you will need to come up with a final project for this course.
  - Topic: some problem in computational linguistics, using methods that were taught in this course.
  - Workload: roughly as much as one assignment.
  - Deadline: March 22

- Please write a half-page project proposal and submit it via Google Form by January 28.

- I will individually discuss your projects with you in early February.

Outline

1. Dependency parsing.

2. Transition-based dependency parsing.

3. Graph-based dependency parsing.

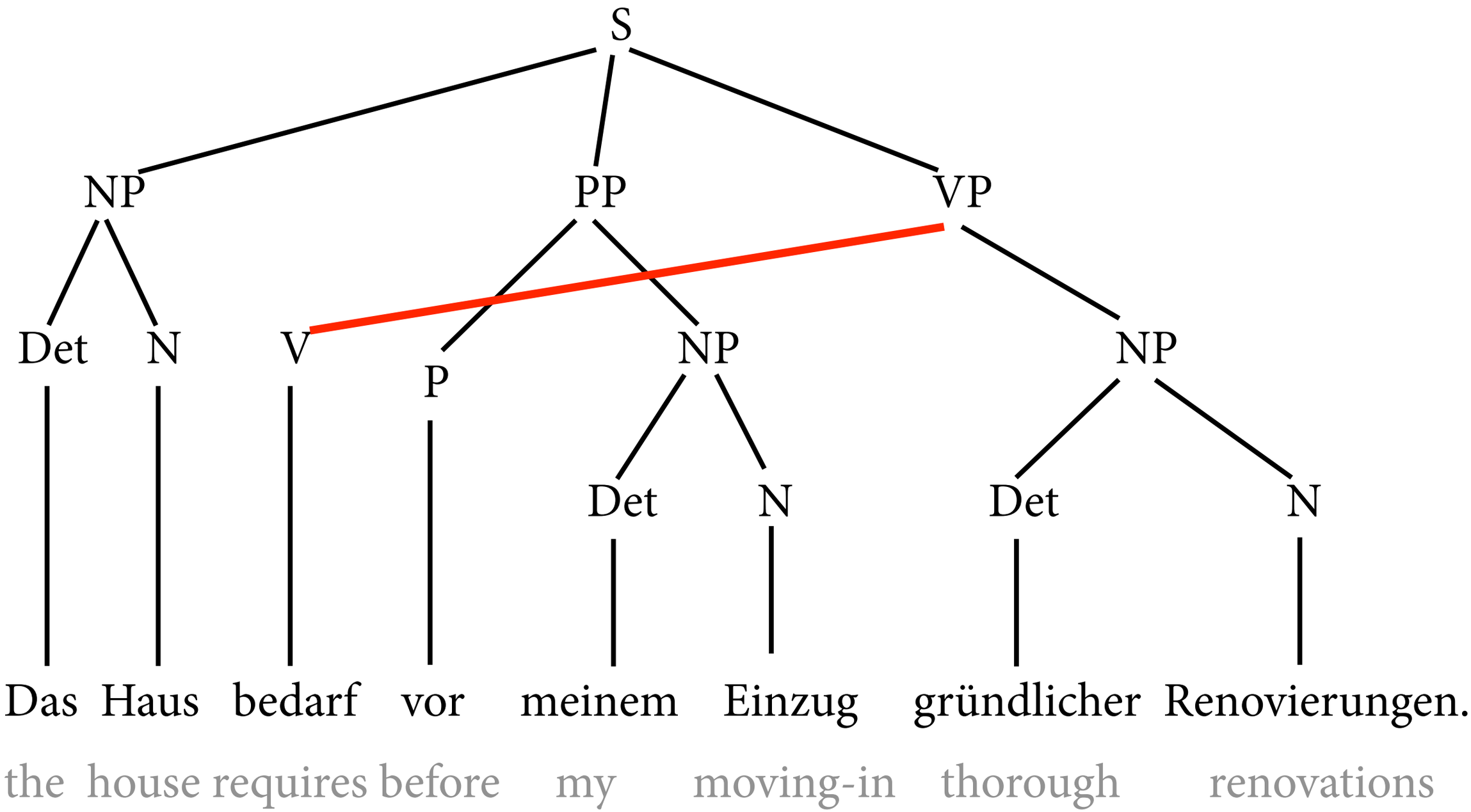
4. Evaluation.

Discontinuous constituents

- So far, we have talked about *phrase-structure* parsing.
  - substrings form constituents of various syntactic categories
  - every constituent must be a contiguous substring

- This assumption mostly correct for English.
  For other languages, it doesn't work so well.

Example

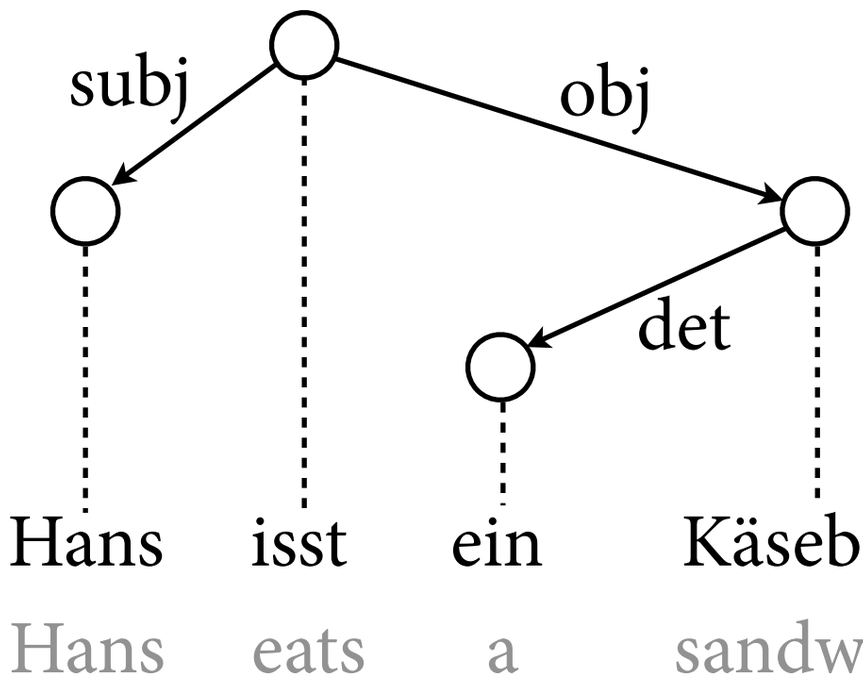

S
NP
PP
VP
Det
N
V
P
NP
Det
N
NP
Det
N
Das Haus bedarf vor meinem Einzug gründlicher Renovierungen.
the house requires before my moving-in thorough renovations

Dependency trees

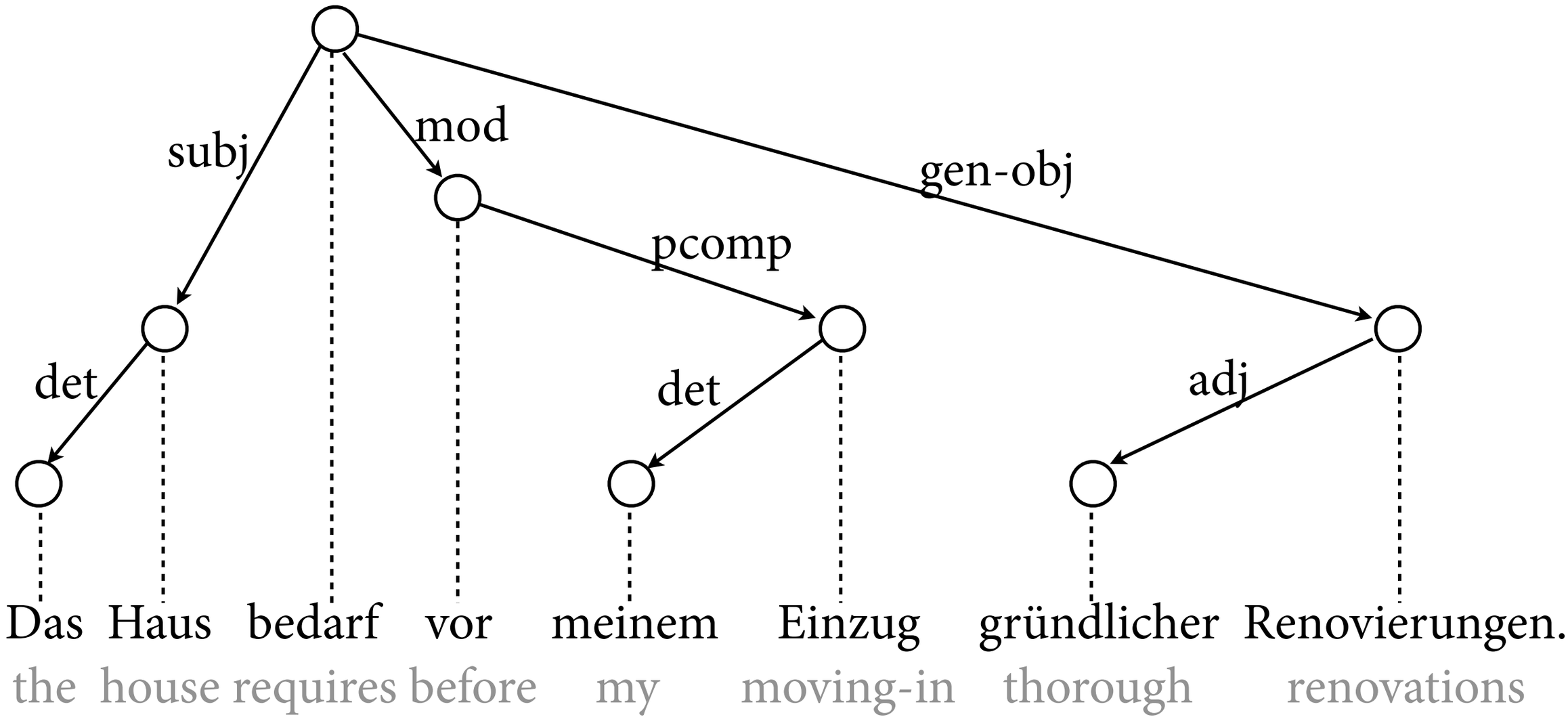
- Basic idea:
  - no constituents, just relations between words
  - nodes of tree = words; edges = relations
  - grammar specifies valency of each word

- Brief history:
  - Tesniere 1953, posthumously
  - Prague School during Cold War
  - very important in CL since 2005 or so (Nivre, McDonald)

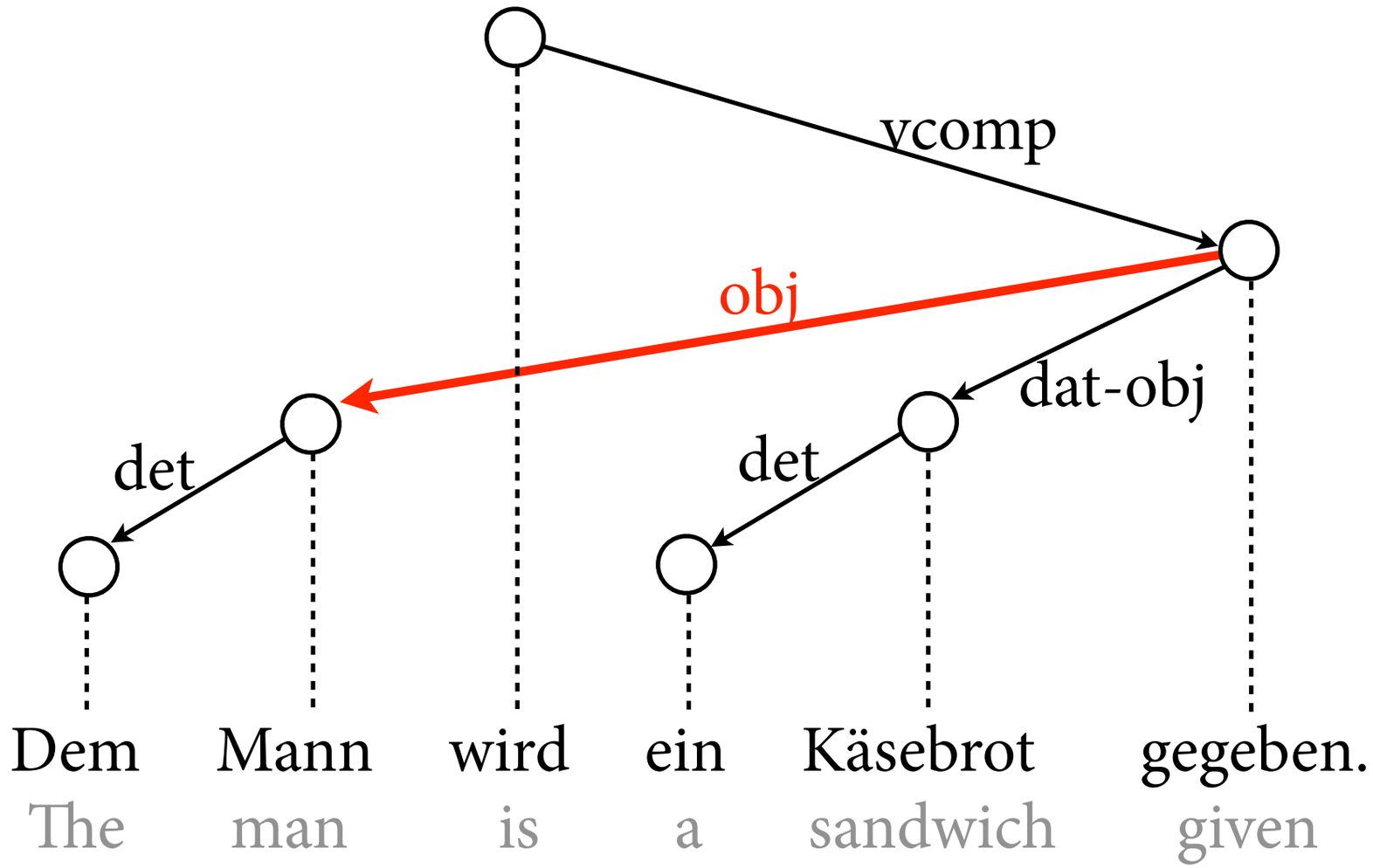
A dependency tree



A dependency tree

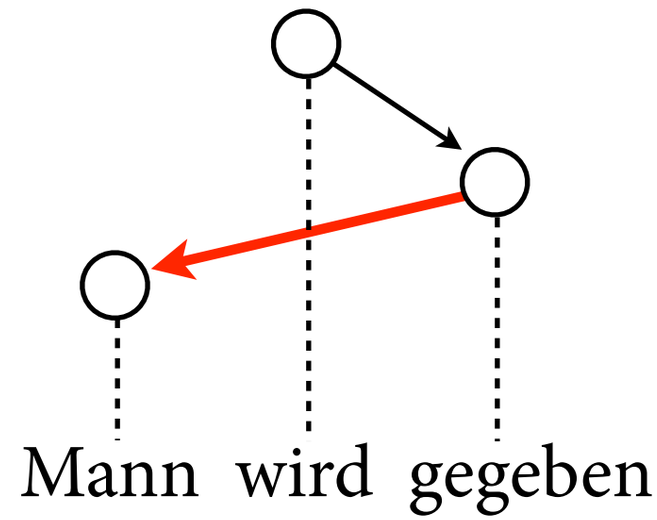


A dependency tree



Projectivity

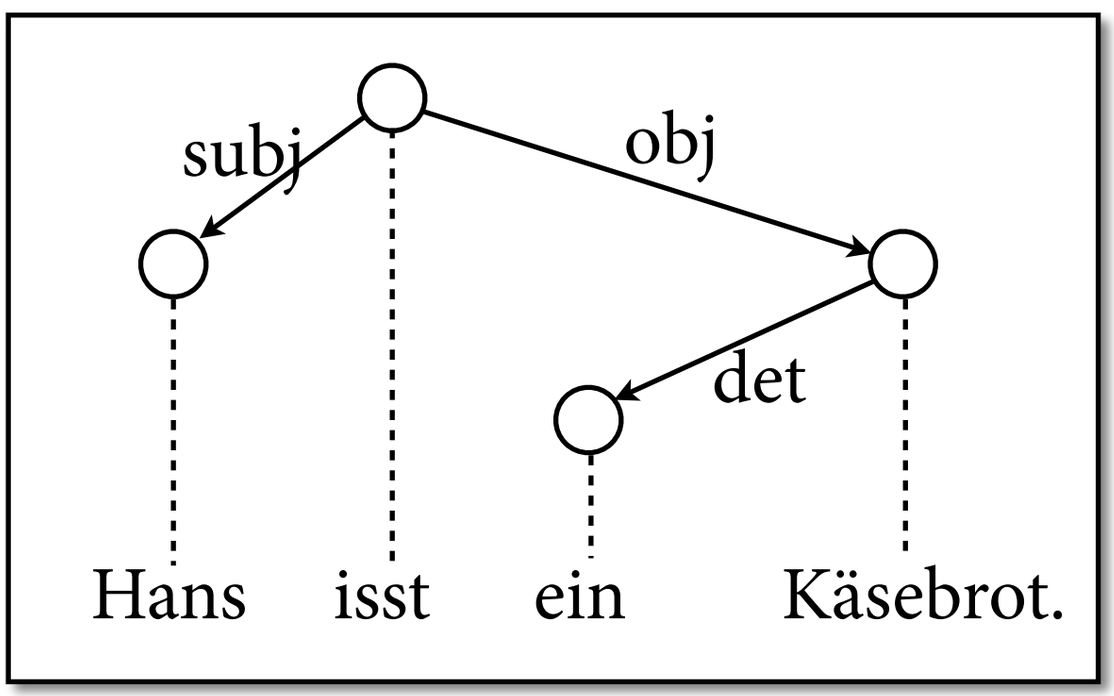
- Dependency tree may have *crossing edges*, which cross the *projection line* of another word.



- A dependency tree is called *projective* iff it has no crossing edges.

Transition-based dependency parsing

- Idea by Joakim Nivre (2003):
  - read sentence word by word, left to right
  - after each word, select a *parser operation* from large set by consulting a machine-learned classifier
  - original algorithm constructs only projective trees; can be extended to non-projective parsing too

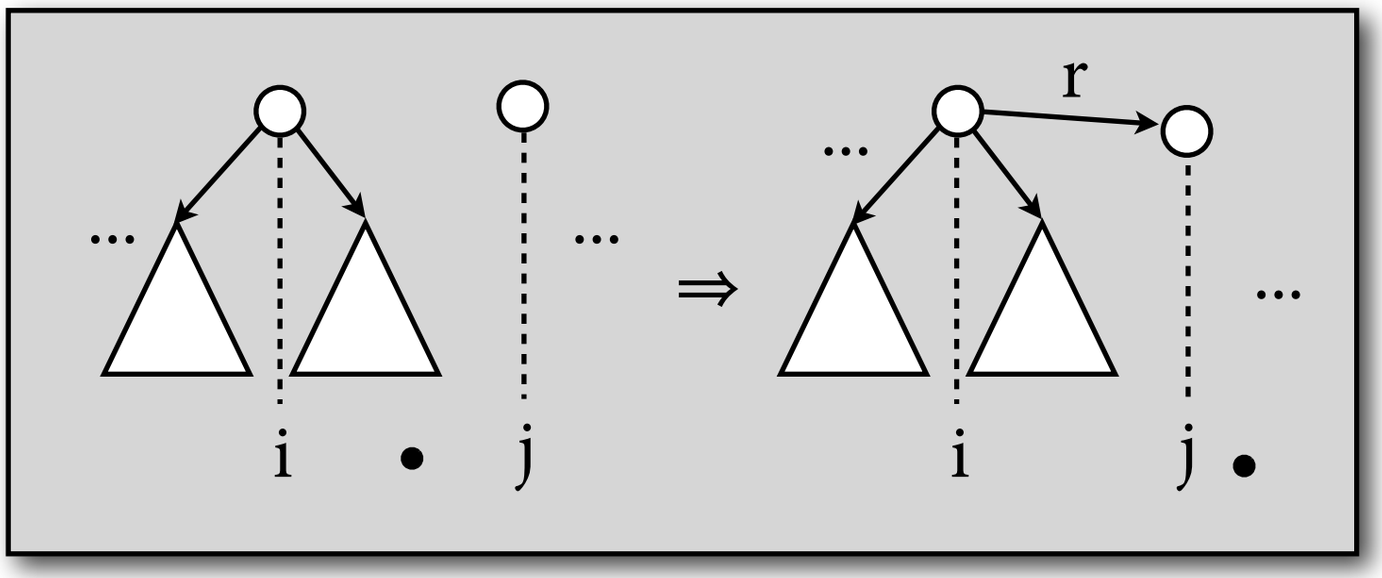


Right-Arc operation

- Right-Arc(r): Input token *j* becomes (right) r-child of topmost stack token *i*.

stack buffer

$$\frac{(\sigma \cdot i,\ \ j \cdot \tau,\ \ h, d) \quad h(j) = 0}{(\sigma \cdot i \cdot j,\ \ \tau,\ \ \ h[j \mapsto i], d[j \mapsto r])}$$

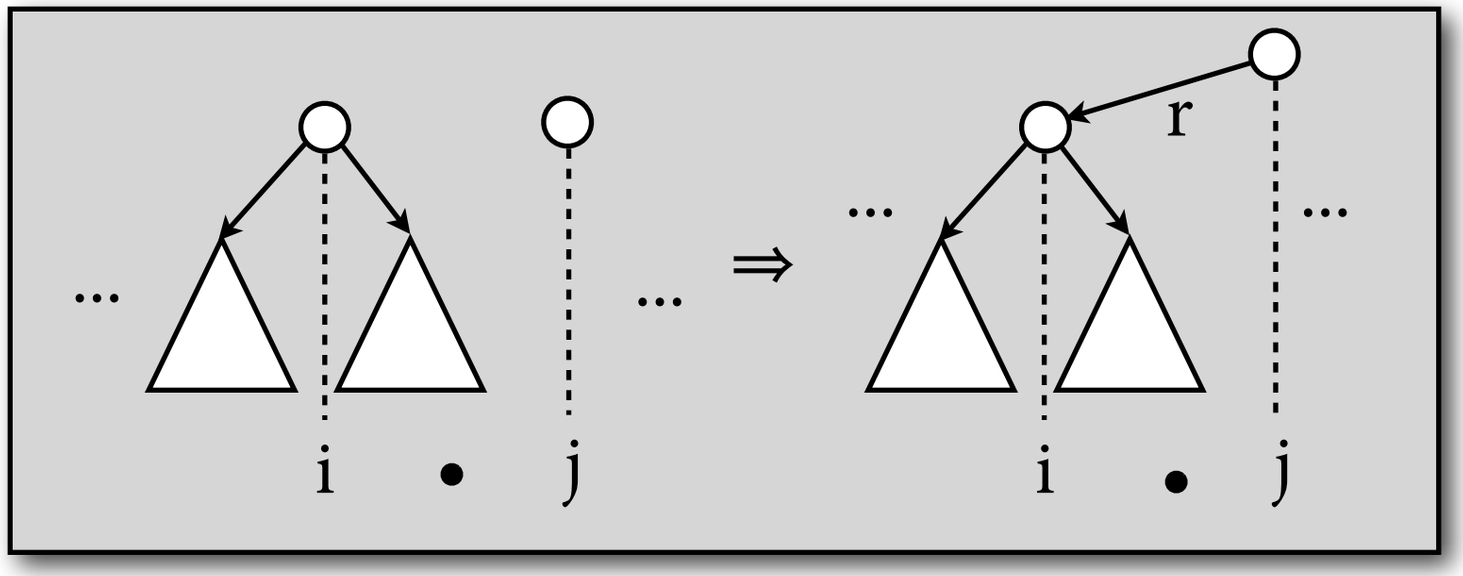


- *i*, *j* both remain on stack because they can receive further children (on the right).

Left-Arc operation

- Left-Arc(r): Topmost token *i* on stack becomes left r-child of next input token *j*.

$$\frac{(\sigma \cdot i,\ j \cdot \tau,\quad h, d)\quad h(i) = 0}{(\sigma,\quad j \cdot \tau,\quad h[i \mapsto j], d[i \mapsto r])}$$

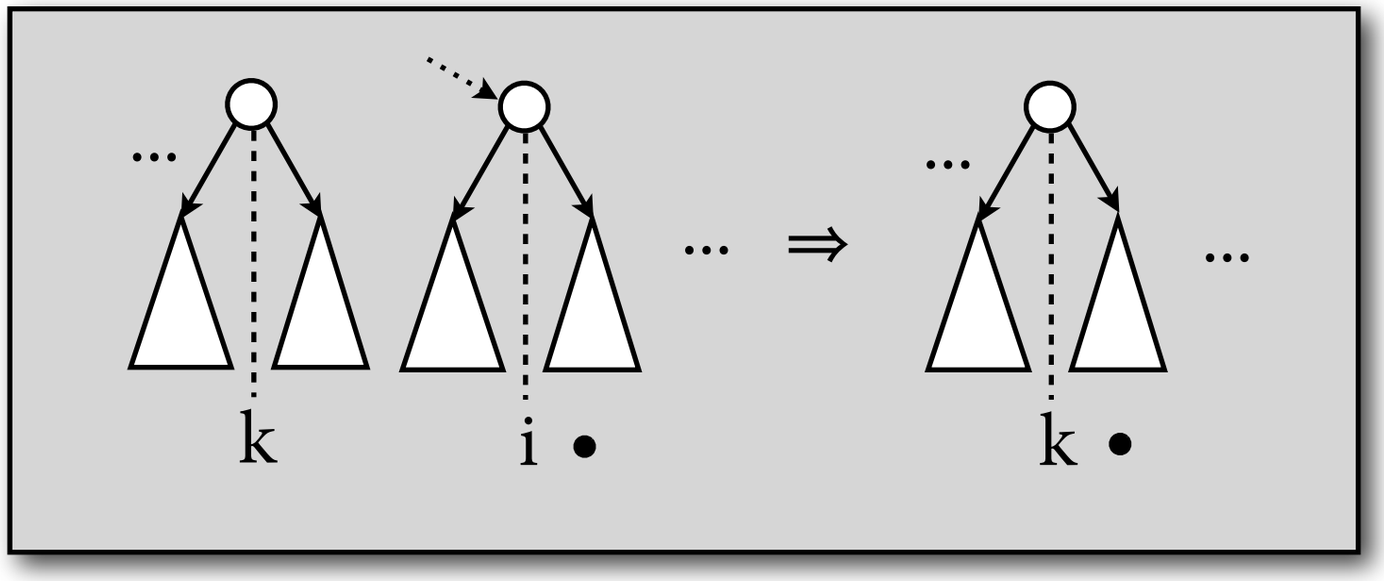


- *i* disappears from stack, because *i* can't get further children in a projective tree

Reduce operation

- Reduce: Remove topmost token from stack.

$$\frac{(\sigma \cdot i,\ \tau,\ h, d) \quad h(i) \neq 0}{(\sigma,\ \tau,\ h, d)}$$



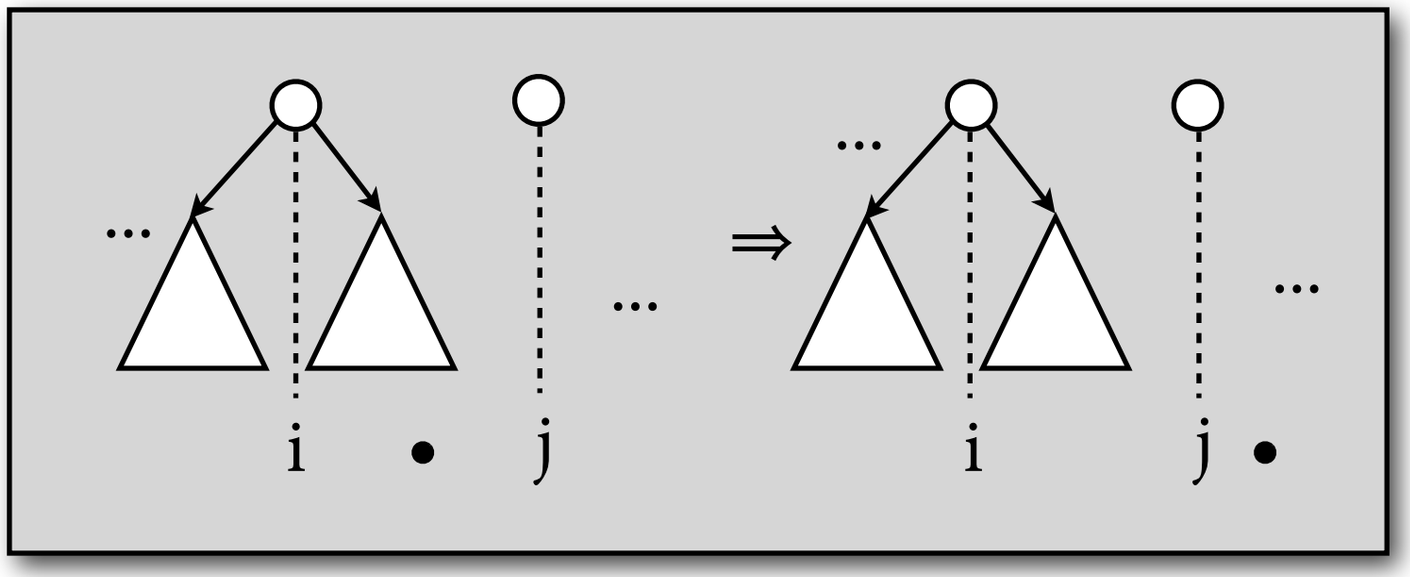
- This decides that we have seen all children of *i*, and makes words further to the left available for receiving further right children.

- Rule requires that *i* already has a parent.

Shift operation

- Shift: Moves next input token *j* to stack.

$$\frac{(\sigma,\ \ j\cdot\tau,\ \ h, d)}{(\sigma\cdot j,\ \tau,\ \ h, d)}$$



- Decides that we have seen all edges between *j* and any word *i* on stack.

Example run

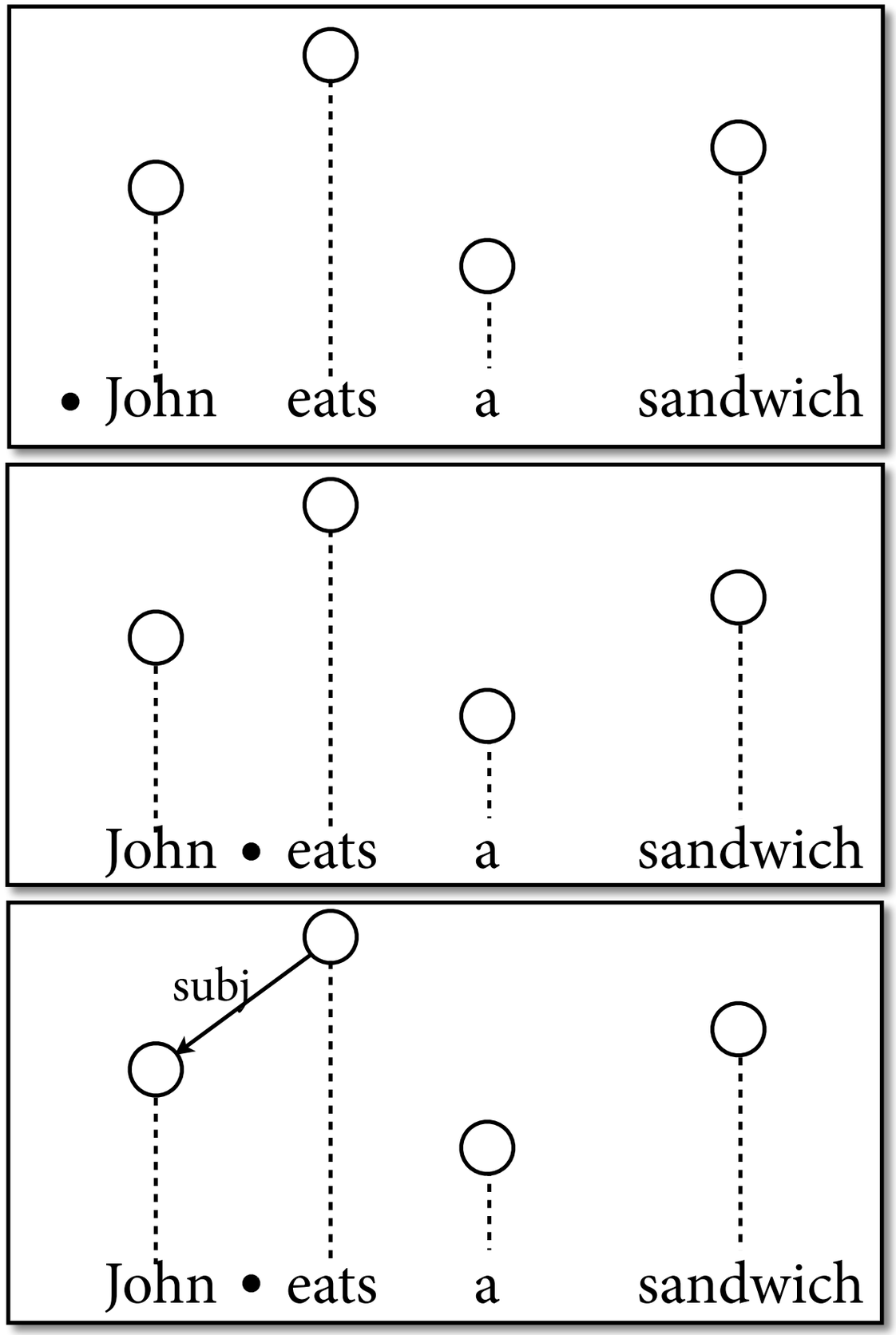
(ε, J eats a sw)

⇓ Shift

(J, eats a sw)

⇓ Left-Arc(subj)

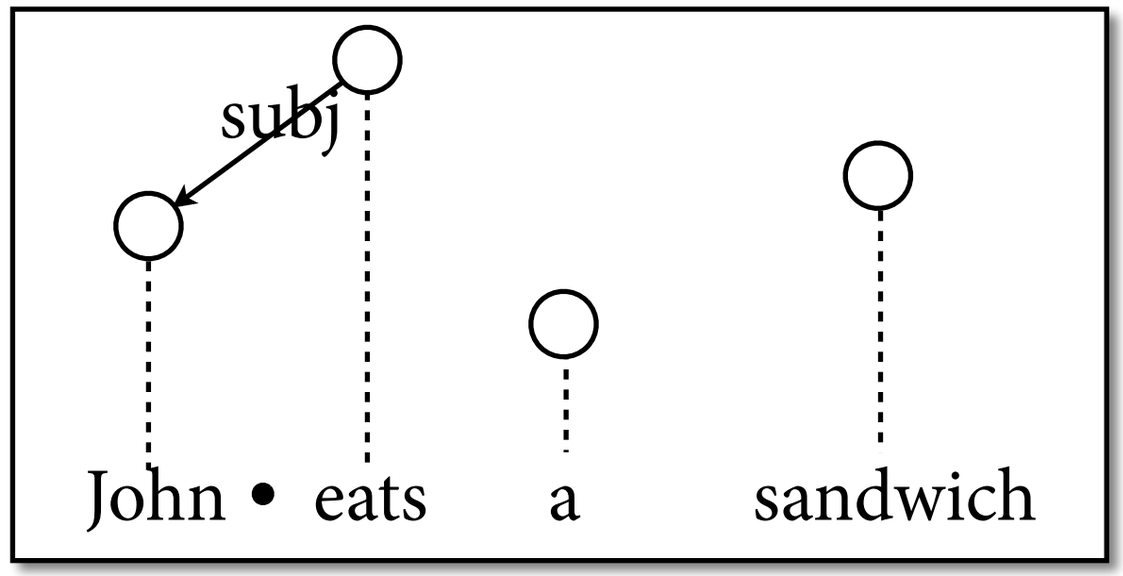
(ε, eats a sw)



Example run

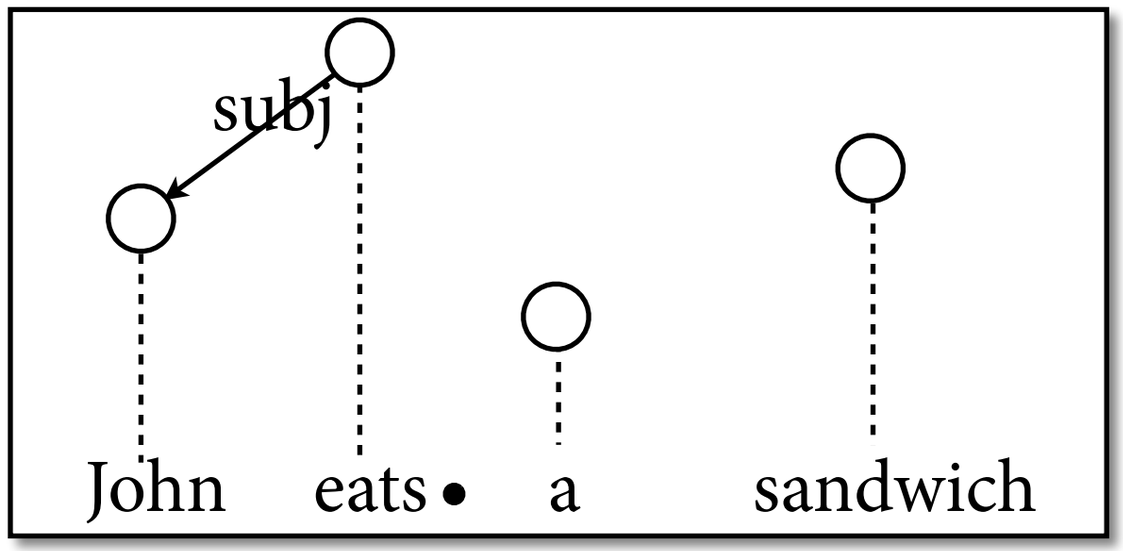
(ε, eats a sw)

⇓ Shift

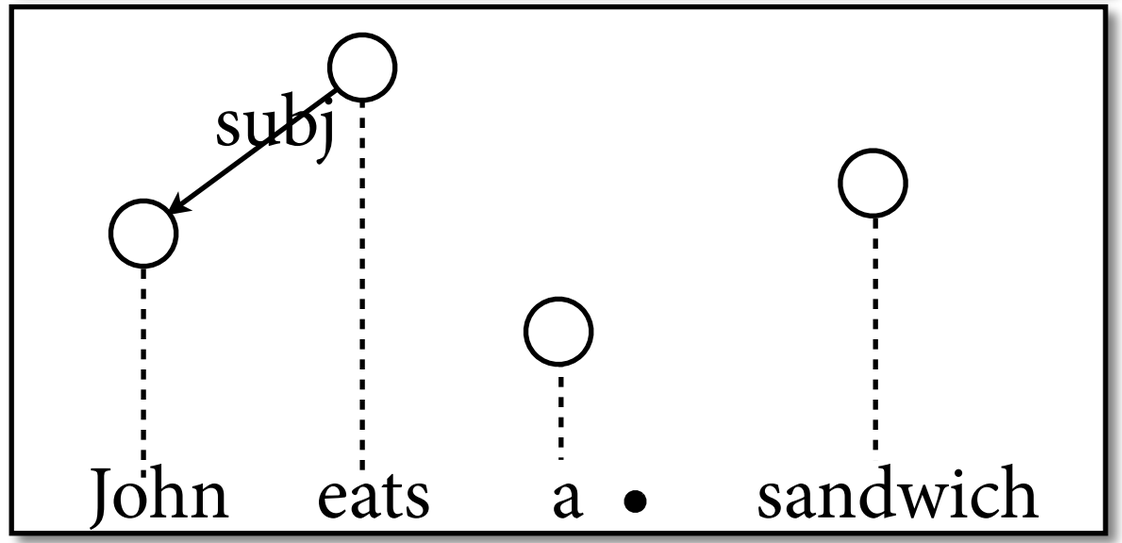


(eats, a sw)

⇓ Shift



(eats a, sw)



Example run

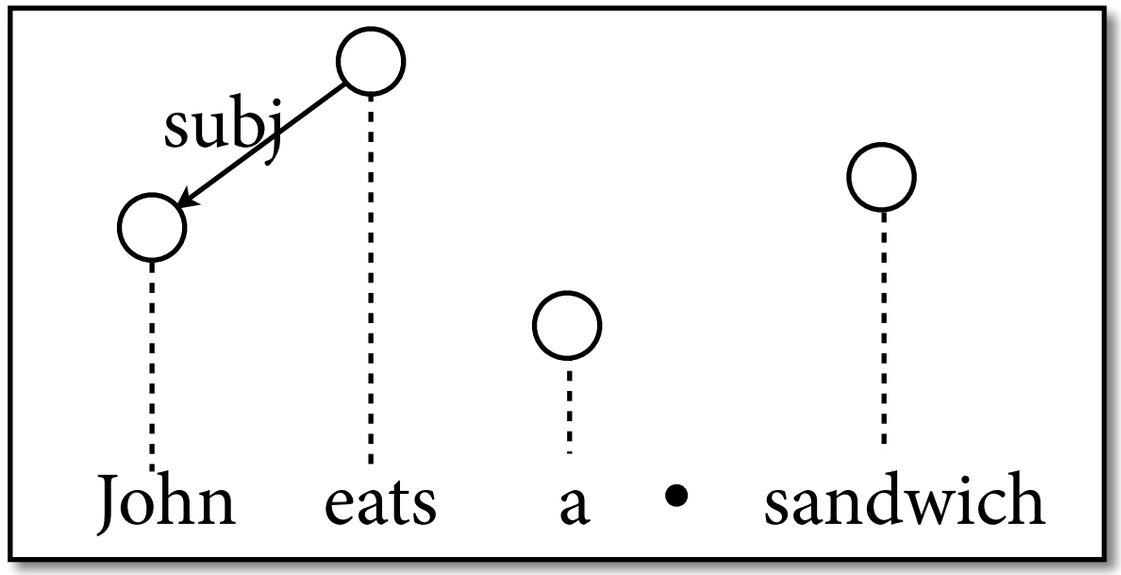
(eats a, sw)

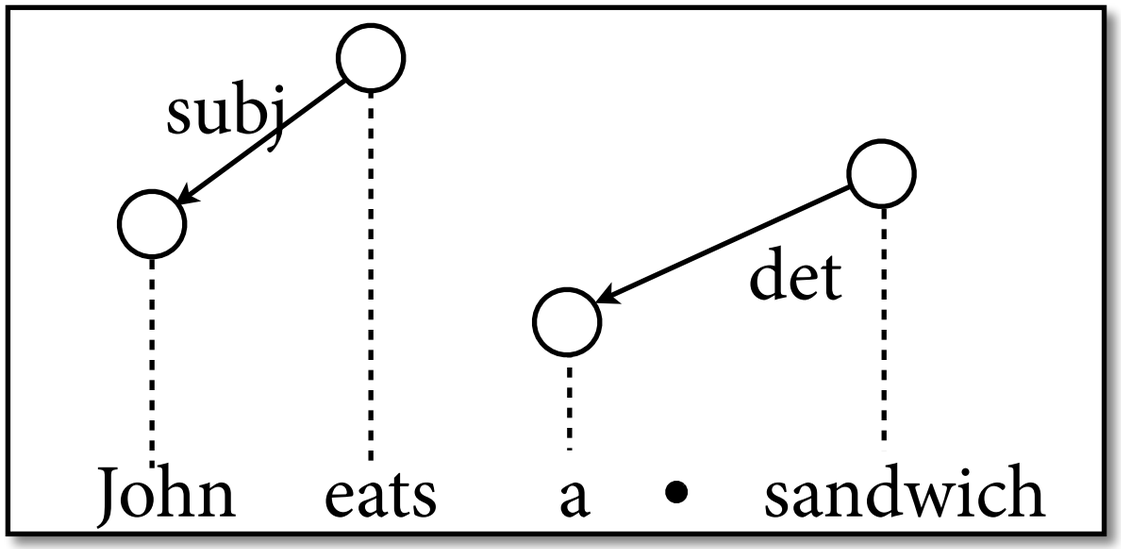
⇓ Left-Arc(det)

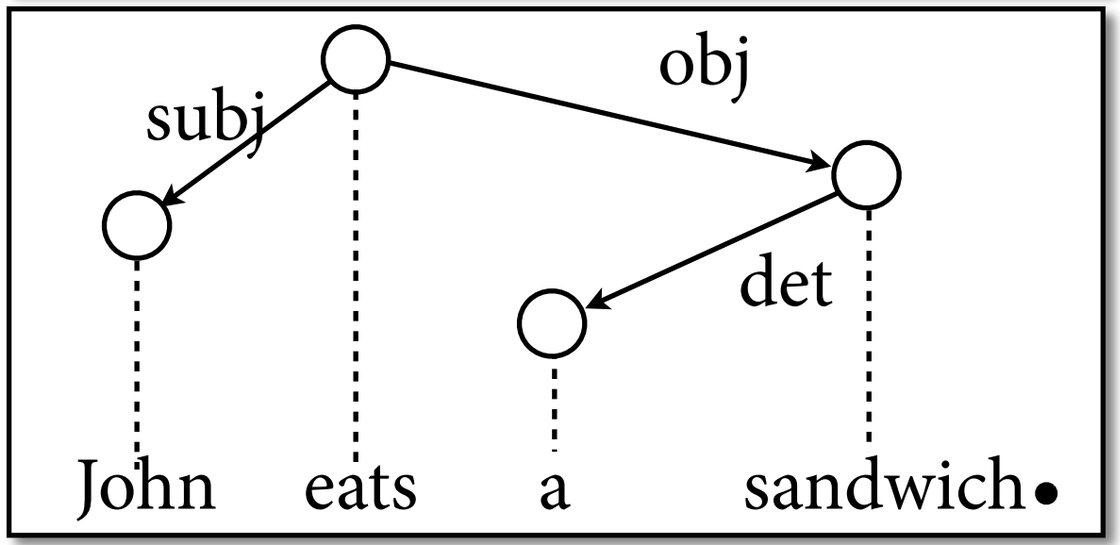
(eats, sw)

⇓ Right-Arc(obj)

(eats sw, ε)







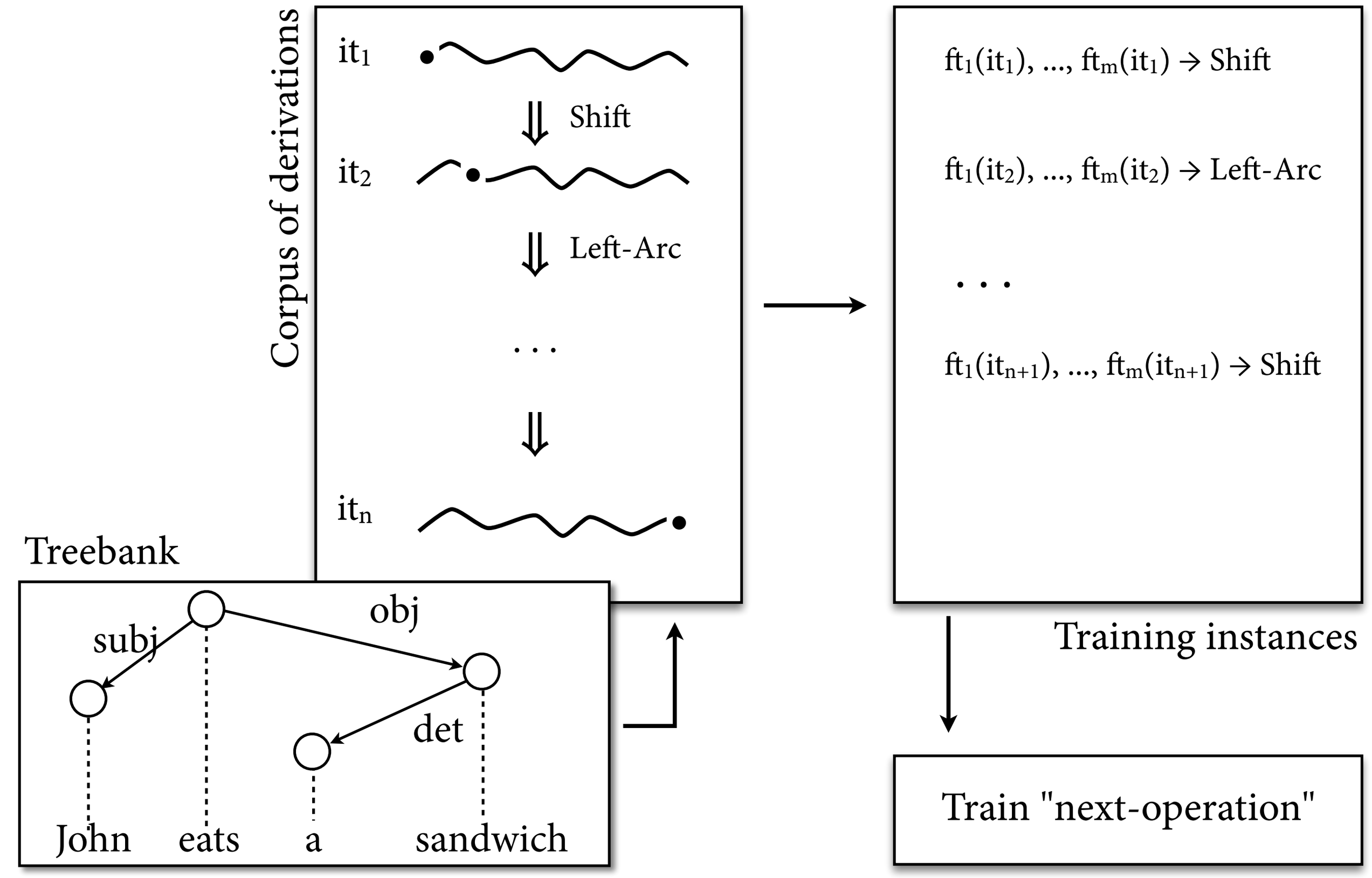
Parsing as Classification

- Can now do deterministic parsing as follows:

```
c = start-item
while (c not goal-item and can apply
        at least one parsing operation to c):
    op = next-operation(c)
    c = perform-operation(c, op)
```

- "next-operation" chooses parsing operation to be applied to c. How do we get it?

Training the classifier



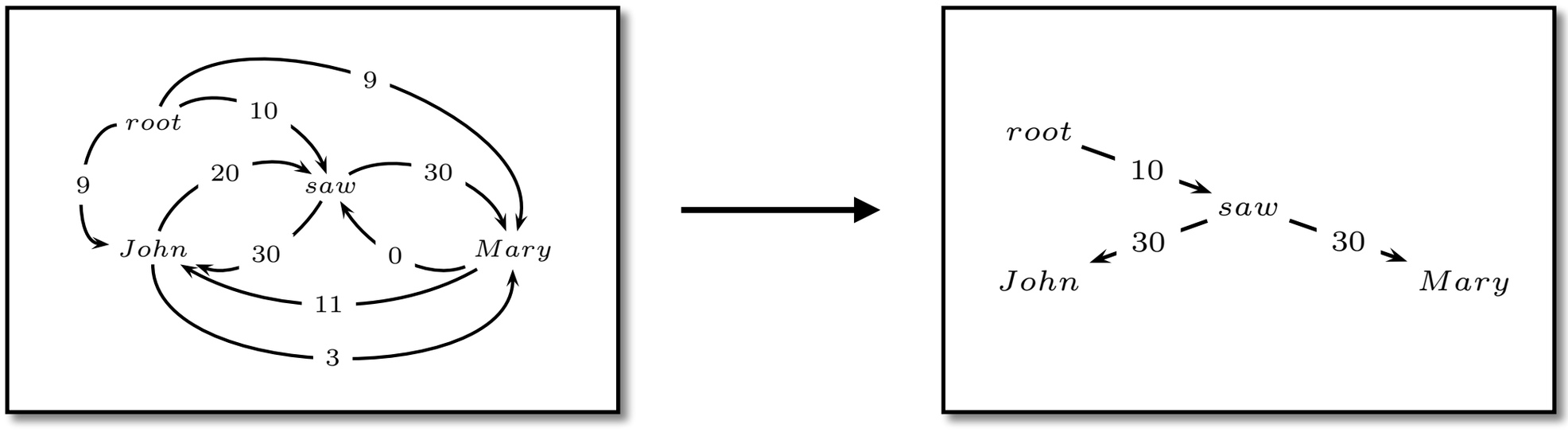
Features in MaltParser

- MaltParser (= standard implementation of Nivre algorithm) offers "toolbox" for features:
  - σ_i: i-th stack token (from the top)
  - τ_i: i-th token in remaining input
  - h(x): parent of x in the tree
  - l(x), r(x): leftmost (rightmost) child of x in the tree
  - p(x): POS tag of x
  - d(x): edge label from h(x) into x
  - build arbitrary terms from these, e.g. $p(l(\sigma_0))$

- Instead of engineering the features, can also use neural network classifier → e.g. Google SyntaxNet.

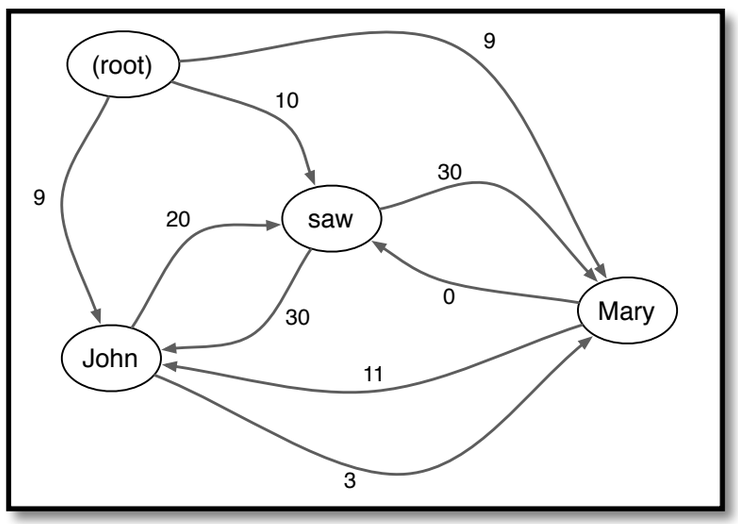
The MST Parser

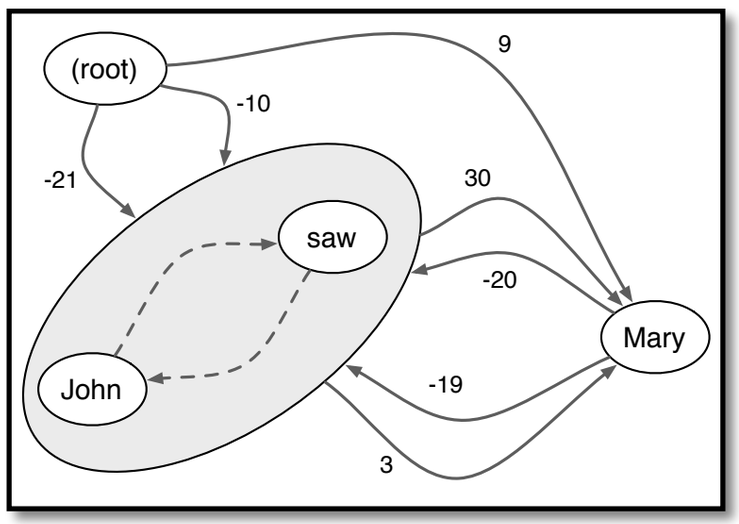
- Alternative idea (McDonald & Pereira, ca 2005):
  - take graph where nodes are words of sentence,
    and a directed edge between each two nodes
  - weight of edge represents how plausible a statistical model finds this edge
  - then calculate *maximum spanning tree*, i.e. tree that contains all nodes and has maximum sum of edge weights.



Computing MSTs

Using the Chu-Liu-Edmonds algorithm, runtime $O(n^2)$



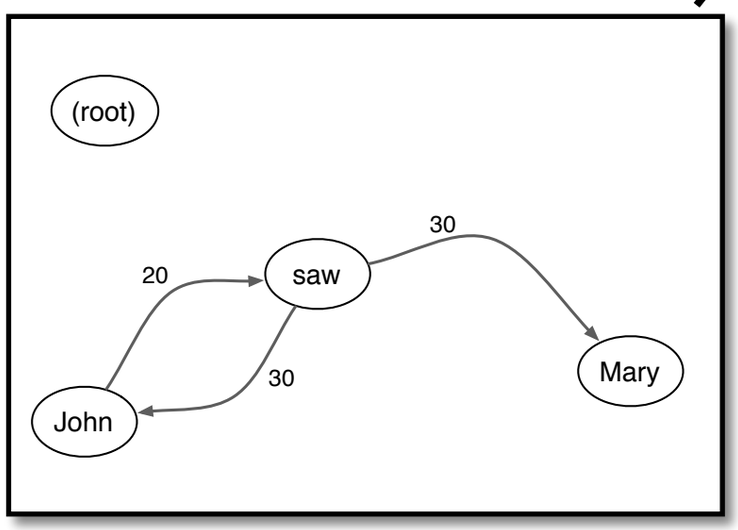


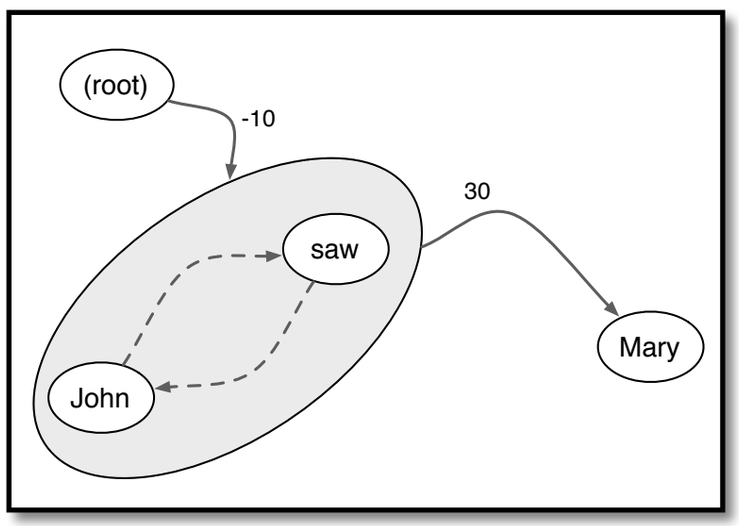
weight of new edge =
weight of old edge (u,v)
- weight of best edge into v

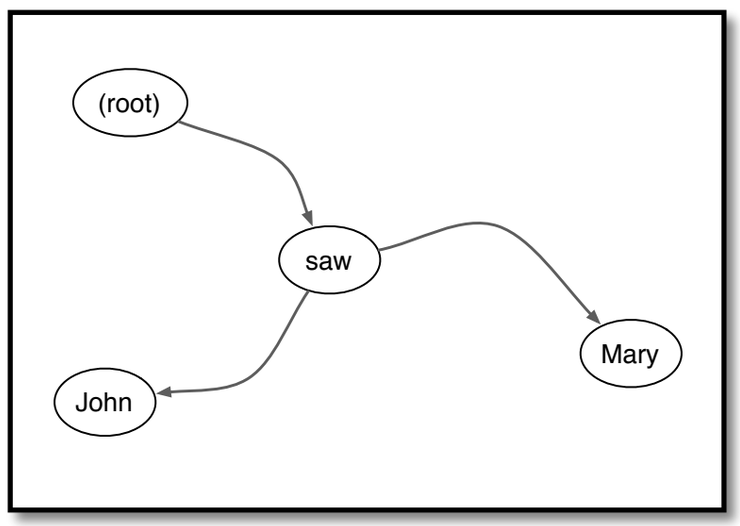
pick best incoming edges

contract cycles

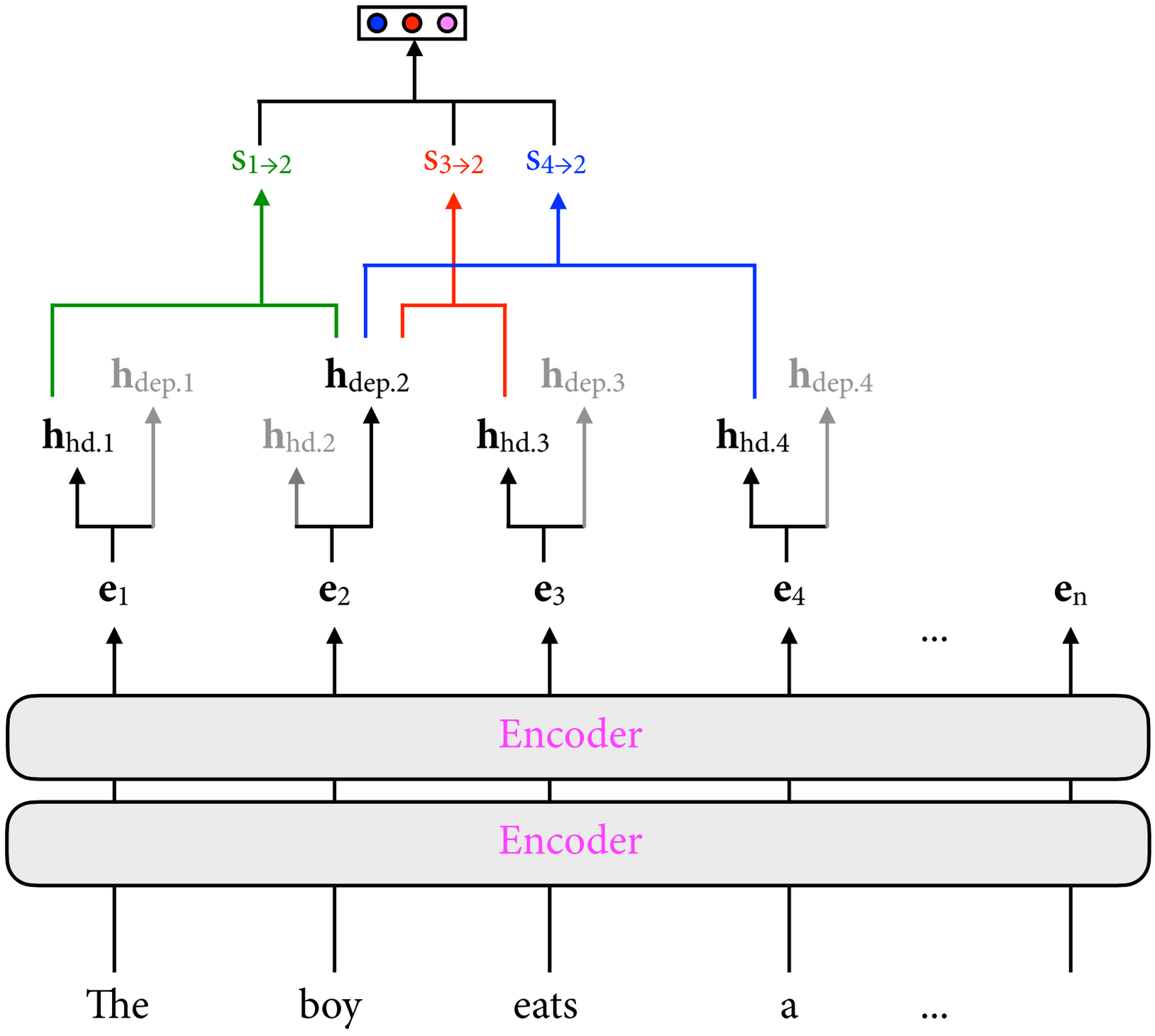
pick best incoming edges

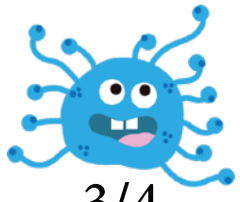






Neural graph-based parsing





3/4

(Kiperwasser & Goldberg 16; Dozat & Manning 17)

Evaluation

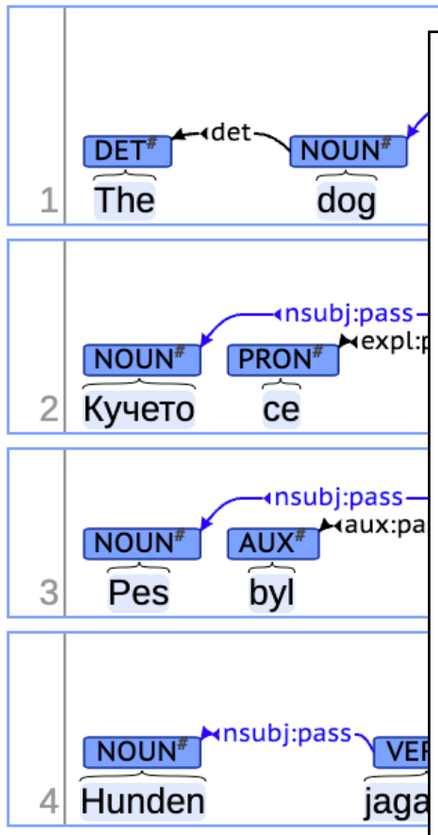
- Which proportion of edges predicted correctly?
  - *label accuracy:*
    #(nodes with correct label of incoming edge) / #nodes
  - *unlabeled attachment score:*
    #(nodes with correct parent) / #nodes
  - *labeled attachment score (LAS):*
    #(nodes with correct parent and edge label) / #nodes

Nivre vs McDonald

	McDonald	Nivre
Arabic	66.91	66.71
Bulgarian	87.57	87.41
Chinese	85.90	86.92
Czech	80.18	78.42
Danish	84.79	84.77
Dutch	79.19	78.59
German	87.34	85.82
Japanese	90.71	91.65
Portuguese	86.82	87.60
Slovene	73.44	70.30
Spanish	82.25	81.29
Swedish	82.55	84.58
Turkish	63.19	65.68
Overall	80.83	80.75

LAS in CoNLL-X Shared Task on Multilingual Dependency Parsing (2006)

Universal Dependencies



Current UD Languages

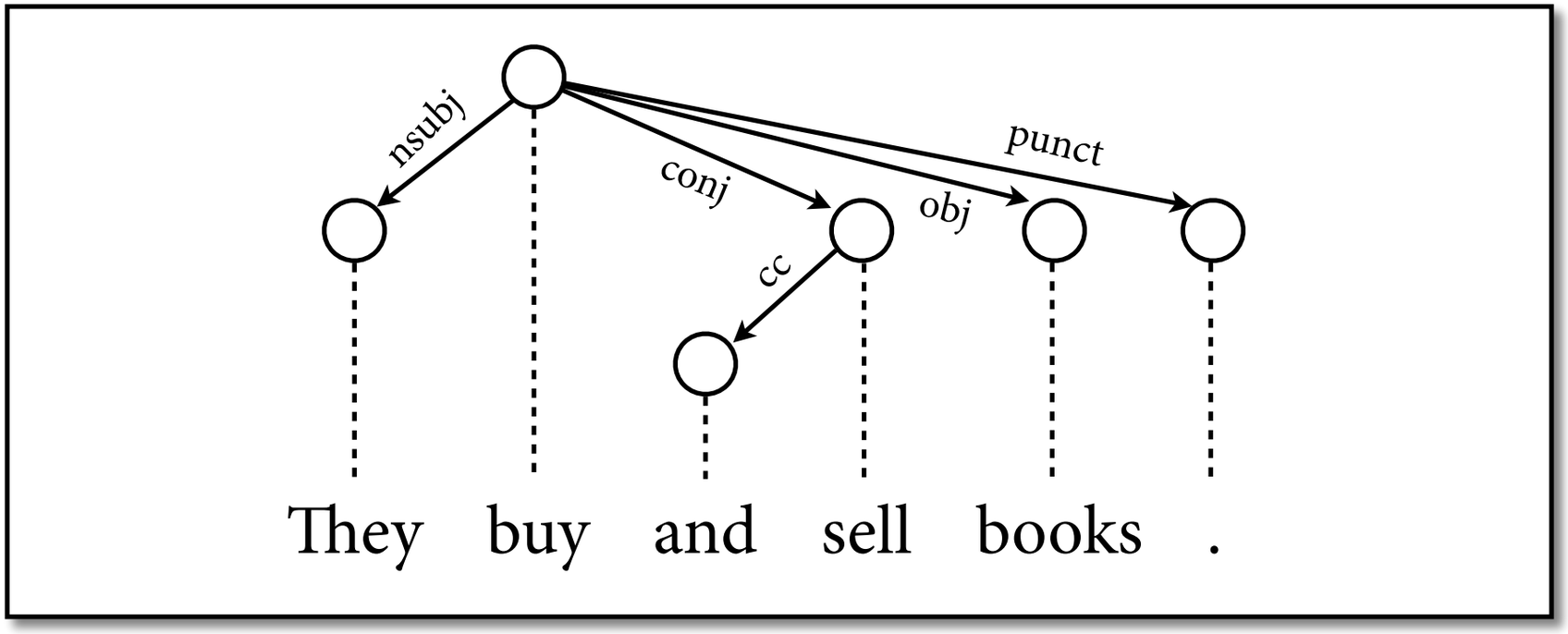
Information about language families (and genera for families with multiple branches) is mostly taken from WALS Online (IE = Indo-European).

Abaza	1	<1K	Northwest Caucasian
Afrikaans	1	49K	IE, Germanic
Akkadian	2	25K	Afro-Asiatic, Semitic
Akuntsu	1	1K	Tupian, Tupari
Albanian	1	<1K	IE, Albanian
Amharic	1	10K	Afro-Asiatic, Semitic
Ancient Greek	3	456K	IE, Greek
Ancient Hebrew	1	39K	Afro-Asiatic, Semitic
Apurina	1	<1K	Arawakan
Arabic	3	1,042K	Afro-Asiatic, Semitic
Armenian	2	94K	IE, Armenian
Assyrian	1	<1K	Afro-Asiatic, Semitic
Bambara	1	13K	Mande
Basque	1	121K	Basque
Beja	1	1K	Afro-Asiatic, Cushitic
Belarusian	1	305K	IE, Slavic
Bengali	1	<1K	IE, Indic
Bhojpuri	1	6K	IE, Indic
Bororo	1	1K	Bororoan
Breton	1	10K	IE, Celtic
Bulgarian	1	156K	IE, Slavic
Buryat	1	10K	Mongolic
Cantonese	1	13K	Sino-Tibetan
Catalan	1	553K	IE, Romance
Cebuano	1	1K	Austronesian, Central Philippine
Chinese	7	309K	Sino-Tibetan
Chukchi	1	6K	Chukotko-Kamchatkan
Classical Armenian	1	13K	IE, Armenian
Classical Chinese	1	433K	Sino-Tibetan
Coptic	1	57K	Afro-Asiatic, Egyptian
Croatian	1	199K	IE, Slavic
Czech	6	2,253K	IE, Slavic

(Nivre et al. 2016; https://universaldependencies.org; 259 treebanks for 148 languages as of 2023)

CoNLL-U

```
1    They      they     PRON    PRP    Case=Nom|Number=Plur                  2    nsubj
2    buy       buy      VERB    VBP    Number=Plur|Person=3|Tense=Pres       0    root
3    and       and      CCONJ   CC     _                                     4    cc
4    sell      sell     VERB    VBP    Number=Plur|Person=3|Tense=Pres       2    conj
5    books     book     NOUN    NNS    Number=Plur                           2    obj
6    .         .        PUNCT   .      _                                     2    punct
```



https://universaldependencies.org/format.html

Summary

- Dependency parsing: fundamentally different style of parsing algorithm than with PCFGs.

- Much newer parsing style, but now just as popular as phrase-structure parsing in current research.

- Very fast in practice (e.g. MaltParser is O(n)); Google SyntaxNet does ~600 words/sec.

- State of the art is LAS in the 90s - still active area of research.